– DOCTORAL DISSERTATION DEFENSE –

STATISTICAL PROGRAM DEPENDENCE APPROXIMATION

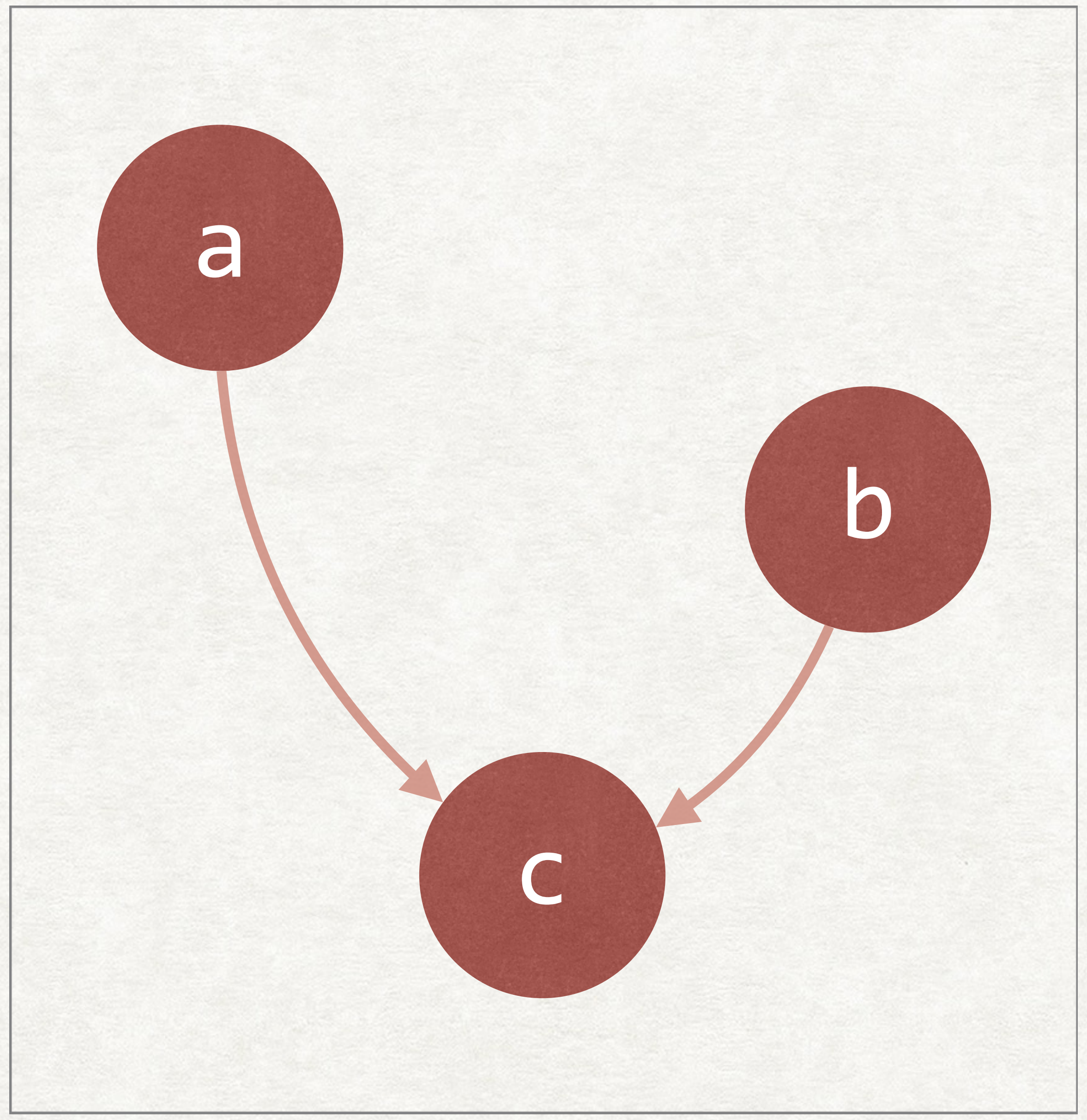
Presenter - SEONGMIN LEE
Advisor - SHIN YOO

2022.03.25

PROGRAM DEPENDENCY ANALYSIS

```
a = 3;
if (b > 0) {
   c = a + 42;
}
```

Program

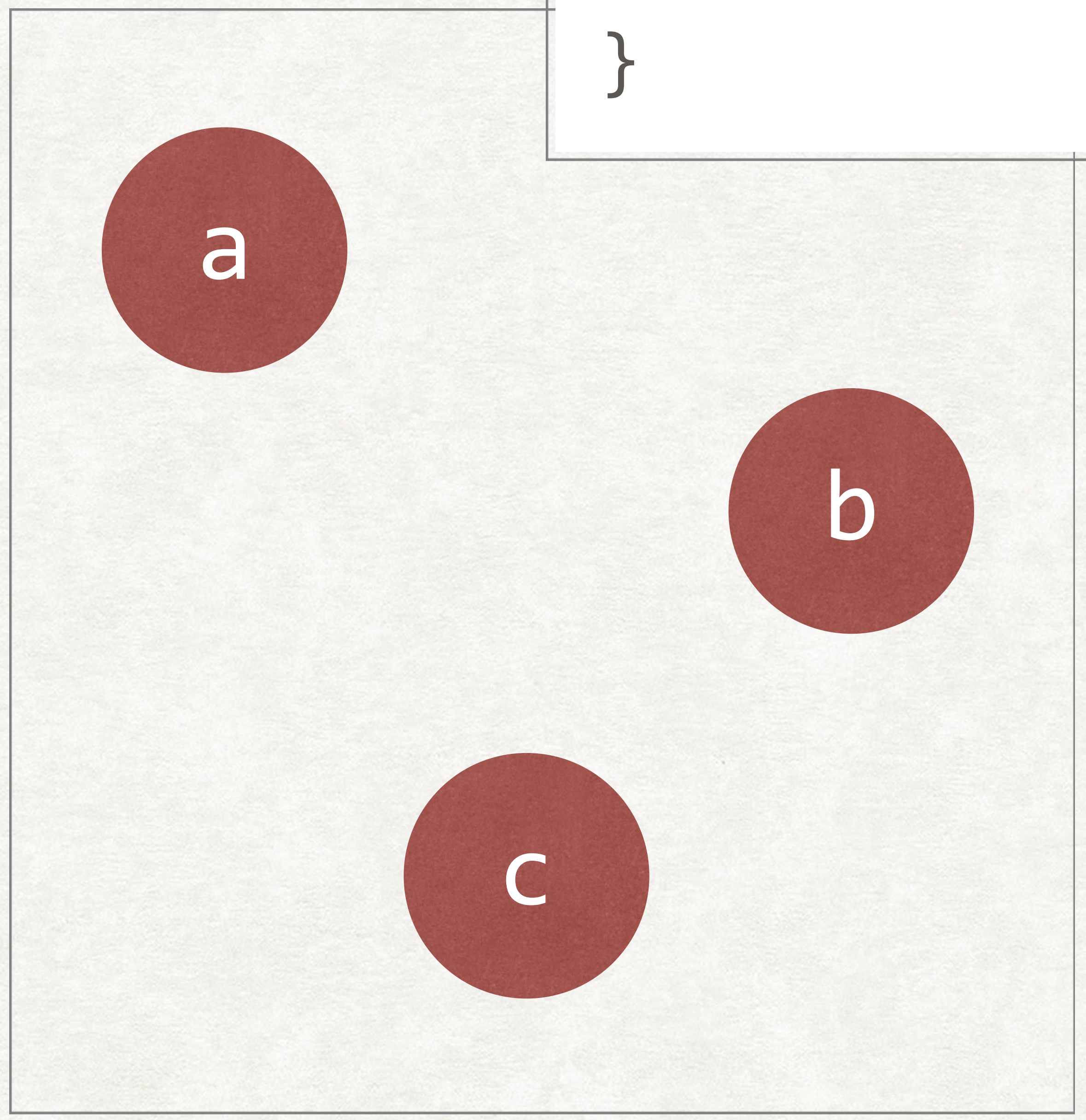


Program Dependency

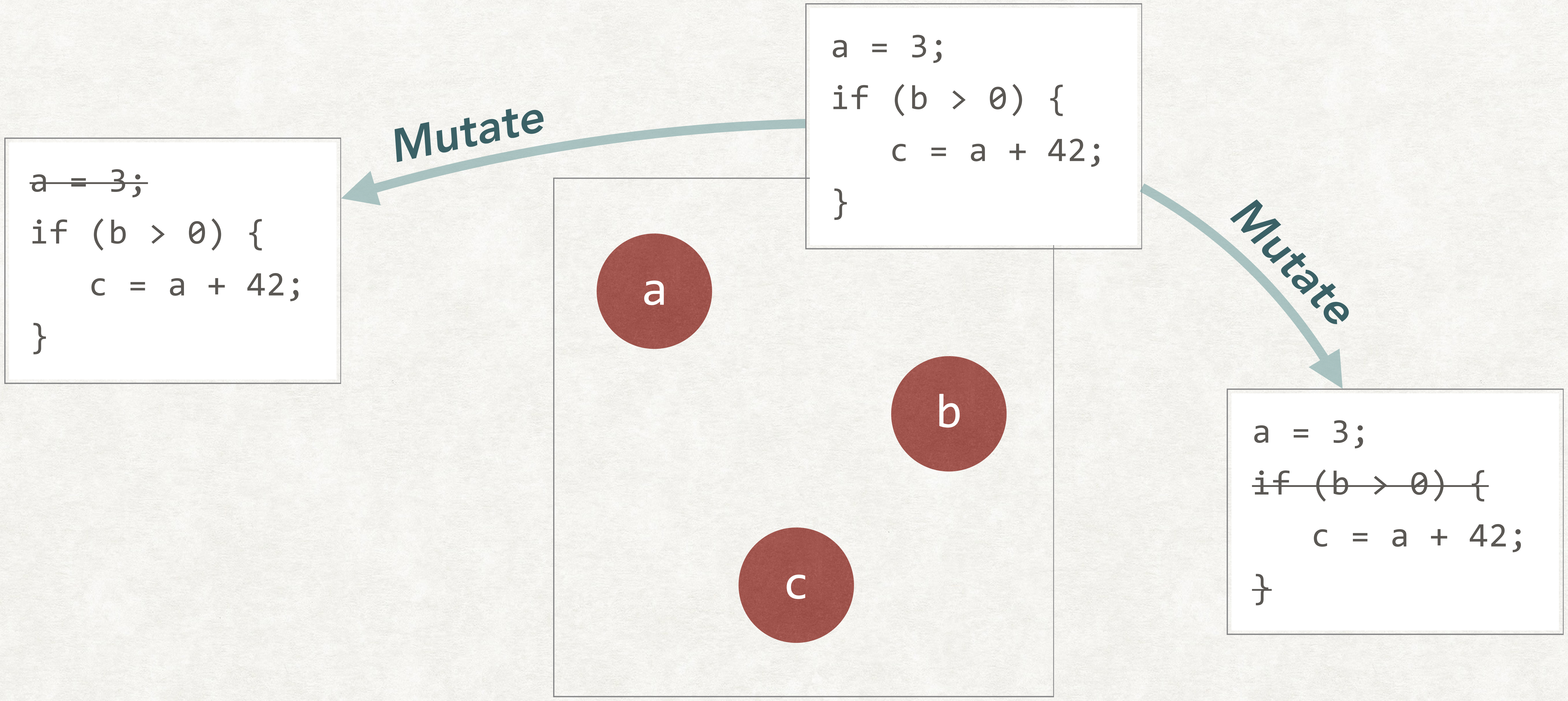
- **Program comprehension**
- **Software maintenance and evolution**
  - Which part of the program should be considered for the task
  - Shrink the search space of the source code
  - e.g. fault localization, refactoring, code reuse, etc.

OBSERVATION BASED ANALYSIS

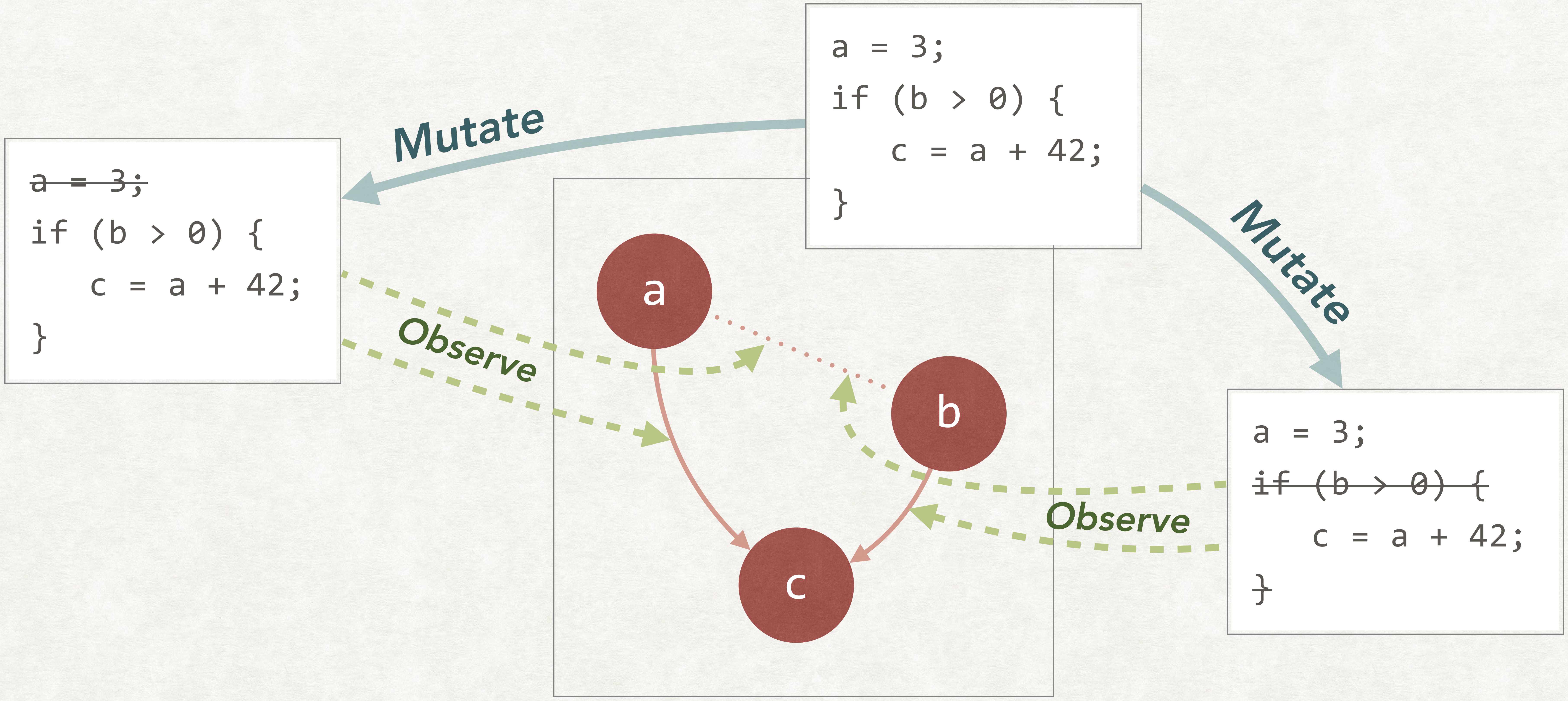
```
a = 3;
if (b > 0) {
   c = a + 42;
}
```



OBSERVATION BASED ANALYSIS



OBSERVATION BASED ANALYSIS



OBSERVATION BASED ANALYSIS

```
s = “Linux”
with open(“/tmp/arch.txt”, “w”) as f:
   f.write(s)
...
with open(“/tmp/arch.txt”) as f:
   arch = f.read()
```

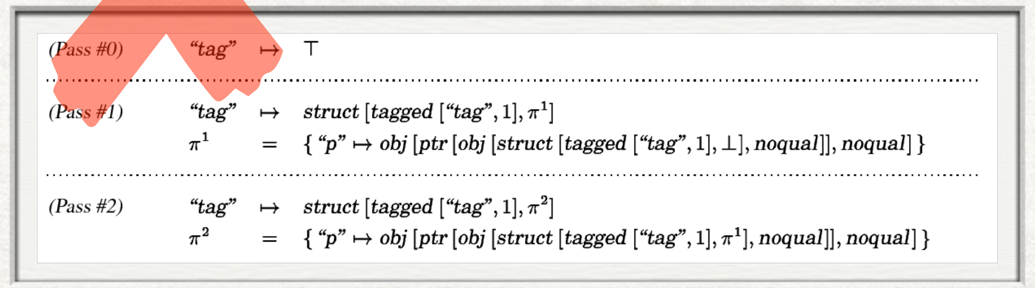
Linux

OBSERVATION BASED ANALYSIS

```
s = “Linux”
with open(“/tmp/arch.txt”, “w”) as f:
   f.write(s)
...
with open(“/tmp/arch.txt”) as f:
   arch = f.read()
```

Linux

Formal semantics



$$
\begin{array}{lccl}
\textit{(Pass \#0)} & \text{“tag”} & \mapsto & \top \\
\hline
\textit{(Pass \#1)} & \text{“tag”} & \mapsto & \textit{struct}\,[\textit{tagged}\,[\text{“tag”}, 1], \pi^1] \\
 & \pi^1 & = & \{\, \text{“p”} \mapsto \textit{obj}\,[\textit{ptr}\,[\textit{obj}\,[\textit{struct}\,[\textit{tagged}\,[\text{“tag”}, 1], \bot], \textit{noqual}]], \textit{noqual}] \,\} \\
\hline
\textit{(Pass \#2)} & \text{“tag”} & \mapsto & \textit{struct}\,[\textit{tagged}\,[\text{“tag”}, 1], \pi^2] \\
 & \pi^2 & = & \{\, \text{“p”} \mapsto \textit{obj}\,[\textit{ptr}\,[\textit{obj}\,[\textit{struct}\,[\textit{tagged}\,[\text{“tag”}, 1], \pi^1], \textit{noqual}]], \textit{noqual}] \,\}
\end{array}
$$

OBSERVATION BASED ANALYSIS

Mutate

```
s = “Linux”   →   s = “Window”
with open(“/tmp/arch.txt”, “w”) as f:
   f.write(s)
...
with open(“/tmp/arch.txt”) as f:
   arch = f.read()
```

Linux → Window : **dynamically observable**

Formal semantics

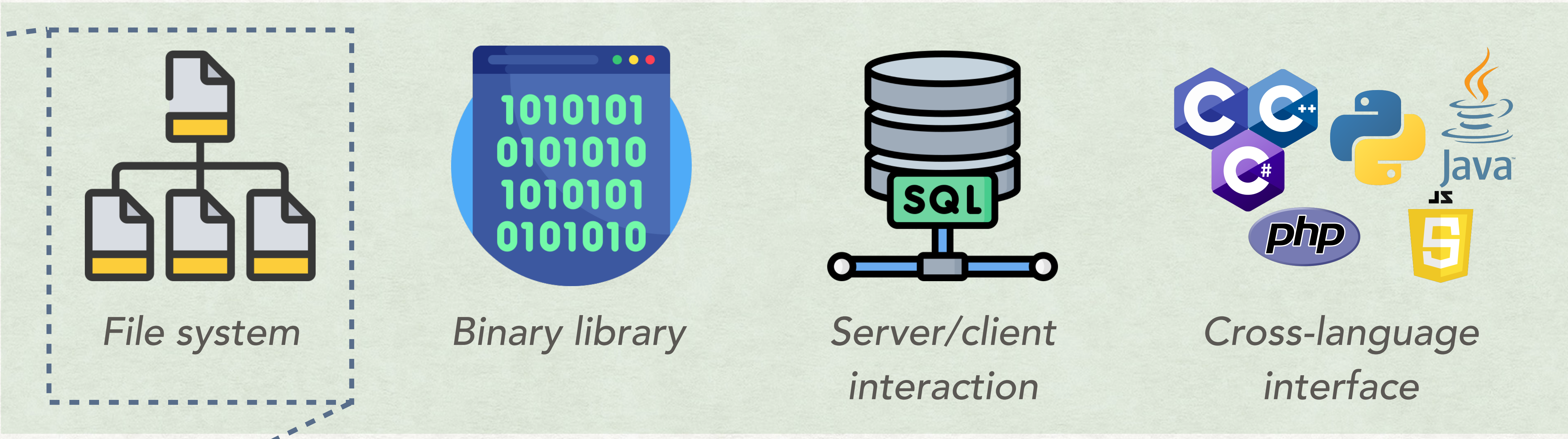
$$
\begin{array}{lrcl}
(\textit{Pass \#0}) & \text{“tag”} & \mapsto & \top \\
(\textit{Pass \#1}) & \text{“tag”} & \mapsto & \textit{struct}\,[\textit{tagged}\,[\text{“tag”}, 1], \pi^1] \\
 & \pi^1 & = & \{\, \text{“p”} \mapsto \textit{obj}\,[\textit{ptr}\,[\textit{obj}\,[\textit{struct}\,[\textit{tagged}\,[\text{“tag”}, 1], \bot], \textit{noqual}]], \textit{noqual}] \,\} \\
(\textit{Pass \#2}) & \text{“tag”} & \mapsto & \textit{struct}\,[\textit{tagged}\,[\text{“tag”}, 1], \pi^2] \\
 & \pi^2 & = & \{\, \text{“p”} \mapsto \textit{obj}\,[\textit{ptr}\,[\textit{obj}\,[\textit{struct}\,[\textit{tagged}\,[\text{“tag”}, 1], \pi^1], \textit{noqual}]], \textit{noqual}] \,\}
\end{array}
$$

OBSERVATION BASED ANALYSIS

Mutate

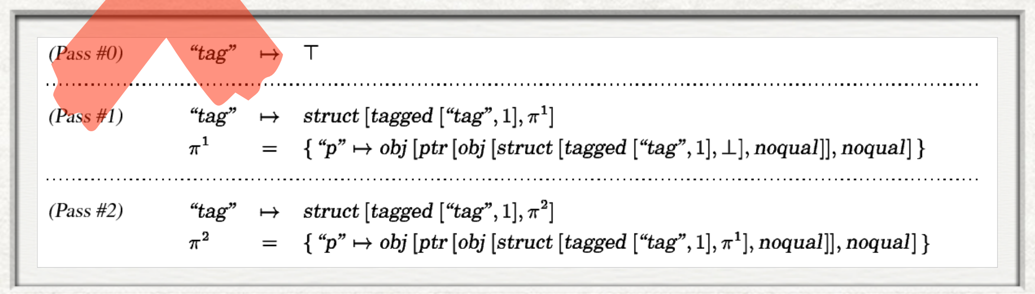
```
s = "Linux"  →  s = "Window"
with open("/tmp/arch.txt", "w") as f:
   f.write(s)
...
with open("/tmp/arch.txt") as f:
   arch = f.read()
```

Linux → Window : **dynamically observable**



Features beyond formal semantics

Formal semantics

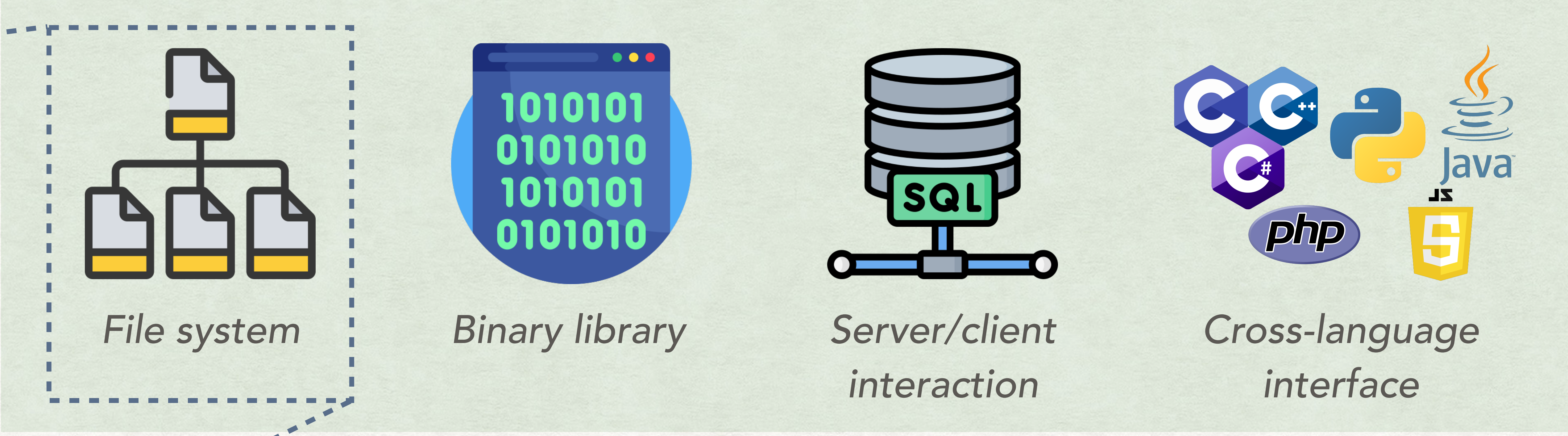


OBSERVATION BASED ANALYSIS

Mutate

```
s = “Linux”  →  s = “Window”
with open(“/tmp/arch.txt”, “w”) as f:
    f.write(s)
...
with open(“/tmp/arch.txt”) as f:
    arch = f.read()
```

Linux → Window : **dynamically observable**



Features beyond formal semantics

Formal semantics

(Pass #0)	“tag”	$\mapsto$	$\top$
(Pass #1)	“tag”	$\mapsto$	*struct* [*tagged* [“tag”, 1], π^1]
	π^1	$=$	{ “p” $\mapsto$ *obj* [*ptr* [*obj* [*struct* [*tagged* [“tag”, 1], $\perp$], *noqual*]], *noqual*] }
(Pass #2)	“tag”	$\mapsto$	*struct* [*tagged* [“tag”, 1], π^2]
	π^2	$=$	{ “p” $\mapsto$ *obj* [*ptr* [*obj* [*struct* [*tagged* [“tag”, 1], π^1], *noqual*]], *noqual*] }

checker.java:

```java
class checker {
  public static void main(String[] args) {
    int dots = 0;
    for (int i = 0; i < args[0].length(); ++i) {
      if (args[0].charAt(i) == '.') {
        ++dots;
      }
    }
  }
```

reader.c:

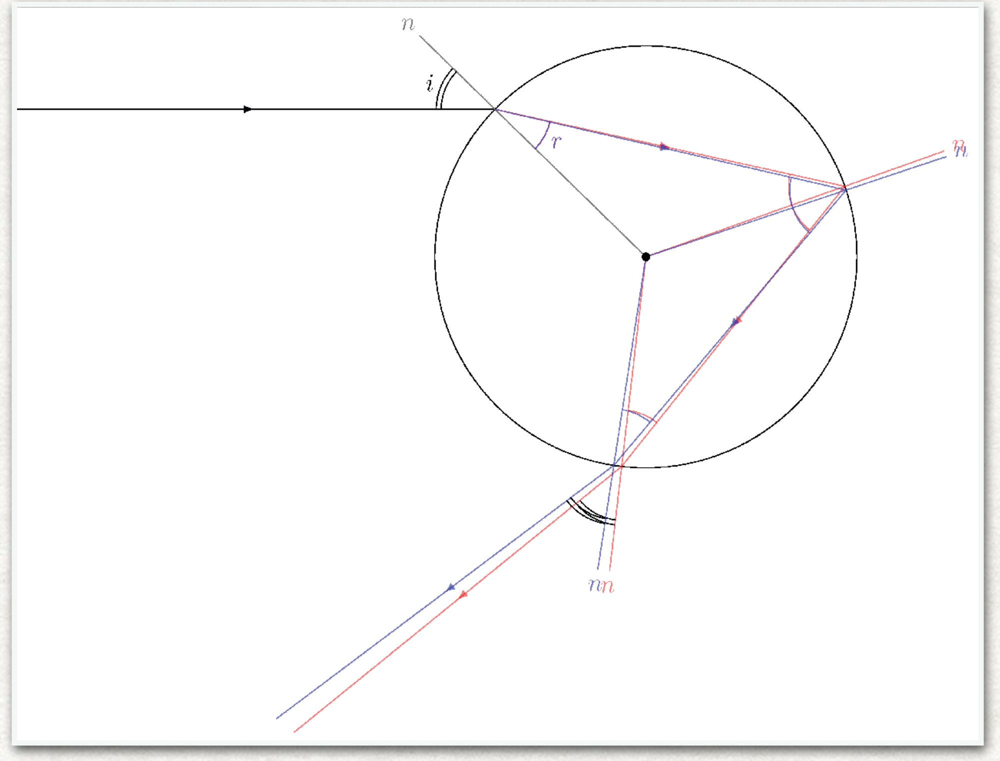
```c
#include <locale.h>
int main(int argc, char **argv) {
  struct lconv *cur_locale = localeconv();
  {
    printf("%s\n", cur_locale->decimal_point);
  }
}
```

glue.py:

```python
import commands
import sys
use_locale = True
currency = "?"
if use_locale:
  decimal = commands.getoutput('./reader 1')
```

Multi-lingual Program

Picture Description Language



Programs with non-conventional semantics

OBSERVATION-BASED SLICING (ORBS)

Original program

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf("%d\n", sum);
    printf("%d\n", i);
}
```

OBSERVATION-BASED SLICING (ORBS)

Original program

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf("%d\n", sum);
    printf("%d\n", i);
}
```

Slicing criterion

OBSERVATION-BASED SLICING (ORBS)

Original program

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf("%d\n", sum);
    printf("%d\n", i);
}
```

Slicing criterion

OBSERVATION-BASED SLICING (ORBS)

Original program

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf("%d\n", sum);
    printf("%d\n", i);
}
```

Slicing criterion

PROGRAM SLICING

Program slice

```
int main() {

    int i = 1;
    while (i < 11) {

        i = i + 1;
    }

    printf("%d\n", i);
}
```

OBSERVATION-BASED SLICING (ORBS)

Original program

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf("%d\n", sum);
    printf("%d\n", i);
}
```

Slicing criterion

PROGRAM SLICING

Program slice

```
int main() {

    int i = 1;
    while (i < 11) {

        i = i + 1;
    }

    printf("%d\n", i);
}
```

Run

```
$ 11
```

Trajectory

OBSERVATION-BASED SLICING (ORBS)

Original program

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf(“%d\n”, sum);
    printf(“%d\n”, i);
}
```

Slicing criterion

PROGRAM SLICING

Program slice

```
int main() {

    int i = 1;
    while (i < 11) {

        i = i + 1;
    }

    printf(“%d\n”, i);
}
```

Run

$ 11

Run

Trajectory

OBSERVATION-BASED SLICING (ORBS)

Program

Slicing criterion

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf("%d\n", sum);
    printf("%d\n", i);
}
```

OBSERVATION-BASED SLICING (ORBS)

Program

Slicing criterion

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf(“%d\n”, sum);
    printf(“%d\n”, i);
}
```

Run

$ 11

Oracle

OBSERVATION-BASED SLICING (ORBS)

OBSERVE IMPACT ON OTHERS

MUTATE PROGRAM

Program

Slicing criterion

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf("%d\n", sum);
    printf("%d\n", i);
}
```

curr. line →

Run

$ 11

Oracle

OBSERVATION-BASED SLICING (ORBS)

Program

Slicing criterion

MUTATE PROGRAM

OBSERVE IMPACT ON OTHERS

curr. line →

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf("%d\n", sum);
    printf("%d\n", i);
}
```

Run

$ Not compile

The line/lines may be needed for the slice.

OBSERVATION-BASED SLICING (ORBS)

Program

Slicing criterion

MUTATE PROGRAM

WINDOW DELETION

OBSERVE
IMPACT ON
OTHERS

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf("%d\n", sum);
    printf("%d\n", i);
}
```

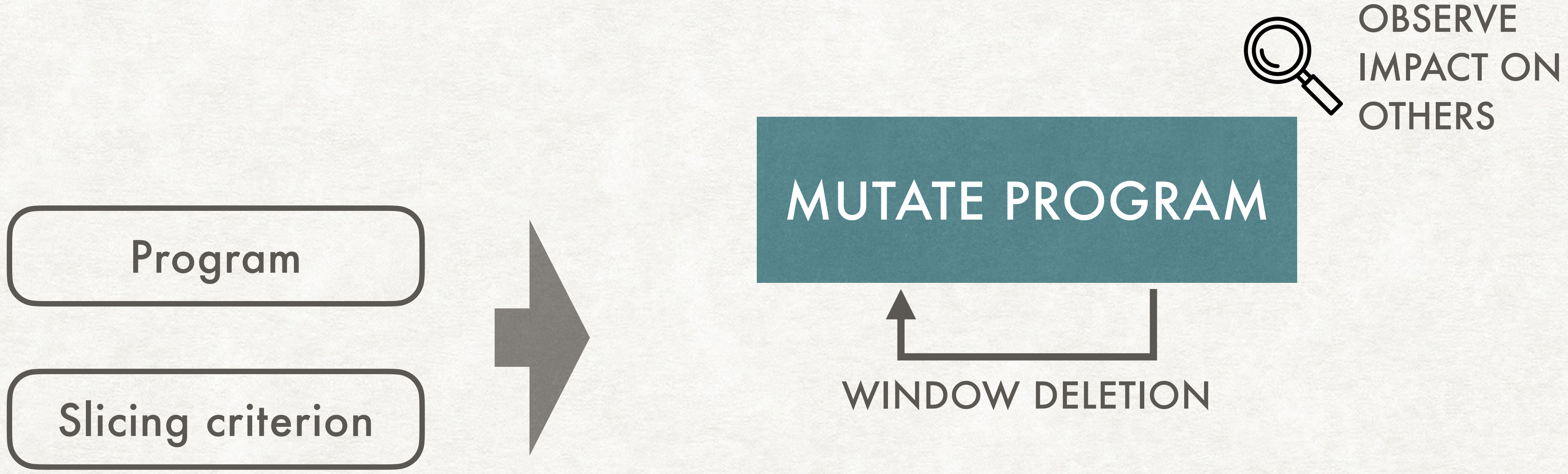
curr. line →

Run

```
$ Not compile
```

The line/lines may be needed for the slice.

OBSERVATION-BASED SLICING (ORBS)




```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf("%d\n", sum);
    printf("%d\n", i);
}
```

curr. line

The line/lines may be needed for the slice.

OBSERVATION-BASED SLICING (ORBS)

Program

Slicing criterion

MUTATE PROGRAM

WINDOW DELETION

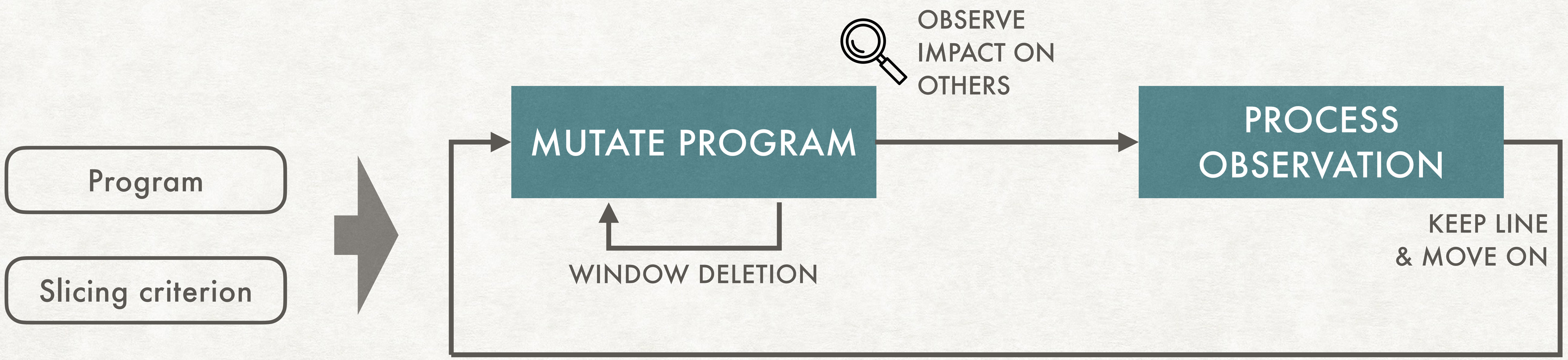
OBSERVE IMPACT ON OTHERS

curr. line →

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf("%d\n", sum);
    printf("%d\n", i);
}
```

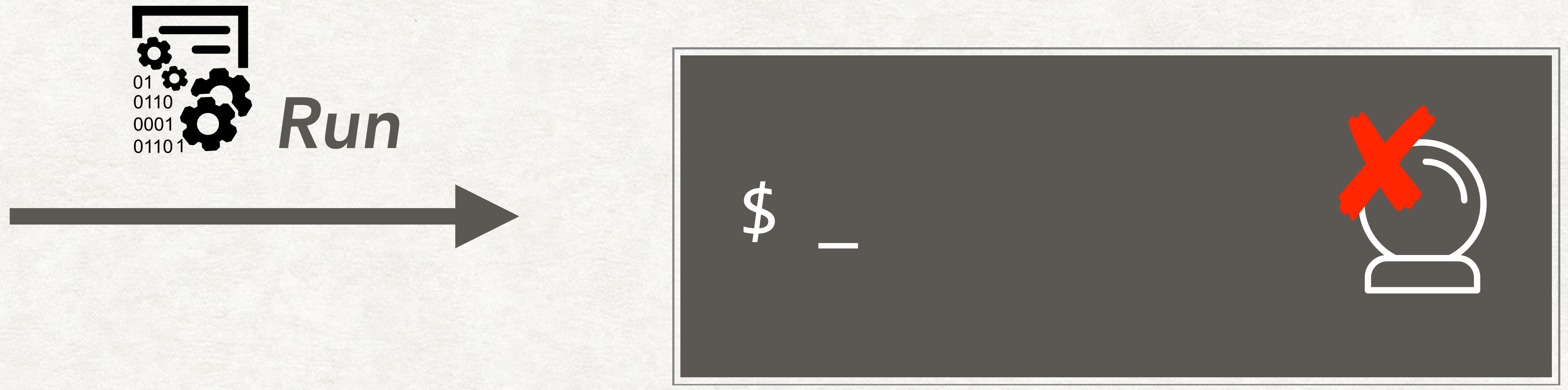
Run

OBSERVATION-BASED SLICING (ORBS)



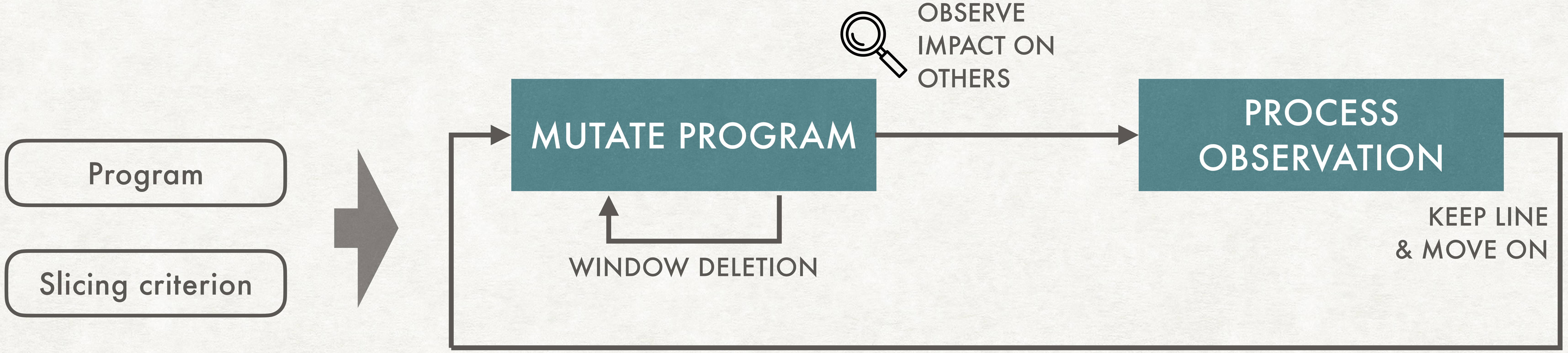

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf("%d\n", sum);
    printf("%d\n", i);
}
```

curr. line →



The line/lines may be needed for the slice.

OBSERVATION-BASED SLICING (ORBS)

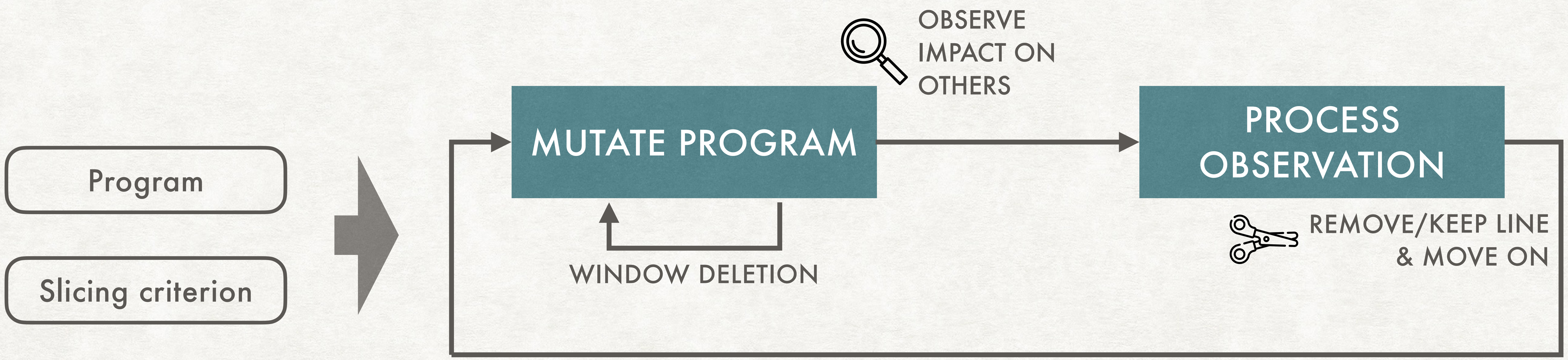


curr. line →

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf("%d\n", sum);
    printf("%d\n", i);
}
```

The line/lines do not need to be in the slice.

OBSERVATION-BASED SLICING (ORBS)

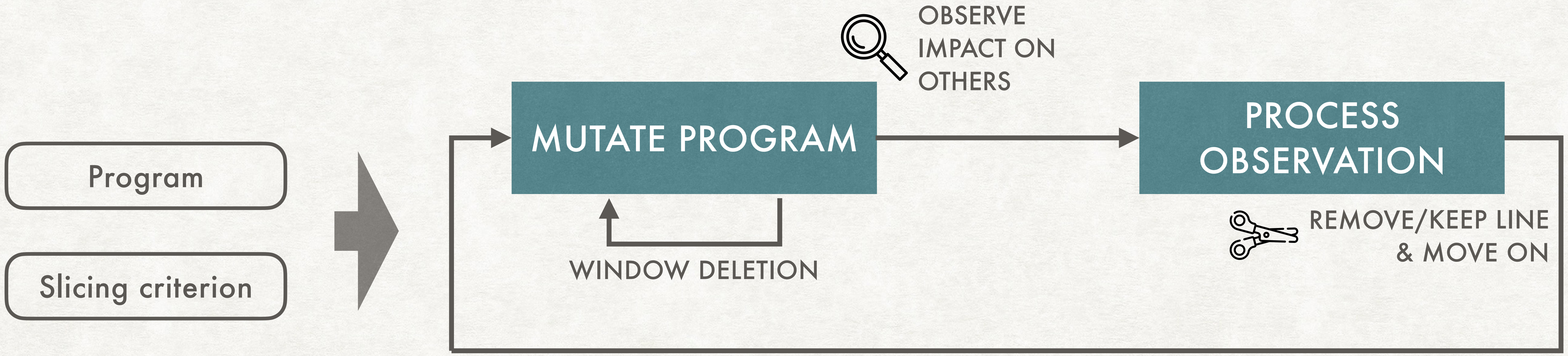


curr. line →

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }

    printf("%d\n", i);
}
```

The line/lines do not need to be in the slice.

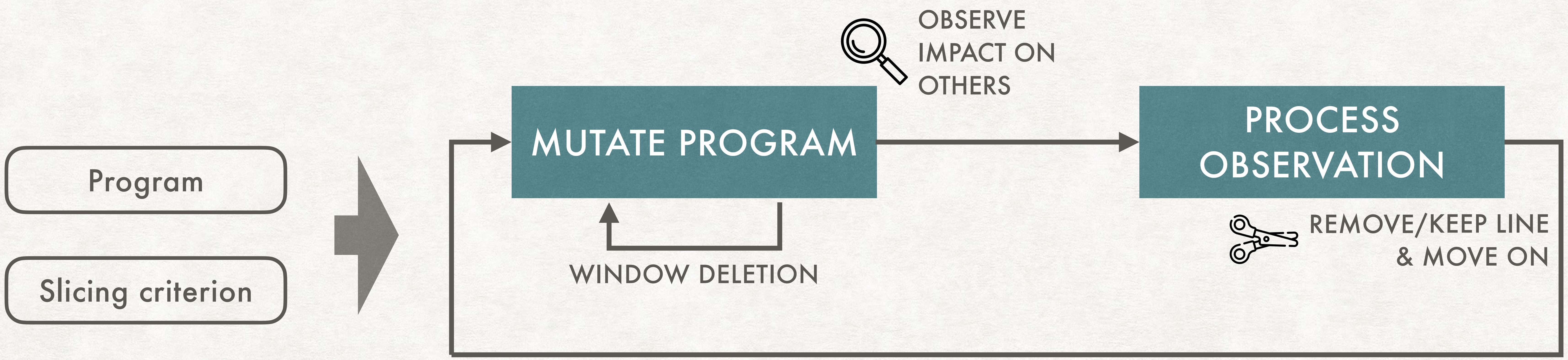
OBSERVATION-BASED SLICING (ORBS)




```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }

    printf("%d\n", i);
}
```

The line/lines do not need to be in the slice.

OBSERVATION-BASED SLICING (ORBS)




```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }

    printf("%d\n", i);
}
```

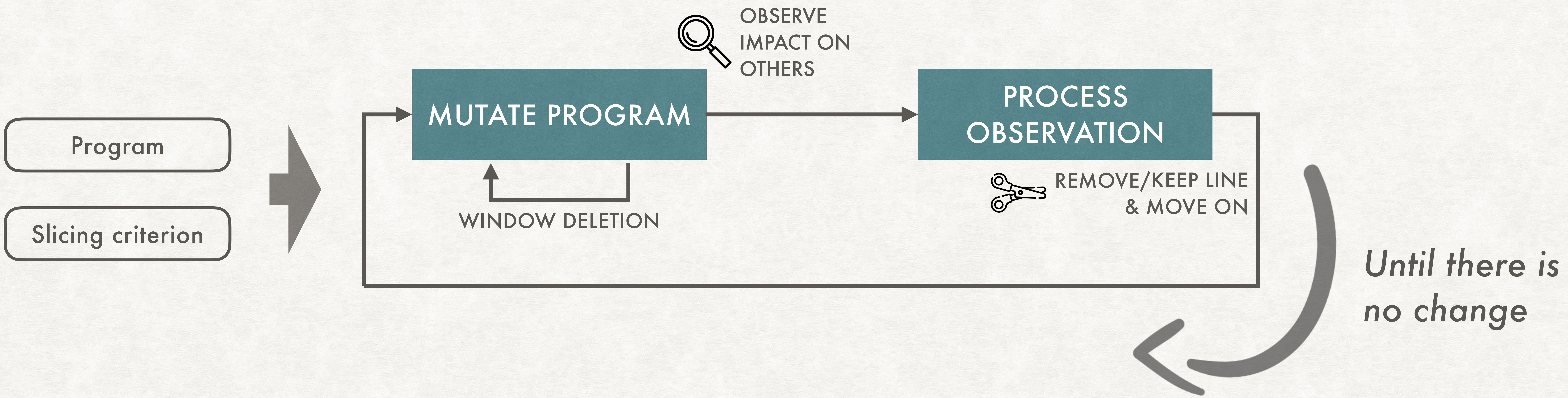
Multiple iterations

Run

```
$ 11
```

The line/lines do not need to be in the slice.

OBSERVATION-BASED SLICING (ORBS)

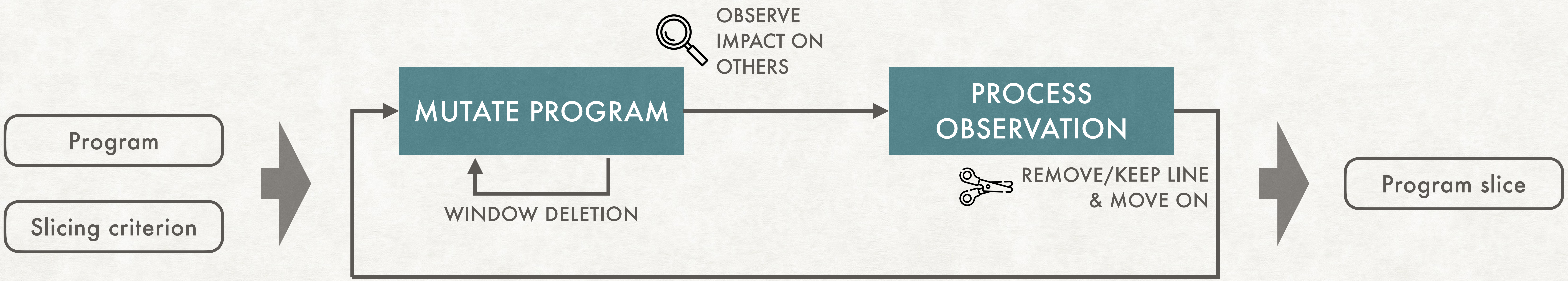



```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }

    printf("%d\n", i);
}
```

Multiple iterations

The line/lines do not need to be in the slice.

OBSERVATION-BASED SLICING (ORBS)

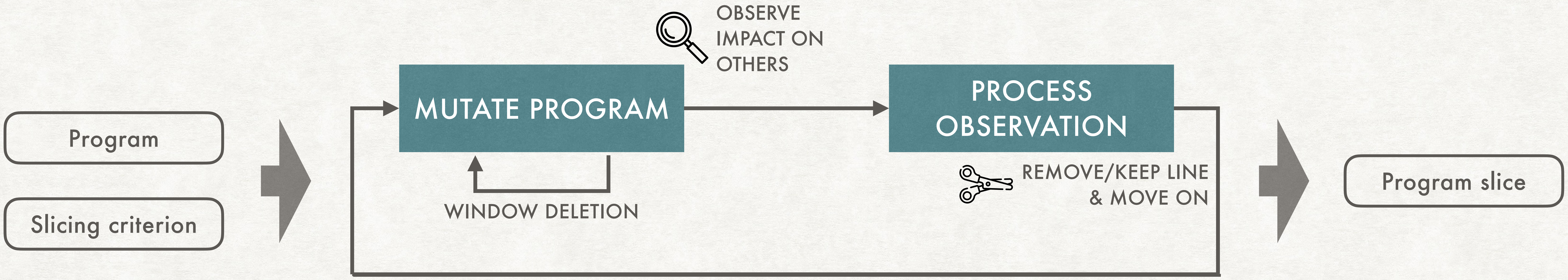



```c
int main() {

    int i = 1;
    while (i < 11) {

        i = i + 1;
    }

    printf("%d\n", i);
}
```

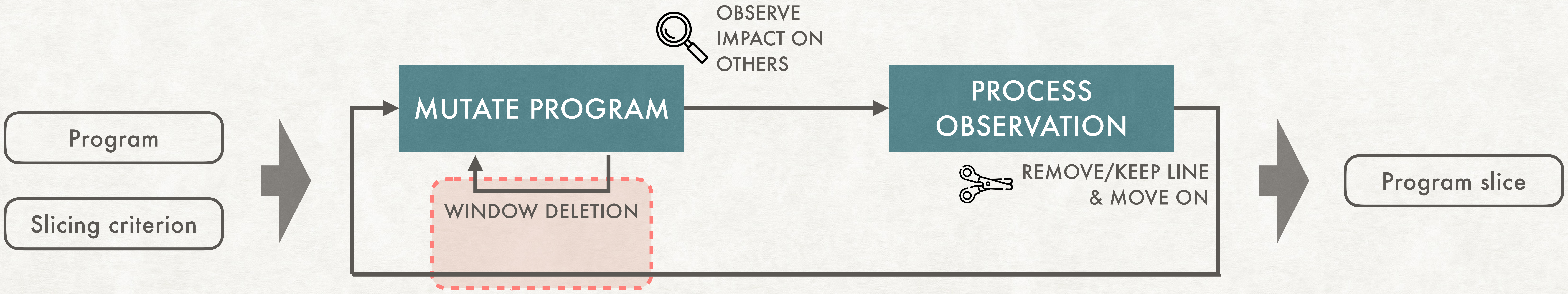
Multiple iterations

The line/lines do not need to be in the slice.

LIMITATIONS OF ORBS



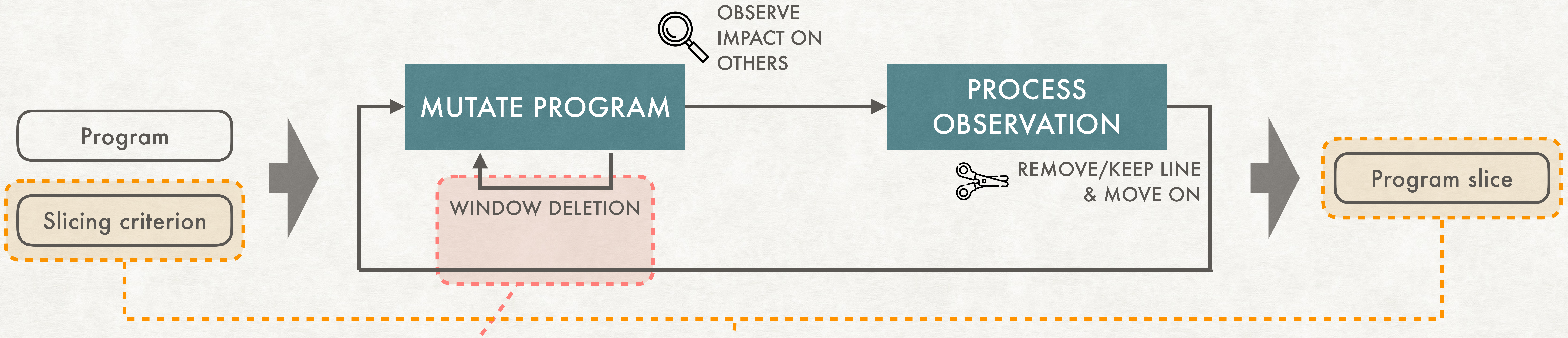
LIMITATIONS OF ORBS



SCALABILITY

- **Costly analysis**
  - Requires a large number of compilations & executions
  - Takes > 2 hours for a single iteration (=skimming every line once) for 1.5K NCLOC.
  - Be critical to the usability

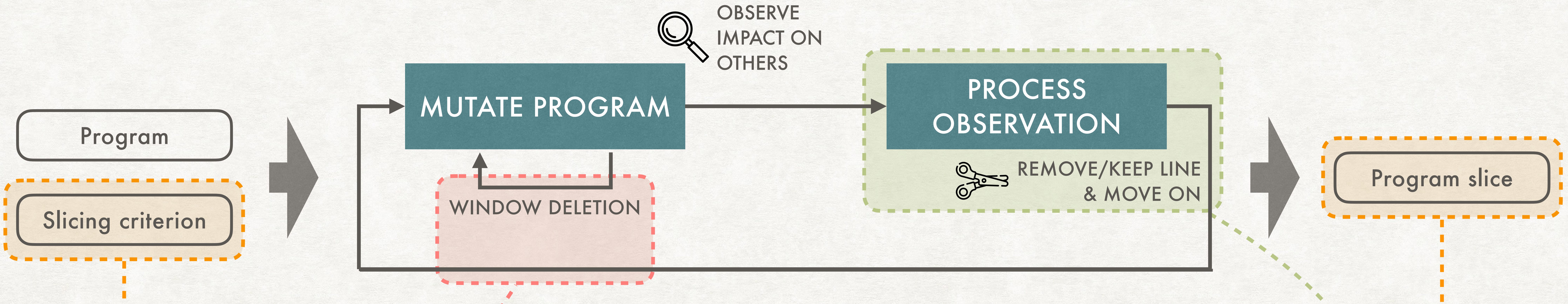
LIMITATIONS OF ORBS



SCALABILITY

- **Costly analysis**
  - Requires a large number of compilations & executions
  - Takes > 2 hours for a single iteration (=skimming every line once) for 1.5K NCLOC.
  - Be critical to the usability
- **Partial analysis**
  - Produce a single program slice
  - Does not provide dependence for other program elements
  - Requires running numerous times for complete dependence analysis

LIMITATIONS OF ORBS



SCALABILITY

- **Costly analysis**
  - Requires a large number of compilations & executions
  - Takes > 2 hours for a single iteration (=skimming every line once) for 1.5K NCLOC.
  - Be critical to the usability
- **Partial analysis**
  - Produce a single program slice
  - Does not provide dependence for other program elements
  - Requires running numerous times for complete dependence analysis

INTERPRETABILITY

- **No structural reasoning**
  - Cannot reason why one depends on another
- **Binary dependency**
  - Cannot explain how much one depends on another

PROBLEM STATEMENT

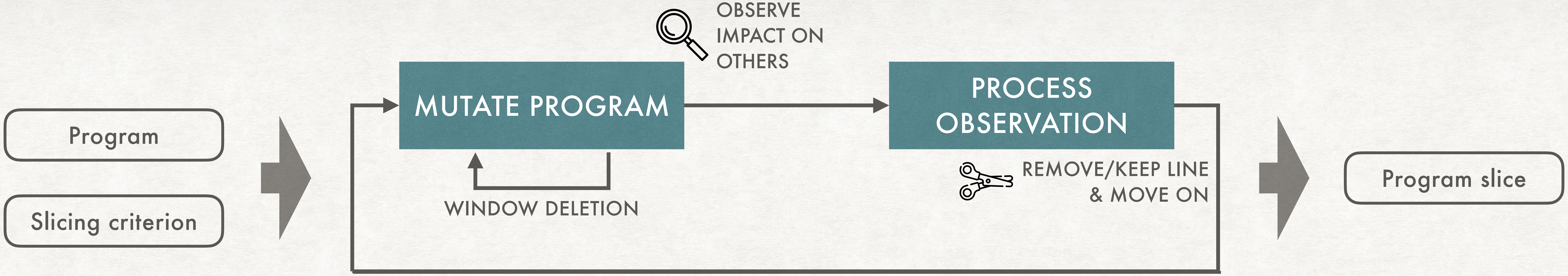
Although an **observation-based analysis** had been proposed
to overcome the limitation of formal semantics-based dependency analysis,
existing observation-based analysis lacks *scalability* and *interpretability*.

THESIS

Although an **observation-based analysis** had been proposed to overcome the limitation of formal semantics-based dependency analysis, existing observation-based analysis lacks *scalability* and *interpretability*.

↓

Statistically modeling program dependence can improve the scalability and the interpretability of the observation-based dependence analysis.



SCALABILITY

- Costly observation
- Partial analysis

COMPREHENSION

- No structural reasoning
- Binary dependency

Program

Slicing criterion

OBSERVE IMPACT ON OTHERS

MUTATE PROGRAM

WINDOW DELETION

PROCESS OBSERVATION

REMOVE/KEEP LINE & MOVE ON

Program slice

SCALABILITY

- Costly observation
- Partial analysis

COMPREHENSION

- No structural reasoning
- Binary dependency

1

MOBS

– LEXICAL MODEL

- Approximate the dependence from the lexical feature
- Efficiently observe the mutation

Program

Slicing criterion

MUTATE PROGRAM

WINDOW DELETION

OBSERVE IMPACT ON OTHERS

PROCESS OBSERVATION

REMOVE/KEEP LINE & MOVE ON

Program slice

SCALABILITY

- Costly observation
- Partial analysis

COMPREHENSION

- No structural reasoning
- Binary dependency

1

MOBS

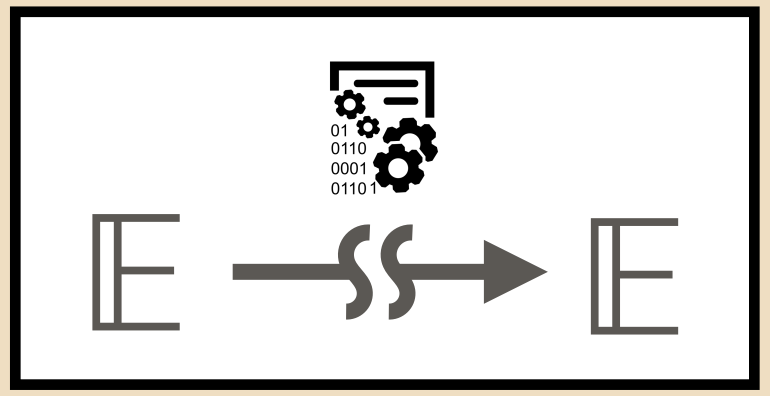
– LEXICAL MODEL

- Approximate the dependence from the lexical feature
- Efficiently observe the mutation

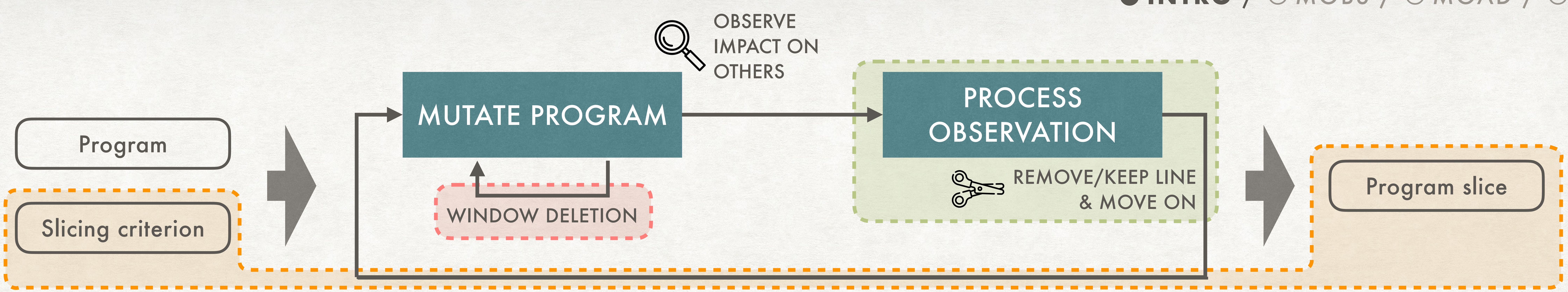
2

MOAD

– STATISTICAL MODEL



- Expand from the slicing method to the general dependence approximation method



SCALABILITY

- Costly observation
- Partial analysis

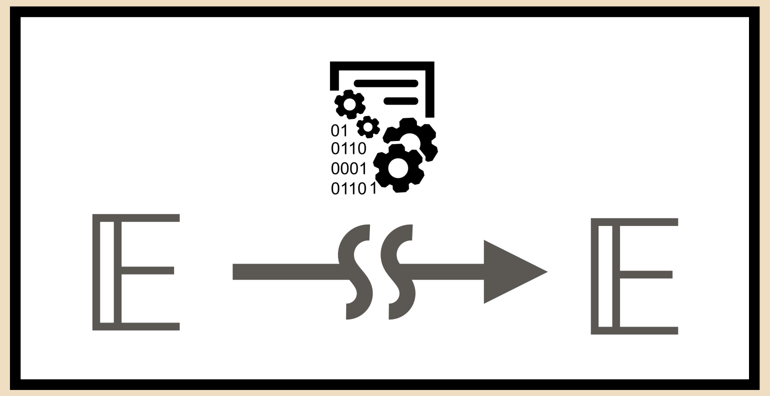
COMPREHENSION

- No structural reasoning
- Binary dependency

1 MOBS – LEXICAL MODEL

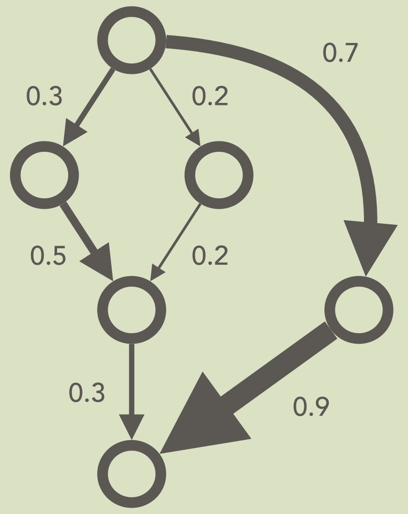
- Approximate the dependence from the lexical feature
- Efficiently observe the mutation

2 MOAD – STATISTICAL MODEL



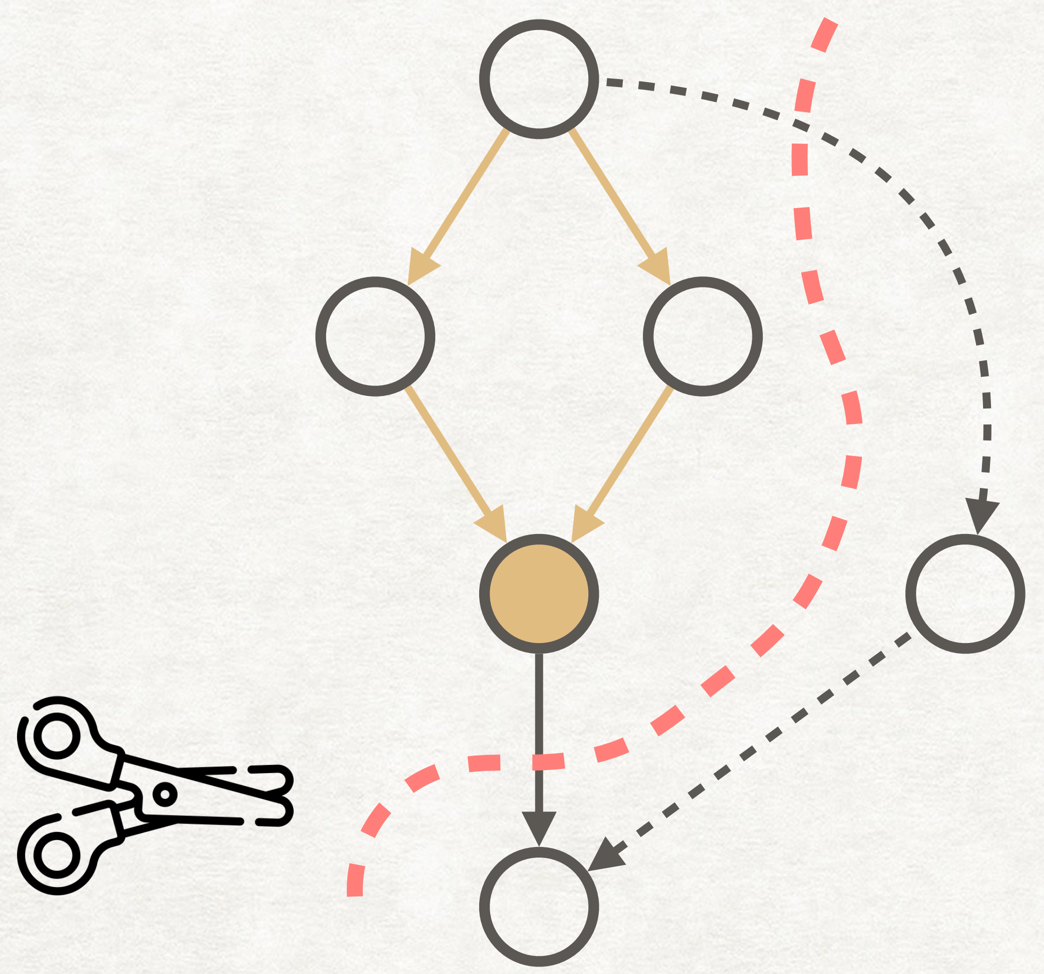
- Expand from the slicing method to the general dependence approximation method

3 CPDA – CAUSAL INFERENCE



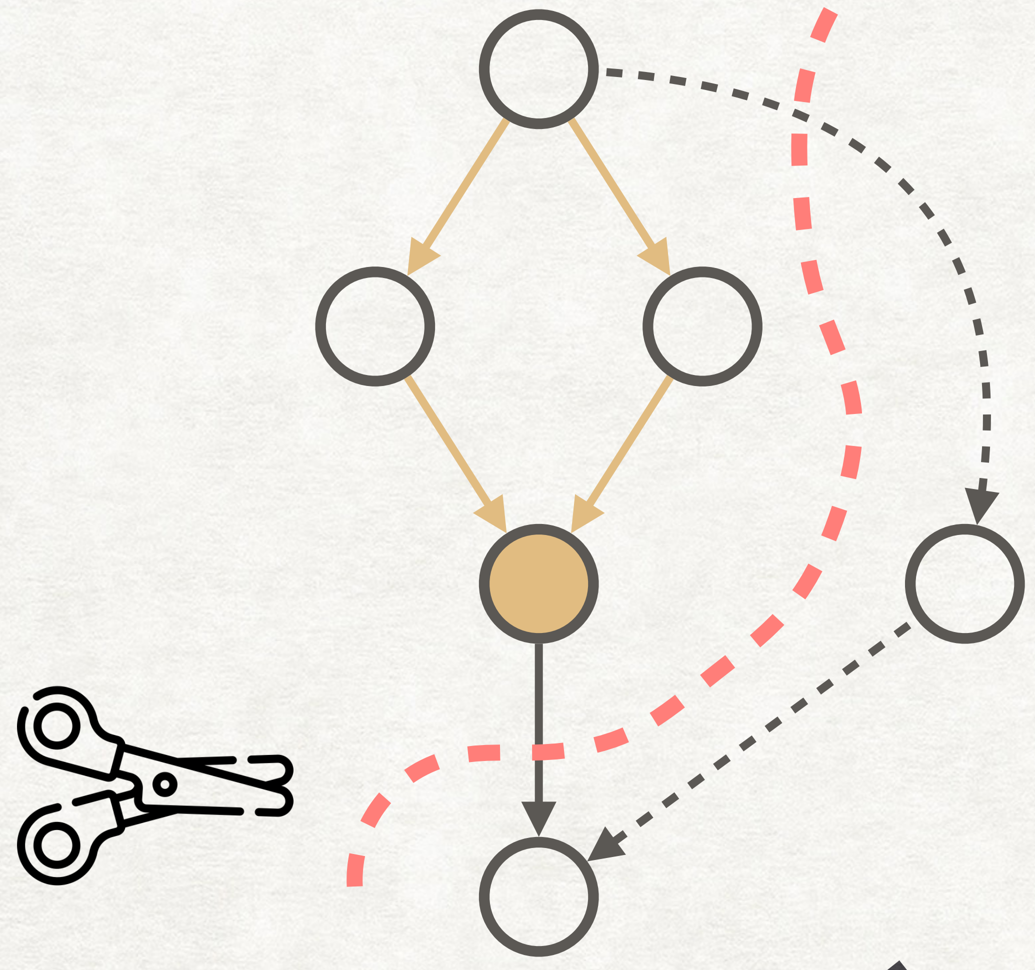
- Inference the dependency structure
- Approximate the degree of the program dependence

GOAL



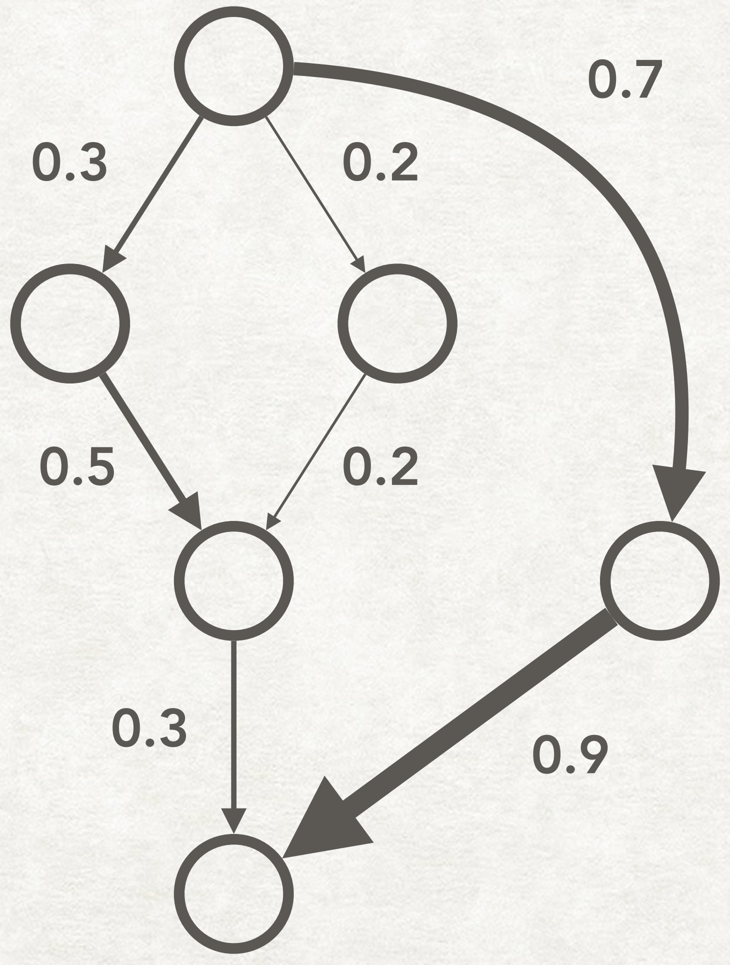
Program slicing technique

GOAL



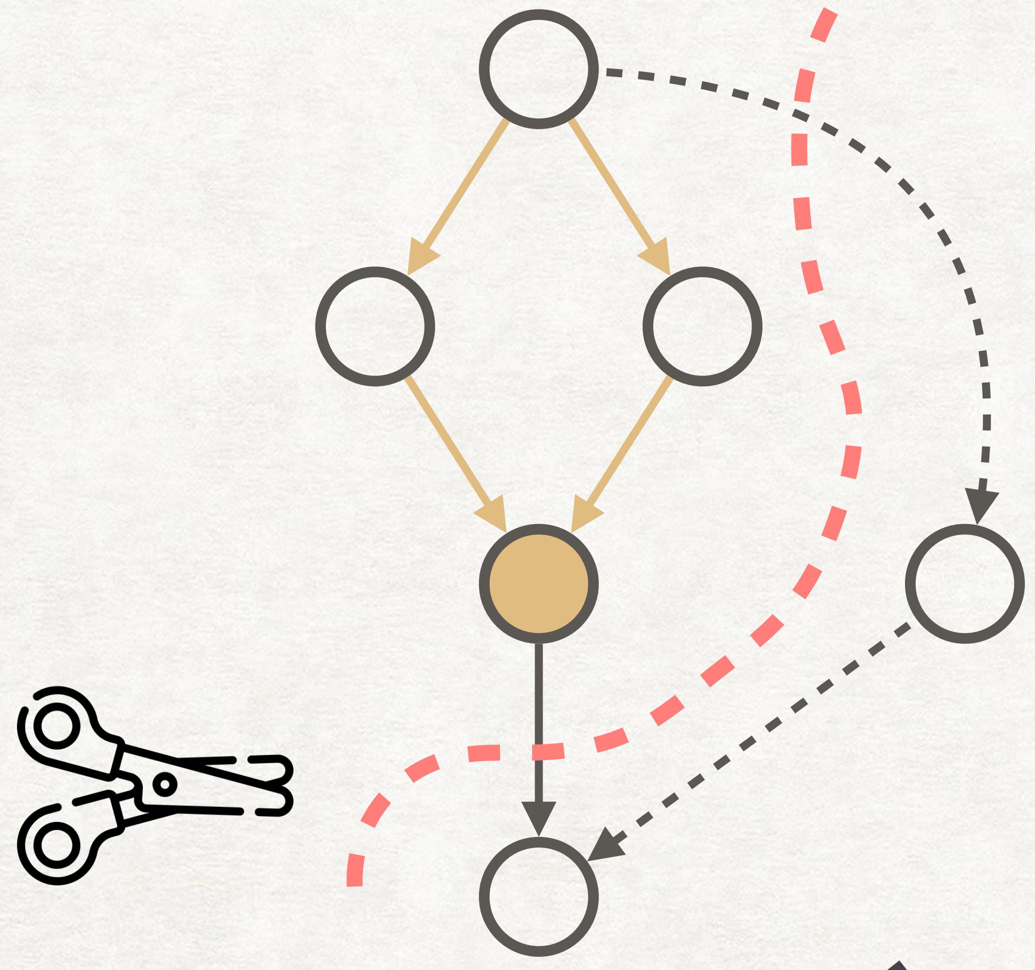
Program slicing technique

Enhance



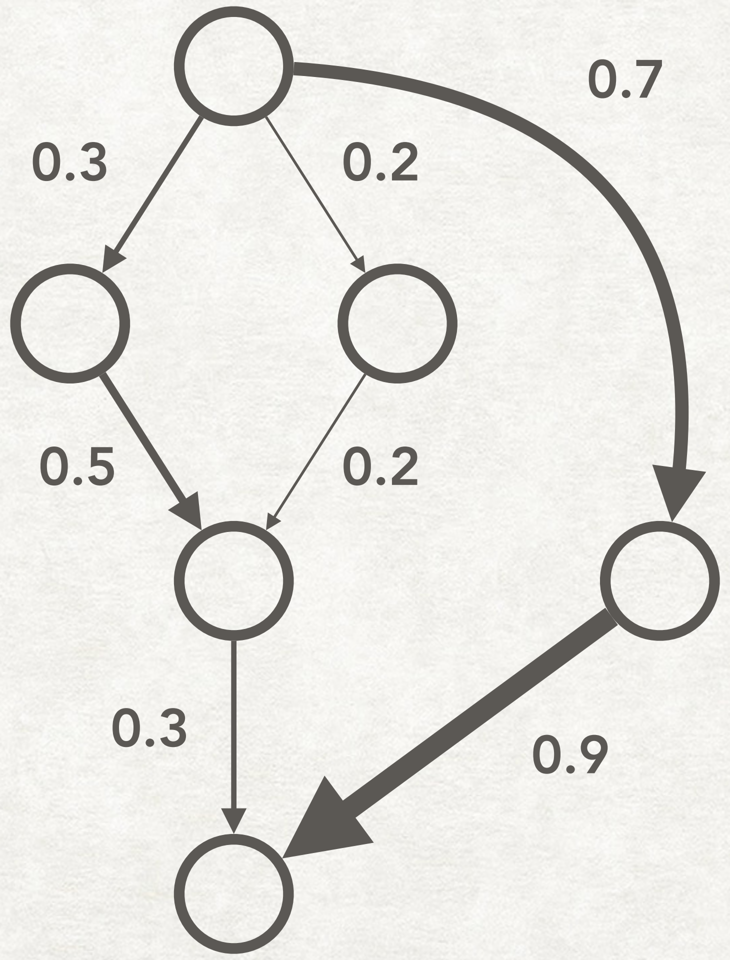
Dependency analysis technique

GOAL



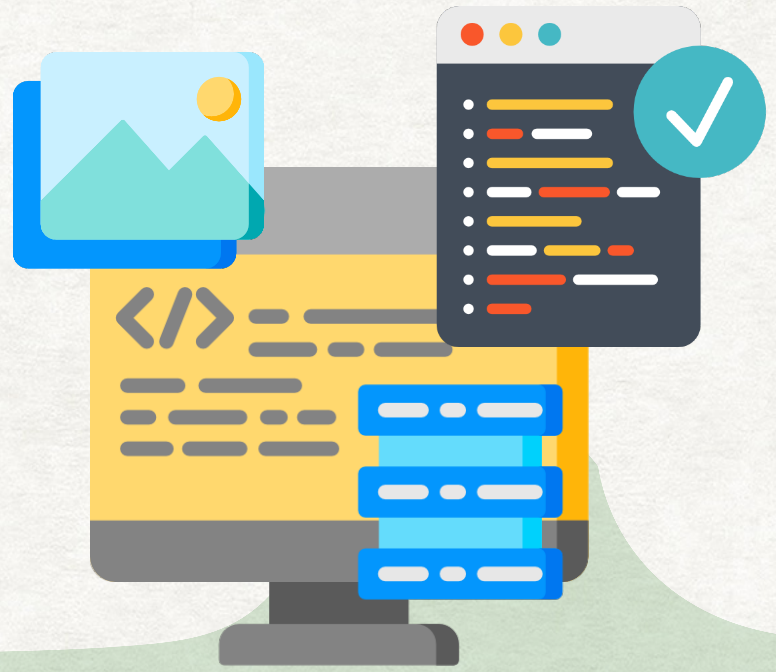
Program slicing technique

Enhance

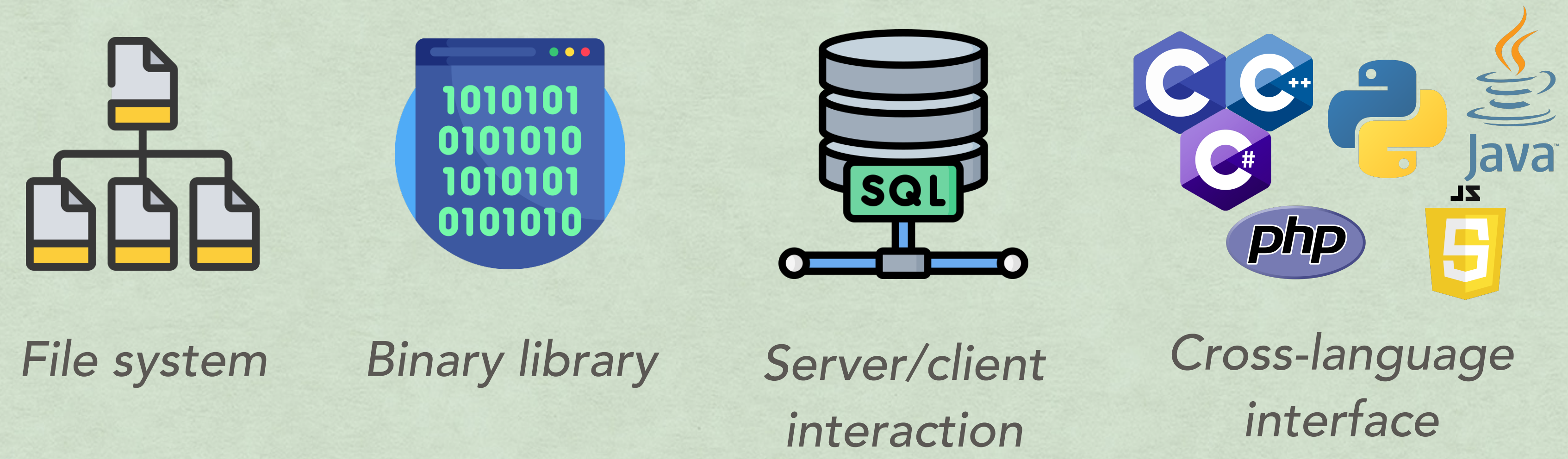


Dependency analysis technique

DEPENDENCY ANALYSIS



Software with non-conventional semantics



PUBLICATIONS

Research area	Publications
Lexical approximation (MOBS)	(**S. Lee**, D. Binkley, N. Gold, S. Islam, J. Krinke, S. Yoo) • "Evaluating lexical approximation of program dependence," *JSS'20* • [Poster] "MOBS: Multi-Operator Observation-based Slicing using Lexical Approximation of Program Dependence," *ICSE'18* • [Short] "Hyperheuristic Observation Based Slicing of Guava," *International Symposium on Search Based Software Engineering (SSBSE), 2017* • [Technical report] **S. Lee**, S. You, "Using Source Code Lexical Similarity to Improve Efficiency of Observation Based Slicing"
Statistical modeling (MOAD)	(**S. Lee**, D. Binkley, R.Feldt, N. Gold, S. Yoo) • "Observation-based Approximate Dependency Modeling and its Use for Program Slicing," *JSS'21* • "MOAD: Modeling Observation-based Approximate Dependency," *International Working Conference on Source Code Analysis and Manipulation (SCAM), 2019*
Causal inference (CPDA)	(**S. Lee**, D. Binkley, R.Feldt, N. Gold, S. Yoo) • "Causal Program Dependence Analysis," *to be submitted* • [Short paper] S. Oh, **S. Lee**, S. Yoo, "Effectively Sampling Higher Order Mutants Using Causal Effect," *International Workshop on Mutation Analysis (MUTATION), 2021* • [Technical report] "Causal Program Dependence Analysis and Causal Fault Localization"
Others	• [SE Note] W. B. Langdon, W. Weimer, J, Petke, E. Feredericks, **S. Lee**, E. Winter, M. Basios, M. B. Cohen, A. Blot, M. Wagner, B. R. Bruce, S. Yoo, S. Gerasimou, O. Karuss, Y. Huang, M. C. Gerten, "Genetic Improvement @ ICSE 2020," *ACM SIGSOFT Software Engineering Notes, 2020* • [Doctoral Symposium] **S. Lee**, "Scalable and Approximate Program Dependence Analysis," *ICSE'20* • [Industry] **S. Lee**, S. Hong, J, Yi, T. Kim, C. Kim, S. Yoo, "Classifying False Positive Static Checker Alarms in Continuous Integration Using Convolutional Neural Networks," *ICST'19* • [Short] G. An, J. Kim, **S. Lee**, S. Yoo, PyGGI: Python General framework for Genetic Improvement," *Korea Software Congress (KSC), 2017* • [Short] J. Sohn, **S. Lee**, S. Yoo, "Amortised Deep Parameter Optimisation of GPGPU Work Group Size for OpenCV," *SSBSE'16*

PUBLICATIONS

Research area	Publications
Lexical approximation (MOBS)	(**S. Lee**, D. Binkley, N. Gold, S. Islam, J. Krinke, S. Yoo) • "Evaluating lexical approximation of program dependence," *JSS'20* • [Poster] "MOBS: Multi-Operator Observation-based Slicing using Lexical Approximation of Program Dependence," *ICSE'18* • [Short] "Hyperheuristic Observation Based Slicing of Guava," *International Symposium on Search Based Software Engineering (SSBSE), 2017* • [Technical report] **S. Lee**, S. You, "Using Source Code Lexical Similarity to Improve Efficiency of Observation Based Slicing"
Statistical modeling (MOAD)	(**S. Lee**, D. Binkley, R.Feldt, N. Gold, S. Yoo) • "Observation-based Approximate Dependency Modeling and its Use for Program Slicing," *JSS'21* • "MOAD: Modeling Observation-based Approximate Dependency," *International Working Conference on Source Code Analysis and Manipulation (SCAM), 2019*
Causal inference (CPDA)	(**S. Lee**, D. Binkley, R.Feldt, N. Gold, S. Yoo) • "Causal Program Dependence Analysis," *to be submitted* • [Short paper] S. Oh, **S. Lee**, S. Yoo, "Effectively Sampling Higher Order Mutants Using Causal Effect," *International Workshop on Mutation Analysis (MUTATION), 2021* • [Technical report] "Causal Program Dependence Analysis and Causal Fault Localization"
Others	• [SE Note] W. B. Langdon, W. Weimer, J, Petke, E. Feredericks, **S. Lee**, E. Winter, M. Basios, M. B. Cohen, A. Blot, M. Wagner, B. R. Bruce, S. Yoo, S. Gerasimou, O. Karuss, Y. Huang, M. C. Gerten, "Genetic Improvement @ ICSE 2020," *ACM SIGSOFT Software Engineering Notes, 2020* • [Doctoral Symposium] **S. Lee**, "Scalable and Approximate Program Dependence Analysis," *ICSE'20* • [Industry] **S. Lee**, S. Hong, J, Yi, T. Kim, C. Kim, S. Yoo, "Classifying False Positive Static Checker Alarms in Continuous Integration Using Convolutional Neural Networks," *ICST'19* • [Short] G. An, J. Kim, **S. Lee**, S. Yoo, PyGGI: Python General framework for Genetic Improvement," *Korea Software Congress (KSC), 2017* • [Short] J. Sohn, **S. Lee**, S. Yoo, "Amortised Deep Parameter Optimisation of GPGPU Work Group Size for OpenCV," *SSBSE'16*

PUBLICATIONS

Research area	Publications
Lexical approximation (MOBS)	(**S. Lee**, D. Binkley, N. Gold, S. Islam, J. Krinke, S. Yoo) • "Evaluating lexical approximation of program dependence," *JSS'20* • [Poster] "MOBS: Multi-Operator Observation-based Slicing using Lexical Approximation of Program Dependence," *ICSE'18* • [Short] "Hyperheuristic Observation Based Slicing of Guava," *International Symposium on Search Based Software Engineering (SSBSE), 2017* • [Technical report] **S. Lee**, S. You, "Using Source Code Lexical Similarity to Improve Efficiency of Observation Based Slicing"
Statistical modeling (MOAD)	(**S. Lee**, D. Binkley, R.Feldt, N. Gold, S. Yoo) • "Observation-based Approximate Dependency Modeling and its Use for Program Slicing," *JSS'21* • "MOAD: Modeling Observation-based Approximate Dependency," *International Working Conference on Source Code Analysis and Manipulation (SCAM), 2019*
Causal inference (CPDA)	(**S. Lee**, D. Binkley, R.Feldt, N. Gold, S. Yoo) • "Causal Program Dependence Analysis," *to be submitted* • [Short paper] S. Oh, **S. Lee**, S. Yoo, "Effectively Sampling Higher Order Mutants Using Causal Effect," *International Workshop on Mutation Analysis (MUTATION), 2021* • [Technical report] "Causal Program Dependence Analysis and Causal Fault Localization"
Others	• [SE Note] W. B. Langdon, W. Weimer, J, Petke, E. Feredericks, **S. Lee**, E. Winter, M. Basios, M. B. Cohen, A. Blot, M. Wagner, B. R. Bruce, S. Yoo, S. Gerasimou, O. Karuss, Y. Huang, M. C. Gerten, "Genetic Improvement @ ICSE 2020," *ACM SIGSOFT Software Engineering Notes, 2020* • [Doctoral Symposium] **S. Lee**, "Scalable and Approximate Program Dependence Analysis," *ICSE'20* • [Industry] **S. Lee**, S. Hong, J, Yi, T. Kim, C. Kim, S. Yoo, "Classifying False Positive Static Checker Alarms in Continuous Integration Using Convolutional Neural Networks," *ICST'19* • [Short] G. An, J. Kim, **S. Lee**, S. Yoo, PyGGI: Python General framework for Genetic Improvement," *Korea Software Congress (KSC), 2017* • [Short] J. Sohn, **S. Lee**, S. Yoo, "Amortised Deep Parameter Optimisation of GPGPU Work Group Size for OpenCV," *SSBSE'16*

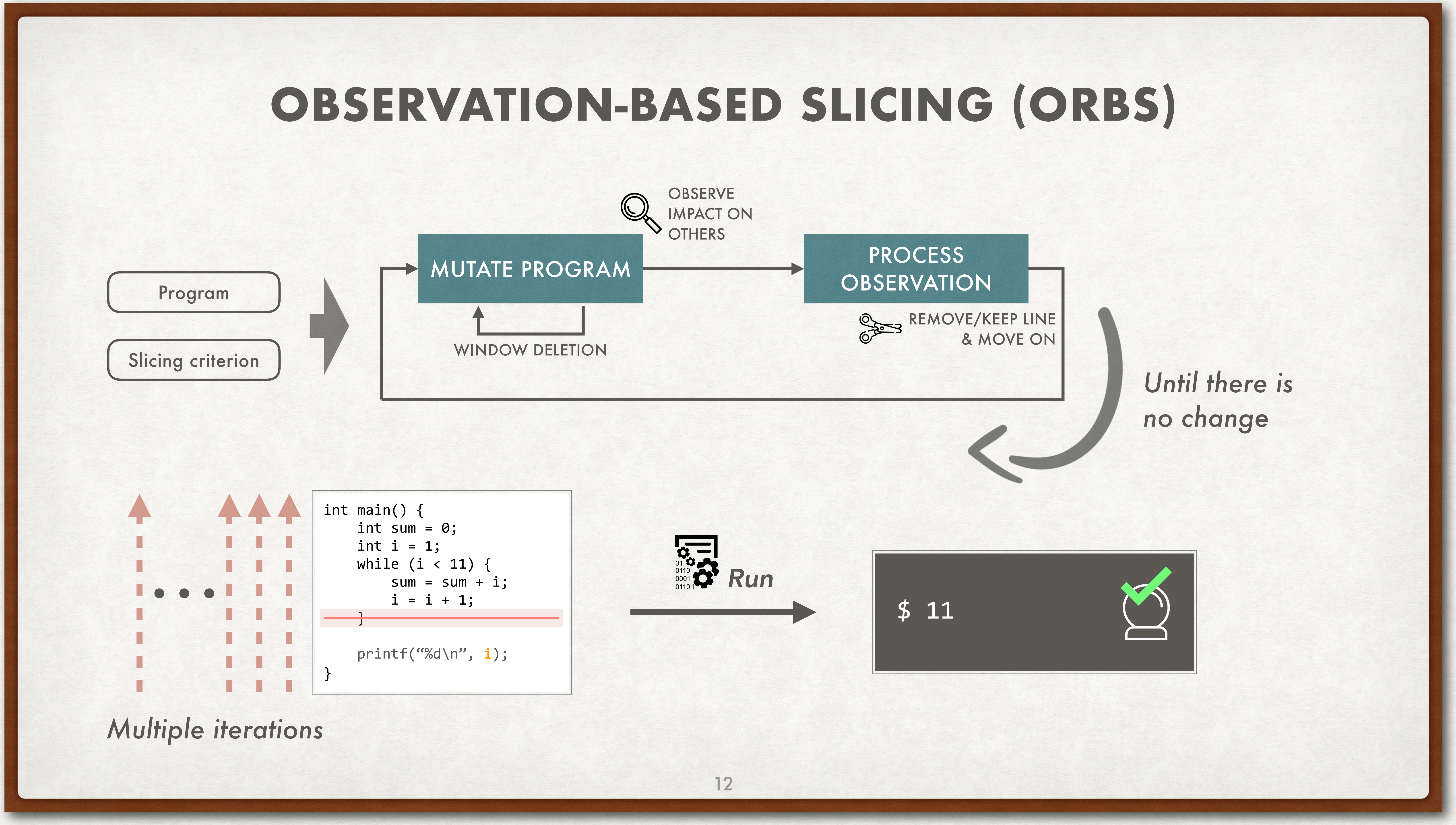
PUBLICATIONS

Research area	Publications
Lexical approximation (MOBS)	(**S. Lee**, D. Binkley, N. Gold, S. Islam, J. Krinke, S. Yoo) • "Evaluating lexical approximation of program dependence," *JSS'20* • [Poster] "MOBS: Multi-Operator Observation-based Slicing using Lexical Approximation of Program Dependence," *ICSE'18* • [Short] "Hyperheuristic Observation Based Slicing of Guava," *International Symposium on Search Based Software Engineering (SSBSE), 2017* • [Technical report] **S. Lee**, S. You, "Using Source Code Lexical Similarity to Improve Efficiency of Observation Based Slicing"
Statistical modeling (MOAD)	(**S. Lee**, D. Binkley, R.Feldt, N. Gold, S. Yoo) • "Observation-based Approximate Dependency Modeling and its Use for Program Slicing," *JSS'21* • "MOAD: Modeling Observation-based Approximate Dependency," *International Working Conference on Source Code Analysis and Manipulation (SCAM), 2019*
Causal inference (CPDA)	(**S. Lee**, D. Binkley, R.Feldt, N. Gold, S. Yoo) • "Causal Program Dependence Analysis," *to be submitted* • [Short paper] S. Oh, **S. Lee**, S. Yoo, "Effectively Sampling Higher Order Mutants Using Causal Effect," *International Workshop on Mutation Analysis (MUTATION), 2021* • [Technical report] "Causal Program Dependence Analysis and Causal Fault Localization"
Others	• [SE Note] W. B. Langdon, W. Weimer, J, Petke, E. Feredericks, **S. Lee**, E. Winter, M. Basios, M. B. Cohen, A. Blot, M. Wagner, B. R. Bruce, S. Yoo, S. Gerasimou, O. Karuss, Y. Huang, M. C. Gerten, "Genetic Improvement @ ICSE 2020," *ACM SIGSOFT Software Engineering Notes, 2020* • [Doctoral Symposium] **S. Lee**, "Scalable and Approximate Program Dependence Analysis," *ICSE'20* • [Industry] **S. Lee**, S. Hong, J, Yi, T. Kim, C. Kim, S. Yoo, "Classifying False Positive Static Checker Alarms in Continuous Integration Using Convolutional Neural Networks," *ICST'19* • [Short] G. An, J. Kim, **S. Lee**, S. Yoo, PyGGI: Python General framework for Genetic Improvement," *Korea Software Congress (KSC), 2017* • [Short] J. Sohn, **S. Lee**, S. Yoo, "Amortised Deep Parameter Optimisation of GPGPU Work Group Size for OpenCV," *SSBSE'16*

MOBS

[*Approximate the dependence utilizing the lexical model and increase the scalability*]

RECALL ORBS

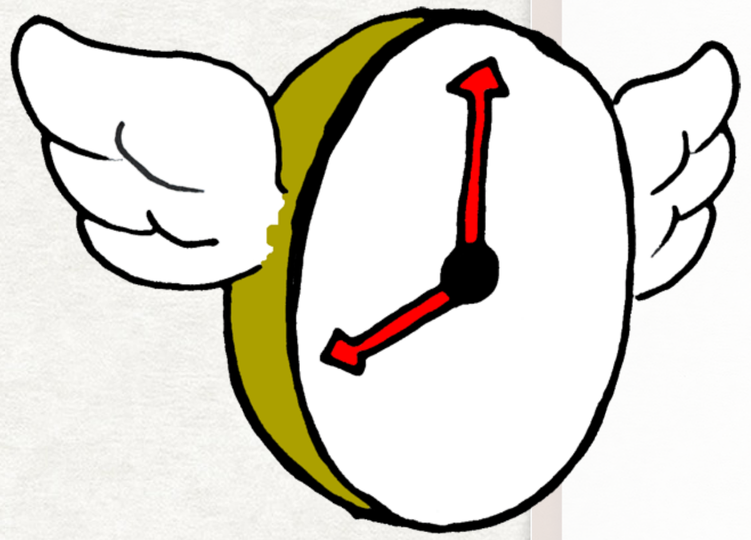

OBSERVATION-BASED SLICING (ORBS)
OBSERVE IMPACT ON OTHERS
Program
Slicing criterion
MUTATE PROGRAM
PROCESS OBSERVATION
WINDOW DELETION
REMOVE/KEEP LINE & MOVE ON
Until there is no change
int main() {
int sum = 0;
int i = 1;
while (i < 11) {
sum = sum + i;
i = i + 1;
}
printf("%d\n", i);
}
Run
$ 11
Multiple iterations
12

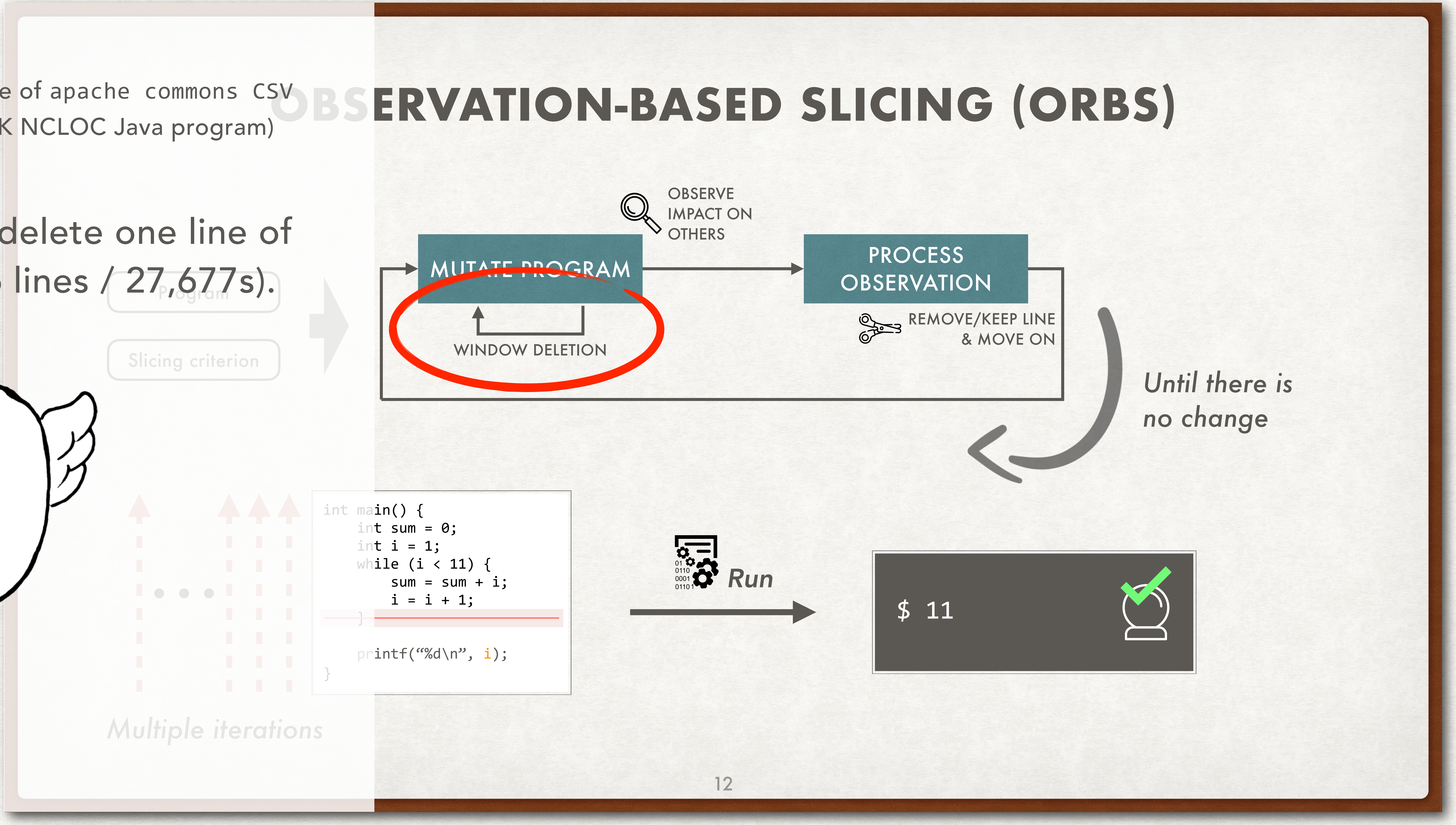
RECALL ORBS

Scalability

In case of apache commons CSV (1.5K NCLOC Java program)

- ORBS takes ~40s to delete one line of the source code (696 lines / 27,677s).





RECALL ORBS

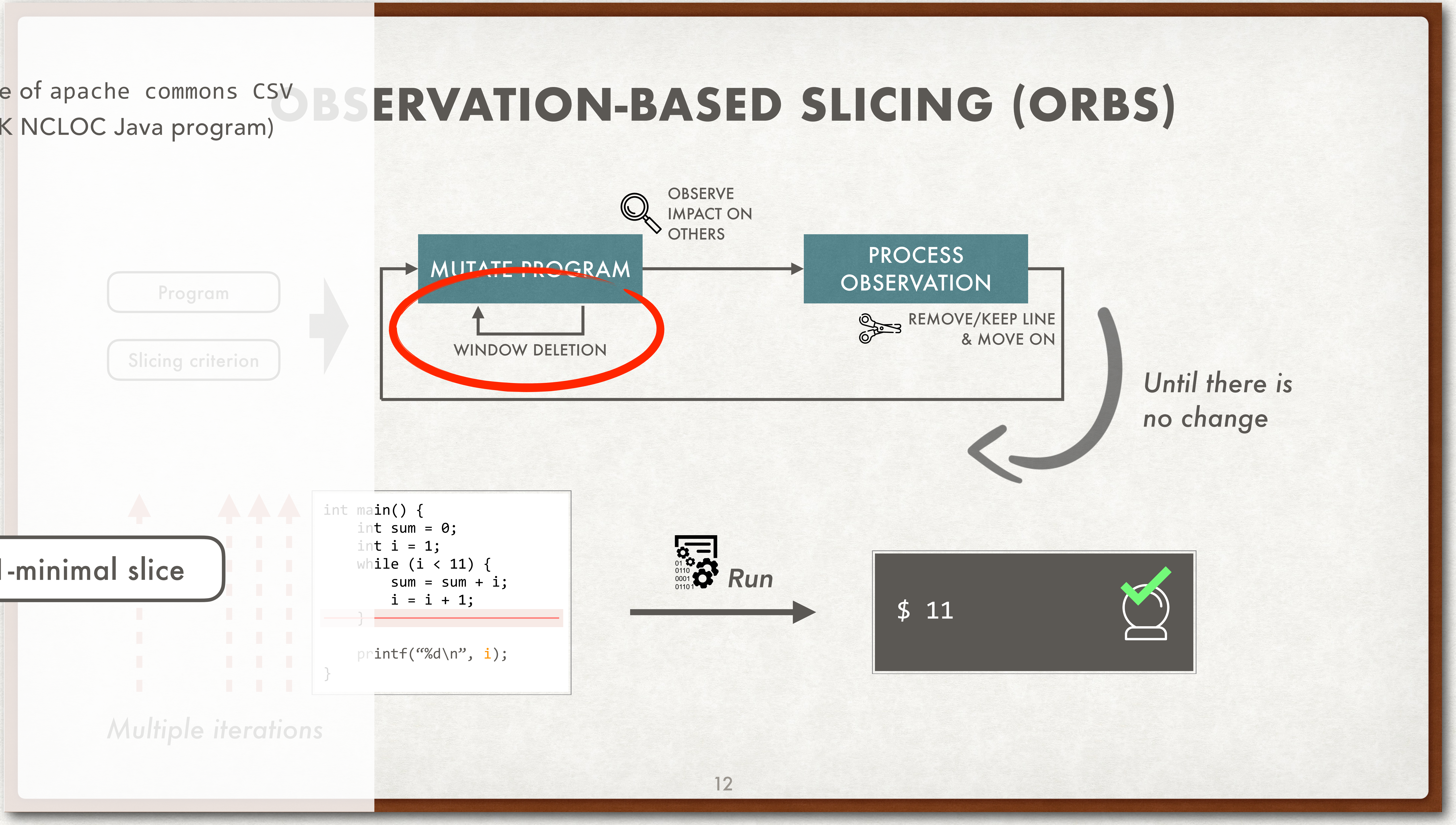
Scalability

In case of apache commons CSV
(1.5K NCLOC Java program)

Iterations

...

1-minimal slice



RECALL ORBS

Scalability

In case of apache commons CSV (1.5K NCLOC Java program)

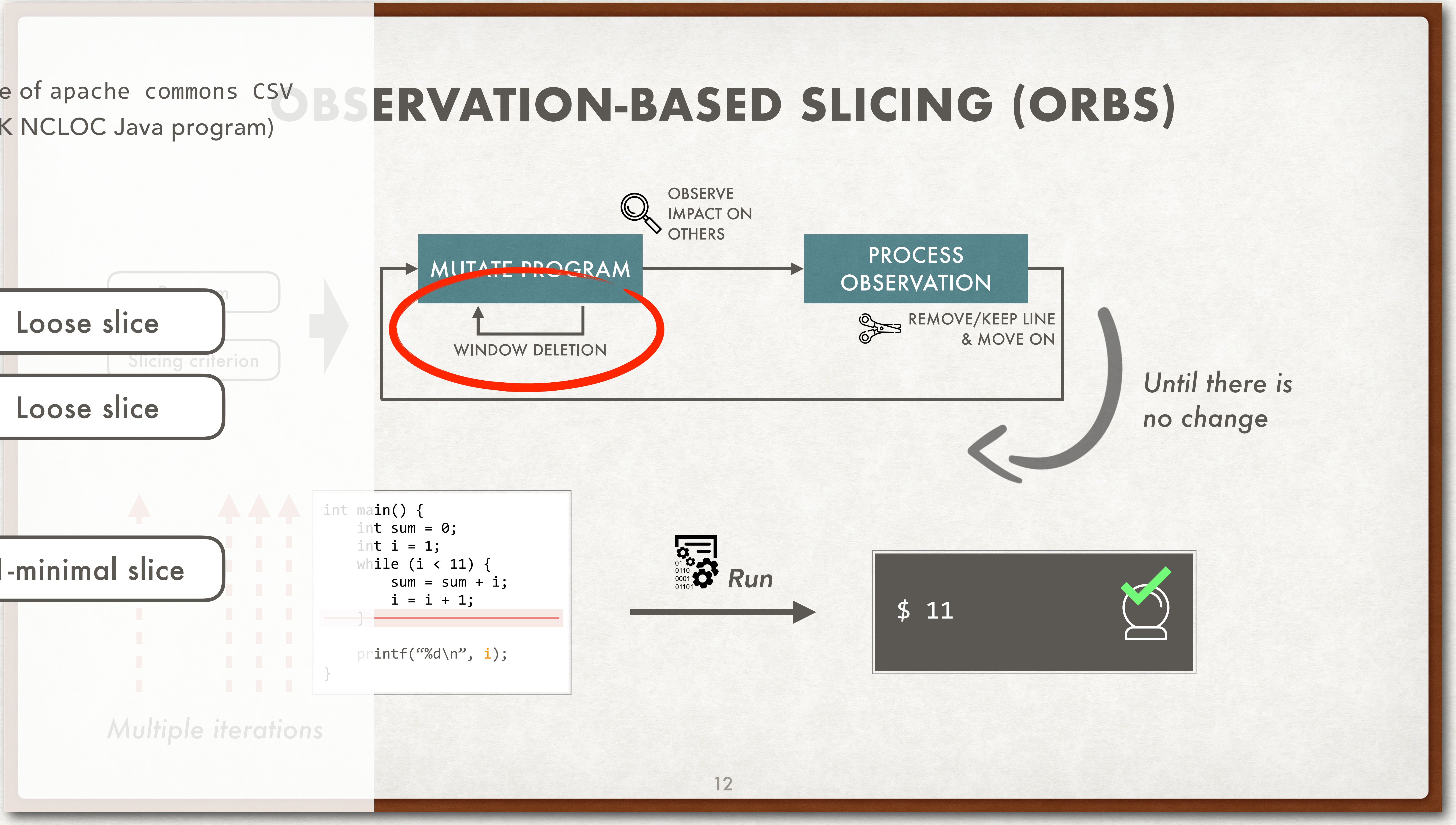
Iterations

Loose slice

Loose slice

• • •

1-minimal slice



RECALL ORBS

Scalability

In case of apache commons CSV
(1.5K NCLOC Java program)

Iterations

~ 9500s

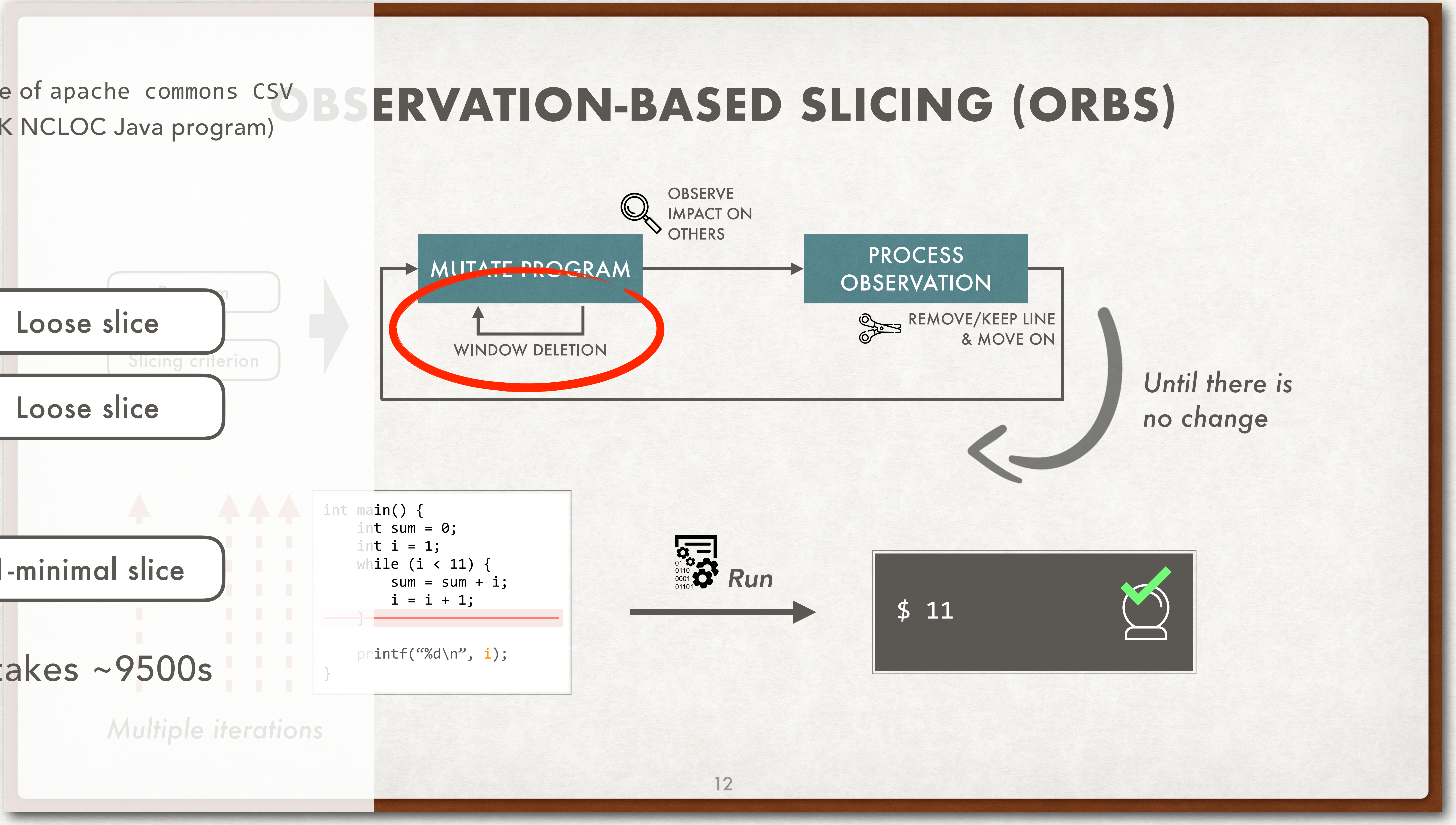
Loose slice

Loose slice

• • •

1-minimal slice

- 1st ORBS iteration takes ~9500s

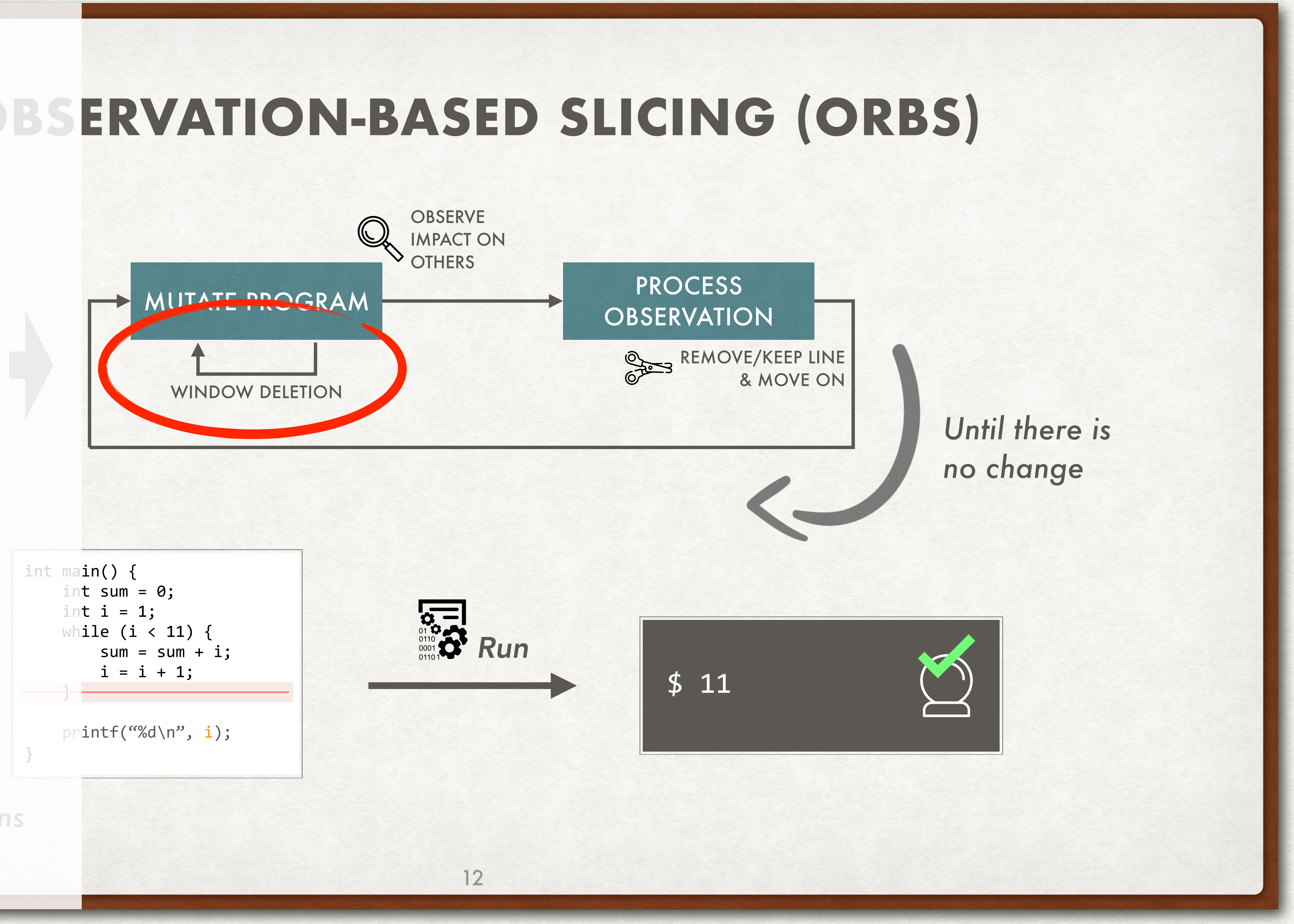


RECALL ORBS

Window deletion operator

```
for window ⇐ 1 to n; do
|   succ, P′ ⇐ delete(P, lineno, lineno − 1,
|                       ..., lineno − window + 1)
|   if (succ)
|   |   P ⇐ P′
|   end
end
lineno = lineno − 1
```

- Aims to find structural dependency
- Deletes maximum 1 line per deletion
- Deletes consecutive lines only



LEXICAL INFORMATION IN SOURCE CODE

LEXICAL INFORMATION IN SOURCE CODE

- Java

```java

    private static final Logger logger = Logger.getLogger(FinalizableReferenceQueue.class.getNa

    private static final String FINALIZER_CLASS_NAME = "com.google.common.base.internal.Finalize

```

```java
        try {
          ((FinalizableReference) reference).finalizeReferent();
        } catch (Throwable t) {
          logger.log(Level.SEVERE, "Error cleaning up after reference.", t);
        }
```

- Python

```python
        except Exception:
            if not from_error_handler:
                raise
            self.logger.exception('Request finalizing failed with an ' 'error while handling
        return response
```

- Code lines handling the logging function contain the word 'log.'

LEXICAL INFORMATION IN SOURCE CODE

- Java

```java
private static final Logger logger = Logger.getLogger(FinalizableReferenceQueue.class.getNa

private static final String FINALIZER_CLASS_NAME = "com.google.common.base.internal.Finaliz
```

```java
    try {
      ((FinalizableReference) reference).finalizeReferent();
    } catch (Throwable t) {
      logger.log(Level.SEVERE, "Error cleaning up after reference.", t);
    }
```

- Python

```python
        except Exception:
            if not from_error_handler:
                raise
            self.logger.exception('Request finalizing failed with an ' 'error while handling
        return response
```

- Code lines handling the logging function contain the word 'log.'

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 34, NO. 2, MARCH/APRIL 2008 287

Using the Conceptual Cohesion of Classes for Fault Prediction in Object-Oriented Systems

Andrian Marcus, *Member, IEEE Computer Society*,
Denys Poshyvanyk, *Student Member, IEEE*, and Rudolf Ferenc

Abstract—High cohesion is a desirable property of software as it positively impacts understanding, reuse, and maintenance. Currently proposed measures for cohesion in Object-Oriented (OO) software reflect particular interpretations of cohesion and capture different aspects of it. Existing approaches are largely based on using the structural information from the source code, such as attribute references, in methods to measure cohesion. This paper proposes a new measure for the cohesion of classes in OO software systems based on the analysis of the unstructured information embedded in the source code, such as comments and identifiers. The measure, named the Conceptual Cohesion of Classes (C3), is inspired by the mechanisms used to measure textual coherence in cognitive psychology and computational linguistics. This paper presents the principles and the technology that stand behind the C3 measure. A large case study on three open source software systems is presented which compares the new measure with an extensive set of existing metrics and uses them to construct models that predict software faults. The case study shows that the novel measure captures different aspects of class cohesion compared to any of the existing cohesion measures. In addition, combining C3 with existing structural cohesion metrics proves to be a better predictor of faulty classes when compared to different combinations of structural cohesion metrics.

Index Terms—Software cohesion, textual coherence, fault prediction, fault proneness, program comprehension, information retrieval, Latent Semantic Indexing

The Journal of Systems and Software 82 (2009) 1793–1803

ELSEVIER

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Increasing diversity: Natural language measures for software fault prediction

David Binkley [a], Henry Feild [b], Dawn Lawrie [a,*], Maurizio Pighin [c]

[a] *Loyola College Baltimore, MD 21210, USA*
[b] *University of Massachusetts, Amherst, MA 01003, USA*
[c] *Universitá degli Studi di Udine, Italy*

A R T I C L E I N F O

Article history:
Available online 26 June 2009

Keywords:
Information retrieval
Code comprehension
Fault prediction
Linear regression models

A B S T R A C T

While challenging, the ability to predict faulty modules of a program is valuable to a software project because it can reduce the cost of software development, as well as software maintenance and evolution. Three language-processing based measures are introduced and applied to the problem of fault prediction. The first measure is based on the usage of natural language in a program's identifiers. The second measure concerns the conciseness and consistency of identifiers. The third measure, referred to as the QALP score, makes use of techniques from information retrieval to judge software quality. The QALP score has been shown to correlate with human judgments of software quality.

2015 IEEE/ACM 37th IEEE International Conference on Software Engineering

An Information Retrieval Approach for Regression Test Prioritization Based on Program Changes

Ripon K. Saha* Lingming Zhang† Sarfraz Khurshid* Dewayne E. Perry*
*Electrical and Computer Engineering, The University of Texas at Austin, USA 78712
Email: ripon@utexas.edu, khurshid@ece.utexas.edu, perry@ece.utexas.edu
† Department of Computer Science, The University of Texas at Dallas, USA 75080
Email: lingming.zhang@utdallas.edu

***Abstract*—Regression testing is widely used in practice for validating program changes. However, running large regression suites can be costly. Researchers have developed several techniques for *prioritizing* tests such that the higher-priority tests have a higher likelihood of finding bugs. A vast majority of these techniques are based on *dynamic* analysis, which can be precise but can also have significant overhead (e.g., for program instrumentation and test-coverage collection). We introduce a new approach, REPiR, to address the problem of regression test prioritization by reducing it to a standard Information Retrieval problem such that the differences between two program versions form the *query* and the tests constitute the *document collection*.**

values of the remaining test cases taking into account the influence of already prioritized test cases.

Although a number of RTP techniques (specifically coverage-based ones) have been widely used, they have two key limitations [39]. First, coverage profiling overhead (in terms of time and space) can be significant. Second, in the context of certain program changes (which modify behavior significantly) the coverage information from the previous version can be imprecise to guide test prioritization for the current version. Although the static techniques [39], [66] address

LEXICAL INFORMATION IN SOURCE CODE

- Java

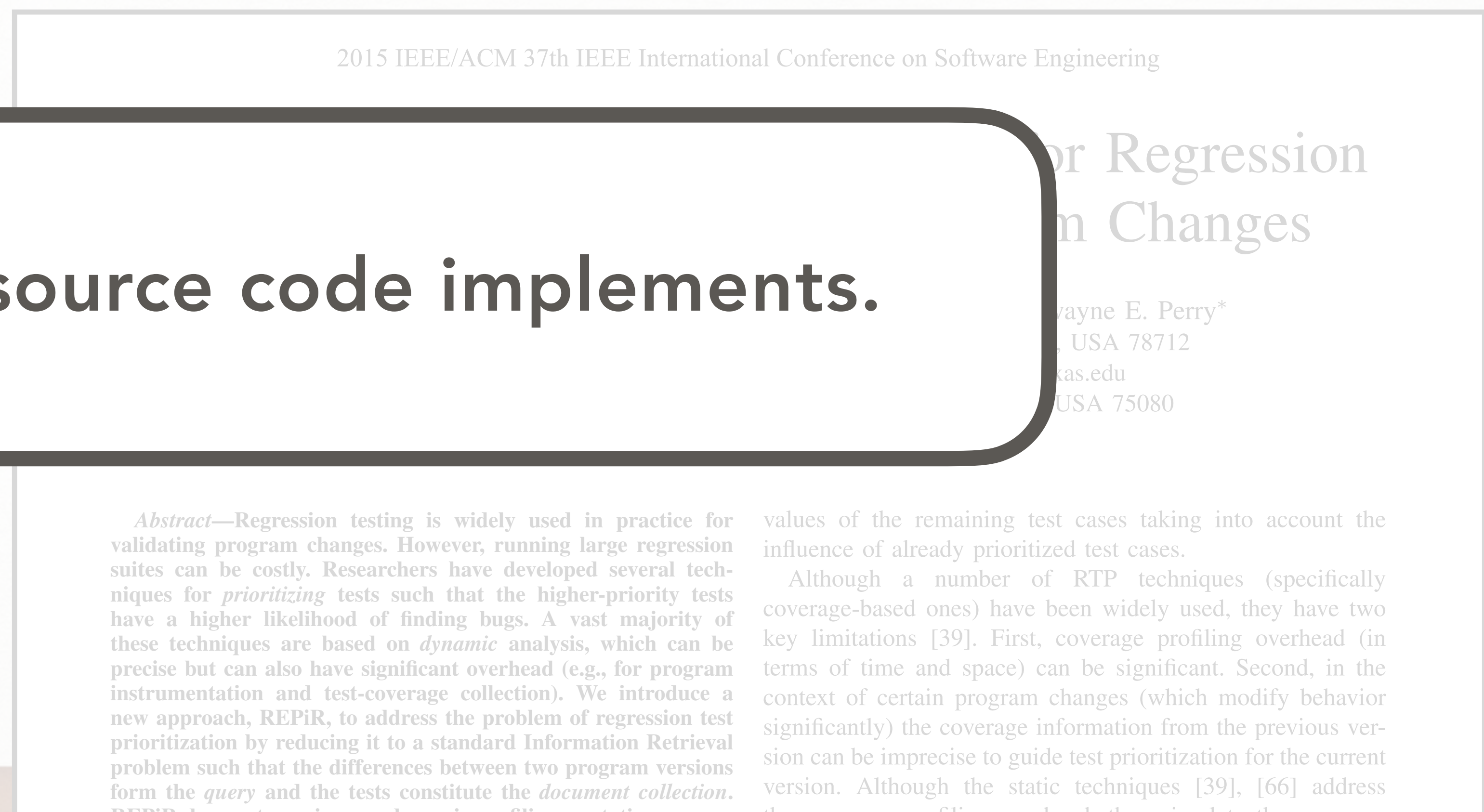
```java
private static final Logger logger = Logger.getLogger(FinalizableReferenceQueue.class.getNar

private static final String FINALIZER_CLASS_NAME = "com.google.common.base.internal.Finalize
```

```java
try {
  ((FinalizableReference) reference).finalizeReferent();
} catch (Throwable t) {
  logger.log(Level.SEVERE, "Error cleaning up after reference.", t);
}
```

- Python

```python
except Exception:
    if not from_error_handler:
        raise
    self.logger.exception('Request finalizing failed with an ' 'error while handling
return response
```

- Code lines handling the logging function contain the word 'log.'



The tokens used in the source code reflect the functionality that the source code implements.

LEXICAL DELETION OPERATOR

```
...
logger.log(Level.SEVER, "…");
...

logger.log(Level.WARNING, "…");
...
Logger logger = Logger.getLogger(…);
...
```

LEXICAL DELETION OPERATOR

```
...
logger.log(Level.SEVER, "…");     0.85
...

logger.log(Level.WARNING, "…");   0.8
...
Logger logger = Logger.getLogger(…);
...
```

LEXICAL DELETION OPERATOR

SHARE THE FUNCTIONALITY

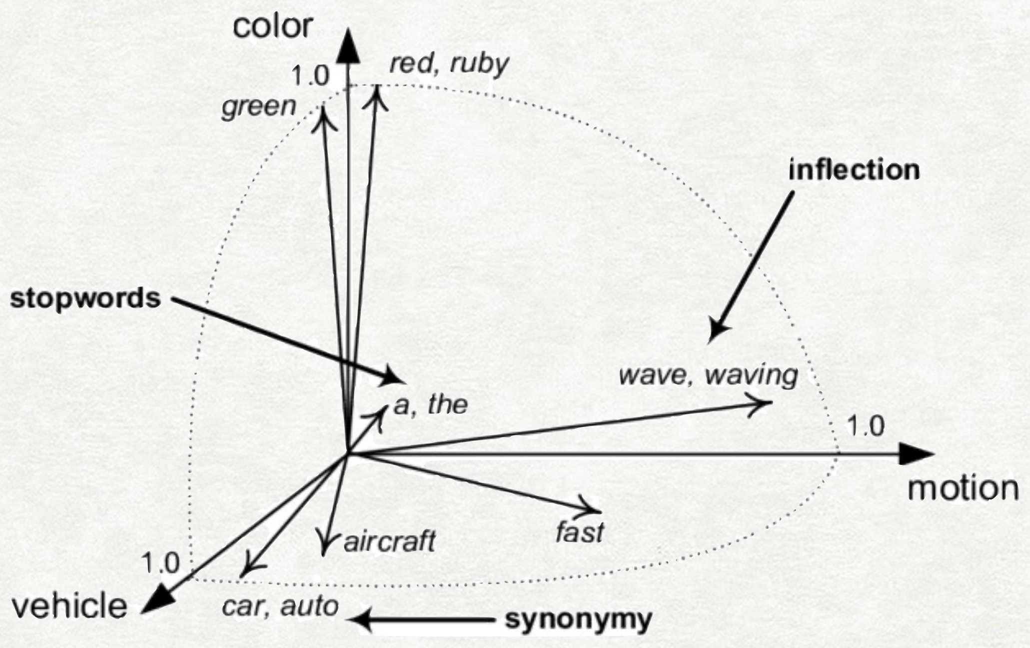
```
...
logger.log(Level.SEVER, "…");      0.85
...
logger.log(Level.WARNING, "…");    0.8
...
Logger logger = Logger.getLogger(…);
...
```

LEXICAL DELETION OPERATOR

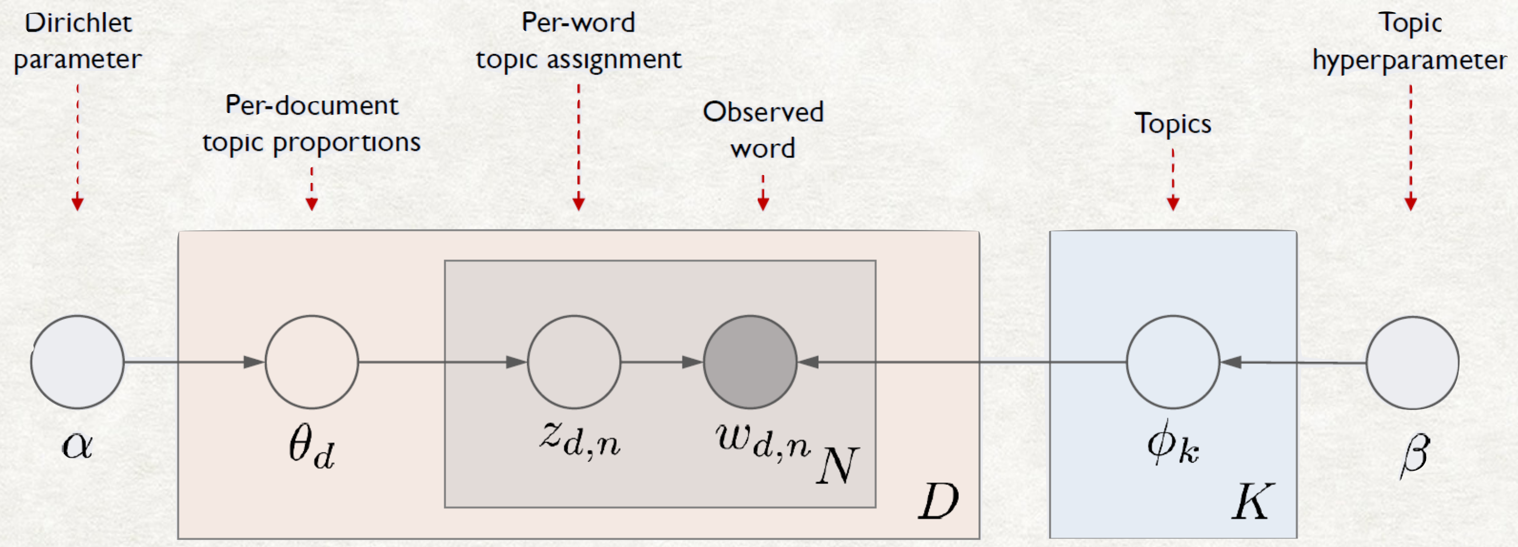
SHARE THE FUNCTIONALITY

Two language models

1. Vector Space Model (VSM)



2. Latent Dirichlet Allocation (LDA)

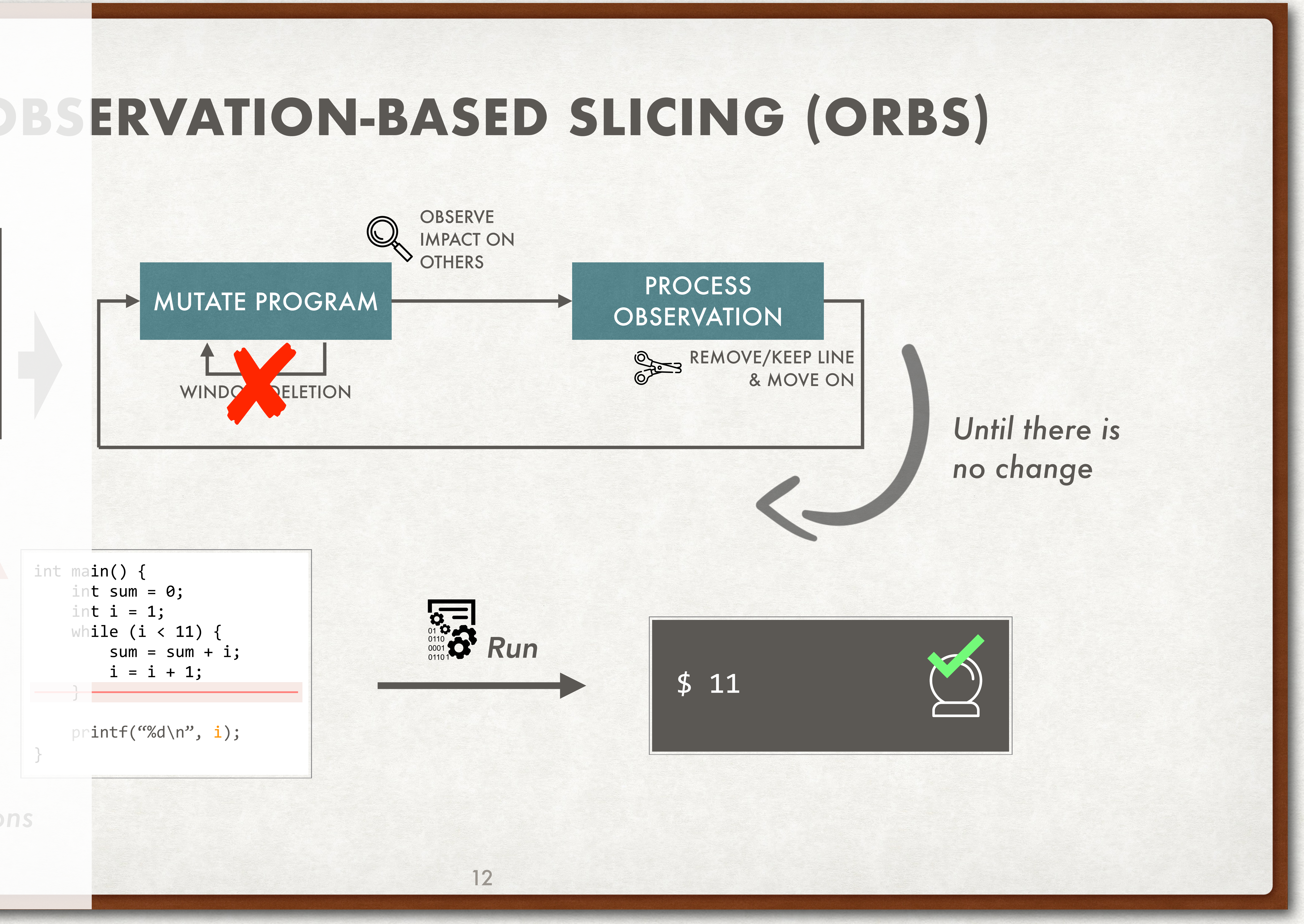


RECALL ORBS

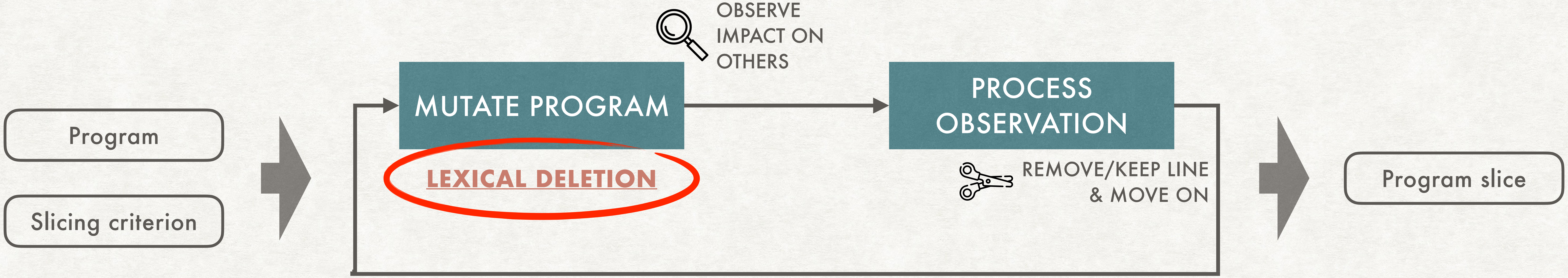
Lexical deletion operator

```
similar_lines = Lex_model(curr_line, threshold)
succ, P' ⇐ delete(P, curr_line, similar_lines)
if (succ)
|   P ⇐ P'
end
lineno = lineno - 1
```

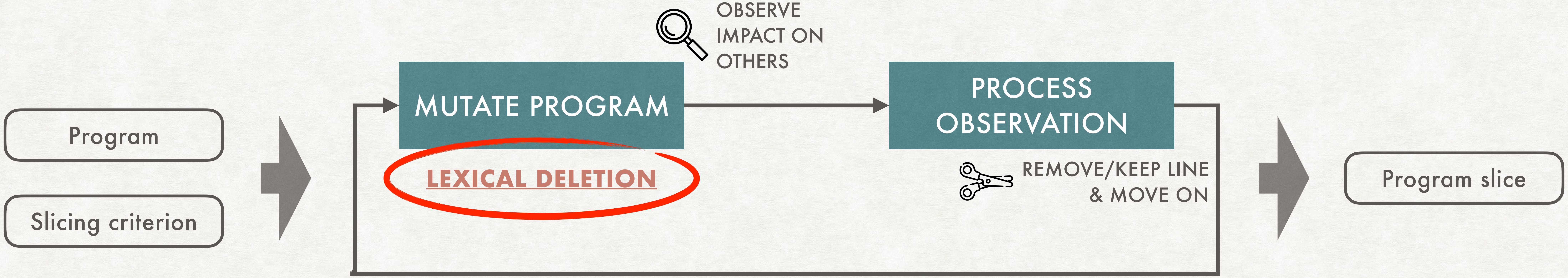
- Aims to find lexical dependency
- Can delete
  - arbitrary number of similar lines
  - non-consecutive lines



LEXICAL SIMILARITY-BASED ORBS (LS-ORBS)

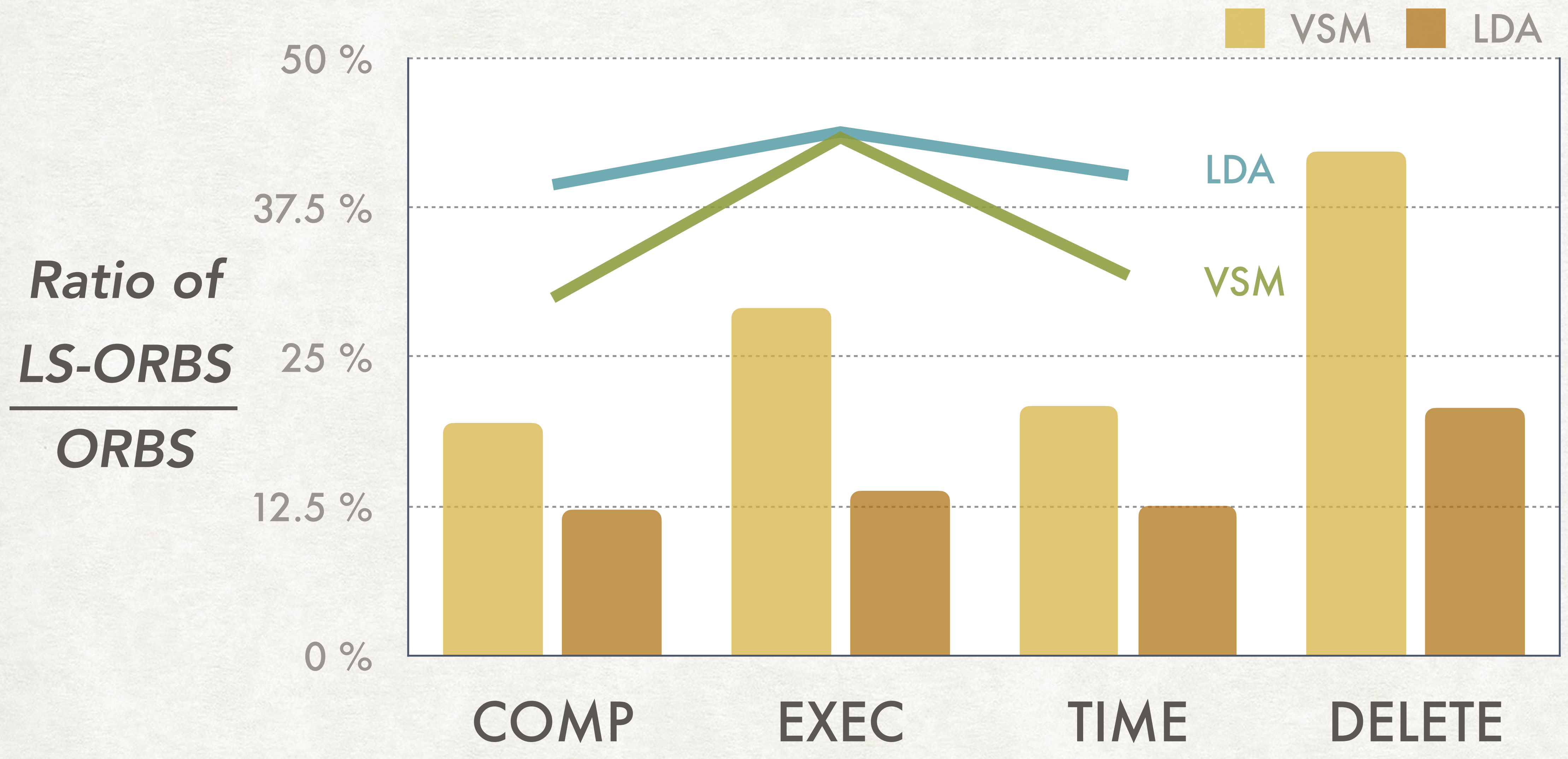


LEXICAL SIMILARITY-BASED ORBS (LS-ORBS)



ORBS VS. LS-ORBS

- Benchmarks: 18 slicing criteria from Java and C programs
  - Java: apache commons CSV, CLI, and guava library
  - C: Siemens suite



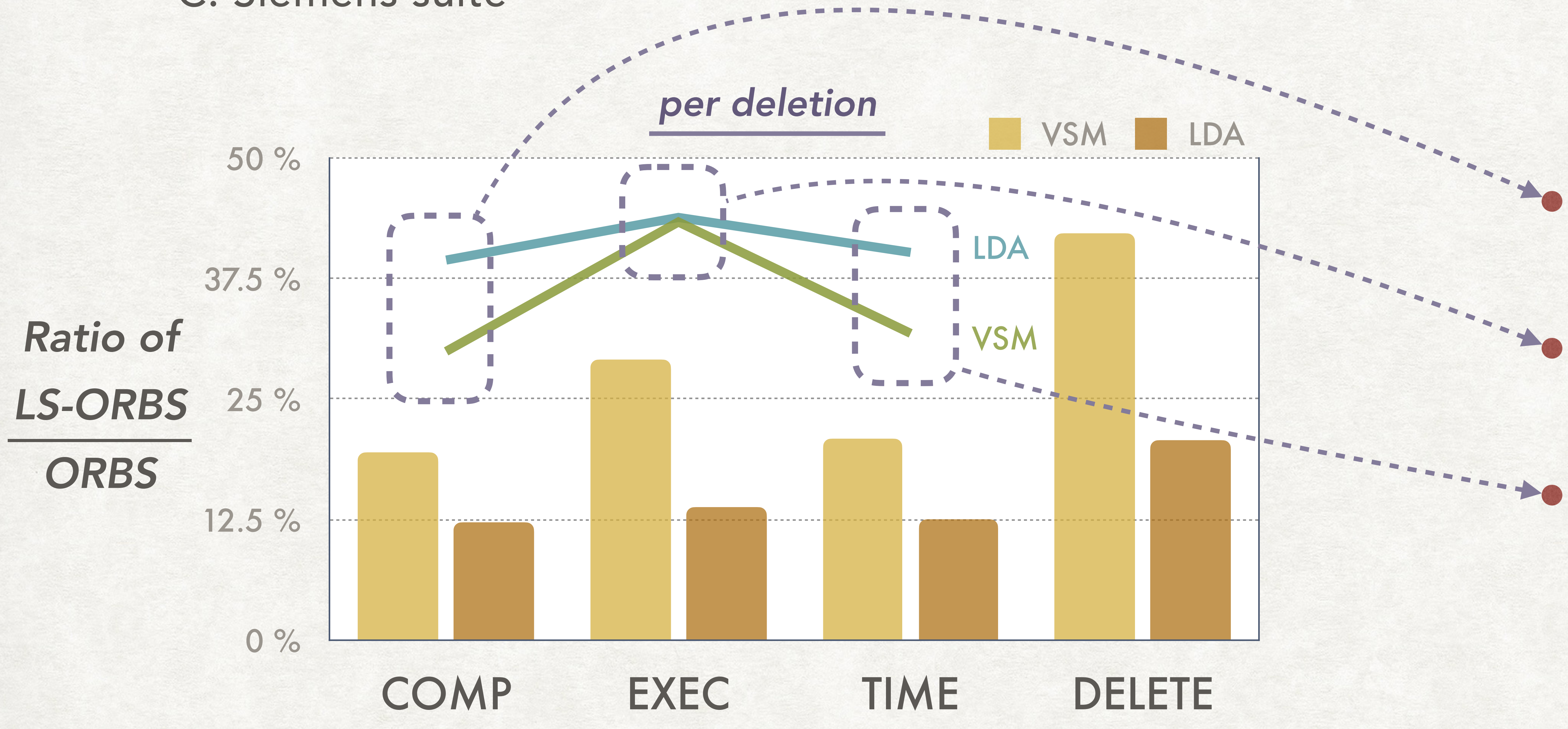
LS-ORBS achieves, or uses

- 👍 **55%** # of compilations / del,
- 👍 **70%** # of executions / del,
- 👍 **57%** time taken / del,
- 👎 **38%** # of deleted lines

compared to ORBS

ORBS VS. LS-ORBS

- Benchmarks: 18 slicing criteria from Java and C programs
    - Java: apache commons CSV, CLI, and guava library
    - C: Siemens suite



LS-ORBS achieves, or uses

👍 **55%** # of compilations / del,

👍 **70%** # of executions / del,

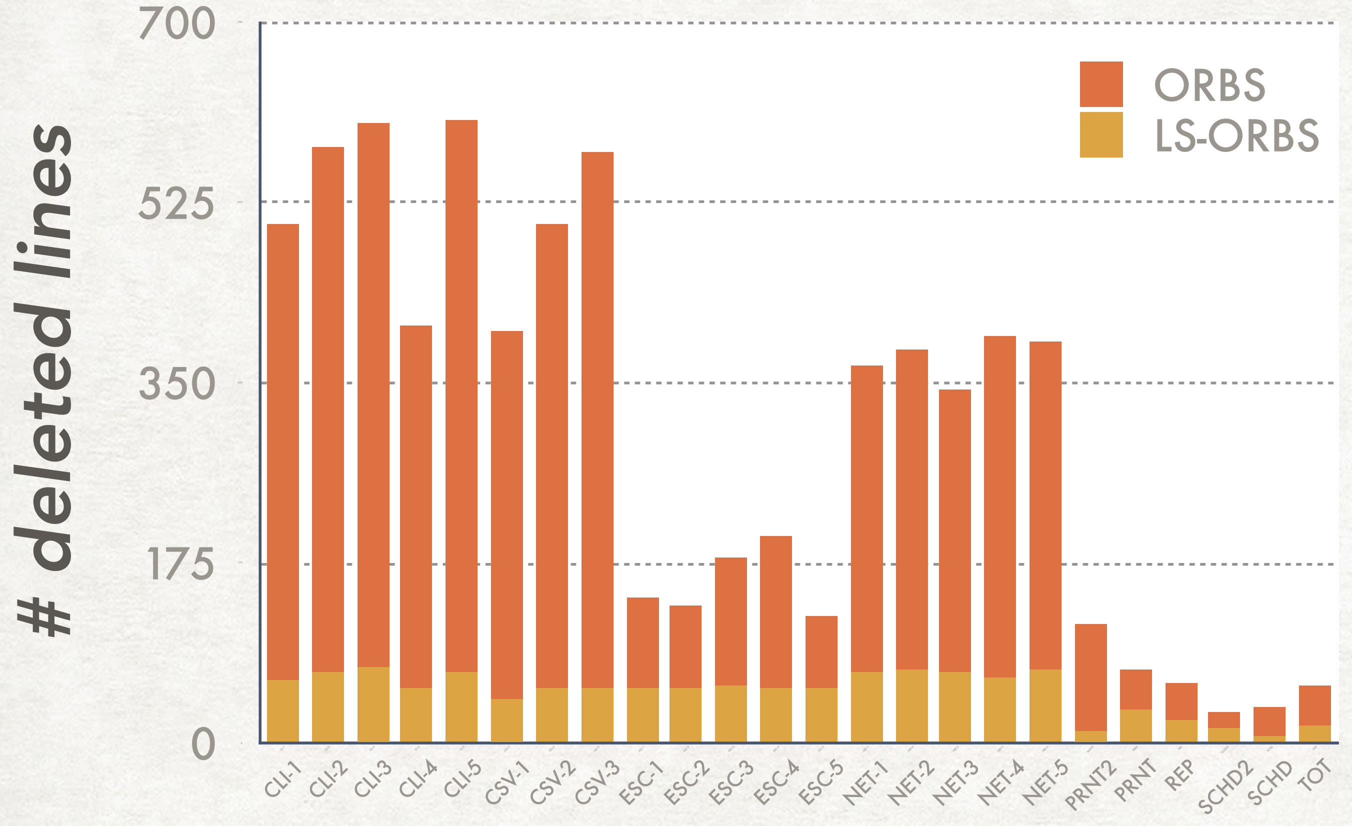
👍 **57%** time taken / del,

👎 **38%** # of deleted lines

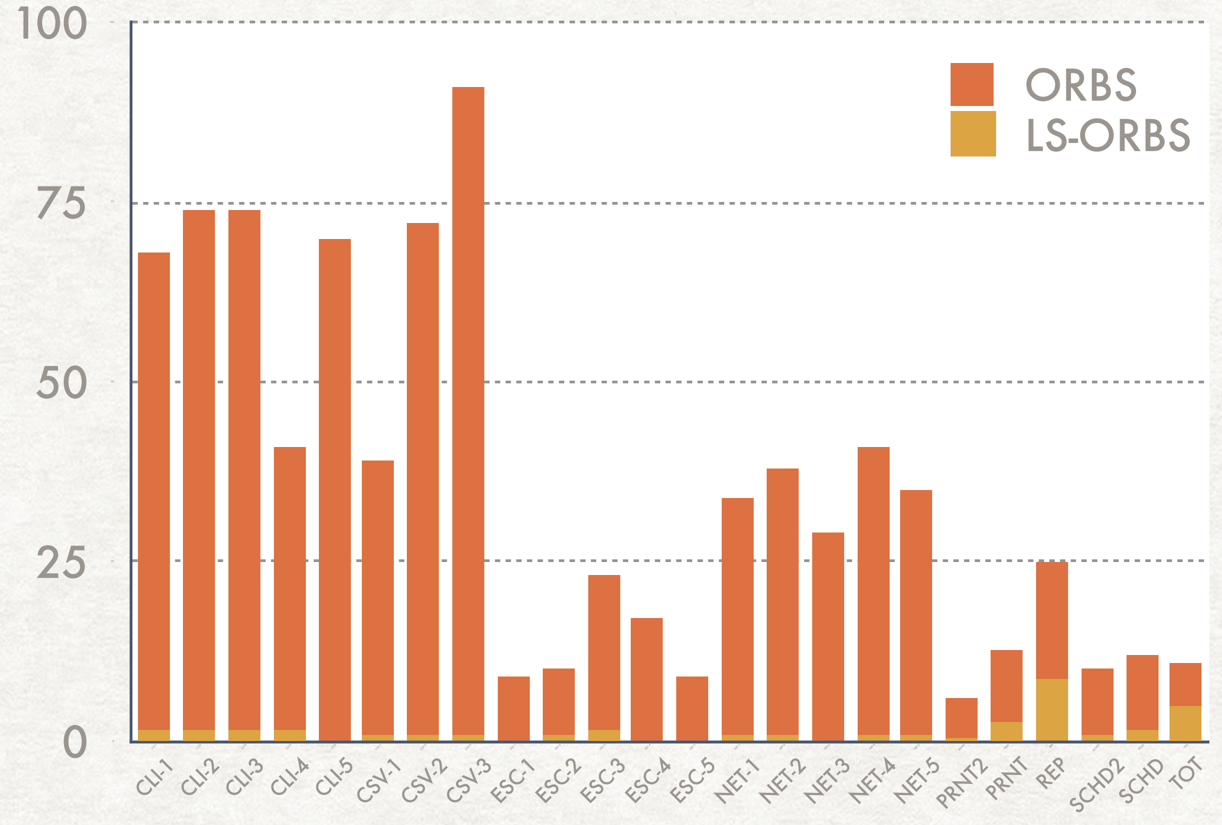
compared to ORBS

WHEN ARE LEXICAL DELETION OPERATORS EFFECTIVE / INEFFECTIVE?

Multi-line statements

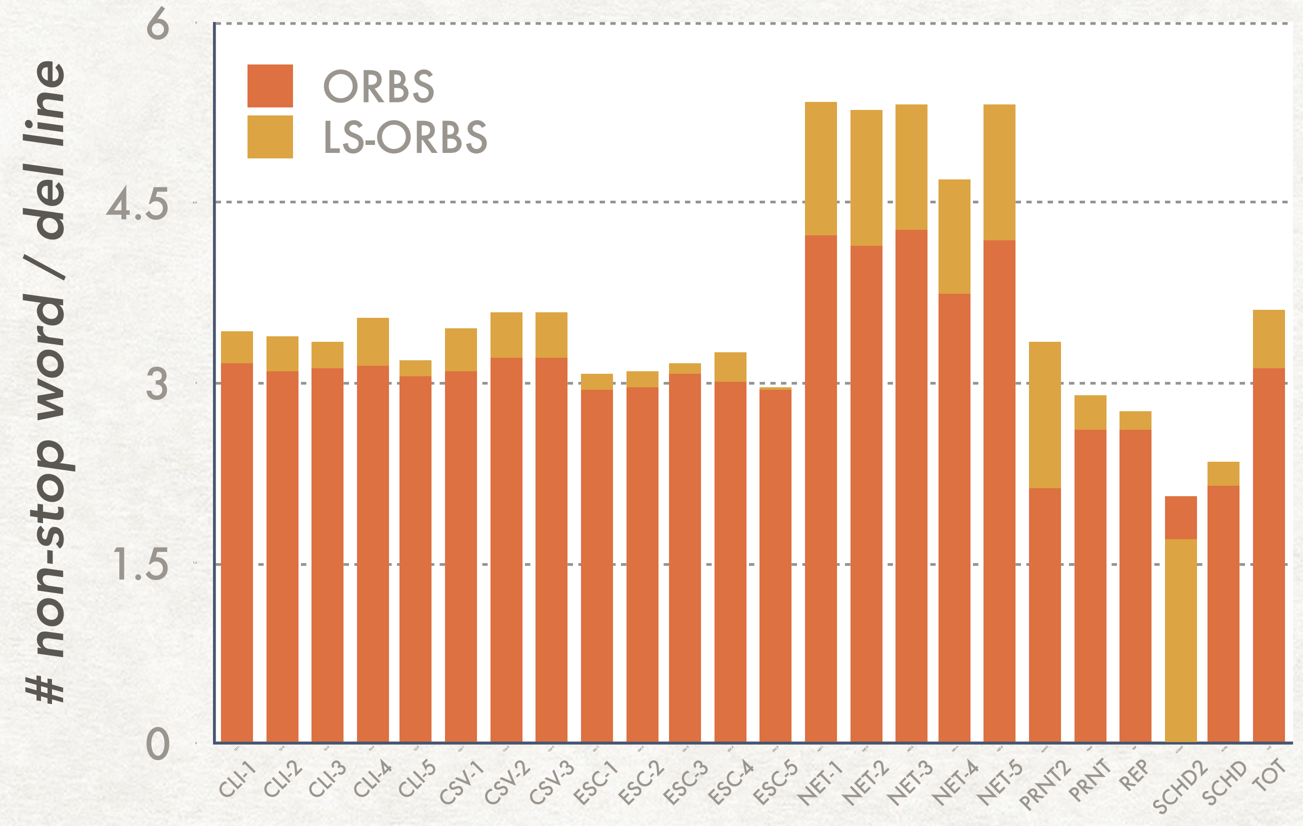


Declarations



Stop words: `if`, `while`, `class`, `def`, etc.

Non-stop words in deleted lines

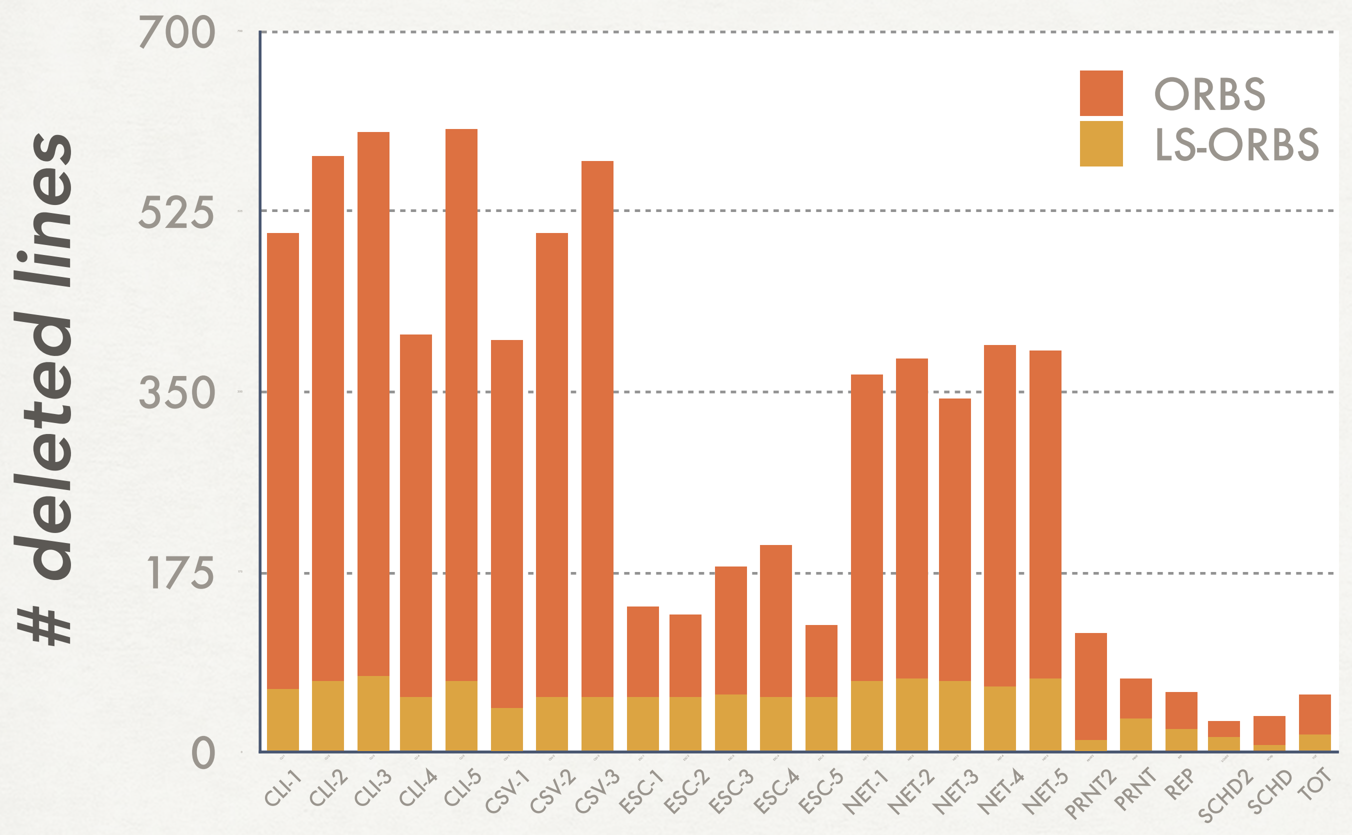


Syntactic structures in source code is challenging to the lexical deletion operators

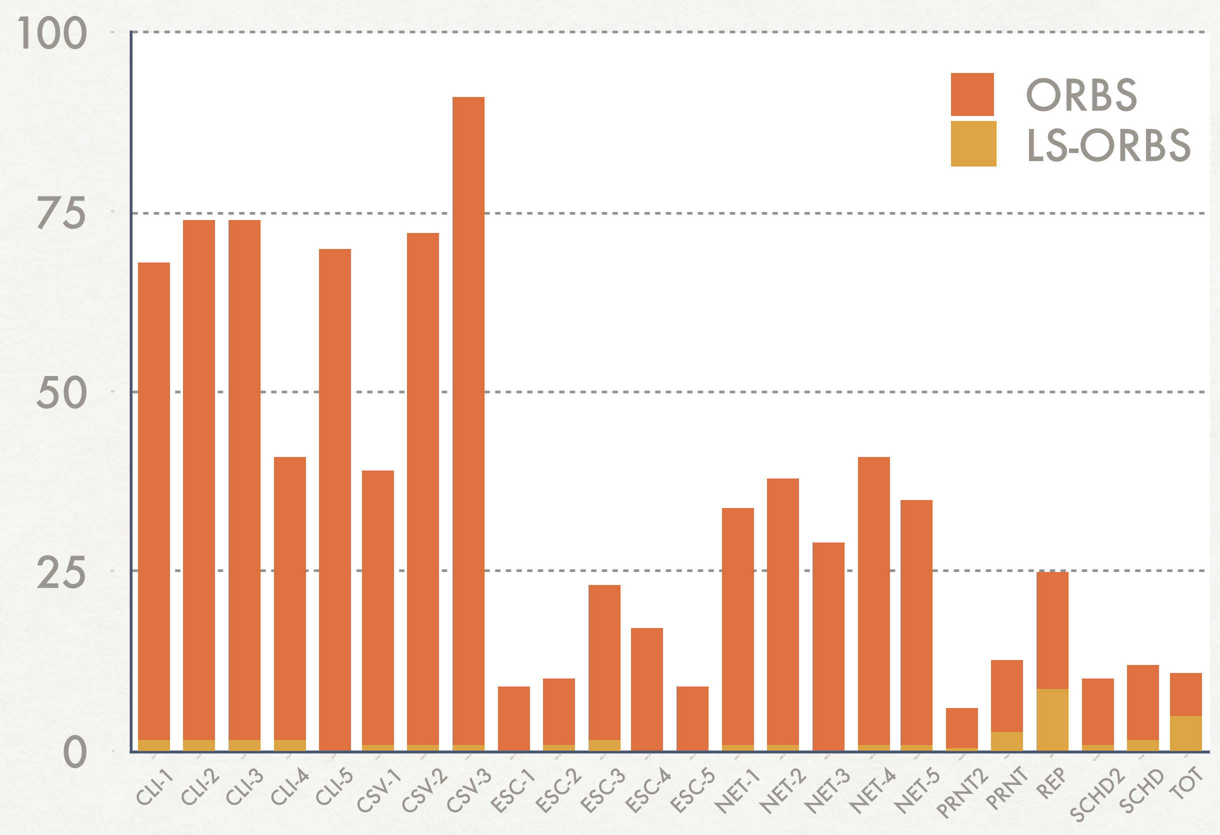
Lexical deletion operators are effective in statements with non-stop words.

WHEN ARE LEXICAL DELETION OPERATORS EFFECTIVE / INEFFECTIVE?

Multi-line statements



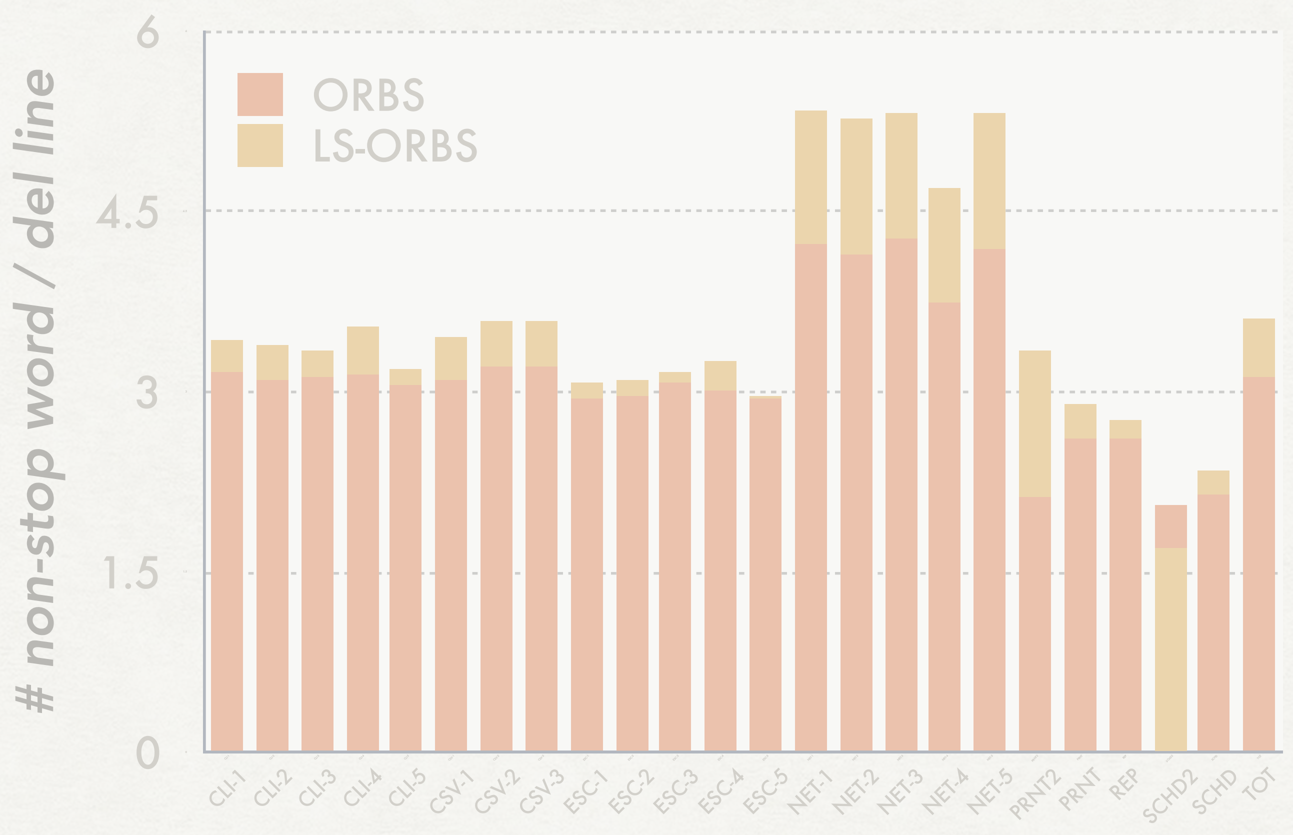
Declarations



Syntactic structures in source code is challenging to the lexical deletion operators

Stop words: `if`, `while`, `class`, `def`, etc.

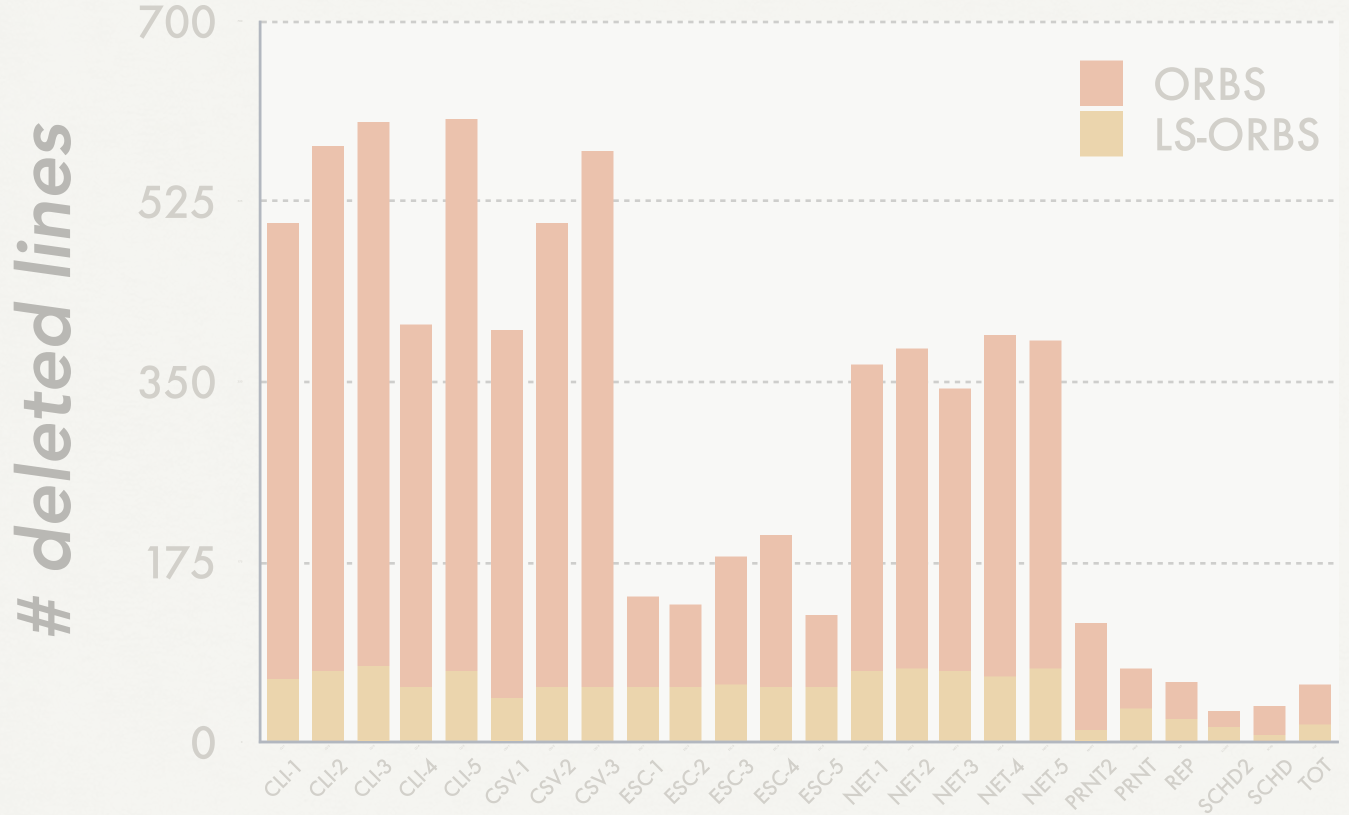
Non-stop words in deleted lines



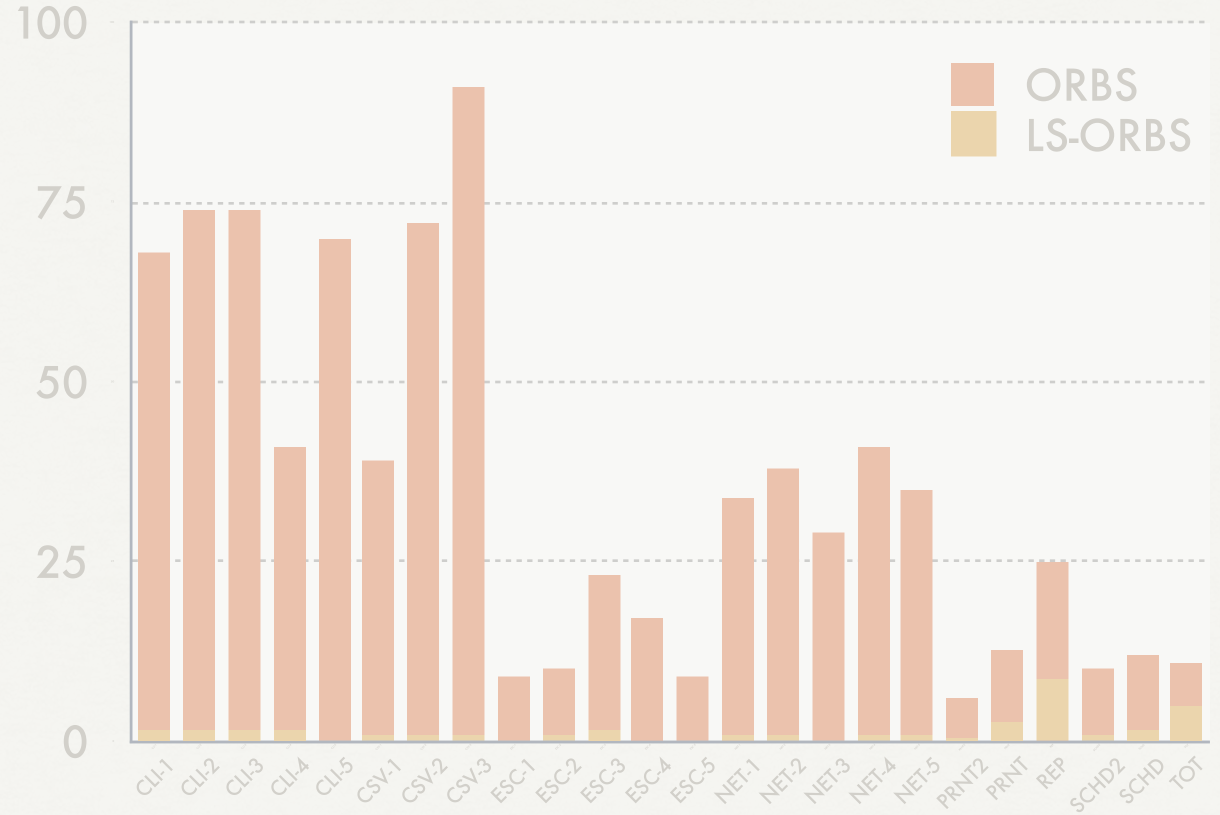
Lexical deletion operators are effective in statements with non-stop words.

WHEN ARE LEXICAL DELETION OPERATORS EFFECTIVE / INEFFECTIVE?

Multi-line statements



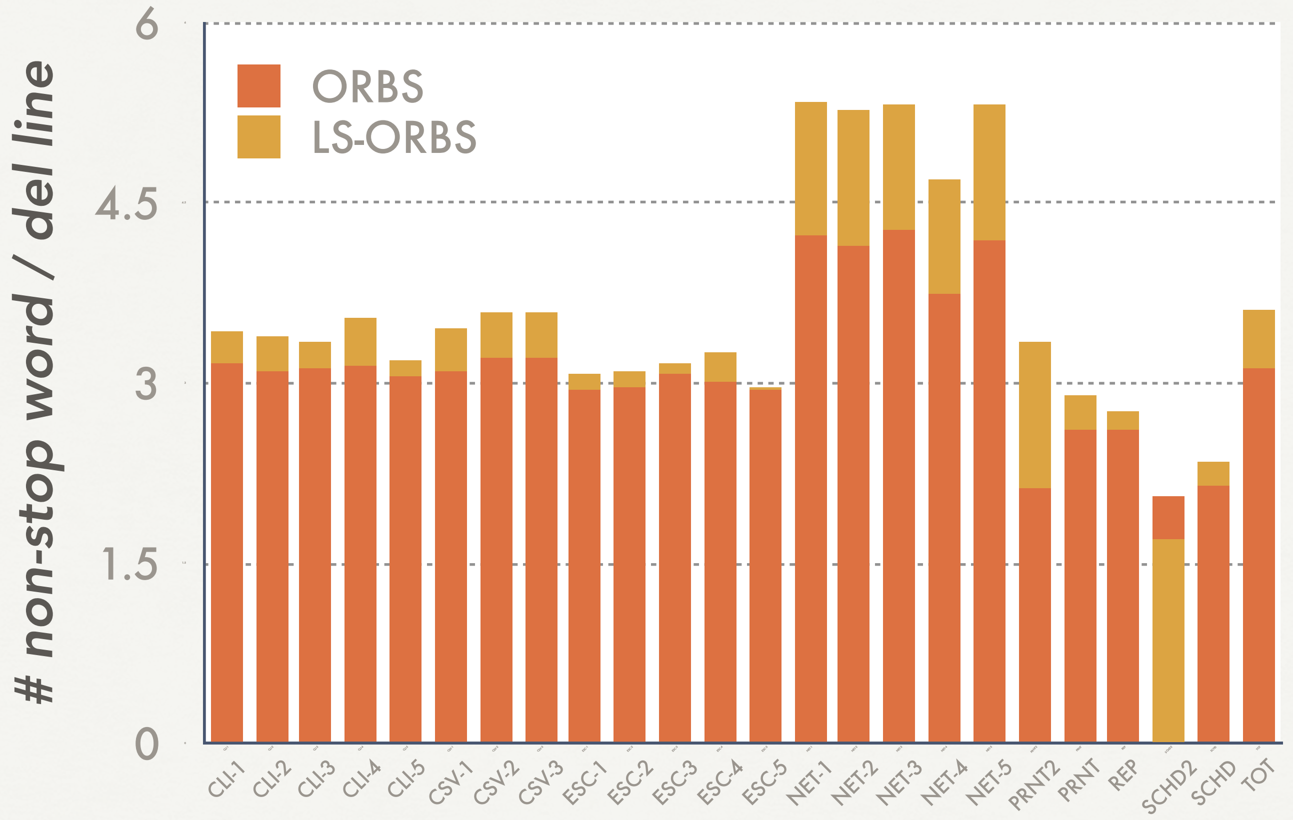
Declarations



Syntactic structures in source code is challenging to the lexical deletion operators

Stop words: `if`, `while`, `class`, `def`, etc.

Non-stop words in deleted lines

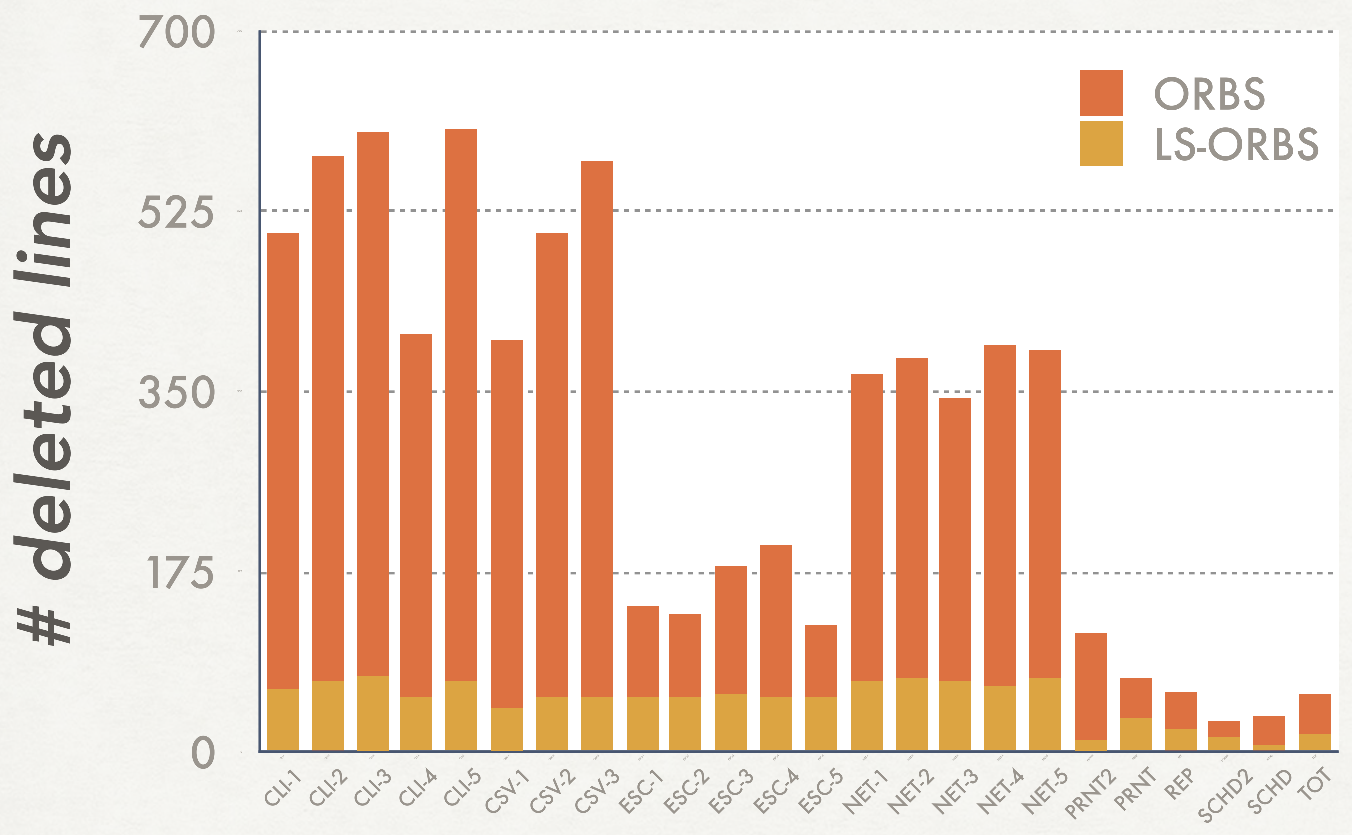


Lexical deletion operators are effective in statements with non-stop words.

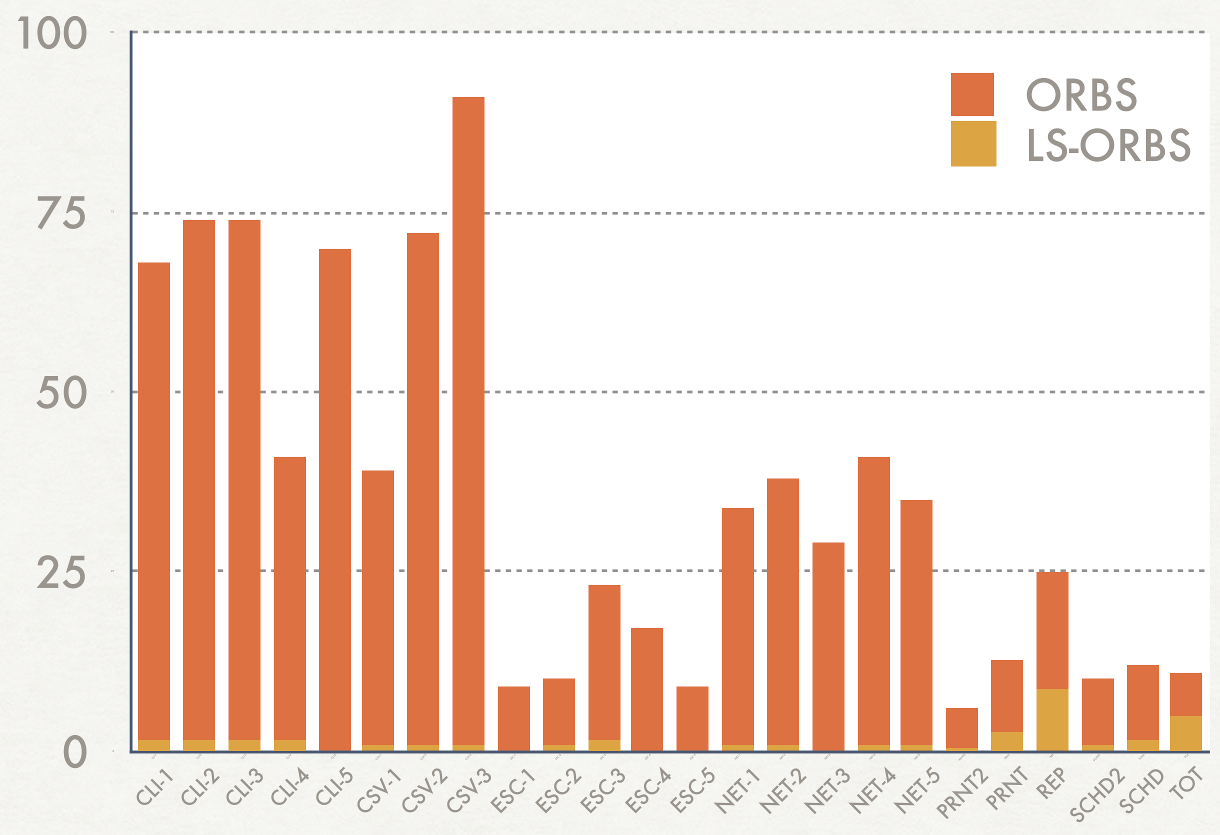
WHEN ARE LEXICAL DELETION OPERATORS EFFECTIVE / INEFFECTIVE?

Stop words: `if`, `while`, `class`, `def`, etc.

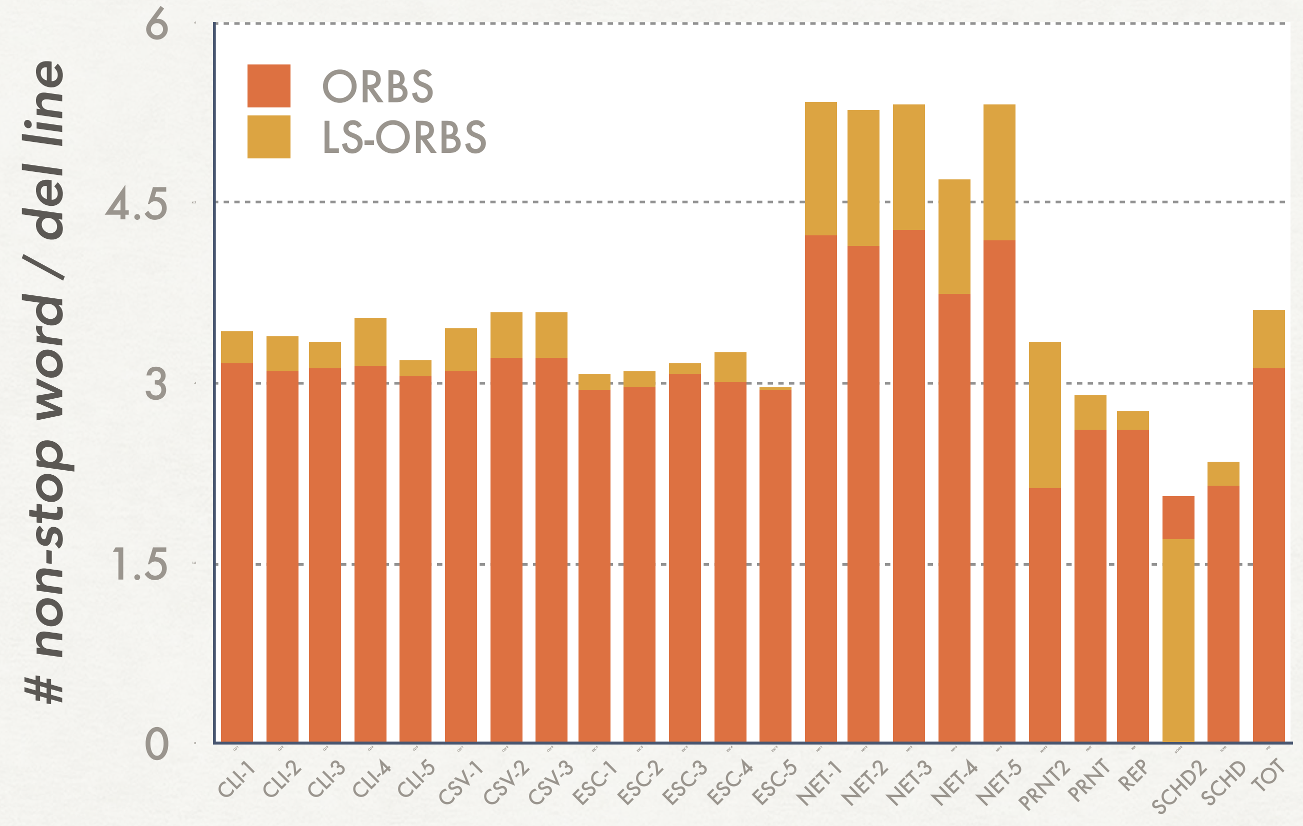
Multi-line statements



Declarations



Non-stop words in deleted lines



Syntactic structures in source code is challenging to the lexical deletion operators

Lexical deletion operators are effective in statements with non-stop words.

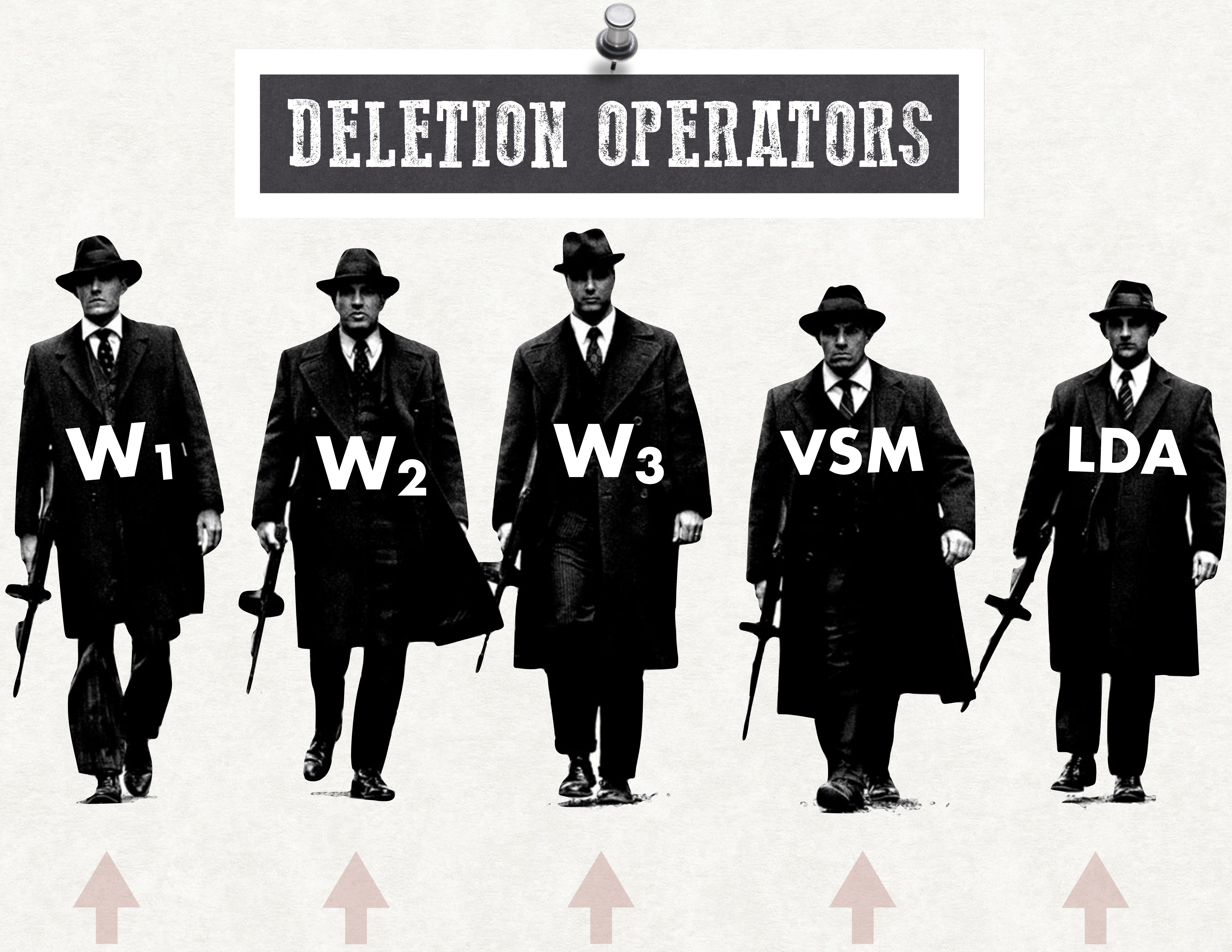
There is a complementary relation between window deletion and lexical deletion.

MOBS: MULTI-OPERATOR ORBS

```python
obs_matrix_dict = OrderedDict()
for obs_dir in obs_dir_list:
    itv_state_idx = get_itv_state_idx(work_dir, obs_dir)
    cmp_dict = get_cmp_dict(obs_dir)
    for testname, obs_dict in cmp_dict.items():
        obs_row = get_obs_row(itv_state_idx, obs_dict)
        if is_stdout:
            oracle_stdout_path = os.path.join(work_dir, "oracle", "test", testname)
            obs_stdout_path = os.path.join(obs_dir, "test", testname)
            obs_row = np.append(
                obs_row, 0 if filecmp.cmp(oracle_stdout_path, obs_stdout_path) else 1
            )
        # When the intervention has no effect, tell there was intervention.
        if itv_state_idx != 0:
            itv_matrix_idx = util.get_matrix_idx_from_state_idx(work_dir, itv_state_idx)
            if obs_row[itv_matrix_idx] == 0:
                if not np.array_equal(obs_row[1:], [0] * (len(obs_row) - 1)):
                    root_logger.debug(
                        f"obs_dir: {obs_dir}, testname: {testname}, itv_state_idx:
                        {itv_state_idx}, obs_row: {obs_row}"
                    )
                    root_logger.error(
                        "Assertion failed: obs_row[1:] != [0] * (len(obs_row) - 1"
                    )
                    root_logger.error(
                        f"obs_dir: {obs_dir}, itv_state_idx: {itv_state_idx}, itv_matrix_idx:
                        {itv_matrix_idx}, testname: {testname}"
                    )
                    root_logger.error(f"obs_row: {obs_row}")
                    raise Exception("Not intervened observation has different behavior.")
        if testname not in obs_matrix_dict:
            obs_matrix_dict[testname] = []
        obs_matrix_dict[testname].append(obs_row)
for testname in obs_matrix_dict.keys():
    obs_matrix = np.array(obs_matrix_dict[testname])

    itv_col = obs_matrix[:, 0]
    unique, counts = np.unique(itv_col, return_counts=True)
```

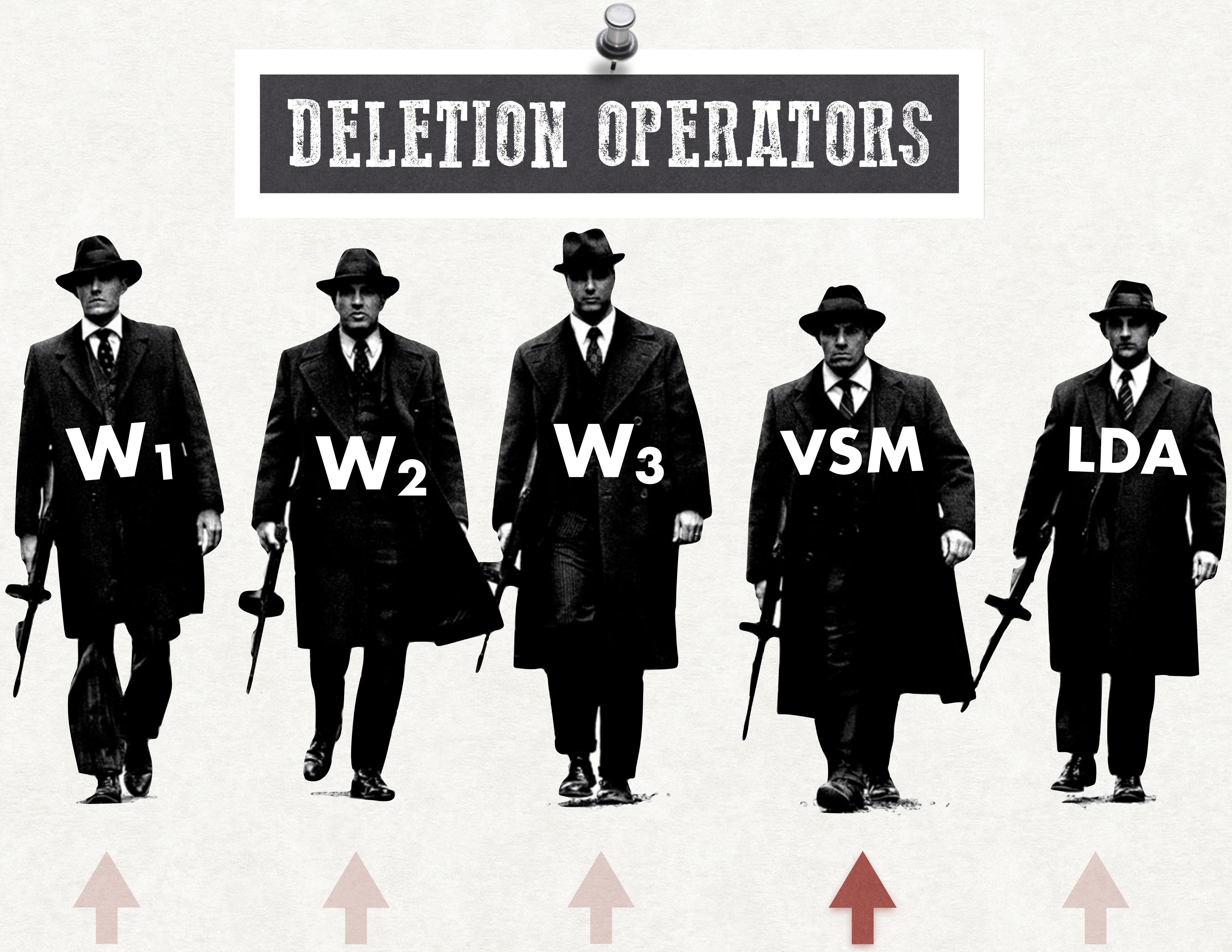
MOBS: MULTI-OPERATOR ORBS

```python
obs_matrix_dict = OrderedDict()
for obs_dir in obs_dir_list:
    itv_state_idx = get_itv_state_idx(work_dir, obs_dir)
    cmp_dict = get_cmp_dict(obs_dir)
    for testname, obs_dict in cmp_dict.items():
        obs_row = get_obs_row(itv_state_idx, obs_dict)
        if is_stdout:
            oracle_stdout_path = os.path.join(work_dir, "oracle", "test", testname)
            obs_stdout_path = os.path.join(obs_dir, "test", testname)
            obs_row = np.append(
                obs_row, 0 if filecmp.cmp(oracle_stdout_path, obs_stdout_path) else 1
            )
        # When the intervention has no effect, tell there was intervention.
        if itv_state_idx != 0:
            itv_matrix_idx = util.get_matrix_idx_from_state_idx(work_dir, itv_state_idx)
            if obs_row[itv_matrix_idx] == 0:
                if not np.array_equal(obs_row[1:], [0] * (len(obs_row) - 1)):
                    root_logger.debug(
                        f"obs_dir: {obs_dir}, testname: {testname}, itv_state_idx:
                        {itv_state_idx}, obs_row: {obs_row}"
                    )
                    root_logger.error(
                        "Assertion failed: obs_row[1:] != [0] * (len(obs_row) - 1"
                    )
                    root_logger.error(
                        f"obs_dir: {obs_dir}, itv_state_idx: {itv_state_idx}, itv_matrix_idx:
                        {itv_matrix_idx}, testname: {testname}"
                    )
                    root_logger.error(f"obs_row: {obs_row}")
                    raise Exception("Not intervened observation has different behavior.")
        if testname not in obs_matrix_dict:
            obs_matrix_dict[testname] = []
        obs_matrix_dict[testname].append(obs_row)
for testname in obs_matrix_dict.keys():
    obs_matrix = np.array(obs_matrix_dict[testname])

    itv_col = obs_matrix[:, 0]
    unique, counts = np.unique(itv_col, return_counts=True)
```



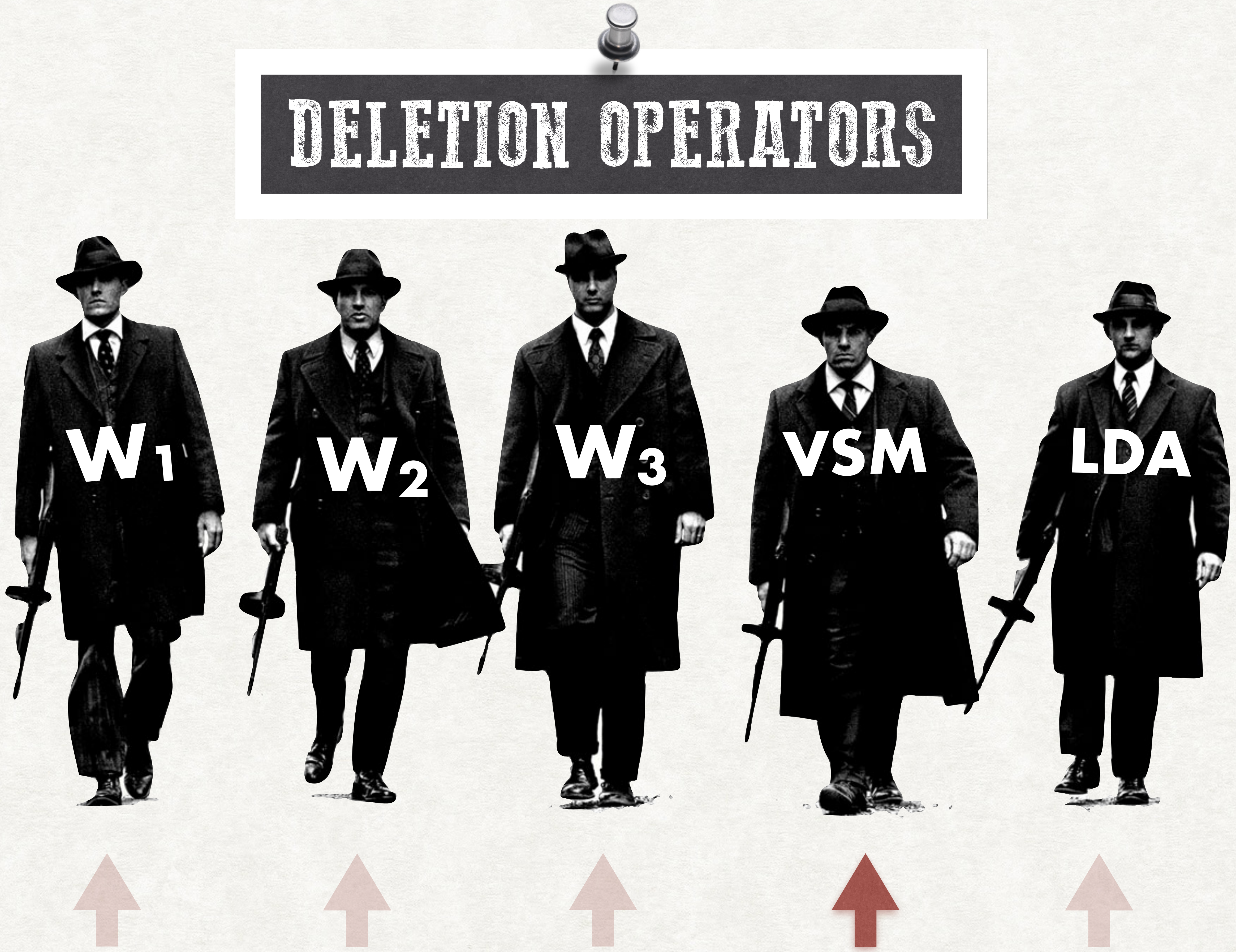
MOBS: MULTI-OPERATOR ORBS

```python
obs_matrix_dict = OrderedDict()
for obs_dir in obs_dir_list:
    itv_state_idx = get_itv_state_idx(work_dir, obs_dir)
    cmp_dict = get_cmp_dict(obs_dir)
    for testname, obs_dict in cmp_dict.items():
        obs_row = get_obs_row(itv_state_idx, obs_dict)
        if is_stdout:
            oracle_stdout_path = os.path.join(work_dir, "oracle", "test", testname)
            obs_stdout_path = os.path.join(obs_dir, "test", testname)
            obs_row = np.append(
                obs_row, 0 if filecmp.cmp(oracle_stdout_path, obs_stdout_path) else 1
            )
        # When the intervention has no effect, tell there was intervention.
        if itv_state_idx != 0:
            itv_matrix_idx = util.get_matrix_idx_from_state_idx(work_dir, itv_state_idx)
            if obs_row[itv_matrix_idx] == 0:
                if not np.array_equal(obs_row[1:], [0] * (len(obs_row) - 1)):
                    root_logger.debug(
                        f"obs_dir: {obs_dir}, testname: {testname}, itv_state_idx:
                        {itv_state_idx}, obs_row: {obs_row}"
                    )
                    root_logger.error(
                        "Assertion failed: obs_row[1:] != [0] * (len(obs_row) - 1"
                    )
                    root_logger.error(
                        f"obs_dir: {obs_dir}, itv_state_idx: {itv_state_idx}, itv_matrix_idx:
                        {itv_matrix_idx}, testname: {testname}"
                    )
                    root_logger.error(f"obs_row: {obs_row}")
                    raise Exception("Not intervened observation has different behavior.")
        if testname not in obs_matrix_dict:
            obs_matrix_dict[testname] = []
        obs_matrix_dict[testname].append(obs_row)
for testname in obs_matrix_dict.keys():
    obs_matrix = np.array(obs_matrix_dict[testname])

    itv_col = obs_matrix[:, 0]
    unique, counts = np.unique(itv_col, return_counts=True)
```



MOBS: MULTI-OPERATOR ORBS

```python
obs_matrix_dict = OrderedDict()
for obs_dir in obs_dir_list:
    itv_state_idx = get_itv_state_idx(work_dir, obs_dir)
    cmp_dict = get_cmp_dict(obs_dir)
    for testname, obs_dict in cmp_dict.items():
        obs_row = get_obs_row(itv_state_idx, obs_dict)
        if is_stdout:
            oracle_stdout_path = os.path.join(work_dir, "oracle", "test", testname)
            obs_stdout_path s.path.join(obs_dir, "test", testname)
            obs_row =
                obs_r cmp.cmp(oracle_stdout_path, obs_stdout_path) else 1
            )
        # When the int s no effect, tell there was intervention.
        if itv_state_idx != 0:
            itv_matrix_idx = util.get_matrix_idx_from_state_idx(work_dir, itv_state_idx)
            if obs_row[itv_matrix_idx] == 0:
                if not np.array_equal(obs_row[1:], [0] row) - 1)):
                    root_logger.debug(
                        f"obs_dir: {obs_dir}, testname: itv_state_idx:
                        {itv_state_idx}, obs_row: {obs_
                    )
                    root_logger.error(
                        "Assertion failed: obs_row[1:] != [0] * (len(obs_row) - 1"
                    )
                    root_logger.erro
                        f"obs_dir }, itv_state_idx: {itv_state_idx}, itv_matrix_idx:
                        {itv_mat stname: {testname}"
                    )
                    root_logger. row: {obs_row}")
                    raise Exception("Not intervened observation has different behavior.")
        if testname not in obs_matrix_dict:
            obs_matrix_dict[testname] = []
        obs_matrix_dict[testname].append(obs_row)
for testname in obs_matrix_dict.keys():
    obs_matrix = np.array(obs_matrix_dict[testname])

    itv_col = obs_matrix[:, 0]
    unique, counts = np.unique(itv_col, return_counts=True)
```



MOBS: MULTI-OPERATOR ORBS

OPERATOR SELECTION USING PROBABILITY DISTRIBUTION

MOBS: MULTI-OPERATOR ORBS

OPERATOR SELECTION USING PROBABILITY DISTRIBUTION

UNIFORM

MOBS: MULTI-OPERATOR ORBS

OPERATOR SELECTION USING PROBABILITY DISTRIBUTION

UNIFORM

FIXED
(APPLICABILITY / EFFECT)

MOBS: MULTI-OPERATOR ORBS

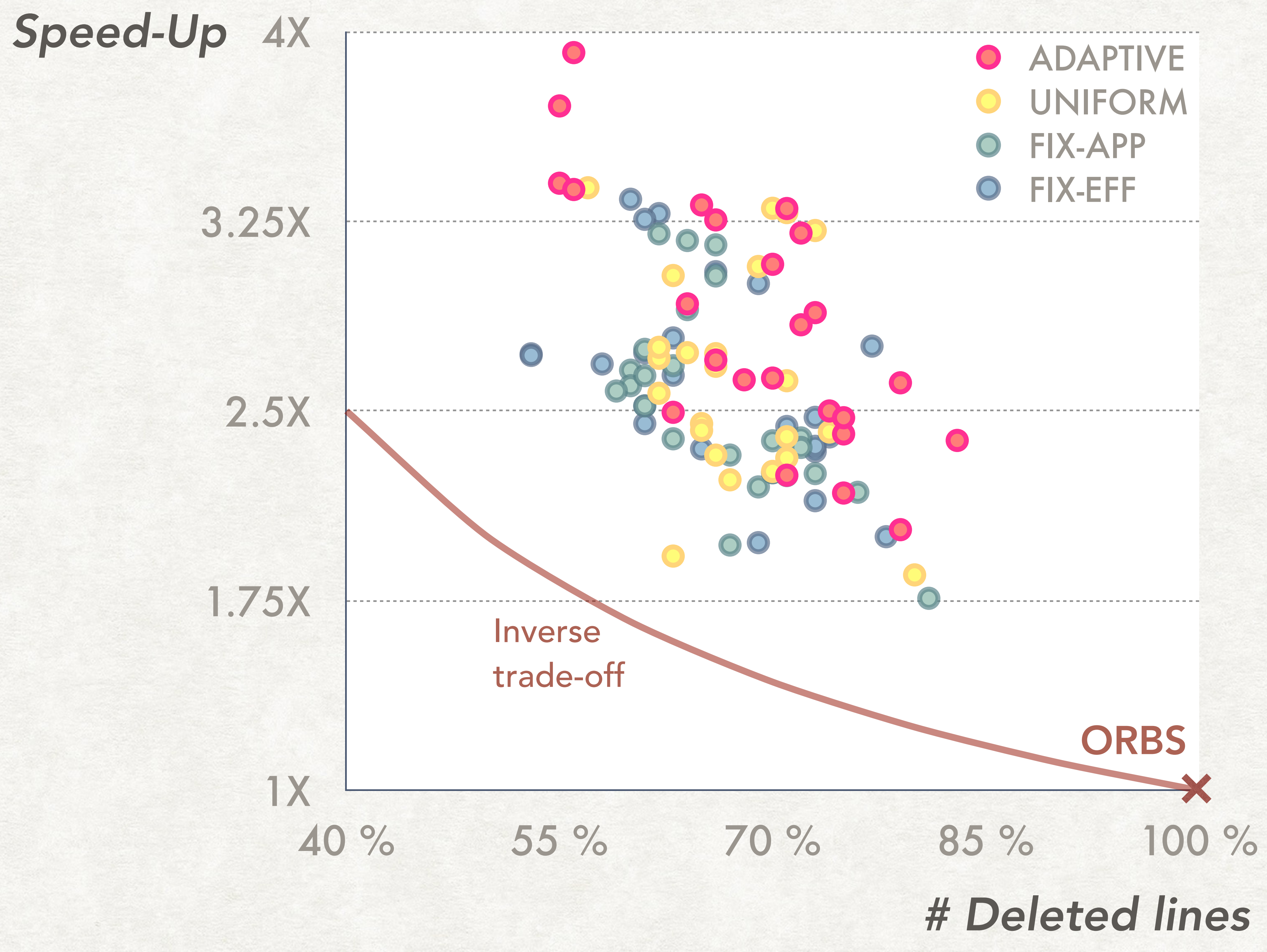
OPERATOR SELECTION USING PROBABILITY DISTRIBUTION

UNIFORM

FIXED
(APPLICABILITY / EFFECT)

ADAPTIVE

MOBS RESULT



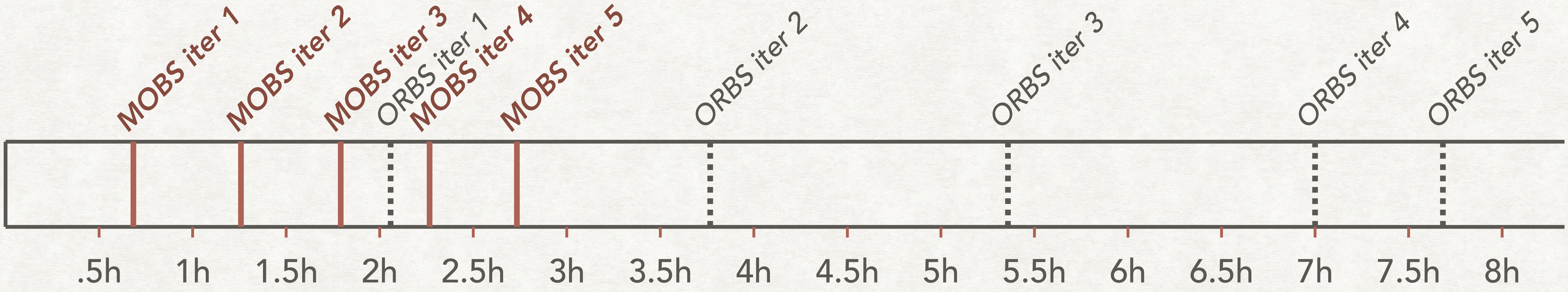
MOBS runs / achieves

- **2.9X** faster,
- **69%** # of deleted lines

compared to ORBS.

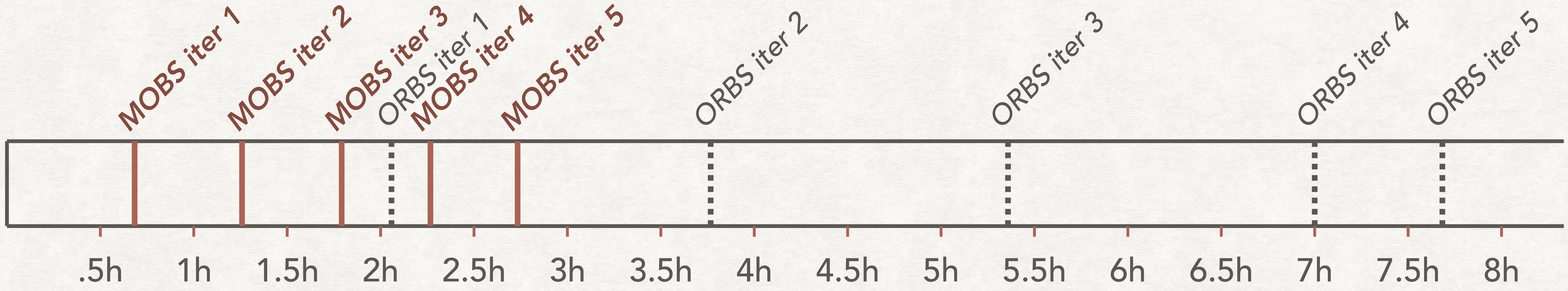
MOBS RESULT

- For apache commons CSV,



MOBS RESULT

- For apache commons CSV,



EXAMPLE. MULTI-LINGUAL DELETION

- Misaka (http://misaka.61924.nl)
  - A Python binding for Hoedown, a markdown parsing C library.
  - Programming language: C, Python

	NCLOC	FILES	TC
C	4,360	10	
Python	473	5	
Total	4,833	15	92

- VSM Deletion operator

```
┌ callbacks.py              (97)   > elif align_bit == TABLE_ALIGN_LEFT:
| callbacks.py              (98)   >     align = ‘left'
└ hoedown/html.c            (393)  > case HOEDOWN_TABLE_ALIGN_LEFT:
```

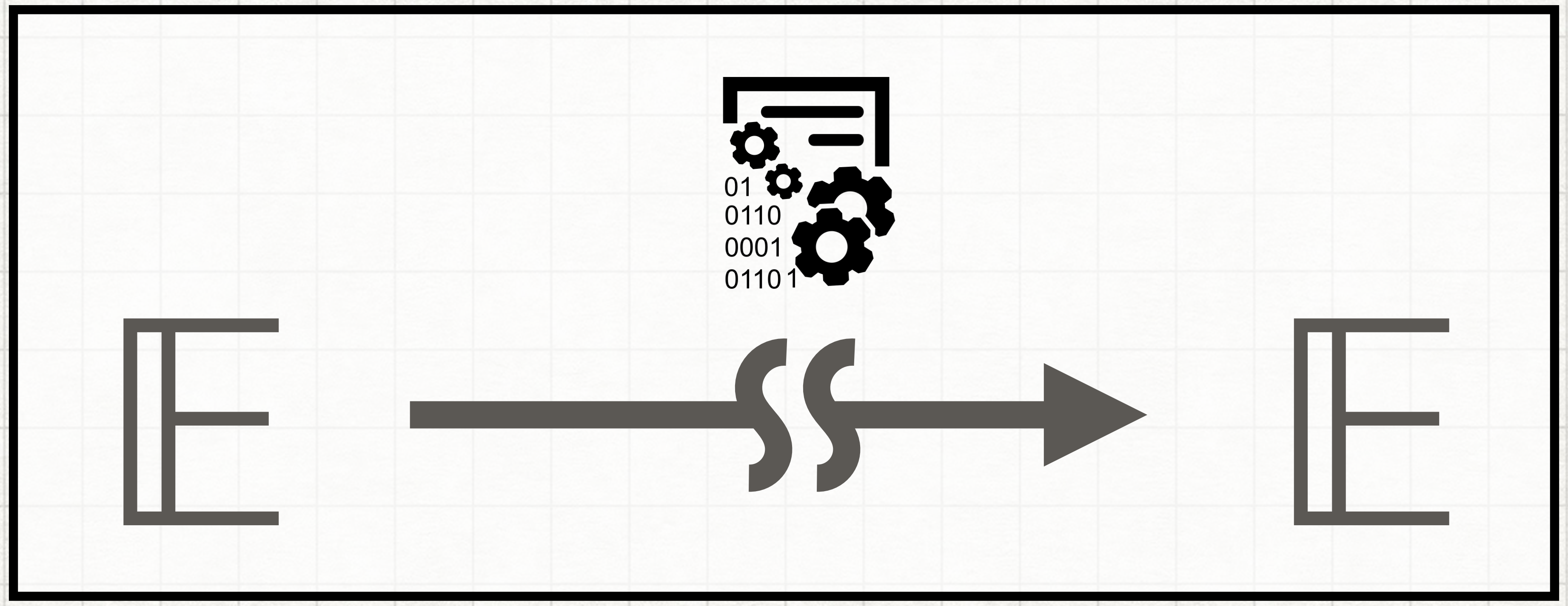
- LDA Deletion operator

```
┌ api.py                    (29)   > lib.hoedown_buffer_puts(ib, text.encode('utf-8'))
| hoedown/document.c        (2490) > hoedown_buffer_free(text);
└ hoedown/html_smartypants.c (195) > hoedown_buffer_putc(ob, text[0]);
```

- Both LDA and VSM Deletion operator

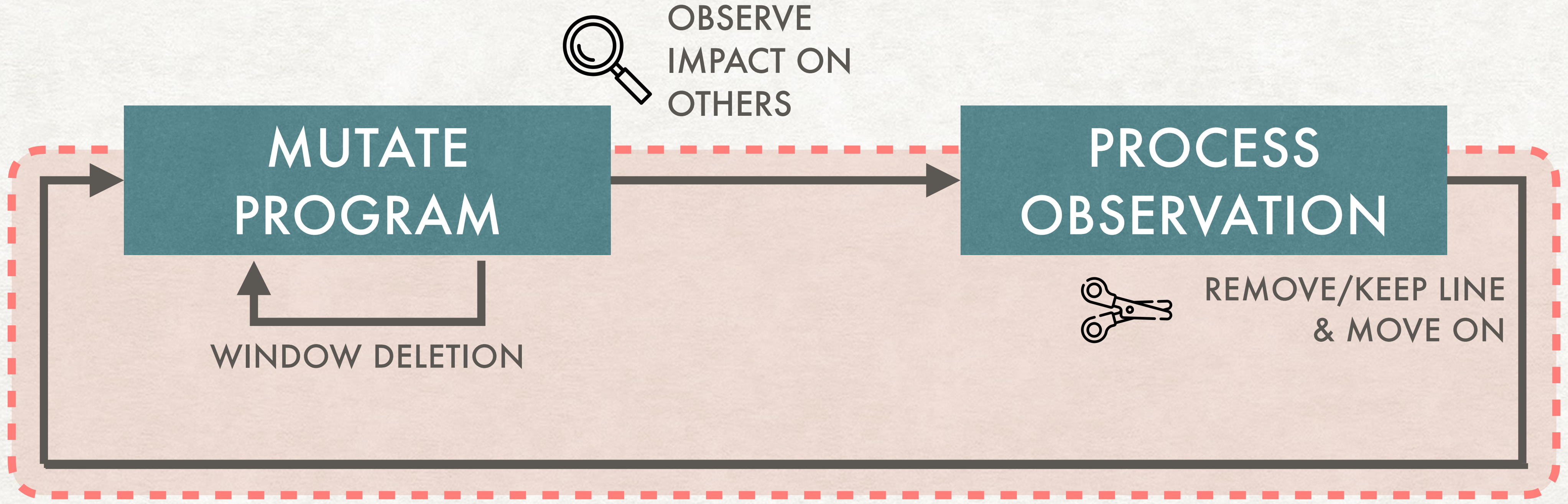
```
┌ callbacks.py              (125)  > result = renderer.blockhtml(text)
└ hoedown/html.c            (635)  > renderer->blockhtml = NULL;
```

MOAD



[*Approximating the general program dependence model by applying statistical models on the observation data*]

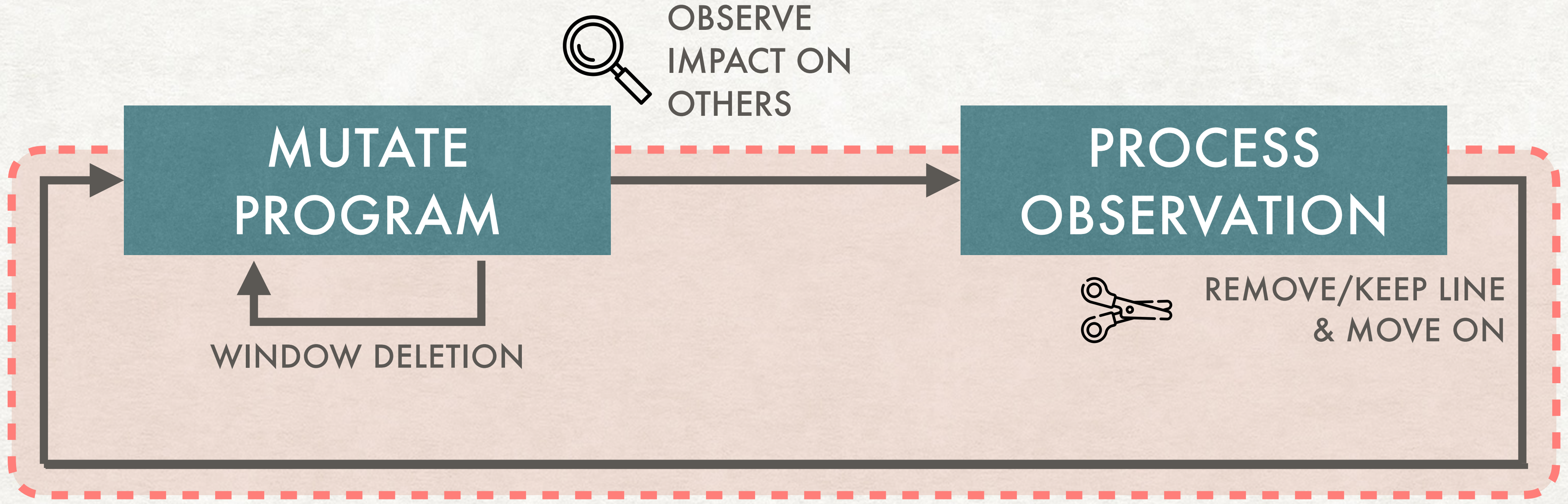
LIMITATION OF ORBS



Recursive / cumulative mutation

Requires a large number of compilations & executions

LIMITATION OF ORBS



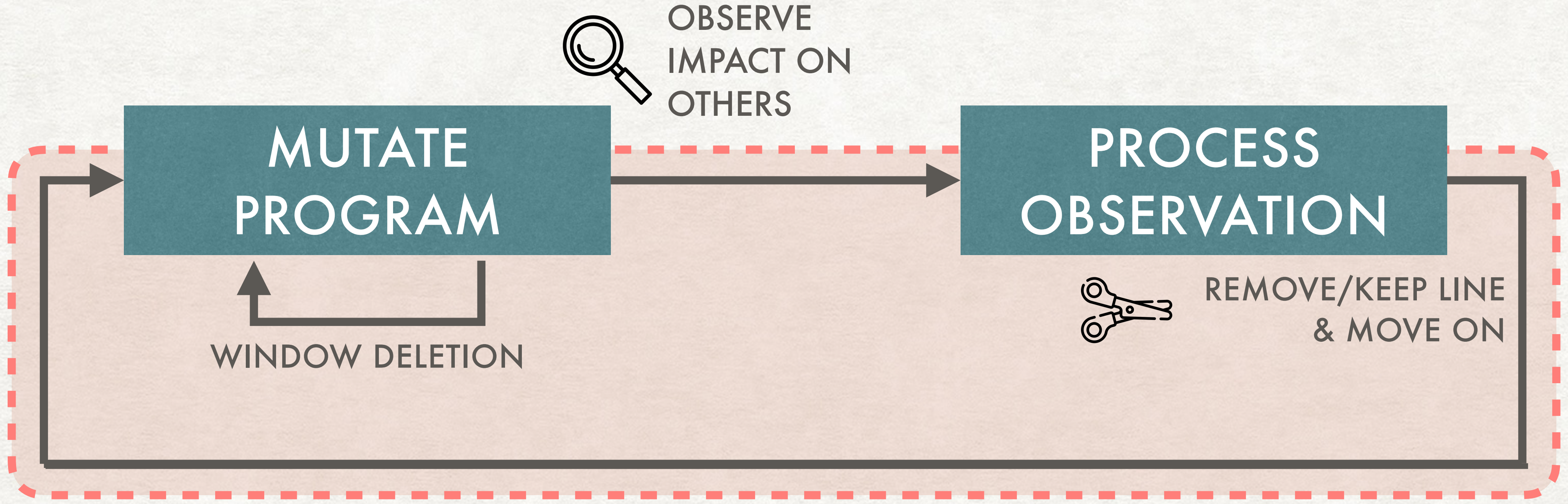
Recursive / cumulative mutation

ALL PROGRAM ELEMENT

$$\mathbb{E} \ni \{e_1, e_2\} \longrightarrow e_{crit.}$$

Requires a large number of compilations & executions

LIMITATION OF ORBS



Recursive / cumulative mutation

ALL PROGRAM ELEMENT

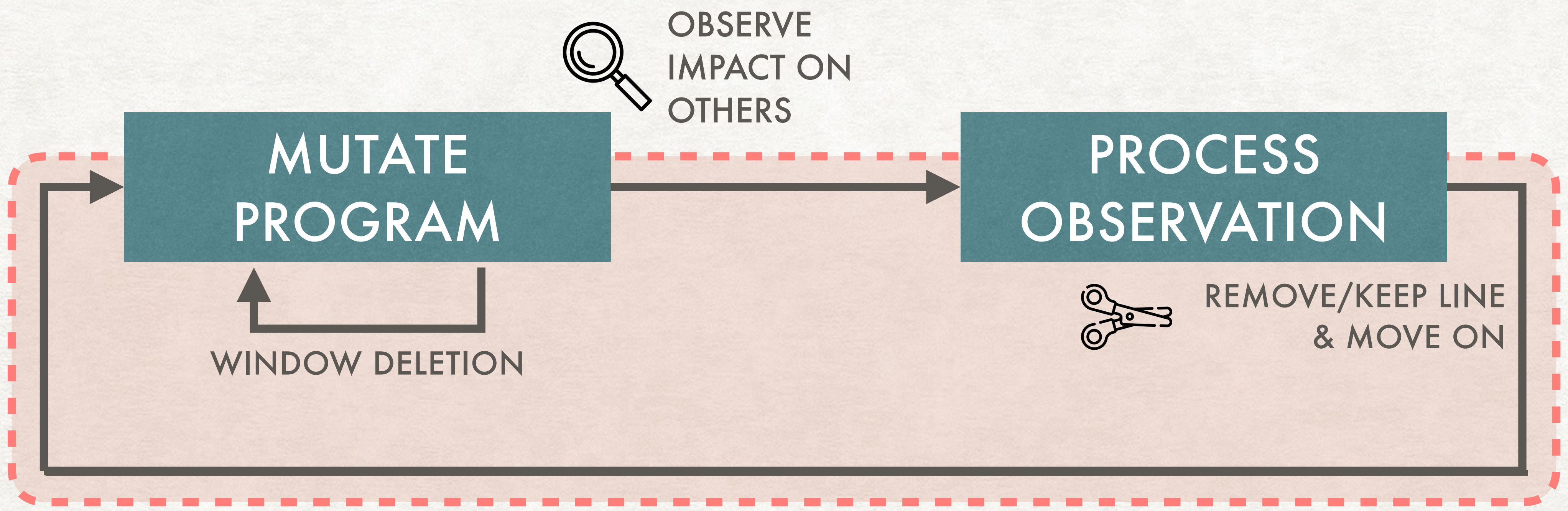
$$\mathbb{E} \ni \{e_1, e_2\} \longrightarrow e_{crit.}$$

$$\mathbb{E} \xrightarrow{?} e_4$$

Another element

Requires a large number of compilations & executions

LIMITATION OF ORBS



Recursive / cumulative mutation

ALL PROGRAM ELEMENT

$$\mathbb{E} \ni \{e_1, e_2\} \longrightarrow e_{crit.}$$

$$\mathbb{E} \xrightarrow{?} e_4$$

Another element

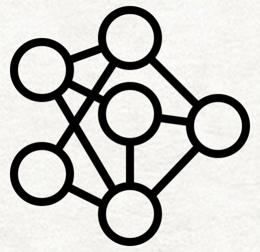
$$e_{crit.} \xrightarrow{?} \mathbb{E}$$

Forward dependency

Requires a large number of compilations & executions

Provides partial dependency information

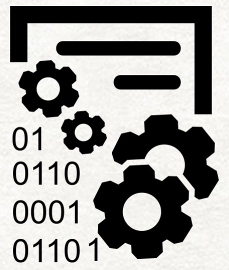
STATIC ANALYSIS



General dependence

$\mathbb{E} \longrightarrow \mathbb{E}$

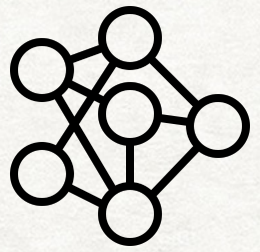
ORBS



Observation-based approach

$\mathbb{E} \xrightarrow{\times} e_c$

STATIC ANALYSIS



General dependence

$$\mathbb{E} \longrightarrow \mathbb{E}$$

ORBS

Observation-based approach

$$\mathbb{E} \longrightarrow e_c$$

Observation-based analysis

STATIC ANALYSIS

General dependence

$\mathbb{E} \longrightarrow \mathbb{E}$

ORBS

Observation-based approach

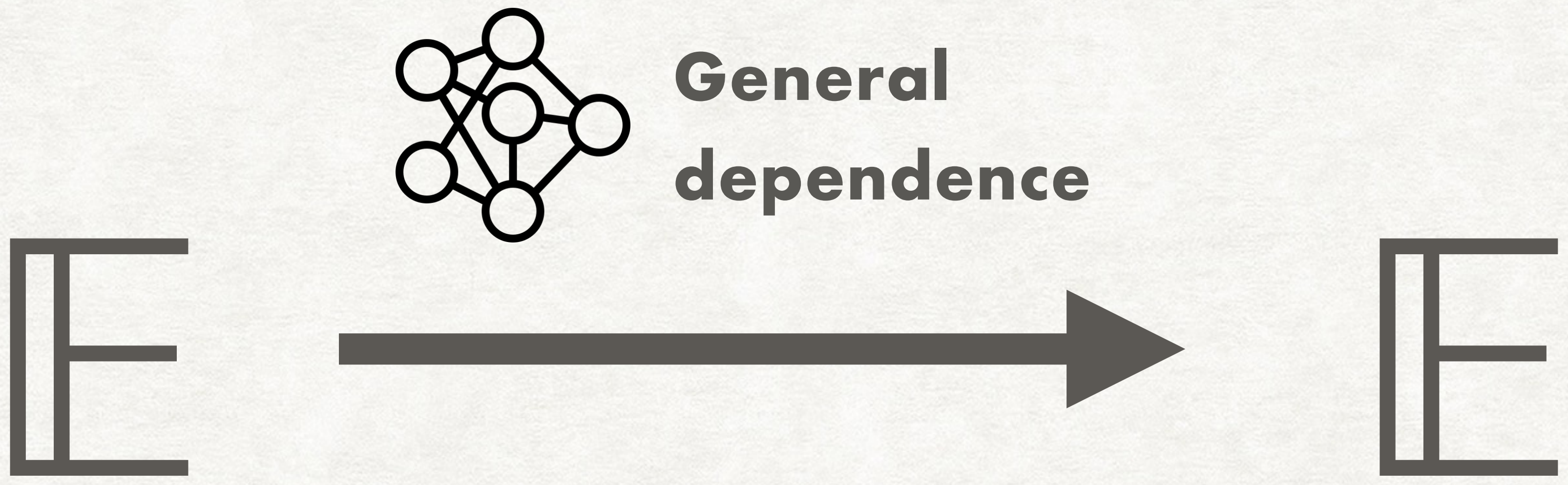
$\mathbb{E} \longrightarrow e_c$

Observation-based analysis

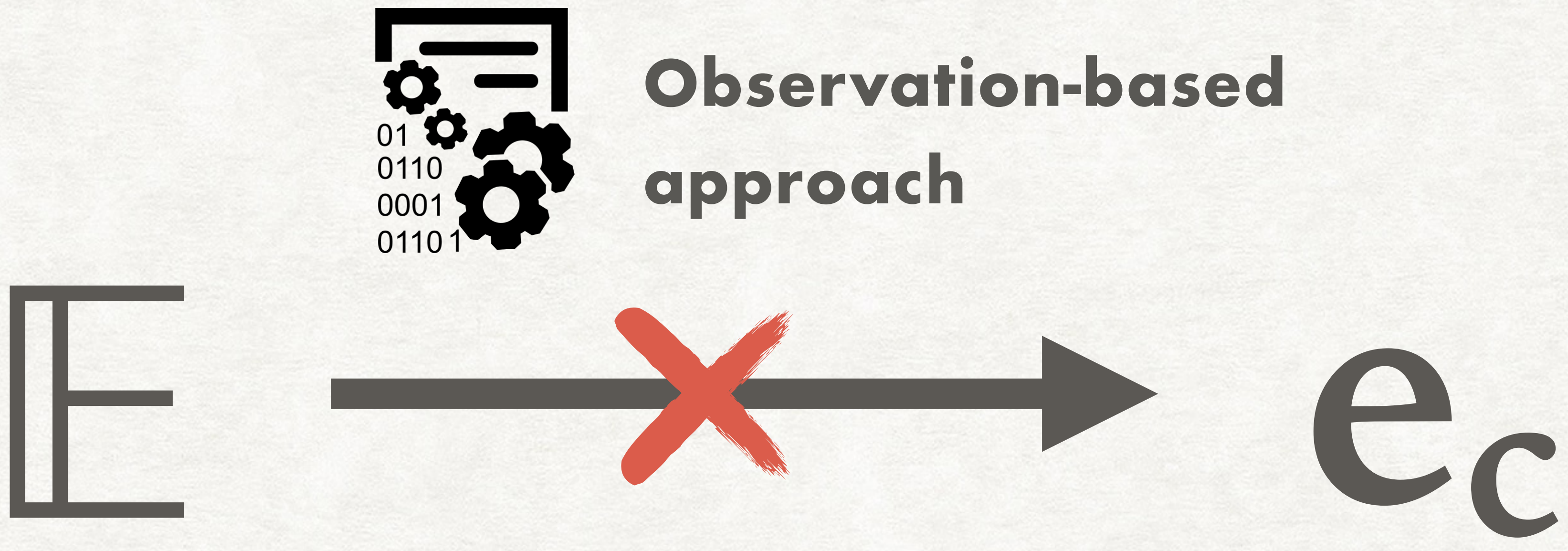
$\mathbb{E} \longrightarrow \mathbb{E}$

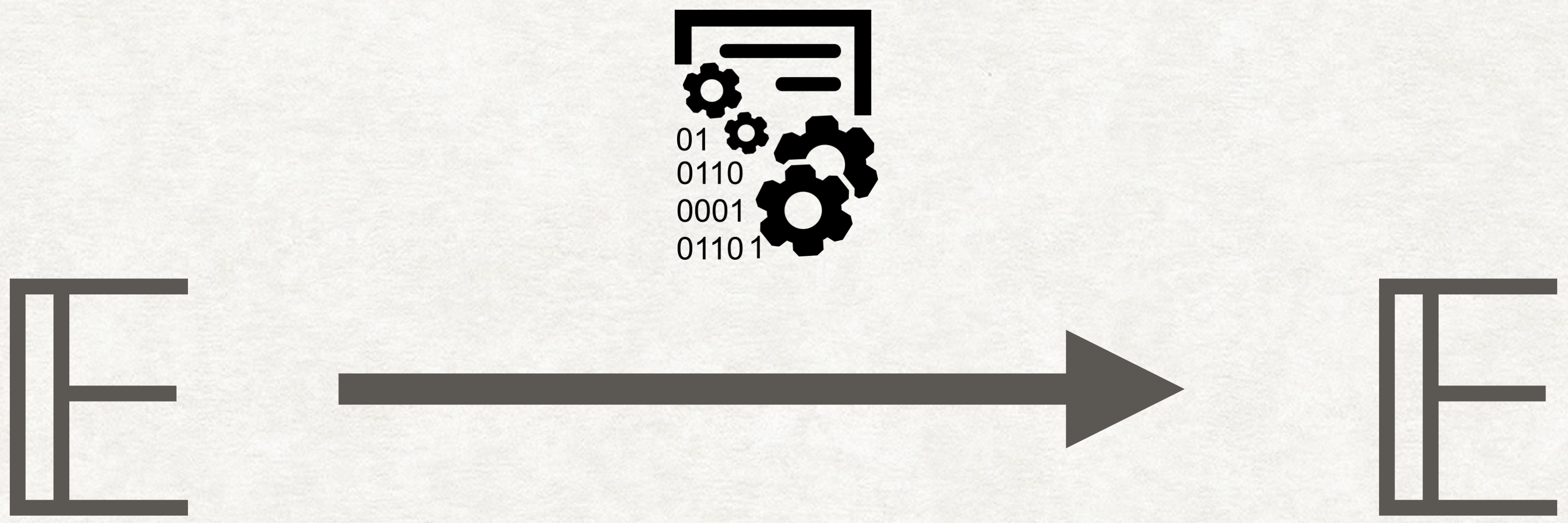
Modeling dependency

STATIC ANALYSIS



ORBS





MODELING
OBSERVATION-BASED
APPROXIMATE
DEPENDENCY

MOAD

Delete! $e_1 \longrightarrow e_c$

MOAD

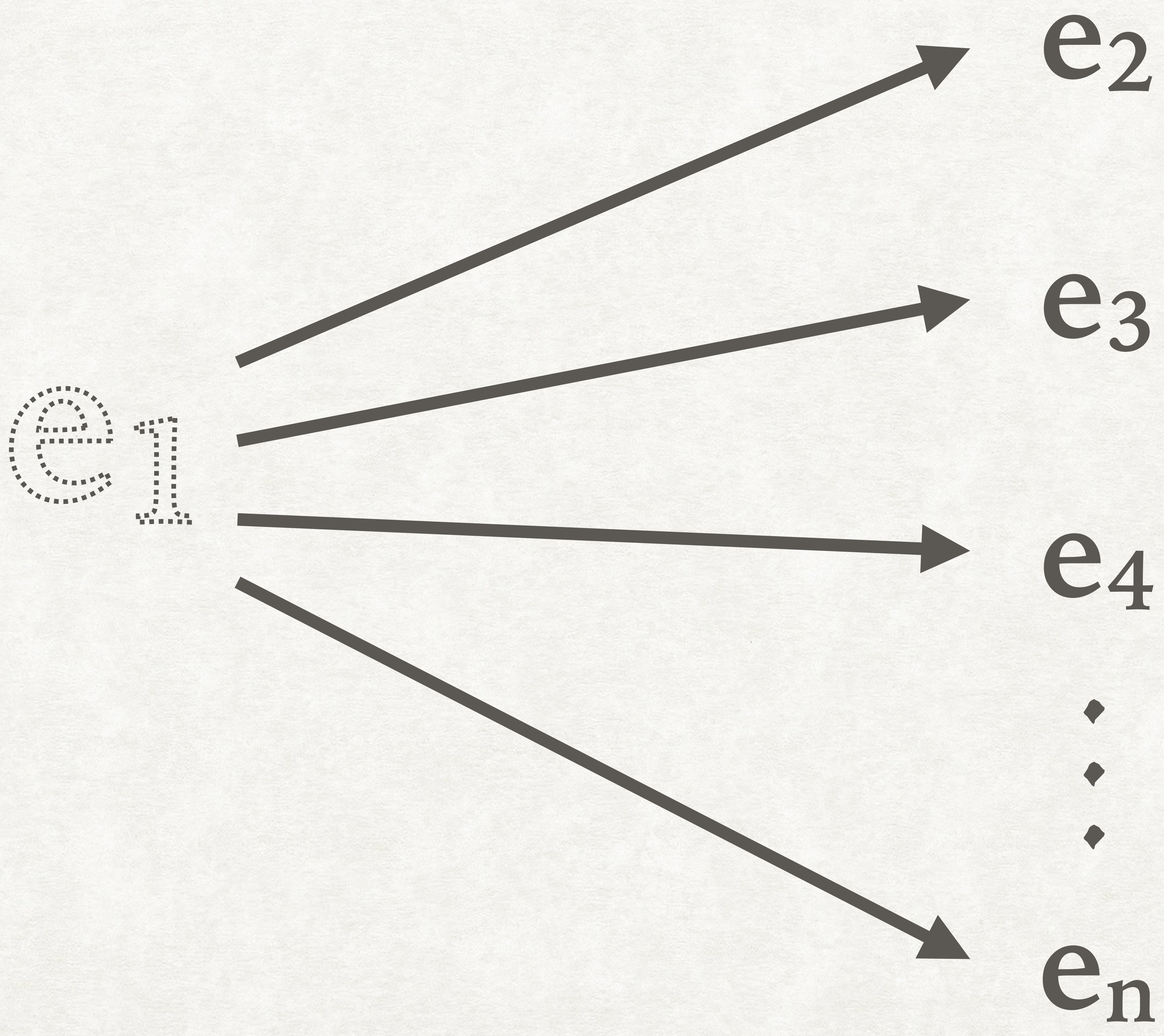
Trajectory preserved

Compile & Execute

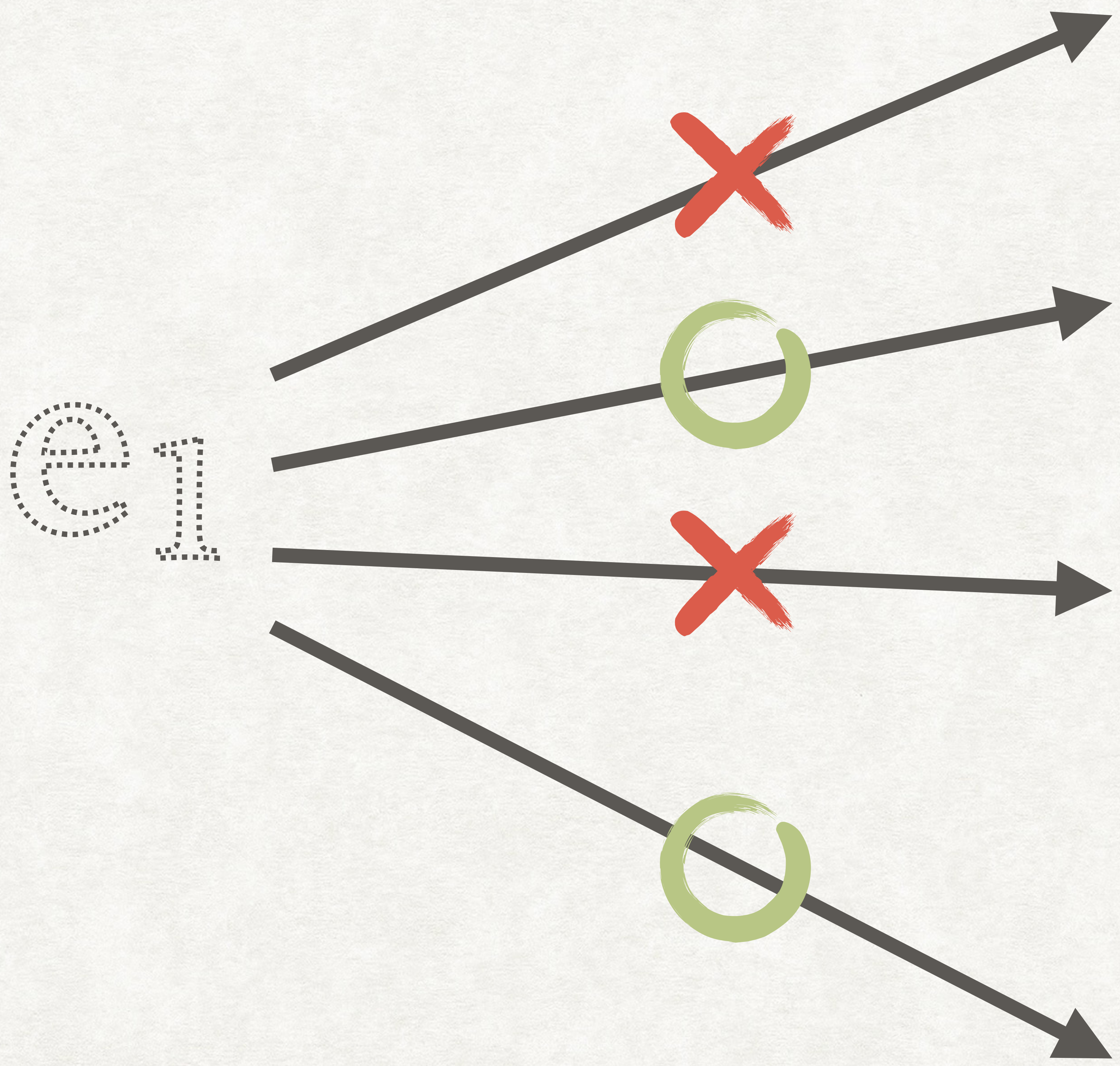
Delete! e_1 $\longrightarrow$ $e_c = 42$

Independent!

MOAD



MOAD



Compile & Execute

$e_2 = 42$

$e_3 = 3.141592$

$e_4 = $ "foo"

$\vdots$

$e_n = \mathrm{bar}()$

MOAD

$e_1 \dashrightarrow e_c$

```
e1: ~~foo = Foo()~~

ec: answer = 42
```

MOAD

$e_1 \not\dashrightarrow e_c$

Preserved!

```
e1: foo = Foo()

ec: answer = 42
```

$$\downarrow$$

$$\mathit{Traj}(\texttt{answer}) = 42$$

MOAD

e1: ~~foo = Foo()~~

e2: foo.bar = 2

ec: answer = 42

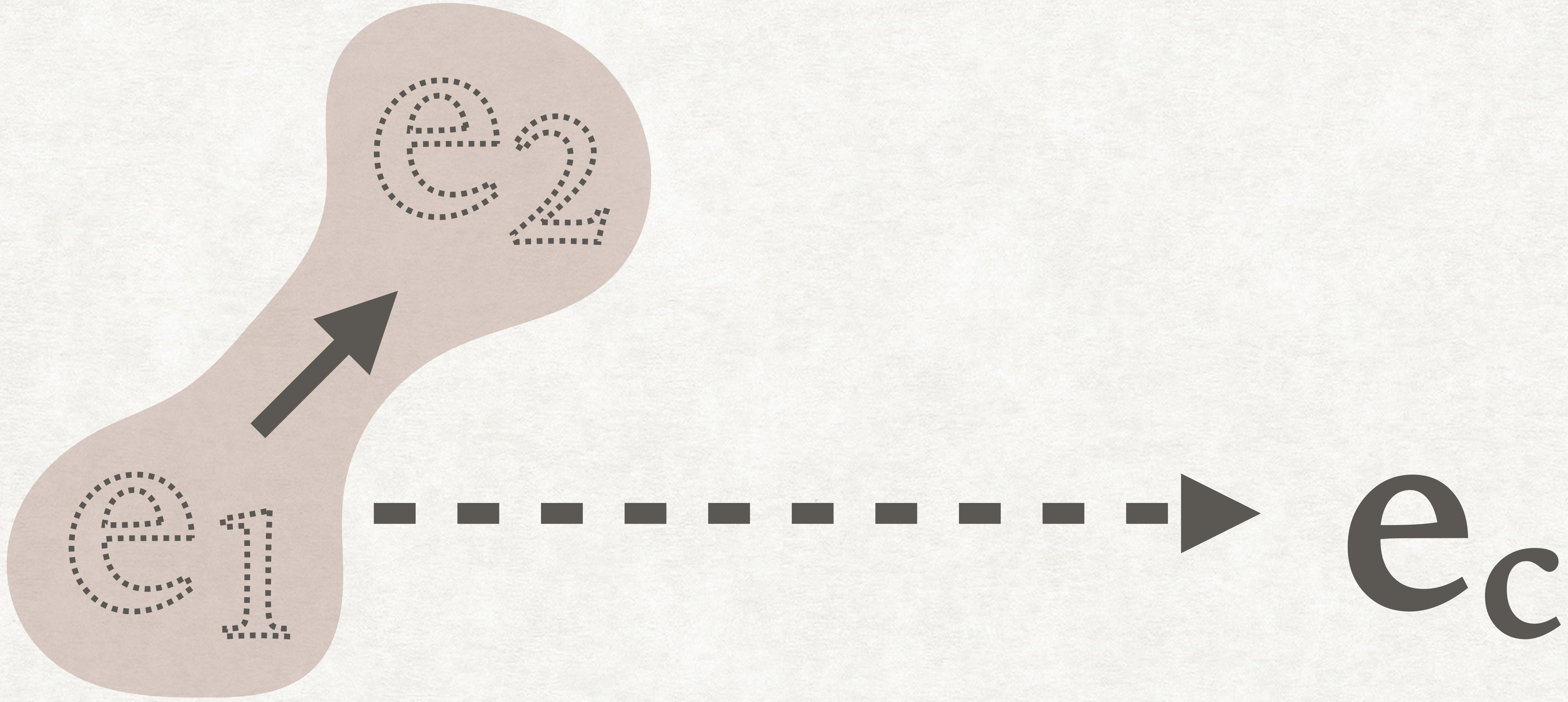
MOAD

`e1:` ~~`foo = Foo()`~~
`e2: foo.bar = 2`
`ec: answer = 42`

$$\downarrow$$

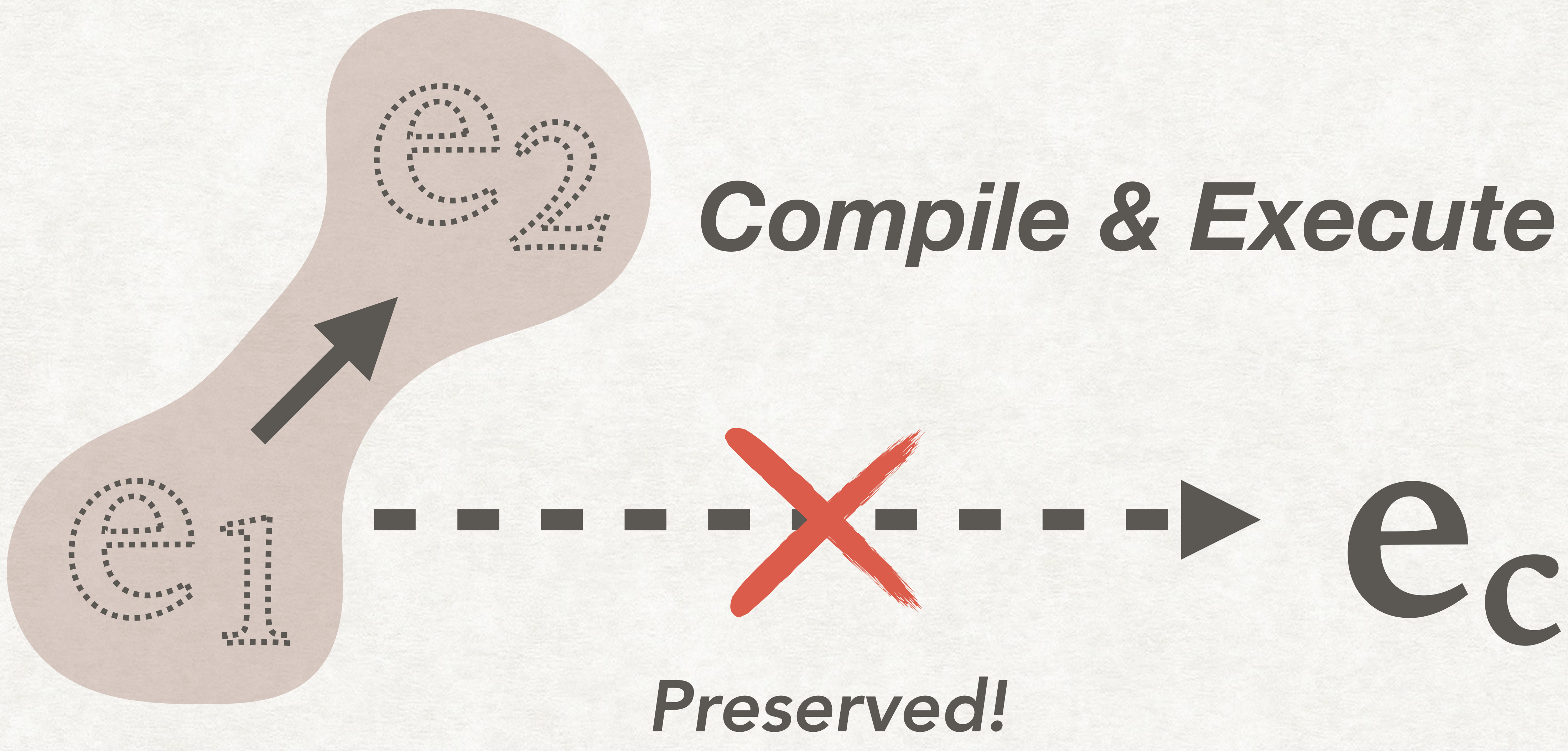
$$Traj(\mathtt{answer}) = \varnothing$$

MOAD




```
e1: ~~foo = Foo()~~
e2: ~~foo.bar = 2~~
ec: answer = 42
```

MOAD



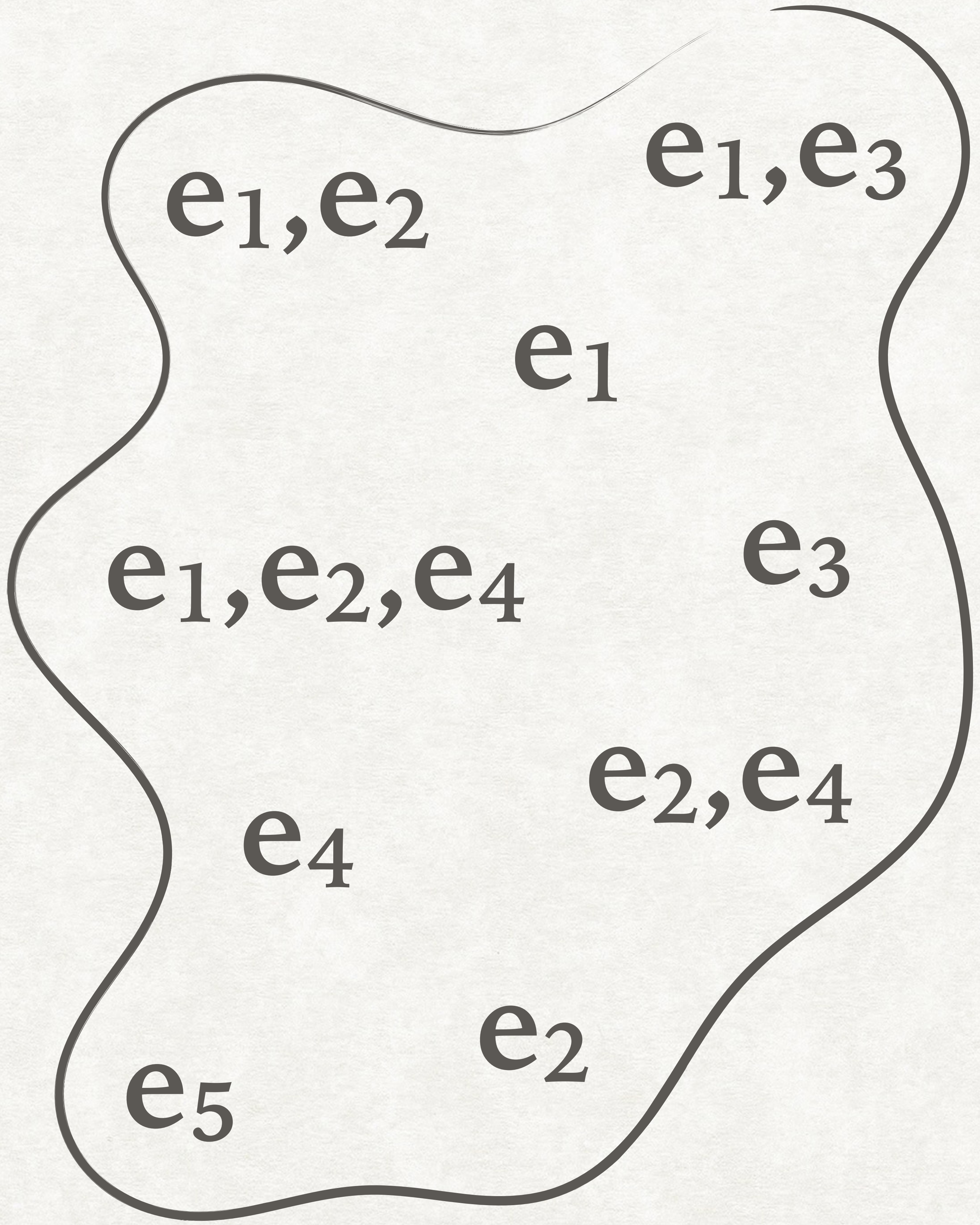
e1: ~~foo = Foo()~~

e2: ~~foo.bar = 2~~

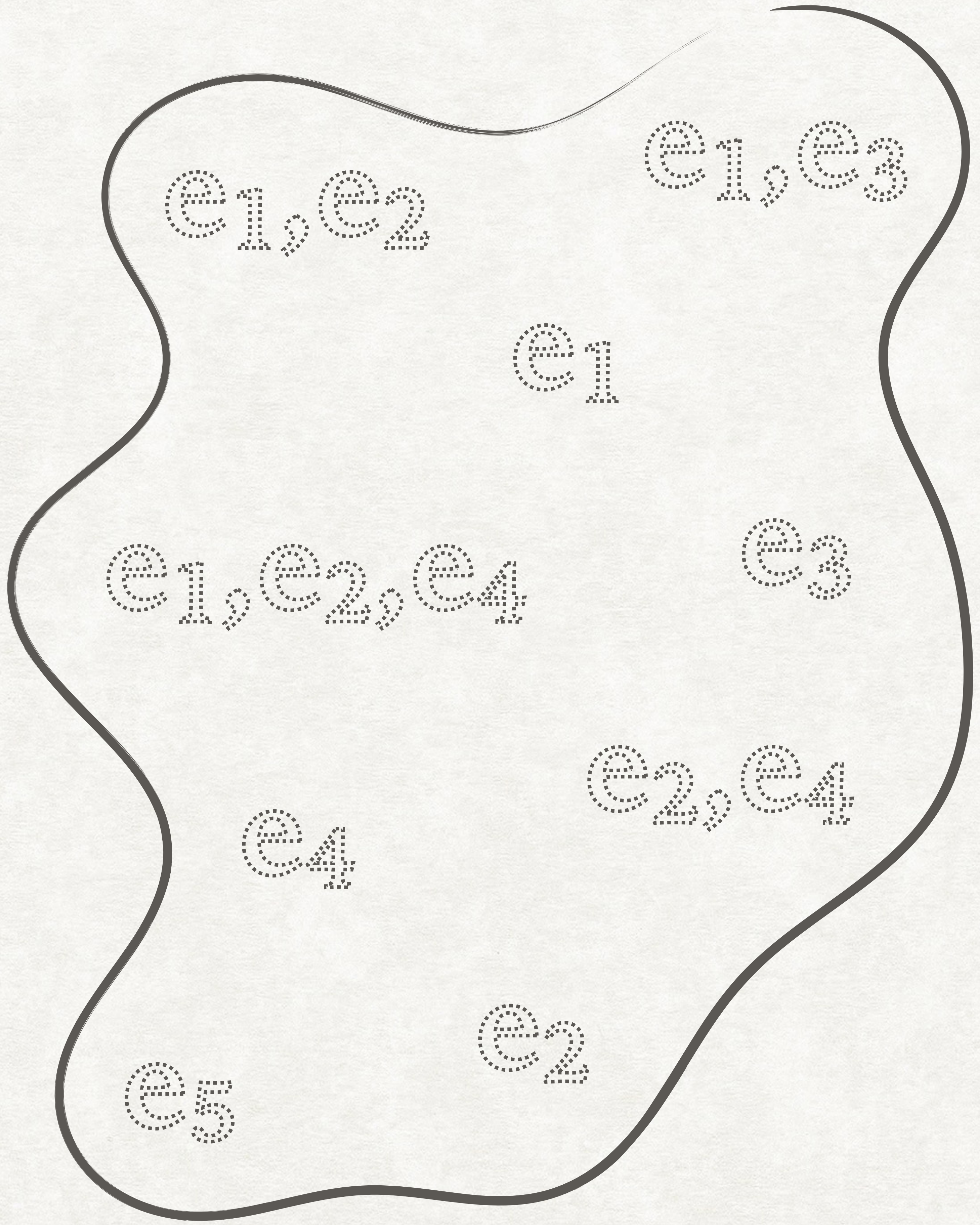
ec: answer = 42

$$Traj(\texttt{answer}) = 42$$

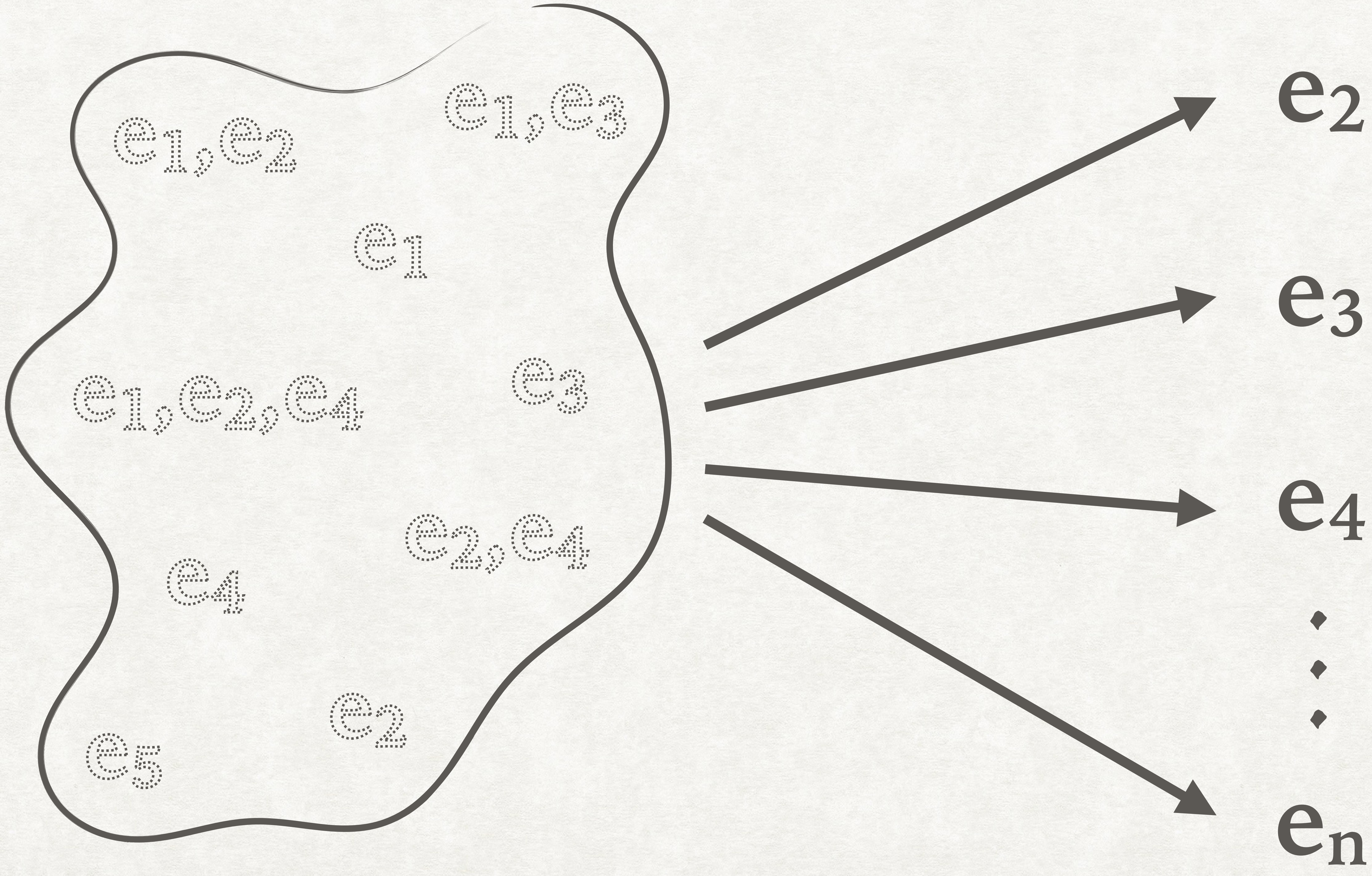
OBSERVATION PHASE



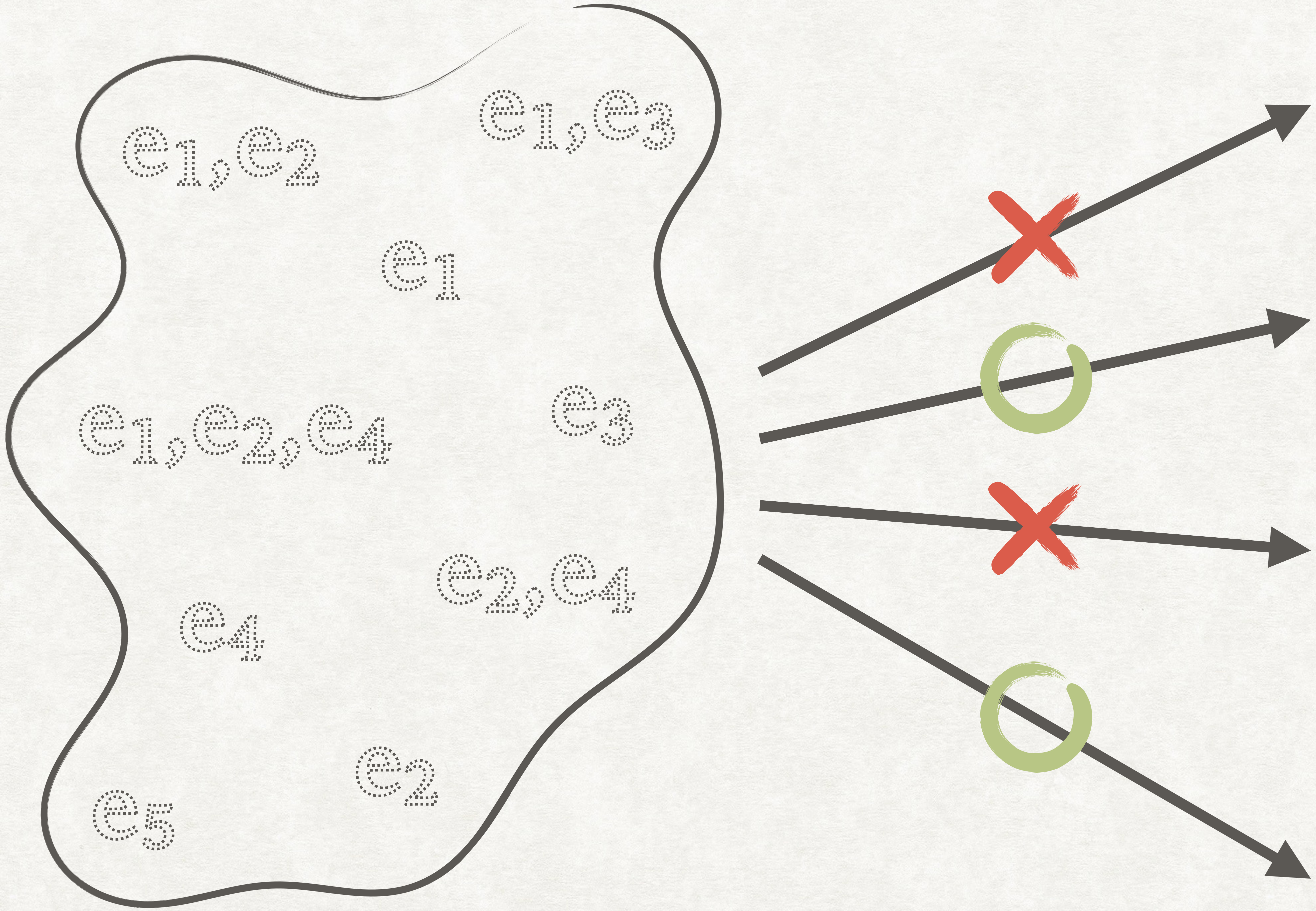
OBSERVATION PHASE



OBSERVATION PHASE



OBSERVATION PHASE



Compile & Execute

$e_2 = 42$

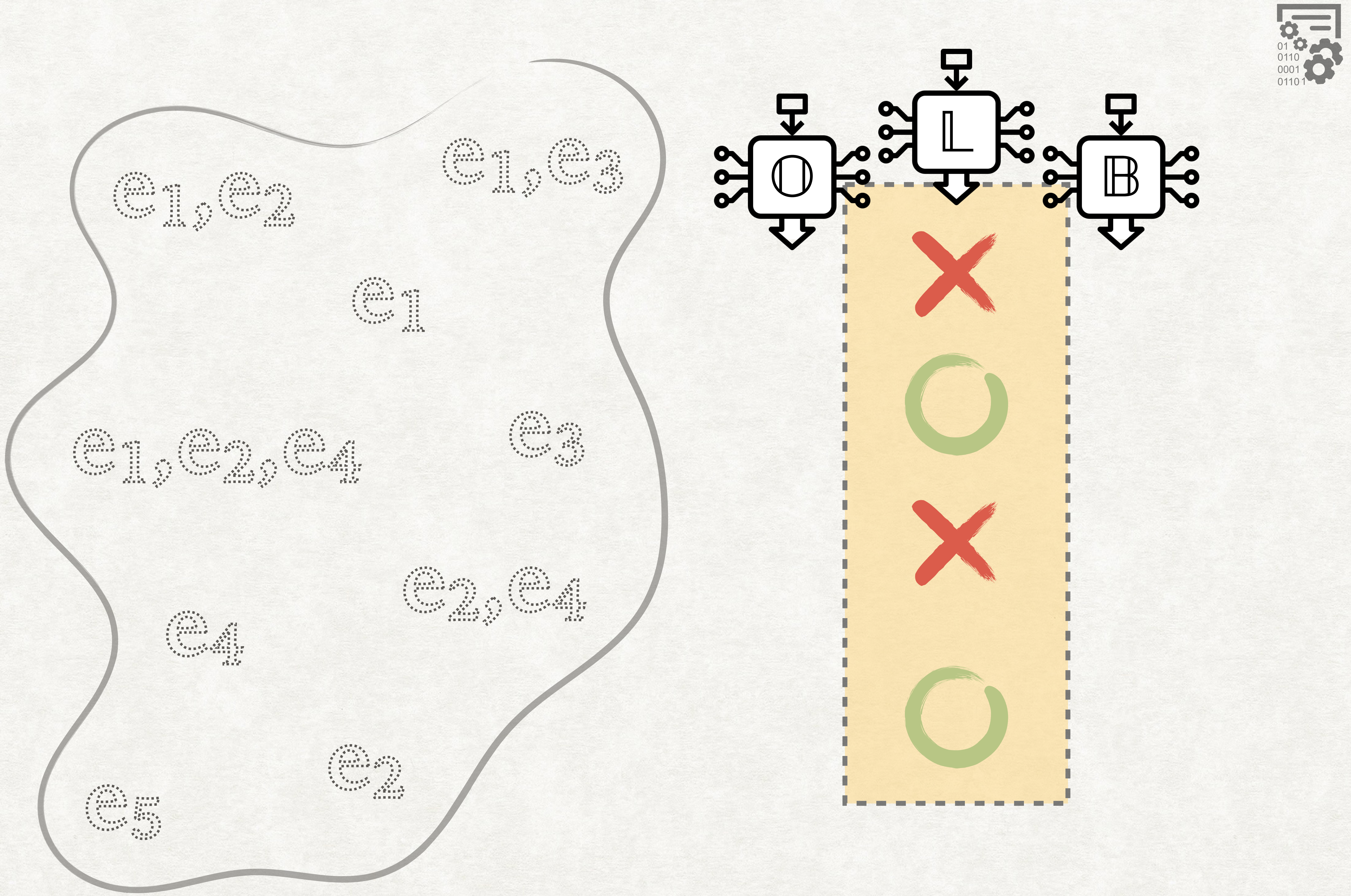
$e_3 = 3.141592$

$e_4 =$ "foo"

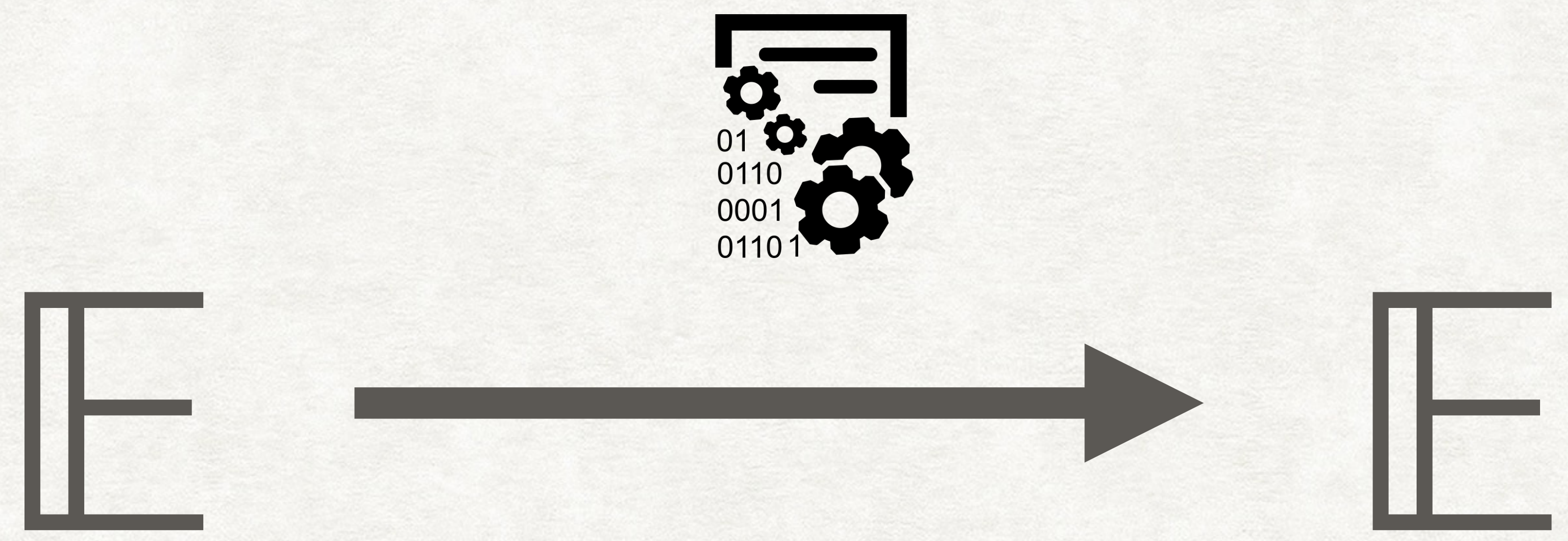
$\vdots$

$e_n =$ bar()

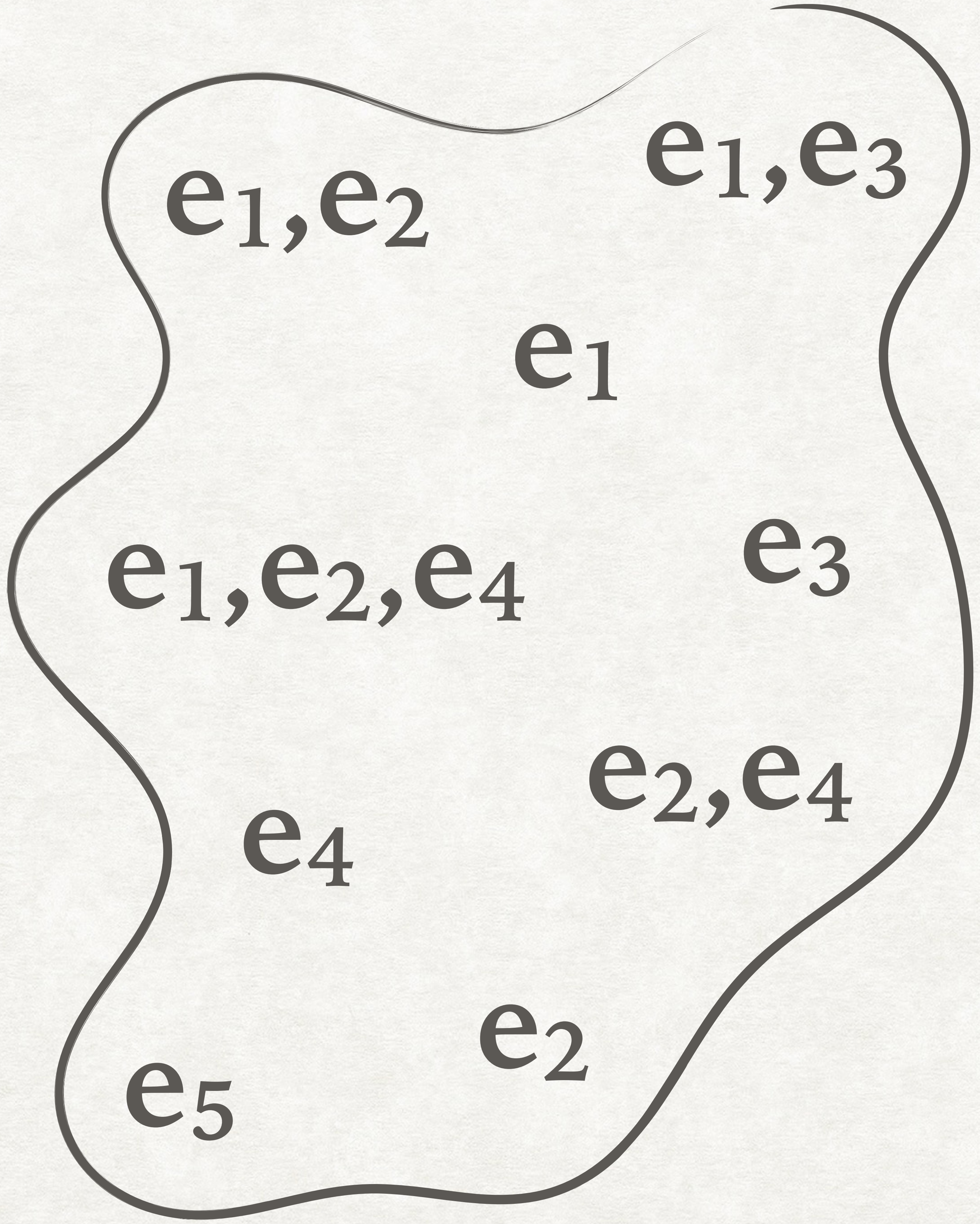
INFERENCE PHASE

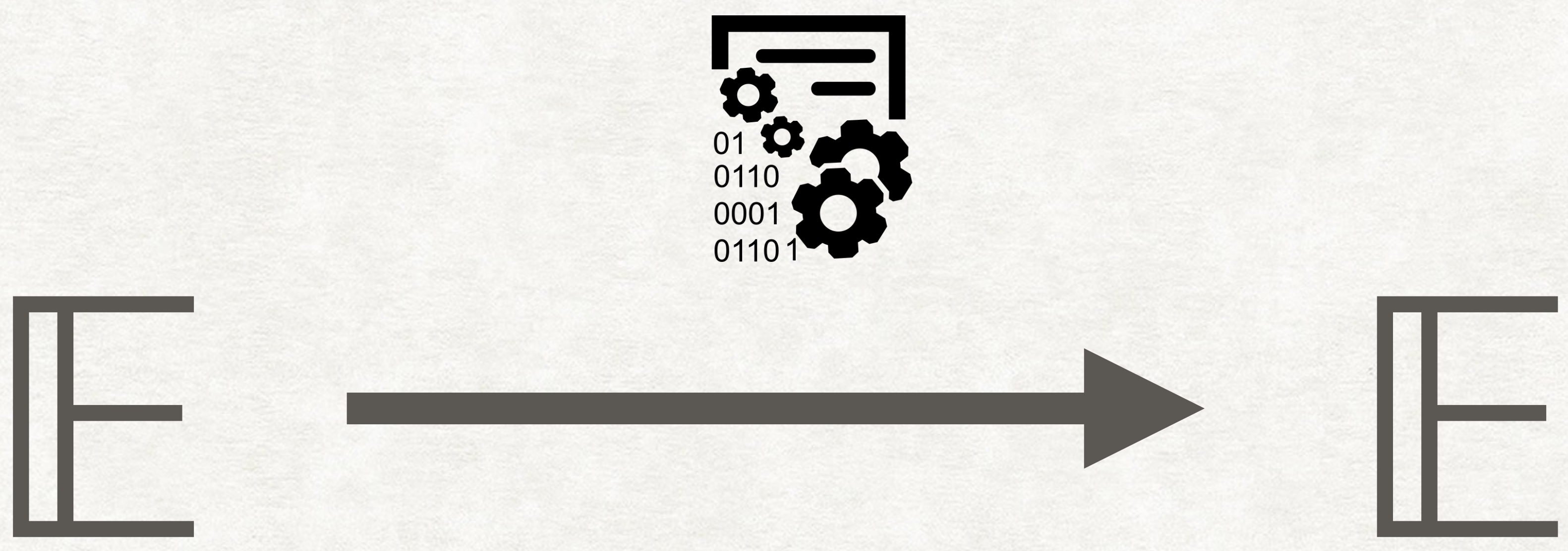


MOAD

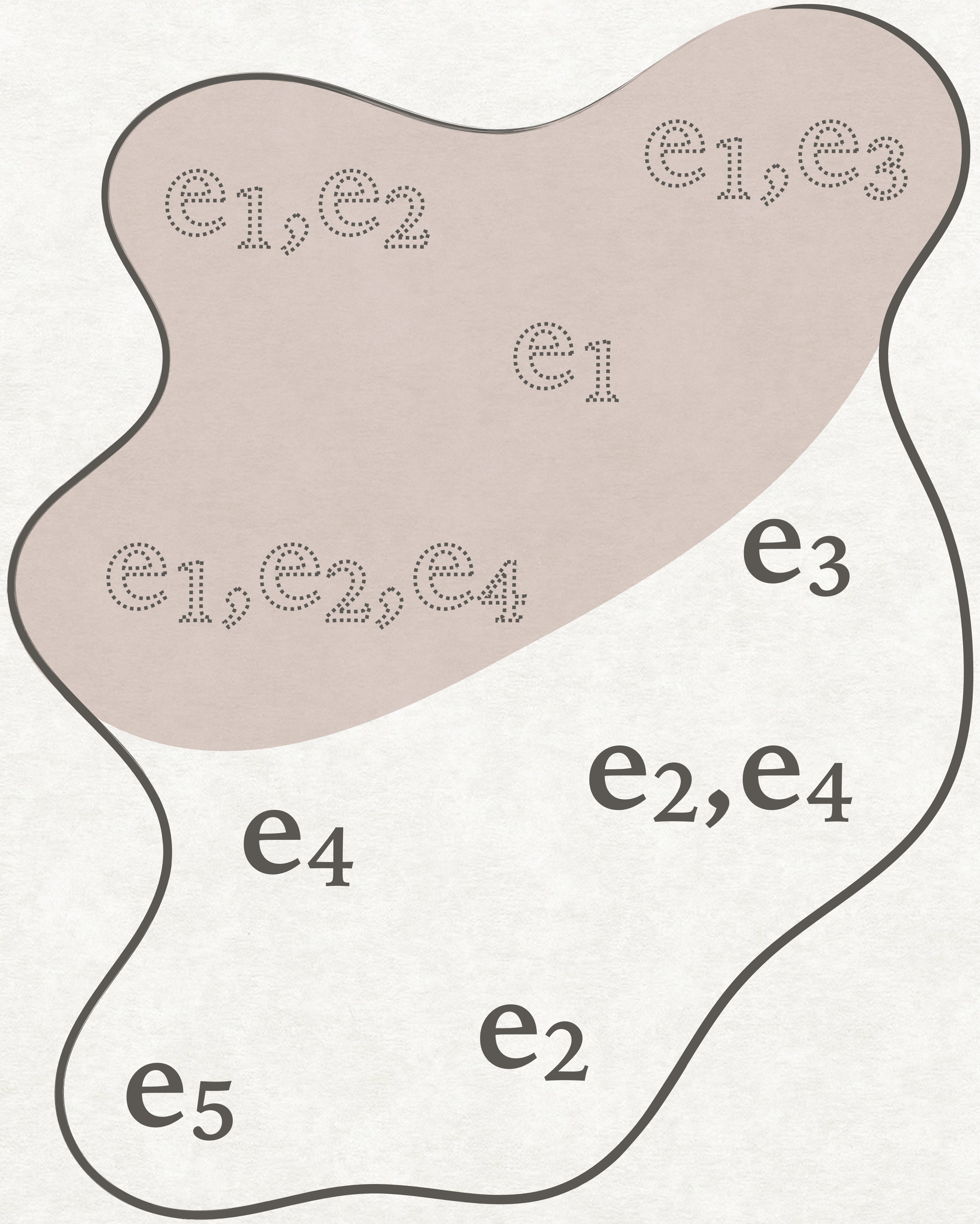


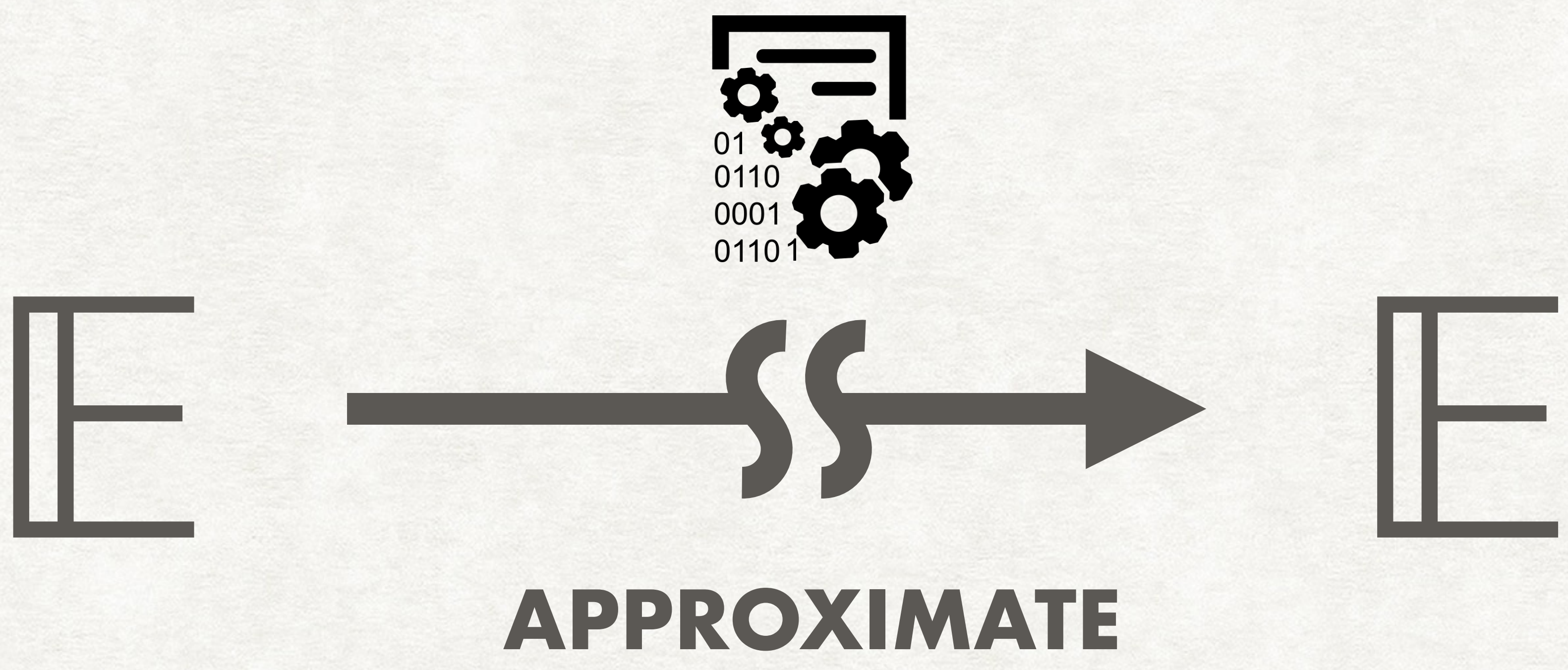
MOAD





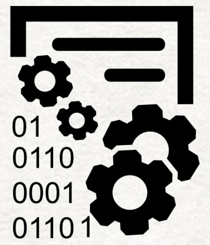
MOAD


e_1, e_2
e_1, e_3
e_1
e_1, e_2, e_4
e_3
e_2, e_4
e_4
e_2
e_5


01
0110
0001
01101
$\mathbb{E}$
$\mathbb{E}$
APPROXIMATE

ADVANTAGE OF MOAD VS. ORBS

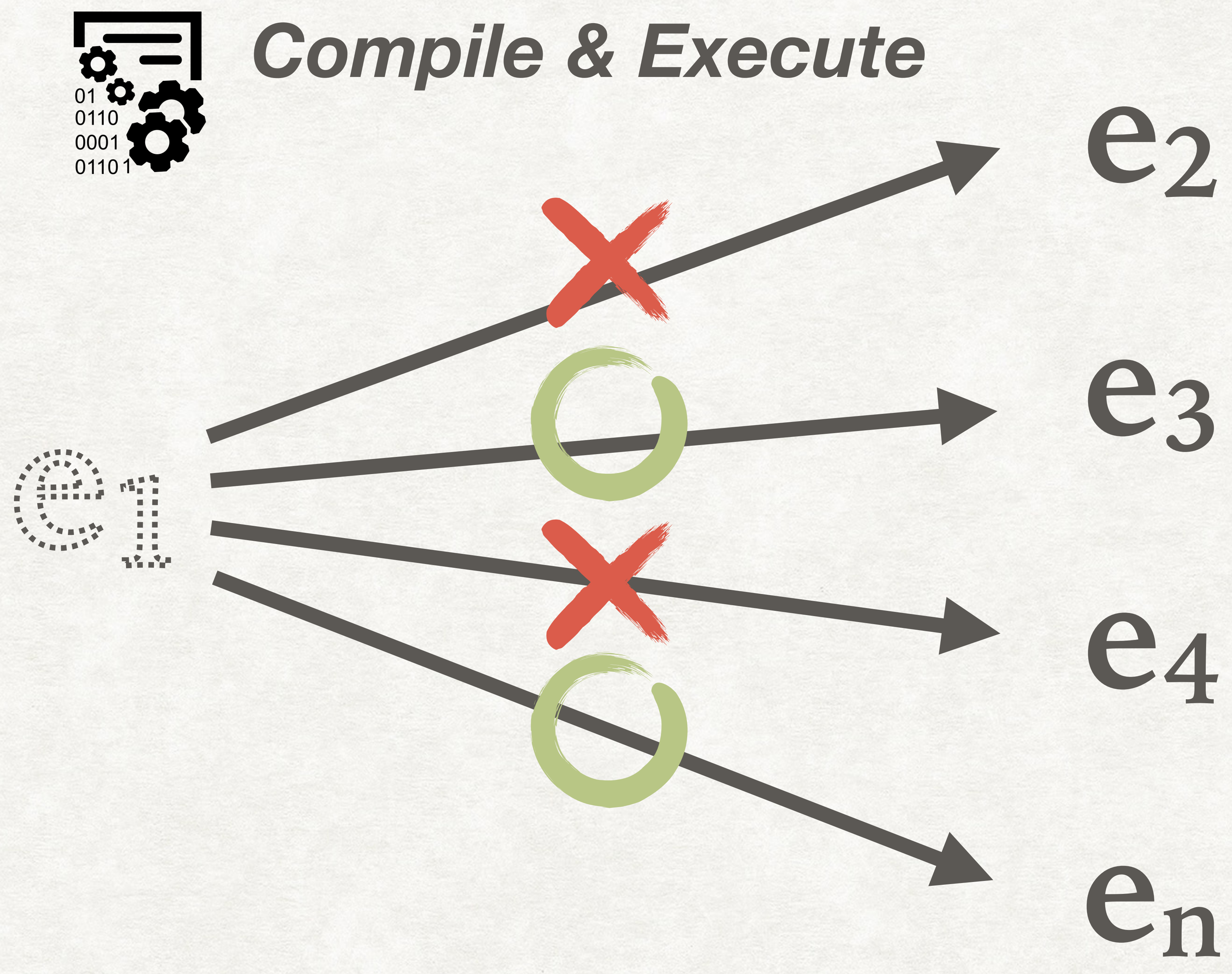
ADVANTAGE OF MOAD VS. ORBS



Compile & Execute

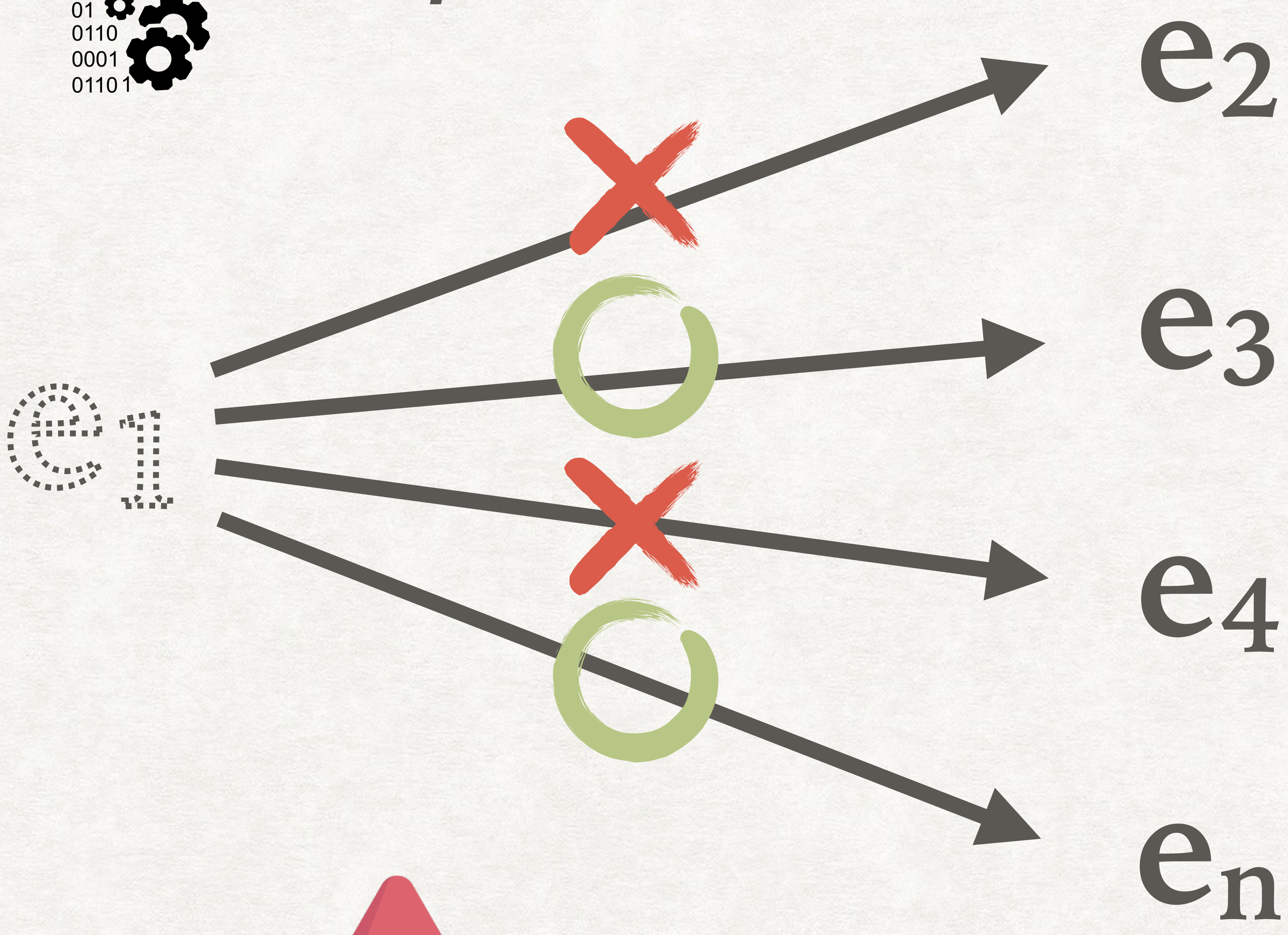
$$e_1 \not\longrightarrow e_c$$

ADVANTAGE OF MOAD VS. ORBS



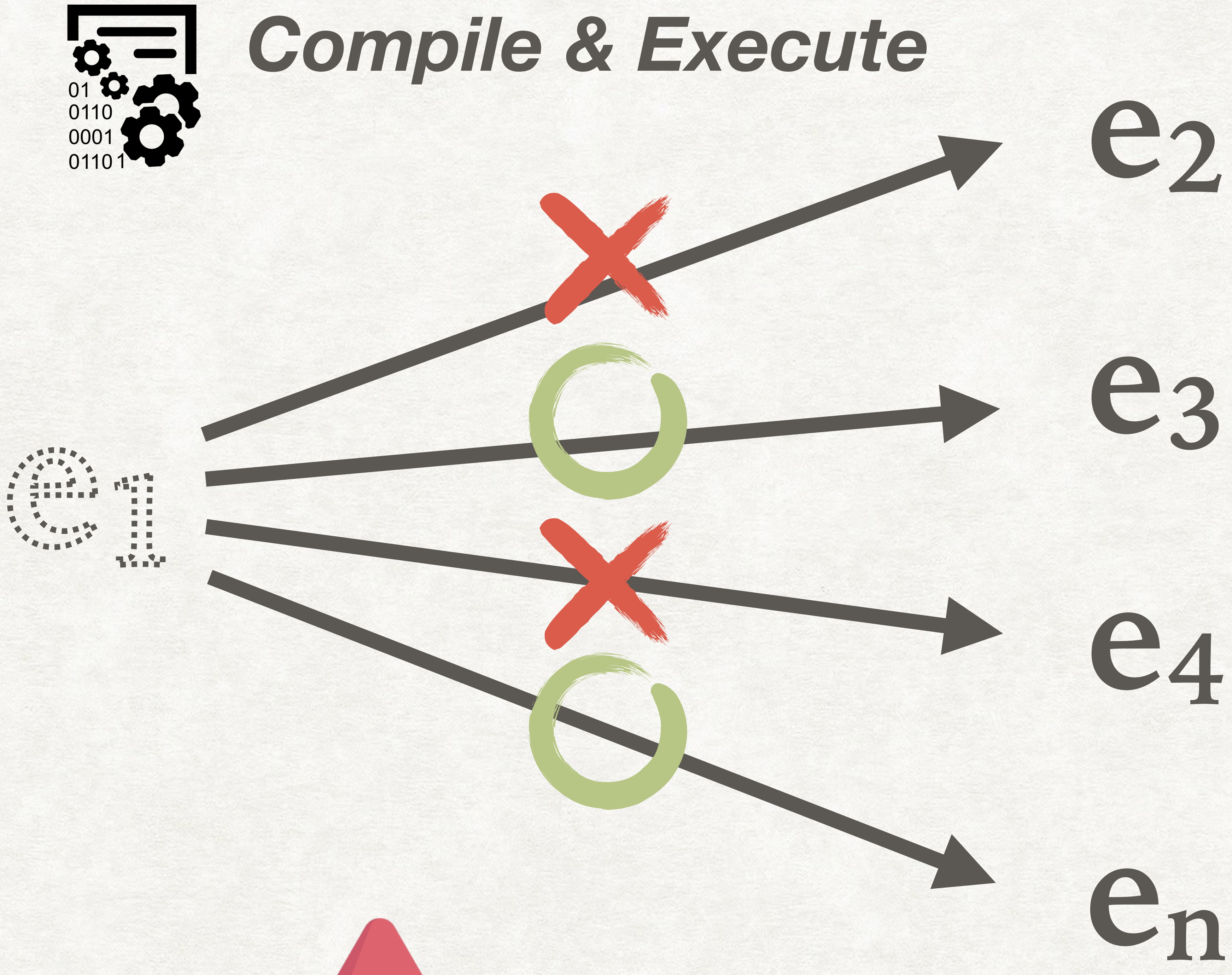
ADVANTAGE OF MOAD VS. ORBS

Compile & Execute

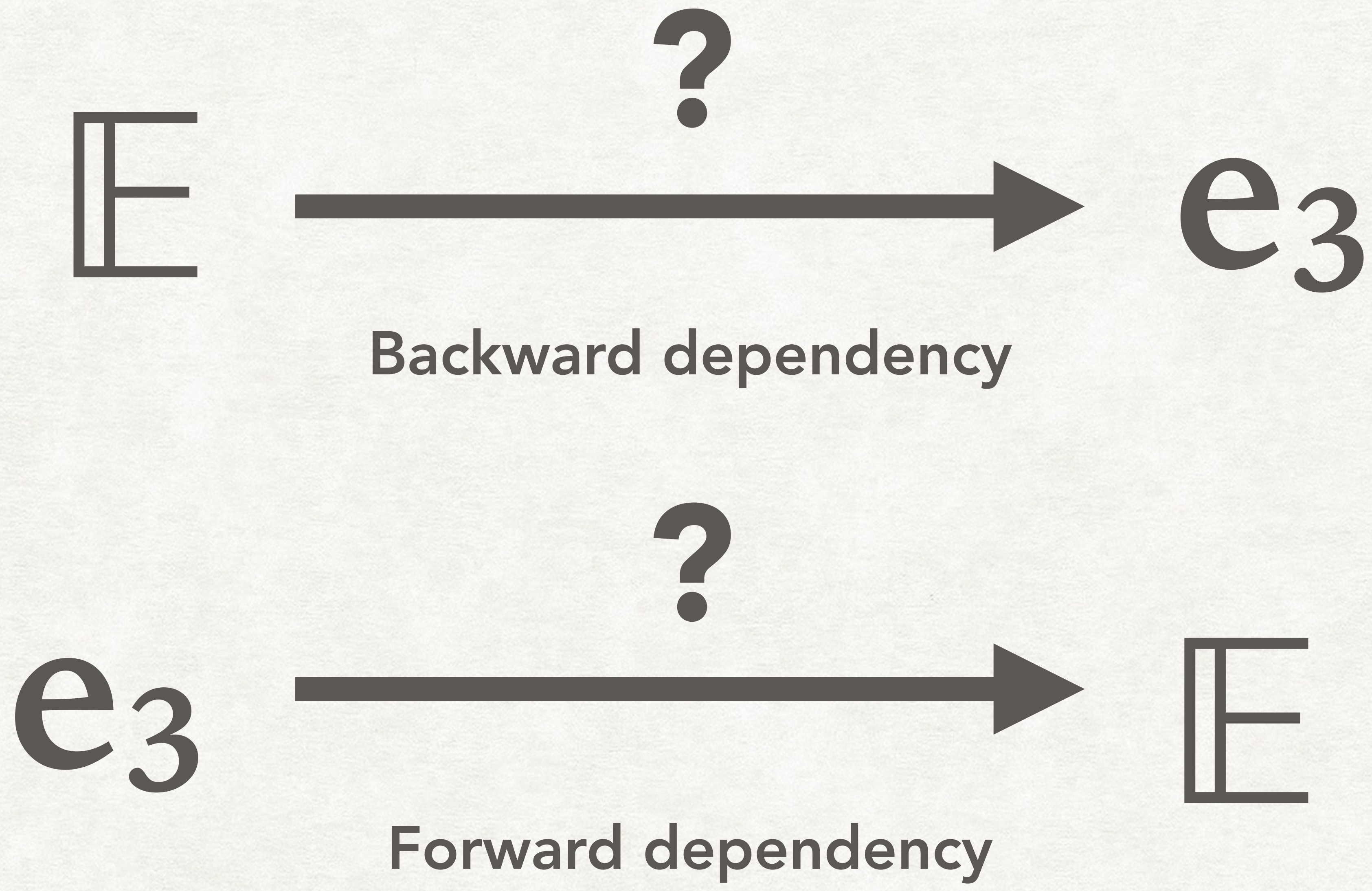


1. <u>EFFICIENCY</u>

ADVANTAGE OF MOAD VS. ORBS



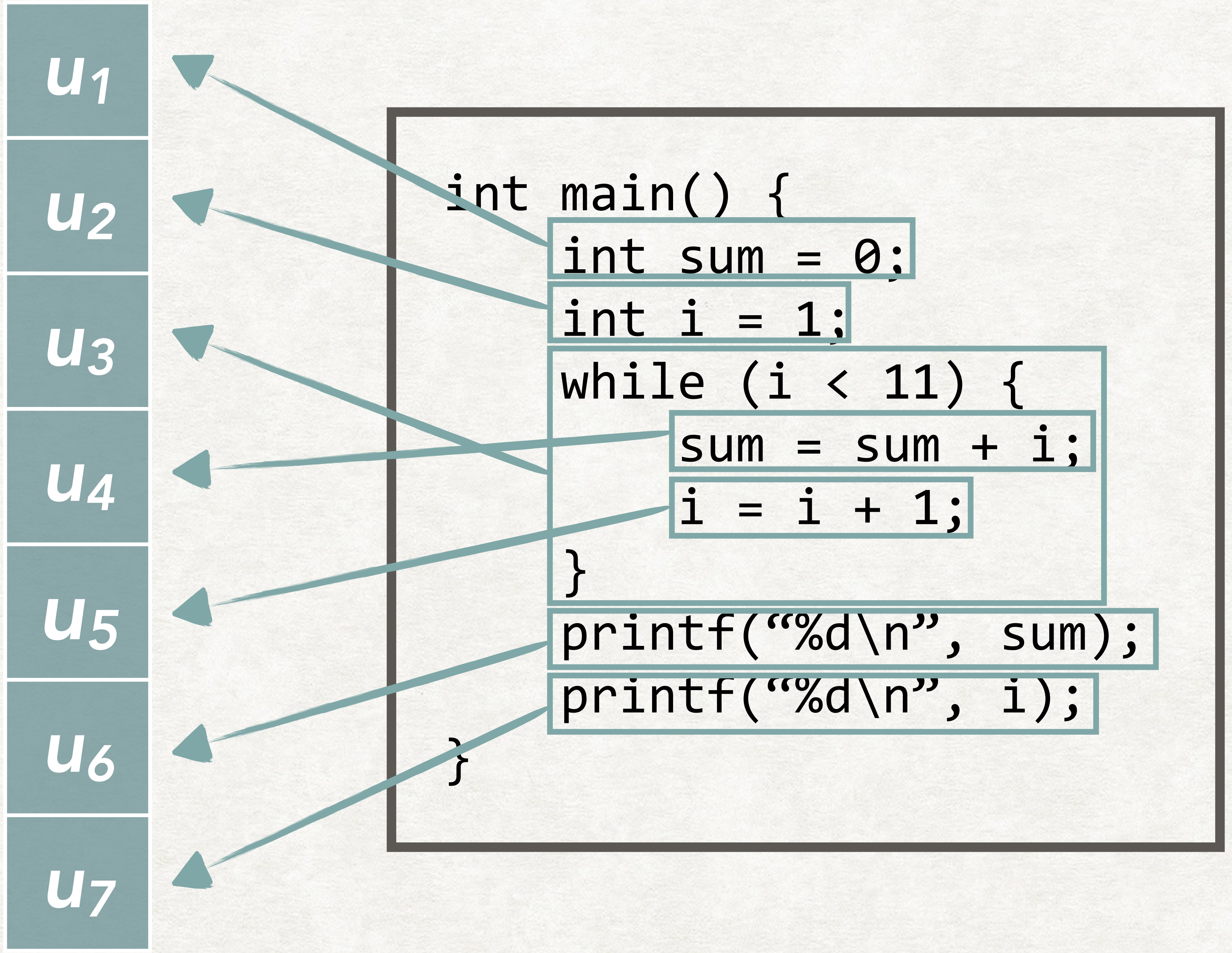
1. EFFICIENCY



2. COMPLETENESS

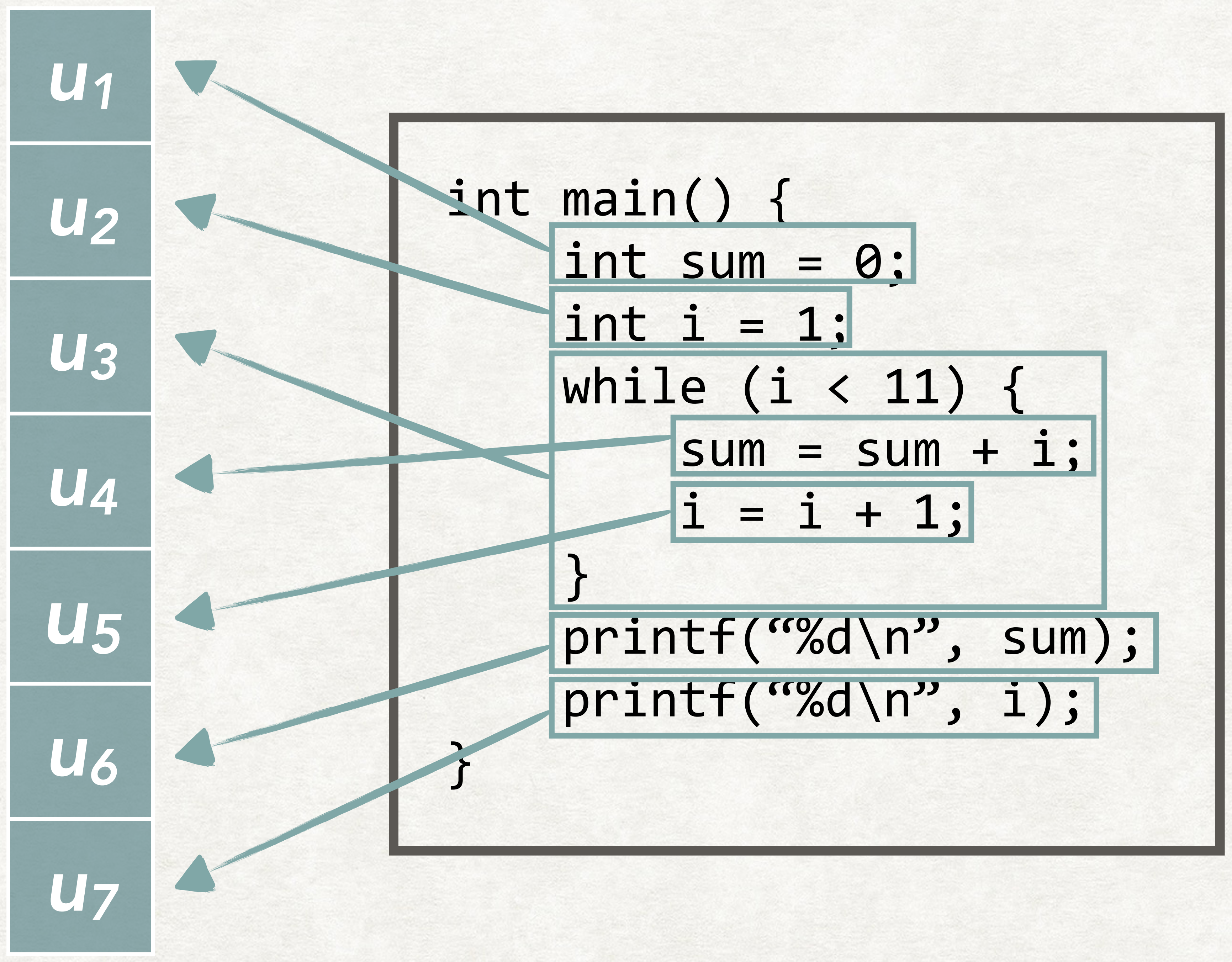
DELETION GENERATION SCHEME

- Generate a set of "*deletions*" (partial programs) to observe by **deletion generation schemes**



DELETION GENERATION SCHEME

- Generate a set of "*deletions*" (partial programs) to observe by **deletion generation schemes**

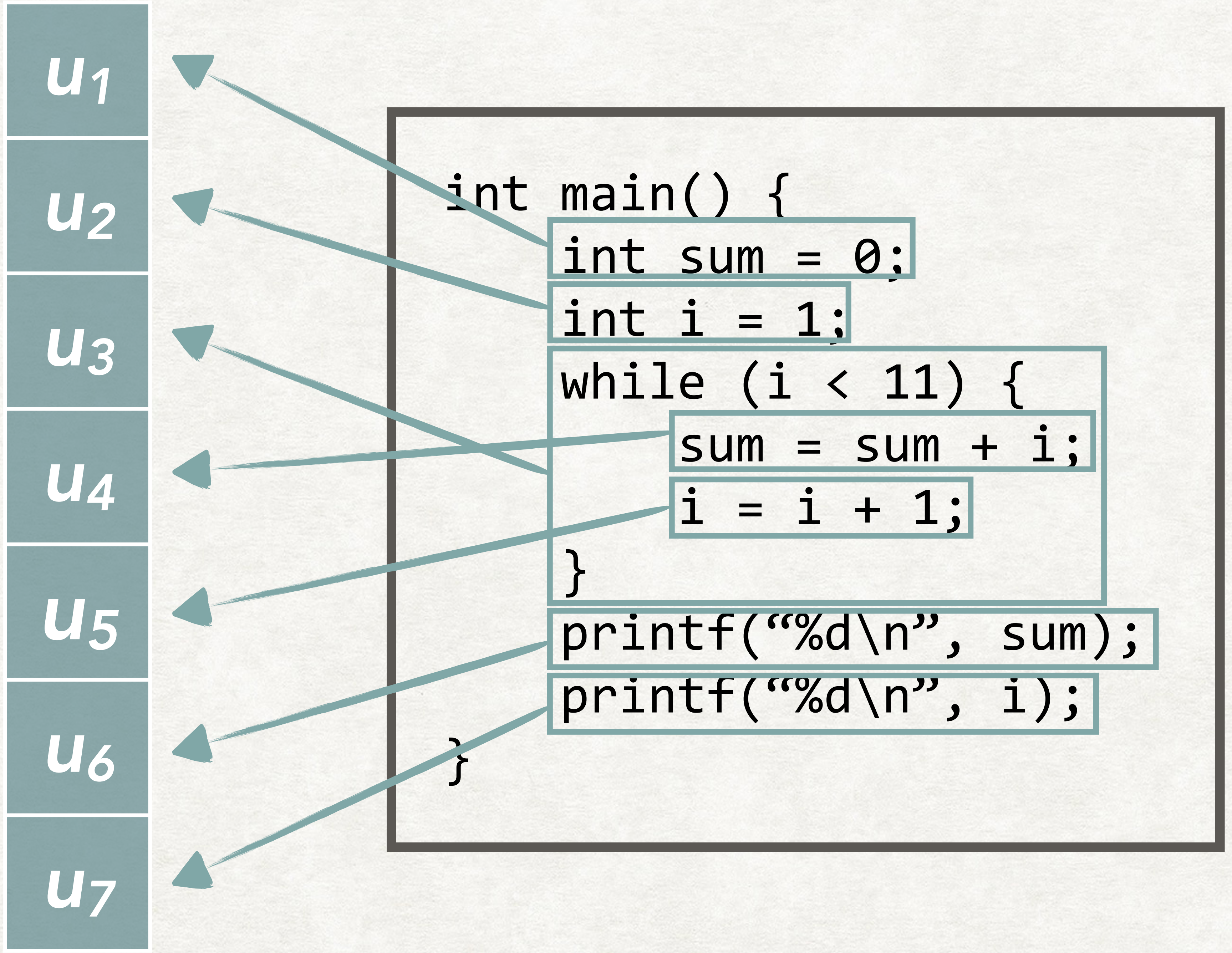


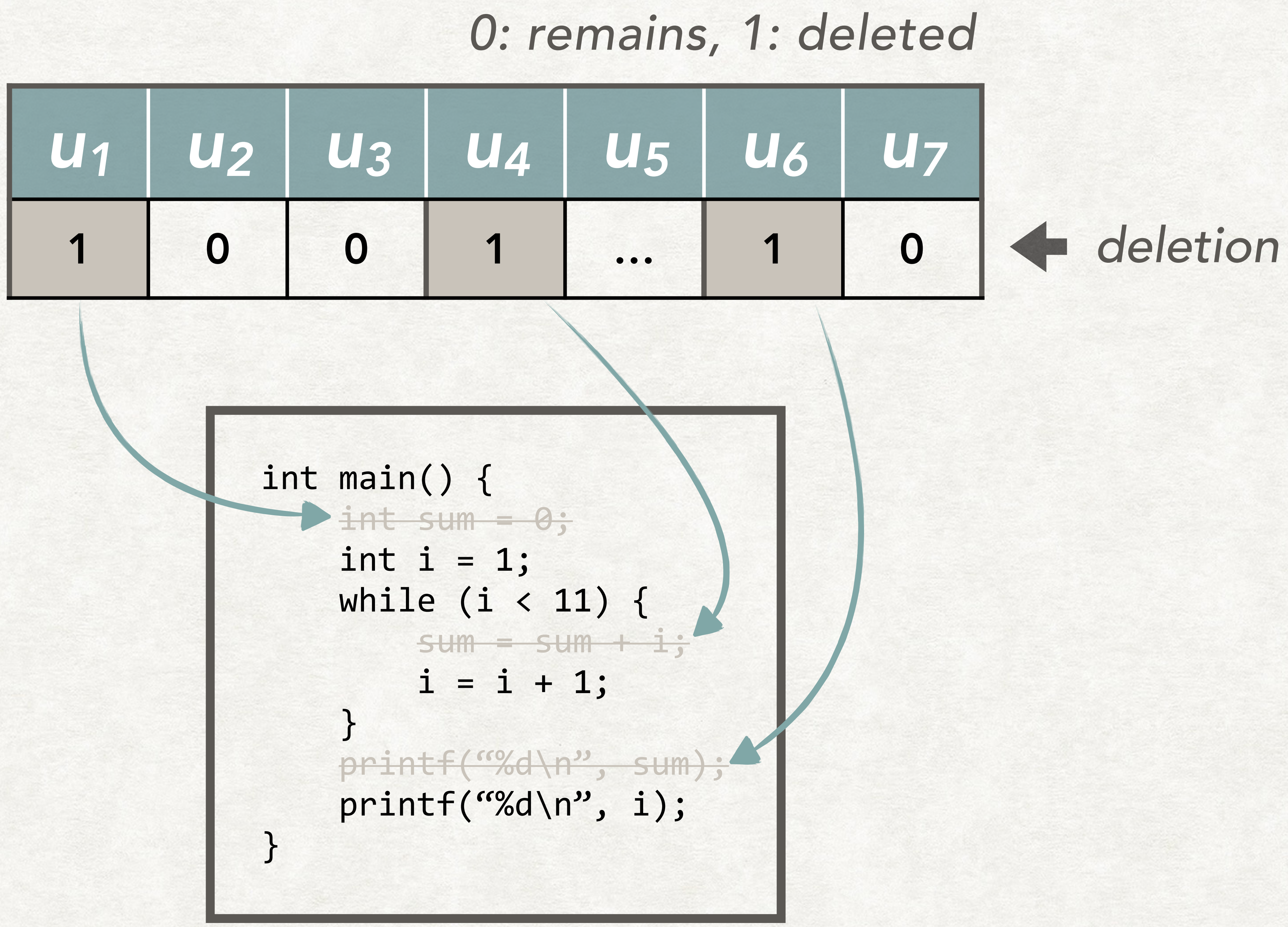
0: remains, 1: deleted

u_1	u_2	u_3	u_4	u_5	u_6	u_7	
1	0	0	1	…	1	0	← *deletion*

DELETION GENERATION SCHEME

- Generate a set of "*deletions*" (partial programs) to observe by **deletion generation schemes**





DELETION GENERATION SCHEME

- Generate a set of "*deletions*" (partial programs) to observe by **deletion generation schemes**
  - 0: remains, 1: deleted

1-HOT : every single statement

	u_1	u_2	u_3	u_4	…	u_{i-1}	u_i
original (no deletion)	0	0	0	0	…	0	0
	1	0	0	0	…	0	0
	0	1	0	0	…	0	0
	…	…	…	…	…	…	…
	0	0	0	0	…	0	1

+

2-HOT : **1-HOT +** every pair of statements

u_1	u_2	u_3	u_4	…	u_{i-1}	u_i
…	…	…	…	…	…	…
1	1	0	0	…	0	0
1	0	1	0	…	0	0
…	…	…	…	…	…	…
0	0	0	0	…	1	1

OBSERVATION PHASE

- Generate a set of "*deletions*" (partial programs) to observe by **deletion generation schemes**
  - 0: remains, 1: deleted
- Run the programs, check whether the trajectory changed (`0`) or not (`1`) for each variable.

u_1	u_2	u_3	u_4	…	u_{i-1}	u_i	Observe	v_1	v_2	v_3	…	v_j
0	0	0	0	…	0	0	➡	1	1	1	…	1
1	0	0	0	…	0	0	➡	0	0	0	…	1
0	1	0	0	…	0	0	➡	1	0	1	…	0
…	…	…	…	…	…	…		…	…	…	…	…
0	0	0	0	…	1	1	➡	0	0	1	…	0

u_1	u_2	u_3	u_4	...	u_{i-1}	u_i	Observe	v_1	v_2	v_3	...	v_j
0	0	0	0	...	0	0	→	1	1	1	...	1
1	0	0	0	...	0	0	→	0	0	0	...	1
0	1	0	0	...	0	0	→	1	0	1	...	0
...	...	...	...	...	...	...		...	...	...	...	...
0	0	0	0	...	1	1	→	0	0	1	...	0

M :

u_1	u_2	u_3	u_4	...	u_{i-1}	u_i
0	0	1	0	...	0	1

→

v_1
?

STATISTICAL MODEL

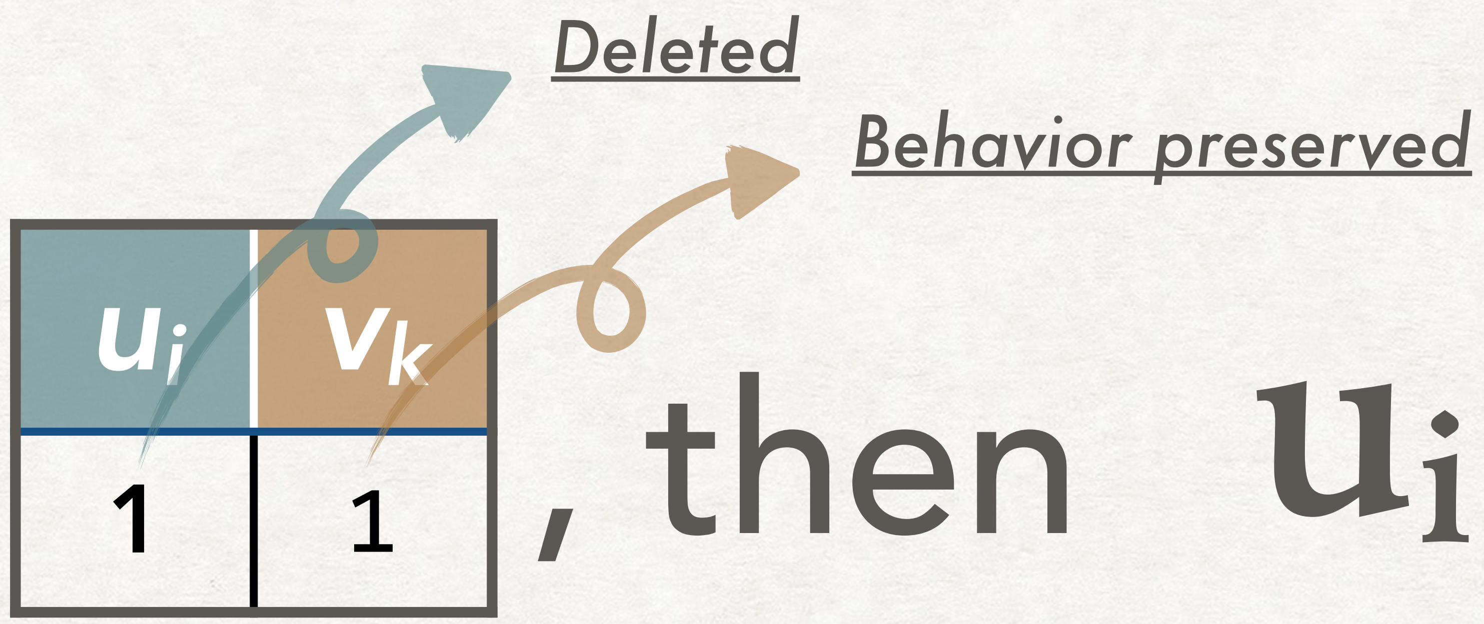
STATISTICAL MODEL → INFER DEPENDENCY

- Main hypothesis:

> A variable v_k is more likely to be independent of a statement u_i if more observations show that v_k preserves its behavior when u_i is deleted.

INFERENCE MODEL

1. Once success (⦷)

If 

, then $u_i \not\longrightarrow v_k$

If the behavior of v_k is preserved at least once when u_i is deleted, then v_k is independent from u_i.

INFERENCE MODEL

1. Once success (◯)

Deleted

Behavior preserved

If

u_i	v_k
1	1

, then $u_i \not\longrightarrow v_k$

~~dict[“a”] = set()~~
...
dict[“a”].add(3)

Good!

If the behavior of v_k is preserved at least once when u_i is deleted, then v_k is independent from u_i.

INFERENCE MODEL

1. Once success (◯)

Deleted

Behavior preserved

If

u_i	v_k
1	1

, then $u_i \not\longrightarrow v_k$

If the behavior of v_k is preserved at least once when u_i is deleted, then v_k is independent from u_i.

Case 1. Okay!

~~`dict["a"] = set()`~~
`...`
`if "a" not in dict: dict["a"] = set()`
`dict["a"].add(3)`

Case 2. Okay!

`dict["a"] = set()`
`...`
~~`if "a" not in dict: dict["a"] = set()`~~
`dict["a"].add(3)`

Case 3. Nope!!

~~`dict["a"] = set()`~~
`...`
~~`if "a" not in dict: dict["a"] = set()`~~
`dict["a"].add(3)`

INFERENCE MODEL

2. Logistic ($\mathbb{L}$)

3. Bayesian ($\mathbb{B}$)

INFERENCE MODEL

2. Logistic ($\mathbb{L}$)

u_1	u_2	…	u_i	v_k
1	0	…	0	0
0	1	…	0	1
…	…	…	…	…
1	0	…	0	0

$$\log \frac{v_k}{1 - v_k} = \beta_0 + \beta_1 u_1 + \beta_2 u_2 + \cdots + \beta_i u_i$$

3. Bayesian ($\mathbb{B}$)

INFERENCE MODEL

2. Logistic ($\mathbb{L}$)

u_1	u_2	…	u_i	v_k
1	0	…	0	0
0	1	…	0	1
…	…	…	…	…
1	0	…	0	0

$$\log \frac{v_k}{1 - v_k} = \beta_0 + \beta_1 u_1 + \beta_2 u_2 + \cdots + \beta_i u_i$$

Coefficients represent the relative impact on dependence

3. Bayesian ($\mathbb{B}$)

INFERENCE MODEL

2. Logistic ($\mathbb{L}$)

u_1	u_2	...	u_i	v_k
1	0	...	0	0
0	1	...	0	1
...	...	...	...	…
1	0	...	0	0

$$\log \frac{v_k}{1 - v_k} = \beta_0 + \beta_1 u_1 + \beta_2 u_2 + \cdots + \beta_i u_i$$

Coefficients represent the relative impact on dependence

If $\beta_i > 0$, then $u_i \nrightarrow v_k$

If $\beta_i \leq 0$, then $u_i \rightarrow v_k$

If β_i, the coefficient for u_i of the logistic regression for v_k, is larger than 0, then v_k is independent from u_i.

3. Bayesian ($\mathbb{B}$)

INFERENCE MODEL

2. Logistic ($\mathbb{L}$)

u_1	u_2	...	u_i	v_k
1	0	...	0	0
0	1	...	0	1
...	...	...	...	…
1	0	...	0	0

$$\log \frac{v_k}{1 - v_k} = \beta_0 + \beta_1 u_1 + \beta_2 u_2 + \cdots + \beta_i u_i$$

Coefficients represent the relative impact on dependence

If $\beta_i > 0$, then $u_i \not\rightarrow v_k$

If $\beta_i \leq 0$, then $u_i \rightarrow v_k$

If β_i, the coefficient for u_i of the logistic regression for v_k, is larger than 0, then v_k is independent from u_i.

3. Bayesian ($\mathbb{B}$)

$Dep\ u_i \rightarrow v_k$: how much does u_k affects when v_k changed

$$\sim P(u_i = 1 \mid v_k = 0) = \frac{P(v_k = 0 \mid u_i = 1)P(u_i = 1)}{P(v_k = 0)}$$

$$\approx P(v_k = 0 \mid u_i = 1) := P(v_k \mid u_i)$$

u_1	v_k
1	0
0	1
1	1

$P(v_k \mid u_i) = 1/2$

INFERENCE MODEL

2. Logistic ($\mathbb{L}$)

u_1	u_2	...	u_i	v_k
1	0	...	0	0
0	1	...	0	1
...	...	...	...	…
1	0	...	0	0

$$\log \frac{v_k}{1 - v_k} = \beta_0 + \beta_1 u_1 + \beta_2 u_2 + \cdots + \beta_i u_i$$

Coefficients represent the relative impact on dependence

If $\beta_i > 0$, then $u_i \not\rightarrow v_k$

If $\beta_i \leq 0$, then $u_i \rightarrow v_k$

If β_i, the coefficient for u_i of the logistic regression for v_k, is larger than 0, then v_k is independent from u_i.

3. Bayesian ($\mathbb{B}$)

Dep $u_i \rightarrow v_k$: how much does u_k affects when v_k changed

$$\sim P(u_i = 1 \mid v_k = 0) = \frac{P(v_k = 0 \mid u_i = 1)P(u_i = 1)}{P(v_k = 0)}$$

$$\approx P(v_k = 0 \mid u_i = 1) := P(v_k \mid u_i)$$

u_1	v_k
1	0
0	1
1	1

$P(v_k \mid u_i) = 1/2$

μ : average of the probability over units

If $\hat{P}(v_k \mid u_i) > \mu$, then $u_i \not\rightarrow v_k$

If $\hat{P}(v_k \mid u_i) \leq \mu$, then $u_i \rightarrow v_k$

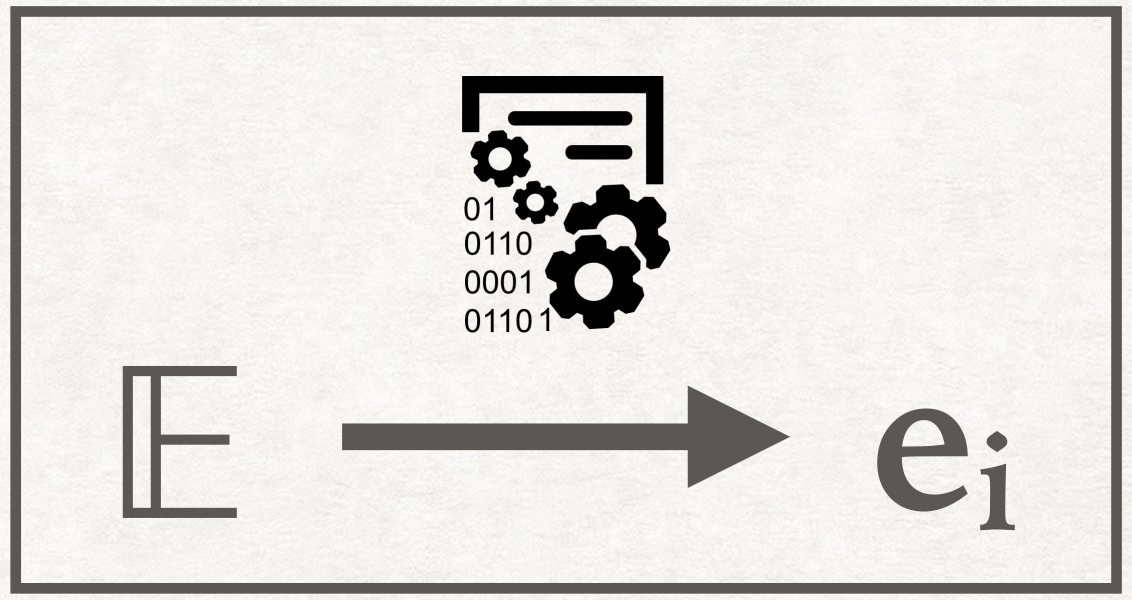
If the $P(v_k$ behaves the same | u_i has been deleted) is larger than the mean, then v_k Is independent from u_i.

EVALUATION in terms of program slicing

EVALUATION in terms of program slicing

	$\mathbb{O}$	$\mathbb{L}$	$\mathbb{B}$
1-HOT			
2-HOT			

VS



ORBS

RQ1. MOAD vs. ORBS

- *Efficiency: number of observation needed*
- *Effectiveness: size of the resulting slice*

EVALUATION in terms of program slicing

	$\mathbb{O}$	$\mathbb{L}$	$\mathbb{B}$
1-HOT			
2-HOT			

VS

$\mathbb{E} \longrightarrow e_i$

ORBS

$\mathbb{E} \longrightarrow e_4$

Backward element

$e_3 \longrightarrow \mathbb{E}$

Forward dependency

RQ1. MOAD vs. ORBS

- *Efficiency: number of observation needed*
- *Effectiveness: size of the resulting slice*

RQ2. MOAD vs. Static slicing

- *Difference between slices*

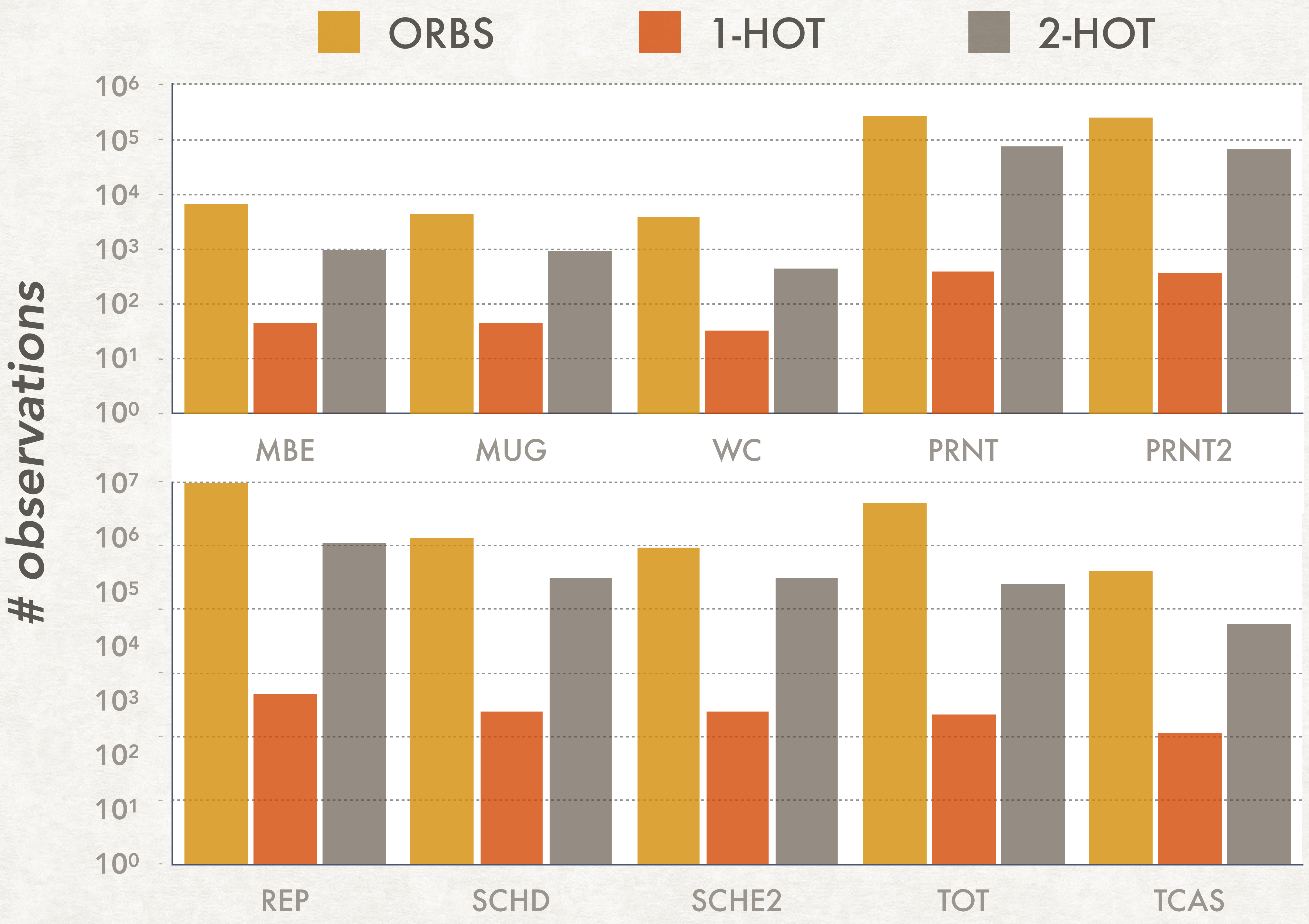
EVALUATION

- Subject

Subject	SLoC	# of statements	# of numeric variables
mbe *	64	45	16
mug *	61	44	13
wc *	46	33	17
print_tokens	410	388	98
print_tokens2	387	364	75
replace	508	465	253
schedule	283	252	75
schedule2	276	248	81
tot_info	314	227	210
tcas	152	110	62

Well-known examples where static analysis fails to identify dependency

RQ1: MOAD VS. ORBS



EFFICIENCY

MOAD with 1-HOT used

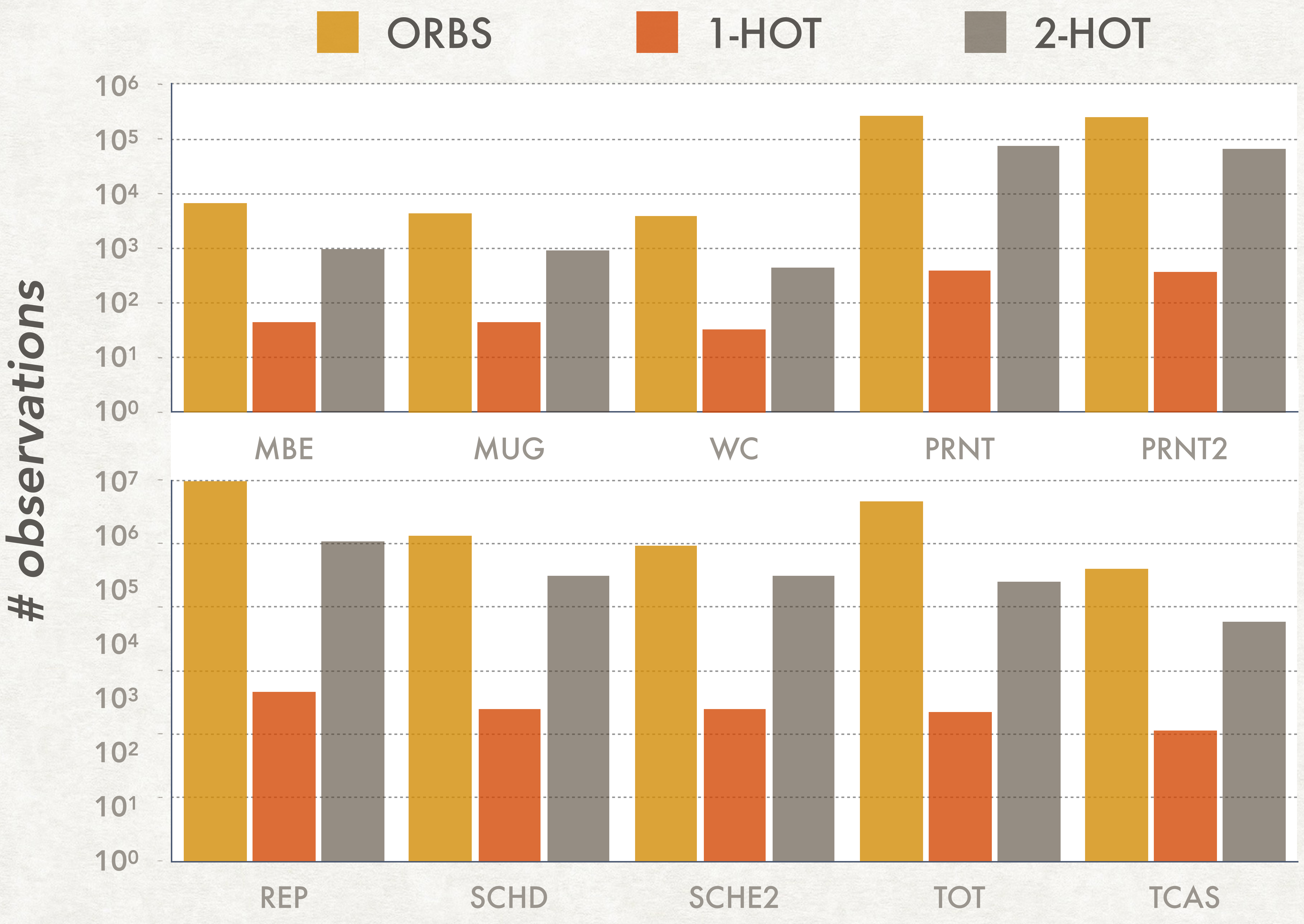
- **0.37%** of the # of observations

MOAD with 2-HOT used

- **18.6%** of the # of observations

compared to ORBS.

RQ1: MOAD VS. ORBS



EFFICIENCY

MOAD with 1-HOT used

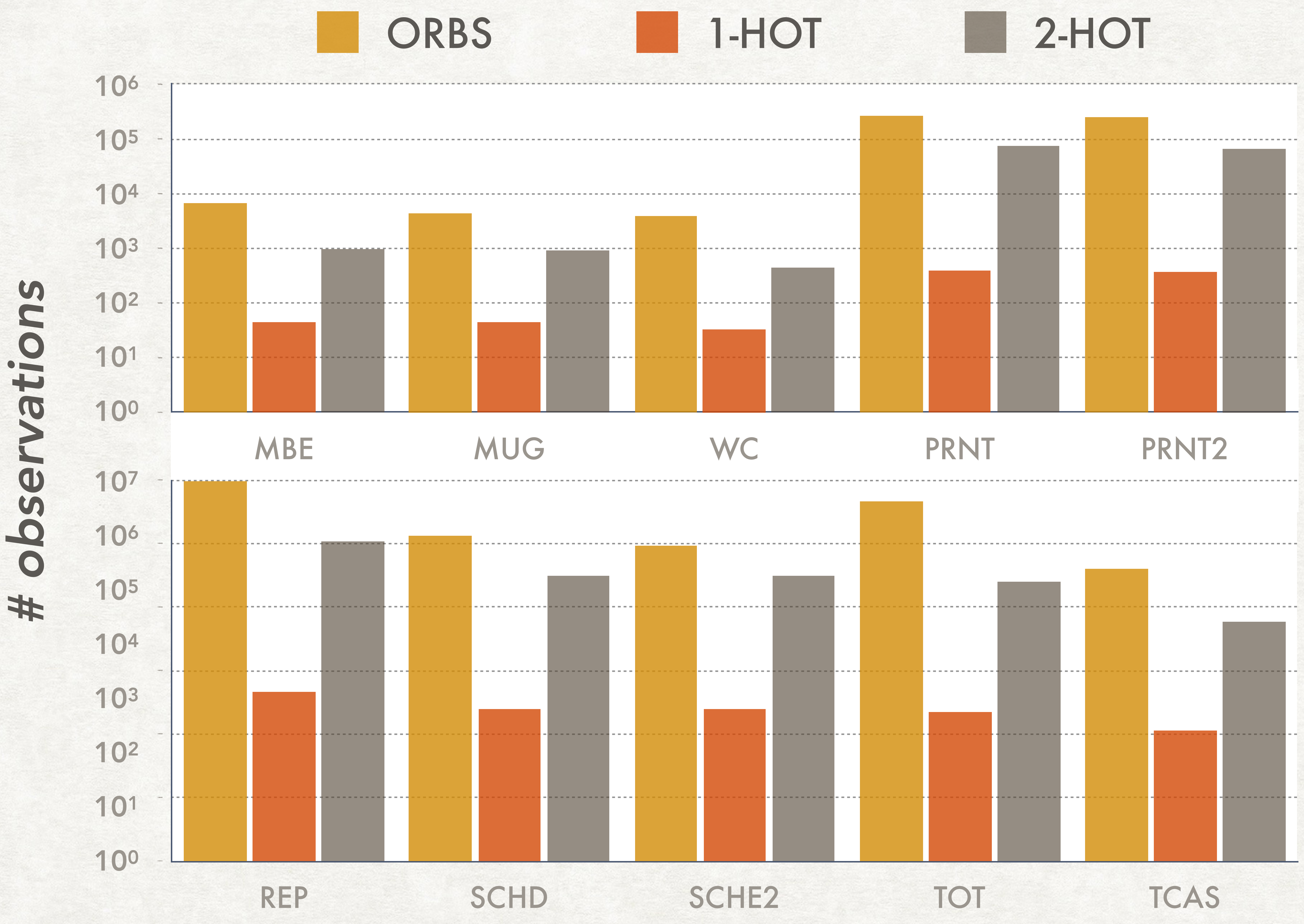
- **0.37%** of the # of observations

MOAD with 2-HOT used

- **18.6%** of the # of observations

compared to ORBS.

RQ1: MOAD VS. ORBS



EFFICIENCY

MOAD with 1-HOT used

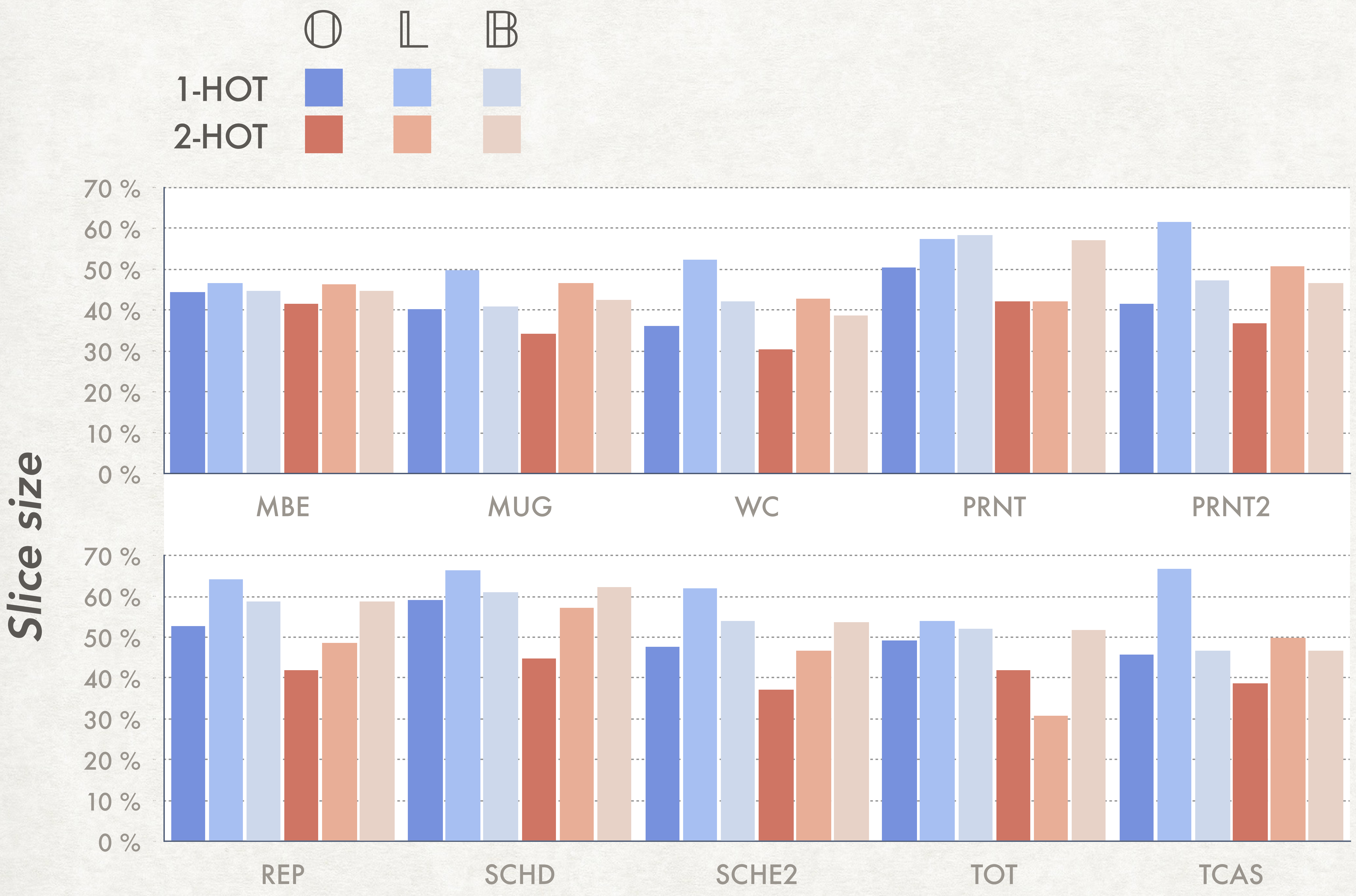
- **0.37%** of the # of observations

MOAD with 2-HOT used

- **18.6%** of the # of observations

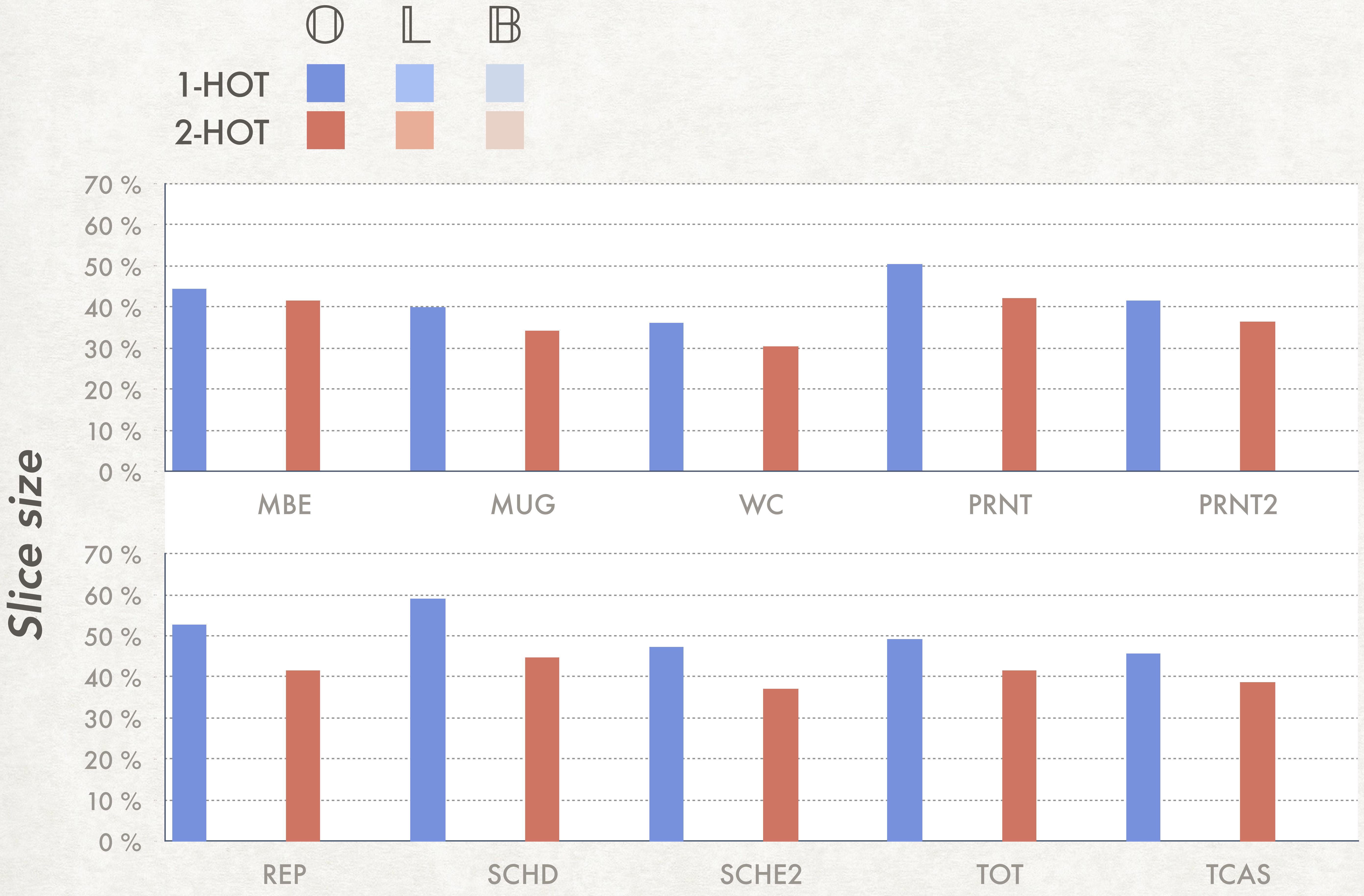
compared to ORBS.

RQ1: MOAD VS. ORBS



EFFECTIVENESS

RQ1: MOAD VS. ORBS

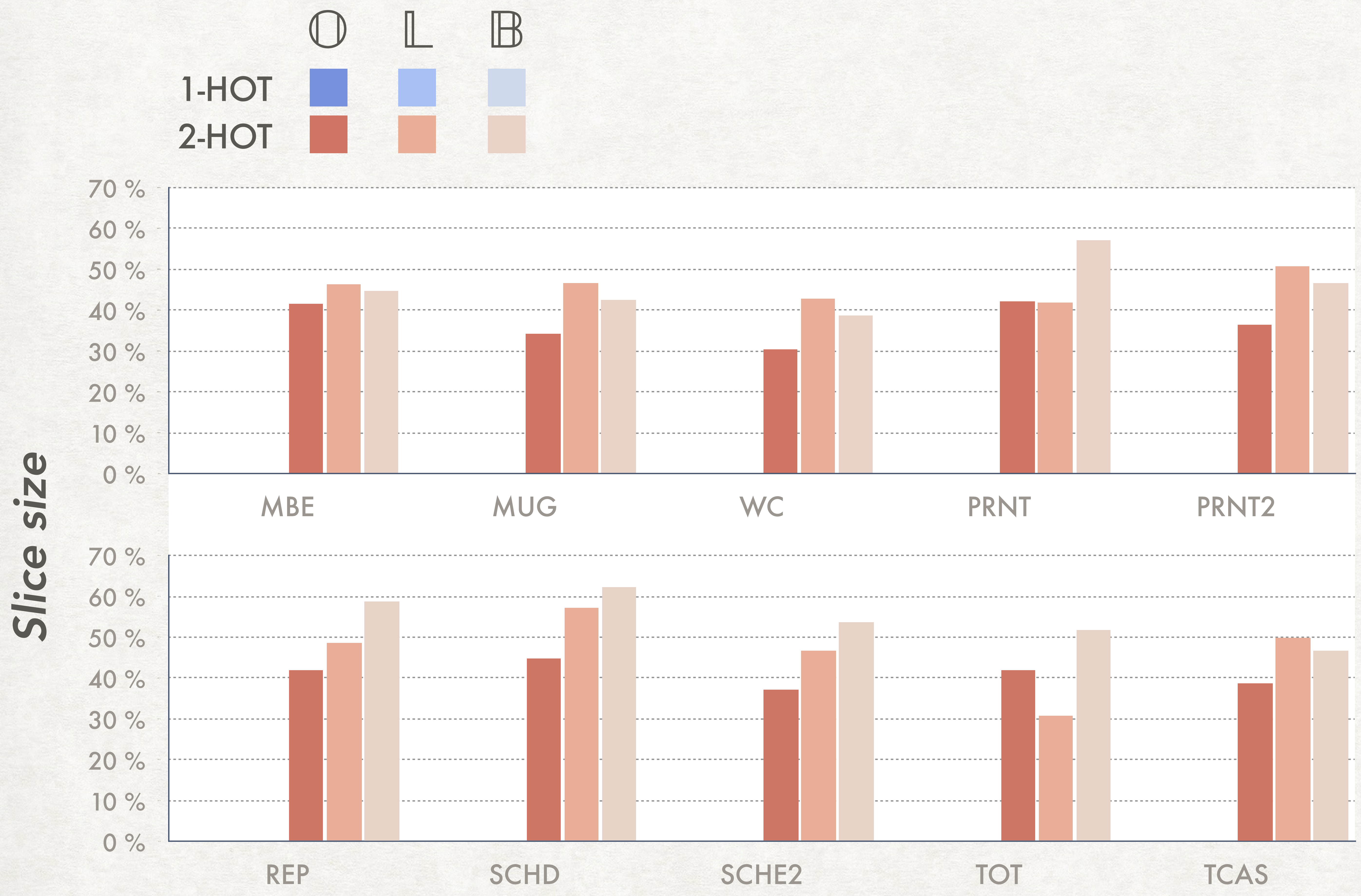


EFFECTIVENESS

For deletion generation scheme,

- 2-HOT < 1-HOT.

RQ1: MOAD VS. ORBS



EFFECTIVENESS

For deletion generation scheme,

- 2-HOT < 1-HOT.

For inference model,

- 𝕆 < 𝕃, 𝔹

RQ1: MOAD VS. ORBS

EFFECTIVENESS

For deletion generation scheme,

- 2-HOT < 1-HOT.

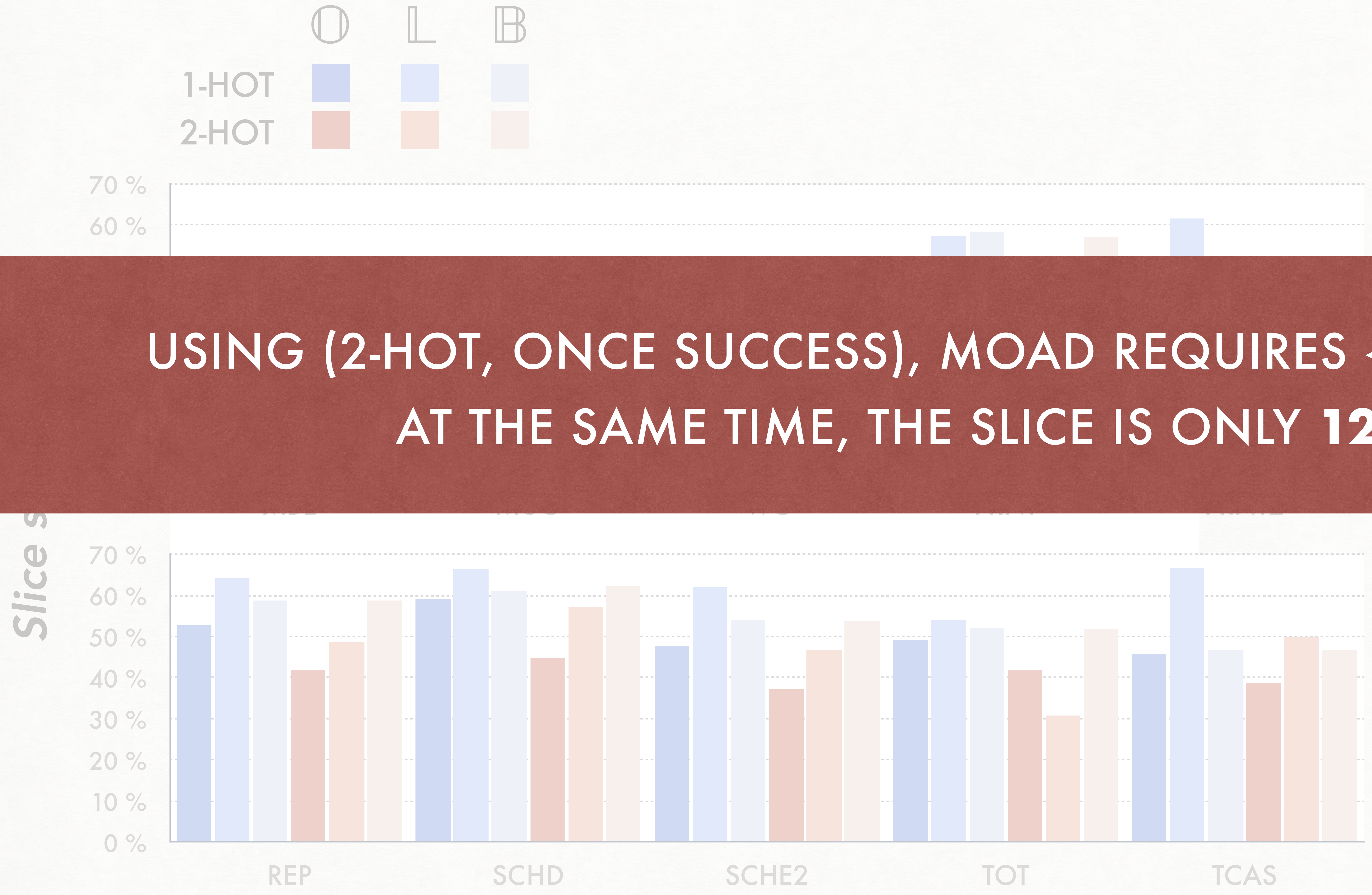
For inference model,

- 𝕆 < 𝕃, 𝔹

MOAD with 2-HOT, 𝕆 generate

- **12%** larger slices compared to ORBS.

RQ1: MOAD VS. ORBS



EFFECTIVENESS

For deletion generation scheme,

USING (2-HOT, ONCE SUCCESS), MOAD REQUIRES < 20% OBSERVATIONS THAN ORBS.

AT THE SAME TIME, THE SLICE IS ONLY 12% LARGER THAN ORBS.

MOAD with 2-HOT, 𝕆 generate

- **12%** larger slices compared to ORBS.

RQ2: MOAD VS. STATIC SLICER

- Static analysis tool: **CodeSurfer** from Grammatech
  - *Miss*: **# of lines only in the MOAD SLICE**
  - *Excess*: **# of lines only in the STATIC SLICE**

# of Lines (min-max)	**Miss**		**Excess**	
	3 small	*Siemens*	*3 small*	*Siemens*
Backward	0-3	8-24	0-1	9-79
Forward	0-0	0-6	0-1	7-37

RQ2: MOAD VS. STATIC SLICER

- Static analysis tool: **CodeSurfer** from Grammatech
  - *Miss*: # of lines only in the MOAD SLICE
  - *Excess*: # of lines only in the STATIC SLICE

# of Lines (min-max)	Miss		Excess	
	3 small	*Siemens*	*3 small*	*Siemens*
Backward	0–3	8–24	0–1	9–79
Forward	0–0	0–6	0–1	7–37

Keeping Declaration

Compilable Slice

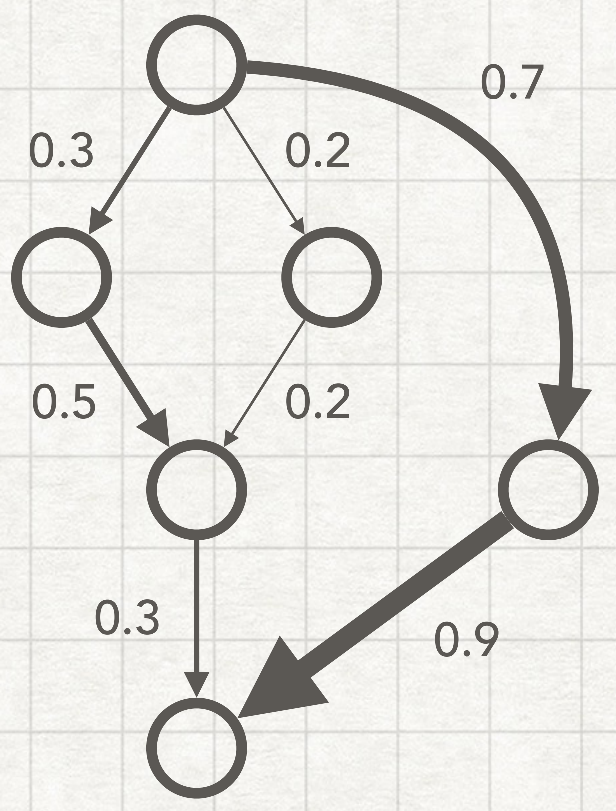
Segmentation Fault

Missing Initialization

Missing Return

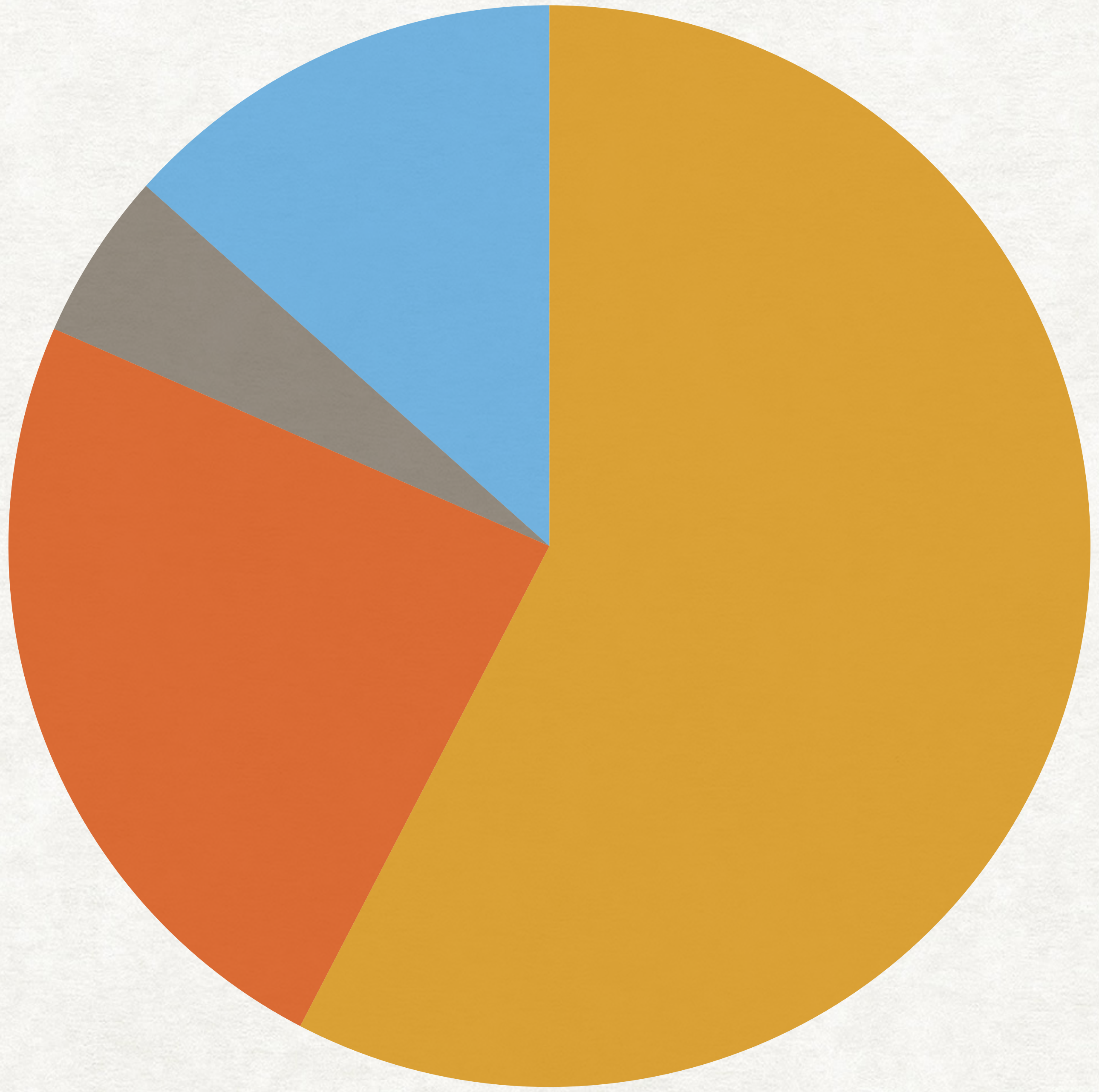
Limit of Static Analysis

CPDA



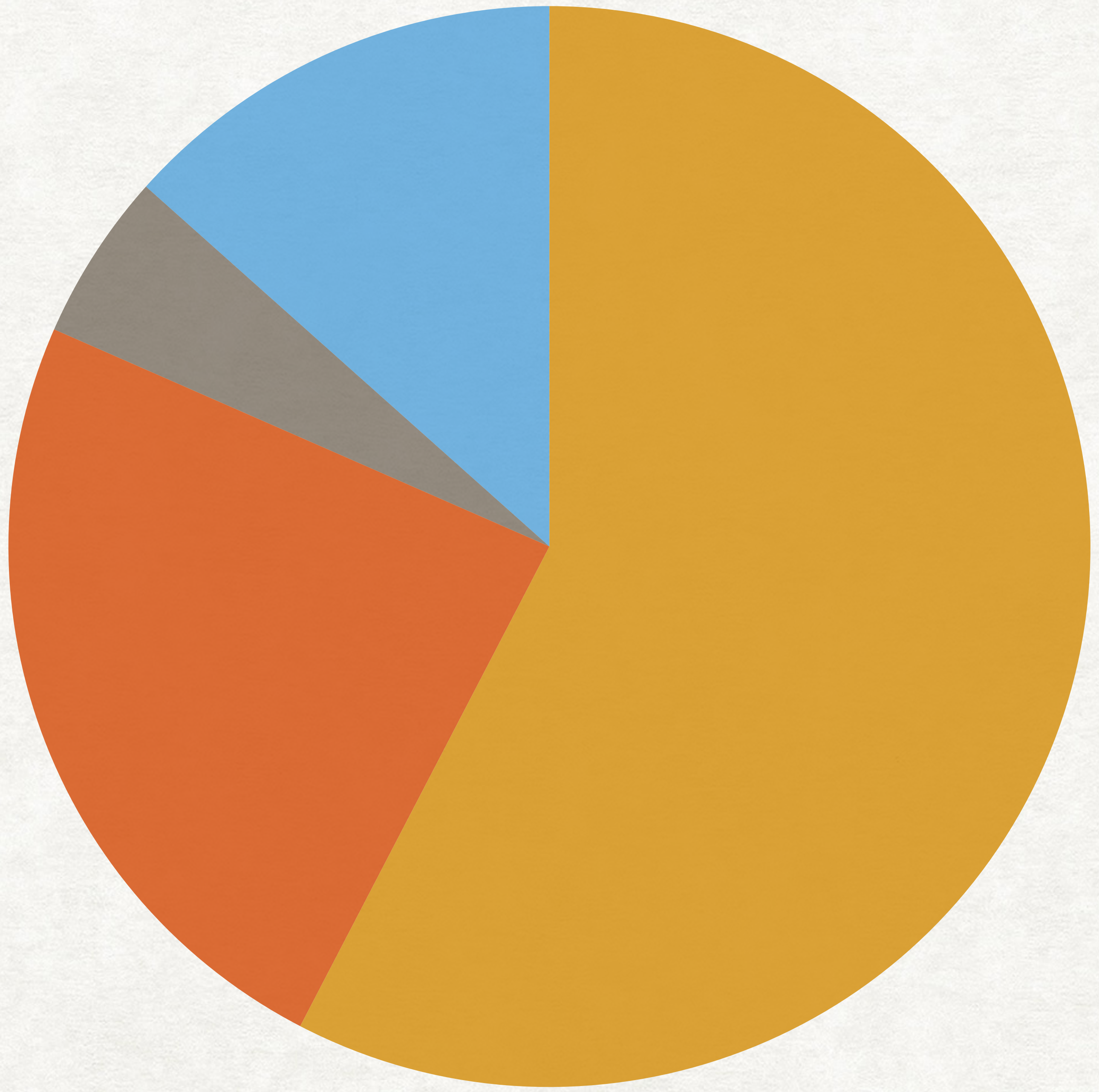
[*Causal inference on the observation data could estimates the degree of dependence between program elements*]

MOTIVATION



Program comprehension takes more than half of the time during software development and maintenance[1]

MOTIVATION



Program comprehension takes more than half of the time during software development and maintenance[1]

• • •

Programmers who used the systematic strategy gathered *knowledge about the causal interaction of the program's functional components.*[2]

• • •

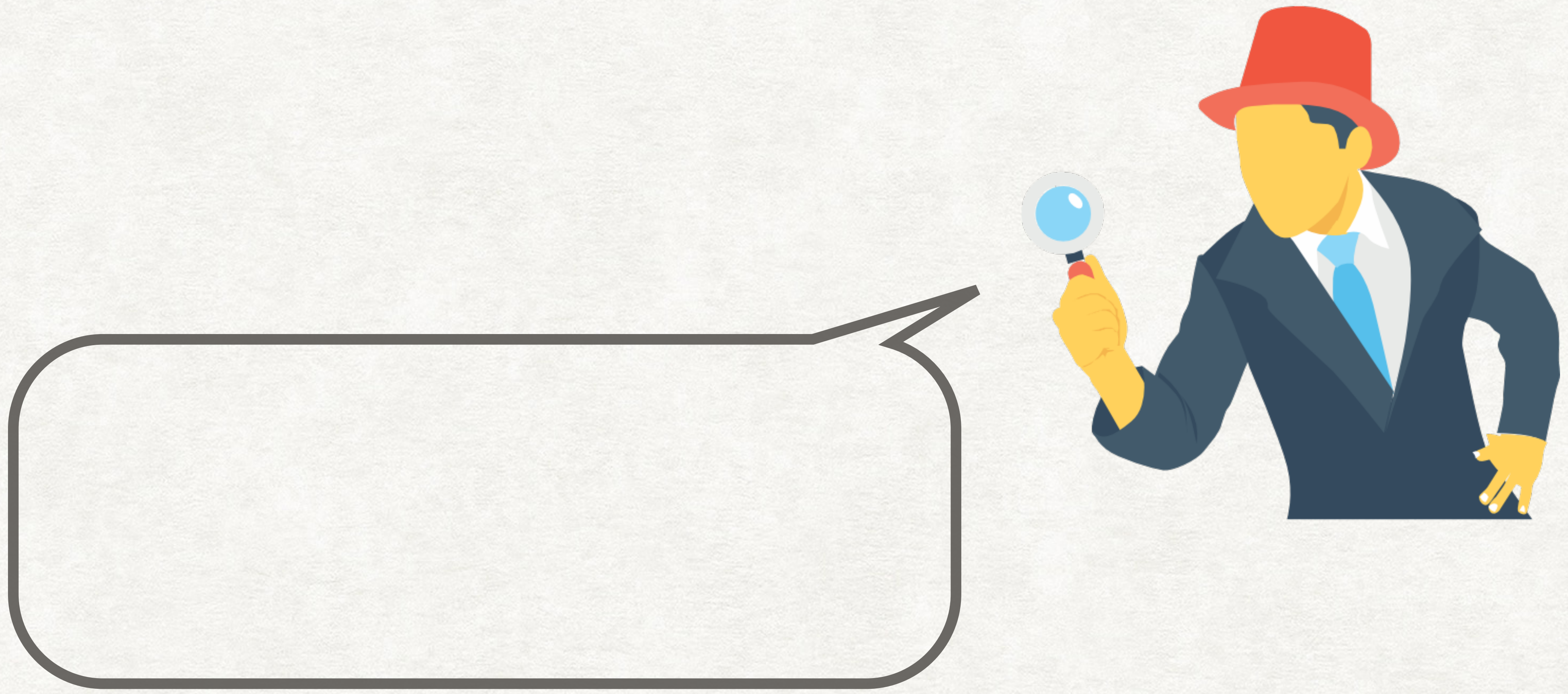
There is a strong relationship between using the systematic approach and *modifying the program successfully.*[2]

• • •

[2] "Program understanding: Challenge for the 1990s," *T. A. Corbi, IBM Systems Journal, 1989*

MOTIVATION

```
a = 42
pred = input() // 0 or 1
b = a + 1
if (pred):
    c = 2a
    d = c - 1
e = 3
```



Static analysis

MOTIVATION

```
a = 42
pred = input() // 0 or 1
b = a + 1
if (pred):
    c = 2a
    d = c - 1
e = 3
```

Static analysis

MOTIVATION

```
1: a = 42
2: pred = input() // 0 or 1
3: b = a + 1
4: if (pred):
5:     c = 2a
6:     d = c - 1
7: e = 3
```

Static analysis

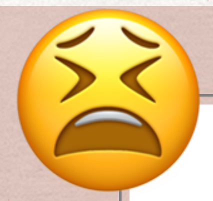
Dep(a → b) ~~>~~ *Dep*(b → c)

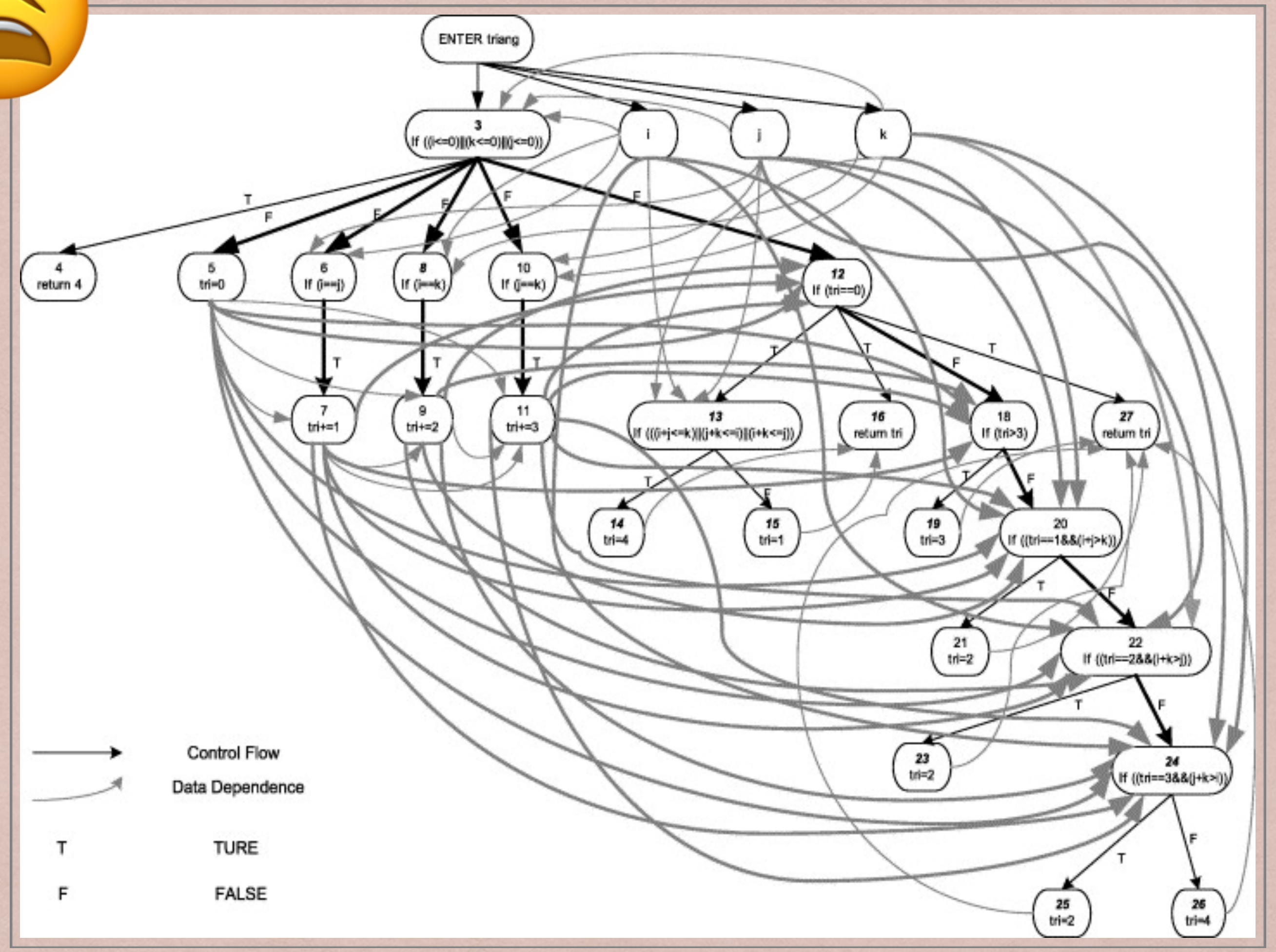
MOTIVATION

```
a = 42
pred = input() // 0 or 1
b = a + 1
if (pred):
    c = 2a
    d = c - 1
e = 3
```

Static analysis

Dep(a → b) $\cancel{>}$ *Dep*(b → c)





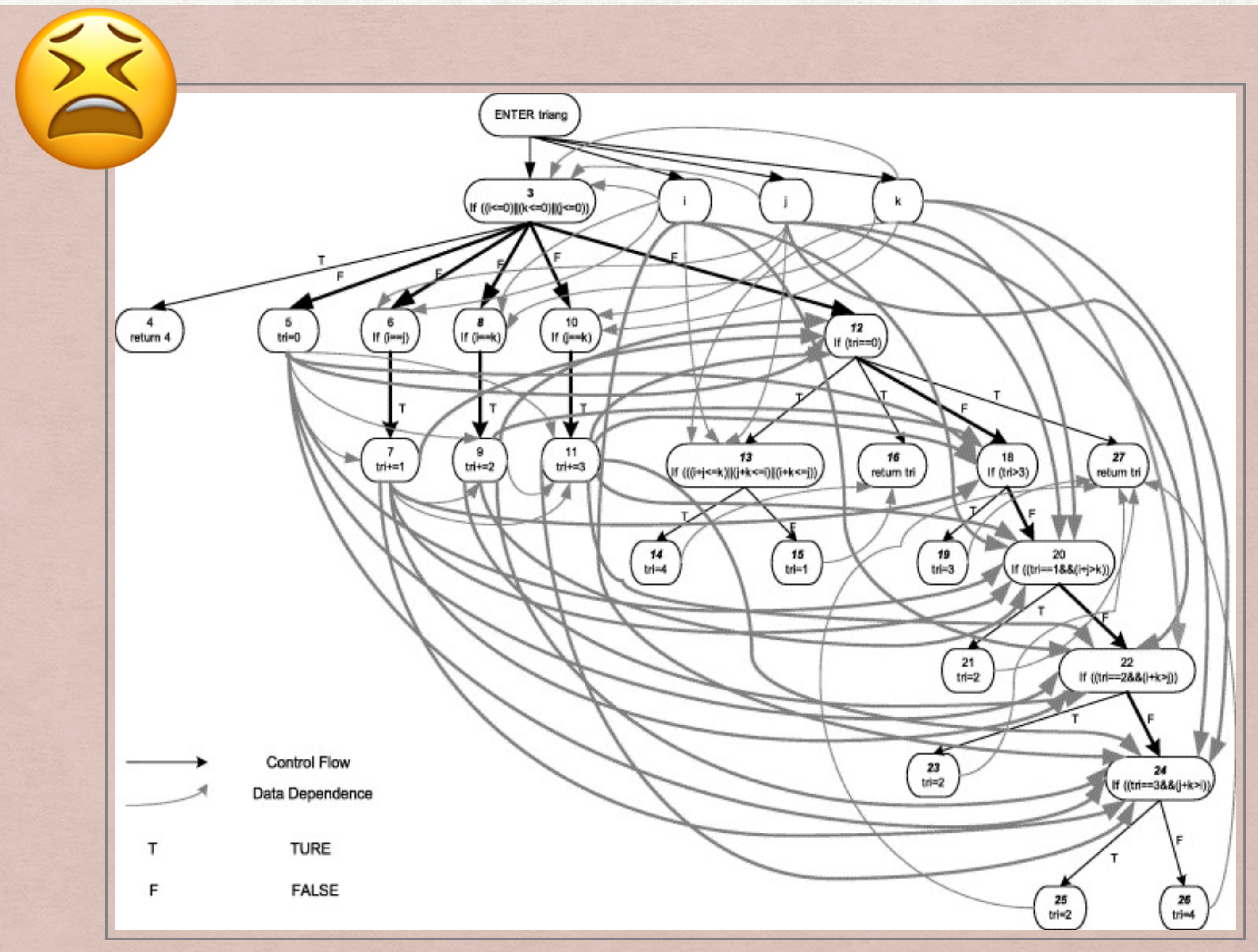
Program dependence graph (PDG)

MOTIVATION

```
a = 42
pred = input() // 0 or 1
b = a + 1
if (pred):
    c = 2a
    d = c - 1
e = 3
```

Static analysis & ORBS

$Dep(a \to b) \cancel{>} Dep(b \to c)$



Program dependence graph (PDG)

MOTIVATION

```
1: a = 42
2: pred = input() // 0 or 1
3: b = a + 1
4: if (pred):
5:     c = 2a
6:     d = c - 1
7: e = 3
```

MOAD

- P(b | "S1: a = 42") = 1.0
- P(c | "S3: b = a + 1") = 0.5
- P(d | "S5: c = 2a") = 1.0

MOTIVATION

MOAD

```
1: a = 42
2: pred = input() // 0 or 1
3: b = a + 1
4: if (pred):
5:    c = 2a
6:    d = c - 1
7: e = 3
```

- P(b | “S1: a = 42”) = 1.0
- P(c | “S3: b = a + 1”) = 0.5
- P(d | “S5: c = 2a”) = 1.0

$\text{stmt}_i \longrightarrow \text{var}_k$

Domain ≠ Co-domain

MOTIVATION

```
1: a = 42
2: pred = input() // 0 or 1
3: b = a + 1
4: if (pred):
5:     c = 2a
6:     d = c - 1
7: e = 3
```

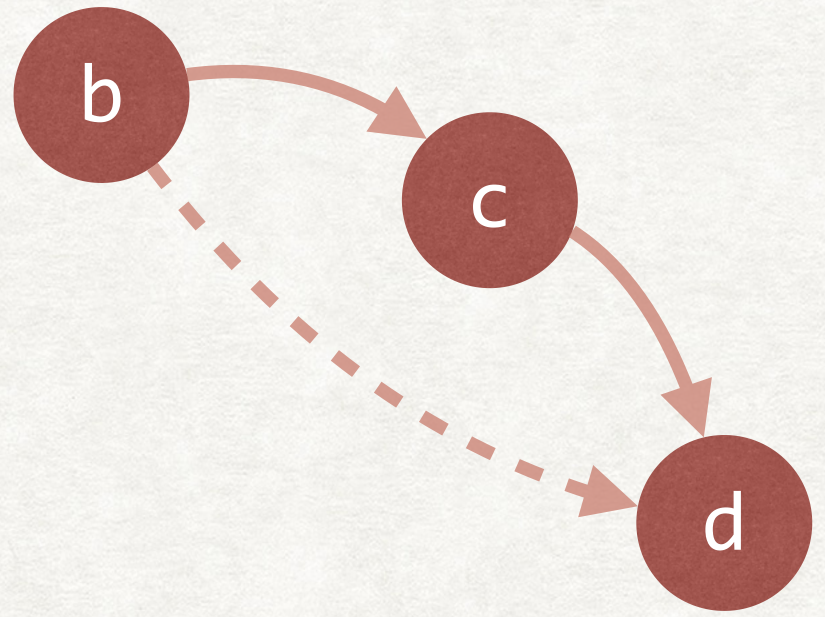
MOAD

- P(b | "S1: a = 42") = 1.0
- P(c | "S3: b = a + 1") = 0.5
- P(d | "S5: c = 2a") = 1.0

But, also

- P(d | "S3: b = a + 1") = 0.5

stmt$_i$ → var$_k$

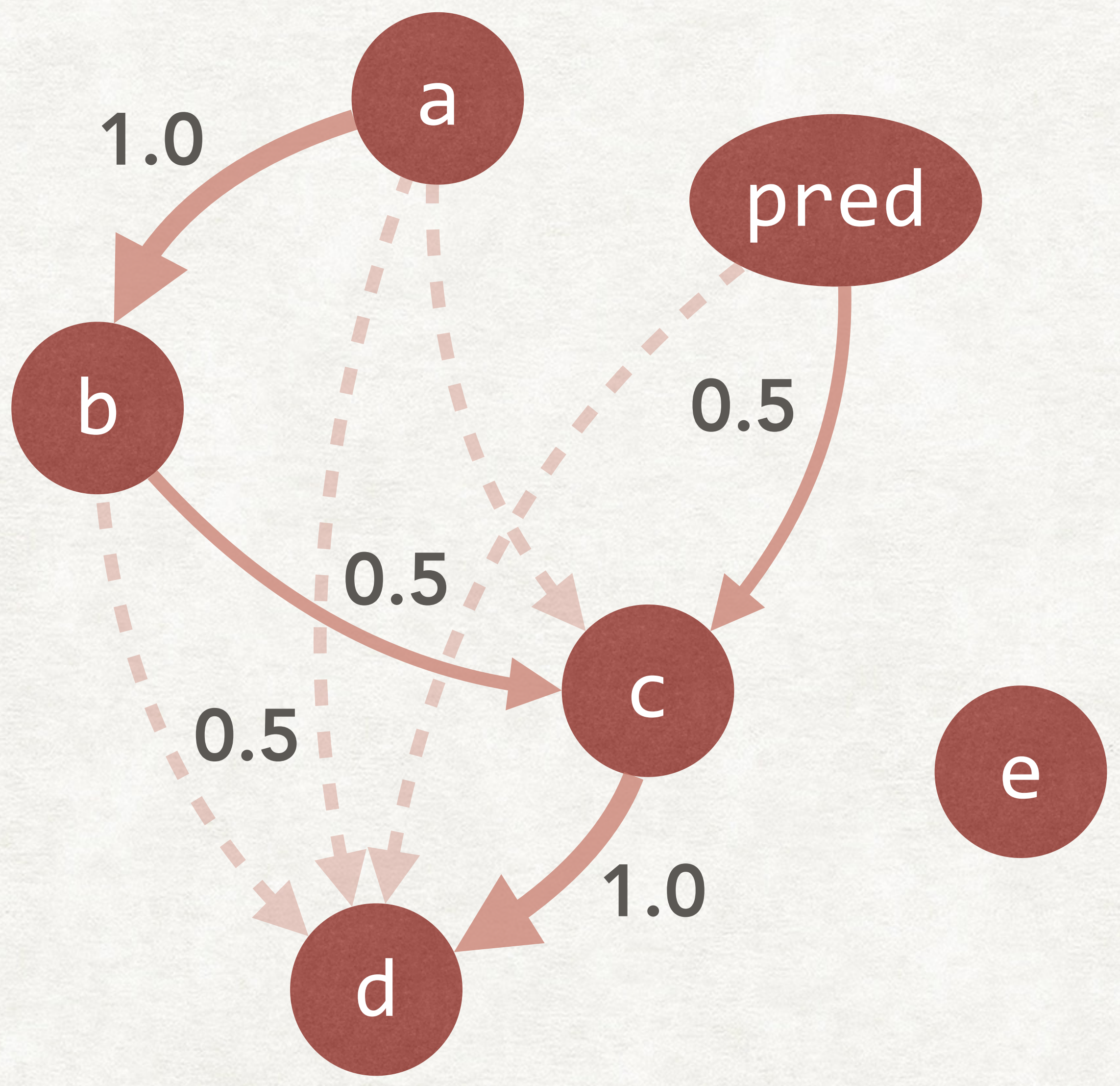


Domain ≠ Co-domain

No structural reasoning

GOAL: QUANTIFIABLE DEPENDENCE

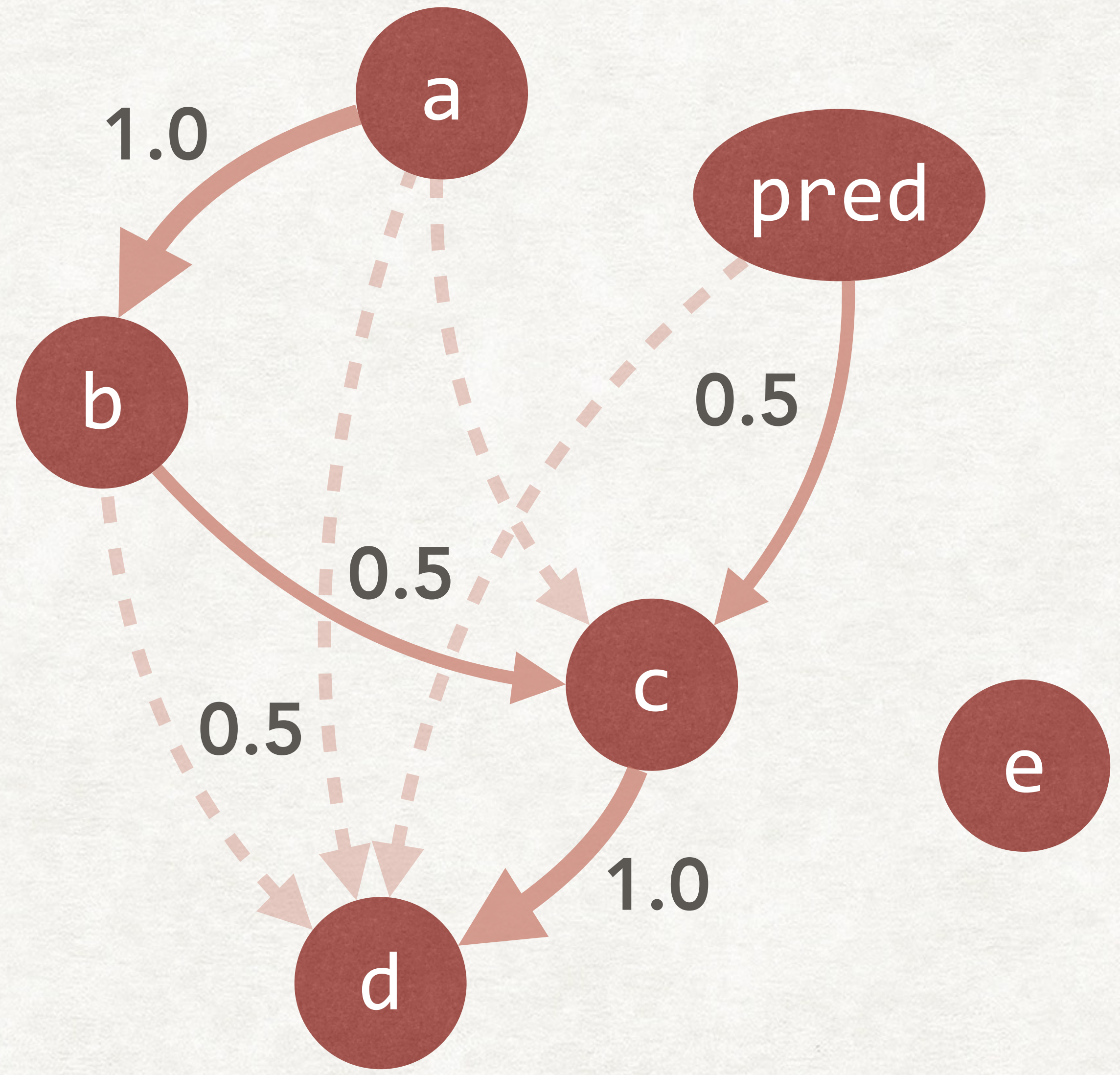
```
a = 42
pred = input() // 0 or 1
b = a + 1
if (pred):
    c = 2a
    d = c - 1
e = 3
```



We represent the **dependence structure** *with the* **degree of dependencies**, *which can aid the understanding/usefulness of the program dependence.*

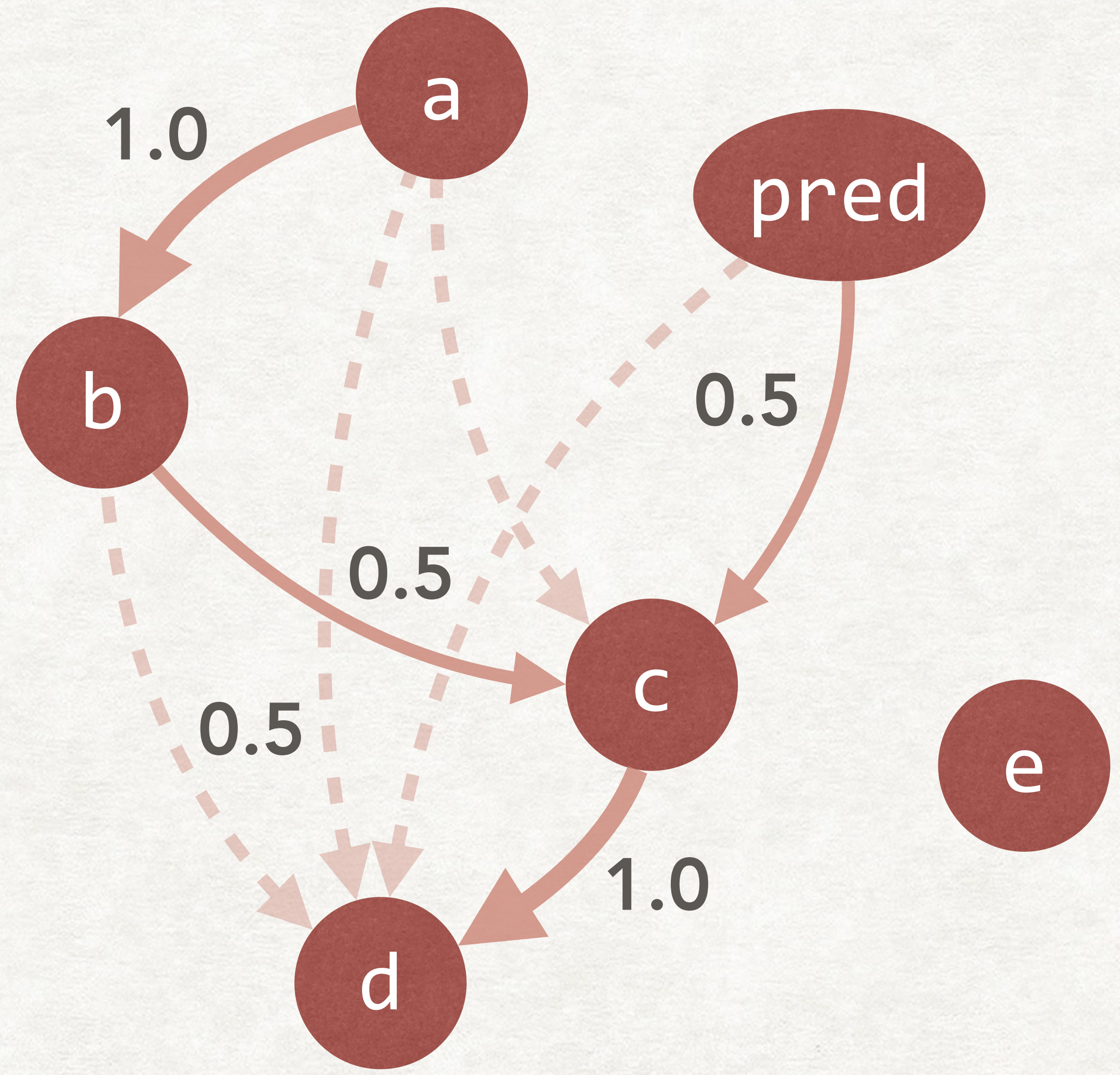
Observation data

Mutated	a	b	c	…	e
a	≠	≠	≠	…	=
b	=	≠	≠	…	=
…	…	…	…	…	…
e	=	=	=	…	≠



Observation data

Mutated	a	b	c	…	e
a	≠	≠	≠	…	=
b	=	≠	≠	…	=
…	…	…	…	…	…
e	=	=	=	…	≠



- Causal analysis -

CAUSAL ANALYSIS

Association data of events

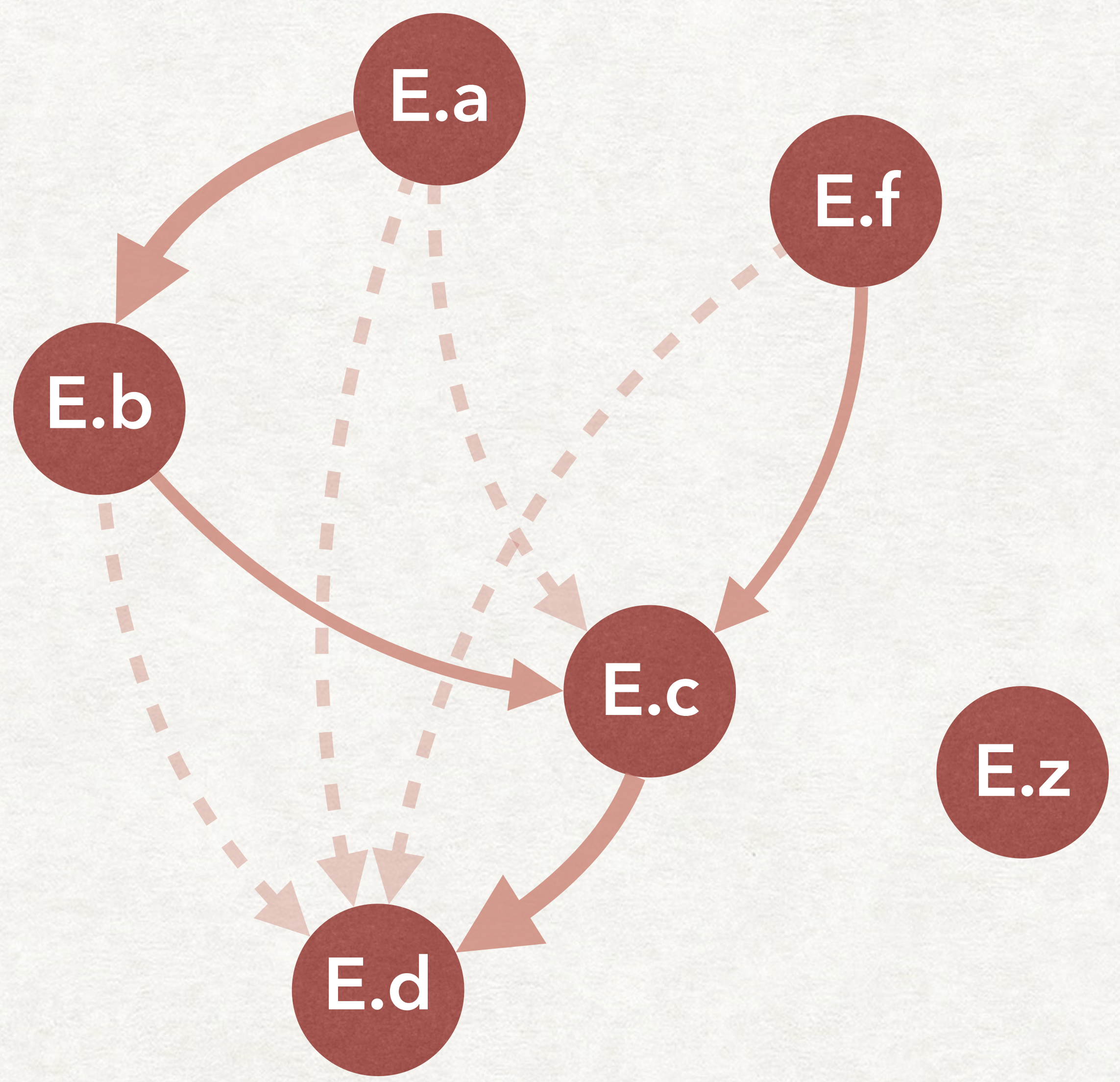
Index	Event a	Event b	Event c	…	Event z
1	O	O	O	…	-
2	-	O	O	…	-
…	…	…	…	…	…
N	-	-	-	…	O

CAUSAL ANALYSIS

1 Structure discovery

Association data of events

Index	Event a	Event b	Event c	…	Event z
1	O	O	O	…	-
2	-	O	O	…	-
…	…	…	…	…	…
N	-	-	-	…	O



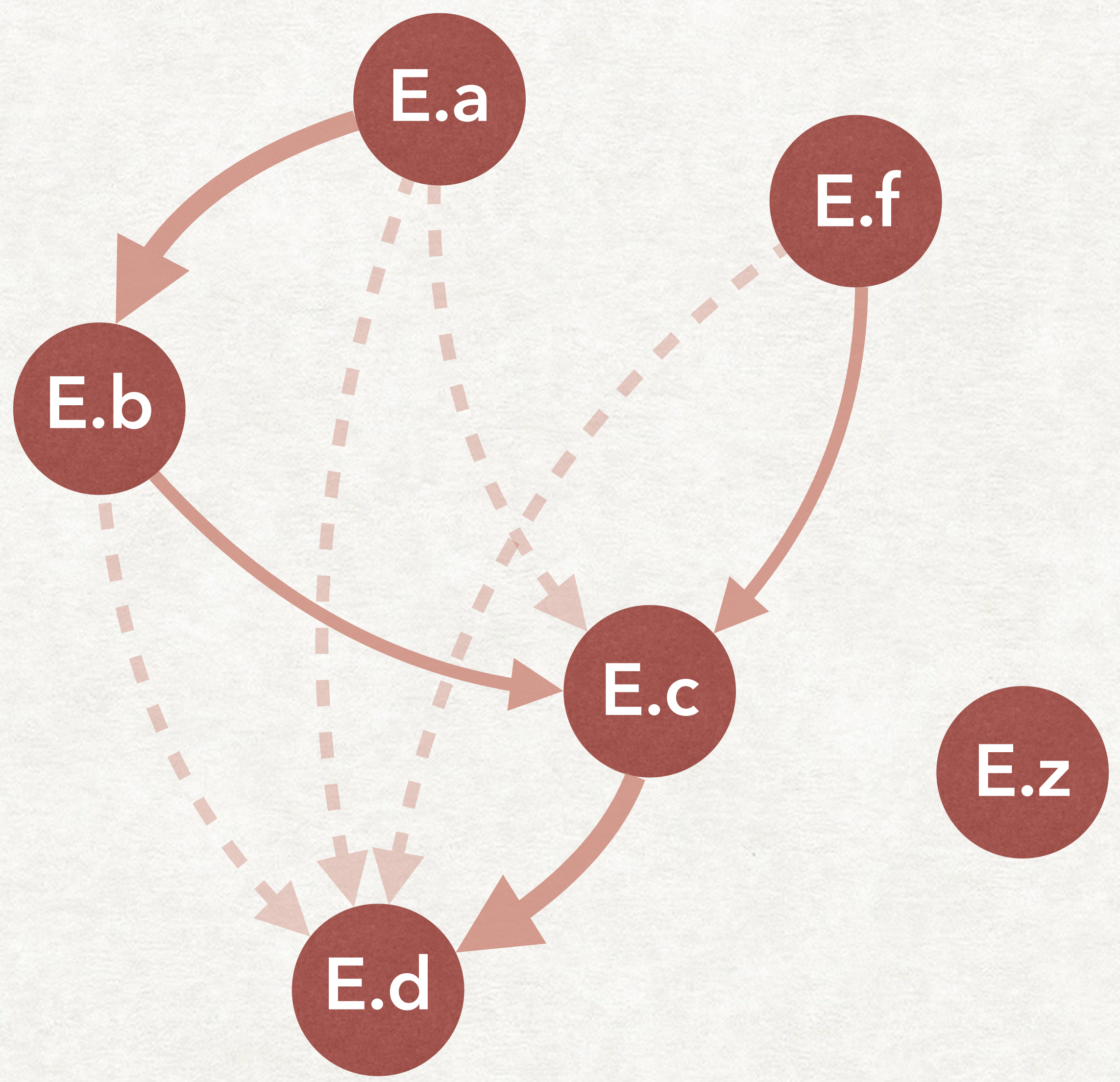
Identify directly affecting relations

CAUSAL ANALYSIS

Association data of events

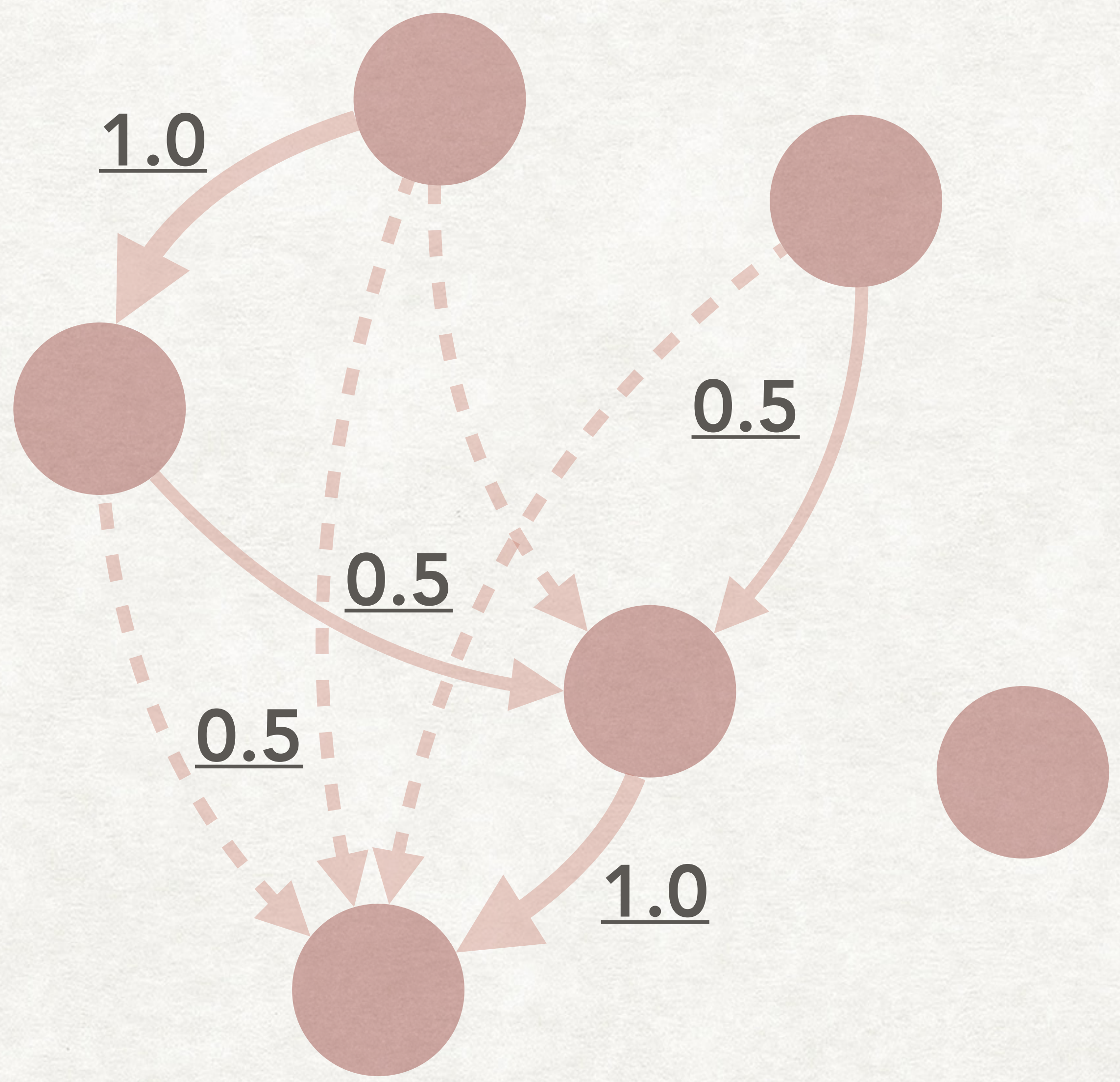
Index	Event a	Event b	Event c	…	Event z
1	O	O	O	…	-
2	-	O	O	…	-
…	…	…	…	…	…
N	-	-	-	…	O

1 Structure discovery



Identify directly affecting relations

2 Causal inference



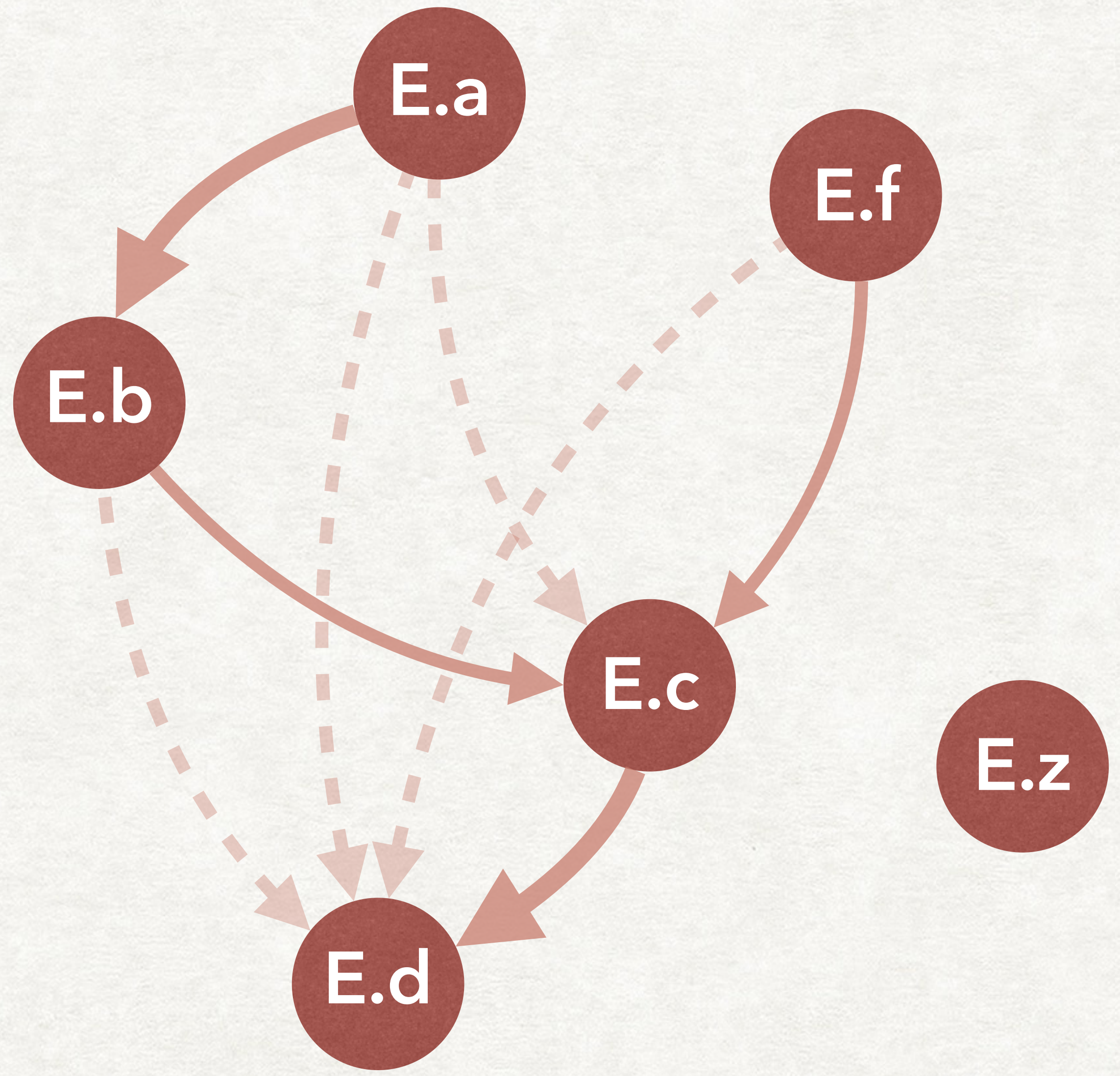
Estimate degree of causations

CAUSAL ANALYSIS

Event := program element's behavior change

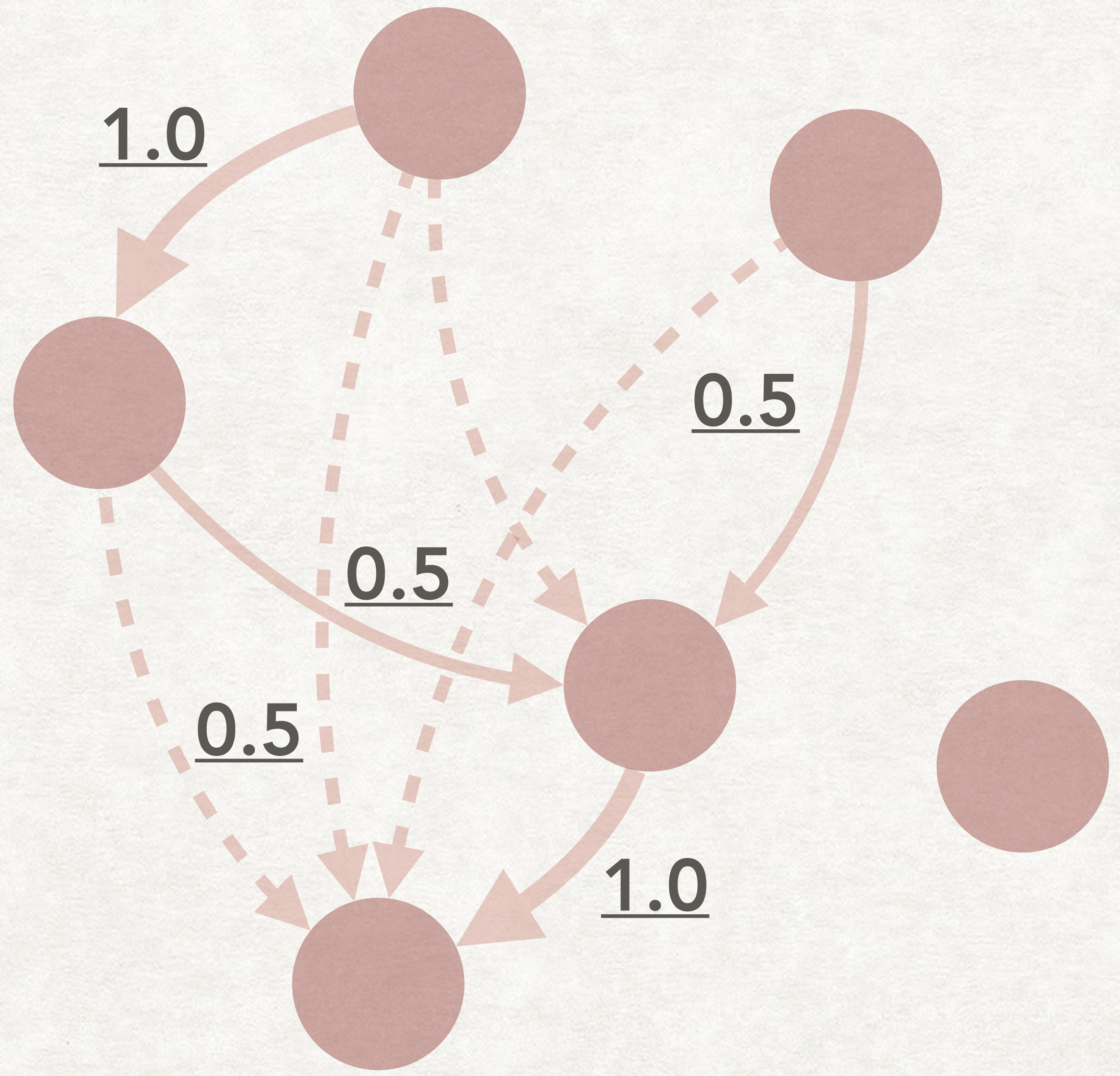
Mutated	a	b	c	…	f
a	≠	≠	≠	…	=
b	=	≠	≠	…	=
…	…	…	…	…	…
f	=	=	=	…	≠

1 Structure discovery



Identify directly affecting relations

2 Causal inference



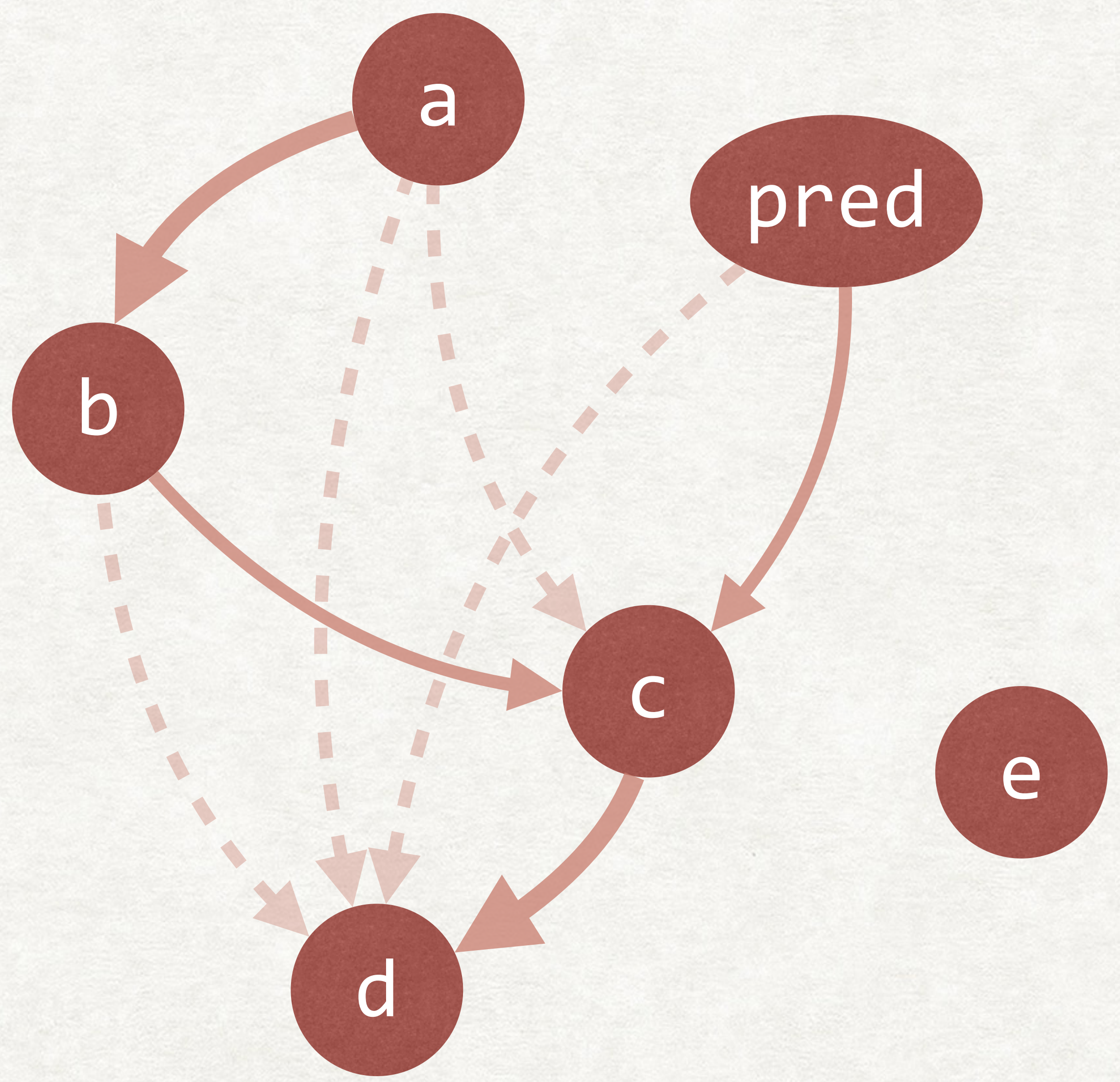
Estimate degree of causations

CAUSAL ANALYSIS

Event := program element's behavior change

Mutated	a	b	c	…	f
a	≠	≠	≠	…	=
b	=	≠	≠	…	=
…	…	…	…	…	…
f	=	=	=	…	≠

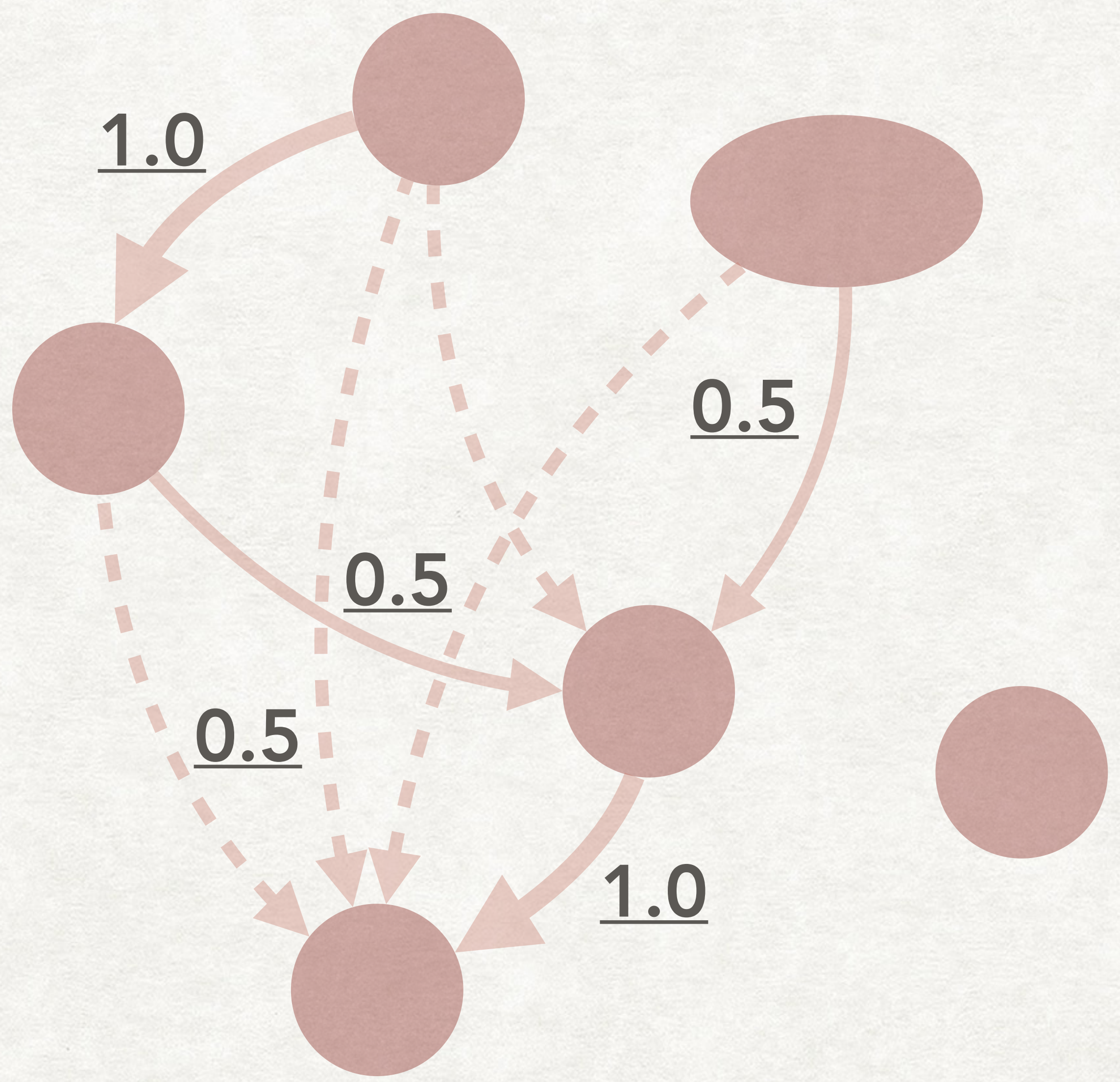
1 Structure discovery



Identify directly affecting relations

≈ Program Dependence Graph (PDG)

2 Causal inference



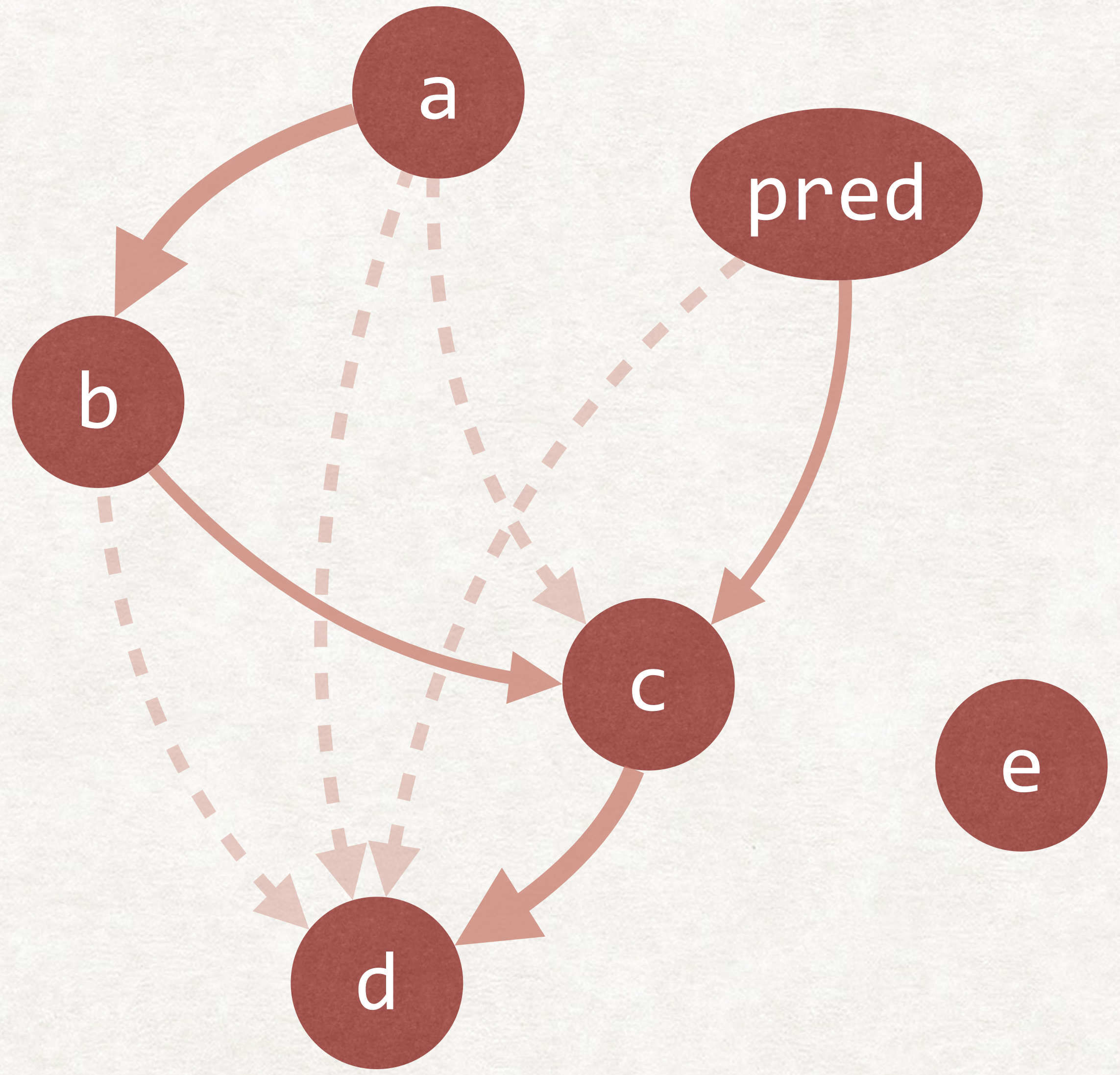
Estimate degree of causations

CAUSAL PROGRAM DEPENDENCE ANALYSIS (CPDA)

Event := program element's behavior change

Mutated	a	b	c	…	f
a	≠	≠	≠	…	=
b	=	≠	≠	…	=
…	…	…	…	…	…
f	=	=	=	…	≠

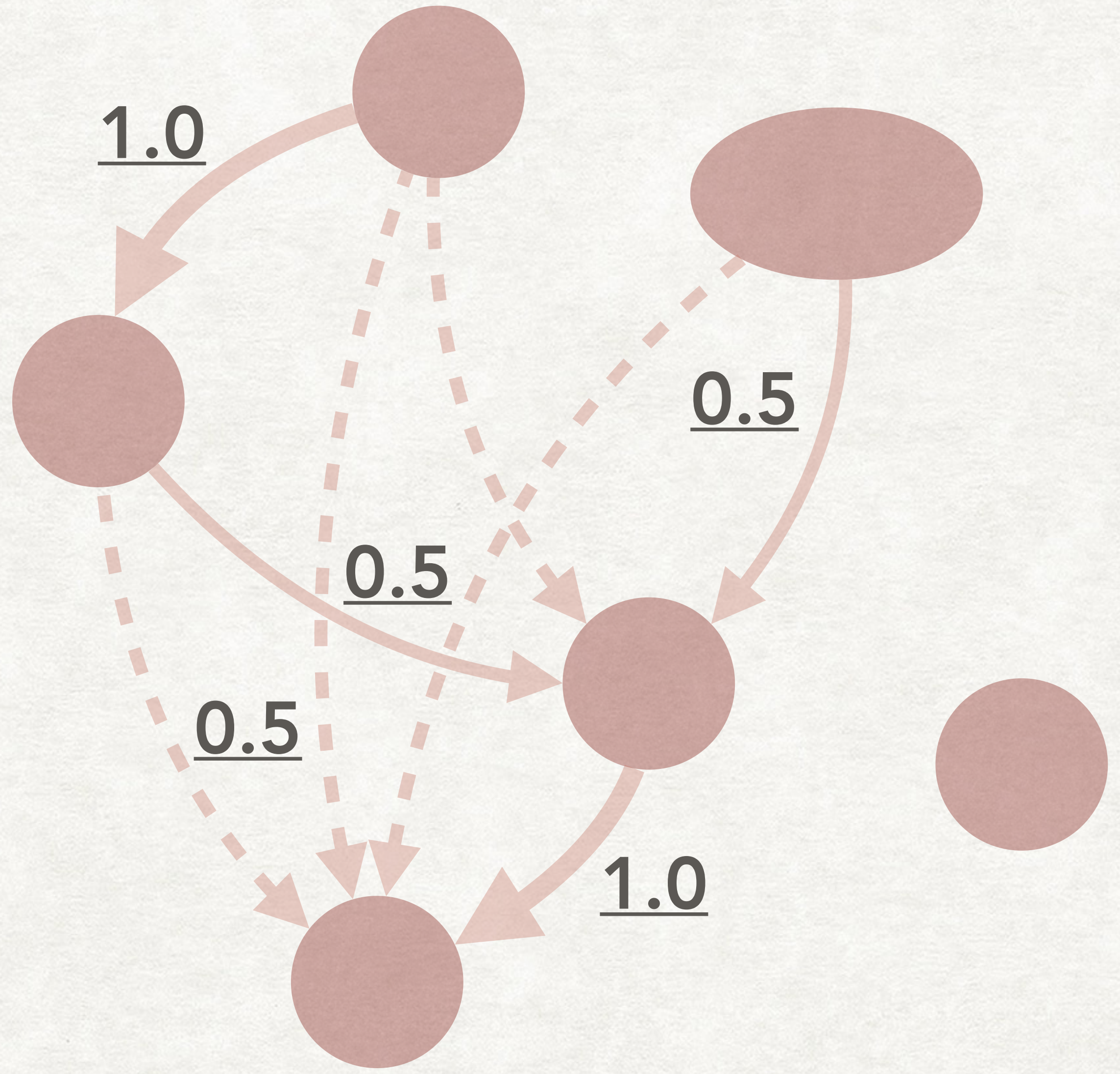
1 Structure discovery



Identify directly affecting relations

≈ Program Dependence Graph (PDG)

2 Causal inference



Estimate degree of causations

FINDING DEPENDENCE STRUCTURE

```
a = 42
pred = input() // 0 or 1
b = a + 1
if (pred):
    c = 2b
    d = c - 1
e = 3
```

Value mutation

Behavior is the same

Behavior changed

No mutation case →

Mut	a	pred	b	c	d	e
∅	=	=	=	=	=	=
a	≠	=	≠	≠/=	≠/=	=
pred	=	≠	=	≠	≠	=
b	=	=	≠	≠/=	≠/=	=
c	=	=	=	≠	≠	=
d	=	=	=	=	≠	=
e	=	=	=	=	=	≠

Observation data

When "pred" True (1) / False (0)

FINDING DEPENDENCE STRUCTURE

```
a = 42
pred = input() // 0 or 1
b = a + 1
if (pred):
    c = 2b
    d = c - 1
e = 3
```

Value mutation

Behavior is the same

Behavior changed

No mutation case →

Mut	a	pred	b	c	d	e
∅	=	=	=	=	=	=
a	≠	=	≠	≠/=	≠/=	=
pred	=	≠	=	≠	≠	=
b	=	=	≠	≠/=	≠/=	=
c	=	=	=	≠	≠	=
d	=	=	=	=	≠	=
e	=	=	=	=	=	≠

Observation data

When “pred” True (1) / False (0)

FINDING DEPENDENCE STRUCTURE

```
1: a = 42
2: pred = input() // 0 or 1
3: b = a + 1
4: if (pred):
5:     c = 2b
6:     d = c - 1
7: e = 3
```

Value mutation

Behavior is the same

Behavior changed

Mut	a	pred	b	c	d	e
No mutation case → ∅	=	=	=	=	=	=
a	≠	=	≠	≠/=	≠/=	=
pred	=	≠	=	≠	≠	=
b	=	=	≠	≠/=	≠/=	=
c	=	=	=	≠	≠	=
d	=	=	=	=	≠	=
e	=	=	=	=	=	≠

Observation data

When "pred" True (1) / False (0)

FINDING DEPENDENCE STRUCTURE

```
1: a = 42
2: pred = input() // 0 or 1
3: b = a + 1
4: if (pred):
5:     c = 2b
6:     d = c - 1
7: e = 3
```

Mut	a	pred	b	c	d	e
∅	=	=	=	=	=	=
a	≠	=	≠	≠/=	≠/=	=
pred	=	≠	=	≠	≠	=
b	=	=	≠	≠/=	≠/=	=
c	=	=	=	≠	≠	=
d	=	=	=	=	≠	=
e	=	=	=	=	=	≠

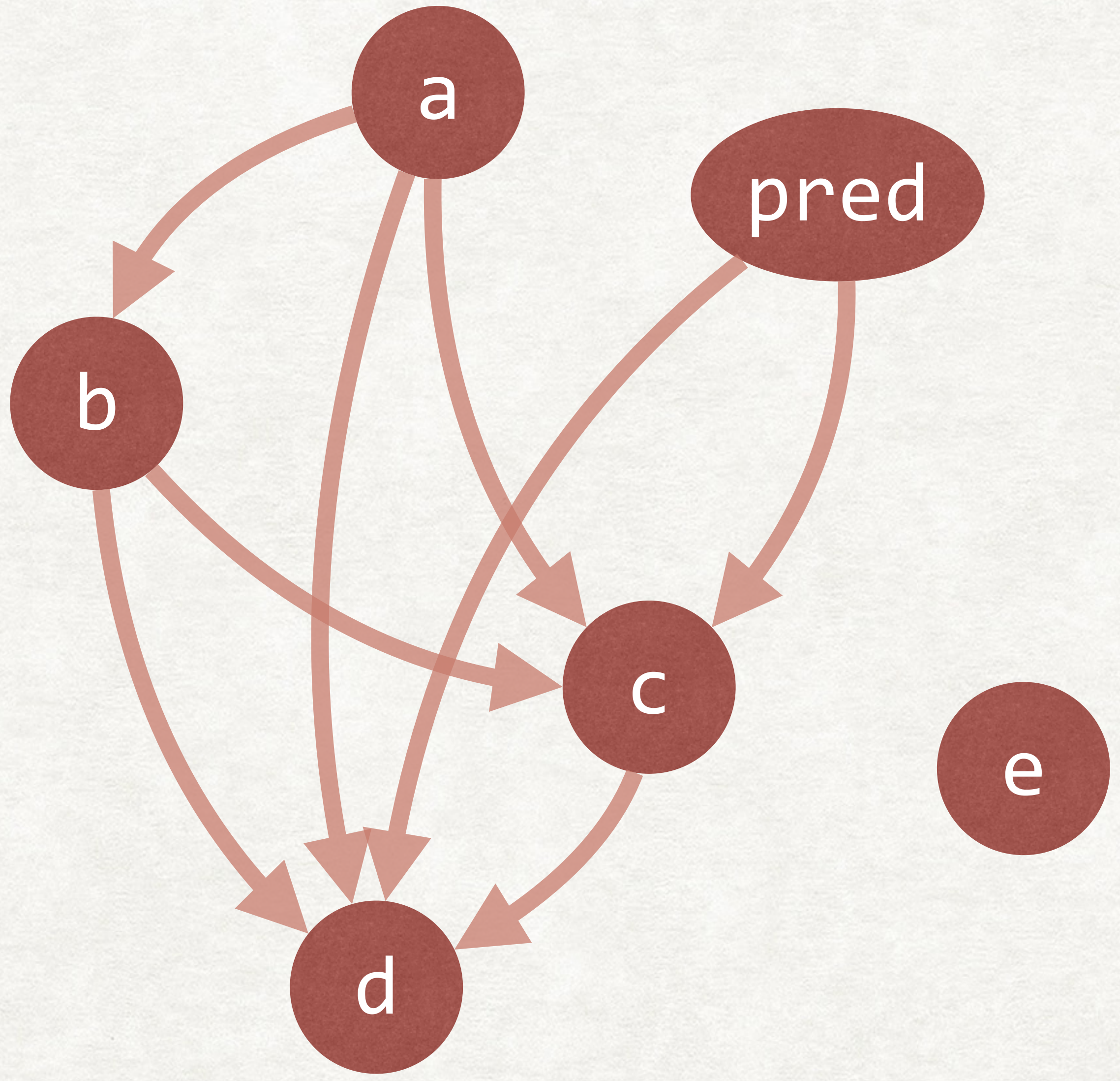
FINDING DEPENDENCE STRUCTURE

```
a = 42
pred = input() // 0 or 1
b = a + 1
if (pred):
    c = 2b
    d = c - 1
e = 3
```

Mut	a	pred	b	c	d	e
∅	=	=	=	=	=	=
a	≠	=	≠	≠/=	≠/=	=
pred	=	≠	=	≠	≠	=
b	=	=	≠	≠/=	≠/=	=
c	=	=	=	≠	≠	=
d	=	=	=	=	≠	=
e	=	=	=	=	=	≠

- MOAD -

```
   a: {}
pred: {}
   b: {a}
   c: {a, pred, b}
   d: {a, pred, b, c}
   e: {}
```



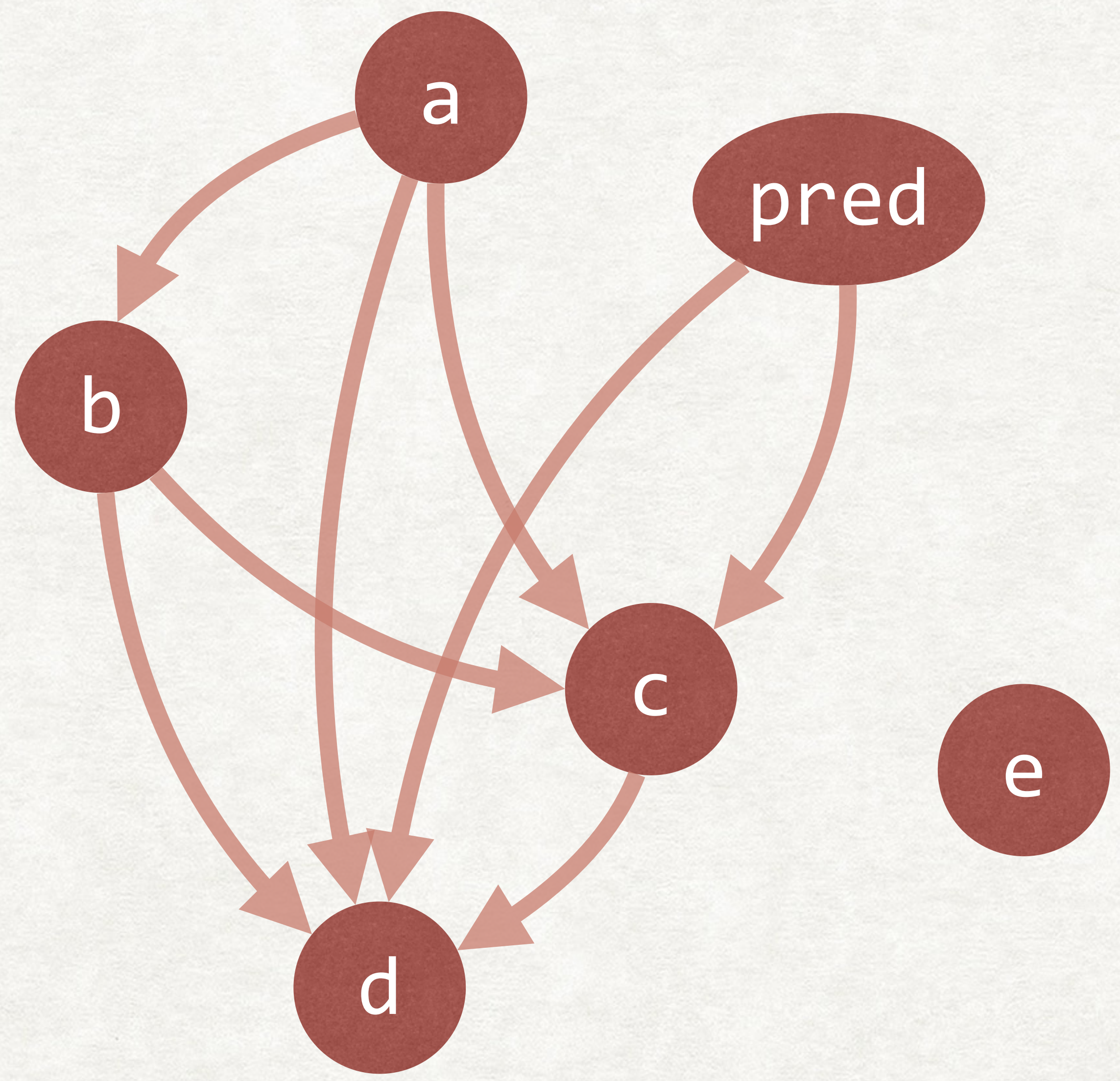
FINDING DEPENDENCE STRUCTURE

```
a = 42
pred = input() // 0 or 1
b = a + 1
if (pred):
    c = 2b
    d = c - 1
e = 3
```

- MOAD -

Mut	a	pred	b	c	d	e
∅	=	=	=	=	=	=
a	≠	=	≠	≠/=	≠/=	=
pred	=	≠	=	≠	≠	=
b	=	=	≠	≠/=	≠/=	=
c	=	=	=	≠	≠	=
d	=	=	=	=	≠	=
e	=	=	=	=	=	≠

```
   a: {}
pred: {}
   b: {a}
   c: {a, pred, b}
   d: {a, pred, b, c}
   e: {}
```



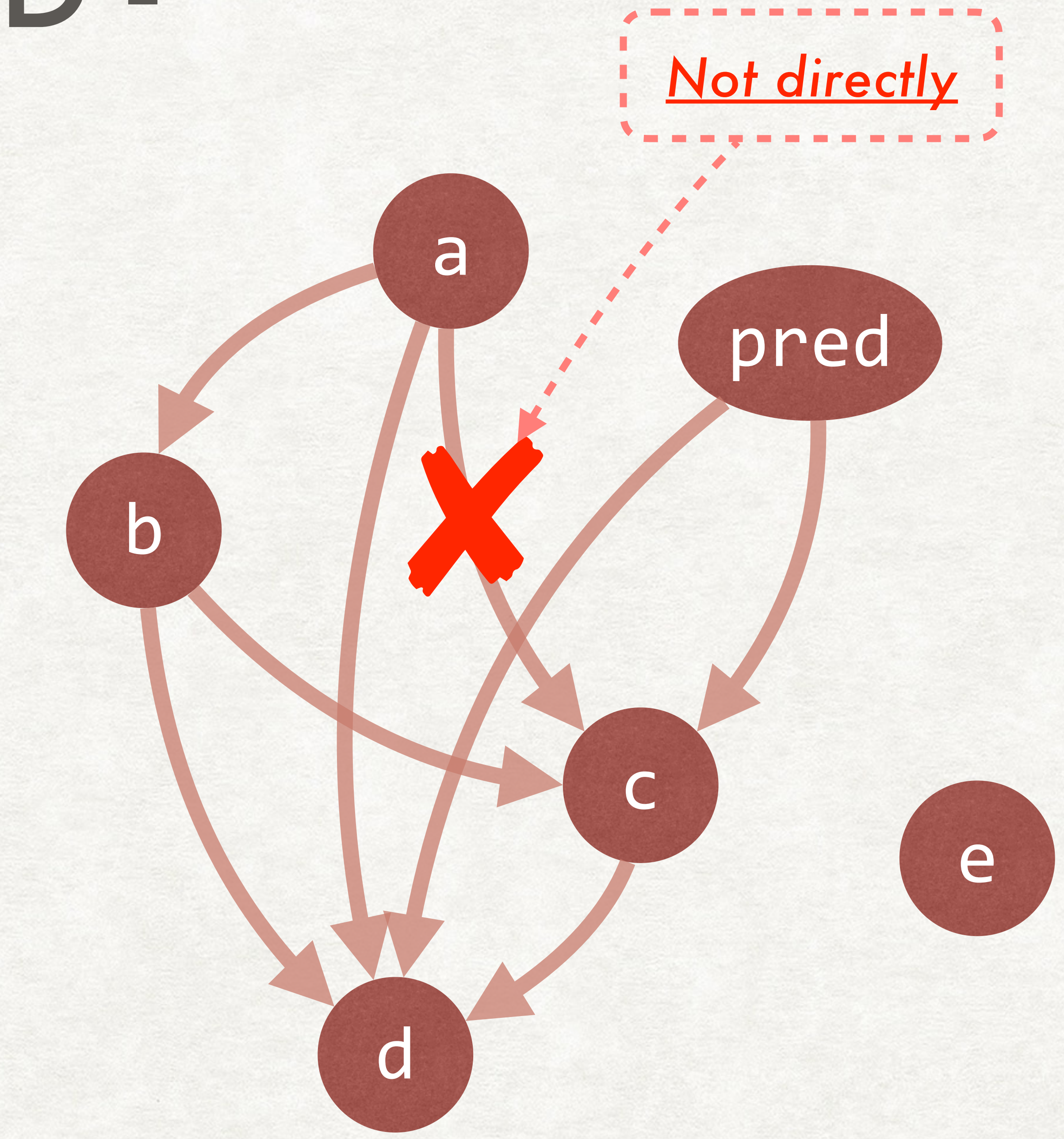
FINDING DEPENDENCE STRUCTURE

```
a = 42
pred = input() // 0 or 1
b = a + 1
if (pred):
    c = 2b
    d = c - 1
e = 3
```

Mut	a	pred	b	c	d	e
∅	=	=	=	=	=	=
a	≠	=	≠	≠/=	≠/=	=
pred	=	≠	=	≠	≠	=
b	=	=	≠	≠/=	≠/=	=
c	=	=	=	≠	≠	=
d	=	=	=	=	≠	=
e	=	=	=	=	=	≠

- MOAD -

```
   a: {}
pred: {}
   b: {a}
   c: {a, pred, b}
   d: {a, pred, b, c}
   e: {}
```



STRUCTURE DISCOVERY

WHICH ONE DIRECTLY AFFECTS ANOTHER?

STRUCTURE DISCOVERY

WHICH ONE DIRECTLY AFFECTS ANOTHER?

- Principle: Y **directly affects** X, if there is a **unique effect** of Y on X.
- Direct predecessors (**parents**) of X, denoted as PA_X, is a minimal set of predecessors **sufficient to describe** the state of X.

$$P(X \mid \text{all predecessors of } X) = P(X \mid PA_X)$$

STRUCTURE DISCOVERY

WHICH ONE DIRECTLY AFFECTS ANOTHER?

- Principle: Y **directly affects** X, if there is a **unique effect** of Y on X.
- Direct predecessors (**parents**) of X, denoted as PA_X, is a minimal set of predecessors **sufficient to describe** the state of X.

$$P(X \mid \text{all predecessors of } X) = P(X \mid PA_X)$$

- If the effect of Z on X is **masked** by other predecessors of X, Z is **not a parent** of X.

$$P(X \mid Z, \text{other preds. of } X) = P(X \mid \text{other preds. of } X) \Rightarrow Z \notin PA_X$$

STRUCTURE DISCOVERY

```
a = 42
pred = input() // 0 or 1
b = a + 1
if (pred):
    c = 2b
    d = c - 1
e = 3
```

[Predecessors]

b: {a}, c: {a, pred, b}, d: {a, pred, b, c}

Mut	a	pred	b	c	d	e
∅	=	=	=	=	=	=
a	≠	=	≠	≠/=	≠/=	=
pred	=	≠	=	≠	≠	=
b	=	=	≠	≠/=	≠/=	=
c	=	=	=	≠	≠	=
d	=	=	=	=	≠	=
e	=	=	=	=	=	≠

":" means the state of the variable, not the value

$[PA_c]$

- a: when pred : '=', b : '≠',

  $P(c : '\neq' \mid a : '\neq') = 0.5$, and

  $P(c : '\neq') = 0.5$

$\Rightarrow$ a is not a parent of c

STRUCTURE DISCOVERY

```
a = 42
pred = input() // 0 or 1
b = a + 1
if (pred):
    c = 2b
    d = c - 1
e = 3
```

[Predecessors]

b: {a}, <u>c: {a, pred, b},</u> d: {a, pred, b, c}

Mut	a	pred	b	c	d	e
∅	=	=	=	=	=	=
a	≠	=	≠	≠/=	≠/=	=
pred	=	≠	=	≠	≠	=
b	=	=	≠	≠/=	≠/=	=
c	=	=	=	≠	≠	=
d	=	=	=	=	≠	=
e	=	=	=	=	=	≠

":" means the state of the variable, not the value

$[PA_c]$

- a: when pred : '=', b : '≠',

  $P(\mathrm{c} : \text{'≠'} \mid \mathrm{a} : \text{'≠'}) = 0.5$, and

  $P(\mathrm{c} : \text{'≠'}) = 0.5$

  $\Rightarrow$ <u>a is not a parent of c</u>

STRUCTURE DISCOVERY

```
a = 42
pred = input() // 0 or 1
b = a + 1
if (pred):
    c = 2b
    d = c - 1
e = 3
```

[Predecessors]

b: {a}, c: {a, pred, b}, d: {a, pred, b, c}

Mut	a	pred	b	c	d	e
∅	=	=	=	=	=	=
a	≠	=	≠	≠/=	≠/=	=
pred	=	≠	=	≠	≠	=
b	=	=	≠	≠/=	≠/=	=
c	=	=	=	≠	≠	=
d	=	=	=	=	≠	=
e	=	=	=	=	=	≠

":" means the state of the variable, not the value

$[PA_c]$

- a: when pred : '=', b : '≠',

  $P(c : '\neq' \mid a : '\neq') = 0.5$, and

  $P(c : '\neq') = 0.5$

$\Rightarrow$ <u>a is not a parent of c</u>

STRUCTURE DISCOVERY

```
a = 42
pred = input() // 0 or 1
b = a + 1
if (pred):
    c = 2b
    d = c - 1
e = 3
```

[Predecessors]

b: {a}, c: {a, pred, b}, d: {a, pred, b, c}

Mut	a	pred	b	c	d	e
∅	=	=	=	=	=	=
a	≠	=	≠	≠/=	≠/=	=
pred	=	≠	=	≠	≠	=
b	=	=	≠	≠/=	≠/=	=
c	=	=	=	≠	≠	=
d	=	=	=	=	≠	=
e	=	=	=	=	=	≠

":" means the state of the variable, not the value

$[PA_c]$

- a: when pred : '=', b : '≠',

  $P(c : \text{'≠'} \mid a : \text{'≠'}) = 0.5$, and

  $P(c : \text{'≠'}) = 0.5$

$\Rightarrow$ a is not a parent of c

STRUCTURE DISCOVERY

```
a = 42
pred = input() // 0 or 1
b = a + 1
if (pred):
    c = 2b
    d = c - 1
e = 3
```

[Predecessors]

b: {a}, c: {a, pred, b}, d: {a, pred, b, c}

Mut	a	pred	b	c	d	e
∅	=	=	=	=	=	=
a	≠	=	≠	≠/=	≠/=	=
pred	=	≠	=	≠	≠	=
b	=	=	≠	≠/=	≠/=	=
c	=	=	=	≠	≠	=
d	=	=	=	=	≠	=
e	=	=	=	=	=	≠

$[PA_{\text{c}}]$

- pred: when b : '=',

  $P(\text{c} : \text{'}\neq\text{'} \mid \text{pred} : \text{'}\neq\text{'}) = 1.0,$ and

  $P(\text{c} : \text{'}\neq\text{'}) = 0.5$

$\Rightarrow$ pred is a parent of c

STRUCTURE DISCOVERY

[Predecessors]

b: {a}, c: {a, pred, b}, d: {a, pred, b, c}

```
1: a = 42
2: pred = input() // 0 or 1
3: b = a + 1
4: if (pred):
5:     c = 2b
6:     d = c - 1
7: e = 3
```

Mut	a	pred	b	c	d	e
∅	=	=	=	=	=	=
a	≠	=	≠	≠/=	≠/=	=
pred	=	≠	=	≠	≠	=
b	=	=	≠	≠/=	≠/=	=
c	=	=	=	≠	≠	=
d	=	=	=	=	≠	=
e	=	=	=	=	=	≠

$[PA_{\mathrm{c}}]$

- b: when pred : '=',

  $P(\mathrm{c} : \text{'}\neq\text{'} \mid \mathrm{b} : \text{'}\neq\text{'}) = 0.5$, and

  $P(\mathrm{c} : \text{'}\neq\text{'}) = 0.25$

$\Rightarrow$ b is a parent of c

STRUCTURE DISCOVERY

```
[Predecessors]
b: {a},      c: {a, pred, b},      d: {a, pred, b, c}
```

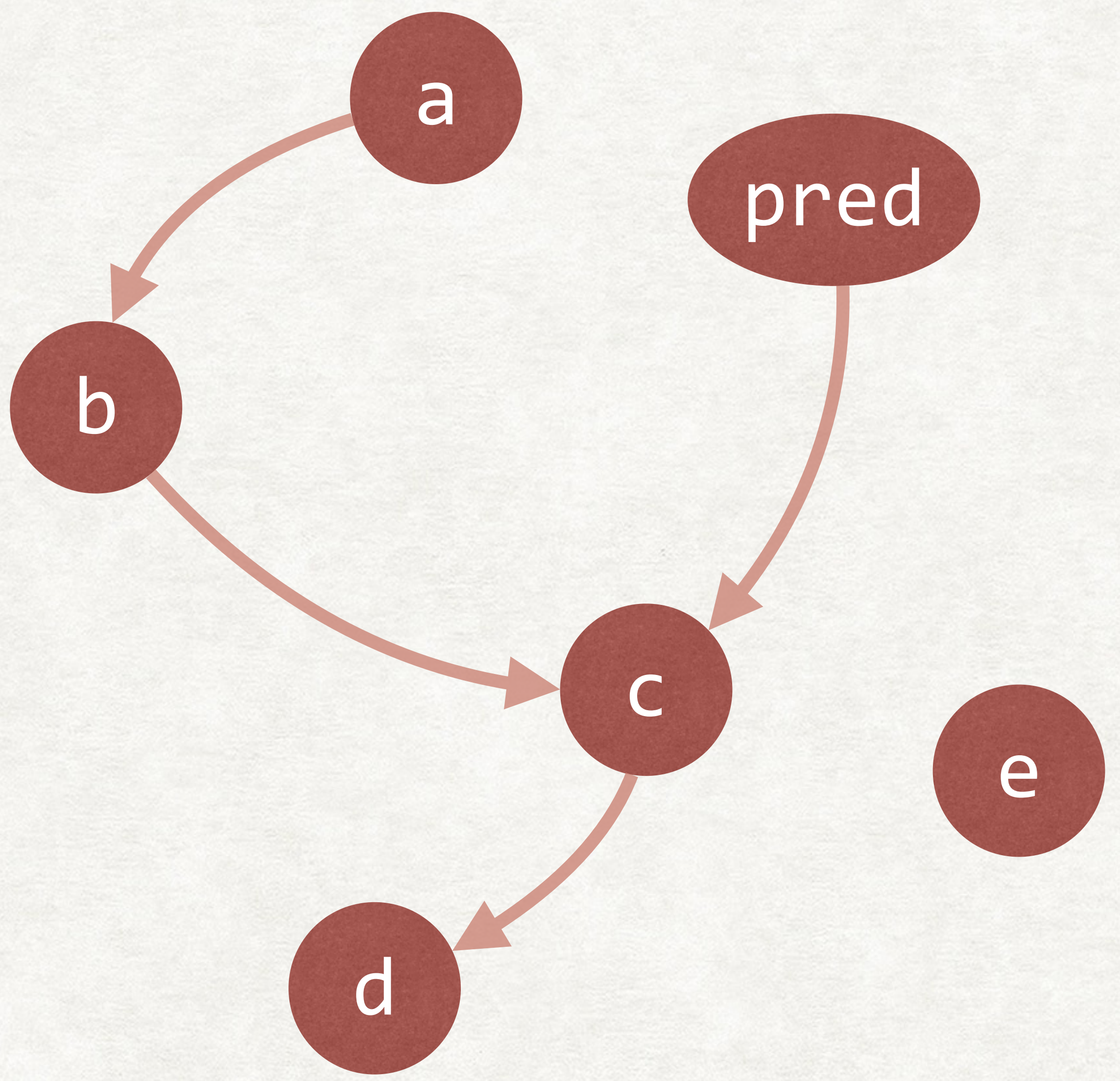
Mut	a	pred	b	c	d	e
∅	=	=	=	=	=	=
a	≠	=	≠	≠/=	≠/=	=
pred	=	≠	=	≠	≠	=
b	=	=	≠	≠/=	≠/=	=
c	=	=	=	≠	≠	=
d	=	=	=	=	≠	=
e	=	=	=	=	=	≠

STRUCTURE DISCOVERY

[Predecessors]

b: {a}, c: {a, pred, b}, d: {a, pred, b, c}

Mut	a	pred	b	c	d	e
∅	=	=	=	=	=	=
a	≠	=	≠	≠/=	≠/=	=
pred	=	≠	=	≠	≠	=
b	=	=	≠	≠/=	≠/=	=
c	=	=	=	≠	≠	=
d	=	=	=	=	≠	=
e	=	=	=	=	=	≠



Program elements & their parents
:= Causal Structure

QUANTIFYING DEPENDENCY

- Quantifying program dependency:

MEASURE: HOW OFTEN ONE'S CHANGE CAUSES ANOTHER TO CHANGE?

QUANTIFYING DEPENDENCY

- Quantifying program dependency:

MEASURE: HOW OFTEN ONE'S CHANGE CAUSES ANOTHER TO CHANGE?

- Naive method: Directly compute the relative frequency of change from mutation attempts.

$$Dep(X \rightarrow Y) = P(Y \text{ is changed} \mid X \text{ is mutated})$$

  - Problem: X may affect Y occasionally, and it could be costly to mutate X multiple times for a confident result.

QUANTIFYING DEPENDENCY

- Quantifying program dependency:

MEASURE: HOW OFTEN ONE'S CHANGE CAUSES ANOTHER TO CHANGE?

- Naive method: Directly compute the relative frequency of change from mutation attempts.

$$Dep(X \to Y) = P(Y \text{ is changed} \mid X \text{ is mutated})$$

  - Problem: X may affect Y occasionally, and it could be costly to mutate X multiple times for a confident result.

- Instead, we employ ***causal inference*** to estimate causation from observation data, so that, we can leverage observations of cases when "X is changed."
  - $\#(X \text{ is changed}) \gg \#(X \text{ is mutated})$

WHY CAUSAL INFERENCE IS NEEDED?

WHY CAUSAL INFERENCE IS NEEDED?

Association $\neq$ Causation

WHY CAUSAL INFERENCE IS NEEDED?

Association $\neq$ Causation

$P(Y$ is changed | X is changed$)$ $\neq$ $Dep(X \rightarrow Y)$

($\approx P(Y$ is changed | X is mutated$)$)

WHY: ASSOCIATION IS NOT CAUSATION

```
pred = input() // 0 or 1
if (pred):
    x = f()
    y = g()
```

	pred: '='		pred: '≠'	
	x: '≠'	x: '='	x: '≠'	x: '='
y: '≠'	0.01	0.09	0.99	ϵ_1
Y: '='	0.09	0.81	ϵ_2	ϵ_3
Sum	1.00		1.00	

Joint probability distribution of observations

WHY: ASSOCIATION IS NOT CAUSATION

- If we let $Dep(X \to Y) = P(Y : \text{'≠'} \mid X : \text{'≠'})$, (association)

```
pred = input() // 0 or 1
if (pred):
    x = f()
    y = g()
```

$$P(\mathrm{y} = \text{'≠'} \mid \mathrm{x} = \text{'≠'}) = \frac{0.01 + 0.99}{0.01 + 0.99 + 0.09 + \epsilon_2} \approx 0.92$$

	pred: '='		pred: '≠'	
	x: '≠'	x: '='	x: '≠'	x: '='
y: '≠'	0.01	0.09	0.99	ϵ_1
Y: '='	0.09	0.81	ϵ_2	ϵ_3
Sum	1.00		1.00	

WHY: ASSOCIATION IS NOT CAUSATION

- If we let $Dep(X \rightarrow Y) = P(Y : \text{'}\neq\text{'} \mid X : \text{'}\neq\text{'})$, (association)

```
1: pred = input() // 0 or 1
2: if (pred):
3:     x = f()
4:     y = g()
```

$$P(\mathrm{y} = \text{'}\neq\text{'} \mid \mathrm{x} = \text{'}\neq\text{'}) = \frac{0.01 + 0.99}{0.01 + 0.99 + 0.09 + \epsilon_2} \approx 0.92$$

$\Rightarrow$ x highly affects y? ✗

	pred: '='		pred: '≠'	
	x: '≠'	x: '='	x: '≠'	x: '='
y: '≠'	0.01	0.09	0.99	ϵ_1
Y: '='	0.09	0.81	ϵ_2	ϵ_3
Sum	1.00		1.00	

WHY: ASSOCIATION IS NOT CAUSATION

- $Dep(X \to Y) \cancel{=} P(Y : \text{'}\neq\text{'} \mid X : \text{'}\neq\text{'})$, (association)

```
pred = input() // 0 or 1
if (pred):
    x = f()
    y = g()
```

$$P(\mathrm{y} = \text{'}\neq\text{'} \mid \mathrm{x} = \text{'}\neq\text{'}) = \frac{0.01 + 0.99}{0.01 + 0.99 + 0.09 + \epsilon_2} \approx 0.92$$

	pred: '='		pred: '≠'	
	x: '≠'	x: '='	x: '≠'	x: '='
y: '≠'	0.01	0.09	0.99	ϵ_1
Y: '='	0.09	0.81	ϵ_2	ϵ_3
Sum	1.00		1.00	

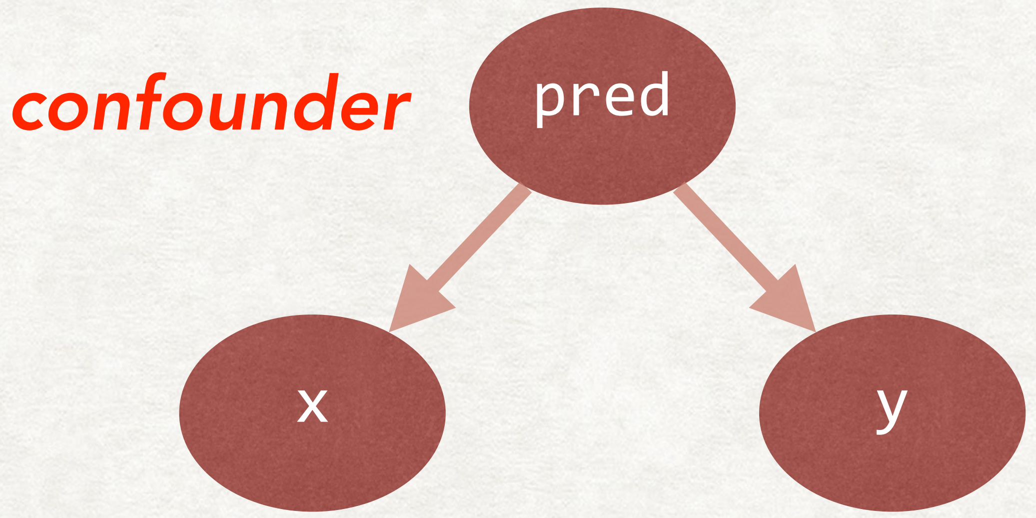
WHY: ASSOCIATION IS NOT CAUSATION

- $Dep(X \rightarrow Y) \cancel{=} P(Y : \text{'≠'} \mid X : \text{'≠'})$, (association)

```
pred = input() // 0 or 1
if (pred):
    x = f()
    y = g()
```

$$P(\mathrm{y} = \text{'≠'} \mid \mathrm{x} = \text{'≠'}) = \frac{0.01 + 0.99}{0.01 + 0.99 + 0.09 + \epsilon_2} \approx 0.92$$

	pred: '='		pred: '≠'	
	x: '≠'	x: '='	x: '≠'	x: '='
y: '≠'	0.01	0.09	0.99	ϵ_1
Y: '='	0.09	0.81	ϵ_2	ϵ_3
Sum	1.00		1.00	



This is because, when pred changes, both x and y change.

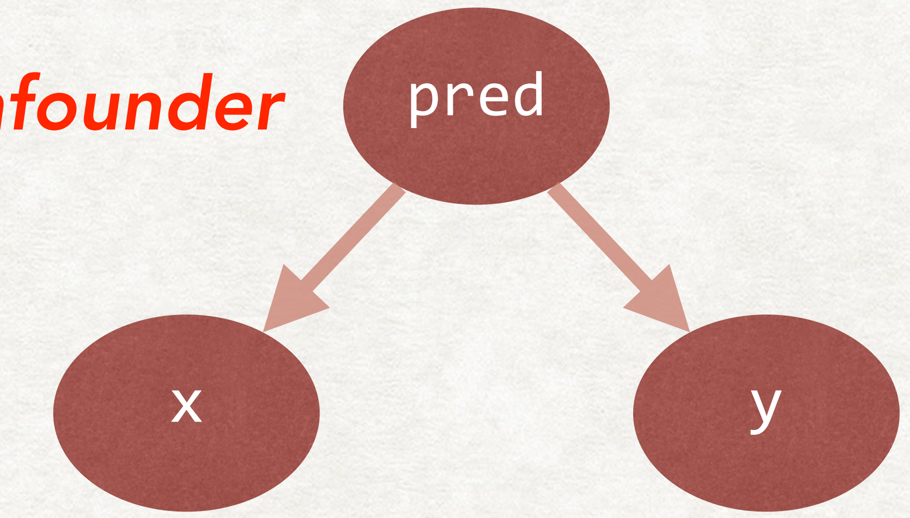
WHY: ASSOCIATION IS NOT CAUSATION

- $Dep(X \to Y) \cancel{=} P(Y : \text{'≠'} \mid X : \text{'≠'})$, (association)

```
pred = input() // 0 or 1
if (pred):
    x = f()
    y = g()
```

$$P(\mathrm{y} = \text{'≠'} \mid \mathrm{x} = \text{'≠'}) = \frac{0.01 + 0.99}{0.01 + 0.99 + 0.09 + \epsilon_2} \approx 0.92$$

confounder



This is because, when pred changes, both x and y change.

	pred: '='		pred: '≠'	
	x: '≠'	x: '='	x: '≠'	x: '='
y: '≠'	0.01	0.09	0.99	ϵ_1
Y: '='	0.09	0.81	ϵ_2	ϵ_3
Sum	1.00		1.00	

Solution to get causation: *control the confounder*

CAUSAL INFERENCE

```
pred = input() // 0 or 1
if (pred):
    x = f()
    y = g()
```

- Compute $Dep(X \rightarrow Y)$ using causal inference.

Case 1. pred: '='

$$P(\mathrm{y} = \text{'}\neq\text{'} \mid \mathrm{x} = \text{'}\neq\text{'}) = \frac{0.01}{0.01 + 0.09} = 0.1$$

	pred: '='		pred: '≠'	
	x: '≠'	x: '='	x: '≠'	x: '='
y: '≠'	0.01	0.09	0.99	ϵ_1
Y: '='	0.09	0.81	ϵ_2	ϵ_3
Sum	1.00		1.00	

CAUSAL INFERENCE

```
pred = input() // 0 or 1
if (pred):
    x = f()
    y = g()
```

- Compute $Dep(X \rightarrow Y)$ using causal inference.

Case 1. pred: '='

$$P(\mathrm{y} = '\neq' \mid \mathrm{x} = '\neq') = \frac{0.01}{0.01 + 0.09} = 0.1$$

Case 2. pred: '≠'

$$P(\mathrm{y} = '\neq' \mid \mathrm{x} = '\neq') = \frac{0.99}{0.99 + \epsilon_2} \approx 1.0$$

	pred: '='		pred: '≠'	
	x: '≠'	x: '='	x: '≠'	x: '='
y: '≠'	0.01	0.09	0.99	ϵ_1
Y: '='	0.09	0.81	ϵ_2	ϵ_3
Sum	1.00		1.00	

CAUSAL INFERENCE

```
pred = input() // 0 or 1
if (pred):
    x = f()
    y = g()
```

- Compute $Dep(X \rightarrow Y)$ using causal inference.

Case 1. pred: '=' $\quad (P(\text{pred: '='}) = 0.5)$

$$P(\text{y} = \text{'≠'} \mid \text{x} = \text{'≠'}) = \frac{0.01}{0.01 + 0.09} = 0.1$$

Case 2. pred: '≠' $\quad (P(\text{pred: '≠'}) = 0.5)$

$$P(\text{y} = \text{'≠'} \mid \text{x} = \text{'≠'}) = \frac{0.99}{0.99 + \epsilon_2} \approx 1.0$$

Weighted sum $\approx 0.1 \times 0.5 + 1.0 \times 0.5 = 0.55$

	pred: '='		pred: '≠'	
	x: '≠'	x: '='	x: '≠'	x: '='
y: '≠'	0.01	0.09	0.99	ϵ_1
Y: '='	0.09	0.81	ϵ_2	ϵ_3
Sum	1.00		1.00	

CAUSAL INFERENCE

```
pred = input() // 0 or 1
if (pred):
    x = f()
    y = g()
```

- Compute $Dep(X \rightarrow Y)$ using causal inference.

Case 1. pred: '=' $\quad (P(\text{pred: '='}) = 0.5)$

$$P(\text{y} = \text{'}\neq\text{'} \mid \text{x} = \text{'}\neq\text{'}) = \frac{0.01}{0.01 + 0.09} = 0.1$$

Case 2. pred: '≠' $\quad (P(\text{pred: '}\neq\text{'}) = 0.5)$

$$P(\text{y} = \text{'}\neq\text{'} \mid \text{x} = \text{'}\neq\text{'}) = \frac{0.99}{0.99 + \epsilon_2} \approx 1.0$$

Weighted sum $\approx 0.1 \times 0.5 + 1.0 \times 0.5 = 0.55$

$$:= P(\text{y} = \text{'}\neq\text{'} \mid do(\text{x} = \text{'}\neq\text{'}))$$

	pred: '='		pred: '≠'	
	x: '≠'	x: '='	x: '≠'	x: '='
y: '≠'	0.01	0.09	0.99	ϵ_1
Y: '='	0.09	0.81	ϵ_2	ϵ_3
Sum	1.00		1.00	

Probability of y: '≠' when x is forcefully set to '≠'.

CAUSAL INFERENCE

```
pred = input() // 0 or 1
if (pred):
    x = f()
    y = g()
```

- Compute $Dep(X \rightarrow Y)$ using causal inference.

	pred: '='		pred: '≠'	
	x: '≠'	x: '='	x: '≠'	x: '='
y: '≠'	0.01	0.09	0.99	ϵ_1
Y: '='	0.09	0.81	ϵ_2	ϵ_3
	1.00		1.00	

Case 1. pred: '=' $\quad (P(\text{pred: '='}) = 0.5)$

$$P(\text{y} = \text{'≠'} \mid \text{x} = \text{'≠'}) = \frac{0.01}{0.01 + 0.09} = 0.1$$

Case 2. pred: '≠' $\quad (P(\text{pred: '≠'}) = 0.5)$

$$P(\text{y} = \text{'≠'} \mid \text{x} = \text{'≠'}) = \frac{0.99}{0.99 + \epsilon_2} \approx 1.0$$

Prob. is almost the same because x does not affect y.

$P(y = \text{'≠'}) \approx 0.55$

Weighted sum $\approx 0.1 \times 0.5 + 1.0 \times 0.5 = 0.55$

$\approx$

$$:= P(\text{y} = \text{'≠'} \mid do(\text{x} = \text{'≠'}))$$

Probability of y:'≠' when x is forcefully set to '≠'.

CAUSAL INFERENCE

```
1: pred = input() // 0 or 1
2: if (pred):
3:     x = f()
4:     y = g()
```

- Compute $Dep(X \rightarrow Y)$ using causal inference.

$$P(\mathrm{y} = \text{'}\neq\text{'} \mid do(\mathrm{x} = \text{'}\neq\text{'})) \approx 0.55$$

	pred: '='		pred: '≠'	
	x: '≠'	x: '='	x: '≠'	x: '='
y: '≠'	0.01	0.09	0.99	ϵ_1
Y: '='	0.09	0.81	ϵ_2	ϵ_3
Sum	1.00		1.00	

CAUSAL INFERENCE

```
pred = input() // 0 or 1
if (pred):
    x = f()
    y = g()
```

- Compute $Dep(X \rightarrow Y)$ using causal inference.

$$P(\text{y} = \text{'}\neq\text{'} \mid do(\text{x} = \text{'}\neq\text{'})) \approx 0.55$$

$$P(\text{y} = \text{'}\neq\text{'} \mid do(\text{x} = \text{'='})) \approx 0.55$$

$$= \frac{0.09}{0.09 + 0.81} \times 0.5 + \frac{\epsilon_1}{\epsilon_1 + \epsilon_3} \times 0.5 \approx (0.1 + 1.0) \times 0.5 = 0.55$$

	pred: '='		pred: '≠'	
	x: '≠'	x: '='	x: '≠'	x: '='
y: '≠'	0.01	0.09	0.99	ϵ_1
Y: '='	0.09	0.81	ϵ_2	ϵ_3
Sum	1.00		1.00	

CAUSAL INFERENCE

```
1: pred = input() // 0 or 1
2: if (pred):
3:     x = f()
4:     y = g()
```

- Compute $Dep(X \rightarrow Y)$ using causal inference.

$$P(\mathrm{y} = '\neq' \mid do(\mathrm{x} = '\neq')) \approx 0.55$$

$$P(\mathrm{y} = '\neq' \mid do(\mathrm{x} = '=')) \approx 0.55$$

$$= \frac{0.09}{0.09 + 0.81} \times 0.5 + \frac{\epsilon_1}{\epsilon_1 + \epsilon_3} \times 0.5 \approx (0.1 + 1.0) \times 0.5 = 0.55$$

	pred: '='		pred: '≠'	
	x: '≠'	x: '='	x: '≠'	x: '='
y: '≠'	0.01	0.09	0.99	ϵ_1
Y: '='	0.09	0.81	ϵ_2	ϵ_3
Sum	1.00		1.00	

$$CD(\mathrm{x} \rightarrow \mathrm{y}) = P(\mathrm{y} = '\neq' \mid do(\mathrm{x} = '\neq')) - P(\mathrm{y} = '\neq' \mid do(\mathrm{x} = '=')) \approx 0.00$$

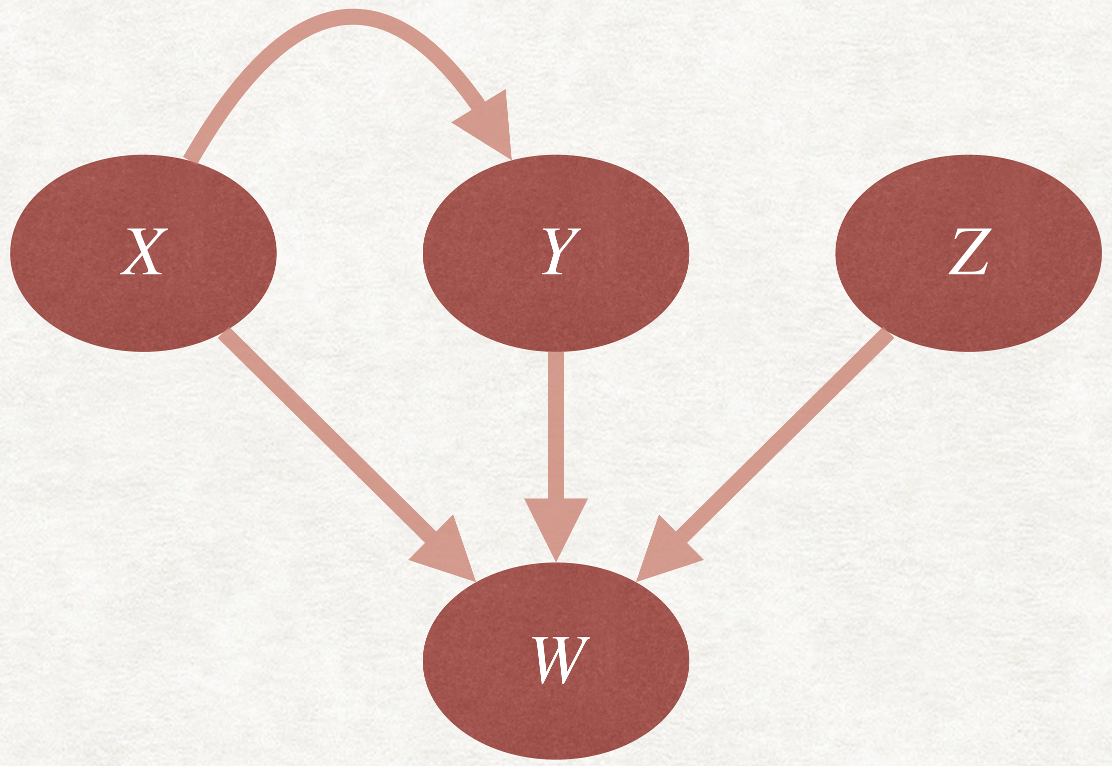
Causal dependence : calculates how often the value of y get affected by the change of x's state (≈ mutating x)

CAUSAL INFERENCE

Direct dependence : computes the direct effect of one element to another

Effect when X : '≠'

– Effect when X : '='



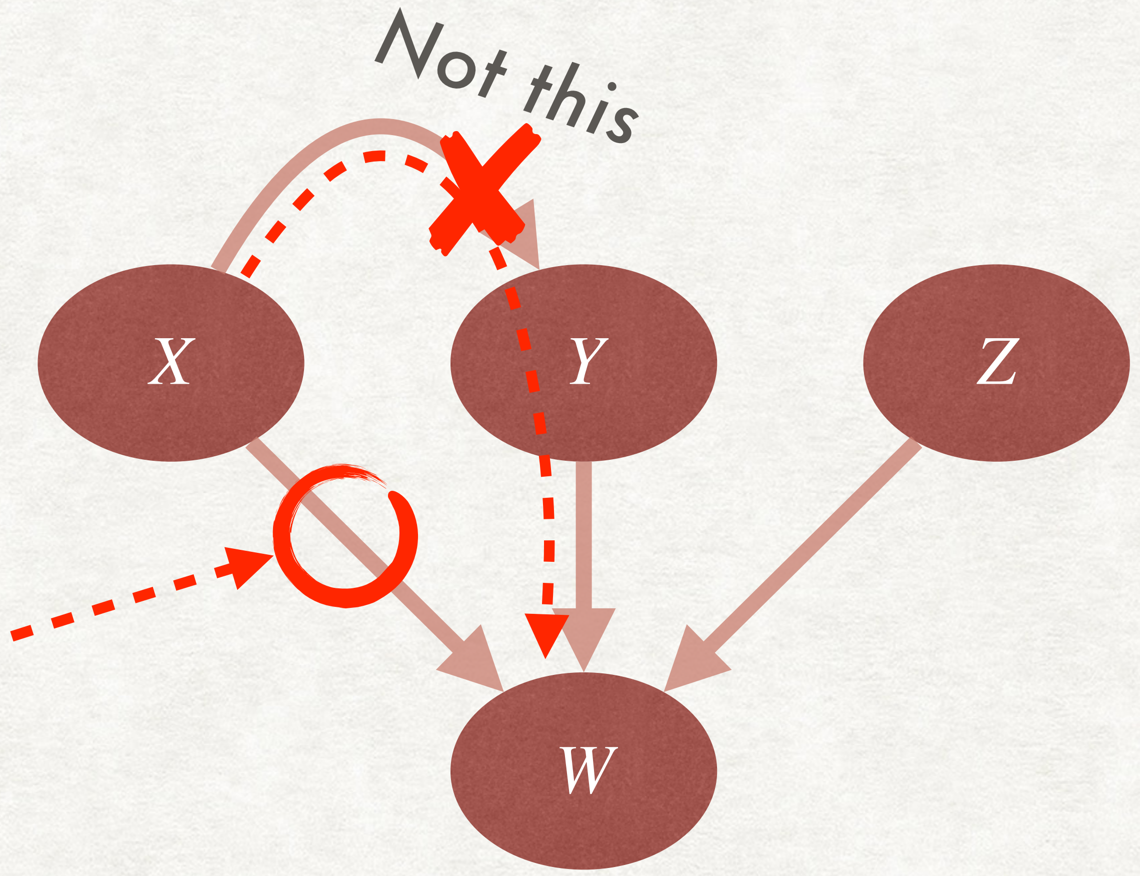
$$DD(X \to W) = \sum_{y,z} \left[P\left(W : '\neq' \mid do(X : '\neq', y, z)\right) - P\left(W : '\neq' \mid do(X : '=', y, z)\right) \right] \times P\left(y, z \mid do(X : '=')\right)$$

CAUSAL INFERENCE

Direct dependence : computes the direct effect of one element to another

Effect when X : '≠'

– Effect when X : '='



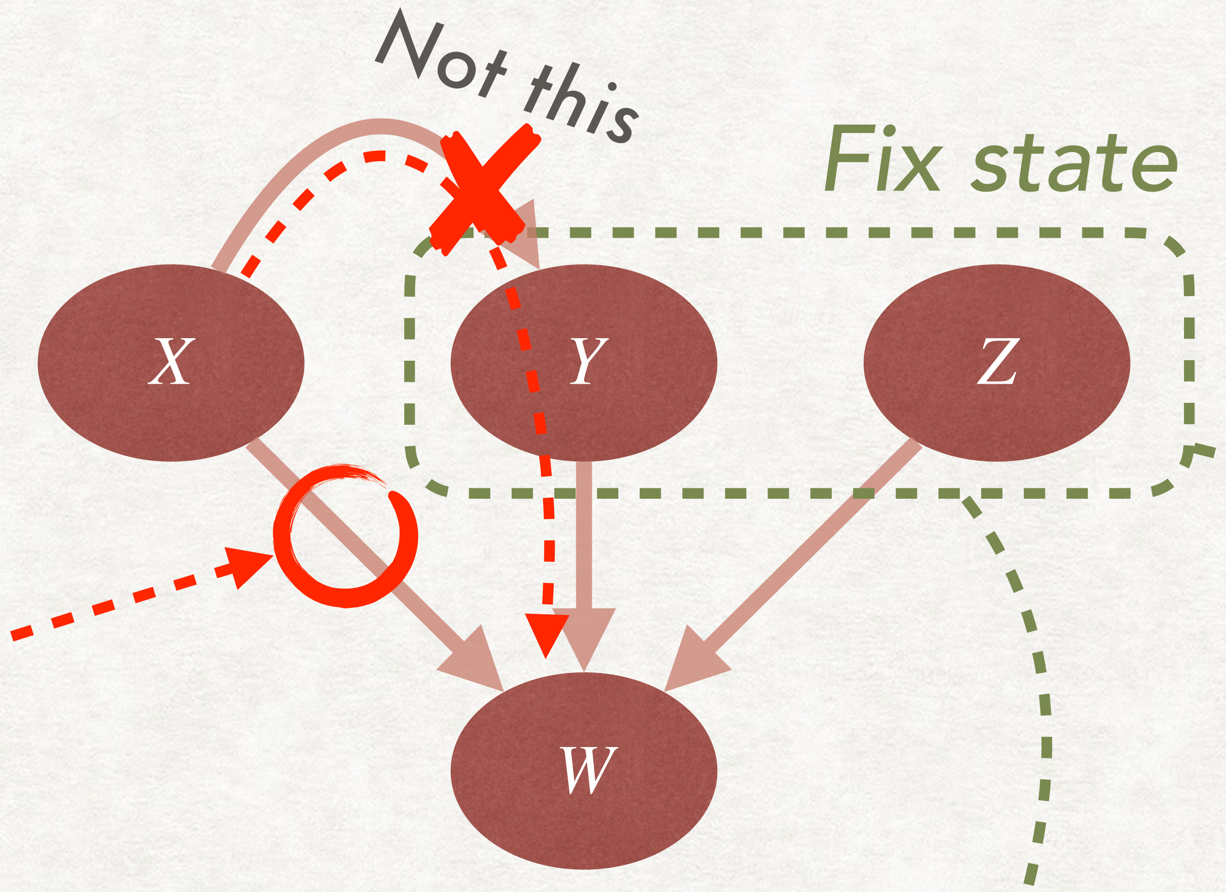
$$DD(X \to W) = \sum_{y,z} \left[P\left(W : '\neq' \mid do(X : '\neq', y, z)\right) - P\left(W : '\neq' \mid do(X : '=', y, z)\right) \right] \times P\left(y, z \mid do(X : '=')\right)$$

CAUSAL INFERENCE

Direct dependence : computes the direct effect of one element to another

Effect when X : '≠'

− Effect when X : '='

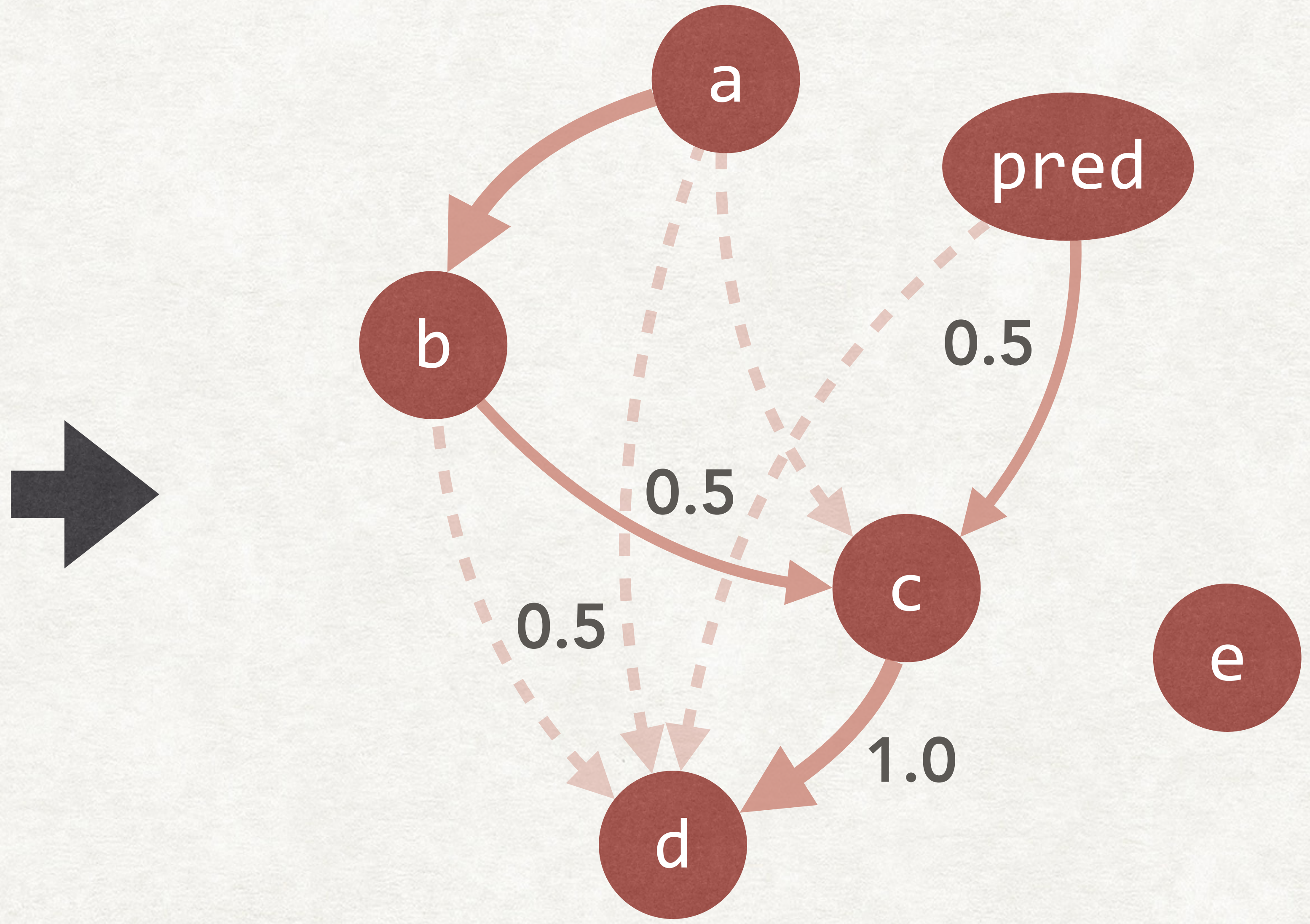


Only this edge

$$DD(X \to W) = \sum_{y,z} \left[P\left(W : '\neq' \mid do(X : '\neq', y, z)\right) - P\left(W : '\neq' \mid do(X : '=', y, z)\right) \right] \times P\left(y, z \mid do(X : '=')\right)$$

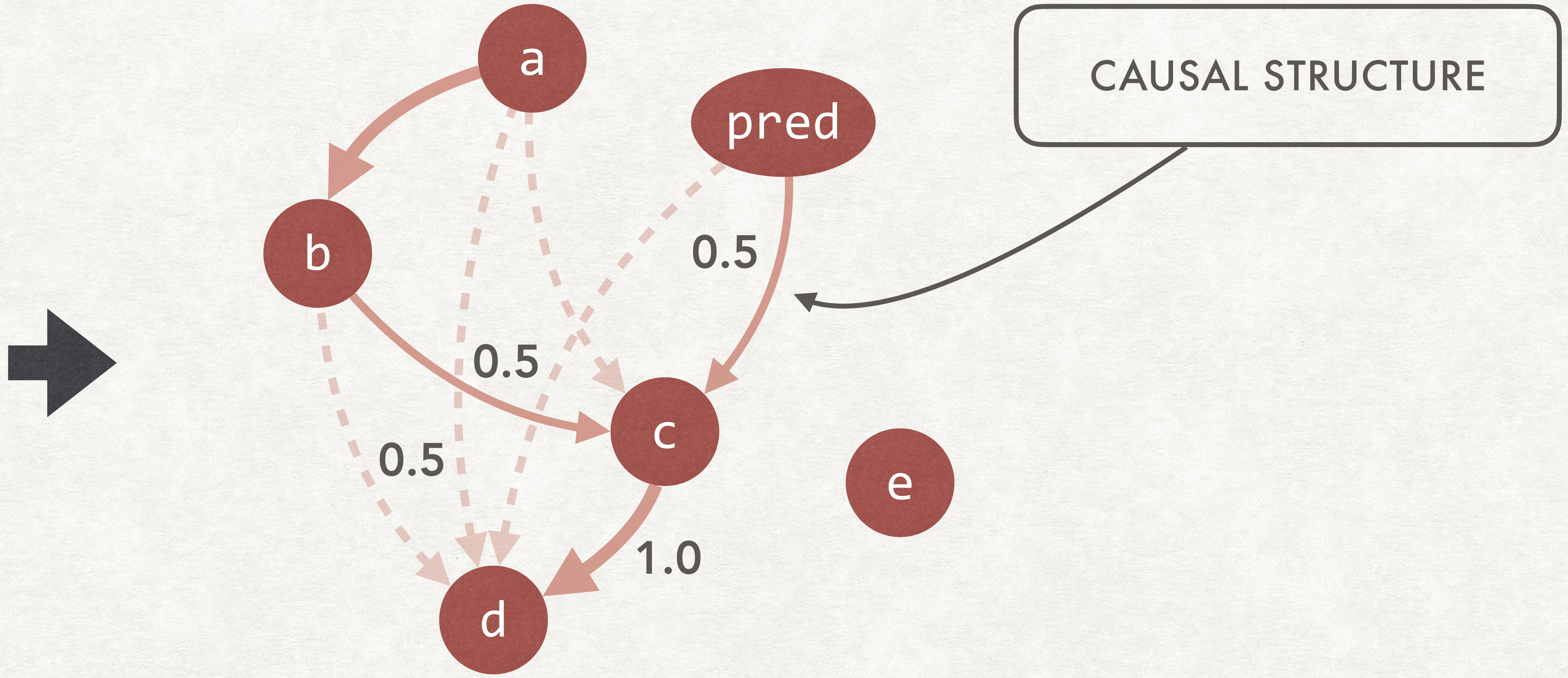
CAUSAL PROGRAM DEPENDENCE MODEL (CPDM)

```
1: a = 42
2: pred = input() // 0 or 1
3: b = a + 1
4: if (pred):
5:     c = 2b
6:     d = c - 1
7: e = 3
```



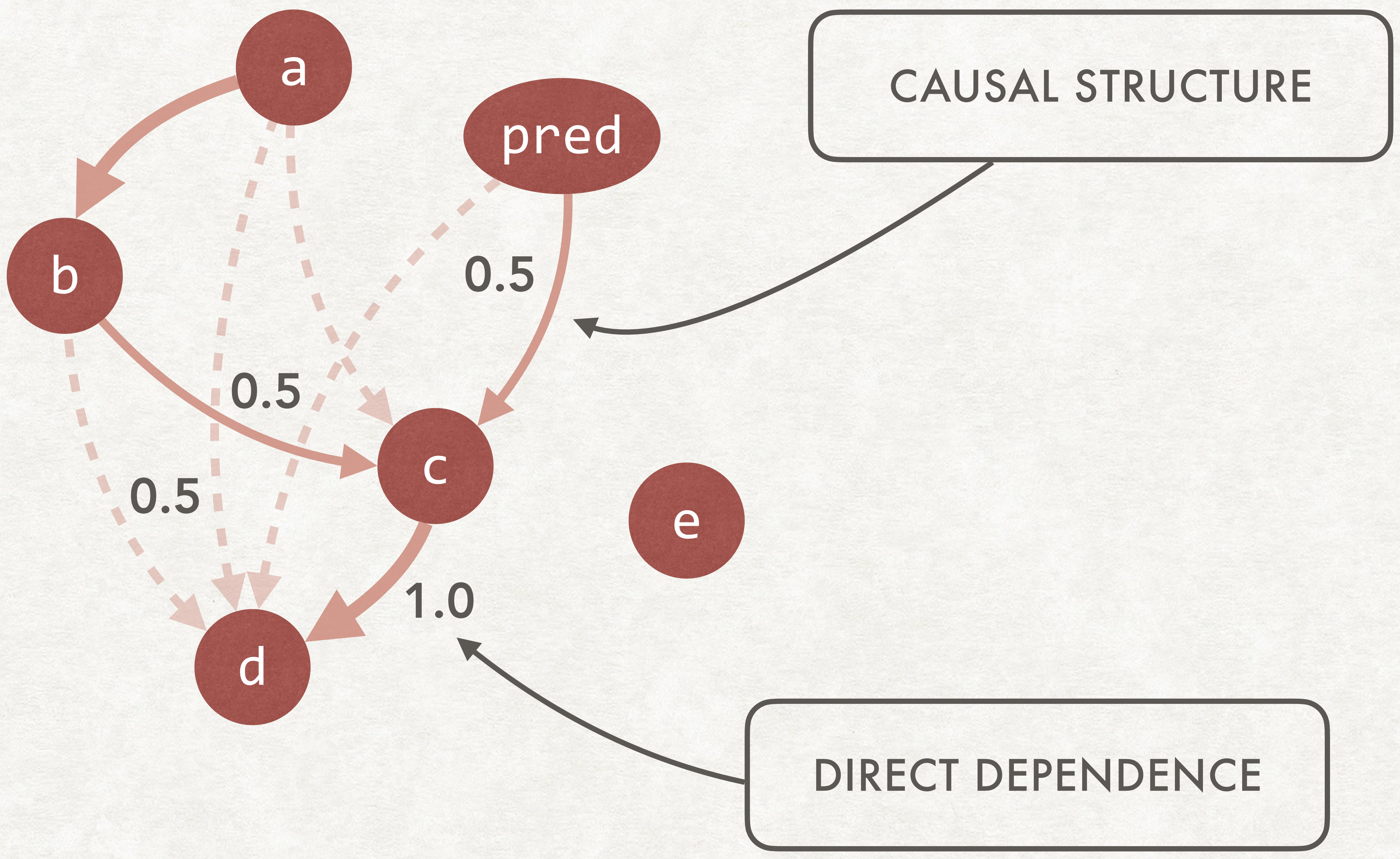
CAUSAL PROGRAM DEPENDENCE MODEL (CPDM)

```
1: a = 42
2: pred = input() // 0 or 1
3: b = a + 1
4: if (pred):
5:     c = 2b
6:     d = c - 1
7: e = 3
```



CAUSAL PROGRAM DEPENDENCE MODEL (CPDM)

```
a = 42
pred = input() // 0 or 1
b = a + 1
if (pred):
    c = 2b
    d = c - 1
e = 3
```



EVALUATION

- Three **program comprehension** scenarios:

VS. program dependence graph (PDG)

1. How this **quantified dependence** can help understanding the program semantics?

Execution awareness of dynamic analysis

2. How does CPDM discriminate the semantics of the same program but **different execution**?

Maintenance application study

3. How quantified dependence can be employed when **debugging**?

EVALUATION

Node index

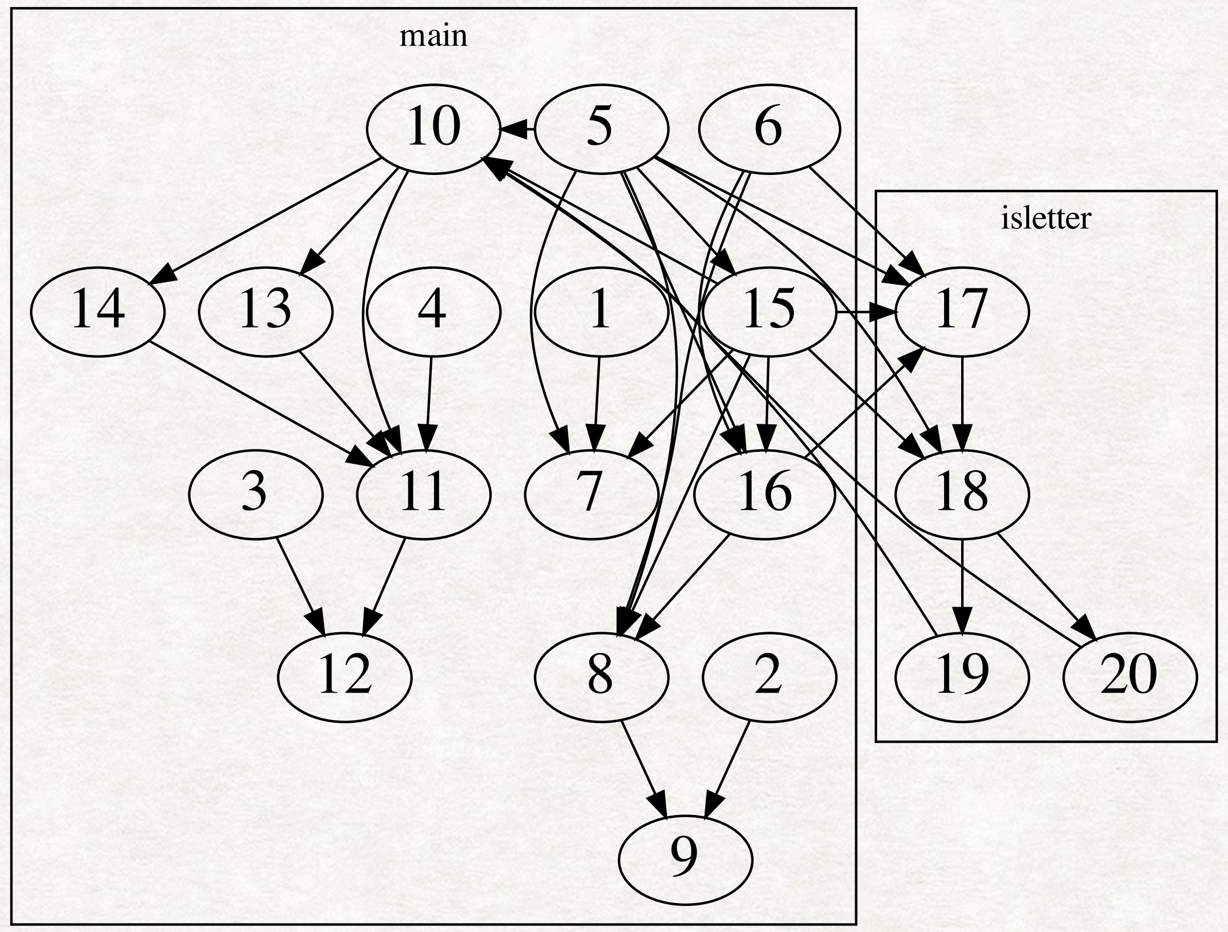
```
def main() {
   <1>characters = 0
   <2>lines = 0
   <3>words = 0
   <4>inword = 0
   <5>_pred1 = getChar(<6>c)
   while (_pred1) {
      <7>characters = characters + 1
      <8>_pred2 = c == '\n'
      if (_pred2)
         <9>lines = lines + 1
      <10>_pred3 = isLetter(c)
      if (_pred3) {
         <11>_pred4 = inword == 0
         if (_pred4) {
            <12>words = words + 1
         }
         <13>inword = 1
      }
      else
         <14>inword = 0
      <15>_pred1 = getChar(<16>c)
   }
}
def isLetter(<17>c) {
   <18>_pred5 = ((c >= 'A' && c <= 'Z')
      || (c >= 'a' && c <= 'z'))
   if (_pred5)
      <19>_ret = True
   else
      <20>_ret = False
   return _ret
}
```

- Subjects: triangle, <u>**word count**</u>, Bill&Ted's
- Word count
  - Get a text input, count the number of
    - characters,
    - lines,
    - and words in the input.

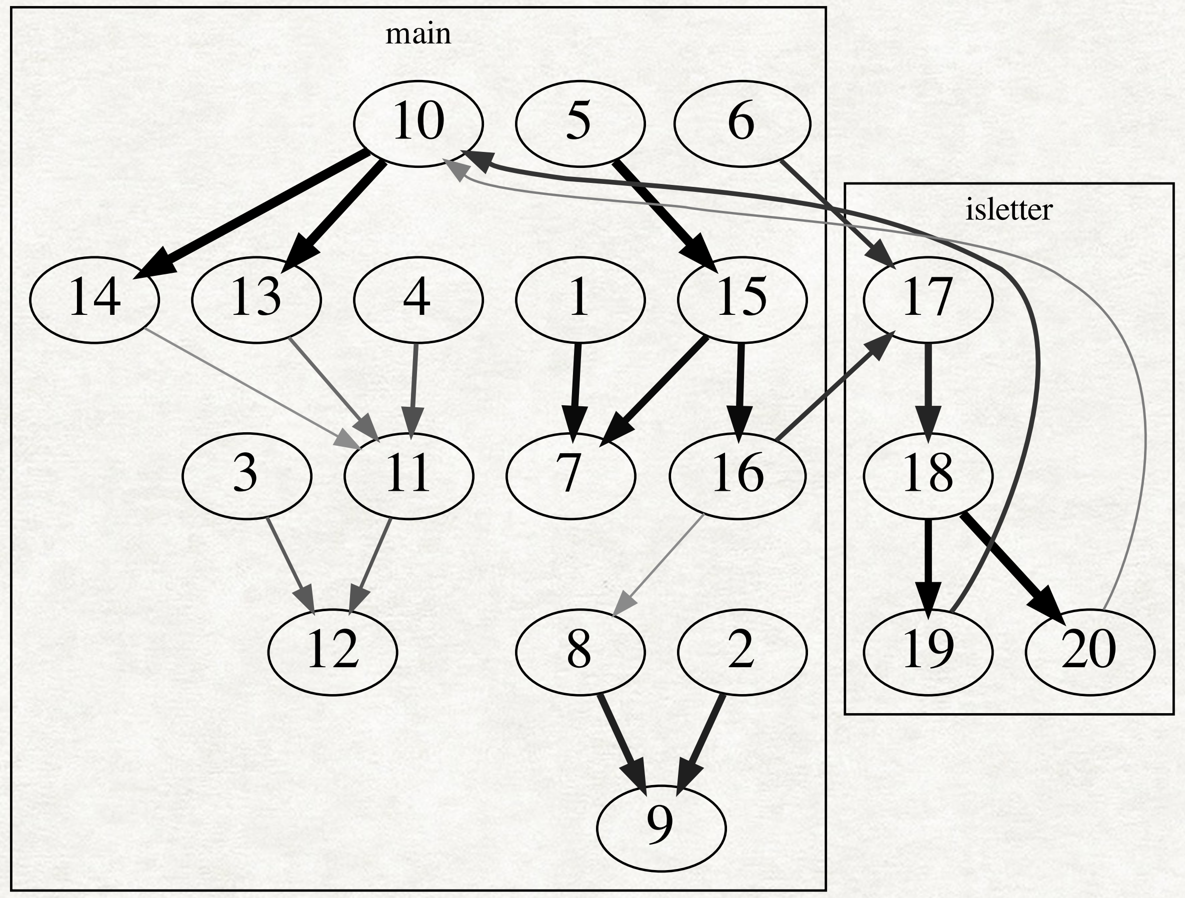
SCENARIO1. CPDM VS PDG

Node index

```
def main() {
   <1>characters = 0
   <2>lines = 0
   <3>words = 0
   <4>inword = 0
   <5>_pred1 = getChar(<6>c)
   while (_pred1) {
      <7>characters = characters + 1
      <8>_pred2 = c == '\n'
      if (_pred2)
         <9>lines = lines + 1
      <10>_pred3 = isLetter(c)
      if (_pred3) {
         <11>_pred4 = inword == 0
         if (_pred4) {
            <12>words = words + 1
         }
         <13>inword = 1
      }
      else
         <14>inword = 0
      <15>_pred1 = getChar(<16>c)
   }
}
def isLetter(<17>c) {
   <18>_pred5 = ((c >= 'A' && c <= 'Z')
      || (c >= 'a' && c <= 'z'))
   if (_pred5)
      <19>_ret = True
   else
      <20>_ret = False
   return _ret
}
```



PDG (34 edges)

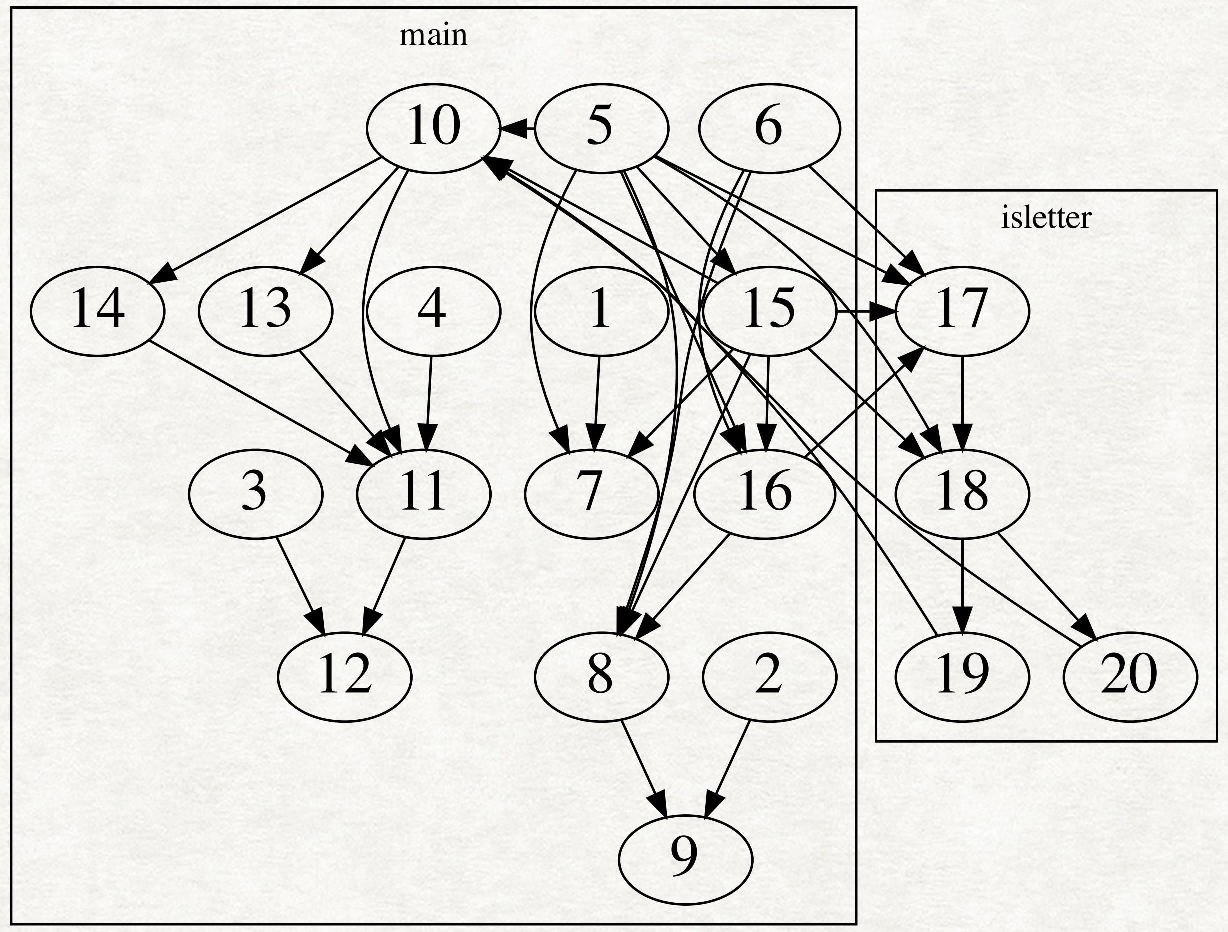


CPDM (21 edges)

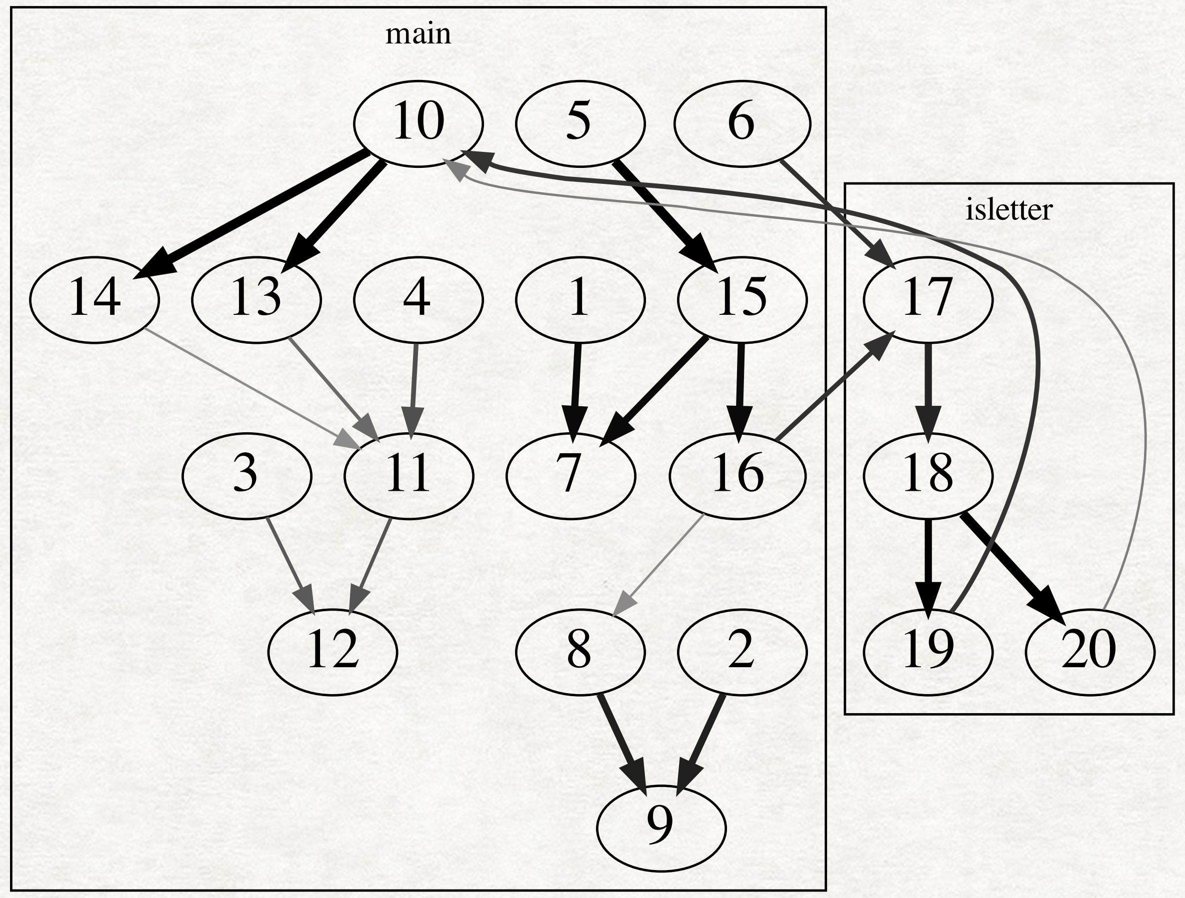
SCENARIO1. CPDM VS PDG

Node index

```
def main() {
   <1>characters = 0
   <2>lines = 0
   <3>words = 0
   <4>inword = 0
   <5>_pred1 = getChar(<6>c)
   while (_pred1) {
      <7>characters = characters + 1
      <8>_pred2 = c == '\n'
      if (_pred2)
         <9>lines = lines + 1
      <10>_pred3 = isLetter(c)
      if (_pred3) {
         <11>_pred4 = inword == 0
         if (_pred4) {
            <12>words = words + 1
         }
         <13>inword = 1
      }
      else
         <14>inword = 0
      <15>_pred1 = getChar(<16>c)
   }
}
def isLetter(<17>c) {
   <18>_pred5 = ((c >= 'A' && c <= 'Z')
      || (c >= 'a' && c <= 'z'))
   if (_pred5)
      <19>_ret = True
   else
      <20>_ret = False
   return _ret
}
```



PDG (34 edges)



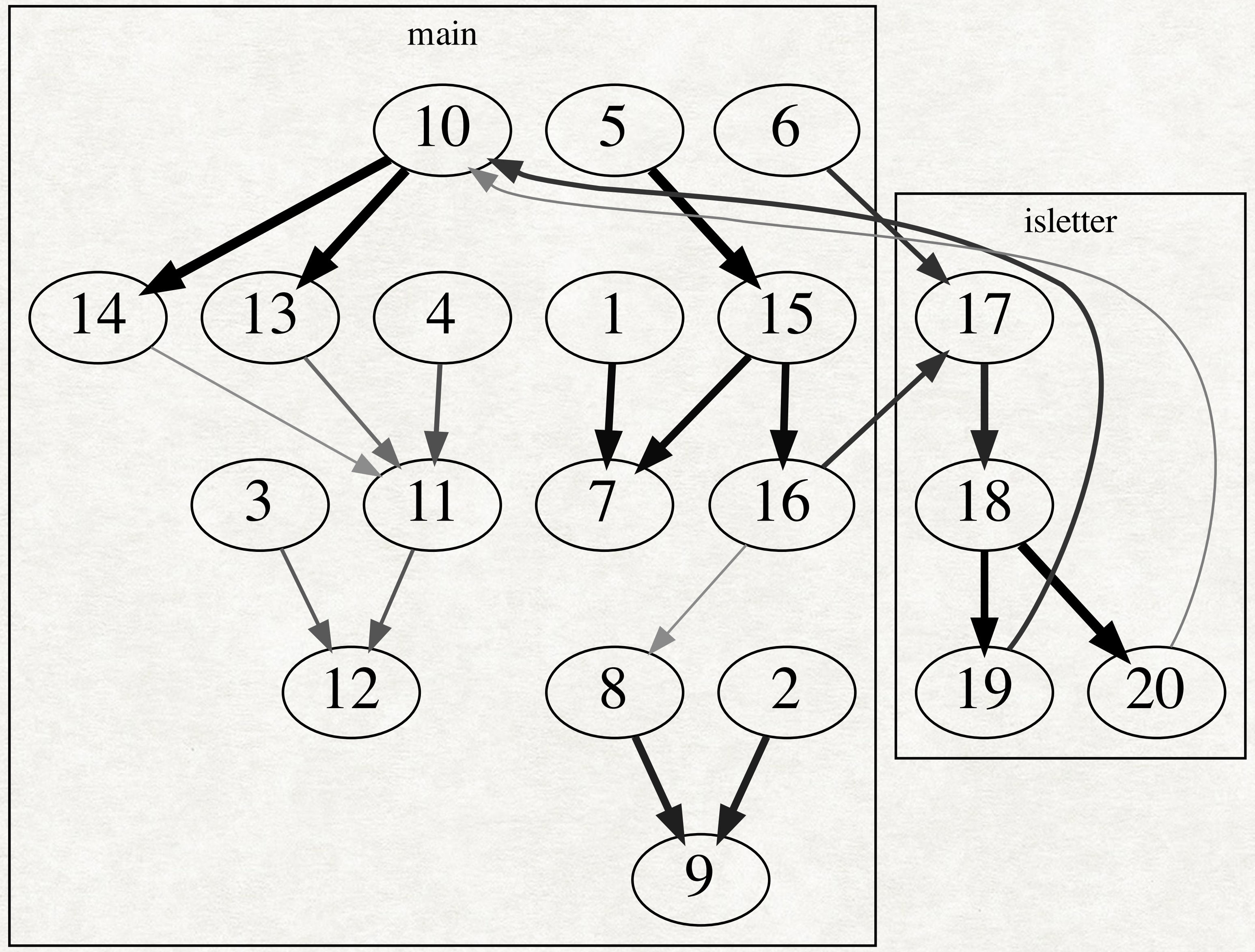
CPDM (21 edges)

CPDM presents edges a smaller number of edges than PDG

SCENARIO1. CPDM VS PDG

Node index

```
def main() {
   <1>characters = 0
   <2>lines = 0
   <3>words = 0
   <4>inword = 0
   <5>_pred1 = getChar(<6>c)
   while (_pred1) {
      <7>characters = characters + 1
      <8>_pred2 = c == '\n'
      if (_pred2)
         <9>lines = lines + 1
      <10>_pred3 = isLetter(c)
      if (_pred3) {
         <11>_pred4 = inword == 0
         if (_pred4) {
            <12>words = words + 1
         }
         <13>inword = 1
      }
      else
         <14>inword = 0
      <15>_pred1 = getChar(<16>c)
   }
}
def isLetter(<17>c) {
   <18>_pred5 = ((c >= 'A' && c <= 'Z')
      || (c >= 'a' && c <= 'z'))
   if (_pred5)
      <19>_ret = True
   else
      <20>_ret = False
   return _ret
}
```



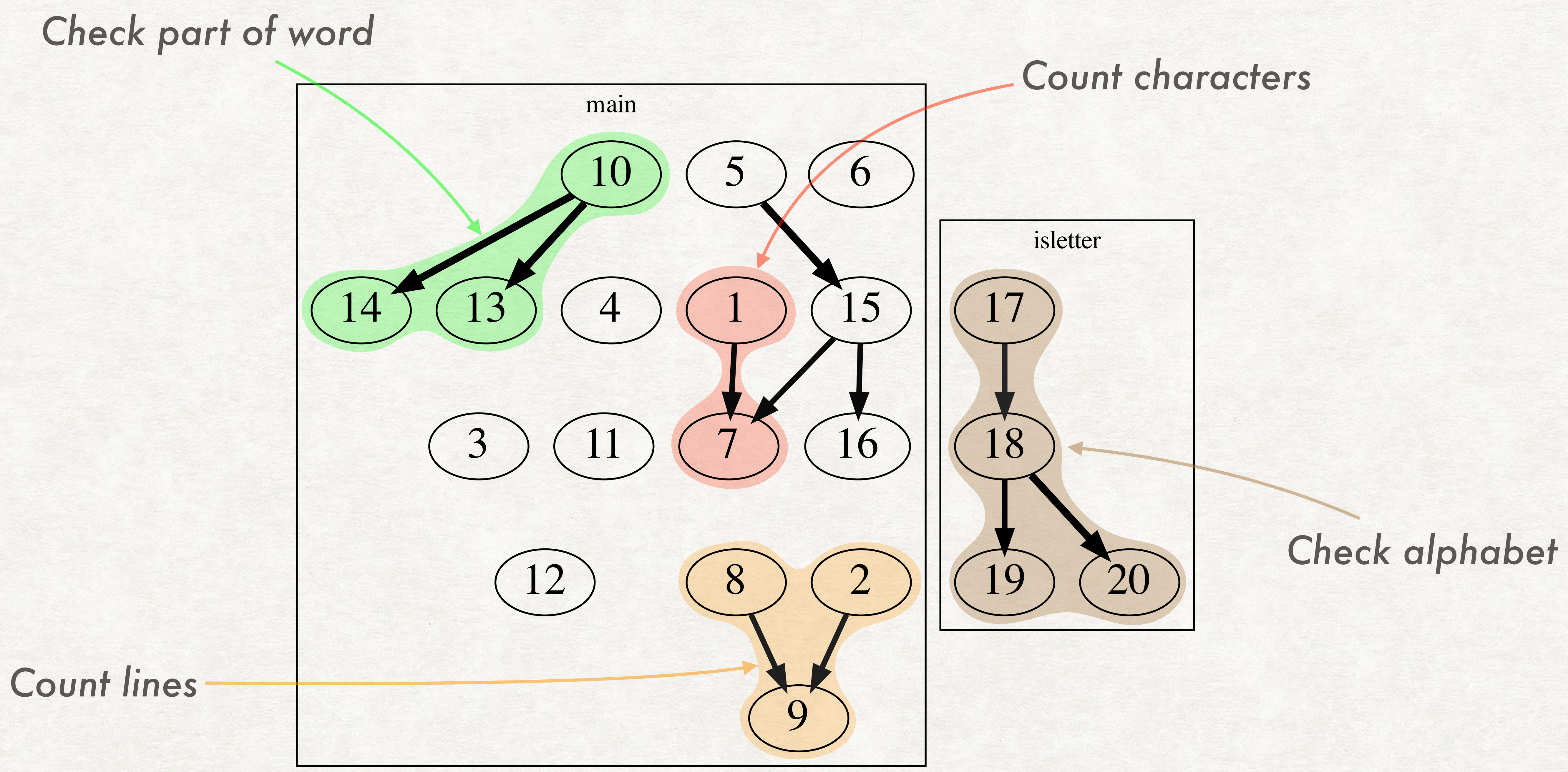
CPDM

SCENARIO1. CPDM VS PDG

Node index

```
def main() {
    <1>characters = 0
    <2>lines = 0
    <3>words = 0
    <4>inword = 0
    <5>_pred1 = getChar(<6>c)
    while (_pred1) {
        <7>characters = characters + 1
        <8>_pred2 = c == '\n'
        if (_pred2)
            <9>lines = lines + 1
        <10>_pred3 = isLetter(c)
        if (_pred3) {
            <11>_pred4 = inword == 0
            if (_pred4) {
                <12>words = words + 1
            }
            <13>inword = 1
        }
        else
            <14>inword = 0
        <15>_pred1 = getChar(<16>c)
    }
}
def isLetter(<17>c) {
    <18>_pred5 = ((c >= 'A' && c <= 'Z')
        || (c >= 'a' && c <= 'z'))
    if (_pred5)
        <19>_ret = True
    else
        <20>_ret = False
    return _ret
}
```

Dependence always happens



CPDM ($DD \geq 0.8$)

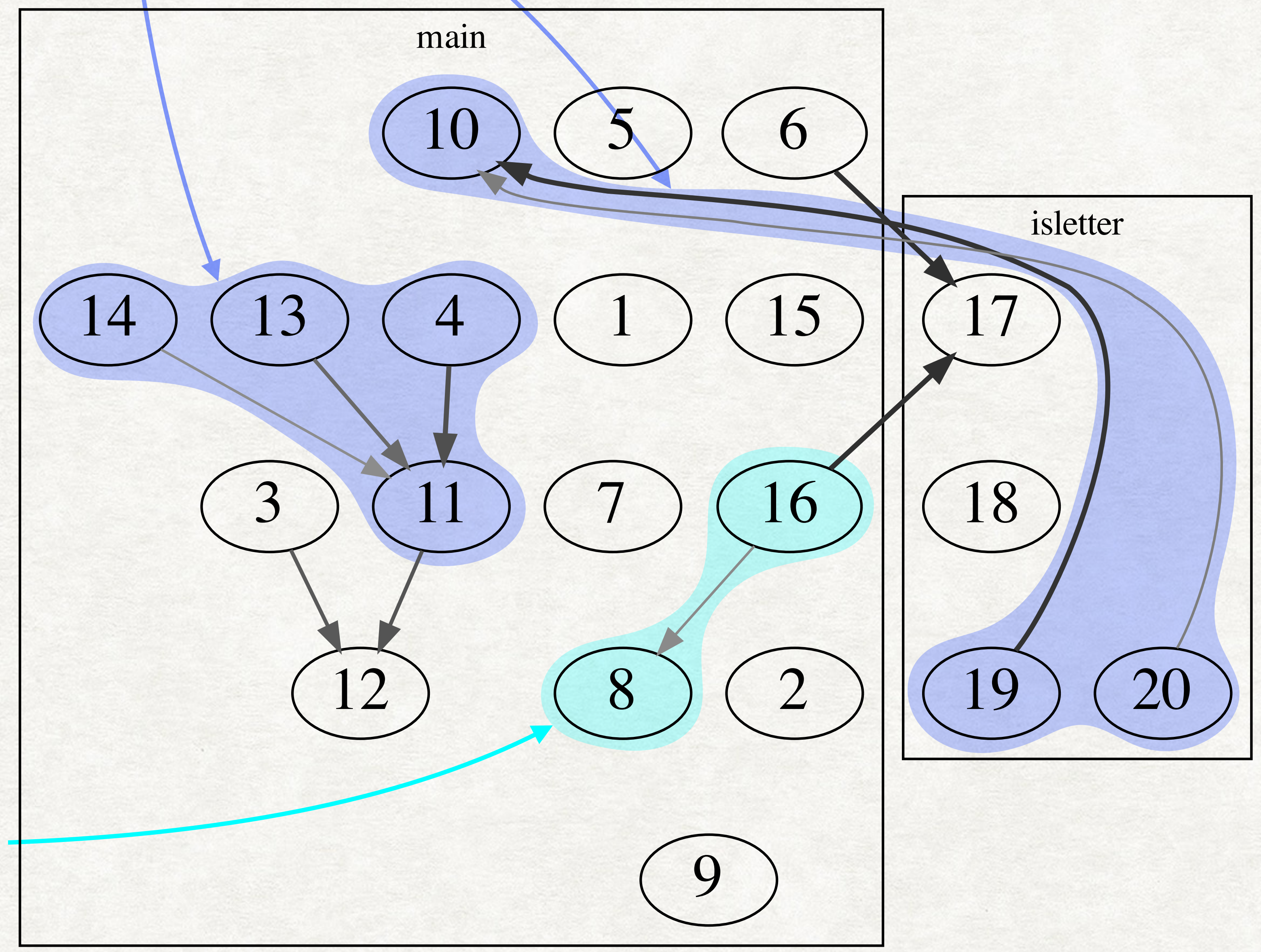
SCENARIO1. CPDM VS PDG

Node index

```
def main() {
    <1>characters = 0
    <2>lines = 0
    <3>words = 0
    <4>inword = 0
    <5>_pred1 = getChar(<6>c)
    while (_pred1) {
        <7>characters = characters + 1
        <8>_pred2 = c == '\n'
        if (_pred2)
            <9>lines = lines + 1
        <10>_pred3 = isLetter(c)
        if (_pred3) {
            <11>_pred4 = inword == 0
            if (_pred4) {
                <12>words = words + 1
            }
            <13>inword = 1
        }
        else
            <14>inword = 0
        <15>_pred1 = getChar(<16>c)
    }
}
def isLetter(<17>c) {
    <18>_pred5 = ((c >= 'A' && c <= 'Z')
        || (c >= 'a' && c <= 'z'))
    if (_pred5)
        <19>_ret = True
    else
        <20>_ret = False
    return _ret
}
```

Dependence occasionally happens

Some tests have a one word, while some have multiple words



Some tests have a single line, while some have multiple lines

CPDM $(0.2 \leq DD < 0.8)$

SCENARIO1. CPDM VS PDG

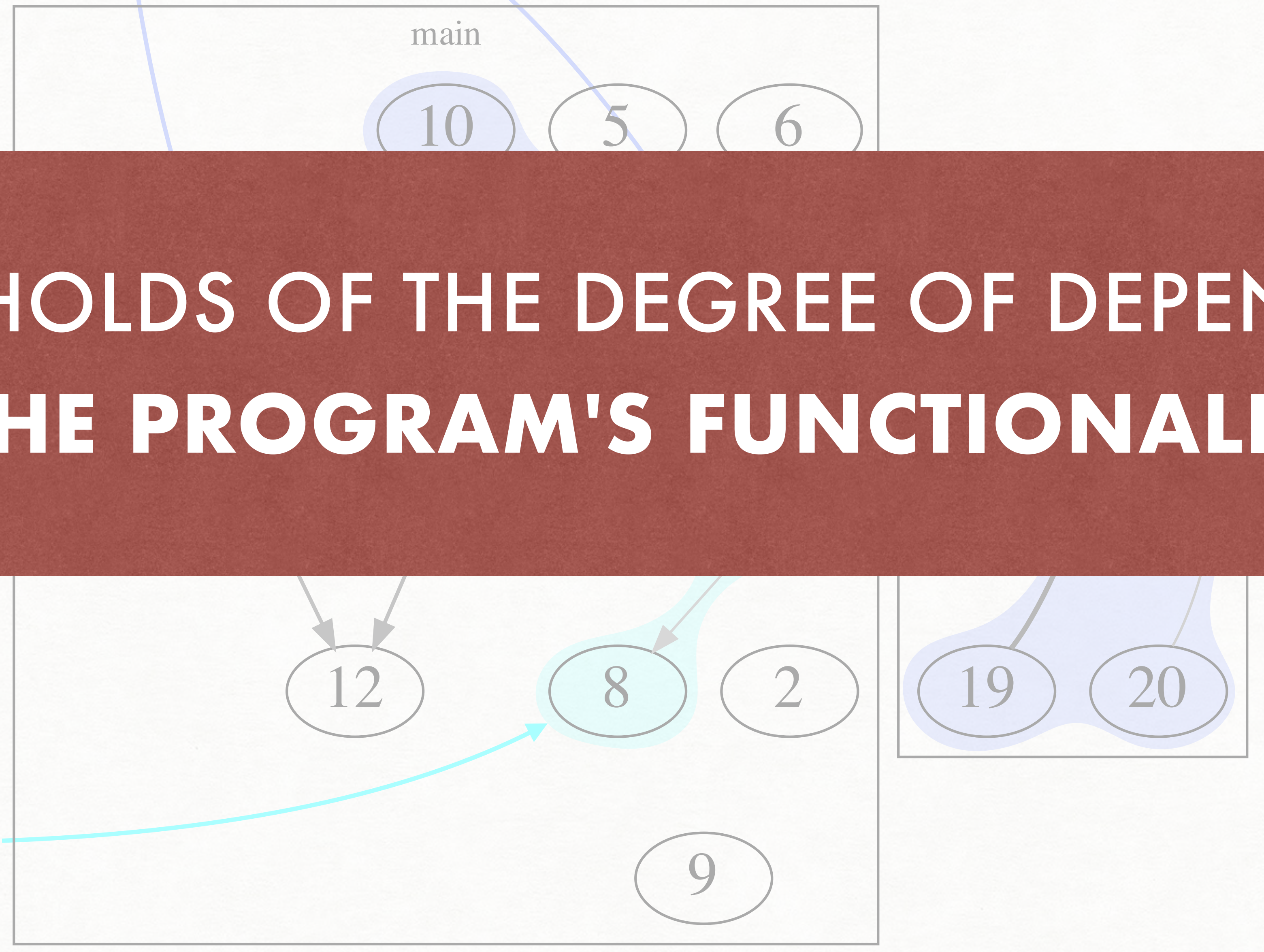
Node index

```
def main() {
    <1>characters = 0
    <2>lines = 0
    <3>words = 0
    <4>inword = 0
    <5>_pred1 = getChar(<6>c)
    while (_pred1) {
        <7>characters = characters + 1
        <8>_pred2 = c == '\n'
```

```
            <13>inword = 1
        }
        else
            <14>inword = 0
        <15>_pred1 = getChar(<16>c)
    }
}
def isLetter(<17>c) {
    <18>_pred5 = ((c >= 'A' && c <= 'Z')
        || (c >= 'a' && c <= 'z'))
    if (_pred5)
        <19>_ret = True
    else
        <20>_ret = False
    return _ret
}
```

Dependence occasionally happens

Some tests have a one word, while some have multiple words

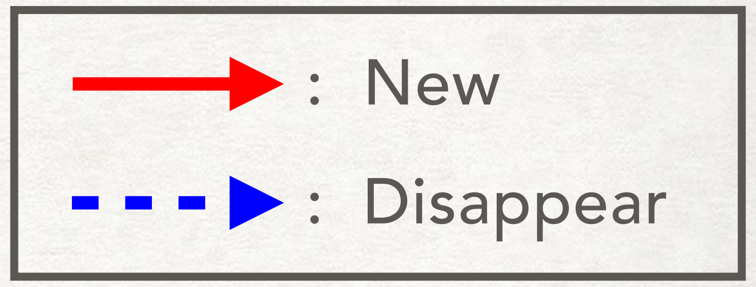


Some tests have a single line, while some have multiple lines

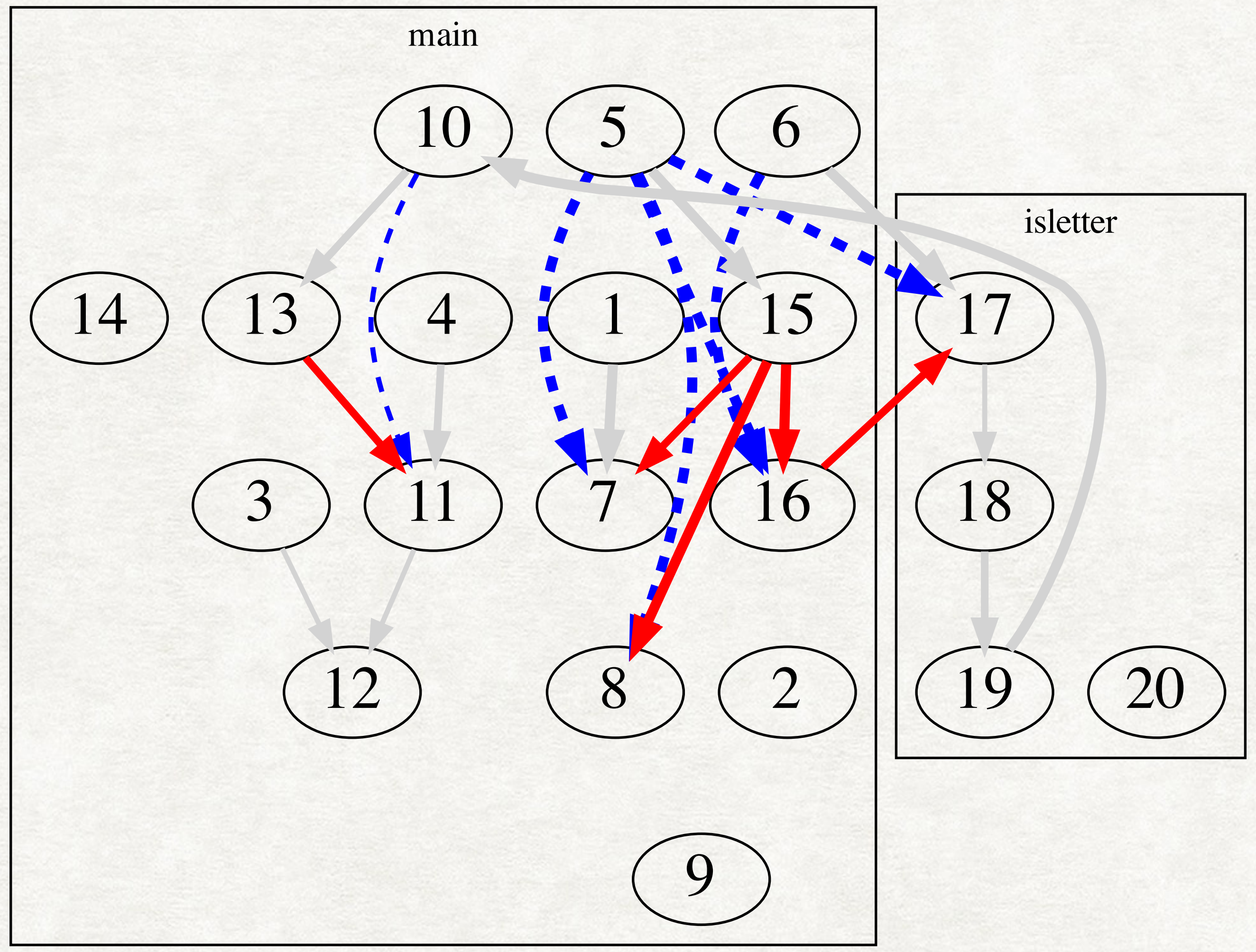
CPDM $(0.2 \leq DD < 0.8)$

BY LOOKING AT THE DIFFERENT THRESHOLDS OF THE DEGREE OF DEPENDENCE, CPDM CAN AID IN **GROUPING THE PROGRAM'S FUNCTIONALITY**.

SCENARIO 2: DIFFERENT EXECUTIONS

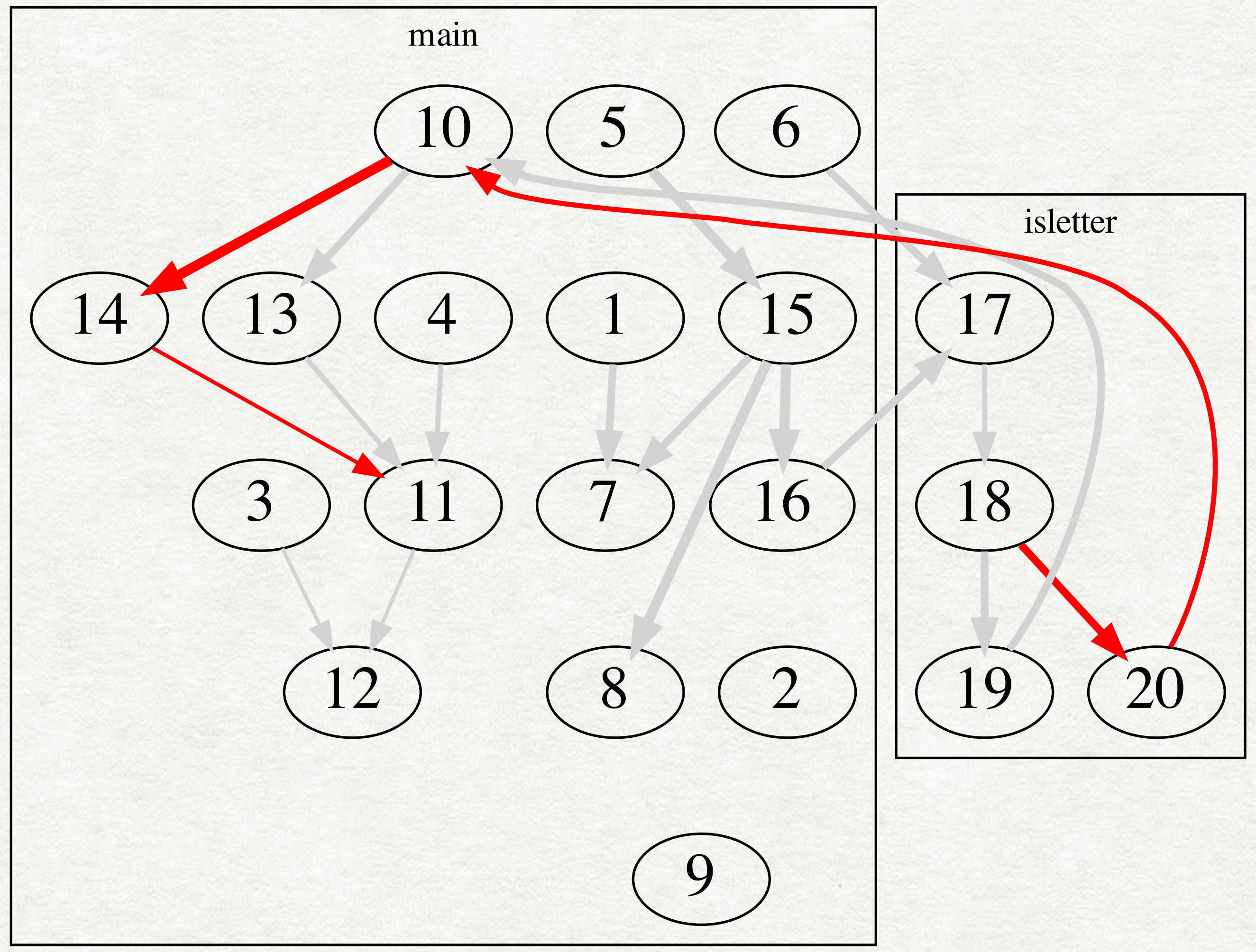


One char TS → Multiple chars TS



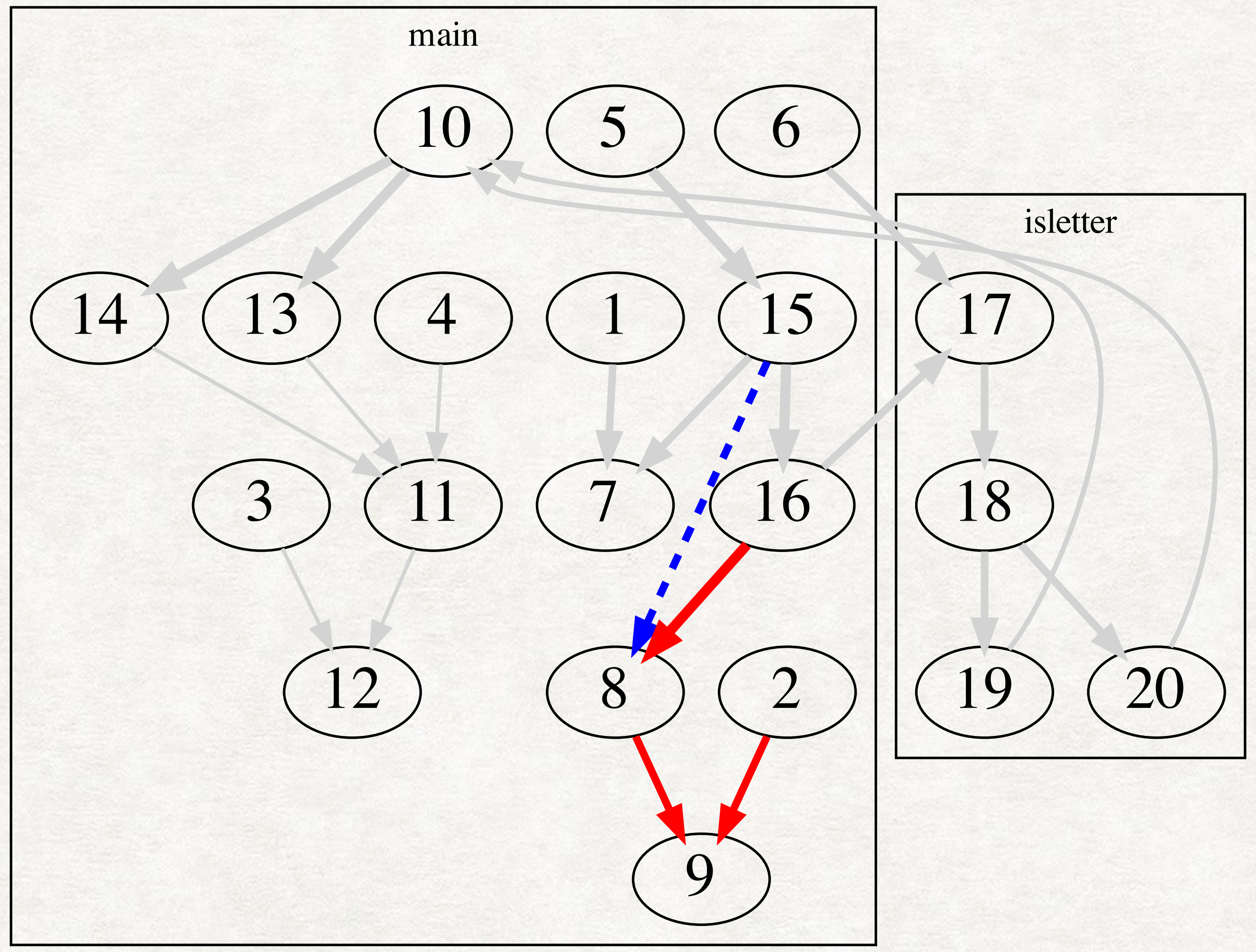
[15], [16], and [13] affects others only if the input contains multiple characters.

One word TS → Multiple words TS



[20] → [10] → [14] → [11] checks if current character is a non-alphabet.

One line TS → Multiple line TS



[2], [8], [9], [16] composes a logic calculating the number of lines.

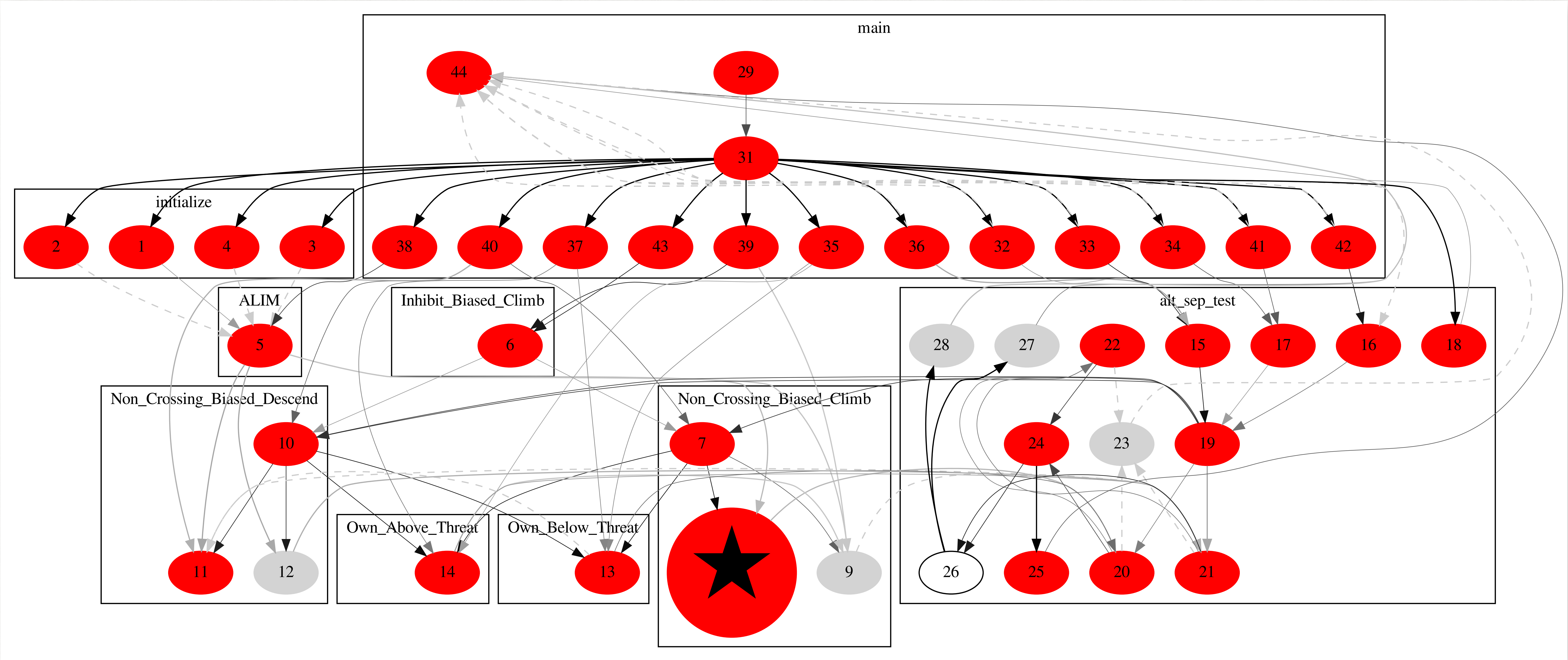
SCENARIO 3: DEBUGGING

```
bool Non_Crossing_Biased_Climb() {
  upward_preferred = Inhibit_Biased_Climb() > Down_Separation;
  if (upward_preferred)
    result = !(Own_Below_Threat()) || ((Own_Below_Threat())
      && (!(Down_Separation > ALIM())));  // bug: > should be >=
  else ...
  return result;
} ...
```

Faulty code in TCAS buggy version 1

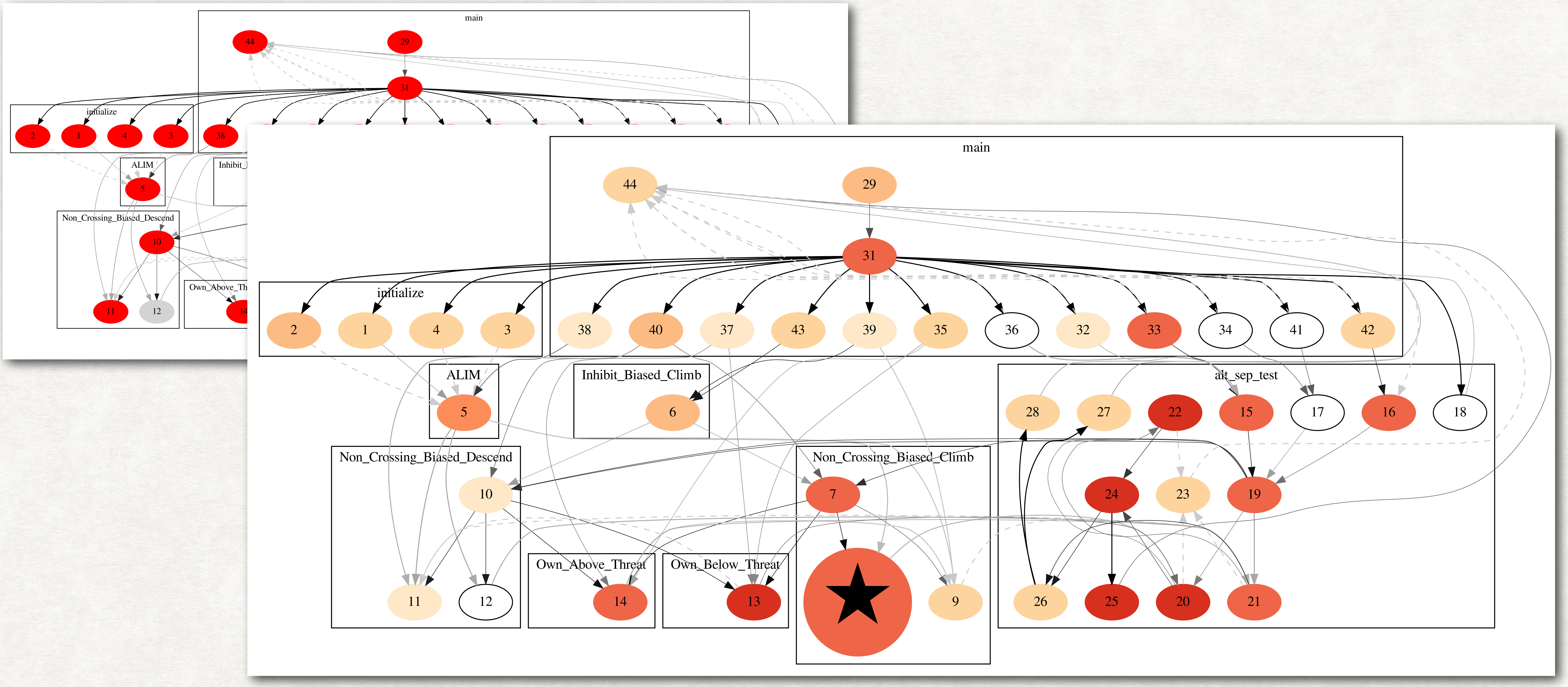
SCENARIO 3: DEBUGGING

Conventional binary dependency analysis (dynamic slicing) shows,



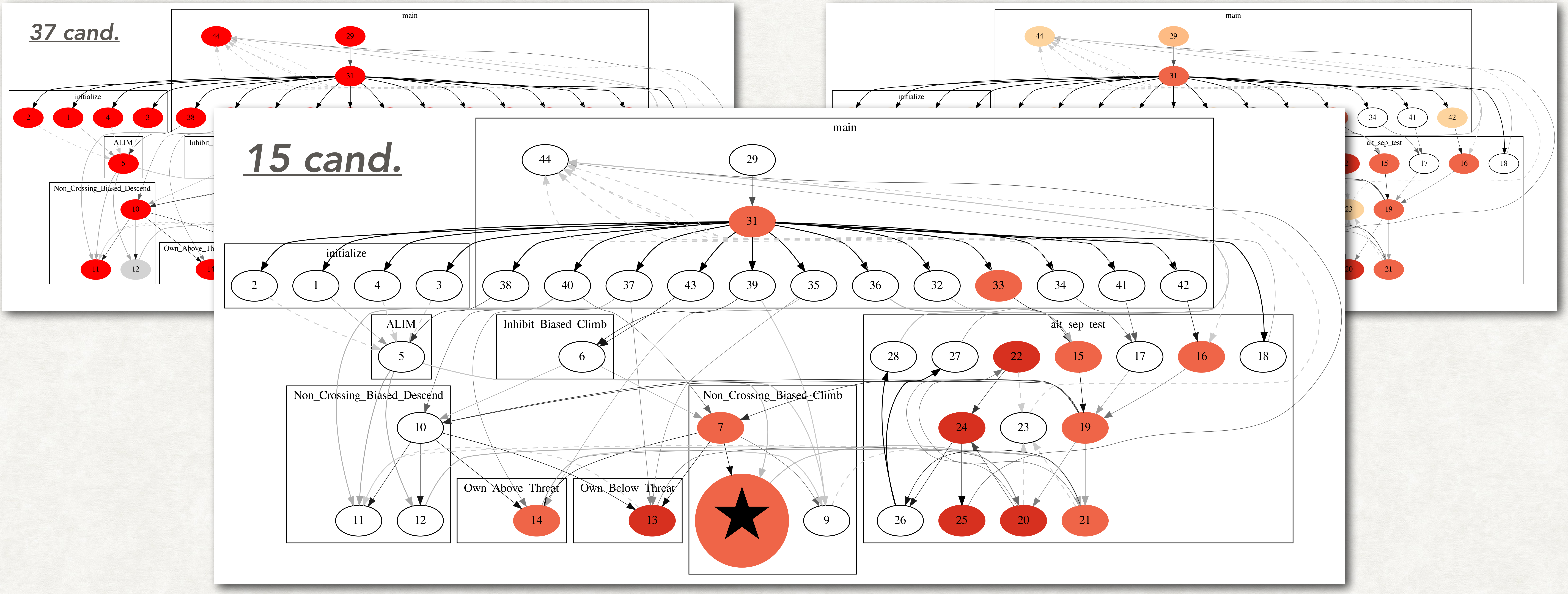
DYNAMIC SLICE OF THE WRONG OUTCOME

SCENARIO 3: DEBUGGING



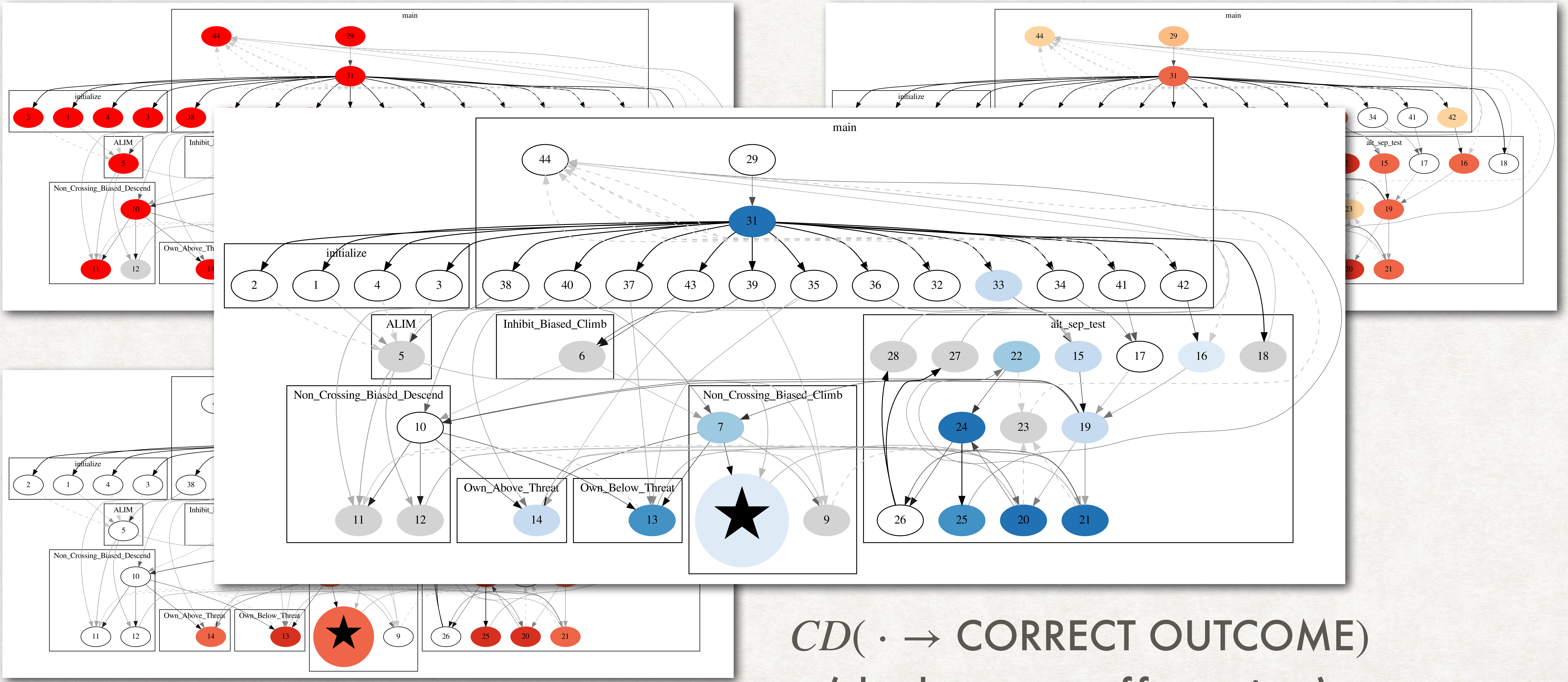
$CD(\,\cdot \rightarrow$ WRONG OUTCOME$)$ (darkness ~ effect size)

SCENARIO 3: DEBUGGING



ONLY $CD(\cdot \to \text{WRONG OUTCOME}) > 0.5$

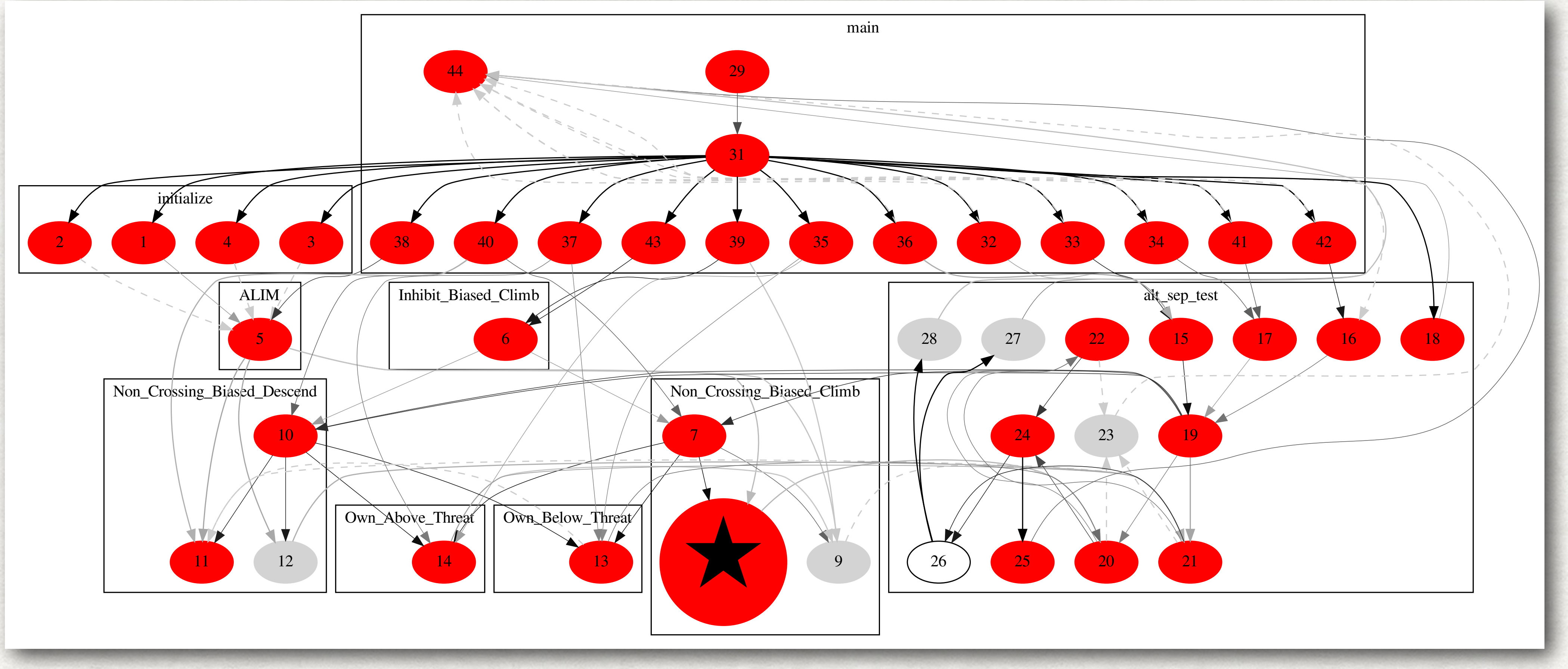
SCENARIO 3: DEBUGGING

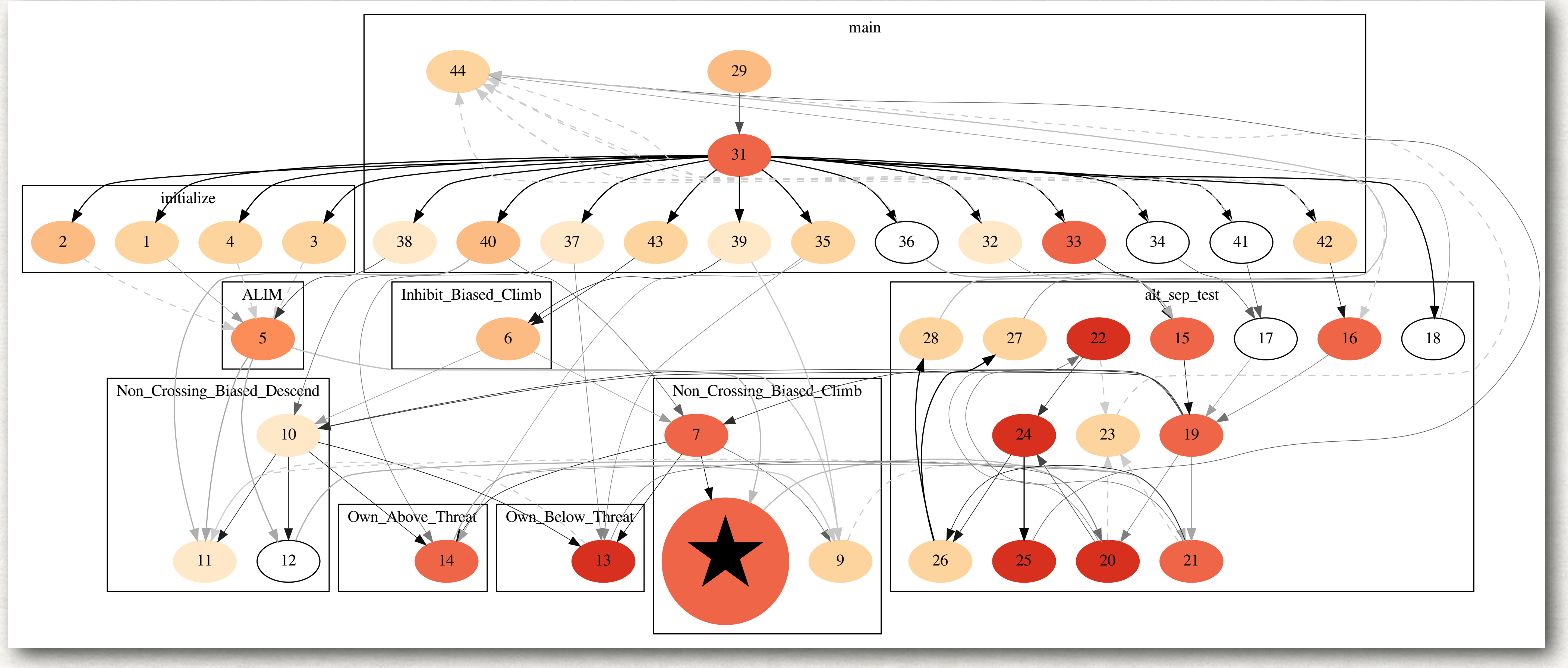


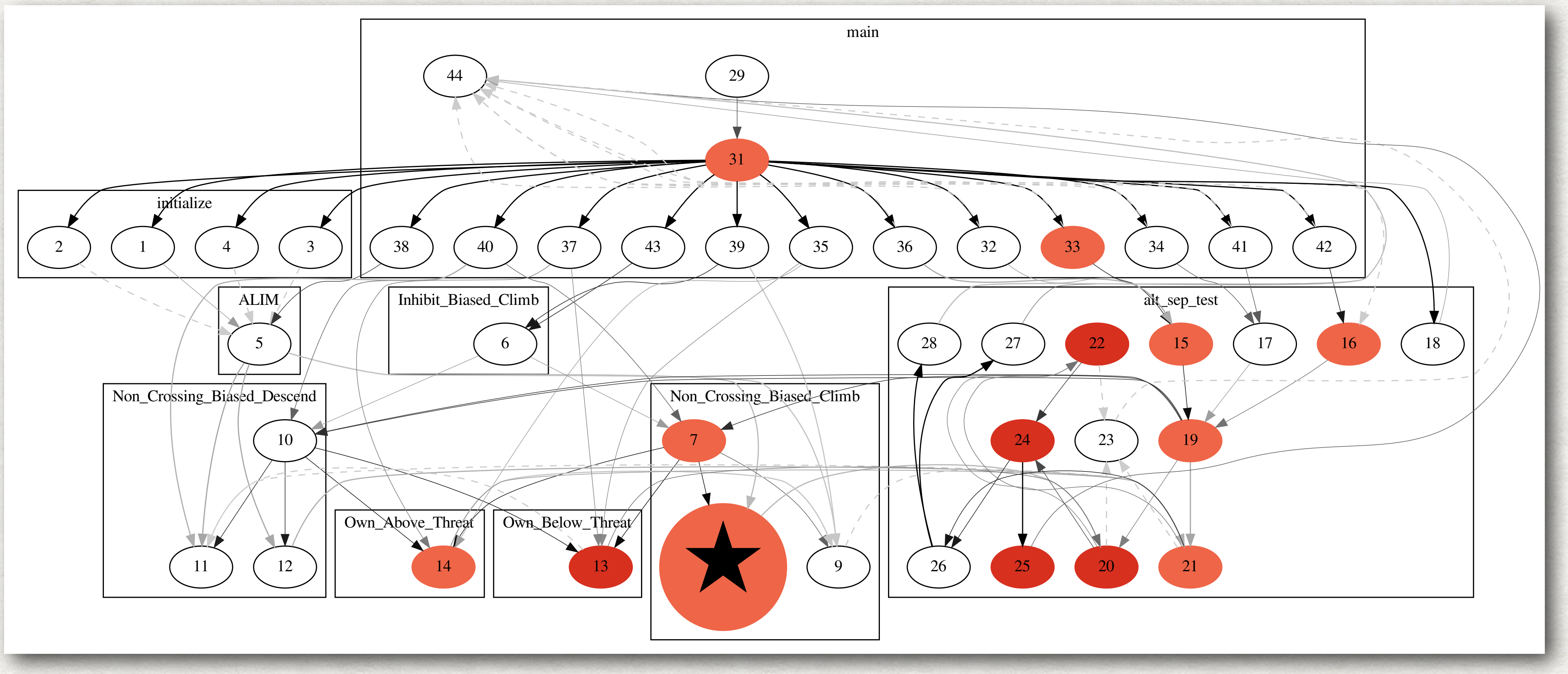
CD(· → CORRECT OUTCOME)

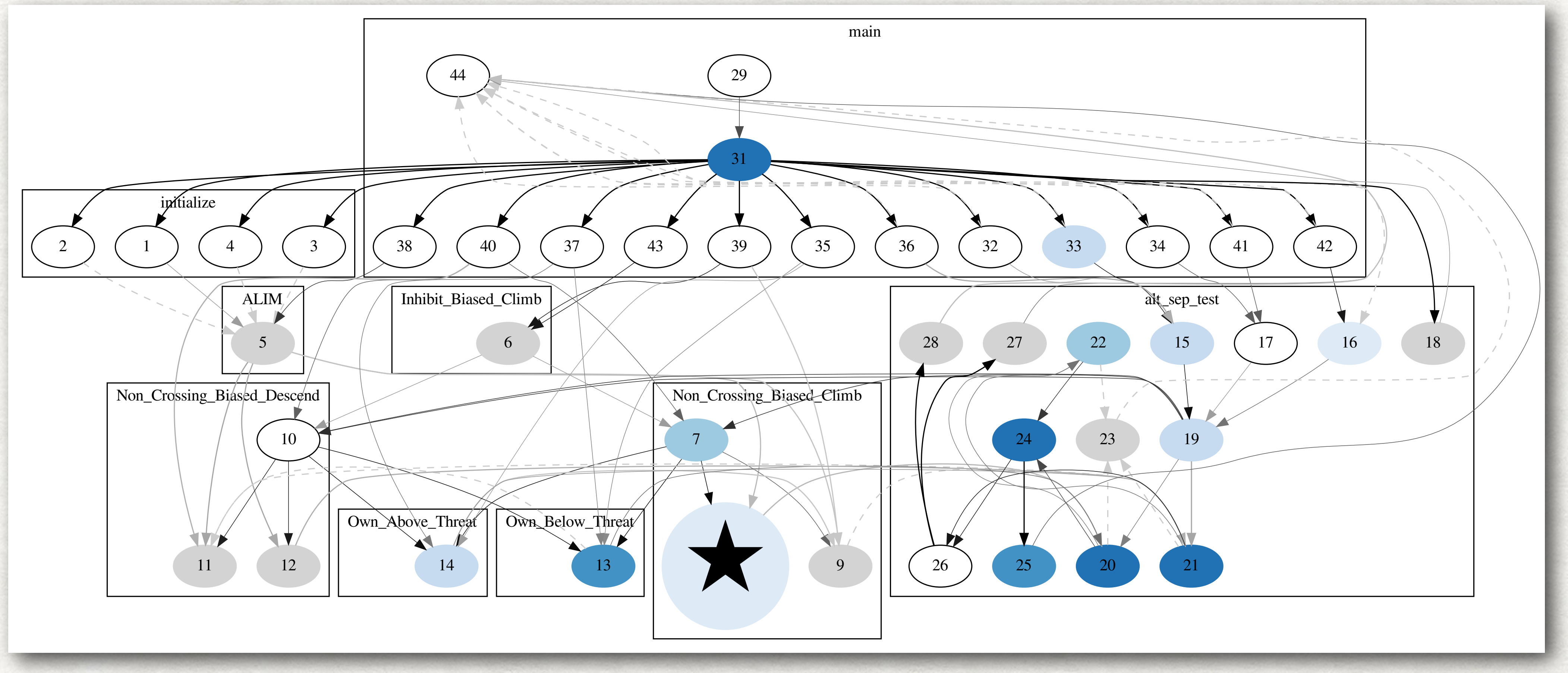
(darkness ~ effect size)

SCENARIO 3: DEBUGGING









SCENARIO 3: DEBUGGING

- *Causal Dependence based Fault Localization*

Suspiciousness: *CD* IN FAILING TEST — *CD* IN PASSING TEST

SCENARIO 3: DEBUGGING

- ***Causal Dependence based Fault Localization***

Suspiciousness: *CD* IN FAILING TEST — *CD* IN PASSING TEST

- **Result:** (Siemens suite, 92 faults)

Acc@n	CDFL	SBFL	Dynamic slicing	Dicing
n = 1	**13**	3	0	3
3	**26**	11	0	7
5	**30**	17	0	7
10	**42**	31	2	7

SCENARIO 3: DEBUGGING

- *Causal Dependence based Fault Localization*

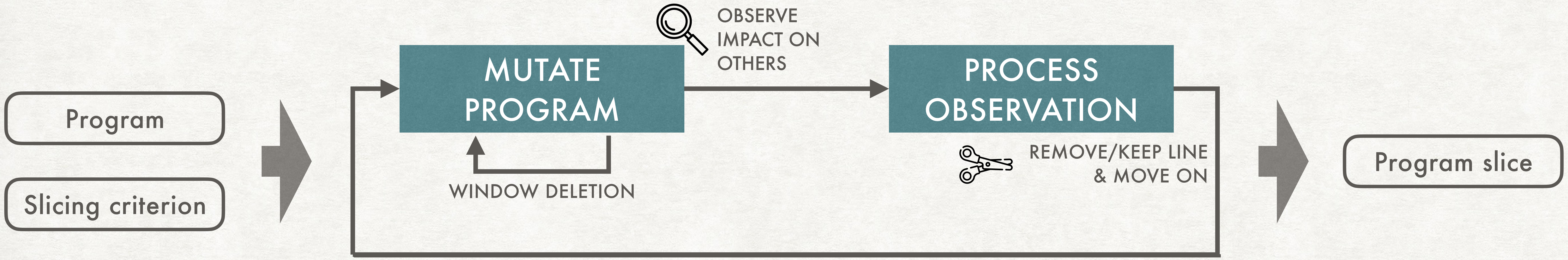
Suspiciousness: *CD* IN FAILING TEST — *CD* IN PASSING TEST

THE FINER GRANULARITY OF DEPENDENCE INFORMATION FROM CPDA CAN **AID THE DEBUGGING PROCESS** MORE THAN THE EXISTING BINARY DEPENDENCE INFORMATION.

Acc@n	CDFL	SBFL	Dynamic slicing	Dicing
n = 1	**13**	3	0	3
3	**26**	11	0	7
5	**30**	17	0	7
10	**42**	31	2	7

Problem statement:

Existing observation-based analysis lacks *scalability* and *interpretability*.



SCALABILITY

- Costly observation
- Partial analysis

COMPREHENSION

- No structural reason
- Binary dependency

Problem statement:

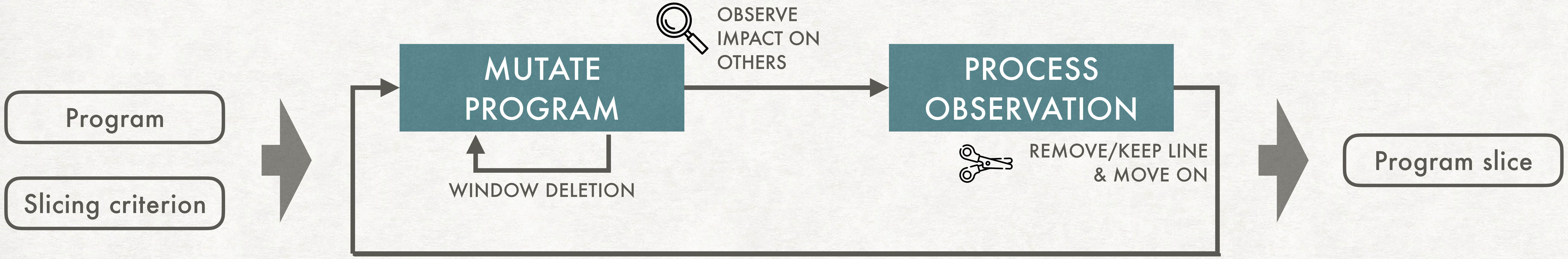
Existing observation-based analysis lacks *scalability* and *interpretability*.

SCALABILITY

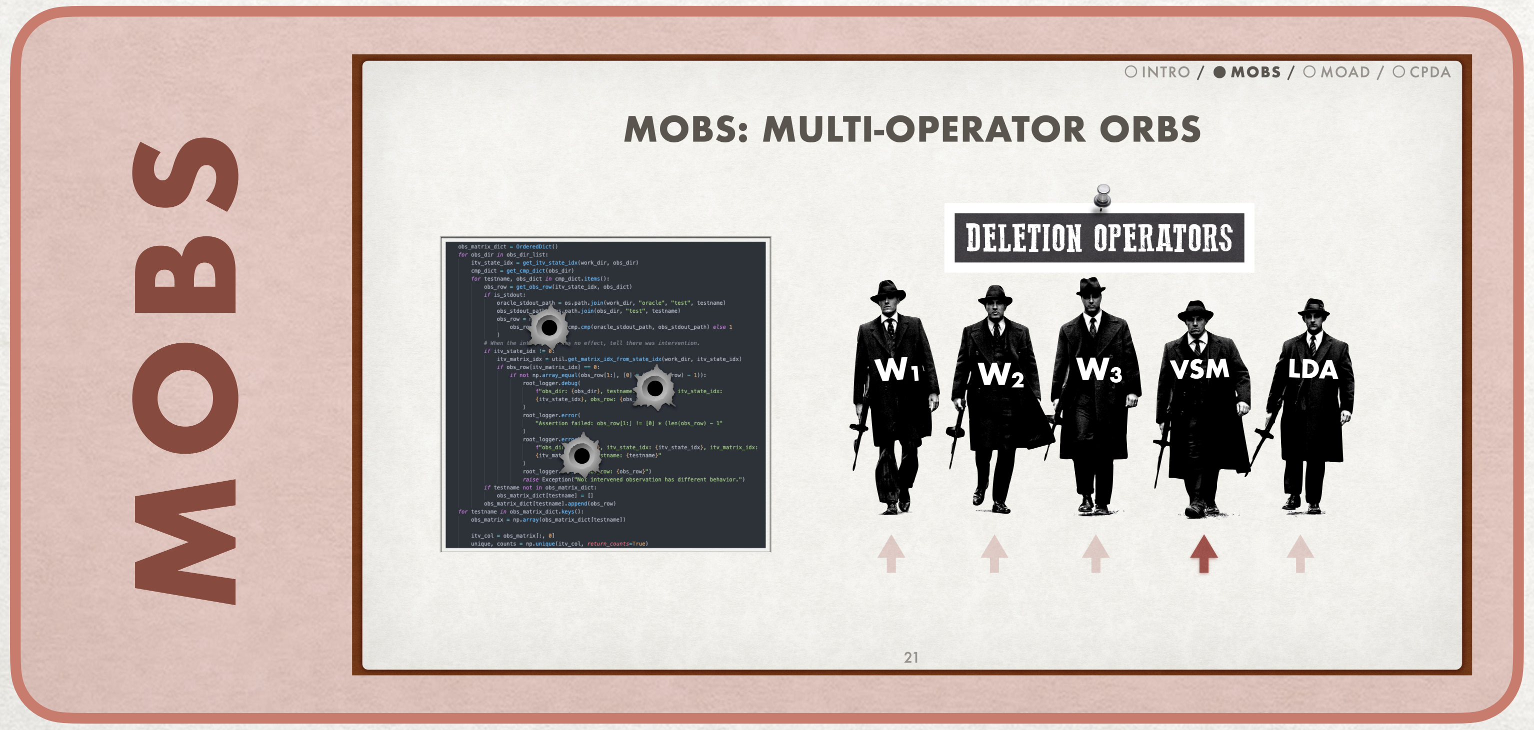
- Costly observation
- Partial analysis

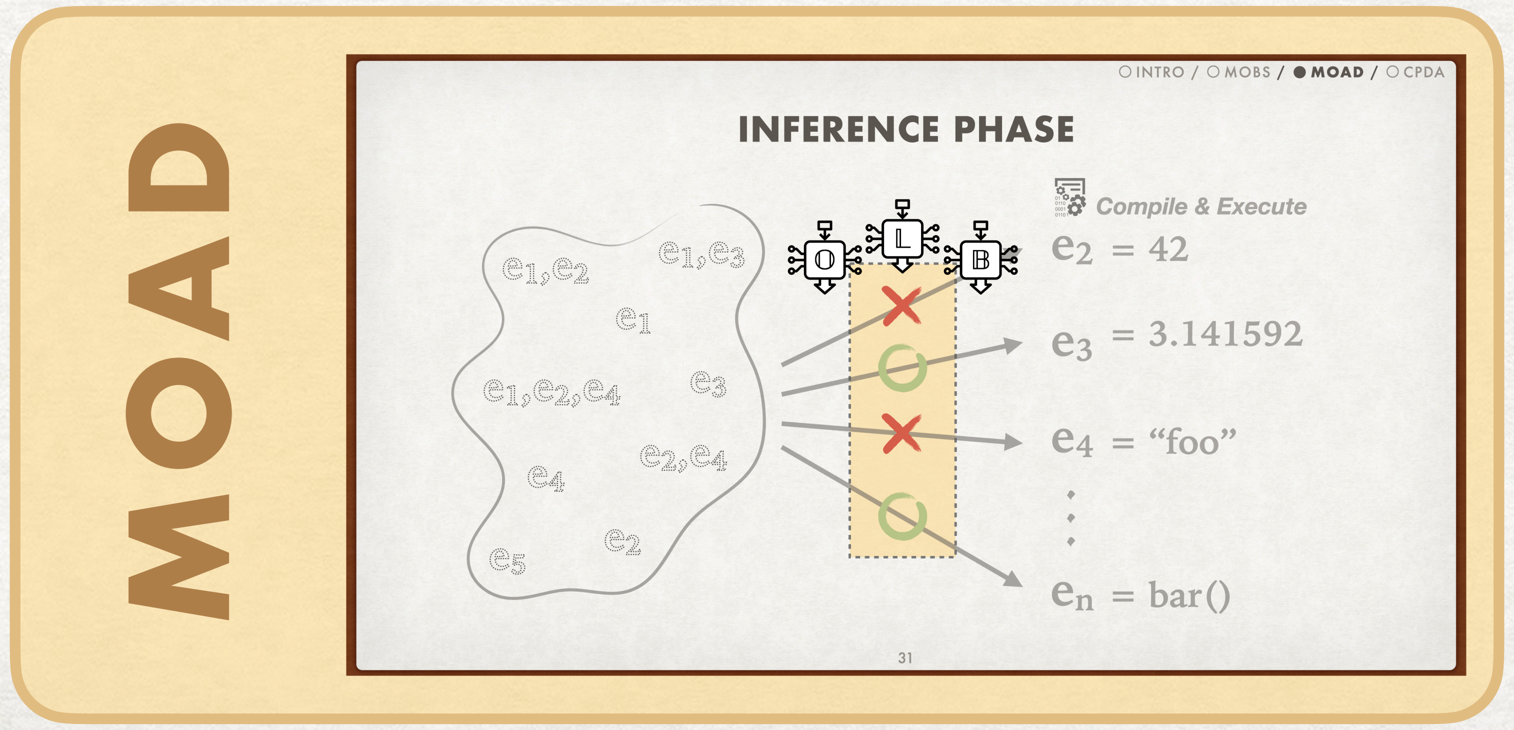
COMPREHENSION

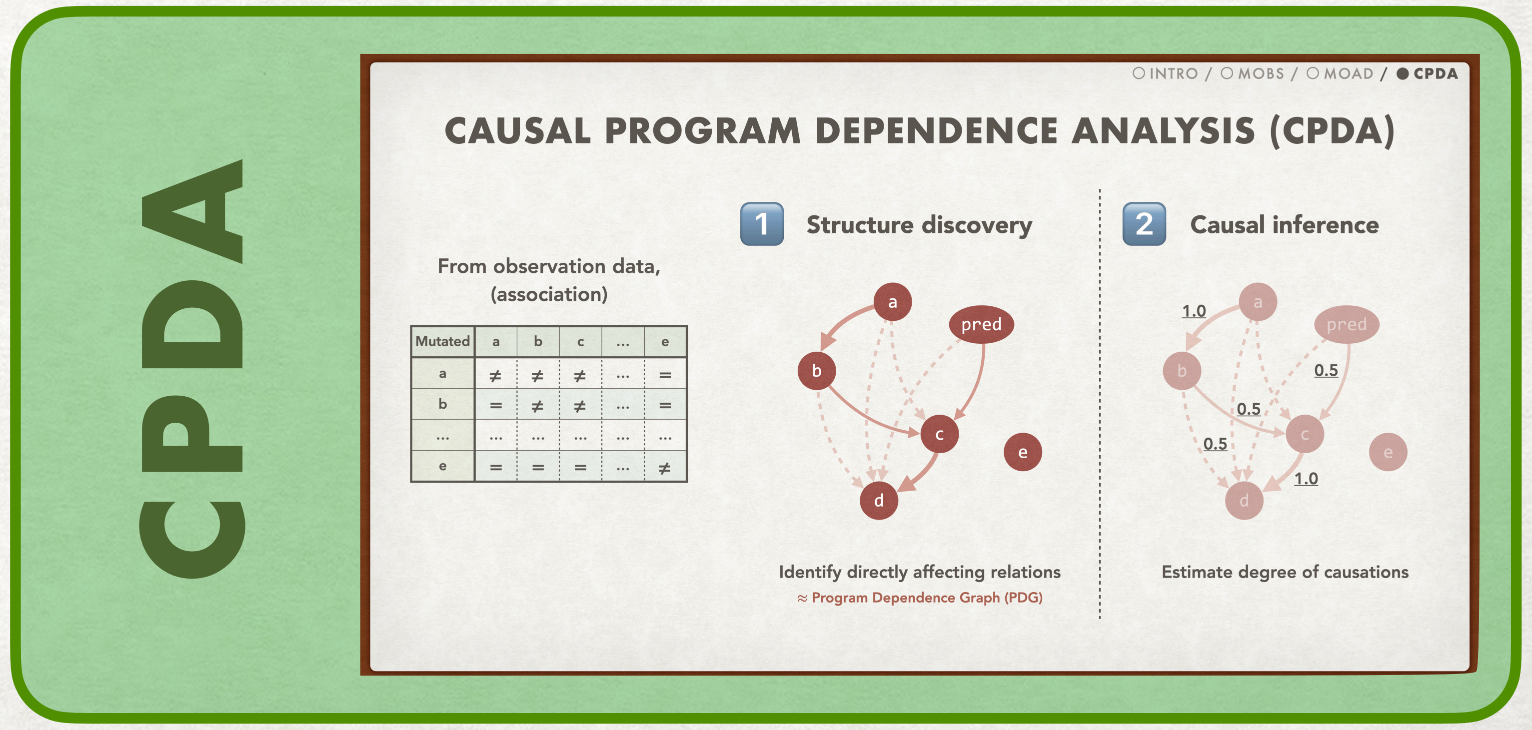
- No structural reason
- Binary dependency



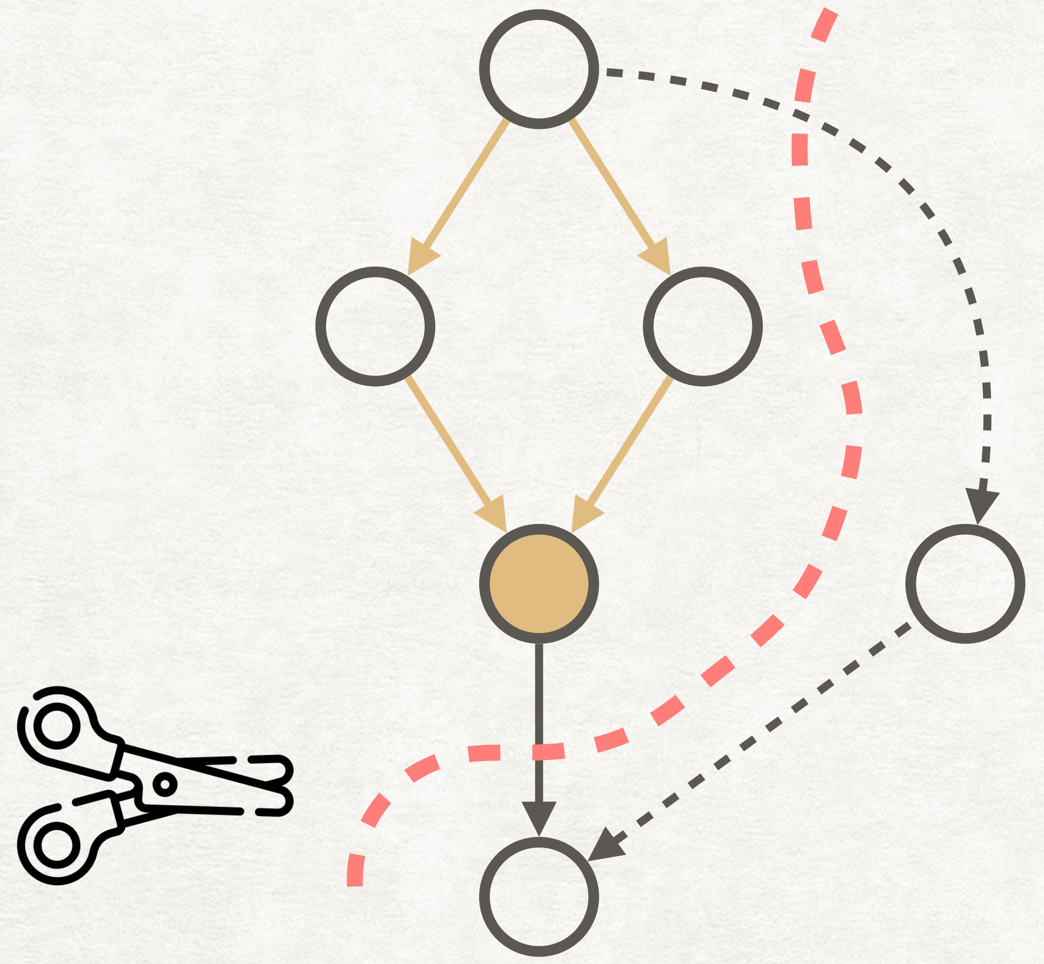
Thesis statement: ***Statistically modeling*** program dependence can improve the scalability and the interpretability of the observation-based analysis.





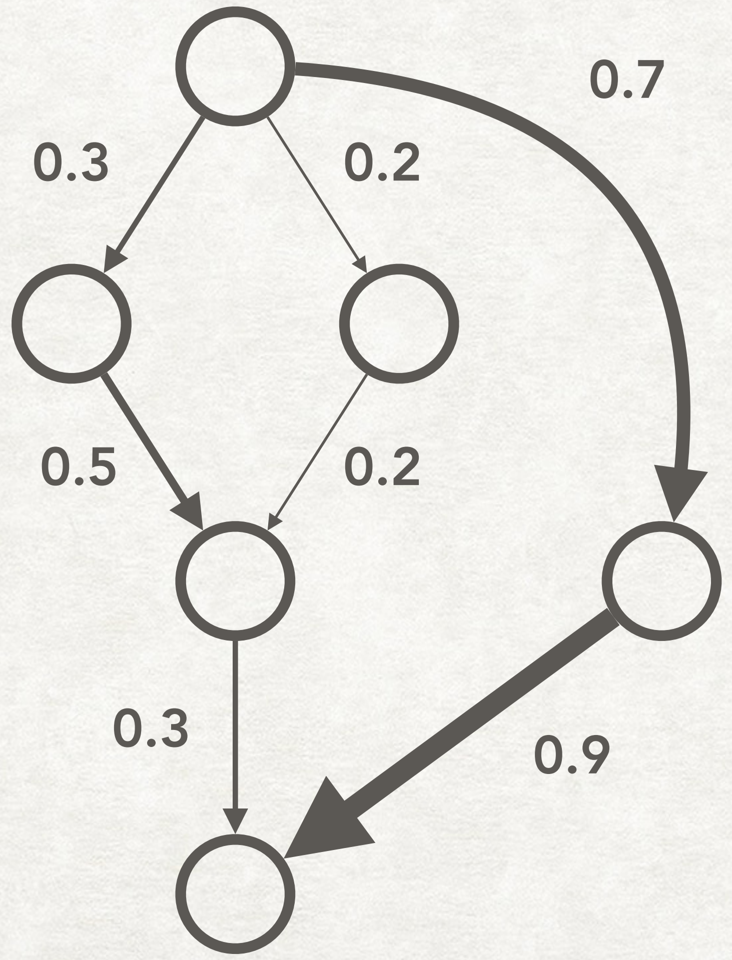


GOAL



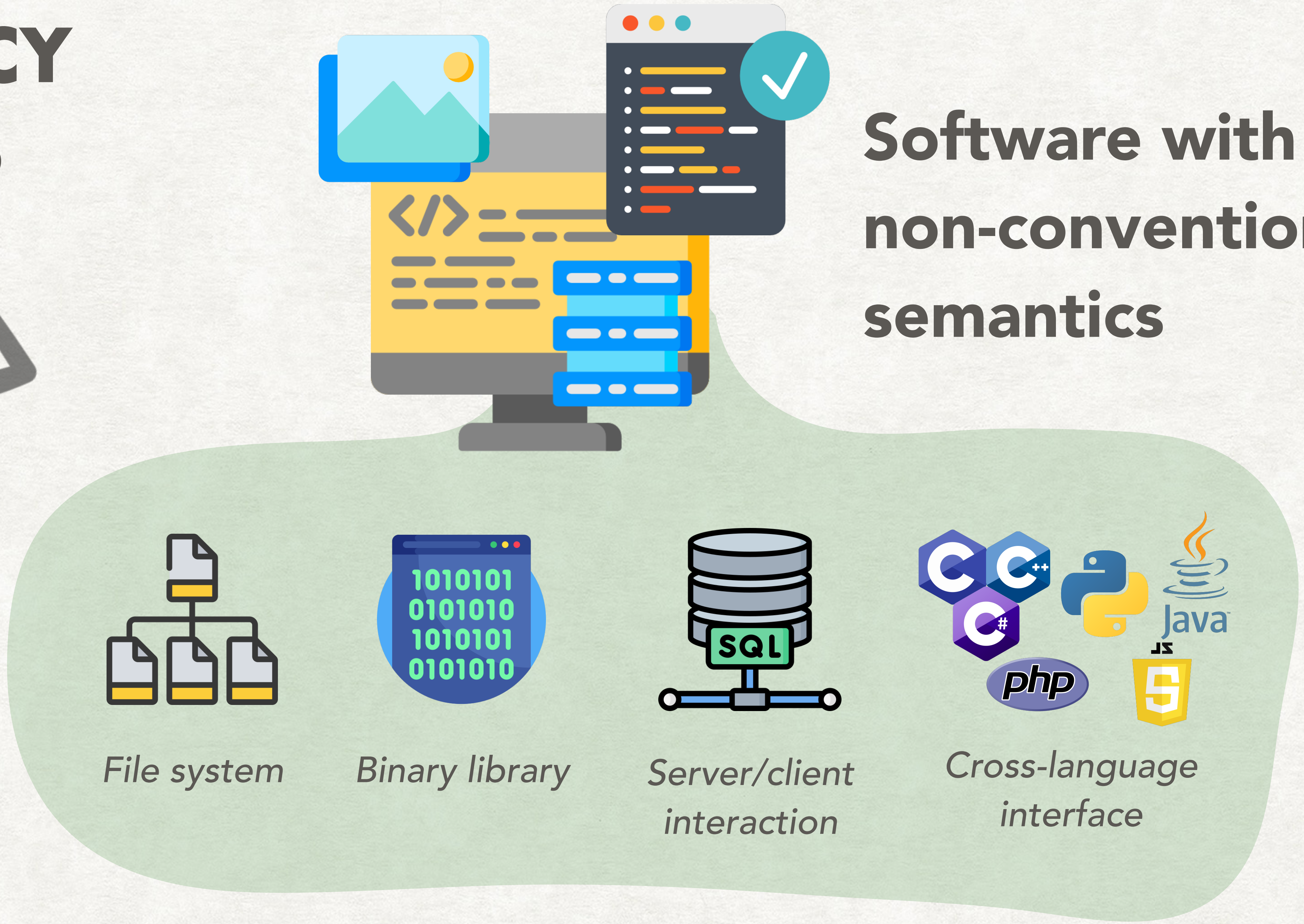
Program slicing technique

Enhance



Dependency analysis technique

DEPENDENCY ANALYSIS



Software with non-conventional semantics

Problem statement:

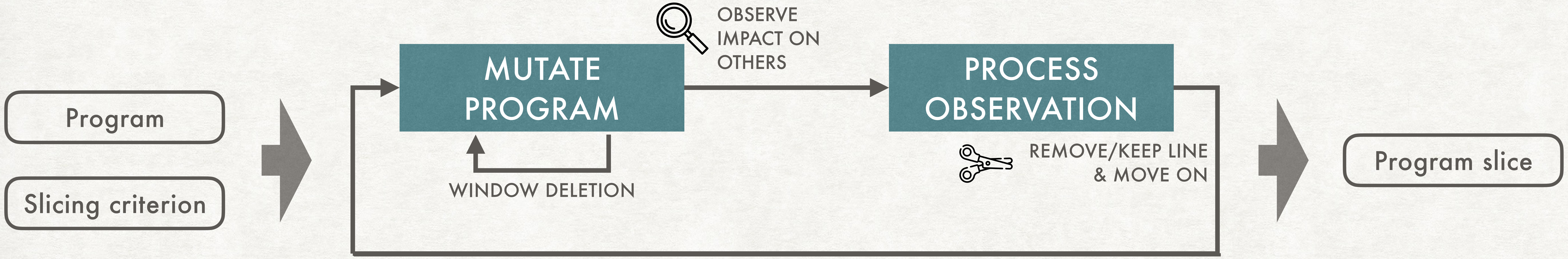
Existing observation-based analysis lacks *scalability* and *interpretability*.

SCALABILITY

- Costly observation
- Partial analysis

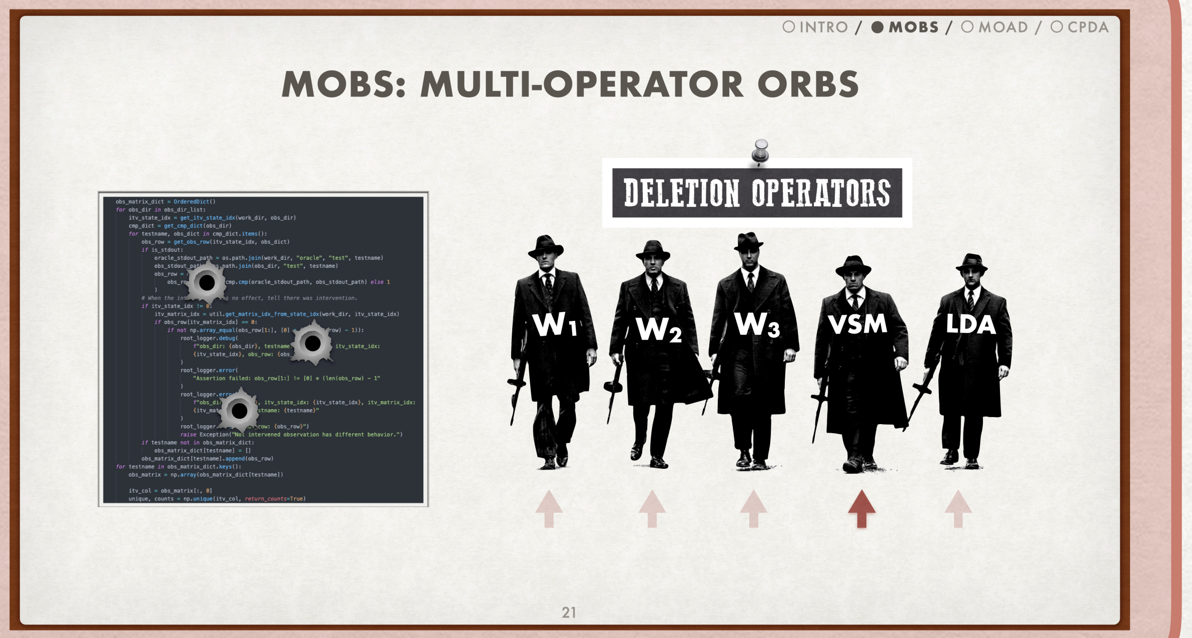
COMPREHENSION

- No structural reason
- Binary dependency

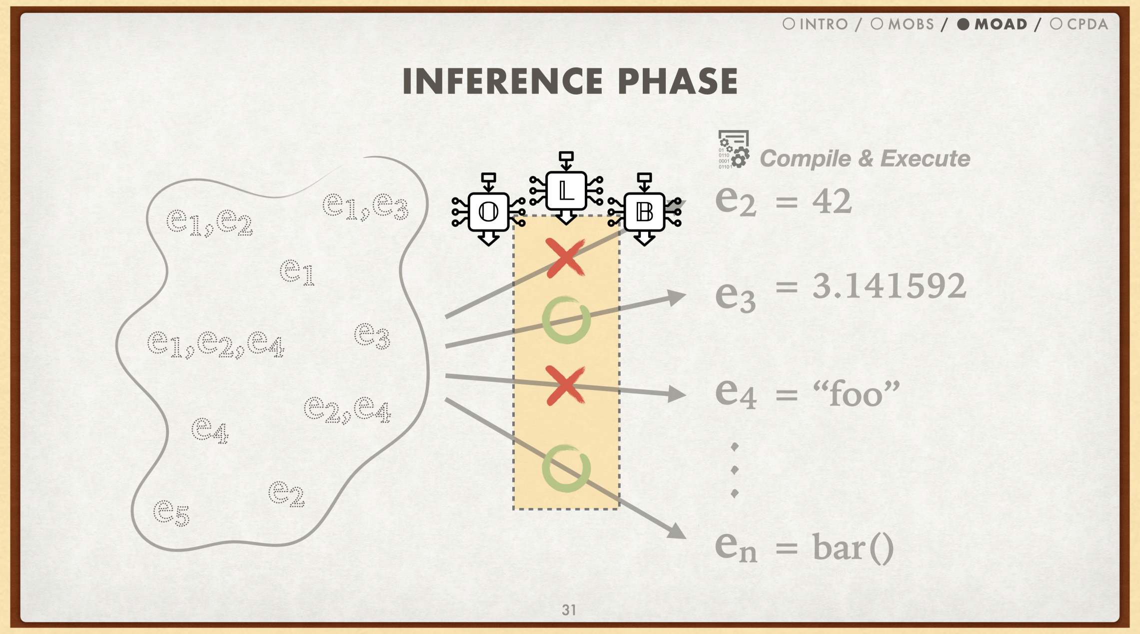


Thesis statement: ***Statistically modeling*** program dependence can improve the scalability and the interpretability of the observation-based analysis.

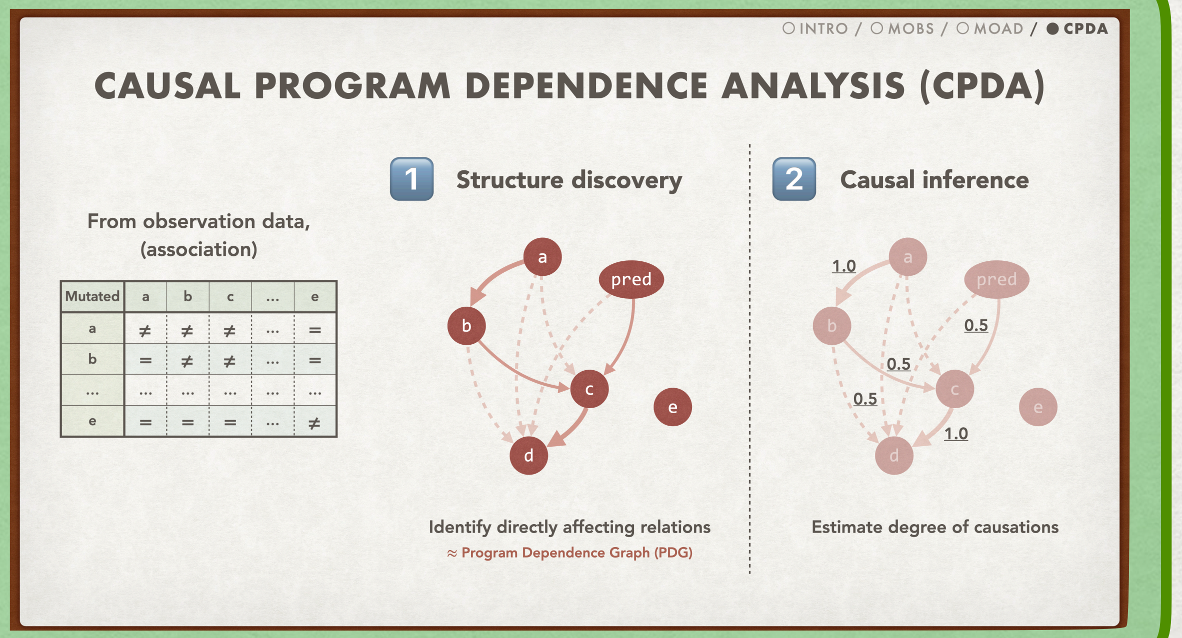
MOBS



MOAD



CPDA



IDLENESS IS AN **APPENDIX** TO NOBILITY.

— Robert Burton

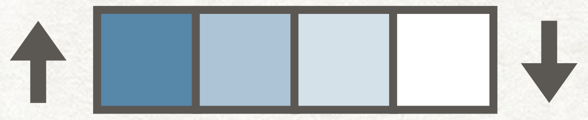
A. OUTPUT OF ORBS AND MOAD

```
int main() {

    int i = 1;
    while (i < 11) {

        i = i + 1;
    }

    printf(“%d\n”, i);
}
```

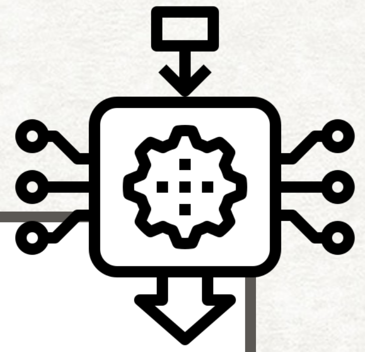
ORBS

- Program slice -

Dependence



```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf(“%d\n”, sum);
    printf(“%d\n”, i);
}
```

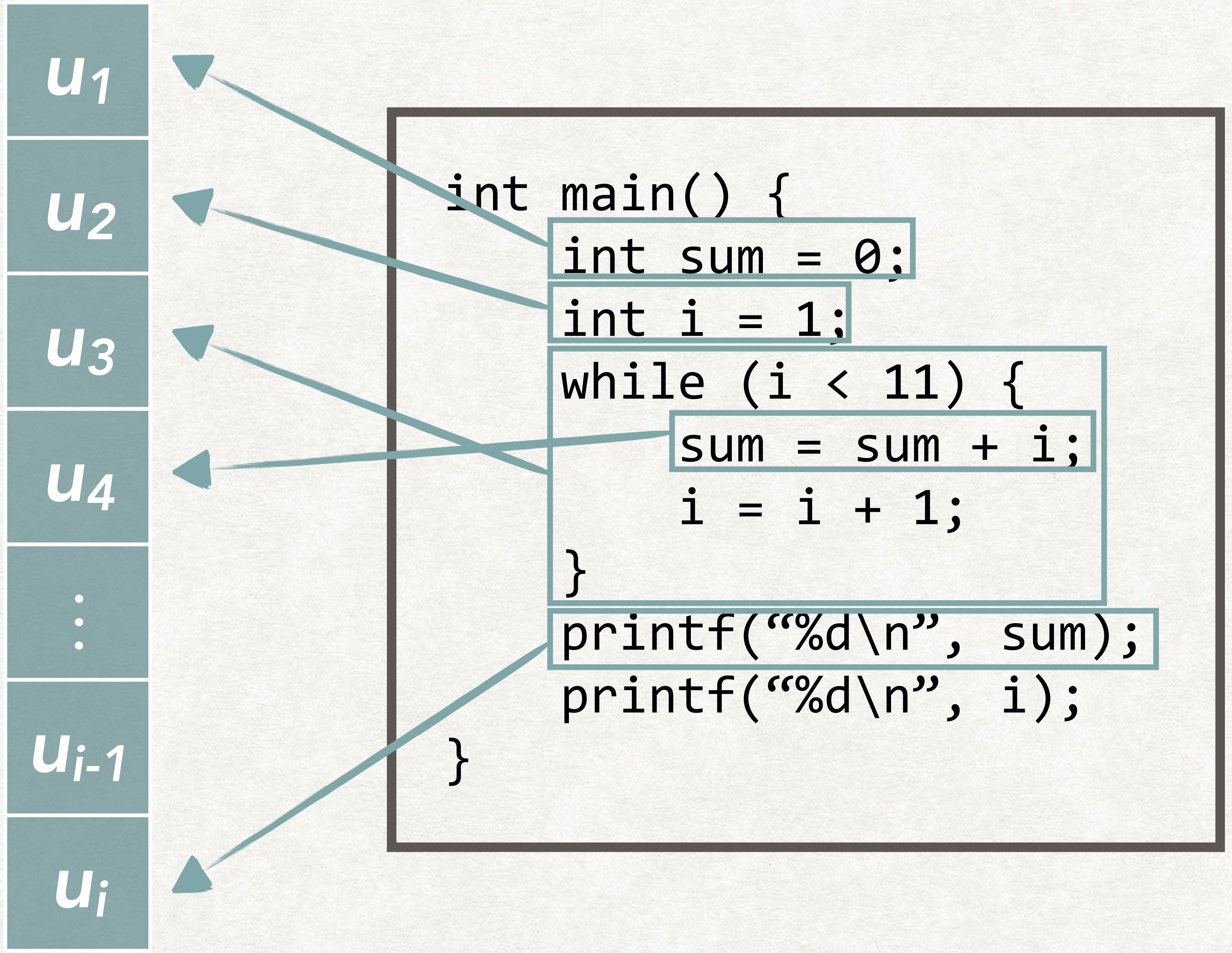


MOAD

- Dependence model -

B. OBSERVATION PHASE

- Identify a set of **deletable units**
  - e.g. all statements



B. OBSERVATION PHASE

- Identify a set of **deletable units**
  - e.g. all statements
- Generate a set of deletion (deleted program) to observe by **deletion generation schemes**
  - 0: remains, 1: deleted

1-hot:

	u_1	u_2	u_3	u_4	…	u_{i-1}	u_i
original ➜	0	0	0	0	…	0	0
	1	0	0	0	…	0	0
	0	1	0	0	…	0	0
	…	…	…	…	…	…	…
	0	0	0	0	…	0	1

+

2-hot:

u_1	u_2	u_3	u_4	…	u_{i-1}	u_i
…	…	…	…	…	…	…
1	1	0	0	…	0	0
1	0	1	0	…	0	0
…	…	…	…	…	…	…
0	0	0	0	…	1	1

B. OBSERVATION PHASE

- Identify a set of **deletable units**
  - e.g. all statements
- Generate a set of deletion (deleted program) to observe by **deletion generation schemes**
  - 0: remains, 1: deleted
- Run the program, check whether the trajectory changed (0) or not (1) for each variable.

u_1	u_2	u_3	u_4	…	u_{i-1}	u_i
0	0	0	0	…	0	0
1	0	0	0	…	0	0
0	1	0	0	…	0	0
…	…	…	…	…	…	…
0	0	0	0	…	1	1

v_1	v_2	v_3	…	v_j
1	1	1	…	1
0	0	0	…	1
1	0	1	…	0
…	…	…	…	…
0	0	1	…	0

C. INFERENCE MODEL

1. Once success ($\mathbb{O}$)
2. **Logistic ($\mathbb{L}$)**

u_1	u_2	...	u_i	v_k
1	0	...	0	0
0	1	...	0	1
...	...	...	...	...
1	0	...	0	0

Coefficients represent the relative impact on dependence

$$\log \frac{v_k}{1 - v_k} = \beta_0 + \beta_1 u_1 + \beta_2 u_2 + \cdots + \beta_i u_i$$

$$\text{If } \frac{\beta_i \le 0}{\beta_i > 0}, \text{ then } \mathbf{u_i} \longleftarrow \mathbf{v_k}$$

If β_i, the coefficient for u_i of the logistic regression for v_k, is larger than 0, then v_k is independent from u_i.

C. INFERENCE MODEL

1. Once success ($\mathbb{O}$)
2. Logistic ($\mathbb{L}$)
3. Bayesian ($\mathbb{B}$)

Estimate with the frequency of behavior preservation

$$
\begin{aligned}
P\left(v_k \mid u_i\right) &= P\left(v_k \text{ behaves the same } \mid \mathrm{u_i} \text{ has been deleted }\right) \\
&= P(v_k = 1 \mid u_i = 1) \\
&= \frac{P(v_k = 1,\ u_i = 1)}{P(u_i = 1)}
\end{aligned}
$$

μ : average of the probability over units

$$
\text{If } \frac{\hat{P}\left(v_k \mid u_i\right) \le \mu}{\hat{P}\left(v_k \mid u_i\right) > \mu}\text{, then } \mathrm{u_i} \xleftarrow[\times]{\bigcirc} \mathrm{v_k}
$$

If the $P(v_k$ behaves the same | u_i has been deleted) is larger than the mean, then v_k Is independent from u_i.

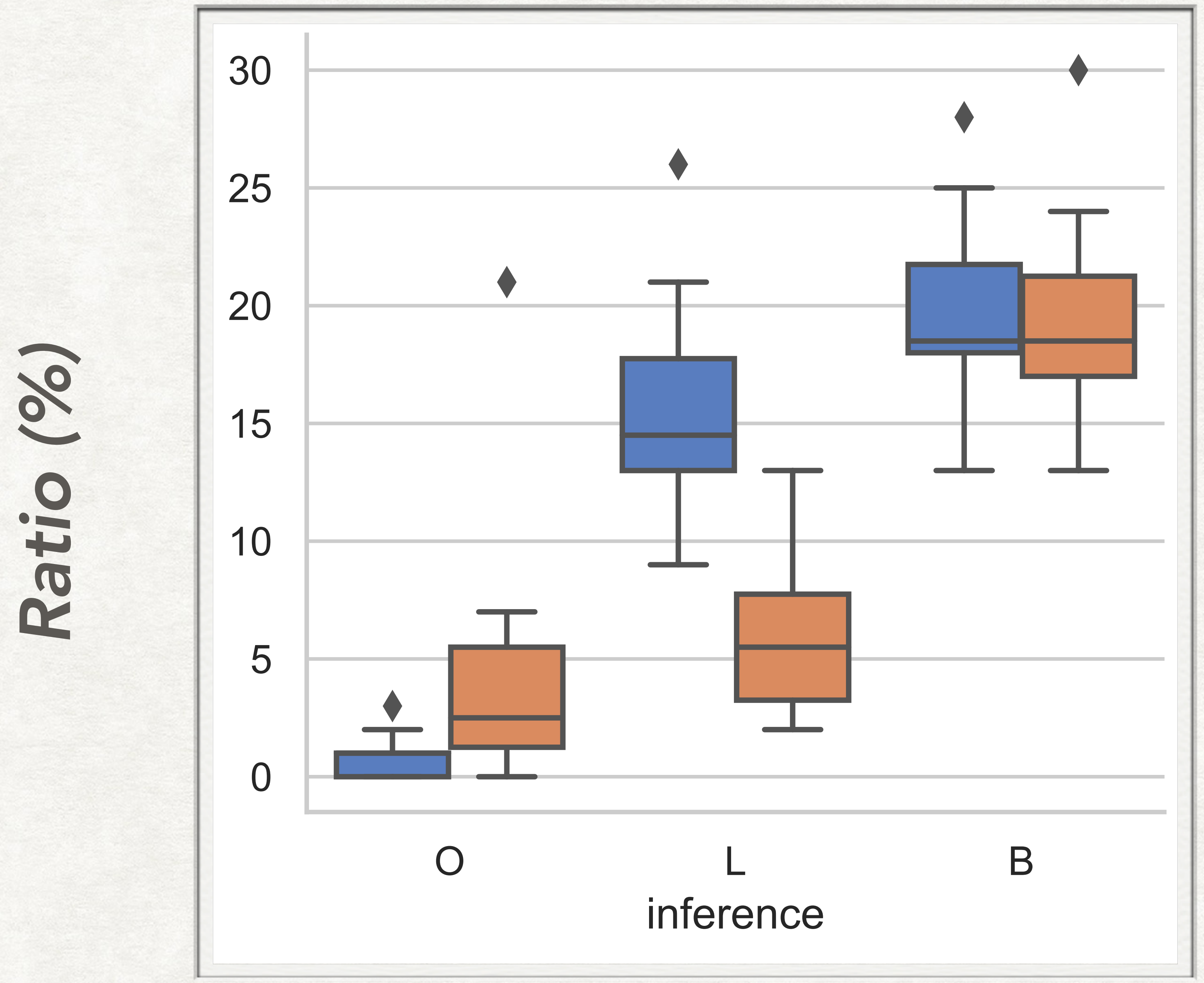
D.

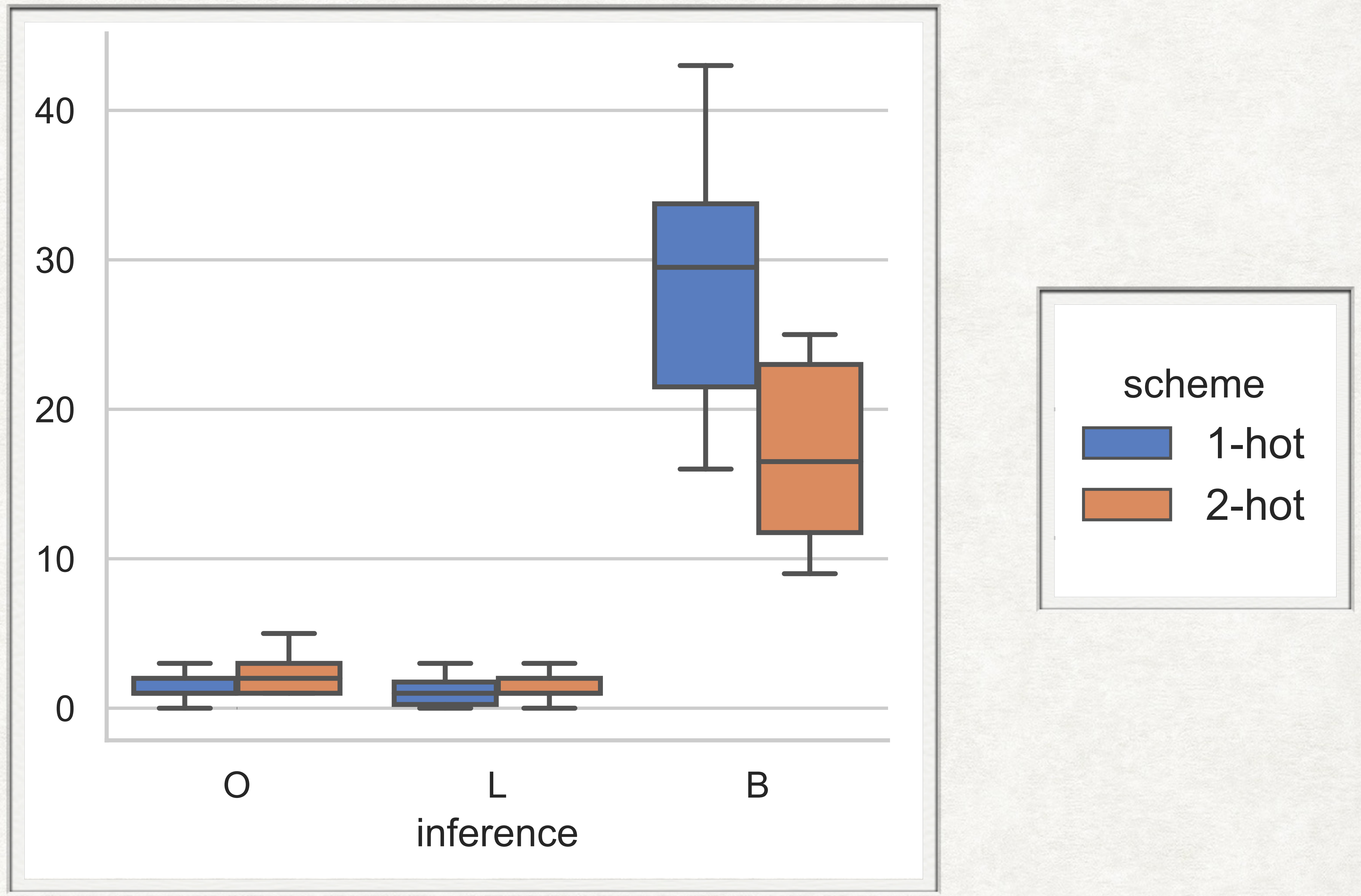
Miss

The number of statements
MOAD fails to delete

Excess

The number of statements
MOAD excessively deletes





O: Once-success *L : Logistic* *B : Bayesian*

E. RESULT: MOAD VS. STATIC SLICER

MISS

Keeping Declaration

MOAD often could not delete '*variable declaration statements.*'

Compilable Slice

MOAD keeps the statements that are needed to be compilable.

EXCESS

Missing Initialization

Dummy values sometimes preserve the behavior.

Missing Return

A value in `rax` register preserves the behavior.

Limit of Static Analysis

The motivation of observation-based analysis.

E. RESULT: MOAD VS. STATIC SLICER

- Limit of Static Analysis

- **Backward**

```
15    while (p(j))
16    {
17        if (q(k))
18        {
19            k = f1(k);
20        }
21        else
22        {
23            k = f2(k);
24            j = f3(j);
25        }
26    }
27    printf("%d\n", j);
28 }
```

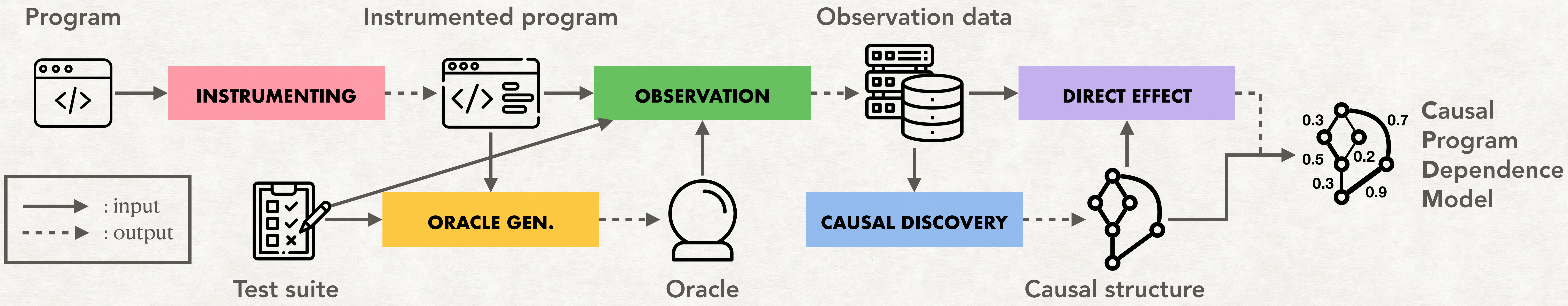
k:23 does not affect j:27

- **Forward**

```
143 ...
144     case 24:
145     case 25:
146     case 32:
147       token_ptr->token_id = special(next_st);
148       token_ptr->token_string[0] = '\0';
149       return (token_ptr);
150     case 27:
151     case 29:
152       token_ptr->token_id = constant(next_st,
           token_str, token_ind);
153       get_actual_token(token_str, token_ind);
154       strcpy(token_ptr->token_string, token_str);
155       return (token_ptr);
156     case 30:
157 ...
```

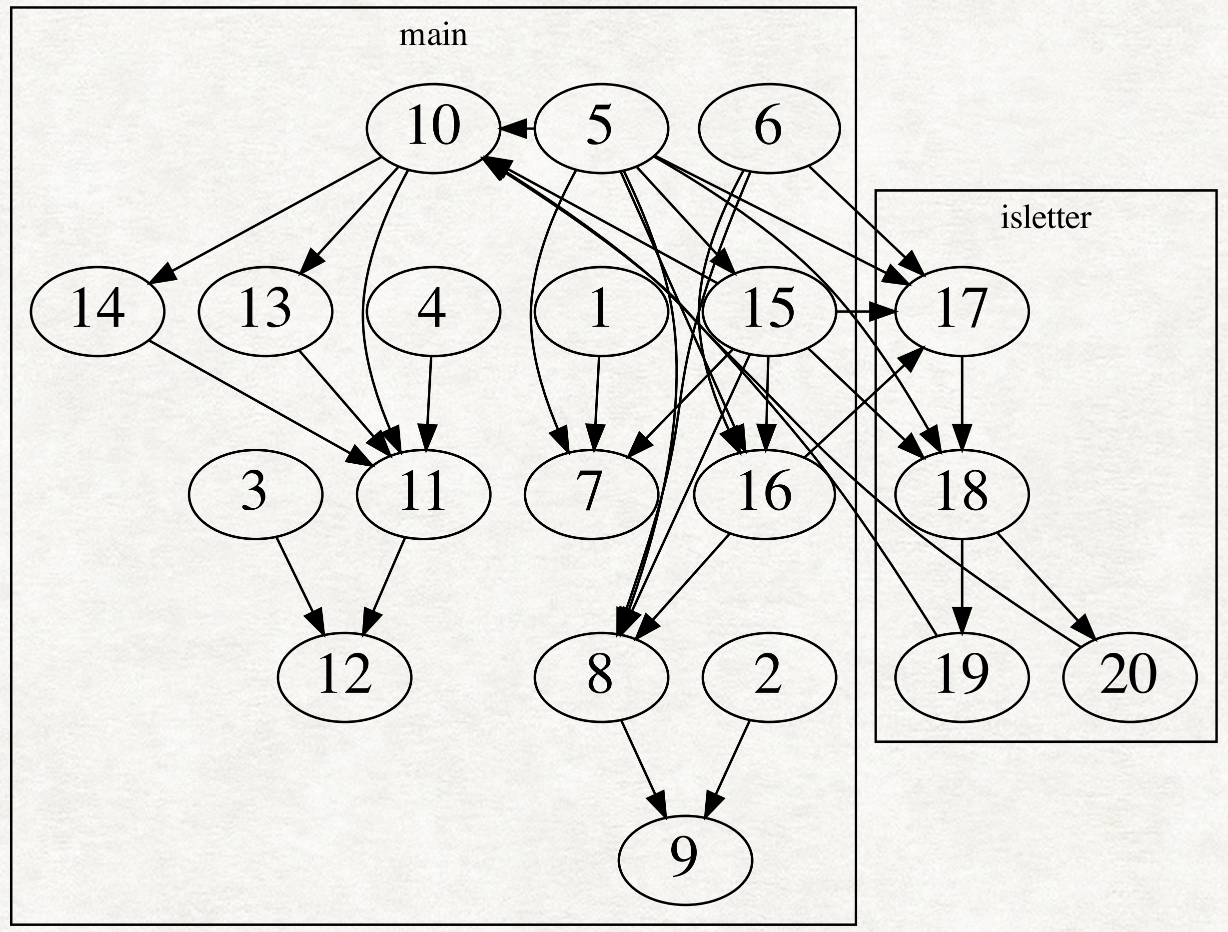
token_id:147 relatively have less effect to *non special token related* codes

F. CAUSAL PROGRAM DEPENDENCE ANALYSIS (CPDA)

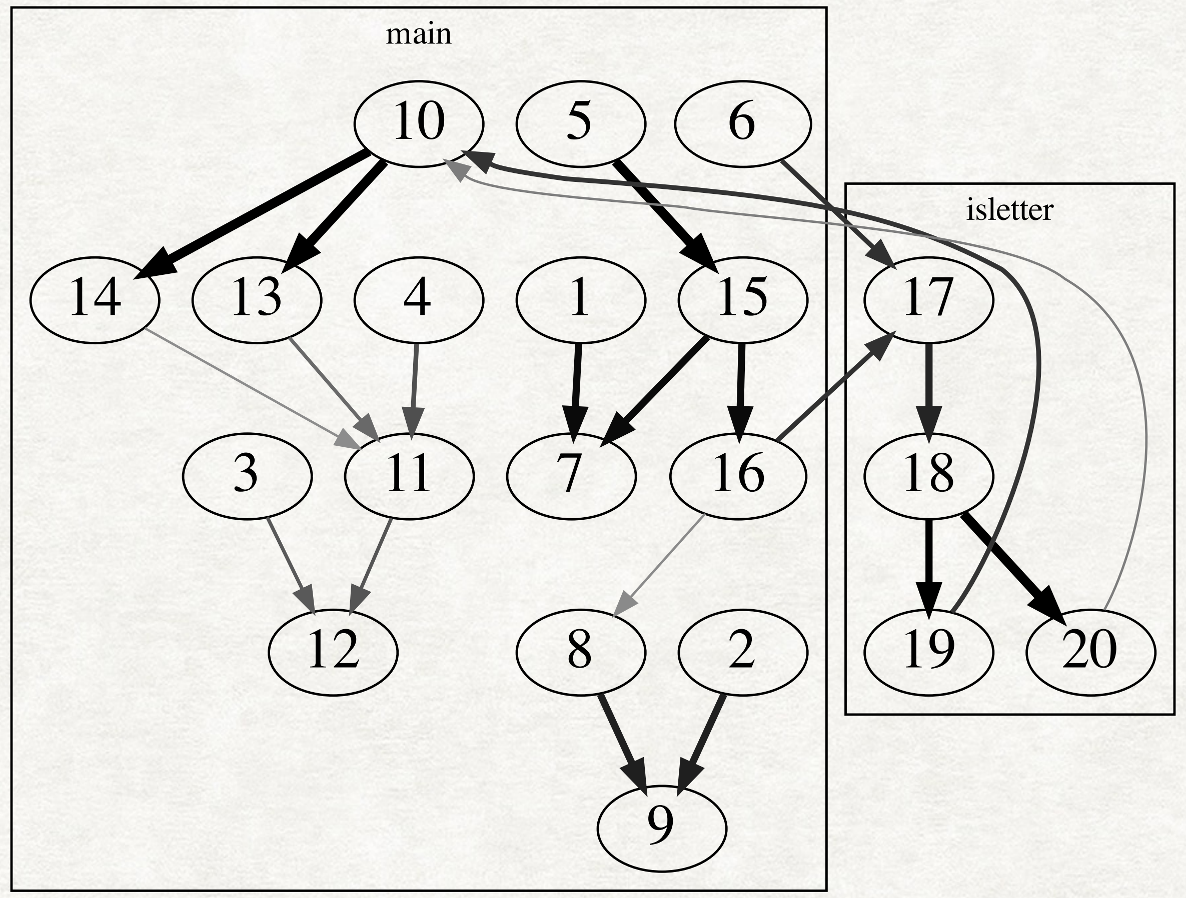


G. RESULT: CPDM VS PDG

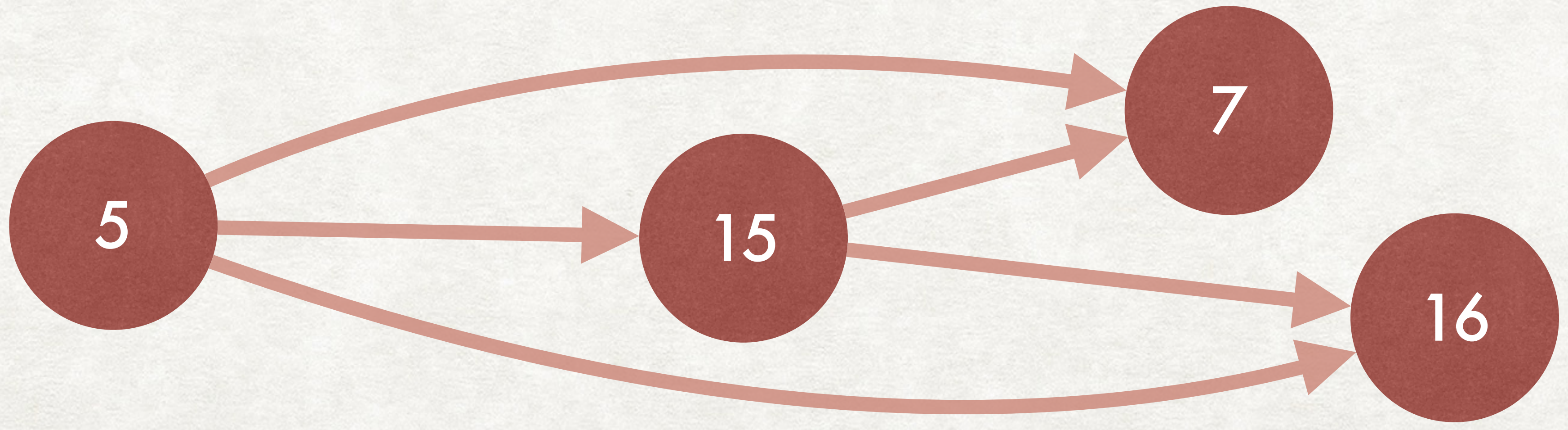
```
def main() {
    <1>characters = 0
    <2>lines = 0
    <3>words = 0
    <4>inword = 0
    <5>_pred1 = getChar(<6>c)
    while (_pred1) {
        <7>characters = characters + 1
        <8>_pred2 = c == '\n'
        if (_pred2)
            <9>lines = lines + 1
        <10>_pred3 = isLetter(c)
        if (_pred3) {
            <11>_pred4 = inword == 0
            if (_pred4) {
                <12>words = words + 1
            }
            <13>inword = 1
        }
        else
            <14>inword = 0
        <15>_pred1 = getChar(<16>c)
    }
}
def isLetter(<17>c) {
    <18>_pred5 = ((c >= 'A' && c <= 'Z')
        || (c >= 'a' && c <= 'z'))
    if (_pred5)
        <19>_ret = True
    else
        <20>_ret = False
    return _ret
}
```



PDG (34 edges)



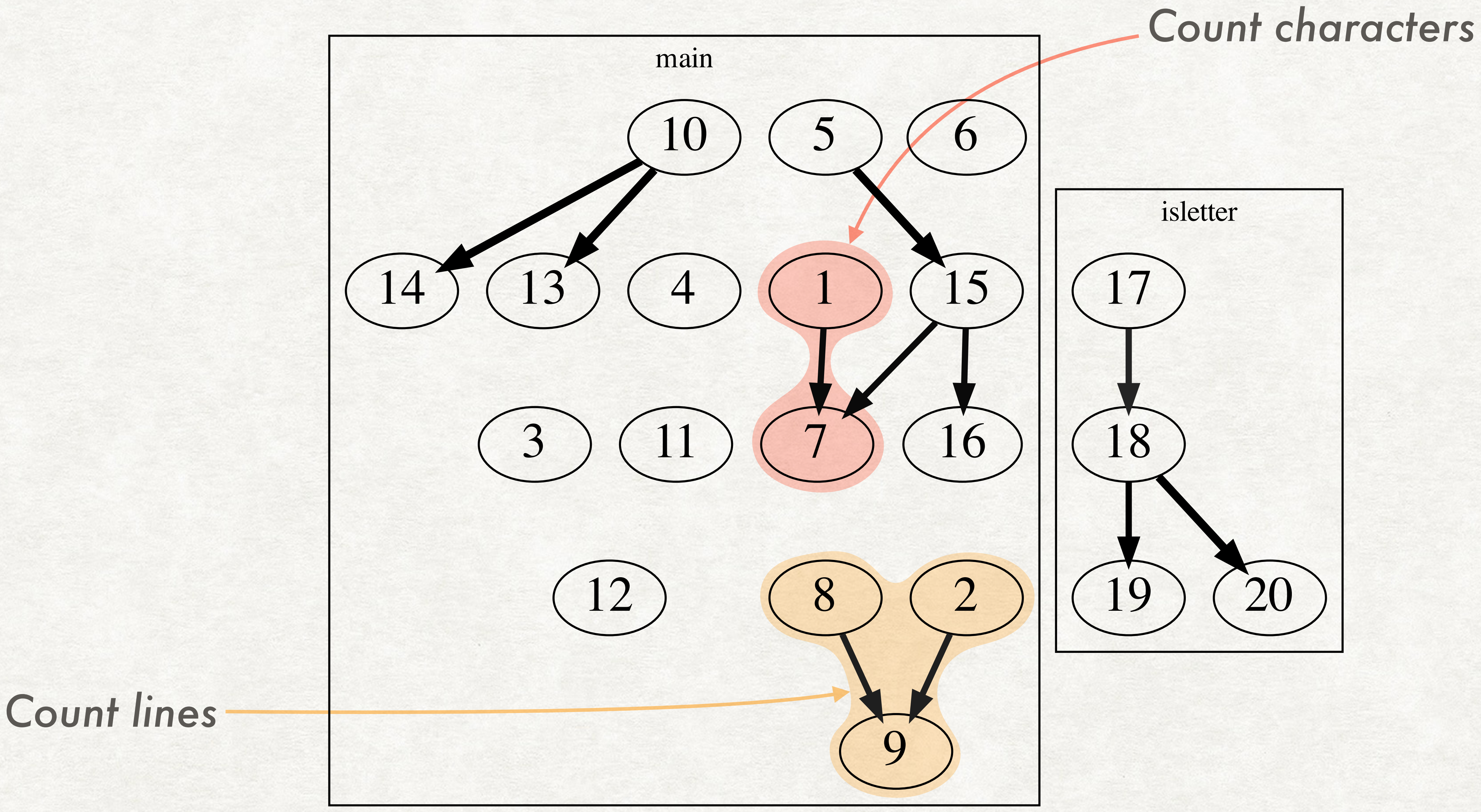
CPDM (21 edges)



G. RESULT: CPDM

```
def main() {
    <1>characters = 0
    <2>lines = 0
    <3>words = 0
    <4>inword = 0
    <5>_pred1 = getChar(<6>c)
    while (_pred1) {
        <7>characters = characters + 1
        <8>_pred2 = c == '\n'
        if (_pred2)
            <9>lines = lines + 1
        <10>_pred3 = isLetter(c)
        if (_pred3) {
            <11>_pred4 = inword == 0
            if (_pred4) {
                <12>words = words + 1
            }
            <13>inword = 1
        }
        else
            <14>inword = 0
        <15>_pred1 = getChar(<16>c)
    }
}
def isLetter(<17>c) {
    <18>_pred5 = ((c >= 'A' && c <= 'Z')
        || (c >= 'a' && c <= 'z'))
    if (_pred5)
        <19>_ret = True
    else
        <20>_ret = False
    return _ret
}
```

Dependence always happens

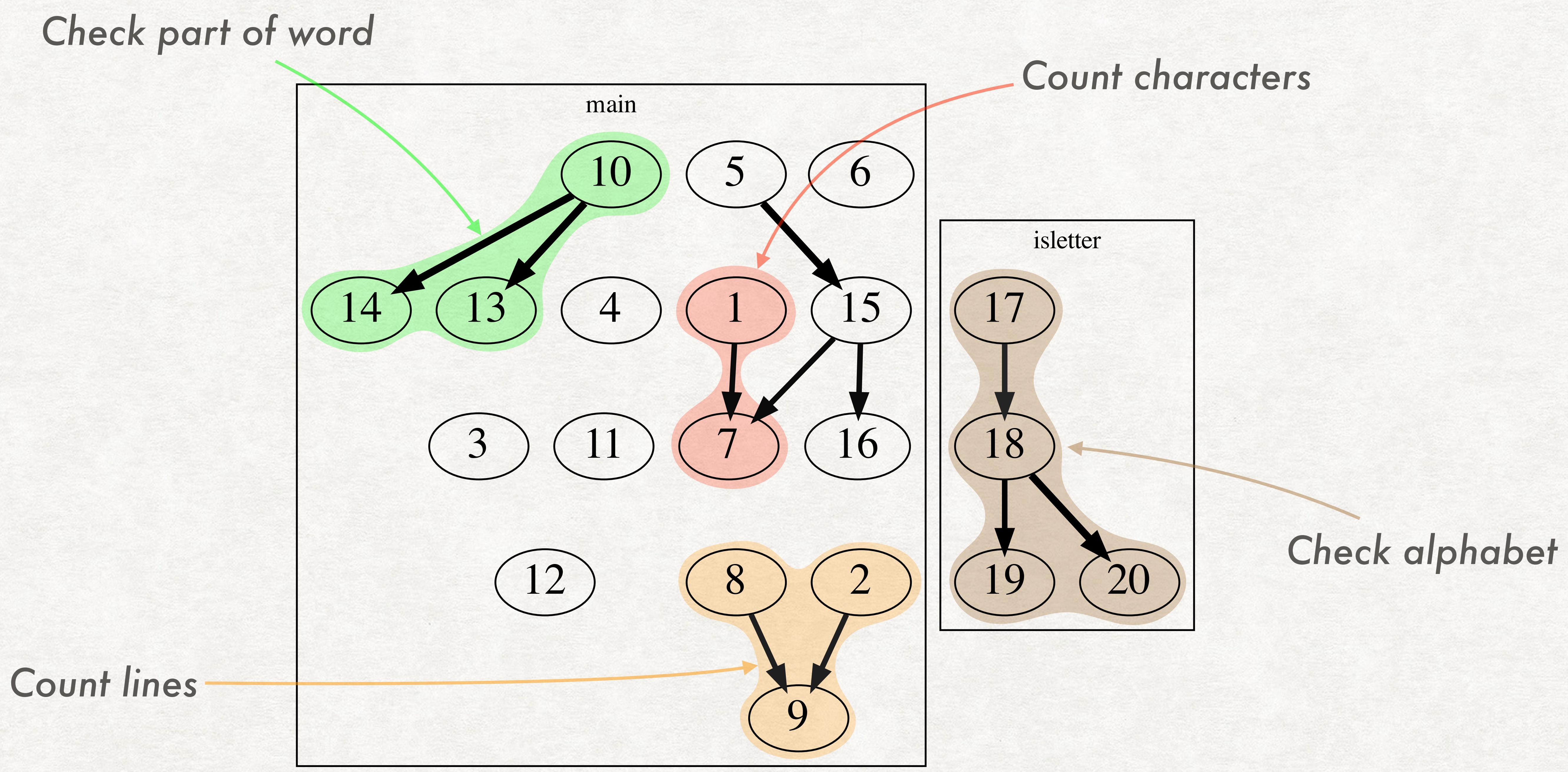


CPDM ($CD \geq 0.8$)

G. RESULT: CPDM

```
def main() {
    <1>characters = 0
    <2>lines = 0
    <3>words = 0
    <4>inword = 0
    <5>_pred1 = getChar(<6>c)
    while (_pred1) {
        <7>characters = characters + 1
        <8>_pred2 = c == '\n'
        if (_pred2)
            <9>lines = lines + 1
        <10>_pred3 = isLetter(c)
        if (_pred3) {
            <11>_pred4 = inword == 0
            if (_pred4) {
                <12>words = words + 1
            }
            <13>inword = 1
        }
        else
            <14>inword = 0
        <15>_pred1 = getChar(<16>c)
    }
}
def isLetter(<17>c) {
    <18>_pred5 = ((c >= 'A' && c <= 'Z')
        || (c >= 'a' && c <= 'z'))
    if (_pred5)
        <19>_ret = True
    else
        <20>_ret = False
    return _ret
}
```

Dependence always happens



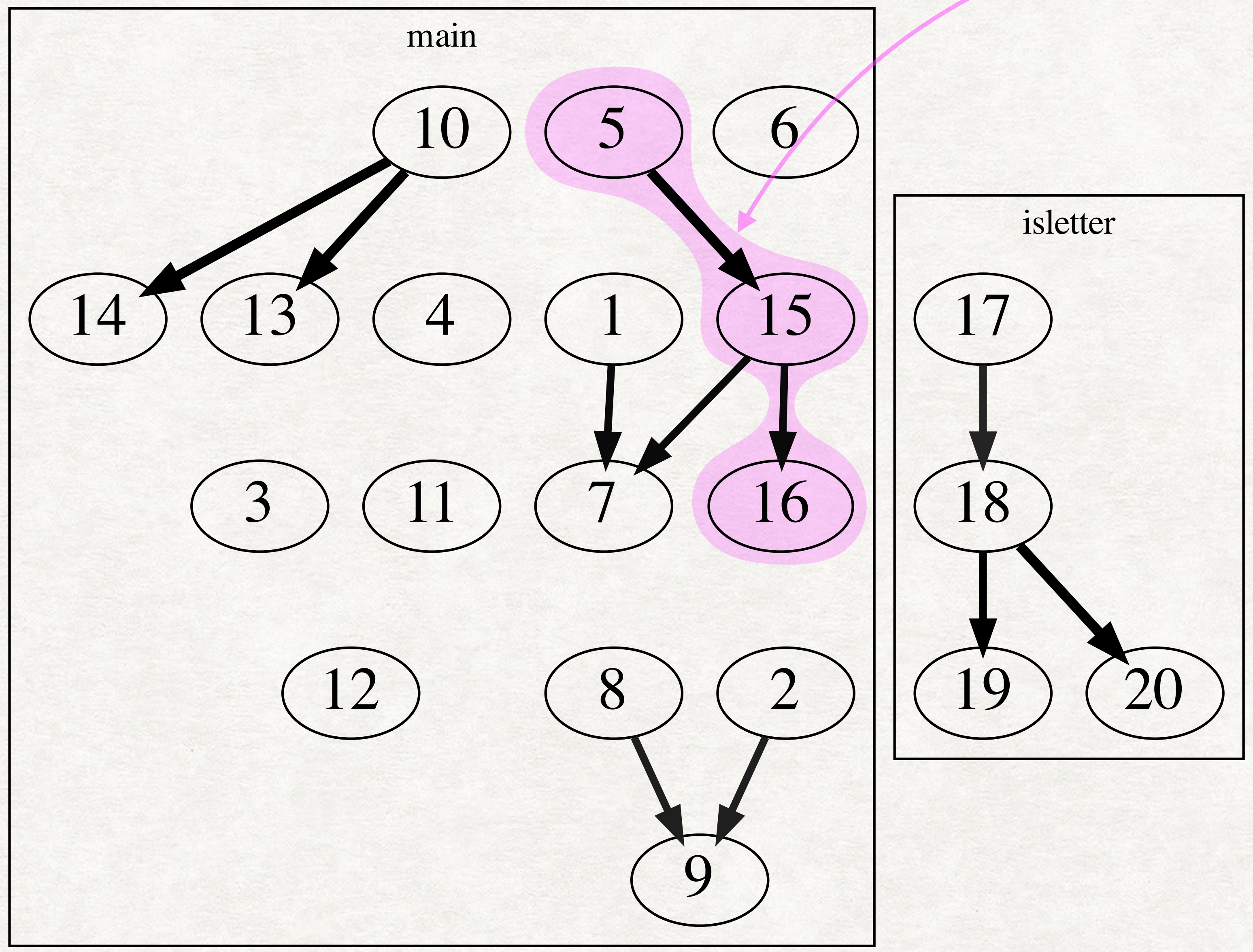
CPDM $(CD \geq 0.8)$

G. RESULT: CPDM

```
def main() {
    <1>characters = 0
    <2>lines = 0
    <3>words = 0
    <4>inword = 0
    <5>_pred1 = getChar(<6>c)
    while (_pred1) {
        <7>characters = characters + 1
        <8>_pred2 = c == '\n'
        if (_pred2)
            <9>lines = lines + 1
        <10>_pred3 = isLetter(c)
        if (_pred3) {
            <11>_pred4 = inword == 0
            if (_pred4) {
                <12>words = words + 1
            }
            <13>inword = 1
        }
        else
            <14>inword = 0
        <15>_pred1 = getChar(<16>c)
    }
}
def isLetter(<17>c) {
    <18>_pred5 = ((c >= 'A' && c <= 'Z')
        || (c >= 'a' && c <= 'z'))
    if (_pred5)
        <19>_ret = True
    else
        <20>_ret = False
    return _ret
}
```

Dependence always happens

Main loop reading input char.



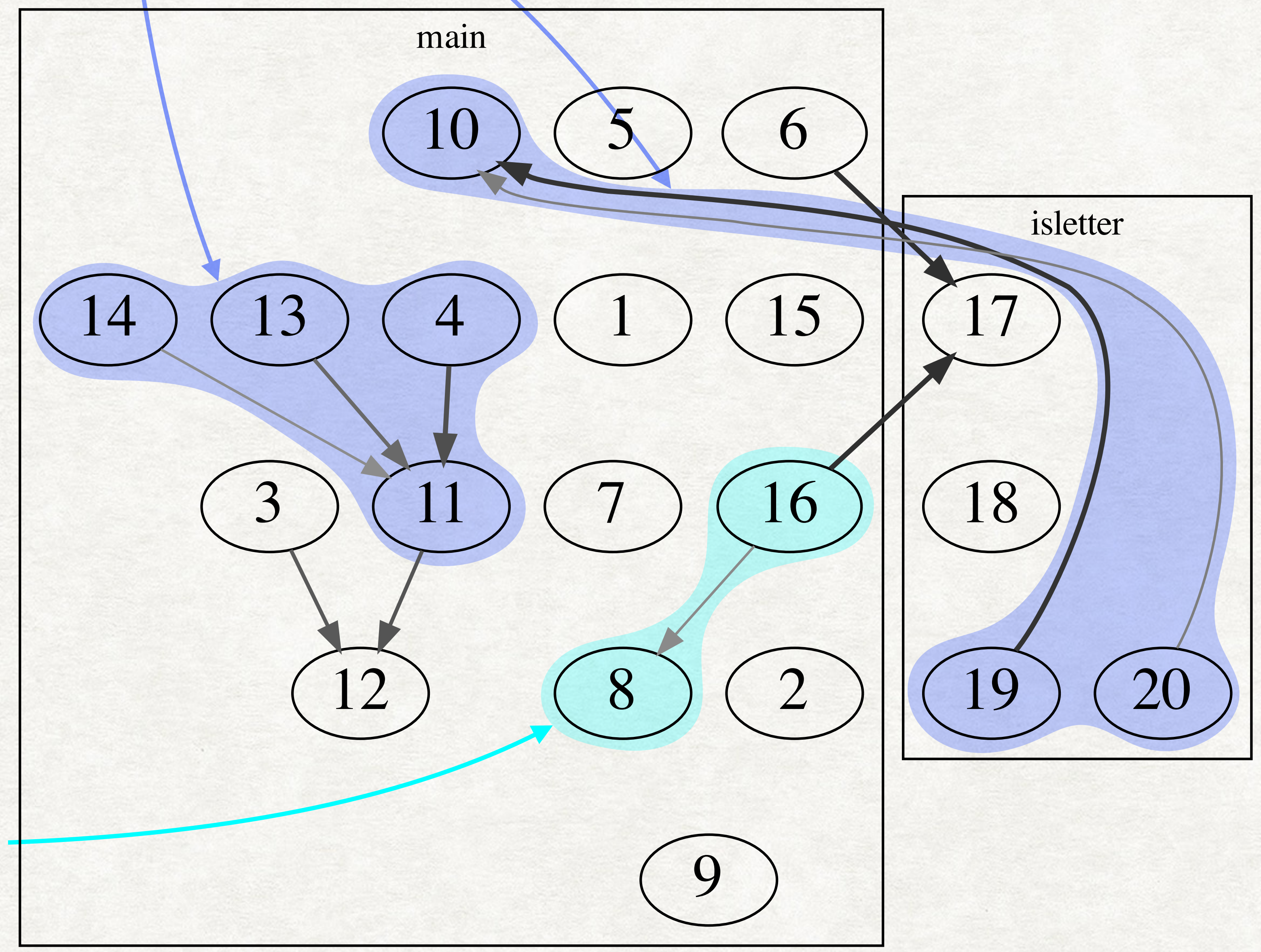
CPDM $(CD \geq 0.8)$

G. RESULT: CPDM

```
def main() {
    <1>characters = 0
    <2>lines = 0
    <3>words = 0
    <4>inword = 0
    <5>_pred1 = getChar(<6>c)
    while (_pred1) {
        <7>characters = characters + 1
        <8>_pred2 = c == '\n'
        if (_pred2)
            <9>lines = lines + 1
        <10>_pred3 = isLetter(c)
        if (_pred3) {
            <11>_pred4 = inword == 0
            if (_pred4) {
                <12>words = words + 1
            }
            <13>inword = 1
        }
        else
            <14>inword = 0
        <15>_pred1 = getChar(<16>c)
    }
}
def isLetter(<17>c) {
    <18>_pred5 = ((c >= 'A' && c <= 'Z')
        || (c >= 'a' && c <= 'z'))
    if (_pred5)
        <19>_ret = True
    else
        <20>_ret = False
    return _ret
}
```

Dependence occasionally happens

Some tests have a one word, while some have multiple words



Some tests have a single line, while some have multiple lines

CPDM $(0.2 \leq CD < 0.8)$

H. ADVANTAGE OF QUANTIFIABLE DEPENDENCE

CDFL vs SBFL

Code	$\text{Rank}_{\text{SBFL}}$	$\text{Rank}_{\text{CDFL}}$
`a = 3`	1	2
`b = 4`	1	3
`c = a % 3 + 1`	1	1
`return c`	-	-

CDFL considers the value in the variable breaking ties in the same basic block.

CDFL vs Dicing, Dynamic slice

Code	Coverage “a”	Coverage “a␣”	Coverage “␣a”	DS	Dice	$\text{Susp}_{\text{CDFL}}$
`s = input()`	1	1	1	1	0	1.0 - 1.0 = 0.0
`pred = isEndSpace(s)`	0	1	1	1	0	1.0 - 0.5 = 0.5
`if (pred) p =` ~~`p.rstrip()`~~ `p.strip()`	0	1	1	1	0	1.0 - 0.5 = 0.5
`return p`	-	-	-	-	-	-
Test Results	P	P	F			

Faulty element may affect in both passing / failing execution.

I. EFFICIENT CPDA

$Traj(P, \text{Var i}) \neq Traj(P', \text{Var i})$ ➡ $Traj(P(c_1), \text{Var i}) \neq Traj(P(c_2), \text{Var i})$

Mutated program → Overhead

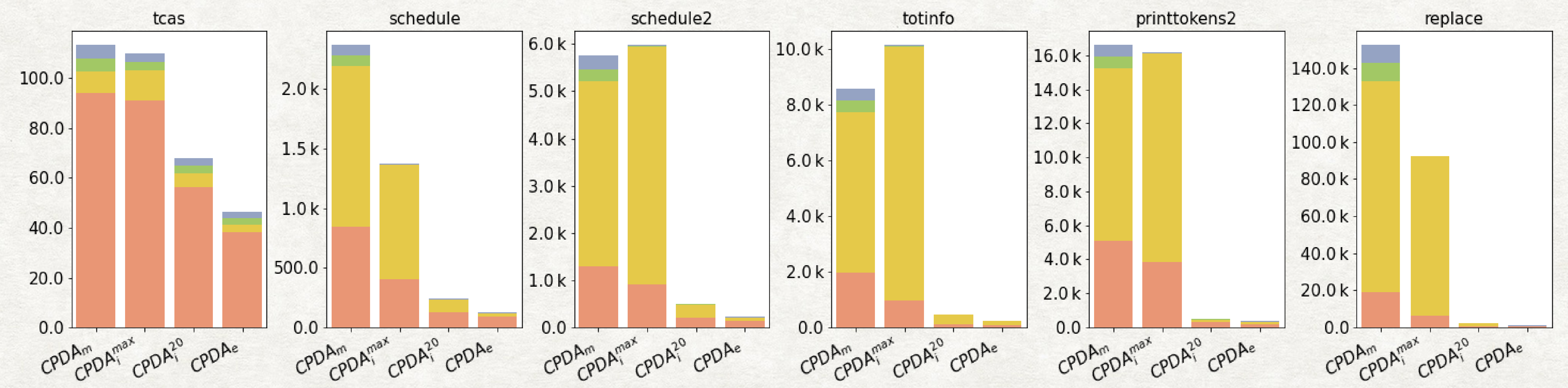
Compare between different execution

- Instead comparing with mutation, compare between different test cases.
  - Considering we only mutate on the input value

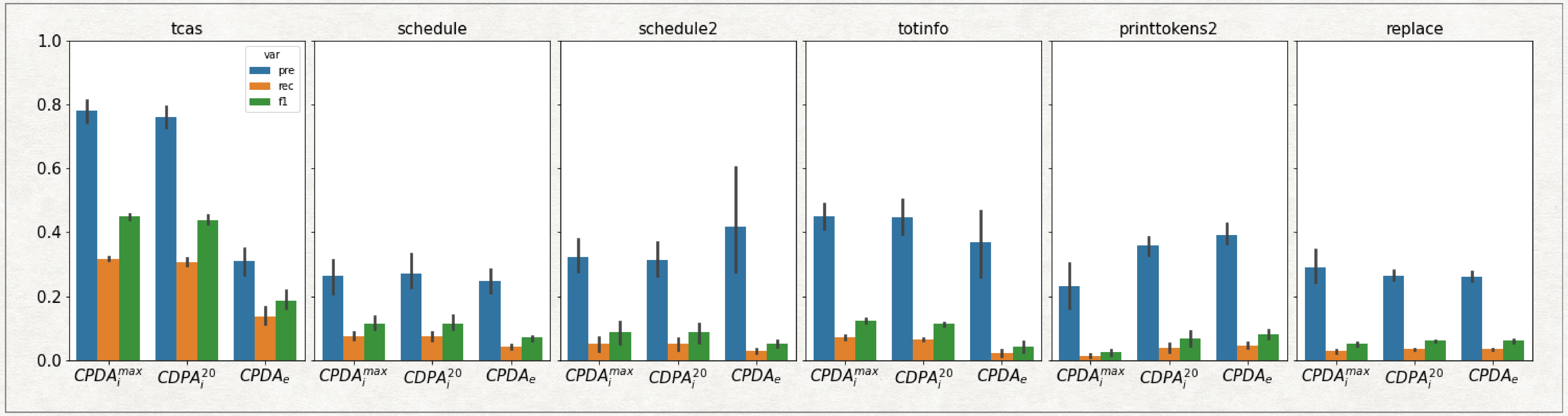
I. EFFICIENT CPDA

Table 4.6: Average number of observations used for analysis. Percentages in the parenthesis show the average ratio of observations used relative to CPDA_m.

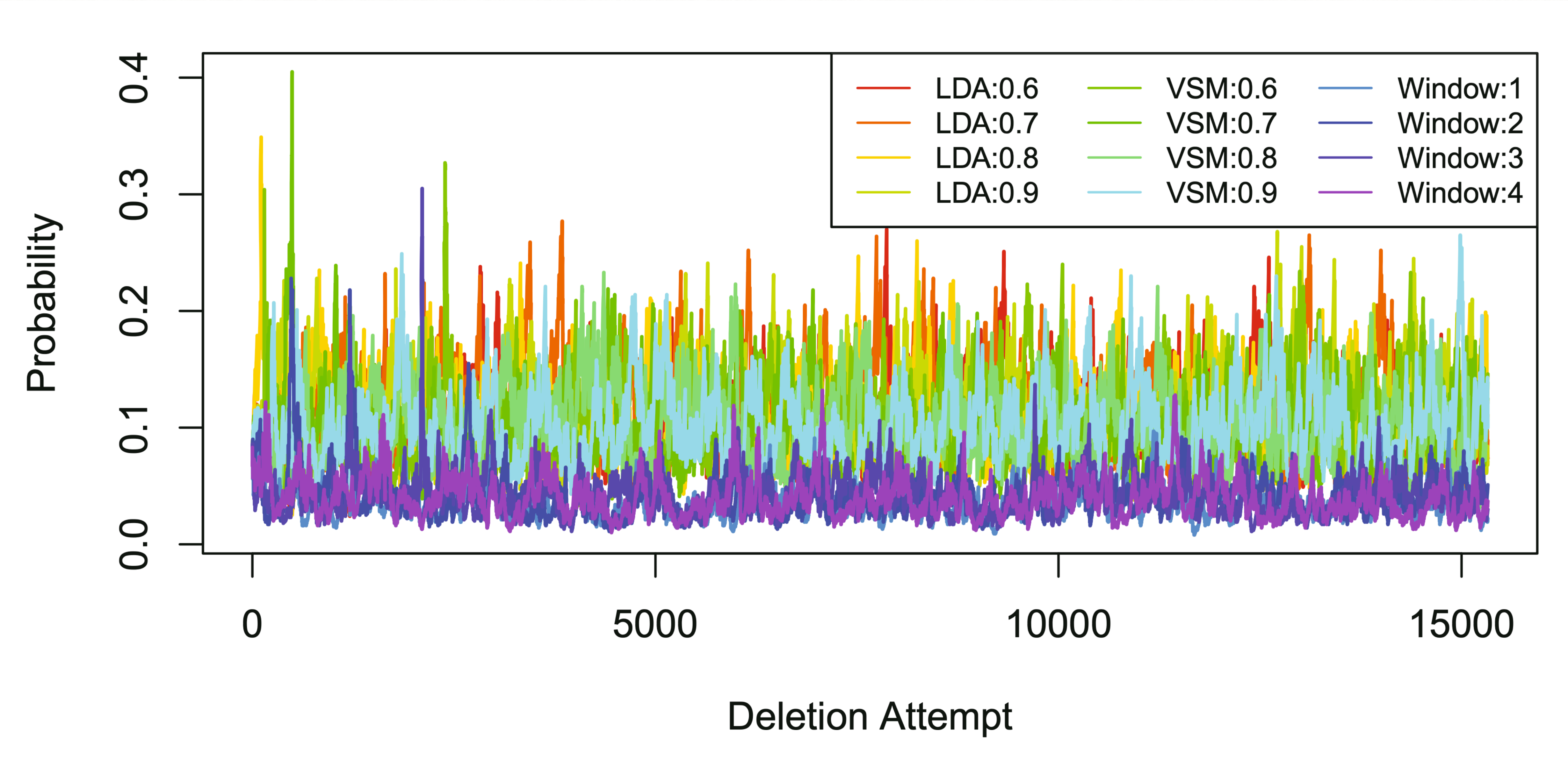
Program	CPDA_m	CPDA_i^{max}	CPDA_i^{20}	CPDA_e
tcas	2,206.4 (100%)	2,088.8 (94.6%)	744.8 (33.7%)	13.0 (0.58%)
sched	9,363.4 (100%)	9,382.8 (100.2%)	814.8 (8.7%)	32.0 (0.33%)
sched2	15,347.8 (100%)	15,013.4 (97.7%)	745.4 (4.8%)	33.0 (0.19%)
totinfo	17,120.2 (100%)	16,292.6 (95.1%)	219.0 (1.2%)	61.0 (0.34%)
prttok2	14,855.6 (100%)	14,981.2 (100.8%)	419.0 (2.8%)	16.2 (0.10%)
replace	72,225.4 (100%)	70,614.2 (97.7%)	1,548.4 (2.1%)	167.6 (0.23%)
Avg. ratio	100%	97.7%	8.9%	0.30%



I. EFFICIENT CPDA



J. ADAPTIVE MOBS PROBABILITY CHANGE



K. CPDA COMPREHENSION RESULT: TRIANGLE

```
Not? = S1 + S2 <= S3 || S1 + S3 <= S2 || S2 + S3 <= S1;
if (Not?)
    type = NOTATRINGLE;
else {
    Equ? = S1 == S2 && S2 == S3;
    if (Equ?)
        type = EQUILATERAL;
    else {
        Iso? = S1 == S2 || S1 == S3 || S2 == S3;
        if (Iso?)
            type = ISOSCELES;
        else
            type = SCALENE;
    }
}
ret = type;
```

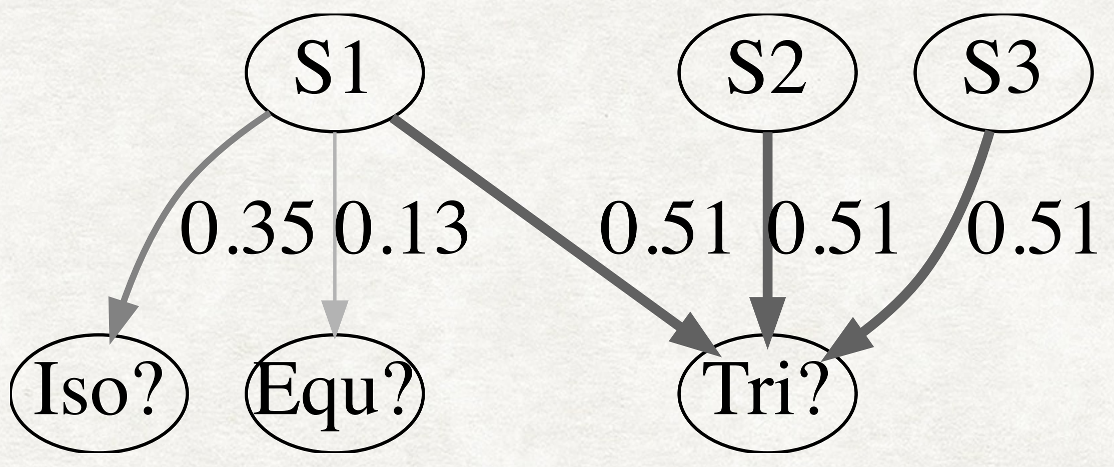


Figure 4.8: Partial graph from CPDM of tri

Difference in the input distribution

Test suite	⟨Tri?⟩	⟨Equ?⟩	⟨Iso?⟩
Total	0.51	0.13	0.35
Valid	**0.97**	0.13	0.35

(a) $\{\langle S1\rangle\} \rightarrow \{\langle \text{predicate node}\rangle\}$

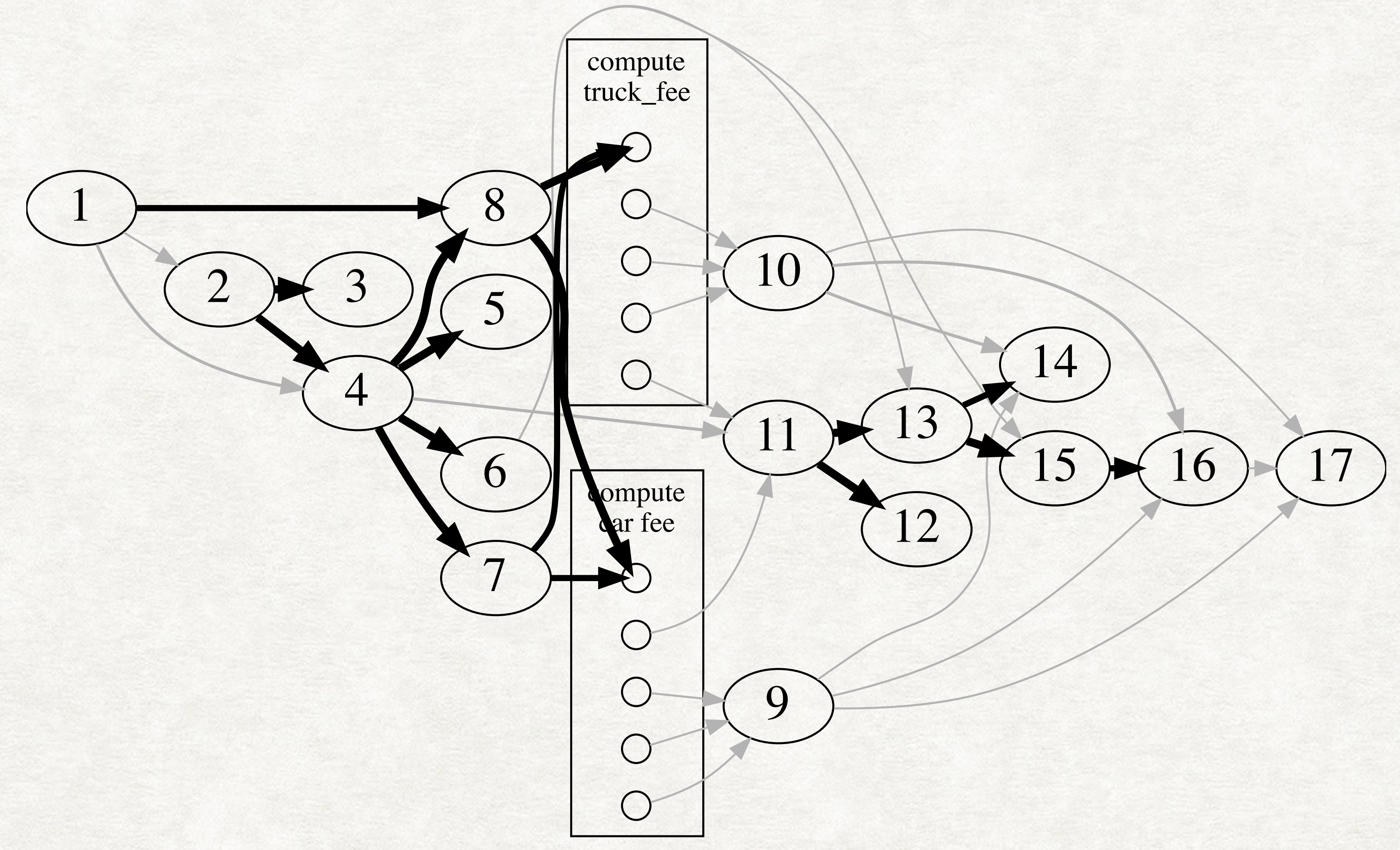
Ordered	⟨S1⟩	⟨S2⟩	⟨S3⟩
⟨Equ?⟩	0.29	**0.18**	0.24
⟨Iso?⟩	0.21	**0.46**	0.33

(b) $\{\langle S^*\rangle\} \rightarrow \{\langle \text{Equ?}\rangle,\langle \text{Iso?}\rangle\}$

Figure 4.10: (a) Select *DD* values involving ⟨S1⟩ (the values for ⟨S2⟩ and ⟨S3⟩ are essentially the same) from tri using all tests (***Total***) and those satisfying the triangle inequality ***Valid***. (b) Select values obtained using the ***Ordered*** test suite.

L. CPDA COMPREHENSION RESULT: BILL&TED

```
def main(args) {
   <1>car_type = args[0]
   <2>_pred1 = car_type == "SENIOR_CITIZEN"
   if (_pred1) <3>fee = 0.0
   else {
      <4>_pred2 = !(car_type == "CAR" || car_type == "TRUCK")
      if (_pred2) <5>fee = -2.0 // INVALID
      else {
         <6>day, <7>duration = args[1], args[2]
         <8>_pred3 = car_type == "CAR"
         if (_pred3) <9>cost = compute_car_fee(duration)
         else <10>cost = compute_truck_fee(duration)
         <11>_pred4 = cost == -1.0 // EXCEED MAX DURATION
         if (_pred4) <12>fee = -1.0
         else {
            <13>_pred5 = day == "THURSDAY"
            if (_pred5) <14>cost = cost * THURSDAY_DISCOUNT
            else {
               <15> _pred6 = day == "SATURDAY"
               if (_pred6) <16>cost = cost * SATURDAY_SURCHARGE
            }
            <17>fee = cost
         ...}}}} // END of main
```



(c) CPDM of B&T, (*black*: $DD > 0.7$, *grey*: $DD > 0.2$)

M. ALTERNATIVE CAUSAL DEPENDENCE

Definition 4.8 (Subtraction form of Causal Dependence) *Given a set of observations O and two nodes S_i and S_j, the subtraction form of causal dependence from S_i to S_j, $CD^s_O(S_i, S_j)$, is defined as follows:*

$$CD^s_O(S_i, S_j) = P_O(S_j = 1 \mid do(S_i = 1)) - P_O(S_j = 1 \mid do(S_i = 0)) \,.$$

Definition 4.9 (Multiplication form of Causal Dependence) *Given a set of observations O and two nodes S_i and S_j, the multiplication form of causal dependence from S_i to S_j, $CD^m_O(S_i, S_j)$, is defined as follows:*

$$\begin{aligned} CD^m_O(S_i, S_j) &= P_O(S_j = 1 \mid do(S_i = 1)) \times (1 - P_O(S_j = 1 \mid do(S_i = 0))) \\ &= P_O(S_j = 1 \mid do(S_i = 1)) \times P_O(S_j = 0 \mid do(S_i = 0)) \,. \end{aligned}$$

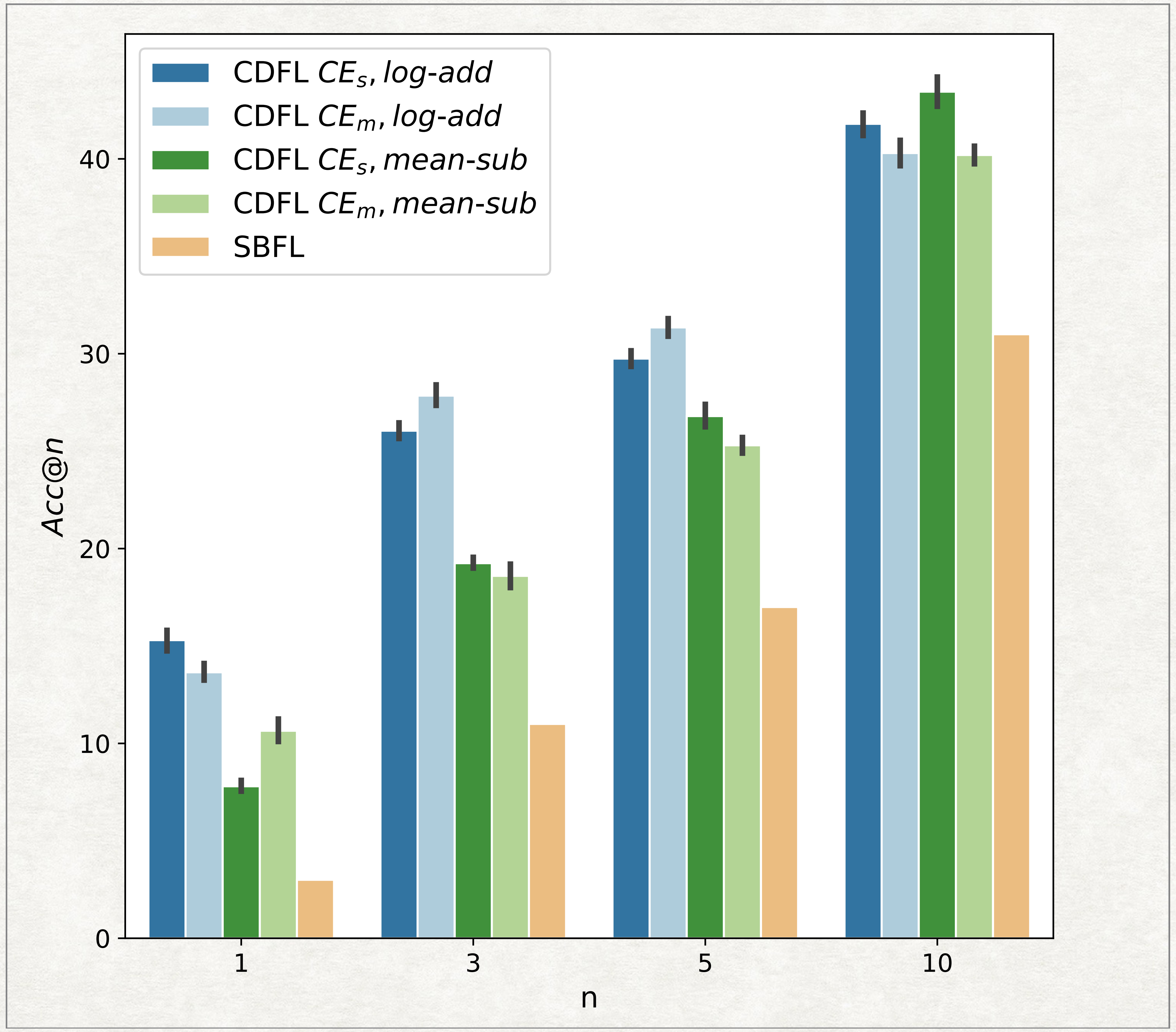
N. TWO SUSPICIOUSNESS FORMULAE

$$susp_{avg}(S_k) = \frac{1}{|\mathcal{I}_{fail}|} \sum_{tc_i \in \mathcal{I}_{fail}} CD_{O_{tc_i}}(S_k, S_{out}) - \frac{1}{|\mathcal{I}_{pass}|} \sum_{tc_j \in \mathcal{I}_{pass}} CD_{O_{tc_j}}(S_k, S_{out}),$$

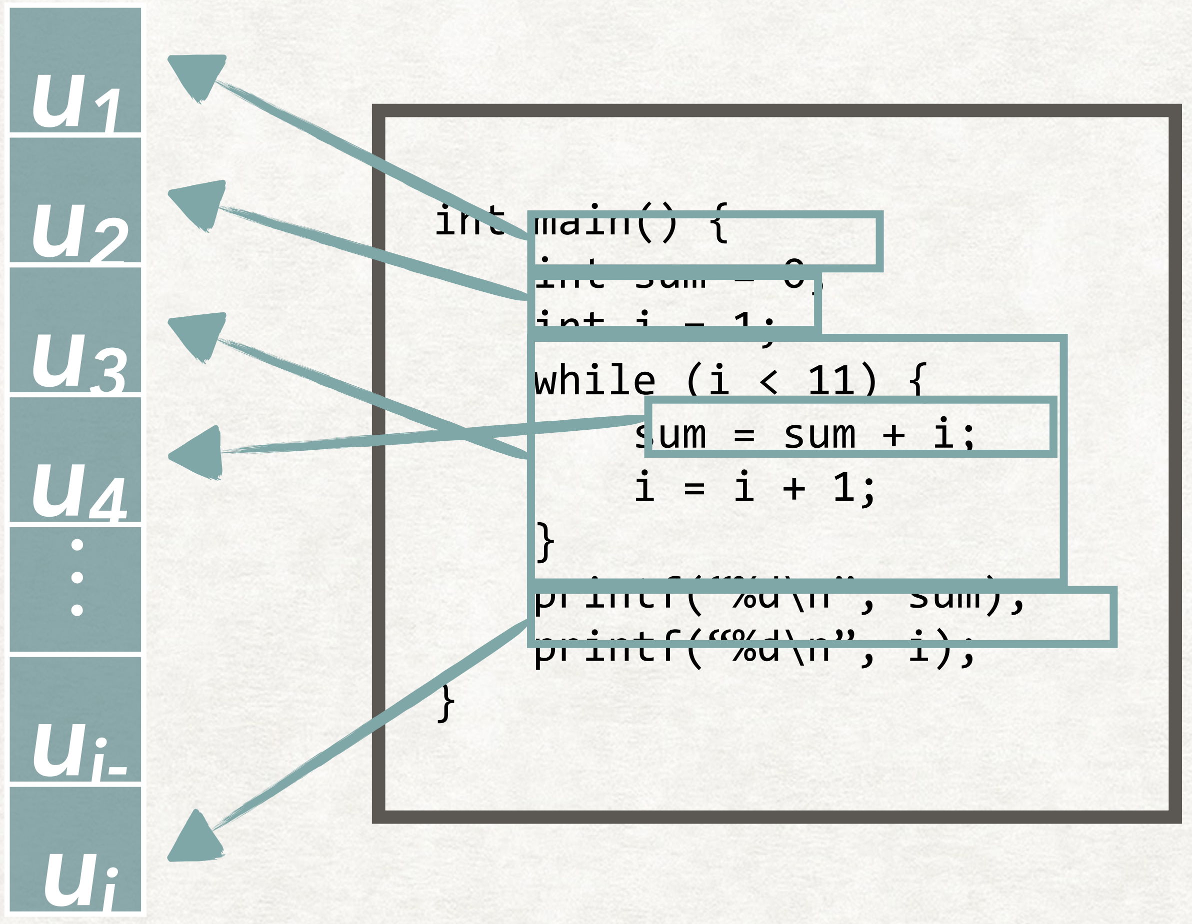
where CD is either CD^s or CD^m, and $\mathcal{I}_{fail}$ and $\mathcal{I}_{pass}$ denote the sets of failing and passing test inputs, respectively. Note that the value of S_{out} represents the change of outcome, and not pass or fail.

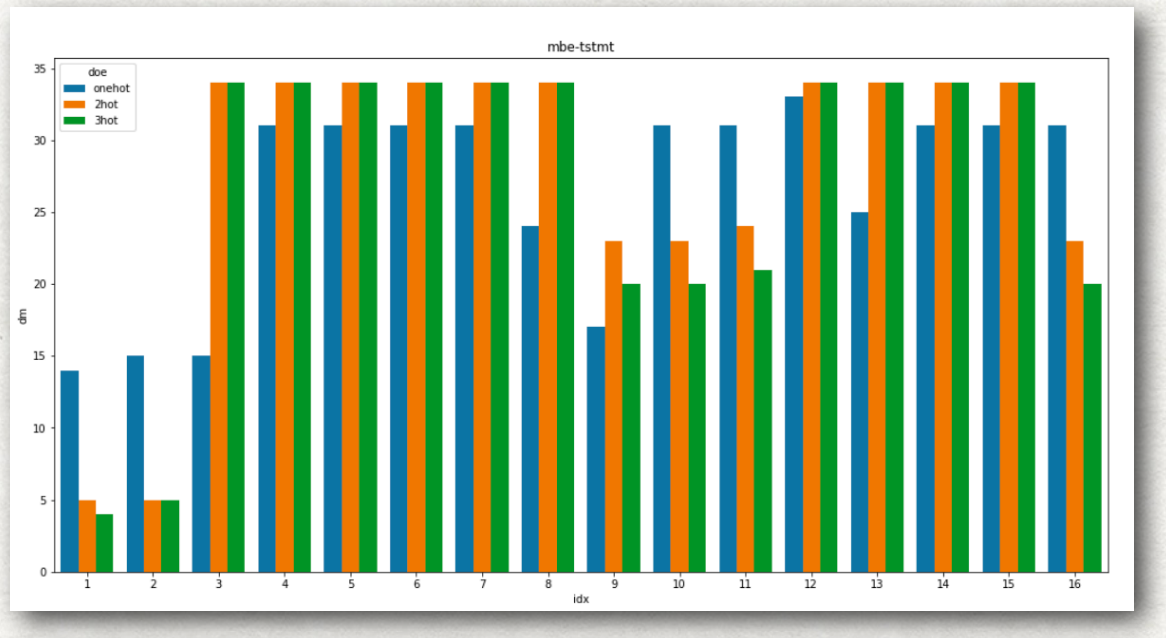
$$\begin{aligned}
susp_{log}(S_k) &= \prod_{tc_i \in I_{fail}} P(\text{fault in } S_k \mid tc_i \text{ fails}) \times \prod_{tc_j \in I_{pass}} P(\text{fault in } S_k \mid tc_j \text{ passes}) \\
&\sim \prod_{tc_i \in I_{fail}} P(tc_i \text{ fails} \mid \text{fault in } S_k) \times \prod_{tc_j \in I_{pass}} P(tc_j \text{ passes} \mid \text{fault in } S_k) \\
&= \prod_{tc_i \in I_{fail}} P(tc_i \text{ fails} \mid \text{fault in } S_k) \times \prod_{tc_j \in I_{pass}} [1 - P(tc_j \text{ fails} \mid \text{fault in } S_k)] \\
&\approx \prod_{tc_i \in I_{fail}} CD_{O_{tc_i}}(S_k, S_{out}) \times \prod_{tc_j \in I_{pass}} \left[1 - CD_{O_{tc_j}}(S_k, S_{out})\right]
\end{aligned}$$

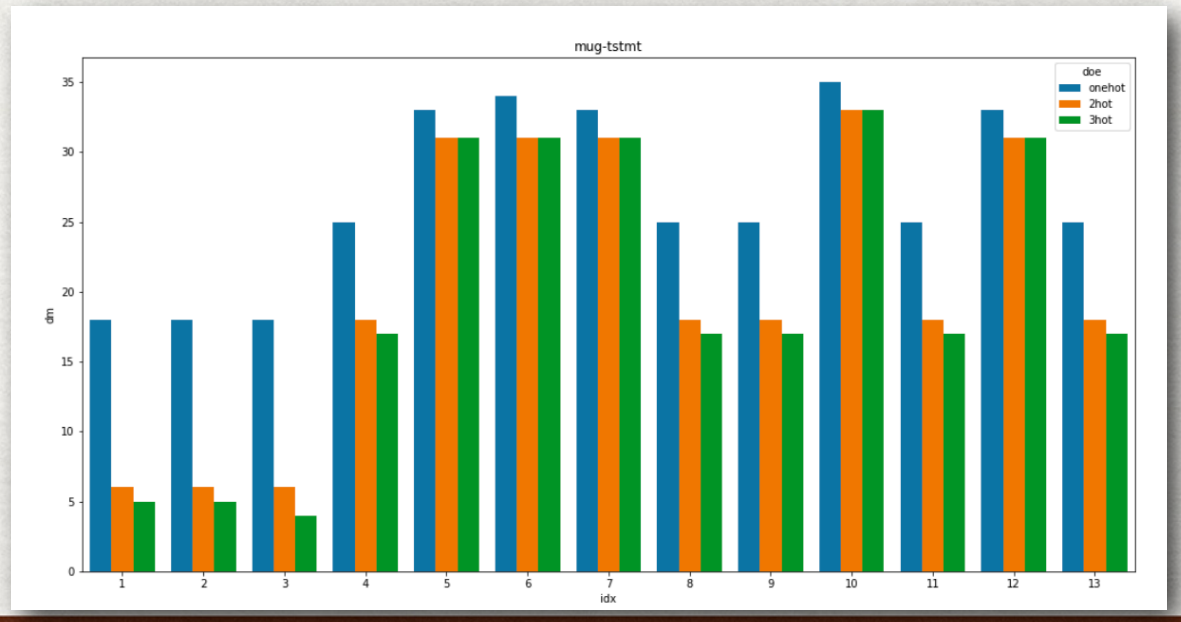
O. CDFL FOR TWO CD AND TWO SUSP



P. MOAD, MORE THAN TWO UNITS DELETION







- We delete block statements, too.
- In general, there can be non-deletable statement with n-hot deletion (requiring (n+1)-hot)
- 2-hot / ORBS = 12%, but 2-hot / 3-hot is much larger
- However, 3-hot does not delete much.
- Random deletion generation scheme.

  - random: each element in the factor is sampled from Bernoulli distribution: $P(X = 1) = p = 1 - Pr(X = 0)$. The initial value of p is $1/|U|$.

  - Observe fixed number
  - 2-hot always produces smaller size
- Exhaustive/random is inefficient. Specific heuristics is needed
  - e.g. language model