

– Journal of Systems and Software, 2021 –

OBSERVATION-BASED APPROXIMATE DEPENDENCY MODELING AND ITS USE FOR PROGRAM SLICING

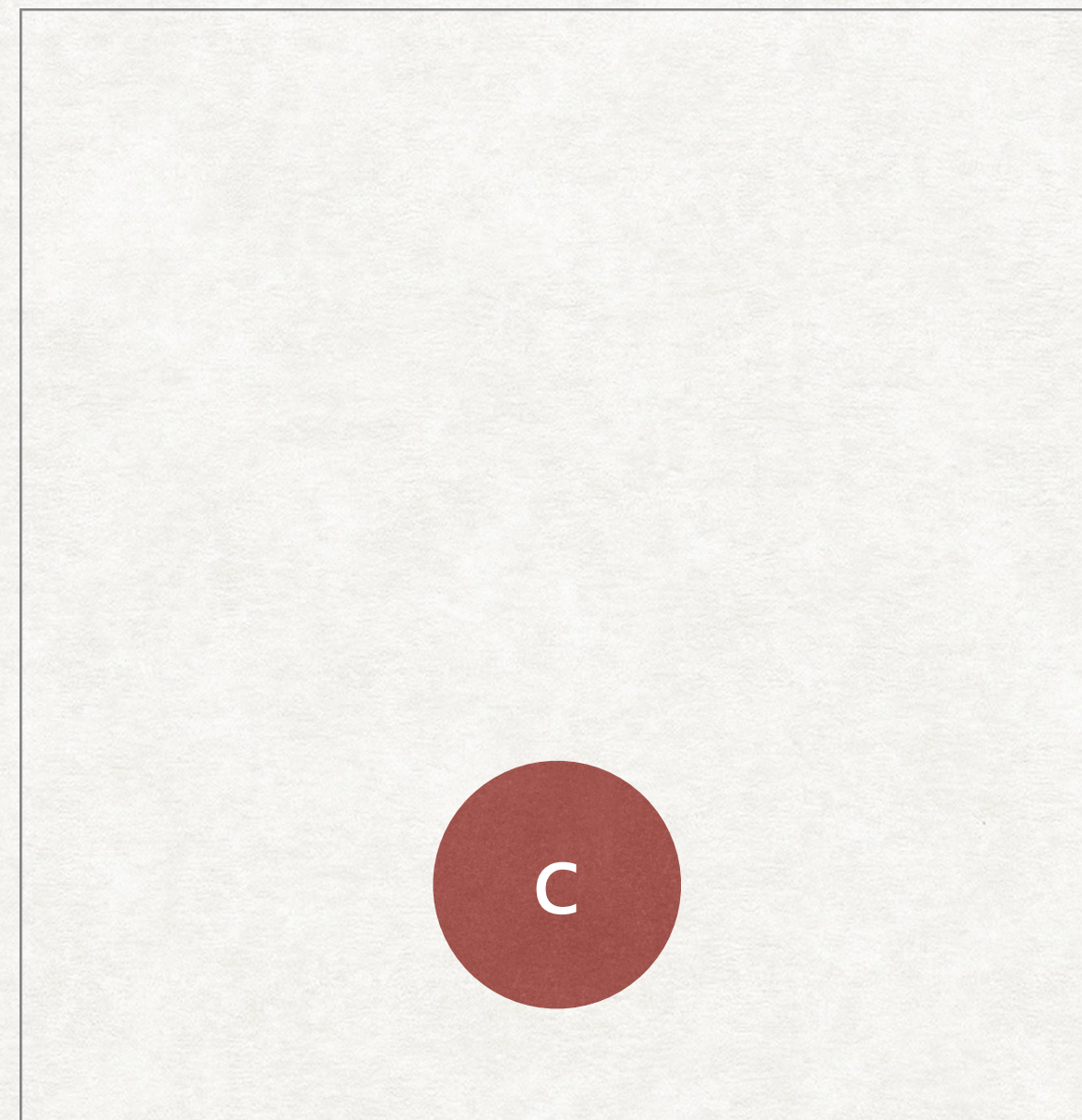
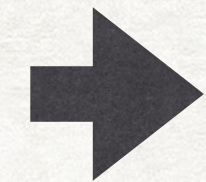
Seongmin Lee¹, David Binkley², Robert Feldt³, Nicolas Gold⁴, Shin Yoo¹

PROGRAM DEPENDENCY ANALYSIS

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

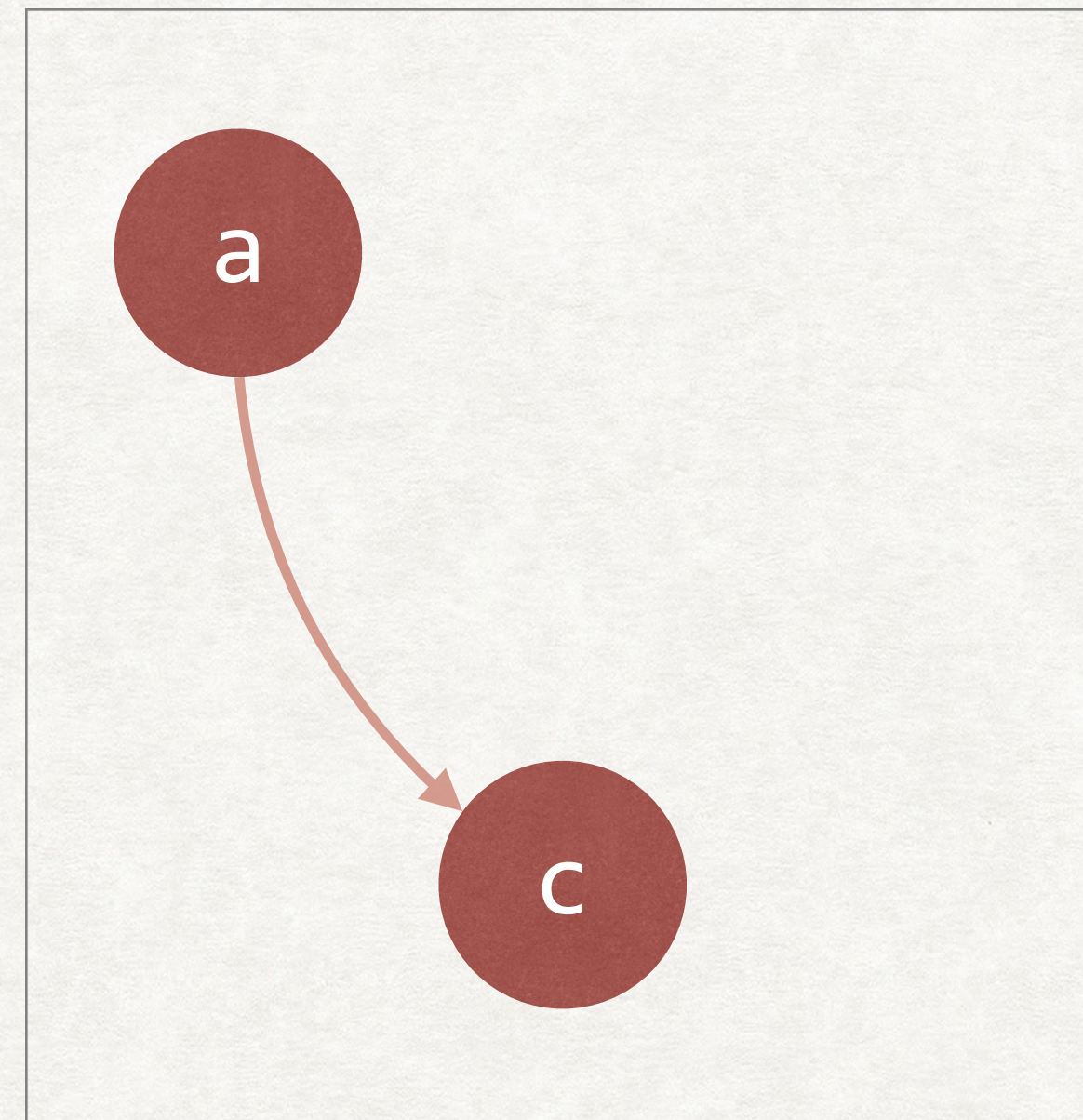
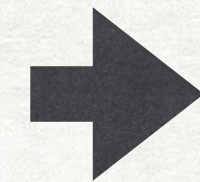
PROGRAM DEPENDENCY ANALYSIS

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



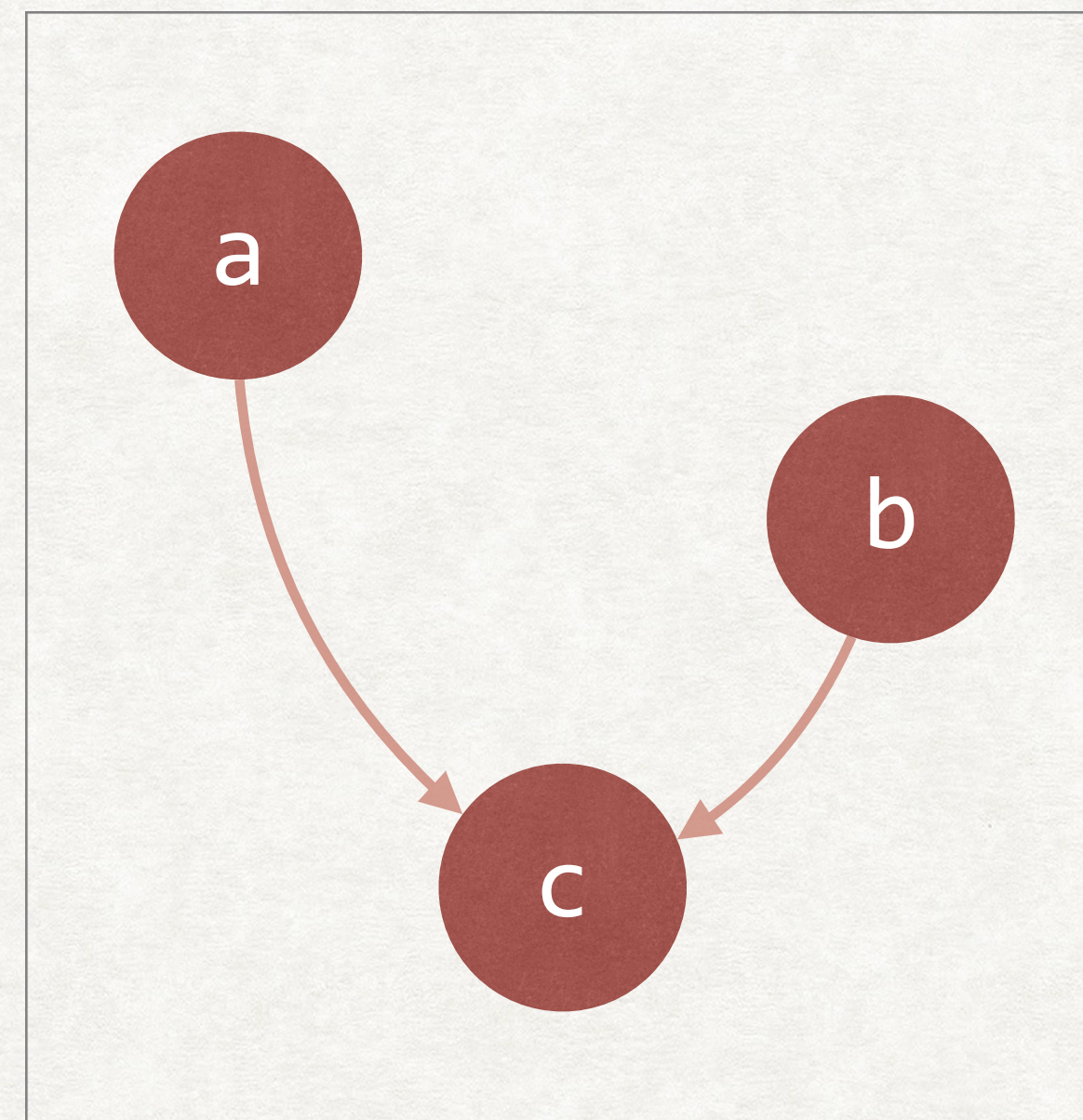
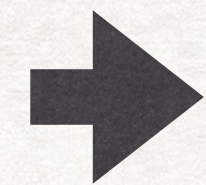
PROGRAM DEPENDENCY ANALYSIS

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



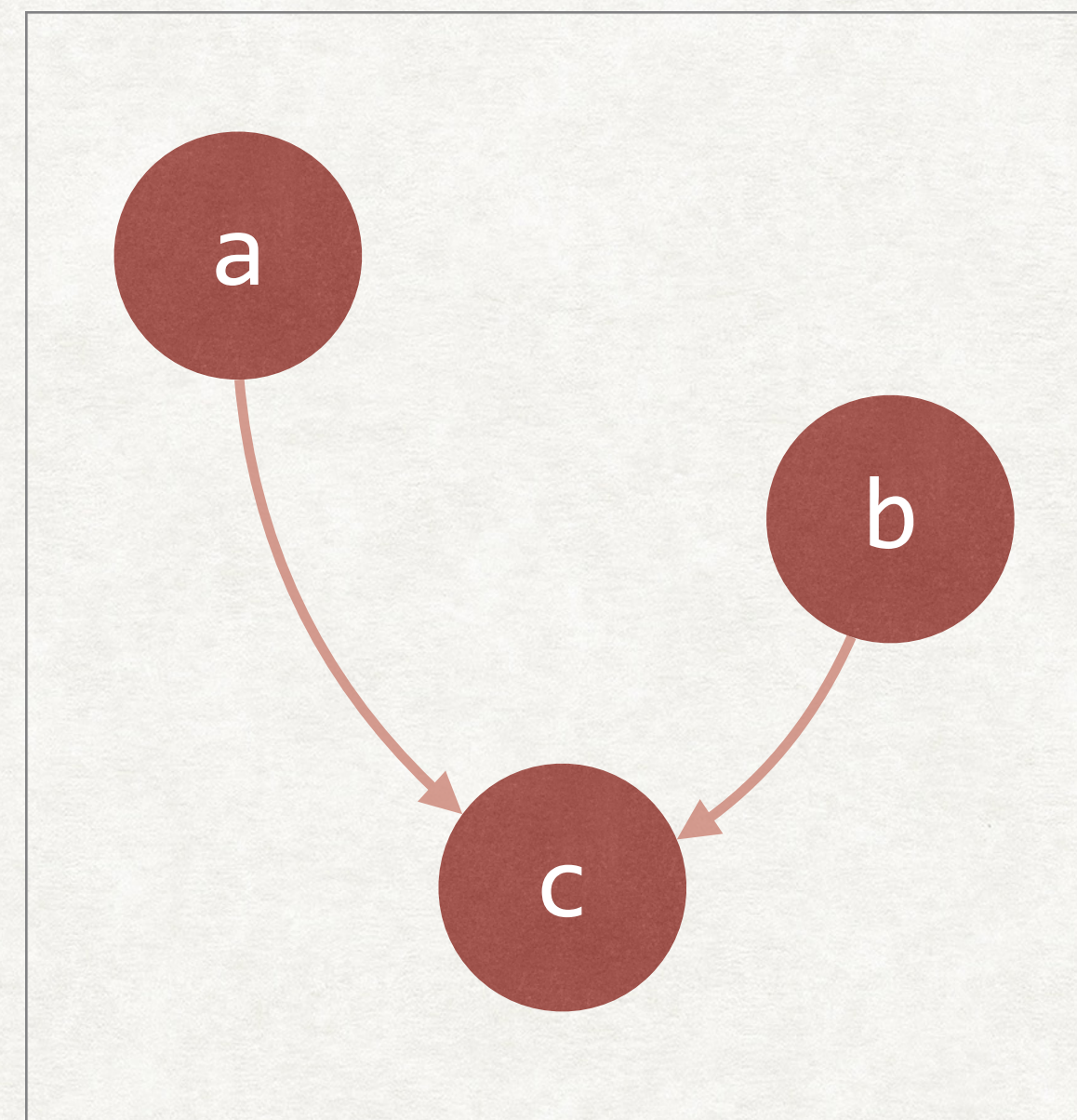
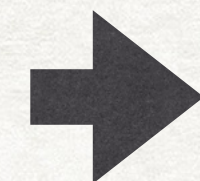
PROGRAM DEPENDENCY ANALYSIS

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



PROGRAM DEPENDENCY ANALYSIS

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



- **Program comprehension**
- **Software maintenance and evolution**
 - Where do we need to change and what are going to be affected by the change
 - Shrink the search space of the source code
 - e.g. fault localization, refactoring, code reuse, etc.

STATIC DEPENDENCY ANALYSIS

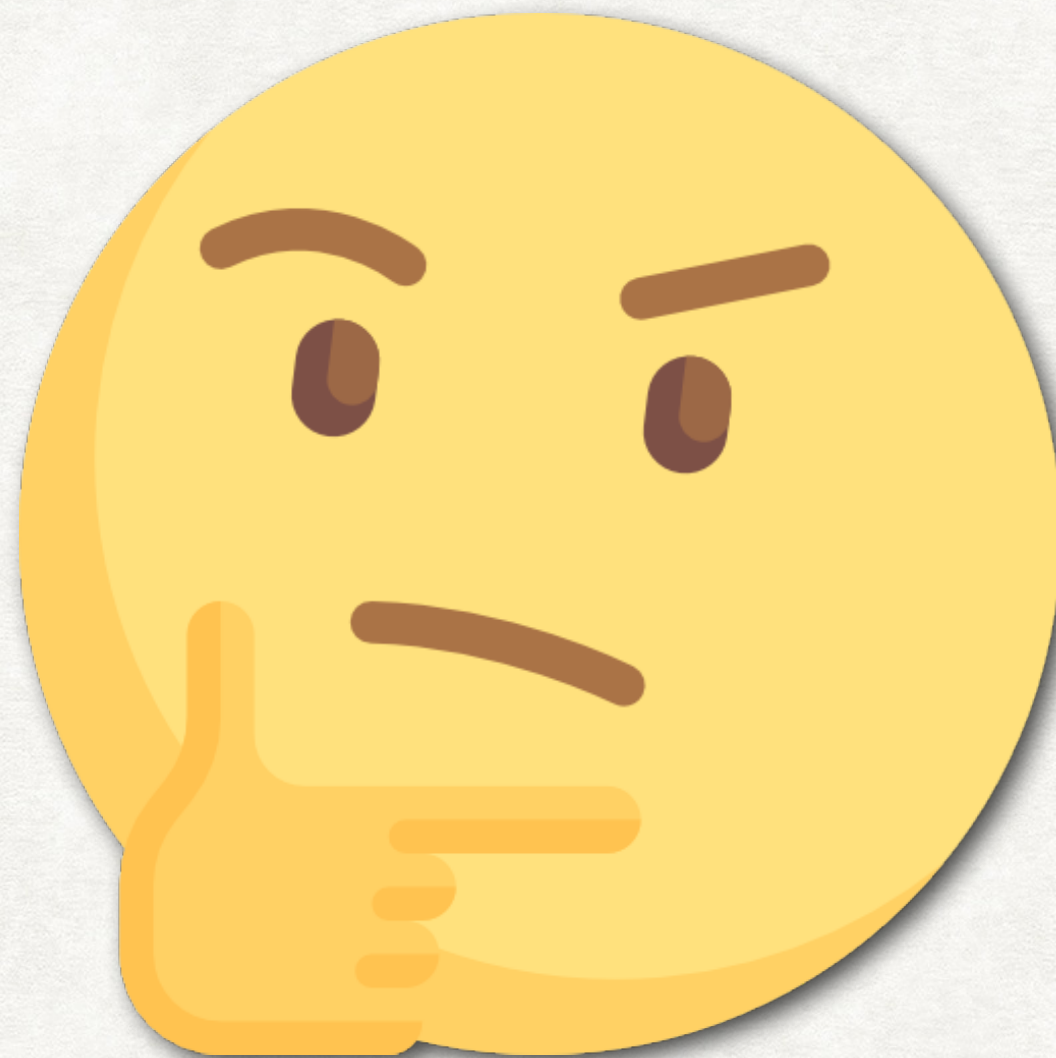
```
a = 3;  
if (b > 0) {  
    c = a + 42;  
}
```

STATIC DEPENDENCY ANALYSIS

Formal semantics

(Pass #0)	"tag" \mapsto	\top
.....		
(Pass #1)	"tag" \mapsto	$\text{struct} [\text{tagged} ["\text{tag}", 1], \pi^1]$
	π^1 =	$\{ "p" \mapsto \text{obj} [\text{ptr} [\text{obj} [\text{struct} [\text{tagged} ["\text{tag}", 1], \perp], \text{noqual}]], \text{noqual}] \}$
.....		
(Pass #2)	"tag" \mapsto	$\text{struct} [\text{tagged} ["\text{tag}", 1], \pi^2]$
	π^2 =	$\{ "p" \mapsto \text{obj} [\text{ptr} [\text{obj} [\text{struct} [\text{tagged} ["\text{tag}", 1], \pi^1], \text{noqual}]], \text{noqual}] \}$

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



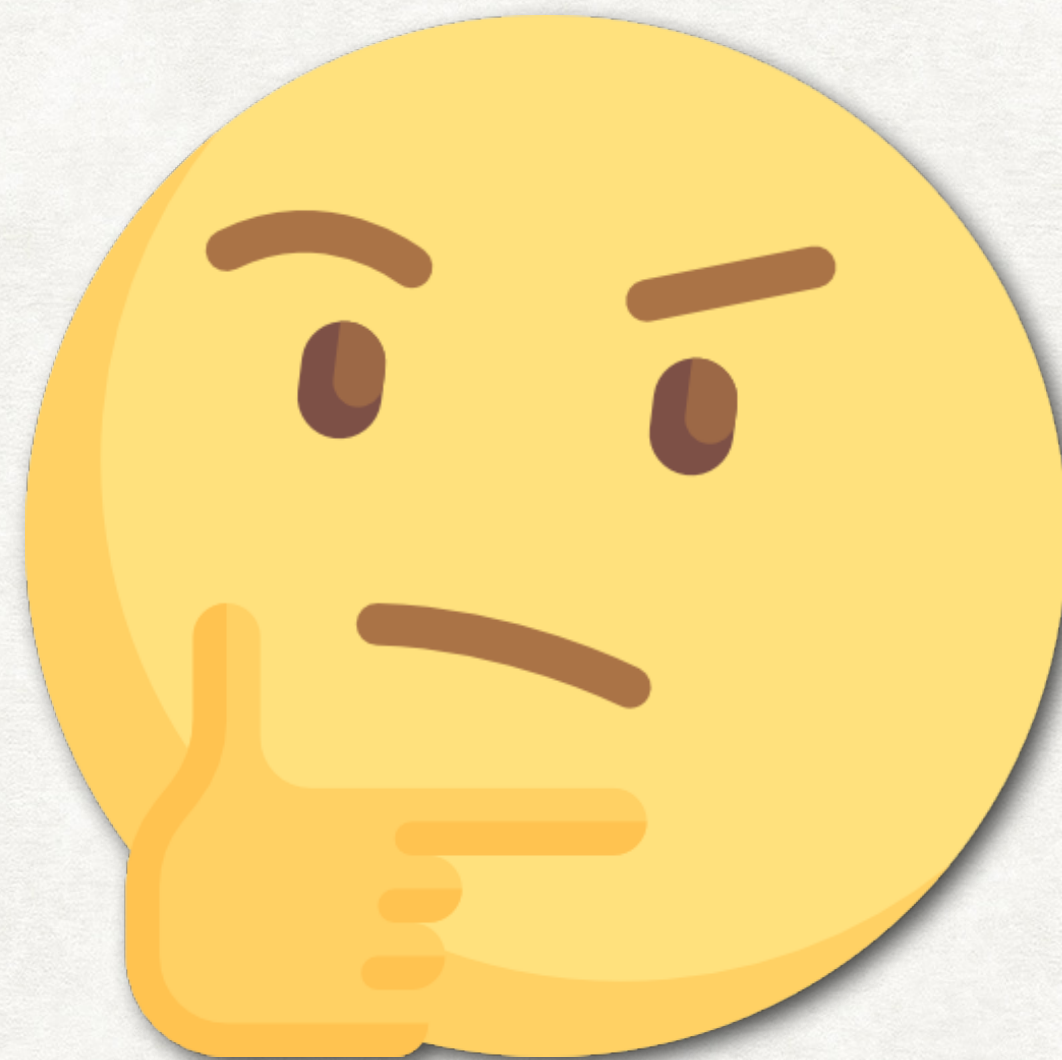
Static analyzer

STATIC DEPENDENCY ANALYSIS

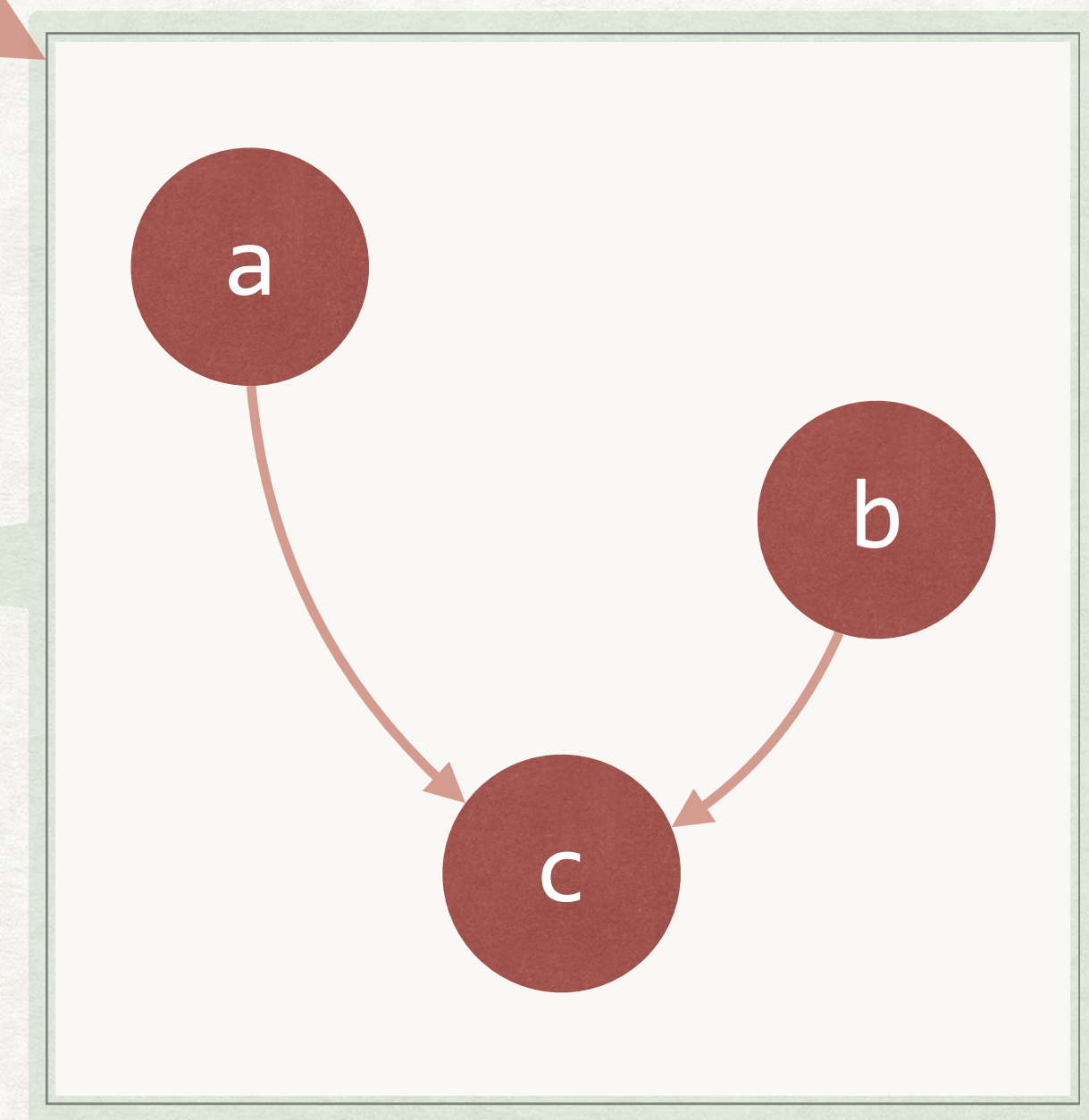
Formal semantics

(Pass #0)	"tag" \mapsto \top
(Pass #1)	"tag" \mapsto $\text{struct}[\text{tagged}["\text{tag}", 1], \pi^1]$ $\pi^1 = \{ "p" \mapsto \text{obj}[\text{ptr}[\text{obj}[\text{struct}[\text{tagged}["\text{tag}", 1], \perp], \text{noqual}]], \text{noqual}] \}$
(Pass #2)	"tag" \mapsto $\text{struct}[\text{tagged}["\text{tag}", 1], \pi^2]$ $\pi^2 = \{ "p" \mapsto \text{obj}[\text{ptr}[\text{obj}[\text{struct}[\text{tagged}["\text{tag}", 1], \pi^1], \text{noqual}]], \text{noqual}] \}$

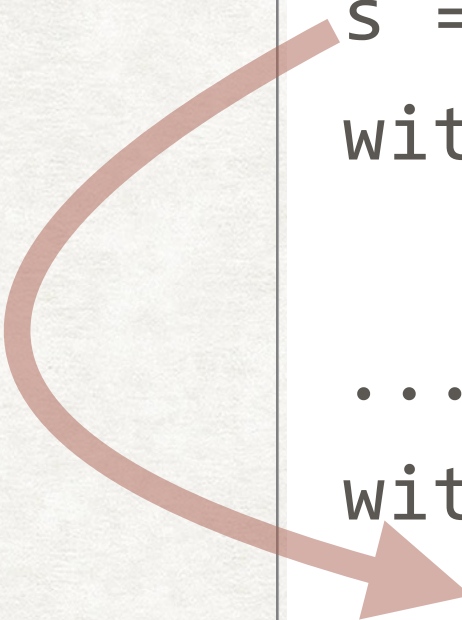
```
a = 3;  
if (b > 0) {  
    c = a + 42;  
}
```




Static analyzer



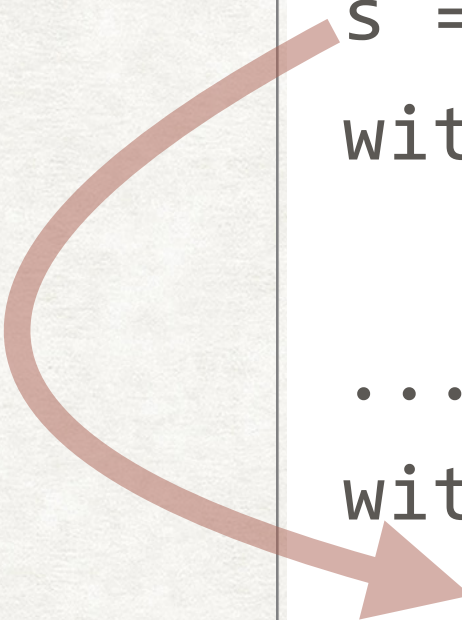
```
s = "Linux"
with open("/tmp/arch.txt", "w") as f:
    f.write(s)
...
with open("/tmp/arch.txt") as f:
    arch = f.read()
```



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



(Pass #0)	$\text{"tag"} \mapsto \top$
(Pass #1)	$\text{"tag"} \mapsto \text{struct} [\text{tagged} [\text{"tag"}, 1], \pi^1]$ $\pi^1 = \{ \text{"p"} \mapsto \text{obj} [\text{ptr} [\text{obj} [\text{struct} [\text{tagged} [\text{"tag"}, 1], \perp], \text{noqual}], \text{noqual}] \}$
(Pass #2)	$\text{"tag"} \mapsto \text{struct} [\text{tagged} [\text{"tag"}, 1], \pi^2]$ $\pi^2 = \{ \text{"p"} \mapsto \text{obj} [\text{ptr} [\text{obj} [\text{struct} [\text{tagged} [\text{"tag"}, 1], \pi^1], \text{noqual}], \text{noqual}] \}$



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

(Pass #0)	$\text{"tag"} \mapsto \top$
(Pass #1)	$\text{"tag"} \mapsto \text{struct}[\text{tagged}[\text{"tag"}, 1], \pi^1]$ $\pi^1 = \{ \text{"p"} \mapsto \text{obj}[\text{ptr}[\text{obj}[\text{struct}[\text{tagged}[\text{"tag"}, 1], \perp], \text{noqual}], \text{noqual}] \}$
(Pass #2)	$\text{"tag"} \mapsto \text{struct}[\text{tagged}[\text{"tag"}, 1], \pi^2]$ $\pi^2 = \{ \text{"p"} \mapsto \text{obj}[\text{ptr}[\text{obj}[\text{struct}[\text{tagged}[\text{"tag"}, 1], \pi^1], \text{noqual}], \text{noqual}] \}$



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

I don't know!



Static analyzer

LIMITATION OF STATIC DEPENDENCE ANALYSIS

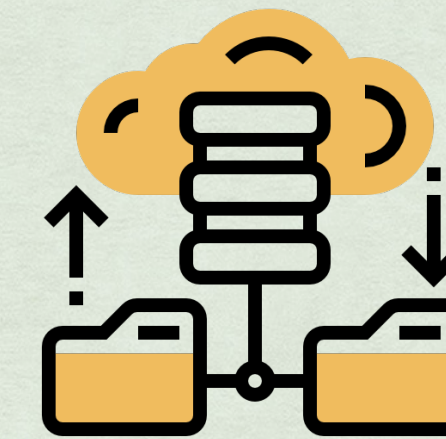
LIMITATION OF STATIC DEPENDENCE ANALYSIS

Various features have no formal semantics

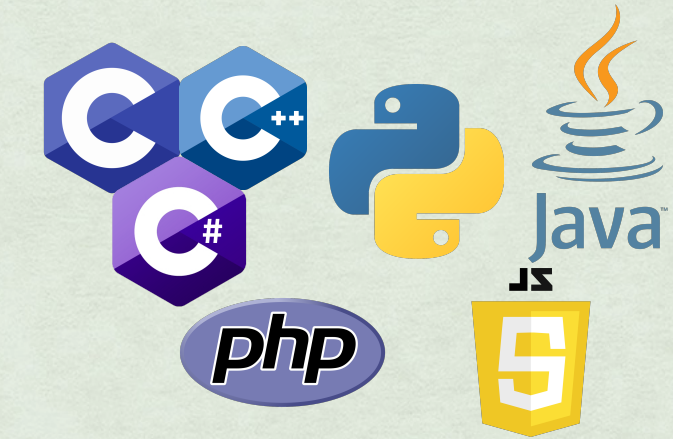
(Pass #0)	"tag" \mapsto \top
(Pass #1)	"tag" \mapsto $\text{struct}[\text{tagged}["\text{tag}", 1], \pi^1]$ $\pi^1 = \{ "p" \mapsto \text{obj}[\text{ptr}[\text{obj}[\text{struct}[\text{tagged}["\text{tag}", 1], \perp], \text{noqual}]], \text{noqual}] \}$
(Pass #2)	"tag" \mapsto $\text{struct}[\text{tagged}["\text{tag}", 1], \pi^2]$ $\pi^2 = \{ "p" \mapsto \text{obj}[\text{ptr}[\text{obj}[\text{struct}[\text{tagged}["\text{tag}", 1], \pi^1], \text{noqual}]], \text{noqual}] \}$



Binary
libraries



External
database



Multi-lingual
program

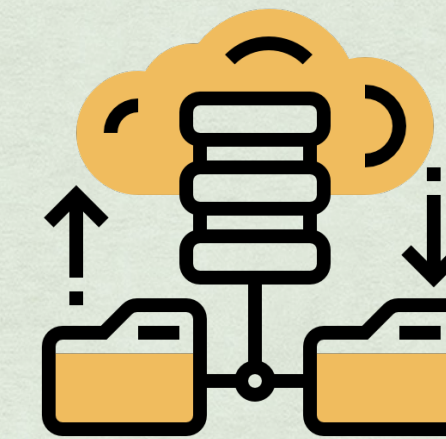
LIMITATION OF STATIC DEPENDENCE ANALYSIS

Various features have no formal semantics

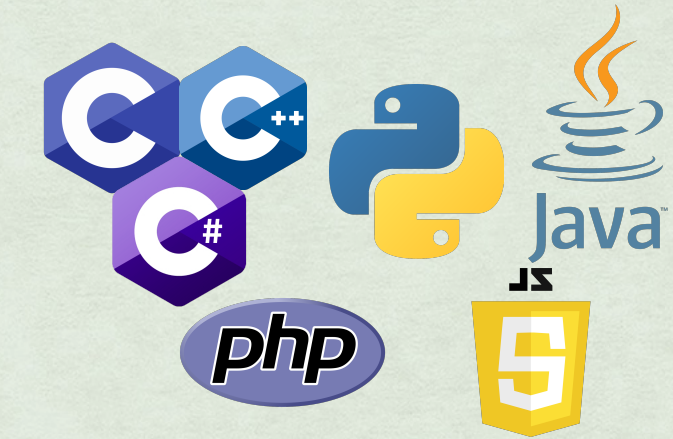
(Pass #0)	"tag" $\mapsto \top$
(Pass #1)	"tag" $\mapsto \text{struct}[\text{tagged}["\text{tag}", 1], \pi^1]$ $\pi^1 = \{ "p" \mapsto \text{obj}[\text{ptr}[\text{obj}[\text{struct}[\text{tagged}["\text{tag}", 1], \perp], \text{noqual}]], \text{noqual}] \}$
(Pass #2)	"tag" $\mapsto \text{struct}[\text{tagged}["\text{tag}", 1], \pi^2]$ $\pi^2 = \{ "p" \mapsto \text{obj}[\text{ptr}[\text{obj}[\text{struct}[\text{tagged}["\text{tag}", 1], \pi^1], \text{noqual}]], \text{noqual}] \}$



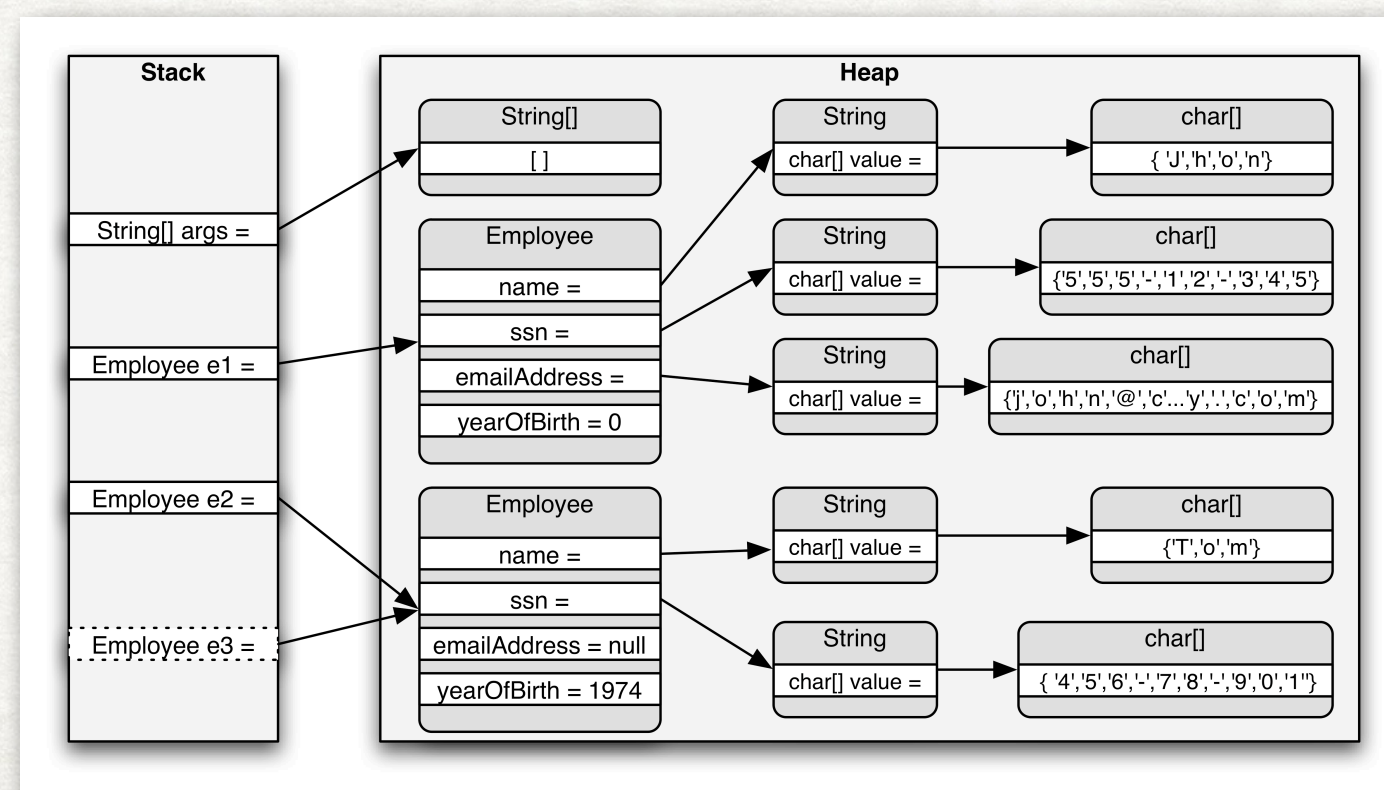
Binary
libraries



External
database

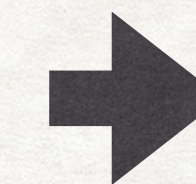


Multi-lingual
program



Heavy computation cost

- E.g. pointer analysis



Lack scalability

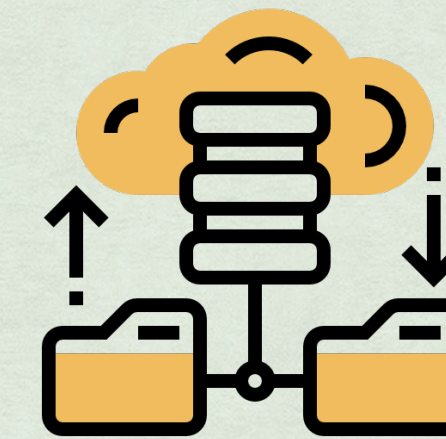
LIMITATION OF STATIC DEPENDENCE ANALYSIS

Various features have no formal semantics

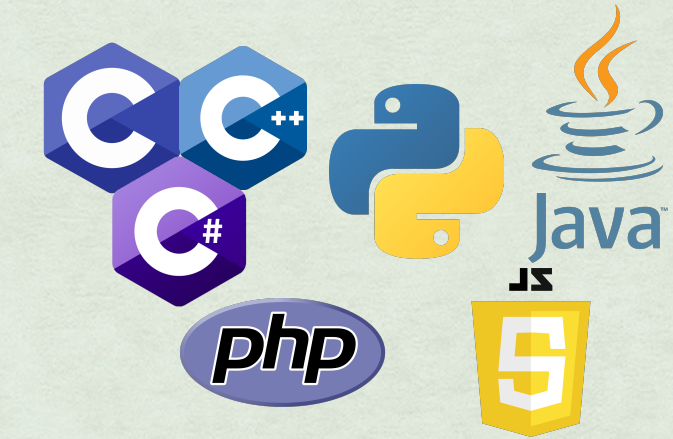
(Pass #0)	"tag" \mapsto \top
(Pass #1)	"tag" \mapsto $\text{struct}[\text{tagged}["\text{tag}", 1], \pi^1]$ $\pi^1 = \{ "p" \mapsto \text{obj}[\text{ptr}[\text{obj}[\text{struct}[\text{tagged}["\text{tag}", 1], \perp], \text{noqual}]], \text{noqual}] \}$
(Pass #2)	"tag" \mapsto $\text{struct}[\text{tagged}["\text{tag}", 1], \pi^2]$ $\pi^2 = \{ "p" \mapsto \text{obj}[\text{ptr}[\text{obj}[\text{struct}[\text{tagged}["\text{tag}", 1], \pi^1], \text{noqual}]], \text{noqual}] \}$



Binary
libraries



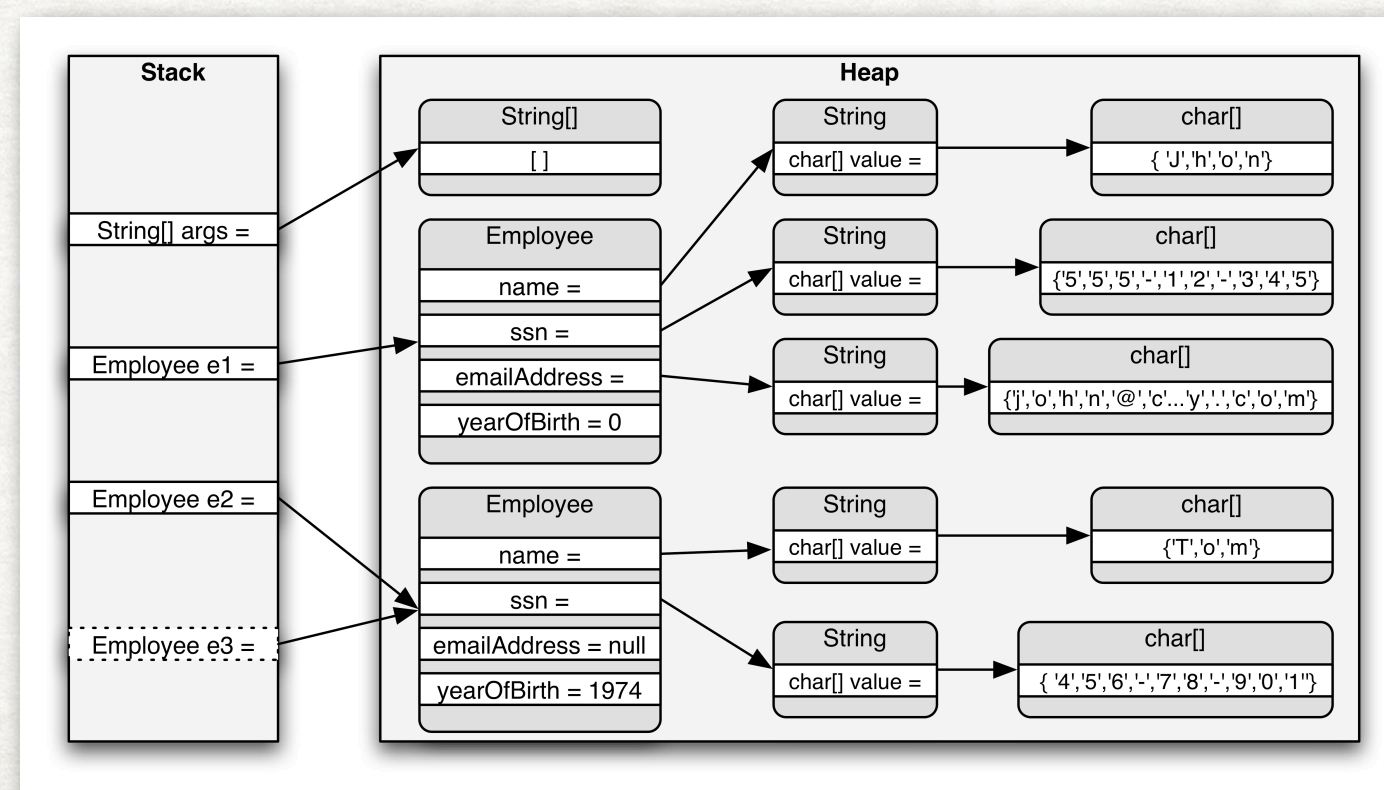
External
database



Multi-lingual
program

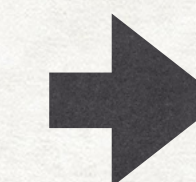


Inaccurate result!



Heavy computation cost

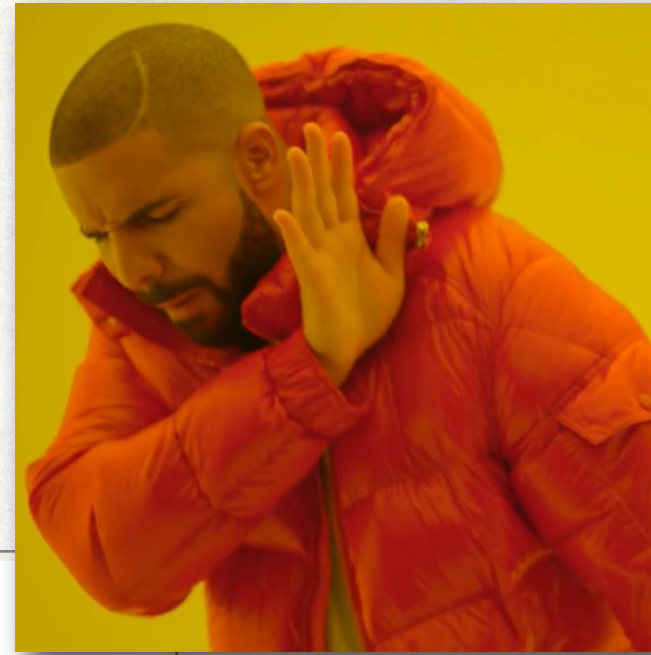
- E.g. pointer analysis



Lack scalability

OBSERVATIONAL KNOWLEDGE

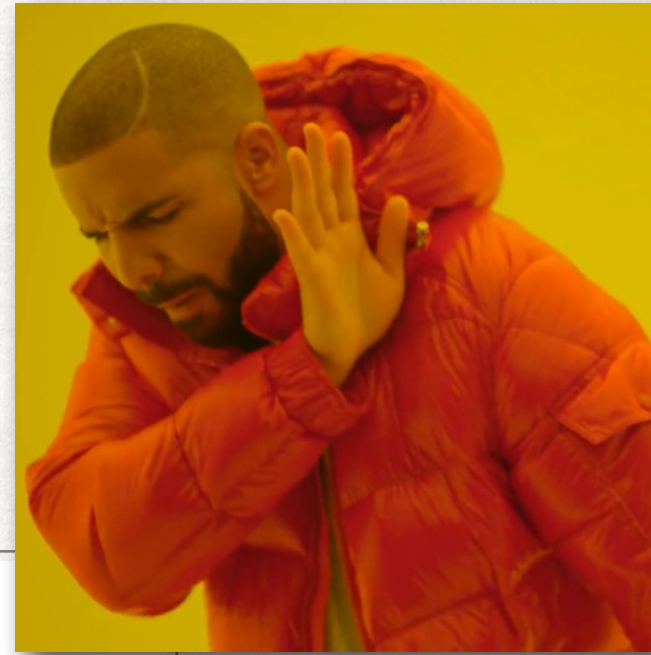
OBSERVATIONAL KNOWLEDGE



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

Linux

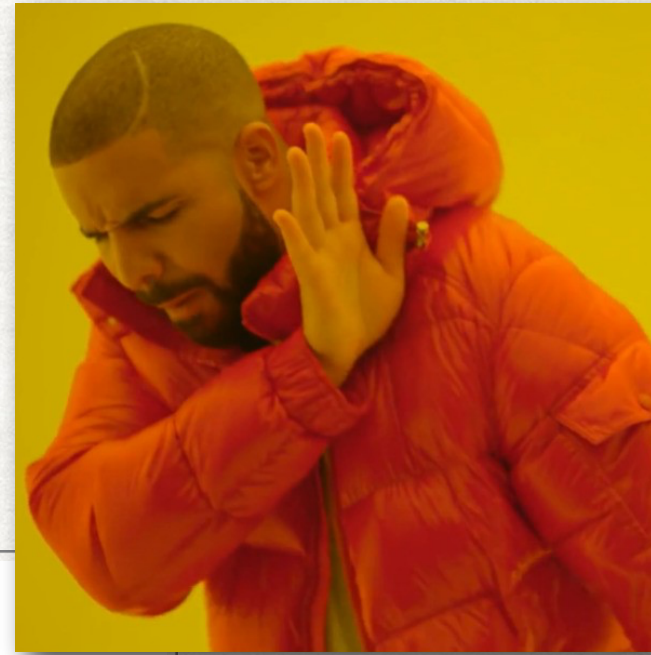
OBSERVATIONAL KNOWLEDGE



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

Linux

OBSERVATIONAL KNOWLEDGE



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

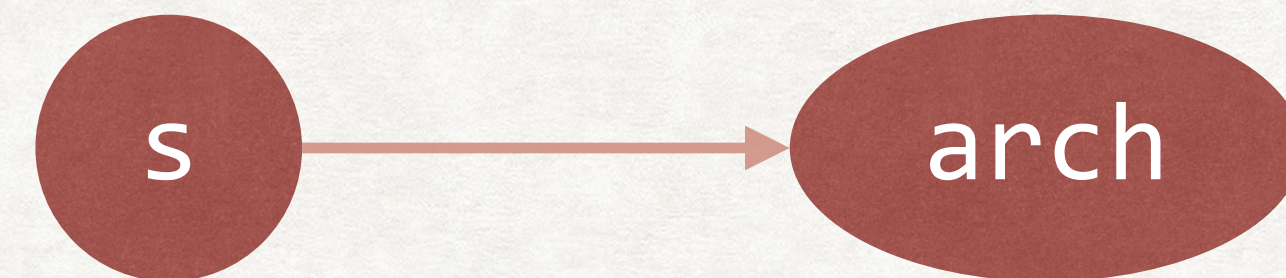
Window

OBSERVATIONAL KNOWLEDGE



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

Window



OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

OBSERVATION BASED SLICING (ORBS)

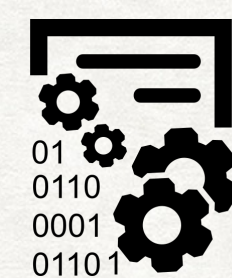
- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

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

OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

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



EXECUTE

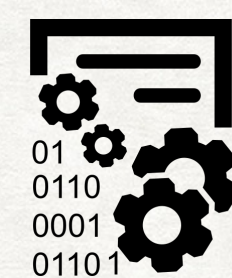


"i = 11"

OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

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



EXECUTE



"i = 11"

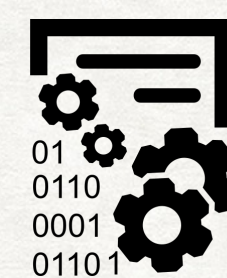


Oracle

OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

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



EXECUTE



Compilation
error!



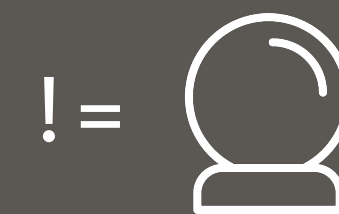
OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

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



Compilation
error!



OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

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



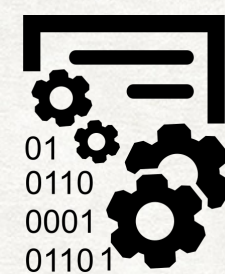
Compilation
error!



OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

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



EXECUTE



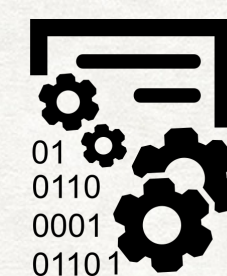
Compilation
error!



OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

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



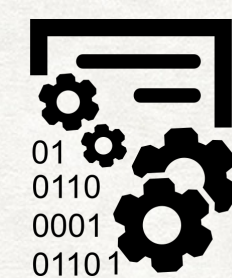
EXECUTE



OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

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



EXECUTE



“ ”

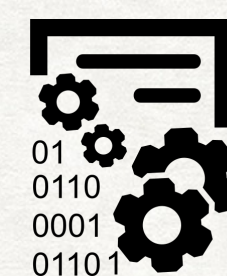
!=



OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

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



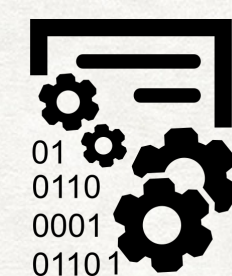
EXECUTE



OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

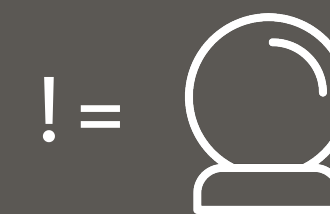
```
int main() {  
    int sum = 0;  
    int i = 1;  
    while (i < 11) {  
        sum = sum + i;  
        i = i + 1;  
    }  
    printf("%d\n", sum);  
    printf("ORBS: %d\n", i);  
}
```



EXECUTE



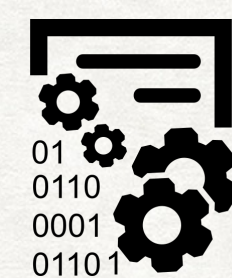
Compilation
error!



OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

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



EXECUTE



Compilation
error!



OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

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



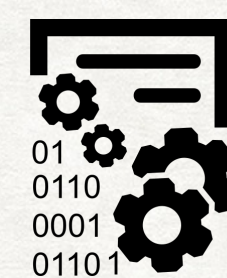
EXECUTE



OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

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



EXECUTE



"i = 11"

==

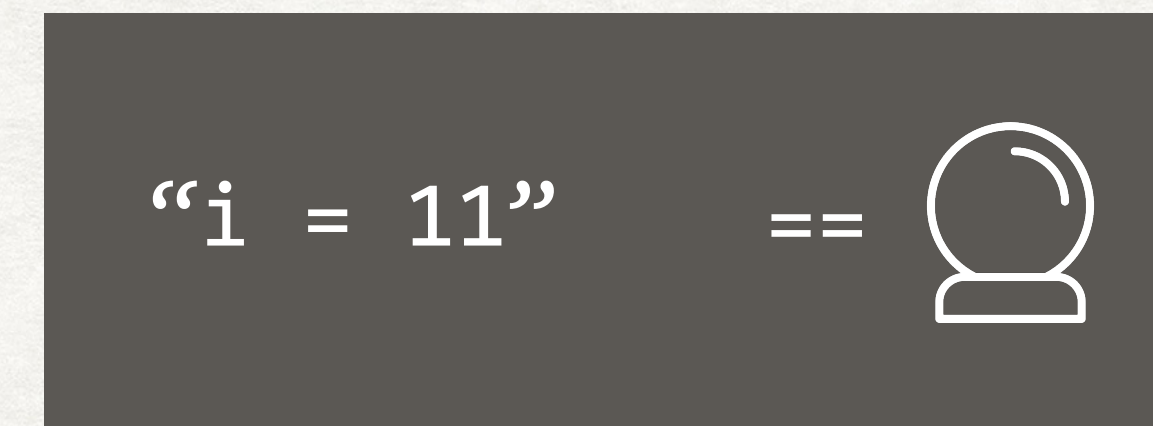


Trajectory preserved!

OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

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

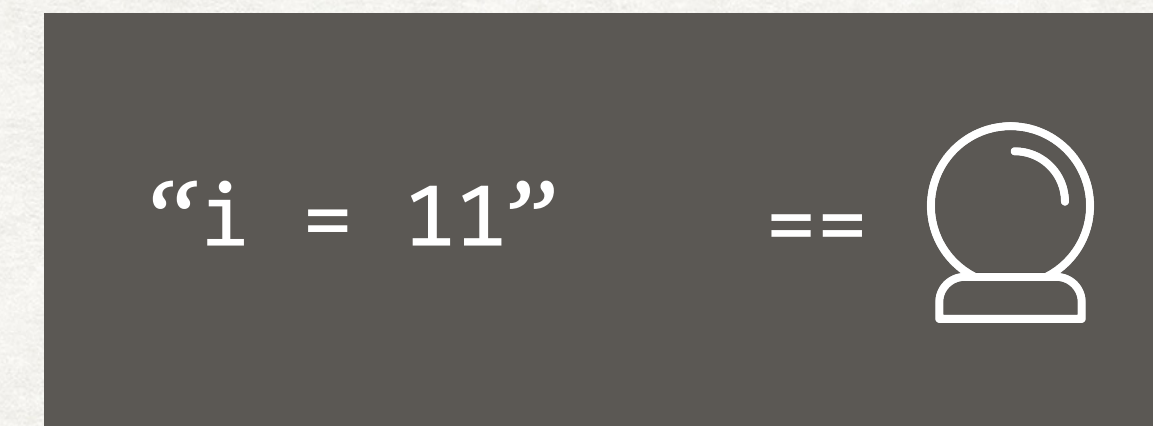


Trajectory preserved!

OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

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



Trajectory preserved!

OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution



OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

Purely dynamic



OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

```
checker.java:
1 class checker {
2   public static void main(String[] args) {
3     int dots = 0;
4     for (int i = 0; i < args[0].length(); ++i) {
5       if (args[0].charAt(i) == '.') {
6         ++dots;
7       }
8     }
9   }
}

reader.c:
1 #include <locale.h>
2 int main(int argc, char **argv) {
3   struct lconv *cur_locale = localeconv();
4   {
5     printf("%s\n", cur_locale->decimal_point);
6   }
7 }

glue.py:
1 import commands
2 import sys
3 use_locale = True
4 currency = "?"
5 if use_locale:
6   decimal = commands.getoutput('./reader 1')
7   cmd = ('java checker ' + currency
8         + sys.argv[1] + decimal + sys.argv[2])
9   print commands.getoutput(cmd)
```

Purely dynamic

Multi-lingual
Program



OBSERVATION BASED SLICING (ORBS)

- **Program slicing:** computes a subset of a program such that executing the subset will have the same behavior for a *slicing criterion*
 - *Slicing criterion:* specified variable at a specified location of interest
 - *Trajectory:* a sequence of values the slicing criterion has during an execution

```
checker.java:
1 class checker {
2   public static void main(String[] args) {
3     int dots = 0;
4     for (int i = 0; i < args[0].length(); ++i) {
5       if (args[0].charAt(i) == '.') {
6         ++dots;
7       }
8     }
9   }
}

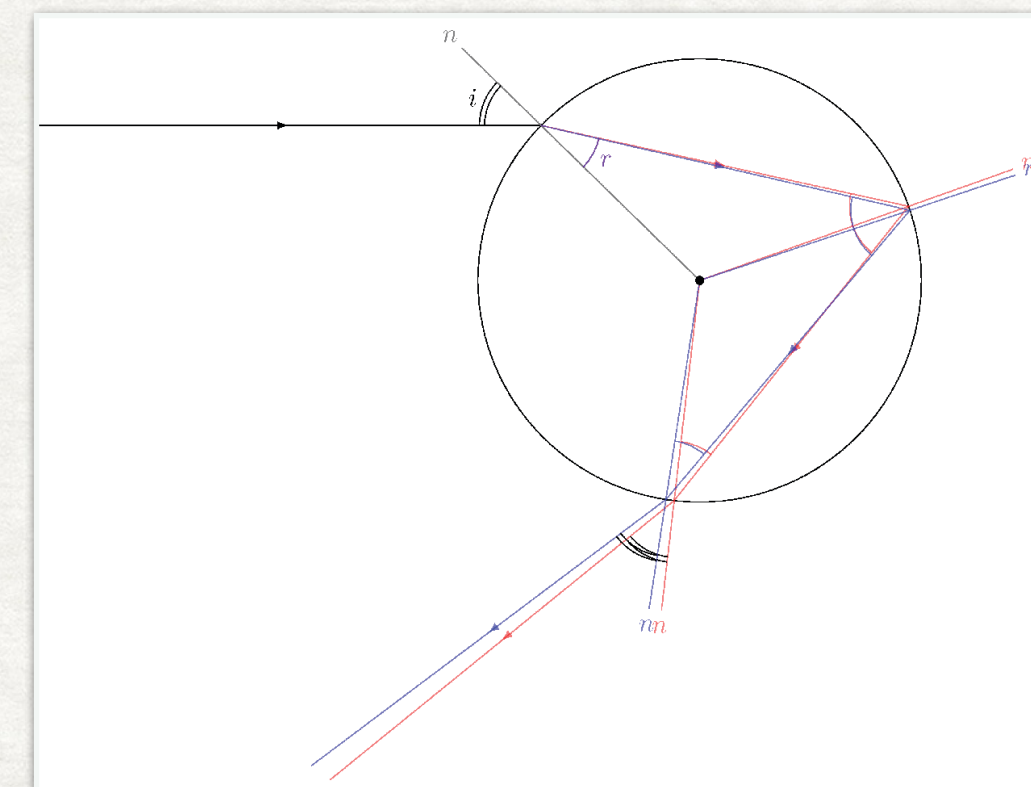
reader.c:
1 #include <locale.h>
2 int main(int argc, char **argv) {
3   struct lconv *cur_locale = localeconv();
4   {
5     printf("%s\n", cur_locale->decimal_point);
6   }
7 }

glue.py:
1 import commands
2 import sys
3 use_locale = True
4 currency = "?"
5 if use_locale:
6   decimal = commands.getoutput('./reader 1')
7   cmd = ('java checker ' + currency
8         + sys.argv[1] + decimal + sys.argv[2])
9   print commands.getoutput(cmd)
```

Purely dynamic

Multi-lingual
Program

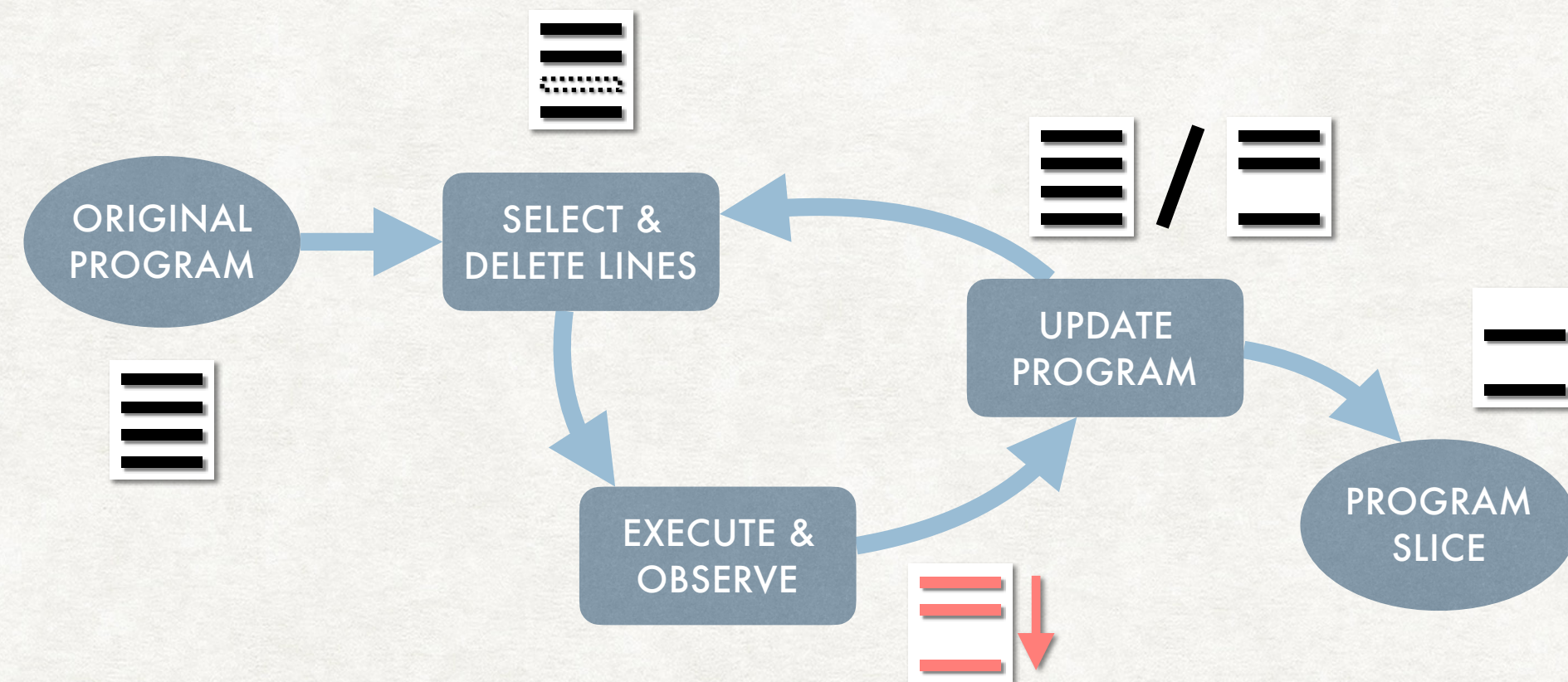
Picture
Description
Language



LIMITATION OF ORBS

LIMITATION OF ORBS

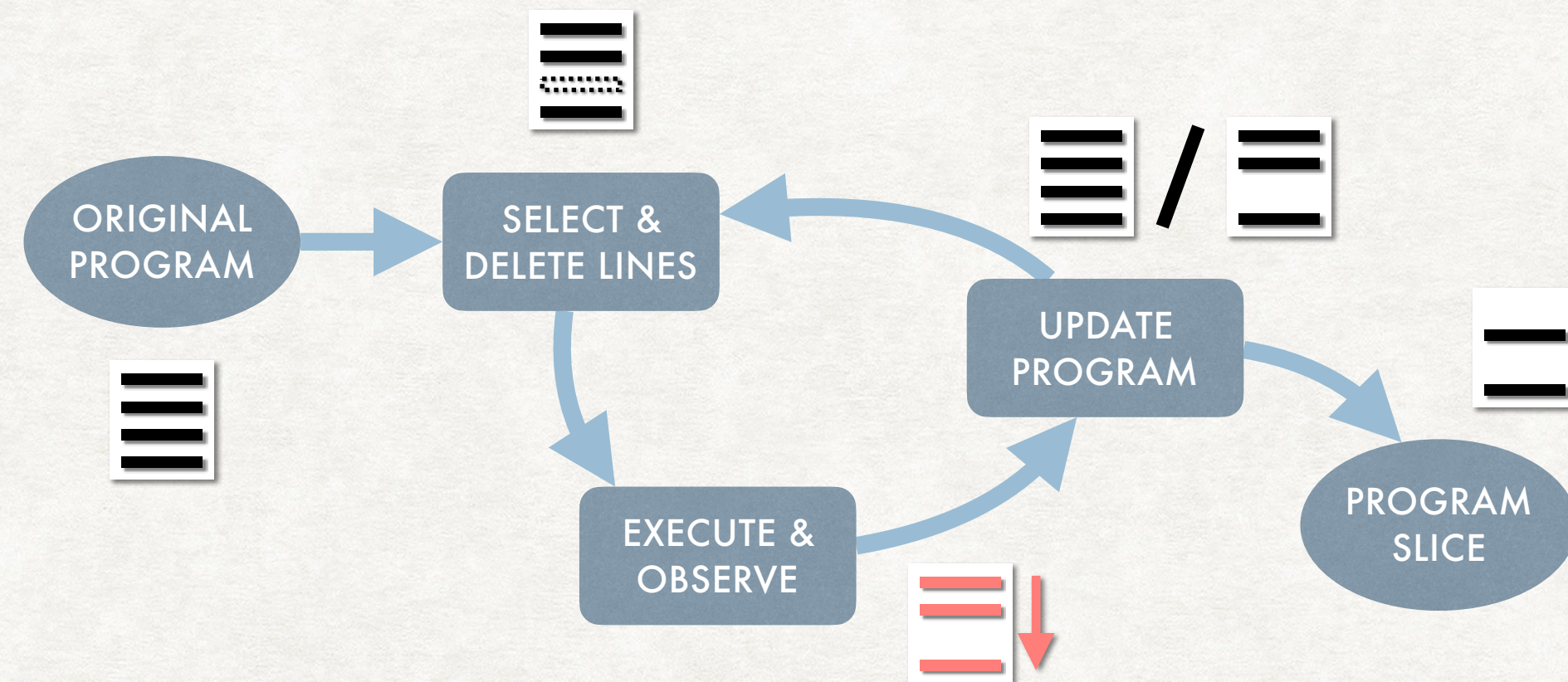
SCALABILITY



Requires a large number of compilations & executions

LIMITATION OF ORBS

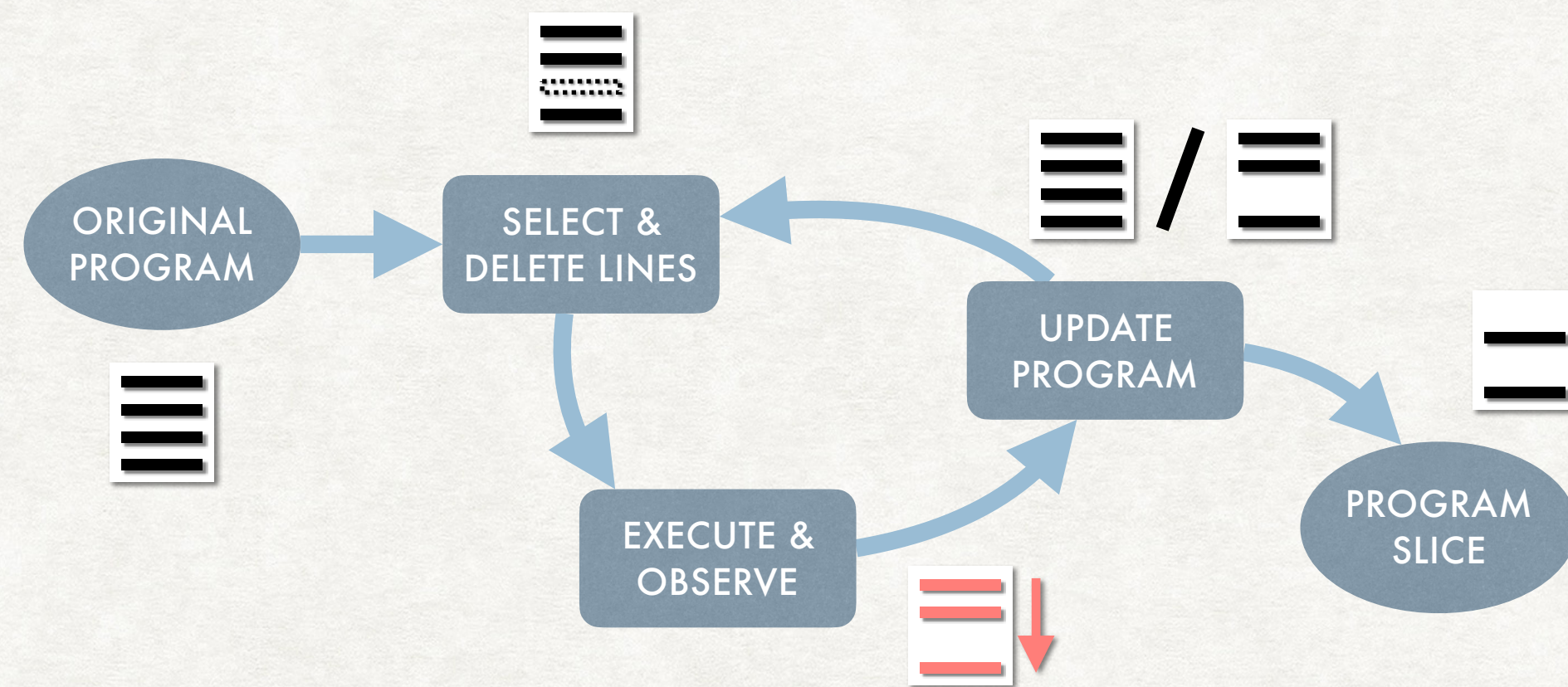
SCALABILITY



Requires a large number of compilations & executions

LIMITATION OF ORBS

SCALABILITY



Requires a large number of compilations & executions

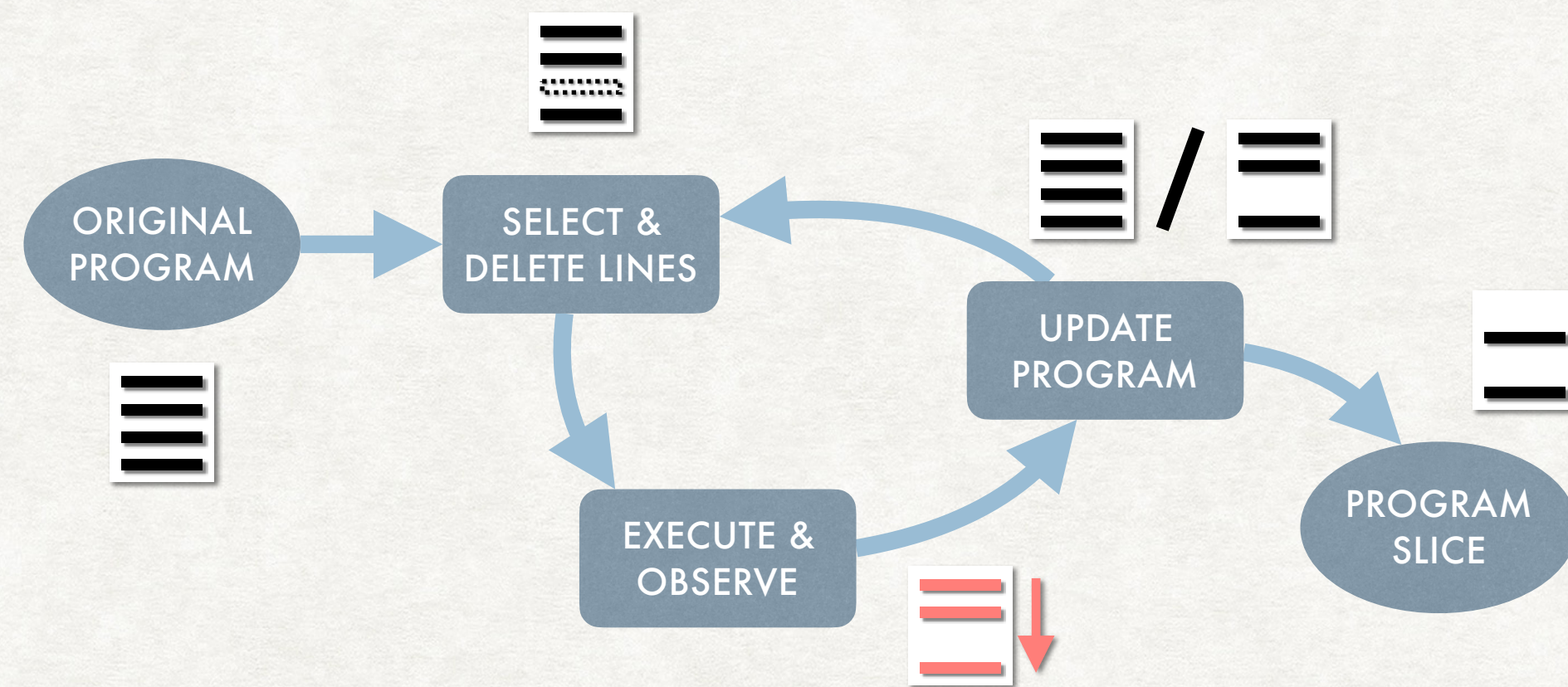
EXPLAINABILITY

ALL
PROGRAM
ELEMENT

$$\mathbb{E} \ni \{e_1, e_2\} \rightarrow e_3$$

LIMITATION OF ORBS

SCALABILITY



Requires a large number of compilations & executions

EXPLAINABILITY

ALL
PROGRAM
ELEMENT

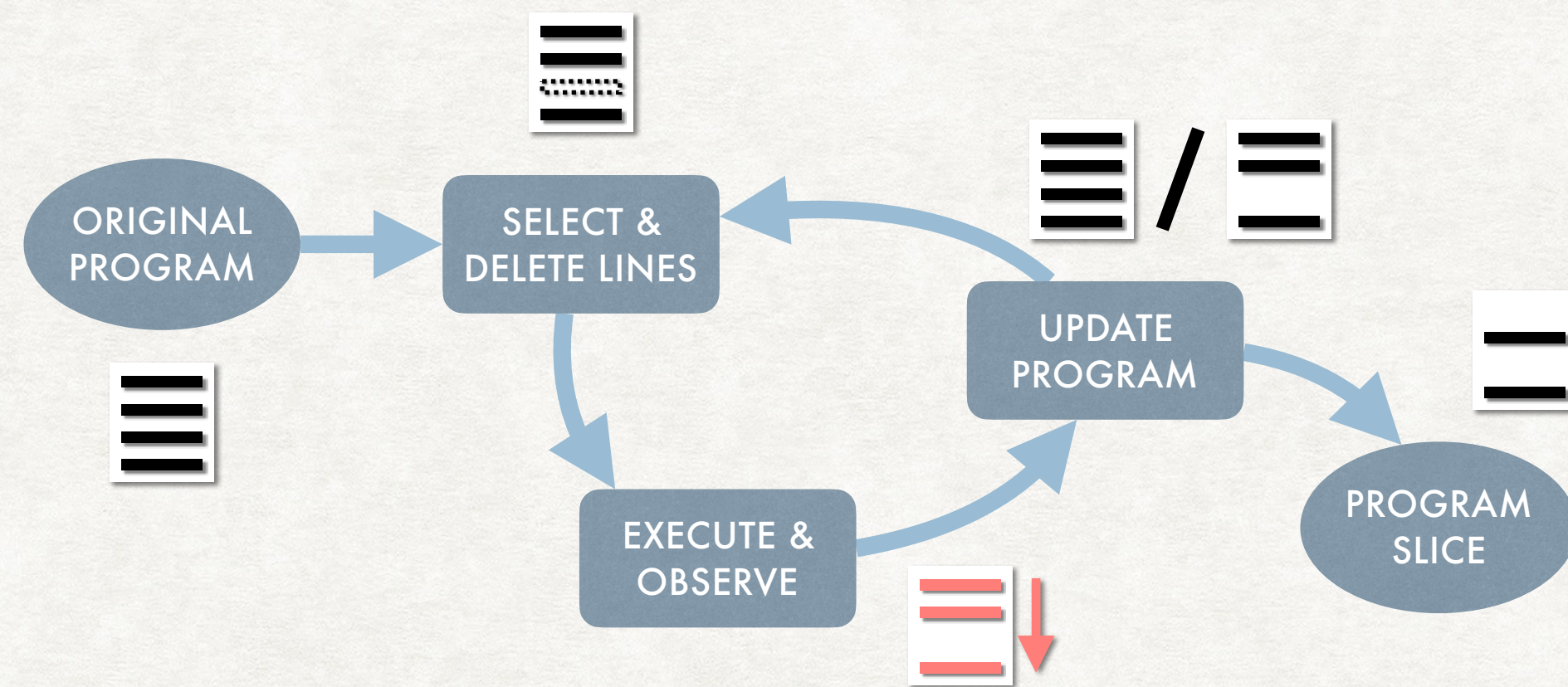
$$\mathbb{E} \ni \{e_1, e_2\} \rightarrow e_3$$

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

Another element

LIMITATION OF ORBS

SCALABILITY

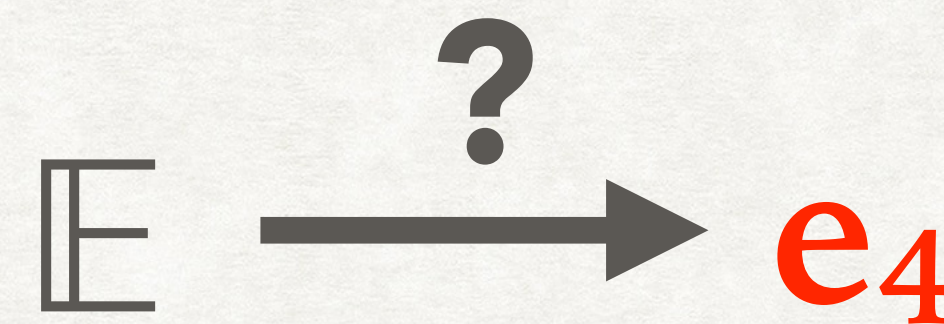


Requires a large number of compilations & executions

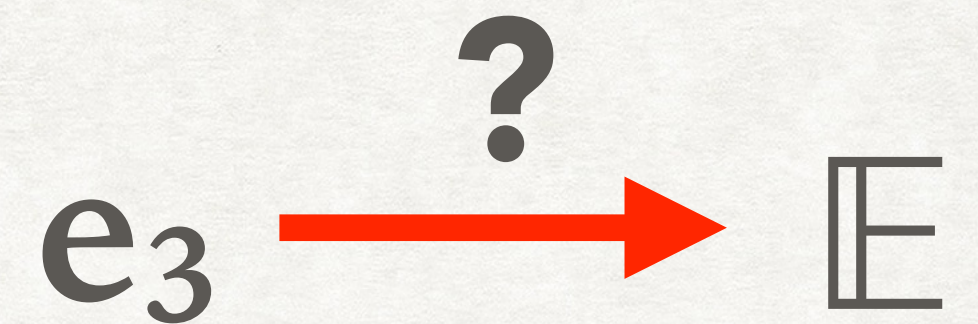
EXPLAINABILITY

ALL
PROGRAM
ELEMENT

$$\mathbb{E} \ni \{e_1, e_2\} \longrightarrow e_3$$



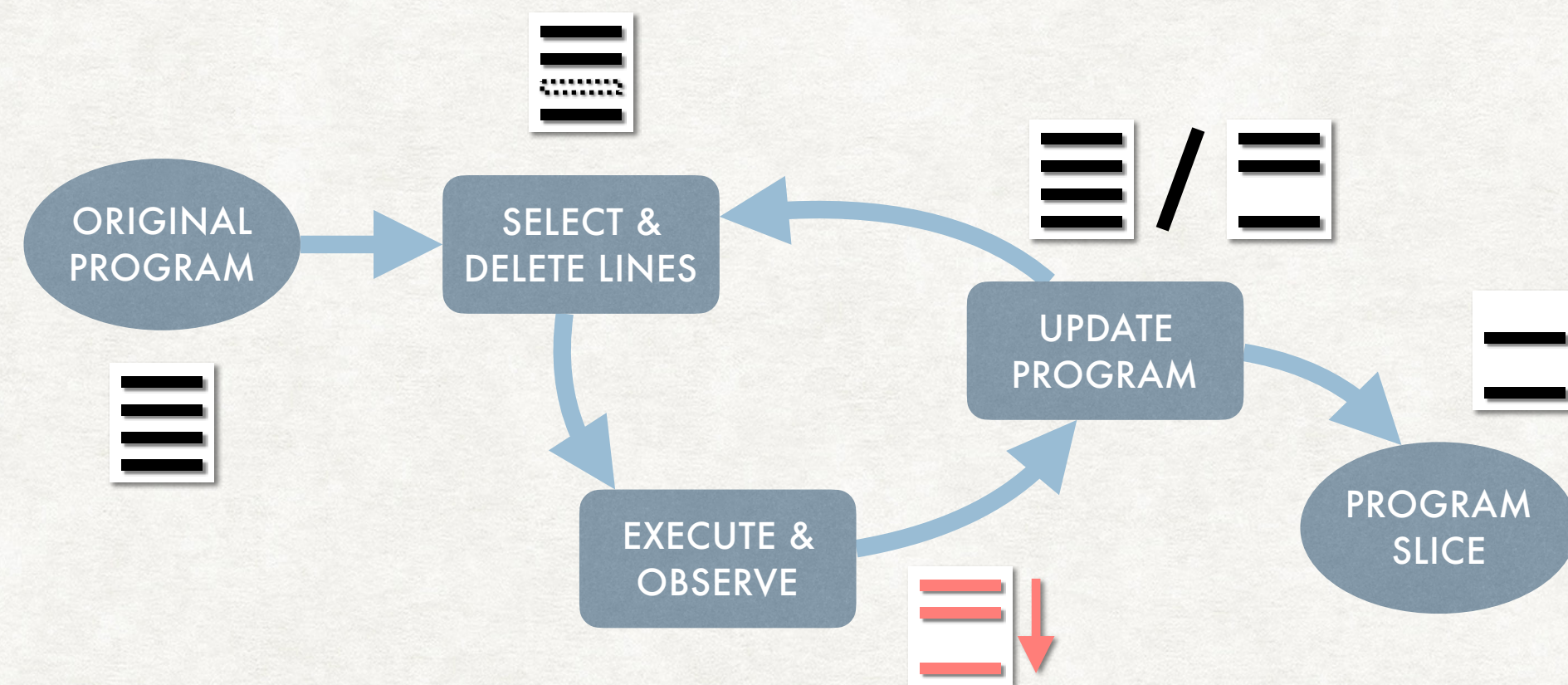
Another element



Forward dependency

LIMITATION OF ORBS

SCALABILITY

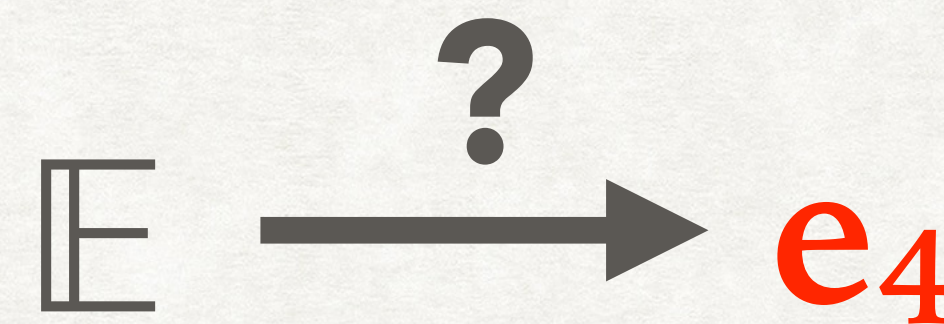


Requires a large number of compilations & executions

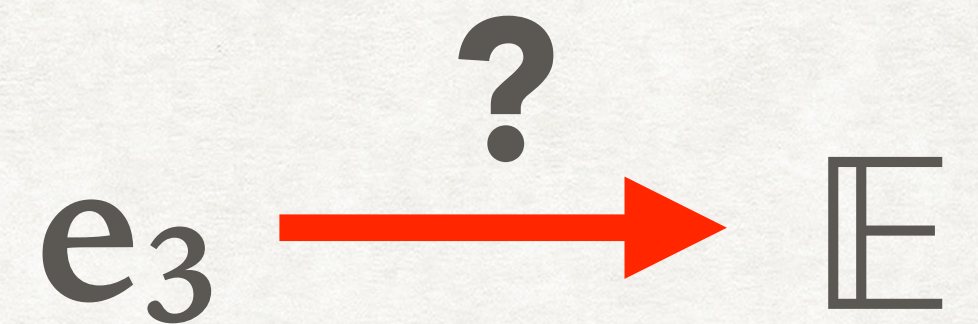
EXPLAINABILITY

ALL
PROGRAM
ELEMENT

$$\mathbb{E} \ni \{e_1, e_2\} \rightarrow e_3$$



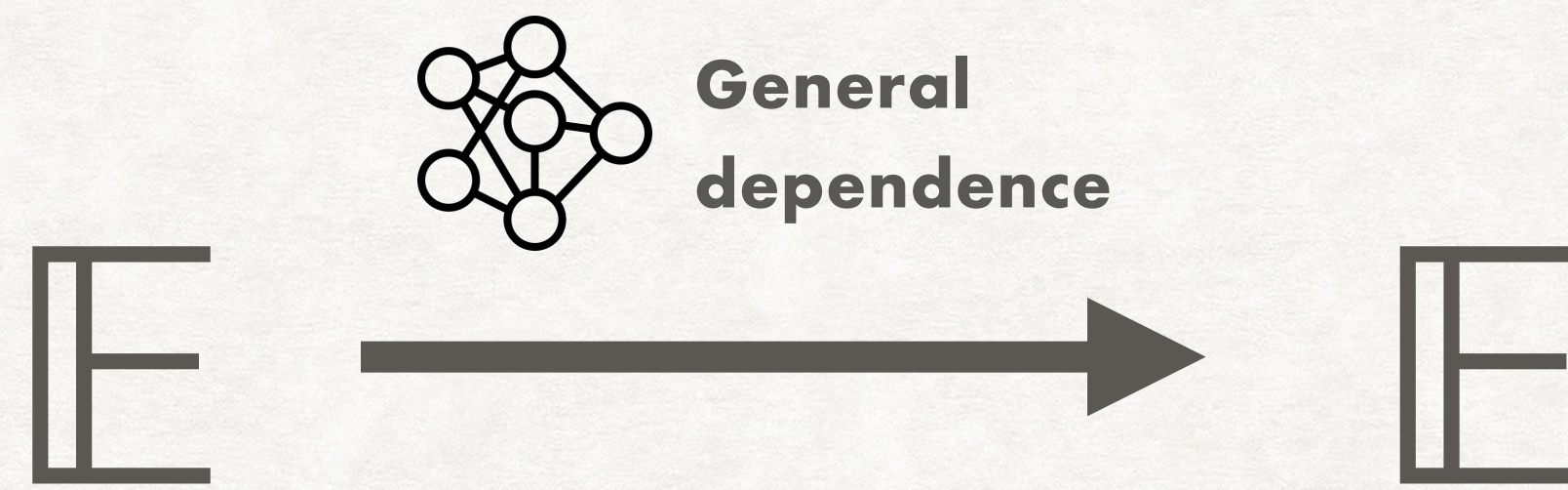
Another element



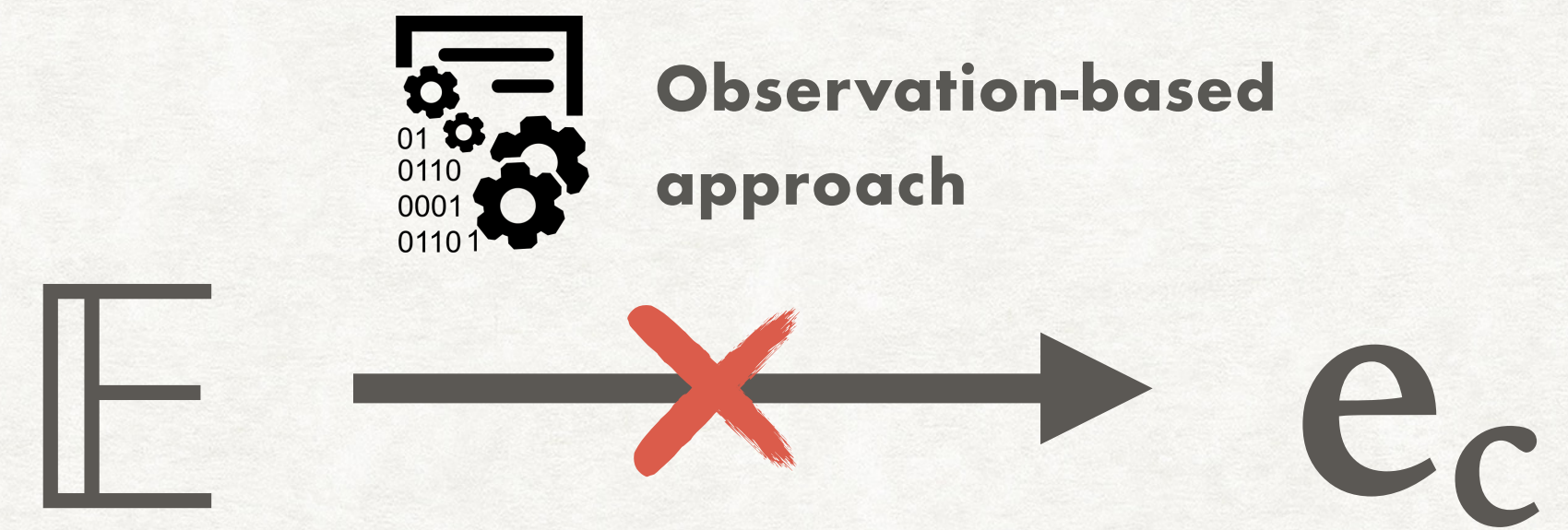
Forward dependency

No general dependence / only a single slice

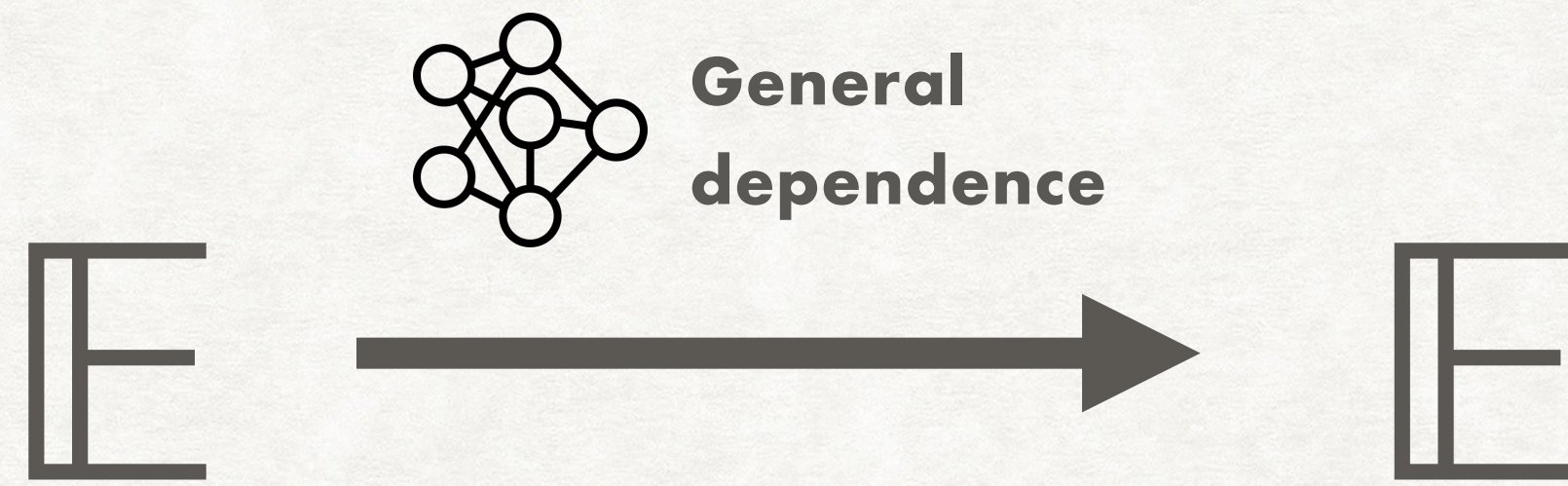
STATIC ANALYSIS



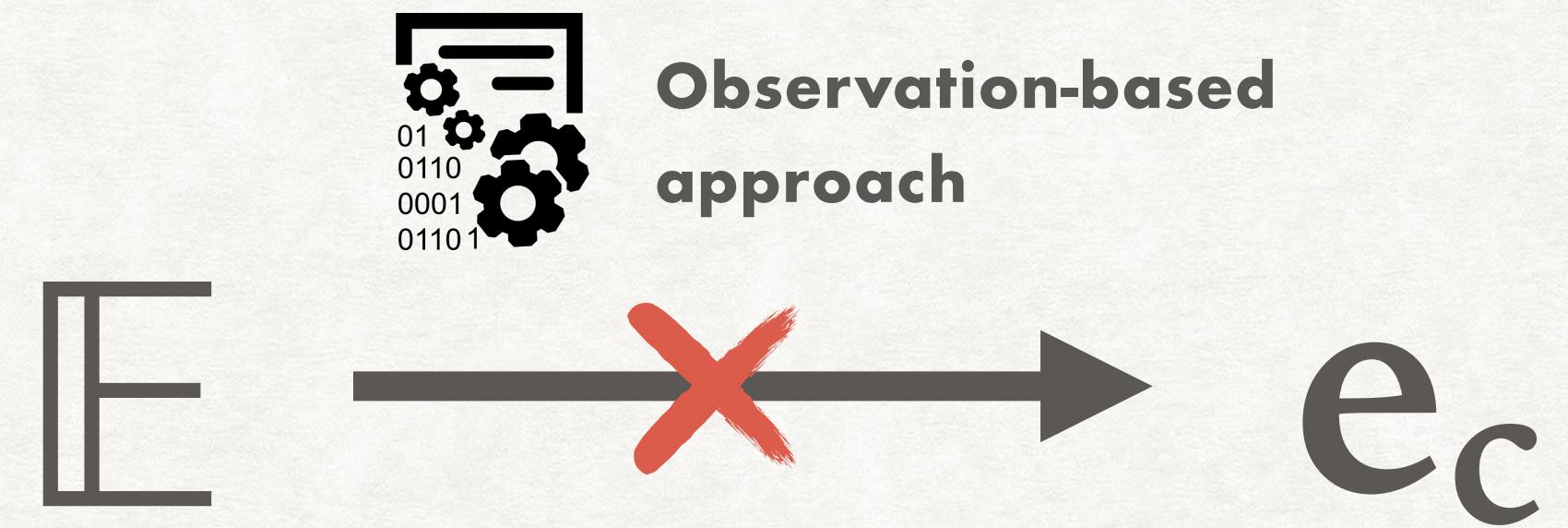
ORBS



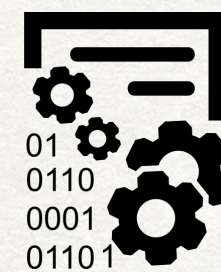
STATIC ANALYSIS



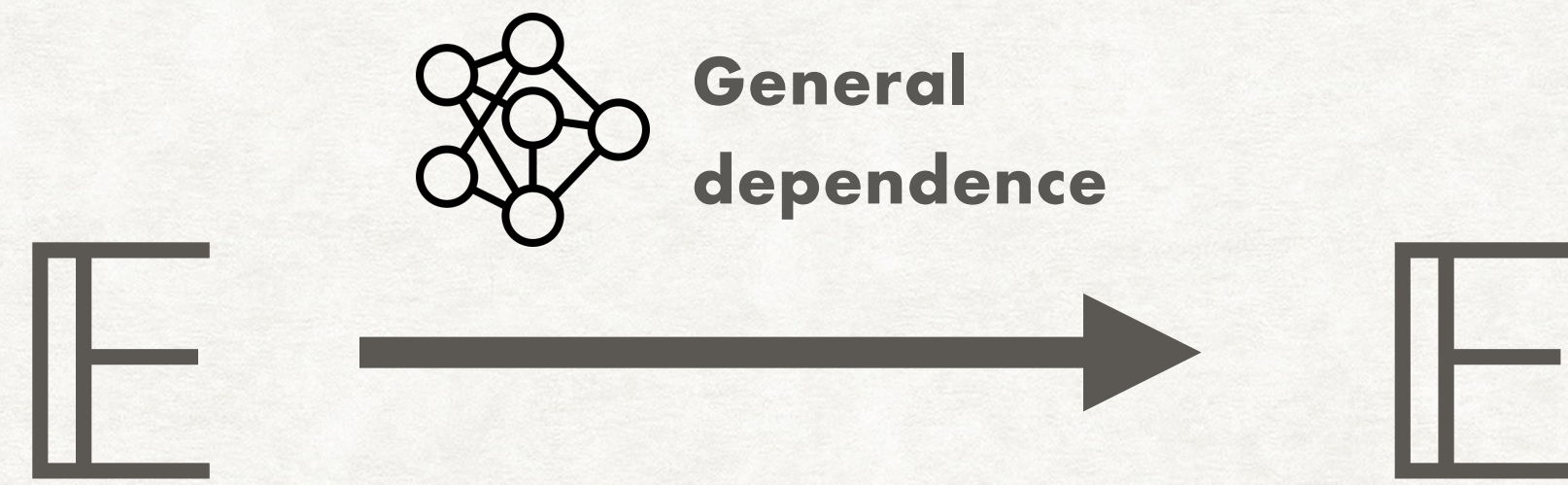
ORBS



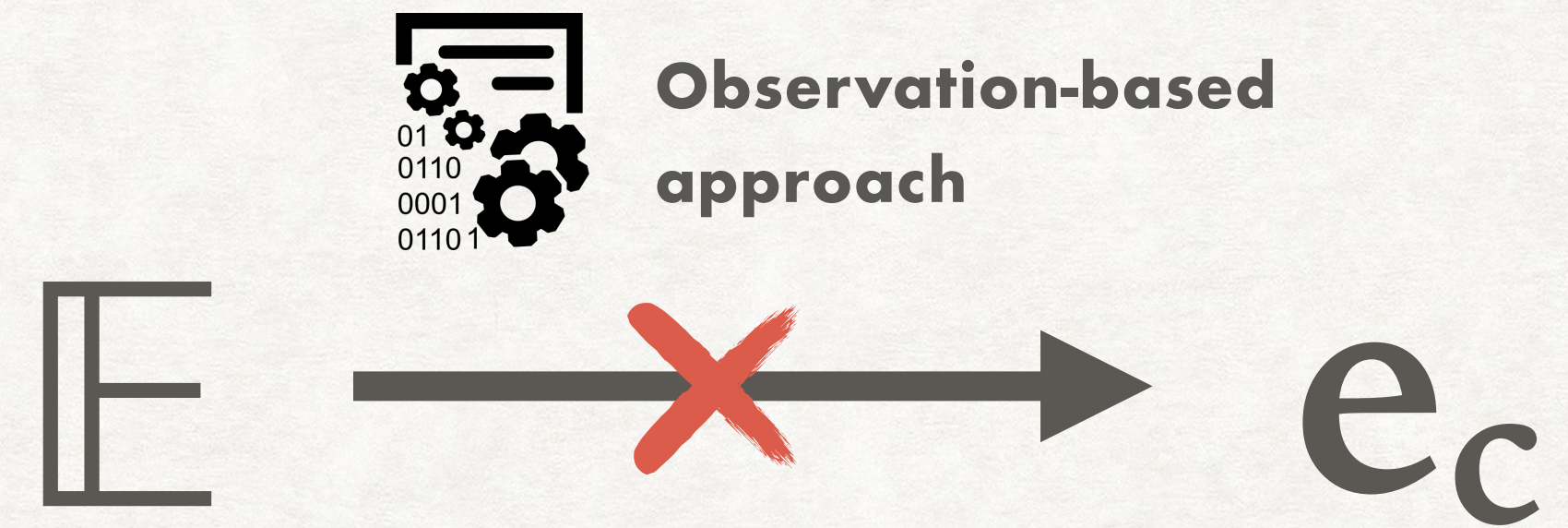
Observation-based analysis



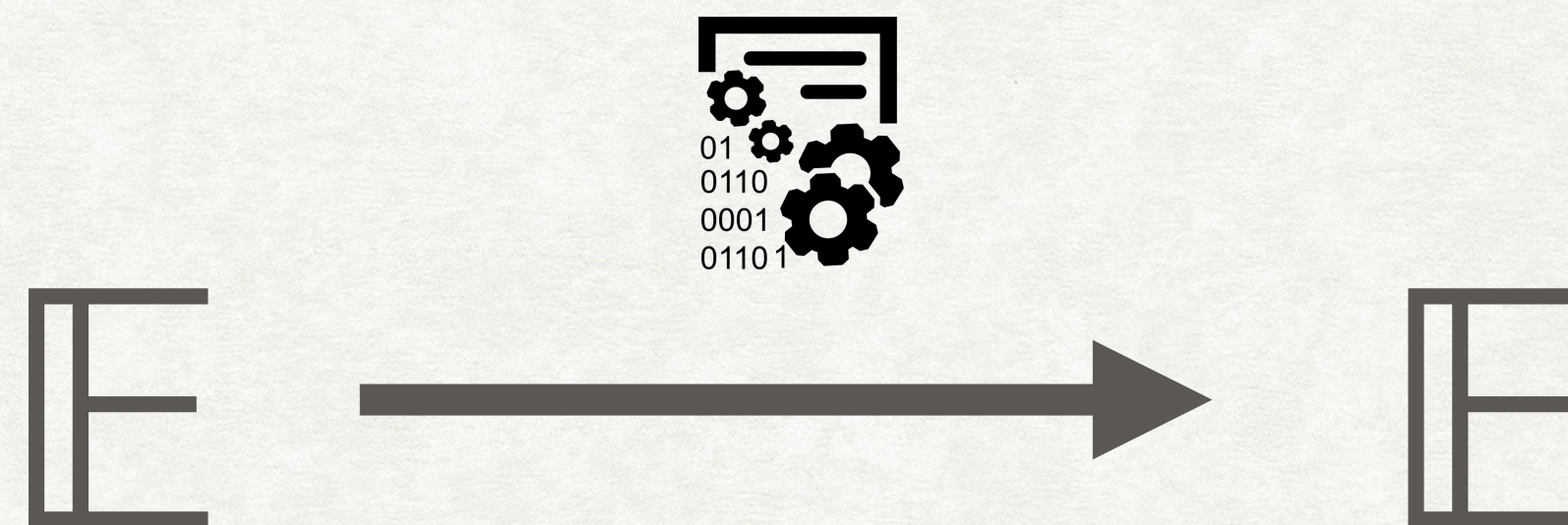
STATIC ANALYSIS



ORBS

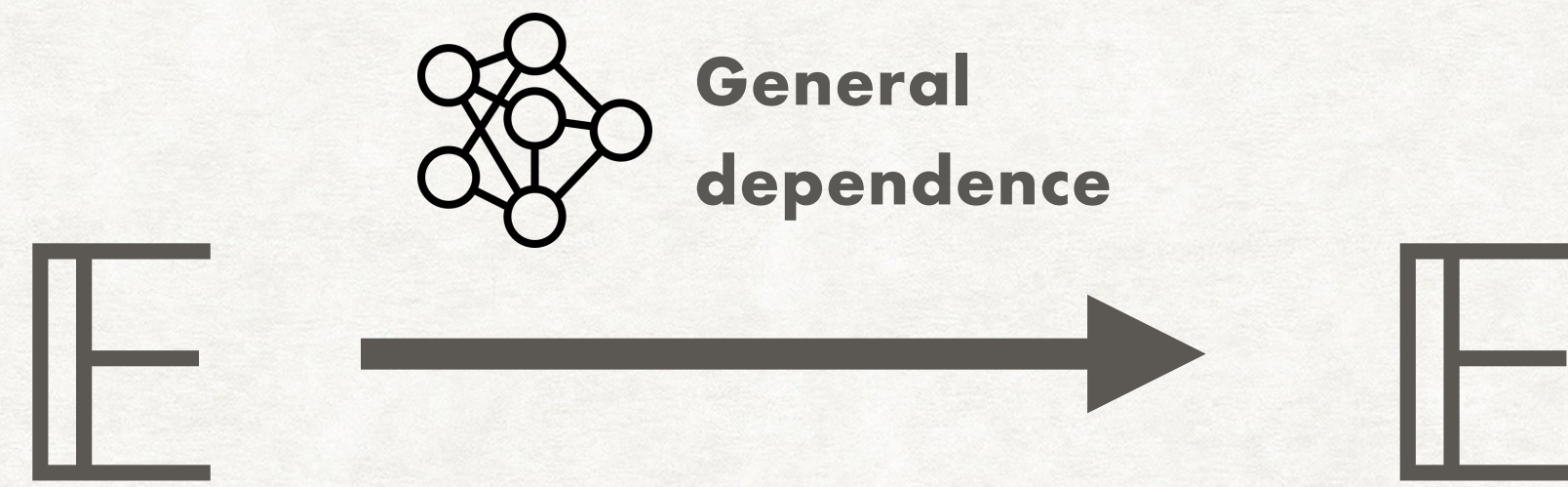


Observation-based analysis

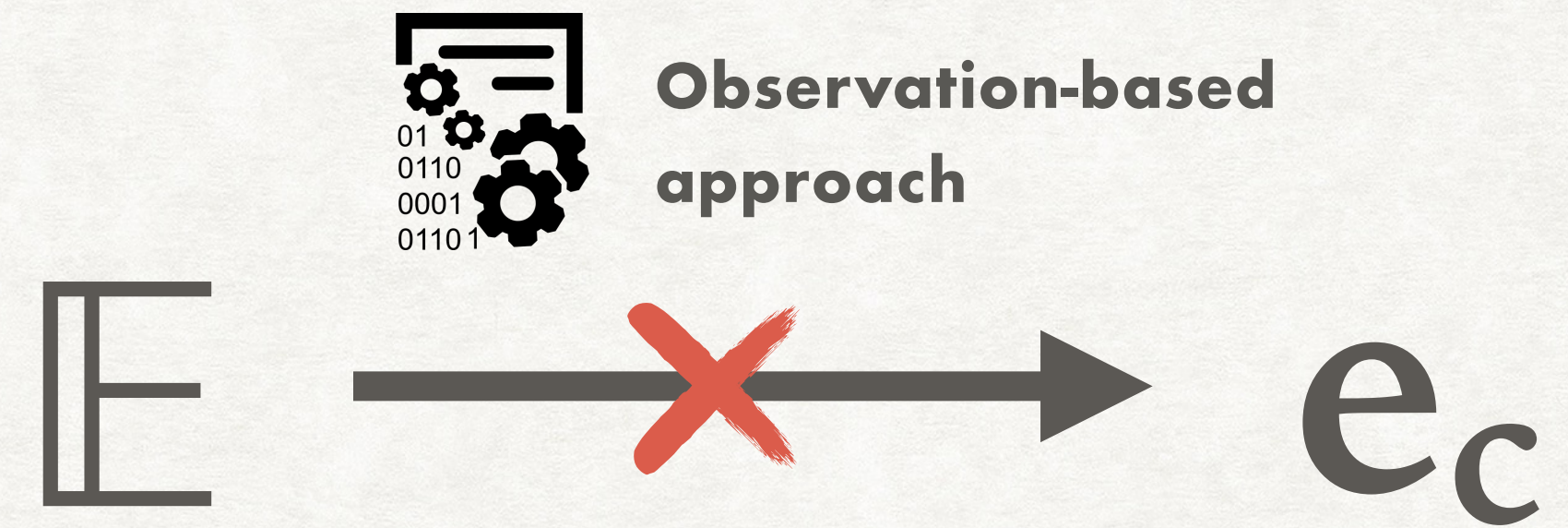


Modeling dependency

STATIC ANALYSIS



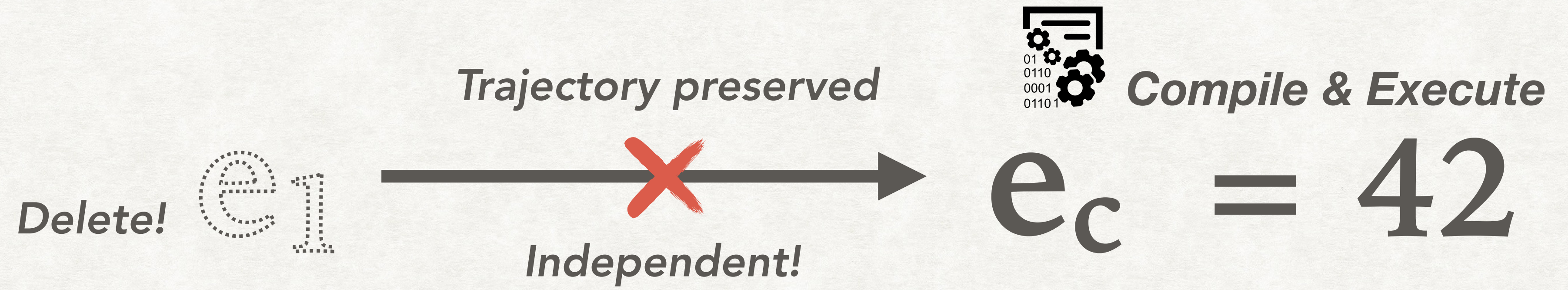
ORBS



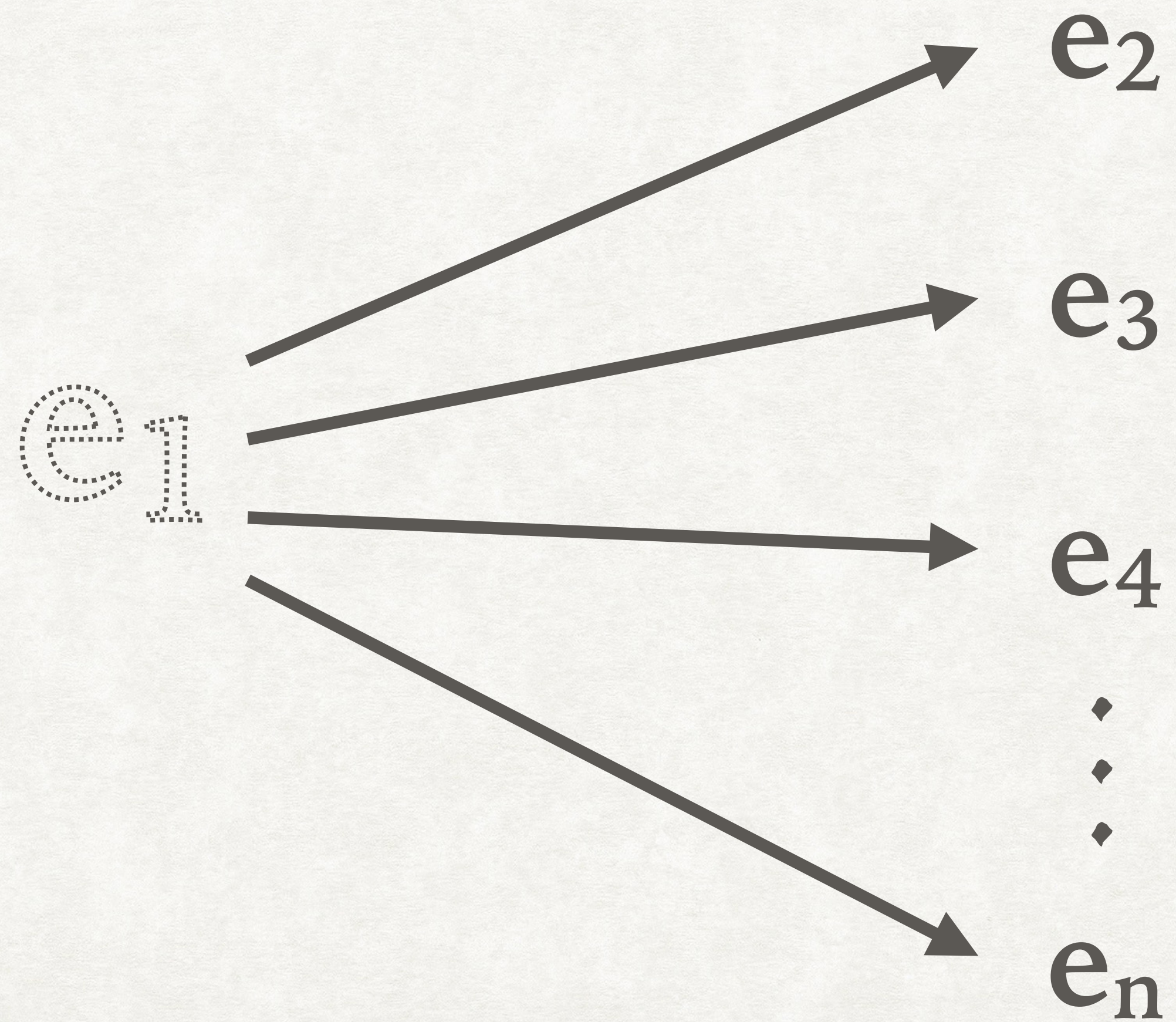
MOAD



MOAD



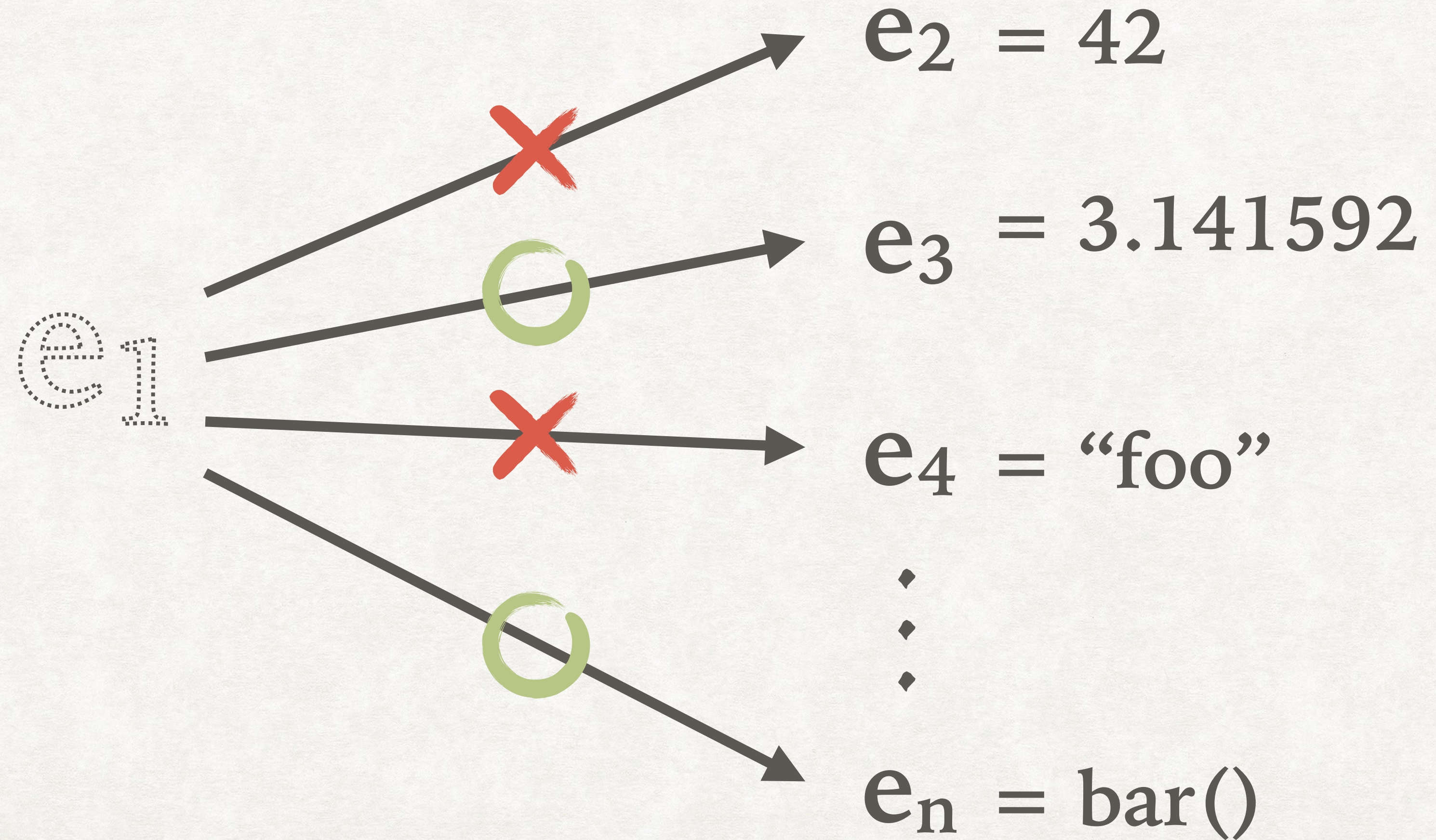
MOAD



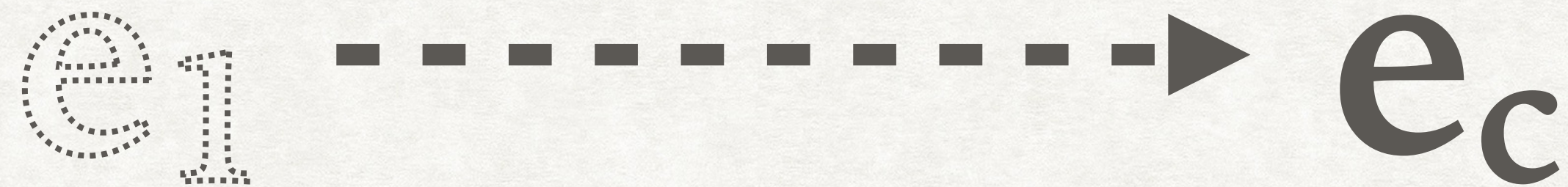
MOAD



Compile & Execute



MOAD



e_1 : ~~foo = Foo()~~

e_c : answer = 42

MOAD



e1: ~~foo = Foo()~~

ec: answer = 42



Traj(answer) = 42

MOAD



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

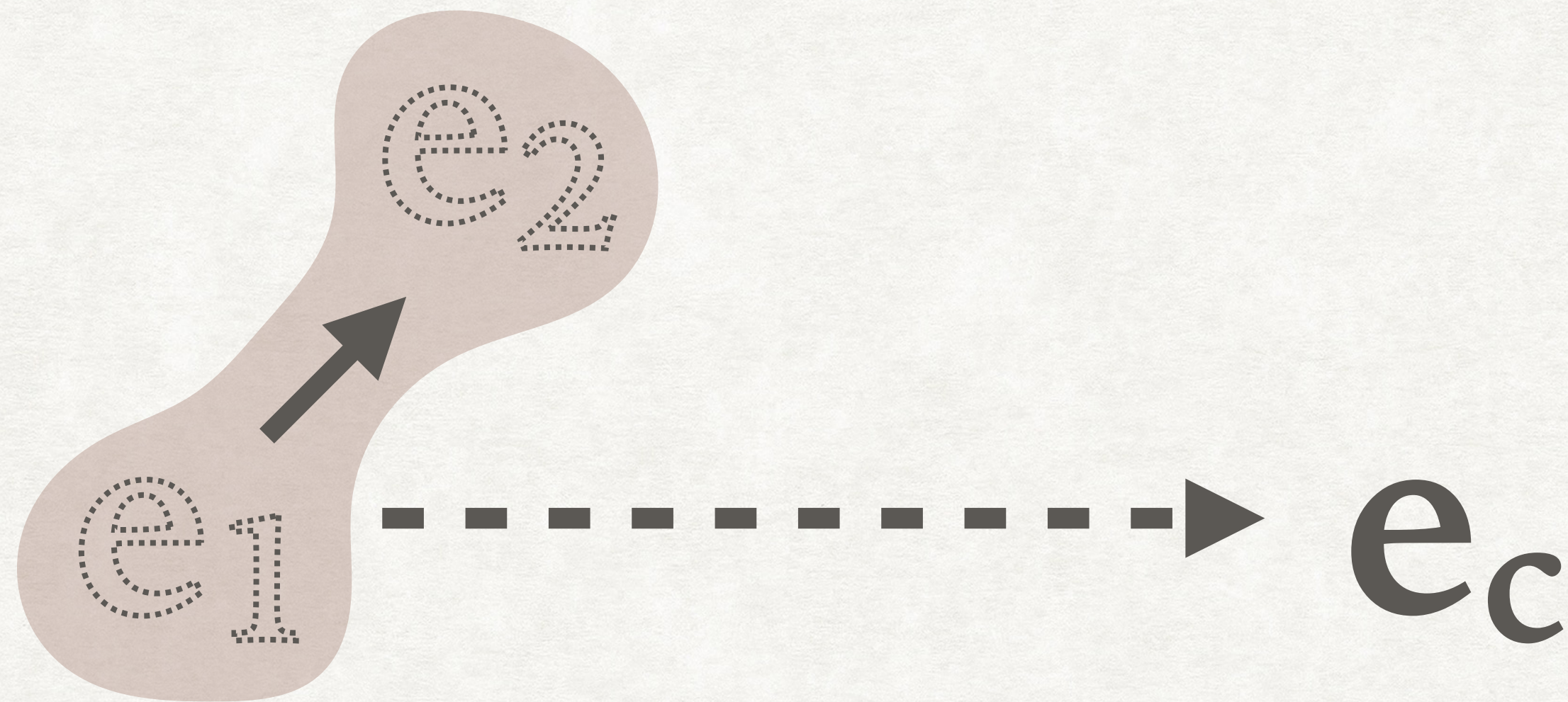
MOAD



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

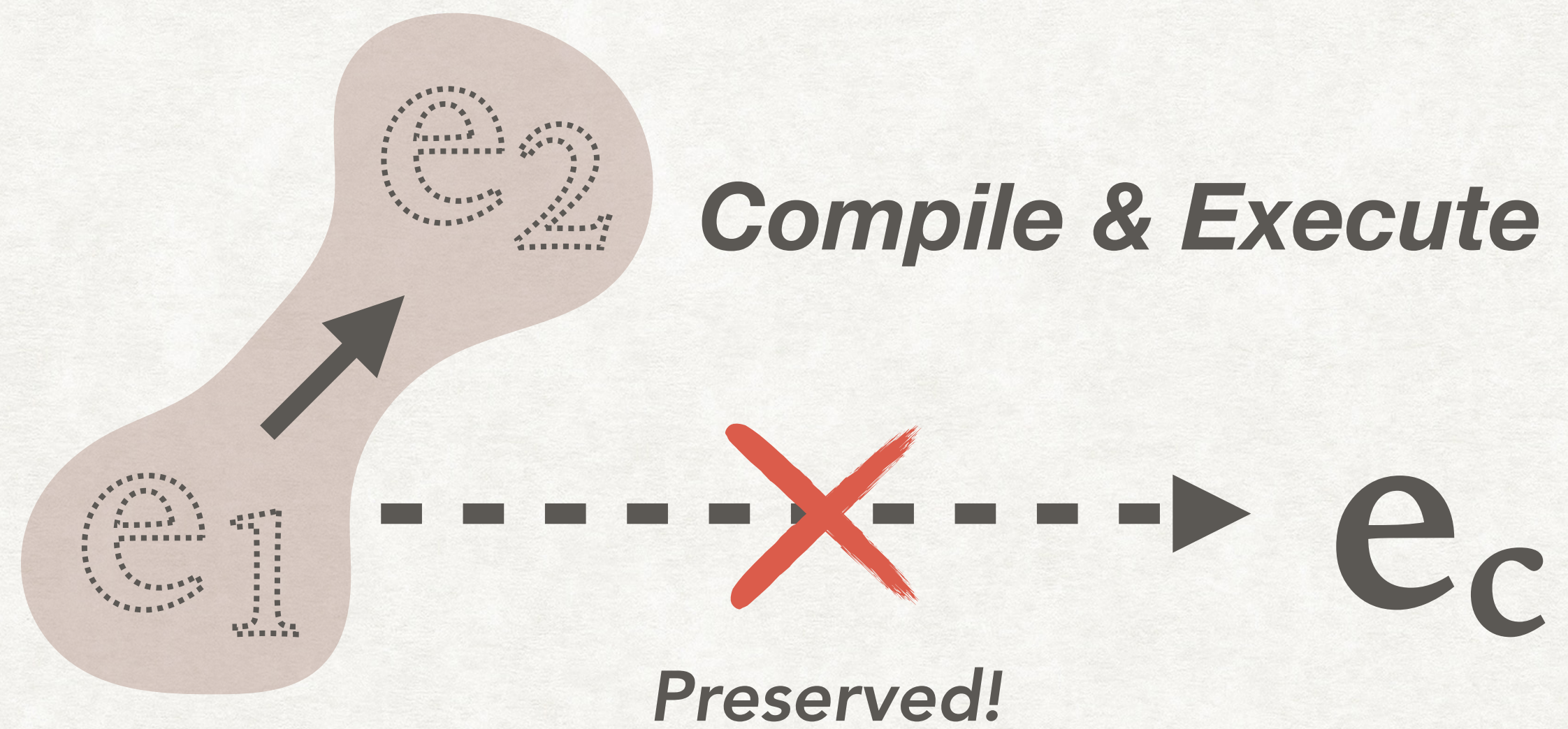
↓
 $Traj(answer) = \emptyset$

MOAD



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

MOAD

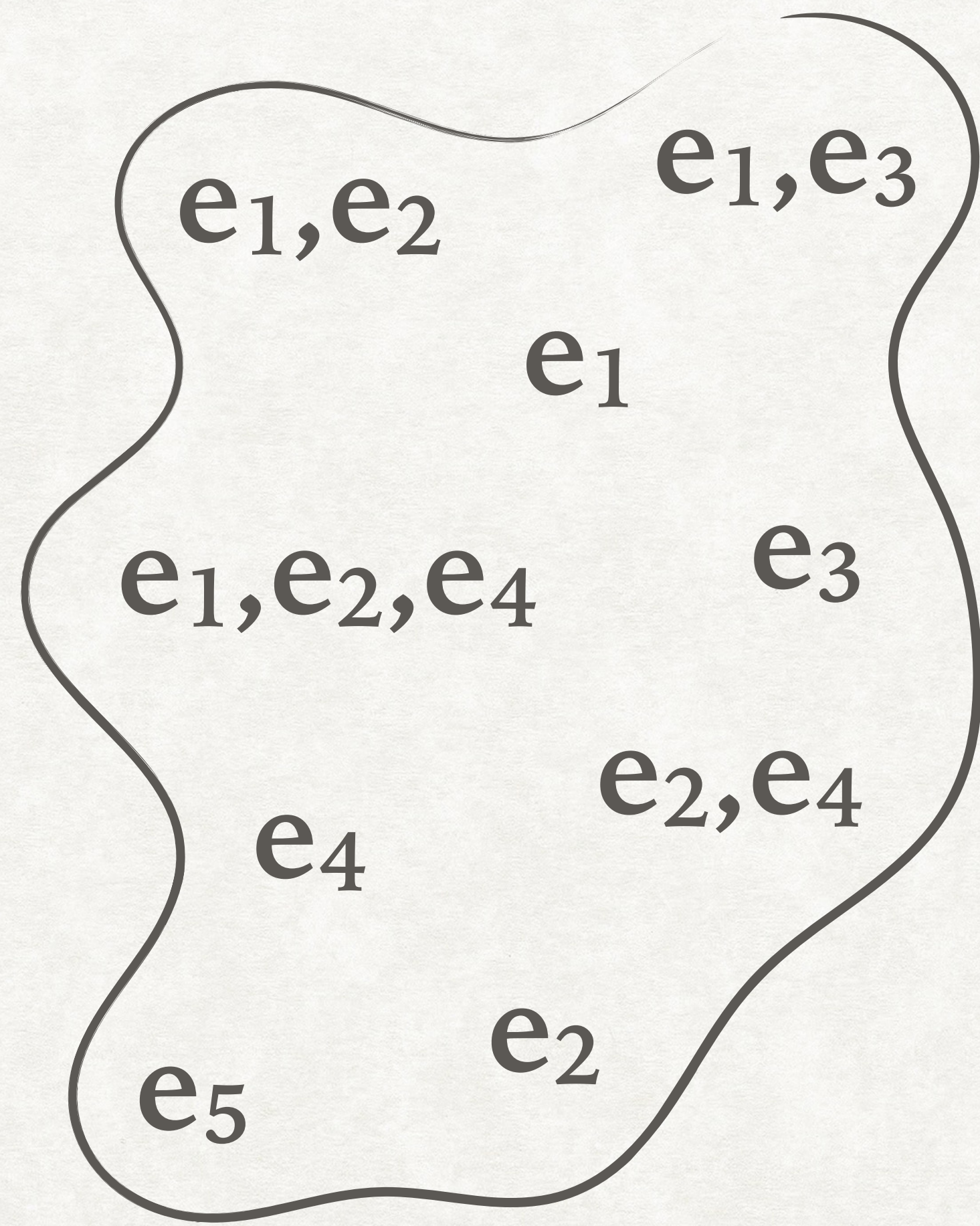


```
e1: foo = Foo()  
e2: foo.bar = 2  
ec: answer = 42
```

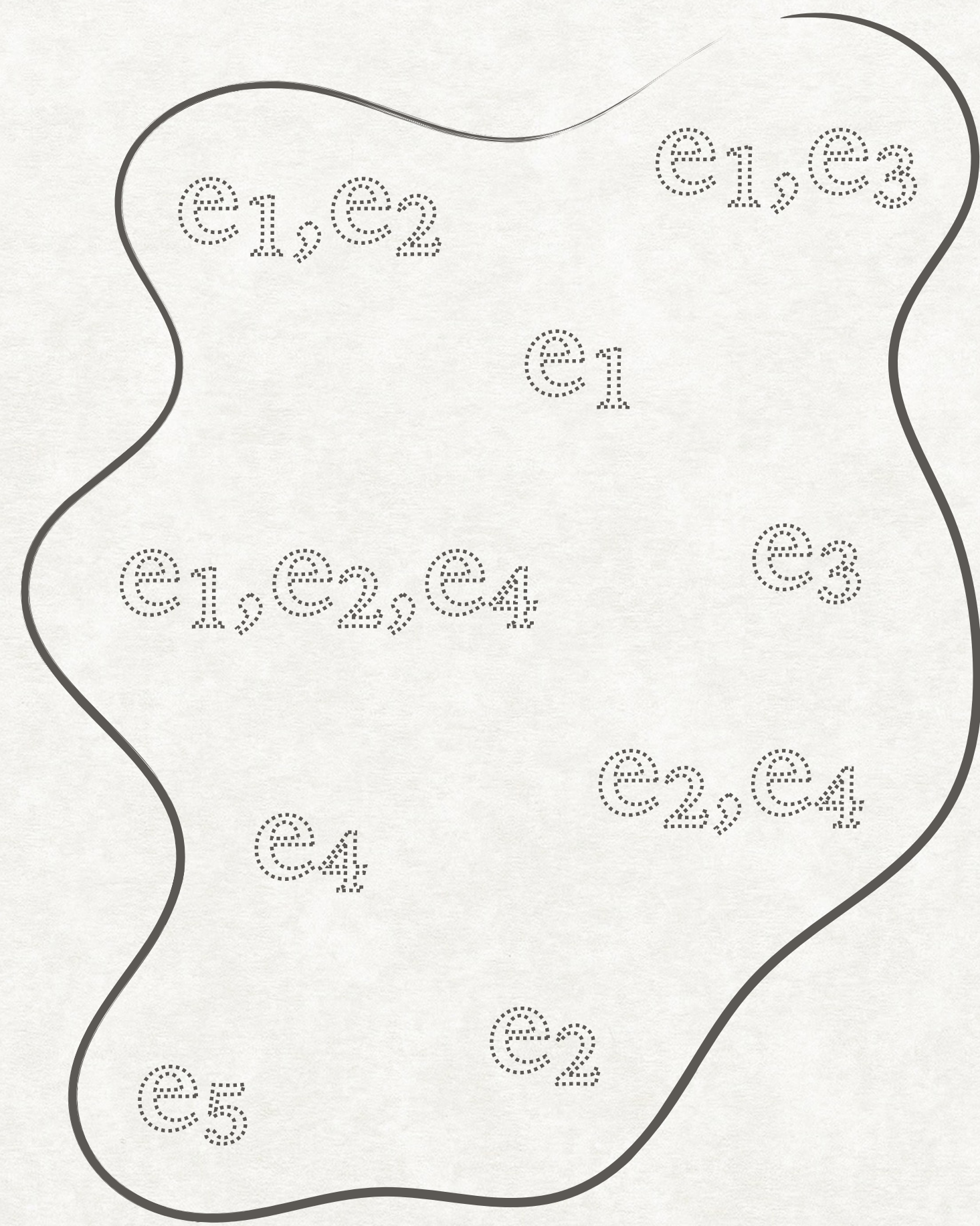
↓

$Traj(answer) = 42$

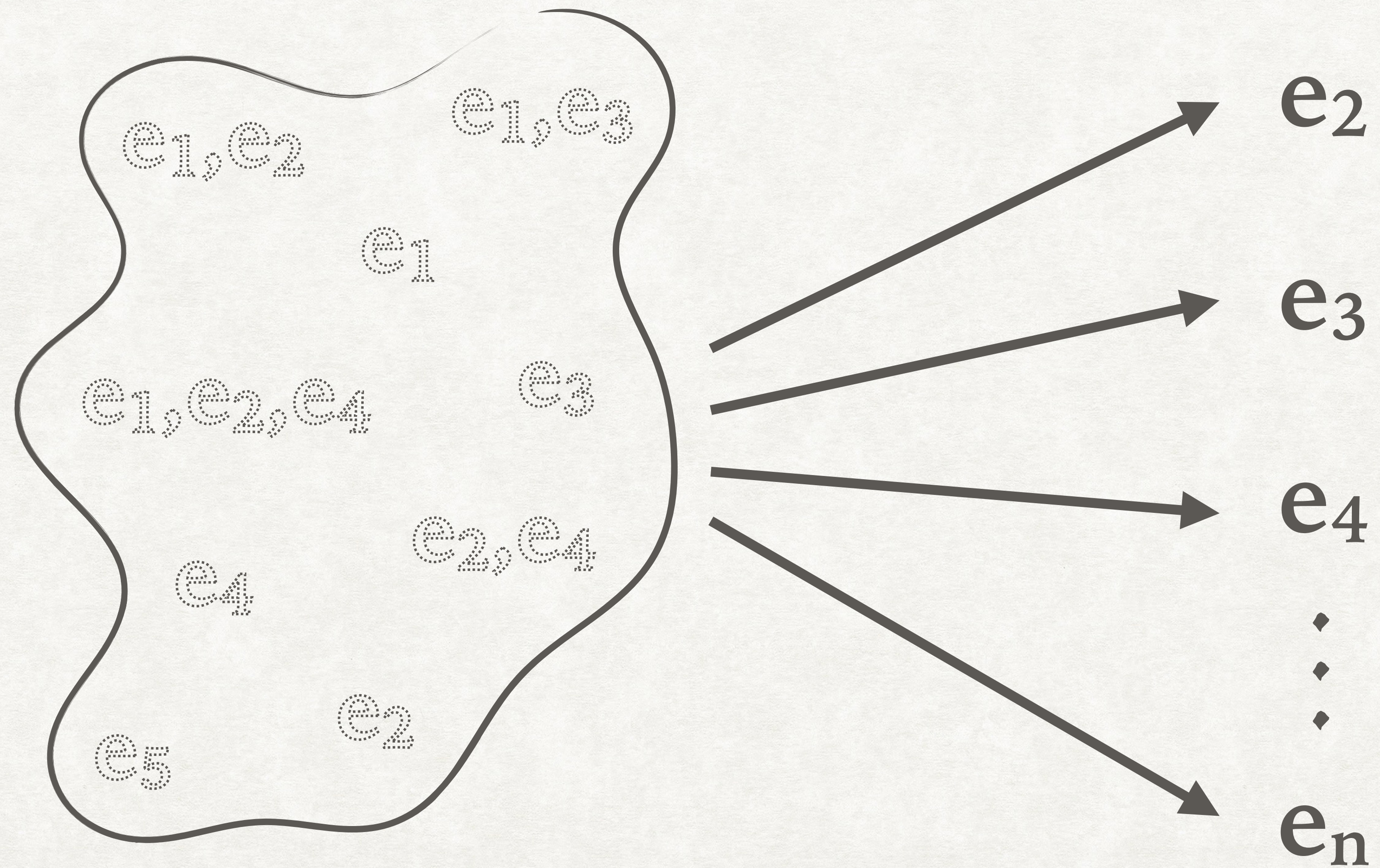
OBSERVATION PHASE



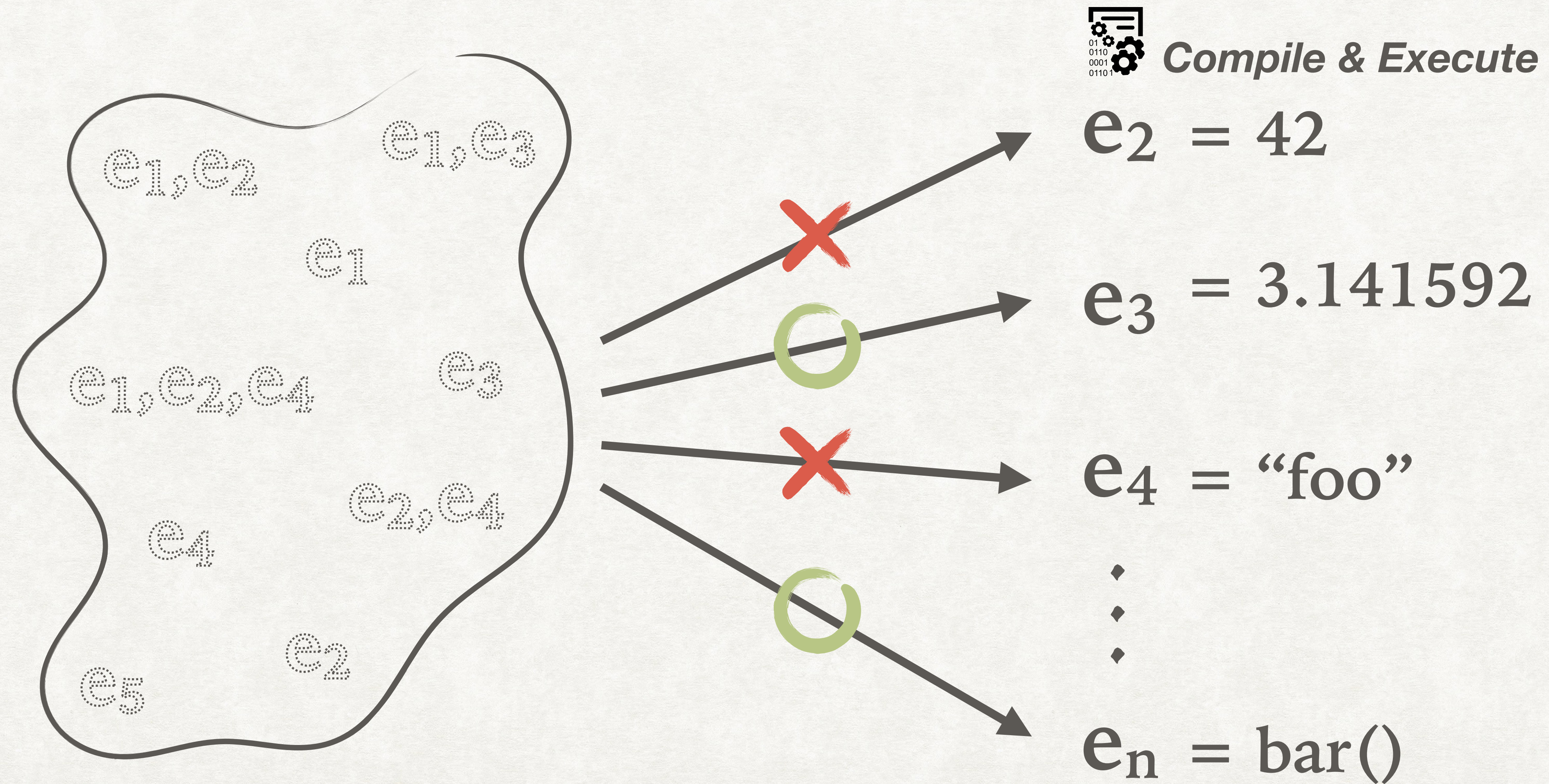
OBSERVATION PHASE



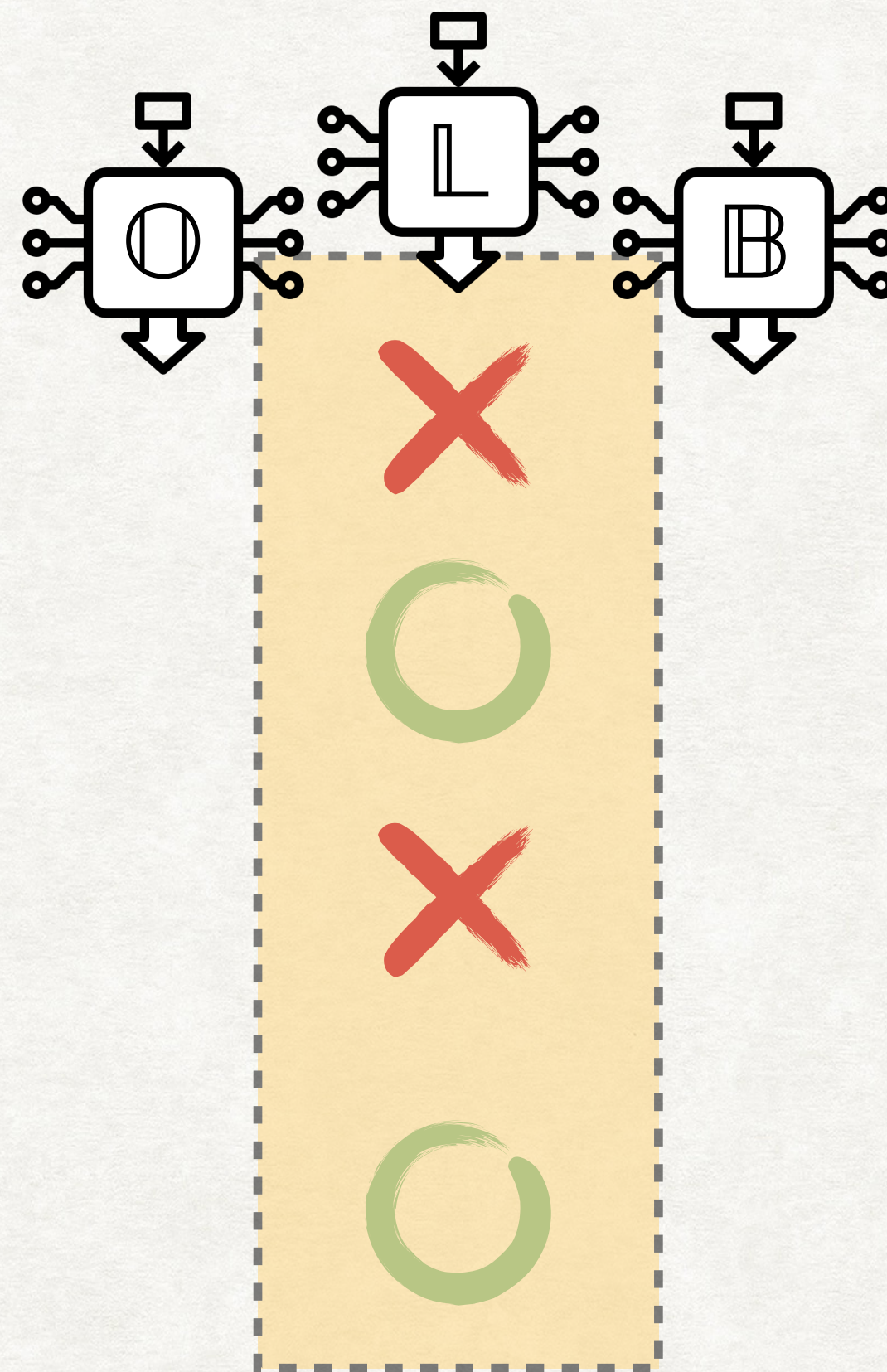
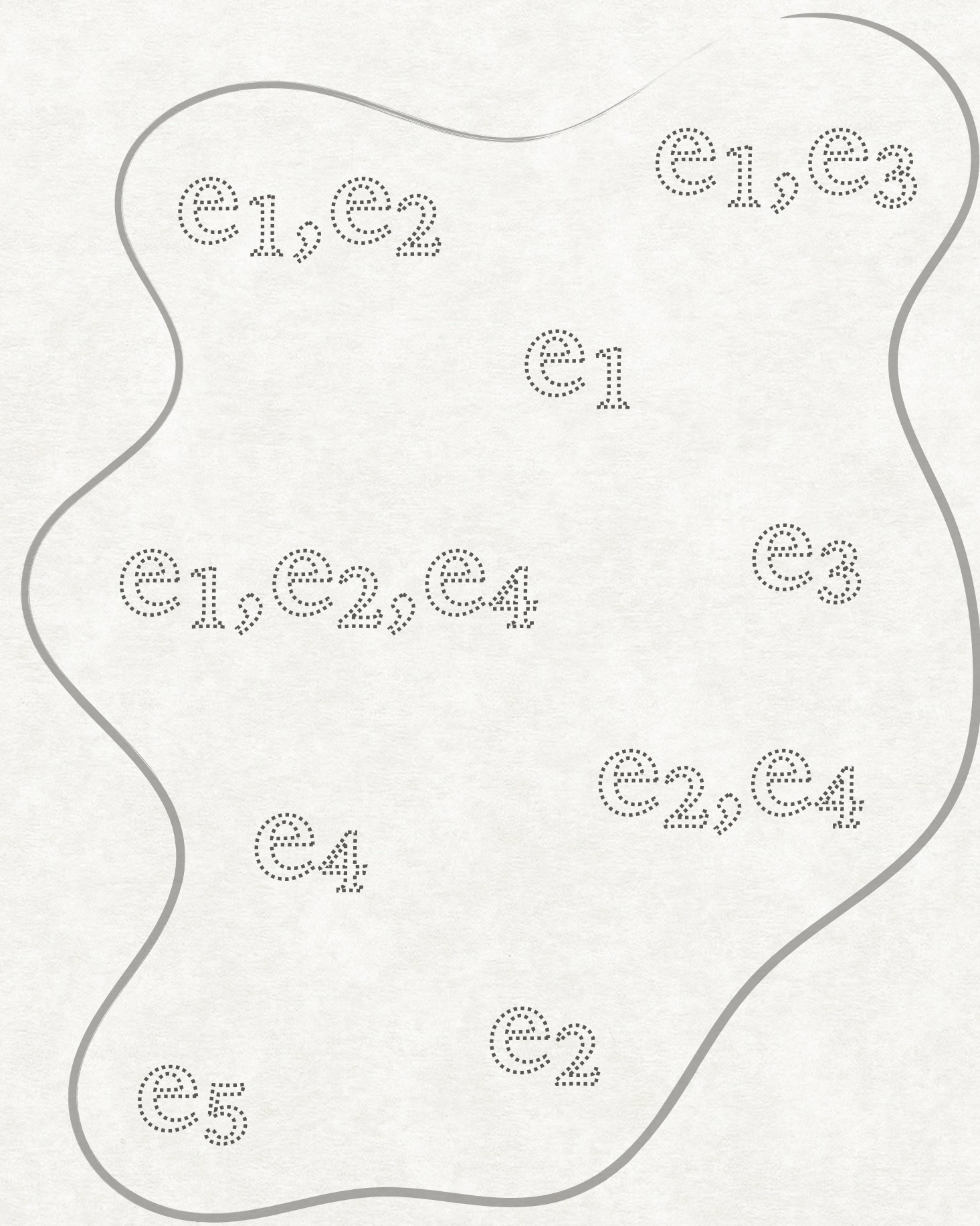
OBSERVATION PHASE



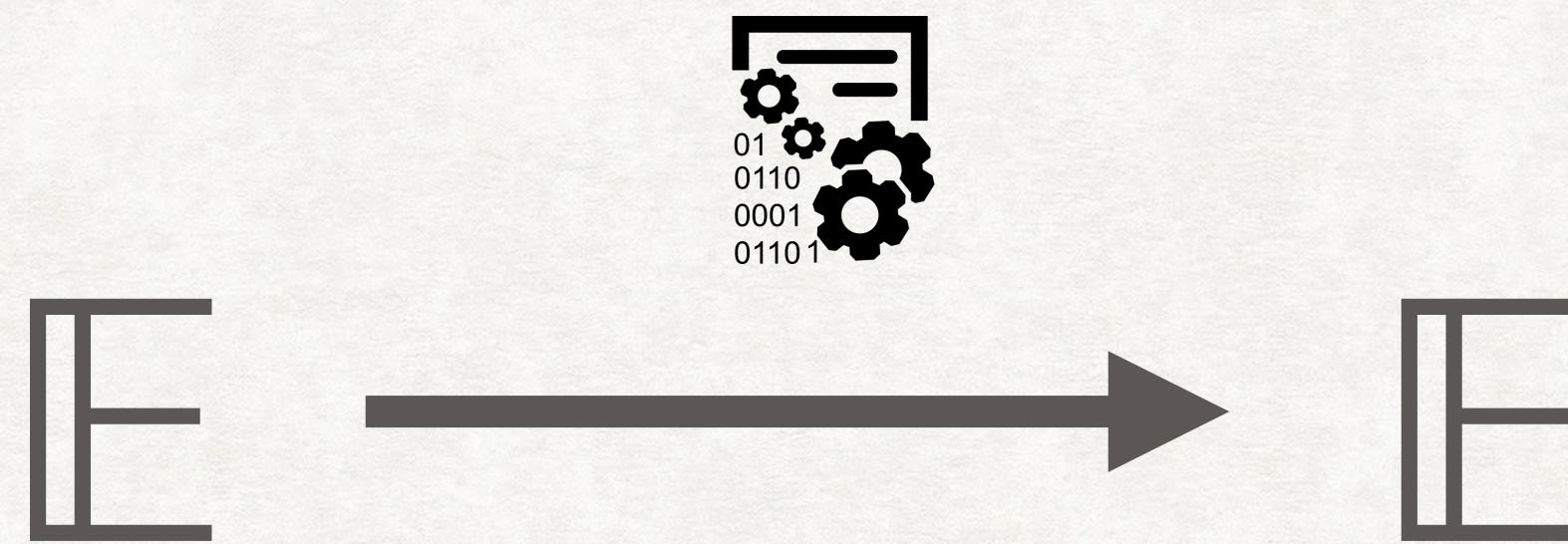
OBSERVATION PHASE



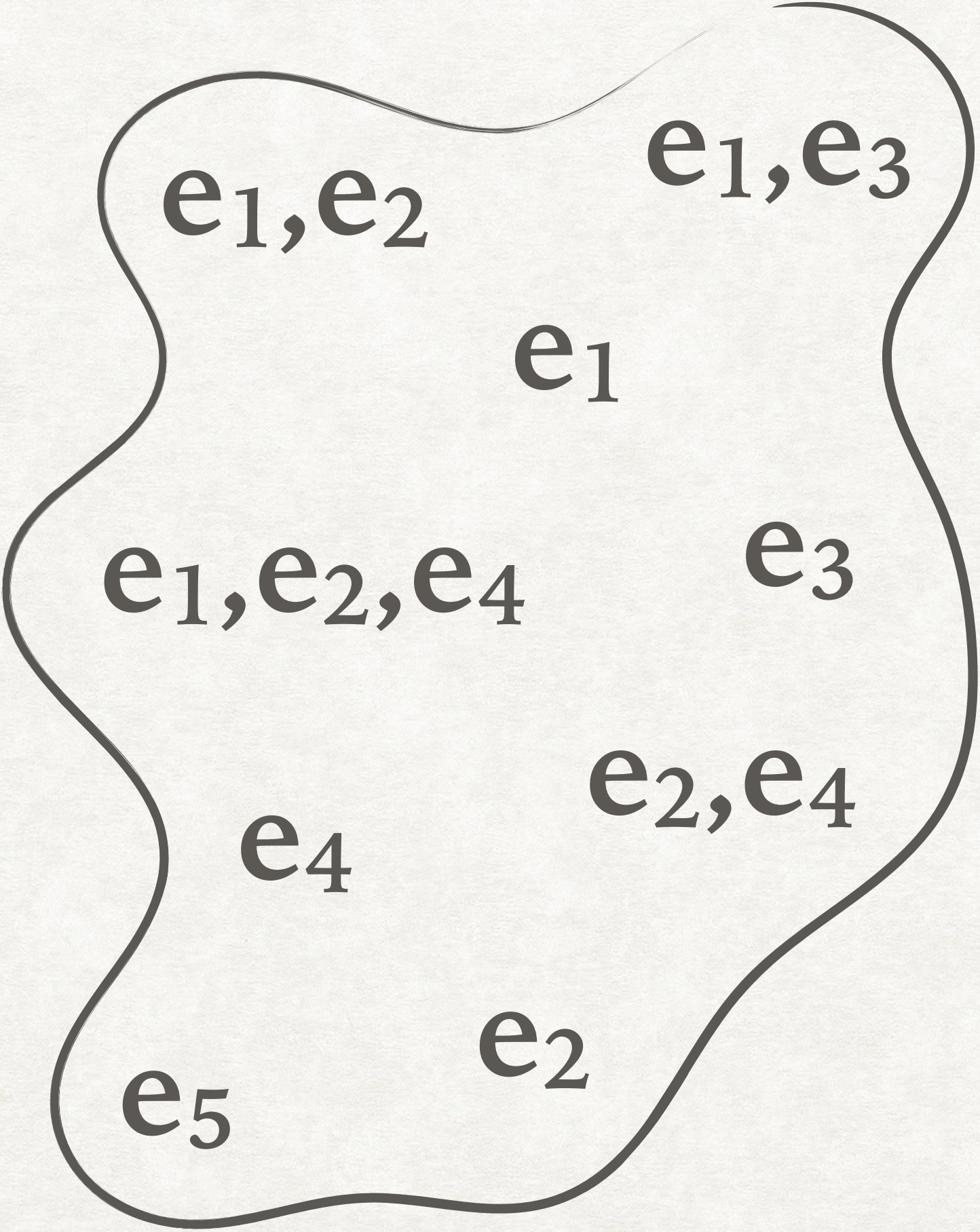
INFERENCE PHASE



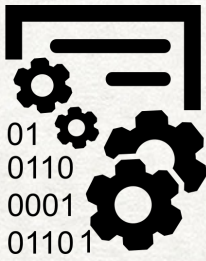
MOAD



MOAD

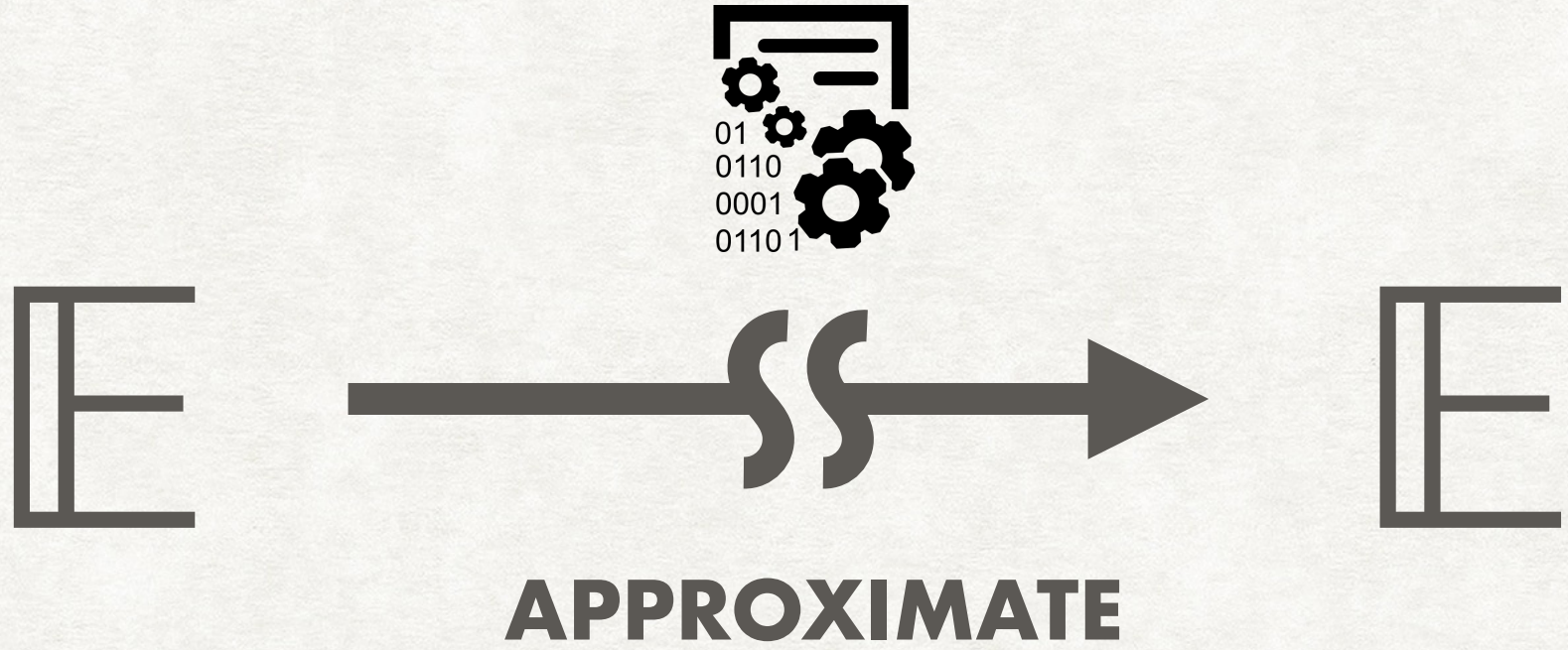
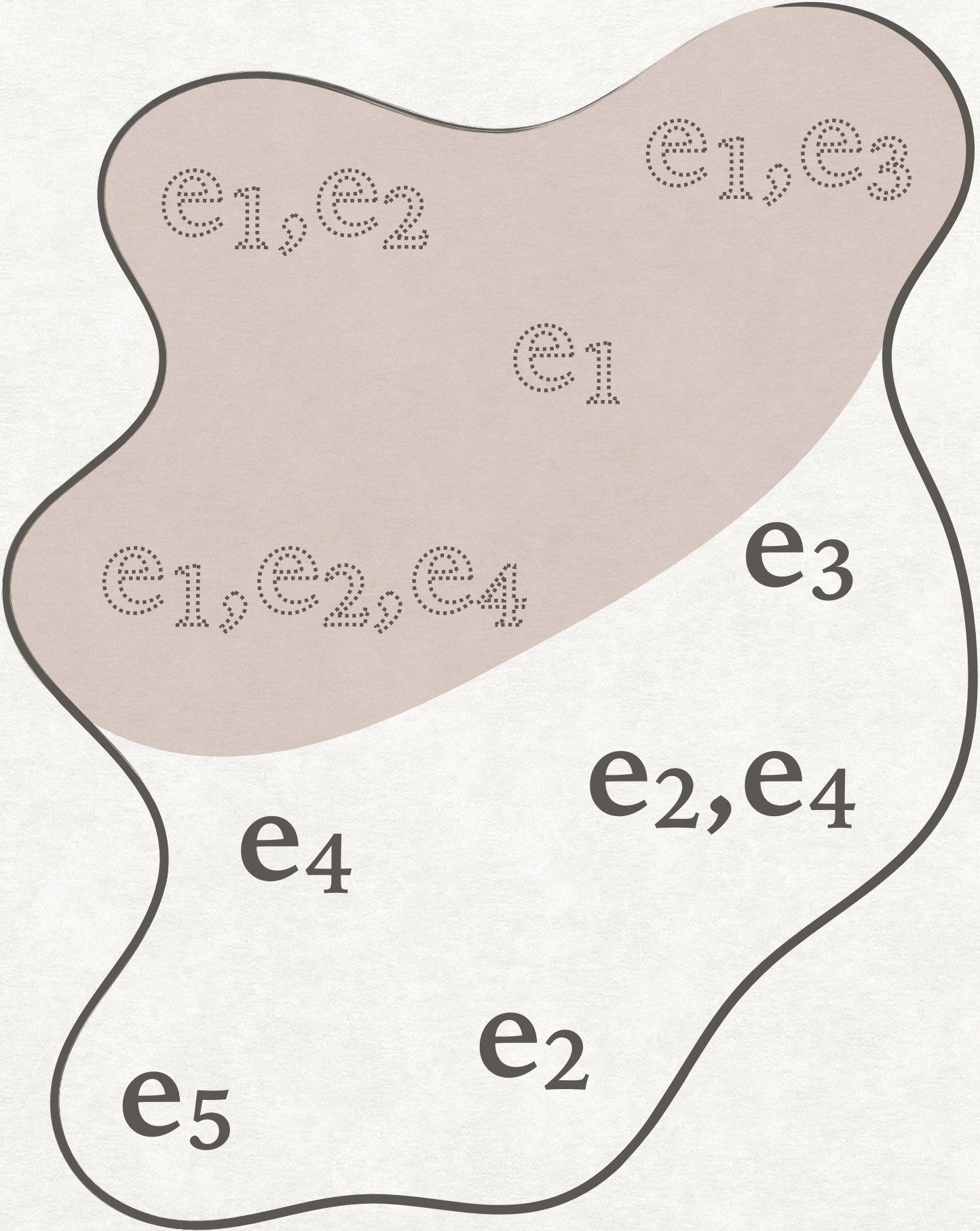


\mathbb{E}



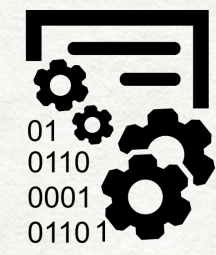
\mathbb{E}

MOAD



ADVANTAGE OF MOAD VS. ORBS

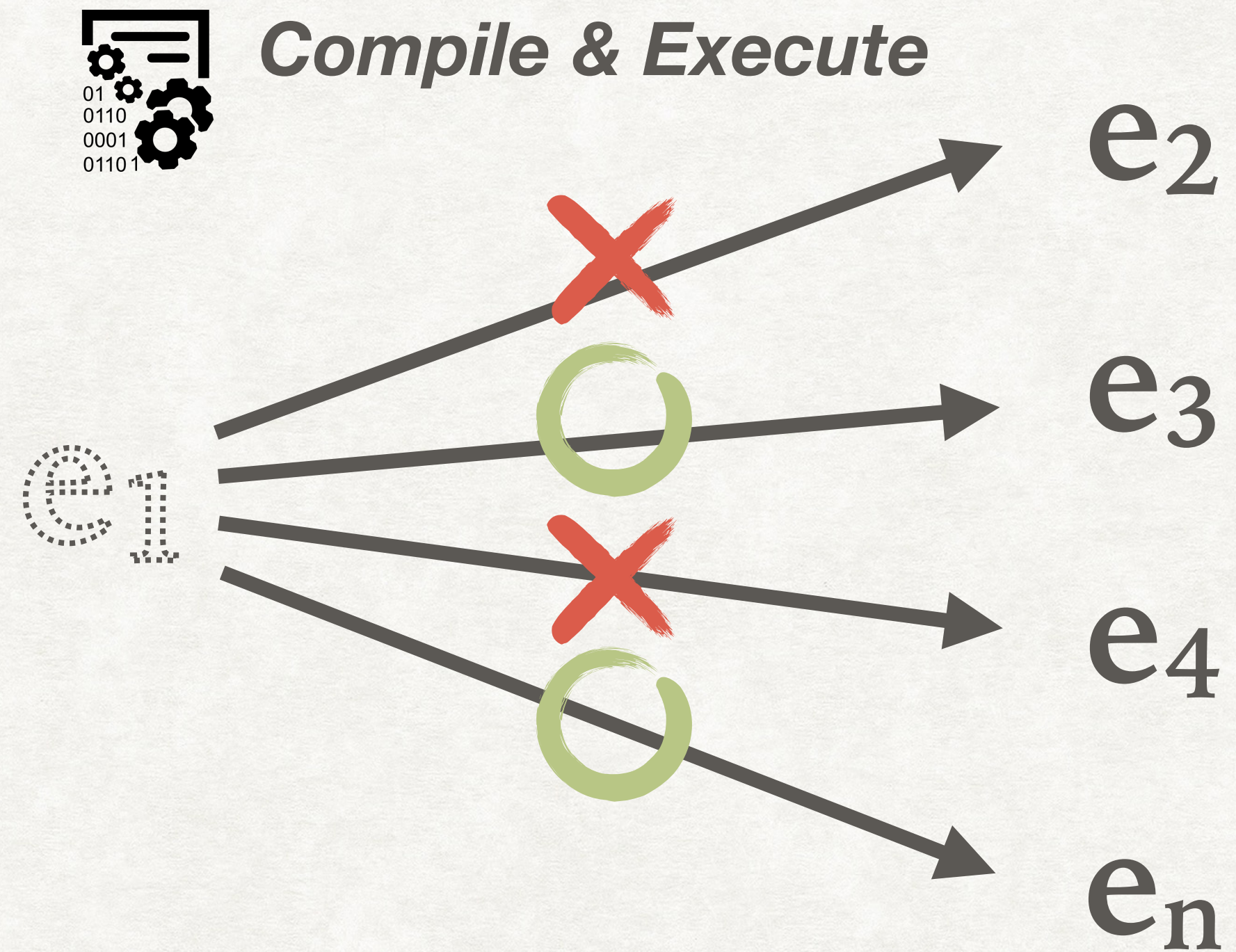
ADVANTAGE OF MOAD VS. ORBS



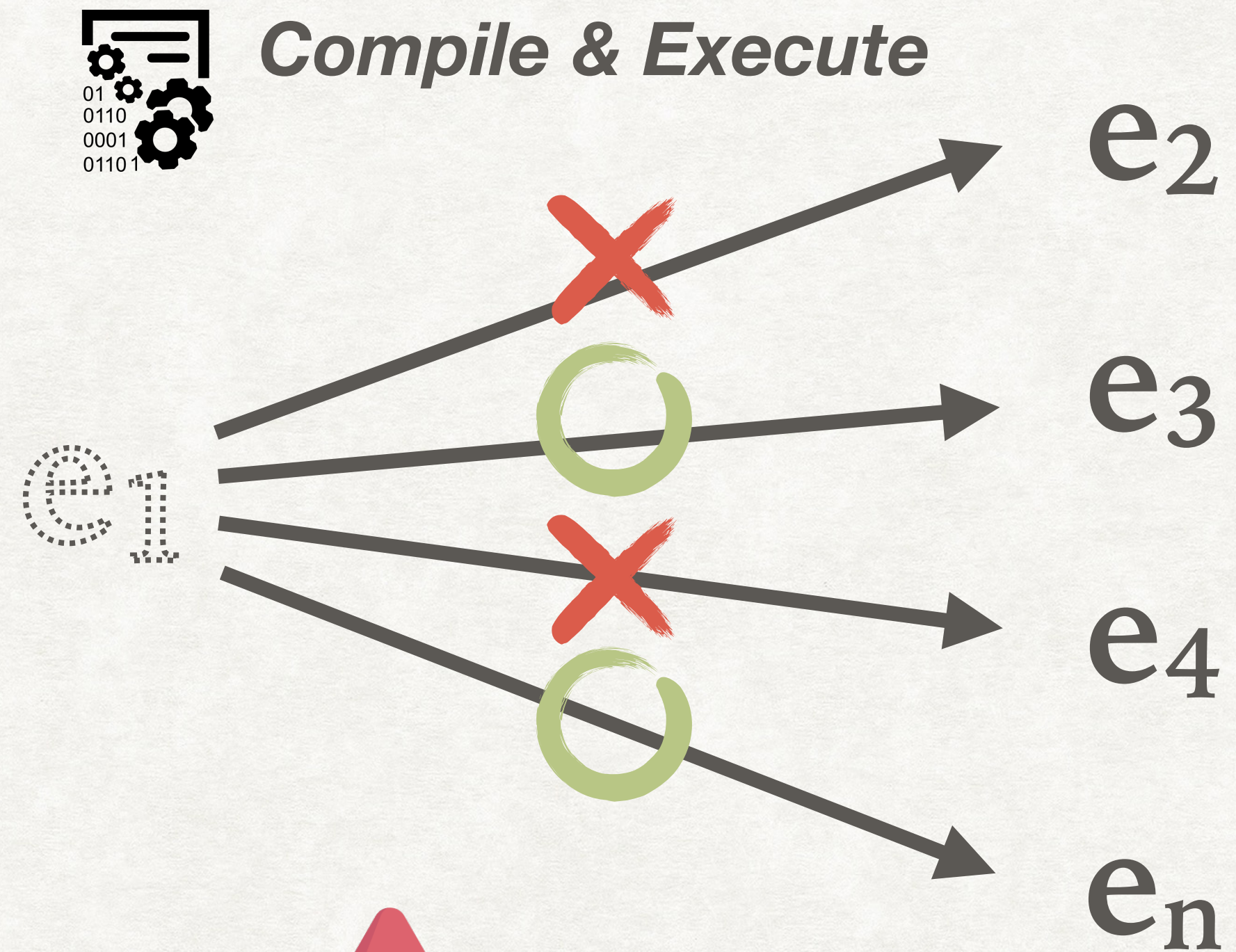
Compile & Execute



ADVANTAGE OF MOAD VS. ORBS

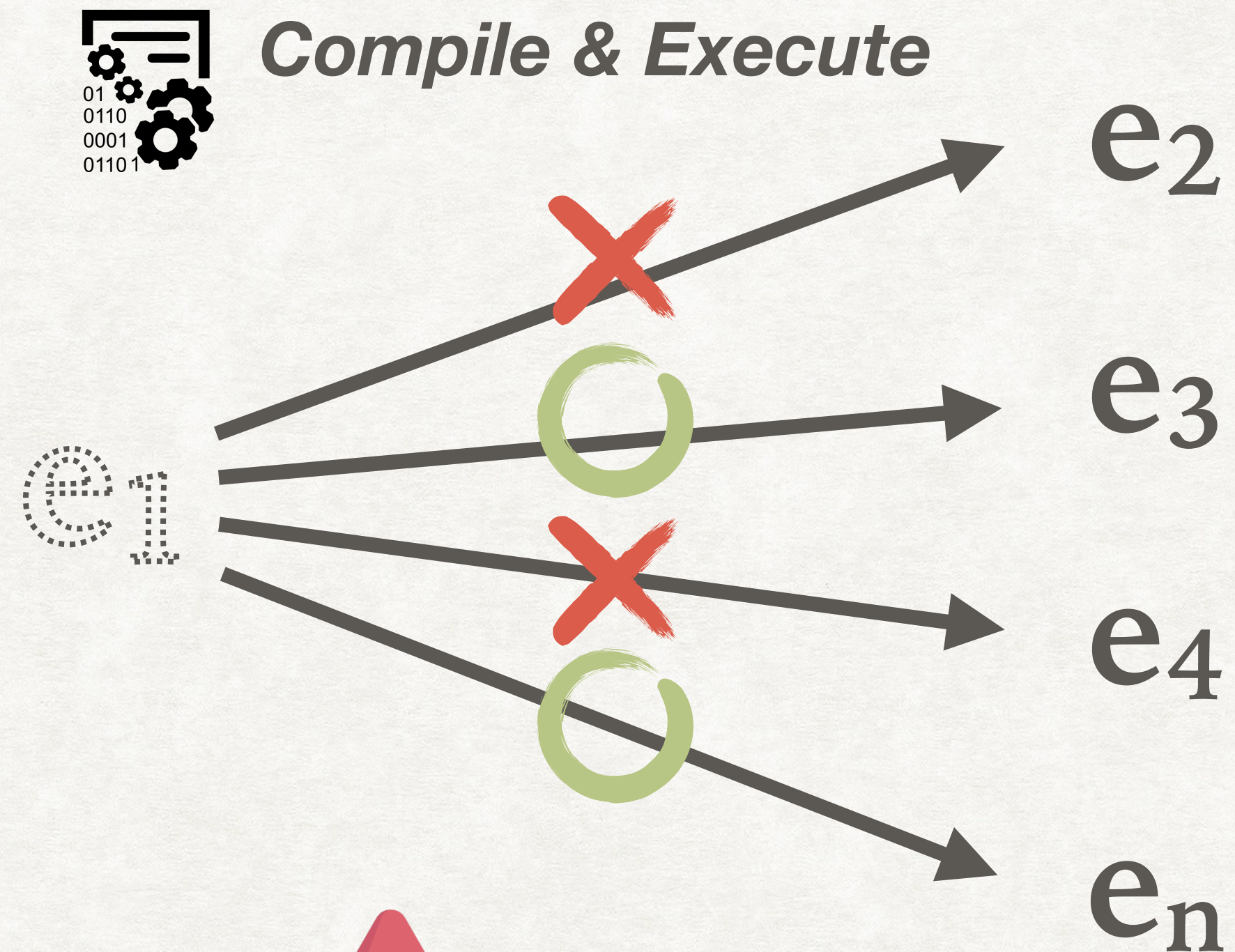


ADVANTAGE OF MOAD VS. ORBS

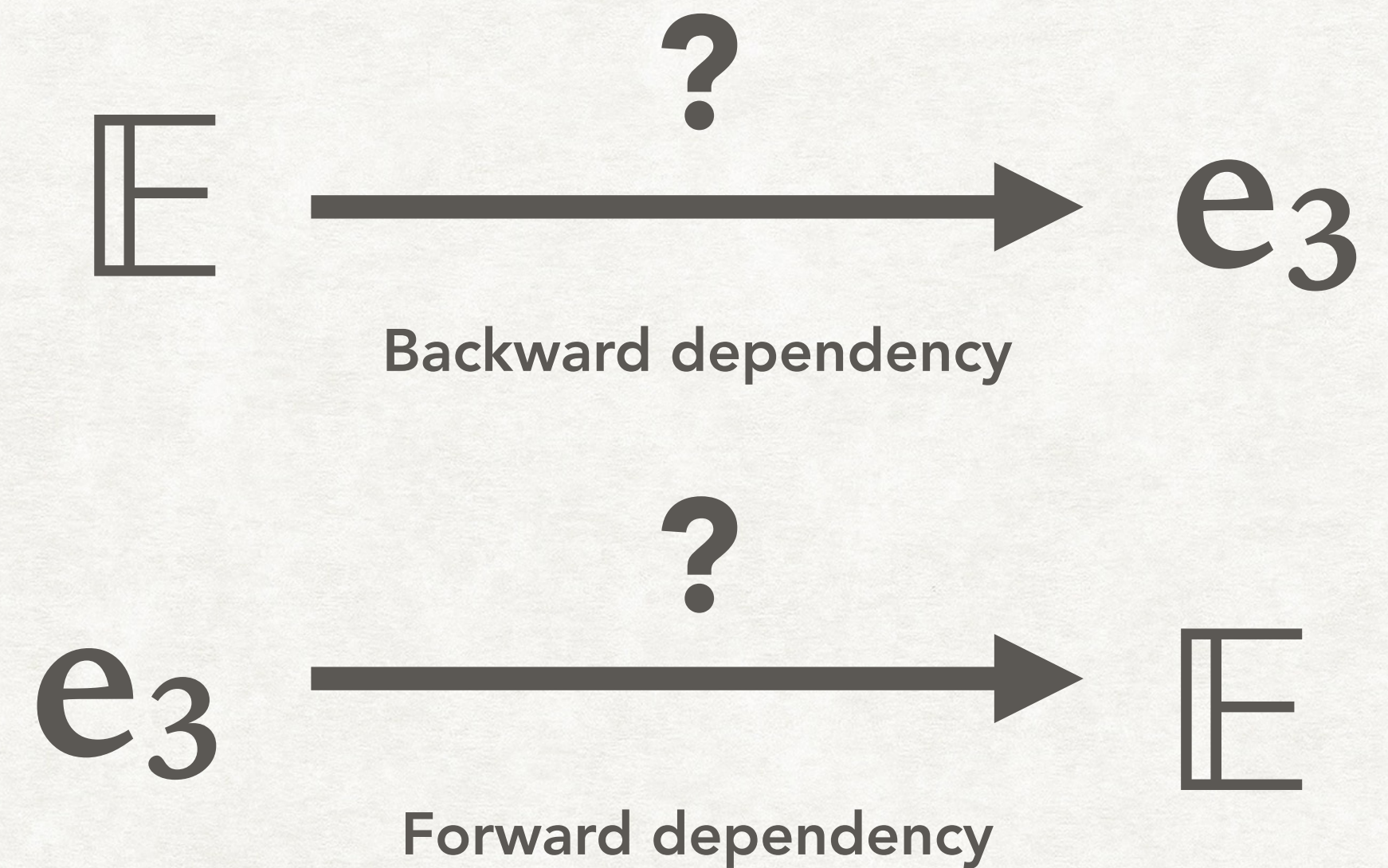


1. EFFICIENCY

ADVANTAGE OF MOAD VS. ORBS



1. EFFICIENCY

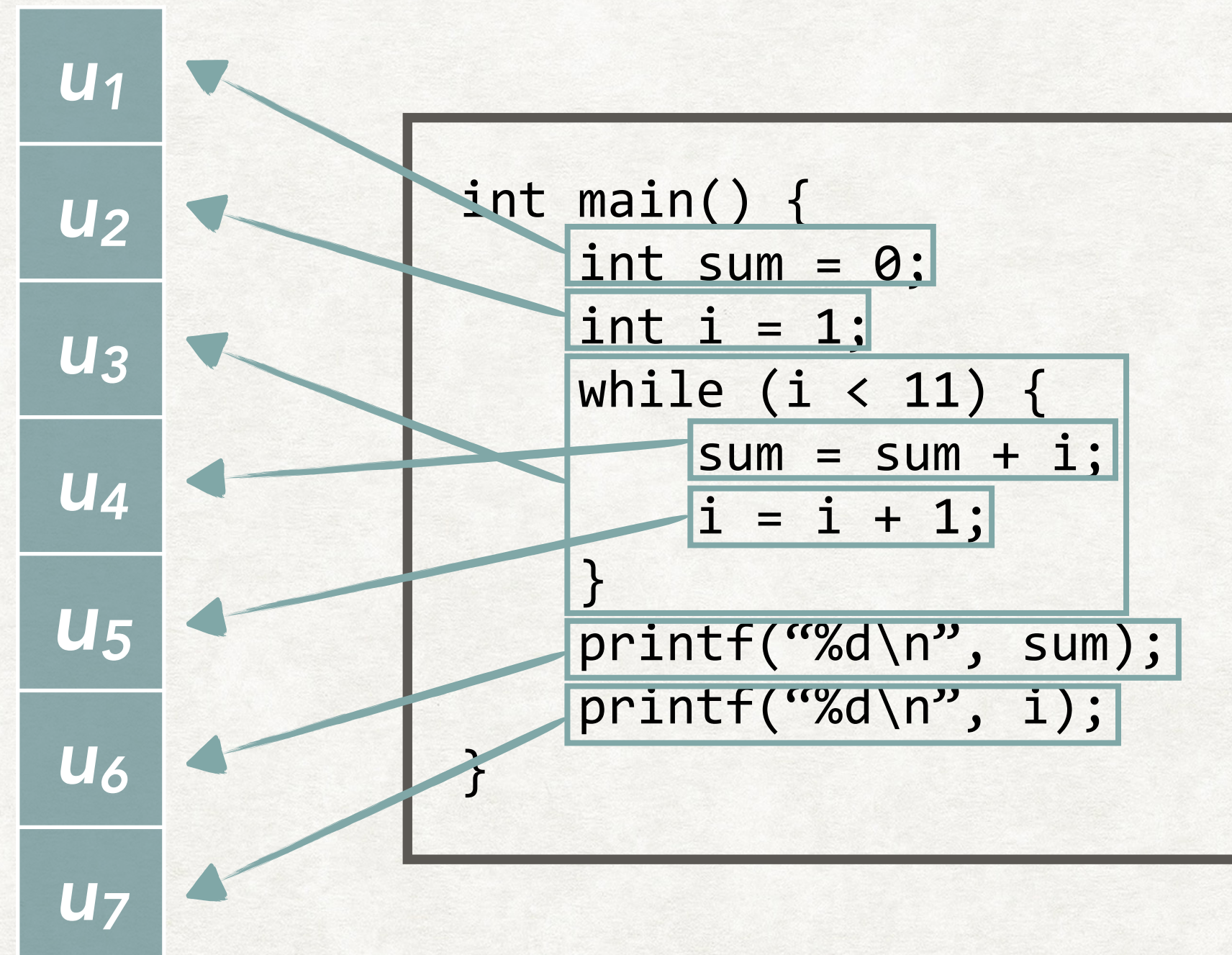


100

2. COMPLETENESS

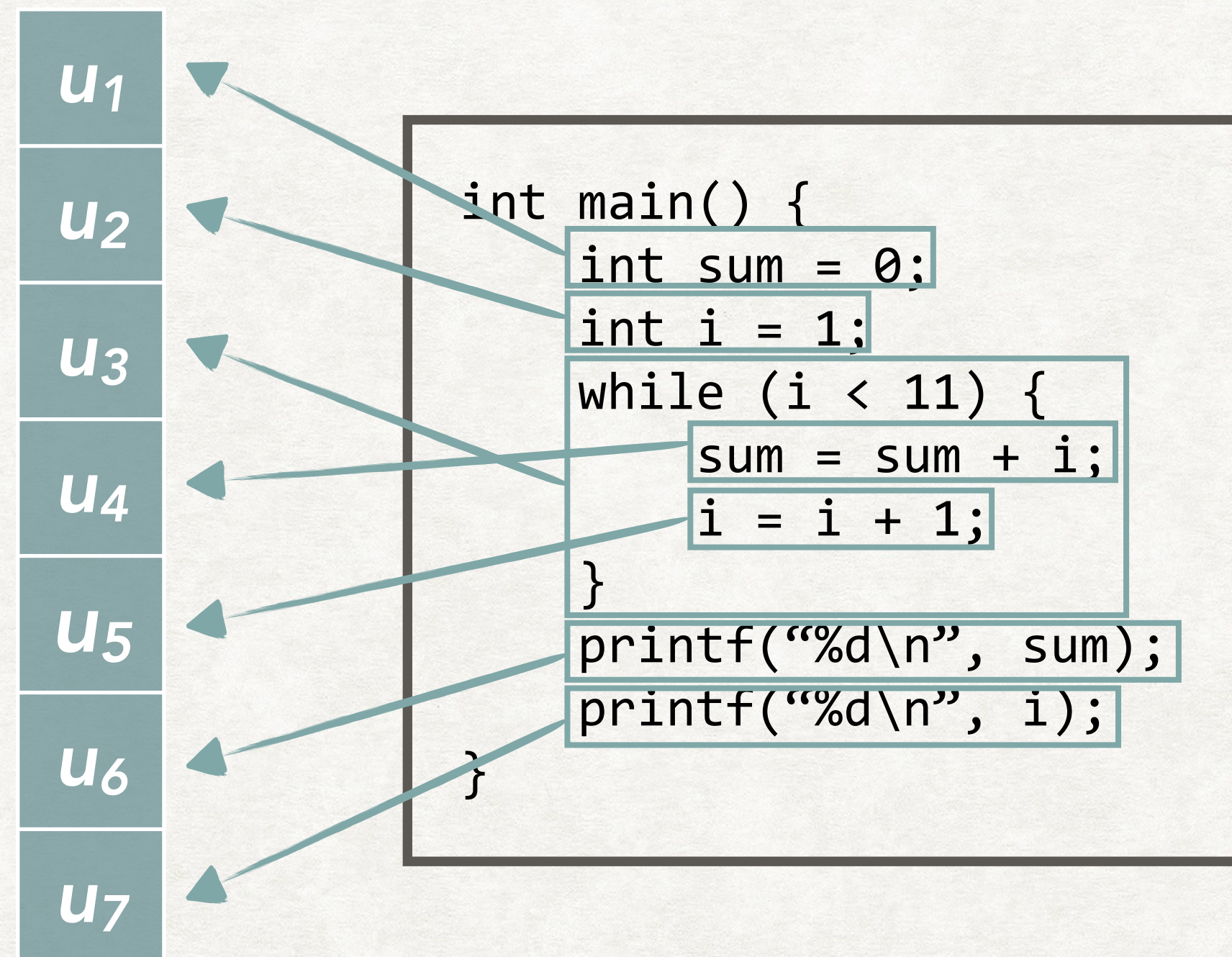
DELETION GENERATION SCHEME

- 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



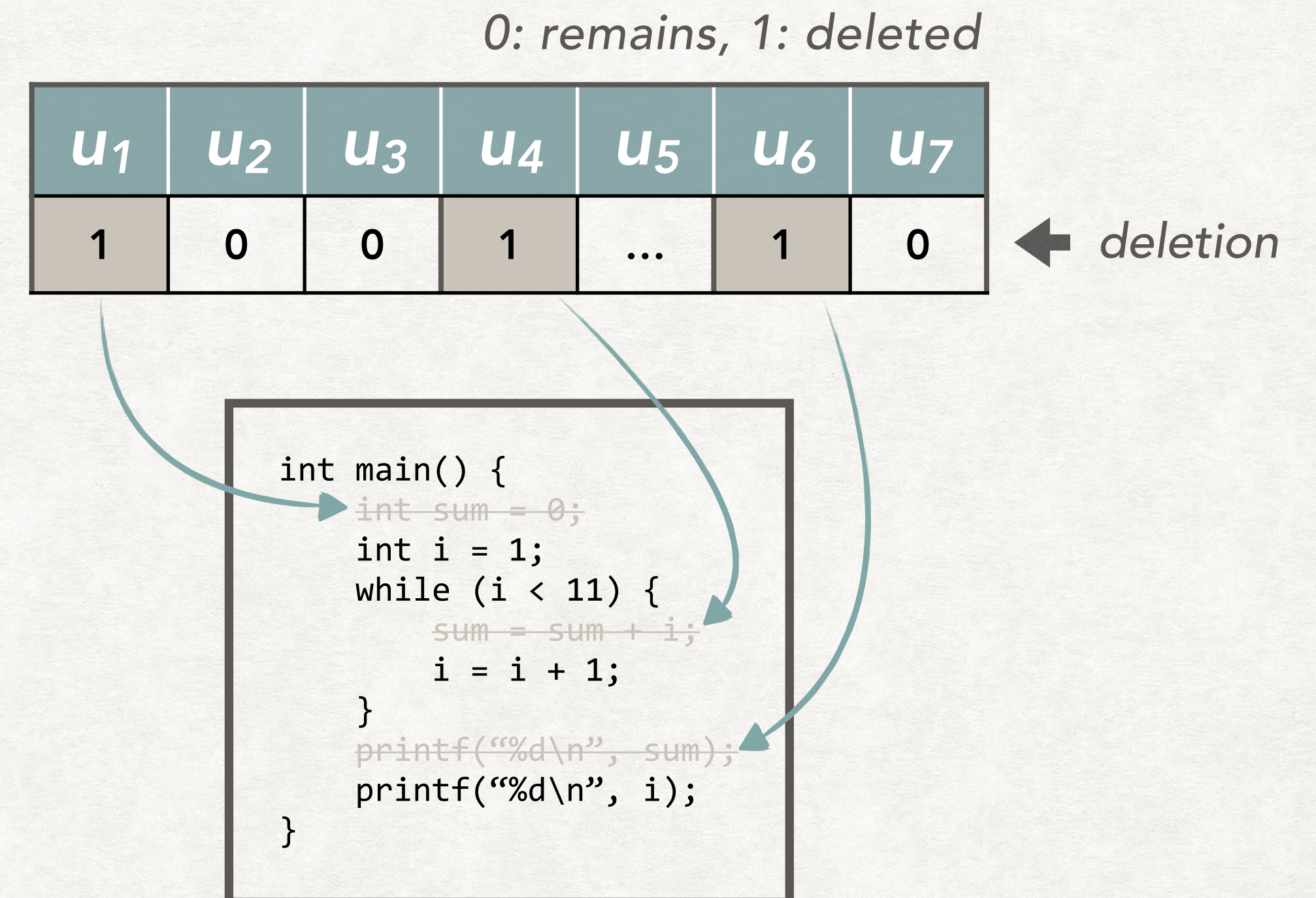
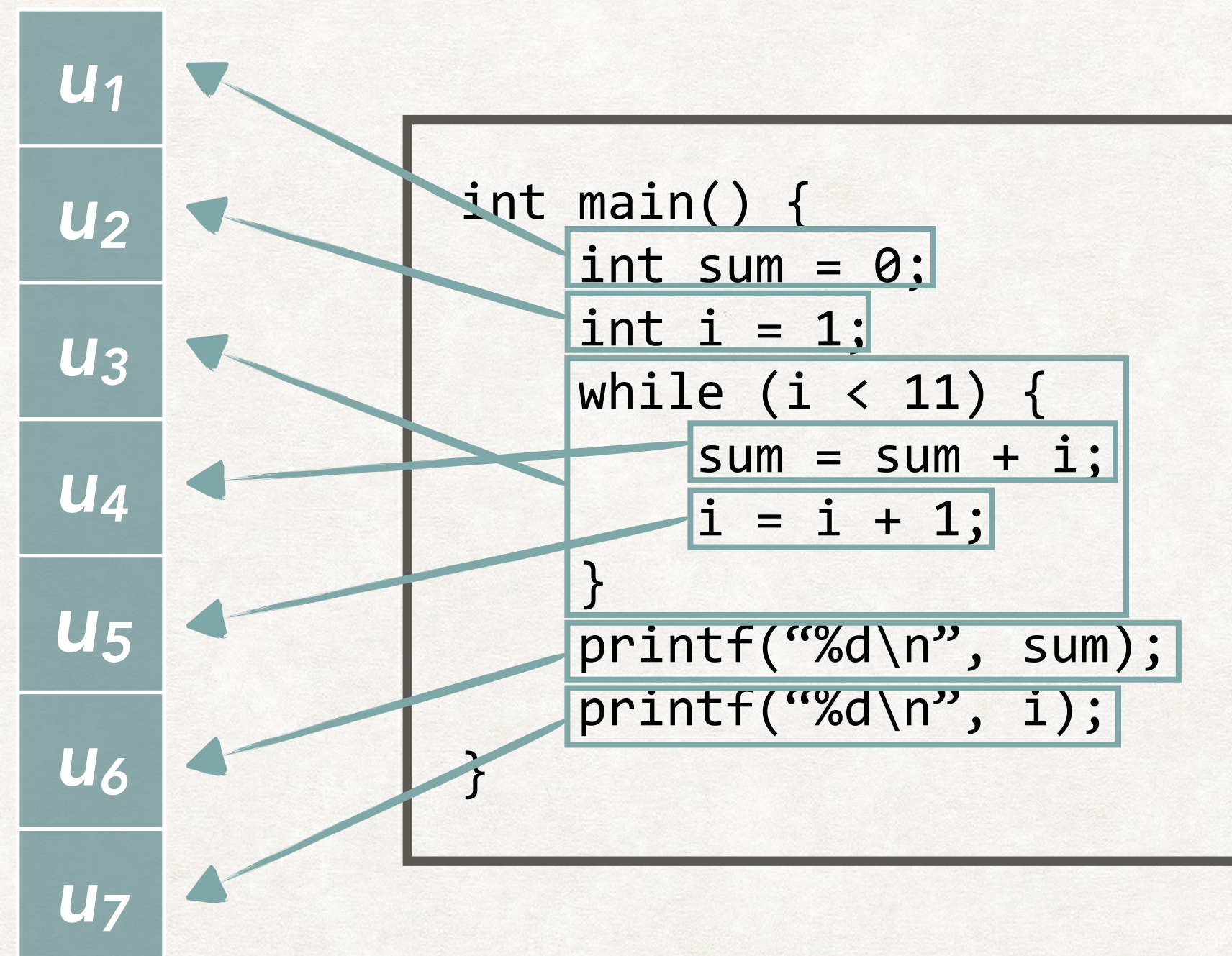
0: remains, 1: deleted

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

original →
(no deletion)

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	...	0	1

+

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

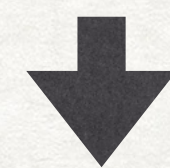
u_1	u_2	u_3	u_4	...	u_{i-1}	u_i
...
1	1	0	0	...	0	0
1	0	1	0	...	0	0
...
0	0	0	0	...	1	1

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

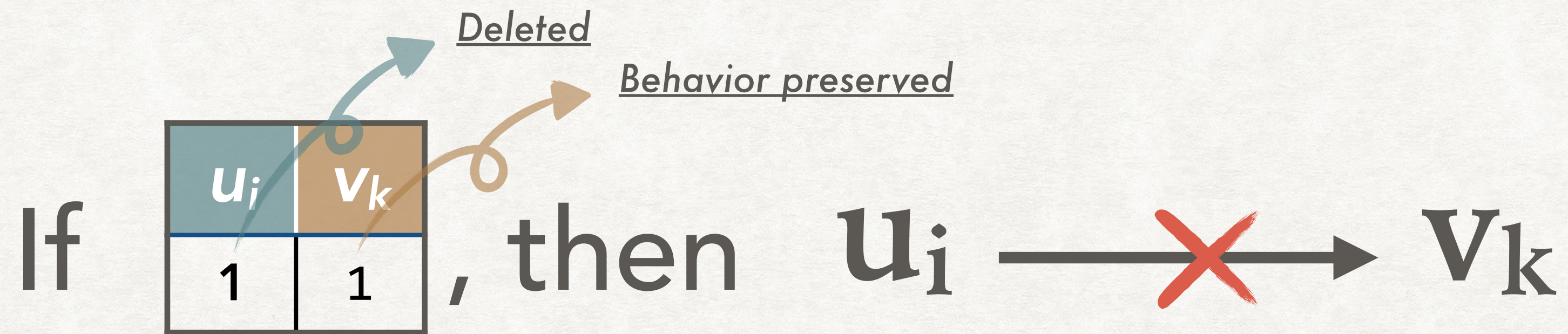
- Main hypothesis:

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

INFERENCE MODEL

1. Once success (⊖)



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

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



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

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



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

Coefficients represent the relative impact on dependence

If $\beta_i > 0$, then $u_i \xrightarrow{\text{X}} v_k$

If $\beta_i \leq 0$, then $u_i \xrightarrow{\text{O}} v_k$

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

$$\log \frac{v_k}{1 - v_k}$$

$$= \beta_0 + \beta_1 u_1 + \beta_2 u_2 + \dots + \beta_i u_i$$

Coefficients represent the relative impact on dependence

If $\beta_i > 0$, then $u_i \xrightarrow{\text{red X}} v_k$

If $\beta_i \leq 0$, then $u_i \xrightarrow{\text{green G}} v_k$

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

$$\log \frac{v_k}{1 - v_k}$$

$$= \beta_0 + \beta_1 u_1 + \beta_2 u_2 + \dots + \beta_i u_i$$

Coefficients represent the relative impact on dependence

If $\beta_i > 0$, then u_i v_k

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

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

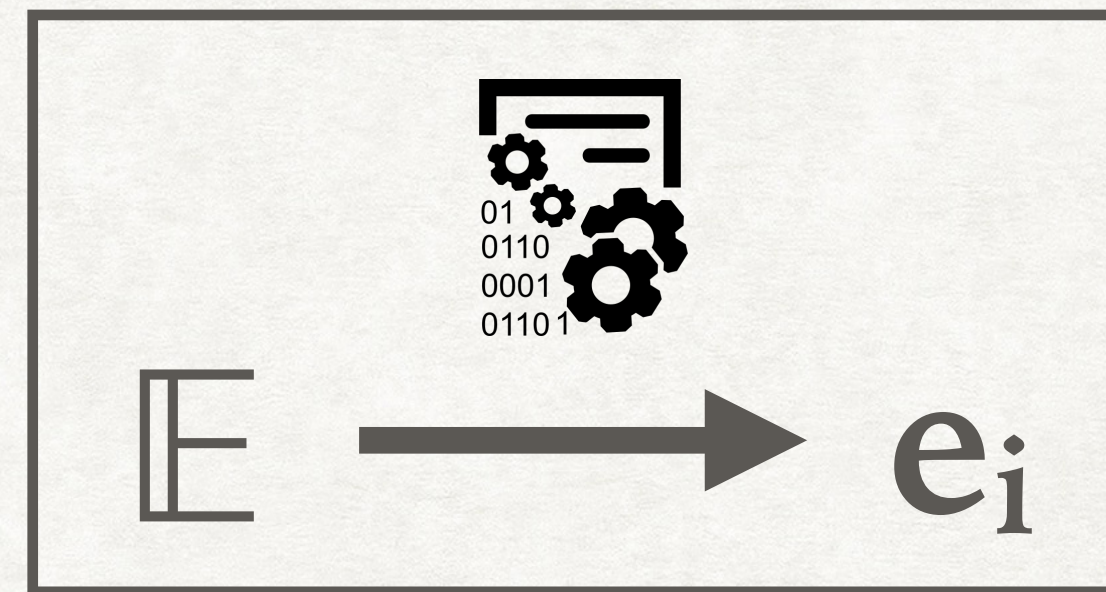
EVALUATION

	0	L	B
1-HOT			
2-HOT			

EVALUATION

	⊙	L	B
1-HOT			
2-HOT			

VS






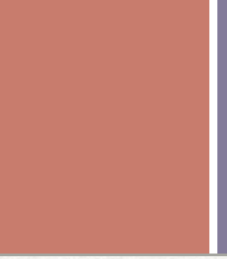



ORBS

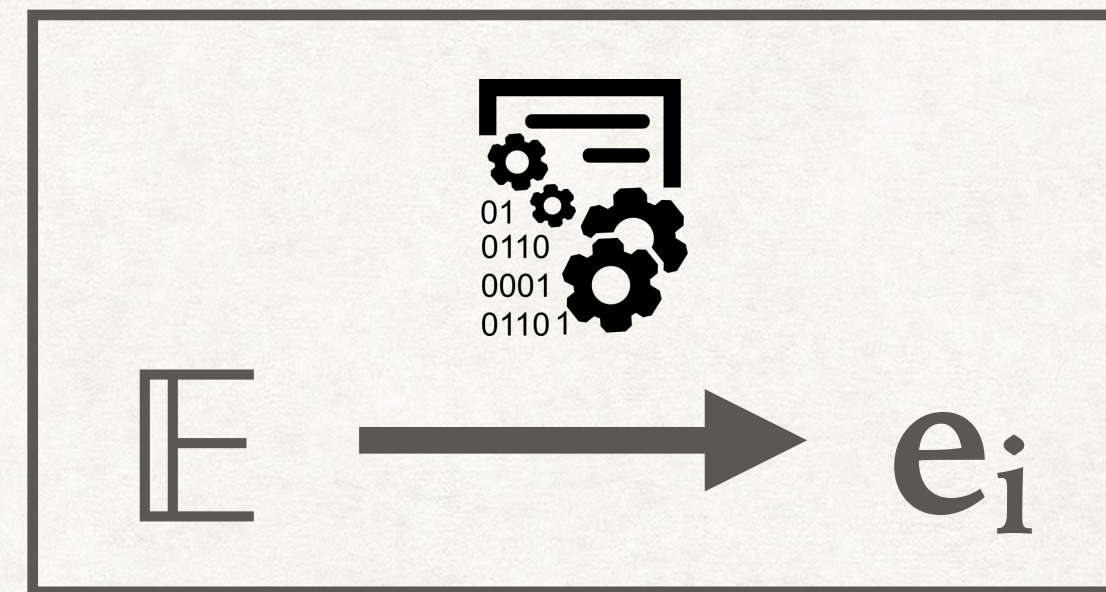
RQ1. MOAD vs. ORBS

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

EVALUATION

		L	B
1-HOT			
2-HOT			

VS



ORBS



Backward element



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

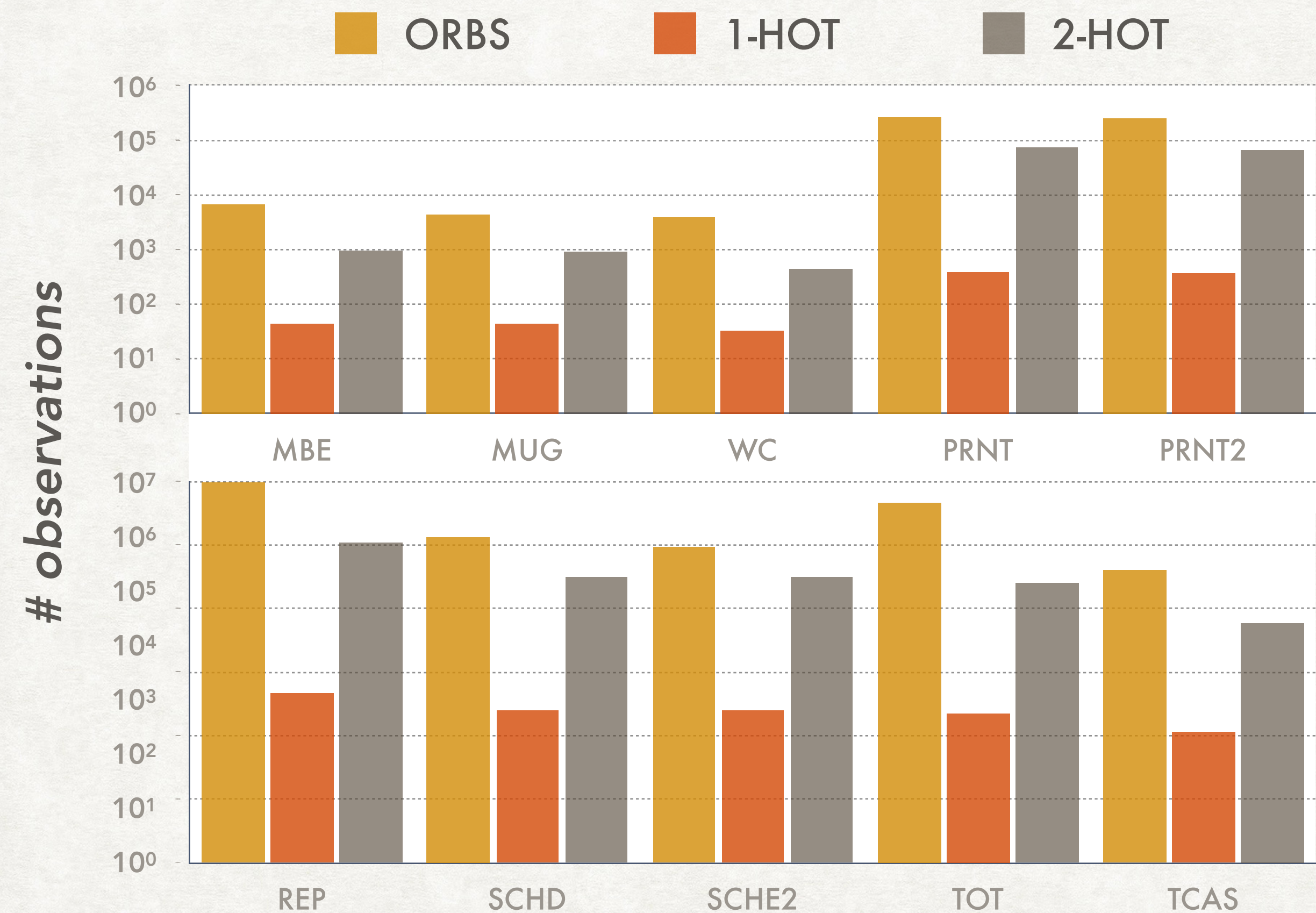
EVALUATION

- 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

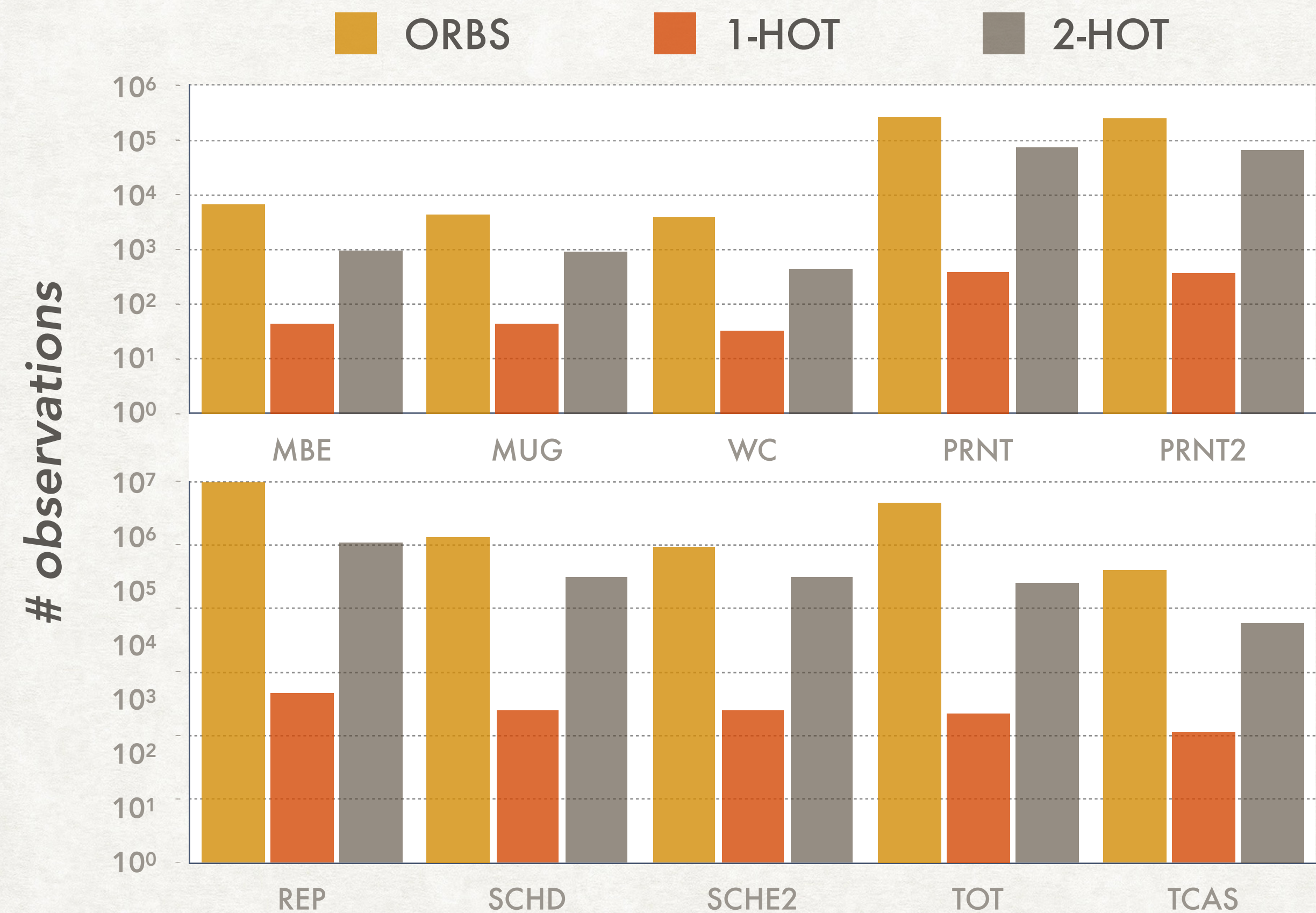
MOAD with 1-HOT used

▸ **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

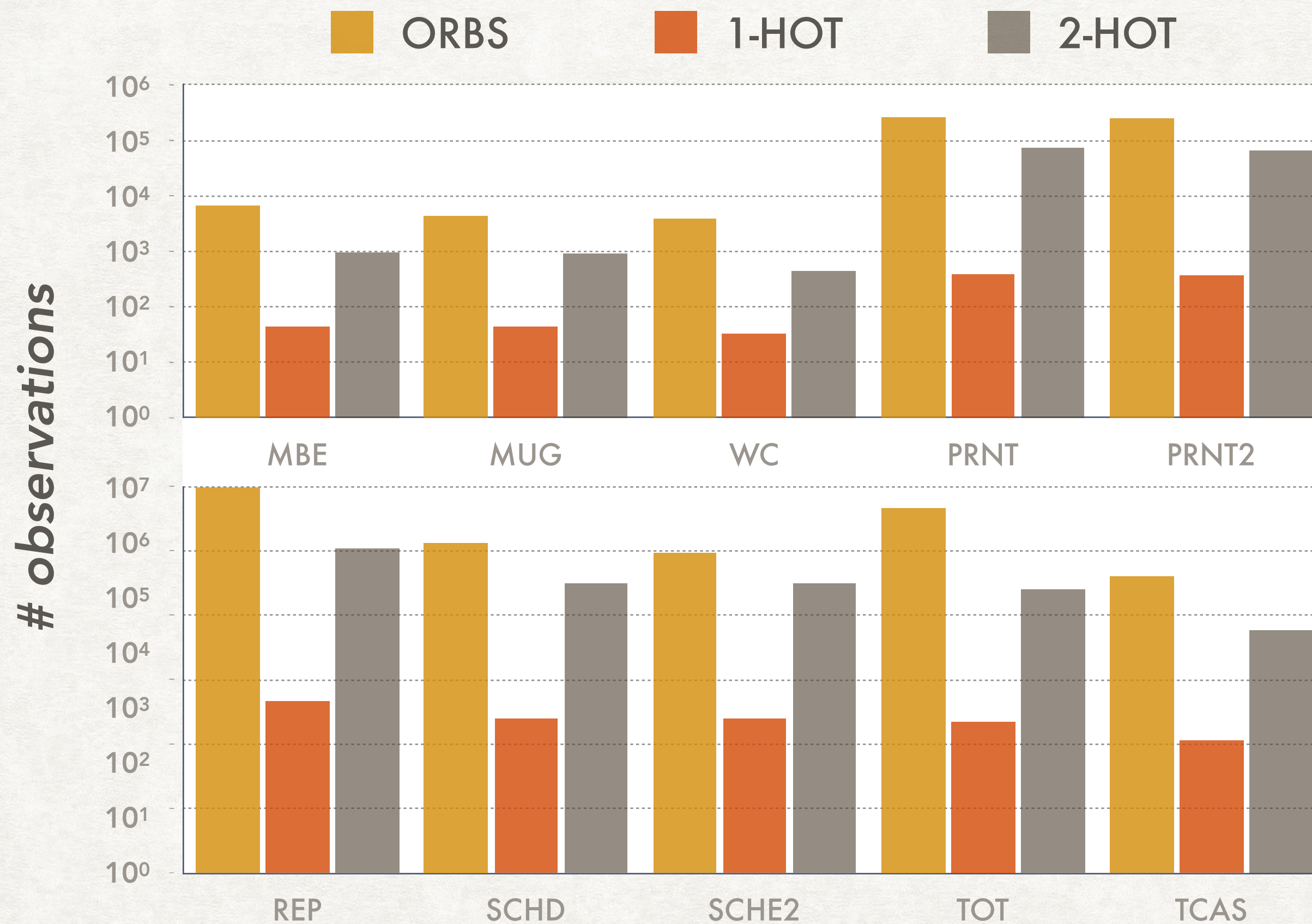
MOAD with 1-HOT used

▸ **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

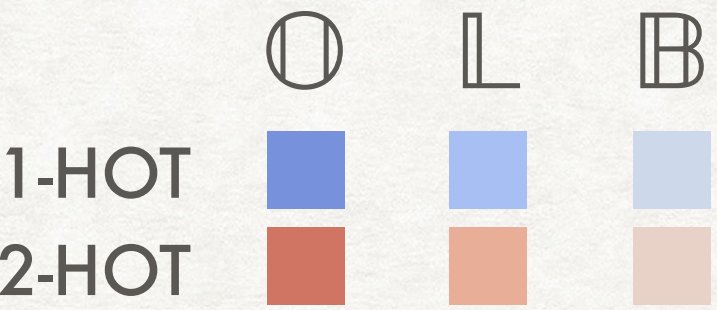
MOAD with 1-HOT used

▸ **0.37%** of the # of observations

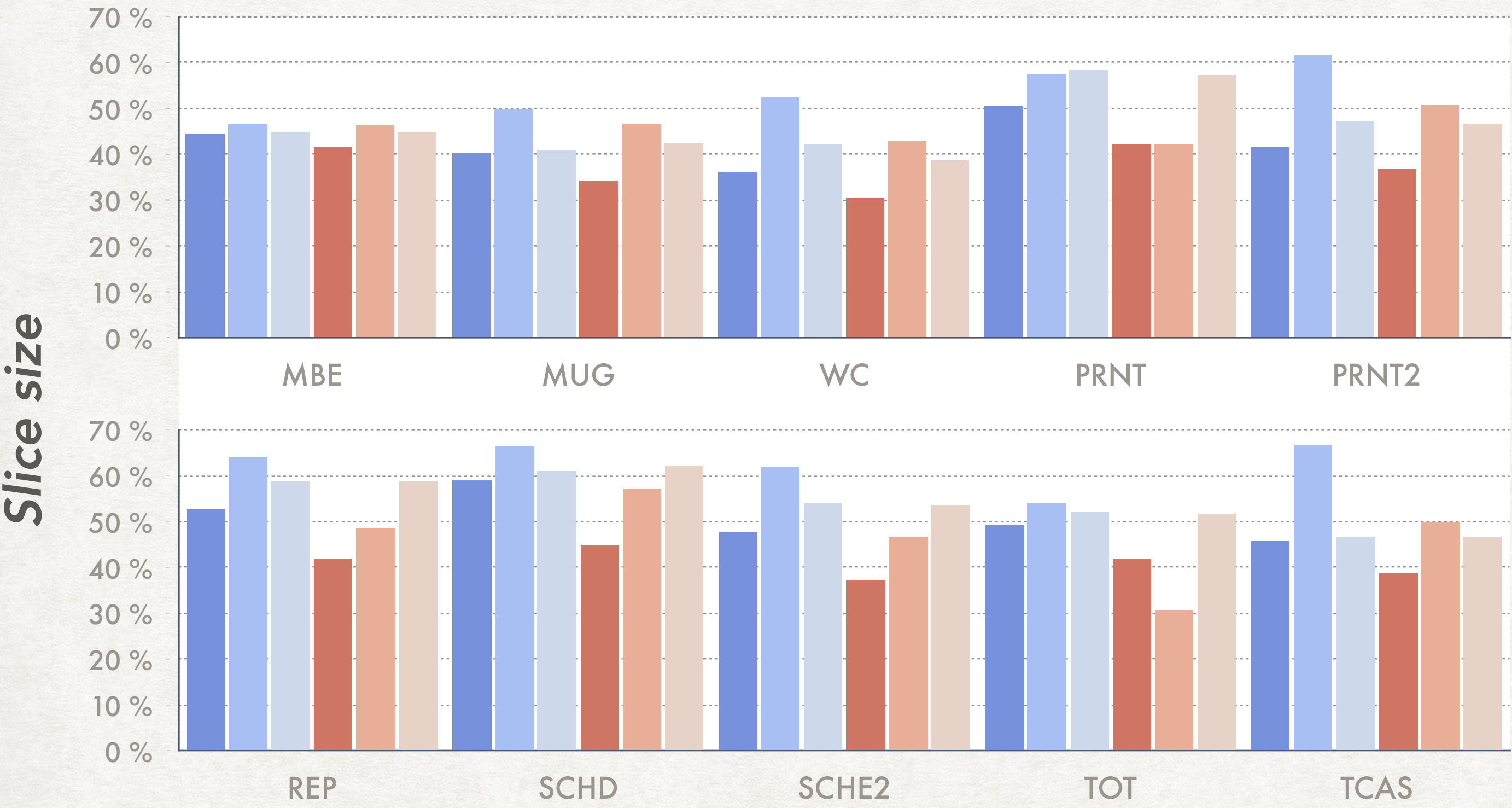
MOAD with 2-HOT used

▸ **18.6%** of the # of observations
compared to ORBS.

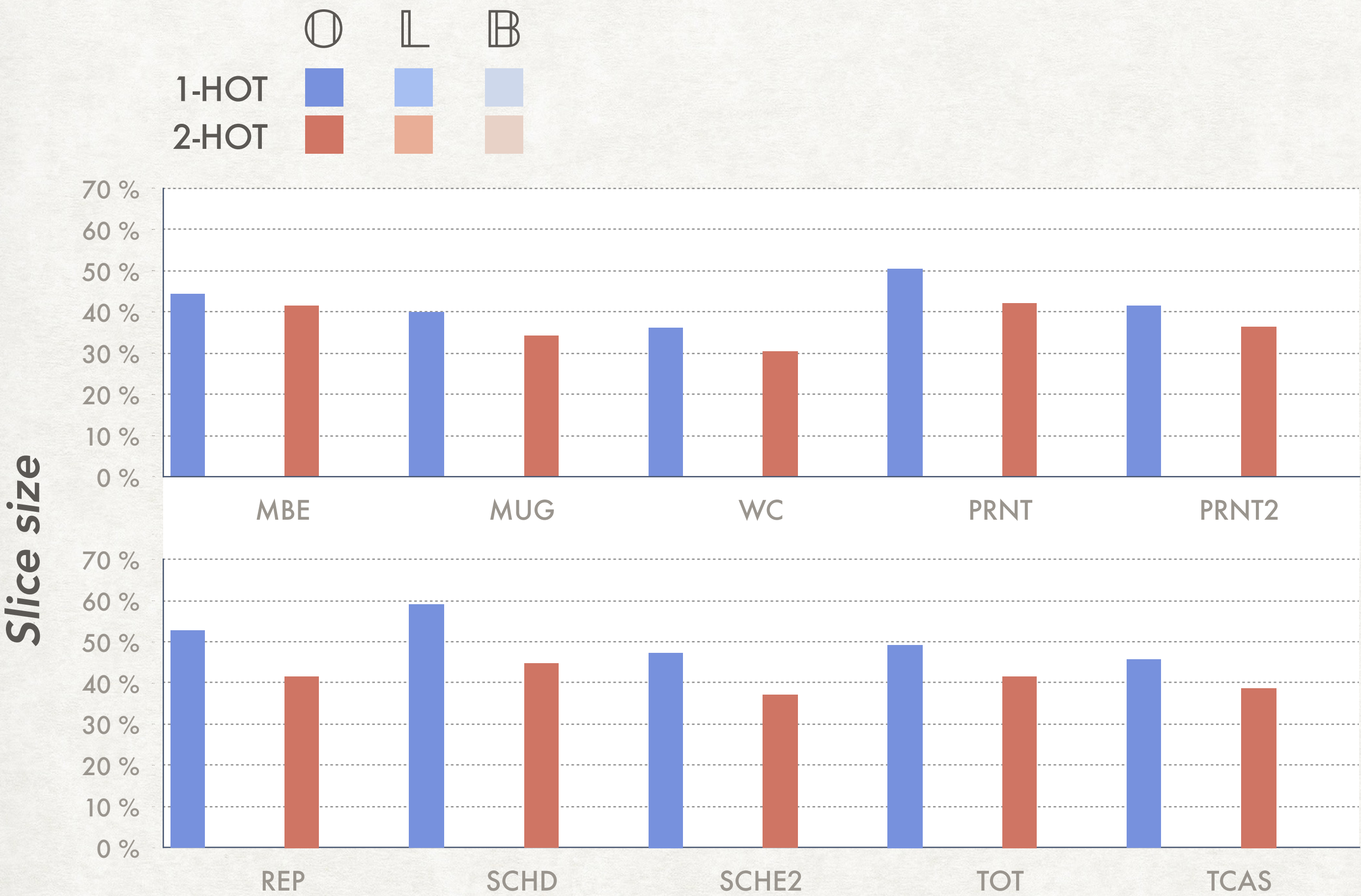
RQ1: MOAD VS. ORBS



EFFECTIVENESS



RQ1: MOAD VS. ORBS

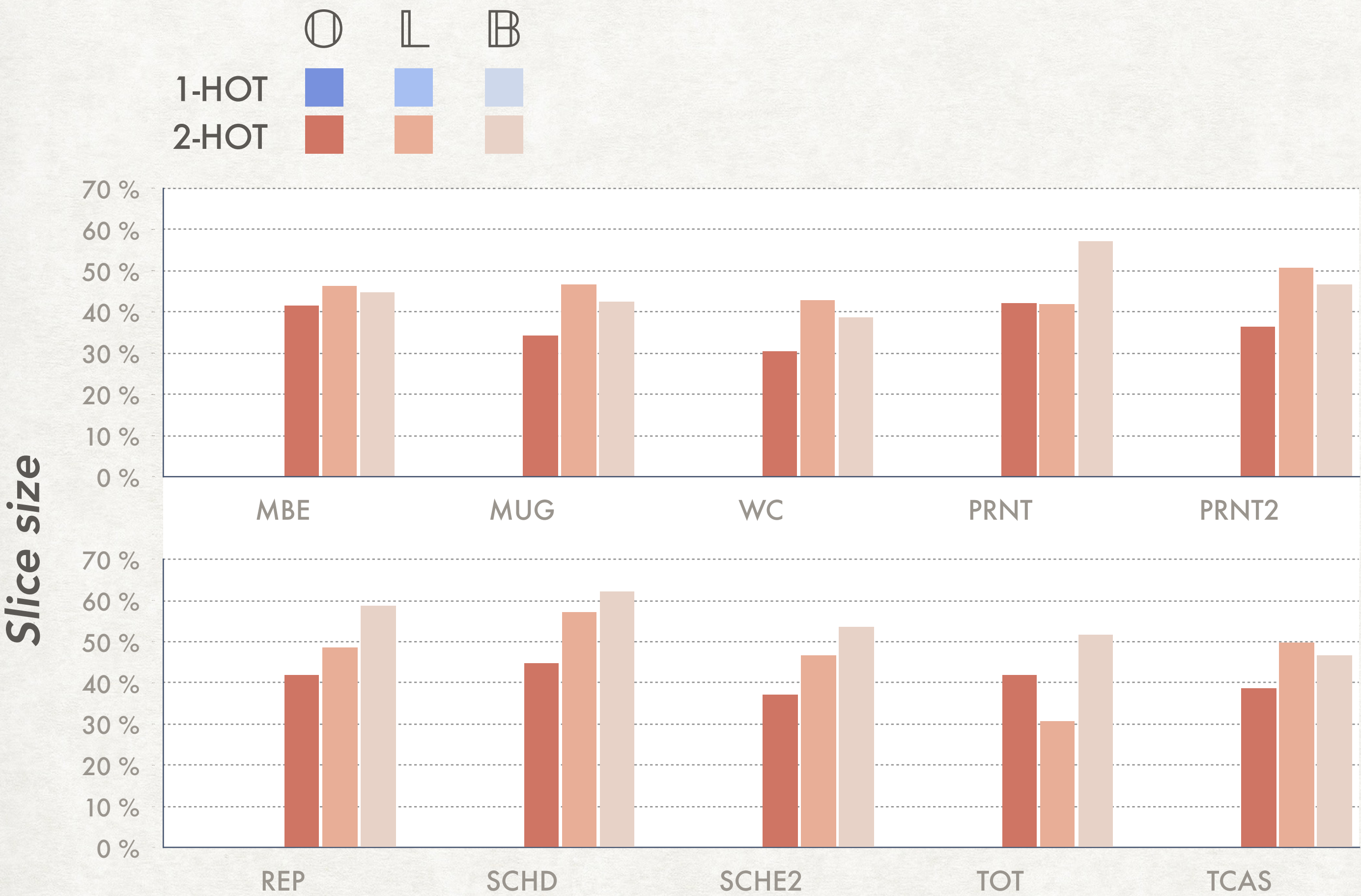


EFFECTIVENESS

For deletion generation scheme,

► **2-HOT** < **1-HOT**.

RQ1: MOAD VS. ORBS



EFFECTIVENESS

For deletion generation scheme,

► **2-HOT** < **1-HOT**.

For inference model,

► \bigcirc < \mathbb{L} , \mathbb{B}

RQ1: MOAD VS. ORBS



EFFECTIVENESS

For deletion generation scheme,

▸ **2-HOT** < **1-HOT**.

For inference model,

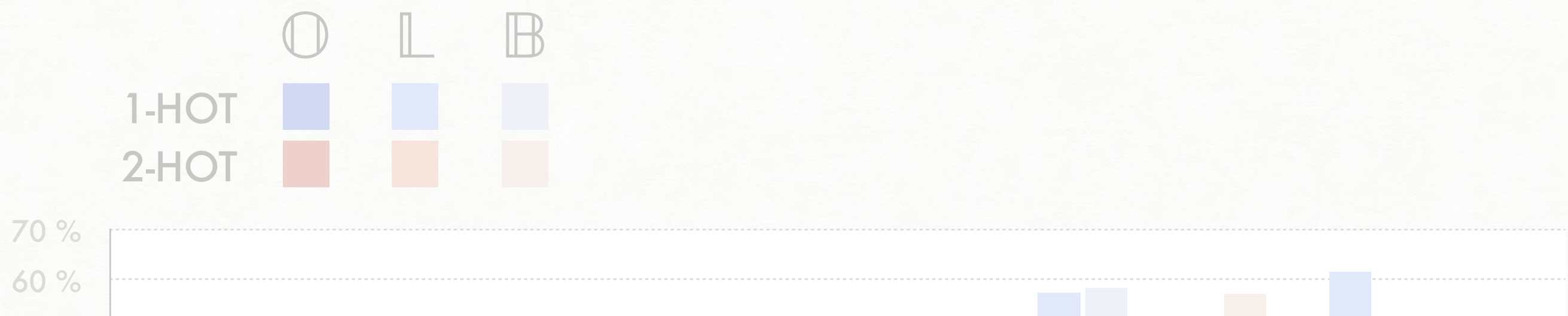
▸ $\oplus < L, B$

MOAD with **2-HOT**, \oplus generate

▸ **12%** larger slices

compared to ORBS.

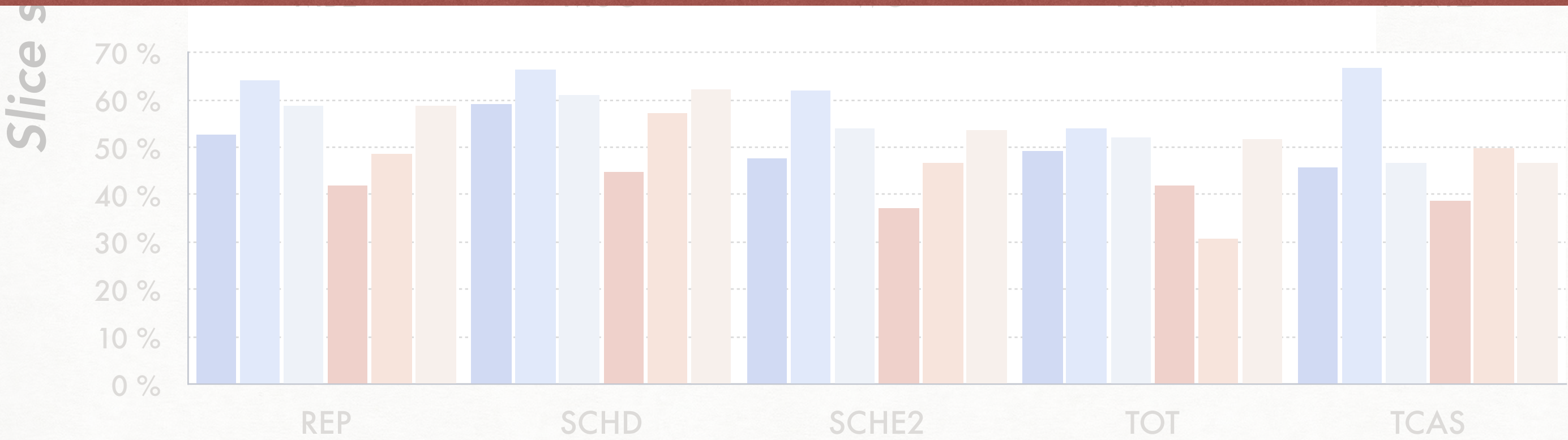
RQ1: MOAD VS. ORBS



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.

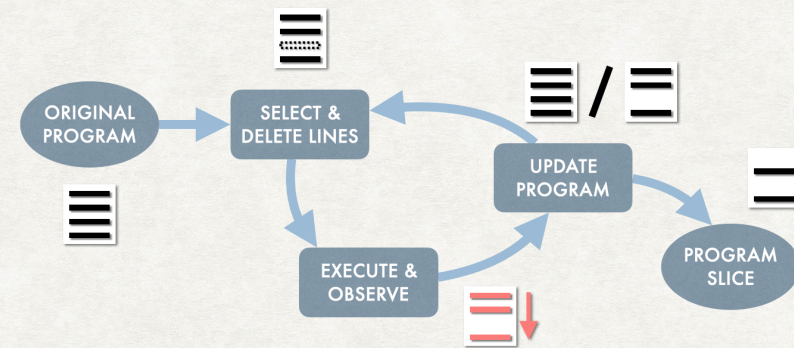
RQ2: MOAD VS. STATIC SLICER

- 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

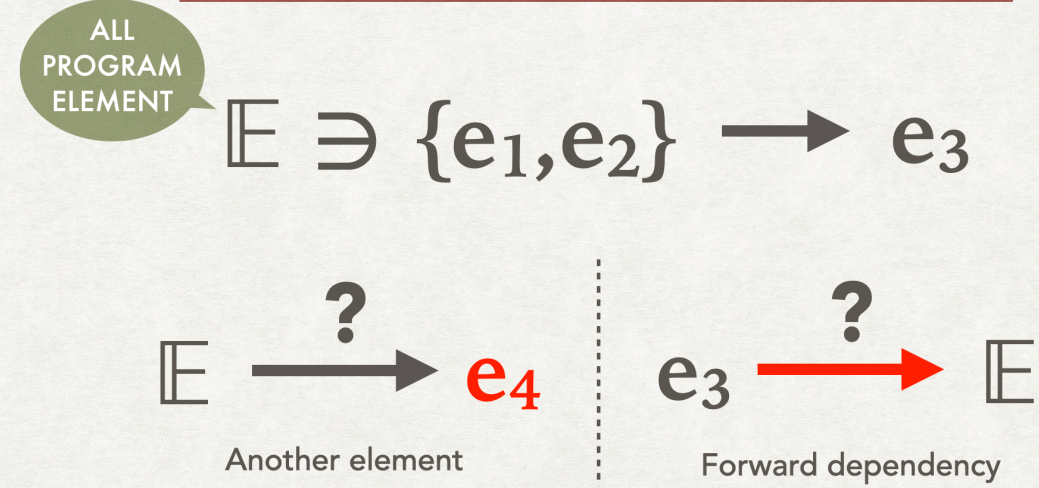
LIMITATION OF ORBS

SCALABILITY



Requires a large number of compilations & executions

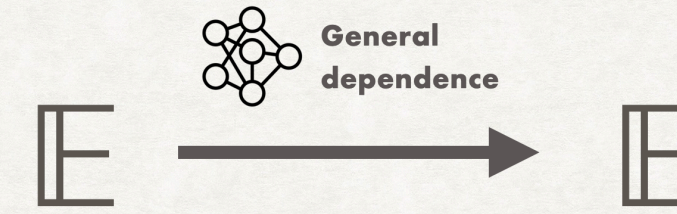
EXPLAINABILITY



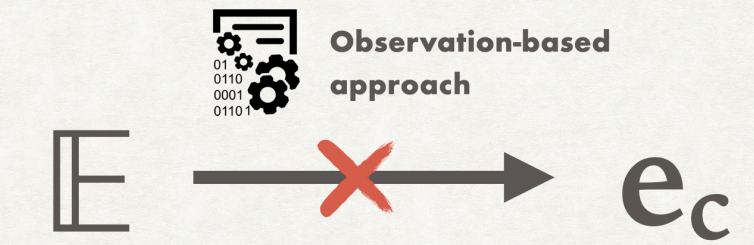
No general dependence / only a single slice

8

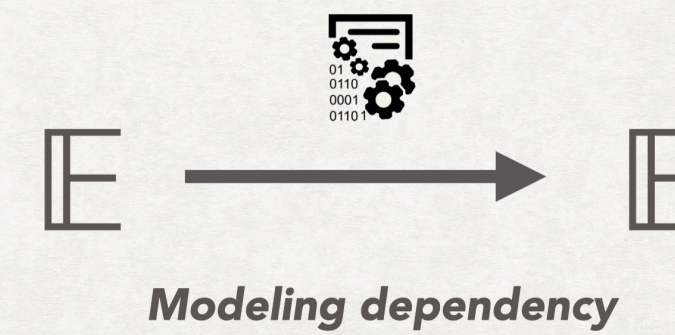
STATIC ANALYSIS



ORBS

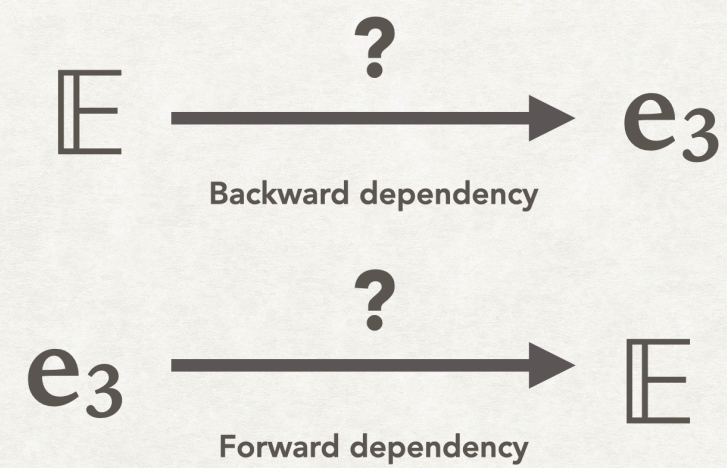
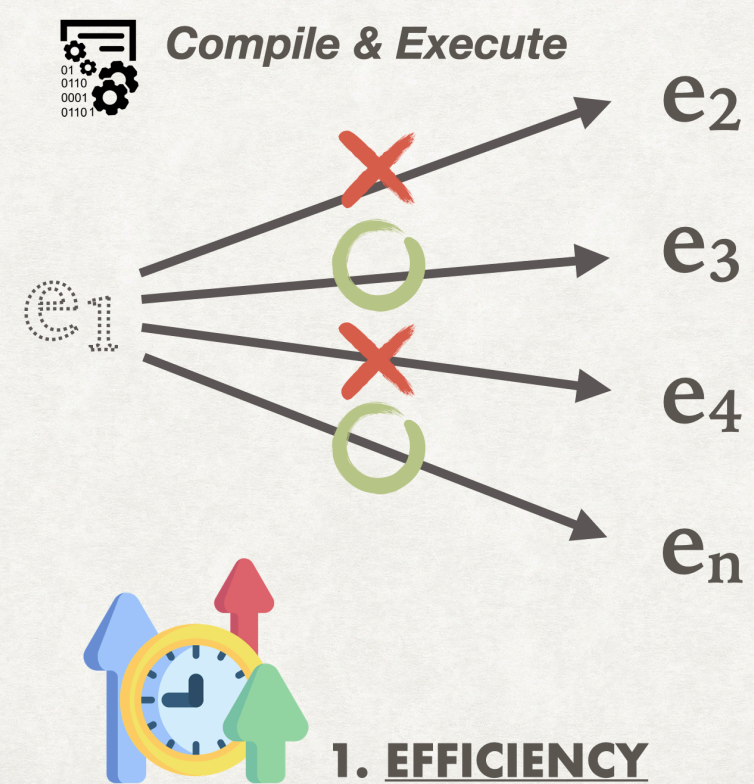


Observation-based analysis



9

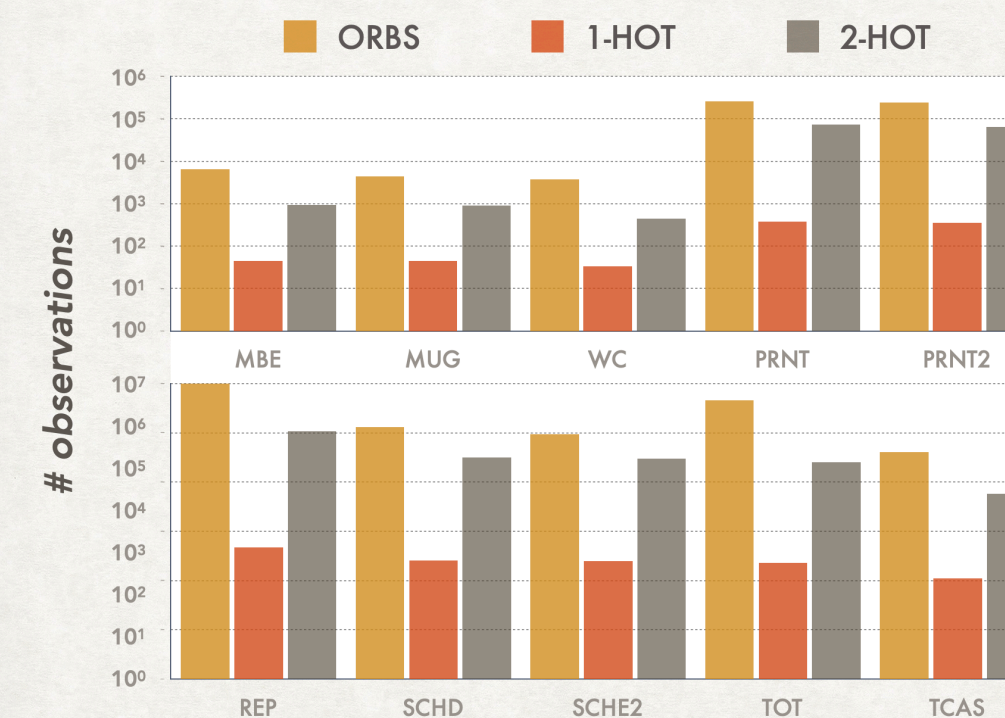
ADVANTAGE OF MOAD VS. ORBS



100
2. COMPLETENESS

13

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.

22