Secure Compilation to Isolated Assembly

Marco Patrignani [1]

[1]iMinds-DistriNet, Dept. Computer Science, KU Leuven, Belgium
first.last@cs.kuleuven.be

7 April 2014

Goal of the Talk

- introduce my research on secure compilation

Goal of the Talk

- introduce my research on secure compilation
- define secure compilation and related notions

Goal of the Talk

- introduce my research on secure compilation
- define secure compilation and related notions
- point out open challenges

Outline

Outline

Why is Protected Modules Architecture (PMA) Interesting?

- it provides deep encapsulation at the lowest level of abstraction

Why is Protected Modules Architecture (PMA) Interesting?

- it provides deep encapsulation at the lowest level of abstraction
- it is the basis of several security-related works

Why is Protected Modules Architecture (PMA) Interesting?

- it provides deep encapsulation at the lowest level of abstraction
- it is the basis of several security-related works
- Intel wants to port it to future processors

What is a Protected Modules Architecture

What is a Protected Modules Architecture

- assembly-level isolation mechanism

What is a Protected Modules Architecture

- assembly-level isolation mechanism
- implemented via Hypervisor, Hardware, Software

What is a Protected Modules Architecture

- assembly-level isolation mechanism
- implemented via Hypervisor, Hardware, Software
- several research and industrial prototypes

What is a Protected Modules Architecture

- assembly-level isolation mechanism
- implemented via Hypervisor, Hardware, Software
- several research and industrial prototypes

Q: How does PMA work?

PMA in action

```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    ...
⋮
0xab00    jmp 0xb53
0xab01    ...
```

- memory space

PMA in action

```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    ...
⋮
0xab00    jmp 0xb53
0xab01    ...
```

- memory space
- protected module = protected memory

PMA in action

```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    ...
⋮
0xab00    jmp 0xb53
0xab01    ...
```

- memory space
- protected module = protected memory
- split in code and data

PMA in action

```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    ...
⋮
0xab00    jmp 0xb53
0xab01    ...
```

r/w

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted

PMA in action

```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    ...
⋮
0xab00    jmp 0xb53
0xab01    ...
```

r/x

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted

PMA in action

```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    ...
⋮
0xab00    jmp 0xb53
0xab01    ...
```

r/w/x

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted

PMA in action

```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    ...
⋮
0xab00    jmp 0xb53
0xab01    ...
```

r/w/x

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted

PMA in action

```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    ...
⋮
0xab00    jmp 0xb53
0xab01    ...
```

r/w/x

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted

PMA in action

```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0x0001
0x0b55    ...
⋮
0xab00    jmp 0xb53
0xab01    ...
```

r/w/x

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted

PMA in action

```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
⋮
■ 0x0b52    movs r0 0x0b55
  0x0b53    call 0x0002
  0x0b54    movs r0 0x0001
  0x0b55    ...
⋮
0xab00    jmp 0xb53
0xab01    ...
```

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted
- entry points for communication (■)

PMA in action

```
            x
0x0001      call 0xb53
0x0002      movs r0 0x0b55
  :
■ 0x0b52    movs r0 0x0b55
  0x0b53    call 0x0002
  0x0b54    movs r0 0x0001
  0x0b55    ...
  :
  0xab00    jmp 0xb53
  0xab01    ...
```

- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted
- entry points for communication (■)

Benefits of Secure Compilation

Benefits of Secure Compilation

Secure compilation preserves source-level abstractions in target-level languages

Benefits of Secure Compilation

Secure compilation preserves source-level abstractions in target-level languages

- protect against code injection attacks

Benefits of Secure Compilation

Secure compilation preserves source-level abstractions in target-level languages

- protect against code injection attacks
- enables source-level reasoning

Outline

Languages of the Compiler

- source language: +/- Java jr

Languages of the Compiler

- source language: +/- Java jr
  - component-based
  - private fields
  - programming to an interface
  - exceptions

Languages of the Compiler

- source language: +/- Java jr
  - component-based
  - private fields
  - programming to an interface
  - exceptions

```
1  package PI;
2    interface Account {
3      public createAccount() : Foo;
4    }
5    extern extAccount : Account;
6
7  package PE;
8    class AccountClass
9      implements PI.Account {
10     AccountClass() { counter = 0; }
11     public createAccount() : Account {
12       return new PE.AccountClass();
13     }
14
15     private counter : Int;
16   }
17   object extAccount : AccountClass;
```

Languages of the Compiler

- source language: +/- Java jr
  - component-based
  - private fields
  - programming to an interface
  - exceptions

Q: How to securely compile this code?

```
1  package PI;
2    interface Account {
3      public createAccount() : Foo;
4    }
5    extern extAccount : Account;
6
7  package PE;
8    class AccountClass
9      implements PI.Account {
10     AccountClass() { counter = 0; }
11     public createAccount() : Account {
12       return new PE.AccountClass();
13     }
14
15     private counter : Int;
16   }
17   object extAccount : AccountClass;
```

Languages of the Compiler

```
1  package PI;
2    interface Account {
3      public createAccount() : Foo;
4    }
5    extern extAccount : Account;
6
7  package PE;
8    class AccountClass
9      implements PI.Account {
10     AccountClass() { counter = 0; }
11     public createAccount() : Account {
12       return new PE.AccountClass();
13     }
14
15     private counter : Int;
16   }
17   object extAccount : AccountClass;
```

Languages of the Compiler

Dynamic dispatch

---

v-tables

Secure stack

```
1  package PI;
2    interface Account {
3      public createAccount() : Foo;
4    }
5    extern extAccount : Account;
6
7  package PE;
8    class AccountClass
9      implements PI.Account {
10     AccountClass() { counter = 0; }
11     public createAccount() : Account {
12       return new PE.AccountClass();
13     }
14
15     private counter : Int;
16   }
17   object extAccount : AccountClass;
```

Languages of the Compiler

- proxy to `createAccount`

Dynamic dispatch

---

v-tables

Secure stack

```
1  package PI;
2    interface Account {
3      public createAccount() : Foo;
4    }
5    extern extAccount : Account;
6
7  package PE;
8    class AccountClass
9      implements PI.Account {
10     AccountClass() { counter = 0; }
11     public createAccount() : Account {
12       return new PE.AccountClass();
13     }
14
15     private counter : Int;
16   }
17   object extAccount : AccountClass;
```

Languages of the Compiler

- proxy to `createAccount`

  `createAccount` body

  constructor

Dynamic dispatch

---

v-tables

Secure stack

extAccount
counter

```
1  package PI;
2    interface Account {
3      public createAccount() : Foo;
4    }
5    extern extAccount : Account;
6
7  package PE;
8    class AccountClass
9      implements PI.Account {
10     AccountClass() { counter = 0; }
11     public createAccount() : Account {
12       return new PE.AccountClass();
13     }
14
15     private counter : Int;
16   }
17   object extAccount : AccountClass;
```

Secure Compilation, Informally

Source level

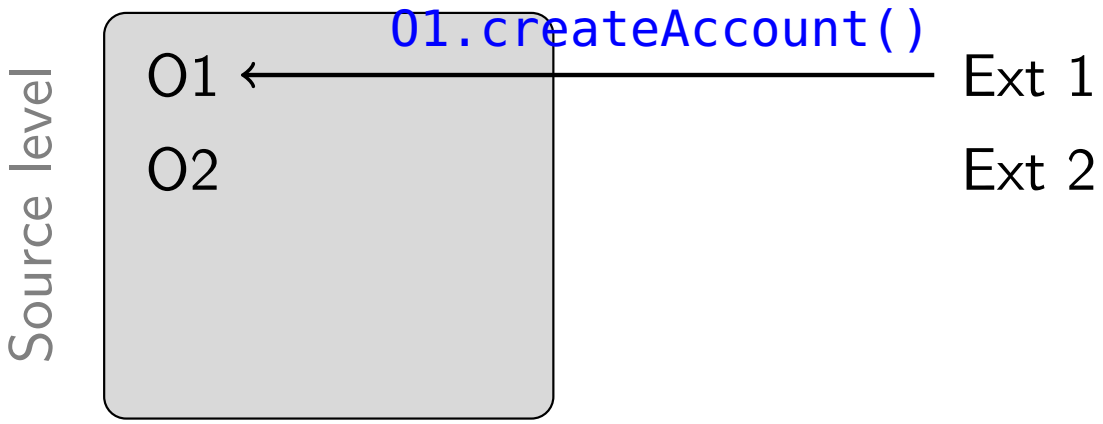
O1

O2

Ext 1

Ext 2

Secure Compilation, Informally



Secure Compilation, Informally

Source level

O1

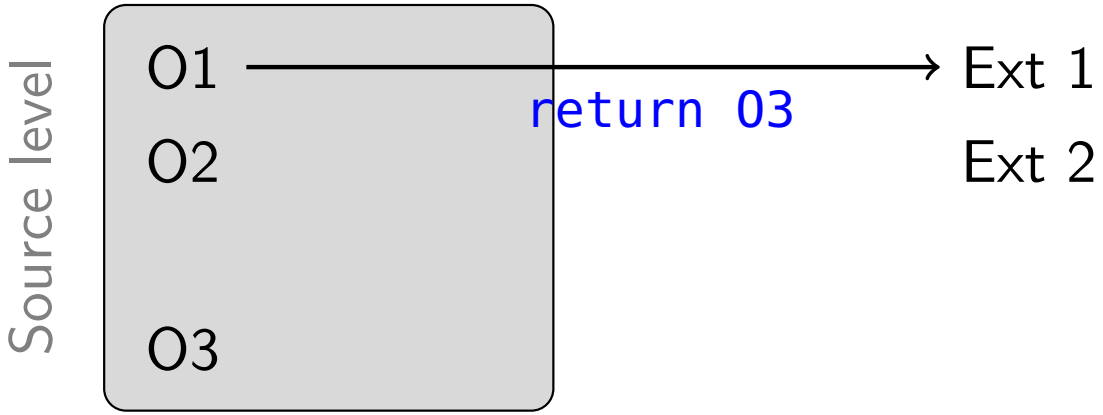
O2

O3

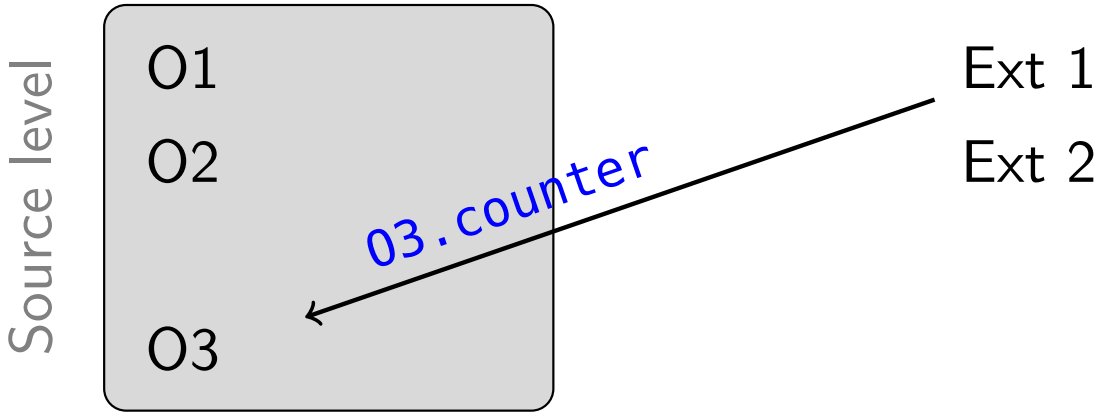
Ext 1

Ext 2

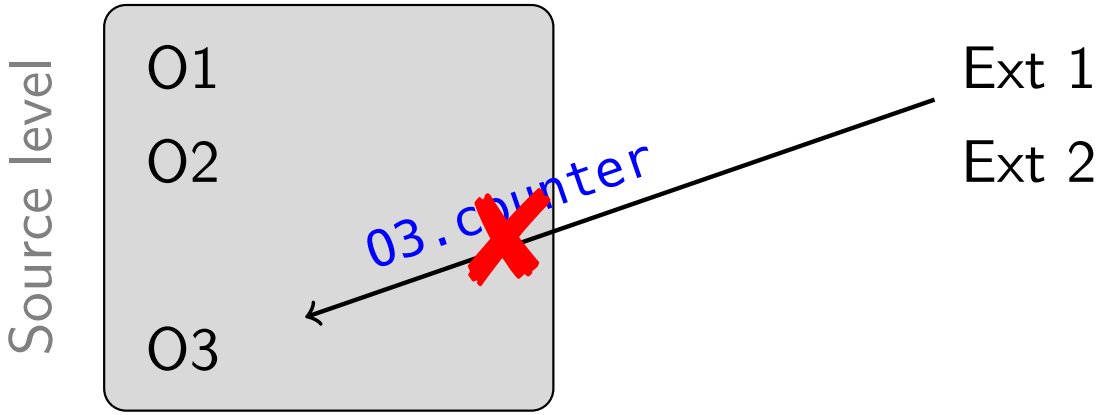
Secure Compilation, Informally



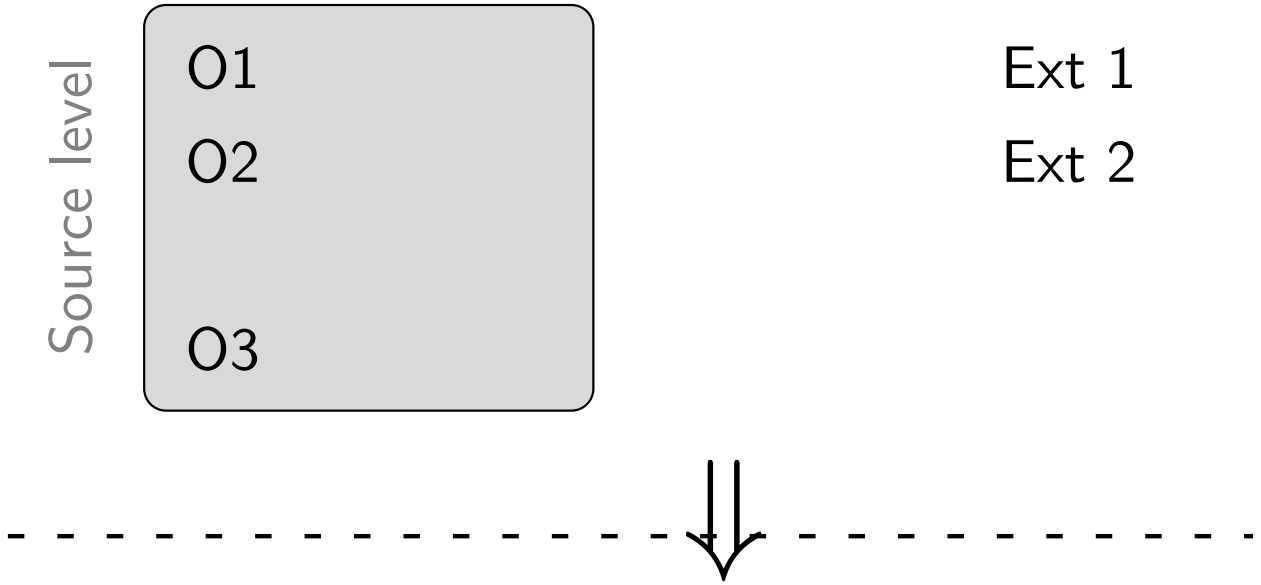
Secure Compilation, Informally



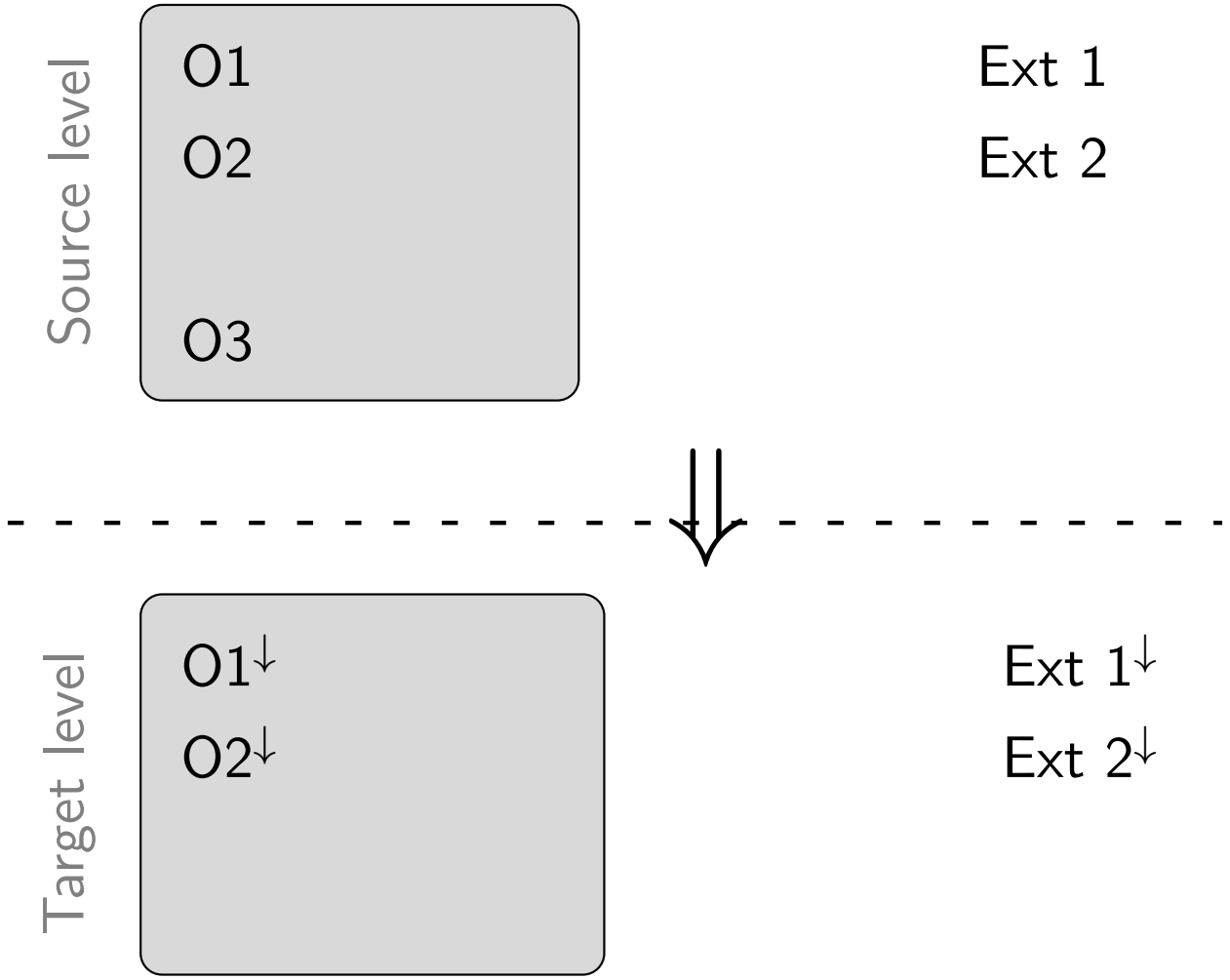
Secure Compilation, Informally



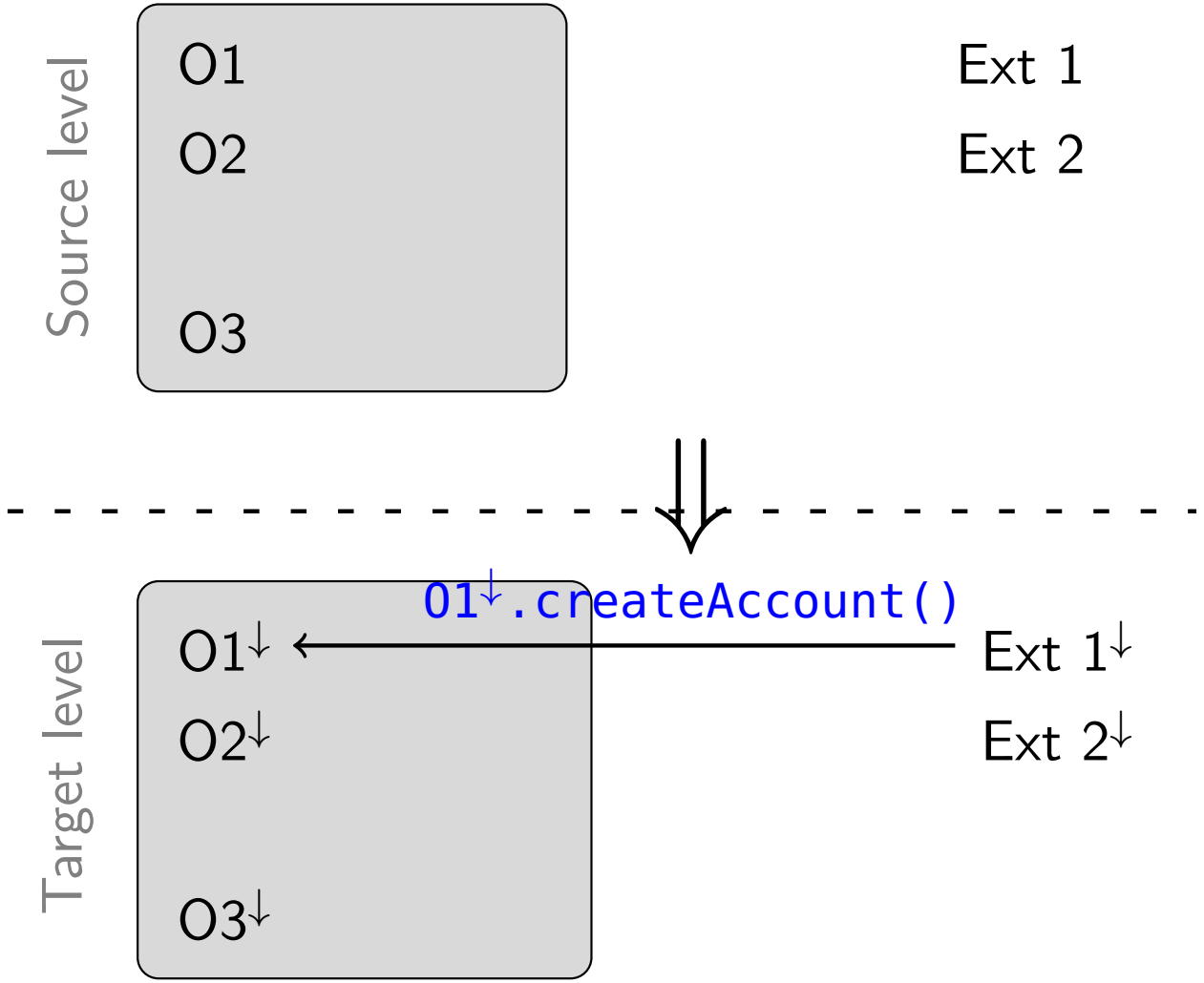
Secure Compilation, Informally



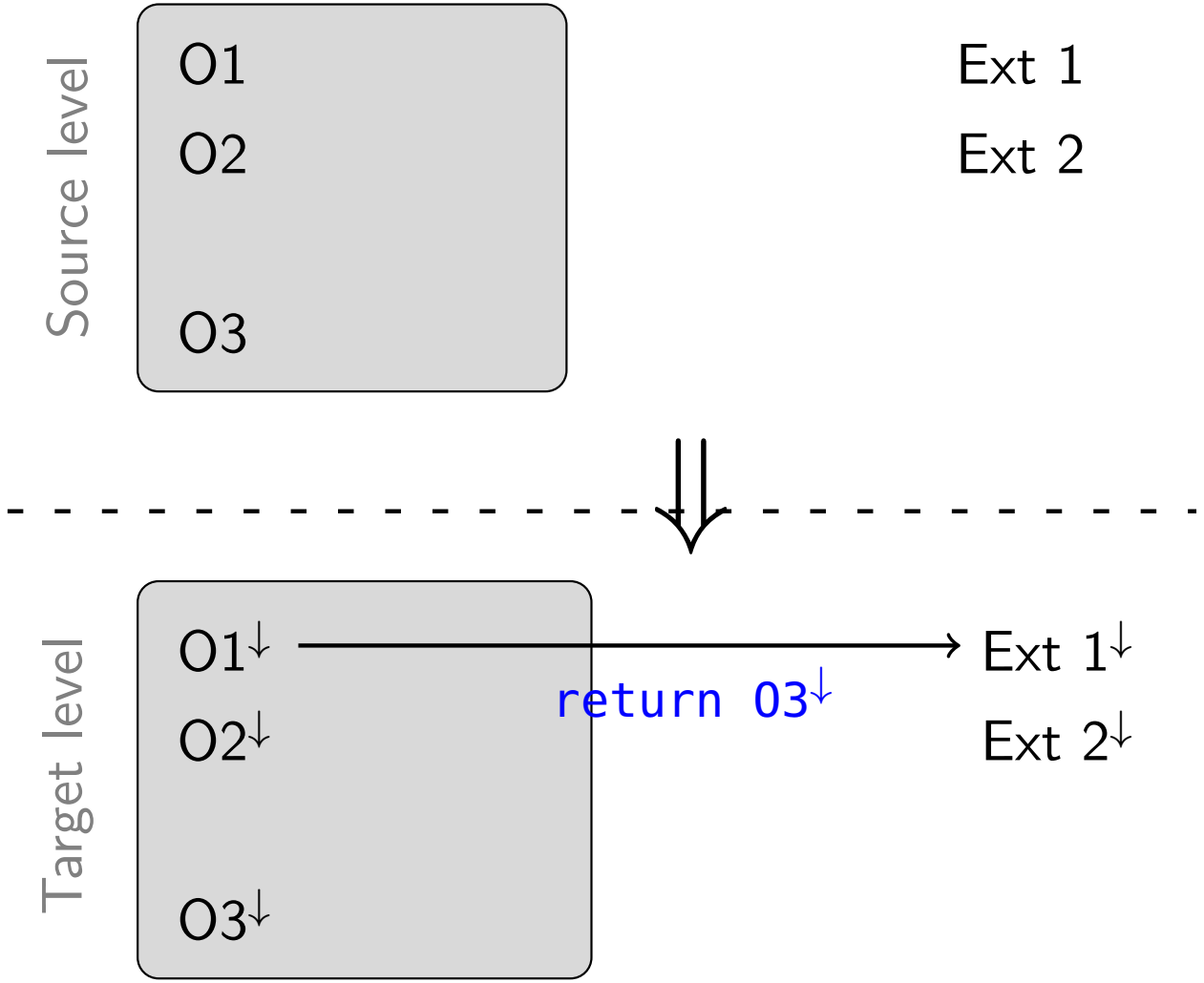
Secure Compilation, Informally



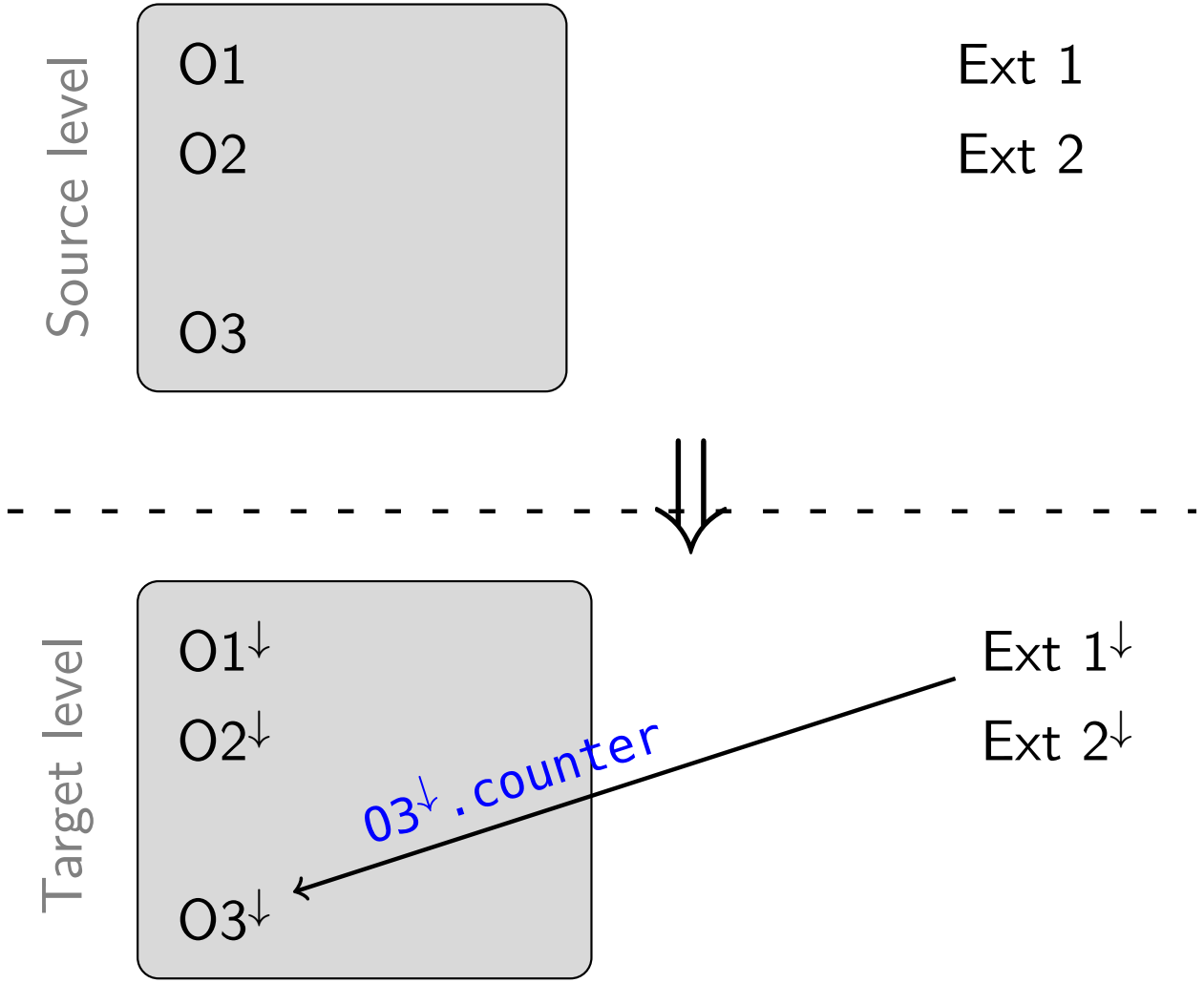
Secure Compilation, Informally



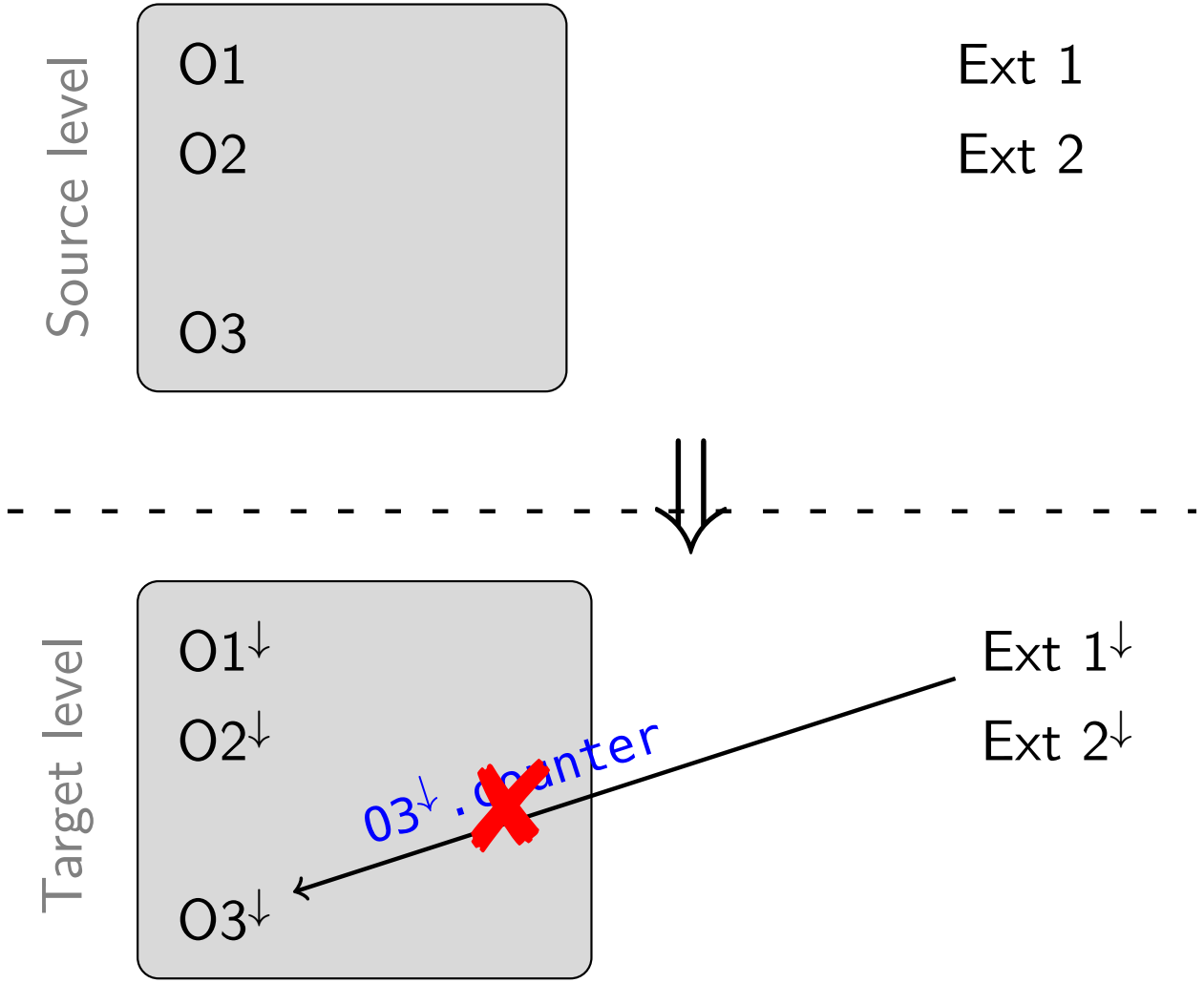
Secure Compilation, Informally



Secure Compilation, Informally

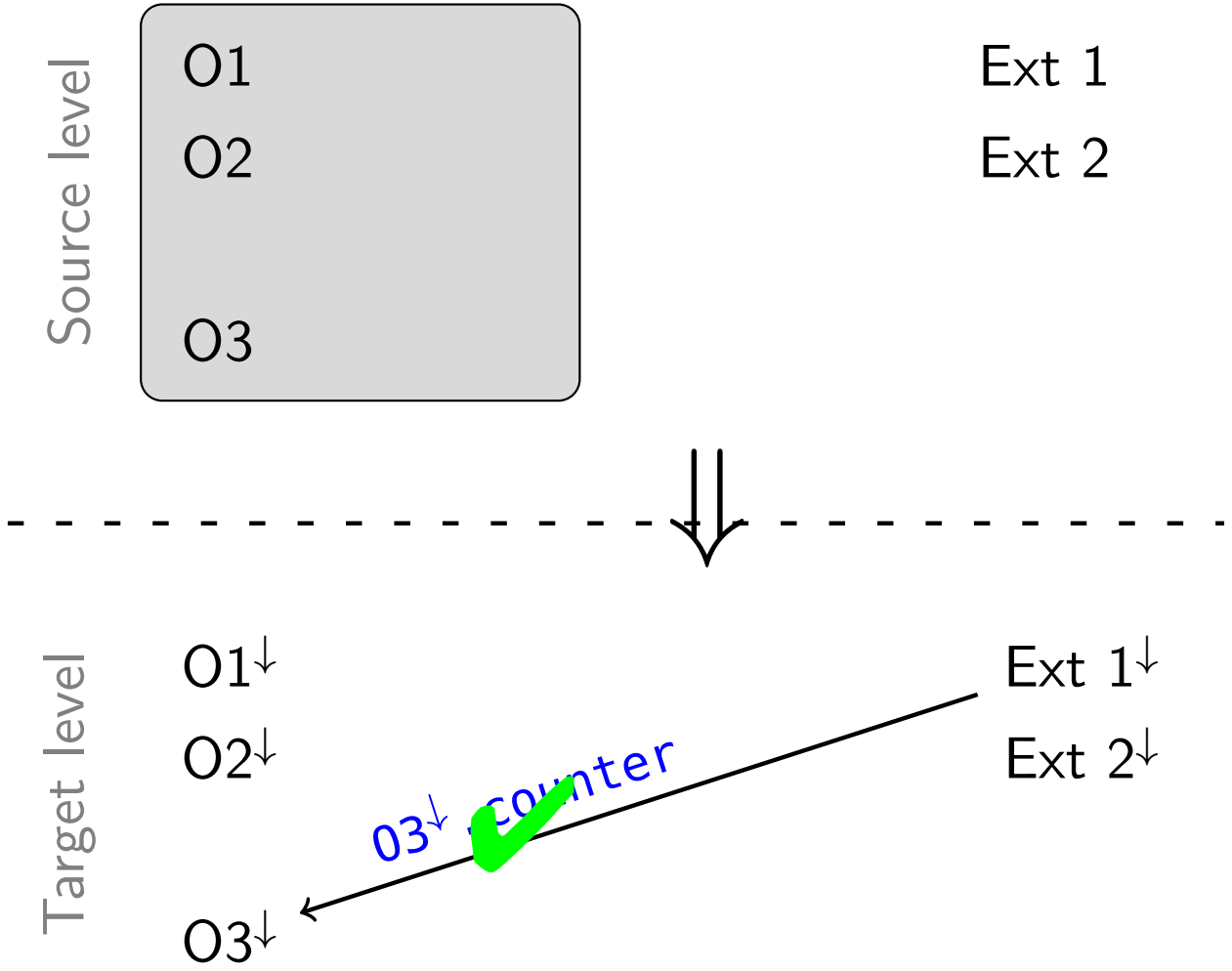


Secure Compilation, Informally



- Protect against low-level attackers

Secure Compilation, Informally



- Protect against low-level attackers
- Target code is vulnerable without PMA

Agten's Work

Q: : Is that all?

Agten's Work

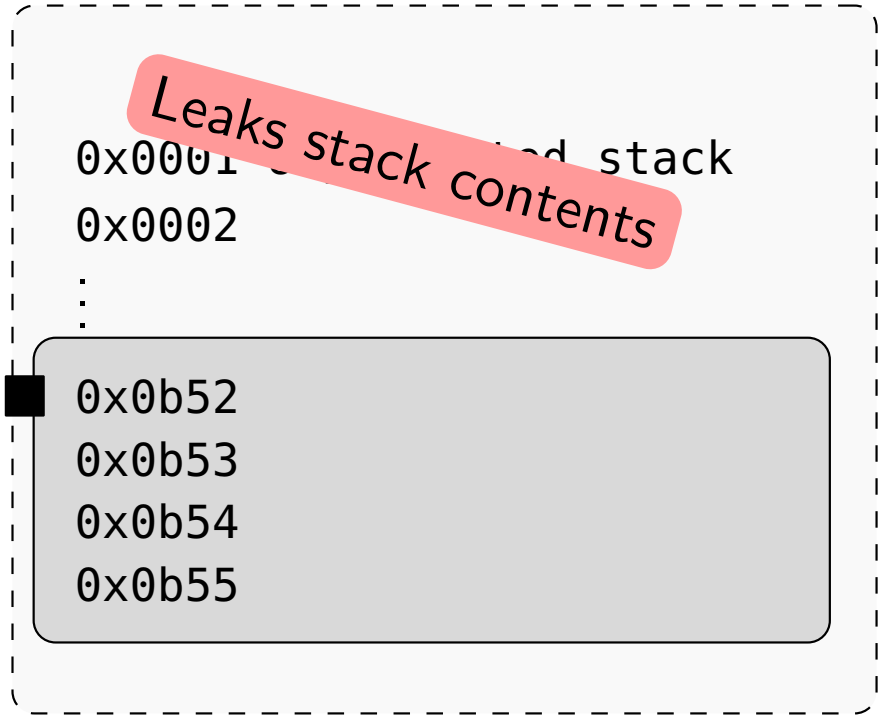
Q: : Is that all?

- protected stack

Agten's Work

Q: : Is that all?

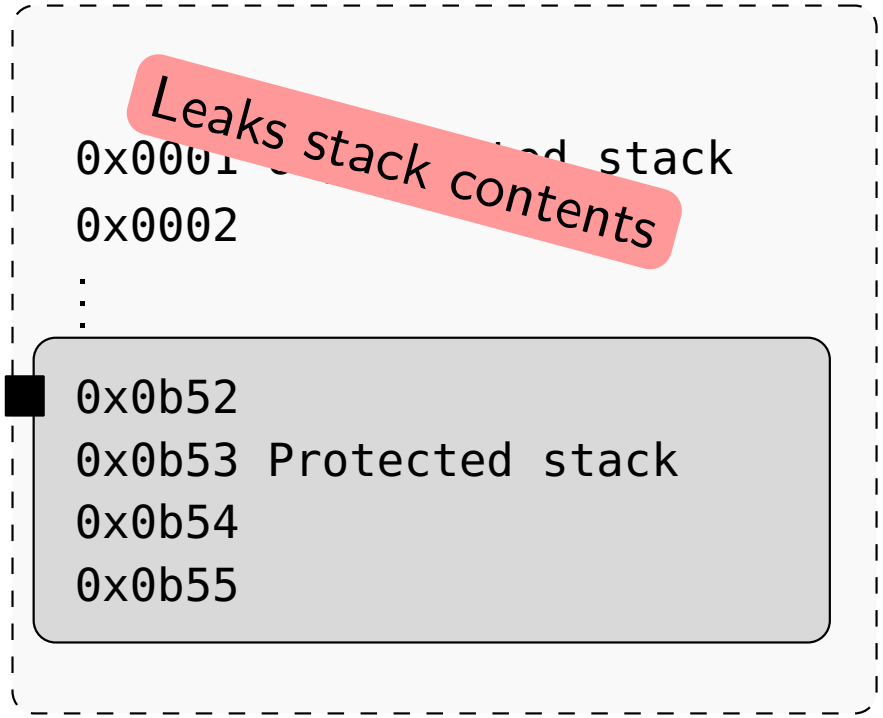
- protected stack



Agten's Work

Q: : Is that all?

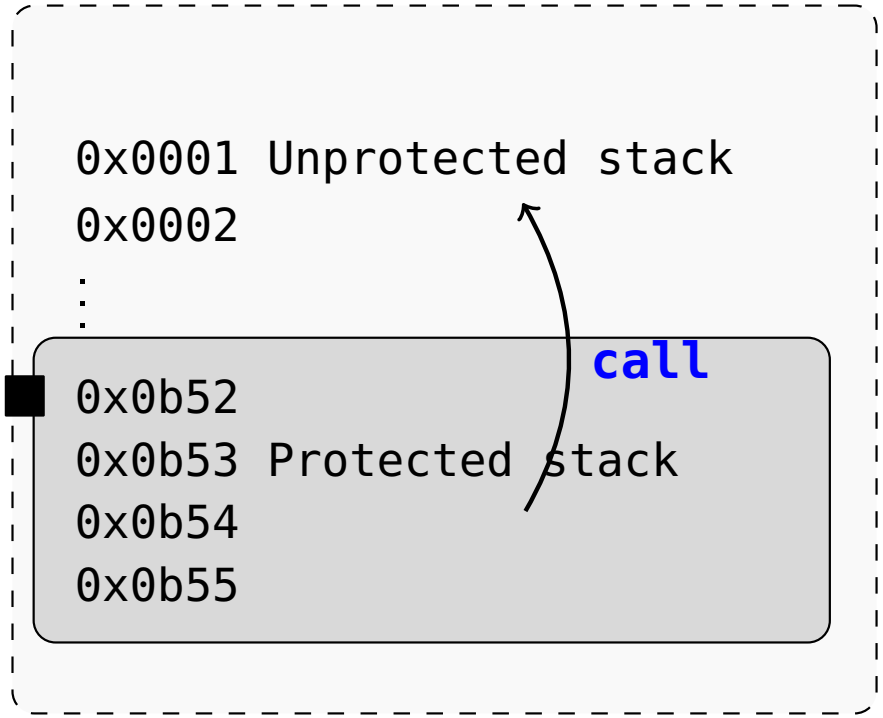
- protected stack



Agten's Work

Q: : Is that all?

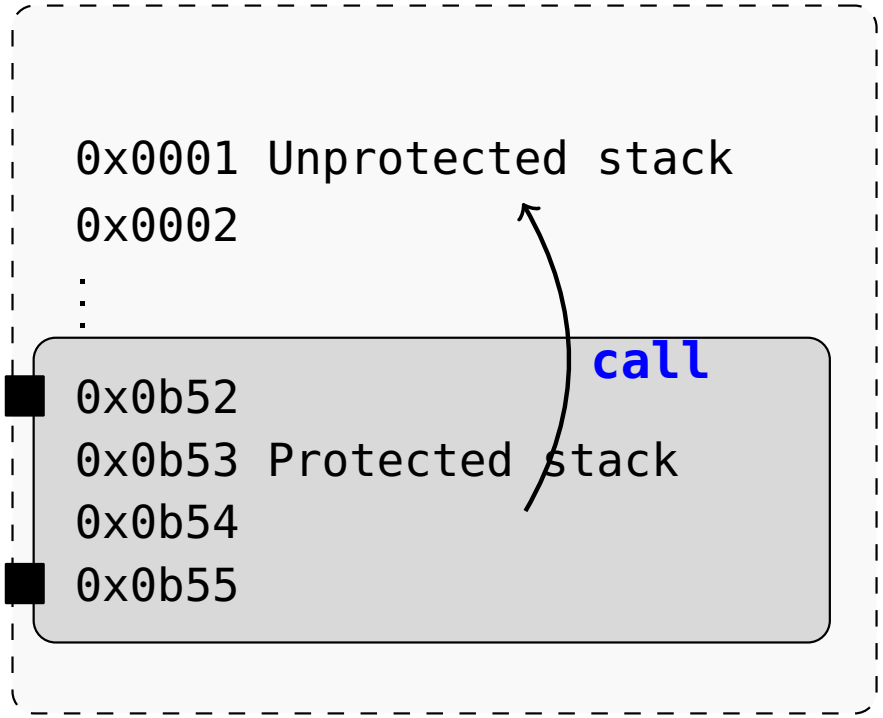
- protected stack
- returnback entry point



Agten's Work

Q: : Is that all?

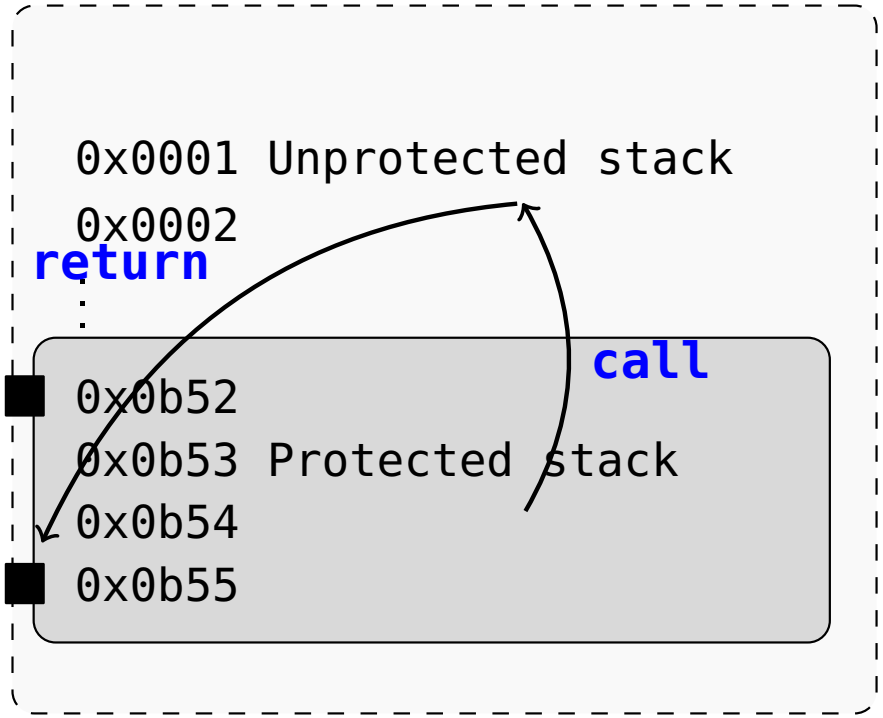
- protected stack
- returnback entry point



Agten's Work

Q: : Is that all?

- protected stack
- returnback entry point



Agten's Work

Q: : Is that all?

- protected stack
- returnback entry point
- reset flags and registers

Agten's Work

Q: : Is that all?

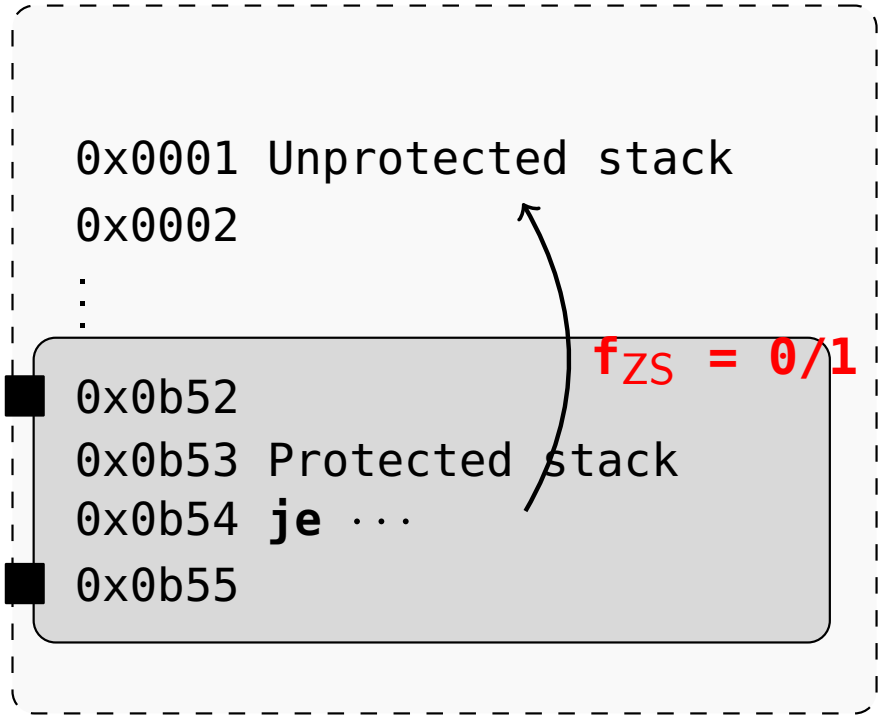
- protected stack
- returnback entry point
- reset flags and registers

```
0x0001 Unprotected stack
0x0002
⋮
0x0b52
0x0b53 Protected stack
0x0b54 je ···
0x0b55
```

Agten's Work

Q: : Is that all?

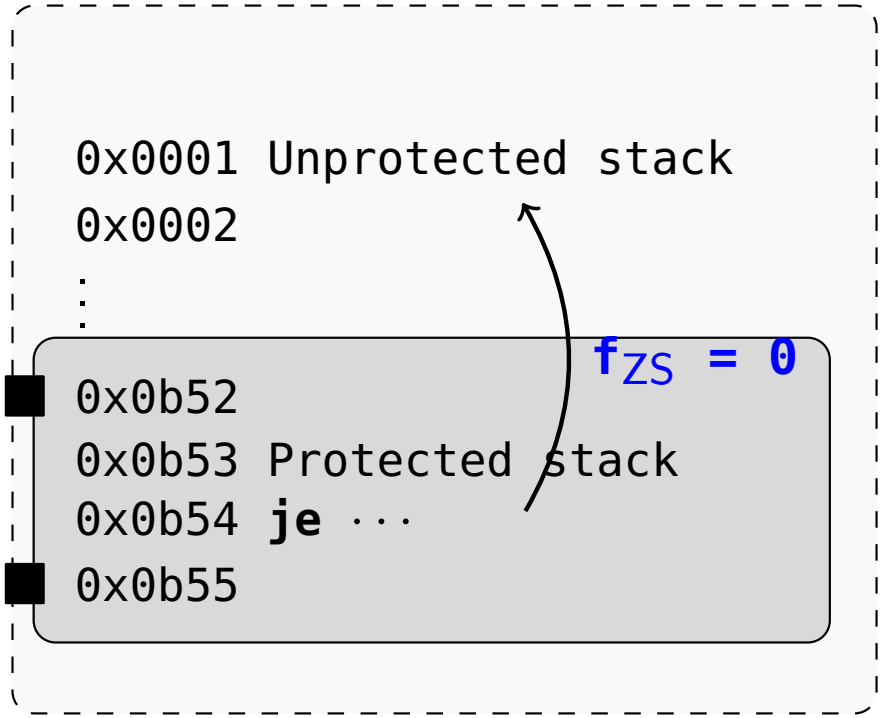
- protected stack
- returnback entry point
- reset flags and registers



Agten's Work

Q: : Is that all?

- protected stack
- returnback entry point
- reset flags and registers



Agten's Work

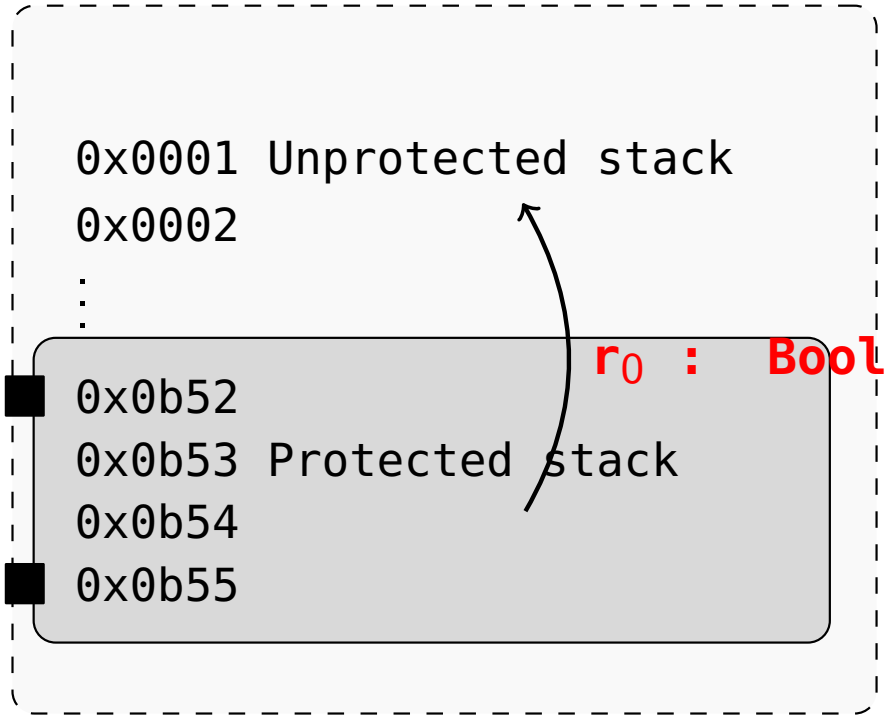
Q: : Is that all?

- protected stack
- returnback entry point
- reset flags and registers
- ground-typed values check

Agten's Work

Q: : Is that all?

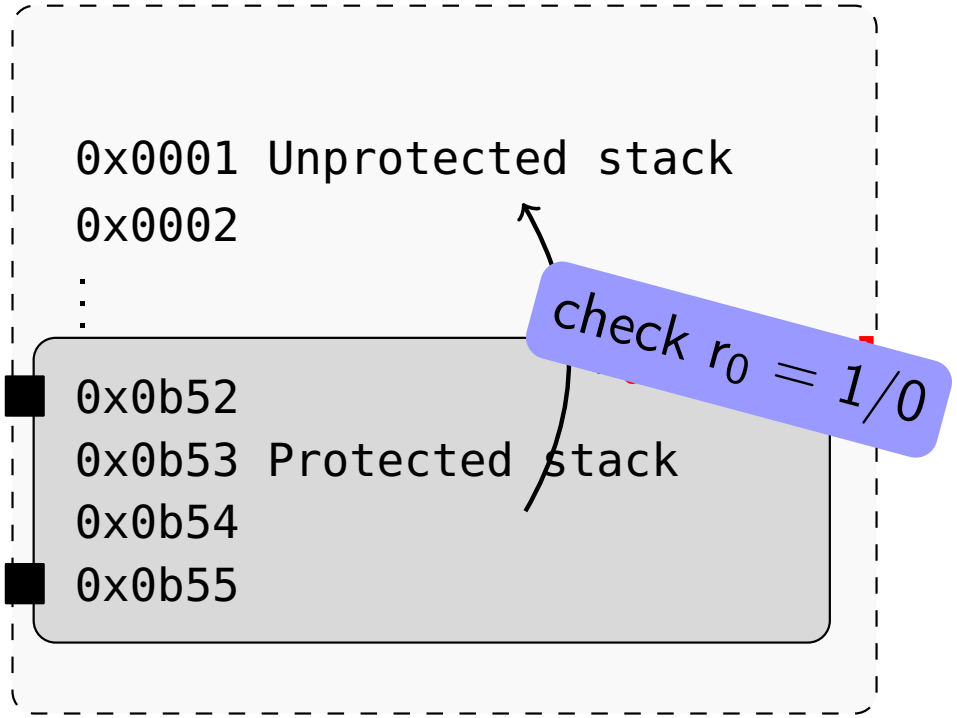
- protected stack
- returnback entry point
- reset flags and registers
- ground-typed values check



Agten's Work

Q: : Is that all?

- protected stack
- returnback entry point
- reset flags and registers
- ground-typed values check

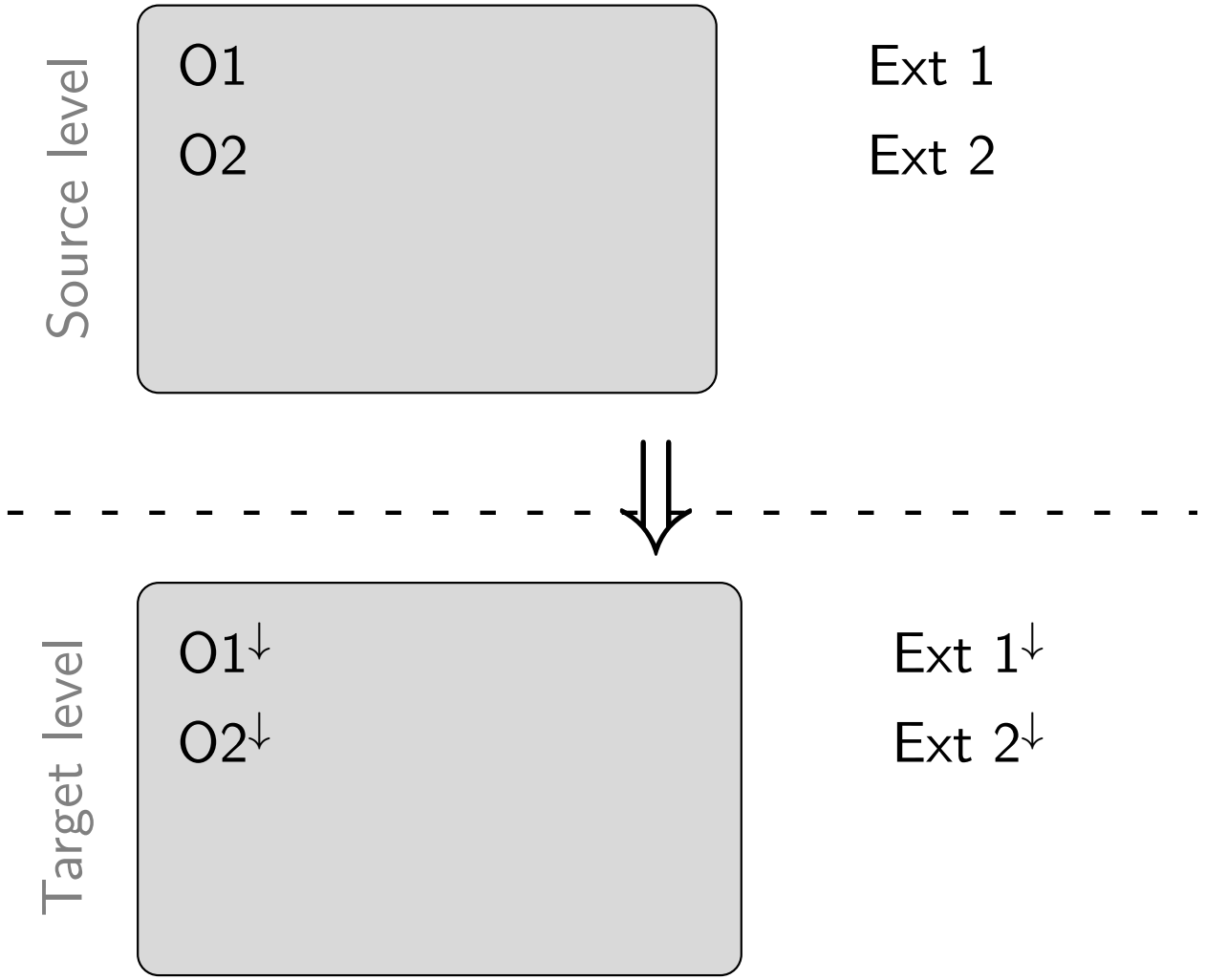


Agten's Work

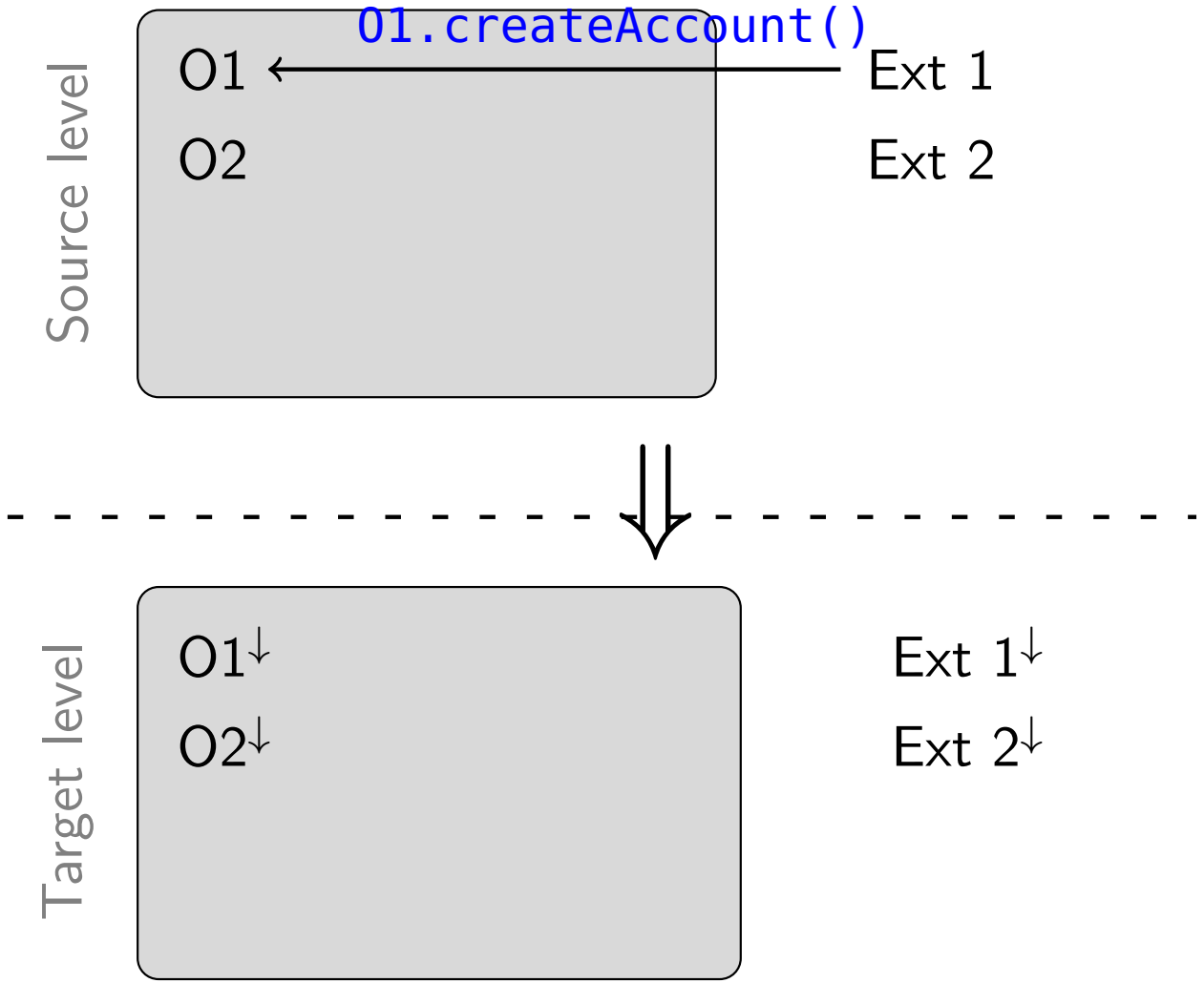
Q: : Is that all?

- protected stack
- returnback entry point
- reset flags and registers
- ground-typed values check
- **Q:** is there more?

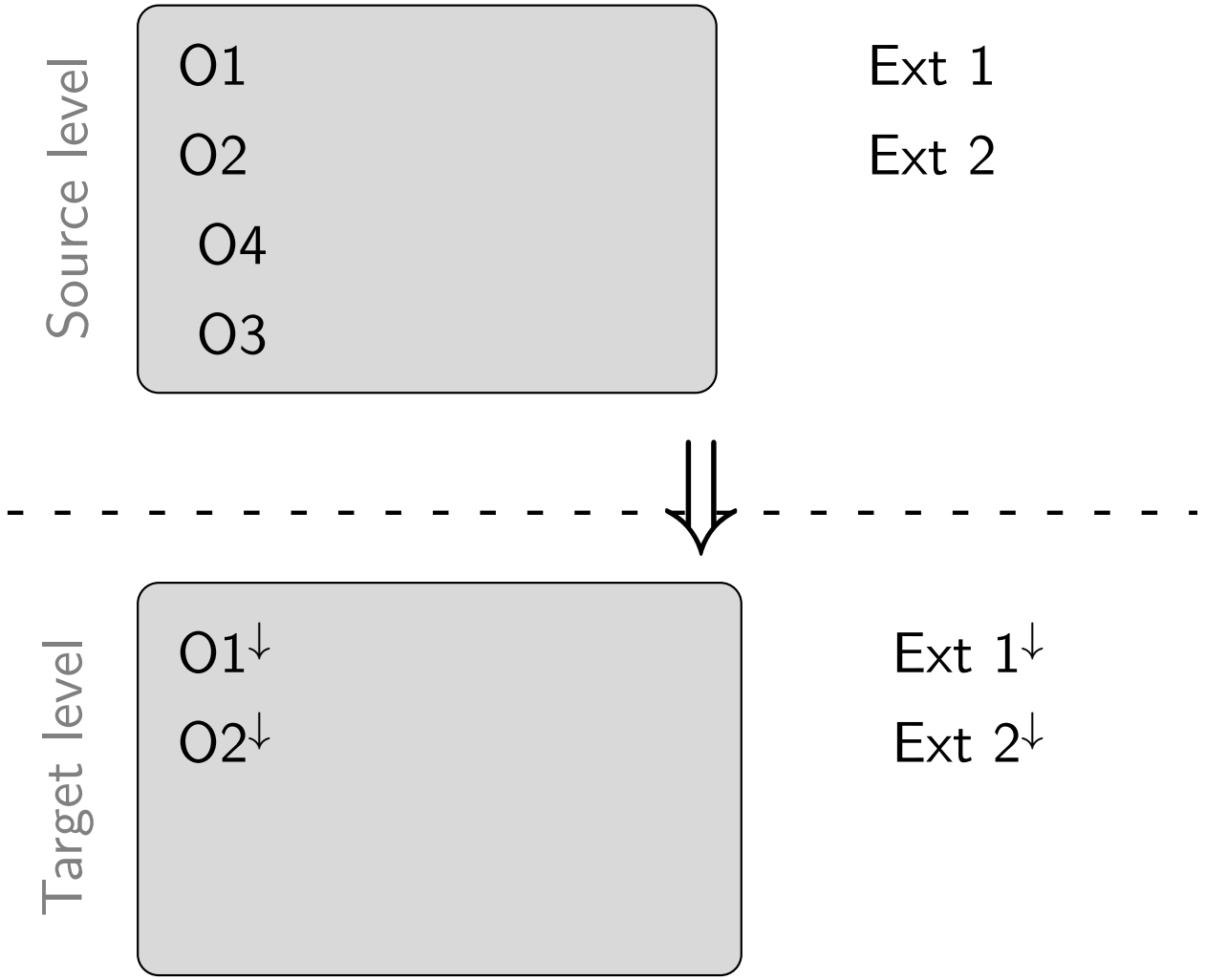
Dynamic Memory Allocation



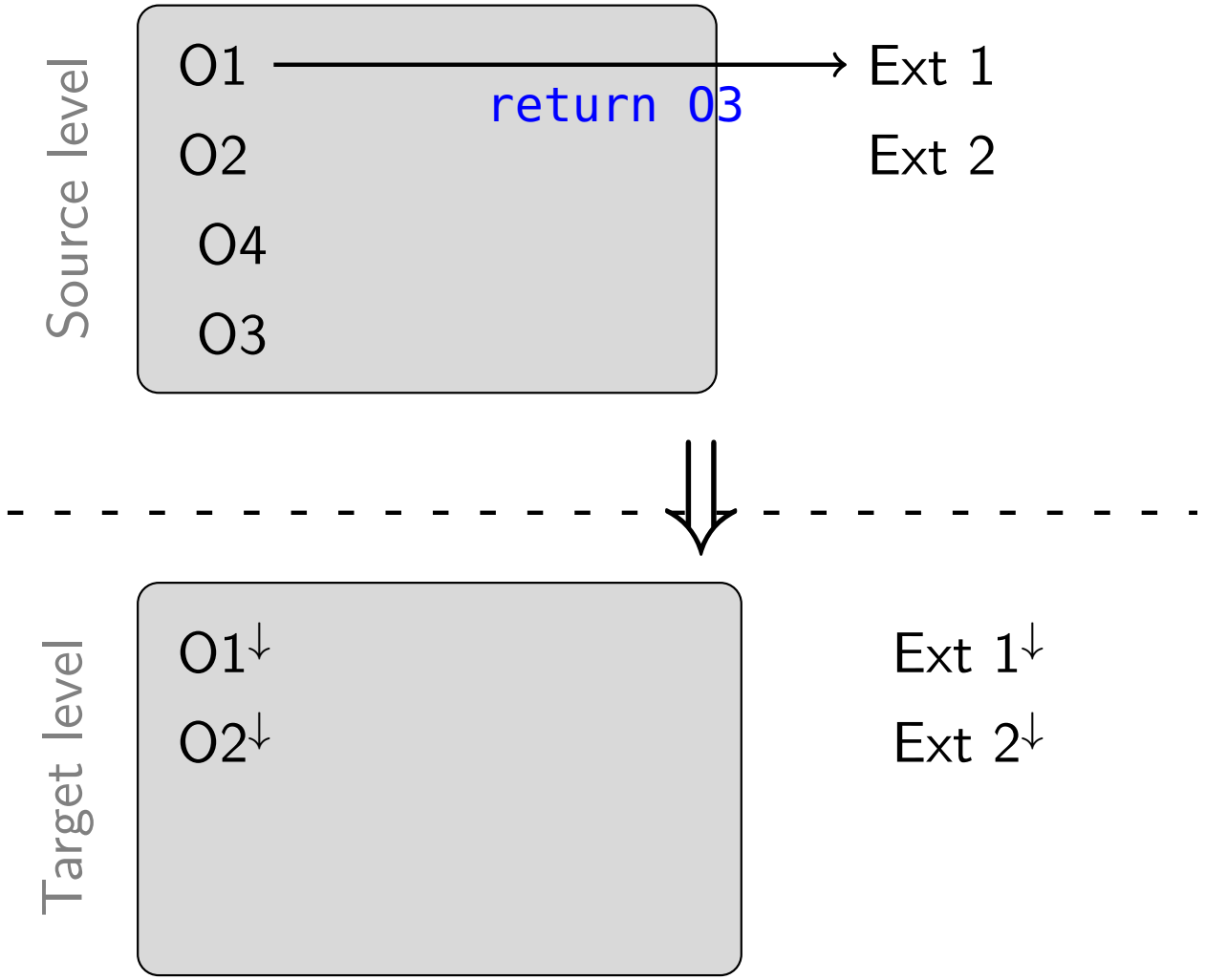
Dynamic Memory Allocation



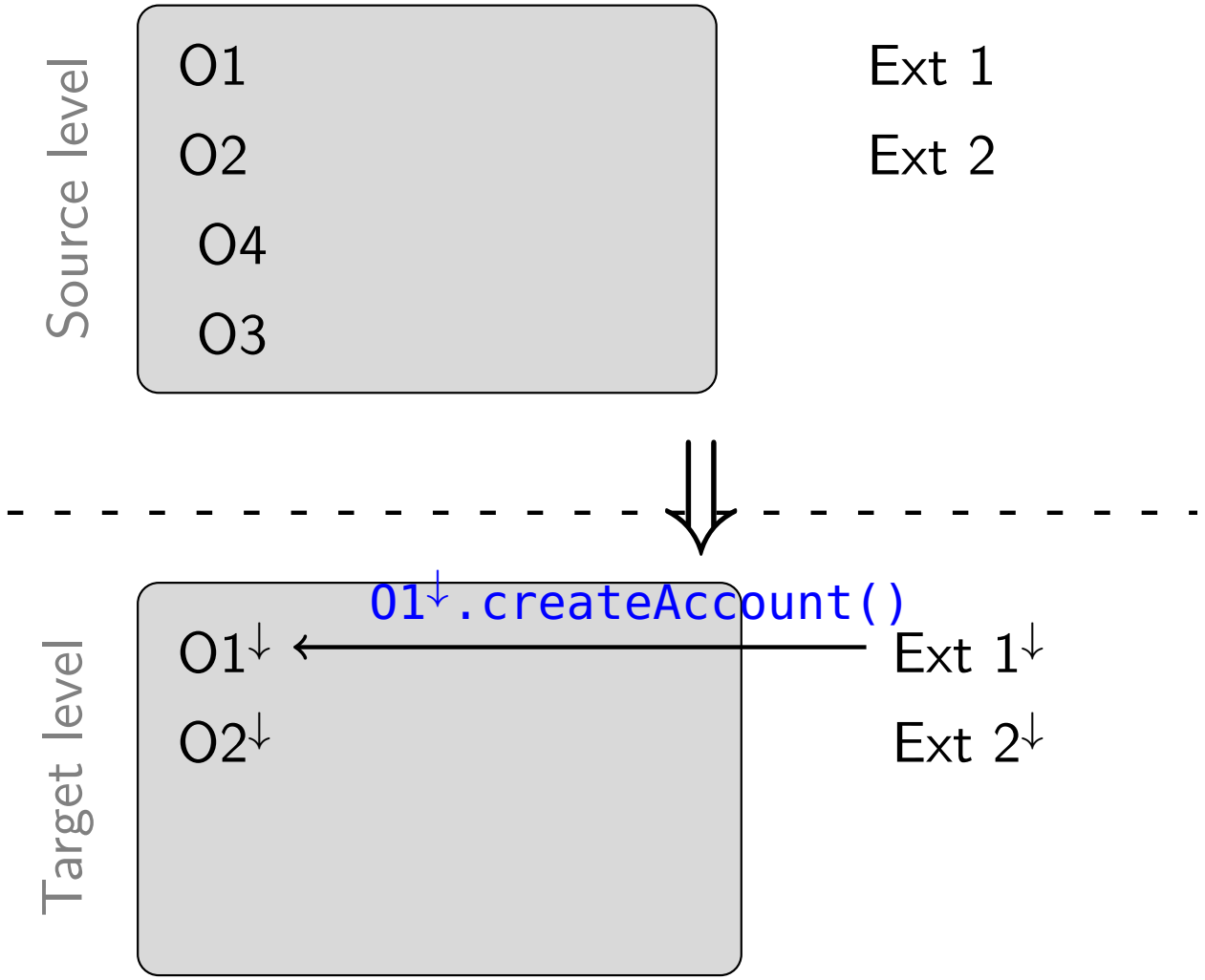
Dynamic Memory Allocation



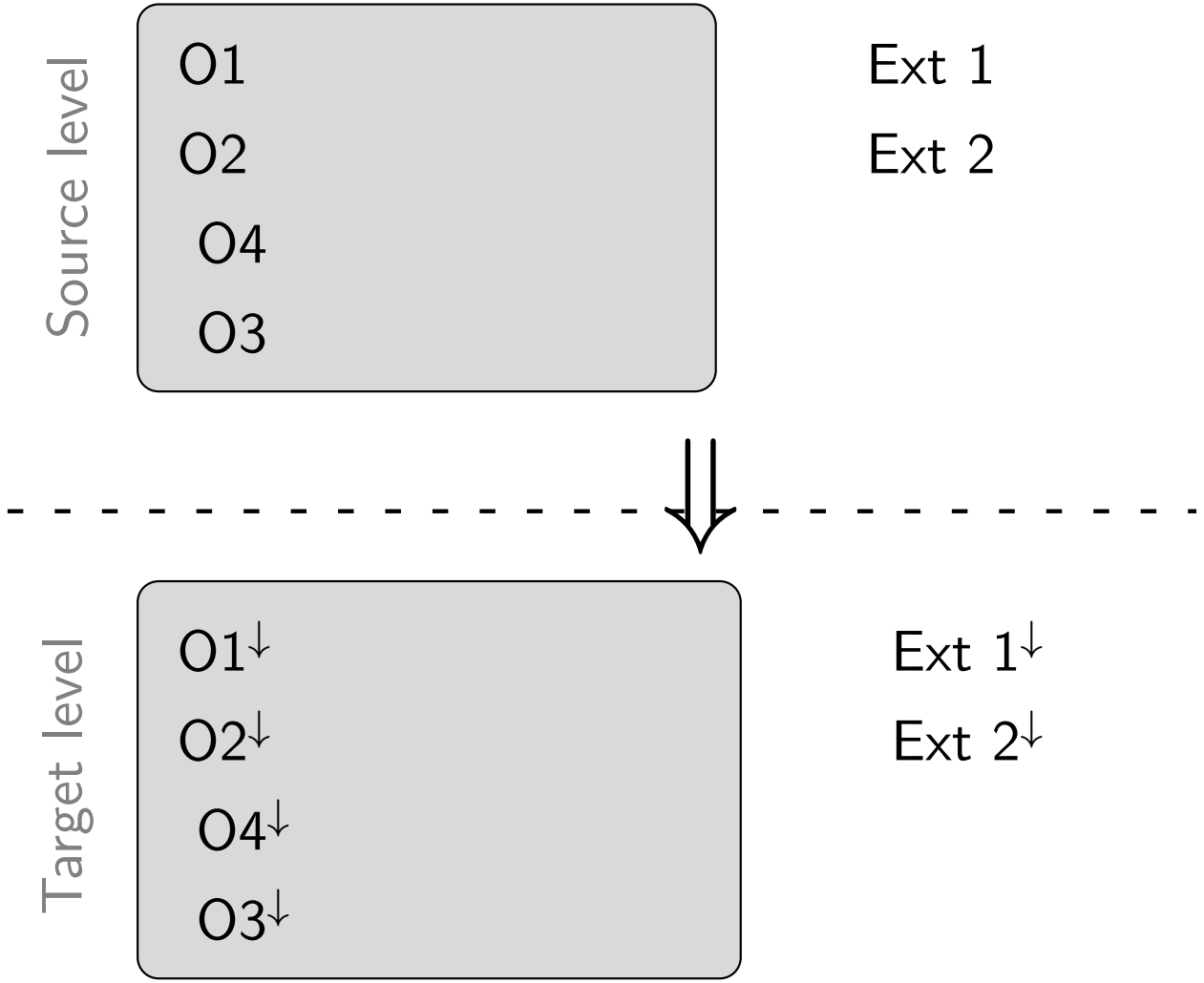
Dynamic Memory Allocation



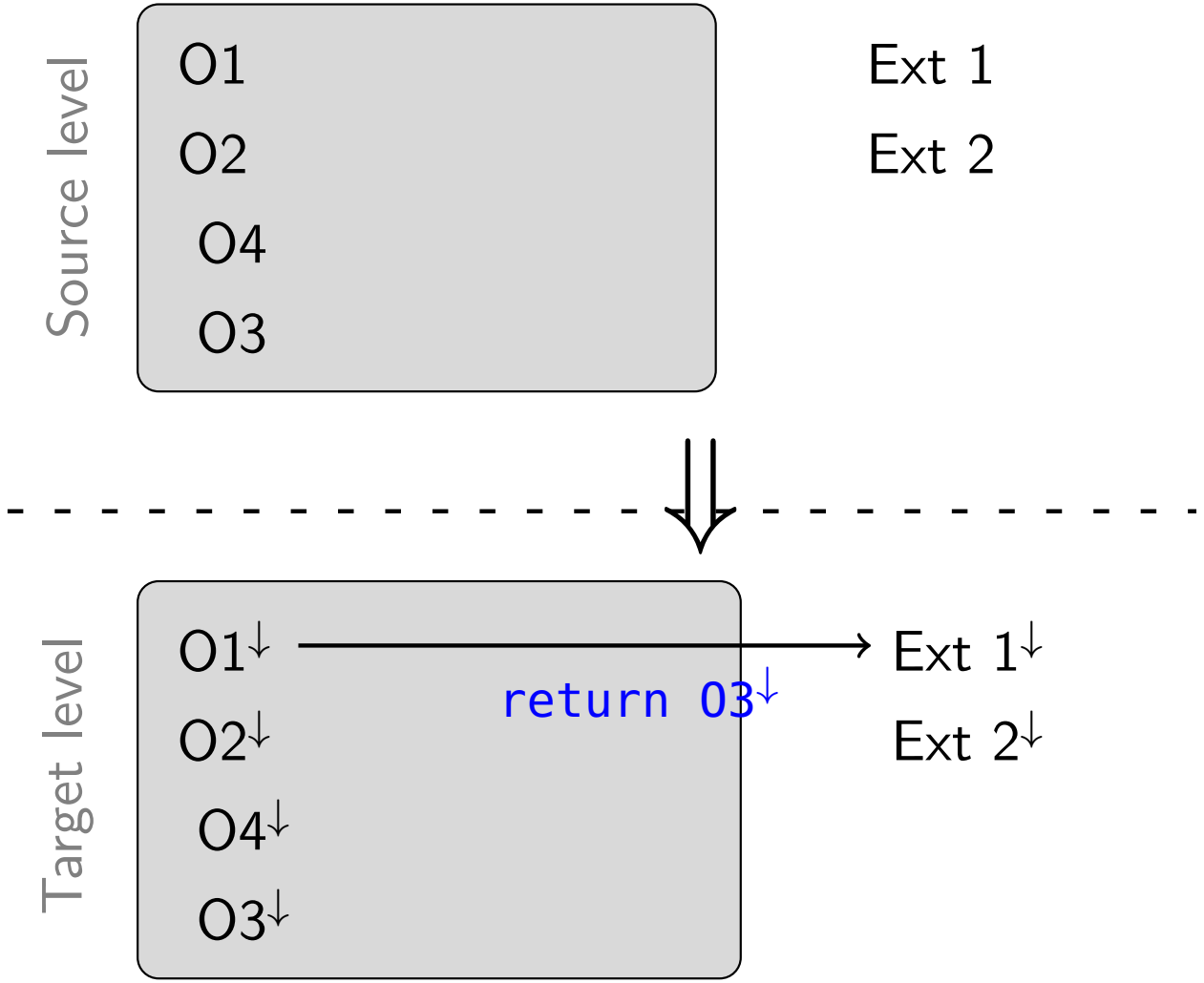
Dynamic Memory Allocation



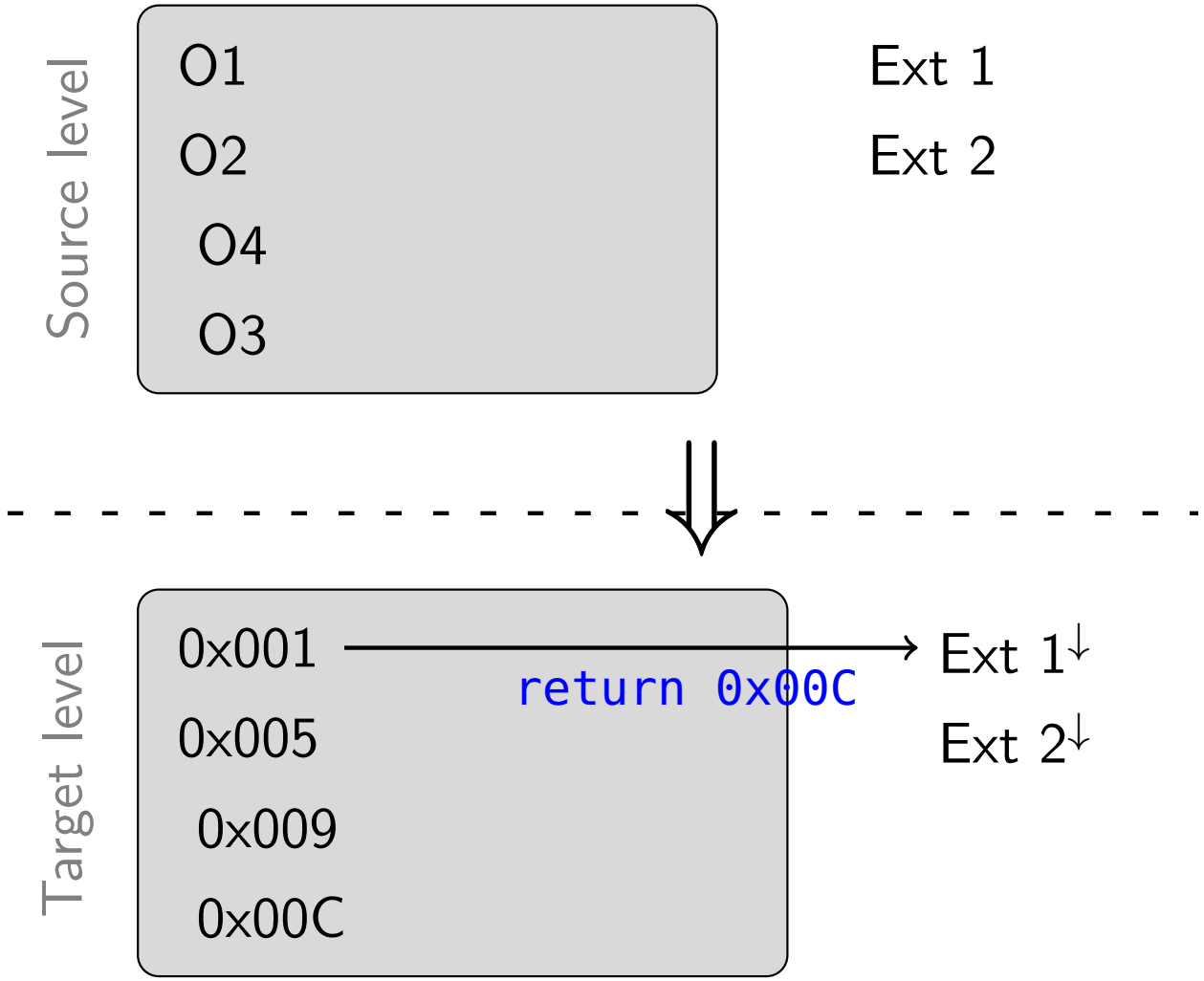
Dynamic Memory Allocation



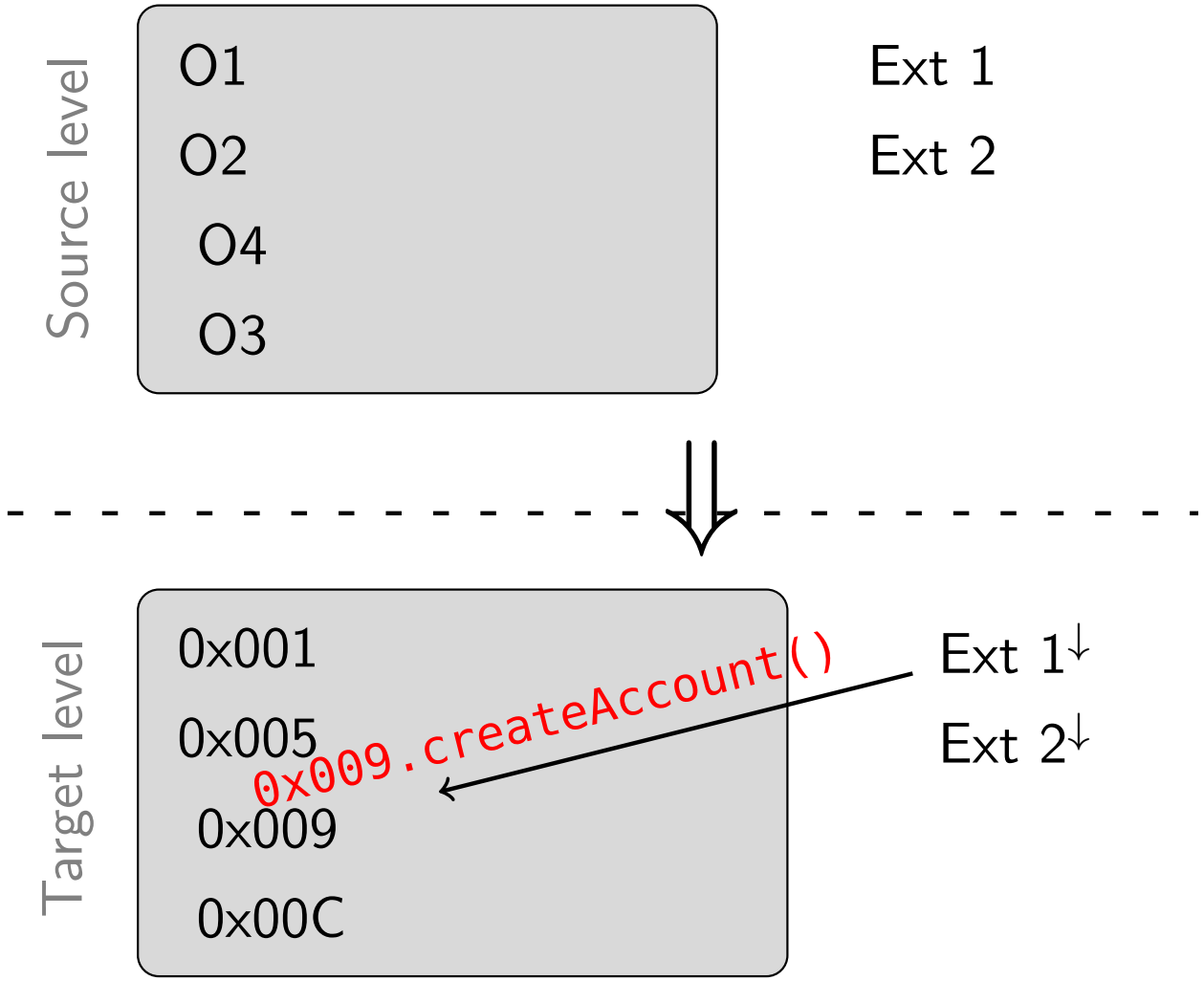
Dynamic Memory Allocation



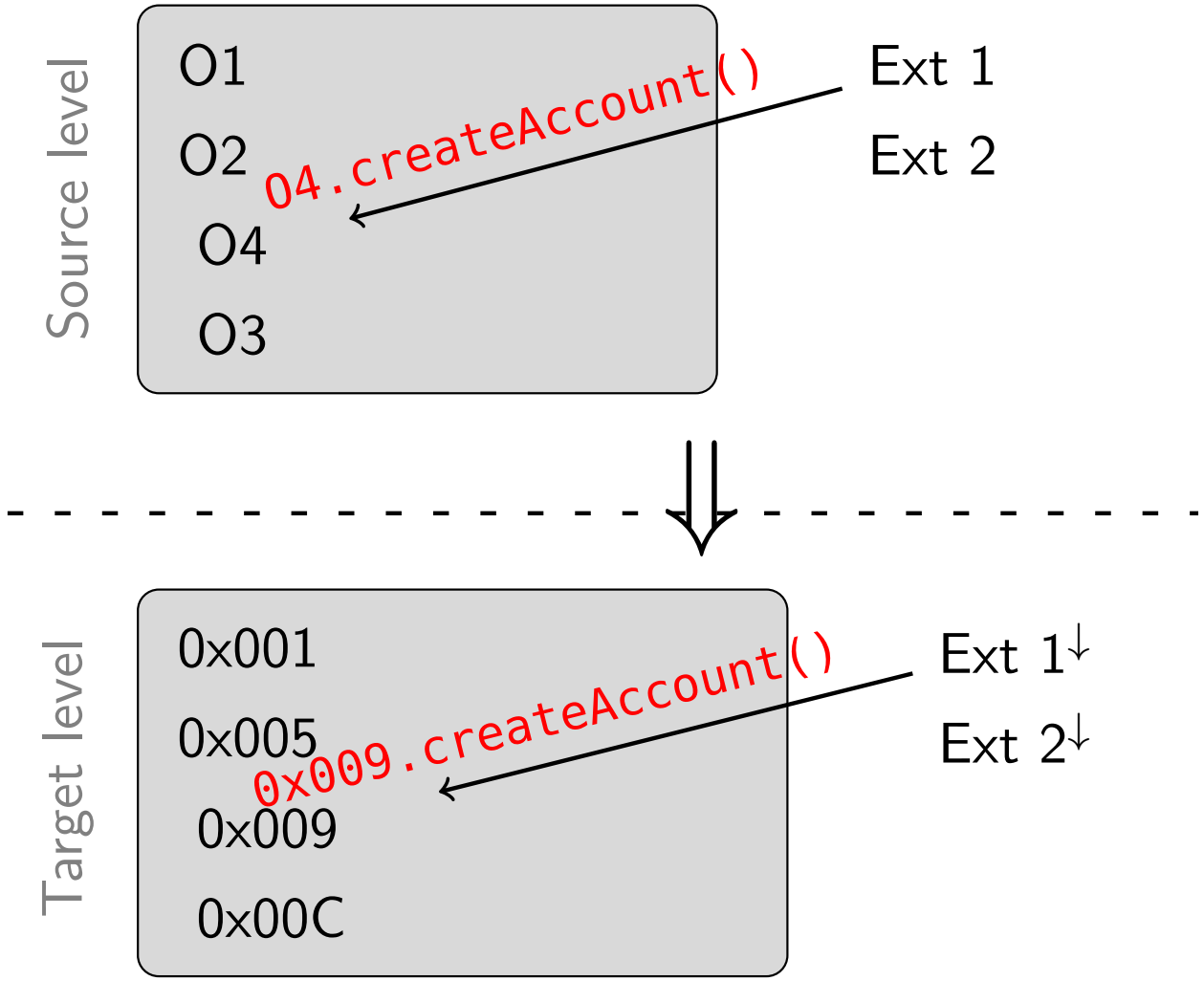
Dynamic Memory Allocation



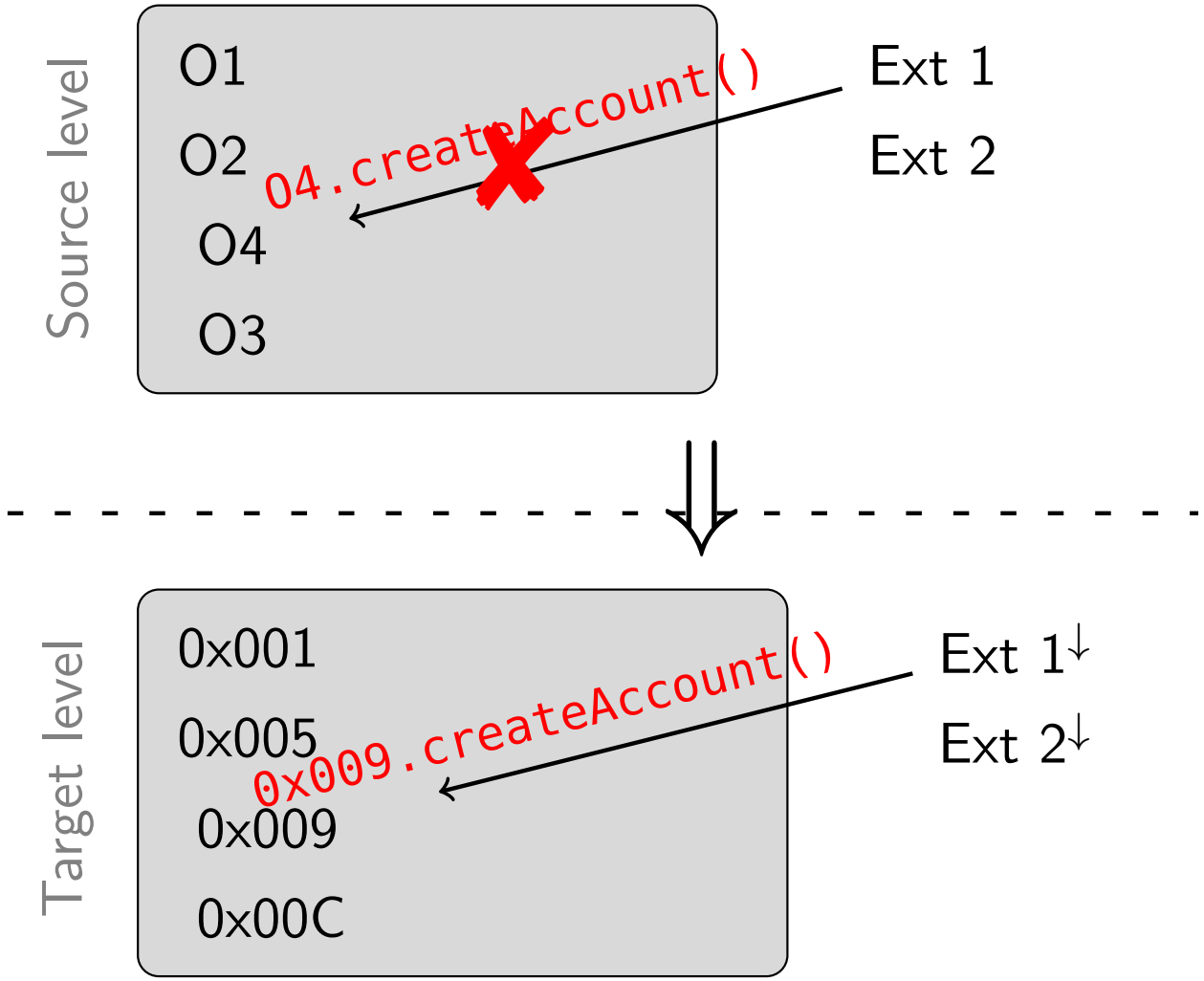
Dynamic Memory Allocation



Dynamic Memory Allocation

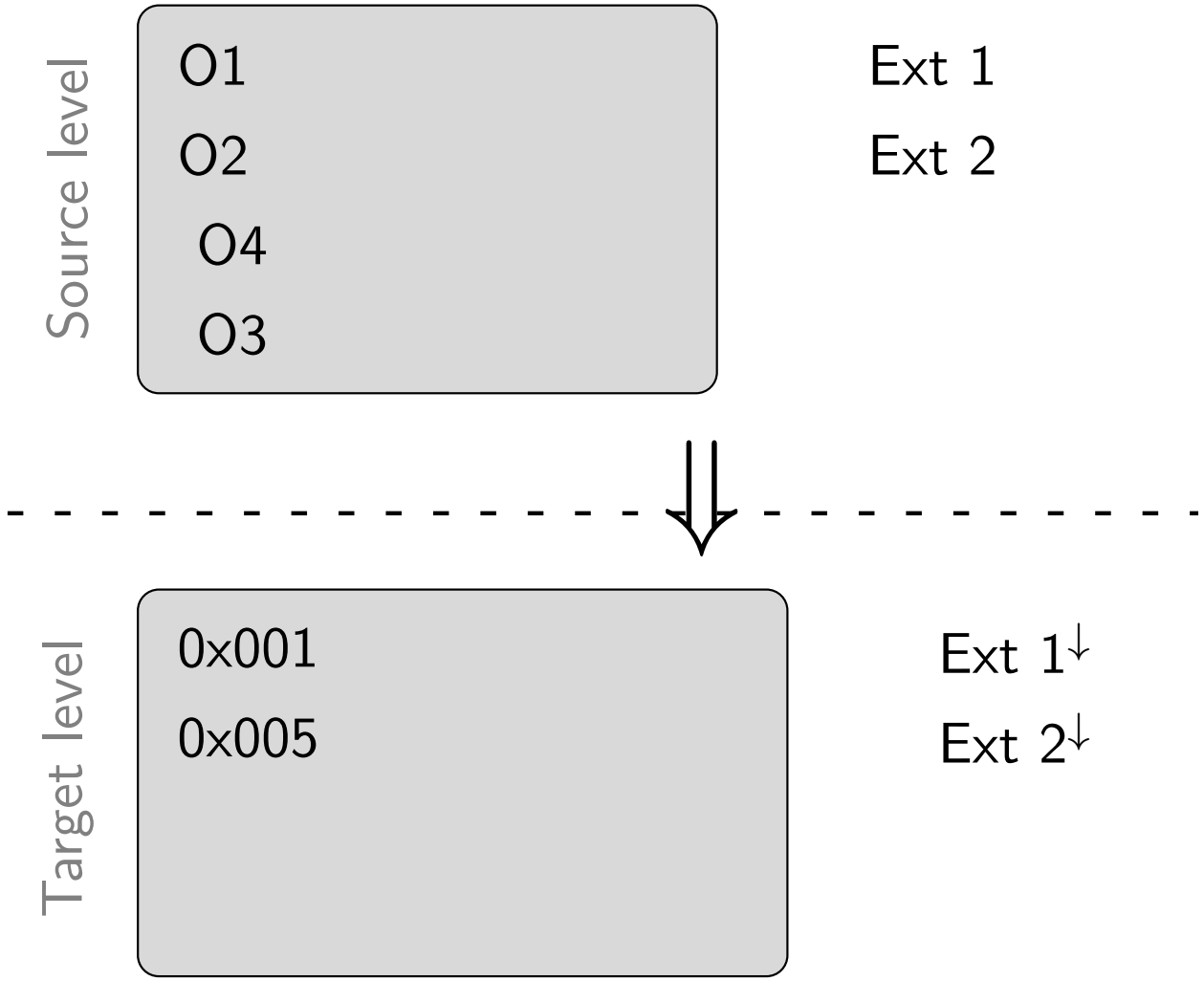


Dynamic Memory Allocation



- Object id guessing

Dynamic Memory Allocation



- Object id guessing
- map Oid to natural numbers

Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Ext 1
Ext 2

- Object id guessing
- map Oid to natural numbers

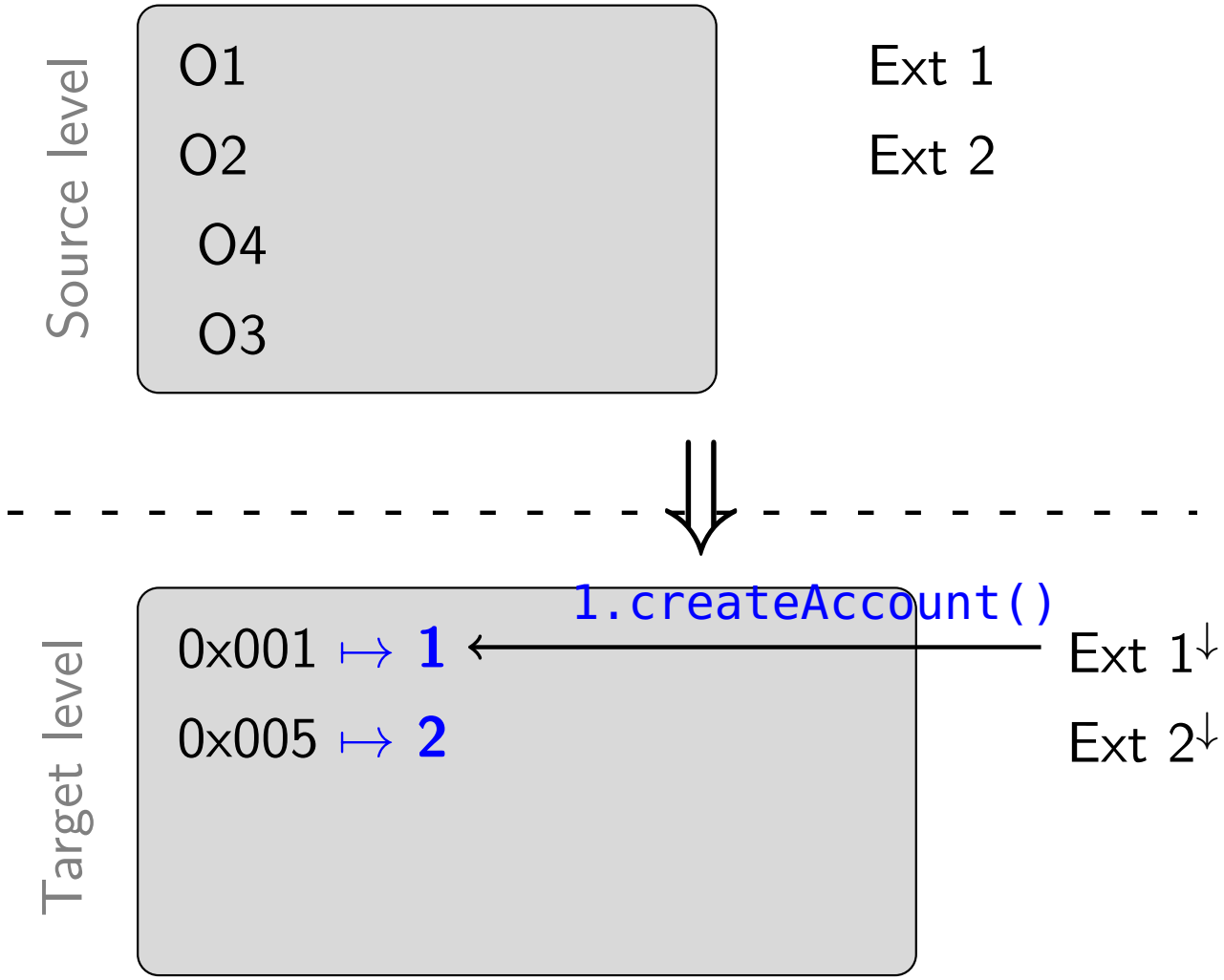
$\Downarrow$

Target level

0x001 $\mapsto$ **1**
0x005 $\mapsto$ **2**

Ext $1^{\downarrow}$
Ext $2^{\downarrow}$

Dynamic Memory Allocation



- Object id guessing
- map Oid to natural numbers

Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Ext 1
Ext 2

- Object id guessing
- map Oid to natural numbers

$\Downarrow$

Target level

0x001 $\mapsto$ **1**
0x005 $\mapsto$ **2**
0x009
0x00C

Ext $1^{\downarrow}$
Ext $2^{\downarrow}$

Dynamic Memory Allocation

Source level

O1
O2
O4
O3

Ext 1
Ext 2

$\Downarrow$

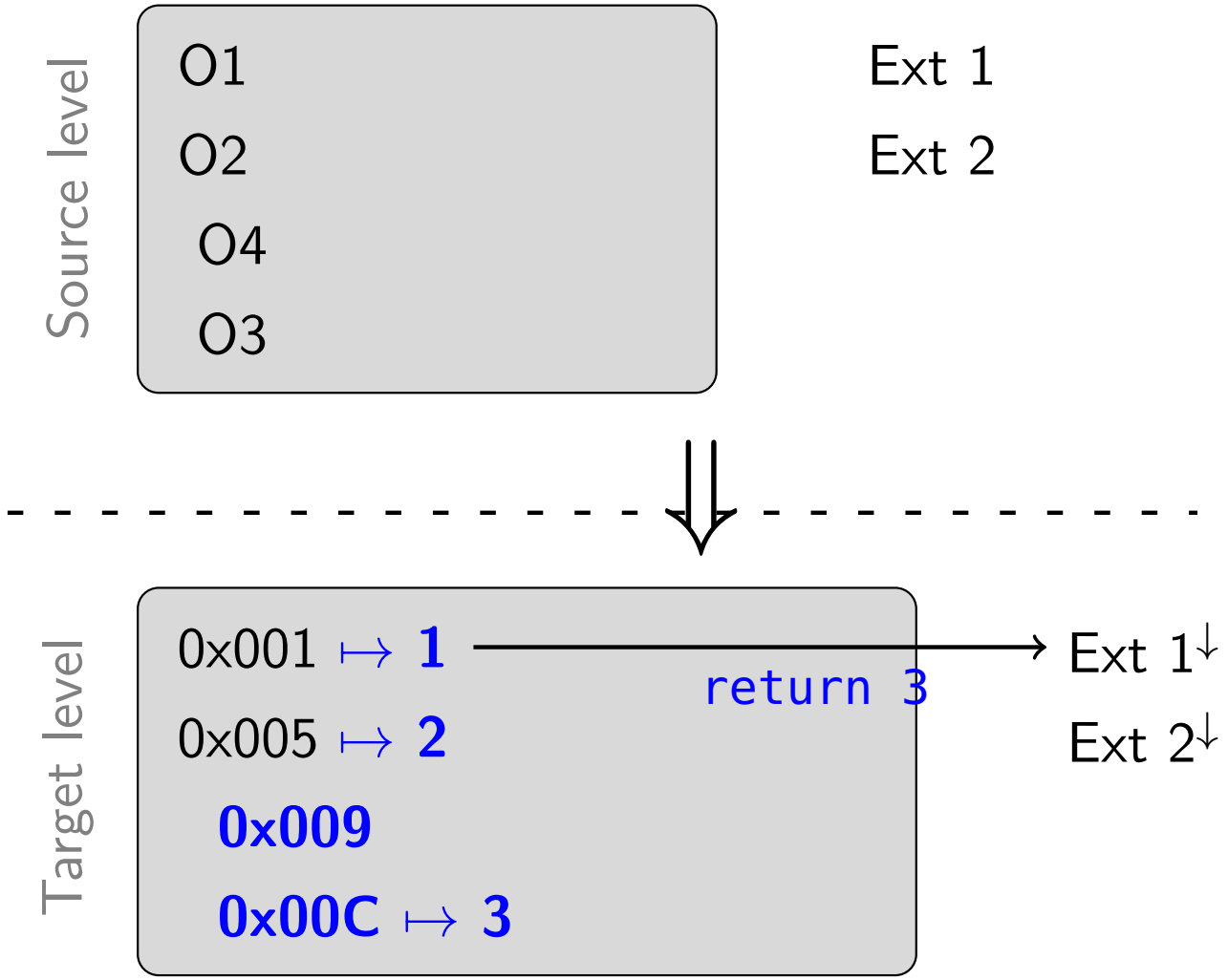
Target level

0x001 $\mapsto$ **1**
0x005 $\mapsto$ **2**
0x009
0x00C $\mapsto$ **3**

Ext $1^{\downarrow}$
Ext $2^{\downarrow}$

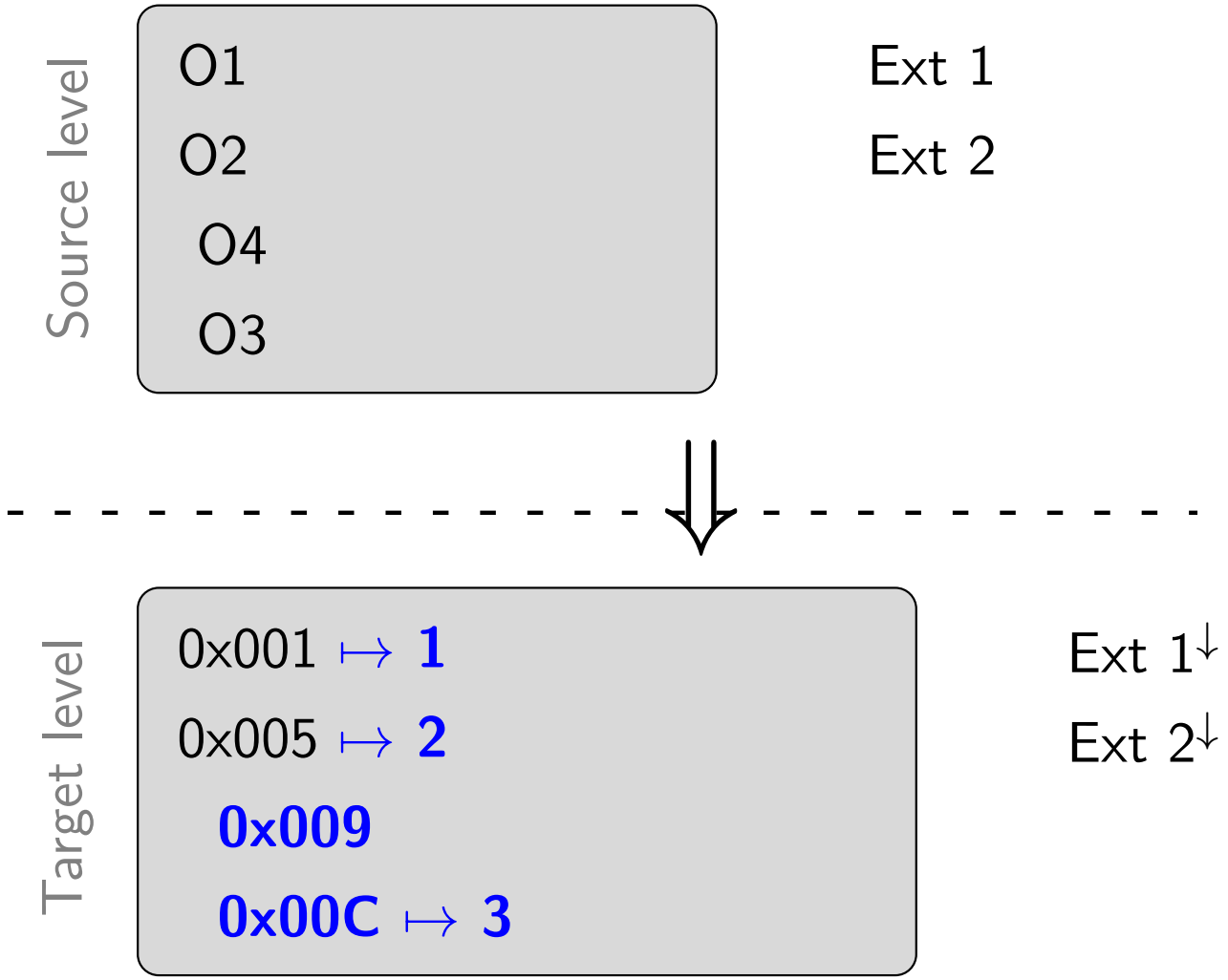
- Object id guessing
- map Oid to natural numbers
- add Oid to map

Dynamic Memory Allocation



- Object id guessing
- map Oid to natural numbers
- add Oid to map

Dynamic Memory Allocation



- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup ($\mathbf{O}(1)$) when number is received

Dynamic Memory Allocation

Source level

O1 : Account
O2 : Pair
O4
O3

Ext 1
Ext 2

- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup ($\mathbf{O}(1)$) when number is received

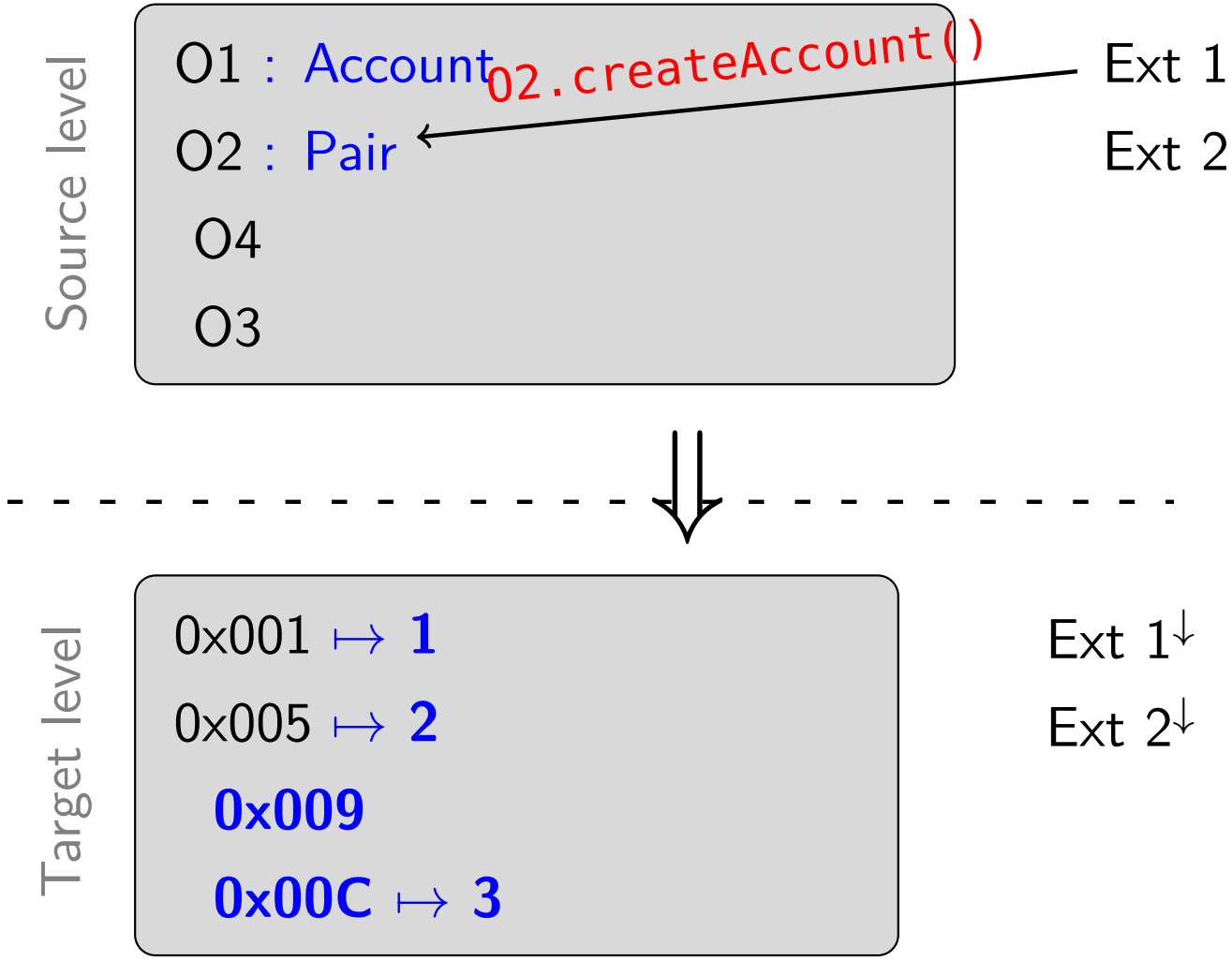
$\Downarrow$

Target level

0x001 $\mapsto$ **1**
0x005 $\mapsto$ **2**
0x009
0x00C $\mapsto$ **3**

Ext $1^{\downarrow}$
Ext $2^{\downarrow}$

Dynamic Memory Allocation



- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup ($\mathbf{O}(1)$) when number is received

Dynamic Memory Allocation

Source level

O1 : Account
O2 : Pair
O4
O3

O2.createAccount()

Ext 1
Ext 2

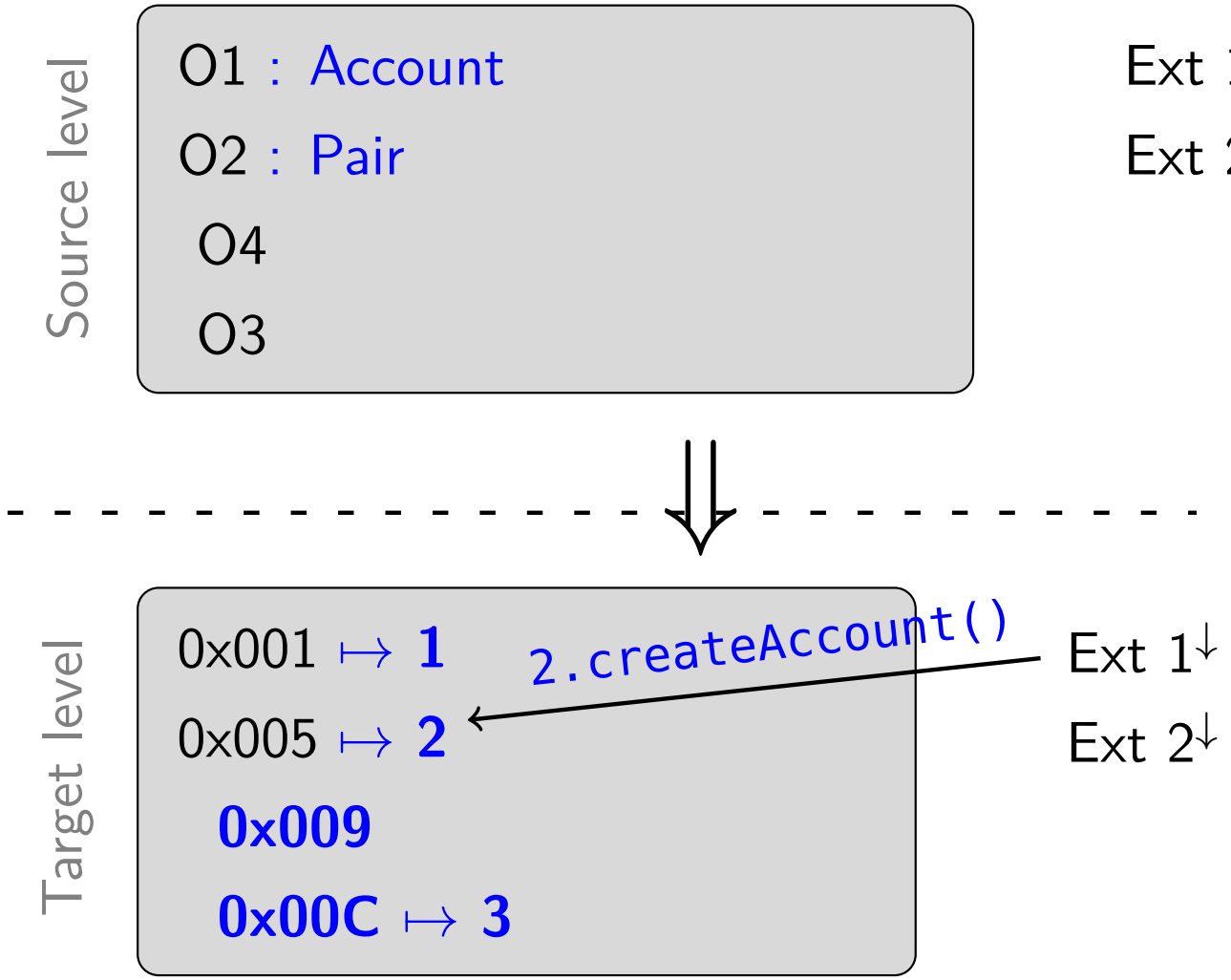
- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup ($\mathbf{O}(1)$) when number is received

Target level

0x001 $\mapsto$ **1**
0x005 $\mapsto$ **2**
0x009
0x00C $\mapsto$ **3**

Ext $1^{\downarrow}$
Ext $2^{\downarrow}$

Dynamic Memory Allocation



- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup ($\mathbf{O}(1)$) when number is received

Dynamic Memory Allocation

Source level

O1 : Account
O2 : Pair
O4
O3

Target level

0x001 ↦ **1**
0x005 ↦ **2**
0x009
0x00C ↦ 3

Ext 1
Ext 2

Ext $1^{\downarrow}$
Ext $2^{\downarrow}$

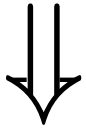
- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup ($\mathbf{O}(1)$) when number is received
- dynamic typecheck for: current object

Dynamic Memory Allocation

Source level

O1 : Account
O2 : Pair
O4
O3

Ext 1
Ext 2



Target level

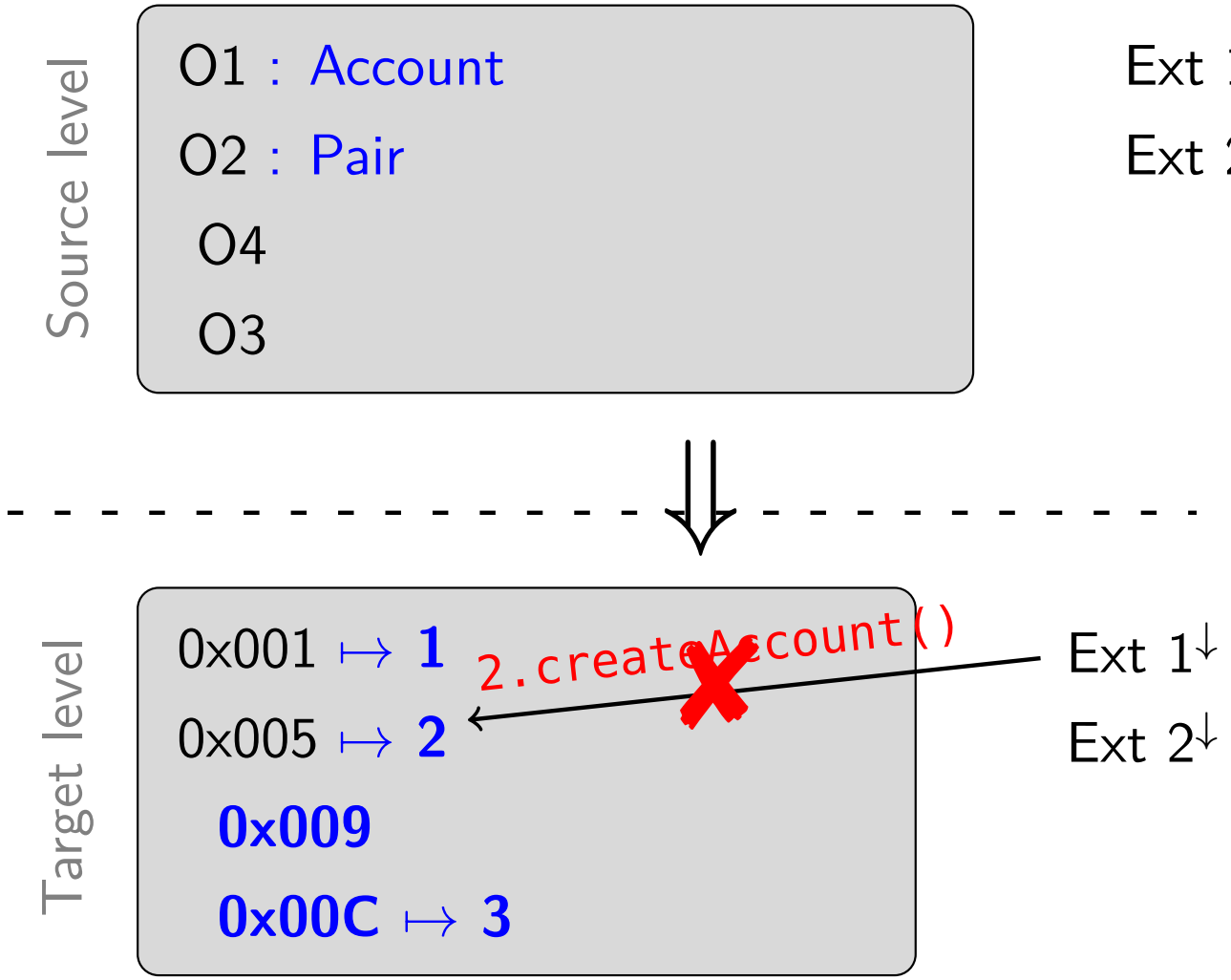
0x001 ↦ **1**
0x005 ↦ **2**
0x009
0x00C ↦ 3

2.createAccount()

Ext 1↓
Ext 2↓

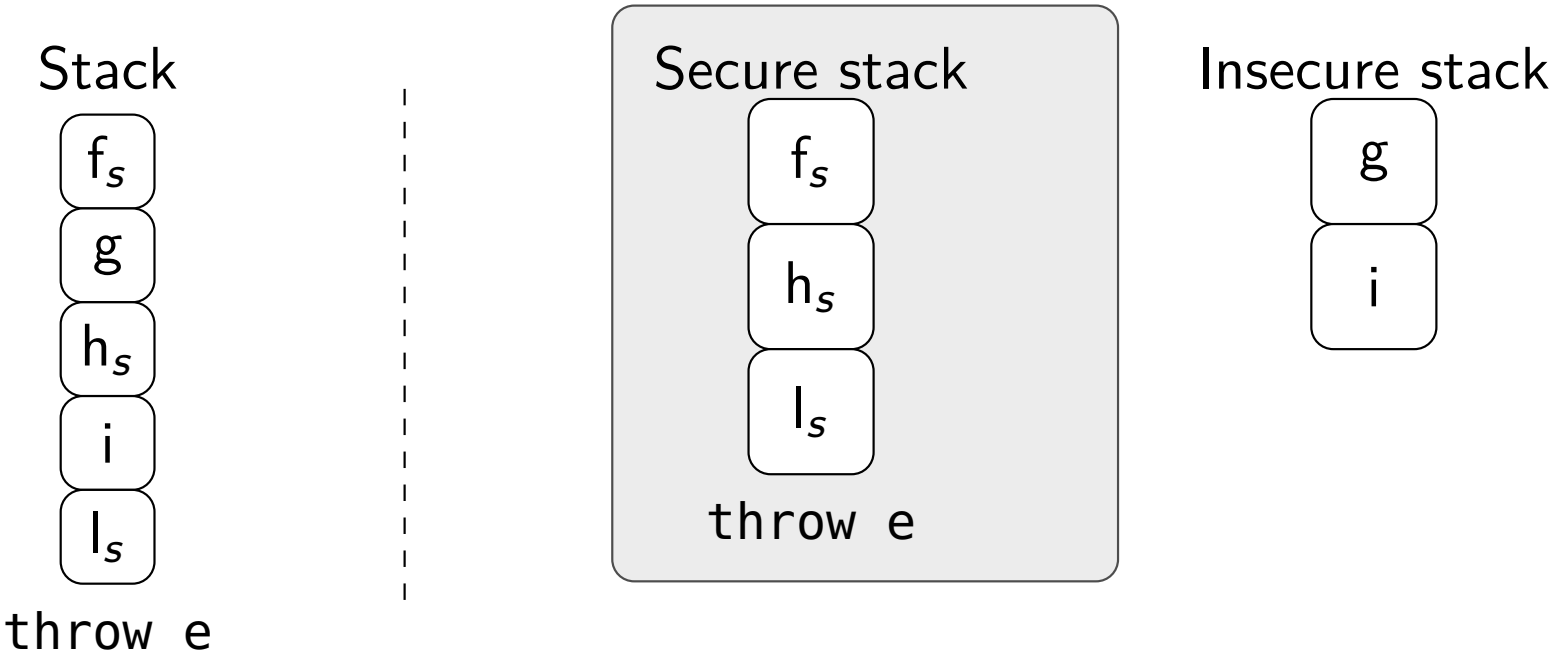
- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup ($\mathbf{O}(1)$) when number is received
- dynamic typecheck for: current object arguments

Dynamic Memory Allocation

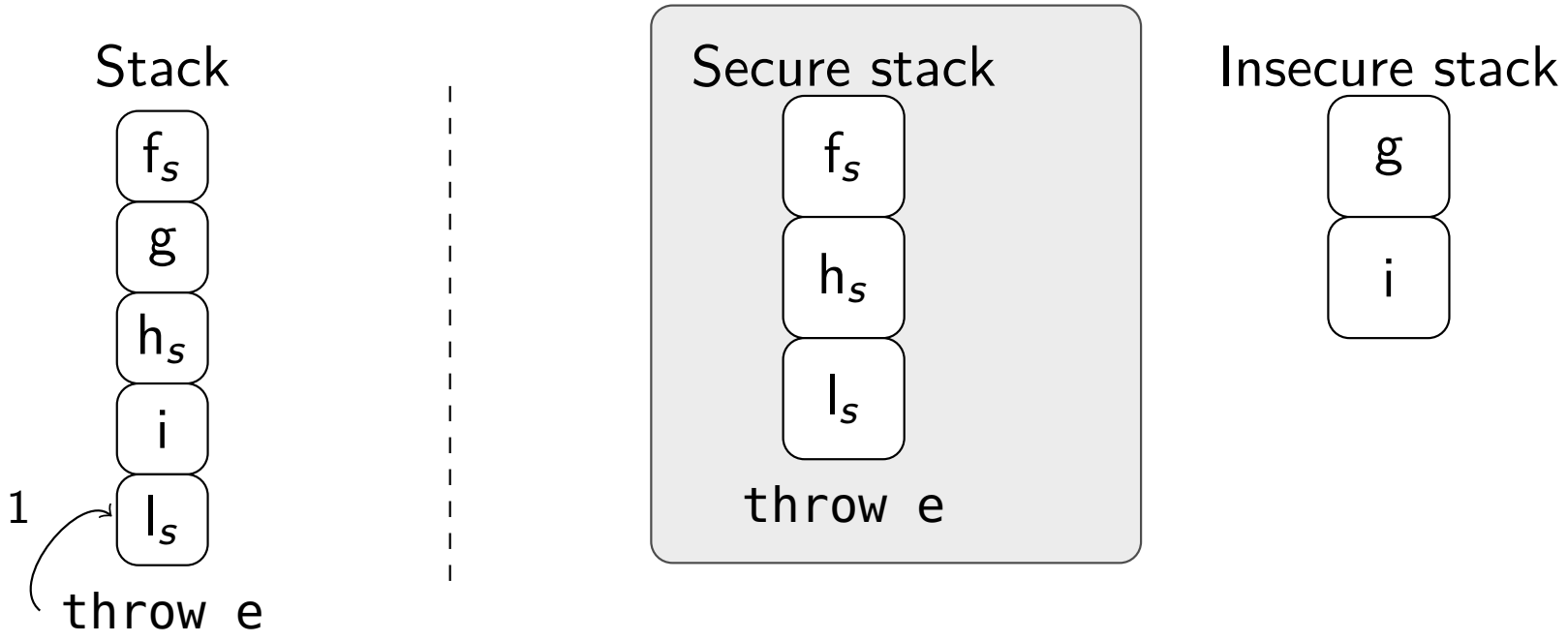


- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup ($\mathbf{O}(1)$) when number is received
- dynamic typecheck for: current object arguments
- no need of extra information

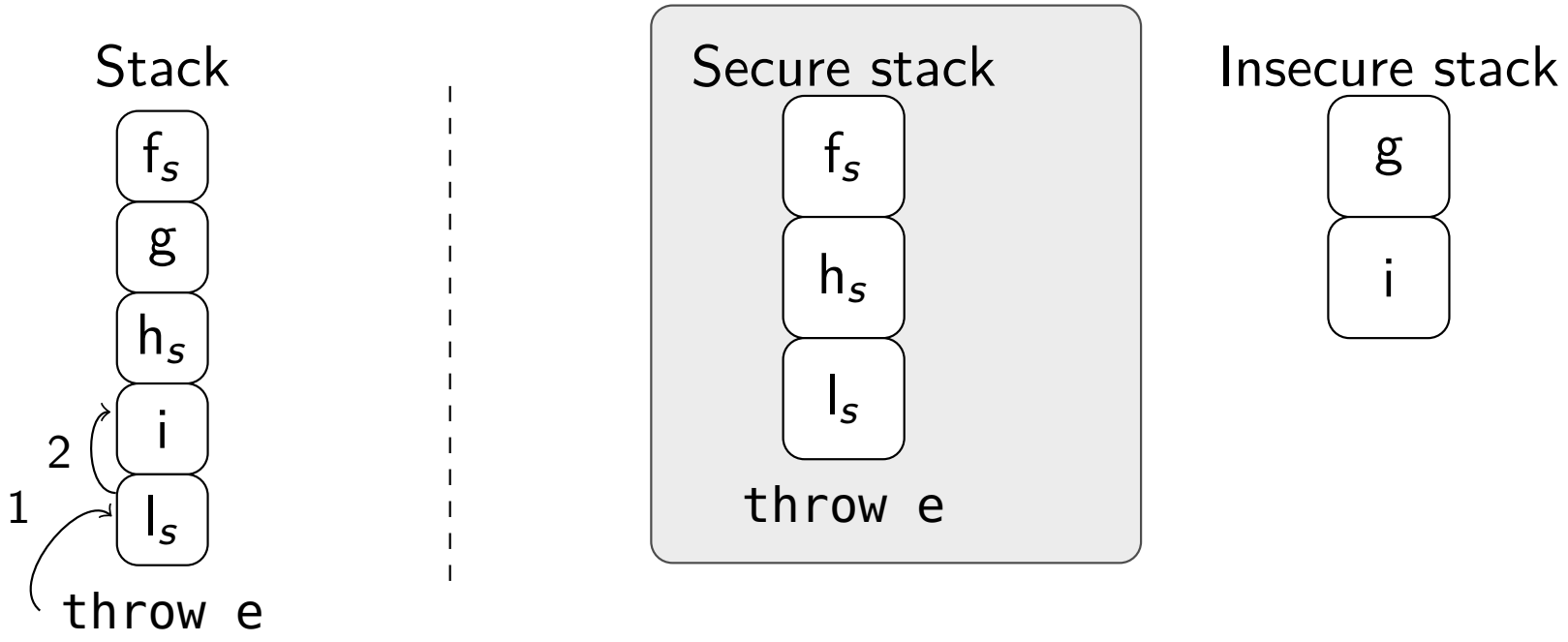
Exceptions


Stack
f_s
g
h_s
i
l_s
throw e
Secure stack
f_s
h_s
l_s
throw e
Insecure stack
g
i

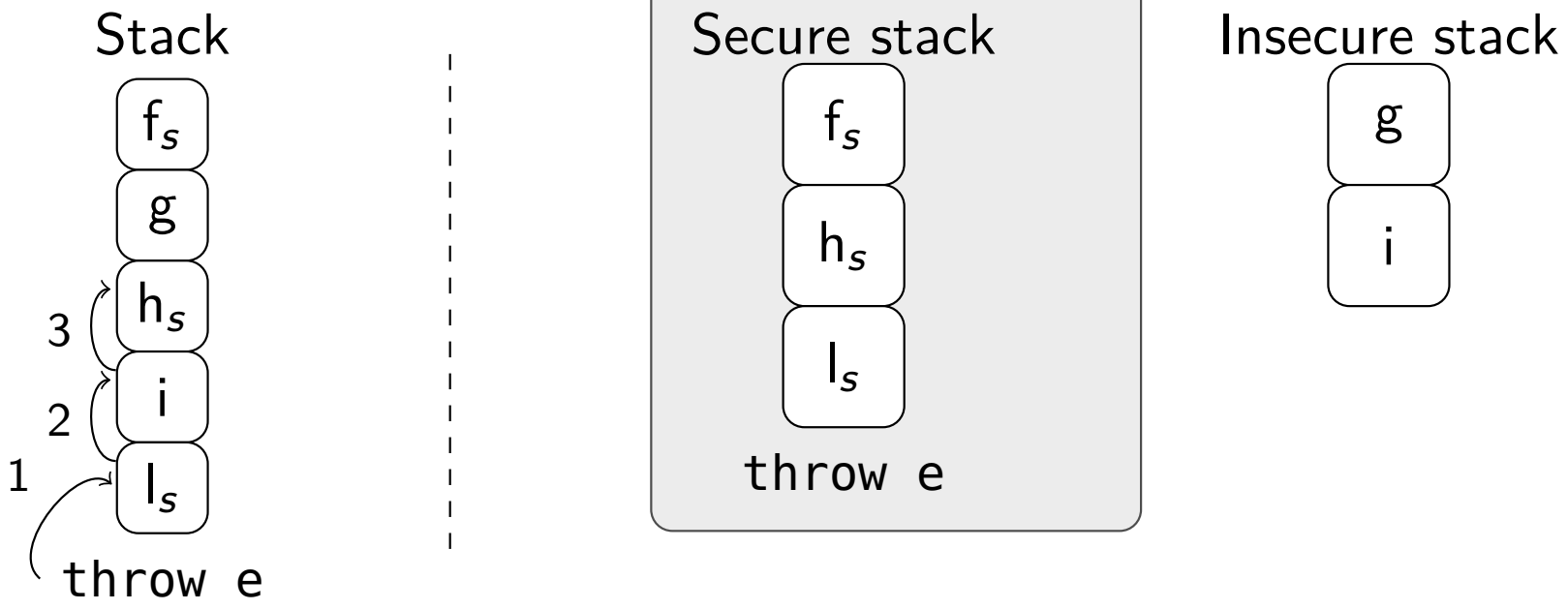
Exceptions


Stack
f_s
g
h_s
i
1
l_s
throw e
Secure stack
f_s
h_s
l_s
throw e
Insecure stack
g
i

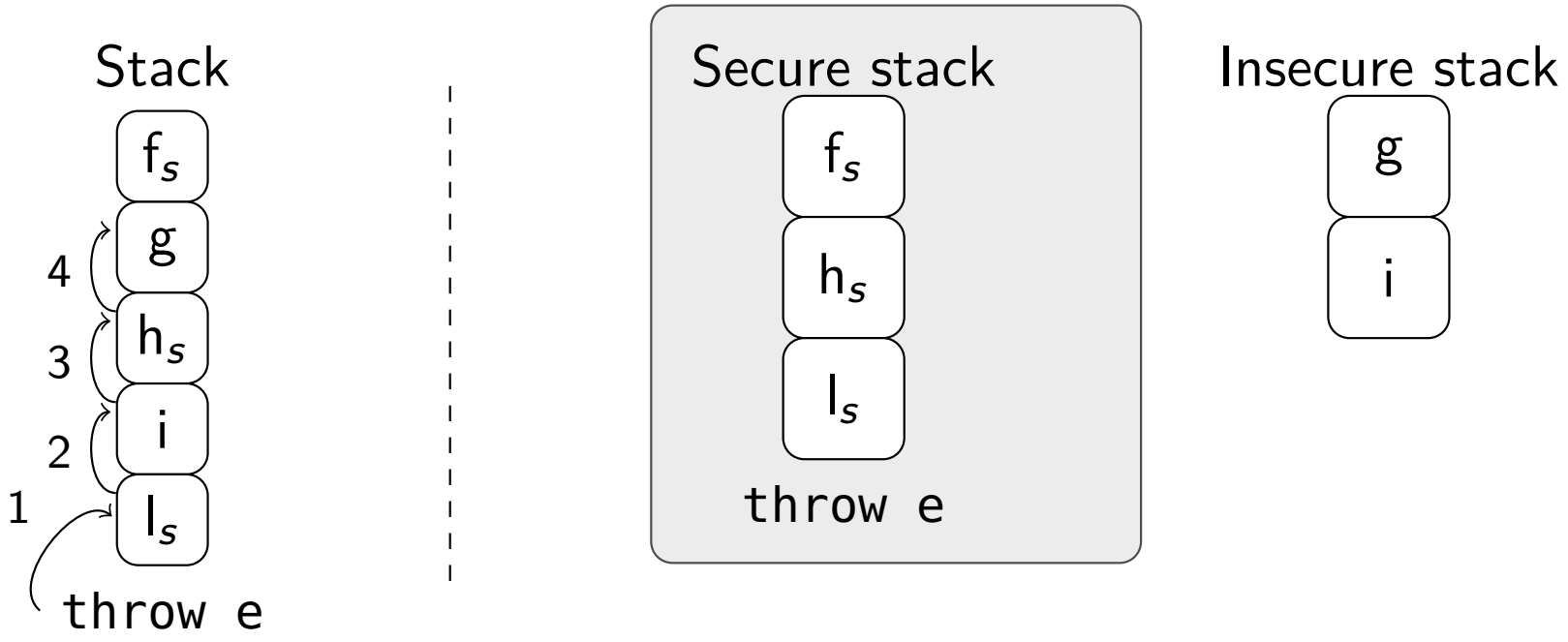
Exceptions


Stack
f_s
g
h_s
i
l_s
2
1
throw e
Secure stack
f_s
h_s
l_s
throw e
Insecure stack
g
i

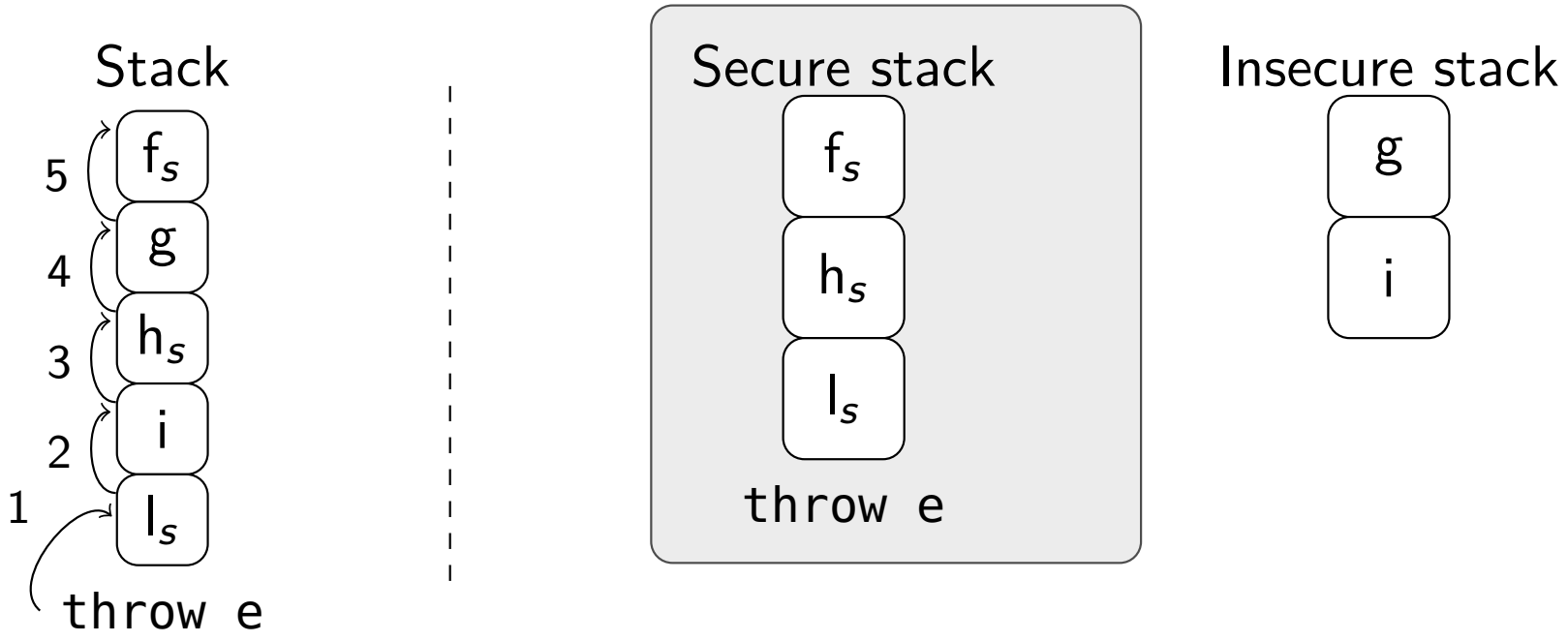
Exceptions


Stack
f_s
g
h_s
i
l_s
3
2
1
throw e
Secure stack
f_s
h_s
l_s
throw e
Insecure stack
g
i

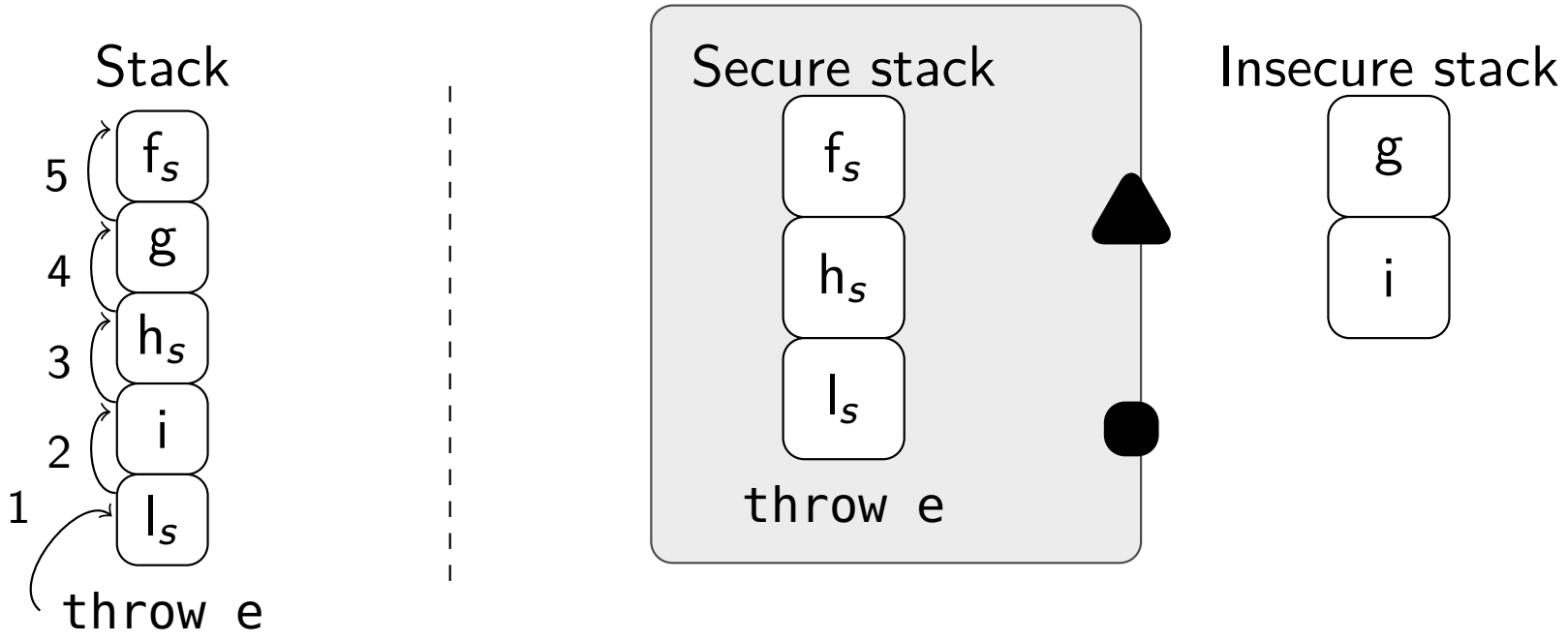
Exceptions


Stack
f_s
g
h_s
i
l_s
4
3
2
1
throw e
Secure stack
f_s
h_s
l_s
throw e
Insecure stack
g
i

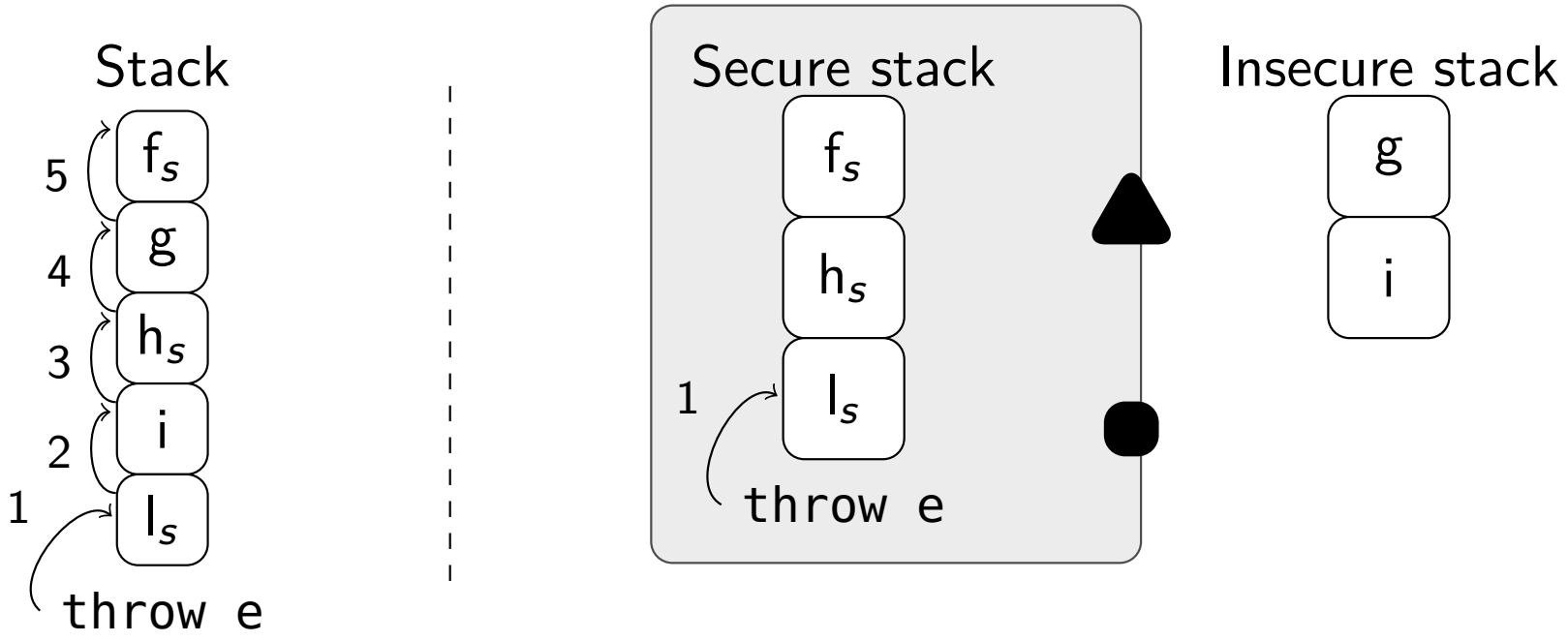
Exceptions


Stack
5
4
3
2
1
f_s
g
h_s
i
l_s
throw e
Secure stack
f_s
h_s
l_s
throw e
Insecure stack
g
i

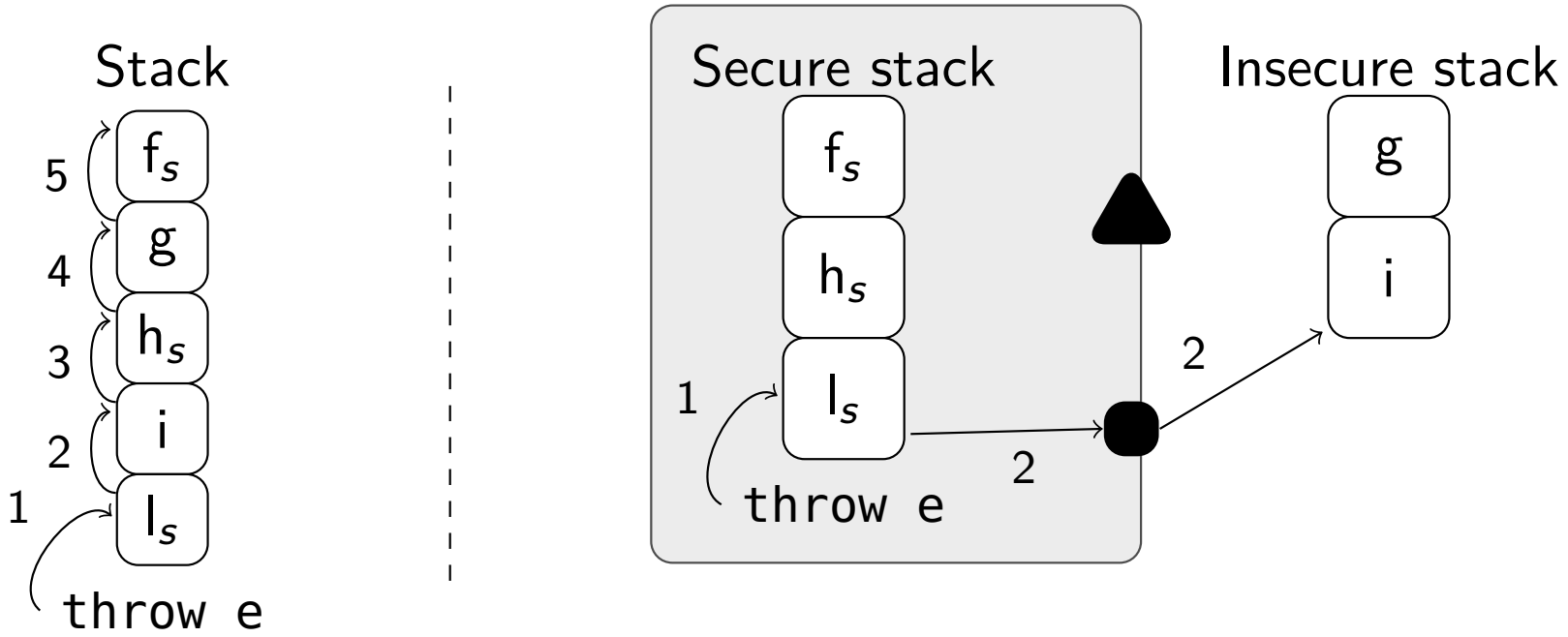
Exceptions


Stack
5
4
3
2
1
f_s
g
h_s
i
l_s
throw e
Secure stack
f_s
h_s
l_s
throw e
Insecure stack
g
i

Exceptions

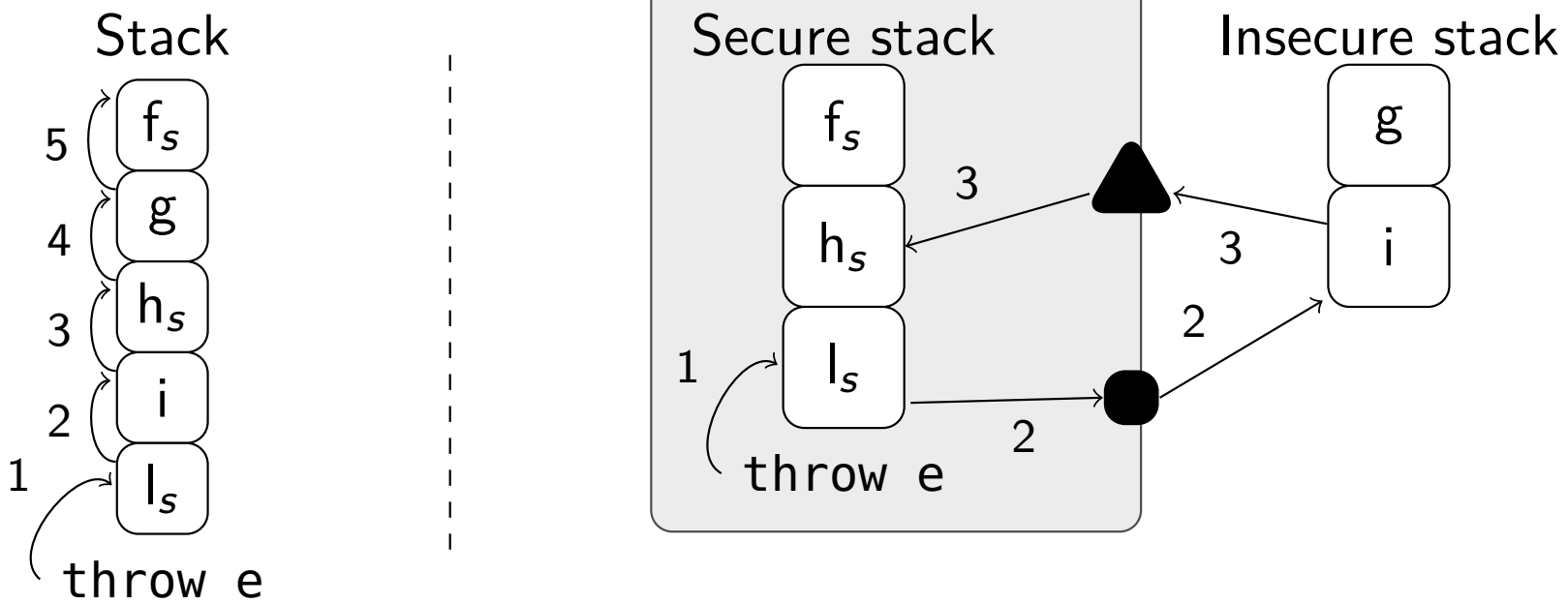

Stack
5
4
3
2
1
f_s
g
h_s
i
l_s
throw e
Secure stack
f_s
h_s
l_s
1
throw e
Insecure stack
g
i

Exceptions



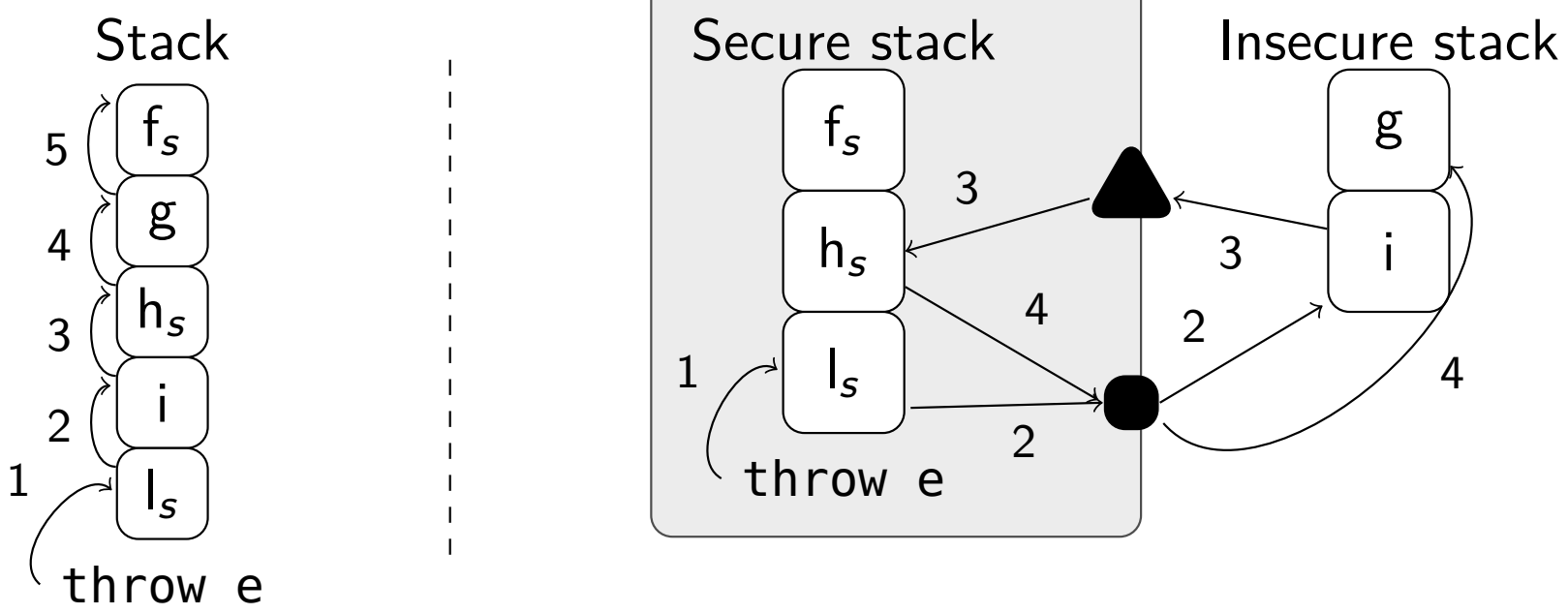
Record passed exceptions

Exceptions



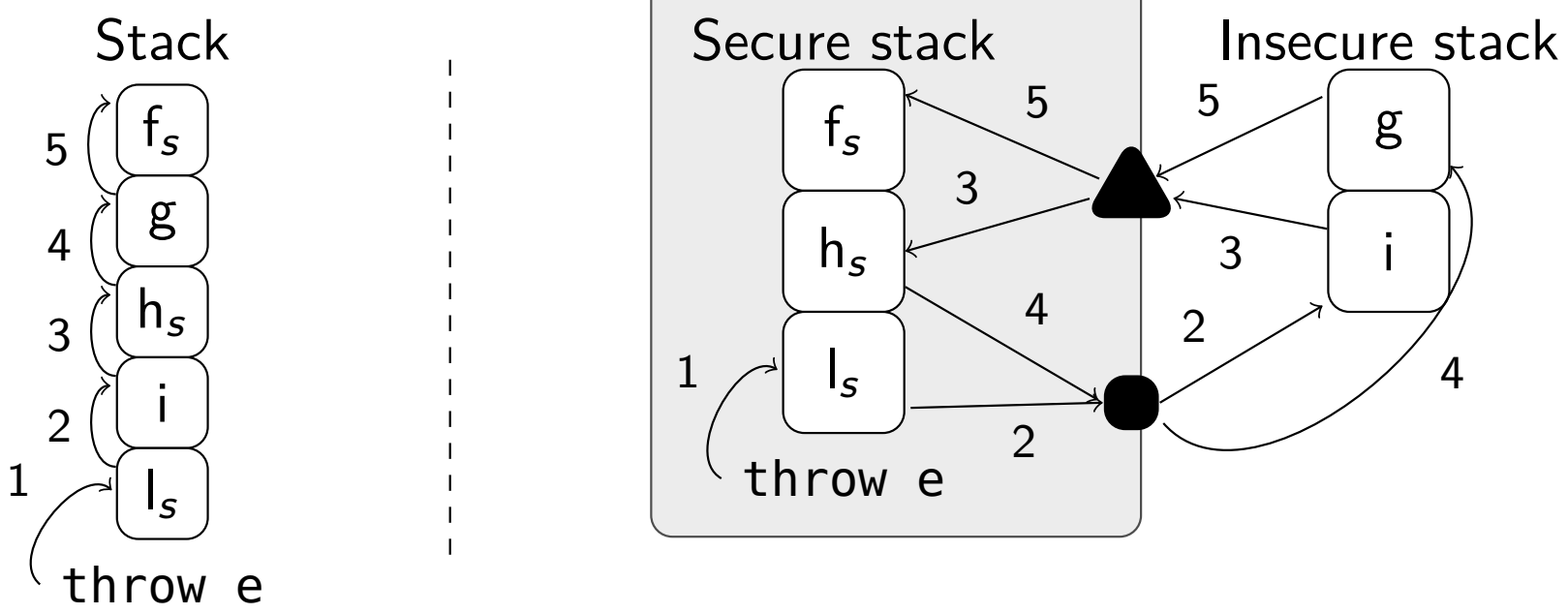
Record passed exceptions
Check that exception could be thrown

Exceptions



Record passed exceptions
Check that exception could be thrown

Exceptions



Record passed exceptions
Check that exception could be thrown

So now...

- We have a strategy to securely compile Java jr code

So now...

- We have a strategy to securely compile Java jr code
- We have the tools to implement it

So now...

- We have a strategy to securely compile Java jr code
- We have the tools to implement it
- We have an idea of the security properties of our secure compilation scheme

So now...

- We have a strategy to securely compile Java jr code
- We have the tools to implement it
- We have an idea of the security properties of our secure compilation scheme

Q: What is missing?

So now...

- We have a strategy to securely compile Java jr code
- We have the tools to implement it
- We have an idea of the security properties of our secure compilation scheme

A PROOF!

Secure Compilation, Formally

$$C_1 \simeq_H C_2 \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

Secure Compilation, Formally

$$C_1 \simeq_H C_2 \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

Secure Compilation, Formally

$$C_1 \simeq_H C_2 \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

Contextual Equivalence

Contextual Equivalence

$$C_1 \simeq C_2 \triangleq \forall \mathbb{C}.\ \mathbb{C}[C_1]\Uparrow \iff \mathbb{C}[C_2]\Uparrow$$

Contextual Equivalence

$$C_1 \simeq C_2 \triangleq \forall \mathbb{C}.\ \mathbb{C}[C_1]\Uparrow \iff \mathbb{C}[C_2]\Uparrow$$

Contextual Equivalence

$$C_1 \simeq C_2 \triangleq \forall \mathbb{C}.\ \mathbb{C}[C_1]\Uparrow \iff \mathbb{C}[C_2]\Uparrow$$

All contexts

Secure Compilation

$$C_1 \simeq_H C_2 \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

Secure Compilation

$$C_1 \simeq_H C_2 \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

$$(\forall \mathbb{C}.\ \mathbb{C}[C_1]\Uparrow \iff \mathbb{C}[C_2]\Uparrow\ ) \iff (\forall \mathbb{M}.\ \mathbb{M}[C_1^{\downarrow}]\Uparrow \iff \mathbb{M}[C_2^{\downarrow}]\Uparrow\ )$$

Secure Compilation

$$C_1 \simeq_H C_2 \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

$$(\forall \mathbb{C}.\ \mathbb{C}[C_1]\Uparrow \iff \mathbb{C}[C_2]\Uparrow\ ) \iff (\forall \mathbb{M}.\ \mathbb{M}[C_1^{\downarrow}]\Uparrow \iff \mathbb{M}[C_2^{\downarrow}]\Uparrow\ )$$

VERY COMPLEX!

Secure Compilation

$$C_1 \simeq_H C_2 \iff C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

Secure Compilation

$$C_1 \simeq_H C_2 \quad \Leftarrow \quad C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

Secure Compilation

$$C_1 \simeq_H C_2 \Leftarrow C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

Secure Compilation

$$C_1 \simeq_H C_2 \quad \Rightarrow \quad C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$

Secure Compilation

$$C_1 \simeq_H C_2 \quad \Rightarrow \quad \boxed{\begin{array}{c} C_1^{\downarrow} \simeq_L C_2^{\downarrow} \\ \Updownarrow \\ \mathsf{Traces}(C_1^{\downarrow}) = \mathsf{Traces}(C_2^{\downarrow}) \end{array}}$$

Fully Abstract Trace Semantics

Secure Compilation

$$C_1 \not\simeq_H C_2 \quad \Leftarrow \quad \mathsf{Traces}(C_1^{\downarrow}) \neq \mathsf{Traces}(C_2^{\downarrow})$$

Secure Compilation

$$C_1 \not\simeq_H C_2 \Leftarrow \quad \mathrm{Traces}(C_1^{\downarrow}) \neq \mathrm{Traces}(C_2^{\downarrow})$$

Secure Compilation

$$C_1 \not\simeq_H C_2 \Leftarrow$$

$$C_1^{\downarrow} \simeq_L C_2^{\downarrow}$$
$$\Updownarrow$$
$$\mathrm{Traces}(C_1^{\downarrow}) = \mathrm{Traces}(C_2^{\downarrow})$$

Fully Abstract Trace Semantics

Trace Semantics for PMA

Trace Semantics for PMA

```
0x0001    call 0xb52
0x0002    ...
:
0x0b52    movi r0 1
0x0b53    movi r1 0x0b56
0x0b54    jl r1
0x0b55    call 0xab01
0x0b56    ret
:
0xab01    ...
```

- behaviour in this case is:

Trace Semantics for PMA

```
0x0001    call 0xb52
0x0002    ...
:
0x0b52    movi r0 1
0x0b53    movi r1 0x0b56
0x0b54    jl r1
0x0b55    call 0xab01
0x0b56    ret
:
0xab01    ...
```

- behaviour in this case is: `call in`

Trace Semantics for PMA

```
0x0001    call 0xb52
0x0002    ...
:
0x0b52    movi r0 1
0x0b53    movi r1 0x0b56
0x0b54    jl r1
0x0b55    call 0xab01
0x0b56    ret
:
0xab01    ...
```

- behaviour in this case is: `call in`, `ret 1`

Trace Semantics for PMA

```
0x0001    call 0xb52
0x0002    ...
:
0x0b52    movi r0 1
0x0b53    movi r1 0x0b56
0x0b54    jl r1
0x0b55    call 0xab01
0x0b56    ret
:
0xab01    ...
```

- behaviour in this case is:
  `call in, ret 1`
  or `call in,`

Trace Semantics for PMA

```
0x0001    call 0xb52
0x0002    ...
:
0x0b52    movi r0 1
0x0b53    movi r1 0x0b56
0x0b54    jl r1
0x0b55    call 0xab01
0x0b56    ret
:
0xab01    ...
```

- behaviour in this case is:
  call in, ret 1
  or call in, call out

Trace Semantics for PMA

```
0x0001    call 0xb52
0x0002    ...
:
0x0b52    movi r0 1
0x0b53    movi r1 0x0b56
0x0b54    jl r1
0x0b55    call 0xab01
0x0b56    ret
:
0xab01    ...
```

- behaviour in this case is:
  `call in, ret 1`
  or `call in, call out`
- traces rely only on the PMA code

Trace Semantics for PMA

```
0x0001    call 0xb52
0x0002    ...
:
0x0b52    movi r0 1
0x0b53    movi r1 0x0b56
0x0b54    jl r1
0x0b55    call 0xab01
0x0b56    ret
:
0xab01    ...
```

- behaviour in this case is:
  `call in, ret 1`
  or `call in, call out`
- traces rely only on the PMA code
- they describe what can be observed from the outside of protected PMA code

Trace Semantics for PMA, Semi-formally

- define states S for programs

Trace Semantics for PMA, Semi-formally

- define states S for programs
- define a semantics for PMA only: $\xrightarrow{i}: S \times S$

Trace Semantics for PMA, Semi-formally

- define states S for programs
- define a semantics for PMA only: $\xrightarrow{i}: S \times S$
- define labels (observables) α

Trace Semantics for PMA, Semi-formally

- define states S for programs
- define a semantics for PMA only: $\xrightarrow{i}: S \times S$
- define labels (observables) α : `call p` $\overline{\mathtt{r}}$

Trace Semantics for PMA, Semi-formally

- define states S for programs
- define a semantics for PMA only: $\xrightarrow{i}: S \times S$
- define labels (observables) α : `call p` $\overline{\mathtt{r}}$ | `ret` $\mathtt{r}_0$

Trace Semantics for PMA, Semi-formally

- define states S for programs
- define a semantics for PMA only: $\xrightarrow{i}: S \times S$
- define labels (observables) α : $\texttt{call p}\ \overline{\texttt{r}} \mid \texttt{ret r}_0$
- define a semantics with labels $\overset{\alpha}{\Longrightarrow}: S \times \alpha \times S$

Trace Semantics for PMA, Semi-formally

- define states S for programs
- define a semantics for PMA only: $\xrightarrow{i}: S \times S$
- define labels (observables) α : `call p` $\overline{\mathtt{r}}$ | `ret` $\mathtt{r_0}$
- define a semantics with labels $\overset{\alpha}{\Longrightarrow}: S \times \alpha \times S$
- $\mathrm{TR}(C) = \{\overline{\alpha} \mid \exists S'.S(C) \overset{\alpha_1}{\Longrightarrow} \cdots \overset{\alpha_n}{\Longrightarrow} S'\}$

Trace Semantics for PMA, Semi-formally

- define states S for programs
- define a semantics for PMA only: $\xrightarrow{i}: S \times S$
- define labels (observables) α : $\texttt{call p}\ \overline{\texttt{r}} \mid \texttt{ret r}_0$
- define a semantics with labels $\overset{\alpha}{\Longrightarrow}: S \times \alpha \times S$
- $\mathrm{TR}(C) = \{\overline{\alpha} \mid \exists S'.S(C) \overset{\alpha_1}{\Longrightarrow} \cdots \overset{\alpha_n}{\Longrightarrow} S'\}$

$$\mathrm{TR} = \left\{ \alpha = \begin{Bmatrix} \xrightarrow{i}; \\ \begin{Bmatrix} \texttt{call p}\ \overline{\texttt{r}} \\ \texttt{ret r}_0 \end{Bmatrix}; \\ \overset{\alpha}{\Longrightarrow} \end{Bmatrix} \right\}$$

Challenge: Precise Reasoning

- formalism to reason about PMA code simply: ✔

Challenge: Precise Reasoning

- formalism to reason about PMA code simply: ✔
- precise formalism? ✗

Challenge: Precise Reasoning

- formalism to reason about PMA code simply: ✔
- precise formalism? ✗
  1. PMA code can write in unprotected memory

Challenge: Precise Reasoning

- formalism to reason about PMA code simply: ✔
- precise formalism? ✗
  1. PMA code can write in unprotected memory
  2. flags convey information across function calls

Challenge: Precise Reasoning

- formalism to reason about PMA code simply: ✔
- precise formalism? ✗
  1. PMA code can write in unprotected memory
  2. flags convey information across function calls
  3. registers besides r_0 in `ret` as well

Fully Abstract Trace Semantics

To ensure maximal precision, prove the trace semantics to be fully abstract

Fully Abstract Trace Semantics

To ensure maximal precision, prove the trace semantics to be fully abstract

i.e. there are no other things that we missed

The Spectrum of Full Abstraction (by Curien)

$$\mathsf{TR} = \left\{ \alpha = \left\{ \begin{array}{c} \xrightarrow{i}; \\ \texttt{call p } \overline{\texttt{r}} \\ \texttt{ret r}_0 \\ \xRightarrow{\alpha} \end{array} \right\}; \right\} \times$$

The Spectrum of Full Abstraction (by Curien)

$$\mathsf{TR} = \left\{ \alpha = \begin{Bmatrix} \xrightarrow{i}; \\ \texttt{call p } \overline{\texttt{r}} \\ \texttt{ret r}_0 \\ \xRightarrow{\alpha} \end{Bmatrix}; \right\} \times$$

$$\mathsf{TR}_\mathsf{L} = \left\{ \alpha = \begin{Bmatrix} \xrightarrow{i}; \\ \texttt{call } \overline{\texttt{r}}\ \overline{\texttt{f}} \\ \texttt{ret } \overline{\texttt{r}}\ \overline{\texttt{f}} \\ \texttt{movs r v} \\ \xRightarrow{\alpha} \end{Bmatrix}; \right\} \checkmark$$

The Spectrum of Full Abstraction (by Curien)

$$\mathsf{TR} = \left\{ \alpha = \begin{Bmatrix} \xrightarrow{i}; \\ \texttt{call p } \overline{\texttt{r}} \\ \texttt{ret r}_0 \end{Bmatrix} \overset{\alpha}{\Longrightarrow}; \right\} \times$$

$$\mathsf{TR}_\mathsf{L} = \left\{ \alpha = \begin{Bmatrix} \xrightarrow{i}; \\ \texttt{call } \overline{\texttt{r}}\ \overline{\texttt{f}} \\ \texttt{ret } \overline{\texttt{r}}\ \overline{\texttt{f}} \\ \texttt{movs r v} \end{Bmatrix} \overset{\alpha}{\Longrightarrow}; \right\} \checkmark$$

$$\mathsf{TR}_\mathsf{S} = \left\{ \alpha = \begin{Bmatrix} \xrightarrow{i}\!\!\!\!\rightarrow; \\ \texttt{call p } \overline{\texttt{r}} \\ \texttt{ret r}_0 \end{Bmatrix} \overset{\alpha}{\Longrightarrow}; \right\} \checkmark$$

The Spectrum of Full Abstraction (by Curien)

$$\mathsf{TR} = \left\{ \alpha = \begin{Bmatrix} \xrightarrow{i}; \\ \texttt{call p } \overline{\texttt{r}} \\ \texttt{ret r}_0 \end{Bmatrix} \overset{\alpha}{\Longrightarrow}; \right\} \times$$

$$\mathsf{TR}_\mathsf{L} = \left\{ \alpha = \begin{Bmatrix} \xrightarrow{i}; \\ \texttt{call } \overline{\texttt{r}}\ \overline{\texttt{f}} \\ \texttt{ret } \overline{\texttt{r}}\ \overline{\texttt{f}} \\ \texttt{movs r v} \end{Bmatrix} \overset{\alpha}{\Longrightarrow}; \right\} \checkmark \qquad \mathsf{TR}_\mathsf{S} = \left\{ \alpha = \begin{Bmatrix} \xrightarrow{i}; \\ \texttt{call p } \overline{\texttt{r}} \\ \texttt{ret r}_0 \end{Bmatrix} \overset{\alpha}{\Longrightarrow}; \right\} \checkmark$$

. . . . . .

The Spectrum of Full Abstraction (by Curien)

$$\mathsf{TR} = \left\{ \alpha = \left\{ \begin{array}{c} \xrightarrow{i}; \\ \texttt{call p } \overline{\texttt{r}} \\ \texttt{ret r}_0 \\ \xRightarrow{\alpha} \end{array} \right\}; \right\} \times$$

$$\mathsf{TR}_\mathsf{L} = \left\{ \alpha = \left\{ \begin{array}{c} \xrightarrow{i}; \\ \texttt{call } \overline{\texttt{r}}\ \overline{\texttt{f}} \\ \texttt{ret } \overline{\texttt{r}}\ \overline{\texttt{f}} \\ \texttt{movs r v} \\ \xRightarrow{\alpha} \end{array} \right\}; \right\} \checkmark \qquad \mathsf{TR}_\mathsf{S} = \left\{ \alpha = \left\{ \begin{array}{c} \xrightarrow{i}; \\ \texttt{call p } \overline{\texttt{r}} \\ \texttt{ret r}_0 \\ \xRightarrow{\alpha} \end{array} \right\}; \right\} \checkmark$$

. . . . . .

$$\mathsf{TR}_\mathsf{X}(C_1) = \mathsf{TR}_\mathsf{X}(C_2) \iff C_1 \simeq C_2$$

Outline

Multilanguage Model (by Adriaan)

- reasoning at assembly level is complex

Multilanguage Model (by Adriaan)

- reasoning at assembly level is complex
- a λ-calculus model simplifies proofs for other language features

Multilanguage Model (by Adriaan)

- reasoning at assembly level is complex
- a λ-calculus model simplifies proofs for other language features

Q: How to proceed?

Multilanguage Model (by Adriaan)

- reasoning at assembly level is complex
- a λ-calculus model simplifies proofs for other language features

Q: How to proceed?

- devise a multilanguage model

Multilanguage Model (by Adriaan)

- reasoning at assembly level is complex
- a λ-calculus model simplifies proofs for other language features

Q: How to proceed?

- devise a multilanguage model
- show that it models precisely PMA

Multilanguage Model (by Adriaan)

- reasoning at assembly level is complex
- a λ-calculus model simplifies proofs for other language features

Q: How to proceed?

- devise a multilanguage model
- show that it models precisely PMA
- adopt it in other proofs!

Multi-principal Languages

- current model has a single secure entity

Multi-principal Languages

- current model has a single secure entity
- current prototypes allow a flat security order

Multi-principal Languages

- current model has a single secure entity
- current prototypes allow a flat security order

Q: How can we improve on this?

Multi-principal Languages

- current model has a single secure entity
- current prototypes allow a flat security order

Q: How can we improve on this?

- model a security lattice with the current prototypes

Multi-principal Languages

- current model has a single secure entity
- current prototypes allow a flat security order

Q: How can we improve on this?

- model a security lattice with the current prototypes
- secure compilation of languages with multiple security principals

Multithreaded Languages

- current PMA prototypes are single-threaded

Multithreaded Languages

- current PMA prototypes are single-threaded
- secure compilation for PMA does not consider concurrency nor distribution

Multithreaded Languages

- current PMA prototypes are single-threaded
- secure compilation for PMA does not consider concurrency nor distribution

Q: What's next on this line?

Multithreaded Languages

- current PMA prototypes are single-threaded
- secure compilation for PMA does not consider concurrency nor distribution

Q: What's next on this line?

- existing works cover concurrency and distribution for secure compilers

Multithreaded Languages

- current PMA prototypes are single-threaded
- secure compilation for PMA does not consider concurrency nor distribution

Q: What's next on this line?

- existing works cover concurrency and distribution for secure compilers
- investigate the implementation: which interrupts to handle?

Multithreaded Languages

- current PMA prototypes are single-threaded
- secure compilation for PMA does not consider concurrency nor distribution

Q: What's next on this line?

- existing works cover concurrency and distribution for secure compilers
- investigate the implementation: which interrupts to handle?
- the single-machine, multithreaded setting is poorly explored

What is the limit?

What is the limit?

Q: Are there language features that cannot be securely compiled through PMA?

What is the limit?

Q: Are there language features that cannot be securely compiled through PMA?

- how to formalise this statement?

What is the limit?

Q: Are there language features that cannot be securely compiled through PMA?

- how to formalise this statement?
- i think the answer is NO

Questions

Thank you!

Qs ?