Secure Compilation to Protected Module Architectures

Marco Patrignani [1] Dave Clarke [2,3] Frank Piessens [2]

[1]Max Planck institute for software systems, Saarbrücken, Germany
first.last@mpi-sws.org
[2]iMinds-DistriNet, Dept. Computer Science, KU Leuven, Belgium
first.last@cs.kuleuven.be
[3]Dept. Information Technology, Uppsala University, Sweden
first.last@it.uu.se

25 January 2016

Goal of the Talk

- present my research on secure compilation

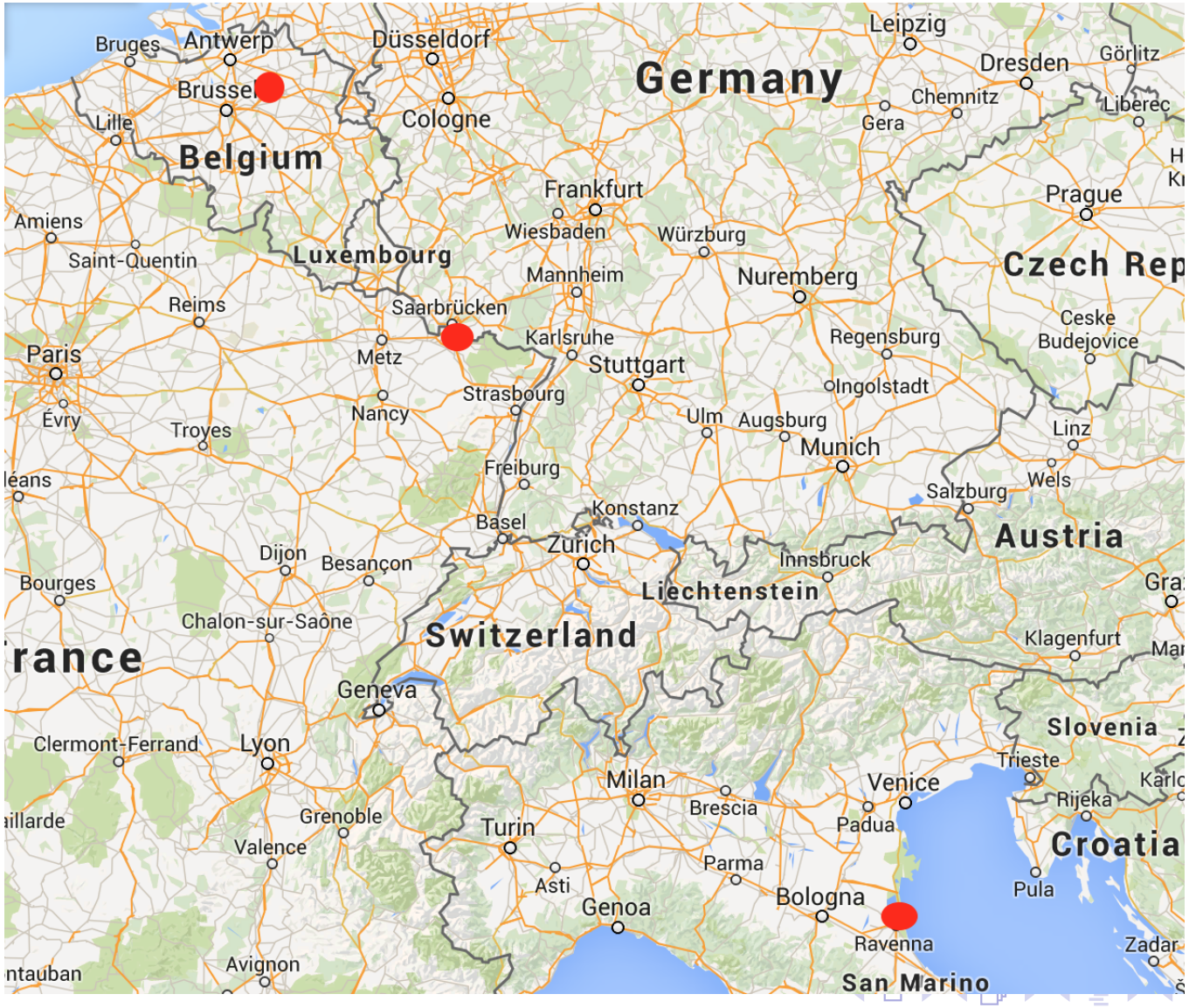
Goal of the Talk

- present my research on secure compilation
- define secure (fully-abstract) compilation

Goal of the Talk

- present my research on secure compilation
- define secure (fully-abstract) compilation
- discuss present and future work

Me


Bruges
Antwerp
Düsseldorf
Leipzig
Germany
Dresden
Görlitz
Brussel
Cologne
Chemnitz
Liberec
Lille
Gera
Belgium
Frankfurt
Prague
Amiens
Wiesbaden
Würzburg
Saint-Quentin
Luxembourg
Mannheim
Nuremberg
Czech Rep
Reims
Saarbrücken
Ceske
Budejovice
Karlsruhe
Regensburg
Paris
Metz
Stuttgart
Ingolstadt
Strasbourg
Évry
Nancy
Ulm
Augsburg
Linz
Troyes
Munich
Freiburg
Wels
Salzburg
Konstanz
Basel
Austria
Zurich
Dijon
Besançon
Innsbruck
Bourges
Liechtenstein
Chalon-sur-Saône
Switzerland
Klagenfurt
rance
Geneva
Slovenia
Clermont-Ferrand
Lyon
Trieste
Milan
Venice
Rijeka
Brescia
Grenoble
Padua
Turin
Croatia
Valence
Parma
Asti
Pula
Genoa
Bologna
Ravenna
Zadar
Avignon
San Marino

Outline

Outline

What is a Secure Program?

- a program is secure if it enjoys at least a security property

What is a Secure Program?

- a program is secure if it enjoys at least a security property
- a security property is one expressible via program equivalence (e.g. confidentiality, integrity, etc.)

What is a Secure Compiler?

- a compiler is a function from source to target programs

What is a Secure Compiler?

- a compiler is a function from source to target programs
- a compiler is secure if it preserves source-level security properties in the programs it generates *no more, no less*

What is a Secure Compiler?

- a compiler is a function from source to target programs
- a compiler is secure if it preserves source-level security properties in the programs it generates *no more, no less*
- a fully abstract compiler is a secure compiler

Benefits of Fully abstract Compilation

Benefits of Fully abstract Compilation

Fully abstract compilation preserves source-level abstractions in target-level languages

Benefits of Fully abstract Compilation

Fully abstract compilation preserves source-level abstractions in target-level languages

- protect against code injection attacks

Benefits of Fully abstract Compilation

Fully abstract compilation preserves source-level abstractions in target-level languages

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

What is a Protected Modules Architecture?

- deep encapsulation at the lowest level of abstraction

What is a Protected Modules Architecture?

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

What is a Protected Modules Architecture?

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

A+I: Untyped Assembly + PMA

```
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

A+I: Untyped Assembly + PMA

```
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

A+I: Untyped Assembly + PMA

```
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

A+I: Untyped Assembly + PMA

```
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

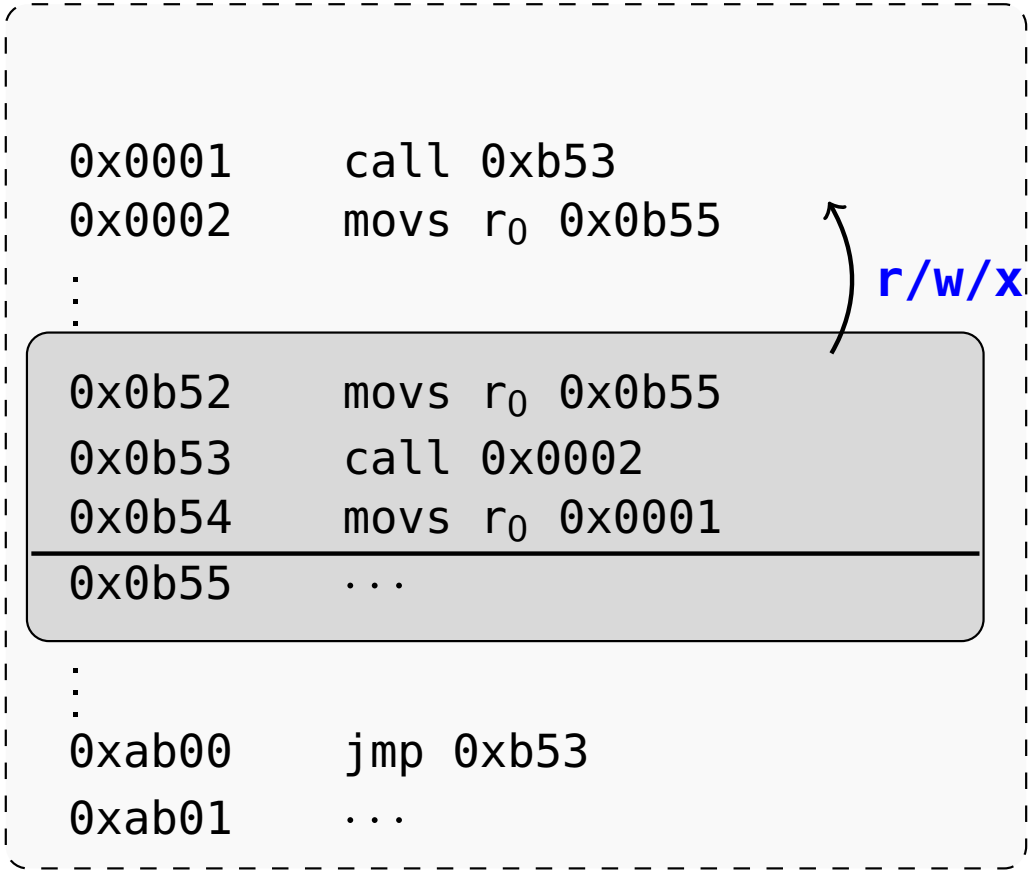
A+I: Untyped Assembly + PMA

```
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

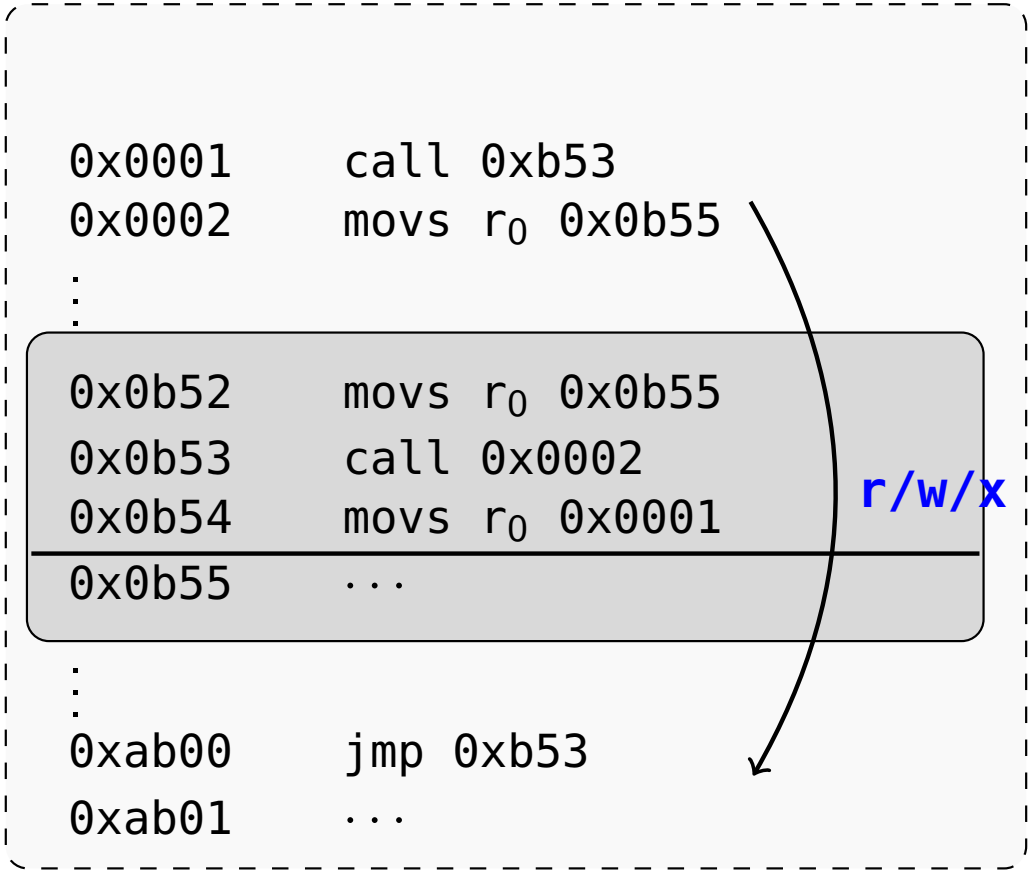
- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted

A+I: Untyped Assembly + PMA



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

A+I: Untyped Assembly + PMA



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

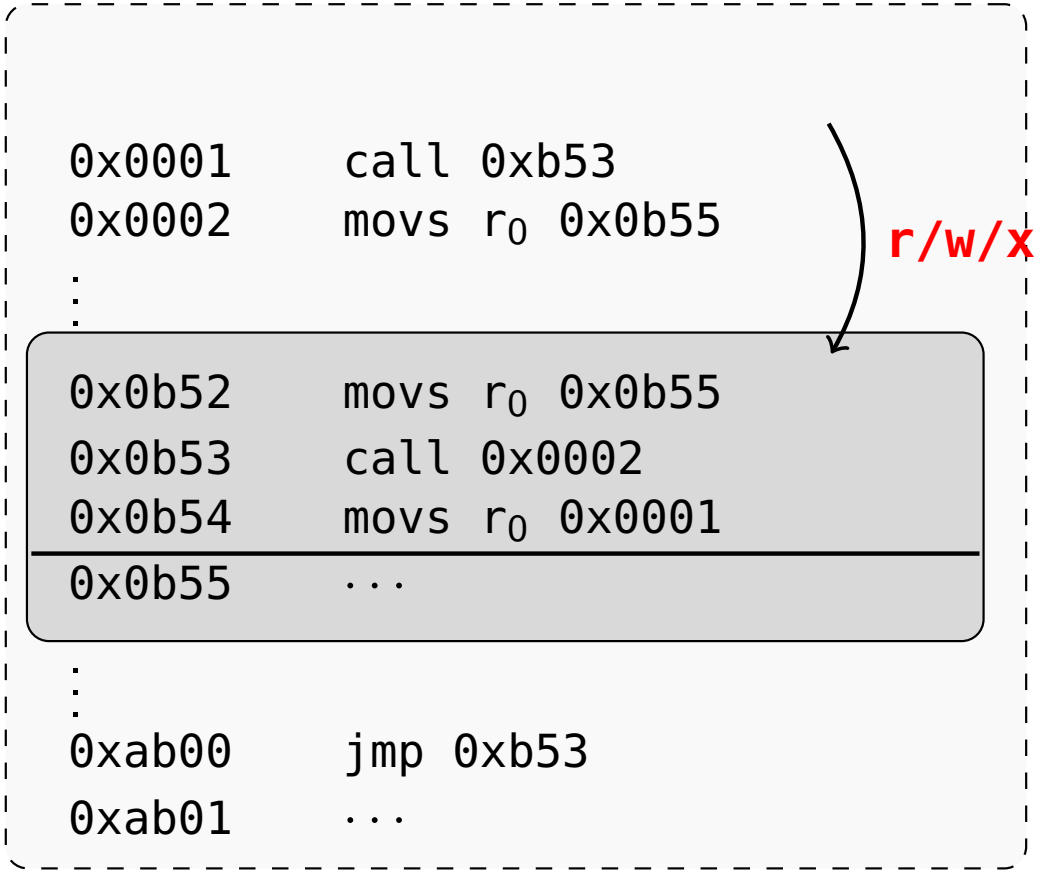
A+I: Untyped Assembly + PMA

```
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

A+I: Untyped Assembly + PMA



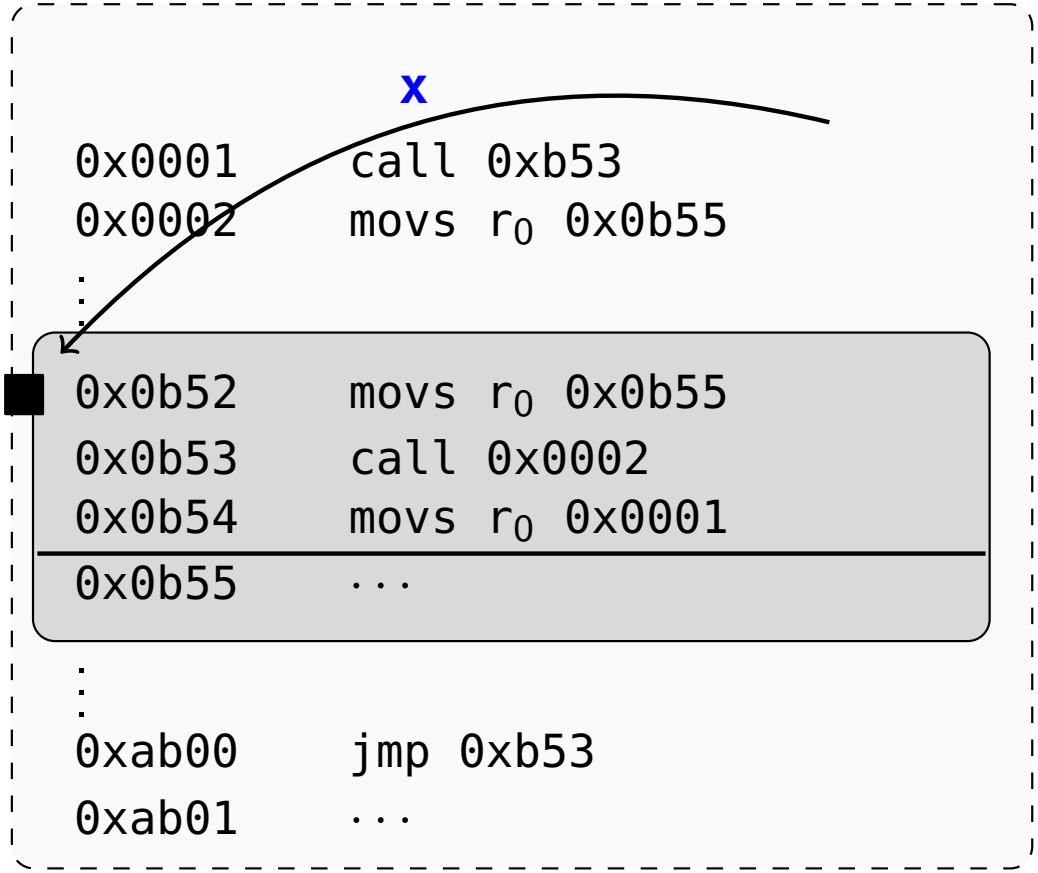
- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted

A+I: Untyped Assembly + PMA

```
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 (■)

A+I: Untyped Assembly + PMA



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

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

Outline

J+E: Java-like Language

- source language: +/- Java jr

J+E: Java-like Language

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

J+E: Java-like Language

- 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;
```

J+E: Java-like Language

- 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;
```

J+E: Java-like Language

```
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;
```

J+E: Java-like Language

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;
```

J+E: Java-like Language

- 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;
```

J+E: Java-like Language

- 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;
```

PMA for Secure Compilation

Source level

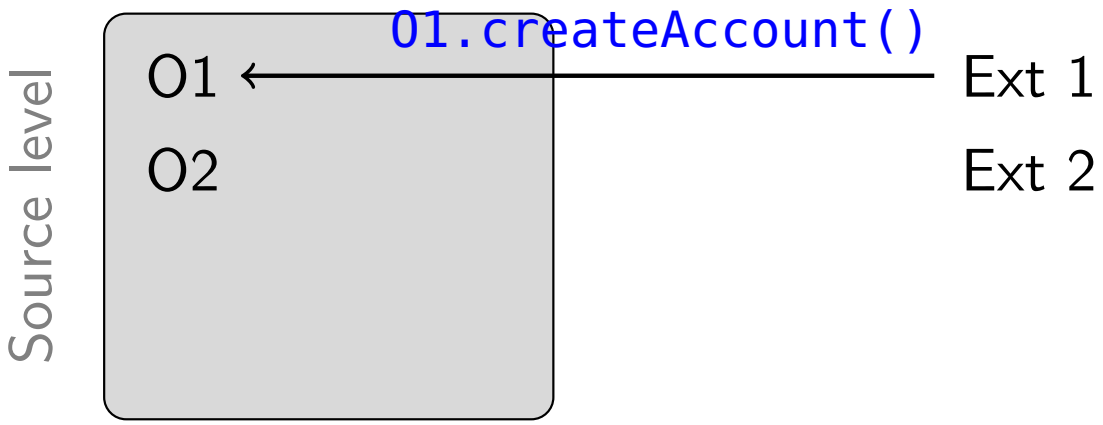
O1

O2

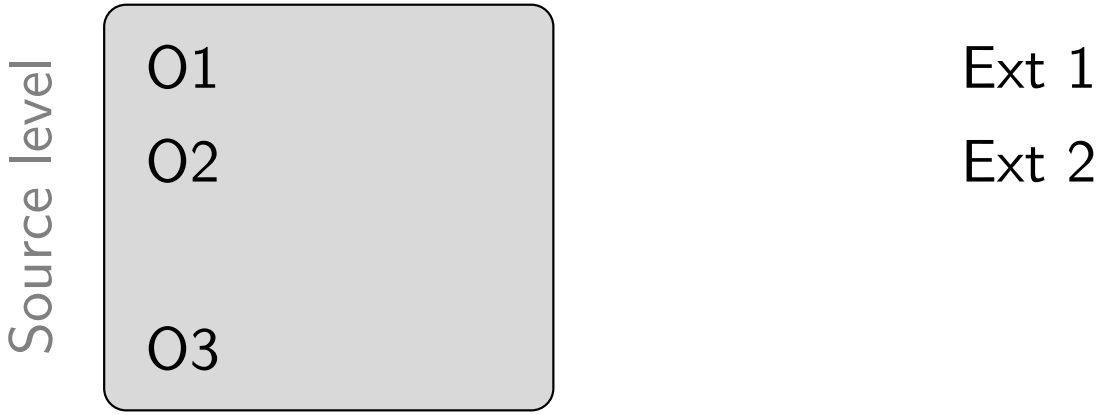
Ext 1

Ext 2

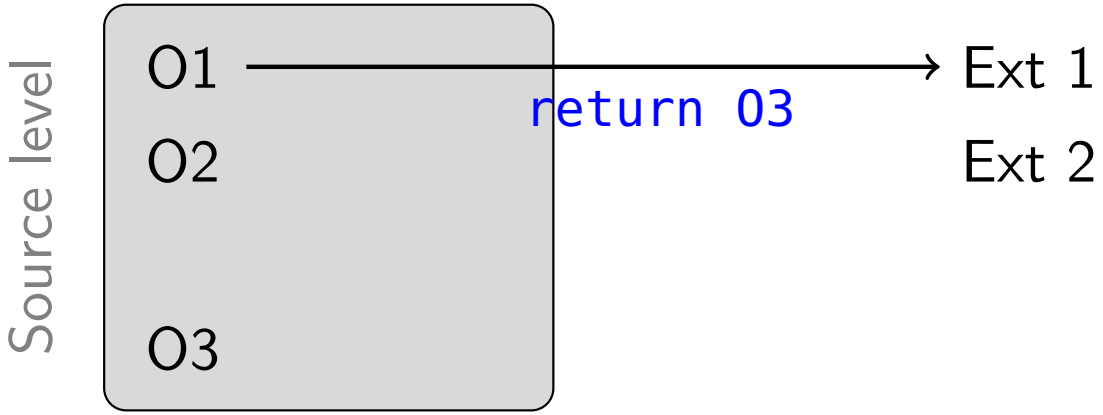
PMA for Secure Compilation



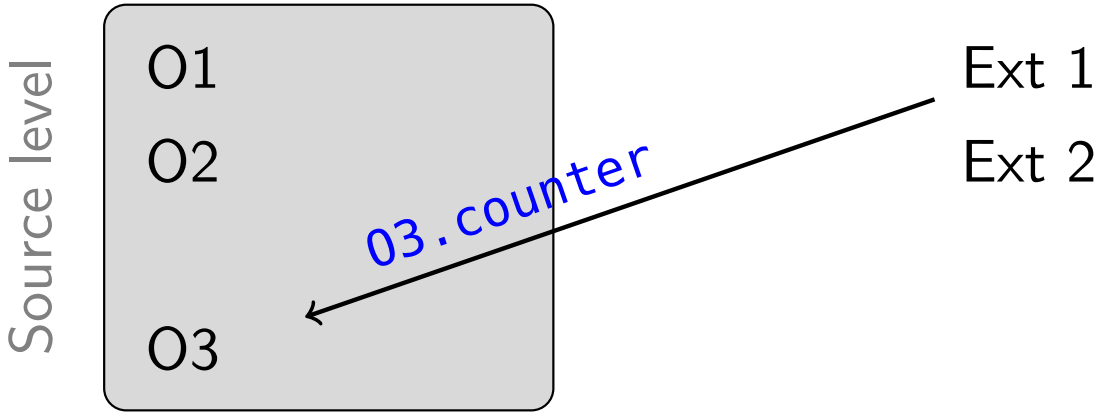
PMA for Secure Compilation



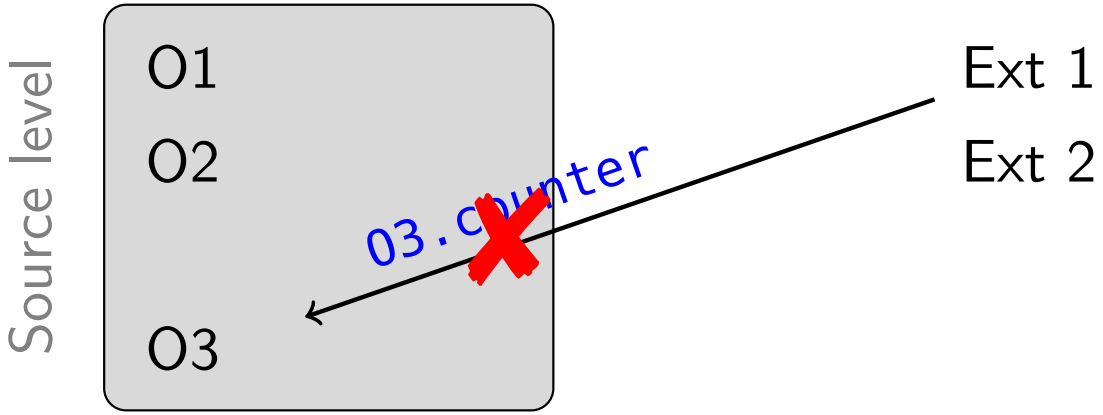
PMA for Secure Compilation



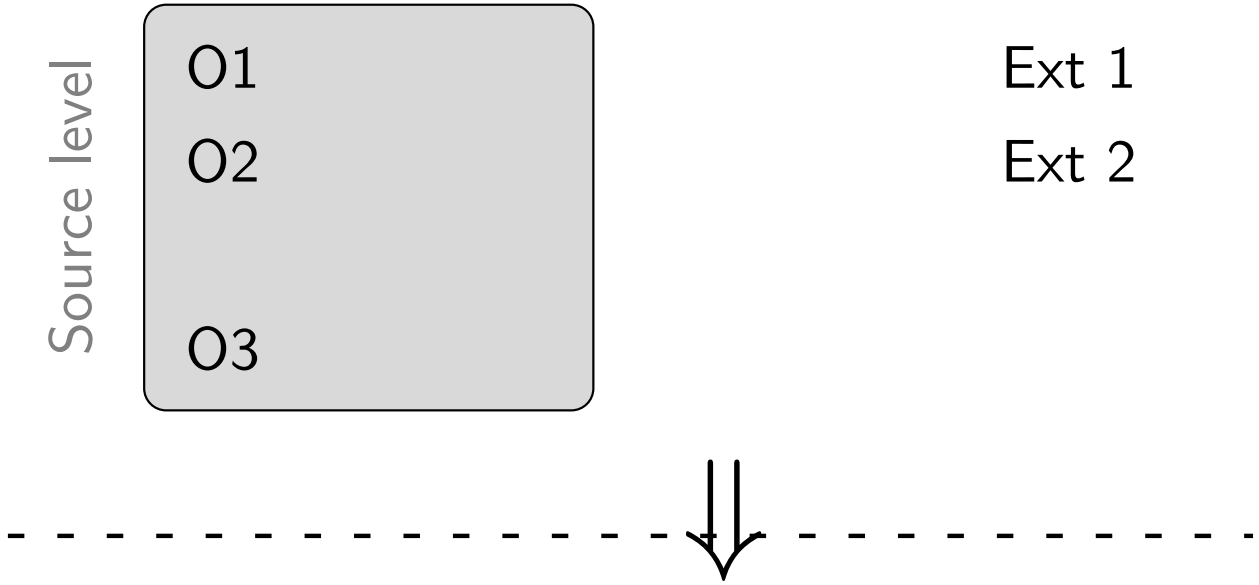
PMA for Secure Compilation



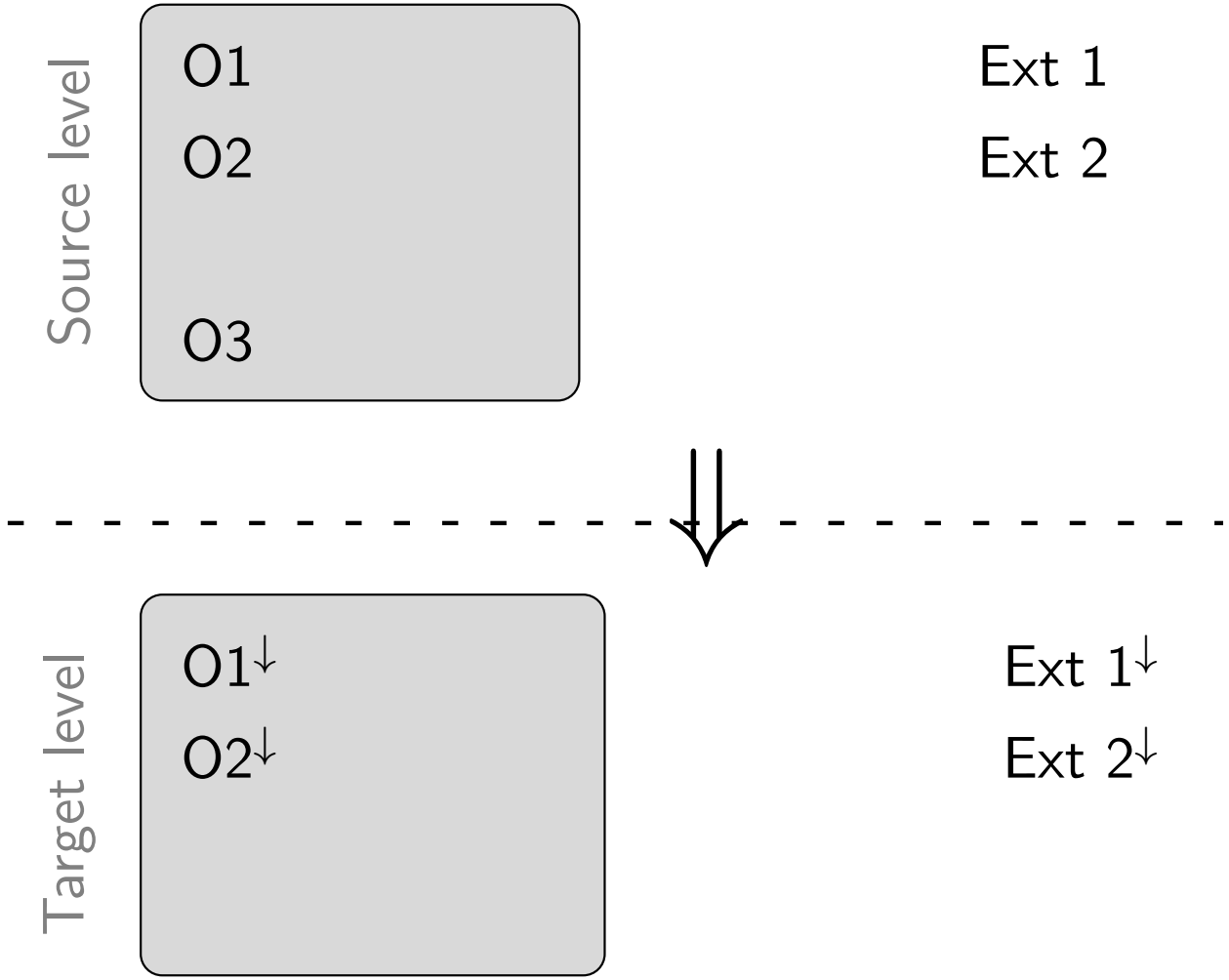
PMA for Secure Compilation



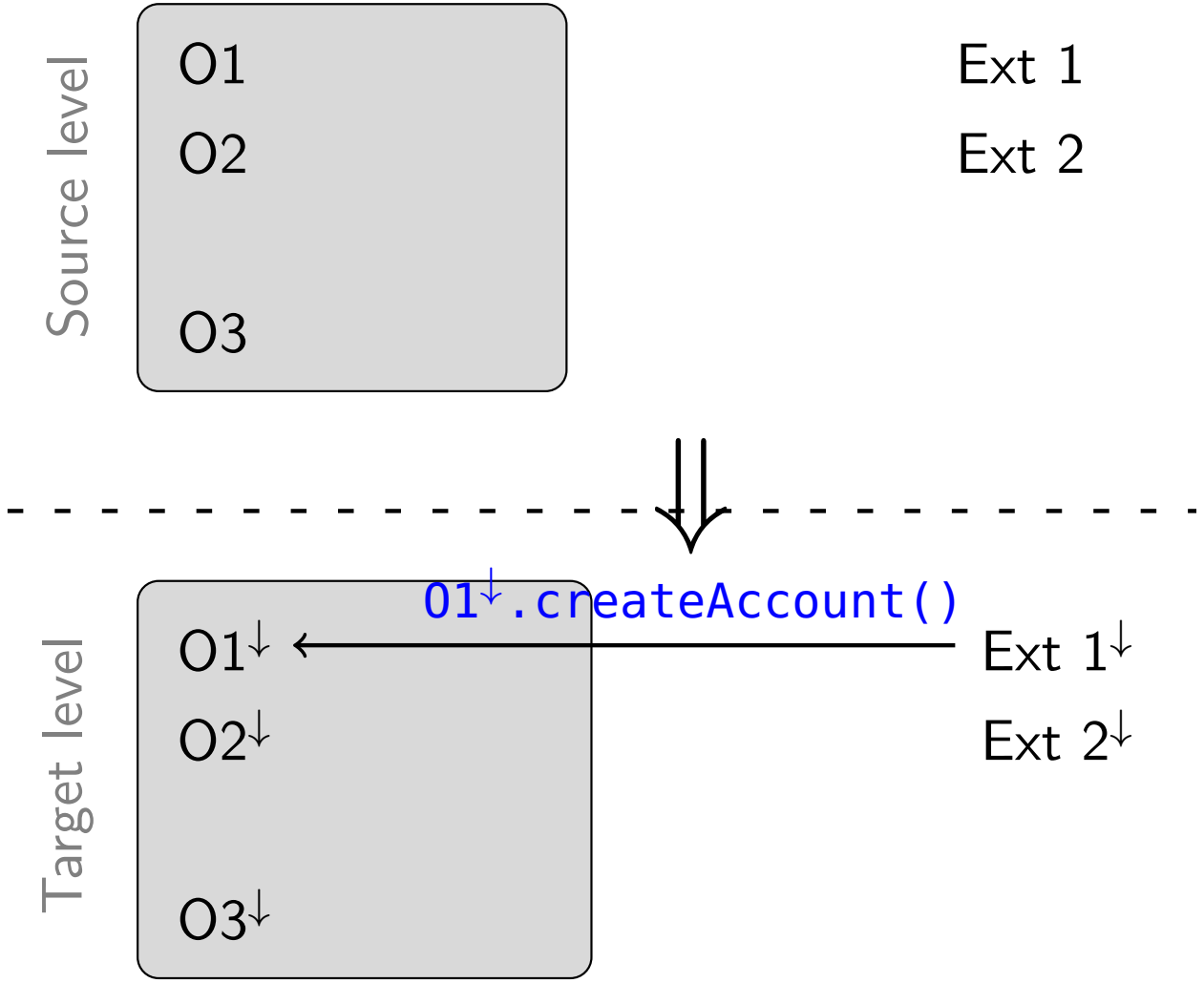
PMA for Secure Compilation



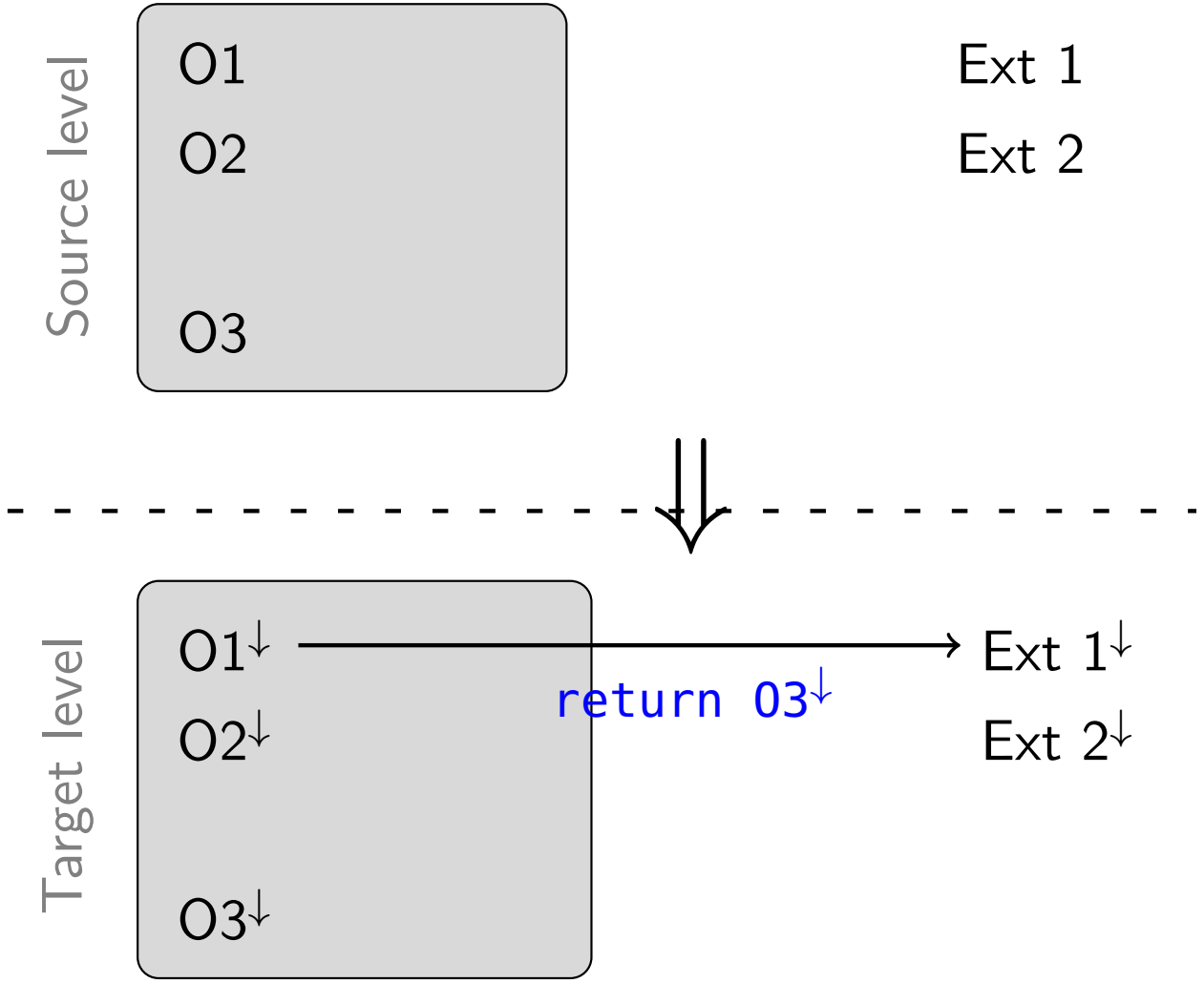
PMA for Secure Compilation



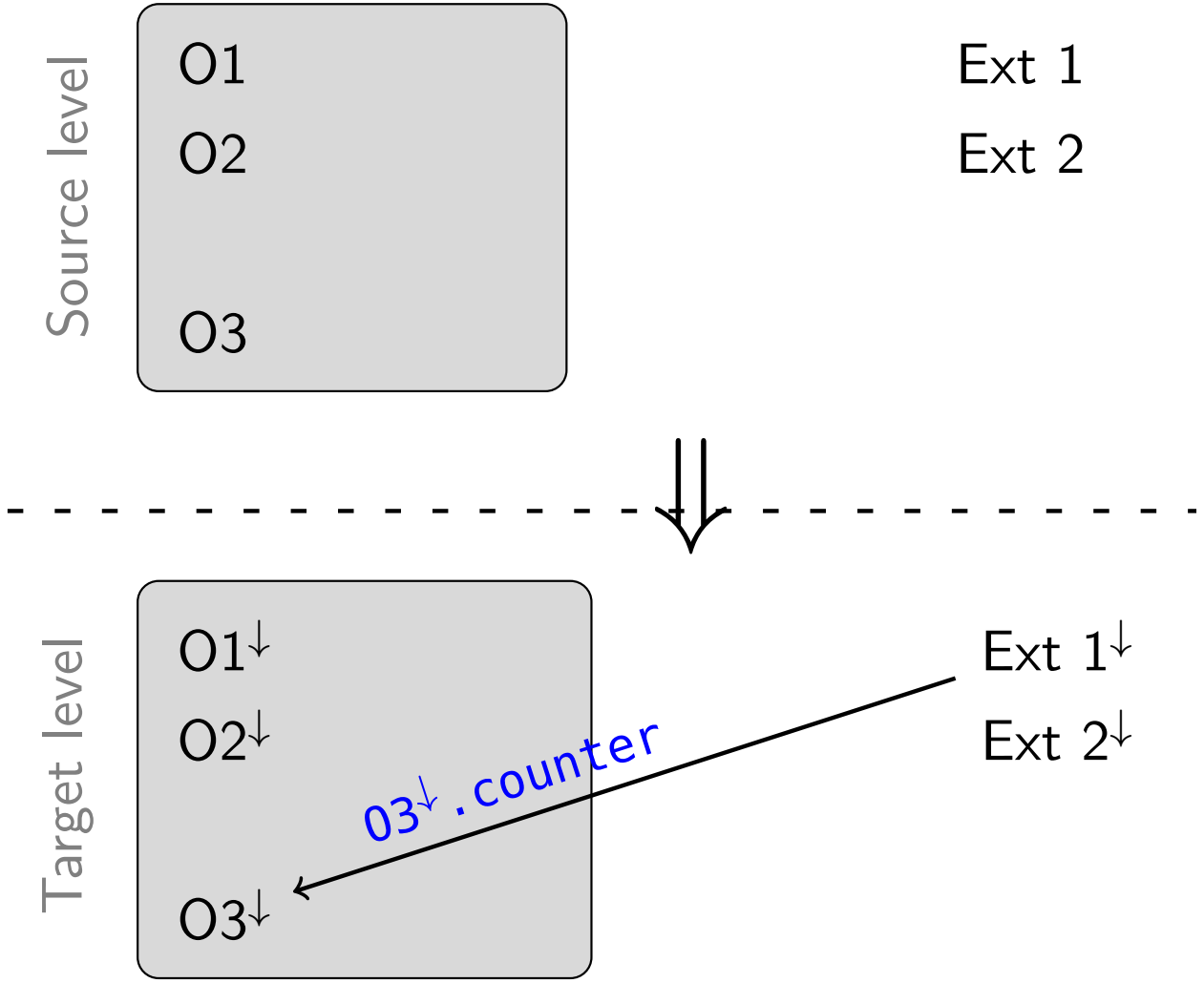
PMA for Secure Compilation



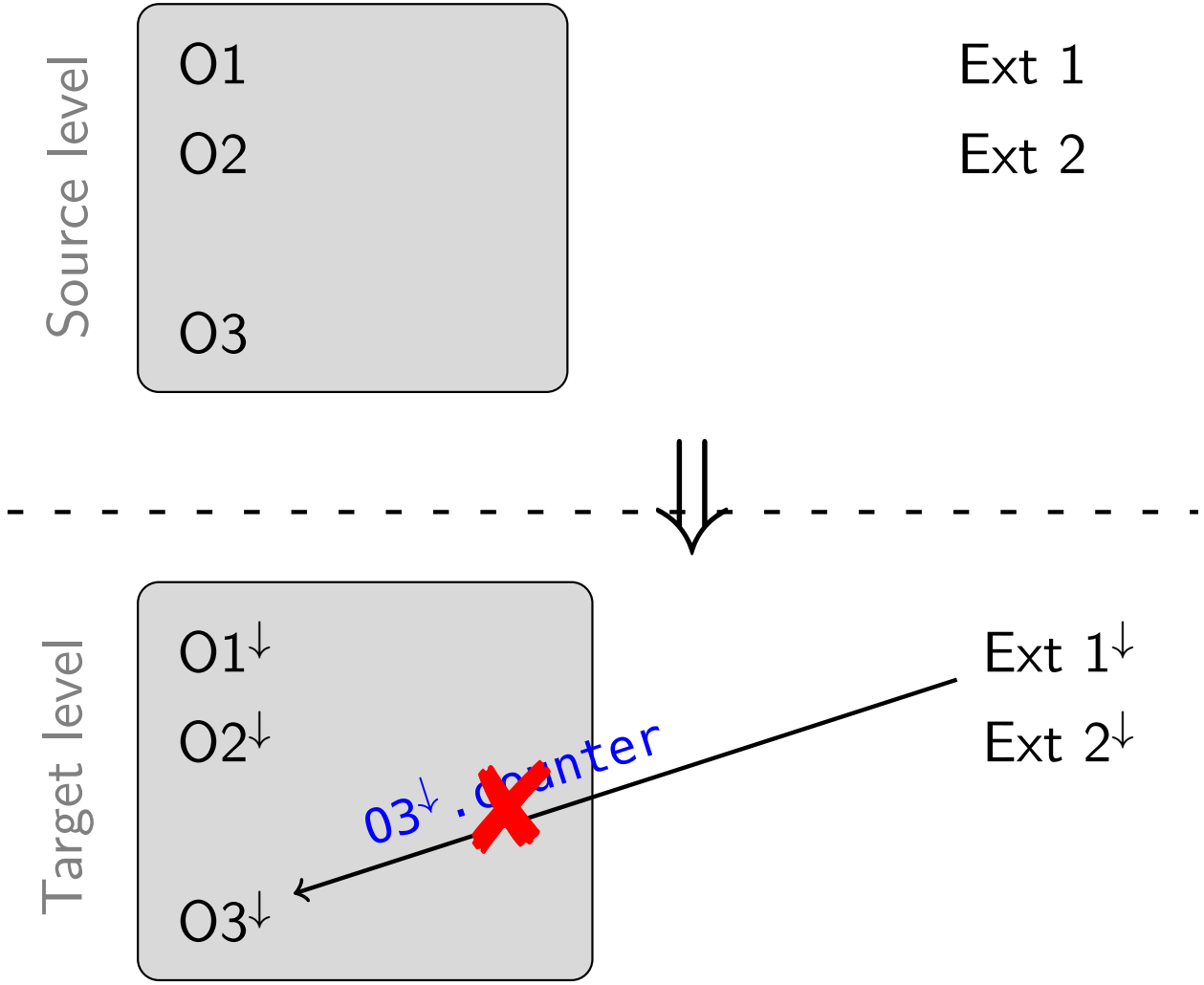
PMA for Secure Compilation



PMA for Secure Compilation

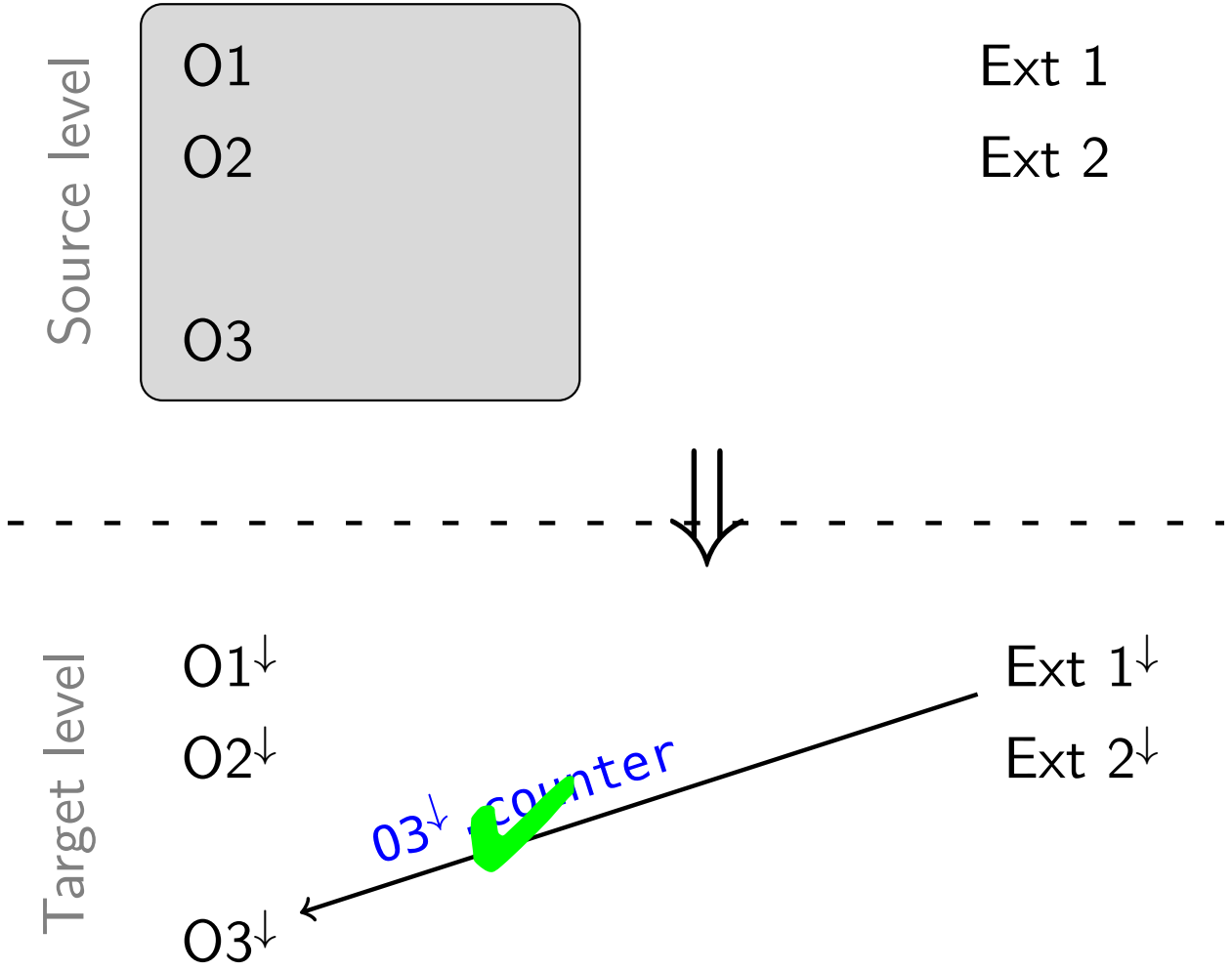


PMA for Secure Compilation



- Protect against low-level attackers

PMA for Secure Compilation



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

Secure Compilation of Outcalls

Q: : Is that all?

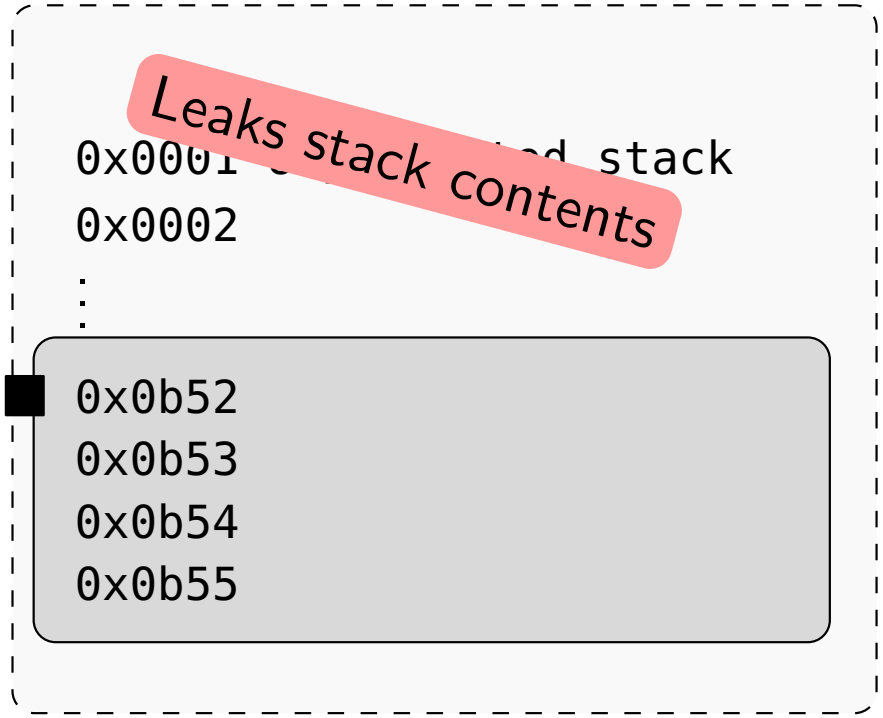
Secure Compilation of Outcalls

Q: : Is that all?

Secure Compilation of Outcalls

Q: : Is that all?

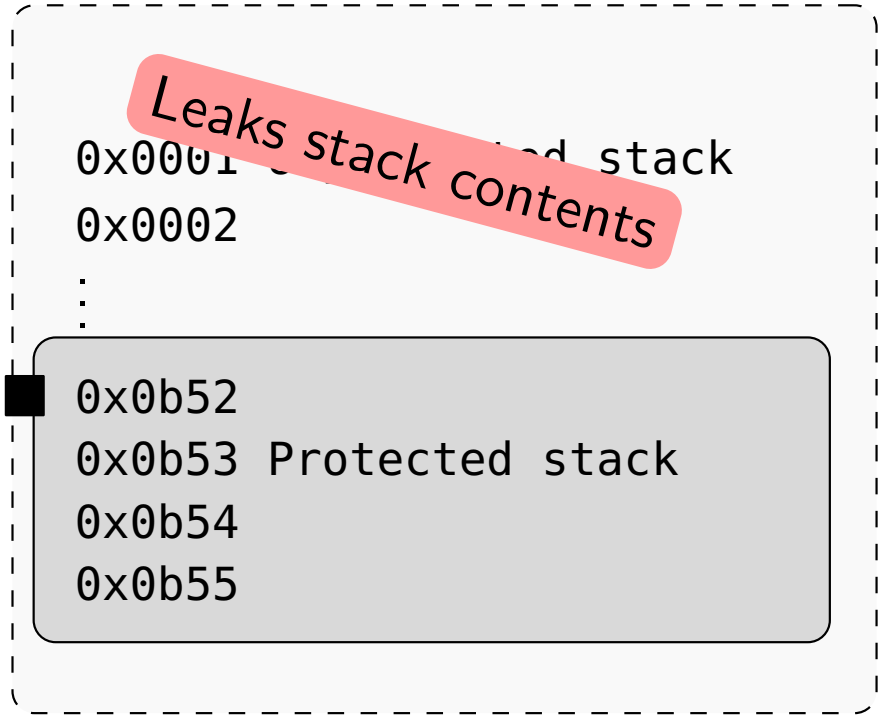
- protected stack



Secure Compilation of Outcalls

Q: : Is that all?

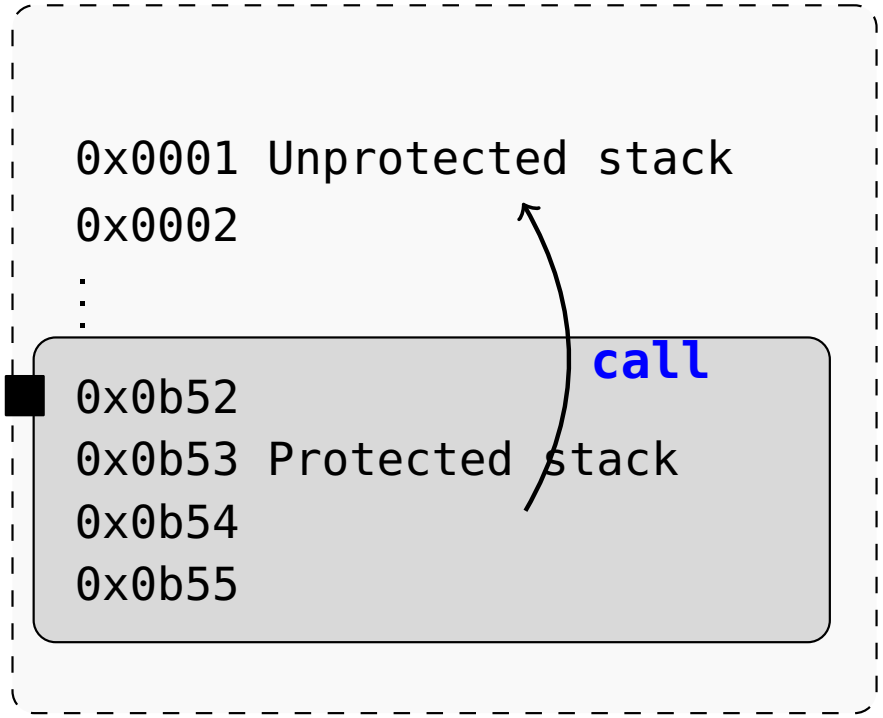
- protected stack



Secure Compilation of Outcalls

Q: : Is that all?

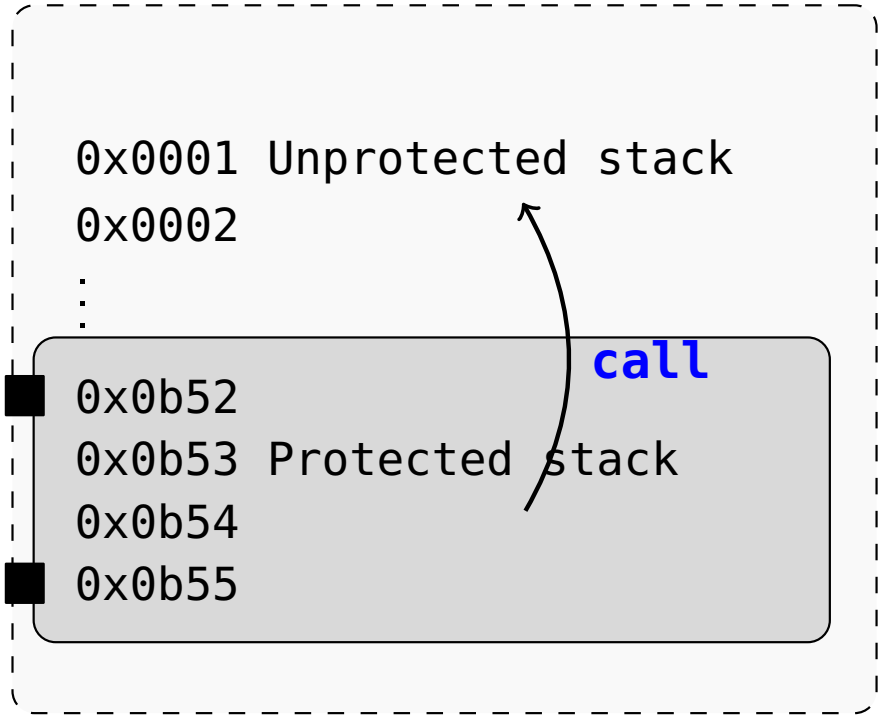
- protected stack
- returnback entry point



Secure Compilation of Outcalls

Q: : Is that all?

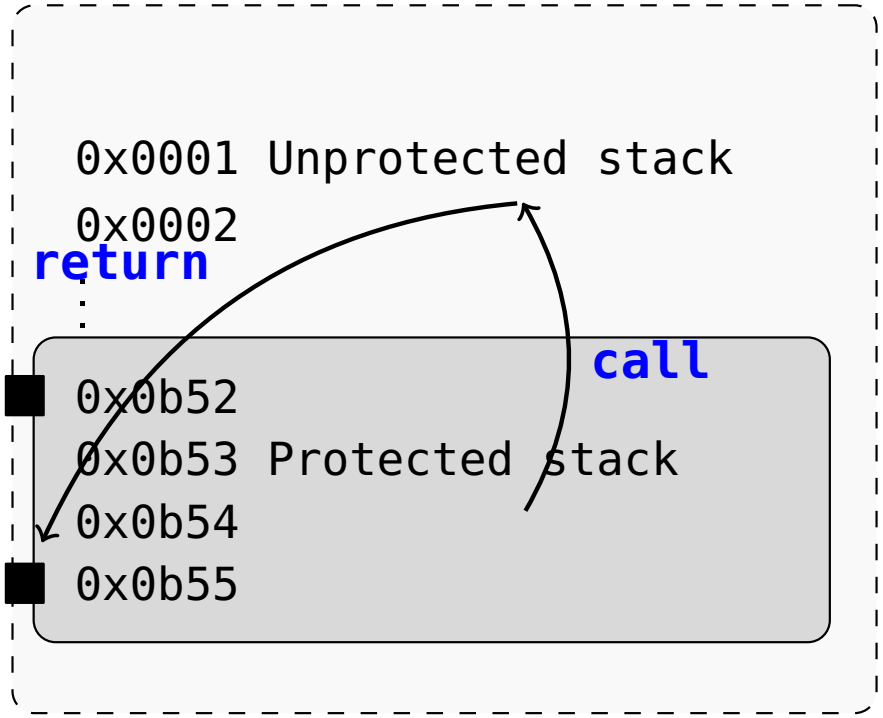
- protected stack
- returnback entry point



Secure Compilation of Outcalls

Q: : Is that all?

- protected stack
- returnback entry point



Secure Compilation of Outcalls

Q: : Is that all?

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

Secure Compilation of Outcalls

Q: : Is that all?

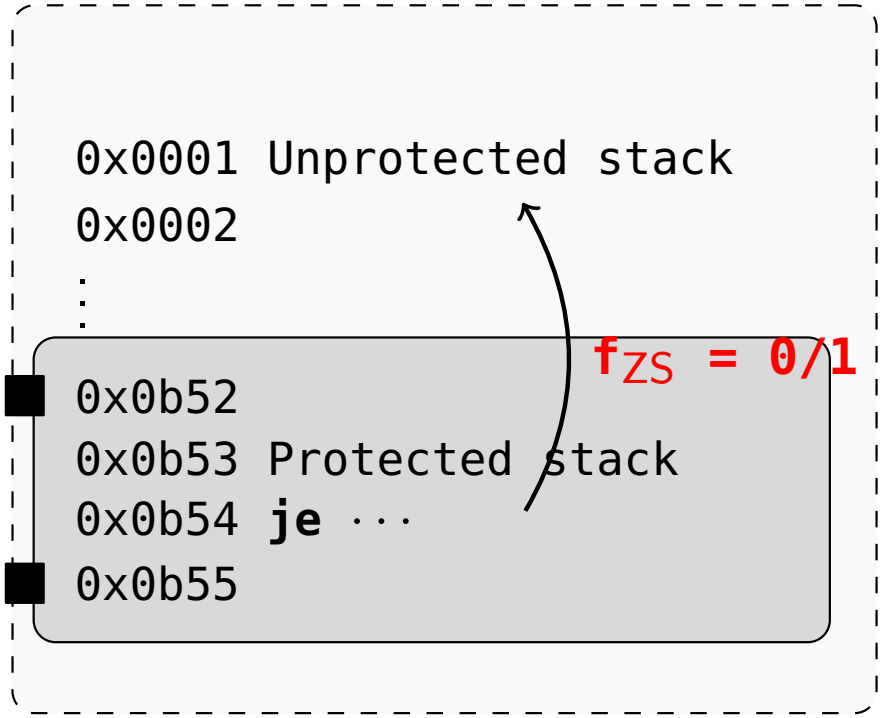
- protected stack
- returnback entry point
- reset flags and registers

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

Secure Compilation of Outcalls

Q: : Is that all?

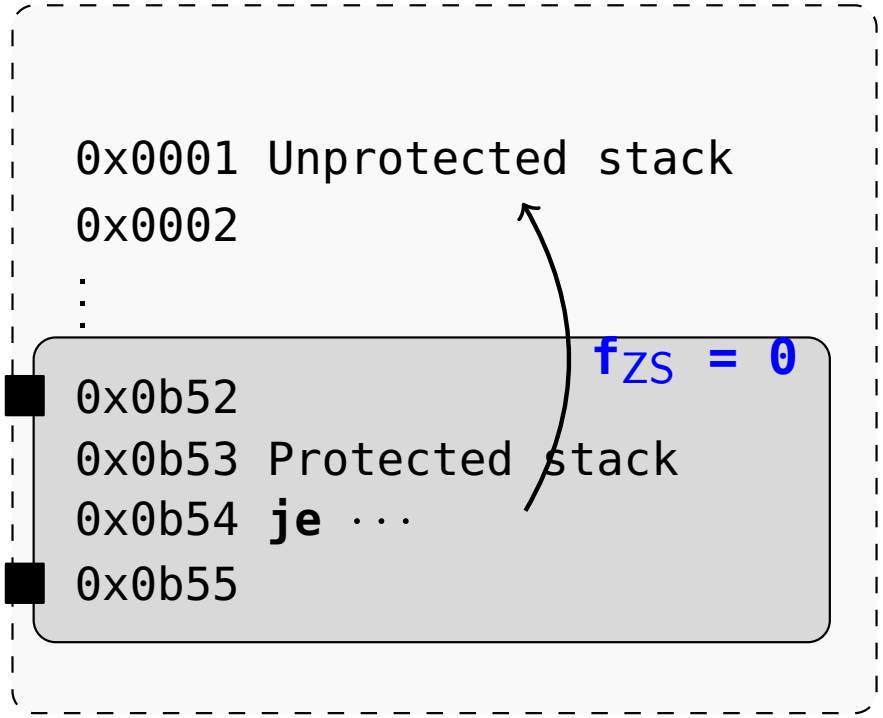
- protected stack
- returnback entry point
- reset flags and registers



Secure Compilation of Outcalls

Q: : Is that all?

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



Secure Compilation of Outcalls

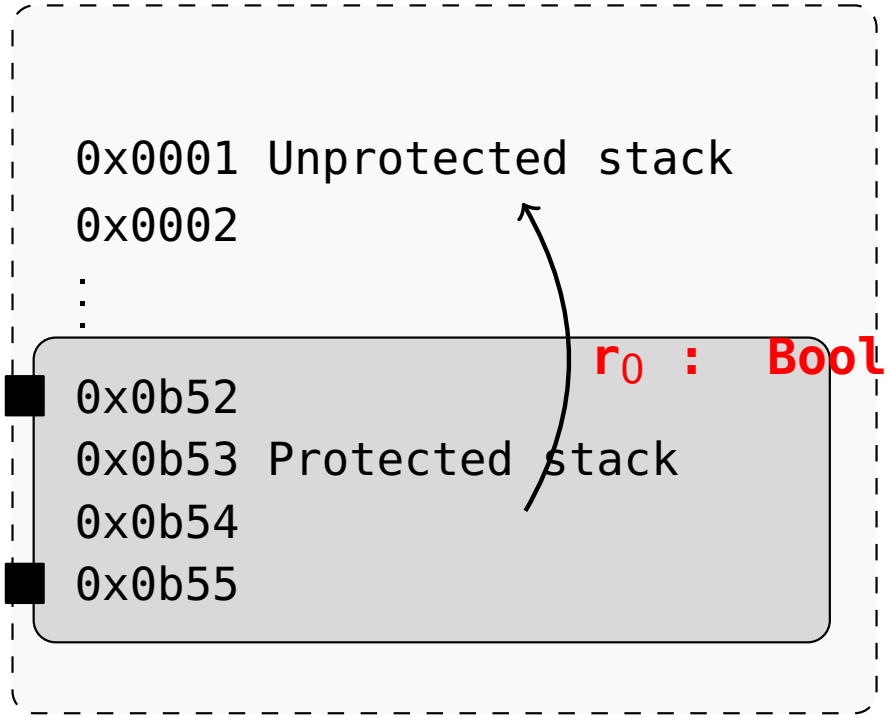
Q: : Is that all?

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

Secure Compilation of Outcalls

Q: : Is that all?

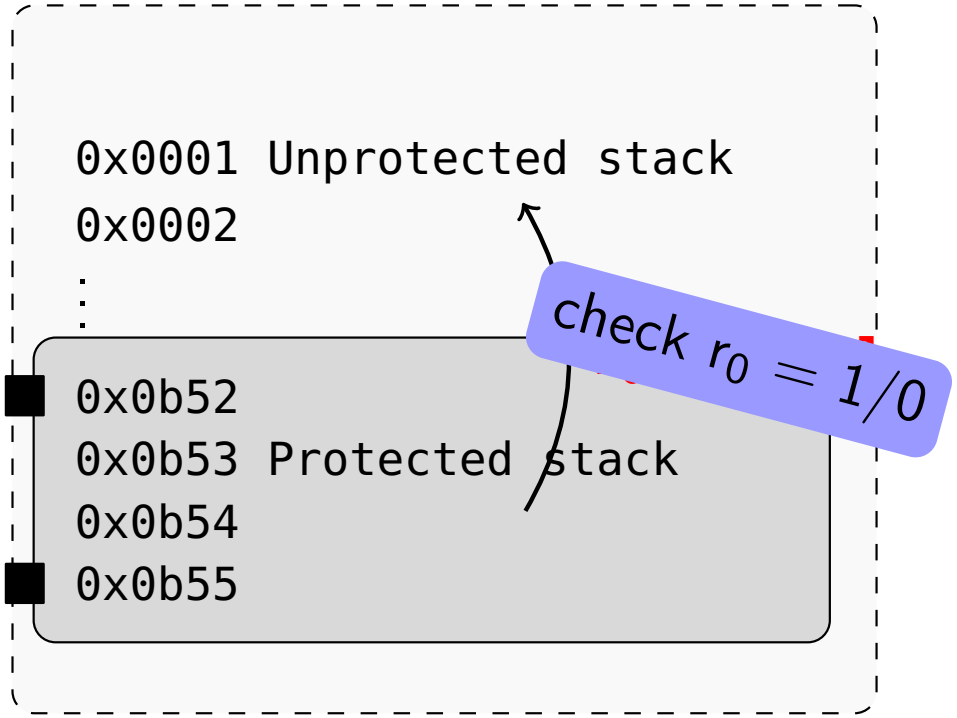
- protected stack
- returnback entry point
- reset flags and registers
- ground-typed values check



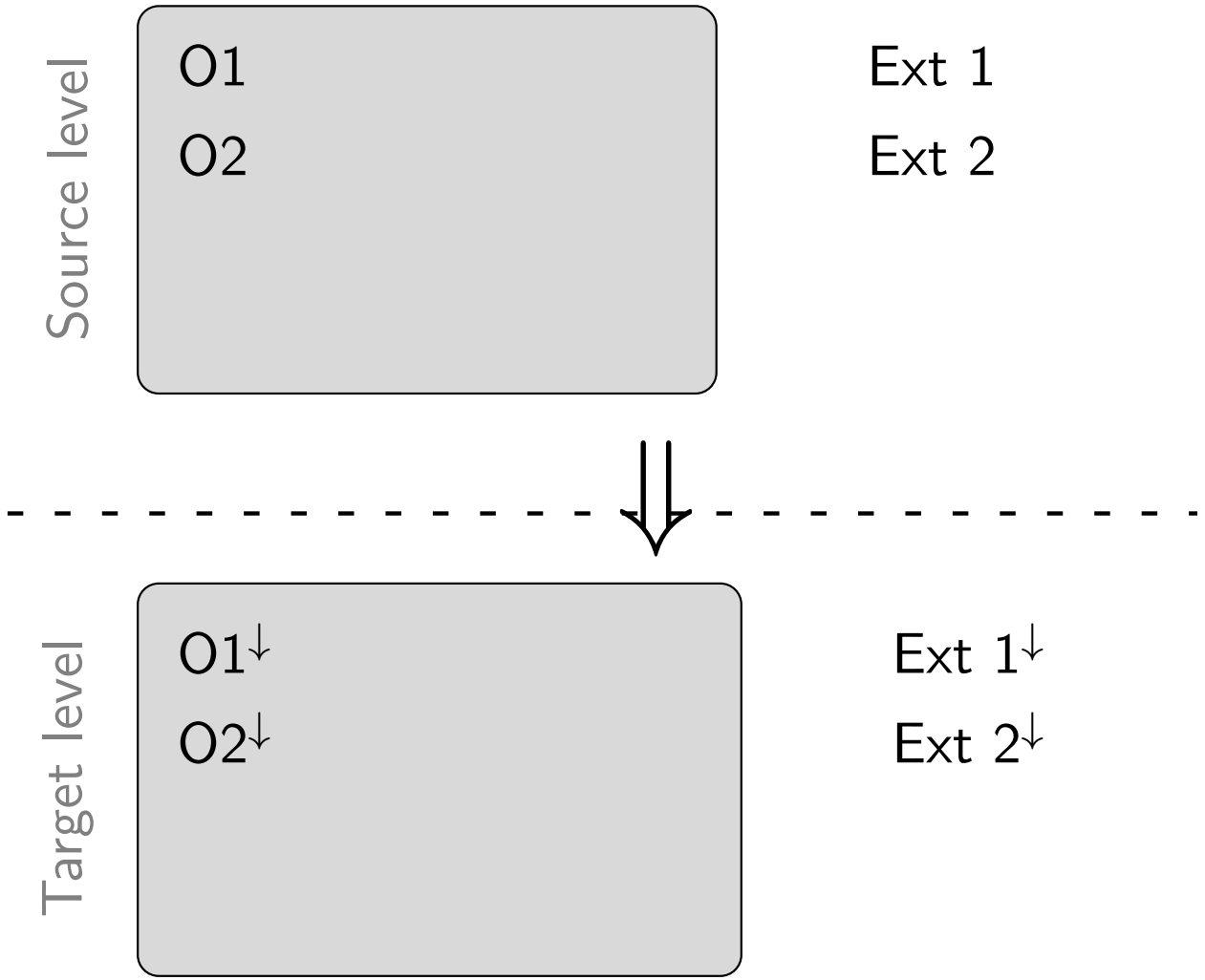
Secure Compilation of Outcalls

Q: : Is that all?

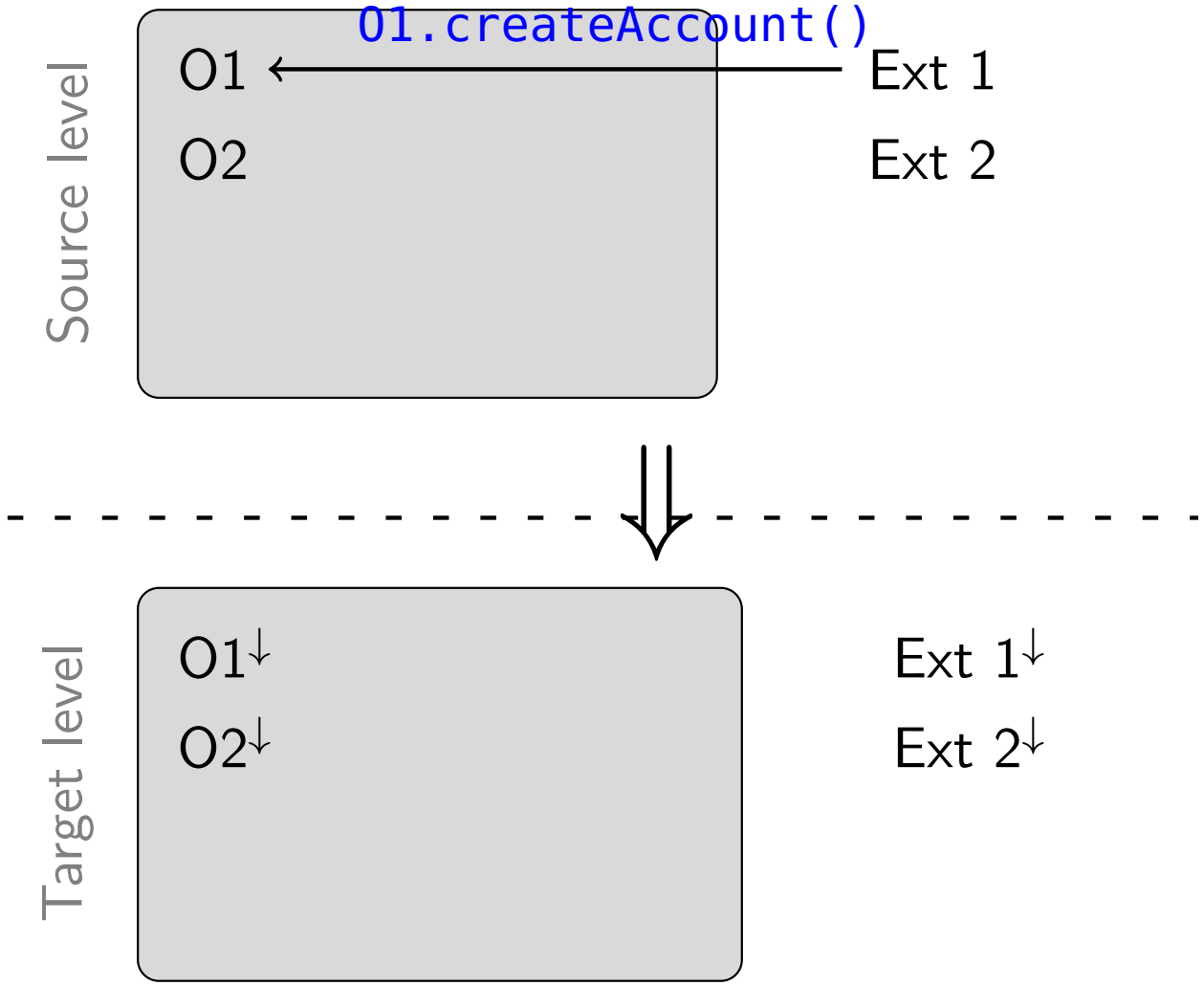
- protected stack
- returnback entry point
- reset flags and registers
- ground-typed values check



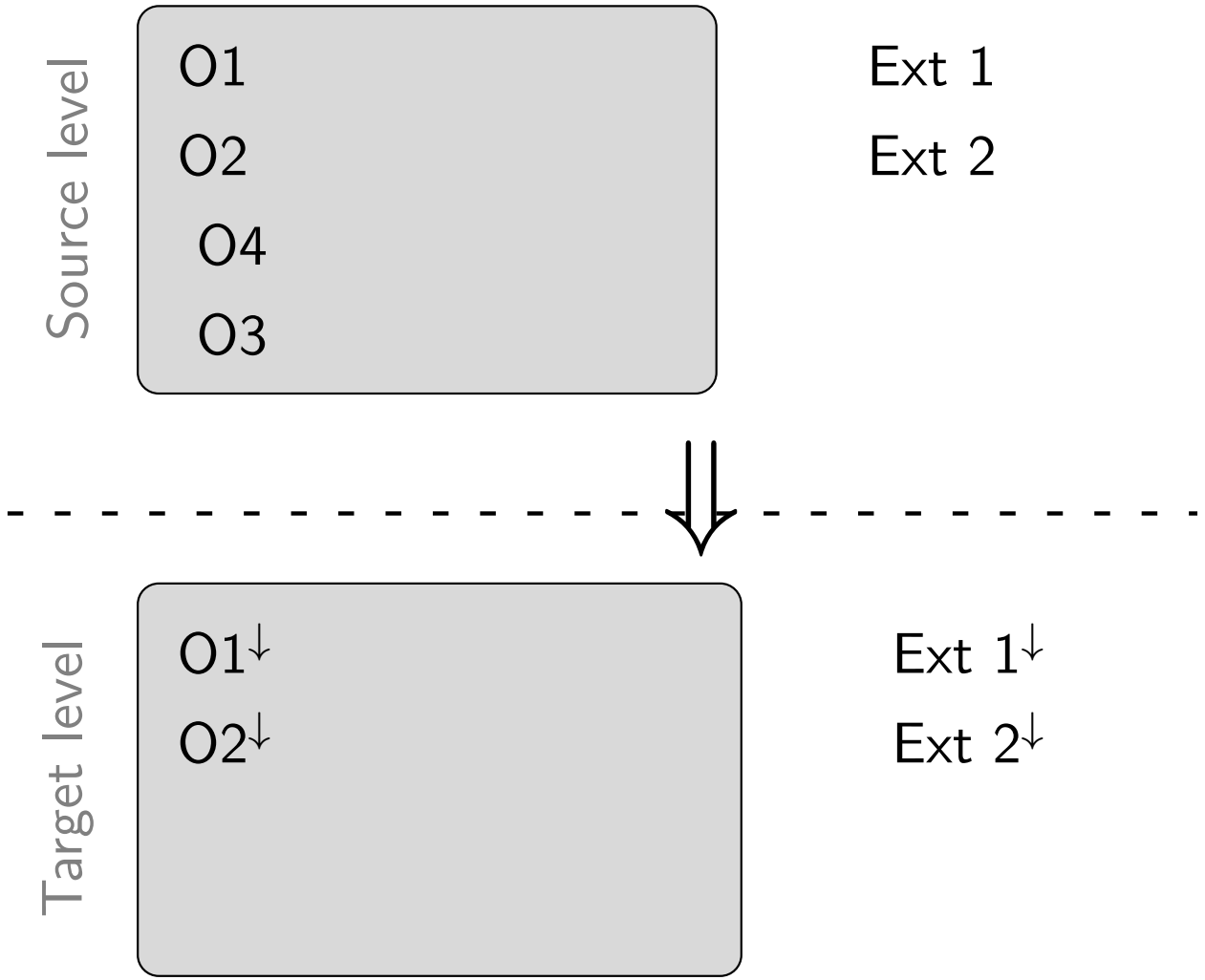
Dynamic Memory Allocation



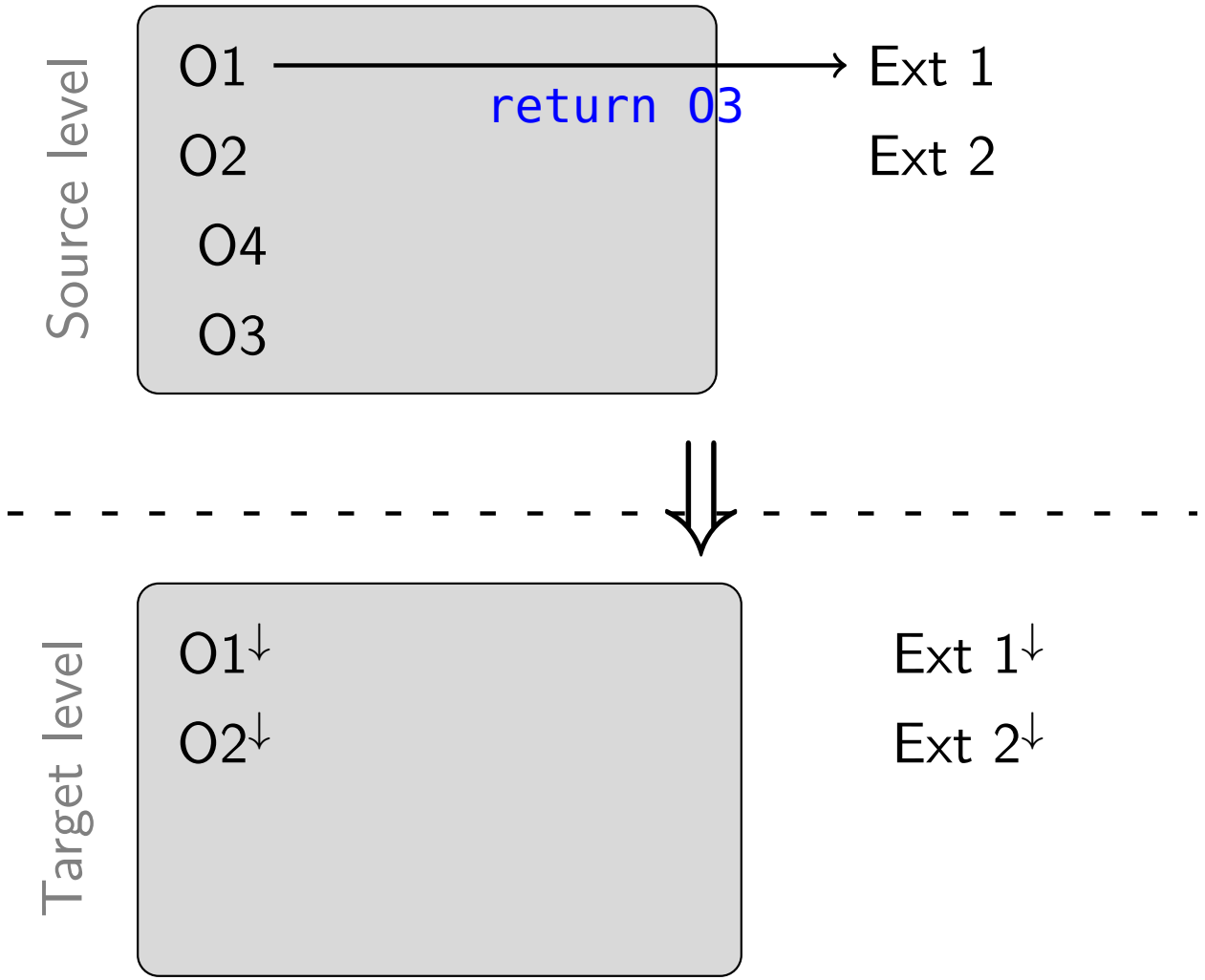
Dynamic Memory Allocation



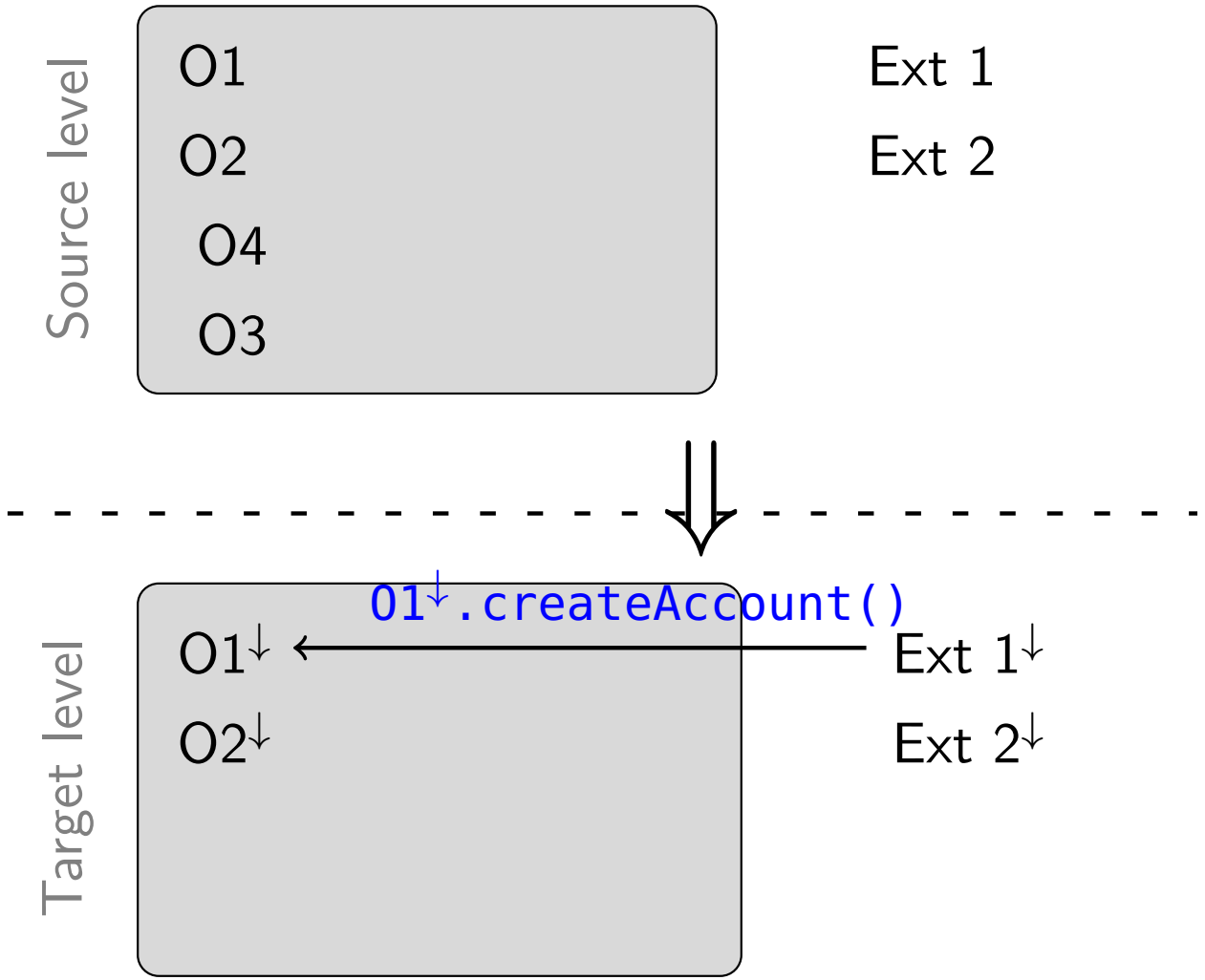
Dynamic Memory Allocation



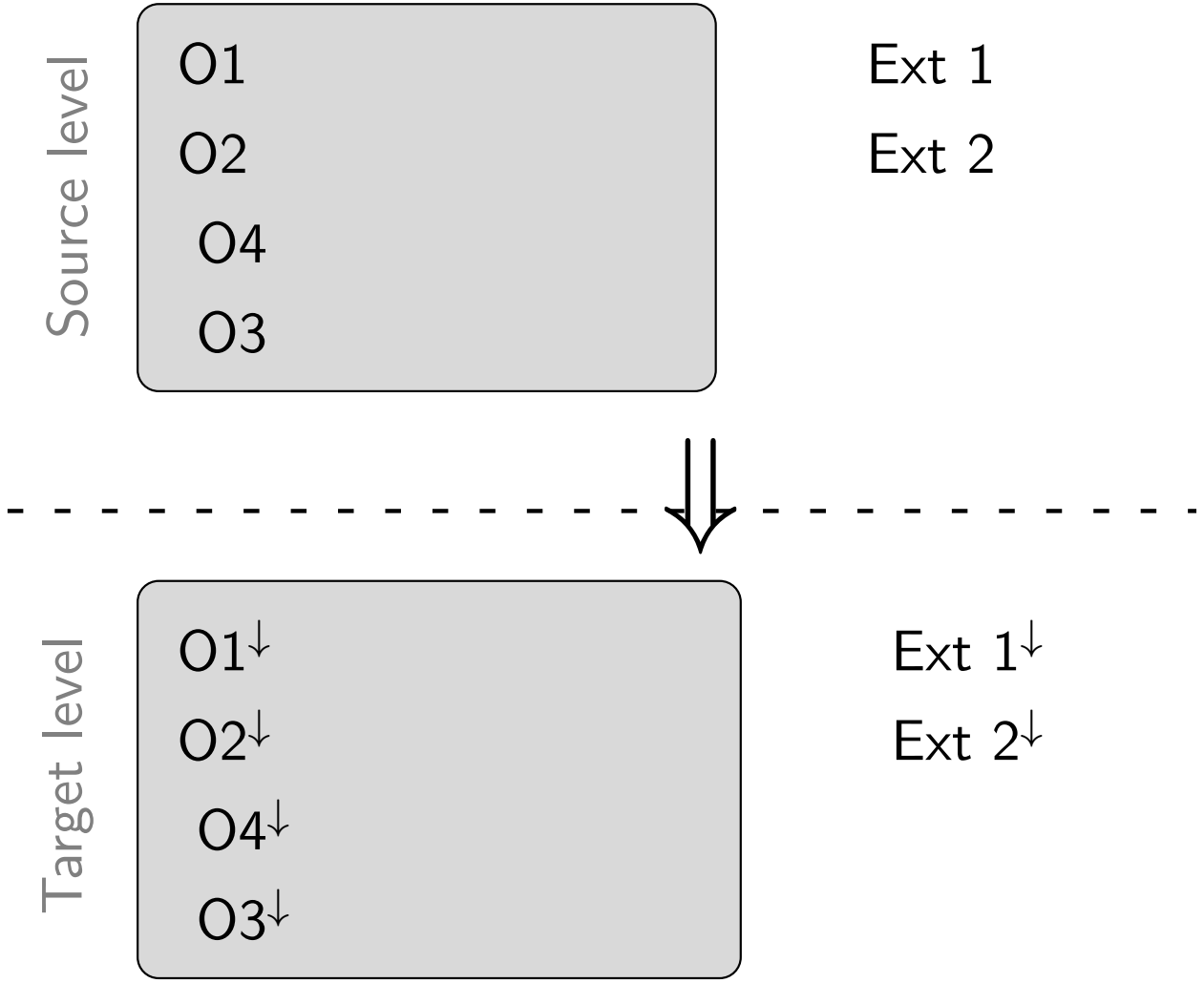
Dynamic Memory Allocation



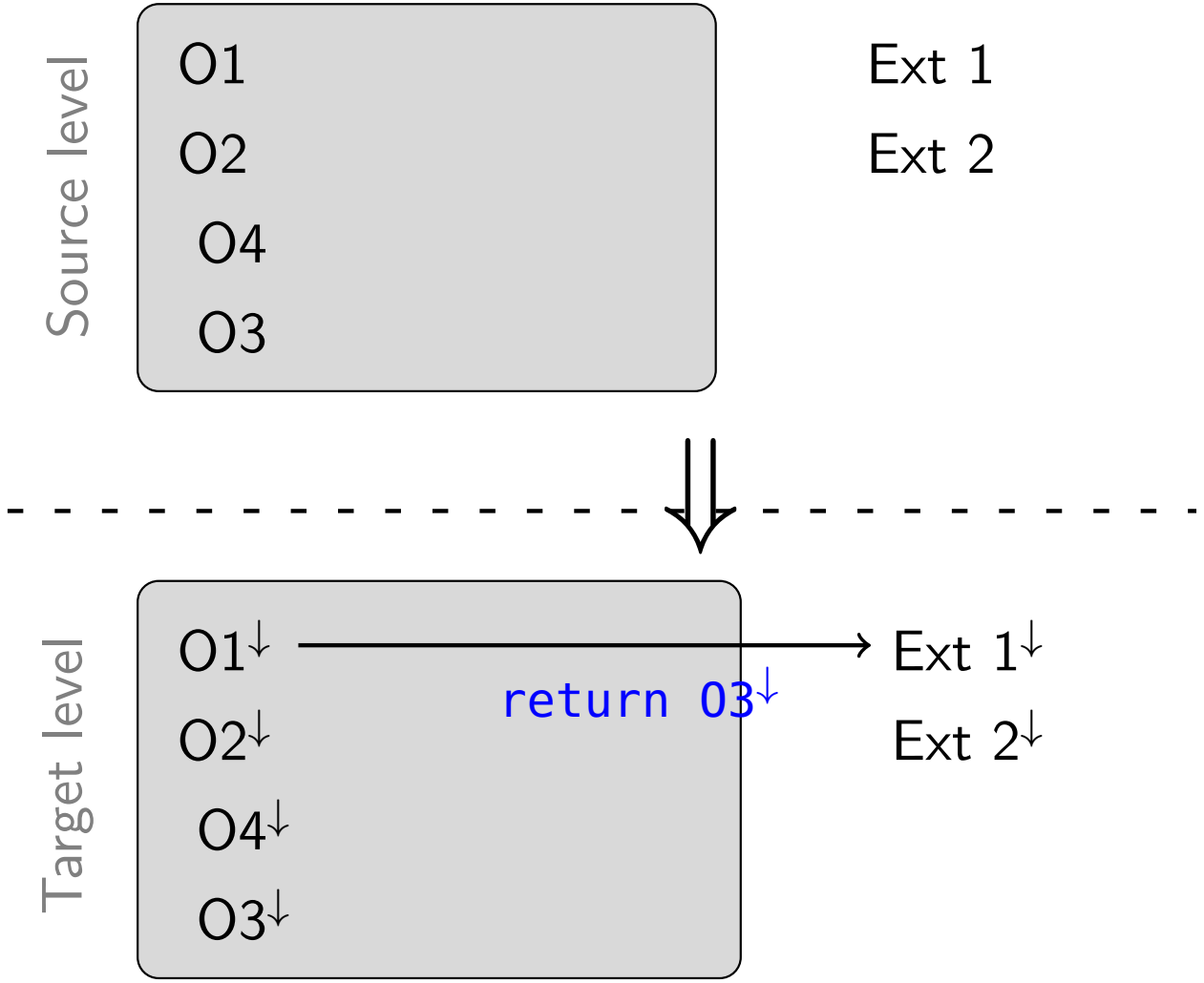
Dynamic Memory Allocation



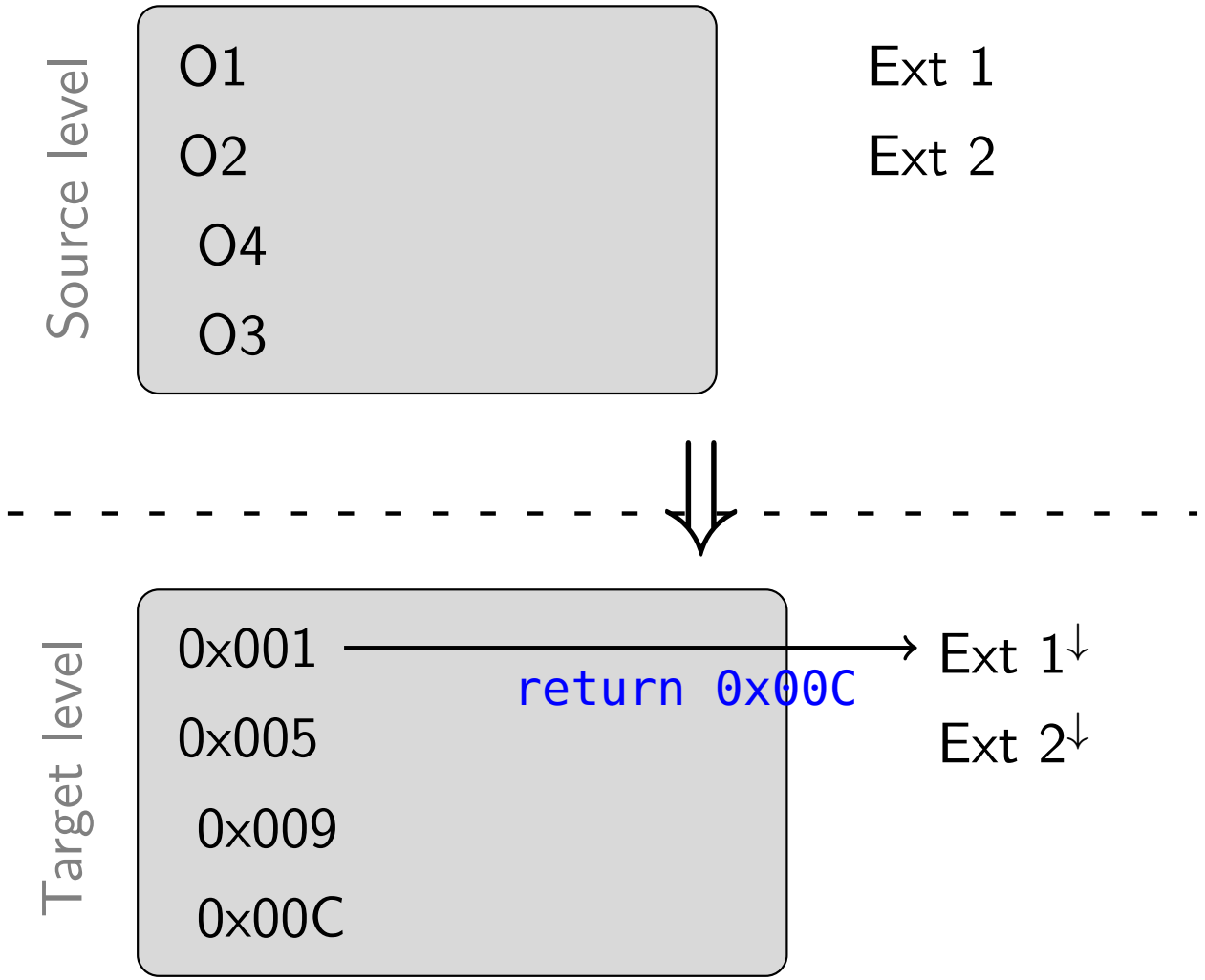
Dynamic Memory Allocation



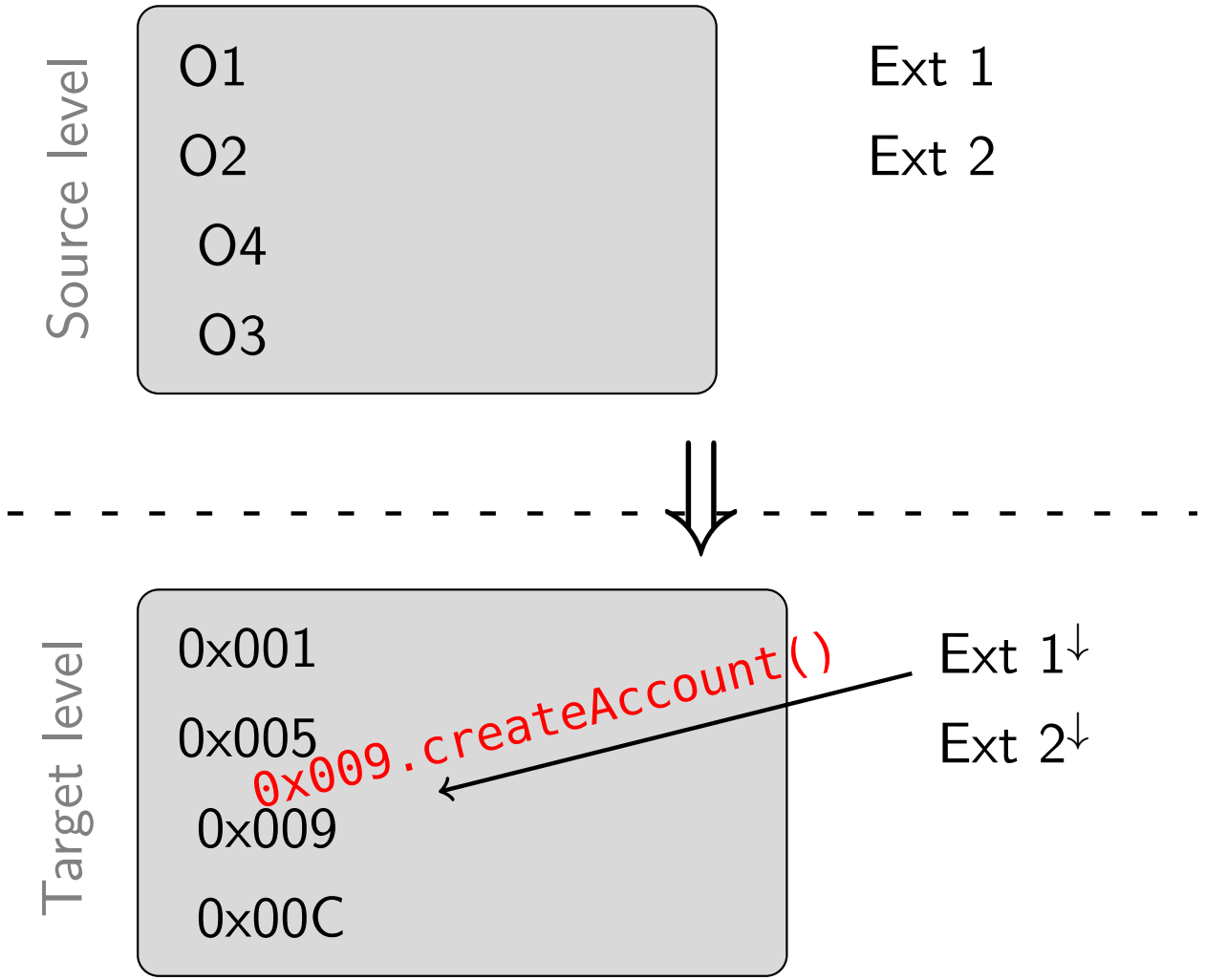
Dynamic Memory Allocation



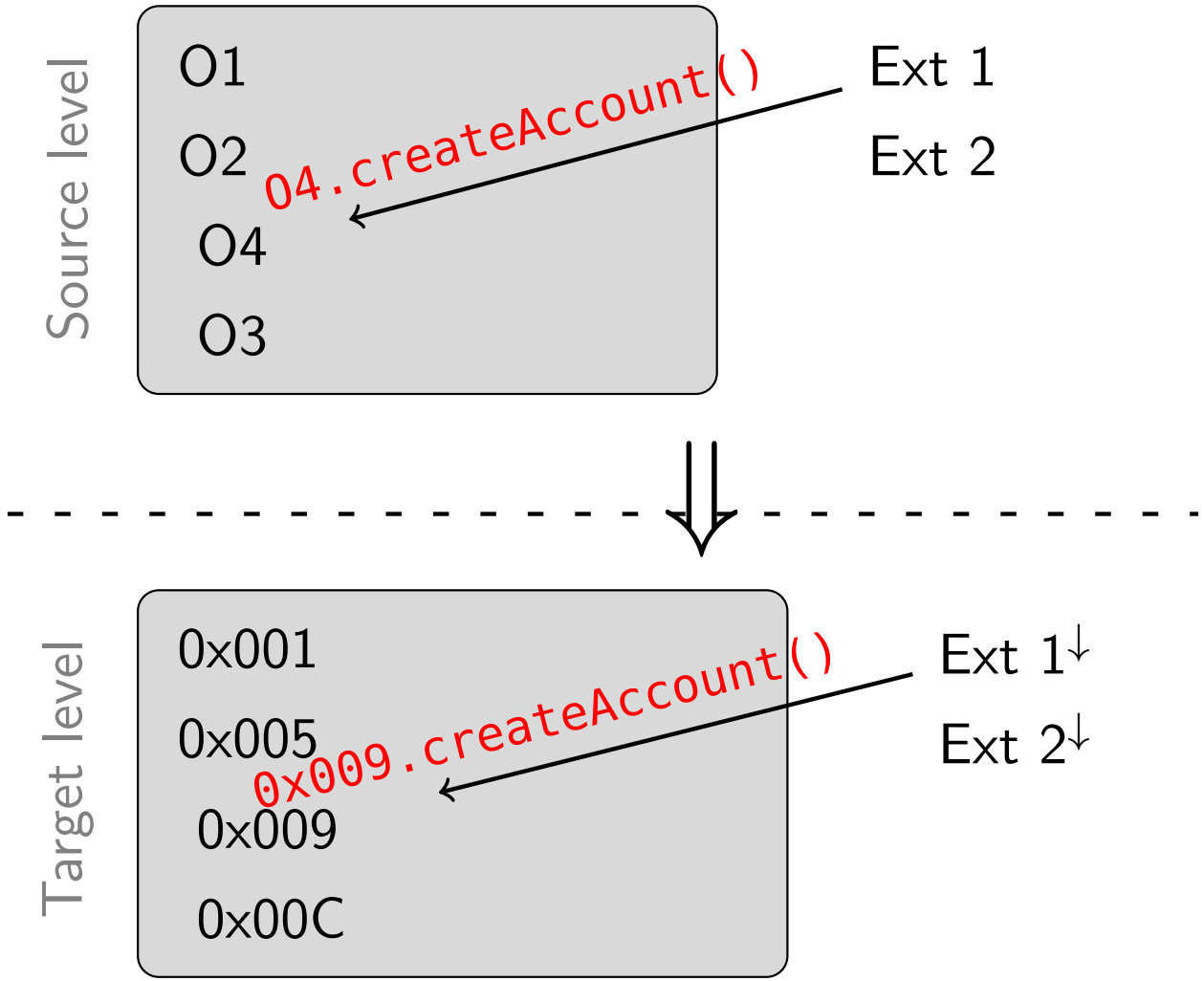
Dynamic Memory Allocation



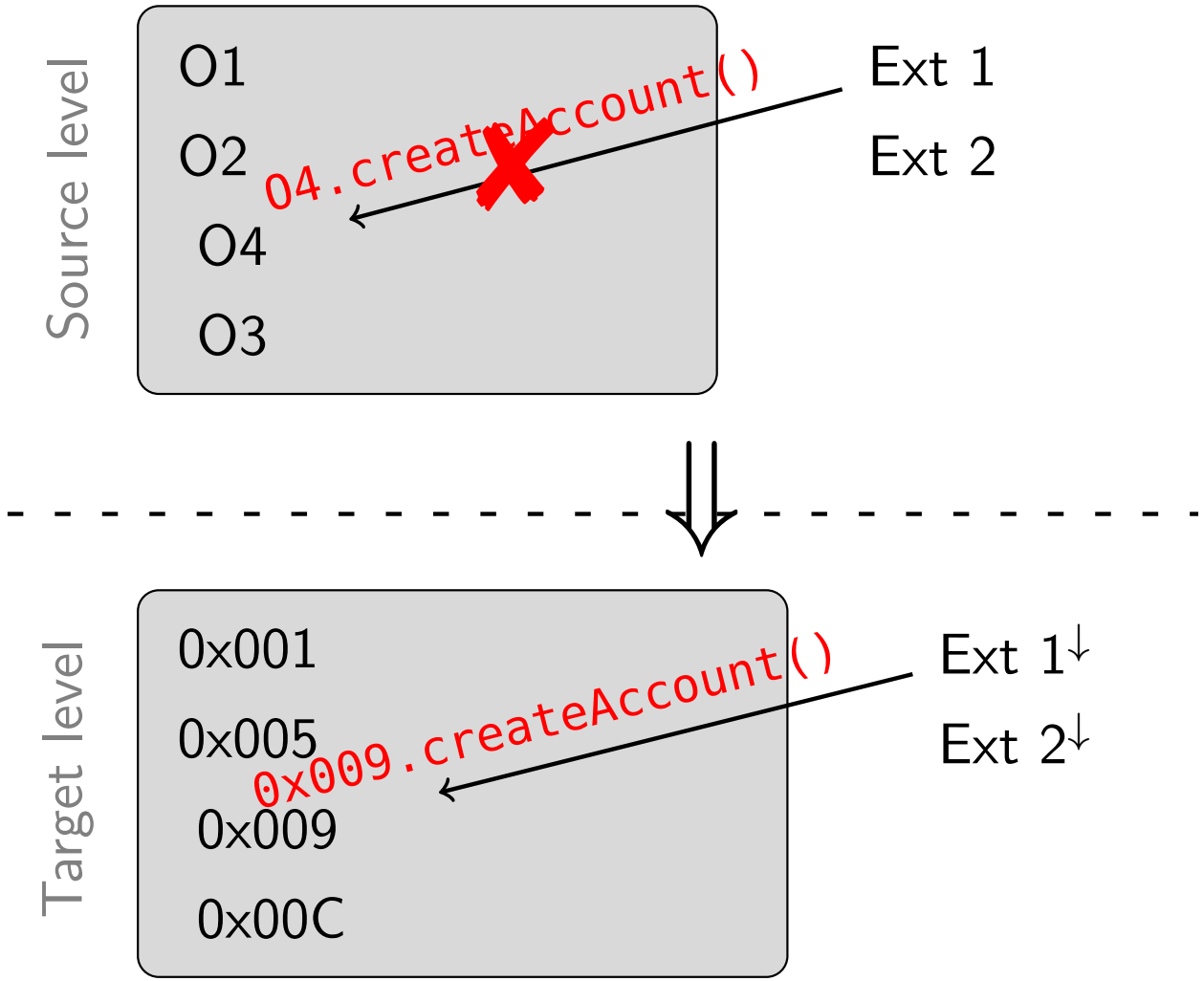
Dynamic Memory Allocation



Dynamic Memory Allocation

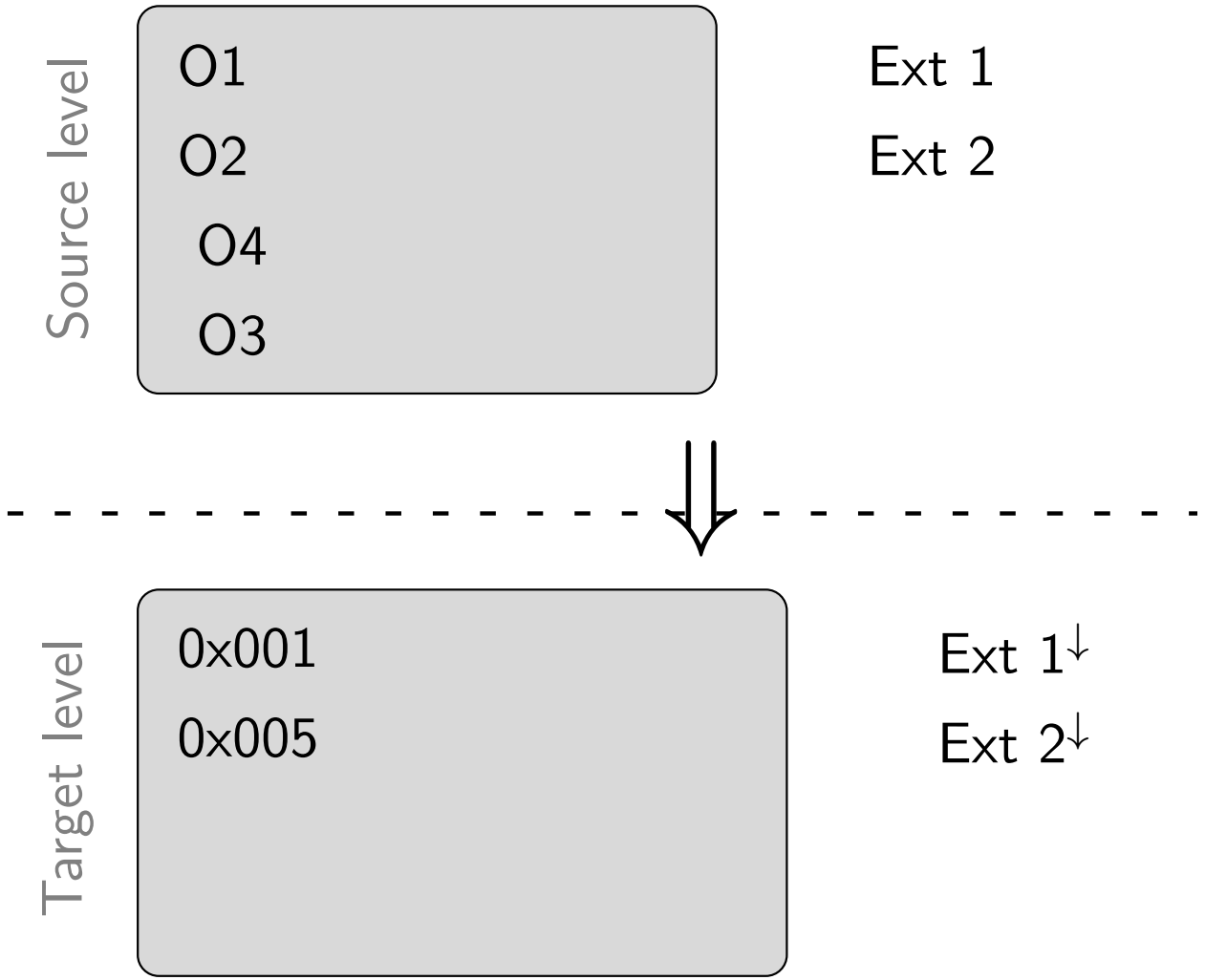


Dynamic Memory Allocation



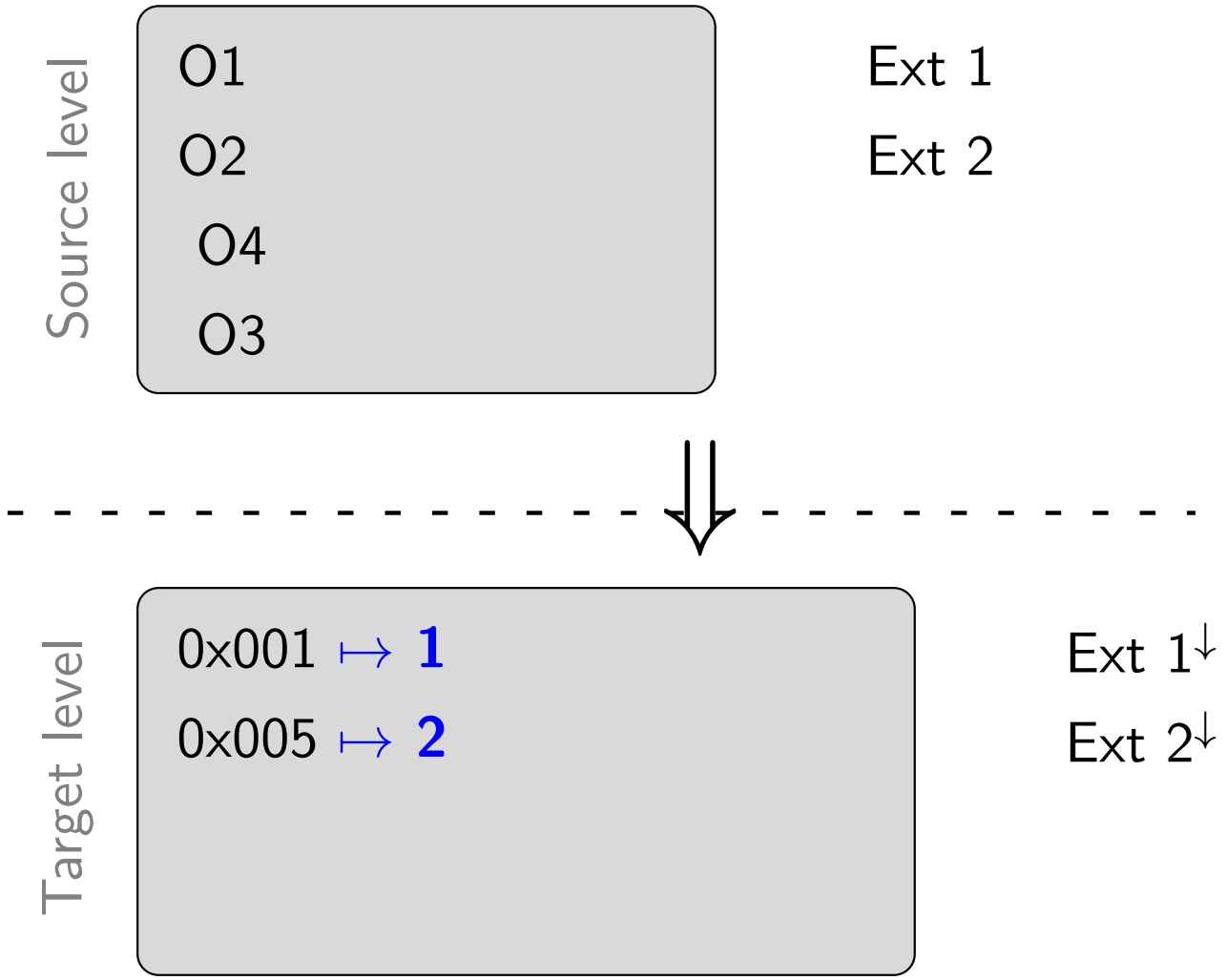
- Object id guessing

Dynamic Memory Allocation



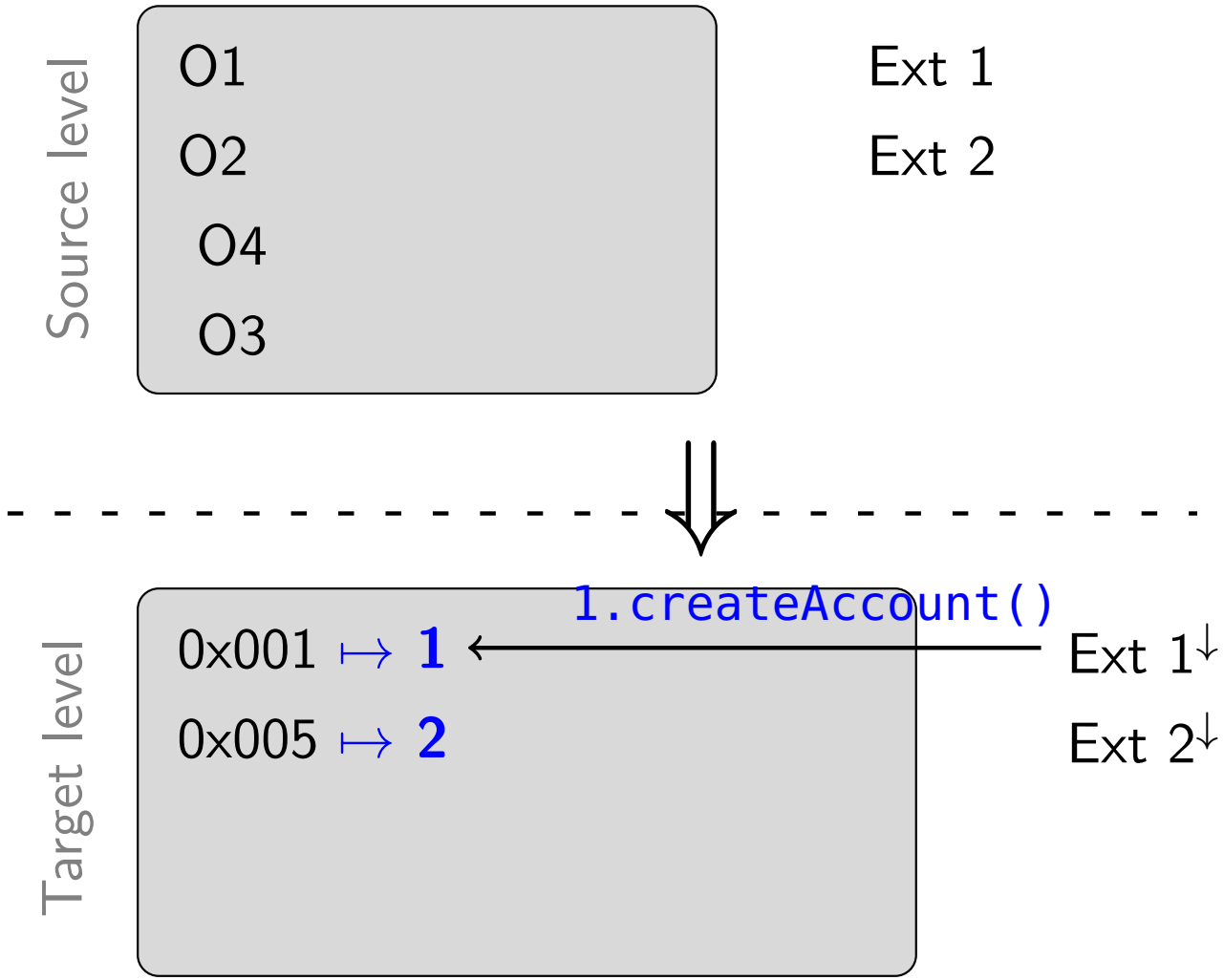
- Object id guessing
- map Oid to natural numbers

Dynamic Memory Allocation



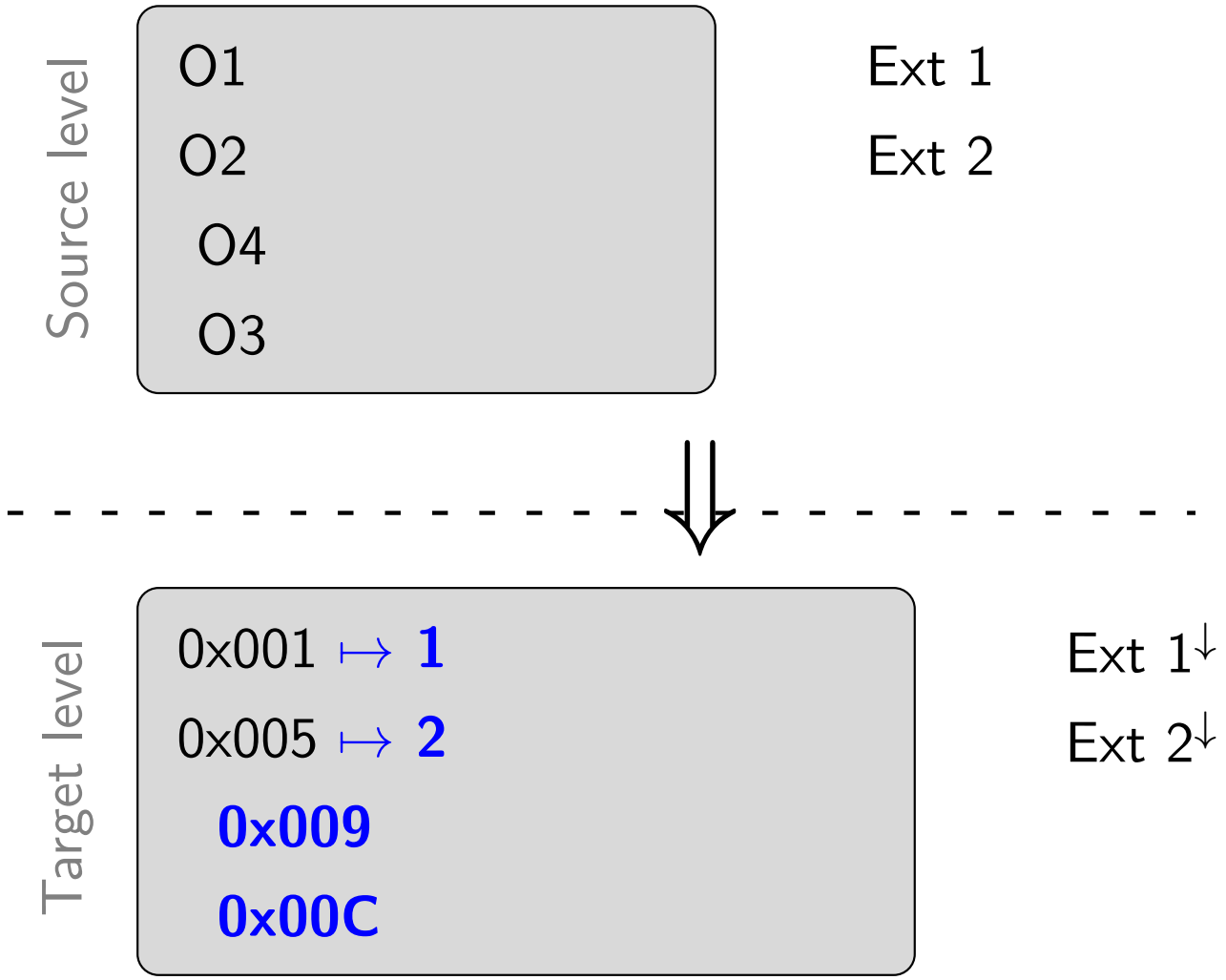
- Object id guessing
- map Oid to natural numbers

Dynamic Memory Allocation



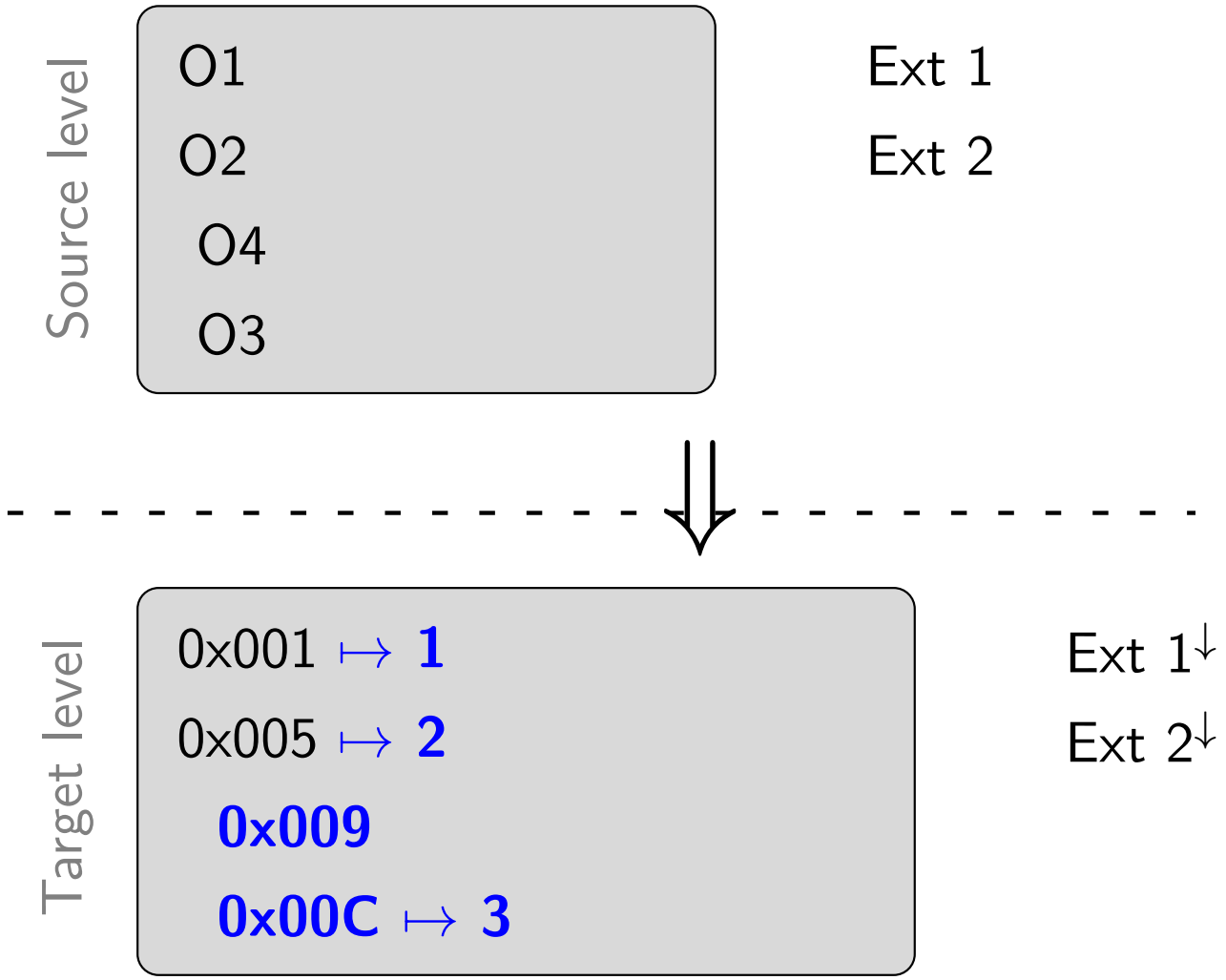
- Object id guessing
- map Oid to natural numbers

Dynamic Memory Allocation



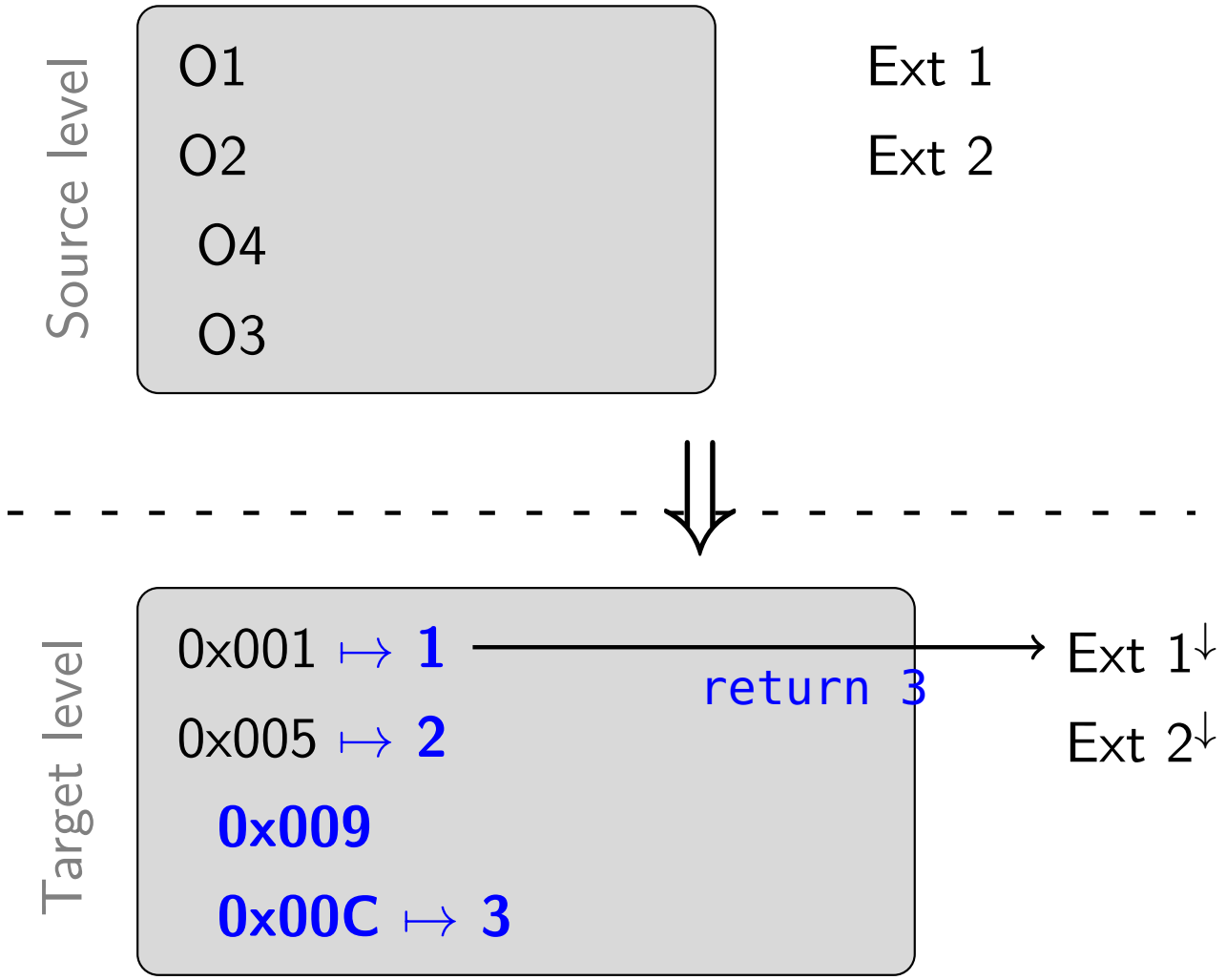
- Object id guessing
- map Oid to natural numbers

Dynamic Memory Allocation



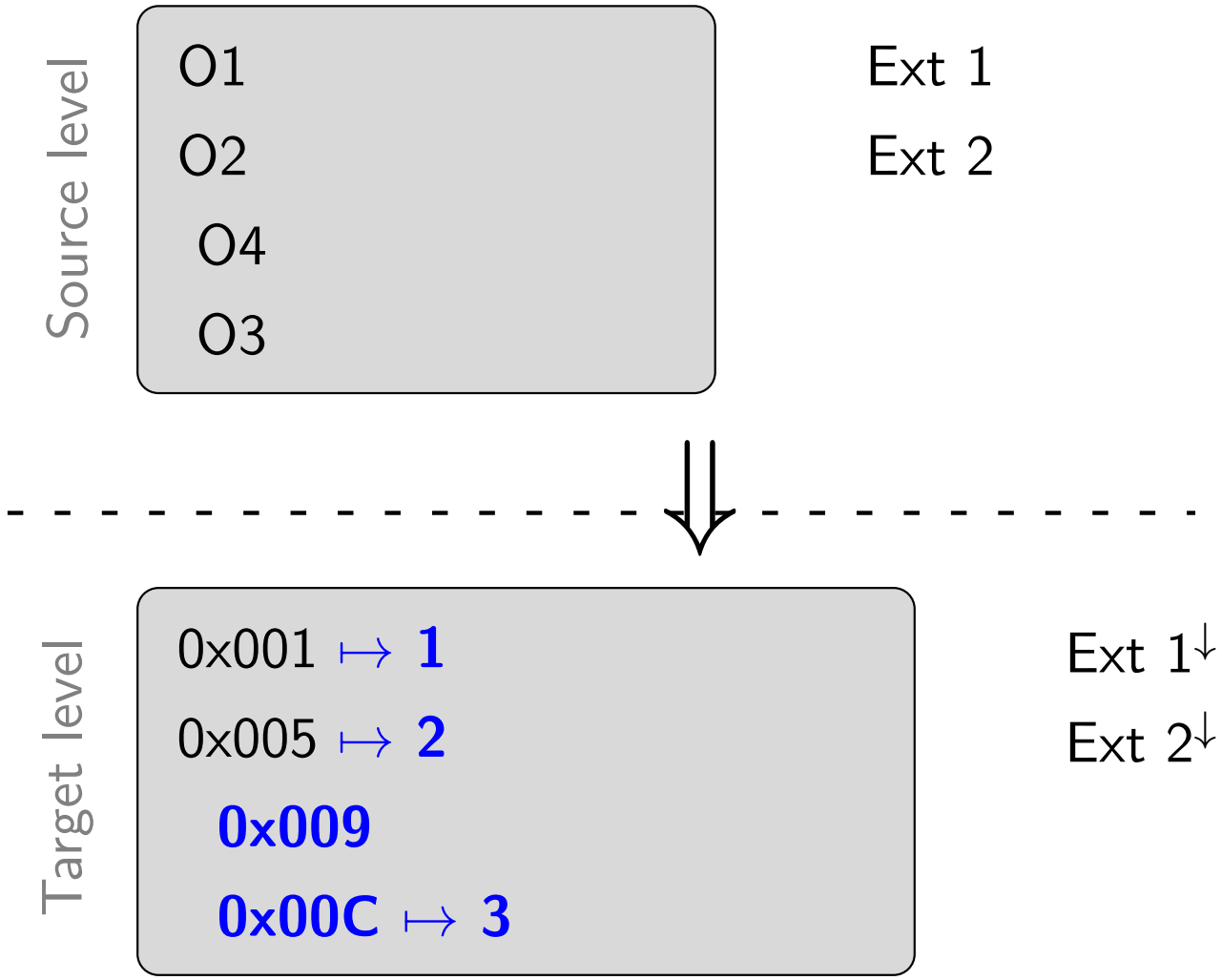
- Object id guessing
- map Oid to natural numbers
- add Oid to map

Dynamic Memory Allocation



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

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

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

$\Downarrow$

Target level

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

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

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

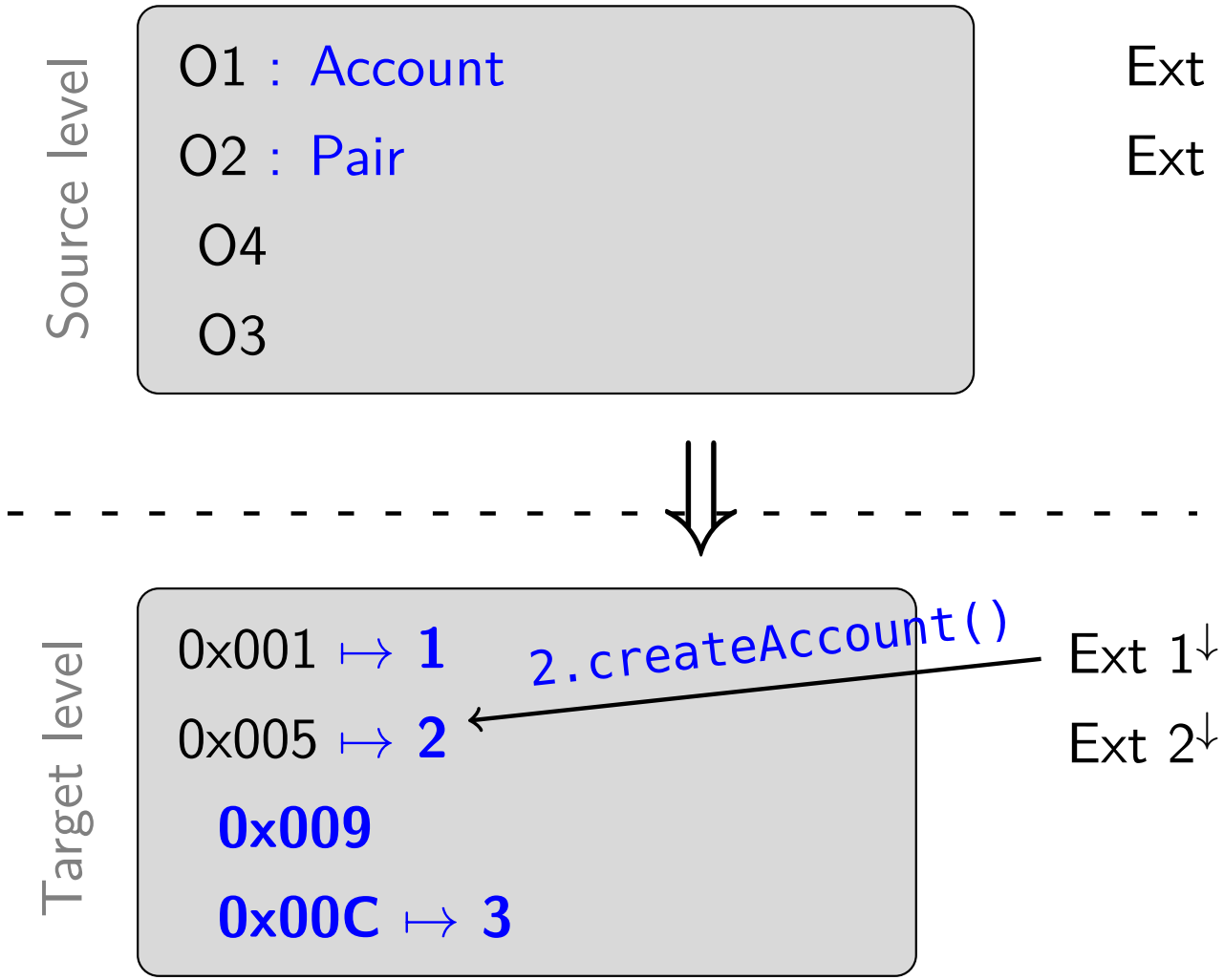
$\Downarrow$

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

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

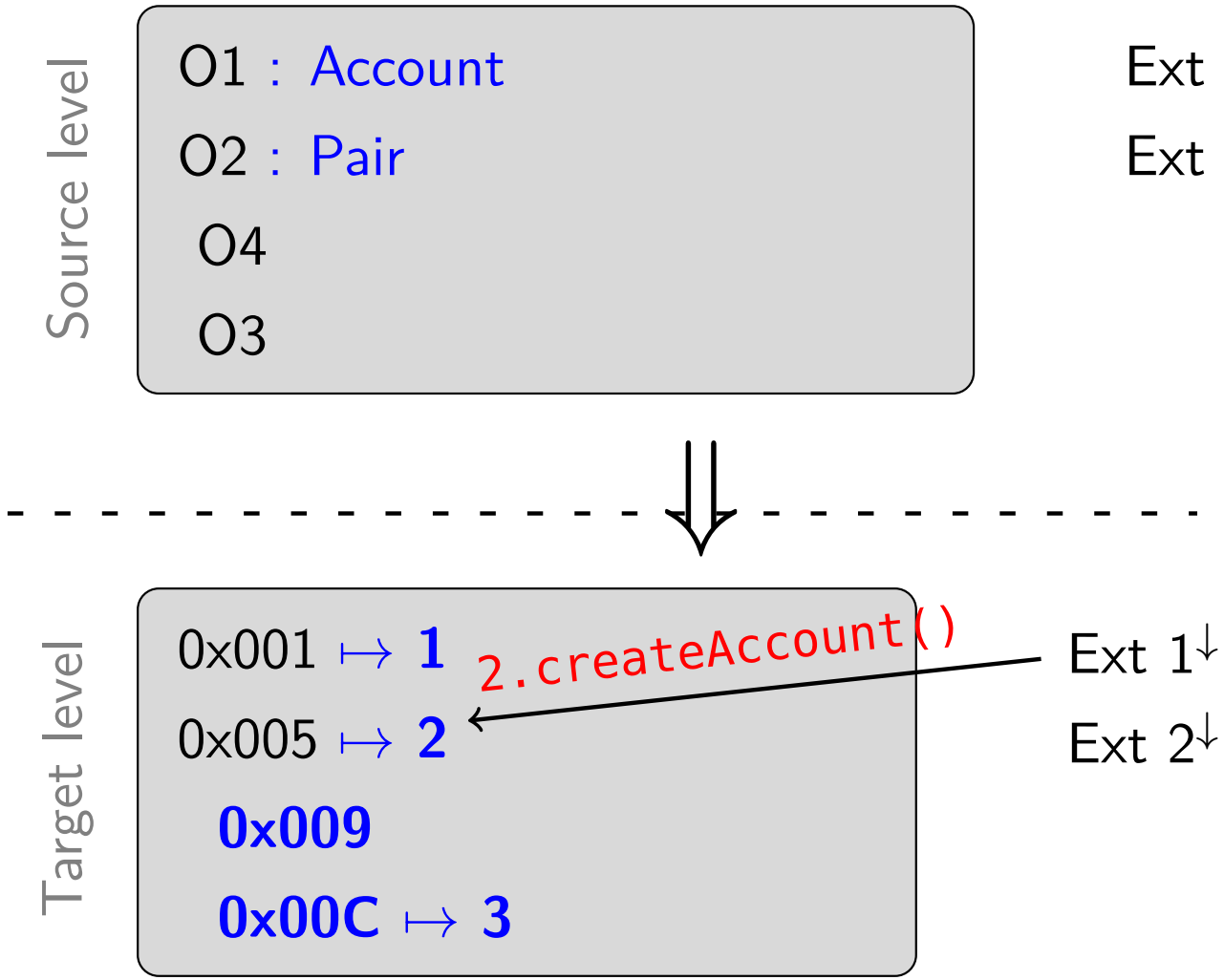
$\Downarrow$

Target level

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

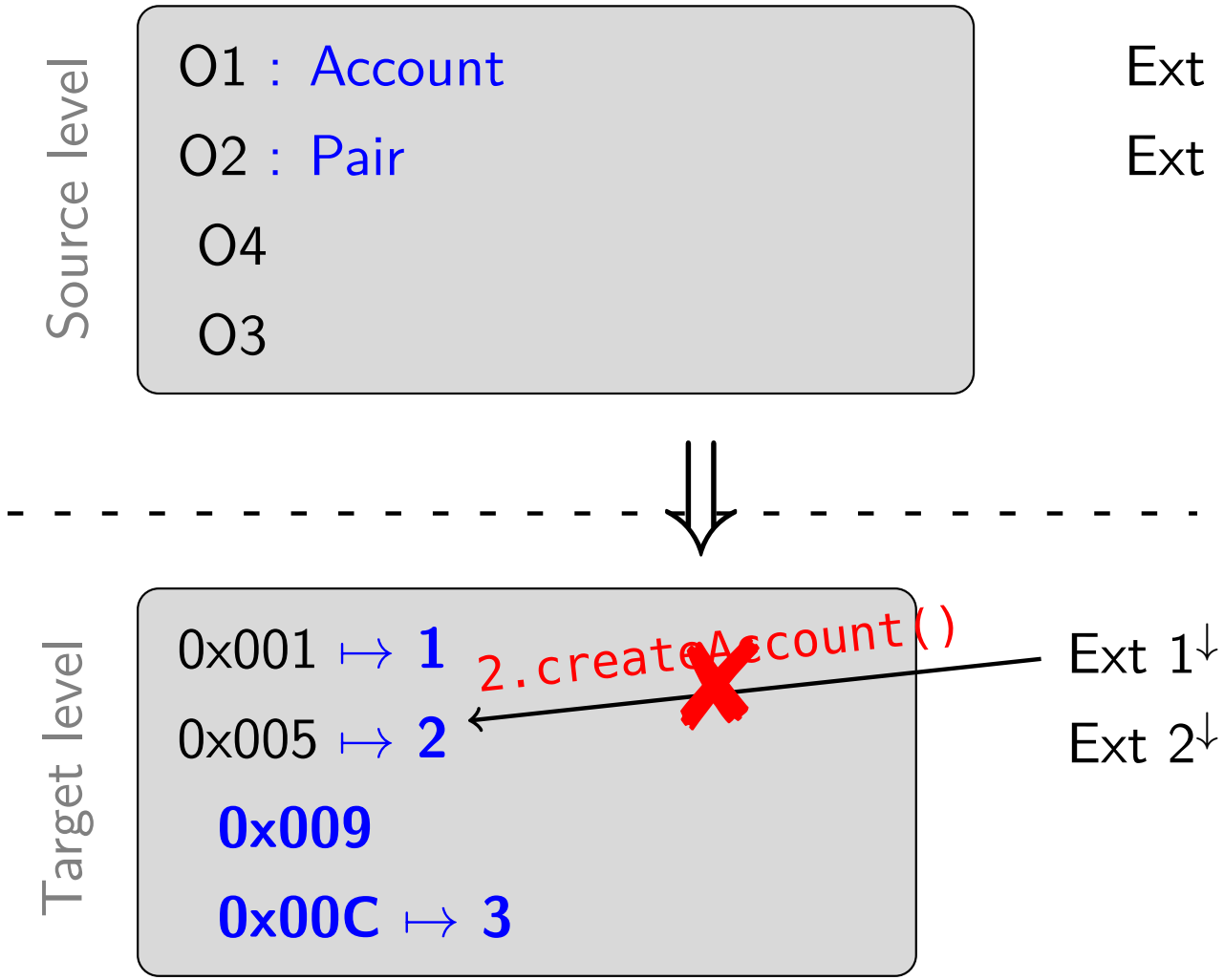
Ext $1^{\downarrow}$
Ext $2^{\downarrow}$

Dynamic Memory Allocation



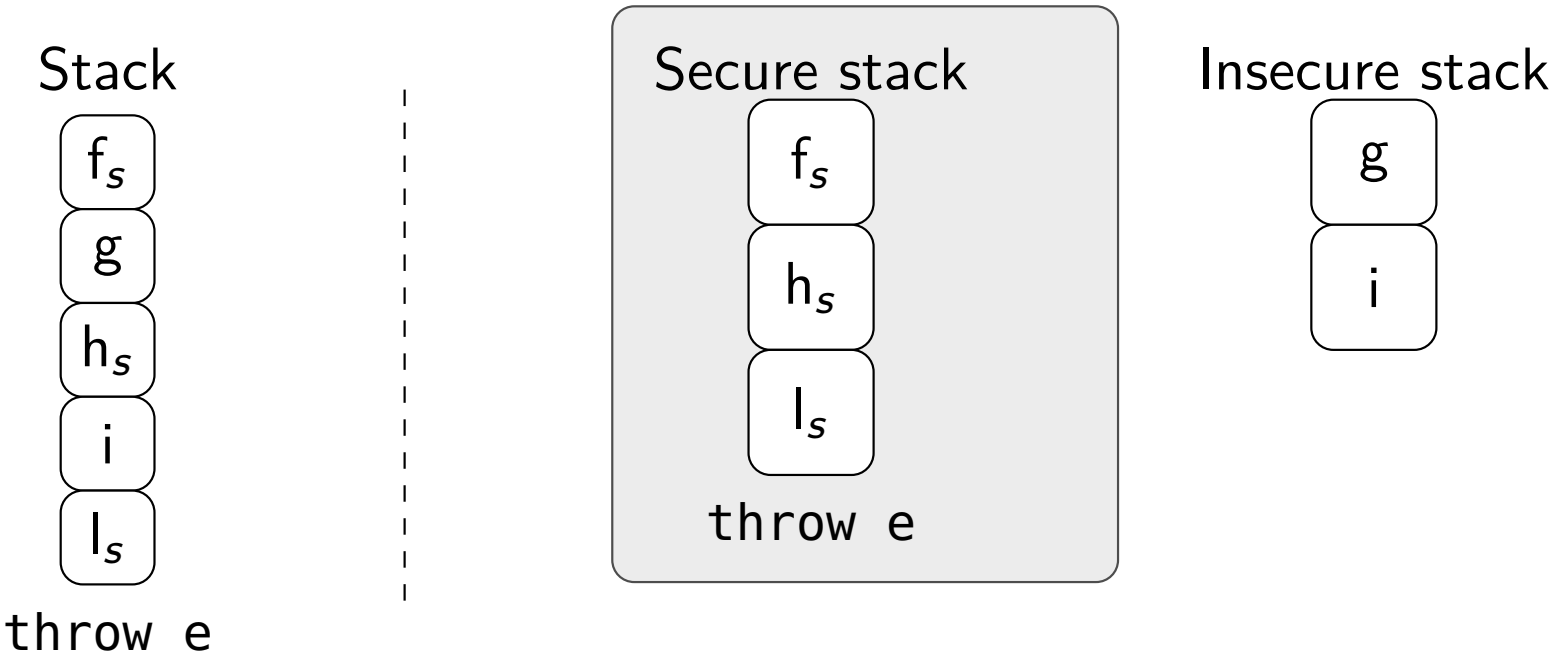
- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup ($\mathbf{O}(1)$) when number is received
- dynamic typecheck for: current object arguments

Dynamic Memory Allocation

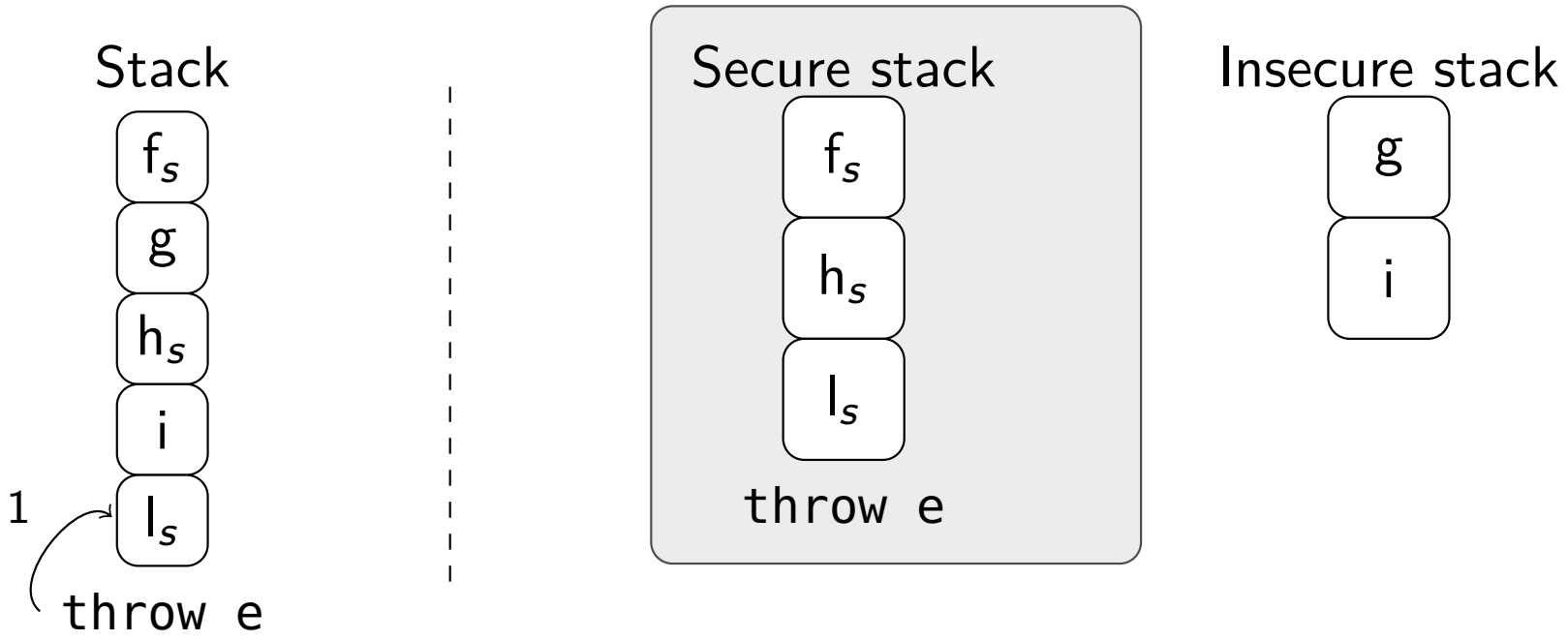


- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup ($\mathbf{O}(1)$) when number is received
- dynamic typecheck for: current object arguments
- no need of extra information

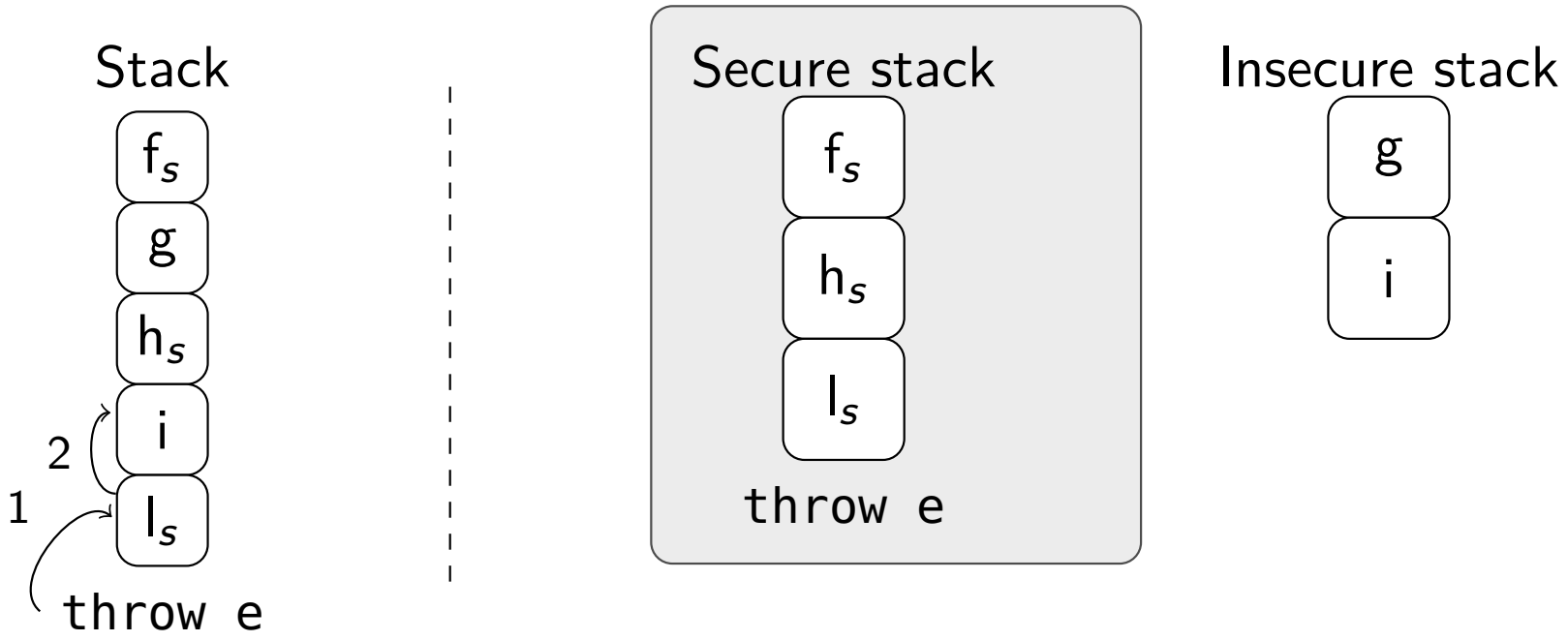
Exceptions


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

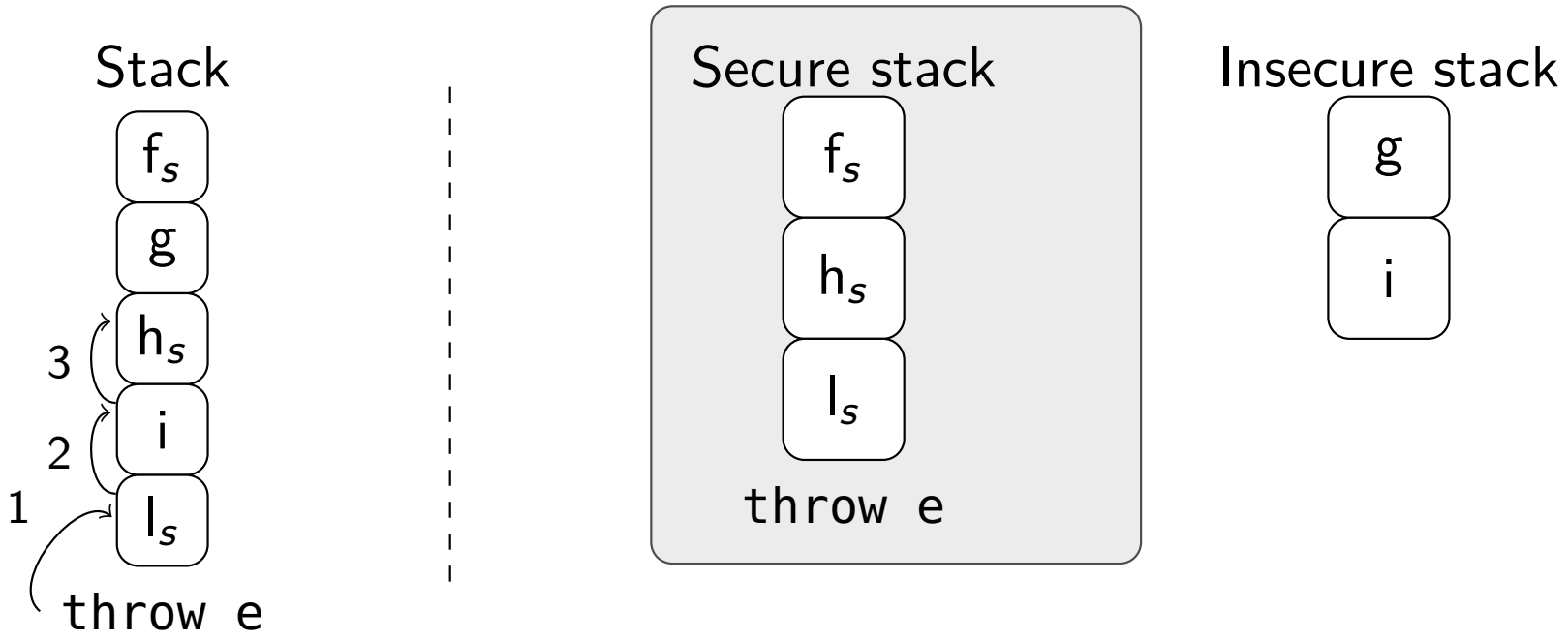
Exceptions


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

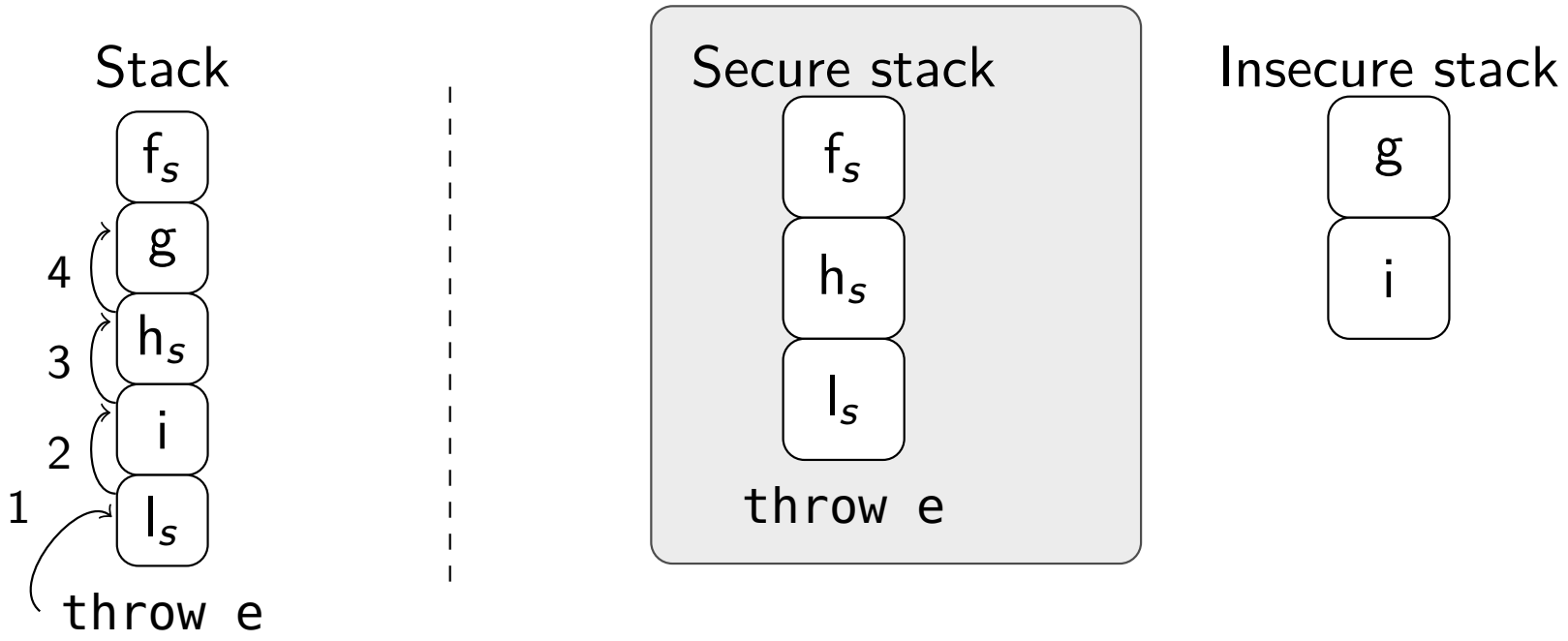
Exceptions


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

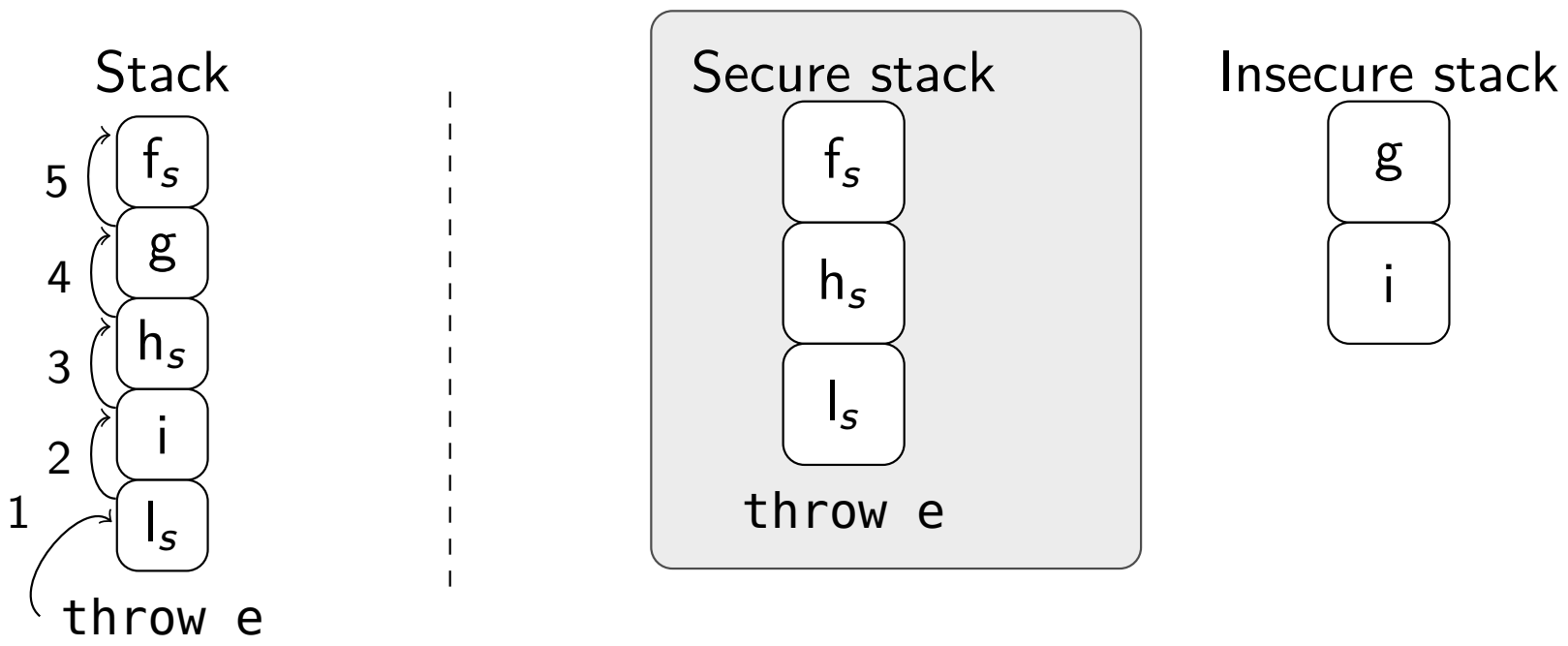
Exceptions


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

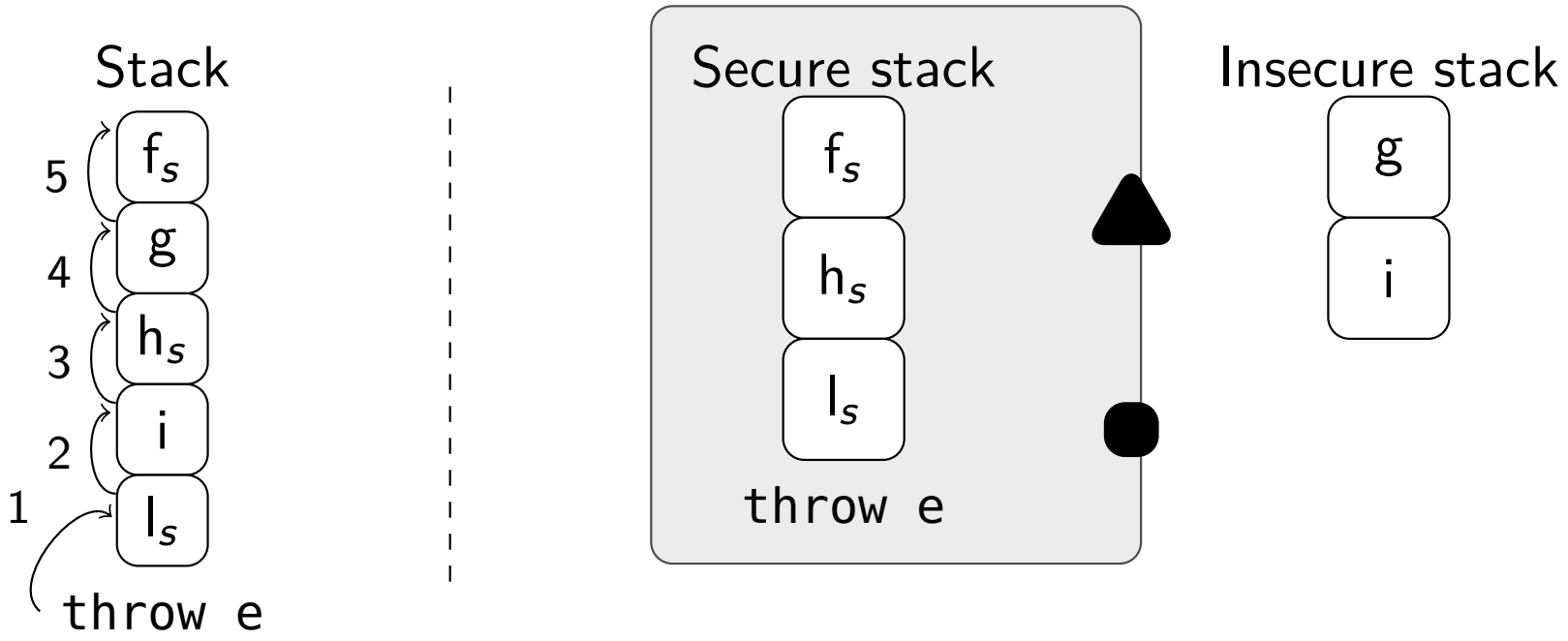
Exceptions


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

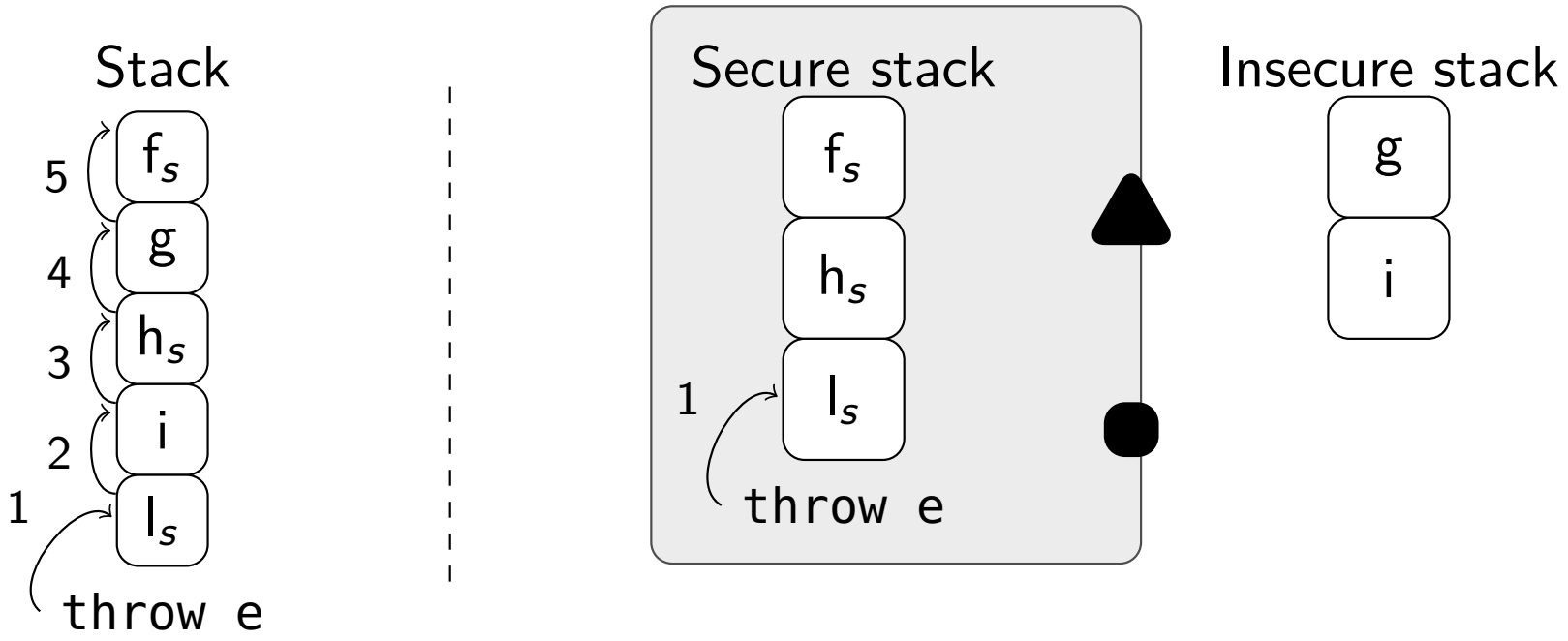
Exceptions


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

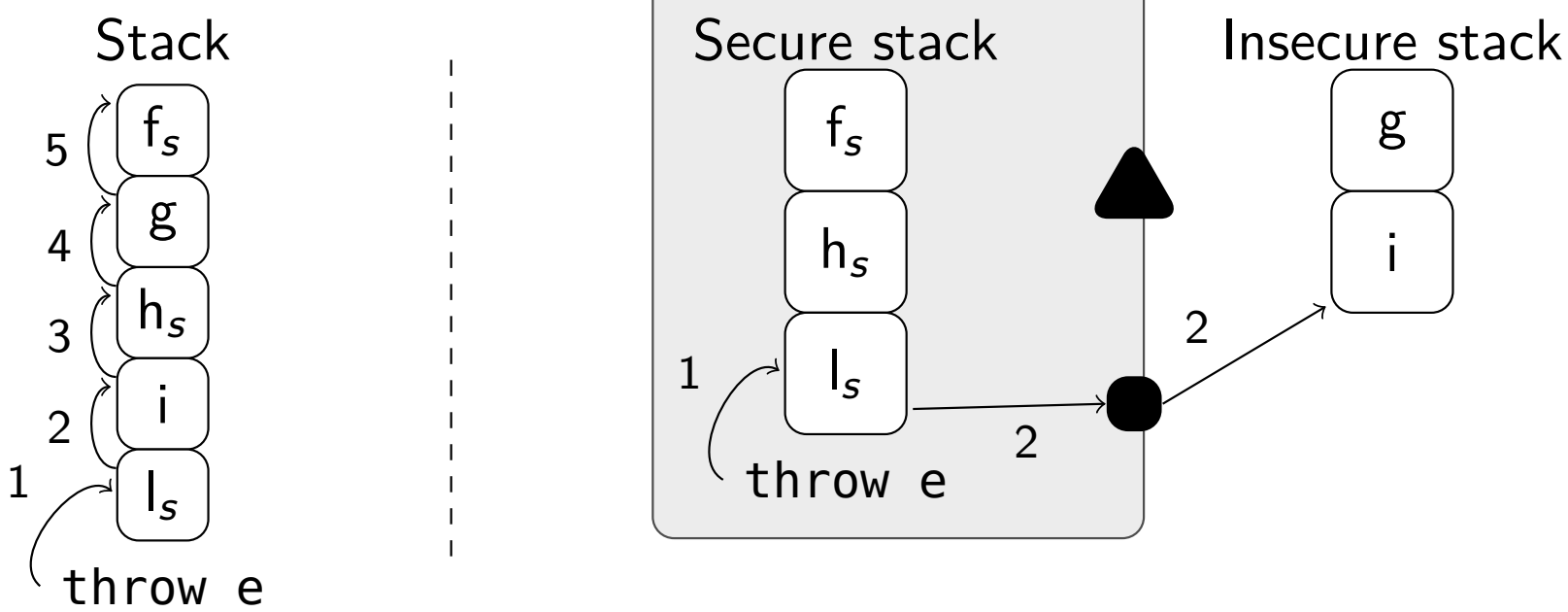
Exceptions


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

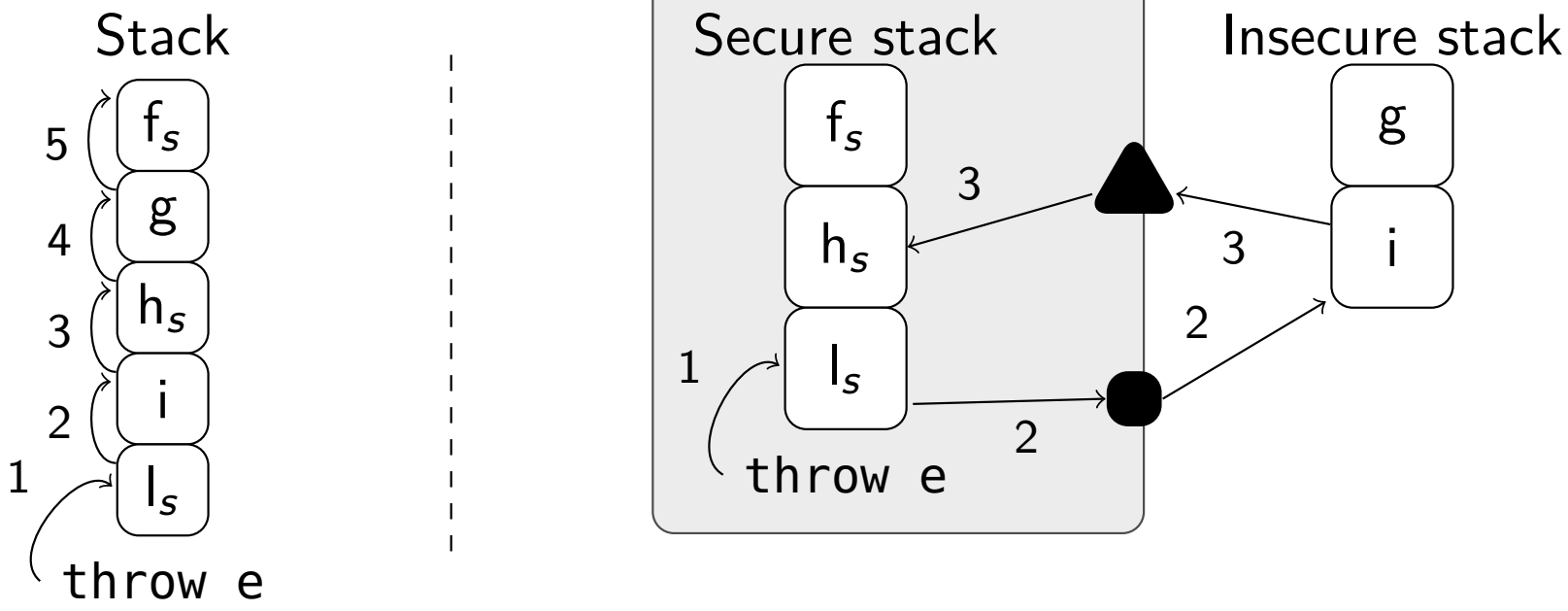

Stack
5
4
3
2
1
f_s
g
h_s
i
l_s
throw e
Secure stack
f_s
h_s
l_s
1
throw e
Insecure stack
g
i

Exceptions



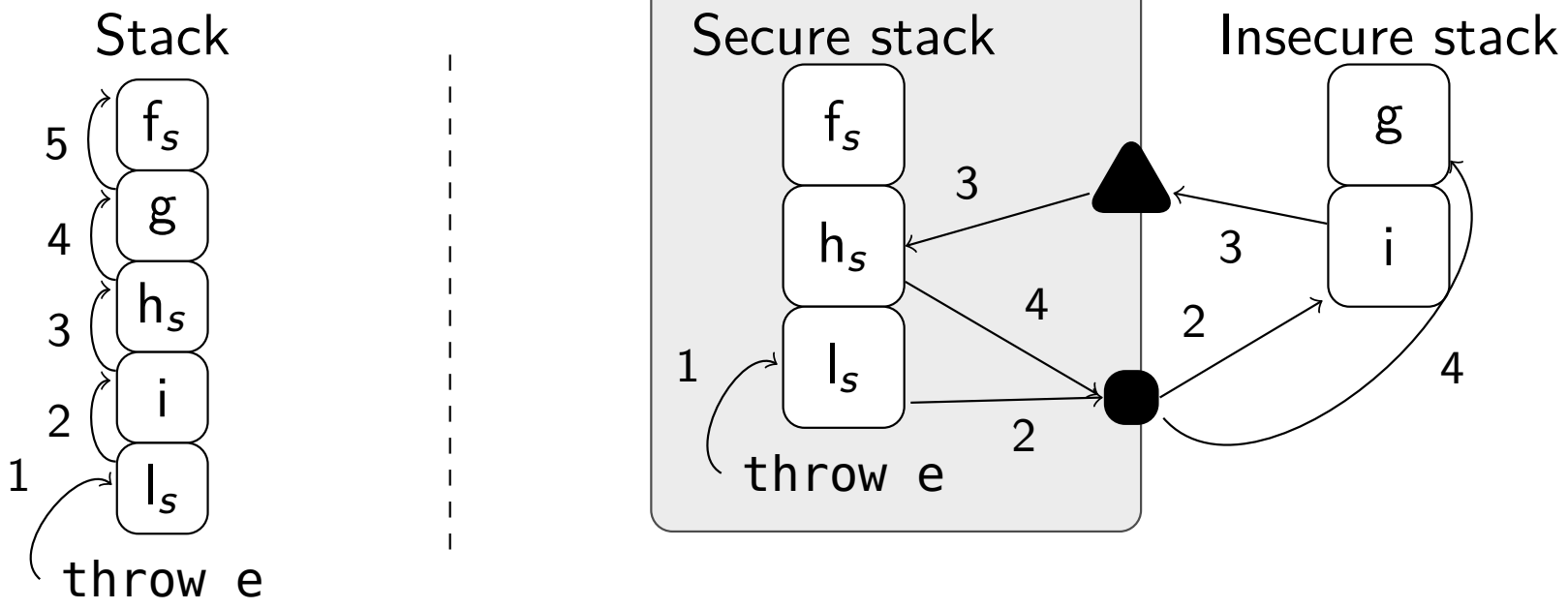
Record passed exceptions

Exceptions



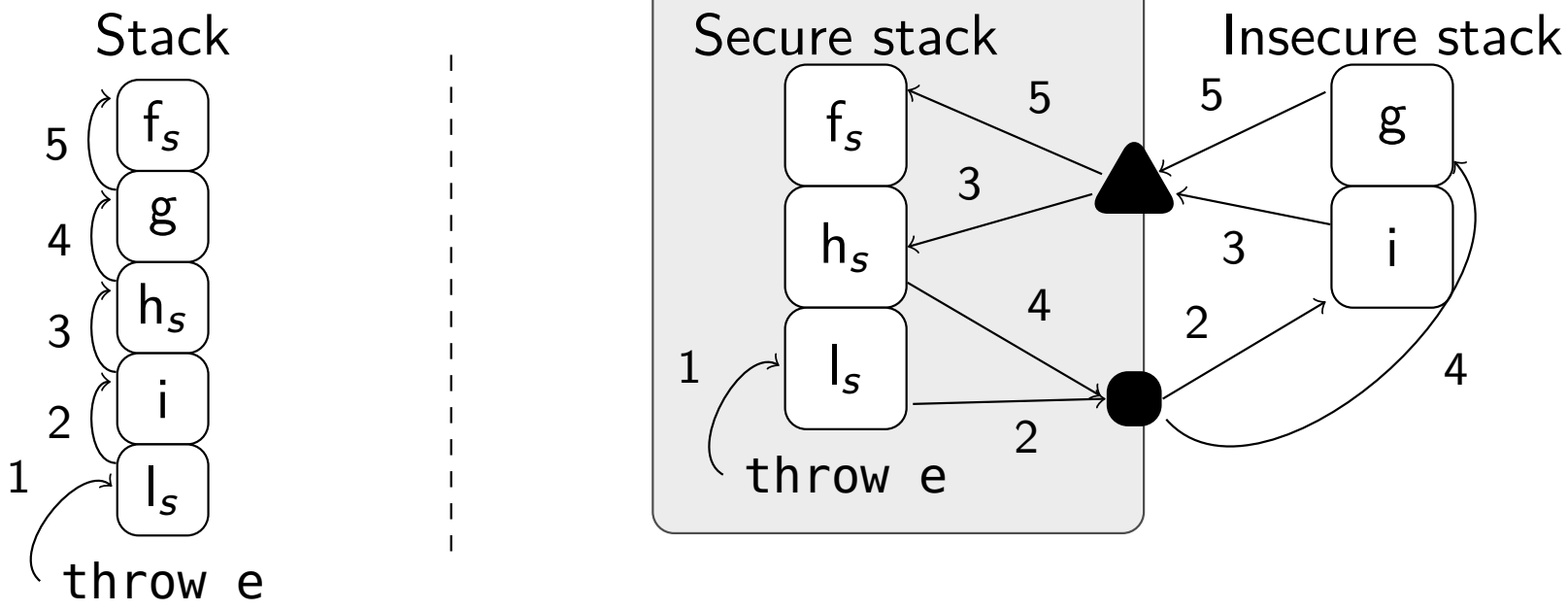
Record passed exceptions
Check that exception could be thrown

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 J+E code

So now...

- We have a strategy to securely compile J+E code
- We have the tools to implement it

So now...

- We have a strategy to securely compile J+E 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 J+E 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 J+E 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

$$C_1 \simeq^{\mathsf{J+E}} C_2 \iff C_1^{\downarrow} \simeq^{\mathsf{A+I}} C_2^{\downarrow}$$

Secure Compilation, Formally

$$C_1 \simeq^{\text{J+E}} C_2 \iff C_1^{\downarrow} \simeq^{\text{A+I}} C_2^{\downarrow}$$

Secure Compilation, Formally

$$C_1 \simeq^{J+E} C_2 \iff C_1^{\downarrow} \simeq^{A+I} C_2^{\downarrow}$$

Contextual Equivalence

Contextual Equivalence

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

Contextual Equivalence

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

Contextual Equivalence

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

All contexts

Secure Compilation

$$C_1 \simeq^{\mathsf{J+E}} C_2 \iff C_1^{\downarrow} \simeq^{\mathsf{A+I}} C_2^{\downarrow}$$

Secure Compilation

$$C_1 \simeq^{\mathsf{J+E}} C_2 \iff C_1^{\downarrow} \simeq^{\mathsf{A+I}} C_2^{\downarrow}$$

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

Secure Compilation

$$C_1 \simeq^{\mathsf{J+E}} C_2 \iff C_1^{\downarrow} \simeq^{\mathsf{A+I}} C_2^{\downarrow}$$

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

VERY COMPLEX!

Secure Compilation

$$C_1 \simeq^{\mathsf{J+E}} C_2 \iff C_1^{\downarrow} \simeq^{\mathsf{A+I}} C_2^{\downarrow}$$

Secure Compilation

$$C_1 \simeq^{\mathsf{J+E}} C_2 \quad \Leftarrow \quad C_1^{\downarrow} \simeq^{\mathsf{A+I}} C_2^{\downarrow}$$

Secure Compilation

$$C_1 \simeq^{\mathsf{J+E}} C_2 \Leftarrow C_1^{\downarrow} \simeq^{\mathsf{A+I}} C_2^{\downarrow}$$

Secure Compilation

$$C_1 \simeq^{\mathsf{J+E}} C_2 \quad \Rightarrow \quad C_1^{\downarrow} \simeq^{\mathsf{A+I}} C_2^{\downarrow}$$

Secure Compilation

$$C_1 \simeq^{\mathsf{J+E}} C_2 \quad \Rightarrow \quad \boxed{\begin{array}{c} C_1^{\downarrow} \simeq^{\mathsf{A+I}} C_2^{\downarrow} \\ \Updownarrow \\ \mathsf{Traces}(C_1^{\downarrow}) = \mathsf{Traces}(C_2^{\downarrow}) \end{array}}$$

Fully Abstract Trace Semantics

Secure Compilation

$$C_1 \not\simeq^{\text{J+E}} C_2 \quad \Leftarrow \quad \text{Traces}(C_1^{\downarrow}) \neq \text{Traces}(C_2^{\downarrow})$$

Secure Compilation

$C_1 \not\simeq^{\text{J+E}} C_2$ ✔ $\Leftarrow$ $\quad \text{Traces}(C_1^{\downarrow}) \neq \text{Traces}(C_2^{\downarrow})$

Secure Compilation

$$C_1 \not\simeq^{\text{J+E}} C_2 \Leftarrow$$

$$C_1^{\downarrow} \simeq^{\text{A+I}} C_2^{\downarrow}$$

$$\Updownarrow$$

$$\text{Traces}(C_1^{\downarrow}) = \text{Traces}(C_2^{\downarrow})$$

Fully Abstract Trace Semantics

Outline

Modular, Secure Compilation

- current model has a single secure module

Modular, Secure Compilation

- current model has a single secure module
- assembly-level linking of securely-compiled modules is not investigated

Modular, Secure Compilation

- current model has a single secure module
- assembly-level linking of securely-compiled modules is not investigated
- can we link securely-compiled modules securely?

Modular, Secure Compilation

- current model has a single secure module
- assembly-level linking of securely-compiled modules is not investigated
- can we link securely-compiled modules securely?
- what attacks arise in the presence of linking?

Logical-relations Based Proof Technique

- Devirese *et al.*@ POPL'16

Logical-relations Based Proof Technique

- Devirese *et al.*@ POPL'16
- proof technique for fully-abstract compilation based on logical relations

Logical-relations Based Proof Technique

- Devirese *et al.*@ POPL'16
- proof technique for fully-abstract compilation based on logical relations
- check out the video on the popl website!

Secure Compilation Statement

- does full abtraction mean security?

Secure Compilation Statement

- does full abtraction mean security?
- can it express all security properties?

Secure Compilation Statement

- does full abtraction mean security?
- can it express all security properties?
- is there a better/more precise notion of secure compilation?

Secure Compilation Statement

- does full abtraction mean security?
- can it express all security properties?
- is there a better/more precise notion of secure compilation?
- can we relate it to hyperproperties (i.e., properties over sets of traces)?

Questions

Thank you!

Qs ?