Secure Compilation of Object-Oriented Components to Protected Module Architectures

Marco Patrignani [1] Dave Clarke [1] Frank Piessens [1]

[1]iMinds-DistriNet, Dept. Computer Science, KU Leuven

10 December 2013

Outline

1. Protected Modules Architectures
2. Secure (Fully Abstract) Compilation
3. What Can Go Wrong With...
   - Dynamic Memory Allocation
   - Exceptions
   - Cross-package Inheritance

Protected Modules Architecture (PMA)

- assembly-level isolation mechanism

Protected Modules Architecture (PMA)

- assembly-level isolation mechanism
- several research prototypes: Fides [SP12], Sancus [NAD+13], Flicker [MPP+08], TrustVisor [MLQ+10], Smart [EFPT12]

Protected Modules Architecture (PMA)

- assembly-level isolation mechanism
- several research prototypes: Fides [SP12], Sancus [NAD+13], Flicker [MPP+08], TrustVisor [MLQ+10], Smart [EFPT12]
- industrial prototype too: Intel SGX [MAB+13]

Protected Modules Architecture (PMA)

- assembly-level isolation mechanism
- several research prototypes: Fides [SP12], Sancus [NAD+13], Flicker [MPP+08], TrustVisor [MLQ+10], Smart [EFPT12]
- industrial prototype too: Intel SGX [MAB+13]
- implemented via Hypervisor, Hardware, Software

Protected Modules Architecture (PMA)

- assembly-level isolation mechanism
- several research prototypes: Fides [SP12], Sancus [NAD+13], Flicker [MPP+08], TrustVisor [MLQ+10], Smart [EFPT12]
- industrial prototype too: Intel SGX [MAB+13]
- implemented via Hypervisor, Hardware, Software

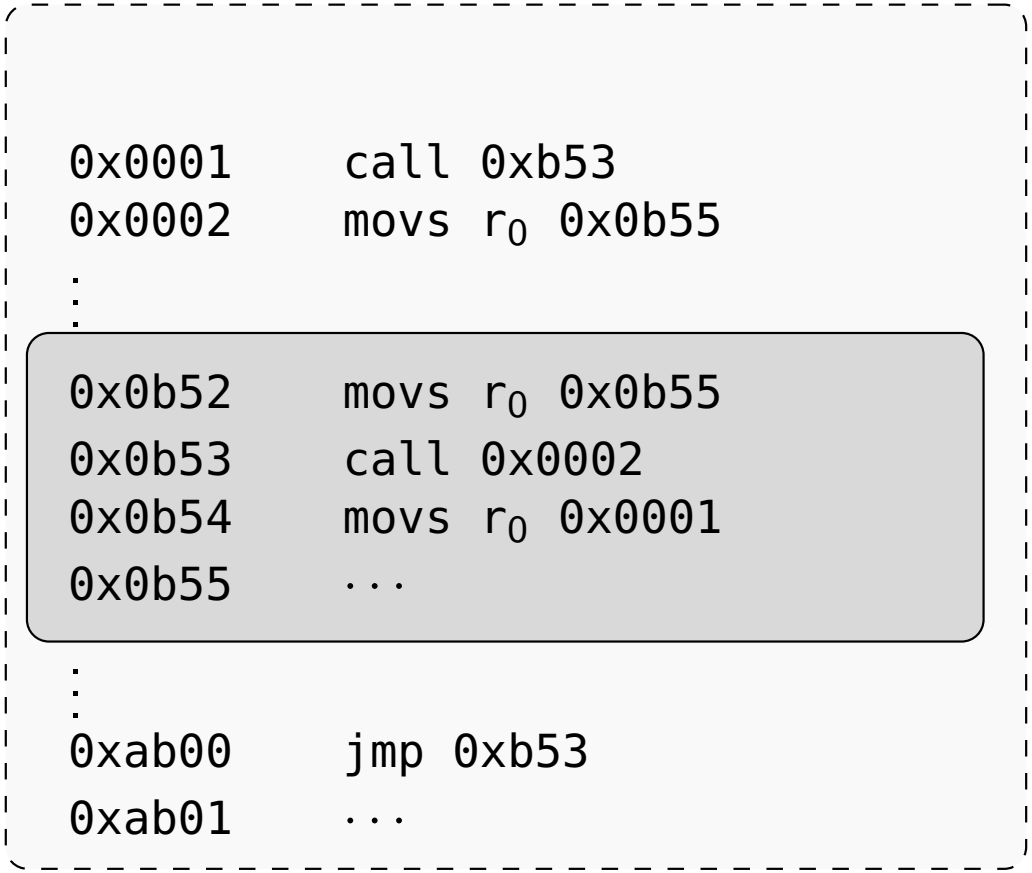
Let's see an example of PMA in action

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    ...
```

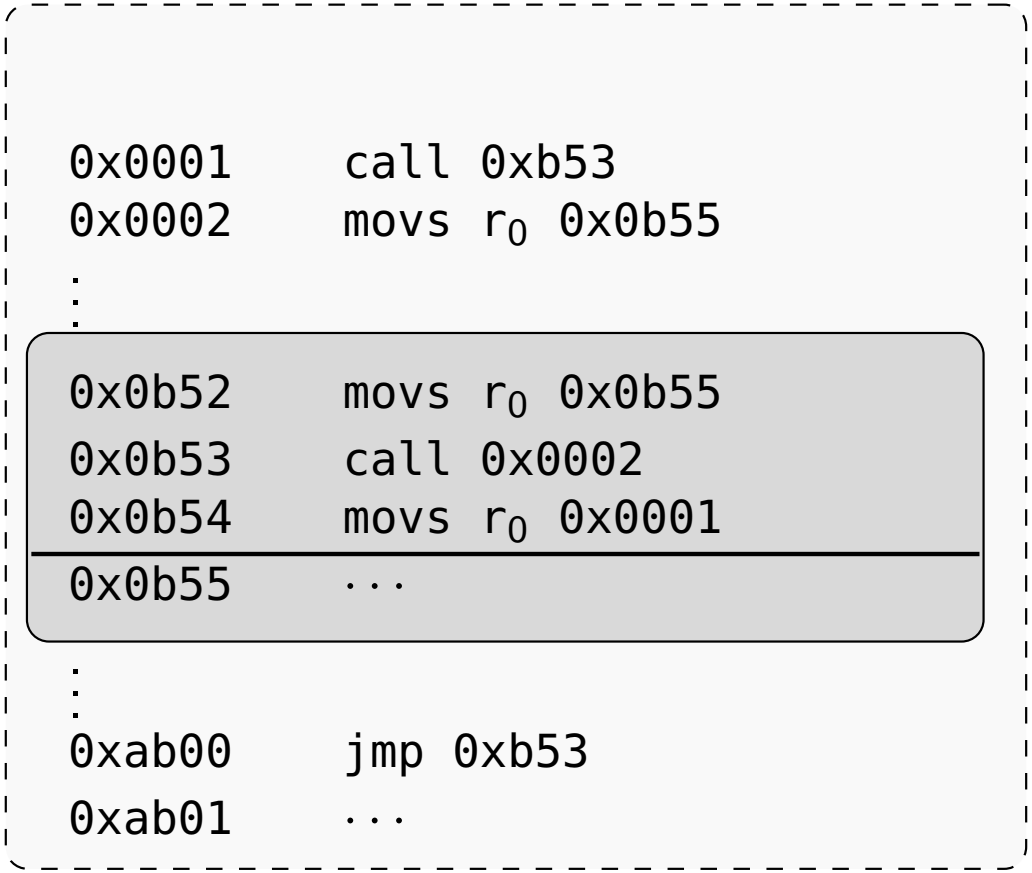
- memory space

PMA in Action



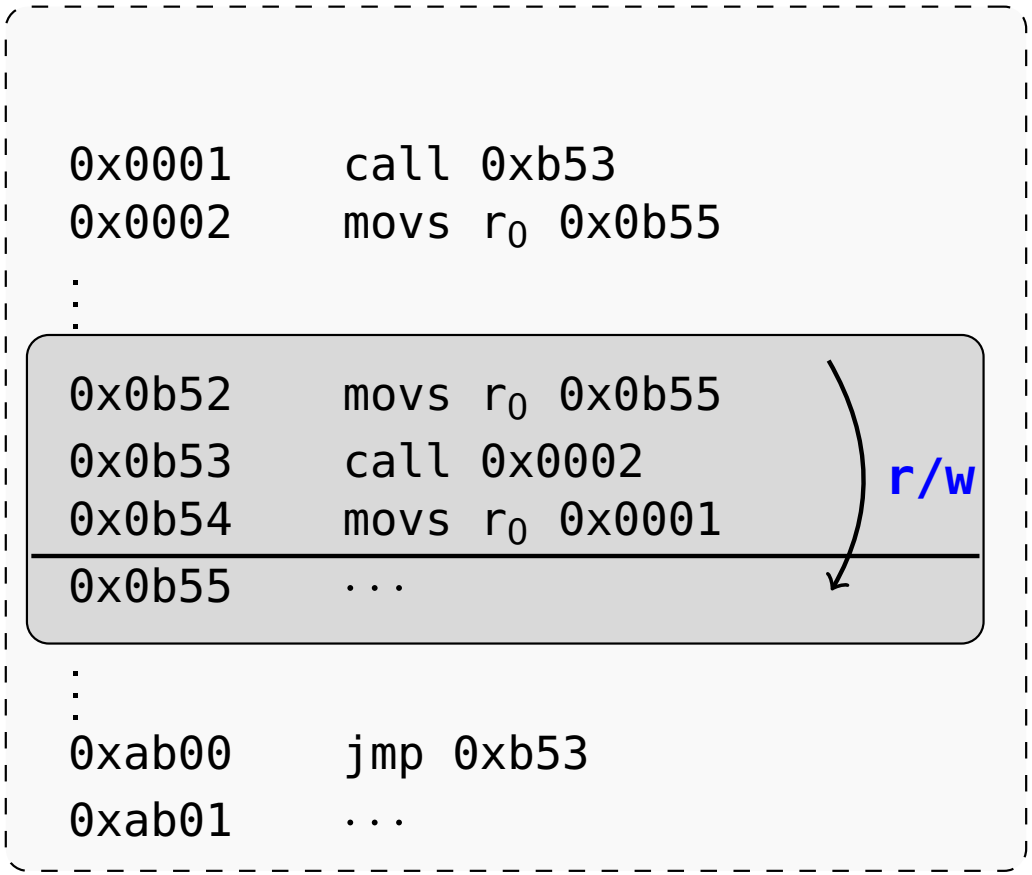
- memory space
- protected module = protected memory

PMA in Action



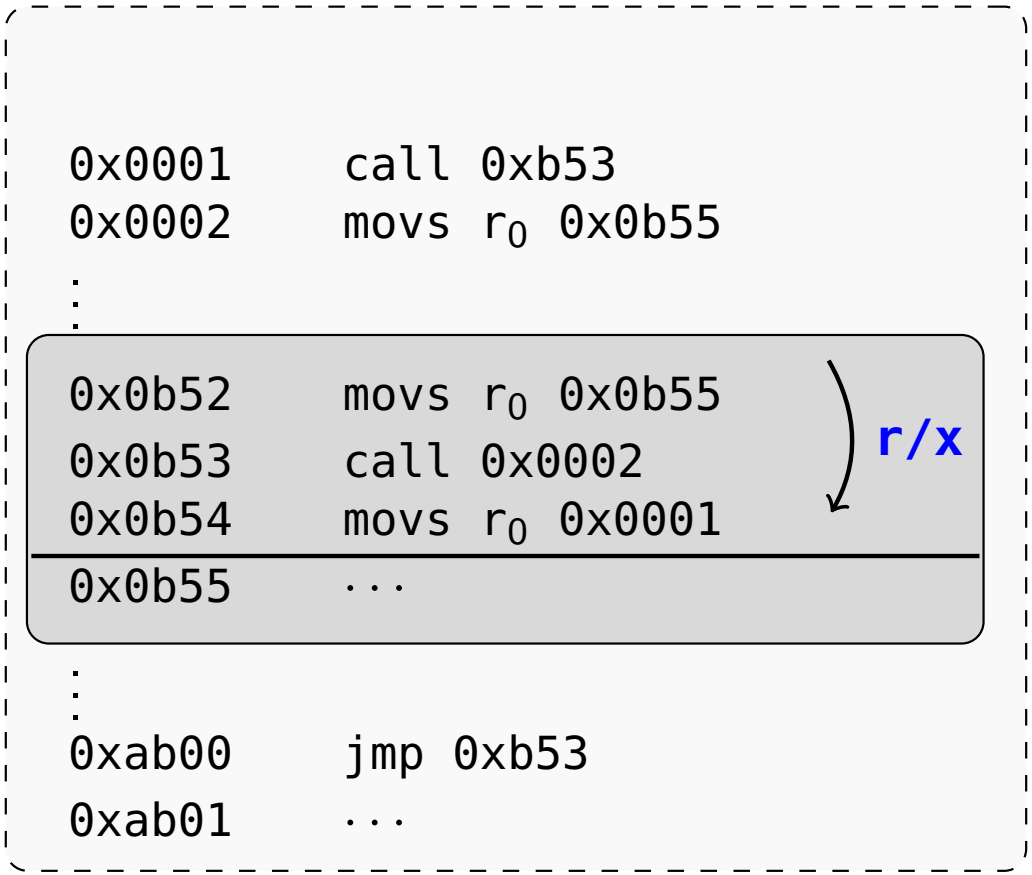
- memory space
- protected module = protected memory
- split in code and data

PMA in Action



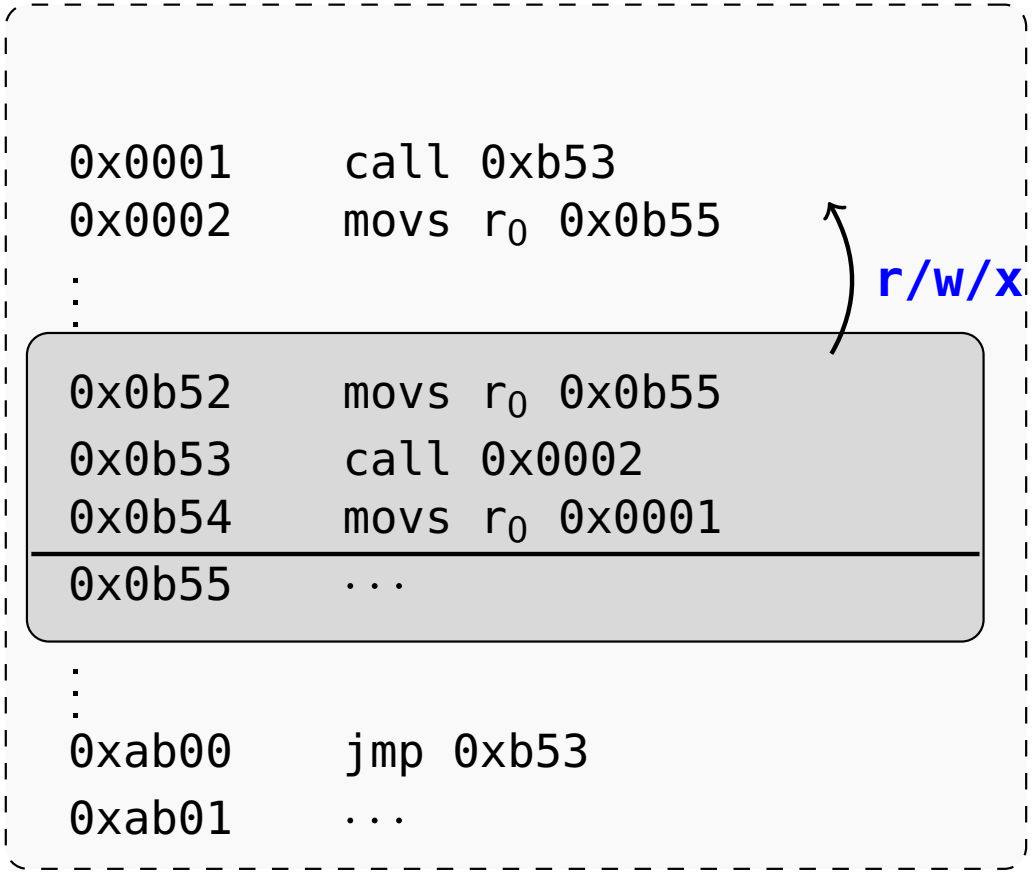
- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted

PMA in Action



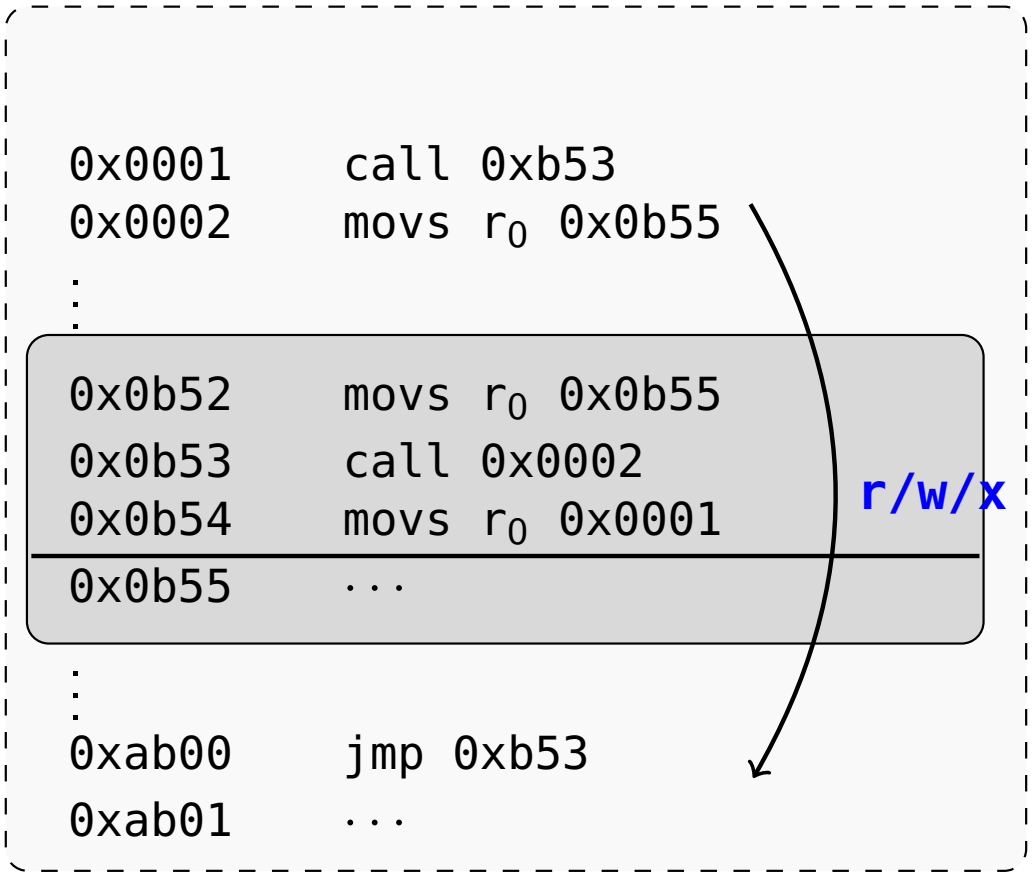
- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted

PMA in Action



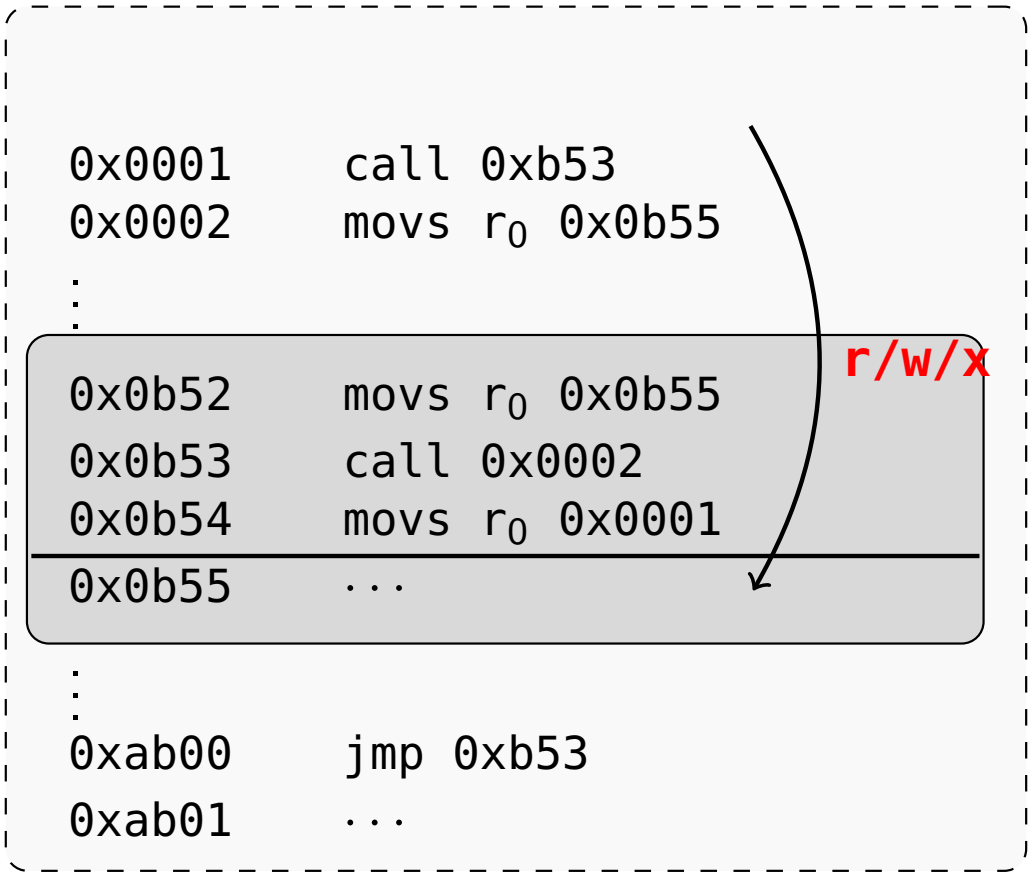
- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted

PMA in Action



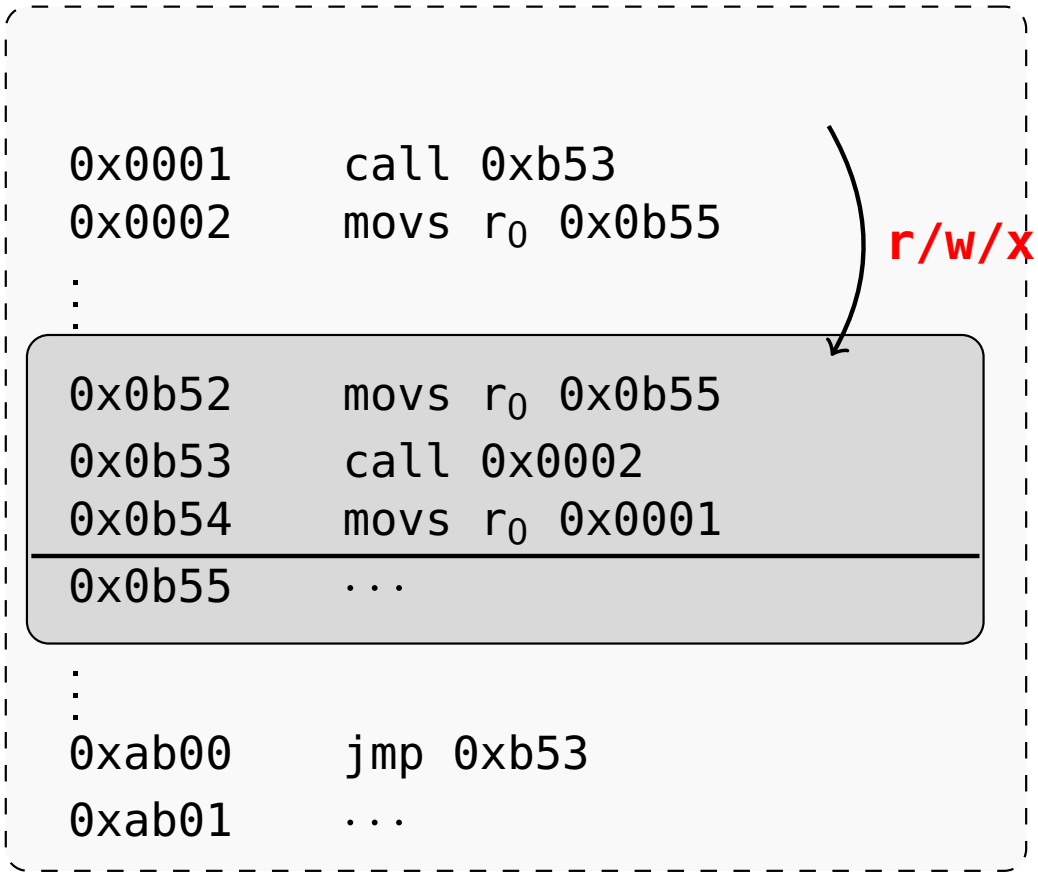
- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted

PMA in Action



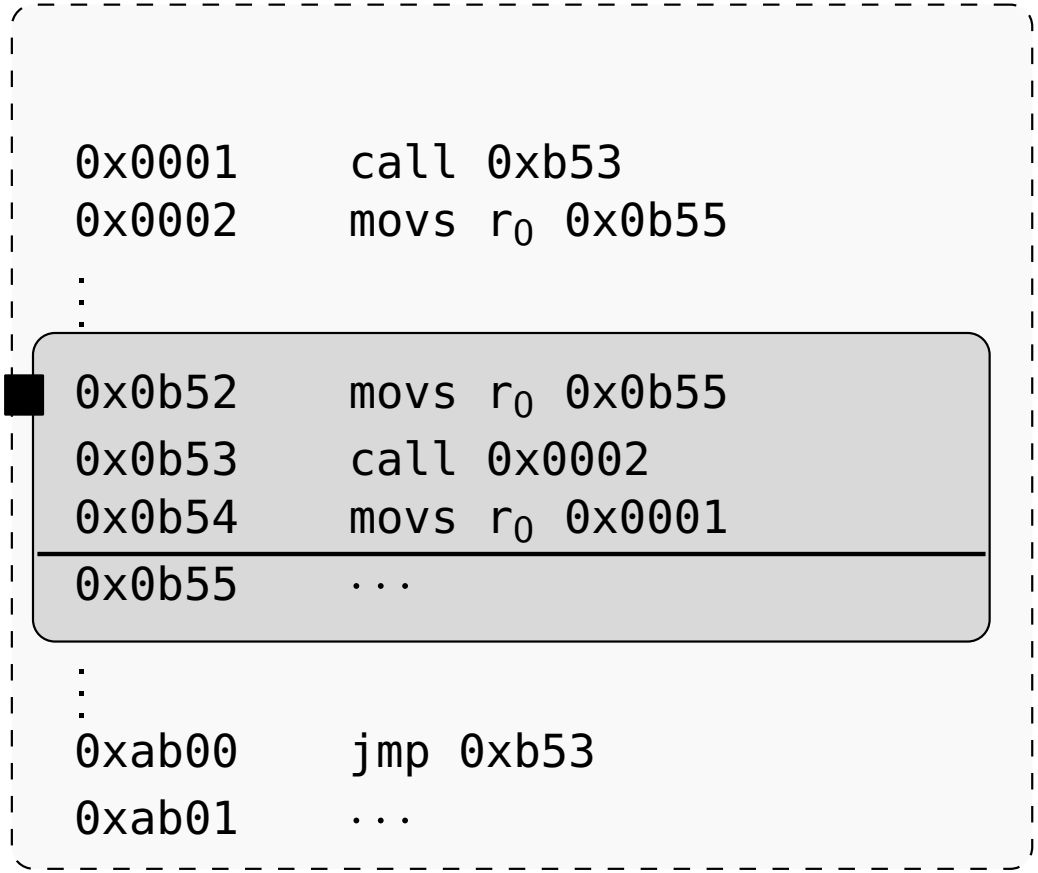
- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted

PMA in Action



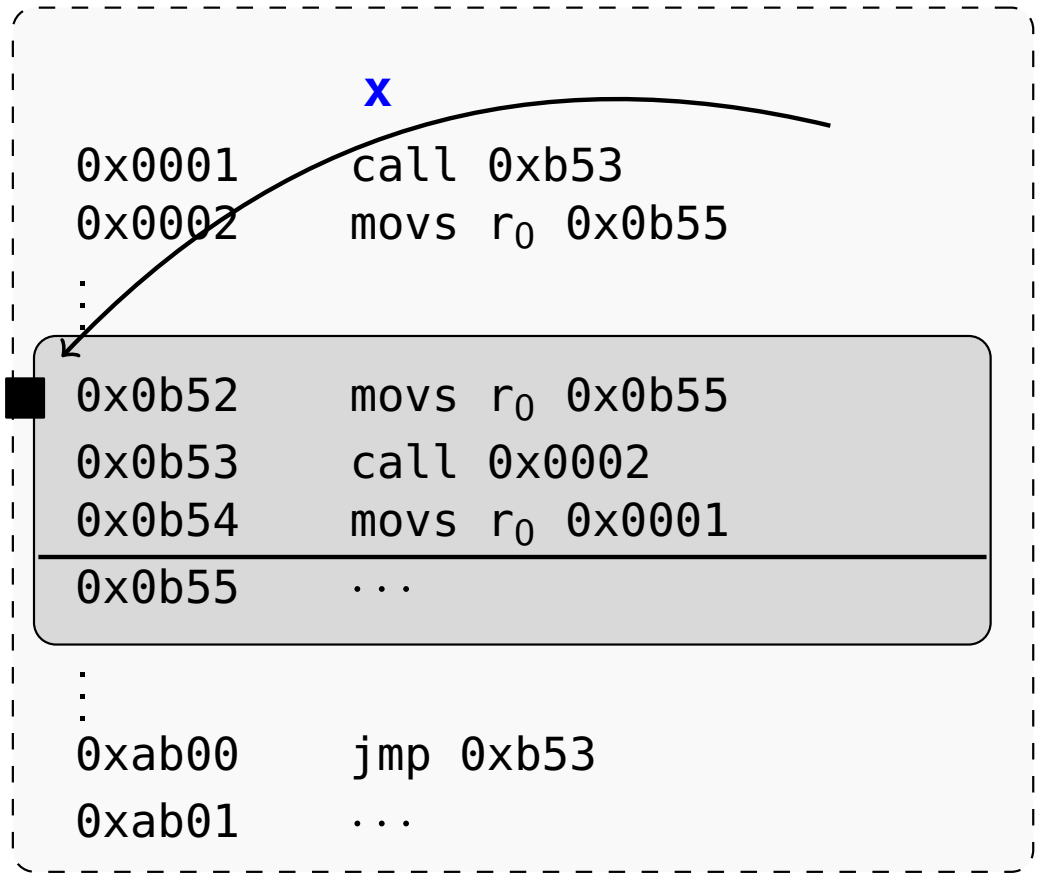
- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted

PMA in Action



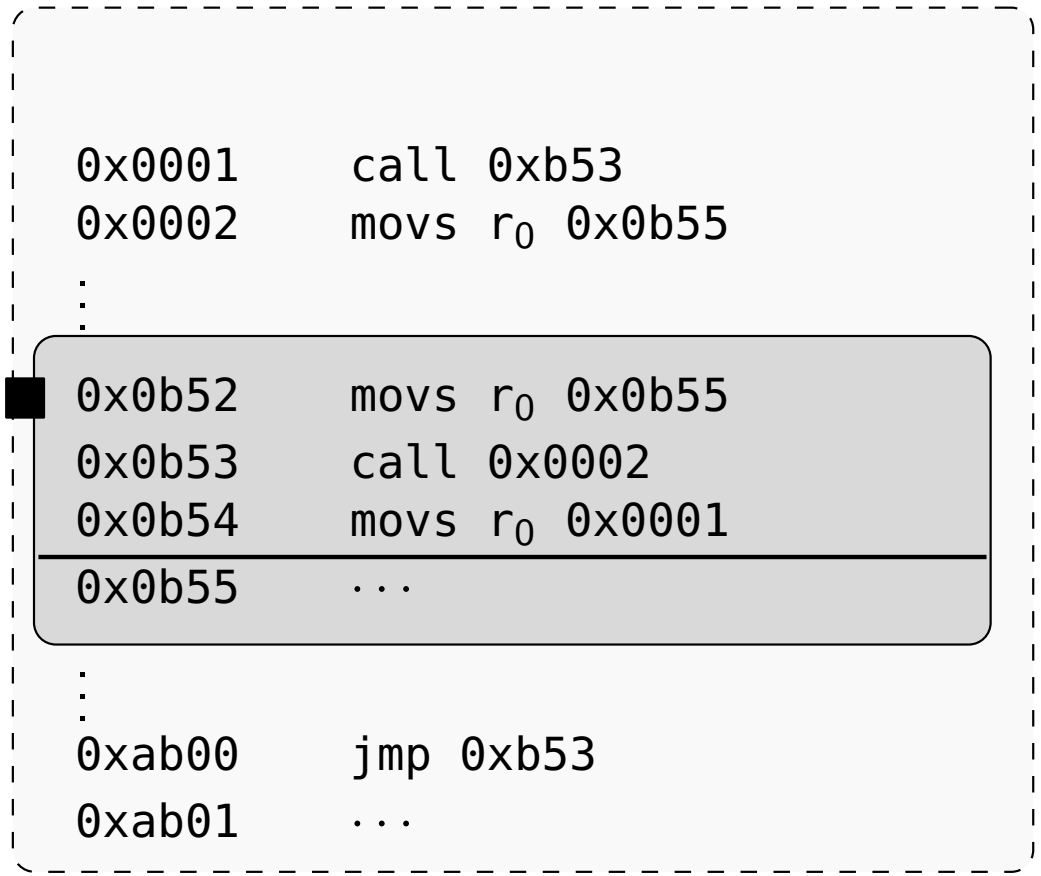
- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted
- entry points for communication (■)

PMA in Action



- 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



- memory space
- protected module = protected memory
- split in code and data
- protected code is unrestricted
- unprotected code is restricted
- entry points for communication (■)
- we need only 1 module

Languages of the Compiler

- target language: assembly

Languages of the Compiler

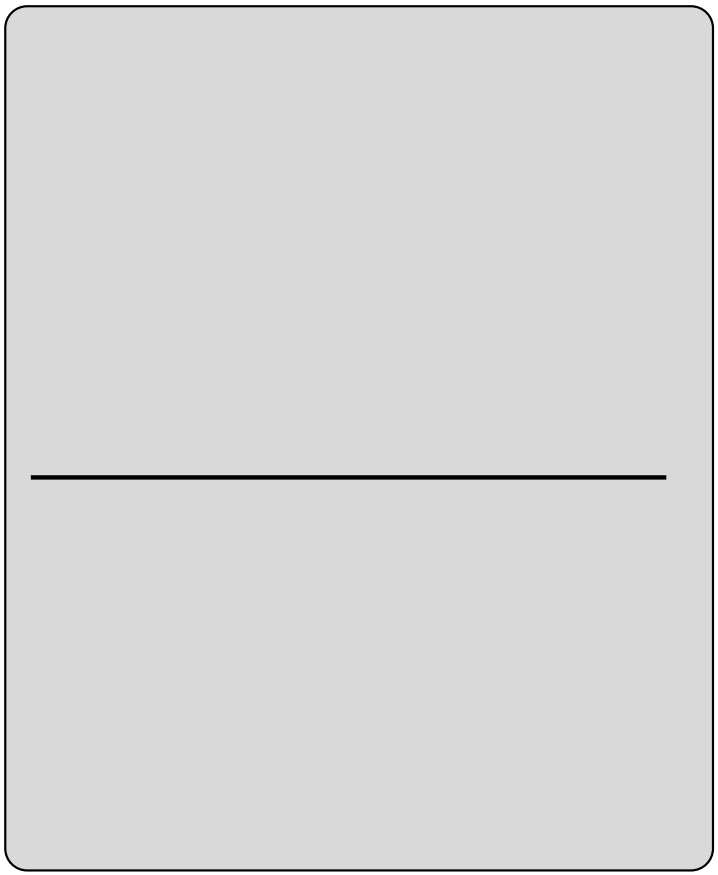
- target language: assembly
- source language: +/- Java jr [JR05]

Languages of the Compiler

- target language: assembly
- source language: +/- Java jr [JR05]
  - 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



```
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

```
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

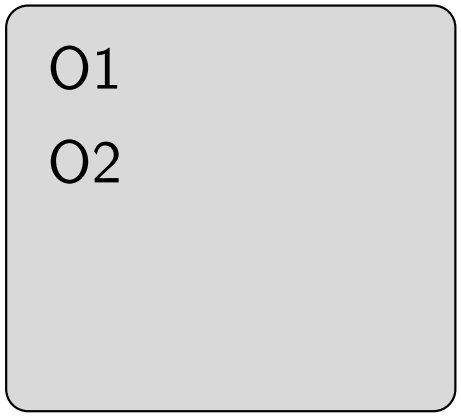
Secure stack

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

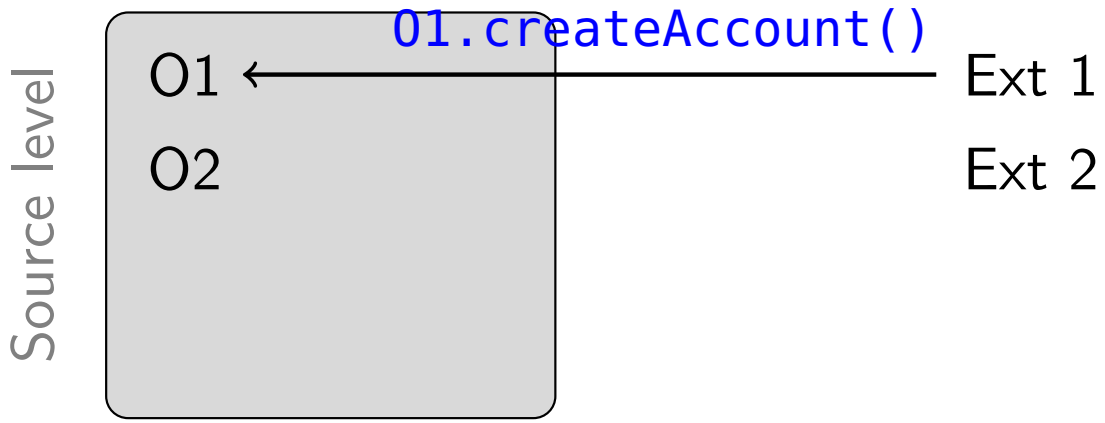


Ext 1

Ext 2

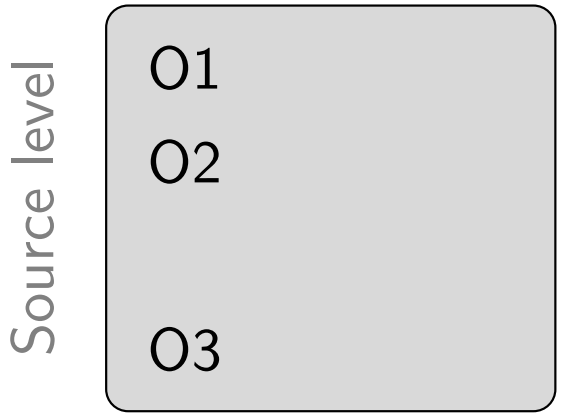
- Protect against low-level attackers

Secure Compilation, Informally



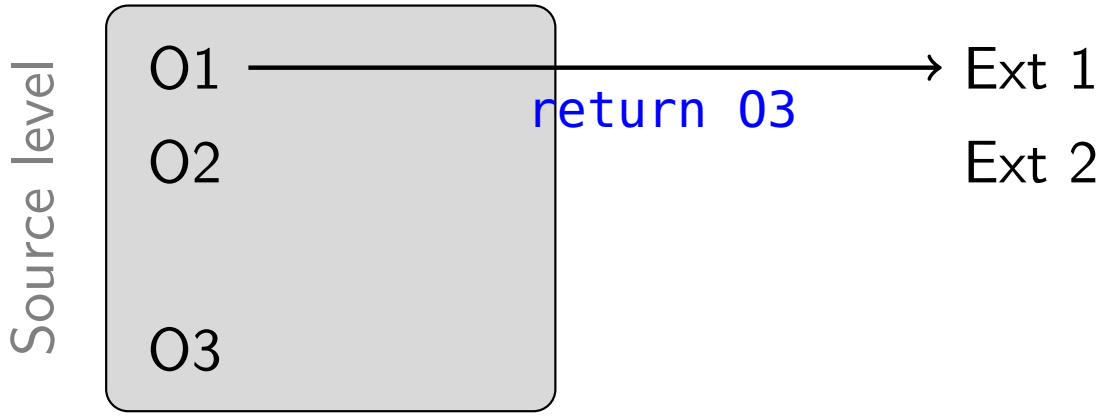
- Protect against low-level attackers

Secure Compilation, Informally



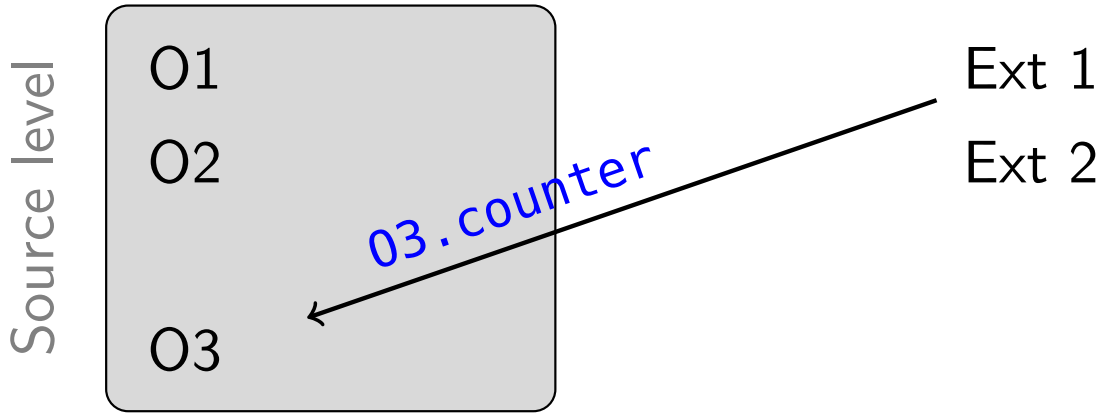
- Protect against low-level attackers

Secure Compilation, Informally



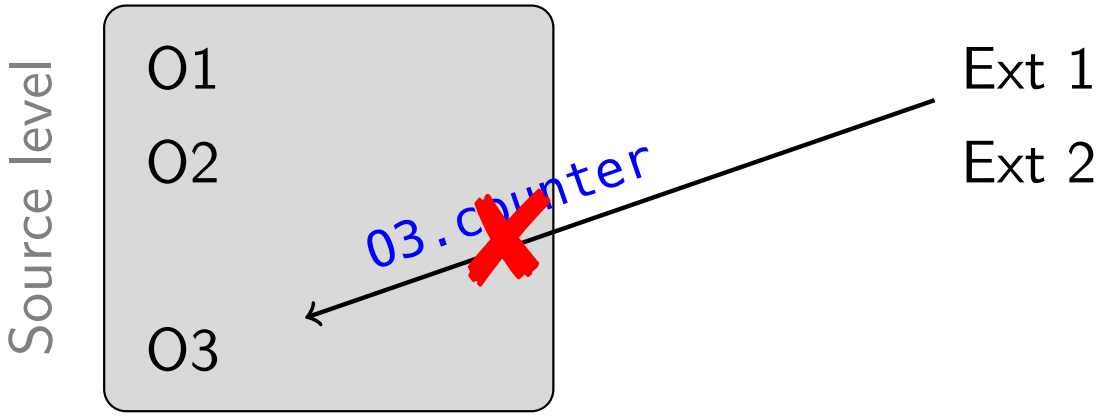
- Protect against low-level attackers

Secure Compilation, Informally



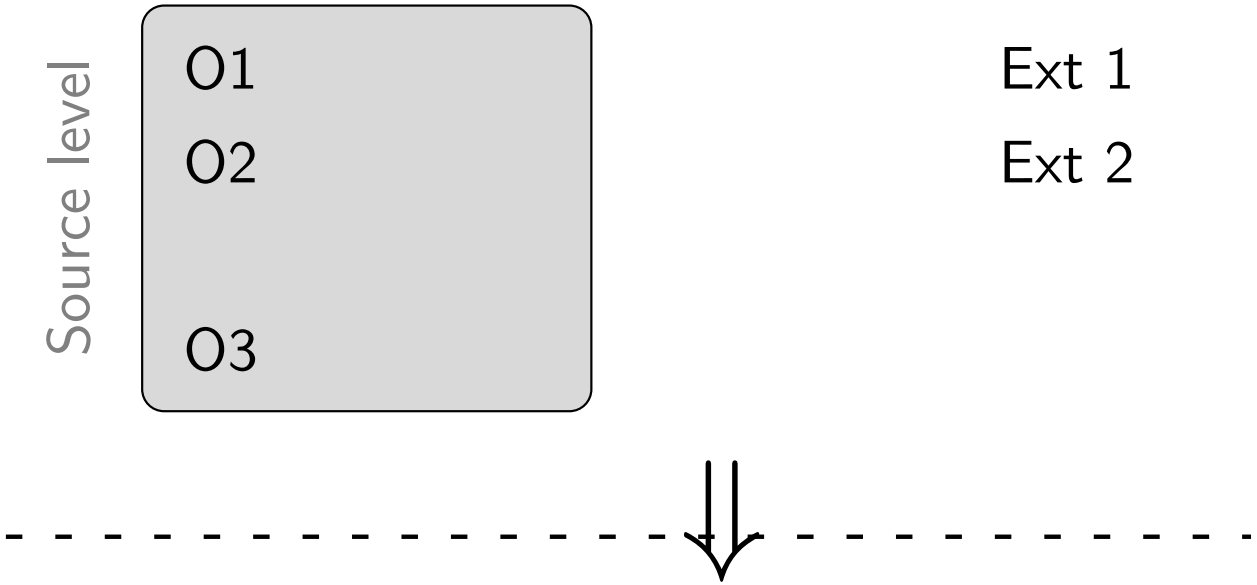
- Protect against low-level attackers

Secure Compilation, Informally



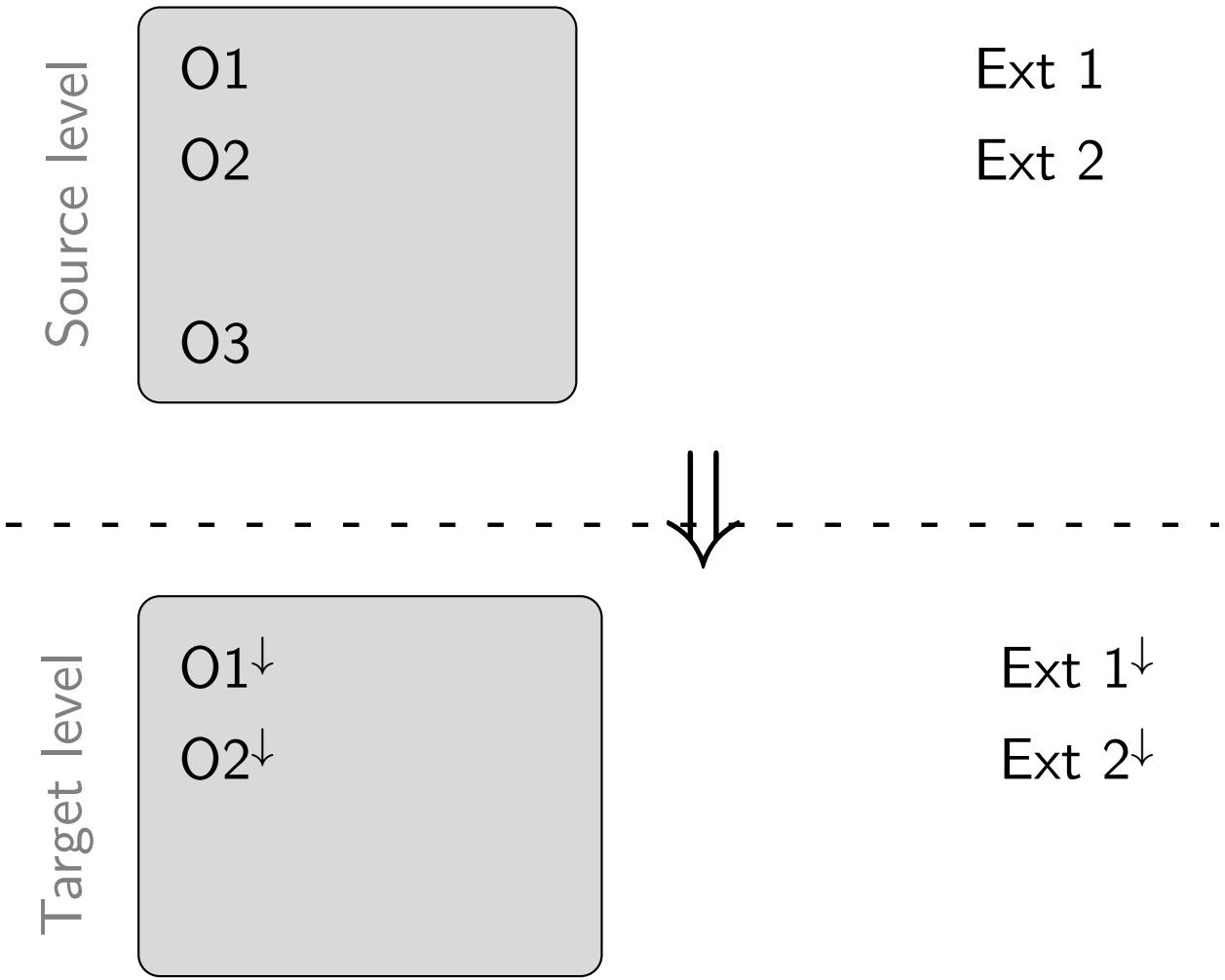
- Protect against low-level attackers

Secure Compilation, Informally



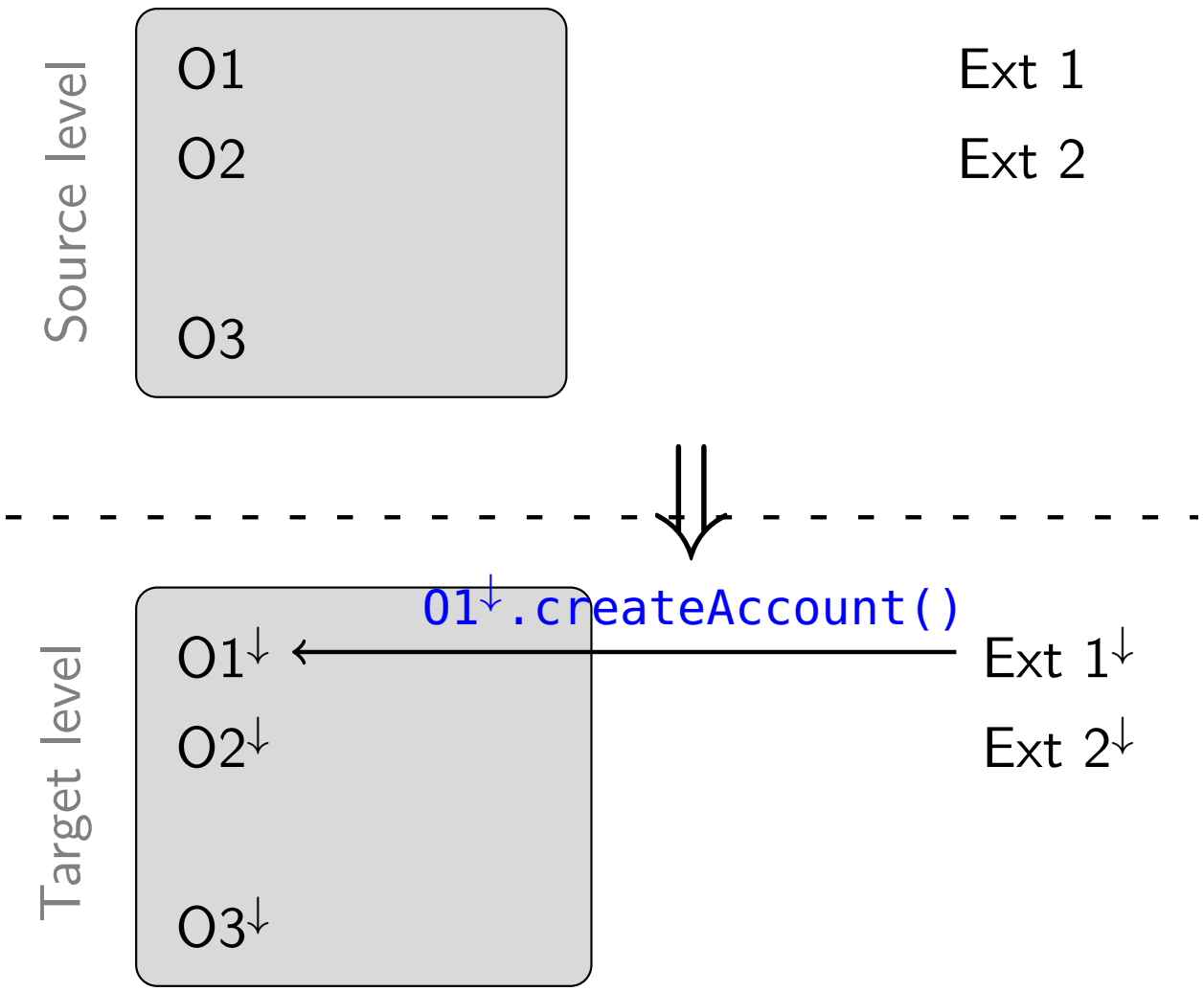
- Protect against low-level attackers

Secure Compilation, Informally



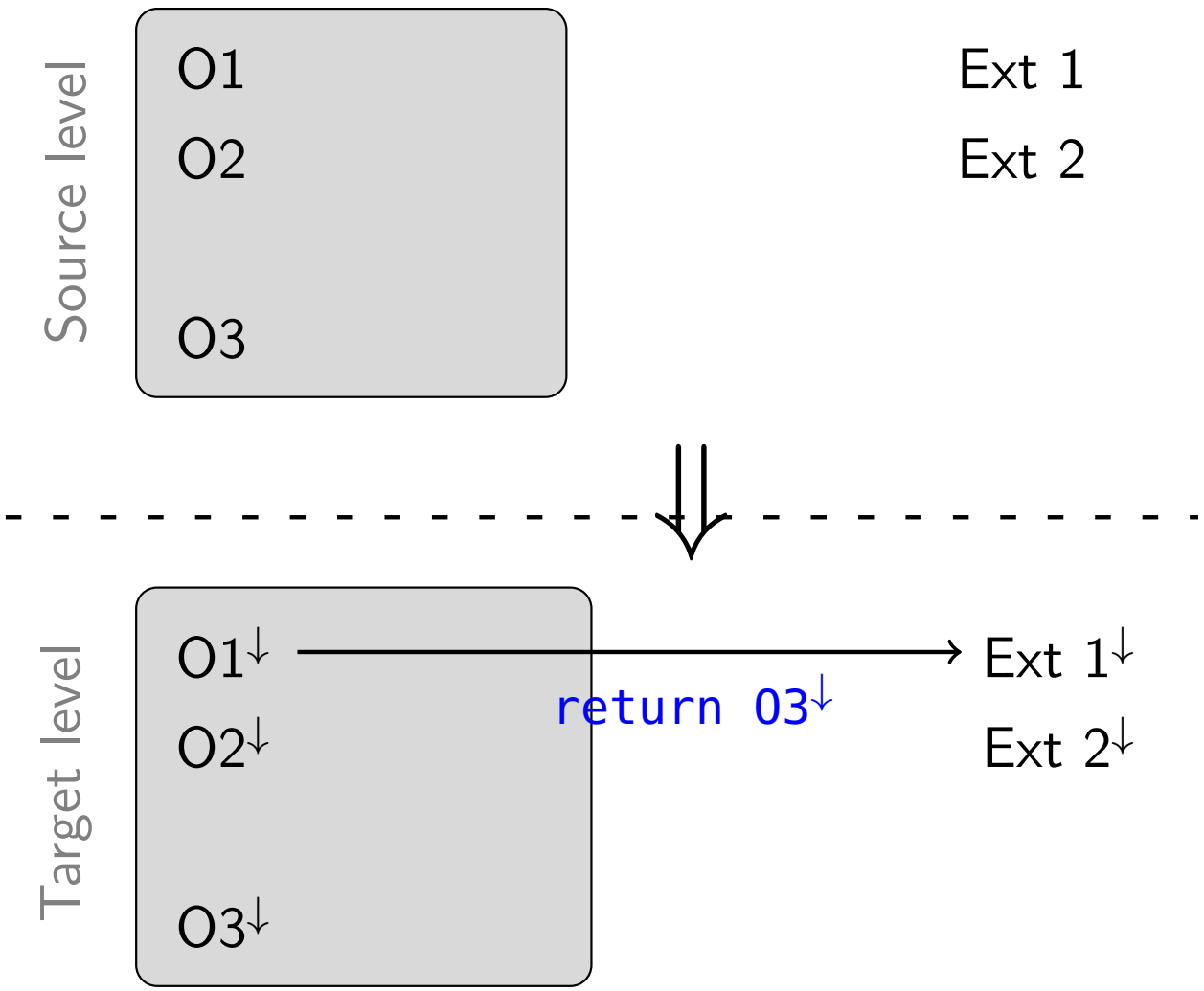
- Protect against low-level attackers

Secure Compilation, Informally



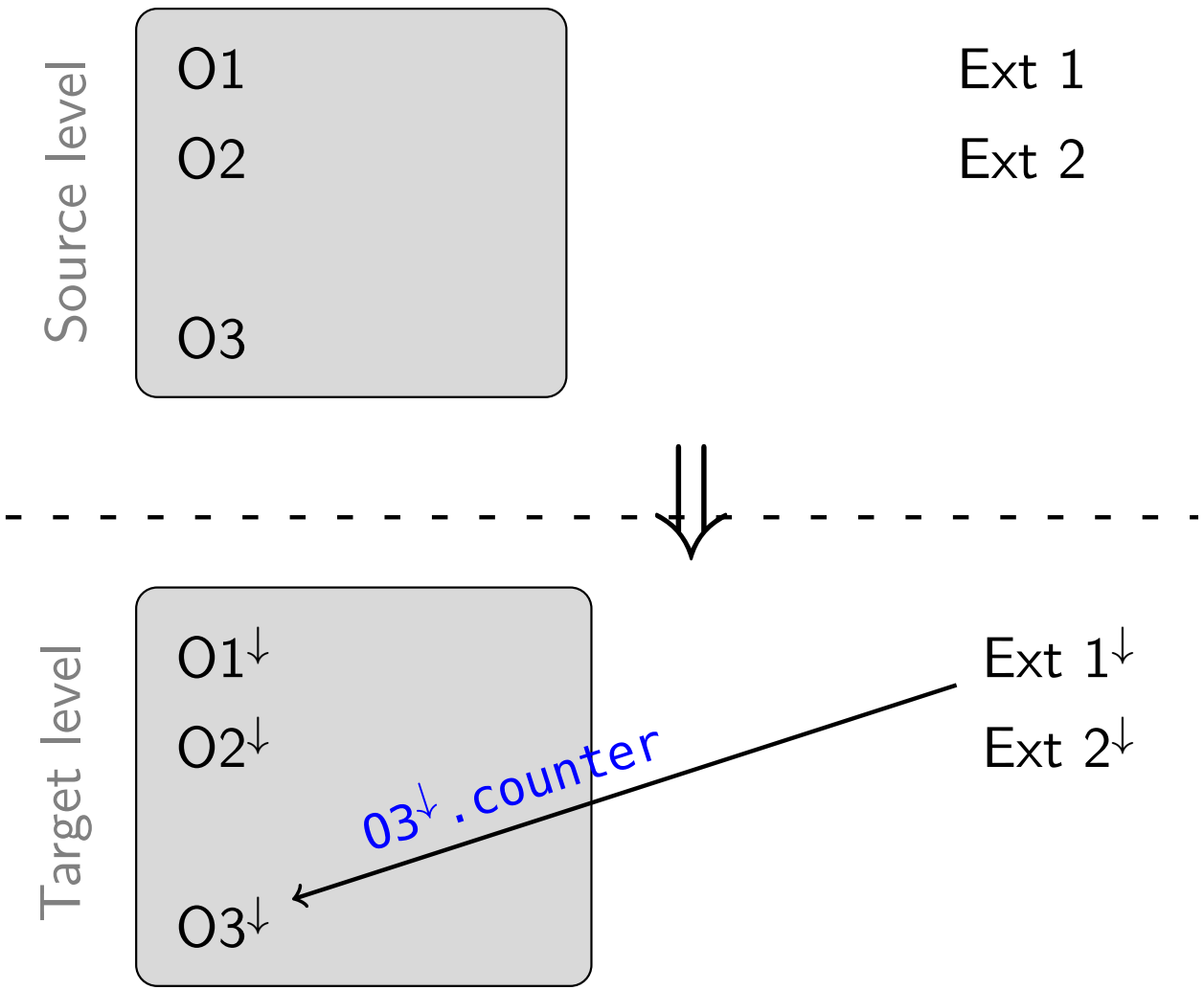
- Protect against low-level attackers

Secure Compilation, Informally



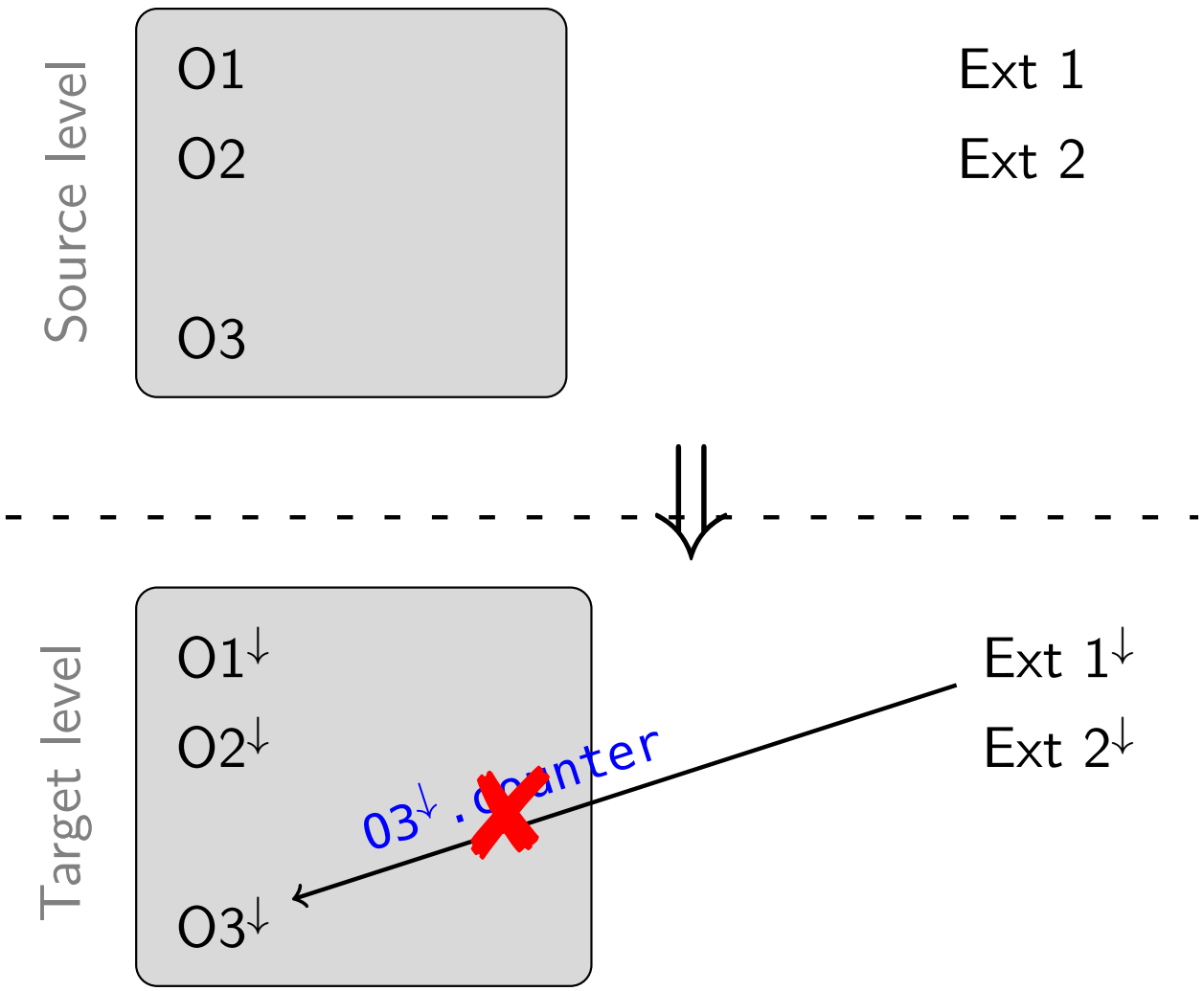
- Protect against low-level attackers

Secure Compilation, Informally



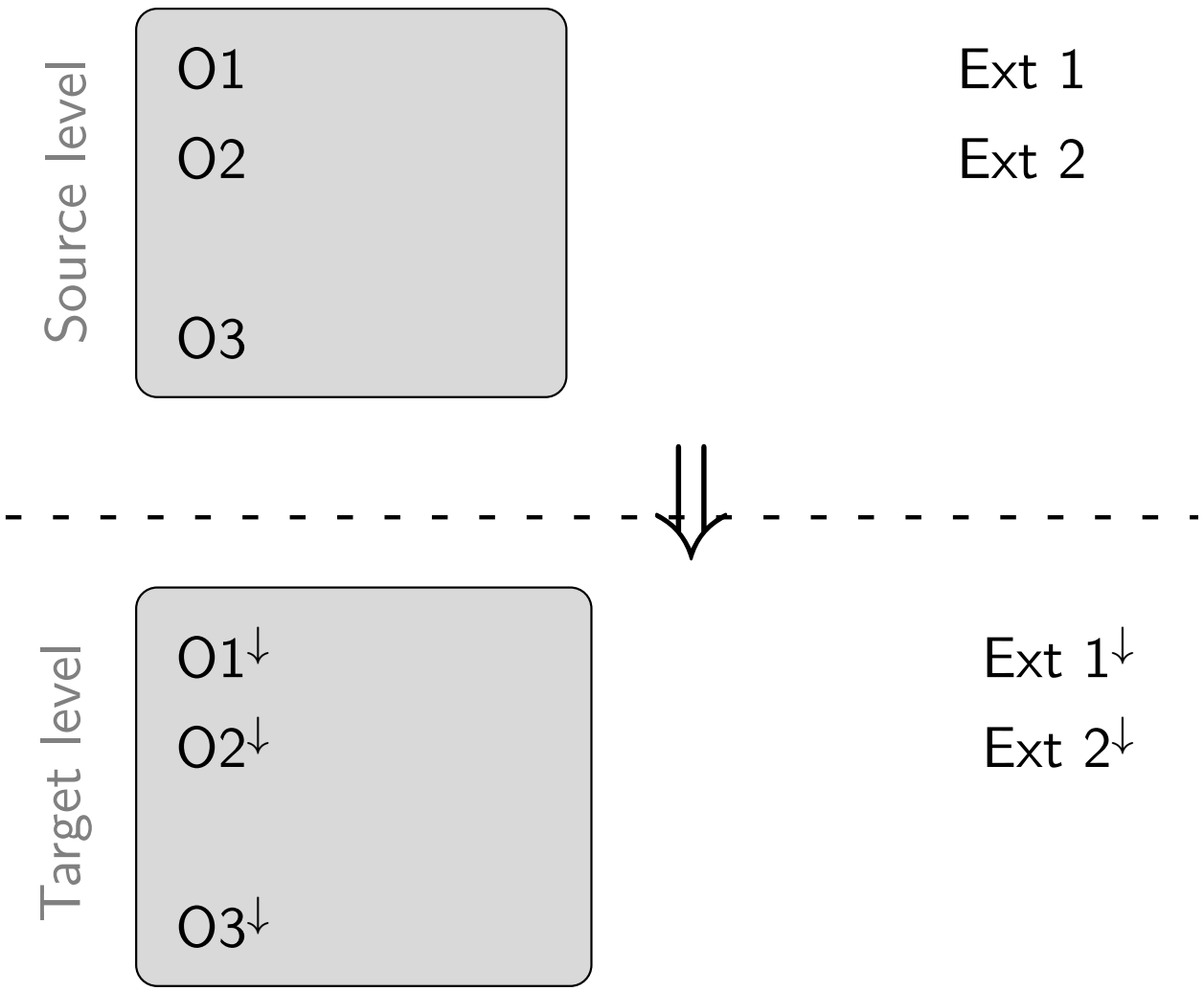
- Protect against low-level attackers

Secure Compilation, Informally



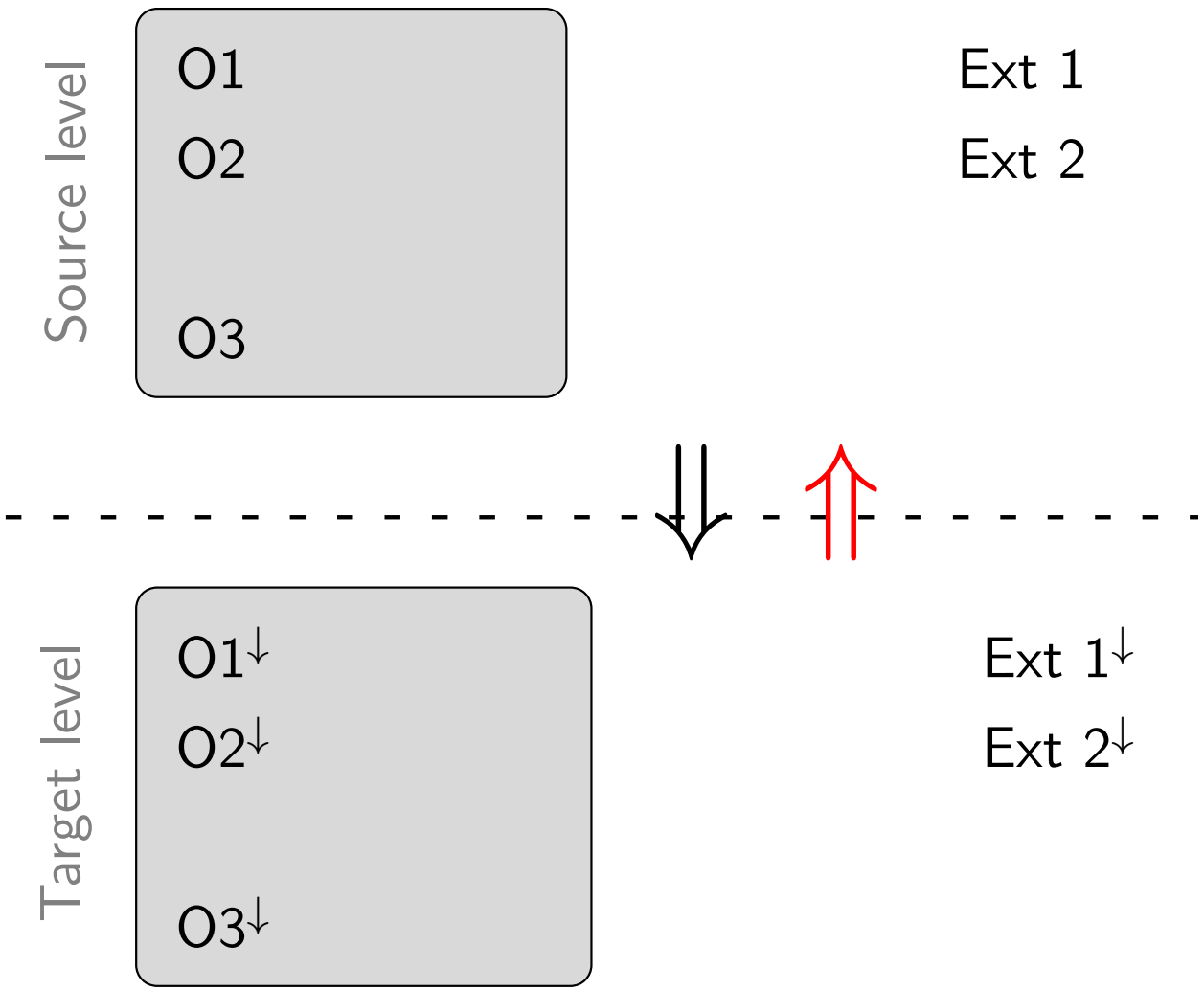
- Protect against low-level attackers

Secure Compilation, Informally



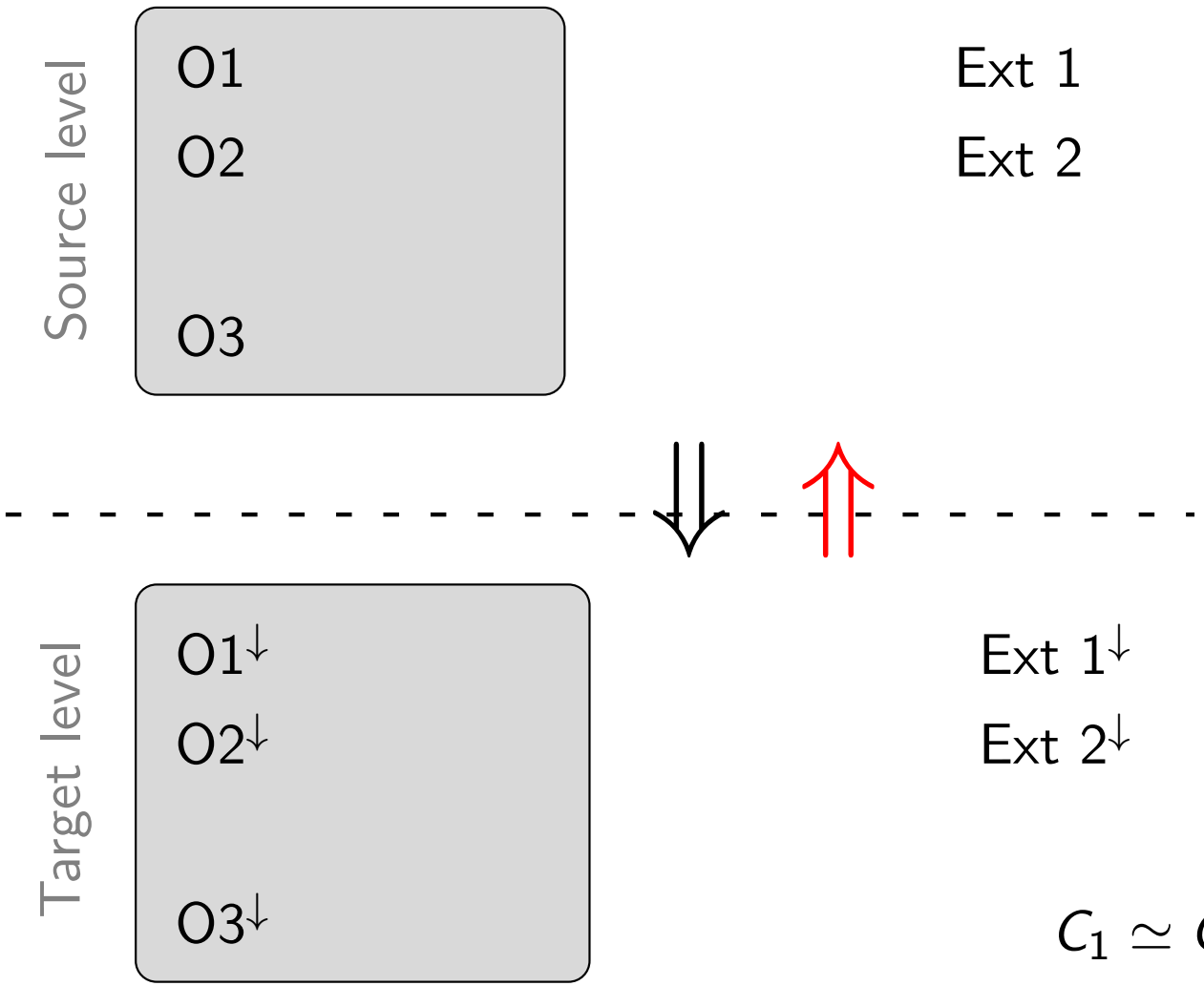
- Protect against low-level attackers
- Preservation of contextual equivalence

Secure Compilation, Informally



- Protect against low-level attackers
- Preservation of contextual equivalence
- Reflection of contextual equivalence (checks needed)

Secure Compilation, Informally



- Protect against low-level attackers
- Preservation of contextual equivalence
- Reflection of contextual equivalence (checks needed)
- Formally:

$$C_1 \simeq C_2 \iff C_1^{\downarrow} \simeq^{\downarrow} C_2^{\downarrow}$$

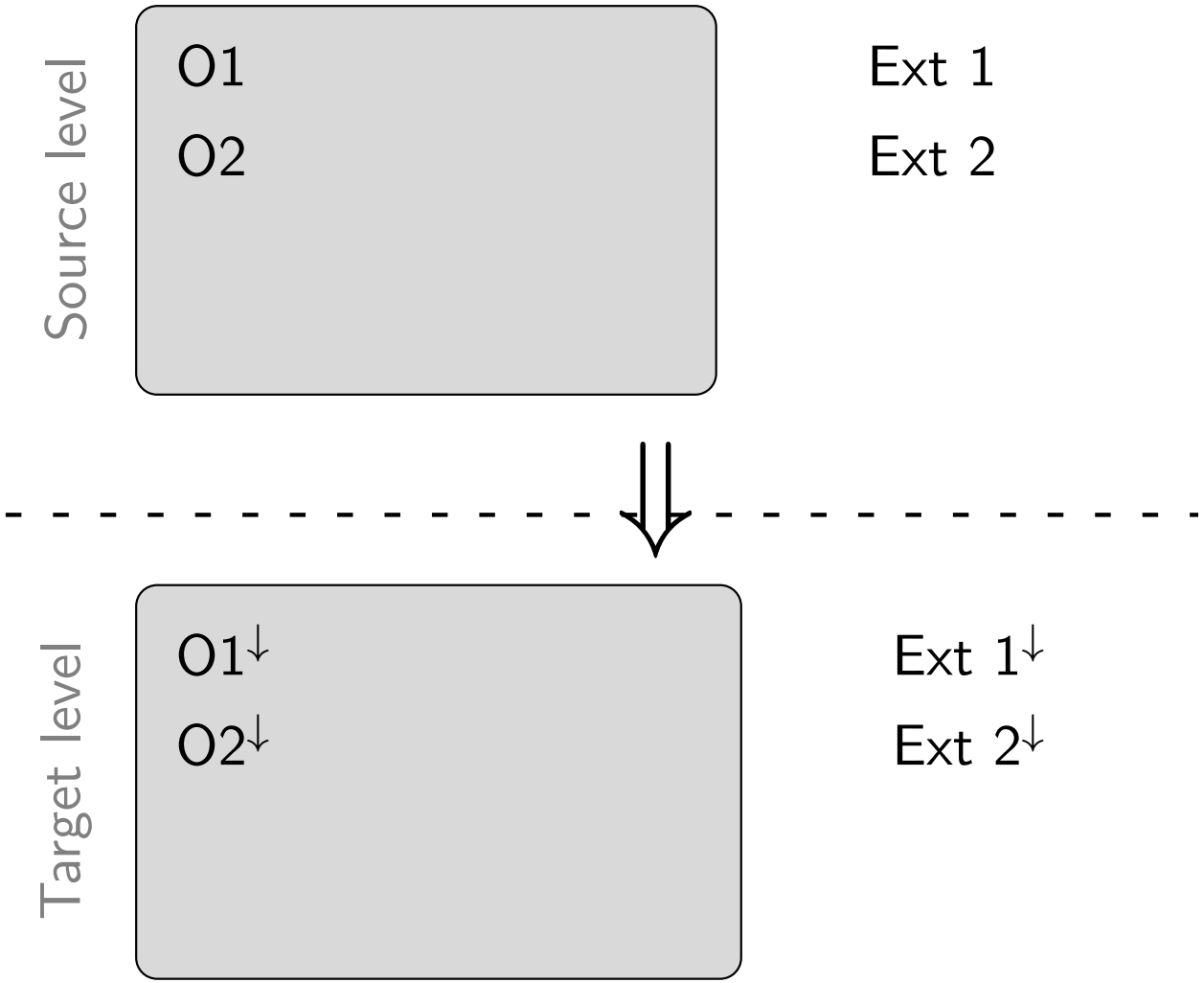
Things That Can Go Wrong

1 Protected Modules Architectures

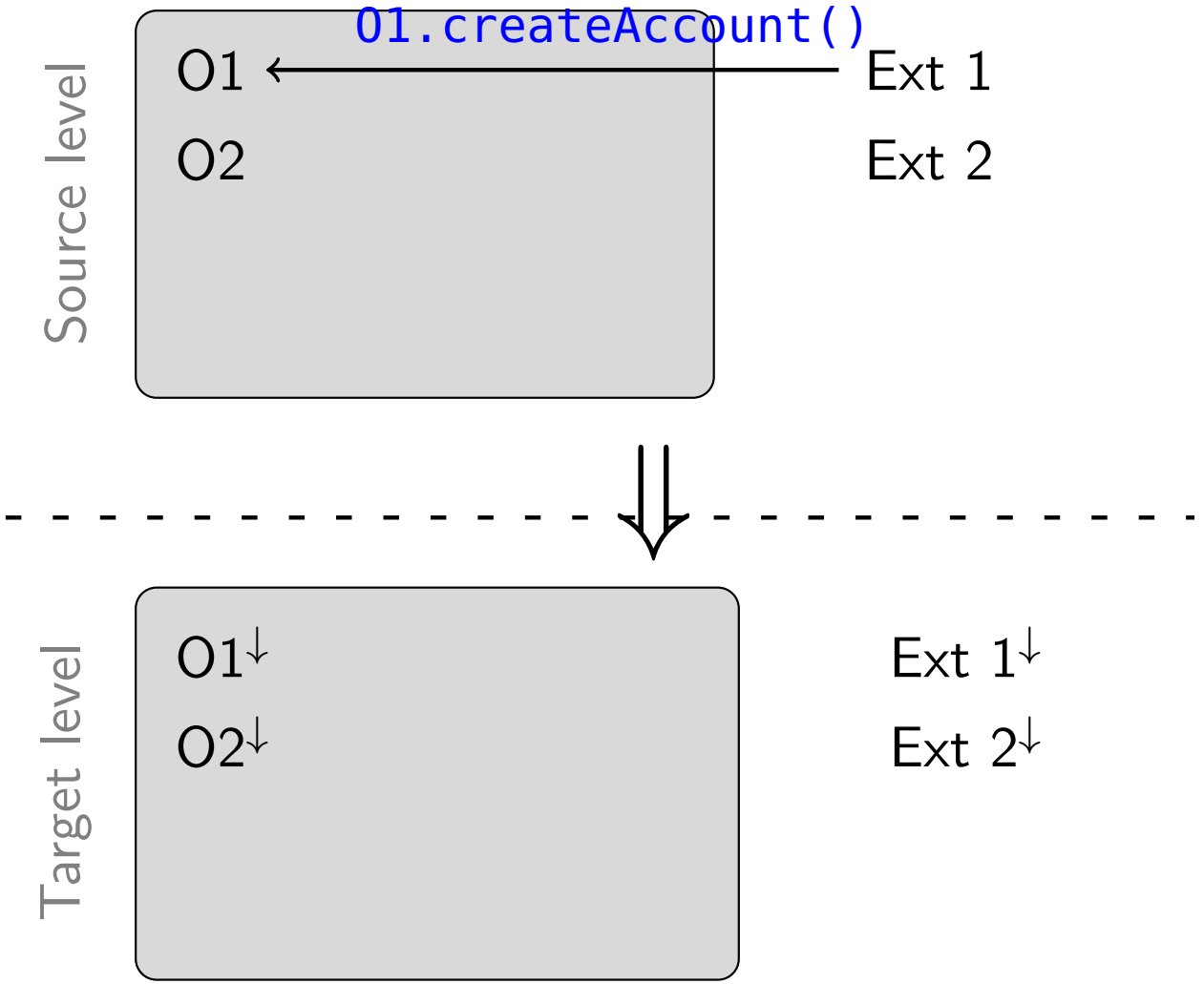
2 Secure (Fully Abstract) Compilation

3 What Can Go Wrong With...
- Dynamic Memory Allocation
- Exceptions
- Cross-package Inheritance

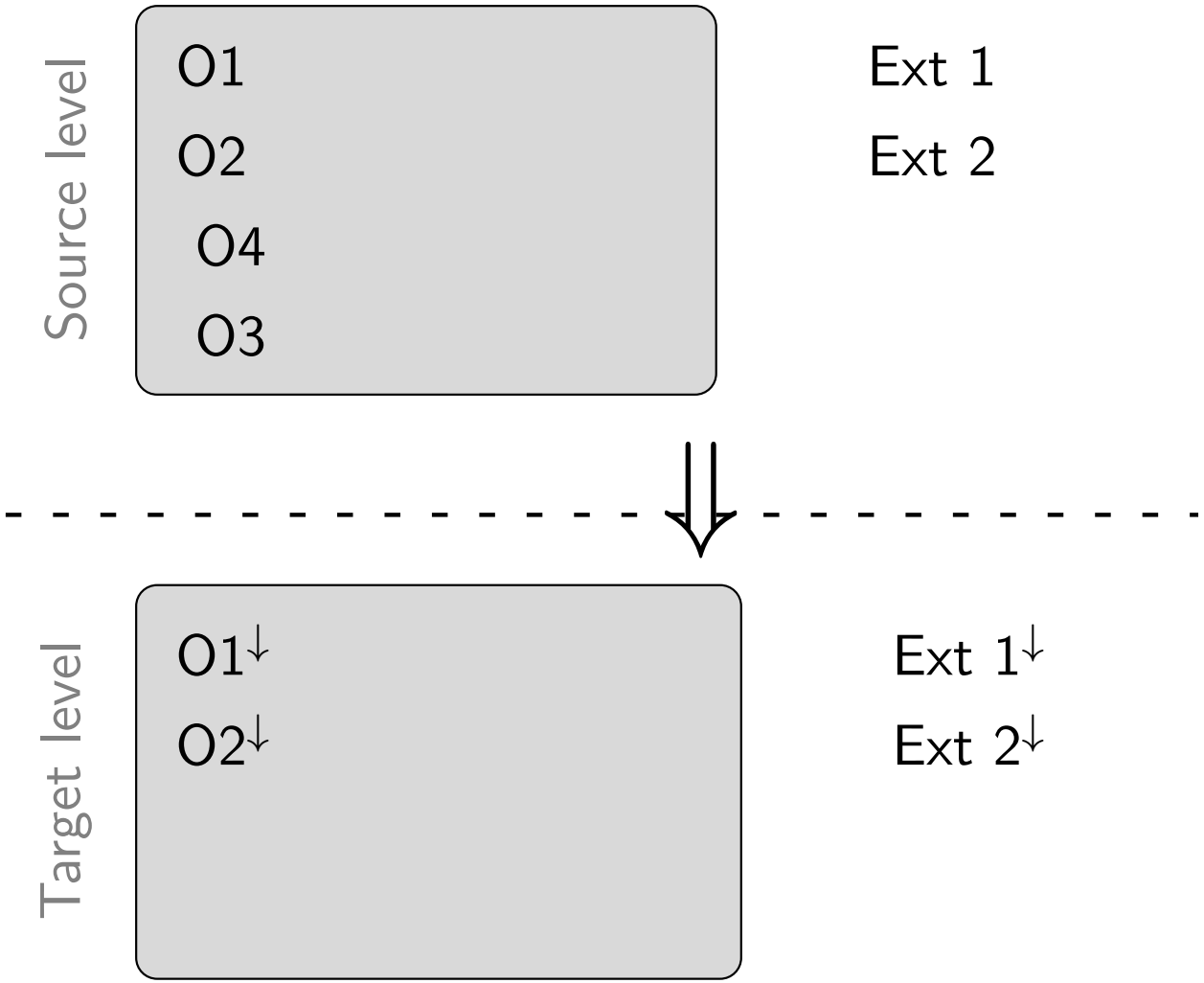
Dynamic Memory Allocation



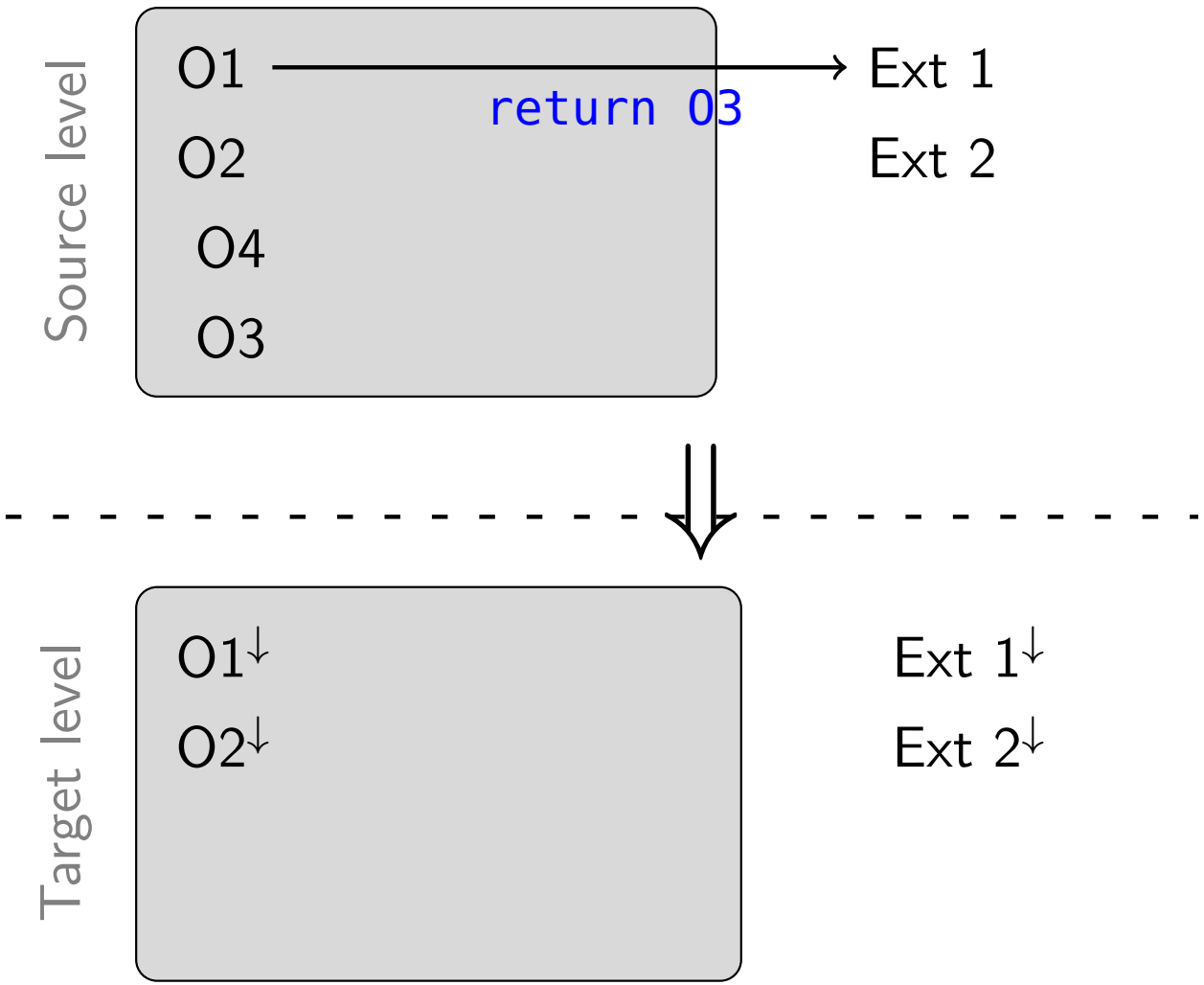
Dynamic Memory Allocation



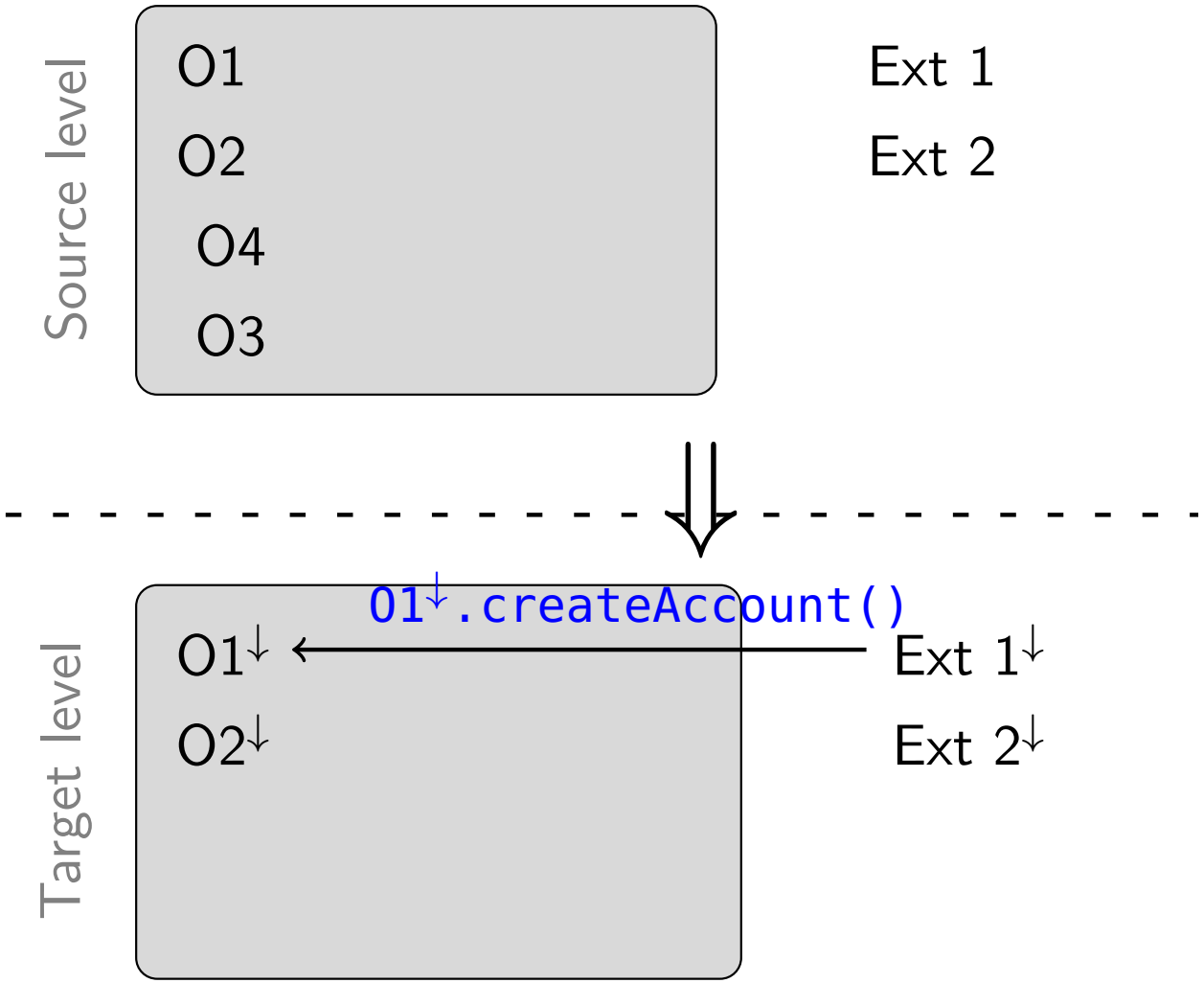
Dynamic Memory Allocation



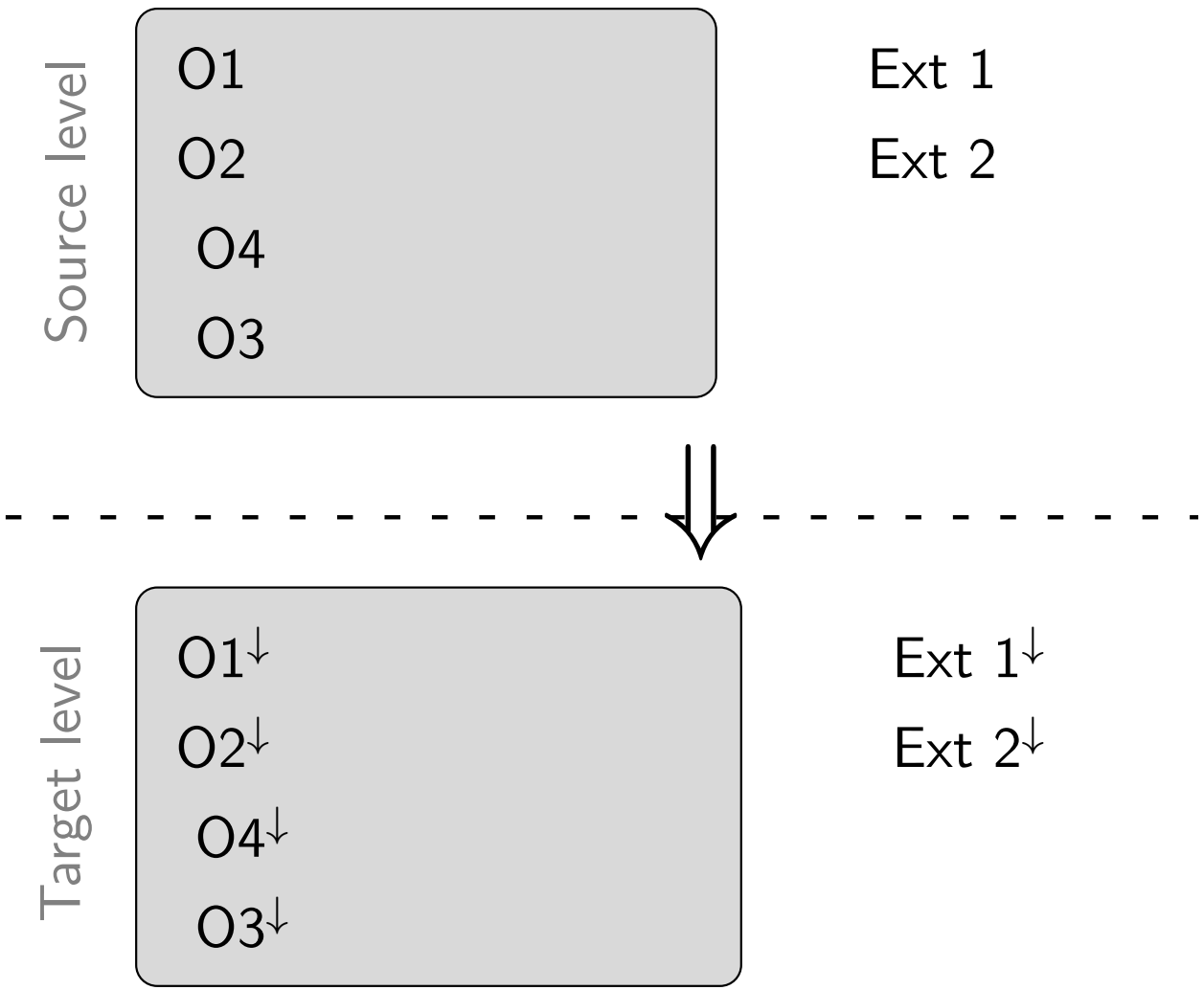
Dynamic Memory Allocation



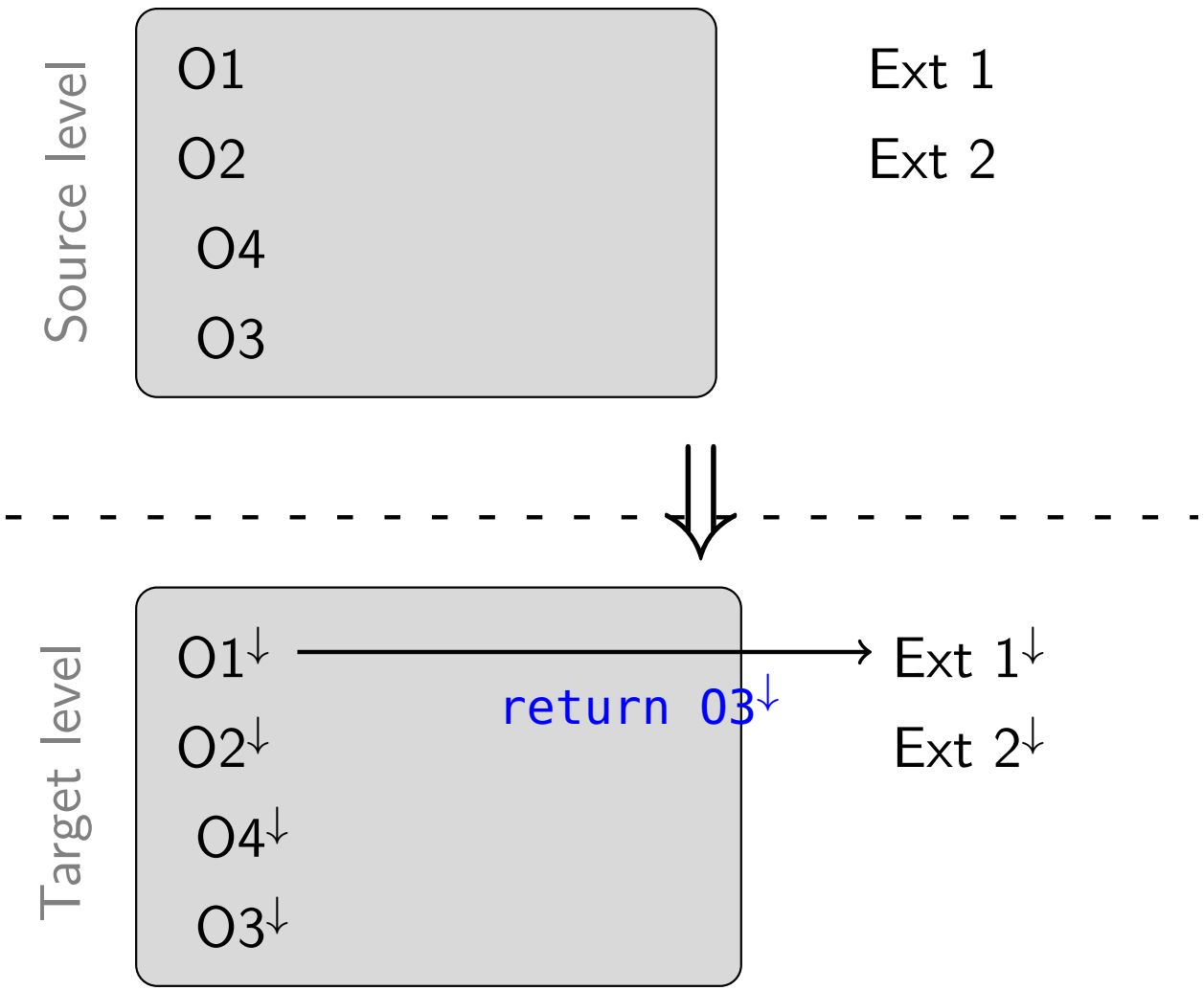
Dynamic Memory Allocation



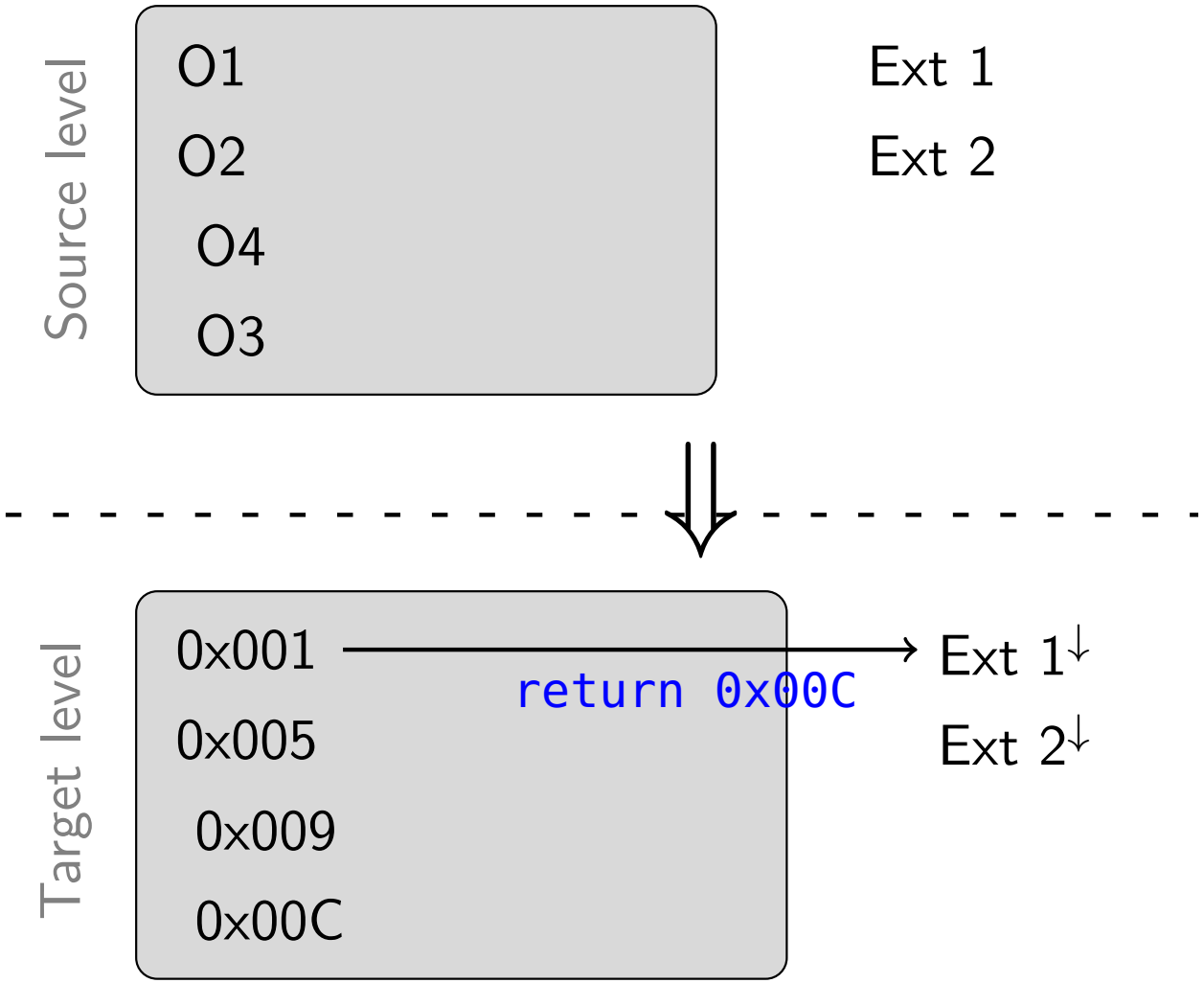
Dynamic Memory Allocation



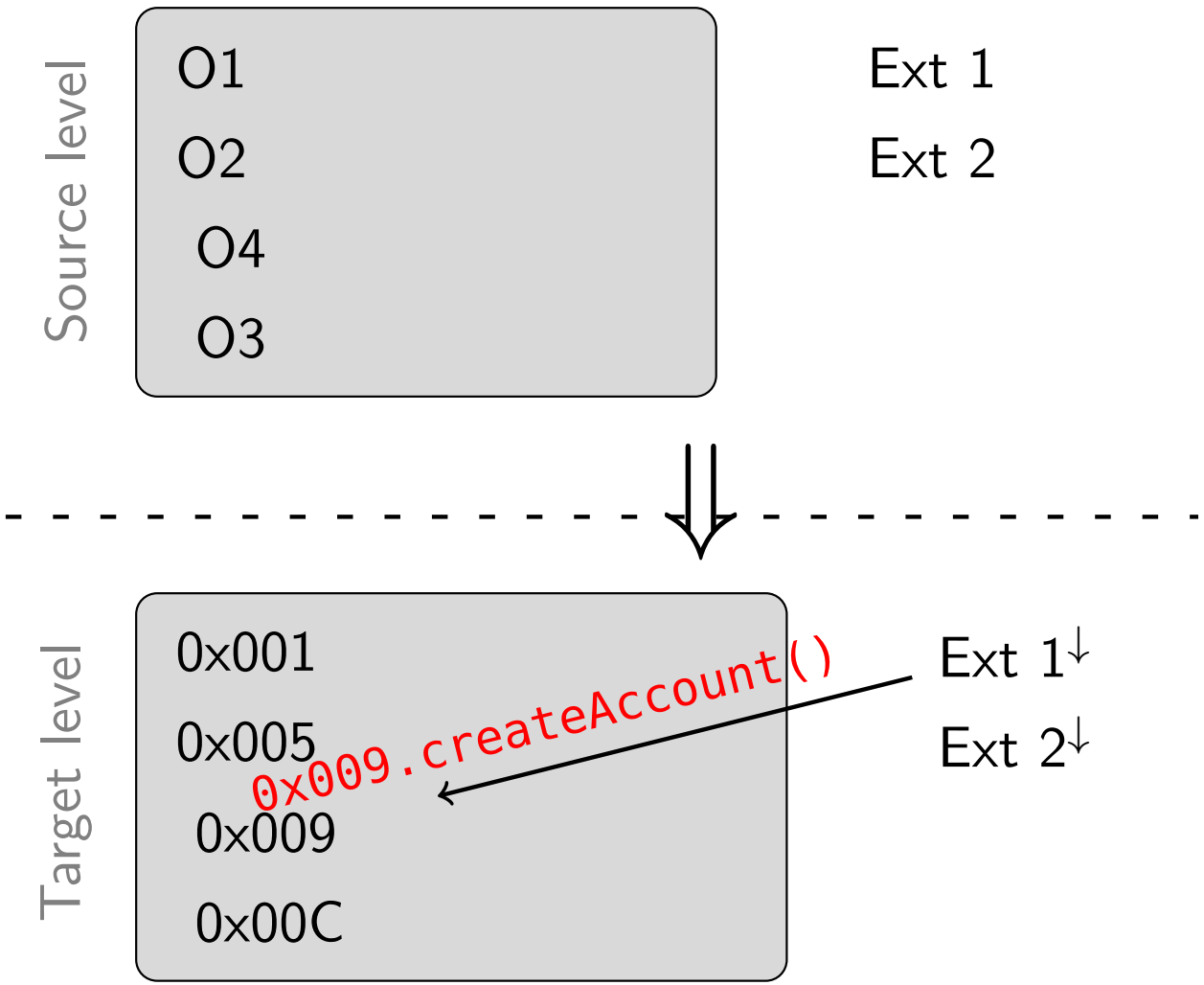
Dynamic Memory Allocation



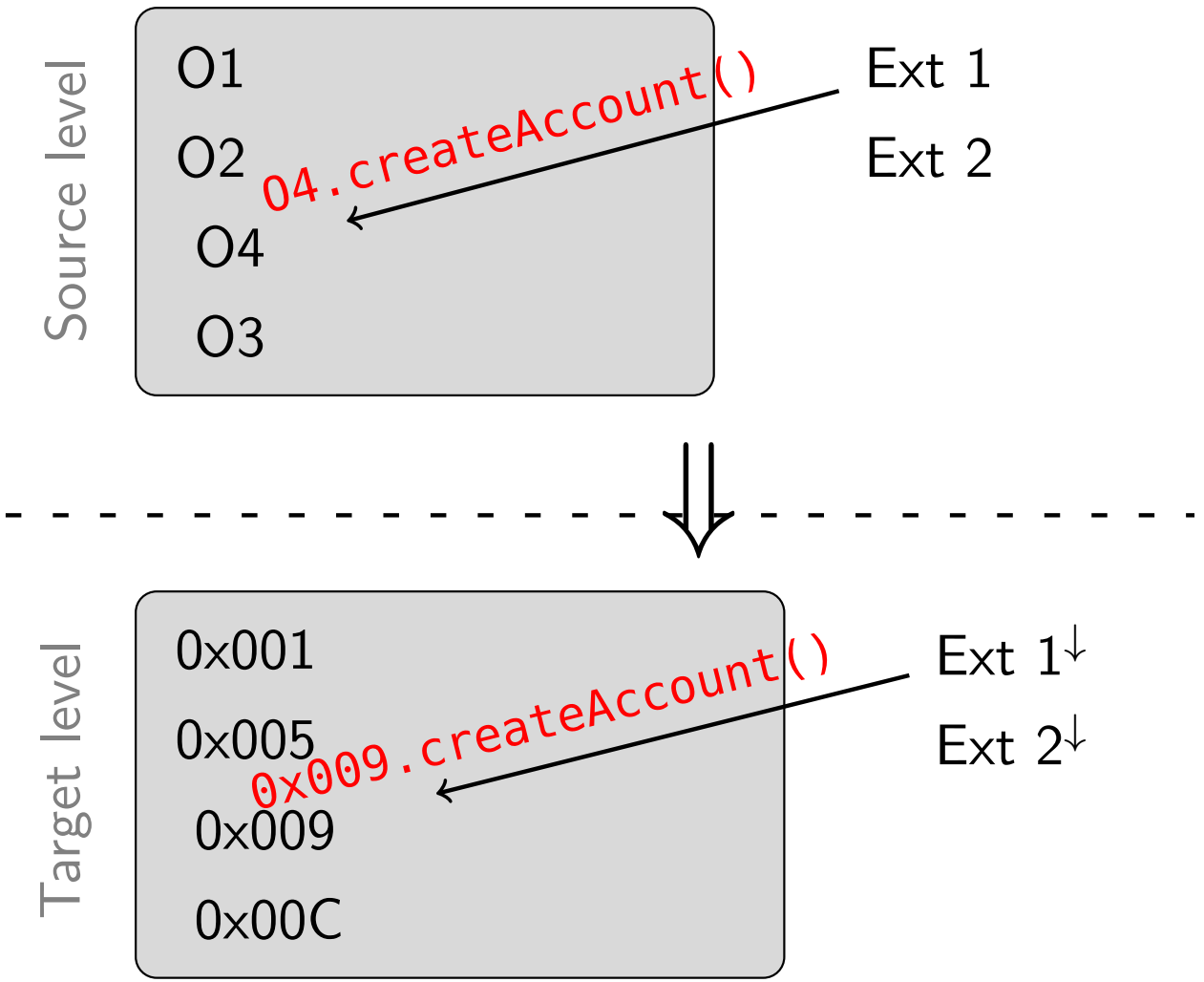
Dynamic Memory Allocation



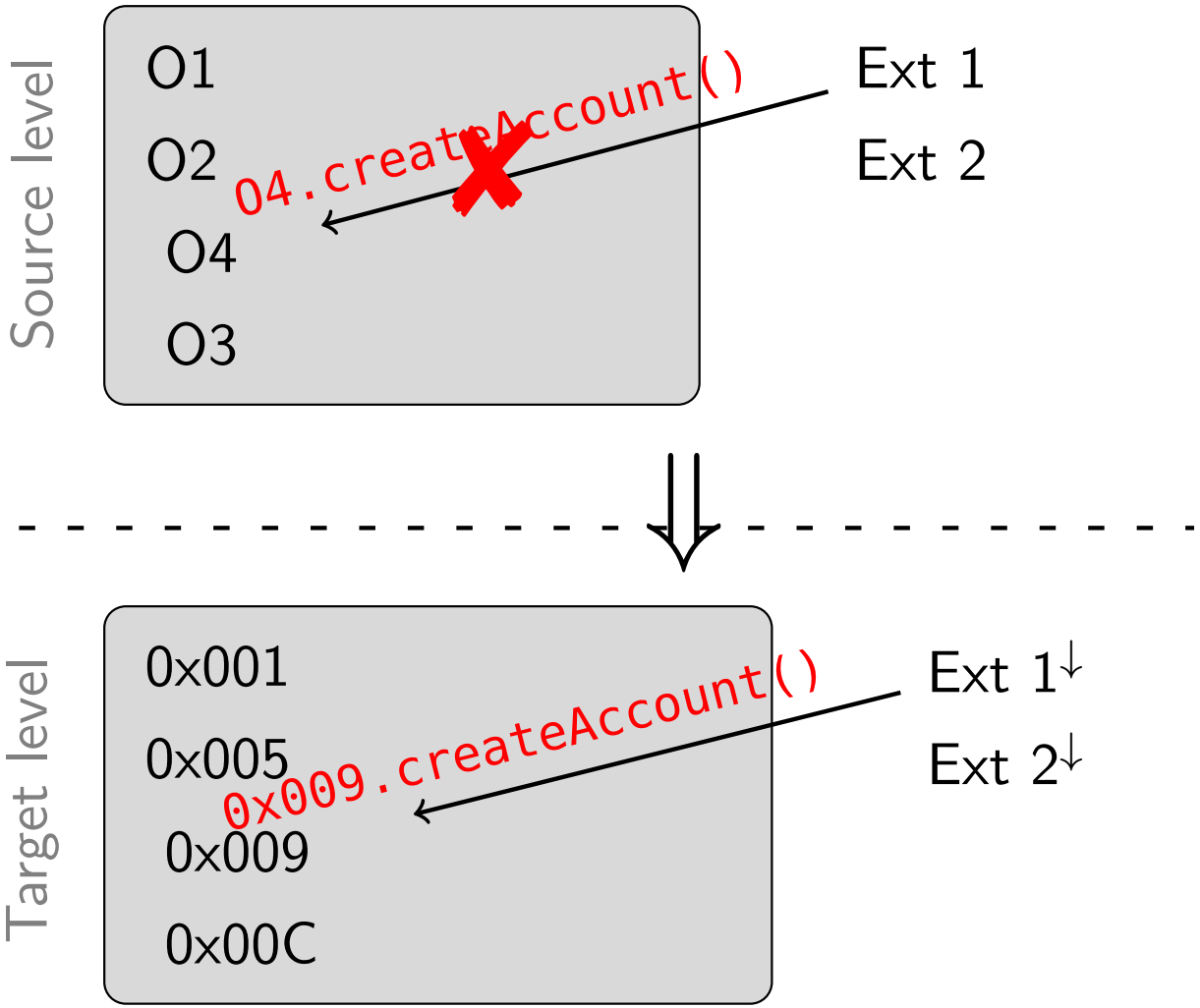
Dynamic Memory Allocation



Dynamic Memory Allocation

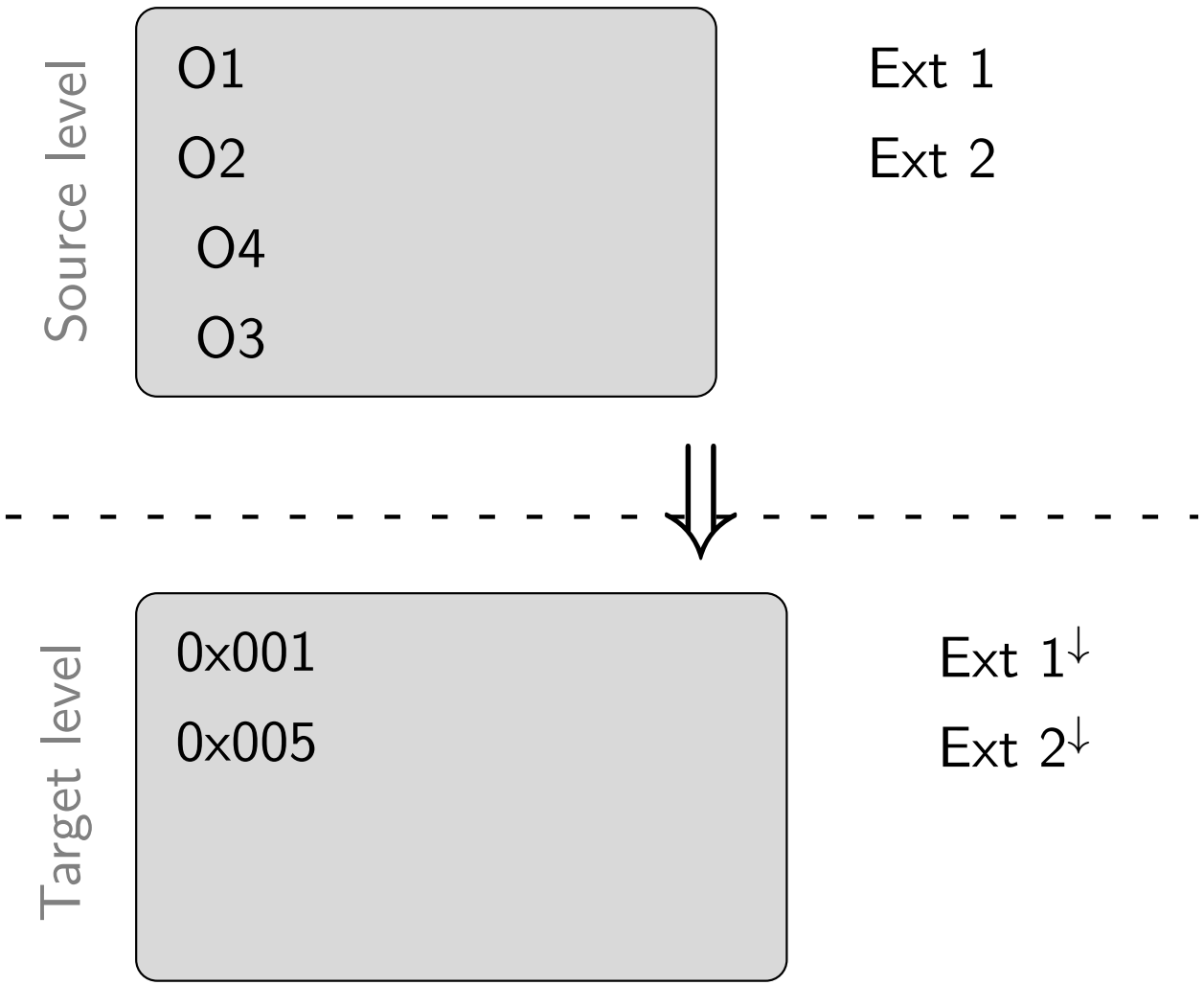


Dynamic Memory Allocation



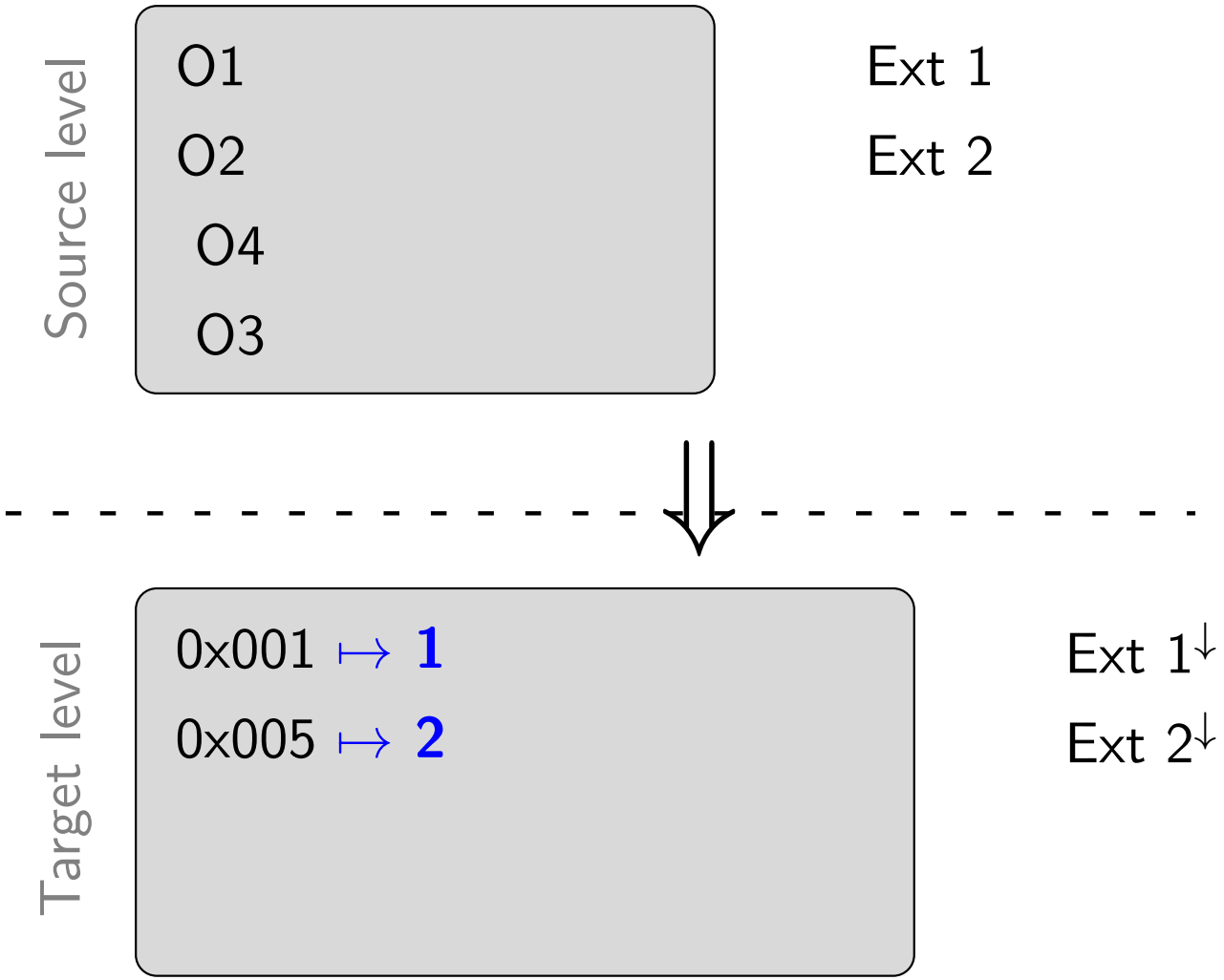
- Object id guessing

Dynamic Memory Allocation



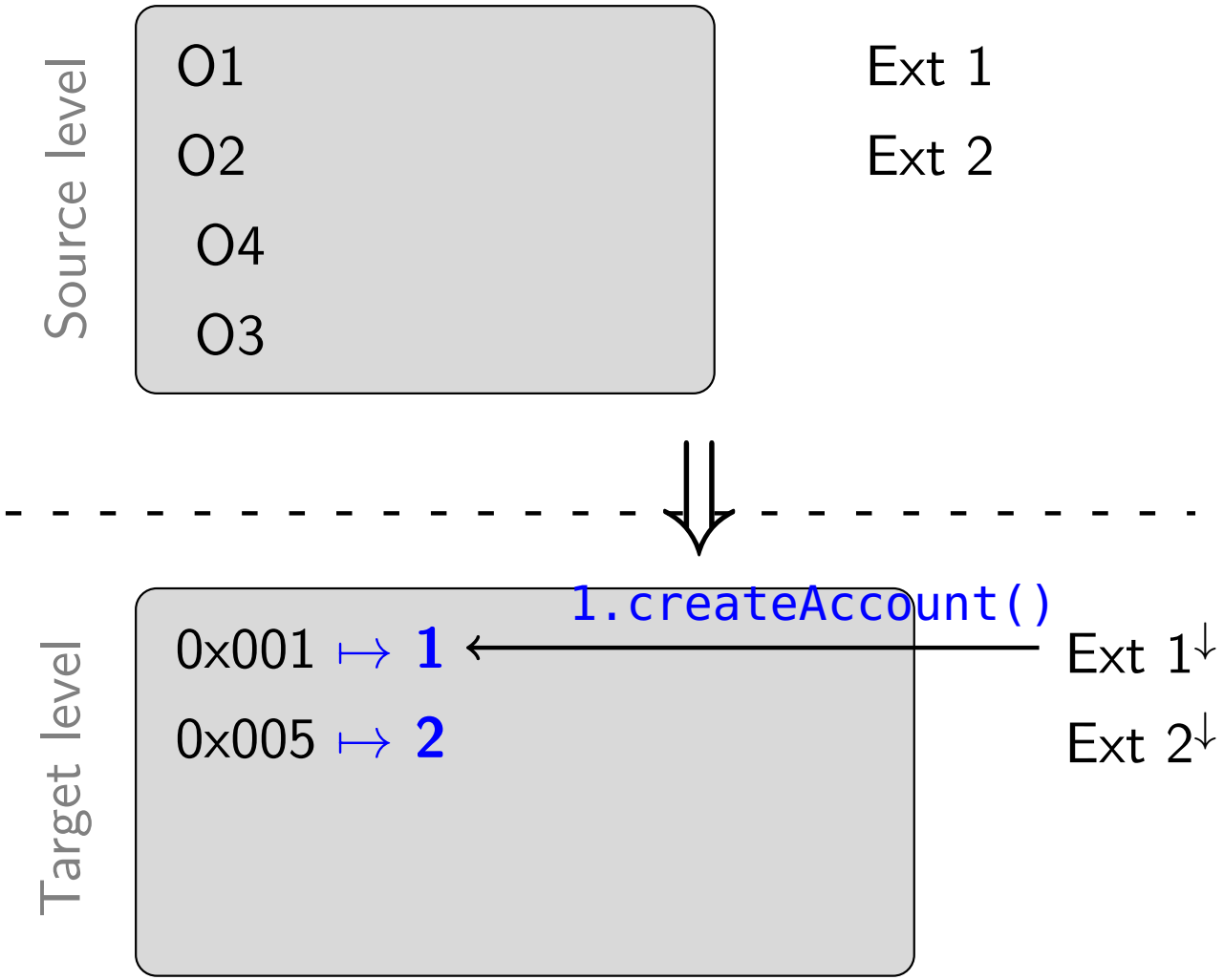
- Object id guessing
- map Oid to natural numbers

Dynamic Memory Allocation



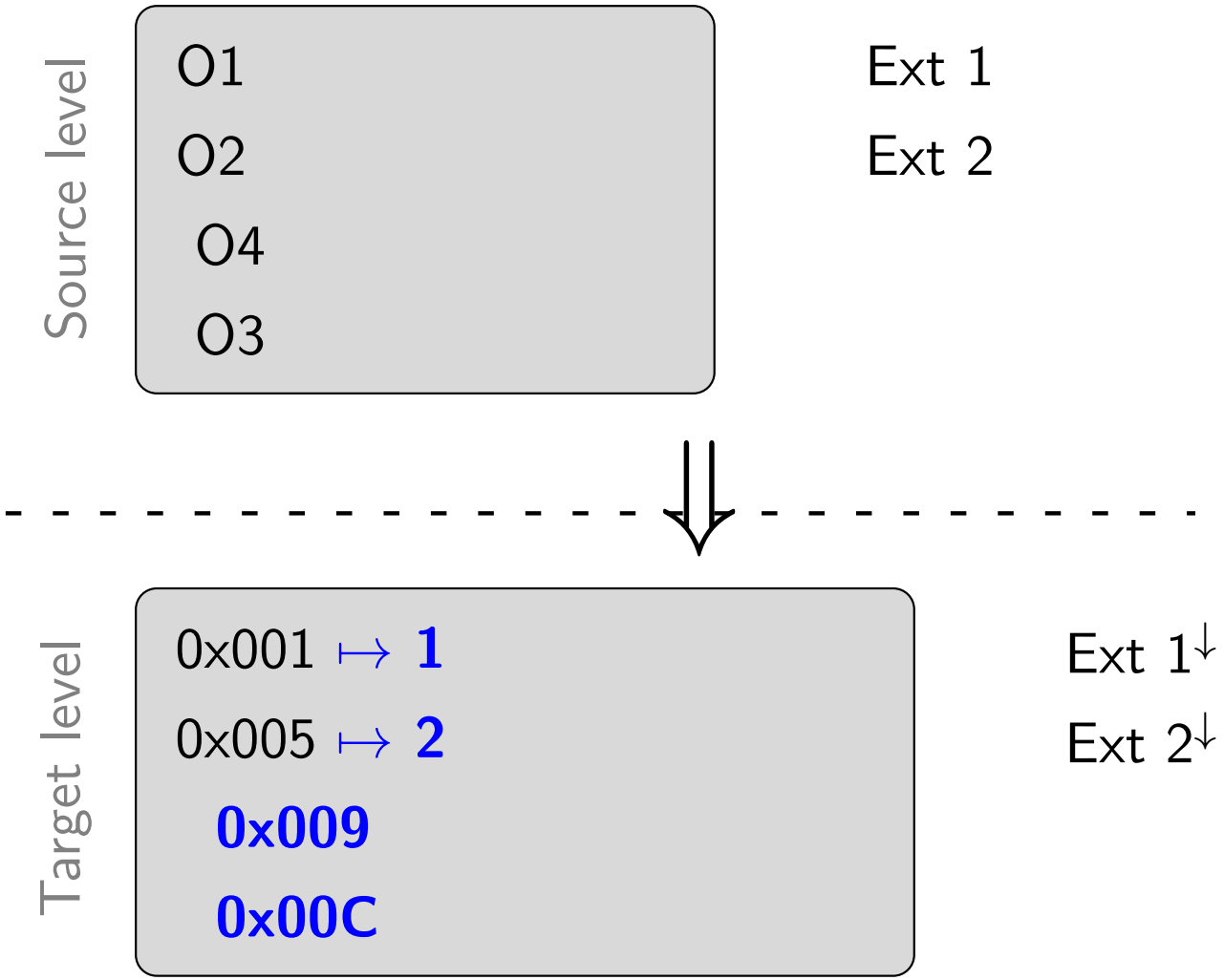
- Object id guessing
- map Oid to natural numbers

Dynamic Memory Allocation



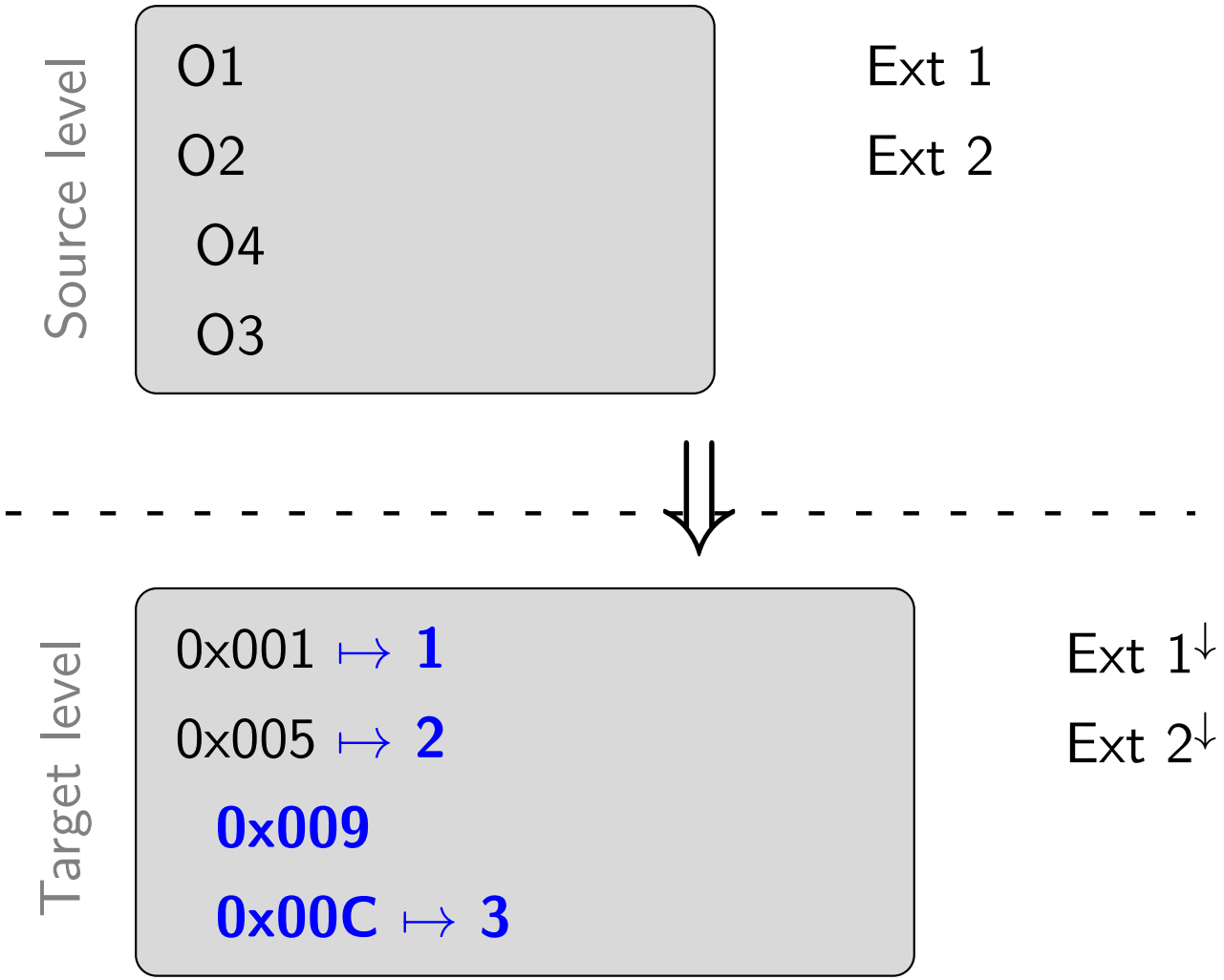
- Object id guessing
- map Oid to natural numbers

Dynamic Memory Allocation



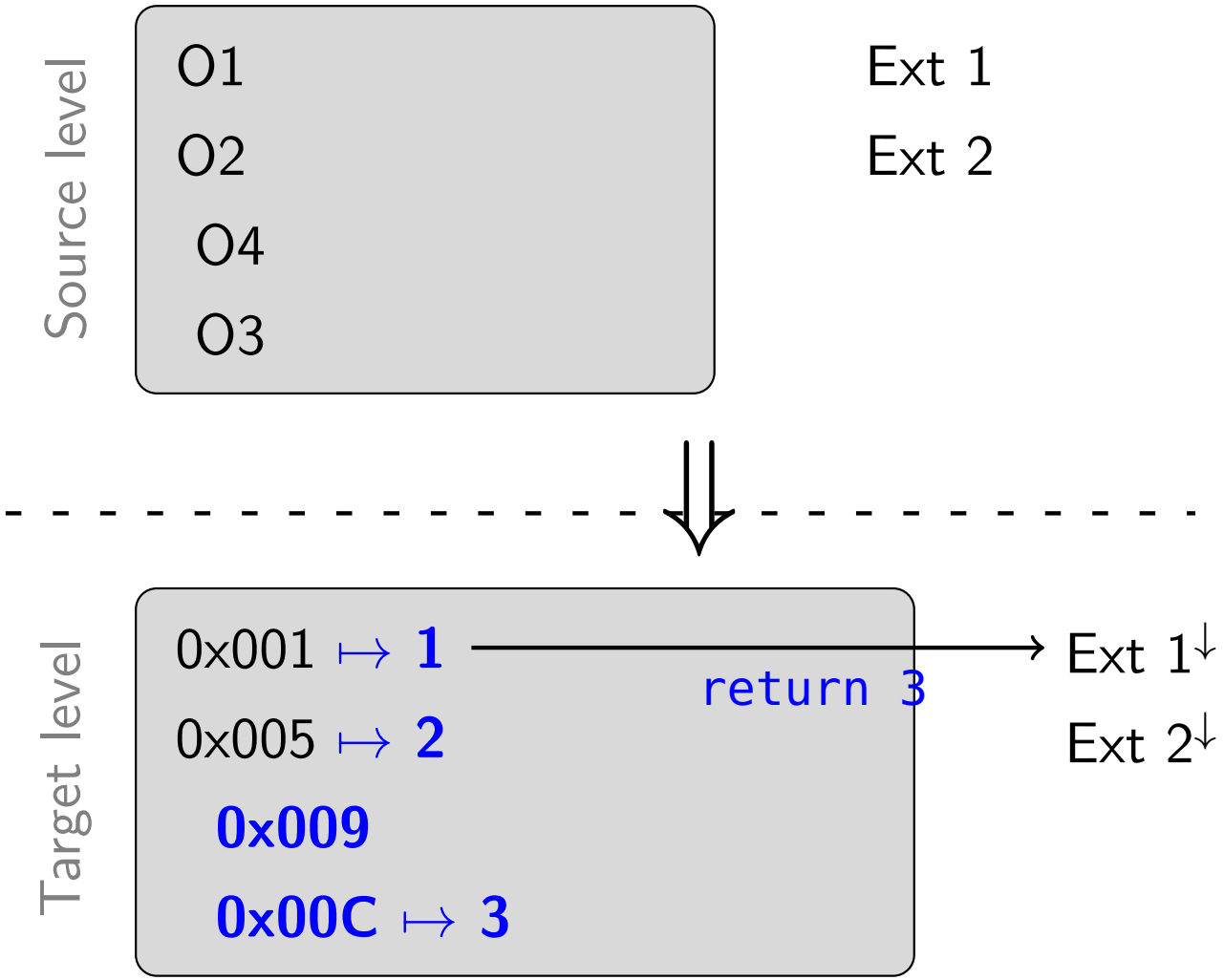
- Object id guessing
- map Oid to natural numbers

Dynamic Memory Allocation



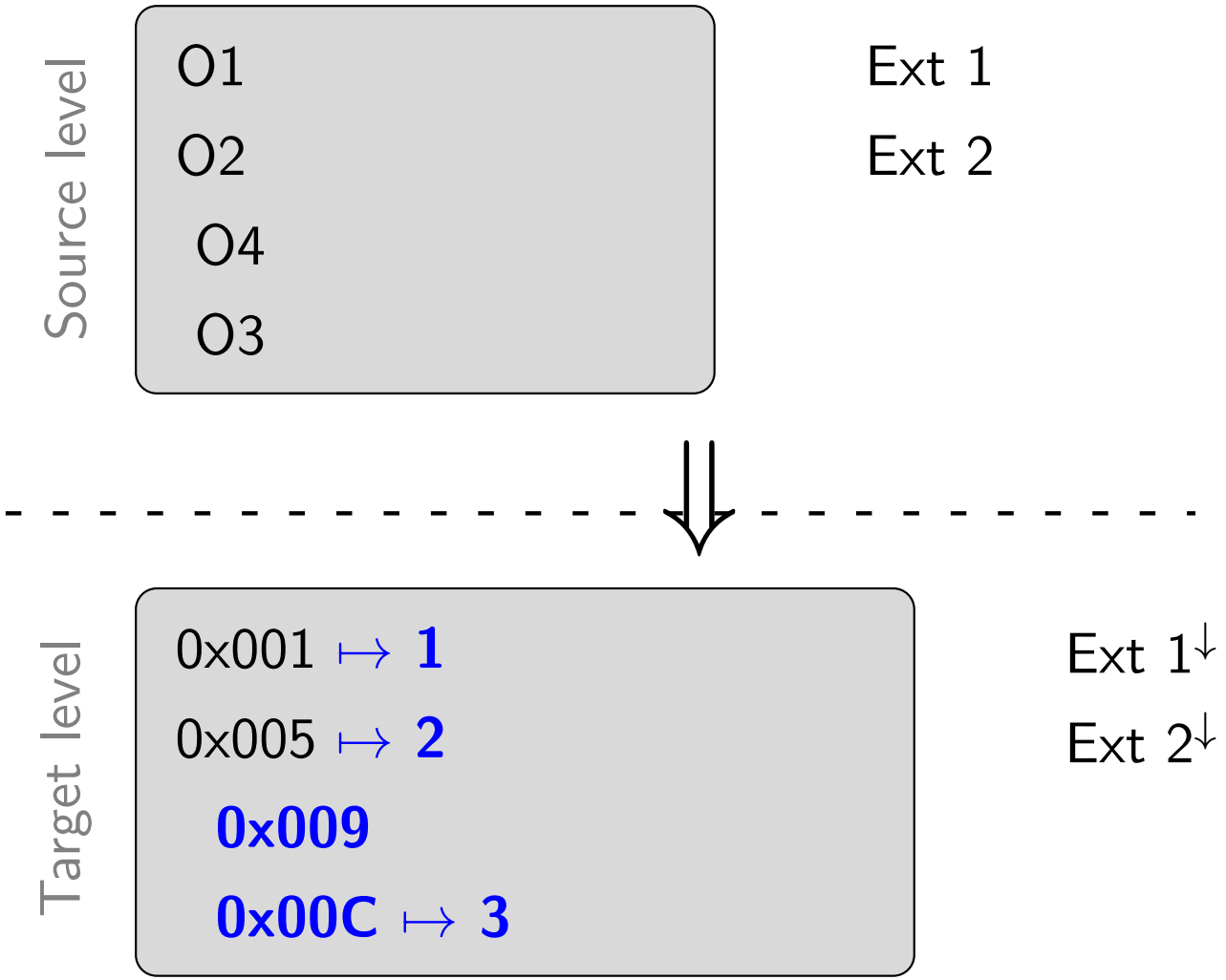
- Object id guessing
- map Oid to natural numbers
- add Oid to map

Dynamic Memory Allocation



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

Dynamic Memory Allocation

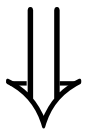


- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup ($\mathbf{O}(1)$) when number is received

Dynamic Memory Allocation

Source level

O1 : Account
O2 : Pair
O4
O3



Target level

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

Ext 1

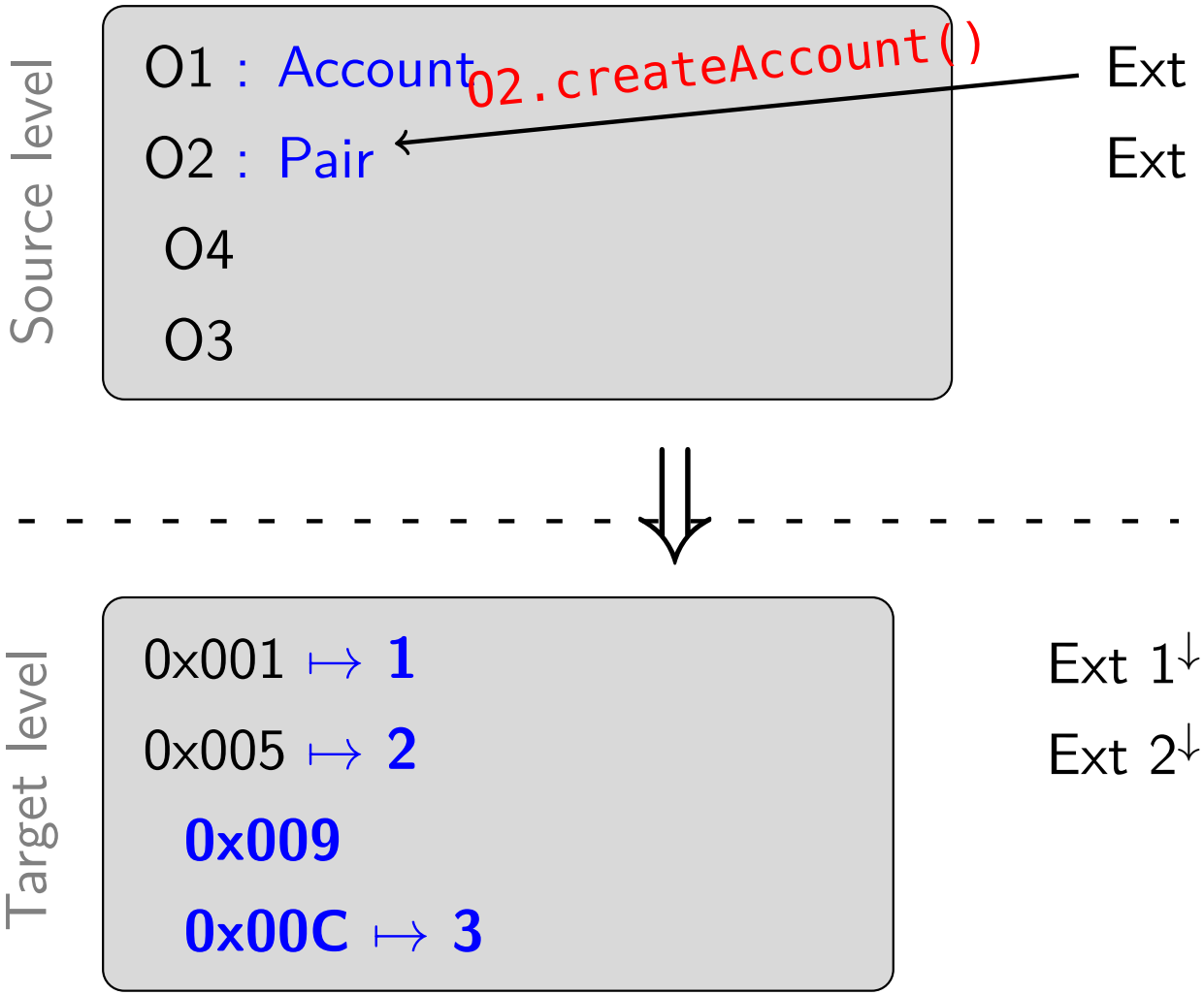
Ext 2

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

Ext $1^{\downarrow}$

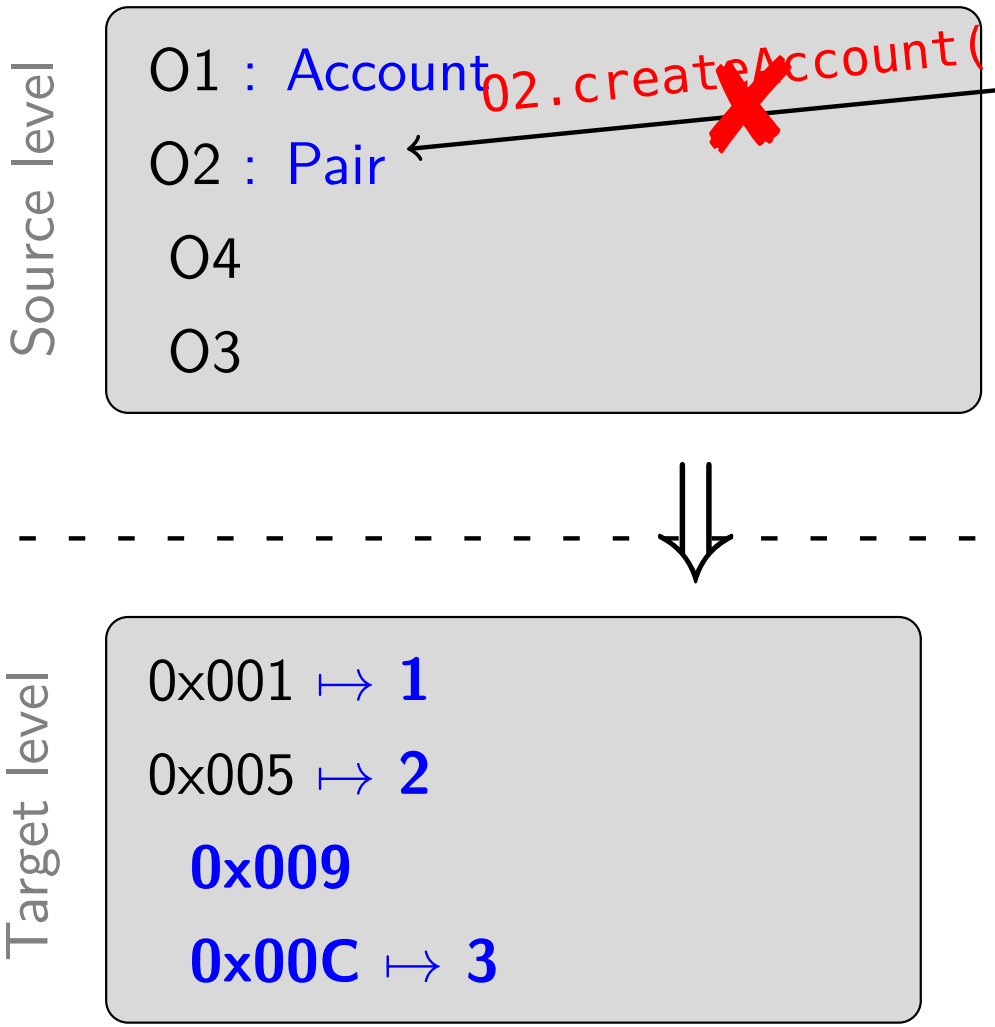
Ext $2^{\downarrow}$

Dynamic Memory Allocation



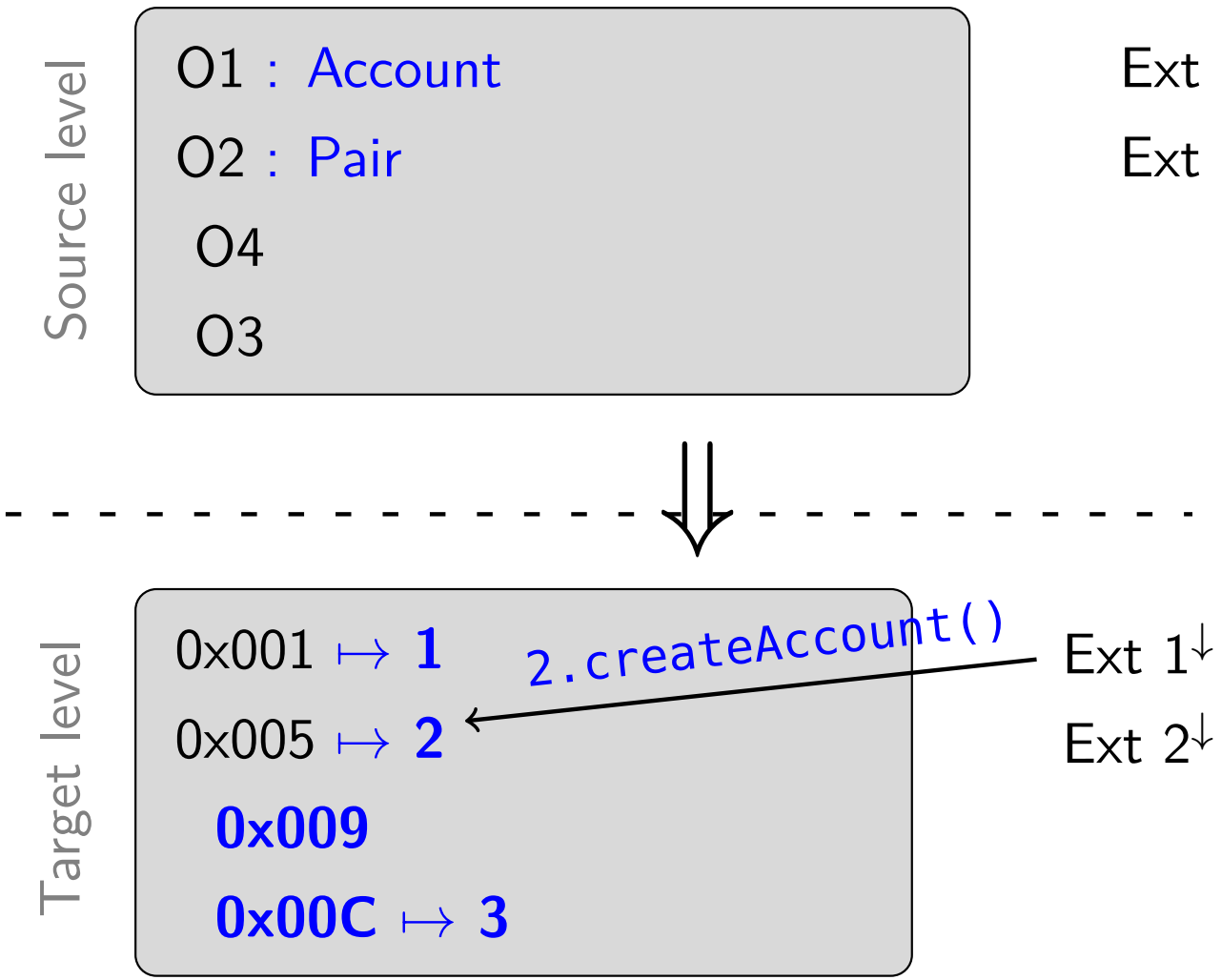
- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup ($\mathbf{O}(1)$) when number is received

Dynamic Memory Allocation



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

Dynamic Memory Allocation

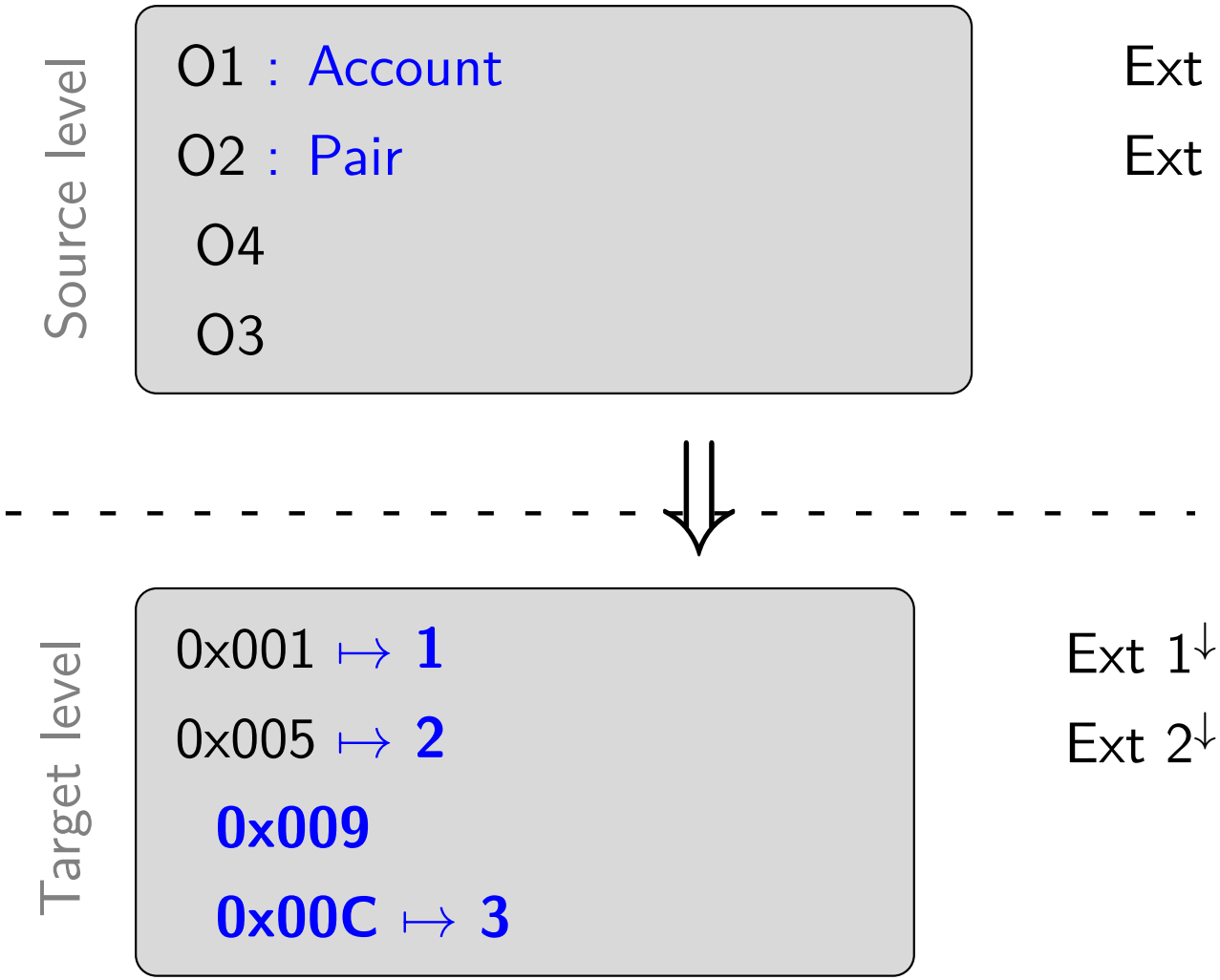


Ext 1

Ext 2

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

Dynamic Memory Allocation



Ext 1

Ext 2

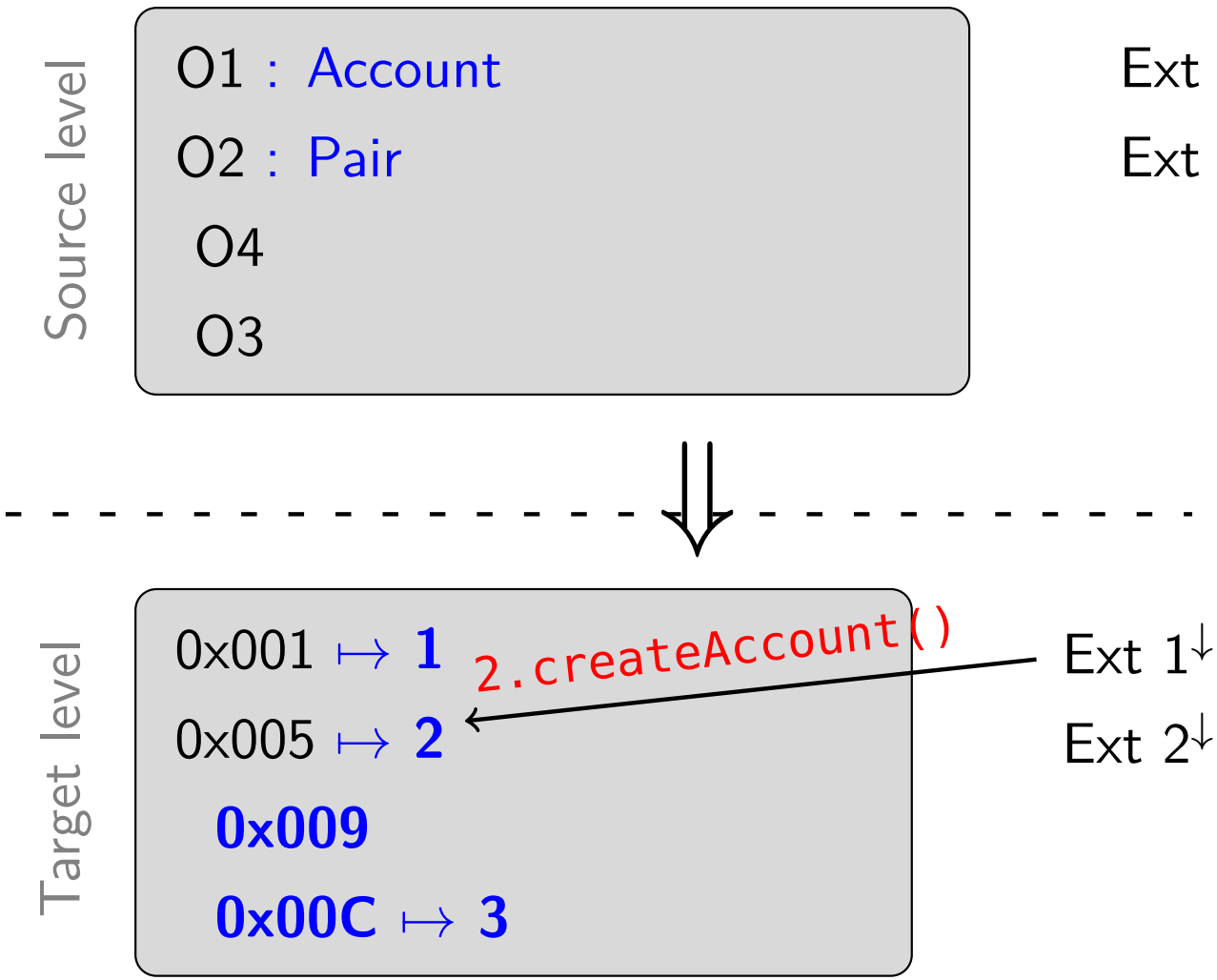
- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup ($\mathbf{O}(1)$) when number is received

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

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

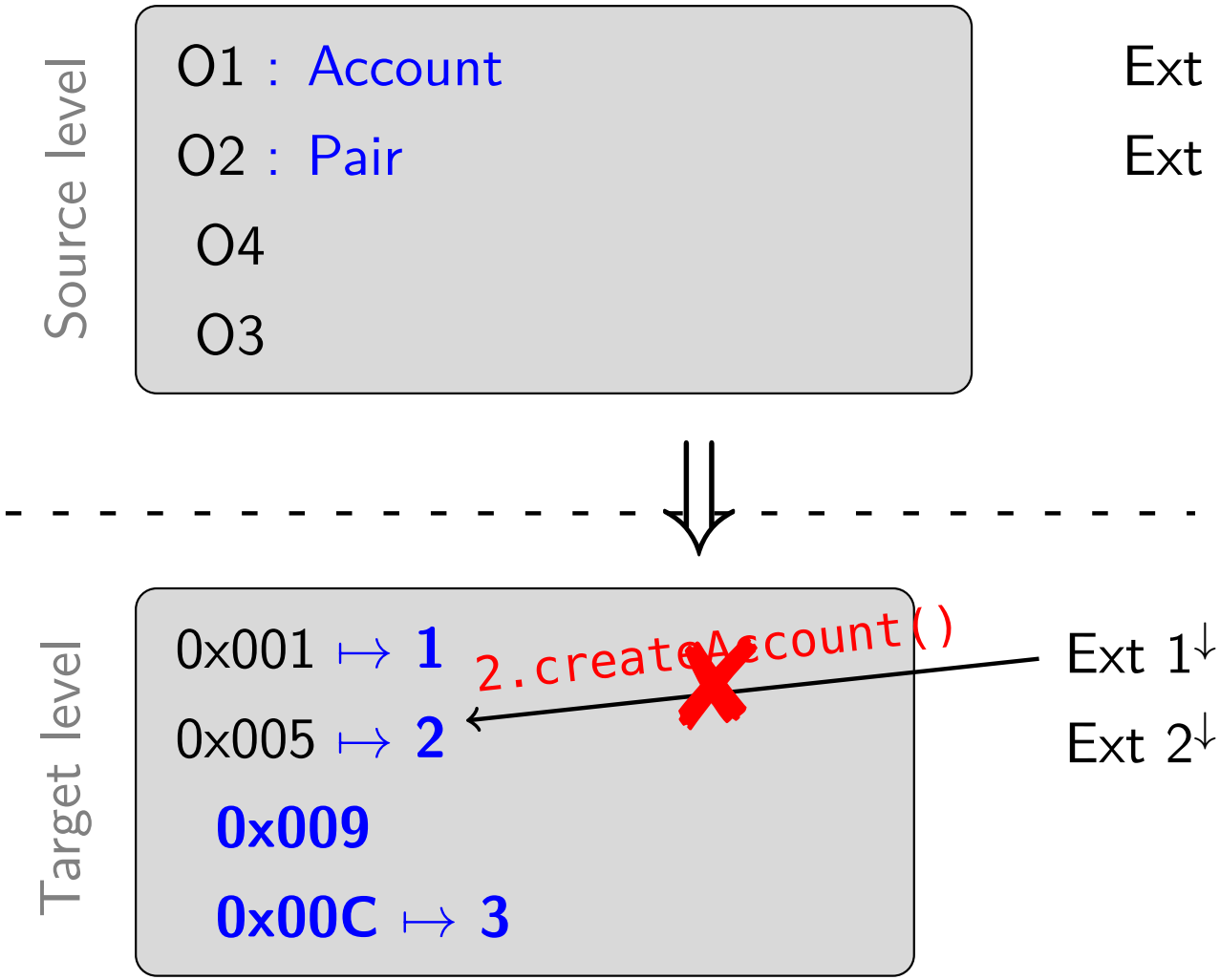
- dynamic typecheck for: current object

Dynamic Memory Allocation



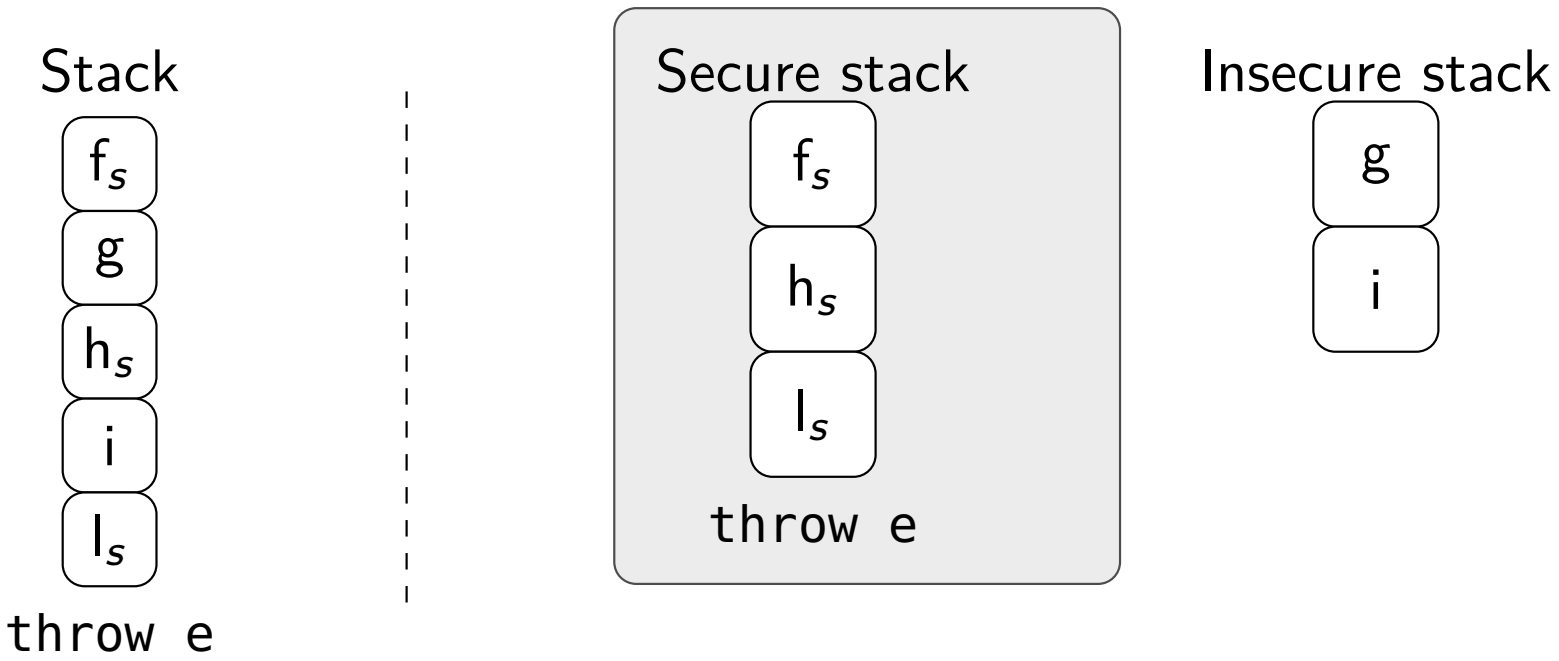
- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup ($\mathbf{O}(1)$) when number is received
- dynamic typecheck for: current object arguments

Dynamic Memory Allocation

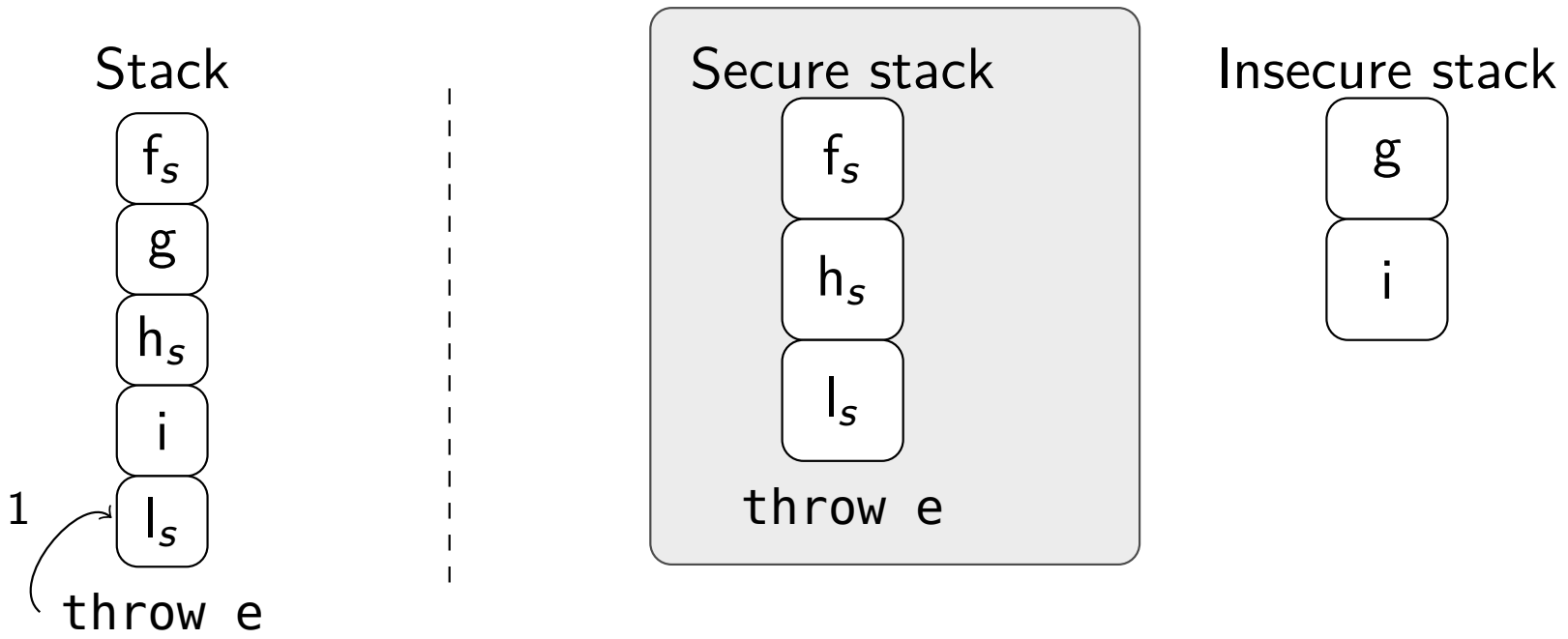


- Object id guessing
- map Oid to natural numbers
- add Oid to map
- lookup ($\mathbf{O}(1)$) when number is received
- dynamic typecheck for: current object arguments
- no need of extra information

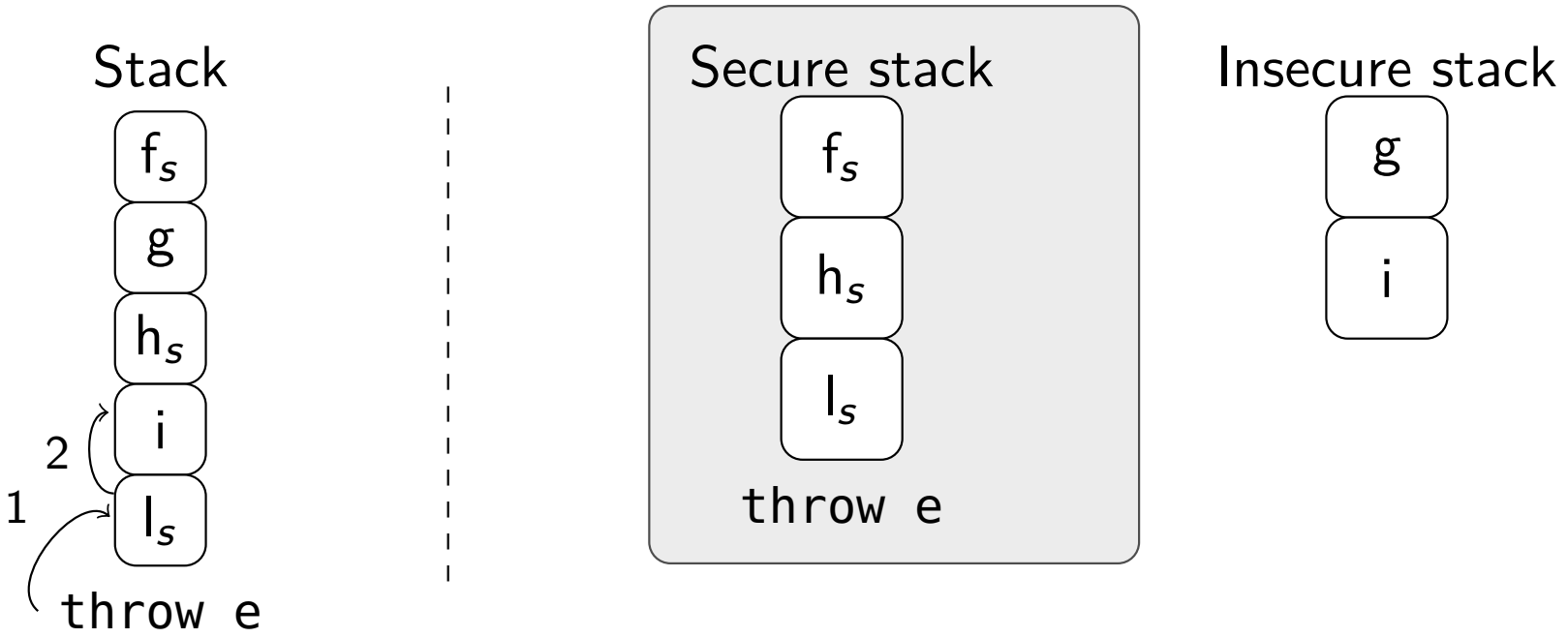
Exceptions


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

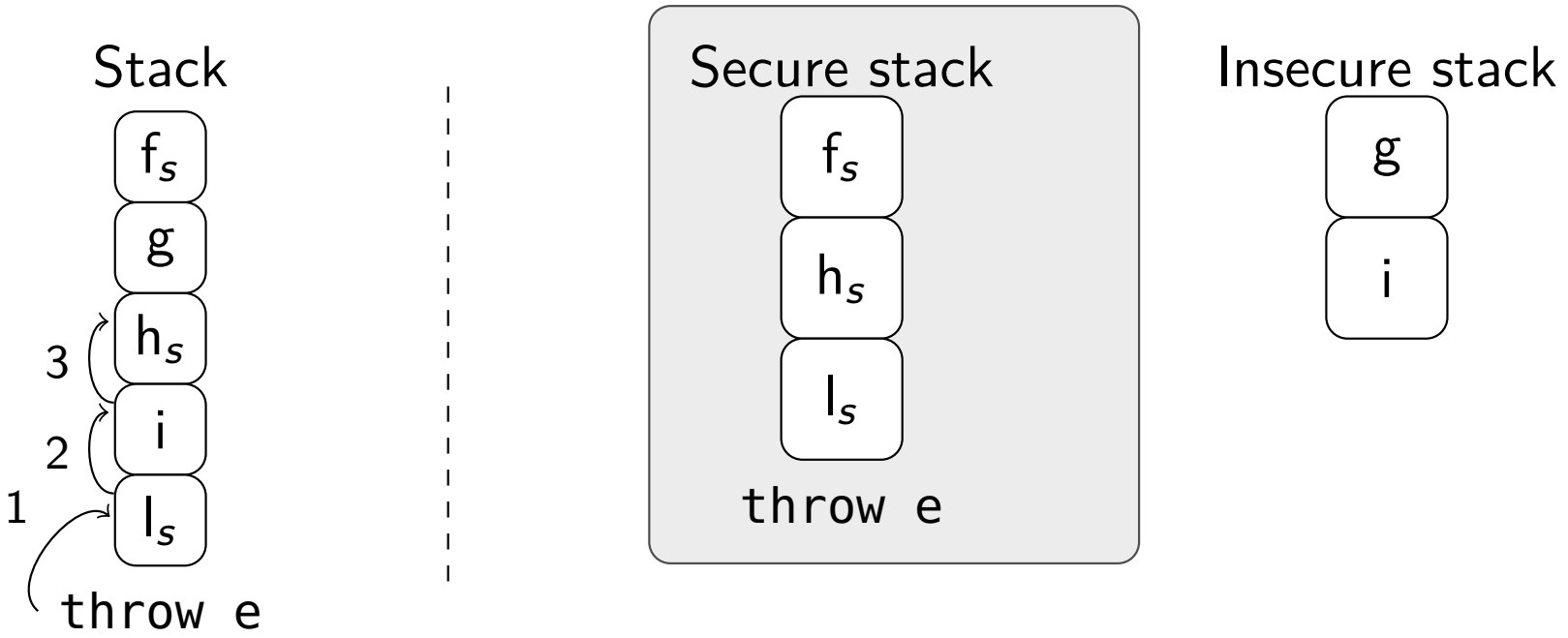
Exceptions


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

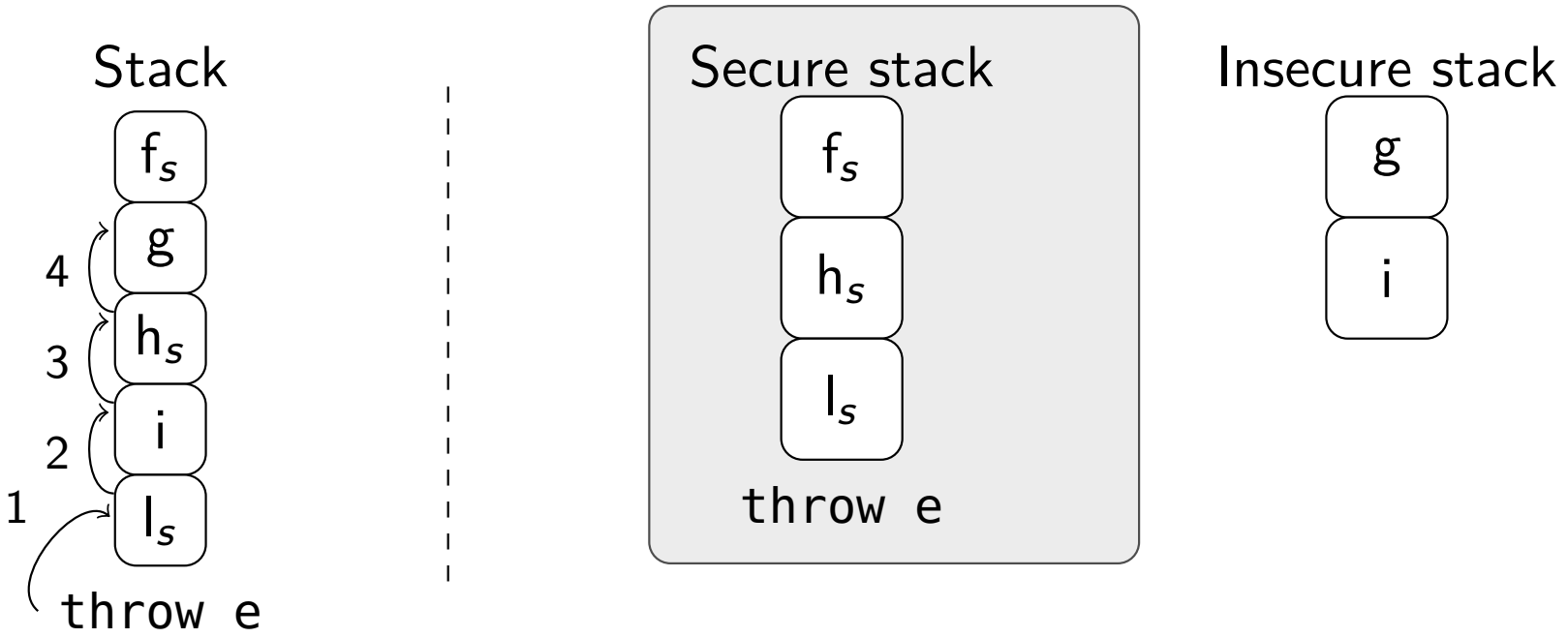
Exceptions


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

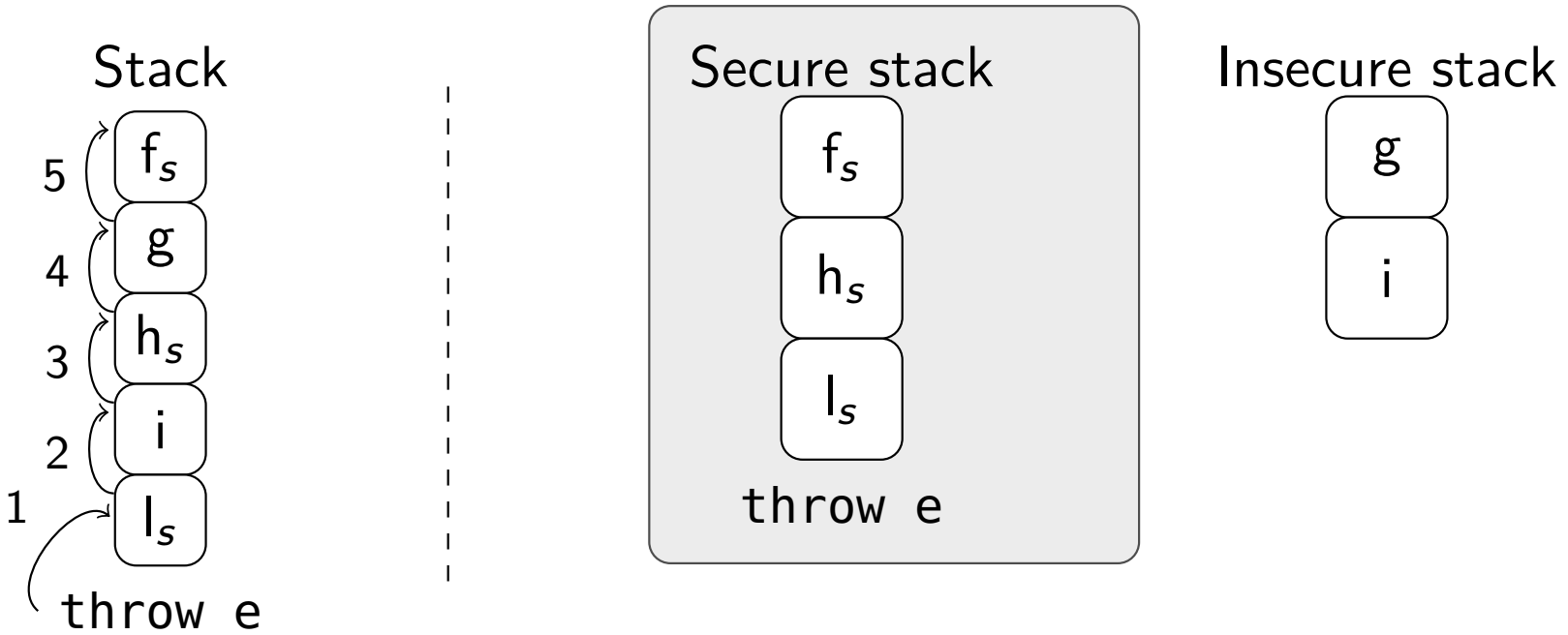
Exceptions


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

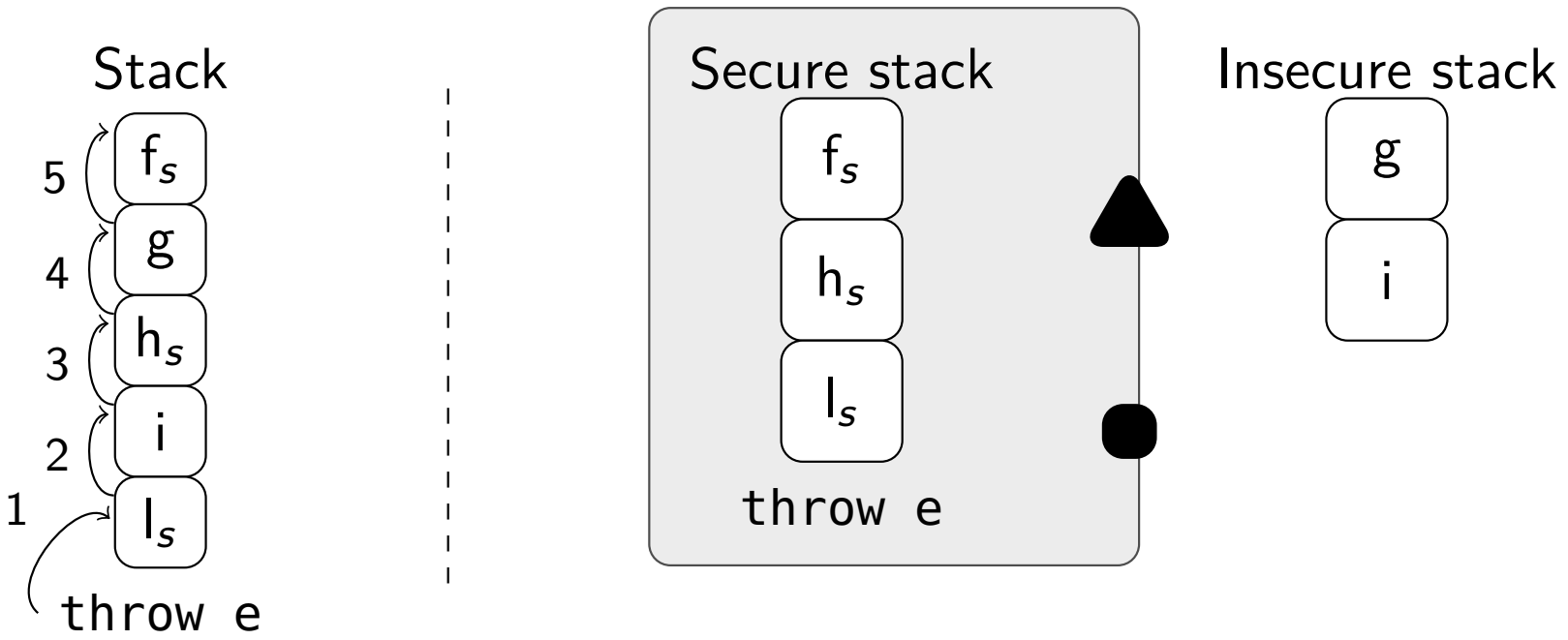
Exceptions


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

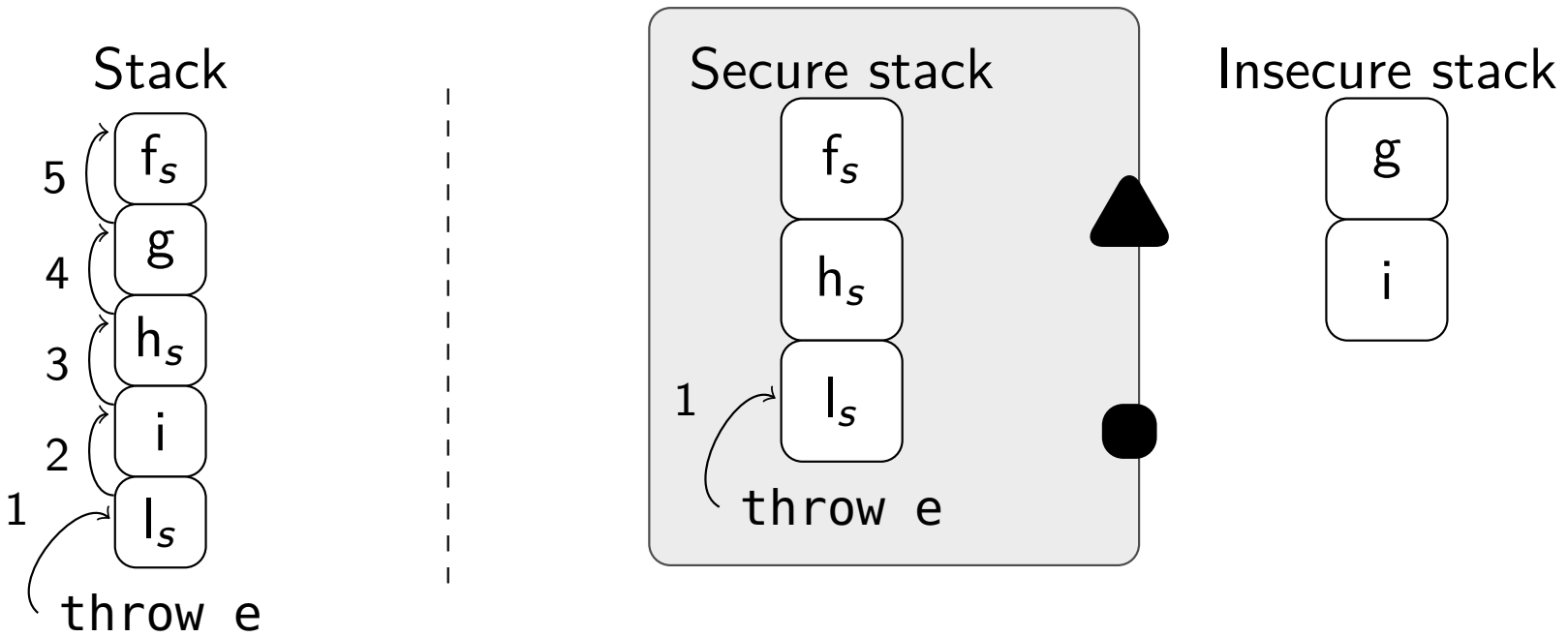
Exceptions


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

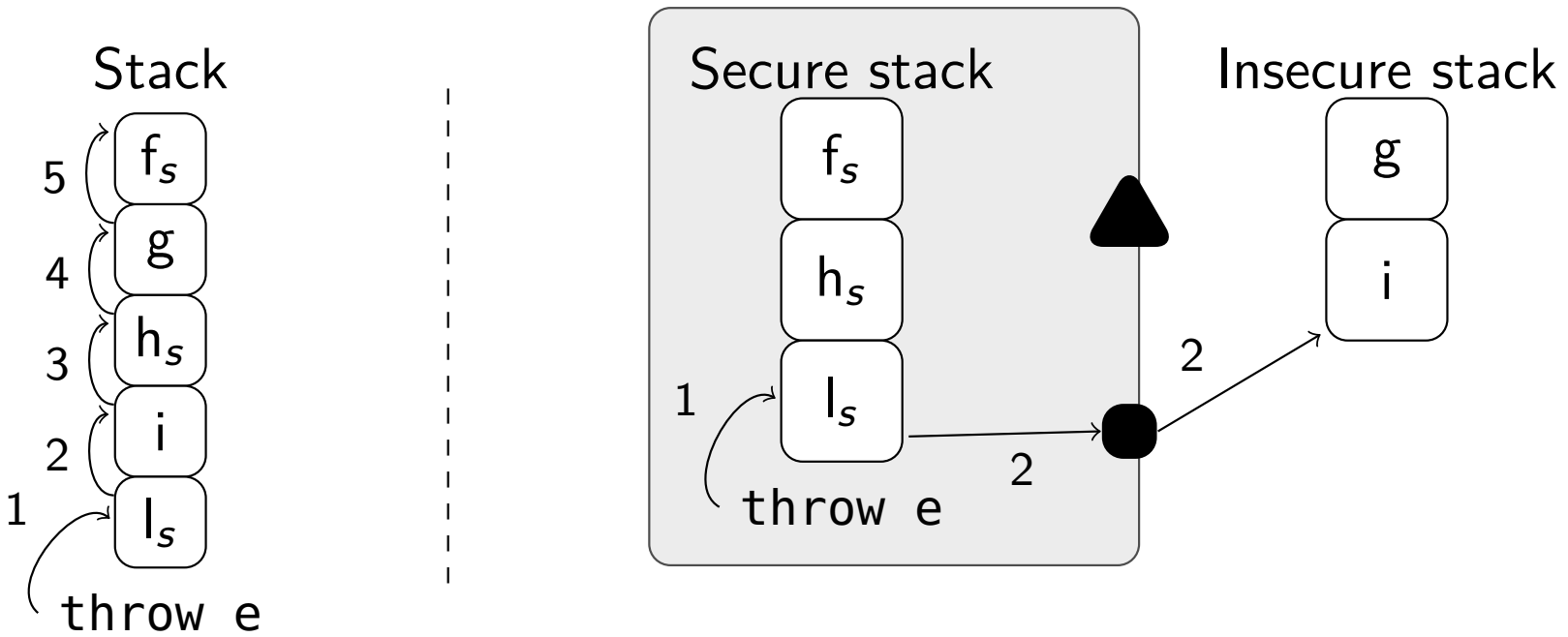
Exceptions


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

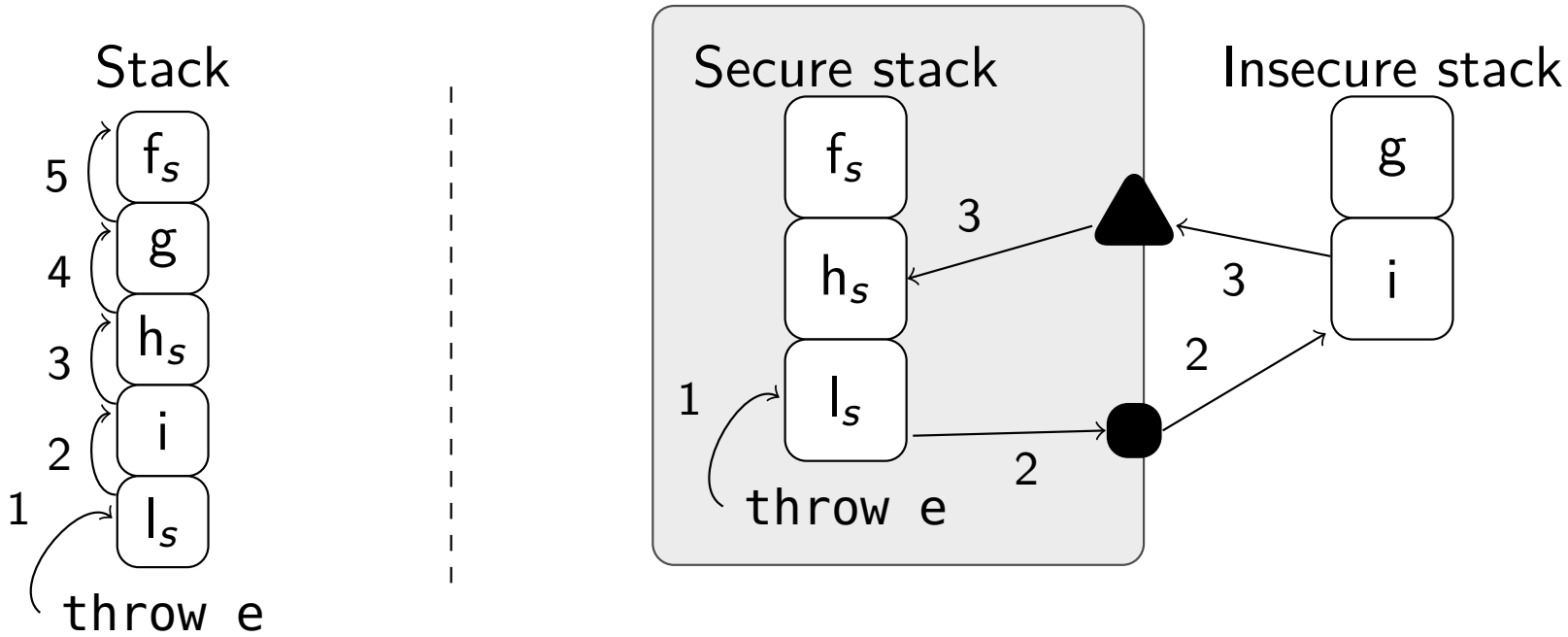

Stack
5
4
3
2
1
f_s
g
h_s
i
l_s
throw e
Secure stack
f_s
h_s
1
l_s
throw e
Insecure stack
g
i

Exceptions



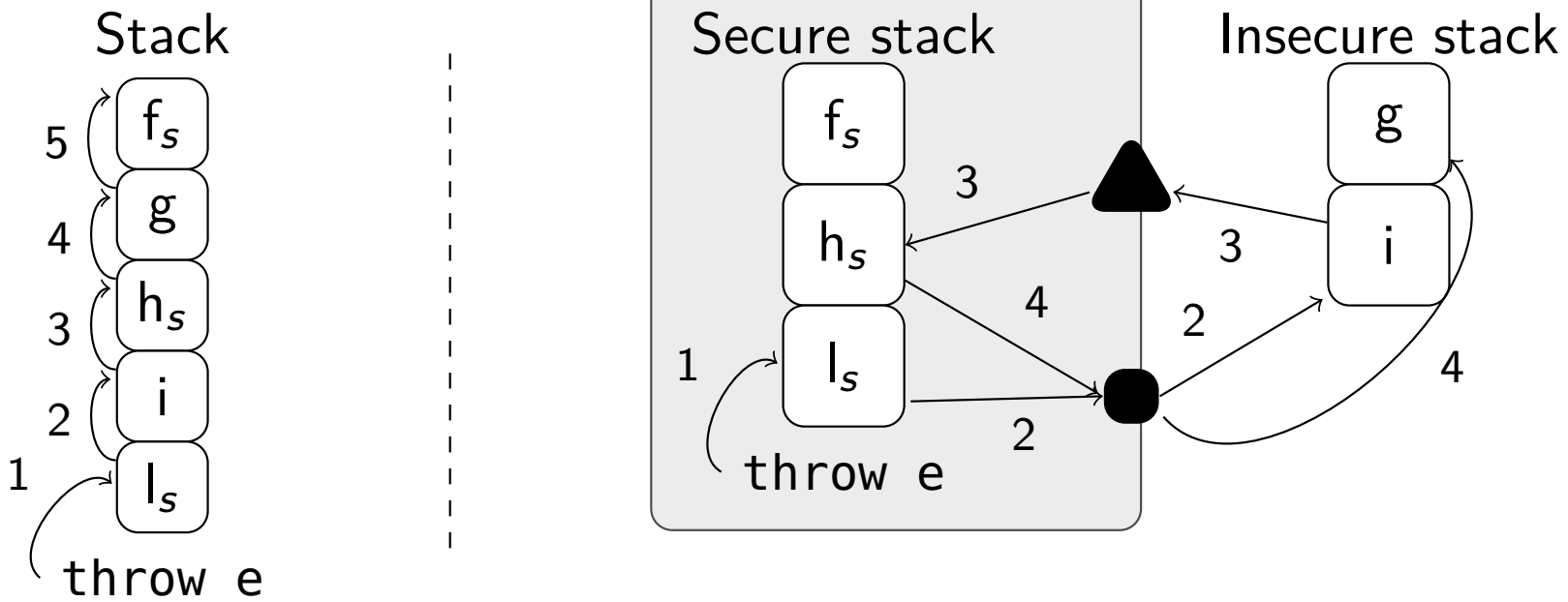
Record passed exceptions

Exceptions



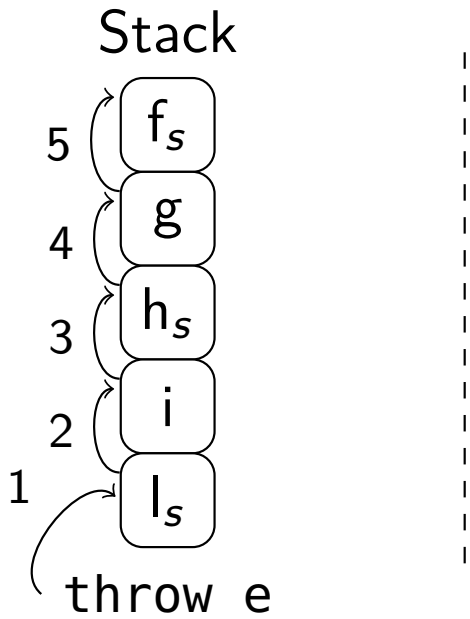
Record passed exceptions
Check that exception could be thrown

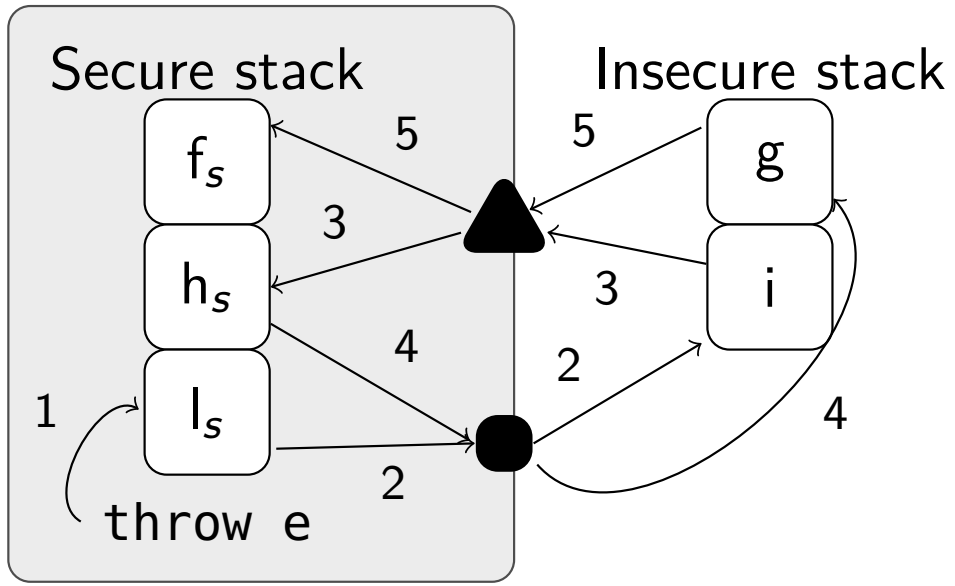
Exceptions



Record passed exceptions
Check that exception could be thrown

Exceptions





Record passed exceptions
Check that exception could be thrown

Cross-package Inheritance

Source level

O1 : JointAccount

O1.counter

O1.limit

```
1  package PSUP;
2    class Account {
3      public getBalance():Int {
4        ...
5      }
6      private counter : Int;
7    }
8
9  package PSUB;
10   class JuniorAccount extends
11       PSUP.Account {
12     public getBalance() : Int {
13       return super.getBalance();
14     }
15     private limit : Int;
16   }
```

- PSUP protected PSUB no

Cross-package Inheritance

Source level

O1 : JointAccount

O1.counter

O1.limit

Target level

$O1^{\downarrow}$

$O1^{\downarrow}$.counter

$O1^{\downarrow}$.limit

```
1  package PSUP;
2    class Account {
3      public getBalance():Int {
4        ...
5      }
6      private counter : Int;
7    }
8
9  package PSUB;
10   class JuniorAccount extends
11       PSUP.Account {
12     public getBalance() : Int {
13       return super.getBalance();
14     }
15     private limit : Int;
16   }
```

- PSUP protected PSUB no

Cross-package Inheritance

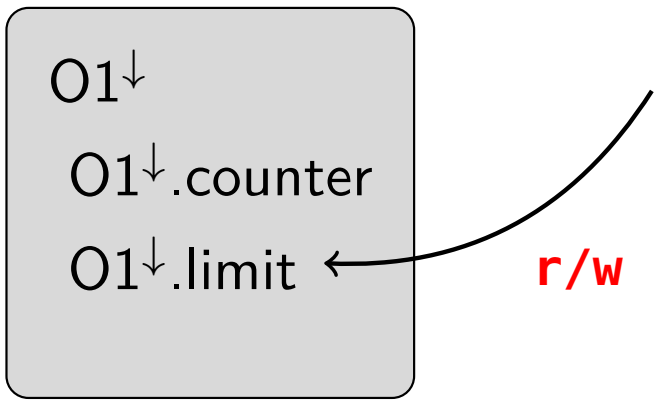
Source level

O1 : JointAccount

O1.counter

O1.limit

Target level




```
1  package PSUP;
2    class Account {
3      public getBalance():Int {
4        ...
5      }
6      private counter : Int;
7    }
8
9  package PSUB;
10   class JuniorAccount extends
11       PSUP.Account {
12     public getBalance() : Int {
13       return super.getBalance();
14     }
15     private limit : Int;
16   }
```

- PSUP protected PSUB no

Cross-package Inheritance

Source level

O1 : JointAccount

O1.counter

O1.limit

- - - - - - - - - - - - - - - - - - - -

Target level

$O1^{\downarrow}$

$O1^{\downarrow}$.counter

$O1^{\downarrow}$.limit

```
1  package PSUP;
2    class Account {
3      public getBalance():Int {
4        ...
5      }
6      private counter : Int;
7    }
8
9  package PSUB;
10   class JuniorAccount extends
11       PSUP.Account {
12     public getBalance() : Int {
13       return super.getBalance();
14     }
15     private limit : Int;
16   }
```

- PSUP protected PSUB no

Cross-package Inheritance

Source level

O1 : JointAccount

O1.counter

O1.limit

- - - - - - - - - - - - - - - - - - - -

Target level

$O1^{\downarrow}$

$O1^{\downarrow}$.counter

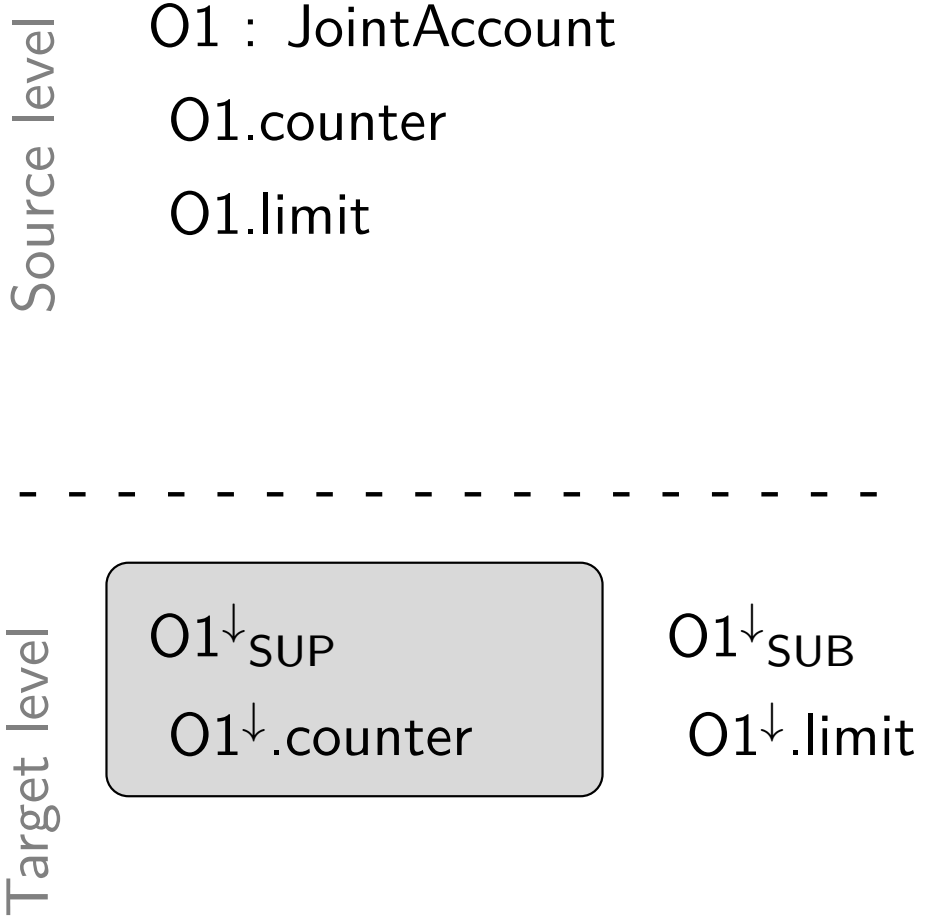
r/w

$O1^{\downarrow}$.limit

```
1  package PSUP;
2    class Account {
3      public getBalance():Int {
4        ...
5      }
6      private counter : Int;
7    }
8
9  package PSUB;
10   class JuniorAccount extends
11       PSUP.Account {
12     public getBalance() : Int {
13       return super.getBalance();
14     }
15     private limit : Int;
16   }
```

- PSUP protected PSUB no

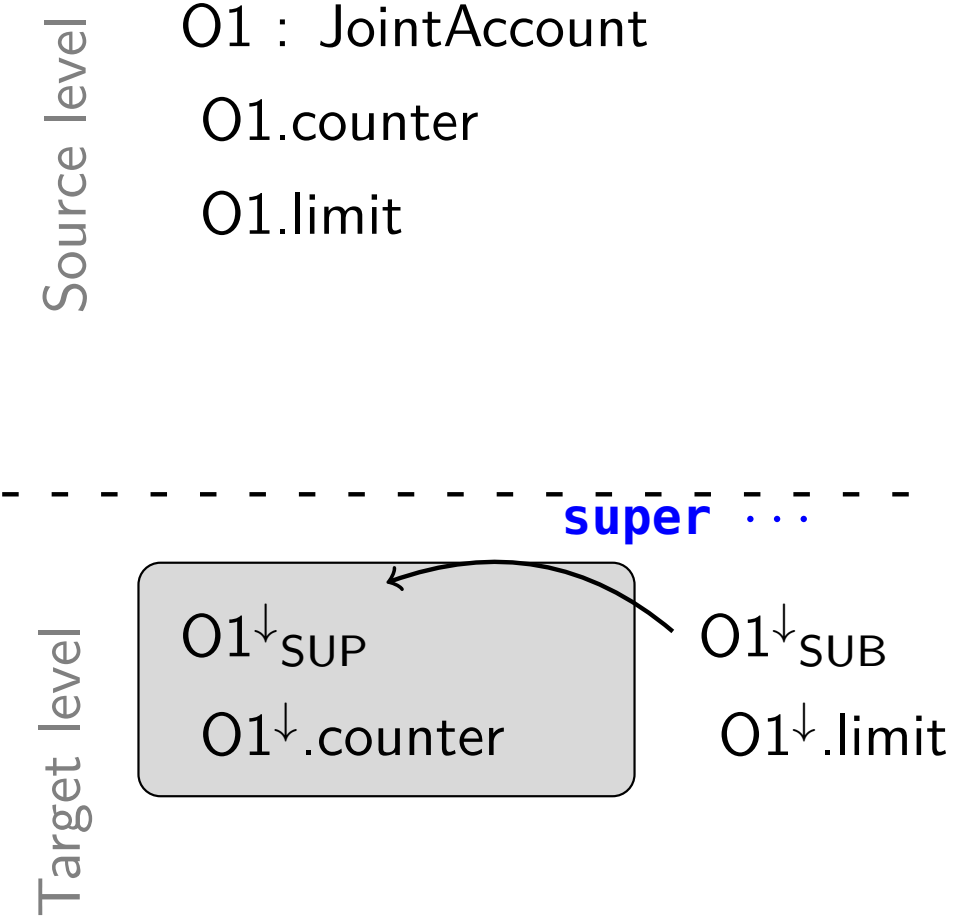
Cross-package Inheritance




```
1  package PSUP;
2    class Account {
3      public getBalance():Int {
4        ...
5      }
6      private counter : Int;
7    }
8
9  package PSUB;
10   class JuniorAccount extends
11       PSUP.Account {
12     public getBalance() : Int {
13       return super.getBalance();
14     }
15     private limit : Int;
16   }
```

- PSUP protected PSUB no

Cross-package Inheritance




```
1  package PSUP;
2    class Account {
3      public getBalance():Int {
4        ...
5      }
6      private counter : Int;
7    }
8
9  package PSUB;
10   class JuniorAccount extends
11       PSUP.Account {
12     public getBalance() : Int {
13       return super.getBalance();
14     }
15     private limit : Int;
16   }
```

- PSUP protected PSUB no

Conclusion

- PMA allows the creation of secure (fully abstract) compilers

Conclusion

- PMA allows the creation of secure (fully abstract) compilers
- identified naïve mistakes

Conclusion

- PMA allows the creation of secure (fully abstract) compilers
- identified naïve mistakes
- proposed sound solution

Questions

?

Bibliographical References I


Karim Eldefrawy, Aurélien Francillon, Daniele Perito, and Gene Tsudik, *SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust*, NDSS 2012, 19th Annual Network and Distributed System Security Symposium (San Diego, United States), 2012.


Alan Jeffrey and Julian Rathke, *Java Jr.: fully abstract trace semantics for a core Java language*, ESOP'05, LNCS, vol. 3444, Springer, 2005, pp. 423–438.


Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar, *Innovative instructions and software model for isolated execution*, HASP '13, ACM, 2013, pp. 10:1–10:1.


Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor, and Adrian Perrig, *Trustvisor: Efficient TCB reduction and attestation*, SP '10 (Washington, DC, USA), IEEE, 2010, pp. 143–158.


Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki, *Flicker: an execution infrastructure for TCB minimization*, SIGOPS Oper. Syst. Rev. **42** (2008), no. 4, 315–328.


Job Noorman, Pieter Agten, Wilfried Daniels, Raoul Strackx, Anthony Van Herrewege, Christophe Huygens, Bart Preneel, Ingrid Verbauwhede, and Frank Piessens, *Sancus: Low-cost trustworthy extensible networked devices with a zero-software Trusted Computing Base*, Proceedings of the 22nd USENIX conference on Security symposium, USENIX Association, 2013.


Raoul Strackx and Frank Piessens, *Fides: selectively hardening software application components against kernel-level or process-level malware*, Proceedings of the 2012 ACM conference on Computer and communications security (New York, NY, USA), CCS '12, ACM, 2012, pp. 2–13.