Lecture 5: The PMA Case Study

CS350

Marco Patrignani

Lecture Goals

- present existing FA compilers that rely on a specific security architecture
- see how to formalise advanced security architecture notions
- see an advanced trace-based backtranslation
- reason about advanced FAC proofs

- Patrignani, Devriese, Piessens: On Modular and Fully-Abstract Compilation. In CSF'16
- Patrignani, Agten, Strackx, Jacobs, Clarke, Piessens: Secure Compilation to Protected Module Architectures. In TOPLAS'15

PMA: High Level (SGX-like)

- enclave: isolated memory region (coarse)
- enclaves are split in code and data
- jump to enclaves through entry points

From ↘ To	Protected			Unprotected
	Entry Point	Code	Data	
Protected	r x	r x	r w	r w x
Unprotected	x			r w x

PMA: Formally

to the board

- protection domains
- structuring ACP
- register allocation and flags

JavaJr: High Level

- Java-like (oo, no reflection, strongly typed, exceptions)
- deep encapsulation: only private fields
- public & private methods
- no inner classes, no cross-package inheritance

- Alan Jeffrey and Julian Rathke. Java Jr.: fully abstract trace semantics for a core Java language. In ESOP'05

JavaJr: Formally

to the board

- exceptions

Assumption 1: Correct Compilation

- $(\!|\cdot|\!) : \mathsf{P} \to \mathbf{P}$ (and for s, e, v, ...)
- $(\!|\cdot|\!)$ is correct

$$\forall \mathsf{s}, (\!|\mathsf{s}\gamma|\!) \hookrightarrow^* (\!|\mathsf{v}|\!) \Rightarrow \mathsf{s}\gamma \hookrightarrow^* \mathsf{v}$$

Desirable Security Properties

- confidentiality and integrity of field contents, of object names and of method bodies
- no control flow alteration apart from when using exceptions
- non reachability of stuck (error) program states

Desirable Security Properties

- confidentiality and integrity of field contents, of object names and of method bodies
- no control flow alteration apart from when using exceptions
- non reachability of stuck (error) program states

All of them can be encoded through program equivalence

Compiler Vulnerabilities

1. stack access
2. information leakage through registers and flags
3. boolean values
4. current object type
5. argument type
6. guessable references
7. excessive exceptions catching

Vulnerability #1

```
1  package p;
2  class CL {
3    private sec : Int = 0;
4
5    public doCallback( cb :
         External ) : Int {
6      var x : Int = sec;
7      cb.callback();
8      return 0;
9    }
10 }
11 object oL : CL
```

```
1  package p;
2  class CR {
3    private sec : Int = 1;
4
5    public doCallback( cb :
         External ) : Int {
6      var x : Int = sec;
7      cb.callback();
8      return 0;
9    }
10 }
11 object oR : CR
```

Vulnerability #1

```
1  package p;
2  class CL {
3    private sec : Int = 0;
4
5    public doCallback( cb :
         External ) : Int {
6      var x : Int = sec;
7      cb.callback();
8      return 0;
9    }
10 }
11 object oL : CL
```

```
1  package p;
2  class CR {
3    private sec : Int = 1;
4
5    public doCallback( cb :
         External ) : Int {
6      var x : Int = sec;
7      cb.callback();
8      return 0;
9    }
10 }
11 object oR : CR
```

- location of x matters

Vulnerability #1

```
1  package p;
2  class CL {
3    private sec : Int = 0;
4
5    public doCallback( cb :
         External ) : Int {
6      var x : Int = sec;
7      cb.callback();
8      return 0;
9    }
10 }
11 object oL : CL
```

```
1  package p;
2  class CR {
3    private sec : Int = 1;
4
5    public doCallback( cb :
         External ) : Int {
6      var x : Int = sec;
7      cb.callback();
8      return 0;
9    }
10 }
11 object oR : CR
```

- location of x matters
- Sol: need a protected stack

Vulnerability #2

```
1  package p;
2  class CL {
3    public testVariable() :
         Int {
4      var x : Int = 0;
5      if ( x == 0 ) {
6        return 0;
7      } else {
8        return 0;
9      }
10   }
11 }
12 object oL : CL
```

```
1  package p;
2  class CR {
3    public testVariable()
         : Int {
4      var x : Int = 1;
5      if ( x == 0 ) {
6        return 0;
7      } else {
8        return 0;
9      }
10   }
11 }
12 object oR : CR
```

Vulnerability #2

```
1  package p;
2  class CL {
3    public testVariable() :
         Int {
4      var x : Int = 0;
5      if ( x == 0 ) {
6        return 0;
7      } else {
8        return 0;
9      }
10   }
11 }
12 object oL : CL
```

```
1  package p;
2  class CR {
3    public testVariable()
         : Int {
4      var x : Int = 1;
5      if ( x == 0 ) {
6        return 0;
7      } else {
8        return 0;
9      }
10   }
11 }
12 object oR : CR
```

- flags and register leak information

Vulnerability #2

```
1  package p;
2  class CL {
3    public testVariable() :
         Int {
4      var x : Int = 0;
5      if ( x == 0 ) {
6        return 0;
7      } else {
8        return 0;
9      }
10   }
11 }
12 object oL : CL
```

```
1  package p;
2  class CR {
3    public testVariable()
         : Int {
4      var x : Int = 1;
5      if ( x == 0 ) {
6        return 0;
7      } else {
8        return 0;
9      }
10   }
11 }
12 object oR : CR
```

- flags and register leak information
- Sol: reset flags and unused registers

Vulnerability #3

```
1  package p;
2  class CL {
3    public identBool( x :
         Bool ) : Bool {
4      if( x == true ){
5        return true;
6      }
7      return false;
8    }
9  }
10 object oL : CL
```

```
1  package p;
2  class CR {
3    public identBool( x :
         Bool ) : Bool {
4
5      return x;
6
7
8    }
9  }
10 object oR : CR
```

Vulnerability #3

```
1  package p;
2  class CL {
3    public identBool( x :
         Bool ) : Bool {
4      if( x == true ){
5        return true;
6      }
7      return false;
8    }
9  }
10 object oL : CL
```

```
1  package p;
2  class CR {
3    public identBool( x :
         Bool ) : Bool {
4
5      return x;
6
7
8    }
9  }
10 object oR : CR
```

- ground values have a fixed value

Vulnerability #3

```
1  package p;
2  class CL {
3    public identBool( x :
         Bool ) : Bool {
4      if( x == true ){
5        return true;
6      }
7      return false;
8    }
9  }
10 object oL : CL
```

```
1  package p;
2  class CR {
3    public identBool( x :
         Bool ) : Bool {
4
5      return x;
6
7
8    }
9  }
10 object oR : CR
```

- ground values have a fixed **value**
- Sol: dynamic typecheck

Vulnerability #4

```
1  package p;
2  class PairL {
3    private fst, snd : Obj =
         null;
4    public getFirst(): Obj {
5      return this.fst;
6    }
7  }
8  class SecretL {
9    private sec : Int = 0;
10 }
11 object oL : SecretL
```

```
1  package p;
2  class PairR {
3    private fst, snd : Obj =
         null;
4    public getFirst(): Obj {
5      return this.fst;
6    }
7  }
8  class SecretR {
9    private sec : Int = 1;
10 }
11 object oR : SecretR
```

Vulnerability #4

```
1  package p;
2  class PairL {
3    private fst, snd : Obj =
         null;
4    public getFirst(): Obj {
5      return this.fst;
6    }
7  }
8  class SecretL {
9    private sec : Int = 0;
10 }
11 object oL : SecretL
```

```
1  package p;
2  class PairR {
3    private fst, snd : Obj =
         null;
4    public getFirst(): Obj {
5      return this.fst;
6    }
7  }
8  class SecretR {
9    private sec : Int = 1;
10 }
11 object oR : SecretR
```

- invoke `getFirst` on `oL`/`oR`

Vulnerability #4

```
1  package p;
2  class PairL {
3    private fst, snd : Obj =
         null;
4    public getFirst(): Obj {
5      return this.fst;
6    }
7  }
8  class SecretL {
9    private sec : Int = 0;
10 }
11 object oL : SecretL
```

```
1  package p;
2  class PairR {
3    private fst, snd : Obj =
         null;
4    public getFirst(): Obj {
5      return this.fst;
6    }
7  }
8  class SecretR {
9    private sec : Int = 1;
10 }
11 object oR : SecretR
```

- invoke `getFirst` on `oL`/`oR`
- Sol: dynamic typecheck the current object

Vulnerability #5

```
1  package p;
2  class ProxyPair {
3    public takeFirst( v :
         Pair ): Obj {
4      return v.getFirst();
5    }
6  }
7  class SecretL {
8    private sec : Int = 0;
9  }
10 object oL : SecretL
```

```
1  package p;
2  class ProxyPair {
3    public takeFirst( v :
         Pair ): Obj {
4      return v.getFirst();
5    }
6  }
7  class SecretR {
8    private sec : Int = 1;
9  }
10 object oR : SecretR
```

Vulnerability #5

```
1  package p;
2  class ProxyPair {
3    public takeFirst( v :
         Pair ): Obj {
4      return v.getFirst();
5    }
6  }
7  class SecretL {
8    private sec : Int = 0;
9  }
10 object oL : SecretL
```

```
1  package p;
2  class ProxyPair {
3    public takeFirst( v :
         Pair ): Obj {
4      return v.getFirst();
5    }
6  }
7  class SecretR {
8    private sec : Int = 1;
9  }
10 object oR : SecretR
```

- invoke `takeFirst` on `oL`/`oR`

Vulnerability #5

```
1  package p;
2  class ProxyPair {
3    public takeFirst( v :
         Pair ): Obj {
4      return v.getFirst();
5    }
6  }
7  class SecretL {
8    private sec : Int = 0;
9  }
10 object oL : SecretL
```

```
1  package p;
2  class ProxyPair {
3    public takeFirst( v :
         Pair ): Obj {
4      return v.getFirst();
5    }
6  }
7  class SecretR {
8    private sec : Int = 1;
9  }
10 object oR : SecretR
```

- invoke `takeFirst` on `oL`/`oR`
- Sol: dynamic typecheck all arguments

Vulnerability #6

```
1  package p;
2  class SecretL {
3    private sec : Int = 0;
4    public createSecret() :
         Secret {
5
6
7      return new Secret();
8    }
9  }
10 object oL1 : SecretL
11 object oL2 : SecretL
```

```
1  package p;
2  class SecretR {
3    private sec : Int = 0;
4    public createSecret() :
         Secret {
5      var x : Secret = new
           Secret();
6      return new Secret();
7    }
8  }
9  object oR1 : SecretR
10 object oR2 : SecretR
```

Vulnerability #6

```
1  package p;
2  class SecretL {
3    private sec : Int = 0;
4    public createSecret() :
       Secret {
5
6
7      return new Secret();
8    }
9  }
10 object oL1 : SecretL
11 object oL2 : SecretL
```

```
1  package p;
2  class SecretR {
3    private sec : Int = 0;
4    public createSecret() :
       Secret {
5      var x : Secret = new
         Secret();
6      return new Secret();
7    }
8  }
9  object oR1 : SecretR
10 object oR2 : SecretR
```

- witness address of returned `Secret`

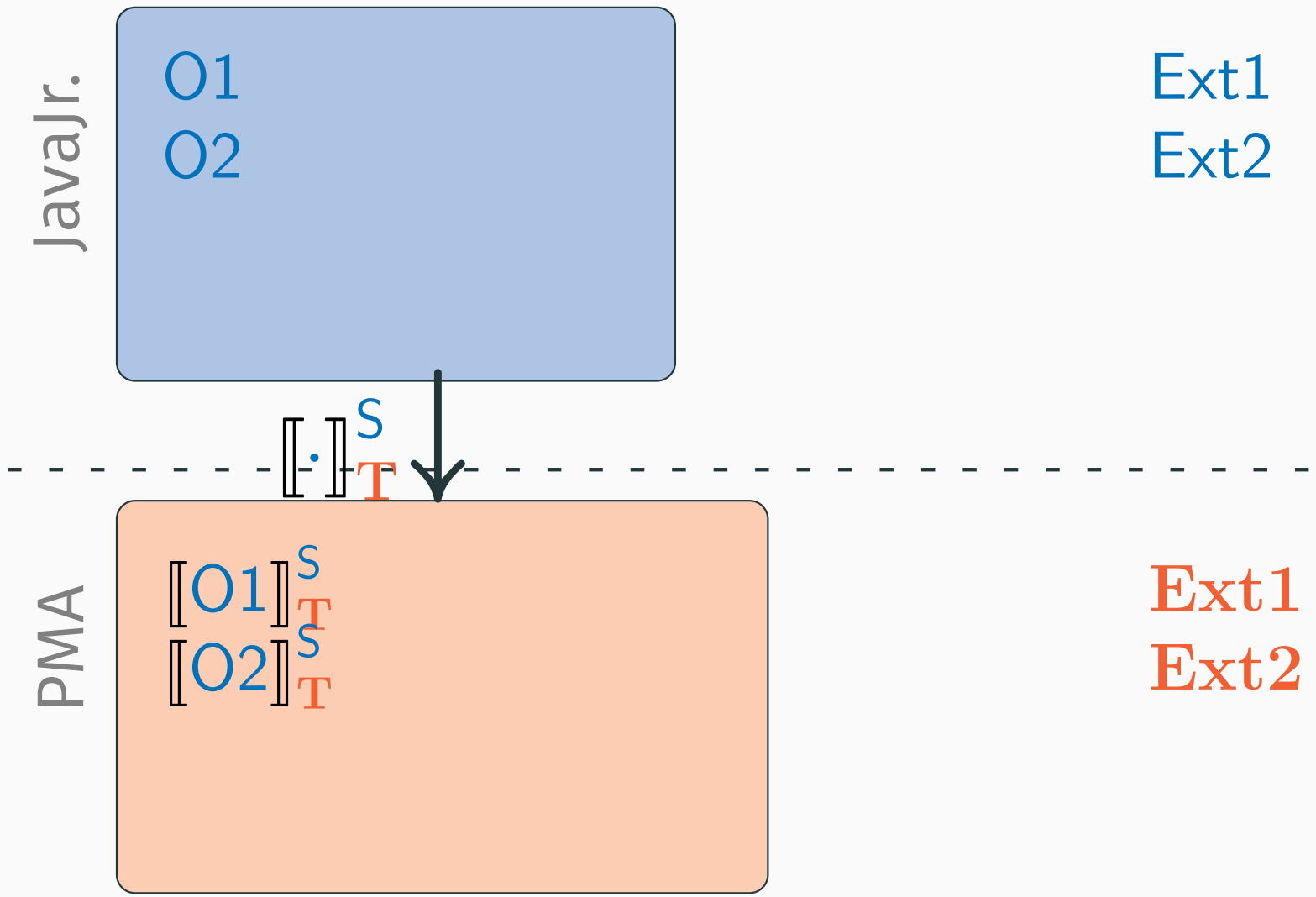
Vulnerability #6

```
1  package p;
2  class SecretL {
3    private sec : Int = 0;
4    public createSecret() :
       Secret {
5
6
7      return new Secret();
8    }
9  }
10 object oL1 : SecretL
11 object oL2 : SecretL
```

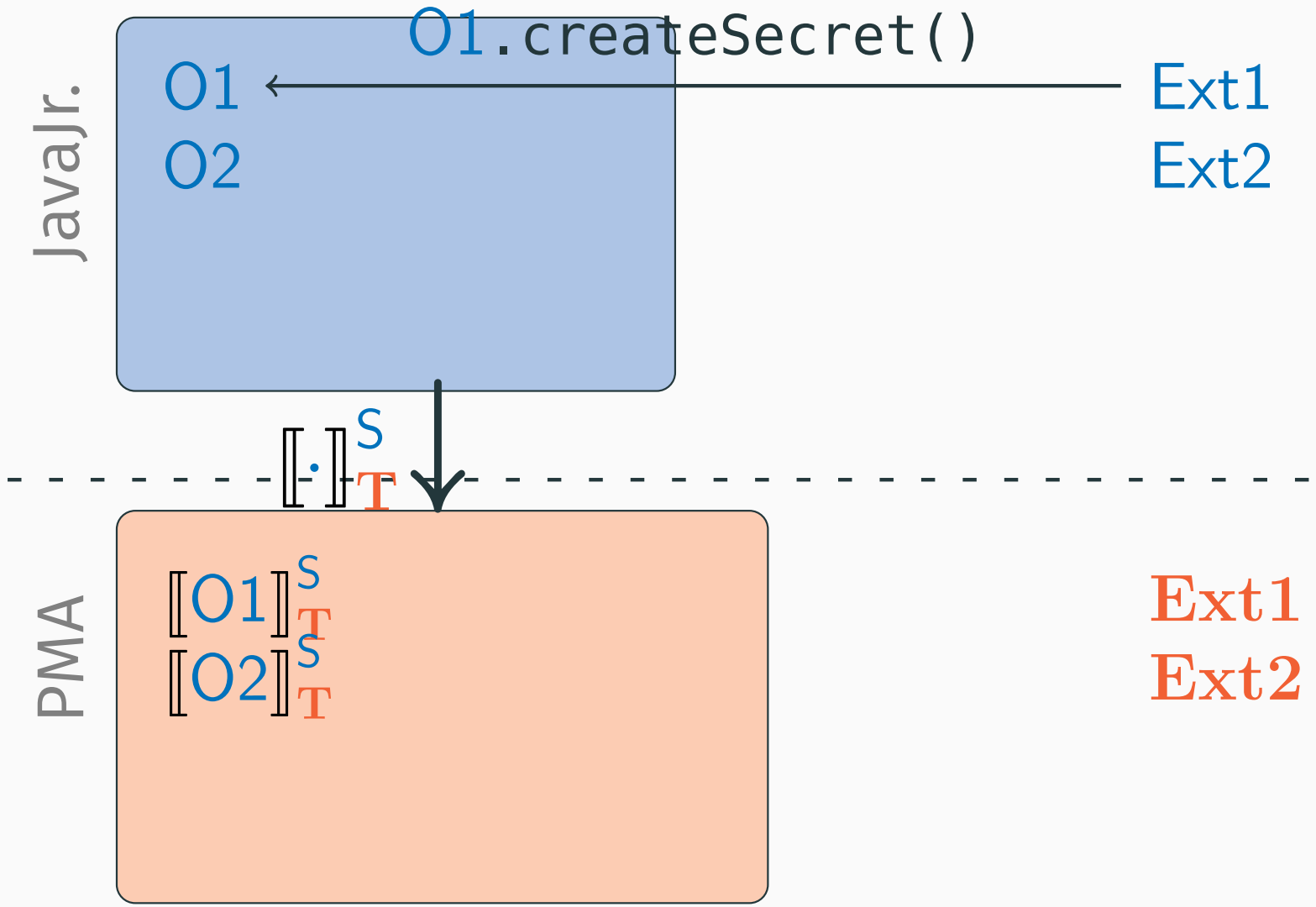
```
1  package p;
2  class SecretR {
3    private sec : Int = 0;
4    public createSecret() :
       Secret {
5      var x : Secret = new
         Secret();
6      return new Secret();
7    }
8  }
9  object oR1 : SecretR
10 object oR2 : SecretR
```

- witness address of returned `Secret`
- Sol: mask objects through proxies

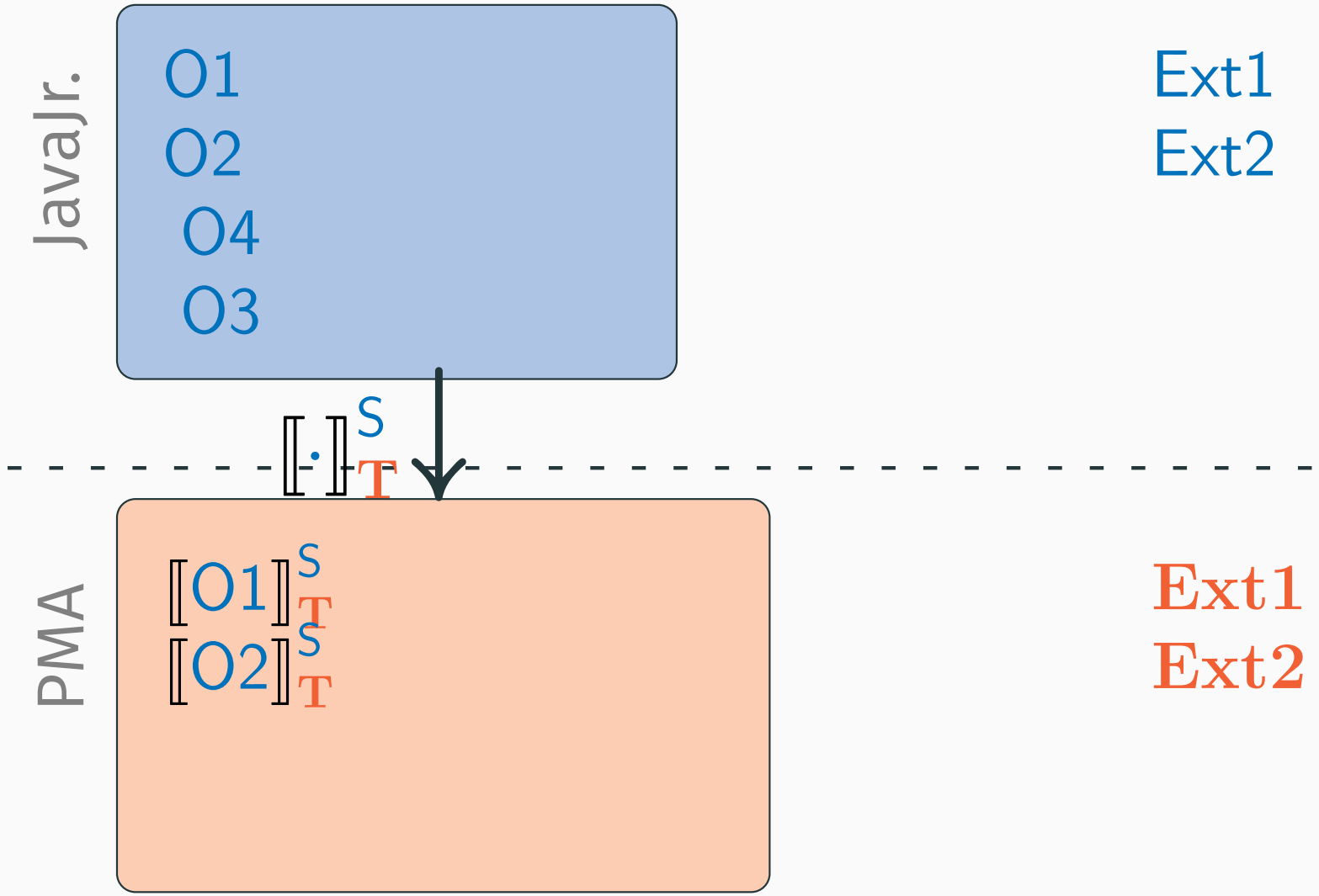
Memory Allocation Issues



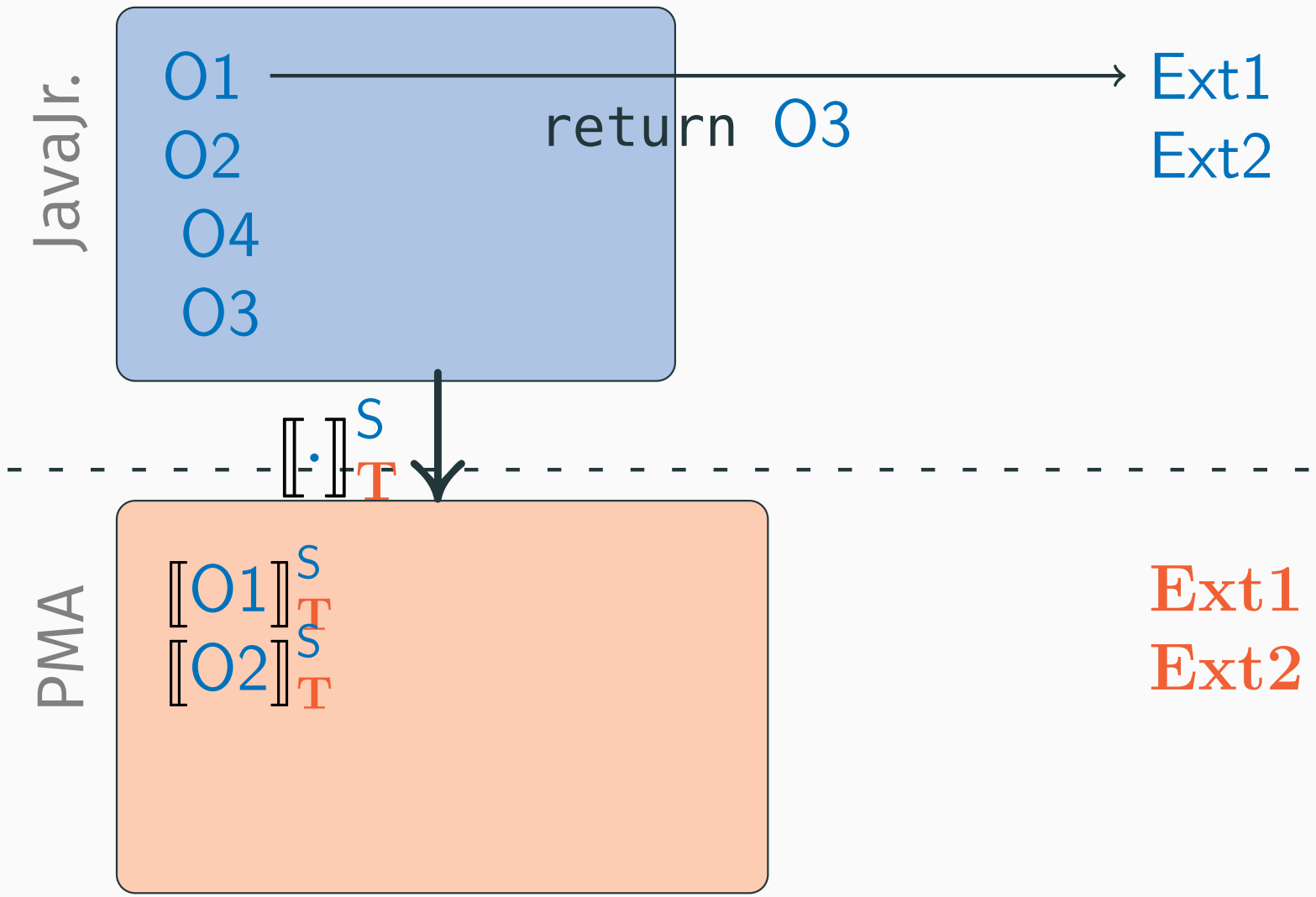
Memory Allocation Issues



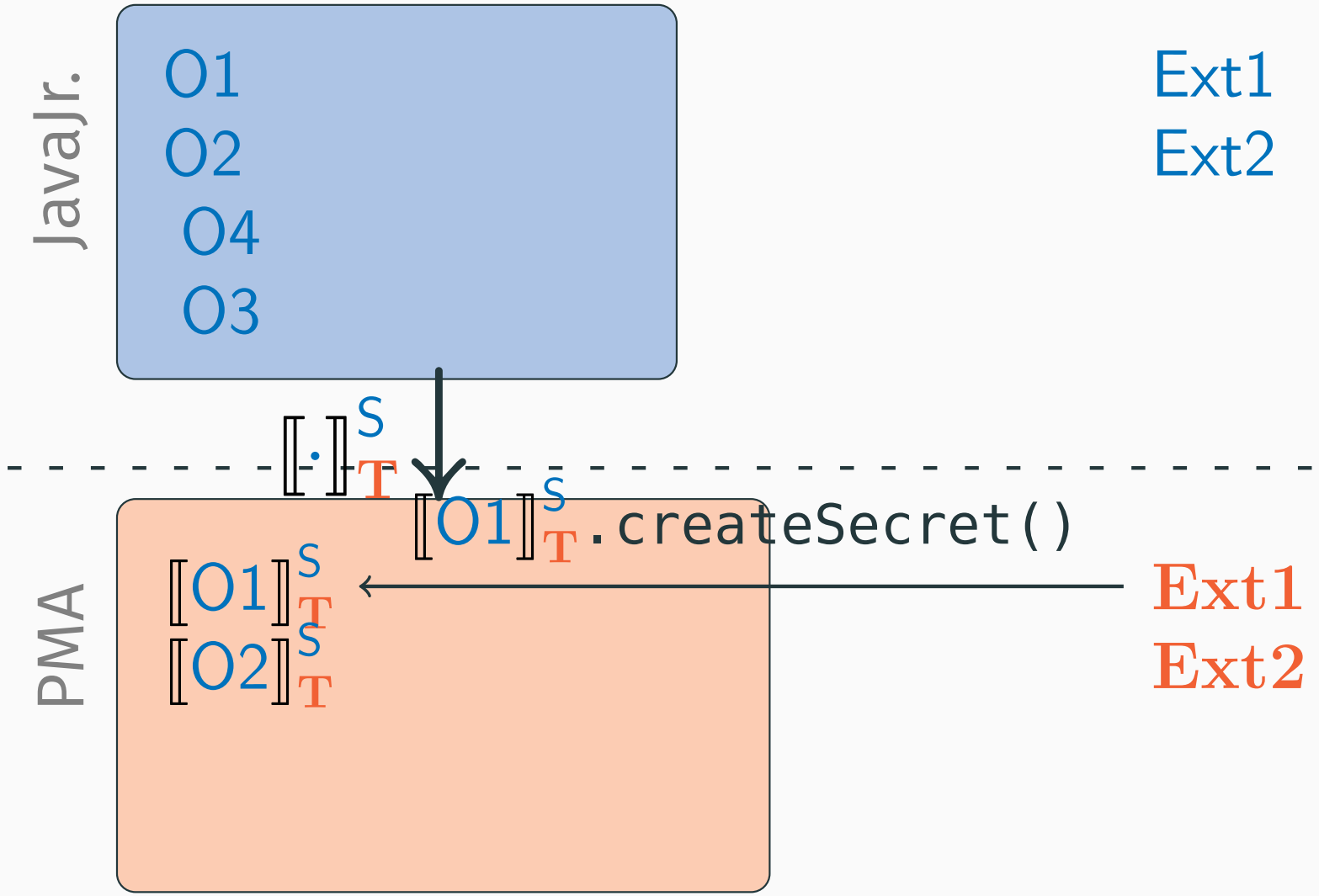
Memory Allocation Issues



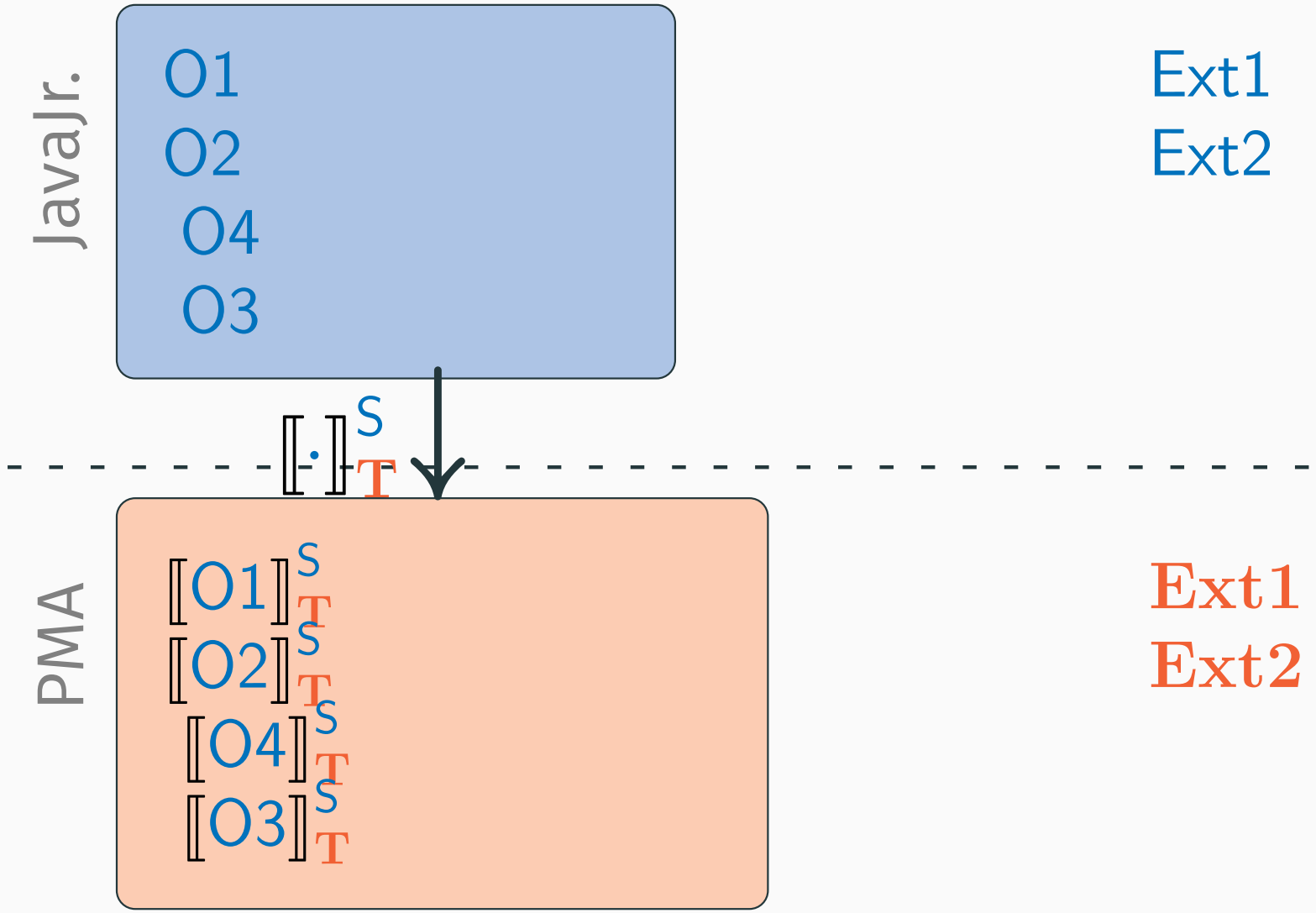
Memory Allocation Issues



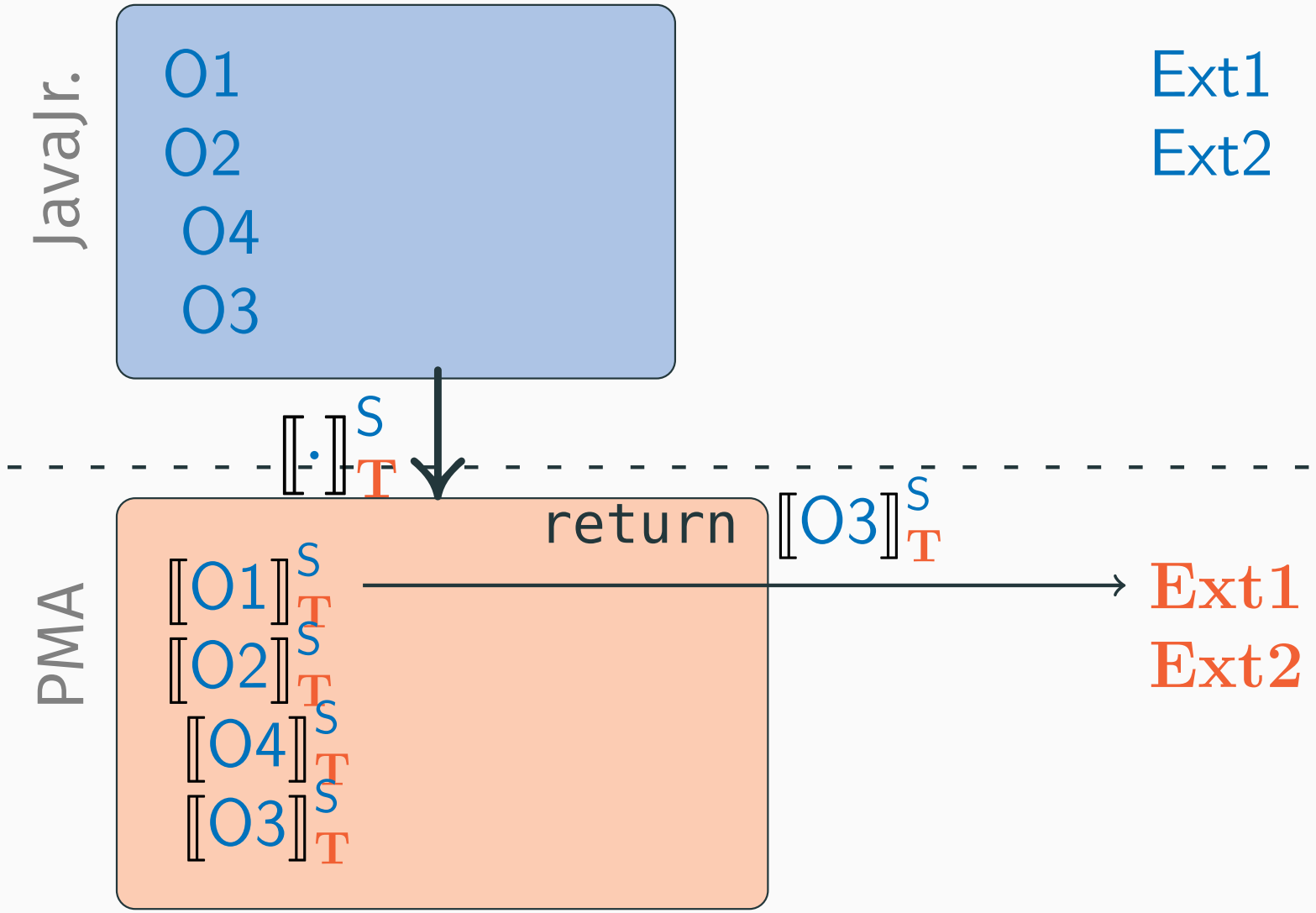
Memory Allocation Issues



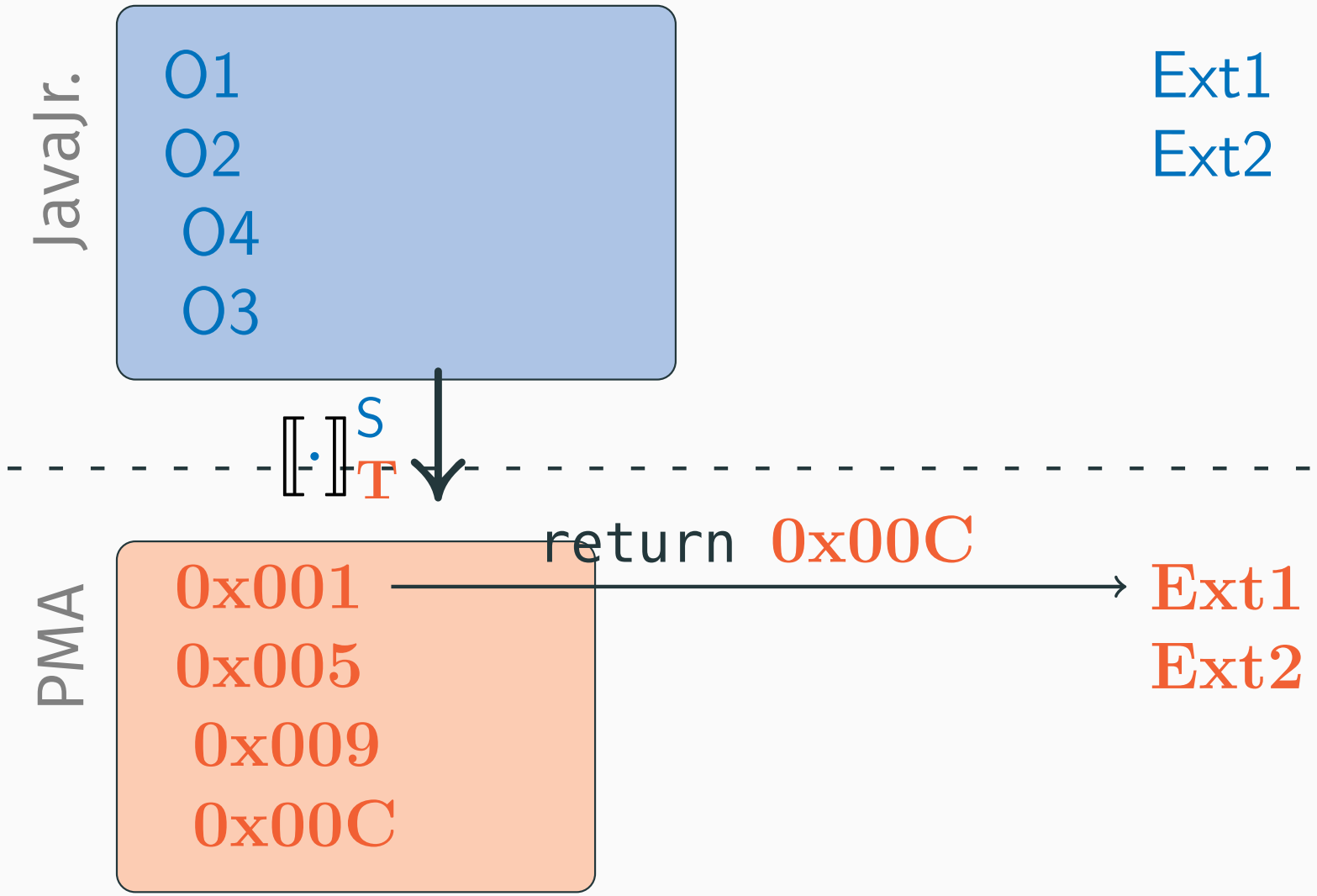
Memory Allocation Issues



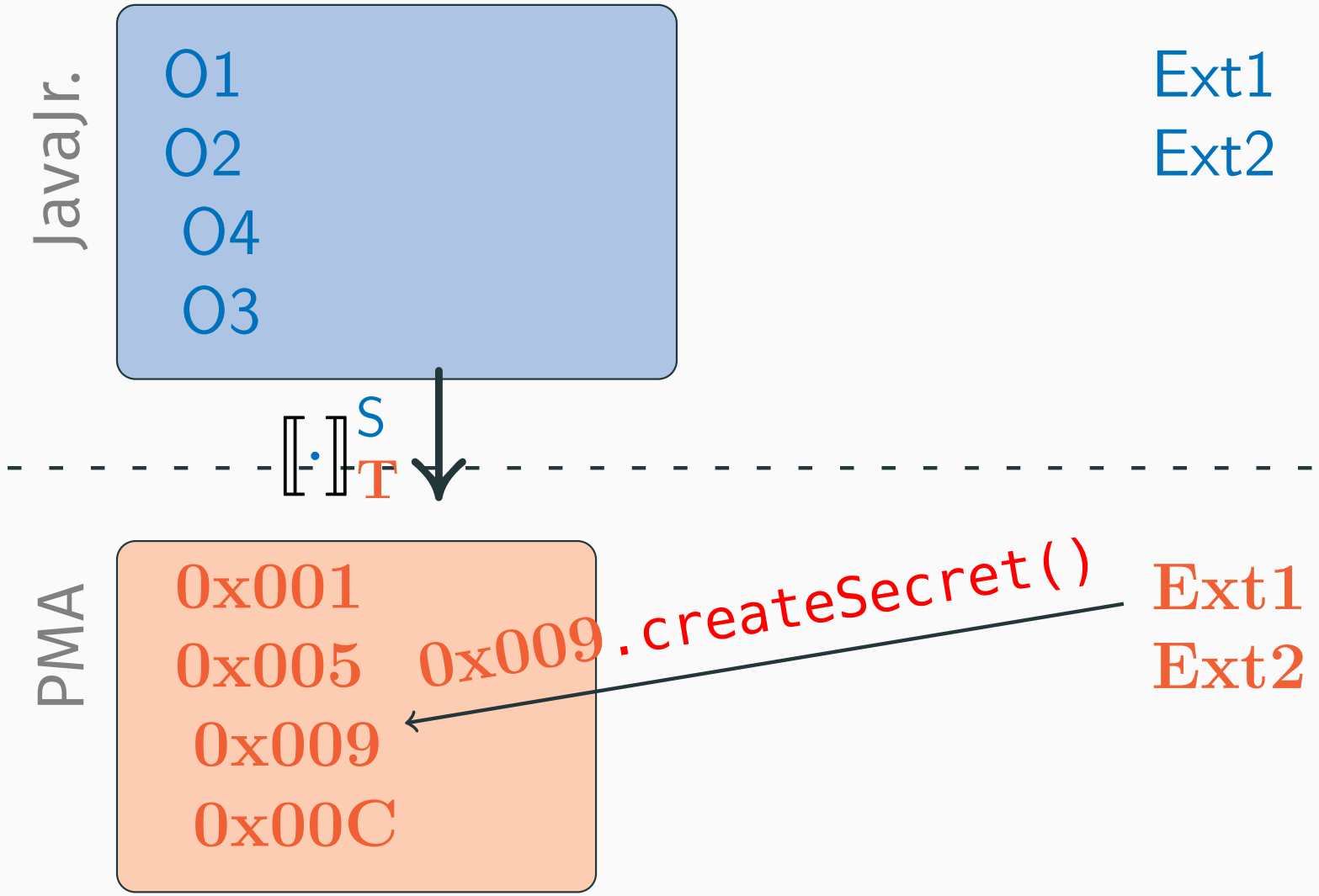
Memory Allocation Issues



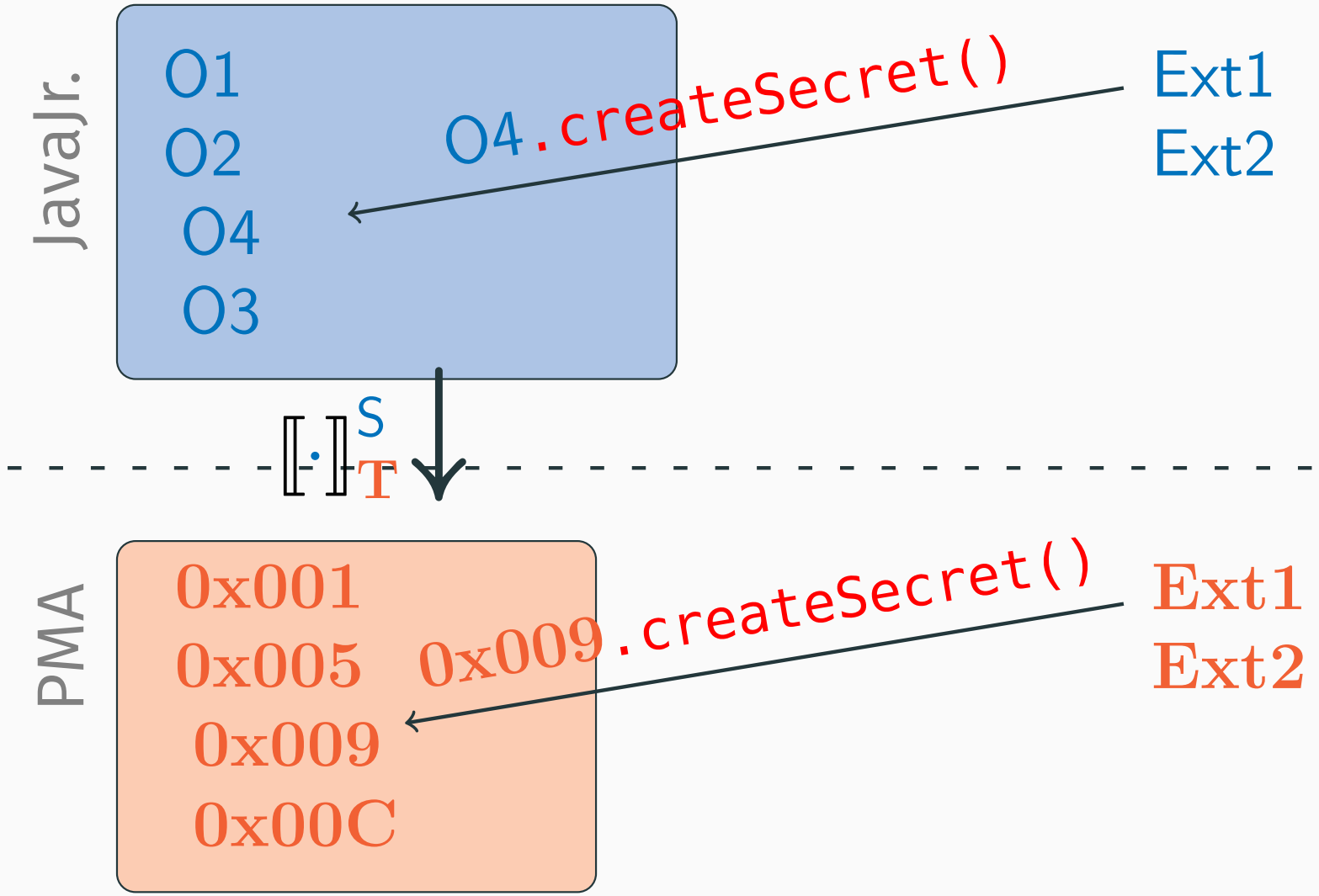
Memory Allocation Issues



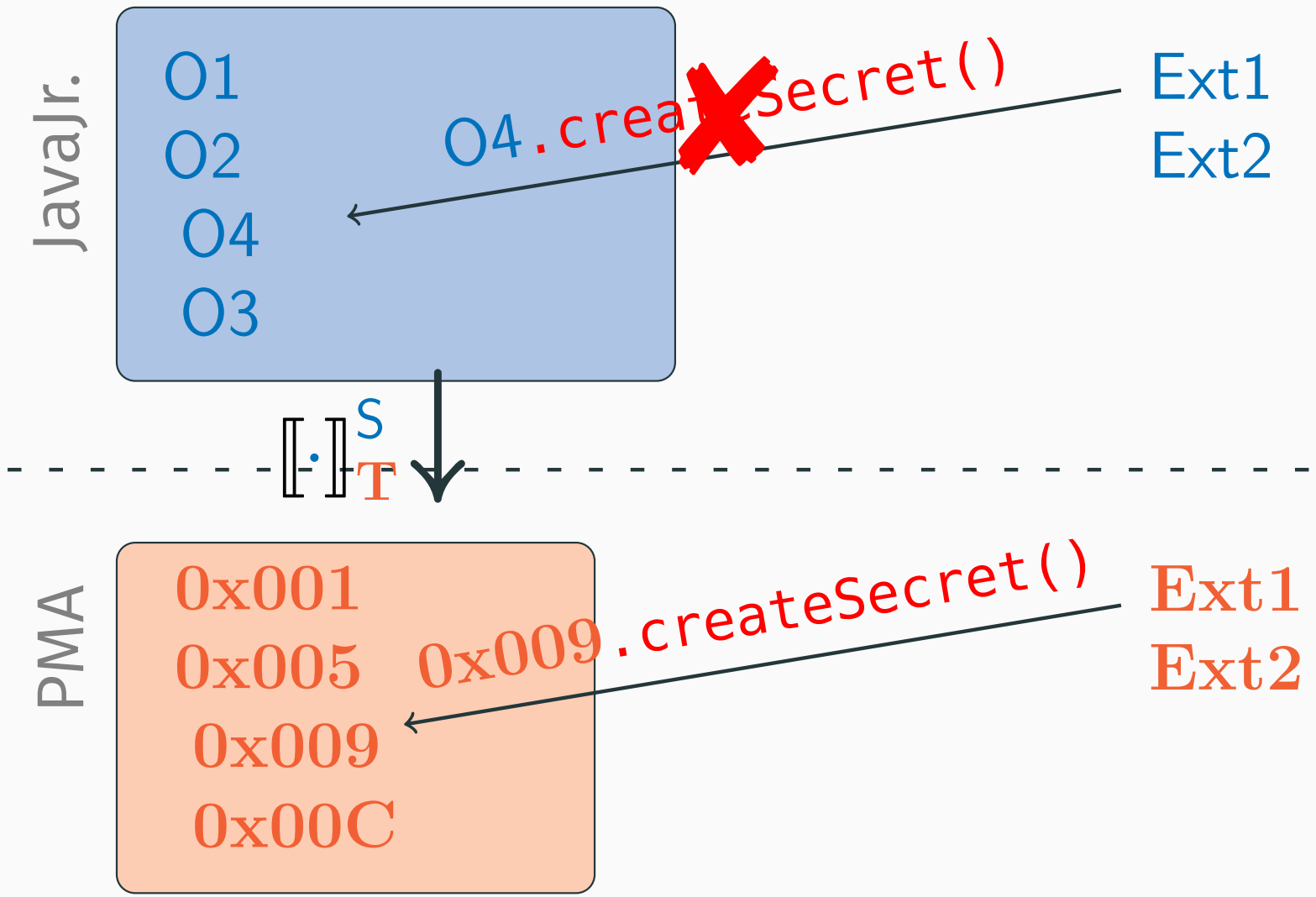
Memory Allocation Issues



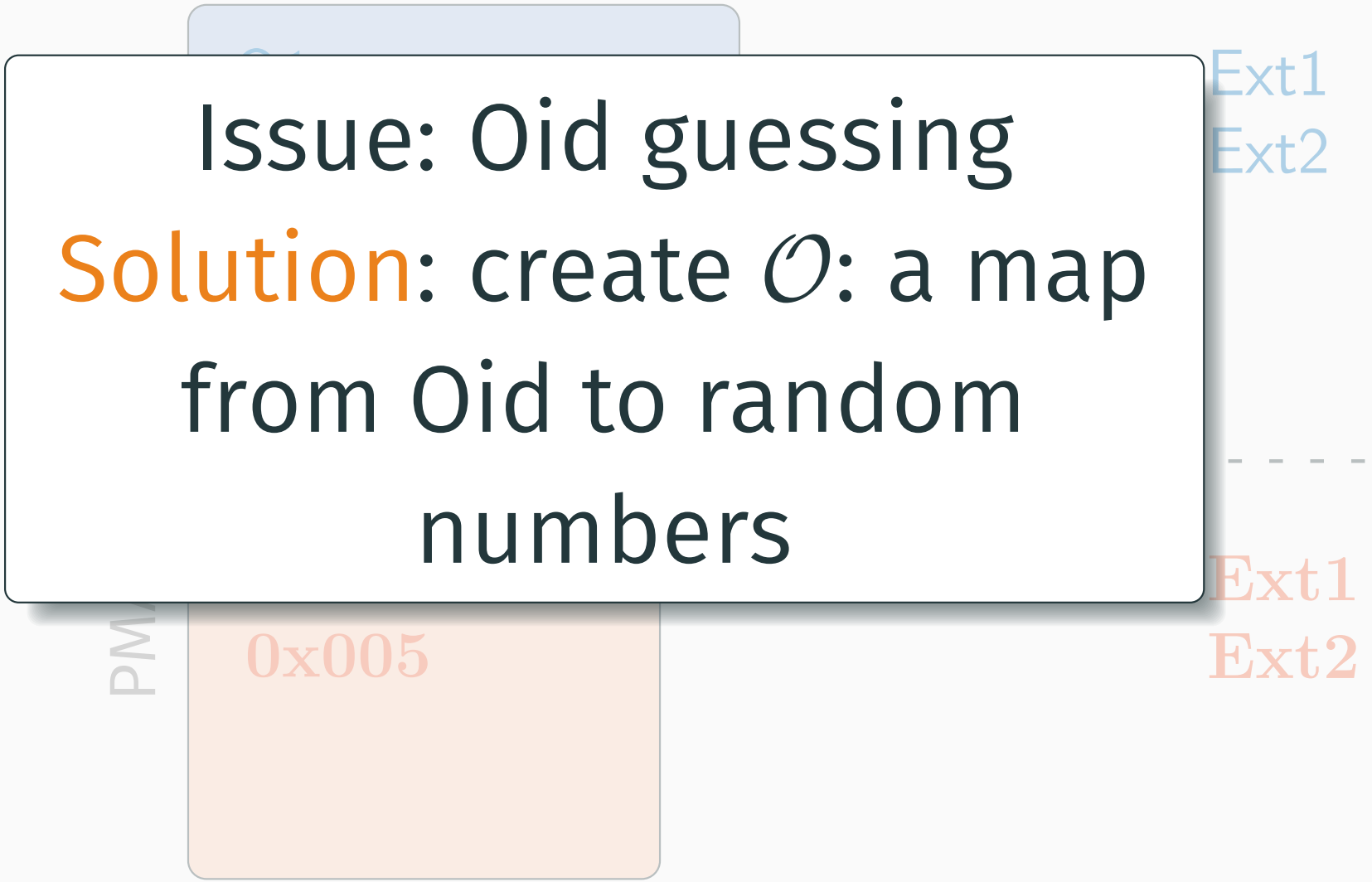
Memory Allocation Issues



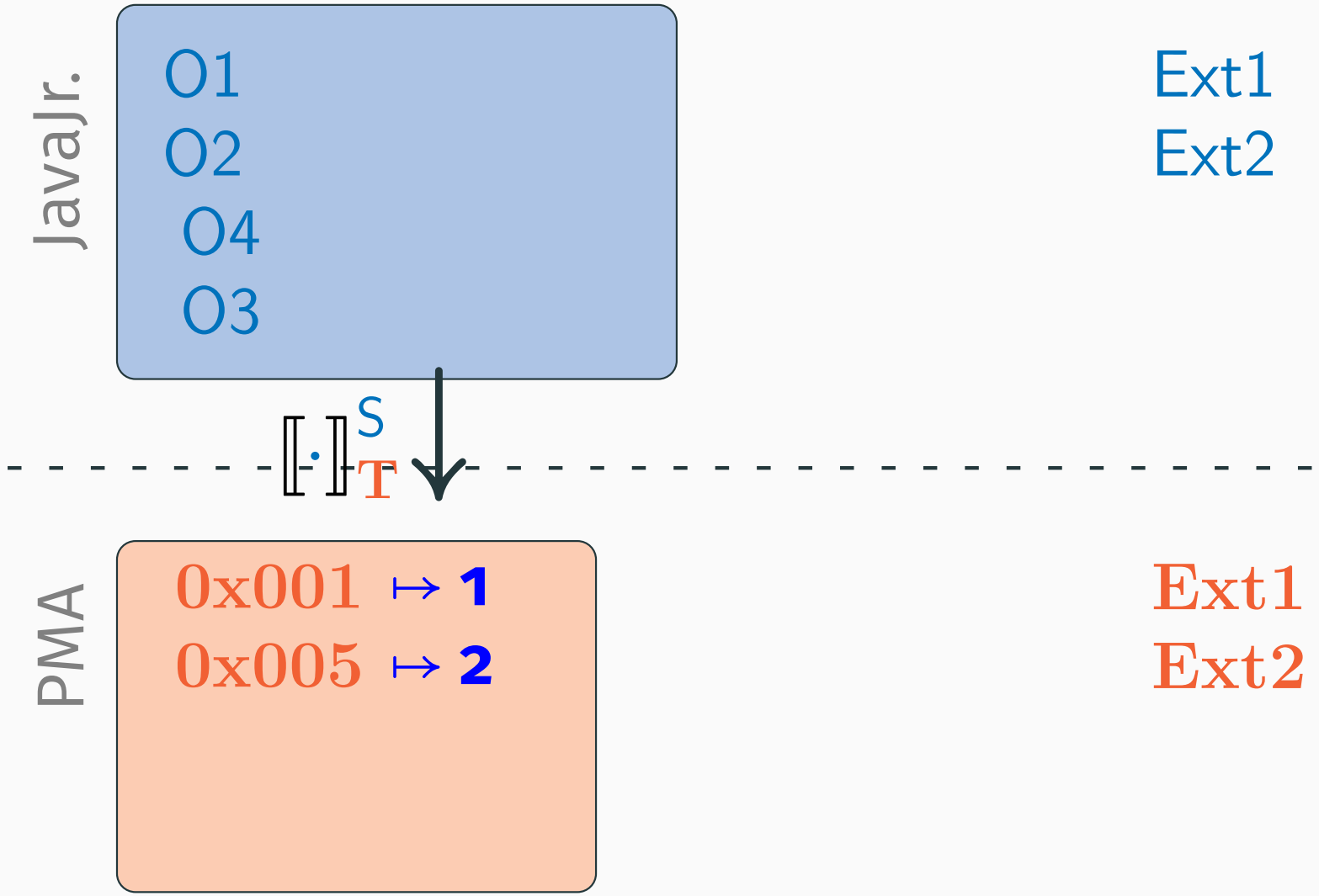
Memory Allocation Issues



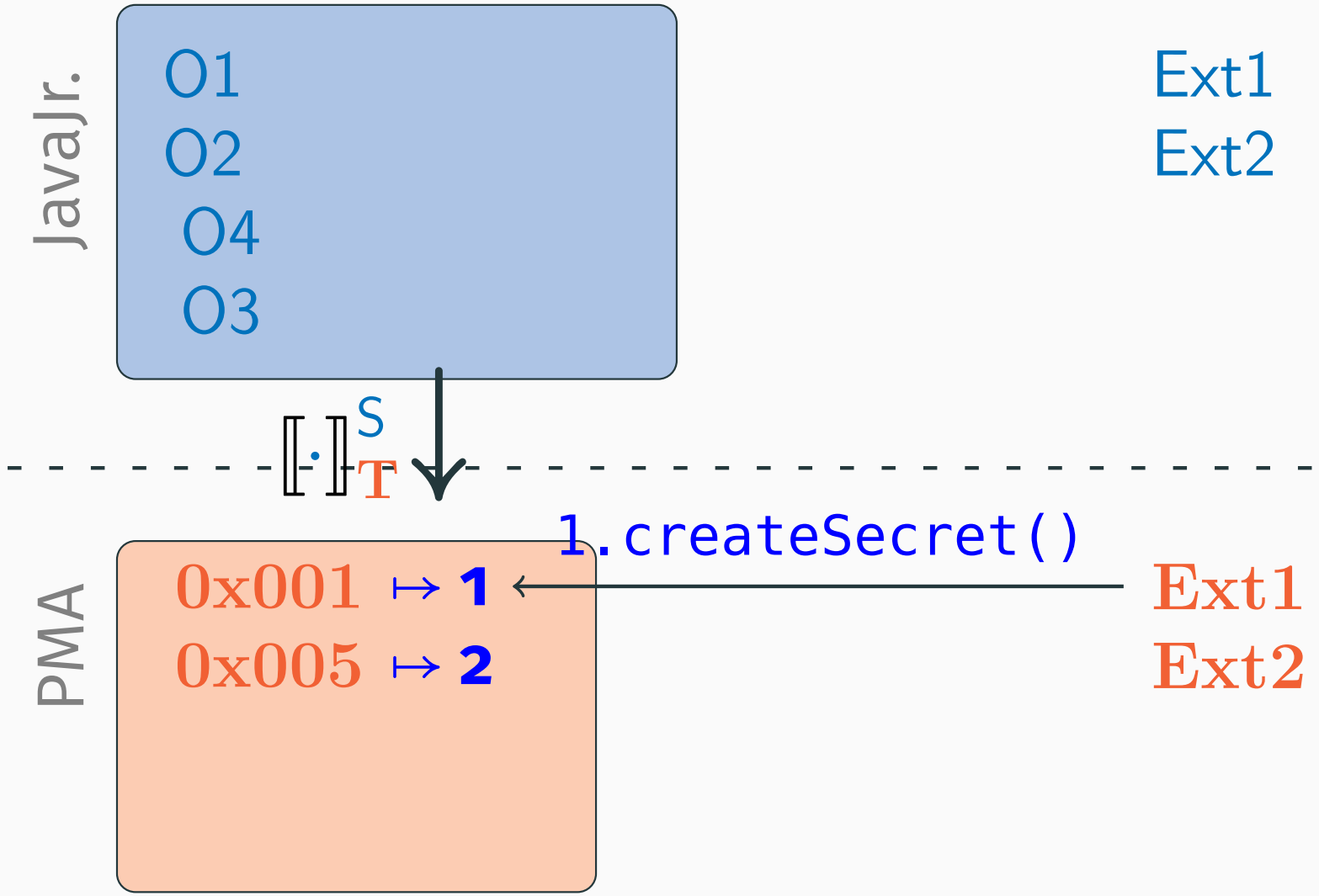
Memory Allocation Issues



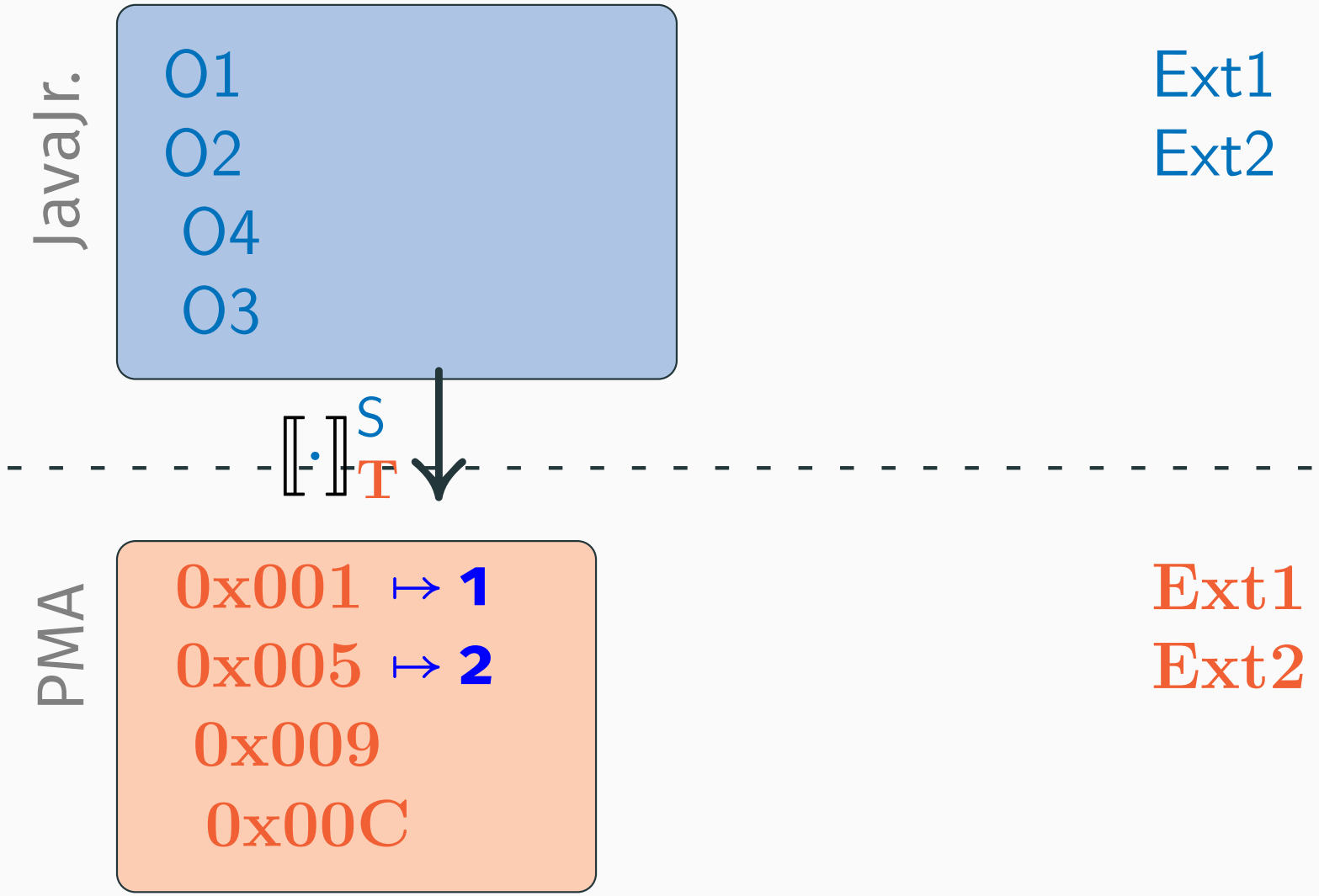
Memory Allocation Issues



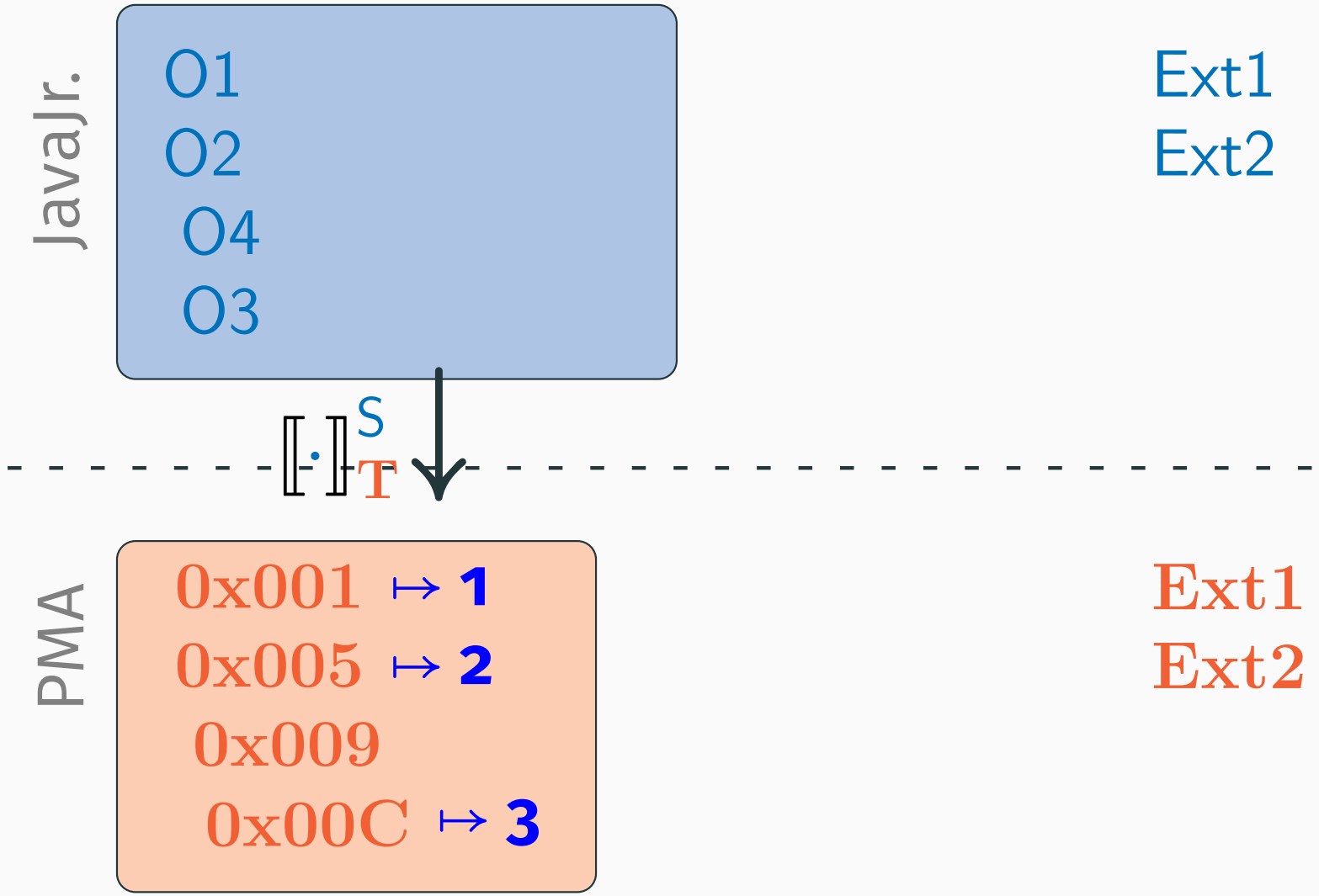
Memory Allocation Issues



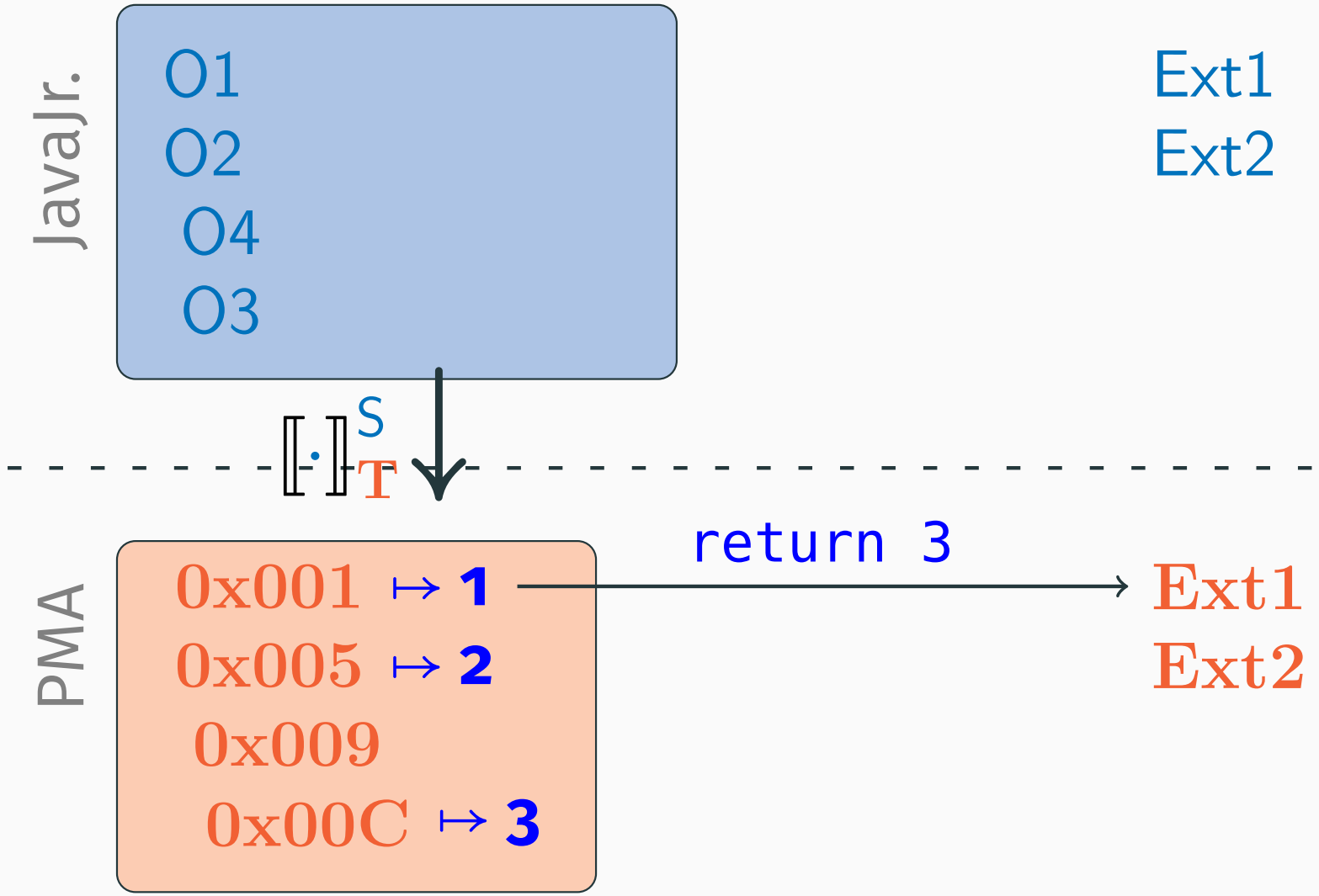
Memory Allocation Issues



Memory Allocation Issues



Memory Allocation Issues



Secure Compiler Structure

$[\![\cdot]\!] : \mathsf{P} \to \mathbf{P}$ must:

- generate correct code (through $(\!|\cdot|\!)$)

Secure Compiler Structure

$[\![\cdot]\!] : \mathrm{P} \to \mathbf{P}$ must:

- generate correct code (through $(\!|\cdot|\!)$)
- place it inside an enclave

Secure Compiler Structure

$[\![\cdot]\!] : \mathsf{P} \to \mathbf{P}$ must:

- generate correct code (through $(\!|\cdot|\!)$)
- place it inside an enclave
- wrap it at entry and exit points with checks

Entry and Exit Points

Method p entry point		Preamble to returnback entry point	
1	Load receiver $v = \mathcal{O}(\mathtt{r}_4)$	a	Push current object $v = \mathtt{r}_4$, return address a and return type m
2	Check that v's class defines method p	b	Reset flags and unused registers
3	Load parameters $\overline{v}$ from $\mathcal{O}$	c	Replace object identities with indexes in $\mathcal{O}$
4	Dynamic typecheck on $\overline{v}$	d	Jump to callback address (*run external code*)
5	Perform dynamic dispatch (*run method p code*)		
Exit point		Returnback entry point	
6	Reset flags and unused registers	e	Pop return type m and check it
7	Replace object identities with indexes in $\mathcal{O}$	f	Dynamic typecheck
		g	Pop return address a, current object v and resume execution

Properties of the Compiler

$$\forall \mathsf{P}_1, \mathsf{P}_2 . \mathsf{P}_1 \simeq_{ctx} \mathsf{P}_2 \iff [\![\mathsf{P}_1]\!] \simeq_{ctx} [\![\mathsf{P}_2]\!]$$

Properties of the Compiler

$$\forall \mathsf{P}_1, \mathsf{P}_2 . \mathsf{P}_1 \simeq_{ctx} \mathsf{P}_2 \iff [\![\mathsf{P}_1]\!] \simeq_{ctx} [\![\mathsf{P}_2]\!]$$

- correctness of $[\![\cdot]\!]$ ($\Rightarrow$) should follow from the correctness of $(\!|\cdot|\!)$

Properties of the Compiler

$$\forall \mathsf{P}_1, \mathsf{P}_2. \mathsf{P}_1 \simeq_{ctx} \mathsf{P}_2 \iff [\![\mathsf{P}_1]\!] \simeq_{ctx} [\![\mathsf{P}_2]\!]$$

- correctness of $[\![\cdot]\!]$ ($\Rightarrow$) should follow from the correctness of $(\!|\cdot|\!)$
- security of $[\![\cdot]\!]$ ($\Leftarrow$) requires BT

Properties of the Compiler

$$\forall P_1, P_2. P_1 \simeq_{ctx} P_2 \iff [\![P_1]\!] \simeq_{ctx} [\![P_2]\!]$$

- correctness of $[\![\cdot]\!]$ ($\Rightarrow$) should follow from the correctness of $(\!|\cdot|\!)$
- security of $[\![\cdot]\!]$ ($\Leftarrow$) requires BT
- the gap between PMA and JavaJr is too big: trace-based BT

Traces for PMA

In this case:

- capture specific component-context interactions
- alternation of call/return

Traces for PMA

In this case:

- capture specific component-context interactions
- alternation of call/return

Generally:

- capture arbitrary component-context interactions
- alternation of call/return plus read and write to shared memory

Traces for PMA: formally

to the board

- reliance on operational semantics
- problems of read/write

$$
\begin{array}{ll}
\textit{labels} & \lambda ::= \alpha \mid \tau \\
\textit{actions} & \alpha ::= \gamma! \mid \gamma? \mid \surd \\
\textit{observables} & \gamma ::= \mathbf{call\ a\ \overline{w}} \mid \mathbf{ret\ a\ w, id}!
\end{array}
$$

Instead of creating the BT now, we look at the multimodule case and show only that BT

Multiple Isolation in PMA

- multiple programmers may not trust each others
- each programmer gets an enclave

Multiple Isolation in PMA

- multiple programmers may not trust each others
- each programmer gets an enclave

<table>
<tr><th rowspan="2">From \ To</th><th rowspan="2">Unprotected</th><th colspan="3">Protected</th></tr>
<tr><th>Entry Point</th><th>Code</th><th>Data</th></tr>
<tr><td>Unprotected</td><td>r w x</td><td>x</td><td></td><td></td></tr>
<tr><td rowspan="3">Protected</td><td rowspan="3">r w x</td><td colspan="3">Same id</td></tr>
<tr><td>r x</td><td>r x</td><td>r w</td></tr>
<tr><td colspan="3">Different id</td></tr>
<tr><td>x</td><td></td><td></td></tr>
</table>

Multiple Isolation in PMA: Formally

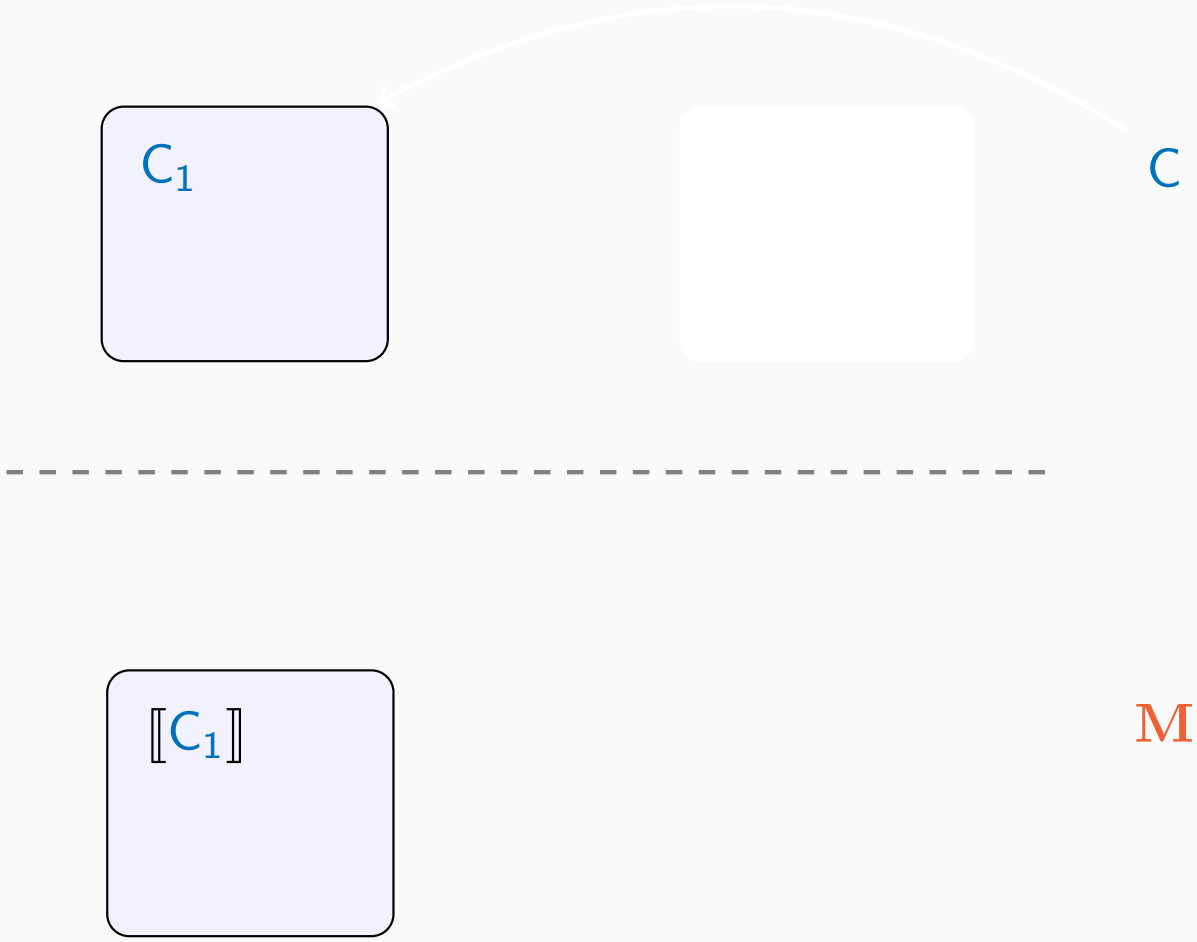
to the board

- multiple domains
- randomisation
- contextual preorders

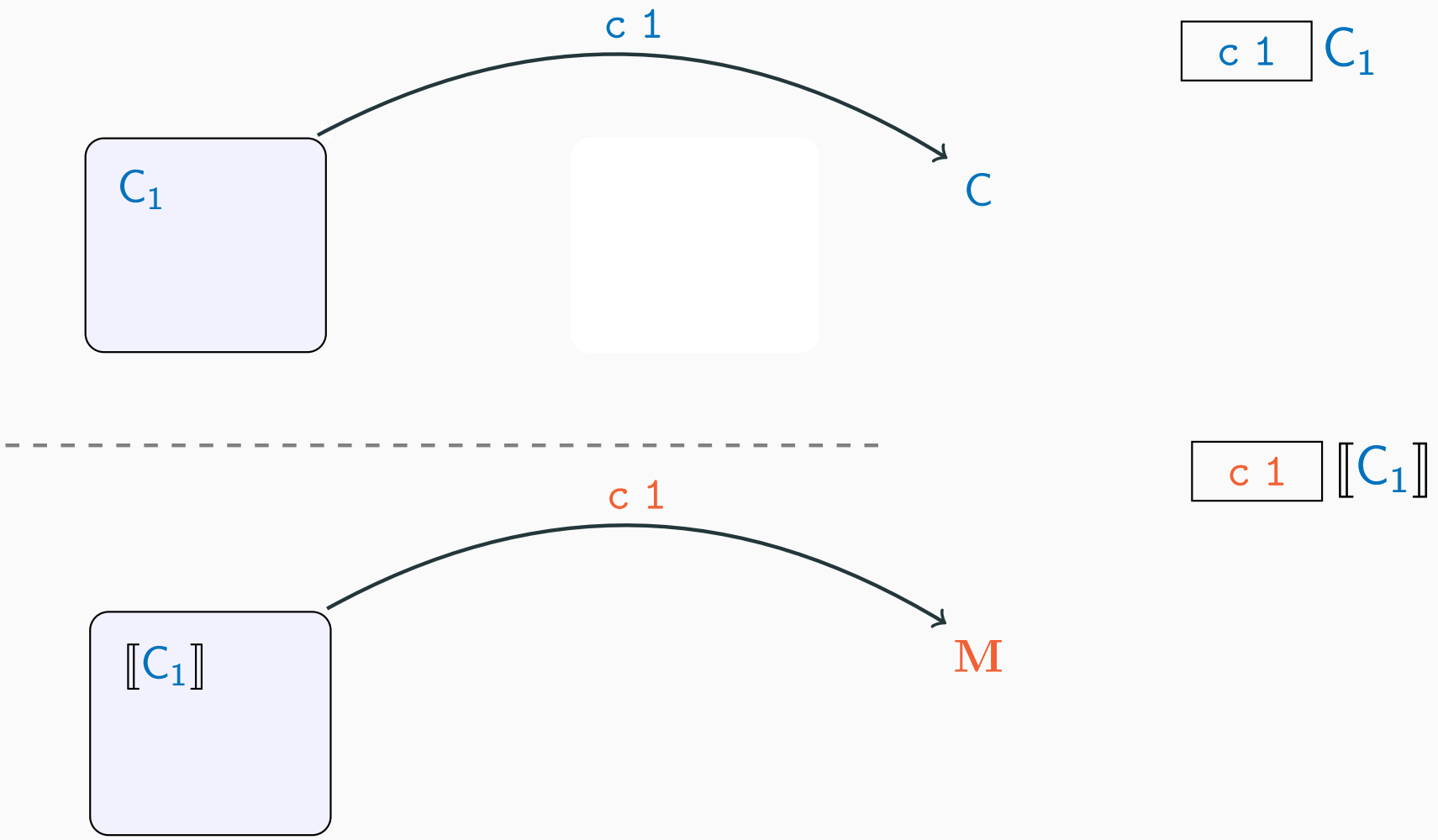
Linking Vulnerabilities

1. Call stack shortcutting;
2. Types of objects in other modules;
3. Existence of objects in other modules.

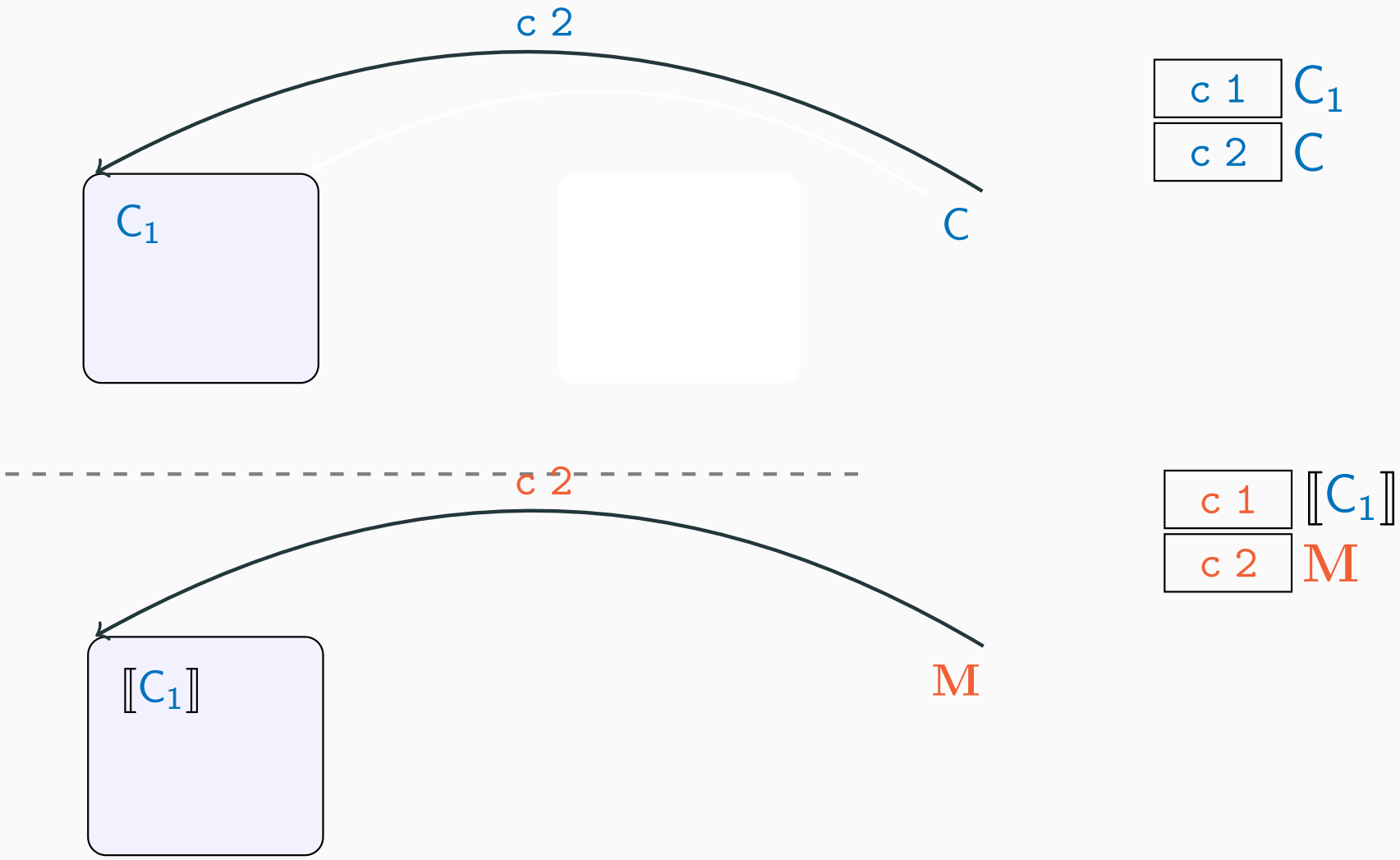
Calls - 1 Component



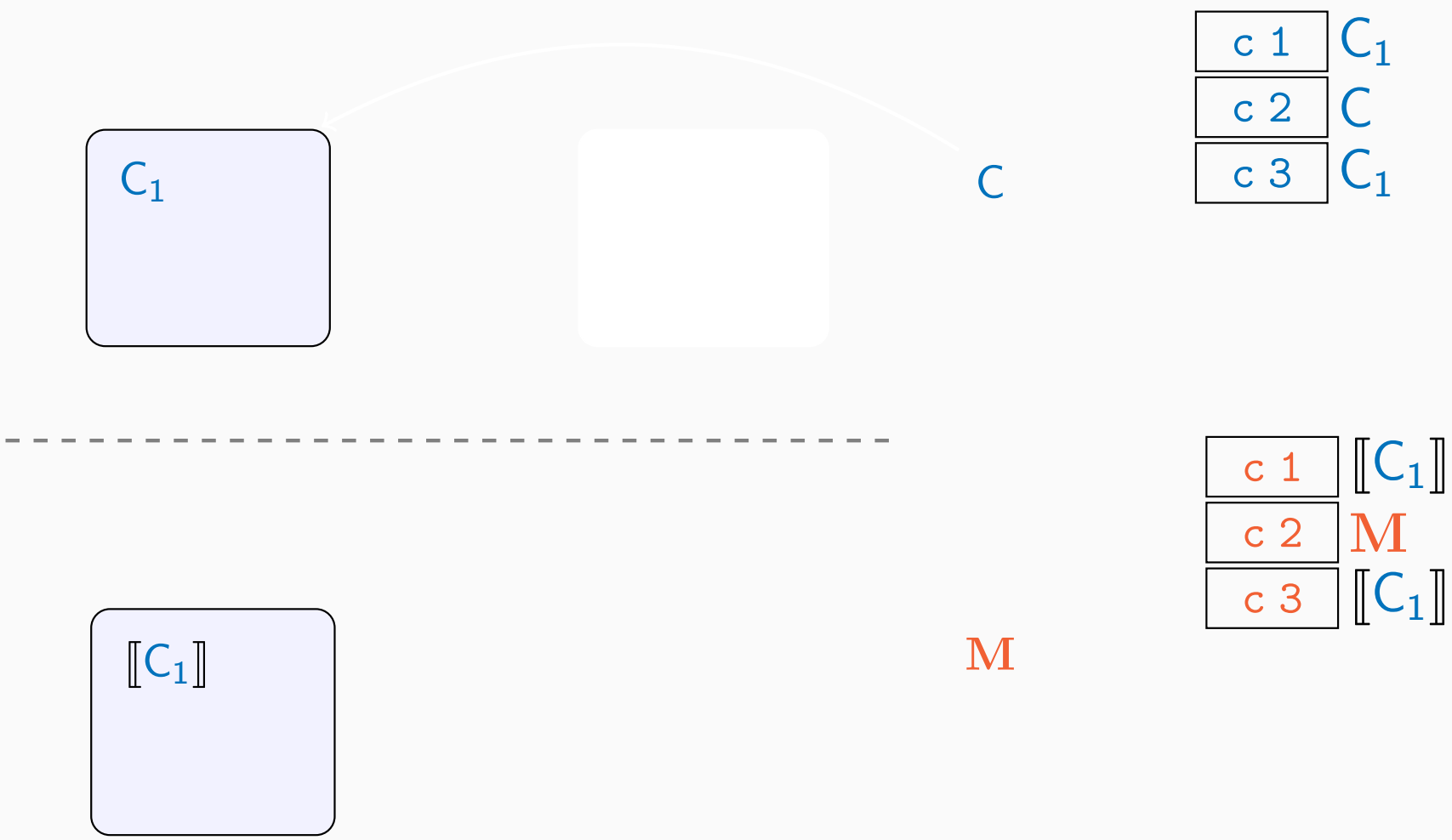
Calls - 1 Component



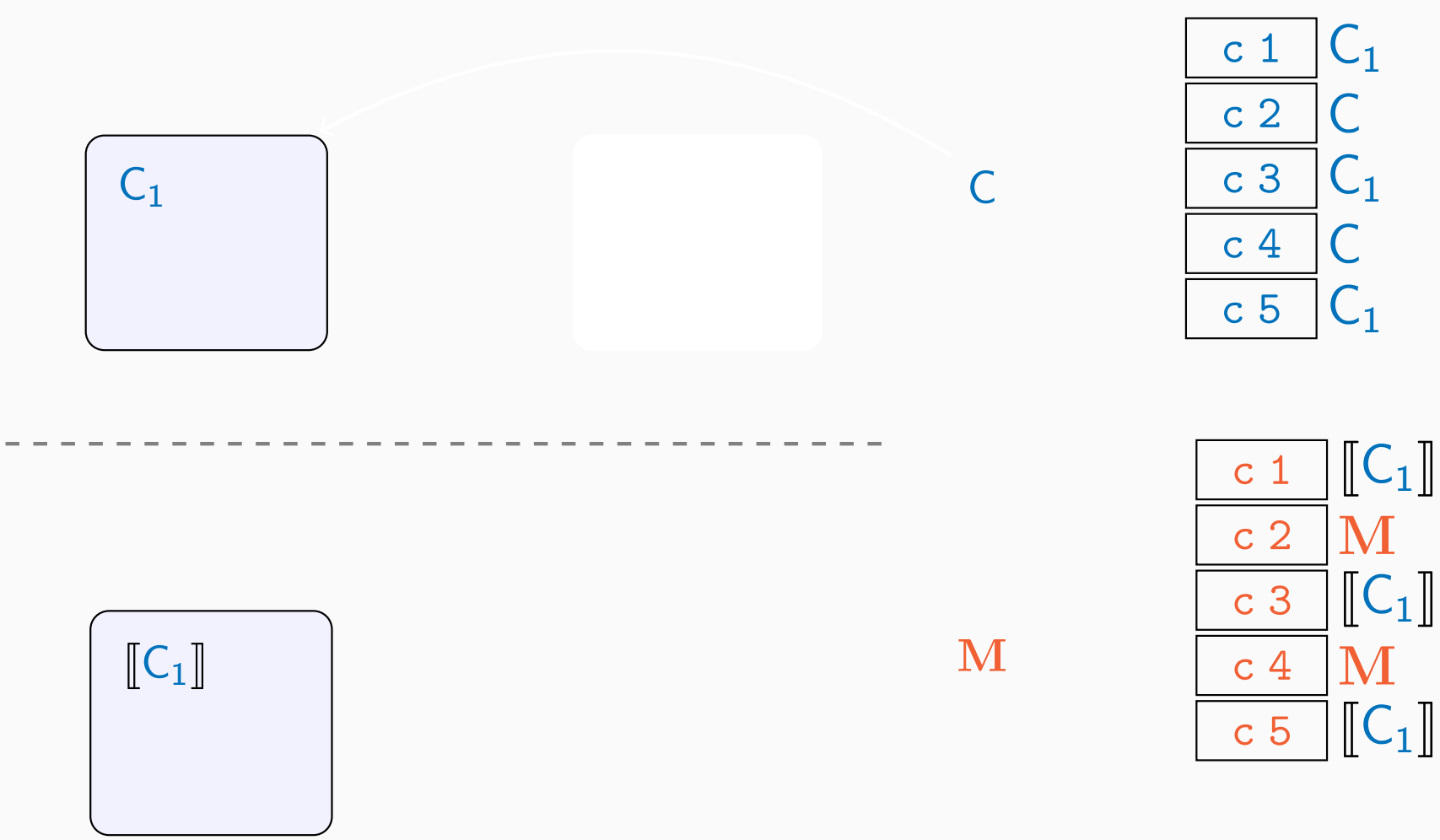
Calls - 1 Component



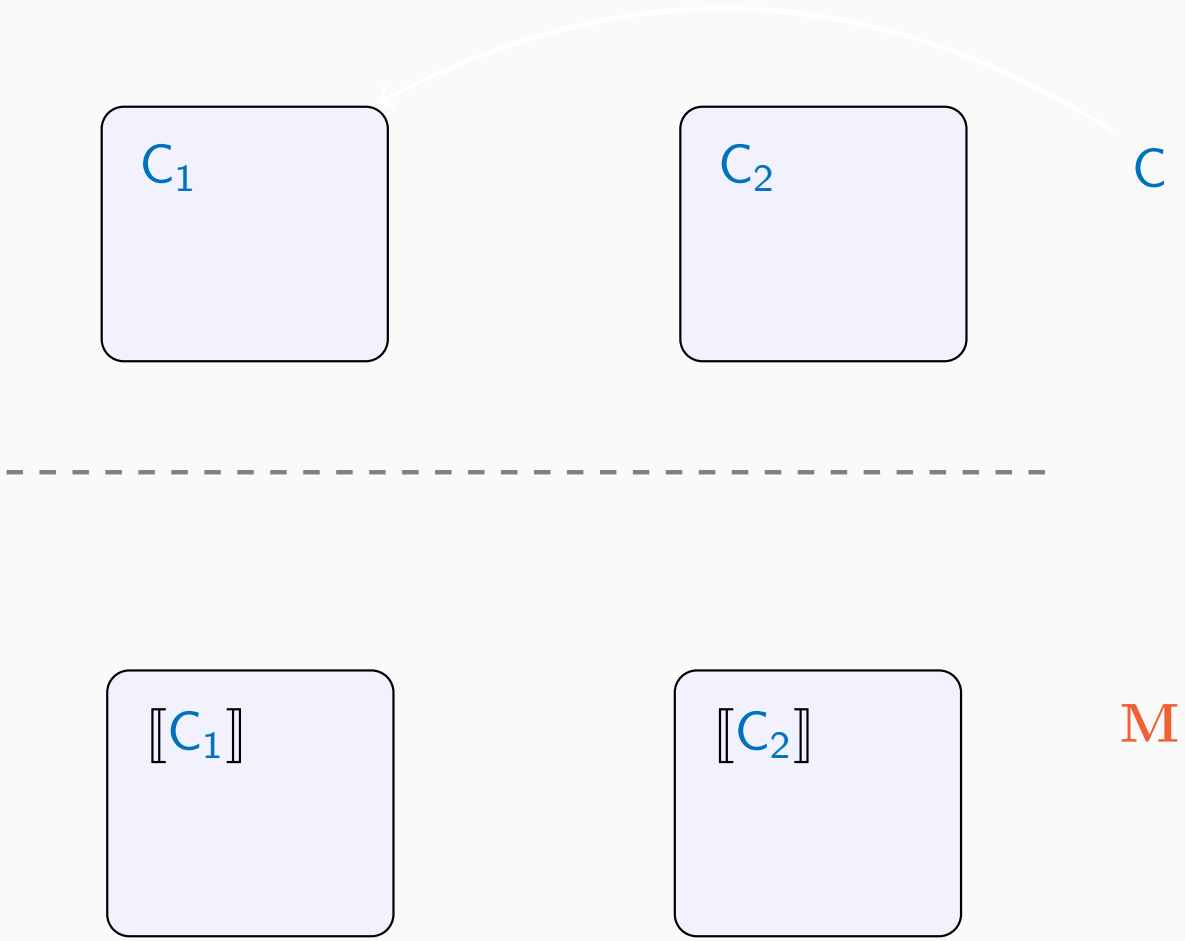
Calls - 1 Component



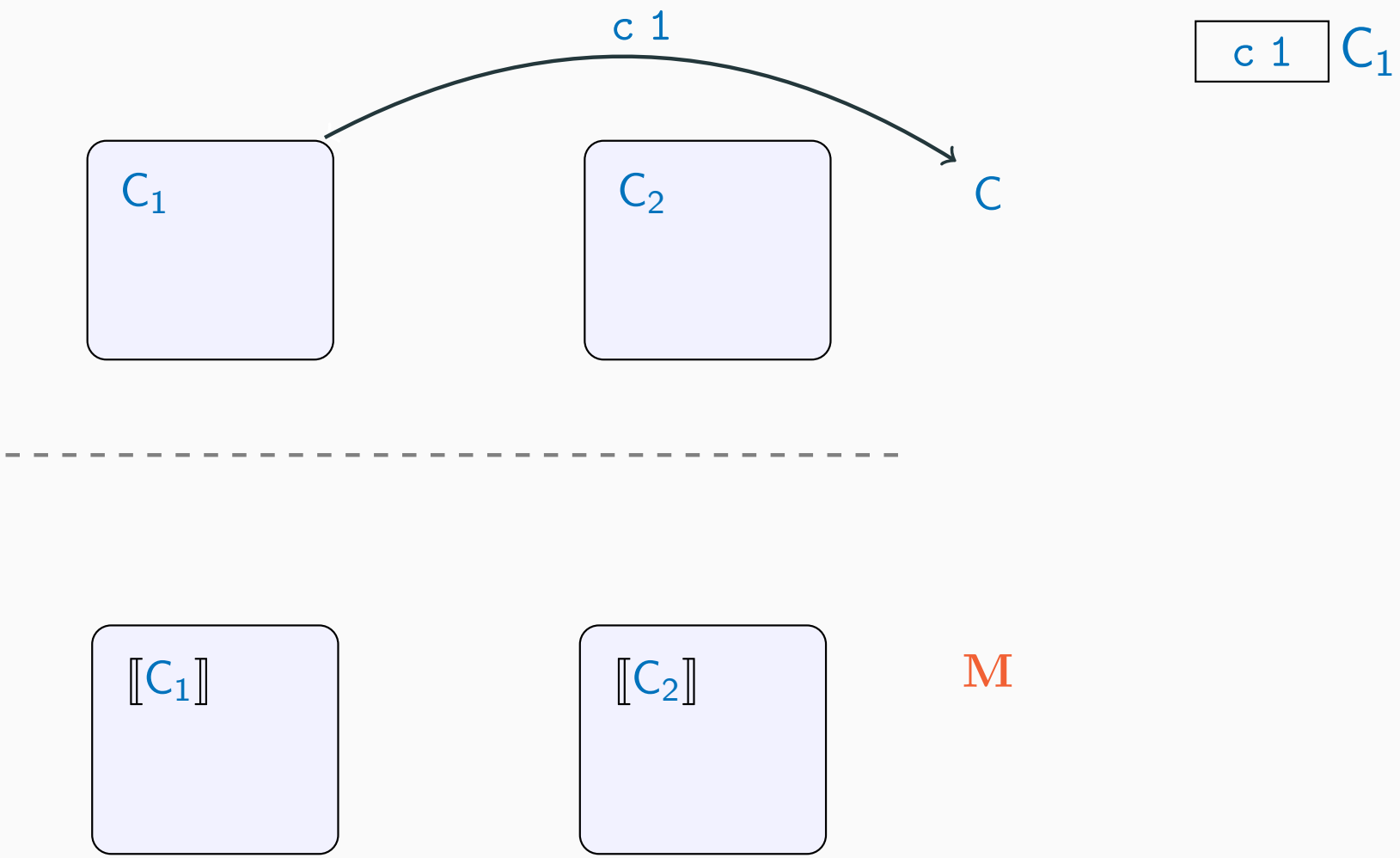
Calls - 1 Component



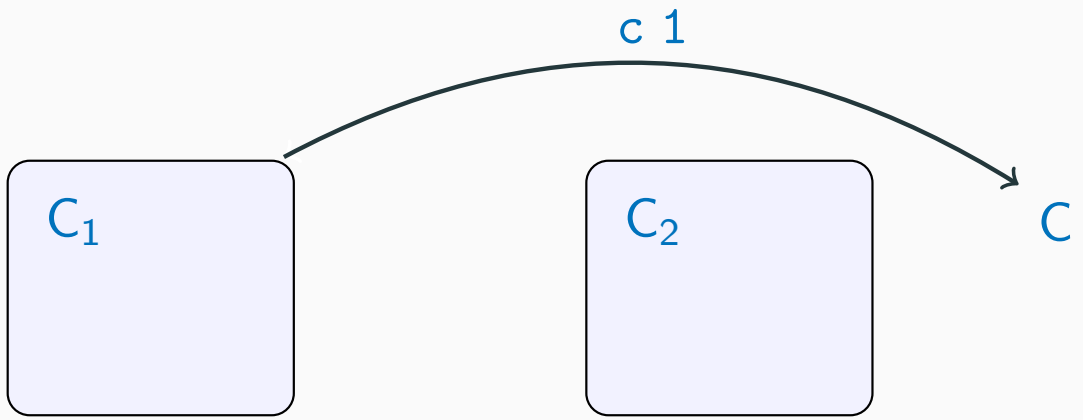
Call Stack Shortcutting



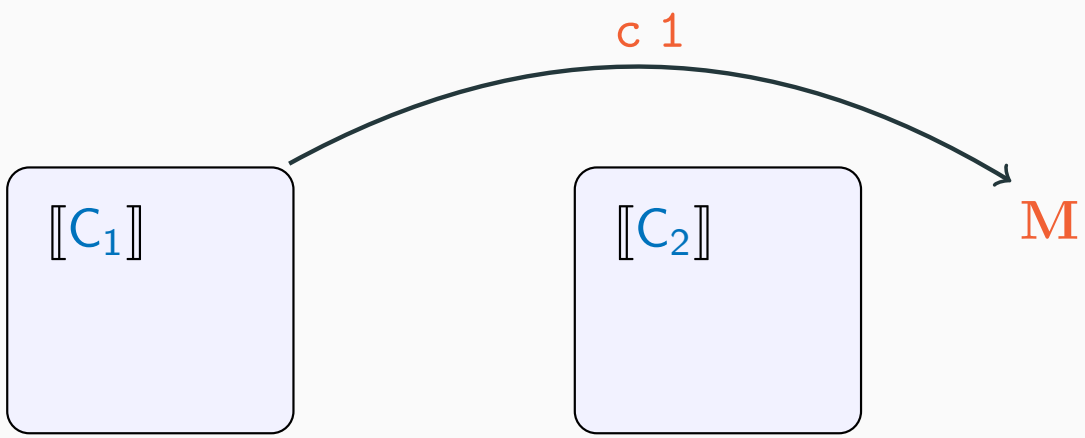
Call Stack Shortcutting



Call Stack Shortcutting

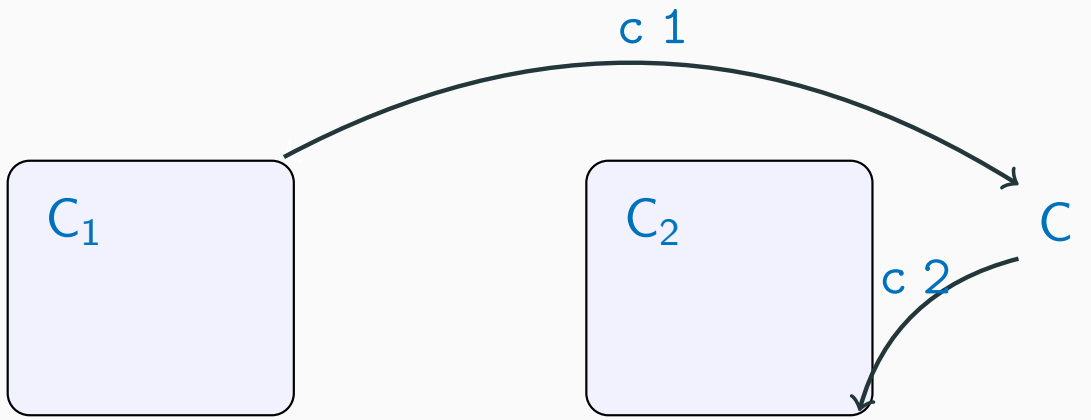


c 1 C_1

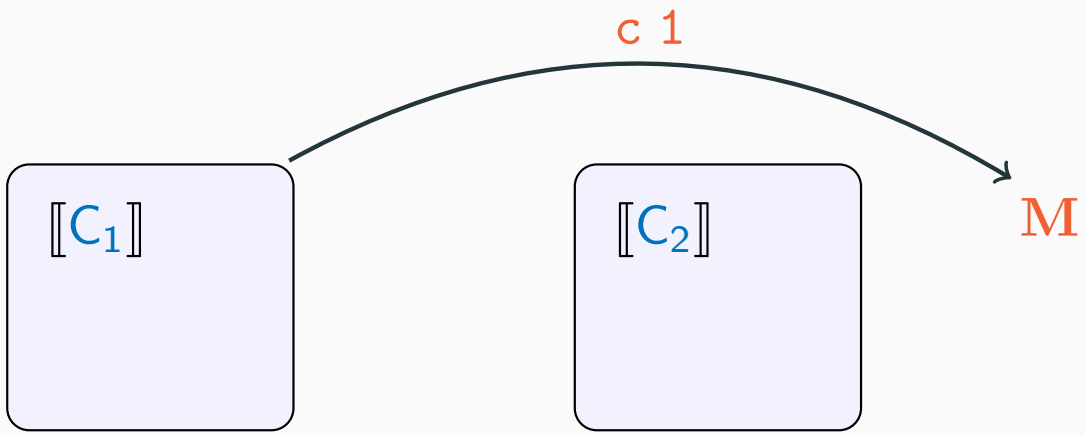


c 1 $[\![C_1]\!]$

Call Stack Shortcutting

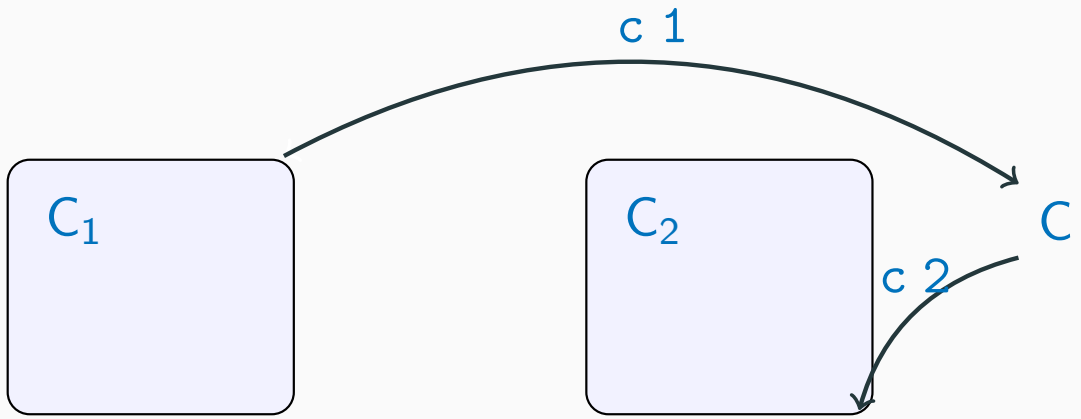


c 1	C_1
c 2	C

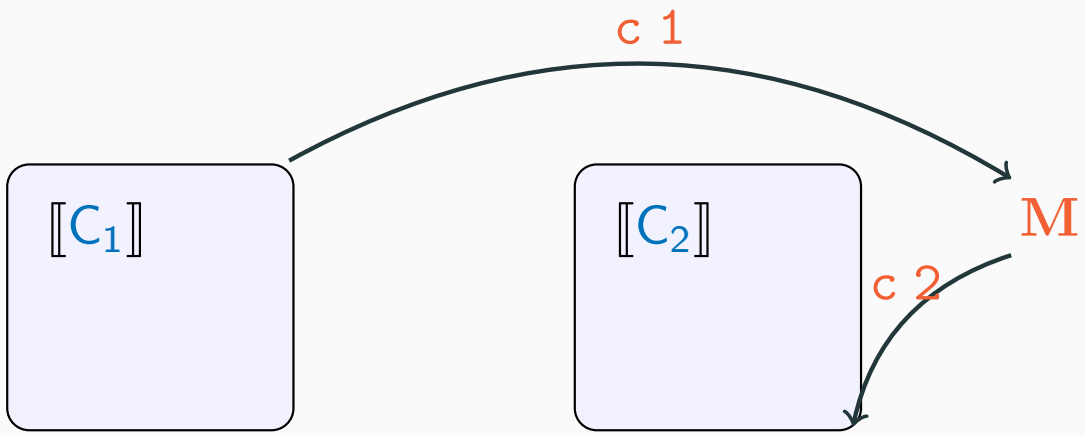


c 1	$[\![C_1]\!]$

Call Stack Shortcutting

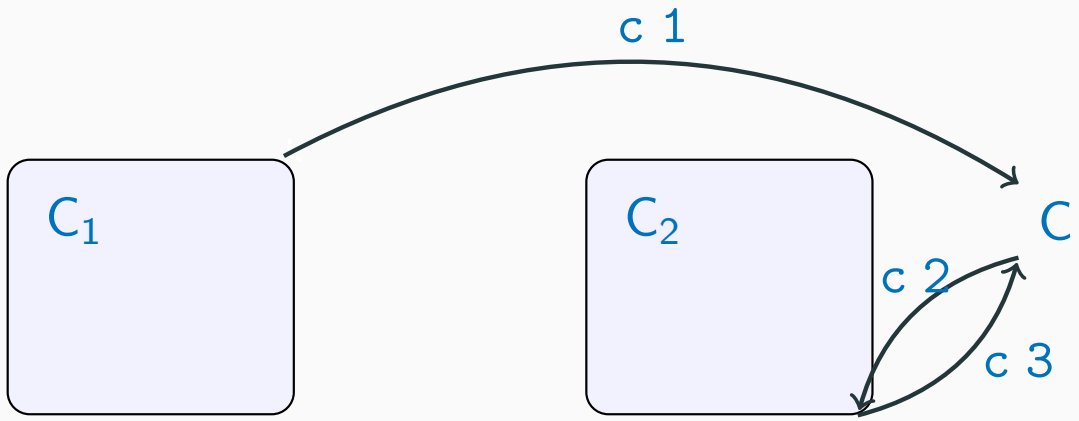


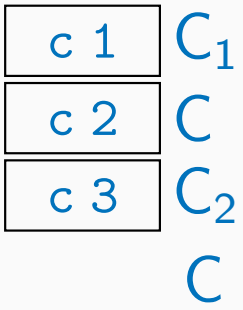
c 1	C_1
c 2	C

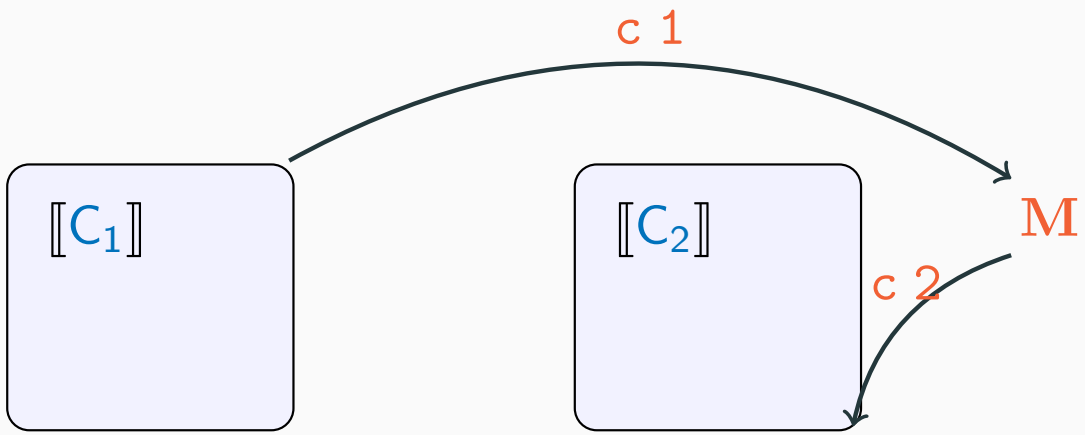


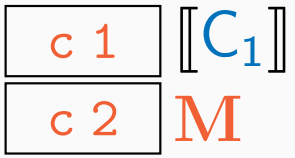
c 1	$[\![C_1]\!]$
c 2	M

Call Stack Shortcutting

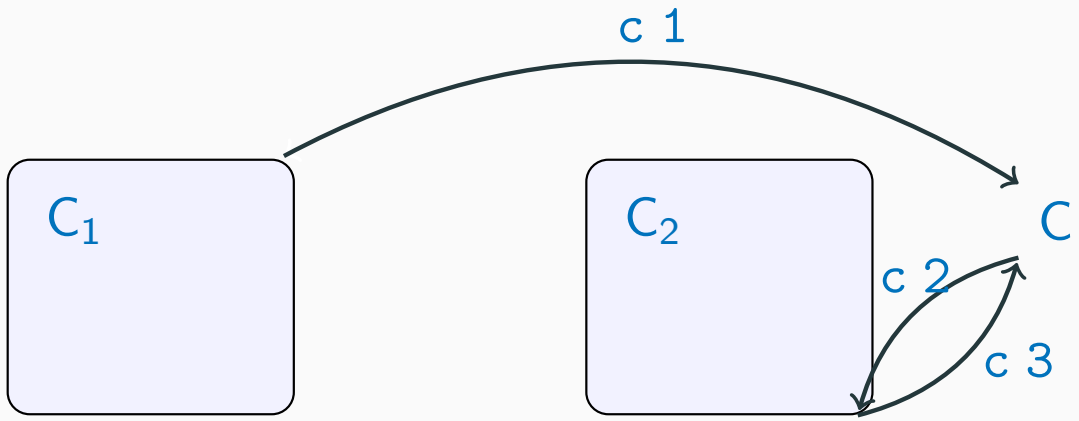




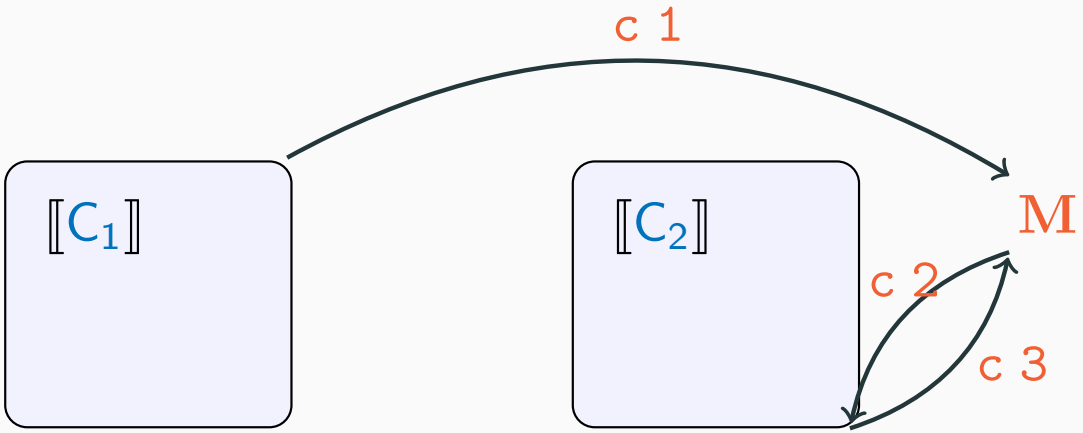




Call Stack Shortcutting

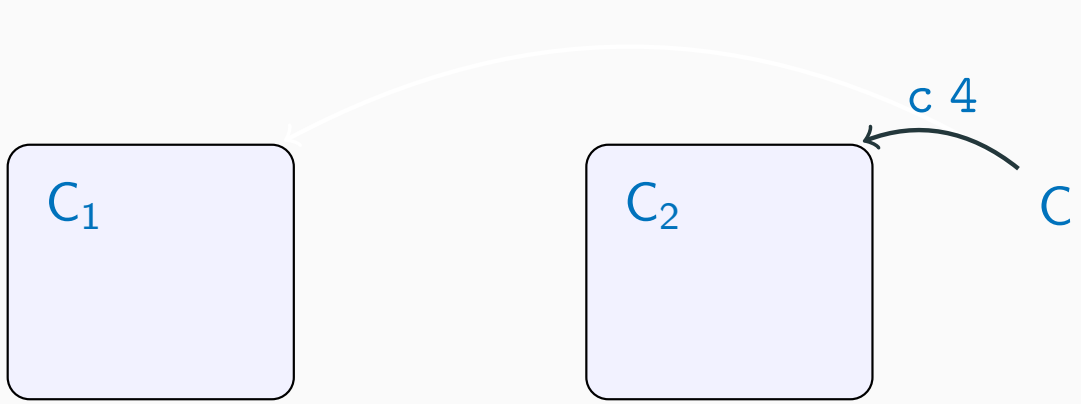


c 1	C_1
c 2	C
c 3	C_2
	C

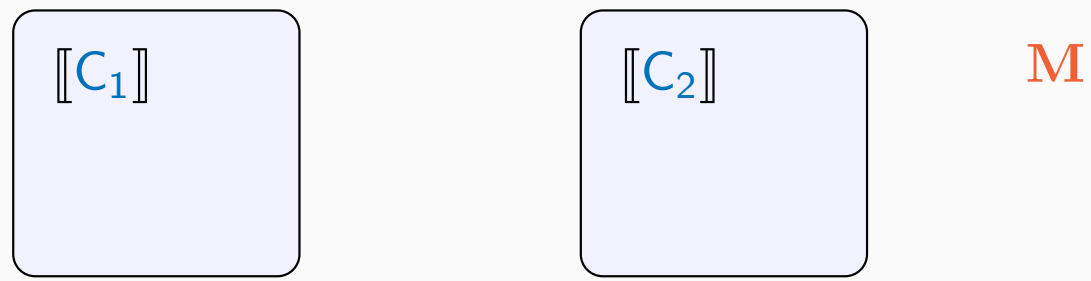


c 1	$[\![C_1]\!]$
c 2	M
c 3	$[\![C_2]\!]$
	M

Call Stack Shortcutting

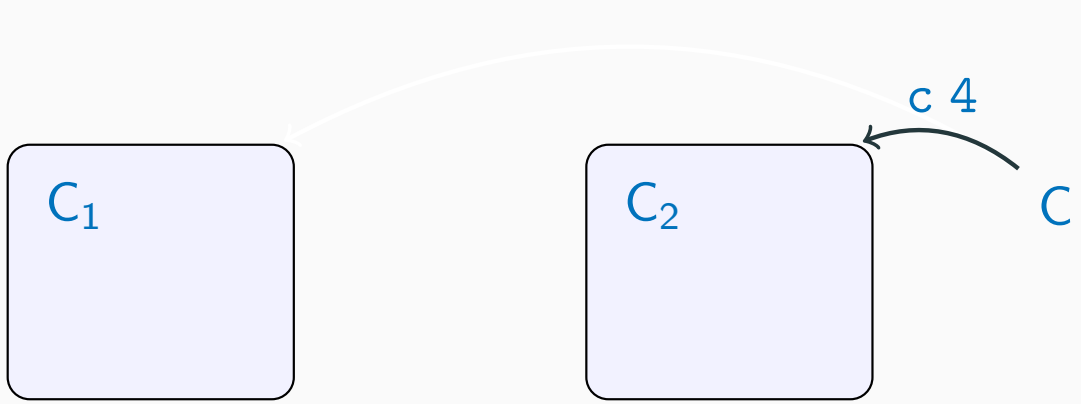


c 1	C_1
c 2	C
c 3	C_2
c 4	C

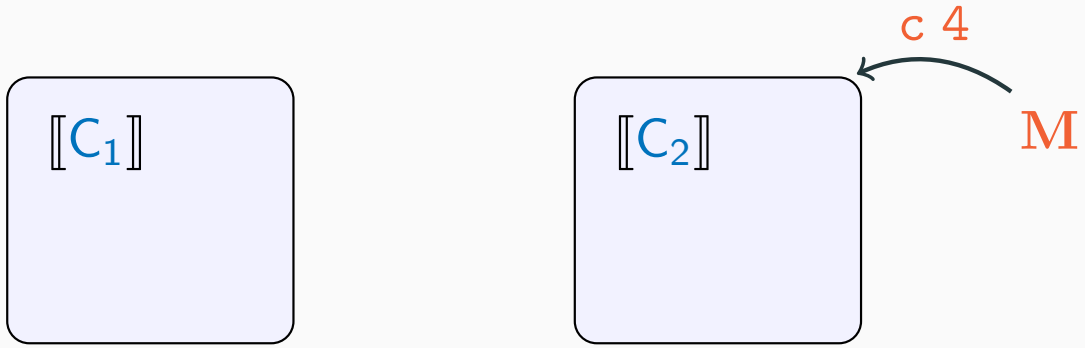


c 1	$[\![C_1]\!]$
c 2	M
c 3	$[\![C_2]\!]$
	M

Call Stack Shortcutting

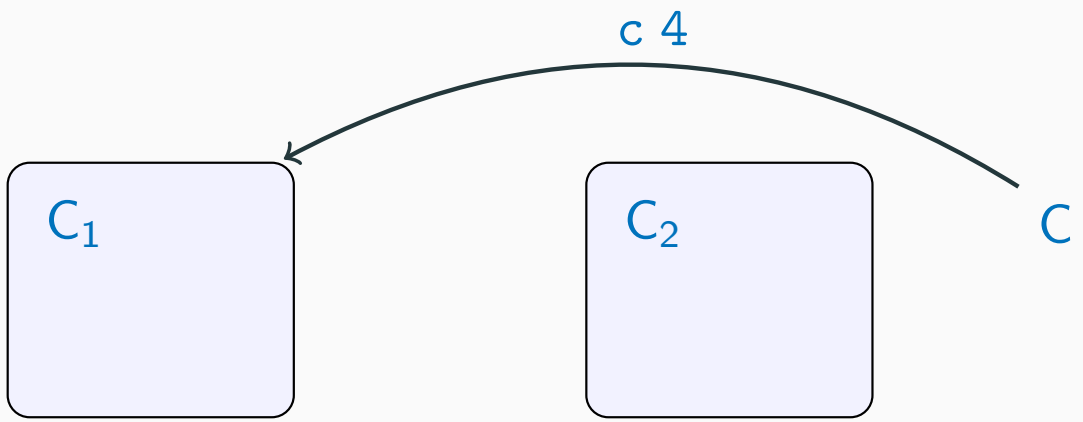


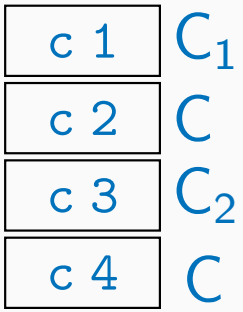
c 1	C_1
c 2	C
c 3	C_2
c 4	C

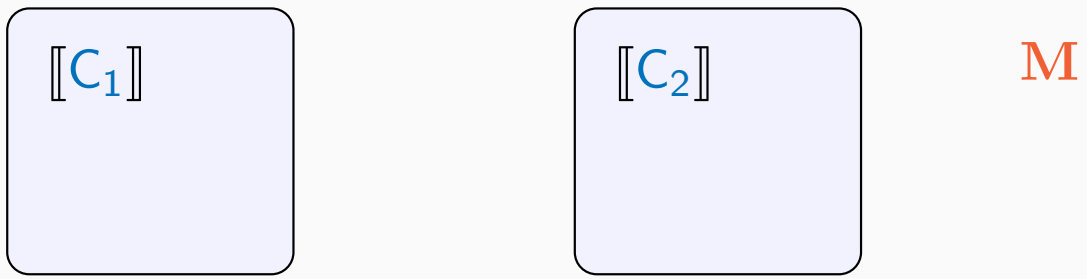


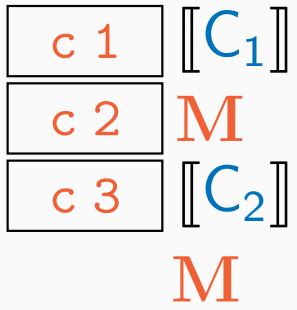
c 1	$[\![C_1]\!]$
c 2	M
c 3	$[\![C_2]\!]$
c 4	M

Call Stack Shortcutting

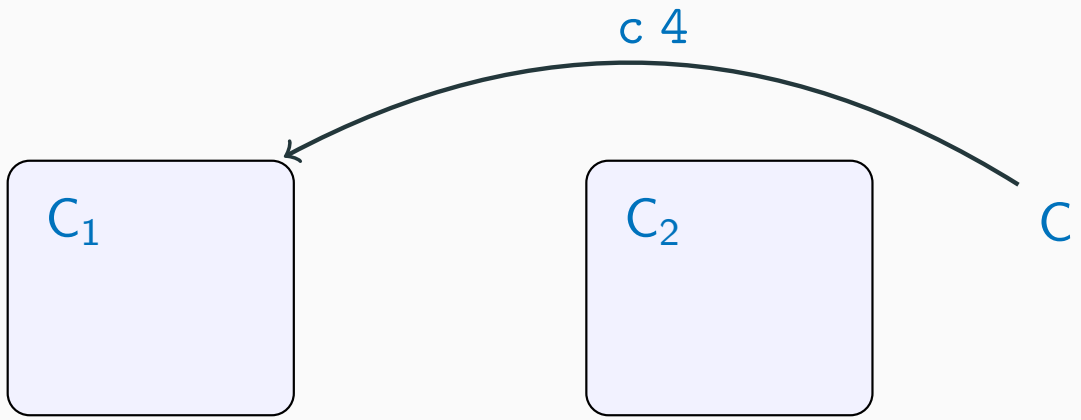




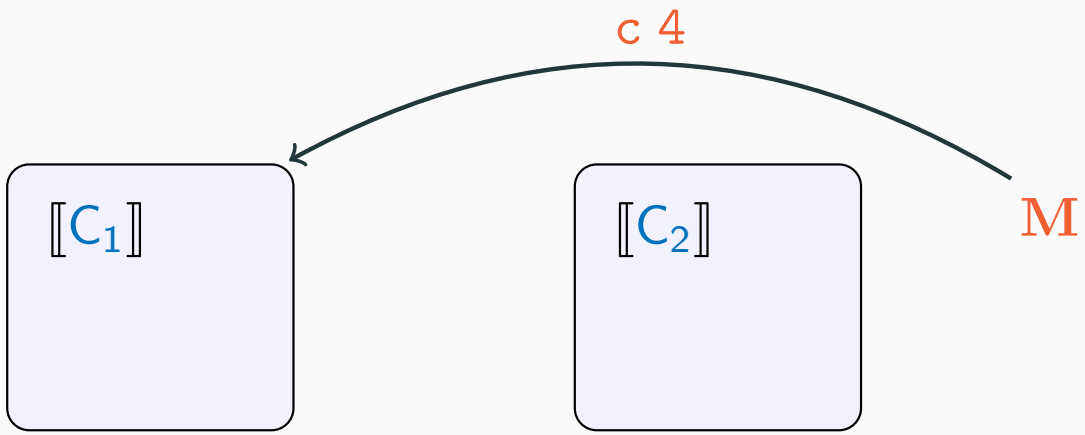




Call Stack Shortcutting

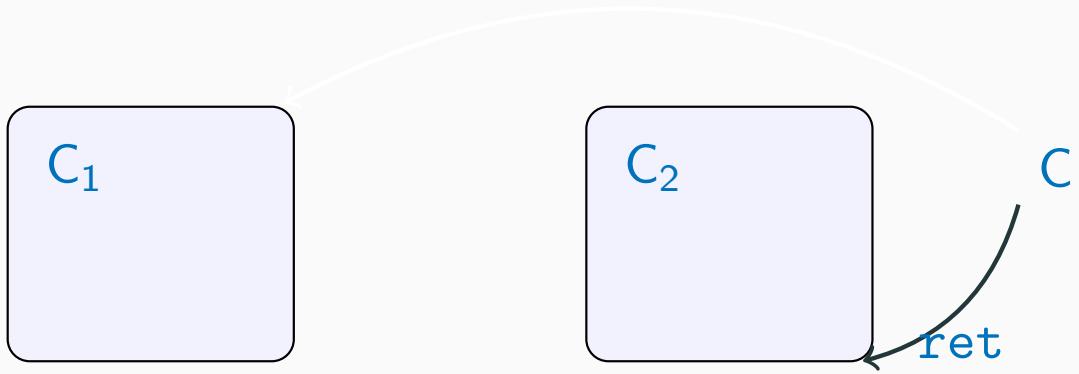


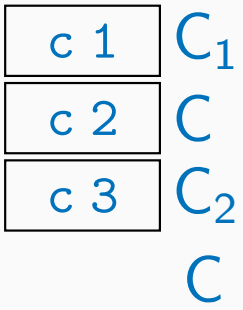
c 1	C_1
c 2	C
c 3	C_2
c 4	C

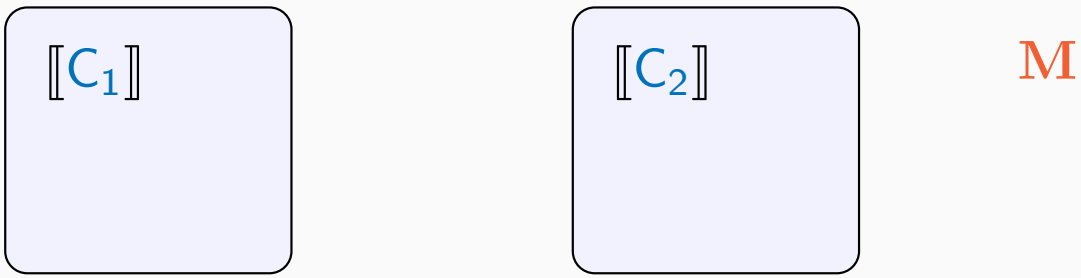


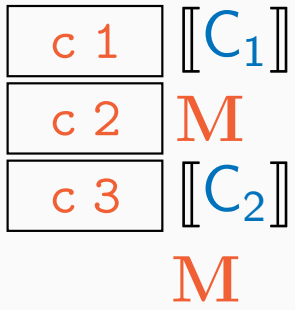
c 1	$[\![C_1]\!]$
c 2	M
c 3	$[\![C_2]\!]$
c 4	M

Call Stack Shortcutting

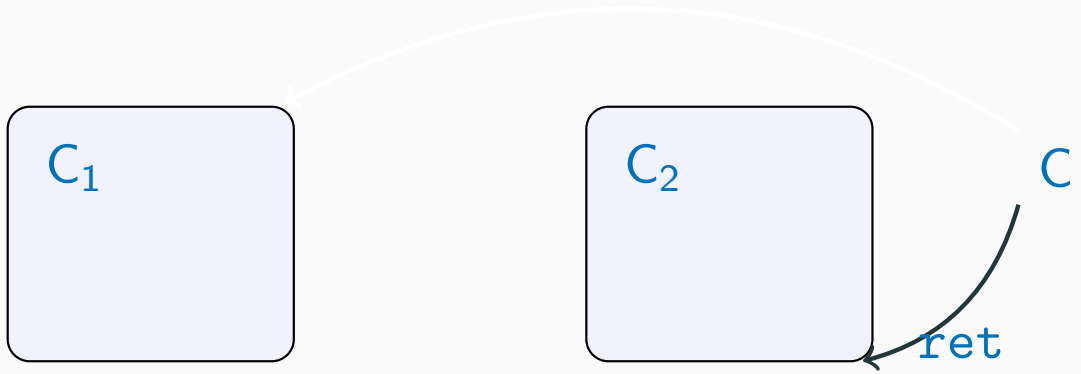


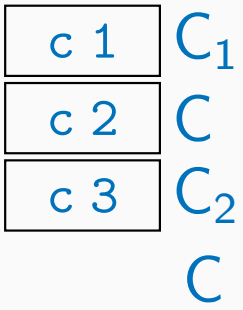


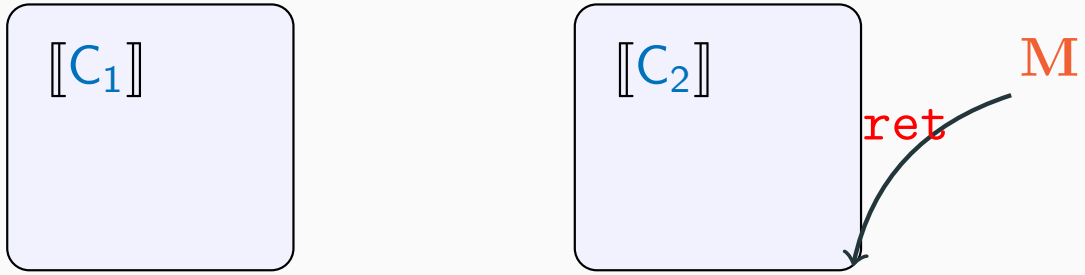


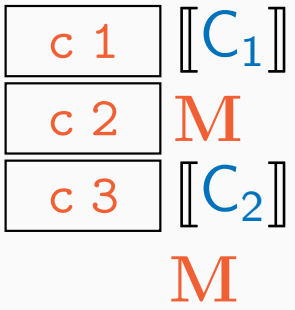


Call Stack Shortcutting

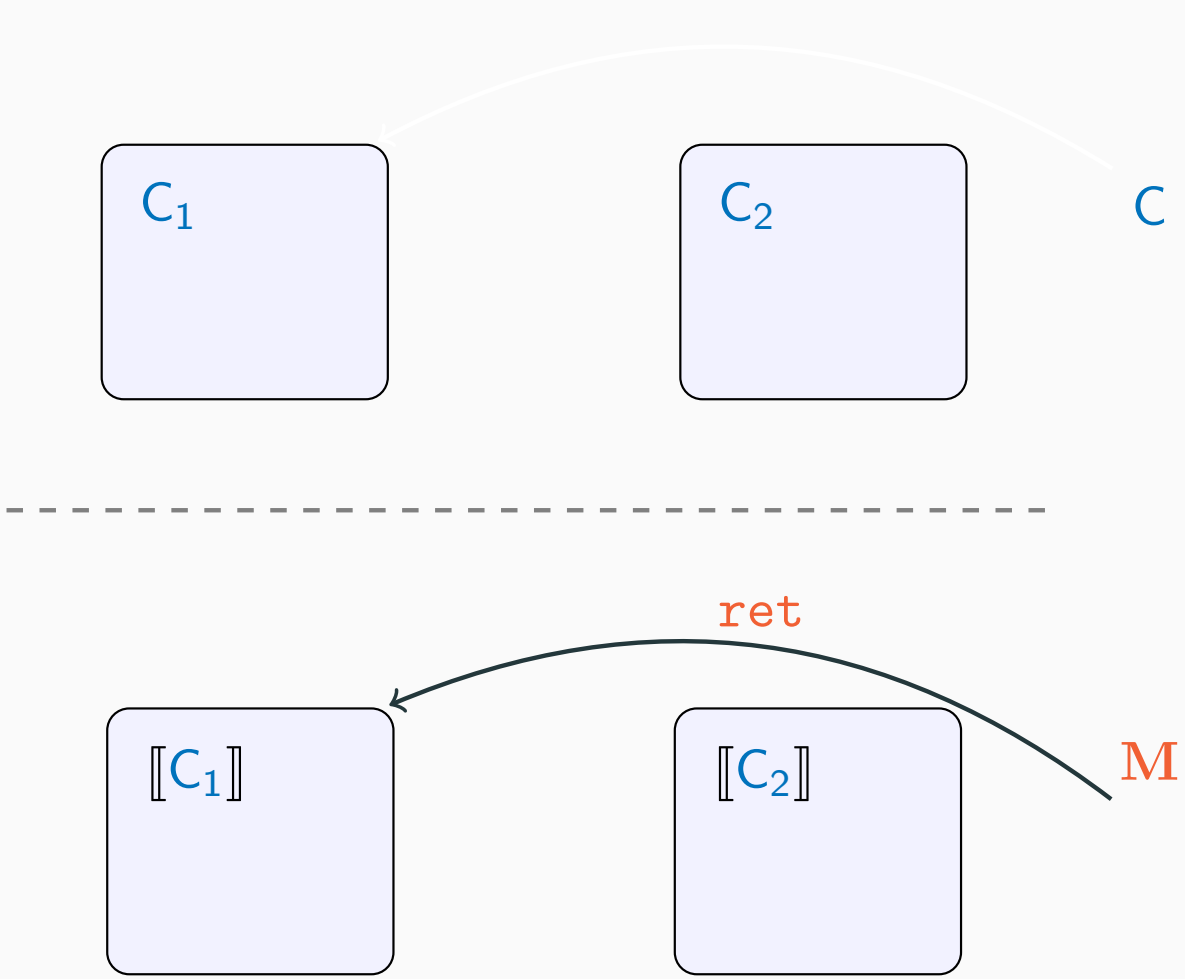


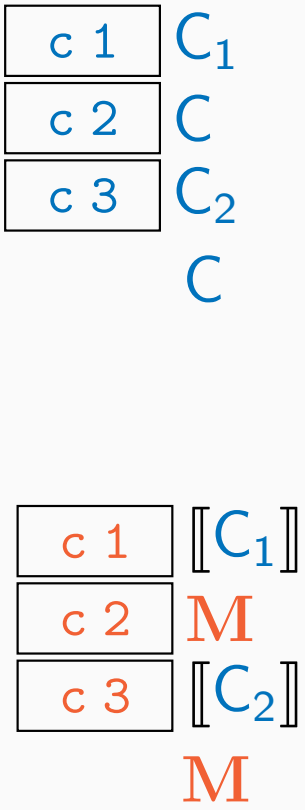




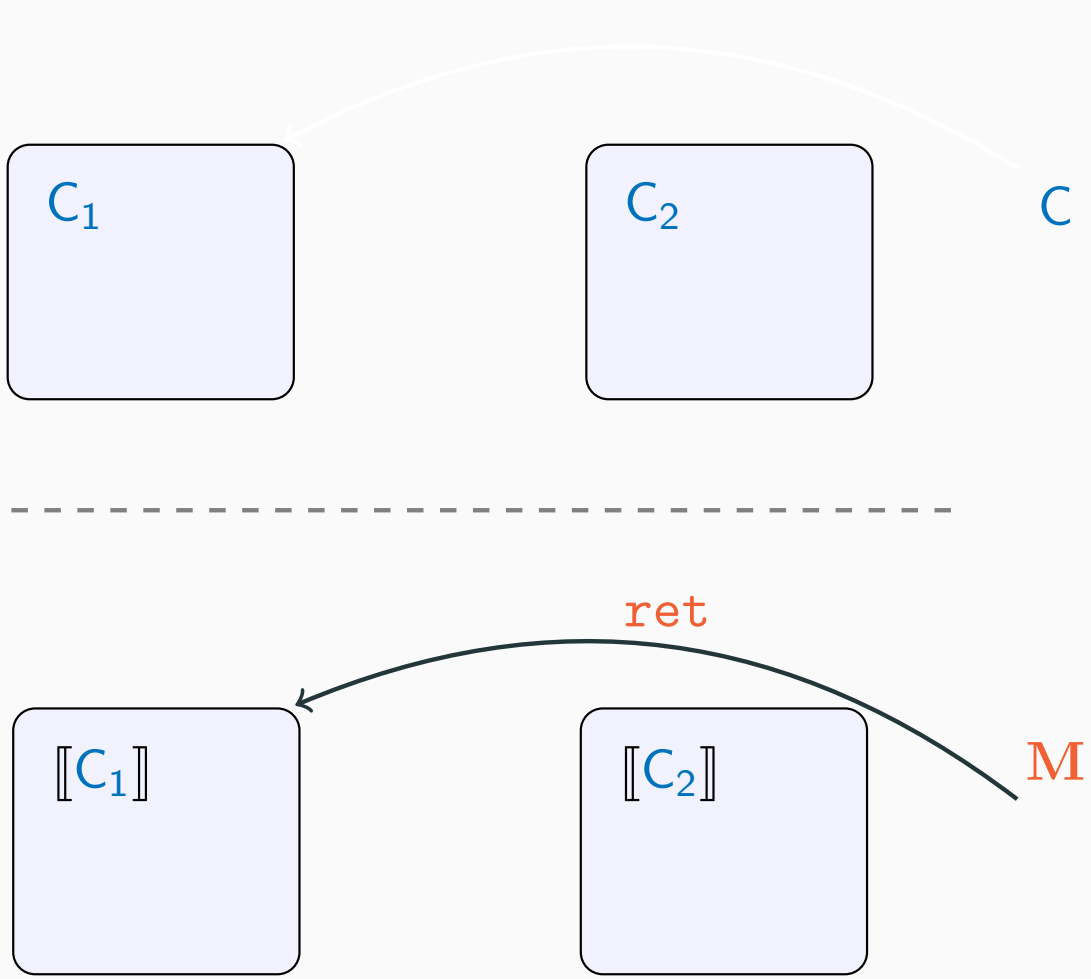


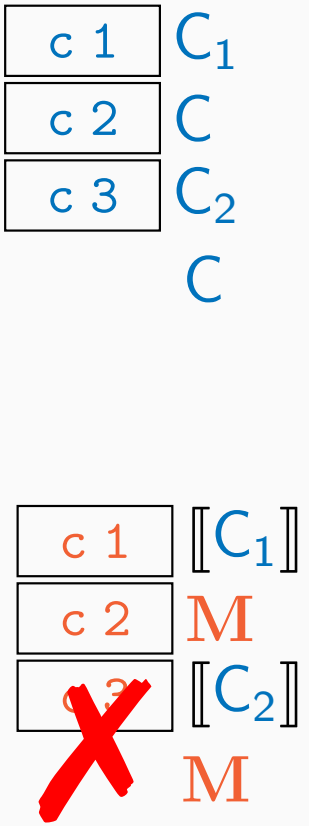
Call Stack Shortcutting



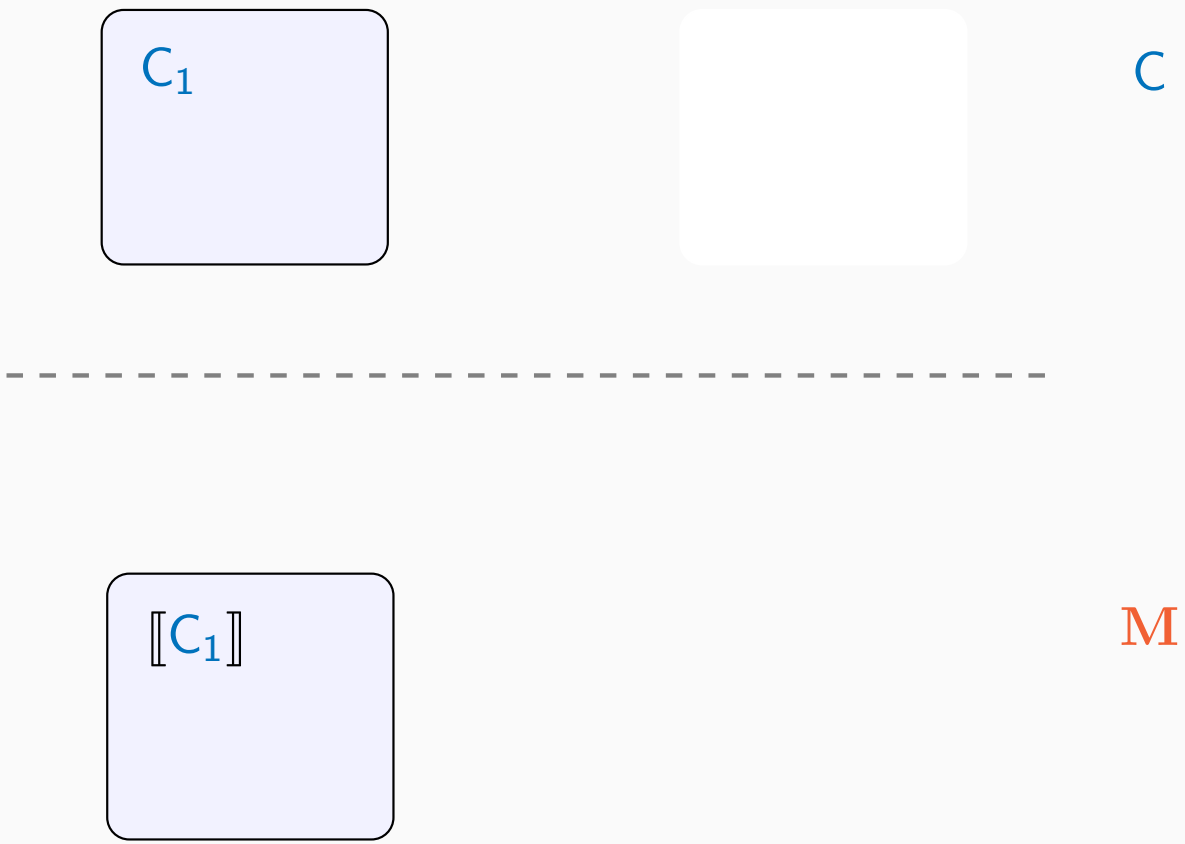


Call Stack Shortcutting

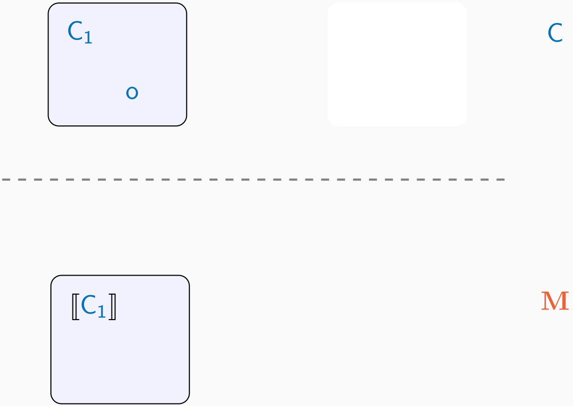




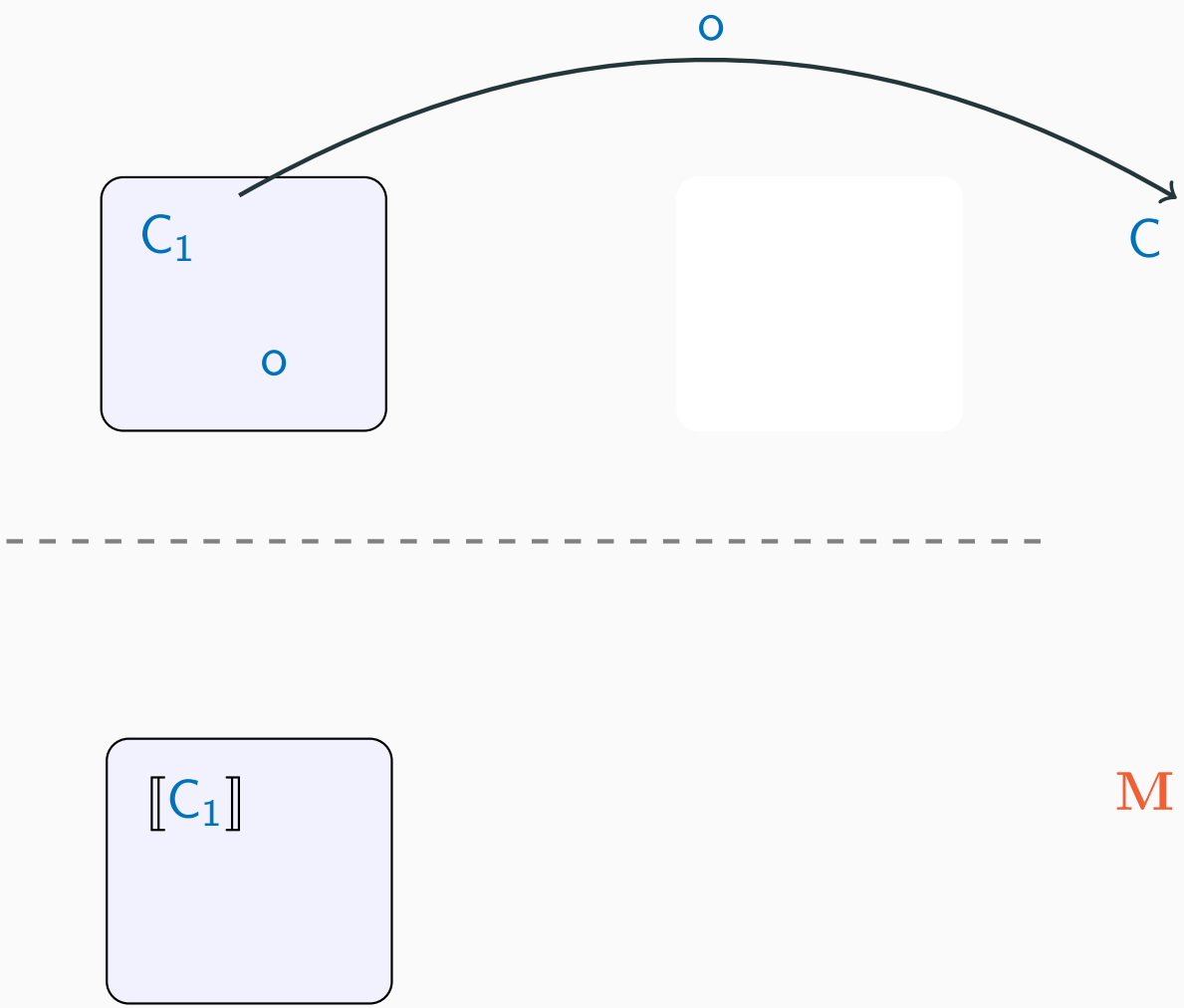
Object Guessing - 1 Component



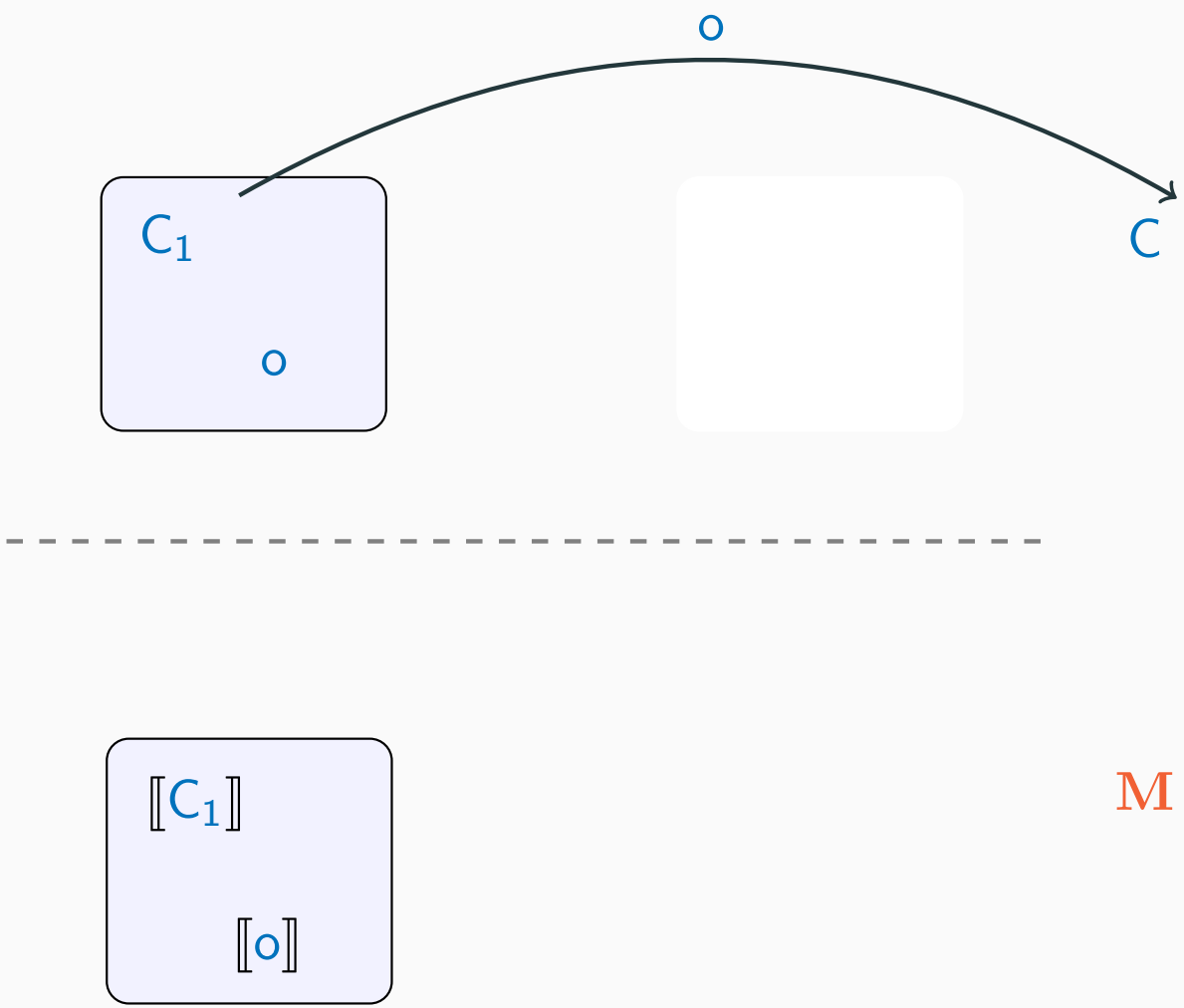
Object Guessing - 1 Component



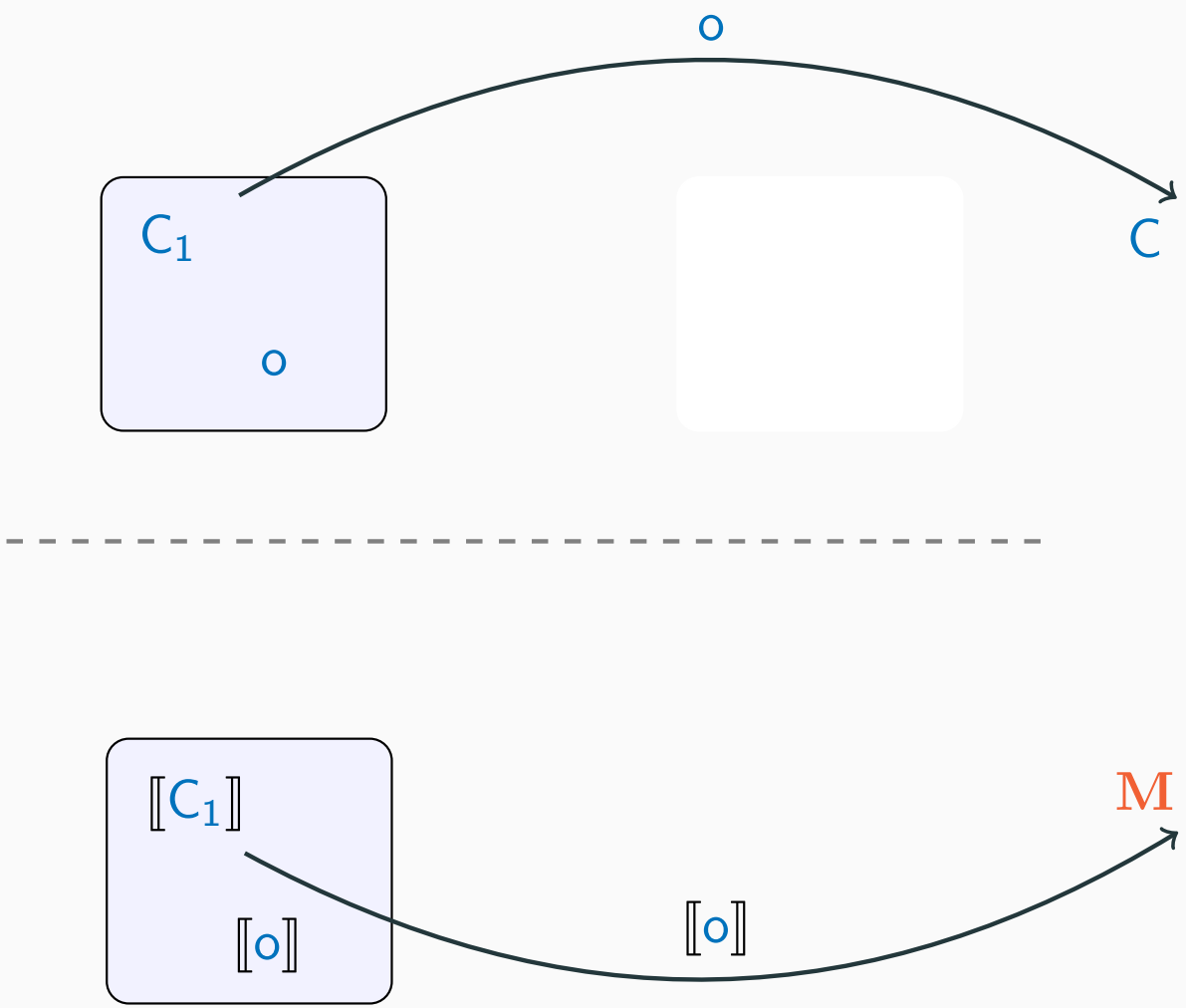
Object Guessing - 1 Component



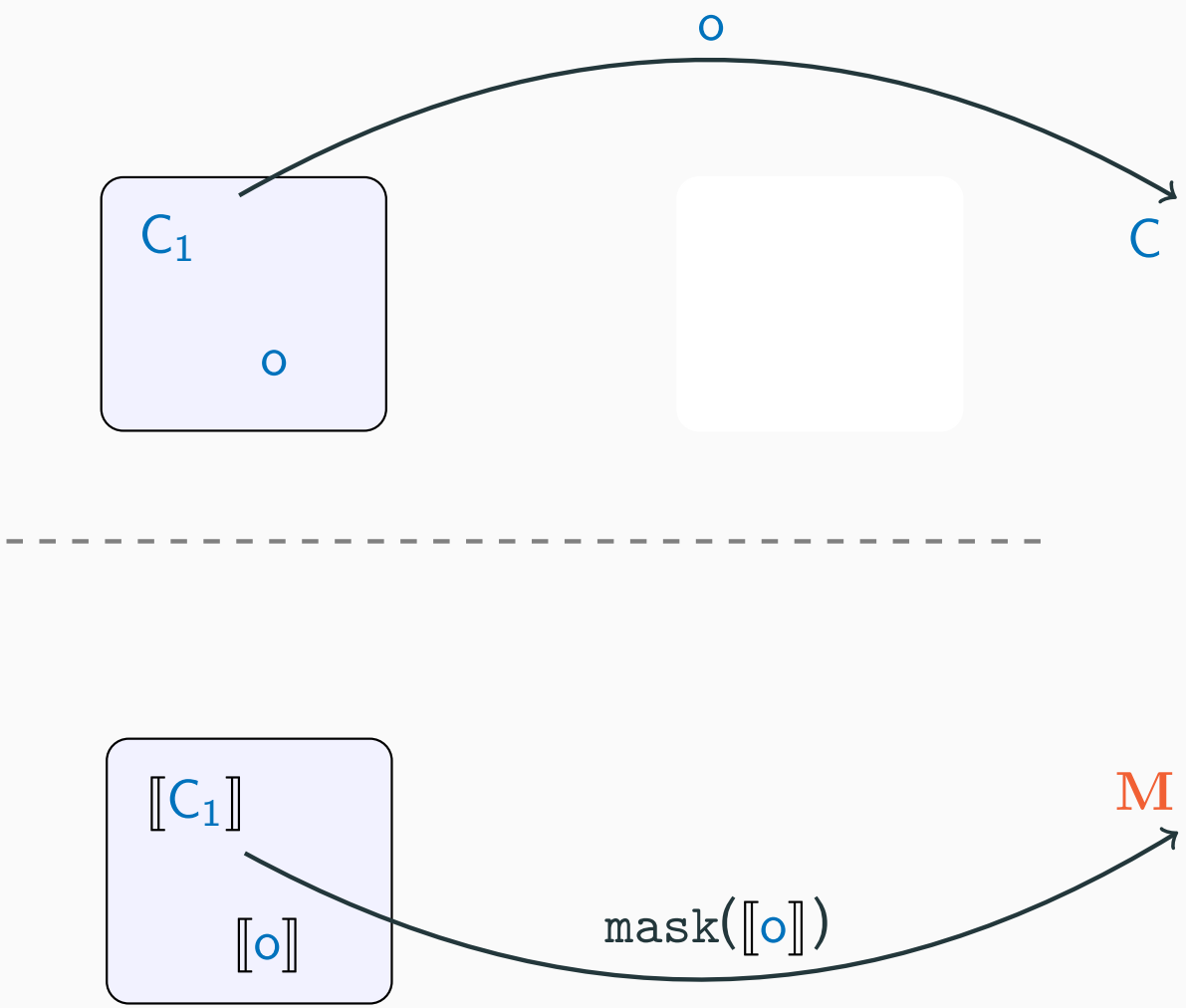
Object Guessing - 1 Component



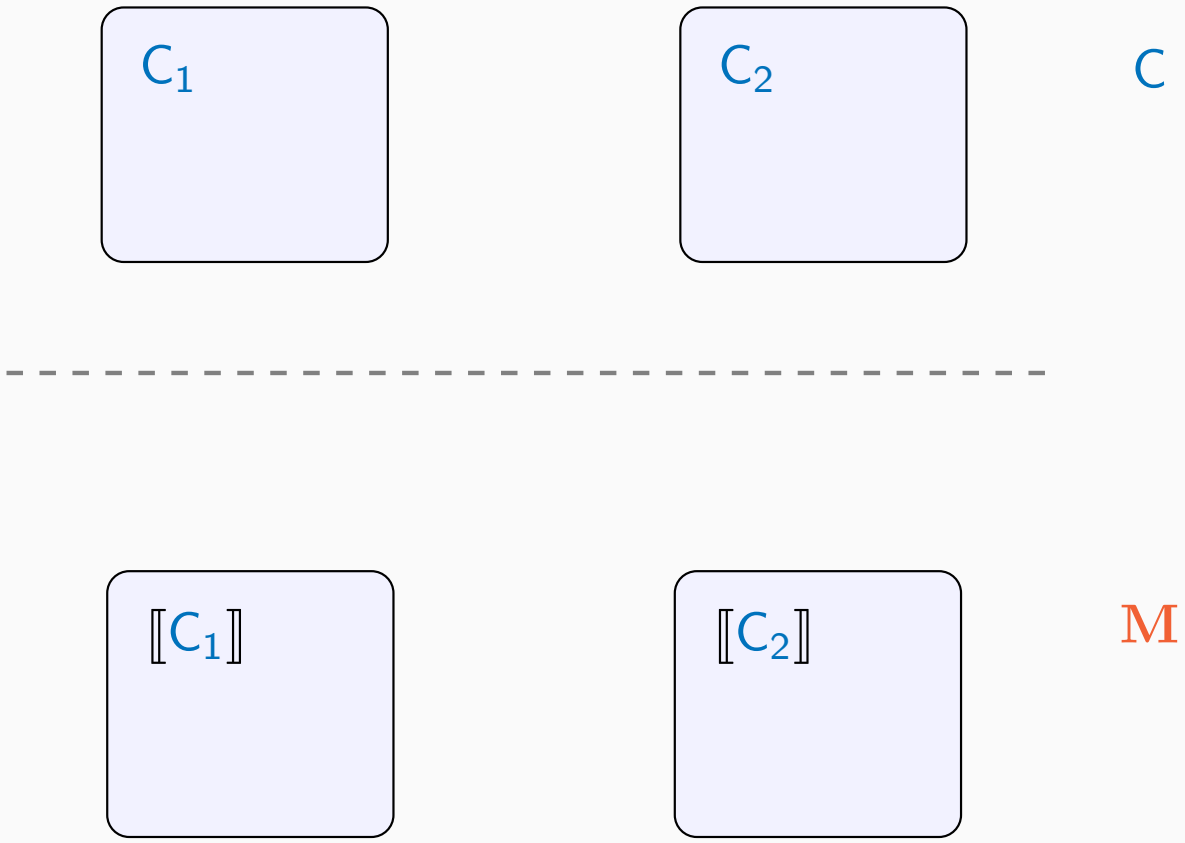
Object Guessing - 1 Component



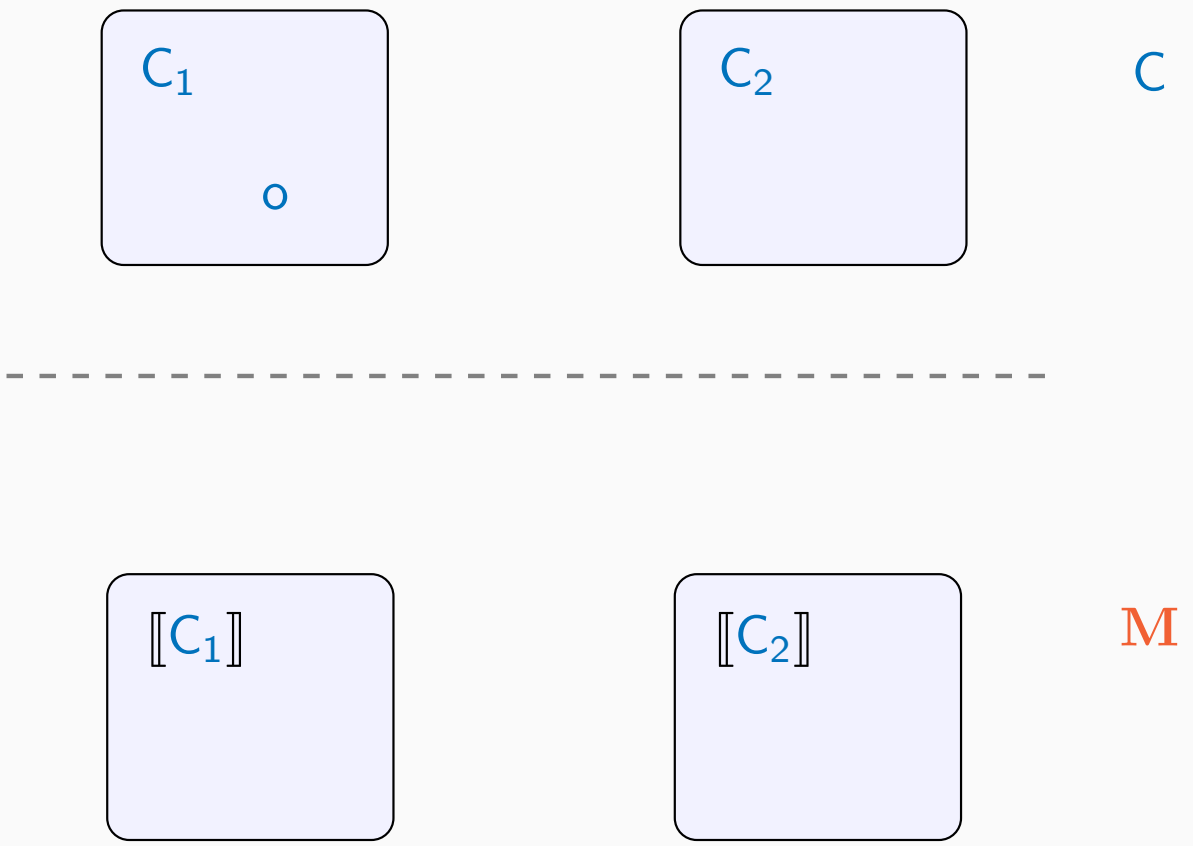
Object Guessing - 1 Component



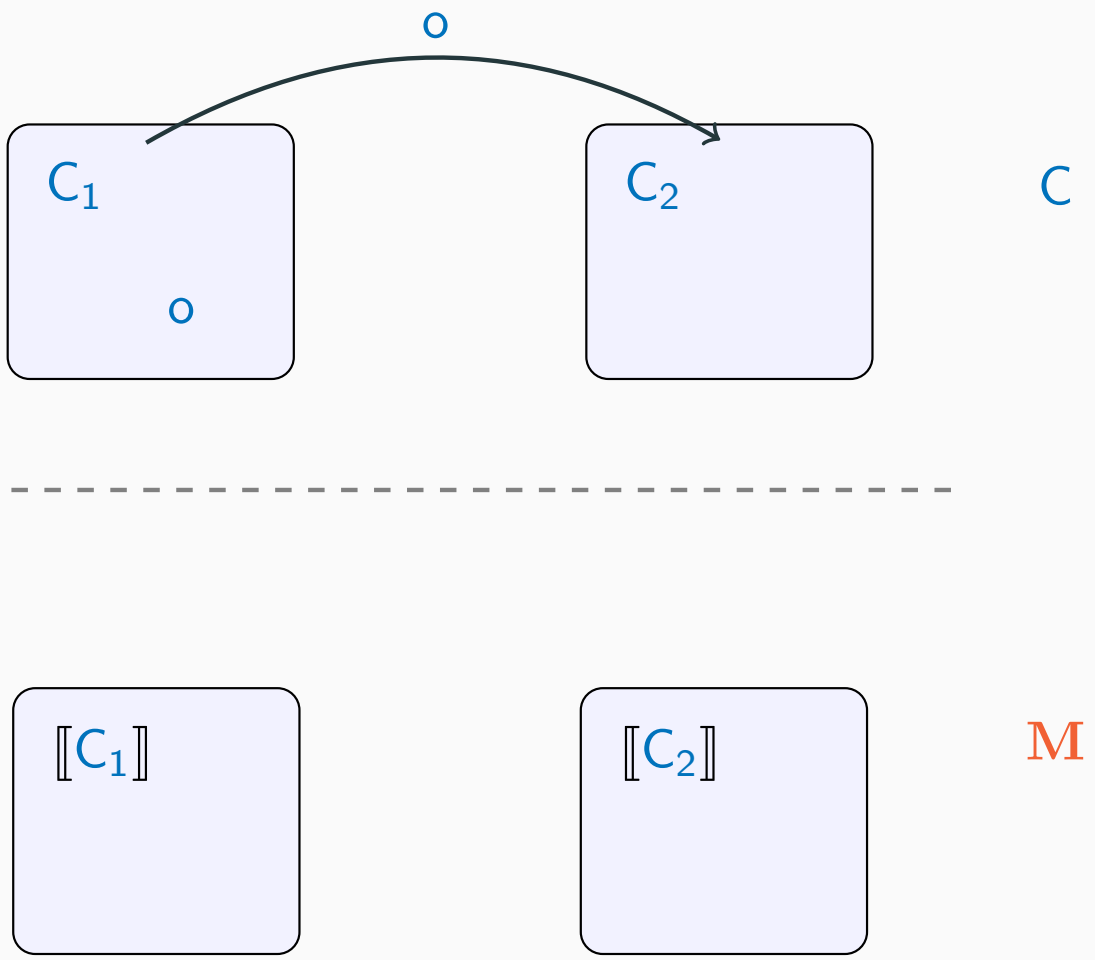
Object Guessing



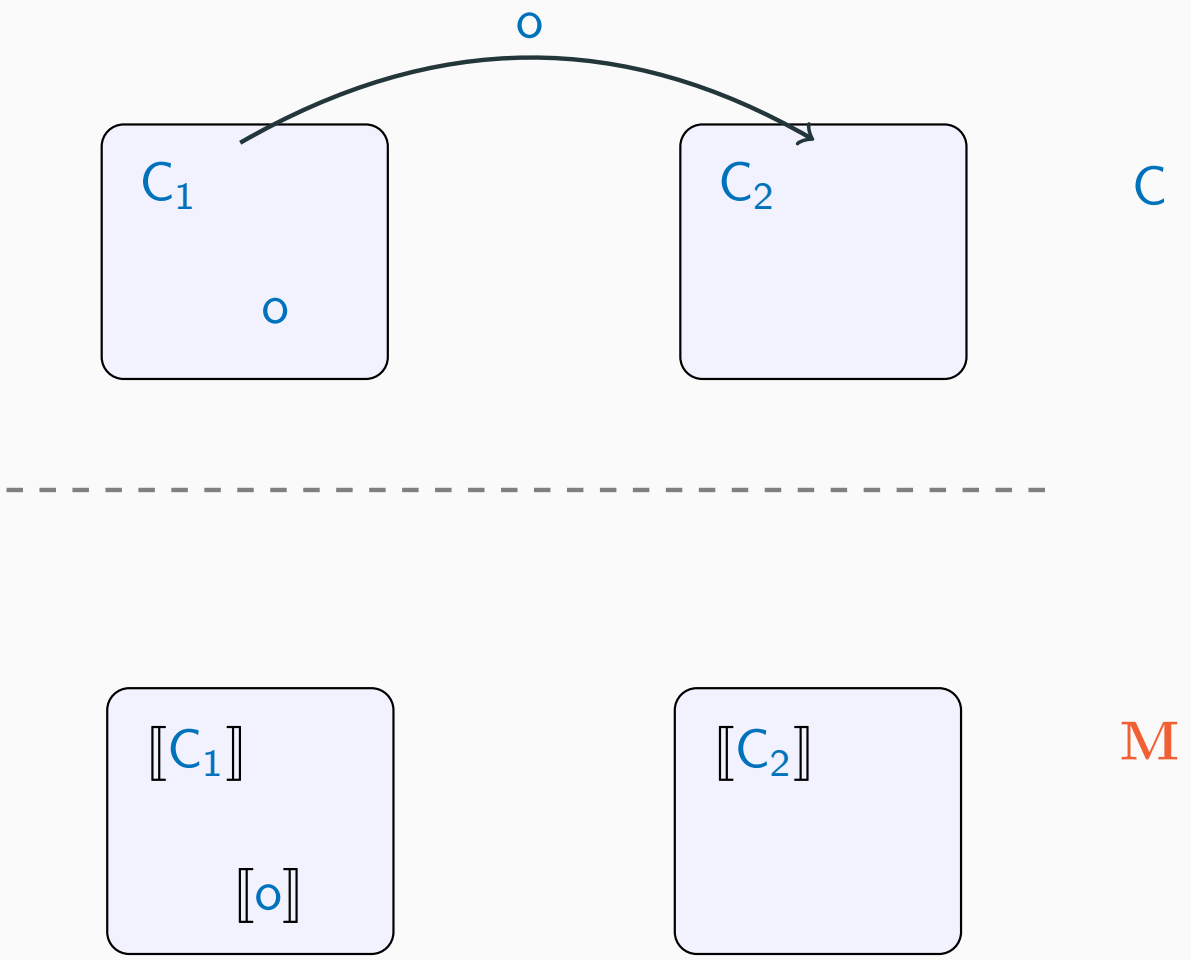
Object Guessing



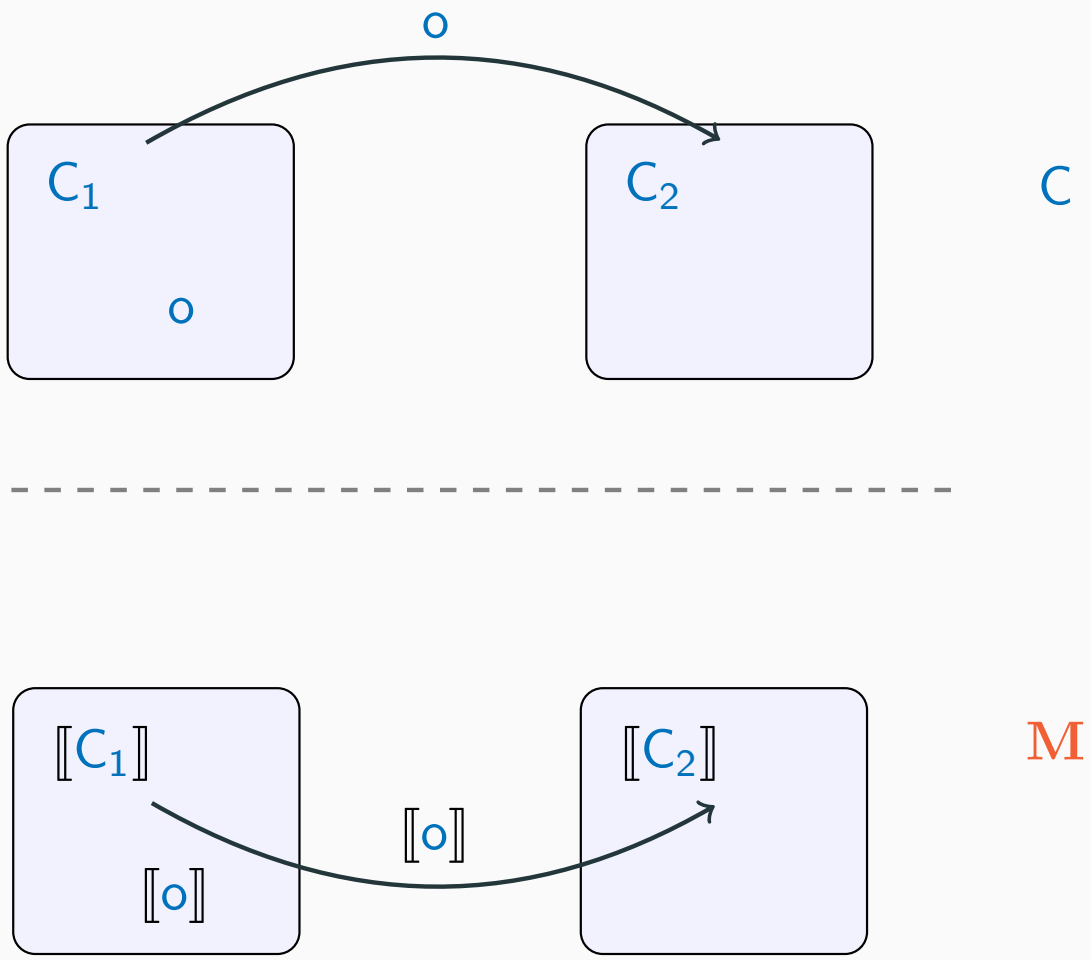
Object Guessing



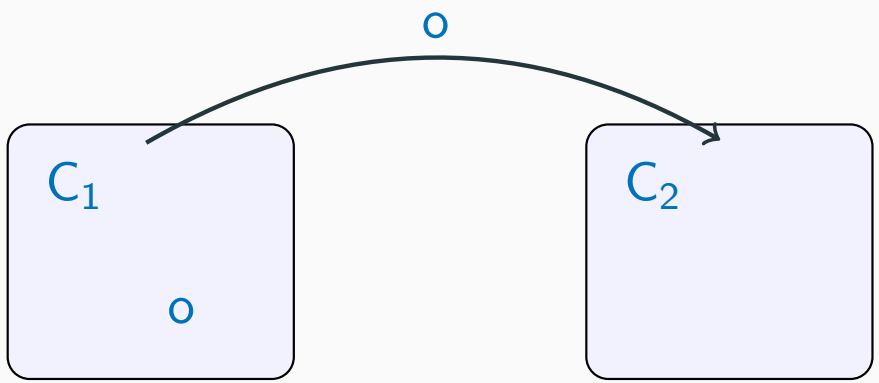
Object Guessing



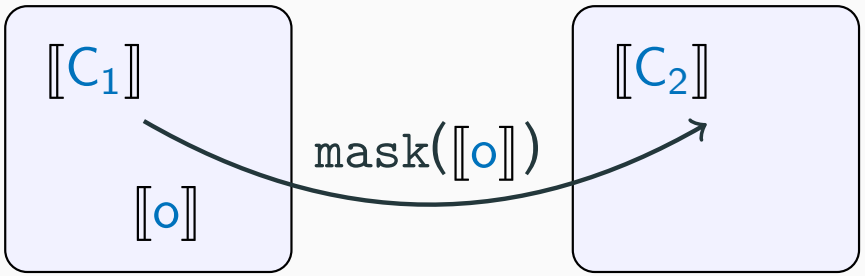
Object Guessing



Object Guessing

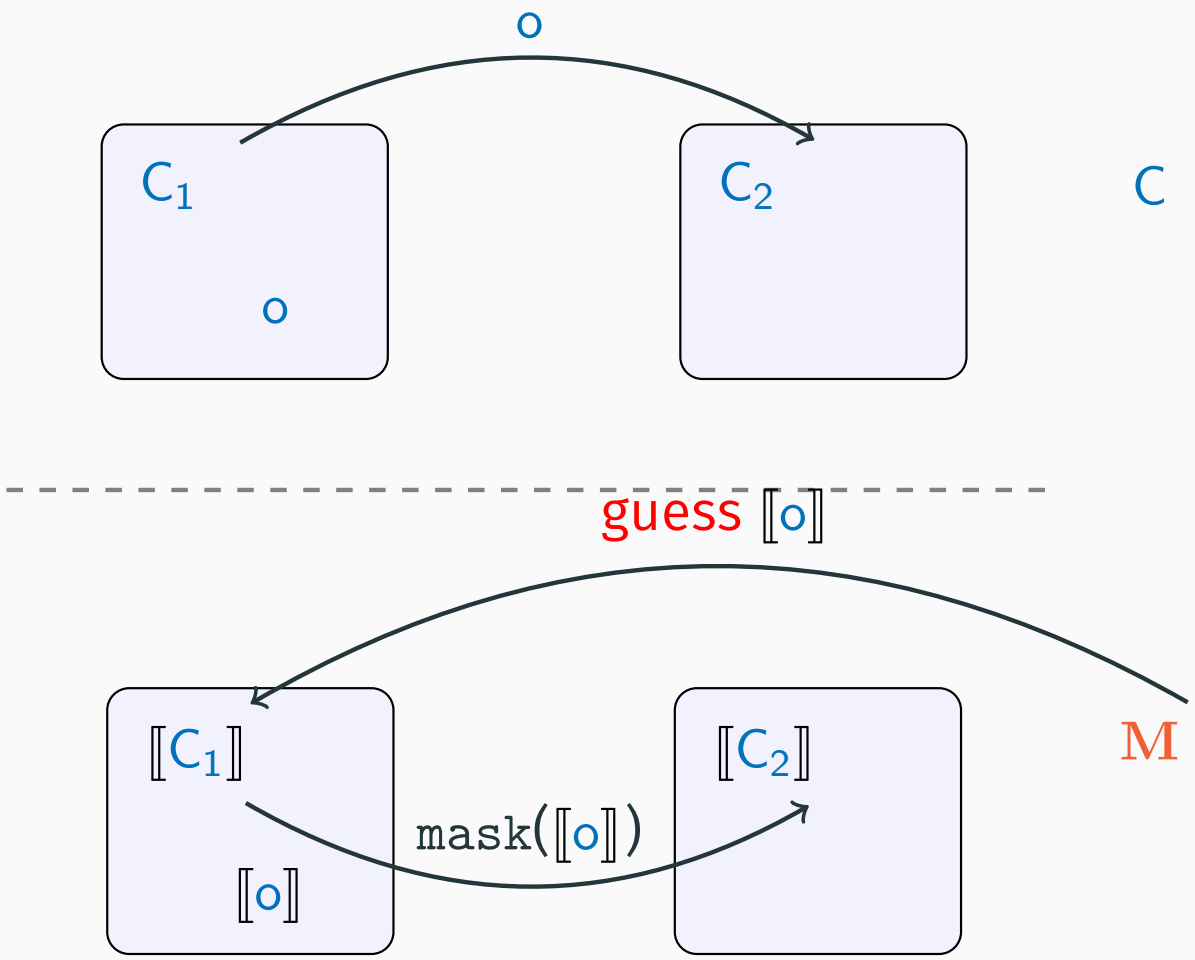


C

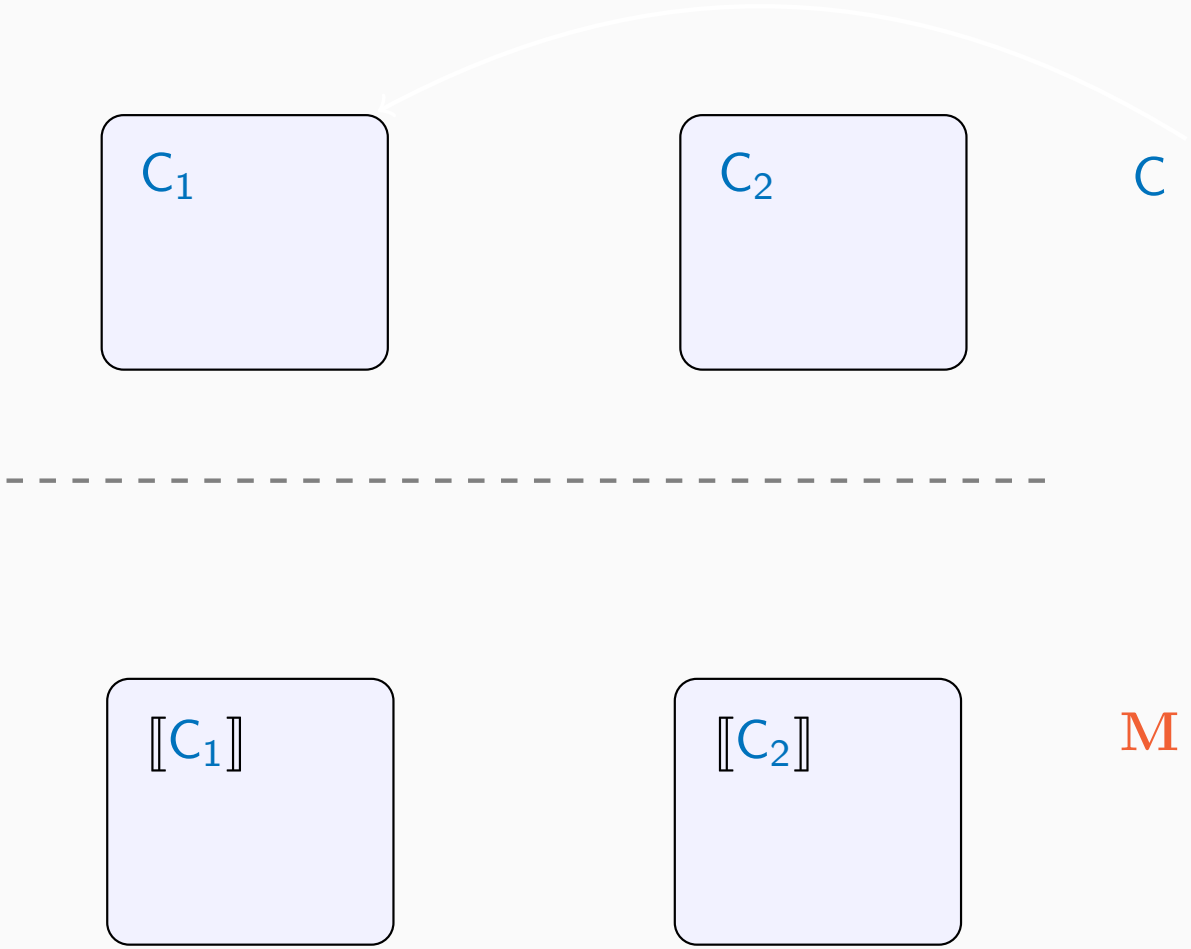


M

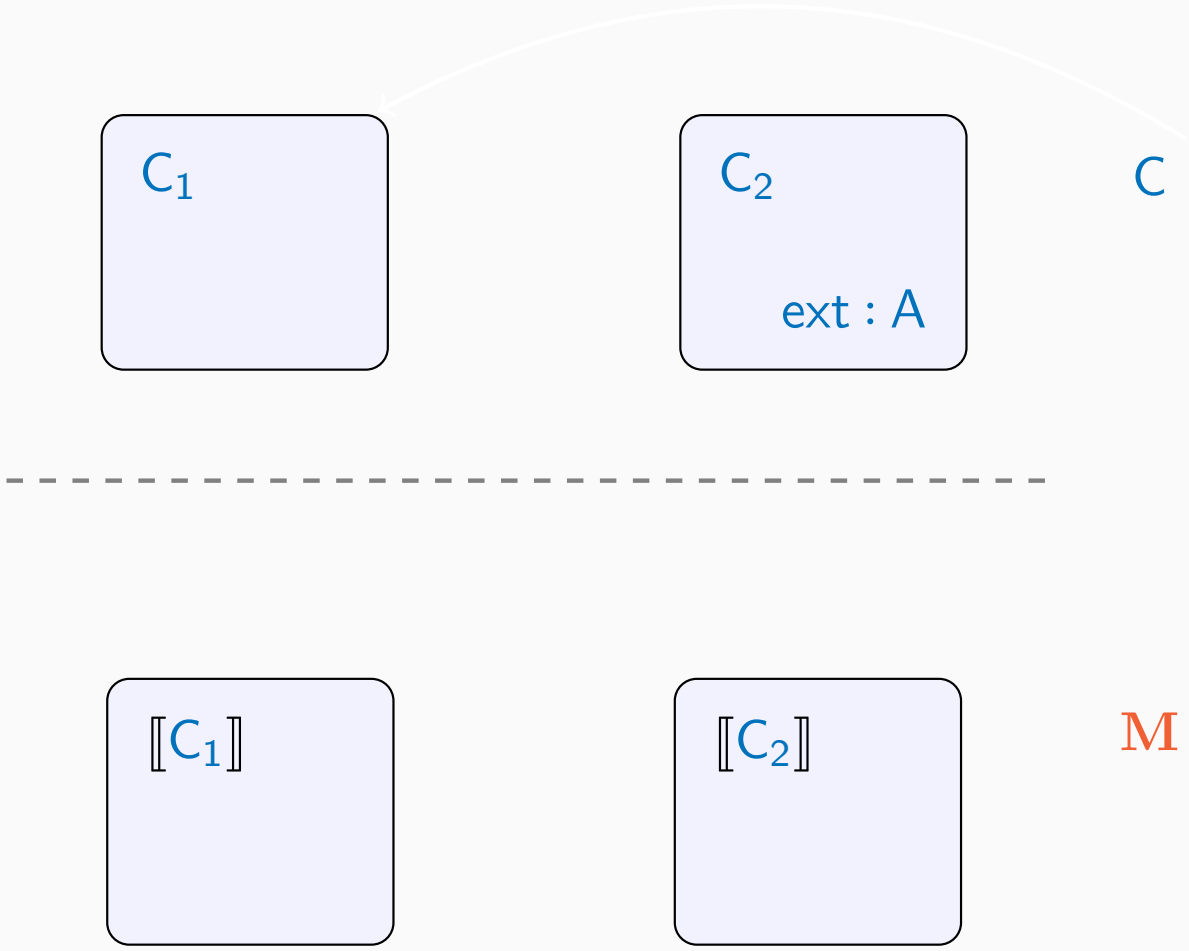
Object Guessing



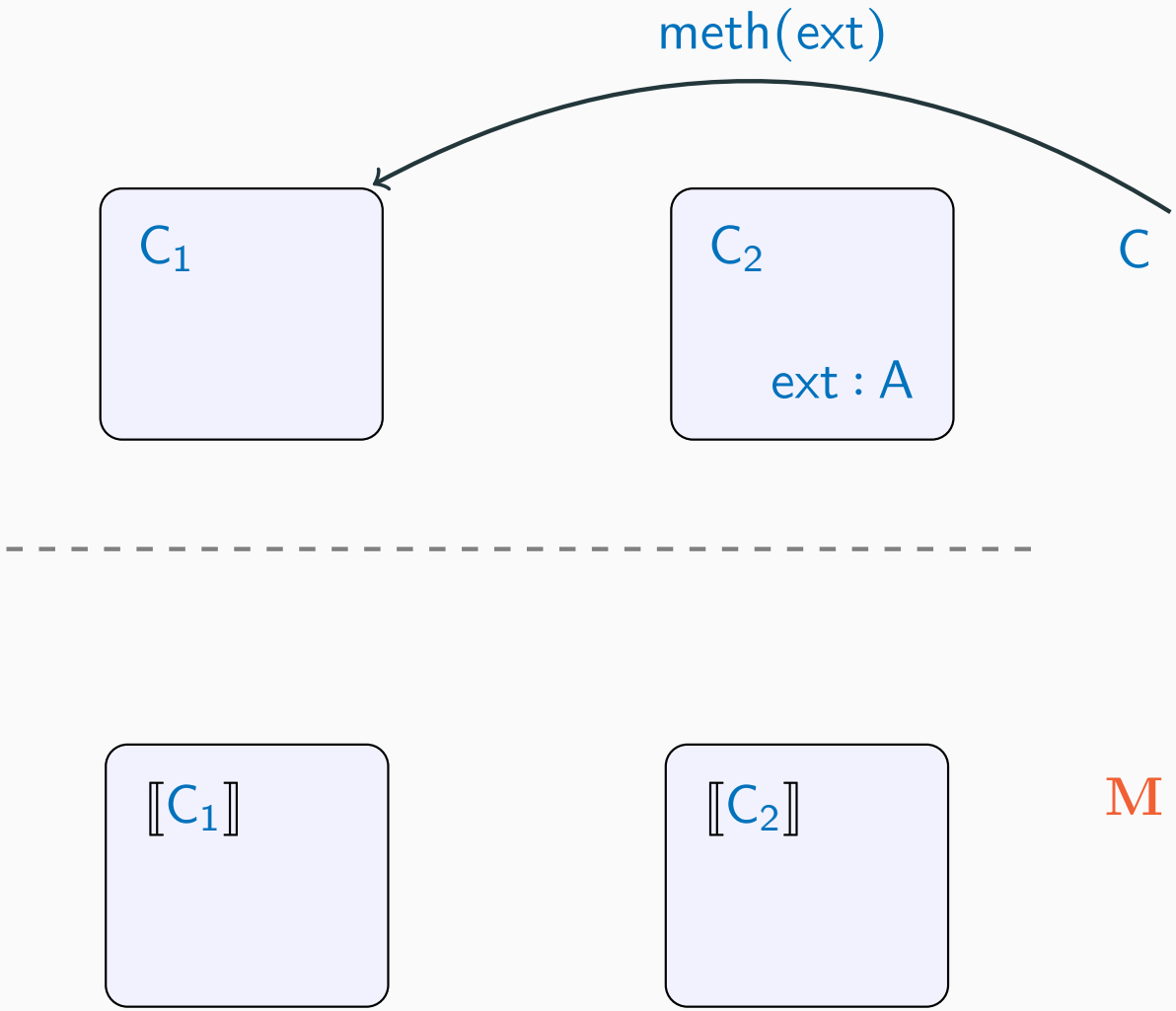
Object Faking



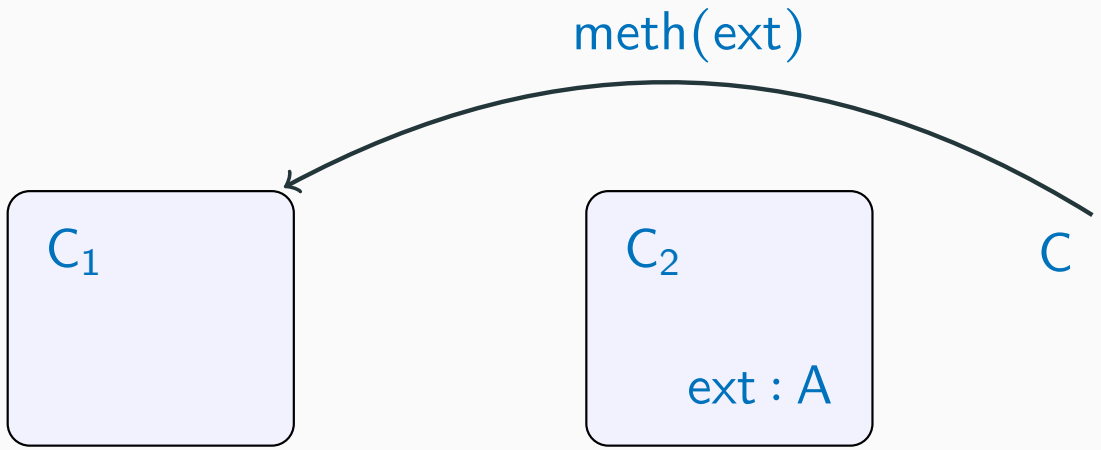
Object Faking



Object Faking

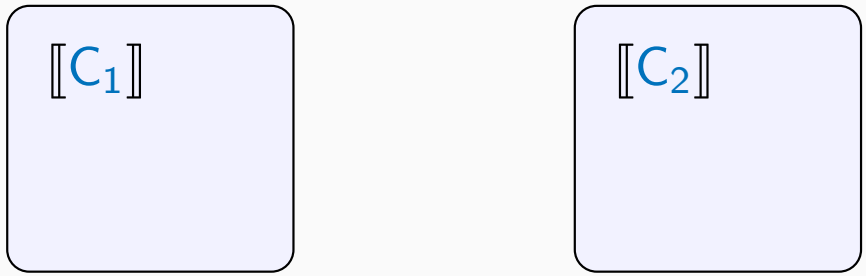


Object Faking



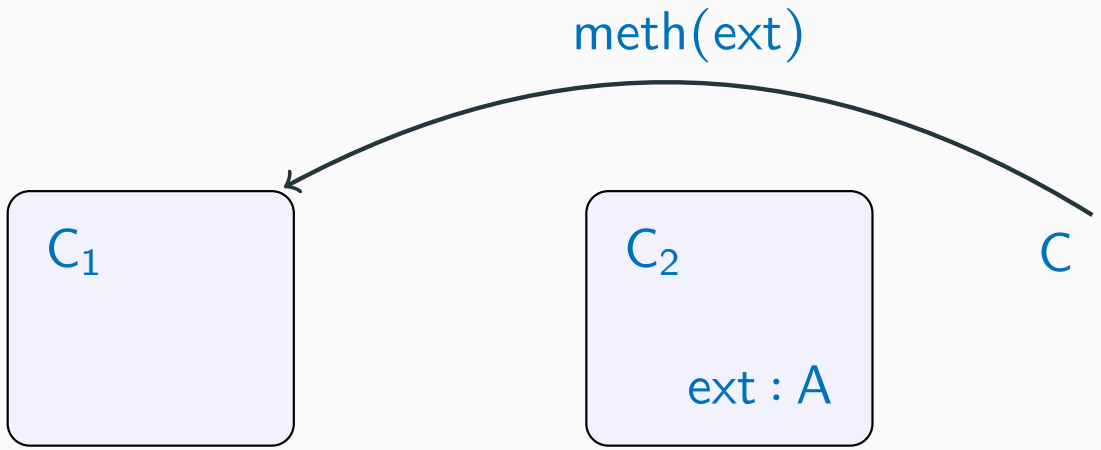
ext = `null`

or obj : A



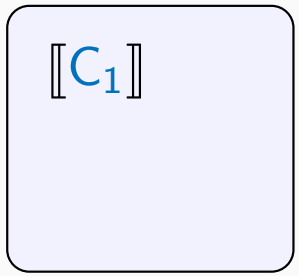
M

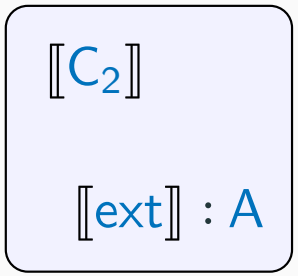
Object Faking



ext = `null`

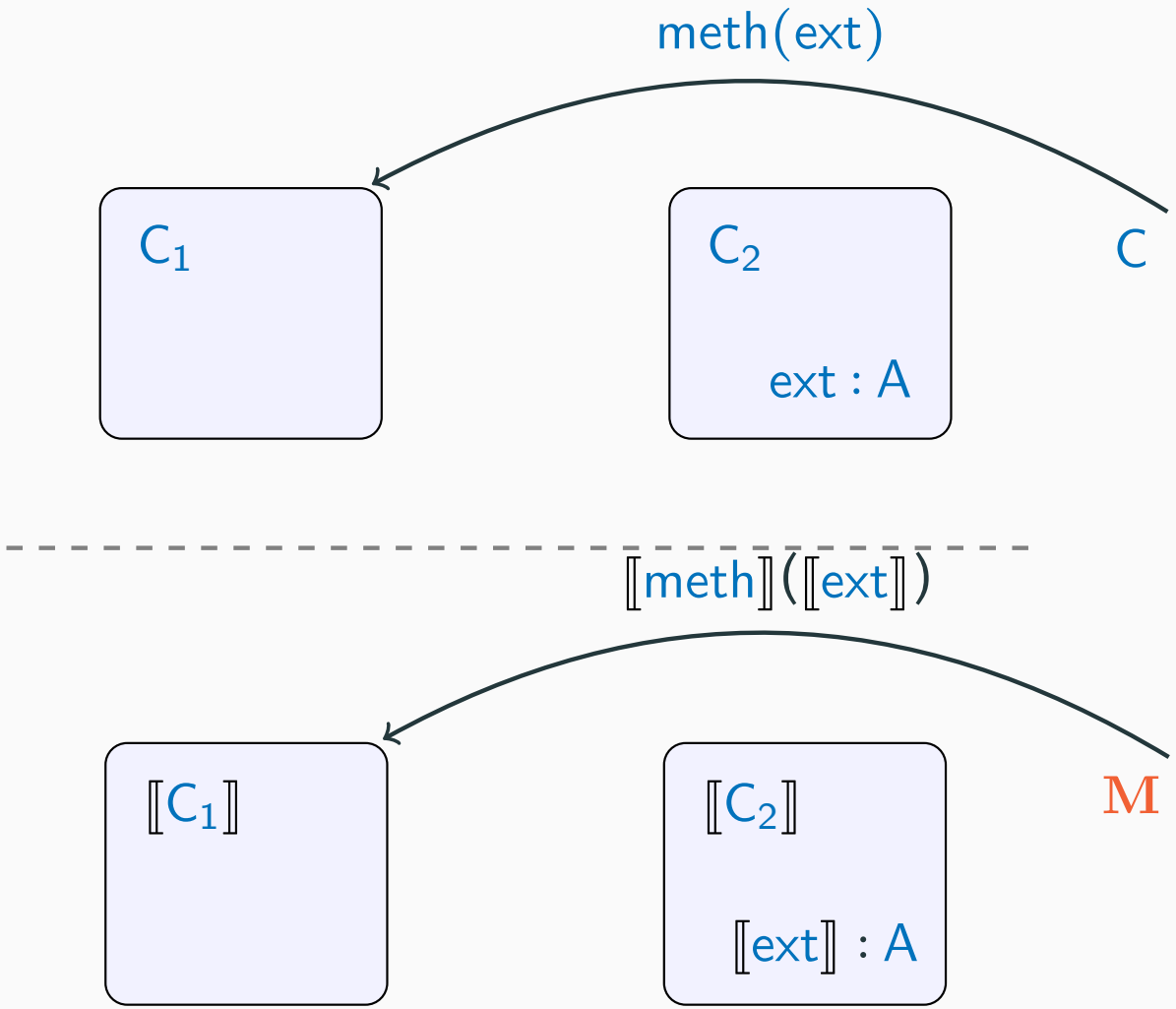
or obj : A





M

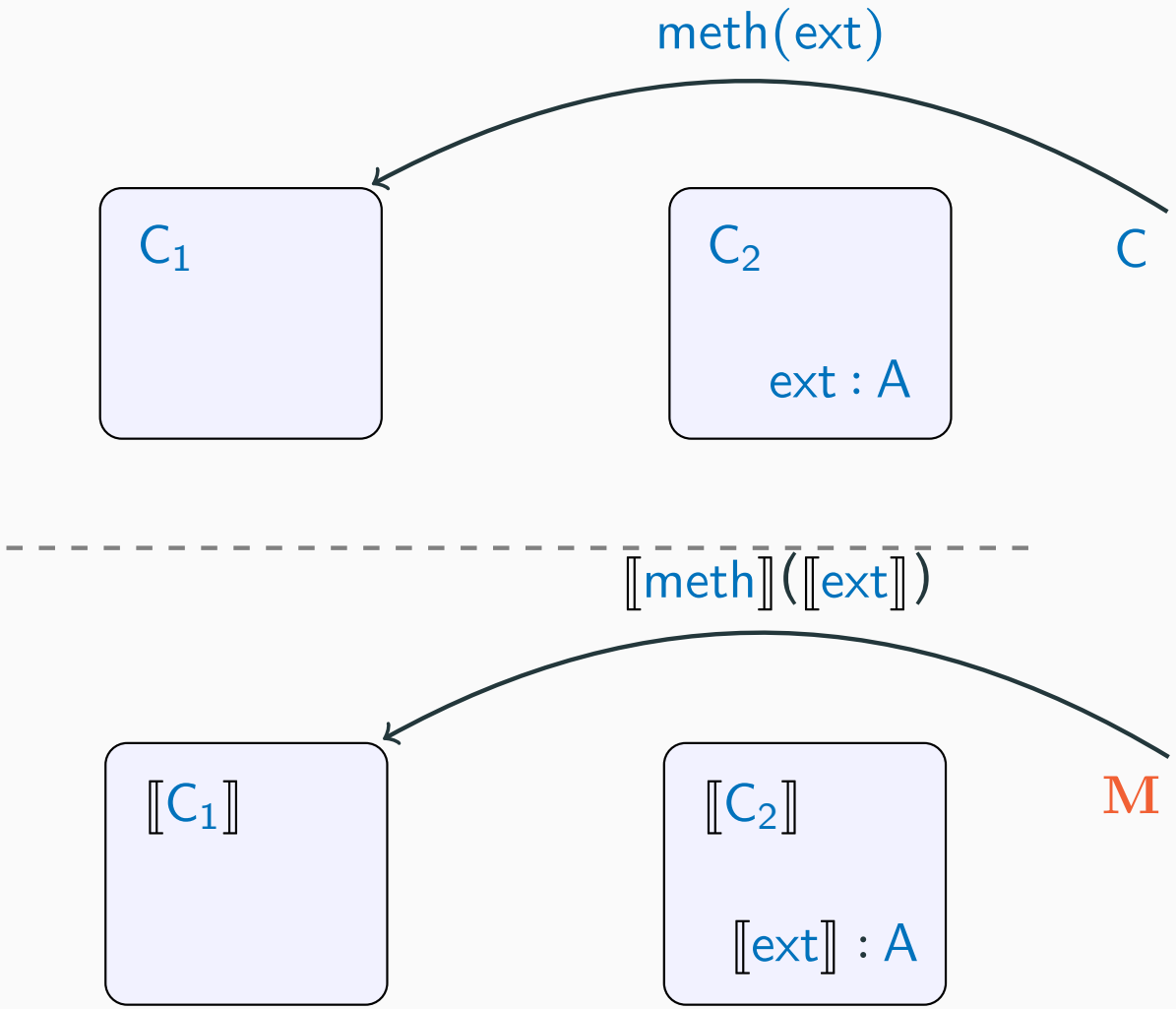
Object Faking



ext = `null`

or obj : A

Object Faking



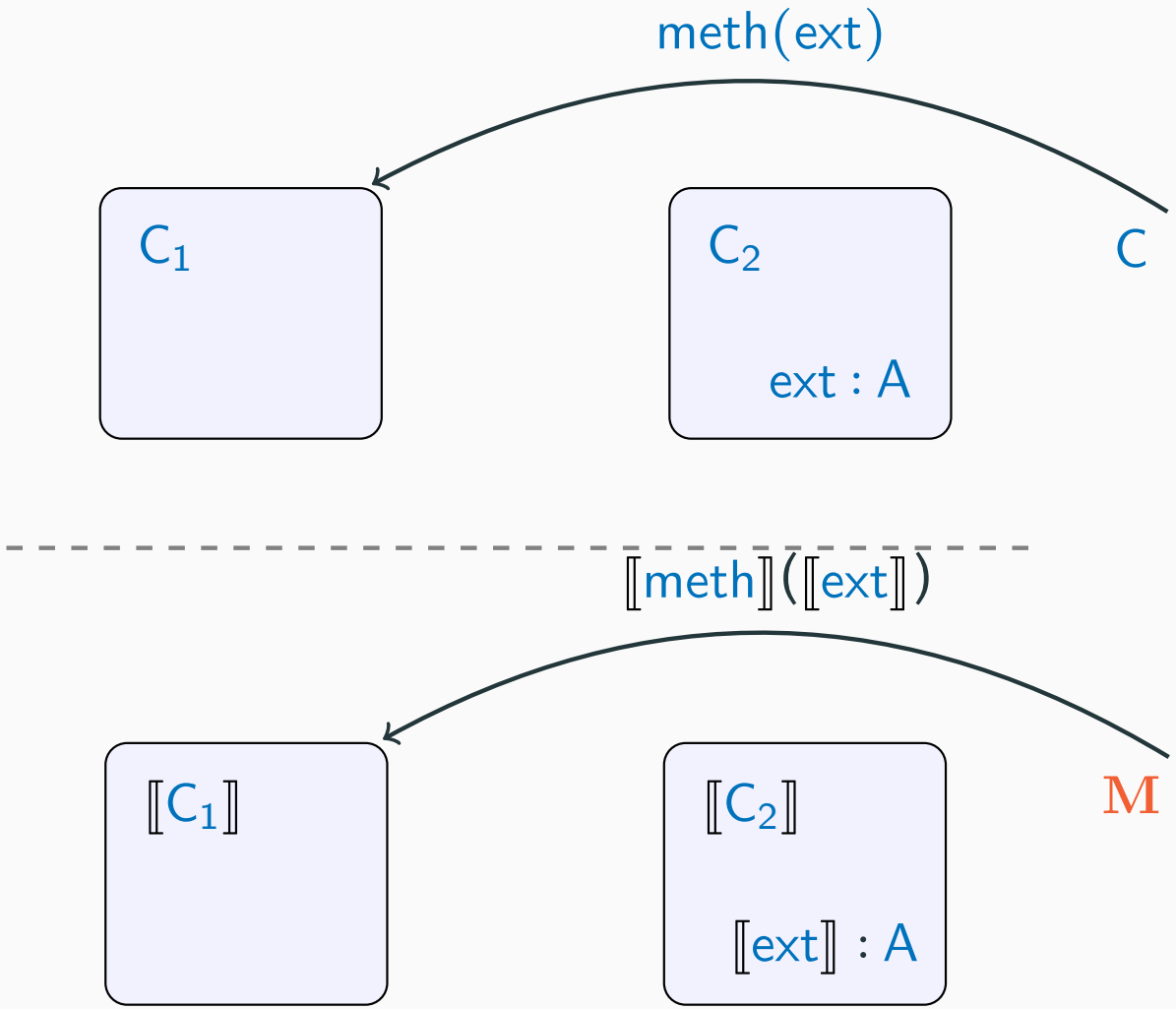
ext = `null`

or obj : A

⟦ext⟧ = ⟦`null`⟧

or obj : A

Object Faking



ext = `null`

or obj : A

⟦ext⟧ = ⟦`null`⟧

or obj : A

or obj : B $\not<:$ A

or garbage

Compiler Structure and Assumptions

- operates on components $\mathcal{C} = \overline{P}$

Compiler Structure and Assumptions

- operates on components $\mathcal{C} = \overline{P}$
- relies on linking table:

$$
\begin{array}{lr}
\textit{symbol tables} & \mathbf{t} ::= \overline{\mathbf{EM}}; \overline{\mathbf{EO}}; \overline{\mathbf{RM}}; \overline{\mathbf{RO}} \\
\textit{exported methods} & \mathbf{EM} ::= \mathsf{m} : \mathsf{M}_\mathsf{t} \mapsto \mathbf{a} \\
\textit{exported objects} & \mathbf{EO} ::= \mathsf{o} : \mathsf{c} \mapsto \mathbf{n} \\
\textit{required methods} & \mathbf{RM} ::= \mathsf{m} : \mathsf{M}_\mathsf{t} \mapsto \iota; \sigma \\
\textit{required objects} & \mathbf{RO} ::= \mathsf{o} : \mathsf{c} \mapsto \sigma
\end{array}
$$

Secure Compiler Structure

Same structure as before plus:

- The System Module Sys
- Different entry/exit point checks
- Secure linker

The System Module Sys

- all calls and returns go through it
  - implements: forwardCall and forwardReturn
  - implements testObj and registerObj
- maintains a global call stack
- registers all objects passed
- relies on caller-callee authentication
  - semantics sets a register to module id on call/return

Entry/exit Point checks

- check that calls comes from Sys
  - relies on caller-callee authentication
- performs dynamic typechecks on arguments
  - reiled on testObj from Sys
- masks and unmasks objects (as before)
- reset flags and registers (as before)

Secure Linker

- creates Sys
- initialises Sys with static global objects

Properties of the Compiler

- FA is not enough: need MFAC

Properties of the Compiler

- FA is not enough: need MFAC

Definition (Modular full-abstraction)

$$\forall \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4.\ \forall \mathbf{P}.[\![\mathcal{C}_2]\!] \simeq_{ctx} \mathbf{P},\ \forall \mathbf{P}'.[\![\mathcal{C}_4]\!] \simeq_{ctx} \mathbf{P}',$$
$$\mathcal{C}_1;\mathcal{C}_2 \simeq_{ctx} \mathcal{C}_3;\mathcal{C}_4 \iff$$
$$\texttt{link}([\![\mathcal{C}_1]\!], \mathbf{P}) \simeq_{ctx} \texttt{link}([\![\mathcal{C}_3]\!], \mathbf{P}').$$

Trace-based Backtranslation

- $\langle\!\langle \mathcal{C}_1, \mathcal{C}_2, \overline{\alpha_1}, \overline{\alpha_2} \rangle\!\rangle = \mathbb{C}$
- $\overline{\alpha_1} \equiv \overline{\alpha}\alpha_1!$
- $\overline{\alpha_2} \equiv \overline{\alpha}\alpha_2!$
- $\alpha_1! \neq \alpha_2!$
- $\overline{\alpha}$ is the *common prefix*
- $\alpha_1!$ and $\alpha_2!$ are the *different actions* at index i.

Skeleton

$\texttt{skeleton}(\mathcal{C}_1, \mathcal{C}_2) : \mathbb{C}$

- implements classes and objects that $\mathcal{C}_1$ and $\mathcal{C}_2$ import
- creates helper functions and objects:
  - tables where all globally-known objects are stored
  - a variable to keep track of the action being emulated

Common Prefix

$\texttt{emulate}\,(\overline{\alpha}, \mathbf{t}) : \mathsf{e@M}$

- $\mathbf{t}$ = linking table
- $\mathbf{call\ a\ \overline{w}?}$: call method m compiled at address $\mathbf{a}$ with $\langle\!\langle \overline{\mathbf{w}} \rangle\!\rangle$
  lookup type of $\langle\!\langle \mathbf{w} \rangle\!\rangle$ in $\mathbf{t}$
  $\langle\!\langle \mathbf{w} \rangle\!\rangle$ is trivial if $\mathbf{w}$ is of ground type
  if $\mathbf{w}$ is an object, it is found in a table with its type and $\mathbf{id}$.
- $\mathbf{ret\ a\ w?}$
  Return the backtranslation of $\mathbf{w}$
- $\mathbf{call\ a\ \overline{w}}$ and $\mathbf{ret\ a\ w!}$
  Update the internal state of $\mathbb{C}$
  E.g., add all newly allocated objects received via $\overline{\mathbf{w}}$ or $\mathbf{w}$ to the table

Differentiator

$\texttt{diff}\,(\alpha_1!, \alpha_2!, i) : \mathsf{e@M}, \mathsf{e@M}$

- case analysis on all differences:

Differentiator

$\texttt{diff}\,(\alpha_1!, \alpha_2!, i) : \mathsf{e@M}, \mathsf{e@M}$

- case analysis on all differences:
  - different length

Differentiator

$\texttt{diff}\,(\alpha_1!, \alpha_2!, i) : \mathsf{e@M}, \mathsf{e@M}$

- case analysis on all differences:
  - different length
  - different actions

Differentiator

$\texttt{diff}\,(\alpha_1!, \alpha_2!, i) : \mathsf{e@M}, \mathsf{e@M}$

- case analysis on all differences:
  - different length
  - different actions
  - different method called

Differentiator

$\texttt{diff}\,(\alpha_1!, \alpha_2!, i) : \mathsf{e@M}, \mathsf{e@M}$

- case analysis on all differences:
  - different length
  - different actions
  - different method called
  - different return

Differentiator

$\texttt{diff}\,(\alpha_1!, \alpha_2!, i) : \mathsf{e@M}, \mathsf{e@M}$

- case analysis on all differences:
  - different length
  - different actions
  - different method called
  - different return
  - different current object

Differentiator

$\texttt{diff}\,(\alpha_1!, \alpha_2!, i) : \mathsf{e@M}, \mathsf{e@M}$

- case analysis on all differences:
  - different length
  - different actions
  - different method called
  - different return
  - different current object
  - different argument