Multi-Module Fully-Abstract Compilation

Marco Patrignani[1] Dominique Devriese[1] Frank Piessens[1]

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

13 July 2015

Outline

Goal

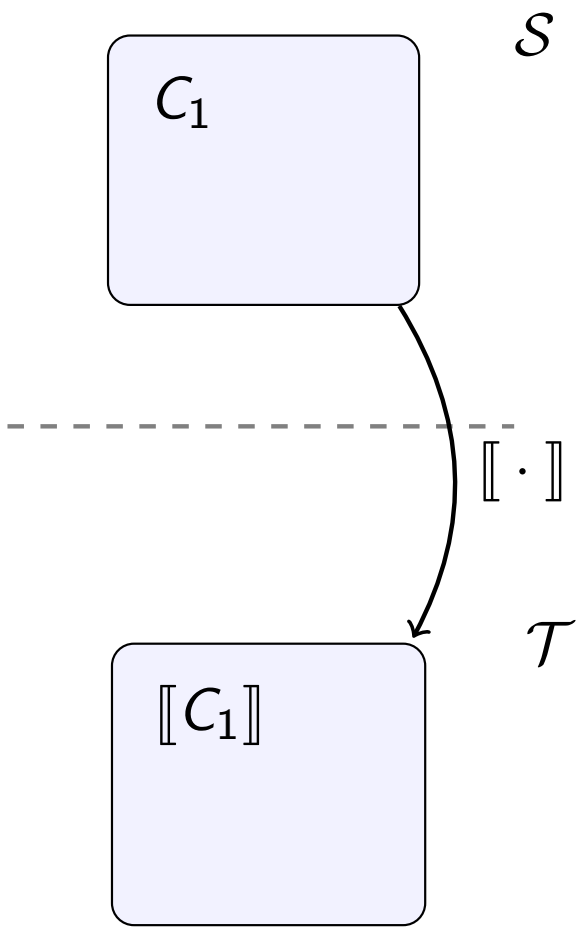
- see secure compilation failures due to linking

Goal

- see secure compilation failures due to linking
- see solutions to them (for PMA & Capability machines)

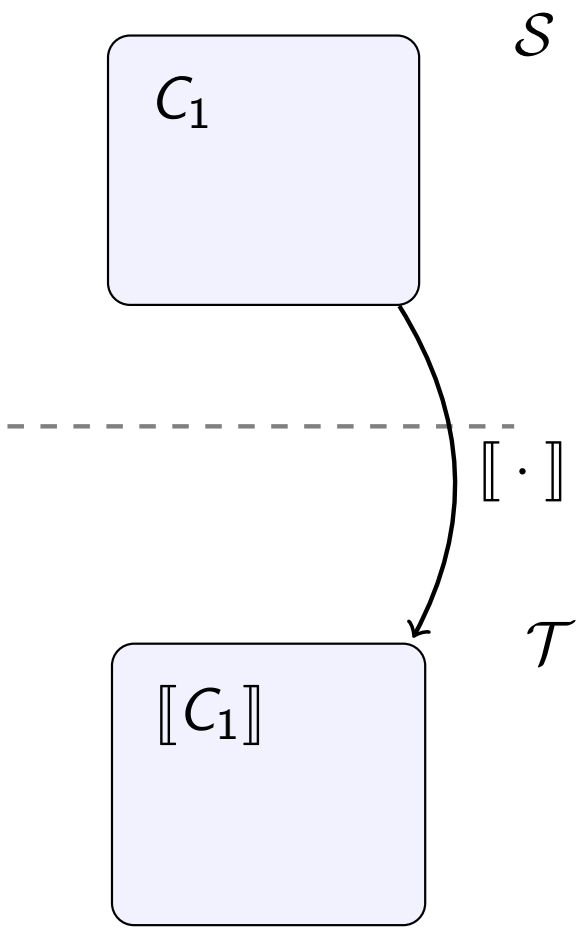
What is a Secure Compiler?

- a compiler is a function from source to target **components**



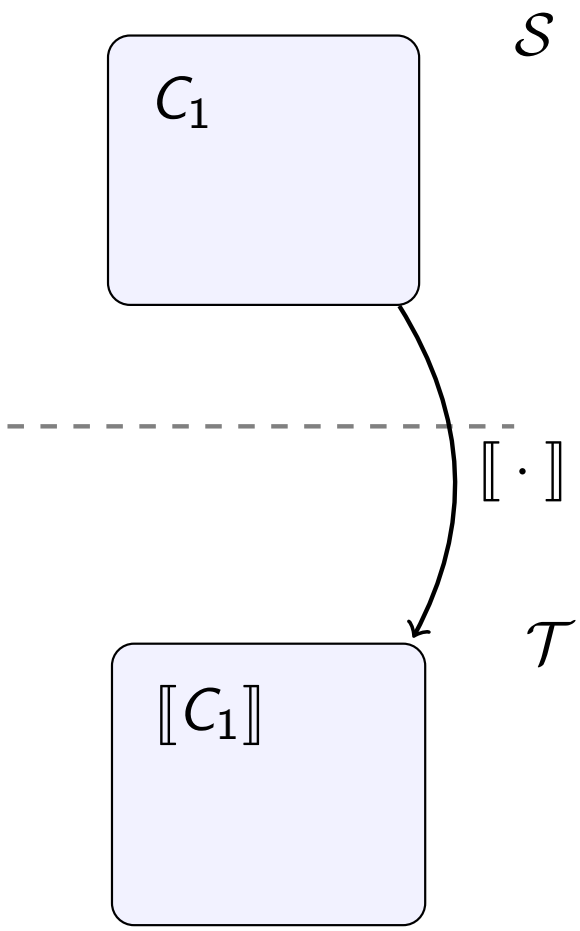
What is a Secure Compiler?

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



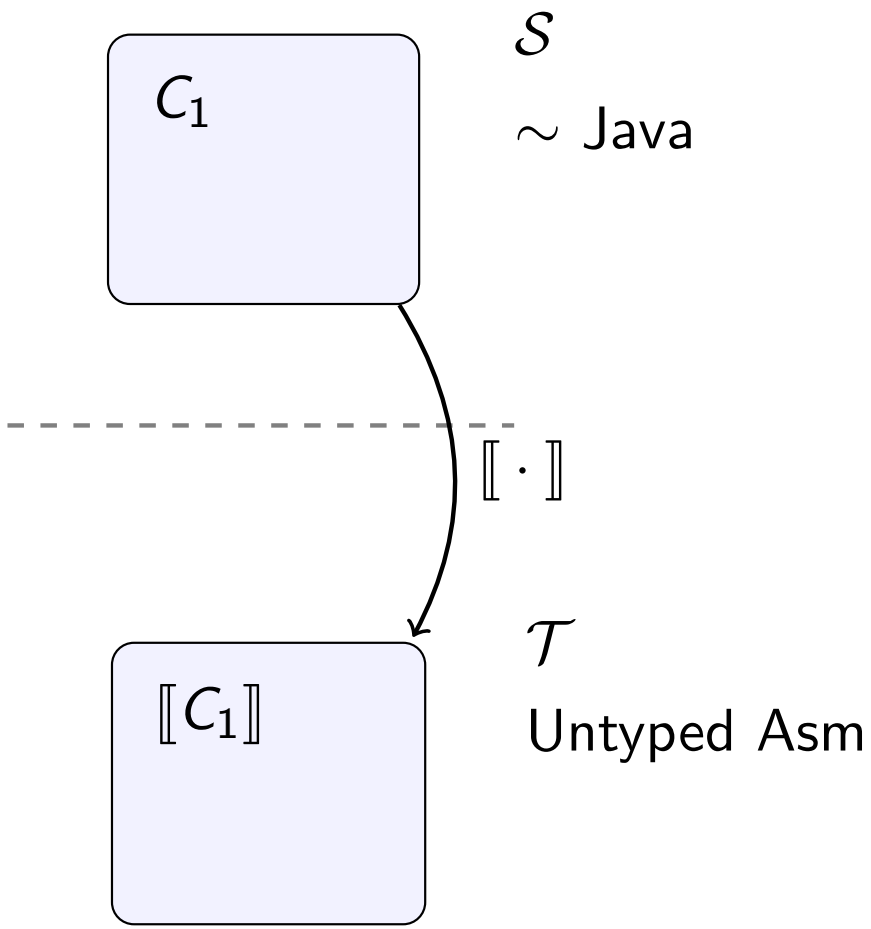
What is a Secure Compiler?

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



What is a Secure Compiler?

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



Fully Abstract Compilation

Fully abstract compilers preserve (and reflect) source-level behaviour in compiled components

Fully Abstract Compilation

Fully abstract compilers preserve (and reflect) source-level behaviour in compiled components

- preserve security properties

Fully Abstract Compilation

Fully abstract compilers preserve (and reflect) source-level behaviour in compiled components

- preserve security properties
- protect against code injection attacks

Fully Abstract Compilation

Fully abstract compilers preserve (and reflect) source-level behaviour in compiled components

- preserve security properties
- protect against code injection attacks
- enable source-level reasoning

Fully Abstract Compilation Formally

$$\forall C_1, C_2 \in \mathcal{S},$$

Fully Abstract Compilation Formally

$$\forall C_1, C_2 \in \mathcal{S}, \quad C_1 \simeq^{\mathcal{S}} C_2$$

Fully Abstract Compilation Formally

$$\forall C_1, C_2 \in \mathcal{S}, \quad C_1 \simeq^{\mathcal{S}} C_2 \iff [\![C_1]\!] \simeq^{\mathcal{T}} [\![C_2]\!]$$

Contextual equivalence

- $\simeq$ denotes behavioural equivalence (program equi-termination or equi-divergence)

Contextual equivalence

- $\simeq$ denotes behavioural equivalence (program equi-termination or equi-divergence)
- $C_1 \simeq^{\mathcal{L}} C_2 \triangleq \forall \mathbb{C},\ \mathbb{C}[C_1] \Uparrow \iff \mathbb{C}[C_2] \Uparrow$

Contextual equivalence

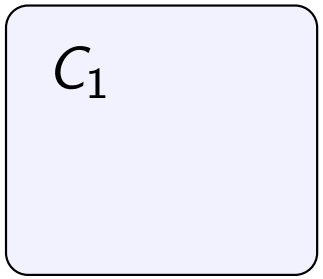
- $\simeq$ denotes behavioural equivalence (program equi-termination or equi-divergence)
- $C_1 \simeq^{\mathcal{L}} C_2 \triangleq \forall \mathbb{C},\ \mathbb{C}[C_1] \Uparrow \iff \mathbb{C}[C_2] \Uparrow$
- $\mathbb{C}$ models an attacker

Contextual equivalence

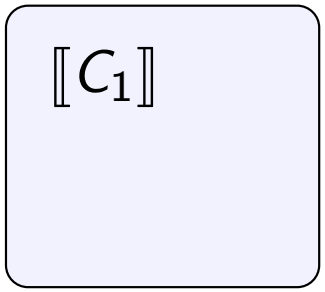
- $\simeq$ denotes behavioural equivalence (program equi-termination or equi-divergence)
- $C_1 \simeq^{\mathcal{L}} C_2 \triangleq \forall \mathbb{C},\ \mathbb{C}[C_1] \Uparrow \iff \mathbb{C}[C_2] \Uparrow$
- $\mathbb{C}$ models an attacker
- $\simeq$ implies security properties e.g., confidentiality, integrity, etc

Outline

Compiler INsecurity

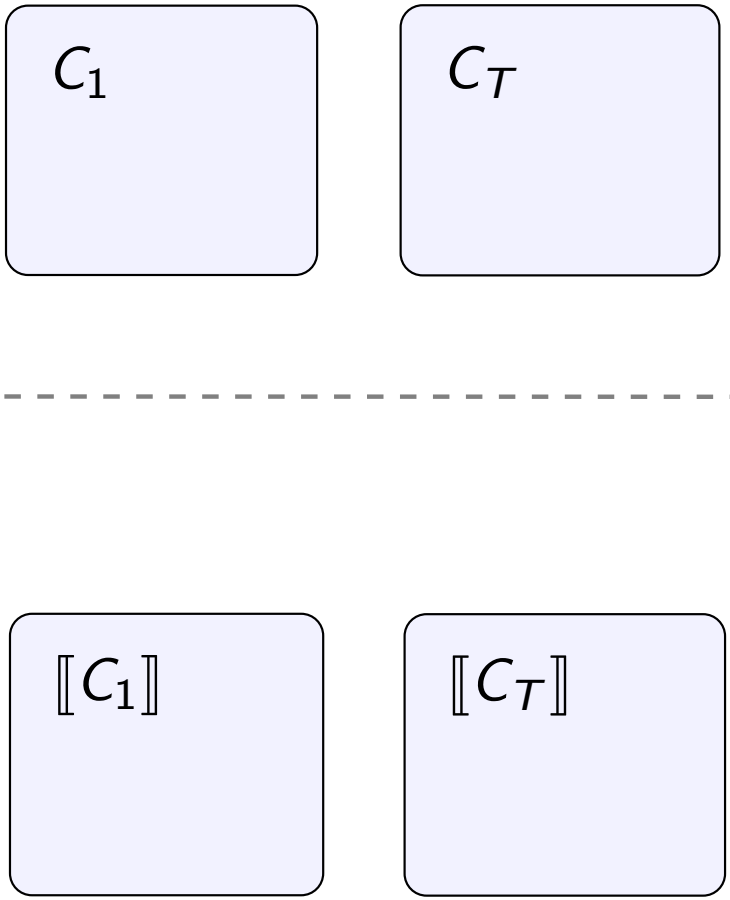


- compiler full abstraction is often studied in simple settings



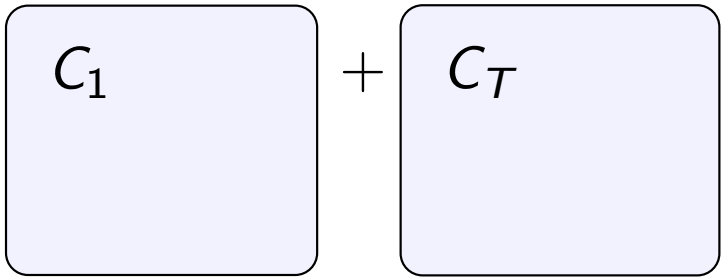
Compiler INsecurity

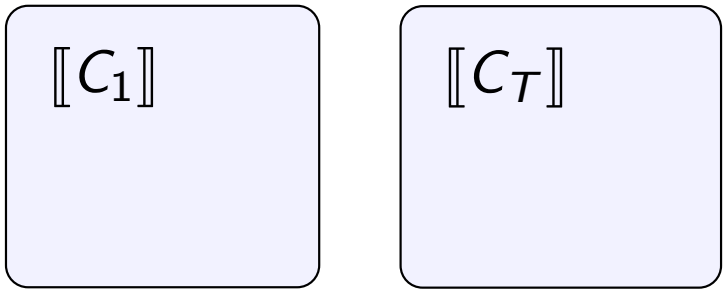
- compiler full abstraction is often studied in simple settings
- existing fully abstractly compiled programs are not linkable



Compiler INsecurity

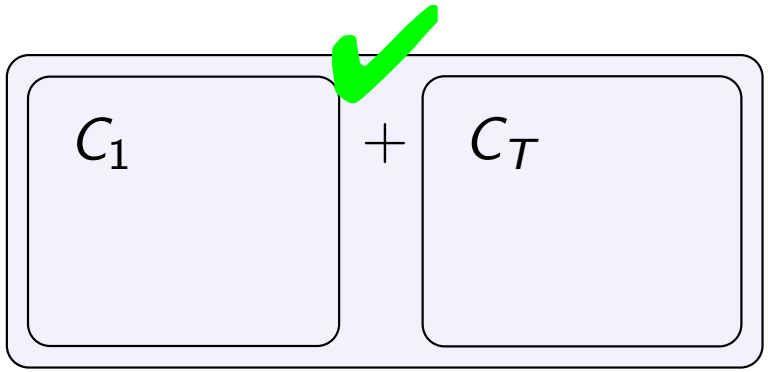
- compiler full abstraction is often studied in simple settings
- existing fully abstractly compiled programs are not linkable

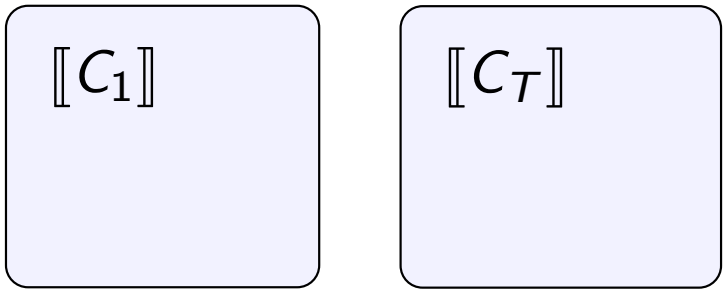




Compiler INsecurity

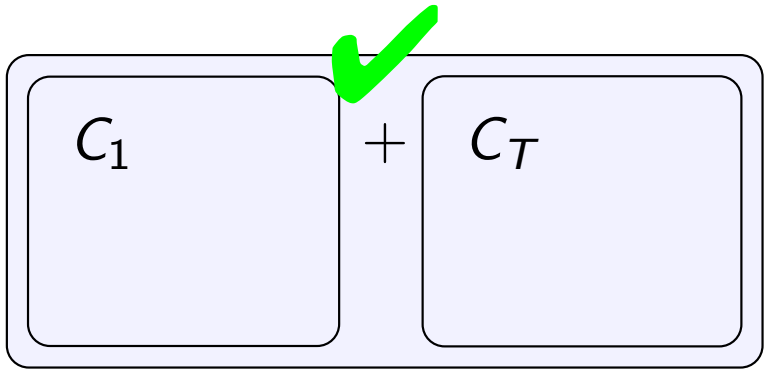
- compiler full abstraction is often studied in simple settings
- existing fully abstractly compiled programs are not linkable

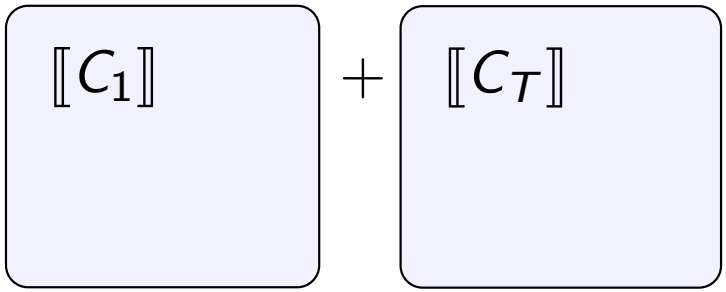




Compiler INsecurity

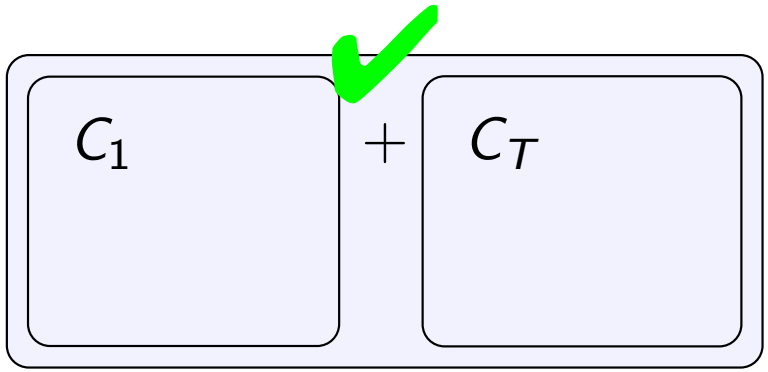
- compiler full abstraction is often studied in simple settings
- existing fully abstractly compiled programs are not linkable

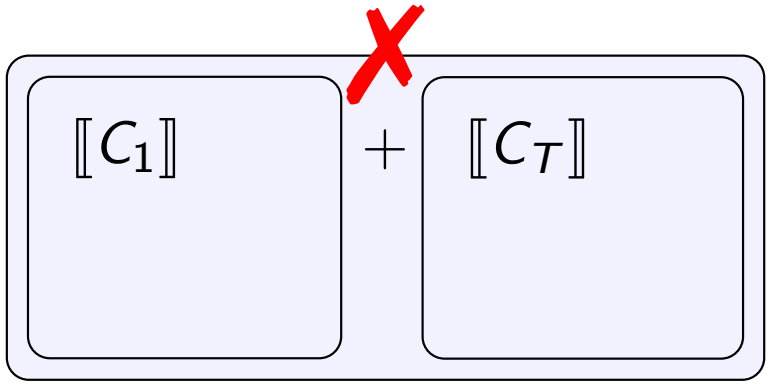




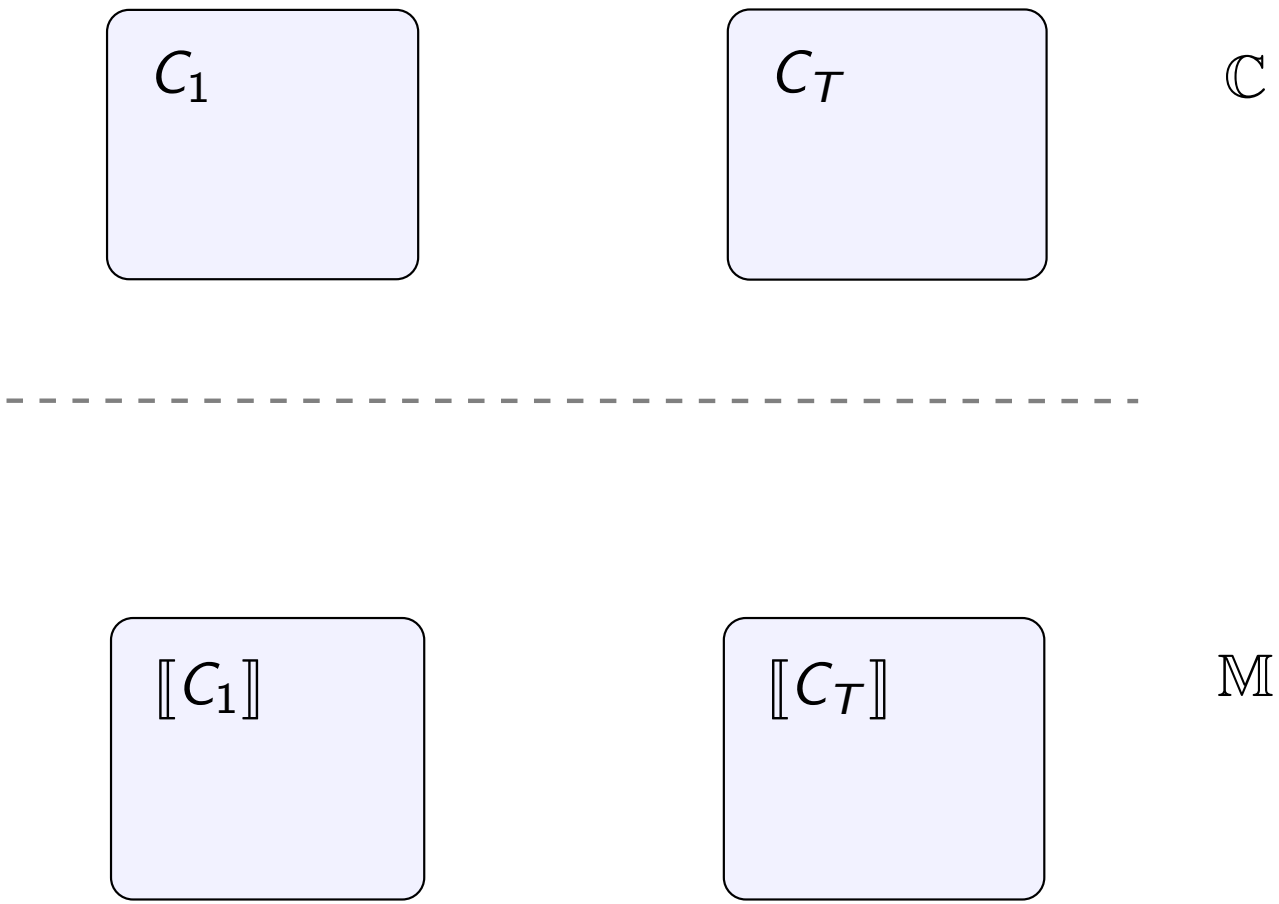
Compiler INsecurity

- compiler full abstraction is often studied in simple settings
- existing fully abstractly compiled programs are not linkable

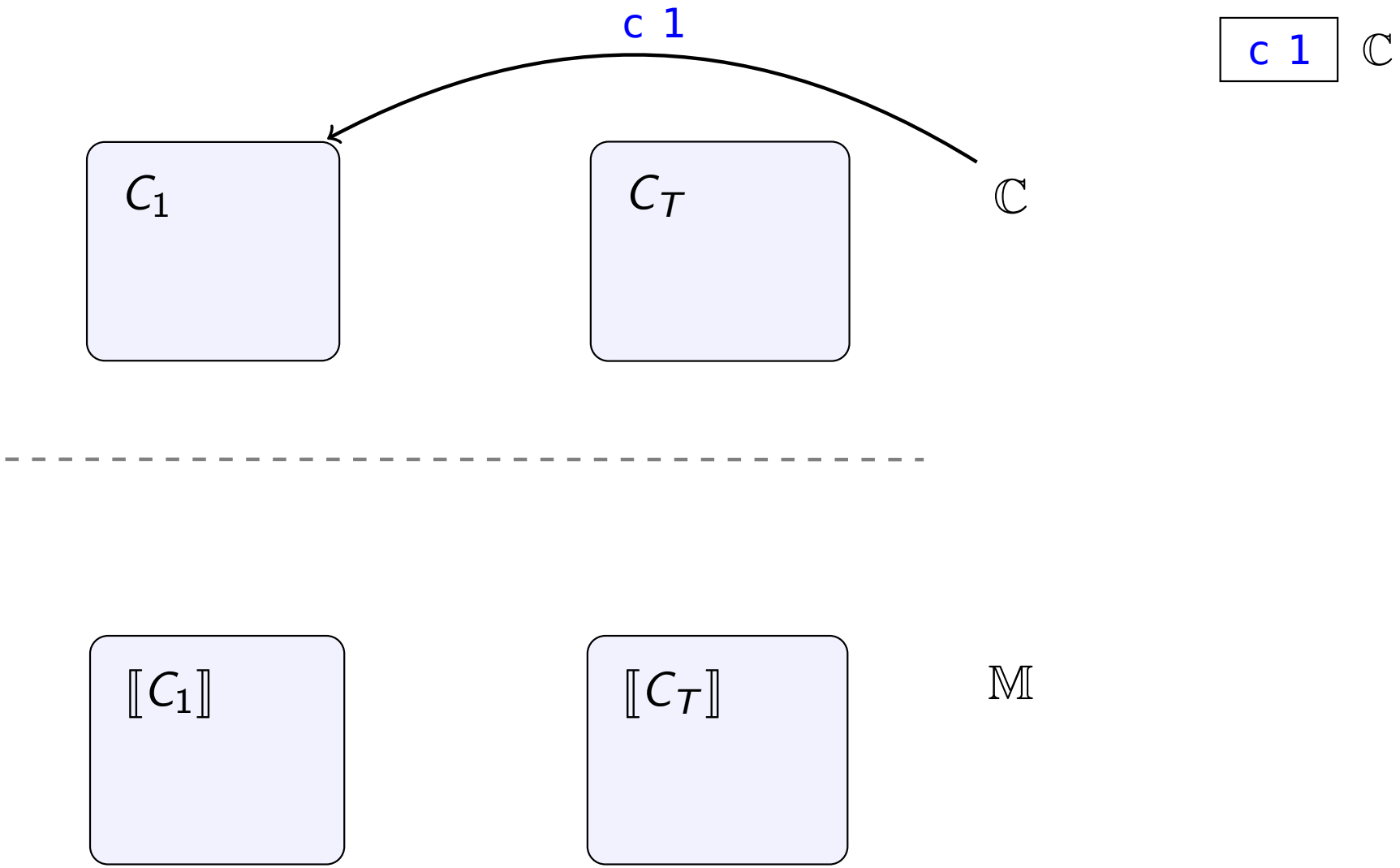




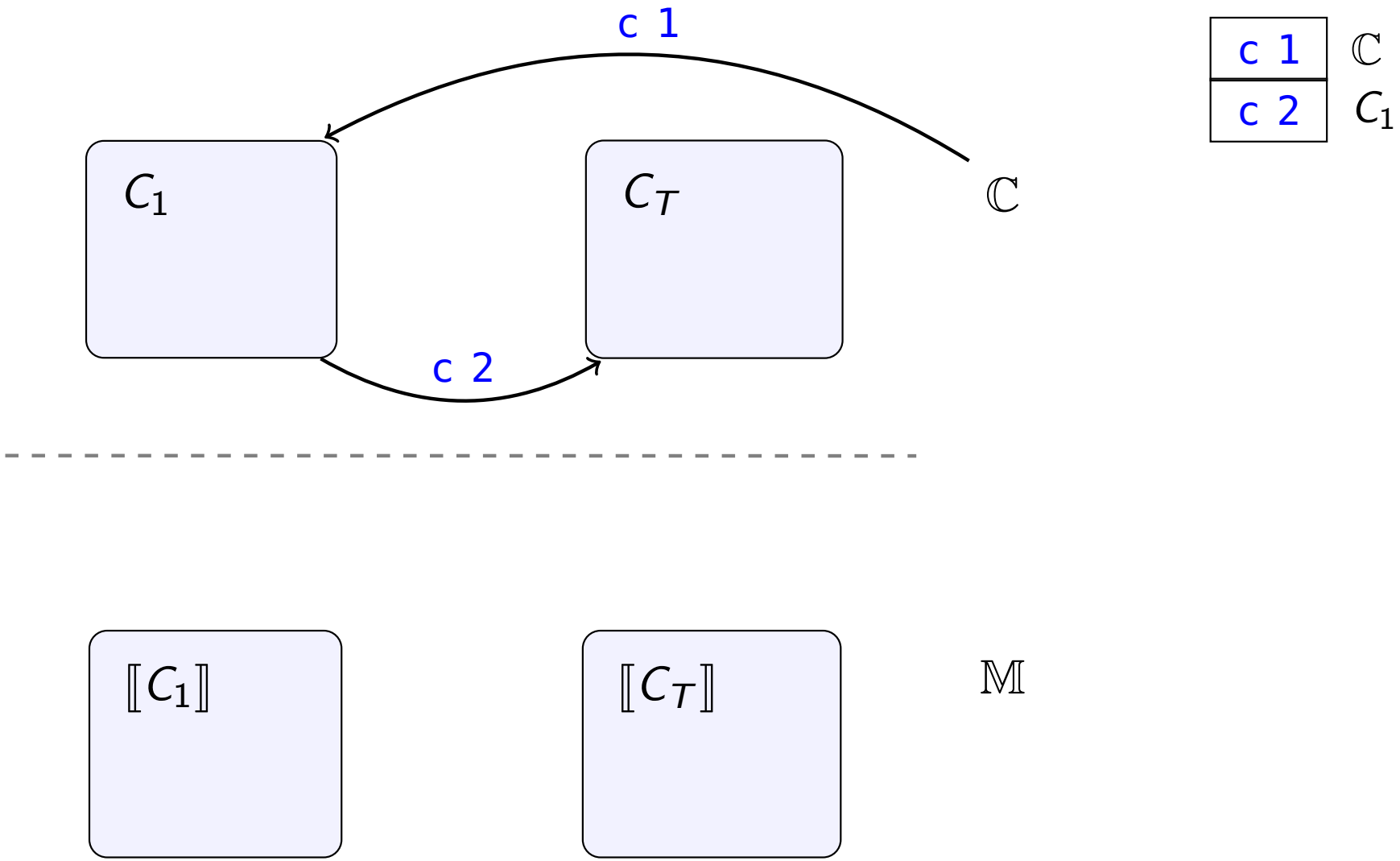
Call Stack Shortcutting



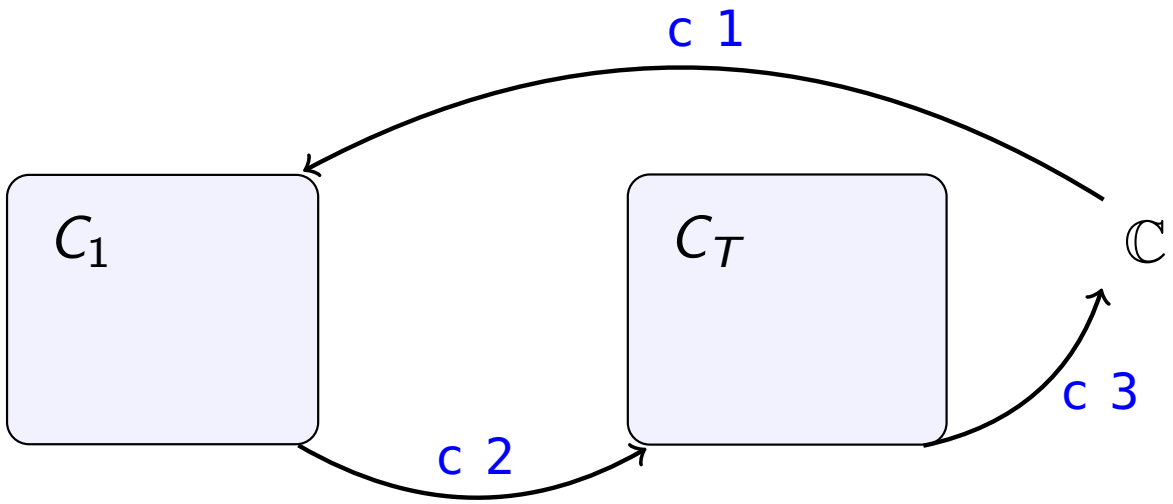
Call Stack Shortcutting

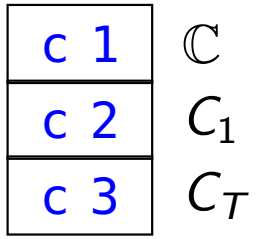


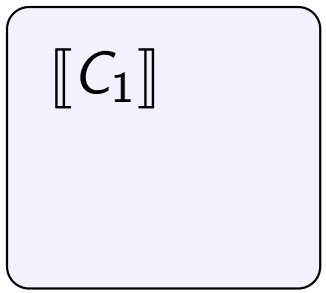
Call Stack Shortcutting

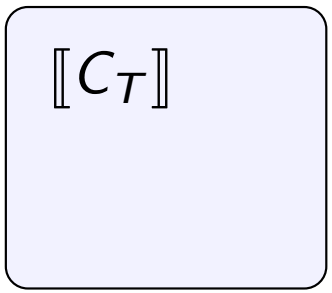


Call Stack Shortcutting



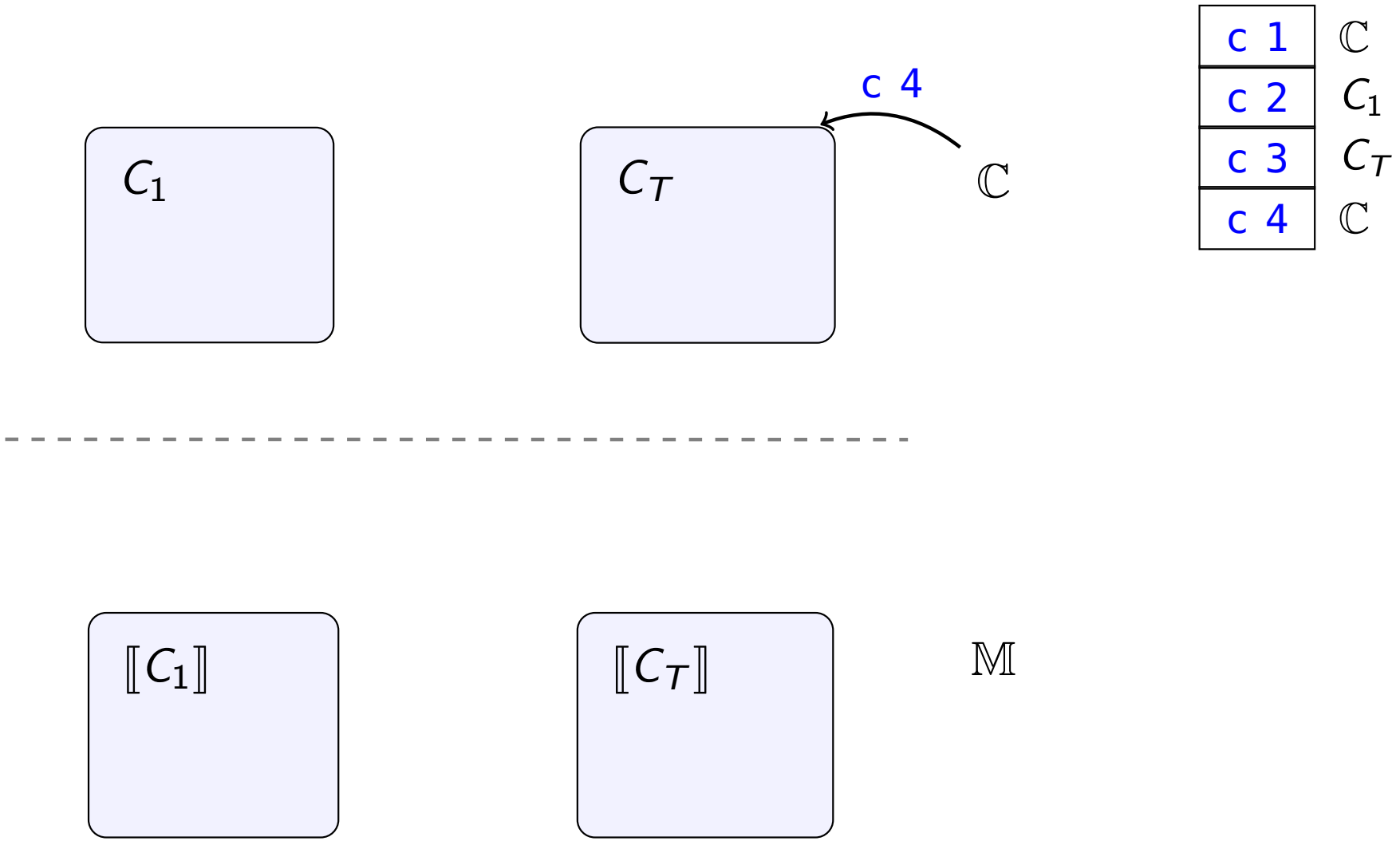




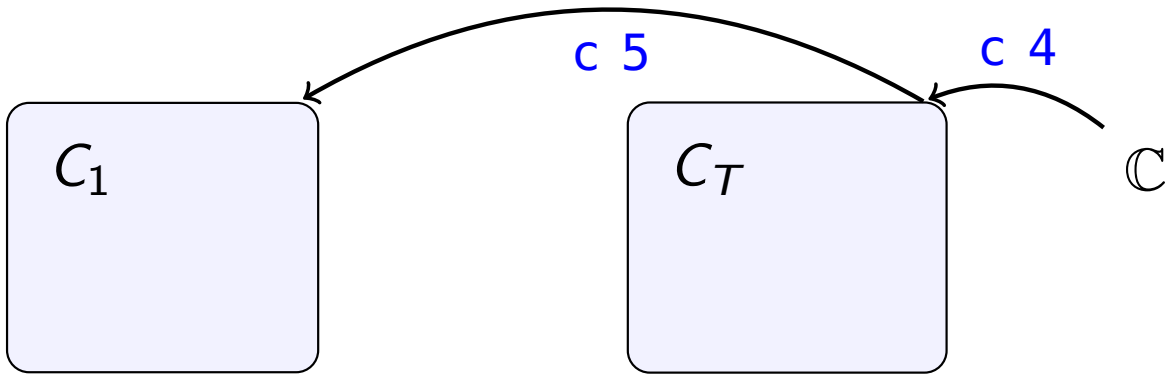


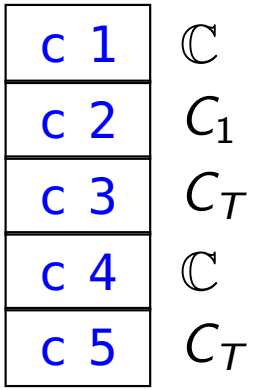
$\mathbb{M}$

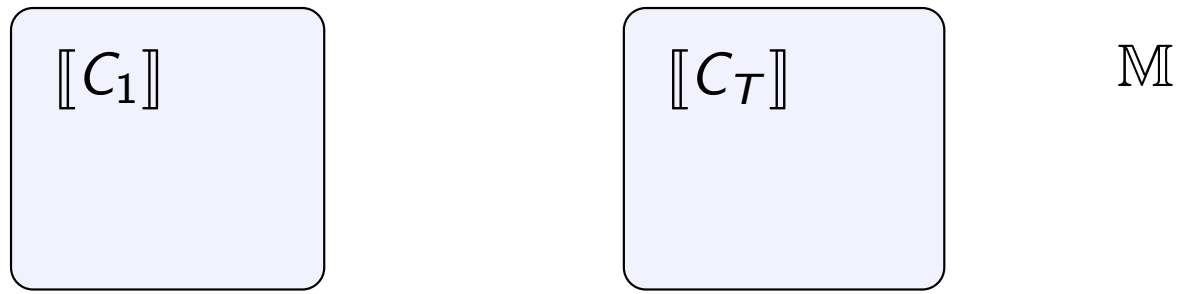
Call Stack Shortcutting



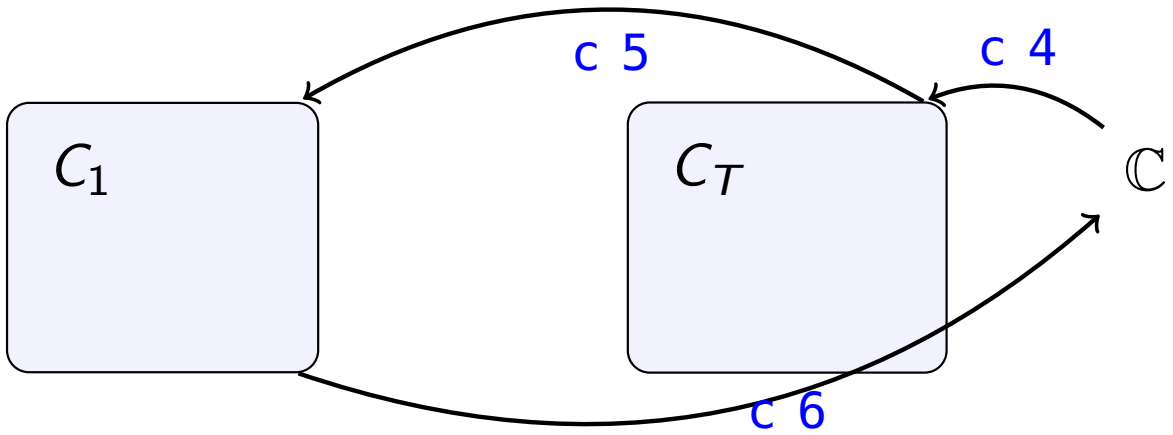
Call Stack Shortcutting



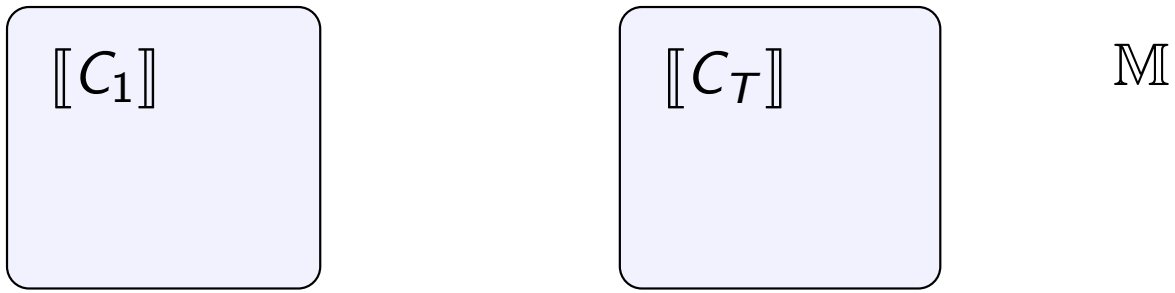




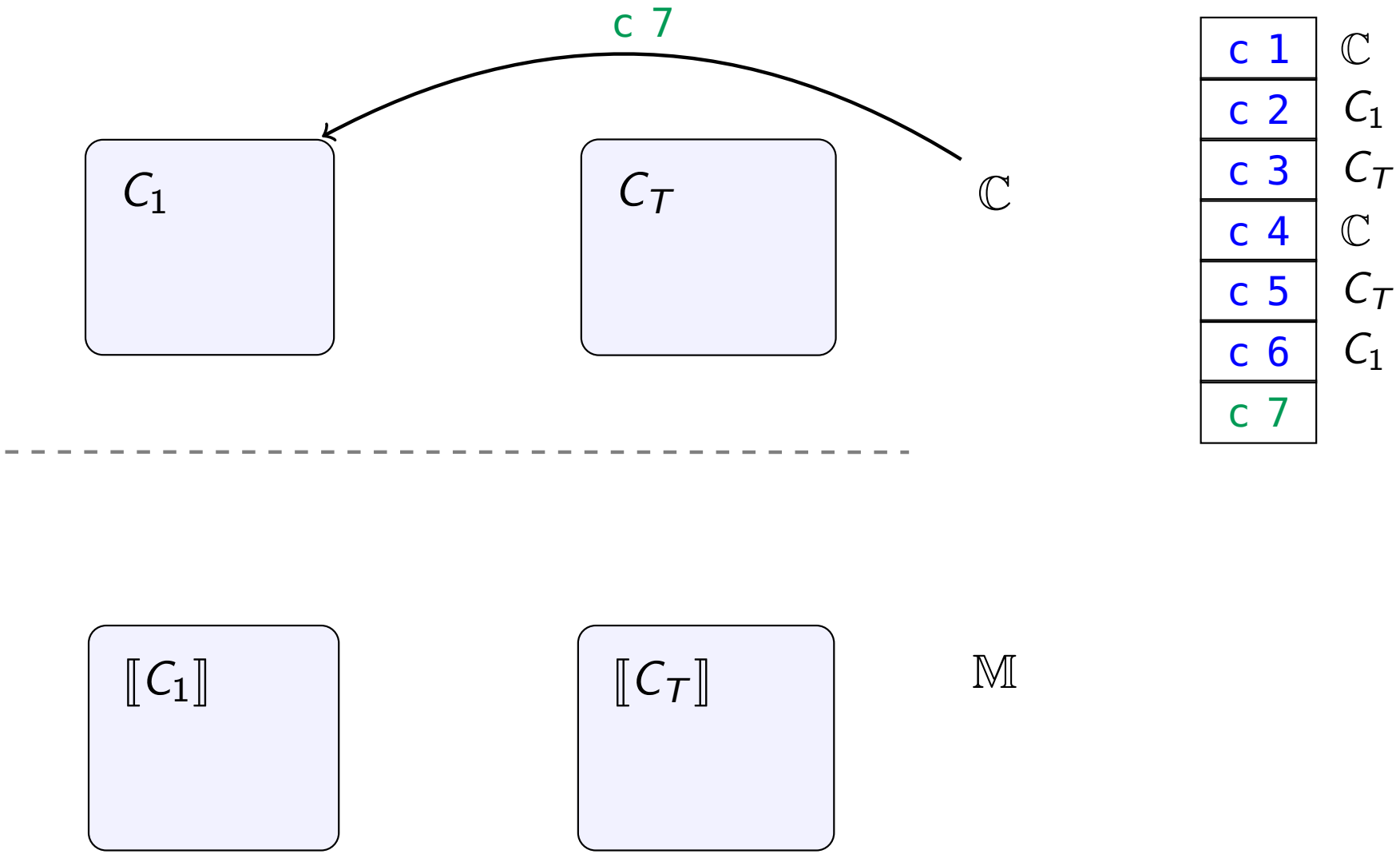
Call Stack Shortcutting



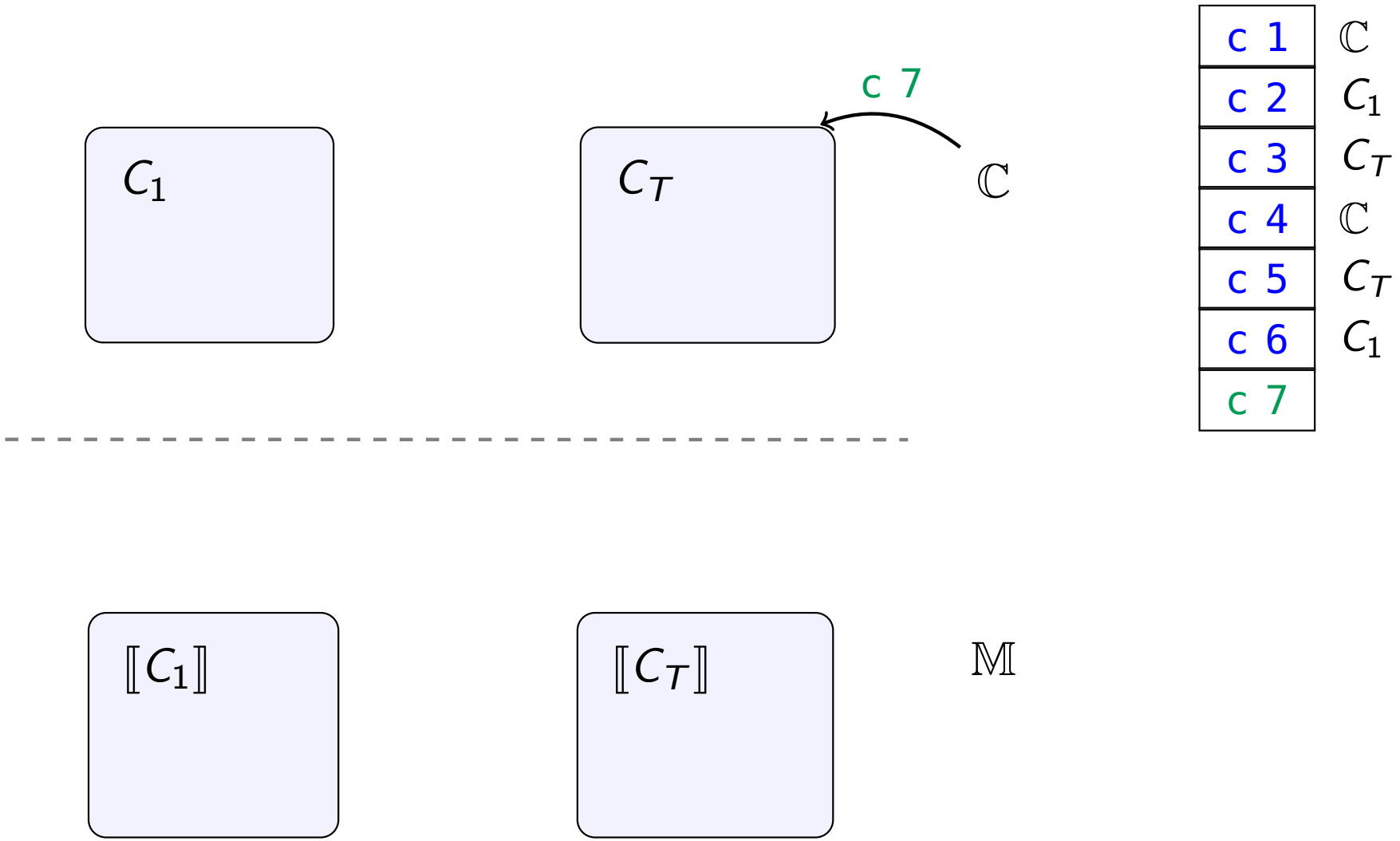
c 1	$\mathbb{C}$
c 2	C_1
c 3	C_T
c 4	$\mathbb{C}$
c 5	C_T
c 6	C_1



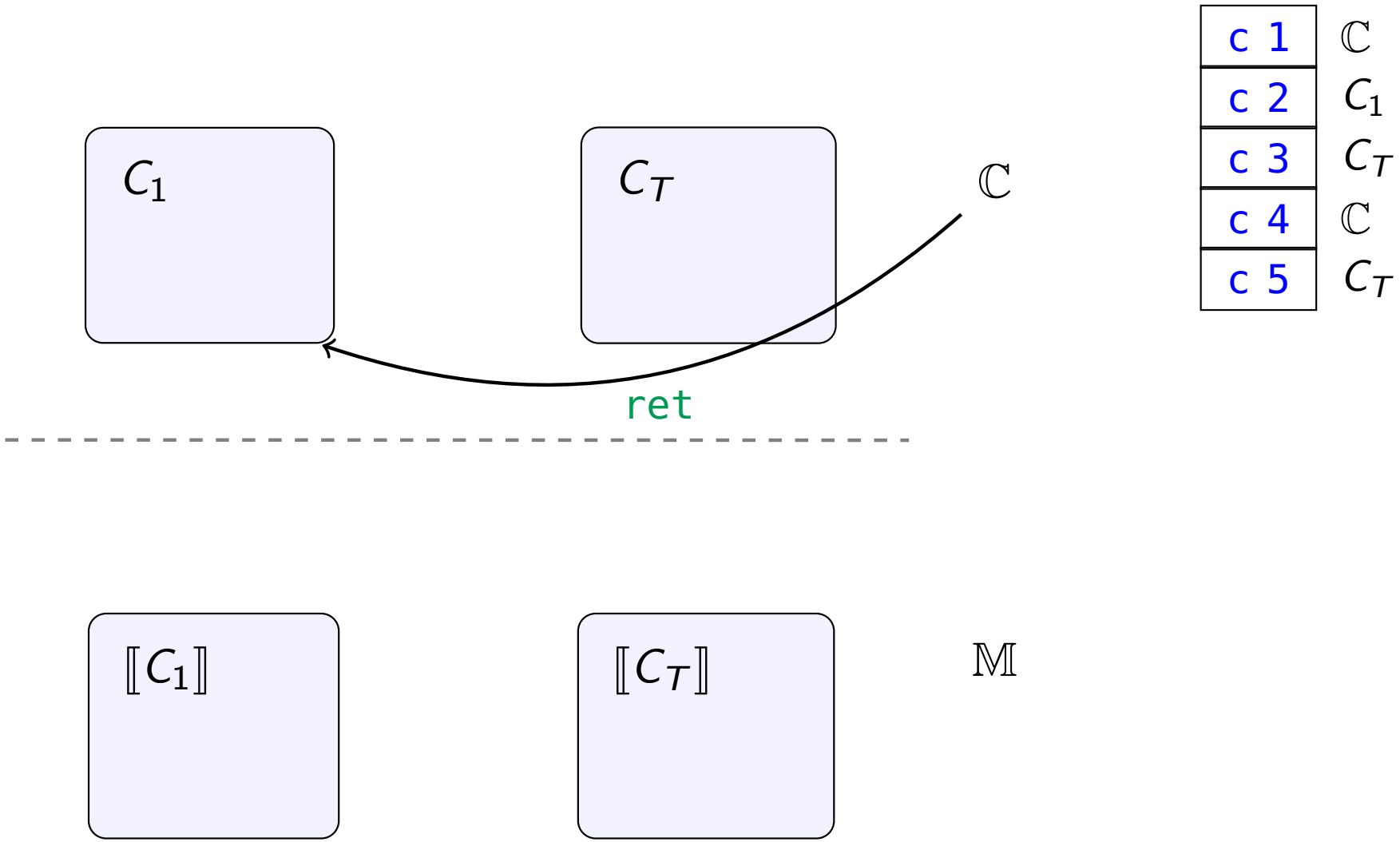
Call Stack Shortcutting



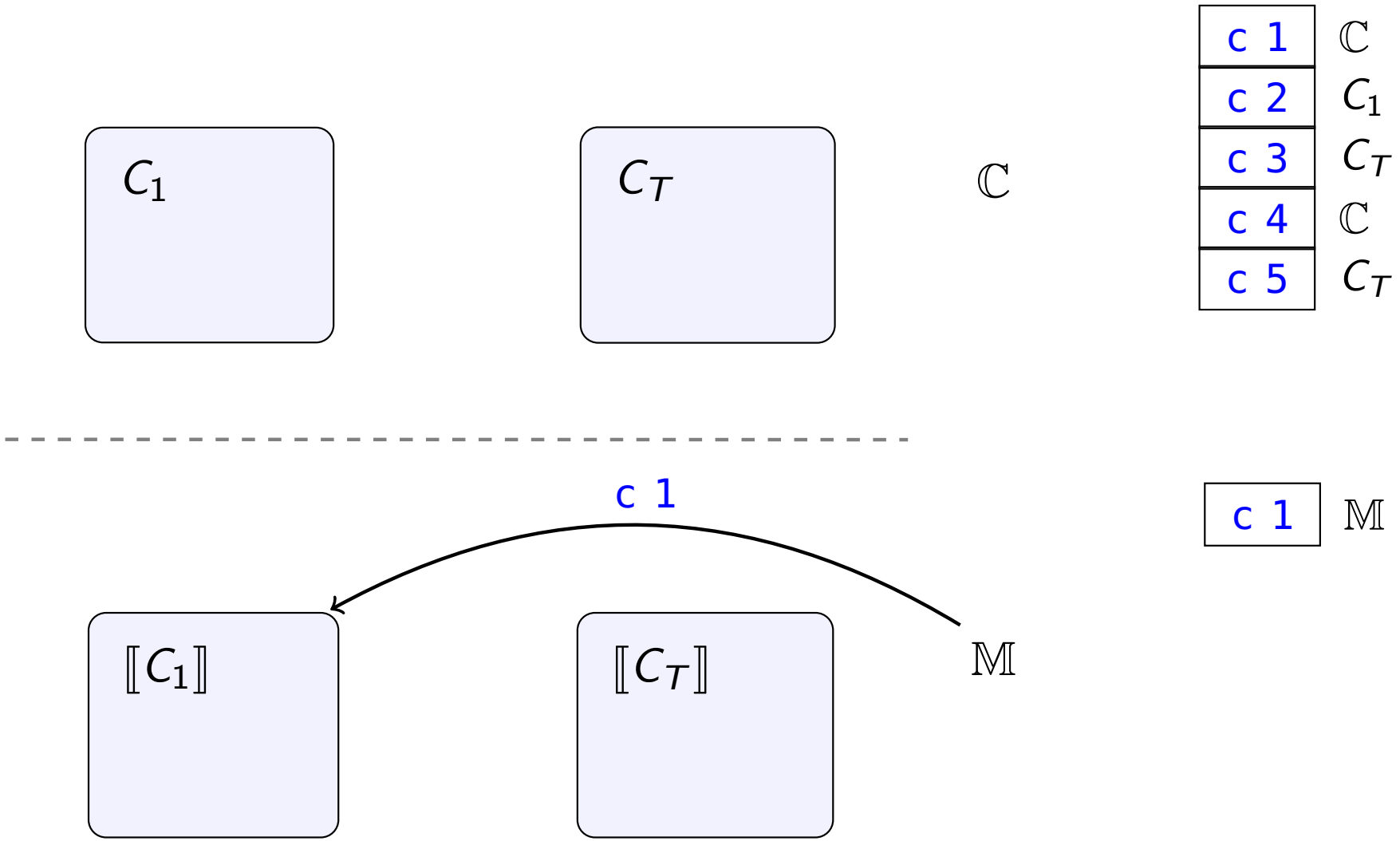
Call Stack Shortcutting



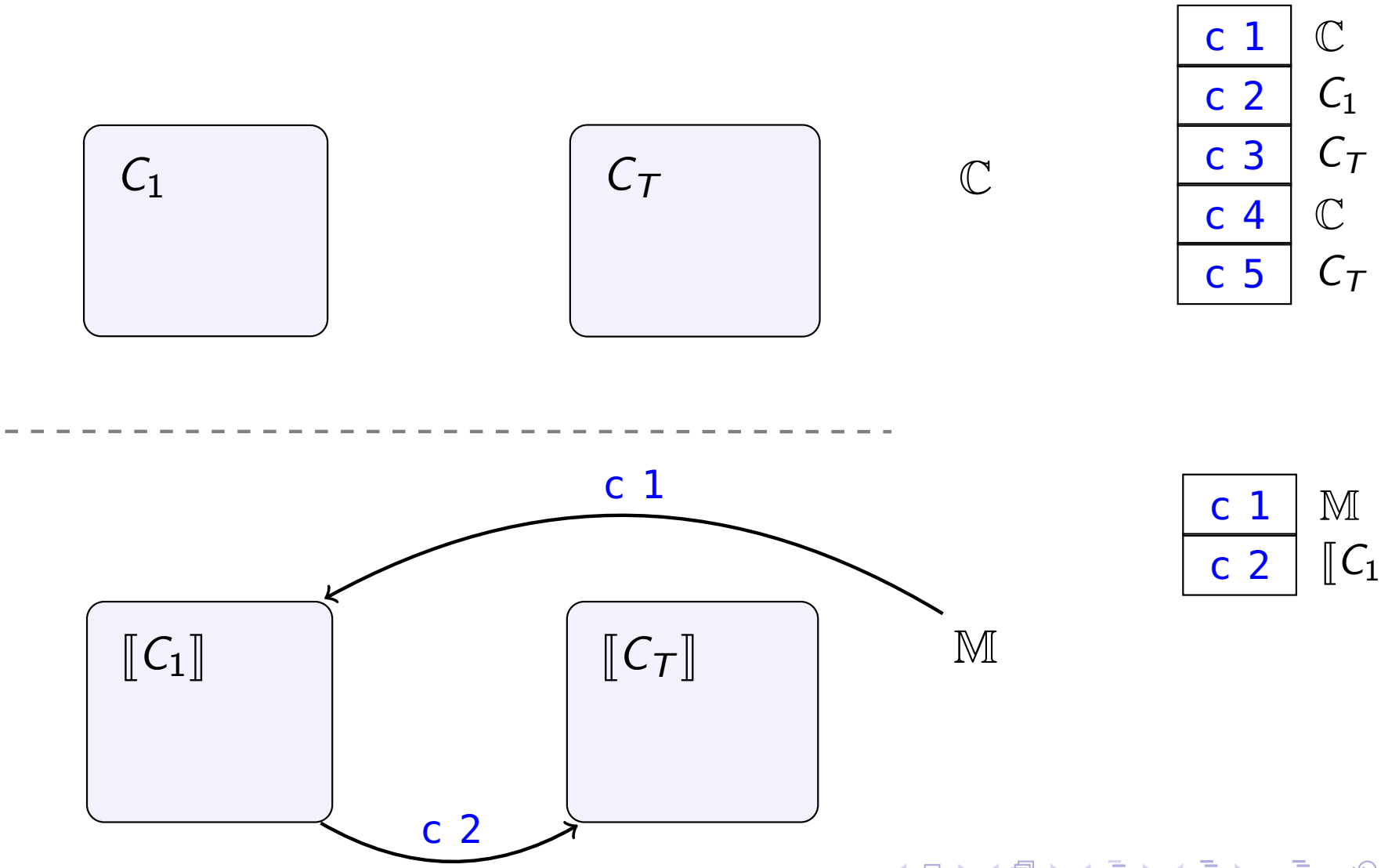
Call Stack Shortcutting



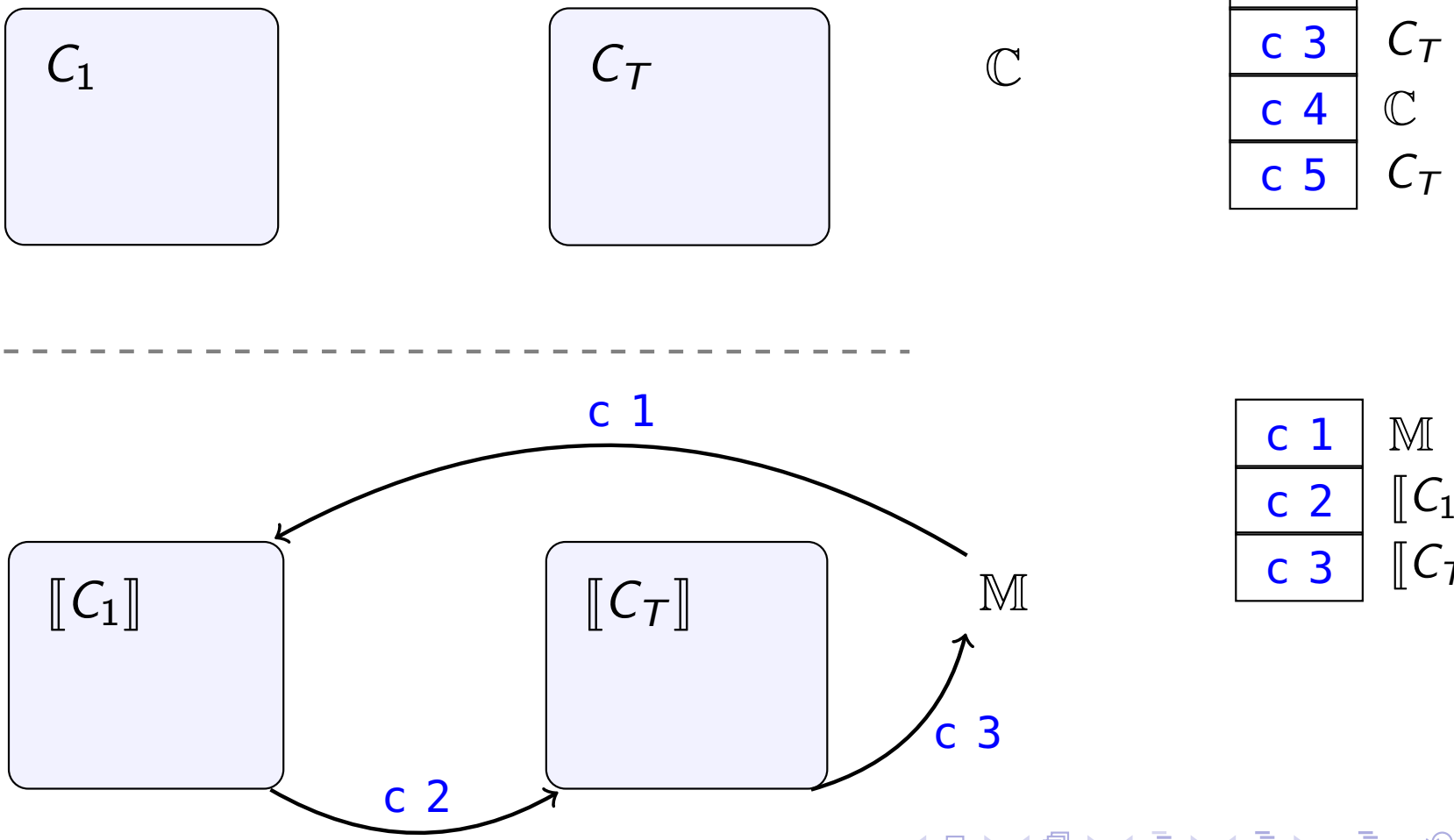
Call Stack Shortcutting



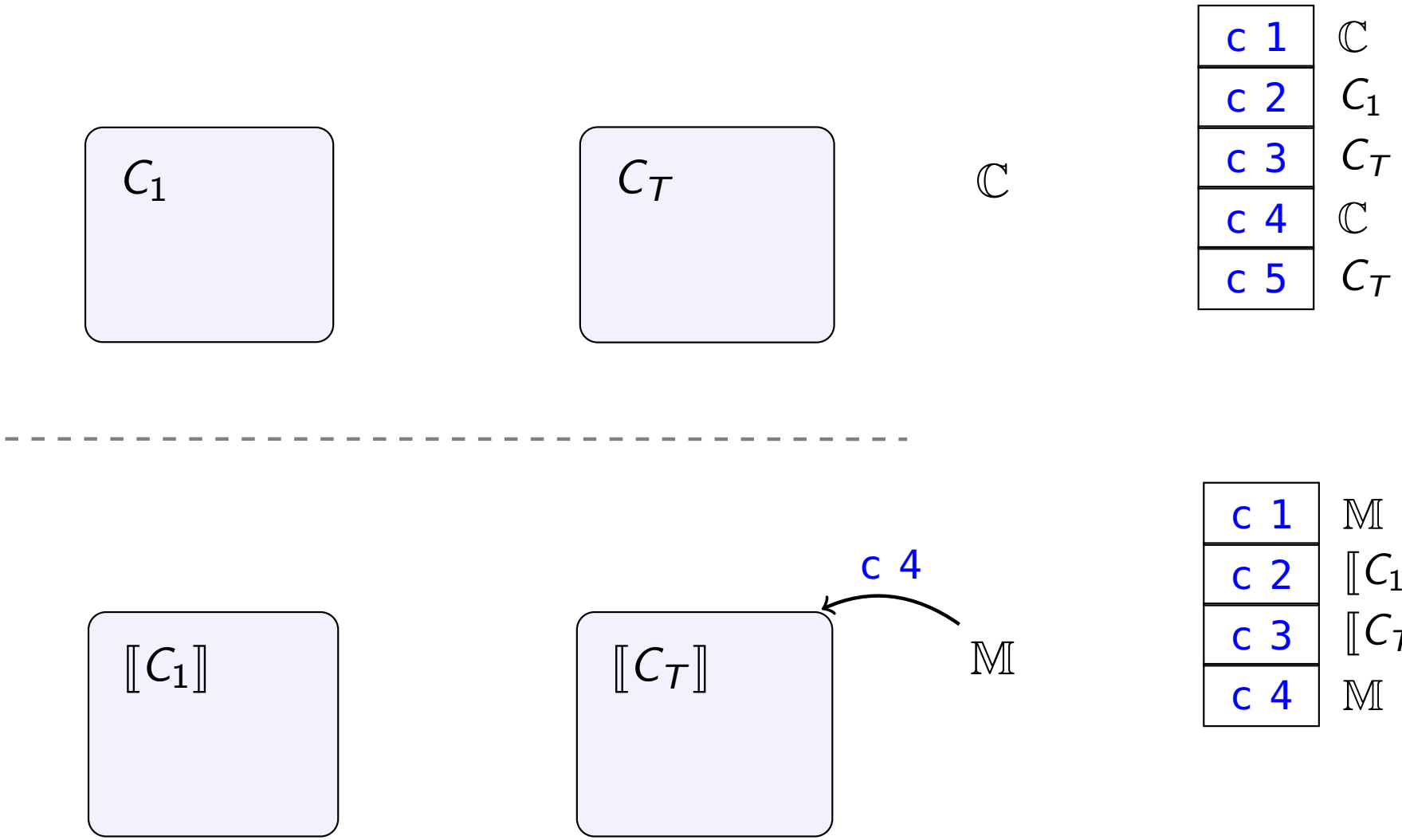
Call Stack Shortcutting



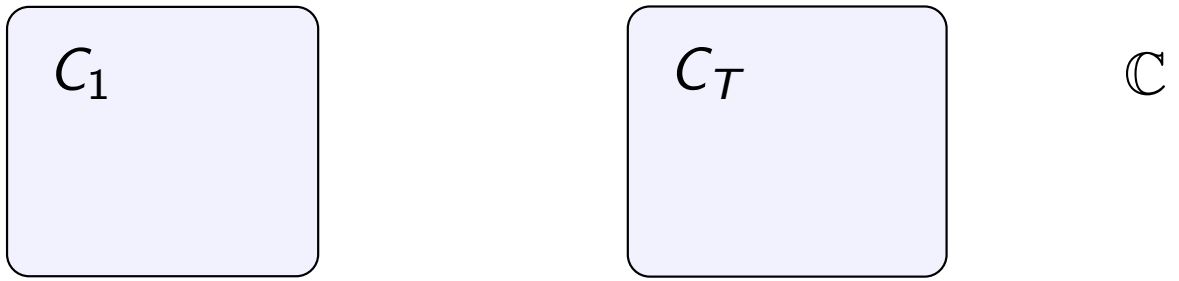
Call Stack Shortcutting



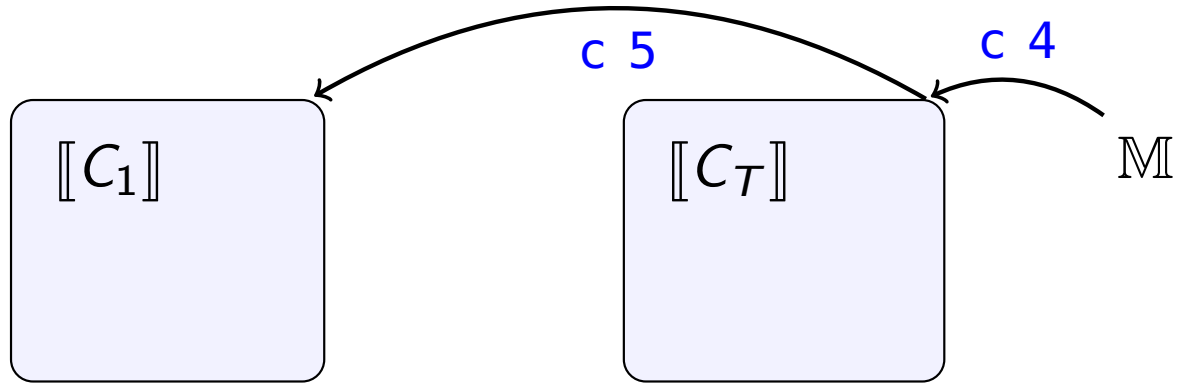
Call Stack Shortcutting



Call Stack Shortcutting

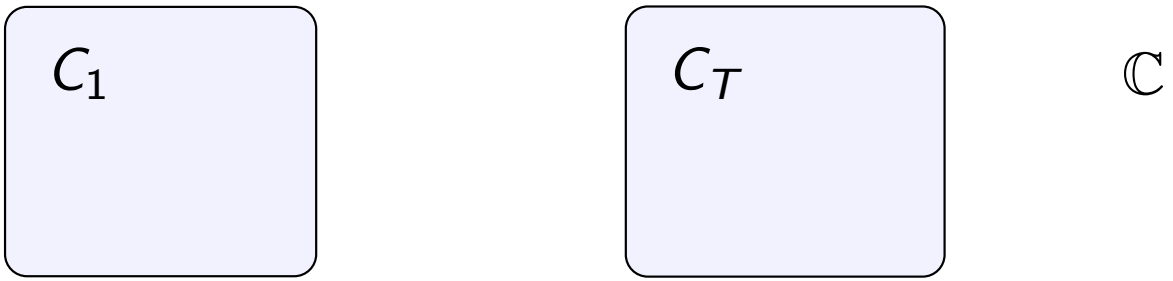


c 1	$\mathbb{C}$
c 2	C_1
c 3	C_T
c 4	$\mathbb{C}$
c 5	C_T

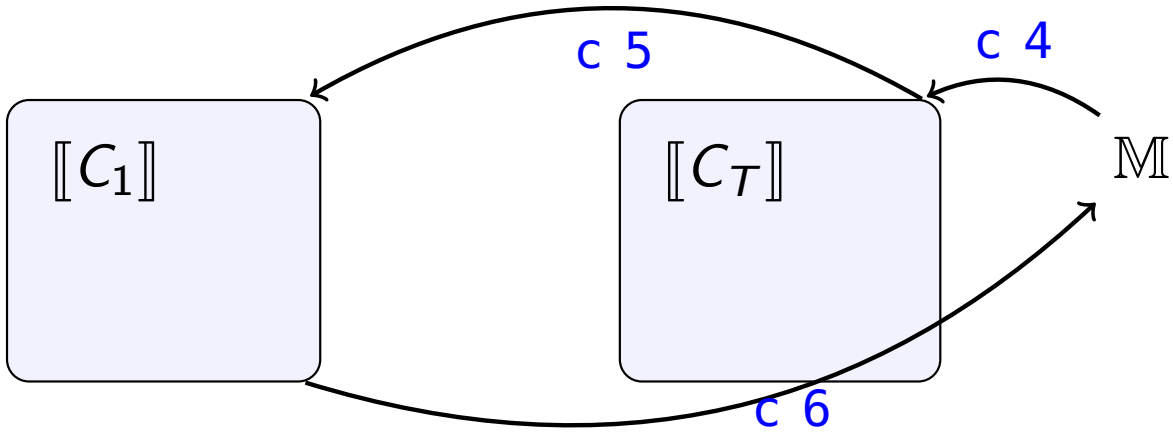


c 1	$\mathbb{M}$
c 2	$[\![C_1]\!]$
c 3	$[\![C_T]\!]$
c 4	$\mathbb{M}$
c 5	$[\![C_T]\!]$

Call Stack Shortcutting

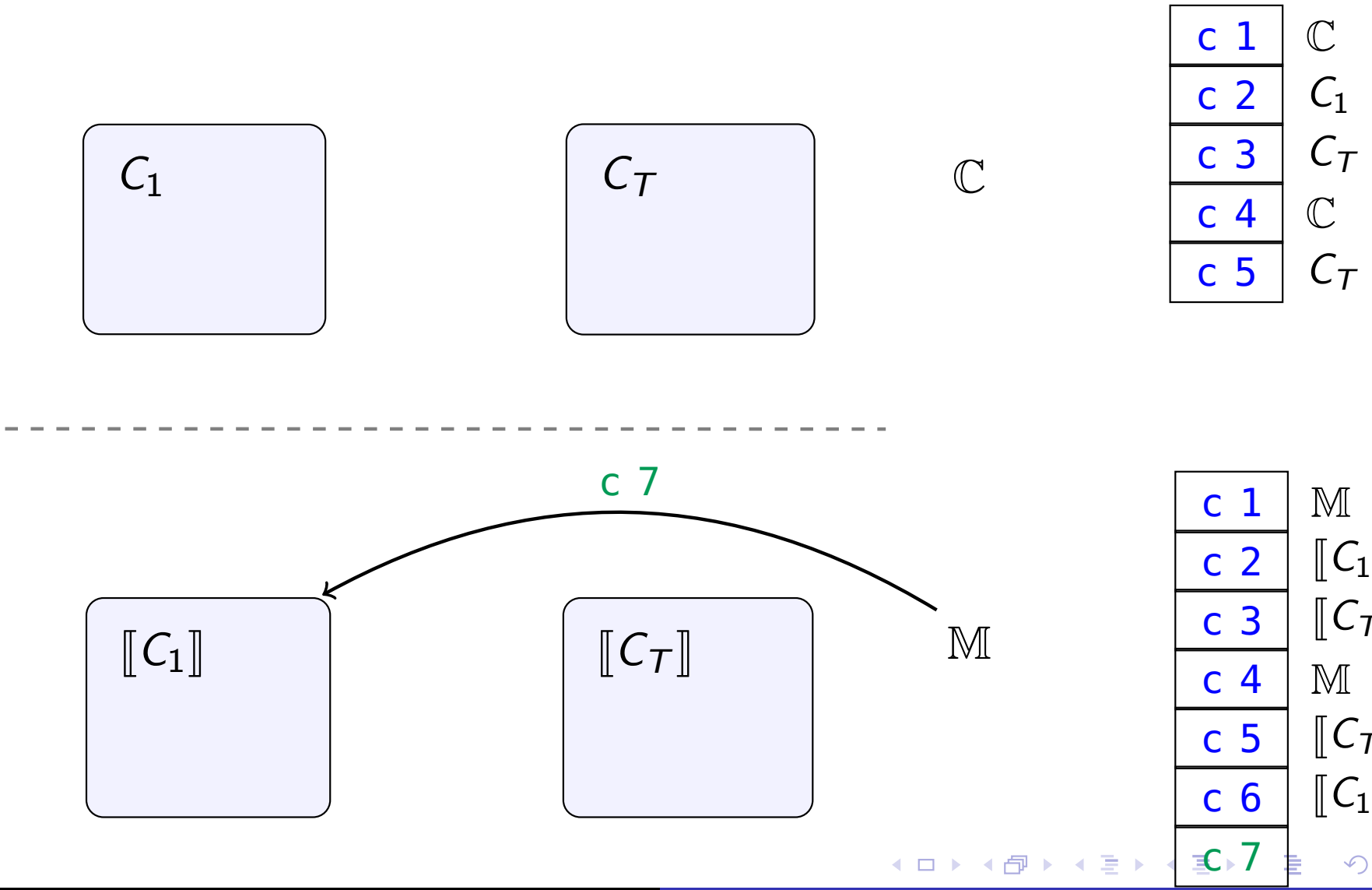


c 1	$\mathbb{C}$
c 2	C_1
c 3	C_T
c 4	$\mathbb{C}$
c 5	C_T

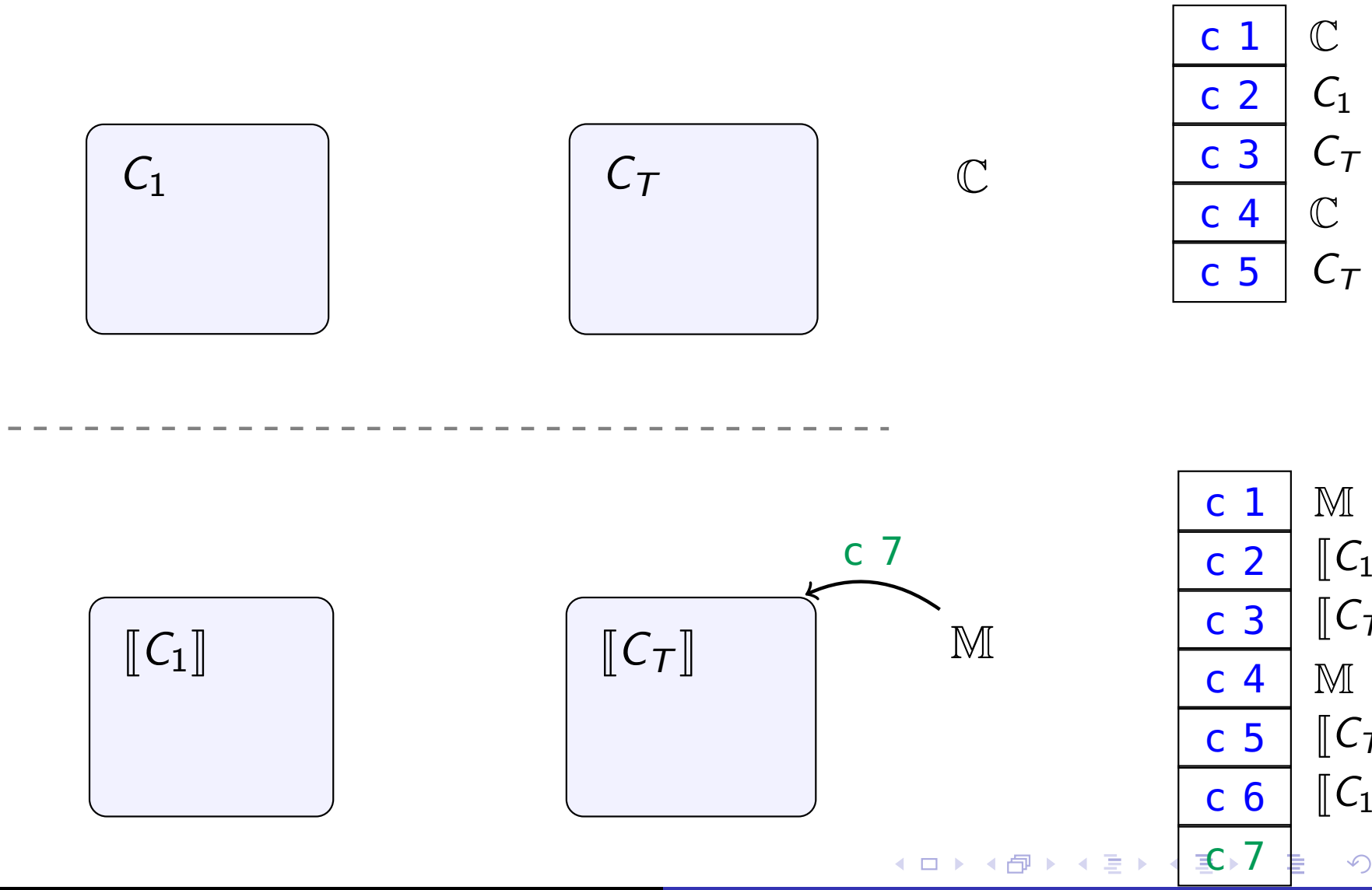


c 1	$\mathbb{M}$
c 2	$[\![C_1]\!]$
c 3	$[\![C_T]\!]$
c 4	$\mathbb{M}$
c 5	$[\![C_T]\!]$
c 6	$[\![C_1]\!]$

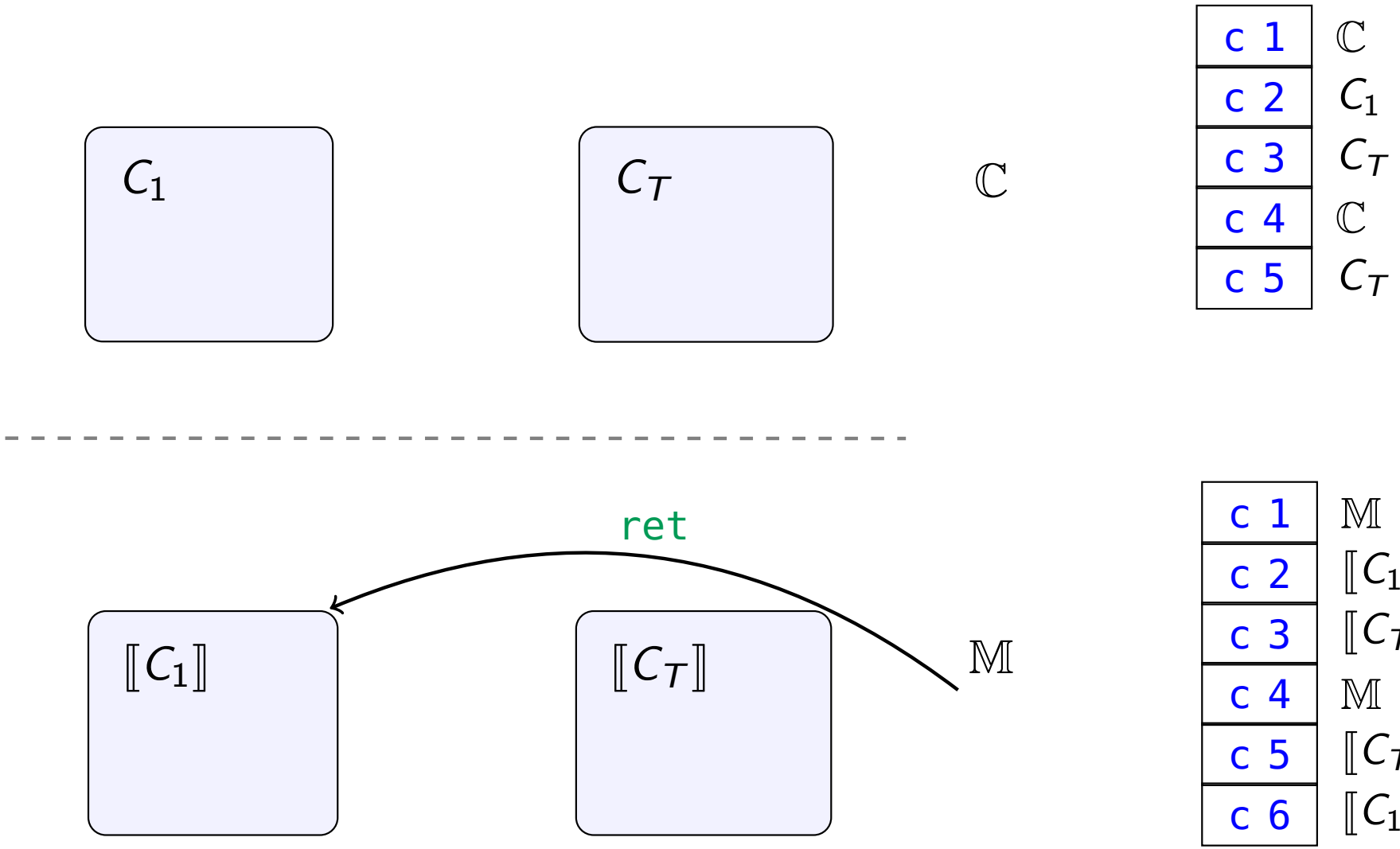
Call Stack Shortcutting



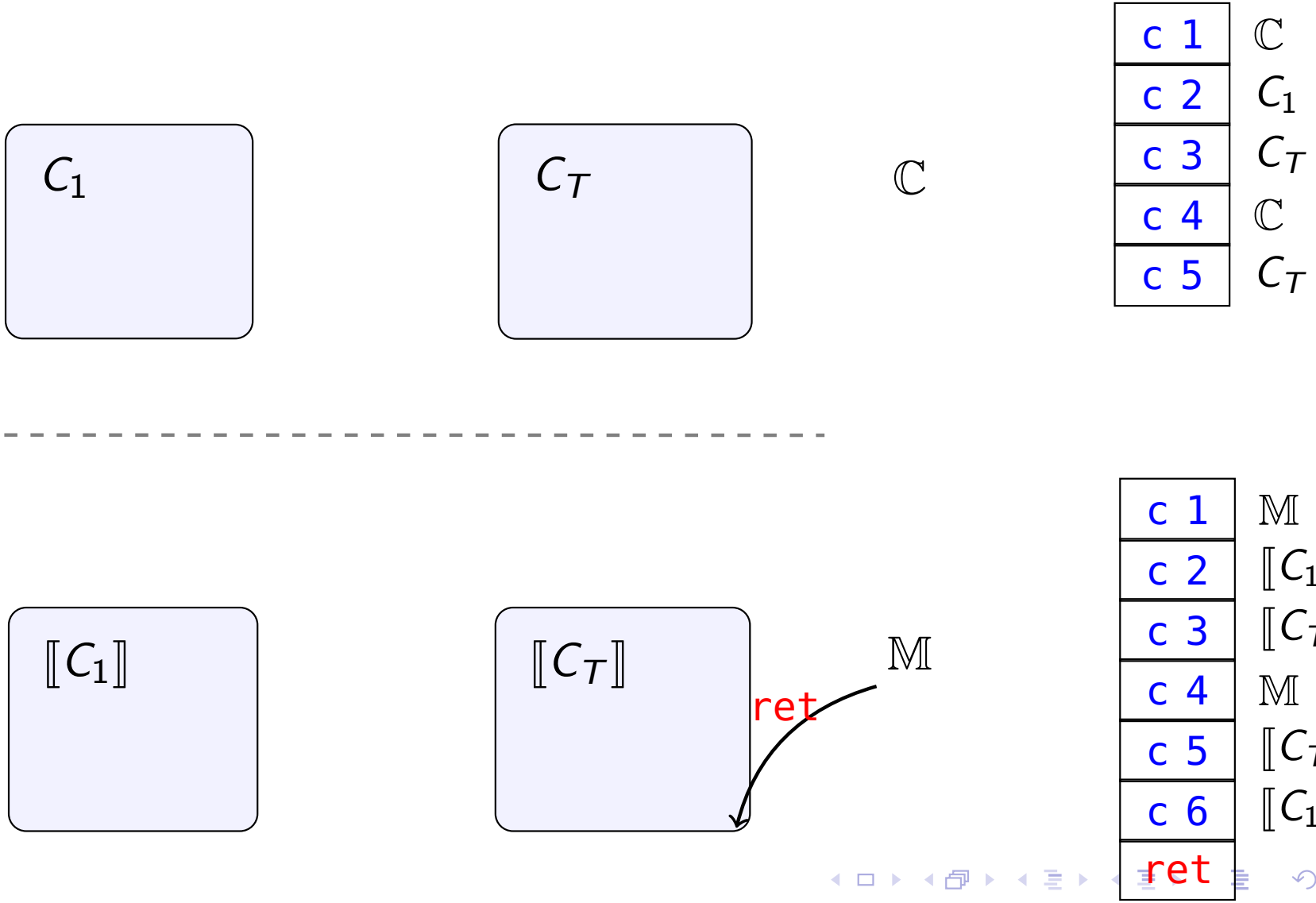
Call Stack Shortcutting



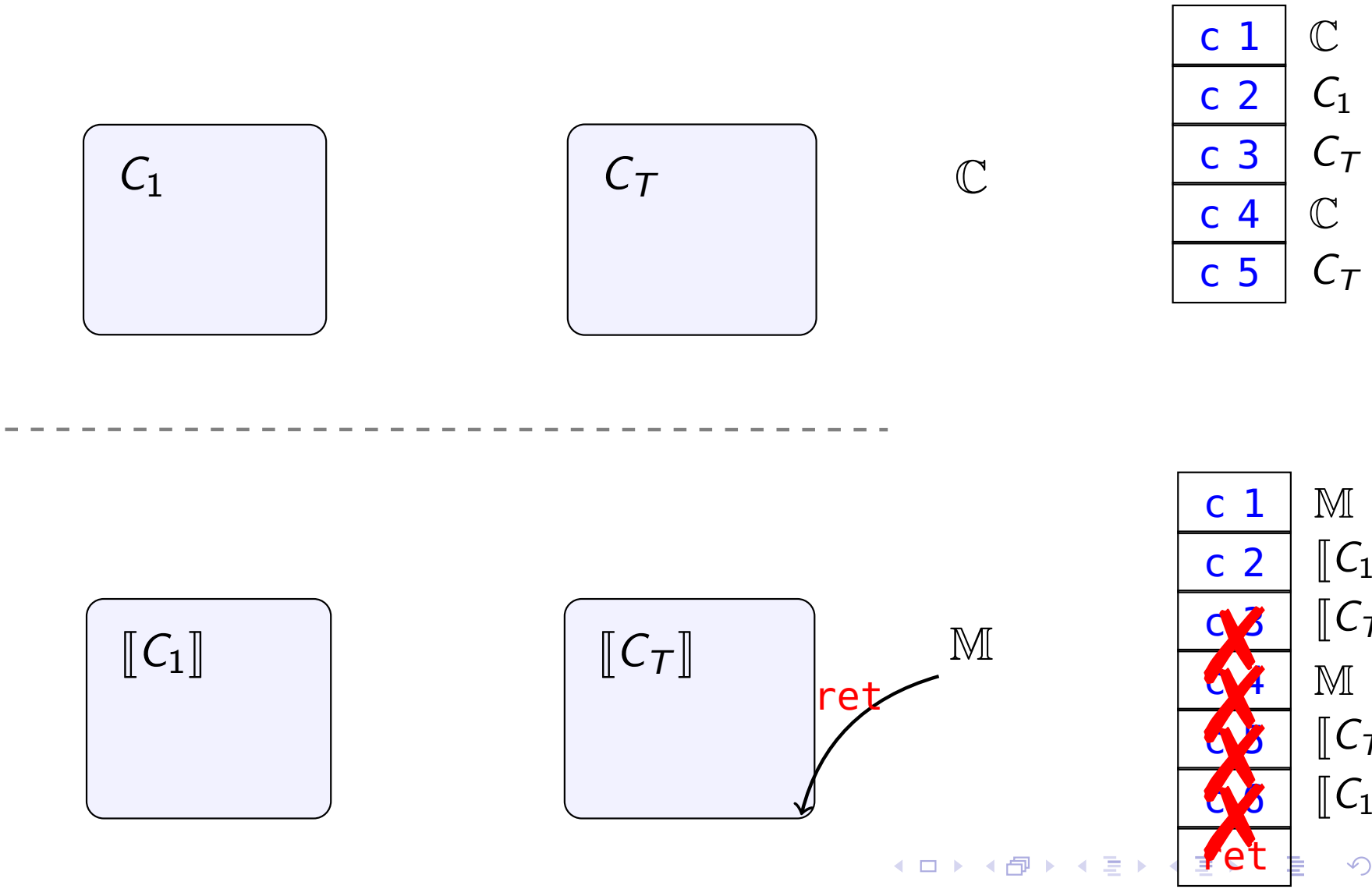
Call Stack Shortcutting



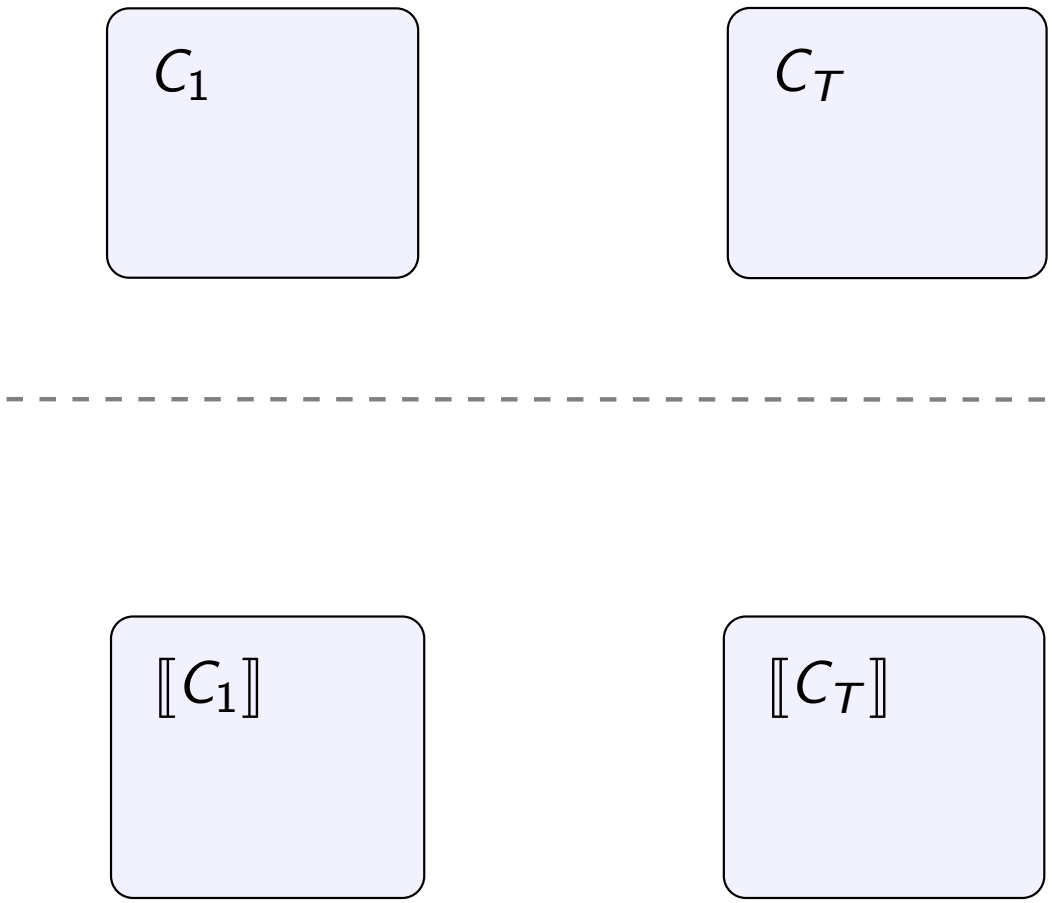
Call Stack Shortcutting



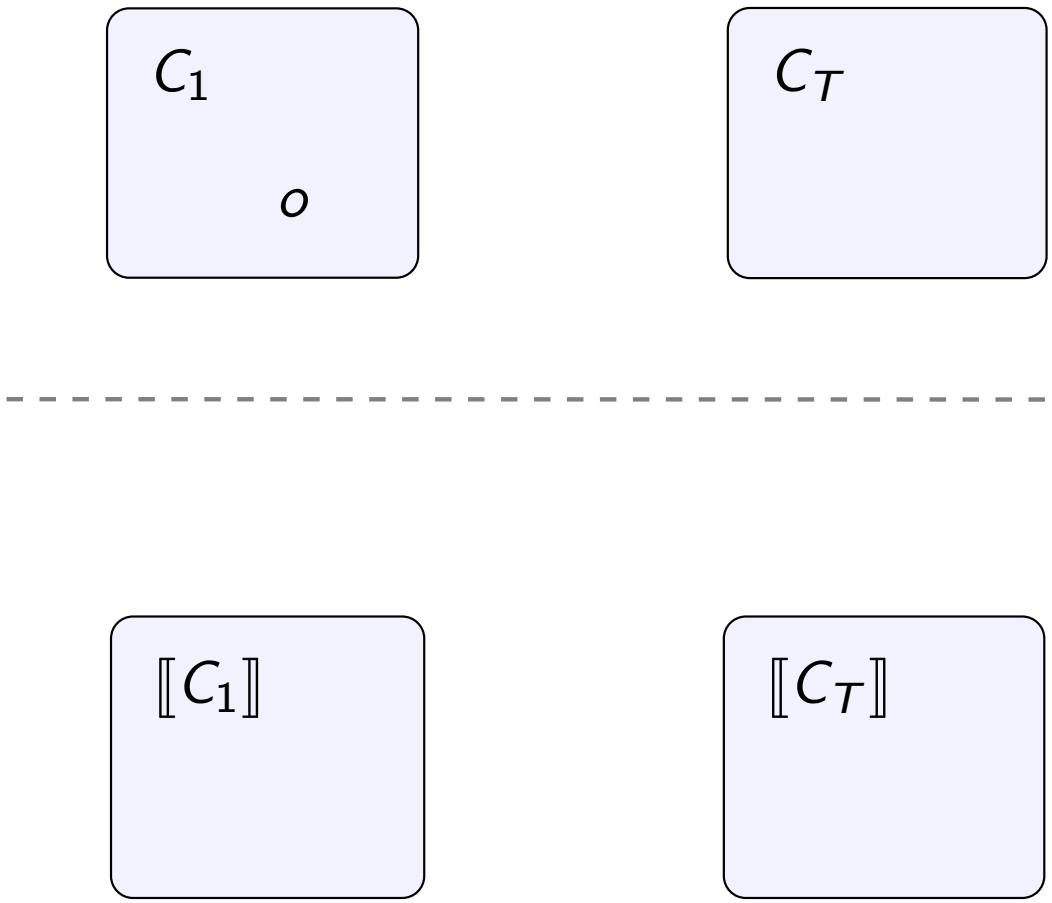
Call Stack Shortcutting



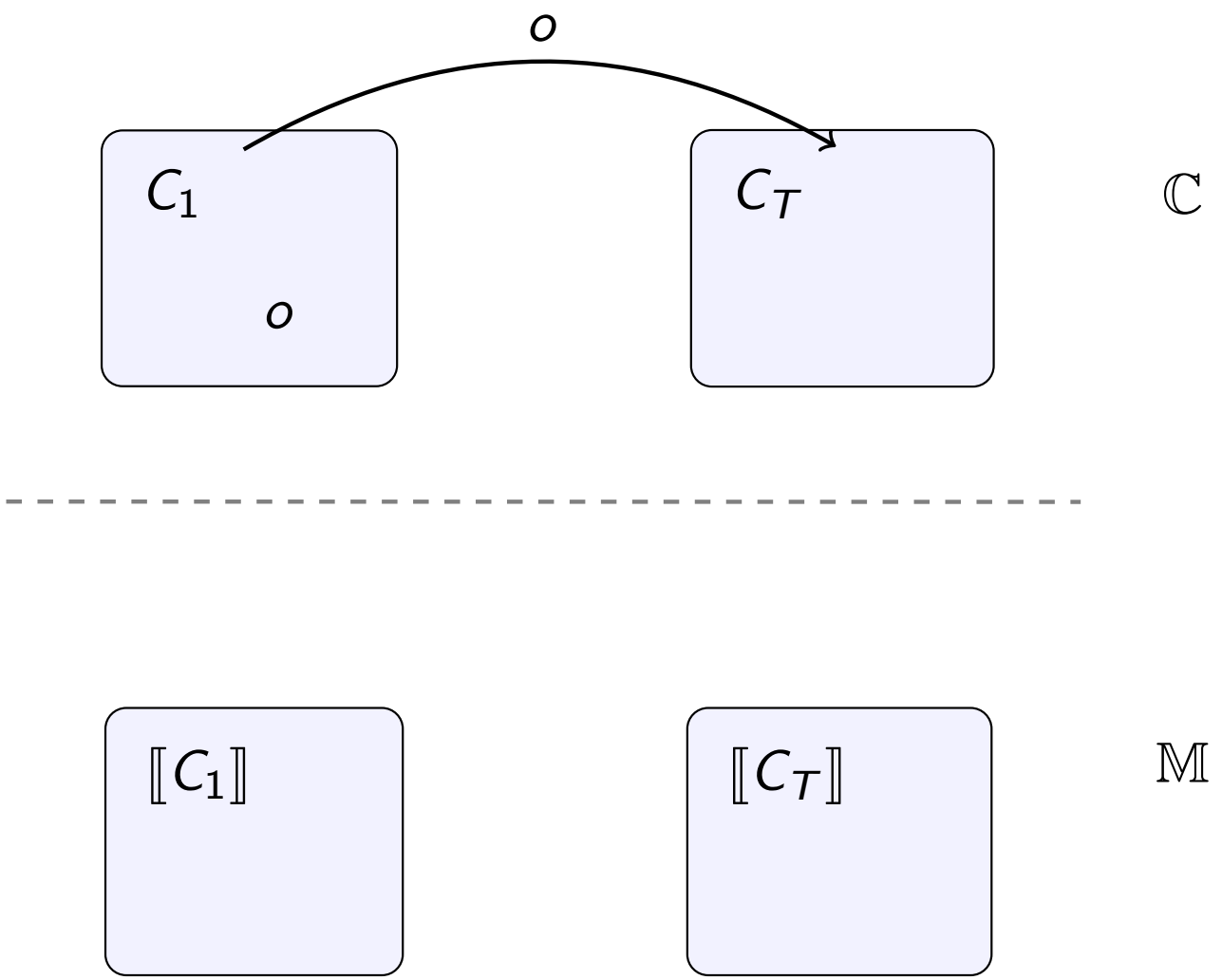
Object Guessing



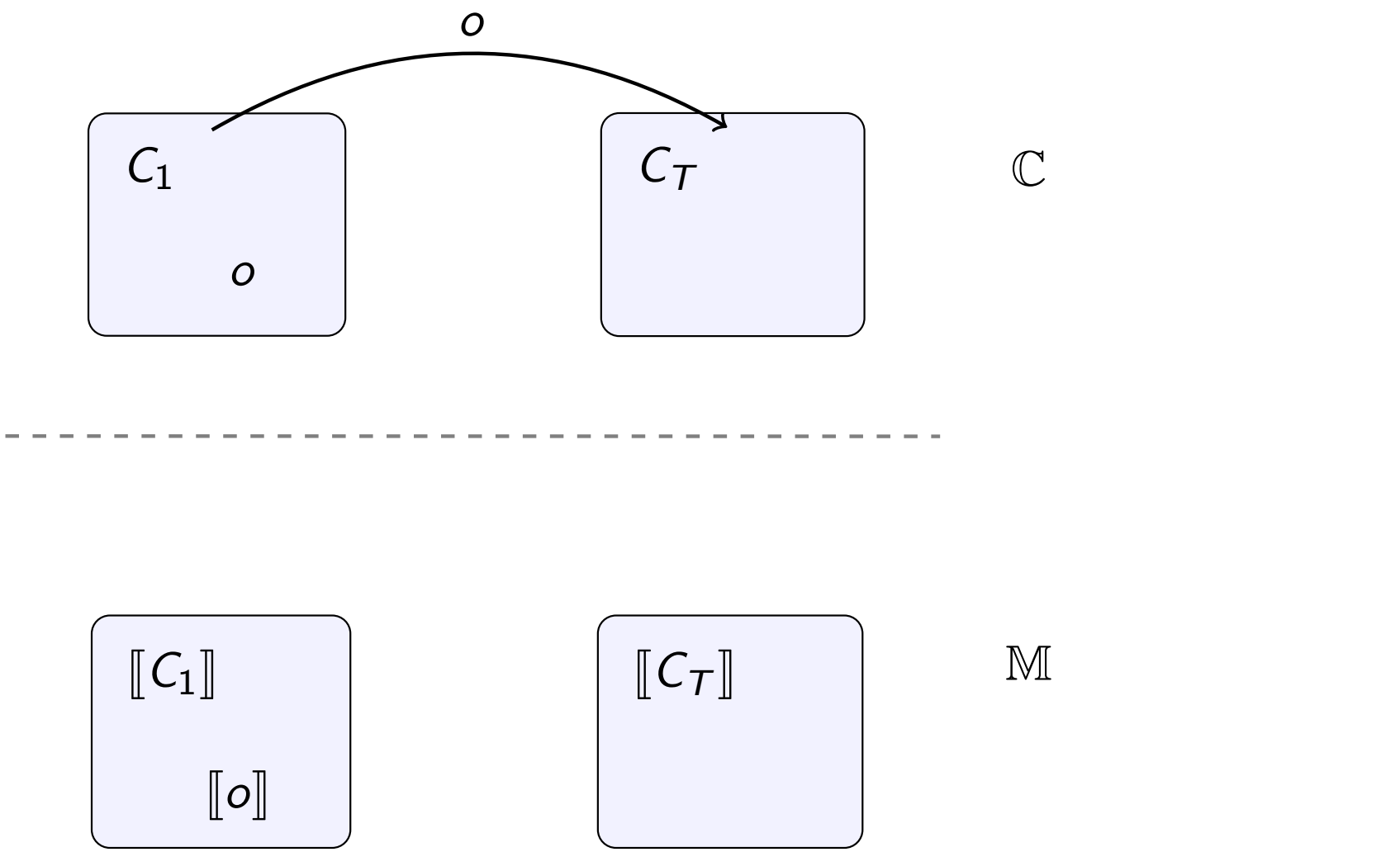
Object Guessing



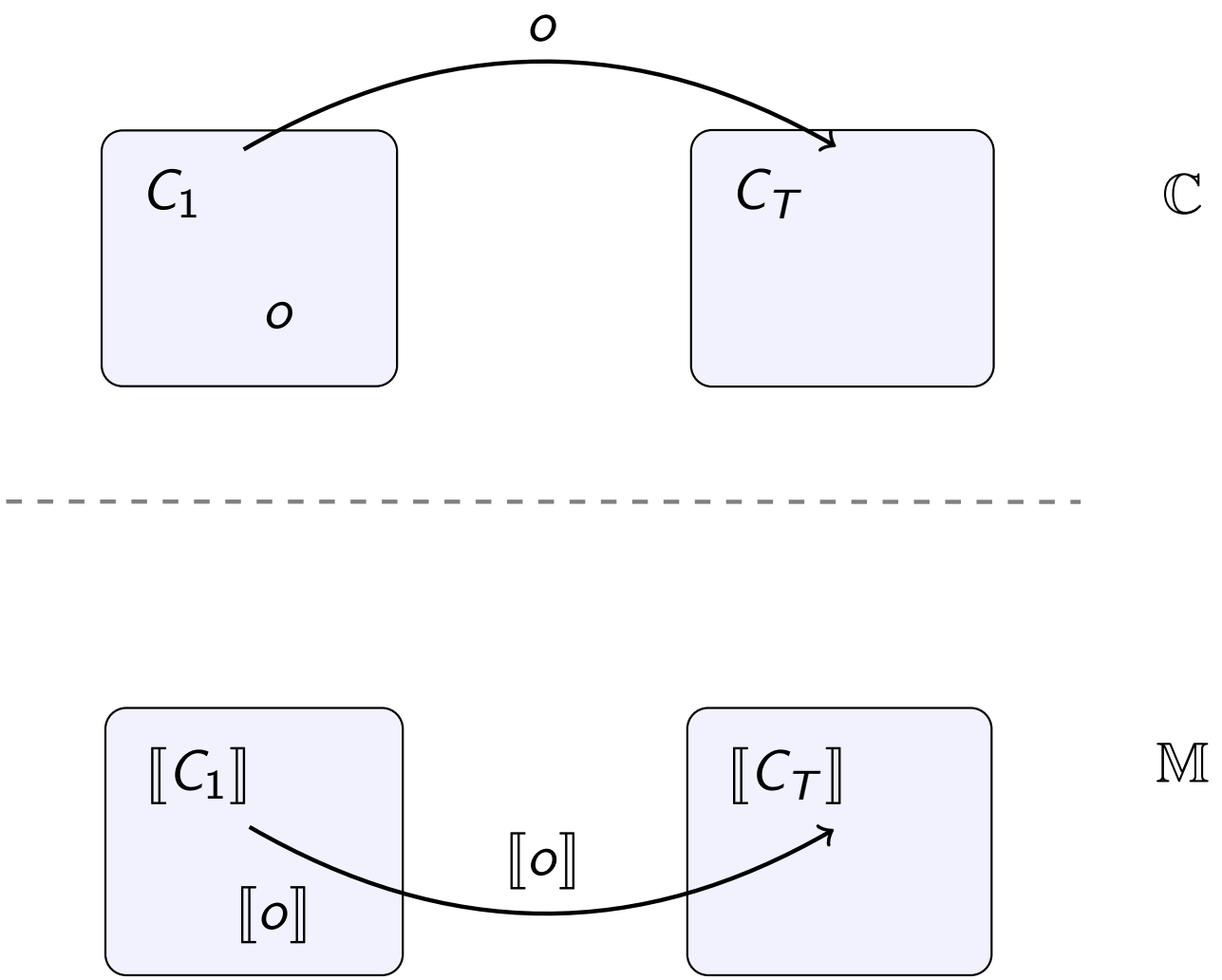
Object Guessing



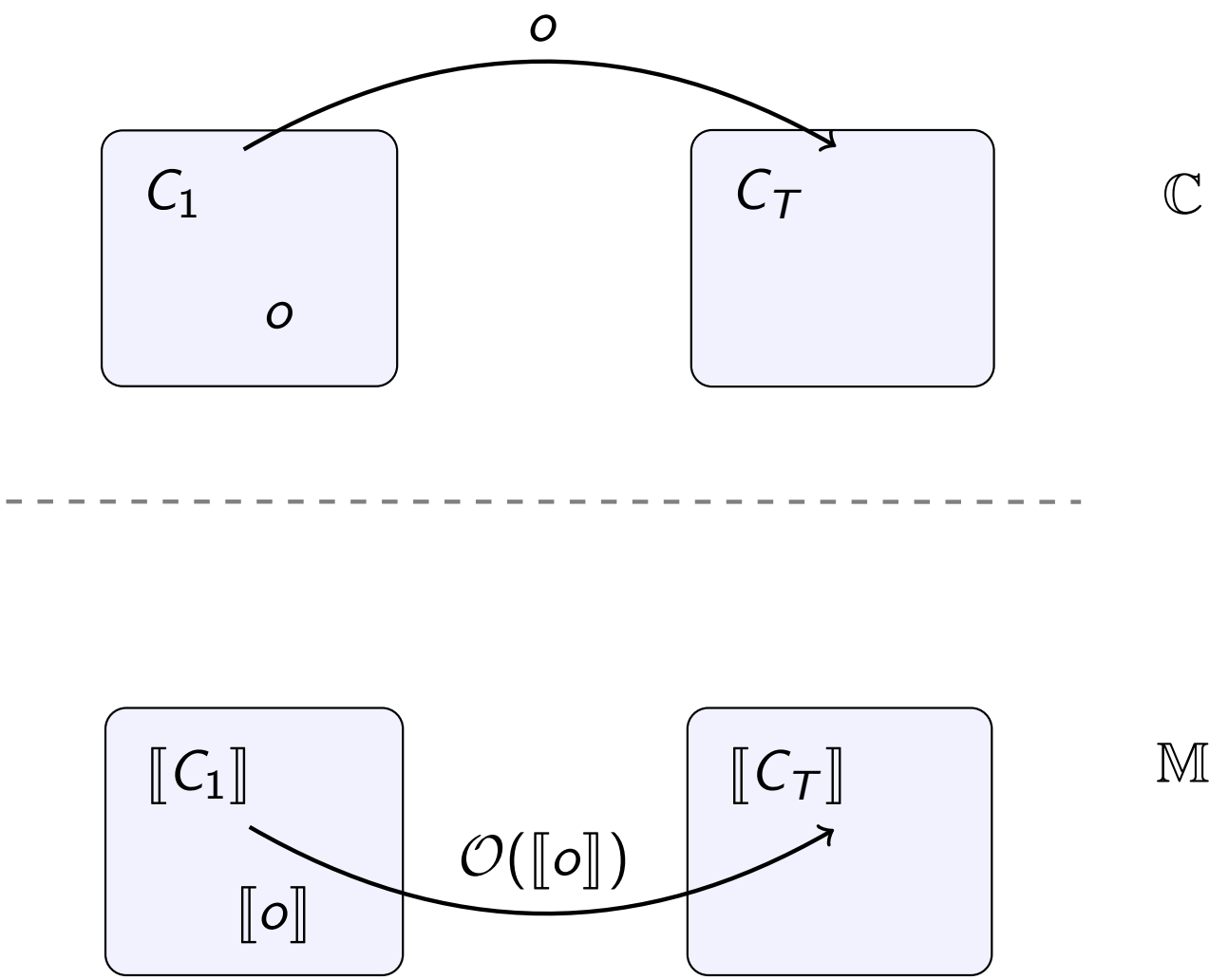
Object Guessing



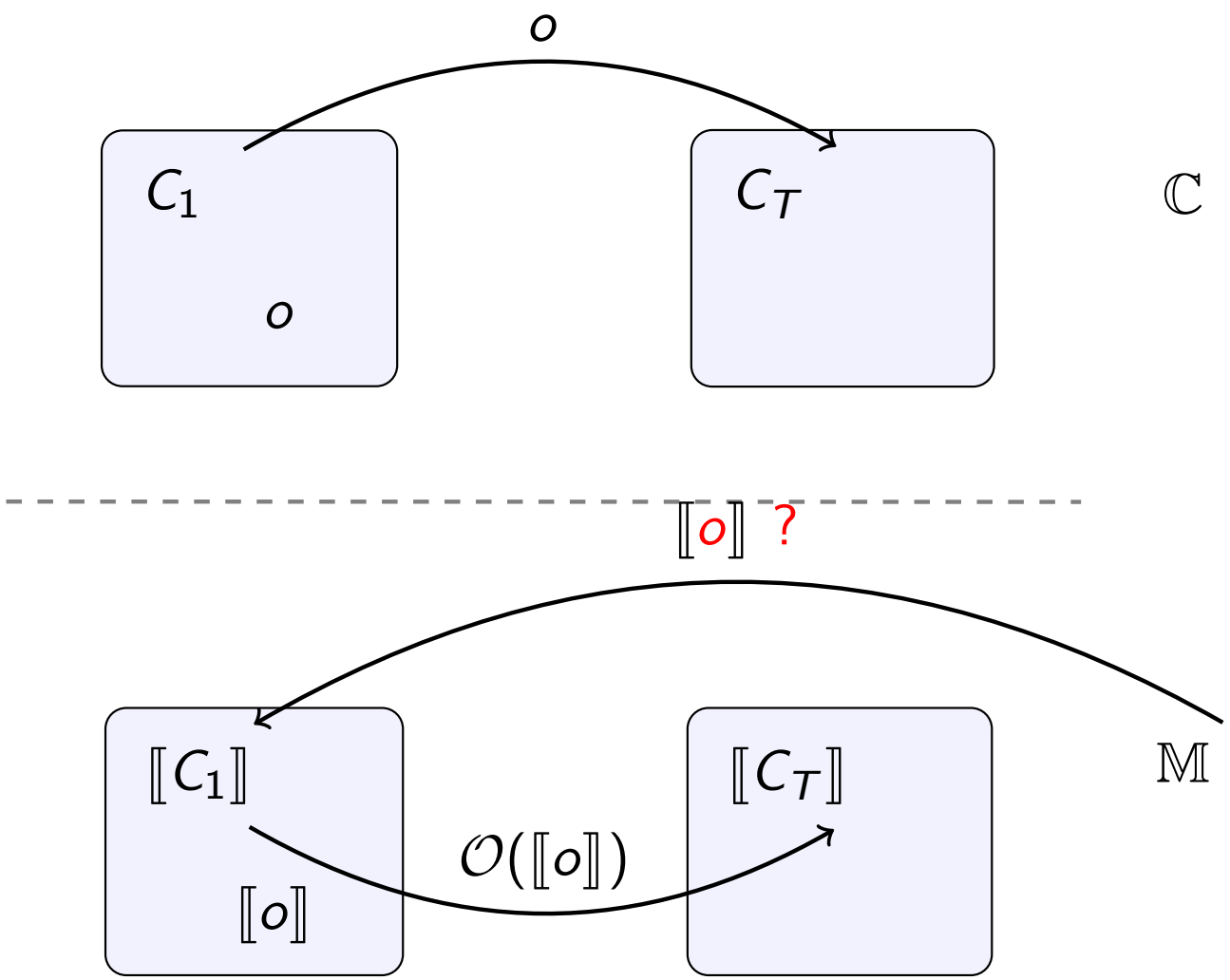
Object Guessing



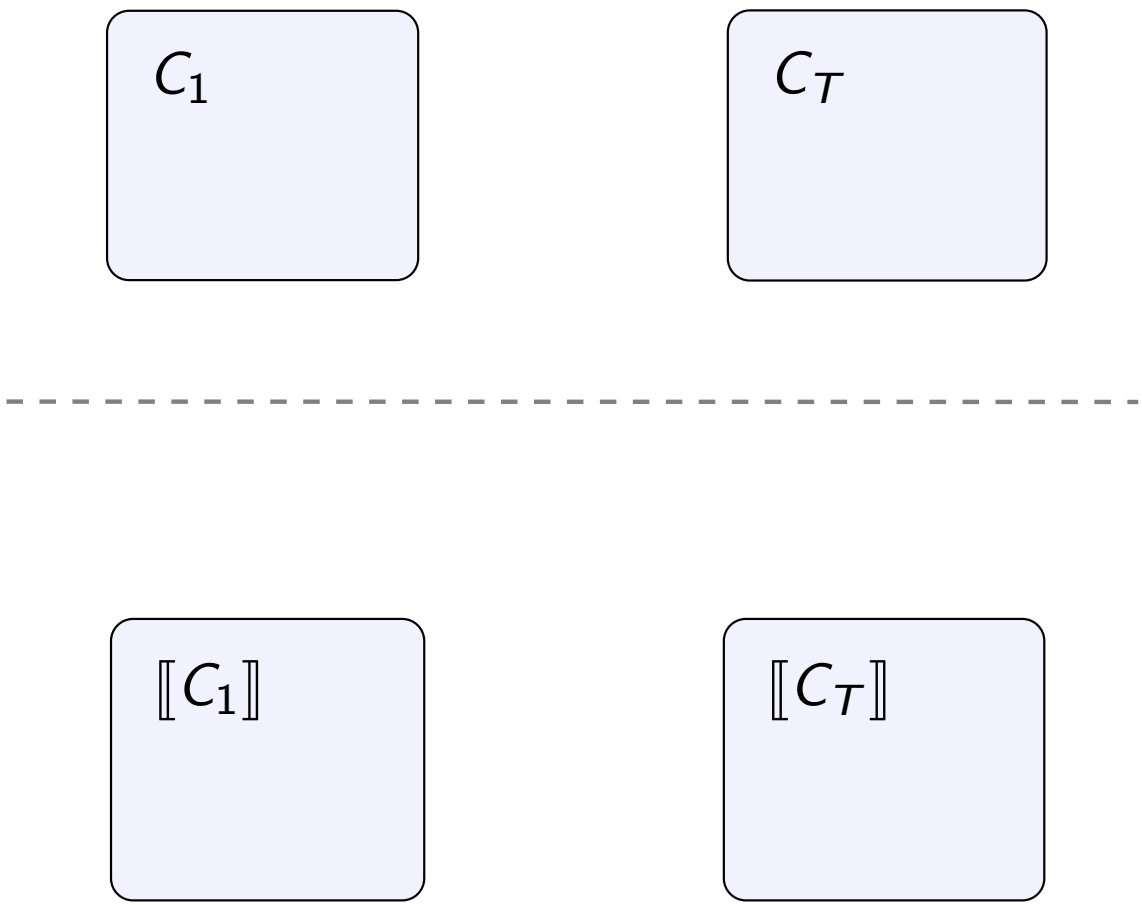
Object Guessing



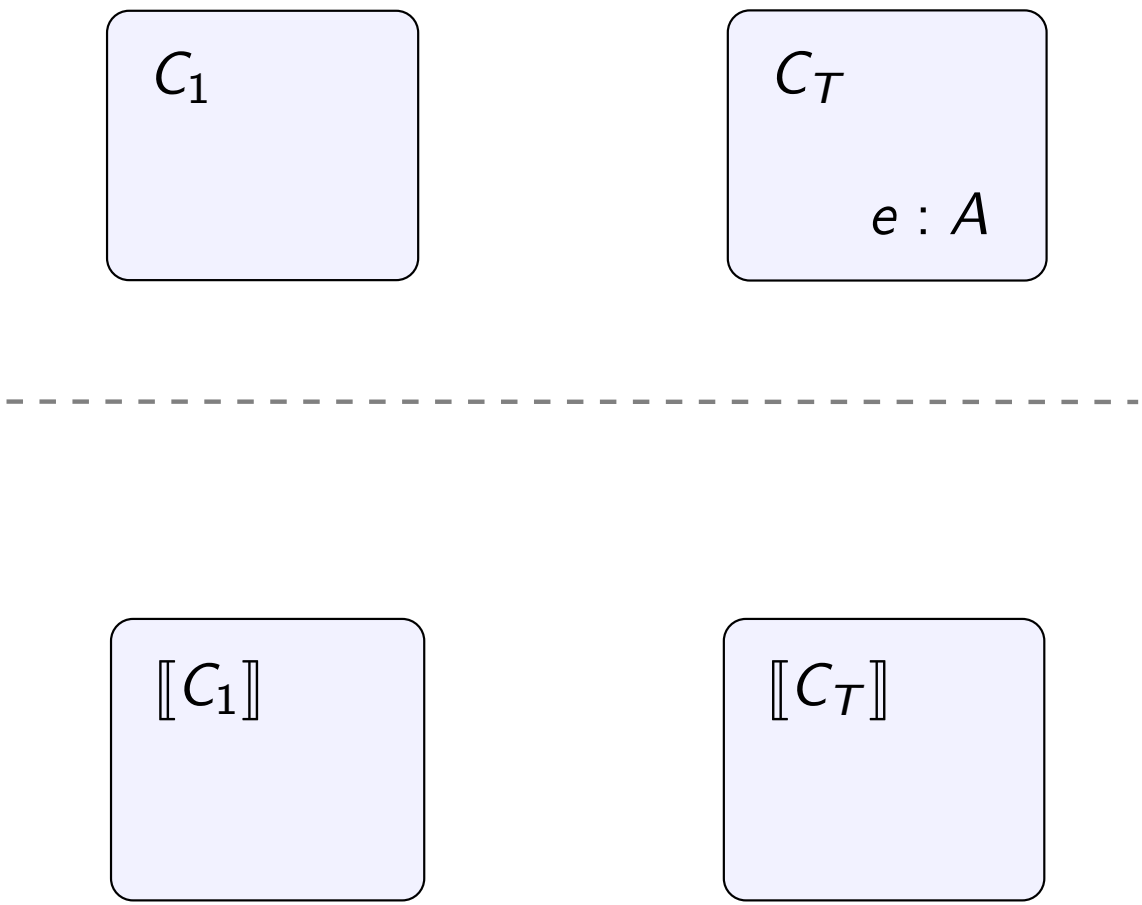
Object Guessing



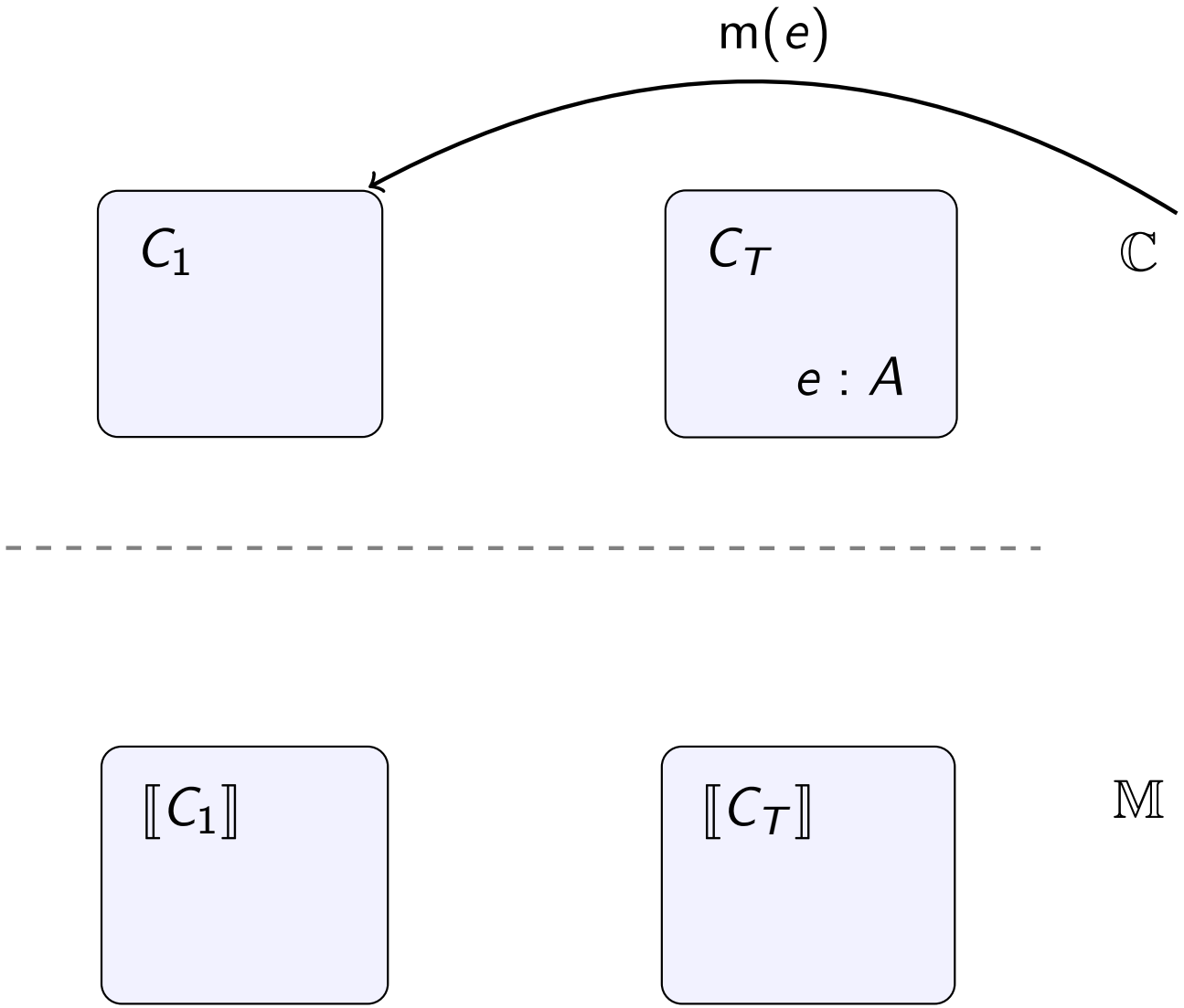
Object Faking



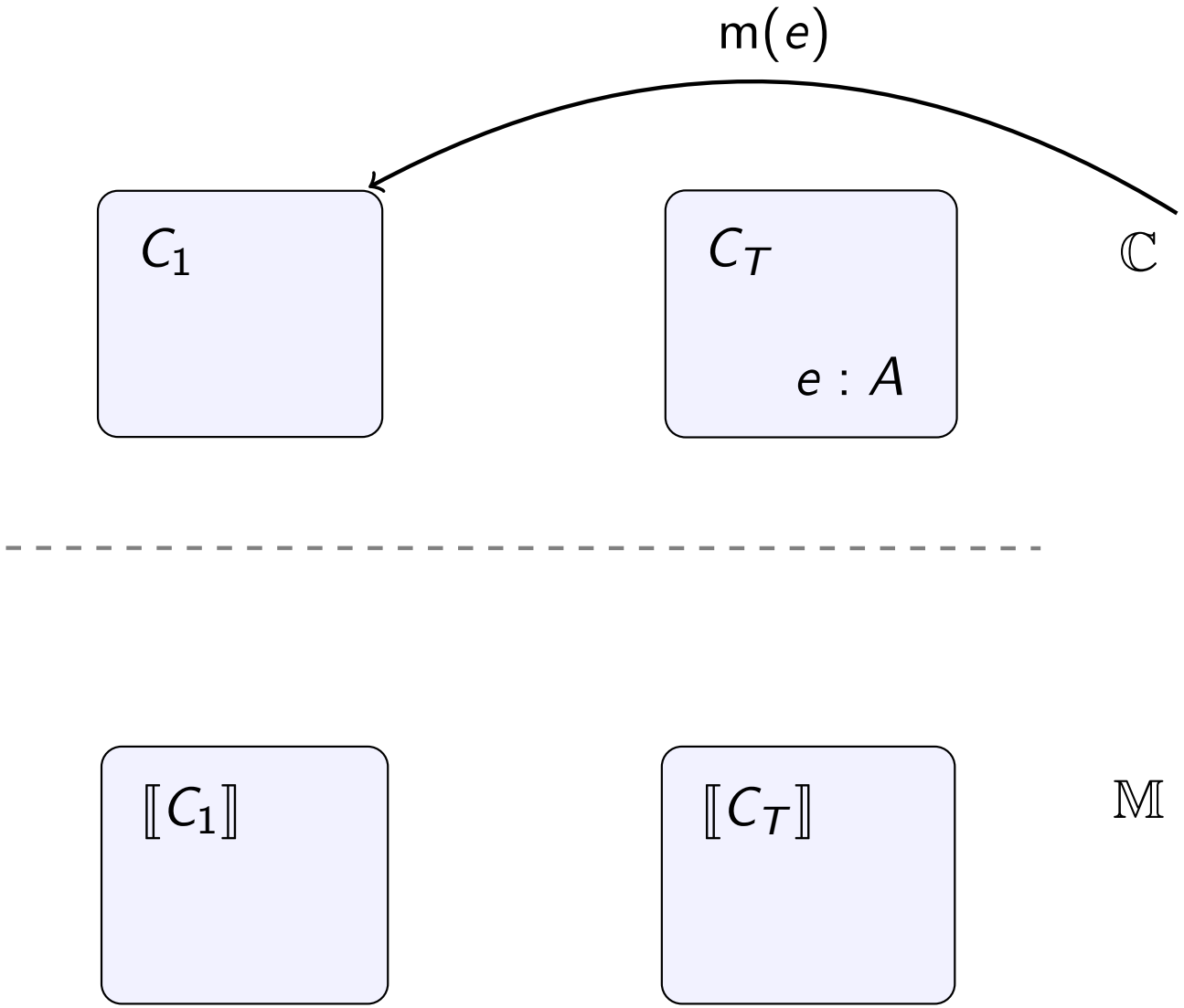
Object Faking



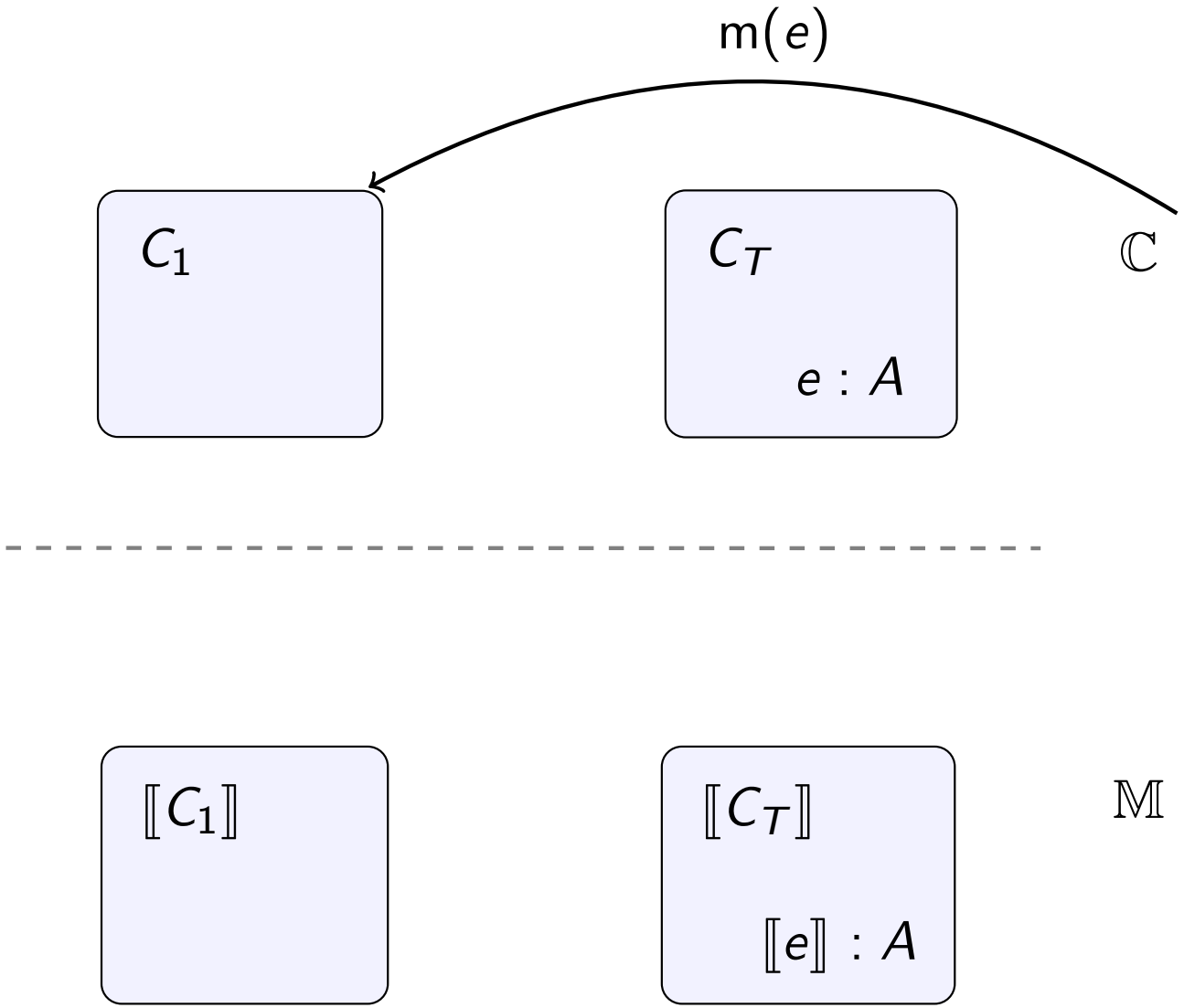
Object Faking



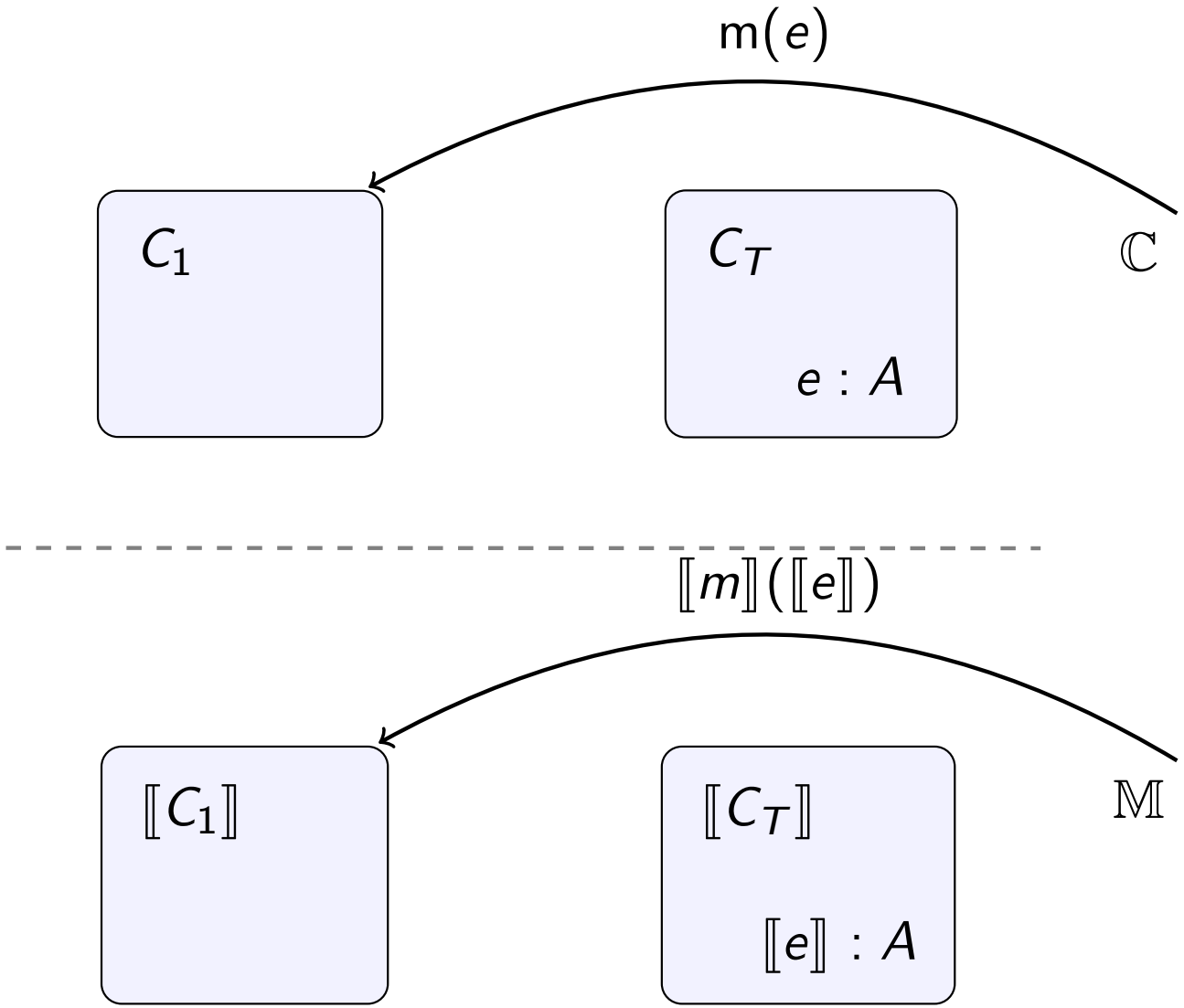
Object Faking



Object Faking

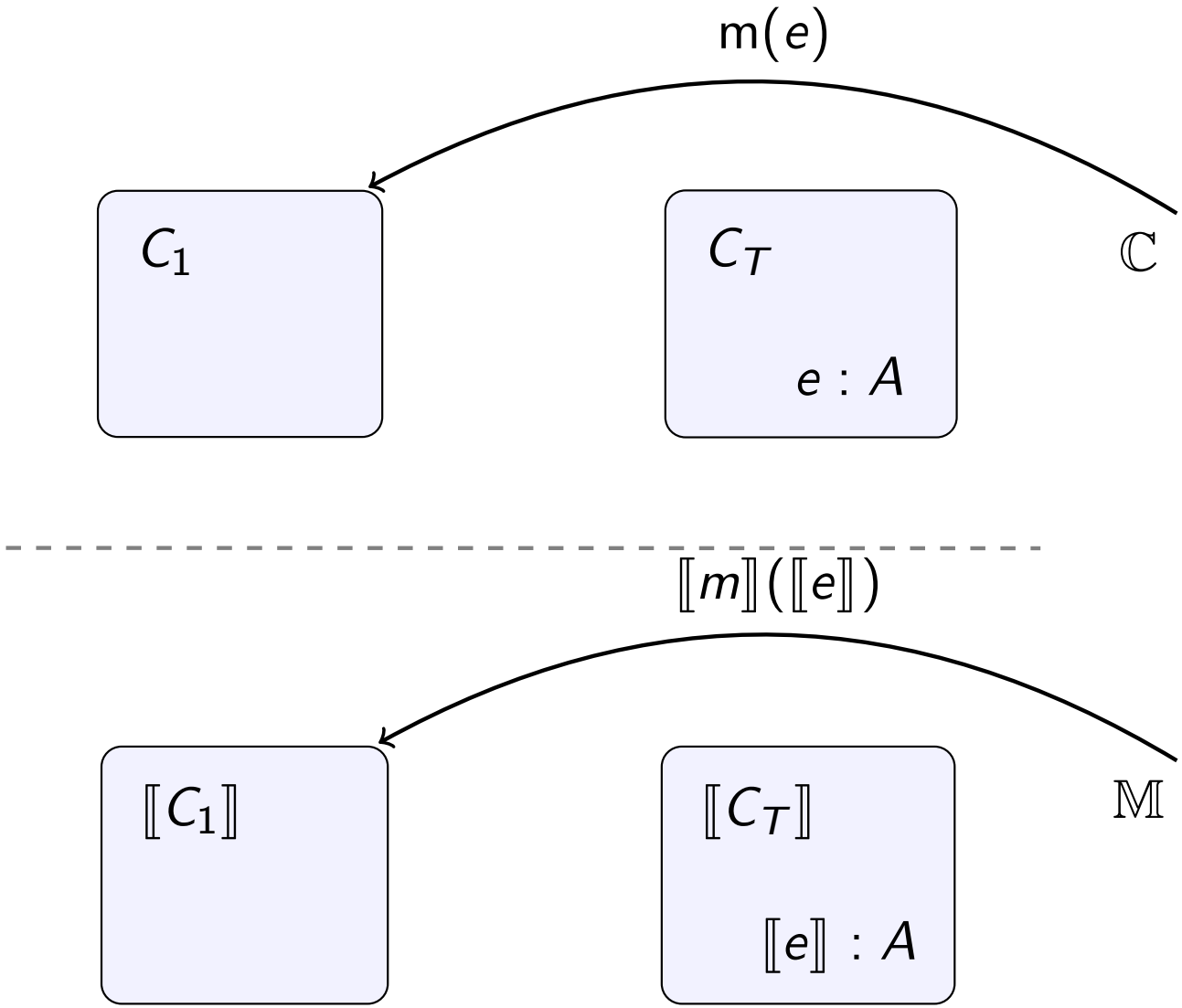


Object Faking



$e = \texttt{null}$
or obj : A

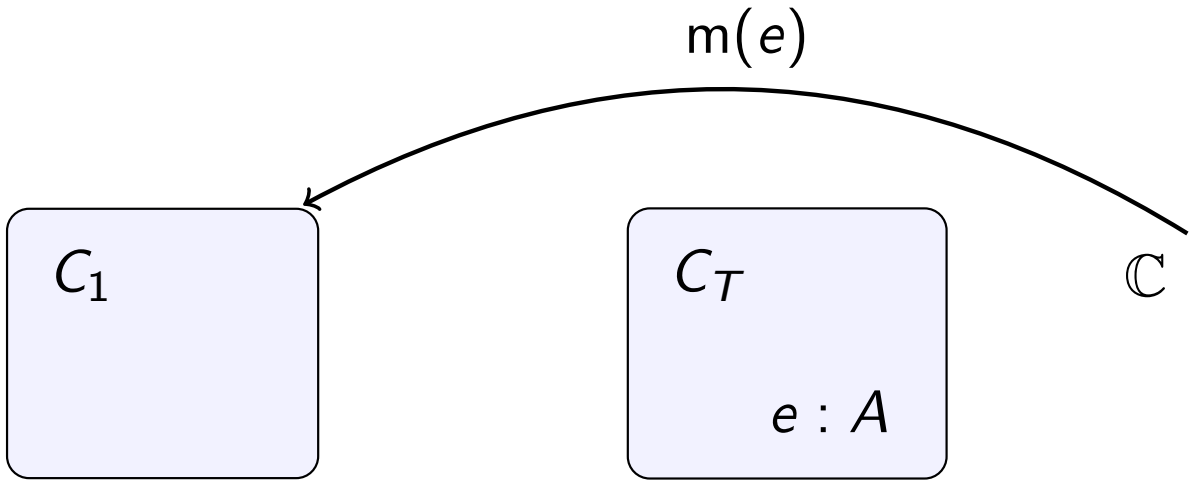
Object Faking



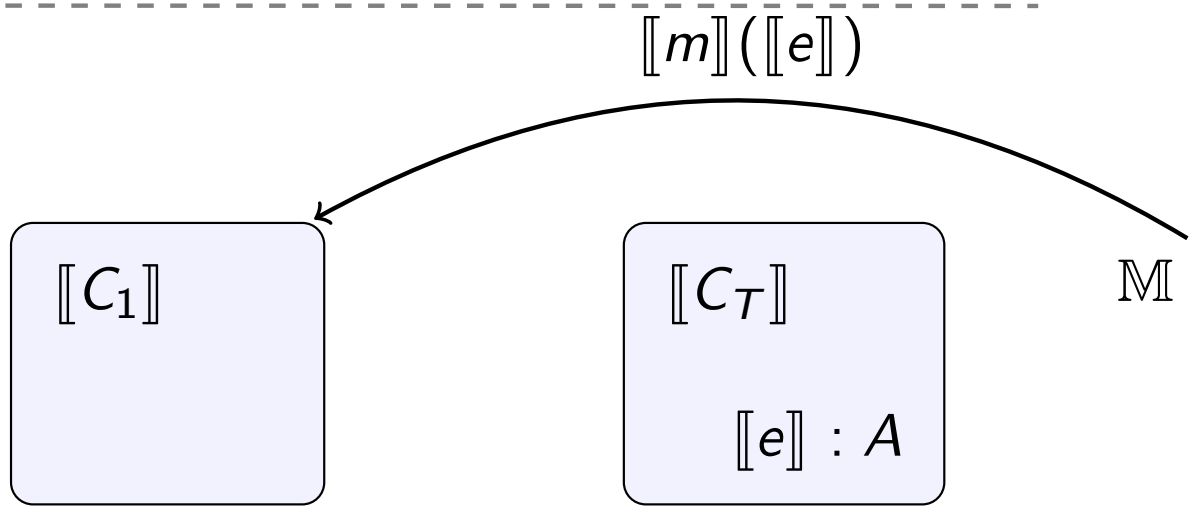
$e = \texttt{null}$
or obj : A

$[\![e]\!] = [\![\texttt{null}]\!]$
or obj : A

Object Faking



$e = \texttt{null}$
or obj : A



$[\![e]\!] = [\![\texttt{null}]\!]$
or obj : A
or obj : $B \not<: A$
or garbage

General Point

- problems always arising when two components are considered

General Point

- problems always arising when two components are considered
- no existing secure compiler deals with linking between compiled components

Outline

Solutions

- Protected modules architectures

Solutions

- Protected modules architectures [read, Intel SGX]

Solutions

- Protected modules architectures [read, Intel SGX] (isolation at assembly)

Solutions

- Protected modules architectures [read, Intel SGX] (isolation at assembly)
- Capability machines

Solutions

- Protected modules architectures [read, Intel SGX] (isolation at assembly)
- Capability machines (subject/objects/operations at assembly)

Solutions

- Protected modules architectures
- Capability machines (subject/objects/operations at assembly)

What is PMA?

- deep encapsulation at the lowest level of abstraction

What is PMA?

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

What is PMA?

- deep encapsulation at the lowest level of abstraction
- the basis of several security-related works
- Intel is porting it into future processors (SGX)

Untyped Assembly + PMA

```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
  ⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0xeb54
0x0b55    ···
  ⋮
0xab00    jmp 0x0b53
  ⋮
0xeb52    movs r0 0xeb54
0xeb53    call 0xab02
0xeb54    ···
  ⋮
```

Untyped Assembly + PMA

```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
   ⋮
0x0b52    movs r0 0x0b55
0x0b53    call 0x0002
0x0b54    movs r0 0xeb54
0x0b55    ...
   ⋮
0xab00    jmp 0x0b53
   ⋮
0xeb52    movs r0 0xeb54
0xeb53    call 0xab02
0xeb54    ...
   ⋮
```

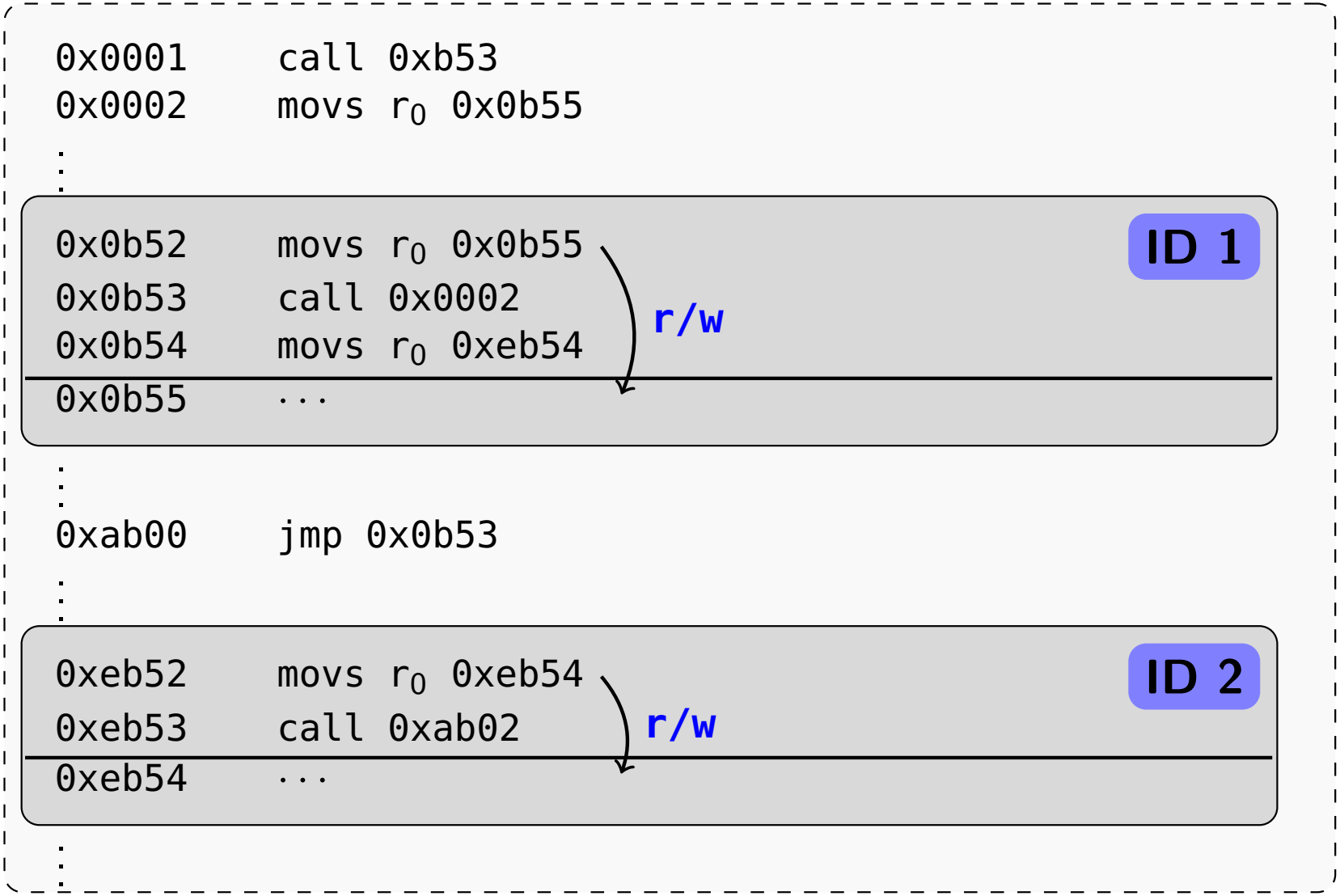
ID 1

ID 2

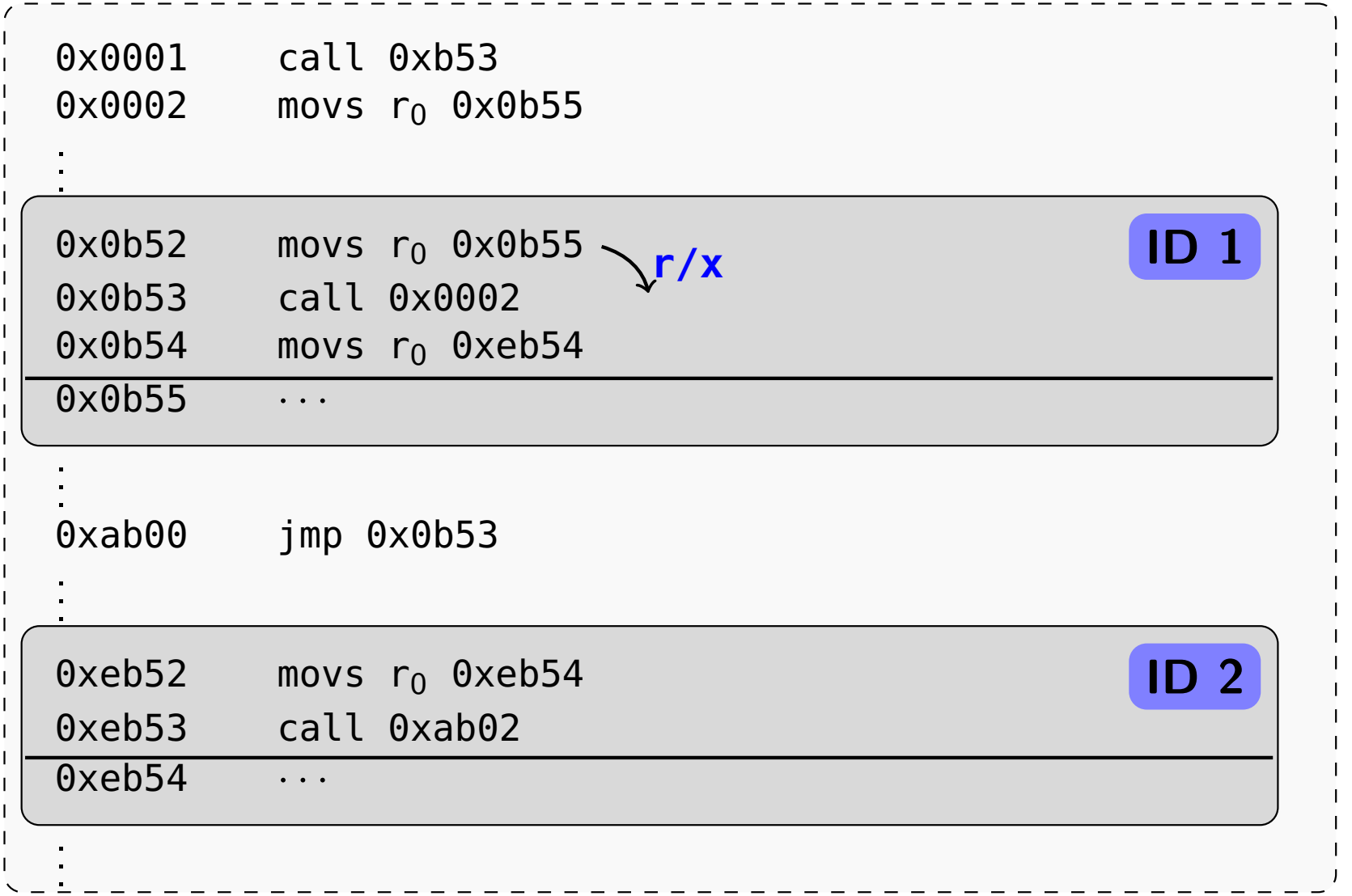
Untyped Assembly + PMA

```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
   ⋮
0x0b52    movs r0 0x0b55        ID 1
0x0b53    call 0x0002
0x0b54    movs r0 0xeb54
0x0b55    ···
   ⋮
0xab00    jmp 0x0b53
   ⋮
0xeb52    movs r0 0xeb54        ID 2
0xeb53    call 0xab02
0xeb54    ···
   ⋮
```

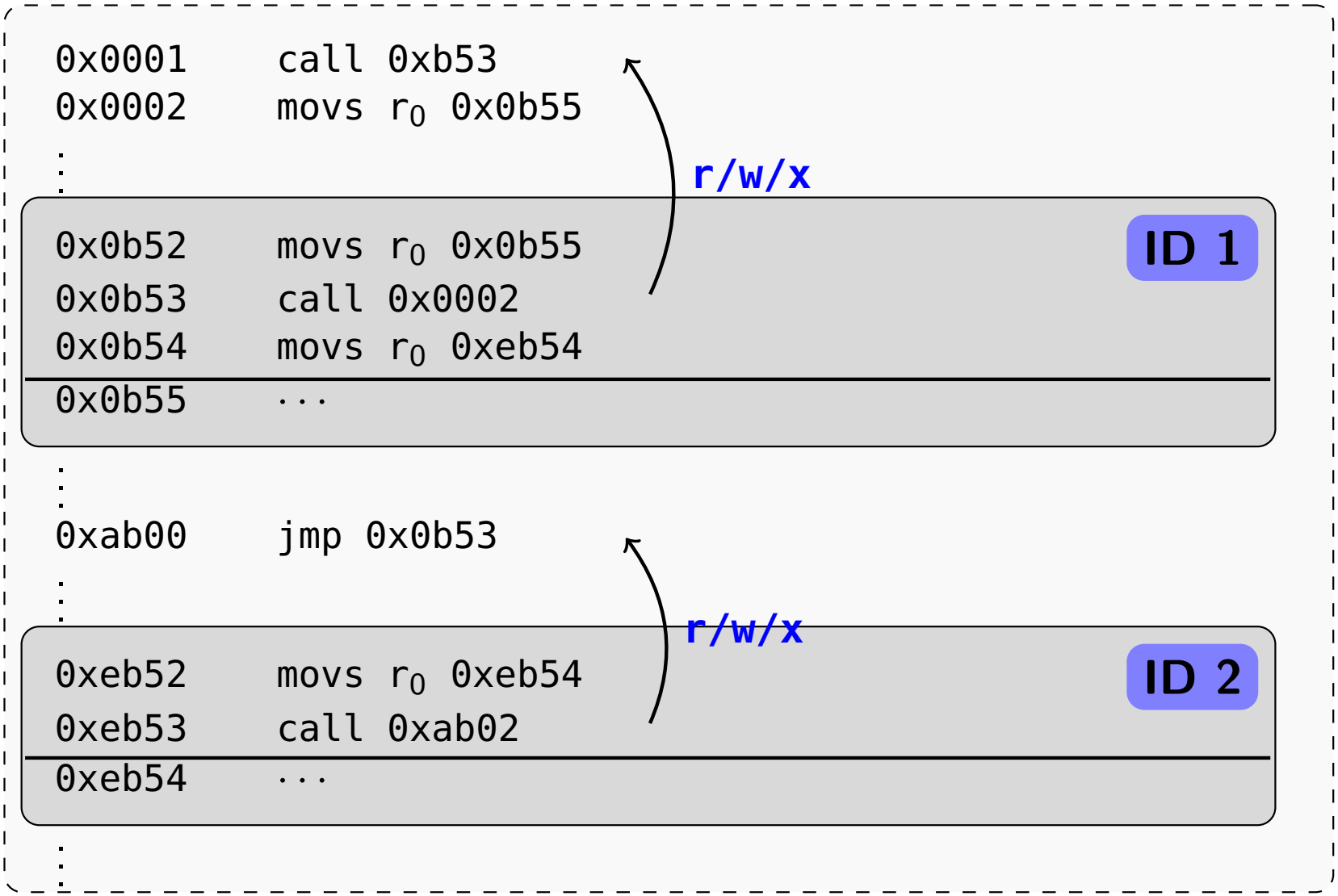
Untyped Assembly + PMA



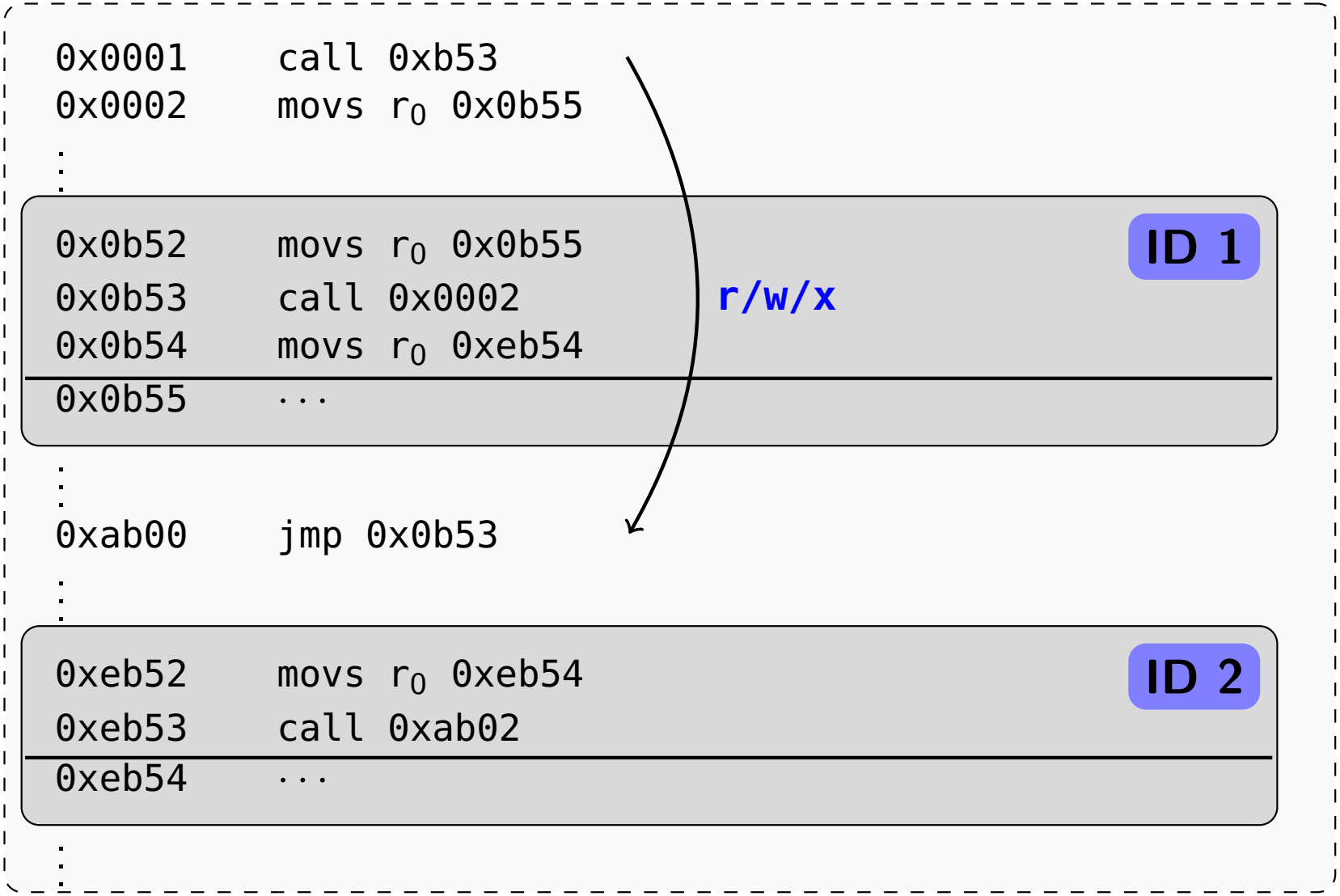
Untyped Assembly + PMA



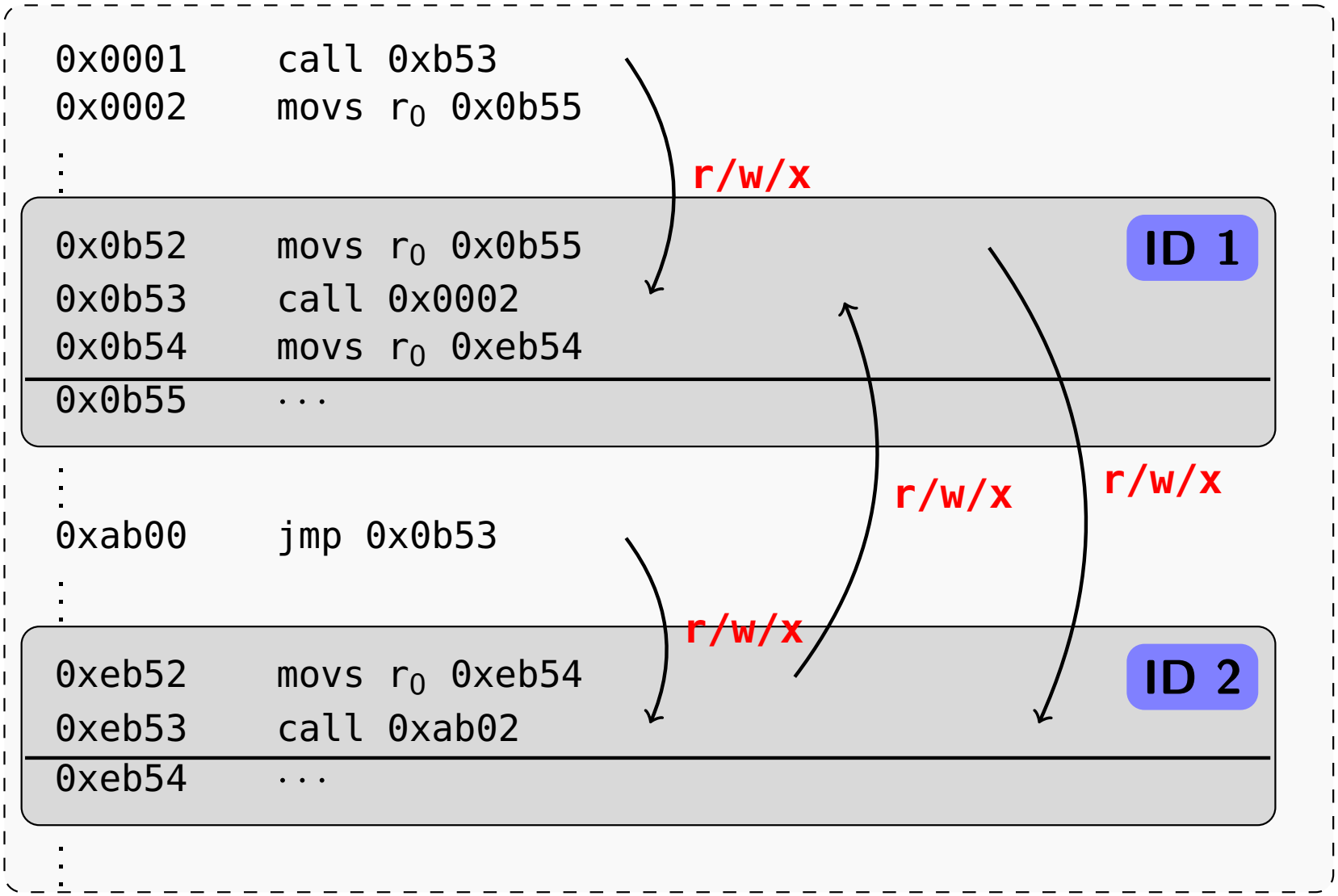
Untyped Assembly + PMA



Untyped Assembly + PMA



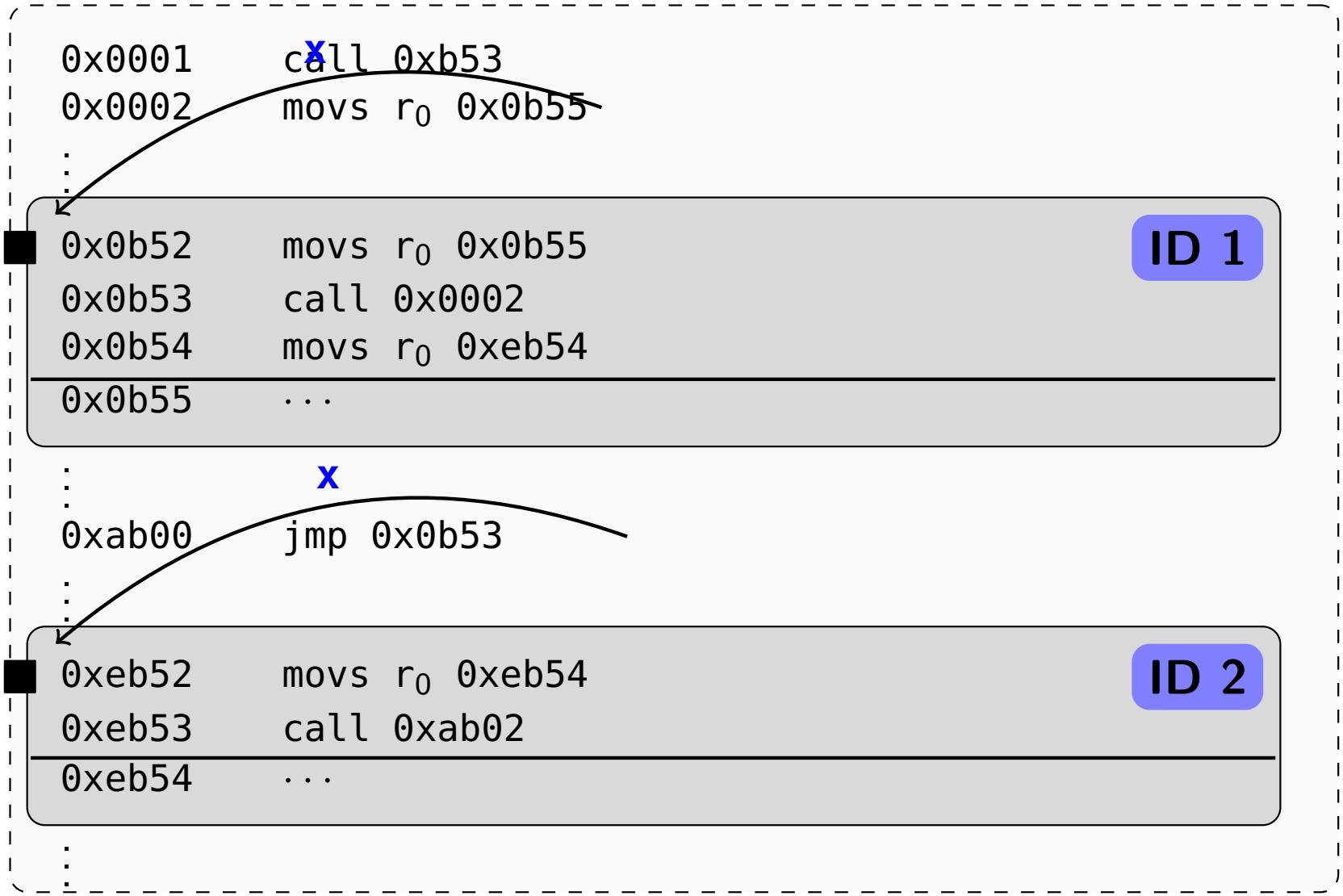
Untyped Assembly + PMA



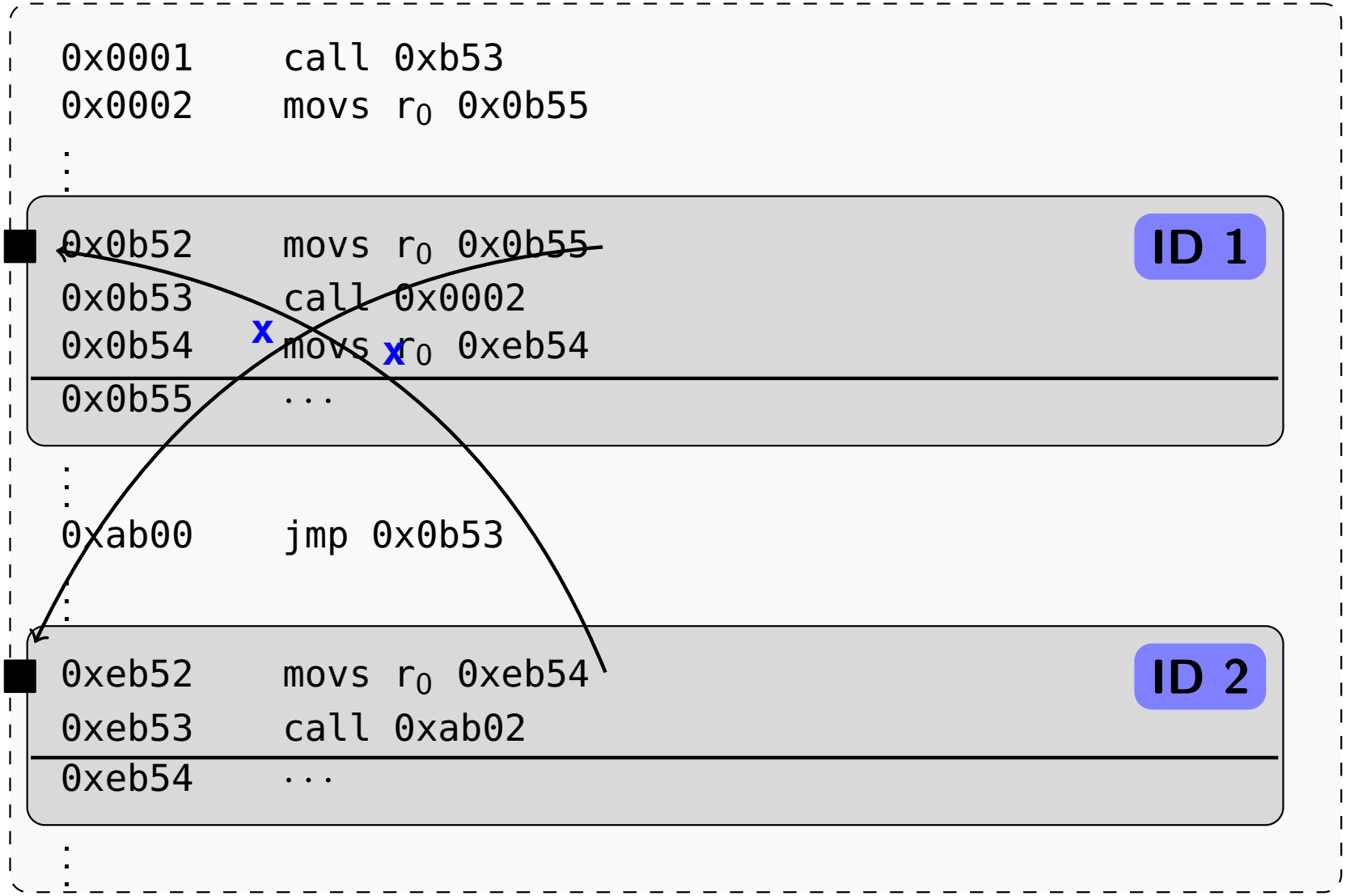
Untyped Assembly + PMA

```
0x0001    call 0xb53
0x0002    movs r0 0x0b55
   ⋮
0x0b52    movs r0 0x0b55        ID 1
0x0b53    call 0x0002
0x0b54    movs r0 0xeb54
0x0b55    ...
   ⋮
0xab00    jmp 0x0b53
   ⋮
0xeb52    movs r0 0xeb54        ID 2
0xeb53    call 0xab02
0xeb54    ...
   ⋮
```

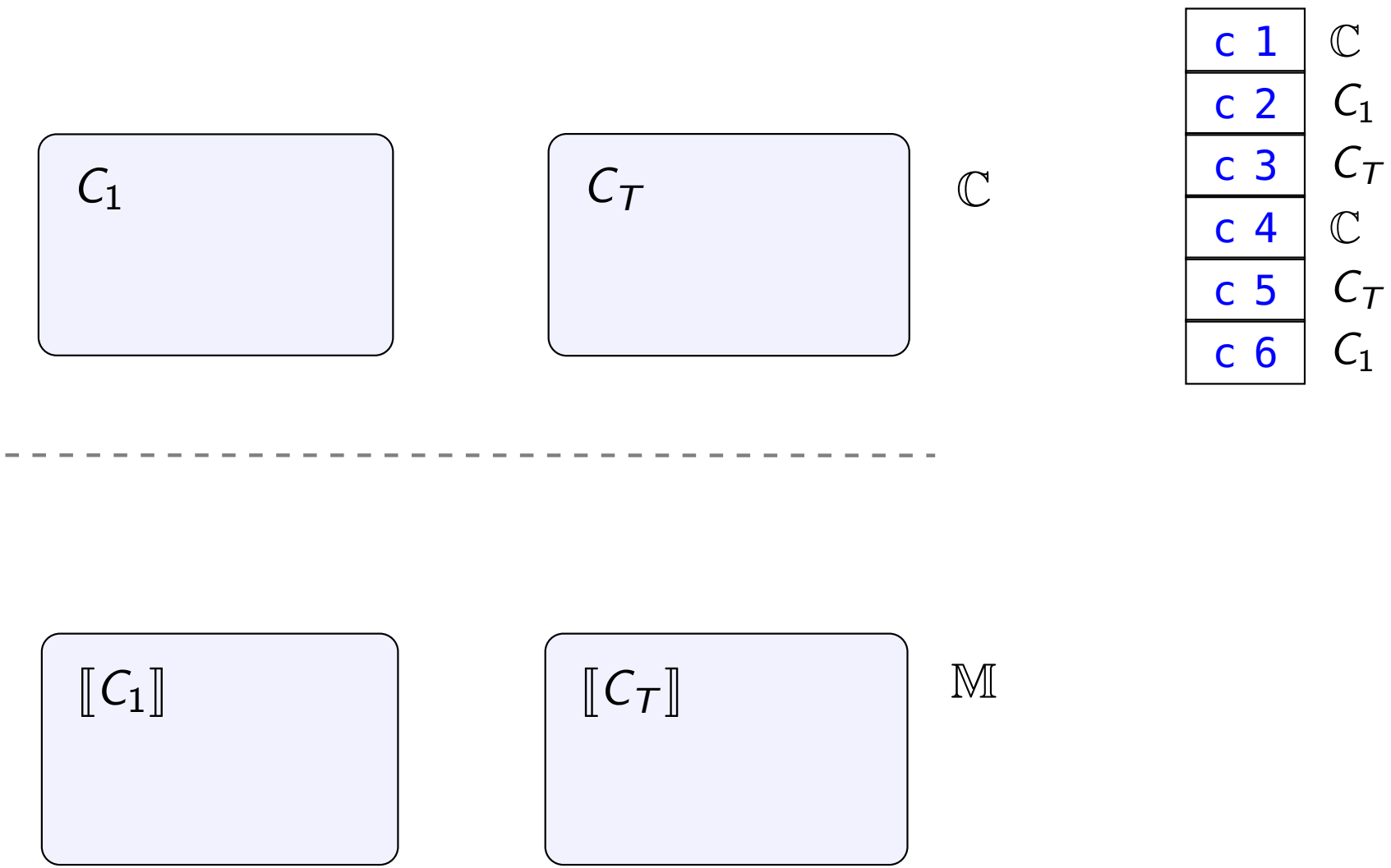
Untyped Assembly + PMA



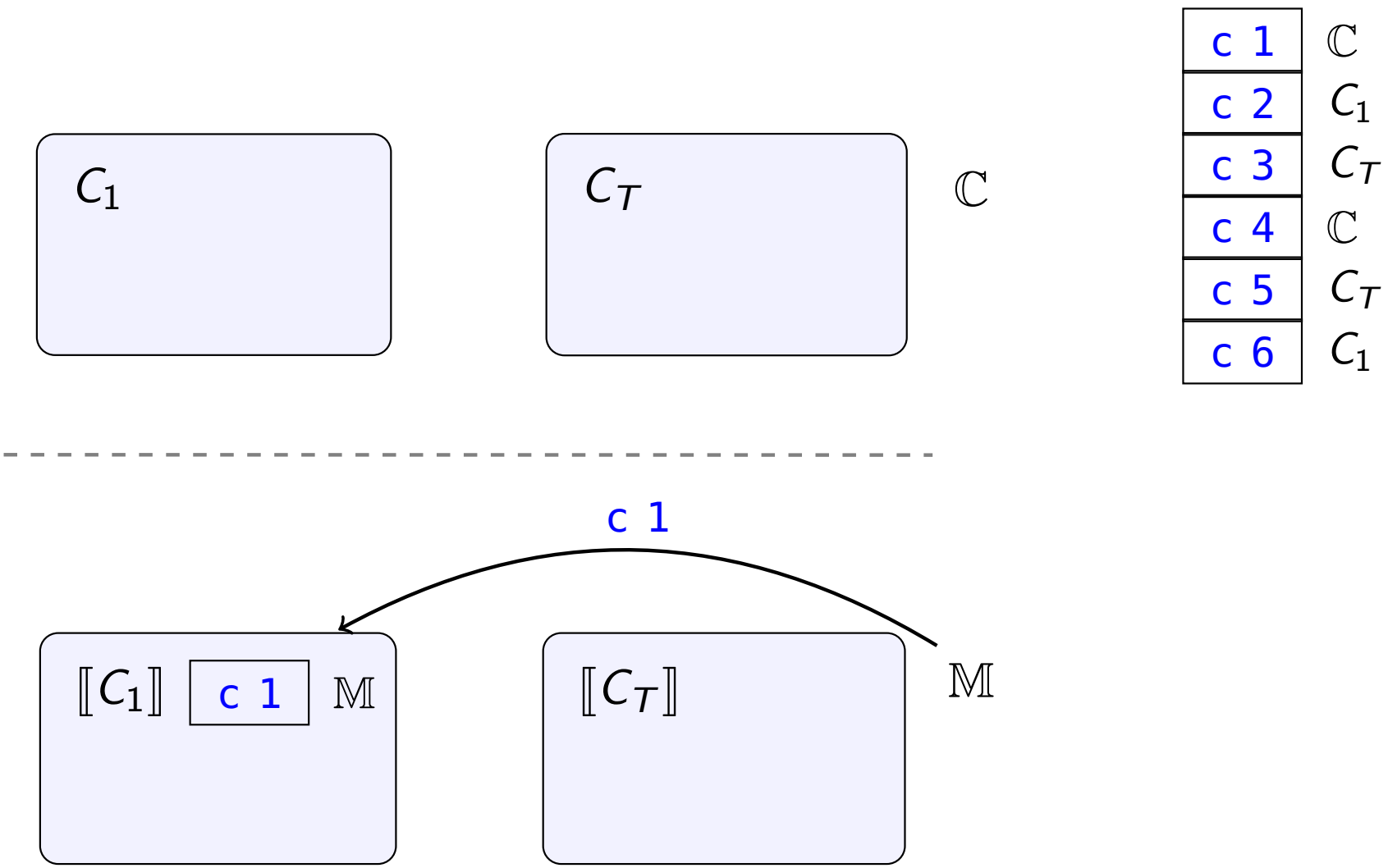
Untyped Assembly + PMA



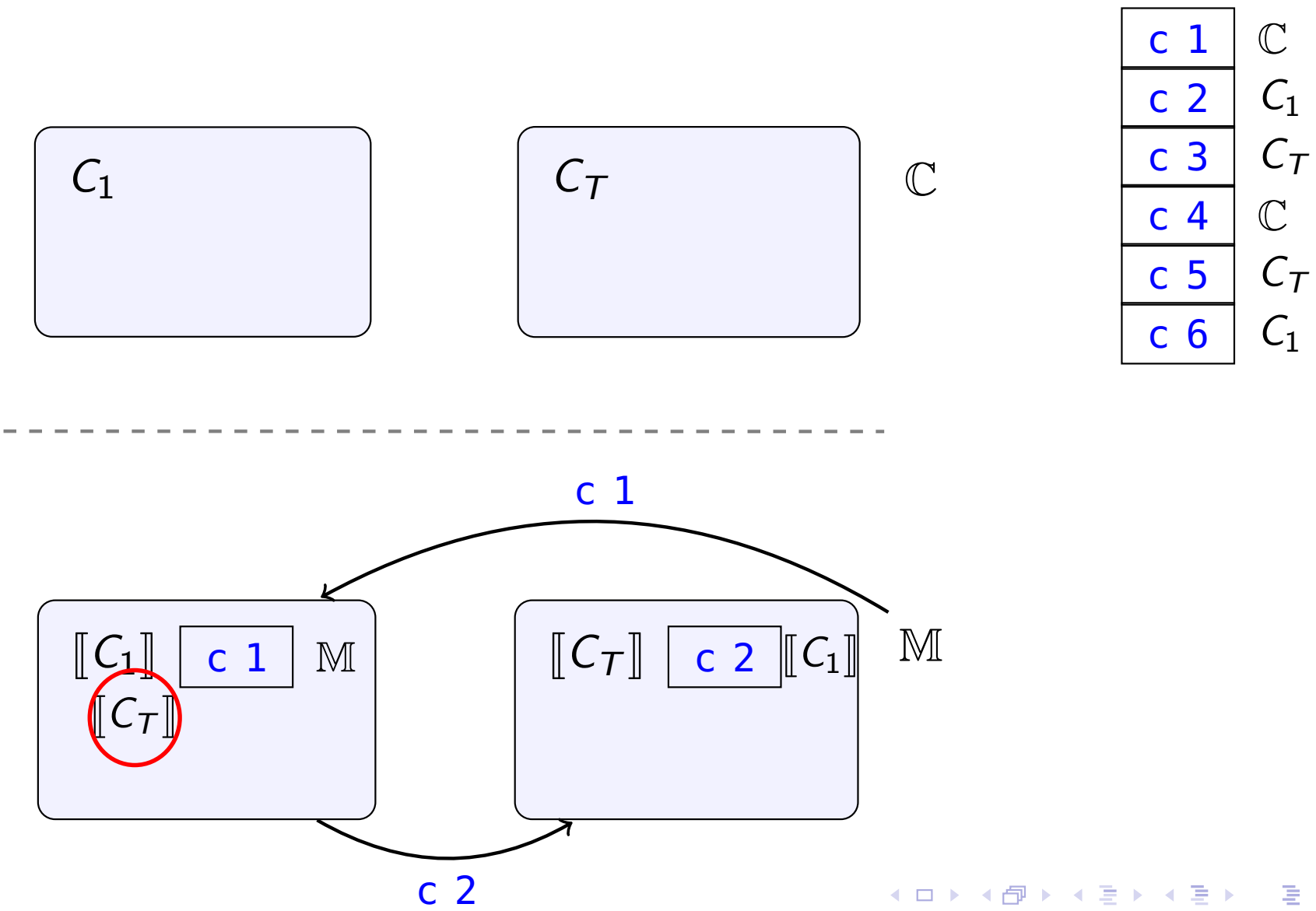
Addressing Call Stack Shortcutting



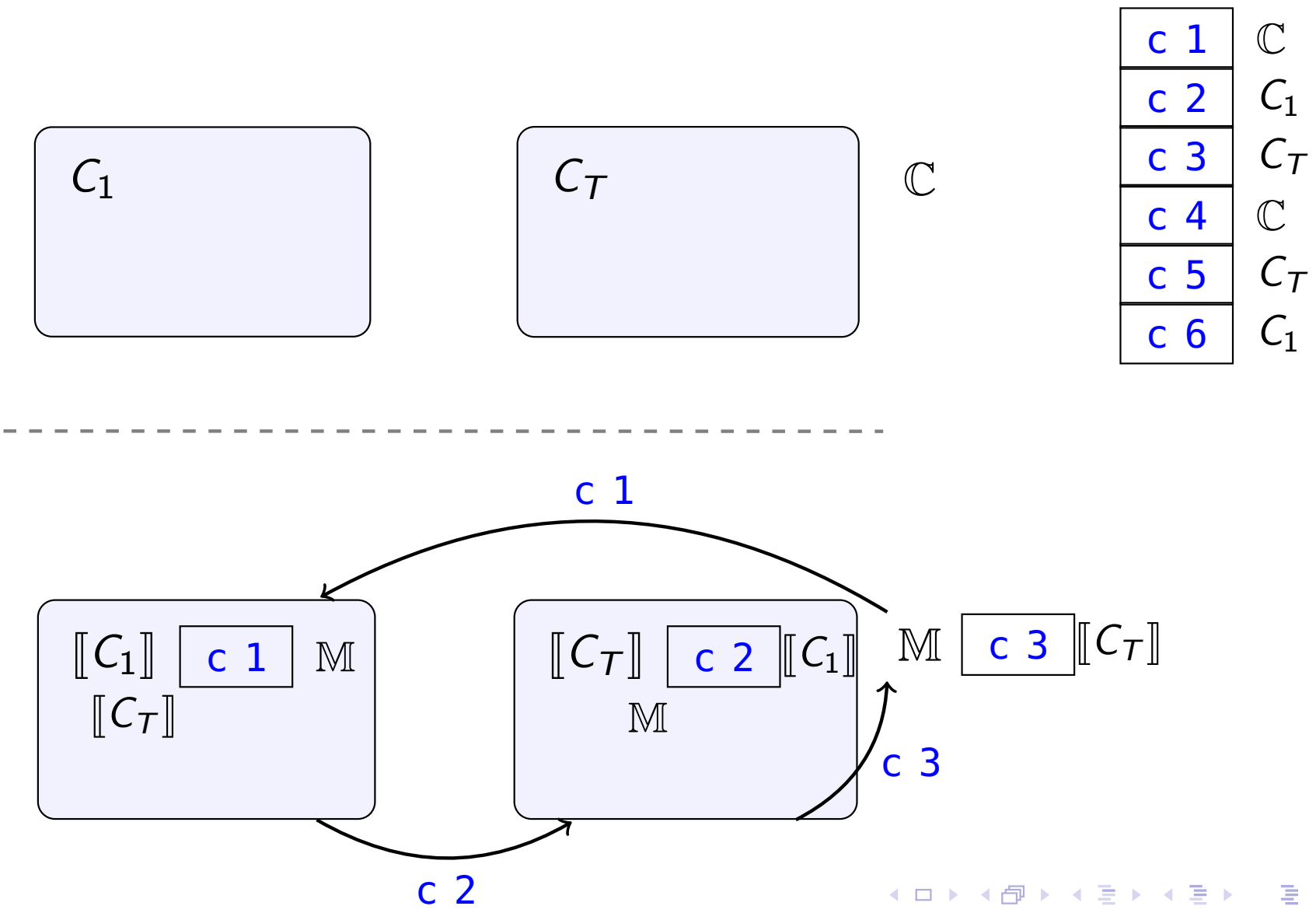
Addressing Call Stack Shortcutting



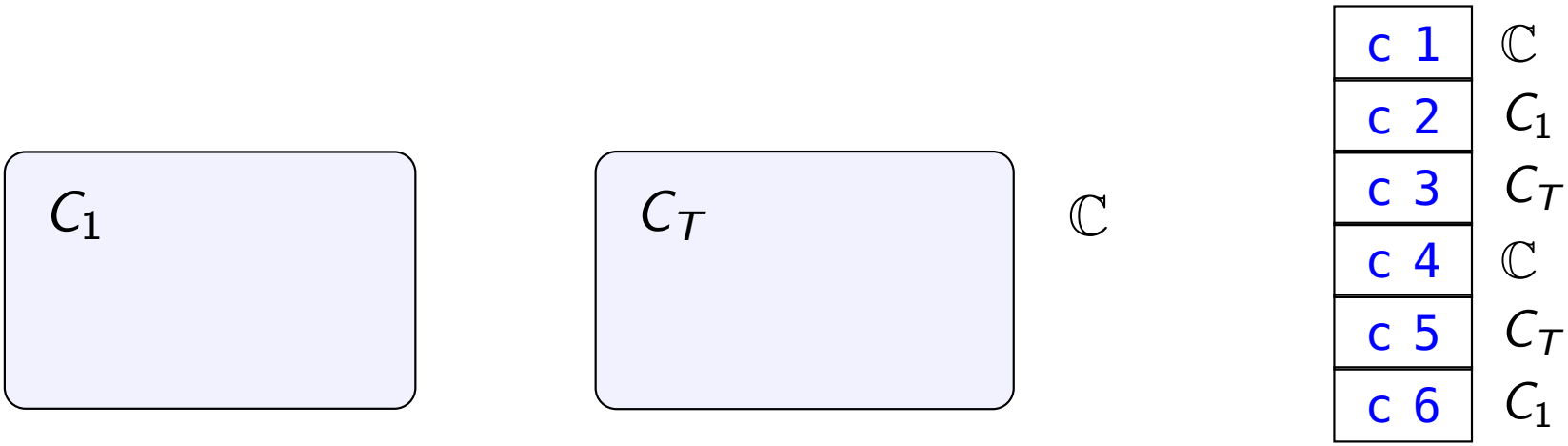
Addressing Call Stack Shortcutting



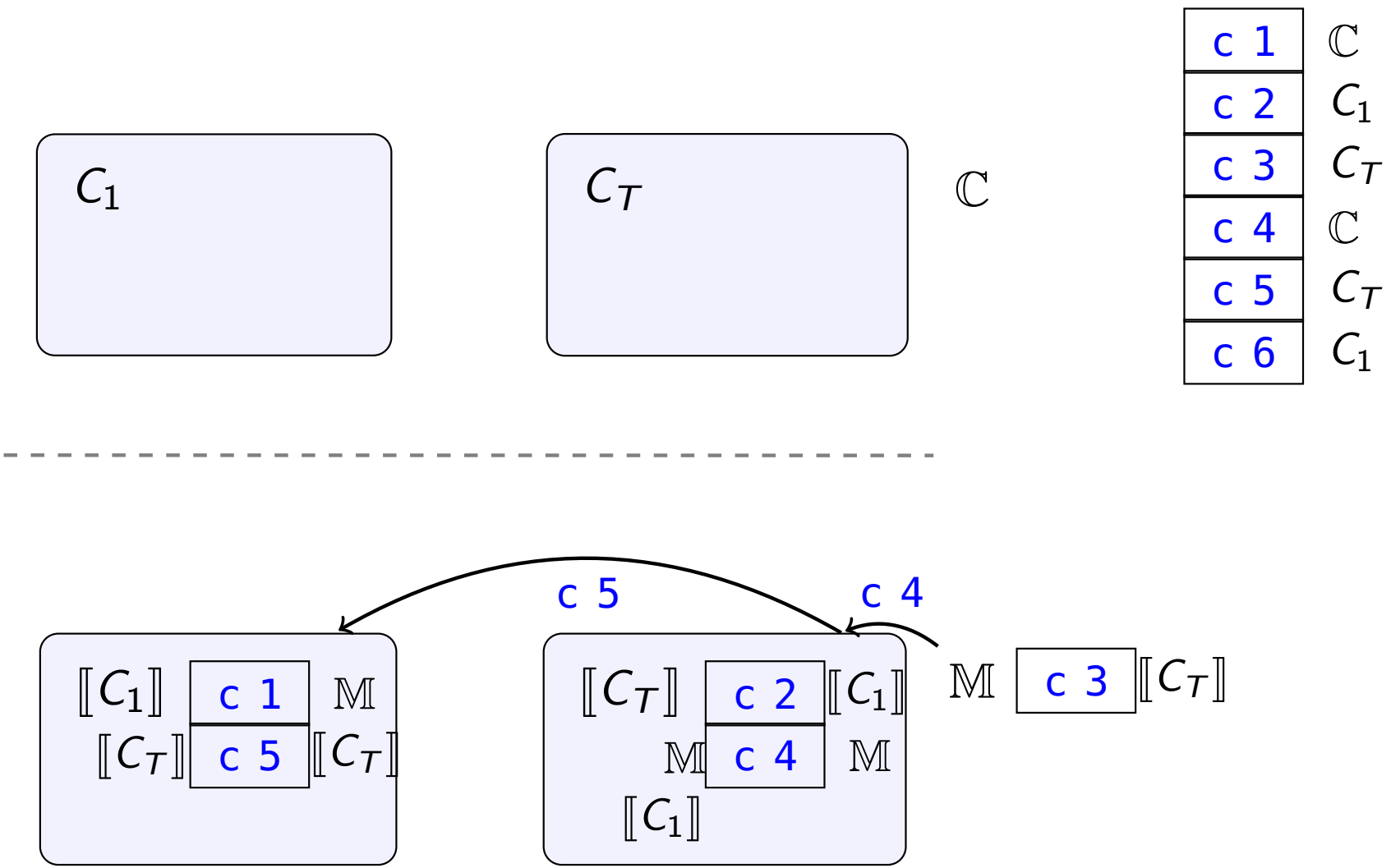
Addressing Call Stack Shortcutting



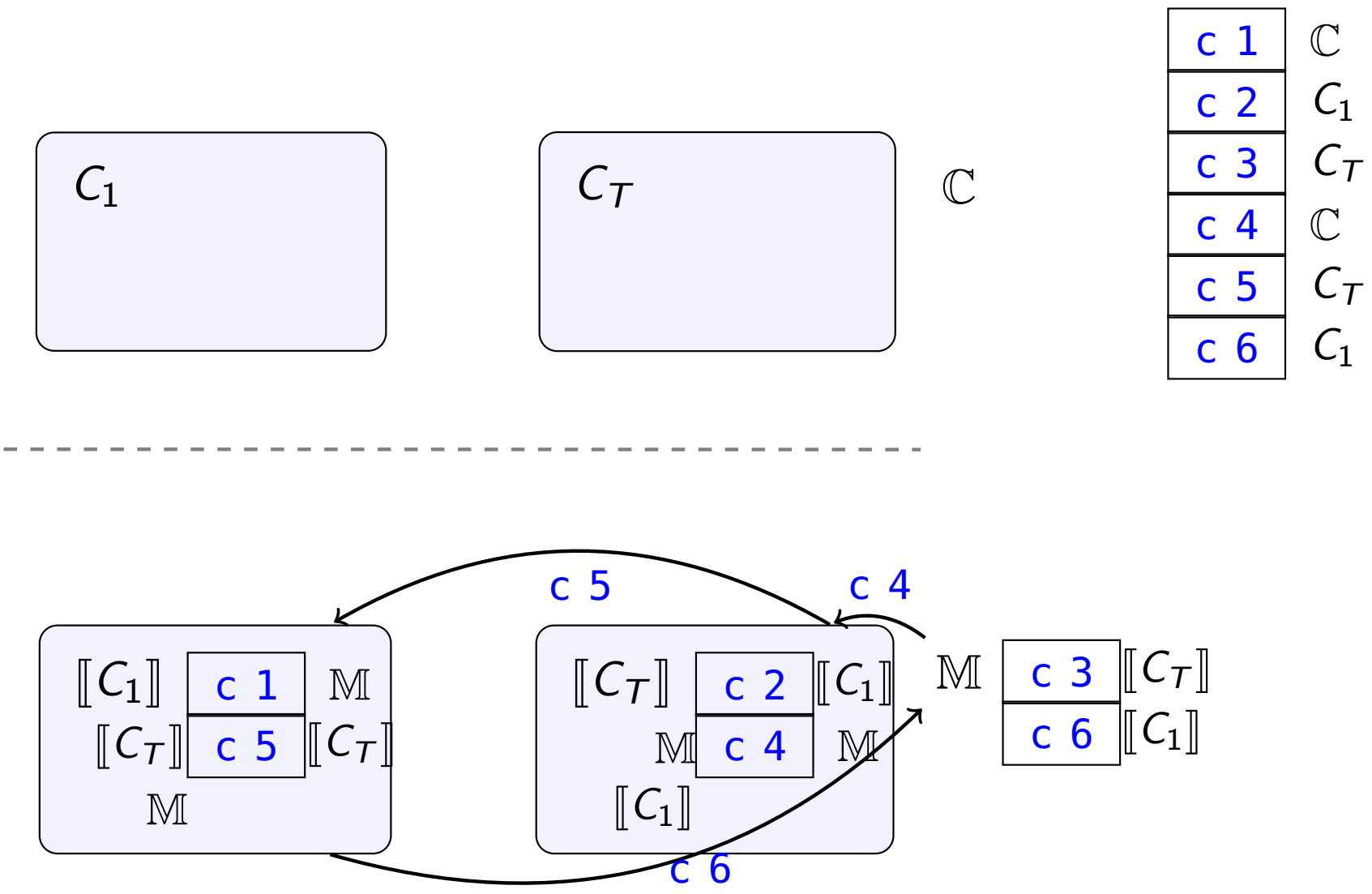
Addressing Call Stack Shortcutting



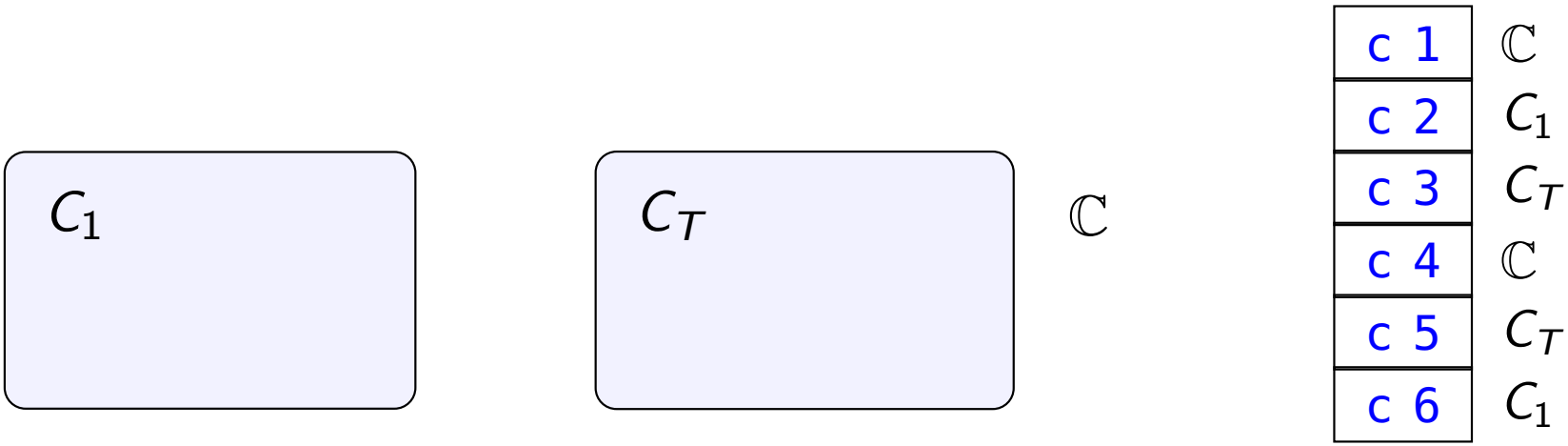
Addressing Call Stack Shortcutting



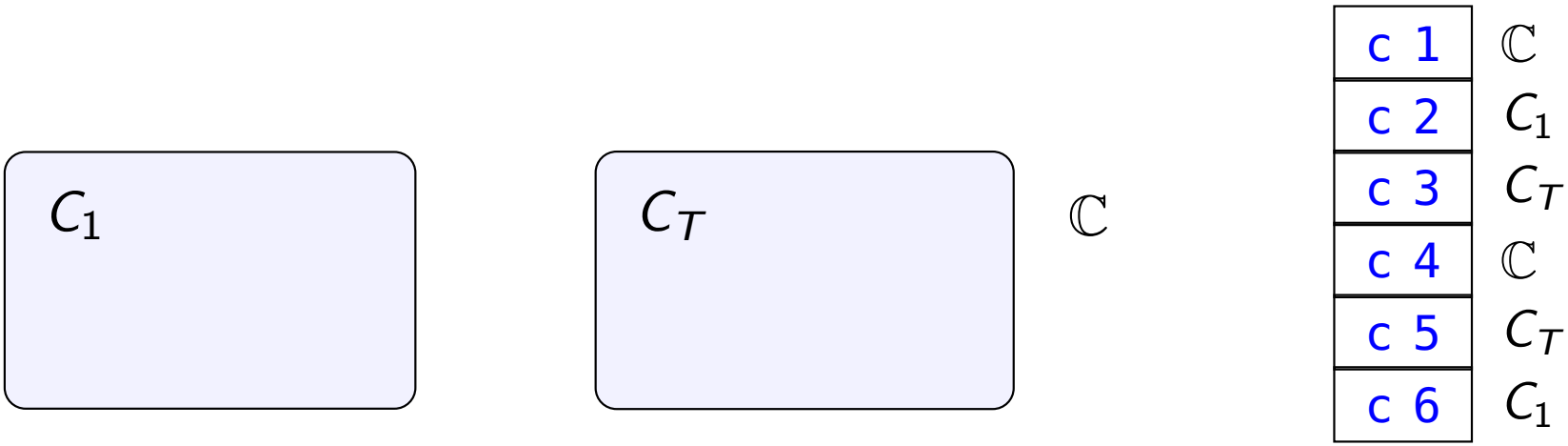
Addressing Call Stack Shortcutting



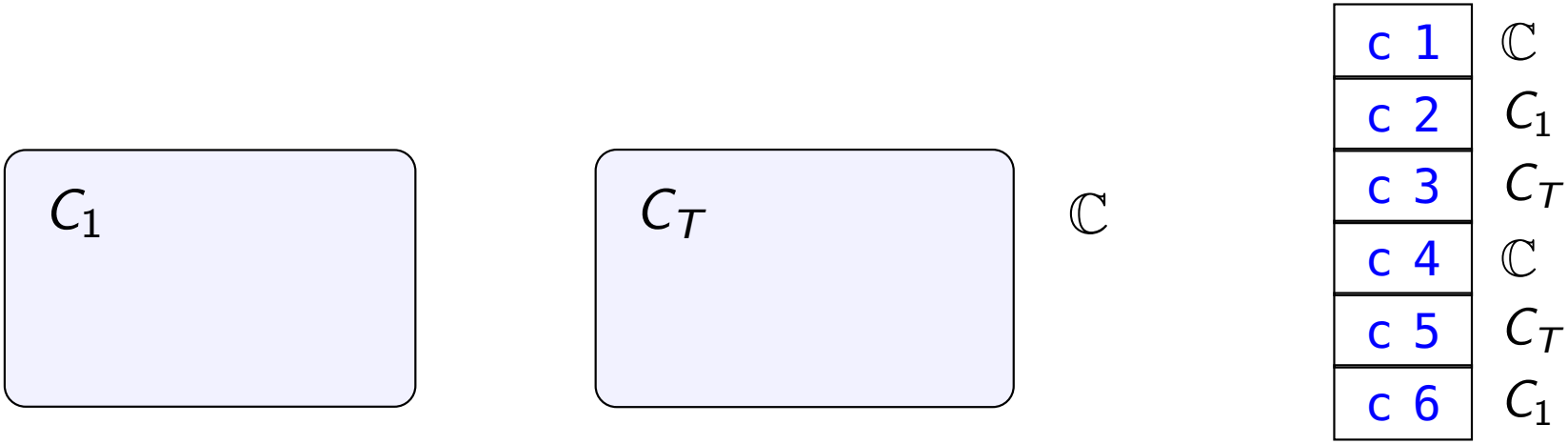
Addressing Call Stack Shortcutting



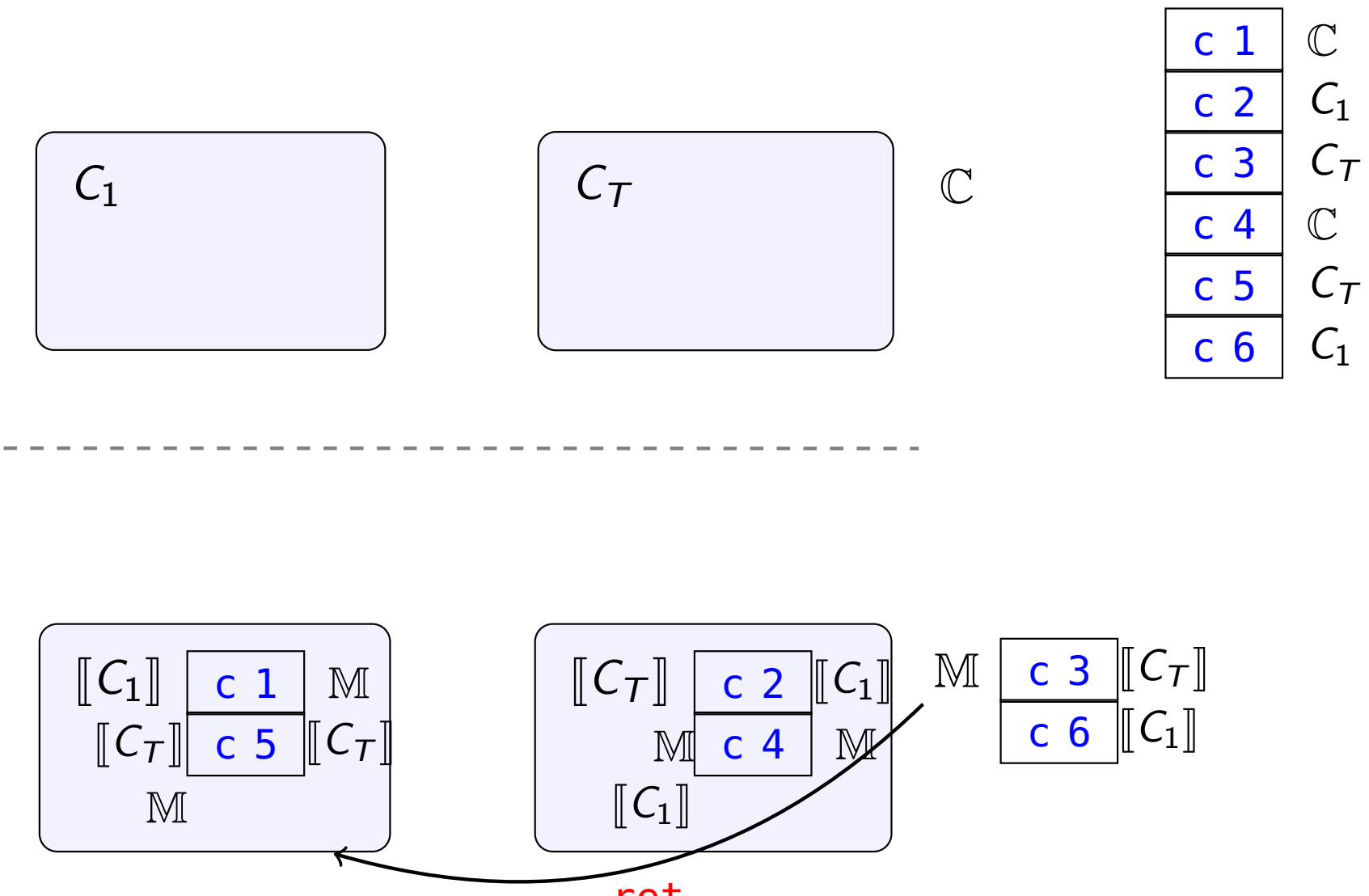
Addressing Call Stack Shortcutting



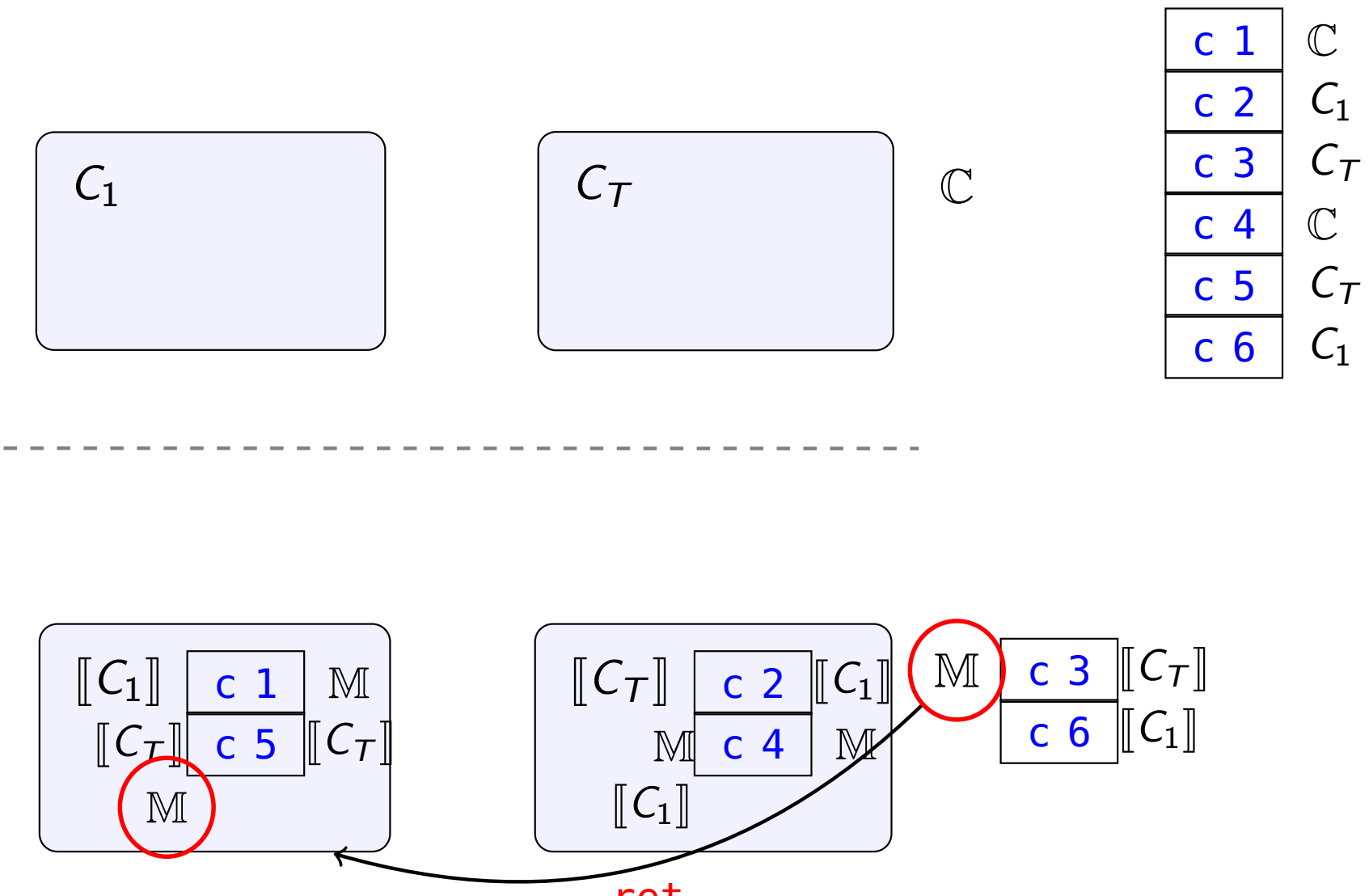
Addressing Call Stack Shortcutting



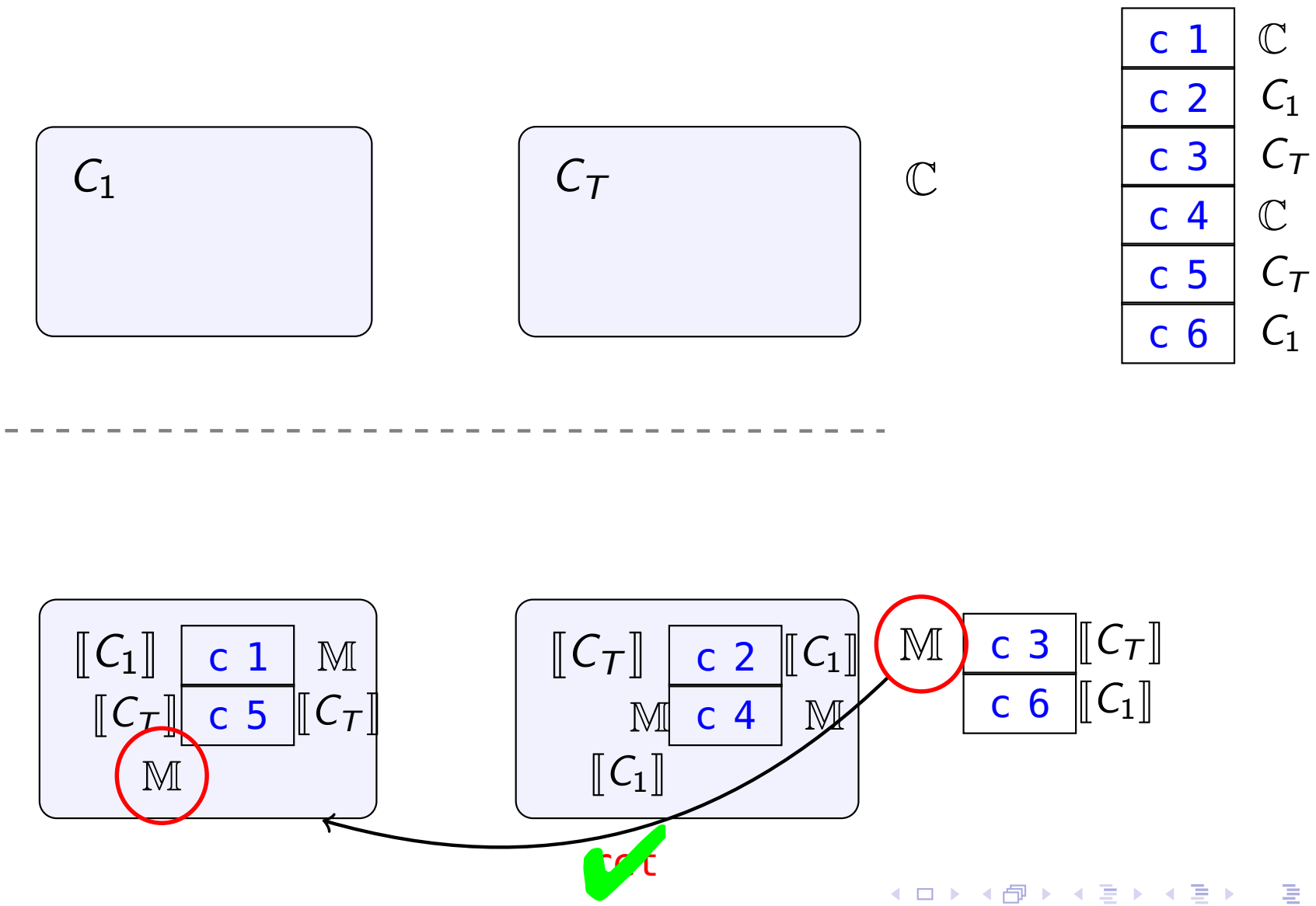
Addressing Call Stack Shortcutting



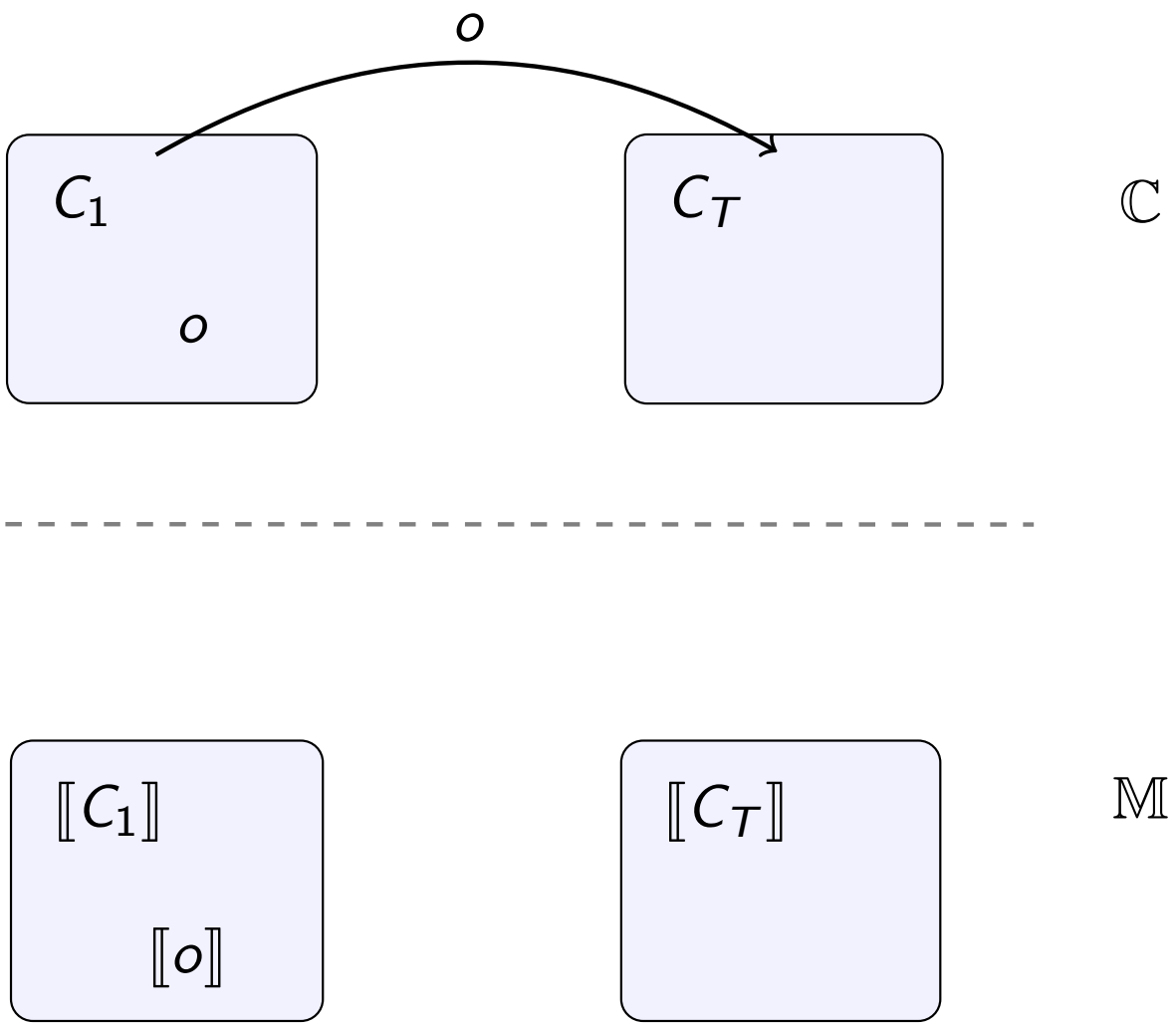
Addressing Call Stack Shortcutting



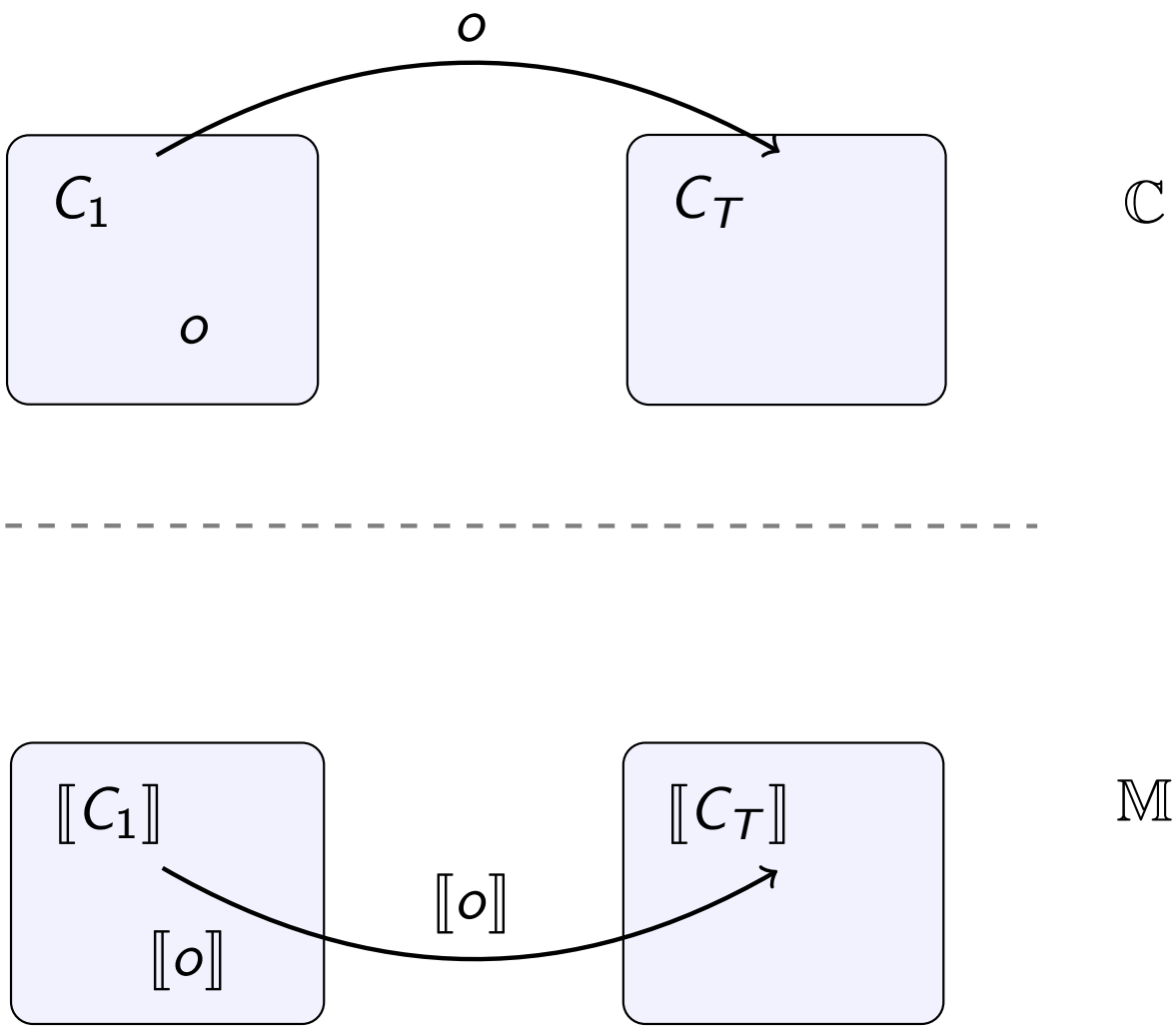
Addressing Call Stack Shortcutting



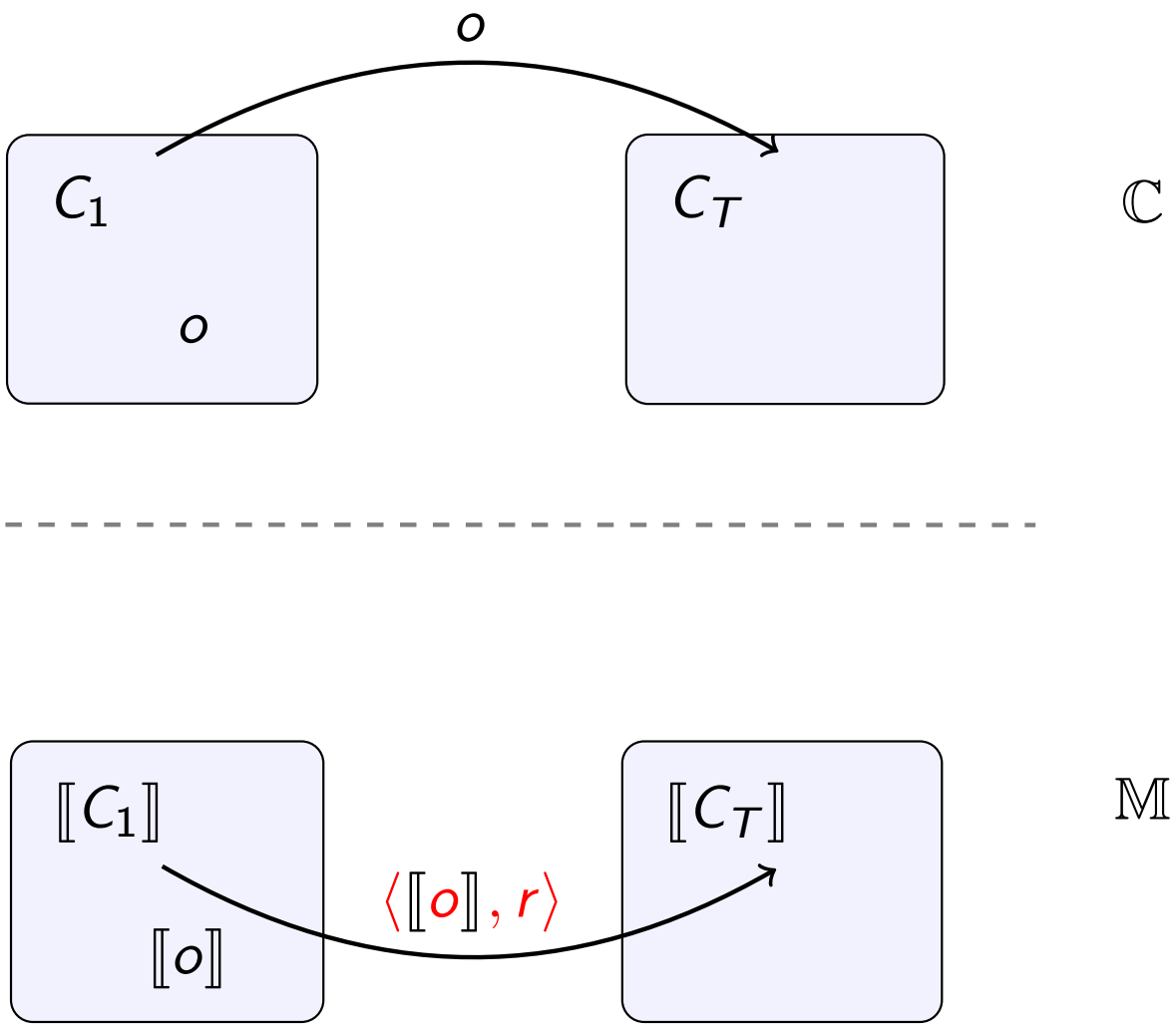
Addressing Object Guessing



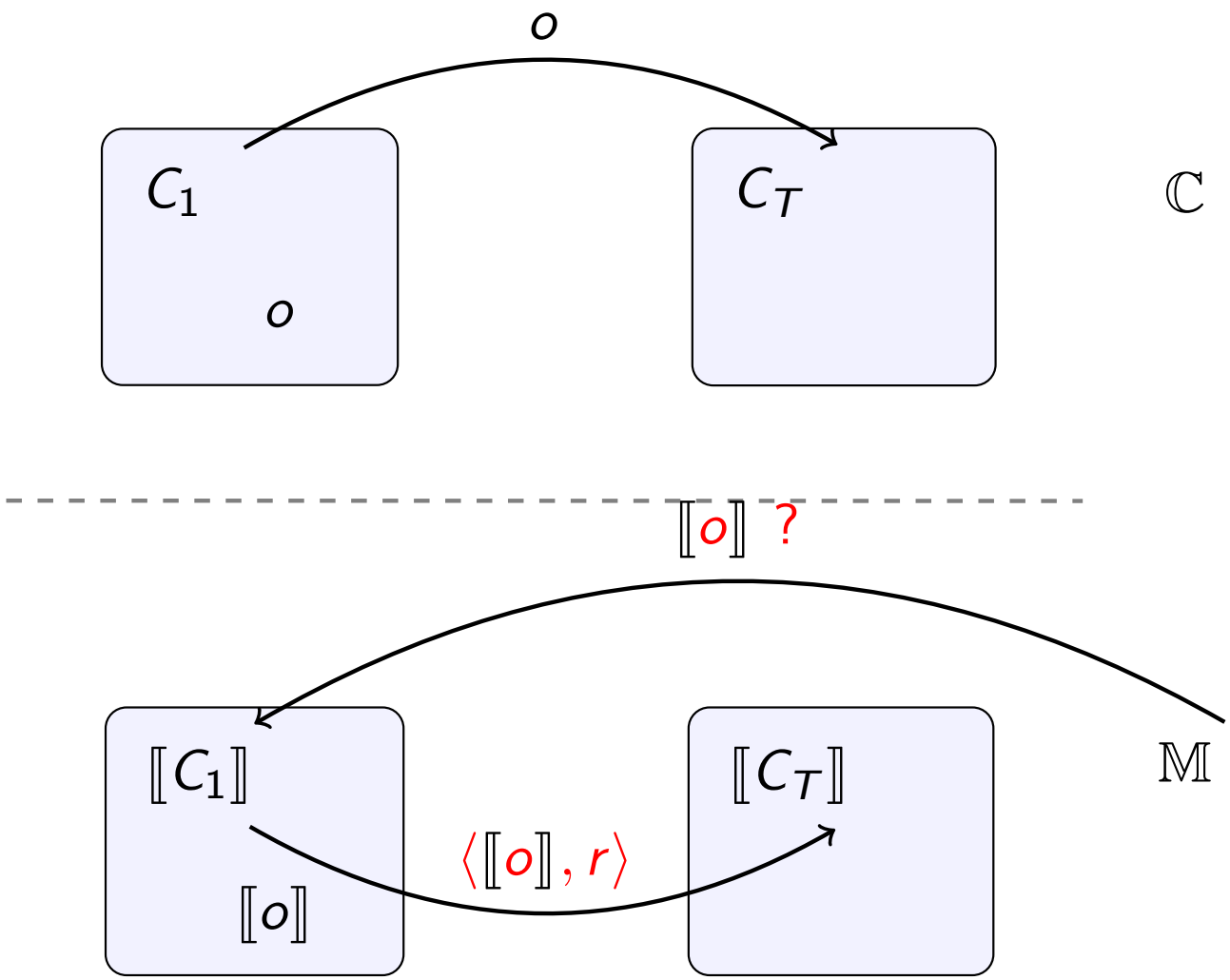
Addressing Object Guessing



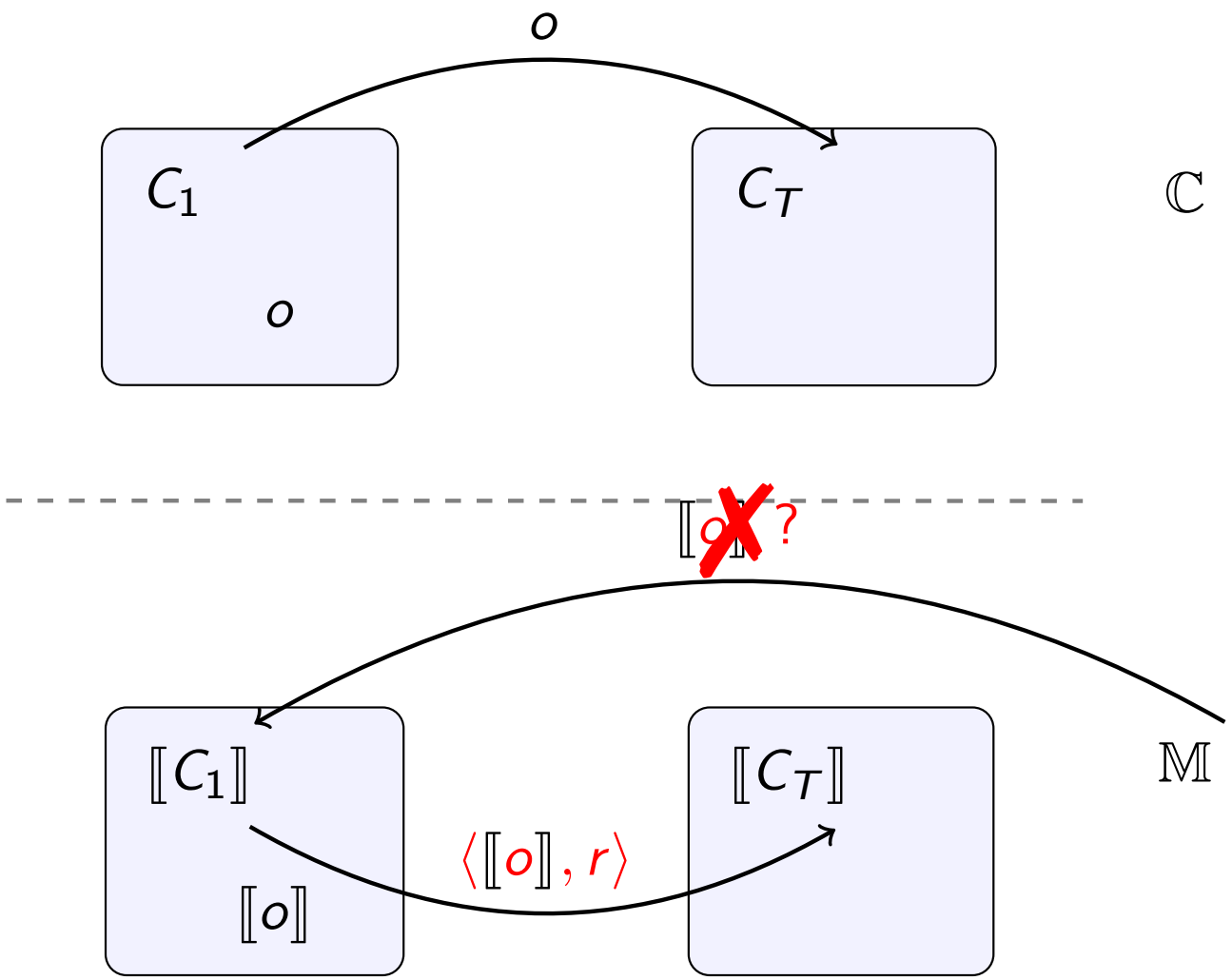
Addressing Object Guessing



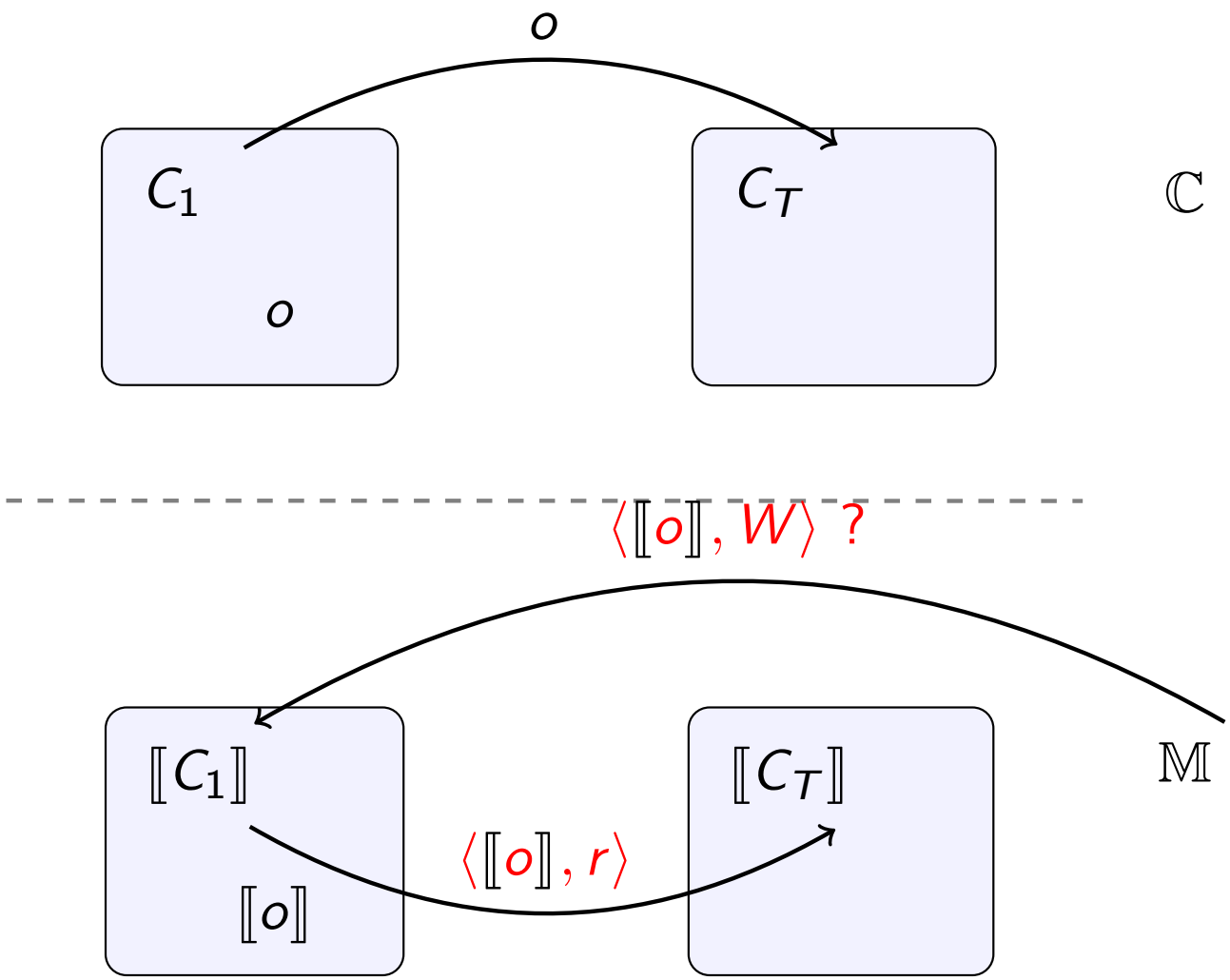
Addressing Object Guessing



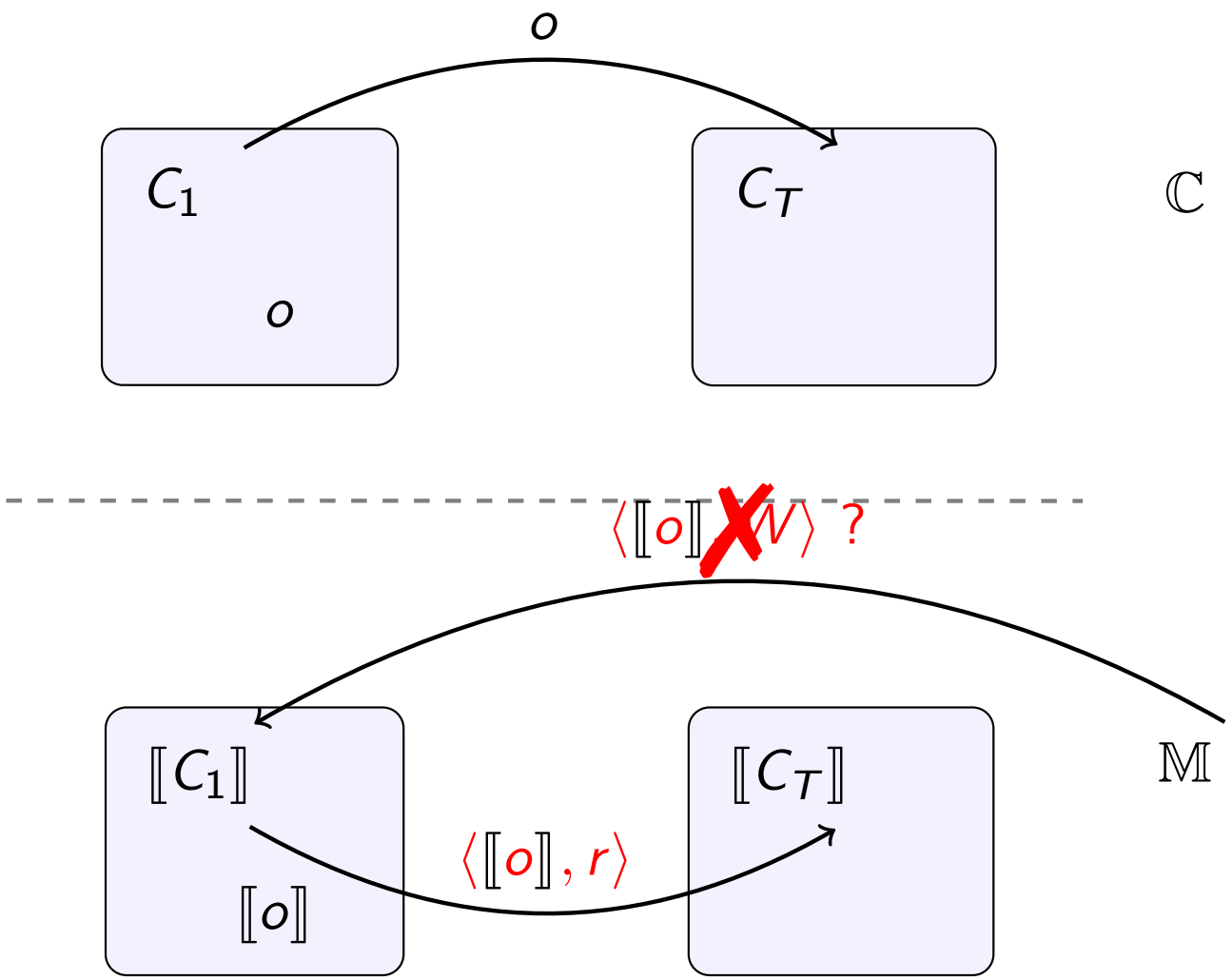
Addressing Object Guessing



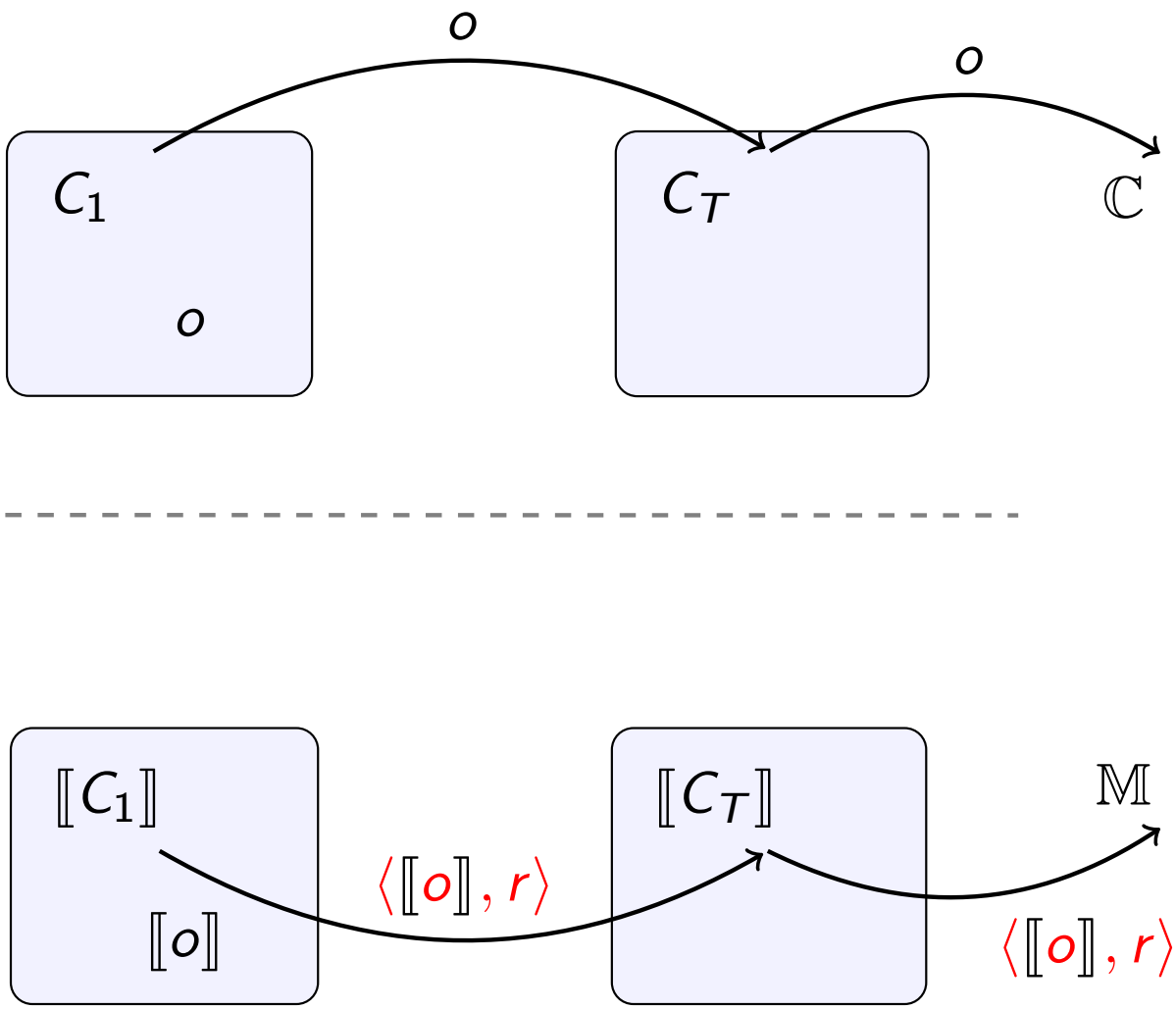
Addressing Object Guessing



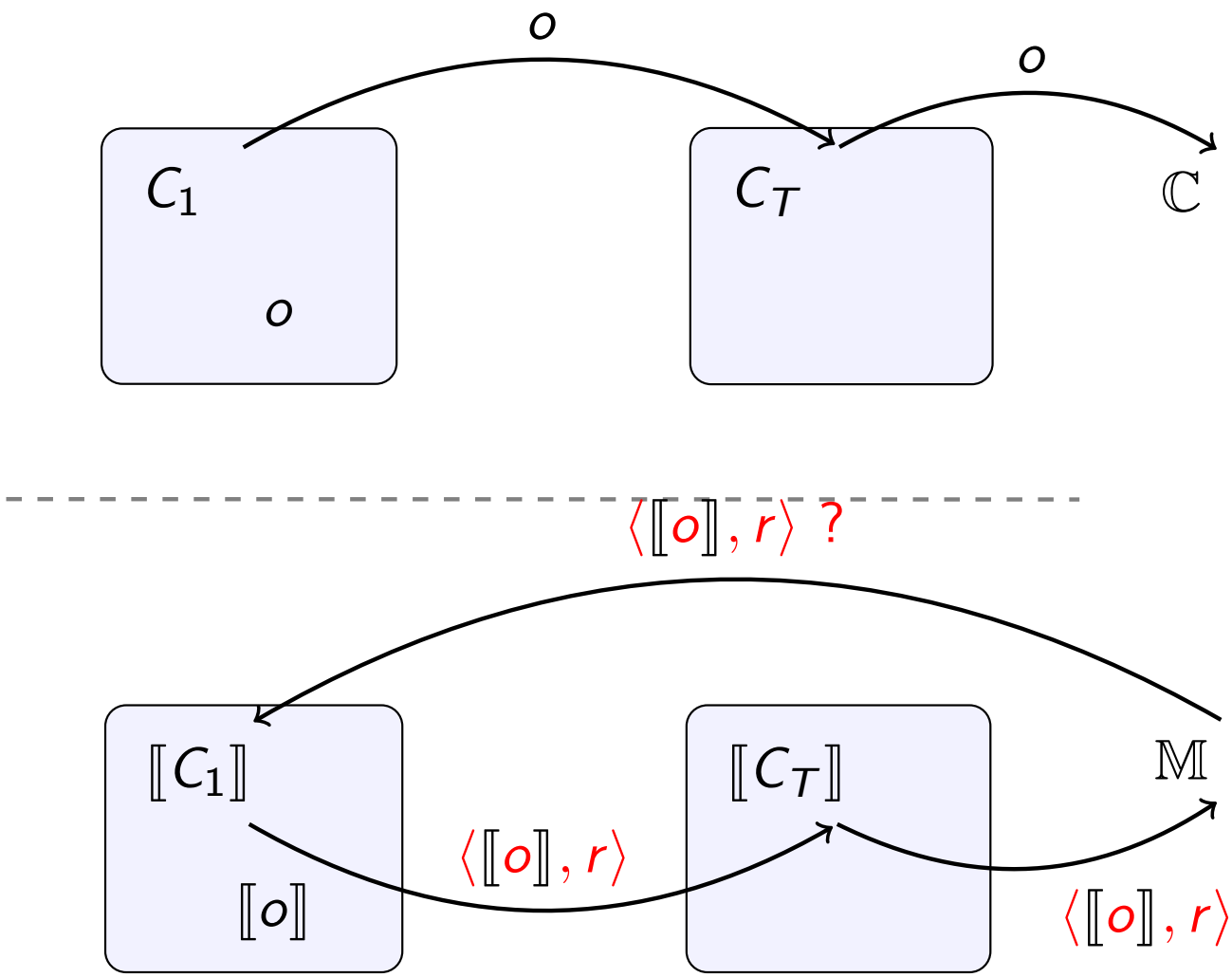
Addressing Object Guessing



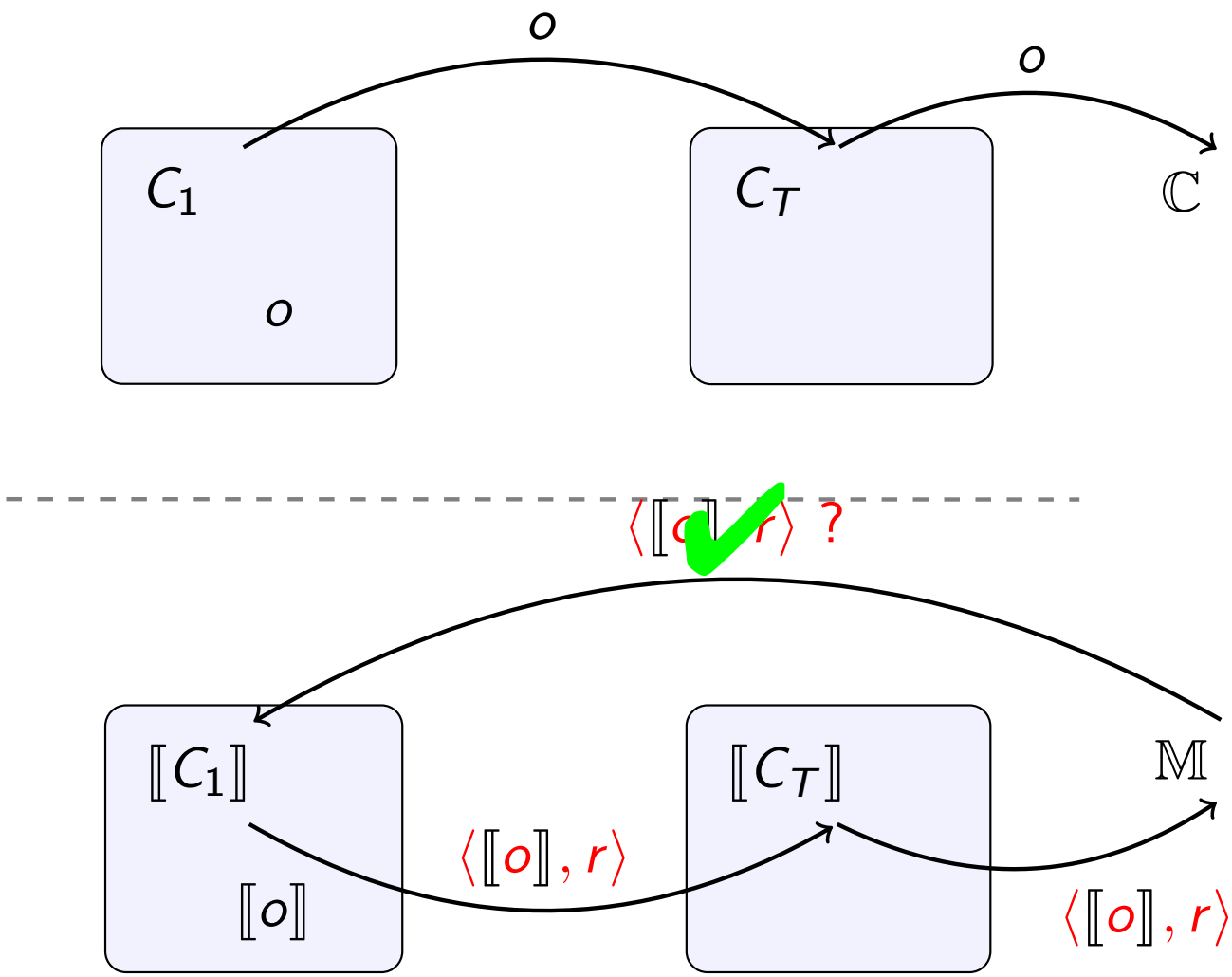
Addressing Object Guessing



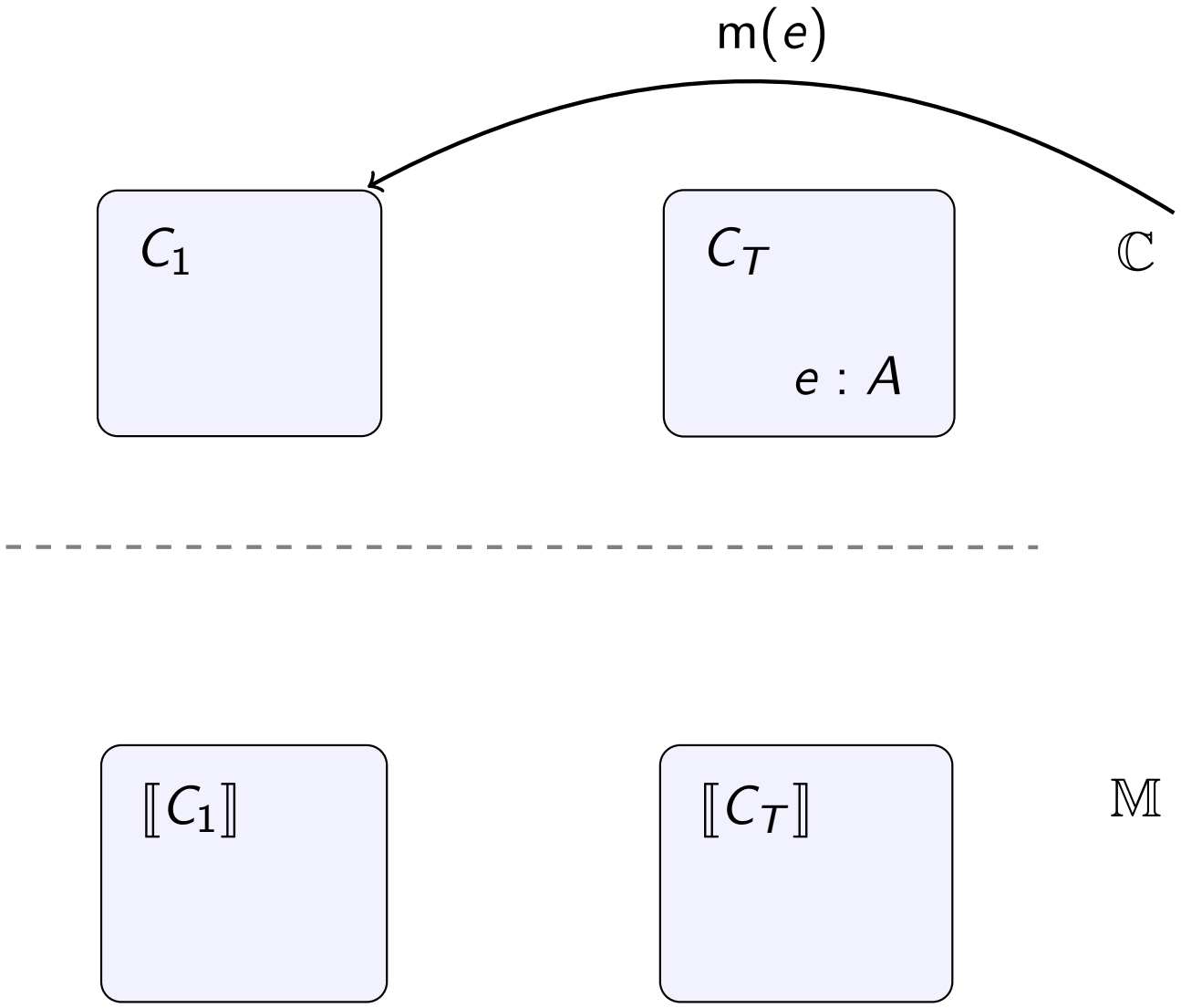
Addressing Object Guessing



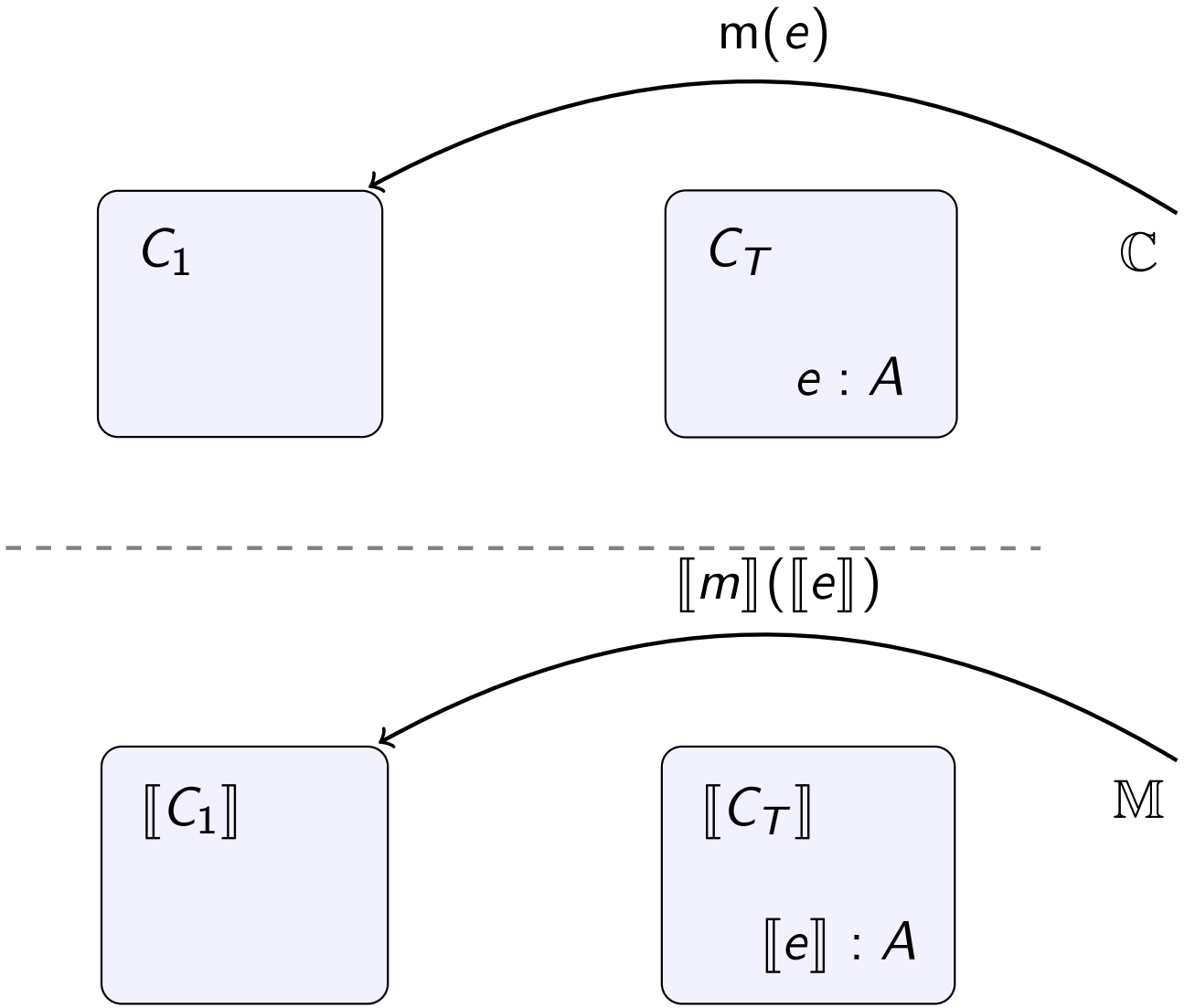
Addressing Object Guessing



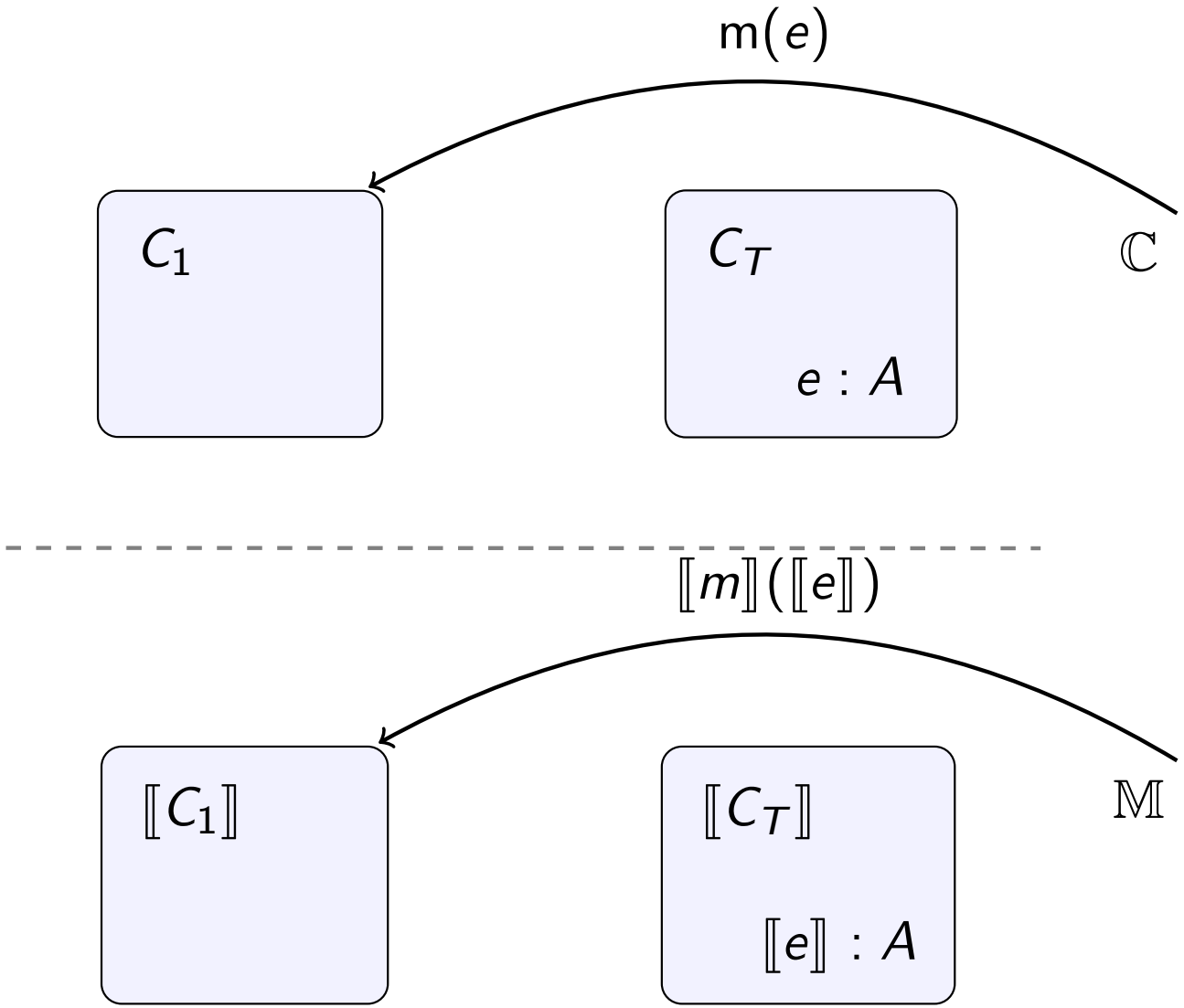
Addressing Object Faking



Addressing Object Faking

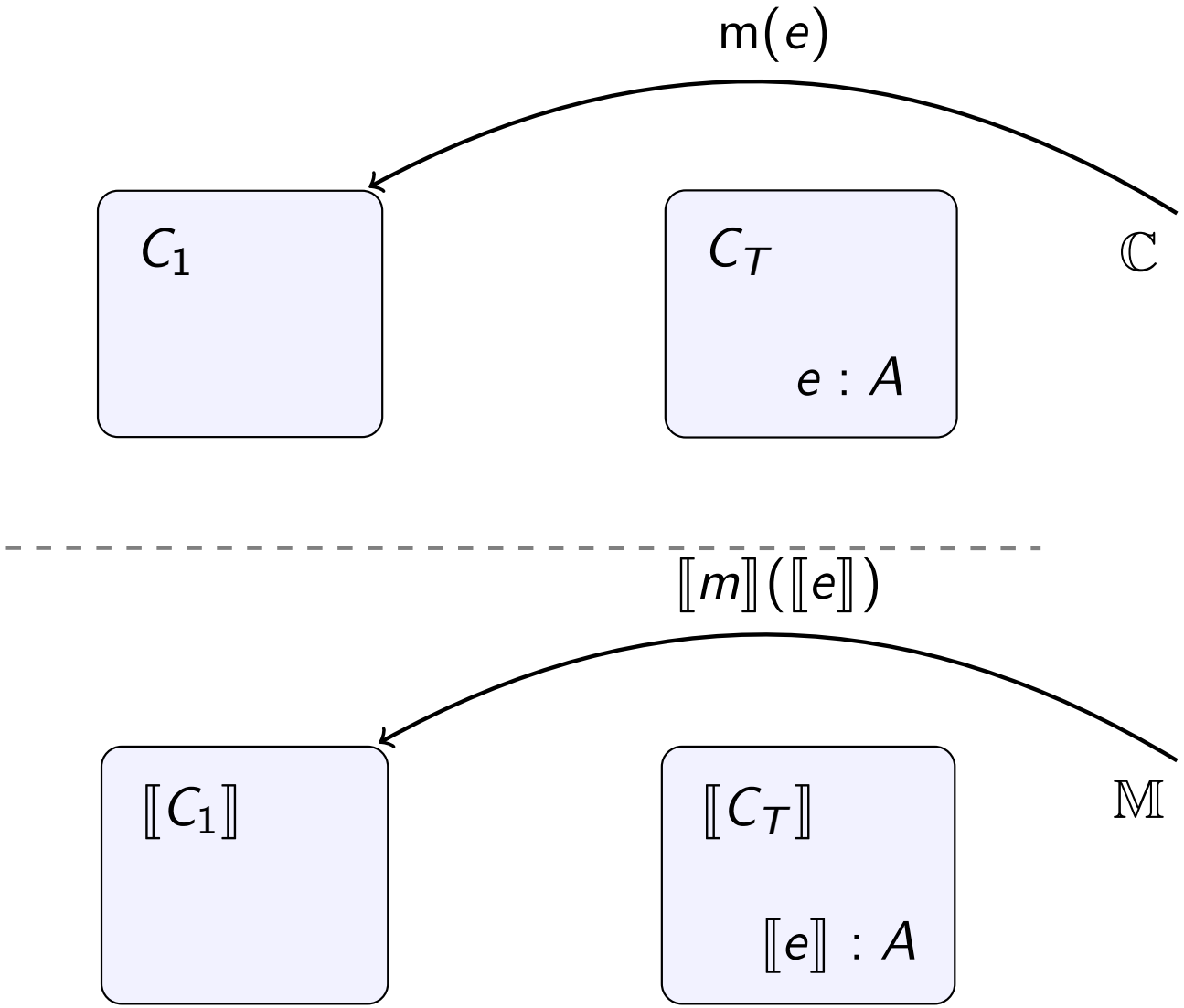


Addressing Object Faking



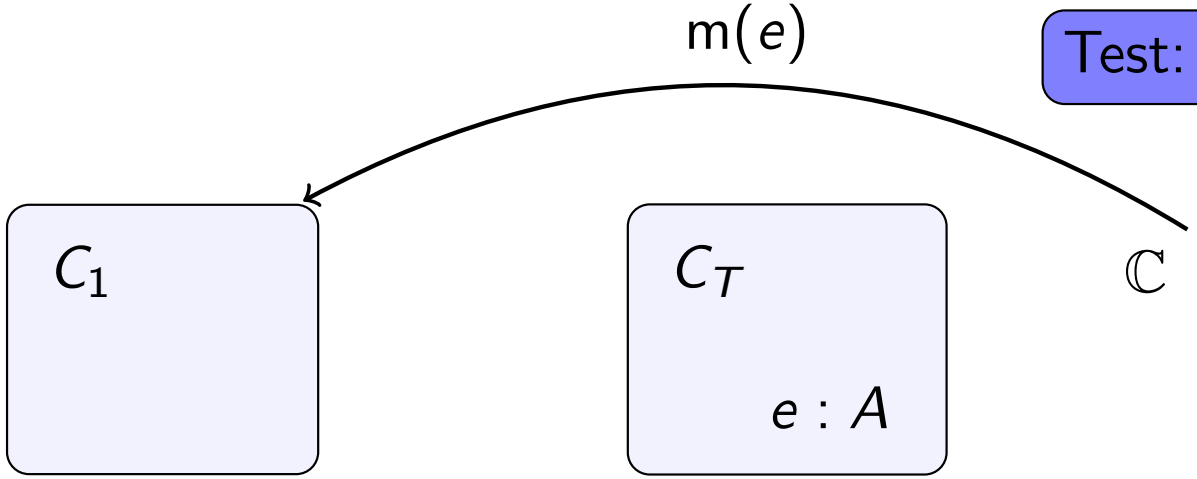
$[\![e]\!] = [\![\texttt{null}]\!]$
or obj : A

Addressing Object Faking



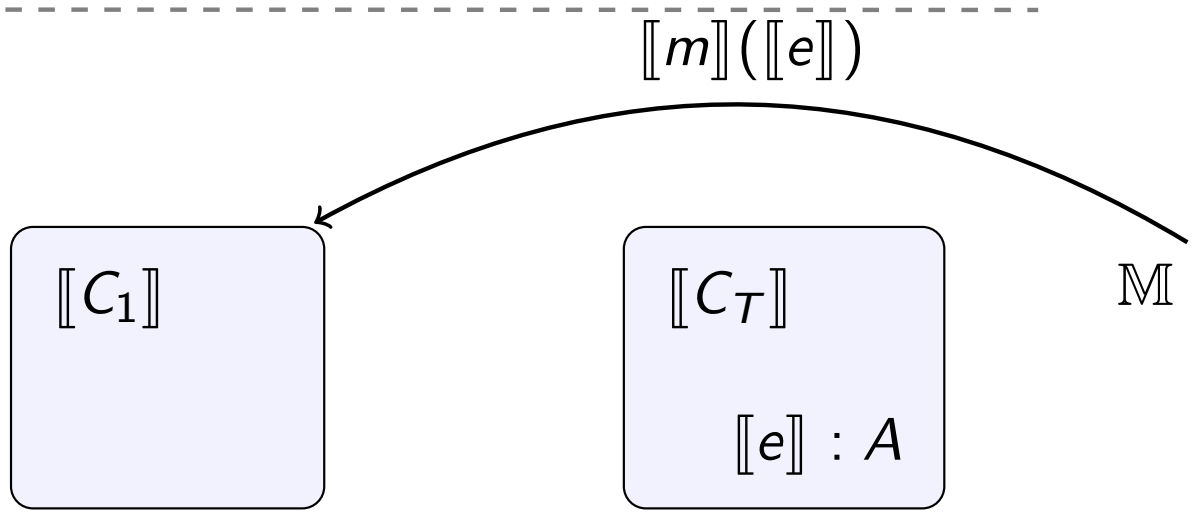
$[\![e]\!] = [\![\texttt{null}]\!]$
or obj : A
or obj : $B \not<: A$
or garbage

Addressing Object Faking

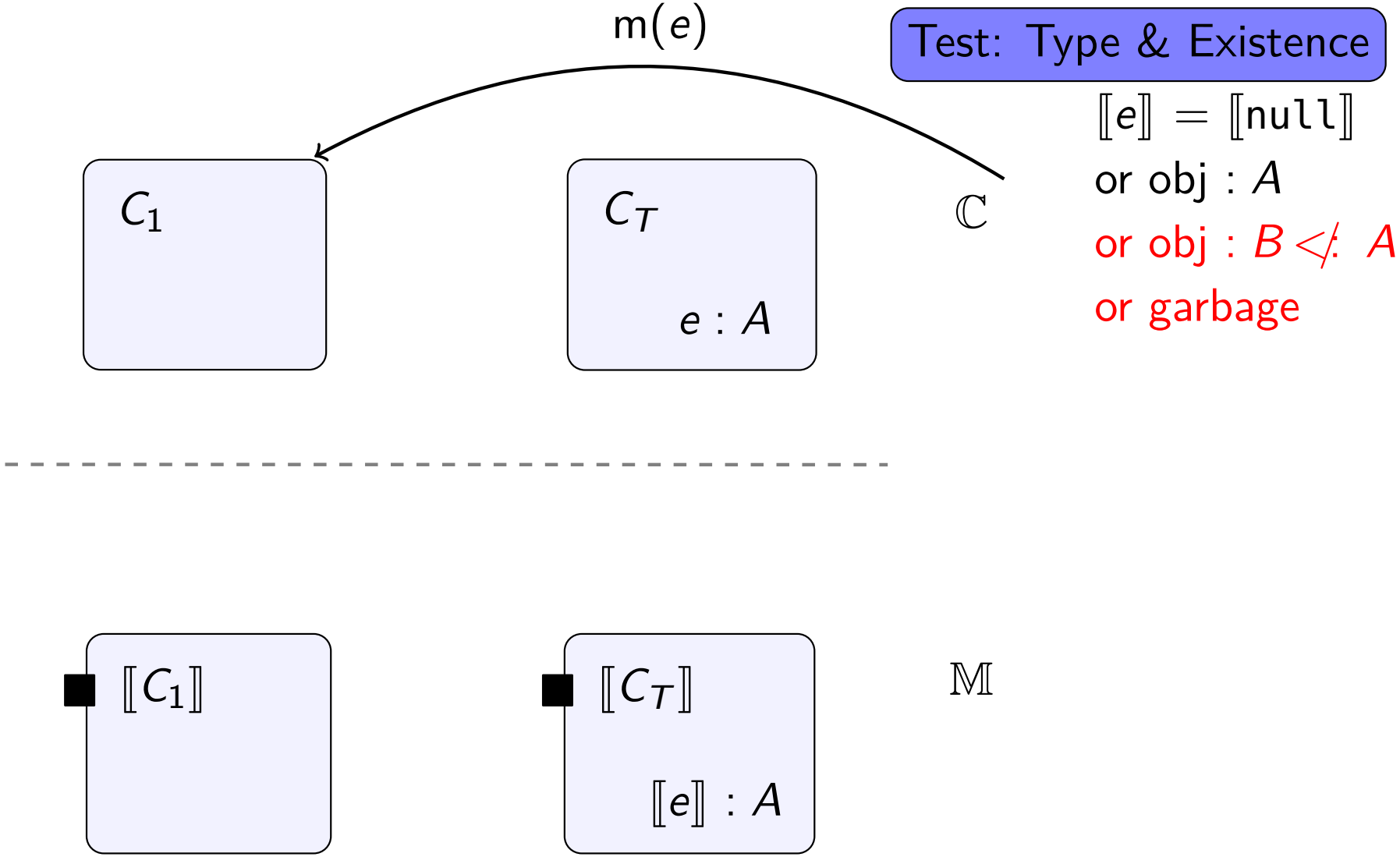


Test: Type & Existence

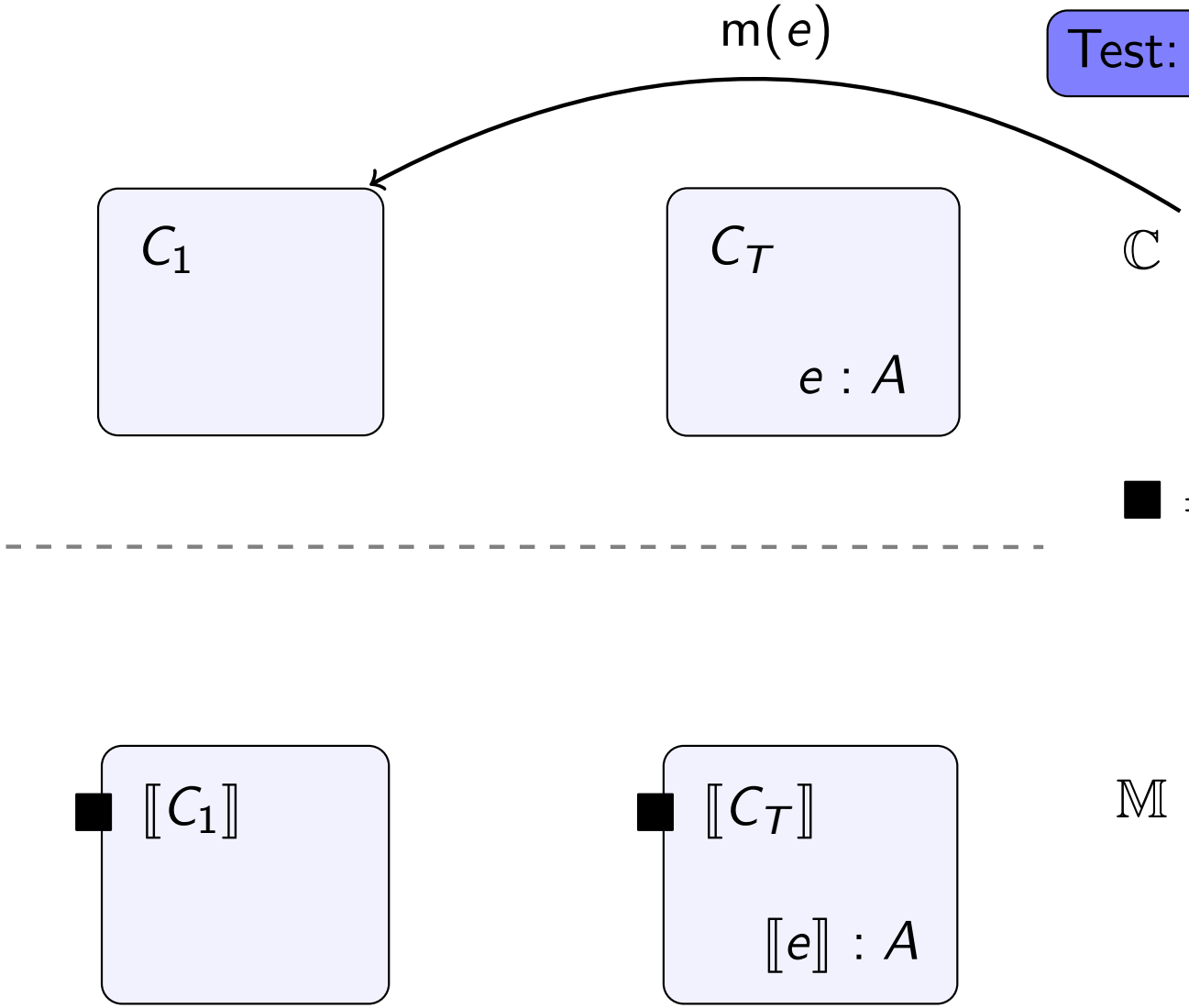
$[\![e]\!] = [\![\texttt{null}]\!]$
or obj : A
or obj : $B \not<: A$
or garbage



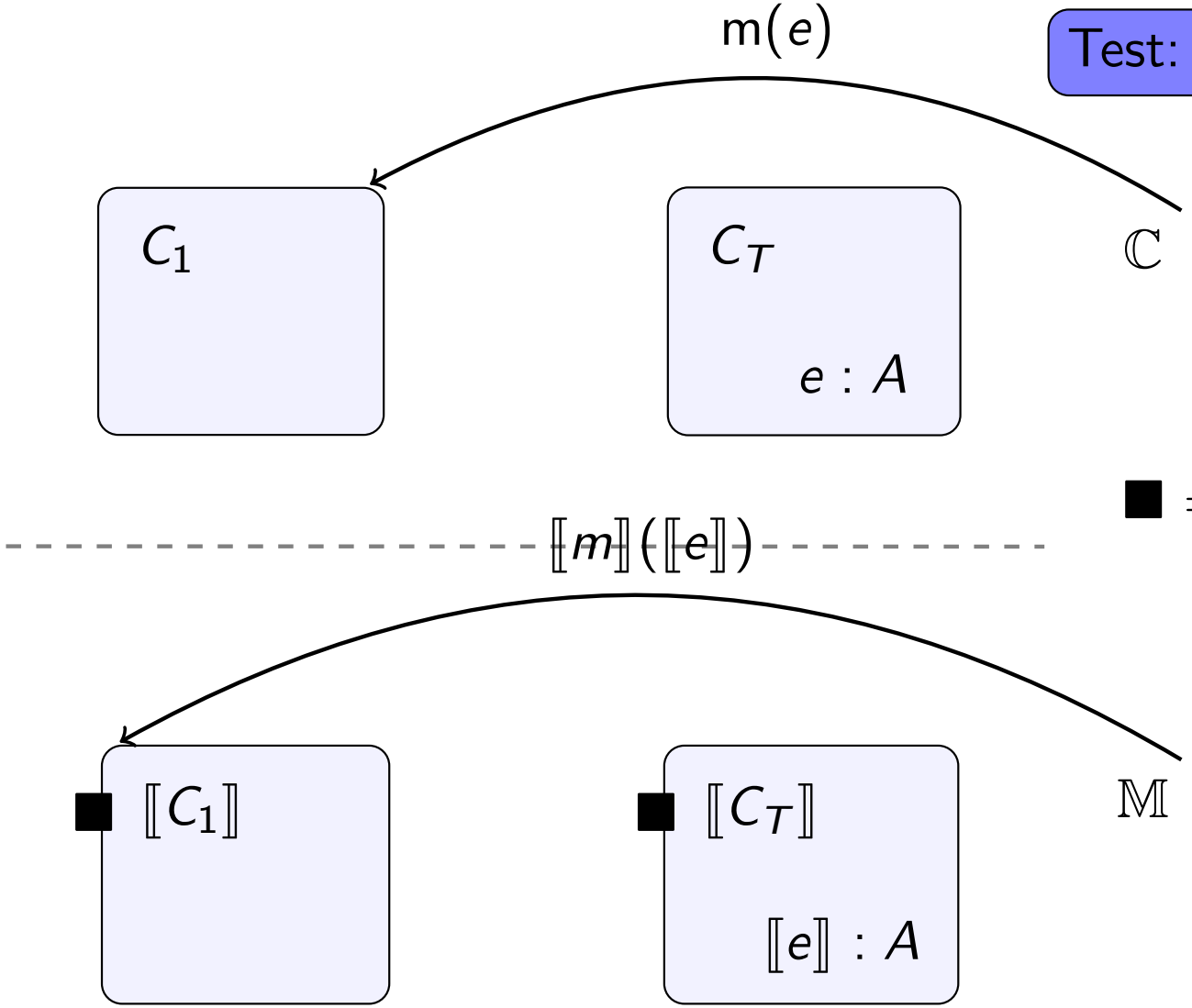
Addressing Object Faking



Addressing Object Faking



Addressing Object Faking

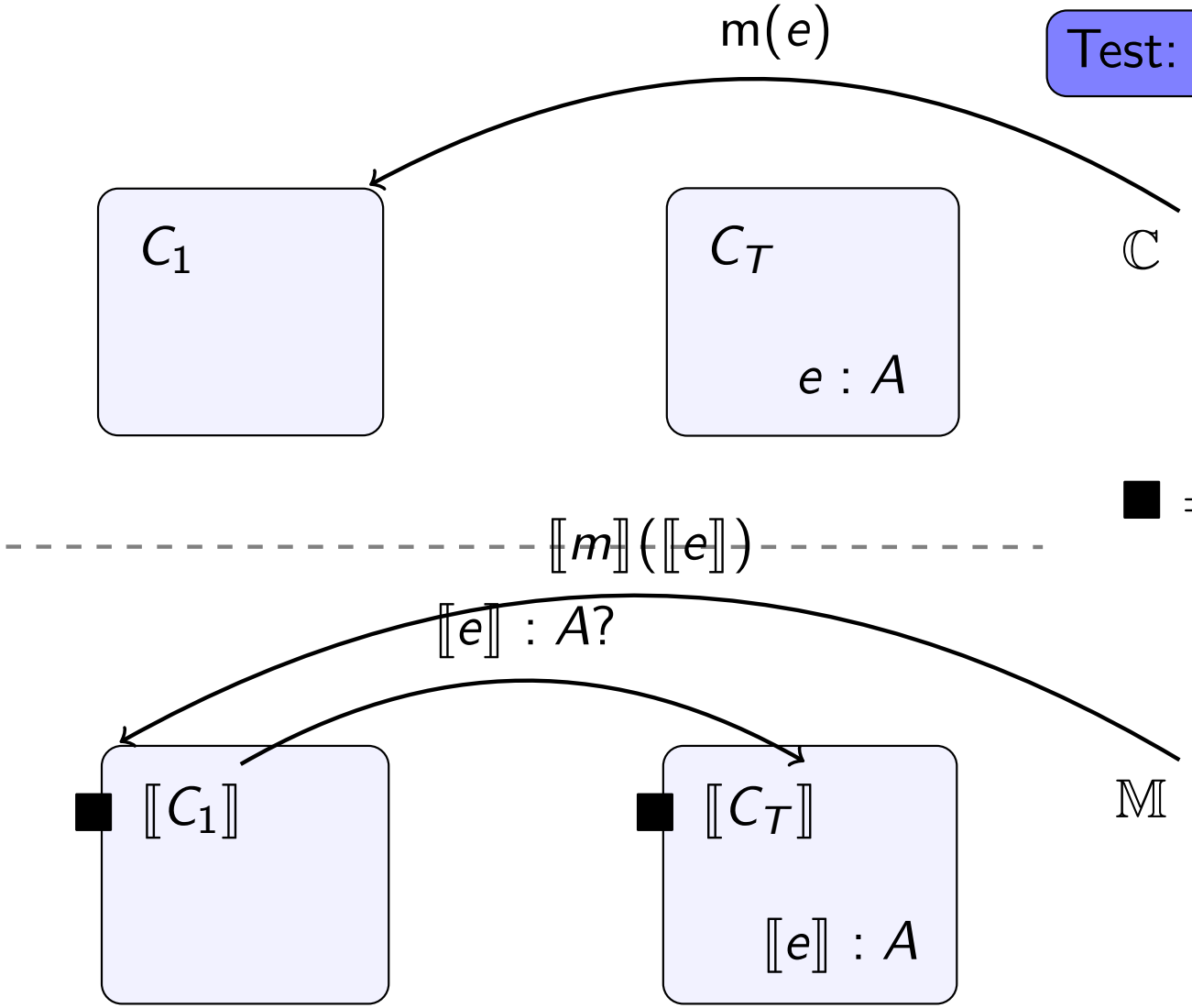


Test: Type & Existence

$[\![e]\!] = [\![\texttt{null}]\!]$
or obj : A
or obj : $B \not<: A$
or garbage

■ = $\texttt{instanceof}(o, A)$
$\hookrightarrow \texttt{0}$ if $o : A \in C_i$
$\hookrightarrow \texttt{1}$ if $o : B \in C_i$
$\hookrightarrow \texttt{1}$ if $o \notin C_i$

Addressing Object Faking

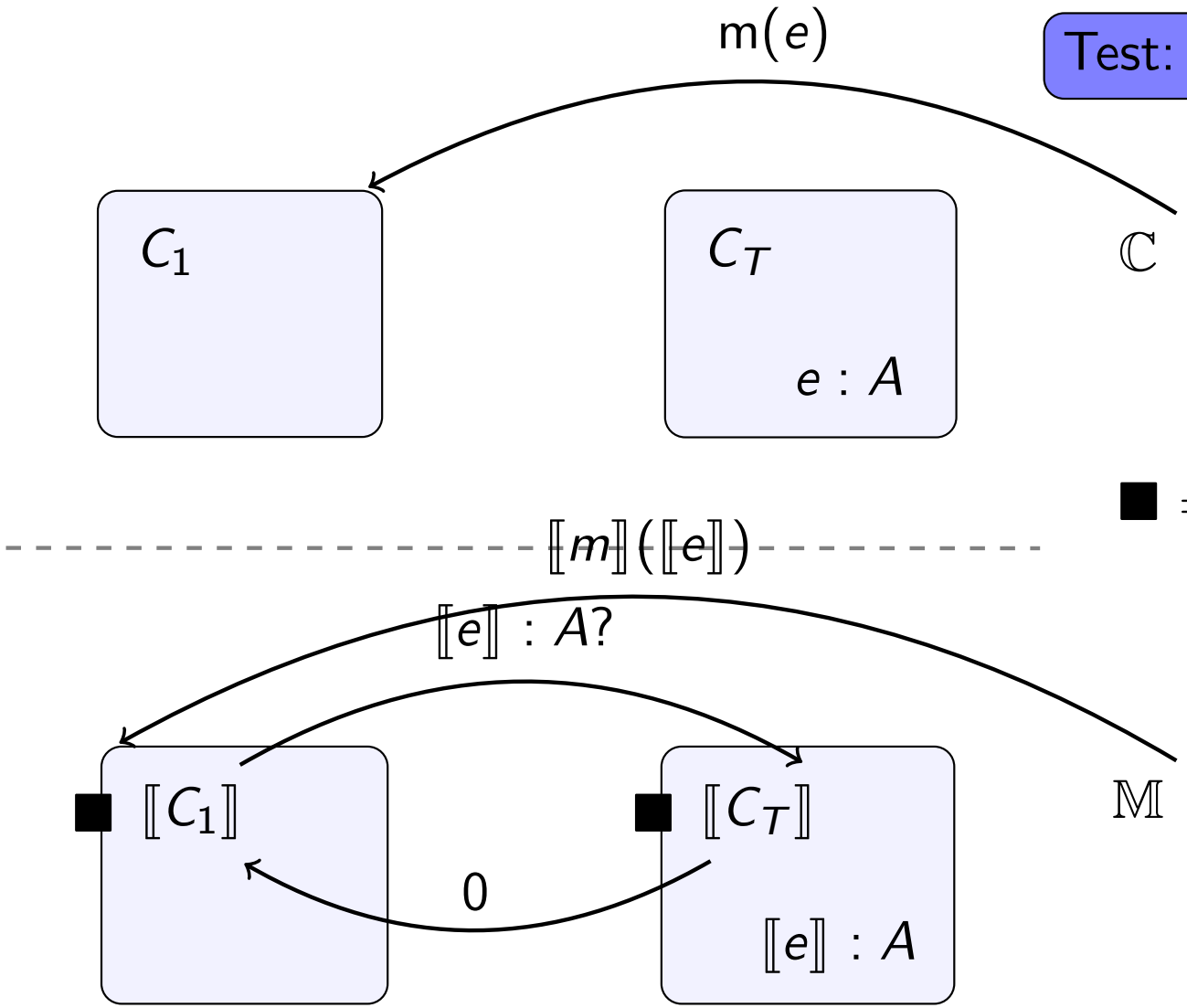


Test: Type & Existence

$[\![e]\!] = [\![\texttt{null}]\!]$
or obj : A
or obj : $B \not<: A$
or garbage

■ = $\texttt{instanceof}(o, A)$
$\hookrightarrow \texttt{0}$ if $o : A \in C_i$
$\hookrightarrow \texttt{1}$ if $o : B \in C_i$
$\hookrightarrow \texttt{1}$ if $o \notin C_i$

Addressing Object Faking

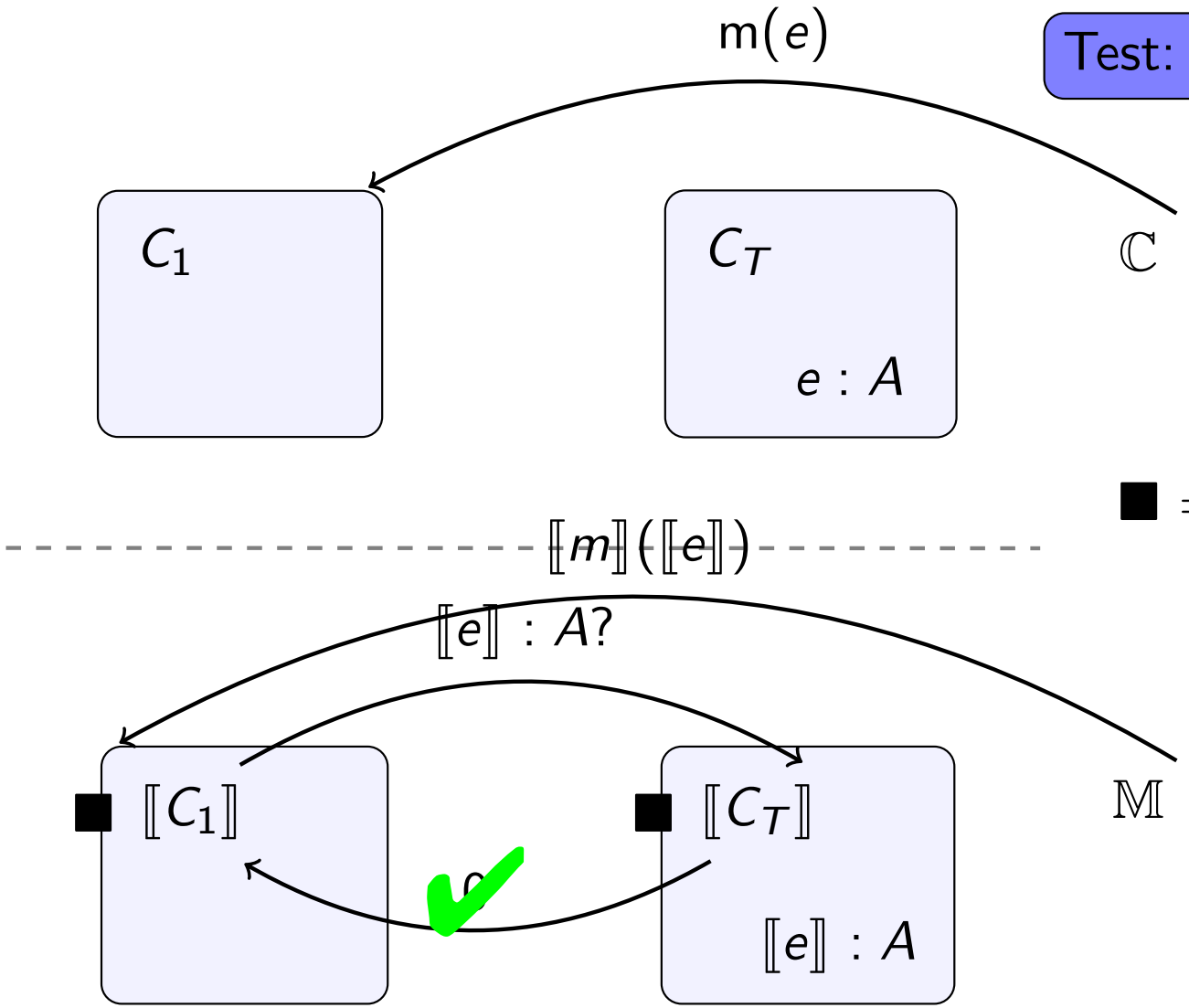


Test: Type & Existence

$[\![e]\!] = [\![\texttt{null}]\!]$
or obj : A
or obj : $B \not<: A$
or garbage

■ = $\texttt{instanceof}(o, A)$
$\hookrightarrow \texttt{0}$ if $o : A \in C_i$
$\hookrightarrow \texttt{1}$ if $o : B \in C_i$
$\hookrightarrow \texttt{1}$ if $o \notin C_i$

Addressing Object Faking

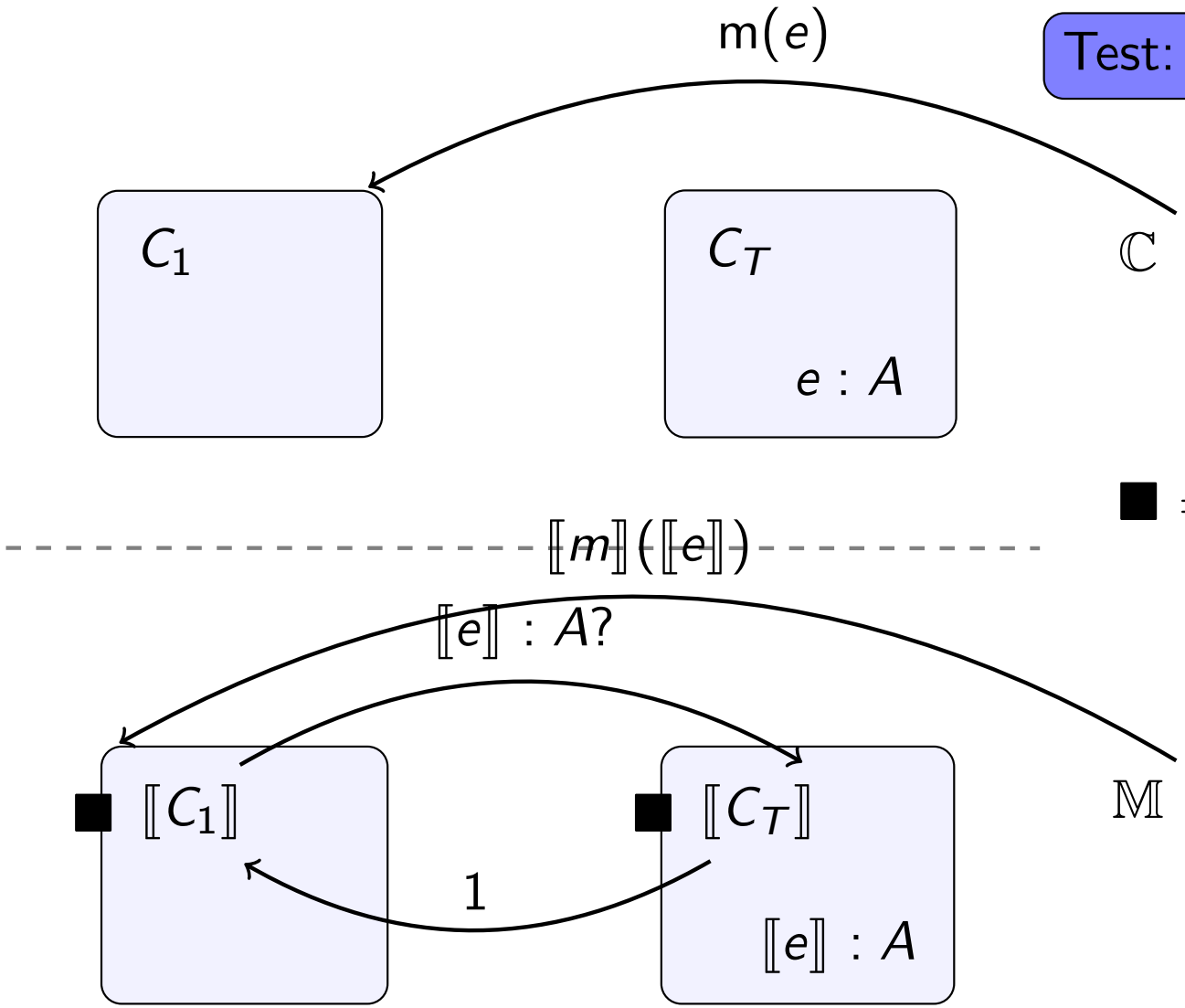


Test: Type & Existence

$[\![e]\!] = [\![\texttt{null}]\!]$
or obj : A
or obj : $B \not<: A$
or garbage

■ = $\texttt{instanceof}(o, A)$
$\hookrightarrow \texttt{0}$ if $o : A \in C_i$
$\hookrightarrow \texttt{1}$ if $o : B \in C_i$
$\hookrightarrow \texttt{1}$ if $o \notin C_i$

Addressing Object Faking

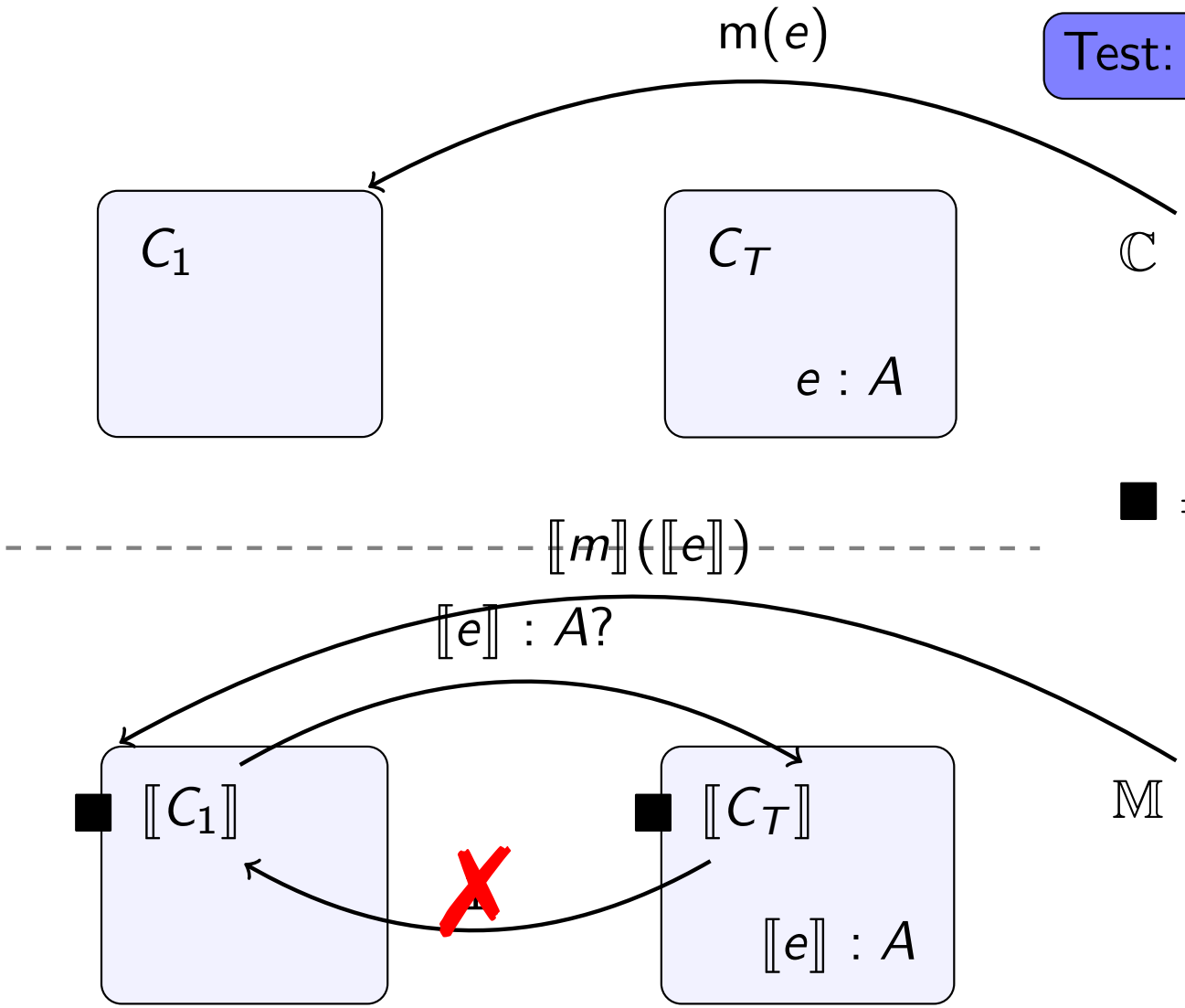


Test: Type & Existence

$[\![e]\!] = [\![\texttt{null}]\!]$
or obj : A
or obj : $B \not<: A$
or garbage

■ = $\texttt{instanceof}(o, A)$
$\hookrightarrow \texttt{0}$ if $o : A \in C_i$
$\hookrightarrow \texttt{1}$ if $o : B \in C_i$
$\hookrightarrow \texttt{1}$ if $o \notin C_i$

Addressing Object Faking

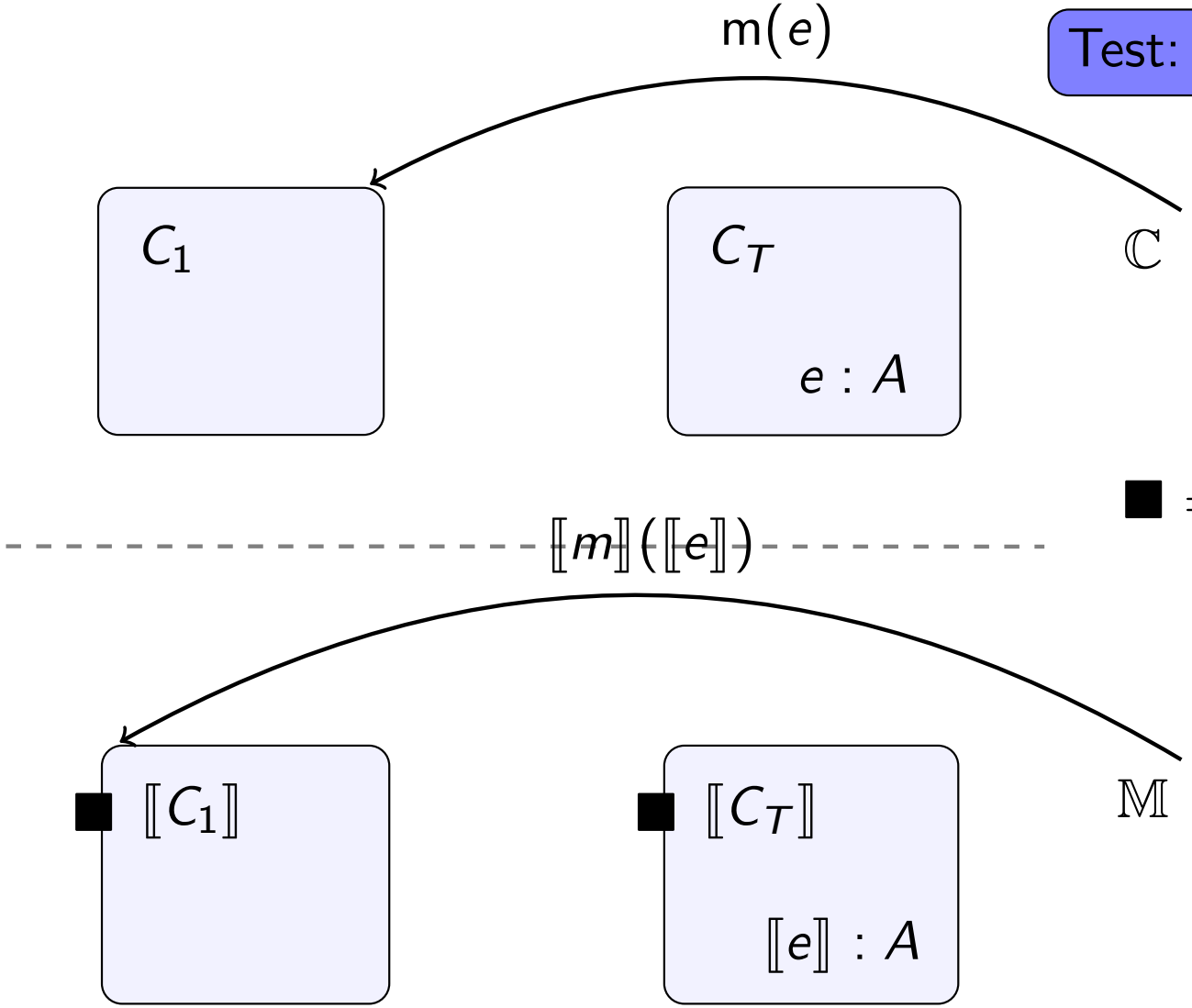


Test: Type & Existence

$[\![e]\!] = [\![\texttt{null}]\!]$
or obj : A
or obj : $B \not<: A$
or garbage

■ = $\texttt{instanceof}(o, A)$
$\hookrightarrow \texttt{0}$ if $o : A \in C_i$
$\hookrightarrow \texttt{1}$ if $o : B \in C_i$
$\hookrightarrow \texttt{1}$ if $o \notin C_i$

Addressing Object Faking



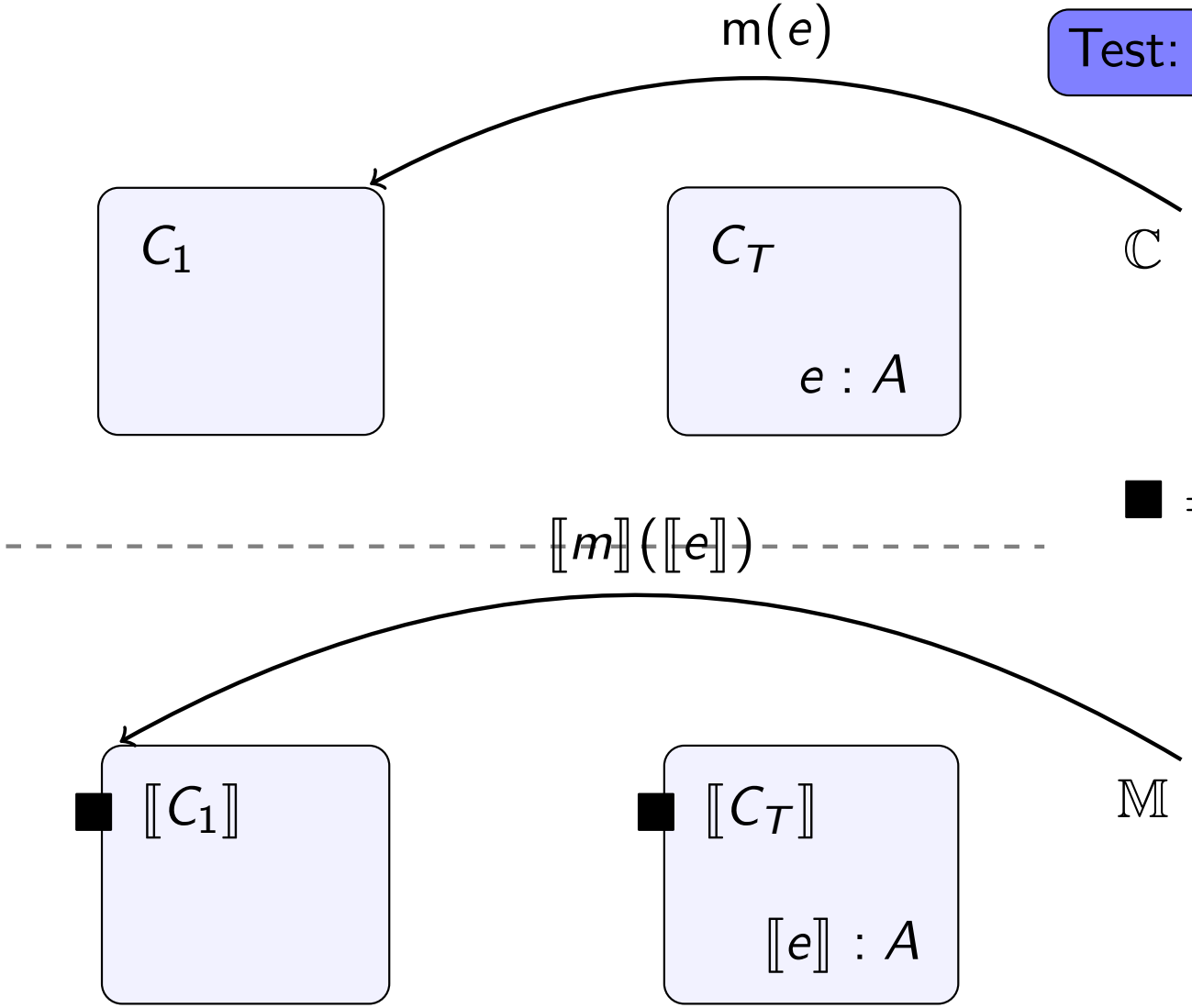
Test: Type & Existence

$[\![e]\!] = [\![\texttt{null}]\!]$
or obj : A
or obj : $B \not<: A$
or garbage

■ = $\texttt{instanceof}(o, A)$
$\hookrightarrow \texttt{0}$ if $o : A \in C_i$
$\hookrightarrow \texttt{1}$ if $o : B \in C_i$
$\hookrightarrow \texttt{1}$ if $o \notin C_i$

■ $\in \mathcal{S}$

Addressing Object Faking



Test: Type & Existence

$[\![e]\!] = [\![\texttt{null}]\!]$
or obj : A
or obj : $B \not<: A$
or garbage

■ = $\texttt{instanceof}(o, A)$
$\hookrightarrow \texttt{0}$ if $o : A \in C_i$
$\hookrightarrow \texttt{1}$ if $o : B \in C_i$
$\hookrightarrow \texttt{1}$ if $o \notin C_i$

■ $\in \mathcal{S}$

■ $\notin C_i$

Questions

Thank you!

Qs ?