Parameterised Linearisability

Andrea Cerone

Joint work with Alexey Gotsman and Hongseok Yang

ICALP - Copenhagen, July 8th, 2014

A Simple Example

- Converting a sequential data structure into a concurrent one

Trivial Solution:

```
LOCK lock;
do_push(z):
    lock.acquire();
    int retval = push(z);
    lock.release();
    return retval;
```

A Simple Example

- Converting a sequential data structure into a concurrent one

Trivial Solution:

```
LOCK lock;
do_push(z):
    lock.acquire();
    int retval = push(z);
    lock.release();
    return retval;
```

- Works for any implementation of push
- but it's inefficient (we can do much better…)

Flat Combiners (Hendler et al, 2010)

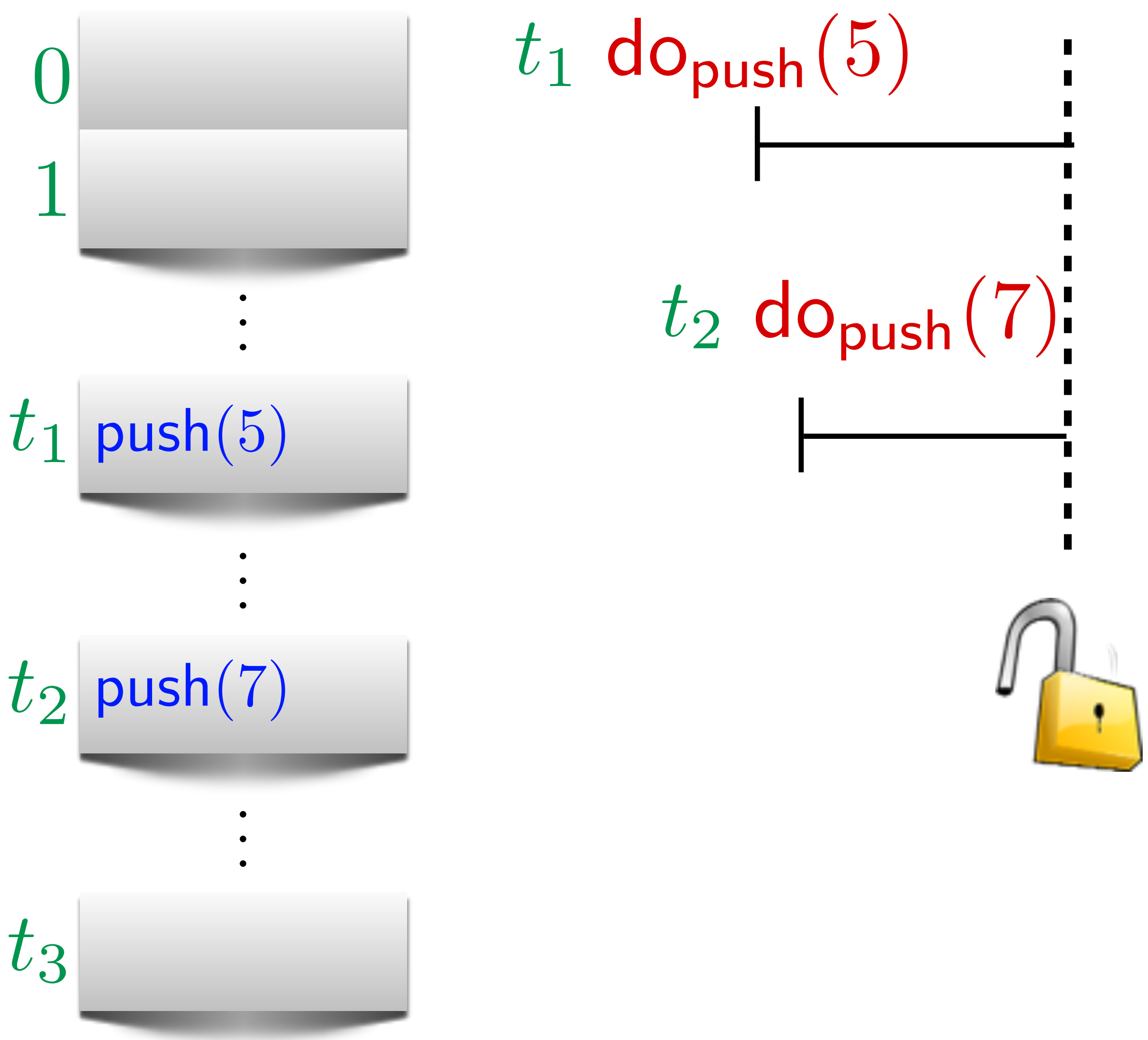
Idea: let a single thread handle all requests

Flat Combiners (Hendler et al, 2010)

Idea: let a single thread handle all requests

Stack provides methods push, pop to clients

FC_{stack} : access to methods in Stack regulated by do_{push}, do_{pop}

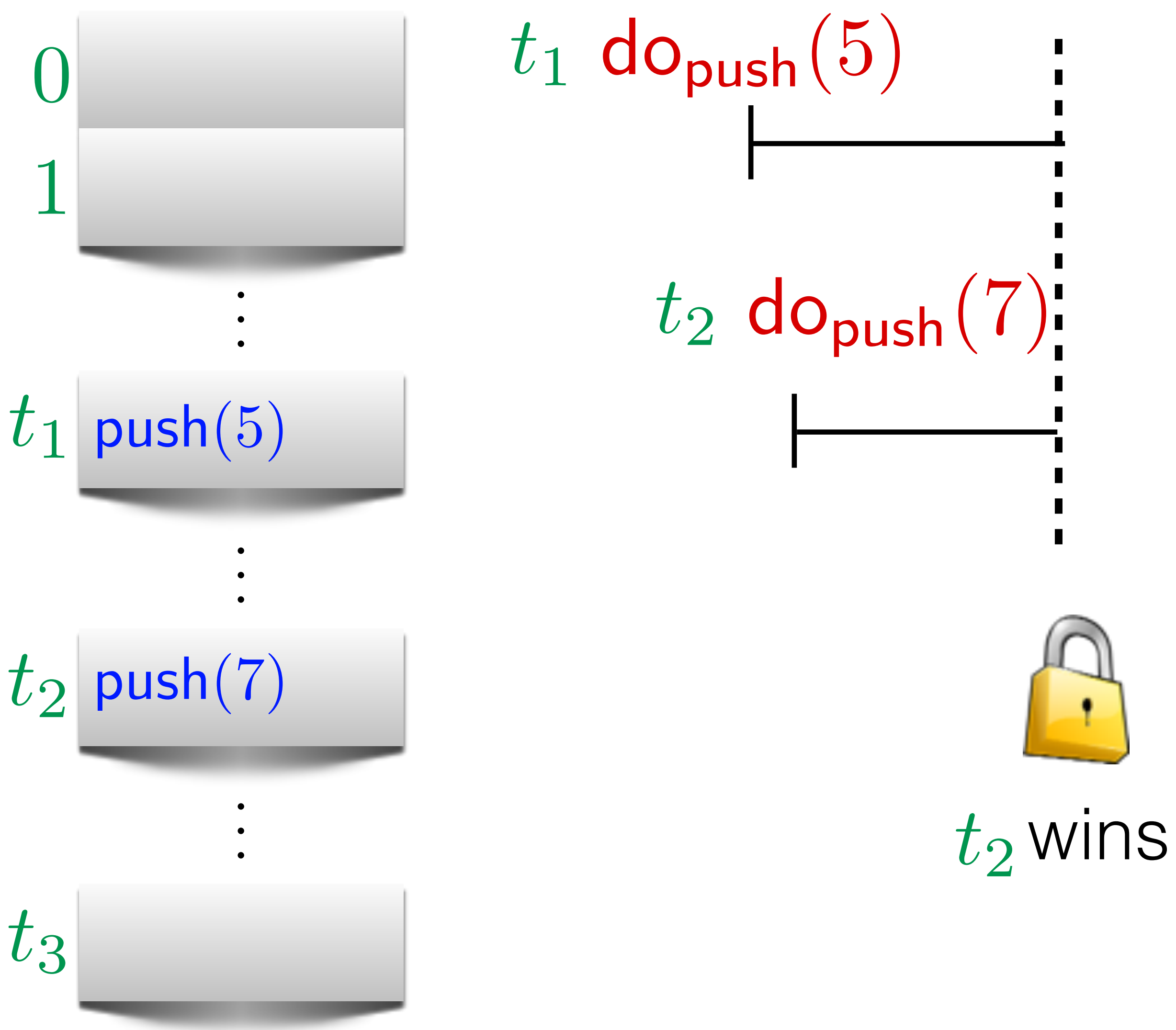


Flat Combiners (Hendler et al, 2010)

Idea: let a single thread handle all requests

Stack provides methods push, pop to clients

$\mathsf{FC}_{\mathsf{stack}}$: access to methods in Stack regulated by $\mathsf{do}_{\mathsf{push}}$, $\mathsf{do}_{\mathsf{pop}}$

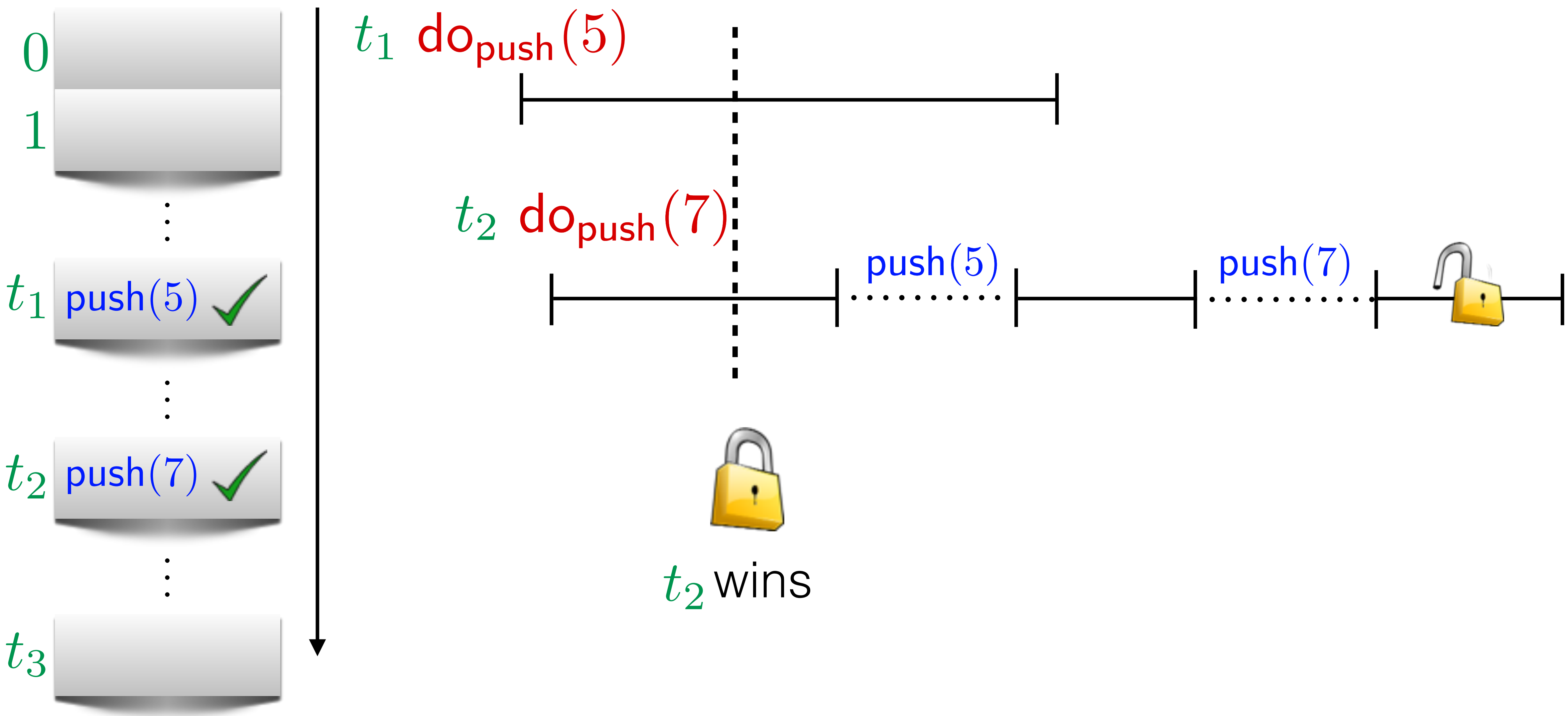


Flat Combiners (Hendler et al, 2010)

Idea: let a single thread handle all requests

Stack provides methods push, pop to clients

$\mathsf{FC}_{\mathsf{stack}}$: access to methods in Stack regulated by $\mathsf{do}_{\mathsf{push}}$, $\mathsf{do}_{\mathsf{pop}}$

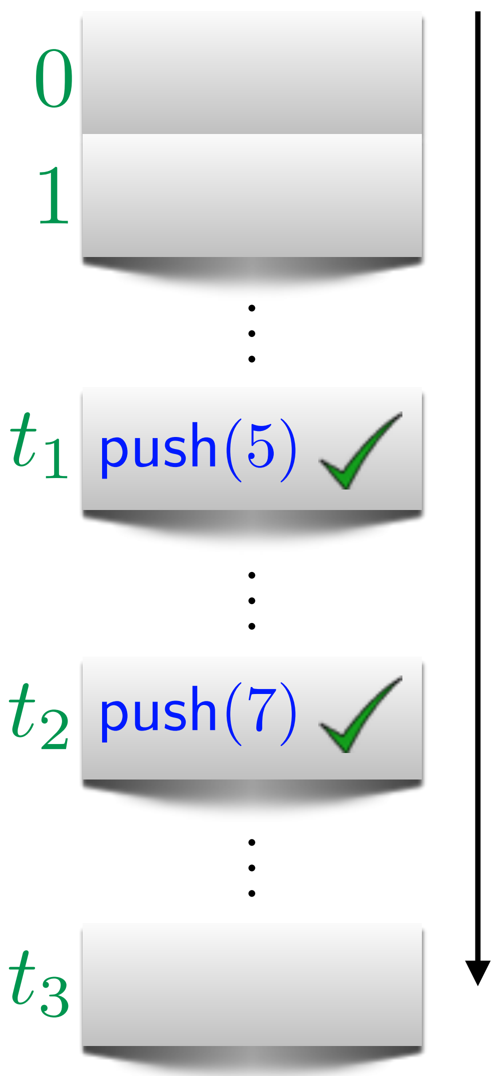


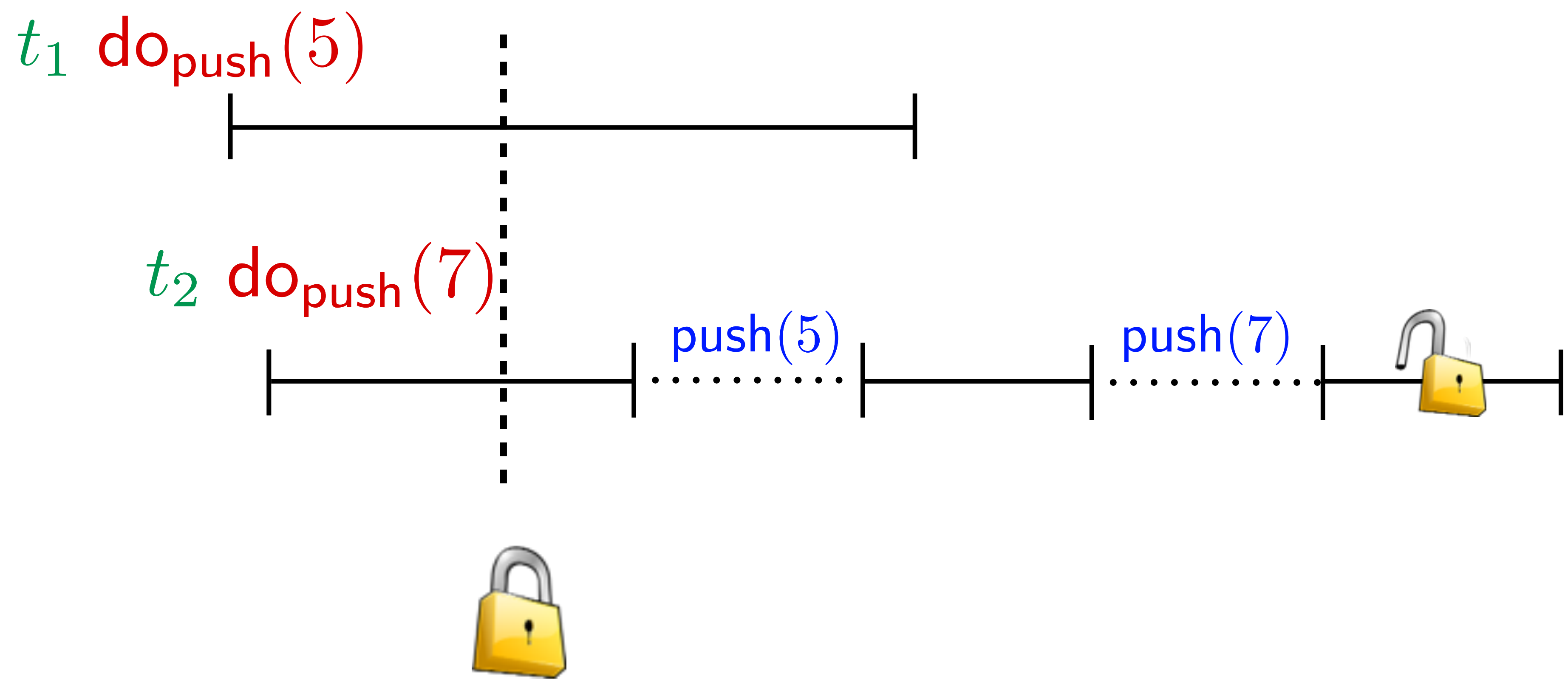
Flat Combiners (Hendler et al, 2010)

Idea: let a single thread handle all requests

Stack provides methods push, pop to clients

$\mathsf{FC}_{\mathsf{stack}}$: access to methods in Stack regulated by $\mathsf{do}_{\mathsf{push}}$, $\mathsf{do}_{\mathsf{pop}}$





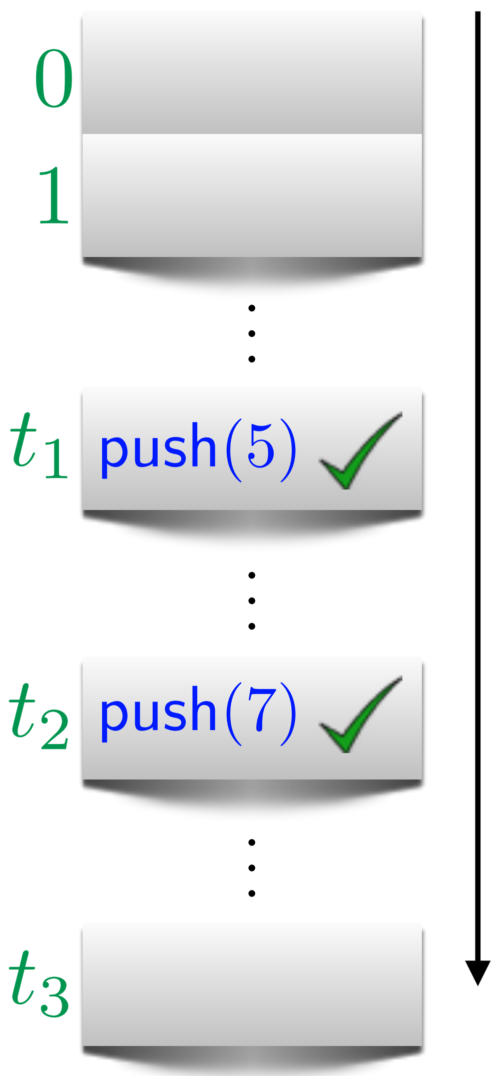
- push, pop not implemented in $\mathsf{FC}_{\mathsf{stack}}$ (abstract methods)

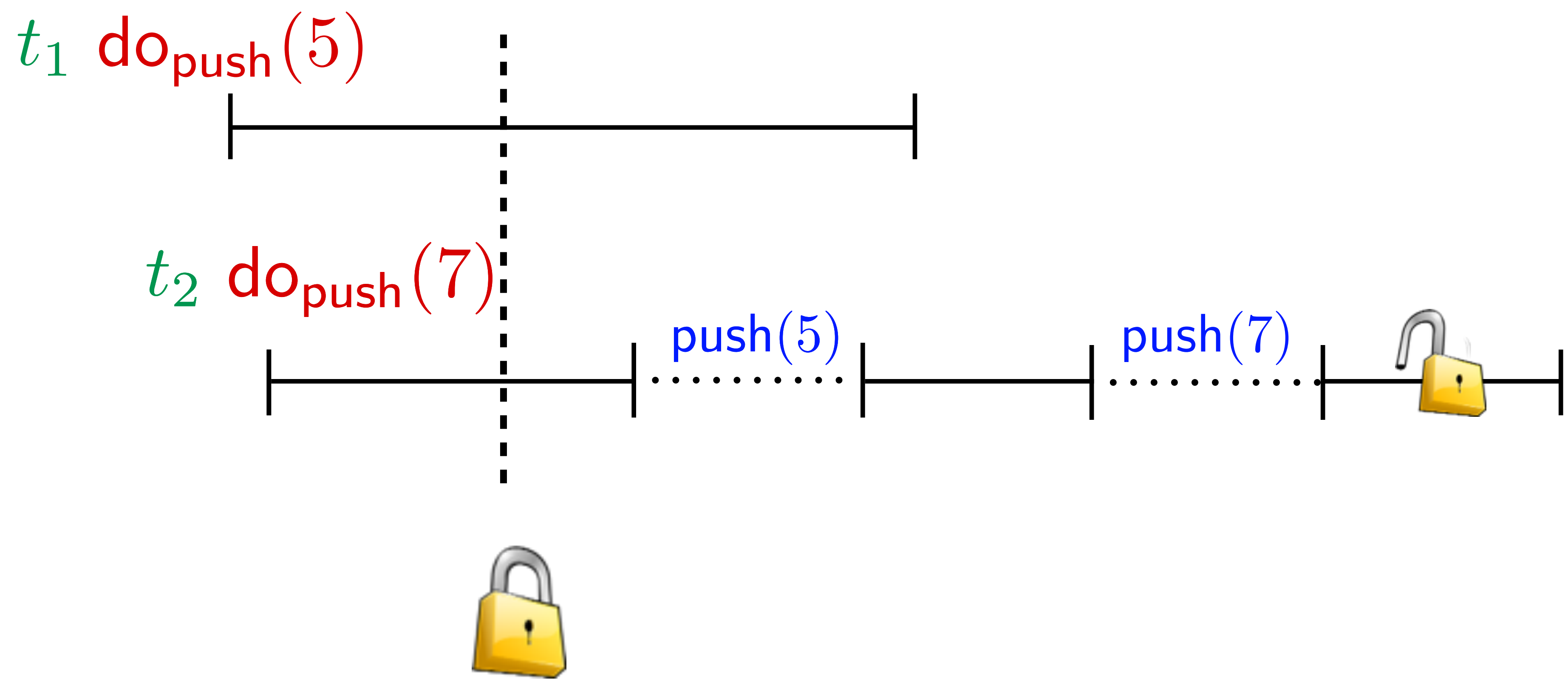
Flat Combiners (Hendler et al, 2010)

Idea: let a single thread handle all requests

Stack provides methods push, pop to clients

$\mathsf{FC}_{\mathsf{stack}}$: access to methods in Stack regulated by $\mathsf{do}_{\mathsf{push}}$, $\mathsf{do}_{\mathsf{pop}}$





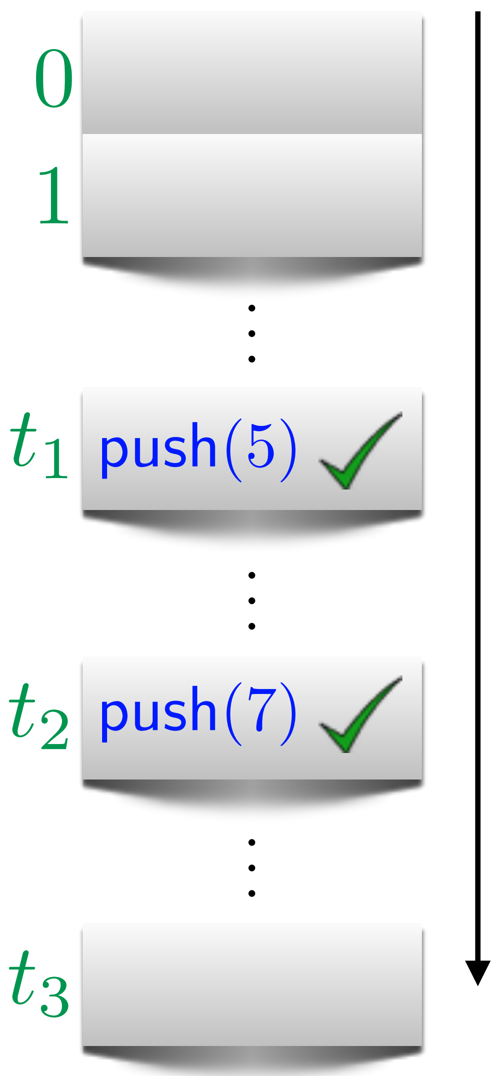
- push, pop not implemented in $\mathsf{FC}_{\mathsf{stack}}$ (abstract methods)
- Guaranteed to have correct behaviour for (**almost**) any implementation of abstract methods

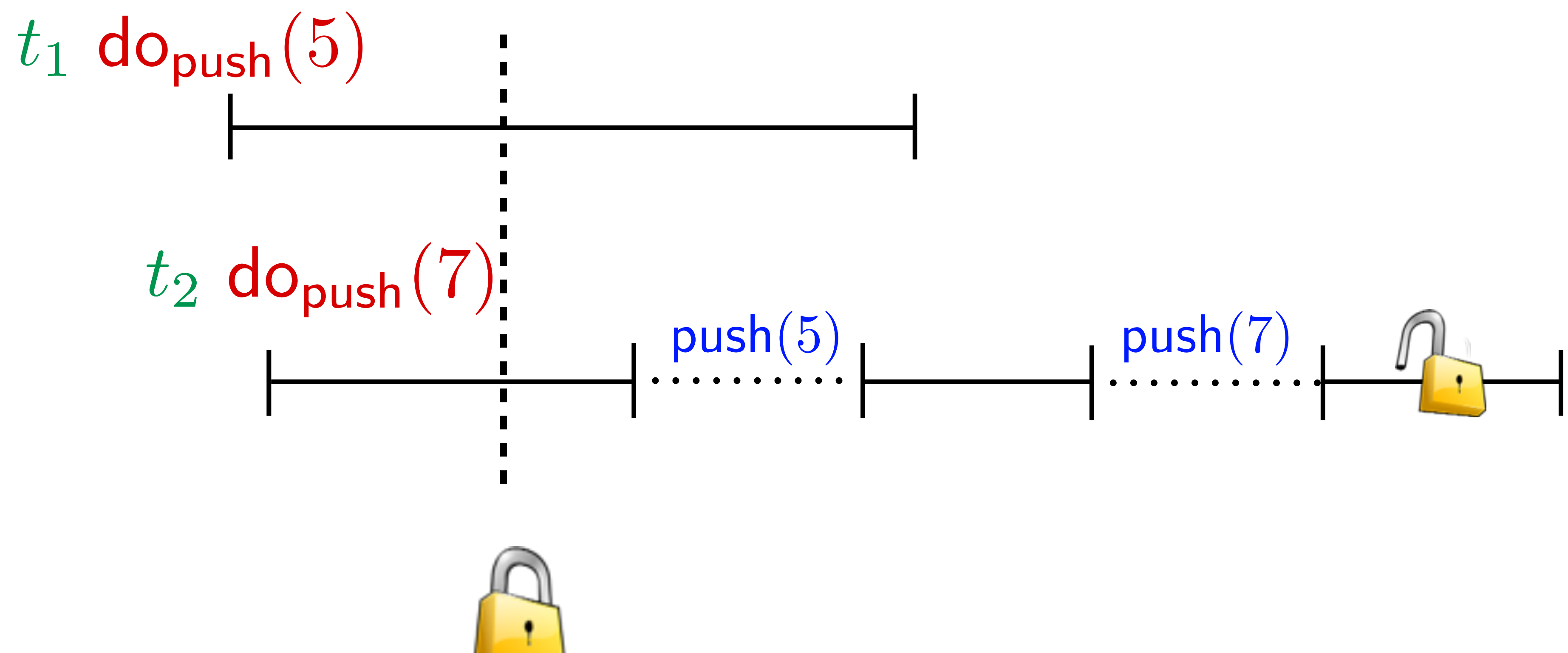
Flat Combiners (Hendler et al, 2010)

Idea: let a single thread handle all requests

Stack provides methods push, pop to clients

$\mathsf{FC}_{\mathsf{stack}}$: access to methods in Stack regulated by $\mathsf{do}_{\mathsf{push}}$, $\mathsf{do}_{\mathsf{pop}}$





It appears to behave as the naive solution

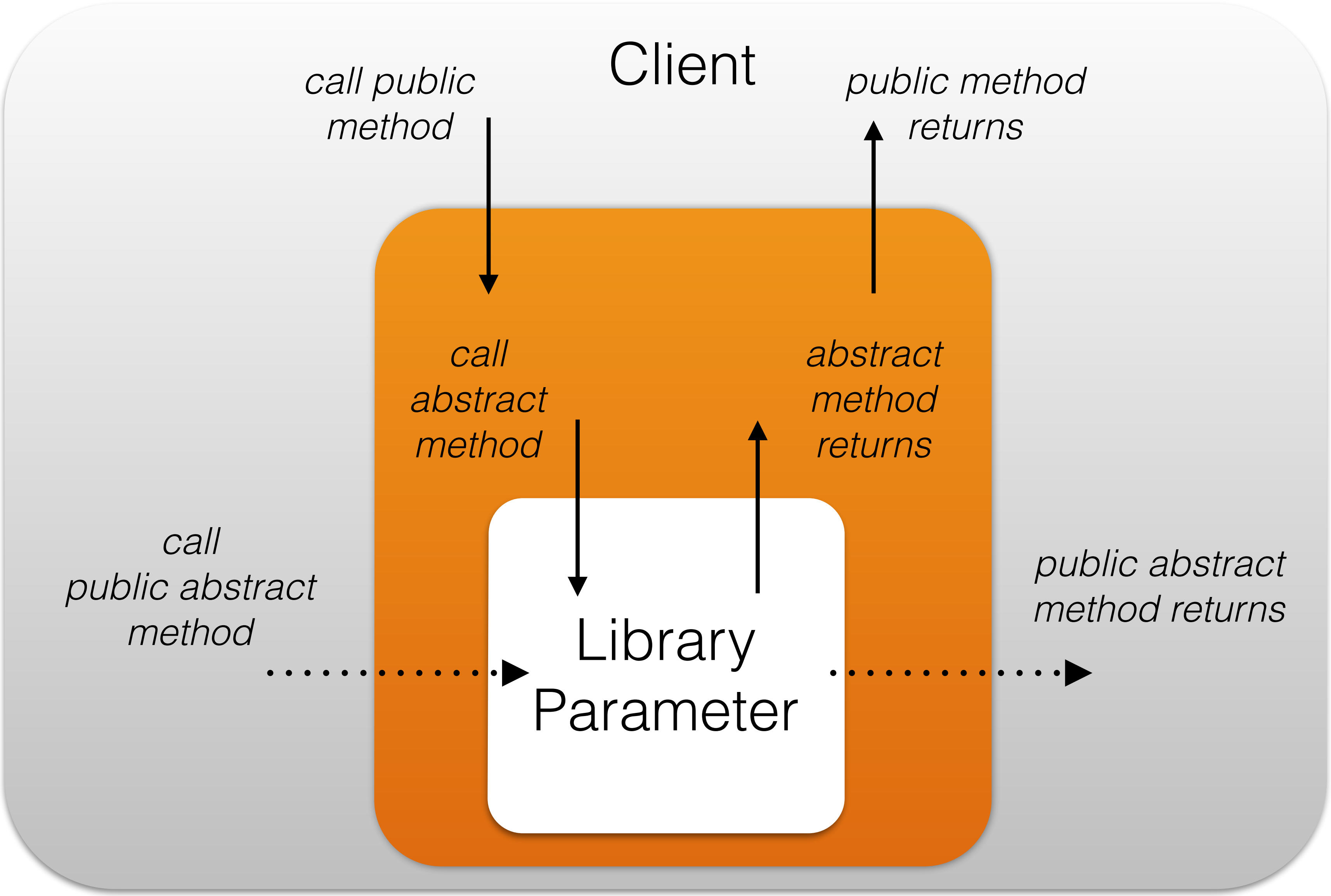
- push, pop nc … ct methods)
- Guaranteed to have correct behaviour for (**almost**) any implementation of abstract methods

Second Order Libraries

Public methods define the operations made available to clients

Abstract methods (without implementation) can be declared

$$L : M' \rightarrow M$$

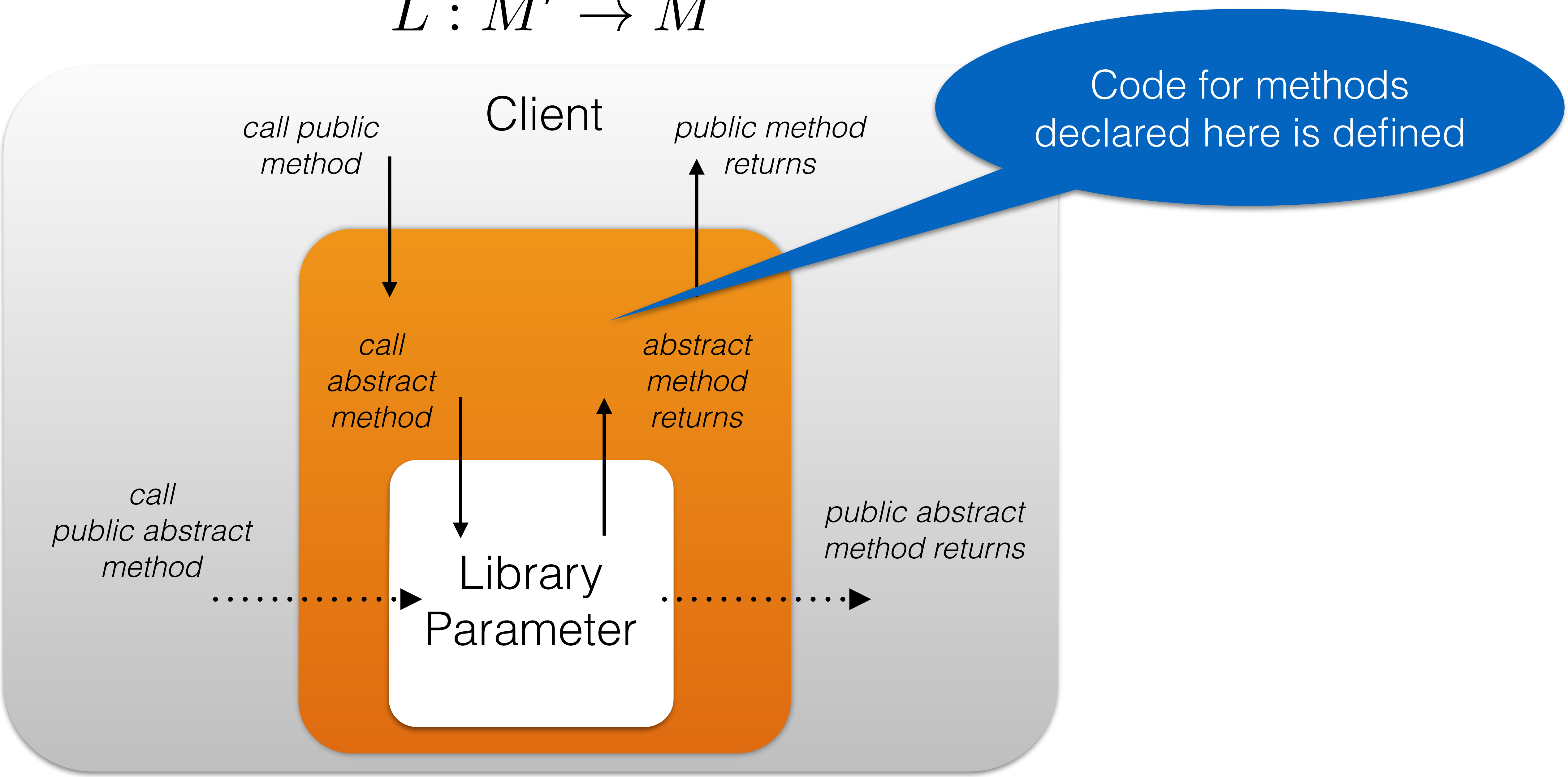


Second Order Libraries

Public methods define the operations made available to clients

Abstract methods (without implementation) can be declared

$$L : M' \rightarrow M$$

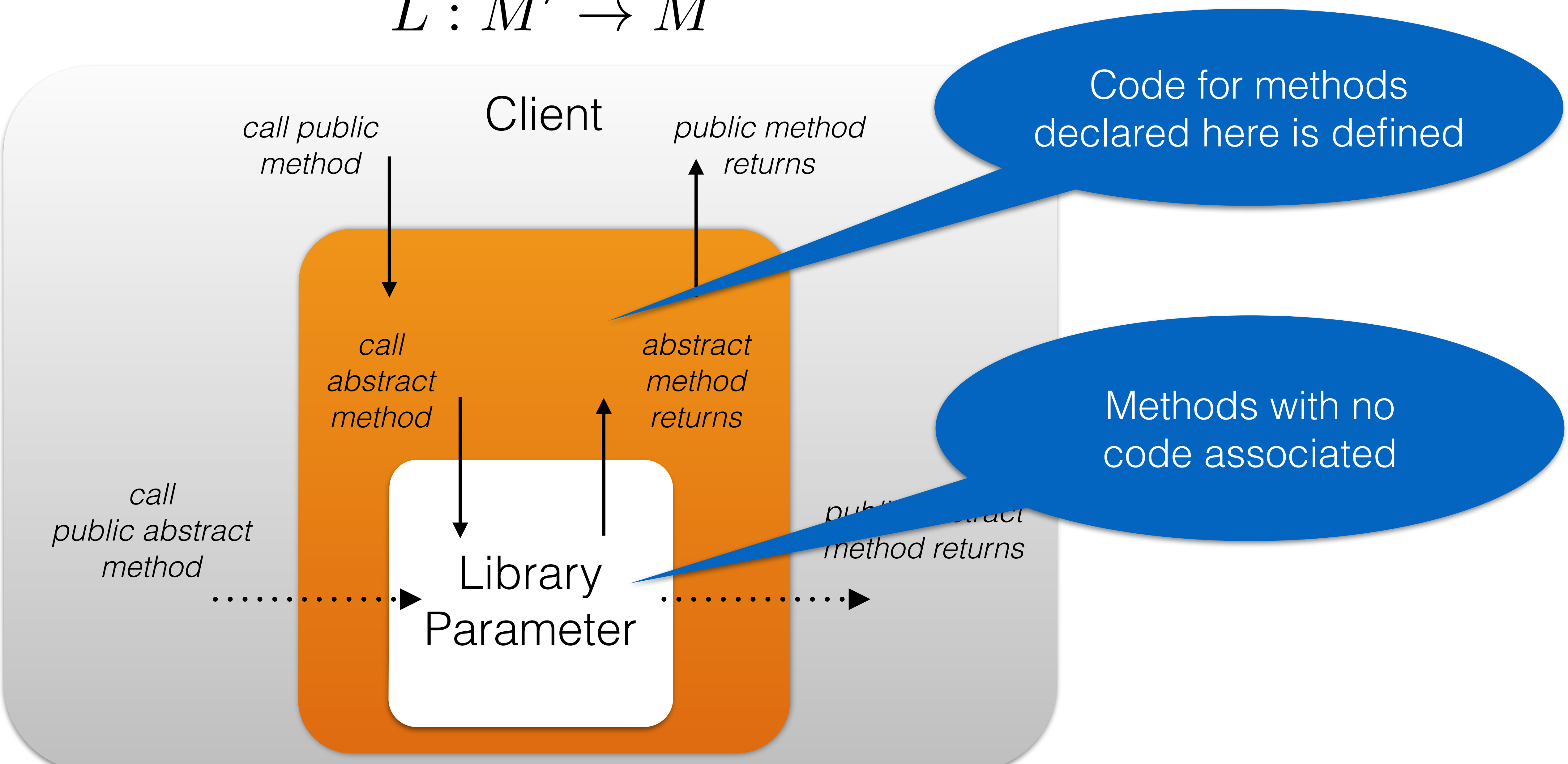


Second Order Libraries

Public methods define the operations made available to clients

Abstract methods (without implementation) can be declared

$$L : M' \to M$$

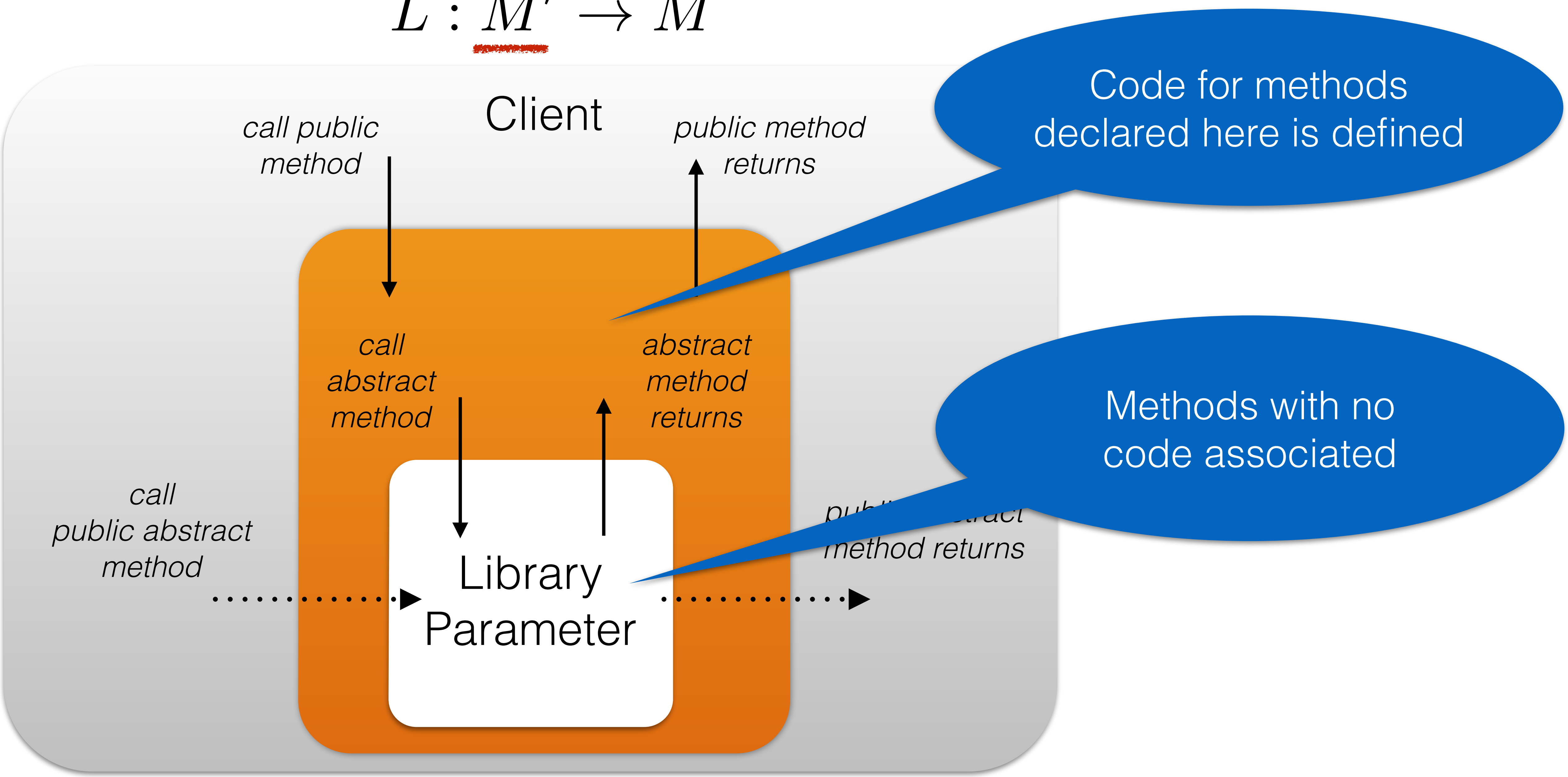


Second Order Libraries

Public methods define the operations made available to clients

Abstract methods (without implementation) can be declared

$$L : M' \rightarrow M$$

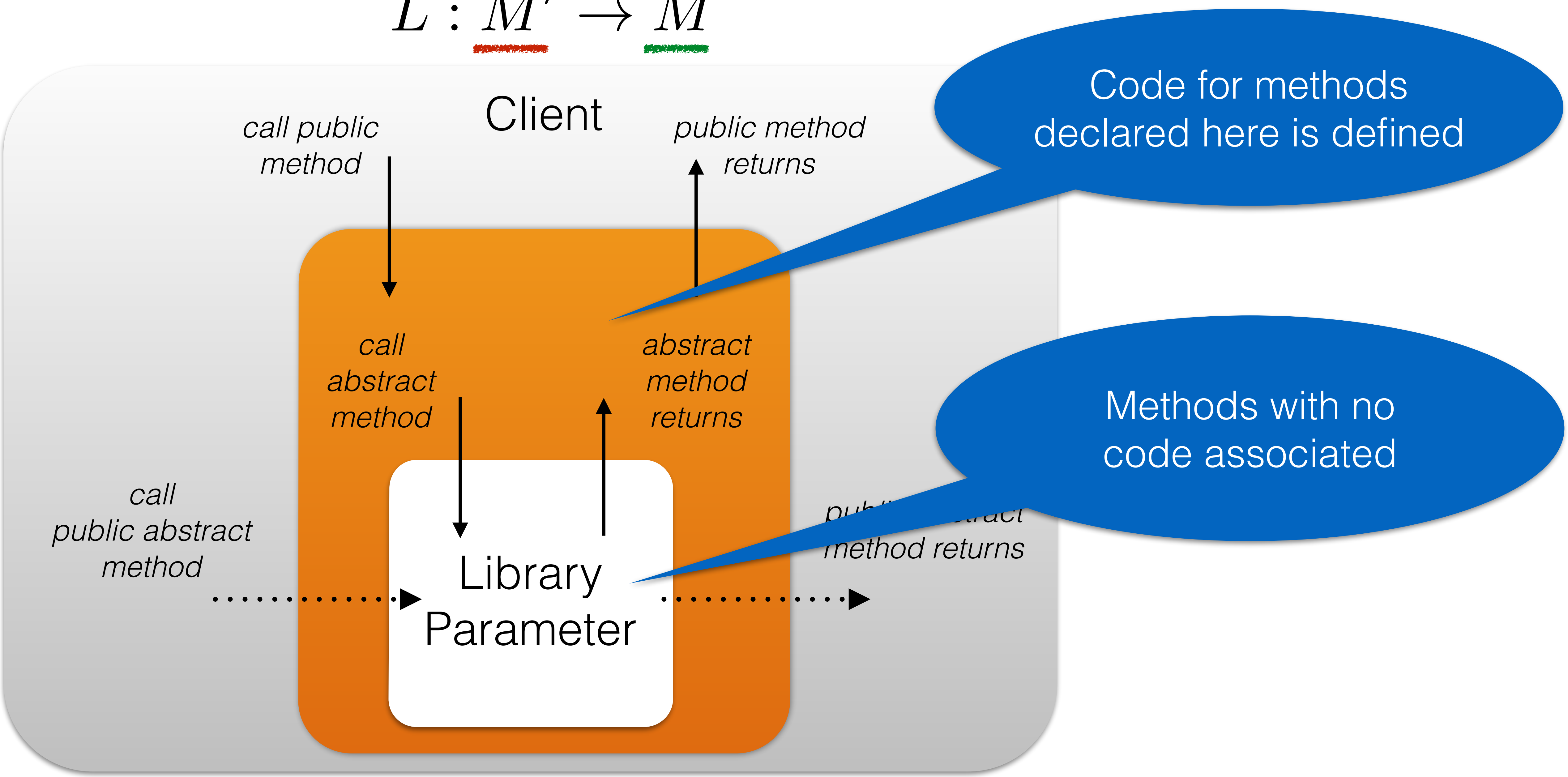


Second Order Libraries

Public methods define the operations made available to clients

Abstract methods (without implementation) can be declared

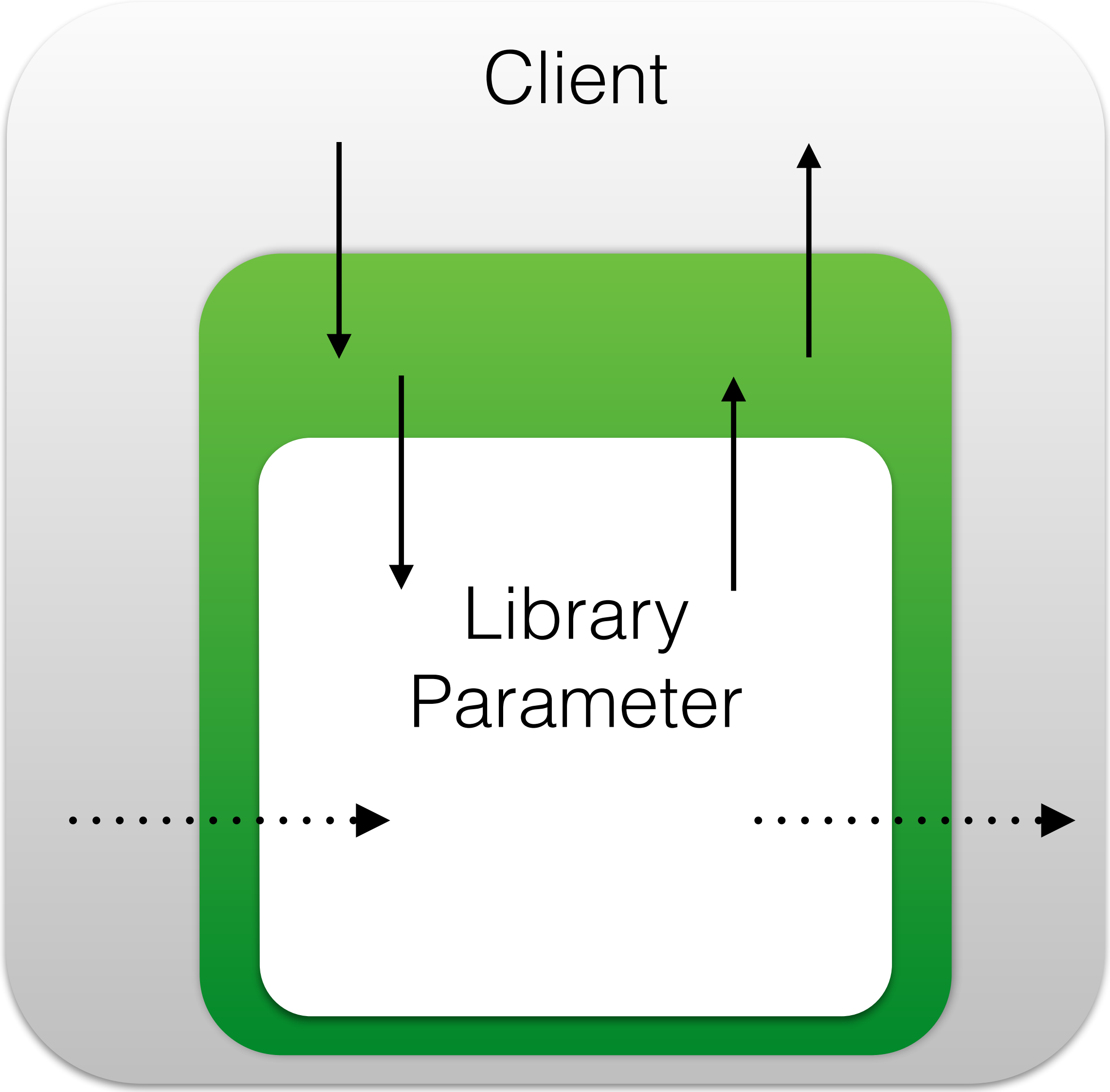
$$L : M' \rightarrow M$$



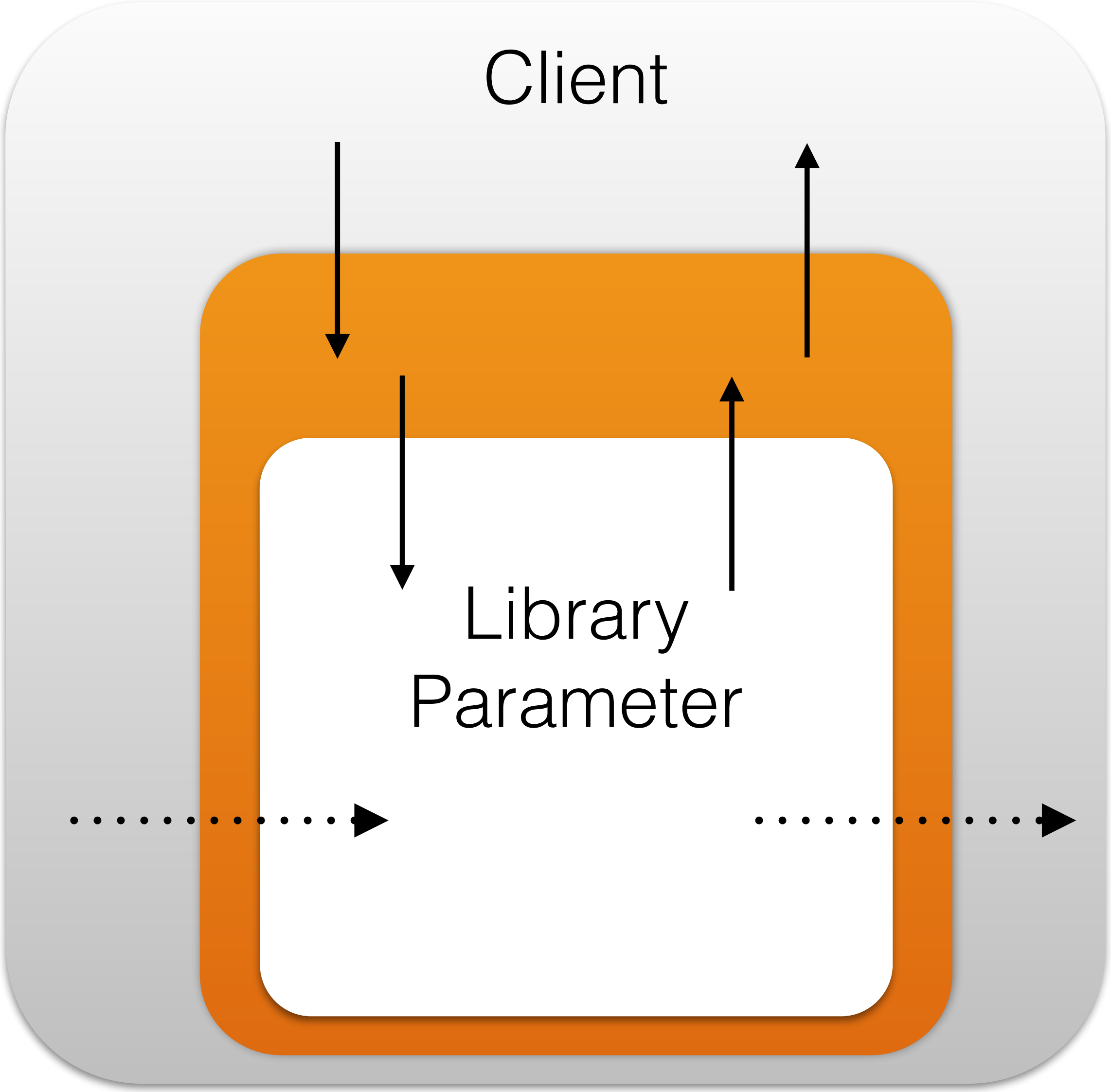
Library Parameter Instantiation

The implementation of abstract methods can be specified by another library

$L_2 : M \rightarrow \underline{M'}$



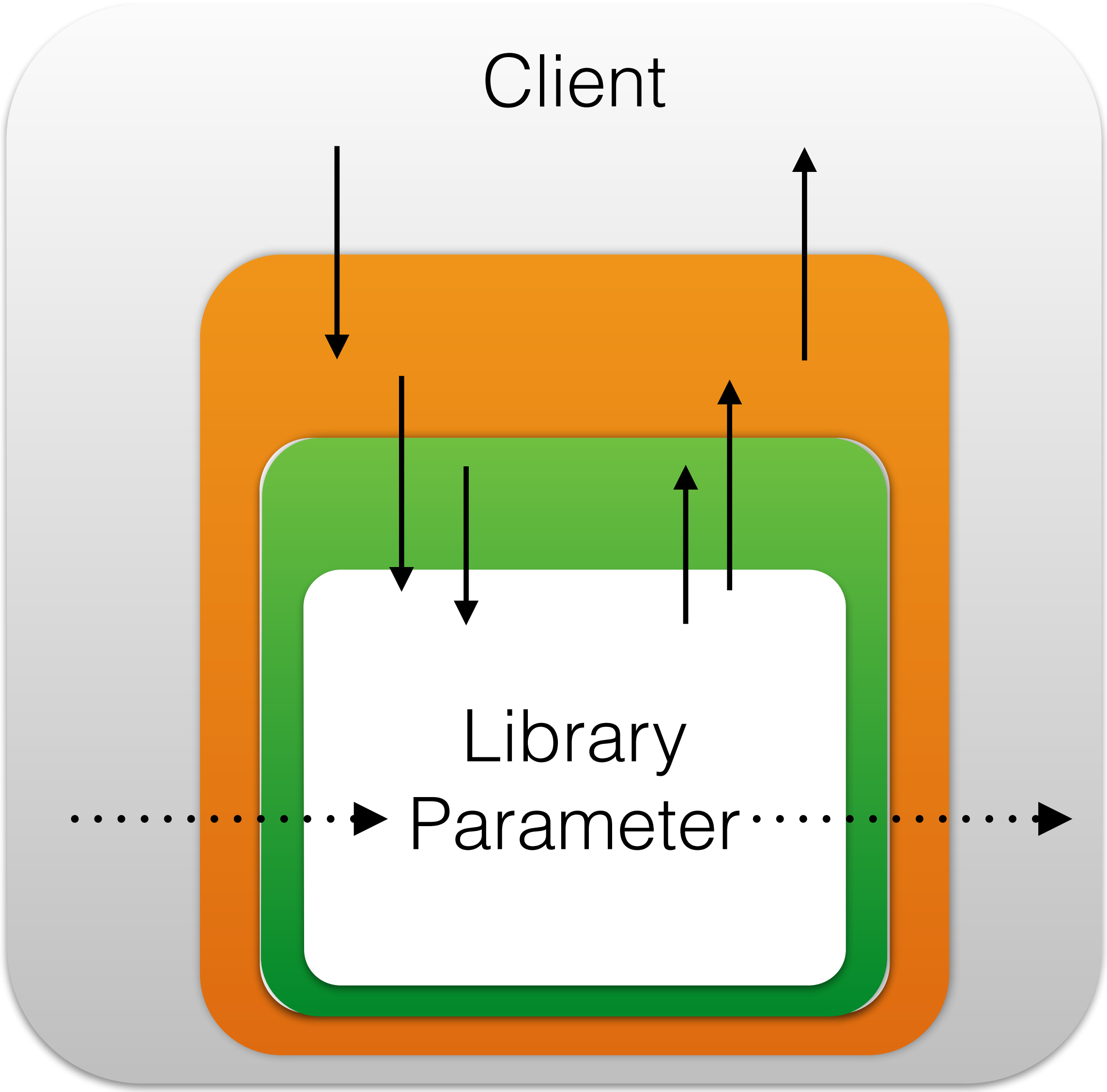
$L_1 : \underline{M'} \rightarrow M''$



Library Parameter Instantiation

The implementation of abstract methods can be specified by another library

$$L_1 \circ L_2 : M \rightarrow M''$$



Observational Refinement

Program: let L in $C_1 \| \cdots \| C_n$

only calls public methods implemented in L

L has no abstract methods

Observational Refinement:

inclusion of client traces, for every possible client
and every possible library parameter

$$L_1 \sqsubseteq_{\mathsf{obs}} L_2 : \forall \mathsf{Cl}. \forall L : \emptyset \to M.\mathsf{Obs}(\mathsf{let}\ (L_1 \circ L)\ \mathsf{in}\ \mathsf{Cl}) \subseteq \mathsf{Obs}(\mathsf{let}\ (L_2 \circ L)\ \mathsf{in}\ \mathsf{Cl})$$

Observational Refinement

Program: let L in $C_1 \| \cdots \| C_n$

only calls public methods implemented in L

L has no abstract methods

Observational Refinement:

inclusion of client traces, for every possible client

and every possible library parameter

$$L_1 \sqsubseteq_{\mathsf{obs}} L_2 : \forall \mathsf{Cl}.\forall L : \emptyset \to M.\mathsf{Obs}(\mathsf{let}\ (L_1 \circ L)\ \mathsf{in}\ \mathsf{Cl}) \subseteq \mathsf{Obs}(\mathsf{let}\ (L_2 \circ L)\ \mathsf{in}\ \mathsf{Cl})$$

Quantification over clients and library parameters

A proof technique is needed

Specifying the behaviour of libraries

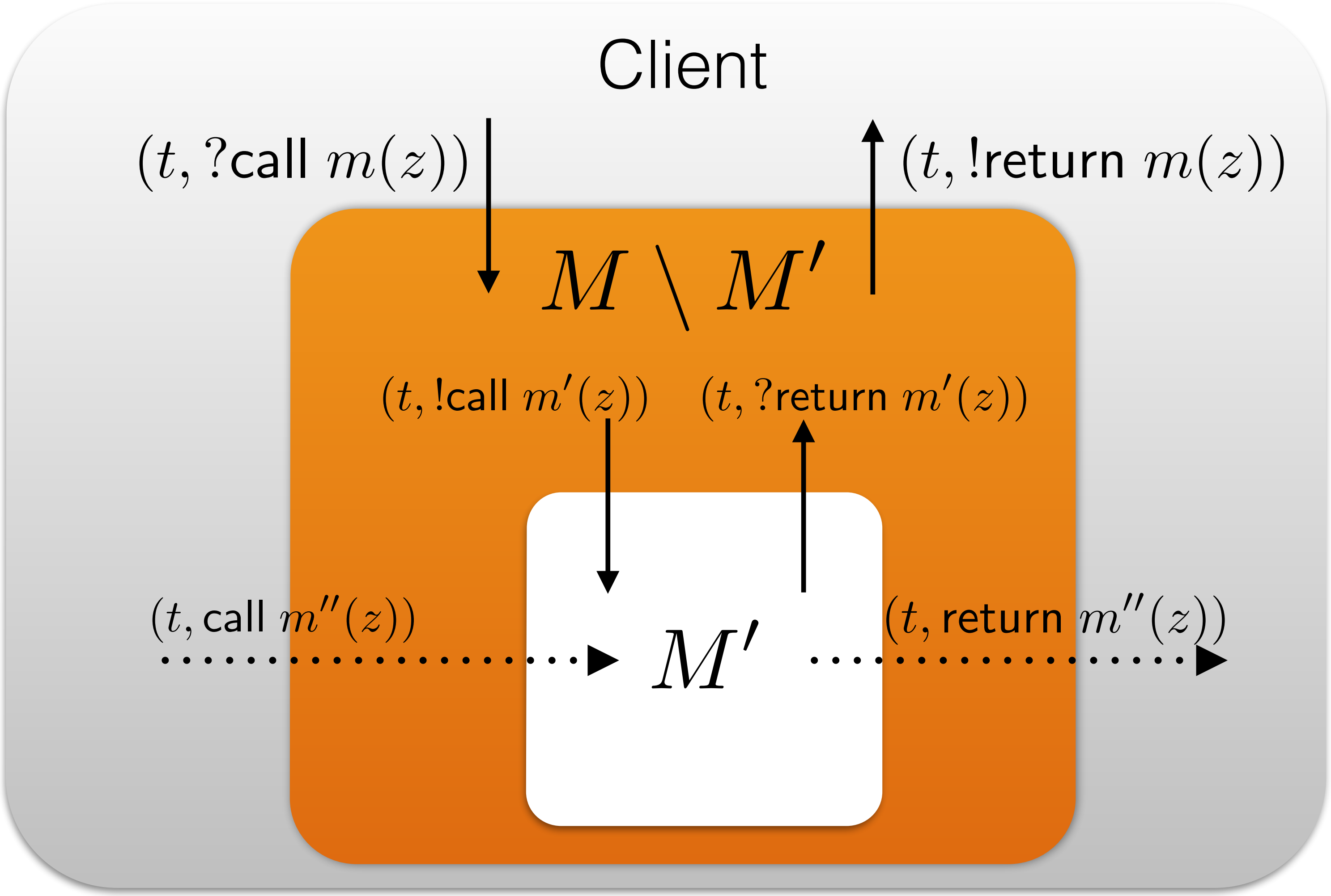
Types for Libraries

$$L : M' \to M$$

Actions:

Observable behaviour of Libraries

(Different entitities execute in different memory spaces)



Specifying the behaviour of libraries

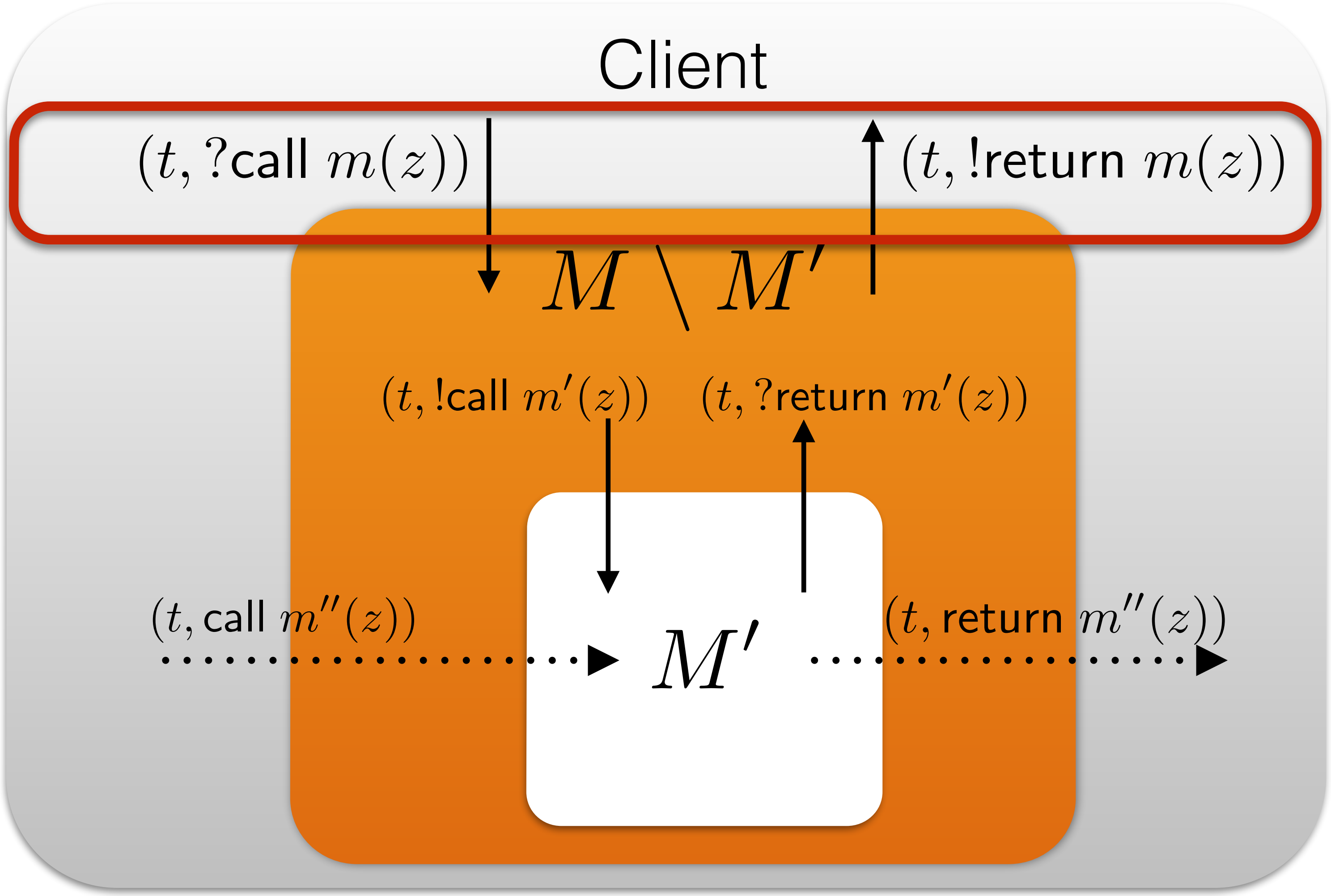
Types for Libraries

$$L : M' \to M$$

Actions:

Observable behaviour of Libraries

(Different entitities execute in different memory spaces)



Interactions with the client $(t, ?\text{call } m(z))$ $(t, !\text{return } m(z))$ $m \in M \setminus M'$

Specifying the behaviour of libraries

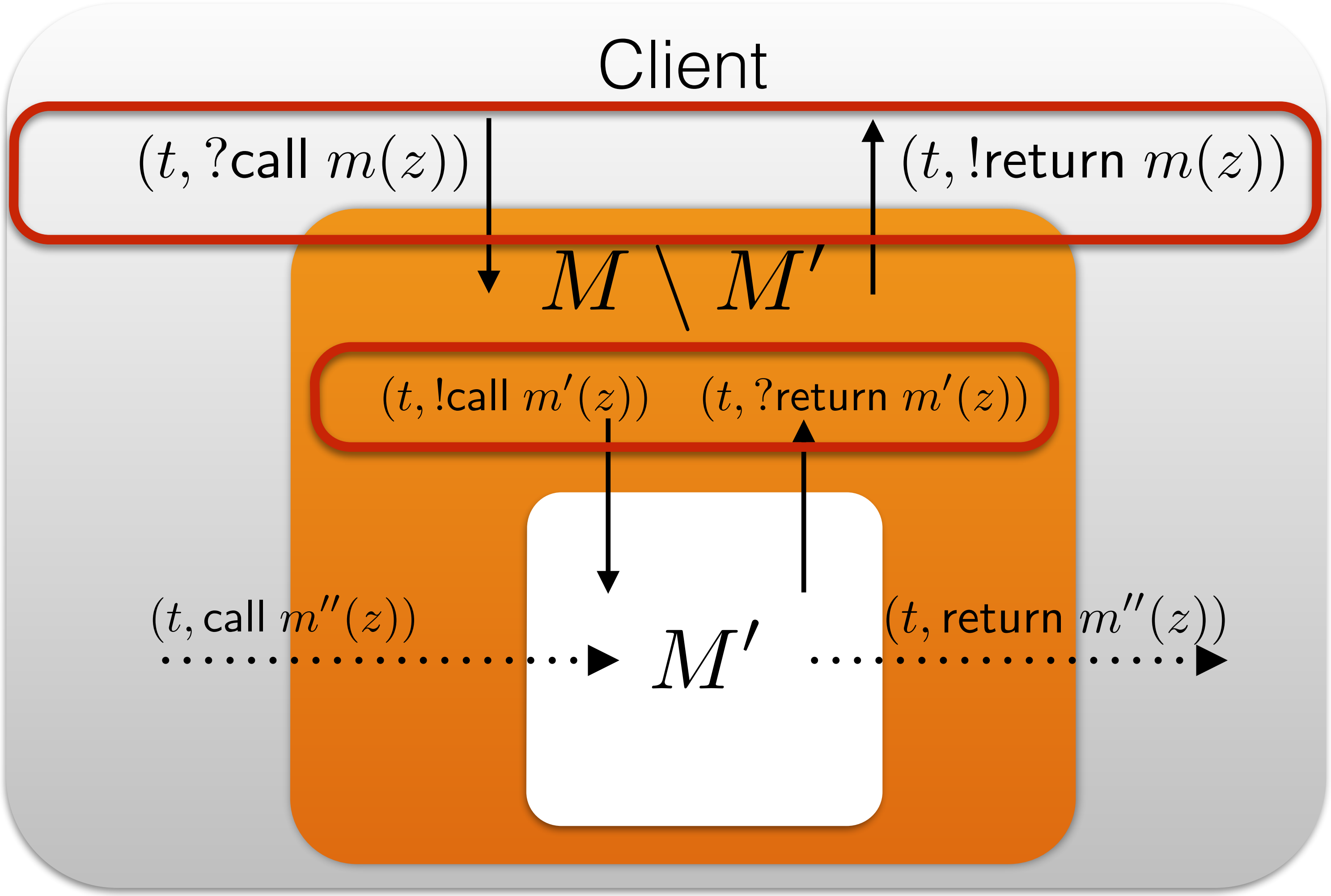
Types for Libraries

$$L : M' \to M$$

Actions:

Observable behaviour of Libraries

(Different entitities execute in different memory spaces)

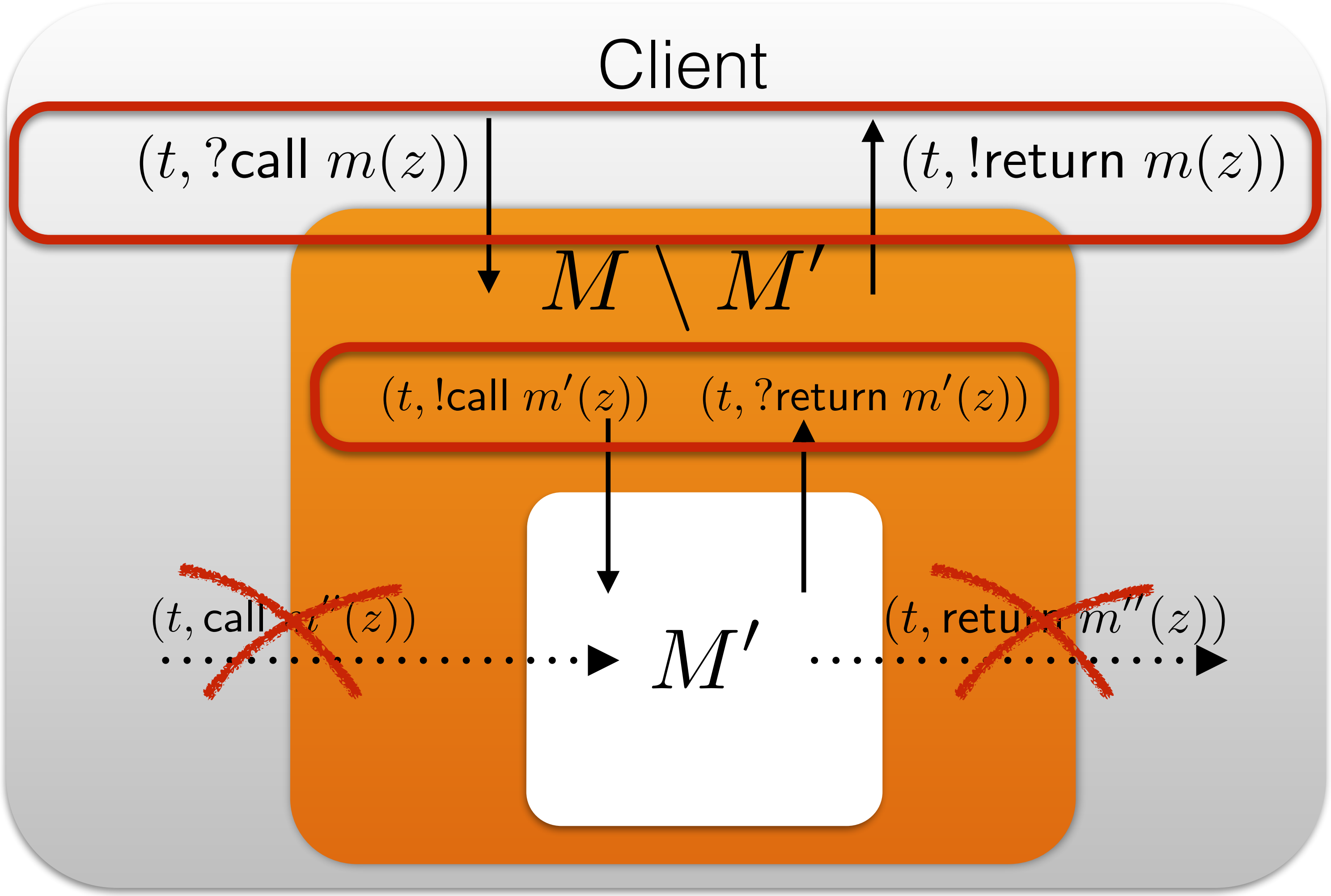


Interactions with the client	$(t, ?\text{call } m(z))$	$(t, !\text{return } m(z))$	$m \in M \setminus M'$
Interactions with the library parameter	$(t, !\text{call } m'(z))$	$(t, ?\text{return } m'(z))$	$m \in M'$

Specifying the behaviour of libraries

Types for Libraries

$$L : M' \to M$$



Actions:

Observable behaviour of Libraries

(Different entitities execute in different memory spaces)

Interactions with the client	$(t, ?\text{call } m(z))$	$(t, !\text{return } m(z))$	$m \in M \setminus M'$
Interactions with the library parameter	$(t, !\text{call } m'(z))$	$(t, ?\text{return } m'(z))$	$m \in M'$

Abstract methods invoked by the client (not observable) $m \in M \cap M'$

Specifying the behaviour of libraries

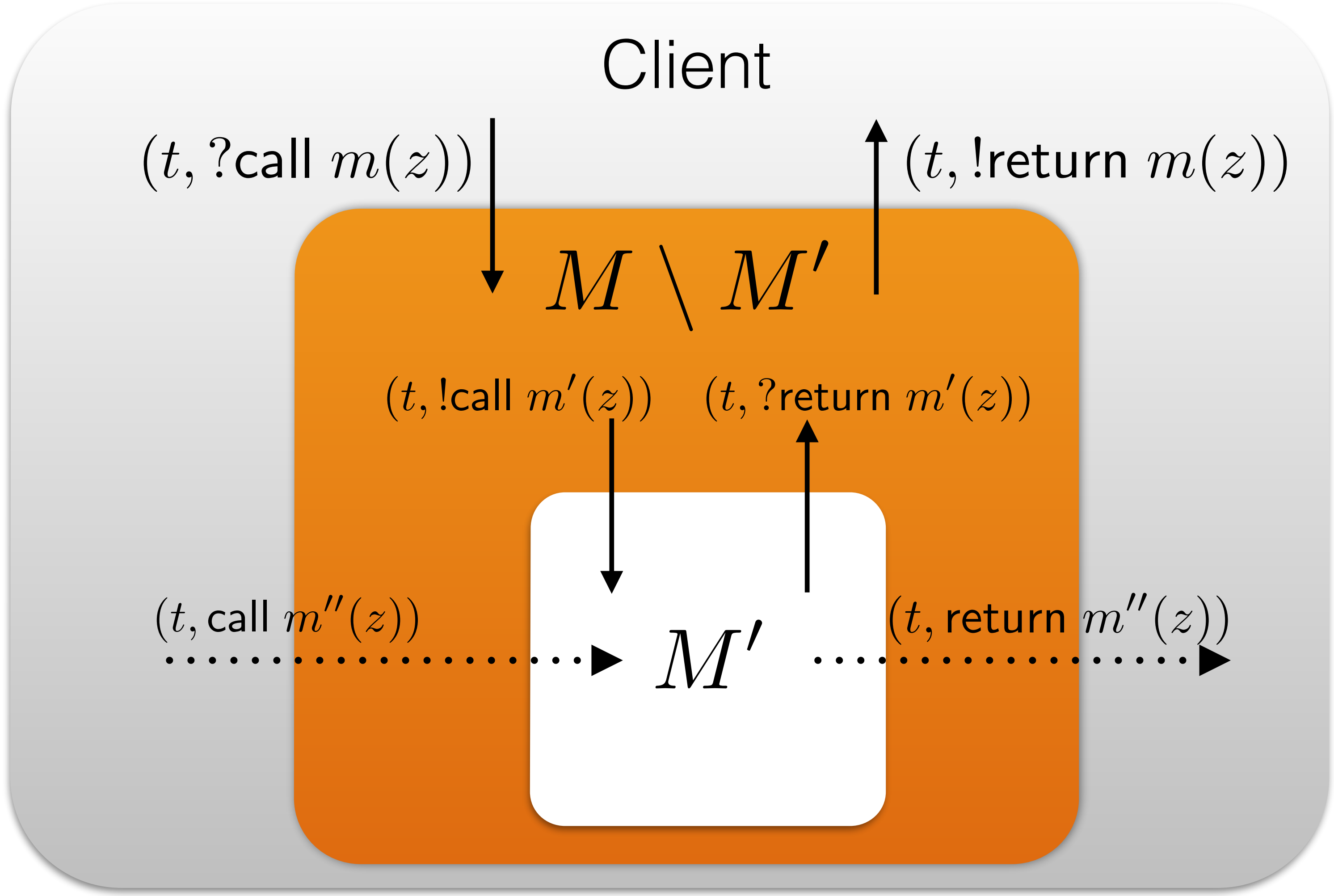
Types for Libraries

$$L : M' \to M$$

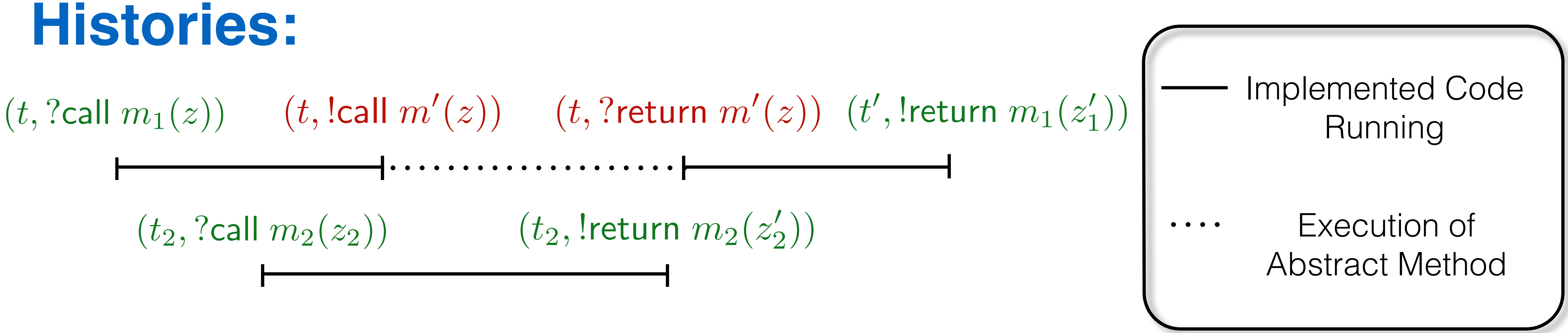
Actions:

Observable behaviour of Libraries

(Different entitities execute in different memory spaces)

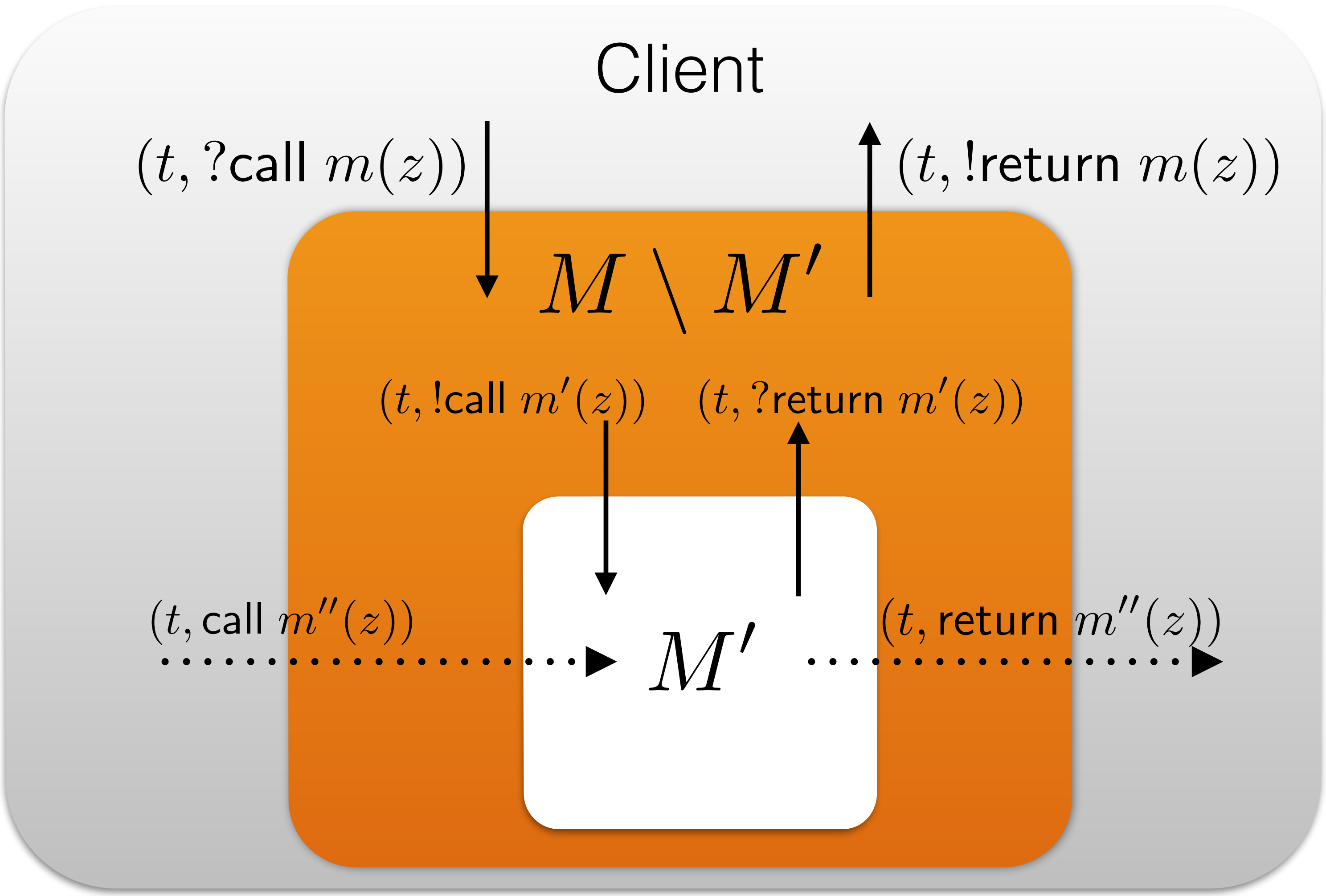


Histories:



Denotational Semantics of Libraries

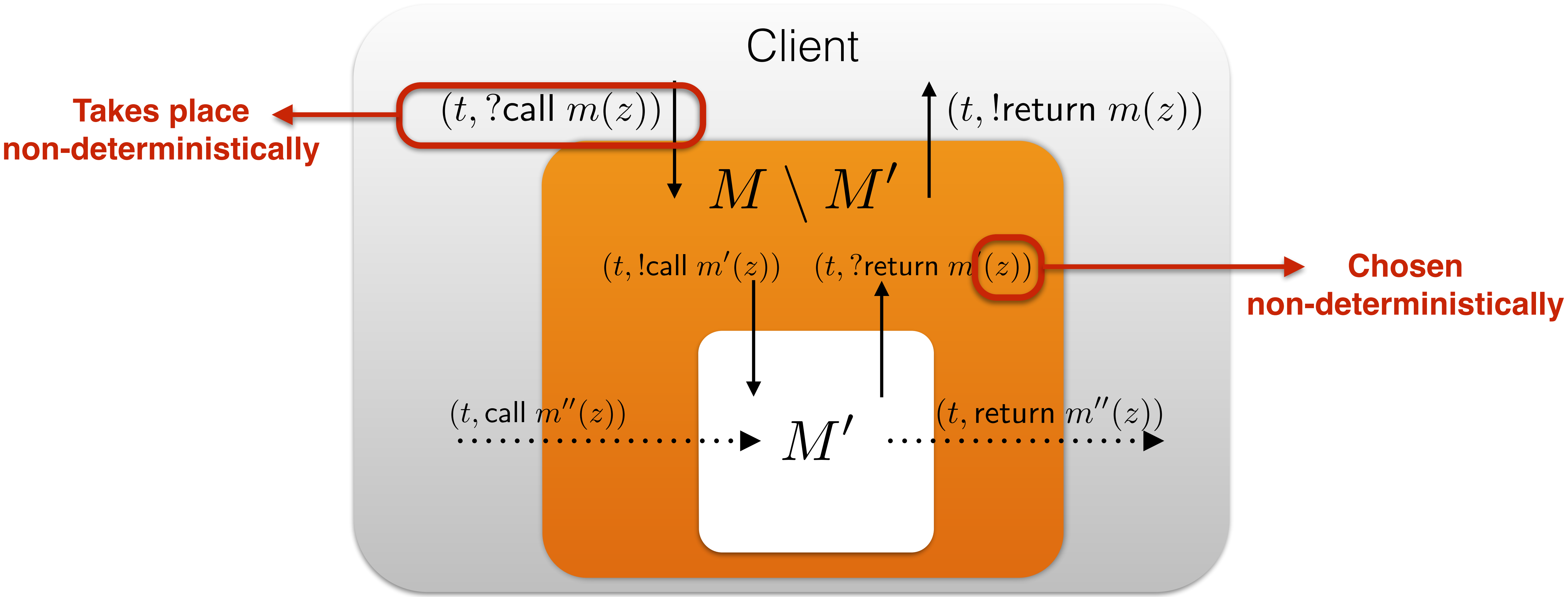
$[\![L]\!]$: histories generated by a library
(arbitrary behaviour of client and library parameter)



Denotational Semantics of Libraries

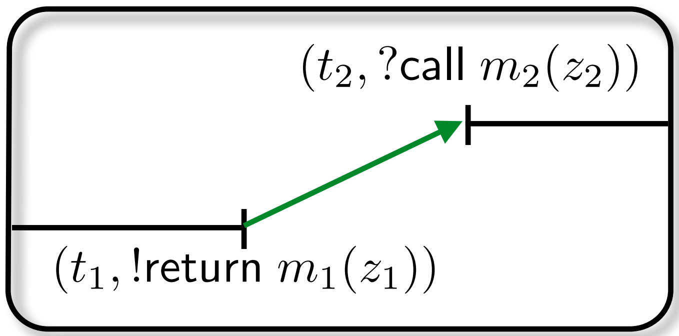
$[\![L]\!]$: histories generated by a library

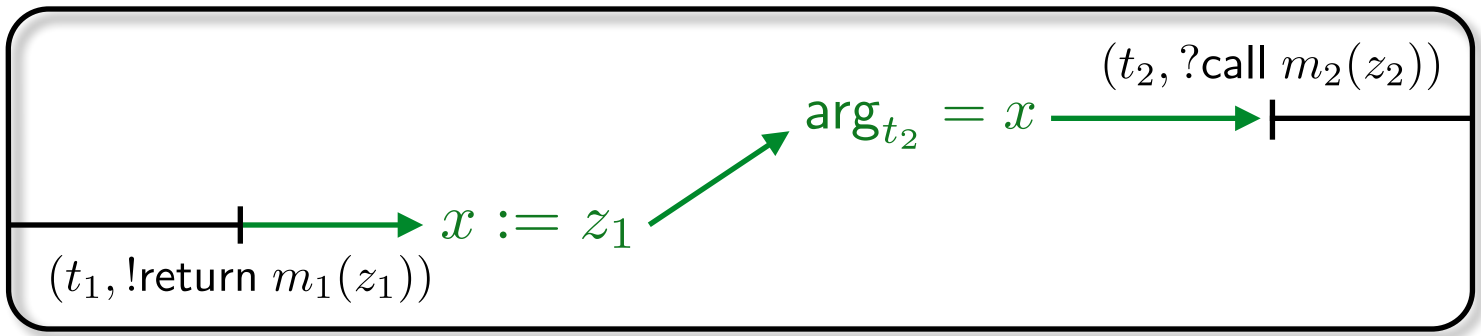
(arbitrary behaviour of client and library parameter)



(General) Linearisability

$h_1 \sqsubseteq h_2$: • h_2 preserves thread-local subhistories of h_1
+ the following order of pairs of actions:

1 

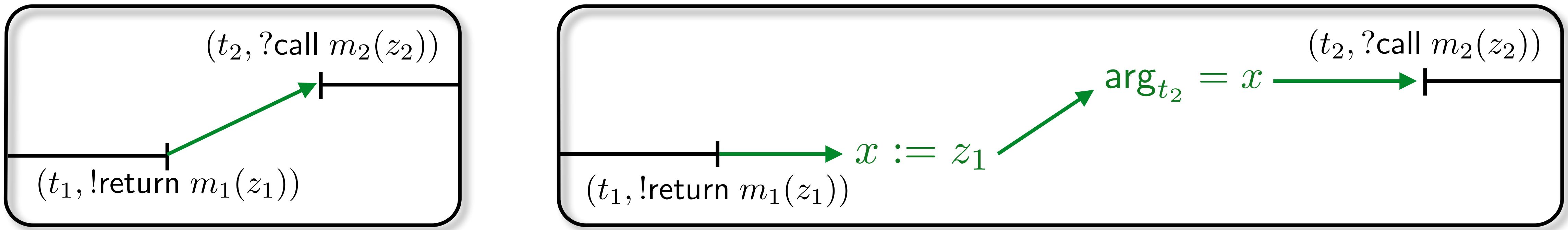


(General) Linearisability

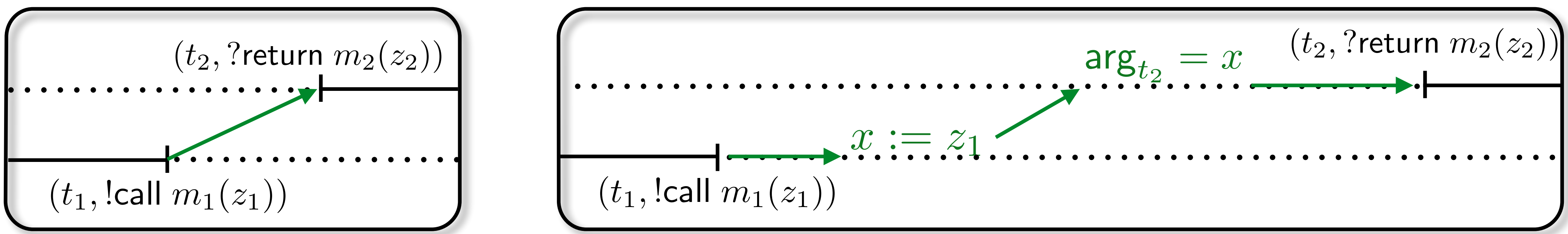
$h_1 \sqsubseteq h_2$: • h_2 preserves thread-local subhistories of h_1

+ the following order of pairs of actions:

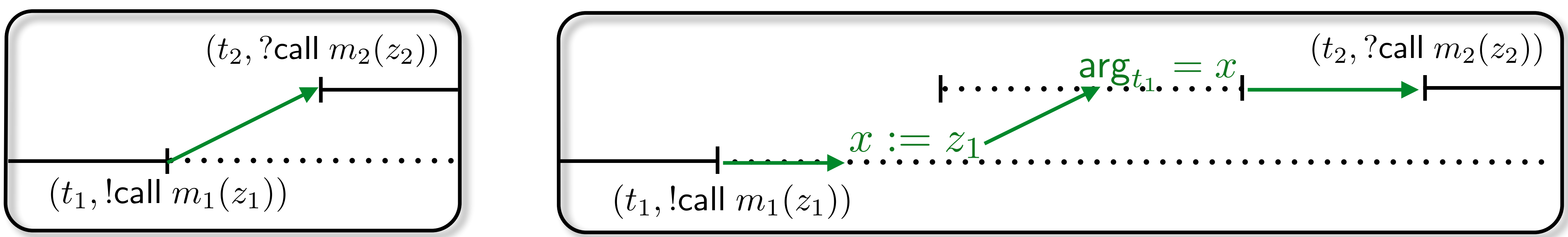
1



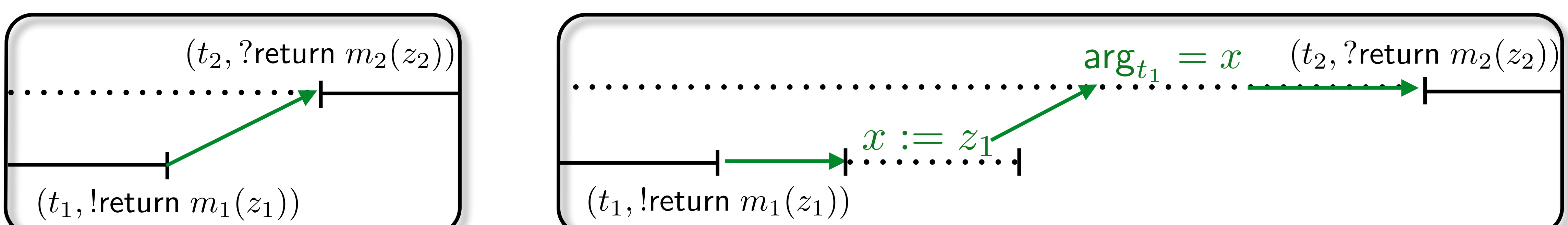
2



3



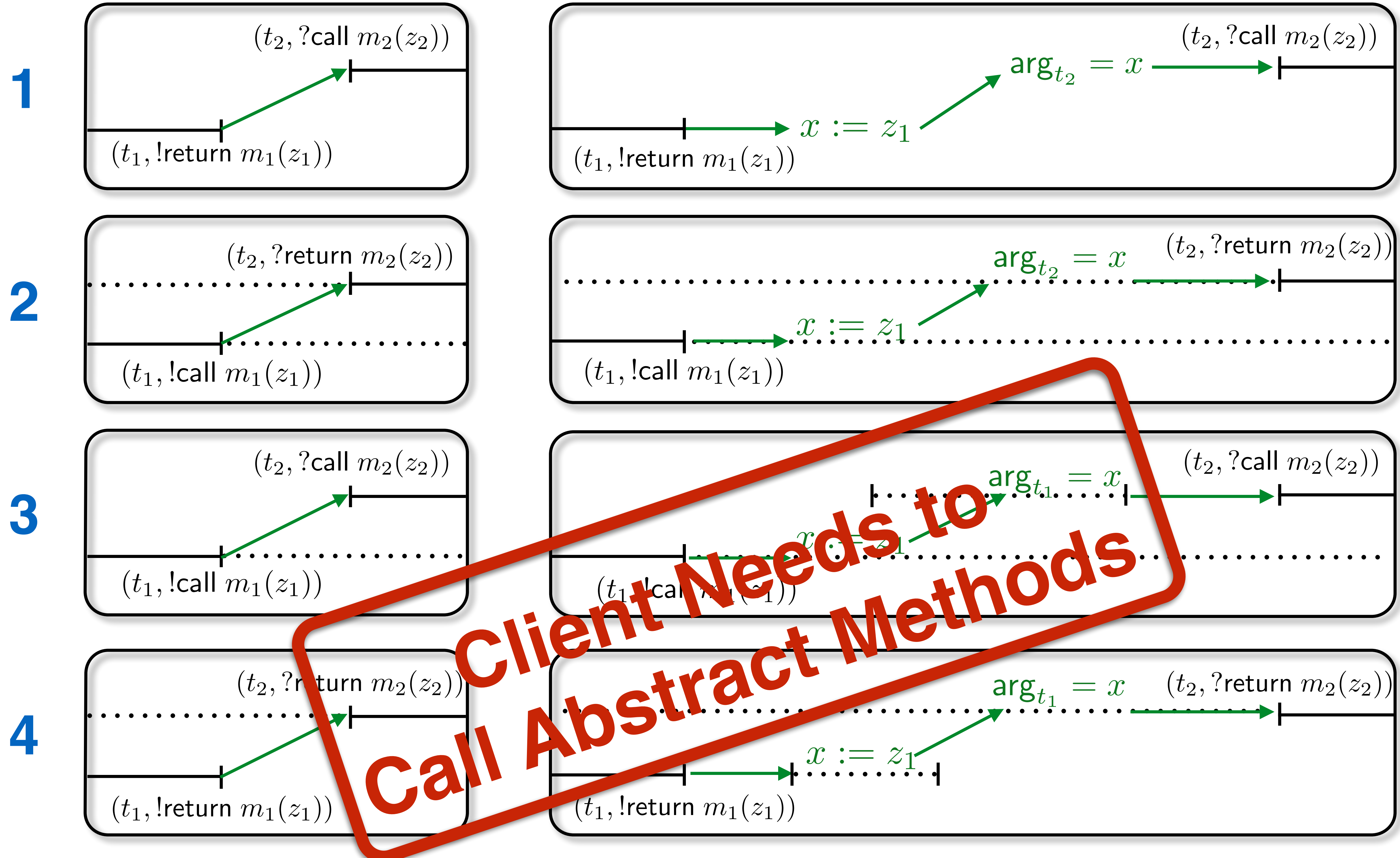
4



(General) Linearisability

$h_1 \sqsubseteq h_2$: • h_2 preserves thread-local subhistories of h_1

+ the following order of pairs of actions:

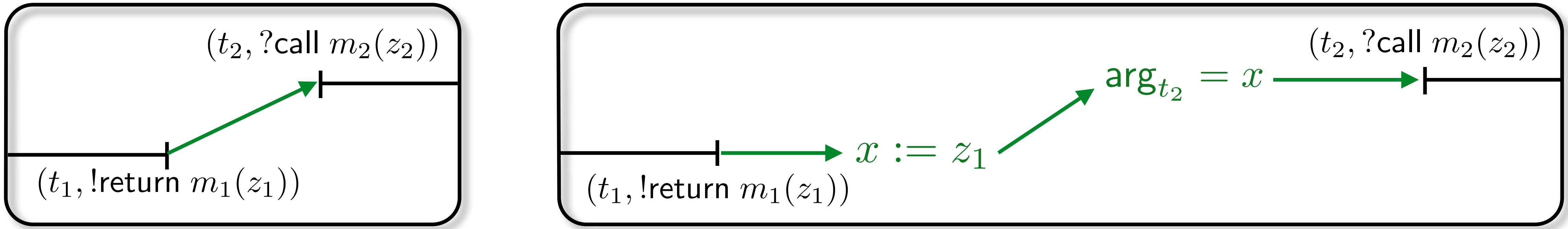


Encapsulated Linearisability

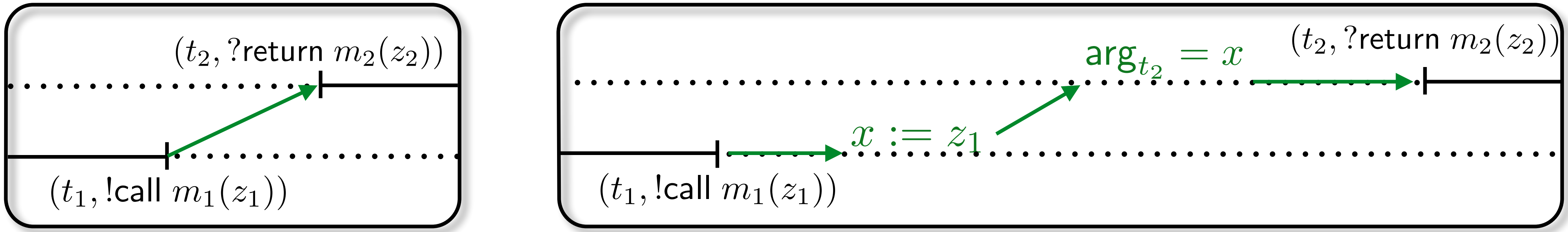
$$L : M' \to M \qquad M' \cap M = \emptyset$$

$h_1 \sqsubseteq_e h_2$: • h_2 preserves thread-local subhistories of h_1
+ the following order of pairs of actions:

1



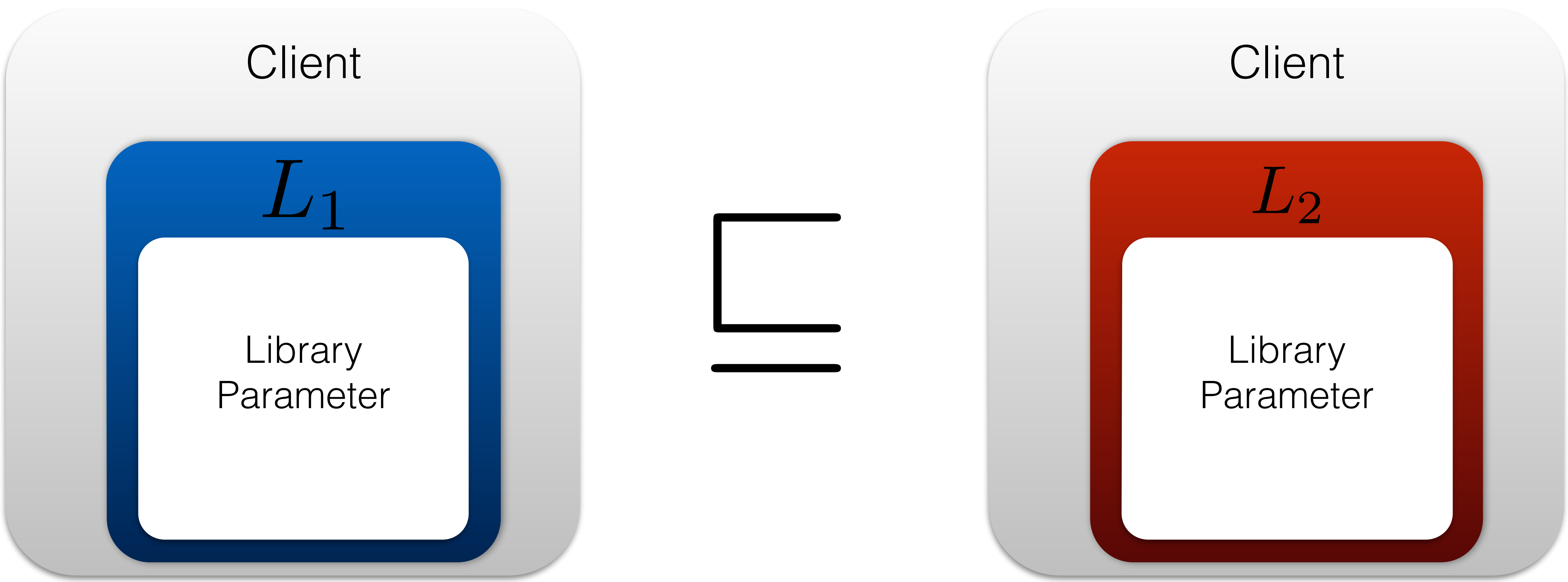
2



Contextuality of Linearisability

Theorem: $L_1, L_2 : M' \to M''$ $\qquad L_{\mathsf{in}} : M \to M'$

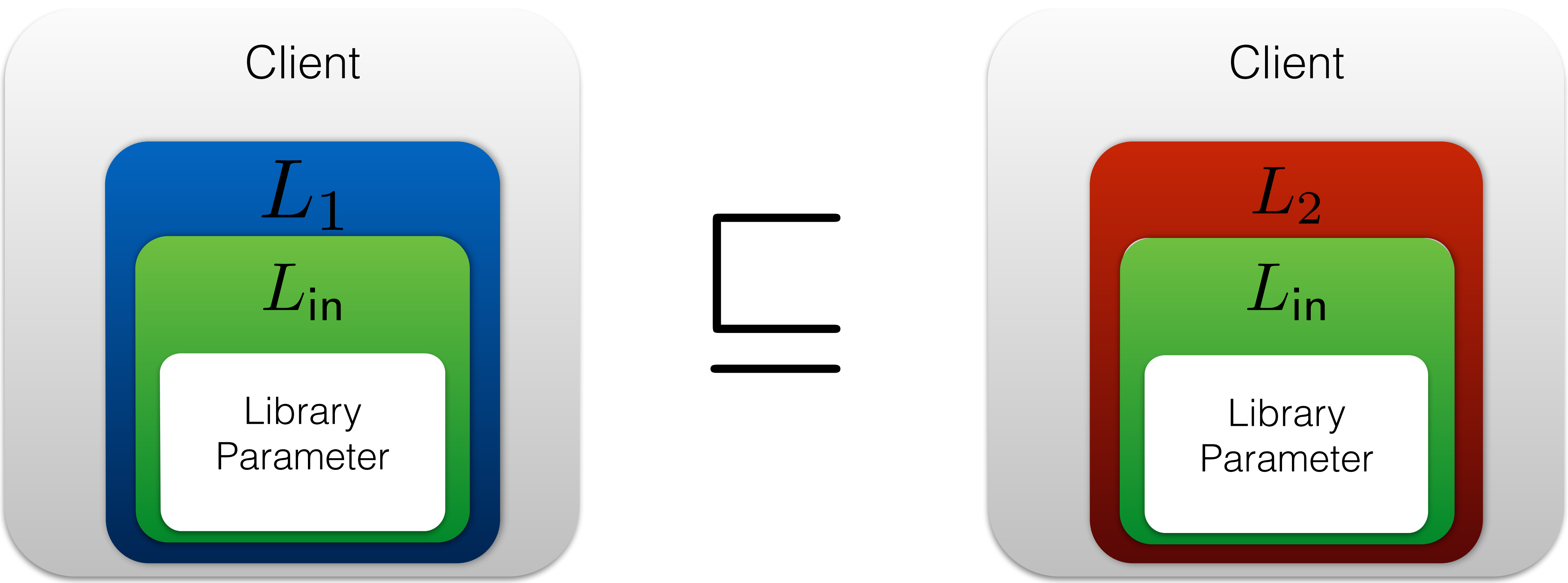
$$L_1 \sqsubseteq L_2 \implies (L_1 \circ L_{\mathsf{in}}) \sqsubseteq (L_2 \circ L_{\mathsf{in}})$$



Contextuality of Linearisability

Theorem: $L_1, L_2 : M' \to M''$ $\quad L_{\mathsf{in}} : M \to M'$

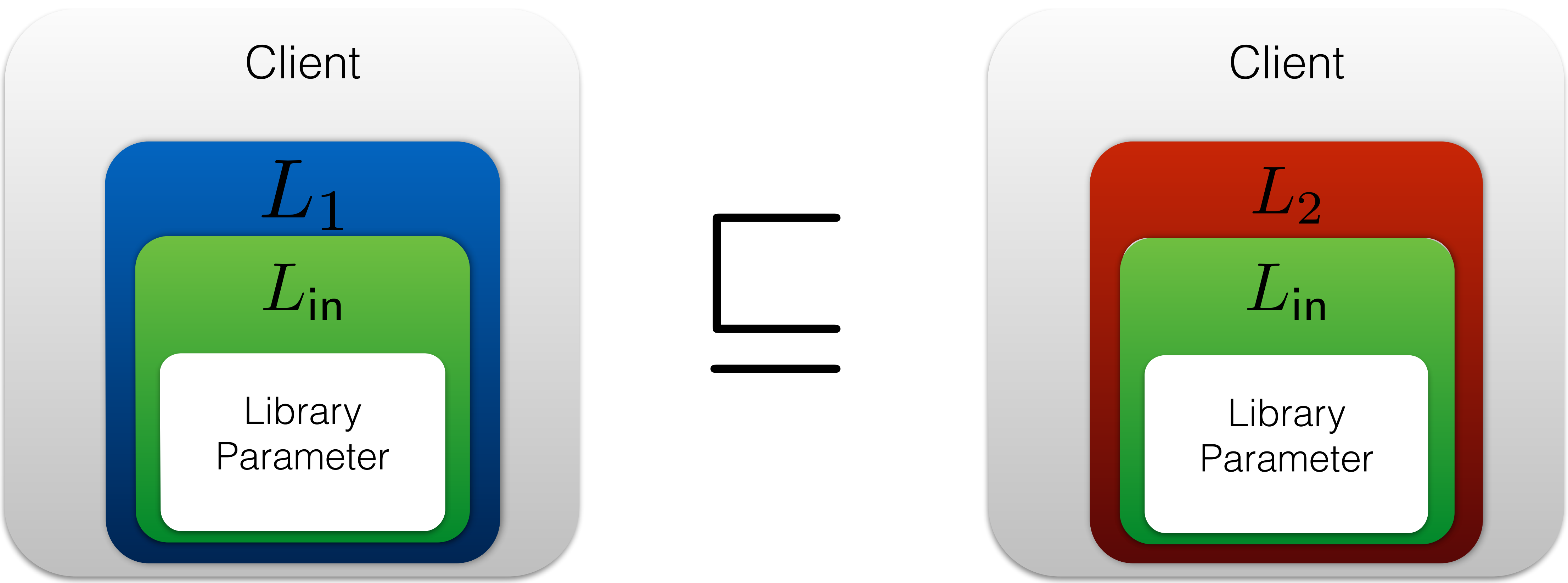
$$L_1 \sqsubseteq L_2 \implies (L_1 \circ L_{\mathsf{in}}) \sqsubseteq (L_2 \circ L_{\mathsf{in}})$$



Contextuality of Linearisability

Theorem: $L_1, L_2 : M' \to M'' \qquad L_{\mathsf{in}} : M \to M'$

$$L_1 \sqsubseteq L_2 \implies (L_1 \circ L_{\mathsf{in}}) \sqsubseteq (L_2 \circ L_{\mathsf{in}})$$



Corollary: $L_1 \sqsubseteq L_2 \implies L_1 \sqsubseteq_{\mathsf{obs}} L_2$

Contextuality of Encapsulated Linearisability

Theorem: for libraries with no public abstract methods encapsulated linearisability is preserved by instantiation of library parameters

Corollary: for libraries with no public abstract methods encapsulated linearisability implies observational refinement

Linearisability does not suffice for Flat Combiners

$$FC : \{m_i\}_{i \in I} \to \{\mathsf{do}_{m_i}\}_{i \in I}$$

a single thread handles all concurrent requests

$$FC_\# : \{m_i\}_{i \in I} \to \{\mathsf{do}_{m_i}\}_{i \in I}$$

calls to abstract methods regulated by a global lock

Instantiating the library parameter

$$int\ m_i()\{return\ \mathsf{getTid}();\}$$

Instantiating the Client

$$t_1 : \mathsf{do}_{m_i}(); \ \| \ t_2 : \mathsf{do}_{m_i}();$$

Linearisability does not suffice for Flat Combiners

$$FC : \{m_i\}_{i \in I} \to \{\mathsf{do}_{m_i}\}_{i \in I}$$

a single thread handles all concurrent requests

$$FC_{\#} : \{m_i\}_{i \in I} \to \{\mathsf{do}_{m_i}\}_{i \in I}$$

calls to abstract methods regulated by a global lock

Instantiating the library parameter

$$int\ m_i()\{return\ \mathsf{getTid}();\}$$

Instantiating the Client

$$t_1 : \mathsf{do}_{m_i}(); \ \| \ t_2 : \mathsf{do}_{m_i}();$$

using $FC_{\#}$:

$\underline{t_1 : \mathsf{do}_{m_i}();}$ $\|$ $\underline{t_2 : \mathsf{do}_{m_i}();}$

always returns t_1 always returns t_2

using FC :

$\underline{t_1 : \mathsf{do}_{m_i}();}$ $\|$ $\underline{t_2 : \mathsf{do}_{m_i}();}$

can return t_2 can return t_1

Linearisability does not suffice for Flat Combiners

$FC : \{m_i\}_{i \in I} \to \{\mathsf{do}_{m_i}\}_{i \in I}$

a single thread handles all concurrent requests

$FC_{\#} : \{m_i\}_{i \in I} \to \{\mathsf{do}_{m_i}\}_{i \in I}$

calls to abstract methods regulated by a global lock

Instantiating the library parameter

$int\ m_i()\{return\ \mathsf{getTid}();\}$

Instantiating the Client

$t_1 : \mathsf{do}_{m_i}(); \parallel t_2 : \mathsf{do}_{m_i}();$

using $FC_{\#}$:

$\underline{t_1 : \mathsf{do}_{m_i}();} \parallel \underline{t_2 : \mathsf{do}_{m_i}();}$

always returns t_1 always returns t_2

using FC :

$\underline{t_1 : \mathsf{do}_{m_i}();} \parallel \underline{t_2 : \mathsf{do}_{m_i}();}$

can return t_2 can return t_1

$$FC \not\sqsubseteq_{\mathsf{Obs}} FC_{\#}$$

$$\Longrightarrow$$

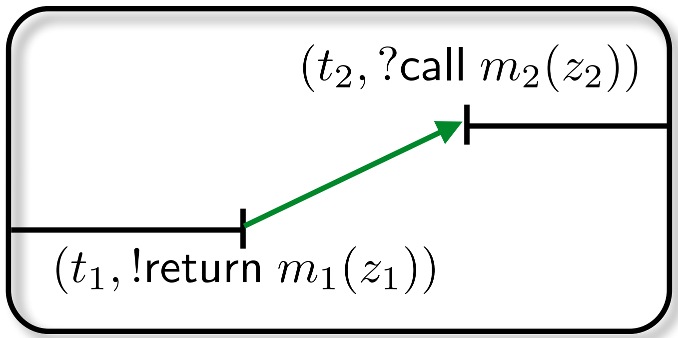
$$FC \not\sqsubseteq FC_{\#}$$

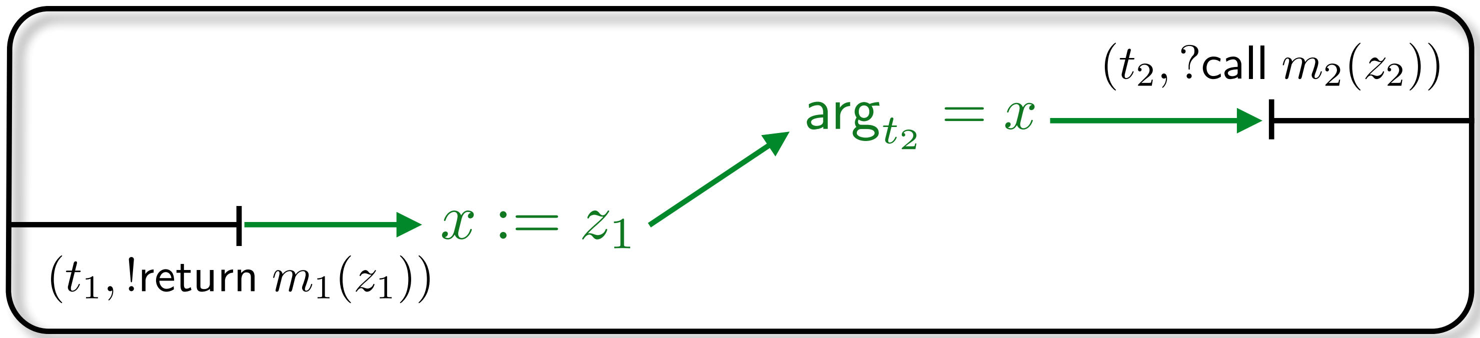
Properties Satisfied by Flat Combiners

$FC_{\#}$ linearises FC

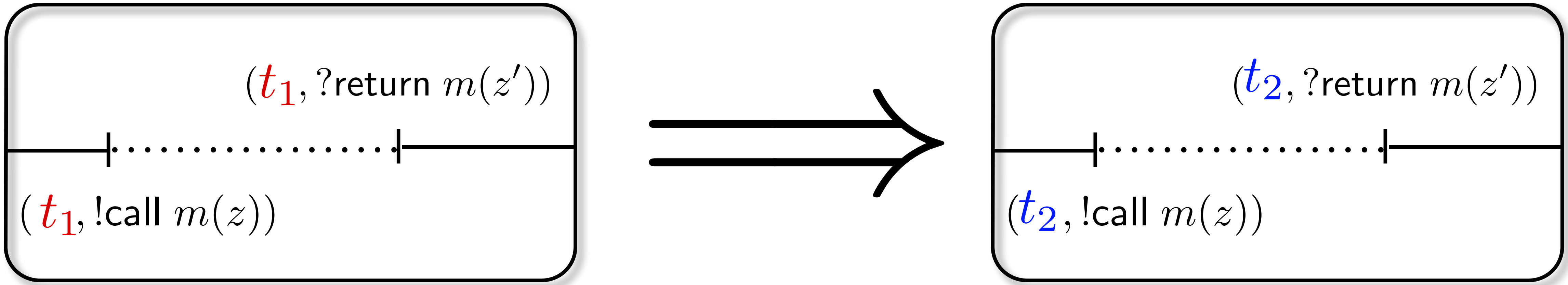
if the library parameter does not use thread local information

1





2



Up-to Linearisability and Observational Refinement

$M' \to M, M' \cap M = \emptyset$

$\mathcal{R}$: binary relation between sequences of calls and returns to methods in M'

$h_1 \sqsubseteq_{\mathcal{R}} h_2$:

- $h_1|_{\mathsf{ClAct}} \sqsubseteq h_2|_{\mathsf{ClAct}}$
- $h_1|_{\mathsf{AbsAct}} \, \mathcal{R} \, h_2|_{\mathsf{AbsAct}}$

Up-to Linearisability and Observational Refinement

$M' \to M, M' \cap M = \emptyset$

$\mathcal{R}$: binary relation between sequences of calls and returns to methods in M'

$h_1 \sqsubseteq_{\mathcal{R}} h_2$:
- $h_1|_{\mathsf{ClAct}} \sqsubseteq h_2|_{\mathsf{ClAct}}$
- $h_1|_{\mathsf{AbsAct}} \, \mathcal{R} \, h_2|_{\mathsf{AbsAct}}$

(a variant of contextuality holds for $\sqsubseteq_{\mathcal{R}}$)

Soundness: $L_1 \sqsubseteq_{\mathcal{R}} L_2 \implies L_1 \sqsubseteq^{\mathcal{R}}_{\mathsf{obs}} L_2$

only $\mathcal{R}$- closed library parameters are considered

Up-to Linearisability and Observational Refinement

$M' \to M, M' \cap M = \emptyset$

$\mathcal{R}$: binary relation between sequences of calls and returns to methods in M'

$h_1 \sqsubseteq_{\mathcal{R}} h_2$:
- $h_1|_{\mathsf{ClAct}} \sqsubseteq h_2|_{\mathsf{ClAct}}$
- $h_1|_{\mathsf{AbsAct}} \mathcal{R}\ h_2|_{\mathsf{AbsAct}}$

(a variant of contextuality holds for $\sqsubseteq_{\mathcal{R}}$)

Soundness: $L_1 \sqsubseteq_{\mathcal{R}} L_2 \implies L_1 \sqsubseteq^{\mathcal{R}}_{\mathsf{obs}} L_2$

only $\mathcal{R}$- closed library parameters are considered

$$\mathsf{FC} \sqsubseteq_{\mathcal{R}_t} \mathsf{FC}^{\#} \implies \mathsf{FC} \sqsubseteq^{\mathcal{R}_t}_{\mathsf{obs}} \mathsf{FC}^{\#}$$

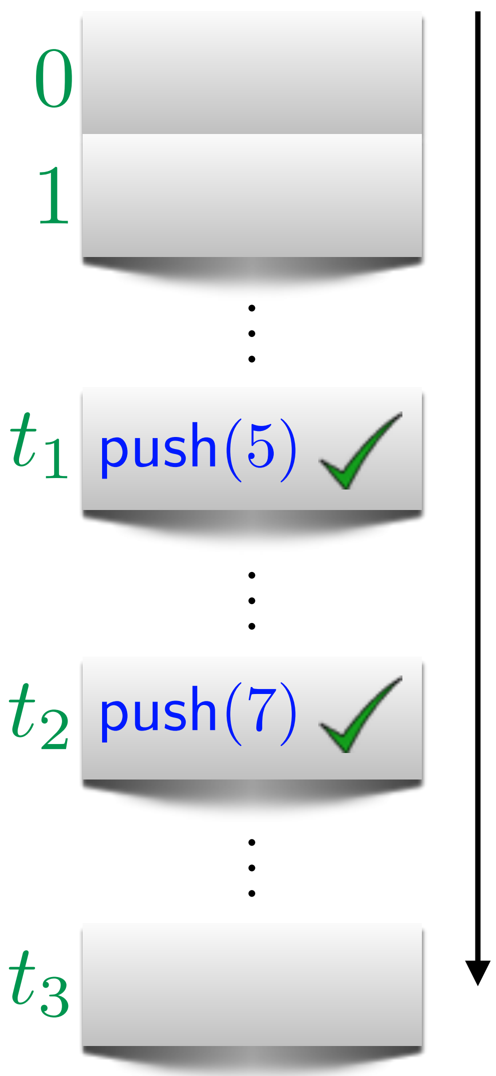
$\mathcal{R}_t$: equivalence of histories up-to thread identifiers

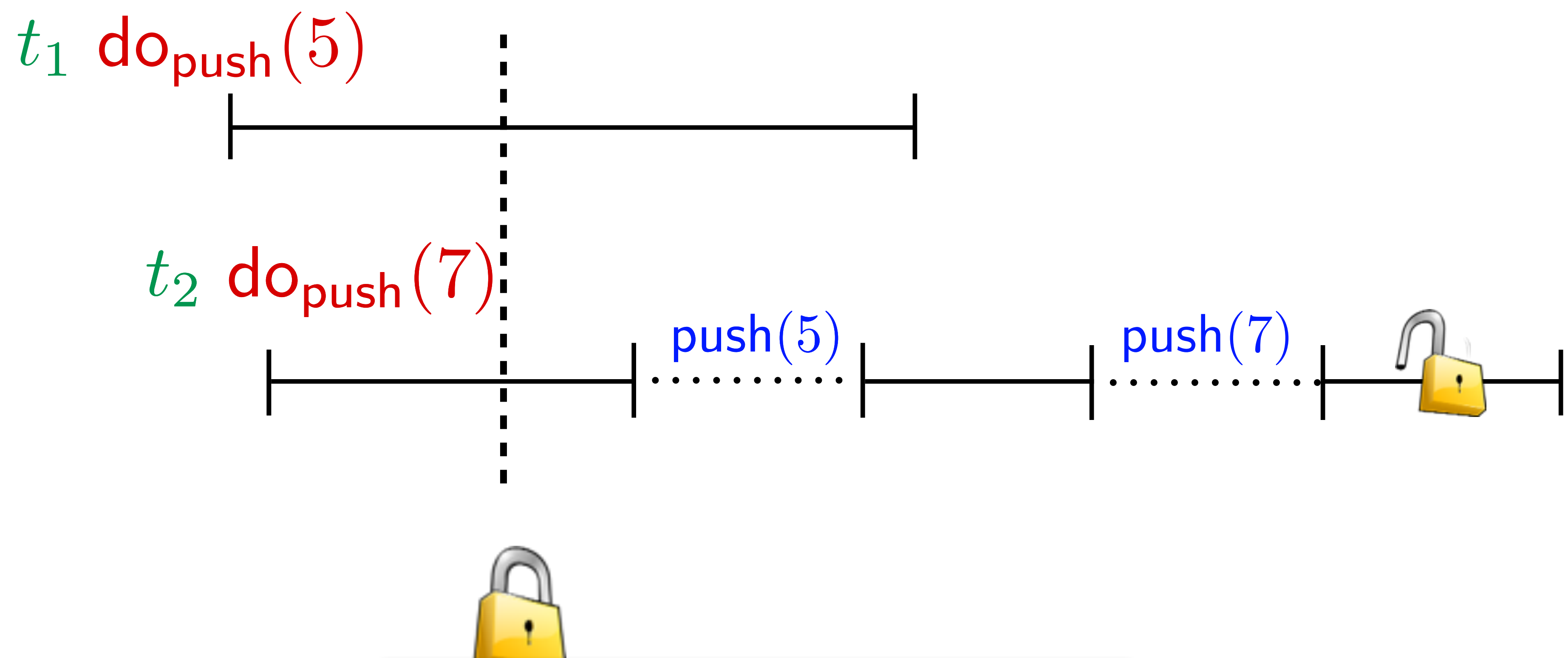
Flat Combiners (Hendler et al, 2010)

Idea: let a single thread handle all requests

Stack provides methods push, pop to clients

$\mathsf{FC}_{\mathsf{stack}}$: access to methods in Stack regulated by $\mathsf{do}_{\mathsf{push}}$, $\mathsf{do}_{\mathsf{pop}}$





It appears to behave as the
naive
solution

- push , pop n... ct methods)
- Guaranteed to have correct behaviour for (**almost**) any implementation of abstract methods

Flat Combiners (Hendler et al, 2010)

Idea: let a single thread handle all requests

Stack provides methods push, pop to clients

$\mathsf{FC_{stack}}$: access to methods in Stack regulated by $\mathsf{do_{push}}$, $\mathsf{do_{pop}}$

0

1

⋮

t_1 push(5) ✓

⋮

t_2 push(7) ✓

⋮

t_3

t_1 $\mathsf{do_{push}}(5)$

t_2 $\mathsf{do_{push}}($

$\mathsf{h}(7)$

Theorem:

$$FC \sqsubseteq^{\mathcal{R}}_{\mathsf{obs}} FC_{\#}$$

- push , pop not i[...] methods)
- Guaranteed to have correct behaviour for (**almost**) any implementation of abstract methods

Conclusions

Future Research:

- Other applications (Joins, Iterators, …)
- STM and Transactional Boosting
- Linearisability for Higher Order Objects
- Connections with other forms of HO-reasoning (e.g. CaReSL)

Conclusions

Future Research:

- Other applications (Joins, Iterators, …)
- STM and Transactional Boosting
- Linearisability for Higher Order Objects
- Connections with other forms of HO-reasoning (e.g. CaReSL)

Verification of Flat Combiners

Goal: $FC \sqsubseteq_{\mathcal{R}_t} FC_{\#}$

Verification of Flat Combiners

Goal: $FC \sqsubseteq_{\mathcal{R}_t} FC_{\#}$

- Construct the set of histories $[\![FC]\!]$
  - Code analysis $\Longrightarrow$ State Transition System
  - Individuation of properties satisfied by $h \in [\![FC]\!]$

Verification of Flat Combiners

Goal: $FC \sqsubseteq_{\mathcal{R}_t} FC_{\#}$

- Construct the set of histories $[\![FC]\!]$
  - Code analysis $\Longrightarrow$ State Transition System
  - Individuation of properties satisfied by $h \in [\![FC]\!]$
- $h \in [\![FC]\!] \mapsto h'$
  - Changing threads in abstract calls and returns to match corresponding requests from client

Verification of Flat Combiners

Goal: $FC \sqsubseteq_{\mathcal{R}_t} FC_{\#}$

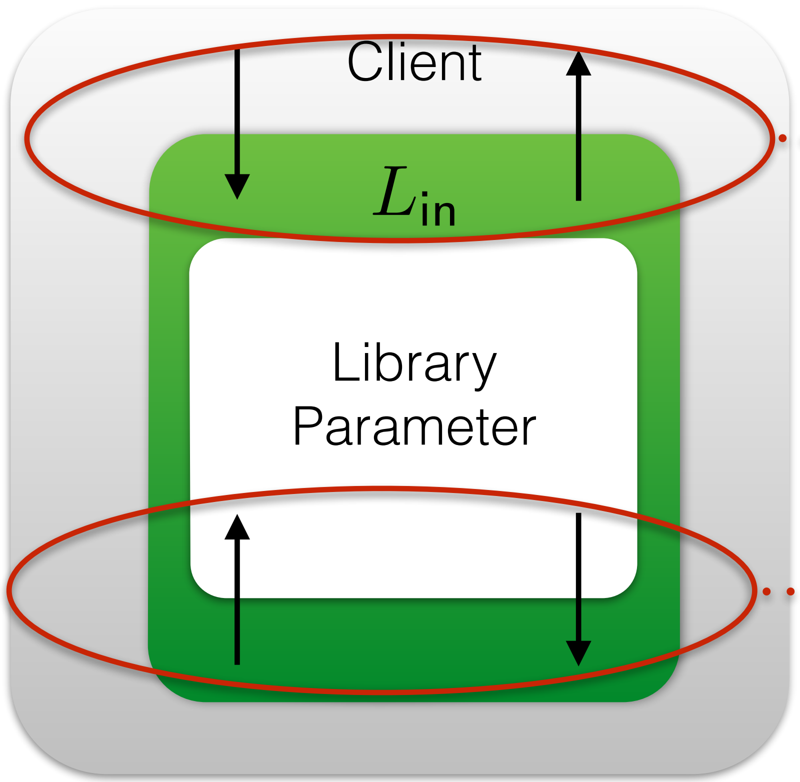
- Construct the set of histories $[\![FC]\!]$
  - Code analysis $\implies$ State Transition System
  - Individuation of properties satisfied by $h \in [\![FC]\!]$
- $h \in [\![FC]\!] \mapsto h'$
  - Changing threads in abstract calls and returns to match corresponding requests from client
- $h' \mapsto h''$
  - $h'|_{\mathsf{ClAct}} \sqsubseteq h''|_{\mathsf{ClAct}}$
  - $h'|_{\mathsf{AbsAct}} = h''|_{\mathsf{AbsAct}}$

Goal: $FC \sqsubseteq_{\mathcal{R}_t} FC_{\#}$

- Construct the set of histories $[\![FC]\!]$
  - Code analysis $\Longrightarrow$ State Transition System
  - Individuation of properties satisfied by $h \in [\![FC]\!]$
- $h \in [\![FC]\!] \mapsto h'$
  - Changing threads in abstract calls and returns to match corresponding requests from client
- $h' \mapsto h''$
  - $h'|_{\mathsf{ClAct}} \sqsubseteq h''|_{\mathsf{ClAct}}$
  - $h'|_{\mathsf{AbsAct}} = h''|_{\mathsf{AbsAct}}$
- Show that $h'' \in [\![FC_{\#}]\!]$

Contextuality of Up-to Linearisability

$$L_{\mathsf{in}} : M \rightarrow M'$$



rearranging this part of a history according to $\mathcal{R}$

causes this part of the history to be rearranged according to $\mathcal{G}$

$\mathcal{R}$ relates sequences of actions belonging to $M' \setminus M$

$\mathcal{G}$ relates sequences of actions belonging to M

$\binom{\mathcal{R}}{\mathcal{G}}$-closure: defines how closure properties of library parameters have to be changed when a inner library is instantiated

Contextuality of Up-to Linearisability

Theorem

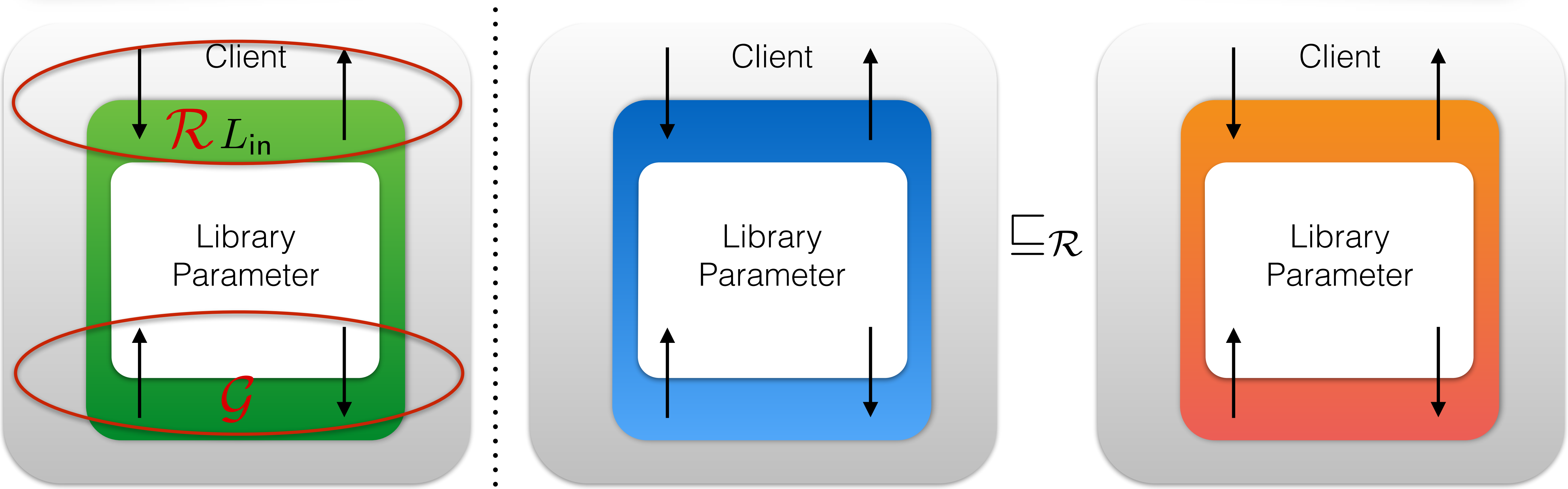
$L_1, L_2 : M' \to M''$

$L_1 \sqsubseteq_{\mathcal{R}} L_2$

$L_{\mathsf{in}} : M \to M'$

L_{in} is $\binom{\mathcal{R}}{\mathcal{G}}$-closed

$$\Longrightarrow$$



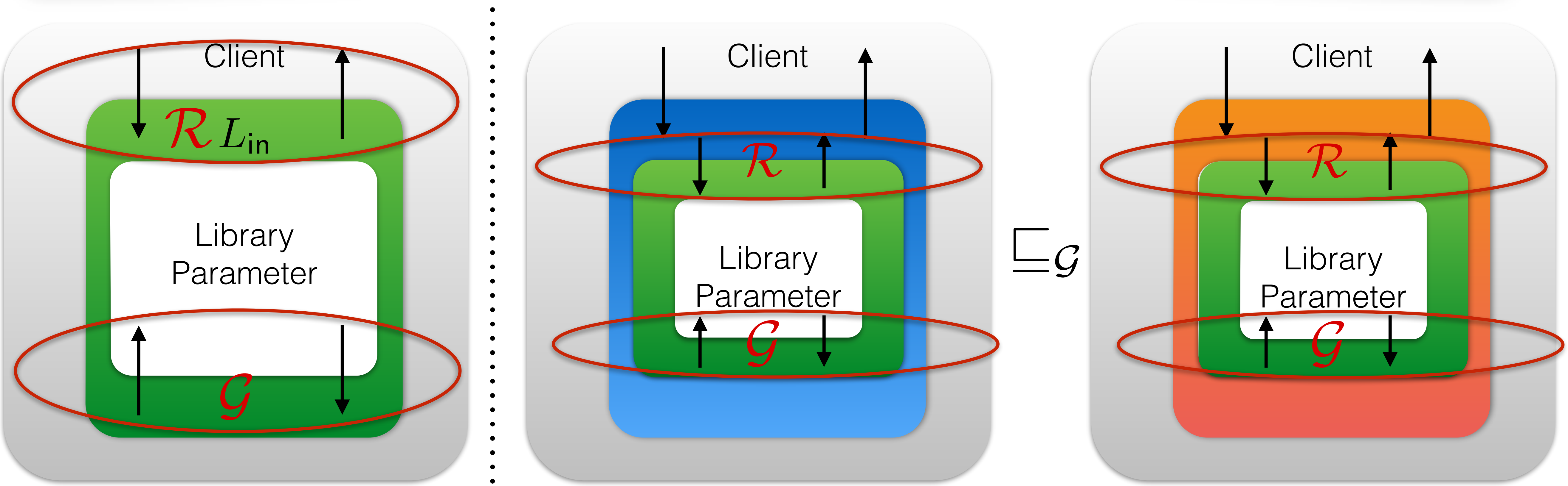
Contextuality of Up-to Linearisability

Theorem

$L_1, L_2 : M' \to M''$
$L_1 \sqsubseteq_{\mathcal{R}} L_2$
$L_{\text{in}} : M \to M'$
L_{in} is $\binom{\mathcal{R}}{\mathcal{G}}$-closed

$$\Longrightarrow$$

$$M' \cap M'' = \emptyset \;\wedge\; M \cap M' = \emptyset \implies (L_1 \circ L_{\text{in}}) \sqsubseteq_{\mathcal{G}} (L_2 \circ L_{\text{in}})$$



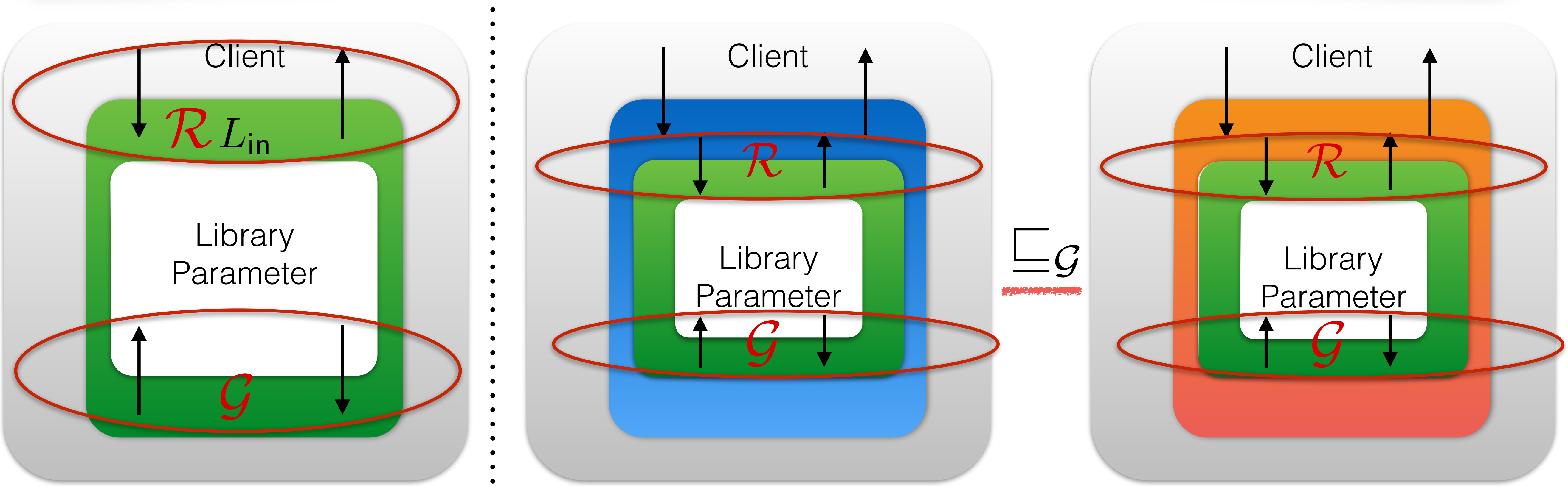
Contextuality of Up-to Linearisability

Theorem

$L_1, L_2 : M' \to M''$
$L_1 \sqsubseteq_{\mathcal{R}} L_2$
$L_{\mathsf{in}} : M \to M'$
L_{in} is $\binom{\mathcal{R}}{\mathcal{G}}$-closed

$$\Longrightarrow$$

$$M' \cap M'' = \emptyset \ \wedge$$
$$M \cap M' = \emptyset \ \Longrightarrow$$
$$(L_1 \circ L_{\mathsf{in}}) \sqsubseteq_{\mathcal{G}} (L_2 \circ L_{\mathsf{in}})$$



Remarks on Observational Refinement

Observational Refinement
=
May-Testing

General Linearisability
=
Histories Inclusion

Bonus Slide!

Remarks on Observational Refinement

Bonus Slide!

Observational Refinement

=

May-Testing

General Linearisability

=

Histories Inclusion

So What About Must Testing?

- Gotsman et al. 2012: Liveness Preserving Atomicity Abstraction
  - Observational Refinement for Liveness Properties (1st order)
  - Sound Proof Technique via a Variant of Linearisability
    Main Observables: calls/returns, divergence, deadlocks
- It remains to be seen whether the results scale to 2nd order