Linearizability, revisited

Radha Jagadeesan* Gustavo Petri† Corin Pitcher* James Riely*

*DePaul University †Purdue University

ESOP 2010, FOSSACS 2012, TLDI 2012, ESOP 2013

Reminiscing 1

1991 August — 1993 June

Reminiscing 2

Guru Brahma Guru Vishnu Guru Devo Maheshwaraha
Guru Saakshat Para Brahma Tasmai Sree Gurave Namaha

Guru is verily the representative of Brahma, Vishnu and Shiva.
He creates, sustains knowledge and destroys the weeds of ignorance.
I salute thee, Guru.

Reminiscing 2

Guru Brahma Guru Vishnu Guru Devo Maheshwaraha
Guru Saakshat Para Brahma Tasmai Sree Gurave Namaha

Guru is verily the representative of Brahma, Vishnu and Shiva.
He creates, sustains knowledge and destroys the weeds of ignorance.
I salute thee, Guru.

Reminiscing 2

Guru Brahma Guru Vishnu Guru Devo Maheshwaraha
Guru Saakshat Para Brahma Tasmai Sree Gurave Namaha

Guru is verily the representative of Brahma, Vishnu and Shiva.
He creates, sustains knowledge and destroys the weeds of ignorance.
I salute thee, Guru.

Linearizability, revisited

What is the interface of a concurrent object?

implicit causality via absence of interleavings . . .

What changes to account for weak memory?

. . . explicit causality via happens-before

Linearizability, revisited

What is the interface of a concurrent object?

implicit causality via absence of interleavings . . .

What changes to account for weak memory?

. . . explicit causality via happens-before

Refer to papers for appropriate and complete references

Spin lock (Library)

- Implementation

```
var v=1;
fun rel () { v=0; }
fun acq () { do skip until v.cas(0,1); }
```

(Initially locked)

(Strong memory)

- Trace of release

$\underbrace{\langle ?\mathsf{call\ rel}\rangle}_{\text{(Input)}}\ \underbrace{\langle \mathsf{wr\ v\ 0}\rangle}_{\text{(Write)}}\ \underbrace{\langle !\,\mathsf{ret\ rel}\rangle}_{\text{(Output)}}$

(Take control)

(Give control)

(Lock's viewpoint)

- Trace of acquire

$\langle ?\mathsf{call\ acq}\rangle\ \underbrace{\langle \mathsf{rd\ v\ 1}\rangle\langle \mathsf{rd\ v\ 1}\rangle\cdots\langle \mathsf{rd\ v\ 1}\rangle}_{\text{(Unsuccessful cas treated as read)}}\ \langle \mathsf{cas\ v\ 0\ 1}\rangle\langle !\,\mathsf{ret\ acq}\rangle$

Spin lock (Library)

- Implementation

```
var v=1;
fun rel () { v=0; }
fun acq () { do skip until v.cas(0,1); }
```

(Initially locked)

(Strong memory)

- Trace of release

$\underbrace{\langle ?\mathsf{call\ rel}\rangle}_{\text{(Input)}\ \text{(Take control)}}\ \underbrace{\langle \mathsf{wr\ v\ 0}\rangle}_{\text{(Write)}}\ \underbrace{\langle !\,\mathsf{ret\ rel}\rangle}_{\text{(Output)}\ \text{(Give control)}}$

(Lock's viewpoint)

- Trace of acquire

$\langle ?\mathsf{call\ acq}\rangle\ \underbrace{\langle \mathsf{rd\ v\ 1}\rangle\langle \mathsf{rd\ v\ 1}\rangle\cdots\langle \mathsf{rd\ v\ 1}\rangle}_{\text{(Unsuccessful cas treated as read)}}\ \langle \mathsf{cas\ v\ 0\ 1}\rangle\langle !\,\mathsf{ret\ acq}\rangle$

Spin lock (Library)

- Implementation

```
var v=1;
fun rel () { v=0; }
fun acq () { do skip until v.cas(0,1); }
```

(Initially locked)

(Strong memory)

- Trace of release

$\underbrace{\langle ?\text{call rel}\rangle}_{\text{(Input)}\ \text{(Take control)}}\ \underbrace{\langle \text{wr v 0}\rangle}_{\text{(Write)}}\ \underbrace{\langle !\,\text{ret rel}\rangle}_{\text{(Output)}\ \text{(Give control)}}$

(Lock's viewpoint)

- Trace of acquire

$\langle ?\text{call acq}\rangle\ \underbrace{\langle \text{rd v 1}\rangle\langle \text{rd v 1}\rangle\cdots\langle \text{rd v 1}\rangle}_{\text{(Unsuccessful cas treated as read)}}\ \langle \text{cas v 0 1}\rangle\langle !\,\text{ret acq}\rangle$

Spin lock (Two threads)

- Implementation

```
var v=1;
fun rel () { v=0; }
fun acq () { do skip until v.cas(0, 1); }
```

- Possible interleaving (Color = thread)

$\langle ?\text{call acq}\rangle\langle \text{rd v 1}\rangle\langle ?\text{call rel}\rangle\langle \text{wr v 0}\rangle\langle \text{cas v 0 1}\rangle\langle !\,\text{ret acq}\rangle\langle !\,\text{ret rel}\rangle$

- Impossible interleaving

$\langle ?\text{call acq}\rangle\langle \text{rd v 1}\rangle\langle \text{cas v 0 1}\rangle\langle !\,\text{ret acq}\rangle\langle ?\text{call rel}\rangle\langle \text{wr v 0}\rangle\langle !\,\text{ret rel}\rangle$

- Looking only at I/O actions:

$\langle ?\text{call rel}\rangle$ must precede $\langle !\,\text{ret acq}\rangle$

Spin lock (Two threads)

- Implementation

```
var v=1;
fun rel () { v=0; }
fun acq () { do skip until v.cas(0,1); }
```

- Possible interleaving (Color = thread)

$\langle ?\text{call acq}\rangle\langle \text{rd v 1}\rangle\langle ?\text{call rel}\rangle\langle \text{wr v 0}\rangle\langle \text{cas v 0 1}\rangle\langle !\,\text{ret acq}\rangle\langle !\,\text{ret rel}\rangle$

- Impossible interleaving

$\langle ?\text{call acq}\rangle\langle \text{rd v 1}\rangle\langle \text{cas v 0 1}\rangle\langle !\,\text{ret acq}\rangle\langle ?\text{call rel}\rangle\langle \text{wr v 0}\rangle\langle !\,\text{ret rel}\rangle$

- Looking only at I/O actions:

$\langle ?\text{call rel}\rangle$ must precede $\langle !\,\text{ret acq}\rangle$

Spin lock (Two threads)

- Implementation

```
var v=1;
fun rel () { v=0; }
fun acq () { do skip until v.cas(0, 1); }
```

- Possible interleaving (Color = thread)

$\langle ?\text{call acq}\rangle\langle \text{rd v 1}\rangle\langle ?\text{call rel}\rangle\langle \text{wr v 0}\rangle\langle \text{cas v 0 1}\rangle\langle !\,\text{ret acq}\rangle\langle !\,\text{ret rel}\rangle$

- Impossible interleaving

$\langle ?\text{call acq}\rangle\langle \text{rd v 1}\rangle\langle \text{cas v 0 1}\rangle\langle !\,\text{ret acq}\rangle\langle ?\text{call rel}\rangle\langle \text{wr v 0}\rangle\langle !\,\text{ret rel}\rangle$

- Looking only at I/O actions:

$\langle ?\text{call rel}\rangle$ must precede $\langle !\,\text{ret acq}\rangle$

Spin lock (Two threads)

- Implementation

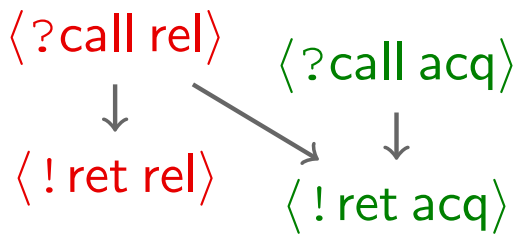
```
var v=1;
fun rel () { v=0; }
fun acq () { do skip until v.cas(0, 1); }
```

- Abbreviate

$$\left\{\begin{array}{l}
\langle ?\mathsf{call\ rel}\rangle\langle !\,\mathsf{ret\ rel}\rangle\langle ?\mathsf{call\ acq}\rangle\langle !\,\mathsf{ret\ acq}\rangle \\
\langle ?\mathsf{call\ rel}\rangle\langle ?\mathsf{call\ acq}\rangle\langle !\,\mathsf{ret\ rel}\rangle\langle !\,\mathsf{ret\ acq}\rangle \\
\langle ?\mathsf{call\ rel}\rangle\langle ?\mathsf{call\ acq}\rangle\langle !\,\mathsf{ret\ acq}\rangle\langle !\,\mathsf{ret\ rel}\rangle \\
\langle ?\mathsf{call\ acq}\rangle\langle ?\mathsf{call\ rel}\rangle\langle !\,\mathsf{ret\ rel}\rangle\langle !\,\mathsf{ret\ acq}\rangle \\
\langle ?\mathsf{call\ acq}\rangle\langle ?\mathsf{call\ rel}\rangle\langle !\,\mathsf{ret\ acq}\rangle\langle !\,\mathsf{ret\ rel}\rangle
\end{array}\right\}$$

$\langle ?\mathsf{call\ acq}\rangle\langle !\,\mathsf{ret\ acq}\rangle\langle ?\mathsf{call\ rel}\rangle\langle !\,\mathsf{ret\ rel}\rangle$

as



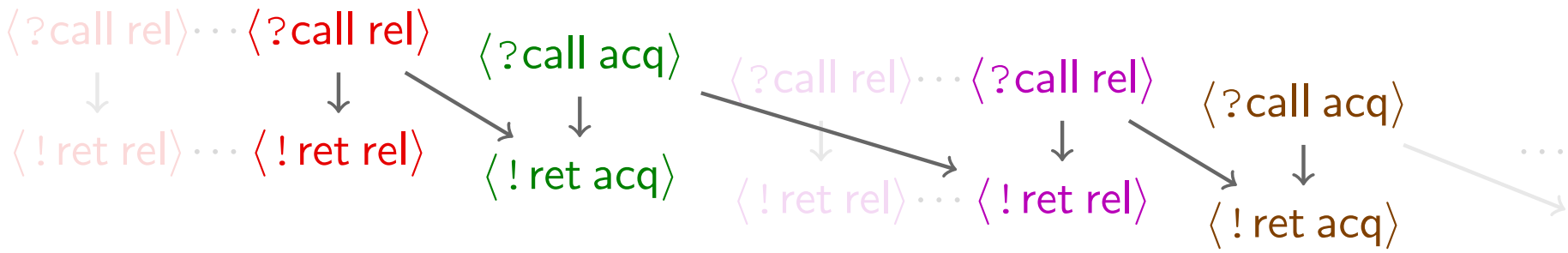
$\longrightarrow$ constrains interleavings

Spin lock (Two threads)

- Implementation

```
var v=1;
fun rel () { v=0; }
fun acq () { do skip until v.cas(0,1); }
```

- Interface: set of traces obeying



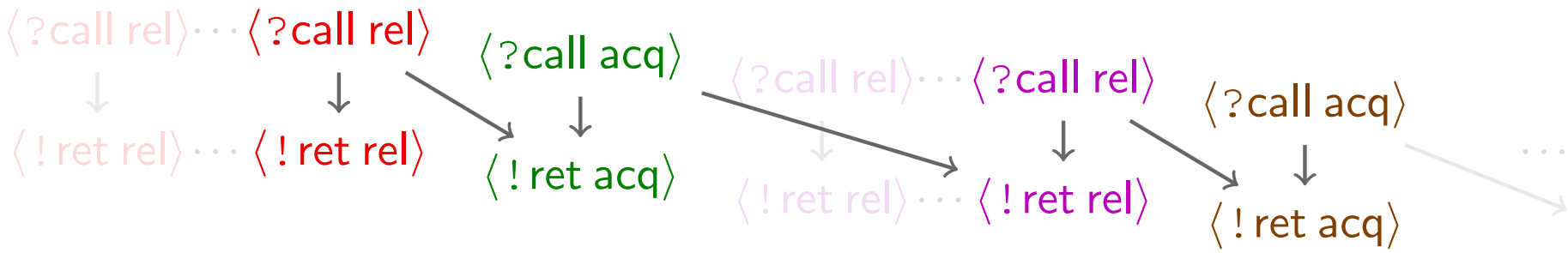
- $\longrightarrow$ constrains interleavings
  $\longrightarrow$ imposed by Lock
- What about client?

Spin lock (Two threads)

- Implementation

```
var v=1;
fun rel () { v=0; }
fun acq () { do skip until v.cas(0,1); }
```

- Interface: set of traces obeying



- $\longrightarrow$ constrains interleavings
  $\longrightarrow$ imposed by Lock
- What about client?

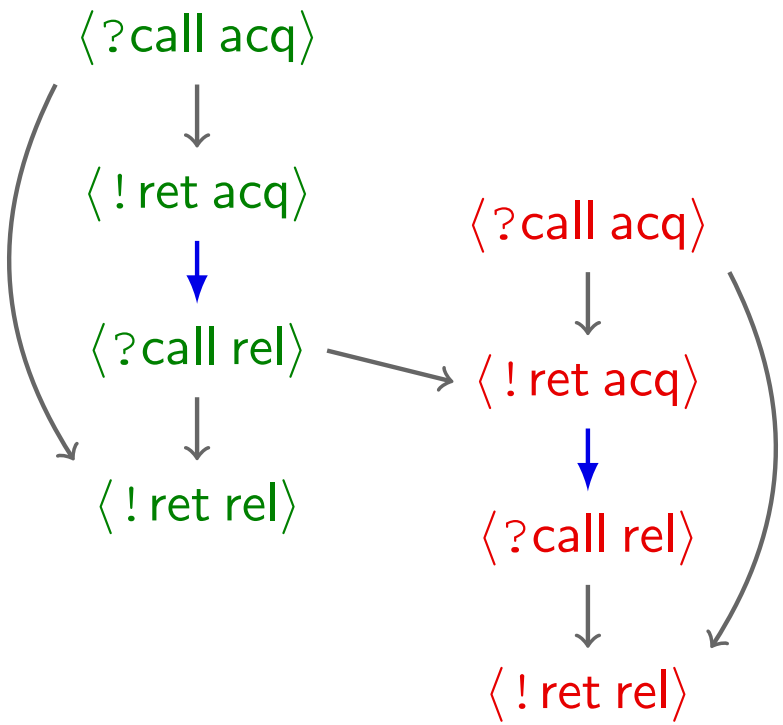
Spin lock (Client constraints)

- Implementation

```
var v=0;
fun rel () { v=0; }
fun acq () { do skip until v.cas(0,1); }
```

(Initially unlocked)

- Example of client order
  Multiple calls from single thread



Two kinds of constraints

? ⟶ ! Imposed by library (In → out)

! ⟶ ? Imposed by client (Out → in)

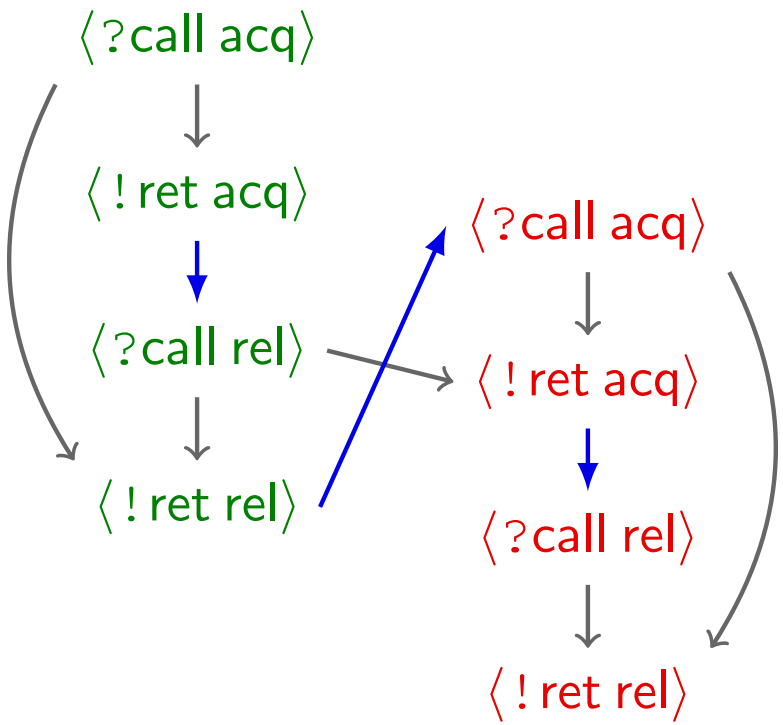
Spin lock (Client constraints)

- Implementation

```
var v=0;
fun rel () { v=0; }
fun acq () { do skip until v.cas(0,1); }
```

(Initially unlocked)

- Example of client order
  Multiple calls from single thread



Arrows constrain interleavings

More arrows = smaller set

Fully constrained = singleton, sequential

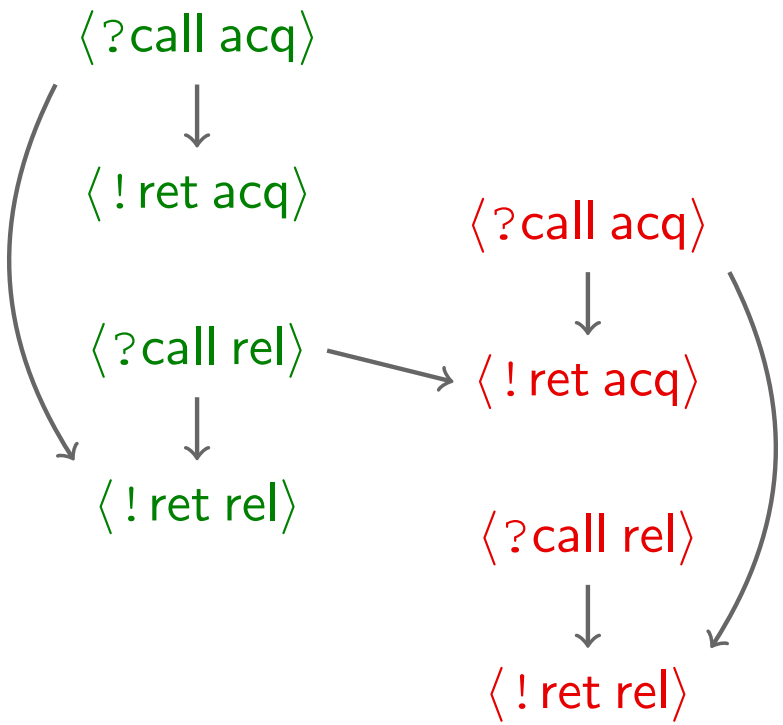
Spin lock (Client constraints)

- Implementation

```
var v=0;
fun rel () { v=0; }
fun acq () { do skip until v.cas(0,1); }
```

(Initially unlocked)

- Example of client order
  Multiple calls from single thread



Arrows constrain interleavings

More arrows = smaller set

No constraints = all allowed interleavings

Two locks (for one place buffer)

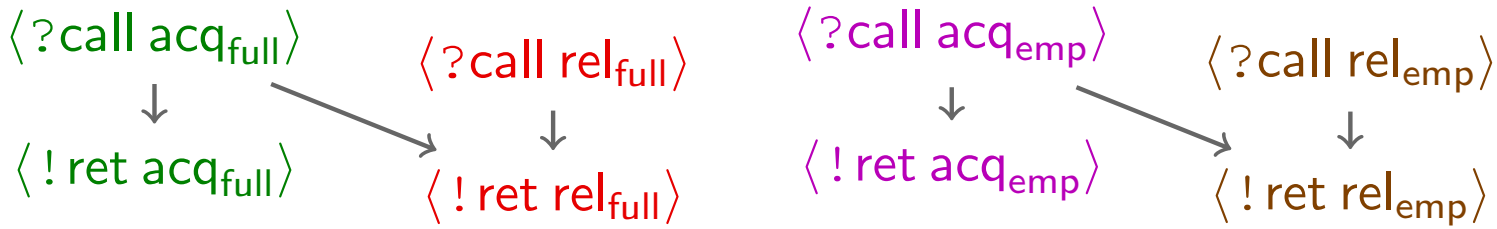
(Initially locked)

```
var v_full=1;
fun rel_full () { v_full=0; }
fun acq_full () { ··· v_full.cas(0,1); }
```

(Initially unlocked)

```
var v_emp=0;
fun rel_emp () { v_emp=0; }
fun acq_emp () { ··· v_emp.cas(0,1); }
```

- Locks are independent



- Client may create dependency

Two locks (for one place buffer)

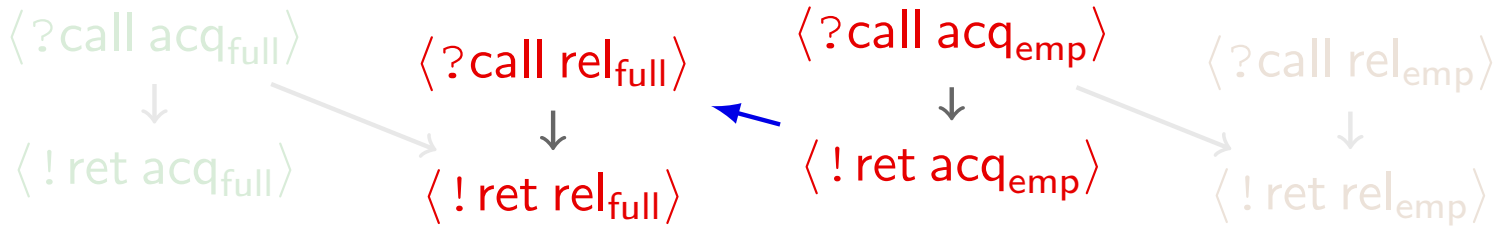
(Initially locked)

```
var v_full=1;
fun rel_full () { v_full=0; }
fun acq_full () { ··· v_full.cas(0,1); }
```

(Initially unlocked)

```
var v_emp=0;
fun rel_emp () { v_emp=0; }
fun acq_emp () { ··· v_emp.cas(0,1); }
```

- Locks are independent



- Client may create dependency

One place buffer (Initially empty)

```
var x=0;
fun put (r)  { acq_emp (); x=r; rel_full (); }
fun get ()   { acq_full (); let r=x; rel_emp (); return r; }
```

(r = register)

(emp unlocked)

(full locked)

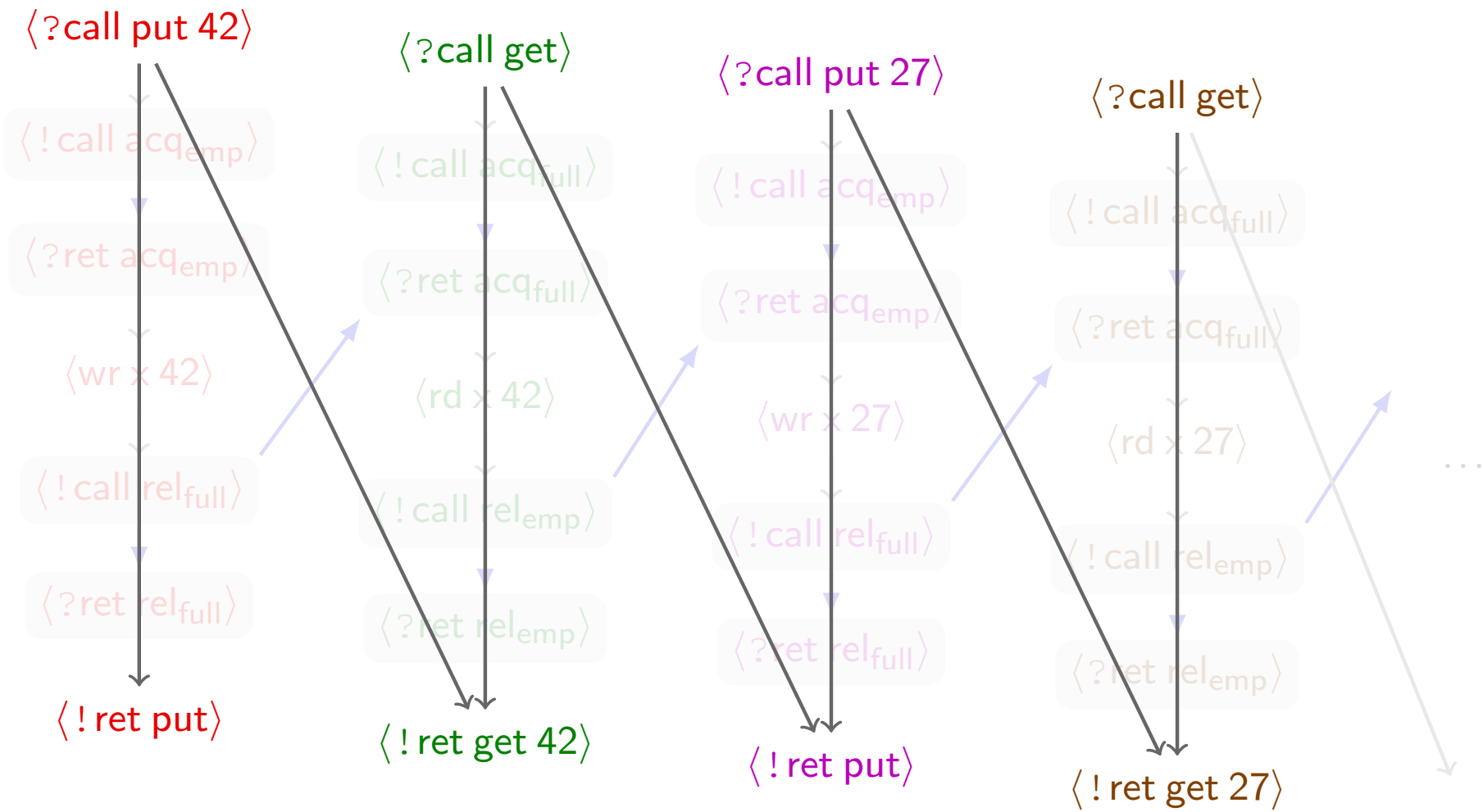
One place buffer (Initially empty)

```
var x=0;
fun put (r)  { acq_emp (); x=r; rel_full (); }
fun get ()   { acq_full (); let r=x; rel_emp (); return r; }
```

(r = register)

(emp unlocked)

(full locked)



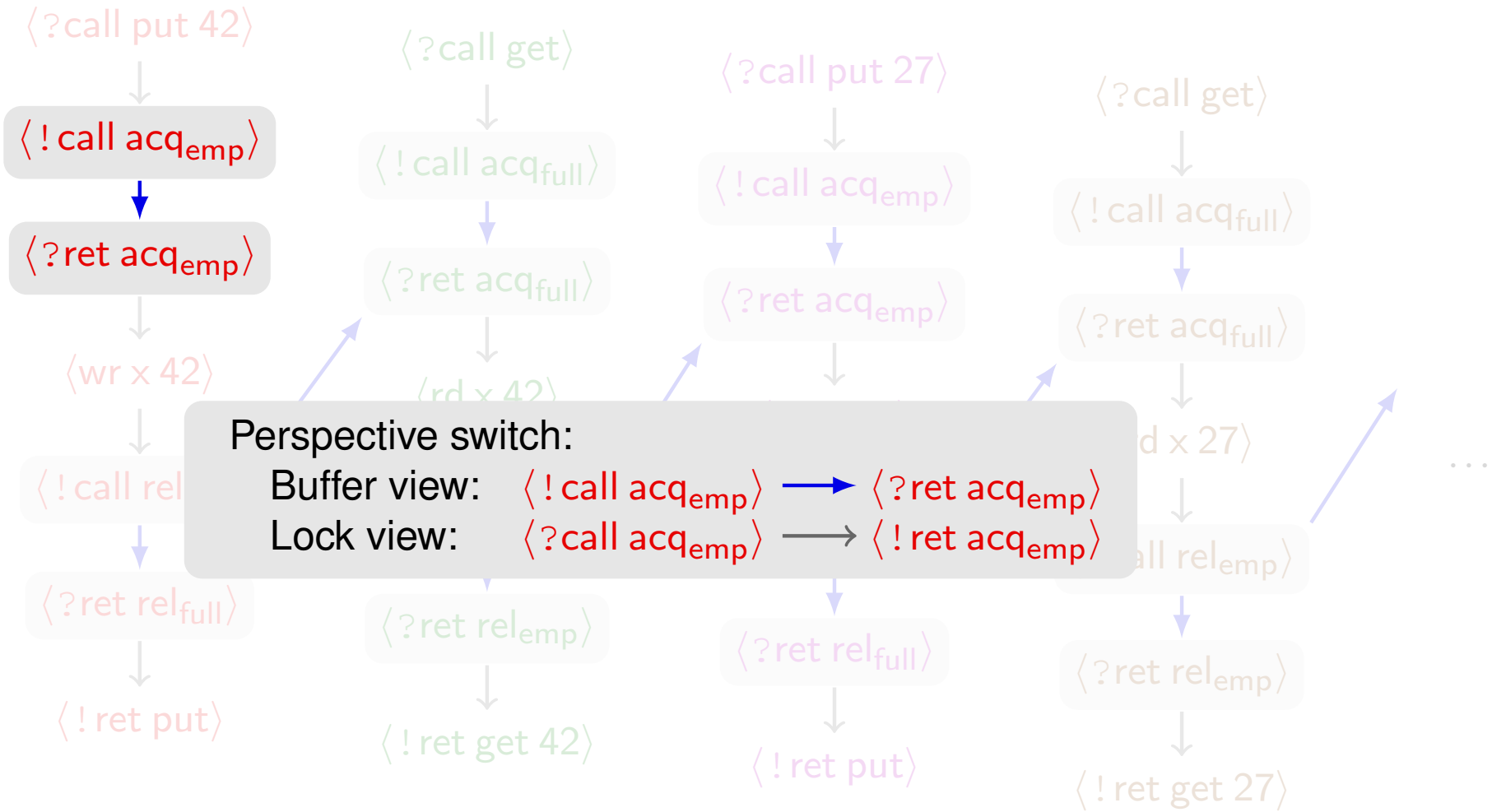
One place buffer

```
var x=0;
fun put (r)  { acq_emp (); x=r; rel_full (); }
fun get ()   { acq_full (); let r=x; rel_emp (); return r; }
```

(r = register)

(emp unlocked)

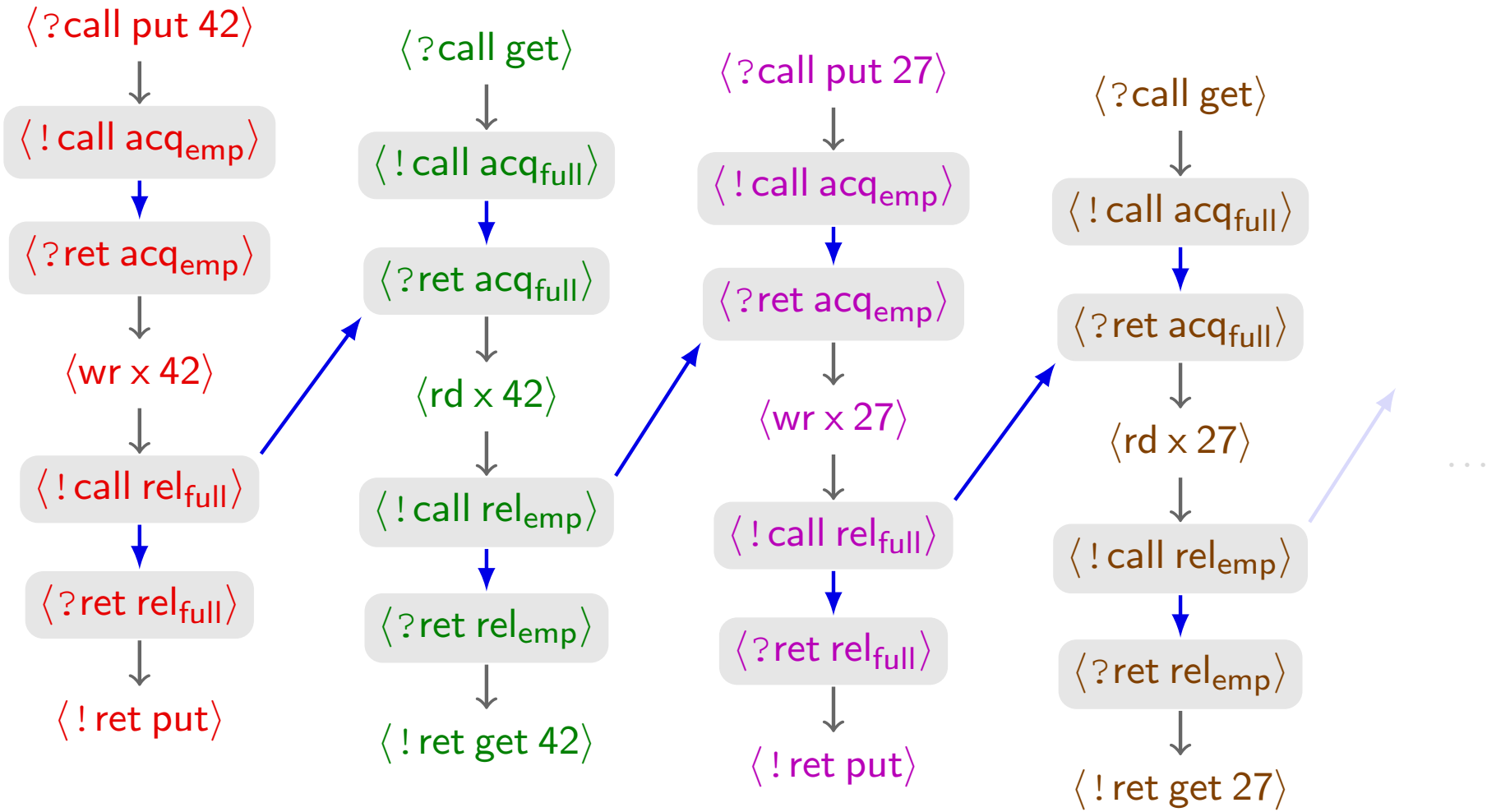
(full locked)



One place buffer (Client interface)

```
var x=0;
fun put (r)  { acq_emp (); x=r; rel_full (); }
fun get ()   { acq_full (); let r=x; rel_emp (); return r; }
```

(r = register)
(emp unlocked)
(full locked)



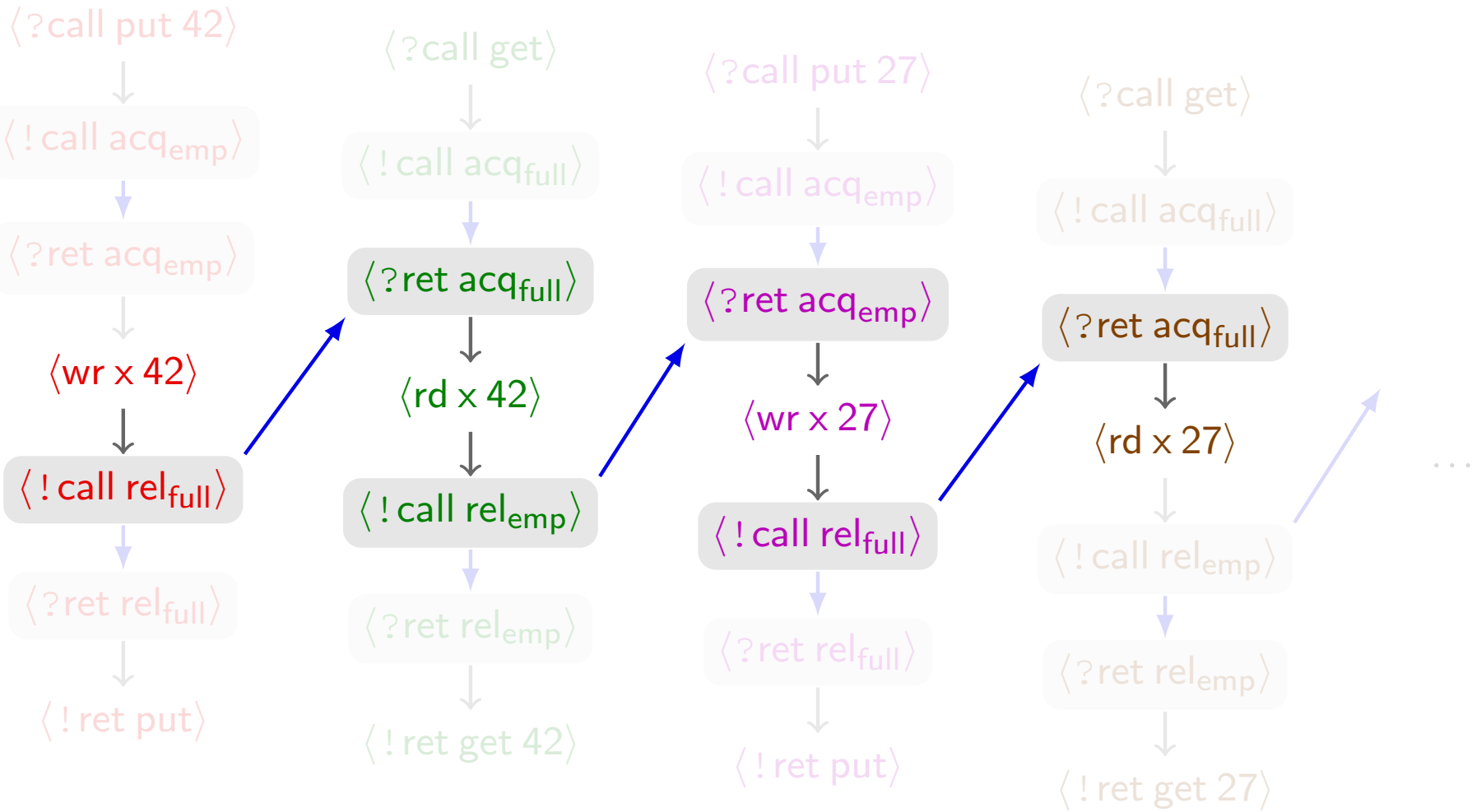
One place buffer (Memory actions)

```
var x=0;
fun put (r)  { acq_emp (); x=r; rel_full (); }
fun get ()   { acq_full (); let r=x; rel_emp (); return r; }
```

(r = register)

(emp unlocked)

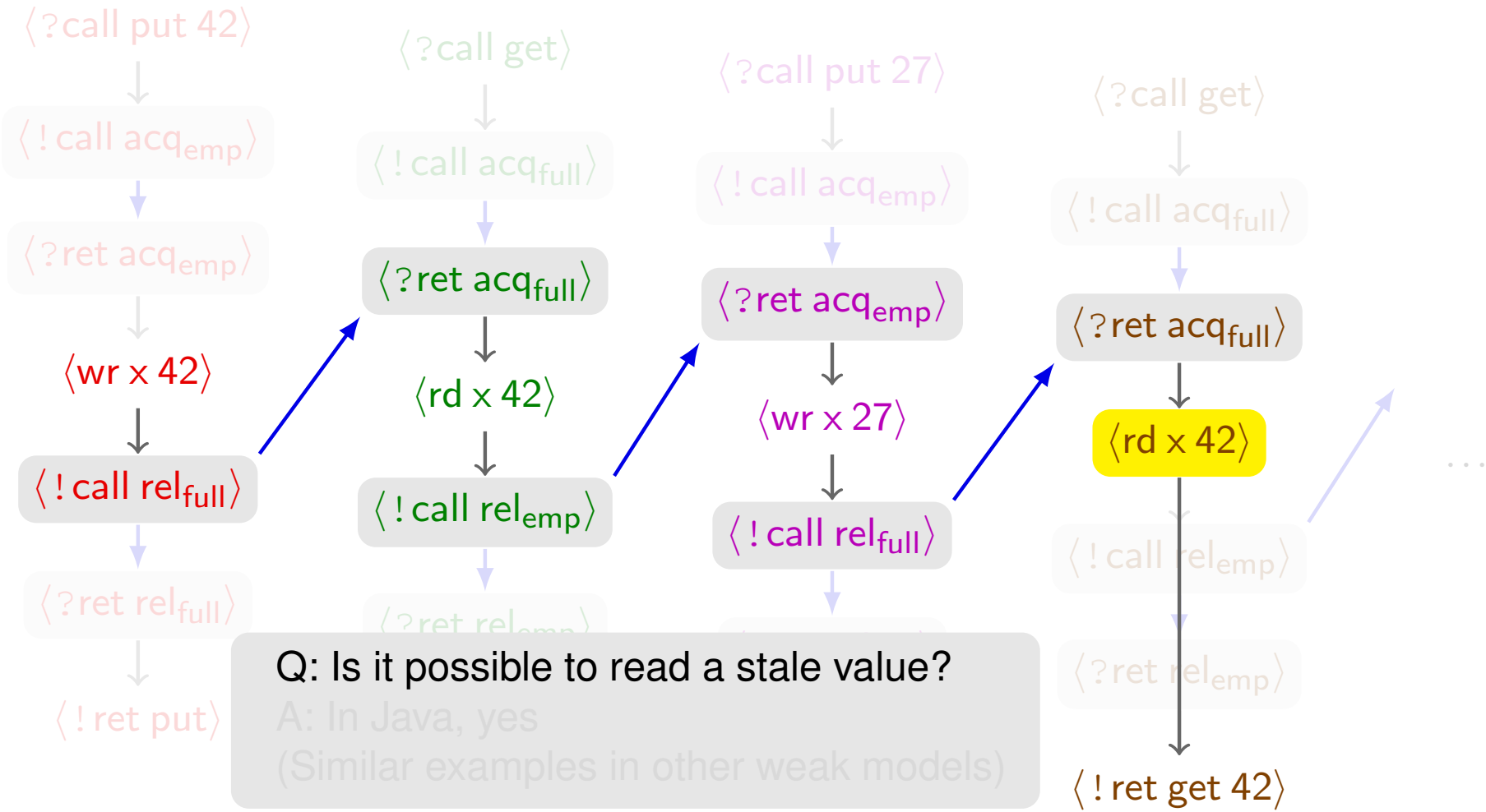
(full locked)



One place buffer (Relaxed memory)

```
var x=0;
fun put (r)  { acq_emp (); x=r; rel_full (); }
fun get ()   { acq_full (); let r=x; rel_emp (); return r; }
```

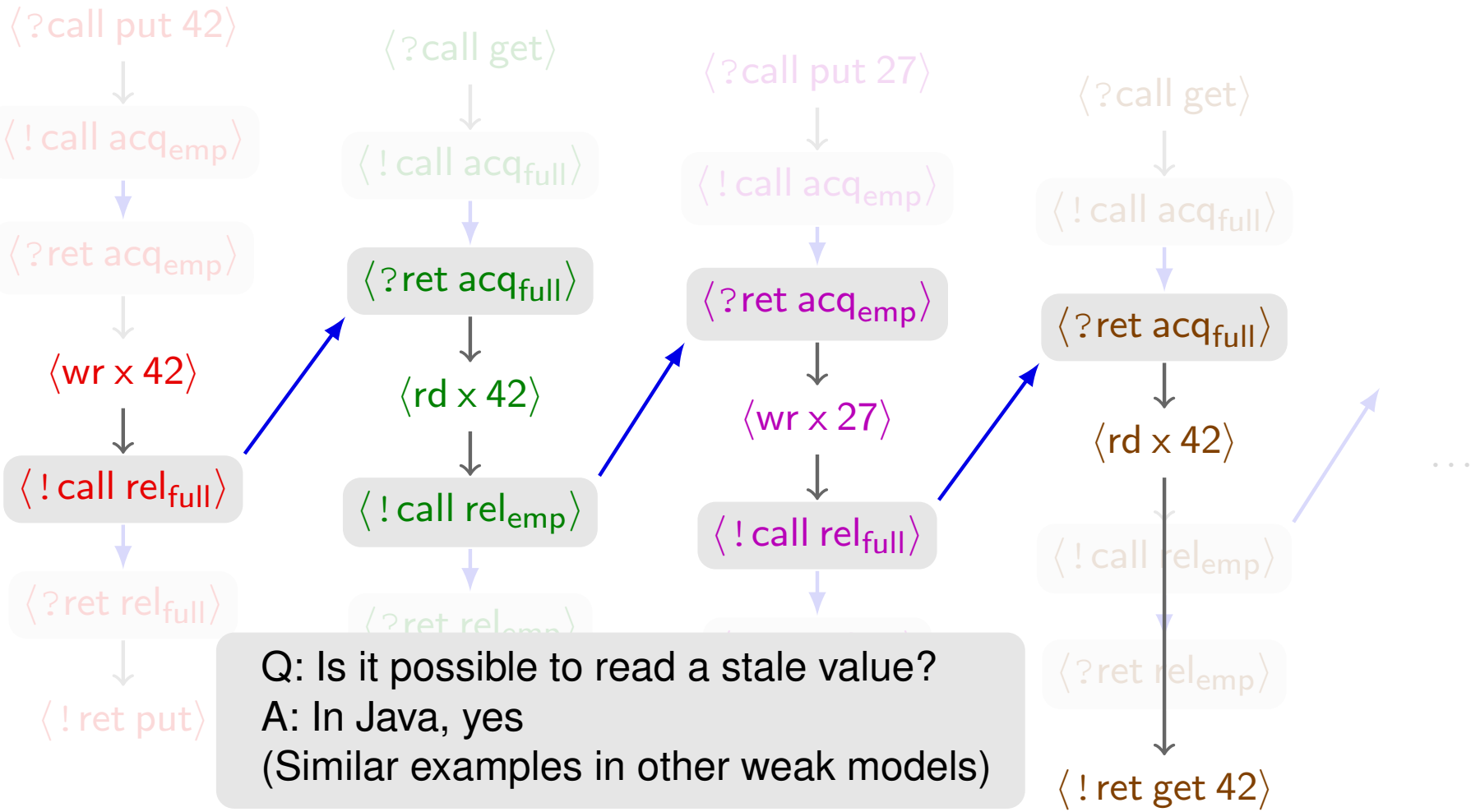
(r = register)
(emp unlocked)
(full locked)



One place buffer (Relaxed memory)

```
var x=0;
fun put (r)  { acq_emp (); x=r; rel_full (); }
fun get ()   { acq_full (); let r=x; rel_emp (); return r; }
```

(r = register)
(emp unlocked)
(full locked)

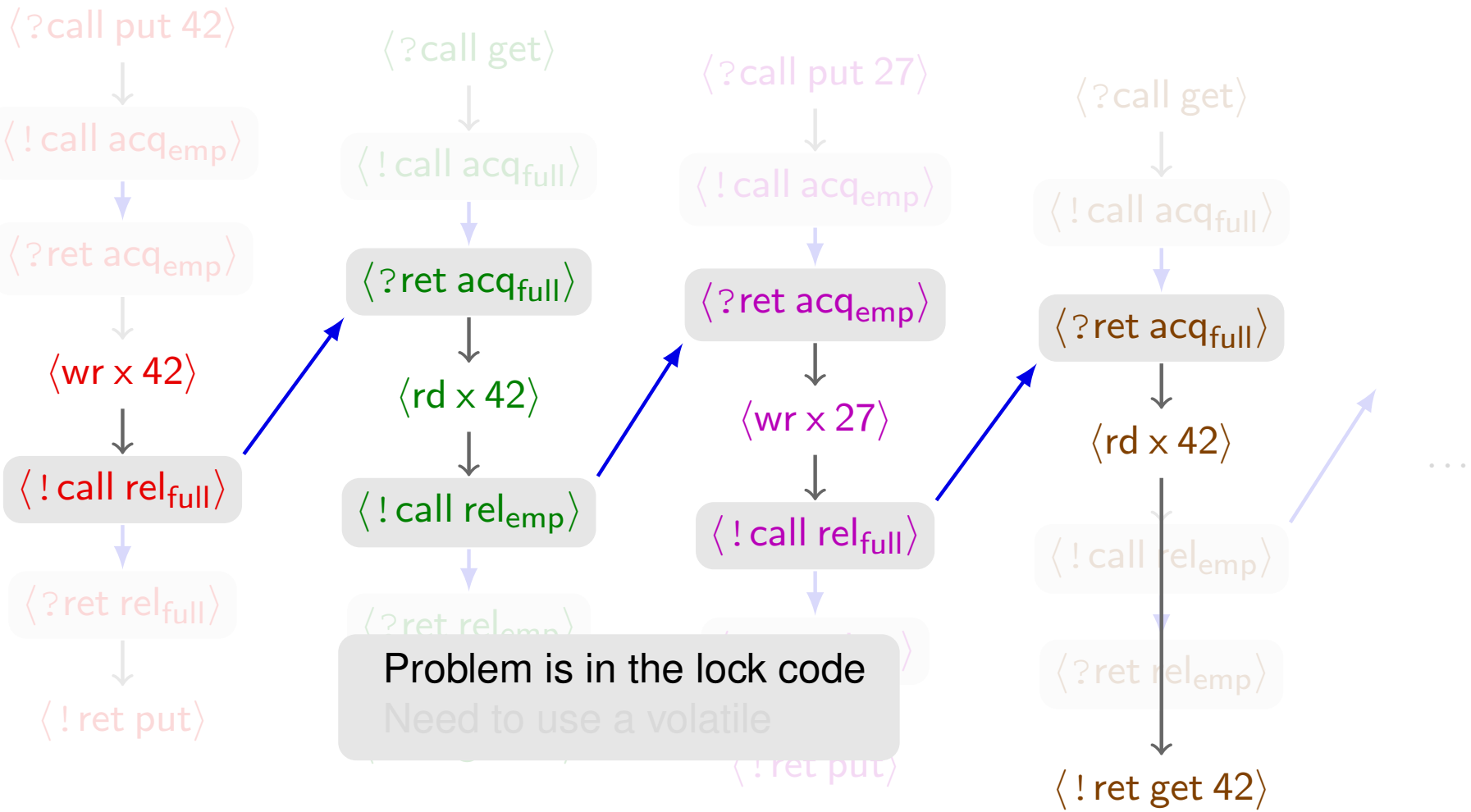


Q: Is it possible to read a stale value?
A: In Java, yes
(Similar examples in other weak models)

One place buffer (Relaxed memory)

```
var v=1;
fun rel () { v=0; }
fun acq () { do skip until v.cas(0,1); }
```

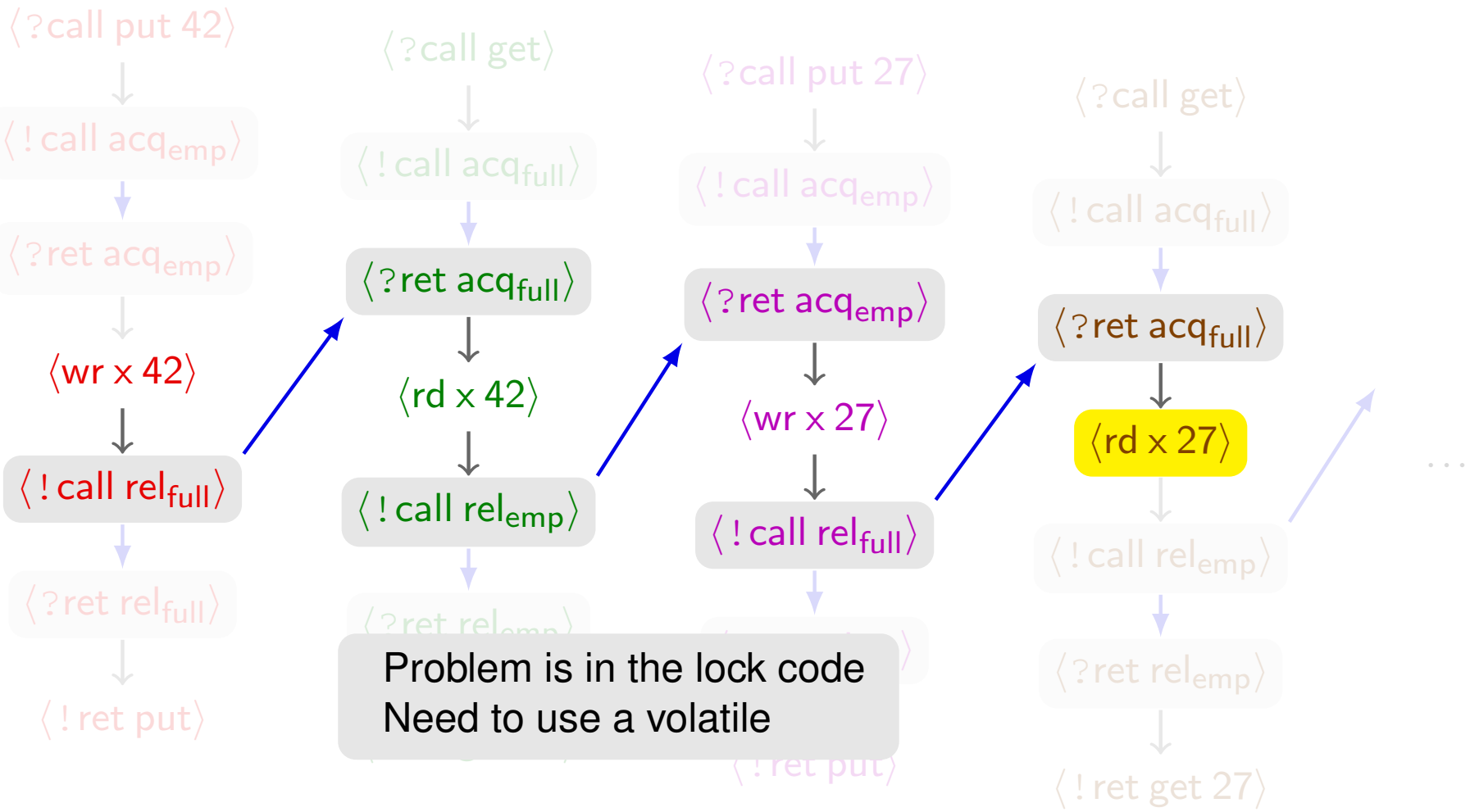
(Lock code)



One place buffer (Relaxed memory)

```
volatile v=1;
fun rel () { v=0; }
fun acq () { do skip until v.cas(0,1); }
```

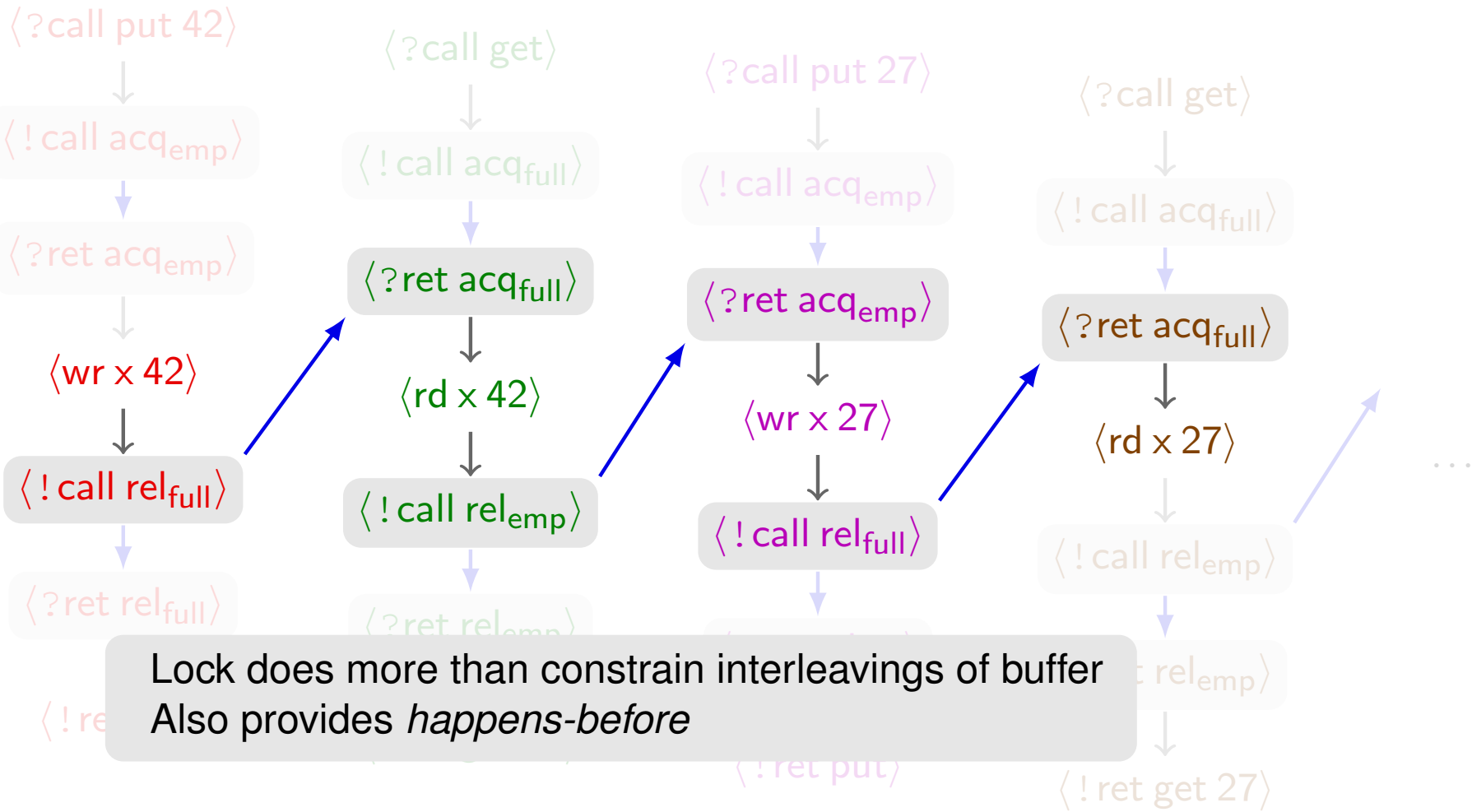
(Lock code)



One place buffer (Relaxed memory)

```
var x=0;
fun put (r)  { acq_emp (); x=r; rel_full (); }
fun get ()   { acq_full (); let r=x; rel_emp (); return r; }
```

(Buffer code)

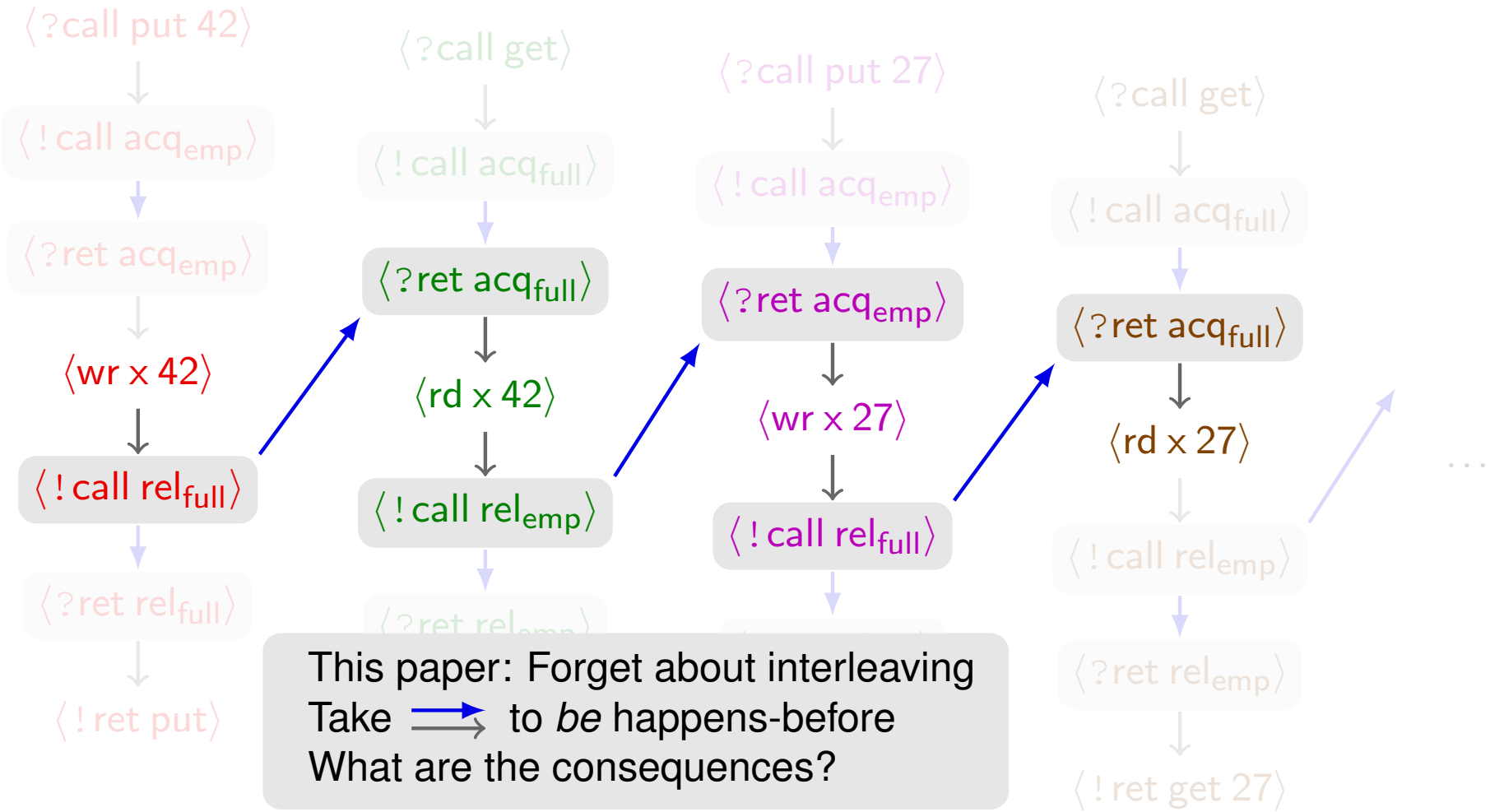


Lock does more than constrain interleavings of buffer
Also provides *happens-before*

One place buffer (Relaxed memory)

```
var x=0;
fun put (r)  { acq_emp (); x=r; rel_full (); }
fun get ()   { acq_full (); let r=x; rel_emp (); return r; }
```

(Buffer code)



This paper: Forget about interleaving
Take $\Longrightarrow$ to *be* happens-before
What are the consequences?

Plan

Traditional notions of correctness

Happens-before

Results

Compositionality

Notions of correctness

- Sequential consistency (SC) = methods appear atomic
  (Lamport, IEEE Trans. Comput. 1979)

$$(\forall \sigma \in \mathit{Impl})\ (\exists \phi \in \mathit{SequentialSpec})\ (\forall s \in \mathit{Thread})\ \sigma|_s = \phi|_s$$

- Linearizability = Serializability + compositionality
  (Herlihy/Wing, POPL 1987, TOPLAS 1990)

  $\cdots$ and ϕ must respect order of nonoverlapping calls in σ

- Example ($\mathit{Impl} \sqsubseteq \mathit{Spec}$)

That is,

$$\{\langle ?\text{call } f\rangle\langle !\text{ ret } f\rangle\langle ?\text{call } g\rangle\langle !\text{ ret } g\rangle\} \quad \sqsubseteq \quad \{\langle ?\text{call } g\rangle\langle !\text{ ret } g\rangle\langle ?\text{call } f\rangle\langle !\text{ ret } f\rangle\}$$

Notions of correctness

- Serializability = methods appear atomic
  (Lamport, IEEE Trans. Comput. 1979)

$$(\forall \sigma \in \mathit{Impl})\ (\exists \phi \in \mathit{SequentialSpec})\ (\forall s \in \mathit{Thread})\ \sigma|_s = \phi|_s$$

- Linearizability = Serializability + compositionality
  (Herlihy/Wing, POPL 1987, TOPLAS 1990)

$\cdots$ and ϕ must respect order of nonoverlapping calls in σ

- Example ($\mathit{Impl} \sqsubseteq \mathit{Spec}$)

That is,

$$\{\langle ?\text{call } f\rangle\langle !\text{ ret } f\rangle\langle ?\text{call } g\rangle\langle !\text{ ret } g\rangle\} \quad \sqsubseteq \quad \{\langle ?\text{call } g\rangle\langle !\text{ ret } g\rangle\langle ?\text{call } f\rangle\langle !\text{ ret } f\rangle\}$$

Notions of correctness

- Serializability = methods appear atomic
  (Lamport, IEEE Trans. Comput. 1979)

$$(\forall \sigma \in \mathit{Impl})\ (\exists \phi \in \mathit{SequentialSpec})\ (\forall s \in \mathit{Thread})\ \sigma|_s = \phi|_s$$

- Linearizability = Serializability + compositionality
  (Herlihy/Wing, POPL 1987, TOPLAS 1990)

$\cdots$ and ϕ must respect order of nonoverlapping calls in σ

- Example ($\mathit{Impl} \sqsubseteq \mathit{Spec}$)

That is,

$$\{\langle ?\text{call } f\rangle\langle !\text{ ret } f\rangle\langle ?\text{call } g\rangle\langle !\text{ ret } g\rangle\} \quad \sqsubseteq \quad \{\langle ?\text{call } g\rangle\langle !\text{ ret } g\rangle\langle ?\text{call } f\rangle\langle !\text{ ret } f\rangle\}$$

Notions of correctness

- Serializability = methods appear atomic
  (Lamport, IEEE Trans. Comput. 1979)

$$(\forall \sigma \in \mathit{Impl})\ (\exists \phi \in \mathit{SequentialSpec})\ (\forall s \in \mathit{Thread})\ \sigma|_s = \phi|_s$$

- Linearizability = Serializability + compositionality
  (Herlihy/Wing, POPL 1987, TOPLAS 1990)

  $\cdots$ and ϕ must respect order of nonoverlapping calls in σ

- Example ($\mathit{Impl} \sqsubseteq \mathit{Spec}$)

That is,

$$\{\langle ?\text{call } f\rangle\langle !\text{ ret } f\rangle\langle ?\text{call } g\rangle\langle !\text{ ret } g\rangle\} \quad \sqsubseteq \quad \{\langle ?\text{call } g\rangle\langle !\text{ ret } g\rangle\langle ?\text{call } f\rangle\langle !\text{ ret } f\rangle\}$$

Happens-before

- Semantics as sets Σ, Φ of traces σ, ϕ with named actions
  - Four memory models: $\mathcal{W} \in \{\mathsf{strong}, \mathsf{tso}, \mathsf{pso}, \mathsf{jmm}\}$
  - Order recovered by relation $i <^{\sigma}_{\mathcal{W}} j$
  - Informally $(<^{\sigma}_{\mathcal{W}}) = (\Longrightarrow)$ (only one relation, color distinguishes polarity)
- In a specification:

$$\langle ?\mathsf{call}\ f \rangle \longrightarrow \langle !\,\mathsf{ret}\ g \rangle \text{ if } \langle ?\mathsf{call}\ f\ a \rangle \cdots \langle !\,\mathsf{ret}\ g\ \{a\} \rangle$$
$$\langle !\,\mathsf{ret}\ f \rangle \longrightarrow \langle ?\mathsf{call}\ g \rangle \text{ if } \langle !\,\mathsf{ret}\ f\ b \rangle \cdots \langle ?\mathsf{call}\ g\ \{b\} \rangle$$

- In opsem (thread s, actions a, b, volatile v):

$$\langle s\ \ a \rangle \longrightarrow \langle s\ \ b \rangle \text{ if } \langle s\ \ a \rangle \cdots \langle s\ \ b \rangle \quad \text{(thread order)}$$
$$\langle \mathsf{wr}\ v \rangle \longrightarrow \langle \mathsf{rd}\ v \rangle \text{ if } \langle \mathsf{wr}\ v \rangle \cdots \langle \mathsf{rd}\ v \rangle \quad \text{(synchronization)}$$
$$\langle \mathsf{wr}\ v \rangle \longrightarrow \langle \mathsf{cas}\ v \rangle \text{ if } \langle \mathsf{wr}\ v \rangle \cdots \langle \mathsf{cas}\ v \rangle \quad \text{(synchronization)}$$
$$\langle \mathsf{cas}\ v \rangle \longrightarrow \langle \mathsf{rd}\ v \rangle \text{ if } \langle \mathsf{cas}\ v \rangle \cdots \langle \mathsf{rd}\ v \rangle \quad \text{(synchronization)}$$
$$\langle \mathsf{cas}\ v \rangle \longrightarrow \langle \mathsf{cas}\ v \rangle \text{ if } \langle \mathsf{cas}\ v \rangle \cdots \langle \mathsf{cas}\ v \rangle \quad \text{(synchronization)}$$

- This defines $<^{\sigma}_{\mathsf{jmm}}$
  - $<^{\sigma}_{\mathsf{strong}}$ includes conflicts on all variables
  - $<^{\sigma}_{\mathsf{tso}}$ and $<^{\sigma}_{\mathsf{pso}}$ in between

Happens-before

- Semantics as sets Σ, Φ of traces σ, ϕ with named actions
  - Four memory models: $\mathcal{W} \in \{\text{strong}, \text{tso}, \text{pso}, \text{jmm}\}$
  - Order recovered by relation $i <^{\sigma}_{\mathcal{W}} j$
  - Informally $(<^{\sigma}_{\mathcal{W}}) = (\Longrightarrow)$ (only one relation, color distinguishes polarity)
- In a specification:

$$\langle ?\text{call } f \rangle \longrightarrow \langle !\,\text{ret } g \rangle \text{ if } \langle ?\text{call } f\ a \rangle \cdots \langle !\,\text{ret } g\ \{a\} \rangle$$
$$\langle !\,\text{ret } f \rangle \longrightarrow \langle ?\text{call } g \rangle \text{ if } \langle !\,\text{ret } f\ b \rangle \cdots \langle ?\text{call } g\ \{b\} \rangle$$

- In opsem (thread s, actions a, b, volatile v):

$$\langle s\ \ a \rangle \longrightarrow \langle s\ \ b \rangle \text{ if } \langle s\ \ a \rangle \cdots \langle s\ \ b \rangle \quad \text{(thread order)}$$
$$\langle \text{wr } v \rangle \longrightarrow \langle \text{rd } v \rangle \text{ if } \langle \text{wr } v \rangle \cdots \langle \text{rd } v \rangle \quad \text{(synchronization)}$$
$$\langle \text{wr } v \rangle \longrightarrow \langle \text{cas } v \rangle \text{ if } \langle \text{wr } v \rangle \cdots \langle \text{cas } v \rangle \quad \text{(synchronization)}$$
$$\langle \text{cas } v \rangle \longrightarrow \langle \text{rd } v \rangle \text{ if } \langle \text{cas } v \rangle \cdots \langle \text{rd } v \rangle \quad \text{(synchronization)}$$
$$\langle \text{cas } v \rangle \longrightarrow \langle \text{cas } v \rangle \text{ if } \langle \text{cas } v \rangle \cdots \langle \text{cas } v \rangle \quad \text{(synchronization)}$$

- This defines $<^{\sigma}_{\text{jmm}}$
  - $<^{\sigma}_{\text{strong}}$ includes conflicts on all variables
  - $<^{\sigma}_{\text{tso}}$ and $<^{\sigma}_{\text{pso}}$ in between

Happens-before

- Semantics as sets Σ, Φ of traces σ, ϕ with named actions
  - Four memory models: $\mathcal{W} \in \{\mathsf{strong}, \mathsf{tso}, \mathsf{pso}, \mathsf{jmm}\}$
  - Order recovered by relation $i <^{\sigma}_{\mathcal{W}} j$
  - Informally $(<^{\sigma}_{\mathcal{W}}) = (\Longrightarrow)$ (only one relation, color distinguishes polarity)
- In a specification:

$$\langle ?\mathsf{call}\ f\rangle \longrightarrow \langle !\,\mathsf{ret}\ g\rangle \text{ if } \langle ?\mathsf{call}\ f\ a\rangle \cdots \langle !\,\mathsf{ret}\ g\ \{a\}\rangle$$
$$\langle !\,\mathsf{ret}\ f\rangle \longrightarrow \langle ?\mathsf{call}\ g\rangle \text{ if } \langle !\,\mathsf{ret}\ f\ b\rangle \cdots \langle ?\mathsf{call}\ g\ \{b\}\rangle$$

- In opsem (thread s, actions a, b, volatile v):

$$\langle s\ \ a\rangle \longrightarrow \langle s\ \ b\rangle \text{ if } \langle s\ \ a\rangle \cdots \langle s\ \ b\rangle \quad \text{(thread order)}$$
$$\langle \mathsf{wr}\ v\rangle \longrightarrow \langle \mathsf{rd}\ v\rangle \text{ if } \langle \mathsf{wr}\ v\rangle \cdots \langle \mathsf{rd}\ v\rangle \quad \text{(synchronization)}$$
$$\langle \mathsf{wr}\ v\rangle \longrightarrow \langle \mathsf{cas}\ v\rangle \text{ if } \langle \mathsf{wr}\ v\rangle \cdots \langle \mathsf{cas}\ v\rangle \quad \text{(synchronization)}$$
$$\langle \mathsf{cas}\ v\rangle \longrightarrow \langle \mathsf{rd}\ v\rangle \text{ if } \langle \mathsf{cas}\ v\rangle \cdots \langle \mathsf{rd}\ v\rangle \quad \text{(synchronization)}$$
$$\langle \mathsf{cas}\ v\rangle \longrightarrow \langle \mathsf{cas}\ v\rangle \text{ if } \langle \mathsf{cas}\ v\rangle \cdots \langle \mathsf{cas}\ v\rangle \quad \text{(synchronization)}$$

- This defines $<^{\sigma}_{\mathsf{jmm}}$
  - $<^{\sigma}_{\mathsf{strong}}$ includes conflicts on all variables
  - $<^{\sigma}_{\mathsf{tso}}$ and $<^{\sigma}_{\mathsf{pso}}$ in between

Happens-before

- Semantics as sets Σ, Φ of traces σ, ϕ with named actions
  - Four memory models: $\mathscr{W} \in \{\mathsf{strong}, \mathsf{tso}, \mathsf{pso}, \mathsf{jmm}\}$
  - Order recovered by relation $i <^{\sigma}_{\mathscr{W}} j$
  - Informally $(<^{\sigma}_{\mathscr{W}}) = (\longrightarrow)$ (only one relation, color distinguishes polarity)
- In a specification:

$$\langle ?\mathsf{call}\ f\rangle \longrightarrow \langle !\,\mathsf{ret}\ g\rangle \text{ if } \langle ?\mathsf{call}\ f\ a\rangle \cdots \langle !\,\mathsf{ret}\ g\ \{a\}\rangle$$
$$\langle !\,\mathsf{ret}\ f\rangle \longrightarrow \langle ?\mathsf{call}\ g\rangle \text{ if } \langle !\,\mathsf{ret}\ f\ b\rangle \cdots \langle ?\mathsf{call}\ g\ \{b\}\rangle$$

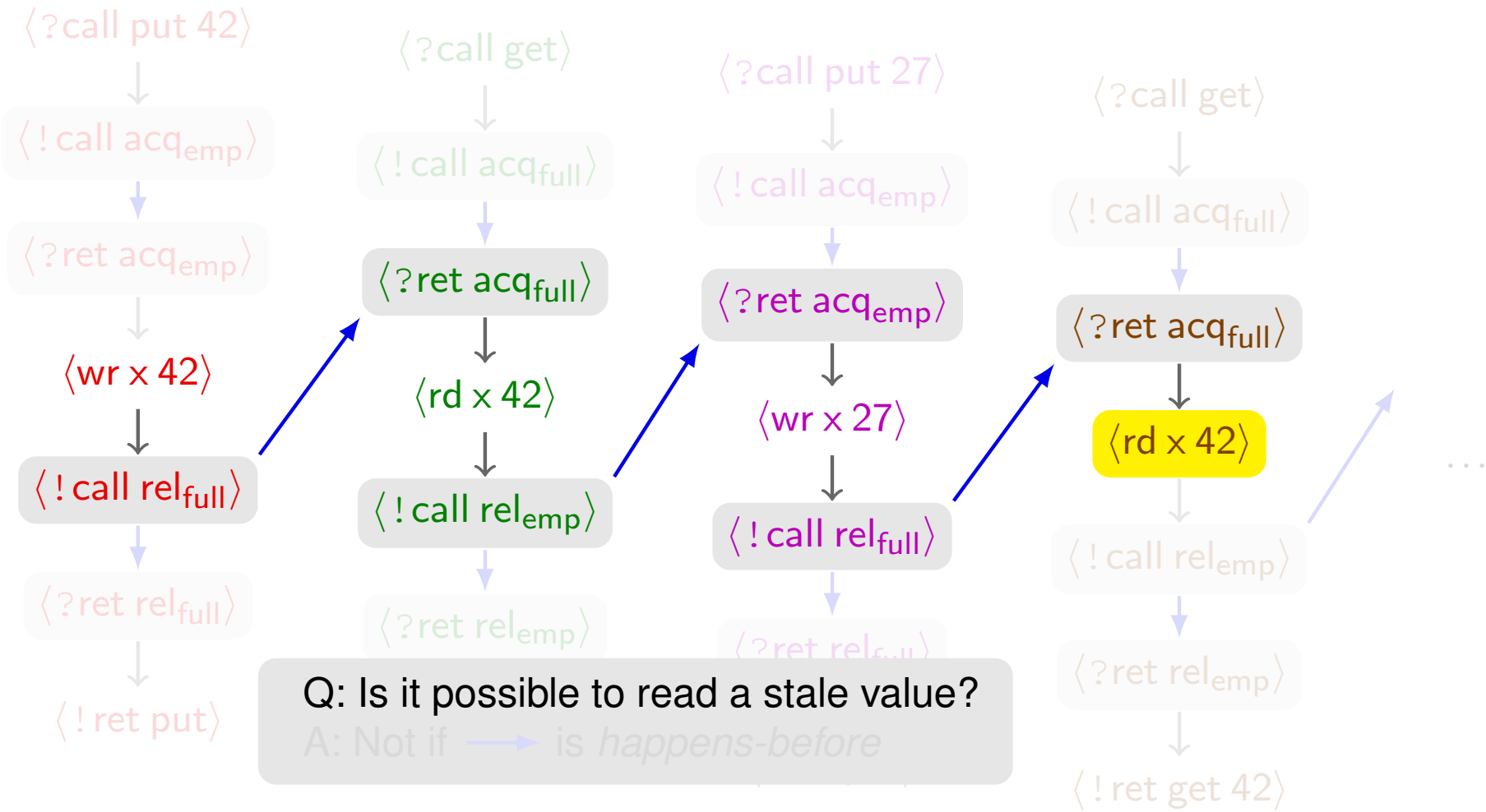
- In opsem (thread s, actions a, b, volatile v):

$$\langle s\ \ a\rangle \longrightarrow \langle s\ \ b\rangle \text{ if } \langle s\ \ a\rangle \cdots \langle s\ \ b\rangle \quad \text{(thread order)}$$
$$\langle \mathsf{wr}\ v\rangle \longrightarrow \langle \mathsf{rd}\ v\rangle \text{ if } \langle \mathsf{wr}\ v\rangle \cdots \langle \mathsf{rd}\ v\rangle \quad \text{(synchronization)}$$
$$\langle \mathsf{wr}\ v\rangle \longrightarrow \langle \mathsf{cas}\ v\rangle \text{ if } \langle \mathsf{wr}\ v\rangle \cdots \langle \mathsf{cas}\ v\rangle \quad \text{(synchronization)}$$
$$\langle \mathsf{cas}\ v\rangle \longrightarrow \langle \mathsf{rd}\ v\rangle \text{ if } \langle \mathsf{cas}\ v\rangle \cdots \langle \mathsf{rd}\ v\rangle \quad \text{(synchronization)}$$
$$\langle \mathsf{cas}\ v\rangle \longrightarrow \langle \mathsf{cas}\ v\rangle \text{ if } \langle \mathsf{cas}\ v\rangle \cdots \langle \mathsf{cas}\ v\rangle \quad \text{(synchronization)}$$

- This defines $<^{\sigma}_{\mathsf{jmm}}$
  - $<^{\sigma}_{\mathsf{strong}}$ includes conflicts on all variables
  - $<^{\sigma}_{\mathsf{tso}}$ and $<^{\sigma}_{\mathsf{pso}}$ in between

One place buffer (Revisited)

```
var x=0;
fun put (r)  { acq_emp (); x=r; rel_full (); }
fun get ()   { acq_full (); let r=x; rel_emp (); return r; }
```

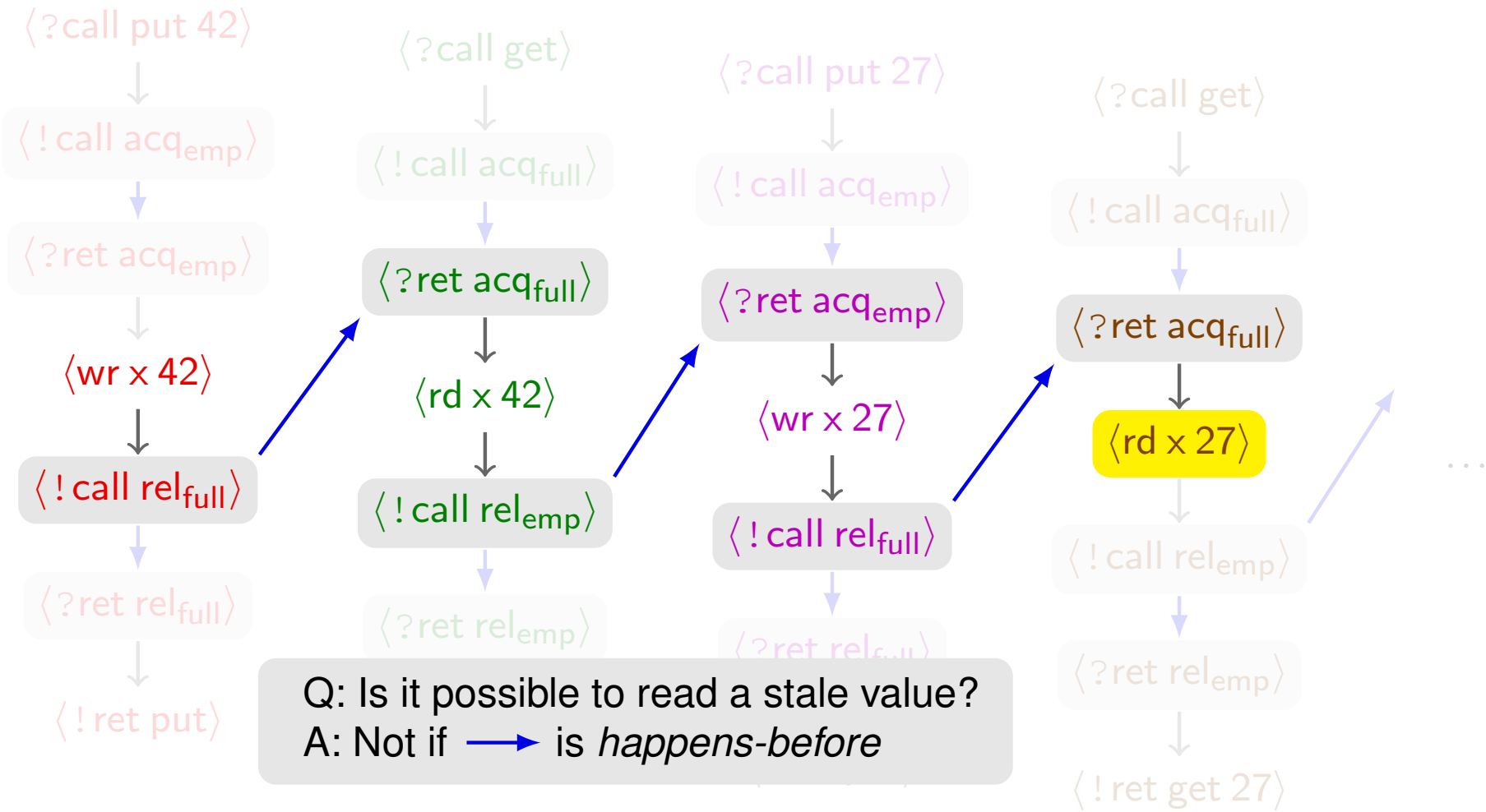


Q: Is it possible to read a stale value?

A: Not if $\longrightarrow$ is *happens-before*

One place buffer (Revisited)

```
var x=0;
fun put (r)  { acq_emp (); x=r; rel_full (); }
fun get ()   { acq_full (); let r=x; rel_emp (); return r; }
```



Q: Is it possible to read a stale value?
A: Not if $\longrightarrow$ is *happens-before*

Details

Define $\Sigma \sqsubseteq_{\mathsf{Lin}} \Phi$ as

$\forall \sigma \in \Sigma.$

$\exists \phi \in \Phi.$

$\exists \pi : [1 \ldots |\sigma|] \to [1 \ldots |\phi|].$

if either $\sigma_i, \sigma_{\pi(i)}$ is $\langle ? \rangle, \langle ! \rangle$ then $\sigma_i = \phi_{\pi(i)}$

if σ_i, σ_j are $\langle ? \rangle, \langle ! \rangle$ and $\pi(i) <^{\phi}_{\mathsf{thrd}} \pi(j)$ then $i < j$

if σ_i, σ_j are $\langle ? \rangle, \langle ! \rangle$ and $i <^{\sigma}_{\mathsf{thrd}} j$ then $\pi(i) < \pi(j)$

if $\sigma_i = \langle ! \rangle$, $\sigma_j = \langle ? \rangle$ and $i < j$ then $\pi(i) < \pi(j)$

σ has same I/O actions as ϕ

σ has same thread order as ϕ

nonoverlapping order of σ respected by ϕ

Details

Define $\Sigma \sqsubseteq_{\mathsf{Lin}} \Phi$ as

$\forall \sigma \in \Sigma.$
$\exists \phi \in \Phi.$
$\exists \pi : [1 \ldots |\sigma|] \to [1 \ldots |\phi|].$

if either $\sigma_i, \sigma_{\pi(i)}$ is $\langle ? \rangle, \langle ! \rangle$ then $\sigma_i = \phi_{\pi(i)}$

if σ_i, σ_j are $\langle ? \rangle, \langle ! \rangle$ and $\pi(i) <^{\phi}_{\mathsf{thrd}} \pi(j)$ then $i < j$

if σ_i, σ_j are $\langle ? \rangle, \langle ! \rangle$ and $i <^{\sigma}_{\mathsf{thrd}} j$ then $\pi(i) < \pi(j)$

if $\sigma_i = \langle ! \rangle$, $\sigma_j = \langle ? \rangle$ and $i < j$ then $\pi(i) < \pi(j)$

σ has same I/O actions as ϕ

σ has same thread order as ϕ

nonoverlapping order of σ respected by ϕ

Details

Define $\Sigma \sqsubseteq_{\mathsf{Lin}} \Phi$ as

$\forall \sigma \in \Sigma.$
$\exists \phi \in \Phi.$
$\exists \pi : [1 \ldots |\sigma|] \to [1 \ldots |\phi|].$

if either $\sigma_i, \sigma_{\pi(i)}$ is $\langle ? \rangle, \langle ! \rangle$ then $\sigma_i = \phi_{\pi(i)}$

if σ_i, σ_j are $\langle ? \rangle, \langle ! \rangle$ and $\pi(i) <^{\phi}_{\mathsf{thrd}} \pi(j)$ then $i < j$

if σ_i, σ_j are $\langle ? \rangle, \langle ! \rangle$ and $i <^{\sigma}_{\mathsf{thrd}} j$ then $\pi(i) < \pi(j)$

if $\sigma_i = \langle ! \rangle$, $\sigma_j = \langle ? \rangle$ and $i < j$ then $\pi(i) < \pi(j)$

σ has same I/O actions as ϕ

σ has same thread order as ϕ

nonoverlapping order of σ respected by ϕ

Details

Define $\Sigma \sqsubseteq_{\mathscr{W}} \Phi$ as

$\forall \sigma \in \Sigma.$

$\exists \phi \in \Phi.$

$\exists \pi : [1 \ldots |\sigma|] \to [1 \ldots |\phi|].$

if either σ_i, $\sigma_{\pi(i)}$ is $\langle ? \rangle$, $\langle ! \rangle$ then $\sigma_i = \phi_{\pi(i)}$

if $\sigma_i = \langle ? \rangle$, $\sigma_j = \langle ! \rangle$ and $\pi(i) <^{\phi}_{\mathscr{W}} \pi(j)$ then $i <^{\sigma}_{\mathscr{W}} j$

if $\sigma_i = \langle ? \rangle$, $\sigma_j = \langle ! \rangle$ and $i <^{\sigma}_{\mathscr{W}} j$ then $\pi(i) < \pi(j)$

if $\sigma_i = \langle ! \rangle$, $\sigma_j = \langle ? \rangle$ then $\pi(i) <^{\phi}_{\mathscr{W}} \pi(j)$ iff $i <^{\sigma}_{\mathscr{W}} j$

σ has same I/O actions as ϕ

σ has more $\longrightarrow$ than ϕ

extra order of σ does not contradict ϕ

σ has same $\longrightarrow$ as ϕ

Details

Define $\Sigma \sqsubseteq_{\mathscr{W}} \Phi$ as

$\forall \sigma \in \Sigma.$

$\exists \phi \in \Phi.$

$\exists \pi : [1 \ldots |\sigma|] \to [1 \ldots |\phi|].$

if either $\sigma_i, \sigma_{\pi(i)}$ is $\langle ? \rangle, \langle ! \rangle$ then $\sigma_i = \phi_{\pi(i)}$

if $\sigma_i = \langle ? \rangle$, $\sigma_j = \langle ! \rangle$ and $\pi(i) <^{\phi}_{\mathscr{W}} \pi(j)$ then $i <^{\sigma}_{\mathscr{W}} j$

if $\sigma_i = \langle ? \rangle$, $\sigma_j = \langle ! \rangle$ and $i <^{\sigma}_{\mathscr{W}} j$ then $\pi(i) < \pi(j)$

if $\sigma_i = \langle ! \rangle$, $\sigma_j = \langle ? \rangle$ then $\pi(i) <^{\phi}_{\mathscr{W}} \pi(j)$ iff $i <^{\sigma}_{\mathscr{W}} j$

σ has same I/O actions as ϕ

σ has more $\longrightarrow$ than ϕ

extra order of σ does not contradict ϕ

σ has same $\longrightarrow$ as ϕ

Details

Define $\Sigma \sqsubseteq_{\mathscr{W}} \Phi$ as

$\forall \sigma \in \Sigma.$

$\exists \phi \in \Phi.$

$\exists \pi : [1 \ldots |\sigma|] \to [1 \ldots |\phi|].$

if either $\sigma_i, \sigma_{\pi(i)}$ is $\langle ? \rangle, \langle ! \rangle$ then $\sigma_i = \phi_{\pi(i)}$

if $\sigma_i = \langle ? \rangle$, $\sigma_j = \langle ! \rangle$ and $\pi(i) <^{\phi}_{\mathscr{W}} \pi(j)$ then $i <^{\sigma}_{\mathscr{W}} j$

if $\sigma_i = \langle ? \rangle$, $\sigma_j = \langle ! \rangle$ and $i <^{\sigma}_{\mathscr{W}} j$ then $\pi(i) < \pi(j)$

if $\sigma_i = \langle ! \rangle$, $\sigma_j = \langle ? \rangle$ then $\pi(i) <^{\phi}_{\mathscr{W}} \pi(j)$ iff $i <^{\sigma}_{\mathscr{W}} j$

σ has same I/O actions as ϕ

σ has more $\longrightarrow$ than ϕ

extra order of σ does not contradict ϕ

σ has same $\longrightarrow$ as ϕ

Refinement theorems (client and library have disjoint variables)

- Theorem (Filipović/O'Hearn/Rinetzky/Yang, ESOP 2009)

  if $\Sigma \sqsubseteq_{\text{strong}} \Phi$
  then $[\![P]\!](\Sigma) \subseteq [\![P]\!](\Phi)$

  $\sqsubseteq$ = linearizability
  $\subseteq$ = observational refinement

- Theorem (Burckhardt/Gotsman/Musuvathi/Yang, ESOP 2012)

  if $[\![Q_{\text{impl}}]\!] \sqsubseteq_{\text{tso}} [\![Q_{\text{spec}}]\!]$
  then $[\![P \parallel Q_{\text{impl}}]\!] \subseteq [\![P \parallel Q_{\text{spec}}]\!]$

- Theorem (This paper)

  if $[\![Q]\!] \sqsubseteq_{\mathscr{W}} \Phi_Q$
  and $[\![P]\!] \otimes \Phi_Q \sqsubseteq_{\mathscr{W}} \Phi_P$
  then $[\![P \parallel Q]\!] \sqsubseteq_{\mathscr{W}} \Phi_P$

Refinement theorems (client and library have disjoint variables)

- Theorem (Filipović/O'Hearn/Rinetzky/Yang, ESOP 2009)

  if $\quad \Sigma \sqsubseteq_{\text{strong}} \Phi$

  then $\quad [\![P]\!](\Sigma) \subseteq [\![P]\!](\Phi)$

  "client" P
  "library" impl Σ, spec Φ

- Theorem (Burckhardt/Gotsman/Musuvathi/Yang, ESOP 2012)

  if $\quad [\![Q_{\text{impl}}]\!] \sqsubseteq_{\text{tso}} [\![Q_{\text{spec}}]\!]$

  then $\quad [\![P \parallel Q_{\text{impl}}]\!] \subseteq [\![P \parallel Q_{\text{spec}}]\!]$

- Theorem (This paper)

  if $\quad [\![Q]\!] \sqsubseteq_{\mathscr{W}} \Phi_Q$

  and $\quad [\![P]\!] \otimes \Phi_Q \sqsubseteq_{\mathscr{W}} \Phi_P$

  then $\quad [\![P \parallel Q]\!] \sqsubseteq_{\mathscr{W}} \Phi_P$

Refinement theorems (client and library have disjoint variables)

- Theorem (Filipović/O'Hearn/Rinetzky/Yang, ESOP 2009)

  if $\quad \Sigma \sqsubseteq_{\text{strong}} \Phi$

  then $\quad [\![P]\!](\Sigma) \subseteq [\![P]\!](\Phi)$

- Theorem (Burckhardt/Gotsman/Musuvathi/Yang, ESOP 2012)

  if $\quad [\![Q_{\text{impl}}]\!] \sqsubseteq_{\text{tso}} [\![Q_{\text{spec}}]\!]$

  then $\quad [\![P \parallel Q_{\text{impl}}]\!] \subseteq [\![P \parallel Q_{\text{spec}}]\!]$

  Operational composition

- Theorem (This paper)

  if $\quad [\![Q]\!] \sqsubseteq_{\mathscr{W}} \Phi_Q$

  and $\quad [\![P]\!] \otimes \Phi_Q \sqsubseteq_{\mathscr{W}} \Phi_P$

  then $\quad [\![P \parallel Q]\!] \sqsubseteq_{\mathscr{W}} \Phi_P$

Refinement theorems (client and library have disjoint variables)

- Theorem (Filipović/O'Hearn/Rinetzky/Yang, ESOP 2009)

  if $\quad \Sigma \sqsubseteq_{\text{strong}} \Phi$

  then $\quad [\![P]\!](\Sigma) \subseteq [\![P]\!](\Phi)$

- Theorem (Burckhardt/Gotsman/Musuvathi/Yang, ESOP 2012)

  if $\quad [\![Q_{\text{impl}}]\!] \sqsubseteq_{\text{tso}} [\![Q_{\text{spec}}]\!]$

  then $\quad [\![P \parallel Q_{\text{impl}}]\!] \subseteq [\![P \parallel Q_{\text{spec}}]\!]$

  Operational composition
  Operational spec

- Theorem (This paper)

  if $\quad [\![Q]\!] \sqsubseteq_{\mathscr{W}} \Phi_Q$

  and $\quad [\![P]\!] \otimes \Phi_Q \sqsubseteq_{\mathscr{W}} \Phi_P$

  then $\quad [\![P \parallel Q]\!] \sqsubseteq_{\mathscr{W}} \Phi_P$

Refinement theorems (client and library have disjoint variables)

- Theorem (Filipović/O'Hearn/Rinetzky/Yang, ESOP 2009)

  if $\quad \Sigma \sqsubseteq_{\mathsf{strong}} \Phi$

  then $\quad [\![P]\!](\Sigma) \subseteq [\![P]\!](\Phi)$

- Theorem (Burckhardt/Gotsman/Musuvathi/Yang, ESOP 2012)

  if $\quad [\![Q_{\mathsf{impl}}]\!] \sqsubseteq_{\mathsf{tso}} [\![Q_{\mathsf{spec}}]\!]$

  then $\quad [\![P \parallel Q_{\mathsf{impl}}]\!] \subseteq [\![P \parallel Q_{\mathsf{spec}}]\!]$

- Theorem (This paper)

  if $\quad [\![Q]\!] \sqsubseteq_{\mathscr{W}} \Phi_Q$

  and $\quad [\![P]\!] \otimes \Phi_Q \sqsubseteq_{\mathscr{W}} \Phi_P$

  then $\quad [\![P \parallel Q]\!] \sqsubseteq_{\mathscr{W}} \Phi_P$

Arbitrary spec for library

Refinement theorems (client and library have disjoint variables)

- Theorem (Filipović/O'Hearn/Rinetzky/Yang, ESOP 2009)

  if $\quad \Sigma \sqsubseteq_{\text{strong}} \Phi$

  then $\quad [\![P]\!](\Sigma) \subseteq [\![P]\!](\Phi)$

- Theorem (Burckhardt/Gotsman/Musuvathi/Yang, ESOP 2012)

  if $\quad [\![Q_{\text{impl}}]\!] \sqsubseteq_{\text{tso}} [\![Q_{\text{spec}}]\!]$

  then $\quad [\![P \parallel Q_{\text{impl}}]\!] \subseteq [\![P \parallel Q_{\text{spec}}]\!]$

- Theorem (This paper)

  if $\quad [\![Q]\!] \sqsubseteq_{\mathscr{W}} \Phi_Q$

  and $\quad [\![P]\!] \otimes \Phi_Q \sqsubseteq_{\mathscr{W}} \Phi_P$

  then $\quad [\![P \parallel Q]\!] \sqsubseteq_{\mathscr{W}} \Phi_P$

Arbitrary spec for library
Explicit tensor

Refinement theorems (client and library have disjoint variables)

- Theorem (Filipović/O'Hearn/Rinetzky/Yang, ESOP 2009)

  if $\quad \Sigma \sqsubseteq_{\mathsf{strong}} \Phi$

  then $\quad [\![P]\!](\Sigma) \subseteq [\![P]\!](\Phi)$

- Theorem (Burckhardt/Gotsman/Musuvathi/Yang, ESOP 2012)

  if $\quad [\![Q_{\mathsf{impl}}]\!] \sqsubseteq_{\mathsf{tso}} [\![Q_{\mathsf{spec}}]\!]$

  then $\quad [\![P \parallel Q_{\mathsf{impl}}]\!] \subseteq [\![P \parallel Q_{\mathsf{spec}}]\!]$

- Theorem (This paper)

  if $\quad [\![Q]\!] \sqsubseteq_{\mathscr{W}} \Phi_Q$

  and $\quad [\![P]\!] \otimes \Phi_Q \sqsubseteq_{\mathscr{W}} \Phi_P$

  then $\quad [\![P \parallel Q]\!] \sqsubseteq_{\mathscr{W}} \Phi_P$

if library satisfies spec
and client correct using spec
then composed system correct
$\mathscr{W} \in \{\mathsf{strong}, \mathsf{tso}, \mathsf{pso}, \mathsf{jmm}\}$

Refinement theorems (client and library have disjoint variables)

- Theorem (Filipović/O'Hearn/Rinetzky/Yang, ESOP 2009)

  if $\quad \Sigma \sqsubseteq_{\text{strong}} \Phi$

  then $\quad [\![P]\!](\Sigma) \subseteq [\![P]\!](\Phi)$

- Theorem (Burckhardt/Gotsman/Musuvathi/Yang, ESOP 2012)

  if $\quad [\![Q_{\text{impl}}]\!] \sqsubseteq_{\text{tso}} [\![Q_{\text{spec}}]\!]$

  then $\quad [\![P \parallel Q_{\text{impl}}]\!] \subseteq [\![P \parallel Q_{\text{spec}}]\!]$

- Corollary (This paper)

  if $\quad [\![Q]\!] \sqsubseteq_{\mathscr{W}} \Phi_Q$

  and $\quad [\![P]\!] \otimes \Phi_Q \sqsubseteq_{\text{strong}} \sqsubseteq_{\mathscr{W}} \Phi_P$

  and $\quad [\![P]\!]$ is locally SC, ...

  then $\quad [\![P \parallel Q]\!] \sqsubseteq_{\mathscr{W}} \Phi_P$

Well synchronized clients are not affected by races in library

Compositionality

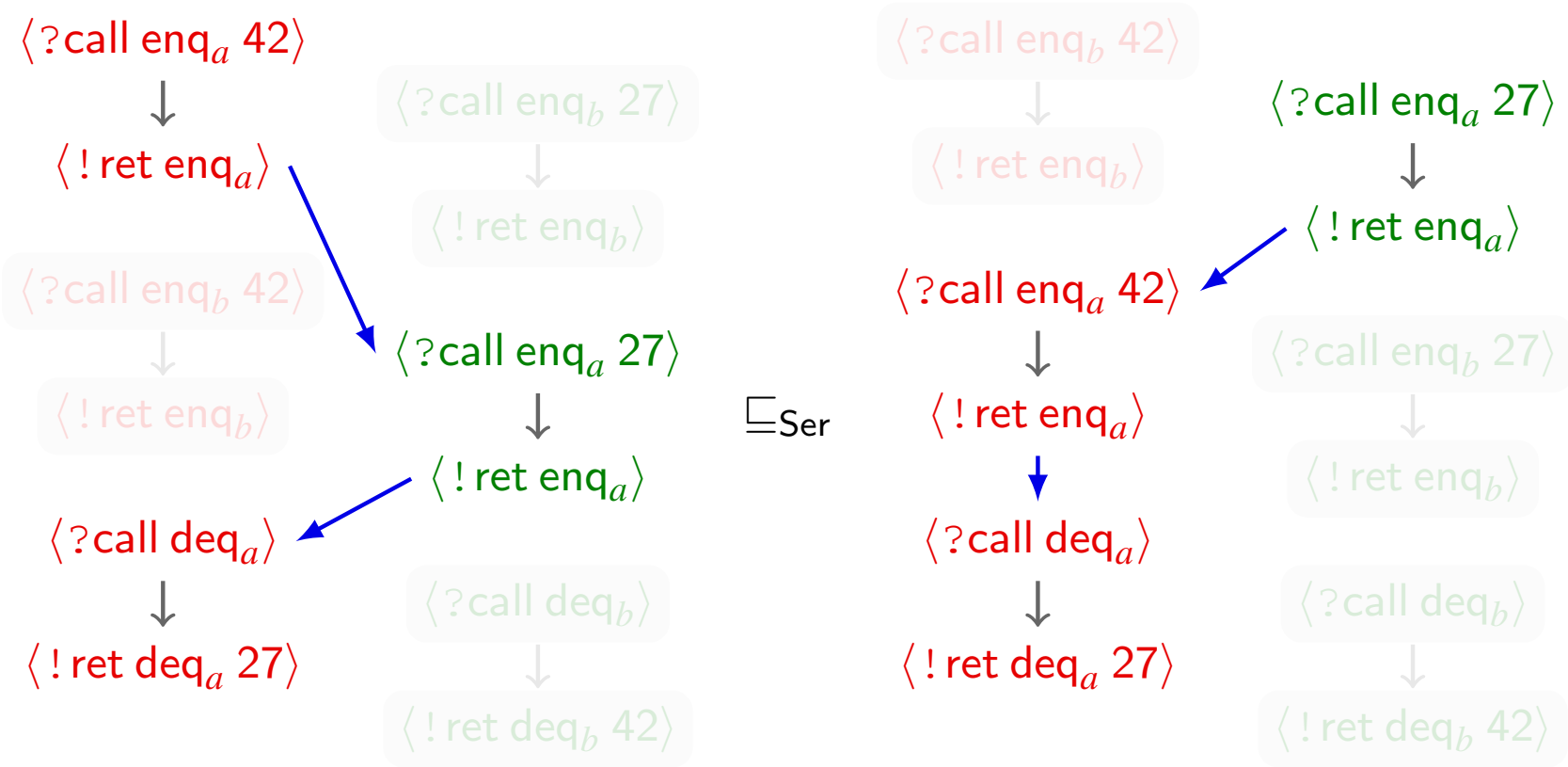
if $[\![P_i]\!] \sqsubseteq_{\mathscr{W}} \Sigma_i$ then $[\![P_1 \parallel P_2]\!] \sqsubseteq_{\mathscr{W}} \Sigma_1 \otimes \Sigma_2$

When does it hold?

(P_i have disjoint variables)
(Not a corollary of refinement)

Compositionality counterexample (Herlihy/Wing)

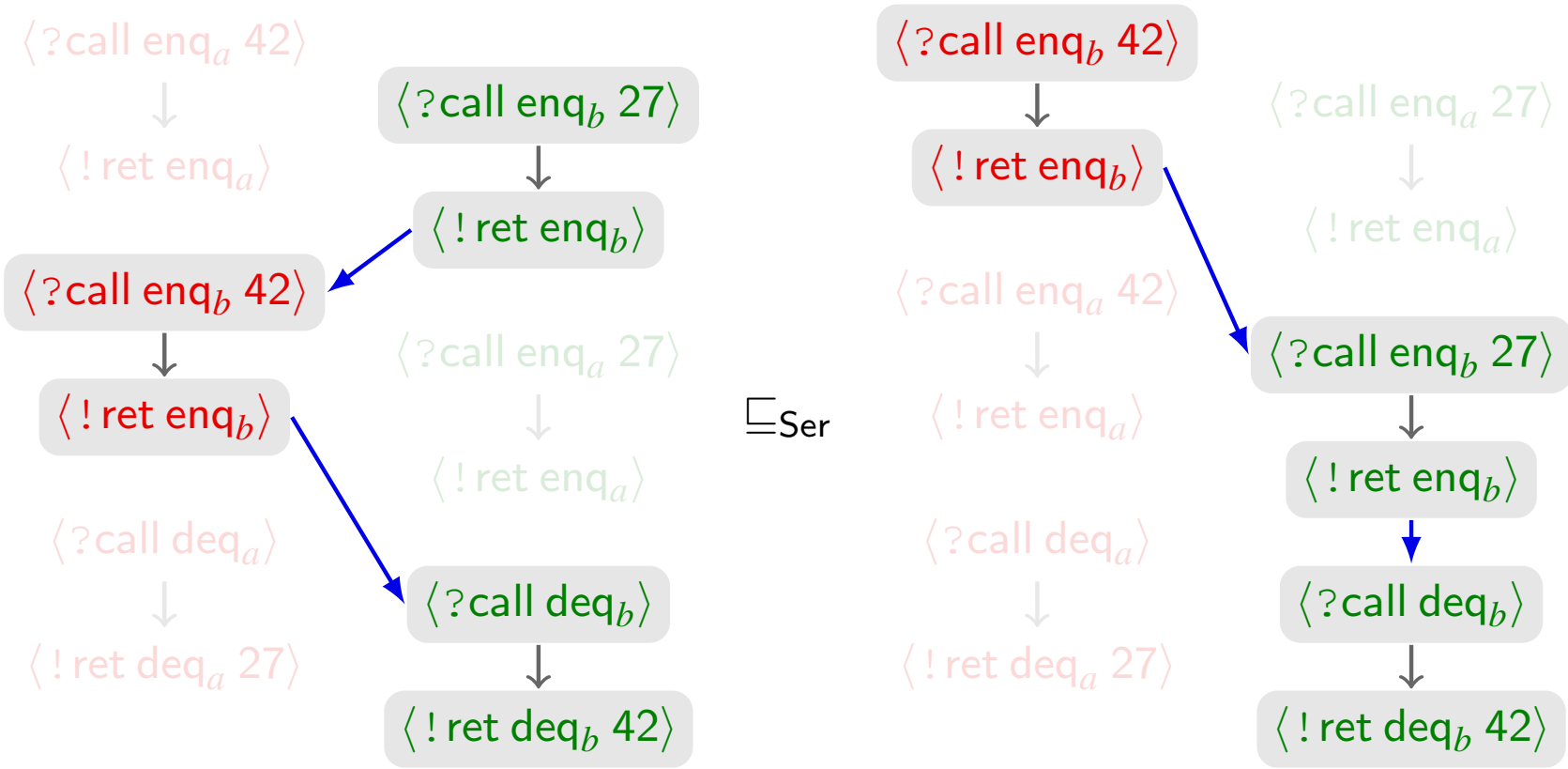
Serializable trace:



→ = “non-overlapping”: return before call.

Compositionality counterexample (Herlihy/Wing)

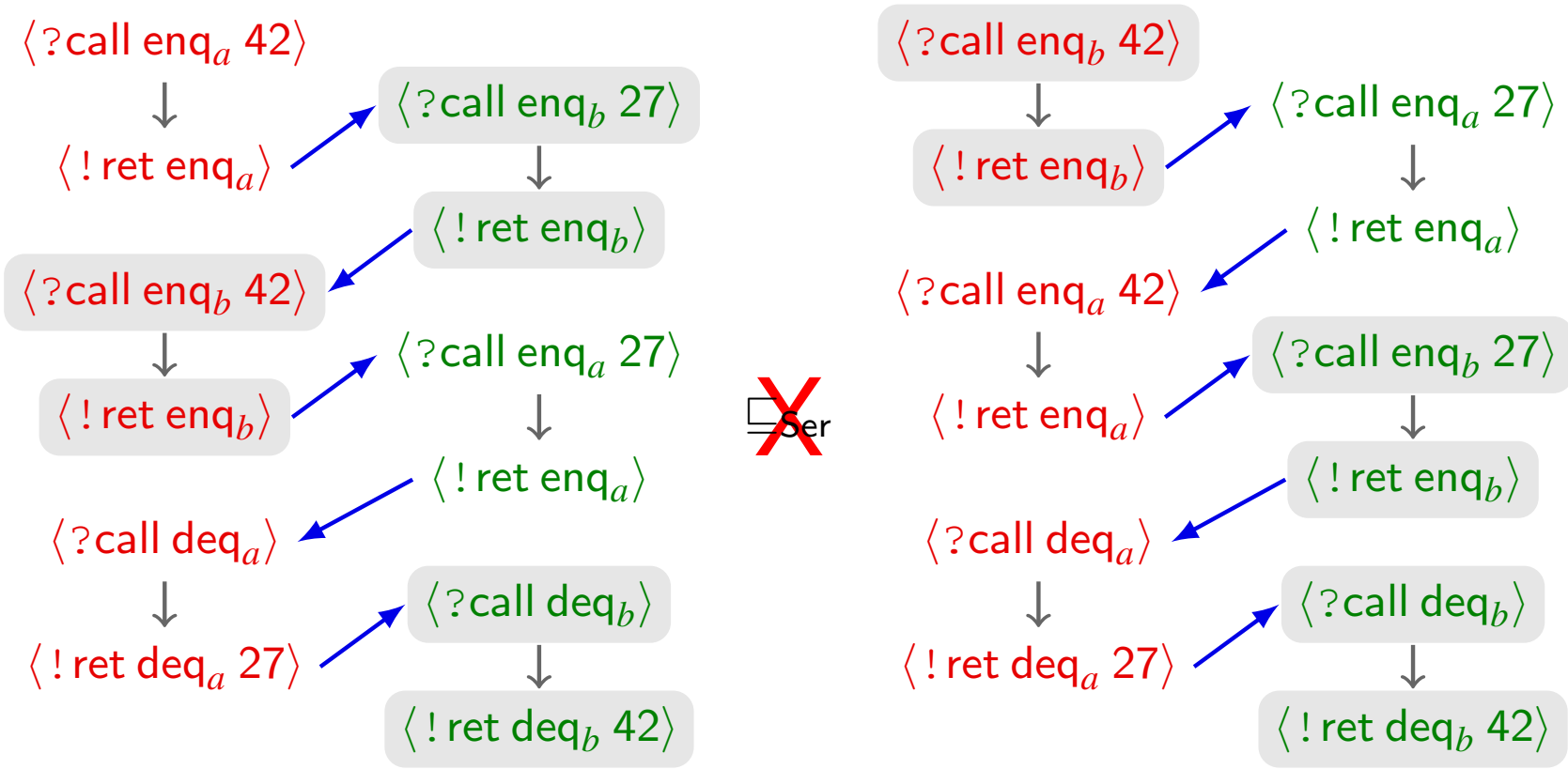
Serializable trace:



→ = “non-overlapping”: return before call.

Compositionality counterexample (Herlihy/Wing)

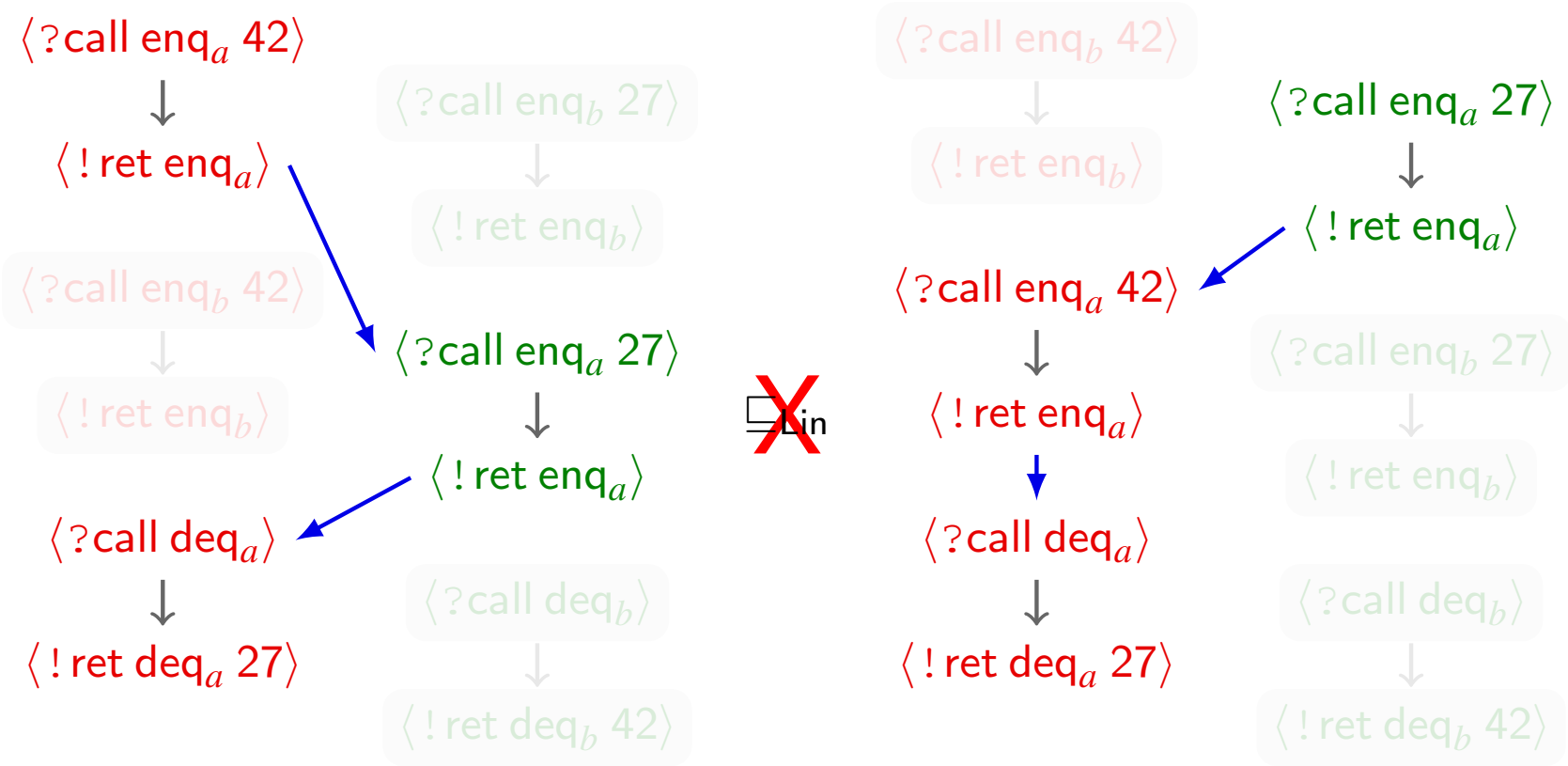
Non-Serializable trace:



→ = “non-overlapping”: return before call.

Compositionality counterexample (Herlihy/Wing)

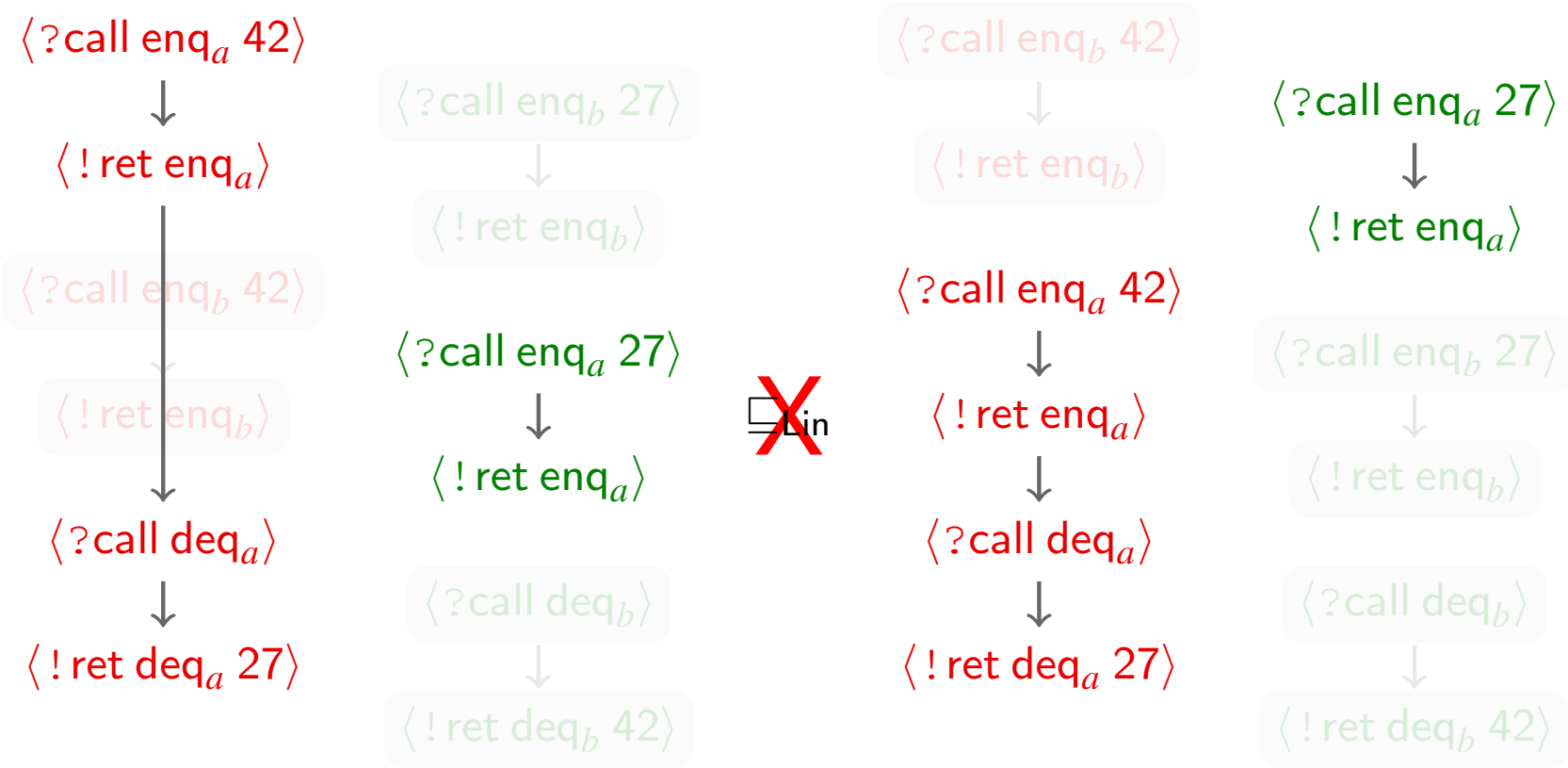
Non-Linearizable trace:



→ = “non-overlapping”: return before call.

Compositionality counterexample (Herlihy/Wing)

Non-Linearizable trace:

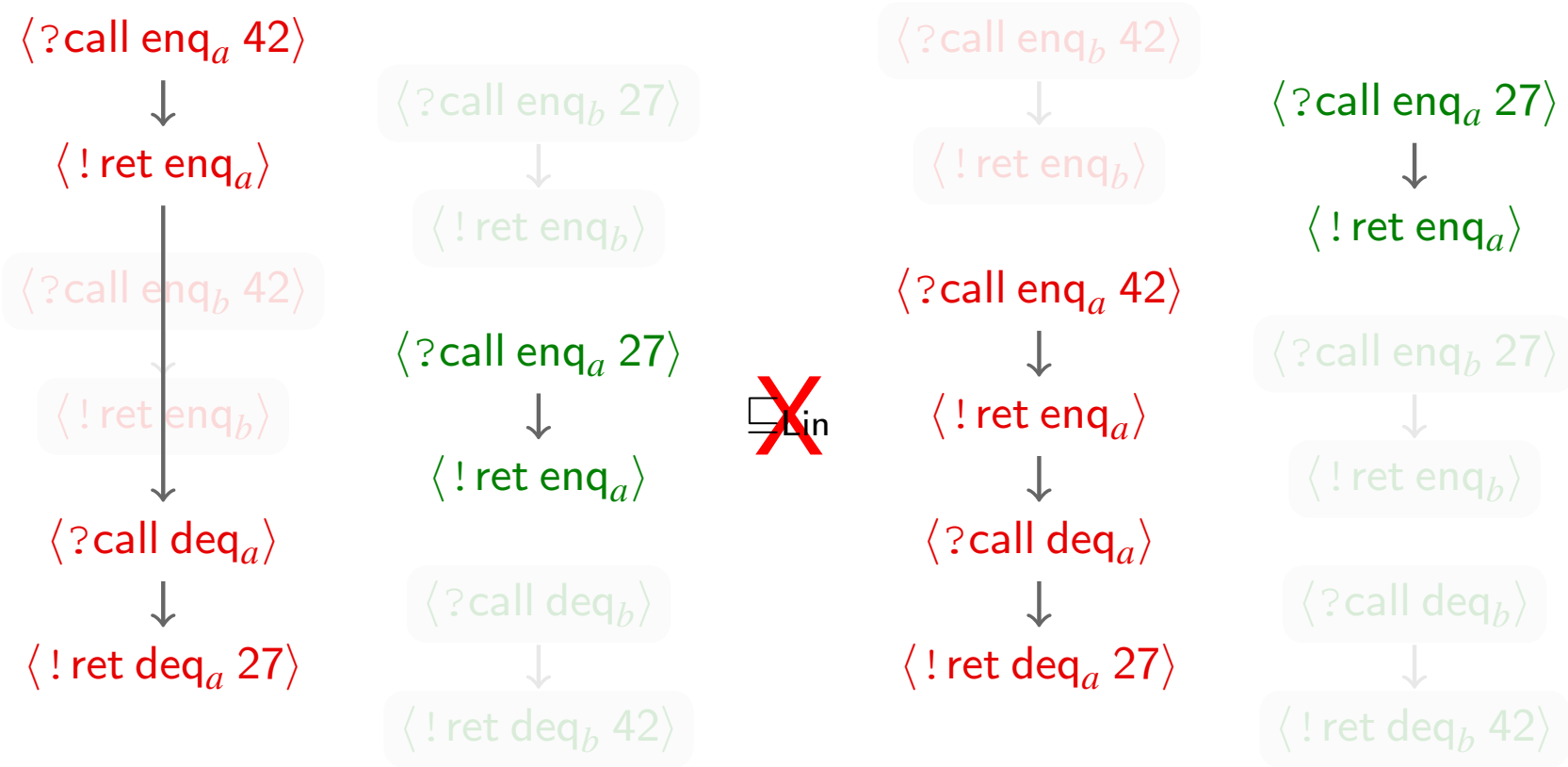


Crucial point:

$$\{\langle ?\ f\rangle\langle !\ f\rangle\langle ?\ g\rangle\langle !\ g\rangle\langle ?\ h\rangle\langle !\ h\rangle\} \not\sqsubseteq_{\mathsf{Lin}} \{\langle ?\ g\rangle\langle !\ g\rangle\langle ?\ f\rangle\langle !\ f\rangle\langle ?\ h\rangle\langle !\ h\rangle\}$$

Compositionality counterexample (Herlihy/Wing)

Non-Linearizable trace:

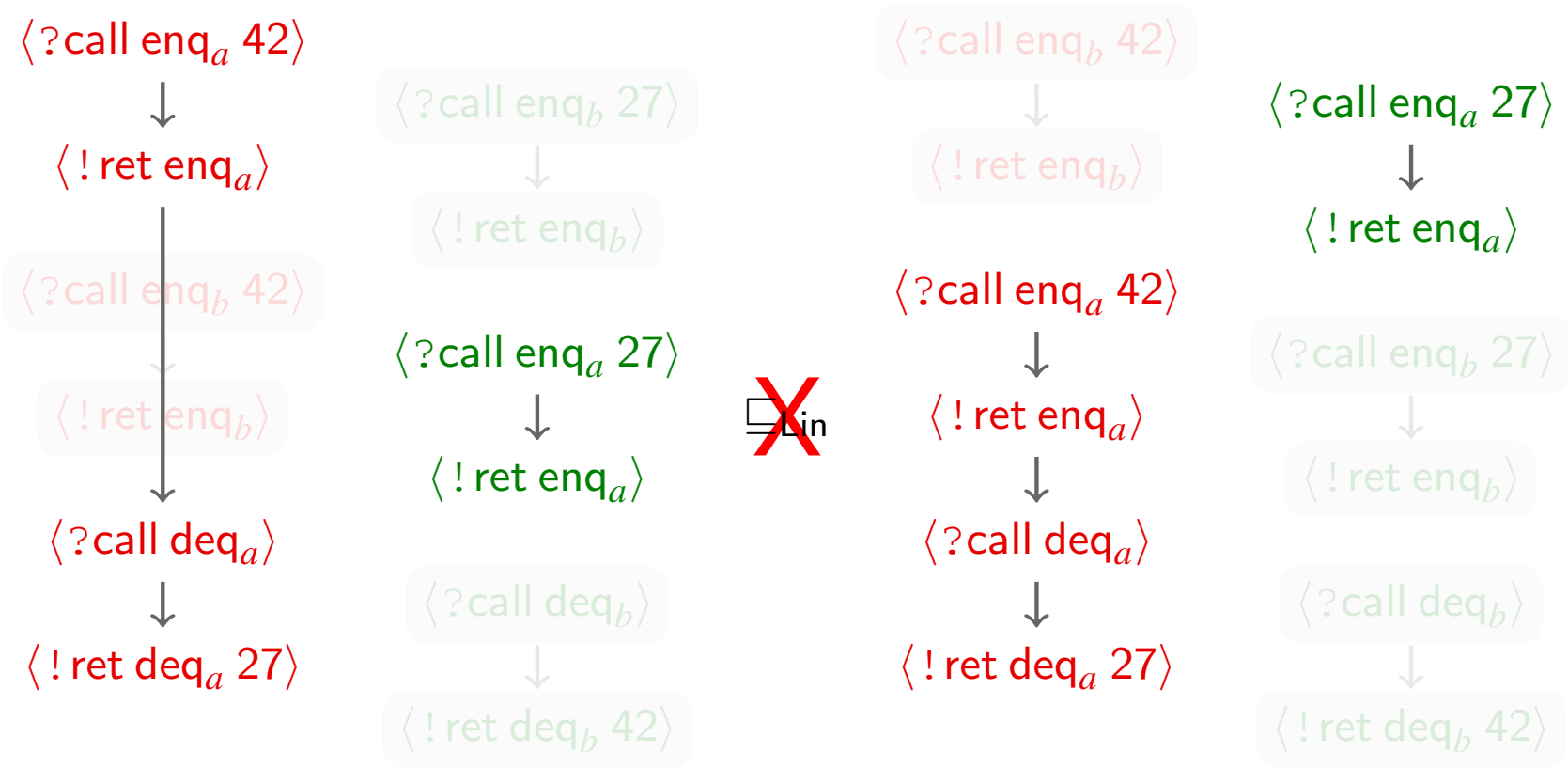


Crucial point:

$$\{\langle ?\ f\rangle\langle !\ f\rangle\langle ?\ g\rangle\langle !\ g\rangle\langle ?\ h\rangle\langle !\ h\rangle\} \sqsubseteq_{\mathsf{Lin}} \left\{ \begin{array}{l} \langle ?\ g\rangle\langle !\ g\rangle\langle ?\ f\rangle\langle !\ f\rangle\langle ?\ h\rangle\langle !\ h\rangle \\ \langle ?\ f\rangle\langle !\ f\rangle\langle ?\ g\rangle\langle !\ g\rangle\langle ?\ h\rangle\langle !\ h\rangle \\ \langle ?\ f\rangle\langle !\ f\rangle\langle ?\ h\rangle\langle !\ h\rangle\langle ?\ g\rangle\langle !\ g\rangle \end{array} \right\}$$

Compositionality counterexample (Herlihy/Wing)

Non-Linearizable trace:

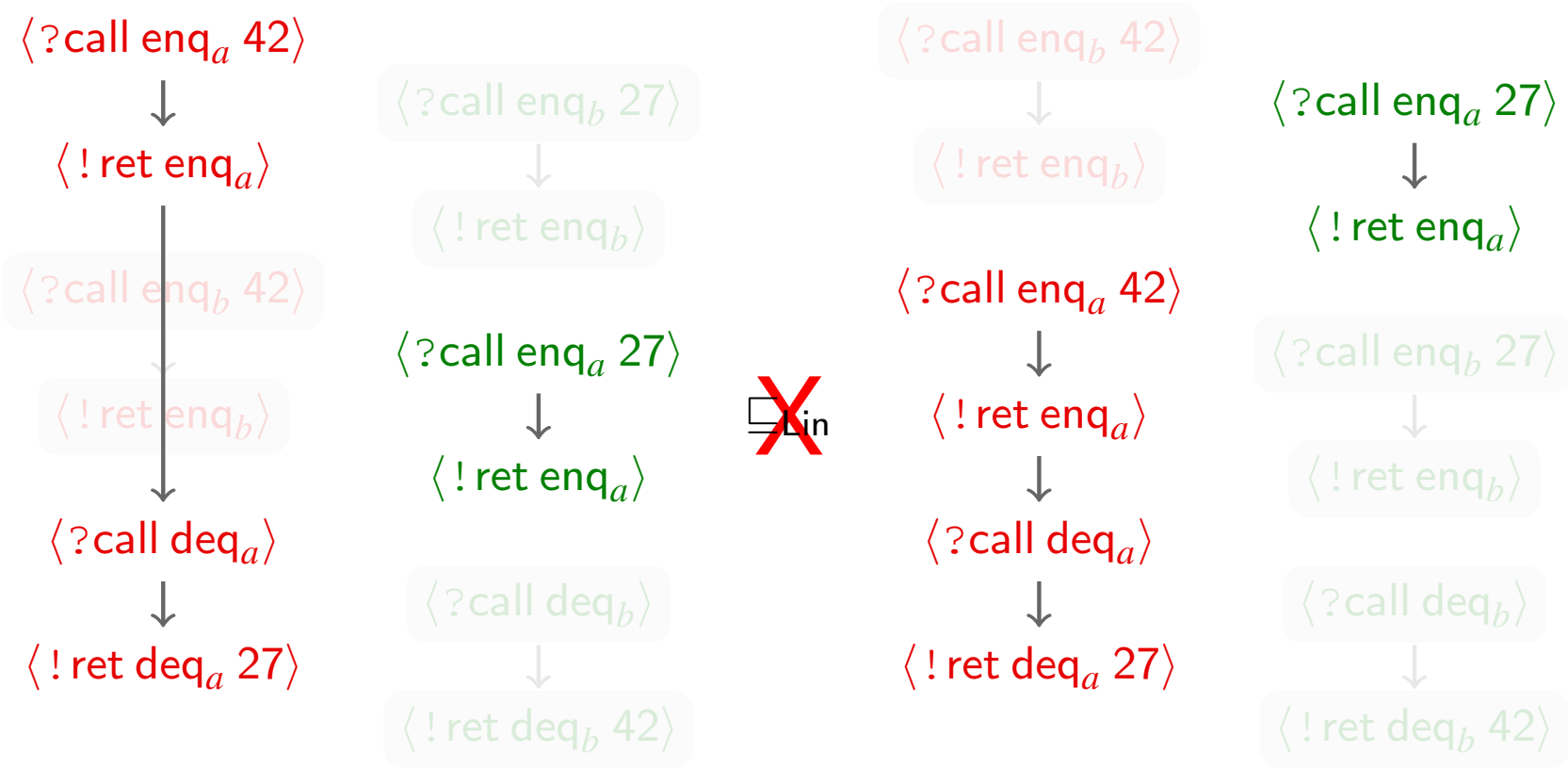


By our definition:

$$\{\langle ?\ f\rangle\langle !\ f\rangle\langle ?\ g\rangle\langle !\ g\rangle\langle ?\ h\rangle\langle !\ h\rangle\} \quad \sqsubseteq_{\mathscr{W}} \quad \{\langle ?\ g\rangle\langle !\ g\rangle\langle ?\ f\rangle\langle !\ f\rangle\langle ?\ h\rangle\langle !\ h\rangle\}$$

Compositionality counterexample (Herlihy/Wing)

Non-Linearizable trace:

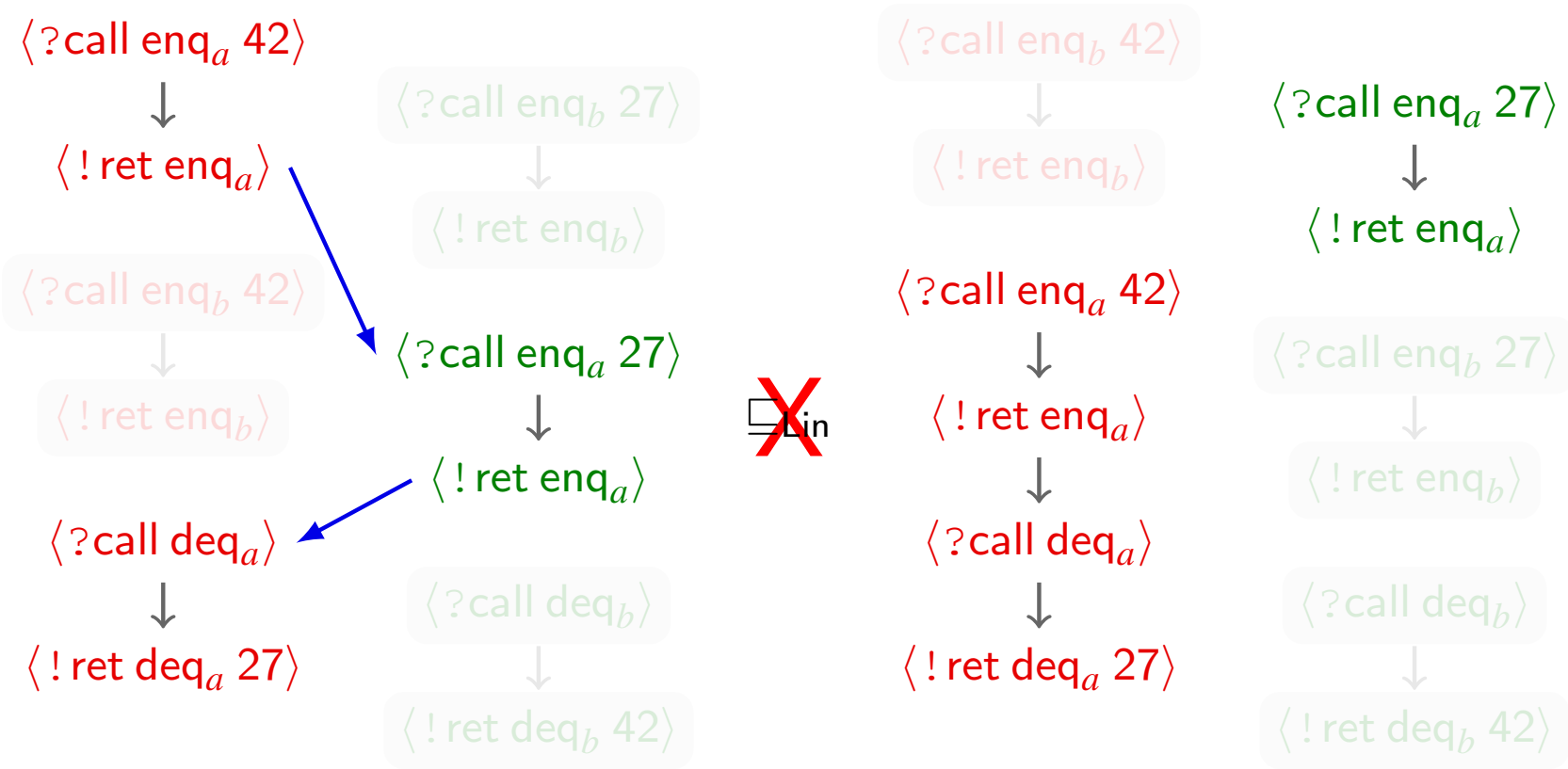


By our definition:

$$\{\langle ?\ f\rangle\langle !\ f\rangle\langle ?\ g\rangle\langle !\ g\rangle\langle ?\ h\rangle\langle !\ h\rangle\} \sqsubseteq_{\mathscr{W}} \left\{ \begin{array}{l} \langle ?\ g\rangle\langle !\ g\rangle\langle ?\ f\rangle\langle !\ f\rangle\langle ?\ h\rangle\langle !\ h\rangle \\ \langle ?\ f\rangle\langle !\ f\rangle\langle ?\ g\rangle\langle !\ g\rangle\langle ?\ h\rangle\langle !\ h\rangle \\ \langle ?\ f\rangle\langle !\ f\rangle\langle ?\ h\rangle\langle !\ h\rangle\langle ?\ g\rangle\langle !\ g\rangle \end{array} \right\}$$

Compositionality counterexample (Herlihy/Wing)

Non-Linearizable trace:

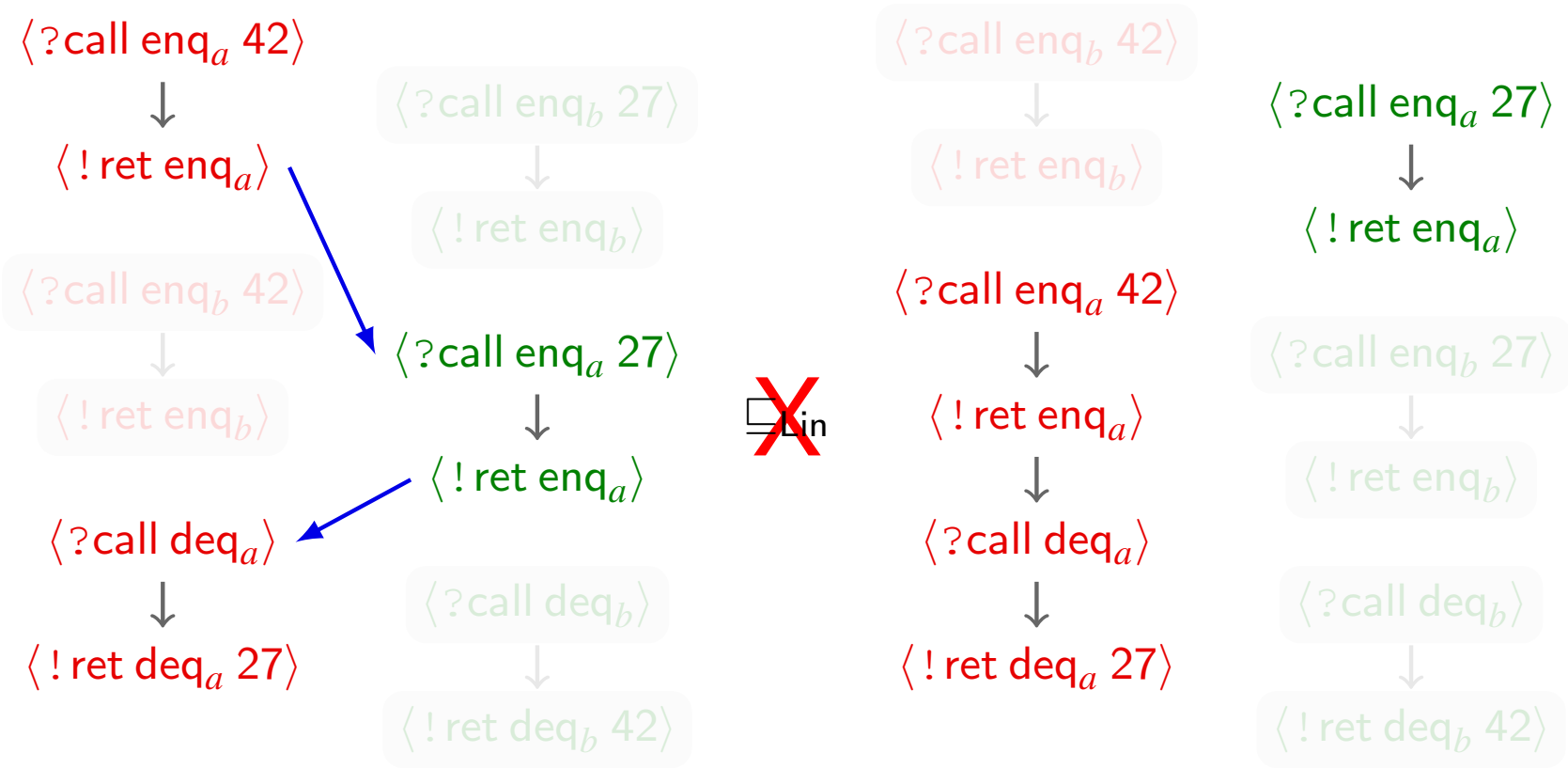


By our definition:

$$\{\langle ?\ f\rangle\langle !\ f\rangle\langle ?\ g\rangle\langle !\ g\rangle\langle ?\ h\rangle\langle !\ h\rangle\} \sqsubseteq_{\mathscr{W}} \left\{ \begin{array}{l} \langle ?\ g\rangle\langle !\ g\rangle\langle ?\ f\rangle\langle !\ f\rangle\langle ?\ h\rangle\langle !\ h\rangle \\ \langle ?\ f\rangle\langle !\ f\rangle\langle ?\ g\rangle\langle !\ g\rangle\langle ?\ h\rangle\langle !\ h\rangle \\ \langle ?\ f\rangle\langle !\ f\rangle\langle ?\ h\rangle\langle !\ h\rangle\langle ?\ g\rangle\langle !\ g\rangle \end{array} \right\}$$

Compositionality counterexample (Herlihy/Wing)

Non-Linearizable trace:



In the absence of the *happens-before* edge $\langle !\ g\rangle \longrightarrow \langle ?\ f\rangle$, is this:

$$\{\langle ?\ g\rangle\langle !\ g\rangle\langle ?\ f\rangle\langle !\ f\rangle\langle ?\ h\rangle\langle !\ h\rangle\}$$

a reasonable spec?

“Accidental” versus “essential” order

- Same under strong memory, not weak
- Another route to compositionality:

  Ban accidental order in specs (closure property)
- Bags, not stacks ...