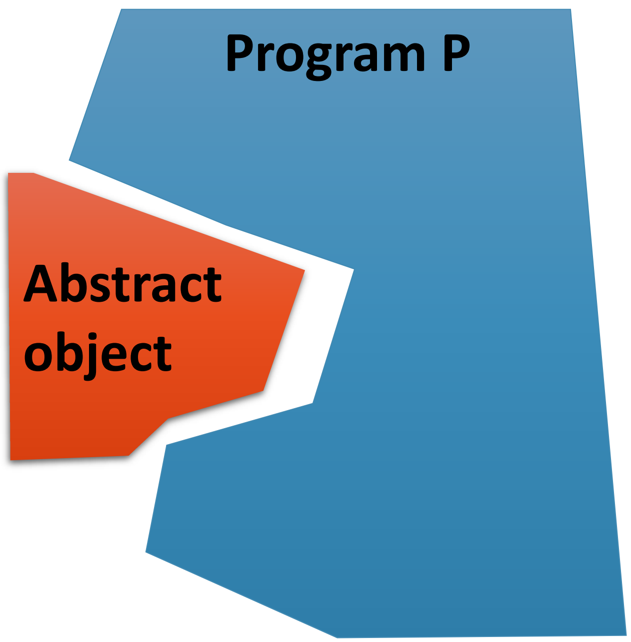
Preserving Hyperproperties when using Concurrent Objects

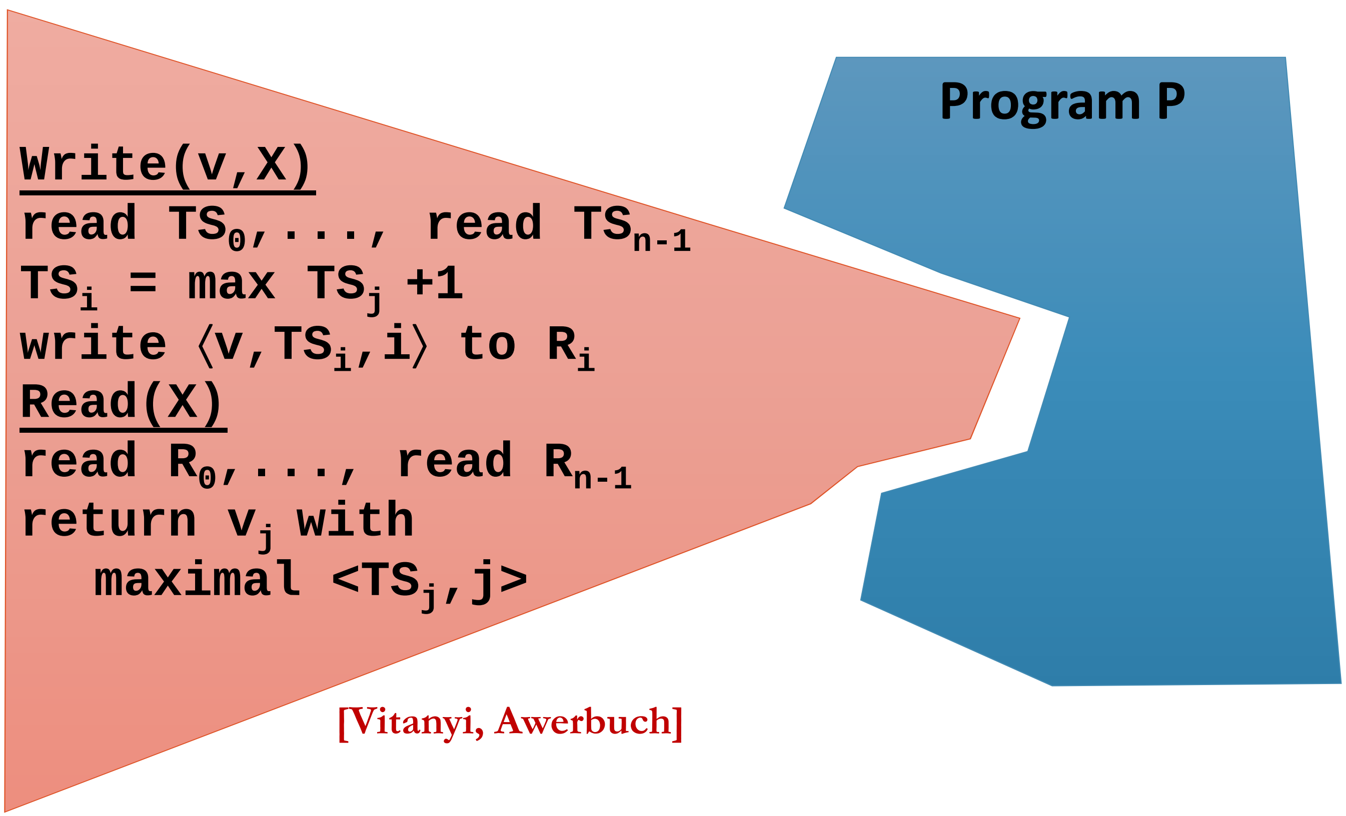
Hagit Attiya (Technion)
Constantin Enea (Ecole Polytechnique)
Jennifer Welch (Texas A&M University)

Abstraction



E.g., Using an Abstract Multi-Writer Register

Implemented from Single-Writer Registers



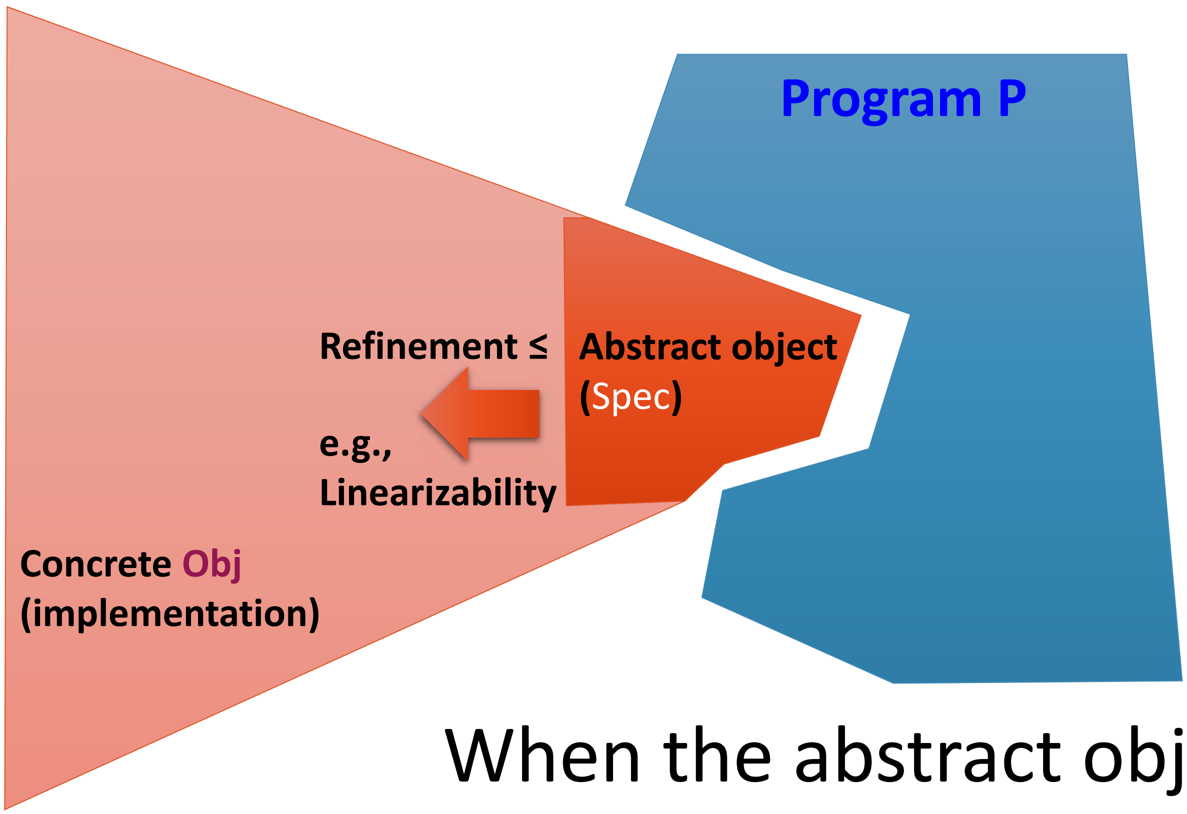
Or in Message-Passing

```
Write(v)
bcast ⟨"query"⟩ & wait
   for > n/2 replies
t = largest timestamp
bcast ⟨"update",v,t+1⟩ & wait
   for > n/2 acks
Read()
bcast ⟨"query"⟩ & wait
   for > n/2 replies
(v,t) = pair with largest t
bcast ⟨"update",v,t⟩ & wait
   for > n/2 acks
return v
```

[A, Bar-Noy, Dolev]
[Lynch, Schwarzmann]

Refinement (Trace Inclusion)

Obj ≤ Spec iff ∀ program P, Traces(P X Obj) ⊆ Traces(P X Spec)

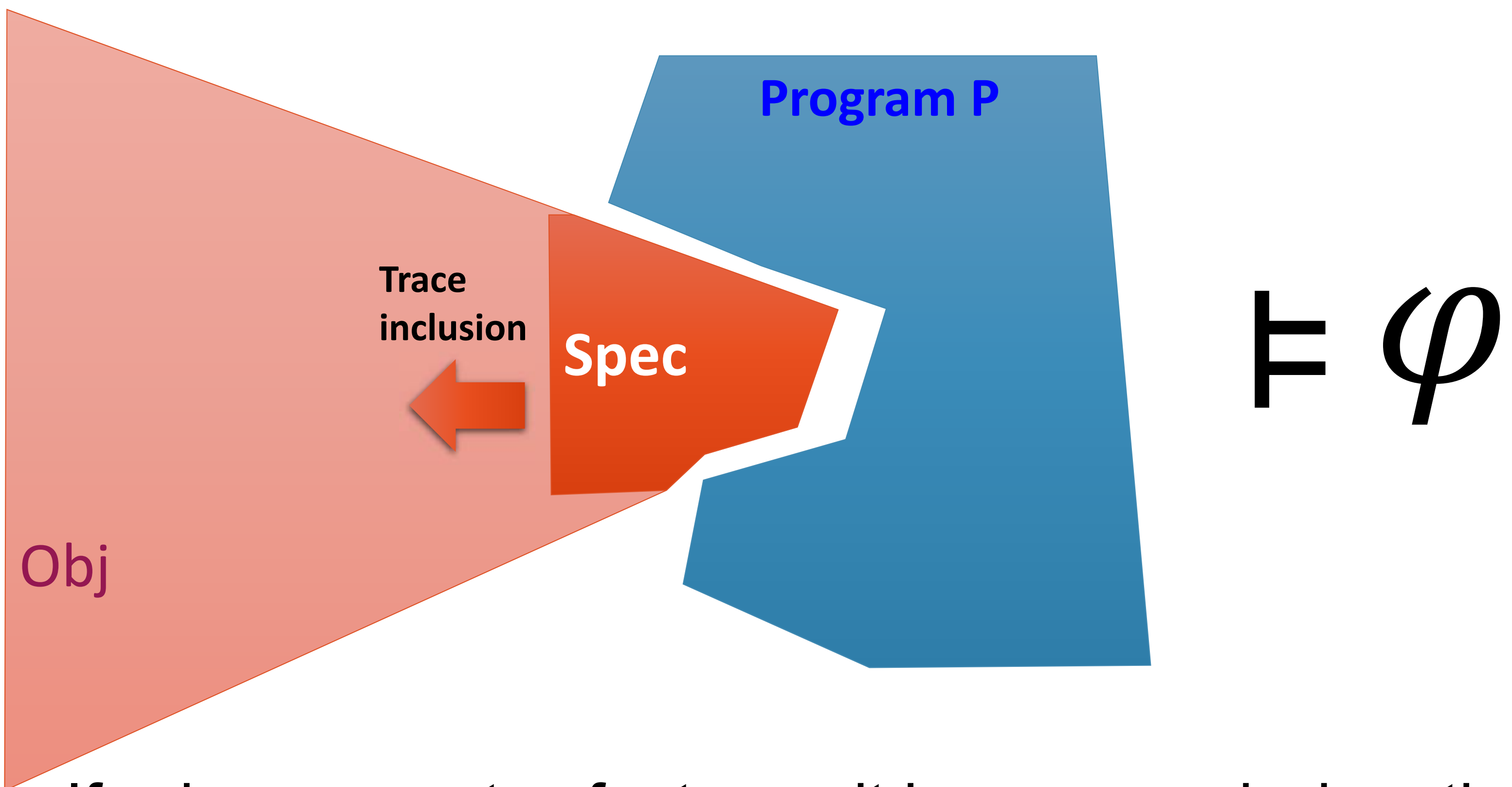


When the abstract object is sequential,

≡ linearizability [Herlihy, Wing]

Refinement Preserves Trace Properties

Obj ≤ Spec iff ∀ program P, Traces(P X Obj) ⊆ Traces(P X Spec)

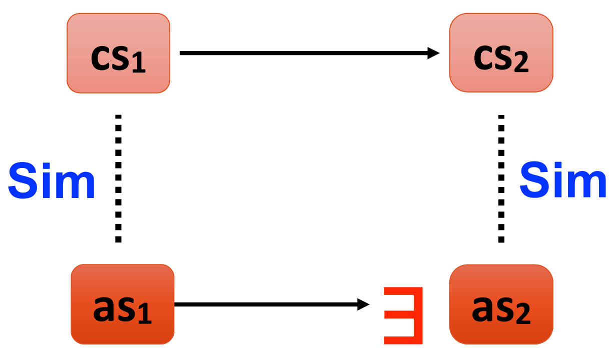


If φ is a property of a trace, it is preserved when the atomic object is replaced with a linearizable implementation

Simulations

Prove refinement by relating states of **abstract** and **concrete** objects

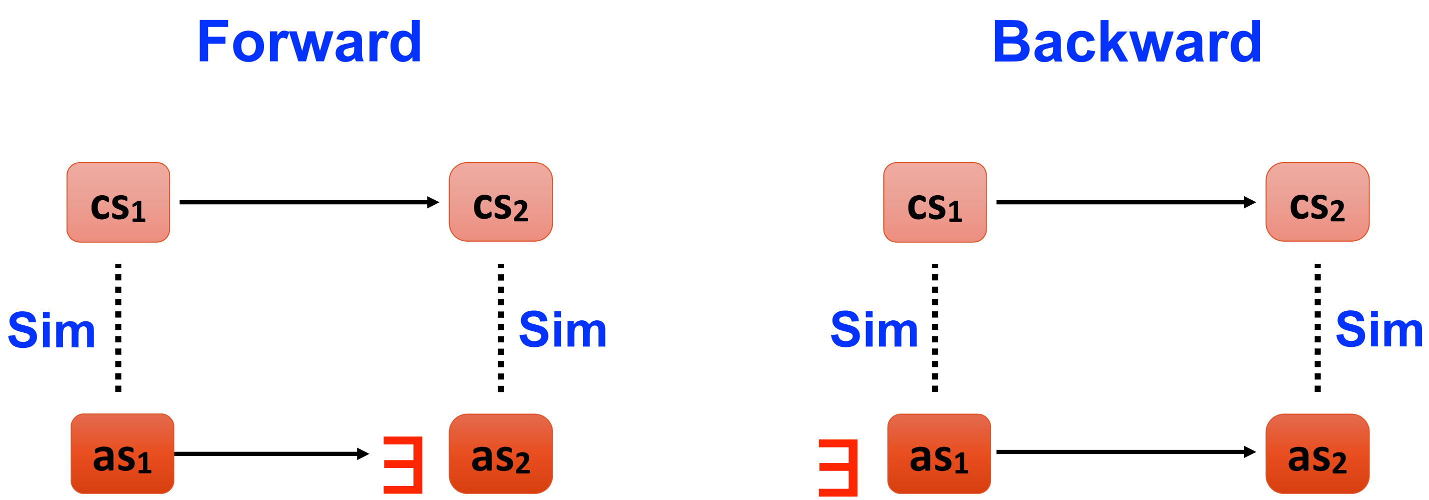
Forward



Forward simulations ≡ proofs based on **explicit** linearization points, e.g., universal constructions using consensus objects or Compare&Swap

Simulations

Prove refinement by relating states of **abstract** and **concrete** objects



In some cases, find an after-the-fact relation (e.g, based on timestamps)

Linearizability can always be proved with forward & backward simulation

[Lynch, Vaandrager]

Hyper (Safety) Properties

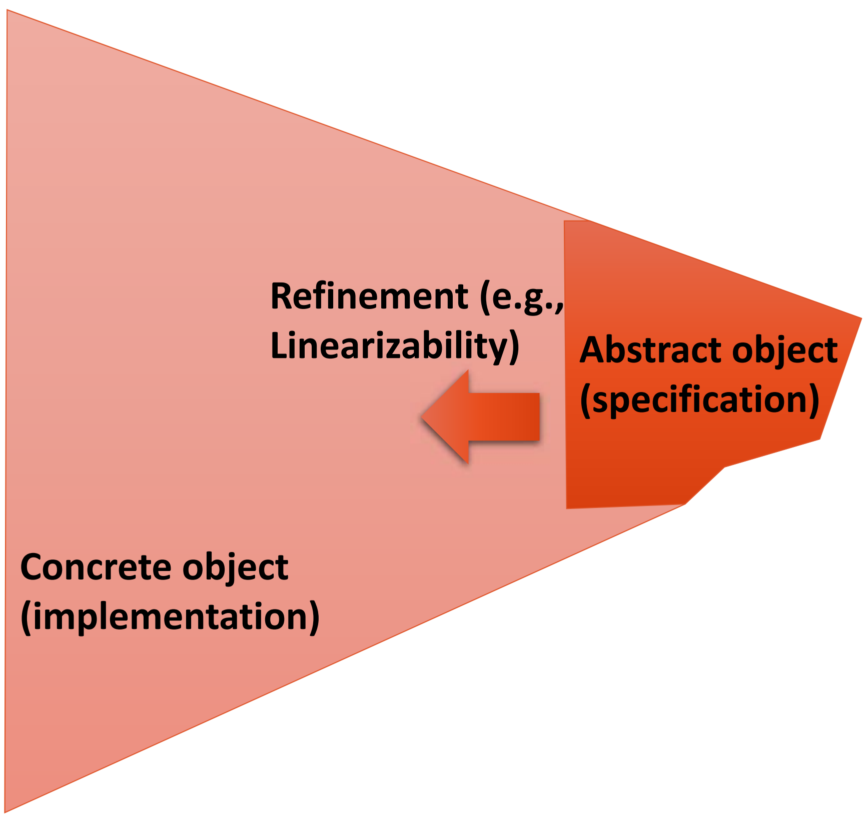
Security policies: e.g., noninterference (high clearance values cannot be observed by low clearance users)

Quantitative properties: termination w.h.p., mean response time, probability distributions

Hyperproperties are properties of sets of traces

Hyper safety properties are properties of sets of finite traces

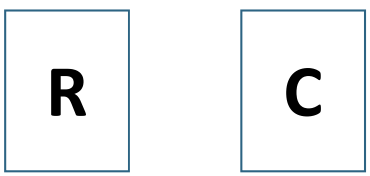
Hyperproperties vs. Refinement



Refinement does not preserve hyperproperties [McLean 1994]

Linearizability does not preserve probability distributions under strong / weak adversaries [Golab, Higham, Woelfel, STOC 2011]

Example w/ MWSR Register



Initially $R = \perp,\ C = -1$

<u>Code for p_0, p_1</u>:

$R \leftarrow i$

if $i = 0$ then $C \leftarrow$ flip 0 or 1

<u>Code for p_2</u>:

$r \leftarrow R;\ c \leftarrow C$

if $(c = 0 \wedge r = \perp) \vee (c = 1 \wedge r \neq \perp)$

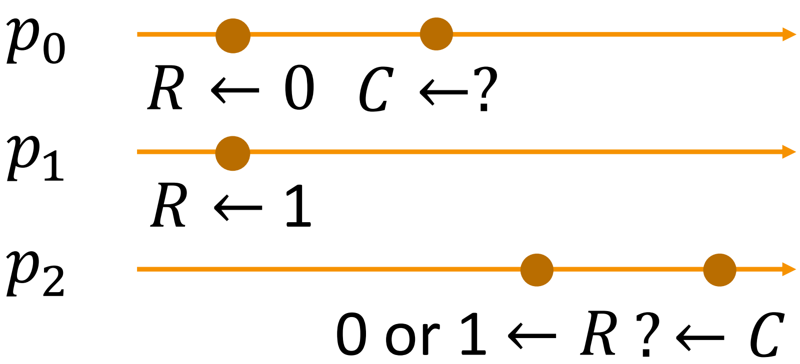
then loop forever

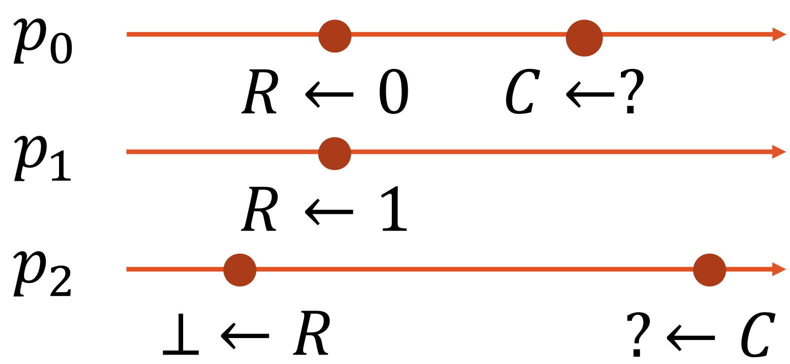
else terminate

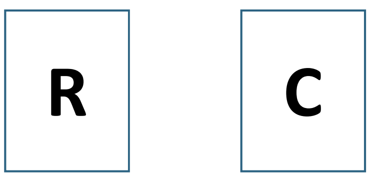
p_2 terminates with probability > ½ w/ atomic mwsr register

Example by Noa Schiller

Example w/ MWSR Register



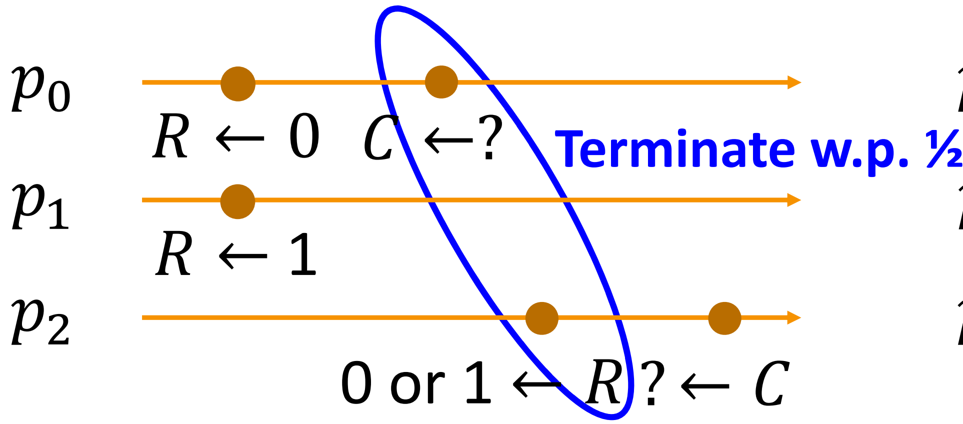


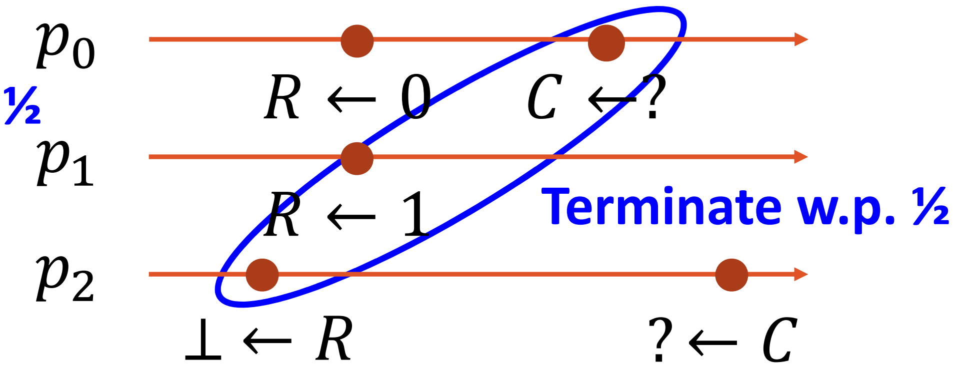


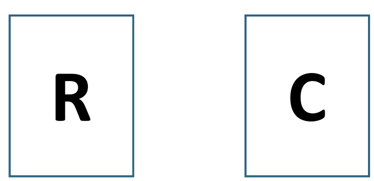

```
Initially R =⊥, C = −1
Code for p0, p1:
R ← i
if i = 0 then C ← flip 0 or 1

Code for p2:
r ← R; c ← C
if (c = 0 ∧ r =⊥) ∨ (c = 1 ∧ r ≠⊥)
      then loop forever
else terminate
```

p_2 terminates with probability > ½ w/ atomic mwsr register

Example w/ MWSR Register







Initially $R = \perp$, $C = -1$

<u>Code for p_0, p_1</u>:

$R \leftarrow i$

if $i = 0$ **then** $C \leftarrow$ **flip 0 or 1**

<u>Code for p_2</u>:

$r \leftarrow R$; $c \leftarrow C$

if $(c = 0 \wedge r = \perp) \vee (c = 1 \wedge r \neq \perp)$

then loop forever

else terminate

p_2 terminates with probability > ½ w/ atomic mwsr register

Using VA Implementation

Write(v,R)
read TS_0,...,read TS_{n-1}
TS_i = max TS_j +1
write ⟨v,TS_i,i⟩ to R_i

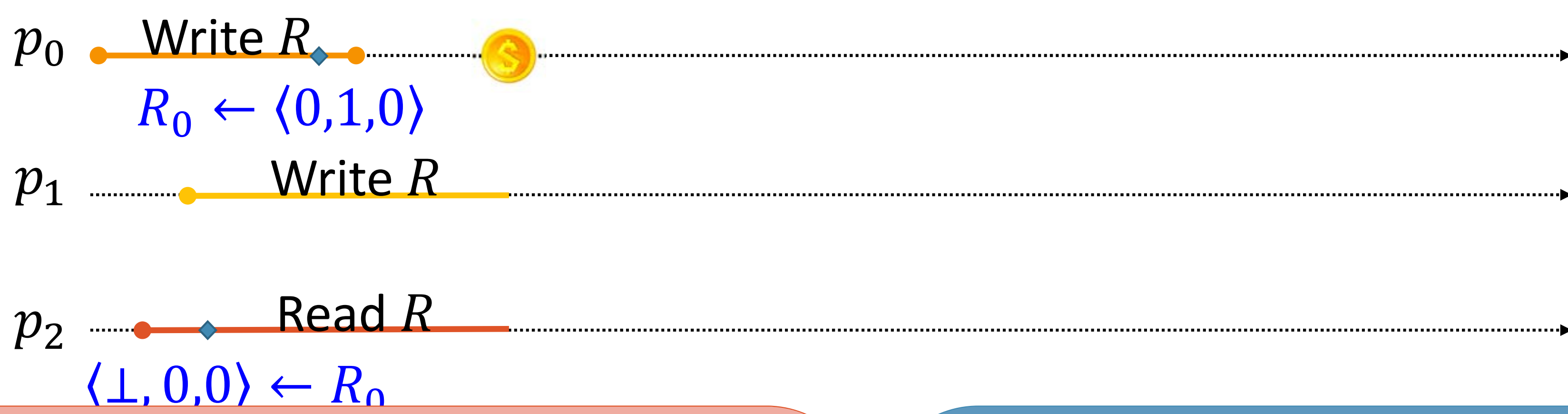
Read(R)
read R_0,...,read R_{n-1}
return v_j with maximal <TS_j,j>

Initially $R = \perp$, $C = -1$
Code for p_0, p_1:
$R \leftarrow i$
if $i = 0$ then $C \leftarrow$ flip 0 or 1

Code for p_2:
$r \leftarrow R$; $c \leftarrow C$
if $(c = 0 \wedge r = \perp) \vee (c = 1 \wedge r \neq \perp)$
then loop forever
else terminate

A strong adversary can make p_2 always loop forever

Using VA Implementation



Write(v,R)
read TS_0,...,read TS_{n-1}
TS_i = max TS_j +1
write $\langle$v,TS_i,i$\rangle$ to R_i

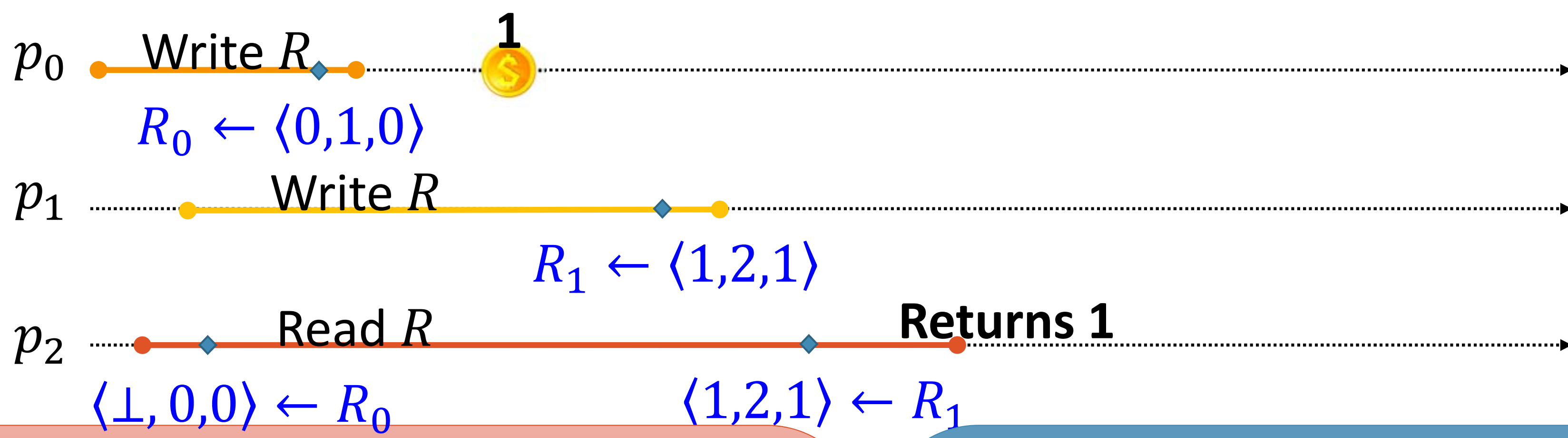
Read(R)
read R_0,...,read R_{n-1}
return v_j with maximal <TS_j,j>

Initially $R = \bot$, $C = -1$
Code for p_0, p_1:
$R \leftarrow i$
if $i = 0$ then $C \leftarrow$ flip 0 or 1

Code for p_2:
$r \leftarrow R$; $c \leftarrow C$
if $(c = 0 \wedge r = \bot) \vee (c = 1 \wedge r \neq \bot)$
then loop forever
else terminate

A strong adversary can make p_2 always loop forever

Example w/ VA: Coin = 1



Write(v,R)
read TS_0,...,read TS_{n-1}
TS_i = max TS_j +1
write $\langle v, TS_i, i \rangle$ to R_i

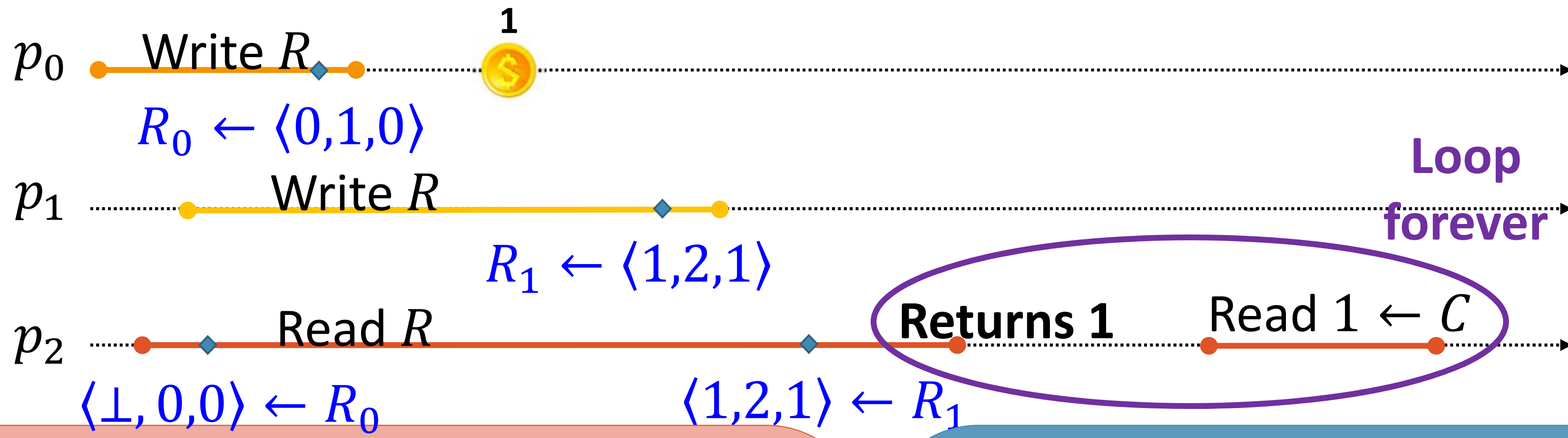
Read(R)
read R_0,...,read R_{n-1}
return v_j with maximal $\langle TS_j, j \rangle$

Initially $R = \bot$, $C = -1$
Code for p_0, p_1:
$R \leftarrow i$
if $i = 0$ then $C \leftarrow$ flip 0 or 1

Code for p_2:
$r \leftarrow R$; $c \leftarrow C$
if $(c = 0 \wedge r = \bot) \vee (c = 1 \wedge r \neq \bot)$
then loop forever
else terminate

A strong adversary can make p_2 always loop forever

Example w/ VA: Coin = 1



Write(v,R)

read TS_0,...,read TS_{n-1}

TS_i = max TS_j +1

write ⟨v,TS_i,i⟩ to R_i

Read(R)

read R_0,...,read R_{n-1}

return v_j with maximal <TS_j,j>

Initially $R = \perp$, $C = -1$

Code for p_0, p_1:

$R \leftarrow i$

if $i = 0$ then $C \leftarrow$ flip 0 or 1

Code for p_2:

$r \leftarrow R$; $c \leftarrow C$

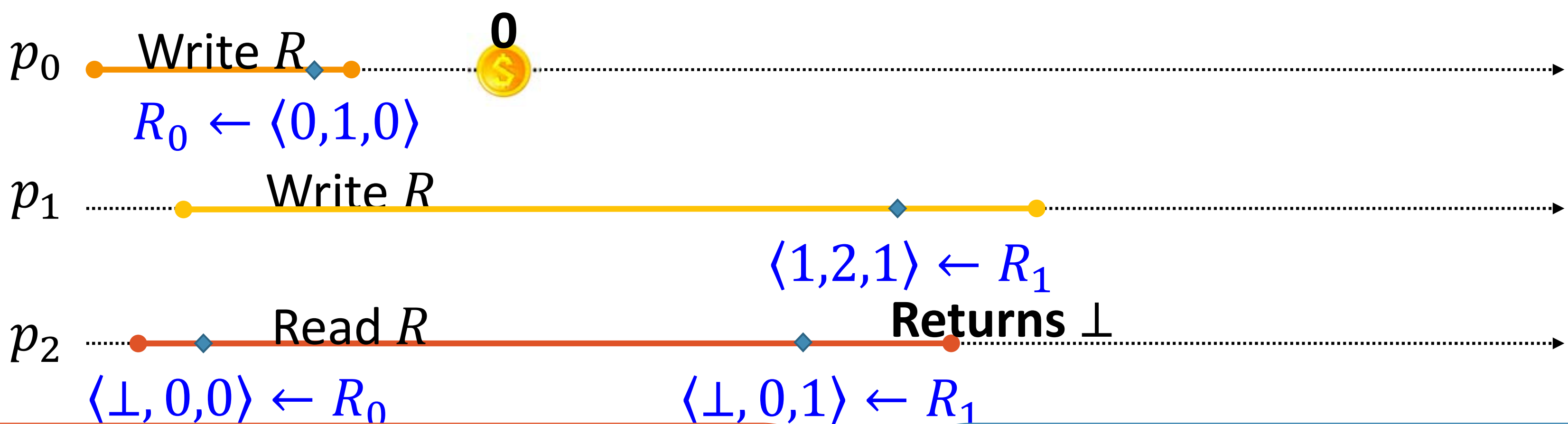
if $(c = 0 \wedge r = \perp) \vee (c = 1 \wedge r \neq \perp)$

then loop forever

else terminate

A strong adversary can make p_2 always loop forever

Example w/ VA: Coin = 0



Write(v,R)

read TS_0,...,read TS_{n-1}

TS_i = max TS_j +1

write $\langle v, TS_i, i \rangle$ to R_i

Read(R)

read R_0,...,read R_{n-1}

return v_j with maximal <TS_j,j>

Initially $R = \perp$, $C = -1$

Code for p_0, p_1:

$R \leftarrow i$

if $i = 0$ then $C \leftarrow$ flip 0 or 1

Code for p_2:

$r \leftarrow R$; $c \leftarrow C$

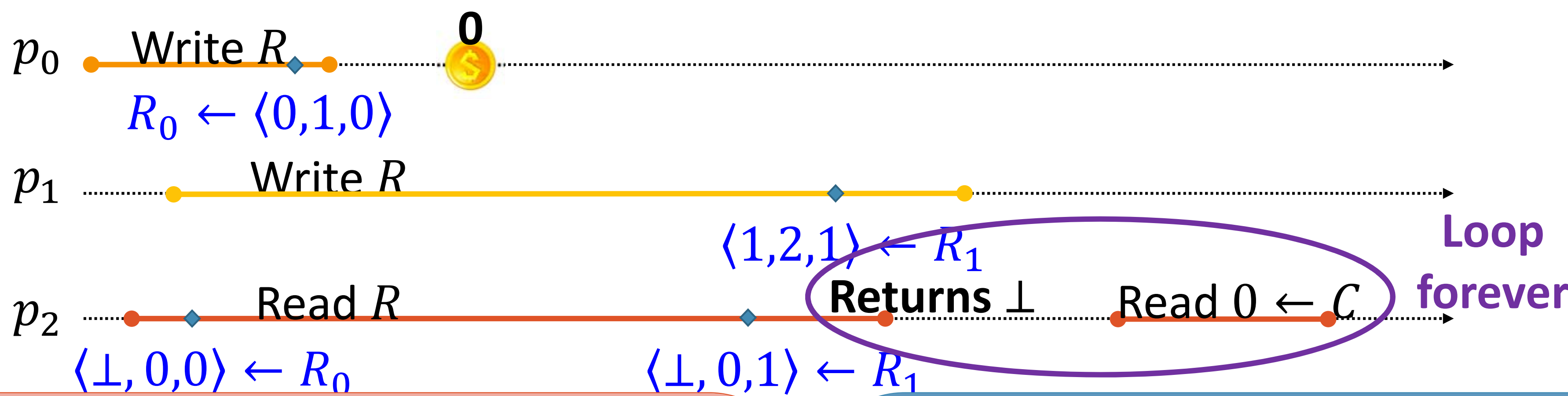
if $(c = 0 \wedge r = \perp) \vee (c = 1 \wedge r \neq \perp)$

then loop forever

else terminate

A strong adversary can make p_2 always loop forever

Example w/ VA: Coin = 0




```
Write(v,R)
read TS_0,...,read TS_n-1
TS_i = max TS_j +1
write ⟨v,TS_i,i⟩ to R_i

Read(R)
read R_0,...,read R_n-1
return v_j with maximal <TS_j,j>
```

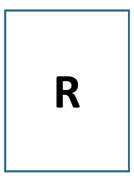
```
Initially R =⊥, C = −1
Code for p_0, p_1:
R ← i
if i = 0 then C ← flip 0 or 1

Code for p_2:
r ← R; c ← C
if (c = 0 ∧ r =⊥) ∨ (c = 1 ∧ r ≠⊥)
     then loop forever
else terminate
```

A strong adversary can make p_2 always loop forever

Another Example

When R is an atomic register,

p_2 terminates with probability > ½



Initially: R = ⊥, C = −1

Code for p_i , i = 0, 1:
R ← i
if (i == 1) then C ← flip fair coin (0 or 1)

Code for p_2:
u1 ← R; u2 ← R; c ← C
if ((u1 ≠ c) or (u2 ≠ 1 − c)) then loop forever
else terminate

[A, Enea, Welch, arxiv 2106.15554]
Distilled from [Hadzilacos, Hu, Toueg, PODC 2021]

Example w/ ABD

When R is implemented in message-passing,

a strong adversary can make p_2 always loop forever

Write(v)
bcast ⟨"query"⟩ and wait for > n/2 replies (v,t)
t = largest timestamp
bcast ⟨"update",v,t+1⟩ and wait for > n/2 acks

Read()
bcast ⟨"query"⟩ and wait for > n/2 replies
(v,t) = pair with largest timestamp
bcast ⟨"update",v,t⟩ and wait for > n/2 acks
return v

Initially: $R = \bot$, $C = -1$

Code for p_i , i = 0, 1:
$R \leftarrow i$
if (i == 1) then $C \leftarrow$ flip fair coin (0 or 1)

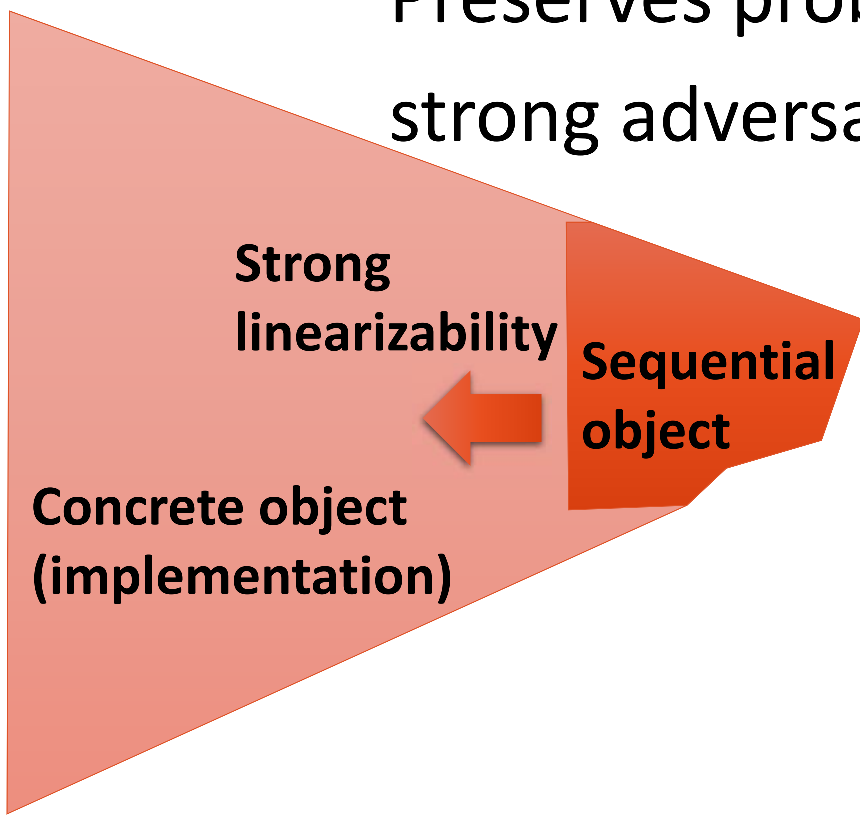
Code for p_2:
$u1 \leftarrow R$; $u2 \leftarrow R$; $c \leftarrow C$
if (($u1 \neq c$) or ($u2 \neq 1 - c$)) then loop forever
else terminate

[A, Enea, Welch, arxiv 2106.15554]
Distilled from [Hadzilacos, Hu, Toueg, PODC 2021]

Strong Linearizability

Linearization points are prefix preserving

Preserves probability distributions under strong adversaries



[Golab, Higham, Woelfel, STOC 2011]

More Generally, Strong Refinement

Obj $\leq_s$ Spec iff ∀ program P, ∀ deterministic scheduler S_1 of P X Obj,

∃ deterministic scheduler S_2 of P X Spec,

Traces(P X Obj X S_1) = Traces(P X Spec X S_2)

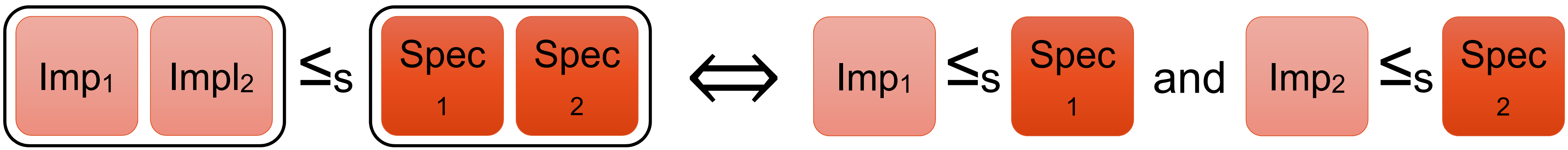
Preserves Hyperproperties

≡ Forward Simulation

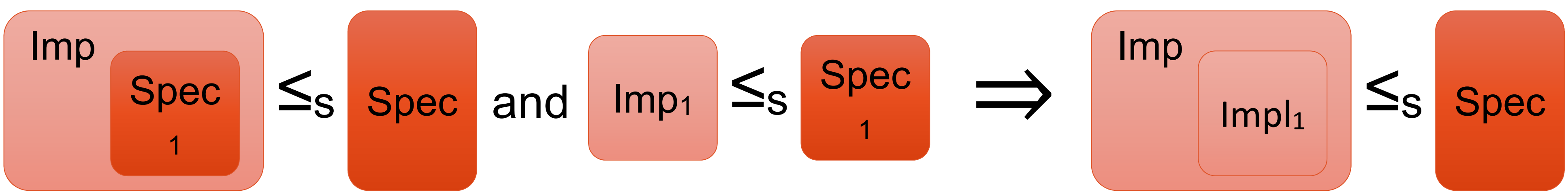
[A, Enea, DISC 2019]

⇒ Strong Refinement Composes

Locality (horizontal composition)



Parametrized objects (hierarchical composition)



Many Objects Don't Have Strongly Linearizable Implementations

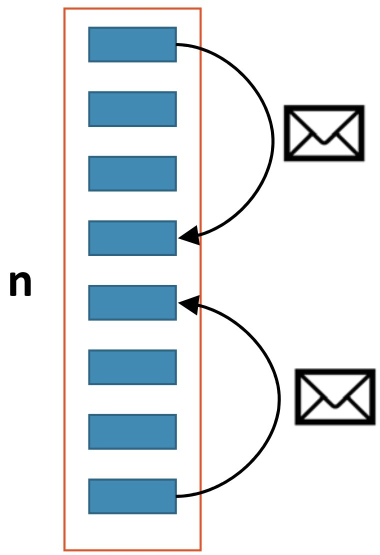
Counter-example ⇨ V&A MW register is not strongly linearizable

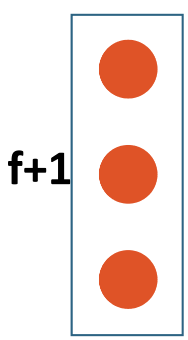
In fact, there is no wait-free strongly-linearizable MW register implementation from SW registers

Also, no wait-free strongly-linearizable snapshot implementation

[Helmi, Higham, Woelfel, PODC 2012]

No message-passing simulation of a register





Given a SL message-passing implementation
of a multi-writer multi-reader register
n processes, f << n possible failures

Obtain a SL shared-memory implementation
of a multi-writer multi-reader register,
using only single-writer multi-reader registers
f+1 processes, f possible failures
(strong simulation)

Which is impossible by [Helmi et al. 2012]

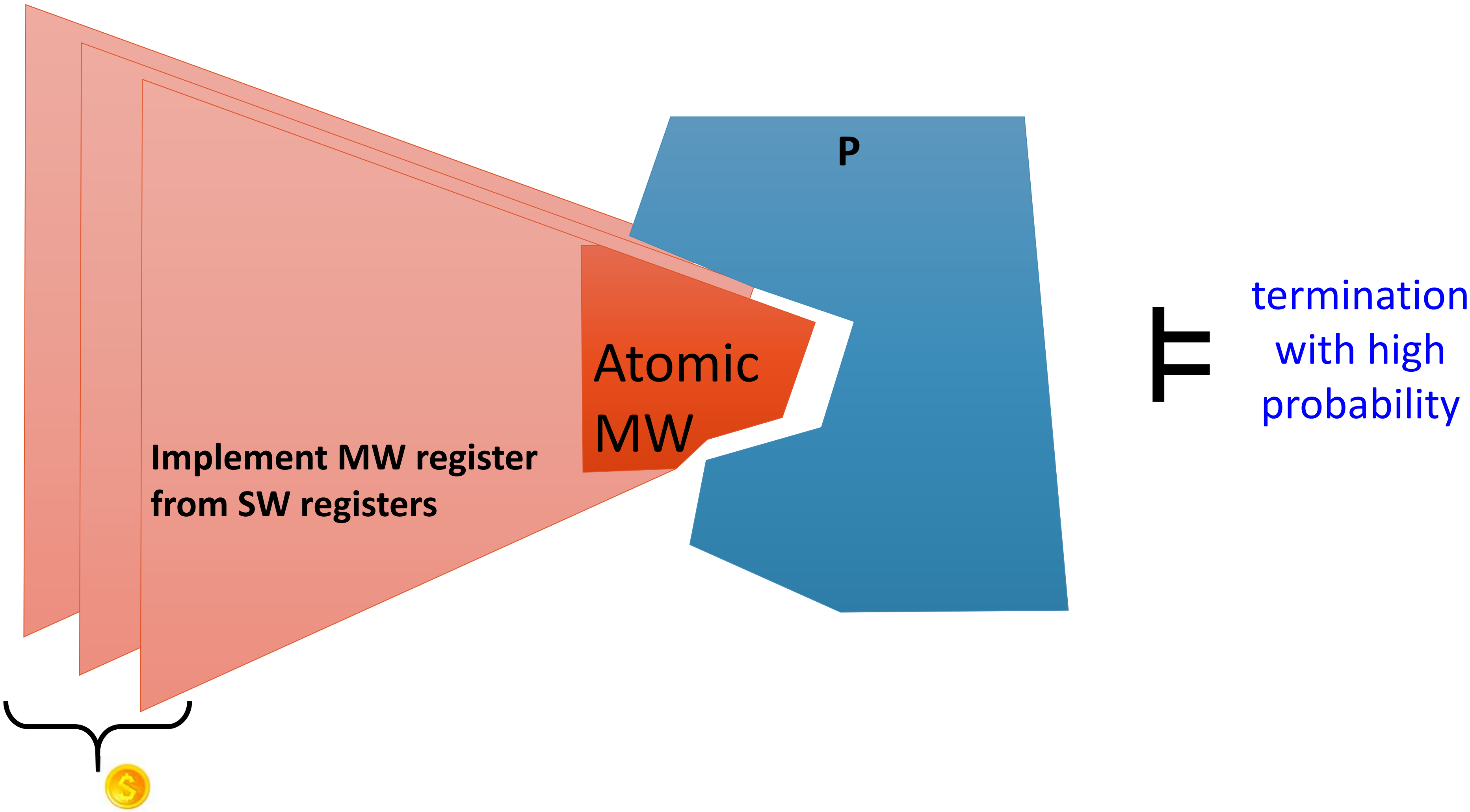
[A, Enea, Welch, DISC 2021]
A direct proof in [Chan, Hadzilacos, Hu, Toueg, arxiv 2108.01651]

Take inspiration from program indistinguishable obfuscation [Barak et al. 2012] and oblivious RAMs [Goldreich and Ostrovsky, 1996]

Perturb the concrete object to blunt the adversary,
while keeping its functionality indistinguishable

Use randomization…

Use Randomization to Blunt the Adversary



E.g., Blunting w/ One Coin Flip

p_2 terminates with constant probability with VA^2

```
Write(v,X)
read TS_0,...,read TS_n-1
TS_i = max TS_j +1
write ⟨v,TS_i,i⟩ to R_i
Read(X)
read R_0,...,read R_n-1
v1 = v_j with maximal <TS_j,j>
read R_0,...,read R_n-1
v2 = v_j with maximal <TS_j,j>
return v1 or v2 with prob. ½
```

```
Initially R = ⊥, C = −1
Code for p_0, p_1:
R ← i
if i = 0 then C ← flip 0 or 1

Code for p_2:
r ← R; c ← C
if (c = 0 ∧ r = ⊥) ∨ (c = 1 ∧ r ≠ ⊥)
     then loop forever
else terminate
```

Tail Strong Linearizability

Identify a preamble of the operation, after which, it is

mapped in a prefix-preserving manner (“strongly linearizable”)

```
Write(v,X)
read TS_0,...,read TS_n-1
TS_i = max TS_j +1
write ⟨v,TS_i,i⟩ to R_i
Read(X)
read R_0,...,read R_n-1
v = v_j with maximal <TS_j,j>


return v
```

Effect-free preamble doesn’t impact concurrently-running processes

Blunting

Tail strongly linearizable objects with an effect-free preamble

- Repeat the preamble k times
- Randomly pick one of the iterations to continue with

```
Write(v,X)
read TS0,...,read TSn-1
TSi = max TSj +1
write ⟨v,TSi,i⟩ to Ri
Read(X)
read R0,...,read Rn-1
v1 = vj with maximal <TSj,j>
read R0,...,read Rn-1
v2 = vj with maximal <TSj,j>
return v1 or v2 with prob. ½
```

Blunting, Specifically

Tail strongly linearizable objects with a read-only preamble, e.g.,

- Multi-reader registers from single-reader registers [Israeli, Li 1993]
- Multi-writer registers from single-writer registers [Vitanyi, Awerbuch 1986]
- ABD, Snapshots [Afek et al.]

For an n-process program P with r coin flips, using tail-strongly-linearizable objects O with effect-free preambles & any $k \geq r$,

$$\Pr[\, O^k] \leq \Pr[O_a] + (\Pr[O] - \Pr[\, O_a])\left(1 - \left(\frac{k-r}{k}\right)^{n-1}\right)$$

probability of a **bad** outcome B when P uses k-preamble-iterated versions of objects in O

probability of B when P uses atomic versions of objects in O

probability of B when P uses objects in O

E.g., in Our Example

p_2 terminates with probability $> {}^1/_8$ with VA^2

and $> {}^2/_9$ with VA^3

$1 - \left(\frac{3-1}{3}\right)^2 = {}^5/_9$

Non-termination probability w/ VA^2

${}^1/_2$

$1 - {}^1/_2$

$1 - \left(\frac{2-1}{2}\right)^2 = {}^3/_4$

$$\Pr[\,O^k] \leq \Pr[O_a] + (\Pr[O] - \Pr[\,O_a])\left(1 - \left(\frac{k-r}{k}\right)^{n-1}\right)$$

probability of a **bad** outcome B when P uses k-preamble-iterated versions of objects in O

probability of B when P uses atomic versions of objects in O

probability of B when P uses objects in O

Let X be the event that all random choices in O^k objects return an iteration of the preamble that does NOT overlap any random step of the program, then

$$\Pr[O^k] = \Pr[O^k | X] \cdot \Pr[X] + \Pr[O^k | \neg X] \cdot (1 - \Pr[X])$$

$$\leq \Pr[O_a] \cdot \Pr[X]$$

when chosen preambles don't overlap any program random steps, O^k objects behave like atomic objects

when chosen preamble overlaps program random step, O^k objects are no worse than O objects

$$+ \Pr[O] \cdot (1 - \Pr[X])$$

Since $\Pr[X] \geq \left(\frac{k-r}{k}\right)^{n-1}$, rearrangement gives that

$$\Pr[O^k] \leq \Pr[O_a] + (\Pr[O] - \Pr[O_a])\left(1 - \left(\frac{k-r}{k}\right)^{n-1}\right)$$

- $\Pr[O^k]$: probability of a **bad** outcome B when P uses k-preamble-iterated versions of objects in O
- $\Pr[O_a]$: probability of B when P uses atomic versions of objects in O
- $\Pr[O]$: probability of B when P uses objects in O

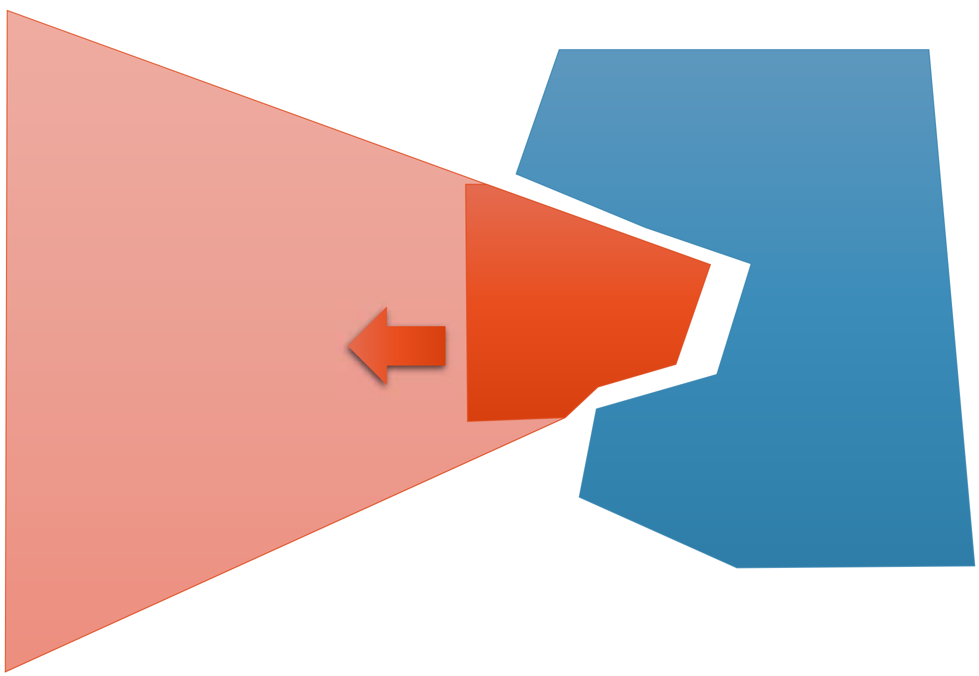
Wrap-Up

Write strong linearizability [Hadzilacos, Hu, Toueg, PODC 2021]
does not help with our example

Tradeoff between # iterations and decreased prob. of bad outcome
Reduce # random steps considered in the analysis, based on program structure (e.g., communication-closed layers)

Object implementations
w/o effect-free preambles

Transactions &
Cryptographic protocols



References

- Attiya, Enea, Welch: Blunting an Adversary Against Randomized Concurrent Programs with Linearizable Implementations. PODC 2022
- Attiya, Enea, Welch: Impossibility of Strongly-Linearizable Message-Passing Objects via Simulation by Single-Writer Registers. DISC 2021
- Attiya, Enea: Putting Strong Linearizability in Context: Preserving Hyperproperties in Programs that Use Concurrent Objects. DISC 2019