GPS:

*Navigating weak memory with **g**hosts, **p**rotocols, and **s**eparation*

Aaron Turon
Viktor Vafeiadis
Derek Dreyer
MPI-SWS

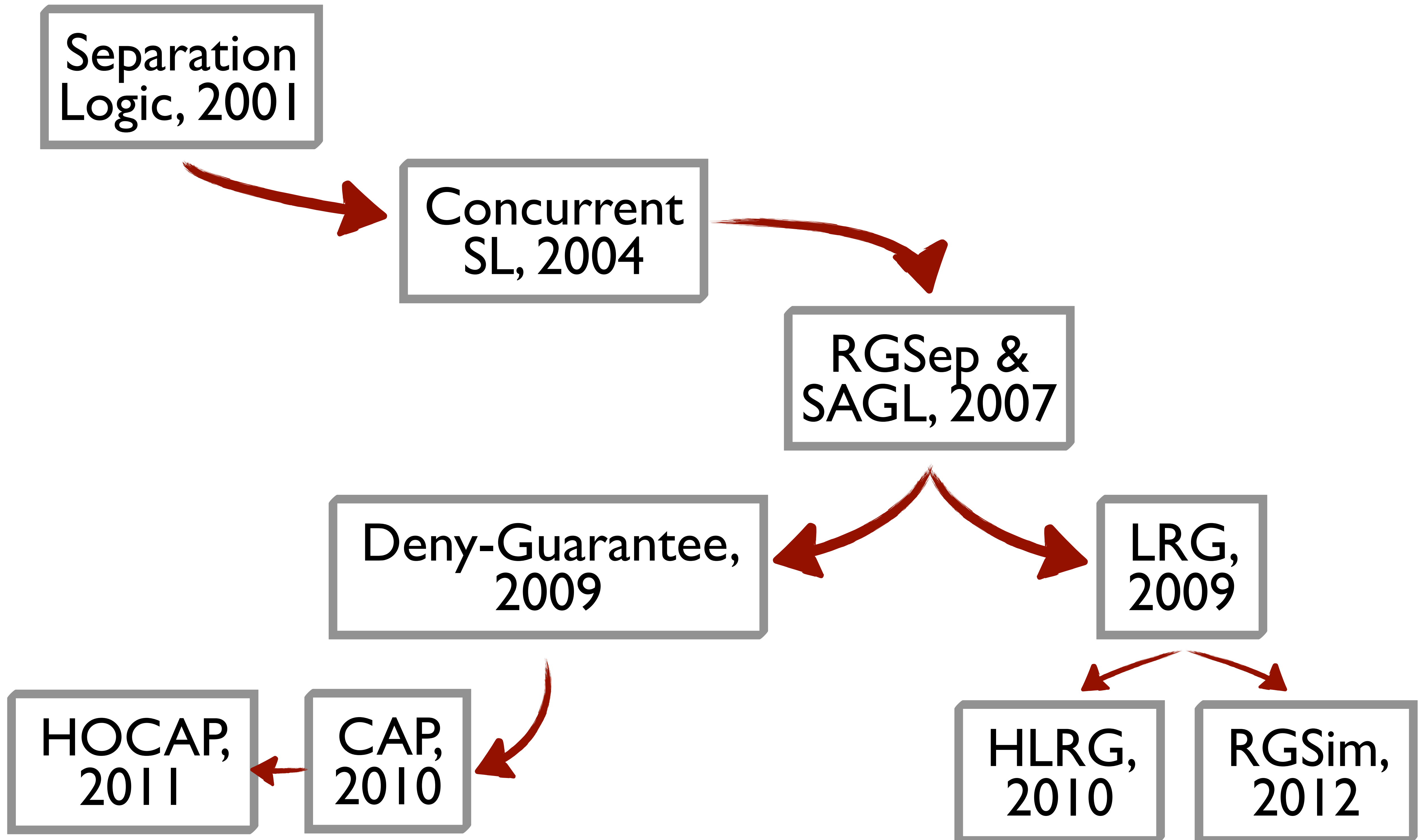
GPS:

To appear in OOPSLA'14
(don't shoot me)

*Navigating weak memory with **g**hosts, **p**rotocols, and **s**eparation*

Aaron Turon
Viktor Vafeiadis
Derek Dreyer
MPI-SWS

Concurrent Program Logics



Concurrent Program Logics

Separation Logic, 2001

Geared toward reasoning about racy, high-performance (e.g. lock-free) concurrent data structures

Deny-Guarantee, 2009

LRG, 2009

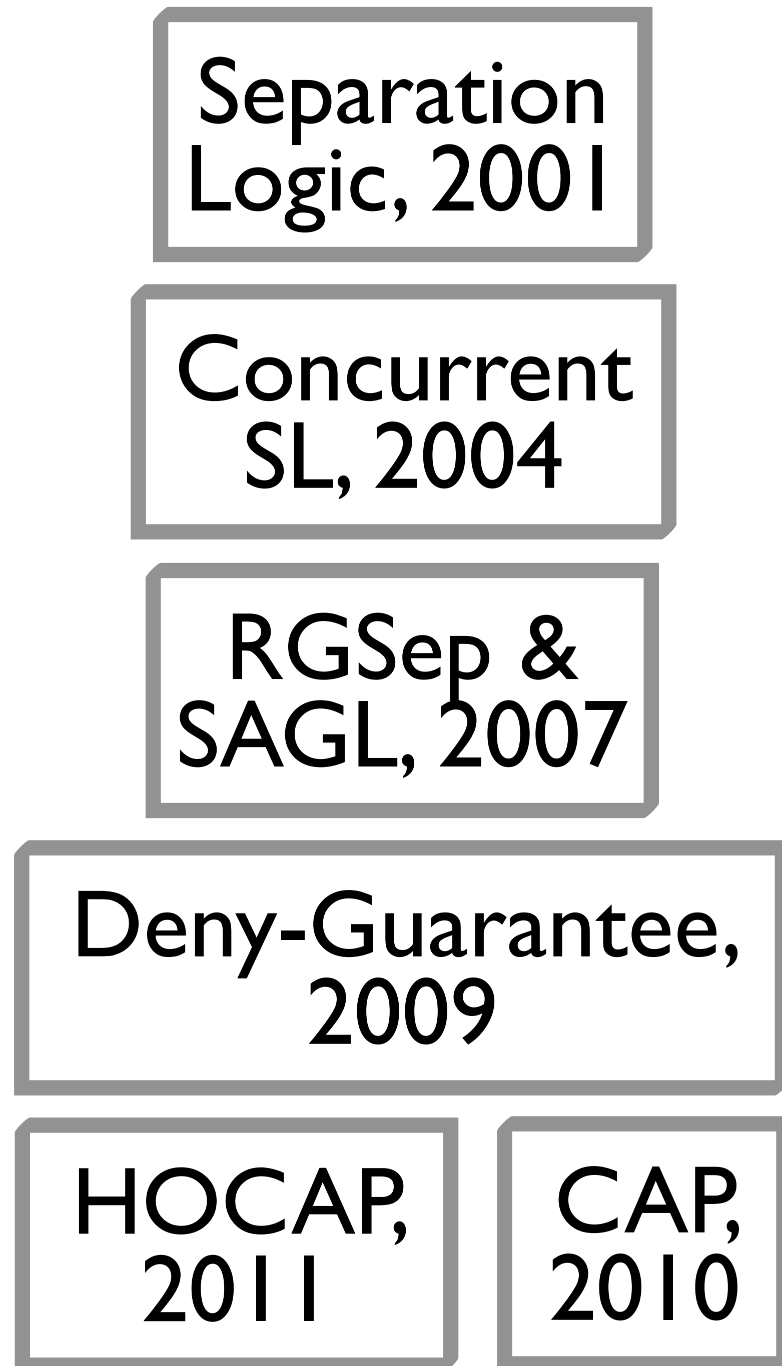
HOCAP, 2011

CAP, 2010

HLRG, 2010

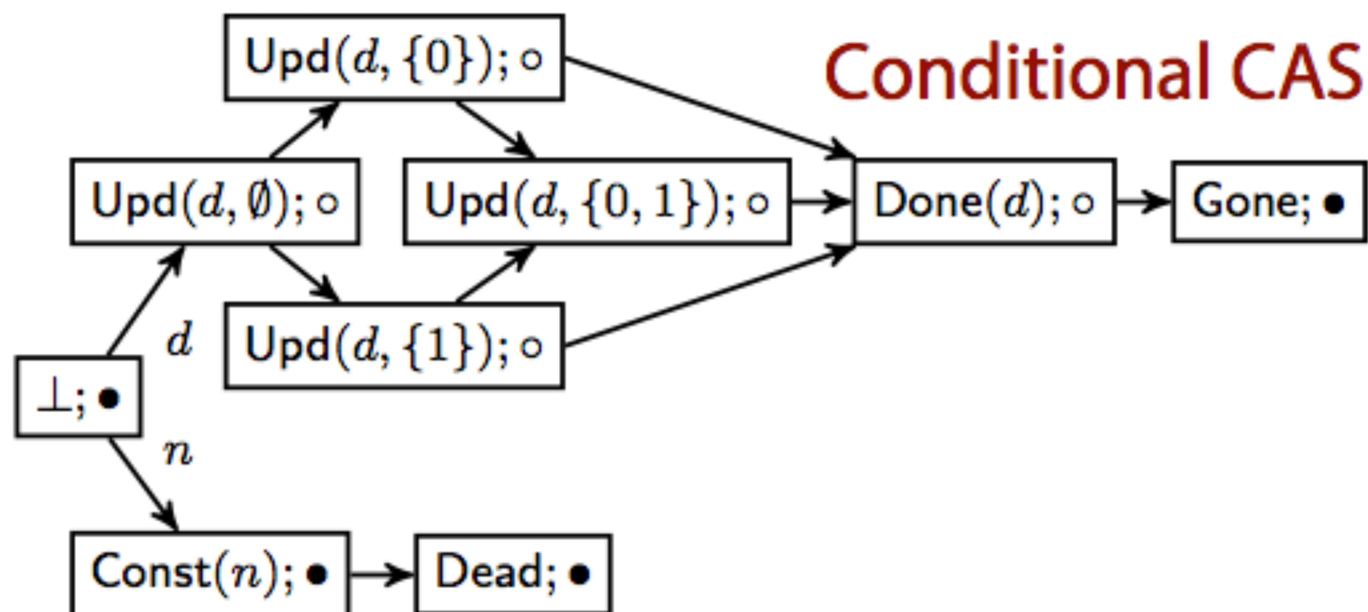
RGSim, 2012

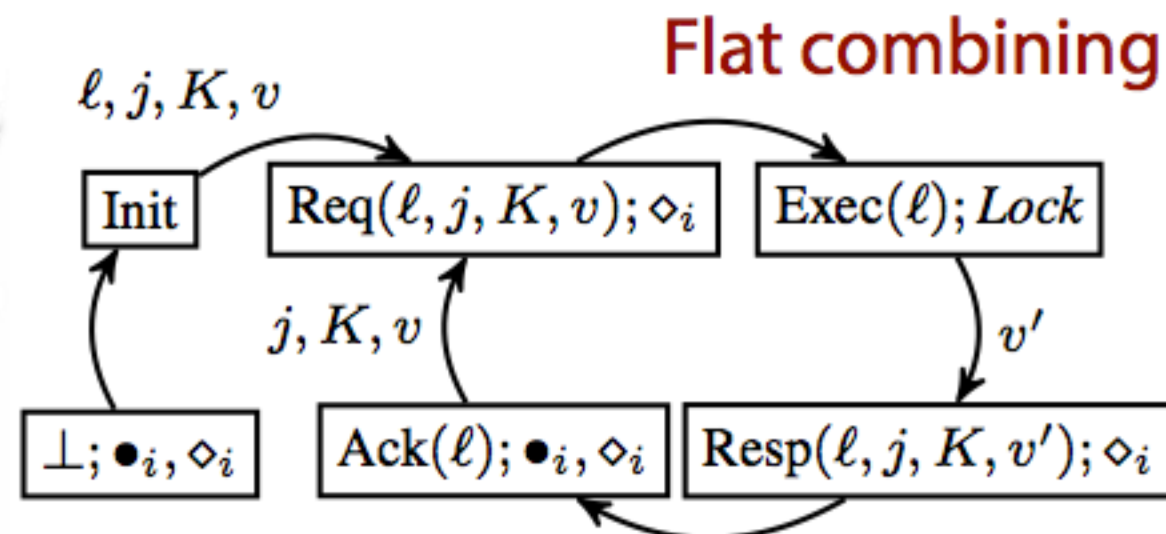
Concurrent Program Logics



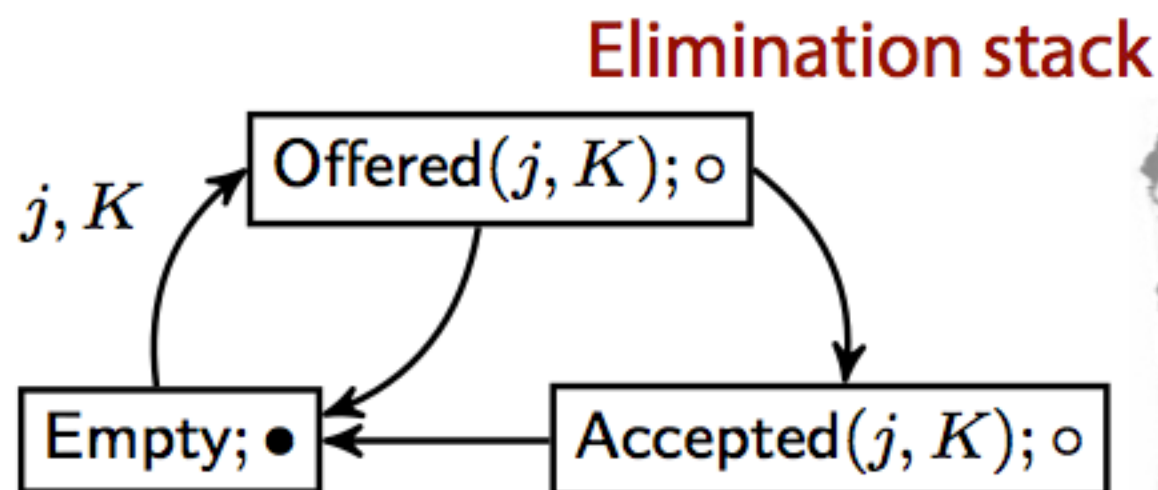
Increasing Modularity:

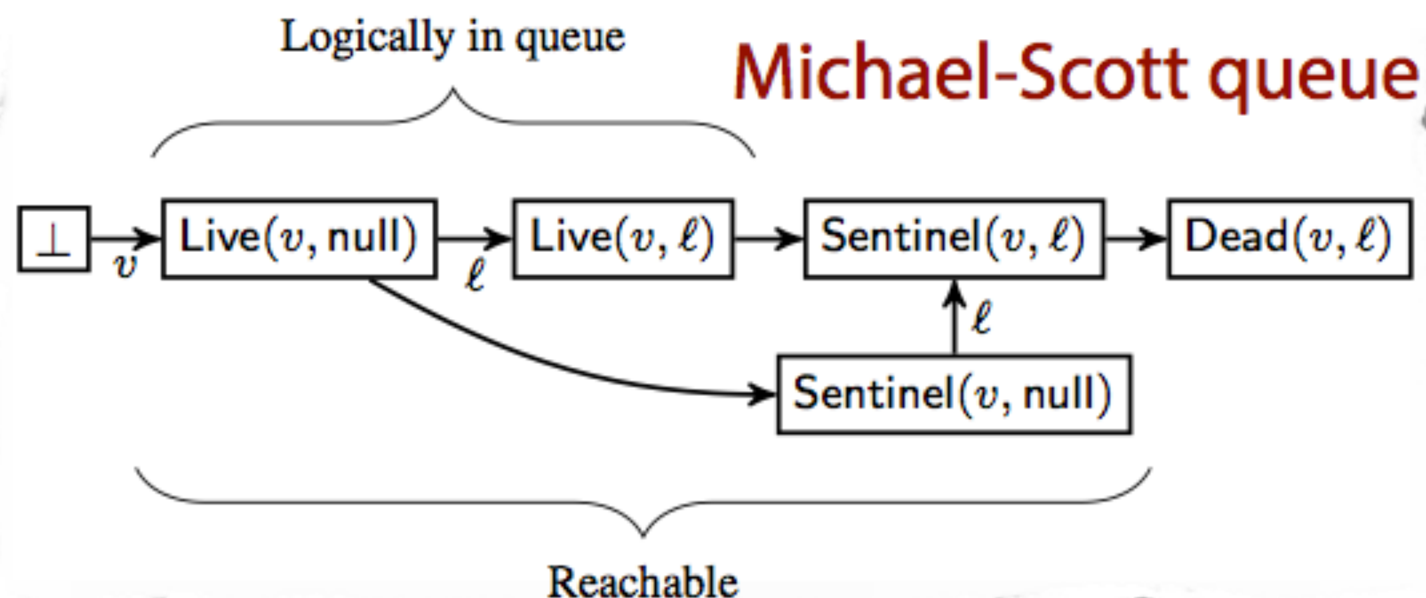
Ownership & separation
Rely/guarantee
Hidden state
etc.





CaReSL's "protocols" [Turon et al. 2013] concisely characterize the dynamically evolving interference on some piece of shared state.





The Problem

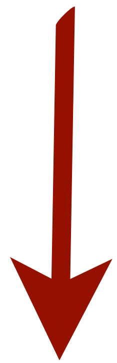
All these logics assume:

- **unrestricted data races**
- **sequential consistency (SC)**

This is not a realistic model of concurrency for high-performance algorithms!

```
x := 1
y := 2
x := 3
```

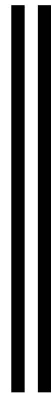
```
x := 1
y := 2
x := 3
```



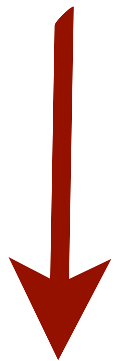
~~x := 1~~

```
y := 2
x := 3
```

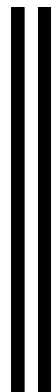
```
x := 1
y := 2
x := 3
```



```
print y
print x
```

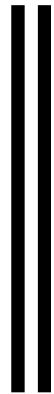


```
~~x := 1~~
y := 2
x := 3
```

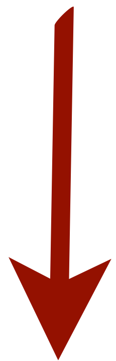


```
print y
print x
```

```
x := 1
y := 2
x := 3
```

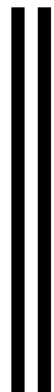


```
print y
print x
```



~~x := 1~~

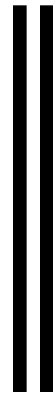
```
y := 2
x := 3
```



```
print y
print x
```

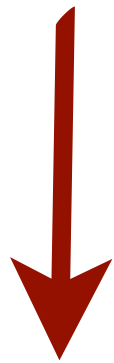
$\longrightarrow$ 2 0

```
x := 1
y := 2
x := 3
```



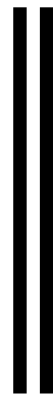
```
print y
print x
```

$\not\longrightarrow$ 2 0



~~x := 1~~

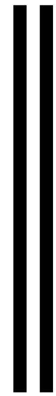
```
y := 2
x := 3
```



```
print y
print x
```

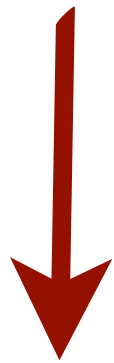
$\longrightarrow$ 2 0

```
x := 1
y := 2
x := 3
```



```
print y
print x
```

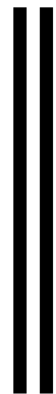
$\not\longrightarrow$ 2 0



Unsound!

~~x := 1~~

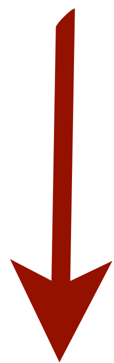
```
y := 2
x := 3
```



```
print y
print x
```

$\longrightarrow$ 2 0

```
x := 1
y := 2
x := 3
```



```
x := 1   (struck through)
y := 2
x := 3
```

Option #1: Prohibit data races

- 😎 Makes sense for most code
- 😎 Validates seq. optimizations
- 😢 Doesn't help if we want to be racy for high performance

```
x := 1        ||    print y
y := 2        ||    print x
x := 3        ||    ↛ 2 0
```

Option #2: Allow data races for certain variables

😎 Good for hi-perf concurrent algorithms

😢 SC semantics is too expensive to implement on modern hardware (must add fences)

```
x := 1        ║    print y
y := 2        ║    print x
x := 3        ║    → 2 0
```

Option #2: Allow data races for certain variables

😎 Good for hi-perf concurrent algorithms

😢 SC semantics is too expensive to implement on modern hardware (must add fences)

😳 Unnecessary in many cases: better to let experts use WEAKER consistency semantics

C11 Memory Model

Nonatomics (Data)

- Intended for most users (data race-free)
- If no data races exist, they behave as SC
- If data races exist, all bets are off
- Validate standard sequential optimizations

Atomics (Synchronization)

- Intended for experts (data races permitted)
- Several consistency levels:
  - SC
  - Release/acquire
  - Release/consume
  - Relaxed
- Weaker consistency => More reordering permitted by compilers and hardware

Uh Oh...

$$\nexists x.\ \mathsf{hb}(x,x) \qquad \text{(IrreflexiveHB)}$$

$$\forall \ell.\ \ \mathsf{totalorder}(\{a \in \mathcal{A} \mid \mathsf{iswrite}_\ell(a)\}, \mathsf{mo}) \land \mathsf{hb}_\ell \subseteq \mathsf{mo} \qquad \text{(ConsistentMO)}$$

$$\mathsf{totalorder}(\{a \in \mathcal{A} \mid \mathsf{isSeqCst}(a)\}, \mathsf{sc}) \land \mathsf{hb}_{\mathsf{SeqCst}} \subseteq \mathsf{sc} \land \mathsf{mo}_{\mathsf{SeqCst}} \subseteq \mathsf{sc} \qquad \text{(ConsistentSC)}$$

$$\forall b.\ \ \mathsf{rf}(b) \neq \bot \iff \exists \ell, a.\ \mathsf{iswrite}_\ell(a) \land \mathsf{isread}_\ell(b) \land \mathsf{hb}(a,b) \qquad \text{(ConsistentRFdom)}$$

$$\forall a, b.\ \ \mathsf{rf}(b) = a \implies \exists \ell, v.\ \mathsf{iswrite}_{\ell,v}(a) \land \mathsf{isread}_{\ell,v}(b) \land \neg\mathsf{hb}(b,a) \qquad \text{(ConsistentRF)}$$

$$\forall a, b.\ \ \mathsf{rf}(b) = a \land (\mathsf{mode}(a) = \mathrm{na} \lor \mathsf{mode}(b) = \mathrm{na}) \implies \mathsf{hb}(a,b) \qquad \text{(ConsistentRFna)}$$

$$\forall a, b.\ \ \mathsf{rf}(b) = a \land \mathsf{isSeqCst}(b) \implies \mathsf{isc}(a,b) \lor \neg\mathsf{isSeqCst}(a) \land (\forall x.\ \mathsf{isc}(x,b) \Rightarrow \neg\mathsf{hb}(a,x)) \qquad \text{(RestrSCReads)}$$

$$\nexists a, b.\ \ \mathsf{hb}(a,b) \land \mathsf{mo}(\mathsf{rf}(b), \mathsf{rf}(a)) \land \mathsf{locs}(a) = \mathsf{locs}(b) \qquad \text{(CoherentRR)}$$

$$\nexists a, b.\ \ \mathsf{hb}(a,b) \land \mathsf{mo}(\mathsf{rf}(b), a) \land \mathsf{iswrite}(a) \land \mathsf{locs}(a) = \mathsf{locs}(b) \qquad \text{(CoherentWR)}$$

$$\nexists a, b.\ \ \mathsf{hb}(a,b) \land \mathsf{mo}(b, \mathsf{rf}(a)) \land \mathsf{iswrite}(b) \land \mathsf{locs}(a) = \mathsf{locs}(b) \qquad \text{(CoherentRW)}$$

$$\forall a.\ \ \mathsf{isrmw}(a) \land \mathsf{rf}(a) \neq \bot \implies \mathsf{mo}(\mathsf{rf}(a), a) \land \nexists c.\ \mathsf{mo}(\mathsf{rf}(a), c) \land \mathsf{mo}(c, a) \qquad \text{(AtomicRMW)}$$

$$\forall a, b, \ell.\ \ \mathsf{lab}(a) = \mathsf{lab}(b) = \mathrm{A}(\ell) \implies a = b \qquad \text{(ConsistentAlloc)}$$

where $\mathsf{iswrite}_{\ell,v}(a) \stackrel{\mathrm{def}}{=} \exists X, v_{\mathrm{old}}.\ \mathsf{lab}(a) \in \{\mathrm{W}_X(\ell,v), \mathrm{RMW}_X(\ell, v_{\mathrm{old}}, v)\}$ $\qquad \mathsf{iswrite}_\ell(a) \stackrel{\mathrm{def}}{=} \exists v.\ \mathsf{iswrite}_{\ell,v}(a)$

$\mathsf{isread}_{\ell,v}(a) \stackrel{\mathrm{def}}{=} \exists X, v_{\mathrm{new}}.\ \mathsf{lab}(a) \in \{\mathrm{R}_X(\ell,v), \mathrm{RMW}_X(\ell, v, v_{\mathrm{new}})\}$ $\qquad$ etc.

$\mathsf{rsElem}(a,b) \stackrel{\mathrm{def}}{=} \mathsf{sameThread}(a,b) \lor \mathsf{isrmw}(b)$

$\mathsf{rseq}(a) \stackrel{\mathrm{def}}{=} \{a\} \cup \{b \mid \mathsf{rsElem}(a,b) \land \mathsf{mo}(a,b) \land (\forall c.\ \mathsf{mo}(a,c) \land \mathsf{mo}(c,b) \Rightarrow \mathsf{rsElem}(a,c))\}$

$\mathsf{sw} \stackrel{\mathrm{def}}{=} \{(a,b) \mid \mathsf{mode}(a) \in \{\mathrm{rel}, \mathrm{rel_acq}, \mathrm{sc}\} \land \mathsf{mode}(b) \in \{\mathrm{acq}, \mathrm{rel_acq}, \mathrm{sc}\} \land \mathsf{rf}(b) \in \mathsf{rseq}(a)\}$

$\mathsf{hb} \stackrel{\mathrm{def}}{=} (\mathsf{sb} \cup \mathsf{sw})^+$

$\mathsf{hb}_\ell \stackrel{\mathrm{def}}{=} \{(a,b) \in \mathsf{hb} \mid \mathsf{iswrite}_\ell(a) \land \mathsf{iswrite}_\ell(b)\}$

$X_{\mathsf{SeqCst}} \stackrel{\mathrm{def}}{=} \{(a,b) \in X \mid \mathsf{isSeqCst}(a) \land \mathsf{isSeqCst}(b)\}$

$\mathsf{isc}(a,b) \stackrel{\mathrm{def}}{=} \mathsf{iswrite}_{\mathsf{locs}(b)}(a) \land \mathsf{sc}(a,b) \land \nexists c.\ \mathsf{sc}(a,c) \land \mathsf{sc}(c,b) \land \mathsf{iswrite}_{\mathsf{locs}(b)}(c)$

Uh Oh…

$$\nexists x.\ \mathsf{hb}(x,x) \qquad \text{(IrreflexiveHB)}$$

$$\forall \ell.\ \ \mathsf{totalorder}(\{a \in \mathcal{A} \mid \mathsf{iswrite}_\ell(a)\}, \mathsf{mo}) \wedge \mathsf{hb}_\ell \subseteq \mathsf{mo} \qquad \text{(ConsistentMO)}$$

$$\mathsf{totalorder}(\{a \in \mathcal{A} \mid \mathsf{isSeqCst}(a)\}, \mathsf{sc}) \wedge \mathsf{hb}_{\mathsf{SeqCst}} \subseteq \mathsf{sc} \wedge \mathsf{mo}_{\mathsf{SeqCst}} \subseteq \mathsf{sc} \qquad \text{(ConsistentSC)}$$

$$\forall b.\ \ \mathsf{rf}(b) \neq \bot \iff \exists \ell, a.\ \mathsf{iswrite}_\ell(a) \wedge \mathsf{isread}_\ell(b) \wedge \mathsf{hb}(a,b) \qquad \text{(ConsistentRFdom)}$$

$$\forall a, b.\ \ \mathsf{rf}(b) = a \implies \exists \ell, v.\ \mathsf{iswrite}_{\ell,v}(a) \wedge \mathsf{isread}_{\ell,v}(b) \wedge \neg\mathsf{hb}(b,a) \qquad \text{(ConsistentRF)}$$

$\vee\ \mathsf{mode}(b)$

$\neg\mathsf{isSeqCs}$

$\mathsf{o}(\mathsf{rf}(b), \mathsf{rf}($

$(b), a) \wedge \mathsf{is}$

$\mathsf{rf}(a)) \wedge \mathsf{is}$

$\mathsf{f}(a), a) \wedge$

$$\forall a, b, \ell.\ \ \mathsf{lab}(a) = \mathsf{lab}(b) = \mathrm{A}(\ell) \implies a = b \qquad \text{(ConsistentAlloc)}$$

Event graph axioms are **global**, **subtle**, and **very low-level**

Want: **local** reasoning with **clear**, **high-level** specifications

$$\text{where } \mathsf{iswrite}_{\ell,v}(a) \stackrel{\text{def}}{=} \exists X, v_{\text{old}}.\ \mathsf{lab}(a) \in \{\mathrm{W}_X(\ell,v), \mathrm{RMW}_X(\ell, v_{\text{old}}, v)\} \qquad \mathsf{iswrite}_\ell(a) \stackrel{\text{def}}{=} \exists v.\ \mathsf{iswrite}_{\ell,v}(a)$$

$$\mathsf{isread}_{\ell,v}(a) \stackrel{\text{def}}{=} \exists X, v_{\text{new}}.\ \mathsf{lab}(a) \in \{\mathrm{R}_X(\ell,v), \mathrm{RMW}_X(\ell, v, v_{\text{new}})\} \qquad \text{etc.}$$

$$\mathsf{rsElem}(a,b) \stackrel{\text{def}}{=} \mathsf{sameThread}(a,b) \vee \mathsf{isrmw}(b)$$

$$\mathsf{rseq}(a) \stackrel{\text{def}}{=} \{a\} \cup \{b \mid \mathsf{rsElem}(a,b) \wedge \mathsf{mo}(a,b) \wedge (\forall c.\ \mathsf{mo}(a,c) \wedge \mathsf{mo}(c,b) \Rightarrow \mathsf{rsElem}(a,c))\}$$

$$\mathsf{sw} \stackrel{\text{def}}{=} \{(a,b) \mid \mathsf{mode}(a) \in \{\mathrm{rel}, \mathrm{rel_acq}, \mathrm{sc}\} \wedge \mathsf{mode}(b) \in \{\mathrm{acq}, \mathrm{rel_acq}, \mathrm{sc}\} \wedge \mathsf{rf}(b) \in \mathsf{rseq}(a)\}$$

$$\mathsf{hb} \stackrel{\text{def}}{=} (\mathsf{sb} \cup \mathsf{sw})^+$$

$$\mathsf{hb}_\ell \stackrel{\text{def}}{=} \{(a,b) \in \mathsf{hb} \mid \mathsf{iswrite}_\ell(a) \wedge \mathsf{iswrite}_\ell(b)\}$$

$$X_{\mathsf{SeqCst}} \stackrel{\text{def}}{=} \{(a,b) \in X \mid \mathsf{isSeqCst}(a) \wedge \mathsf{isSeqCst}(b)\}$$

$$\mathsf{isc}(a,b) \stackrel{\text{def}}{=} \mathsf{iswrite}_{\mathsf{locs}(b)}(a) \wedge \mathsf{sc}(a,b) \wedge \nexists c.\ \mathsf{sc}(a,c) \wedge \mathsf{sc}(c,b) \wedge \mathsf{iswrite}_{\mathsf{locs}(b)}(c)$$

Our Contribution

GPS: a “modern” separation logic
supporting a carefully restricted form of
- protocols
- ghost state
- ownership transfer

that is sound for the **C11 weak memory model**

Our major focus is showing how to reason about the ***release-acquire*** *consistency mode.*

Our Contribution

GPS: a "modern" separation logic
supporting a carefully restricted form of

th **model**

about

the ***release-acquire*** *consistency mode.*

Takeaway

Separation logic *can* make sense
of (a form of) weak memory!

Circular Buffers

(Linux kernel)

$$\{\mathsf{true}\}\ \ \texttt{newBuffer}()\ \ \{q.\ \mathsf{Prod}(q) * \mathsf{Cons}(q)\}$$
$$\{\mathsf{Prod}(q) * P(x)\}\ \texttt{tryProd}(q, x)\ \{z.\ \mathsf{Prod}(q) * (z \neq 0 \lor P(x))\}$$
$$\{\mathsf{Cons}(q)\}\ \ \texttt{tryCons}(q)\ \ \{x.\ \mathsf{Cons}(q) * (x = 0 \lor P(x))\}$$

$\texttt{newBuffer}() \triangleq$
$\quad \texttt{let}\ q = \texttt{alloc}(N + 2)$
$\quad [q + \texttt{ri}]_{\texttt{at}} := 0;$
$\quad [q + \texttt{wi}]_{\texttt{at}} := 0;$
$\quad q$

$\texttt{tryProd}(q, x) \triangleq$
$\quad \texttt{let}\ w = [q + \texttt{wi}]_{\texttt{at}}$
$\quad \texttt{let}\ r = [q + \texttt{ri}]_{\texttt{at}}$
$\quad \texttt{let}\ w' = w + 1\ \texttt{mod}\ N$
$\quad \texttt{if}\ w' == r\ \texttt{then}\ 0$
$\quad \texttt{else}\ [q + \texttt{b} + w]_{\texttt{na}} := x;$
$\qquad\quad [q + \texttt{wi}]_{\texttt{at}} := w';$
$\qquad\quad 1$

$\texttt{tryCons}(q) \triangleq$
$\quad \texttt{let}\ w = [q + \texttt{wi}]_{\texttt{at}}$
$\quad \texttt{let}\ r = [q + \texttt{ri}]_{\texttt{at}}$
$\quad \texttt{if}\ w == r\ \texttt{then}\ 0$
$\quad \texttt{else}\ \texttt{let}\ x = [q + \texttt{b} + r]_{\texttt{na}}$
$\qquad\quad [q + \texttt{ri}]_{\texttt{at}} := r + 1\ \texttt{mod}\ N;$
$\qquad\quad x$

Nonatomics

$$\{\mathtt{true}\}\,\mathtt{alloc}_{\mathtt{na}}(v)\,\{x.\, x \hookrightarrow v\}$$

$$\{\ell \hookrightarrow -\}\,[\ell]_{\mathtt{na}} \mathrel{\mathtt{:=}} v\,\{\ell \hookrightarrow v\}$$

$$\{\ell \hookrightarrow v\}\,[\ell]_{\mathtt{na}}\,\{x.\, x = v * \ell \hookrightarrow v\}$$

$$\frac{\{P_1\}\,e_1\,\{Q_1\} \qquad \{P_2\}\,e_2\,\{Q_2\}}{\{P_1 * P_2\}\,e_1 \parallel e_2\,\{Q_1 * Q_2\}} \qquad \frac{\{P\}\,e\,\{x.\, Q\}}{\{P * R\}\,e\,\{x.\, Q * R\}}$$

A Brief Introduction to
Release/Acquire

Dekker's Algorithm

$$
\begin{array}{l|l}
[x]_{\texttt{at}} \texttt{ := } 1 & [y]_{\texttt{at}} \texttt{ := } 1 \\
\texttt{if } [y]_{\texttt{at}} \texttt{ == } 0 \texttt{ then} & \texttt{if } [x]_{\texttt{at}} \texttt{ == } 0 \texttt{ then} \\
\quad \texttt{/*}\ \textit{crit. section}\ \texttt{*/} & \quad \texttt{/*}\ \textit{crit. section}\ \texttt{*/}
\end{array}
$$

Dekker's Algorithm

$[x]_{\texttt{at}}$ `:=` 1	$\|\|$	$[y]_{\texttt{at}}$ `:=` 1
`if` $[y]_{\texttt{at}}$ `==` 0 **`then`**		**`if`** $[x]_{\texttt{at}}$ `==` 0 **`then`**
/* *crit. section* */		/* *crit. section* */

Under SC, both threads can lose.

Dekker's Algorithm

$[x]_{\texttt{at}} \texttt{:=} 1$
 $\texttt{if}\ [y]_{\texttt{at}} \texttt{==} 0\ \texttt{then}$
     /* *crit. section* */
	$\|\|$	$[y]_{\texttt{at}} \texttt{:=} 1$
		 $\texttt{if}\ [x]_{\texttt{at}} \texttt{==} 0\ \texttt{then}$
		     /* *crit. section* */

Under SC, both threads can lose.

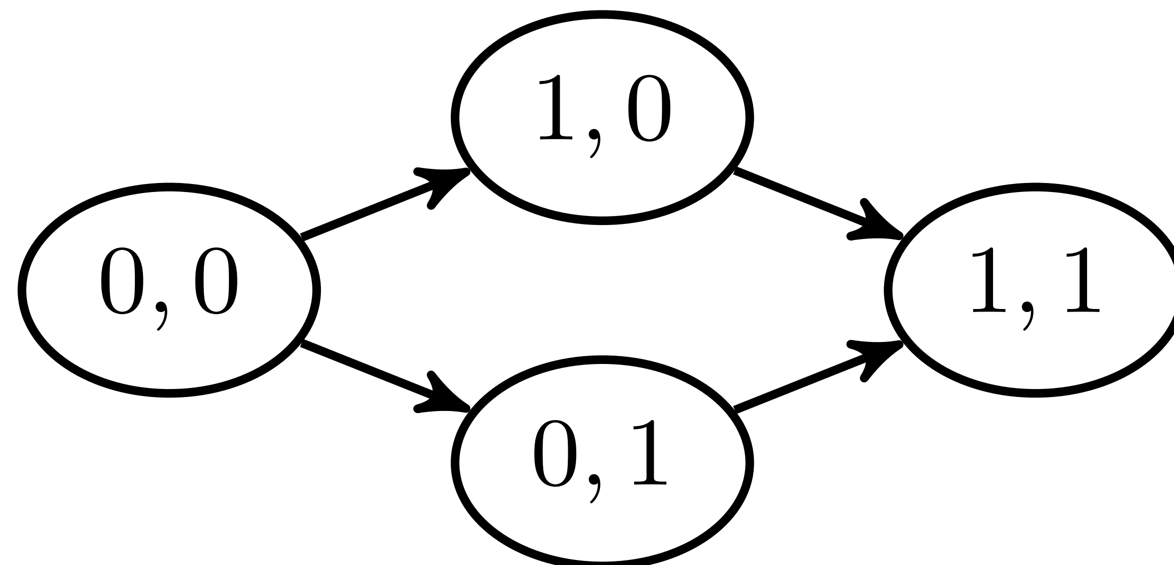
Under Rel/Acq, both threads can also *win*!

Dekker's Algorithm

$[x]_{\texttt{at}} \texttt{ := } 1$
$\texttt{if } [y]_{\texttt{at}} \texttt{ == } 0 \texttt{ then}$
    /* *crit. section* */

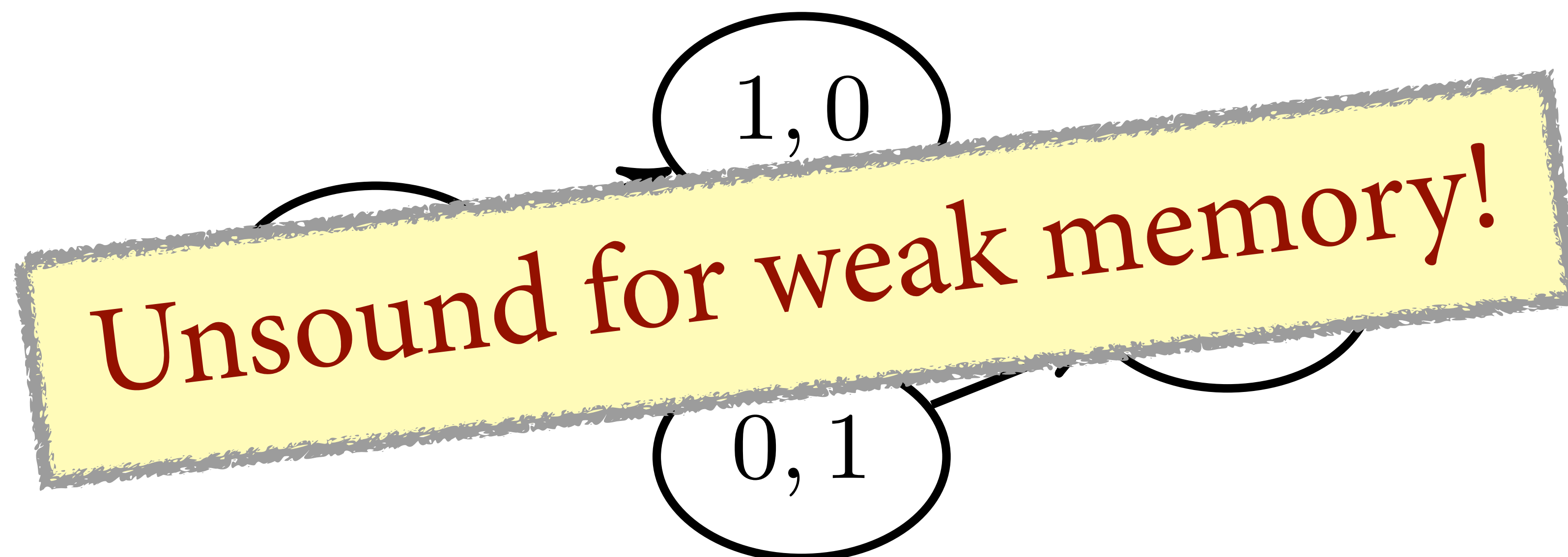
||

$[y]_{\texttt{at}} \texttt{ := } 1$
$\texttt{if } [x]_{\texttt{at}} \texttt{ == } 0 \texttt{ then}$
    /* *crit. section* */



Dekker's Algorithm

$[x]_{\texttt{at}}$ `:=` 1
`if` $[y]_{\texttt{at}}$ `==` 0 `then`
    /* *crit. section* */
	$\|$	$[y]_{\texttt{at}}$ `:=` 1
		`if` $[x]_{\texttt{at}}$ `==` 0 `then`
		    /* *crit. section* */



“IRIW”

$$[x]_{\texttt{at}} \texttt{ := } 1 \;\Big\|\; [y]_{\texttt{at}} \texttt{ := } 1 \;\Big\|\; \begin{array}{l}\texttt{print } [x]_{\texttt{at}} \\ \texttt{print } [y]_{\texttt{at}}\end{array} \;\Big\|\; \begin{array}{l}\texttt{print } [y]_{\texttt{at}} \\ \texttt{print } [x]_{\texttt{at}}\end{array}$$

Both threads can print 1, 0

No global total store ordering (unlike TSO)

Message Passing

$$
\begin{array}{l|l}
[x]_{\mathtt{na}} := 37; & \mathtt{repeat}\ [y]_{\mathtt{at}}\ \mathtt{end}; \\
[y]_{\mathtt{at}} := 1; & [x]_{\mathtt{na}}
\end{array}
$$

Message Passing

$$
\begin{array}{l|l}
[x]_{\texttt{na}} := 37; & \texttt{repeat } [y]_{\texttt{at}} \texttt{ end}; \\
[y]_{\texttt{at}} := 1; & [x]_{\texttt{na}}
\end{array}
$$

(If a thread sees a write,
it sees everything that
"*happened before*" that write)

Message Passing

$$\begin{array}{l} [x]_{\texttt{na}} := 37; \\ [y]_{\texttt{at}} := 1; \end{array} \;\Big\|\; \begin{array}{l} \texttt{repeat}\ [y]_{\texttt{at}}\ \texttt{end}; \\ [x]_{\texttt{na}} \end{array}$$

(If a thread sees a write, it sees everything that "*happened before*" that write)

Coherence

$$[x]_{\texttt{at}} \texttt{ := } 1 \;\Big\|\; [x]_{\texttt{at}} \texttt{ := } 2 \;\Big\|\; \begin{array}{l}\texttt{print } [x]_{\texttt{at}} \\ \texttt{print } [x]_{\texttt{at}}\end{array} \;\Big\|\; \begin{array}{l}\texttt{print } [x]_{\texttt{at}} \\ \texttt{print } [x]_{\texttt{at}}\end{array}$$

Cannot get 1, 2 and 2, 1

Total store ordering at each *independent* location

Coherence

$[x]_{\texttt{at}}$: … ‖ … ‖ print $[x]$… ‖ print $[x]_{\texttt{at}}$

…t $[x]_{\texttt{at}}$

… 2, 1

Tota… …tion

Key Idea

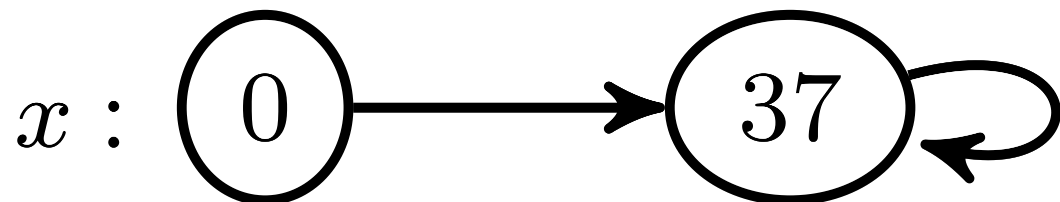
Leverage per-location *coherence*
for sound per-location *protocols*

Release/Acquire

$$\begin{array}{l} [x]_{\texttt{at}} \texttt{:=} 37; \\ [y]_{\texttt{at}} \texttt{:=} 1; \end{array} \;\Bigg\|\; \ldots \;\Bigg\|\; \begin{array}{l} [x]_{\texttt{at}} \texttt{:=} 37; \\ [y]_{\texttt{at}} \texttt{:=} 1; \end{array} \;\Bigg\|\; \begin{array}{l} \texttt{repeat } [y]_{\texttt{at}} \texttt{ end}; \\ [x]_{\texttt{at}} \end{array}$$

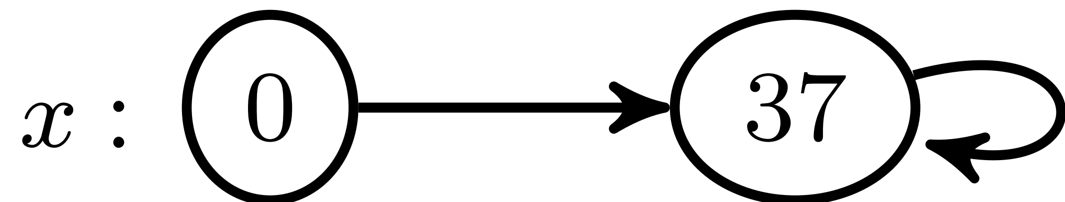
Release/Acquire

$$[x]_{\texttt{at}} \mathrel{\texttt{:=}} 37; \quad \| \quad \ldots \quad \| \quad [x]_{\texttt{at}} \mathrel{\texttt{:=}} 37; \quad \| \quad \texttt{repeat } [y]_{\texttt{at}} \texttt{ end};$$
$$[y]_{\texttt{at}} \mathrel{\texttt{:=}} 1; \quad \| \quad \quad \| \quad [y]_{\texttt{at}} \mathrel{\texttt{:=}} 1; \quad \| \quad [x]_{\texttt{at}}$$



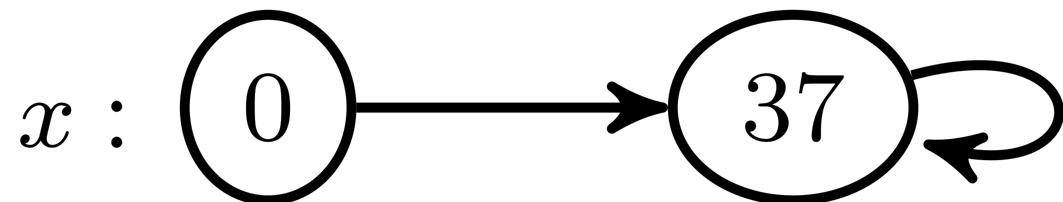
Release/Acquire

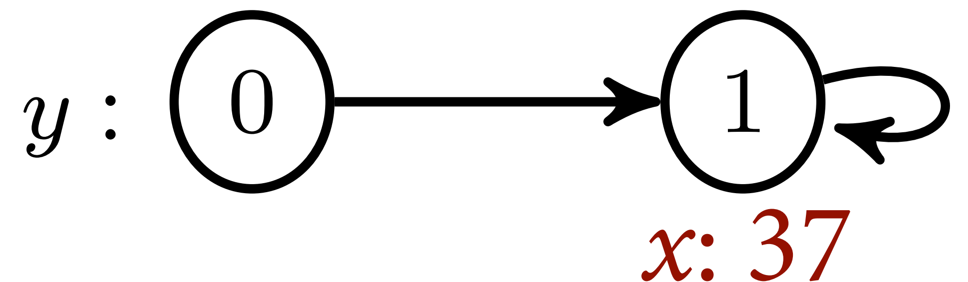
$$[x]_{\mathtt{at}} \mathrel{\mathtt{:=}} 37; \quad [y]_{\mathtt{at}} \mathrel{\mathtt{:=}} 1; \;\Big\|\; \ldots \;\Big\|\; [x]_{\mathtt{at}} \mathrel{\mathtt{:=}} 37; \quad [y]_{\mathtt{at}} \mathrel{\mathtt{:=}} 1; \;\Big\|\; \mathtt{repeat}\ [y]_{\mathtt{at}}\ \mathtt{end}; \quad [x]_{\mathtt{at}}$$



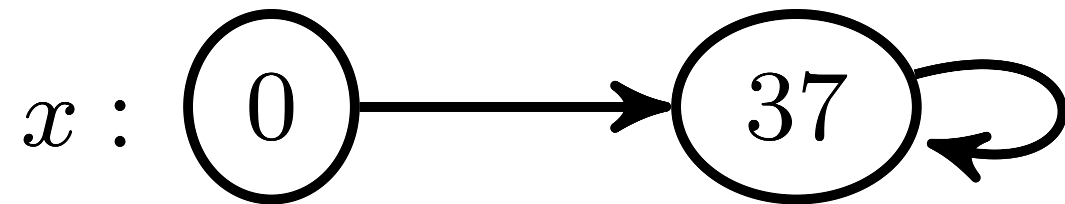
Release/Acquire

$[x]_{\texttt{at}}$:= 37; $[y]_{\texttt{at}}$:= 1; ‖ … ‖ $[x]_{\texttt{at}}$:= 37; $[y]_{\texttt{at}}$:= 1; ‖ **repeat** $[y]_{\texttt{at}}$ **end**; $[x]_{\texttt{at}}$

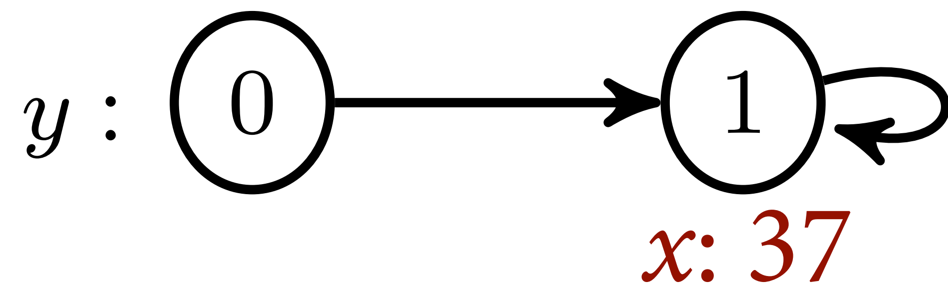




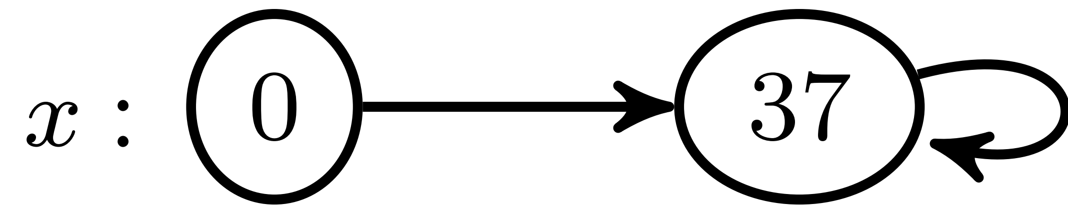
Protocol name: *data*



Protocol name: *flag*

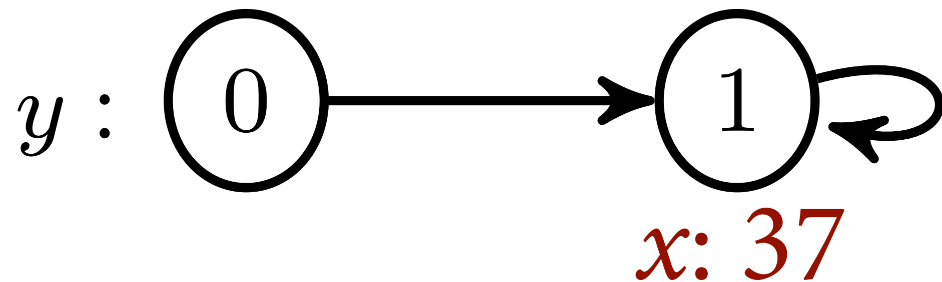


Protocol name: *data*

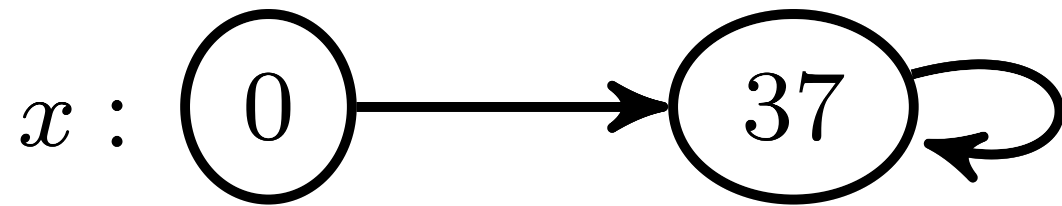


$$data(\mathbf{0}, z) \triangleq z = 0$$
$$data(\mathbf{37}, z) \triangleq z = 37$$

Protocol name: *flag*



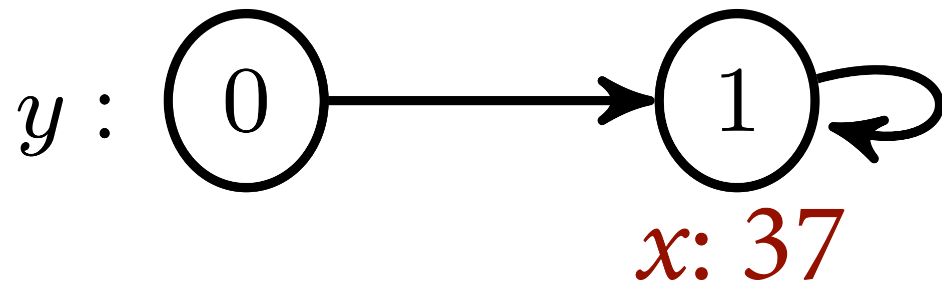
Protocol name: *data*



$data(\mathbf{0}, z) \triangleq z = 0$

$data(\mathbf{37}, z) \triangleq z = 37$

Protocol name: *flag*



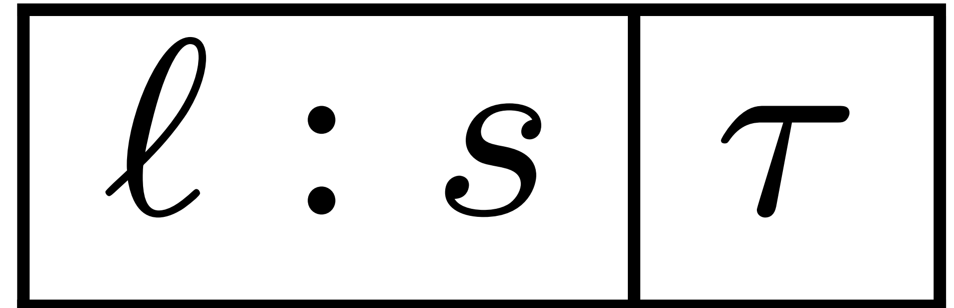
$flag(\mathbf{0}, z) \triangleq z = 0$

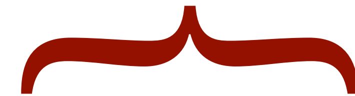
$flag(\mathbf{1}, z) \triangleq z = 1 \land$

x: **37**	*data*

Resources *vs.* Knowledge

$$\ell \hookrightarrow v$$





Lower bound;
No stability check!

Resources *vs.* Knowledge

$$t = t' \Rightarrow \square(t = t')$$

$$\boxed{\ell : s \mid \tau} \Rightarrow \square\, \boxed{\ell : s \mid \tau}$$

$$\square P \Rightarrow \square P * P$$

Operation	*Gain*	*Lose*
Read	Knowledge	-
Write	-	Resources
CAS: Success	Resources	Resources
CAS: Failure	Knowledge	-

Operation	***Gain***	***Lose***
Read	Knowledge	-
Write	-	Resources
CAS: Success	Resources	Resources
CAS: Failure	Knowledge	-

Acquire Reads

$$\frac{\forall s' \sqsupseteq_\tau s.\, \forall z.\, \tau(s', z) \Rightarrow \Box Q}{\left\{ \boxed{\ell : s \mid \tau} \right\} [\ell]_{\mathtt{at}} \left\{ z.\, \exists s'.\, \boxed{\ell : s' \mid \tau} * \Box Q \right\}}$$

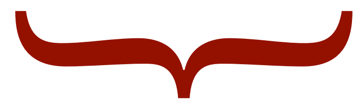
Acquire Reads

$$\frac{\forall s' \sqsupseteq_\tau s.\ \forall z.\ \tau(s', z) \Rightarrow \Box Q}{\left\{\boxed{\ell : s \mid \tau}\right\} [\ell]_{\texttt{at}} \left\{z.\ \exists s'.\ \boxed{\ell : s' \mid \tau} * \Box Q\right\}}$$

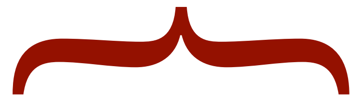
Lower bound

Acquire Reads

New state



$$\frac{\forall s' \sqsupseteq_{\tau} s.\ \forall z.\ \tau(s', z) \Rightarrow \Box Q}{\left\{ \boxed{\ell : s \mid \tau} \right\}\ [\ell]_{\mathtt{at}}\ \left\{ z.\ \exists s'.\ \boxed{\ell : s' \mid \tau} * \Box Q \right\}}$$



Lower bound

Acquire Reads

New state *Interpretation*

$$\frac{\forall s' \sqsupseteq_\tau s.\ \forall z.\ \tau(s', z) \Rightarrow \Box Q}{\left\{\boxed{\ell : s \mid \tau}\right\}\ [\ell]_{\mathtt{at}}\ \left\{z.\ \exists s'.\ \boxed{\ell : s' \mid \tau} * \Box Q\right\}}$$

Lower bound

Acquire Reads

$$\frac{\overbrace{\forall s' \sqsupseteq_\tau s}^{\textit{New state}}.\ \forall z.\ \overbrace{\tau(s', z)}^{\textit{Interpretation}} \Rightarrow \overbrace{\Box Q}^{\textit{Gained knowledge}}}{\left\{\underbrace{\boxed{\ell : s} \boxed{\tau}}_{\textit{Lower bound}}\right\} [\ell]_{\texttt{at}} \left\{z.\ \exists s'.\ \boxed{\ell : s'} \boxed{\tau} * \Box Q\right\}}$$

Acquire Reads

New state — *Interpretation* — *Gained knowledge*

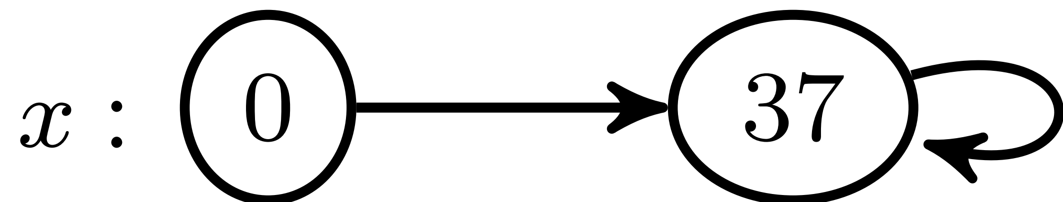
$$\forall s' \sqsupseteq_\tau s.\ \forall z.\ \tau(s', z) \Rightarrow \Box Q$$

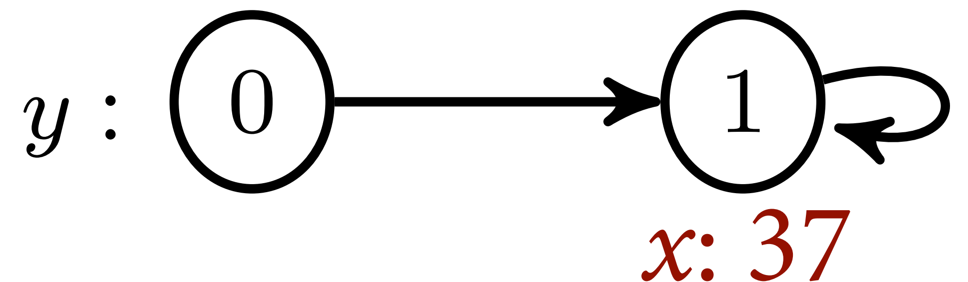
$$\left\{ \boxed{\ell : s \mid \tau} \right\} [\ell]_{\mathtt{at}} \left\{ z.\ \exists s'.\ \boxed{\ell : s' \mid \tau} * \Box Q \right\}$$

Lower bound — *New lower bound*

Release/Acquire

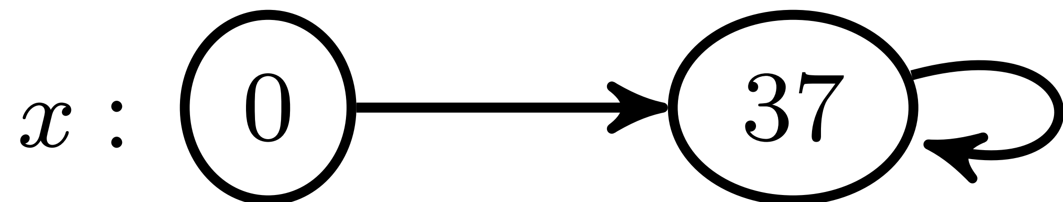
$$\begin{array}{l} [x]_{\texttt{at}} \texttt{:=} 37; \\ [y]_{\texttt{at}} \texttt{:=} 1; \end{array} \;\Big\|\; \ldots \;\Big\|\; \begin{array}{l} [x]_{\texttt{at}} \texttt{:=} 37; \\ [y]_{\texttt{at}} \texttt{:=} 1; \end{array} \;\Big\|\; \begin{array}{l} \texttt{repeat } [y]_{\texttt{at}} \texttt{ end}; \\ [x]_{\texttt{at}} \end{array}$$

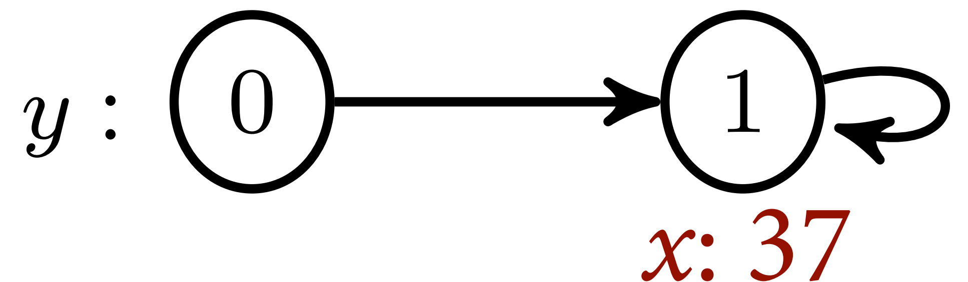


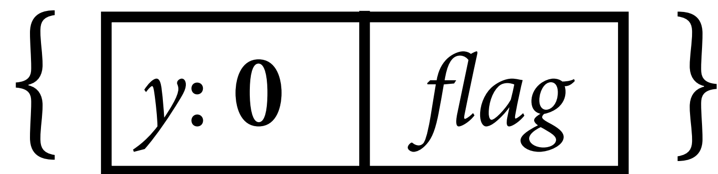


Release/Acquire

$$\begin{array}{l}[x]_{\mathtt{at}} := 37; \\ [y]_{\mathtt{at}} := 1;\end{array} \;\Bigg\|\; \ldots \;\Bigg\|\; \begin{array}{l}[x]_{\mathtt{at}} := 37; \\ [y]_{\mathtt{at}} := 1;\end{array} \;\Bigg\|\; \begin{array}{l}\mathtt{repeat}\ [y]_{\mathtt{at}}\ \mathtt{end}; \\ [x]_{\mathtt{at}}\end{array}$$





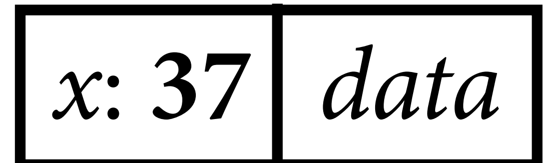
$\{$ 

$\}$

$[y]_{\mathsf{AT}}$

$flag(\mathbf{0}, z) \triangleq z = 0$

$flag(\mathbf{1}, z) \triangleq z = 1 \wedge$



$$flag(\mathbf{0}, z) \triangleq z = 0$$

$$flag(\mathbf{1}, z) \triangleq z = 1 \land \boxed{x\text{: } \mathbf{37} \mid data}$$

$$\{\ \boxed{y\text{: } \mathbf{0} \mid flag}\ \}$$

$$[y]_{\mathsf{AT}}$$

$$\{\ z.\ \ z = 0 * \boxed{y\text{: } \mathbf{0} \mid flag}$$

$$\lor\ z = 1 * \boxed{y\text{: } \mathbf{1} \mid flag} * \boxed{x\text{: } \mathbf{37} \mid data}\ \}$$

$$\{ \; \boxed{x\!: \mathbf{37}} \boxed{\mathit{data}} \; \}$$

$[x]_{\mathsf{AT}}$

$$data(\mathbf{0}, z) \triangleq z = 0$$
$$data(\mathbf{37}, z) \triangleq z = 37$$

$$data(\mathbf{0}, z) \triangleq z = 0$$
$$data(\mathbf{37}, z) \triangleq z = 37$$

$$\left\{ \boxed{x\colon \mathbf{37}} \boxed{data} \right\}$$

$[x]_{\mathsf{AT}}$

$$\left\{ z.\ z = 37 * \boxed{x\colon \mathbf{37}} \boxed{data} \right\}$$

Operation	*Gain*	*Lose*
Read	Knowledge	-
Write	-	Resources
CAS: Success	Resources	Resources
CAS: Failure	Knowledge	-

Ownership Transfer

$$\begin{array}{l} [x]_{\texttt{na}} := 37; \\ [y]_{\texttt{at}} := 1; \end{array} \quad \Bigg\| \quad \begin{array}{l} \texttt{repeat } [y]_{\texttt{at}} \texttt{ end}; \\ [x]_{\texttt{na}} \end{array}$$

How can we verify this ownership transfer if the read of *y* can only gain **knowledge**?

Ownership Transfer

$[x]_{\texttt{na}} := 37; \quad || \quad \texttt{repeat}\ [y]_{\texttt{at}}\ \texttt{end};$

$[y$

Key Idea

Allow implicit resource transfers through **escrows**

Ghost Resources

$$\frac{\{P\}\, e\, \{x.\, Q\}}{\{P\}\, e\, \{x.\, Q * \exists i.\, \mathrm{Token}(i)\}}$$

$$\mathrm{Token}(i) * \mathrm{Token}(i) \Rightarrow \mathsf{false}$$

Escrows

Creation

$$\frac{R * R \Rightarrow \mathsf{false} \qquad \{P\}\, e\, \{x.\, Q\}}{\{P\}\, e\, \{x.\, [R \rightsquigarrow Q]\}}$$

Escrows

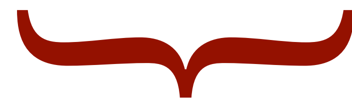
Condition

Creation

$$\frac{R * R \Rightarrow \mathsf{false} \qquad \{P\}\, e\, \{x.\, Q\}}{\{P\}\, e\, \{x.\, [R \rightsquigarrow Q]\}}$$

Escrows

Condition



Creation

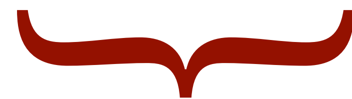
$$\frac{R * R \Rightarrow \mathsf{false} \qquad \{P\}\, e\, \{x.\ Q\}}{\{P\}\, e\, \{x.\ [R \rightsquigarrow Q]\}}$$

Fulfillment

$$\frac{\{P\}\, e\, \{x.\ Q\}}{\{R * [R \rightsquigarrow P]\}\, e\, \{x.\ Q\}}$$

Escrows

Condition



Creation

$$\frac{R * R \Rightarrow \mathsf{false} \qquad \{P\}\, e\, \{x.\, Q\}}{\{P\}\, e\, \{x.\, [R \rightsquigarrow Q]\}}$$

Fulfillment

$$\frac{\{P\}\, e\, \{x.\, Q\}}{\{R * [R \rightsquigarrow P]\}\, e\, \{x.\, Q\}}$$

Knowledge

$$[R \rightsquigarrow P] \Rightarrow \Box[R \rightsquigarrow P]$$

$$\mathbf{XE}(\gamma): \quad \boxed{\gamma : \diamond \mid \mathsf{Tok}} \quad \rightsquigarrow \quad x \hookrightarrow 37$$

$$\mathbf{YP}(\gamma)(0, z) \triangleq z = 0$$

$$\mathbf{YP}(\gamma)(1, z) \triangleq z = 1 * [\mathbf{XE}(\gamma)]$$

$$\left\{ x \hookrightarrow 0 * \boxed{y : 0 \mid \mathbf{YP}(\gamma)} \right\}$$
$$[x]_{\mathtt{na}} := 37;$$
$$\left\{ x \hookrightarrow 37 * \boxed{y : 0 \mid \mathbf{YP}(\gamma)} \right\}$$
$$\left\{ [\mathbf{XE}(\gamma)] * \boxed{y : 0 \mid \mathbf{YP}(\gamma)} \right\}$$
$$[y]_{\mathtt{at}} := 1;$$
$$\left\{ \boxed{y : 1 \mid \mathbf{YP}(\gamma)} \right\}$$

$$\left\{ \boxed{\gamma : \diamond} * \boxed{y : 0 \mid \mathbf{YP}(\gamma)} \right\}$$
$$\mathtt{repeat}\ [y]_{\mathtt{at}}\ \mathtt{end};$$
$$\left\{ \boxed{\gamma : \diamond} * \boxed{y : 1 \mid \mathbf{YP}(\gamma)} * [\mathbf{XE}(\gamma)] \right\}$$
$$\{ x \hookrightarrow 37 \}$$
$$[x]_{\mathtt{na}}$$
$$\{ z.\ z = 37 * x \hookrightarrow 37 \}$$

Soundness

Soundness

…ask Viktor!

Principles of the Model

- Label *hb* (happens-before) edges with resources/knowledge
- "Concurrent" edges $\Rightarrow$ compatible labels
- For each location, *mo* simulated by protocol

The Big Picture

What We've Done

- Extends Viktor's previous RSL logic
- Release-acquire protocols
- Ghost state and escrows
- Case studies:
    - Michael-Scott queue
    - Linux *bounded* ticket lock
    - Linux circular buffer
- Complete soundness proof in Coq!

What We've Done

- Extends Viktor's previous RSL logic
- Release-acquire protocols

Takeaway

Separation logic *can* make sense of (a form of) weak memory!

- Linux circular buffer
- Complete soundness proof in Coq!

What We Want to Do

- Full C11:
    - Add release/consume
    - Add relaxed *reads*
    - Relaxed *writes* appear broken
- Refinement reasoning
- Weak data structure specifications