Based on joint work with Ohad Kammar, Ori Lahav, and Gordon Plotkin:

MONADIC AND ALGEBRAIC DENOTATIONAL SEMANTICS FOR CONCURRENT SHARED STATE

TARTU UNIVERSITY

Yotam Dvir

2025

Brookes's Denotational Semantics
for Shared State Concurrency

Algebraic Effects
Refinement

Relaxed Memory
Extension

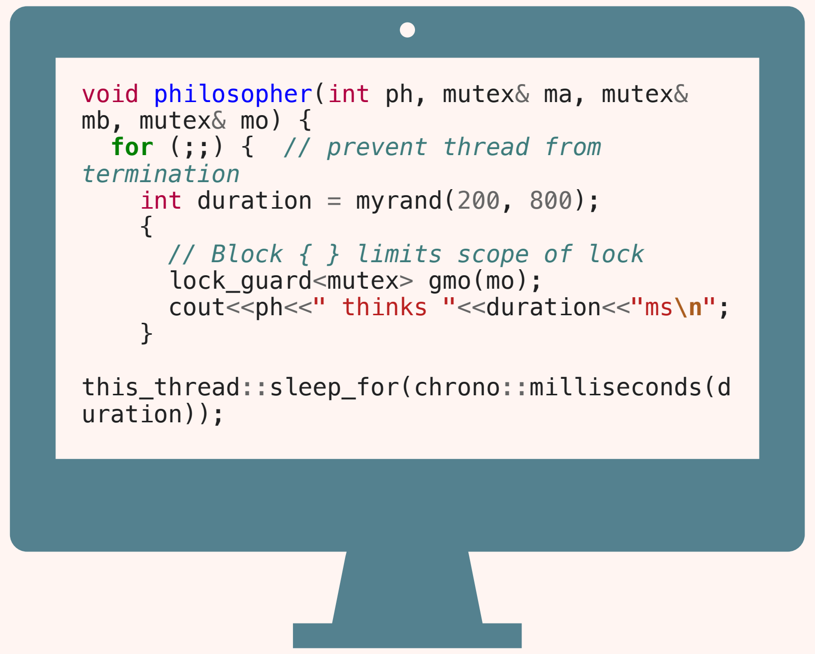
SEQUENTIAL SETTING

SMALL-STEP SEMANTICS

$$\begin{aligned}
&\sigma, (l := 0 \; ; \; \textbf{ifz } l? \textbf{ then } \text{“ok” } \textbf{else } \text{“bug”}) \\
\rightarrow\; &\sigma[l \mapsto 0], (\textbf{ifz } l? \textbf{ then } \text{“ok” } \textbf{else } \text{“bug”}) \\
\rightarrow\; &\sigma[l \mapsto 0], (\textbf{ifz } 0 \textbf{ then } \text{“ok” } \textbf{else } \text{“bug”}) \\
\rightarrow\; &\sigma[l \mapsto 0], (\text{“ok”}) \quad \blacksquare
\end{aligned}$$

DENOTATIONAL SEMANTICS

World of Code



Program Fragments

$$\xrightarrow{[\![-]\!]_{\mathrm{prog}}}$$

World of Meaning

Denotational Domain

Sequential setting
— state transformers: $\underline{T}X \triangleq (\mathbb{S} \to \mathbb{S} \times X)$

$$[\![l := 0 \,;\, \textbf{ifz } l? \textbf{ then } \text{“ok” } \textbf{else } \text{“bug”}]\!]_{\mathrm{prog}} = \lambda\sigma.\, \langle \sigma[l \mapsto 0], \text{“ok”}\rangle \in \underline{T}\mathsf{String}$$
$$= [\![l := 0 \,;\, \text{“ok”}]\!]_{\mathrm{prog}}$$

MONAD-BASED SEMANTICS

$$[\![l := 0 \ ; \ \textbf{ifz}\ l?\ \textbf{then}\ \text{“ok”}\ \textbf{else}\ \text{“bug”}]\!]_{\text{prog}}$$

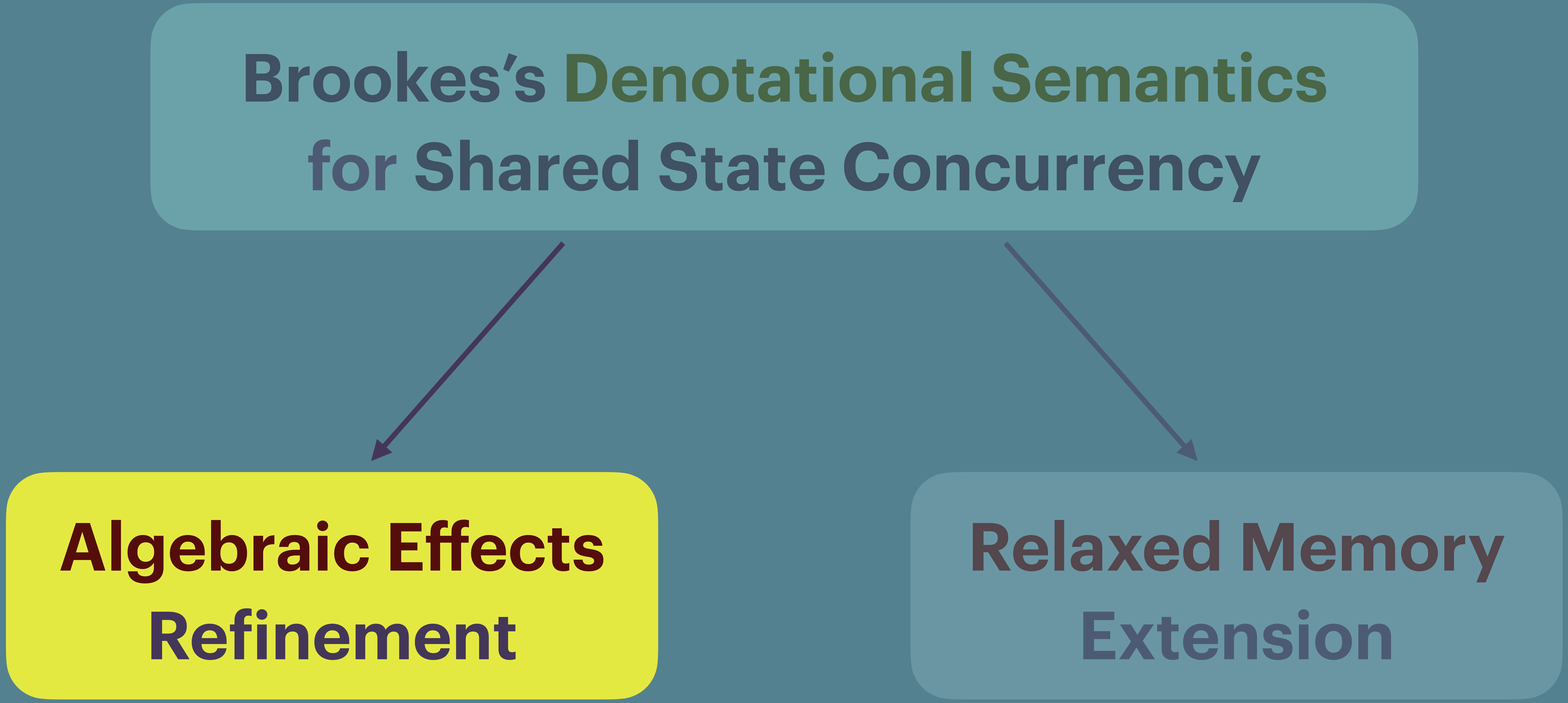
$$= [\![l := 0]\!]_{\text{prog}} \gg\!= \lambda\langle\rangle.\ [\![l?]\!]_{\text{prog}} \gg\!= \lambda b.\ (\text{ifz}\ b\ \text{then}\ \eta\,\text{“ok”}\ \text{else}\ \eta\,\text{“bug”})$$

$$= \lambda\sigma.\ \langle \sigma[l \mapsto 0], \text{“ok”} \rangle \in \underline{T\text{String}}$$

- Domain: state transformers $\underline{TX} \triangleq (\mathbb{S} \to \mathbb{S} \times X)$
- Extends $e : X \to \underline{TY}$ to $\gg\!= e : \underline{TX} \to \underline{TY}$ as follows:

  $f \gg\!= e \triangleq \lambda\sigma.\ \text{let}\ \langle \rho, y \rangle = f\sigma\ \text{in}\ ey\rho$ (modified memory ρ propagates)

  $\gg\!=$ is associative η is neutral for $\gg\!=$
- Unit: $\eta : X \to \underline{TX}$ $\eta x \triangleq \lambda\sigma.\ \langle \sigma, x \rangle$ (no change or dependency on state σ)
- Write: $[\![l := v]\!]_{\text{prog}} = \lambda\sigma.\ \langle \sigma[l \mapsto v], \langle\rangle \rangle \in \underline{T\mathbf{1}}$ Read: $[\![l?]\!]_{\text{prog}} = \lambda\sigma.\ \langle \sigma, \sigma_l \rangle \in \underline{T\mathbf{Val}}$


Brookes's Denotational Semantics
for Shared State Concurrency
Algebraic Effects
Refinement
Relaxed Memory
Extension

ALGEBRAIC EFFECTS

$$[\![l := \mathsf{0} \;;\; \mathbf{ifz}\; l?\; \mathbf{then}\; \text{“}\mathsf{ok}\text{”}\; \mathbf{else}\; \text{“}\mathsf{bug}\text{”}]\!]_{\mathrm{prog}} = \lambda\sigma.\, \langle \sigma[l \mapsto \mathsf{0}], \text{“}\mathsf{ok}\text{”}\rangle = [\![l := \mathsf{0} \;;\; \text{“}\mathsf{ok}\text{”}]\!]_{\mathrm{prog}}$$

ALGEBRAIC EFFECTS

Global State Axiom

$$\boxed{(\texttt{UL})\ \ \mathsf{U}_{l,v}\,\mathsf{L}_l(x_0, x_1) = \mathsf{U}_{l,v}\, x_v}$$

$\mathsf{U}_{l,0}$ $\mathsf{L}_l\,(\text{“ok”}, \text{“bug”})$ $\mathsf{U}_{l,0}$ “ok”

$$[\![l := 0 \,;\, \textbf{ifz}\ l?\ \textbf{then}\ \text{“ok”}\ \textbf{else}\ \text{“bug”}]\!]_{\text{prog}} = \lambda\sigma.\,\langle \sigma[l \mapsto 0], \text{“ok”}\rangle = [\![l := 0 \,;\, \text{“ok”}]\!]_{\text{prog}}$$

$$\| \qquad\qquad \|$$

$$[\![\mathsf{U}_{l,0}\,\mathsf{L}_l\,(\text{“ok”}, \text{“bug”})]\!]_{\text{term}} = [\![\mathsf{U}_{l,0}\ \text{“ok”}]\!]_{\text{term}}$$

$$[\![\mathsf{U}_{l,v}]\!]_{\text{op}} f \triangleq \lambda\sigma \in \mathbb{S}.\, f\,(\sigma[l \mapsto v]) \qquad [\![\mathsf{L}_l]\!]_{\text{op}}(f_0, f_1) \triangleq \lambda\sigma \in \mathbb{S}.\, f_{\sigma_l}\sigma$$

$$[\![\mathsf{U}_{l,v}\langle\rangle]\!]_{\text{term}} = [\![\mathsf{U}_{l,v}]\!]_{\text{op}}\,(\eta\,\langle\rangle) = [\![l := v]\!]_{\text{prog}} \qquad [\![\mathsf{L}_l(0,1)]\!]_{\text{term}} = [\![\mathsf{L}_l]\!]_{\text{op}}(\eta 0, \eta 1) = [\![l?]\!]_{\text{prog}}$$

GLOBAL STATE & NON-DETERMINISM

Global State:

* Operators for updating $\mathsf{U}_{l,v} : 1$ and looking up $\mathsf{L}_l : 2$ bits in storage
* Axioms such as $(\mathtt{UL})\ \ \mathsf{U}_{l,v}\,\mathsf{L}_l(x_0, x_1) = \mathsf{U}_{l,v}\,x_v$

Adding Non-determinism:

Countable non-determinism is similar

* Operators for choice: binary $\vee : 2$ and empty $\perp : 0$
* Axioms of semilattice, e.g.: (Symmetry) $x \vee y = y \vee x$ (Neutrality) $x \vee \perp = x$
* Axioms of interaction, e.g.: $(\vee\text{-}\mathtt{U})\ \ \mathsf{U}_{l,v}(x \vee y) = (\mathsf{U}_{l,v}\,x) \vee (\mathsf{U}_{l,v}\,y)$ $(\perp\text{-}\mathtt{U})\ \ \mathsf{U}_{l,v}\perp = \perp$

$$t \geq r \ \triangleq\ t \vee r = t$$

COOPERATIVE CONCURRENCY

SMALL-STEP SEMANTICS

$$
\begin{aligned}
& \sigma, (l := \mathtt{1} \;\; \| \;\; l := \mathtt{0} \;;\; \textbf{yield} \;;\; \textbf{ifz}\; l?\; \textbf{then}\; \text{“ok”}\; \textbf{else}\; \text{“bug”}) \\
\rightarrow\; & \sigma, (l := \mathtt{1} \;\; \|\rangle\; l := \mathtt{0} \;;\; \textbf{yield} \;;\; \textbf{ifz}\; l?\; \textbf{then}\; \text{“ok”}\; \textbf{else}\; \text{“bug”}) \\
\rightarrow\; & \sigma[l \mapsto \mathtt{0}], (l := \mathtt{1} \;\; \|\rangle\; \textbf{yield} \;;\; \textbf{ifz}\; l?\; \textbf{then}\; \text{“ok”}\; \textbf{else}\; \text{“bug”}) \\
\rightarrow\; & \sigma[l \mapsto \mathtt{0}], (l := \mathtt{1} \;\; \| \;\; \textbf{ifz}\; l?\; \textbf{then}\; \text{“ok”}\; \textbf{else}\; \text{“bug”}) \\
\rightarrow\; & \ldots
\end{aligned}
$$

MONAD-BASED SEMANTICS

ALGEBRAIC EFFECTS: RESUMPTIONS

$$[\![l := 0 \,;\, \textbf{ifz}\ l?\ \textbf{then}\ \text{“ok”}\ \textbf{else}\ (\textbf{yield}\ ;\ \text{“bug”})]\!]_{\text{prog}} = [\![\mathsf{U}_{l,0}\, \mathsf{L}_l\, (\text{“ok”}, \mathsf{Y}\ \text{“bug”})]\!]_{\text{term}}$$

$$\overset{(\mathtt{UL})}{=} [\![\mathsf{U}_{l,0}\ \text{“ok”}]\!]_{\text{term}} = [\![l := 0 \,;\, \text{“ok”}]\!]_{\text{prog}}$$

The theory of resumptions Res takes non-deterministic global state and adds:

* Operator for yielding to the concurrent environment $\mathsf{Y} : 1$
* Axioms of closure: (Pure) $\mathsf{Y}\, x \geq x$ (Join) $\mathsf{Y}\,\mathsf{Y}\, x = \mathsf{Y}\, x$
* Axioms of interaction: ($\vee$-$\mathtt{Y}$) $\mathsf{Y}(x \vee y) = (\mathsf{Y}\, x) \vee (\mathsf{Y}\, y)$ ($\bot$-$\mathtt{Y}$) $\mathsf{Y} \bot = \bot$

ALGEBRAIC EFFECTS: RESUMPTIONS

$$[\![l := 0 \,;\, \textbf{yield} \,;\, \textbf{ifz}\ l?\ \textbf{then}\ \text{“ok”}\ \textbf{else}\ \text{“bug”}]\!]_{\text{prog}} = [\![\mathsf{U}_{l,0}\, \mathsf{Y}\, \mathsf{L}_l\, (\text{“ok”}, \text{“bug”})]\!]_{\text{term}}$$

$$\overset{(\mathsf{Pure})}{\geq} [\![\mathsf{U}_{l,0}\, \mathsf{L}_l\, (\text{“ok”}, \text{“bug”})]\!]_{\text{term}} \overset{(\mathtt{UL})}{=} [\![\mathsf{U}_{l,0}\, \text{“ok”}]\!]_{\text{term}} = [\![l := 0 \,;\, \text{“ok”}]\!]_{\text{prog}}$$

The theory of resumptions Res takes non-deterministic global state and adds:

* Operator for yielding to the concurrent environment $\mathsf{Y} : 1$
* Axioms of closure: (Pure) $\mathsf{Y}\, x \geq x$ (Join) $\mathsf{Y}\,\mathsf{Y}\, x = \mathsf{Y}\, x$
* Axioms of interaction: ($\vee$-$\mathtt{Y}$) $\mathsf{Y}(x \vee y) = (\mathsf{Y}\, x) \vee (\mathsf{Y}\, y)$ ($\bot$-$\mathtt{Y}$) $\mathsf{Y}\, \bot = \bot$

PREEMPTIVE CONCURRENCY

SMALL-STEP SEMANTICS

$$\begin{aligned}
&\sigma, (l := \mathtt{1} \;\; \| \;\; l := \mathtt{0} \; ; \textbf{ifz } l? \textbf{ then } \text{“ok” } \textbf{else } \text{“bug”}) \\
\to\ & \sigma[l \mapsto \mathtt{0}], (l := \mathtt{1} \;\; \| \;\; \textbf{ifz } l? \textbf{ then } \text{“ok” } \textbf{else } \text{“bug”}) \\
\to\ & \sigma[l \mapsto \mathtt{1}], (\langle\rangle \;\; \| \;\; \textbf{ifz } l? \textbf{ then } \text{“ok” } \textbf{else } \text{“bug”}) \\
\to\ & \ldots
\end{aligned}$$

MONAD-BASED SEMANTICS

ALGEBRAIC EFFECTS: RESUMPTIONS?

- Possible intuition: "preemptive interleaving implicitly yields between steps"
- Algebraically — use the yield operator even though there's no yield construct
- Problem: does the read construct yield?

Abstraction issue

$$[\![l?]\!]_{\mathrm{prog}} = [\![\mathsf{Y}\,\mathsf{L}_l(\mathsf{Y}\,0, \mathsf{Y}\,1)]\!]_{\mathrm{term}} \qquad [\![\textbf{ifz } l? \textbf{ then } \text{“ok”} \textbf{ else } \text{“ok”}]\!]_{\mathrm{prog}} \neq [\![\text{“ok”}]\!]_{\mathrm{prog}}$$

Not even sound

$$[\![l?]\!]_{\mathrm{prog}} = [\![\mathsf{L}_l(0, 1)]\!]_{\mathrm{term}} \qquad [\![\textbf{ifz } l? \textbf{ then } l? \textbf{ else } 0]\!]_{\mathrm{prog}} = [\![0]\!]_{\mathrm{prog}}$$

Variations fail too (no-go theorem)

ALGEBRAIC EFFECTS: RESUMPTIONS?

- Possible intuition: “preemptive interleaving implicitly yields between steps”
- Algebraically — use the yield operator even though there’s no yield construct
- Problem: does the read construct yield?

Abstraction issue

$$[\![l?]\!]_{\mathrm{prog}} = [\![\mathsf{Y}\,\mathsf{L}_l(\mathsf{Y}\,0, \mathsf{Y}\,1)]\!]_{\mathrm{term}} \qquad [\![\textbf{ifz } l? \textbf{ then } \text{“ok” } \textbf{else } \text{“ok”}]\!]_{\mathrm{prog}} \neq [\![\text{“ok”}]\!]_{\mathrm{prog}}$$

Not even sound

$$[\![l?]\!]_{\mathrm{prog}} = [\![\mathsf{L}_l(0, 1)]\!]_{\mathrm{term}} \qquad [\![\textbf{ifz } l? \textbf{ then } l? \textbf{ else } 0]\!]_{\mathrm{prog}} = [\![0]\!]_{\mathrm{prog}}$$

Variations fail too (no-go theorem)

Setting	Sequential	Cooperative Concurrency	Preemptive Concurrency
Monad	State Transformers	...	? ?
Alg. Theory	Global State	Resumptions	

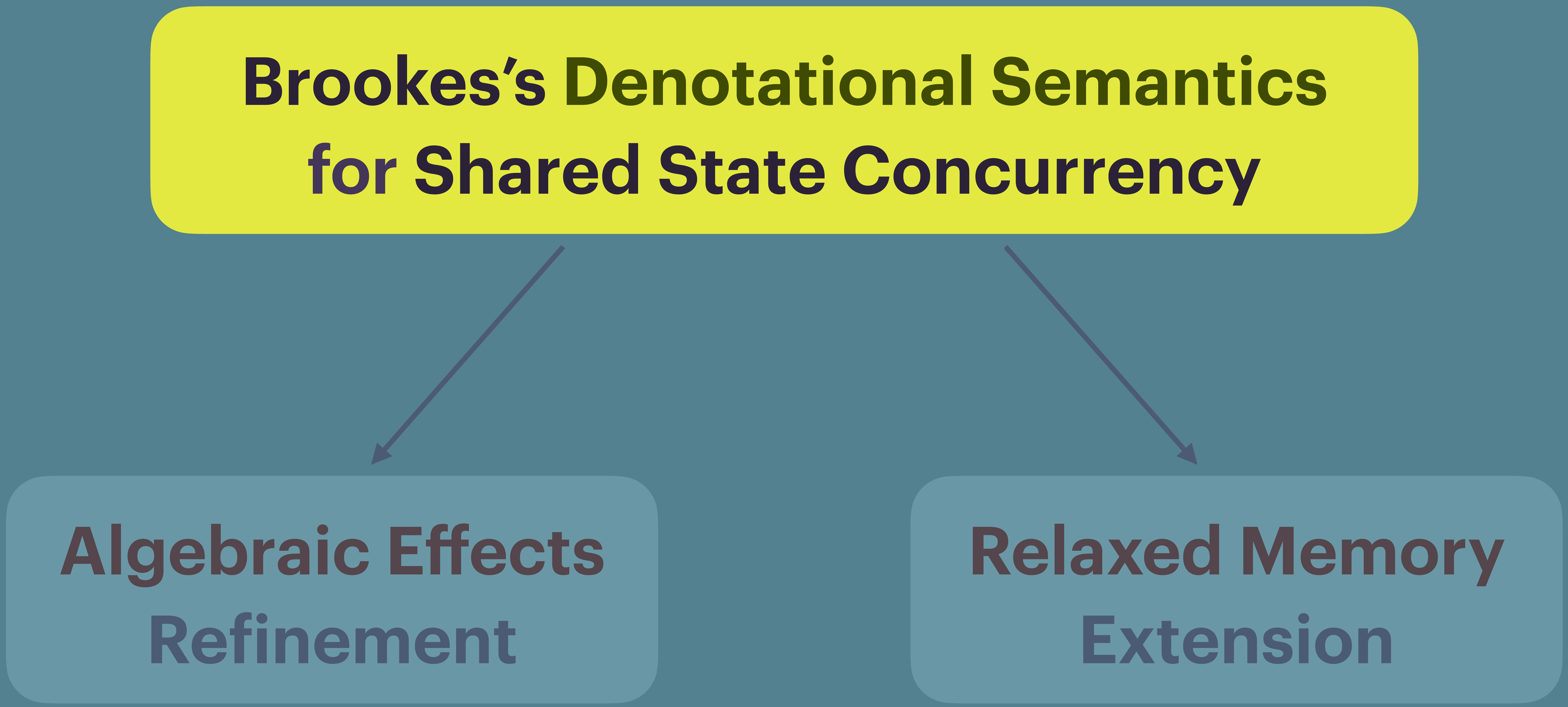
the process is a kind of reverse engineering

~ Hyland & Power, 2007

TARGET: THE BROOKES MONAD

Setting	Sequential	Cooperative Concurrency	Preemptive Concurrency
Monad	State Transformers	...	Brookes Monad 👀
Alg. Theory	Global State	Resumptions	? ?

* Highly Abstract: e.g. has $[\![\textbf{ifz } l? \textbf{ then } \text{“ok”} \textbf{ else } \text{“ok”}]\!]_{\text{prog}} = [\![\text{“ok”}]\!]_{\text{prog}}$
* Extensible: e.g. infinite executions, type-and-effect systems, allocations, relaxed memory



Brookes's Denotational Semantics for Shared State Concurrency
Algebraic Effects Refinement
Relaxed Memory Extension

TRACE-BASED SEMANTICS

Brookes [1996]

A denotation is a set of protocols (traces) that a pool of threads may adhere to

No interference

Possible interference

$$\boxed{\langle \mu_1, \rho_1 \rangle \langle \mu_2, \rho_2 \rangle \ldots \langle \mu_{n-1}, \rho_{n-1} \rangle \langle \mu_n, \rho_n \rangle}$$

TRACE-BASED SEMANTICS

Brookes [1996]

A denotation is a set of protocols (traces) that a pool of threads may adhere to

No interference

Possible interference

$$\langle \mu_1, \rho_1 \rangle \langle \mu_2, \rho_2 \rangle \ldots \langle \mu_{n-1}, \rho_{n-1} \rangle \langle \mu_n, \rho_n \rangle$$

TRACE-BASED SEMANTICS

Brookes [1996]

A denotation is a set of protocols (traces) that a pool of threads may adhere to

No interference

Possible interference

$$\langle \mu_1, \rho_1 \rangle \langle \mu_2, \rho_2 \rangle \dots \langle \mu_{n-1}, \rho_{n-1} \rangle \langle \mu_n, \rho_n \rangle$$

TRACE-BASED SEMANTICS

Brookes [1996]

A denotation is a set of protocols (traces) that a pool of threads may adhere to

No interference

Possible interference

$$\langle \mu_1, \rho_1 \rangle \langle \mu_2, \rho_2 \rangle \ldots \langle \mu_{n-1}, \rho_{n-1} \rangle \langle \mu_n, \rho_n \rangle$$

$\langle \mu_1, \mu_1' \rangle$ $\langle \mu_2, \mu_2' \rangle$ ... $\langle \mu_n, \mu_n' \rangle$

$\langle \rho_1, \rho_1' \rangle$ $\langle \rho_2, \rho_2' \rangle$ ... $\langle \rho_n, \rho_n' \rangle$

TRACE-BASED SEMANTICS

Brookes [1996]

A denotation is a set of protocols (traces) that a pool of threads may adhere to

No interference

Possible interference

$$\langle \mu_1, \rho_1 \rangle \langle \mu_2, \rho_2 \rangle \ldots \langle \mu_{n-1}, \rho_{n-1} \rangle \langle \mu_n, \rho_n \rangle$$

$$\langle \mu_1, \mu_1' \rangle \;\; \langle \mu_2, \mu_2' \rangle \;\; \ldots \;\; \langle \mu_n, \mu_n' \rangle \;\; \langle \rho_1, \rho_1' \rangle \;\; \langle \rho_2, \rho_2' \rangle \;\; \ldots \;\; \langle \rho_n, \rho_n' \rangle$$

SEQUENCE

TRACE-BASED SEMANTICS

Brookes [1996]

A denotation is a set of protocols (traces) that a pool of threads may adhere to

No interference

Possible interference

$$\langle \mu_1, \rho_1 \rangle \langle \mu_2, \rho_2 \rangle \ldots \langle \mu_{n-1}, \rho_{n-1} \rangle \langle \mu_n, \rho_n \rangle$$

$$\langle \rho_1, \rho_1' \rangle \quad \langle \mu_1, \mu_1' \rangle \quad \langle \mu_2, \mu_2' \rangle \quad \langle \rho_2, \rho_2' \rangle \quad \ldots \quad \langle \mu_n, \mu_n' \rangle \quad \langle \rho_n, \rho_n' \rangle$$

INTERLEAVE

RELY/GUARANTEE INTUITION

Rely On μ_1
To Guarantee ρ_1

Then

Rely On μ_2
To Guarantee ρ_2

Then...

Before or || ‖ ‖ ‖ ‖ After or || After

$$\underbrace{\langle \mu_1, \rho_1 \rangle \langle \mu_2, \rho_2 \rangle \cdots \langle \mu_{n-1}, \rho_{n-1} \rangle \langle \mu_n, \rho_n \rangle}_{\text{Sequence of Transitions}} \underbrace{r}_{\text{Returns}}$$

TRACE DEDUCTIONS CLOSURE RULES

- **Close denotations under trace deductions**

x-Deduction Closure

$$\frac{\pi \; r \in [\![M]\!]}{\tau \; r \in [\![M]\!]} \; \pi \xrightarrow{\mathsf{x}} \tau$$

- **Never introduce externally observable behavior**

$$\xi\eta \xrightarrow{\textbf{stutter}} \xi\langle\mu,\mu\rangle\eta$$

Propagate Reliance as a Guarantee

$$\xi\langle\mu,\rho\rangle\langle\rho,\theta\rangle\eta \xrightarrow{\textbf{mumble}} \xi\langle\mu,\theta\rangle\eta$$

Rely on an omitted Guarantee

THE BROOKES MONAD

* Domain: $\dagger$-closed sets of traces $\underline{BX} \triangleq \mathcal{P}^{\dagger}(\mathsf{T}X)$
* Unit: $\eta : X \to \underline{BX}$ $\qquad \eta x \triangleq \{\langle \sigma, \sigma \rangle x \mid \sigma \in \mathbb{S}\}^{\dagger}$
* Extends $e : X \to \underline{BY}$ to $\gg\!= e : \underline{BX} \to \underline{BY}$ as follows:

$$K \gg\!= e \triangleq \{\xi_1 \xi_2 y \mid \xi_1 x \in K, \xi_2 y \in ex\}^{\dagger}$$

* Write: $[\![l := v]\!]_{\mathrm{prog}} \triangleq \{\langle \sigma, \sigma[l \mapsto v] \rangle \langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger} \in \underline{B\mathbf{1}}$
* Read: $[\![l?]\!]_{\mathrm{prog}} \triangleq \{\langle \sigma, \sigma \rangle \sigma_l \mid \sigma \in \mathbb{S}\}^{\dagger} \in \underline{B\mathbf{Val}}$

BROOKES MONAD EXAMPLE

$$[\![l := \mathsf{0} \,;\, \mathbf{ifz}\ l?\ \mathbf{then}\ \text{“ok”}\ \mathbf{else}\ \text{“bug”}]\!]_{\text{prog}}$$

$$= [\![l := \mathsf{0}]\!]_{\text{prog}} \gg\!= \lambda\langle\rangle.\, [\![l?]\!]_{\text{prog}} \gg\!= \lambda b.\, \eta\,(\text{ifz}\ b\ \text{then}\ \text{“ok”}\ \text{else}\ \text{“bug”})$$

$$[\![l := \mathsf{0} \,;\, \text{“ok”}]\!]_{\text{prog}}$$

BROOKES MONAD EXAMPLE

$$\llbracket l := 0 \,;\, \textbf{ifz } l? \textbf{ then } \text{“ok”} \textbf{ else } \text{“bug”} \rrbracket_{\text{prog}}$$

$$= \llbracket l := 0 \rrbracket_{\text{prog}} \gg\!= \lambda\langle\rangle.\, \llbracket l? \rrbracket_{\text{prog}} \gg\!= \lambda b.\, \eta\, (\text{ifz } b \text{ then “ok” else “bug”})$$

$$= \{\langle \sigma, \sigma[l \mapsto 0] \rangle \langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger} \gg\!= \lambda\langle\rangle.$$

$$\llbracket l := 0 \,;\, \text{“ok”} \rrbracket_{\text{prog}}$$

BROOKES MONAD EXAMPLE

$$
\begin{aligned}
&\llbracket l := \mathsf{0} \mathbin{;} \mathbf{ifz}\ l?\ \mathbf{then}\ \text{“}\mathsf{ok}\text{”}\ \mathbf{else}\ \text{“}\mathsf{bug}\text{”} \rrbracket_{\mathrm{prog}} \\
&= \llbracket l := \mathsf{0} \rrbracket_{\mathrm{prog}} \gg\!= \lambda\langle\rangle.\, \llbracket l? \rrbracket_{\mathrm{prog}} \gg\!= \lambda b.\, \eta\, (\mathrm{ifz}\ b\ \mathrm{then}\ \text{“}\mathsf{ok}\text{”}\ \mathrm{else}\ \text{“}\mathsf{bug}\text{”}) \\
&= \{\langle \sigma, \sigma[l \mapsto \mathsf{0}] \rangle \langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger} \gg\!= \lambda\langle\rangle. \\
&\qquad\qquad \{\langle \rho, \rho \rangle \rho_l \mid \rho \in \mathbb{S}\}^{\dagger} \gg\!= \lambda b.
\end{aligned}
$$

$$\llbracket l := \mathsf{0} \mathbin{;} \text{“}\mathsf{ok}\text{”} \rrbracket_{\mathrm{prog}}$$

BROOKES MONAD EXAMPLE

$$[\![l := 0 \,;\, \textbf{ifz}\ l?\ \textbf{then}\ \text{“ok”}\ \textbf{else}\ \text{“bug”}]\!]_{\text{prog}}$$

$$= [\![l := 0]\!]_{\text{prog}} \gg\!= \lambda\langle\rangle.\, [\![l?]\!]_{\text{prog}} \gg\!= \lambda b.\, \eta\, (\text{ifz}\ b\ \text{then “ok” else “bug”})$$

$$= \{\langle\sigma, \sigma[l \mapsto 0]\rangle\langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger} \gg\!= \lambda\langle\rangle.$$

$$\{\langle\rho, \rho\rangle\rho_l \mid \rho \in \mathbb{S}\}^{\dagger} \gg\!= \lambda b.\, \{\langle\theta, \theta\rangle\, (\text{ifz}\ b\ \text{then “ok” else “bug”}) \mid \theta \in \mathbb{S}\}^{\dagger}$$

$$[\![l := 0 \,;\, \text{“ok”}]\!]_{\text{prog}}$$

BROOKES MONAD EXAMPLE

$$\llbracket l := \mathsf{0} \ ; \ \textbf{ifz } l? \textbf{ then } \text{“}\mathsf{ok}\text{”} \textbf{ else } \text{“}\mathsf{bug}\text{”}\rrbracket_{\text{prog}}$$

$$= \llbracket l := \mathsf{0}\rrbracket_{\text{prog}} \gg\!= \lambda\langle\rangle. \llbracket l?\rrbracket_{\text{prog}} \gg\!= \lambda b. \eta\,(\text{ifz } b \text{ then “}\mathsf{ok}\text{” else “}\mathsf{bug}\text{”})$$

$$= \{\langle\sigma, \sigma[l \mapsto \mathsf{0}]\rangle\langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger} \gg\!= \lambda\langle\rangle.$$
$$\{\langle\rho, \rho\rangle\rho_l \mid \rho \in \mathbb{S}\}^{\dagger} \gg\!= \lambda b. \{\langle\theta, \theta\rangle\,(\text{ifz } b \text{ then “}\mathsf{ok}\text{” else “}\mathsf{bug}\text{”}) \mid \theta \in \mathbb{S}\}^{\dagger}$$

$$= \{\langle\sigma, \sigma[l \mapsto \mathsf{0}]\rangle\langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger} \gg\!= \lambda\langle\rangle.$$
$$\{\langle\rho, \rho\rangle\langle\theta, \theta\rangle\,(\text{ifz } \rho_l \text{ then “}\mathsf{ok}\text{” else “}\mathsf{bug}\text{”}) \mid \rho, \theta \in \mathbb{S}\}^{\dagger}$$

$$\llbracket l := \mathsf{0} \ ; \ \text{“}\mathsf{ok}\text{”}\rrbracket_{\text{prog}}$$

BROOKES MONAD EXAMPLE

$$[\![l := 0 \,;\, \textbf{ifz}\ l?\ \textbf{then}\ \text{“ok”}\ \textbf{else}\ \text{“bug”}]\!]_{\text{prog}}$$

$$= [\![l := 0]\!]_{\text{prog}} \gg\!= \lambda\langle\rangle.\, [\![l?]\!]_{\text{prog}} \gg\!= \lambda b.\, \eta\,(\text{ifz}\ b\ \text{then “ok” else “bug”})$$

$$= \{\langle \sigma, \sigma[l \mapsto 0]\rangle\langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger} \gg\!= \lambda\langle\rangle.$$

$$\{\langle \rho, \rho\rangle \rho_l \mid \rho \in \mathbb{S}\}^{\dagger} \gg\!= \lambda b.\, \{\langle \theta, \theta\rangle\,(\text{ifz}\ b\ \text{then “ok” else “bug”}) \mid \theta \in \mathbb{S}\}^{\dagger}$$

$$= \{\langle \sigma, \sigma[l \mapsto 0]\rangle\langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger} \gg\!= \lambda\langle\rangle.$$

$$\{\langle \rho, \rho\rangle \cancel{\langle \theta, \theta\rangle}\,(\text{ifz}\ \rho_l\ \text{then “ok” else “bug”}) \mid \rho, \cancel{\theta} \in \mathbb{S}\}^{\dagger}$$

$$[\![l := 0 \,;\, \text{“ok”}]\!]_{\text{prog}}$$

BROOKES MONAD EXAMPLE

$$[\![l := 0 \,;\, \textbf{ifz}\ l?\ \textbf{then}\ \text{“ok”}\ \textbf{else}\ \text{“bug”}]\!]_{\text{prog}}$$

$$= [\![l := 0]\!]_{\text{prog}} \ggg\!= \lambda\langle\rangle.\, [\![l?]\!]_{\text{prog}} \ggg\!= \lambda b.\, \eta\,(\text{ifz}\ b\ \text{then “ok” else “bug”})$$

$$= \{\langle\sigma, \sigma[l \mapsto 0]\rangle\langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger} \ggg\!= \lambda\langle\rangle.$$

$$\{\langle\rho, \rho\rangle\rho_l \mid \rho \in \mathbb{S}\}^{\dagger} \ggg\!= \lambda b.\, \{\langle\theta, \theta\rangle\,(\text{ifz}\ b\ \text{then “ok” else “bug”}) \mid \theta \in \mathbb{S}\}^{\dagger}$$

$$= \{\langle\sigma, \sigma[l \mapsto 0]\rangle\langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger} \ggg\!= \lambda\langle\rangle.$$

$$\{\langle\rho, \rho\rangle\cancel{\langle\theta, \theta\rangle}\,(\text{ifz}\ \rho_l\ \text{then “ok” else “bug”}) \mid \rho, \cancel{\theta} \in \mathbb{S}\}^{\dagger}$$

$$= \{\langle\sigma, \sigma[l \mapsto 0]\rangle\langle\rho, \rho\rangle\,(\text{ifz}\ \rho_l\ \text{then “ok” else “bug”}) \mid \sigma, \rho \in \mathbb{S}\}^{\dagger}$$

$$[\![l := 0 \,;\, \text{“ok”}]\!]_{\text{prog}}$$

BROOKES MONAD EXAMPLE

$$[\![l := 0 \,;\, \textbf{ifz}\ l?\ \textbf{then}\ \text{“ok”}\ \textbf{else}\ \text{“bug”}]\!]_{\mathrm{prog}}$$

$$= [\![l := 0]\!]_{\mathrm{prog}} \gg\!\!= \lambda\langle\rangle.\, [\![l?]\!]_{\mathrm{prog}} \gg\!\!= \lambda b.\, \eta\, (\mathrm{ifz}\ b\ \mathrm{then}\ \text{“ok”}\ \mathrm{else}\ \text{“bug”})$$

$$= \{\langle \sigma, \sigma[l \mapsto 0] \rangle \langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger} \gg\!\!= \lambda\langle\rangle.$$

$$\{\langle \rho, \rho \rangle \rho_l \mid \rho \in \mathbb{S}\}^{\dagger} \gg\!\!= \lambda b.\, \{\langle \theta, \theta \rangle\, (\mathrm{ifz}\ b\ \mathrm{then}\ \text{“ok”}\ \mathrm{else}\ \text{“bug”}) \mid \theta \in \mathbb{S}\}^{\dagger}$$

$$= \{\langle \sigma, \sigma[l \mapsto 0] \rangle \langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger} \gg\!\!= \lambda\langle\rangle.$$

$$\{\langle \rho, \rho \rangle \cancel{\langle \theta, \theta \rangle}\, (\mathrm{ifz}\ \rho_l\ \mathrm{then}\ \text{“ok”}\ \mathrm{else}\ \text{“bug”}) \mid \rho, \cancel{\theta} \in \mathbb{S}\}^{\dagger}$$

$$= \{\langle \sigma, \sigma[l \mapsto 0] \rangle \langle \rho, \rho \rangle\, (\mathrm{ifz}\ \rho_l\ \mathrm{then}\ \text{“ok”}\ \mathrm{else}\ \text{“bug”}) \mid \sigma, \underline{\rho} \in \mathbb{S}\}^{\dagger}$$

$$[\![l := 0 \,;\, \text{“ok”}]\!]_{\mathrm{prog}}$$

BROOKES MONAD EXAMPLE

$$
\begin{aligned}
& [\![l := \mathsf{0} \;;\; \textbf{ifz}\ l?\ \textbf{then}\ \text{“}\mathsf{ok}\text{”}\ \textbf{else}\ \text{“}\mathsf{bug}\text{”}]\!]_{\text{prog}} \\
&= [\![l := \mathsf{0}]\!]_{\text{prog}} \gg= \lambda\langle\rangle.\, [\![l?]\!]_{\text{prog}} \gg= \lambda b.\, \eta\,(\text{ifz}\ b\ \text{then “}\mathsf{ok}\text{” else “}\mathsf{bug}\text{”}) \\
&= \{\langle \sigma, \sigma[l \mapsto \mathsf{0}]\rangle\langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger} \gg= \lambda\langle\rangle. \\
&\qquad\qquad \{\langle \rho, \rho\rangle \rho_l \mid \rho \in \mathbb{S}\}^{\dagger} \gg= \lambda b.\, \{\langle \theta, \theta\rangle\,(\text{ifz}\ b\ \text{then “}\mathsf{ok}\text{” else “}\mathsf{bug}\text{”}) \mid \theta \in \mathbb{S}\}^{\dagger} \\
&= \{\langle \sigma, \sigma[l \mapsto \mathsf{0}]\rangle\langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger} \gg= \lambda\langle\rangle. \\
&\qquad\qquad \{\langle \rho, \rho\rangle \cancel{\langle \theta, \theta\rangle}\,(\text{ifz}\ \rho_l\ \text{then “}\mathsf{ok}\text{” else “}\mathsf{bug}\text{”}) \mid \rho, \cancel{\theta} \in \mathbb{S}\}^{\dagger} \\
&= \{\langle \sigma, \sigma[l \mapsto \mathsf{0}]\rangle\langle \rho, \rho\rangle\,(\text{ifz}\ \rho_l\ \text{then “}\mathsf{ok}\text{” else “}\mathsf{bug}\text{”}) \mid \sigma, \underline{\rho} \in \mathbb{S}\}^{\dagger} \\
&\supseteq \{\langle \sigma, \sigma[l \mapsto \mathsf{0}]\rangle \text{“}\mathsf{ok}\text{”} \mid \sigma \in \mathbb{S}\}^{\dagger} = \cdots = [\![l := \mathsf{0} \;;\; \text{“}\mathsf{ok}\text{”}]\!]_{\text{prog}}
\end{aligned}
$$

BROOKES MONAD EXAMPLE

$$
\begin{aligned}
& [\![l := \mathsf{0} \,;\, \mathbf{ifz}\ l?\ \mathbf{then}\ \text{“}\mathsf{ok}\text{”}\ \mathbf{else}\ \text{“}\mathsf{bug}\text{”}]\!]_{\text{prog}} \\
&= [\![l := \mathsf{0}]\!]_{\text{prog}} \mathbin{>\!\!>\!\!=} \lambda\langle\rangle.\, [\![l?]\!]_{\text{prog}} \mathbin{>\!\!>\!\!=} \lambda b.\, \eta\,(\mathrm{ifz}\ b\ \mathrm{then}\ \text{“}\mathsf{ok}\text{”}\ \mathrm{else}\ \text{“}\mathsf{bug}\text{”}) \\
&= \{\langle \sigma, \sigma[l \mapsto \mathsf{0}] \rangle \langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger} \mathbin{>\!\!>\!\!=} \lambda\langle\rangle. \\
&\qquad\qquad \{\langle \rho, \rho \rangle \rho_l \mid \rho \in \mathbb{S}\}^{\dagger} \mathbin{>\!\!>\!\!=} \lambda b.\, \{\langle \theta, \theta \rangle\,(\mathrm{ifz}\ b\ \mathrm{then}\ \text{“}\mathsf{ok}\text{”}\ \mathrm{else}\ \text{“}\mathsf{bug}\text{”}) \mid \theta \in \mathbb{S}\}^{\dagger} \\
&= \{\langle \sigma, \sigma[l \mapsto \mathsf{0}] \rangle \langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger} \mathbin{>\!\!>\!\!=} \lambda\langle\rangle. \\
&\qquad\qquad \{\langle \rho, \rho \rangle \cancel{\langle \theta, \theta \rangle}\,(\mathrm{ifz}\ \rho_l\ \mathrm{then}\ \text{“}\mathsf{ok}\text{”}\ \mathrm{else}\ \text{“}\mathsf{bug}\text{”}) \mid \rho, \cancel{\theta} \in \mathbb{S}\}^{\dagger} \\
&= \{\langle \sigma, \sigma[l \mapsto \mathsf{0}] \rangle \langle \rho, \rho \rangle\,(\mathrm{ifz}\ \rho_l\ \mathrm{then}\ \text{“}\mathsf{ok}\text{”}\ \mathrm{else}\ \text{“}\mathsf{bug}\text{”}) \mid \sigma, \underline{\rho} \in \mathbb{S}\}^{\dagger} \\
&\supseteq \{\langle \sigma, \sigma[l \mapsto \mathsf{0}] \rangle \text{“}\mathsf{ok}\text{”} \mid \sigma \in \mathbb{S}\}^{\dagger} = \cdots = [\![l := \mathsf{0} \,;\, \text{“}\mathsf{ok}\text{”}]\!]_{\text{prog}}
\end{aligned}
$$

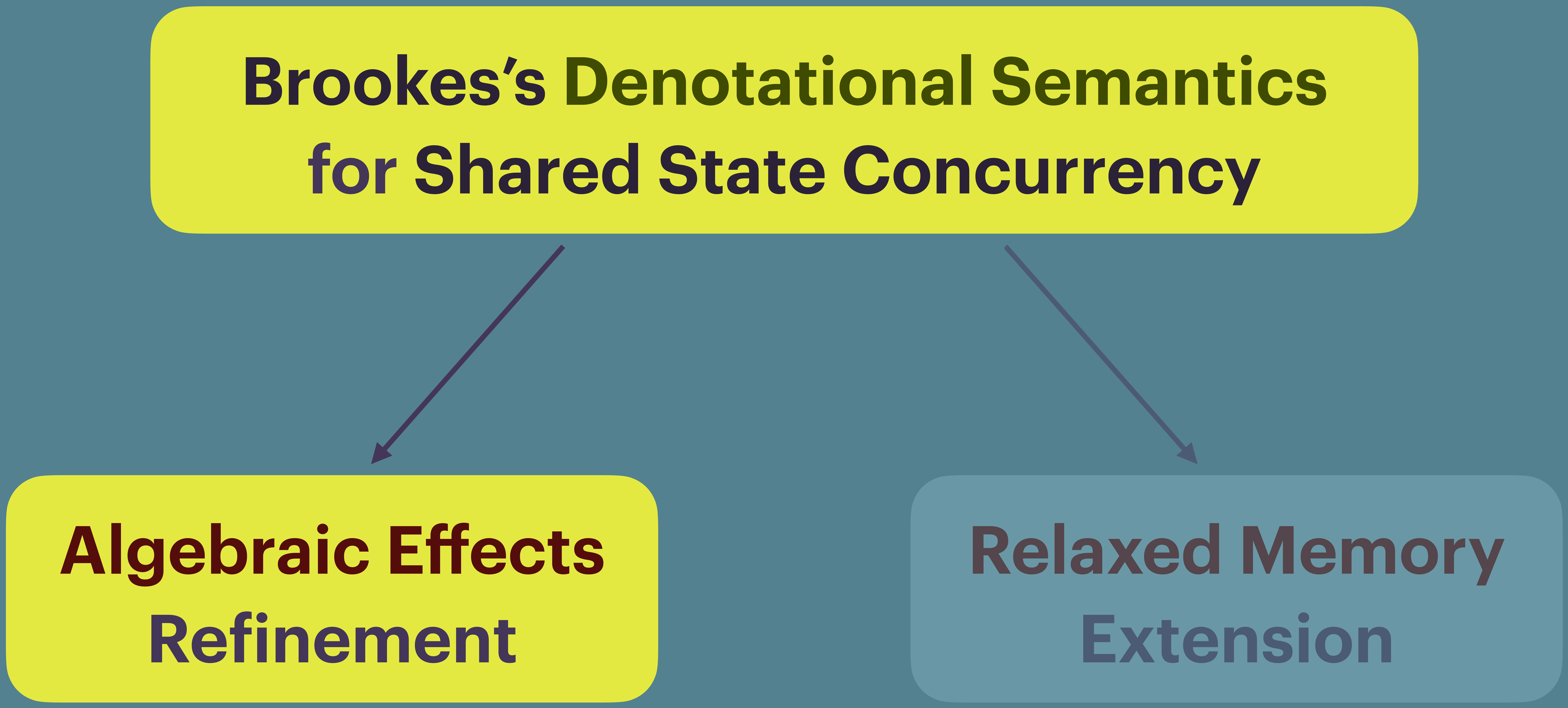
BROOKES INTERLEAVING

$$\llbracket l := 1 \quad \| \quad l := 0 \ ; \ \textbf{ifz}\ l? \ \textbf{then} \text{ “ok” } \textbf{else} \text{ “bug”} \rrbracket_{\text{prog}}$$

$$= \llbracket l := 1 \rrbracket_{\text{prog}} \quad ||| \quad \llbracket l := 0 \ ; \ \textbf{ifz}\ l? \ \textbf{then} \text{ “ok” } \textbf{else} \text{ “bug”} \rrbracket_{\text{prog}}$$

BROOKES INTERLEAVING

$$
\begin{aligned}
& [\![l := 1 \;\; \| \;\; l := 0 \,;\, \mathbf{ifz}\; l?\; \mathbf{then}\; \text{“ok”}\; \mathbf{else}\; \text{“bug”}]\!]_{\text{prog}} \\
= & [\![l := 1]\!]_{\text{prog}} \;\; ||| \;\; [\![l := 0 \,;\, \mathbf{ifz}\; l?\; \mathbf{then}\; \text{“ok”}\; \mathbf{else}\; \text{“bug”}]\!]_{\text{prog}} \\
= & \; \{ \langle \theta, \theta[l \mapsto 1] \rangle \langle \sigma, \sigma[l \mapsto 0] \rangle \langle \rho, \rho \rangle \langle \langle \rangle, (\text{ifz } \rho_l \text{ then “ok” else “bug”}) \rangle \mid \sigma, \rho, \theta \in \mathbb{S} \}^{\dagger} \\
\cup & \; \{ \langle \sigma, \sigma[l \mapsto 0] \rangle \langle \theta, \theta[l \mapsto 1] \rangle \langle \rho, \rho \rangle \langle \langle \rangle, (\text{ifz } \rho_l \text{ then “ok” else “bug”}) \rangle \mid \sigma, \rho, \theta \in \mathbb{S} \}^{\dagger} \\
\cup & \; \{ \langle \sigma, \sigma[l \mapsto 0] \rangle \langle \rho, \rho \rangle \langle \theta, \theta[l \mapsto 1] \rangle \langle \langle \rangle, (\text{ifz } \rho_l \text{ then “ok” else “bug”}) \rangle \mid \sigma, \rho, \theta \in \mathbb{S} \}^{\dagger}
\end{aligned}
$$


Brookes's Denotational Semantics for Shared State Concurrency
Algebraic Effects Refinement
Relaxed Memory Extension

ALGEBRAIC BROOKES

Setting	Sequential	Cooperative Concurrency	Preemptive Concurrency
$[\![l := 0]\!]_{\mathrm{prog}}$	$\lambda\sigma.\,\langle\sigma[l \mapsto 0], \langle\rangle\rangle$	...	$\{\langle\sigma, \sigma[l \mapsto 0]\rangle\,\langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger}$
Alg. Rep.	$[\![\mathsf{U}_{l,0}\langle\rangle]\!]_{\mathrm{term}}$	$[\![\mathsf{U}_{l,0}\langle\rangle]\!]_{\mathrm{term}}$	? ?

ALGEBRAIC BROOKES

Setting	Sequential	Cooperative Concurrency	Preemptive Concurrency
$[\![l := 0]\!]_{\text{prog}}$	$\lambda\sigma.\,\langle \sigma[l \mapsto 0], \langle\rangle\rangle$	$\ldots$	$\{\langle \sigma, \sigma[l \mapsto 0]\rangle\,\langle\rangle \mid \sigma \in \mathbb{S}\}^{\dagger}$
Alg. Rep.	$[\![\mathsf{U}_{l,0}\langle\rangle]\!]_{\text{term}}$	$[\![\mathsf{U}_{l,0}\langle\rangle]\!]_{\text{term}}$	$[\![\triangleleft\, \mathsf{U}_{l,0} \triangleright \langle\rangle]\!]_{\text{term}}$

OUR THEORY OF SHARED STATE

First example of two-sorted algebraic effects

* Sorts: Hold ($\bullet$) & Cede ($\circ$)
* Operators:
  » $\bullet$-sorted update $\mathsf{U}_{l,v} : \bullet\langle\bullet\rangle$ and lookup $\mathsf{L}_l : \bullet\langle\bullet,\bullet\rangle$
  » choice in each sort
  » acquire $\triangleleft : \circ\langle\bullet\rangle$ release $\triangleright : \bullet\langle\circ\rangle$
* Axioms:
  » $\bullet$-copy of the global state axioms
  » Standard choice axioms (including distributivity and strictness)
  » Closure pair axioms: (Empty) $\triangleleft\triangleright x = x$ (Fuse) $\triangleright\triangleleft x \geq x$

REASONING IN SHARED STATE

Represented by a two-sorted generalization $B^{\{\bullet,\circ\}}$ of the Brookes monad B

$$[\![\lhd\, \mathsf{U}_{l,v} \rhd \langle\rangle]\!]_{\mathrm{term}} \cong [\![l := v]\!]_{\mathrm{prog}} \qquad [\![\lhd\, \mathsf{L}_l(\rhd 0, \rhd 1)]\!]_{\mathrm{term}} \cong [\![l?]\!]_{\mathrm{prog}}$$

$$\begin{aligned}[\![l := 0 \,;\, \mathbf{ifz}\ l?\ \mathbf{then}\ \text{“ok”}\ \mathbf{else}\ \text{“bug”}]\!]_{\mathrm{prog}} &\cong [\![\lhd\, \mathsf{U}_{l,0} \rhd \lhd\, \mathsf{L}_l\,(\rhd\text{“ok”}, \rhd\text{“bug”})]\!]_{\mathrm{term}} \\ &\overset{(\mathsf{Fuse})}{\supseteq} [\![\lhd\, \mathsf{U}_{l,0}\, \mathsf{L}_l\,(\rhd\text{“ok”}, \rhd\text{“bug”})]\!]_{\mathrm{term}} \\ &\overset{(\mathsf{UL})}{=} [\![\lhd\, \mathsf{U}_{l,0} \rhd\text{“ok”}]\!]_{\mathrm{term}} \cong [\![l := 0 \,;\, \text{“ok”}]\!]_{\mathrm{prog}}\end{aligned}$$

(Fuse) $(\rhd \lhd x \geq x)$: fusing atomic blocks eliminates potential interference

REASONING IN SHARED STATE

Represented by a two-sorted generalization $B^{\{\bullet,\circ\}}$ of the Brookes monad B

$$[\![\lhd\, \mathsf{U}_{l,v} \rhd \langle\rangle]\!]_{\mathrm{term}} \cong [\![l := v]\!]_{\mathrm{prog}} \qquad [\![\lhd\, \mathsf{L}_l(\rhd 0, \rhd 1)]\!]_{\mathrm{term}} \cong [\![l?]\!]_{\mathrm{prog}}$$

$$[\![\textbf{ifz } l? \textbf{ then } \text{“ok”} \textbf{ else } \text{“ok”}]\!]_{\mathrm{prog}} \cong [\![\lhd\, \mathsf{L}_l\, (\rhd \text{“ok”}, \rhd \text{“ok”})]\!]_{\mathrm{term}}$$

$$\overset{\mathsf{G}}{=} [\![\lhd \rhd \text{“ok”}]\!]_{\mathrm{term}} \overset{(\mathsf{Empty})}{=} [\![\text{“ok”}]\!]_{\mathrm{term}} \cong [\![\text{“ok”}]\!]_{\mathrm{prog}}$$

(Empty) $(\lhd \rhd x = x)$: empty atomic blocks have no observable effect

TWO-SORTED TRACE SEMANTICS

(only) Before

After or ||

Stutter forbidden

$$\underbrace{\bullet}_{\text{Trace sort}}\langle \mu_1, \rho_1 \rangle \dots \langle \mu_n, \rho_n \rangle \underbrace{\circ}_{\text{Value sort}} r$$

TWO-SORTED BROOKES MONAD

Domain for each sort $\square$: closed sets of $\square$-sorted traces $\underline{B^{\{\bullet,\circ\}}\boldsymbol{X}}_{\square} \triangleq \mathcal{P}^{\dagger}((\mathbb{T}\boldsymbol{X})_{\square})$

$$[\![\mathsf{V}]\!]_{\mathrm{op}} \triangleq \bigcup$$

$$[\![\triangleleft]\!]_{\mathrm{op}} K \triangleq \{\circ\xi \diamond x \mid \bullet\xi \diamond x \in K\}^{\dagger} \qquad [\![\triangleright]\!]_{\mathrm{op}} K \triangleq \{\bullet\langle\sigma,\sigma\rangle\xi \diamond x \mid \sigma \in \mathbb{S}, \circ\xi \diamond x \in K\}^{\dagger}$$

$$[\![\mathsf{U}_{l,v}]\!]_{\mathrm{op}} K \triangleq \bigcup_{\sigma\in\mathbb{S}} (\sigma, \sigma[l \mapsto v])\, K$$

$$[\![\mathsf{L}_l]\!]_{\mathrm{op}} (K_0, K_1) \triangleq \bigcup_{\sigma\in\mathbb{S}} (\sigma, \sigma)\, K_{\sigma_l}$$

where $(\sigma, \rho)\, K \triangleq \{\bullet\langle\sigma,\theta\rangle\xi \diamond x \mid \bullet\langle\rho,\theta\rangle\xi \diamond x \in K\}$

RECOVERING BROOKES

* Monad $B^{\{\bullet,\circ\}}$ transformed along $(-)^\circ \dashv (-)_\circ \cong$ Brookes's monad B
  » $X^\circ \triangleq \{x : \circ \mid x \in X\}$
  » $\underline{B^{\{\bullet,\circ\}} X^\circ}_\circ = \mathcal{P}^\dagger((\mathbb{T}X^\circ)_\circ) \cong \mathcal{P}^\dagger(\mathsf{T}X) = \underline{BX}$
  » $[\![\lhd \mathsf{U}_{l,v} \rhd \langle\rangle]\!]_{\text{term}} \cong [\![l := v]\!]_{\text{prog}}$
  » $[\![\lhd \mathsf{L}_l(\rhd 0, \rhd 1)]\!]_{\text{term}} \cong [\![l?]\!]_{\text{prog}}$

$\cong B$ monad

$B^{\{\bullet,\circ\}}$ monad

$$
\begin{tikzcd}
\mathbb{S}\textbf{-Alg} \arrow[rr, bend right, "(\_)"'] \arrow[rr, phantom, "\perp"] & & \textbf{Set}^{\{\bullet,\circ\}} \arrow[ll, bend right, "B^{\{\bullet,\circ\}} \circlearrowright"'] \arrow[rr, bend right, "(-)_\circ"'] \arrow[rr, phantom, "\perp"] & & \textbf{Set} \arrow[ll, bend right, "(-)^\circ"'] \arrow[loop above, "\circlearrowright"]
\end{tikzcd}
$$

RECOVERING RESUMPTIONS

* Monad $B^{\{\bullet,\circ\}}$ transformed along $(-)^{\bullet} \dashv (-)_{\bullet}$ represents the resumptions theory Res
  - Closure axioms ($\mathsf{Y} \mapsto \rhd\lhd$)
    * (Pure) $\rhd\lhd x \geq x$
    * (Join) $\rhd\lhd\rhd\lhd x = \rhd\lhd x$
  - $[\![\rhd\lhd\langle\rangle]\!]_{\mathrm{term}} \cong [\![\mathbf{yield}]\!]_{\mathrm{prog}}$
  - $[\![\mathsf{U}_{l,v}\langle\rangle]\!]_{\mathrm{term}} \cong [\![l := v]\!]_{\mathrm{prog}}$
  - $[\![\mathsf{L}_l(0, 1)]\!]_{\mathrm{term}} \cong [\![l?]\!]_{\mathrm{prog}}$

represents Res

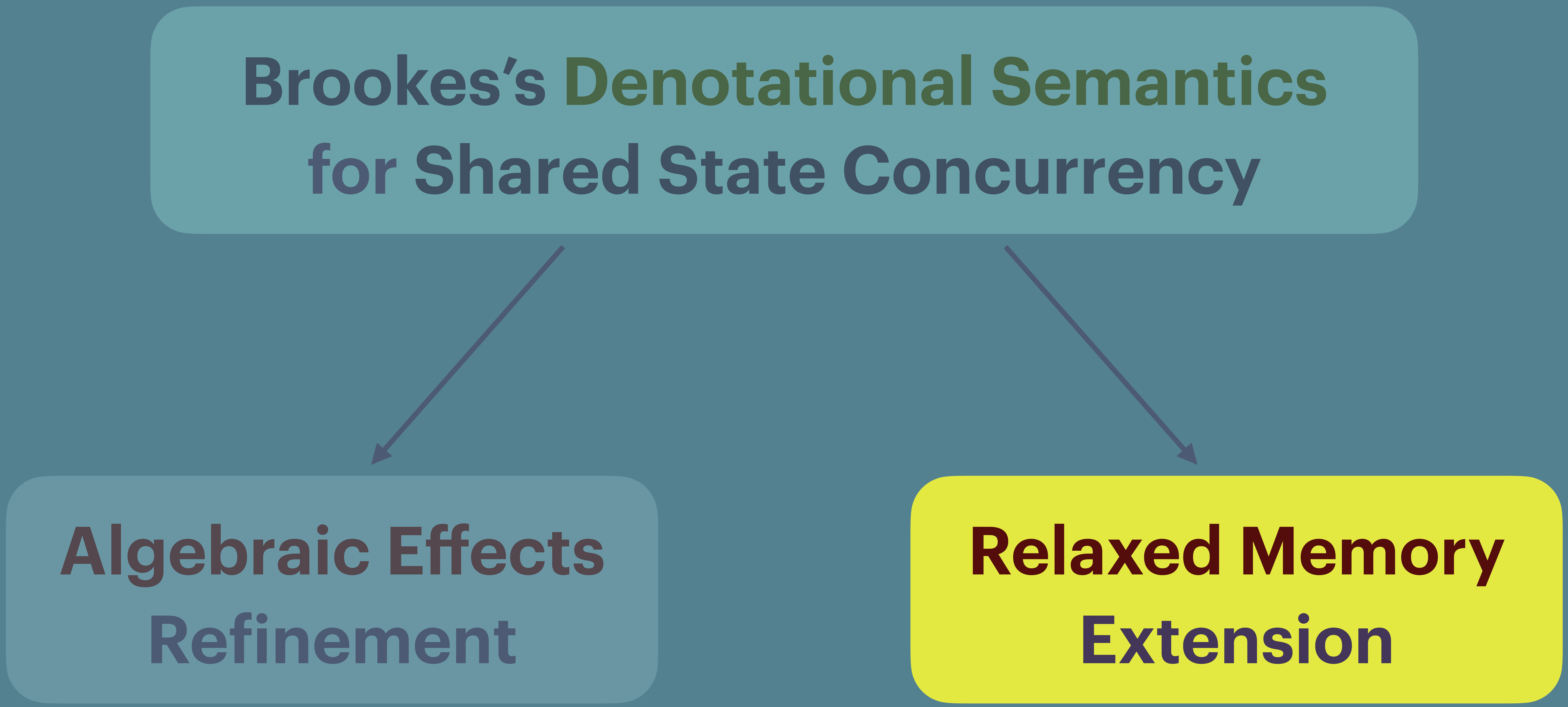
$B^{\{\bullet,\circ\}}$ monad

$$\begin{tikzcd} \mathbb{S}\text{-}\mathbf{Alg} \arrow[r, bend right, "(\_)"'] \arrow[r, phantom, "\perp"] & \mathbf{Set}^{\{\bullet,\circ\}} \arrow[l, bend right, "B^{\{\bullet,\circ\}}"'] \arrow[r, bend right, "(-)_{\bullet}"'] \arrow[r, phantom, "\perp"] & \mathbf{Set} \arrow[l, bend right, "(-)^{\bullet}"'] \end{tikzcd}$$

SUMMARY

A two-sorted algebraic effects theory for **shared state concurrency** $\mathbb{S}$:
(the first example of a multi-sorted algebraic effects theory)

- The sorts Hold $\bullet$ & Cede $\circ$ declare exclusive access to memory
- Classic algebraic effects theories: Global State in $\bullet$, Choice (semilattice) in $\bullet$ and $\circ$
- The Closure Pair theory for managing access: (Empty) $\lhd \rhd x = x$ (Fuse) $\rhd \lhd x \geq x$
- Represented by a two-sorted model recovering known models in each sort:
  - » The $\circ$-adjunction **recovers Brookes's model (preemptive concurrency)**
  - » The $\bullet$-adjunction **represents Resumptions (cooperative concurrency)**


Brookes's Denotational Semantics
for Shared State Concurrency
Algebraic Effects
Refinement
Relaxed Memory
Extension

RELAXING TRACE-BASED SEMANTICS

Paper	Memory Model	Operational Semantics
Brookes [1996]	Sequential Consistency (SC) idealized model	straightforward interleaving
Jagadeesan, Petri, Riely [2012]	Total-Store Order (TSO) hardware model	write buffer per thread
THIS WORK [2024]	Release/Acquire (RA) software model	decentralized communication

WHY RELEASE/ACQUIRE?

RA is an **important fragment** of **C11**, enables decentralized architectures (POWER)

First adaptation of *Brookes's traces* to a relaxed-memory **software** model

Intricate **causal semantics**, not overwhelmingly detailed

acyclic $(\mathtt{po} \cup \mathtt{rf})^+|_{loc} \cup \mathtt{mo} \cup \mathtt{rb}$

Threads can **disagree** about the **order of writes** (non-multi-copy-atomic)

Supports **flag-based synchronization** (e.g. for signaling a data structure is ready)

Supports **important transformations** (e.g. thread sequencing, write-read-reorder)

Supports **read-modify-write atomicity** (e.g. atomic compare-and-swap)

INTUITION VIA LITMUS TESTS

Store Buffering

$$x := 0; y := 0;$$

$$\begin{array}{l|l} x := 1; & y := 1; \\ y? & x? \end{array}$$

Message Passing

$$x := 0; y := 0;$$

$$\begin{array}{l|l} x := 1; & y?; \\ y := 1 & x? \end{array}$$

INTUITION VIA LITMUS TESTS

Store Buffering

$$x := 0; y := 0;$$

$$\begin{array}{l} x := 1; \\ y? \quad /\!/ 0 \end{array} \;\Bigg\|\; \begin{array}{l} y := 1; \\ x? \quad /\!/ 0 \end{array}$$

Message Passing

$$x := 0; y := 0;$$

$$\begin{array}{l} x := 1; \\ y := 1 \end{array} \;\Bigg\|\; \begin{array}{l} y?; \\ x? \end{array}$$

INTUITION VIA LITMUS TESTS

Store Buffering

$$x := 0; y := 0;$$

$$\begin{array}{l|l} x := 1; & y := 1; \\ y? \quad /\!/ 0 & x? \quad /\!/ 0 \end{array}$$

Message Passing

$$x := 0; y := 0;$$

$$\begin{array}{l|l} x := 1; & y?; \\ y := 1 & x? \end{array}$$

INTUITION VIA LITMUS TESTS

Propagation is not instant

Store Buffering

$$x := 0; y := 0;$$

$$\begin{array}{l|l} x := 1; & y := 1; \\ y? \quad /\!/ 0 & x? \quad /\!/ 0 \end{array}$$

✓

Message Passing

$$x := 0; y := 0;$$

$$\begin{array}{l|l} x := 1; & y?; \\ y := 1 & x? \end{array}$$

INTUITION VIA LITMUS TESTS

Propagation is not instant

Store Buffering

$$x := 0; y := 0;$$

$$\begin{array}{l|l} x := 1; & y := 1; \\ y? \quad /\!/0 & x? \quad /\!/0 \end{array}$$

Message Passing

$$x := 0; y := 0;$$

$$\begin{array}{l|l} x := 1; & y?; \; /\!/1 \\ y := 1 & x? \; /\!/0 \end{array}$$

INTUITION VIA LITMUS TESTS

Propagation is not instant

Store Buffering

$$x := 0; y := 0;$$

$$\begin{array}{ll|l|ll} x := 1; & & & y := 1; & \\ y? & /\!/0 & & x? & /\!/0 \end{array}$$

✓

Message Passing

$$x := 0; y := 0;$$

$$\begin{array}{l|ll} x := 1; & y?; & /\!/1 \\ y := 1 & x? & /\!/0 \end{array}$$

INTUITION VIA LITMUS TESTS

Store Buffering

Propagation is not instant

$$x := 0; y := 0;$$

$$\begin{array}{ll|l} x := 1; & & y := 1; \\ y? & /\!/0 & x? \quad /\!/0 \end{array}$$

Message Pas

Propagation respects causality

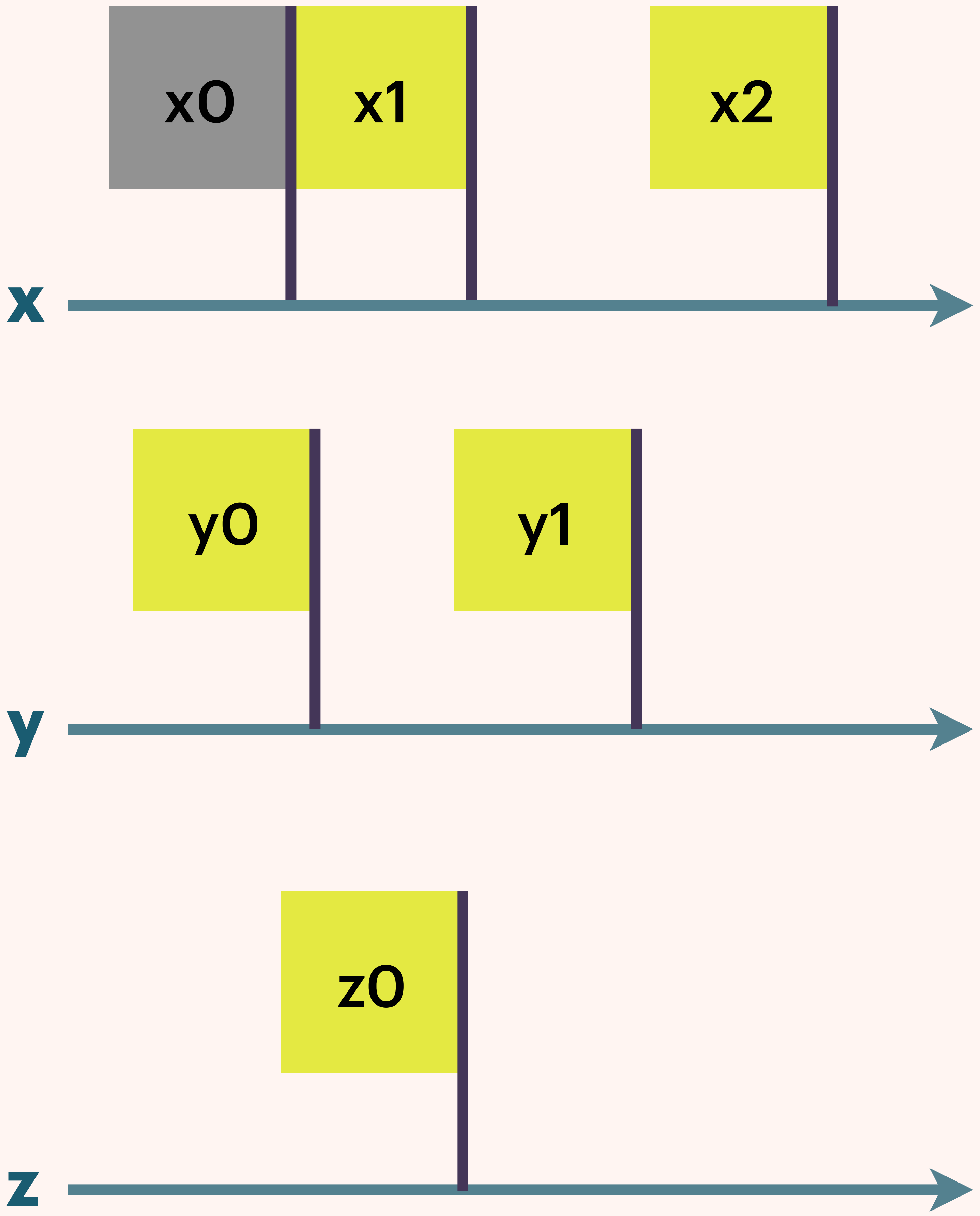
$$x := 0; y := 0;$$

$$\begin{array}{l|ll} x := 1; & y?; & /\!/1 \\ y := 1 & x? & /\!/0 \end{array}$$

RELEASE/ACQUIRE VIEW-BASED OPERATIONAL SEMANTICS

Kang et al. [2017]

- **Memory: Timeline per location**
- **Populated with immutable messages holding values**
- **Each view points to msgs on each timeline**
- **Threads have views — cannot read from "the past"**
- **Msgs have views for enforcing causal propagation**



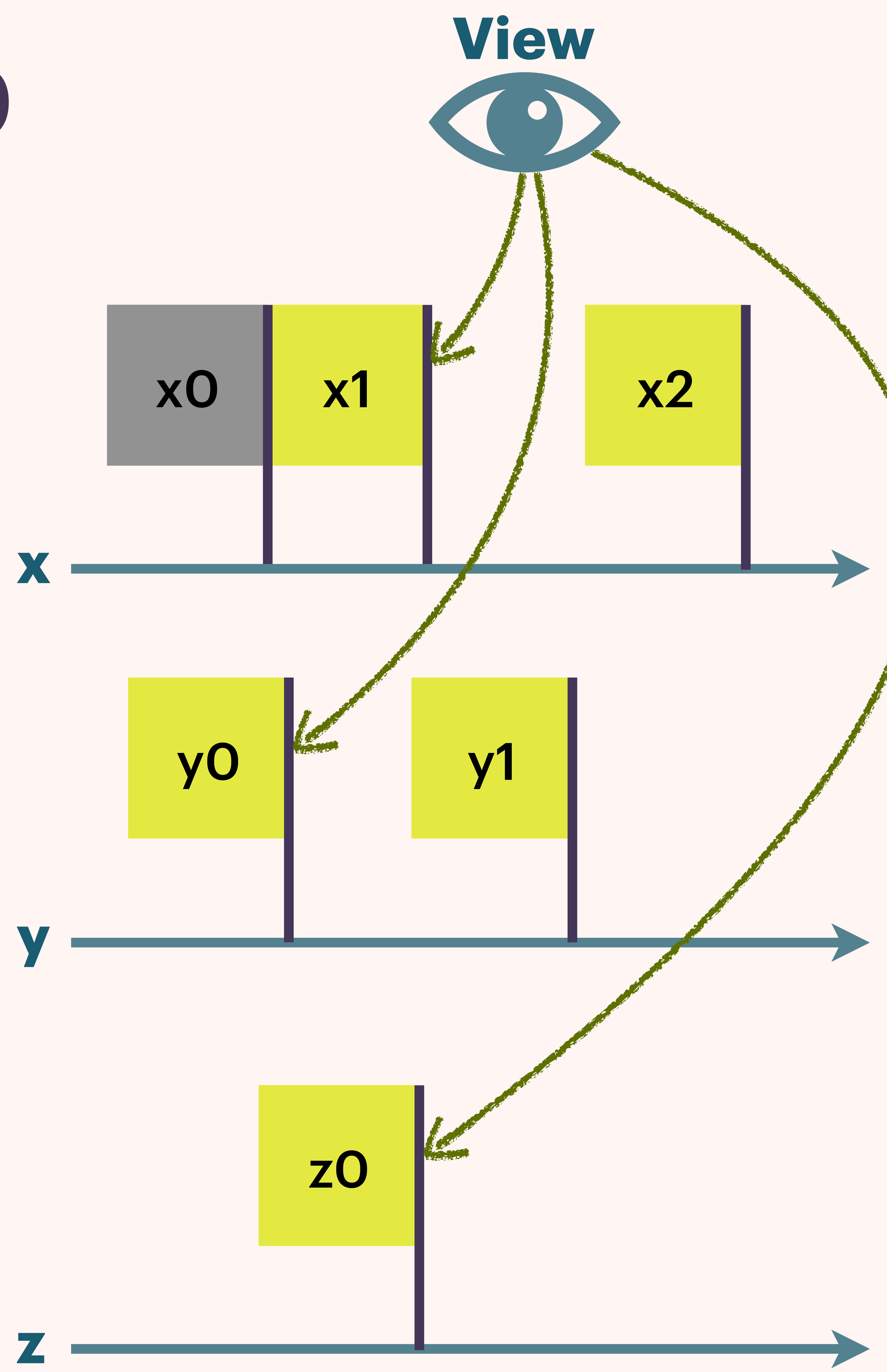
Propagation is not instant

Propagation respects causality

RELEASE/ACQUIRE VIEW-BASED OPERATIONAL SEMANTICS

Kang et al. [2017]

- **Memory:** **Timeline per location**
- **Populated with immutable messages holding values**
- **Each view points to msgs on each timeline**
- **Threads have views — cannot read from "the past"**
- **Msgs have views for enforcing causal propagation**



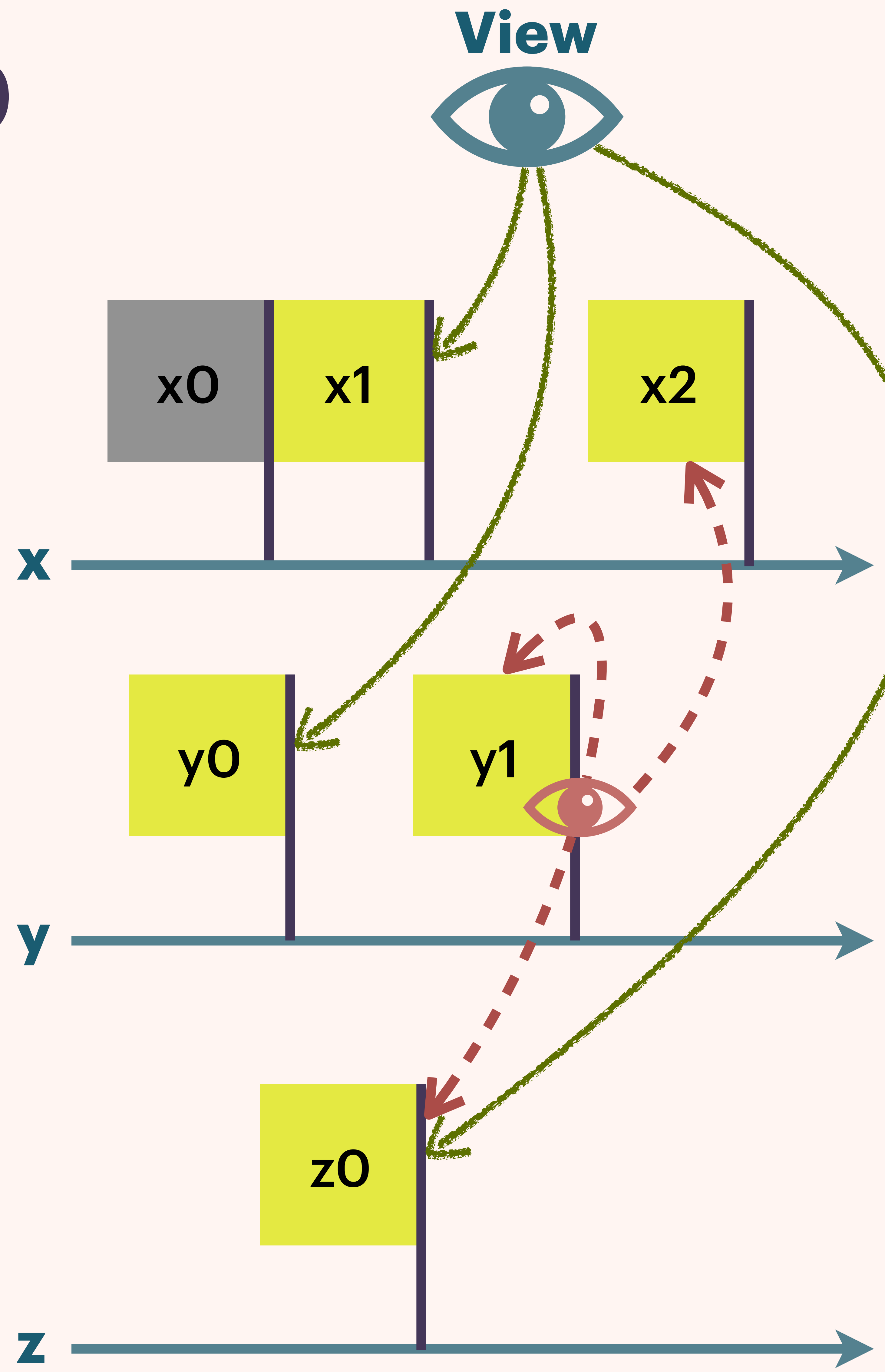
Propagation is not instant

Propagation respects causality

RELEASE/ACQUIRE VIEW-BASED OPERATIONAL SEMANTICS

Kang et al. [2017]

- **Memory: Timeline per location**
- **Populated with immutable messages holding values**
- **Each view points to msgs on each timeline**
- **Threads have views — cannot read from "the past"**
- **Msgs have views for enforcing causal propagation**



Propagation is not instant

Propagation respects causality

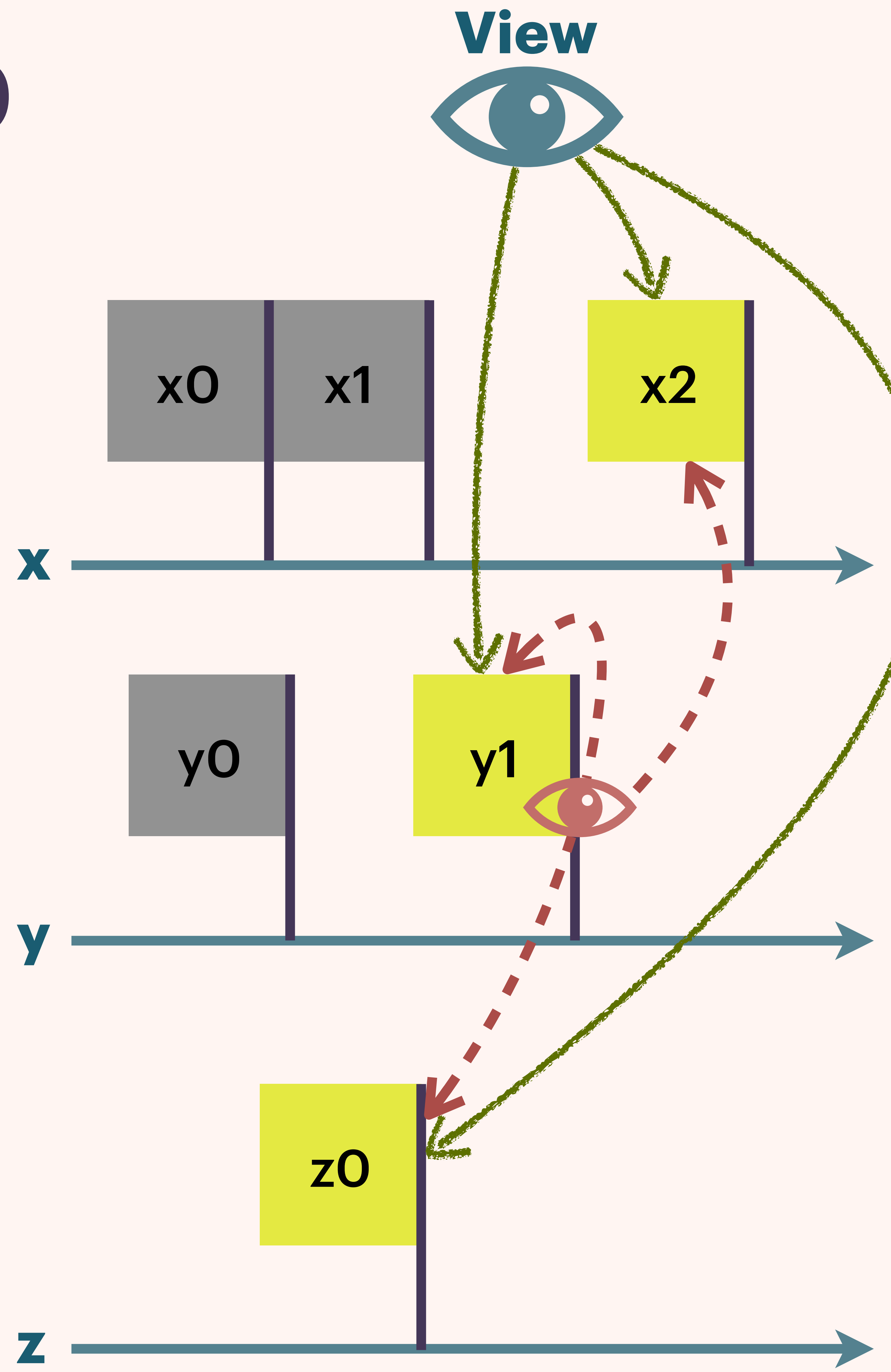
RELEASE/ACQUIRE VIEW-BASED OPERATIONAL SEMANTICS

Kang et al. [2017]

- **Memory: Timeline per location**
- **Populated with immutable messages holding values**
- **Each view points to msgs on each timeline**
- **Threads have views — cannot read from "the past"**
- **Msgs have views for enforcing causal propagation**

Propagation is not instant

Propagation respects causality



SUPPORTING FIRST-CLASS PARALLELISM

In the operational semantics

Traditional op-sem: static view-array

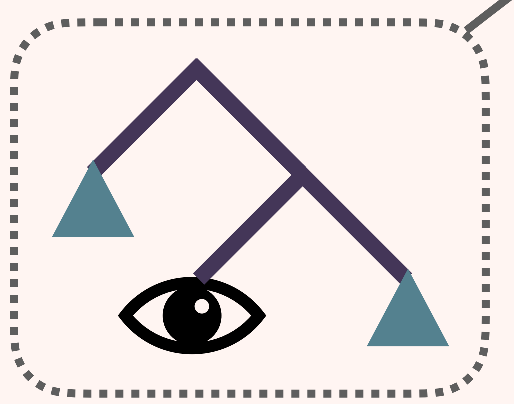
Laws of Parallel Programming, e.g. Left Neutrality

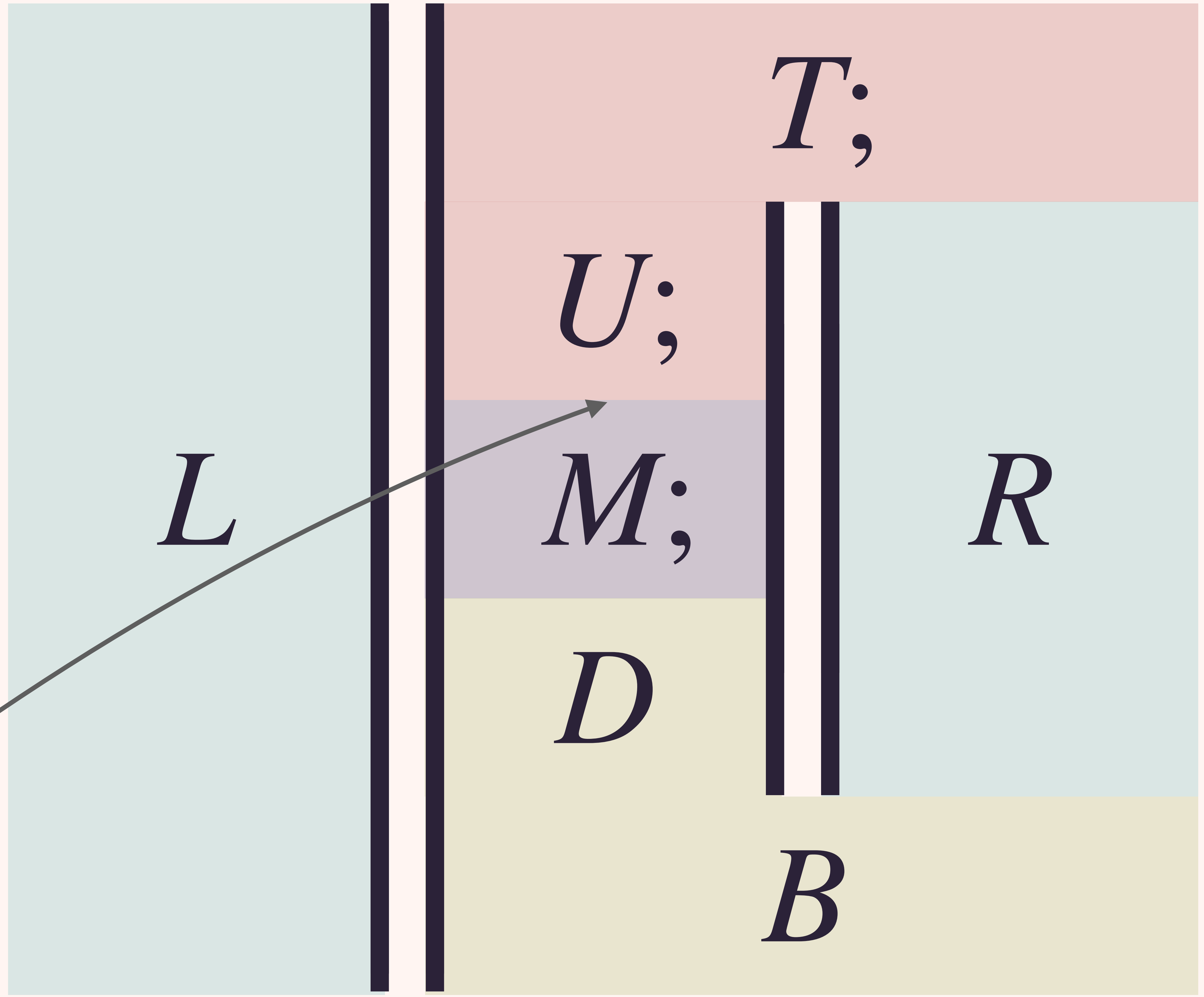
$$[\![M]\!] = [\![(\langle\rangle \parallel M).\mathrm{snd}]\!]$$

Write-Read Deorder (Crucial RA refinement)

$$[\![x := 1; y?]\!] \supseteq [\![(x := 1 \parallel y?).\mathrm{snd}]\!]$$

Extended op-sem: dynamic view-tree





Brookes's Denotational Semantics
for Shared State Concurrency

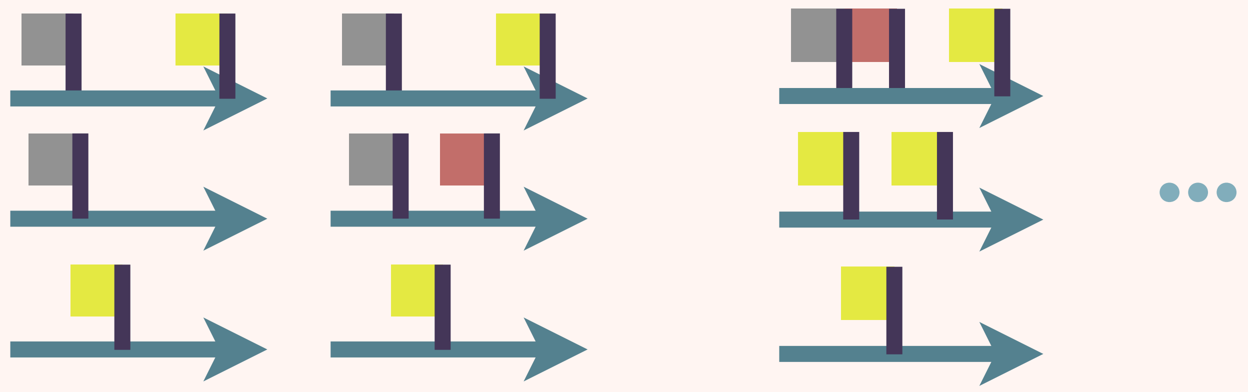
Algebraic Effects
Refinement

Relaxed Memory
Extension

TRACE-BASED SEMANTICS IN RA

Initial View

Final View



$$\alpha \underbrace{\langle \mu_1, \rho_1 \rangle \langle \mu_2, \rho_2 \rangle \ldots \langle \mu_{n-1}, \rho_{n-1} \rangle \langle \mu_n, \rho_n \rangle}_{\text{Sequence of Transitions}} \omega \therefore r$$

Returns

TRACE-BASED SEMANTICS IN RA

Rely on the sequential environment to reveal messages

Guarantee to the sequential environment to reveal messages

Before

Avoid including whole state in transitions

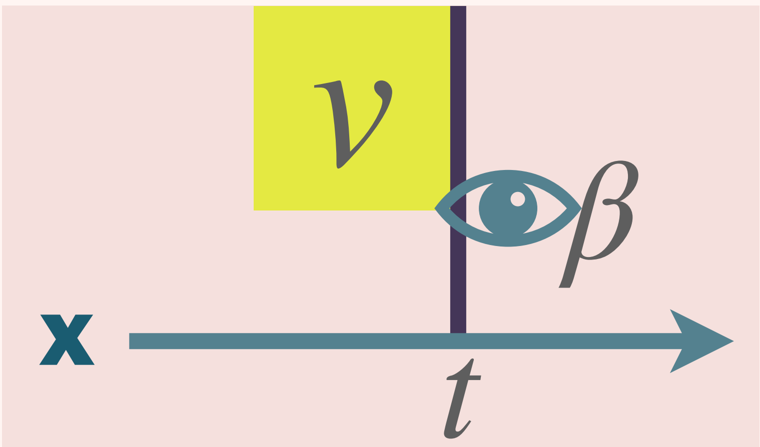
After

$$\underbrace{\alpha}_{\text{Initial View}}\ \underbrace{\langle \mu_1, \rho_1 \rangle \langle \mu_2, \rho_2 \rangle \ldots \langle \mu_{n-1}, \rho_{n-1} \rangle \langle \mu_n, \rho_n \rangle}_{\text{Sequence of Transitions}}\ \underbrace{\omega}_{\text{Final View}}\ \therefore\ \underbrace{r}_{\text{Returns}}$$

MEMORY ACCESS

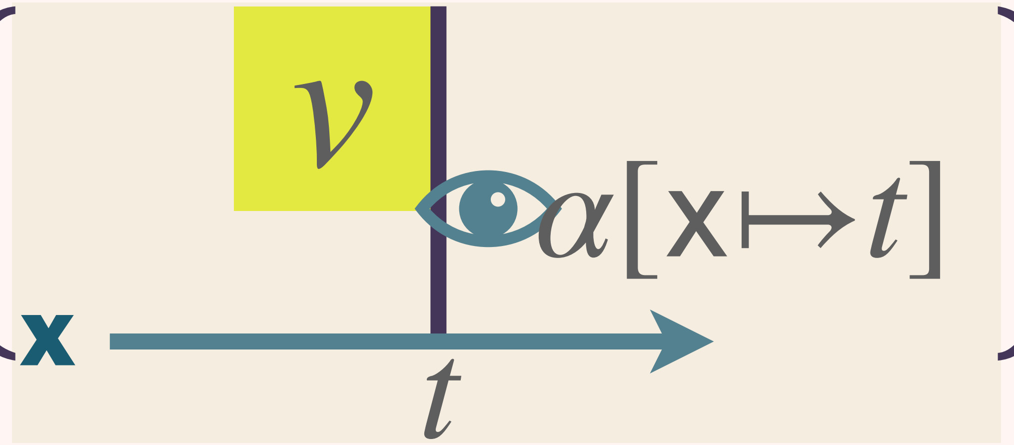
Read

$$\frac{\alpha(x) \leq t \qquad \text{[message]} \in \mu}{\alpha \, \boxed{\langle \mu, \mu \rangle} \, \alpha \sqcup \beta \therefore v \in [\![x?]\!]}$$



Write

$$\frac{\alpha(x) < t \qquad \rho = \mu \uplus \{\text{[message]}\}}{\alpha \, \boxed{\langle \mu, \rho \rangle} \, \alpha[\mathsf{x} \mapsto t] \therefore \langle\rangle \in [\![x := v]\!]}$$



RMW

Read the extended paper

٩(^‿^)۶

COMPOSITION

Sequential

$$\succcurlyeq \quad \frac{\alpha\,\boxed{\xi_1}\,\kappa \therefore r_1 \in [\![M_1]\!] \qquad \kappa\,\boxed{\xi_2}\,\omega \therefore r_2 \in [\![M_2]\!][x \mapsto r_1]}{\alpha\,\boxed{\xi_1\xi_2}\,\omega \therefore r_2 \in [\![\mathbf{let}\, x = M_1 \,\mathbf{in}\, M_2]\!]}$$

Parallel

$$\Vert \quad \frac{\forall i \in \{1,2\}\,.\;\; \alpha\,\boxed{\xi_i}\,\omega \therefore r_i \in [\![M_i]\!] \qquad \xi \in \xi_1 \,\Vert\, \xi_2}{\alpha\,\boxed{\xi}\,\omega \therefore \langle r_1, r_2 \rangle \in [\![M_1 \,\Vert\, M_2]\!]}$$

DEDUCTION CLOSURE RULES

- Close denotations under rewrite rules

x-Deduction Closure

$$\frac{\pi \therefore r \in [\![M]\!]}{\tau \therefore r \in [\![M]\!]} \; \pi \xrightarrow{\mathsf{x}} \tau$$

Brookes

$$\alpha \boxed{\xi\eta} \omega \xrightarrow{\textbf{stutter}} \alpha \boxed{\xi\langle\mu,\mu\rangle\eta} \omega$$

Propagate Reliance as a Guarantee

$$\alpha \boxed{\xi\langle\mu,\rho\rangle\langle\rho,\theta\rangle\eta} \omega \xrightarrow{\textbf{mumble}} \alpha \boxed{\xi\langle\mu,\theta\rangle\eta} \omega$$

Rely on an omitted Guarantee

- Never introduce externally observable behavior

DEDUCTION CLOSURE RULES

- Close denotations under rewrite rules

x-Deduction Closure

$$\frac{\pi \mathrel{\therefore} r \in [\![M]\!]}{\tau \mathrel{\therefore} r \in [\![M]\!]} \; \pi \xrightarrow{\mathsf{x}} \tau$$

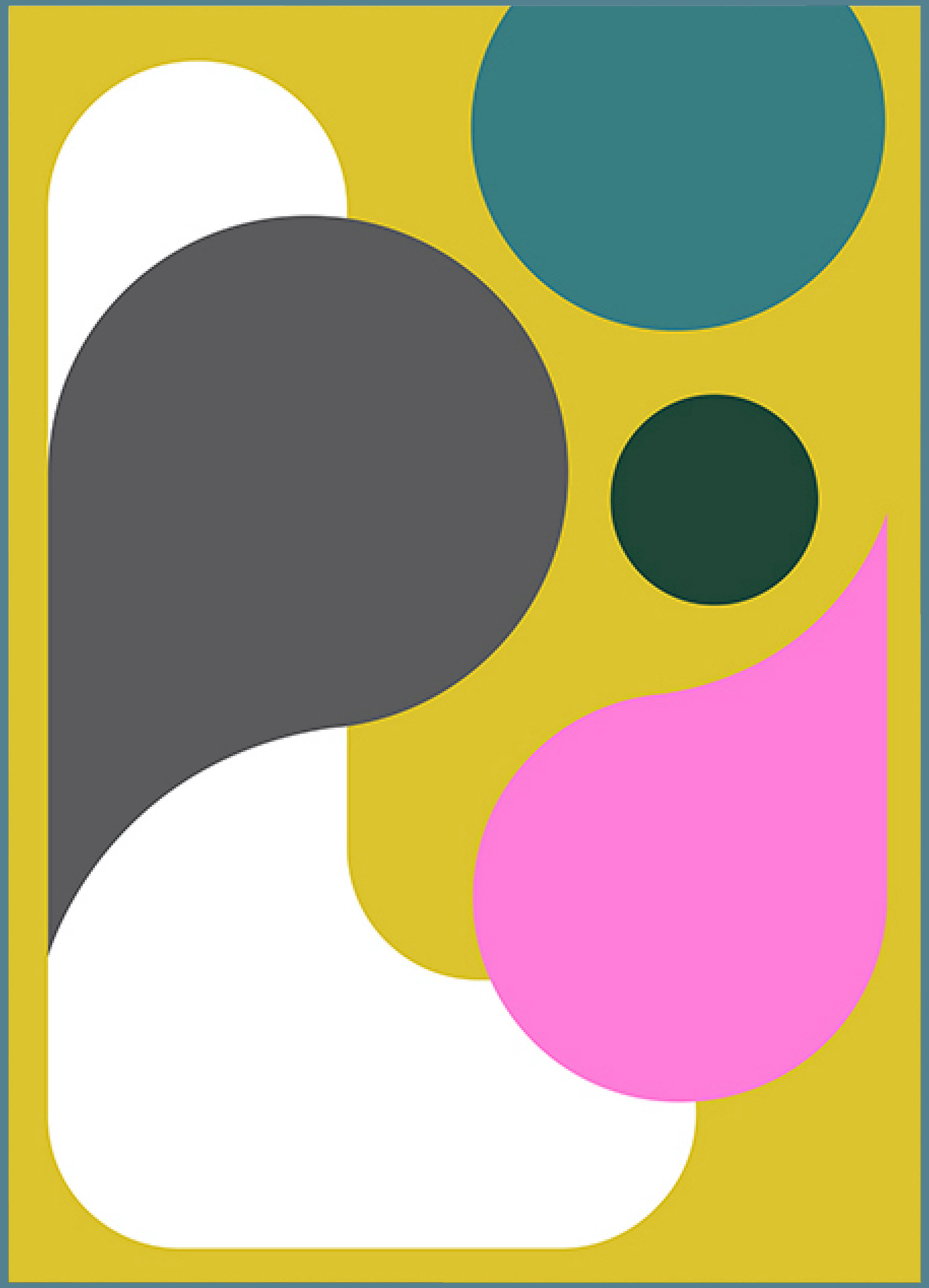
- Never introduce externally observable behavior

$\alpha' \leq \alpha$ $\quad \alpha \boxed{\xi} \omega \xrightarrow{\textbf{rewind}} \alpha' \boxed{\xi} \omega$

Relying on more being revealed

$\omega \leq \omega'$ $\quad \alpha \boxed{\xi} \omega \xrightarrow{\textbf{forward}} \alpha \boxed{\xi} \omega'$

Guaranteeing less being revealed



ABSTRACTION

ABSTRACT DENOTATIONAL SEMANTICS

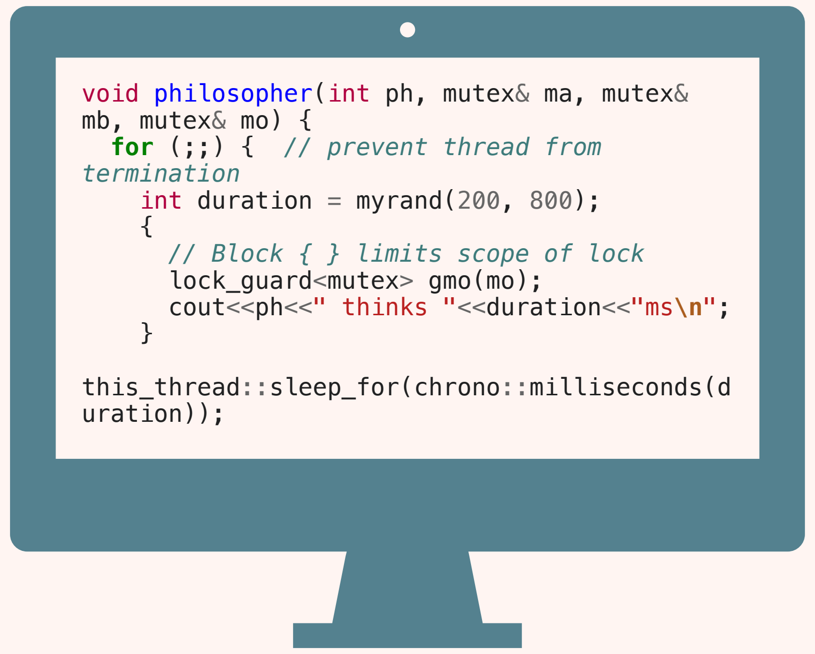
- **Brookes's denotations are fully-abstract**
  - **Proof relies on unrealistically holding exclusive access to the entire memory**
- **Instead, evidence based:**
  - **Which transformations can we validate? (look for counter-ex to full-abstraction)**

$$[\![M]\!] \supseteq [\![K]\!]$$

$\Downarrow$ Adequacy

$\Uparrow$ Full Abst.

$$M \twoheadrightarrow K$$



STRUCTURAL AND PARALLEL LAWS

Monad laws — structural equivalences for free, e.g. Hoisting

$$[\![\ \textbf{if}\ K_{\text{pure}}\ \textbf{then}\ M; P_1\ \textbf{else}\ M; P_2\ ]\!] = [\![\ M; \textbf{if}\ K_{\text{pure}}\ \textbf{then}\ P_1\ \textbf{else}\ P_2\ ]\!]$$

Interleaving — properties of parallel composition, e.g. generalized sequencing

$$[\![\ (M_1; M_2) \parallel (K_1; K_2)\ ]\!] \supseteq [\![\ (M_1 \parallel K_1); (M_2 \parallel K_2)\ ]\!]$$

SOPHISTICATION REQUIRED

Some transformations are valid due to **more complicated reasons**, e.g.:

Redundant Read Elimination

$$y?; M \twoheadrightarrow M$$

holds due to

delicate semantic invariants

Overwritten Write Elimination

$$x := 0; x := 1 \twoheadrightarrow x := 1$$

holds even though

state diverges

DELICATE SEMANTIC INVARIANTS

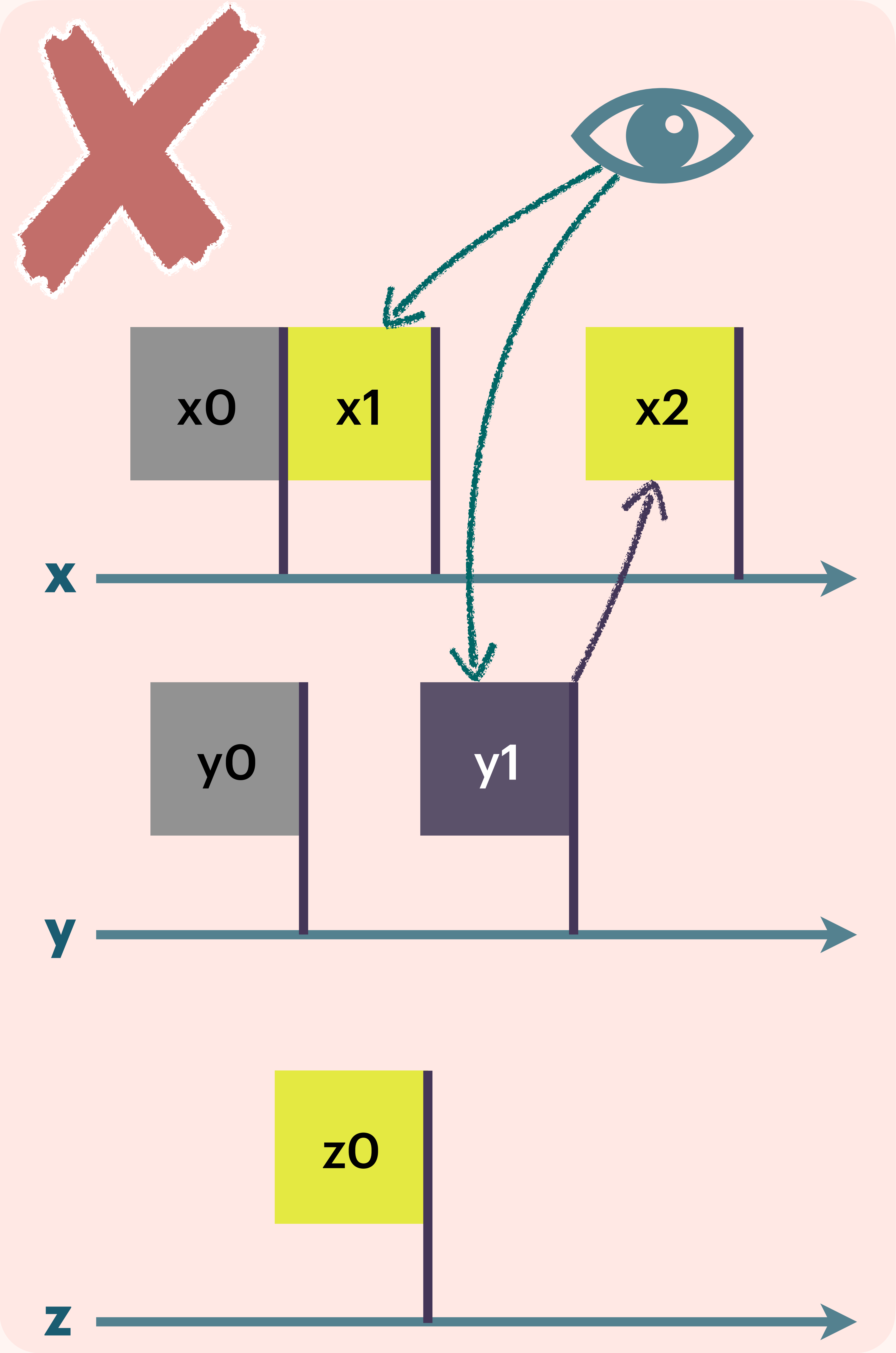
Redundant Read Elimination

$$y?; M \twoheadrightarrow M$$

we identify operational invariants

and impose them as denotational requirements

$$\kappa \boxed{\langle \mu, \mu \rangle} \kappa \therefore \langle\rangle \in [\![\langle\rangle]\!] \implies \exists v . \kappa \boxed{\langle \mu, \mu \rangle} \kappa \therefore v \in [\![y?]\!]$$



DIVERGING STATE

Overwritten Write Elimination

$$x := 0; x := 1 \twoheadrightarrow x := 1$$

DIVERGING STATE

Overwritten Write Elimination

$$x := 0; x := 1 \twoheadrightarrow x := 1$$

$$[\![x := 0; x := 1]\!] \supseteq [\![x := 1]\!]$$

DIVERGING STATE

Overwritten Write Elimination

$$x := 0; x := 1 \twoheadrightarrow x := 1$$

$$\alpha \boxed{\langle \mu, \mu \uplus \{\ 1\ \} \rangle} \omega \therefore \langle \rangle$$

$$\Cup$$

$$[\![x := 0; x := 1]\!] \supseteq [\![x := 1]\!]$$

DIVERGING STATE

Overwritten Write Elimination

$$x := 0; x := 1 \twoheadrightarrow x := 1$$

$$\alpha \left\langle \mu, \mu \uplus \{ \boxed{1} \} \right\rangle \omega \therefore \langle \rangle$$

$$\Cup$$

$$[\![x := 0; x := 1]\!] \supseteq [\![x := 1]\!]$$

$$\Cap$$

$$\alpha \left\langle \mu, \mu \uplus \{ \boxed{0} \} \right\rangle \left\langle \mu \uplus \{ \boxed{0} \}, \mu \uplus \{ \boxed{0}\,\boxed{1} \} \right\rangle \omega \therefore \langle \rangle$$

DIVERGING STATE

Overwritten Write Elimination

$$x := 0; x := 1 \twoheadrightarrow x := 1$$

$$\alpha \langle \mu, \mu \uplus \{ \boxed{1} \} \rangle \omega \therefore \langle \rangle$$

$$\Cup$$

$$[\![x := 0; x := 1]\!] \supseteq [\![x := 1]\!]$$

$$\Cap$$

$$\alpha \langle \mu, \mu \uplus \{ \boxed{0} \boxed{1} \} \rangle \omega \therefore \langle \rangle$$

mumble

$$\alpha \langle \mu, \mu \uplus \{ \boxed{0} \} \rangle \langle \mu \uplus \{ \boxed{0} \}, \mu \uplus \{ \boxed{0} \boxed{1} \} \rangle \omega \therefore \langle \rangle$$

DIVERGING STATE

Overwritten Write Elimination

$$x := 0; x := 1 \twoheadrightarrow x := 1$$

$$\alpha \langle \mu, \mu \uplus \{ \boxed{1} \} \rangle \omega \therefore \langle \rangle$$

absorb

$$[\![x := 0; x := 1]\!] \sqsupseteq [\![x := 1]\!]$$

$$\alpha \langle \mu, \mu \uplus \{ \boxed{0}\boxed{1} \} \rangle \omega \therefore \langle \rangle$$

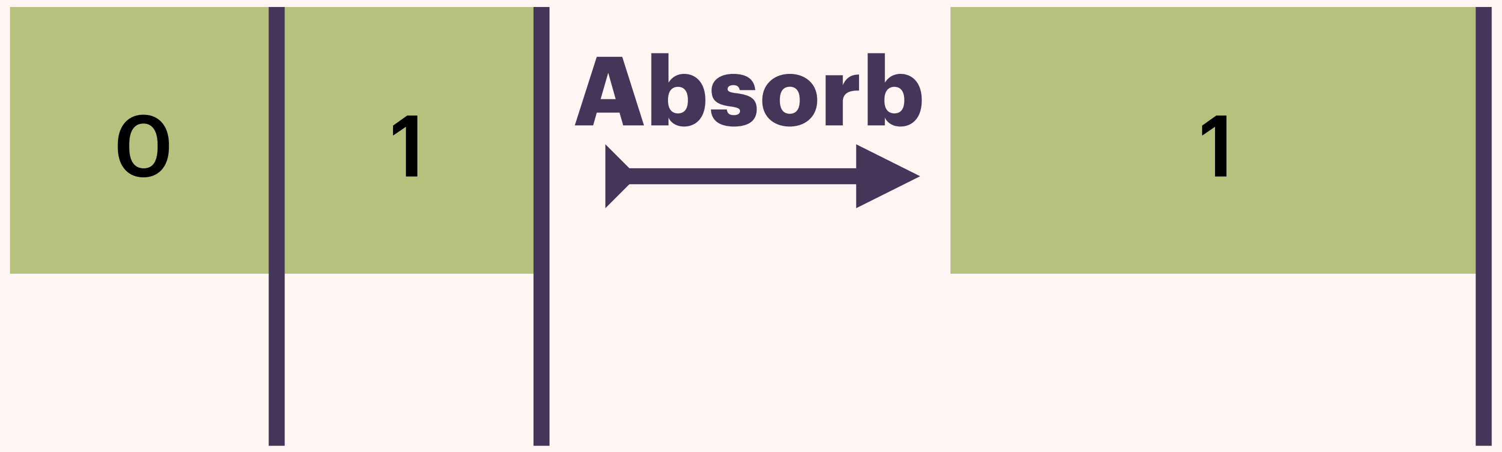
mumble

$$\alpha \langle \mu, \mu \uplus \{ \boxed{0} \} \rangle \langle \mu \uplus \{ \boxed{0} \}, \mu \uplus \{ \boxed{0}\boxed{1} \} \rangle \omega \therefore \langle \rangle$$

NO CORRESPONDENCE WITH INTERRUPTED EXECUTIONS

$$\alpha\,\langle\mu_1,\rho_1\rangle\langle\mu_2,\rho_2\rangle\ldots\langle\mu_{n-1},\rho_{n-1}\rangle\langle\mu_n,\rho_n\rangle\,\omega \therefore r$$

$$\cdots\ \langle\mu_2, -\rangle, M_1 \to^* \langle\rho_2, -\rangle, M_2\ \cdots$$



ALL REWRITE RULES

Loosen	$\alpha\,\boxed{\xi\,(\eta \mathbin{\overline{\uplus}} \{\epsilon\})}\,\omega$	$\xrightarrow{\text{Ls}}$	$\alpha\,\boxed{\xi\,(\eta \mathbin{\overline{\uplus}} \{\nu\})}\,\omega$	$\nu \le_{\mathtt{vw}} \epsilon$
Expel	$\alpha\,\boxed{\xi\,(\eta \mathbin{\overline{\uplus}} \{\epsilon_{\mathtt{i}}^{\nu.\mathtt{i}}\})}\,\omega$	$\xrightarrow{\text{Ex}}$	$\alpha\,\boxed{\xi\,(\eta \mathbin{\overline{\uplus}} \{\nu, \epsilon\})}\,\omega$	$\nu \mathrel{-\!\!\subset} \epsilon$
Condense	$\alpha\,\boxed{\xi\,(\eta \mathbin{\overline{\uplus}} \{\nu, \epsilon\})}\,\omega$	$\xrightarrow{\text{Cn}}$	$\left(\alpha\,\boxed{\xi\,(\eta \mathbin{\overline{\uplus}} \{\nu\})}\,\omega\right)[\uparrow\epsilon]$	$\nu \mathrel{-\!\!\in} \epsilon$
Stutter	$\alpha\,\boxed{\xi\eta}\,\omega$	$\xrightarrow{\text{St}}$	$\alpha\,\boxed{\xi\,\langle\mu,\mu\rangle\,\eta}\,\omega$	
Mumble	$\alpha\,\boxed{\xi\,\langle\mu,\rho\rangle\,\langle\rho,\theta\rangle\,\eta}\,\omega$	$\xrightarrow{\text{Mu}}$	$\alpha\,\boxed{\xi\,\langle\mu,\theta\rangle\,\eta}\,\omega$	
Rewind	$\kappa\,\boxed{\xi}\,\omega$	$\xrightarrow{\text{Rw}}$	$\alpha\,\boxed{\xi}\,\omega$	$\alpha \le \kappa$
Forward	$\alpha\,\boxed{\xi}\,\kappa$	$\xrightarrow{\text{Fw}}$	$\alpha\,\boxed{\xi}\,\omega$	$\kappa \le \omega$
Tighten	$\alpha\,\boxed{\xi\,\langle\mu, \rho \uplus \{\nu\}\rangle\,\eta \mathbin{\overline{\uplus}} \{\nu\}}\,\omega$	$\xrightarrow{\text{Ti}}$	$\alpha\,\boxed{\xi\,\langle\mu, \rho \uplus \{\epsilon\}\rangle\,\eta \mathbin{\overline{\uplus}} \{\epsilon\}}\,\omega$	$\nu \le_{\mathtt{vw}} \epsilon$
Absorb	$\alpha\,\boxed{\xi\,\langle\mu, \rho \uplus \{\nu, \epsilon\}\rangle\,\eta \mathbin{\overline{\uplus}} \{\nu, \epsilon\}}\,\omega$	$\xrightarrow{\text{Ab}}$	$\alpha\,\boxed{\xi\,\left\langle\mu, \rho \uplus \left\{\epsilon_{\mathtt{i}}^{\nu.\mathtt{i}}\right\}\right\rangle\,\eta \mathbin{\overline{\uplus}} \left\{\epsilon_{\mathtt{i}}^{\nu.\mathtt{i}}\right\}}\,\omega$	$\nu \mathrel{-\!\!\subset} \epsilon$
Dilute	$\left(\alpha\,\boxed{\xi\,\langle\mu, \rho \uplus \{\nu\}\rangle\,\eta \mathbin{\overline{\uplus}} \{\nu\}}\,\omega\right)[\uparrow\epsilon]$	$\xrightarrow{\text{Di}}$	$\alpha\,\boxed{\xi\,\langle\mu, \rho \uplus \{\nu, \epsilon\}\rangle\,\eta \mathbin{\overline{\uplus}} \{\nu, \epsilon\}}\,\omega$	$\nu \mathrel{-\!\!\in} \epsilon$

NEW ADEQUACY PROOF IDEA

Traces are not operational — adequacy proof is *significantly* more challenging:

1. **We first define a denotational semantics $\underline{[\![M]\!]}$ but without the abstract rules**

2. **We show it is adequate — easier: traces correspond to interrupted executions**
   (with an admissible view-advancing rule)

3. **We show it is enough to apply the abstract closure $(-)^{\mathfrak{a}}$ on top** $\quad [\![M]\!] = \underline{[\![M]\!]}^{\mathfrak{a}}$

   - ***This is the main technical challenge — complicated commutativity property***

4. **We show that the abstract deduction rules preserve observable results**
   (rather than interrupted executions)

Law				
Laws of Parallel Programming				
Symmetry	$M \parallel N$	$\twoheadrightarrow$	$\mathbf{match}\, N \parallel M \,\mathbf{with}\, \langle y, x\rangle .\ \langle x, y\rangle$	
Generalized Sequencing	$(\mathbf{let}\, x = M_1 \,\mathbf{in}\, M_2) \parallel (\mathbf{let}\, y = N_1 \,\mathbf{in}\, N_2)$	$\twoheadrightarrow$	$\mathbf{match}\, M_1 \parallel N_1 \,\mathbf{with}\, \langle x, y\rangle .\, M_2 \parallel N_2$	
Eliminations				
Irrelevant Read	$\ell? \,;\, \langle\rangle$	$\twoheadrightarrow$	$\langle\rangle$	
Write-Write	$\ell := v \,;\, \ell := w$	$\overset{\mathsf{Ab}}{\twoheadrightarrow}$	$\ell := w$	
Write-Read	$\ell := v \,;\, \ell?$	$\twoheadrightarrow$	$\ell := v \,;\, v$	
Write-FAA	$\ell := v \,;\, \mathrm{FAA}\,(\ell, w)$	$\overset{\mathsf{Ab}}{\twoheadrightarrow}$	$\ell := (v + w) \,;\, v$	
Read-Write	$\mathbf{let}\, x = \ell? \,\mathbf{in}\, \ell := (x + v) \,;\, x$	$\twoheadrightarrow$	$\mathrm{FAA}\,(\ell, v)$	
Read-Read	$\langle \ell?, \ell? \rangle$	$\twoheadrightarrow$	$\mathbf{let}\, x = \ell? \,\mathbf{in}\, \langle x, x\rangle$	
Read-FAA	$\langle \ell?, \mathrm{FAA}\,(\ell, v)\rangle$	$\twoheadrightarrow$	$\mathbf{let}\, x = \mathrm{FAA}\,(\ell, v) \,\mathbf{in}\, \langle x, x\rangle$	
FAA-Read	$\langle \mathrm{FAA}\,(\ell, v), \ell? \rangle$	$\twoheadrightarrow$	$\mathbf{let}\, x = \mathrm{FAA}\,(\ell, v) \,\mathbf{in}\, \langle x, x + v\rangle$	
FAA-FAA	$\langle \mathrm{FAA}\,(\ell, v), \mathrm{FAA}\,(\ell, w)\rangle$	$\overset{\mathsf{Ab}}{\twoheadrightarrow}$	$\mathbf{let}\, x = \mathrm{FAA}\,(\ell, v + w) \,\mathbf{in}\, \langle x, x + v\rangle$	
Others				
Irrelevant Read Introduction	$\langle\rangle$	$\twoheadrightarrow$	$\ell? \,;\, \langle\rangle$	
Read to FAA	$\ell?$	$\overset{\mathsf{Di}}{\twoheadrightarrow}$	$\mathrm{FAA}\,(\ell, 0)$	
Write-Read Deorder	$\langle (\ell := v), \ell'? \rangle$	$\overset{\mathsf{Ti}}{\twoheadrightarrow}$	$(\ell := v) \parallel \ell'?$	$(\ell \neq \ell')$
Write-Read Reorder	$\langle (\ell := v), \ell'? \rangle$	$\overset{\mathsf{Ti}}{\twoheadrightarrow}$	$\mathbf{let}\, x = \ell'? \,\mathbf{in}\, (\ell := v) \,;\, x$	$(\ell \neq \ell')$

Similarly for other Laws of Par. Prog.

Similarly for other RMWs

SUMMARY

- Standard, adequate and fully-compositional denotational semantic for RA
- Sufficiently abstract: validates all RA transformations that we know of
  (memory access, laws of parallel programming, structural transformations)
- More nuanced, complicated traces
  - interpreted as Rely/Guarantee sequences
  - denotations closed under 10 trace deduction rules
- Extended RA view-based machine with compositional (i.e. first-class) parallelism
  (weak-memory models are usually studied with top-level parallelism)

Brookes's Denotational Semantics for Shared State Concurrency

Thank you!

Algebraic Effects Refinement

Relaxed Memory Extension