
A DENOTATIONAL APPROACH TO RELEASE/ACQUIRE CONCURRENCY

GOAL



RELEASE/ACQUIRE

WHY RELEASE/ACQUIRE?



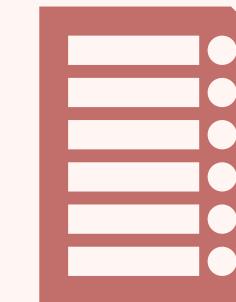
RA is an important fragment of C/C++, enables decentralized architectures (POWER)



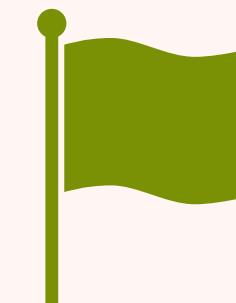
First adaptation of Brookes's traces to a software model (compositional parallelism)



Intricate causal semantics, not overwhelmingly detailed



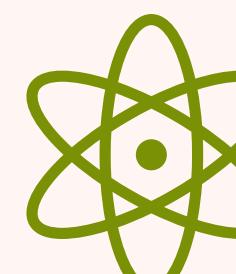
Threads can disagree about the order of writes (non-multi-copy-atomic)



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



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

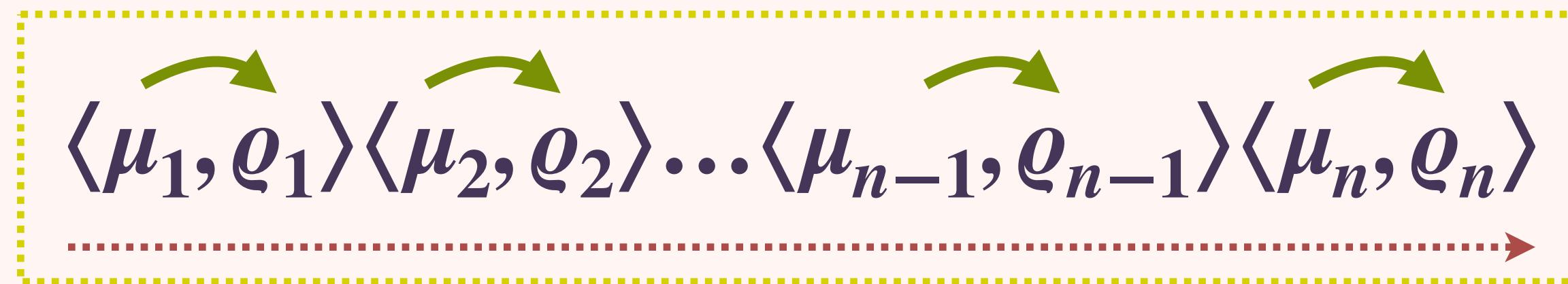


Supports read-modify-write atomicity

TRACE-BASED SEMANTICS

Brookes [1996]

Main ingredient: **linearly-ordered traces of state-transitions that sequence and interleave**



$\langle \mu_1, \mu'_1 \rangle \langle \mu_2, \mu'_2 \rangle \dots \langle \mu_n, \mu'_n \rangle$

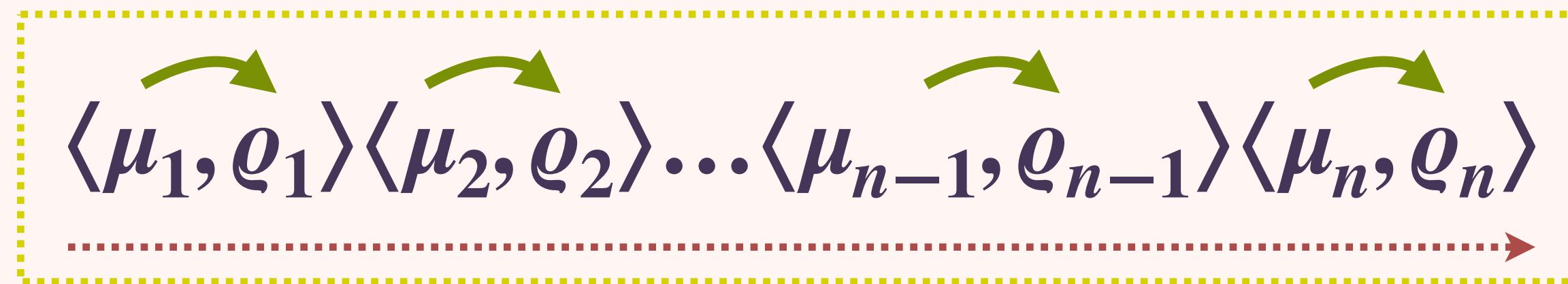
$\langle \varrho_1, \varrho'_1 \rangle \langle \varrho_2, \varrho'_2 \rangle \dots \langle \varrho_n, \varrho'_n \rangle$

SEQUENCE

TRACE-BASED SEMANTICS

Brookes [1996]

Main ingredient: **linearly-ordered traces of state-transitions that sequence and interleave**



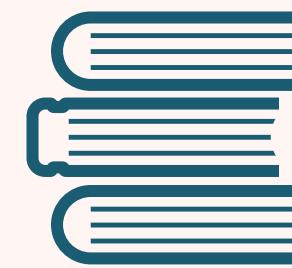
$\langle \mu_1, \mu'_1 \rangle$ $\langle \mu_2, \mu'_2 \rangle$... $\langle \mu_n, \mu'_n \rangle$

$\langle \varrho_1, \varrho'_1 \rangle$ $\langle \varrho_2, \varrho'_2 \rangle$... $\langle \varrho_n, \varrho'_n \rangle$

...

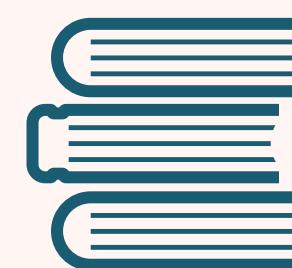
INTERLEAVE

TRACE-BASED SEMANTICS



Brookes [1996]

- Denotational semantics $[] - []$ for concurrency
- Idealized model - Sequential Consistency (SC)
- Follows operational semantics



Jagadeesan, Petri, Riely [2012]

- Adapts traces to TSO (hardware model)
- Follows operational semantics too
- Relatively close to SC

Main ingredient: linearly-ordered traces of state-transitions that sequence and interleave

$$\langle \mu_1, \varrho_1 \rangle \langle \mu_2, \varrho_2 \rangle \dots \langle \mu_{n-1}, \varrho_{n-1} \rangle \langle \mu_n, \varrho_n \rangle$$



This work

- Adapts traces to RA (software model)
- Kang et al. [2017] operational presentation
- Much more complex notion of state

CONTRIBUTION

Directionally Adequate

denotational semantics for RA based on linearly-ordered traces

$$\boxed{\begin{array}{ccc} \text{Refinement} & & \text{Transformation} \\ \llbracket M \rrbracket \supseteq \llbracket K \rrbracket & \implies & M \twoheadrightarrow K \end{array}}$$

Standard (CbV) semantics [Moggi 1991]

enables structural transformations (e.g. $\llbracket K; (M; N) \rrbracket = \llbracket (K; M); N \rrbracket$)

has higher-order functions for free

etc.

**Abstract enough to justify every transformation discussed
in the literature that we know of (but no full-abstraction)**

New challenge — non-operational interpretation:

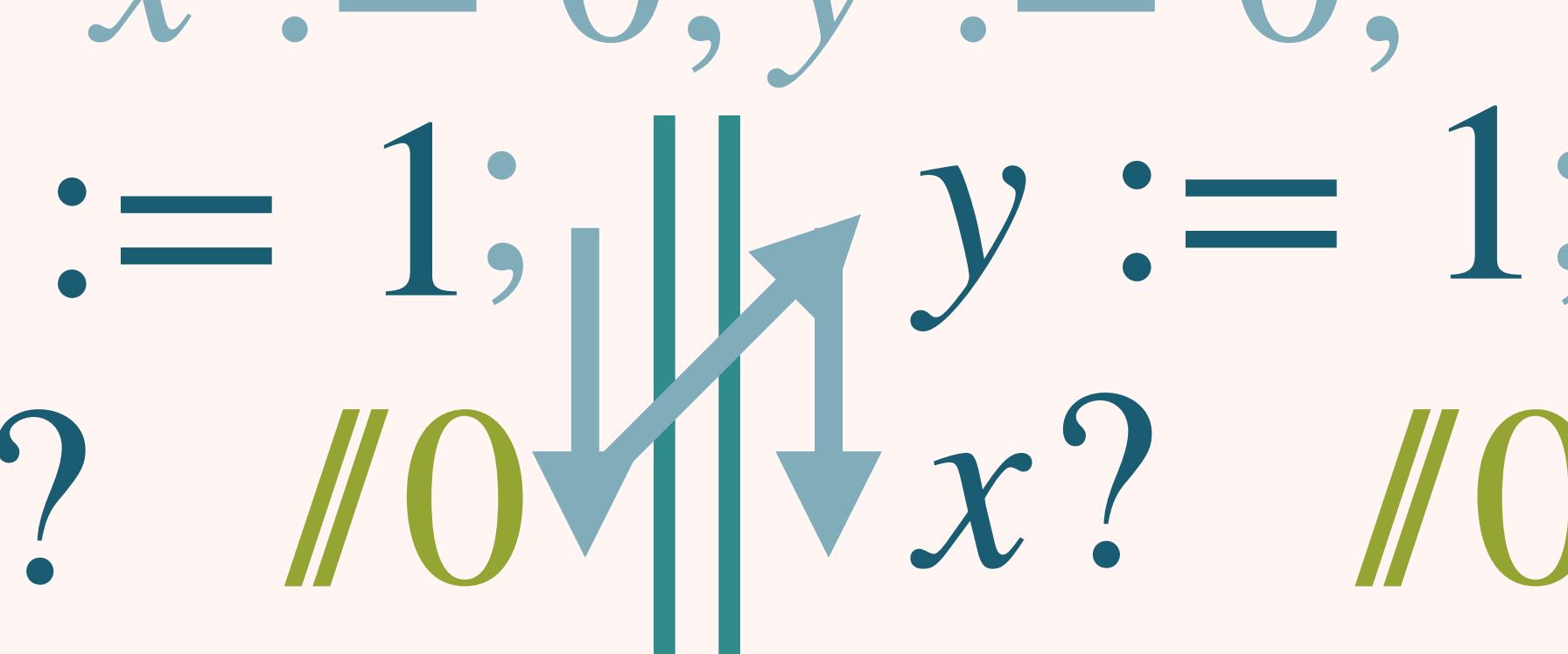
each trace represents a possible behavior as a Rely/Guarantee sequence

RELEASE/ACQUIRE

TYPICAL EXAMPLES

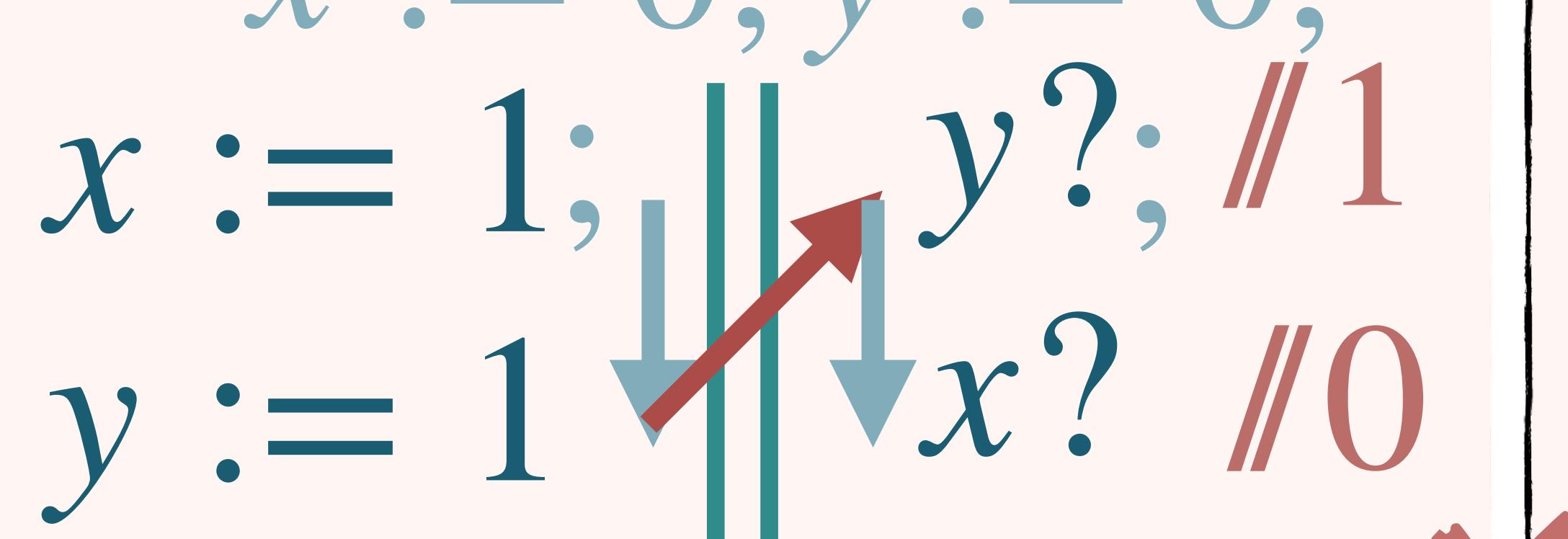
Store Buffering

*Propagation is
not instant*

$$\begin{array}{c} x := 0; y := 0; \\ x := 1; \quad \parallel \quad y := 1; \\ y? \quad //0 \quad \parallel \quad x? \quad //0 \end{array}$$


Message Pass

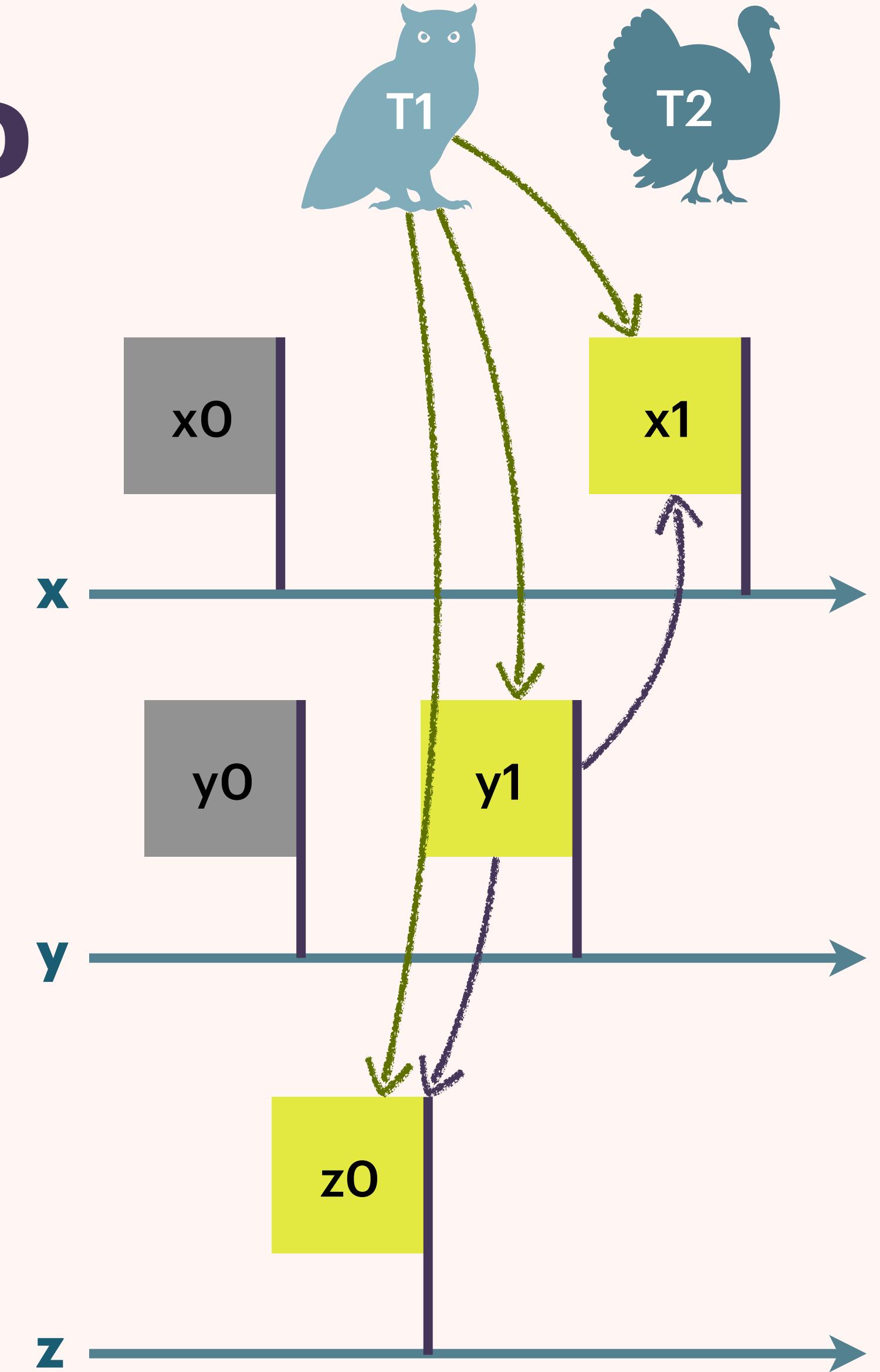
*Propagation
respects causality*

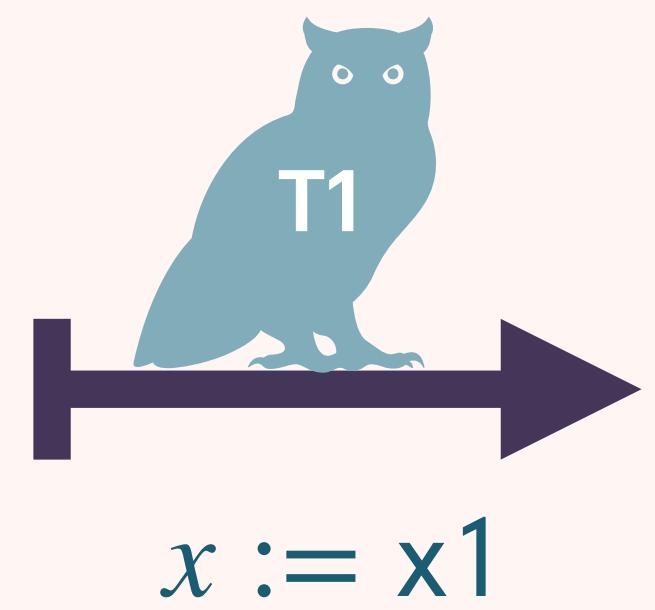
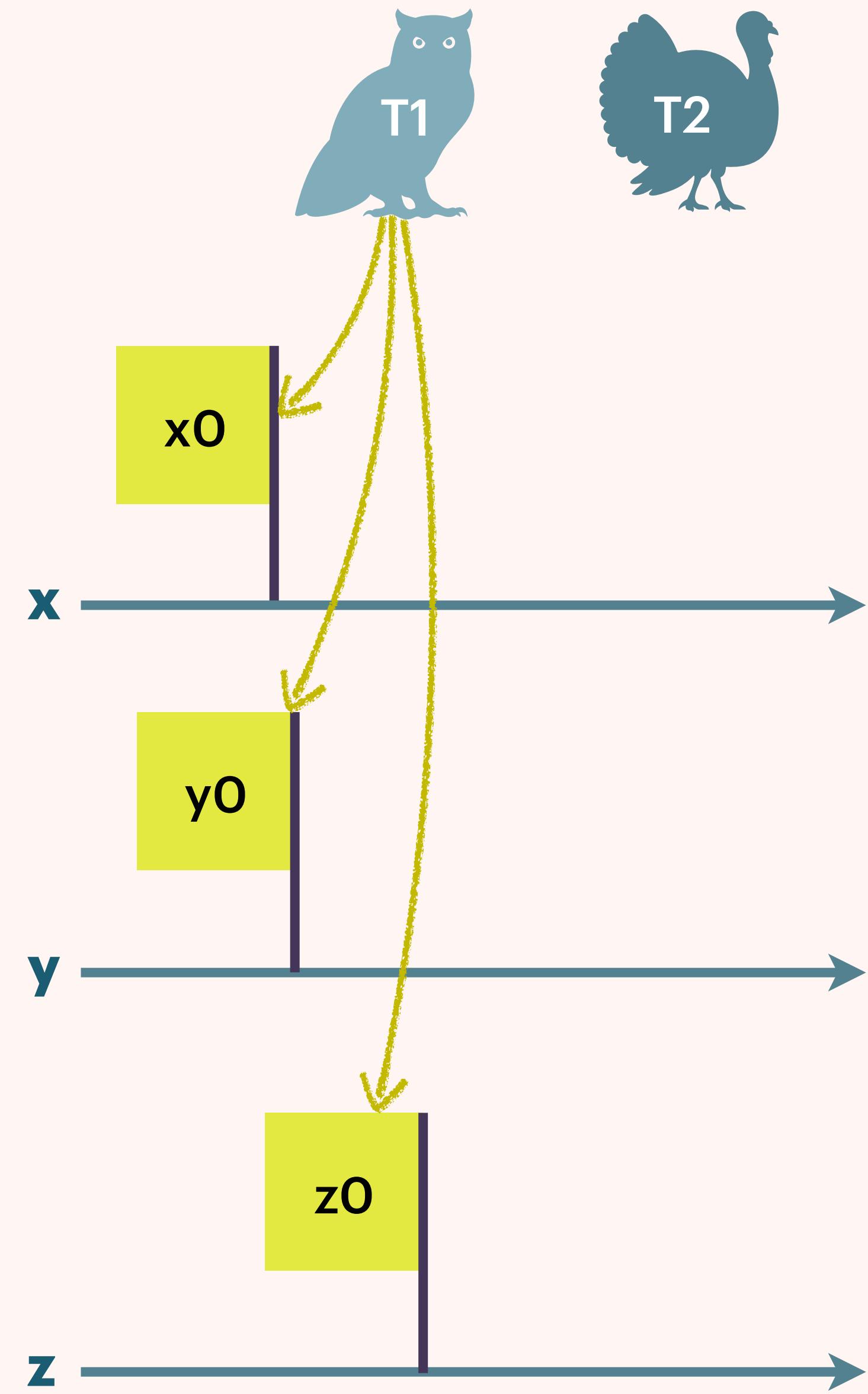
$$\begin{array}{c} x := 0; y := 0; \\ x := 1; \quad \parallel \quad y?; //1 \\ y := 1 \quad \downarrow \quad \parallel \quad x? \quad //0 \end{array}$$


RELEASE/ACQUIRE VIEW-BASED OPERATIONAL SEMANTICS

Kang et al. [2017]

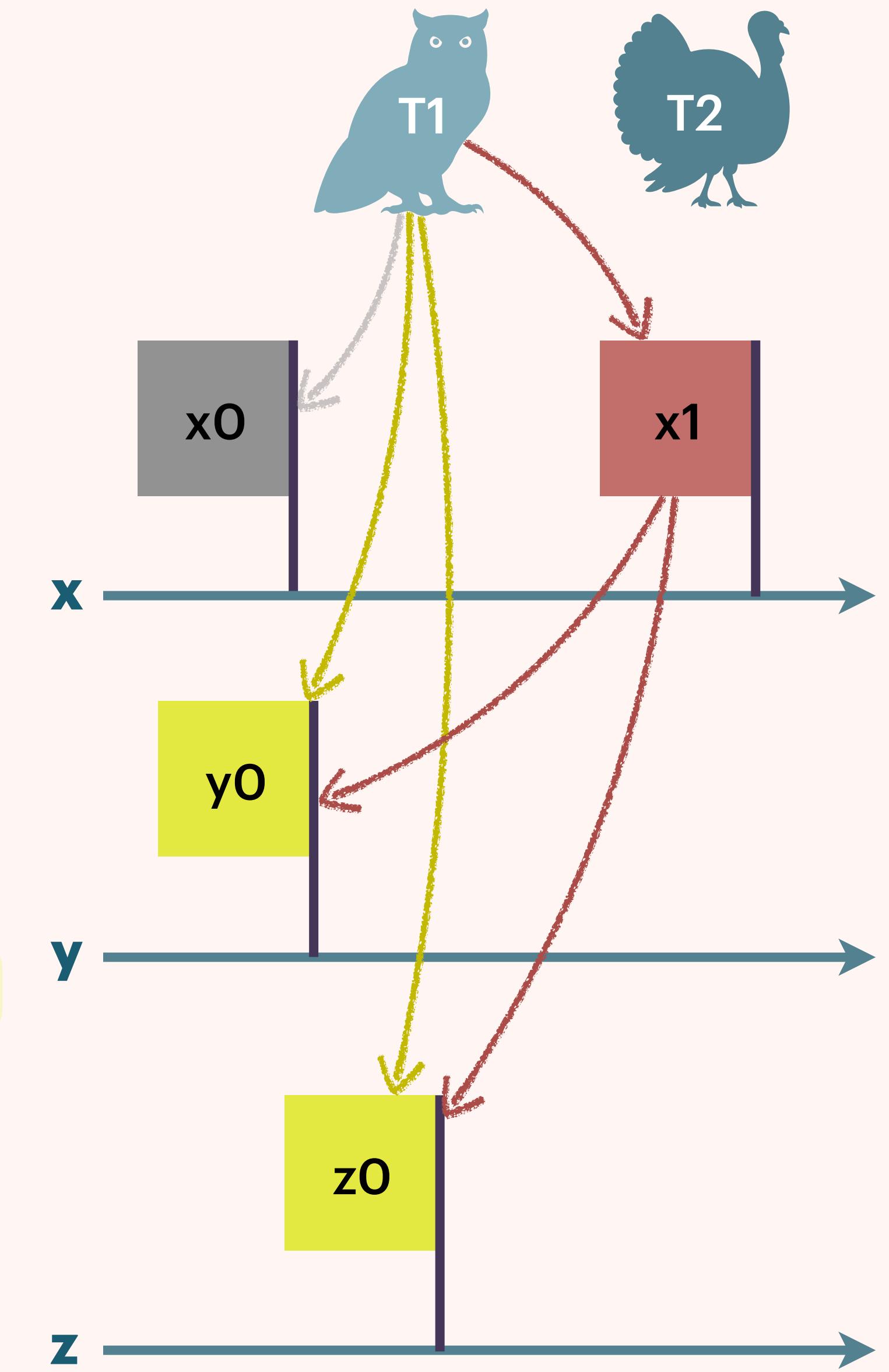
- **Memory:** Timeline per location (e.g. x, y, z)
- Populated with immutable messages (e.g. x0, y0, z0)
- Each thread's view points to a msgs on each timeline (e.g. T1)
- Thread's cannot read from "the past"
- Each msg's view points to a msg on each other timelines (e.g. y1)
- Message views are used for enforcing causal propagation

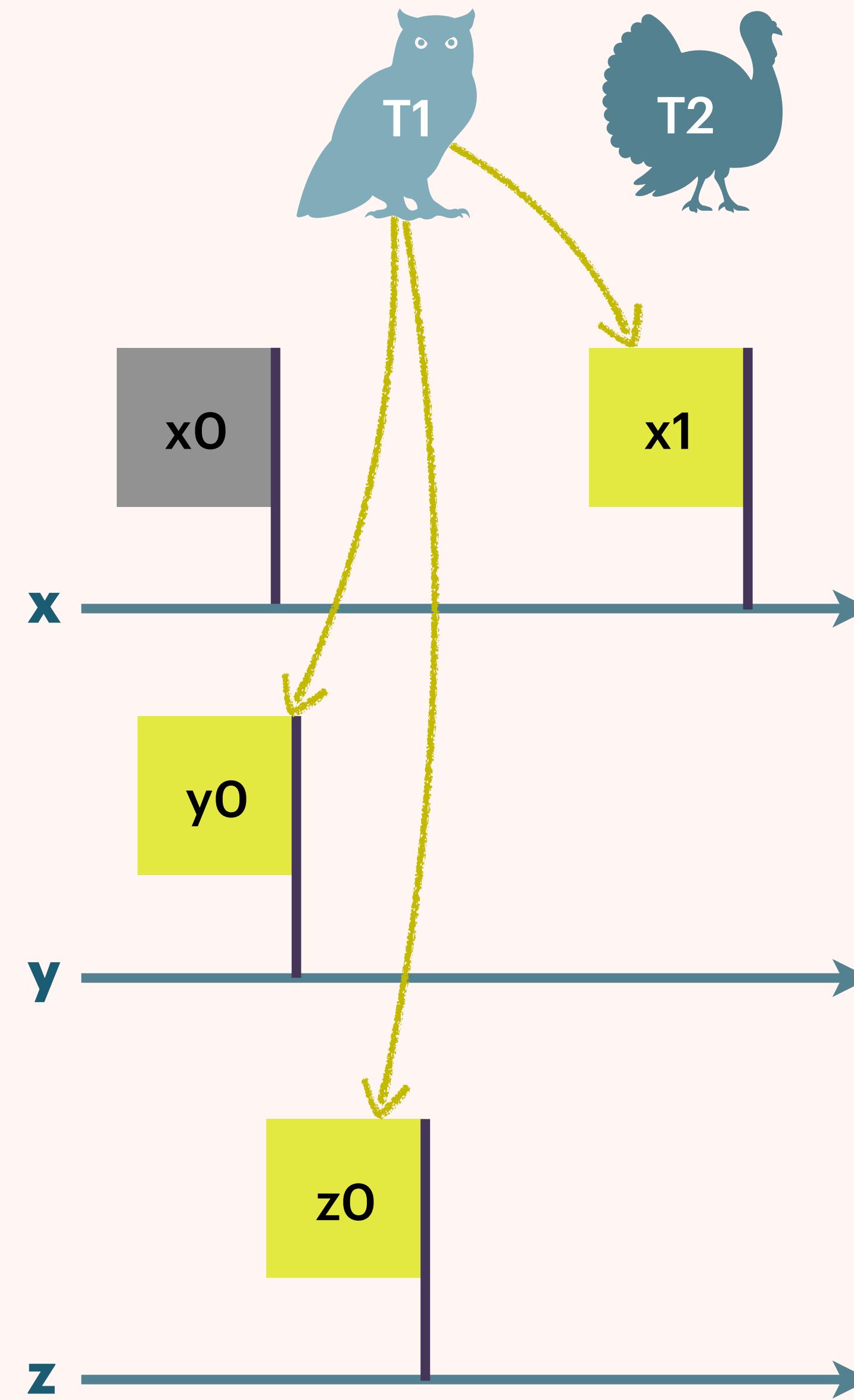




When writing, the message:

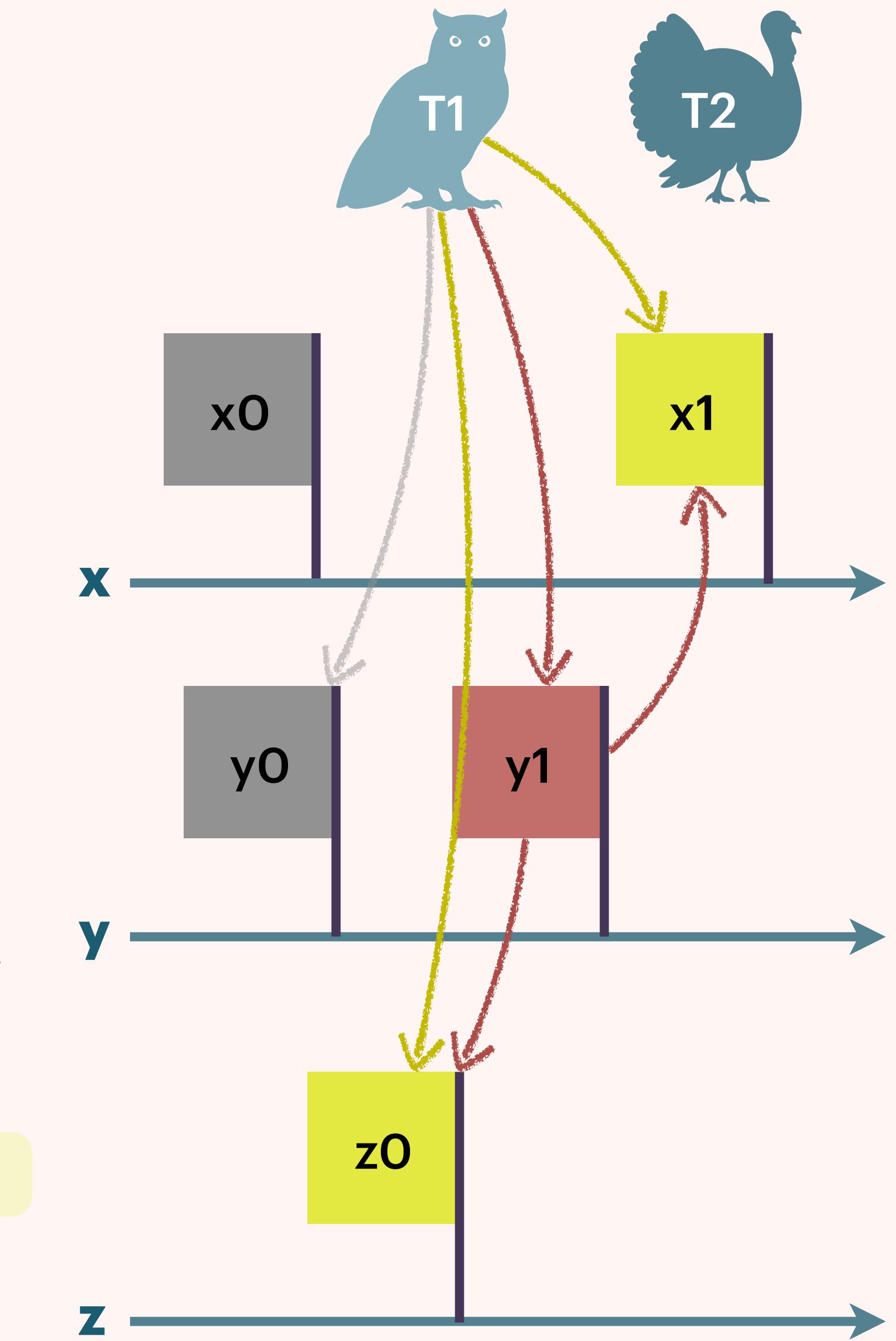
- must be placed after thread's view
- may be placed before others
- copies thread's view

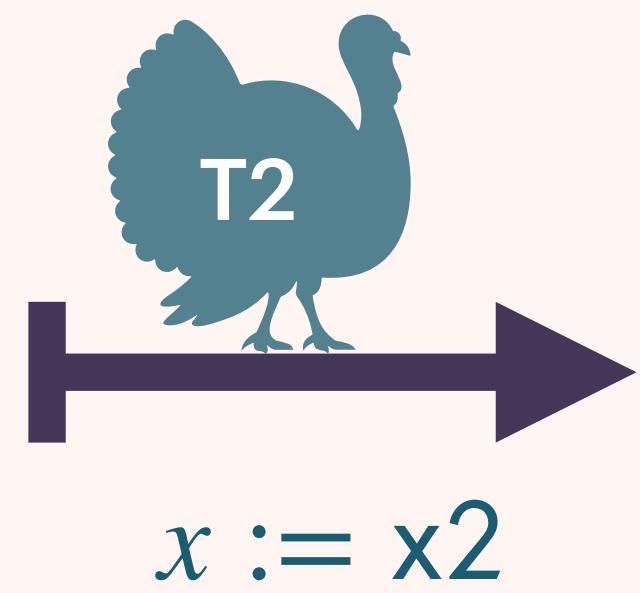
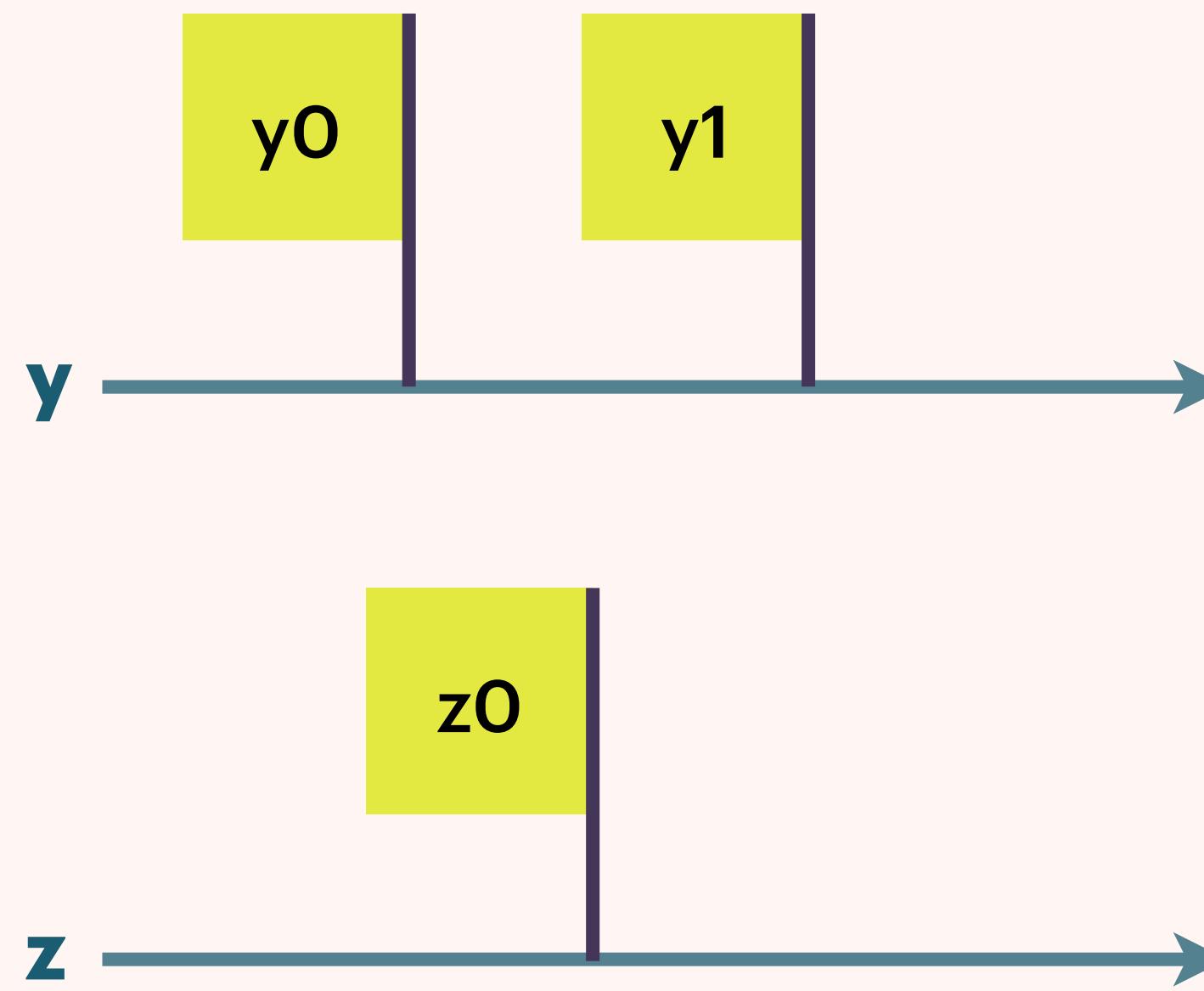
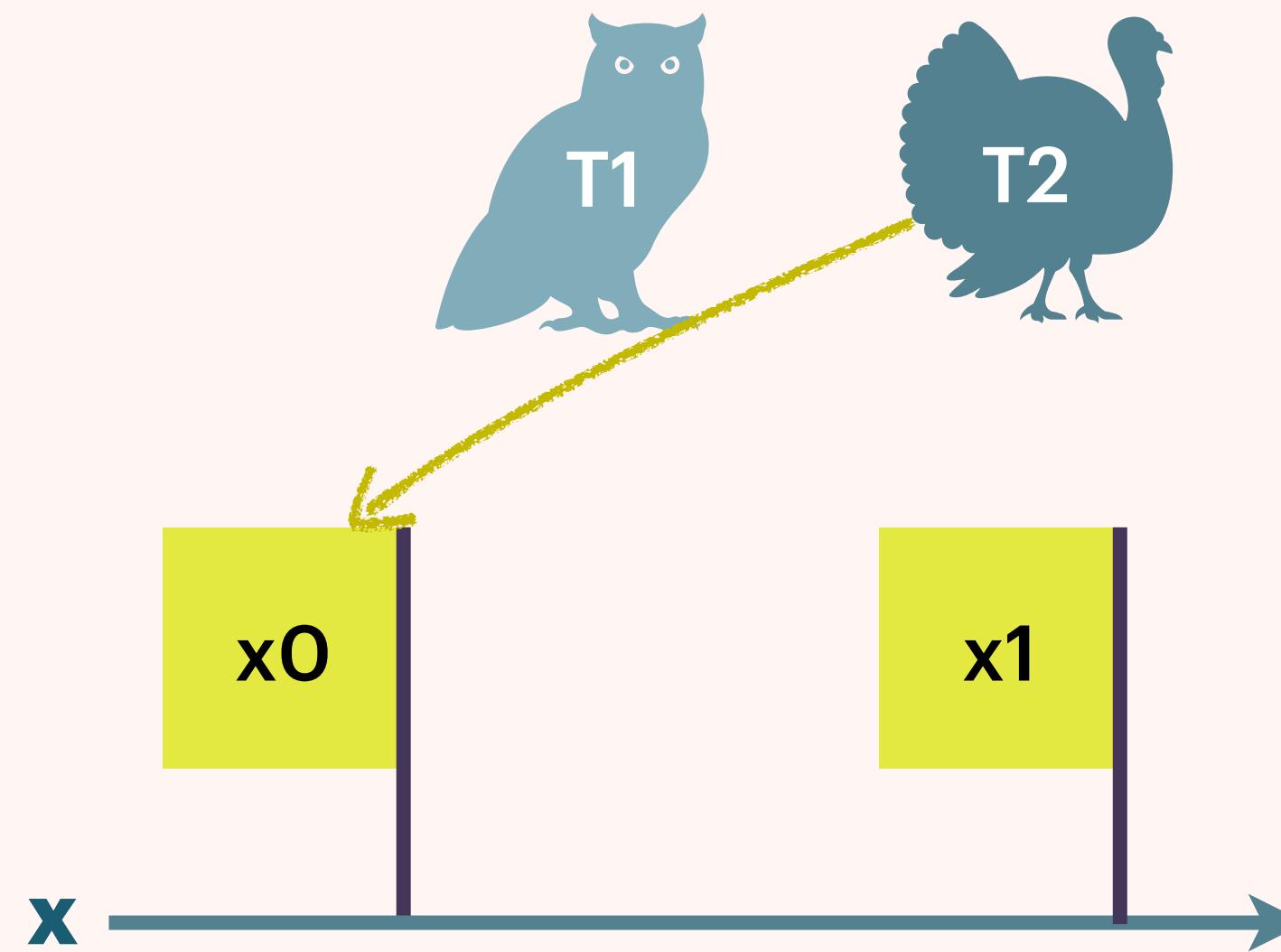




When writing, the message:

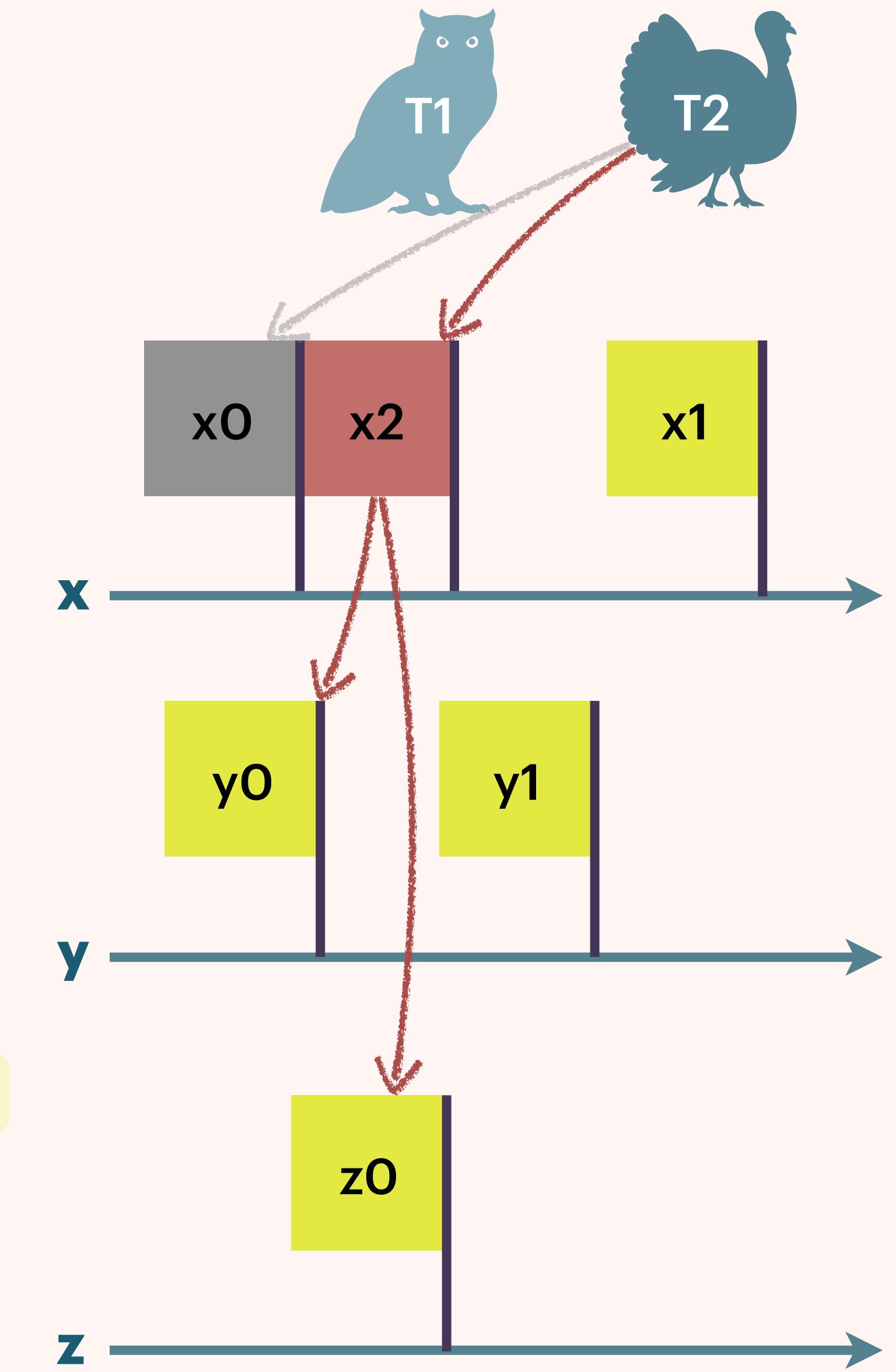
- must be placed after thread's view
- may be placed before others
- copies thread's view

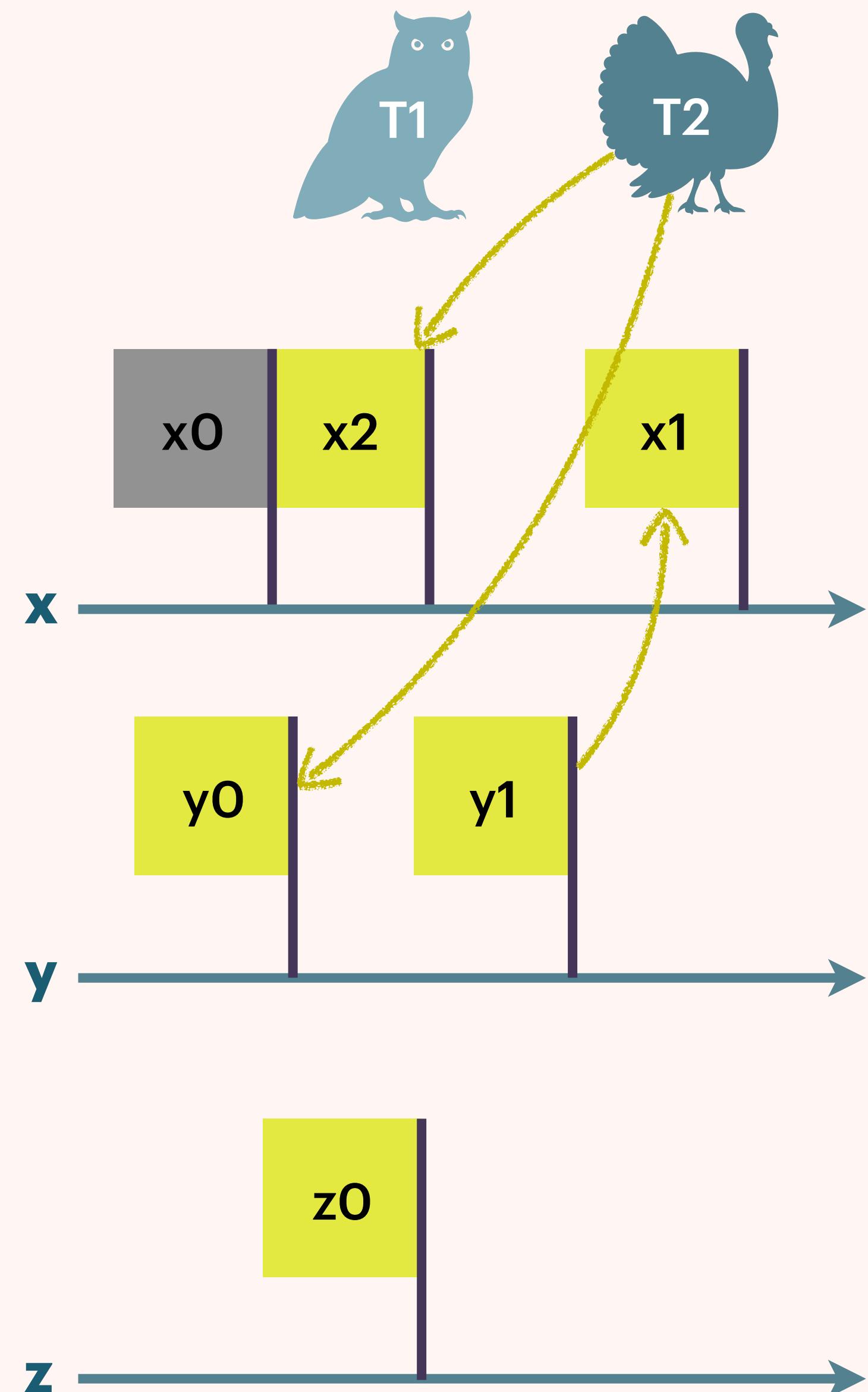




When writing, the message:

- must be placed after thread's view
- may be placed before others
- copies thread's view



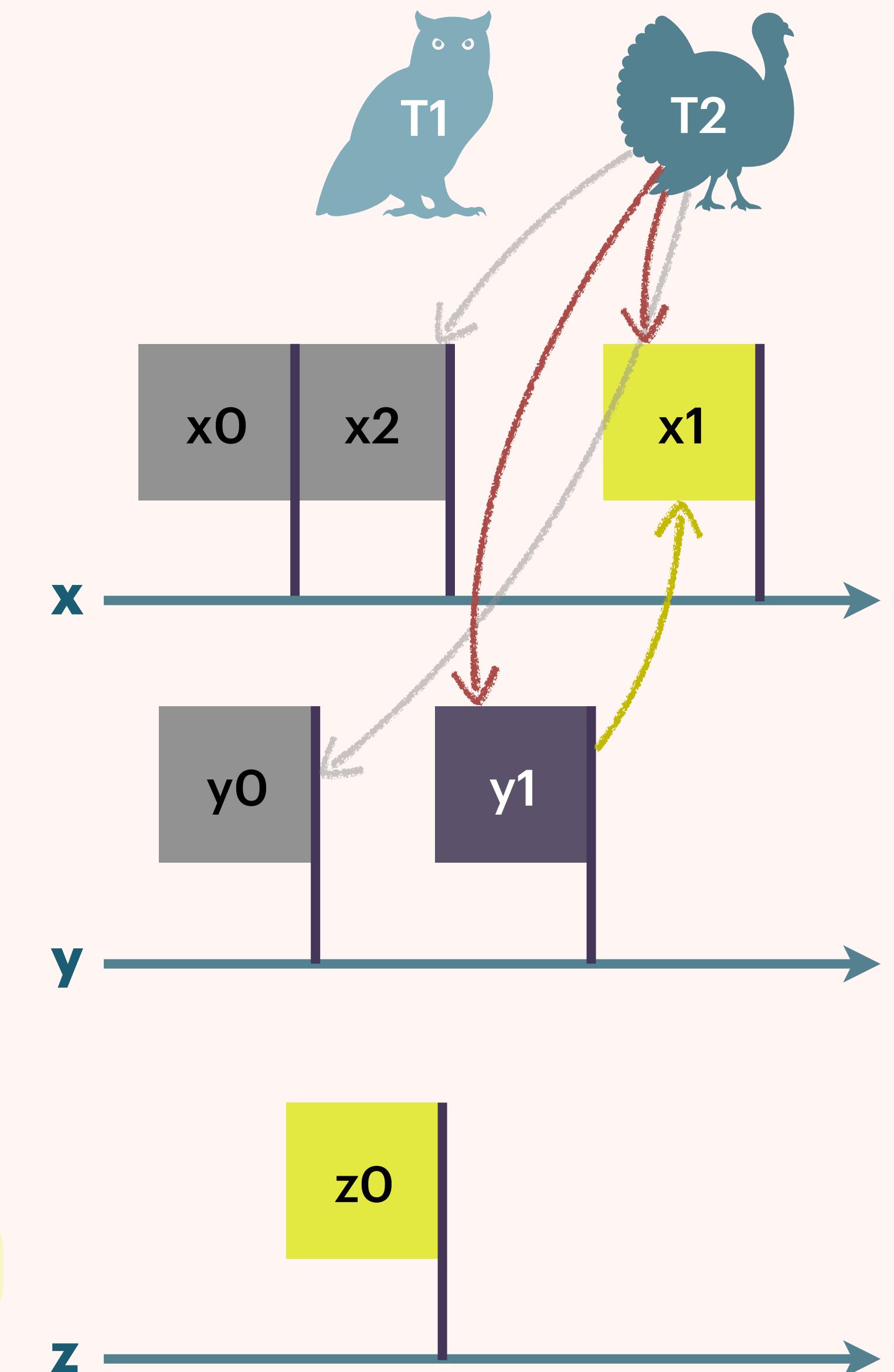


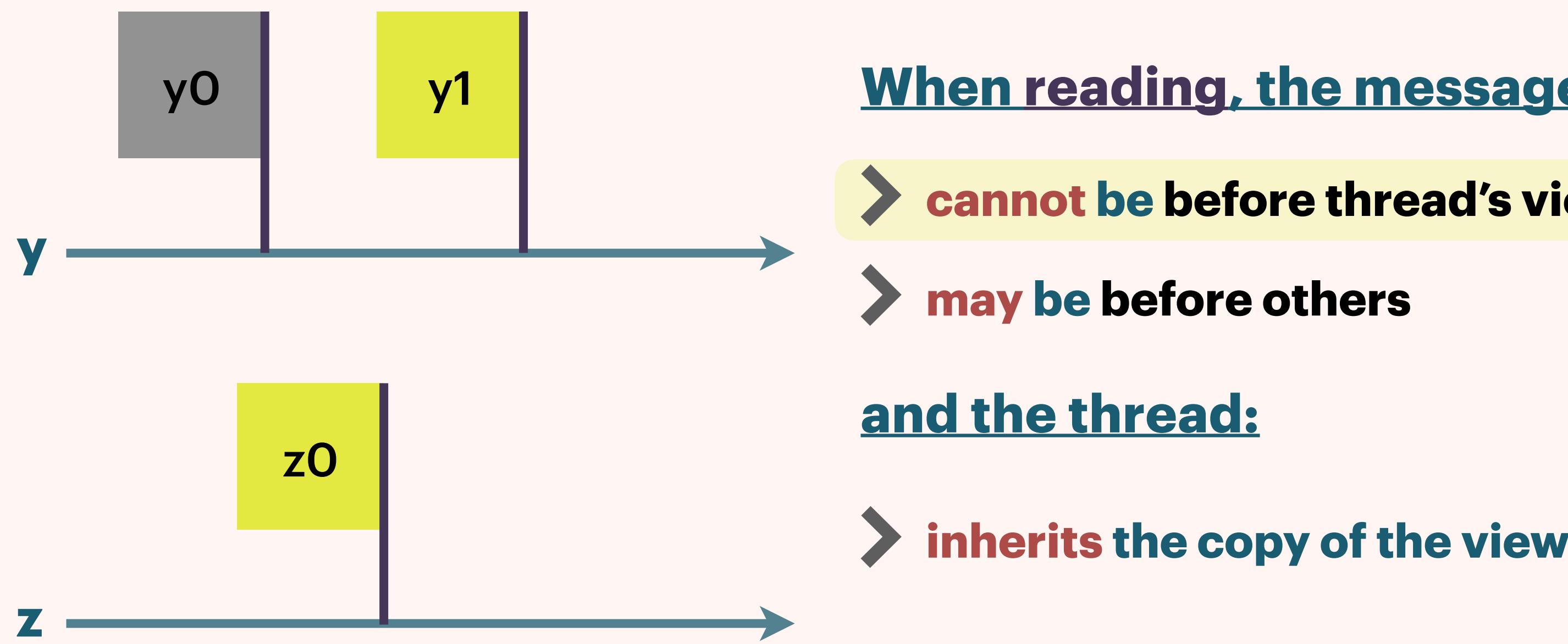
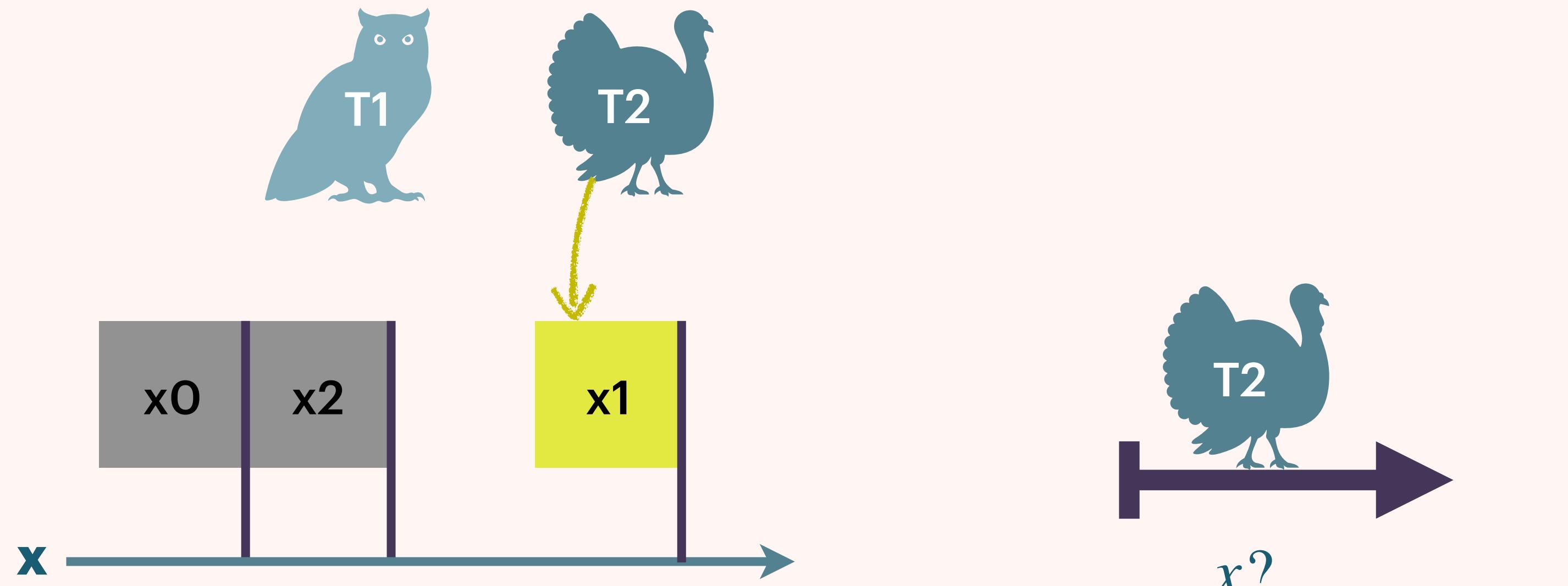
When reading, the message:

- cannot be before thread's view
- may be before others

and the thread:

- inherits the copy of the view



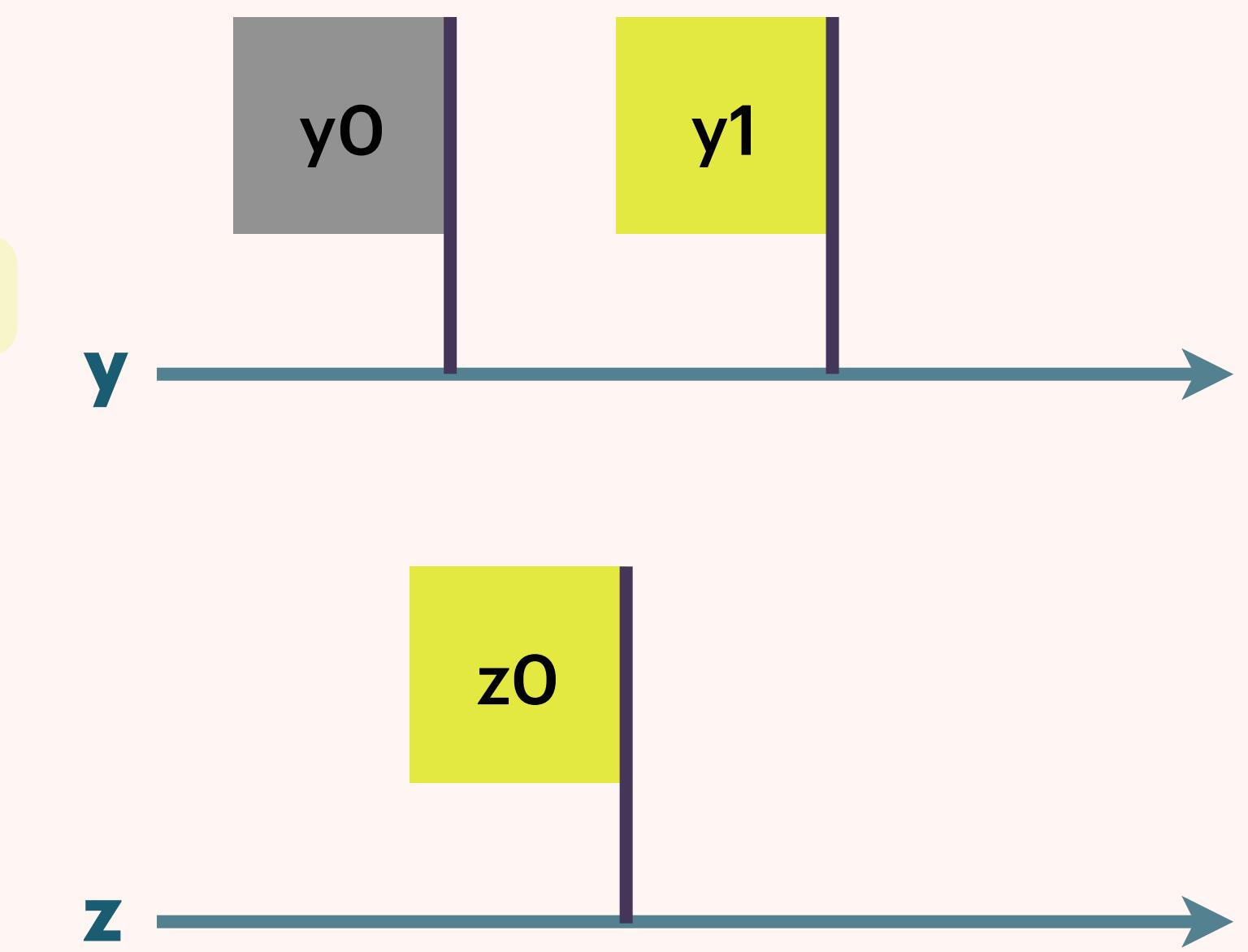
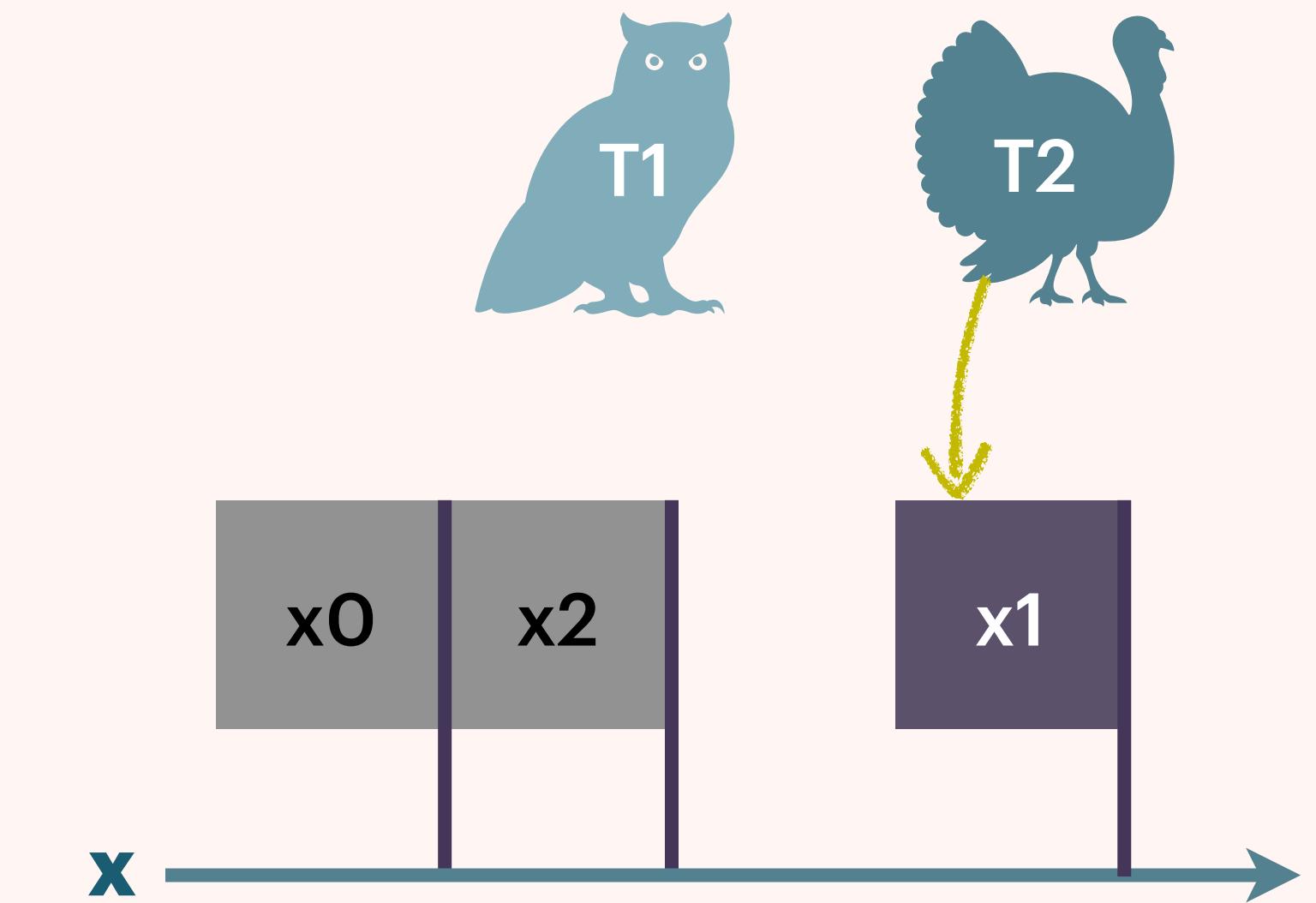


When reading, the message:

- cannot be before thread's view
- may be before others

and the thread:

- inherits the copy of the view



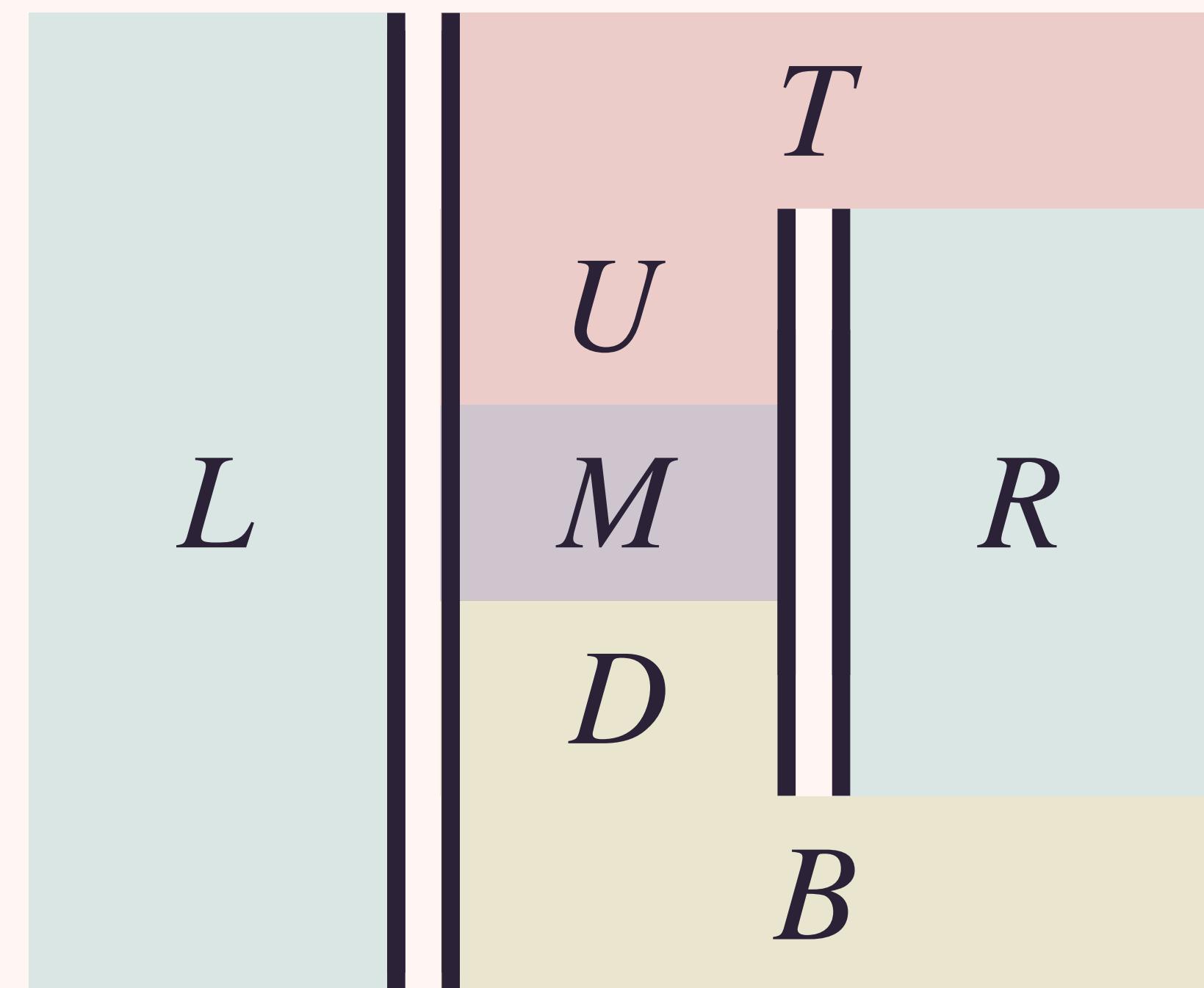
CAUSALITY AND COMPOSITION

With first class parallelism

$$L \parallel (T; ((U; M; D) \parallel R); B)$$

Generalized Sequencing

$$(M_1; M_2) \parallel (K_1; K_2) \Rightarrow (M_1 \parallel K_1); (M_2 \parallel K_2)$$



TRACE-BASED SEMANTICS

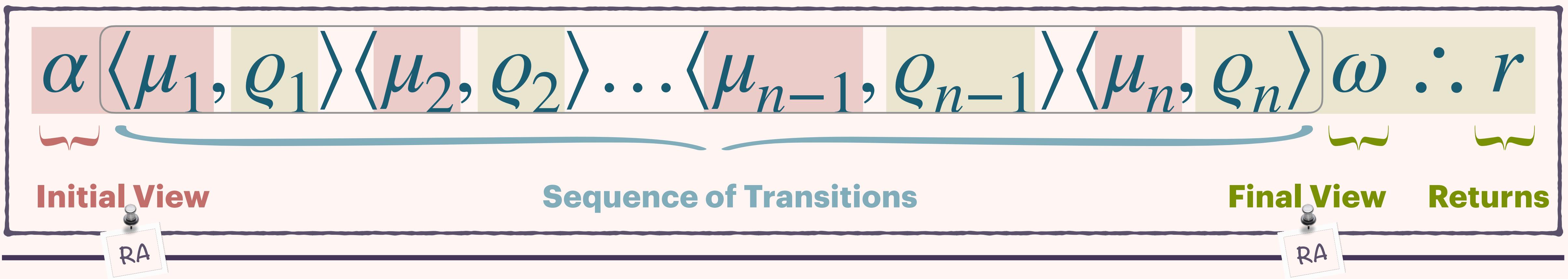


TRACE-BASED SEMANTICS IN RA

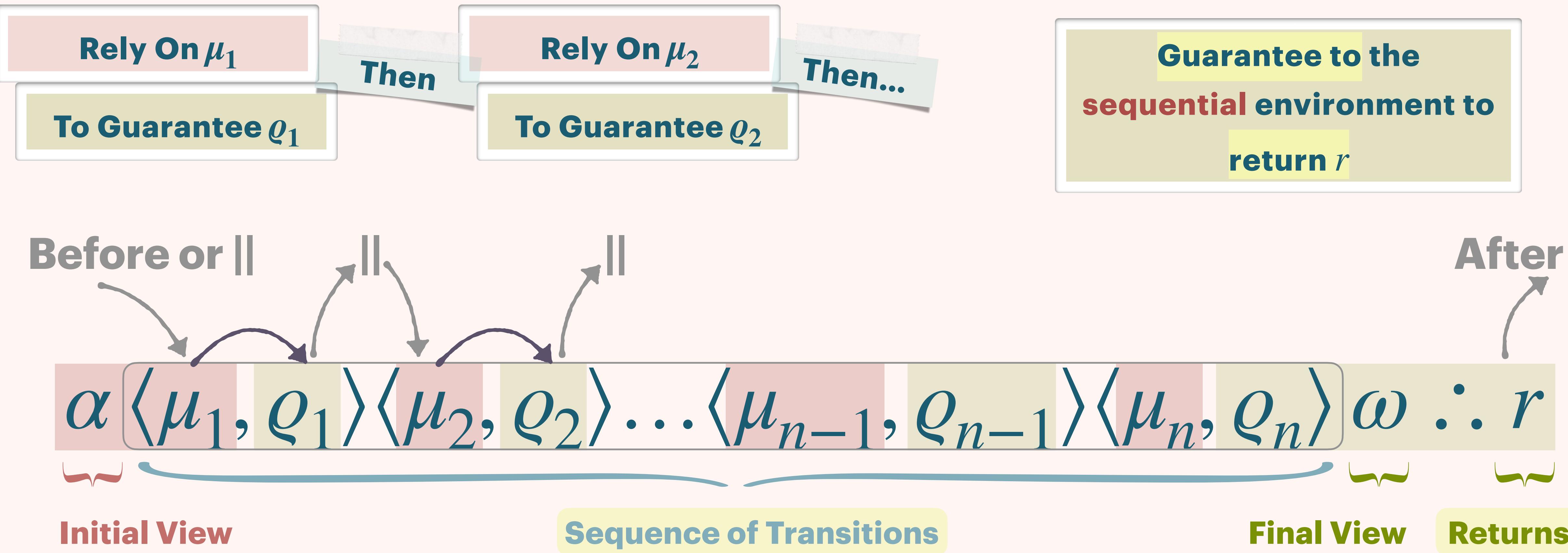
Terms denote sets of traces

$$\llbracket M \rrbracket \ni \tau$$

Each trace represents a possible behavior as a Rely/Guarantee sequence



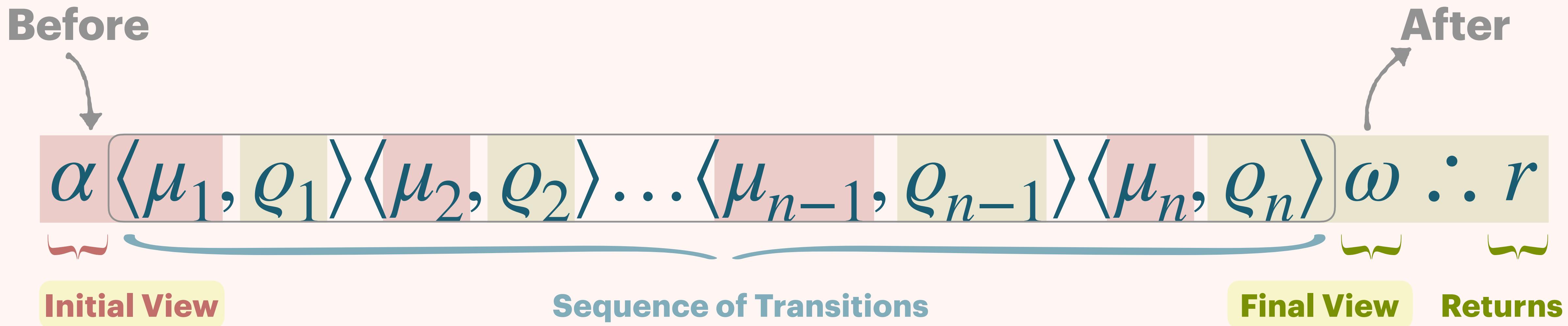
TRACE-BASED SEMANTICS IN RA



TRACE-BASED SEMANTICS IN RA

Rely on the
sequential environment to
reveal messages before α

Guarantee to the
sequential environment to
reveal messages before ω



Analogous
to Brookes's

TRANSITION CLOSURES

Stutter

$$\alpha \boxed{\xi \eta} \omega \because r \in \llbracket M \rrbracket$$

$$\alpha \boxed{\xi \langle \mu, \mu \rangle \eta} \omega \because r \in \llbracket M \rrbracket$$

Propagate Reliance
as a Guarantee

Mumble

$$\alpha \boxed{\xi \langle \mu, \rho \rangle \langle \rho, \theta \rangle \eta} \omega \because r \in \llbracket M \rrbracket$$

$$\alpha \boxed{\xi \langle \mu, \theta \rangle \eta} \omega \because r \in \llbracket M \rrbracket$$

Rely on an
omitted Guarantee

Specific
to RA

VIEW CLOSURES

Rewind

$$\alpha' \leq \alpha$$

$$\alpha[\xi]\omega \because r \in \llbracket M \rrbracket$$

$$\alpha'[\xi]\omega \because r \in \llbracket M \rrbracket$$

Relying on more
being revealed

Forward

$$\alpha[\xi]\omega \because r \in \llbracket M \rrbracket$$

$$\omega \leq \omega'$$

$$\alpha[\xi]\omega' \because r \in \llbracket M \rrbracket$$

Guaranteeing less
being revealed

COMPOSITION

Sequential

$$\frac{\alpha[\xi_1]\kappa \because r_1 \in \llbracket M_1 \rrbracket \quad \kappa[\xi_2]\omega \because r_2 \in \llbracket M_2 \rrbracket[x \mapsto r_1]}{\alpha[\xi_1\xi_2]\omega \because r_2 \in \llbracket \text{let } x = M_1 \text{ in } M_2 \rrbracket}$$

Parallel

$$\frac{\forall i \in \{1,2\} . \alpha[\xi_i]\omega \because r_i \in \llbracket M_i \rrbracket \quad \xi \in \xi_1 \parallel \xi_2}{\alpha[\xi]\omega \because \langle r_1, r_2 \rangle \in \llbracket M_1 \parallel M_2 \rrbracket}$$



ABSTRACTION

WHAT WE CAN JUSTIFY

with Stutter, Mumble, Rewind, and Forward

- Structural equivalences, e.g. if K is effect-free then

Standard Semantics

$$\llbracket \text{if } K \text{ then } M; P_1 \text{ else } M; P_2 \rrbracket = \llbracket M; \text{if } K \text{ then } P_1 \text{ else } P_2 \rrbracket$$

- Laws of Parallel Programming, e.g. Generalized Sequencing

First-class parallelism

$$\llbracket (M_1; M_2) \parallel (K_1; K_2) \rrbracket \supseteq \llbracket (M_1 \parallel K_1); (M_2 \parallel K_2) \rrbracket$$

- Some memory access related transformations, e.g. Read-Read Elimination

$$\llbracket \text{let } a = x? \text{ in let } b = x? \text{ in } \langle a, b \rangle \rrbracket \supseteq \llbracket \text{let } c = x? \text{ in } \langle c, c \rangle \rrbracket$$

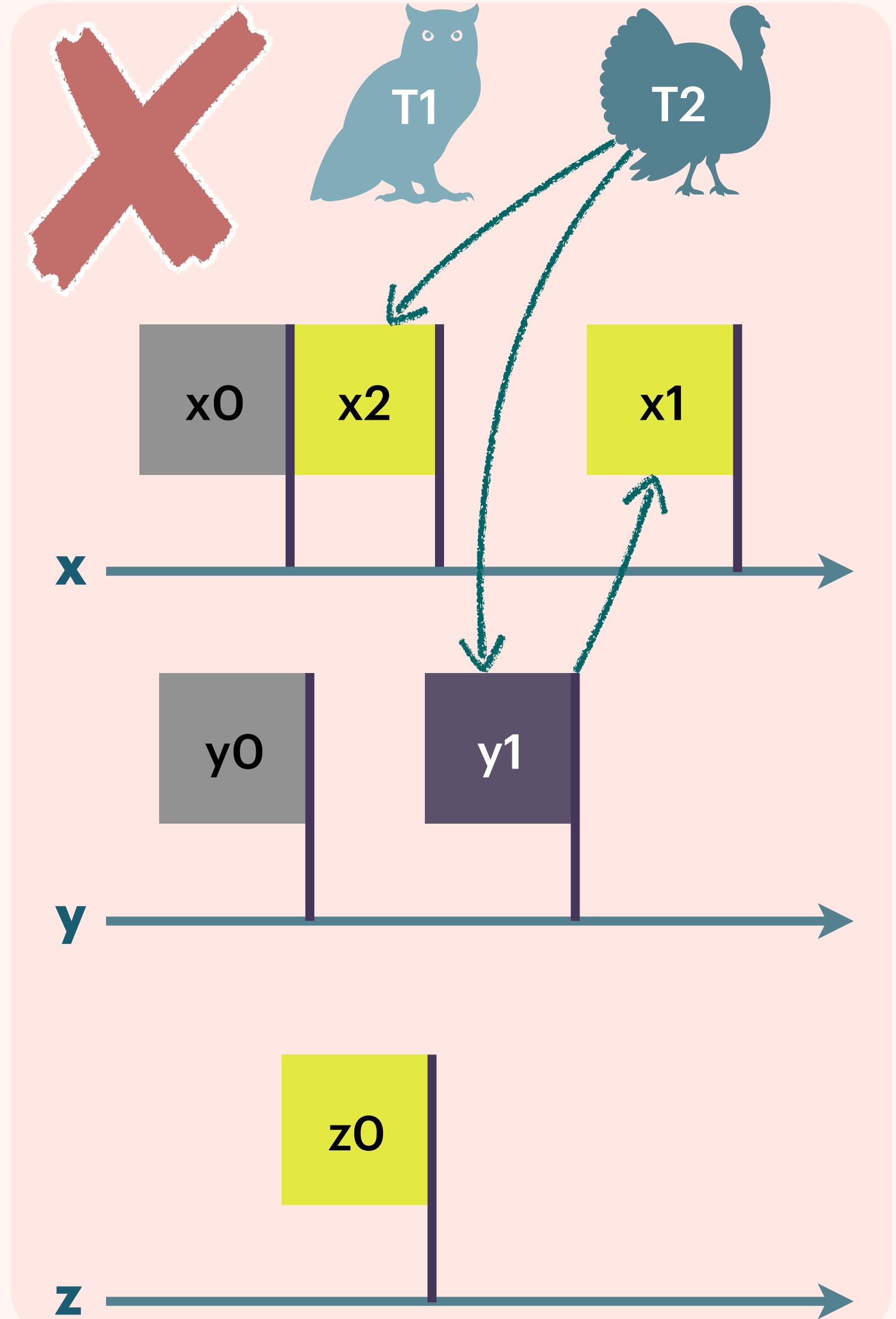
SEMANTIC INVARIANTS ON TRACES

Read Elimination

$x?; M \rightarrow\!\!\! \rightarrow M$

operational invariant becomes denotational requirement
views point to messages that carry a smaller view

$$\kappa \langle \mu, \mu \rangle \kappa .: \langle \rangle \in \llbracket \langle \rangle \rrbracket \implies \exists v. \kappa \langle \mu, \mu \rangle \kappa .: v \in \llbracket x? \rrbracket$$



MORE CLOSURES

- Some transformations are valid even without preserving state
- Traces cannot strictly correspond to operational semantics (e.g. Transition \equiv exec. steps)

$\alpha \langle \mu_1, \varrho_1 \rangle \langle \mu_2, \varrho_2 \rangle \dots \langle \mu_{n-1}, \varrho_{n-1} \rangle \langle \mu_n, \varrho_n \rangle \omega \therefore r$

$\dots \langle \mu_2, - \rangle, M_1 \rightarrow^* \langle \rho_2, - \rangle, M_2 \dots$

Write-Read Reorder

$x := 1;$

$\text{let } a = y? \rightarrow$

$\text{in } M$

$\text{let } a = y?$

$\text{in } x := 1;$

M

\leq

View in message at x

RA Specific
Compiler Optimization

ABSTRACT CLOSURES

- **Absorb a redundant local message into a following one**
(e.g. $\llbracket x := 0; x := 1 \rrbracket \supseteq \llbracket x := 1 \rrbracket$)
- **Dilute a message by a redundant local message**
(e.g. $\llbracket x? \rrbracket \supseteq \llbracket \text{FAA}[x](0) \rrbracket$)
- **Tighten the encumbering view that a local message carries**
(e.g. $\llbracket x := 1; y? \rrbracket \supseteq \llbracket (x := 1 \parallel y?).\text{snd} \rrbracket$)

Specific
to RA

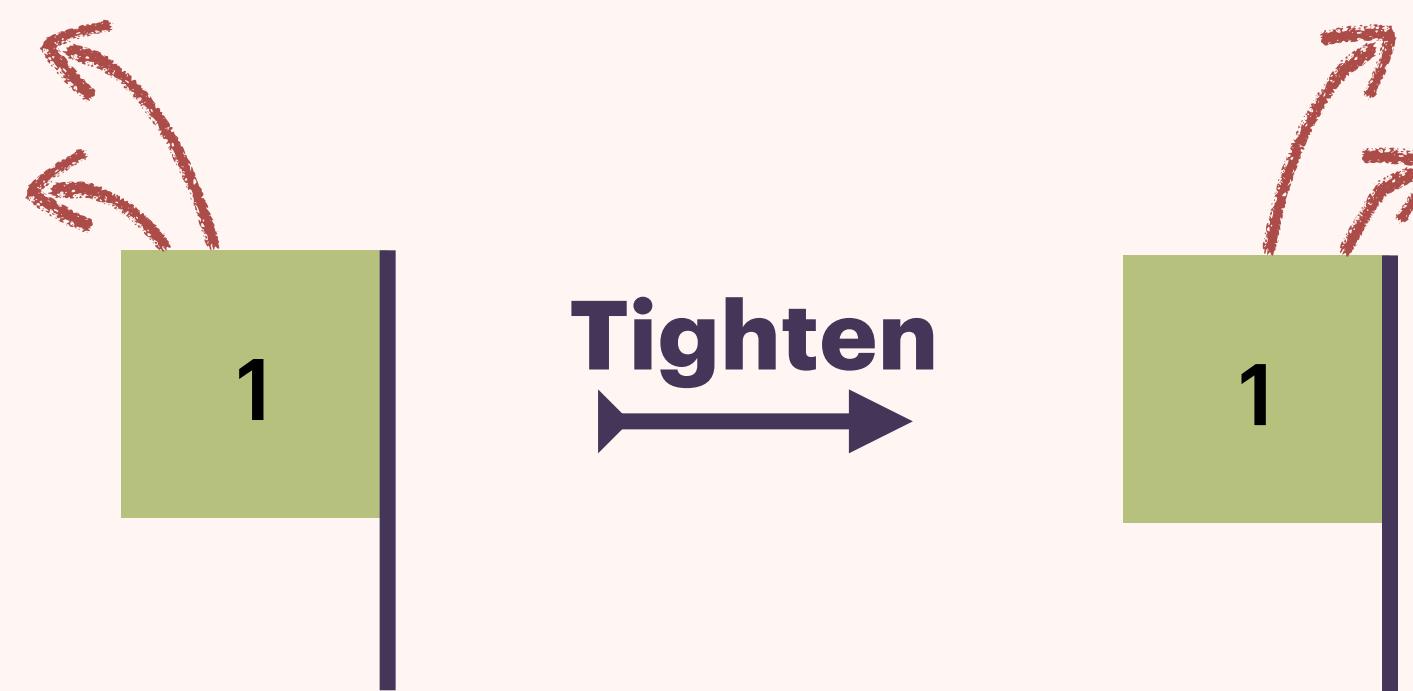
Rewrite

$$\pi \in \llbracket M \rrbracket \quad \pi \mapsto \tau$$

$$\tau \in \llbracket M \rrbracket$$

ABSTRACT REWRITE RULES

Write-Read Deorder + LoPP + Struct \Rightarrow Write-Read Reorder



← **GUARANTEE IS WEAKER
BECAUSE LOADING THIS
MESSAGE OBSCURES MORE**

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

NEW ADEQUACY PROOF IDEA

- Because traces are not operational, the adequacy proof is more nuanced:
- We define a similar denotational semantics $\llbracket \underline{M} \rrbracket$ but without the abstract rules
- We show it is adequate (easier because it has an operational interpretation)
- We show $\llbracket M \rrbracket = \llbracket \underline{M} \rrbracket^\dagger$ — it is enough to apply the closure on top
- We show that the abstract closures preserve observations



Laws of Parallel Programming

Symmetry

$$M \parallel N \rightarrow \mathbf{match} N \parallel M \mathbf{with} \langle y, x \rangle. \langle x, y \rangle$$

Generalized Sequencing

$$(\mathbf{let} x = M_1 \mathbf{in} M_2) \parallel (\mathbf{let} y = N_1 \mathbf{in} N_2) \rightarrow \mathbf{match} M_1 \parallel N_1 \mathbf{with} \langle x, y \rangle. M_2 \parallel N_2$$

Eliminations

Irrelevant Read

$$\ell? ; \langle \rangle \rightarrow \langle \rangle$$

Write-Write

$$\ell := v ; \ell := w \xrightarrow{\text{Ab}} \ell := w$$

Write-Read

$$\ell := v ; \ell? \rightarrow \ell := v ; v$$

Write-FAA

$$\ell := v ; \text{FAA}(\ell, w) \xrightarrow{\text{Ab}} \ell := (v + w) ; v$$

Read-Write

$$\mathbf{let} x = \ell? \mathbf{in} \ell := (x + v) ; x \rightarrow \text{FAA}(\ell, v)$$

Read-Read

$$\langle \ell?, \ell? \rangle \rightarrow \mathbf{let} x = \ell? \mathbf{in} \langle x, x \rangle$$

Read-FAA

$$\langle \ell?, \text{FAA}(\ell, v) \rangle \rightarrow \mathbf{let} x = \text{FAA}(\ell, v) \mathbf{in} \langle x, x \rangle$$

FAA-Read

$$\langle \text{FAA}(\ell, v), \ell? \rangle \rightarrow \mathbf{let} x = \text{FAA}(\ell, v) \mathbf{in} \langle x, x + v \rangle$$

FAA-FAA

$$\langle \text{FAA}(\ell, v), \text{FAA}(\ell, w) \rangle \xrightarrow{\text{Ab}} \mathbf{let} x = \text{FAA}(\ell, v + w) \mathbf{in} \langle x, x + v \rangle$$

Others

Irrelevant Read Introduction

$$\langle \rangle \rightarrow \ell? ; \langle \rangle$$

Read to FAA

$$\ell? \xrightarrow{\text{Di}} \text{FAA}(\ell, 0)$$

Write-Read Deorder

$$\langle (\ell := v), \ell'? \rangle \xrightarrow{\text{Ti}} (\ell := v) \parallel \ell'? \quad (\ell \neq \ell')$$

Write-Read Reorder

$$\langle (\ell := v), \ell'? \rangle \xrightarrow{\text{Ti}} \mathbf{let} x = \ell'? \mathbf{in} (\ell := v) ; x \quad (\ell \neq \ell')$$

CONCLUSION

CONCLUSION

- **Standard, adequate and fully-compositional denotational semantic for RA**
 - **More nuanced traces**
 - **Sufficiently abstract: validates all RA transformations that we know of
(memory access, laws of parallel programming, structural transformations)**
 - **Extended RA view-based machine with compositional (i.e. first-class) parallelism
(weak-memory models are usually studied with top-level parallelism)**
-

LIMITATIONS

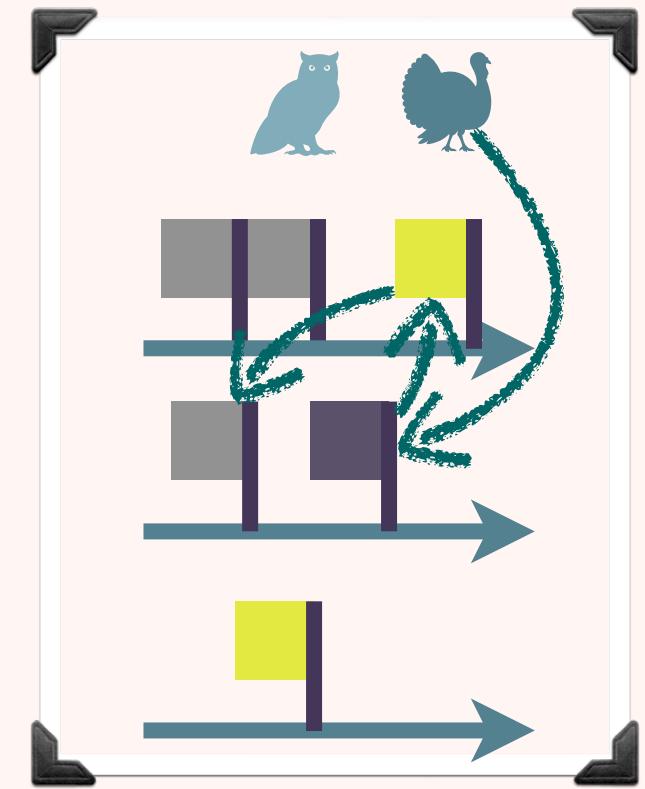
- **Parsimonious in features (e.g. no recursion)**
 - **No type-and-effect system**
 - **No algebraic presentation**
 - **No non-atomics, not the full C/C++ model**
 - **No full abstraction theorem even for first-order**
-

FUTURE DIRECTIONS

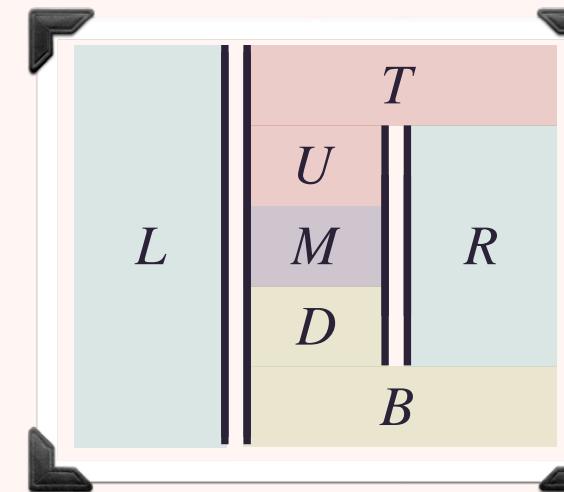
- Address the mentioned limitations, e.g. promising semantics to cover more of C/C++
- Algebraic effects as Rely/Guarantee traces

$$\begin{aligned} (\{-\}) &: \mathbf{Term}_{\{\mathbf{L}, \mathbf{U}\}} X \rightarrow \mathcal{P}_{\text{fin}}(\mathbb{T}X) \\ (\{x\}) &:= \{\langle \rangle :. x\} \\ (\{\mathbf{L}_\ell \langle t_v \rangle_{v \in \mathbf{Val}}\}) &:= \bigcup_{v \in \mathbf{Val}} \{(R_{\ell,v} :: t) :. x \mid t :. x \in (\{t_v\})\} \\ (\{\mathbf{U}_{\ell,v} t\}) &:= \{(G_{\ell,v} :: t) :. x \mid t :. x \in (\{t\})\} \end{aligned}$$

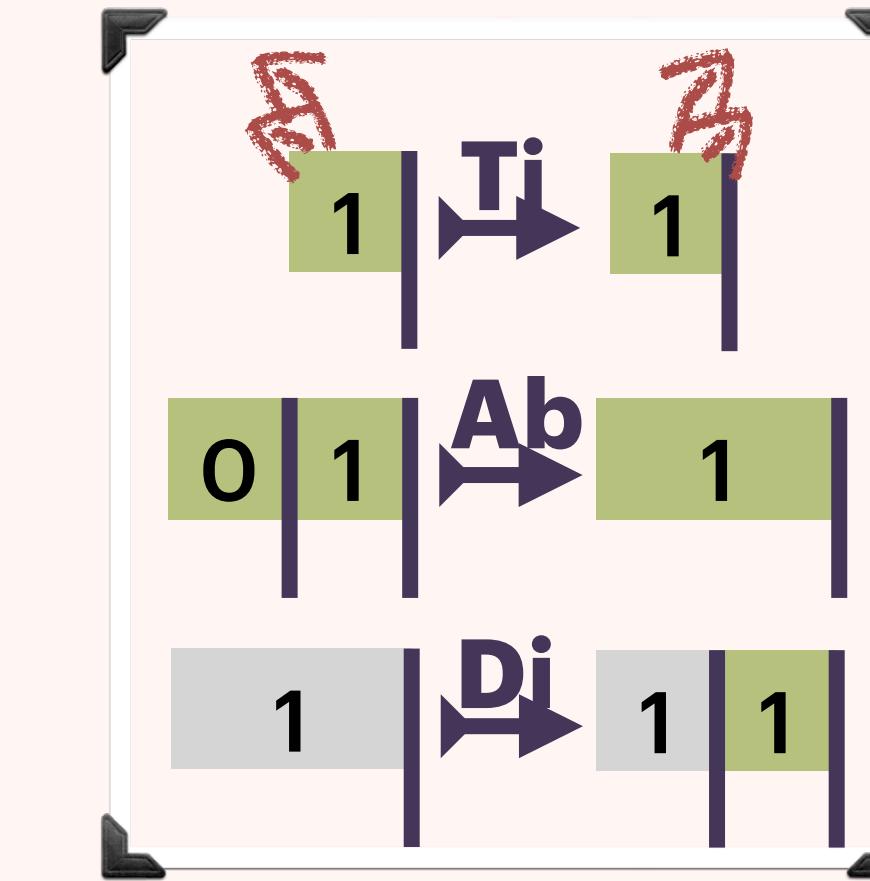
OPERATIONAL SEMANTICS



1ST-CLASS PARALLELISM



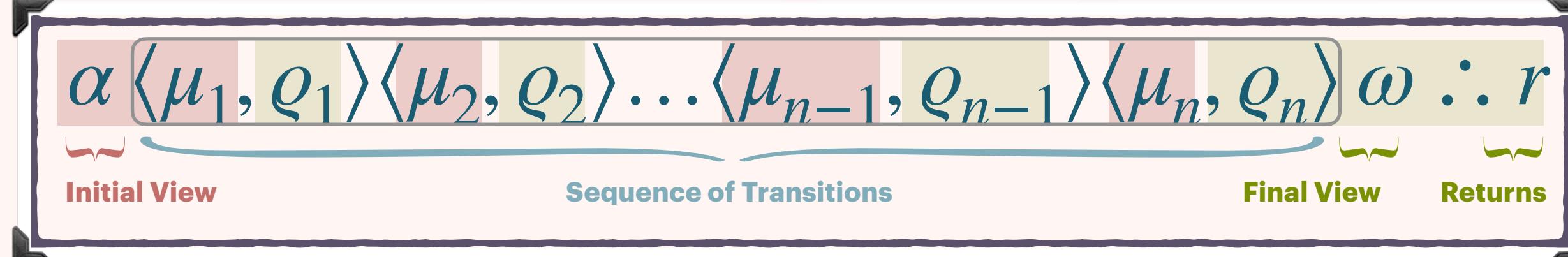
ABSTRACT CLOSURES



ADEQUACY PROOF

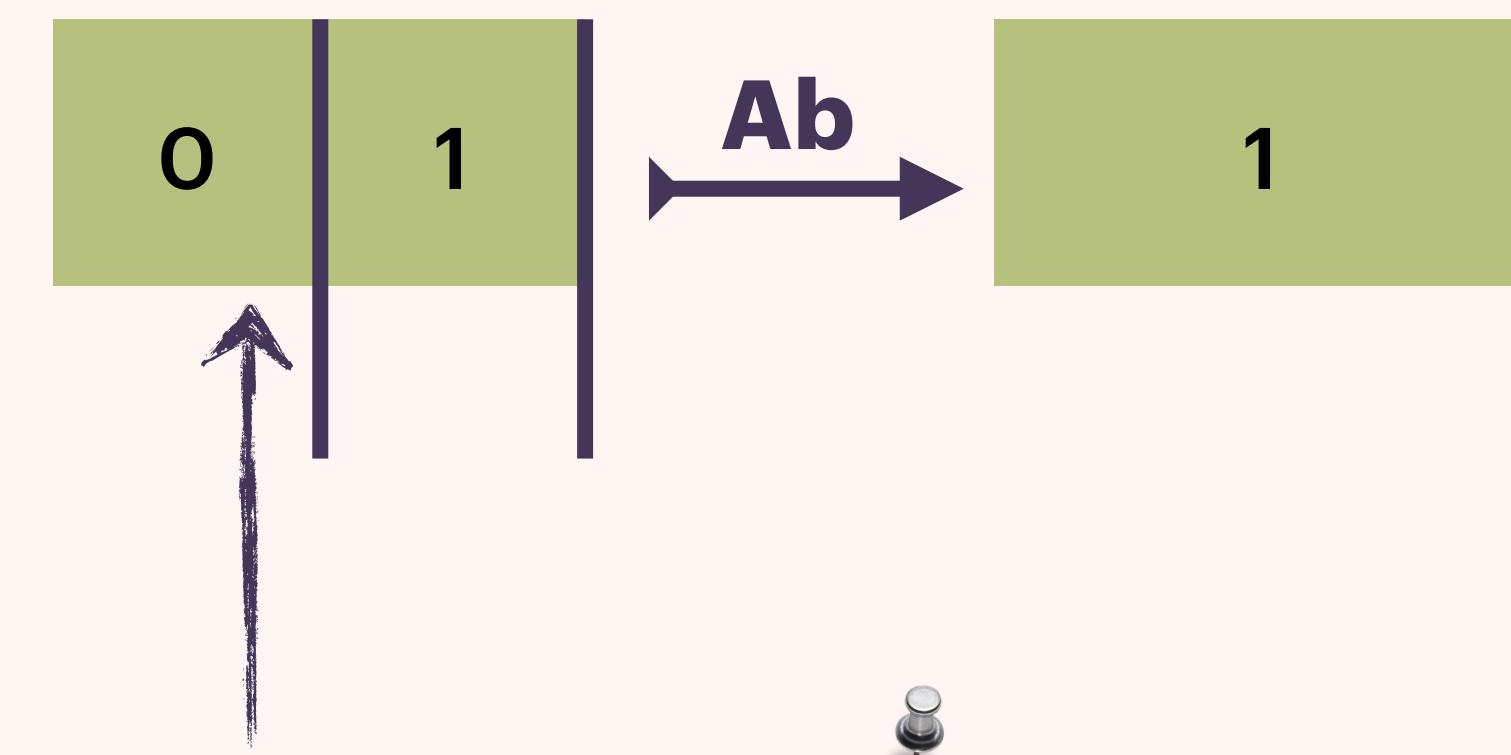
!?

RELY/GUARANTEE TRACES



REWRITE RULE: ABSORB

Write Eliminations

$$x := 0; x := 1 \Rightarrow x := 1$$
$$x := 0; CAS[x](0,1) \Rightarrow x := 1$$


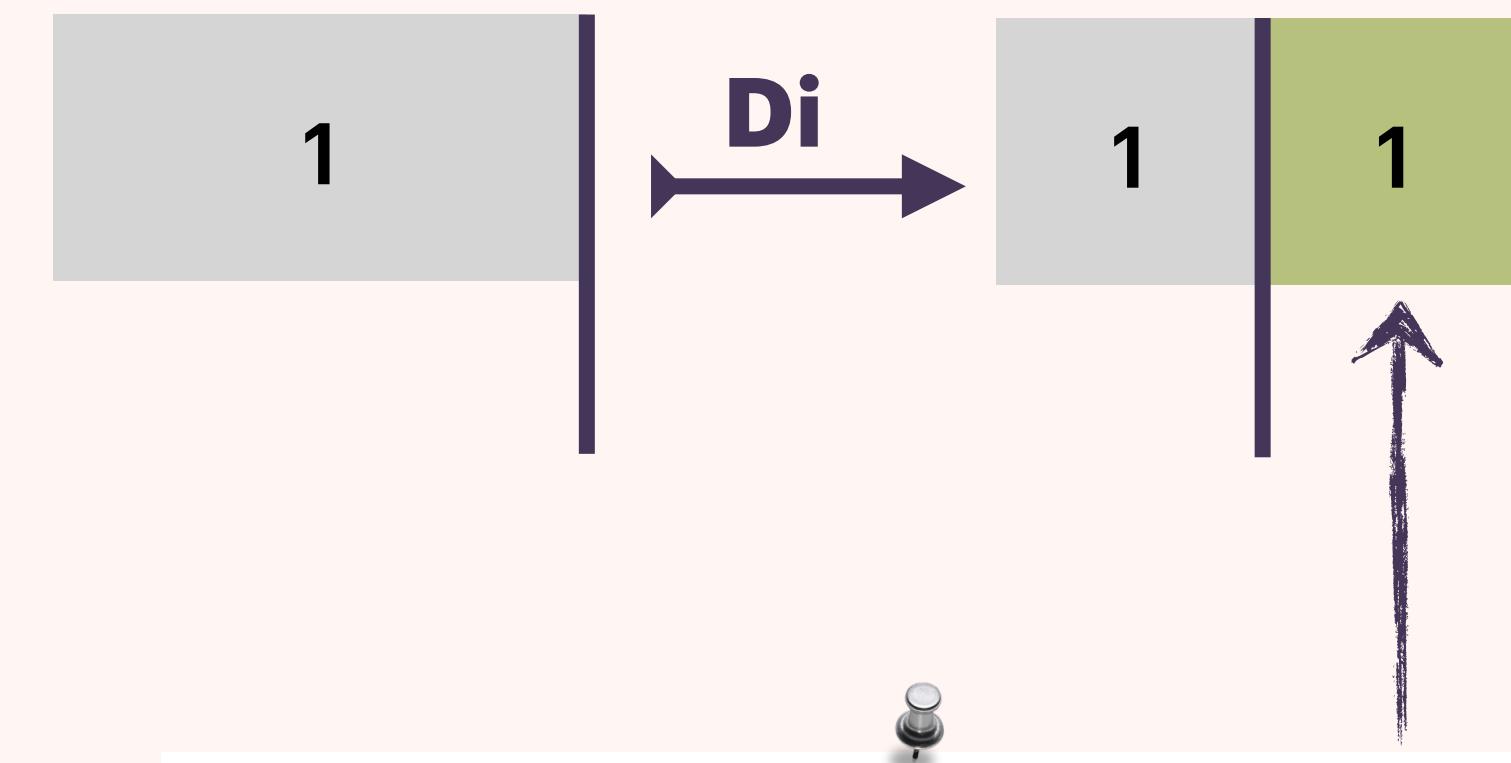
Eliminate redundant message

REWRITE RULE: DILUTE

Write Eliminations

$x? \rightarrow CAS[x](1,1)$

$CAS[x](1,1) \rightarrow FAA[x](0)$



Introduce redundant message