A DENOTATIONAL APPROACH TO RELEASE/ACQUIRE CONCURRENCY

GALOP, POPL 2024

Yotam Dvir, Ohad Kammar, Ori Lahav

IET (LONDON, UK)

GOAL

Release/Acquire

For weak, shared-memory model

Using Brookes-style [1996], totally-ordered traces

Design a standard, monad-based denotational semantics (Moggi [1991])

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

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

TRACE-BASED SEMANTICS

Brookes [1996]

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

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

$\langle \mu_1, \mu_1' \rangle \ \langle \mu_2, \mu_2' \rangle \ \ldots \ \langle \mu_n, \mu_n' \rangle$ $\langle \varrho_1, \varrho_1' \rangle \ \langle \varrho_2, \varrho_2' \rangle \ \ldots \ \langle \varrho_n, \varrho_n' \rangle$

TRACE-BASED SEMANTICS

Brookes [1996]

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

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

$$\langle \mu_1, \mu_1' \rangle \ \langle \mu_2, \mu_2' \rangle \ \ldots \ \langle \mu_n, \mu_n' \rangle \ \langle \varrho_1, \varrho_1' \rangle \ \langle \varrho_2, \varrho_2' \rangle \ \ldots \ \langle \varrho_n, \varrho_n' \rangle$$

SEQUENCE

TRACE-BASED SEMANTICS

Brookes [1996]

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

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

$\langle \varrho_1, \varrho_1' \rangle \ \langle \mu_1, \mu_1' \rangle \ \langle \mu_2, \mu_2' \rangle \ \langle \varrho_2, \varrho_2' \rangle \ \ldots \ \langle \mu_n, \mu_n' \rangle \ \langle \varrho_n, \varrho_n' \rangle$

INTERLEAVE

TRACE-BASED SEMANTICS

Brookes [1996]

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

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

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

Jagadeesan, Petri, Riely [2012]

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

This work

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

CONTRIBUTION

Refinement **Transformation**

***Directionally* Adequate** $[\![M]\!] \supseteq [\![K]\!] \implies M \twoheadrightarrow K$

denotational semantics for RA based on linearly-ordered traces

Standard (CbV) semantics [Moggi 1991]

- **enables structural transformations** (e.g. $[\![K; (M; N)]\!] = [\![(K; M); N]\!]$)
- **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

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

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

Message Passing

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

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

TYPICAL EXAMPLES

Store Buffering

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

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

Message Passing

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

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

TYPICAL EXAMPLES

Store Buffering

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

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

Message Passing

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

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

TYPICAL EXAMPLES

Propagation is not instant

Store Buffering

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

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

✔

Message Passing

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

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

TYPICAL EXAMPLES

Propagation is not instant

Store Buffering

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

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

✔

Message Passing

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

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

TYPICAL EXAMPLES

Propagation is not instant

Store Buffering

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

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

✓

Message Passing

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

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

TYPICAL EXAMPLES

Store Buffering

Propagation is not instant

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

$$x := 1; \quad \| \quad y := 1;$$

$$y? \quad /\!/0 \qquad x? \quad /\!/0$$

✔

Message Pas

Propagation respects causality

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

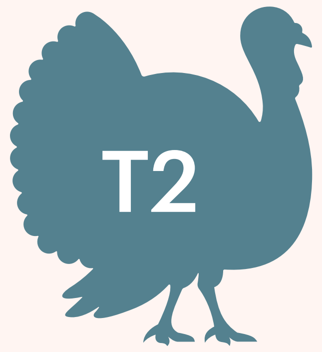
$$x := 1; \quad \| \quad y?; \; /\!/1$$

$$y := 1 \qquad x? \quad /\!/0$$

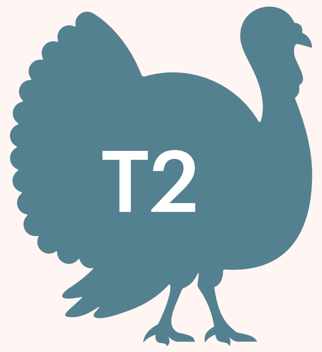
✘

RELEASE/ACQUIRE VIEW-BASED OPERATIONAL SEMANTICS

Kang et al. [2017]



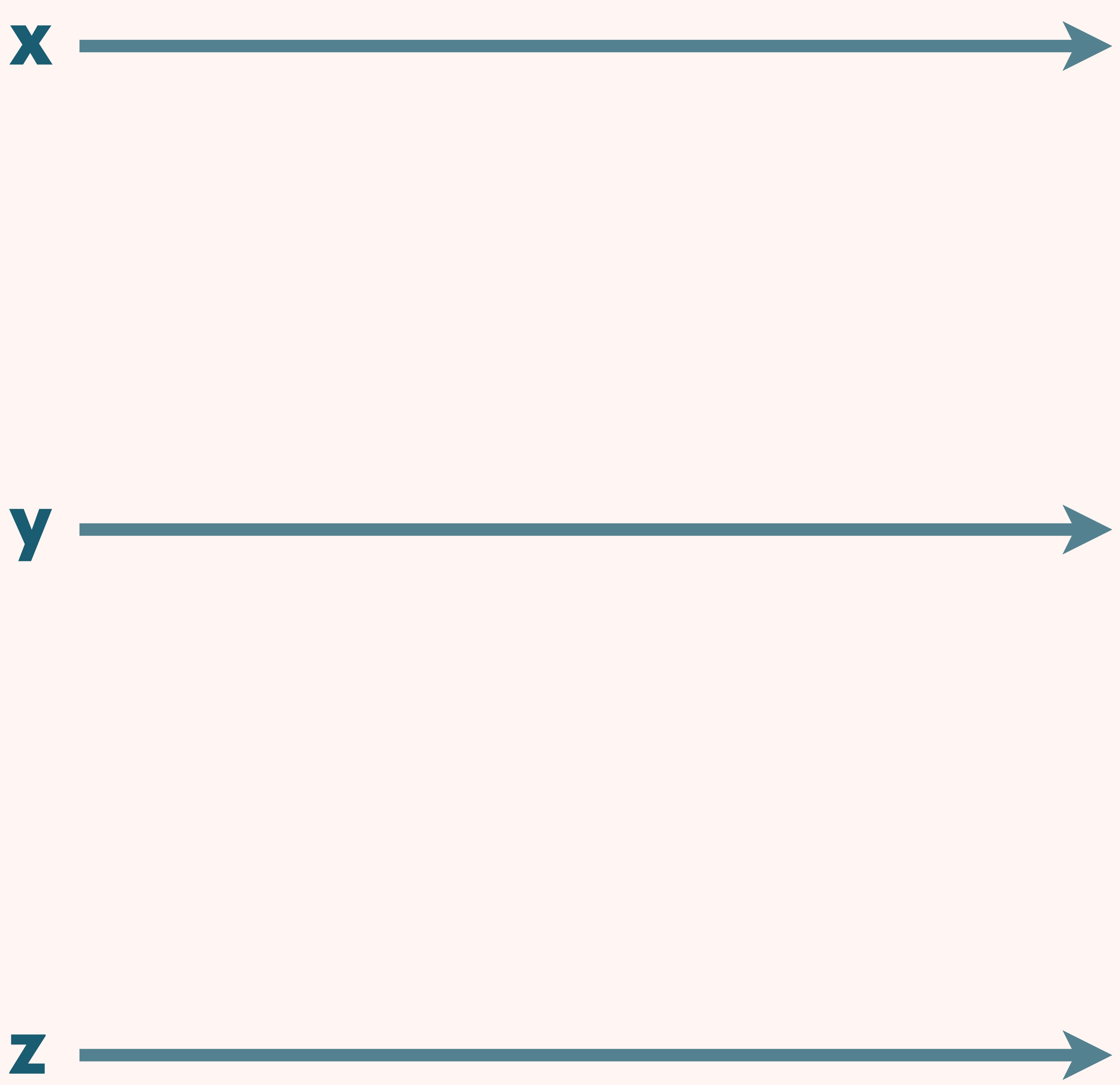
- Memory: Timeline per location (e.g. x, y, z)
- Populated with immutable messages (e.g. x0, y0, z0)
- Each thread's view points to a msgs on each timeline (e.g. T1)
- Thread's cannot read from "the past"
- Each msg's view points to a msg on each other timelines (e.g. y1)
- Message views are used for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



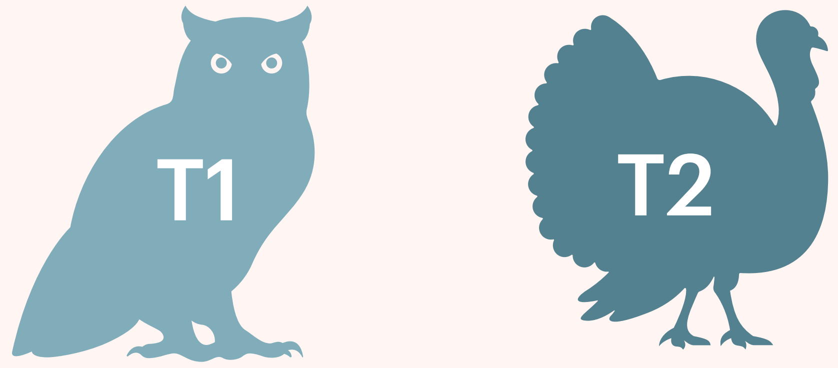
RELEASE/ACQUIRE VIEW-BASED OPERATIONAL SEMANTICS

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

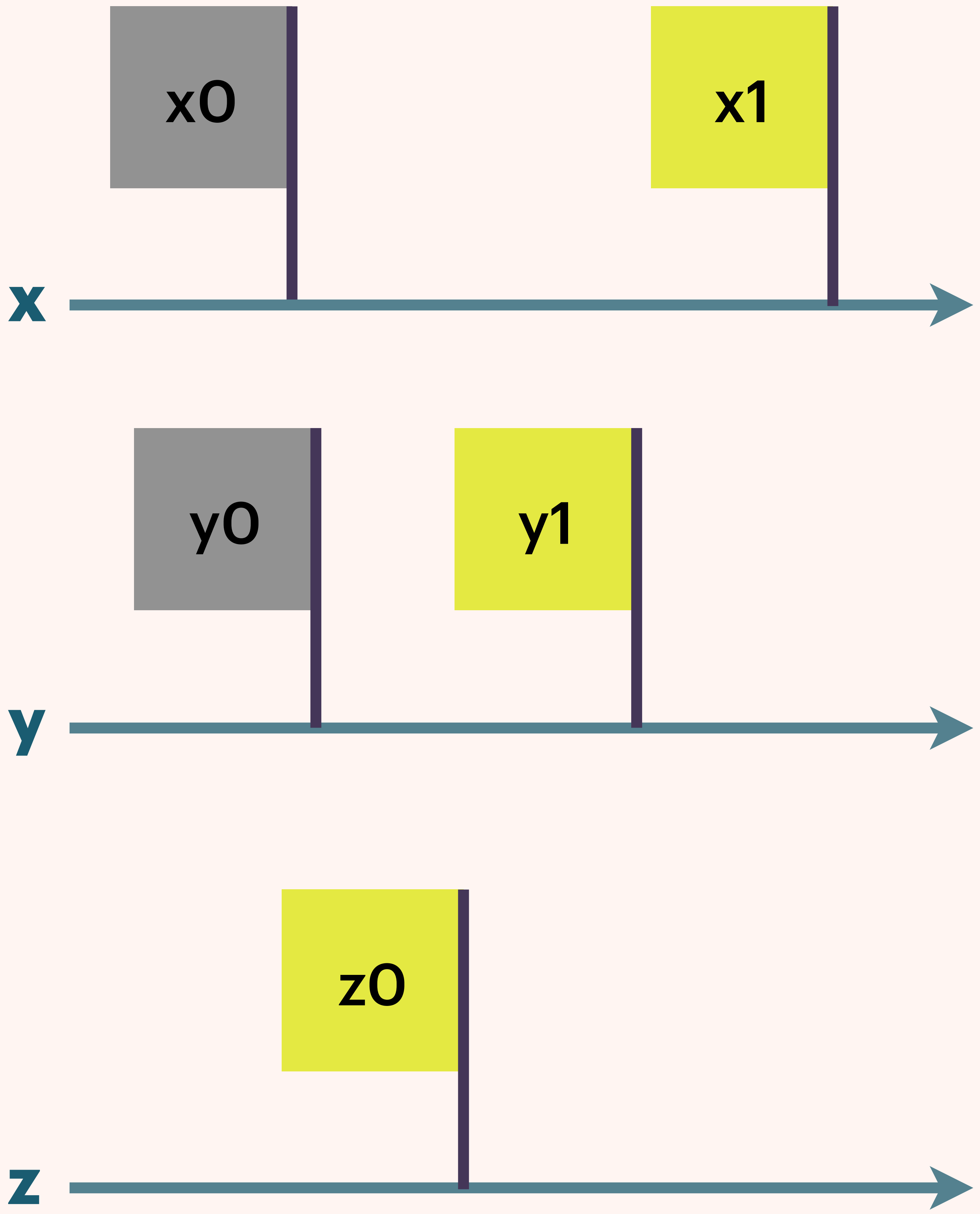


RELEASE/ACQUIRE VIEW-BASED OPERATIONAL SEMANTICS



Kang et al. [2017]

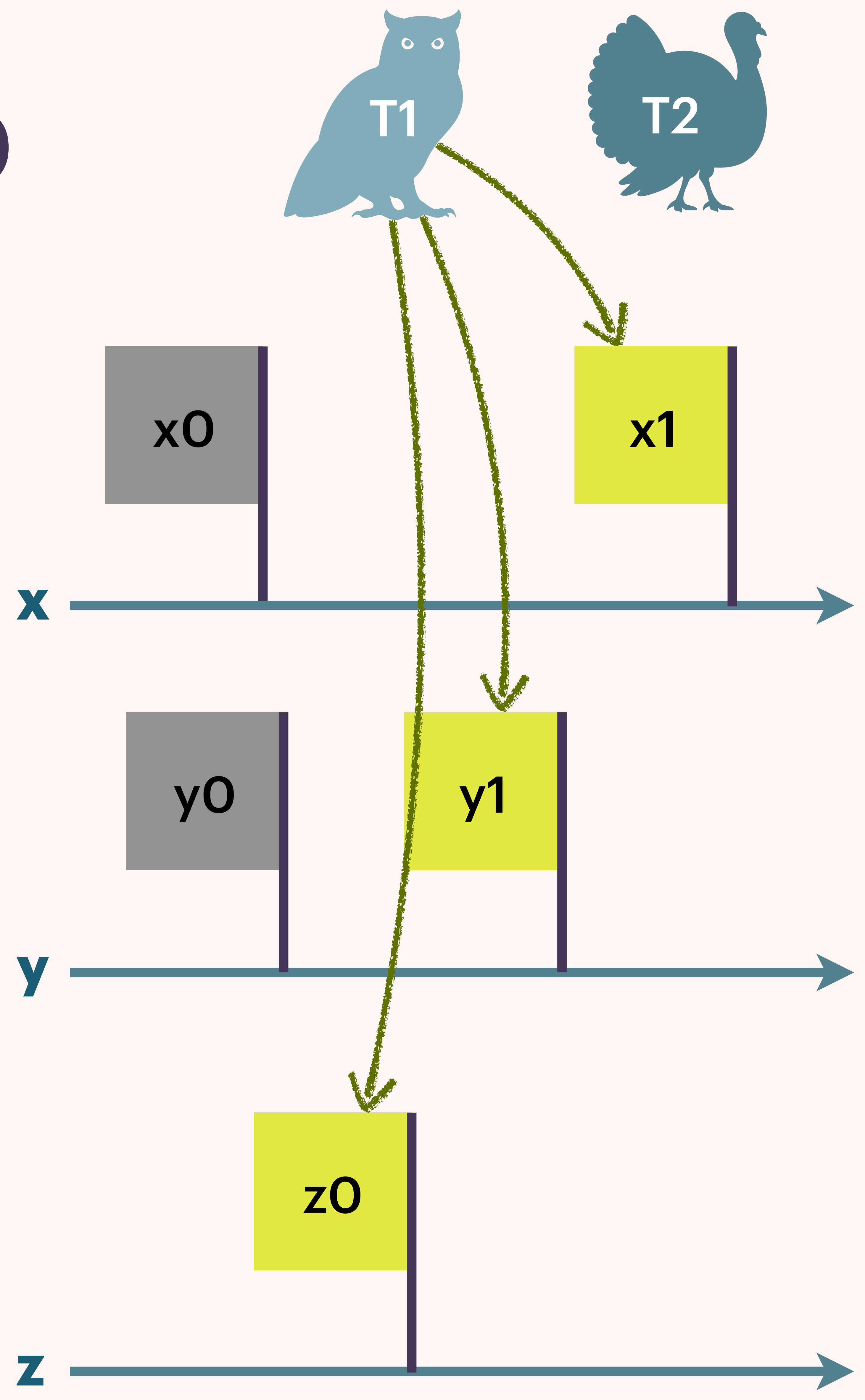
- **Memory: Timeline per location (e.g. x, y, z)**
- **Populated with immutable messages (e.g. x0, y0, z0)**
- **Each thread's view points to a msgs on each timeline (e.g. T1)**
- **Thread's cannot read from "the past"**
- **Each msg's view points to a msg on each other timelines (e.g. y1)**
- **Message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



RELEASE/ACQUIRE VIEW-BASED OPERATIONAL SEMANTICS

Kang et al. [2017]

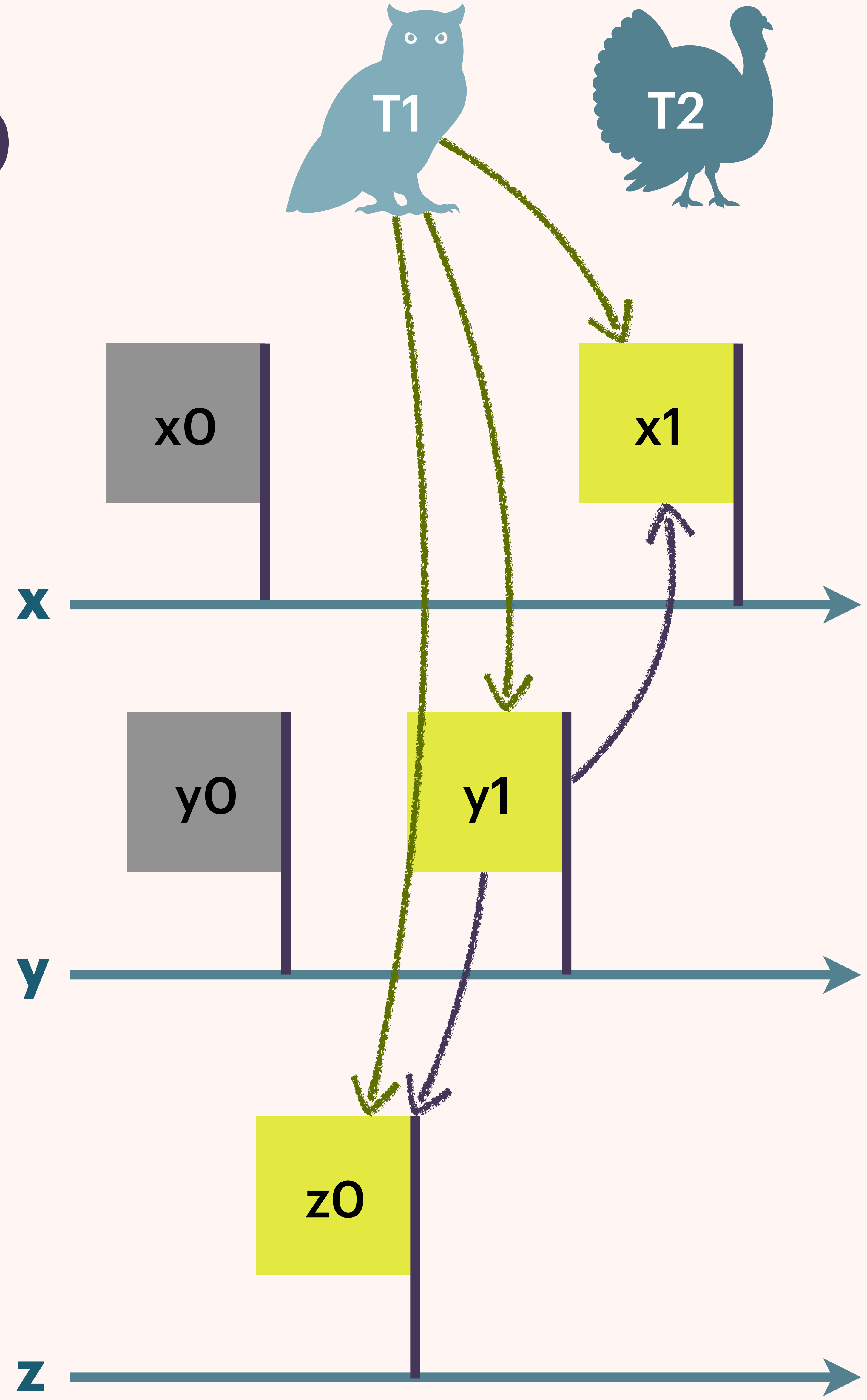
- **Memory:** **Timeline per location** **(e.g. x, y, z)**
- **Populated with immutable messages (e.g. x0, y0, z0)**
- **Each thread's view points to a msgs on each timeline (e.g. T1)**
- **Thread's cannot read from "the past"**
- **Each msg's view points to a msg on each other timelines (e.g. y1)**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

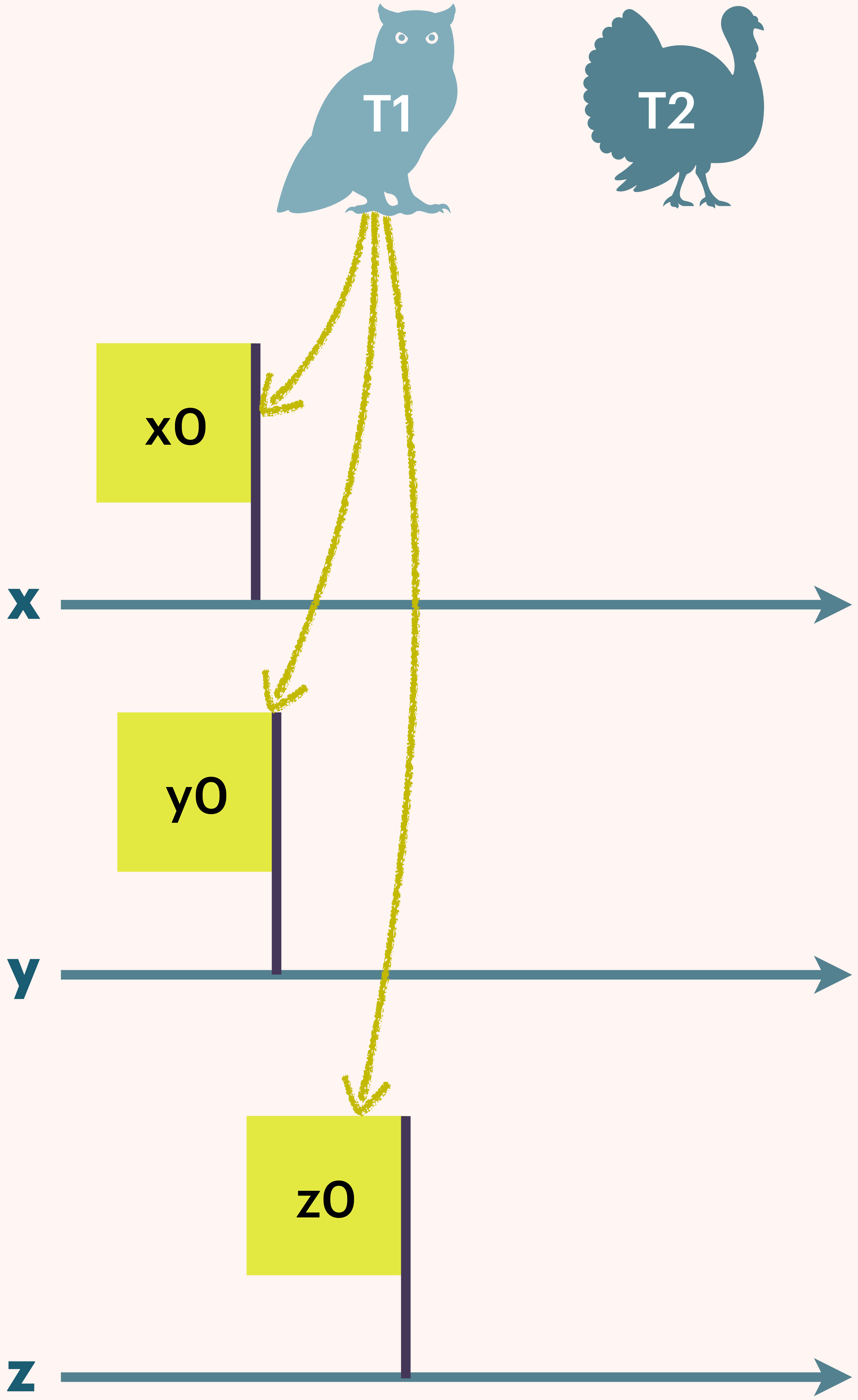


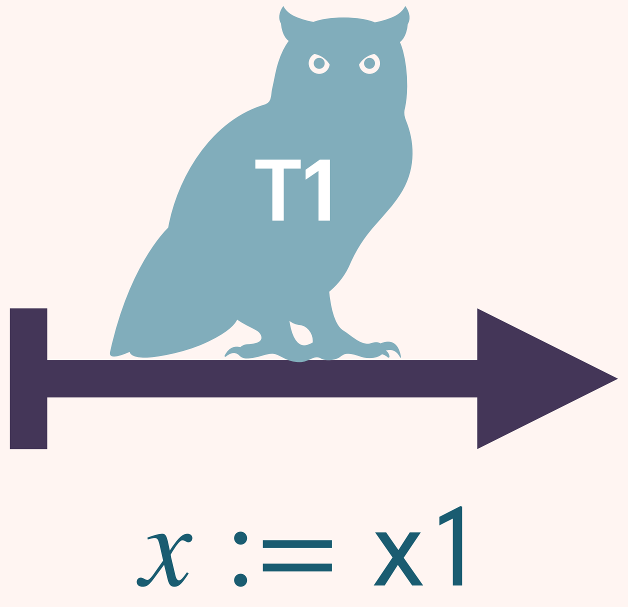
RELEASE/ACQUIRE VIEW-BASED OPERATIONAL SEMANTICS

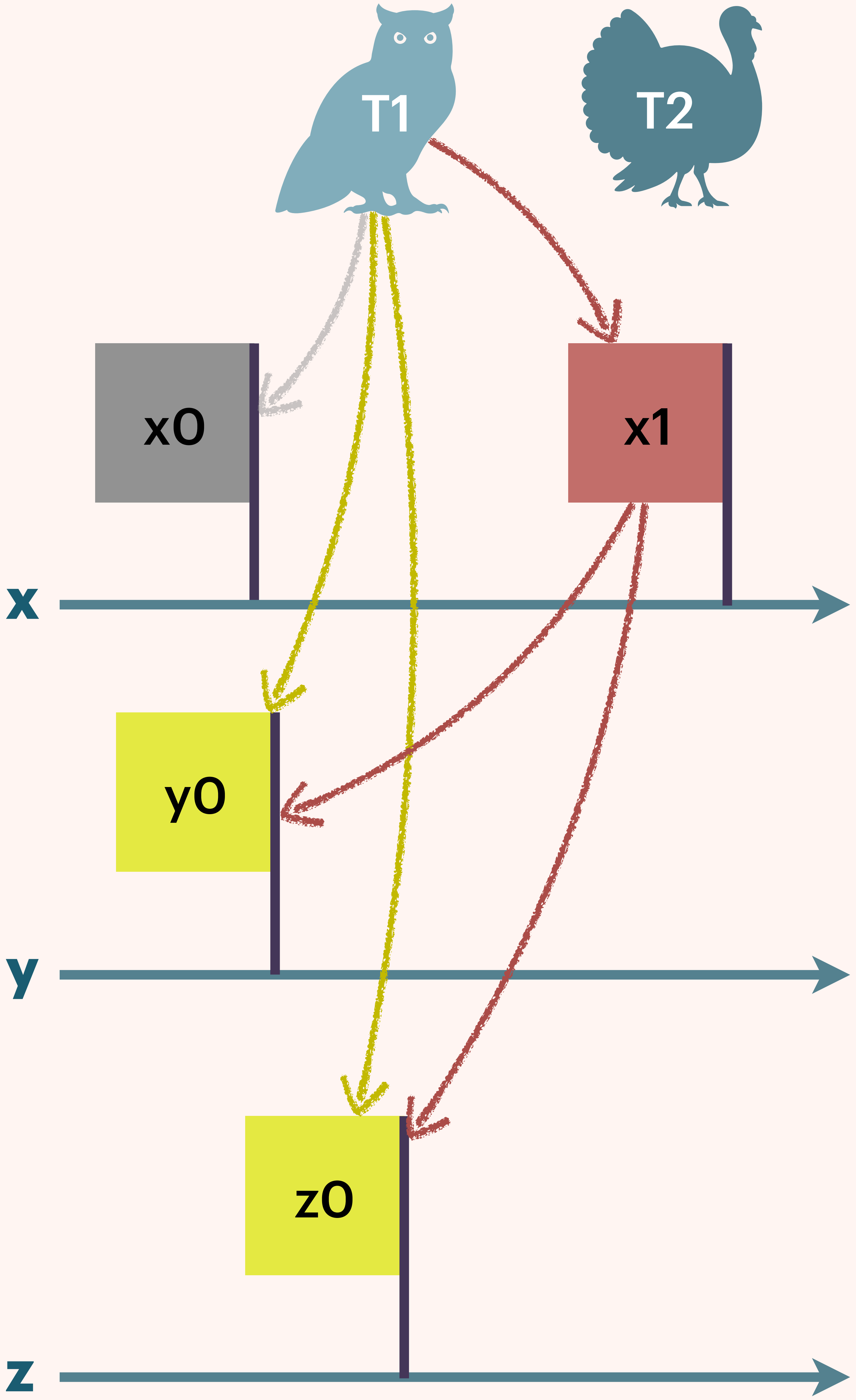
Kang et al. [2017]

- **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



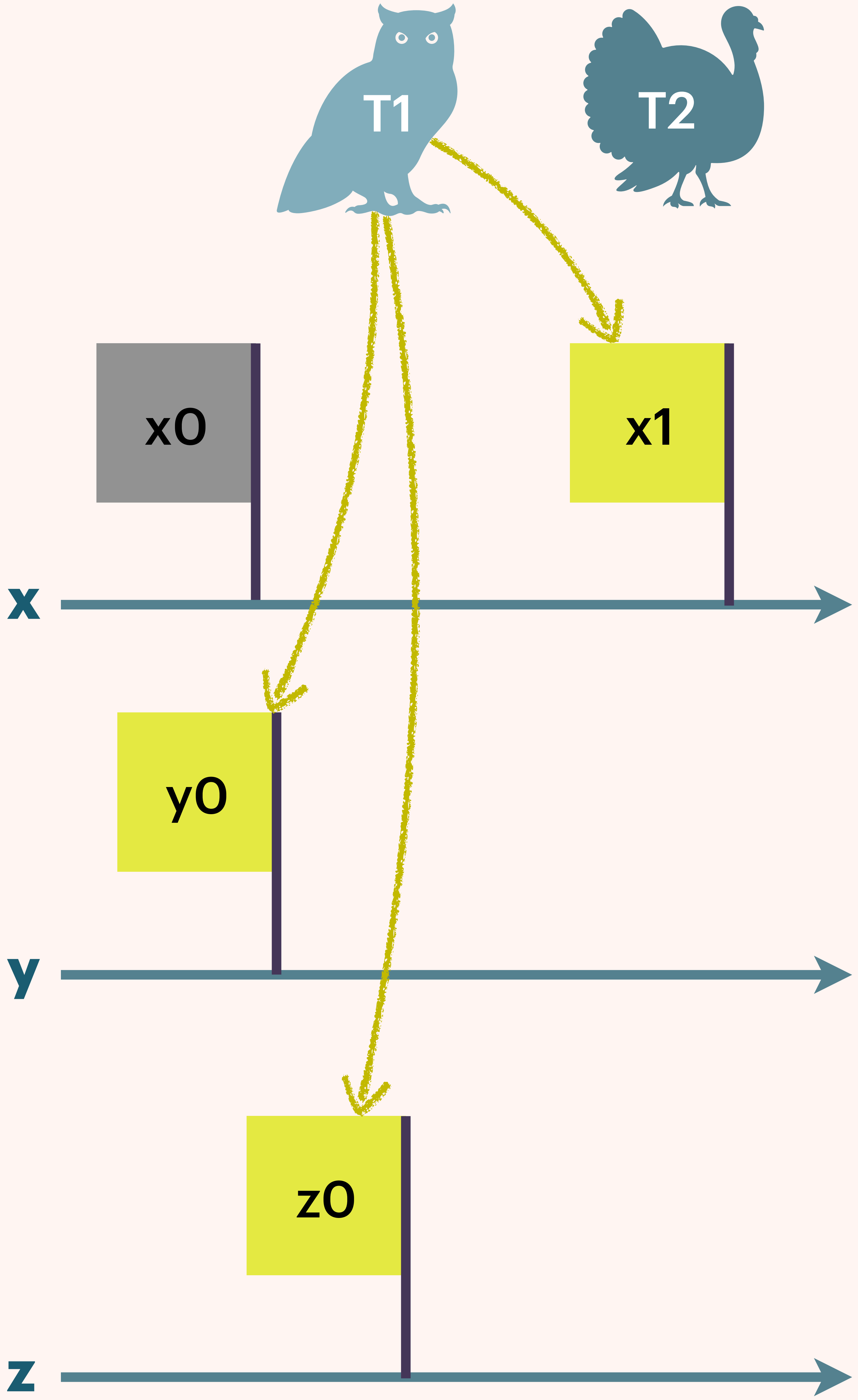


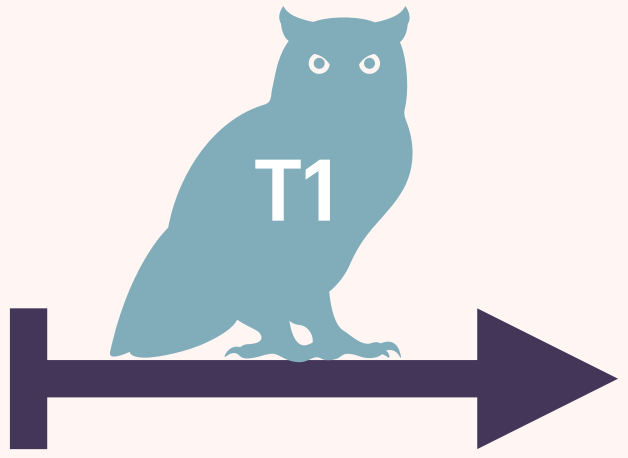




When writing, the message:

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

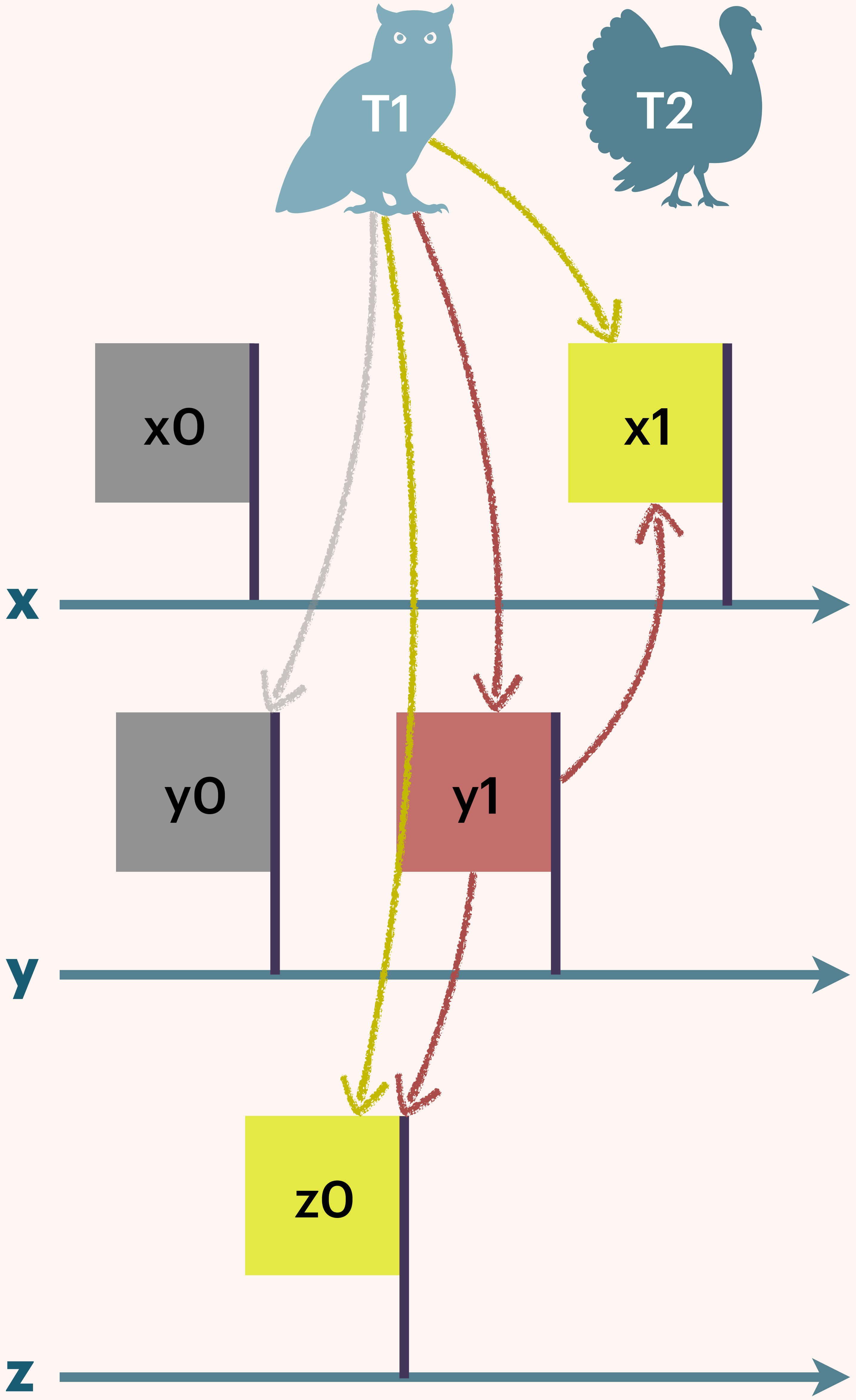


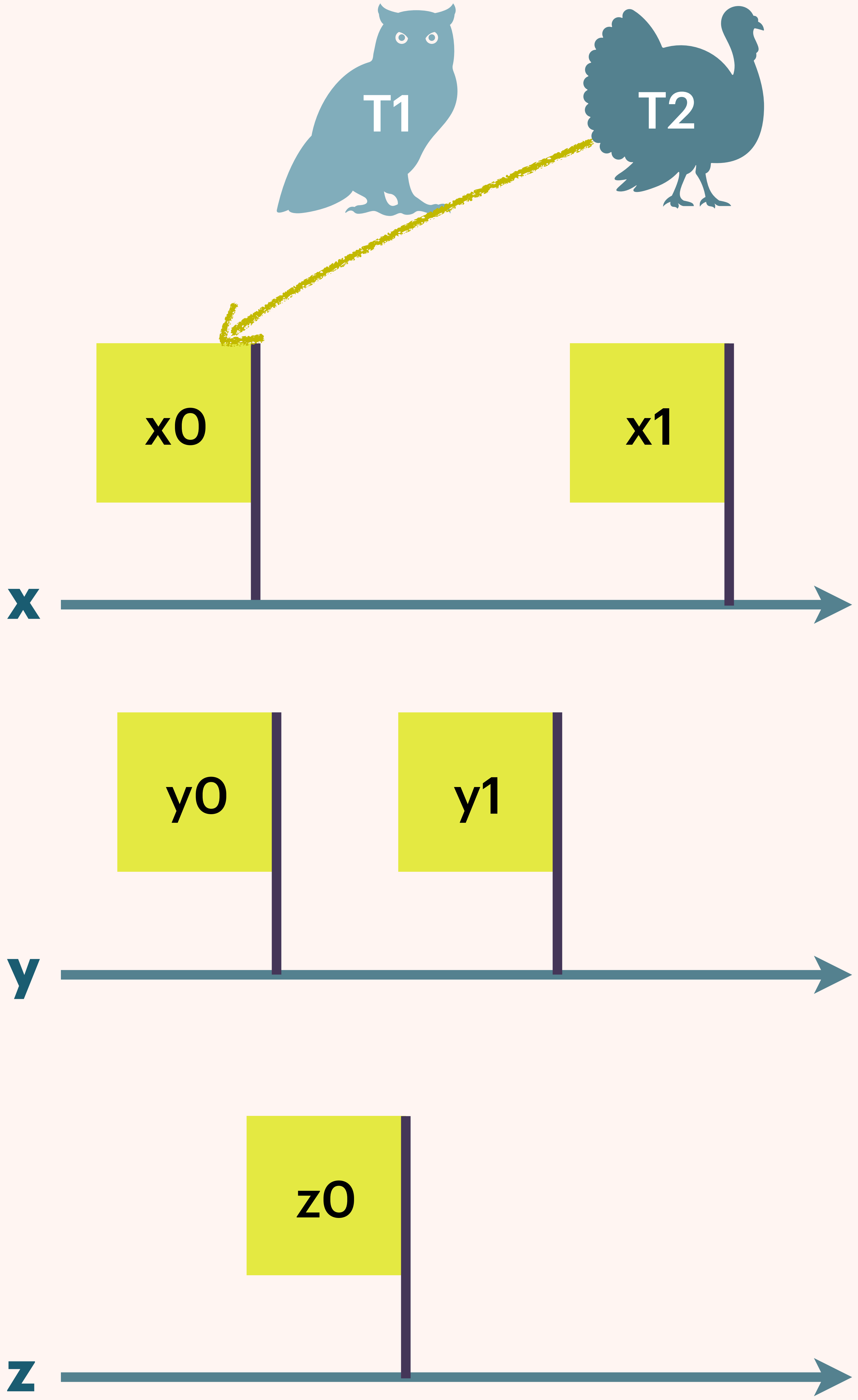


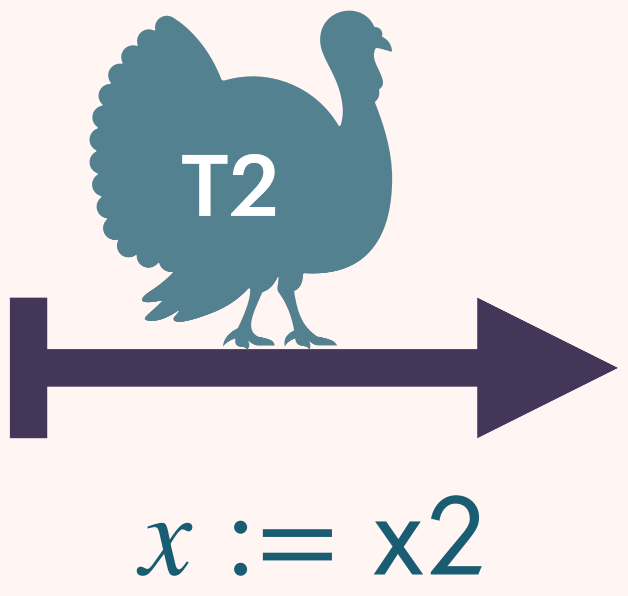
$y := \mathrm{y1}$

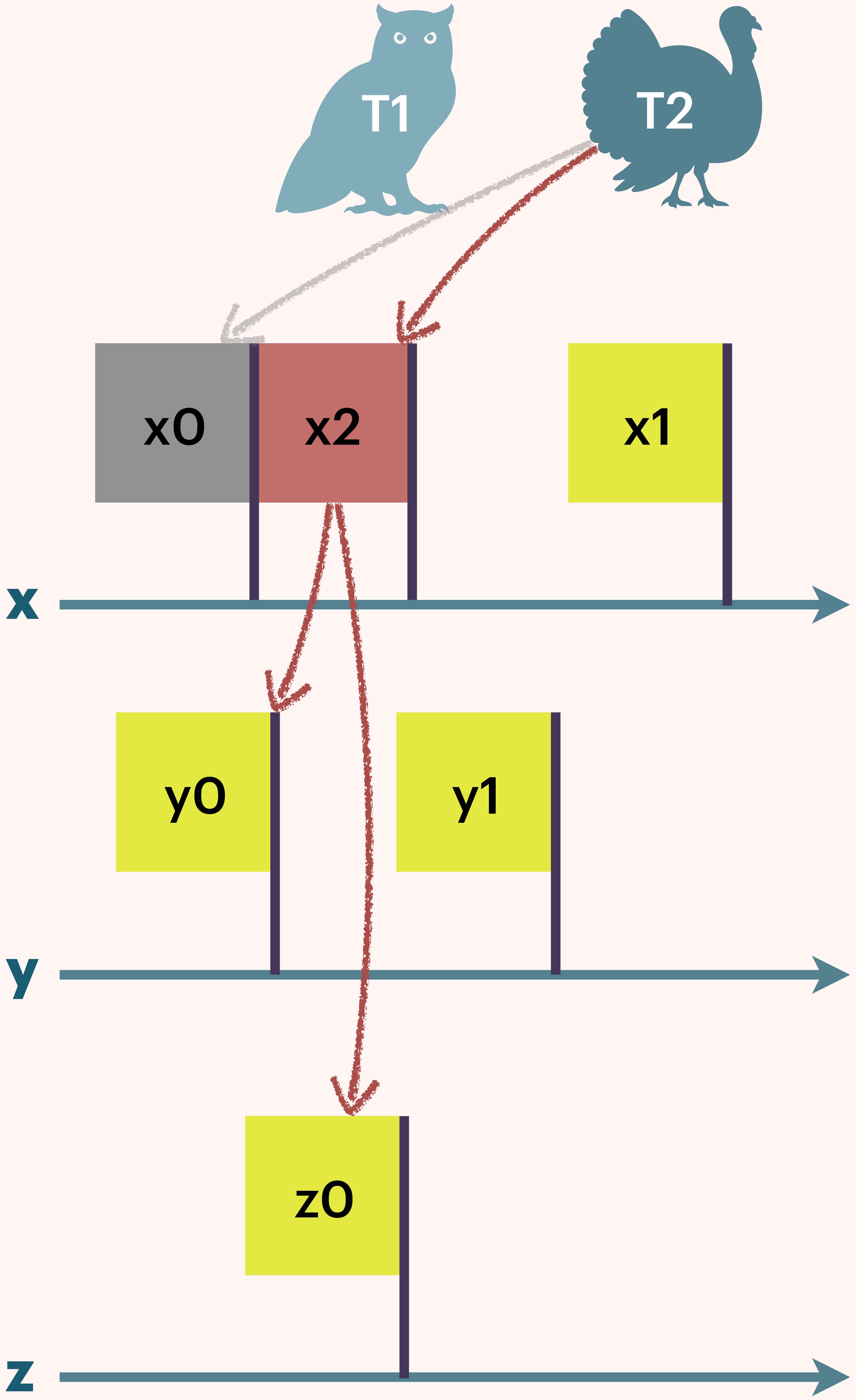
When writing, the message:

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



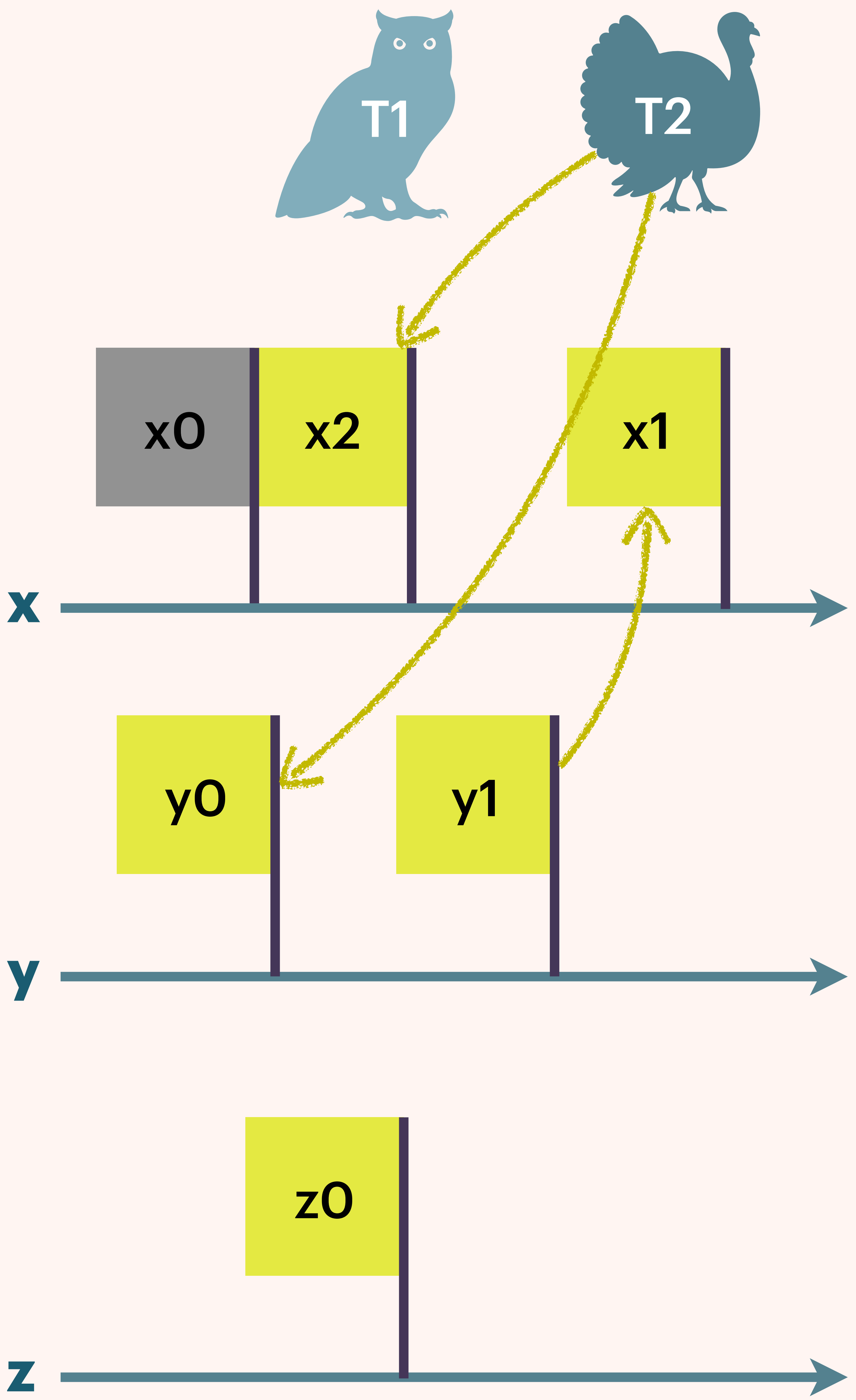


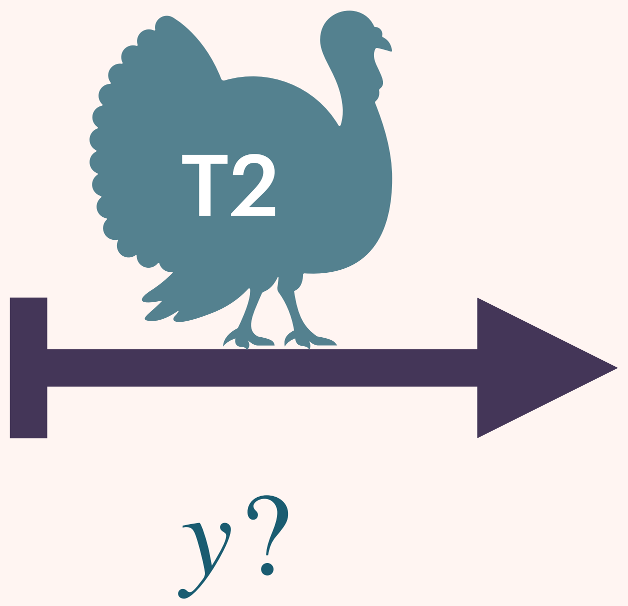




When writing, the message:

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



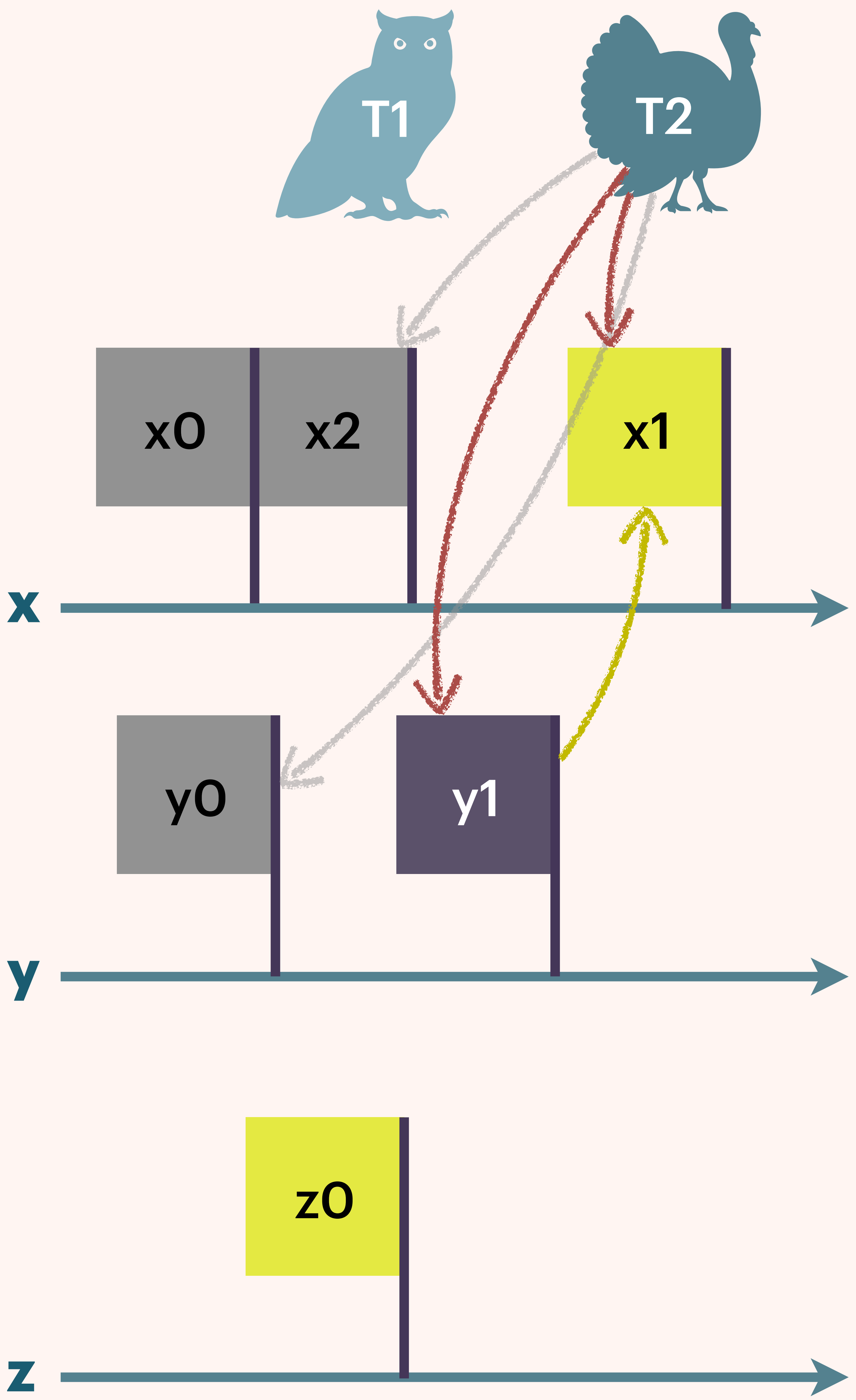


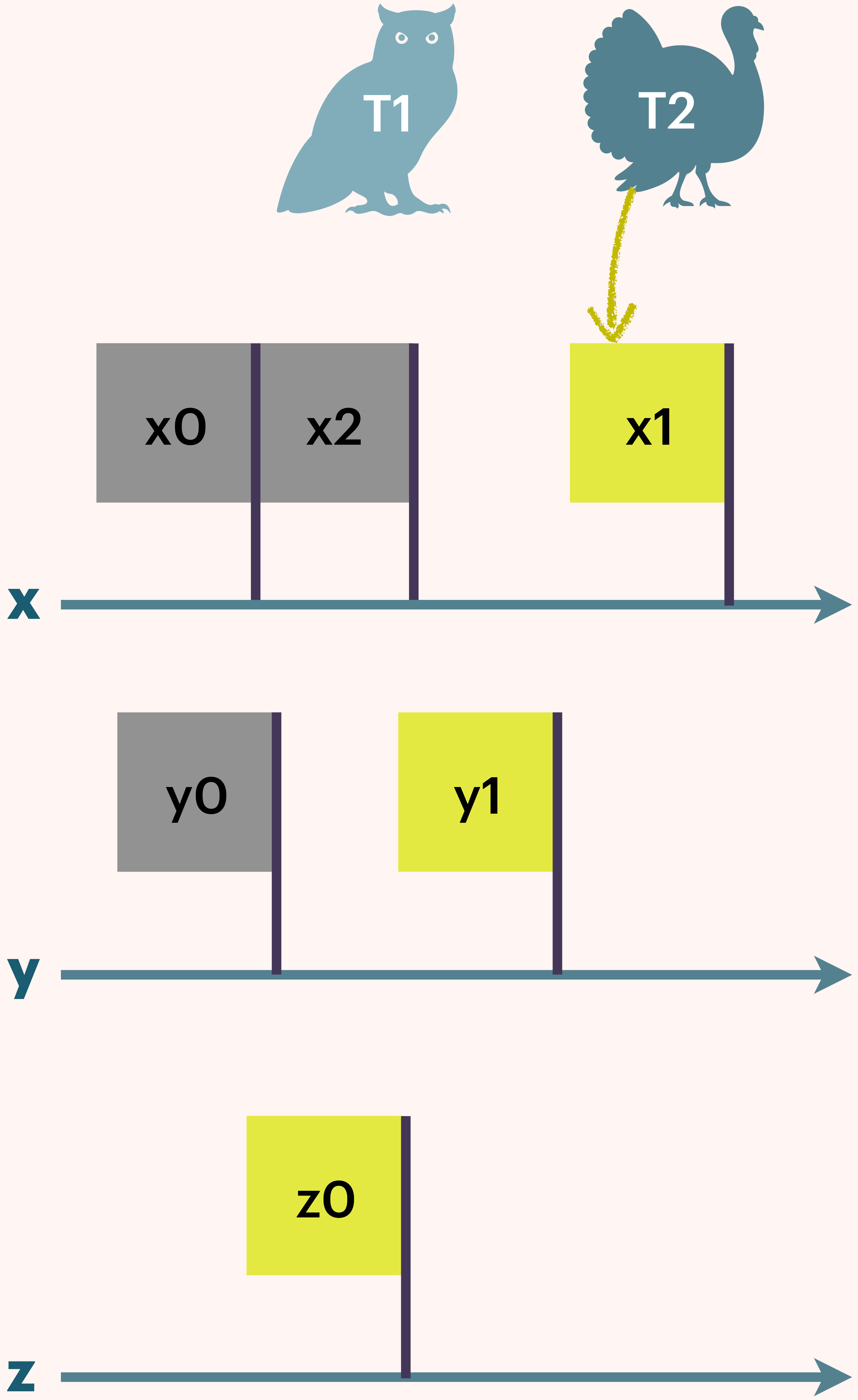
When reading, the message:

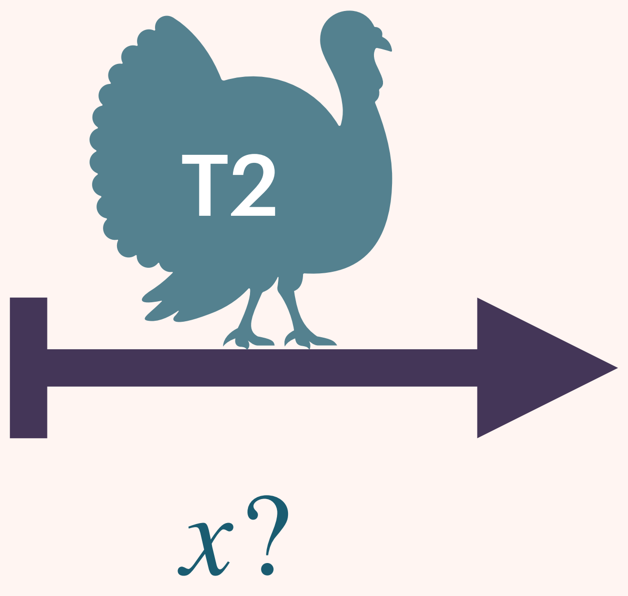
- **cannot be before thread's view**
- **may be before others**

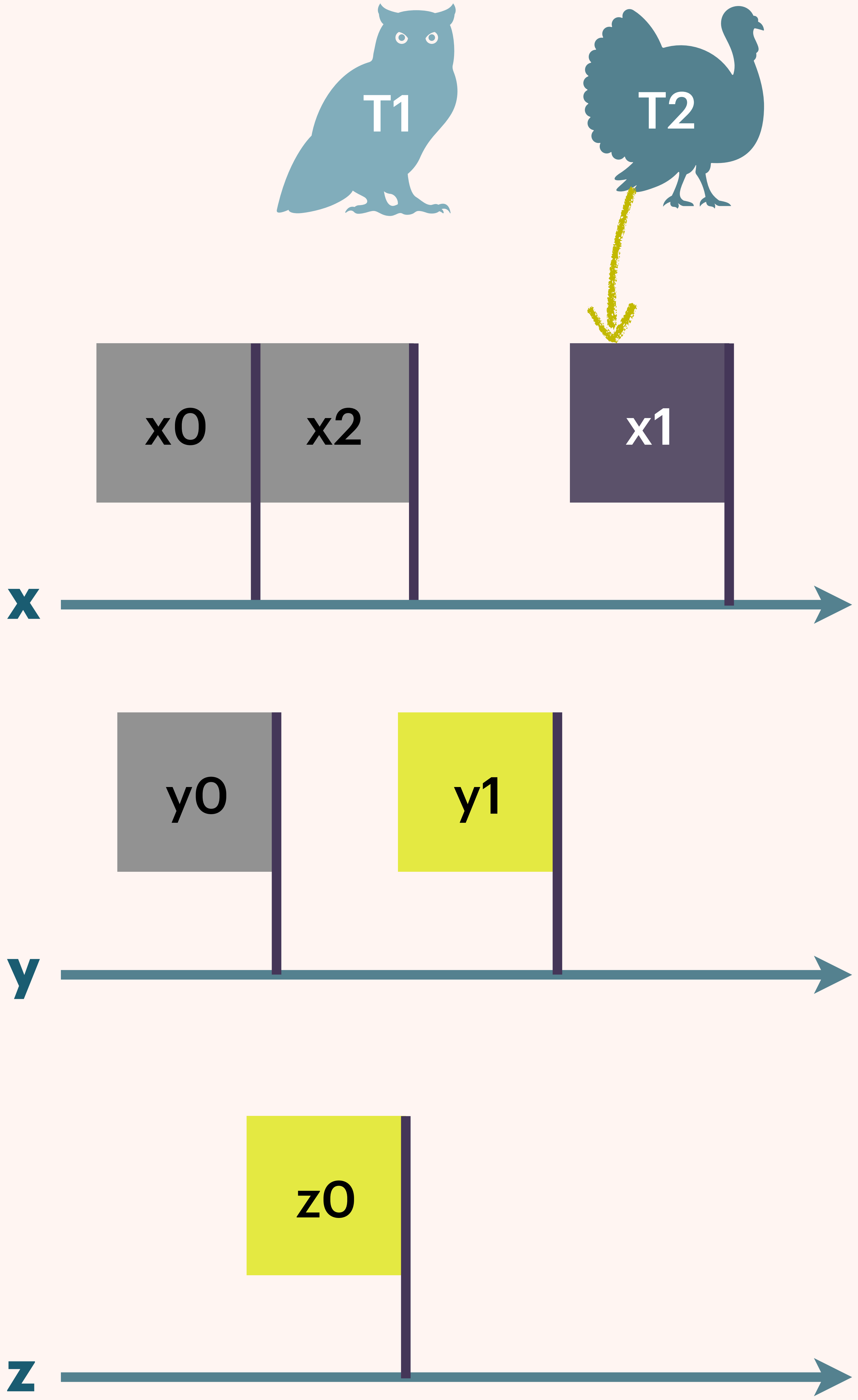
and the thread:

- **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**

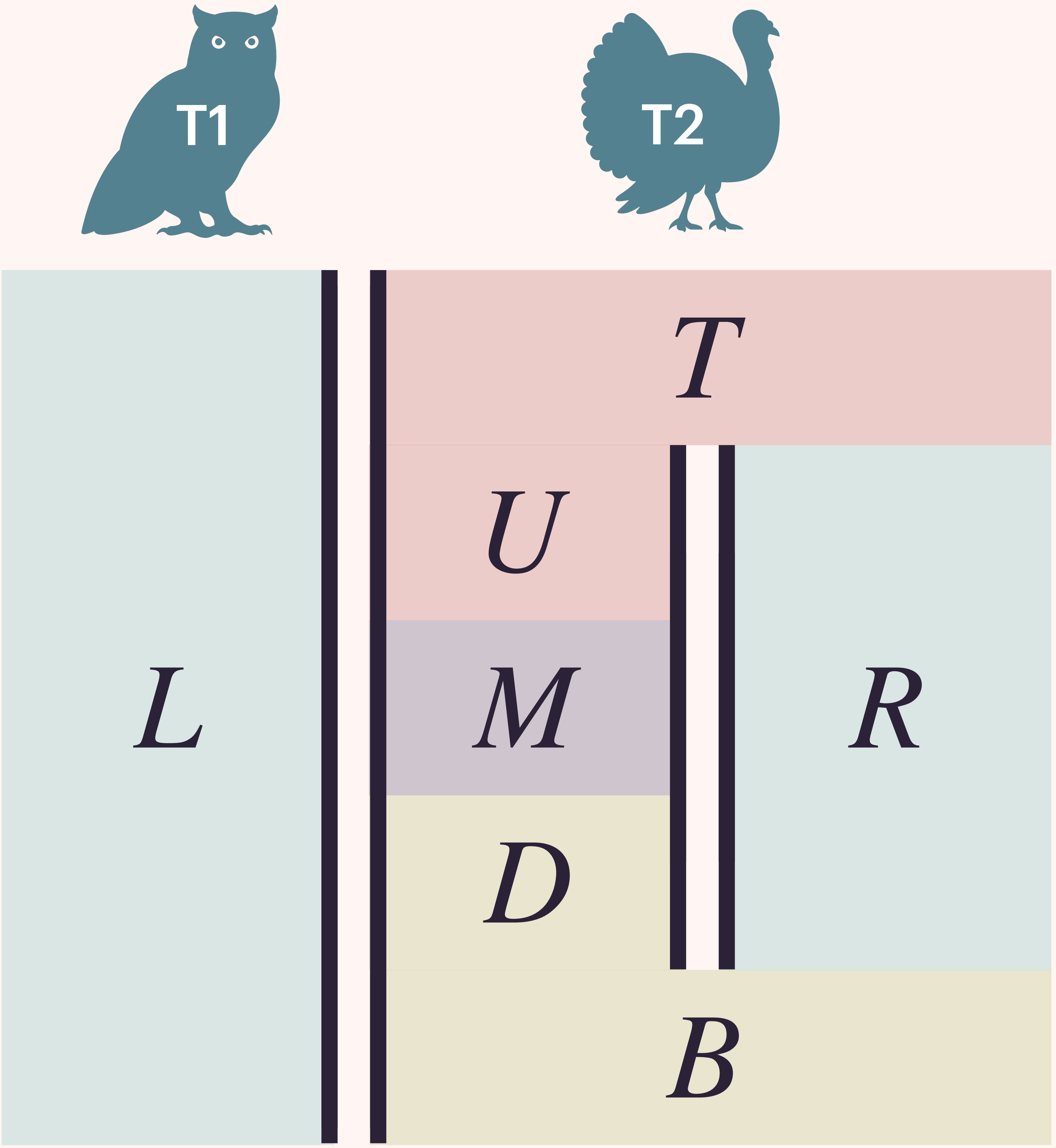
CAUSALITY AND COMPOSITION

With first class parallelism

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

Generalized Sequencing

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



TRACE-BASED SEMANTICS

TRACE-BASED SEMANTICS IN RA

Terms denote sets of traces

$$[\![M]\!] \ni \tau$$

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

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

Initial View — Sequence of Transitions — Final View — Returns

RA RA

TRACE-BASED SEMANTICS IN RA

Rely On μ_1 / **To Guarantee ϱ_1** — **Then** — **Rely On μ_2** / **To Guarantee ϱ_2** — **Then...**

Guarantee to the sequential environment to return r

Before or || — || — || — After

$$\alpha \langle \mu_1, \varrho_1 \rangle \langle \mu_2, \varrho_2 \rangle \ldots \langle \mu_{n-1}, \varrho_{n-1} \rangle \langle \mu_n, \varrho_n \rangle \, \omega \; .\!\!\because r$$

Initial View — **Sequence of Transitions** — **Final View** — **Returns**

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

Before

After

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

Initial View

Sequence of Transitions

Final View

Returns

TRANSITION CLOSURES

Analogous to Brookes's

Stutter

$$\frac{\alpha\,\boxed{\xi\eta}\,\omega \therefore r \in [\![M]\!]}{\alpha\,\boxed{\xi\langle\mu,\mu\rangle\eta}\,\omega \therefore r \in [\![M]\!]}$$

Propagate Reliance as a Guarantee

Mumble

$$\frac{\alpha\,\boxed{\xi\langle\mu,\rho\rangle\langle\rho,\theta\rangle\eta}\,\omega \therefore r \in [\![M]\!]}{\alpha\,\boxed{\xi\langle\mu,\theta\rangle\eta}\,\omega \therefore r \in [\![M]\!]}$$

Rely on an omitted Guarantee

VIEW CLOSURES

Specific to RA

Rewind

$$\frac{\alpha' \le \alpha \qquad \alpha \,\xi\, \omega \therefore r \in [\![M]\!]}{\alpha' \,\xi\, \omega \therefore r \in [\![M]\!]}$$

Relying on more being revealed

Forward

$$\frac{\alpha \,\xi\, \omega \therefore r \in [\![M]\!] \qquad \omega \le \omega'}{\alpha \,\xi\, \omega' \therefore r \in [\![M]\!]}$$

Guaranteeing less being revealed

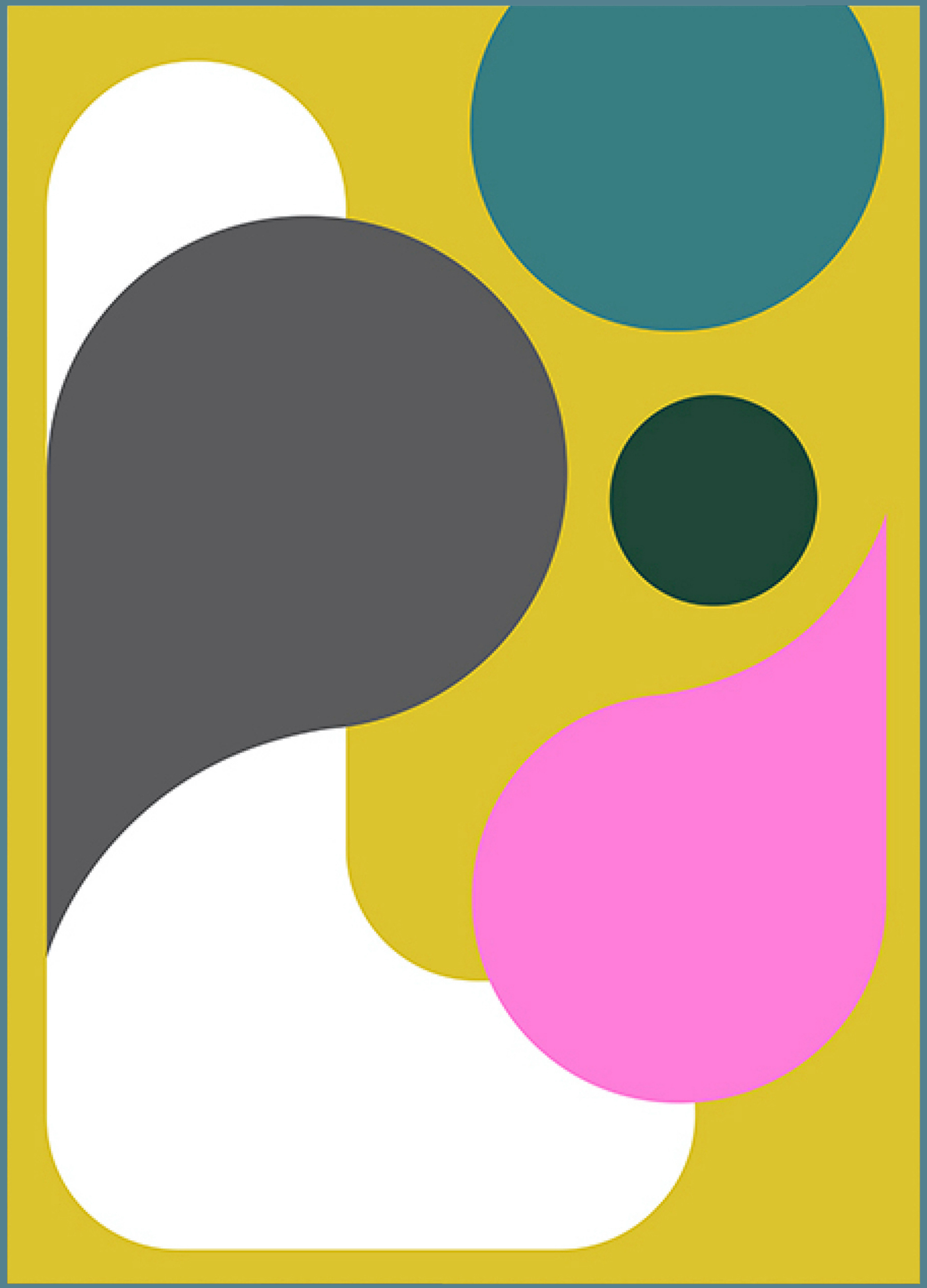
COMPOSITION

Sequential

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

Parallel

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



ABSTRACTION

WHAT WE CAN JUSTIFY

with Stutter, Mumble, Rewind, and Forward

Standard Semantics

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

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

First-class parallelism

Laws of Parallel Programming, e.g. Generalized Sequencing

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

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

$$[\![\textbf{let } a = x? \textbf{ in let } b = x? \textbf{ in } \langle a, b \rangle]\!] \supseteq [\![\textbf{let } c = x? \textbf{ in } \langle c, c \rangle]\!]$$

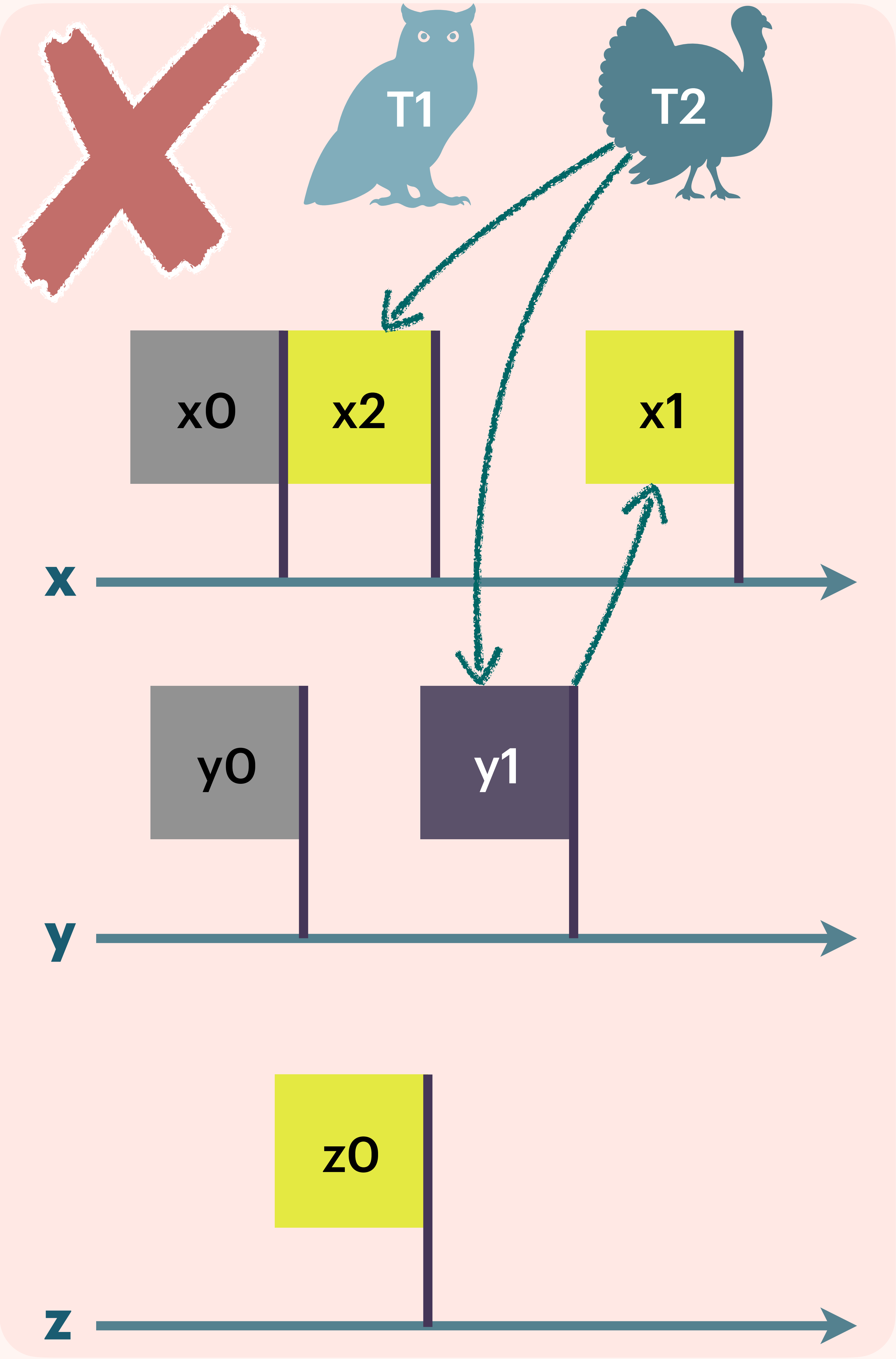
SEMANTIC INVARIANTS ON TRACES

Read Elimination

$$x?; M \twoheadrightarrow M$$

operational invariant becomes denotational requirement

views point to messages that carry a smaller view



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

MORE CLOSURES

RA Specific
Compiler Optimization

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

Write-Read Reorder

$$x := 1;\ \mathbf{let}\ a = y?\ \mathbf{in}\ M \quad \twoheadrightarrow \quad \mathbf{let}\ a = y?\ \mathbf{in}\ x := 1;\ M$$

$\leq$

View in message at x

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

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

ABSTRACT CLOSURES

Specific to RA

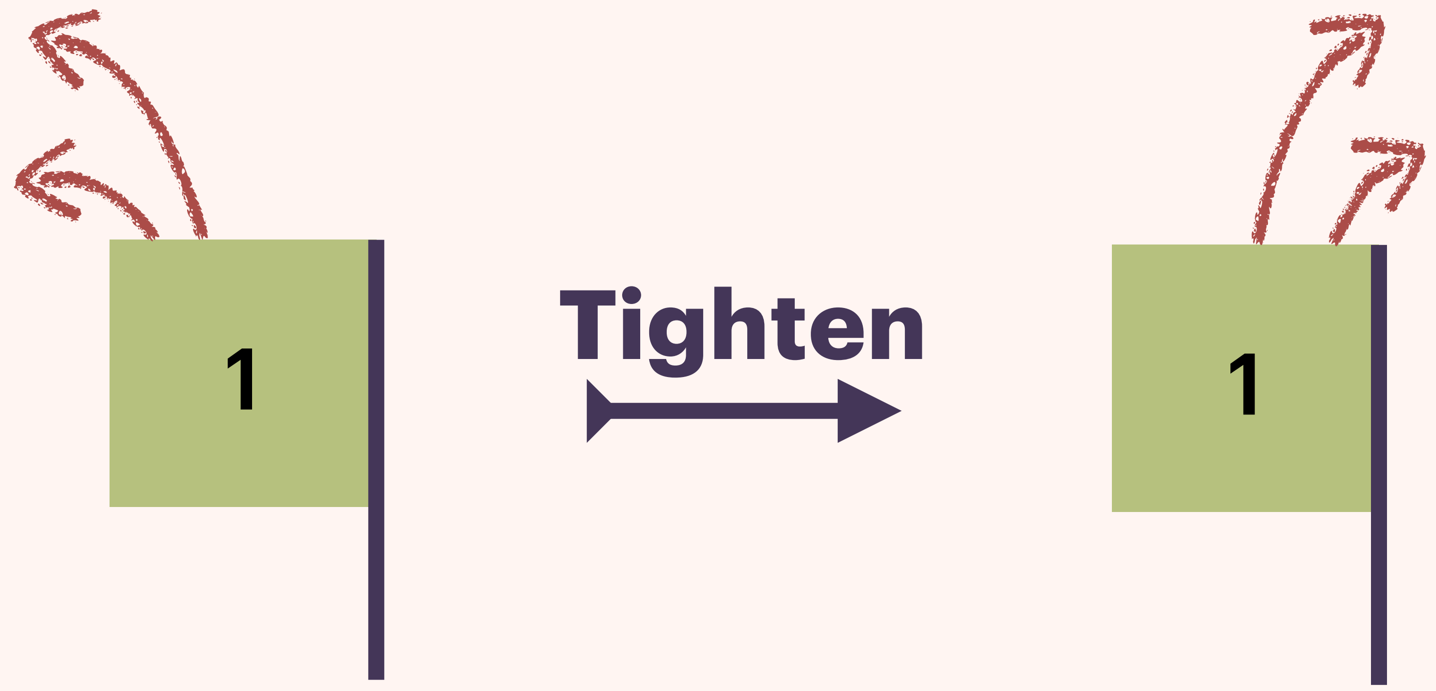
- ***Absorb*** **a redundant local message into a following one**
  (e.g. $[\![x := 0; x := 1]\!] \supseteq [\![x := 1]\!]$)
- ***Dilute*** **a message by a redundant local message**
  (e.g. $[\![x?]\!] \supseteq [\![\mathrm{FAA}[x](0)]\!]$)
- ***Tighten*** **the encumbering view that a local message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

Rewrite

$$\frac{\pi \in [\![M]\!] \qquad \pi \longmapsto \tau}{\tau \in [\![M]\!]}$$

ABSTRACT REWRITE RULES

Write-Read Deorder + LoPP + Struct ⇒ Write-Read Reorder



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

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

NEW ADEQUACY PROOF IDEA

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

Laws of Parallel Programming

Symmetry	$M \parallel N$	$\twoheadrightarrow$	$\mathbf{match}\, N \parallel M\ \mathbf{with}\ \langle y, x\rangle.\ \langle x, y\rangle$
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	$\langle(\ell := v)\,, \ell'?\rangle$	$\overset{\mathsf{Ti}}{\twoheadrightarrow}$	$\mathbf{let}\, x = \ell'?\, \mathbf{in}\, (\ell := v)\, ;\, x$	$(\ell \neq \ell')$

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)

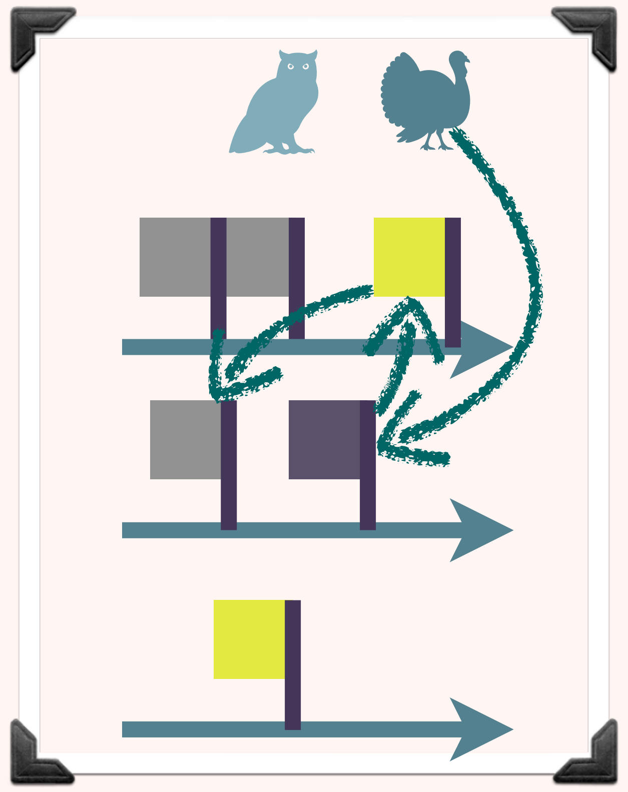
LIMITATIONS

- 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

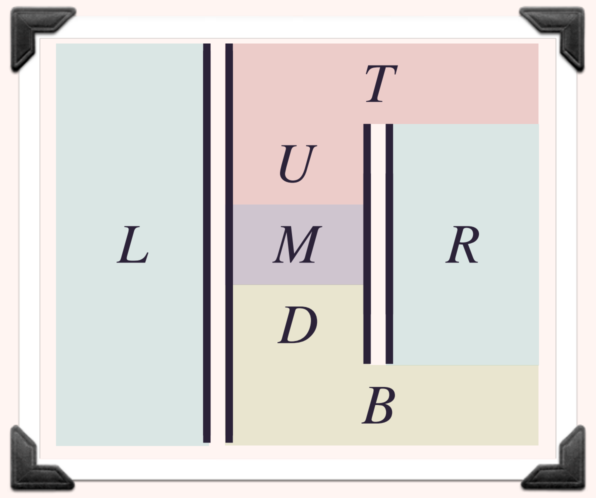
FUTURE DIRECTIONS

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

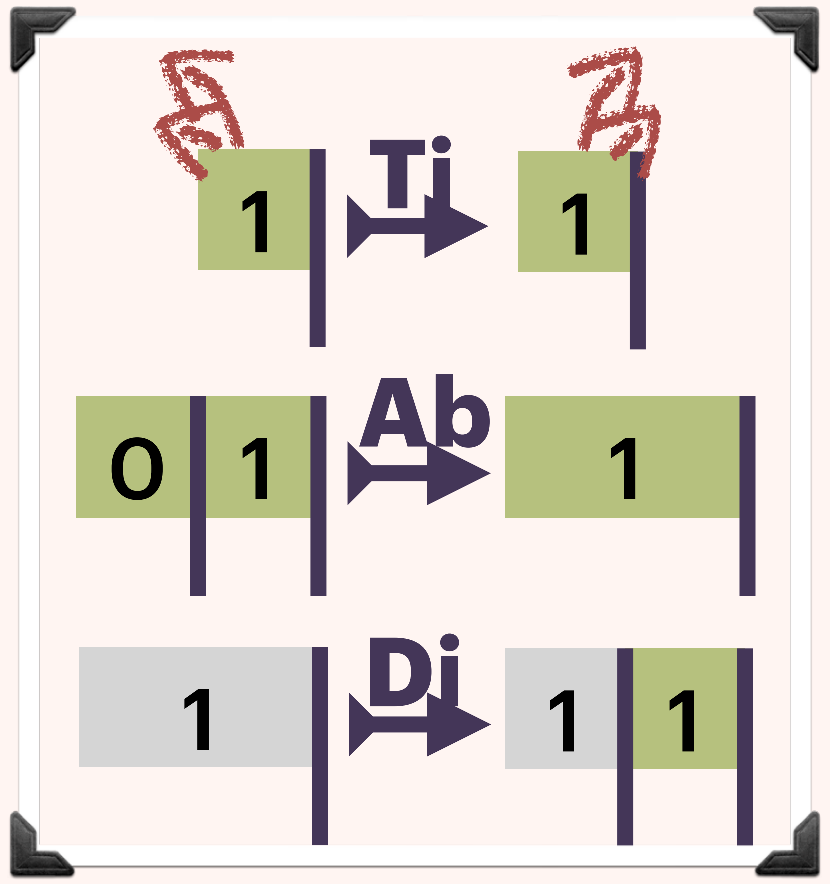
$$
\begin{aligned}
(\!|-|\!) &: \mathbf{Term}_{\{\mathrm{L},\mathrm{U}\}} X \to \mathcal{P}_{\mathrm{fin}}\left(\mathbb{T} X\right) \\
(\!|x|\!) &:= \{\langle\rangle \mathbin{.\!\cdot\!.} x\} \\
(\!|\mathrm{L}_\ell \langle t_v \rangle_{v \in \mathbf{Val}}|\!) &:= \bigcup_{v \in \mathbf{Val}} \{(\mathrm{R}_{\ell,v} :: \mathtt{t}) \mathbin{.\!\cdot\!.} x \mid \mathtt{t} \mathbin{.\!\cdot\!.} x \in (\!|t_v|\!)\} \\
(\!|\mathrm{U}_{\ell,v} t|\!) &:= \{(\mathrm{G}_{\ell,v} :: \mathtt{t}) \mathbin{.\!\cdot\!.} x \mid \mathtt{t} \mathbin{.\!\cdot\!.} x \in (\!|t|\!)\}
\end{aligned}
$$



OPERATIONAL SEMANTICS

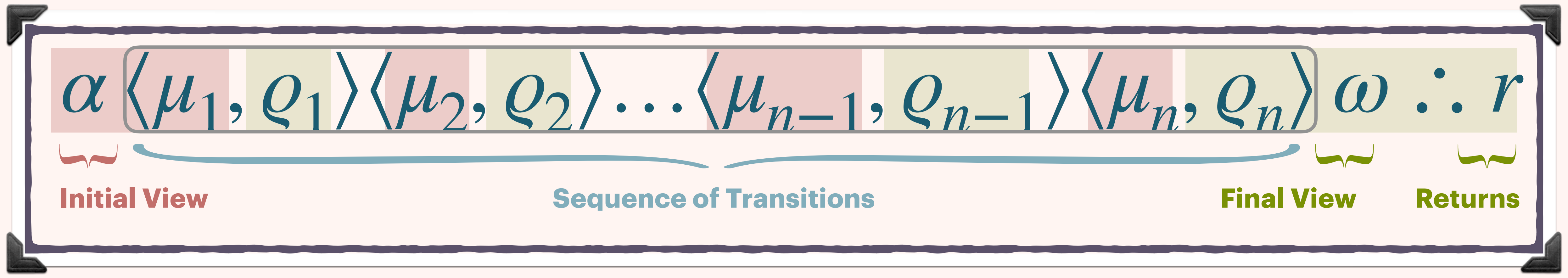


1ST-CLASS PARALLELISM



ABSTRACT CLOSURES

ADEQUACY PROOF



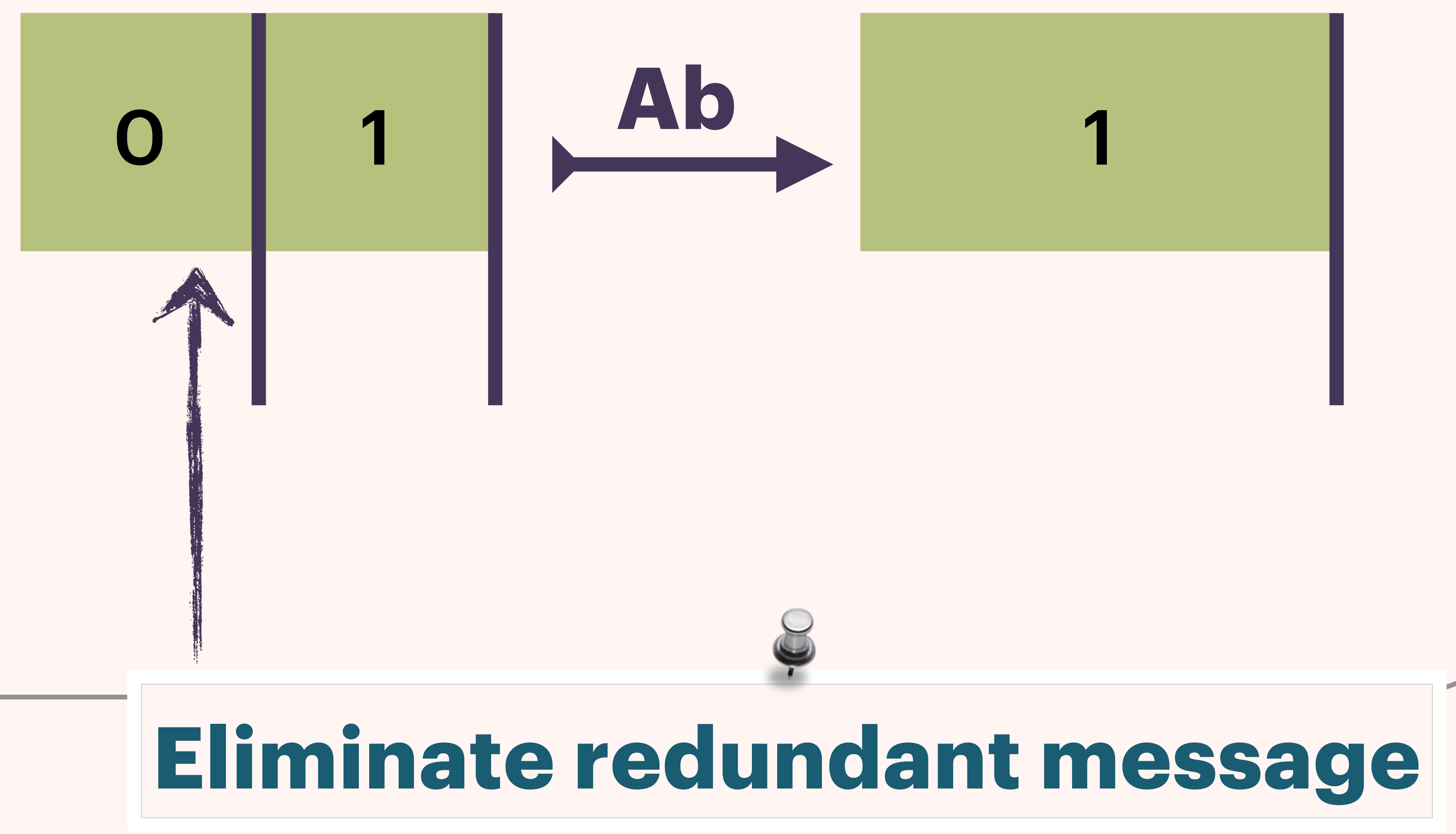
RELY/GUARANTEE TRACES

REWRITE RULE: ABSORB

Write Eliminations

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

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

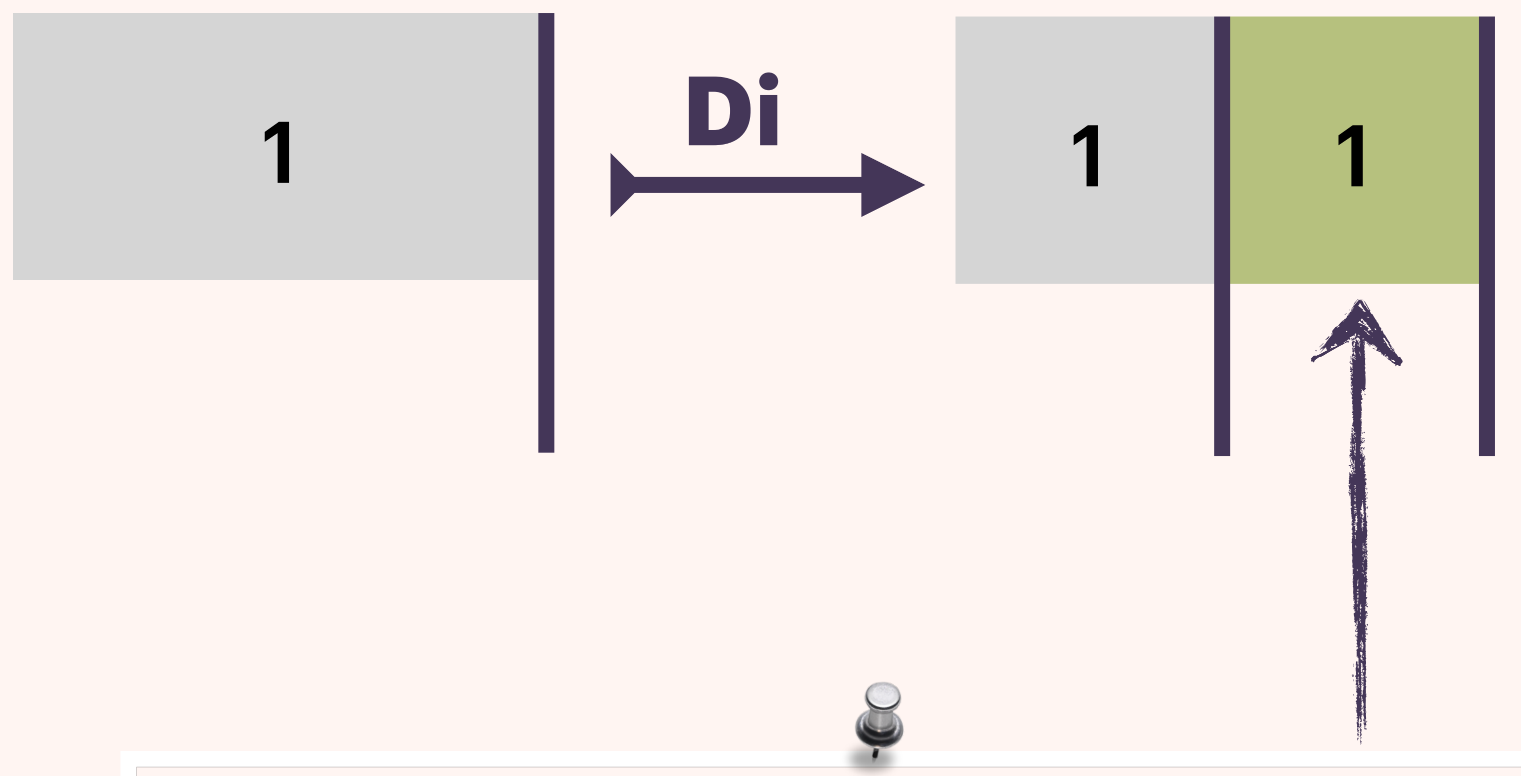


REWRITE RULE: DILUTE

Write Eliminations

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

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



Introduce redundant message