From Quasi-Synchrony to LTTA

Guillaume Baudart
Timothy Bourke
Marc Pouzet

Synchron'13

Dagstuhl, 11.21.2013

Context

Real-Time Fidelity in Synchronous Languages

Study Quasi-Synchrony as an example

- Mix of real- and discrete-time
- Notion of tolerance (e.g., “jitter”)

Outline

1. Quasi-Synchrony
2. Discrete Abstraction
3. Loosely Time-Triggered Architecture

Quasi-Synchrony

[Caspi 2000]

Quasi-Periodic Architecture

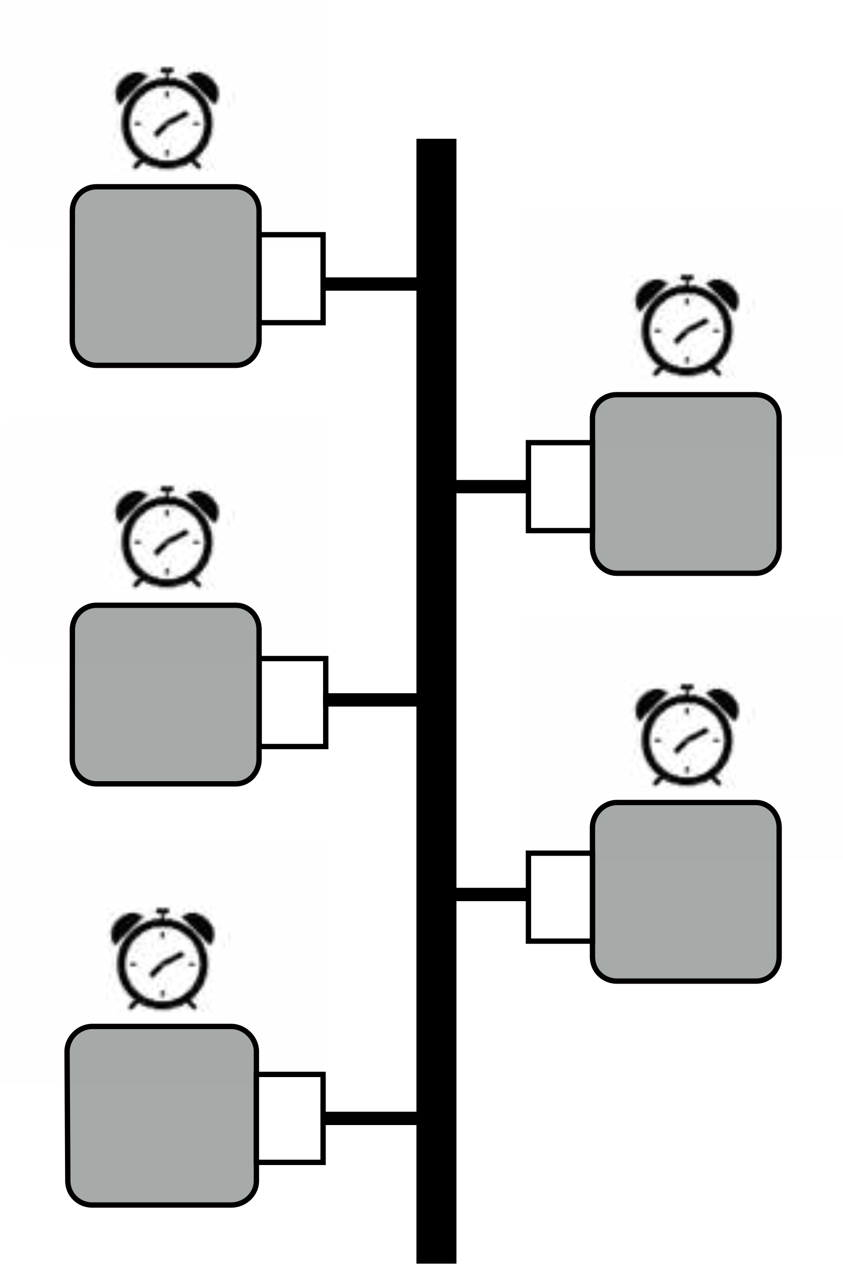
- A set of "quasi-periodic" processes with local clocks and nominal period T^n (jitter ε)

$$0 < T_{\min} \leq T^n \leq T_{\max} \quad \text{or}$$
$$T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon$$

$(\kappa_i)_{i\in\mathbb{N}}$ clock activations

- Buffered bus-based communication

- Bounded communication delay



Quasi-Periodic Architecture

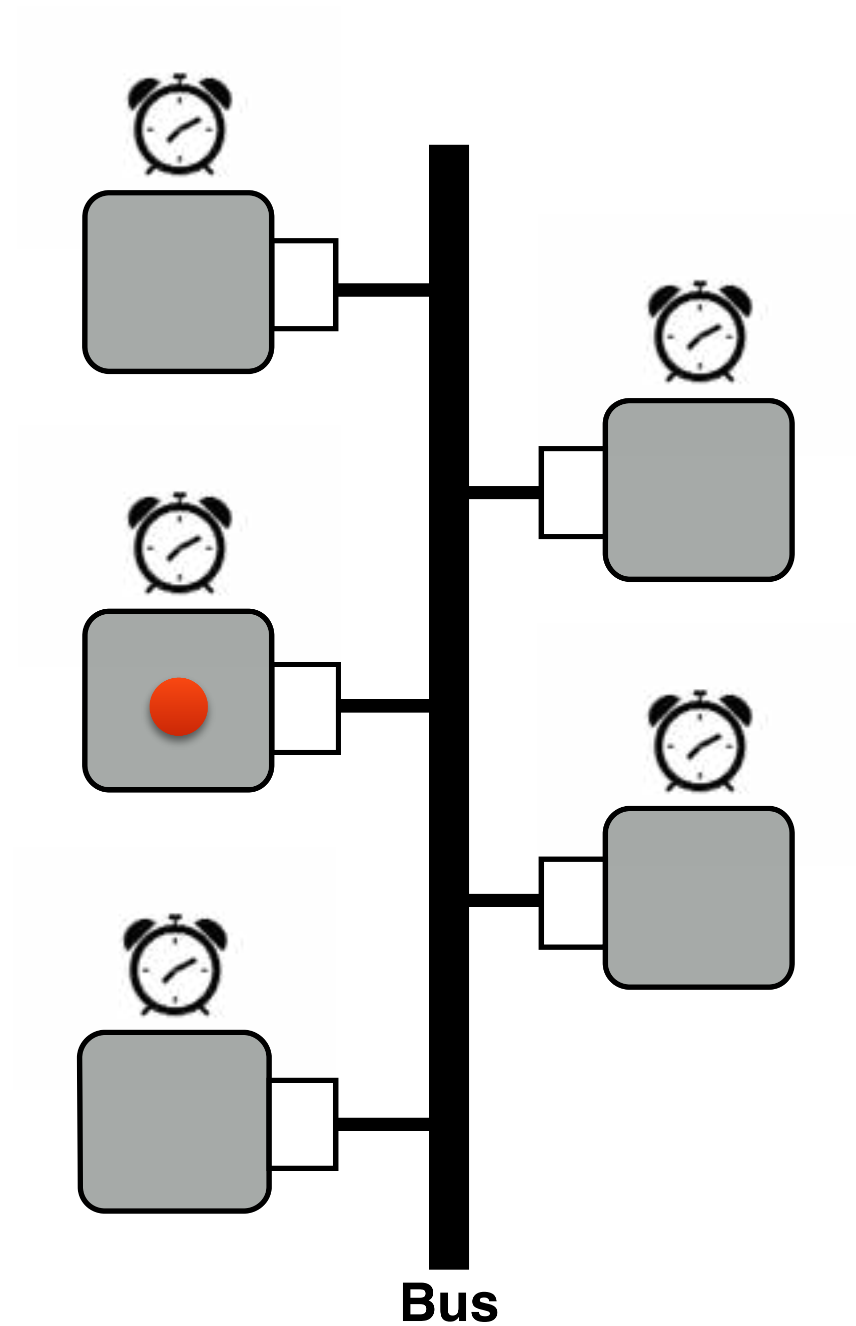
- A set of "quasi-periodic" processes with local clocks and nominal period T^n (jitter ε)

$$0 < T_{\min} \leq T^n \leq T_{\max} \quad \text{or}$$
$$T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon$$

$(\kappa_i)_{i \in \mathbb{N}}$ clock activations

- Buffered bus-based communication

- Bounded communication delay



Quasi-Periodic Architecture

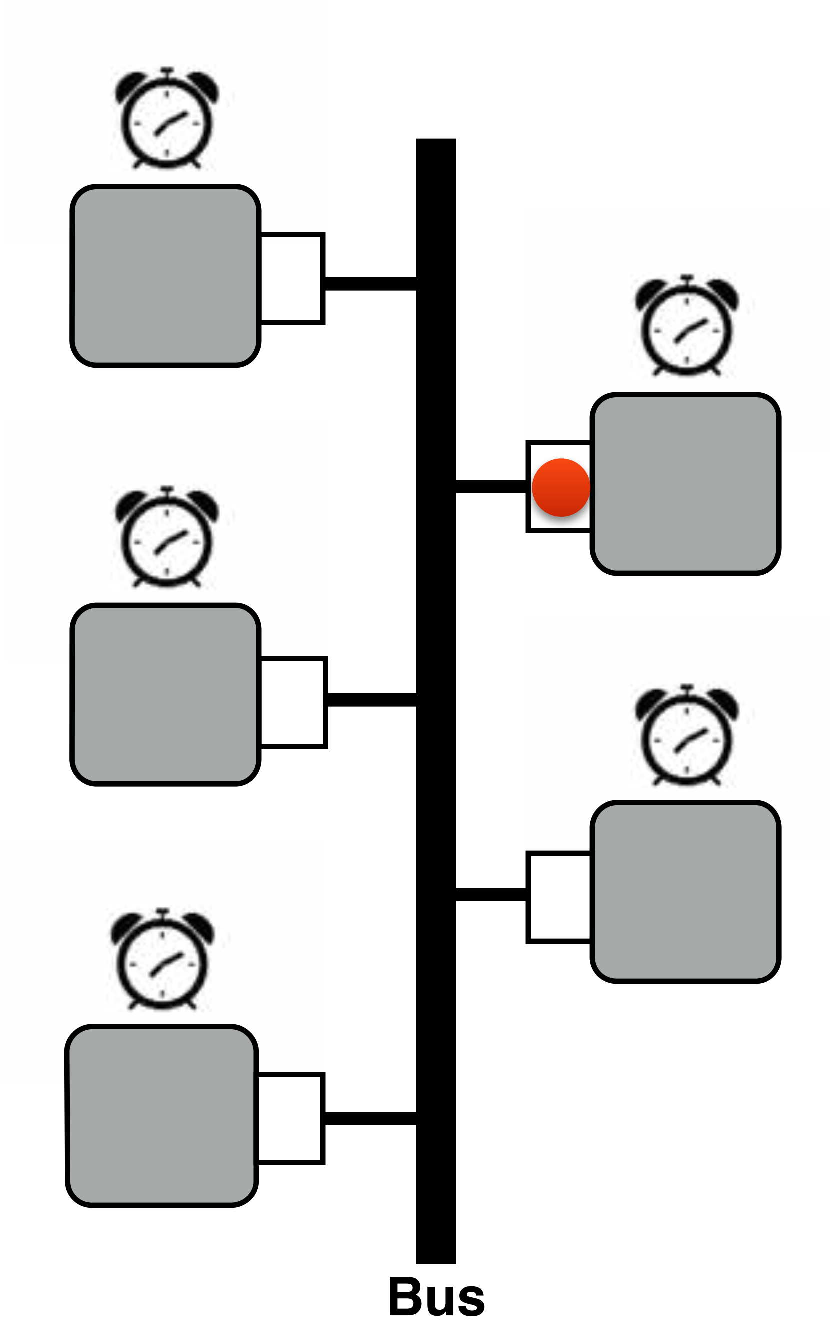
- A set of "quasi-periodic" processes with local clocks and nominal period T^n (jitter ε)

$$0 < T_{\min} \leq T^n \leq T_{\max} \quad \text{or}$$
$$T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon$$

$(\kappa_i)_{i \in \mathbb{N}}$ clock activations

- Buffered bus-based communication

- Bounded communication delay



Quasi-Periodic Architecture

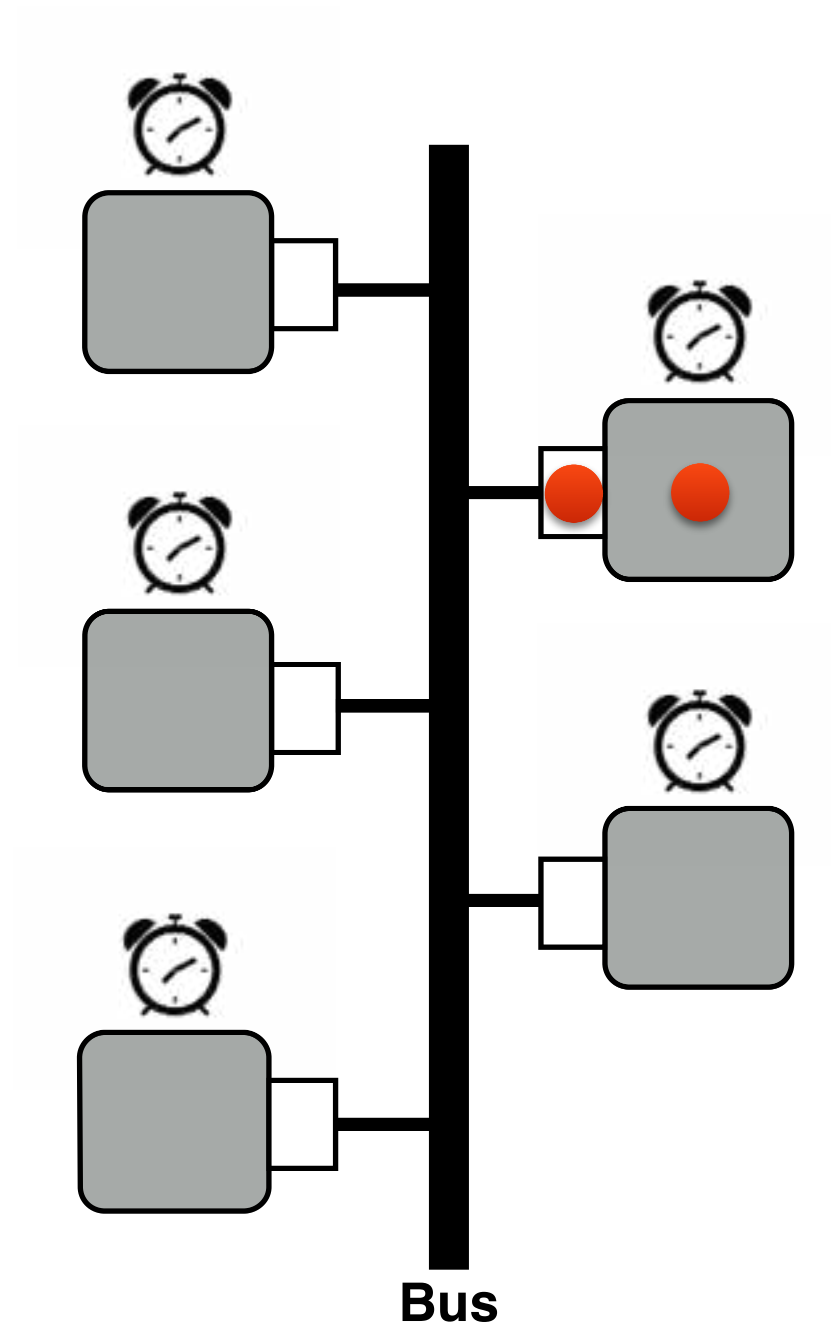
- A set of “quasi-periodic” processes with local clocks and nominal period T^n (jitter ε)

$$0 < T_{\min} \leq T^n \leq T_{\max} \quad \text{or}$$
$$T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon$$

$(\kappa_i)_{i \in \mathbb{N}}$ clock activations

- Buffered bus-based communication

- Bounded communication delay

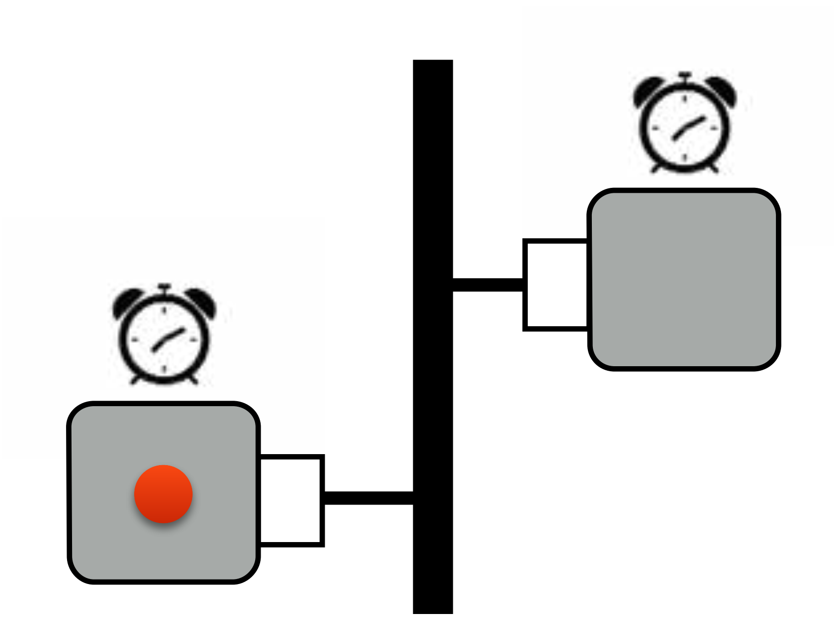


Two phenomena

Overwriting

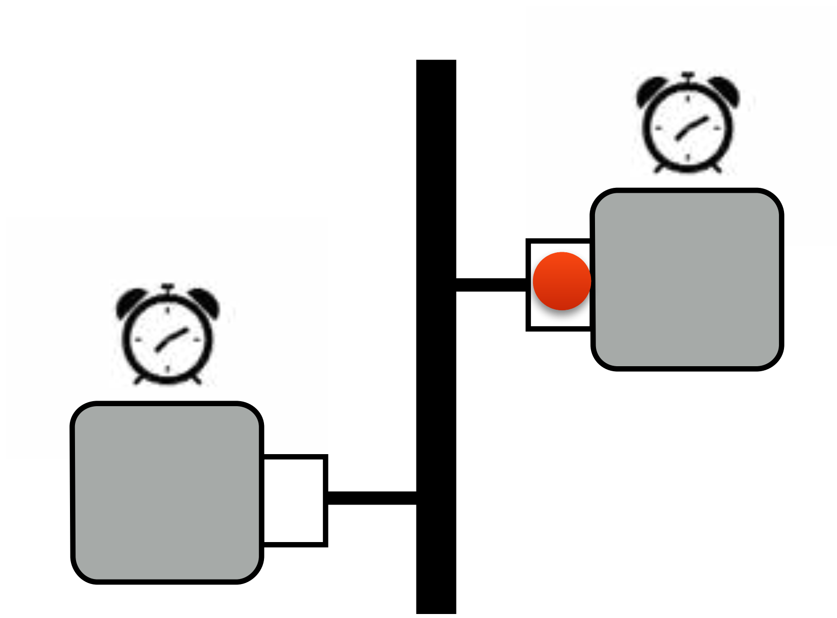
Two phenomena

Overwriting



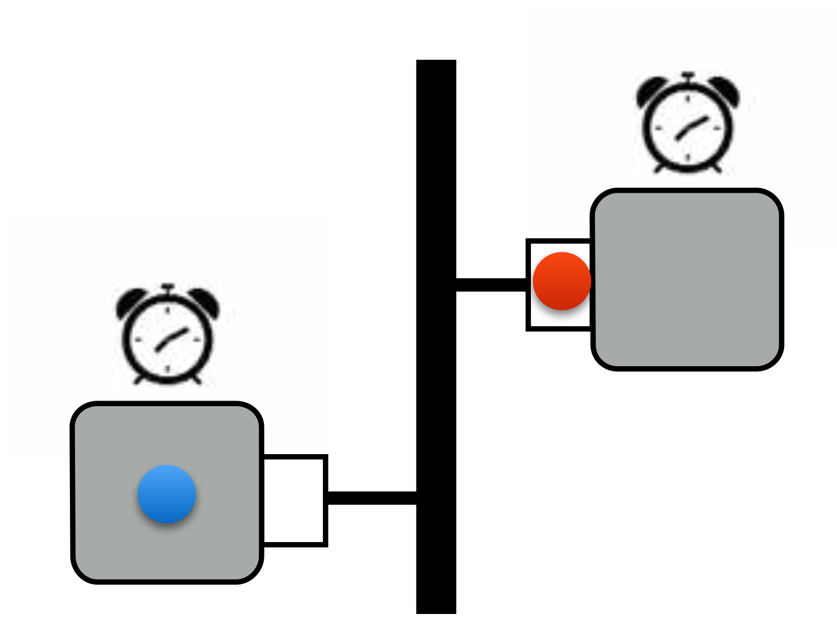
Two phenomena

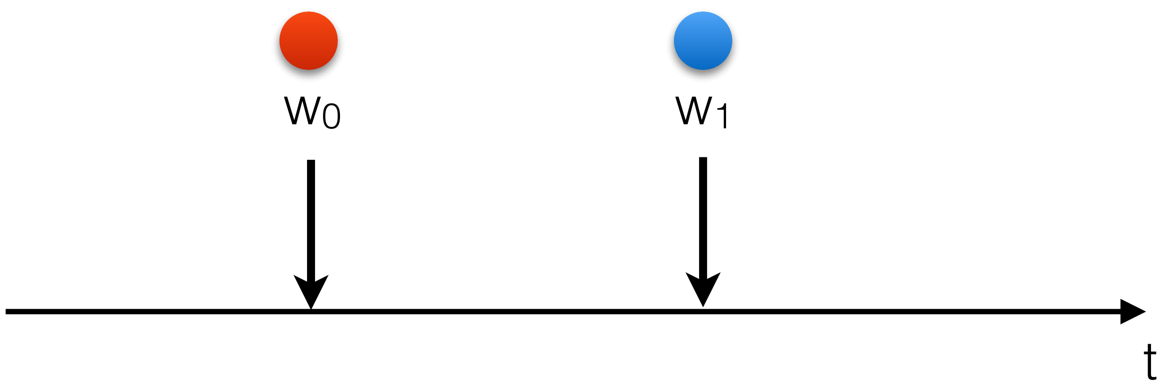
Overwriting



Two phenomena

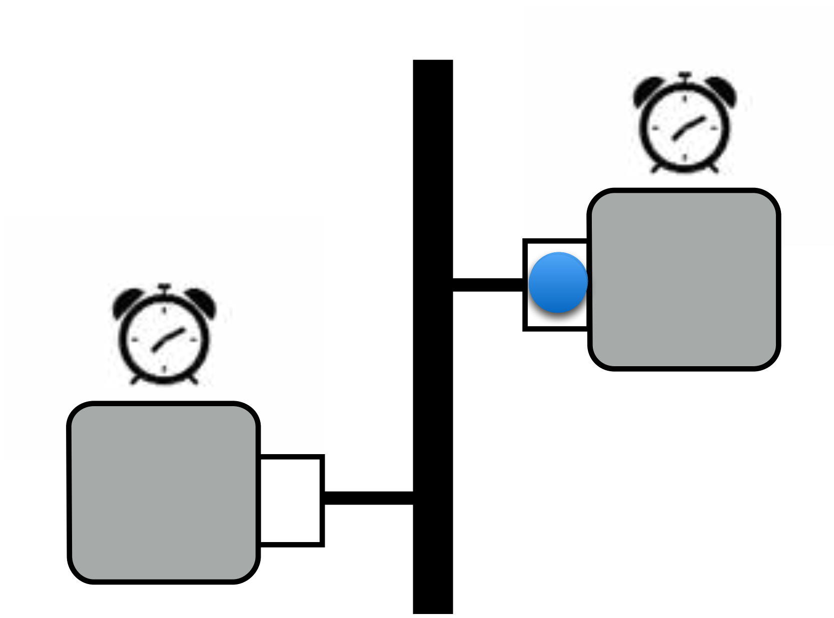
Overwriting

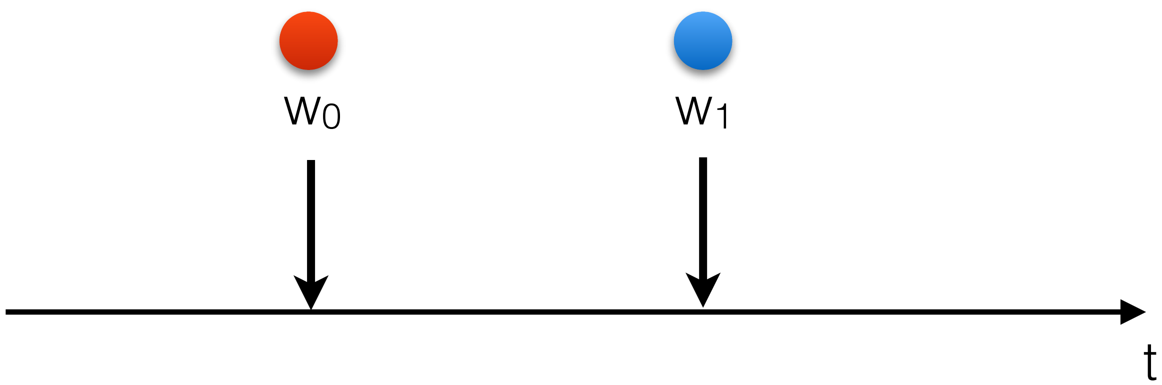




Two phenomena

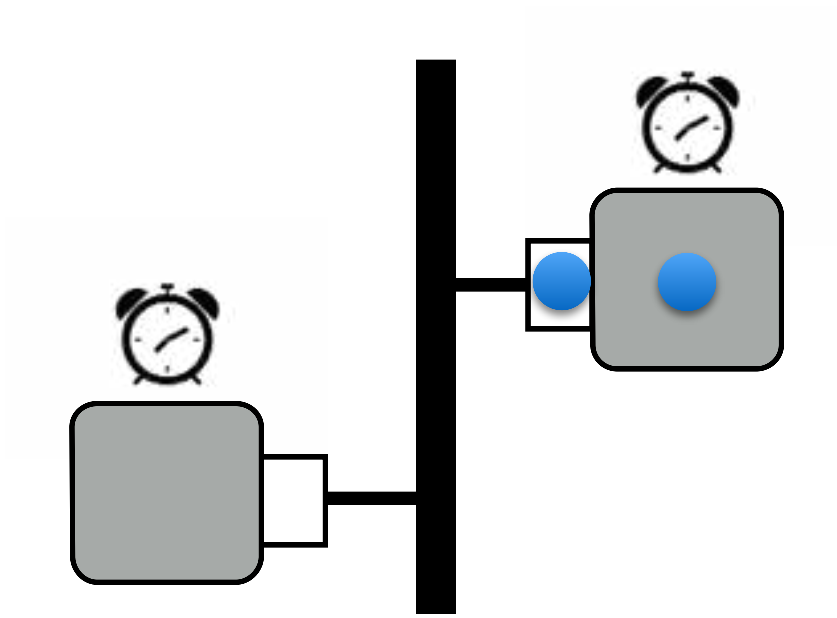
Overwriting

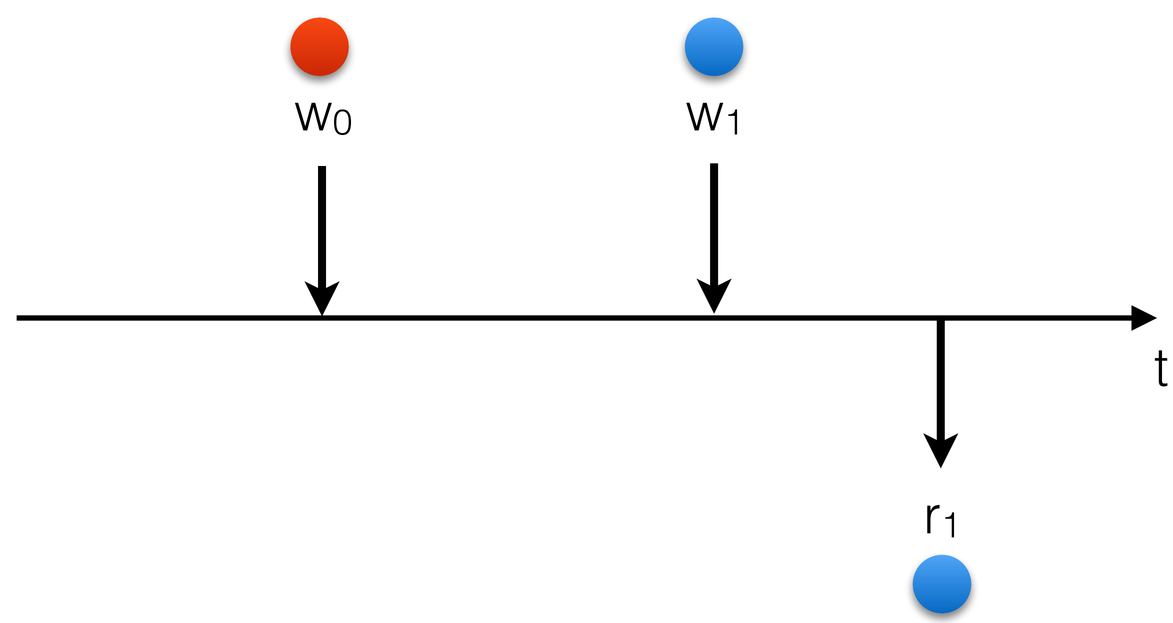




Two phenomena

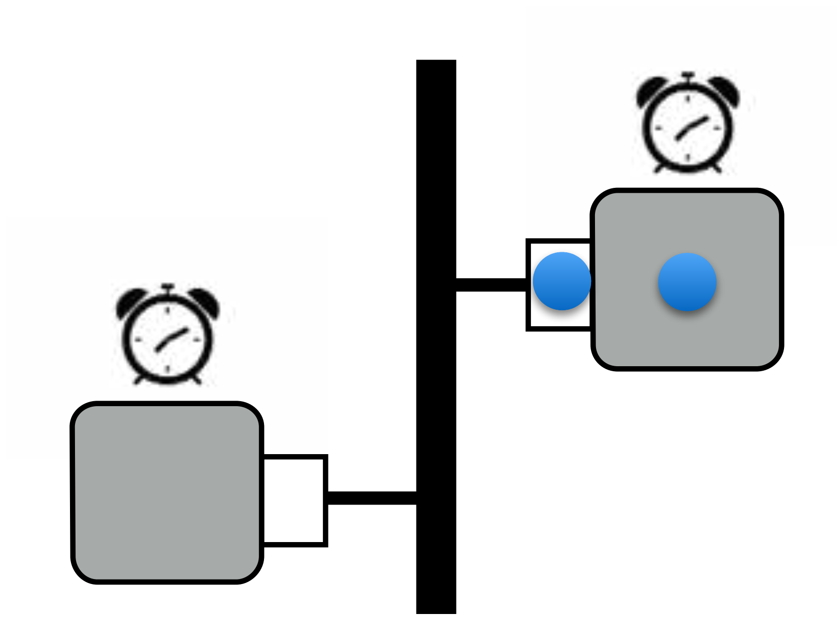
Overwriting

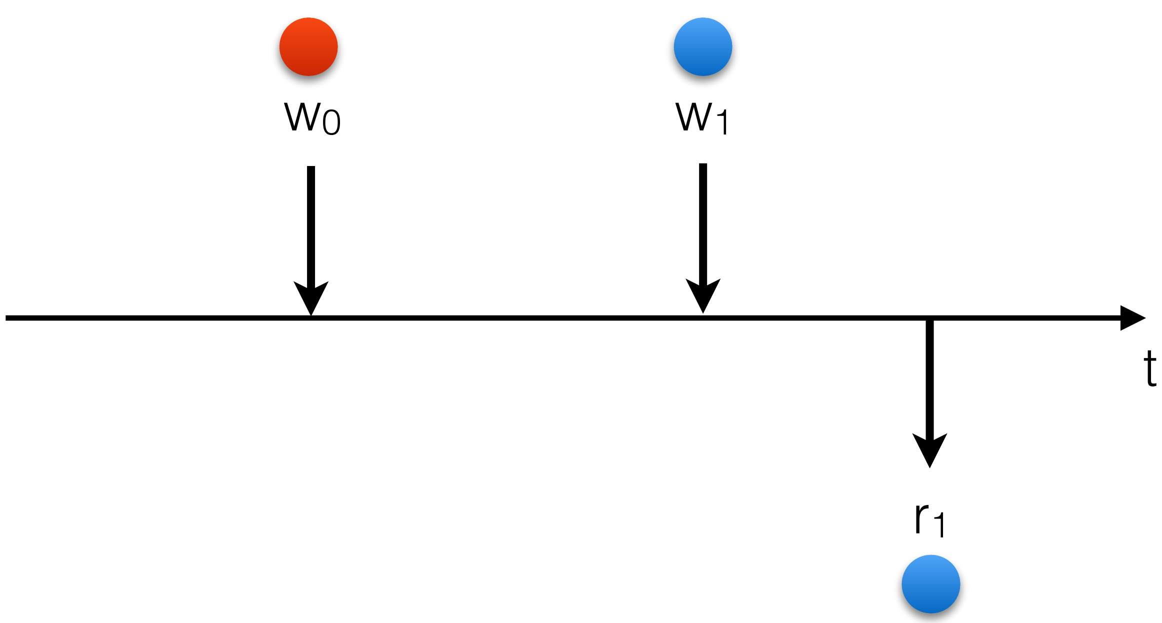




Two phenomena

Overwriting

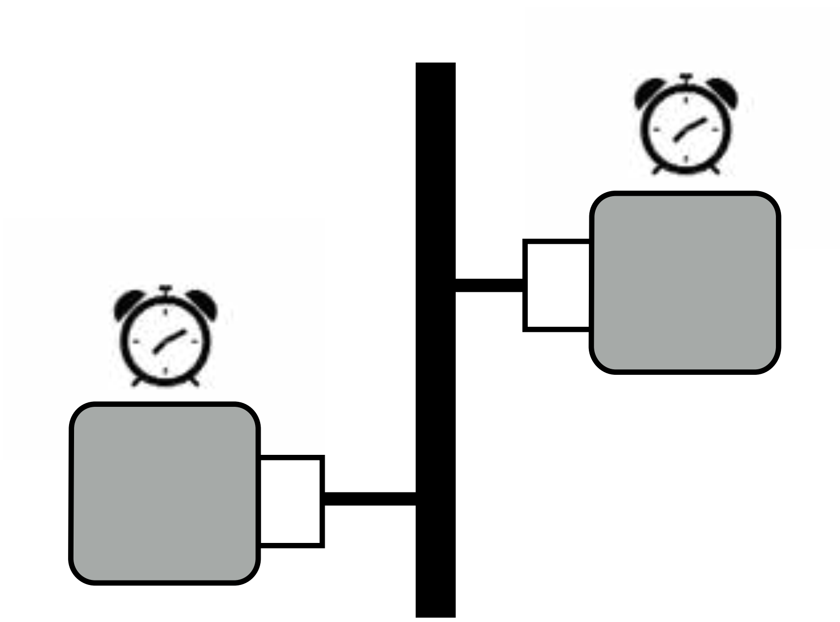


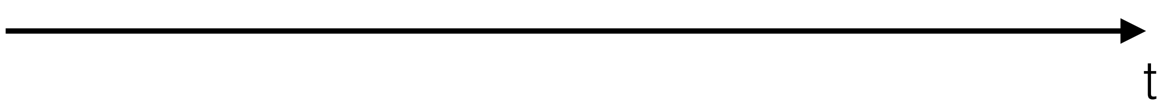


missed

Two phenomena

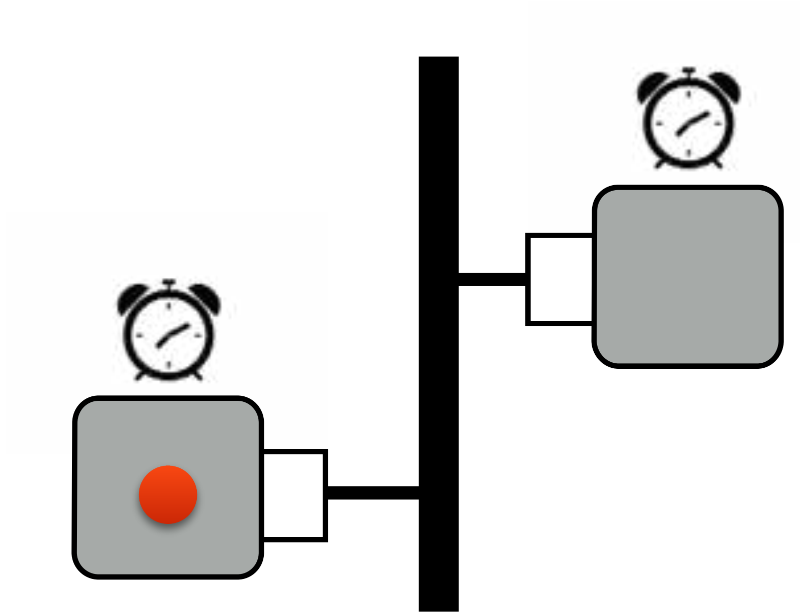
Oversampling





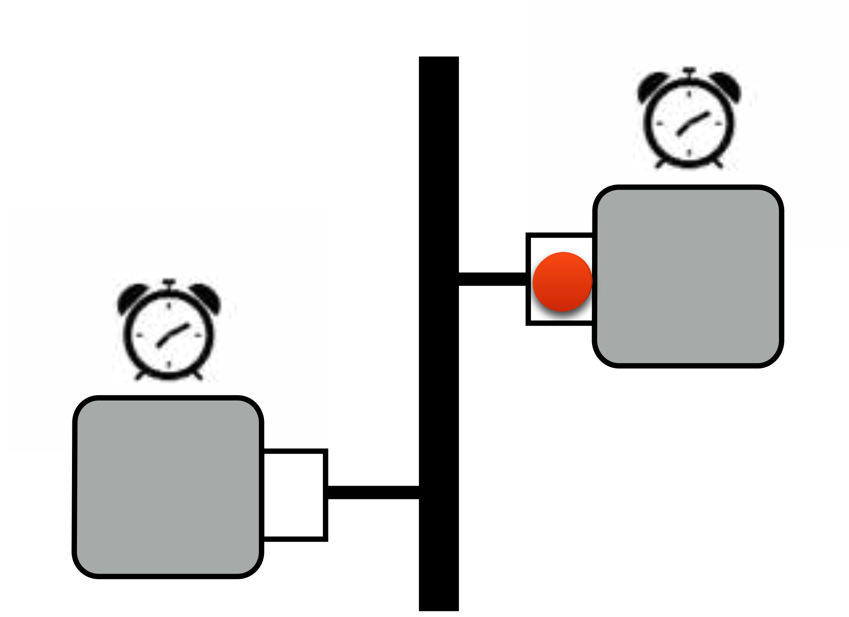
Two phenomena

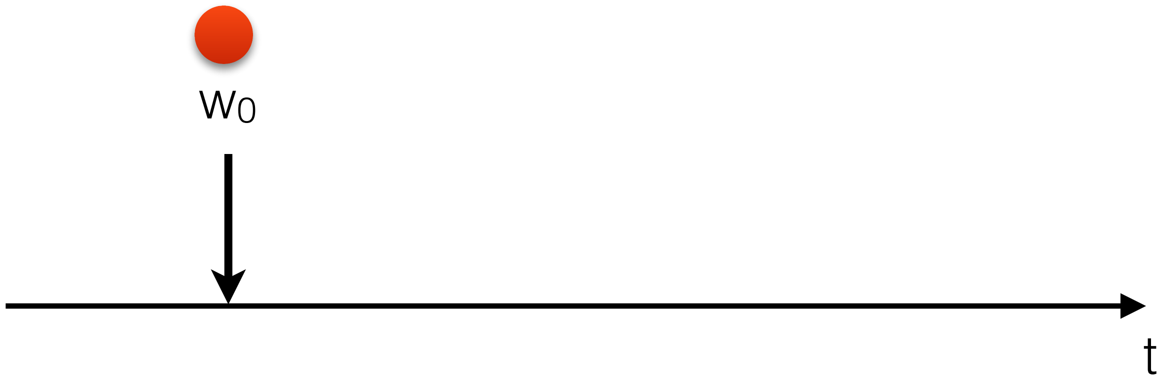
Oversampling



Two phenomena

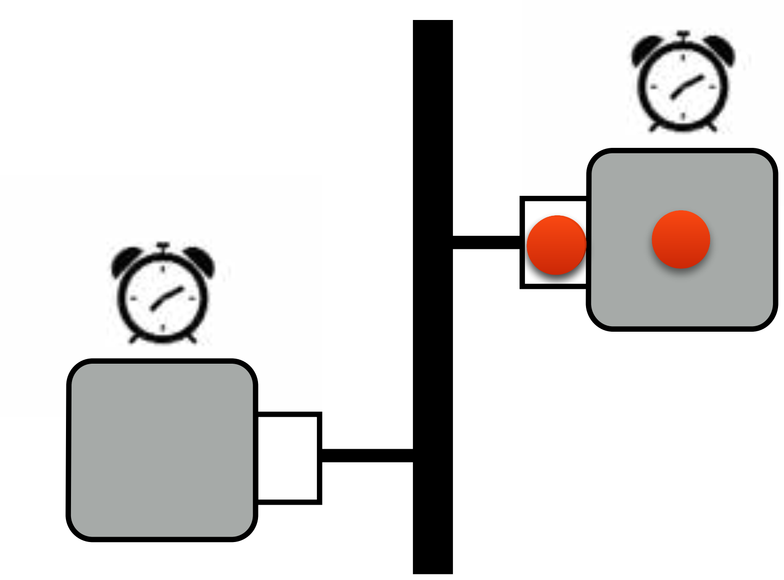
Oversampling

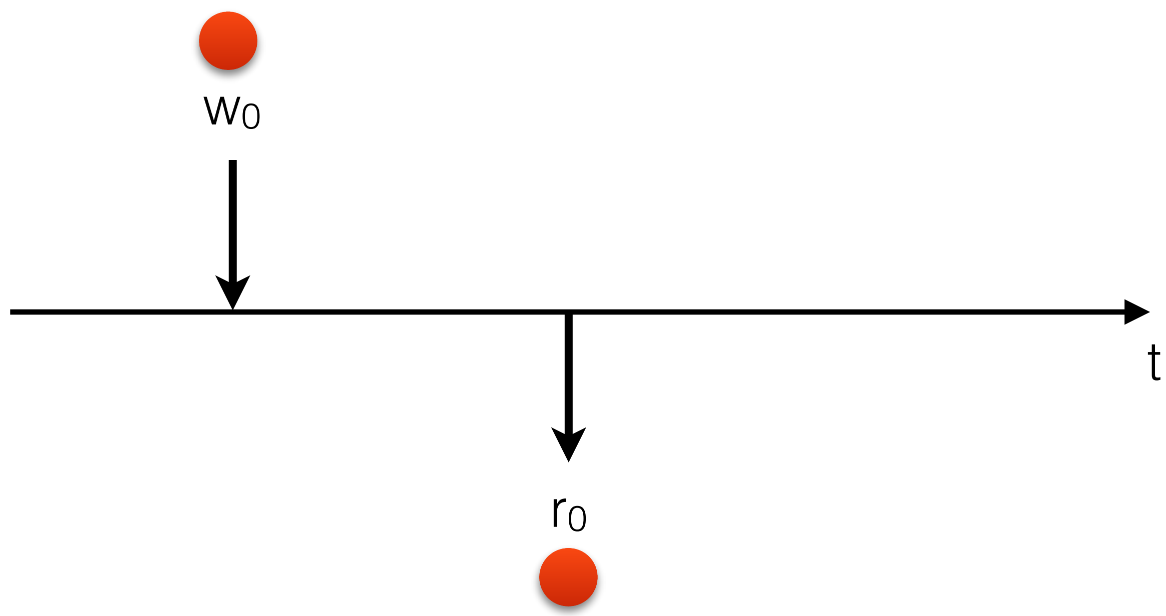




Two phenomena

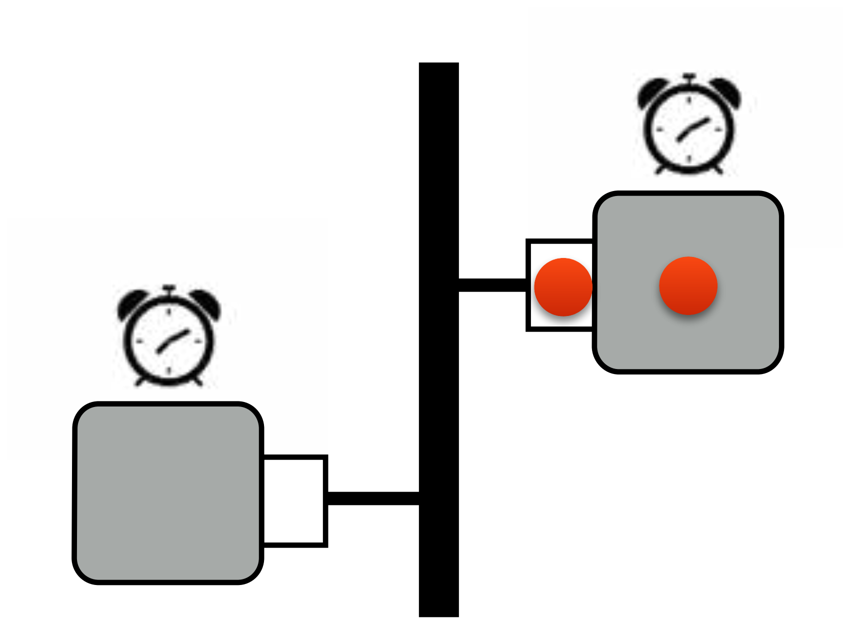
Oversampling

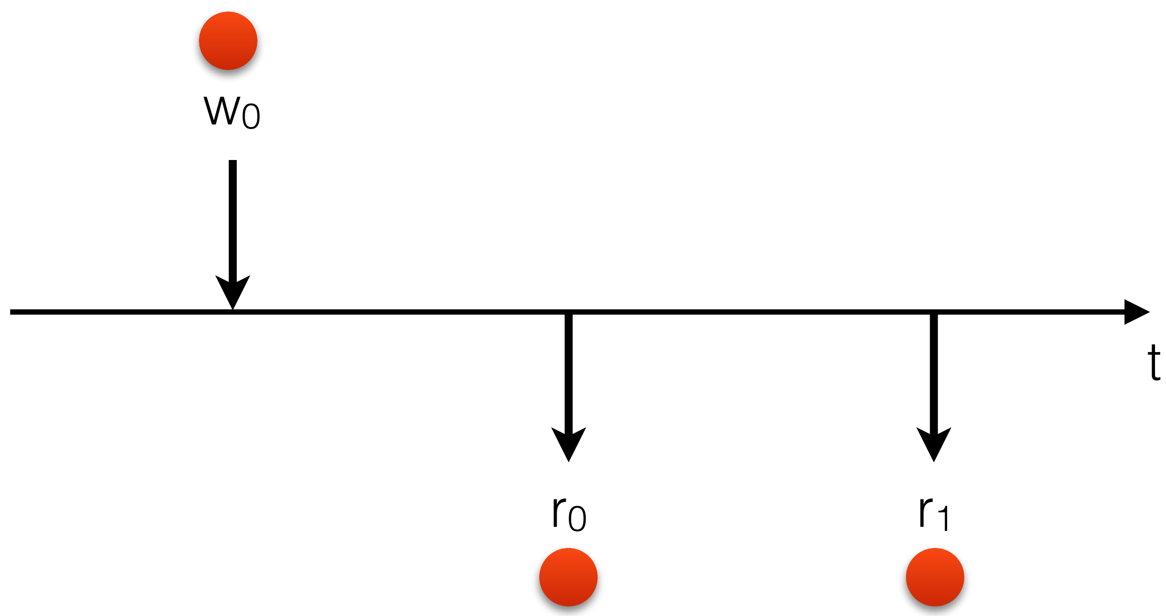




Two phenomena

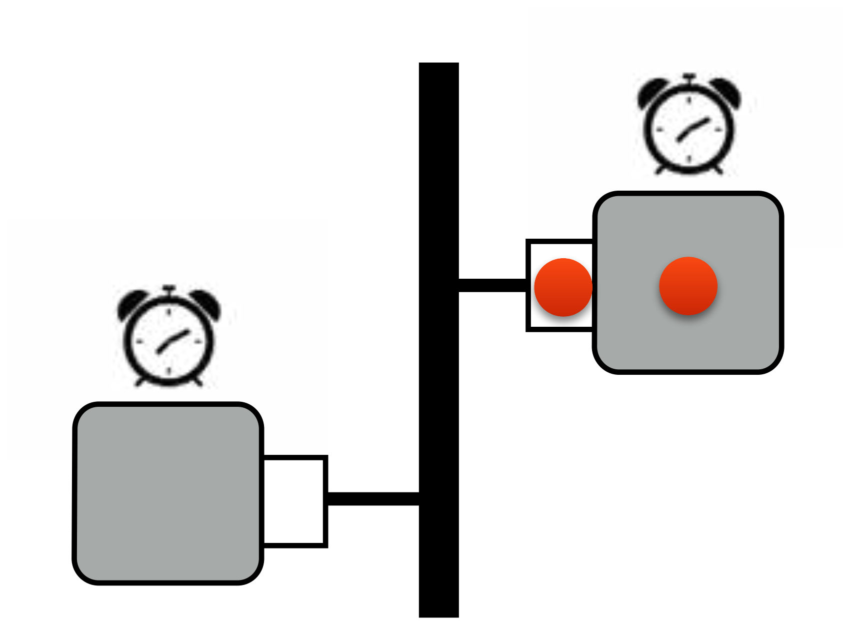
Oversampling

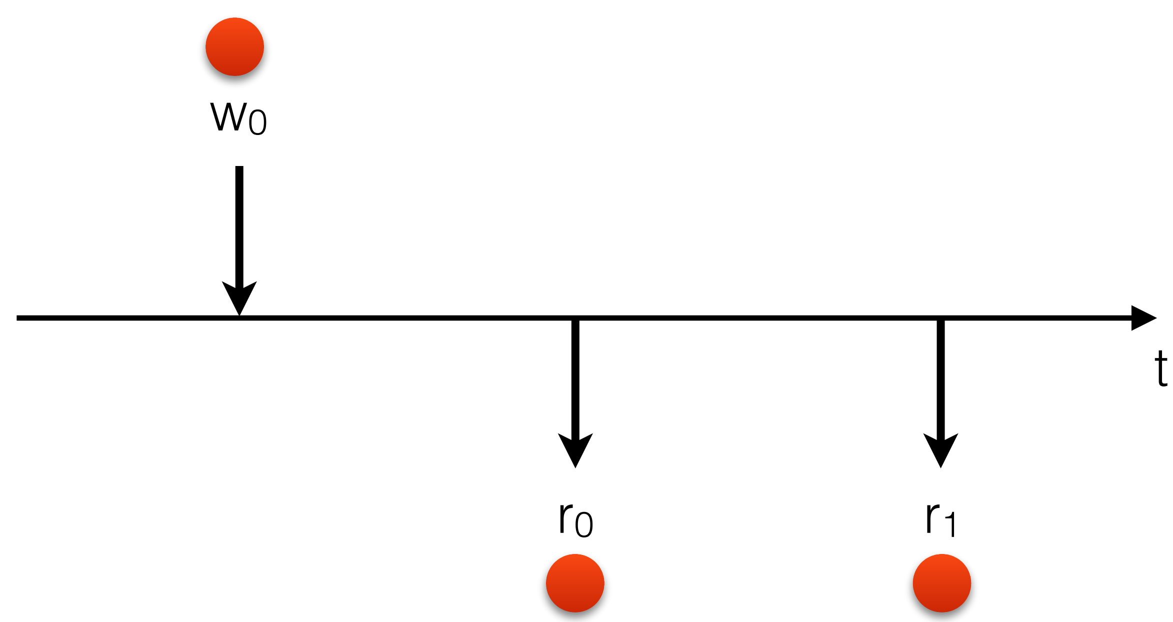




Two phenomena

Oversampling





read twice

Bounding Overwritten Values

Proposition: Given a writer and a reader, the maximum number of consecutive overwrites is

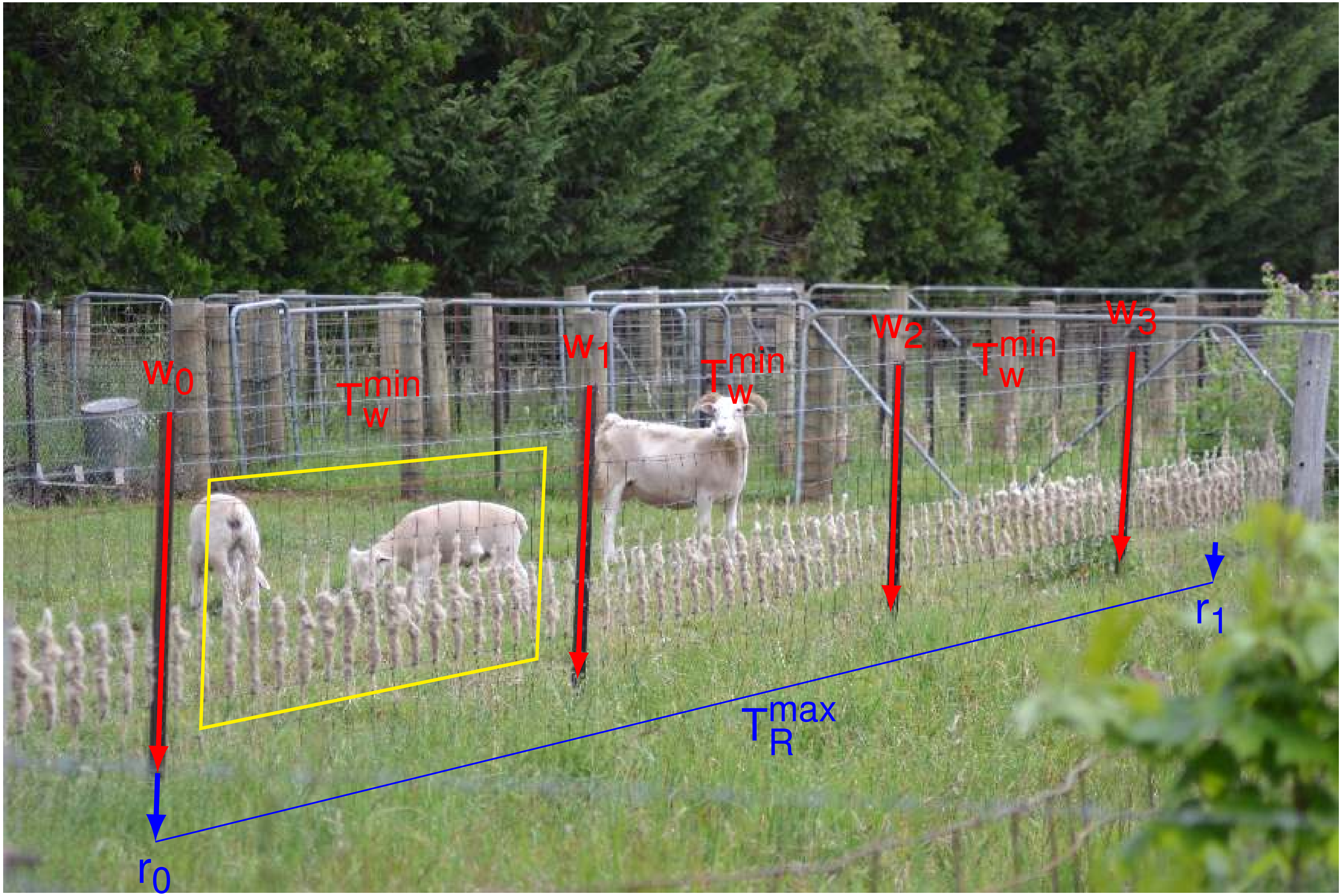
$$n_o = \left\lceil \frac{T_R^{max}}{T_W^{min}} \right\rceil - 1$$

Proof: a Fencepost Problem

Proof: a Fencepost Problem

Proof: a Fencepost Problem

Proof: a Fencepost Problem

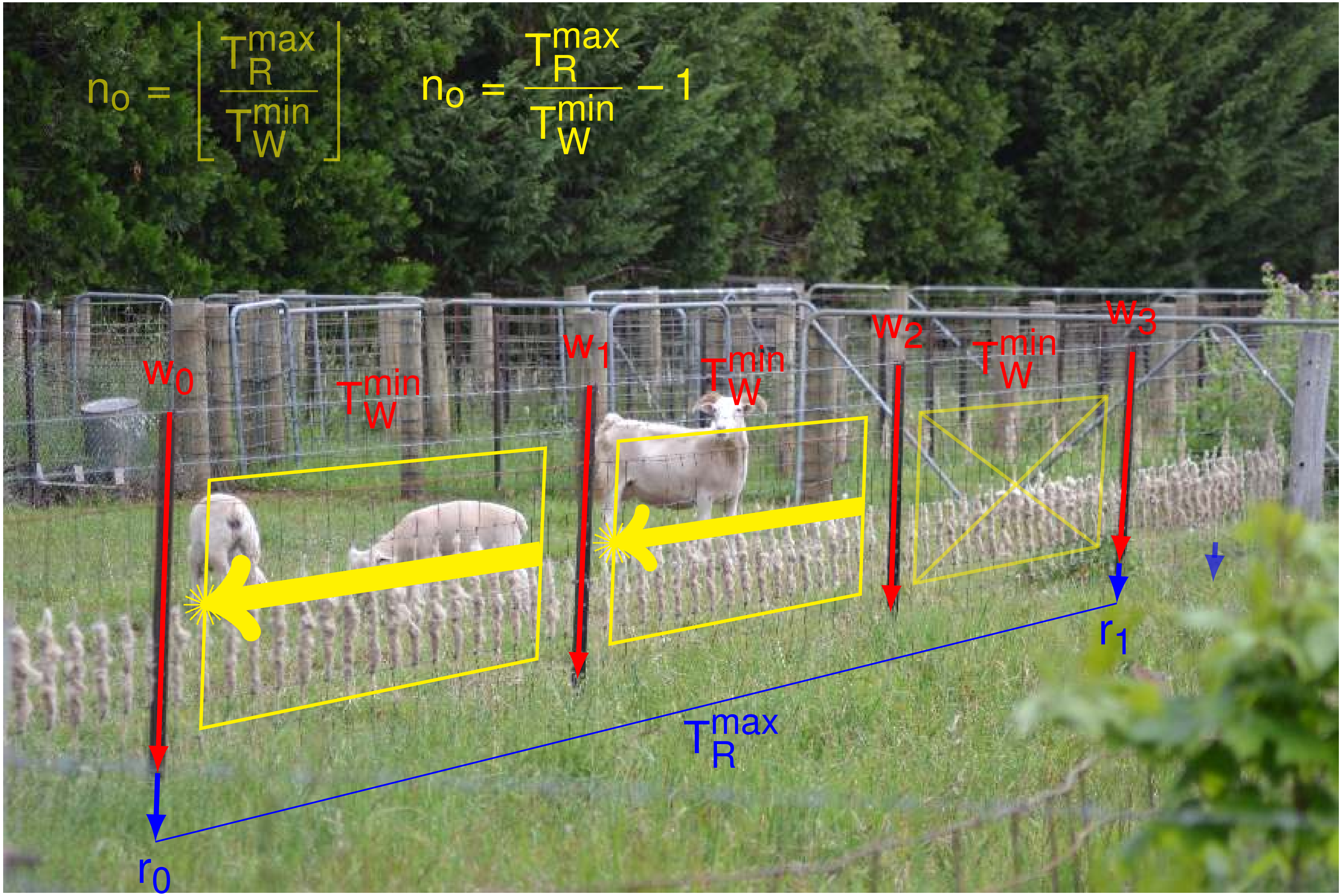


Proof: a Fencepost Problem

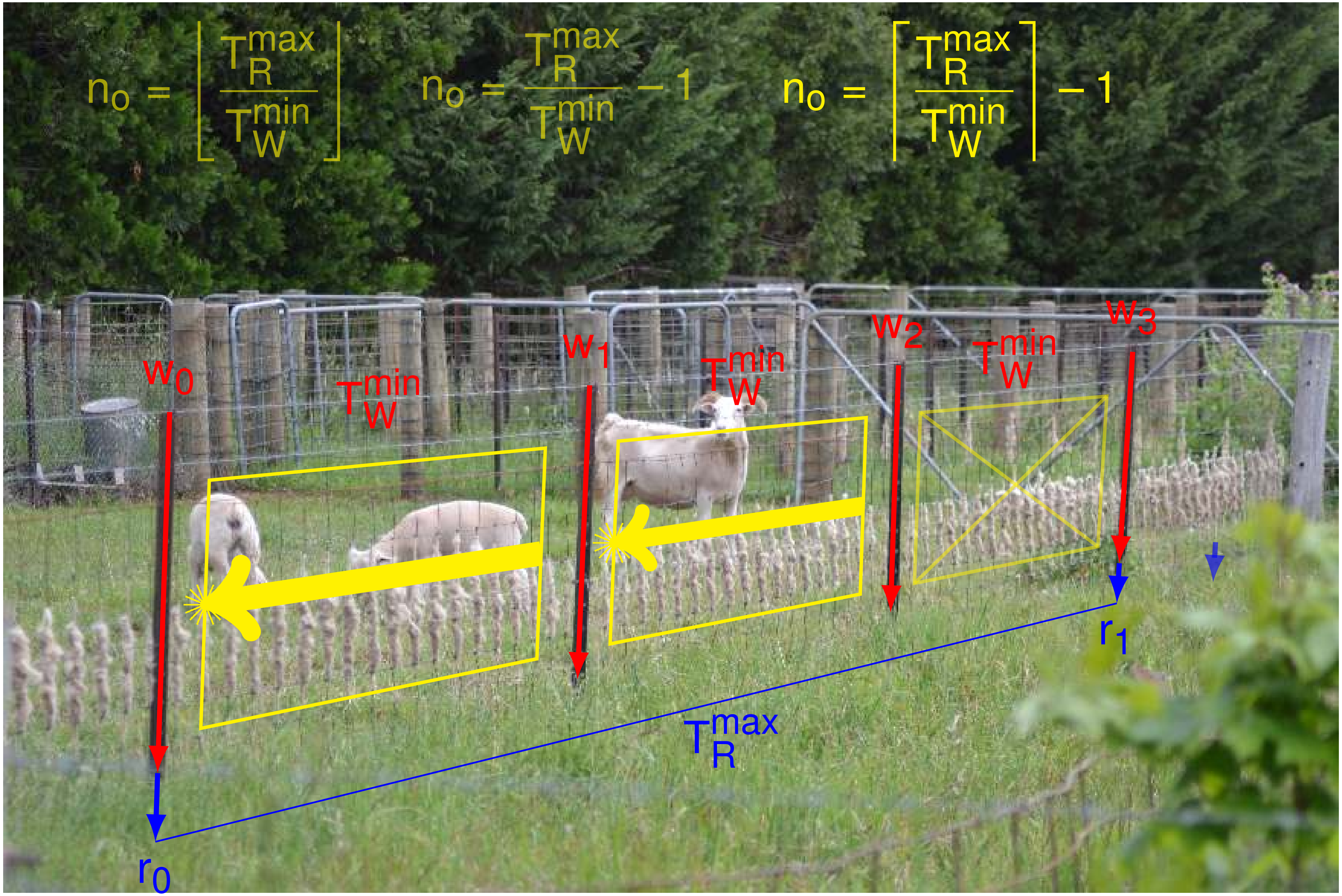
Proof: a Fencepost Problem

Proof: a Fencepost Problem

Proof: a Fencepost Problem



Proof: a Fencepost Problem



Bounding Overwritten Values

Corollary: Given a writer and a reader with the same nominal period and a small jitter of $0 < \varepsilon \leq \frac{1}{3}T^n$ it follows that $n_o = 1$

Proof.
$$\begin{aligned} n_o &= \left\lceil \frac{T^n(1+1/3)}{T^n(1-1/3)} \right\rceil - 1 \\ &= \left\lceil \frac{4/3}{2/3} \right\rceil - 1 \\ &= 1 \end{aligned}$$

Conclusion:

In a Quasi-periodic System where all process have the same period and a small jitter, **only one** value can ever be lost between two reads

Bounding Oversampled Values

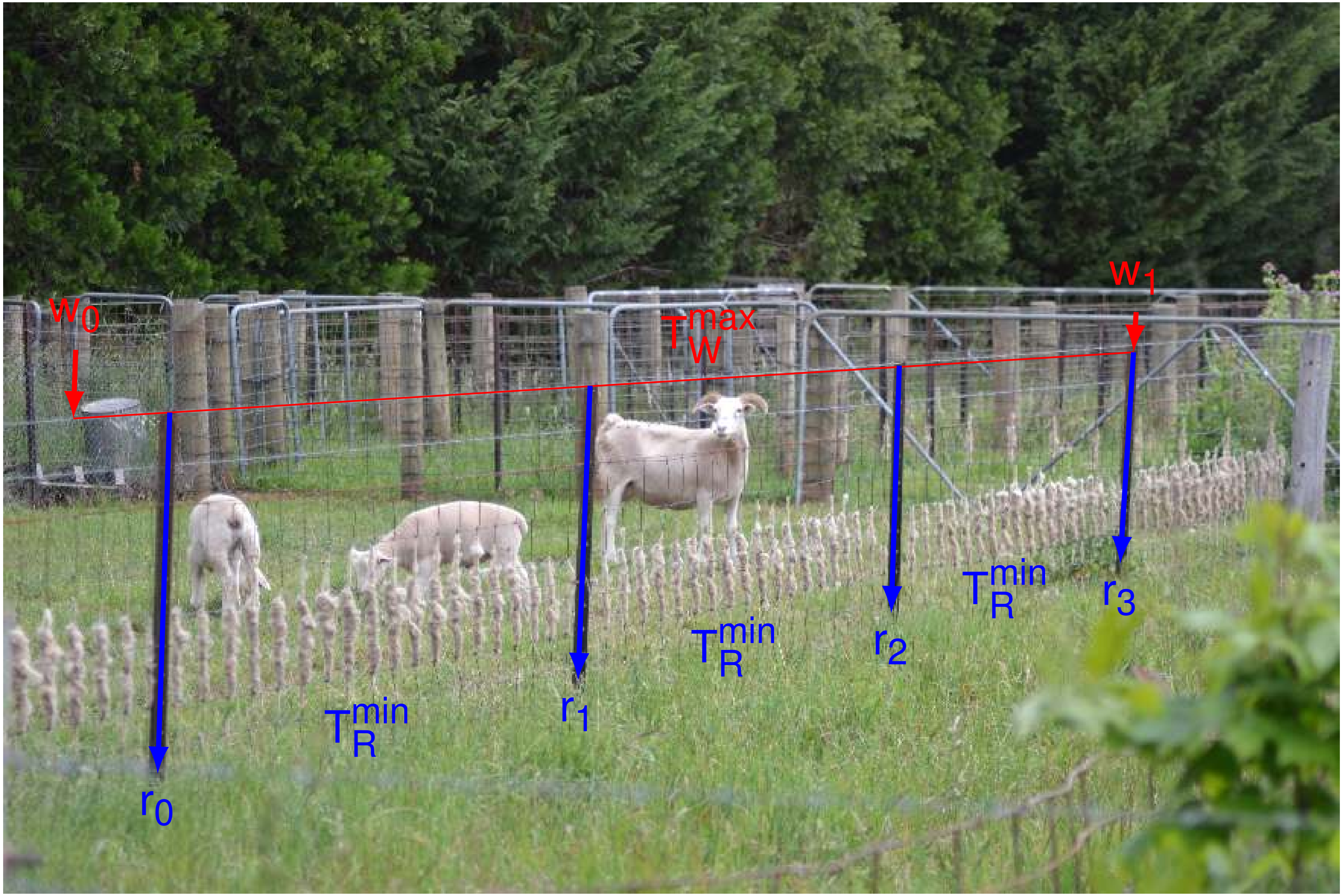
Proposition: Given a writer and a reader, the maximum number of consecutive oversampling is

$$n_s = \left\lceil \frac{T_W^{max}}{T_R^{min}} \right\rceil - 1$$

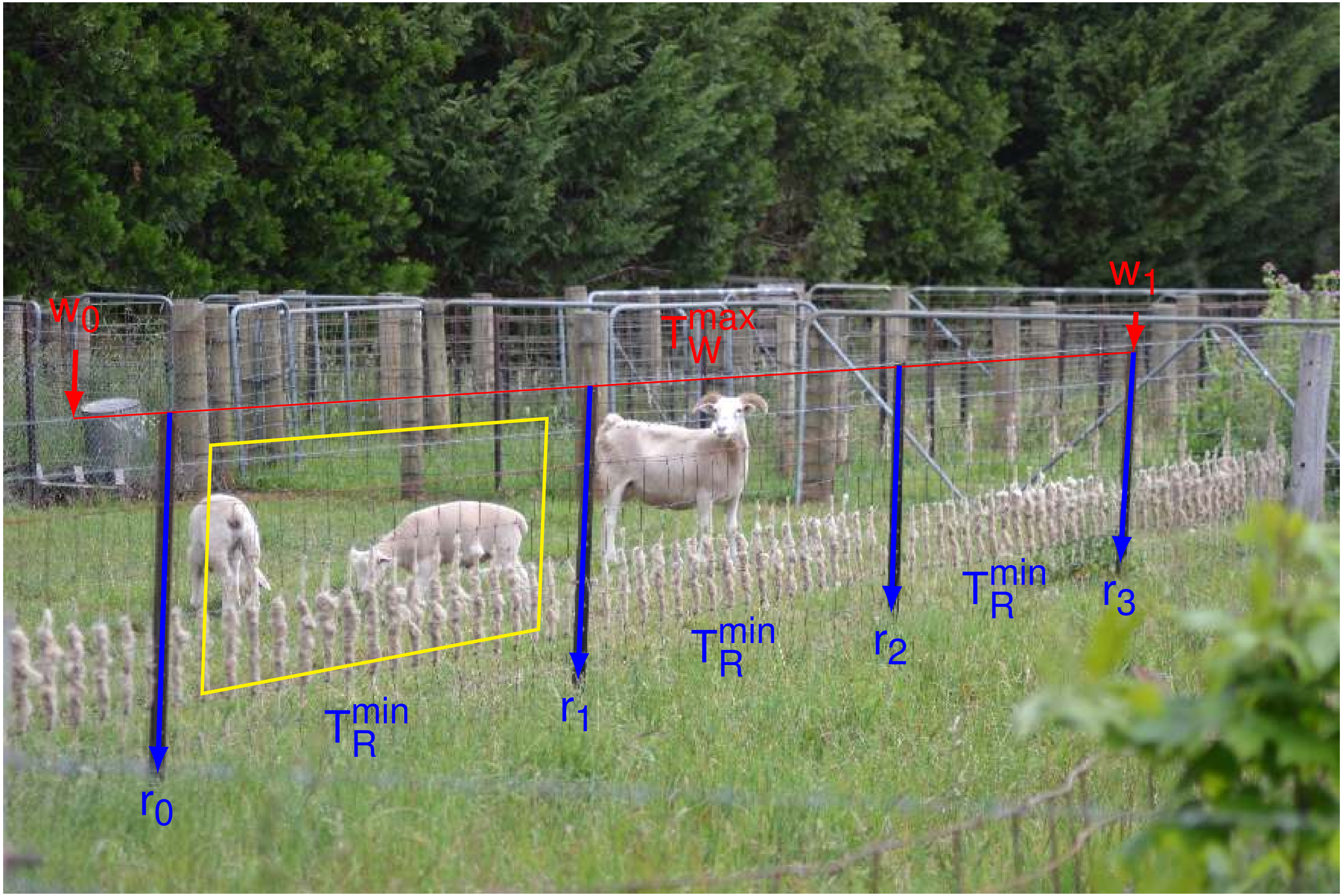
Proof: a Fencepost Problem

Proof: a Fencepost Problem

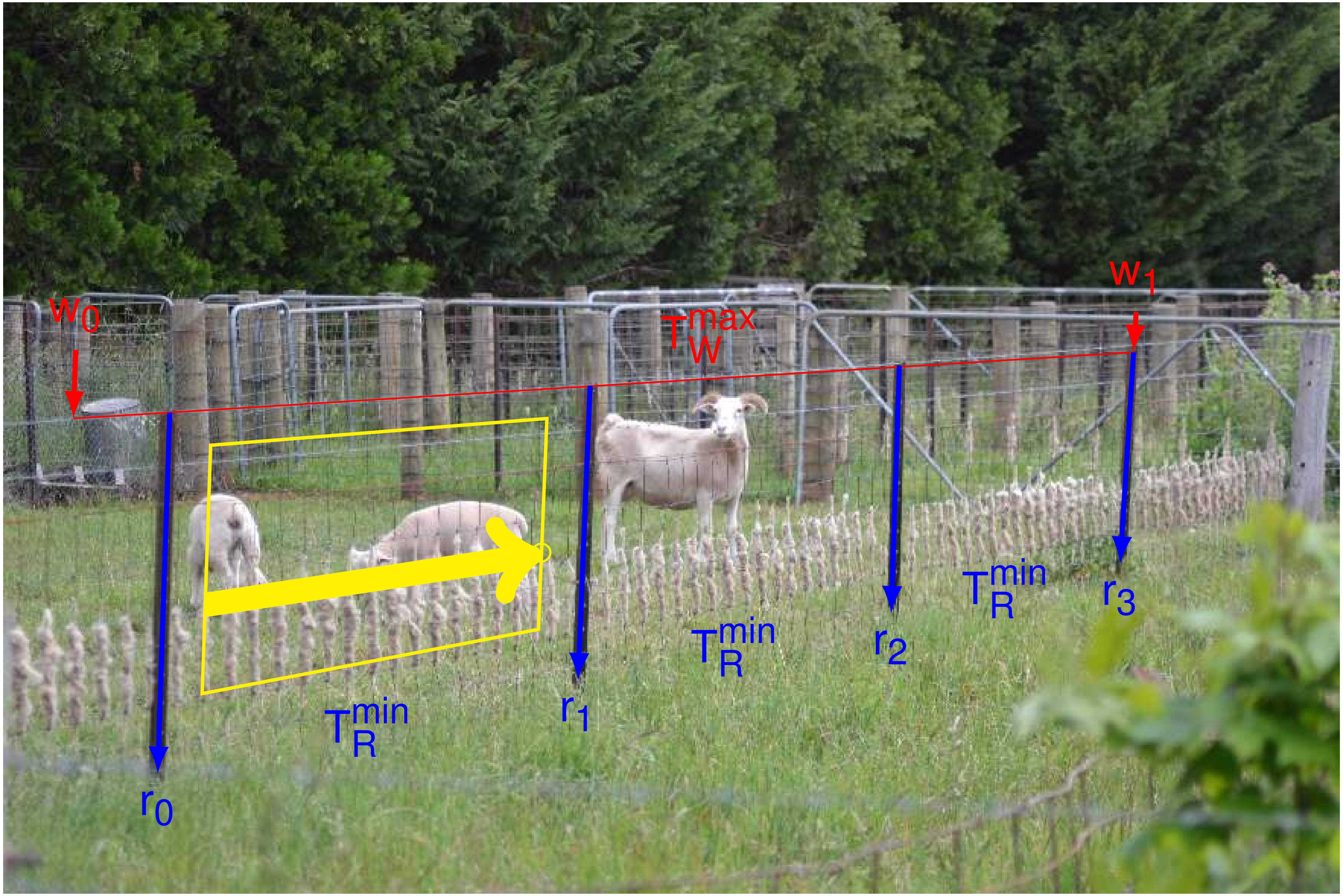
Proof: a Fencepost Problem



Proof: a Fencepost Problem



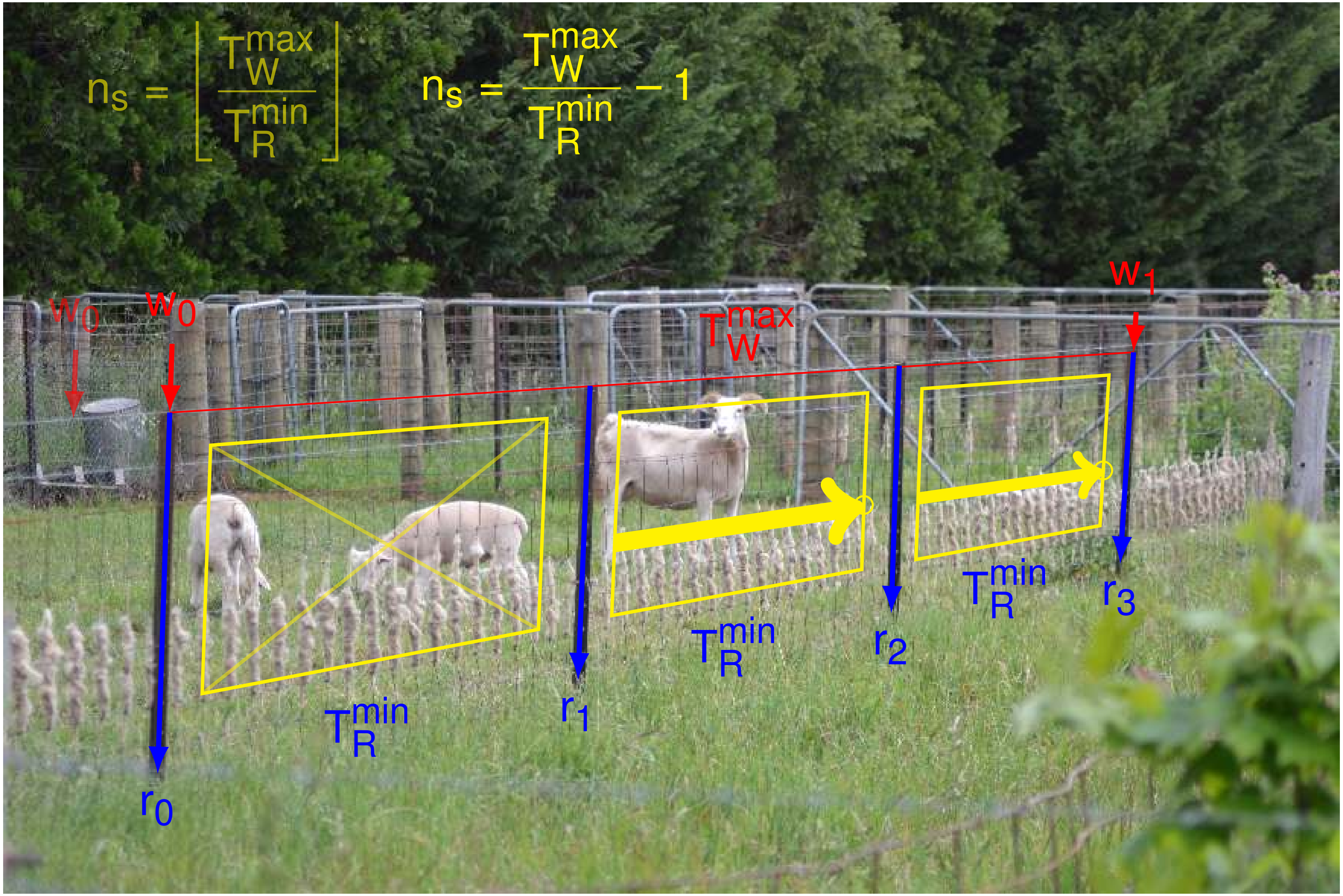
Proof: a Fencepost Problem



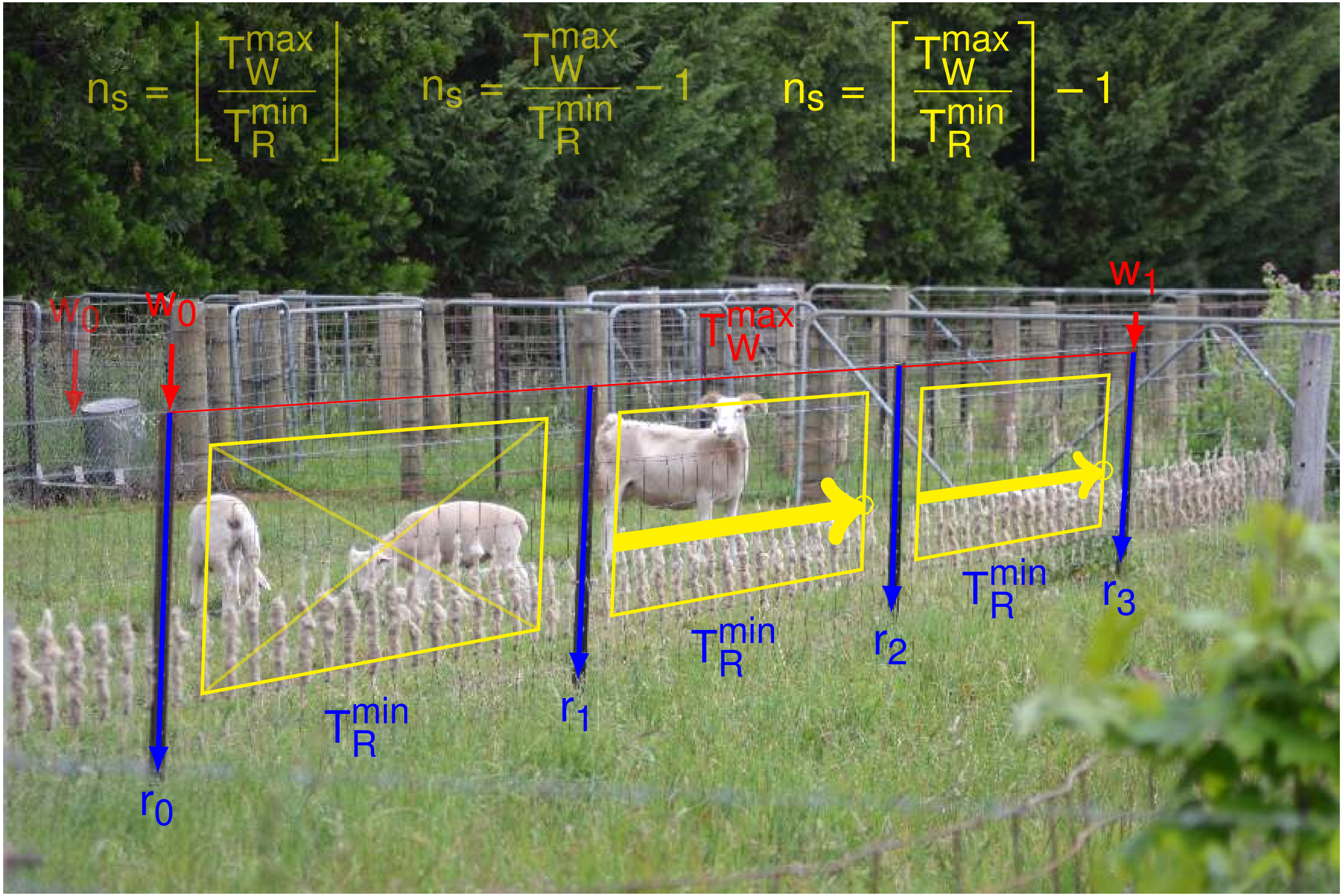
Proof: a Fencepost Problem

Proof: a Fencepost Problem

Proof: a Fencepost Problem



Proof: a Fencepost Problem

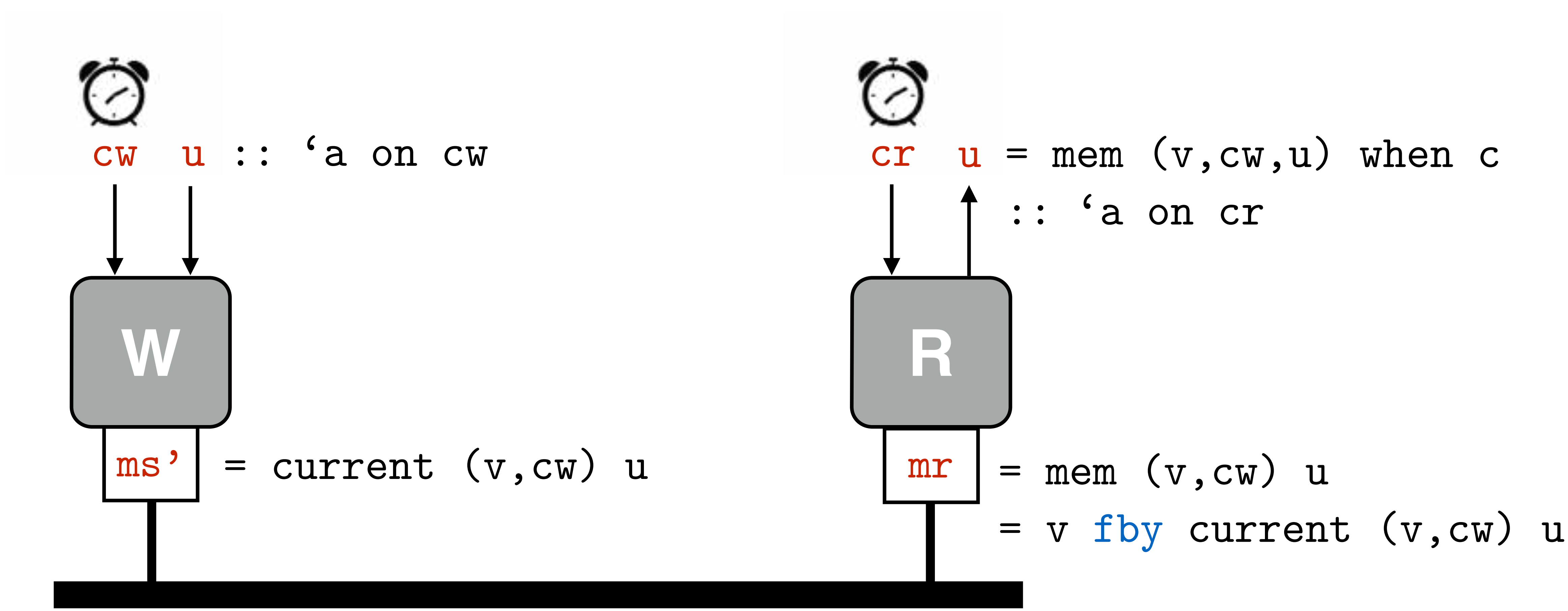


Discrete Abstraction

Discrete Model

[From the Cooking Book]

- Model in Lustre or Lucid Synchrone
- Clocks represent activations of a node
- Transmission delay: one tick of the base clock (`'a`)



Clock Condition

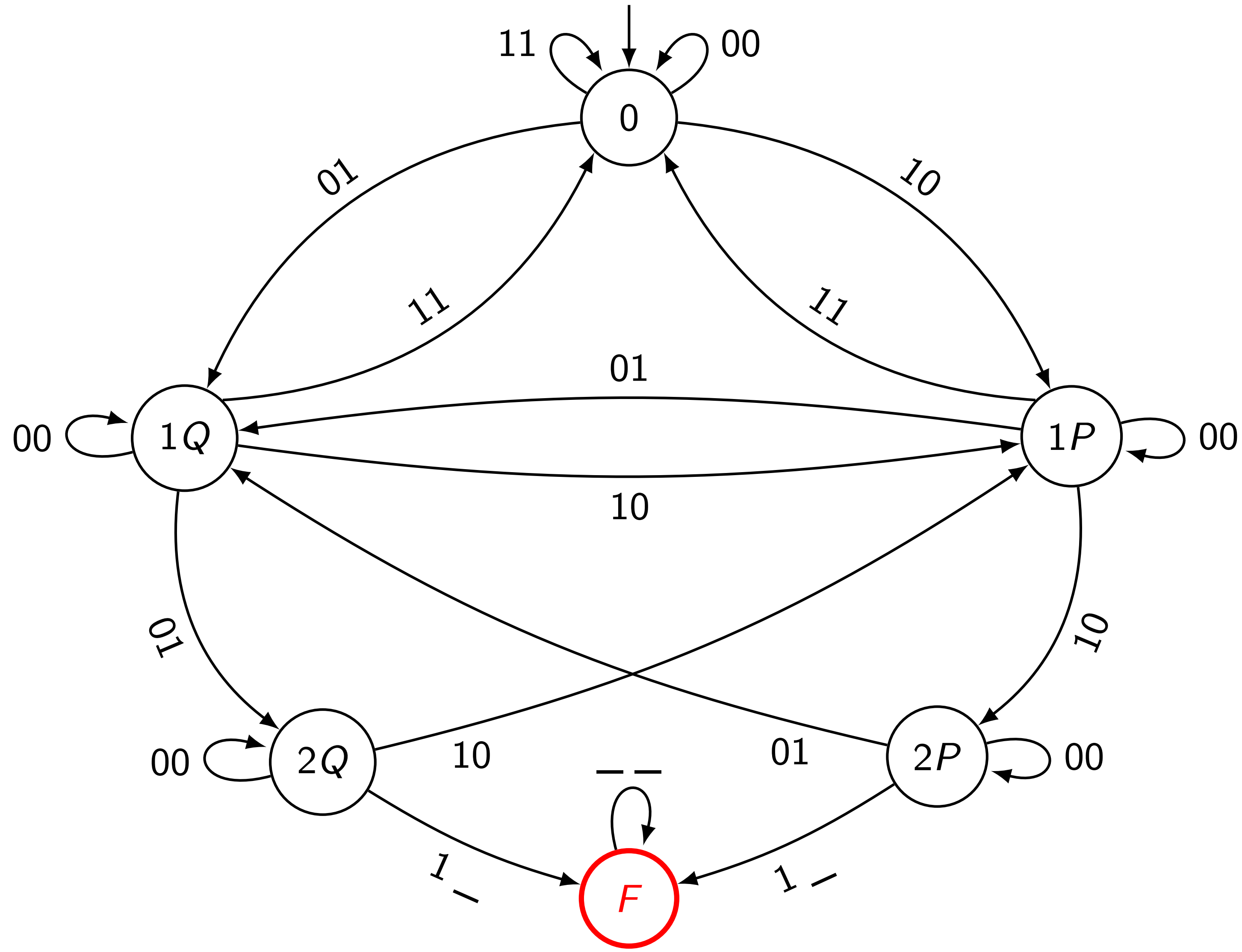
> *Neither of the clocks can take the value 1 more than twice between two successive 1 of the other*
>
> P. Caspi

Idea: forbid the following sequence and the symmetrical one

$$\begin{array}{l} c_1 \\ c_2 \end{array} \quad \begin{bmatrix} 1 \\ \text{\_} \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix}^* \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix}^* \cdot \begin{bmatrix} 1 \\ \text{\_} \end{bmatrix} \qquad \begin{array}{l} n_o = 1 \\ n_s = 1 \end{array}$$

Clock Condition

Directly leads to a finite automaton



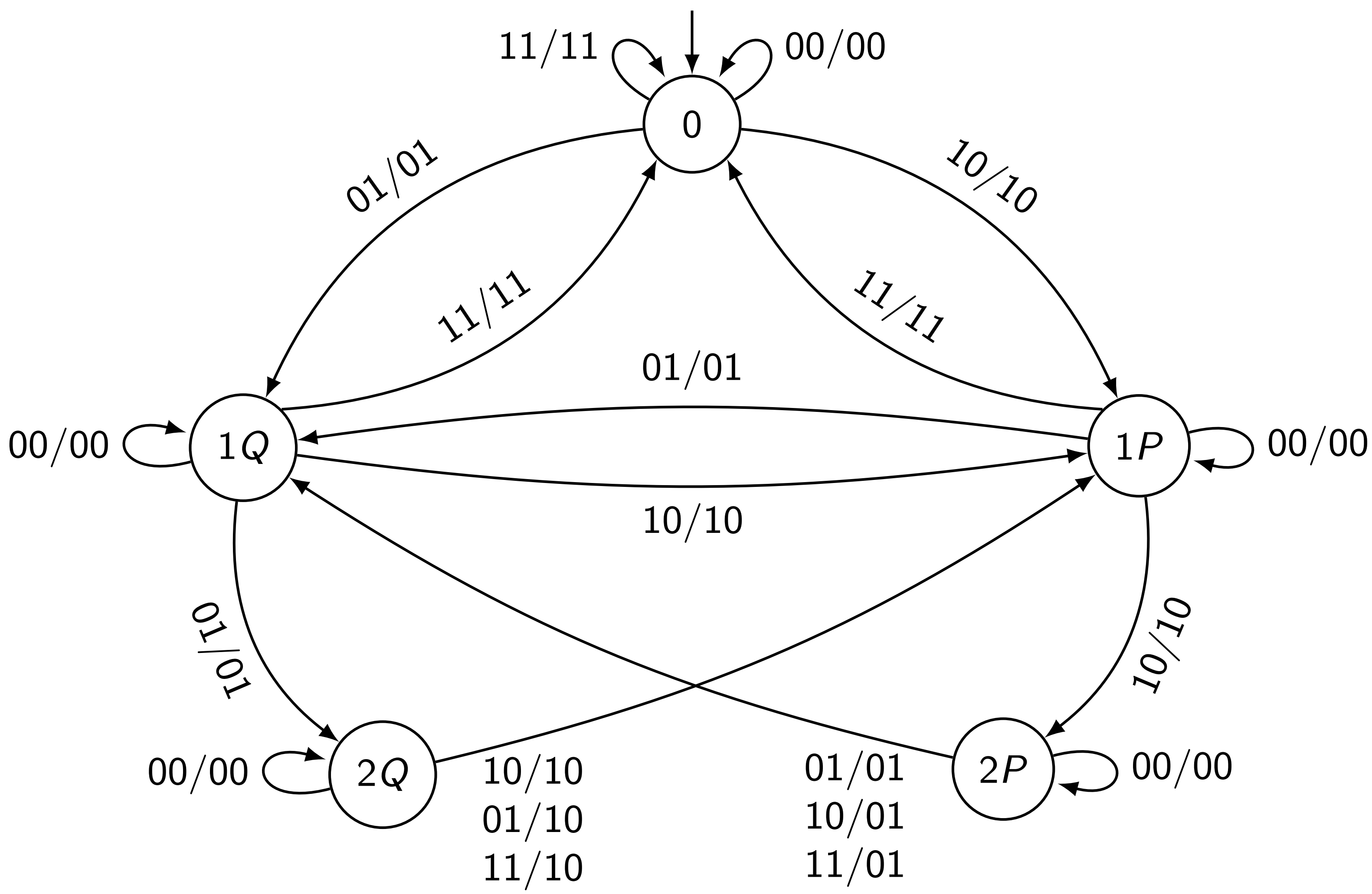
Transitions are labelled by (c_w,c_r)

If state F is reached, the trace is not quasi-synchronous

Clock Condition

Or a scheduler for simulation

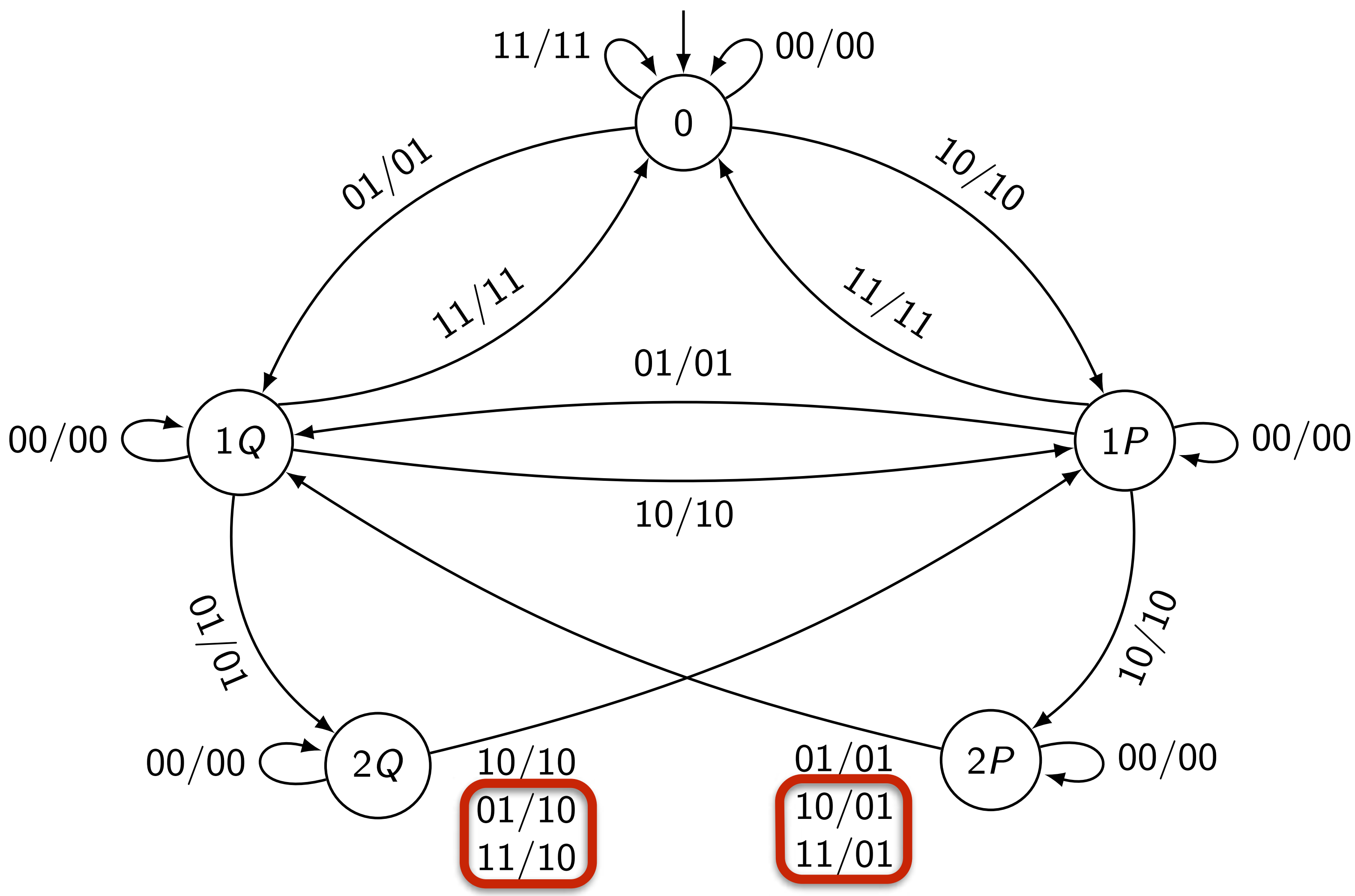
[Halbwachs Mandel 2006]



Clock Condition

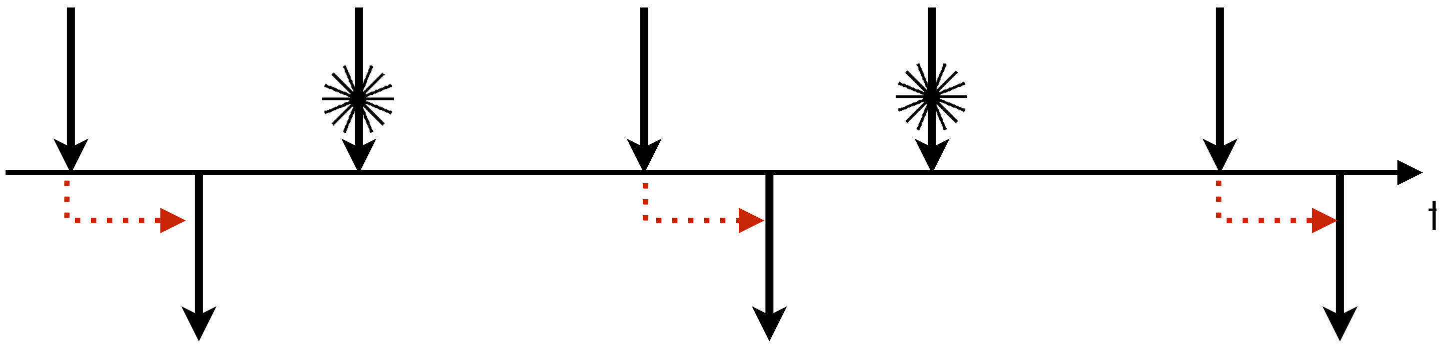
Or a scheduler for simulation

[Halbwachs Mandel 2006]

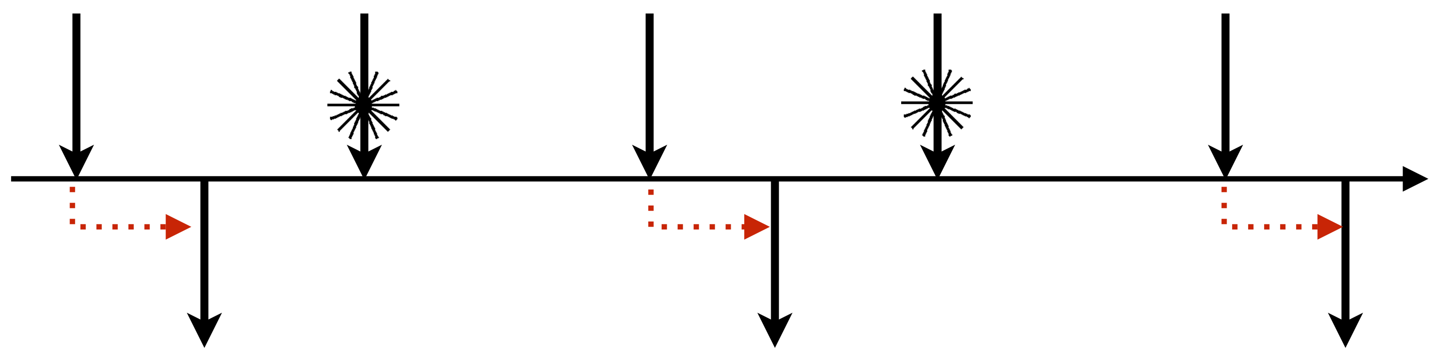


Non quasi-synchronous traces are corrected

Too Restrictive?

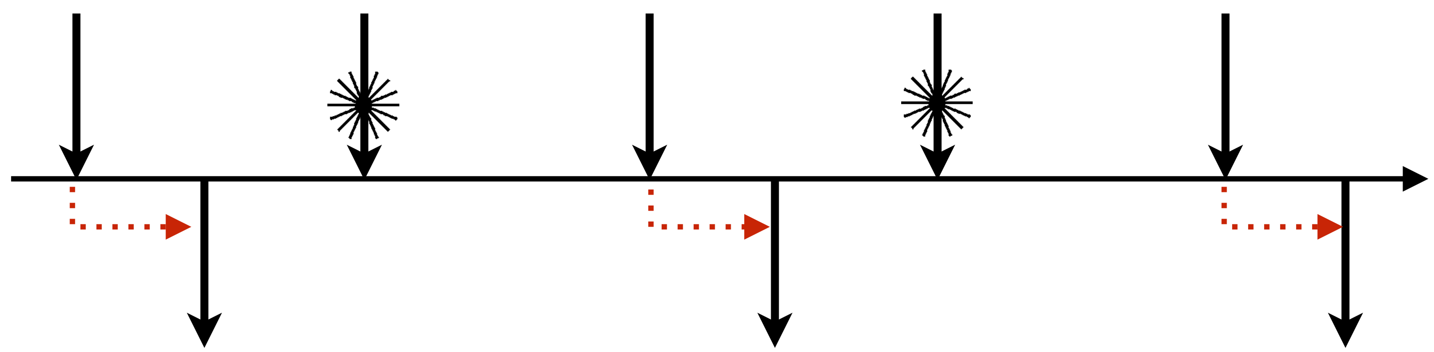


Too Restrictive?



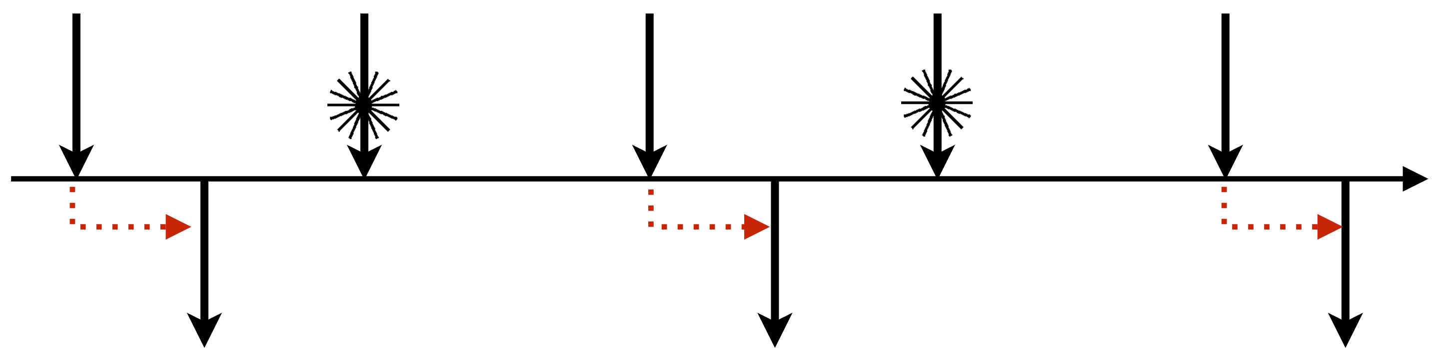
$$\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}$$

Too Restrictive?

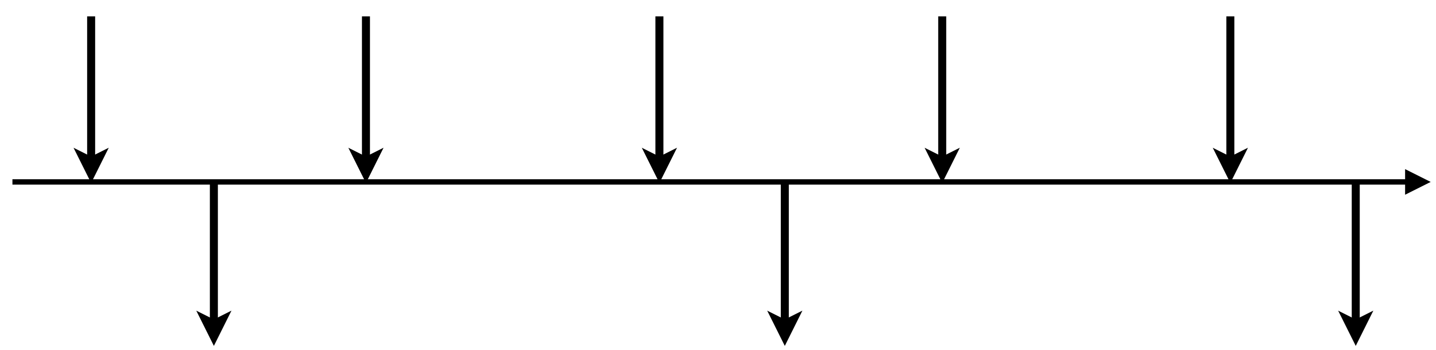


$$\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}$$

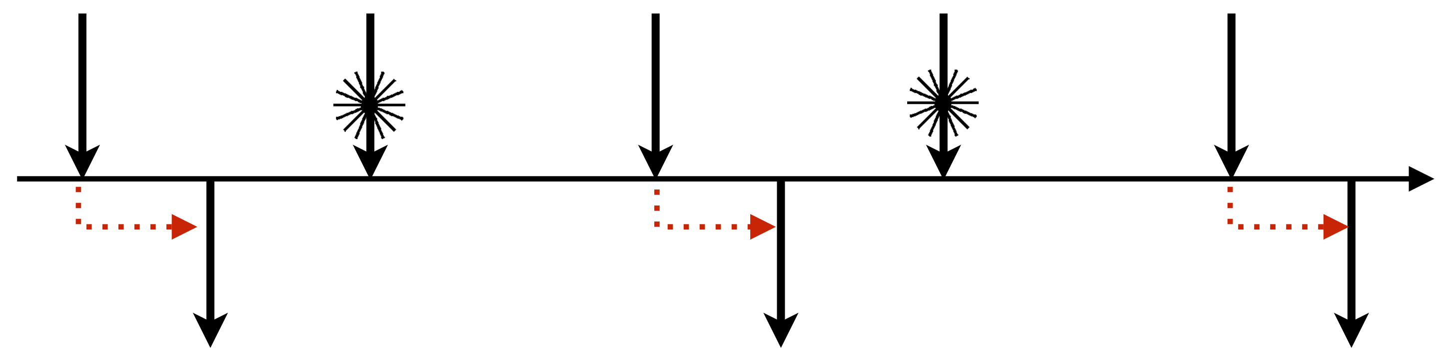
Too Restrictive?



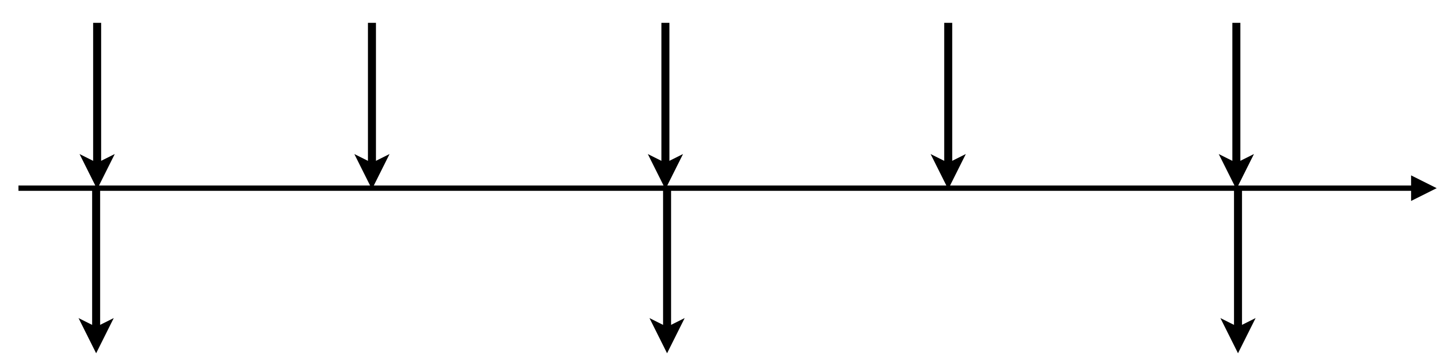
$$\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}$$



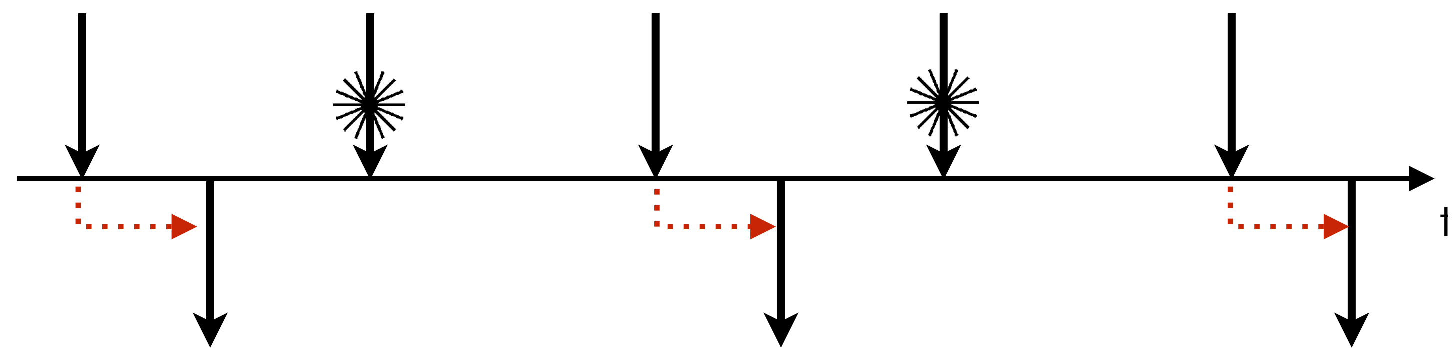
Too Restrictive?



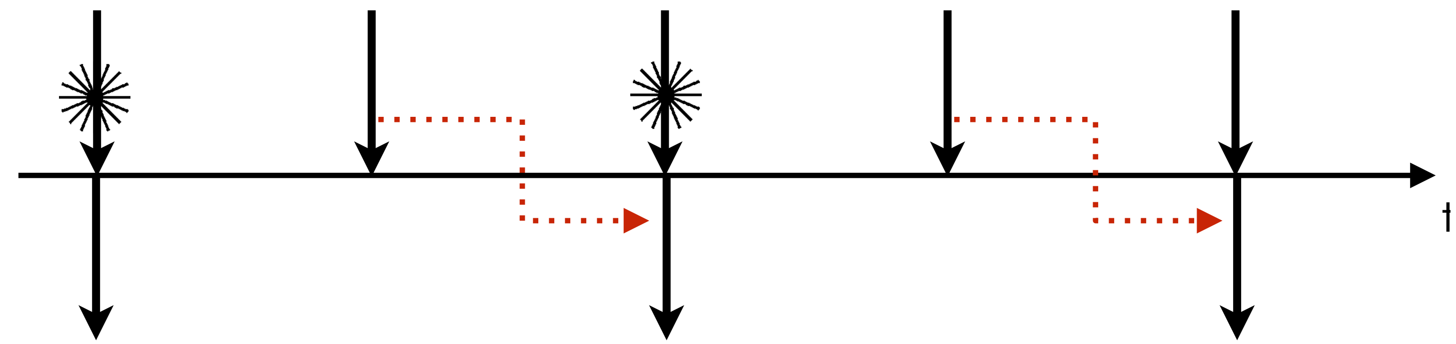
$$\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}$$



Too Restrictive?

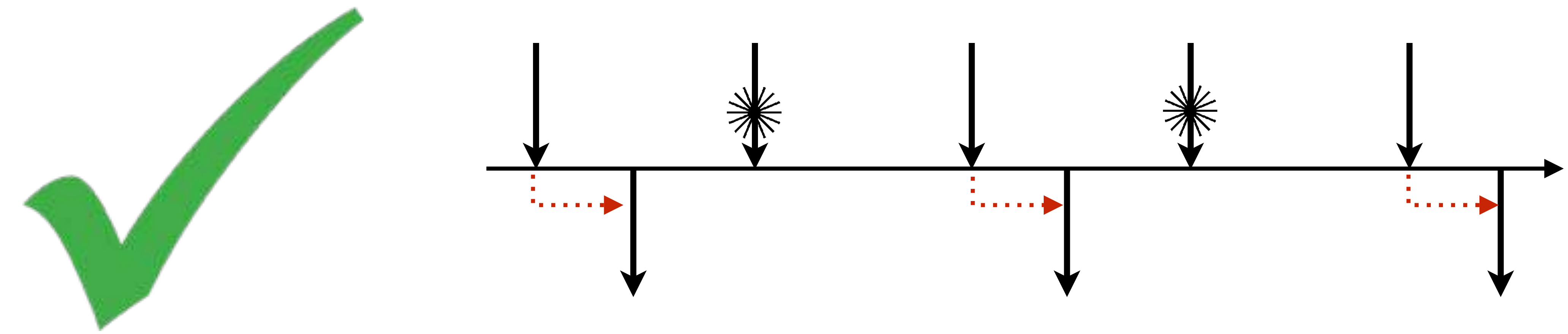


$$\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}$$

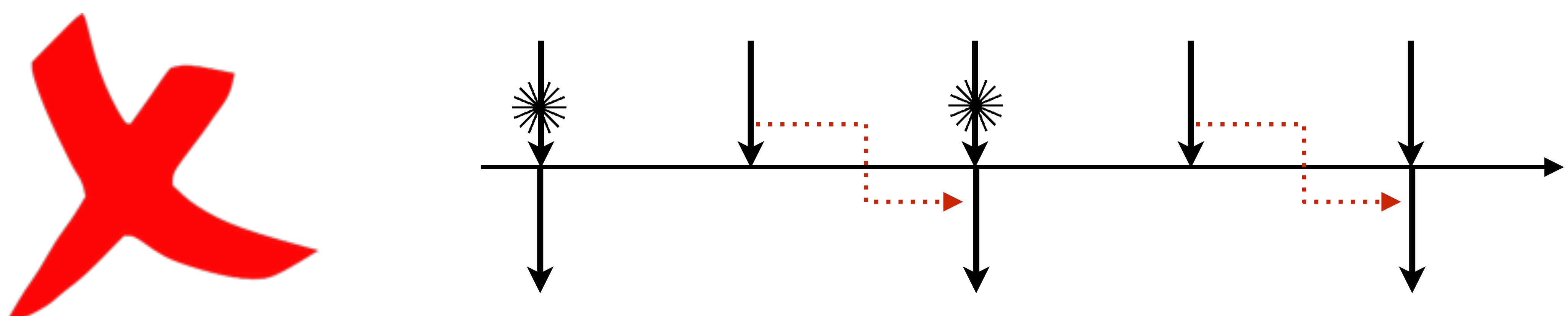


$$\begin{bmatrix}1\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}1\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}1\\1\end{bmatrix}$$

Too Restrictive?



$$\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix}$$



$$\boxed{\begin{bmatrix}1\\1\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}1\\1\end{bmatrix}}\cdot\begin{bmatrix}1\\0\end{bmatrix}\cdot\begin{bmatrix}1\\1\end{bmatrix}$$

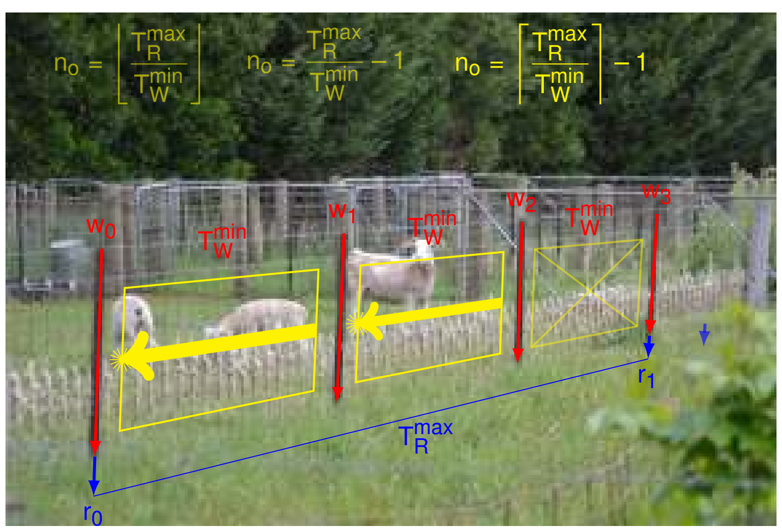
Generalization

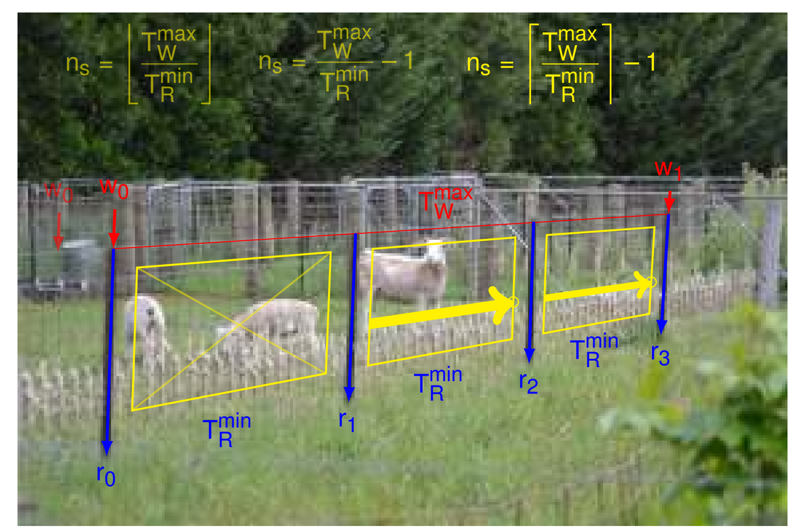
Idea: decoupling overwriting and oversampling

Overwriting | **Oversampling**

$$\begin{array}{c} C_W \\ C_r \end{array} \quad \begin{bmatrix} 1 \\ - \end{bmatrix} \cdot \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}^* \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right)^{n_o+1}$$

$$\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix}^* \right)^{n_s+1} \cdot \begin{bmatrix} - \\ 1 \end{bmatrix}$$





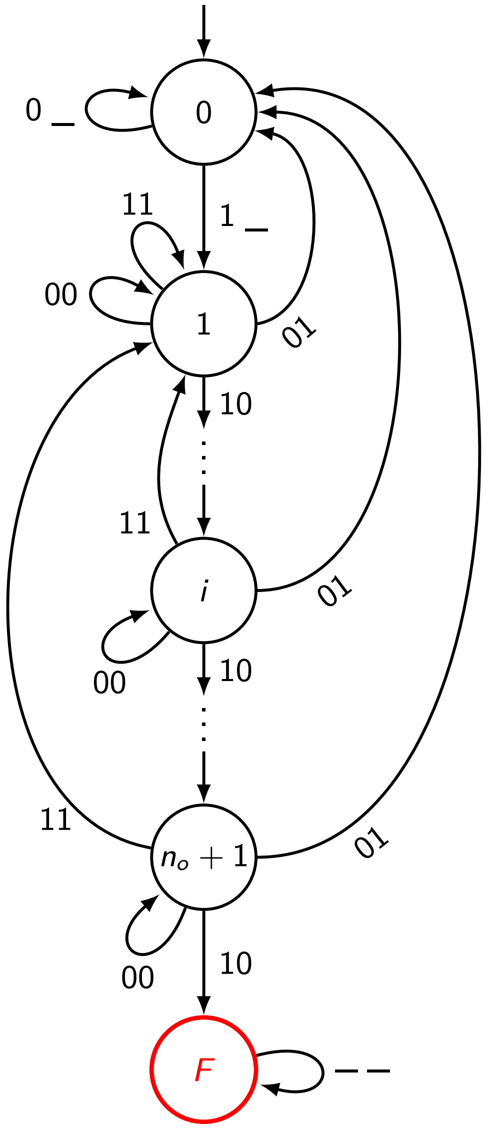
Counters of consecutive activations

Symmetrical formula

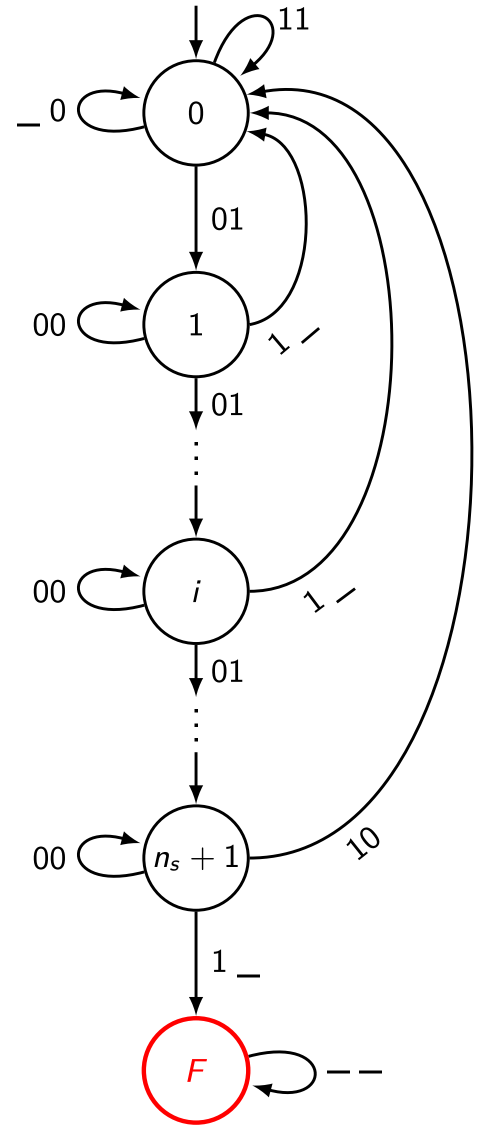
Generalization

Idea: decoupling overwriting and oversampling

Overwriting



Oversampling



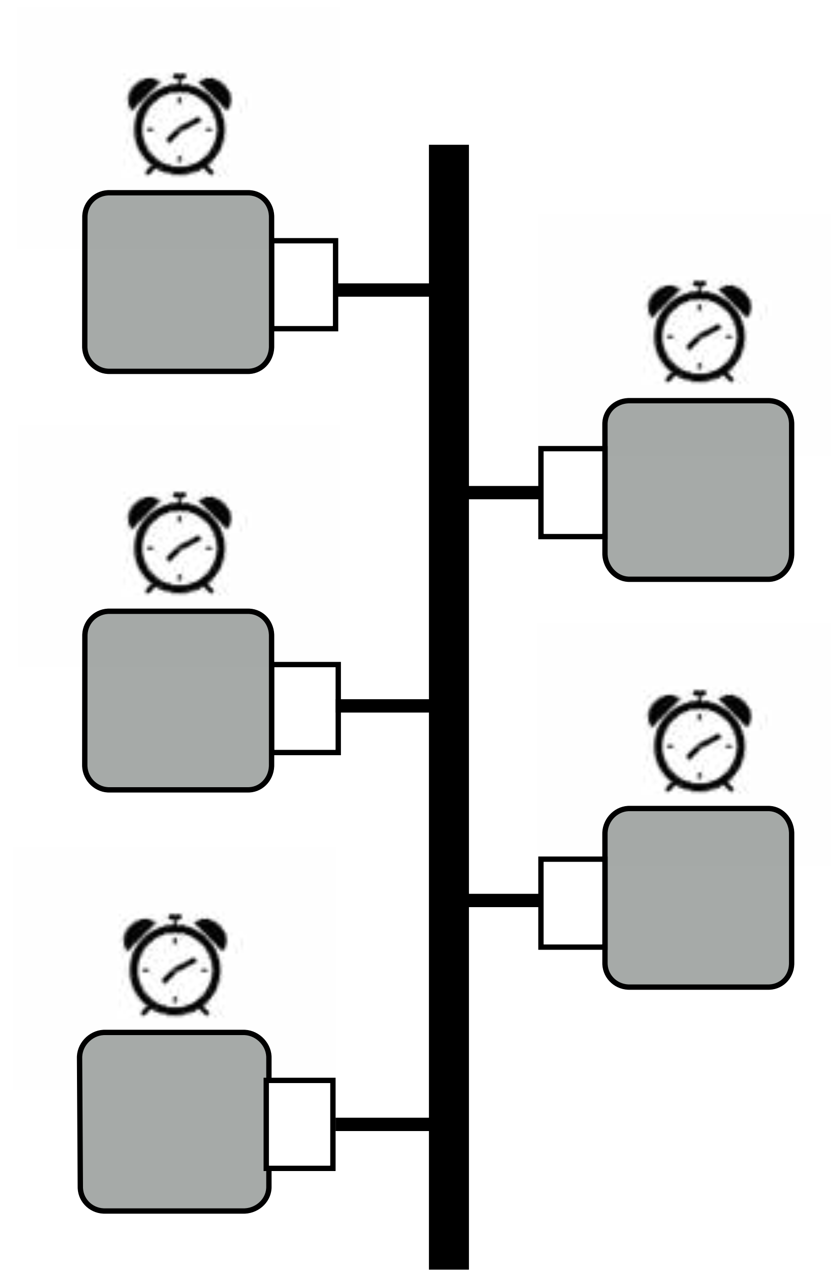
Loosely Time-Triggered Architectures

[Tripakis et al. 2008]

[Caspi, Benveniste 2008]

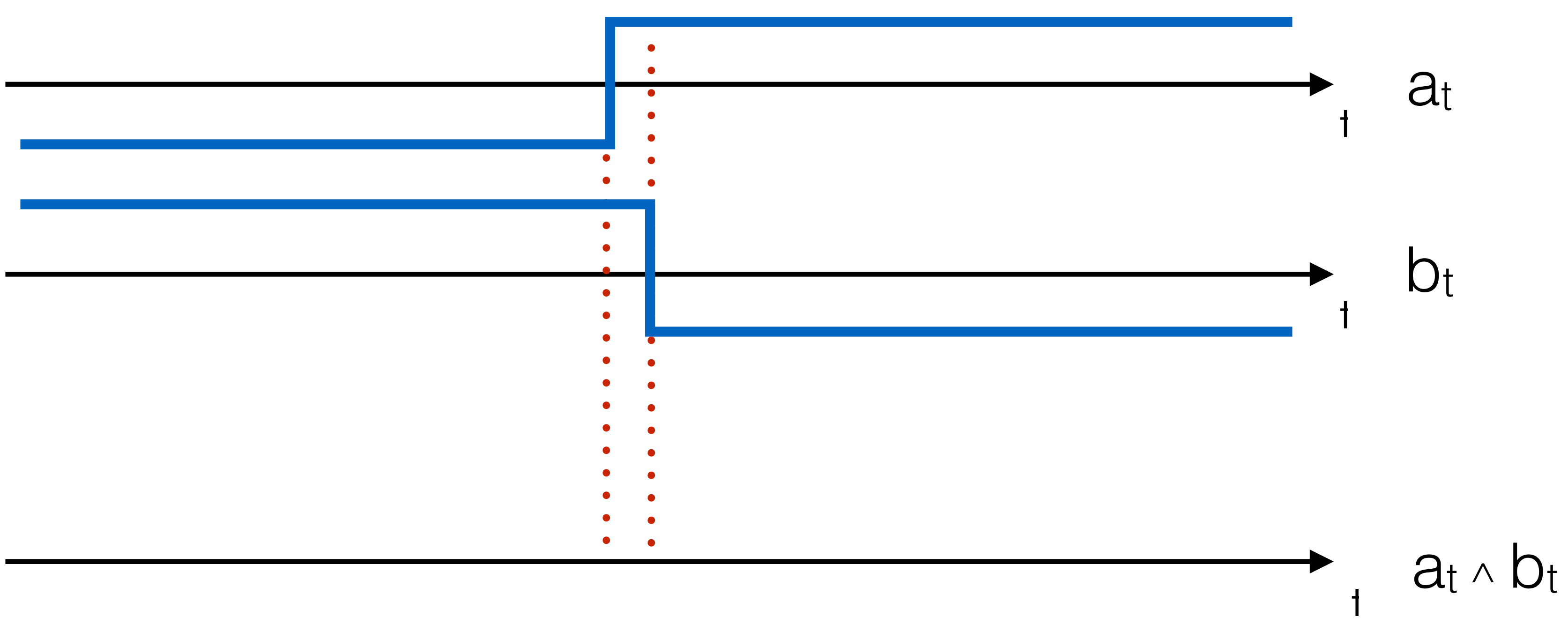
What are LTTA?

- **Base**: A quasi-periodic architecture
- **Goal:** Safely deploy a synchronous model
- **Idea**: Add a layer of middleware



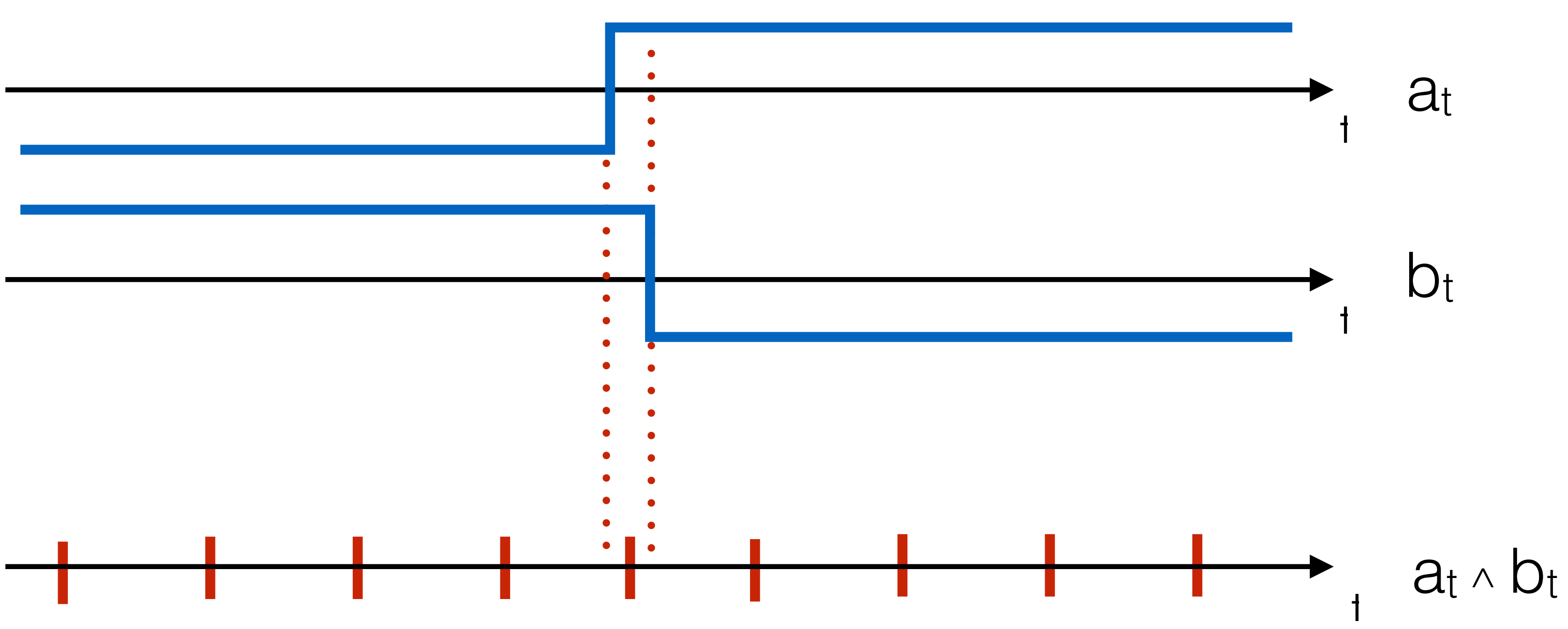
Problems

- **Overwriting**: Lost of values
- **Oversampling:** Duplication of values
- **Combination of signals**



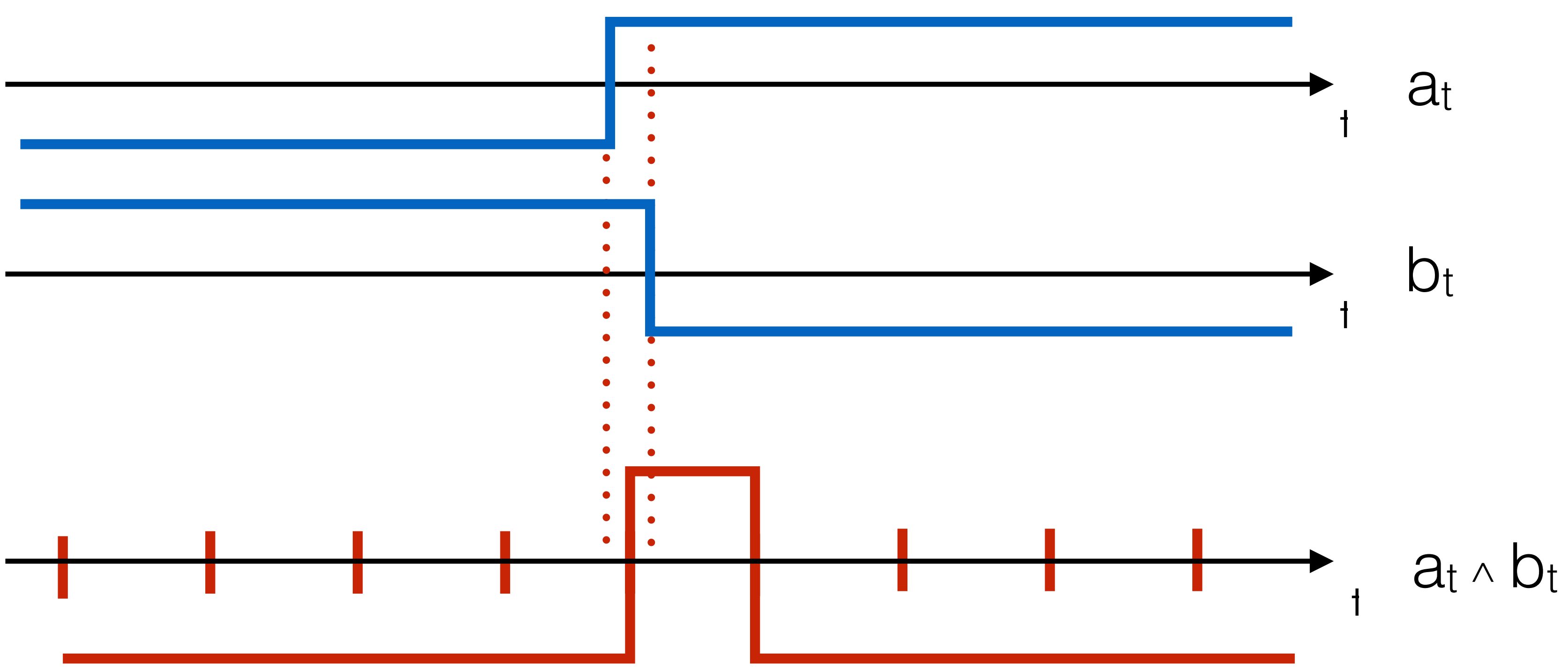
Problems

- **Overwriting**: Lost of values
- **Oversampling:** Duplication of values
- **Combination of signals**



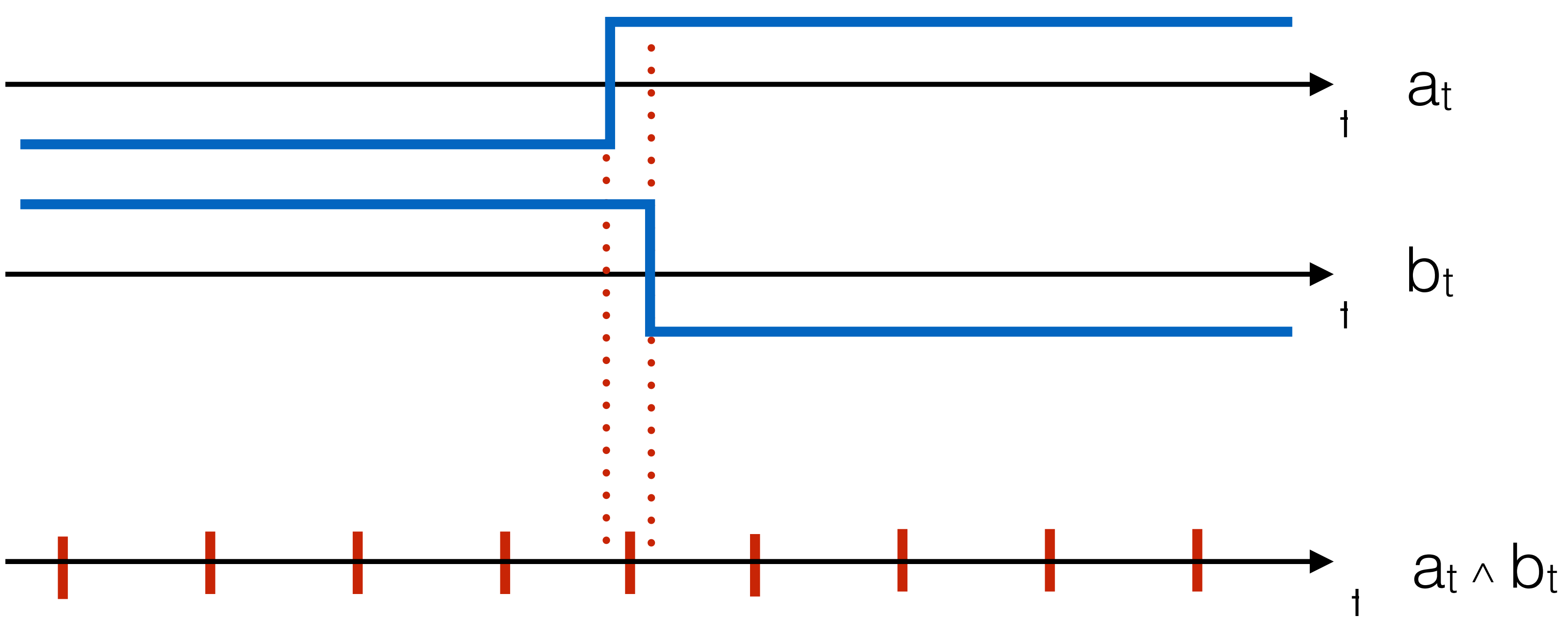
Problems

- **Overwriting**: Lost of values
- **Oversampling:** Duplication of values
- **Combination of signals**



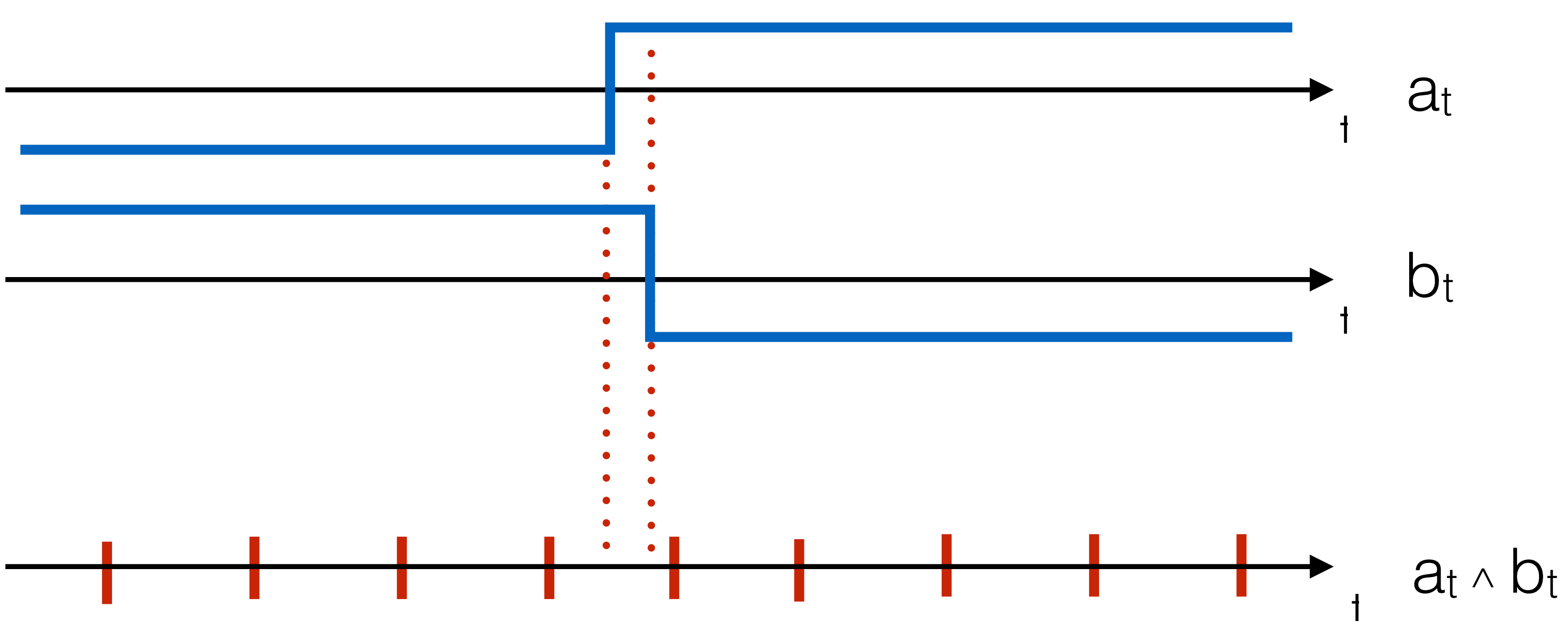
Problems

- **Overwriting**: Lost of values
- **Oversampling:** Duplication of values
- **Combination of signals**



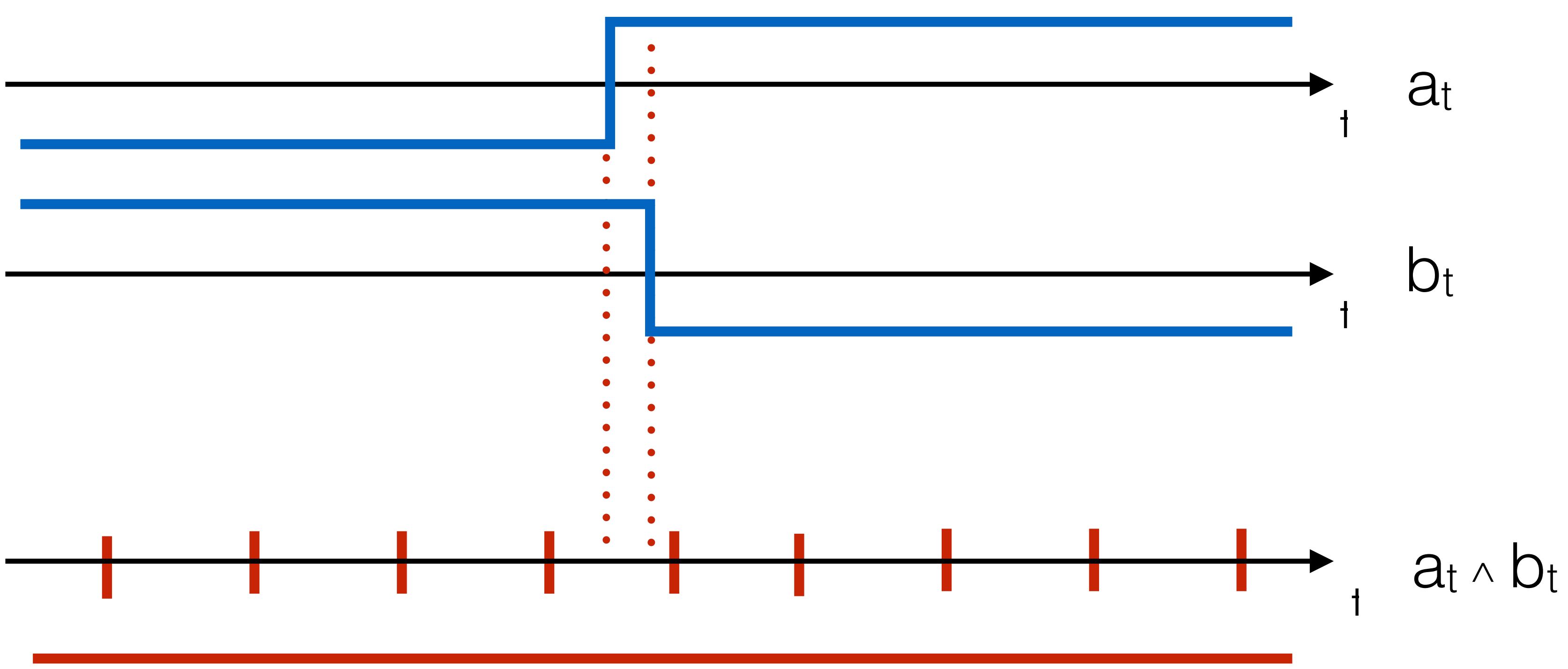
Problems

- **Overwriting**: Lost of values
- **Oversampling:** Duplication of values
- **Combination of signals**



Problems

- **Overwriting**: Lost of values
- **Oversampling:** Duplication of values
- **Combination of signals**

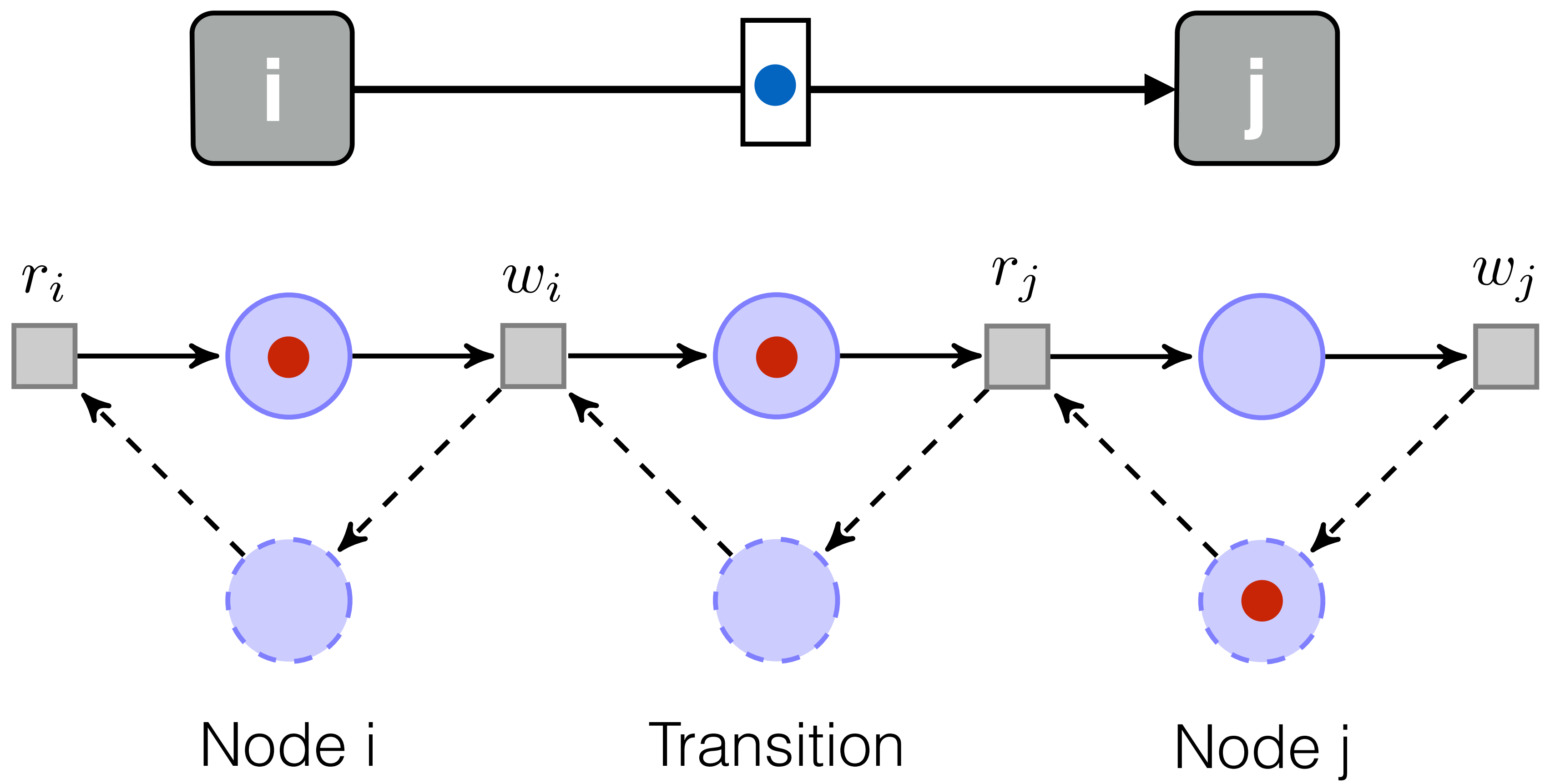


Two solutions

- **Back-Pressure LTTA** [Tripakis et al. 2008]
- **Time-Based LTTA** [Caspi, Benveniste 2008]

Back-Pressure Kahn Network

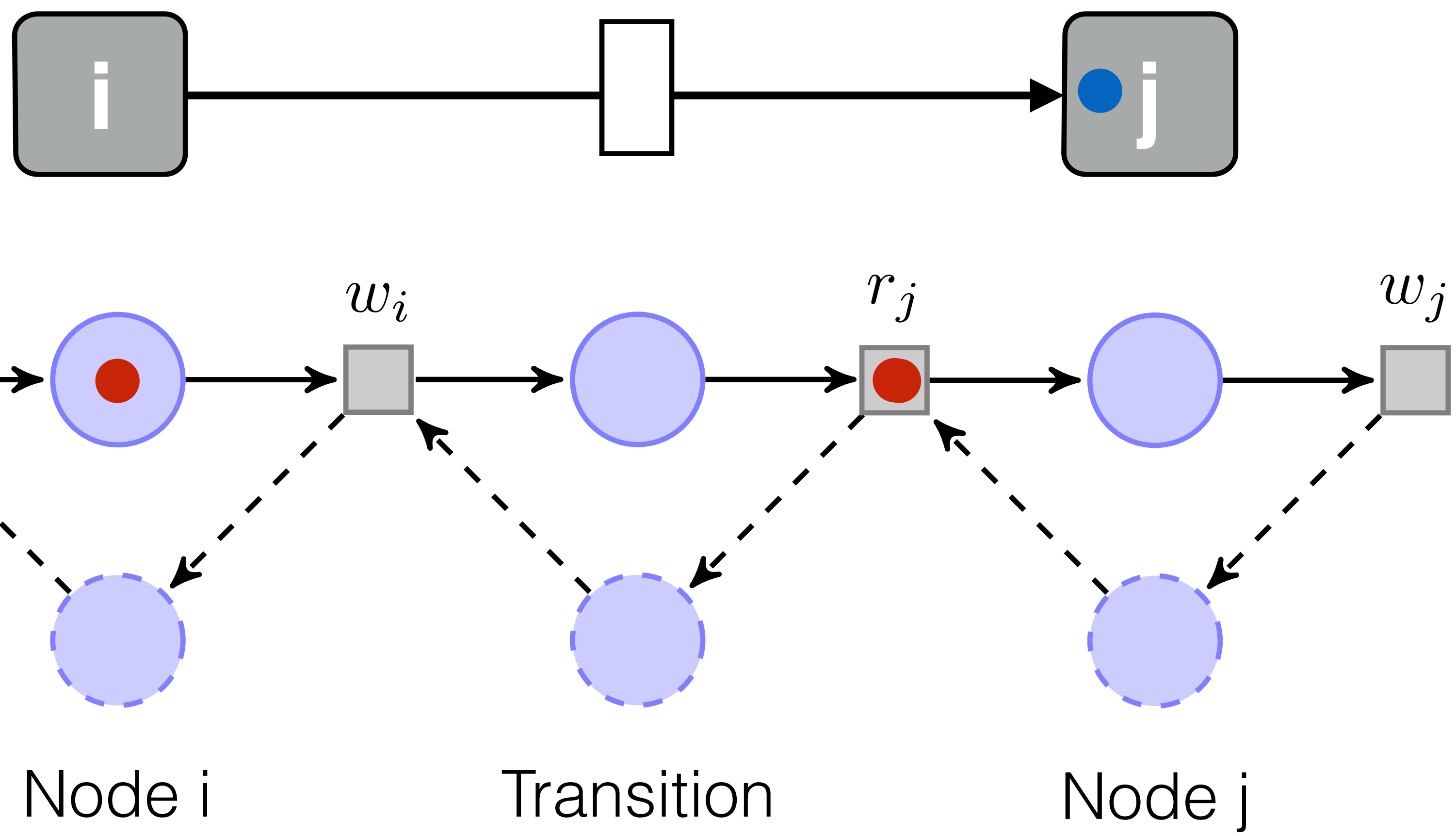
buffer of size 1



- Reading from a buffer is acknowledged to the writer
- Nodes alternate between reads and writes

Back-Pressure Kahn Network

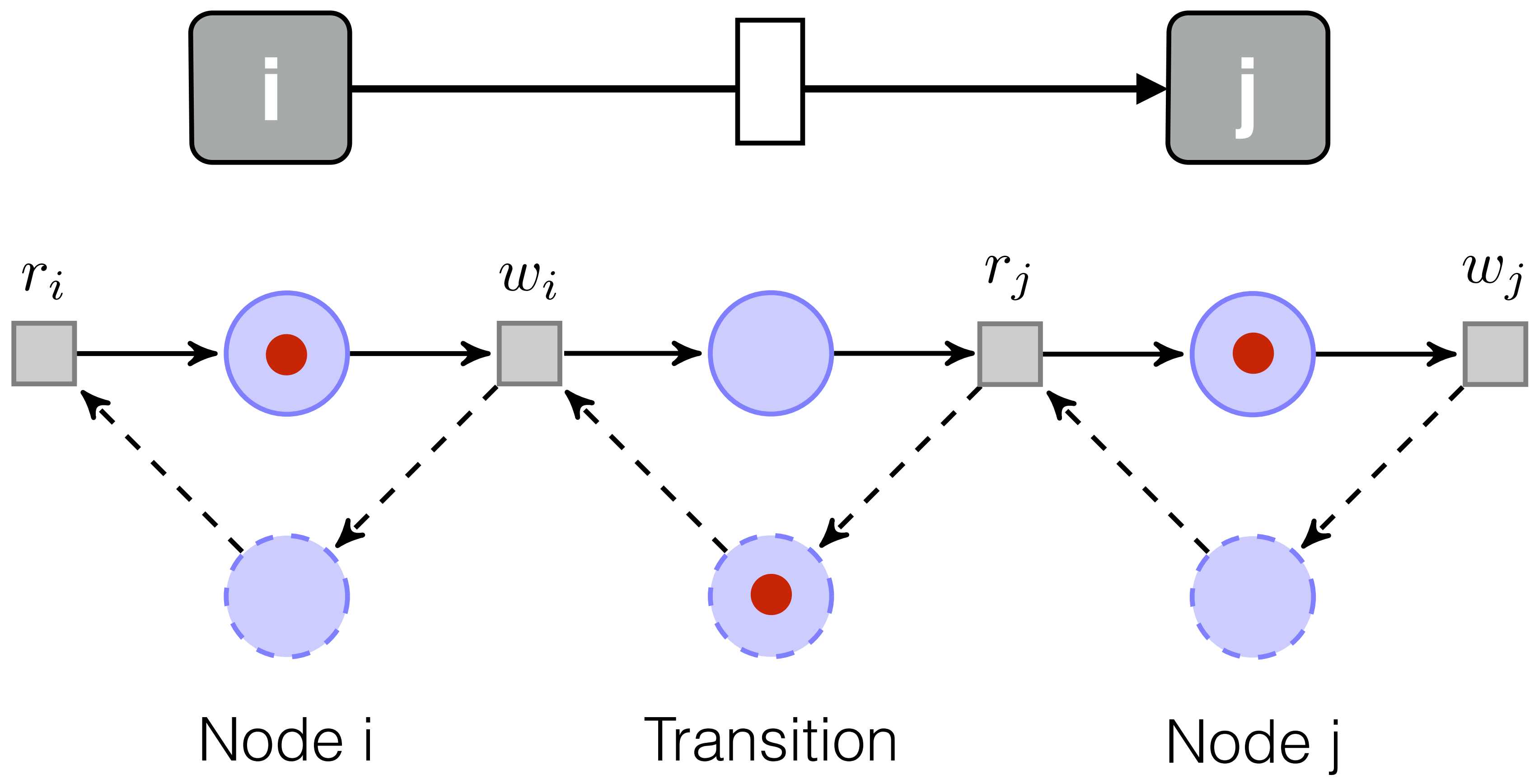
buffer of size 1



- Reading from a buffer is acknowledged to the writer
- Nodes alternate between reads and writes

Back-Pressure Kahn Network

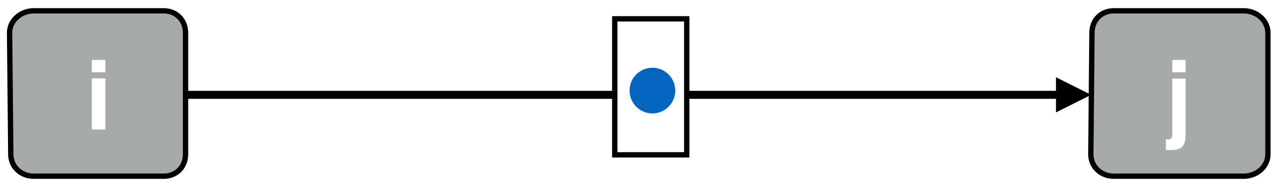
buffer of size 1

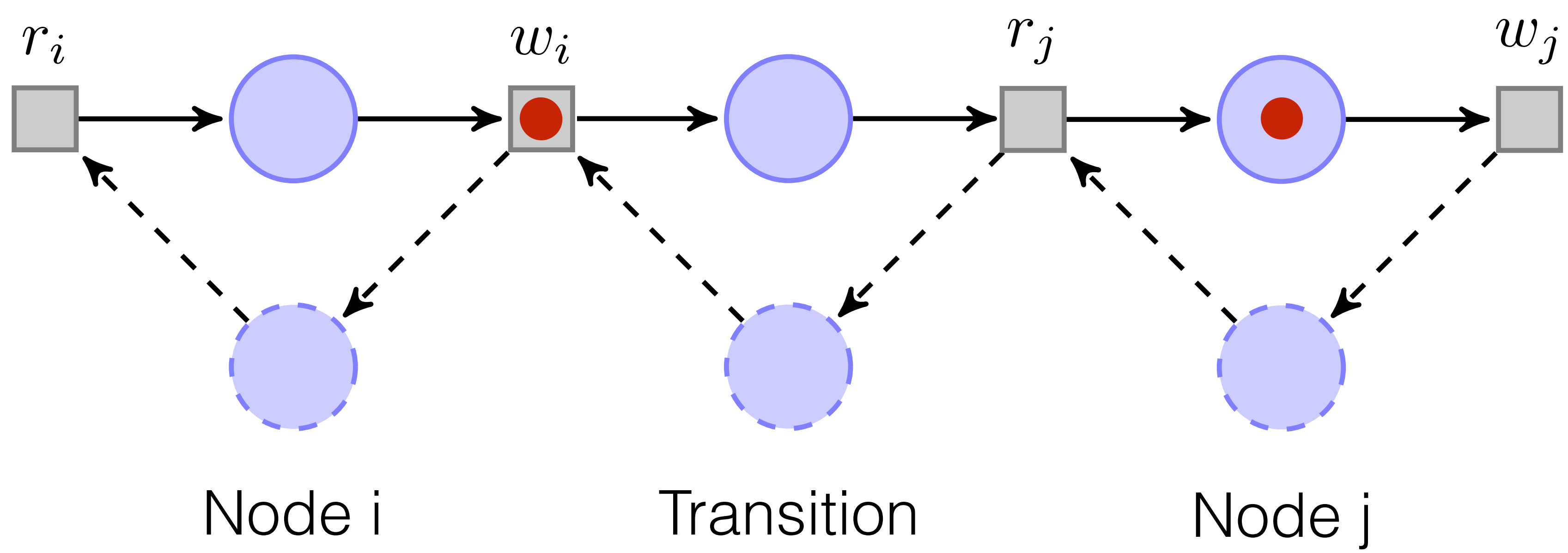


- Reading from a buffer is acknowledged to the writer
- Nodes alternate between reads and writes

Back-Pressure Kahn Network

buffer of size 1

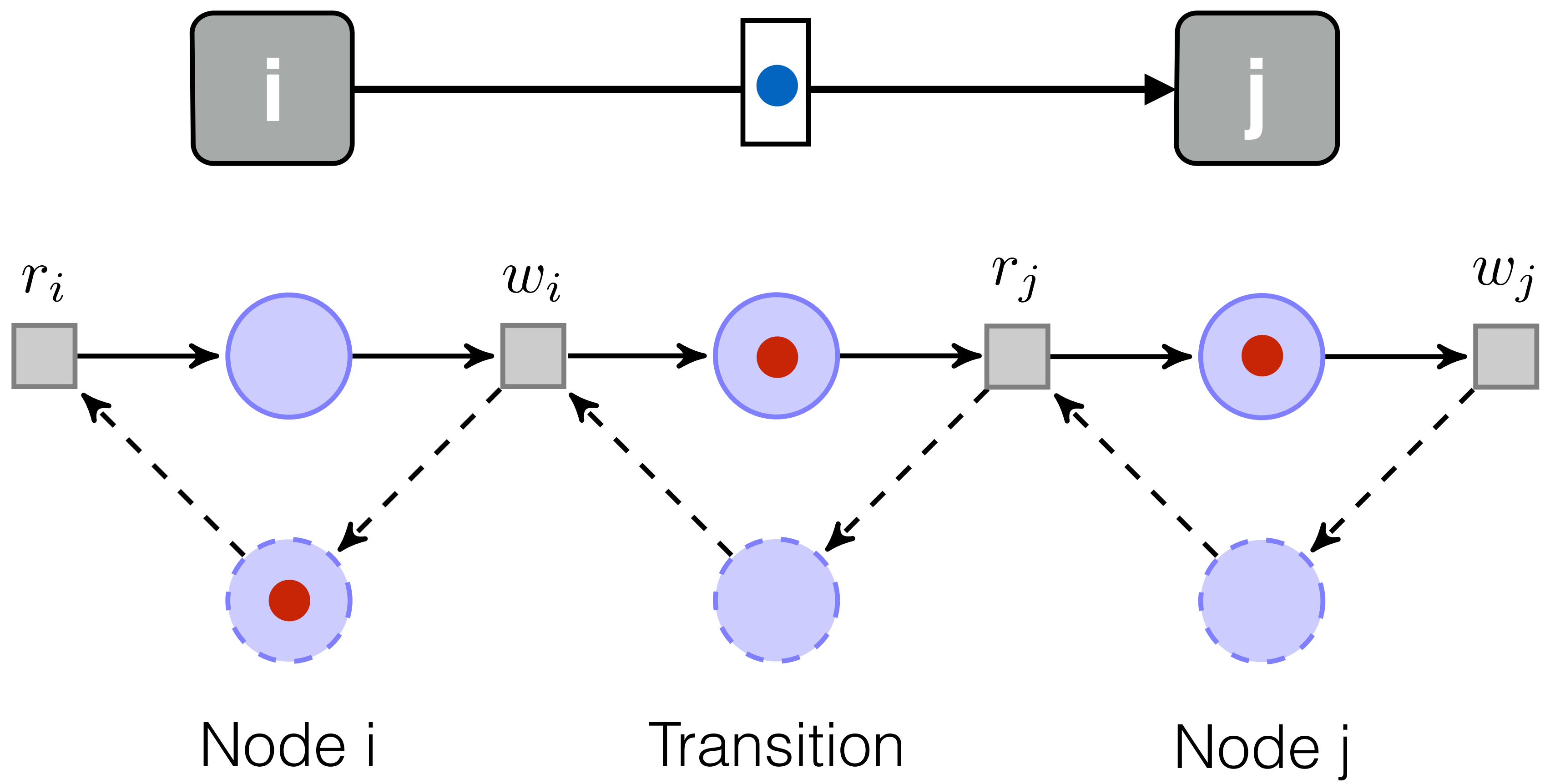




- Reading from a buffer is acknowledged to the writer
- Nodes alternate between reads and writes

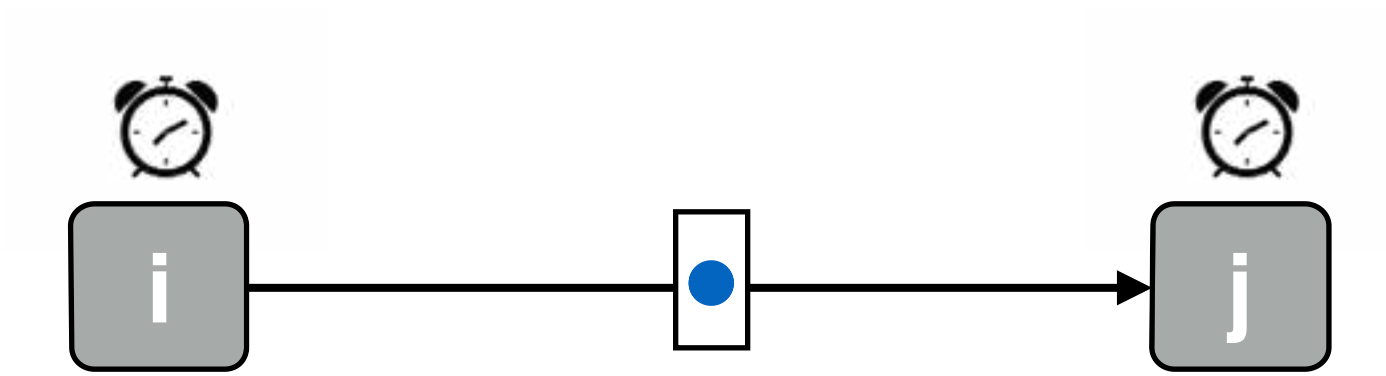
Back-Pressure Kahn Network

buffer of size 1



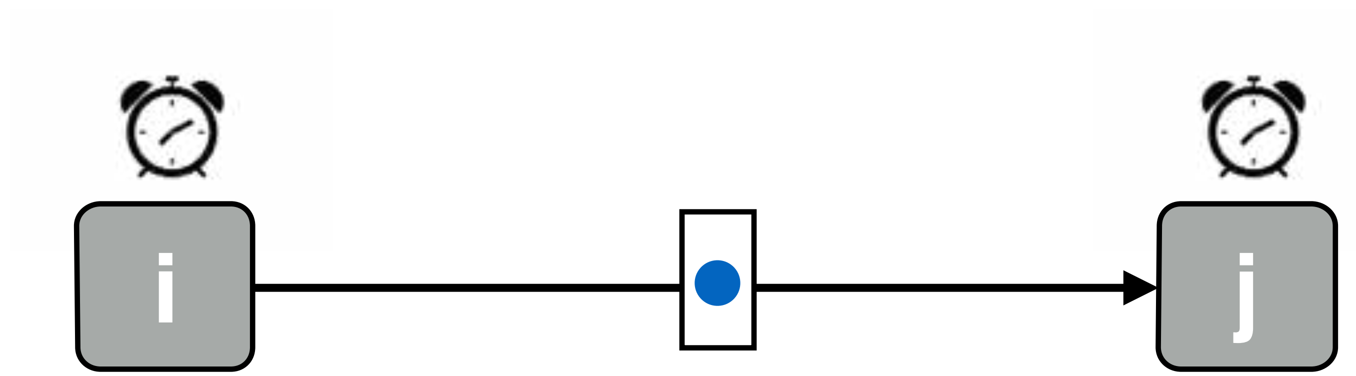
- Reading from a buffer is acknowledged to the writer
- Nodes alternate between reads and writes

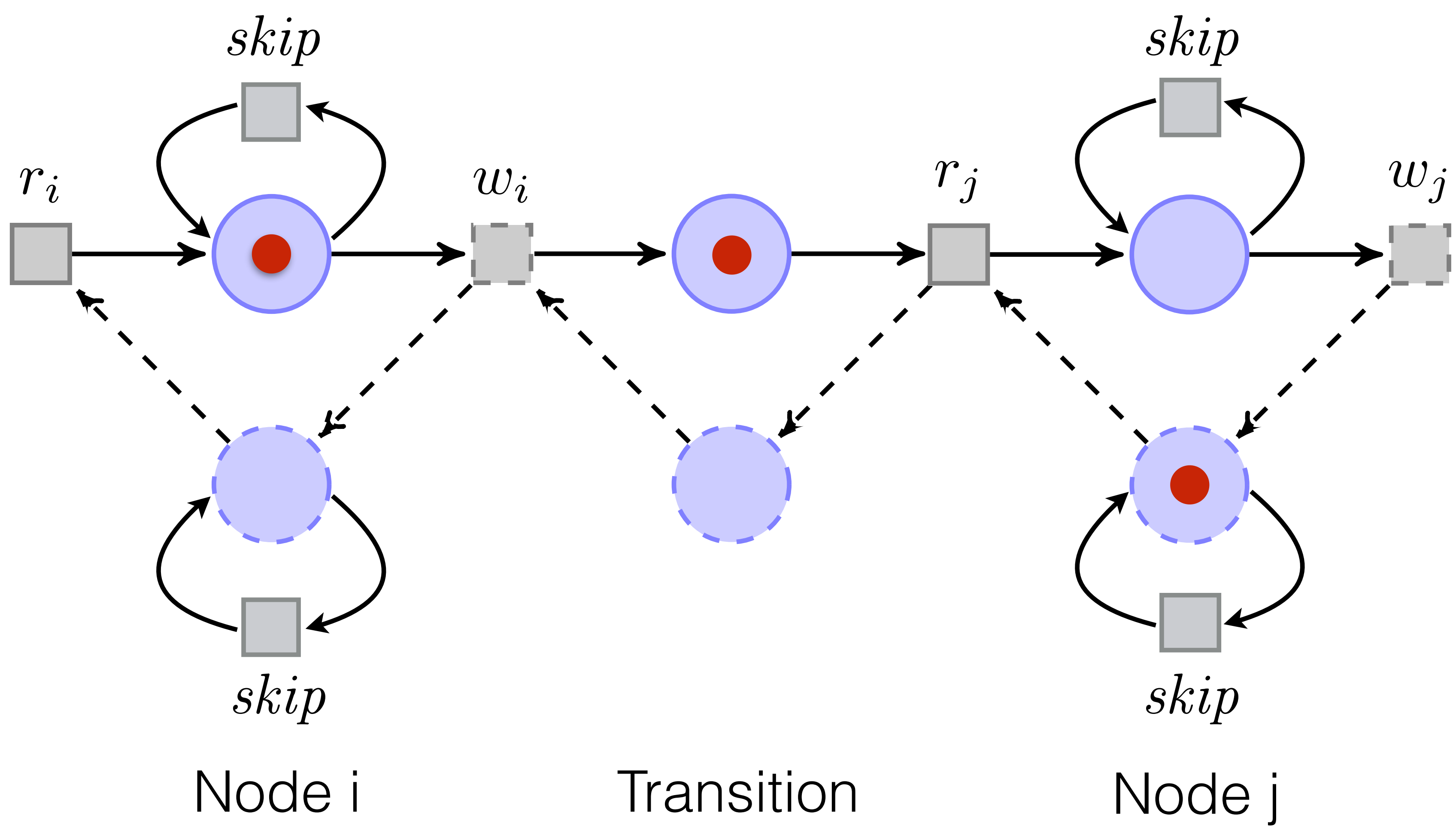
Back-Pressure LTTA



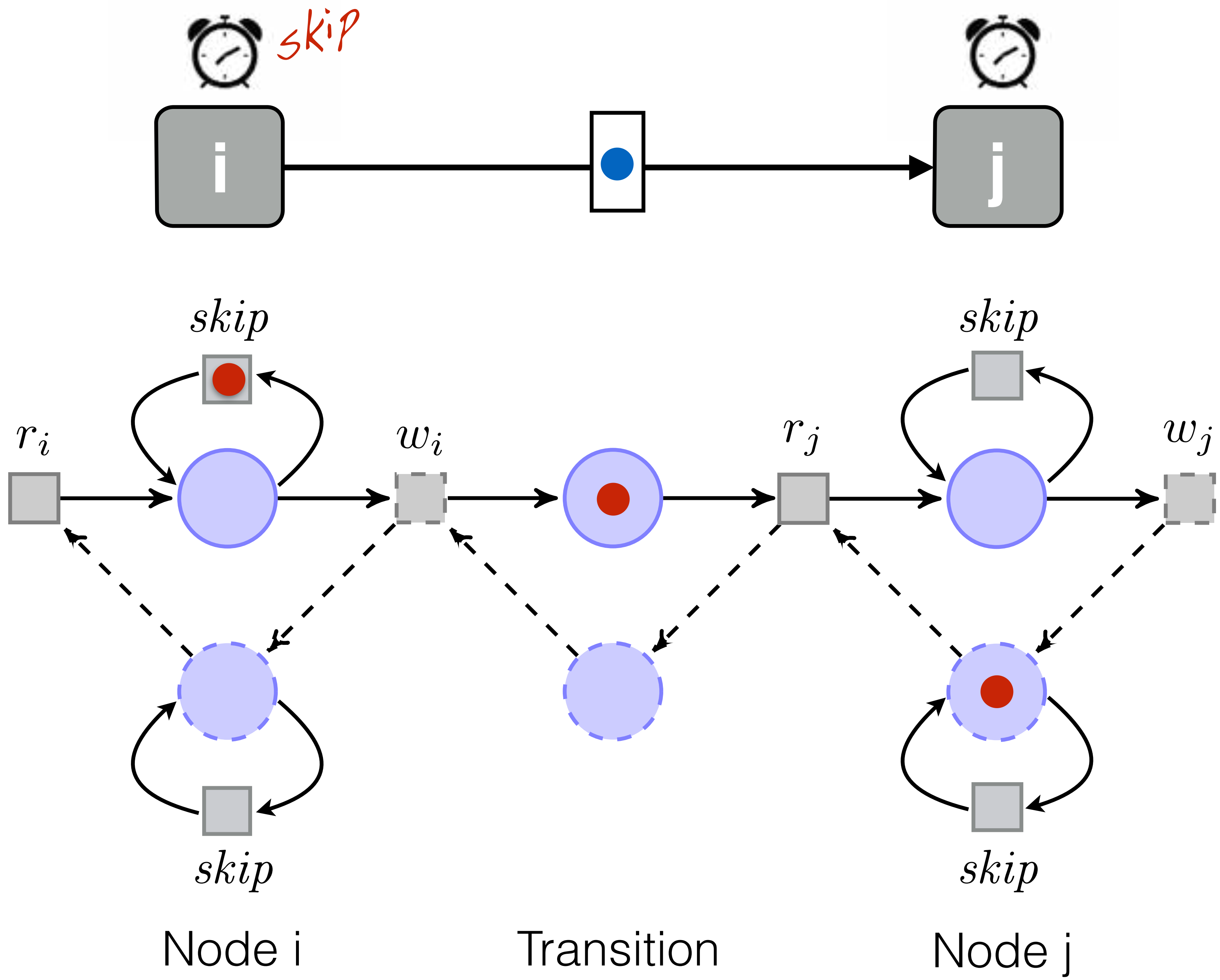
- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism

Back-Pressure LTTA

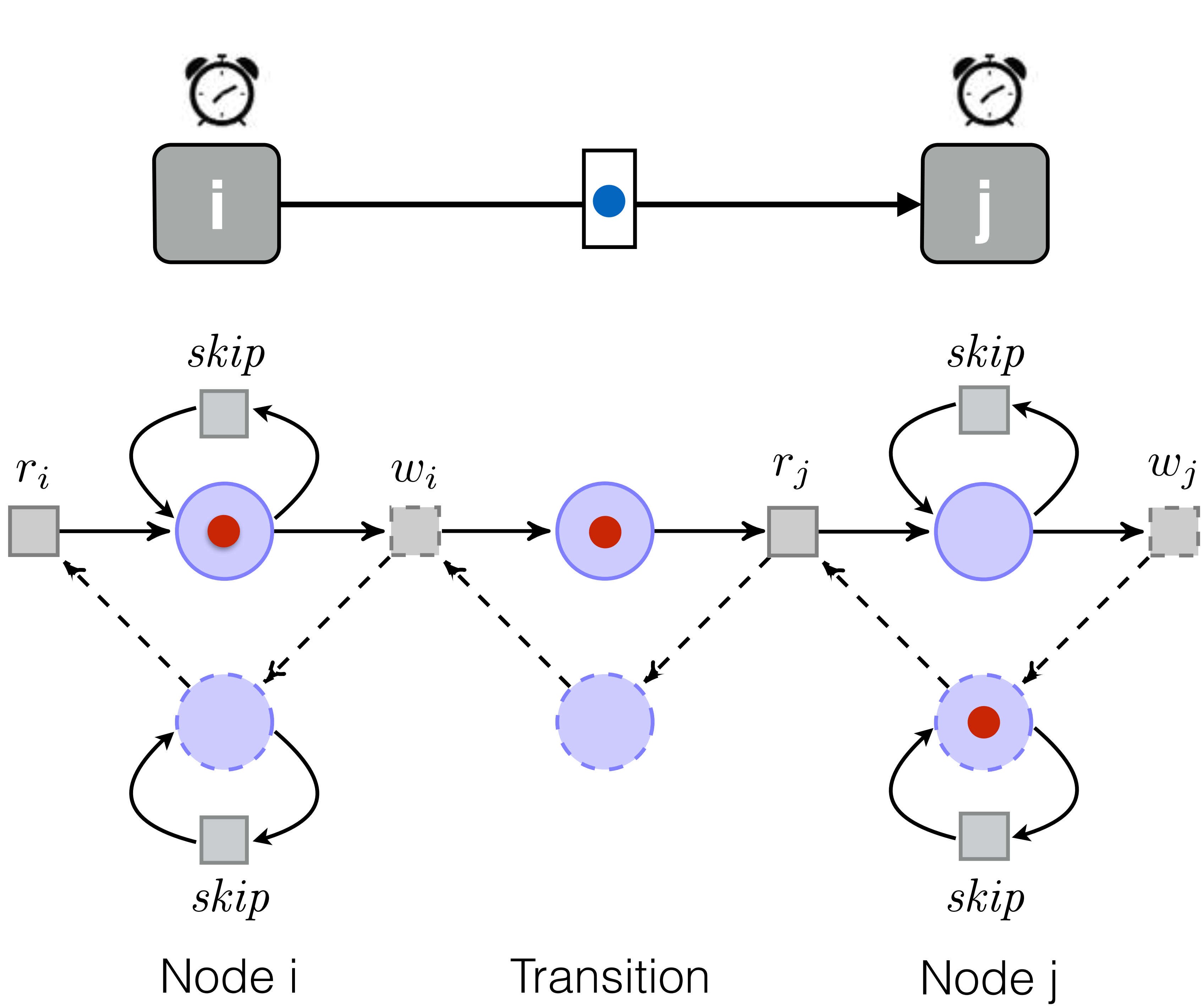




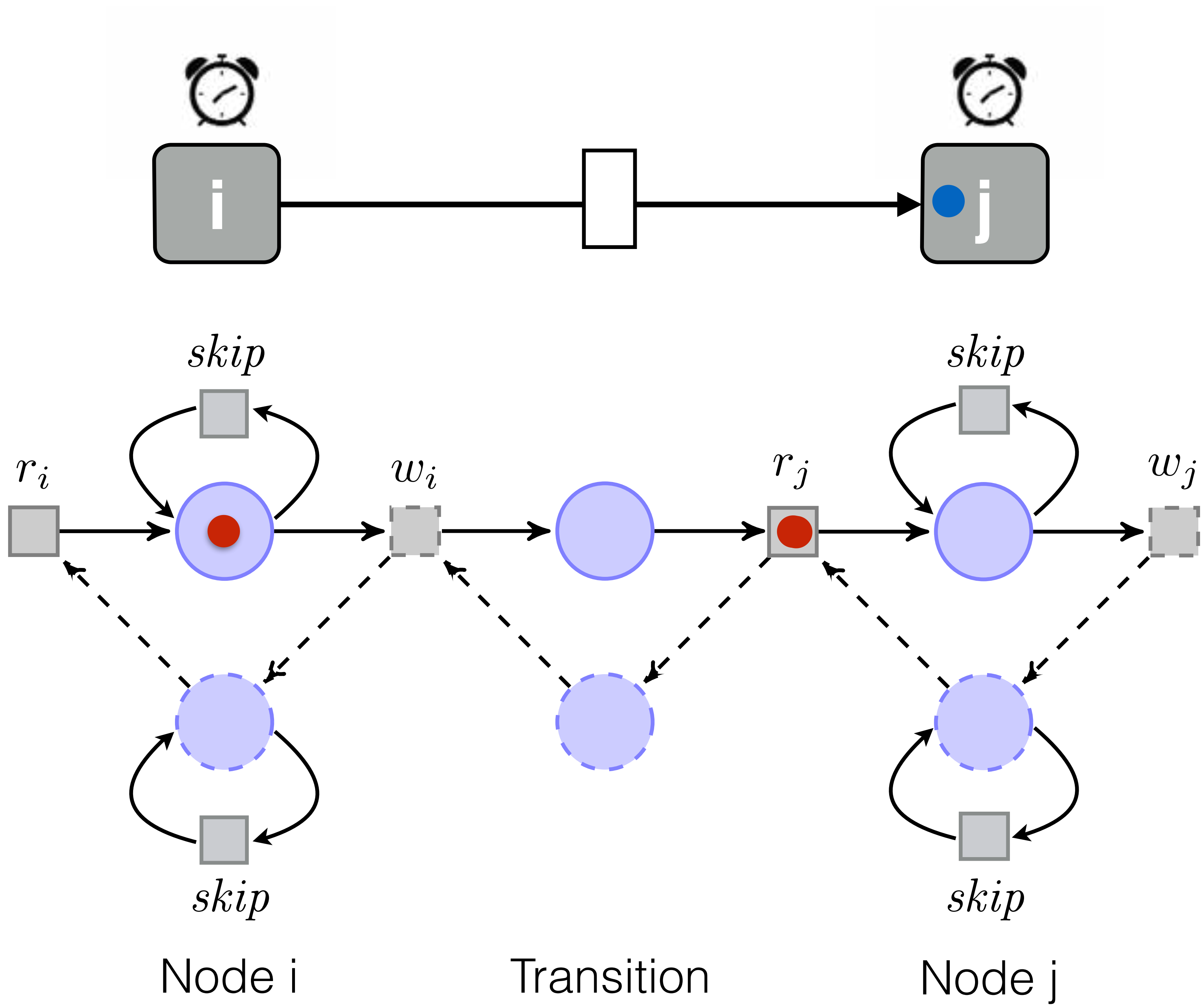
Back-Pressure LTTA



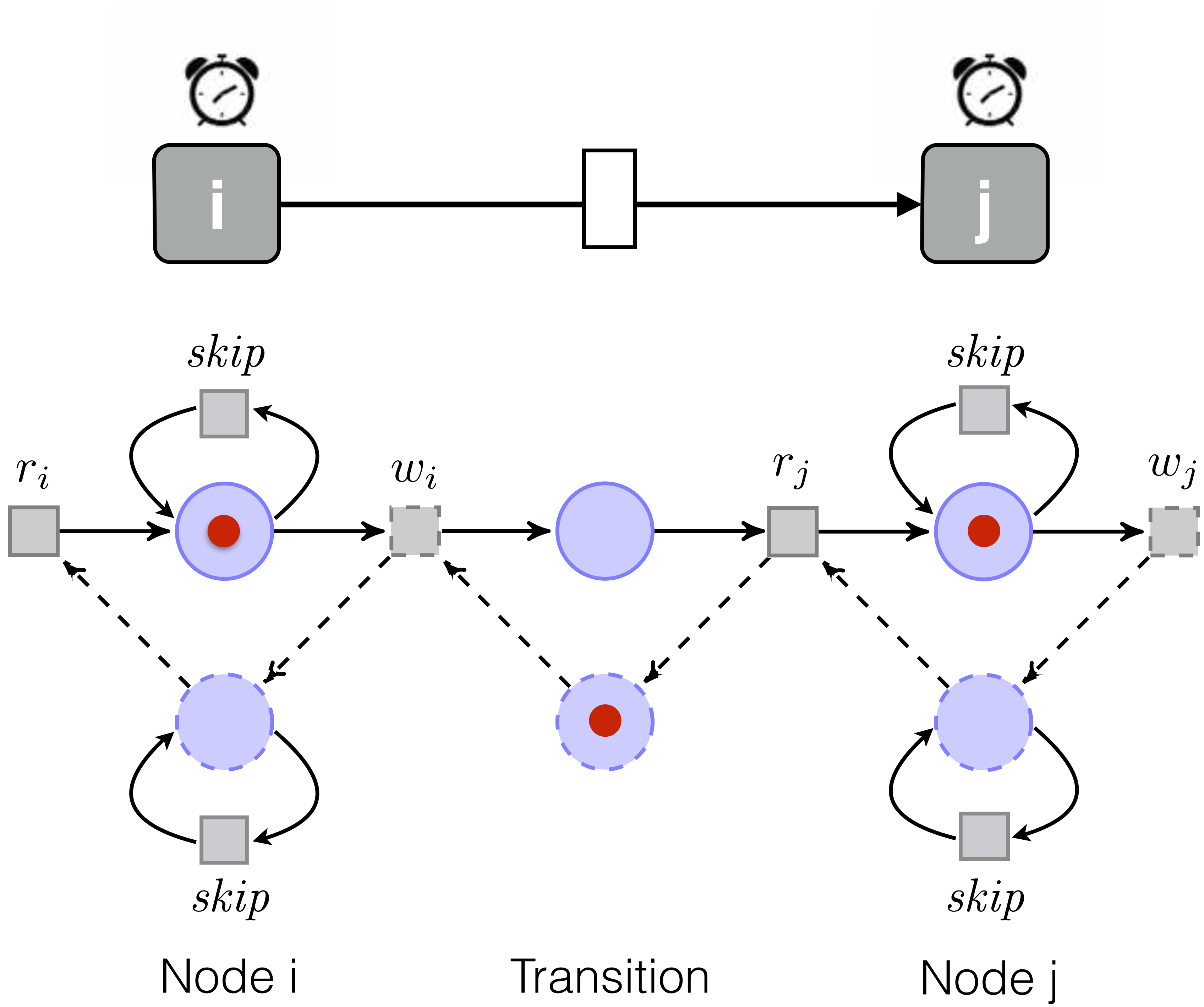
Back-Pressure LTTA



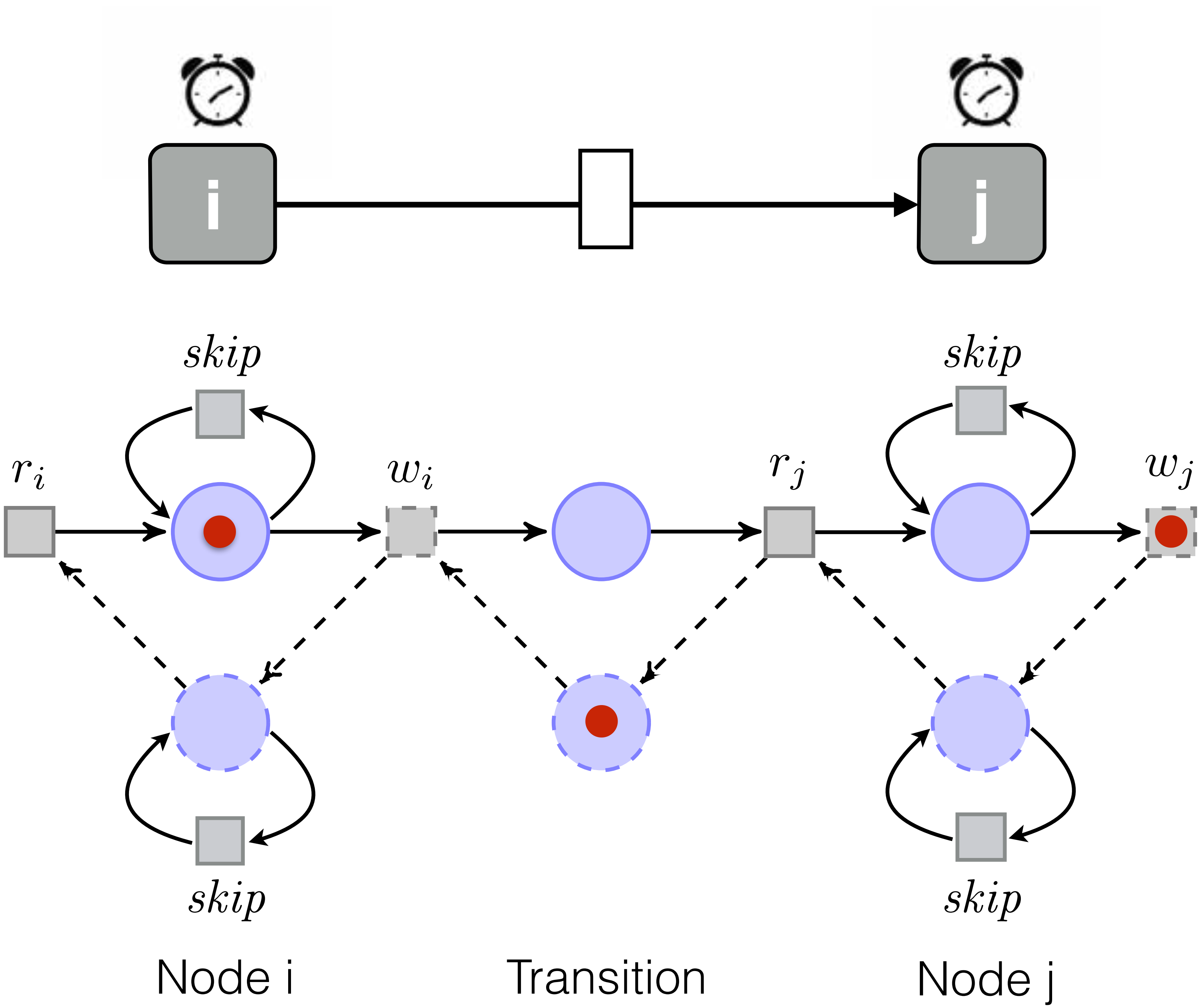
Back-Pressure LTTA



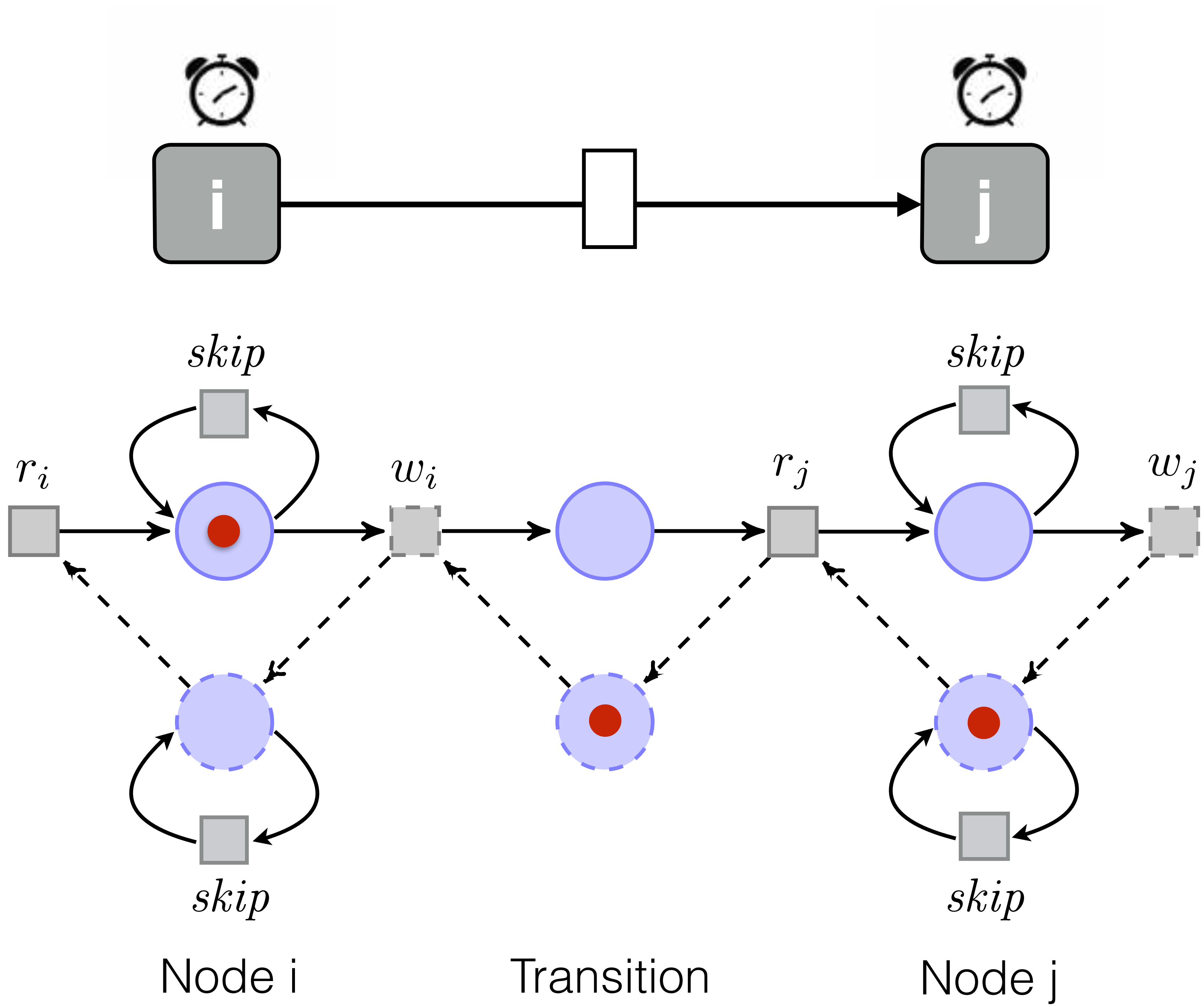
Back-Pressure LTTA



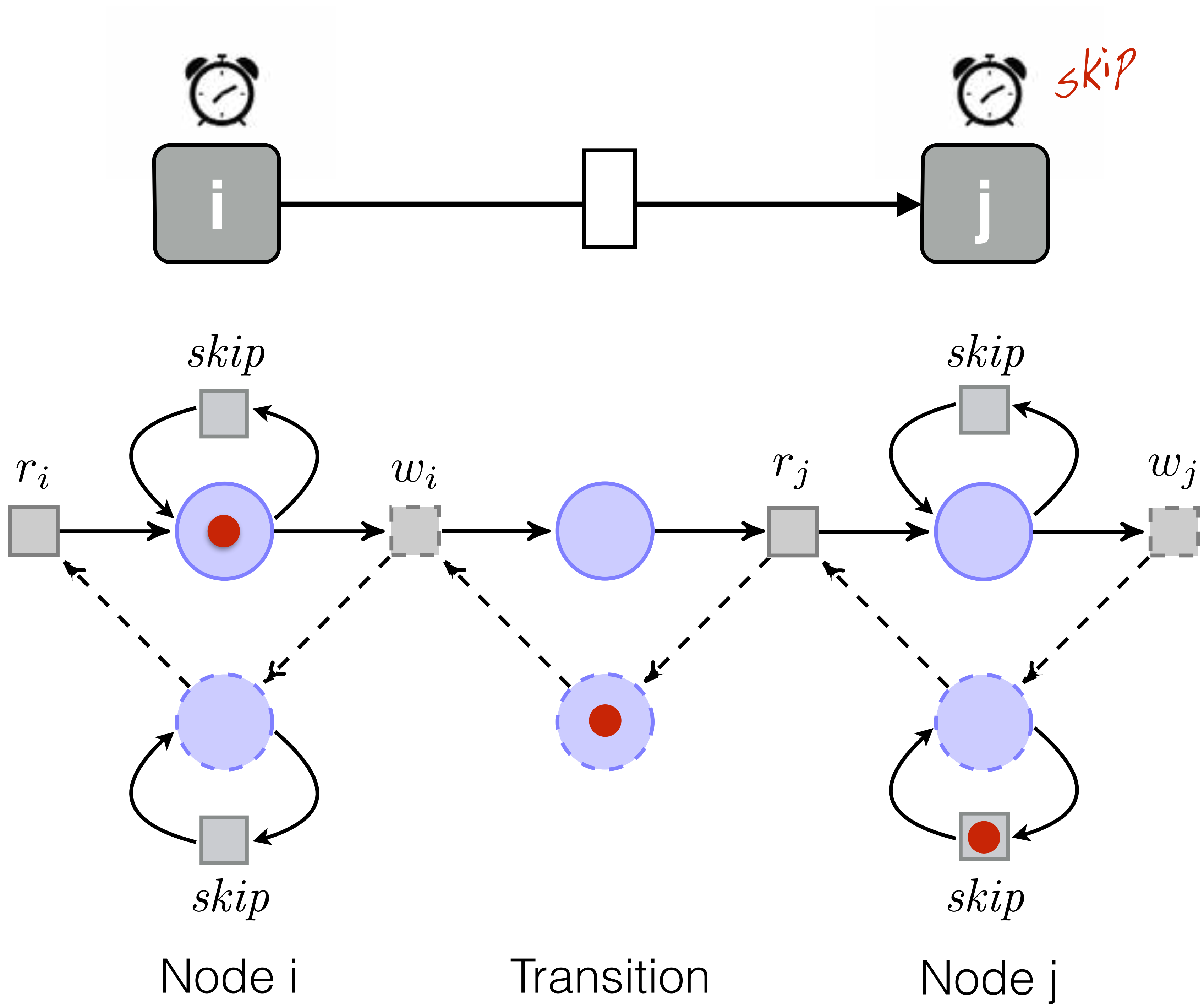
Back-Pressure LTTA



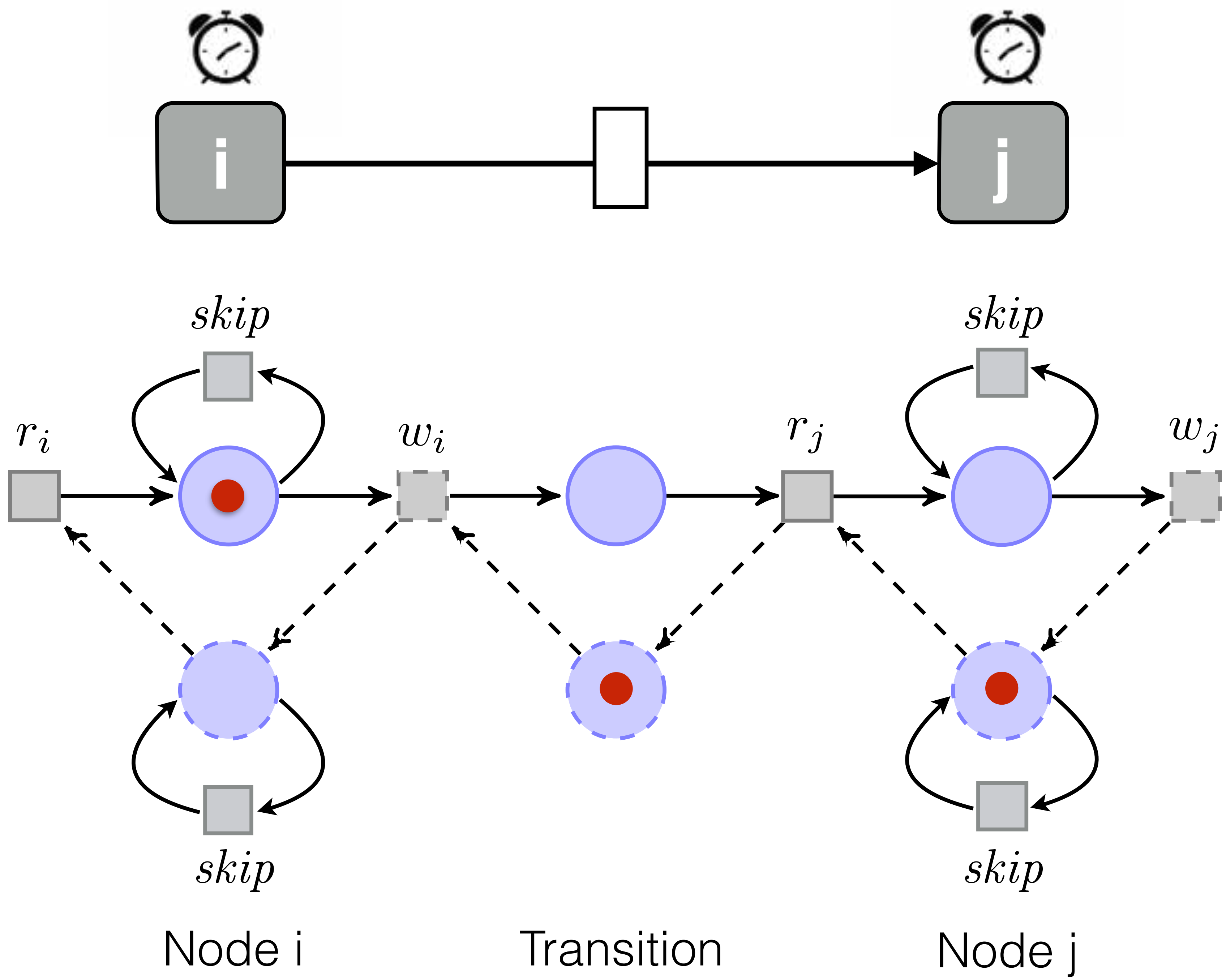
Back-Pressure LTTA



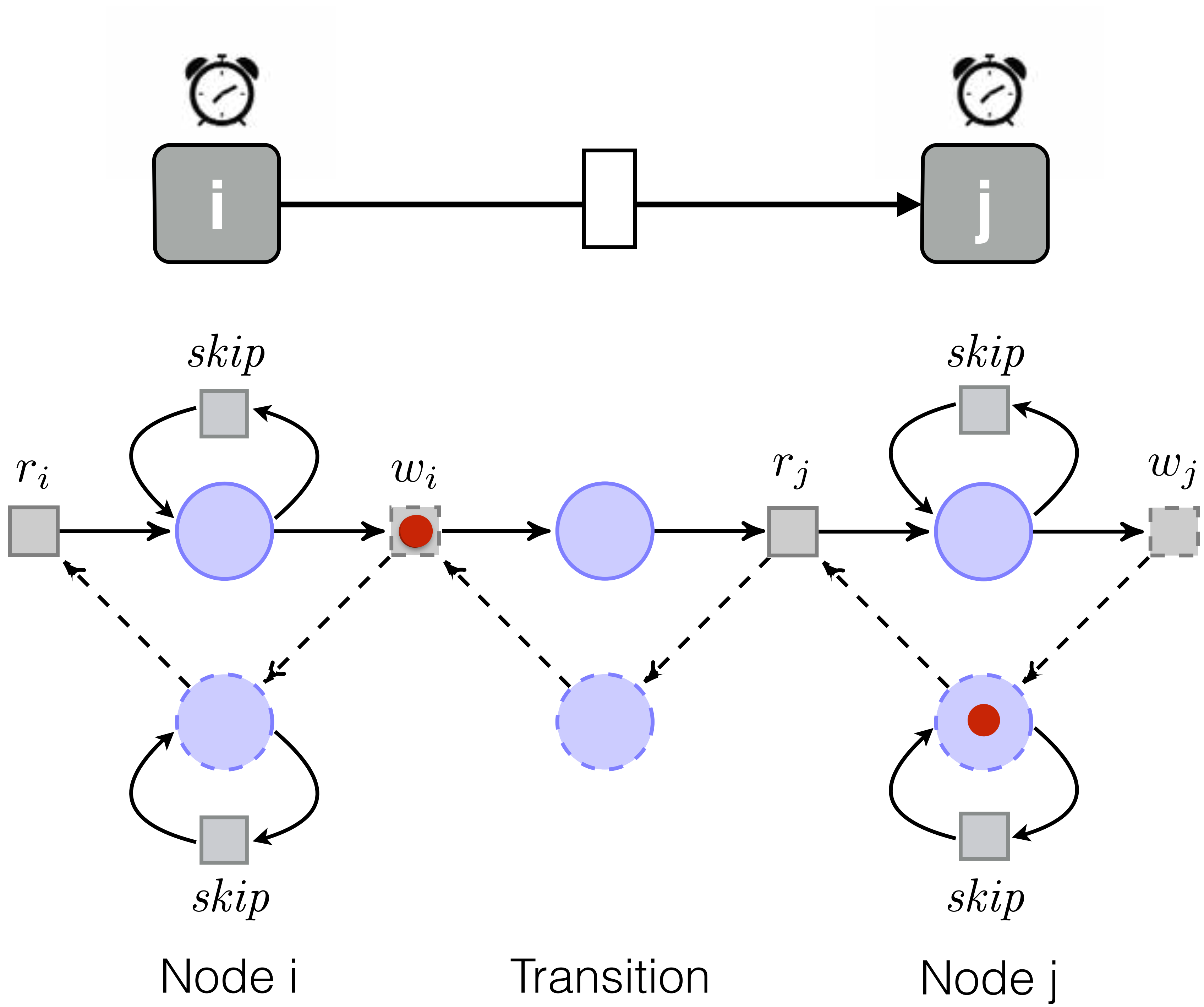
Back-Pressure LTTA



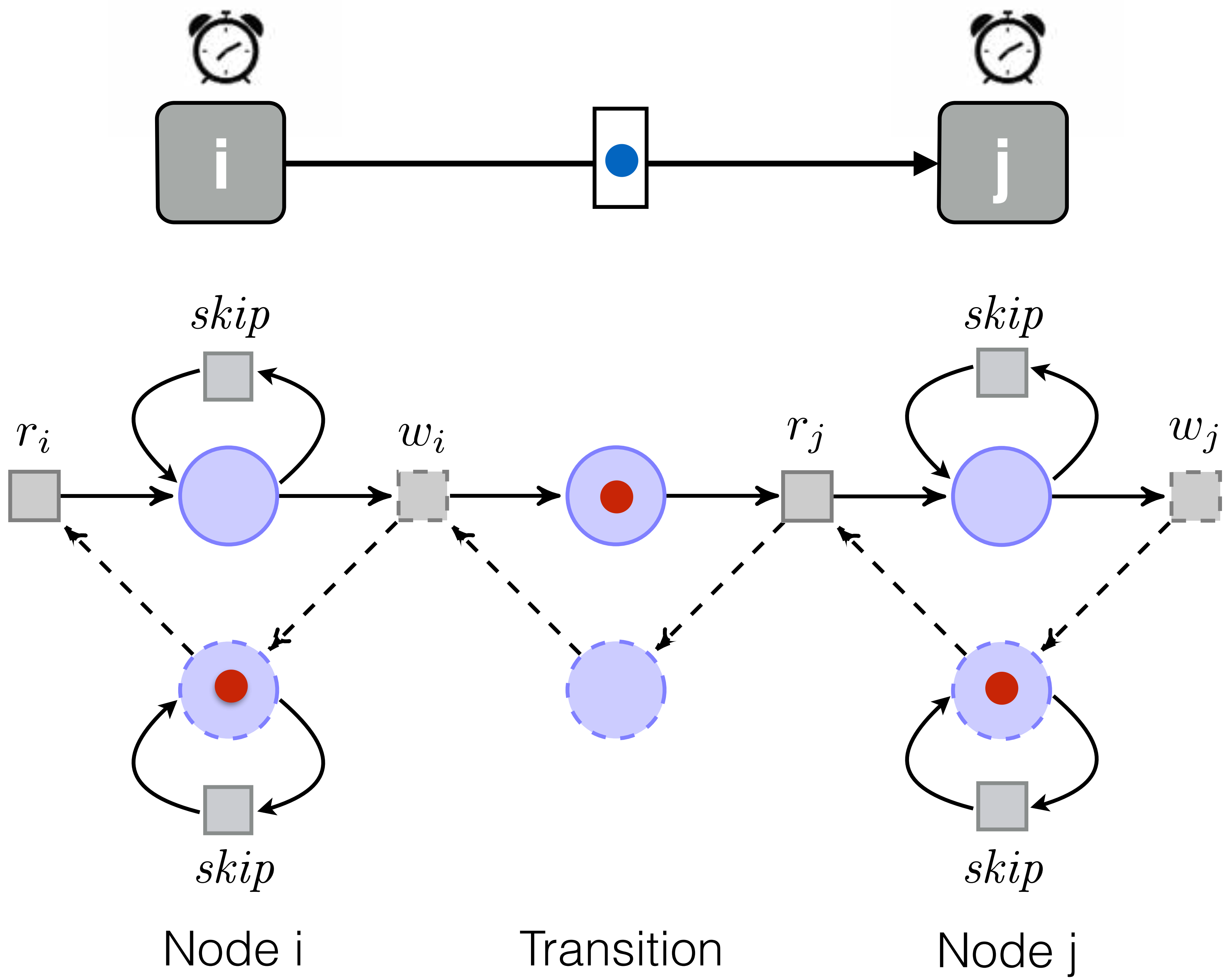
Back-Pressure LTTA



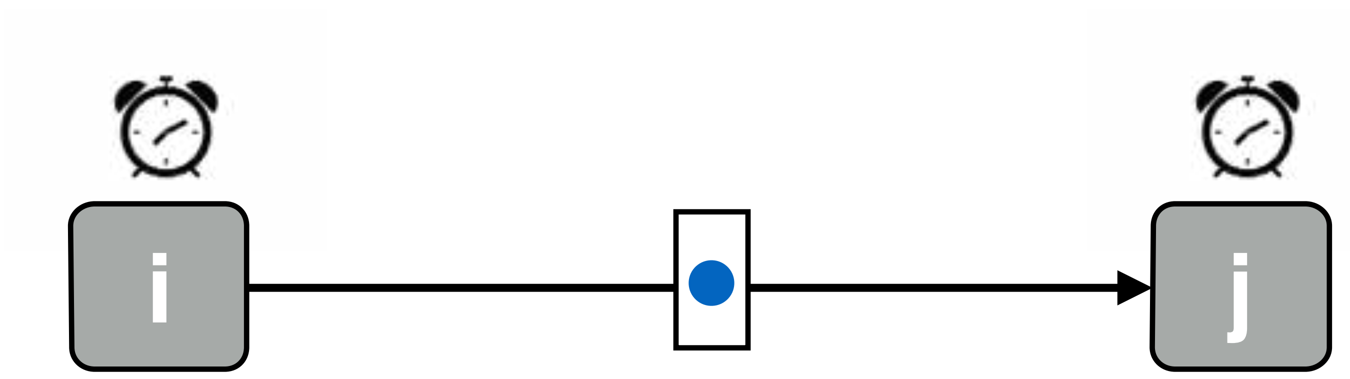
Back-Pressure LTTA



Back-Pressure LTTA



Back-Pressure LTTA

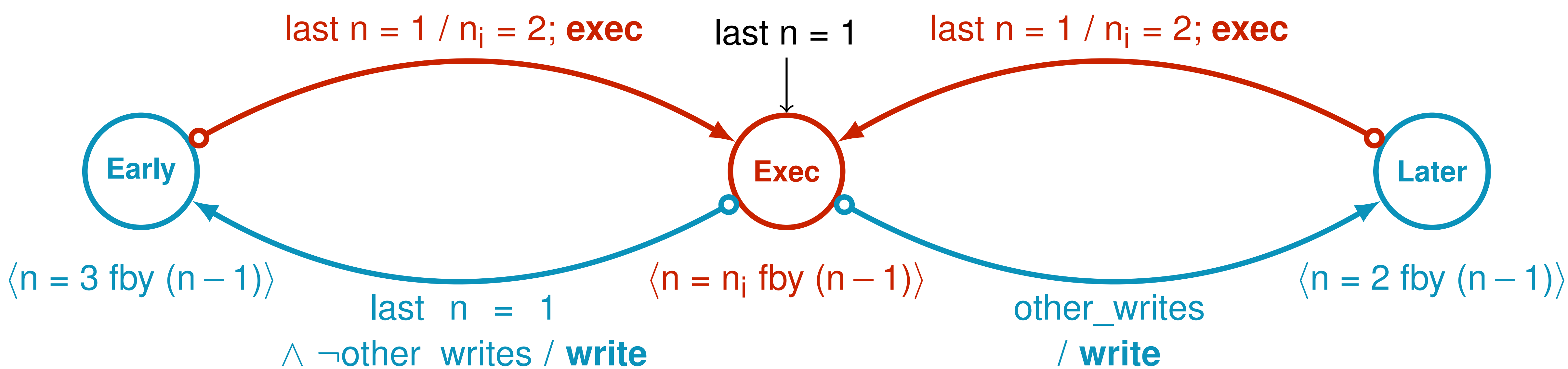


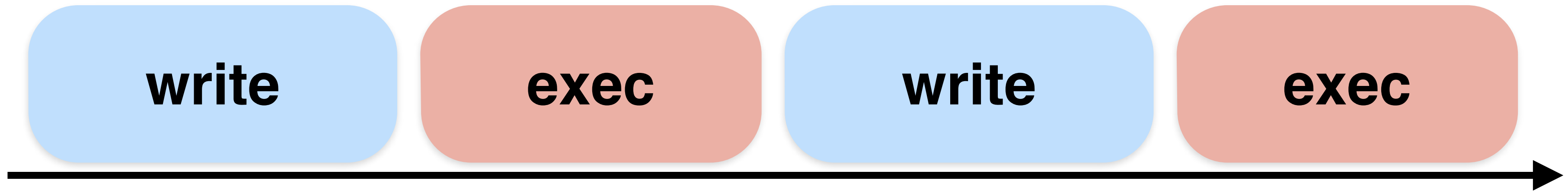
- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism

Preservation of the **discrete** synchronous semantics
(forget the skips)

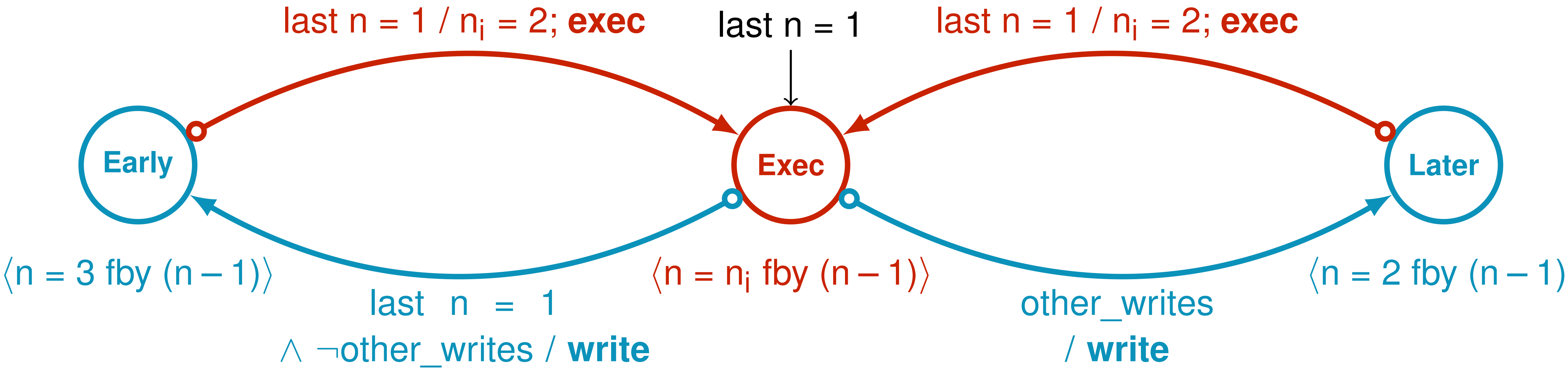
Time-Based LTTA

- Nodes alternate between **execution** and **broadcast**
- Nodes can spend **several ticks** in a mode

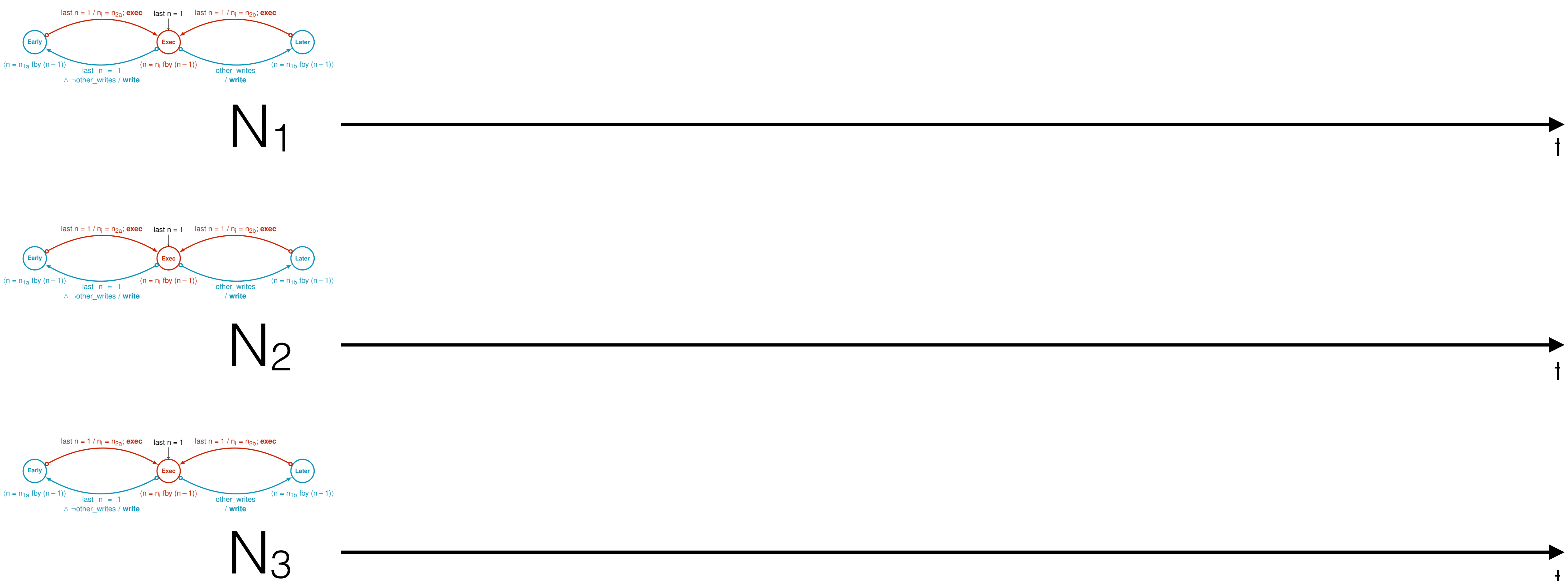




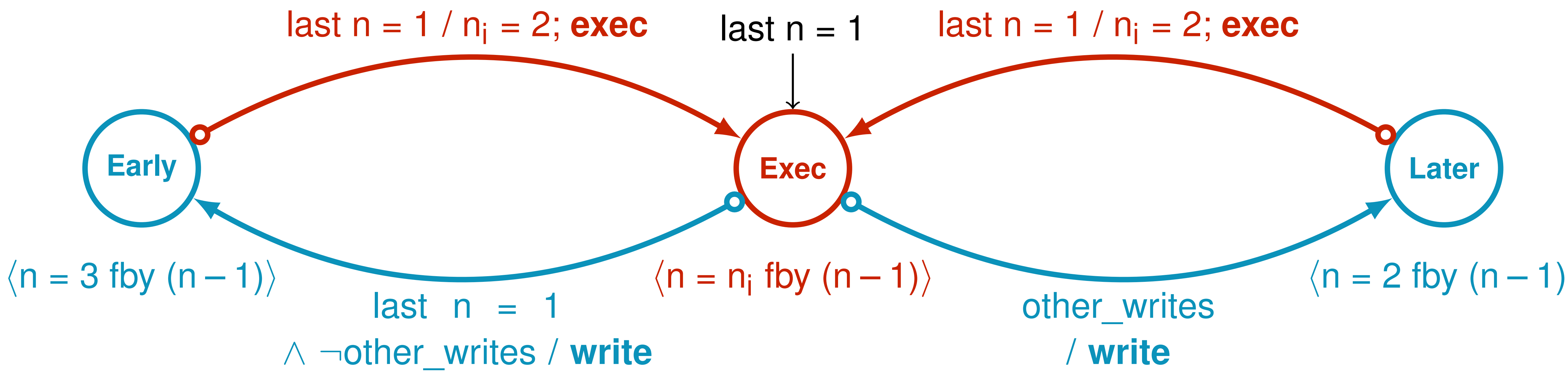
Initial counter values depend on period and communication delay bounds

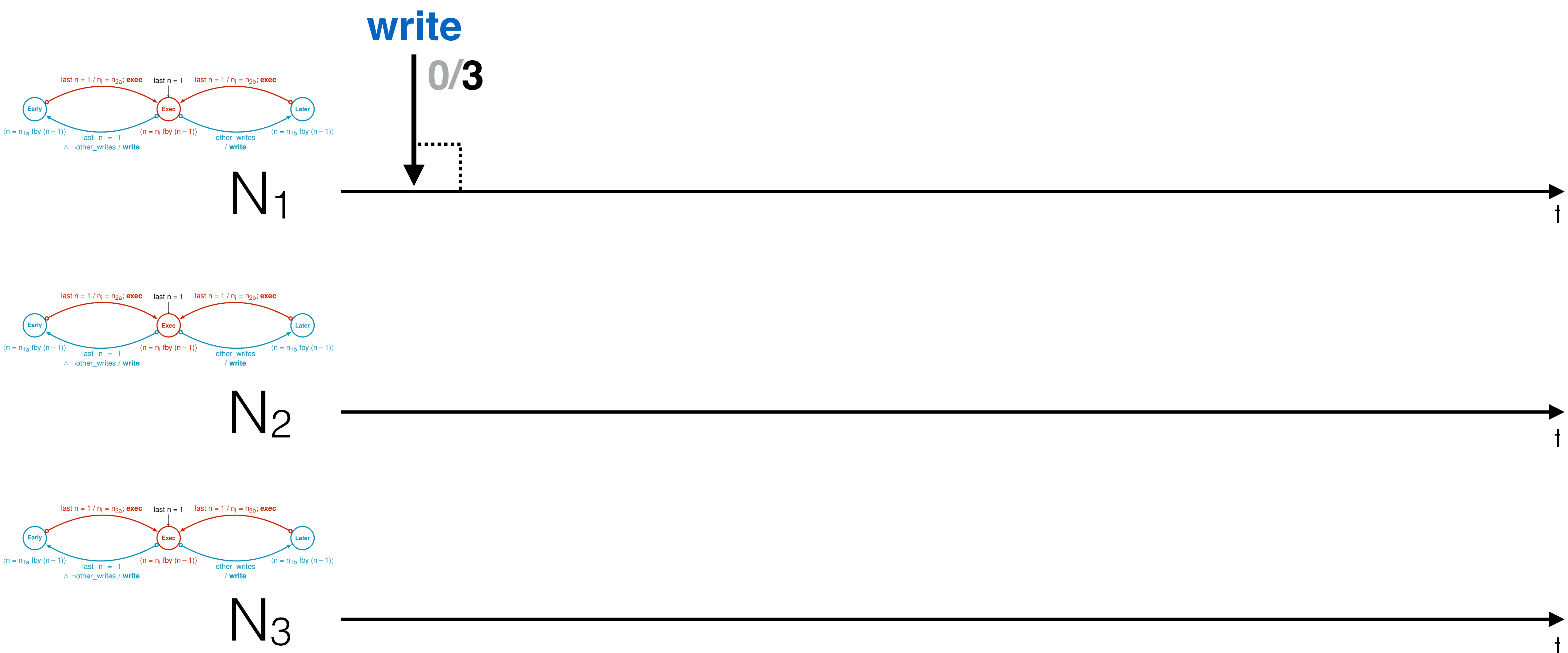


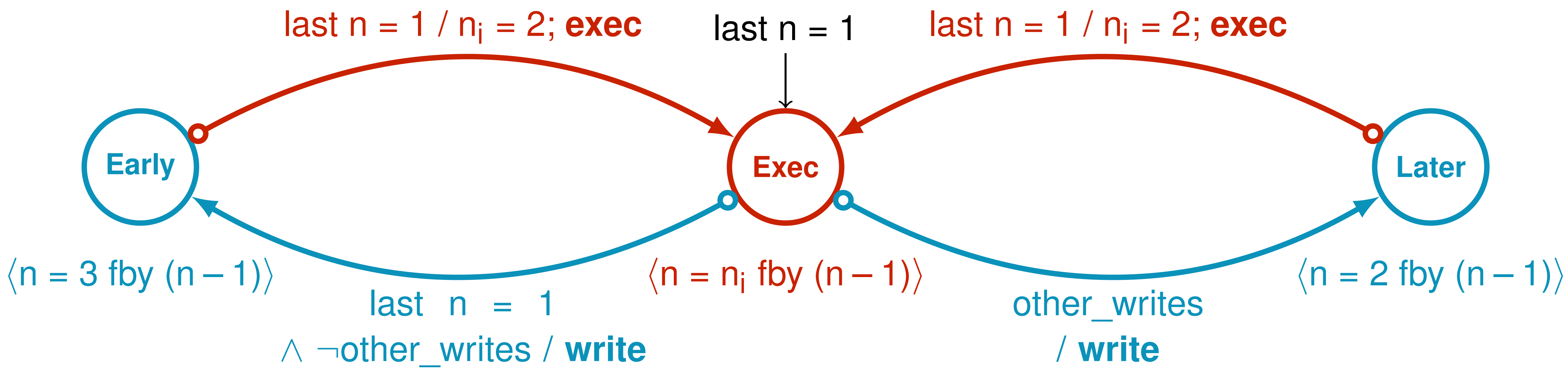
last n = 1 / n_i = 2; exec
last n = 1
last n = 1 / n_i = 2; exec
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / write
other_writes
/ write

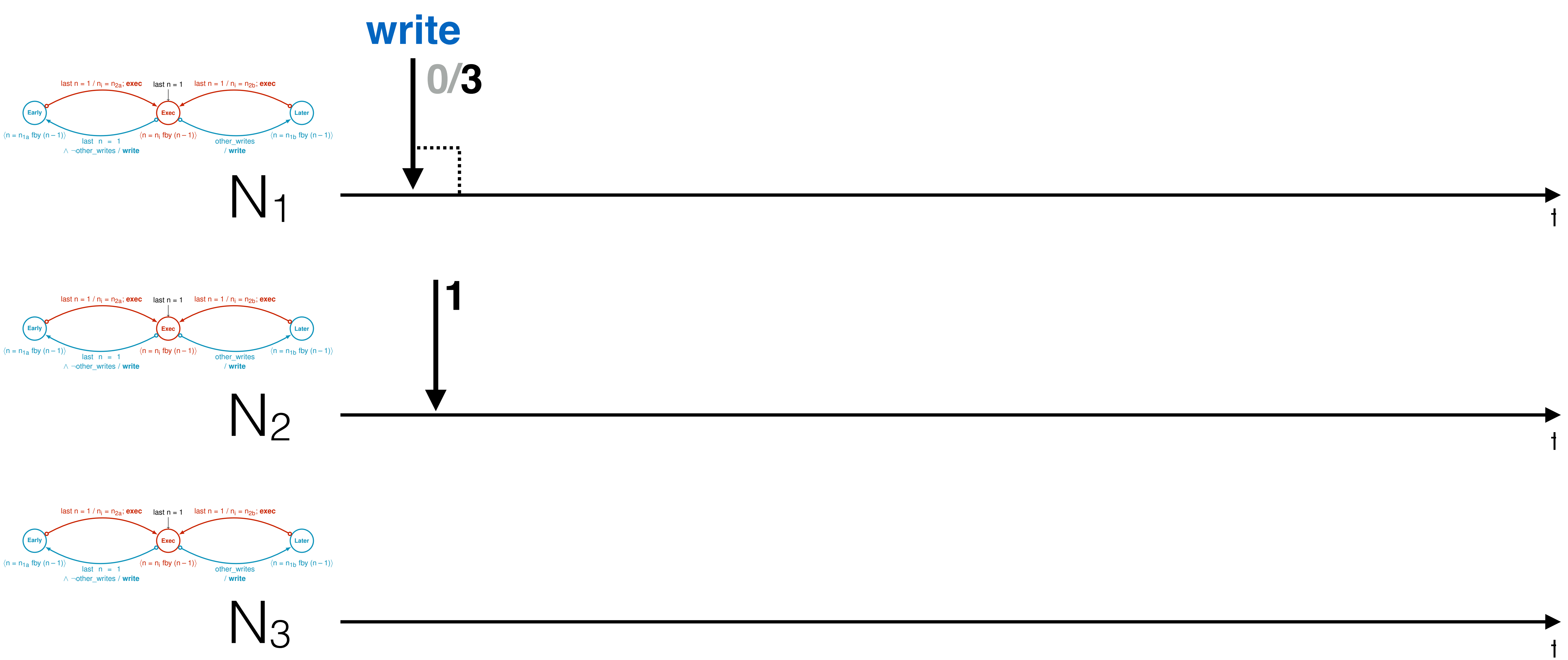


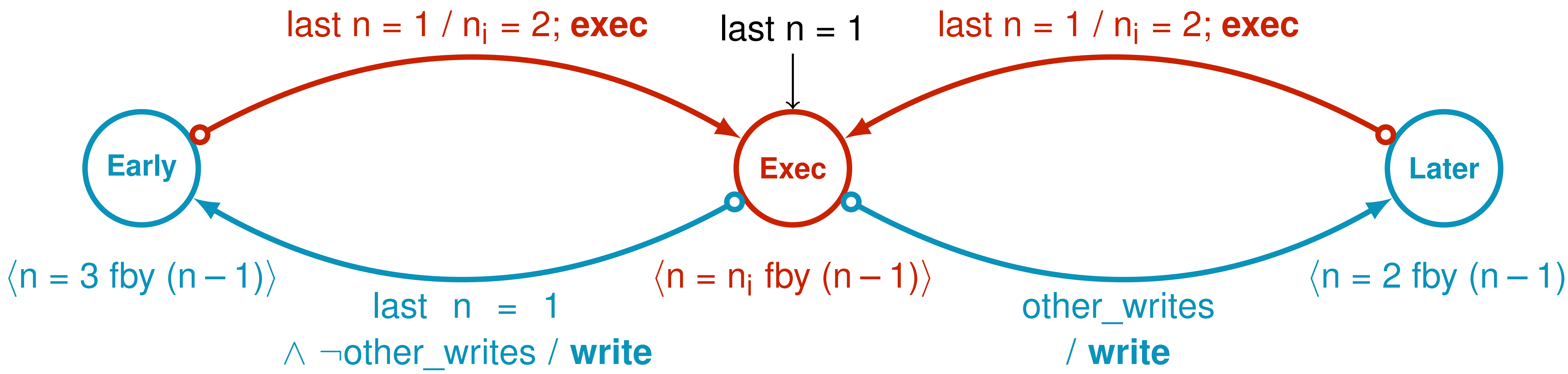
N_1
t
N_2
t
N_3
t


last n = 1 / n_i = 2; **exec**
last n = 1
last n = 1 / n_i = 2; **exec**
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / **write**
other_writes
/ **write**

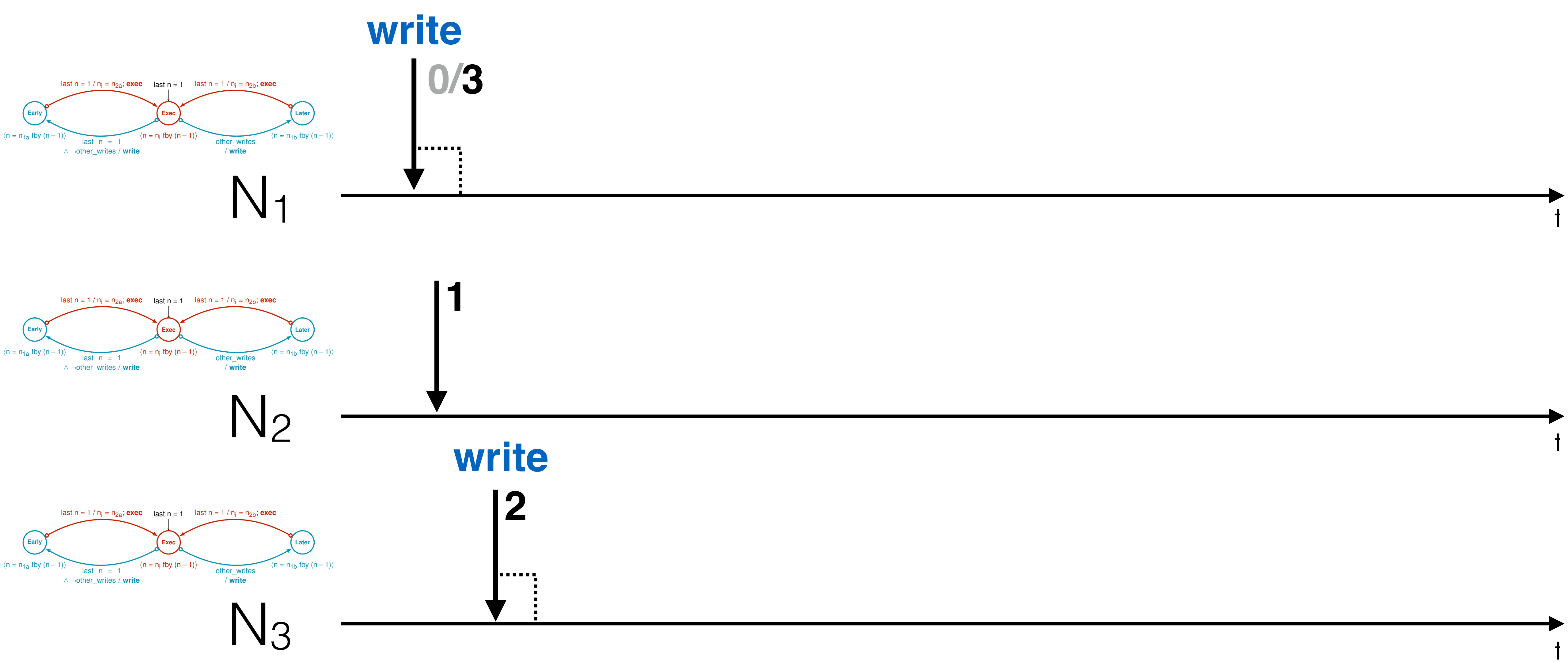

write
0/**3**
N_1
t
N_2
t
N_3
t


last n = 1 / n_i = 2; **exec**
last n = 1
last n = 1 / n_i = 2; **exec**
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / **write**
other_writes
/ **write**

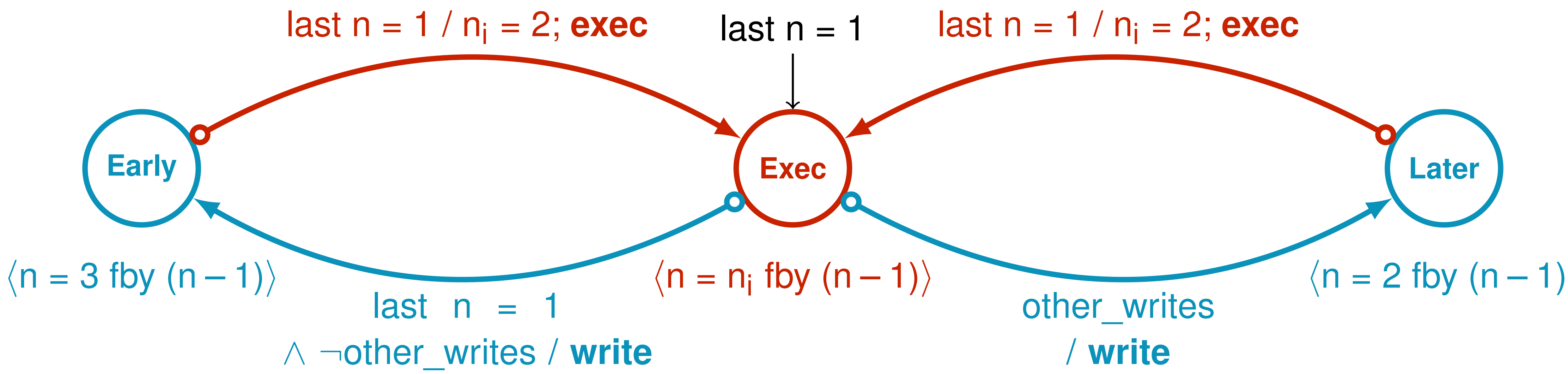

write
0/3
N1
t
1
N2
t
N3
t

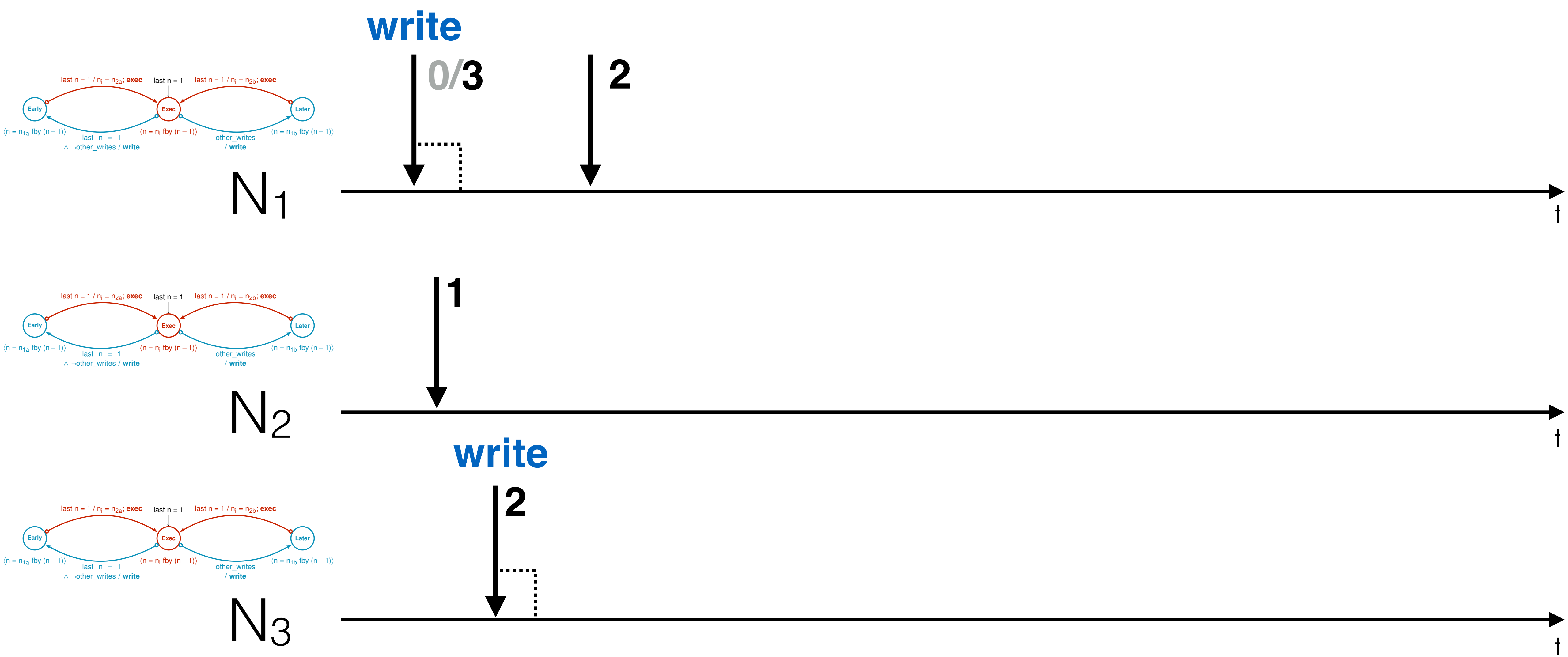


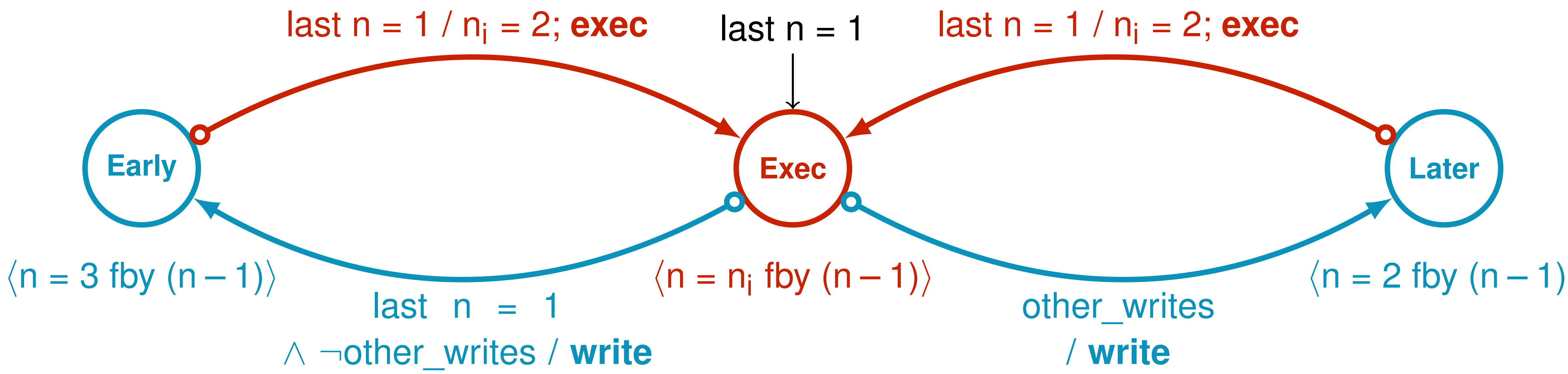
last n = 1 / n_i = 2; exec
last n = 1
last n = 1 / n_i = 2; exec
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / write
other_writes
/ write

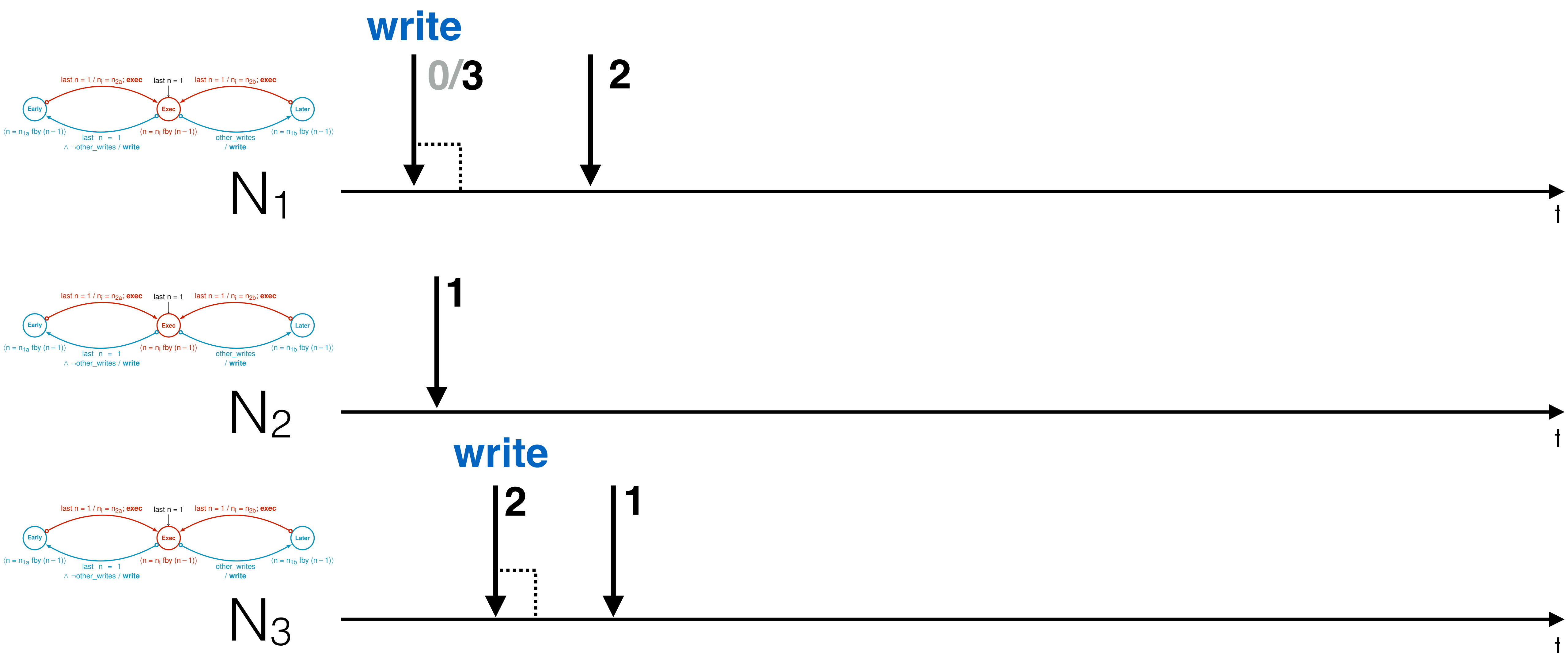


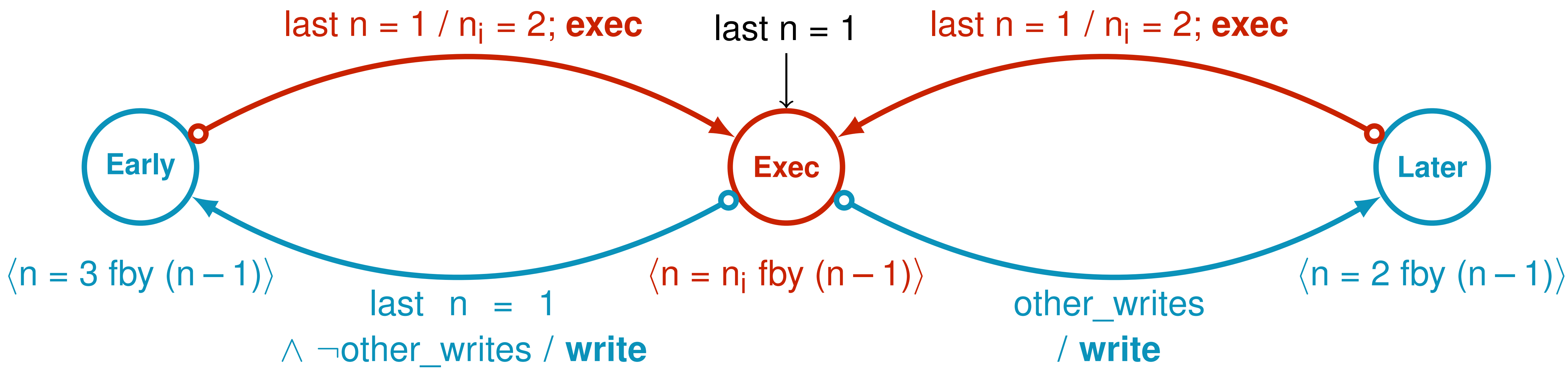
write
0/3
N1
t
1
N2
t
write
2
N3
t

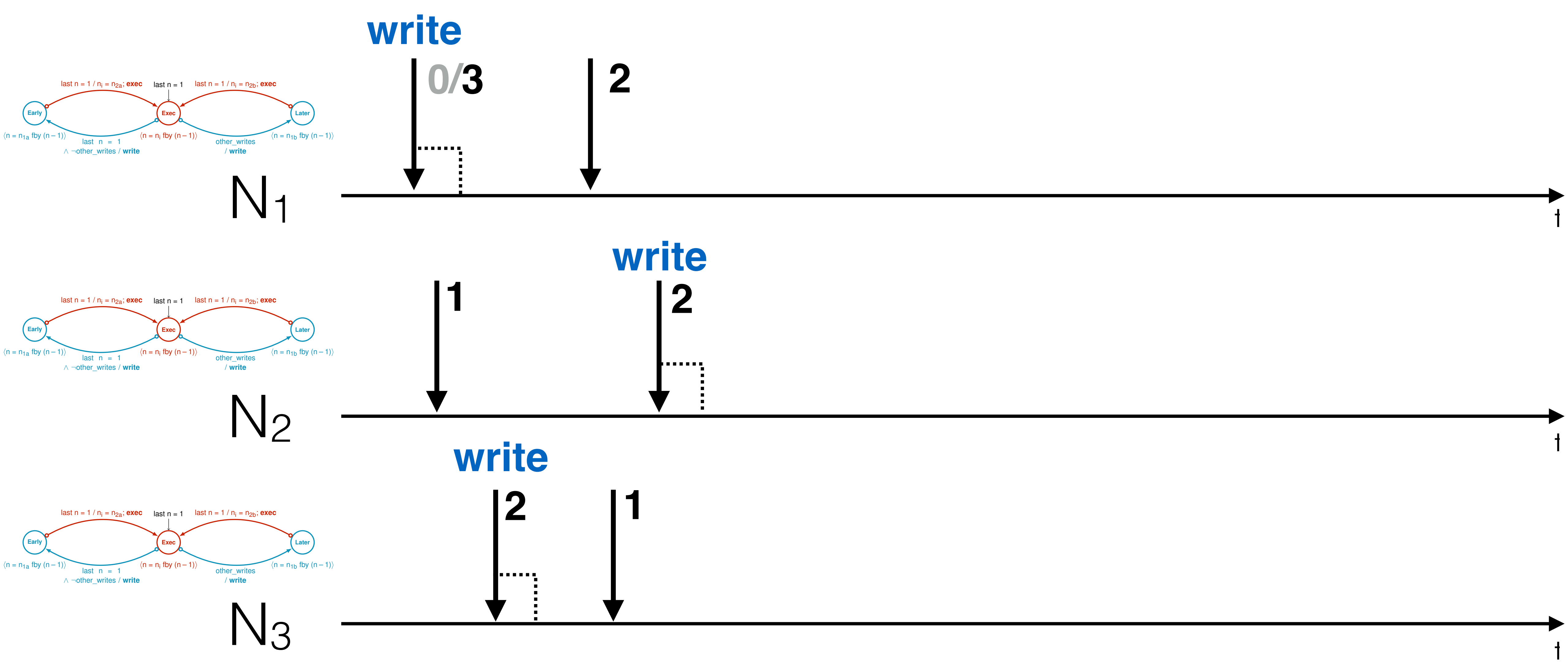

last n = 1 / n_i = 2; **exec**
last n = 1
last n = 1 / n_i = 2; **exec**
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / **write**
other_writes
/ **write**

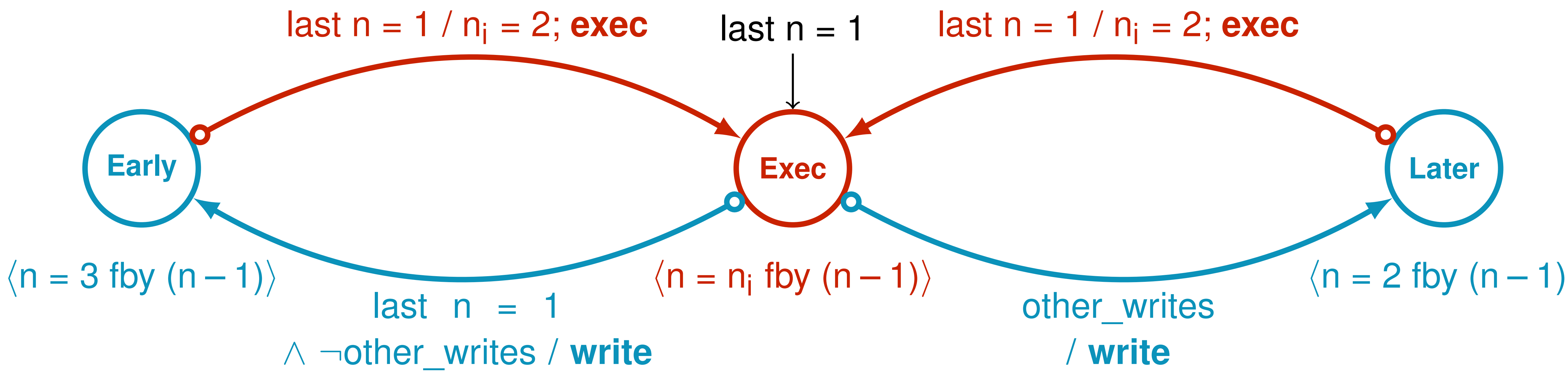

write
0/3
2
N_1
t
1
N_2
t
write
2
N_3
t


last n = 1 / n_i = 2; exec
last n = 1
last n = 1 / n_i = 2; exec
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / write
other_writes
/ write

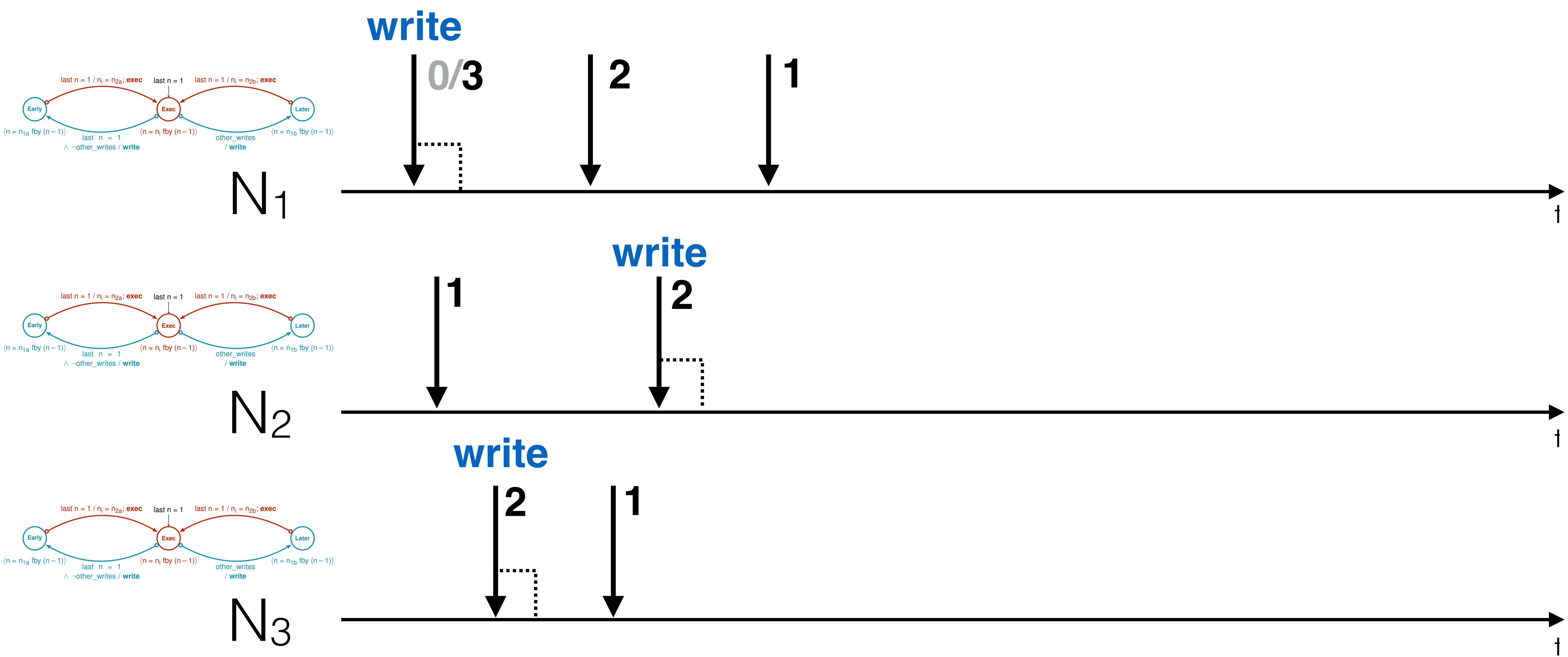

write
0/3
2
N1
t
1
N2
t
write
2
1
N3
t


last n = 1 / n$_i$ = 2; exec
last n = 1
last n = 1 / n$_i$ = 2; exec
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n$_i$ fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / write
other_writes
/ write

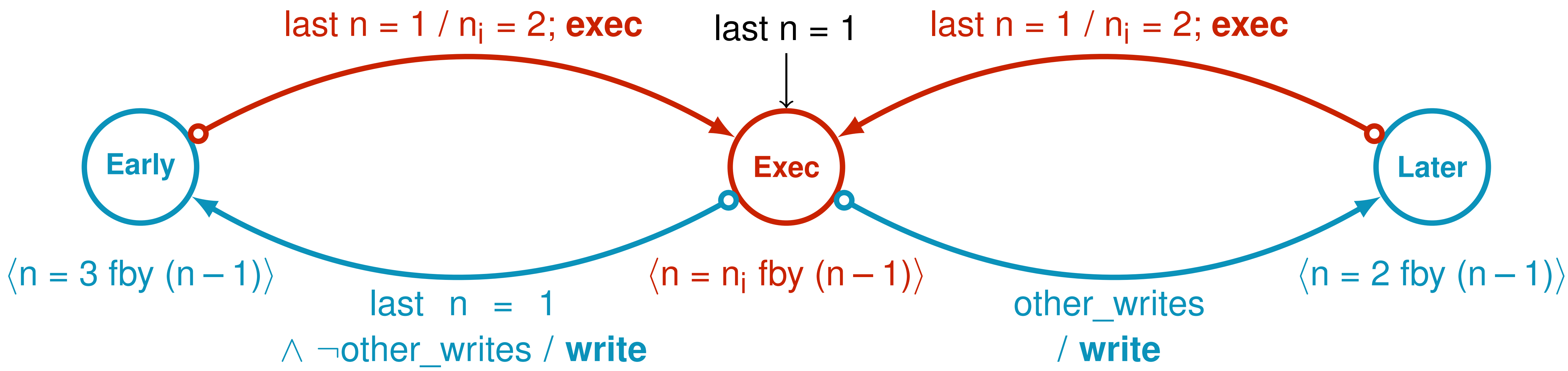

write
0/3
2
N$_1$
t
write
1
2
N$_2$
t
write
2
1
N$_3$
t



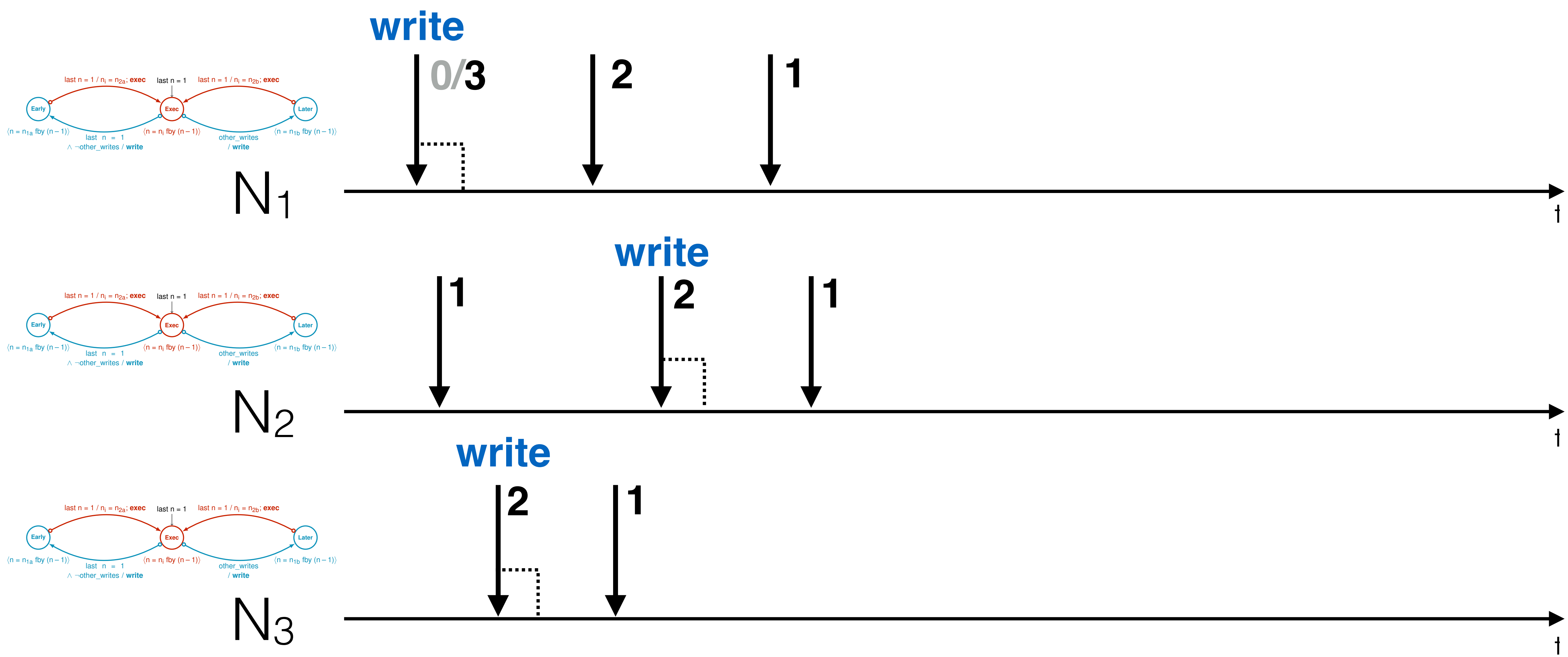
last n = 1 / n_i = 2; **exec**
last n = 1
last n = 1 / n_i = 2; **exec**
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1 ∧ ¬other_writes / **write**
other_writes / **write**



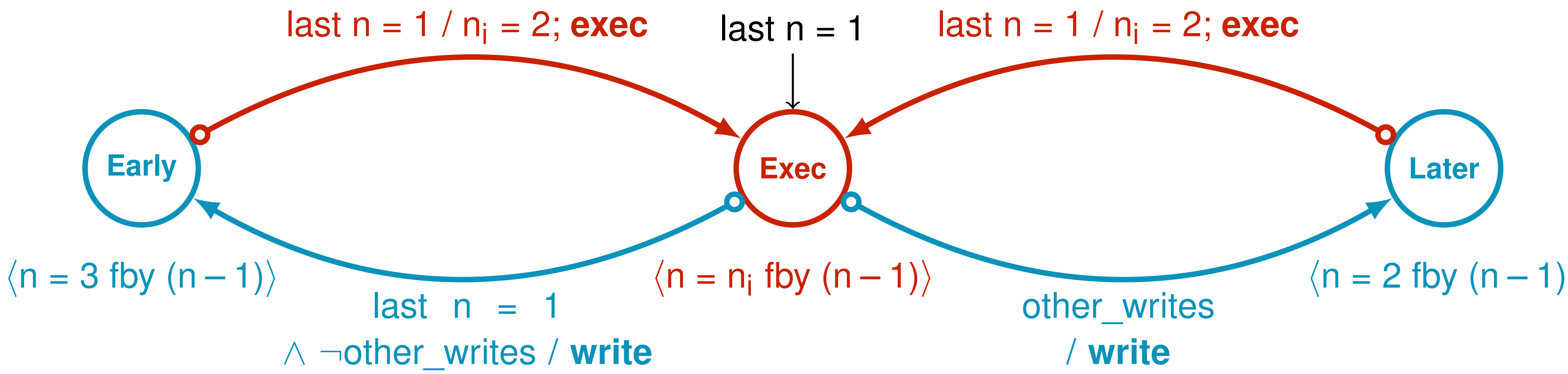
write
0/3
2
1
N_1
t
1
write
2
N_2
t
write
2
1
N_3
t

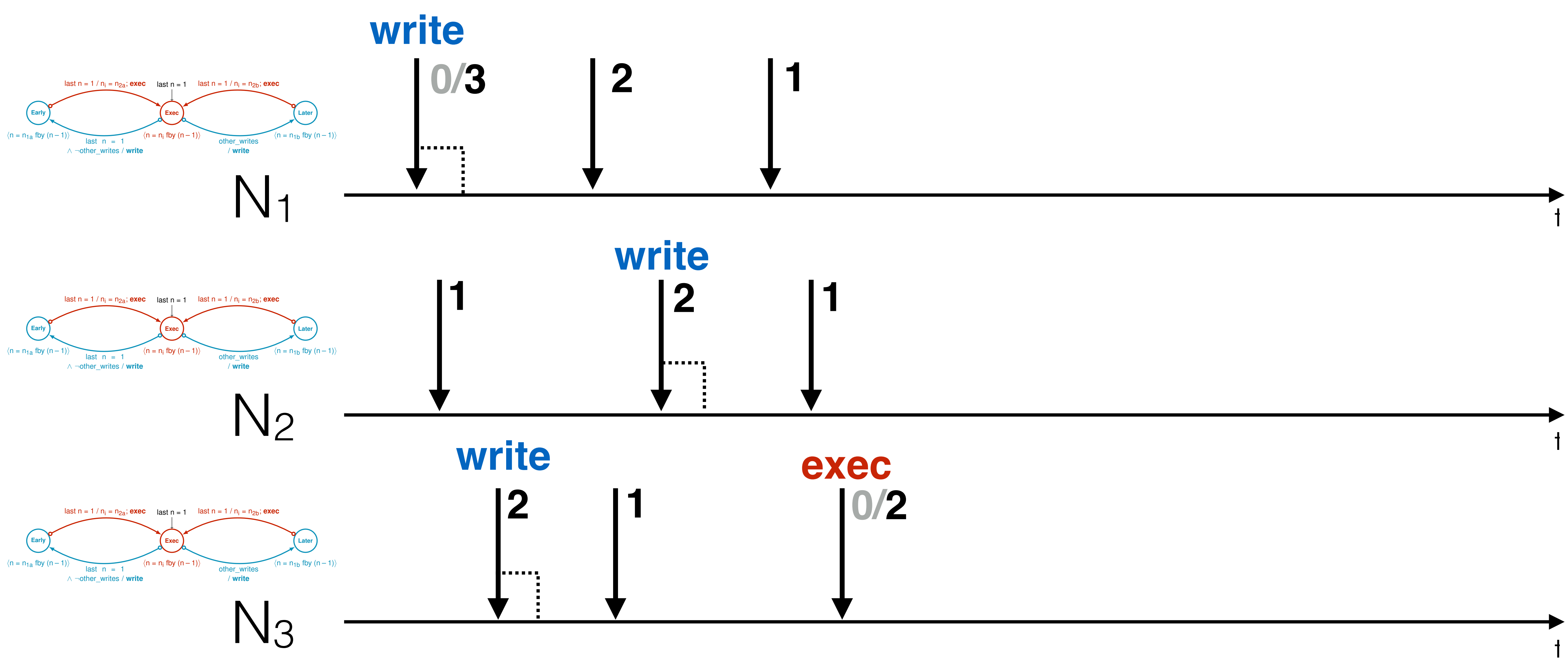


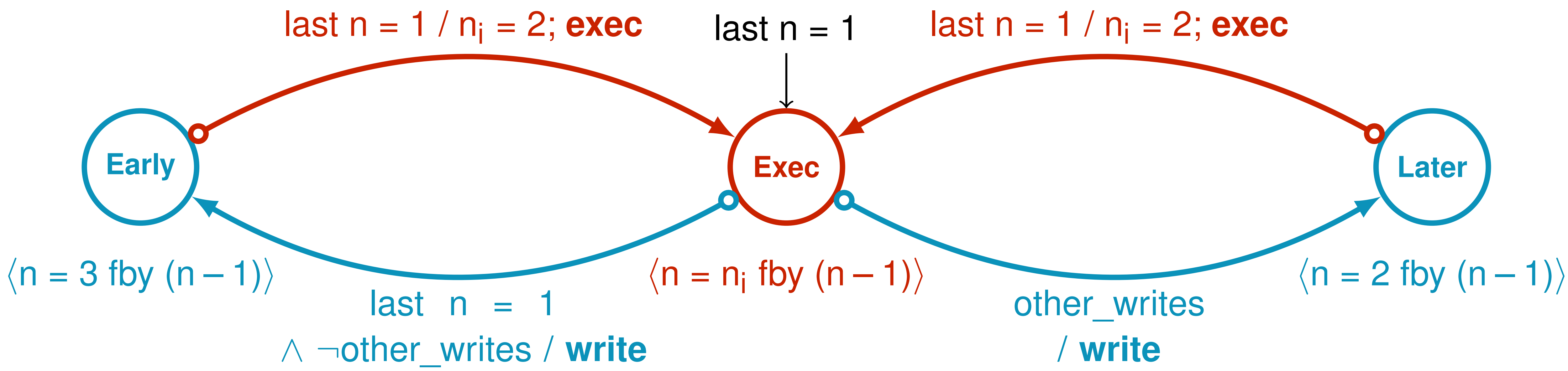
last n = 1 / n_i = 2; **exec**
last n = 1
last n = 1 / n_i = 2; **exec**
Early
Exec
Later
$\langle n = 3 \text{ fby } (n-1)\rangle$
$\langle n = n_i \text{ fby } (n-1)\rangle$
$\langle n = 2 \text{ fby } (n-1)\rangle$
last n = 1
∧ ¬other_writes / **write**
other_writes
/ **write**

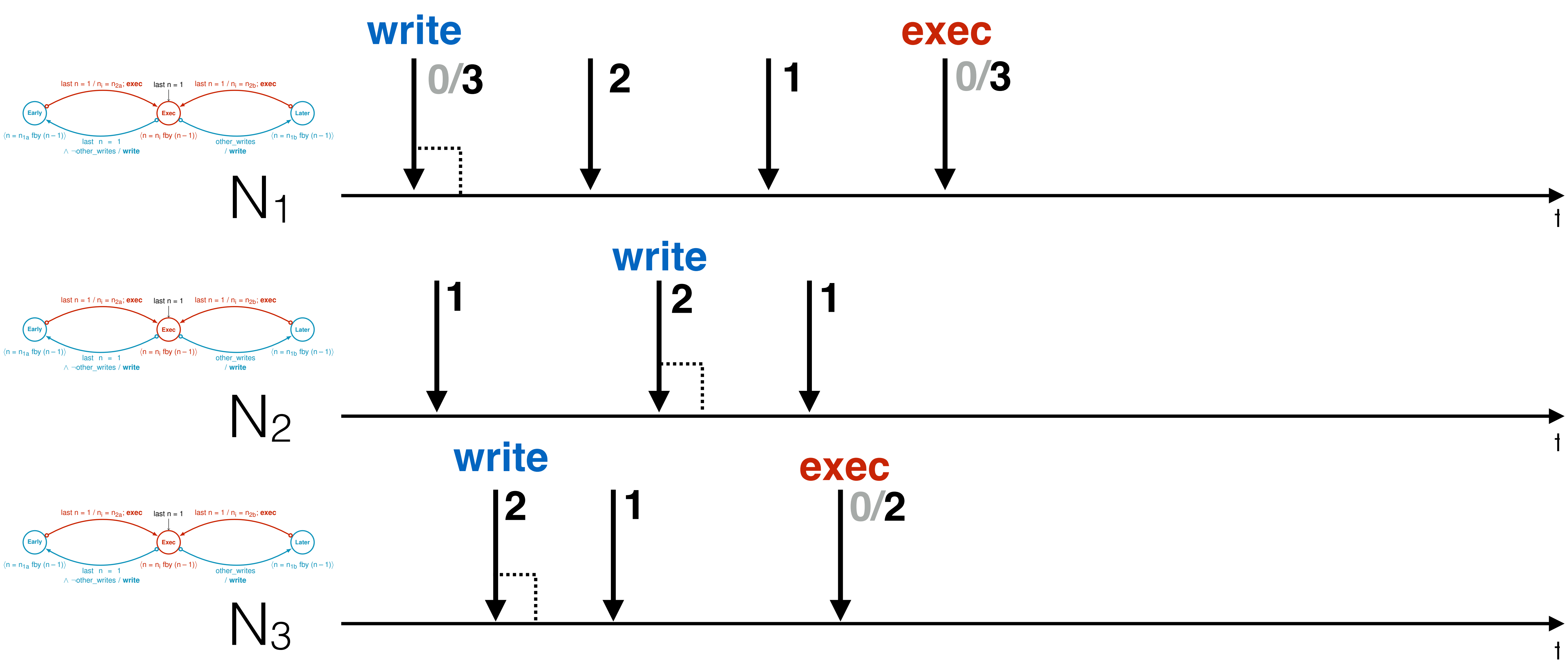


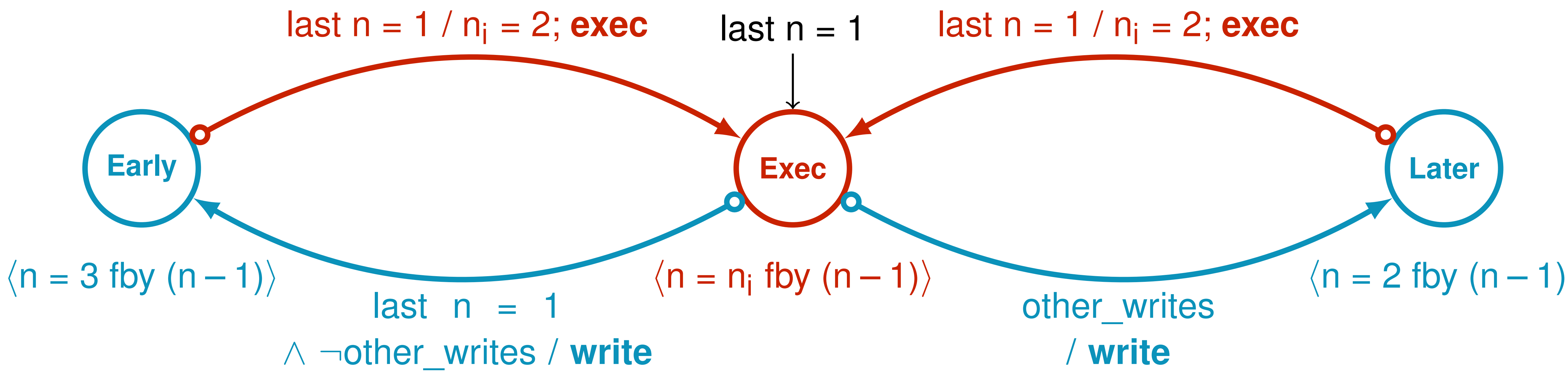
write
0/3
2
1
N_1
t
write
1
2
1
N_2
t
write
2
1
N_3
t


last n = 1 / n_i = 2; exec
last n = 1
last n = 1 / n_i = 2; exec
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / write
other_writes
/ write

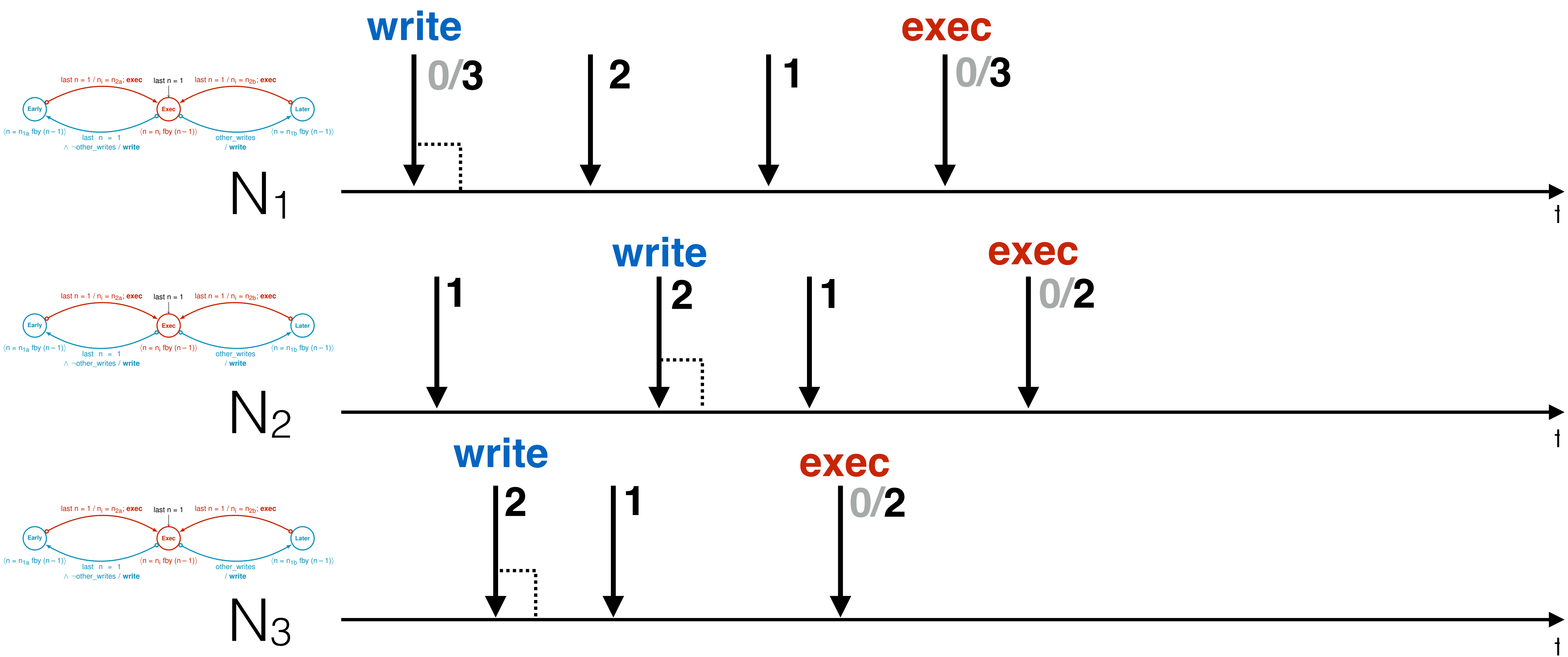

write
0/3
2
1
N1
t
write
1
2
1
N2
t
write
exec
2
1
0/2
N3
t


last n = 1 / n_i = 2; exec
last n = 1
last n = 1 / n_i = 2; exec
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / write
other_writes
/ write

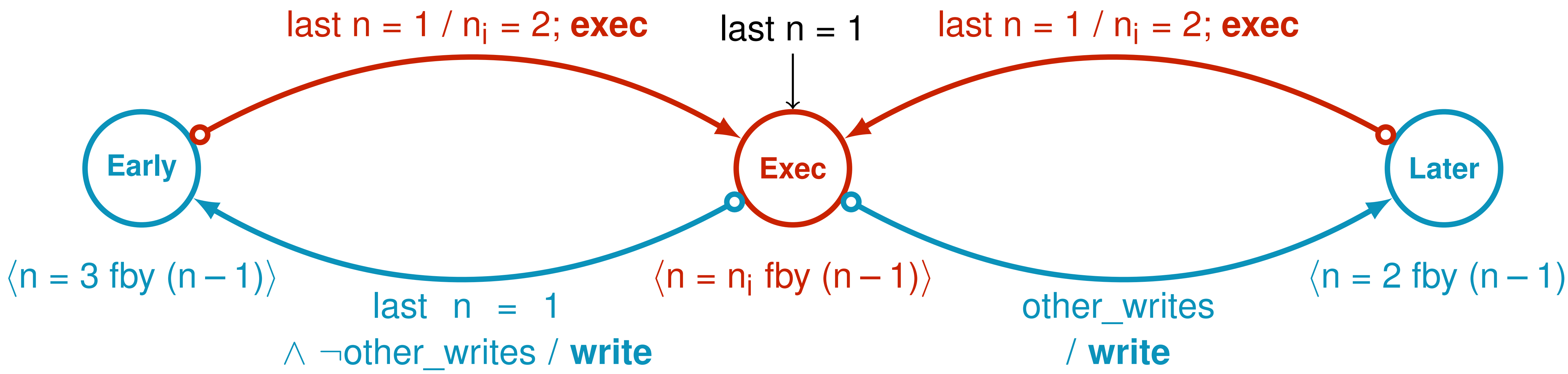

write
0/3
2
1
exec
0/3
N1
t
write
1
2
1
N2
t
write
2
1
exec
0/2
N3
t

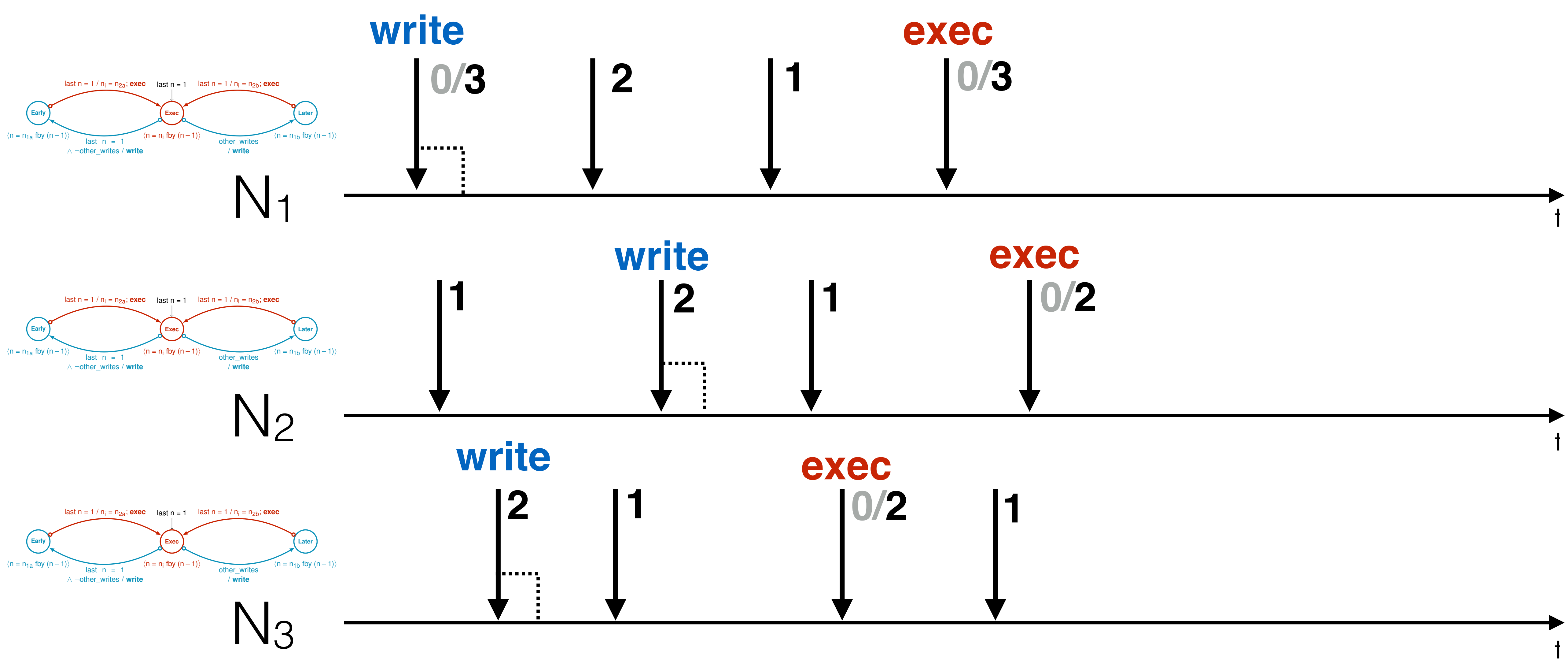


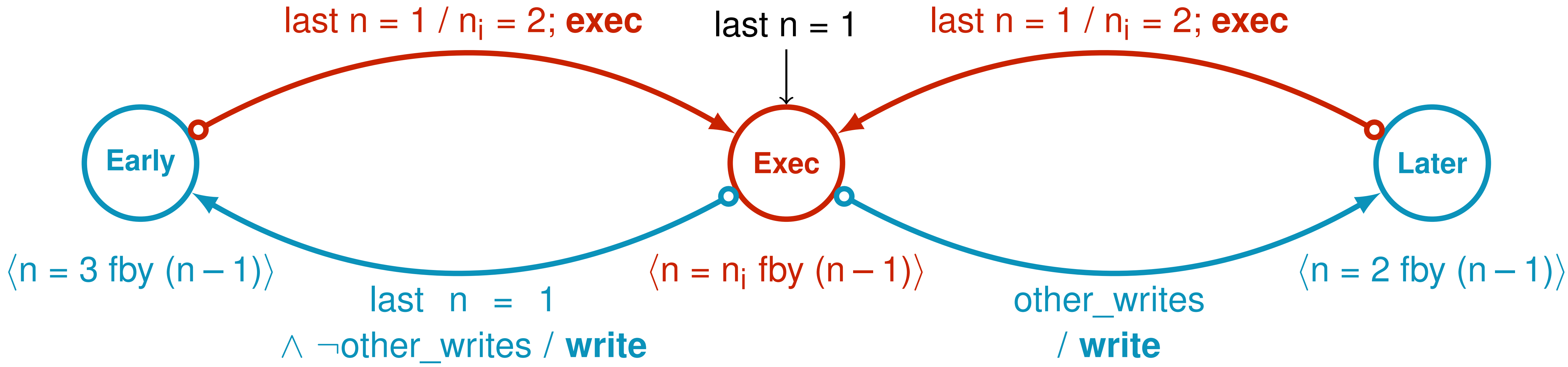
last n = 1 / n_i = 2; exec
last n = 1
last n = 1 / n_i = 2; exec
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1 ∧ ¬other_writes / write
other_writes / write

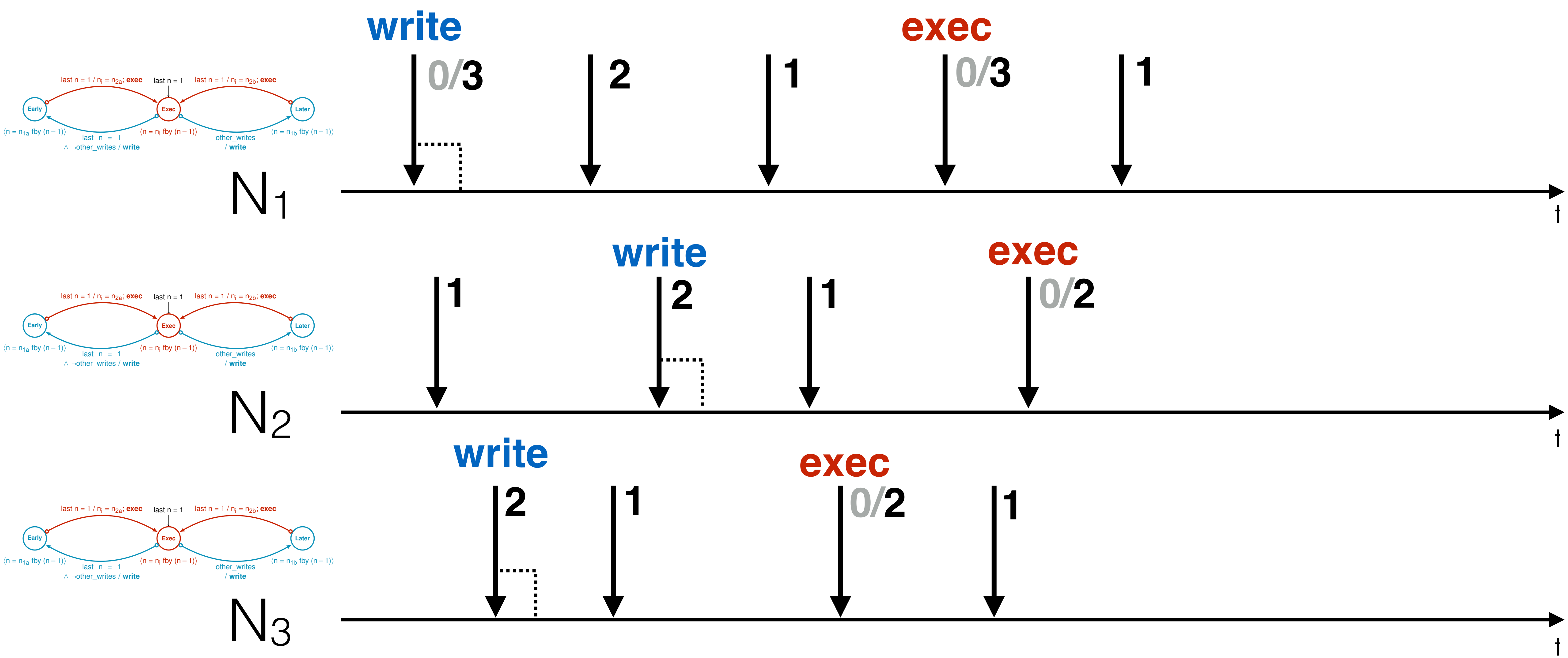


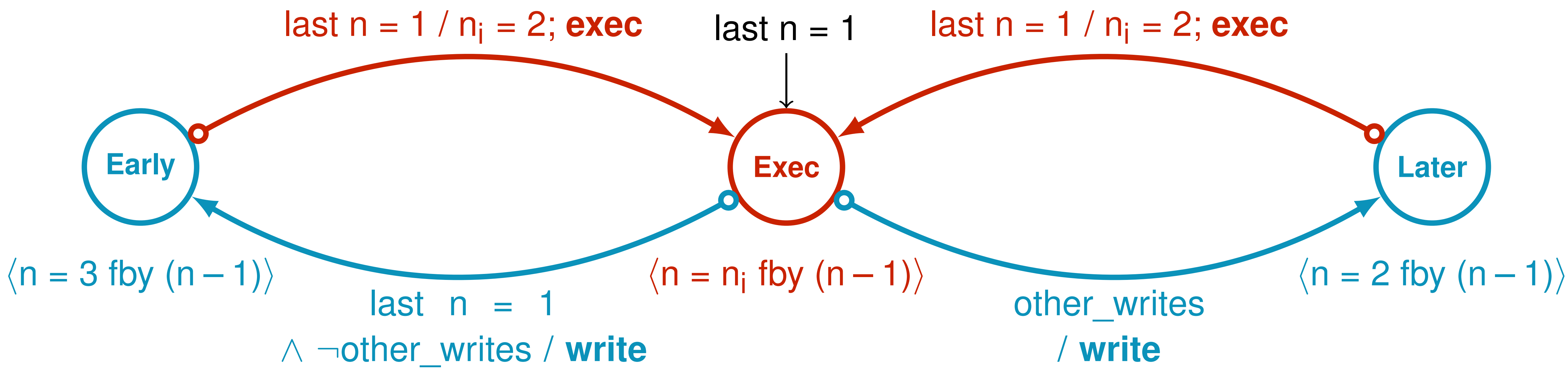
write
exec
0/3
2
1
0/3
N_1
t
write
exec
1
2
1
0/2
N_2
t
write
exec
2
1
0/2
N_3
t

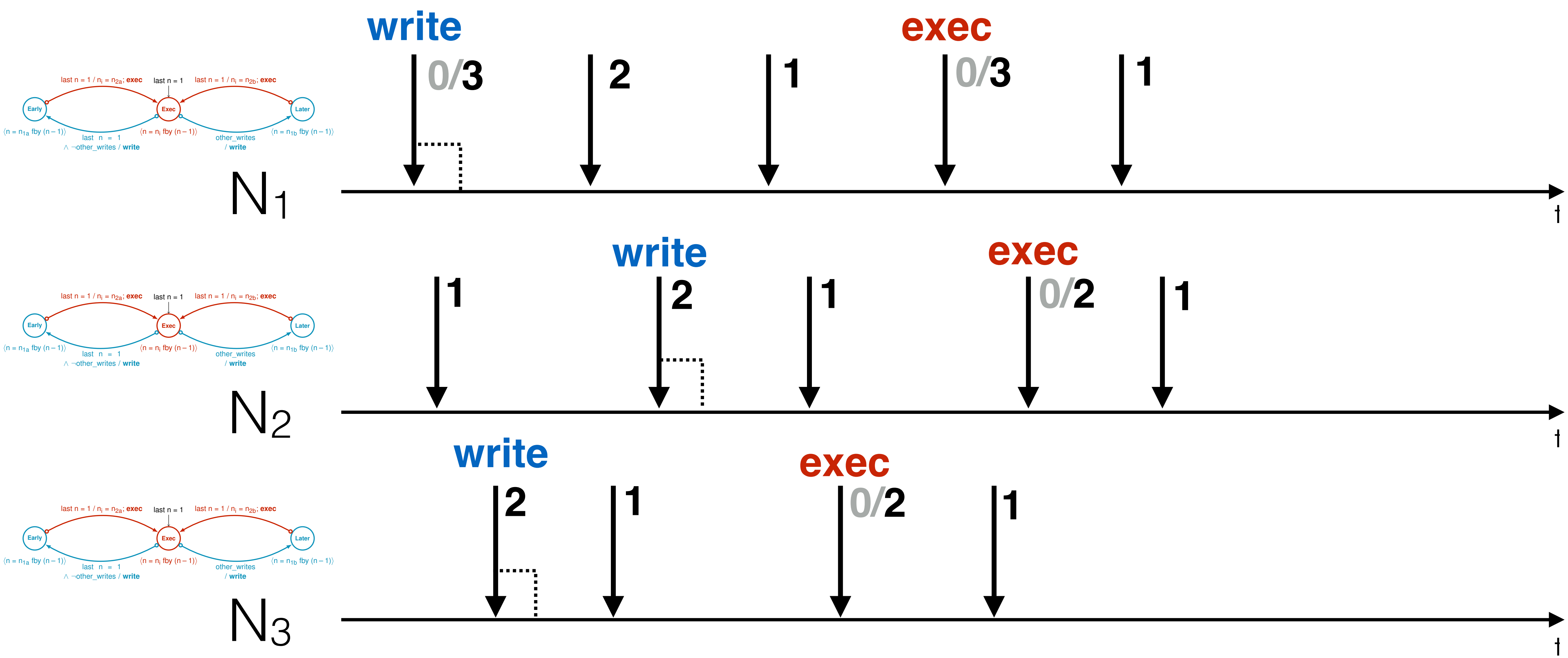

last n = 1 / n_i = 2; exec
last n = 1
last n = 1 / n_i = 2; exec
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / write
other_writes
/ write

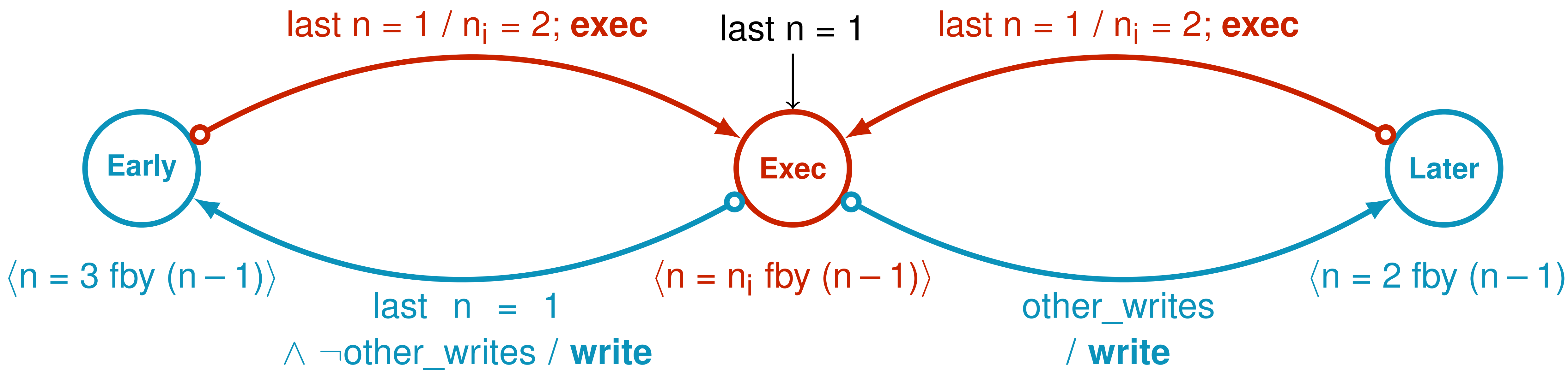

write
exec
0/3
2
1
0/3
N1
t
write
exec
1
2
1
0/2
N2
t
write
exec
2
1
0/2
1
N3
t

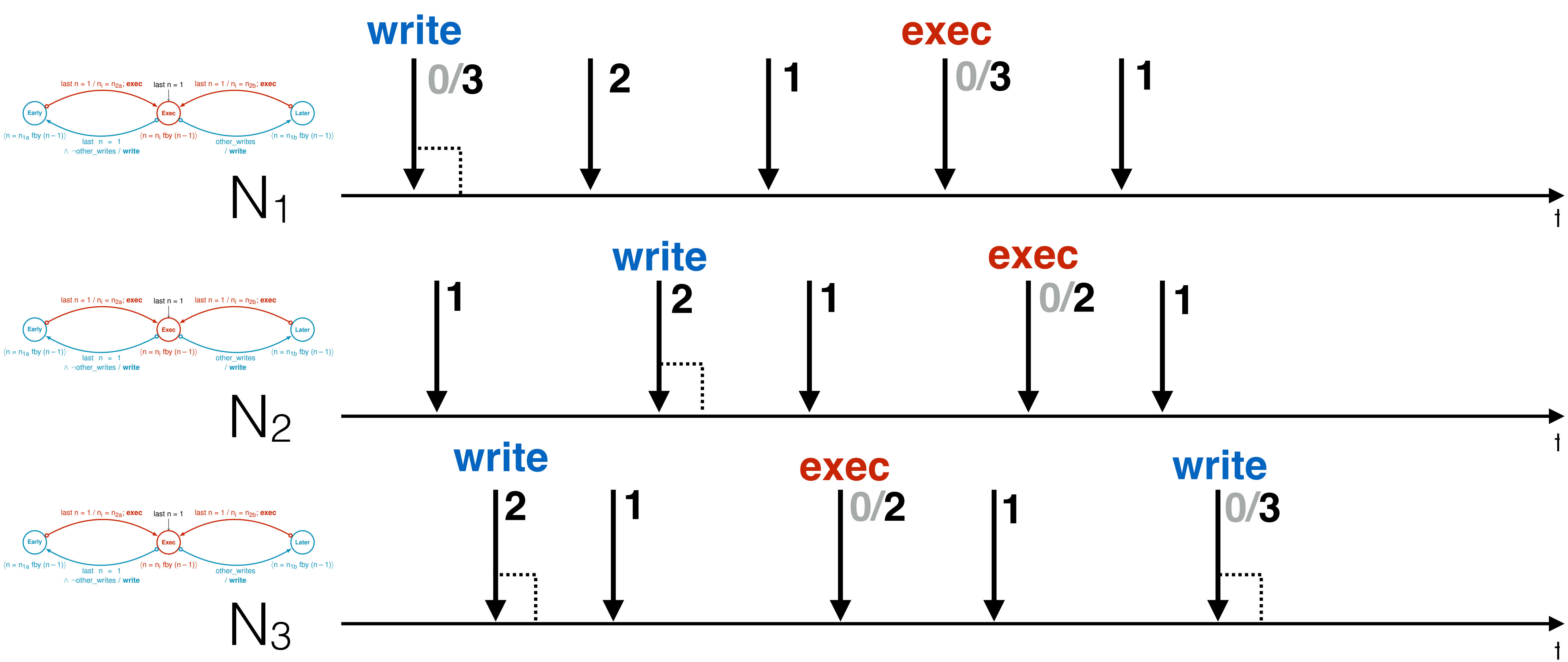

last n = 1 / n_i = 2; exec
last n = 1
last n = 1 / n_i = 2; exec
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / write
other_writes
/ write

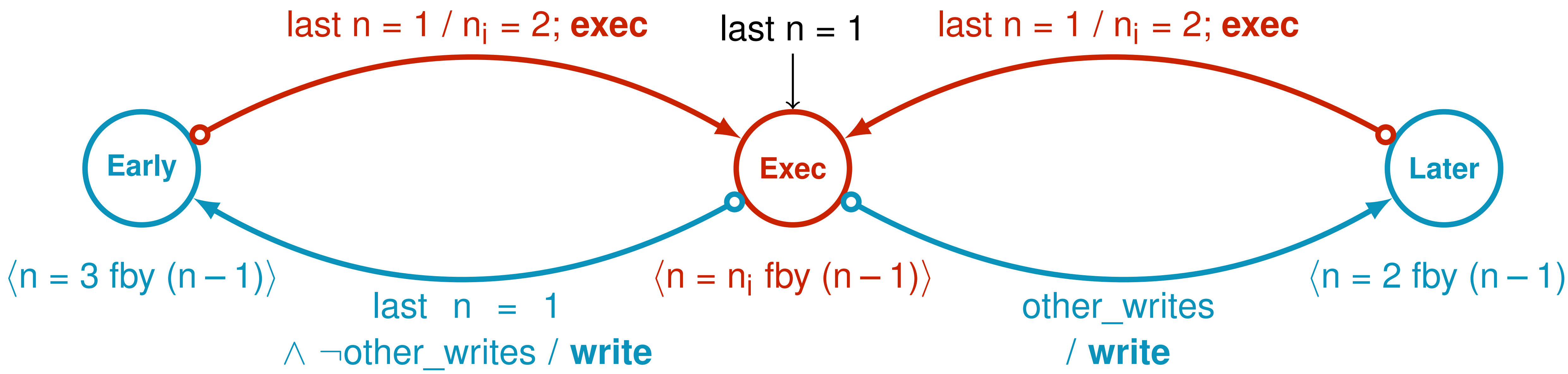

write
exec
0/3
2
1
0/3
1
N1
t
write
exec
1
2
1
0/2
N2
t
write
exec
2
1
0/2
1
N3
t


last n = 1 / n_i = 2; exec
last n = 1
last n = 1 / n_i = 2; exec
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / write
other_writes
/ write

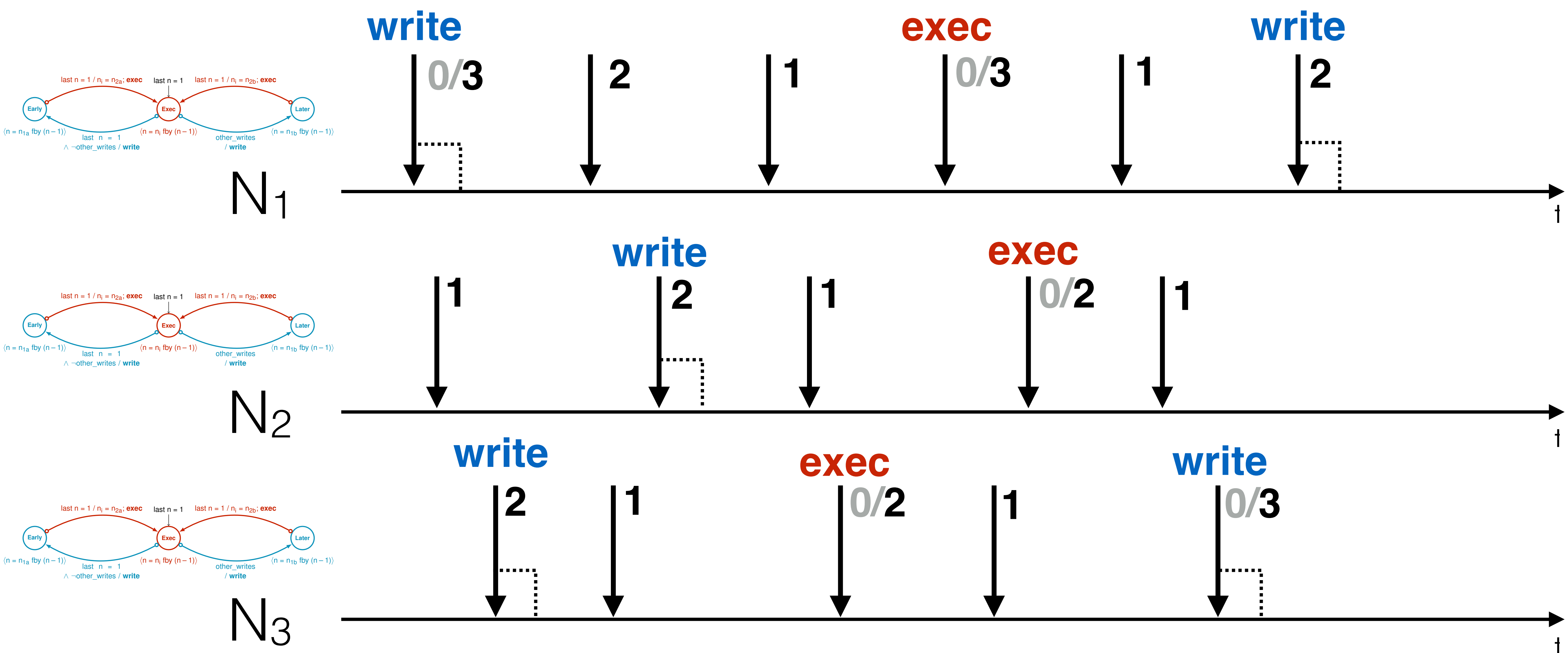

write
exec
0/3
2
1
0/3
1
N1
t
write
exec
1
2
1
0/2
1
N2
t
write
exec
2
1
0/2
1
N3
t


last n = 1 / n_i = 2; exec
last n = 1
last n = 1 / n_i = 2; exec
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / write
other_writes
/ write

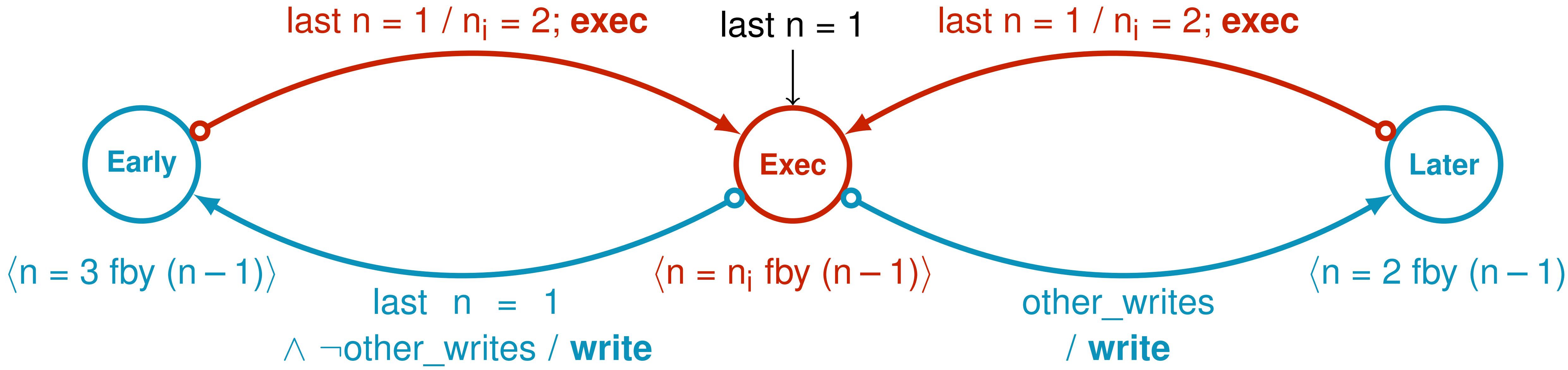

write
exec
0/3
2
1
0/3
1
N_1
t
write
exec
1
2
1
0/2
1
N_2
t
write
exec
write
2
1
0/2
1
0/3
N_3
t

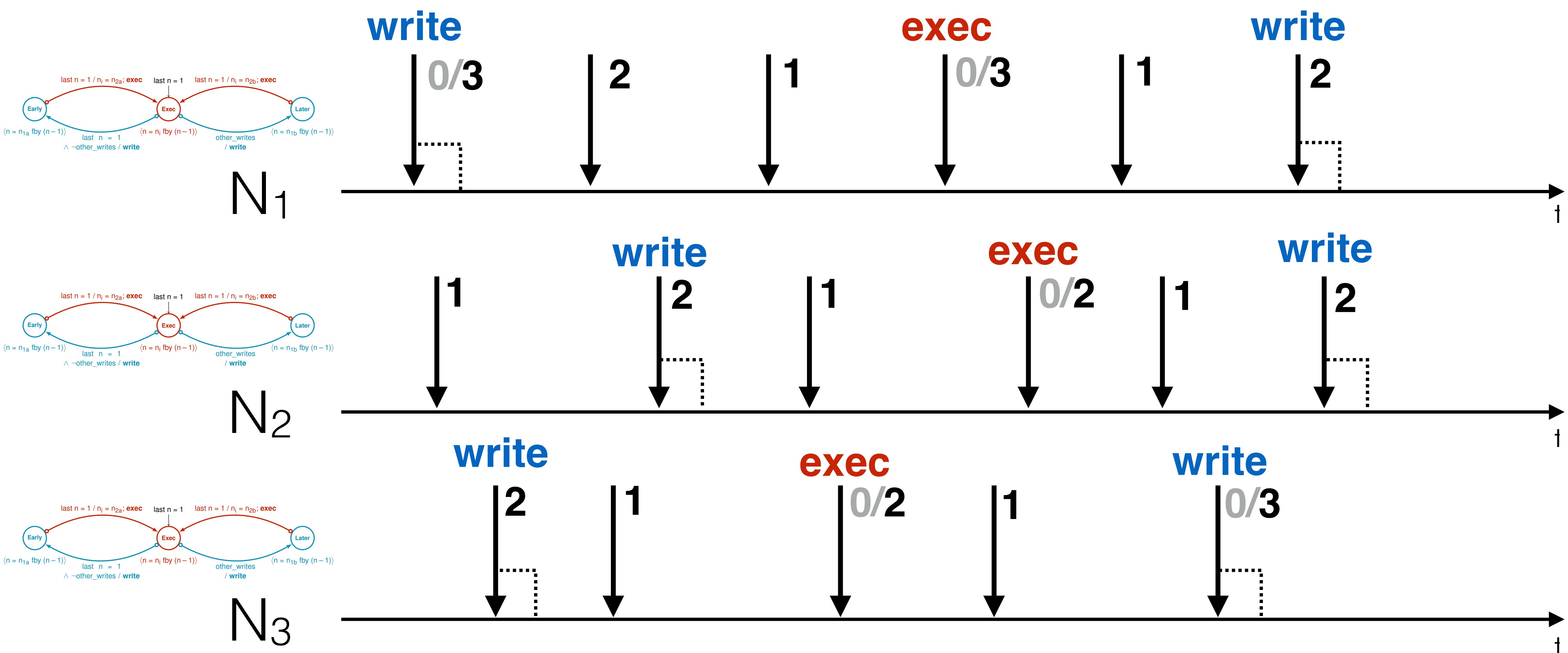


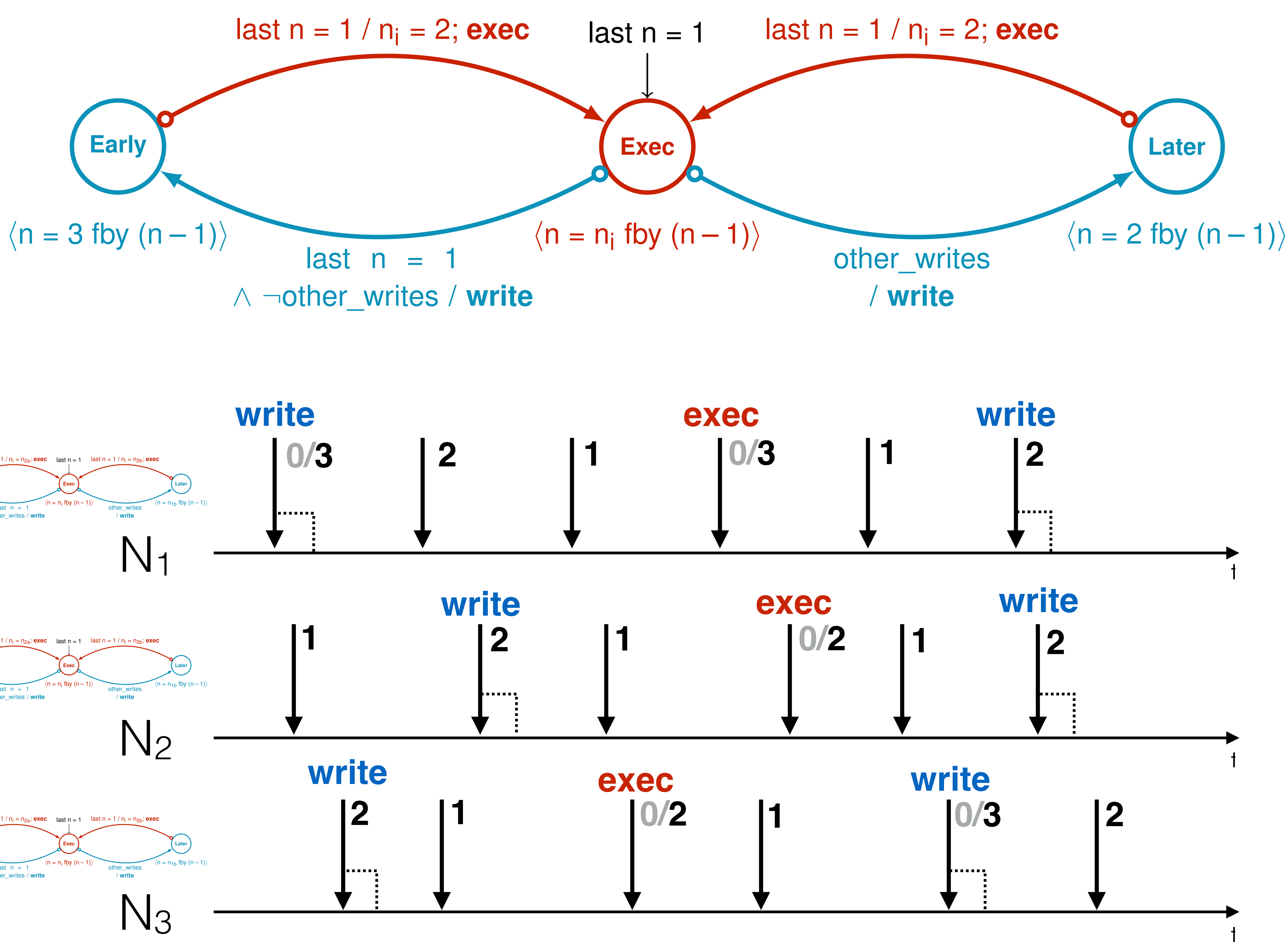
last n = 1 / n_i = 2; exec
last n = 1
last n = 1 / n_i = 2; exec
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / write
other_writes
/ write



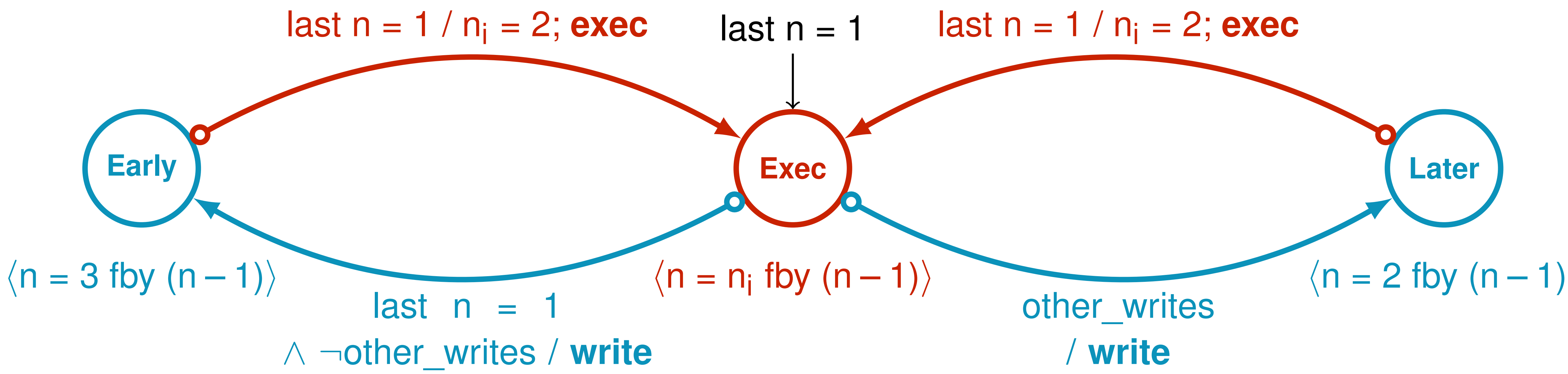
write
exec
write
0/3
2
1
0/3
1
2
N1
t
write
exec
1
2
1
0/2
1
N2
t
write
exec
write
2
1
0/2
1
0/3
N3
t

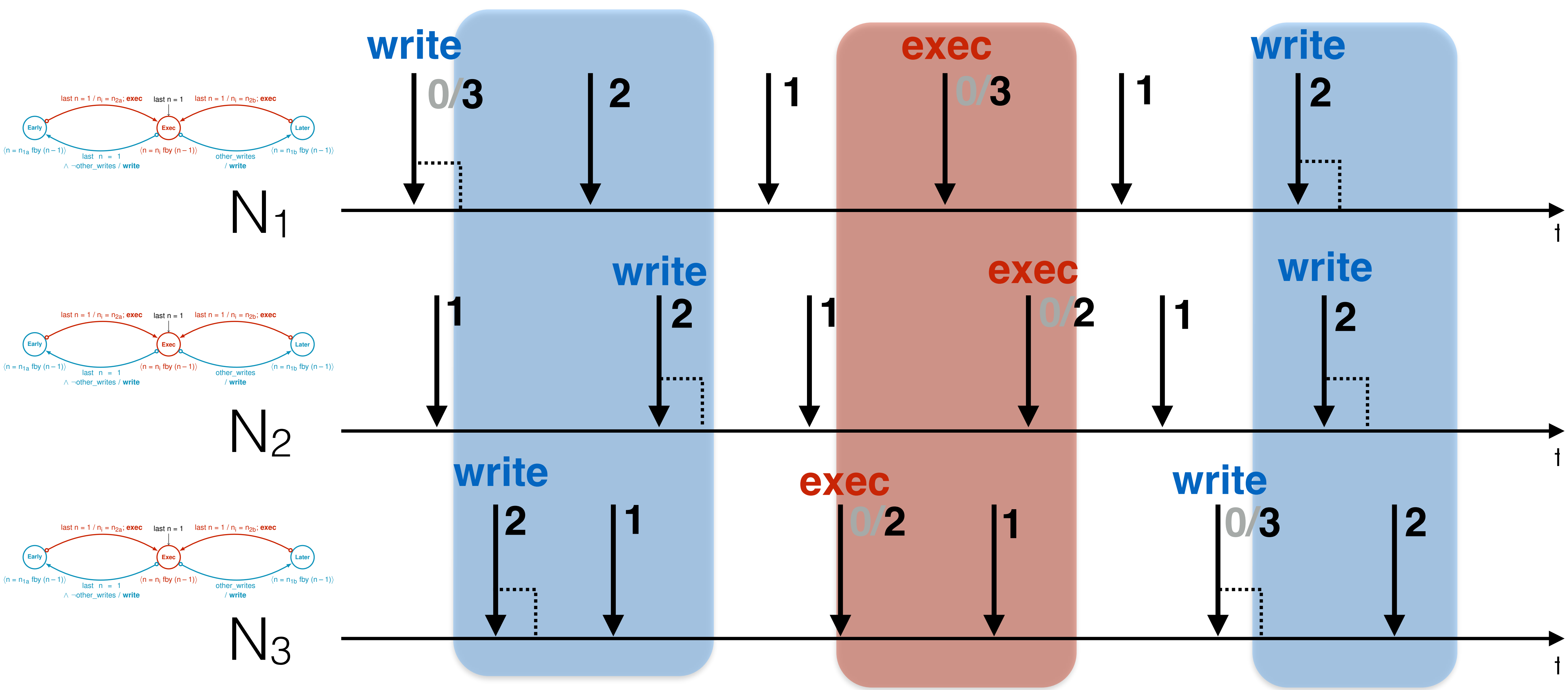

last n = 1 / n_i = 2; exec
last n = 1
last n = 1 / n_i = 2; exec
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / write
other_writes
/ write


N1
write
0/3
2
1
exec
0/3
1
write
2
t
N2
1
write
2
1
exec
0/2
1
write
2
t
N3
write
2
1
exec
0/2
1
write
0/3
t



last n = 1 / n_i = 2; exec
last n = 1
last n = 1 / n_i = 2; exec
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / write
other_writes
/ write
write
exec
write
0/3
2
1
0/3
1
2
N1
t
write
exec
write
1
2
1
0/2
1
2
N2
t
write
exec
write
2
1
0/2
1
0/3
2
N3
t


last n = 1 / n_i = 2; exec
last n = 1
last n = 1 / n_i = 2; exec
Early
Exec
Later
⟨n = 3 fby (n – 1)⟩
⟨n = n_i fby (n – 1)⟩
⟨n = 2 fby (n – 1)⟩
last n = 1
∧ ¬other_writes / write
other_writes
/ write


write
0/3
2
1
exec
0/3
1
write
2
N1
t
write
1
2
1
exec
0/2
1
write
2
N2
t
write
2
1
exec
0/2
1
write
0/3
2
N3
t

Demo in Zélus

Comparison

	Time-Based	Back-Pressure
Flexibility	✗	✓
Robustness	✓	✗

Blend the two approaches [Benveniste et al. 2010]

Future Work

(i.e., my thesis)

- Zélus: language for mixing real- and discrete-time behaviors; LTTA is but one example
- We think the idea of tolerance (e.g., jitter) is important in such languages
- Continue to investigate the link between discrete- and real-time semantics (e.g., effect of skips in LTTA)


Queêêêestions?