Loosely Time-Triggered Architectures: Improvements and Comparisons

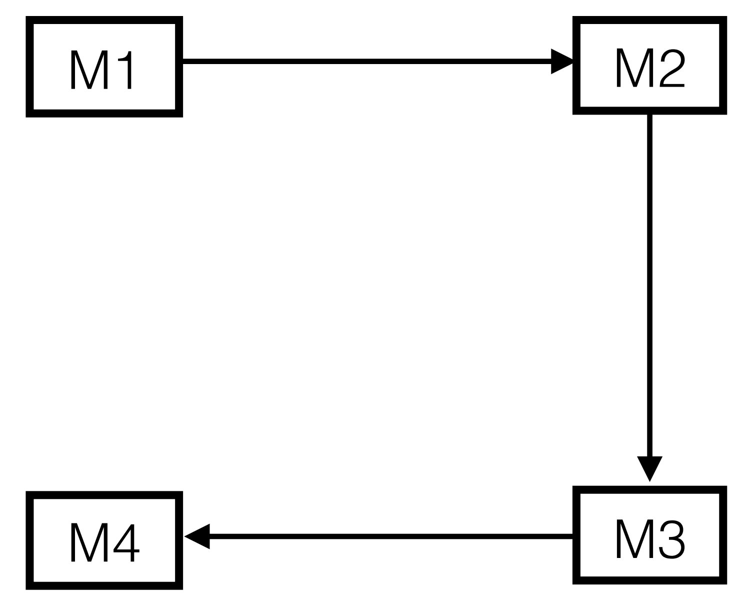
Guillaume Baudart
Albert Benveniste
Timothy Bourke

EMSOFT'15 **Amsterdam, 06-10-2015**

Synchronous Applications…

Composition of Communicating Mealy Machines

- Each machine is characterized by
  - an initial state S_{init}
  - a transition function $F \;:\; \mathcal{S} \times \mathcal{V}^I \rightarrow \mathcal{S} \times \mathcal{V}^O$
- Machines communicate through unit delays.



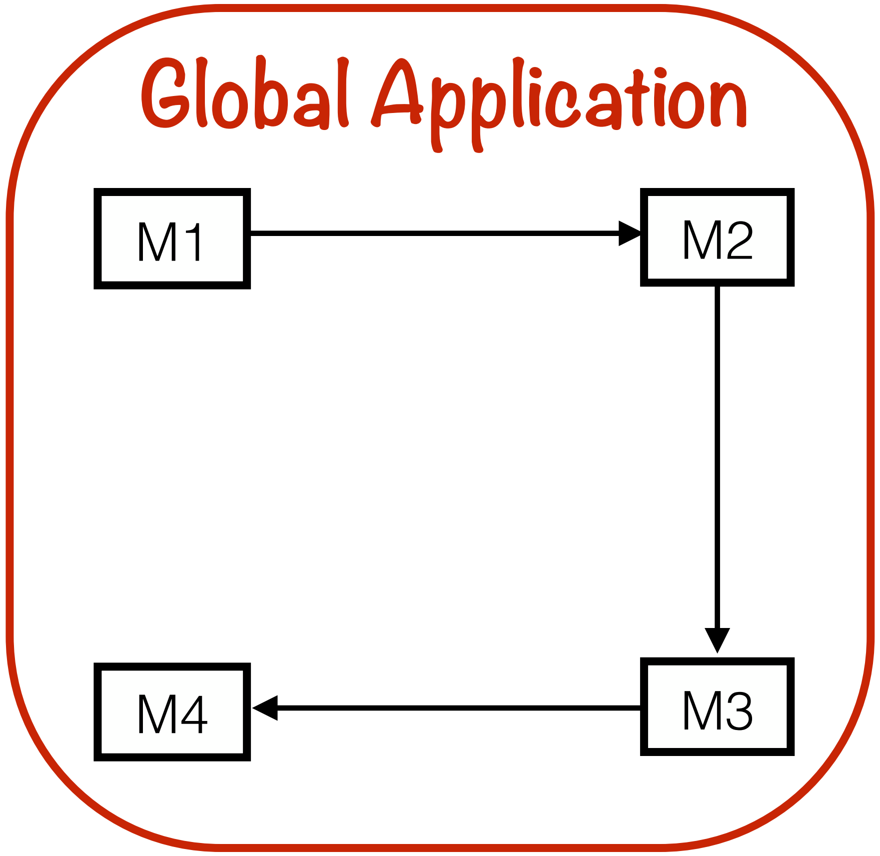
Synchronous Applications…

Composition of Communicating Mealy Machines

- Each machine is characterized by
  - an initial state S_{init}
  - a transition function

    $F \ : \ \mathcal{S} \times \mathcal{V}^I \rightarrow \mathcal{S} \times \mathcal{V}^O$

- Machines communicate through unit delays.



Composition forms a global synchronous application.

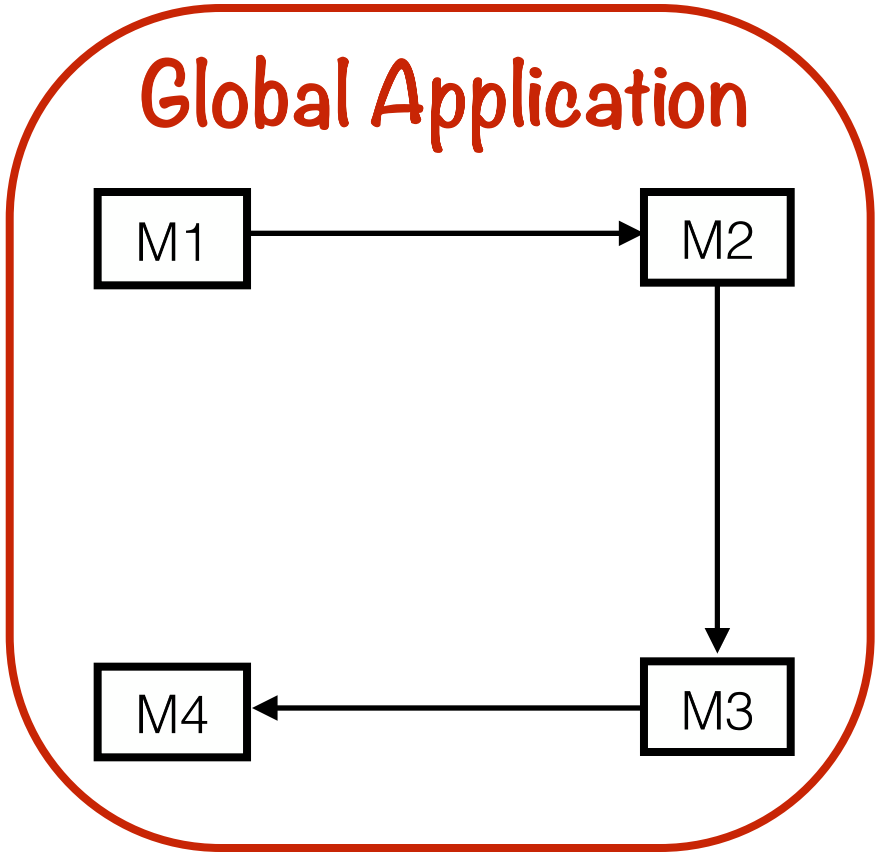
Synchronous Applications...

Composition of Communicating Mealy Machines

- Each machine is characterized by
  - an initial state S_{init}
  - a transition function

    $F \;:\; \mathcal{S} \times \mathcal{V}^I \rightarrow \mathcal{S} \times \mathcal{V}^O$

- Machines communicate through unit delays.



Composition forms a global synchronous application.
Semantics: Sequence of values on each variable

...on Quasi-Periodic Architecture

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

$$0 < T_{\min} \leq T^n \leq T_{\max} \quad \text{or}$$

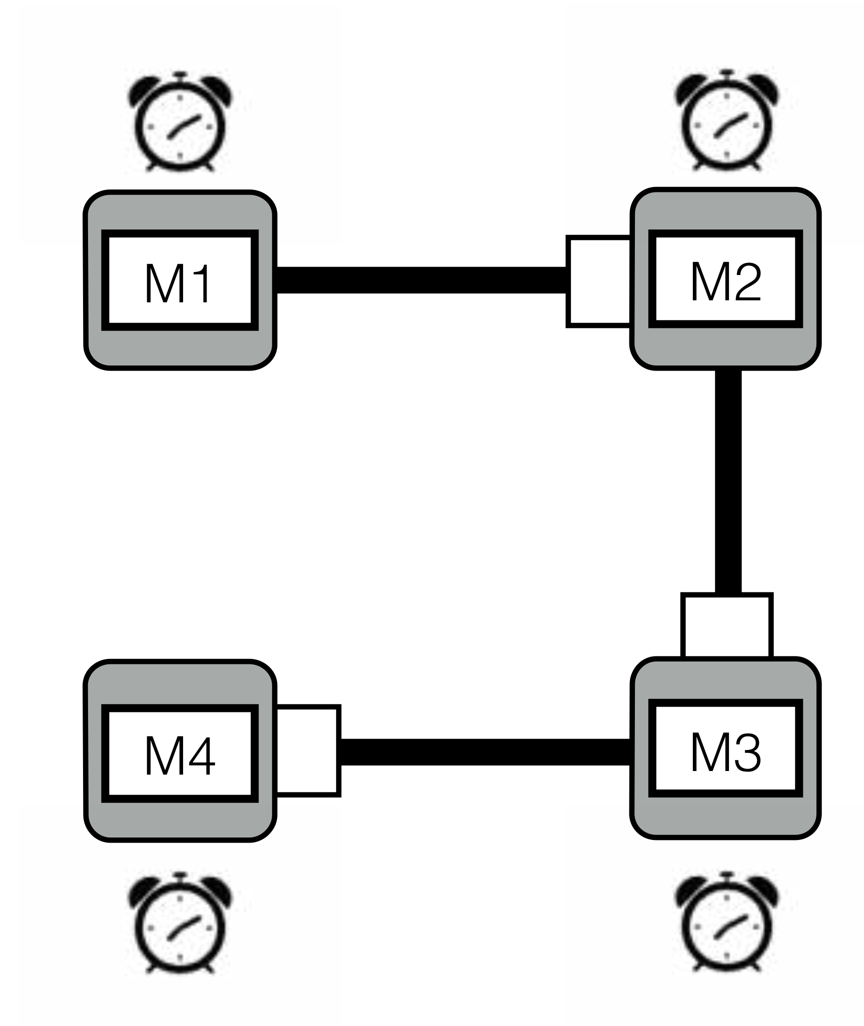
$$T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon$$

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

- Buffered communication without message inversion or loss

- Bounded communication delay

$$\tau_{\min} \leq \tau \leq \tau_{\max}$$



"Cooking Book" [Caspi 2000]

...on Quasi-Periodic Architecture

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

$$0 < T_{\min} \leq T^n \leq T_{\max} \quad \text{or}$$

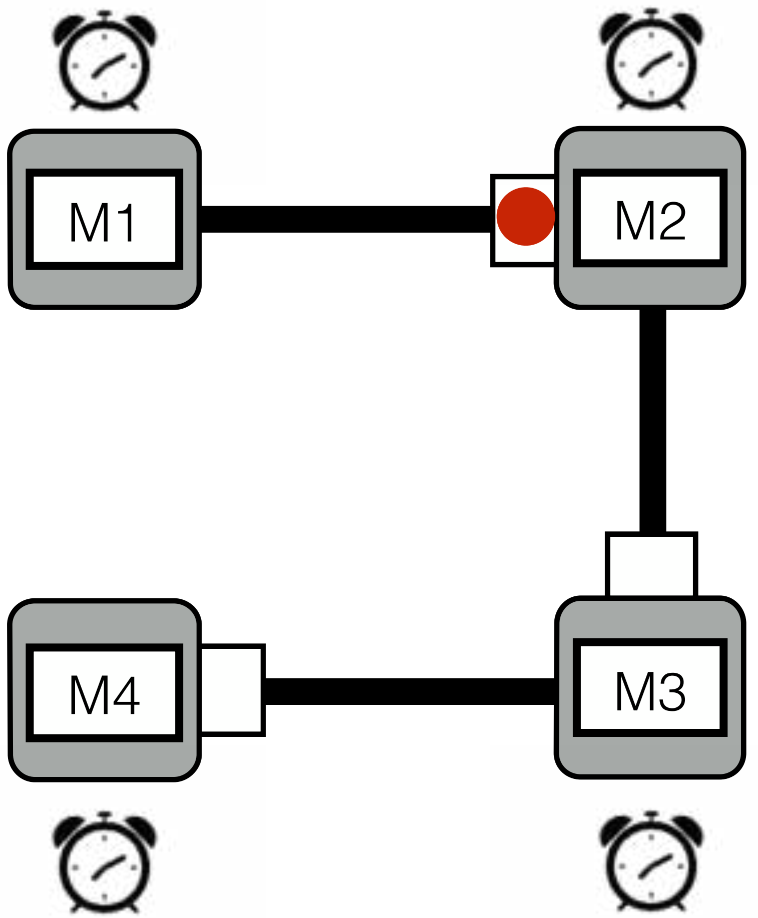
$$T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon$$

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

- Buffered communication without message inversion or loss

- Bounded communication delay

$$\tau_{\min} \leq \tau \leq \tau_{\max}$$



“Cooking Book” [Caspi 2000]

...on Quasi-Periodic Architecture

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

$$0 < T_{\min} \leq T^n \leq T_{\max} \quad \text{or}$$

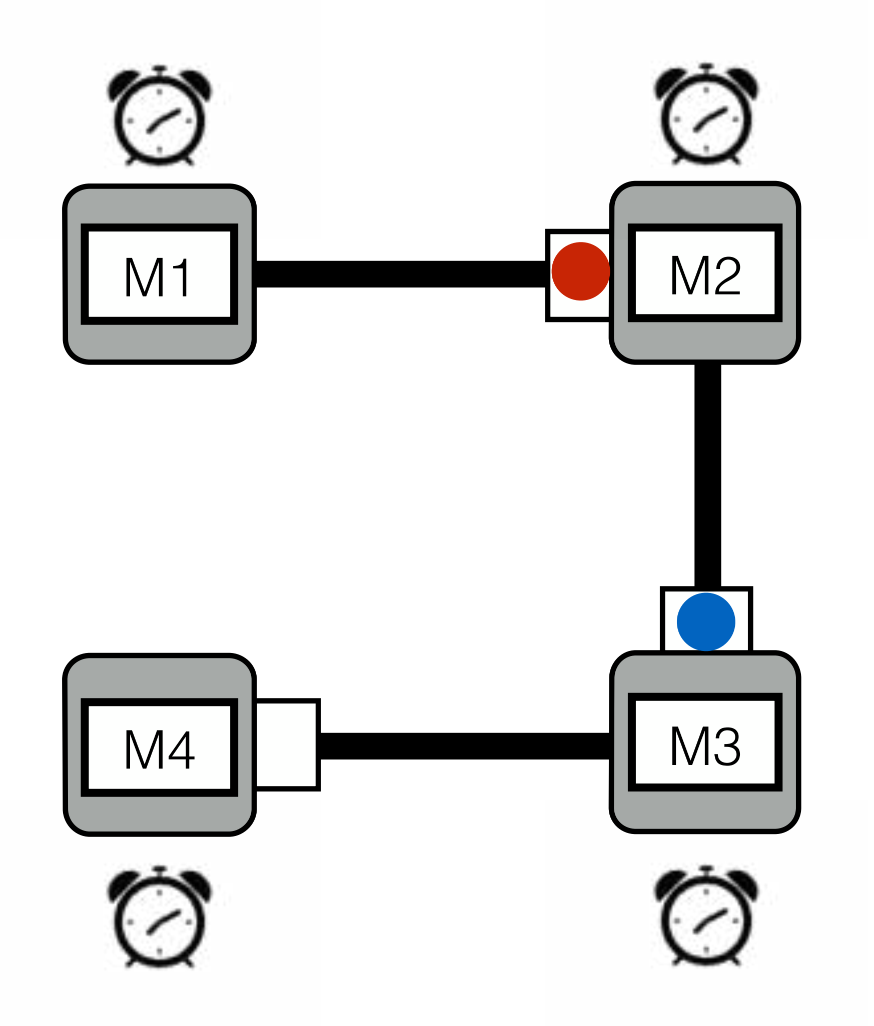
$$T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon$$

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

- Buffered communication without message inversion or loss

- Bounded communication delay

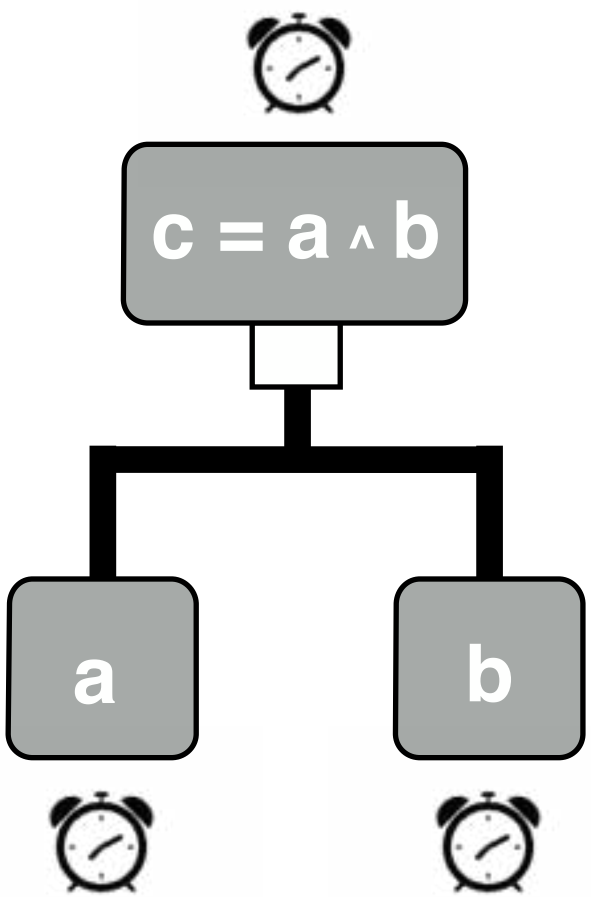
$$\tau_{\min} \leq \tau \leq \tau_{\max}$$

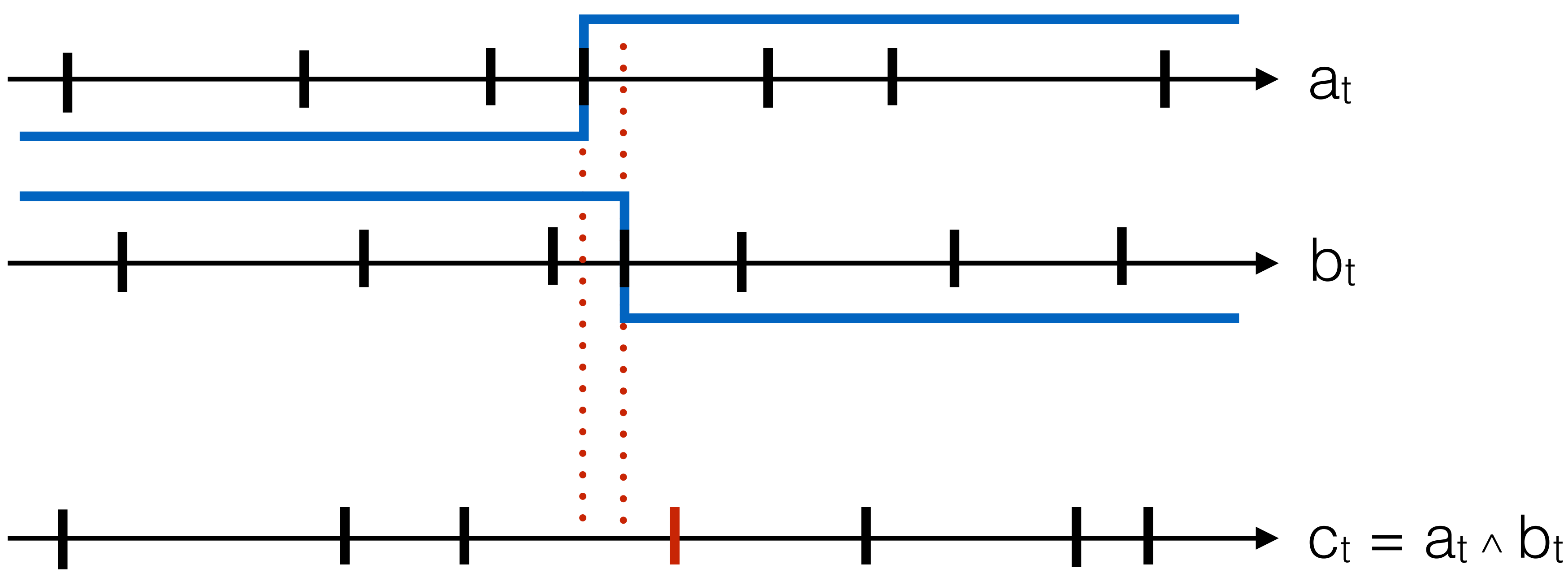


"Cooking Book" [Caspi 2000]

Sampling Artifacts

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

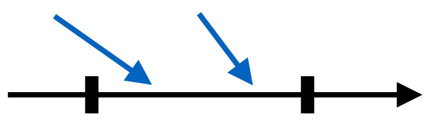




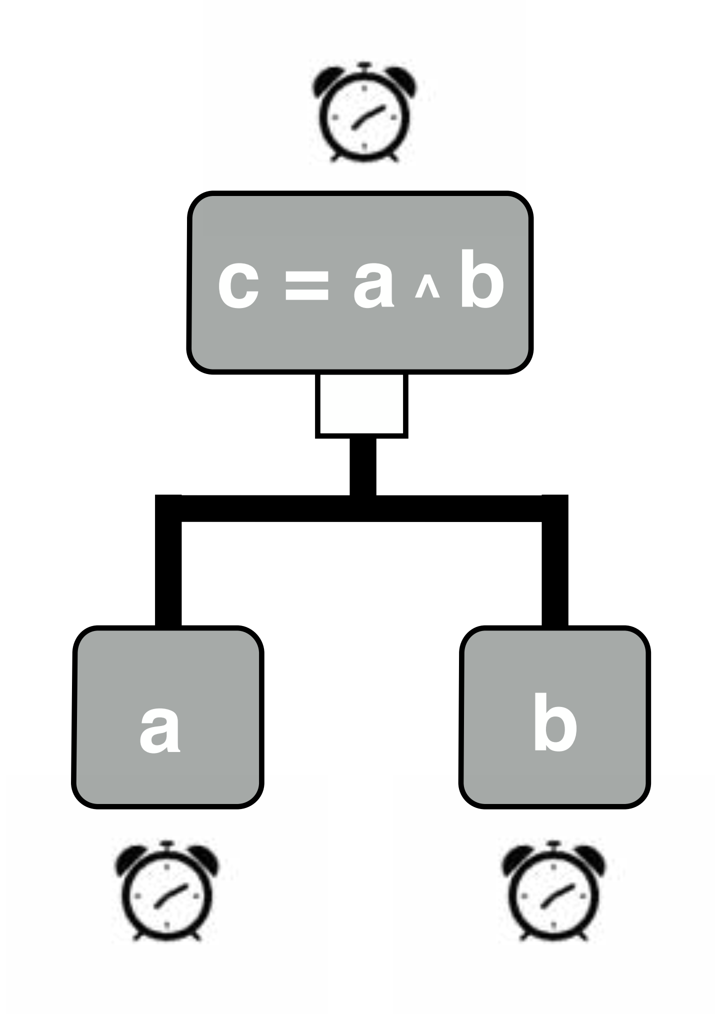
Example from [Caspi 2000]

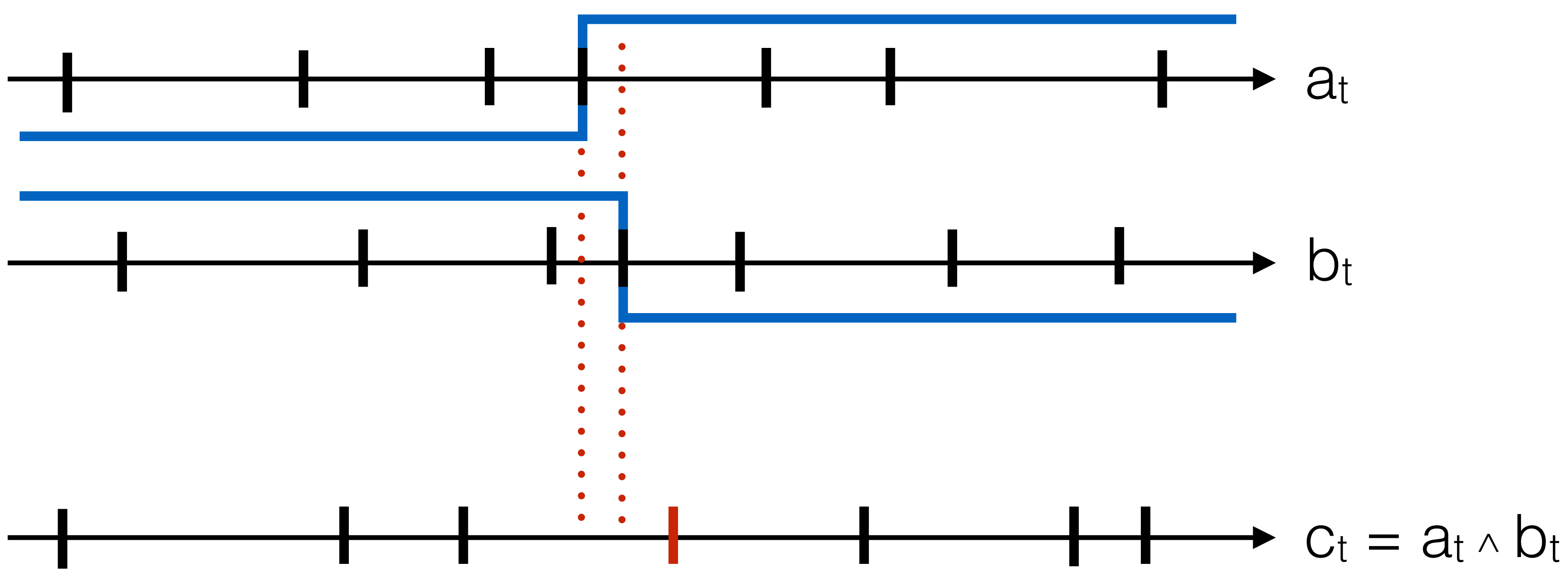
Sampling Artifacts

- **Overwriting:** Loss of values



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

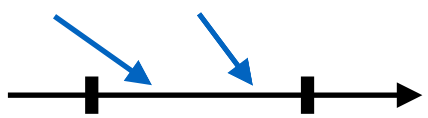


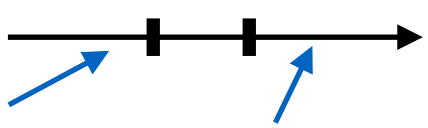


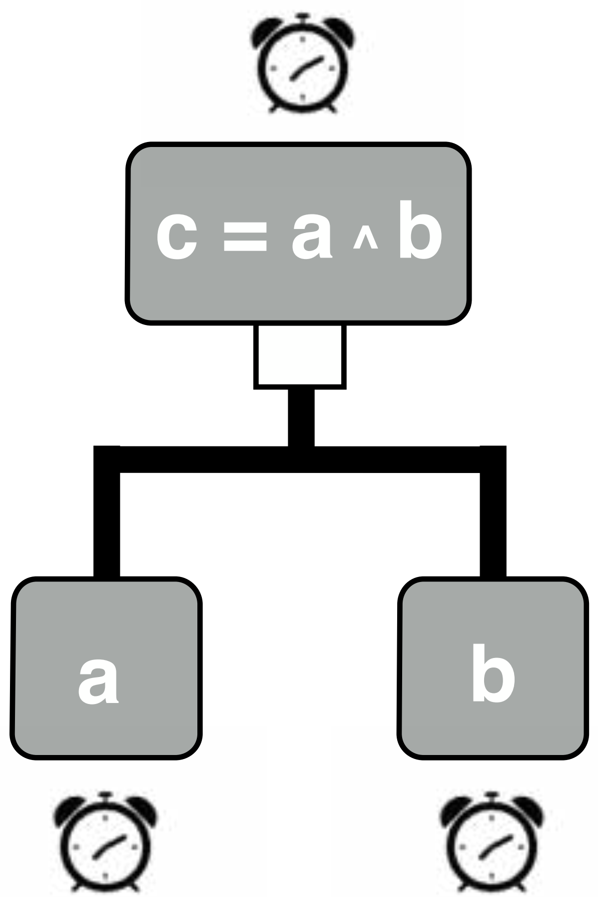
Example from [Caspi 2000]

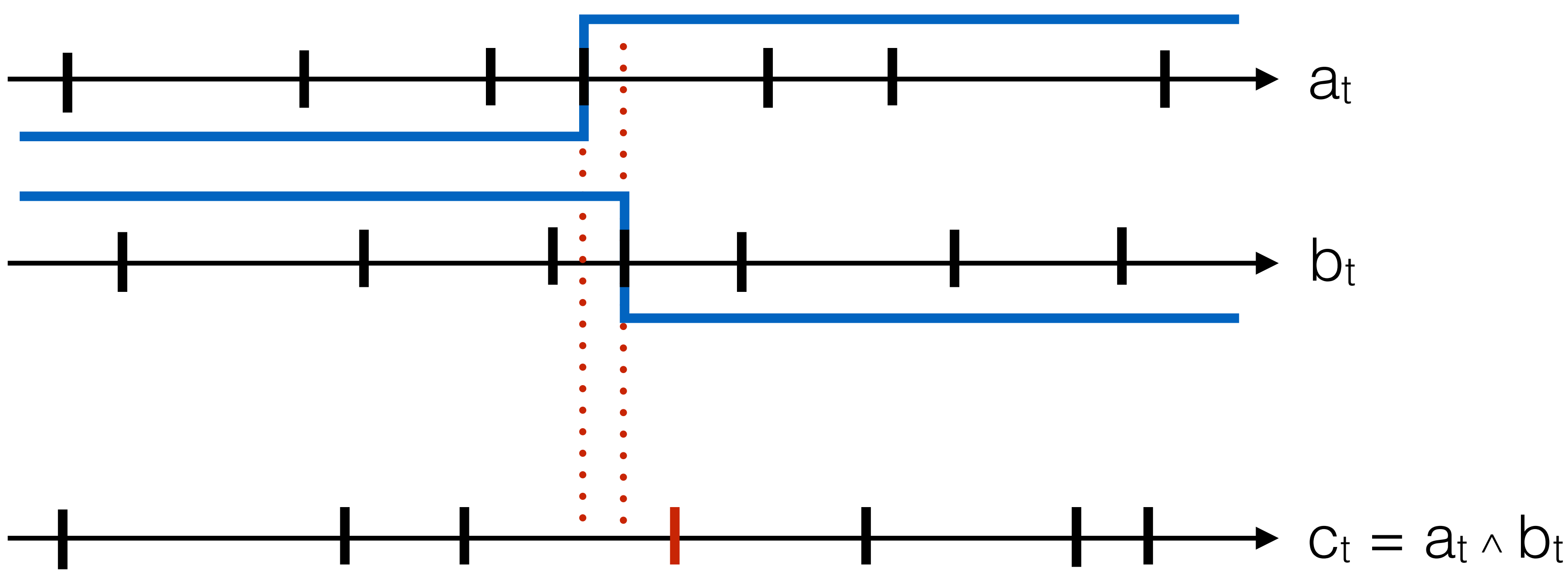
Sampling Artifacts

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





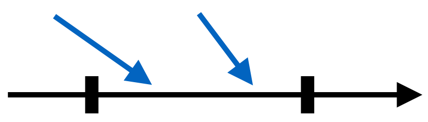




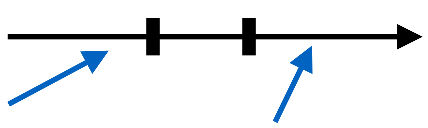
Example from [Caspi 2000]

Sampling Artifacts

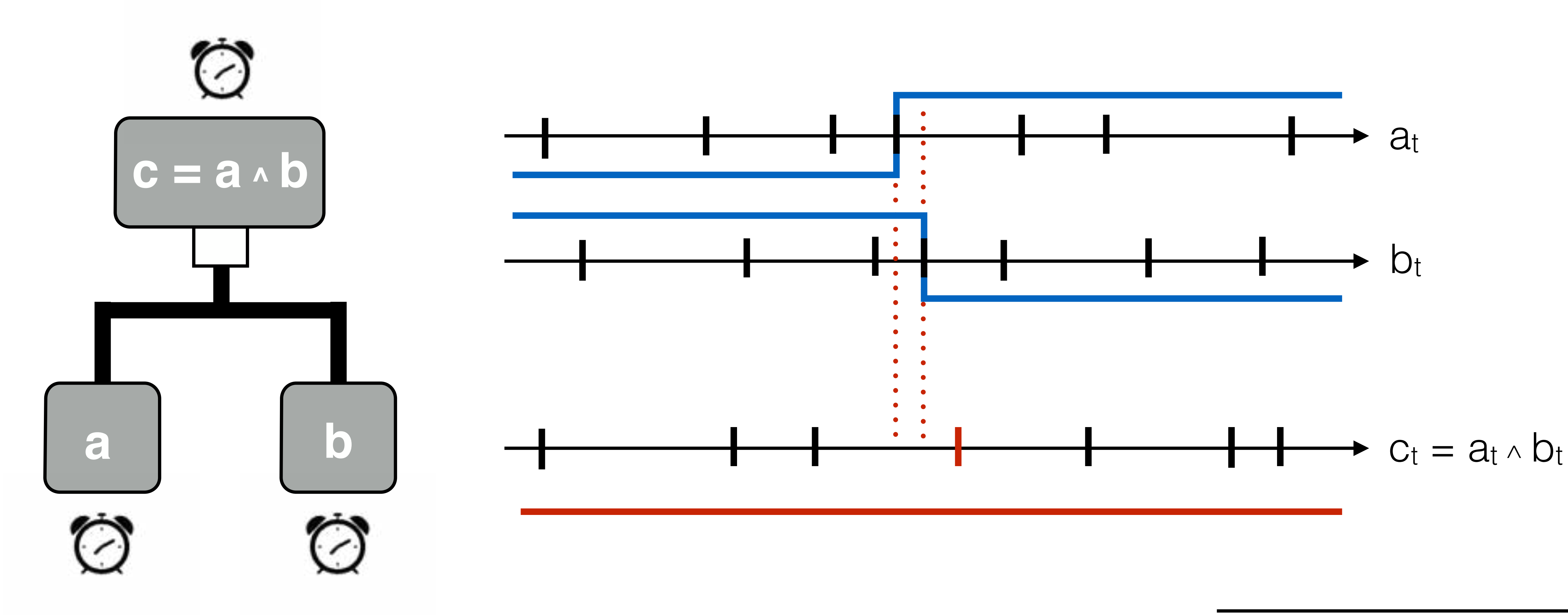
- **Overwriting:** Loss of values



- **Oversampling:** Duplication of values



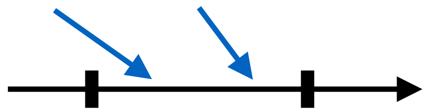
- **Combination of signals**

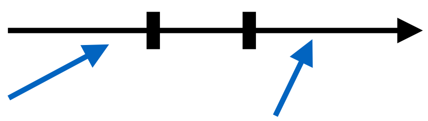


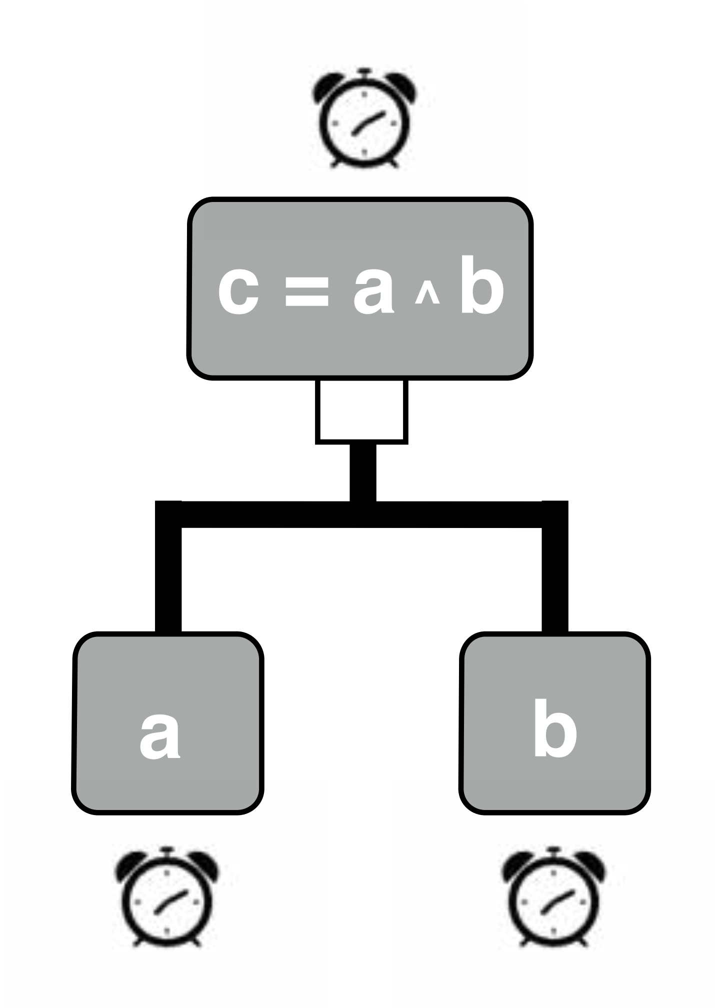
Example from [Caspi 2000]

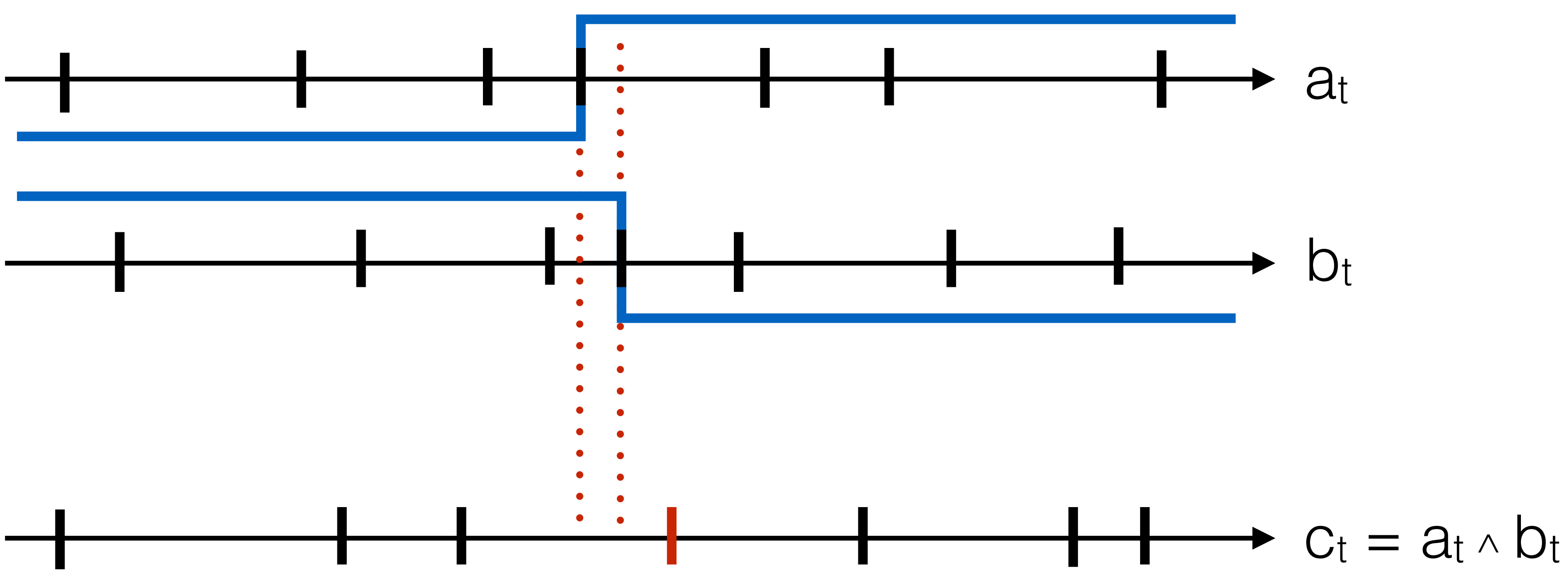
Sampling Artifacts

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





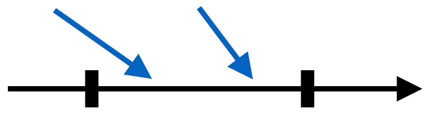


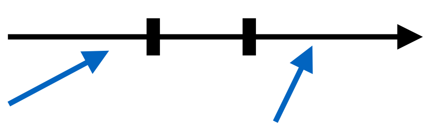


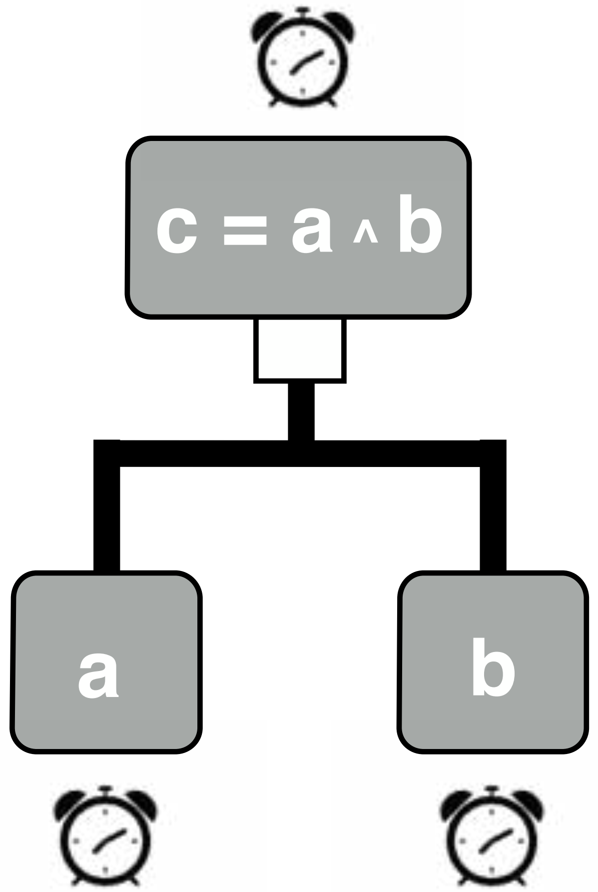
Example from [Caspi 2000]

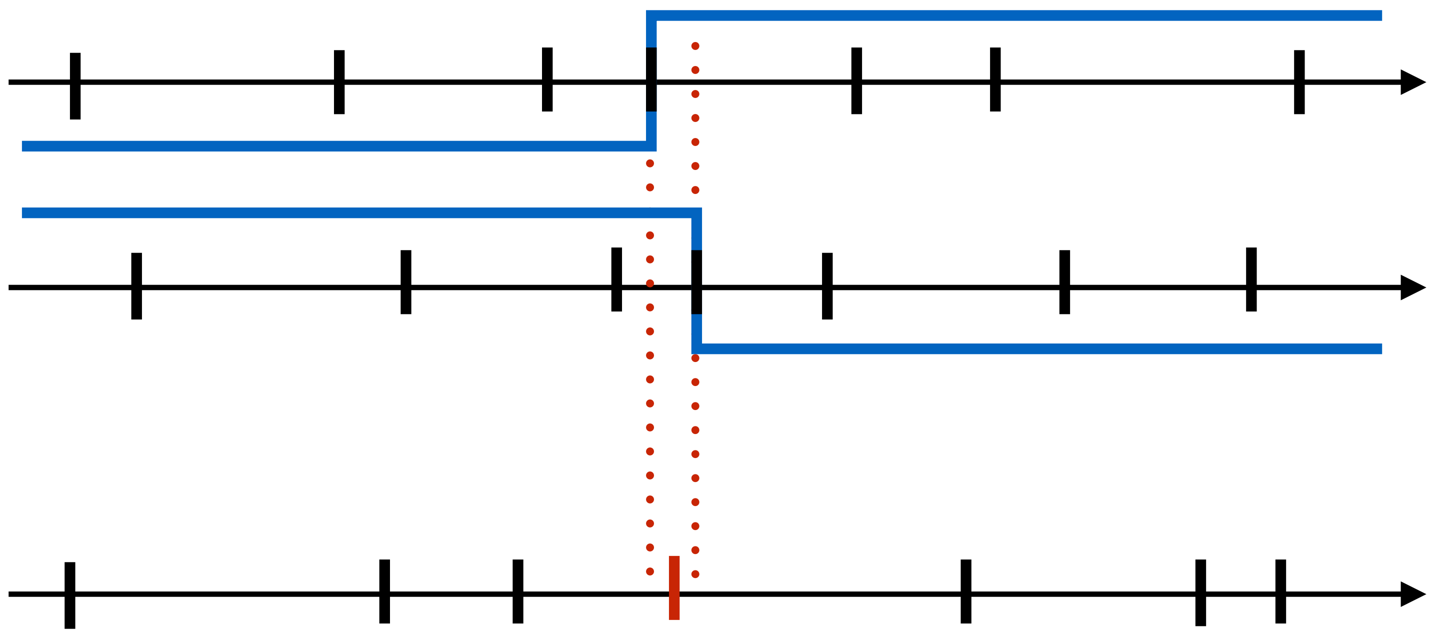
Sampling Artifacts

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





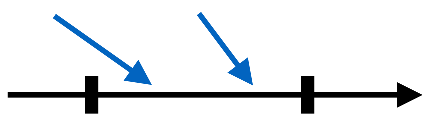


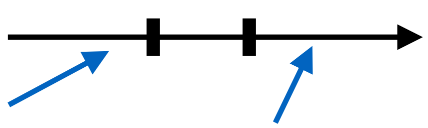


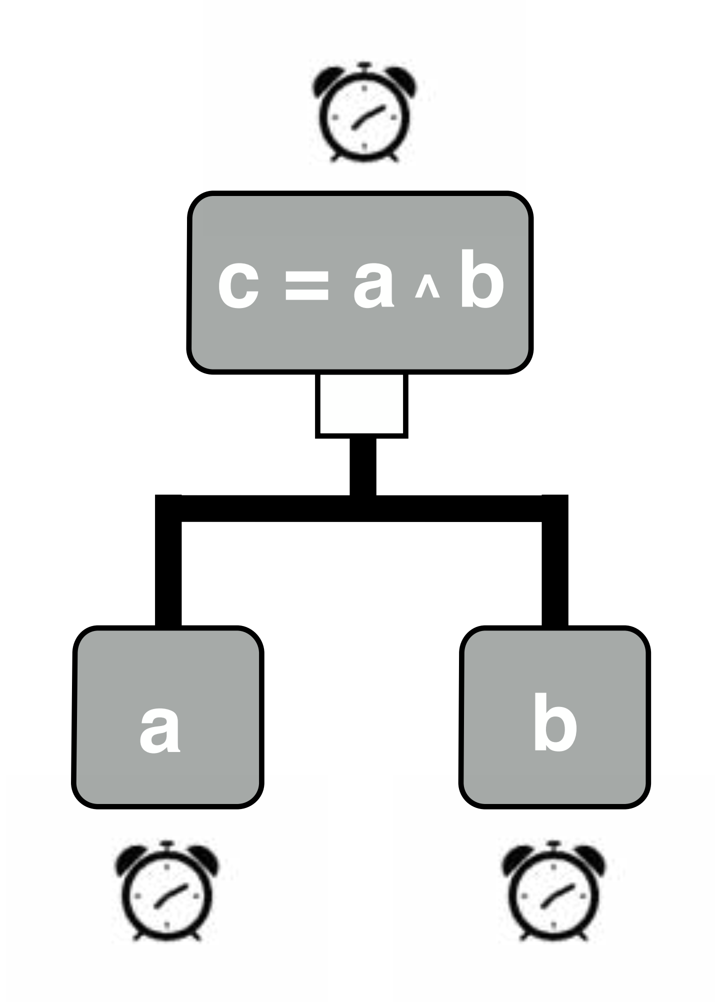
Example from [Caspi 2000]

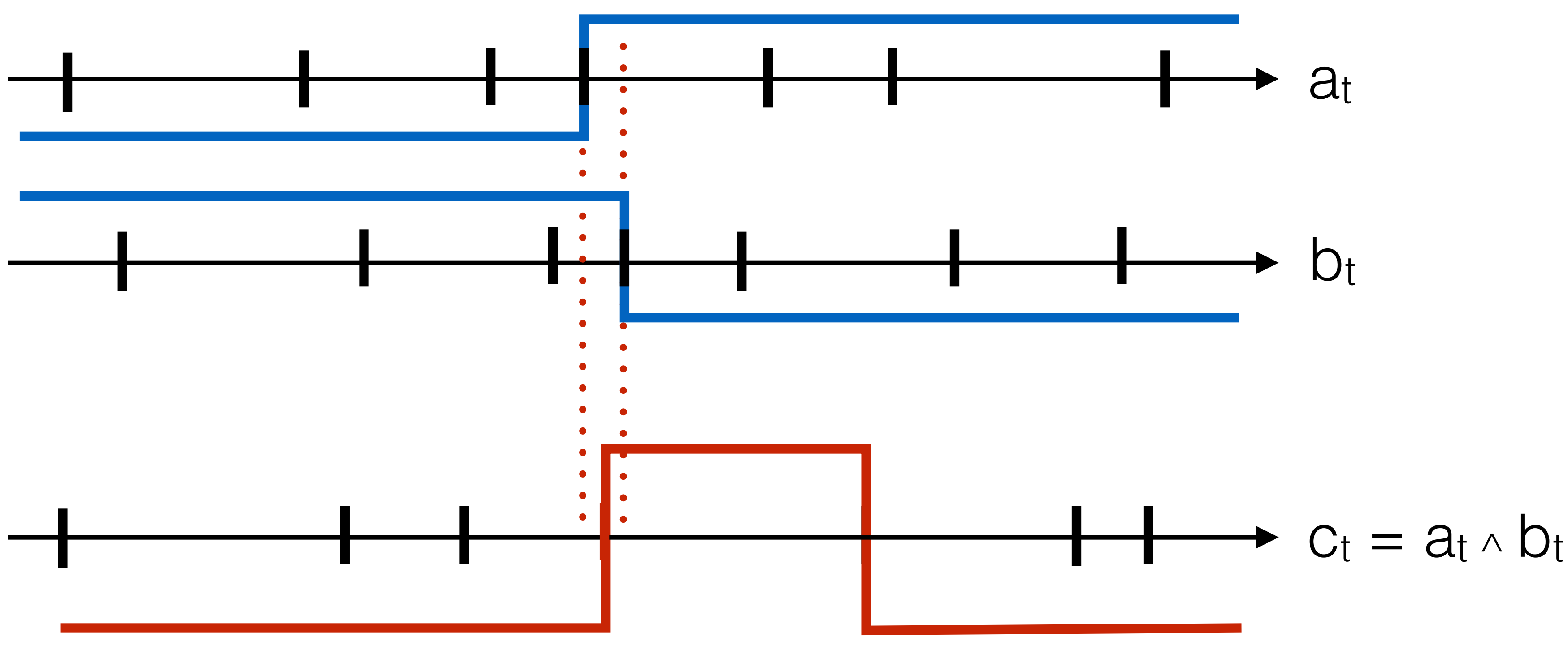
Sampling Artifacts

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









Example from [Caspi 2000]

How to Preserve the Semantics?

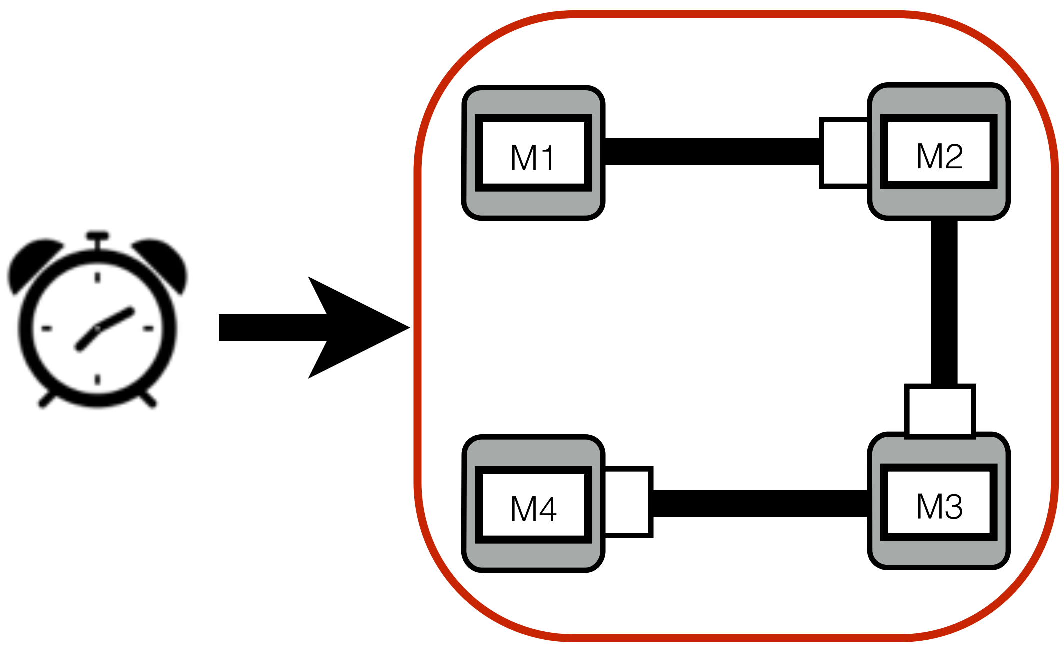
(of a synchronous application on a quasi-periodic architecture)

How to Preserve the Semantics?

(of a synchronous application on a quasi-periodic architecture)

Clock synchronization

e.g. TTA [Kopetz, Bauer 2003]



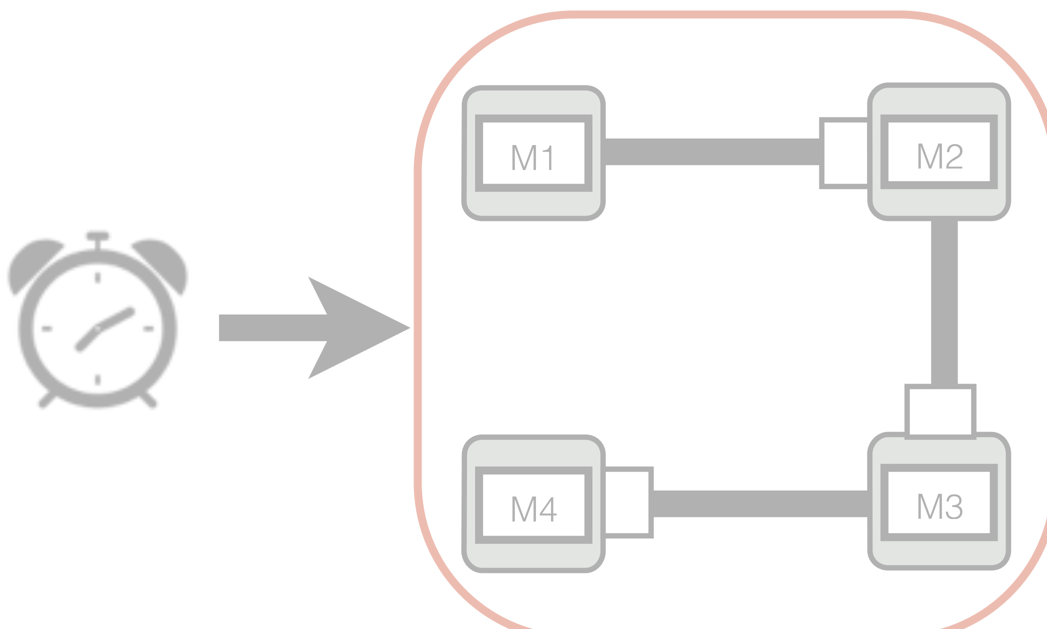
Now efficient and cheap.

How to Preserve the Semantics?

(of a synchronous application on a quasi-periodic architecture)

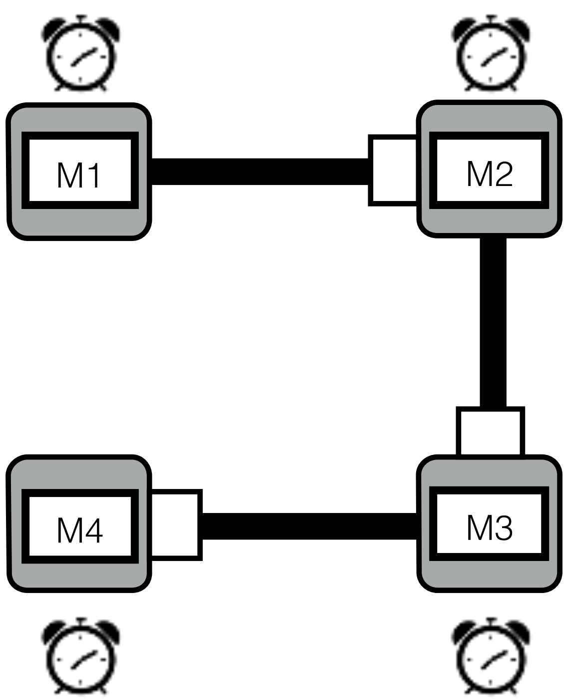
Clock synchronization

e.g. TTA [Kopetz, Bauer 2003]



Now efficient and cheap.

Unsynchronized nodes + Middleware = LTTA

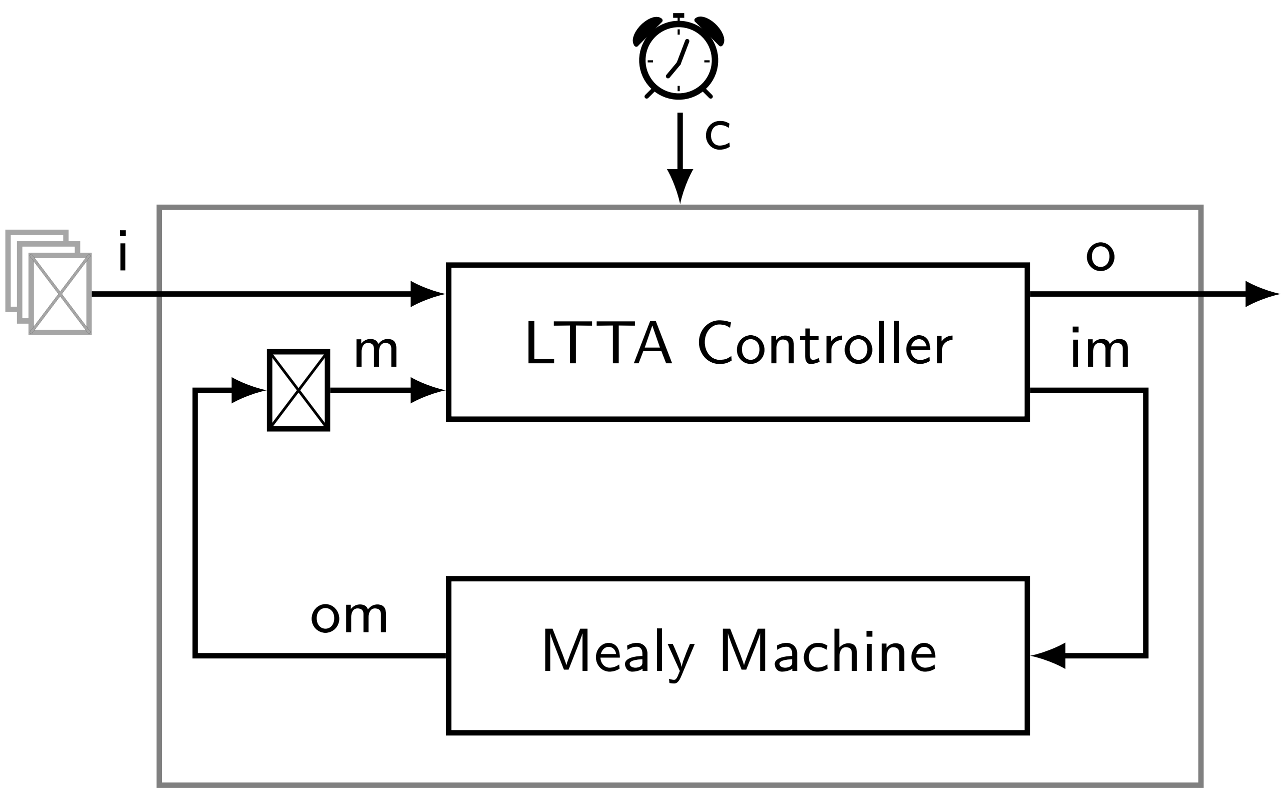


Are they a good idea?

A Synchronous Framework

Everything can be expressed as a **Zélus** program

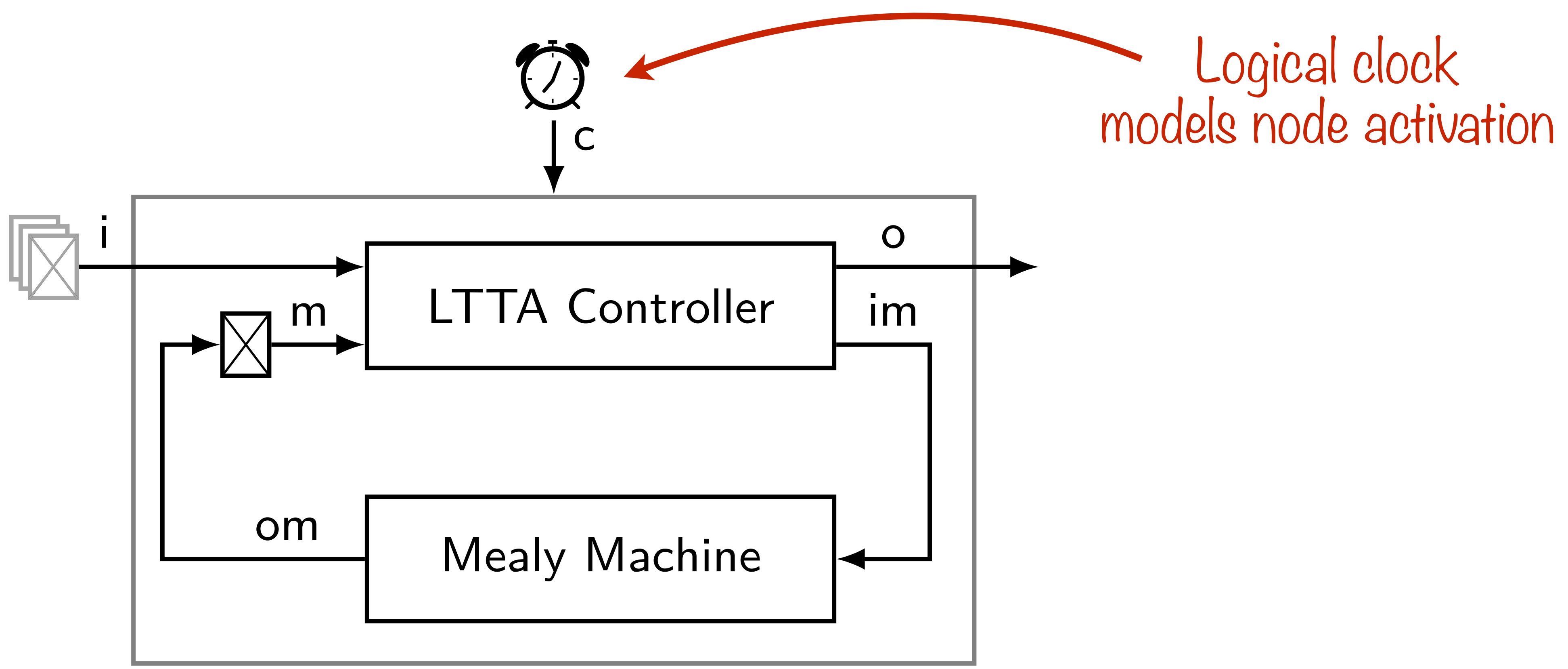
- discrete control (application and controllers)
- continuous time (architecture)



A Synchronous Framework

Everything can be expressed as a **Zélus** program

- discrete control (application and controllers)
- continuous time (architecture)

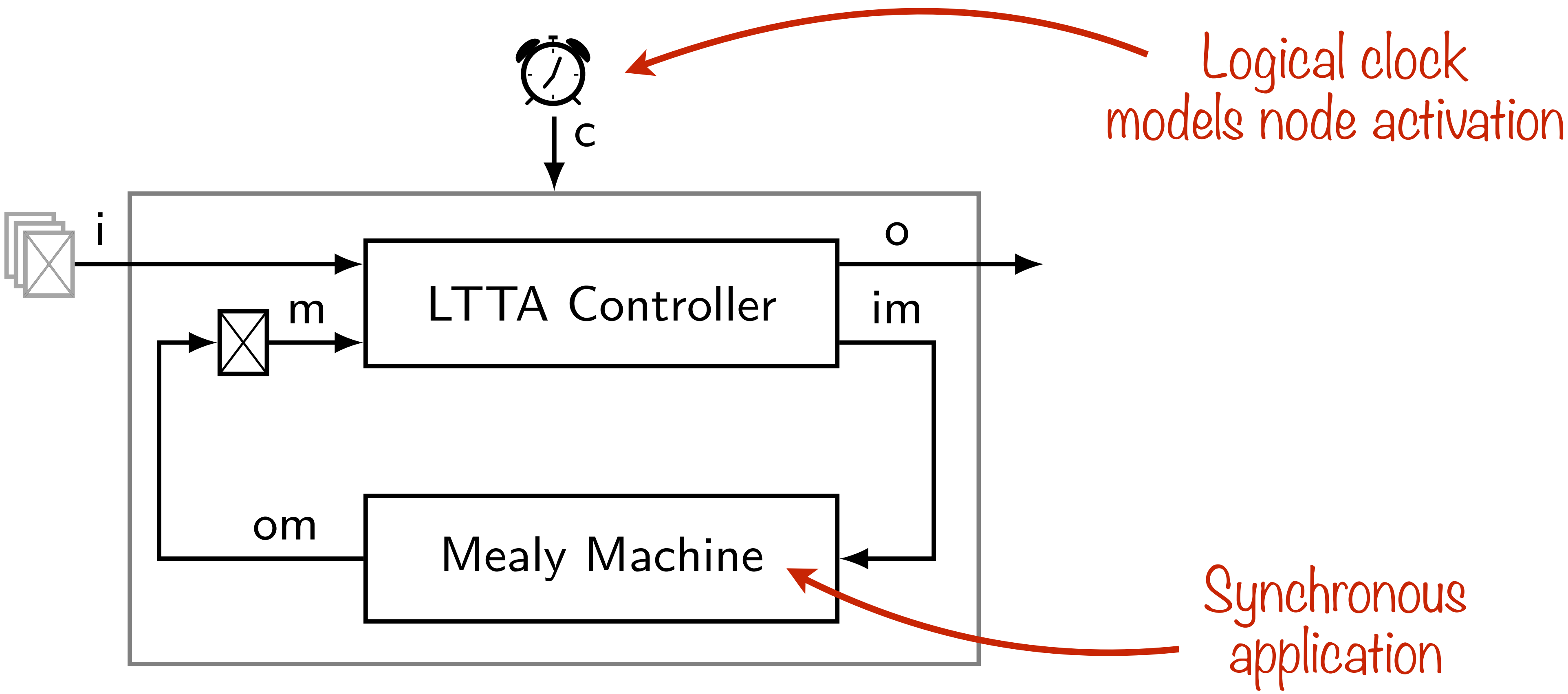


[Halbwachs, Baghdadi 2002]

A Synchronous Framework

Everything can be expressed as a **Zélus** program

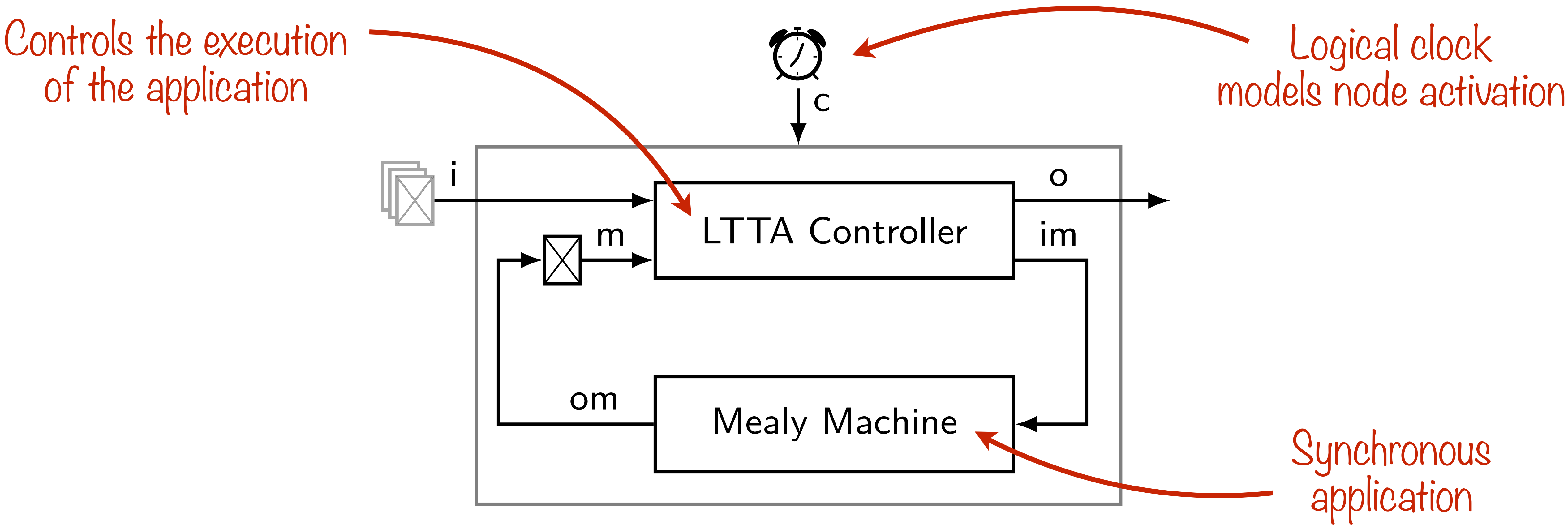
- discrete control (application and controllers)
- continuous time (architecture)



A Synchronous Framework

Everything can be expressed as a **Zélus** program

- discrete control (application and controllers)
- continuous time (architecture)

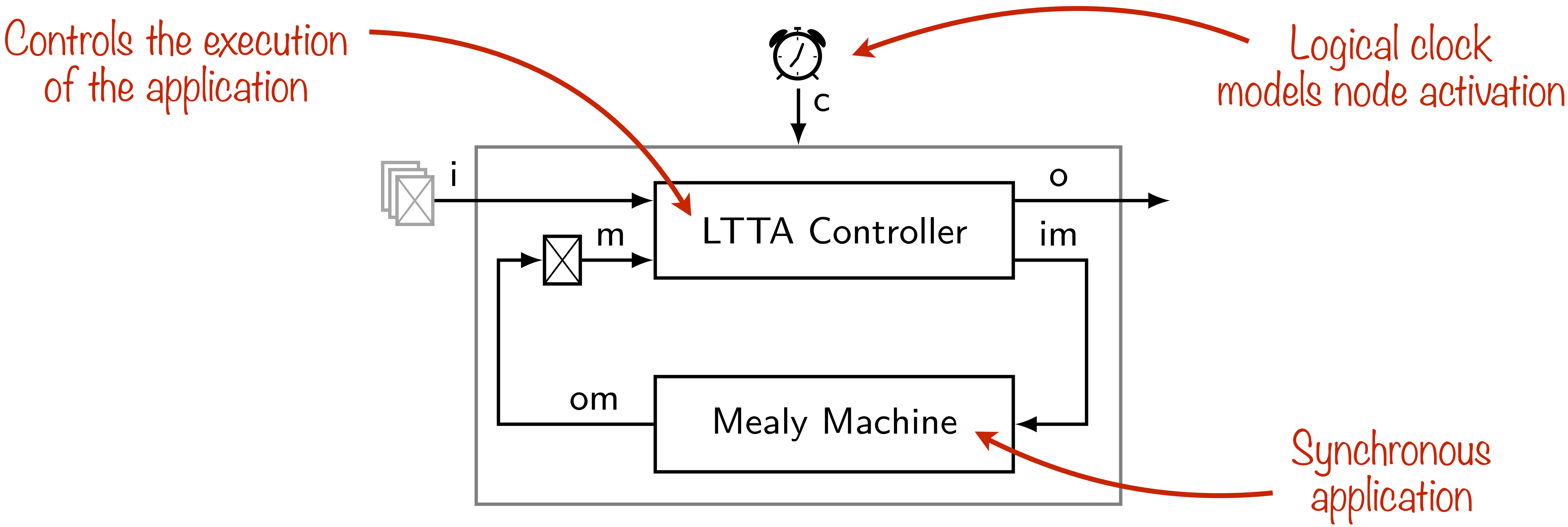


[Halbwachs, Baghdadi 2002]

A Synchronous Framework

Everything can be expressed as a **Zélus** program

- discrete control (application and controllers)
- continuous time (architecture)

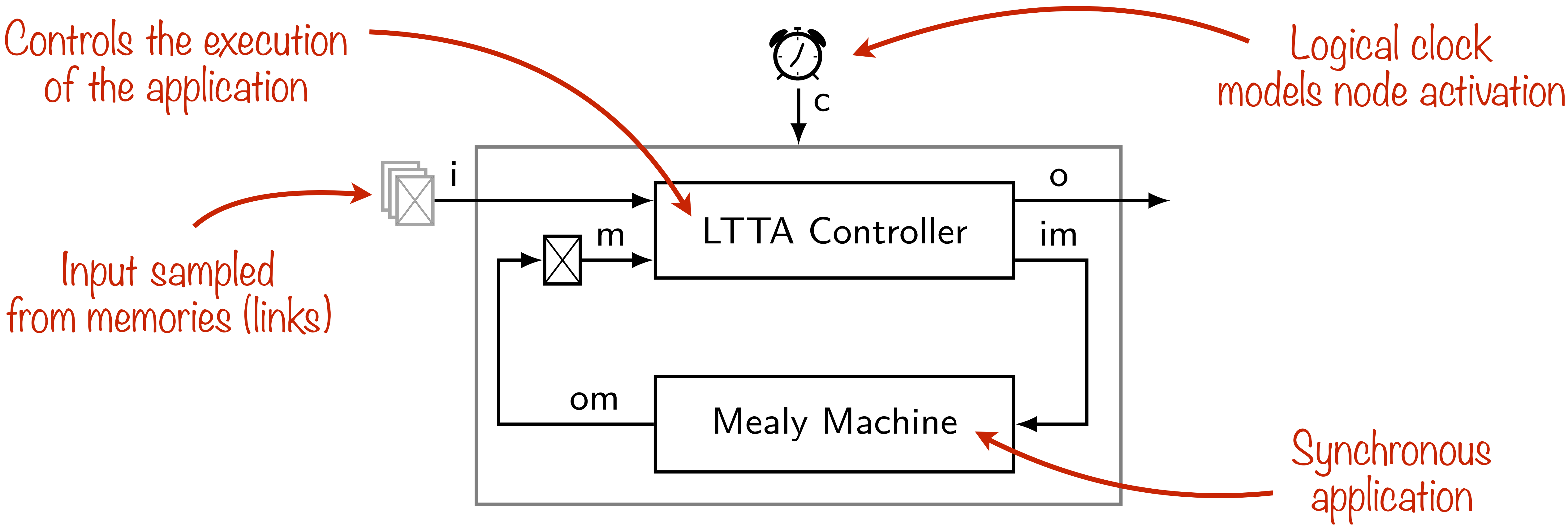


Controller **waits** for new inputs and **delays** publications

A Synchronous Framework

Everything can be expressed as a **Zélus** program

- discrete control (application and controllers)
- continuous time (architecture)

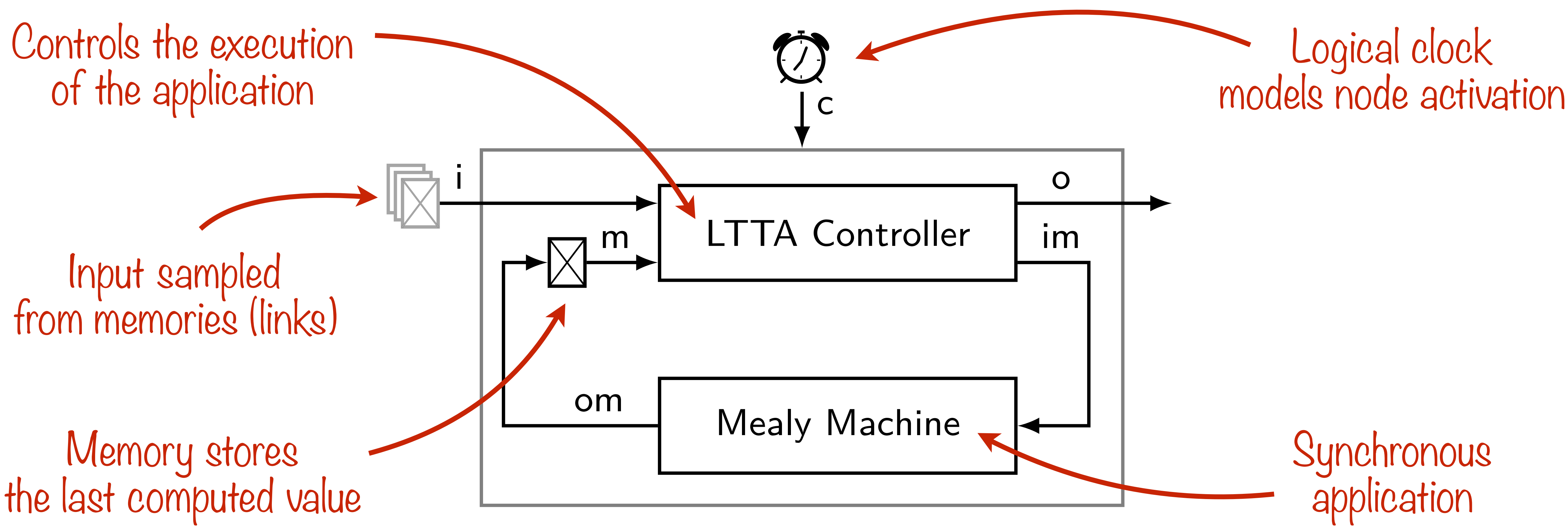


Controller **waits** for new inputs and **delays** publications

A Synchronous Framework

Everything can be expressed as a **Zélus** program

- discrete control (application and controllers)
- continuous time (architecture)



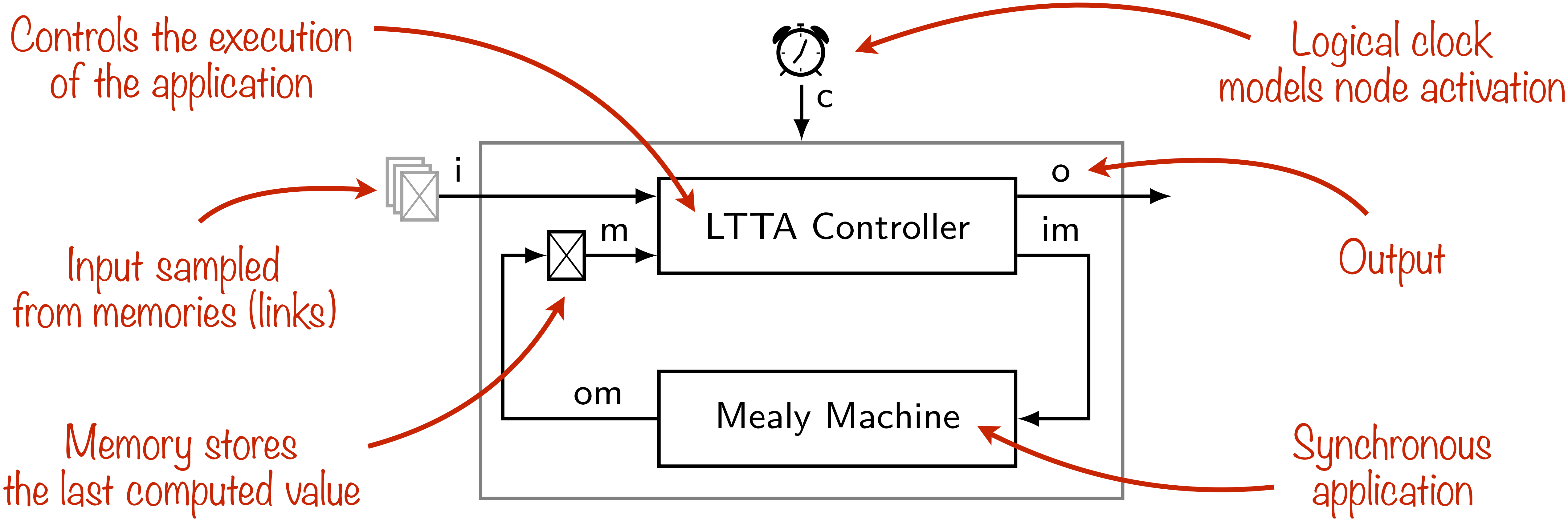
Controller **waits** for new inputs and **delays** publications

[Halbwachs, Baghdadi 2002]

A Synchronous Framework

Everything can be expressed as a **Zélus** program

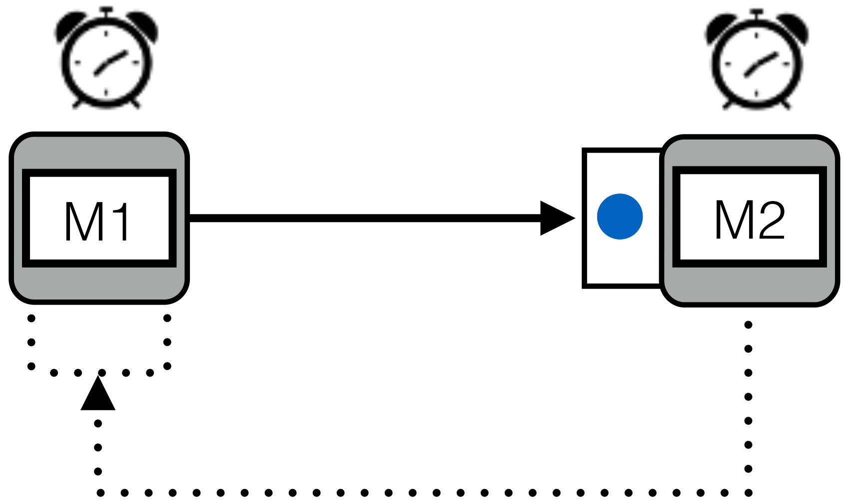
- discrete control (application and controllers)
- continuous time (architecture)



Controller **waits** for new inputs and **delays** publications

A First Idea: Back-Pressure

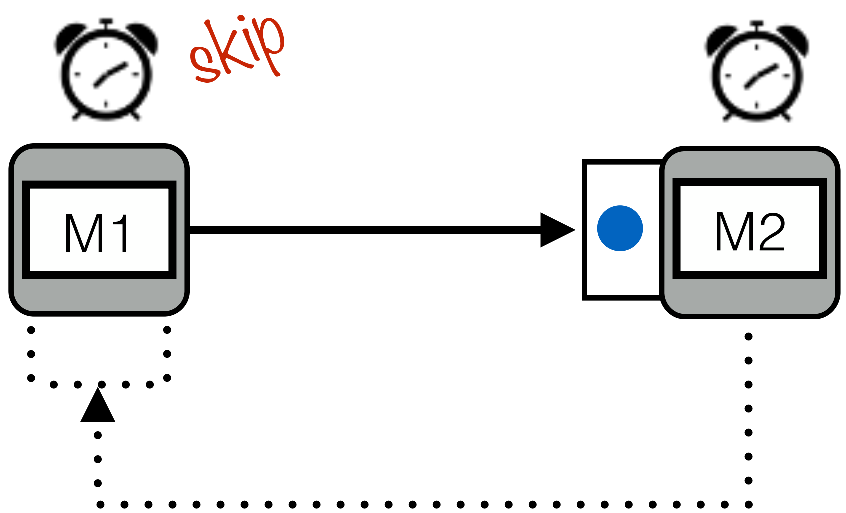
- A producer waits for **acknowledgements** from its consumers before sending a new value.
- Nodes **skip** when no acknowledgement or message has been received.



[Tripakis et al. 2008]

A First Idea: Back-Pressure

- A producer waits for **acknowledgements** from its consumers before sending a new value.
- Nodes **skip** when no acknowledgement or message has been received.

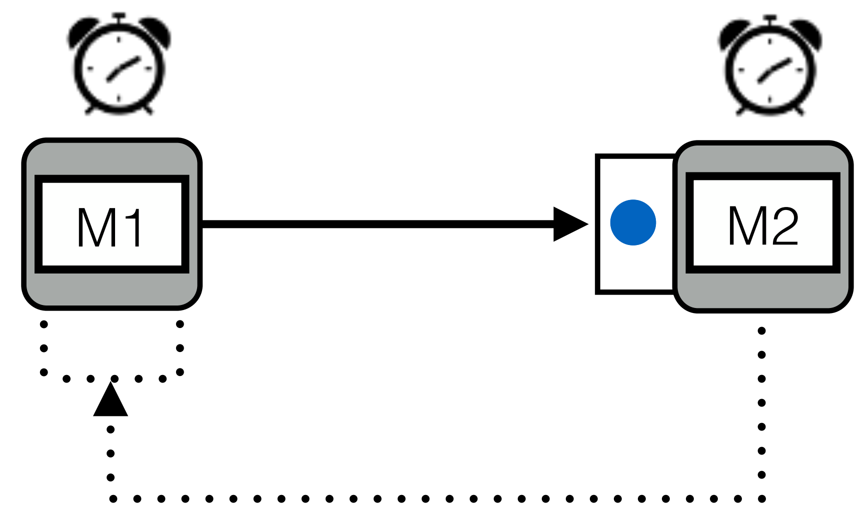


[Tripakis et al. 2008]

A First Idea: Back-Pressure

- A producer waits for **acknowledgements** from its consumers before sending a new value.

- Nodes **skip** when no acknowledgement or message has been received.

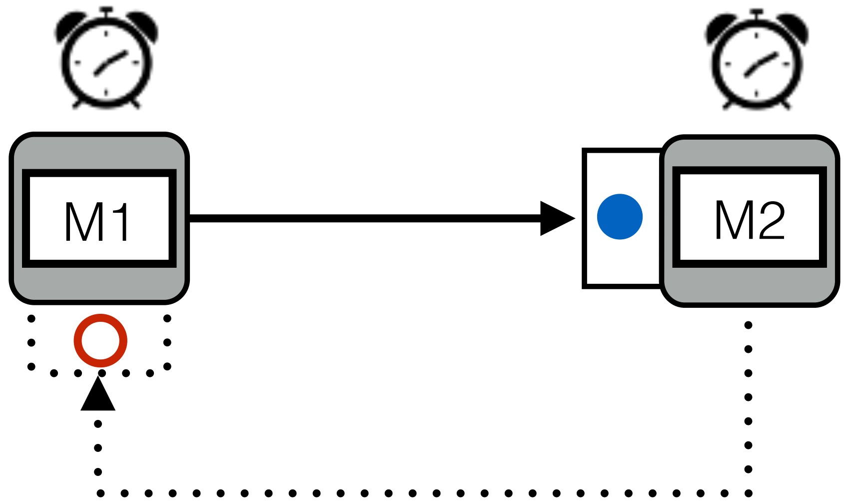


[Tripakis et al. 2008]

A First Idea: Back-Pressure

- A producer waits for **acknowledgements** from its consumers before sending a new value.

- Nodes **skip** when no acknowledgement or message has been received.

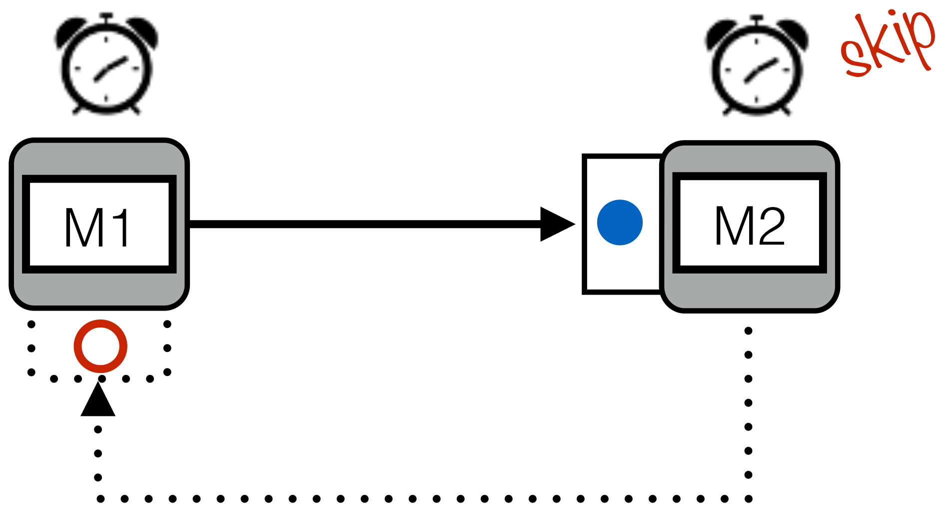


[Tripakis et al. 2008]

A First Idea: Back-Pressure

- A producer waits for **acknowledgements** from its consumers before sending a new value.

- Nodes **skip** when no acknowledgement or message has been received.

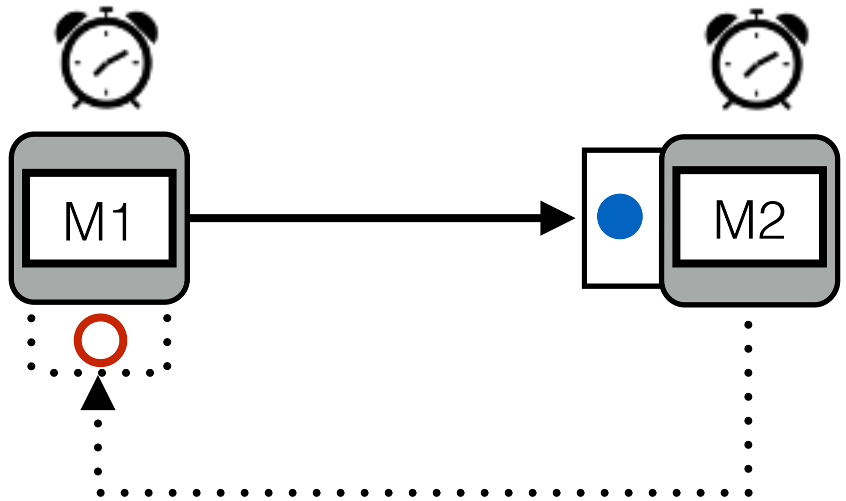


[Tripakis et al. 2008]

A First Idea: Back-Pressure

- A producer waits for **acknowledgements** from its consumers before sending a new value.

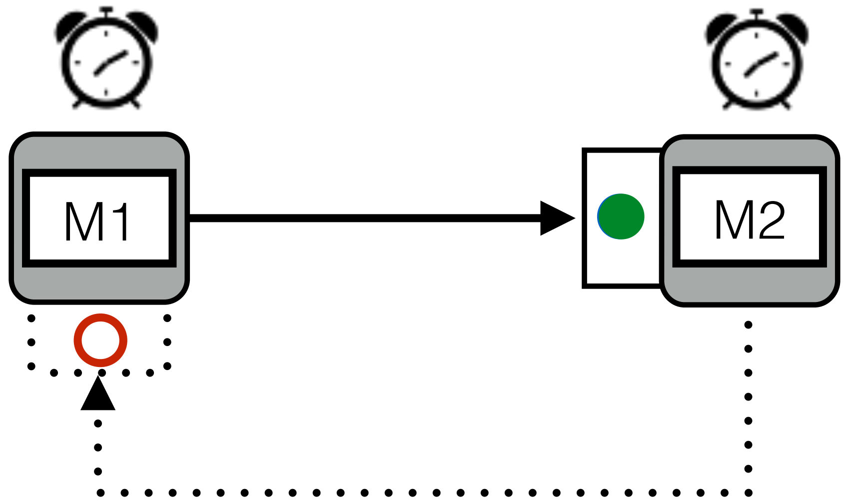
- Nodes **skip** when no acknowledgement or message has been received.



[Tripakis et al. 2008]

A First Idea: Back-Pressure

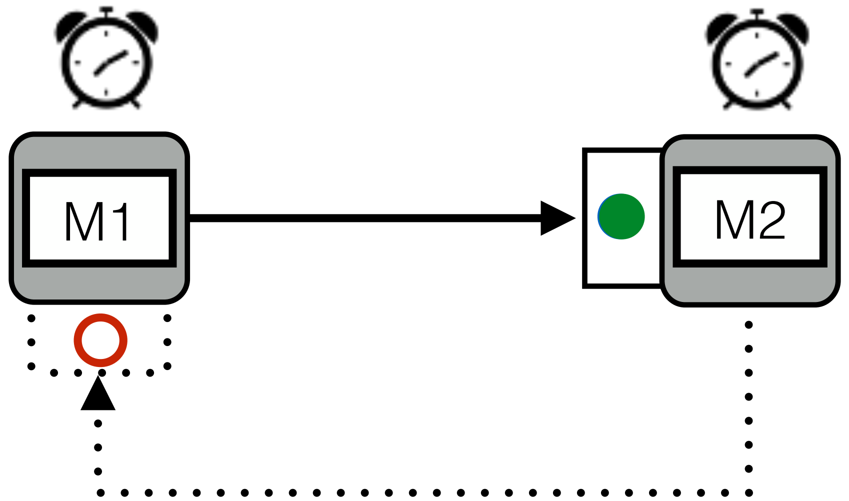
- A producer waits for **acknowledgements** from its consumers before sending a new value.
- Nodes **skip** when no acknowledgement or message has been received.



[Tripakis et al. 2008]

A First Idea: Back-Pressure

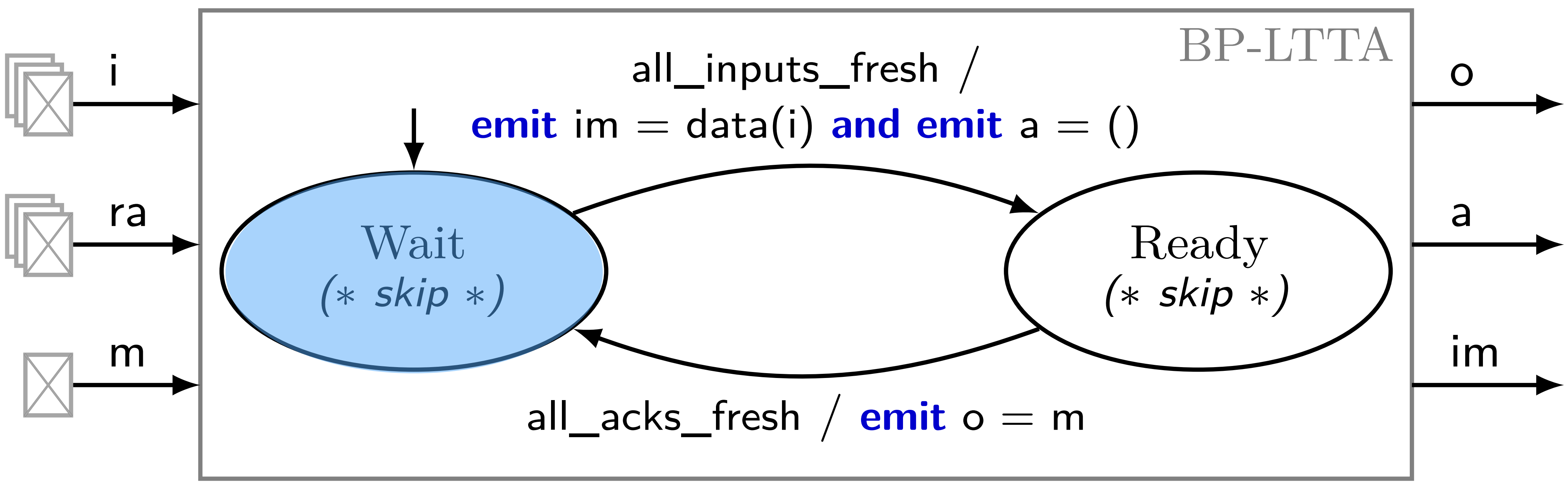
- A producer waits for **acknowledgements** from its consumers before sending a new value.
- Nodes **skip** when no acknowledgement or message has been received.



Flexibility: there is no assumption on the architecture.
Robustness: if a node crashes the entire network is stuck.

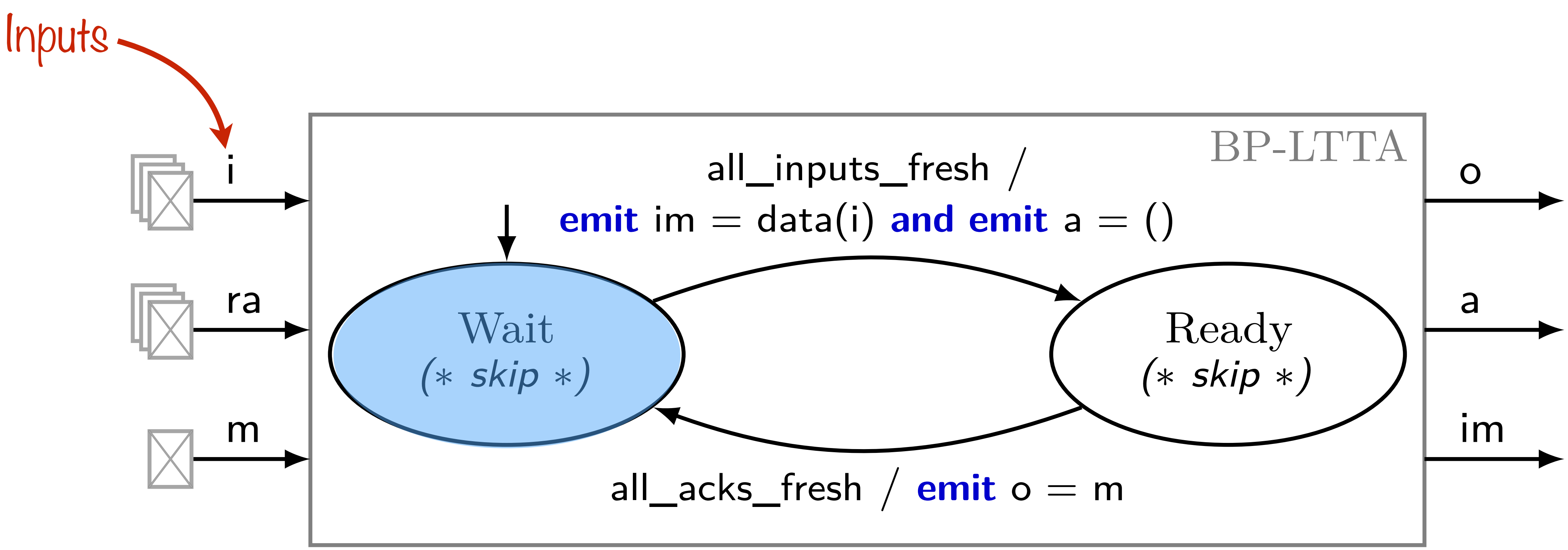
[Tripakis et al. 2008]

A First Idea: Back-Pressure



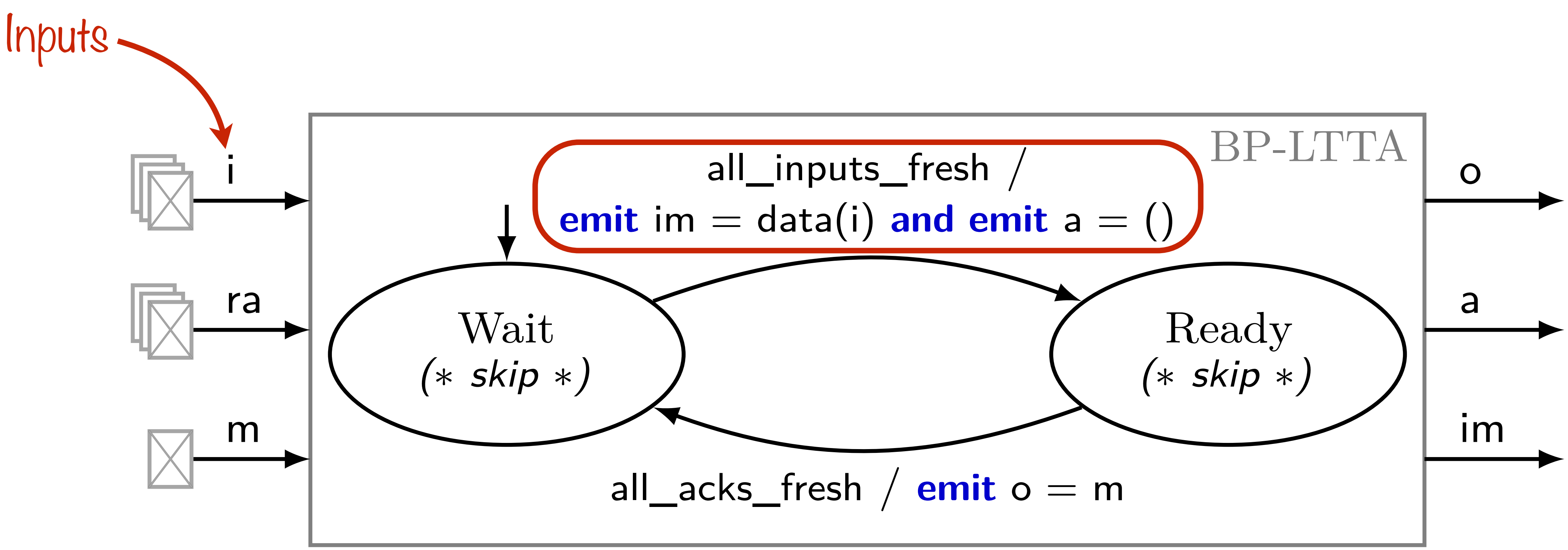
Controller **waits** for new inputs and **delays** publications

A First Idea: Back-Pressure



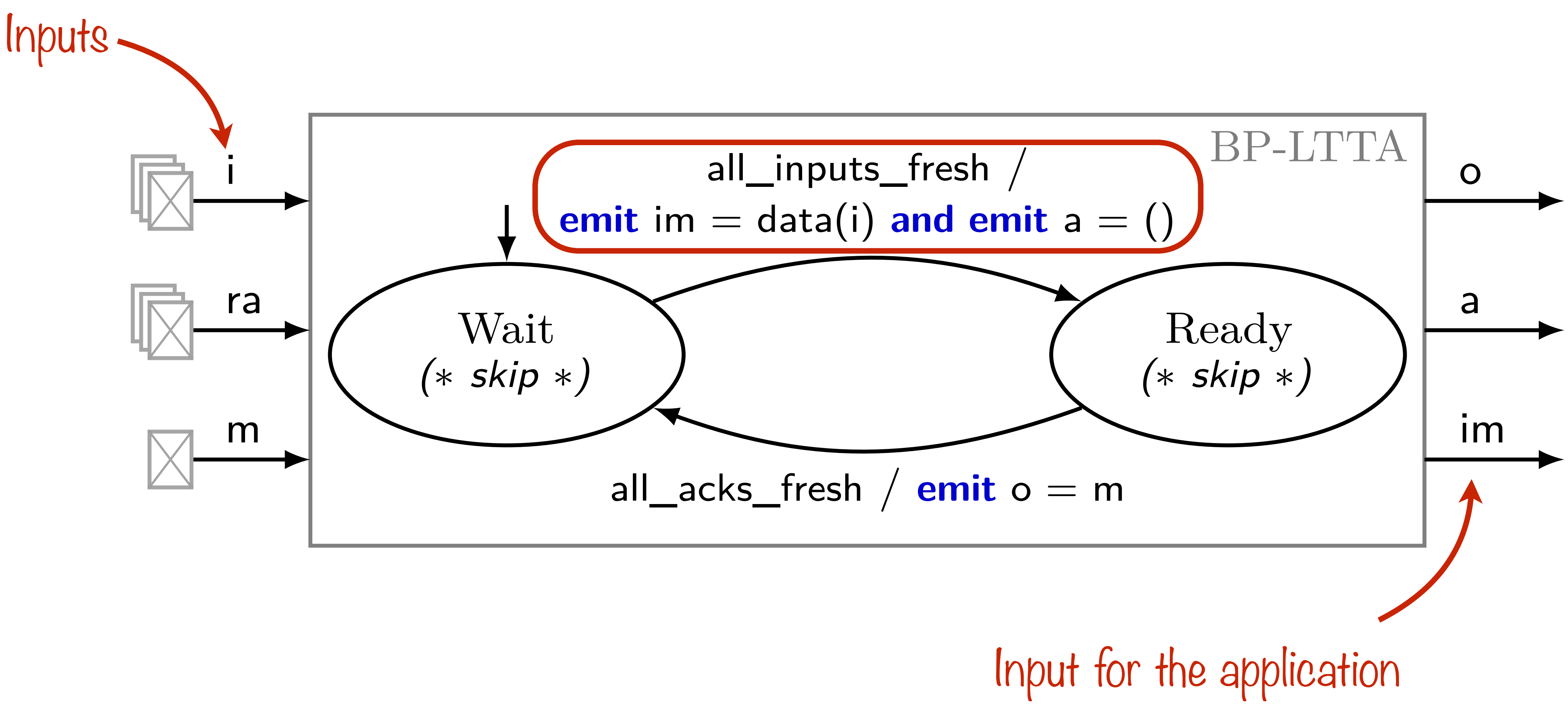
Controller **waits** for new inputs and **delays** publications

A First Idea: Back-Pressure



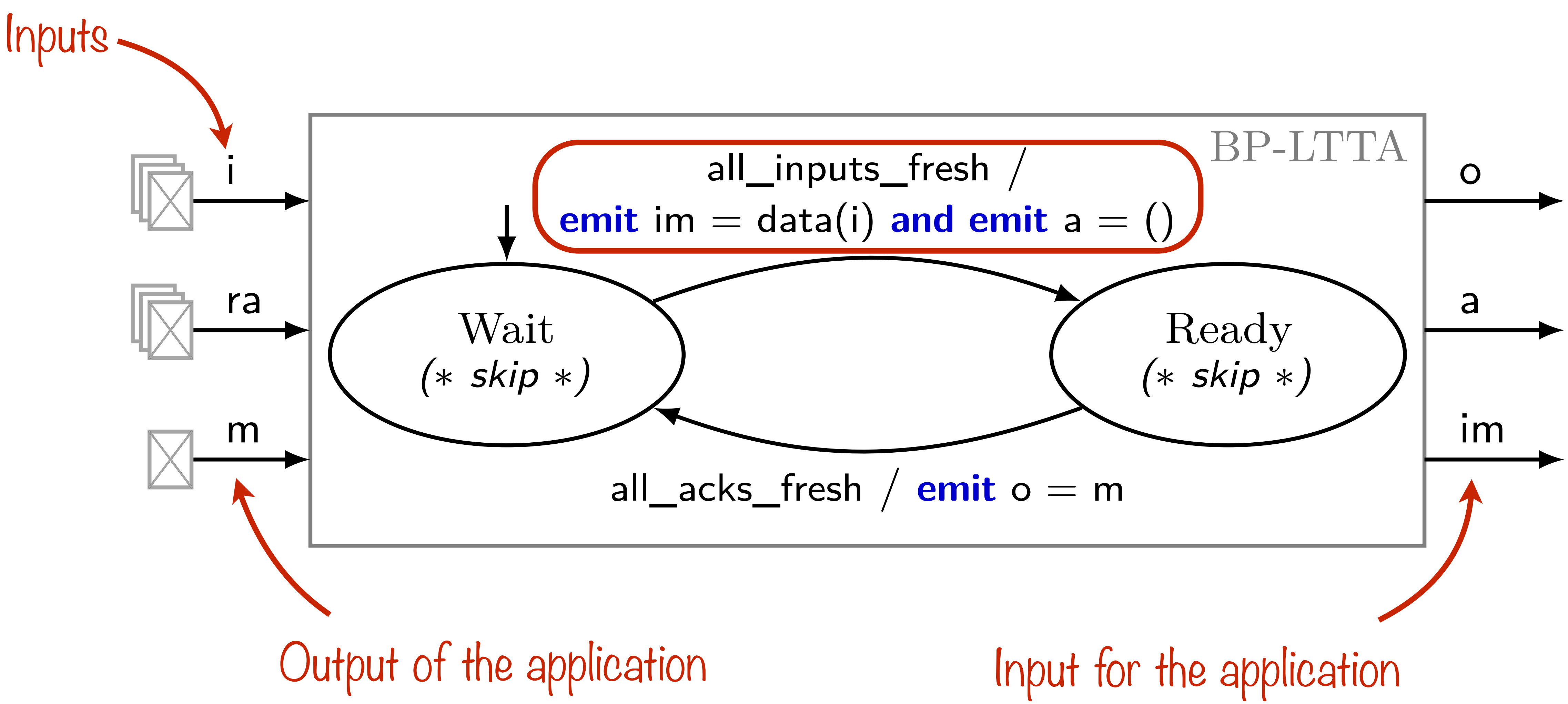
Controller **waits** for new inputs and **delays** publications

A First Idea: Back-Pressure



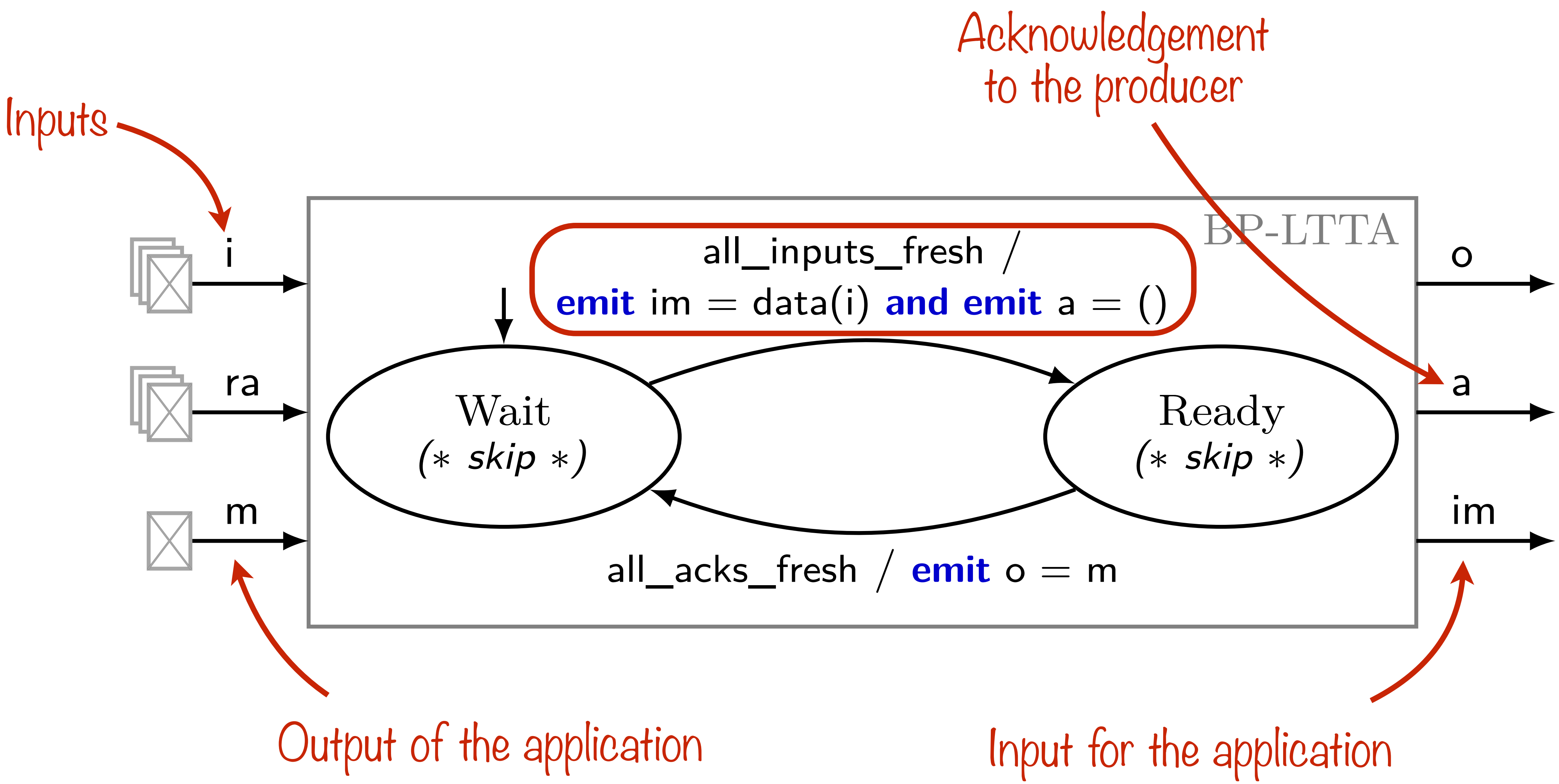
Controller **waits** for new inputs and **delays** publications

A First Idea: Back-Pressure



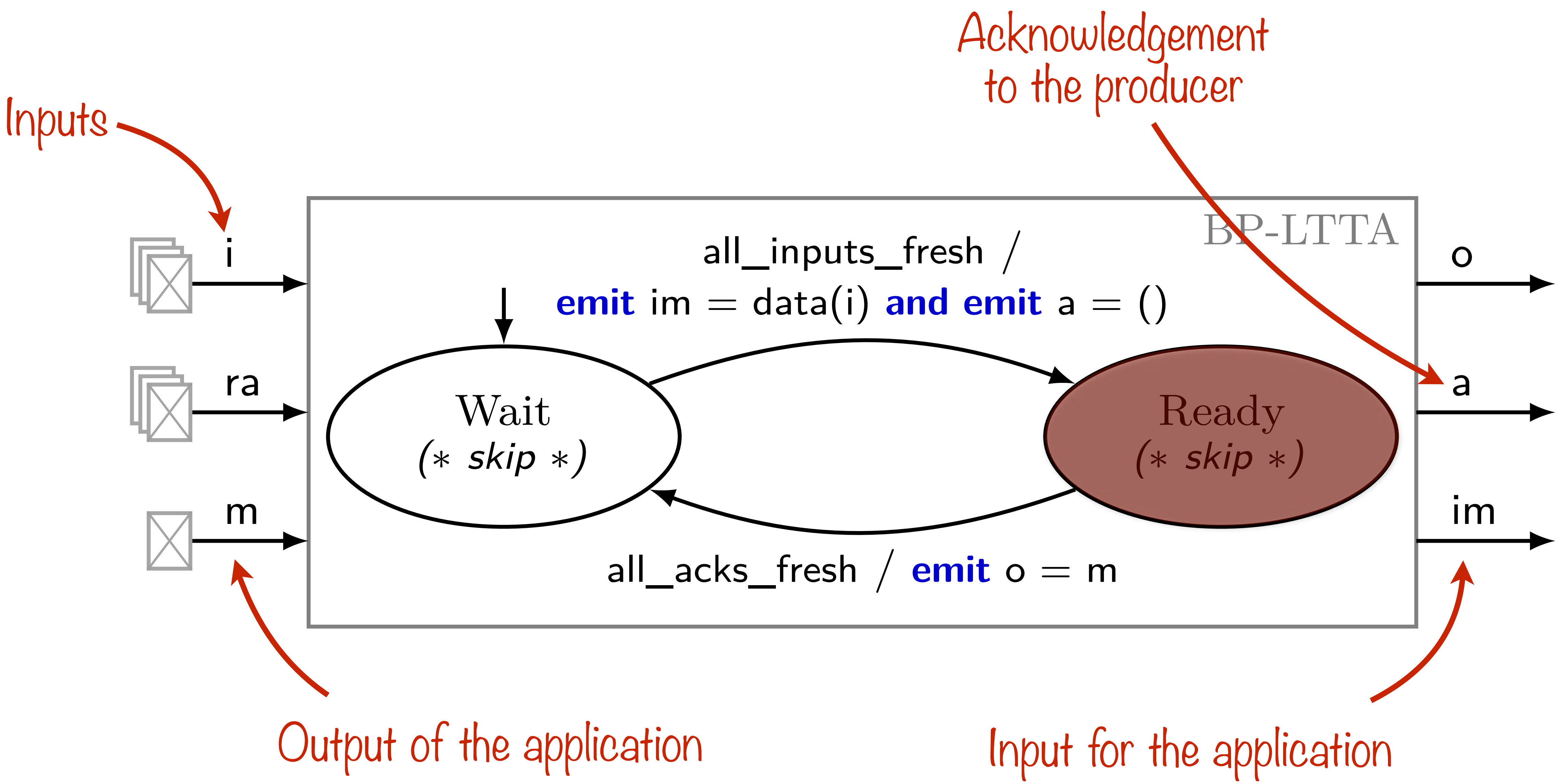
Controller **waits** for new inputs and **delays** publications

A First Idea: Back-Pressure



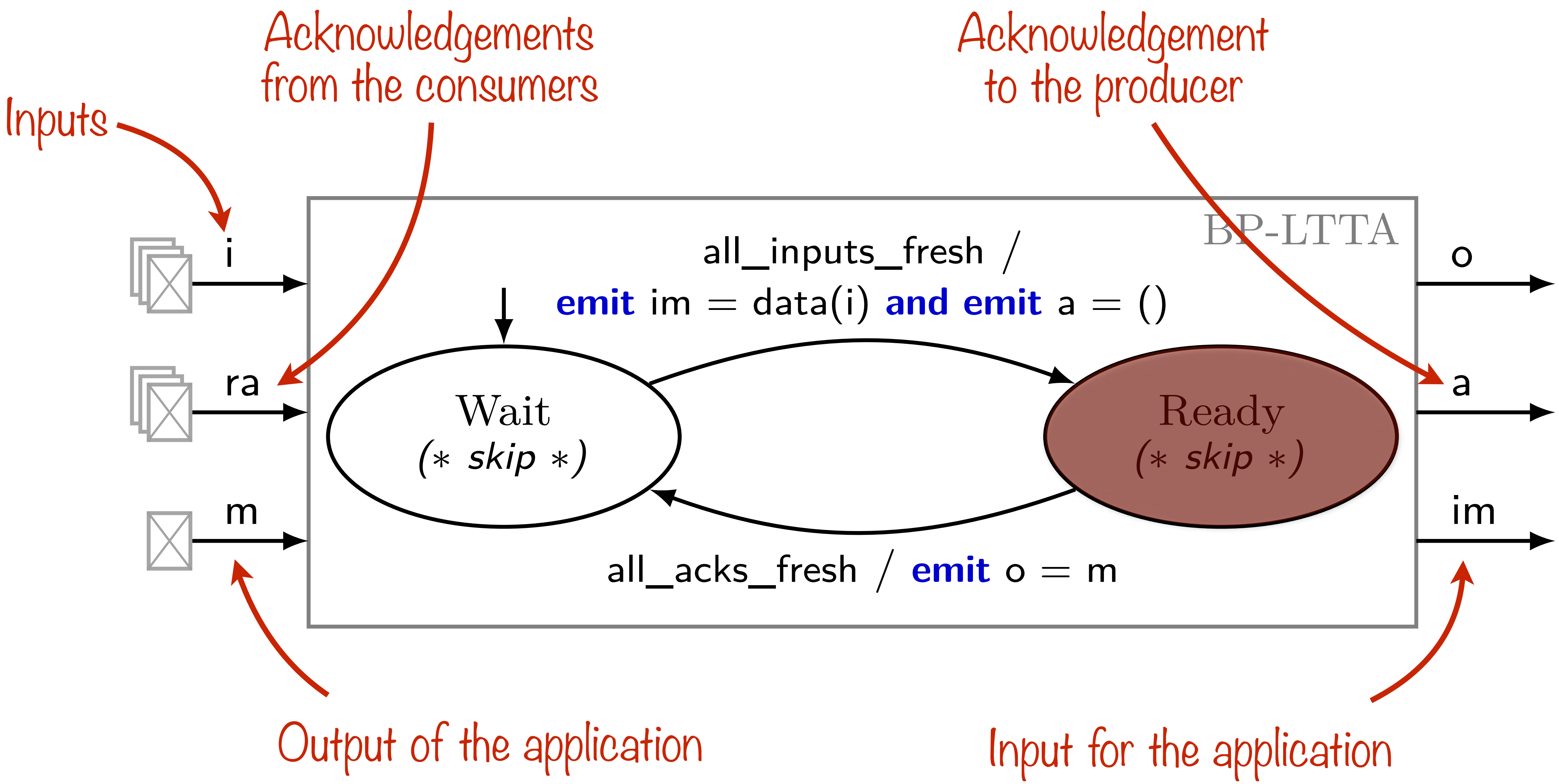
Controller **waits** for new inputs and **delays** publications

A First Idea: Back-Pressure



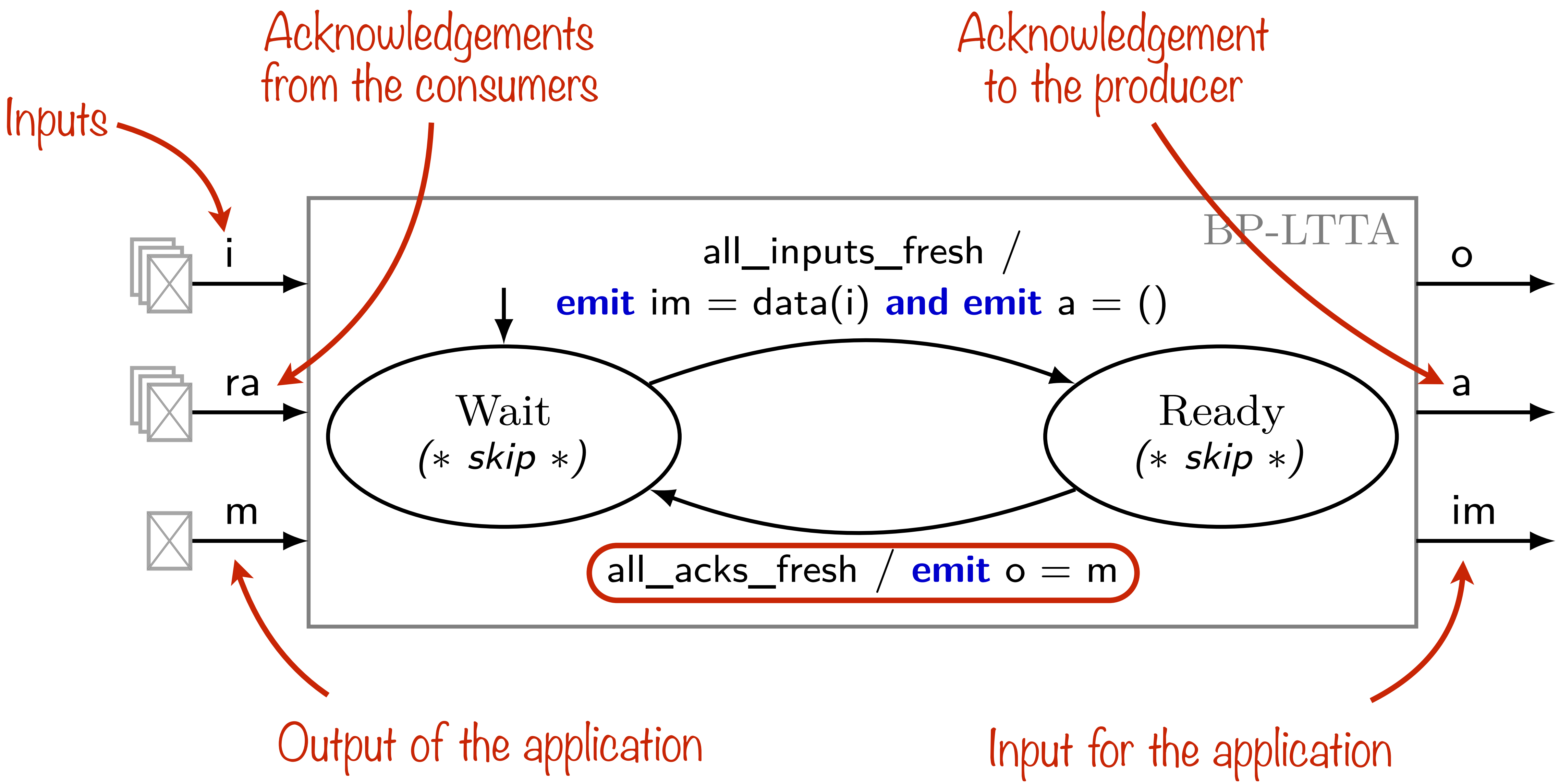
Controller **waits** for new inputs and **delays** publications

A First Idea: Back-Pressure



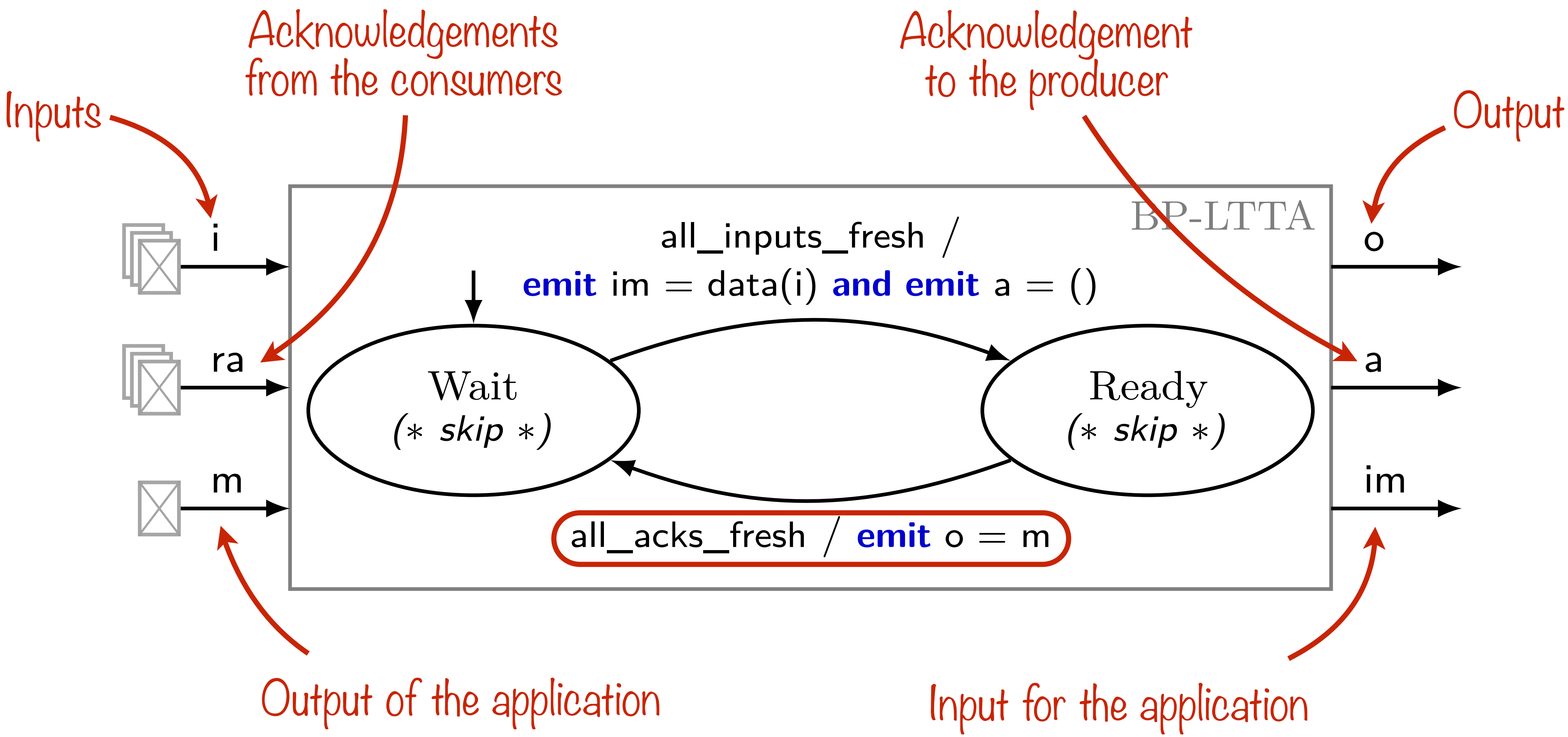
Controller **waits** for new inputs and **delays** publications

A First Idea: Back-Pressure



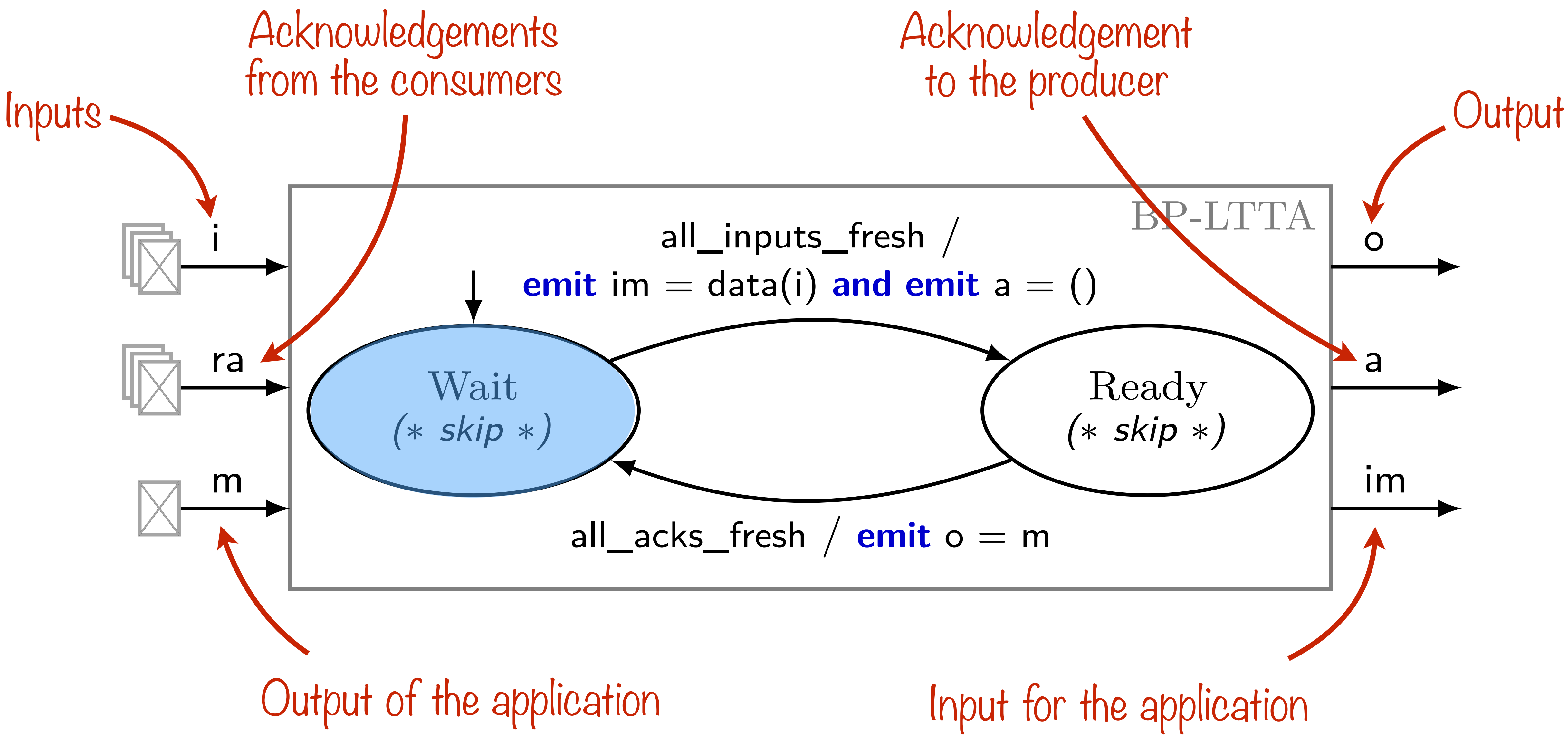
Controller **waits** for new inputs and **delays** publications

A First Idea: Back-Pressure



Controller **waits** for new inputs and **delays** publications

A First Idea: Back-Pressure



Controller **waits** for new inputs and **delays** publications

Time-Based LTTA

Why another protocol?

Back-pressure multiplies the number of messages and memories, and **blocks if a node crashes**

We can take advantage of the **quasi-periodic** nature of the architecture to replace acknowledgment by **waiting**.

At some point, a node can be sure that:

- the last sent data has been read
- a fresh value is available in the memory

[Caspi & Benveniste 2008]

Time-Based LTTA

Why another protocol?

Back-pressure multiplies the number of messages and memories, and **blocks if a node crashes**

We can take advantage of the **quasi-periodic** nature of the architecture to replace acknowledgment by **waiting**.

At some point, a node can be sure that:

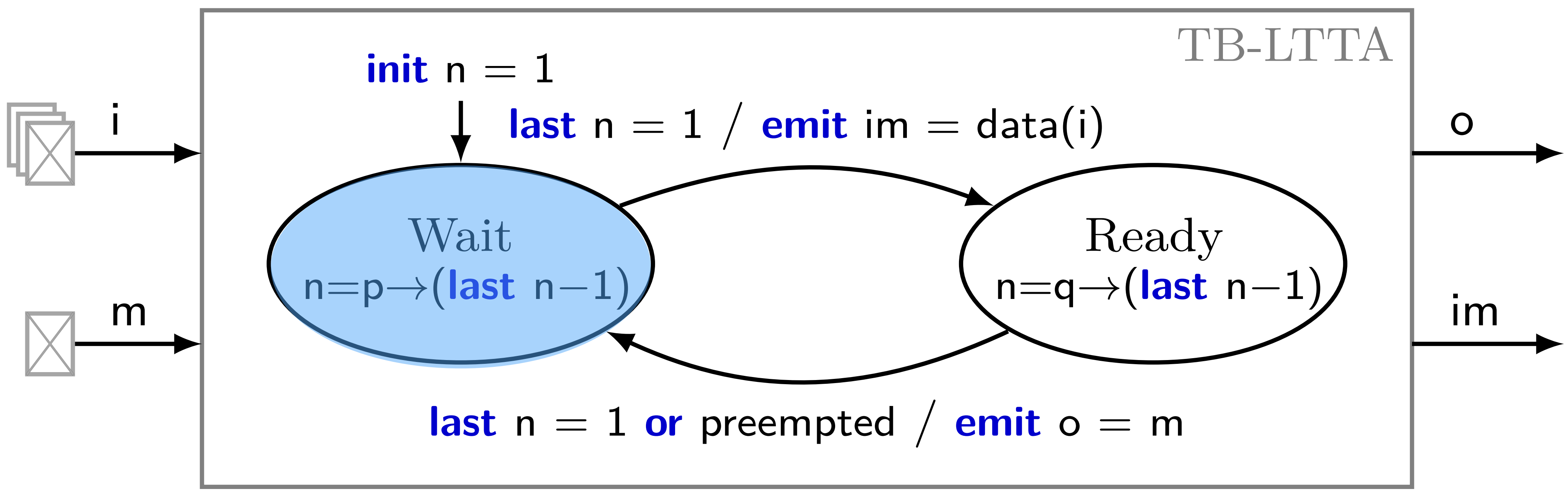
- the last sent data has been read
- a fresh value is available in the memory

Flexibility: it requires architecture characteristics.
Robustness: controllers can run in a degraded mode.

[Caspi & Benveniste 2008]

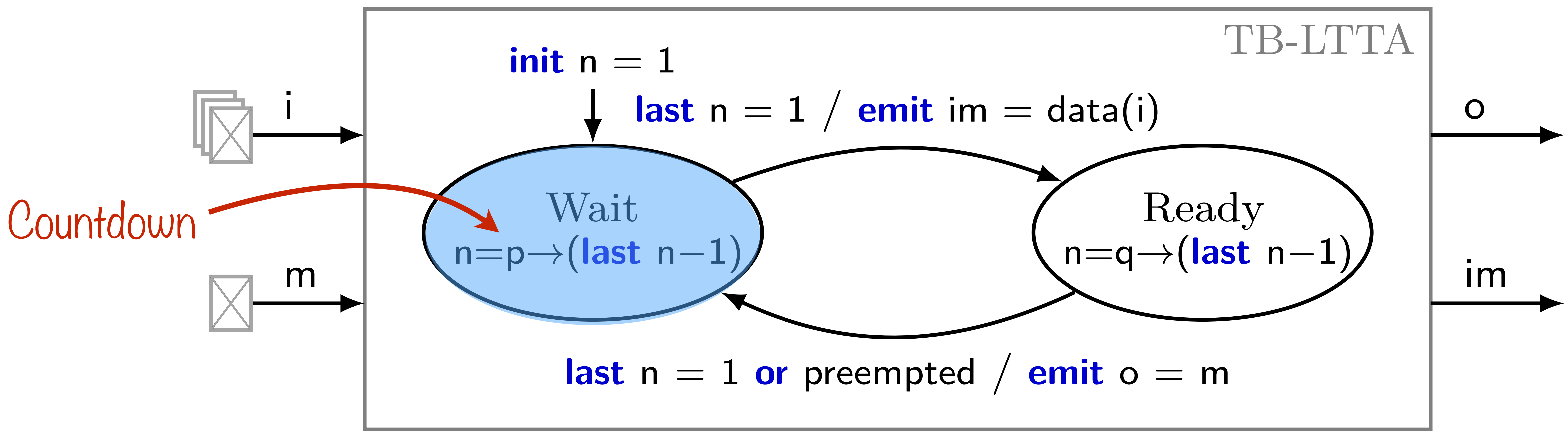
Time-Based LTTA

Wait: await the publication of the slowest node.
Ready: resynchronize with the fastest node.



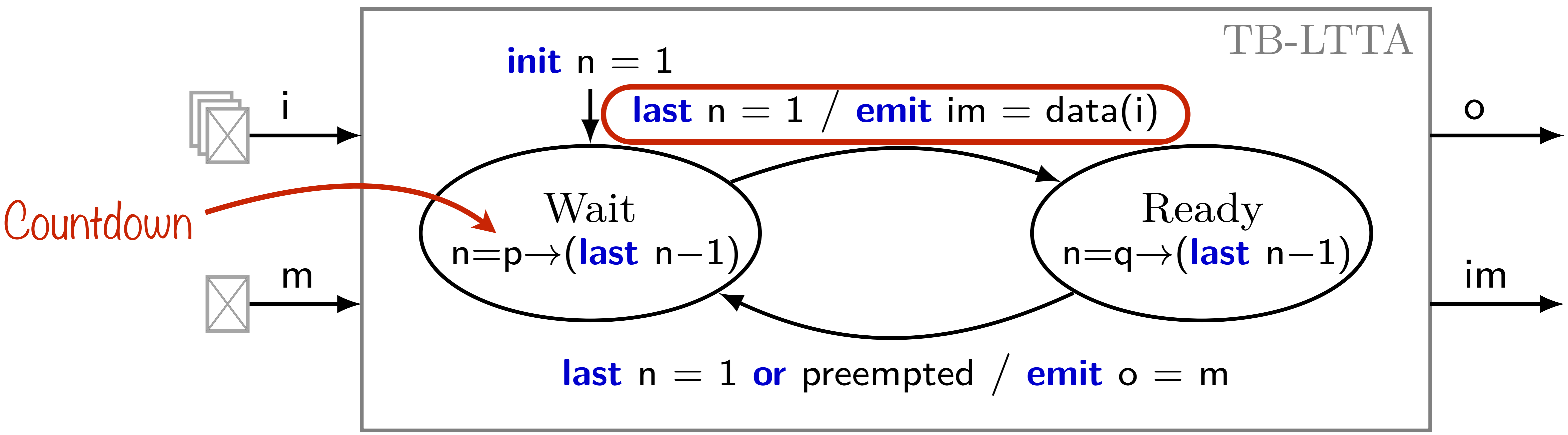
Time-Based LTTA

Wait: await the publication of the slowest node.
Ready: resynchronize with the fastest node.



Time-Based LTTA

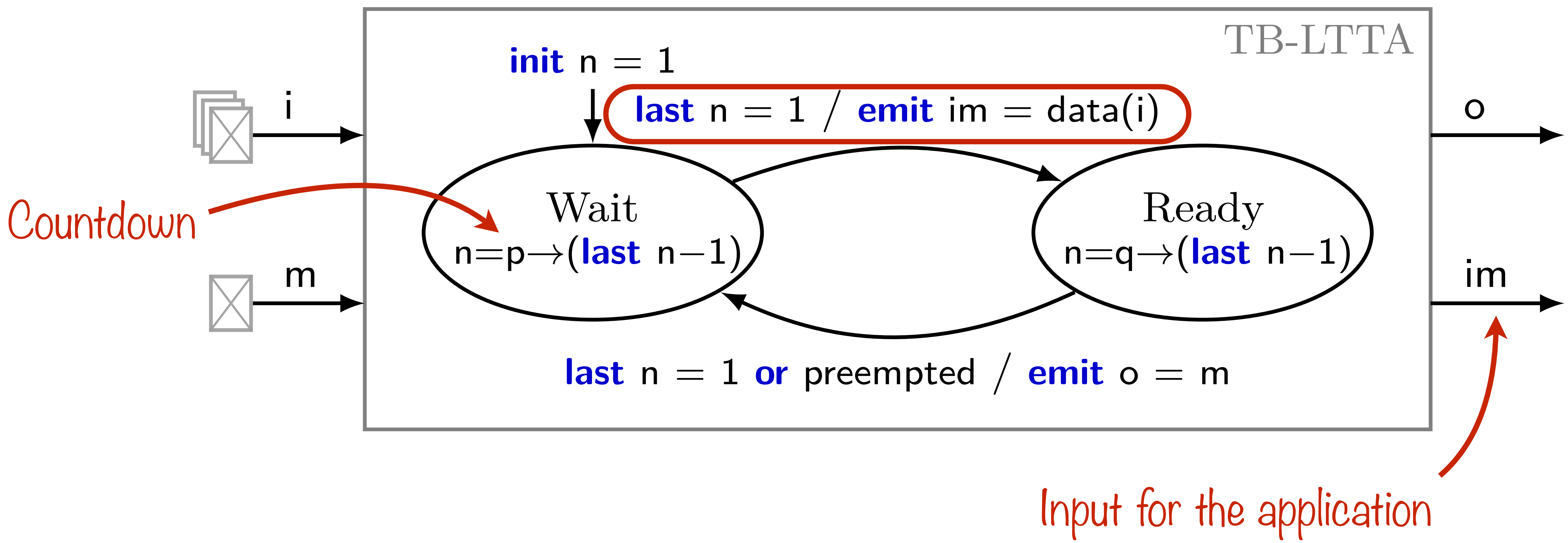
Wait: await the publication of the slowest node.
Ready: resynchronize with the fastest node.



Time-Based LTTA

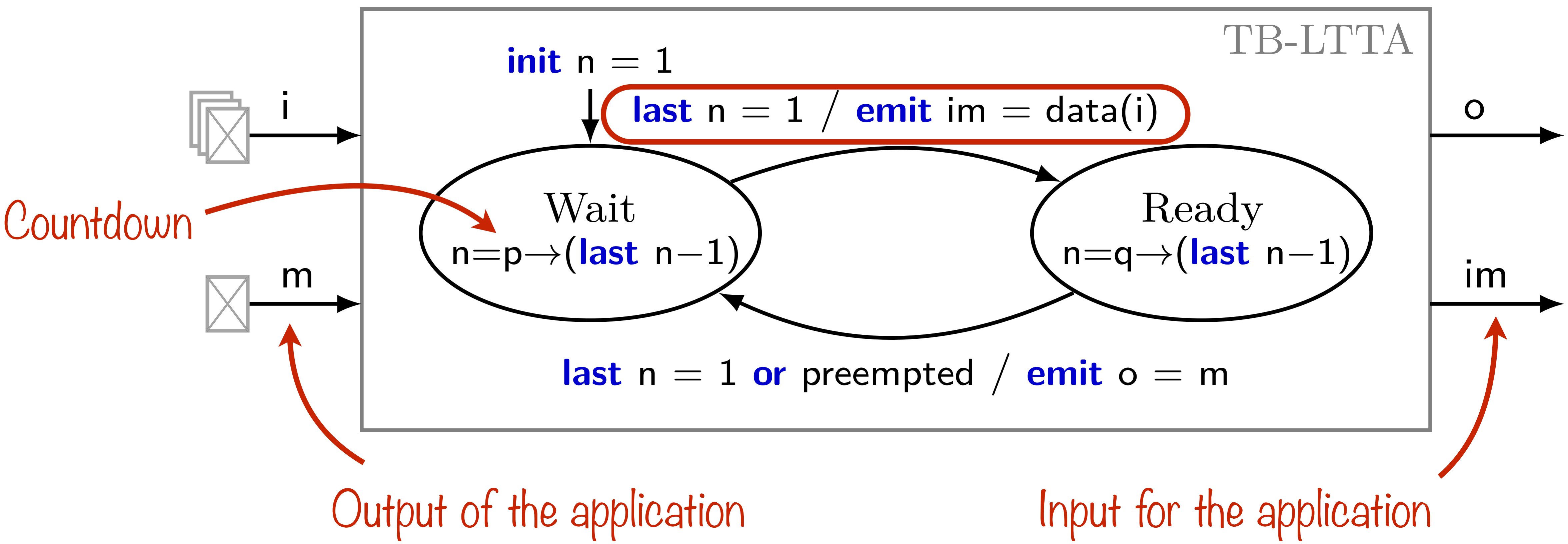
Wait: await the publication of the slowest node.

Ready: resynchronize with the fastest node.



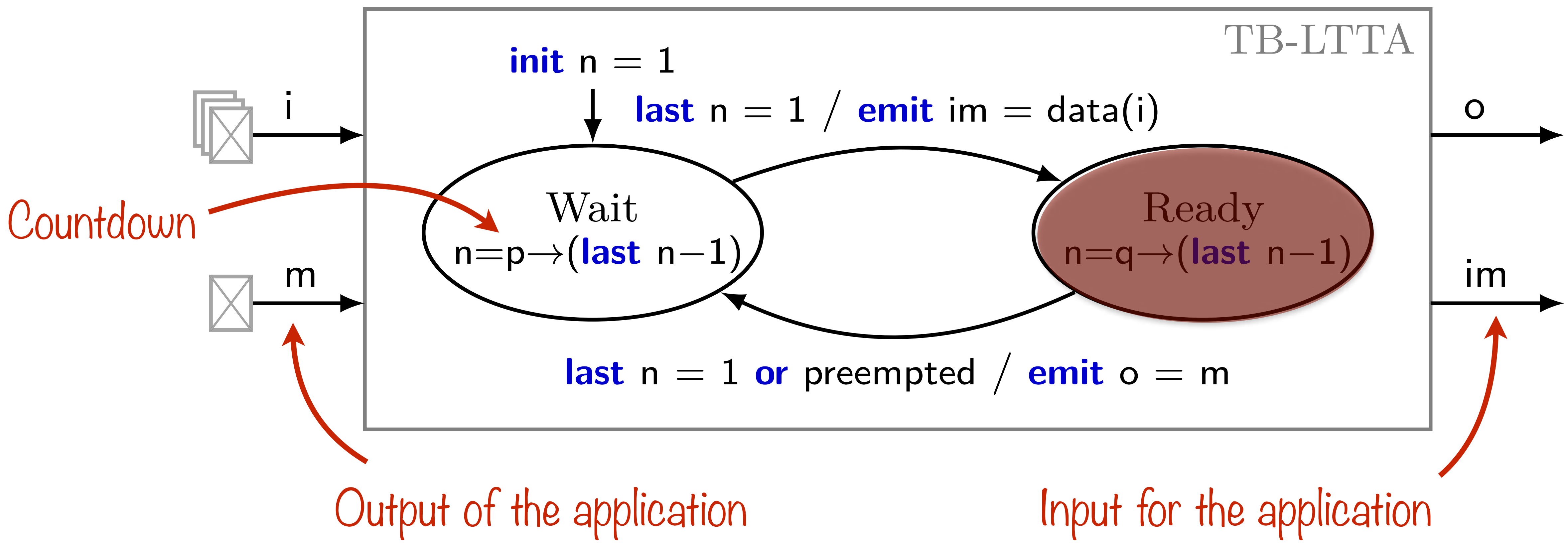
Time-Based LTTA

Wait: await the publication of the slowest node.
Ready: resynchronize with the fastest node.



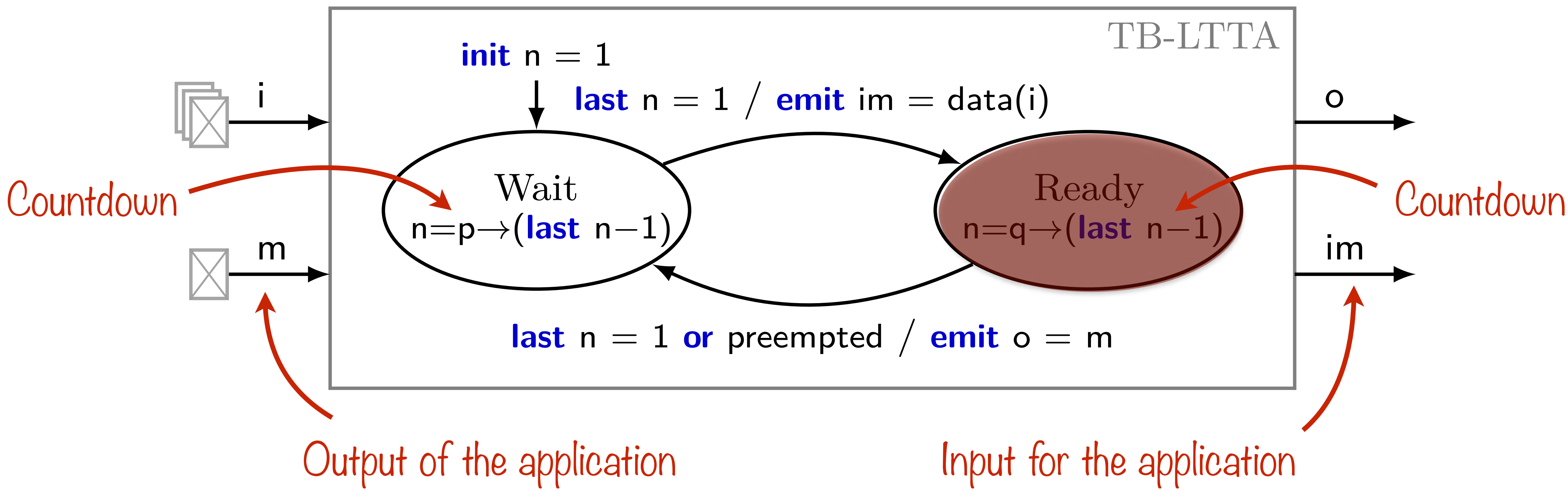
Time-Based LTTA

Wait: await the publication of the slowest node.
Ready: resynchronize with the fastest node.



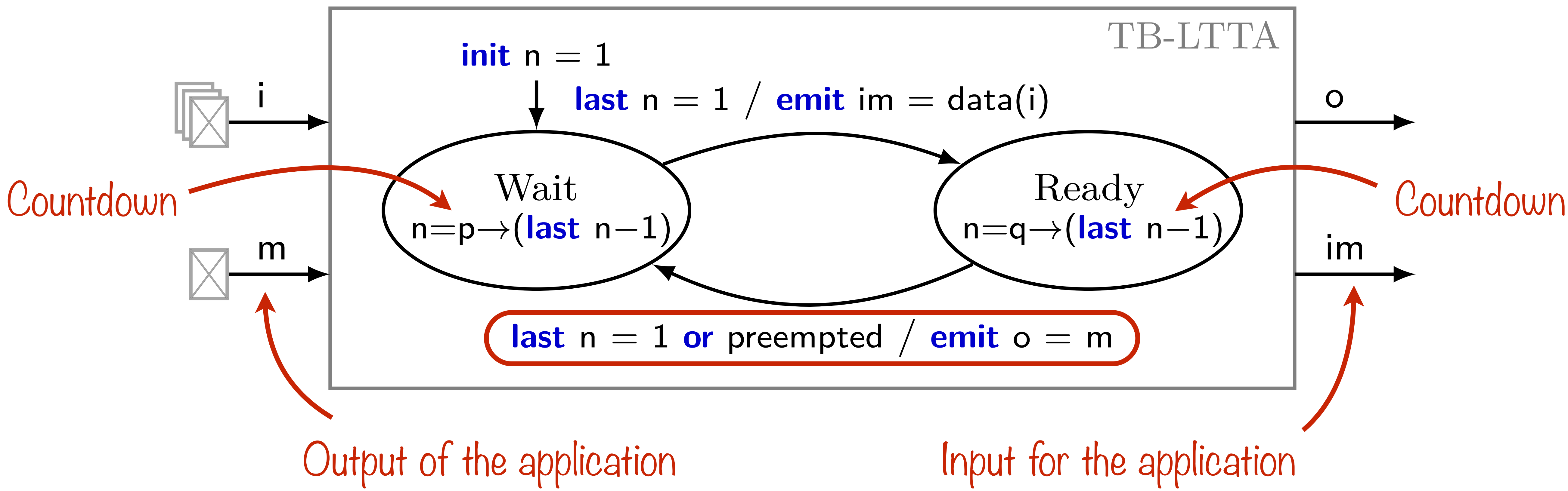
Time-Based LTTA

Wait: await the publication of the slowest node.
Ready: resynchronize with the fastest node.



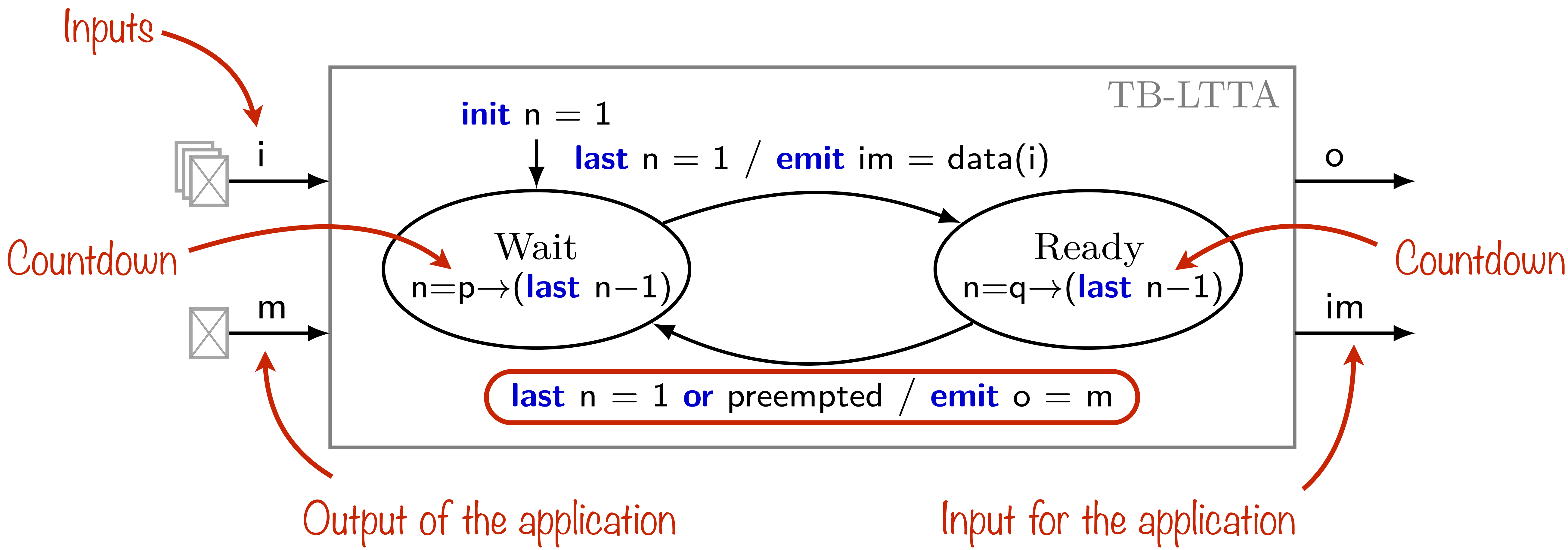
Time-Based LTTA

Wait: await the publication of the slowest node.
Ready: resynchronize with the fastest node.



Time-Based LTTA

Wait: await the publication of the slowest node.
Ready: resynchronize with the fastest node.

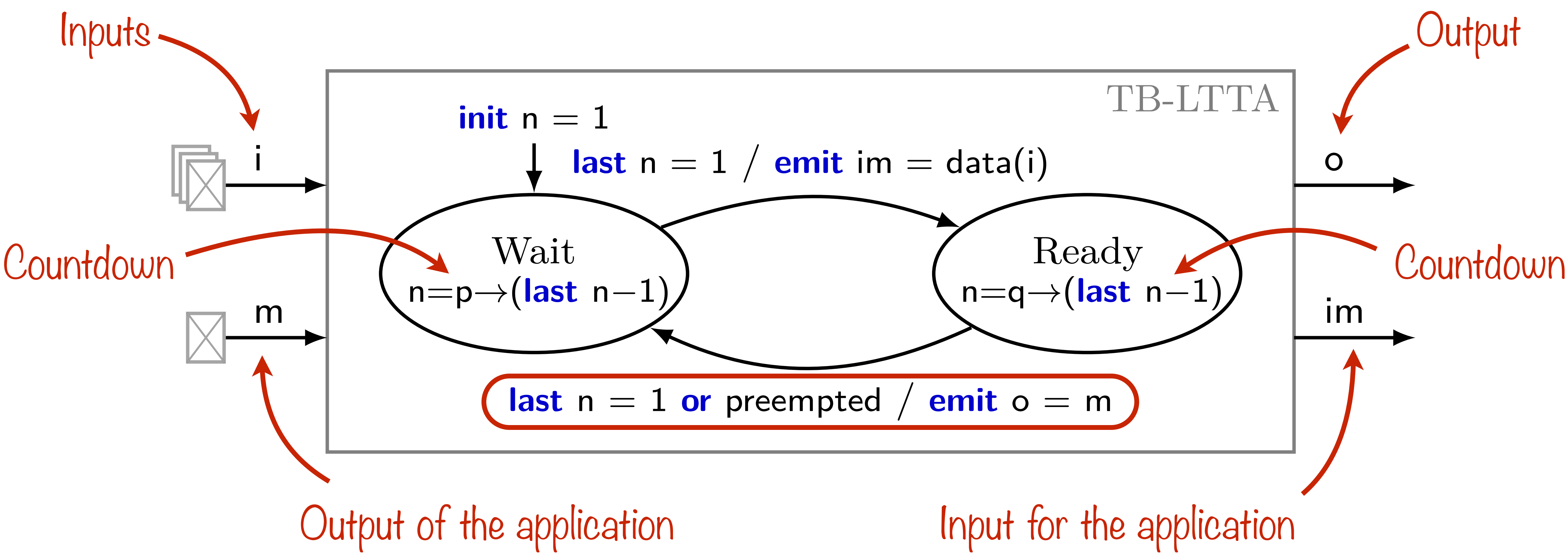


Preemption: when a publication is detected, the consumers have finished executing

Time-Based LTTA

Wait: await the publication of the slowest node.

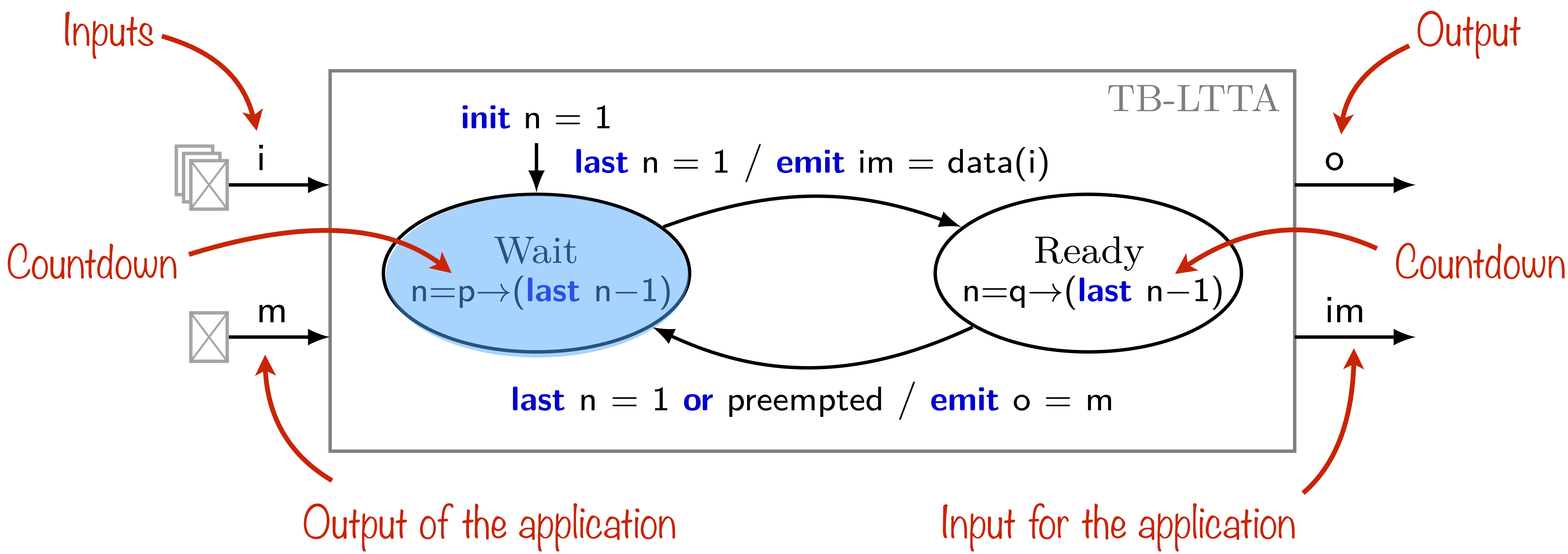
Ready: resynchronize with the fastest node.



Preemption: when a publication is detected, the consumers have finished executing

Time-Based LTTA

Wait: await the publication of the slowest node.
Ready: resynchronize with the fastest node.

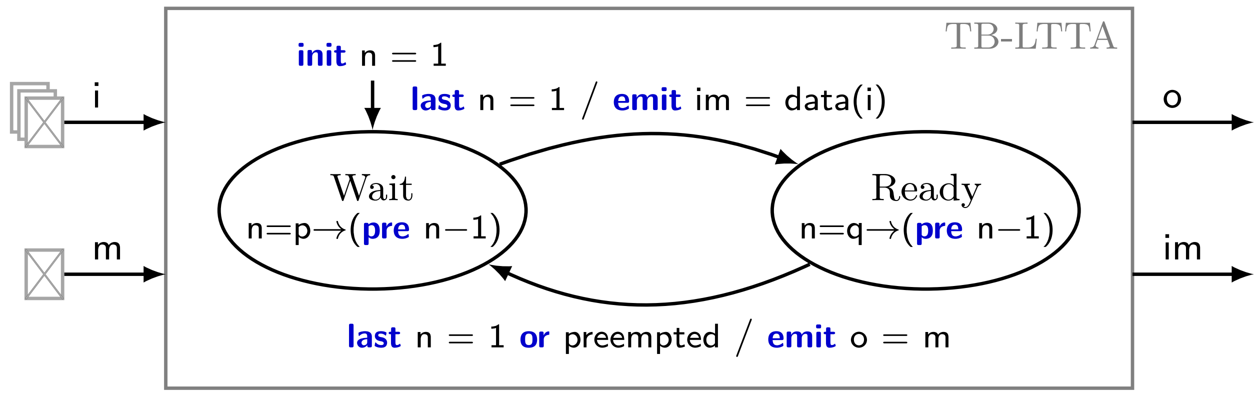


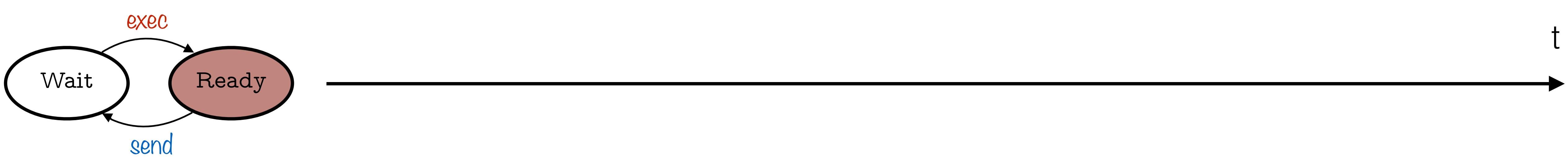
Preemption: when a publication is detected, the consumers have finished executing

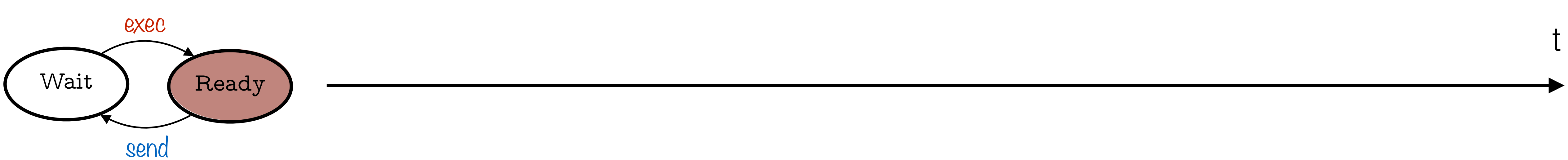
Time-Based LTTA

Simulation p=3 q=2

- Send
- Exec

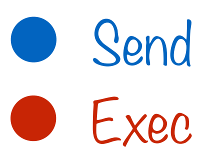


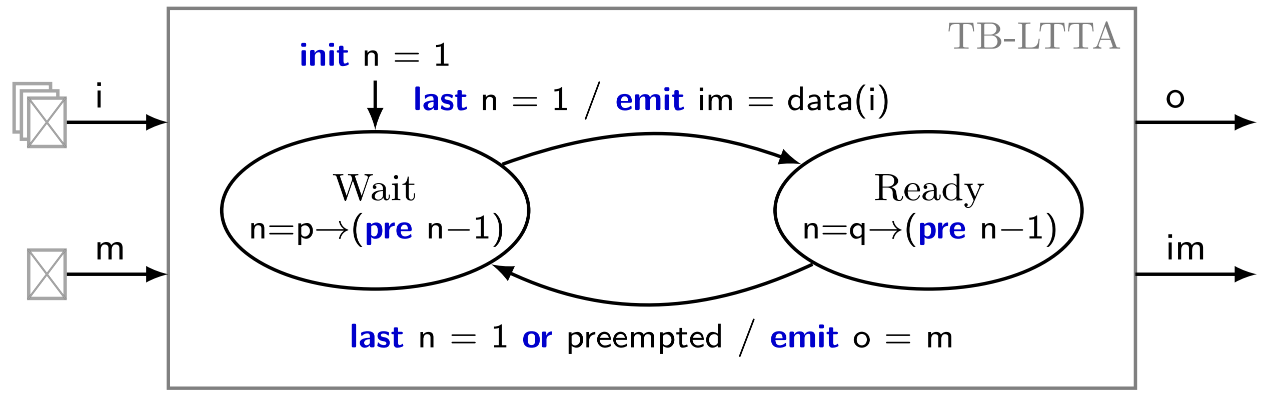


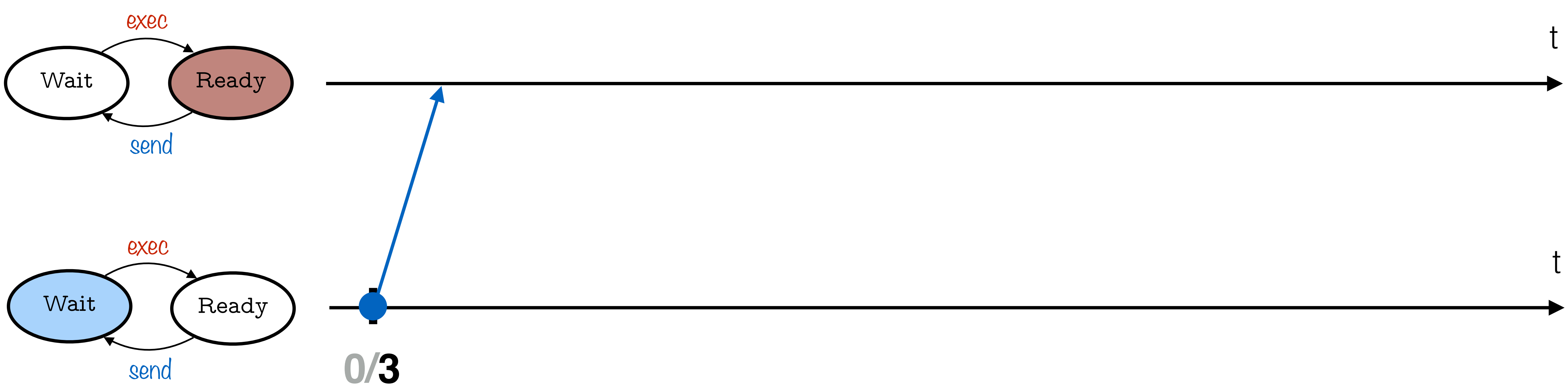


Time-Based LTTA

Simulation p=3 q=2



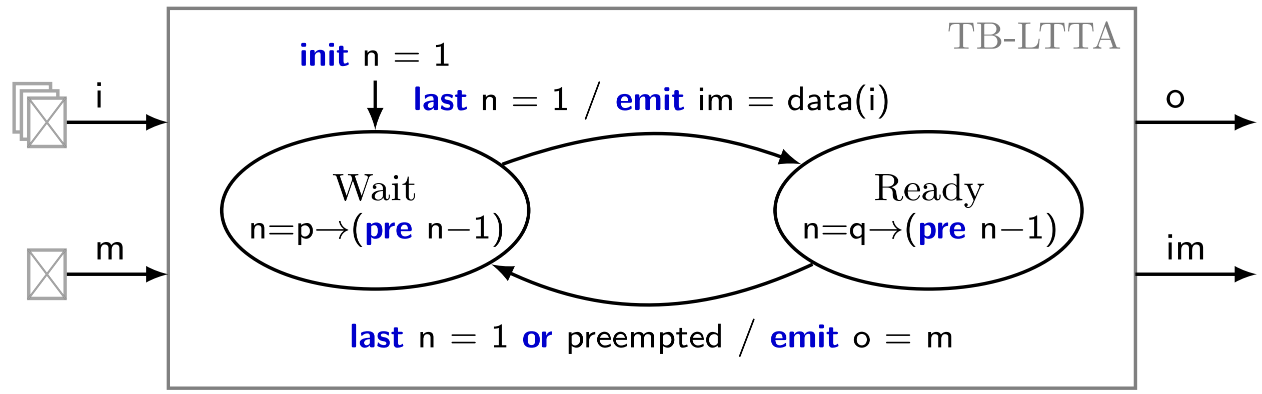


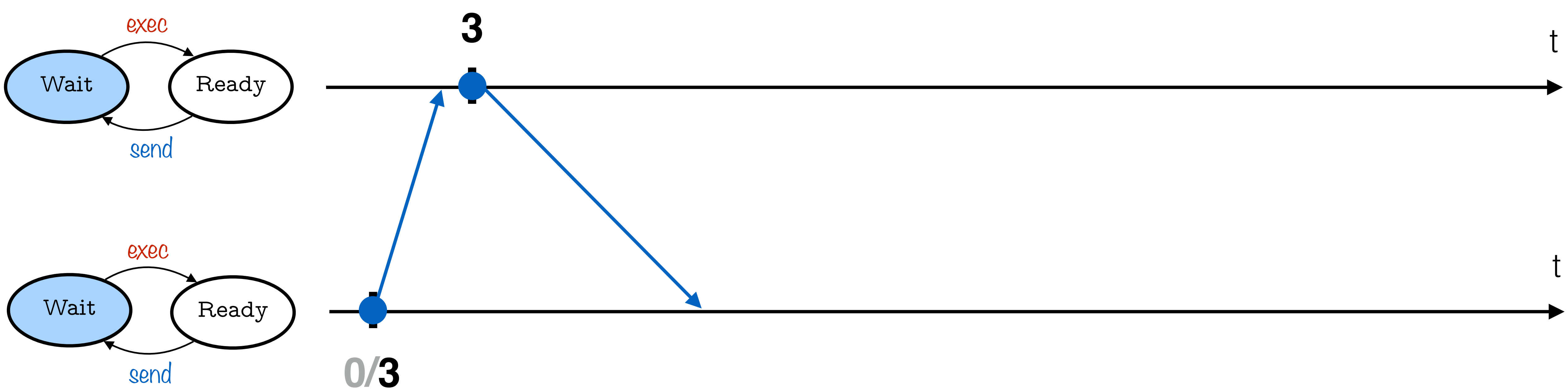


Time-Based LTTA

Simulation p=3 q=2

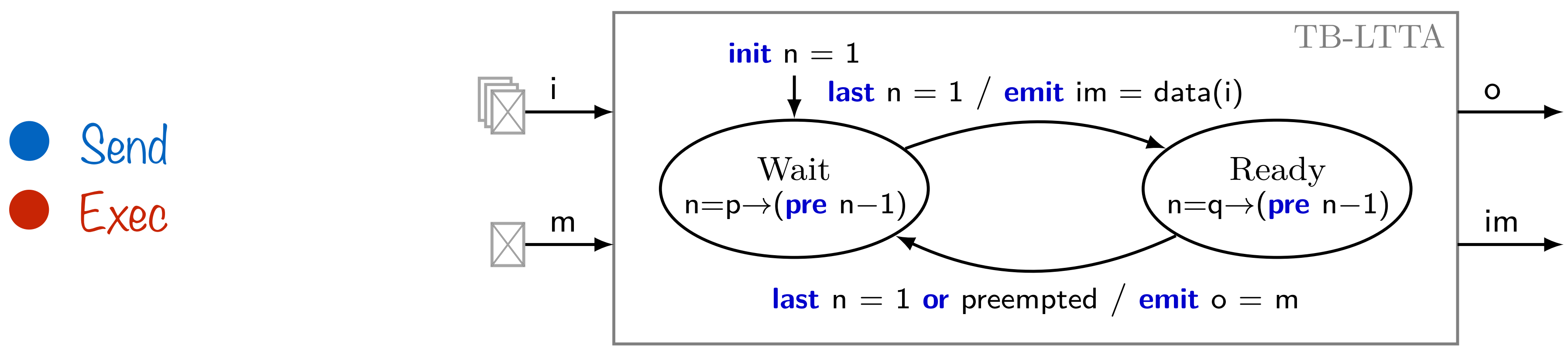
Send
Exec

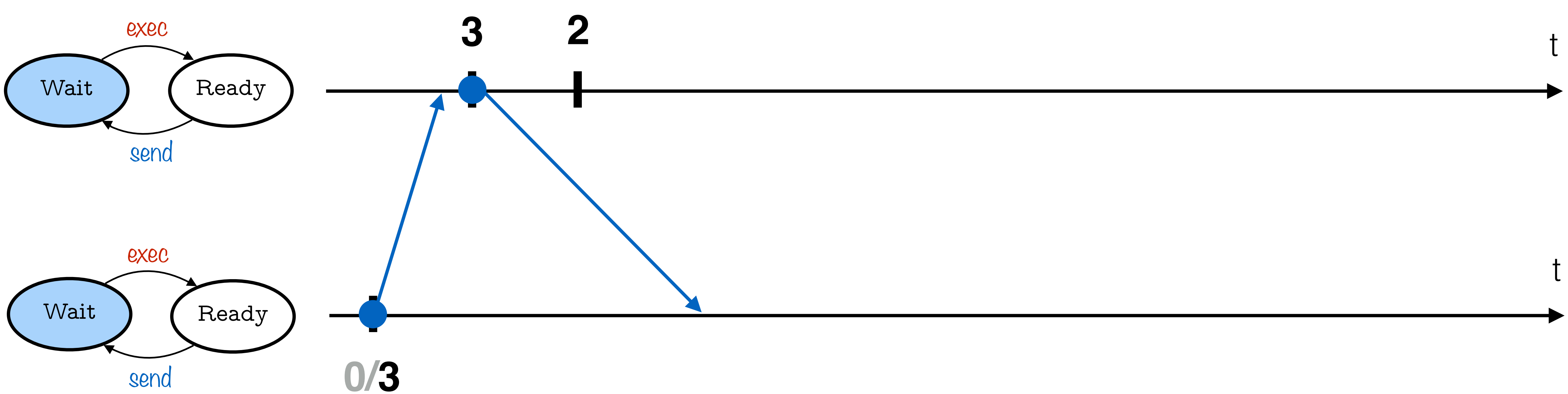




Time-Based LTTA

Simulation p=3 q=2

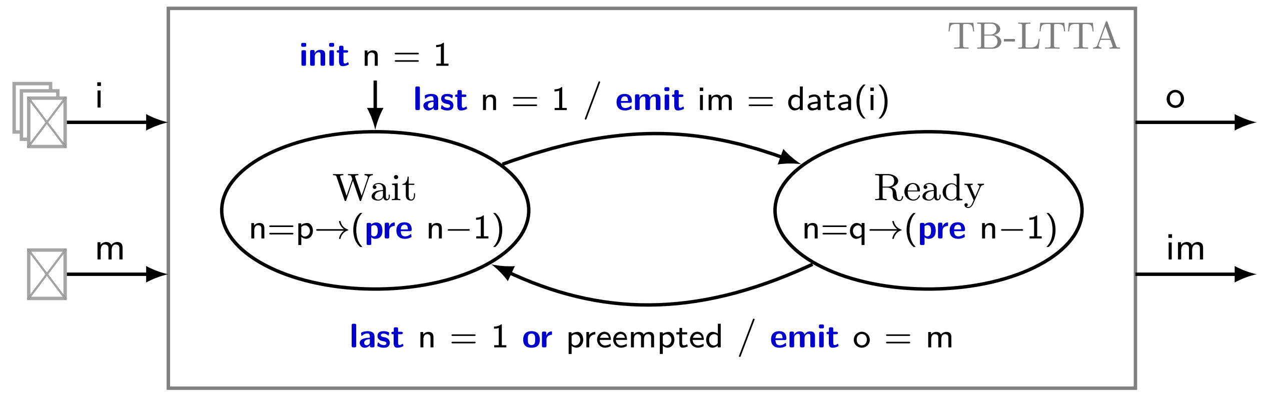


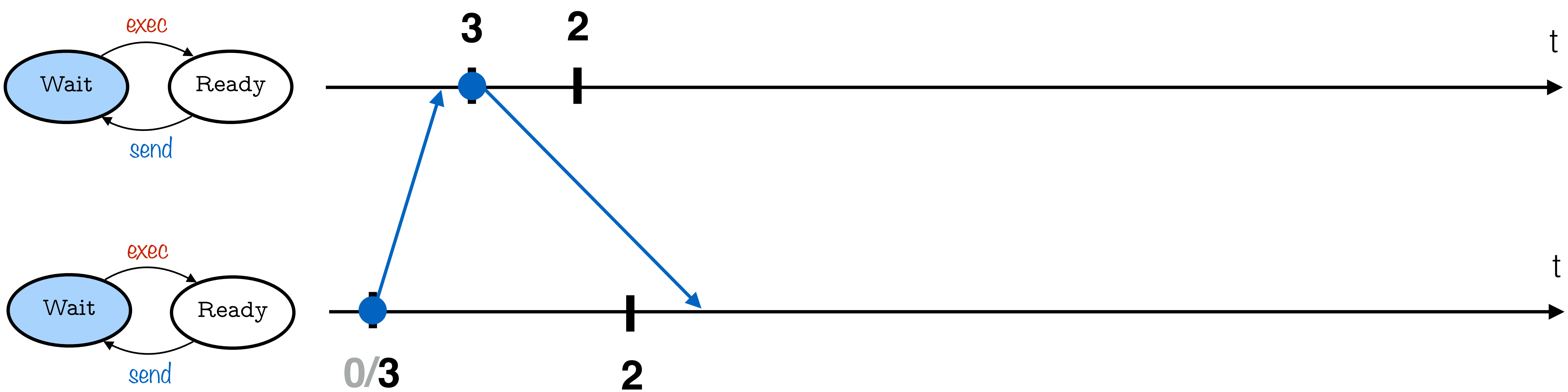


Time-Based LTTA

Simulation p=3 q=2

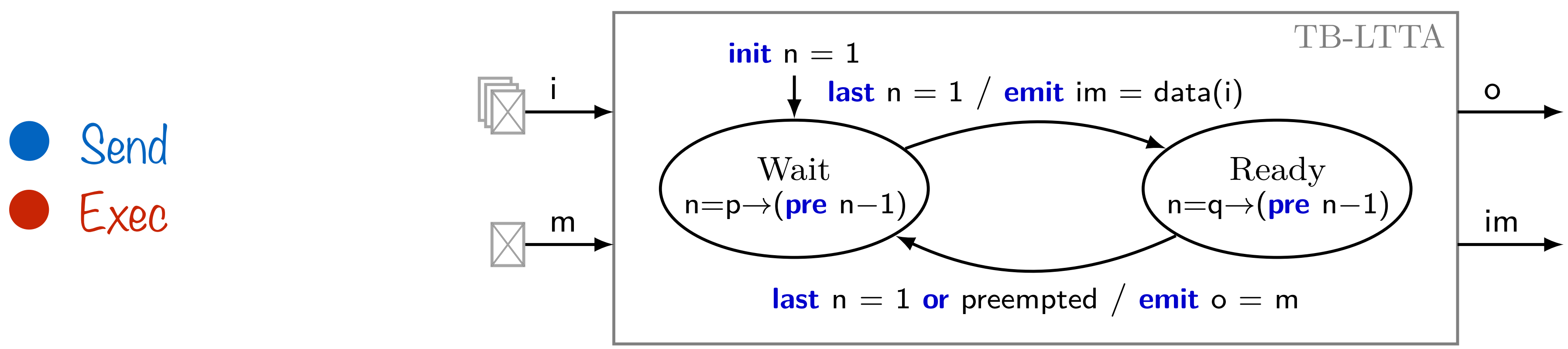
Send
Exec

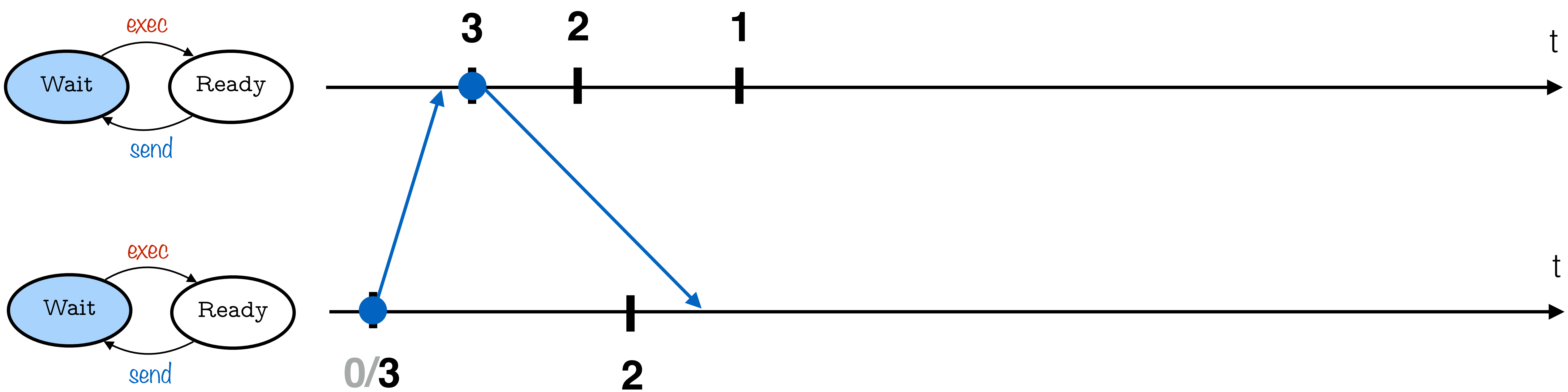




Time-Based LTTA

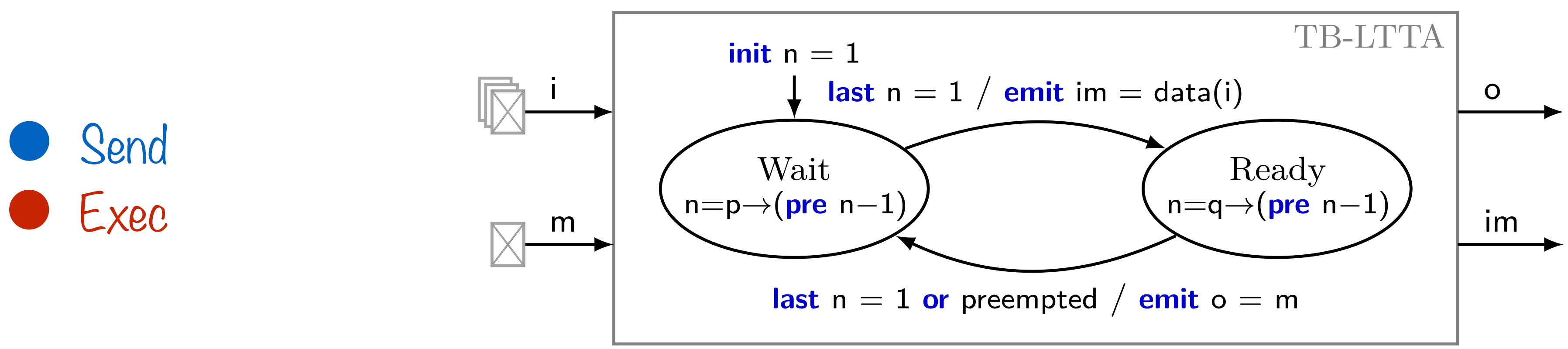
Simulation p=3 q=2

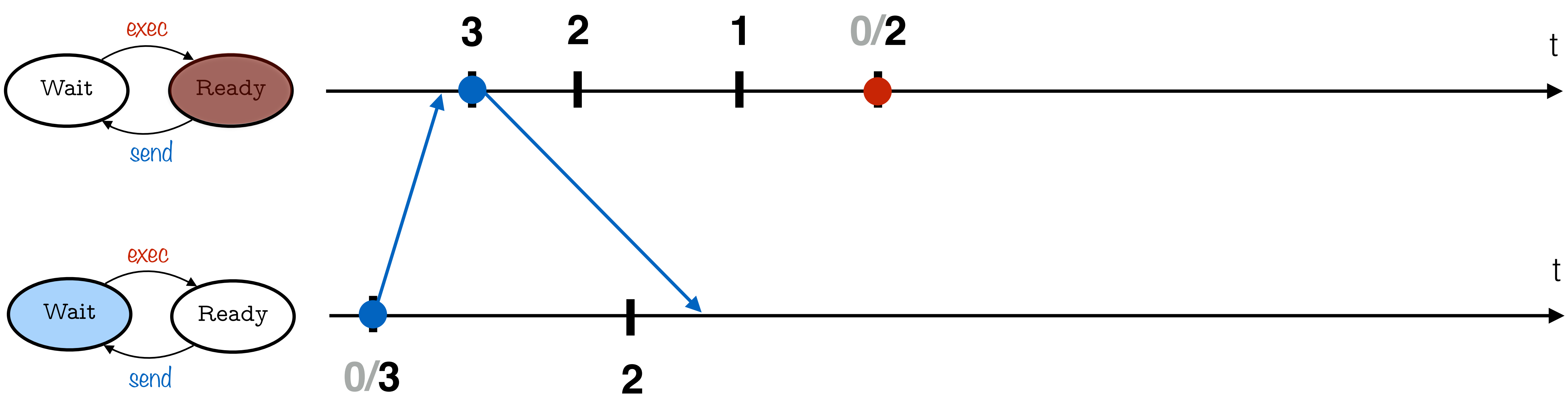




Time-Based LTTA

Simulation p=3 q=2

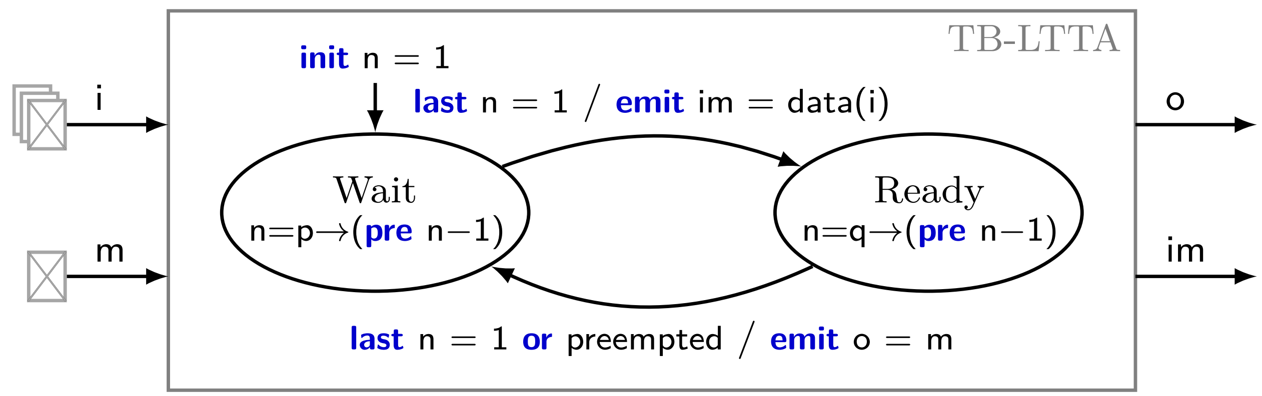


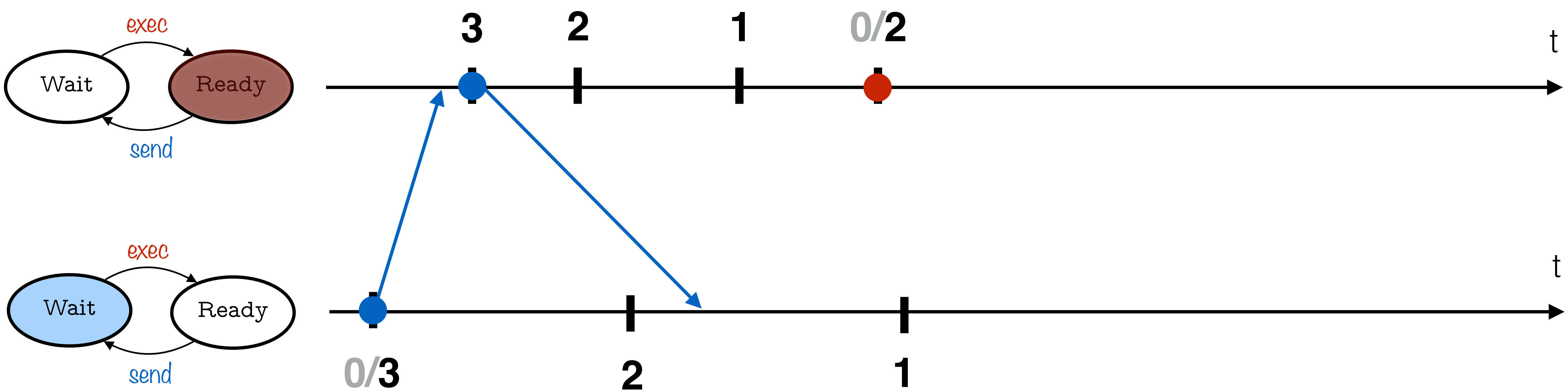


Time-Based LTTA

Simulation p=3 q=2

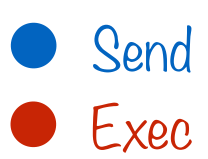
Send
Exec

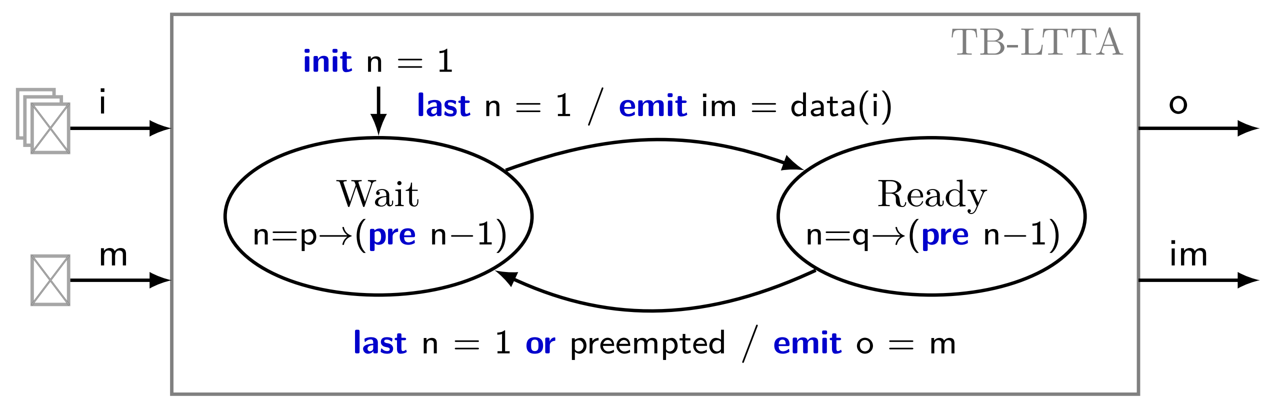


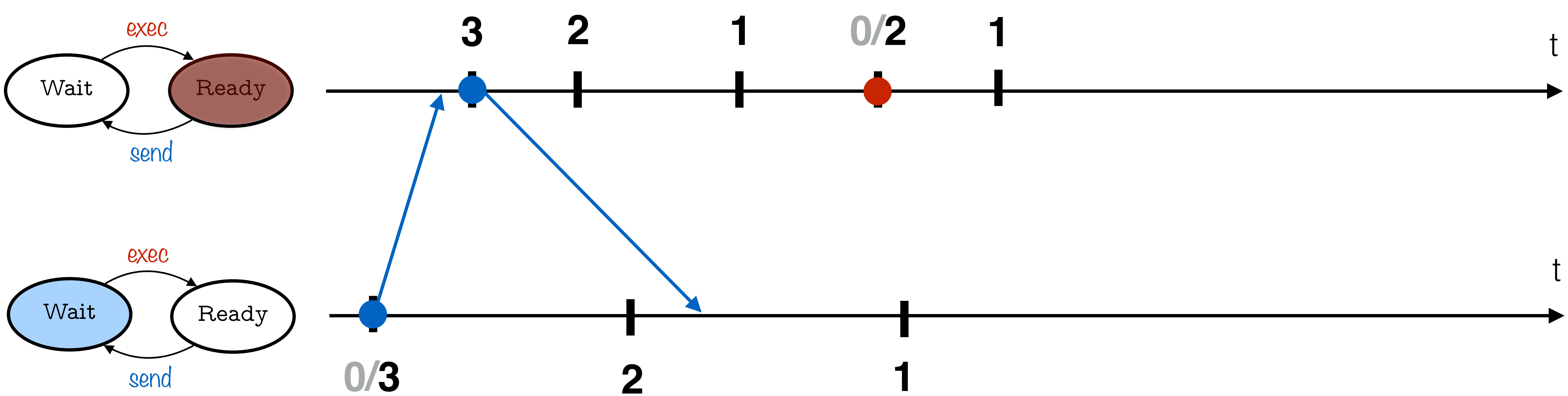


Time-Based LTTA

Simulation p=3 q=2



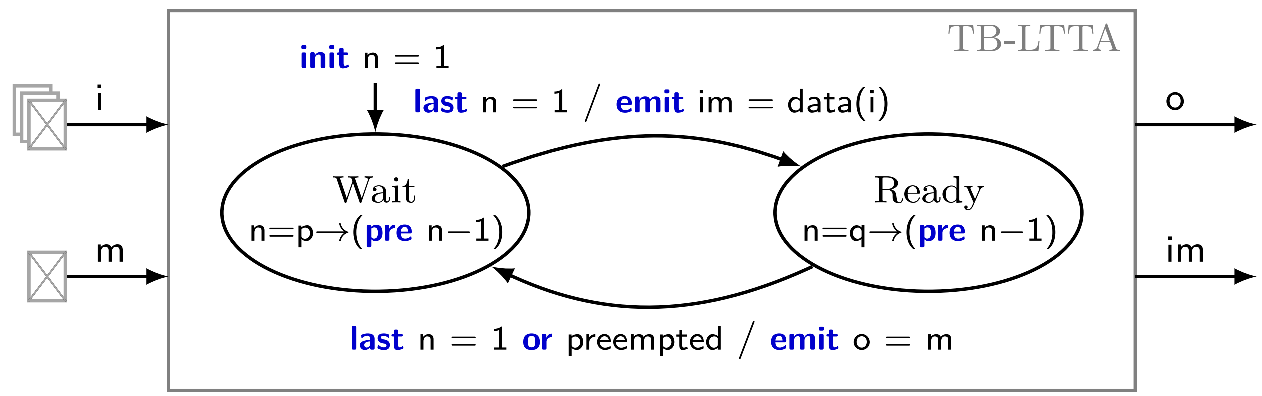


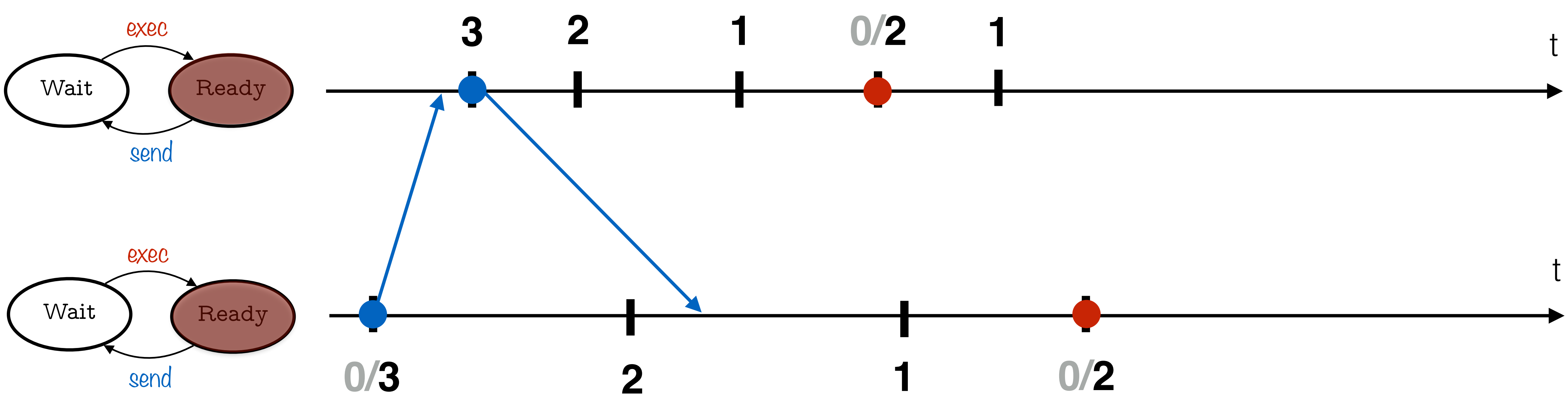


Time-Based LTTA

Simulation p=3 q=2

Send
Exec

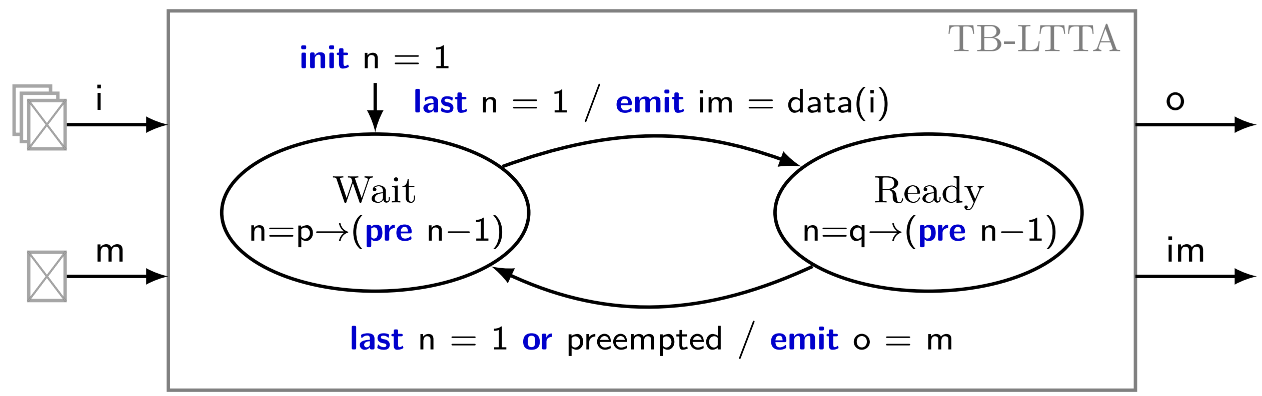


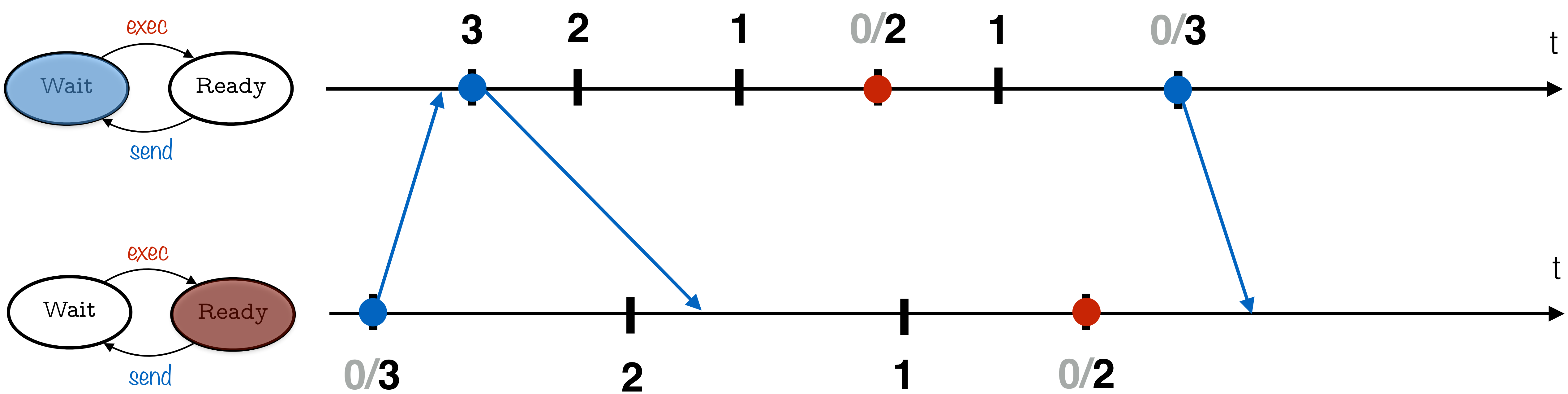


Time-Based LTTA

Simulation p=3 q=2

Send
Exec

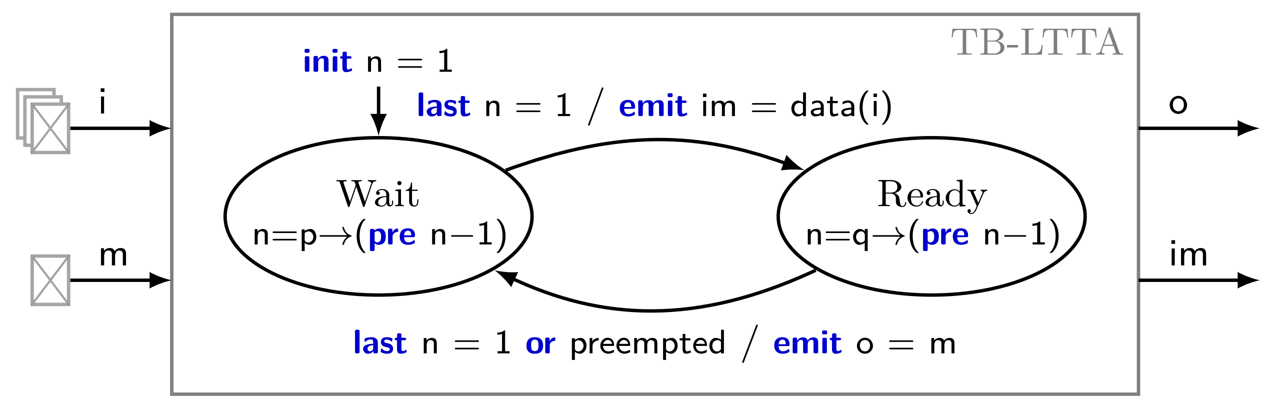


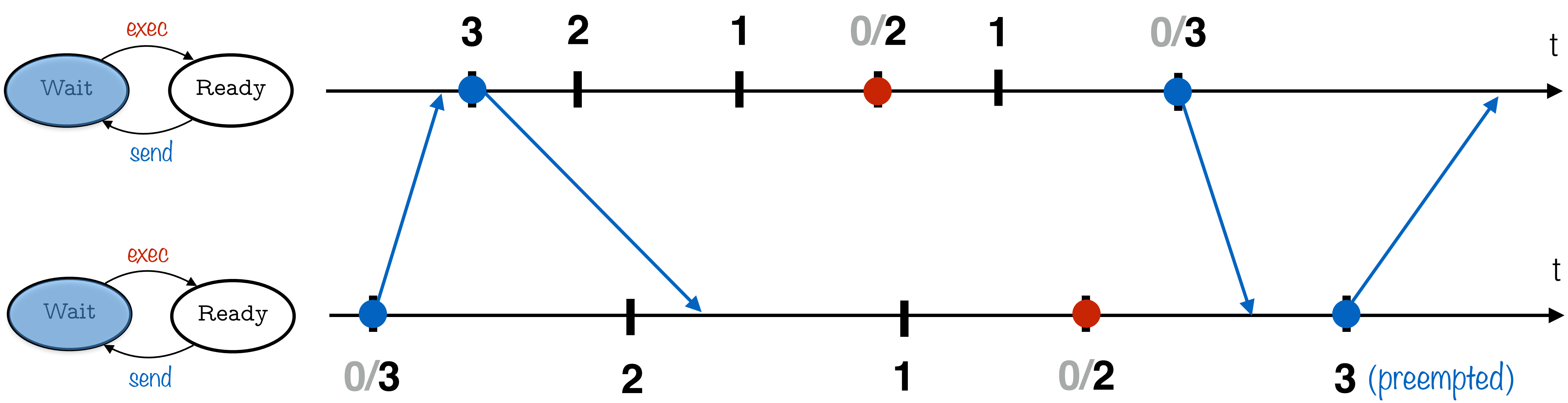


Time-Based LTTA

Simulation p=3 q=2

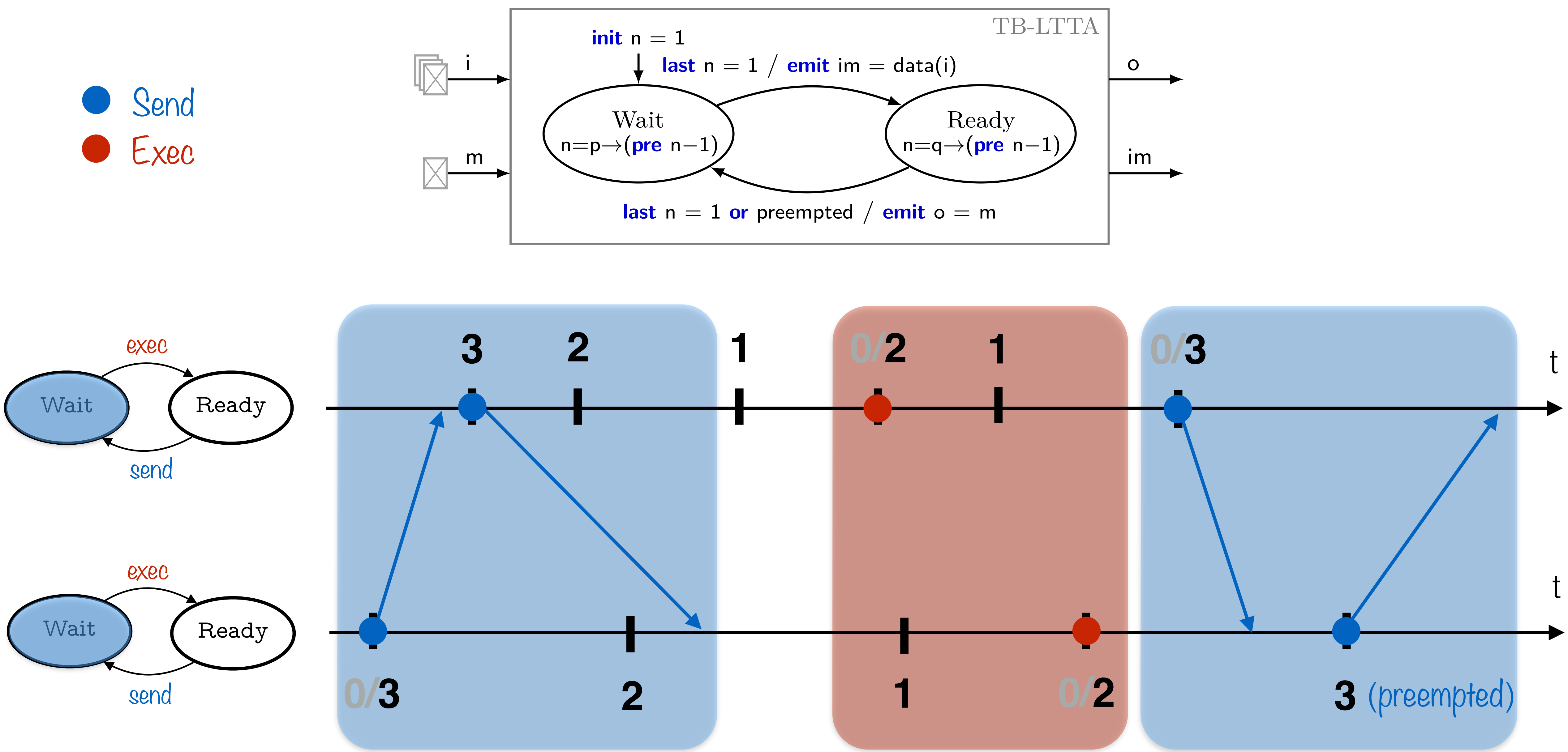
Send
Exec





Time-Based LTTA

Simulation p=3 q=2



Nodes alternate between **send** and **exec** phases

Time-Based LTTA

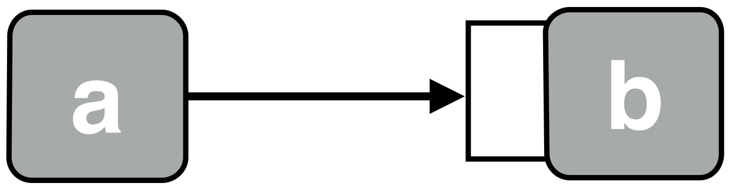
Theorem 1:
The composition of the controller and the application is always well-defined (no causality cycle).

Theorem 2:
The following constraints on the initial counter values ensure the preservation of the semantics

$$
\begin{aligned}
p &> \frac{2\tau_{max} + T_{max}}{T_{min}} \\
q &> \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p
\end{aligned}
$$

Theorem 3:
The worst case throughput is given by $1/\lambda_{\text{TB}} = (p+q)T_{\max}$

Preservation of the Semantics

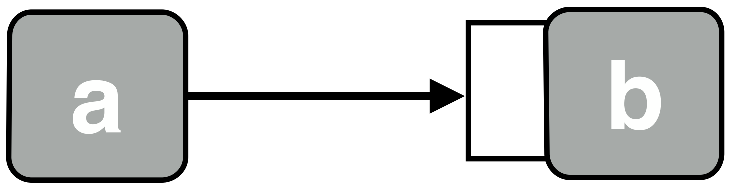


$$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$$

$$q > \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p$$

Property 1: $S^a_{k-1} \prec E^b_k$

Preservation of the Semantics



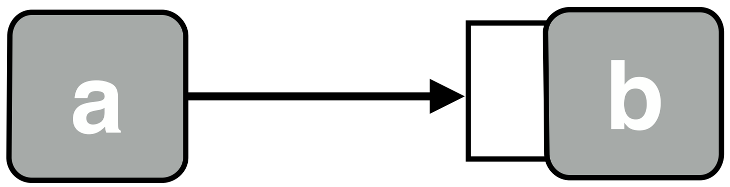
$$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$$

$$q > \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p$$

Property 1: $S^a_{k-1} \prec E^b_k$

S^b_{k-1}

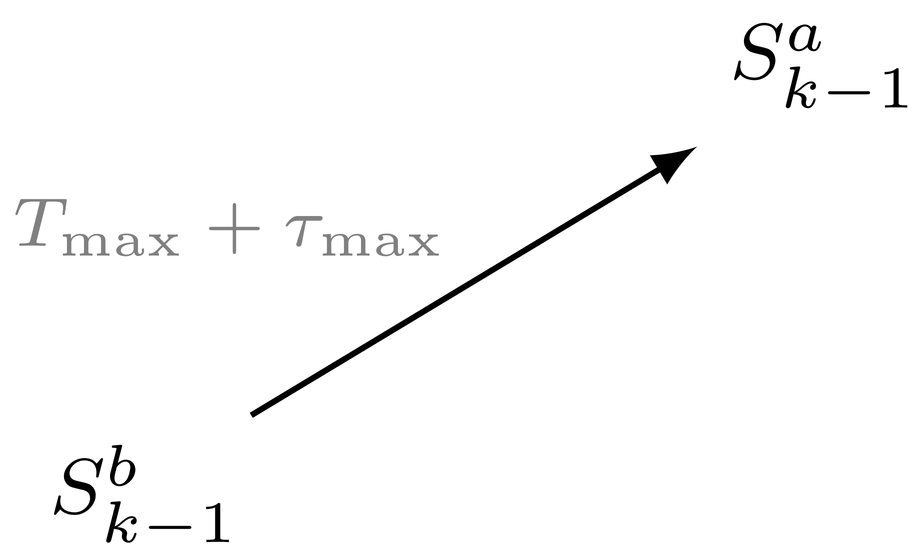
Preservation of the Semantics



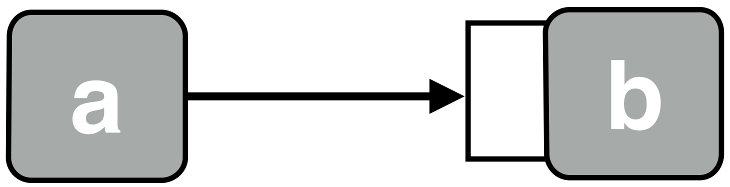
$$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$$

$$q > \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p$$

Property 1: $S^a_{k-1} \prec E^b_k$



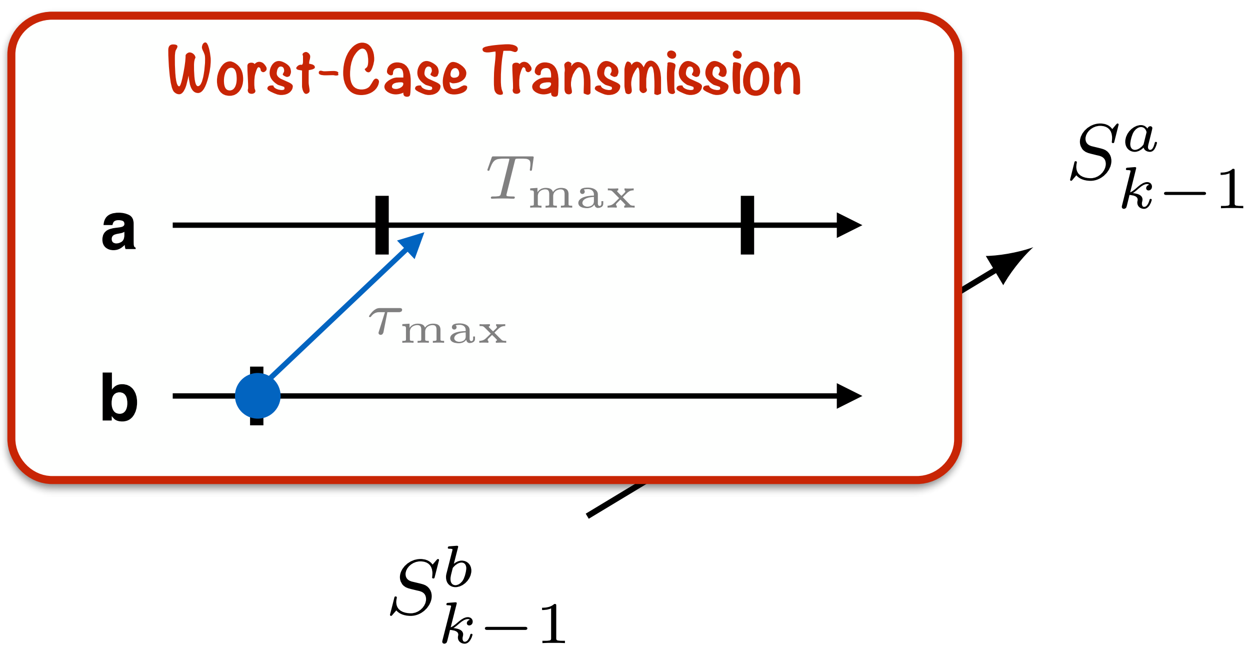
Preservation of the Semantics



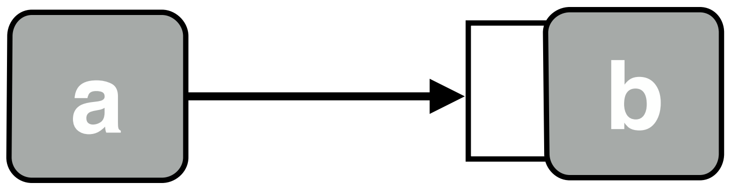
$$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$$

$$q > \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p$$

Property 1: $S^a_{k-1} \prec E^b_k$



Preservation of the Semantics



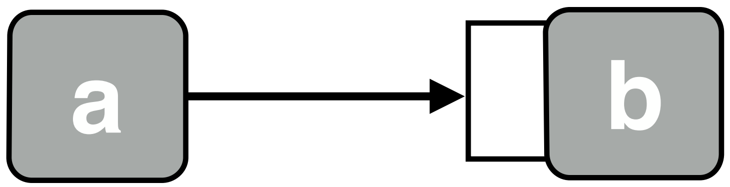
$$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$$

$$q > \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p$$

Property 1: $S^a_{k-1} \prec E^b_k$

$$S^b_{k-1} \xrightarrow{T_{\max} + \tau_{\max}} S^a_{k-1}$$

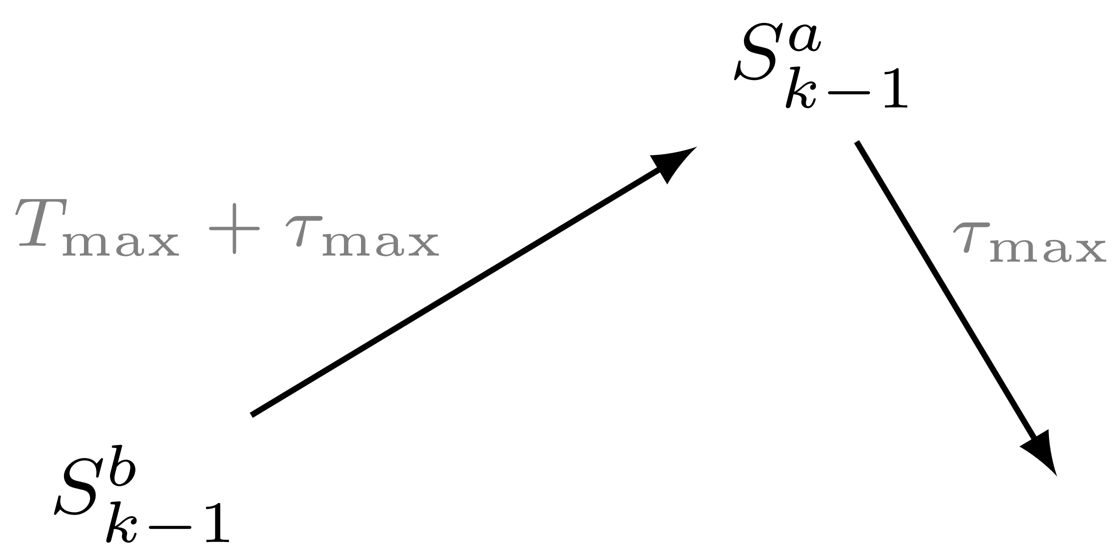
Preservation of the Semantics



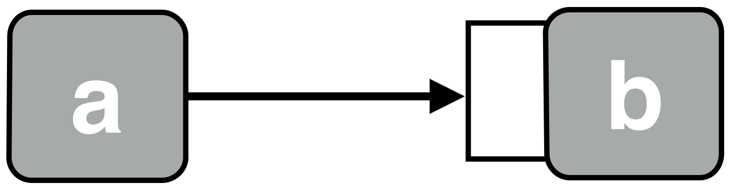
$$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$$

$$q > \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p$$

Property 1: $S^a_{k-1} \prec E^b_k$



Preservation of the Semantics



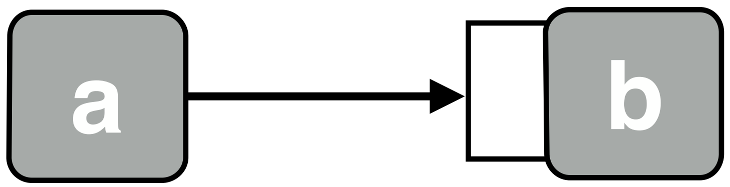
$$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$$

$$q > \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p$$

Property 1: $S^a_{k-1} \prec E^b_k$

$$\begin{tikzcd}
 & S^a_{k-1} \arrow[dr, "\tau_{\max}"] & \\
S^b_{k-1} \arrow[ur, "T_{\max} + \tau_{\max}"] \arrow[rr, "pT_{\min}"'] & & E^b_k
\end{tikzcd}$$

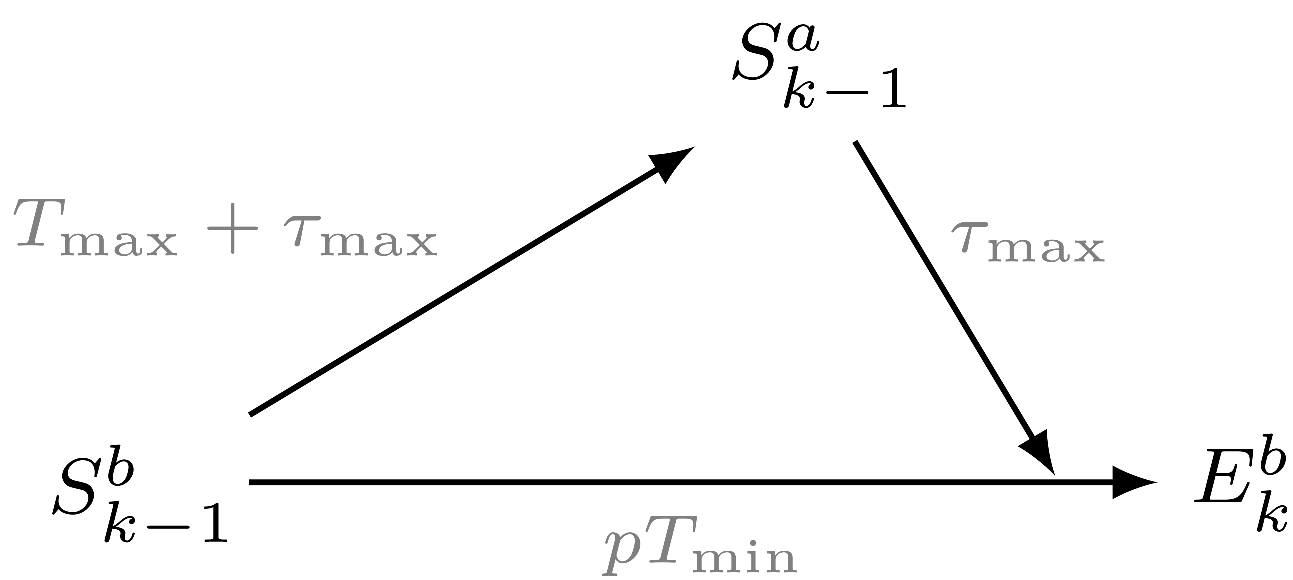
Preservation of the Semantics



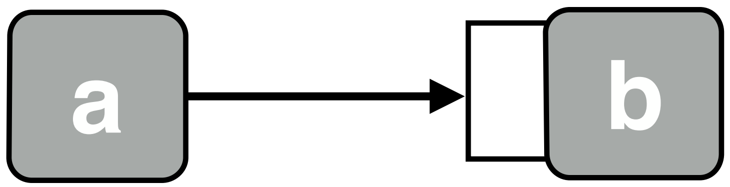
$$p \quad > \quad \frac{2\tau_{max} + T_{max}}{T_{min}}$$

$$q \quad > \quad \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p$$

Property 1: $S^a_{k-1} \prec E^b_k$



Preservation of the Semantics



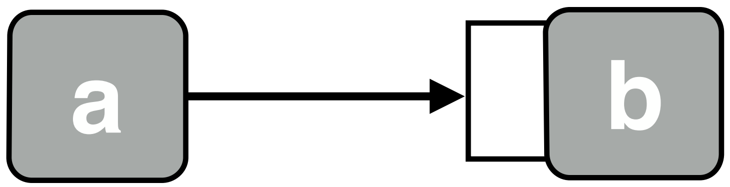
$$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$$

$$q > \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p$$

Property 1: $S^a_{k-1} \prec E^b_k$

Property 2: $E^b_k \prec S^a_k$

Preservation of the Semantics



$$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$$

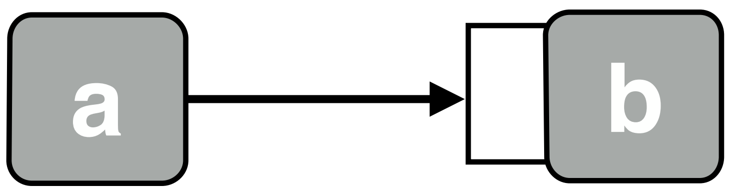
$$q > \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p$$

Property 1: $S^a_{k-1} \prec E^b_k$

Property 2: $E^b_k \prec S^a_k$

S^a_{k-1}

Preservation of the Semantics



$$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$$

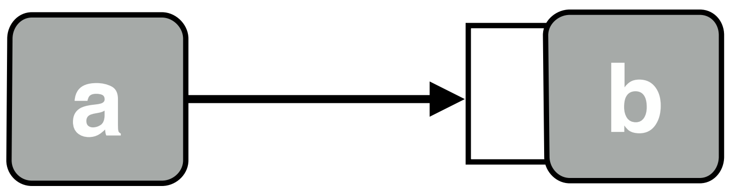
$$q > \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p$$

Property 1: $S^a_{k-1} \prec E^b_k$

Property 2: $E^b_k \prec S^a_k$

$$S^a_{k-1} \xrightarrow{pT_{\min}} E^a_k$$

Preservation of the Semantics



$$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$$

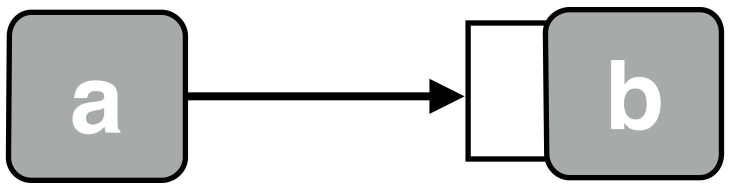
$$q > \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p$$

Property 1: $S^a_{k-1} \prec E^b_k$

Property 2: $E^b_k \prec S^a_k$

$$S^a_{k-1} \xrightarrow{pT_{\min}} E^a_k \xrightarrow{qT_{\min}} S^a_k$$

Preservation of the Semantics



$$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$$

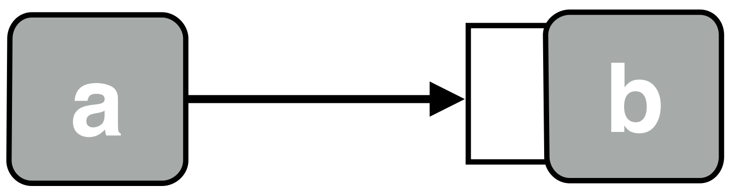
$$q > \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p$$

Property 1: $S^a_{k-1} \prec E^b_k$

Property 2: $E^b_k \prec S^a_k$

$$\begin{tikzcd} S^a_{k-1} \arrow[r, "pT_{\min}"] & E^a_k \arrow[r, "qT_{\min}"] & S^a_k \arrow[dr, "\tau_{\min}"] & \\ & & & {} \end{tikzcd}$$

Preservation of the Semantics

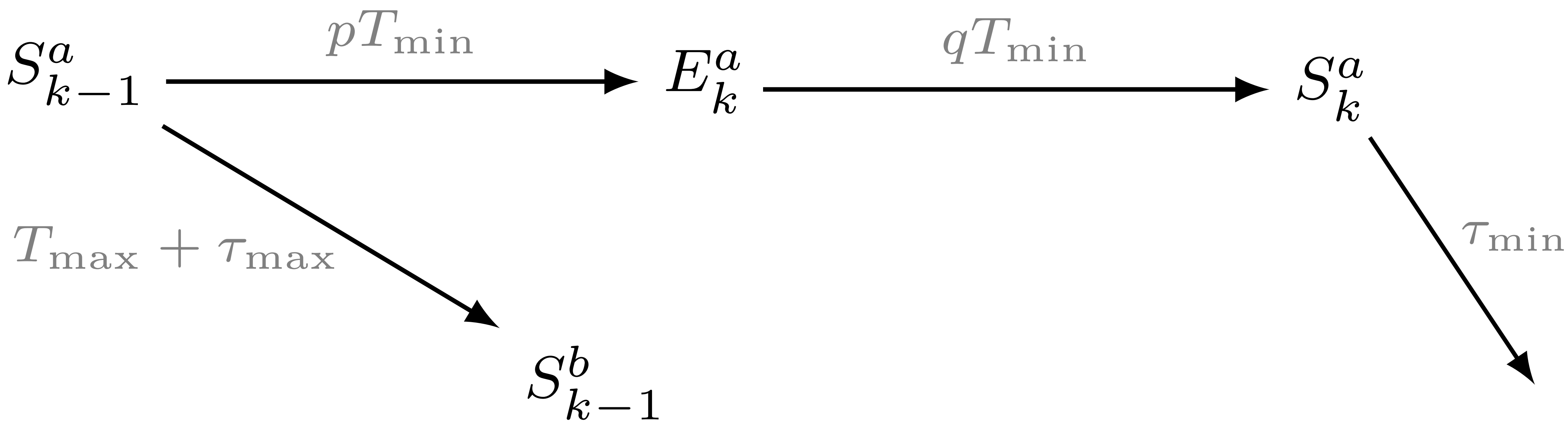


$$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$$

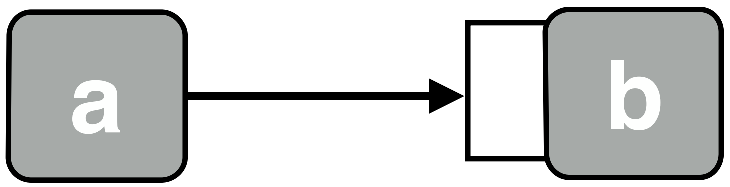
$$q > \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p$$

Property 1: $S^a_{k-1} \prec E^b_k$

Property 2: $E^b_k \prec S^a_k$



Preservation of the Semantics



$$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$$

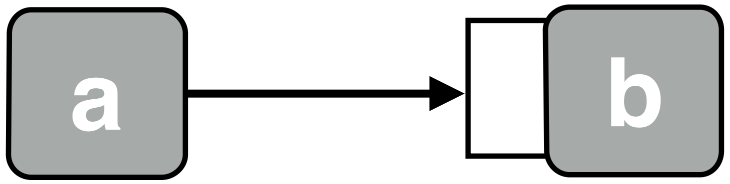
$$q > \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p$$

Property 1: $S^a_{k-1} \prec E^b_k$

Property 2: $E^b_k \prec S^a_k$

$$\begin{tikzcd}
S^a_{k-1} \arrow[rr, "pT_{\min}"] \arrow[dr, "T_{\max} + \tau_{\max}"'] & & E^a_k \arrow[rr, "qT_{\min}"] & & S^a_k \arrow[d, "\tau_{\min}"] \\
& S^b_{k-1} \arrow[rrr, "pT_{\max}"'] & & & E^b_k
\end{tikzcd}$$

Preservation of the Semantics

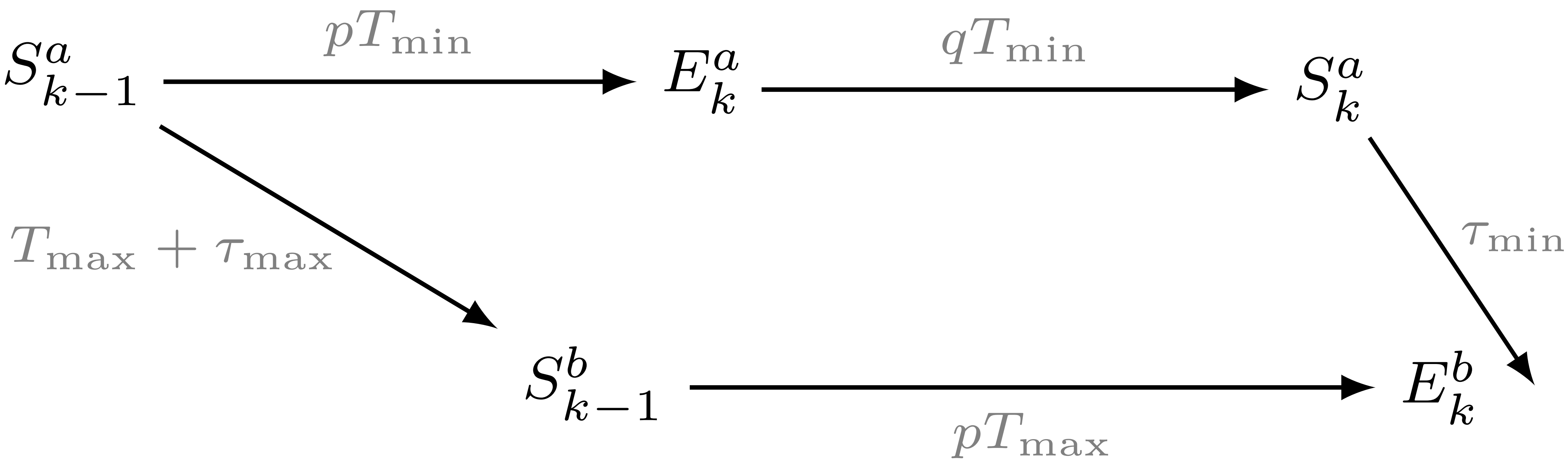


$$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$$

$$q > \frac{\tau_{max} - \tau_{min} + (p+1)T_{max}}{T_{min}} - p$$

Property 1: $S^a_{k-1} \prec E^b_k$

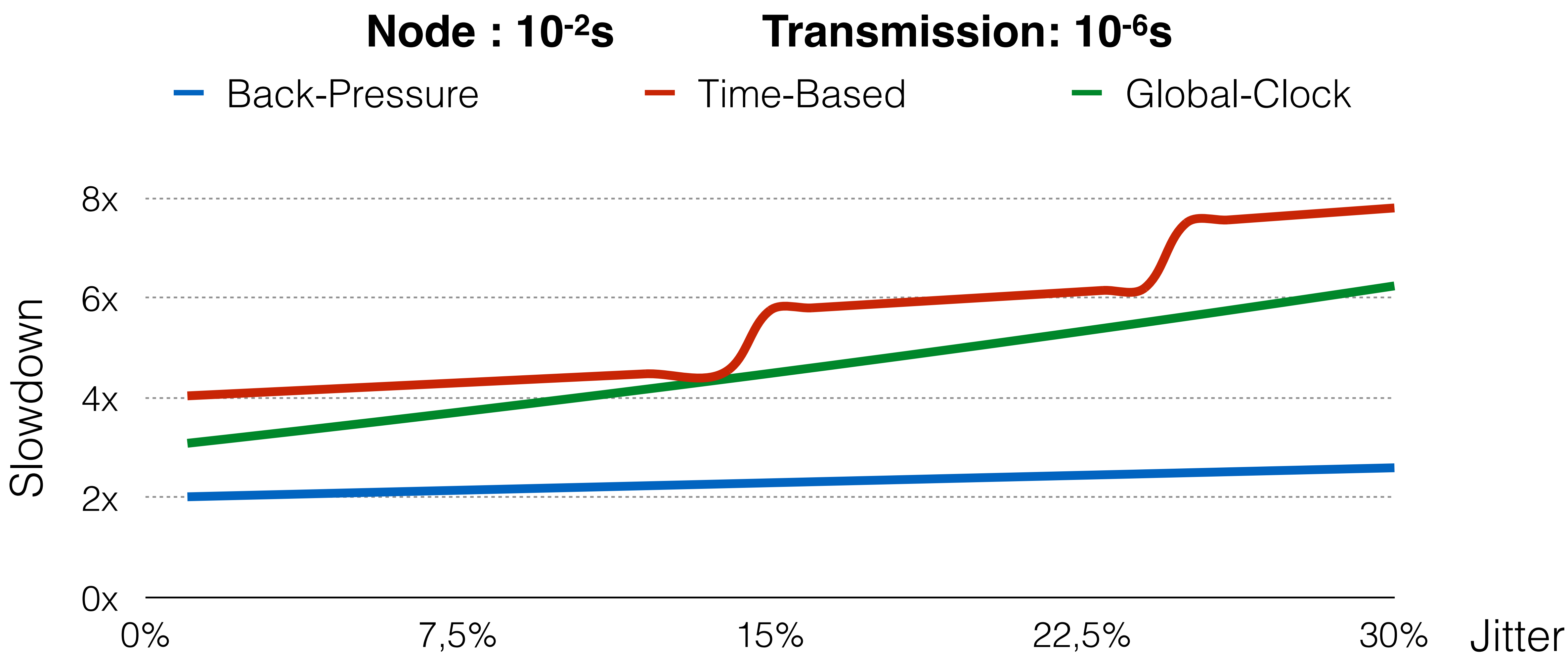
Property 2: $E^b_k \prec S^a_k$



What about Clock Synchronization?

Worst-Case Evaluation

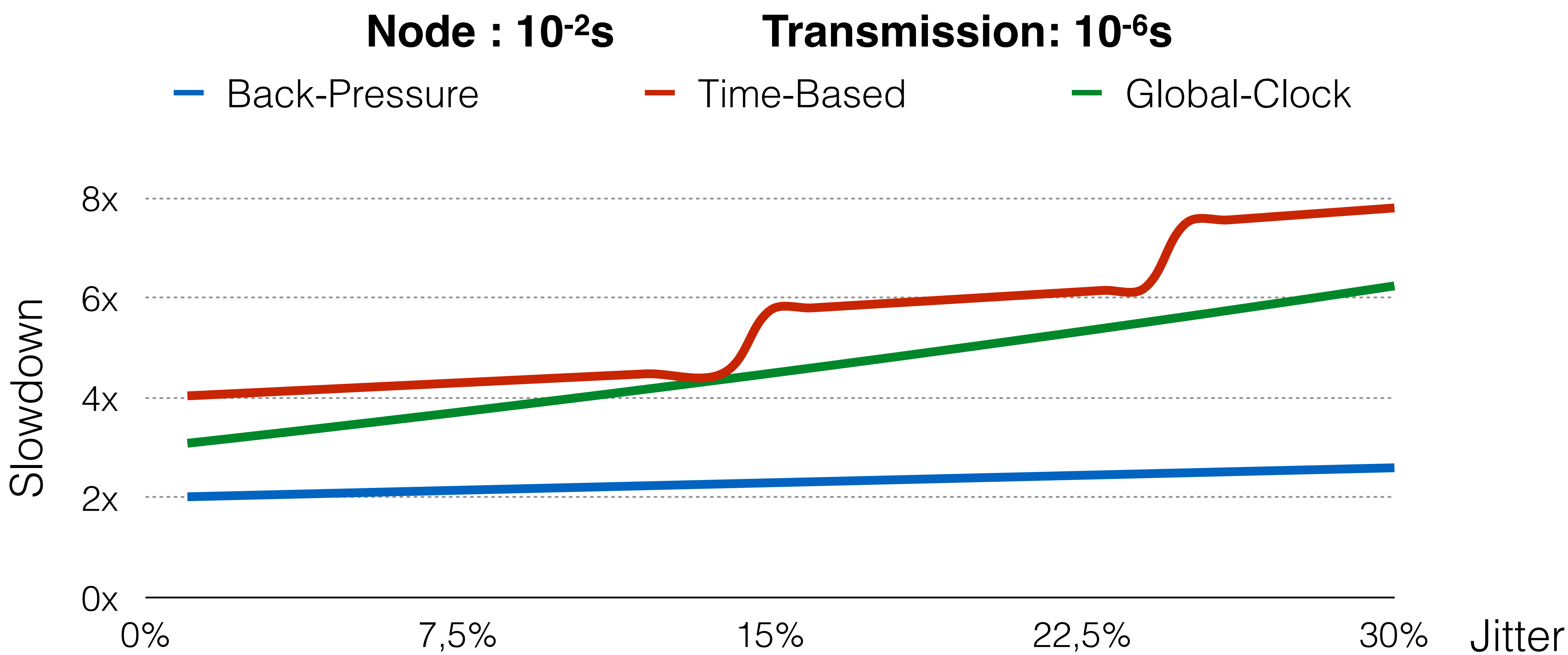
Analytical comparison with synchronous execution*



*The smaller, the better

Worst-Case Evaluation

Analytical comparison with synchronous execution*



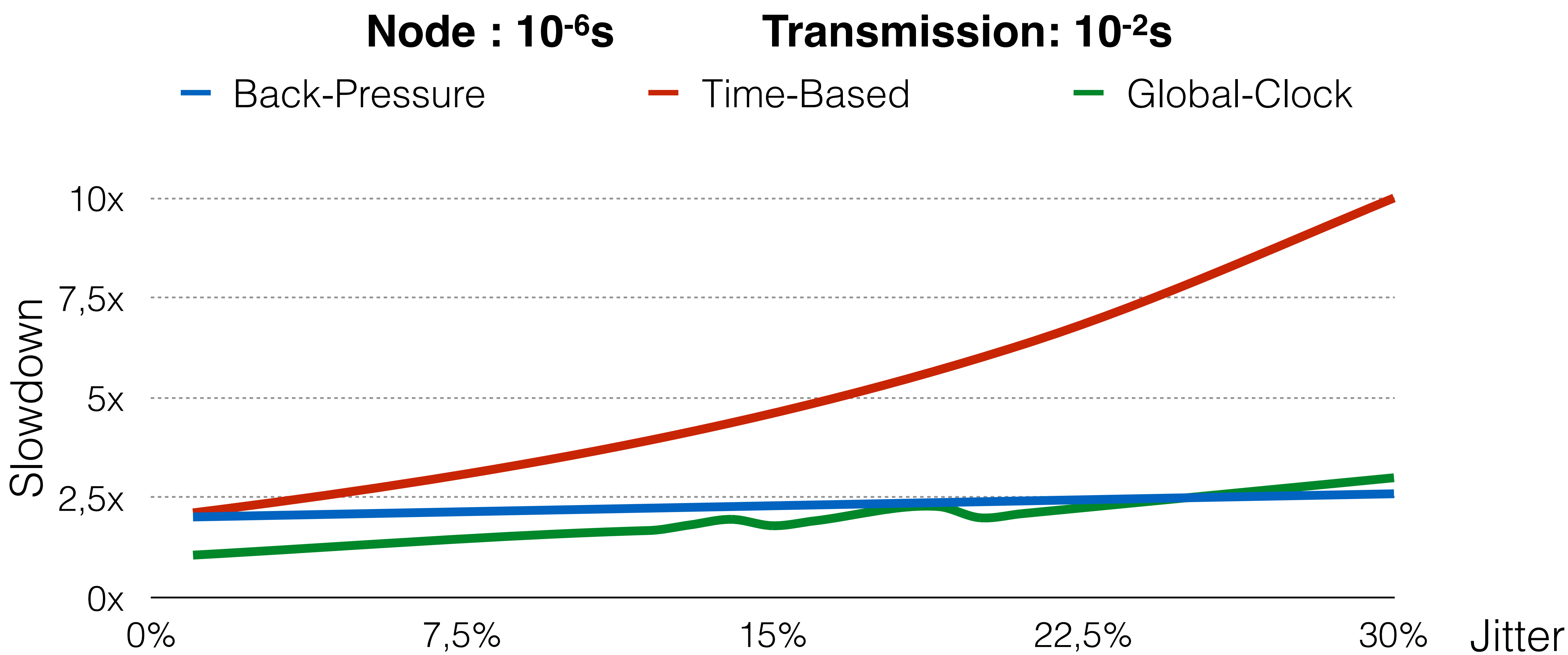
However:

- Global-Clock is as efficient as possible
- LTTA are **simpler** protocols (two control states)
- Time-Based is the **least intrusive**

*The smaller, the better

Worst-Case Evaluation

Analytical comparison with synchronous execution*



However:

- Global-Clock is as efficient as possible
- LTTA are **simpler** protocols (two control states)
- Time-Based is the **least intrusive**

*The smaller, the better

Contributions

- Clarification of the assumptions on the synchronous application

- A unifying synchronous framework for LTTAs
  that gives executable code for simulation

- Simplification of the Time-Based protocol
  - Relaxing broadcast communication
  - No more global synchronization

- Theoretical comparison with clock synchronization deployed on the same architecture

Contributions

- Clarification of the assumptions on the synchronous application

- A unifying synchronous framework for LTTAs
  that gives executable code for simulation

- Simplification of the Time-Based protocol
  ~~- Relaxing broadcast communication~~
  ~~- No more global synchronization~~

  Erratum

- Theoretical comparison with clock synchronization deployed on the same architecture