# A Synchronous View of Loosely Time-Triggered Architectures

Guillaume Baudart
Albert Benveniste
Timothy Bourke

**Séminaire INFRES LINCS'15** **Paris, 25-03-2015**

# Background

- **Quasi-synchrony:** Paul Caspi's work on programming practices of Airbus engineers
  "*no more than two ticks of one clock between two ticks of another one*"
  [Caspi 2000, *Cooking book*]

- **LTTA:** Middleware to safely deploy synchronous applications over quasi-periodic architectures
  [Tripakis et al. 2008]
  [Caspi, Benveniste 2008]

- **Asynchrony:** Synchronous models of asynchronous systems
  [Halbwachs, Baghdadi 2002]
  [Halbwachs, Mandel 2006]

# Outline

1. What is an LTTA?
    1. Quasi-Periodic Architecture
    2. Synchronous Applications
2. General Framework
3. The two protocols
    1. Back-Pressure LTTA
    2. Time-Based LTTA
4. What About Clock Synchronisation?

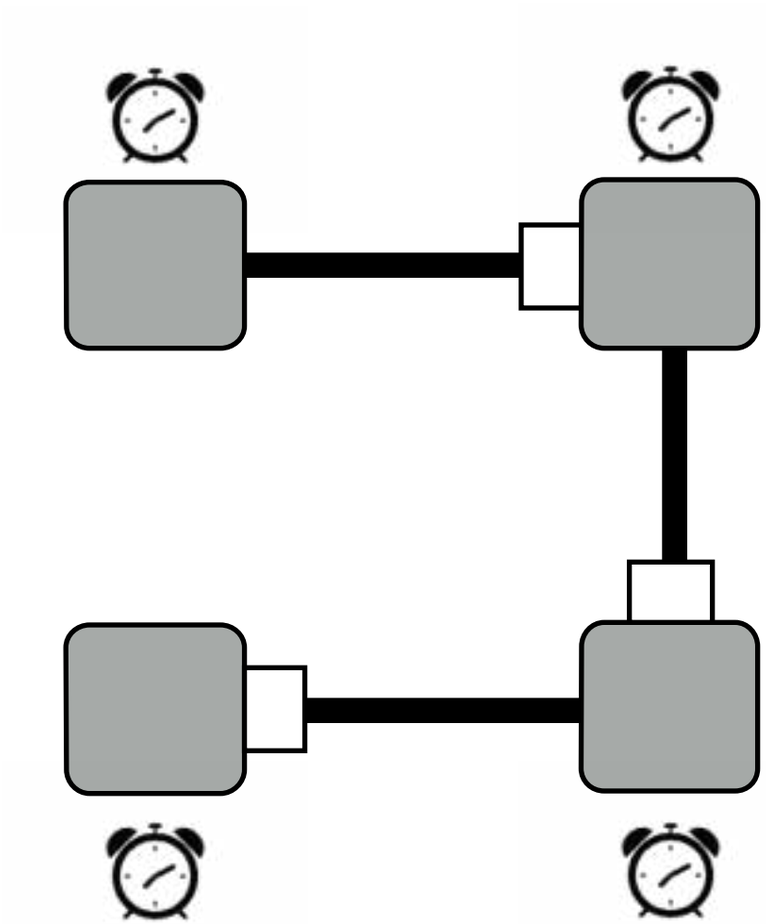
# Outline

# Quasi-Periodic Architecture

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

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

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



- Buffered communication without message inversion or loss

- Bounded communication delay

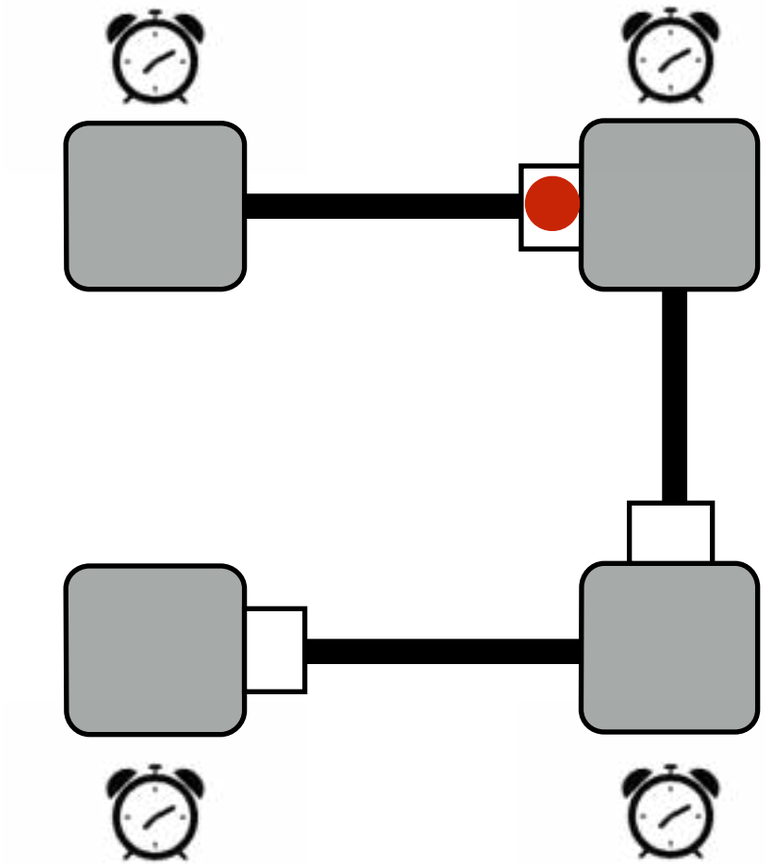
$$\tau_{\min} \leq \tau \leq \tau_{\max}$$

# Quasi-Periodic Architecture

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

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

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



- Buffered communication without message inversion or loss

- Bounded communication delay

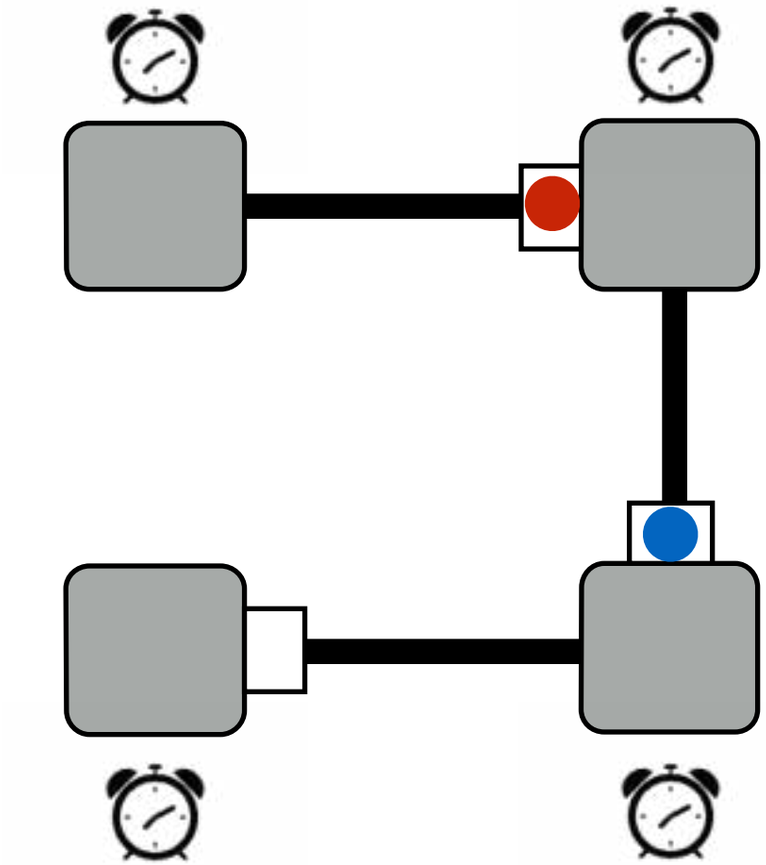
$$\tau_{\min} \leq \tau \leq \tau_{\max}$$

# Quasi-Periodic Architecture

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

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

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



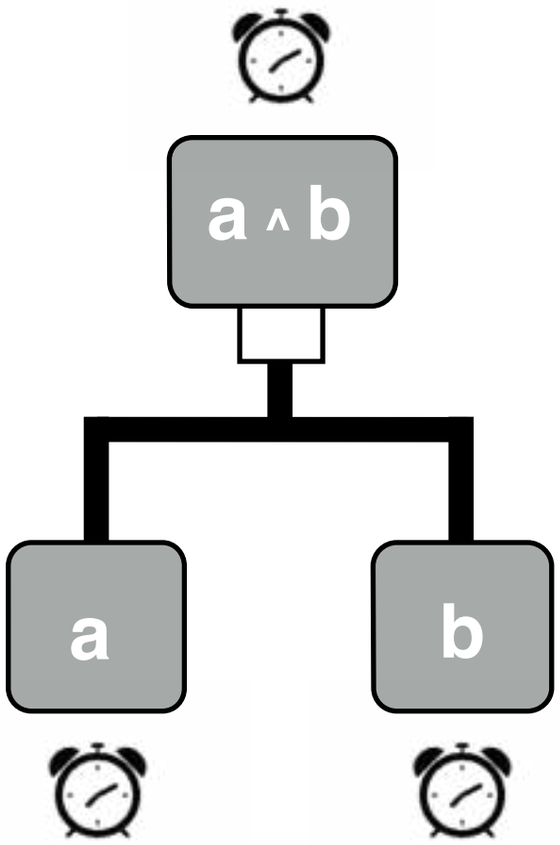
- Buffered communication without message inversion or loss

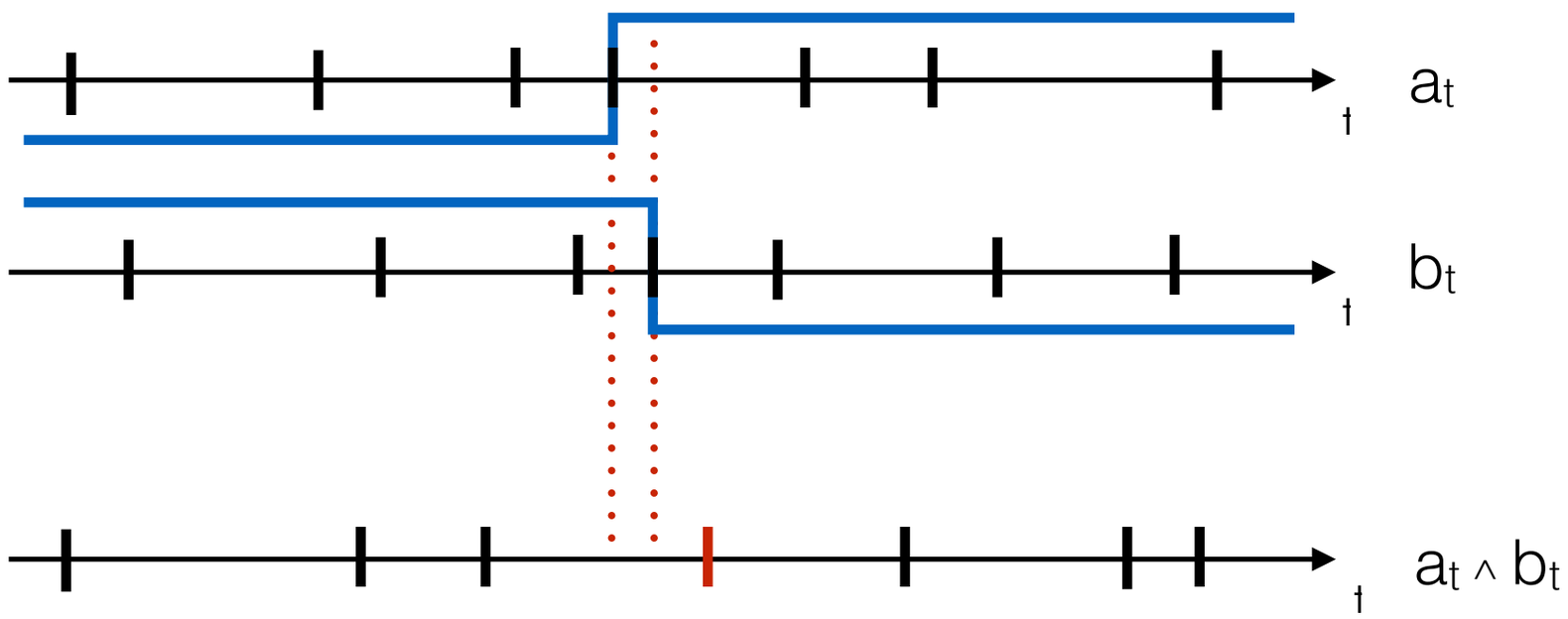
- Bounded communication delay

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

# Problems

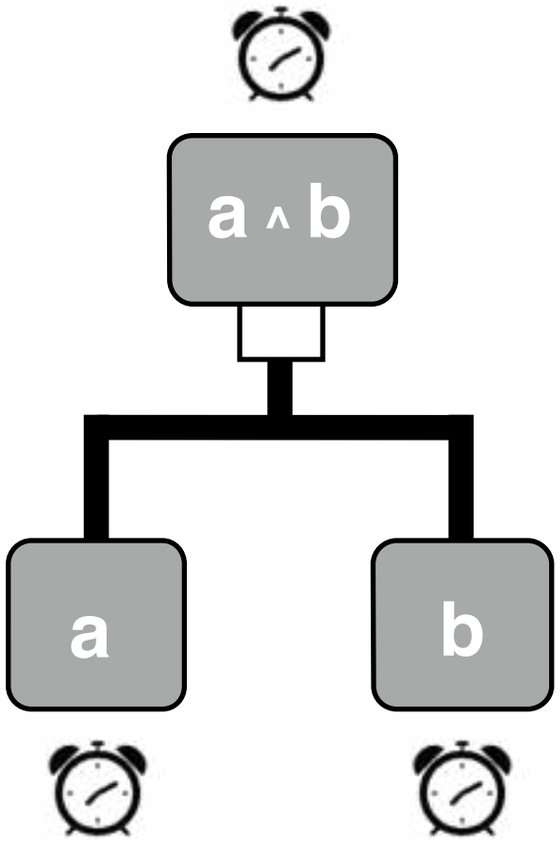
- **Overwriting:** Loss of values
- **Oversampling:** Duplication of values
- **Combination of signals**

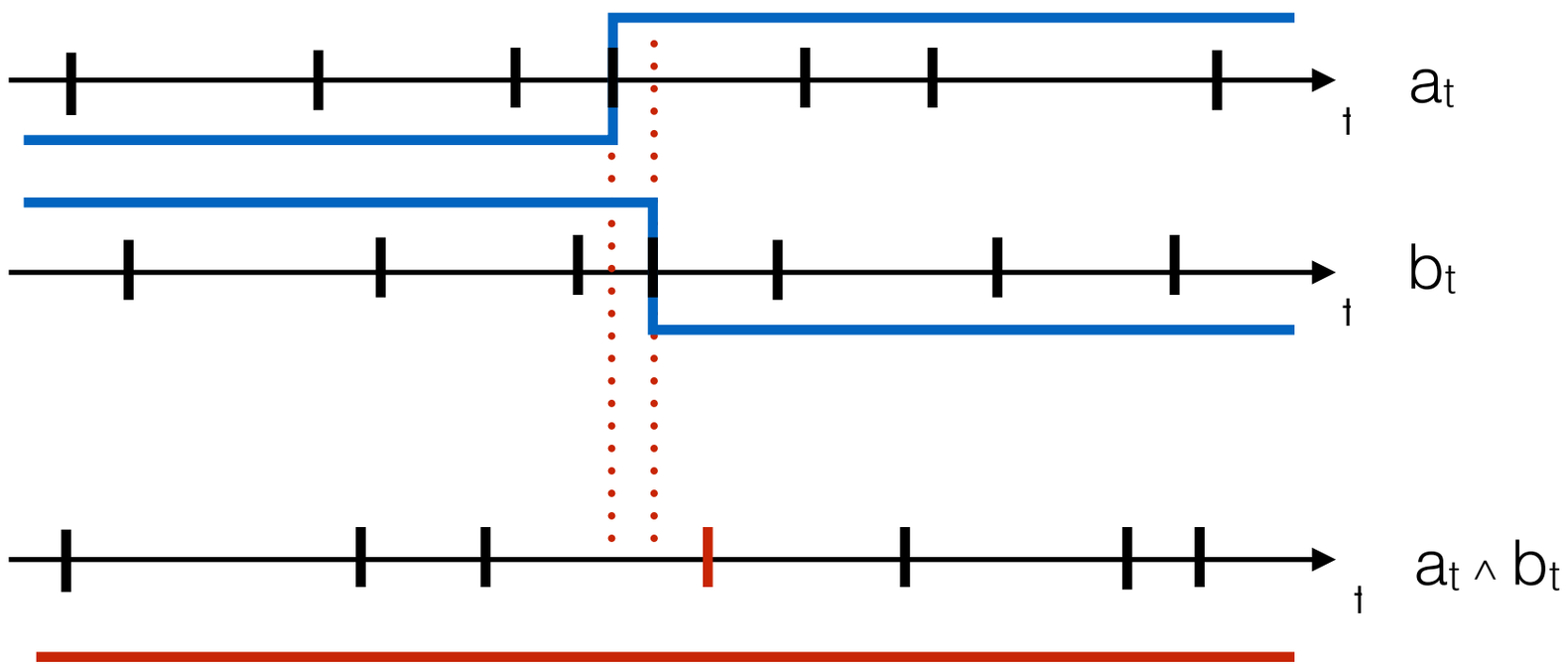


# Problems

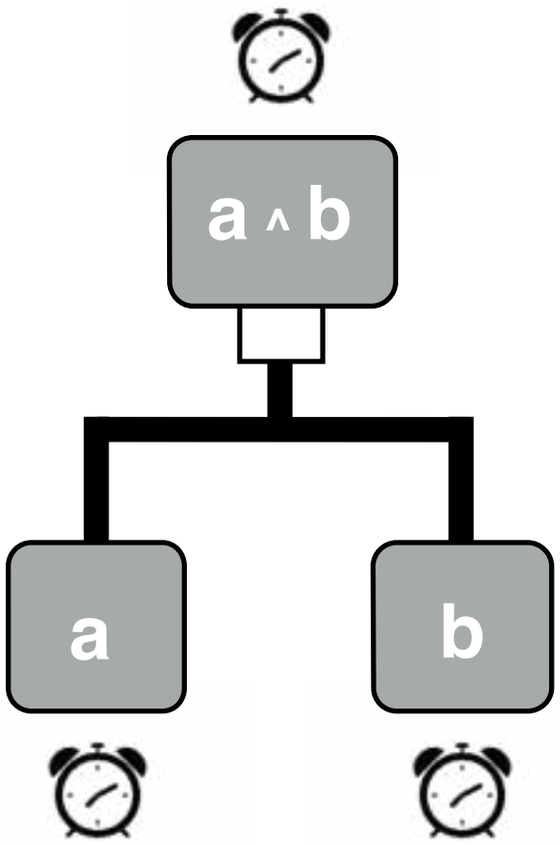
- **Overwriting:** Loss of values
- **Oversampling:** Duplication of values
- **Combination of signals**

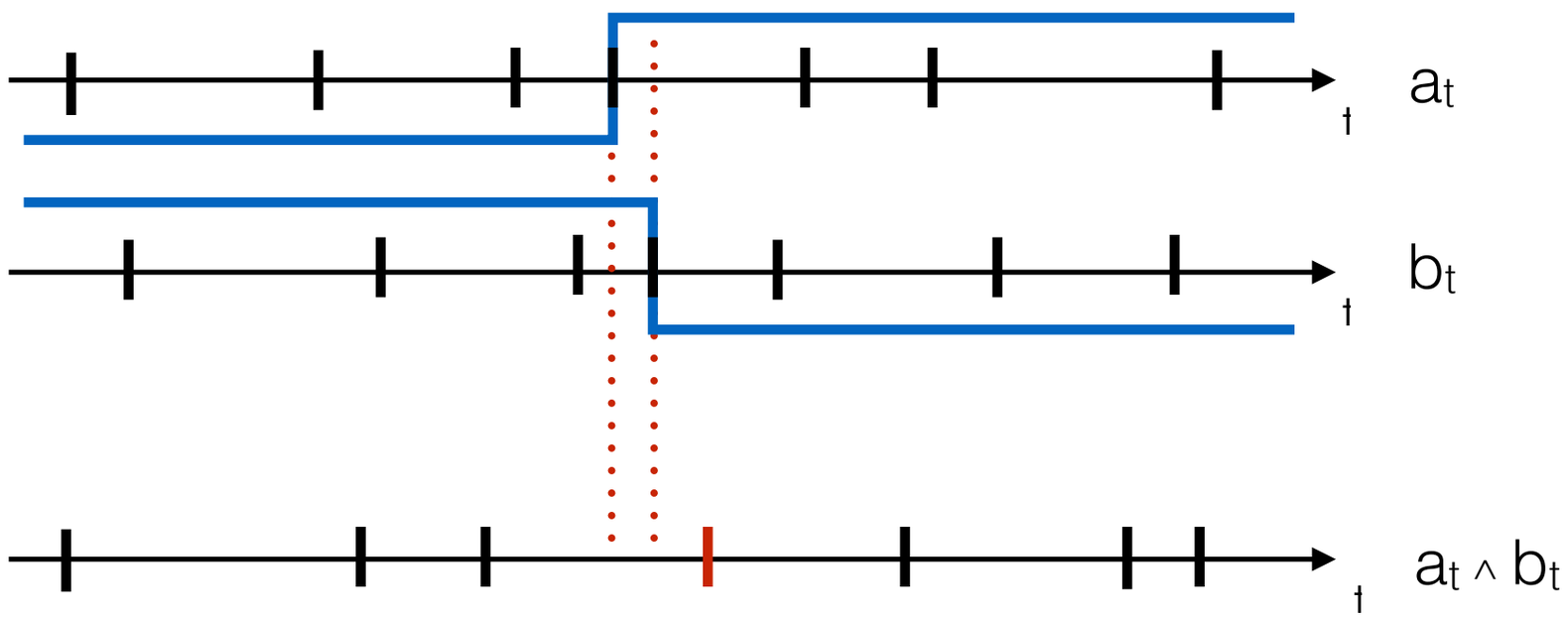


# Problems

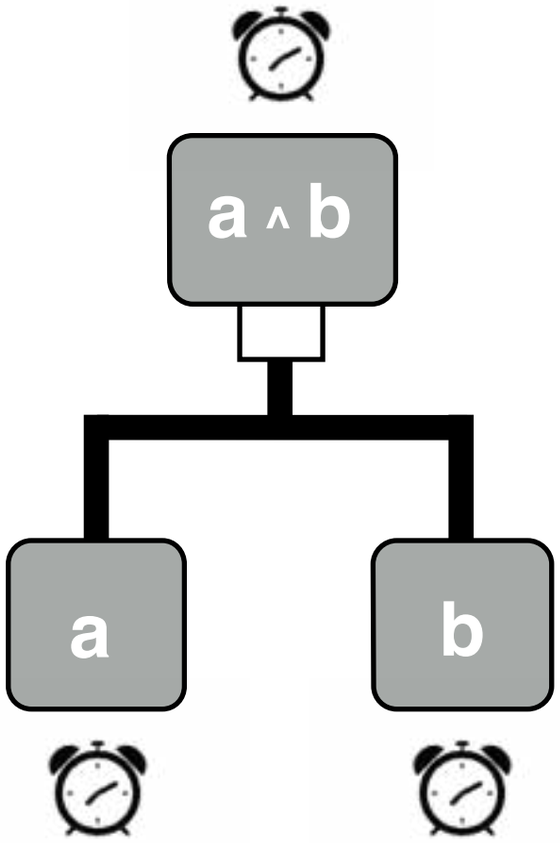
- **Overwriting:** Loss of values
- **Oversampling:** Duplication of values
- **Combination of signals**

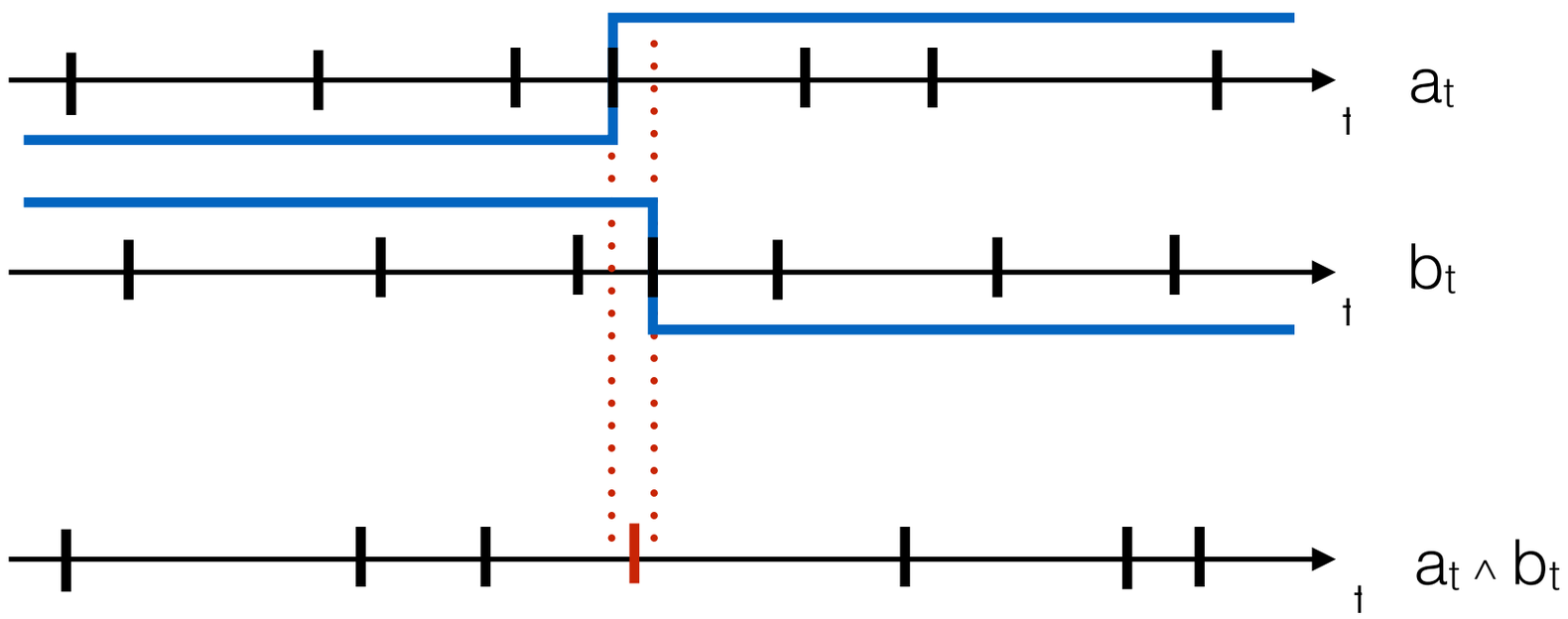


# Problems

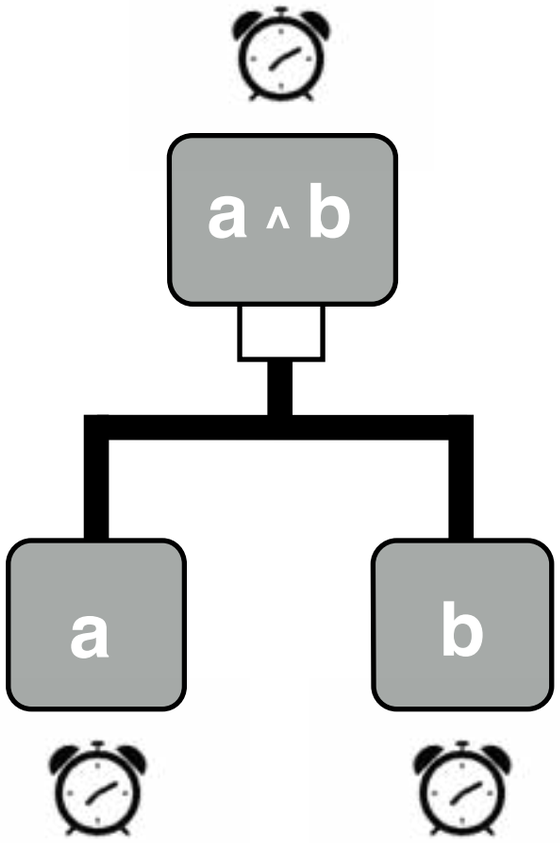
- **Overwriting:** Loss of values
- **Oversampling:** Duplication of values
- **Combination of signals**

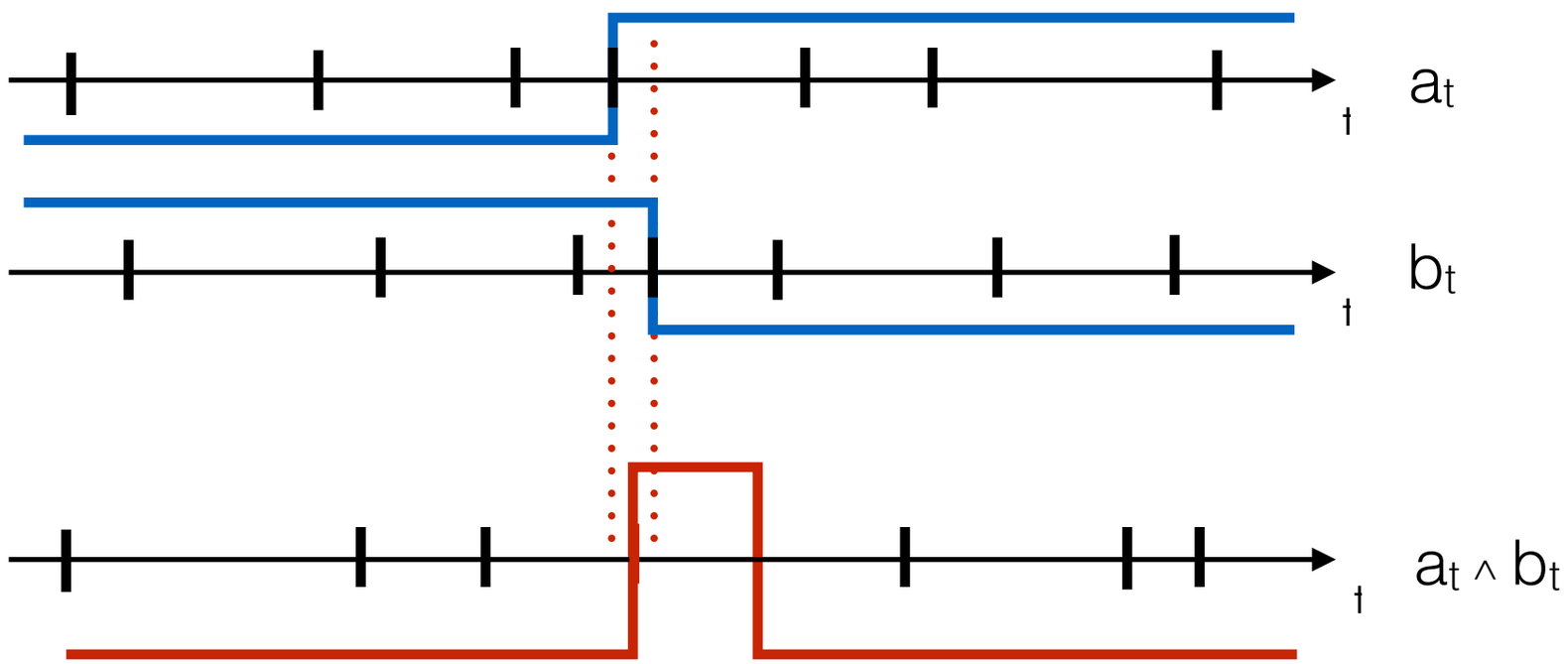


# Problems

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



# Synchronous Applications

## Network of Communicating Mealy Machines

- Initial state $S_{\text{init}}$
- Transition function $F : \mathcal{S} \times \mathcal{V}^{n_i} \to \mathcal{S}' \times \mathcal{V}^{n_o}$

**Semantics**

Synchronous $[\![m]\!]^S : (\mathcal{V}^{n_i})^\infty \to (\mathcal{V}^{n_o})^\infty$

Kahn $[\![m]\!]^K : (\mathcal{V}^\infty)^{n_i} \to (\mathcal{V}^\infty)^{n_o}$

# Synchronous Applications

Network of Communicating Mealy Machines

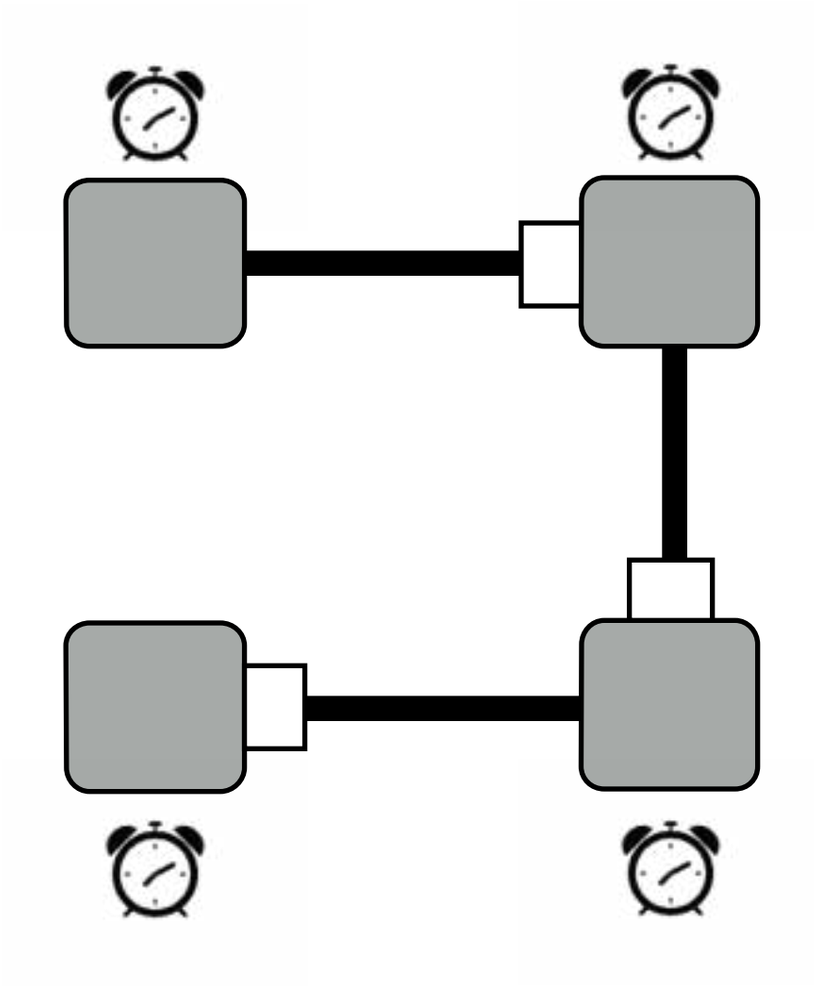
- **Composition:** output to input
- **Causality:** no instantaneous dependency cycles

*Basically, classic synchronous programs ‘mono-clock’.*

- **Assumption:** no instantaneous dependencies between nodes (avoid ‘microschedule’)

# What are LTTAs?

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



# Different Approaches

- **Discrete abstractions,** e.g., quasi-synchrony [Caspi 2000]
  - Allows verifications
  - State explosion, incomplete model

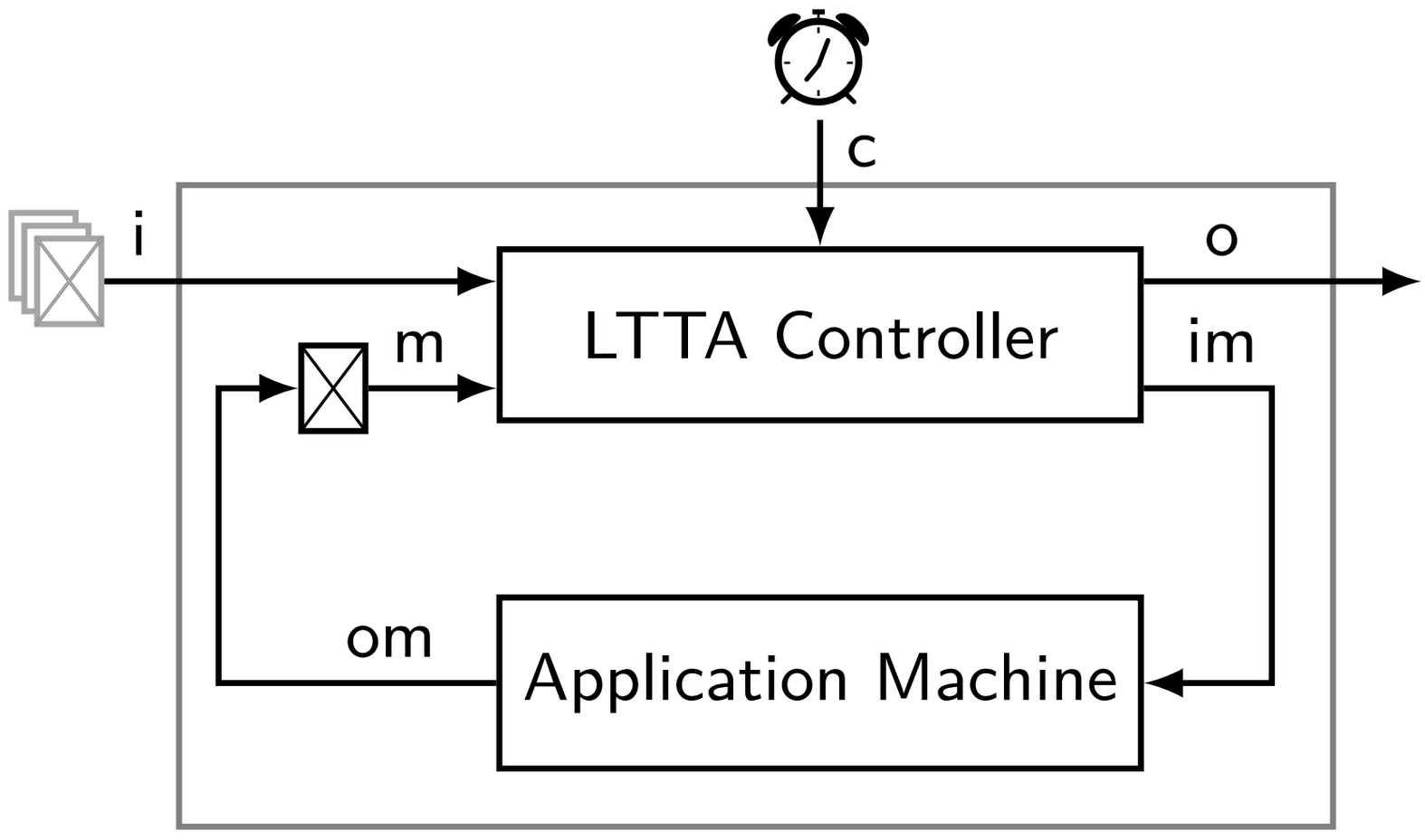
- **Petri nets,** [Benveniste et al. 2010]
  - Unify LTTA protocols
  - Complex model, no implementation

- **Zélus:**
  - A single language for discrete- and real-time,
  - Implementation, simulation using numerical solvers
  - But no verification at this point

# Outline

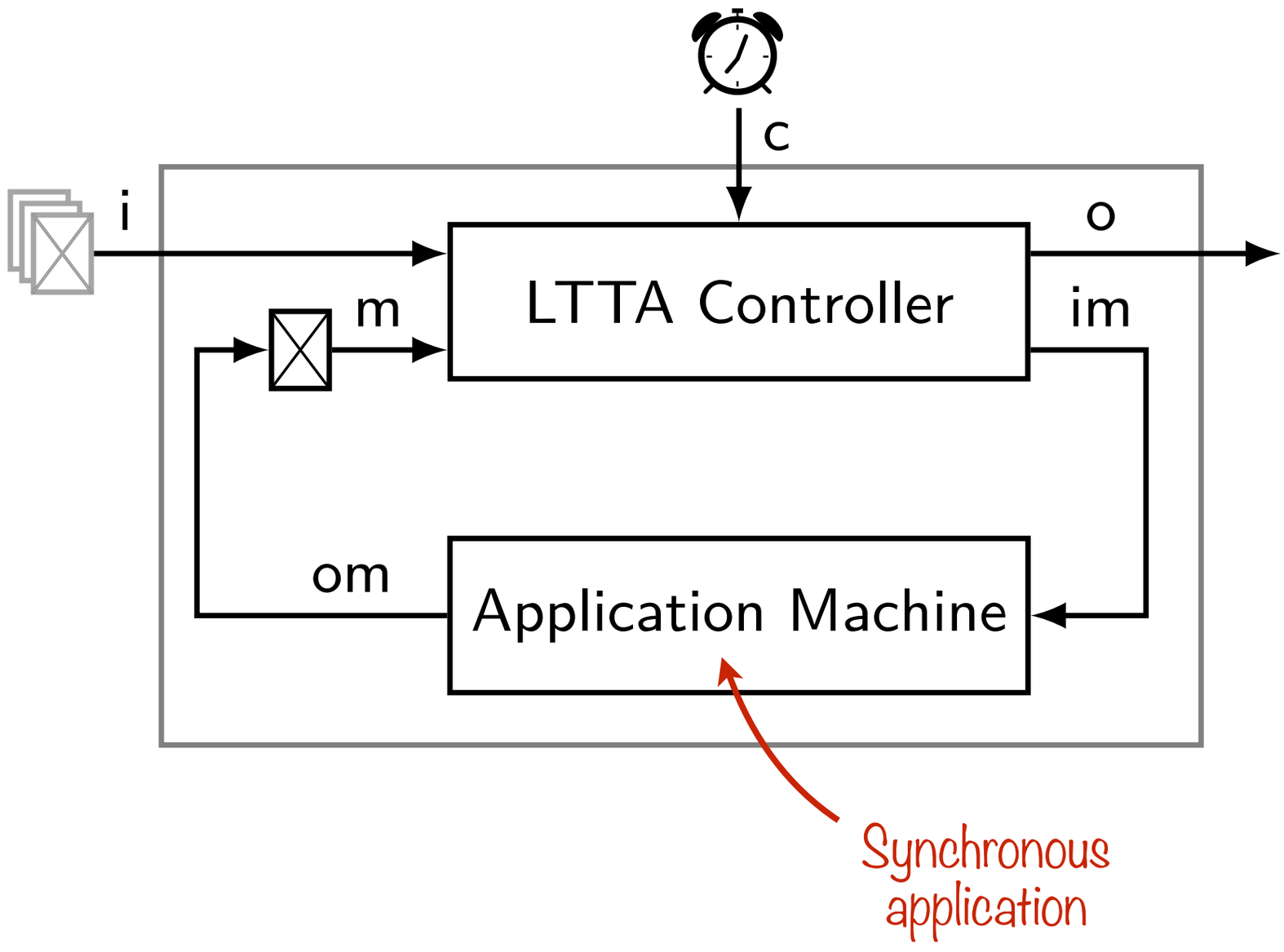
# General Framework

## Modelling nodes



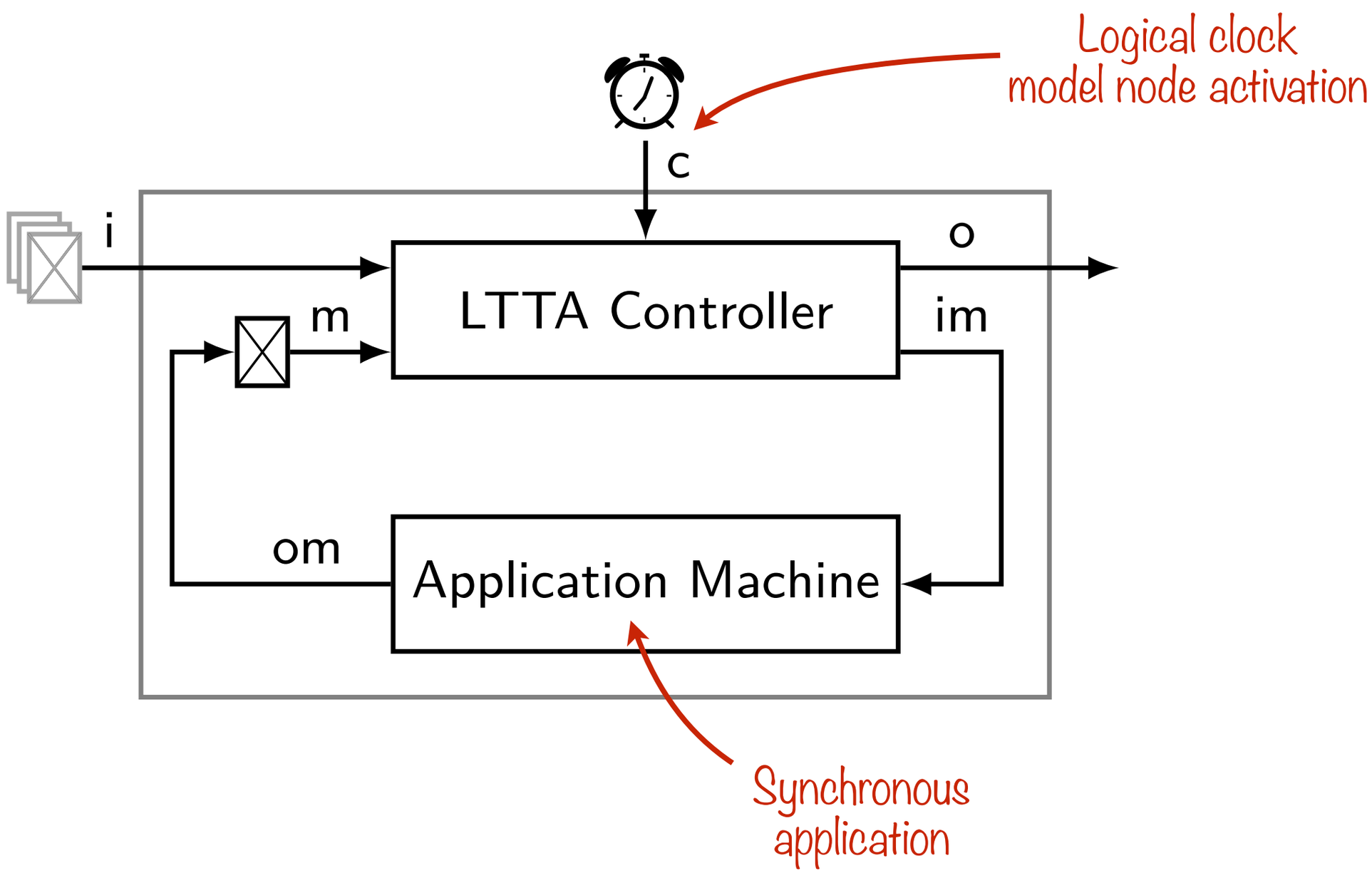
# General Framework

## Modelling nodes



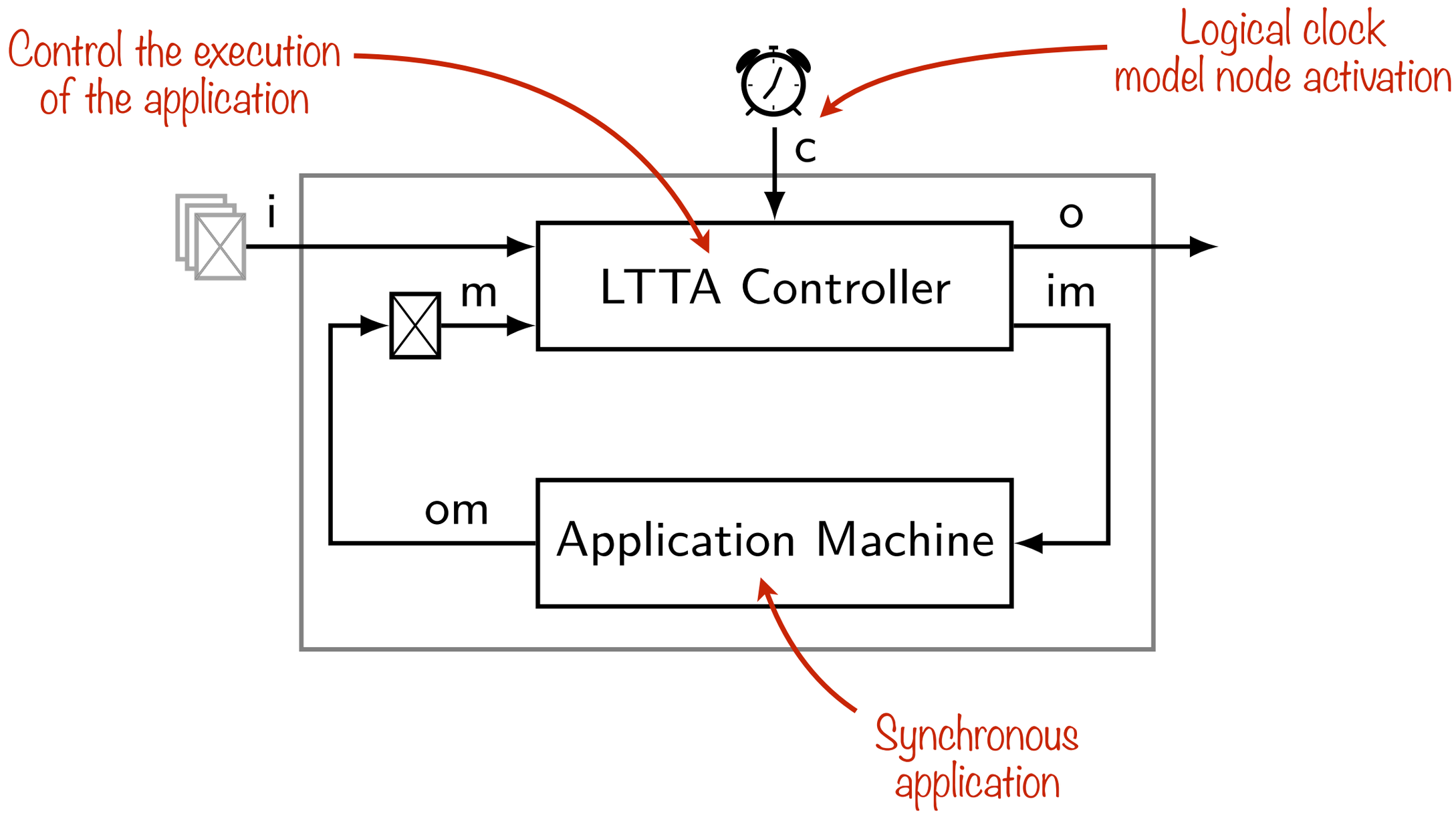
# General Framework

## Modelling nodes



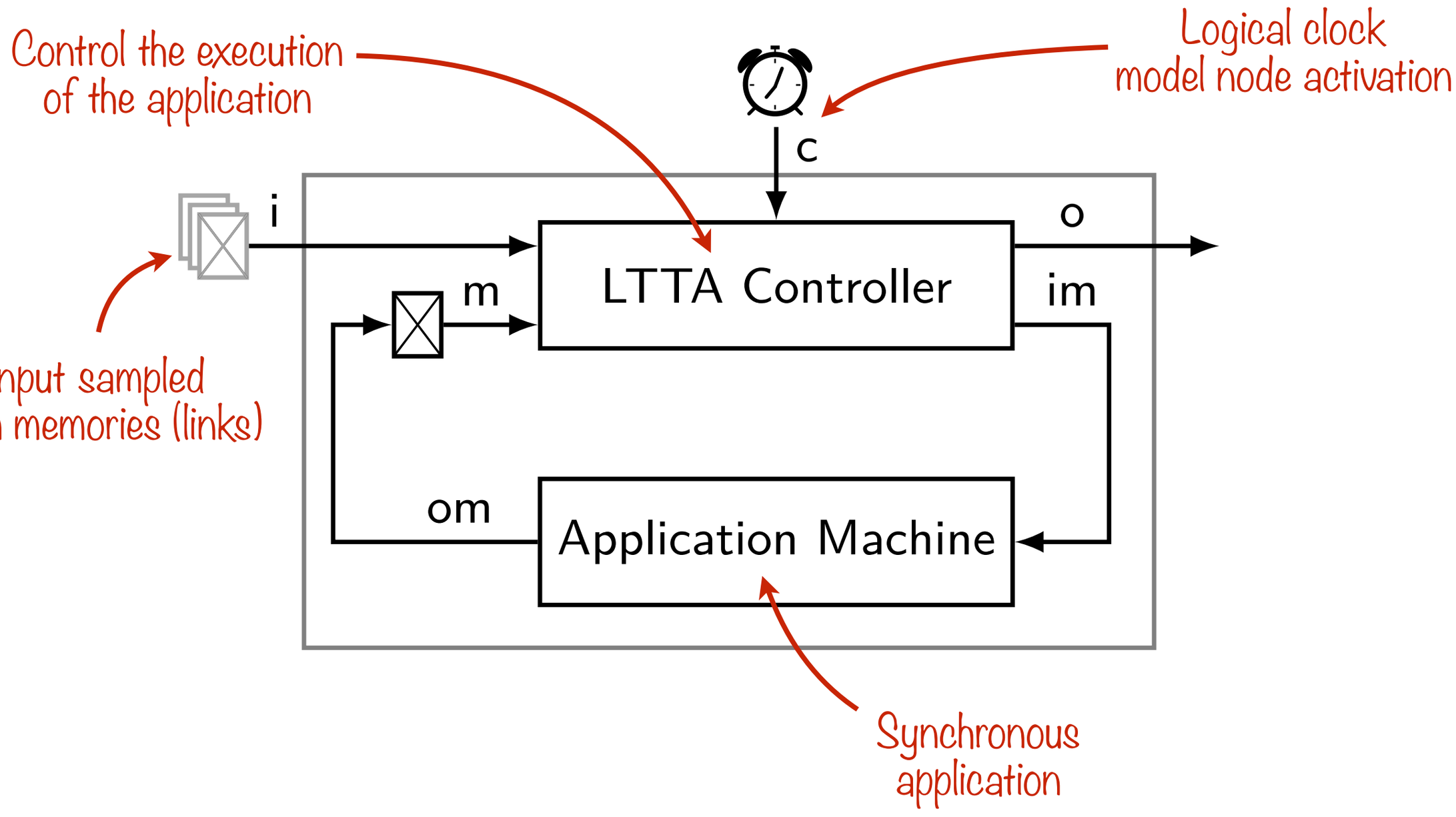
# General Framework

## Modelling nodes



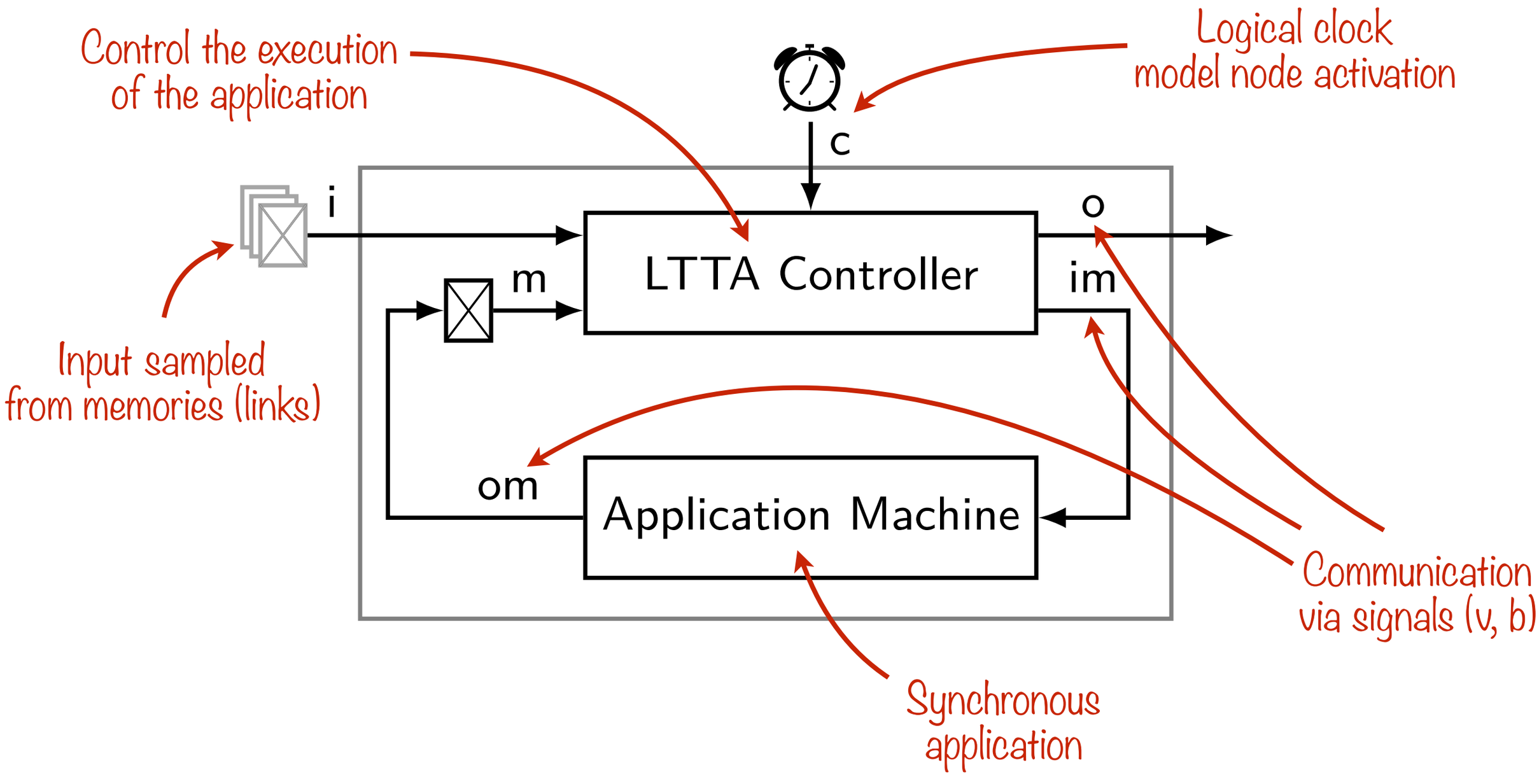
# General Framework

## Modelling nodes



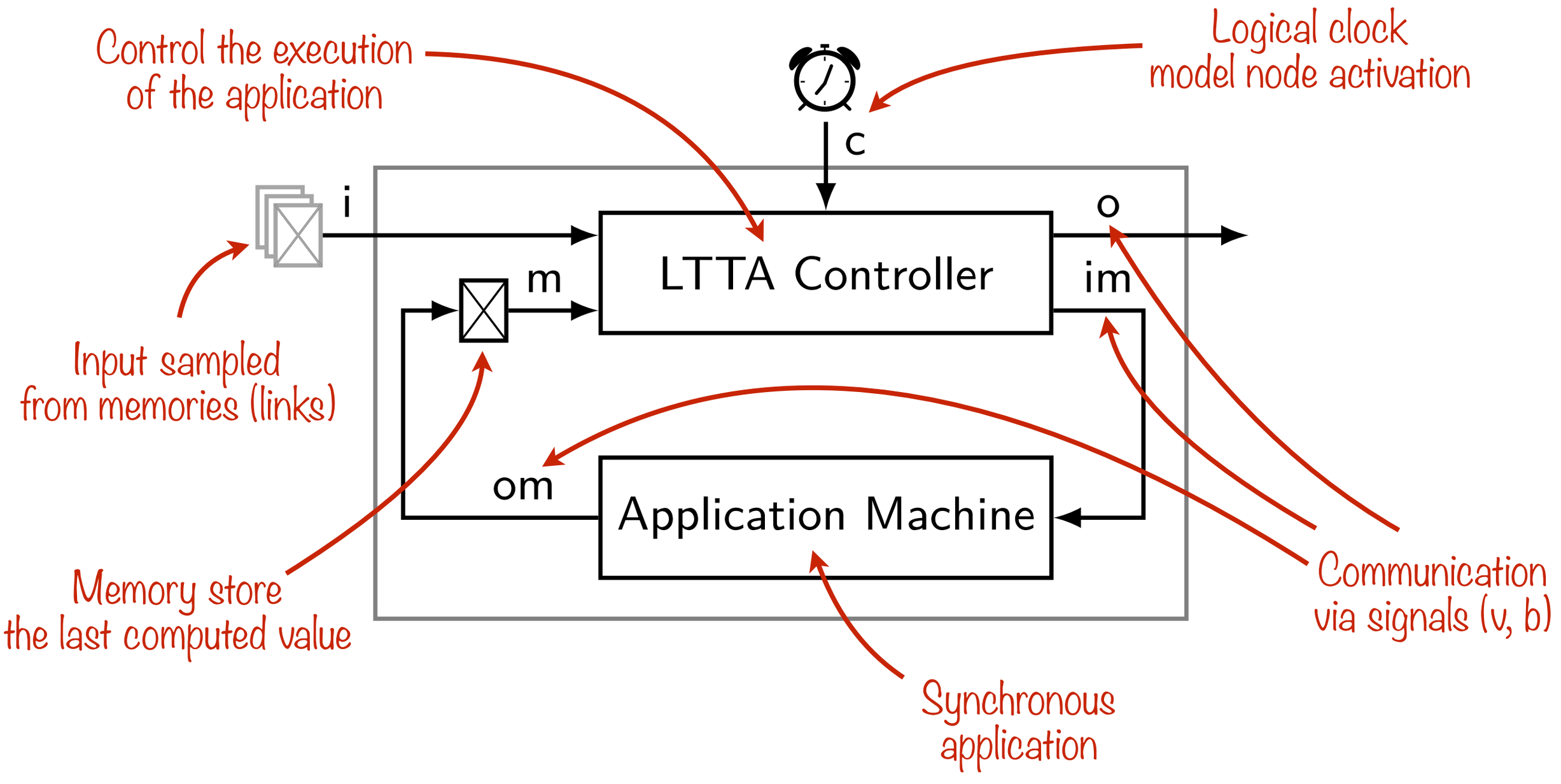
# General Framework

## Modelling nodes



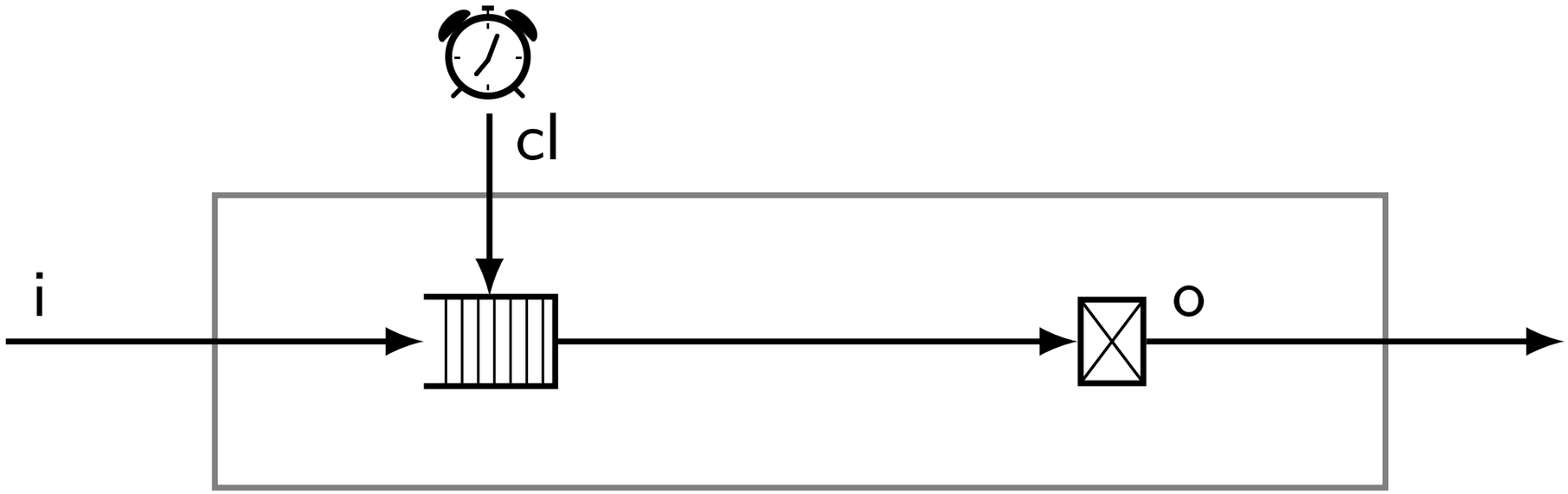
# General Framework

## Modelling nodes



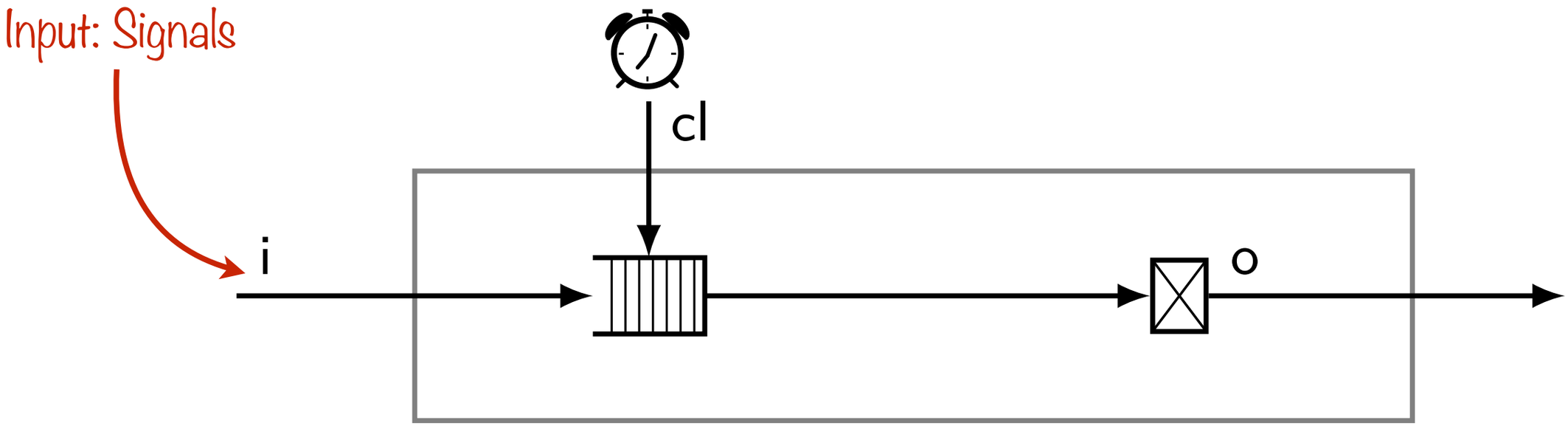
# General Framework

## Modelling Links



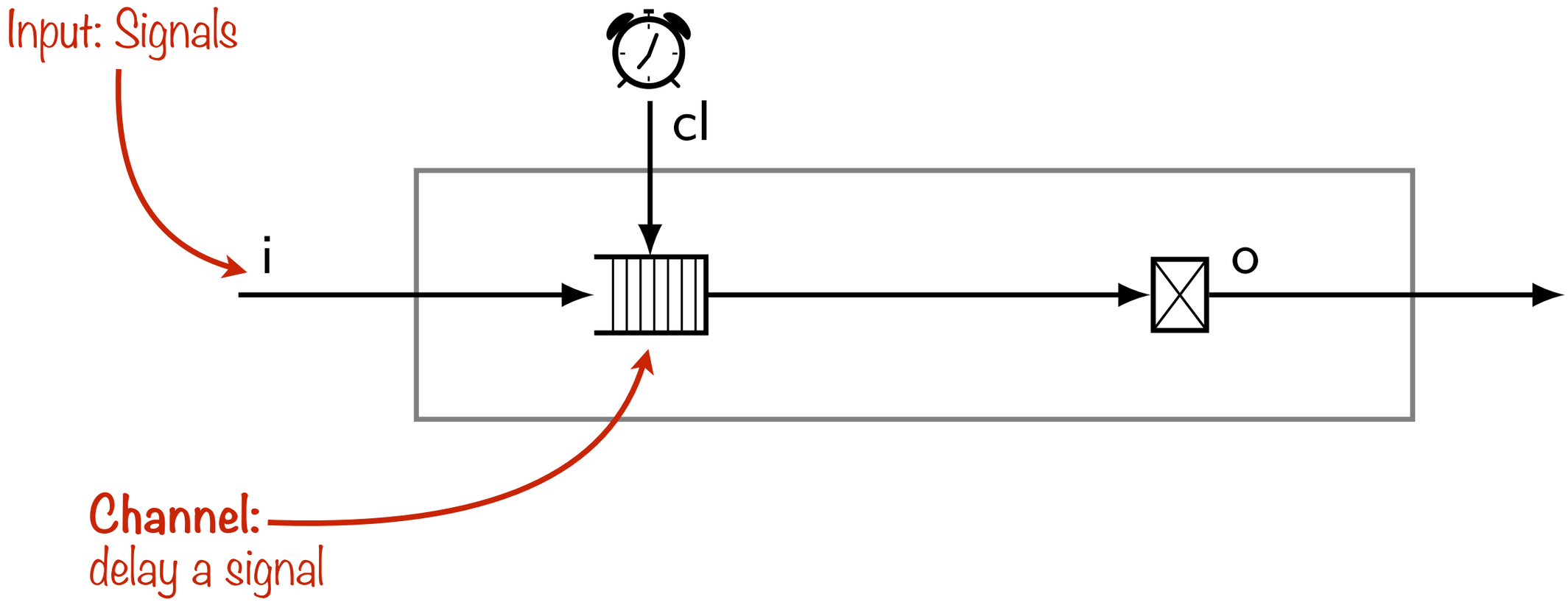
# General Framework

## Modelling Links



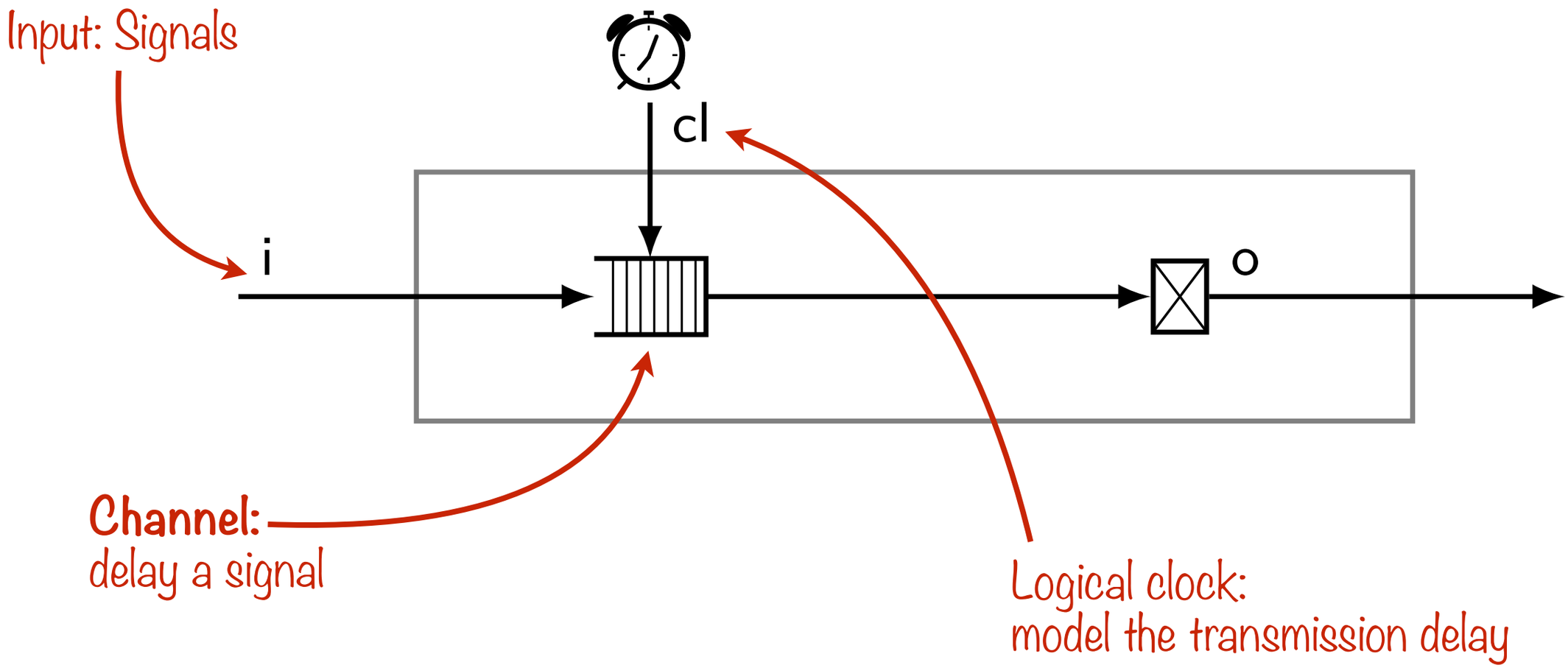
# General Framework

## Modelling Links



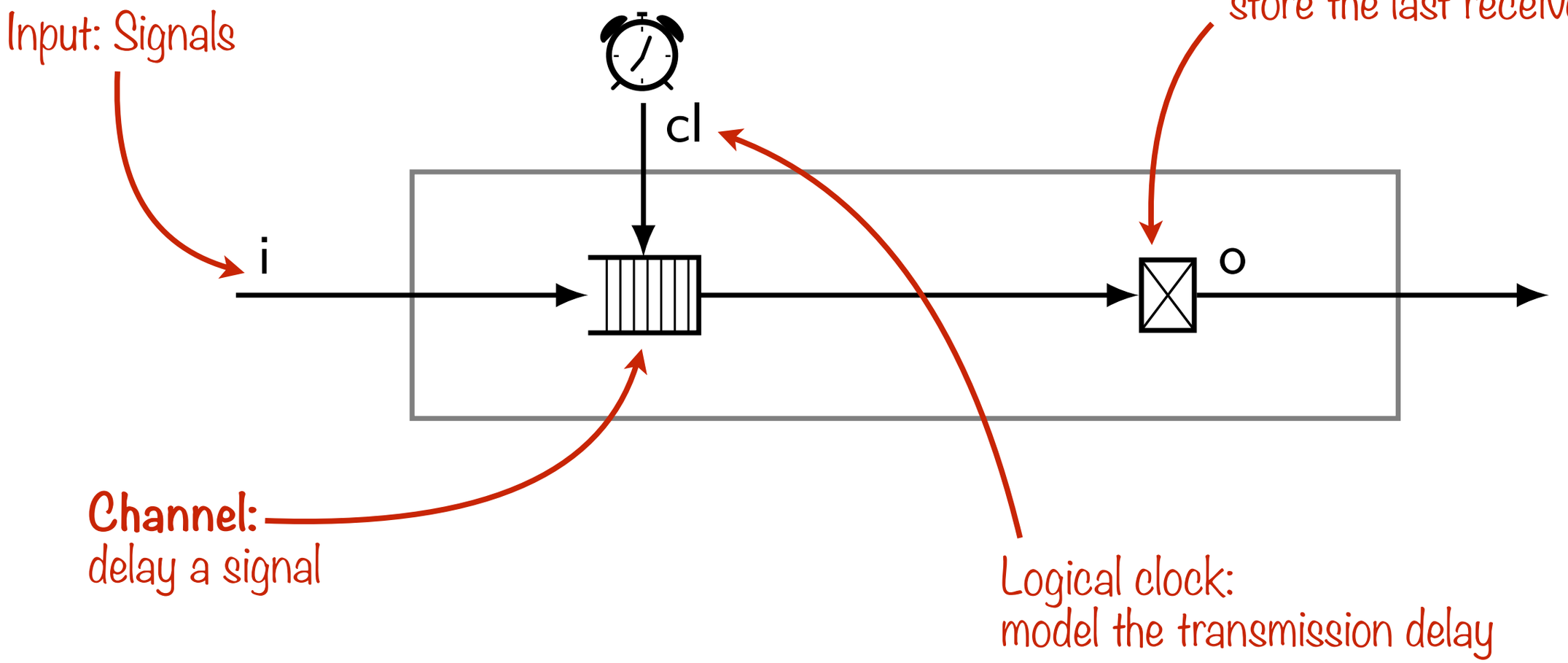
# General Framework

## Modelling Links



# General Framework

## Modelling Links

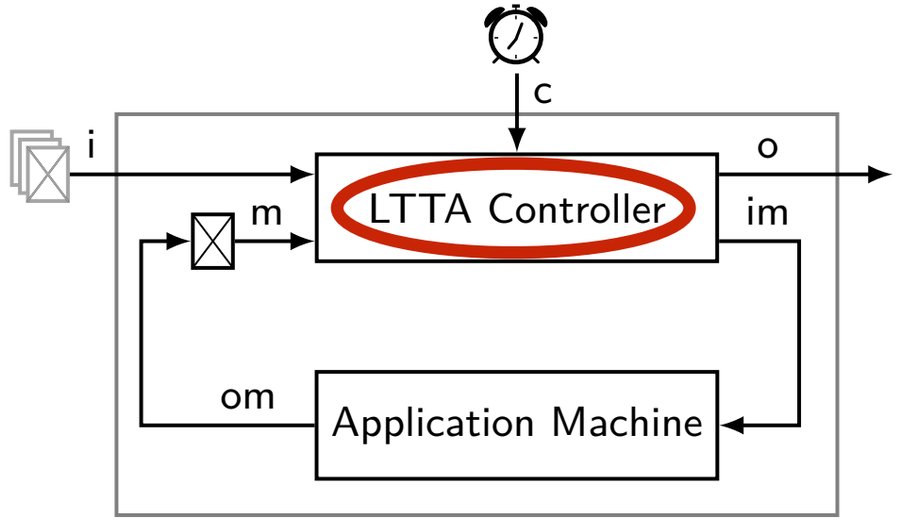


# What's next?

Design controllers that ensure a synchronous execution of embedded machines

- **Back-Pressure LTTA**
  [Tripakis et al. 2008]

- **Time-Based LTTA**
  [Caspi, Benveniste 2008]



# Outline

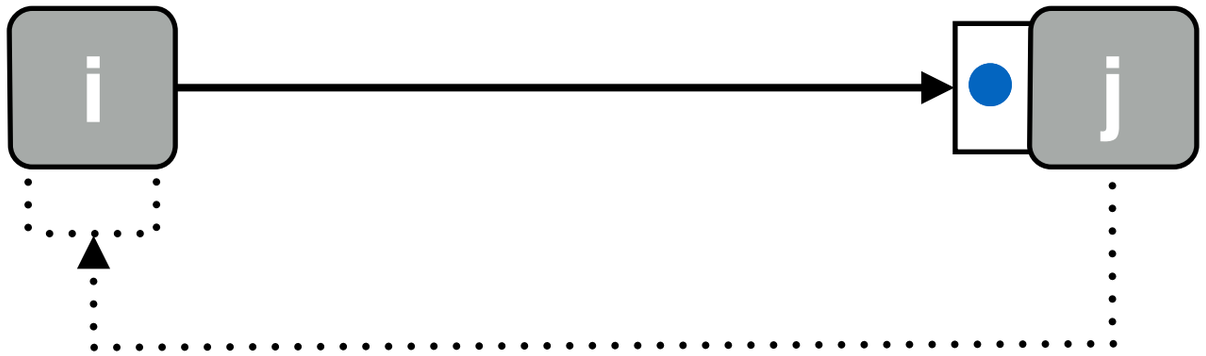
1. What is an LTTA?
    1. Quasi-Periodic Architecture
    2. Synchronous Applications
2. General Framework

3. The two protocols
    1. Back-Pressure LTTA
    2. Time-Based LTTA

4. What About Clock Synchronisation?

# Back-Pressure Kahn Network

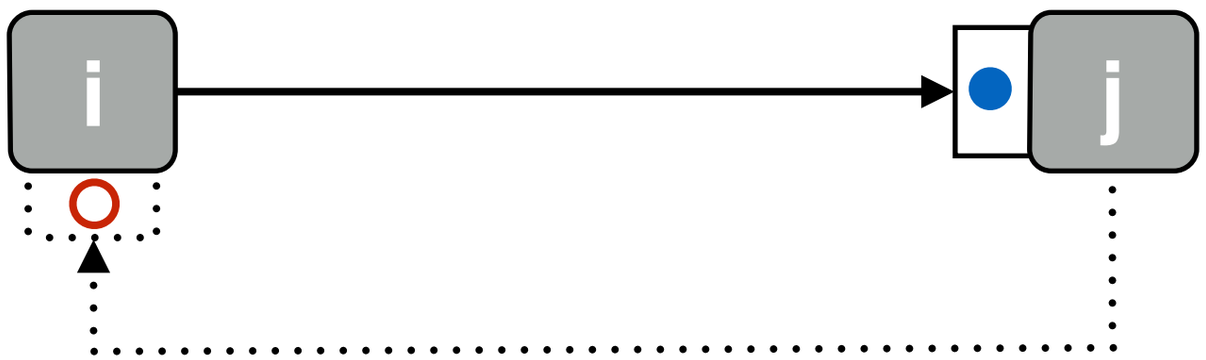
Buffer of Size 1




- Reading from a buffer is acknowledged to the writer
- Nodes alternate between **exec** and **send**

# Back-Pressure Kahn Network

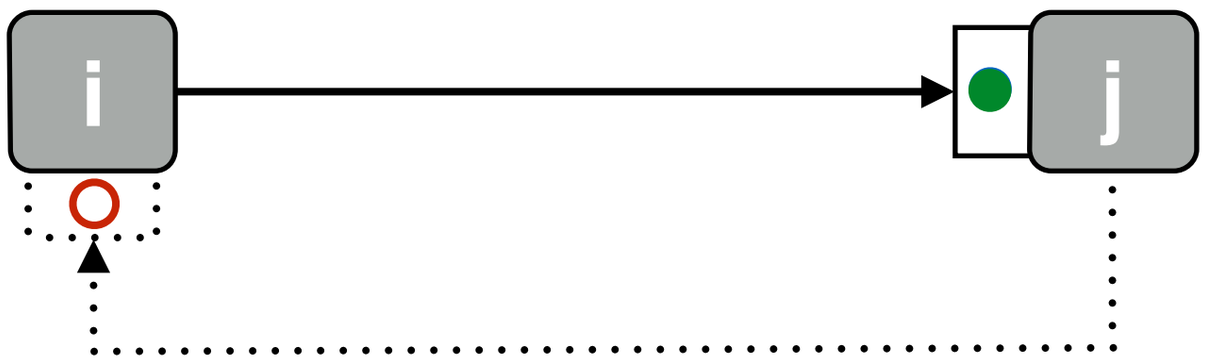
Buffer of Size 1




- Reading from a buffer is acknowledged to the writer
- Nodes alternate between **exec** and **send**

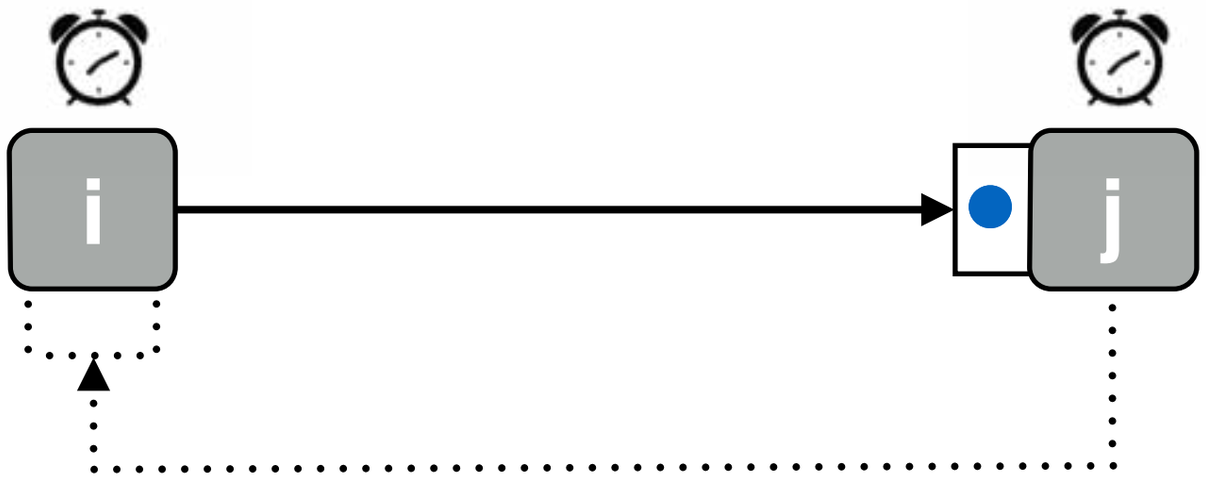
# Back-Pressure Kahn Network

Buffer of Size 1



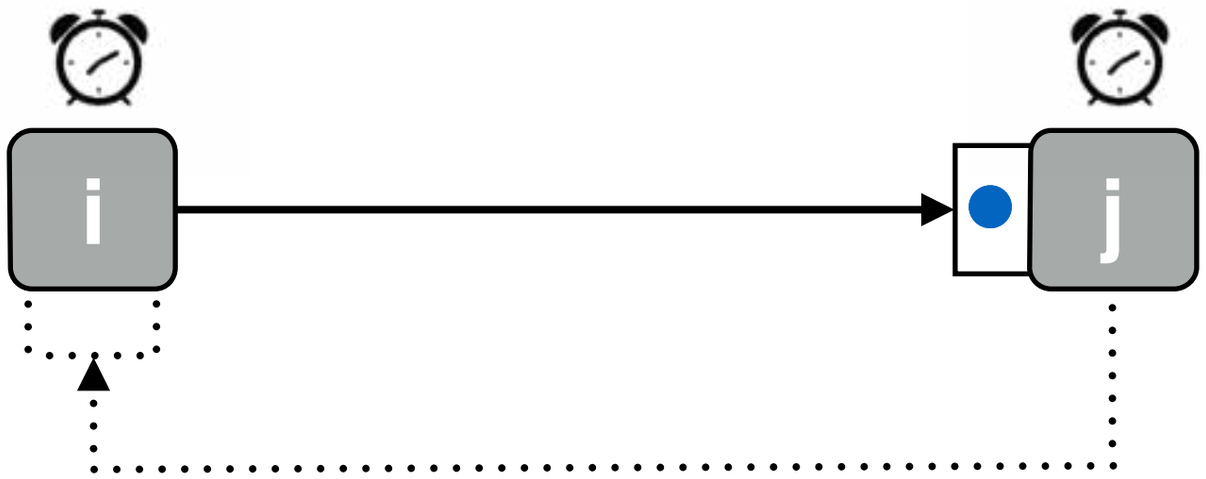

- Reading from a buffer is acknowledged to the writer
- Nodes alternate between **exec** and **send**

# Back-Pressure LTTA



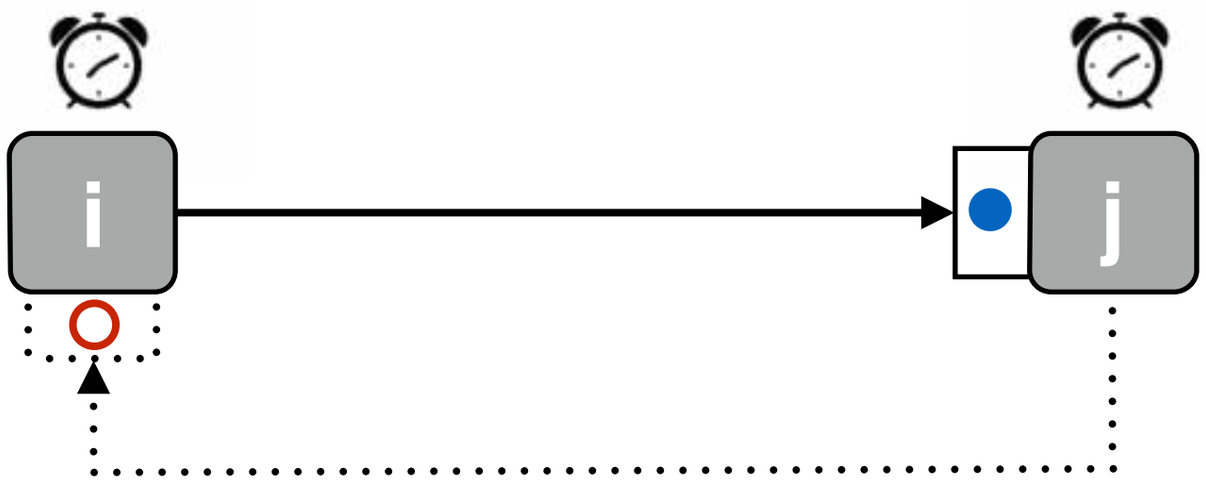

- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism

# Back-Pressure LTTA



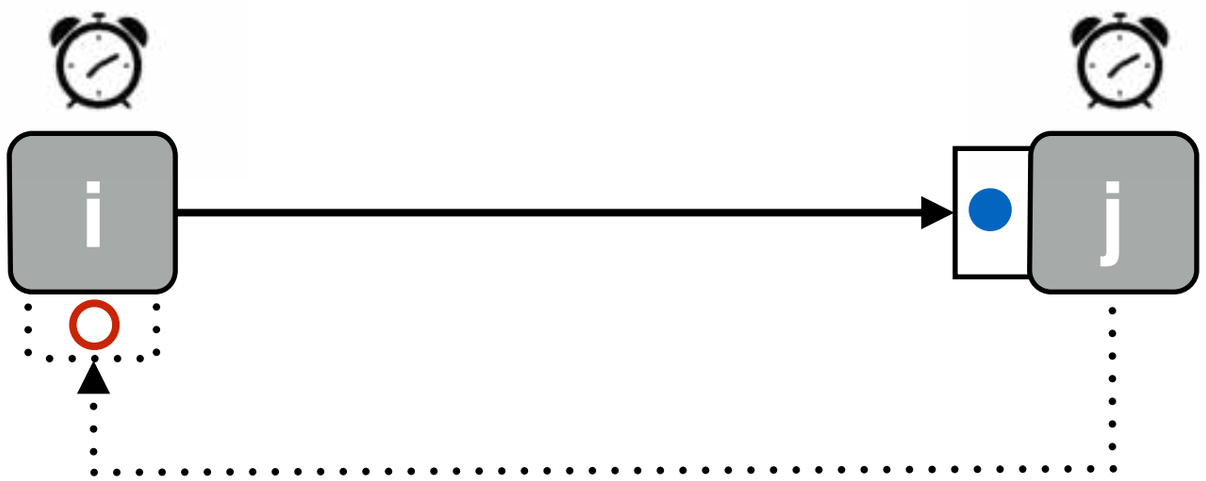

- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism

# Back-Pressure LTTA



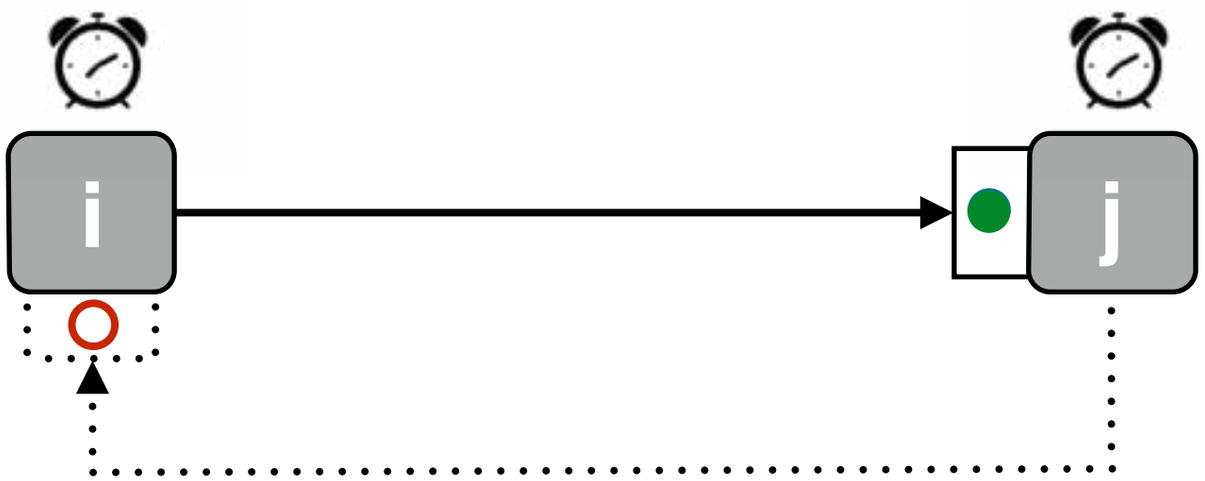

- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism

# Back-Pressure LTTA



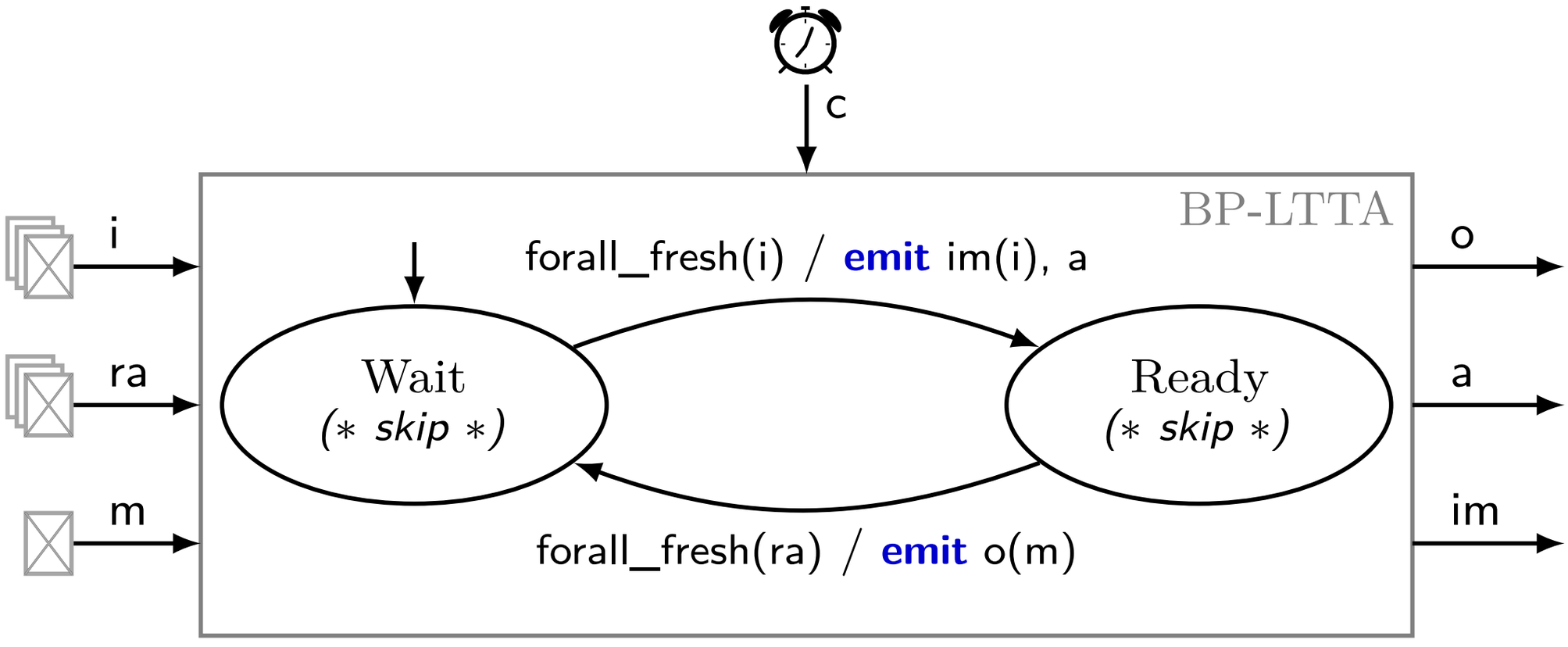

- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism

# Back-Pressure LTTA




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

# Back-Pressure LTTA



# Back-Pressure LTTA

- **Theorem 1:**
  Composition of the controller and the embedded machine is always well-defined (no cycle)

- **Theorem 2:**
  Back-pressure LTTA preserves the Kahn semantics of the embedded application
  (forget the skips)

- **Theorem 3:**
  The worst case throughput is: $1/\lambda_{\mathrm{BP}} = 2(T_{\max} + \tau_{\max})$

# Time-Based LTTA

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

- **Idea:** Replace back-pressure by waiting, using timing characteristics of the architecture

- **First solution:** [Caspi, Benveniste 2008]
  Slow down the nodes to mimic a synchronous architecture, global synchronisation

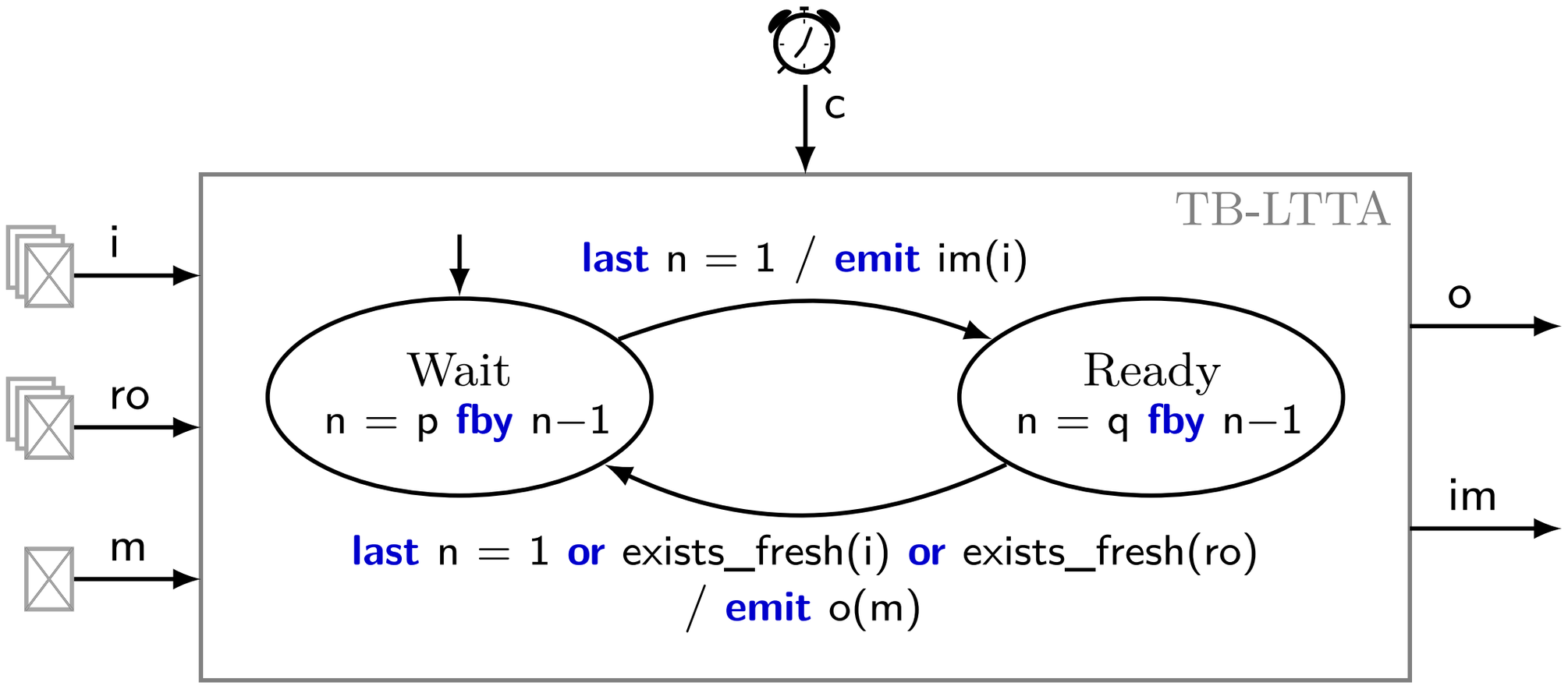
- **Our proposal:** Relax broadcast assumption, localise synchronisations

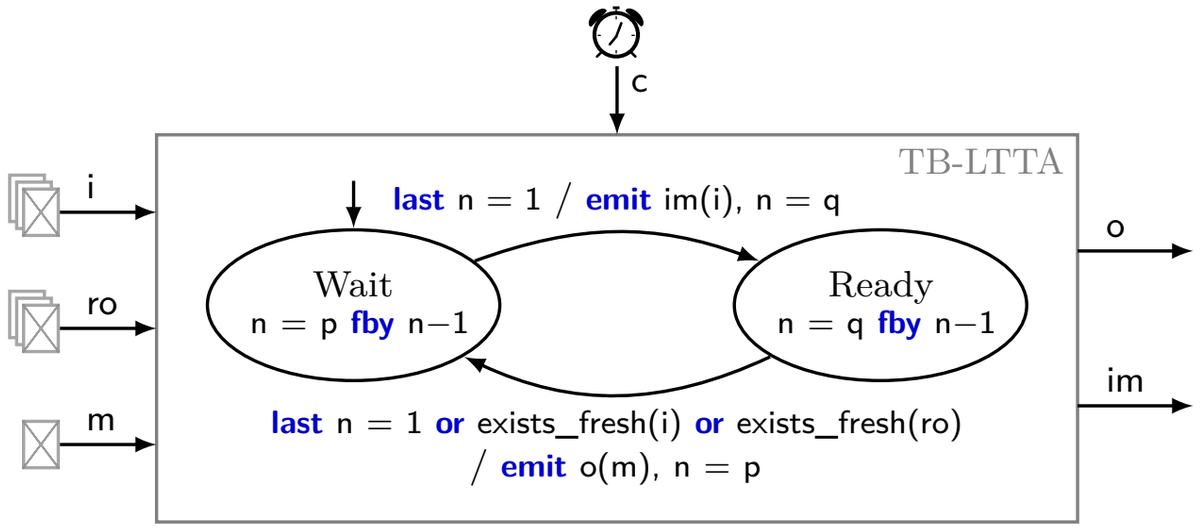
# Time-Based LTTA

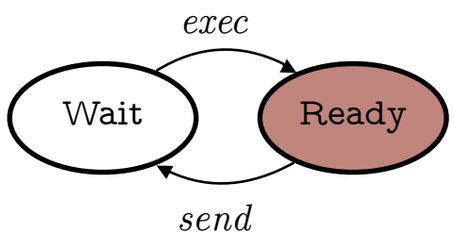
- Nodes alternate between **exec** and **send**
- Sender sees publication of all receivers

- **Idea:** At some point, a node can be sure that:
  - the last sent data has been read
  - a fresh value is available in the memory

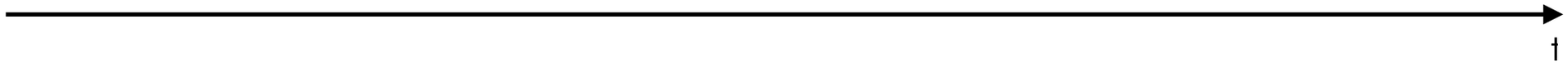
# Time-Based LTTA

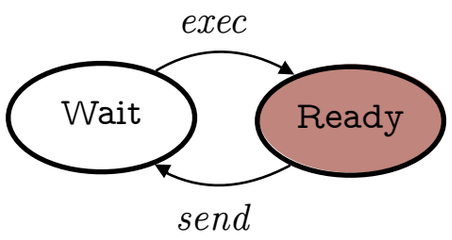




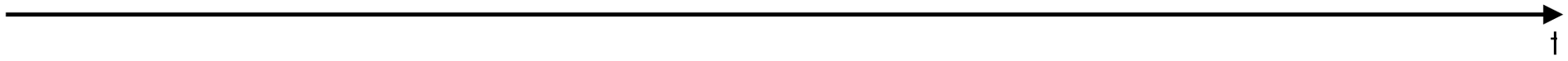



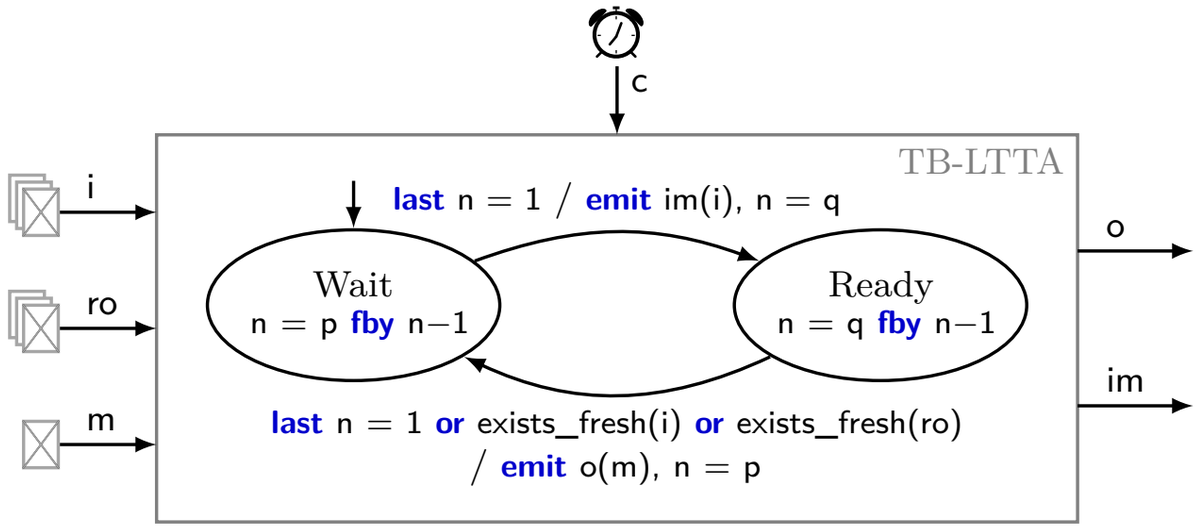

**Sender**

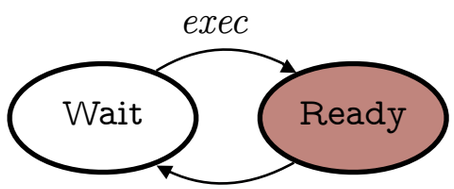


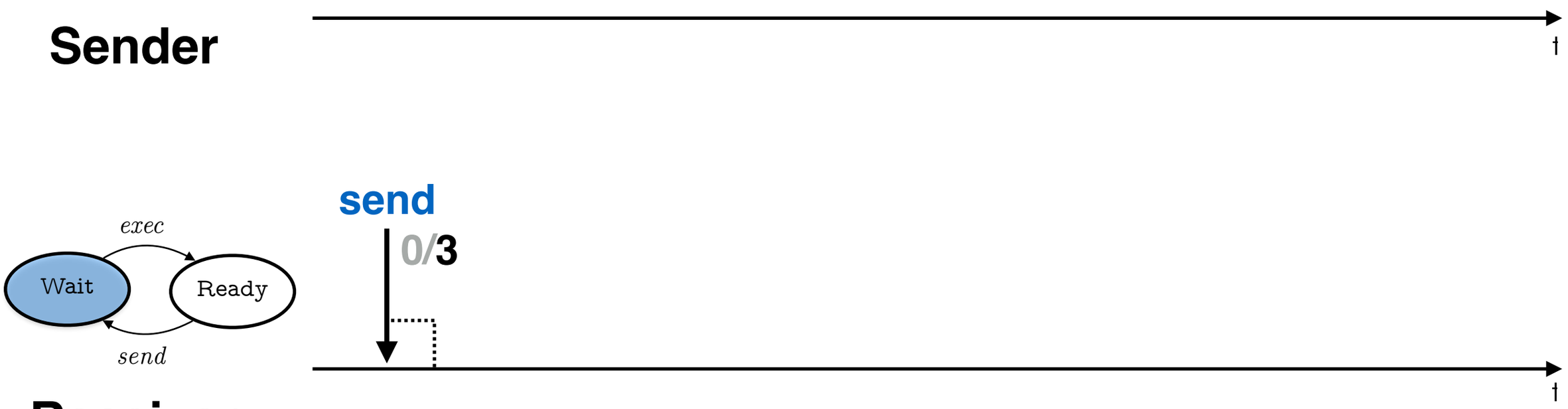

**Receiver**






**Sender**





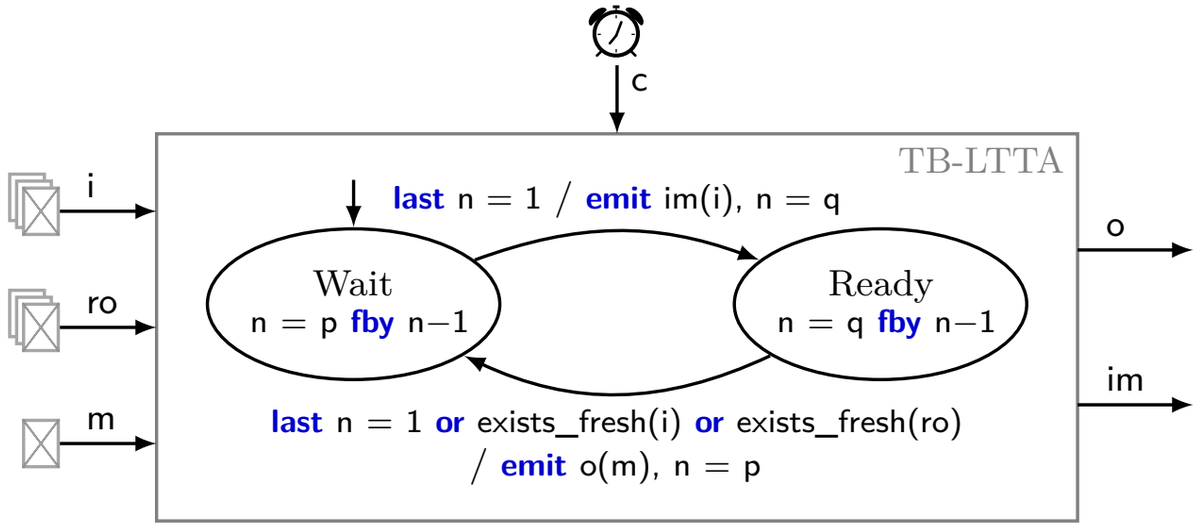


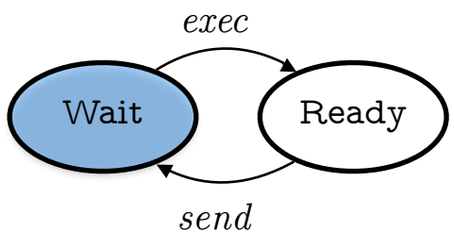
**send**

**0/3**

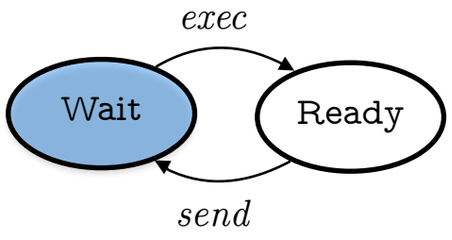



**Receiver**

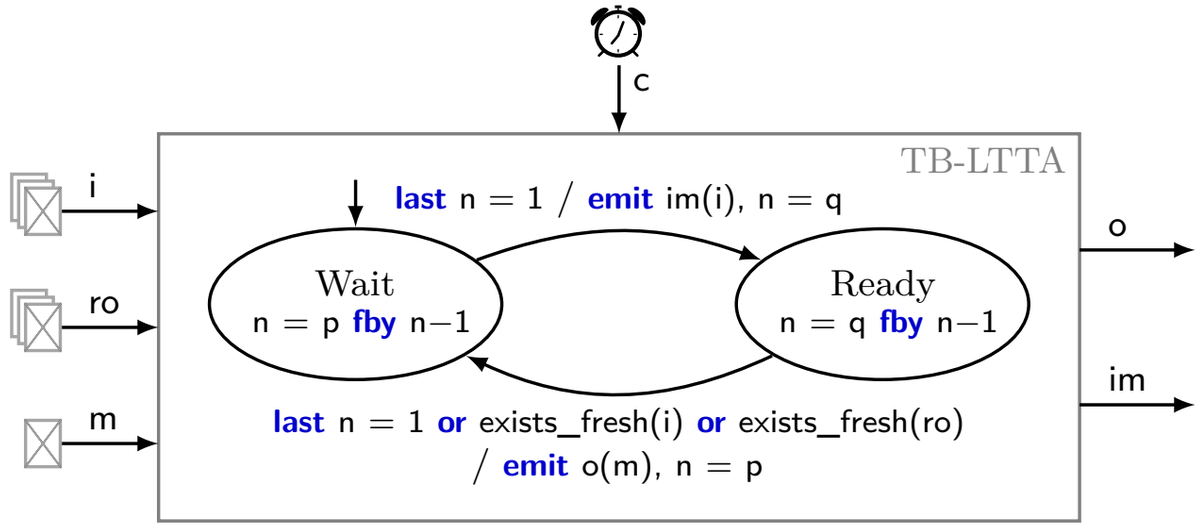


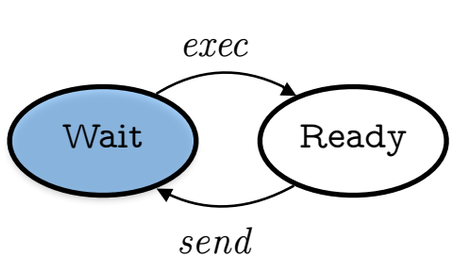




**Sender**

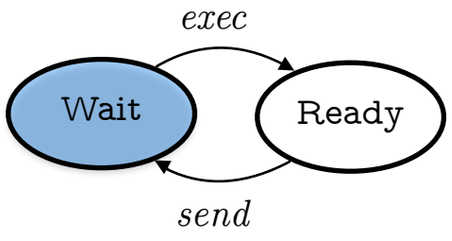




**Receiver**

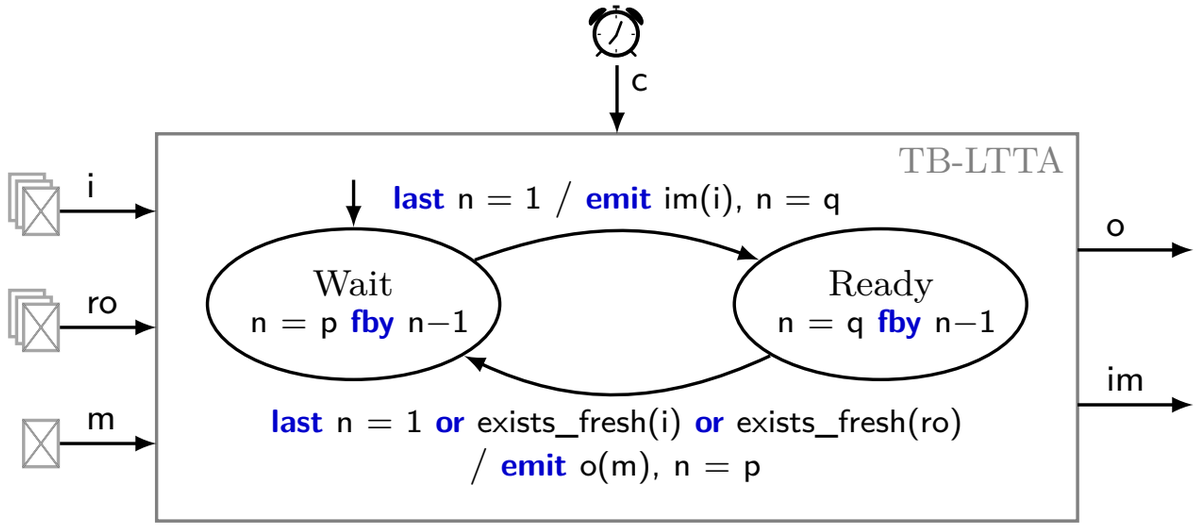


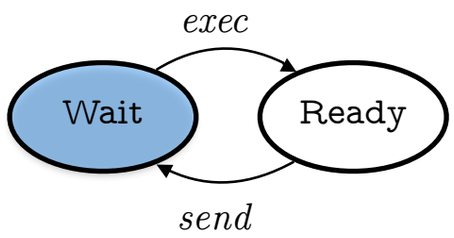




**Sender**

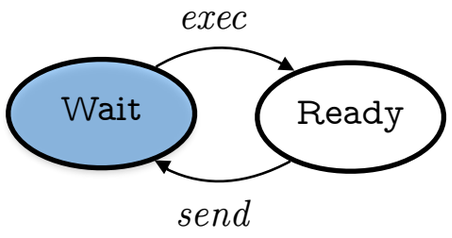




**Receiver**

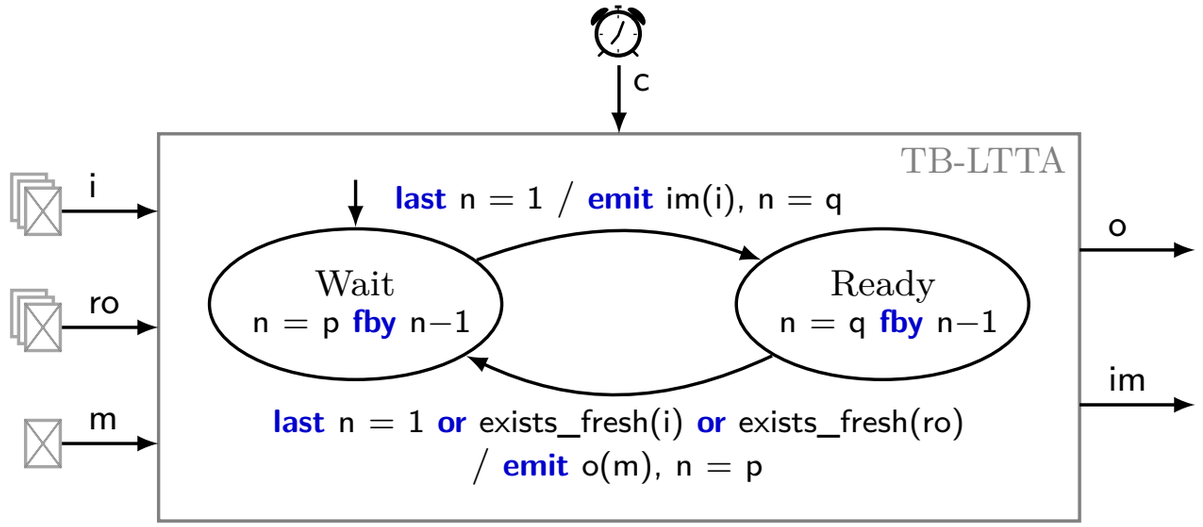


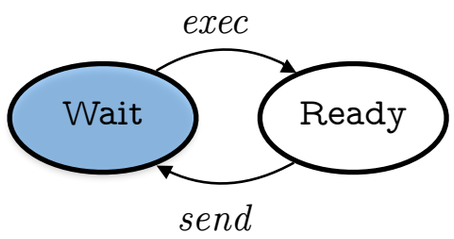




**Sender**

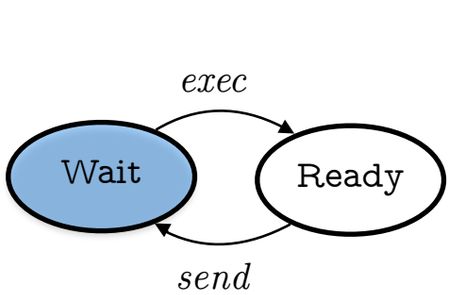




**Receiver**

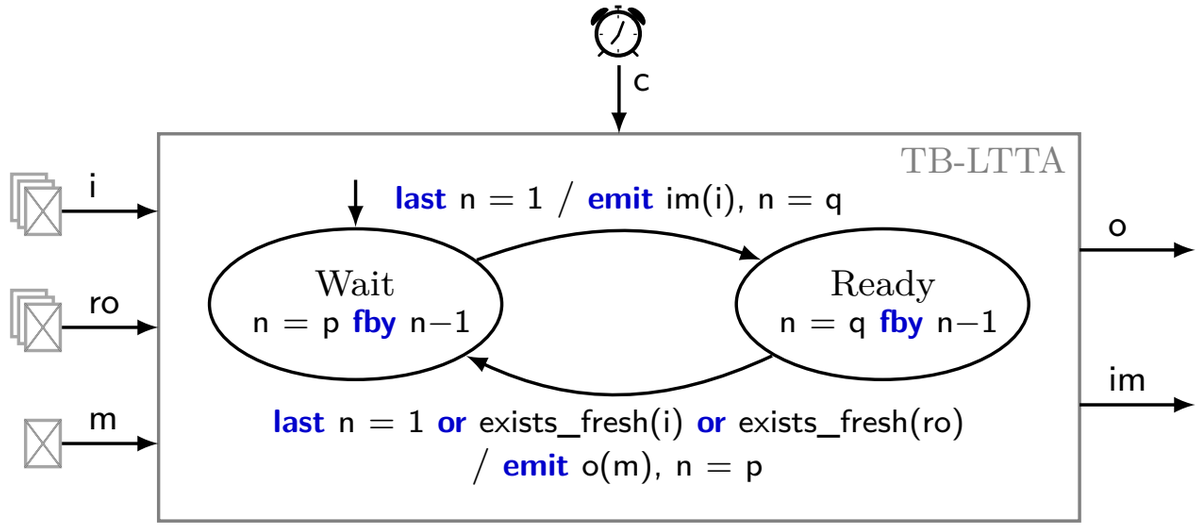


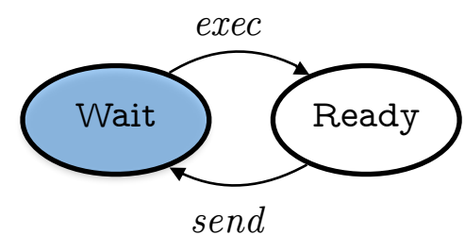




**Sender**

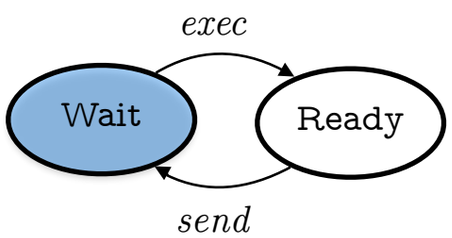




**Receiver**

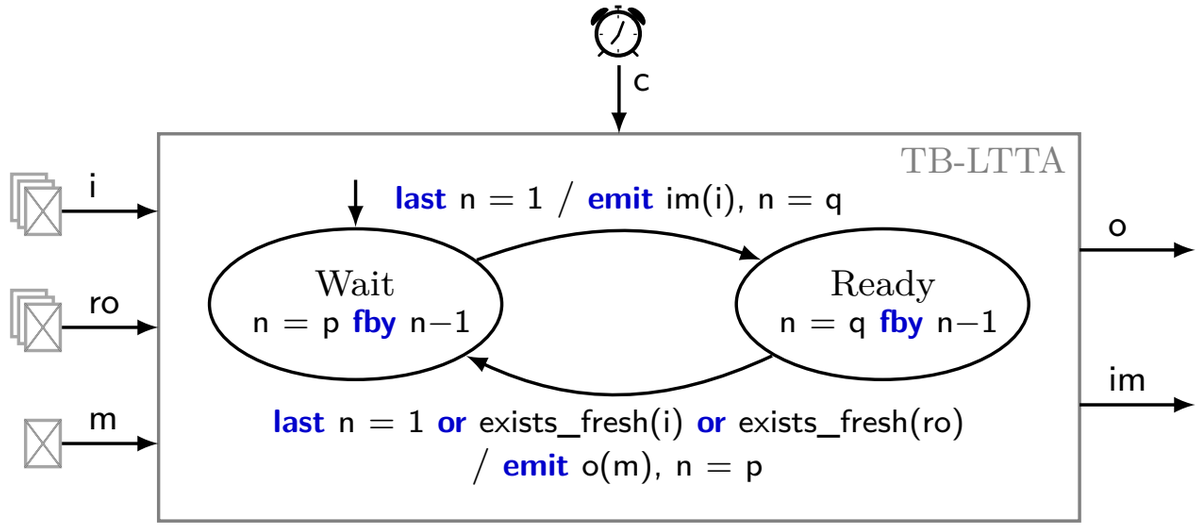


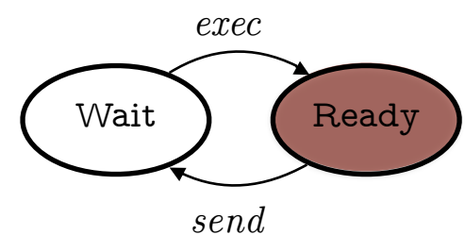




**Sender**

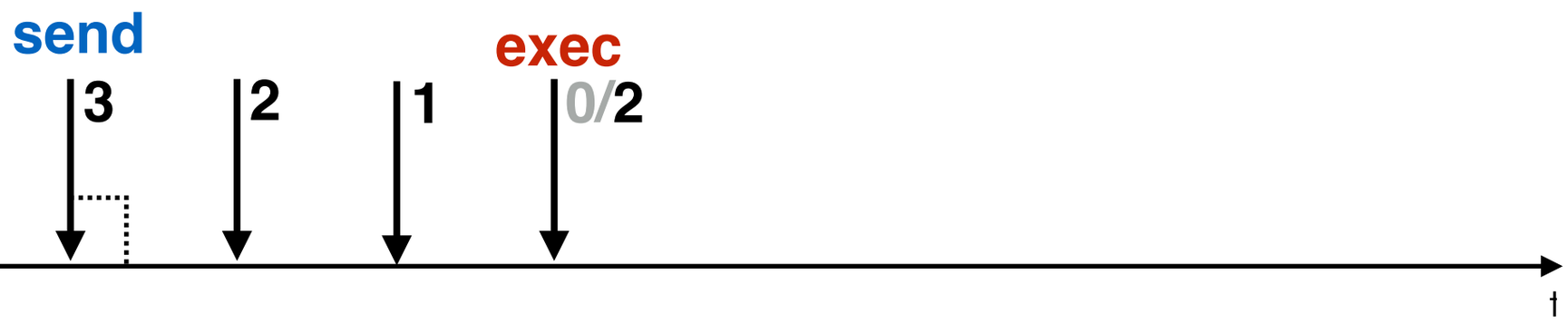



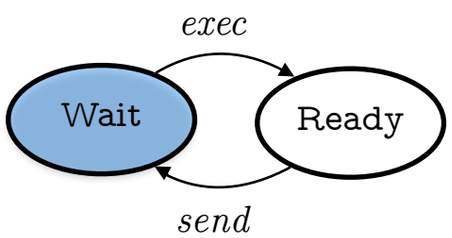

**Receiver**



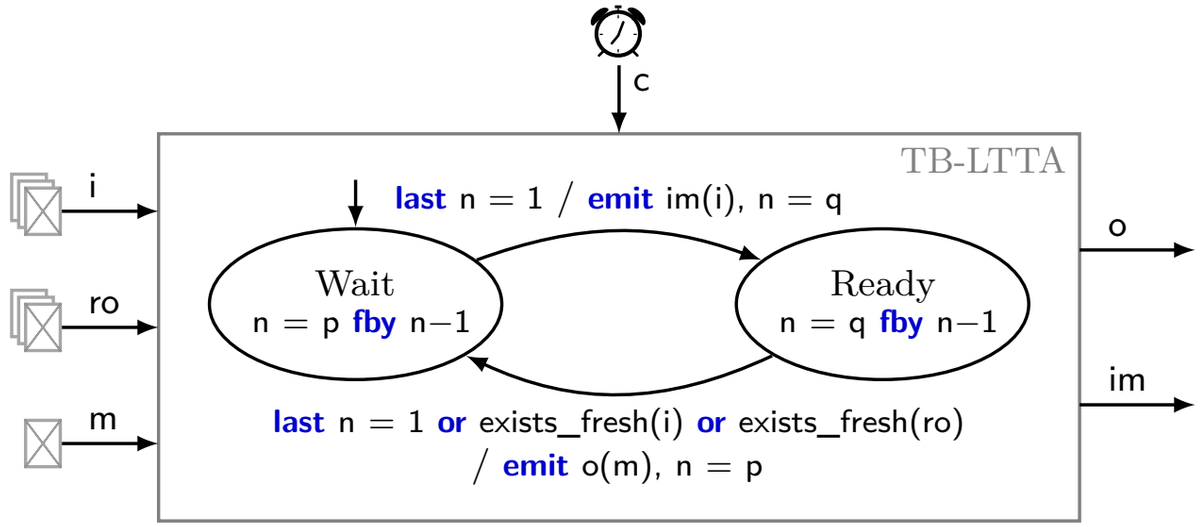



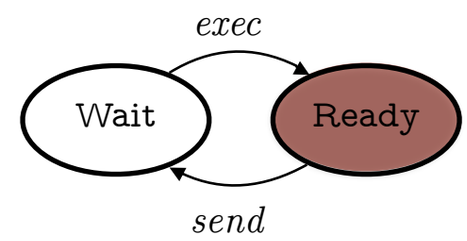

**Sender**



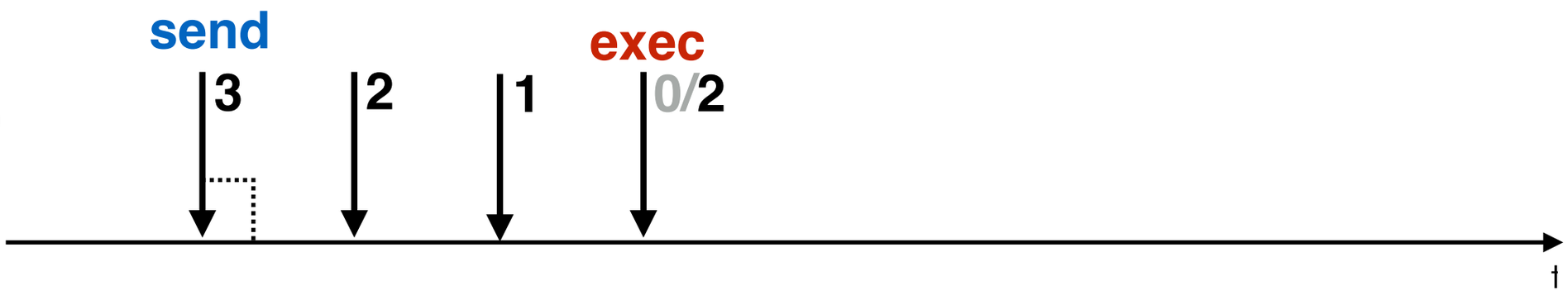



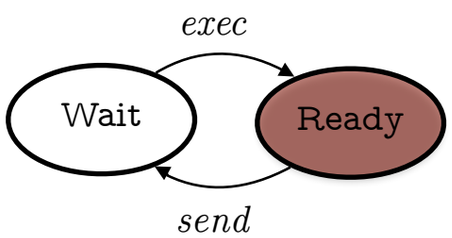

**Receiver**



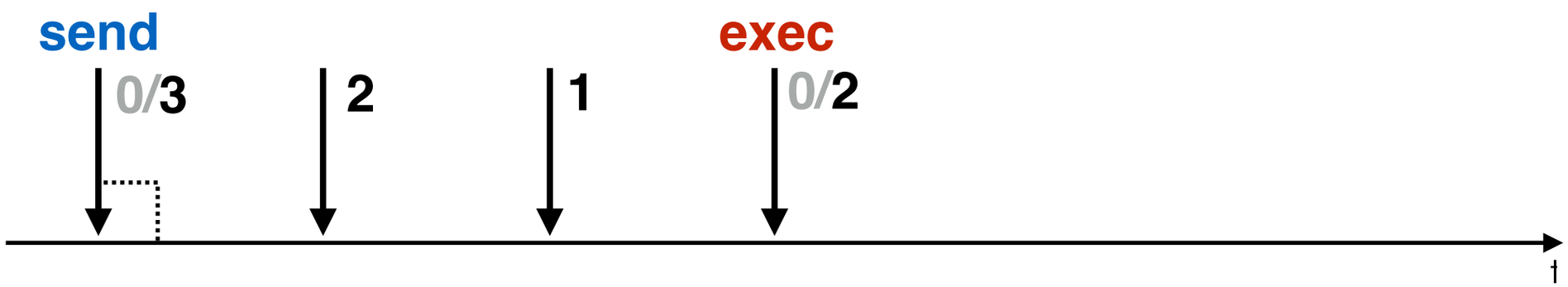



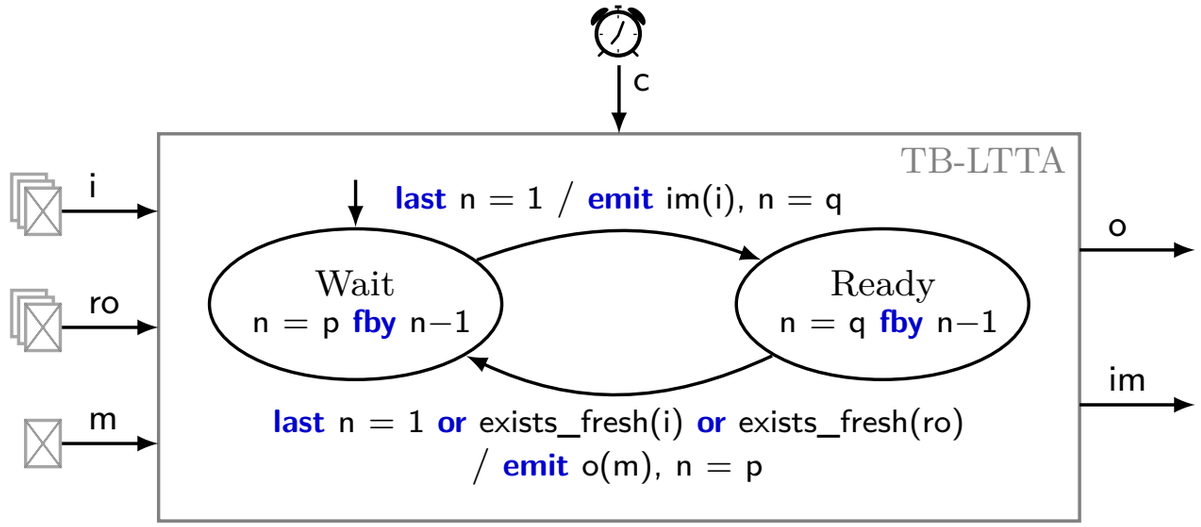

**Sender**

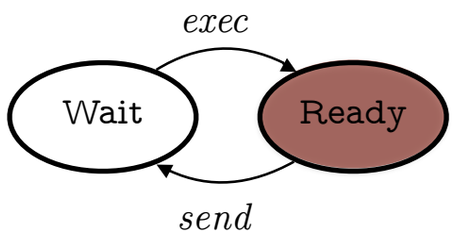


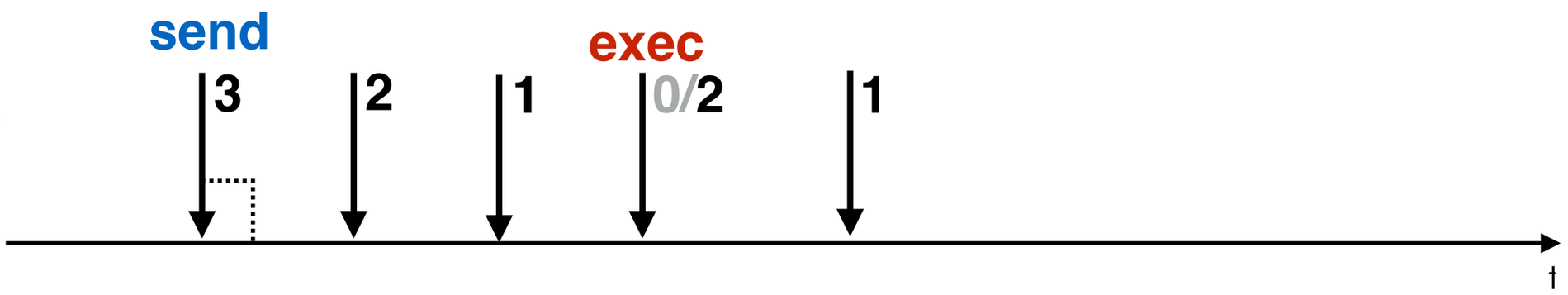

**Receiver**

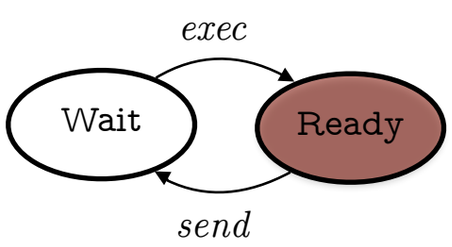




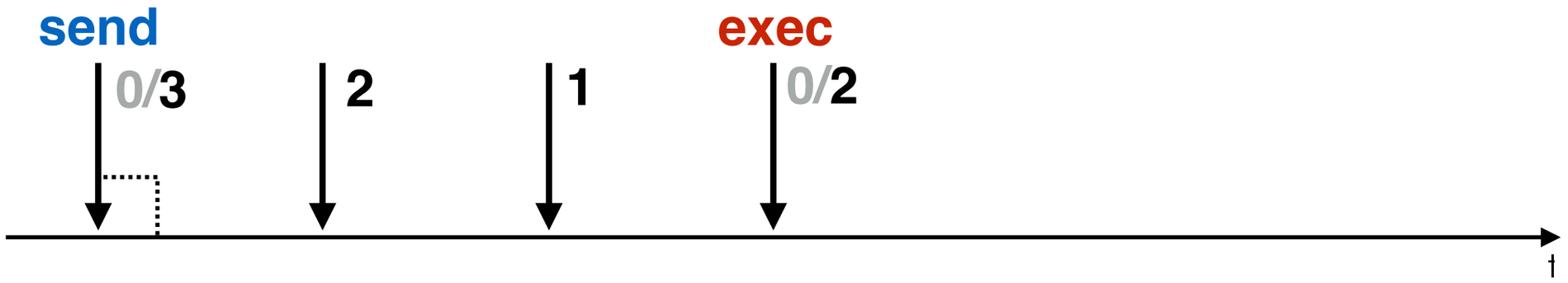



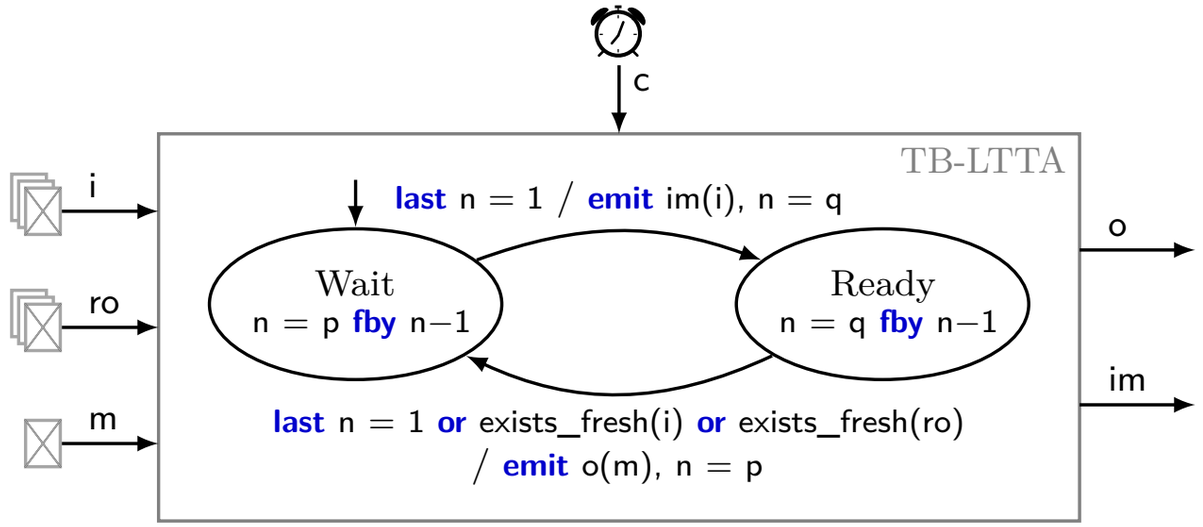

**Sender**

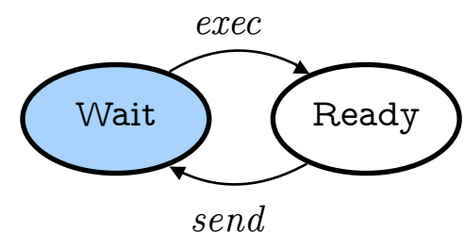


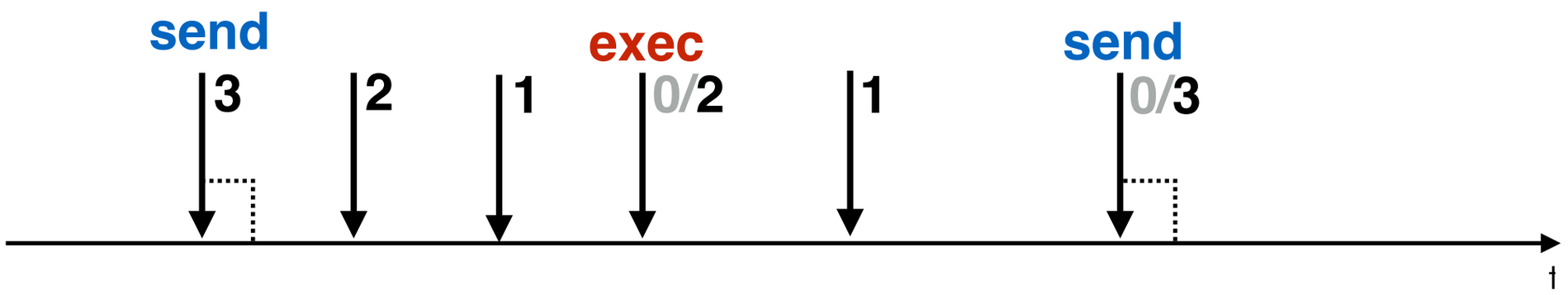

**Receiver**

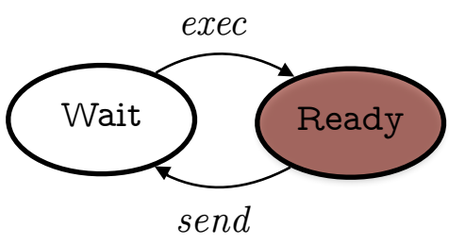




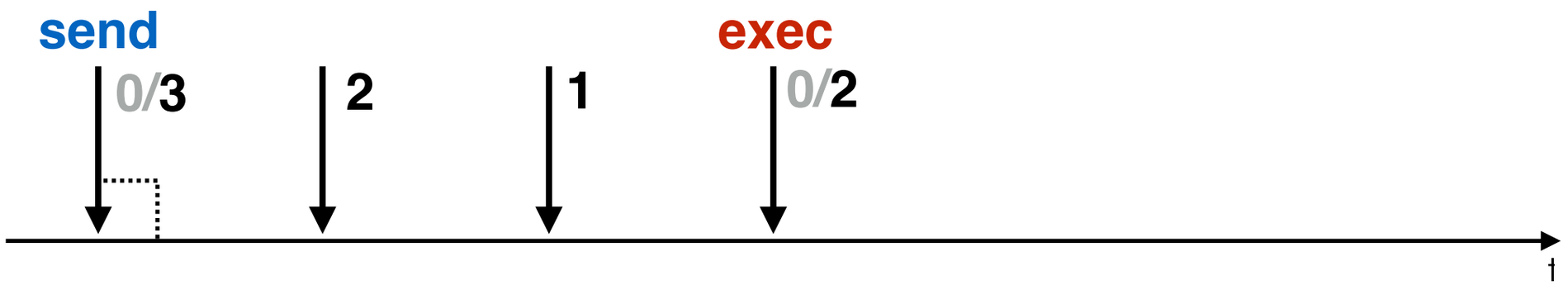



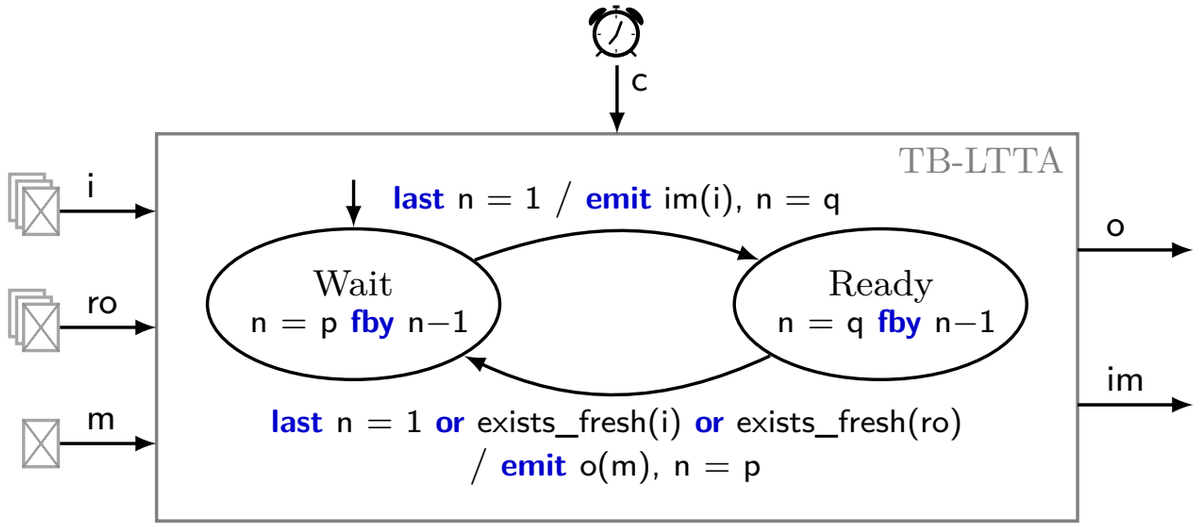

**Sender**

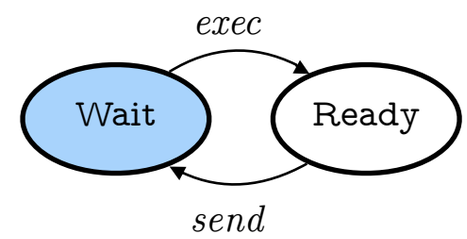


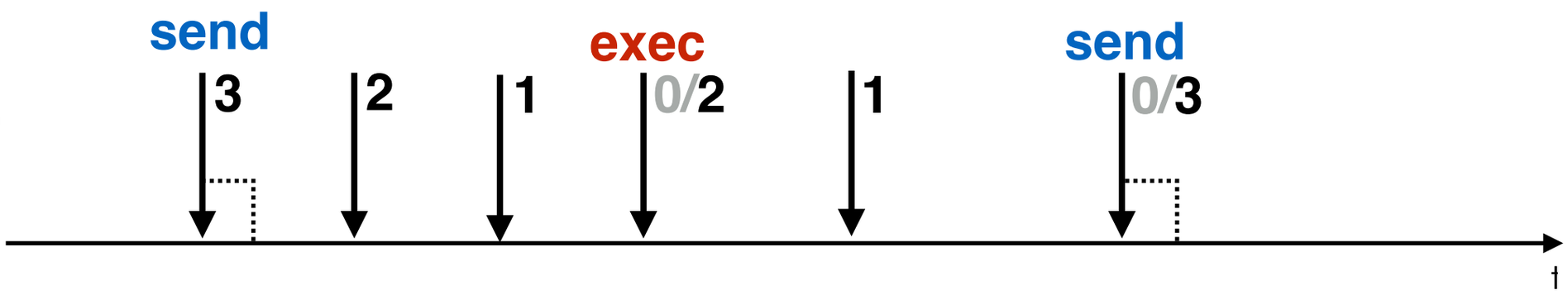

**Receiver**

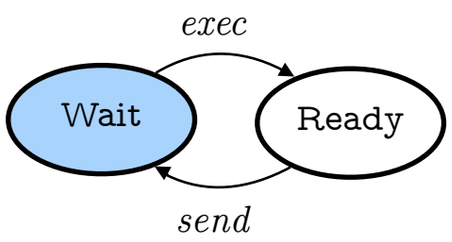




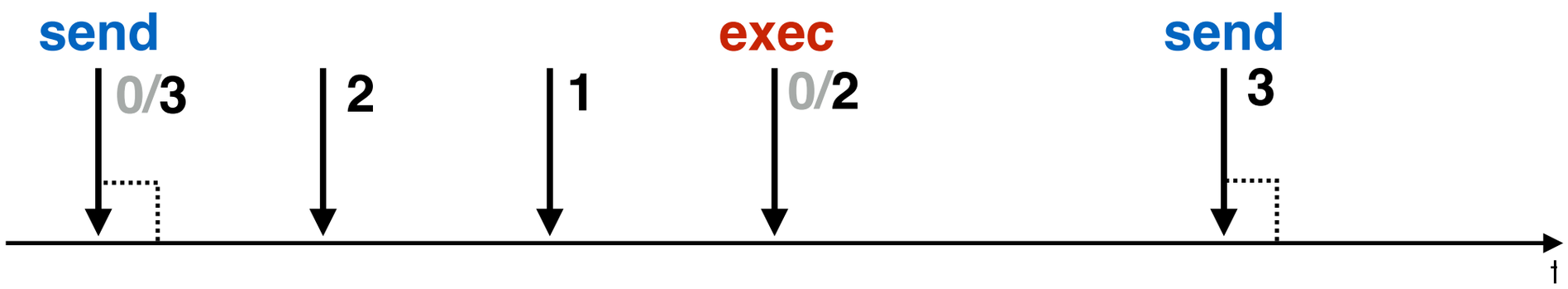



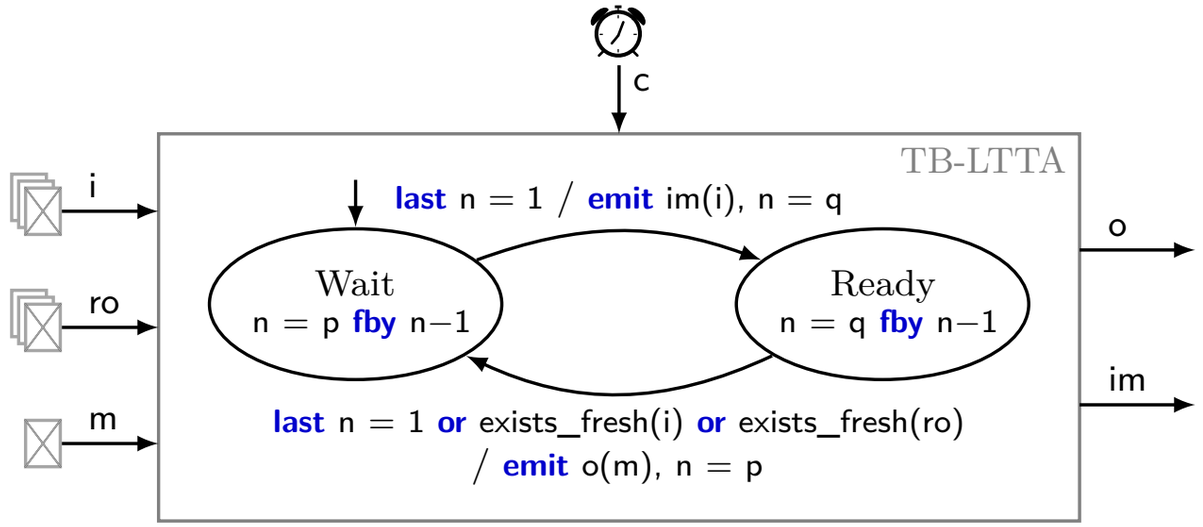

**Sender**

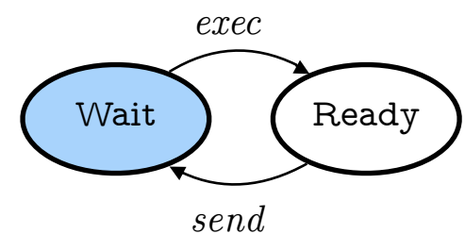


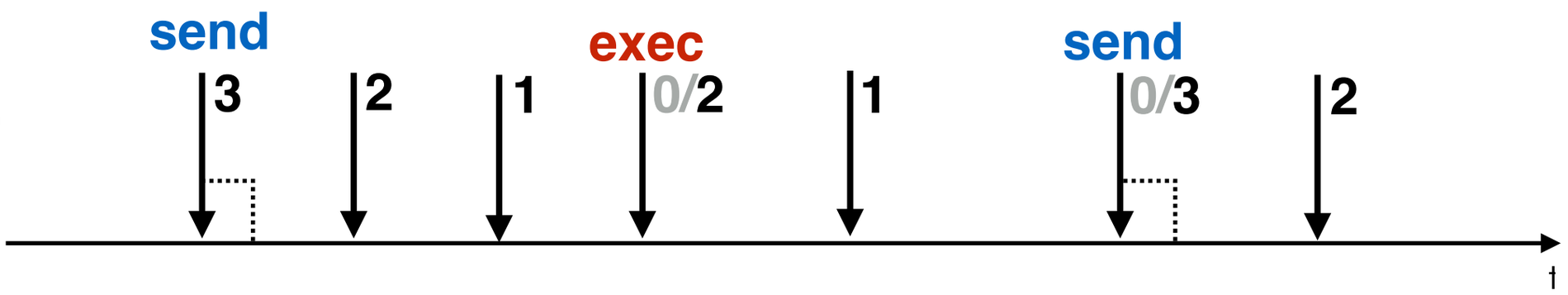

**Receiver**

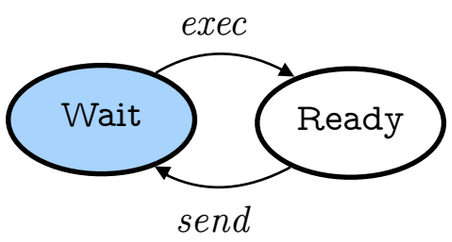




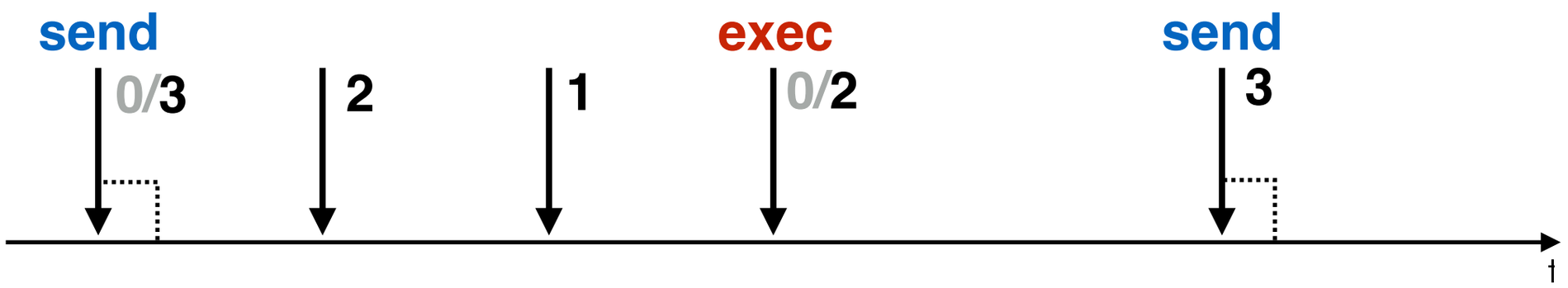



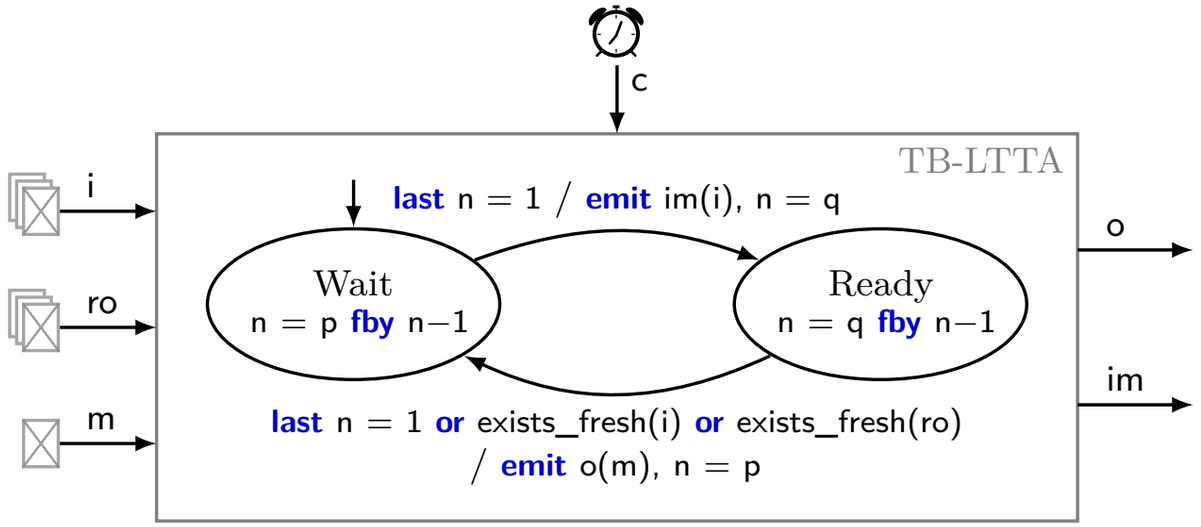

**Sender**

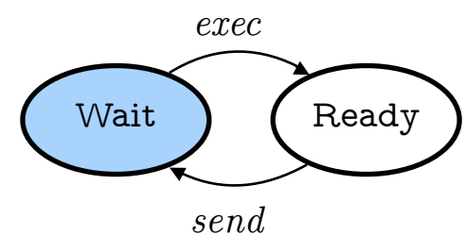


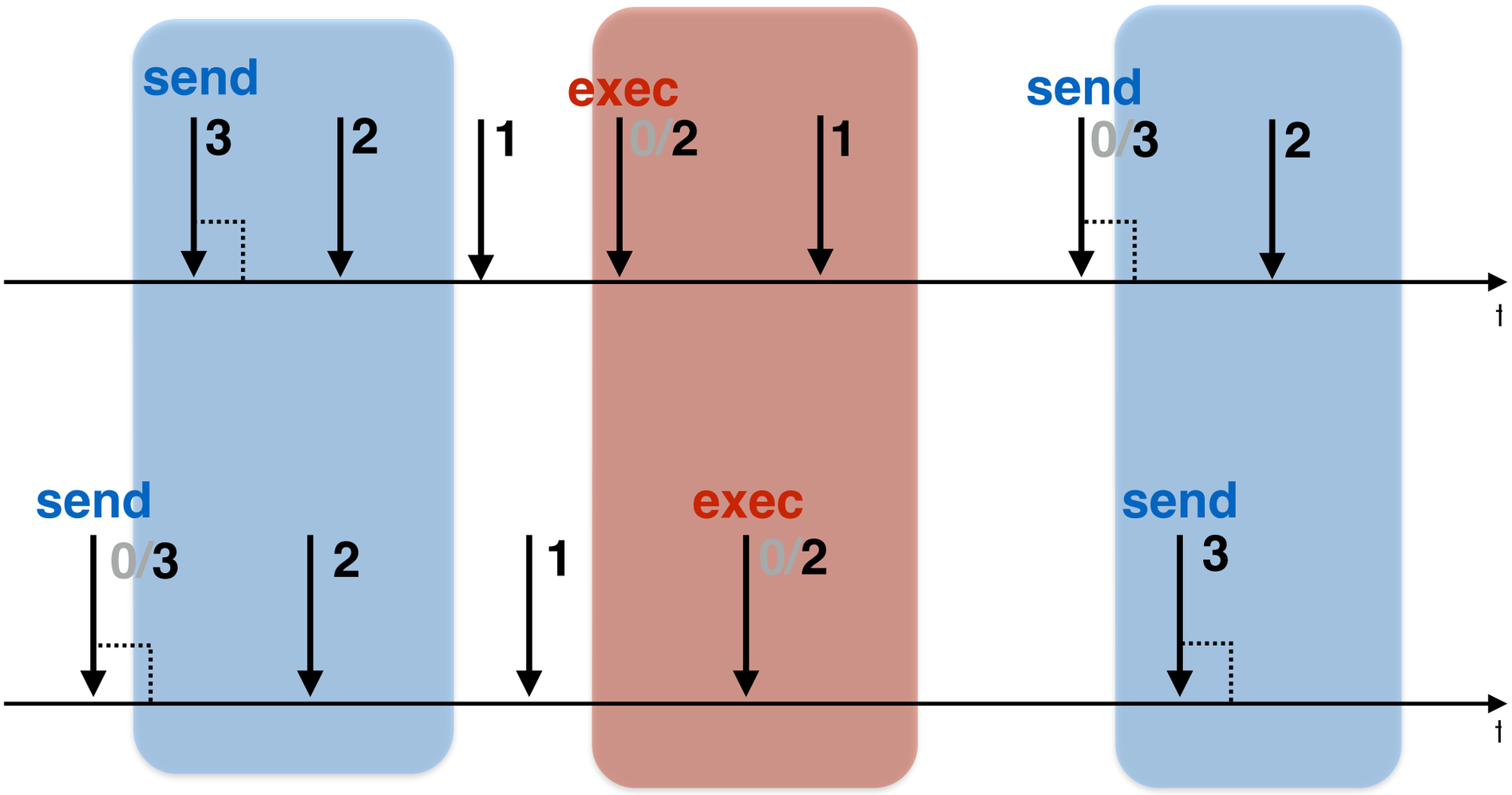

**Receiver**

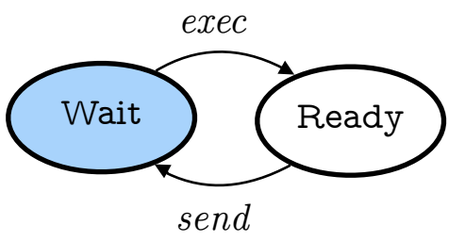





**Sender**




**Receiver**

# Time-Based LTTA

- **Theorem 1:**
  Composition of the controller and the embedded machine is always well-defined (no cycle)

- **Theorem 2:**
  Time-based LTTA preserves the Kahn semantics of the embedded application

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

# The Time-Based Protocol

**Theorem 2:** The following initial counter values ensure the preservation of the Kahn semantics

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

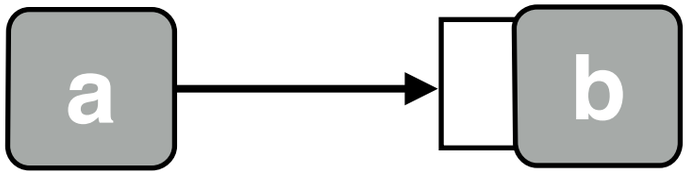
# The Time-Based Protocol

**Proof sketch**

- Worst case reasoning
- Tuning constants $p$ and $q$ (counter initial values)
- Ensure that the receiver always reads the proper data

□

# The Time-Based Protocol

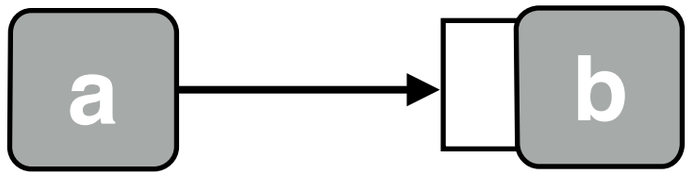



$$p > \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}}$$

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

**Property 1:** $W_{k-1}^{a} \prec E_{k}^{b}$

# The Time-Based Protocol



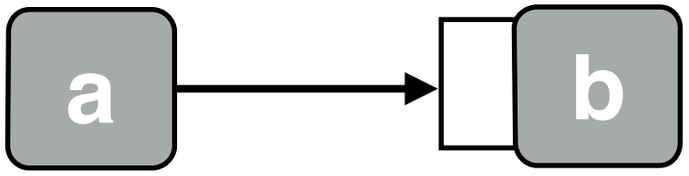

$$p > \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}}$$

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

**Property 1:** $W_{k-1}^{a} \prec E_{k}^{b}$

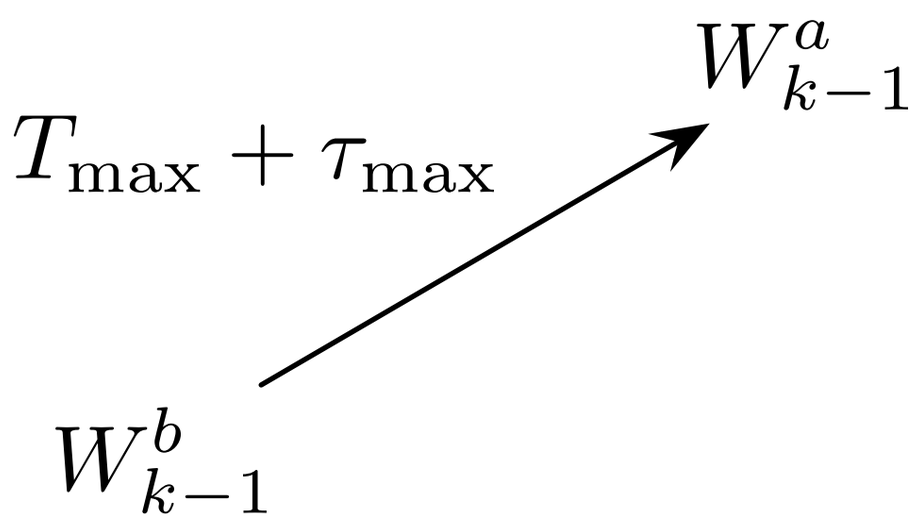
$$W_{k-1}^{b}$$

# The Time-Based Protocol

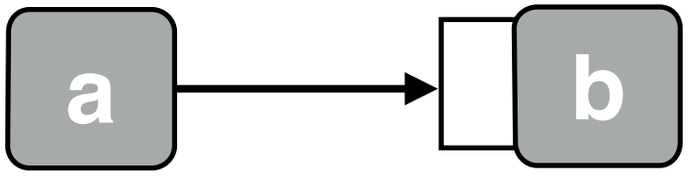



$$
\begin{aligned}
p &> \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} \\
q &> \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)
\end{aligned}
$$

**Property 1:** $W_{k-1}^{a} \prec E_{k}^{b}$

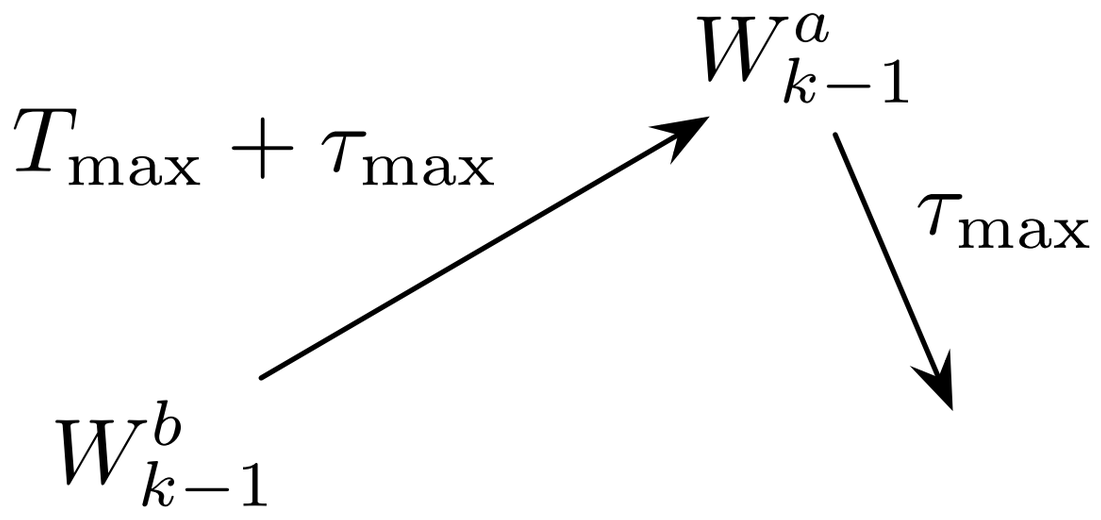


# The Time-Based Protocol

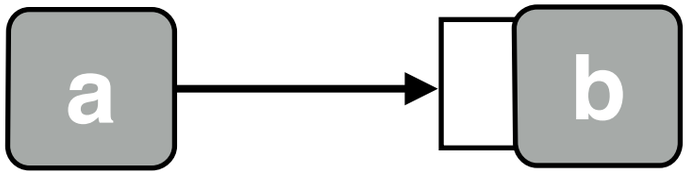



$$
\begin{aligned}
p &> \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} \\
q &> \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)
\end{aligned}
$$

**Property 1:** $W_{k-1}^{a} \prec E_{k}^{b}$

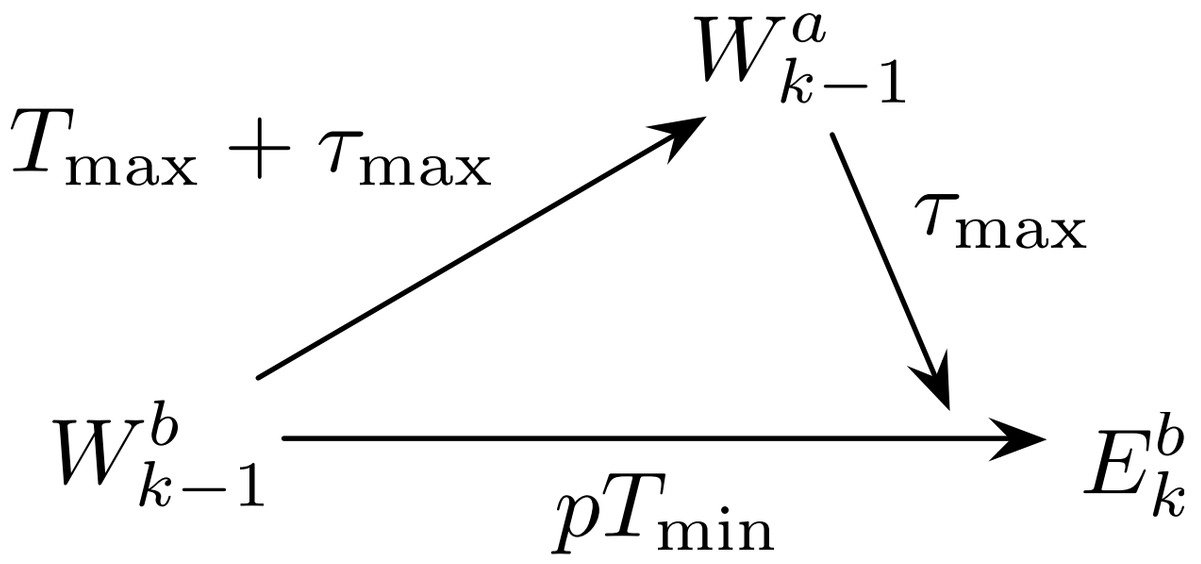


# The Time-Based Protocol

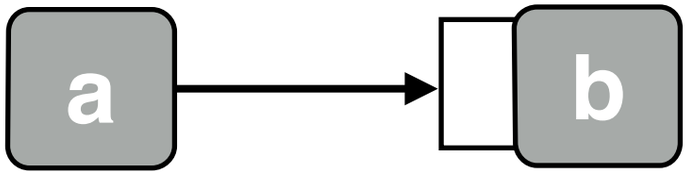



$$
\begin{aligned}
p &> \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} \\
q &> \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)
\end{aligned}
$$

**Property 1:** $W_{k-1}^{a} \prec E_{k}^{b}$



# The Time-Based Protocol



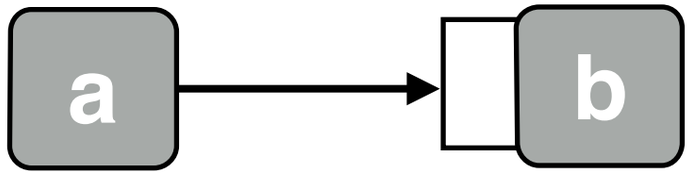

$$p > \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}}$$

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

**Property 1:** $W_{k-1}^{a} \prec E_{k}^{b}$

**Property 2:** $E_{k}^{b} \prec W_{k}^{a}$

# The Time-Based Protocol




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

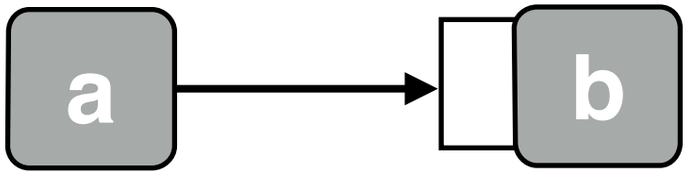
$$q > \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)$$

**Property 1:** $W_{k-1}^{a} \prec E_{k}^{b}$

**Property 2:** $E_{k}^{b} \prec W_{k}^{a}$

$W_{k-1}^{a}$

# The Time-Based Protocol




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

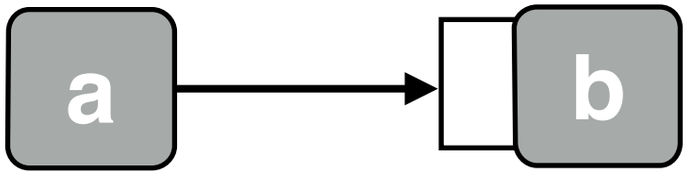
$$q > \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)$$

**Property 1:** $W_{k-1}^{a} \prec E_{k}^{b}$

**Property 2:** $E_{k}^{b} \prec W_{k}^{a}$

$$W_{k-1}^{a} \xrightarrow{pT_{\min}} E_{k}^{a}$$

# The Time-Based Protocol




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

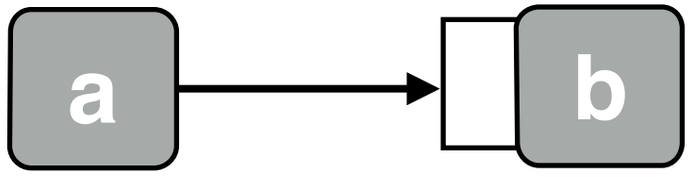
$$q > \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)$$

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

**Property 2:** $E^b_k \prec W^a_k$

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

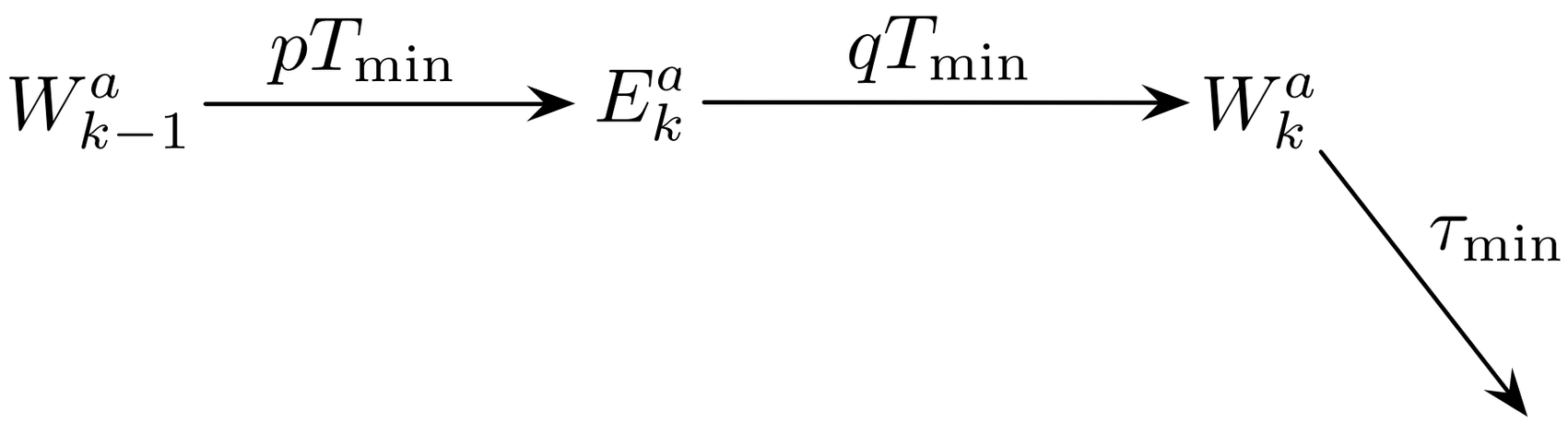
# The Time-Based Protocol



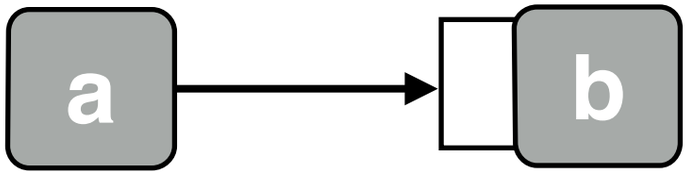

$$
\begin{aligned}
p &> \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} \\
q &> \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)
\end{aligned}
$$

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

**Property 2:** $E^b_k \prec W^a_k$



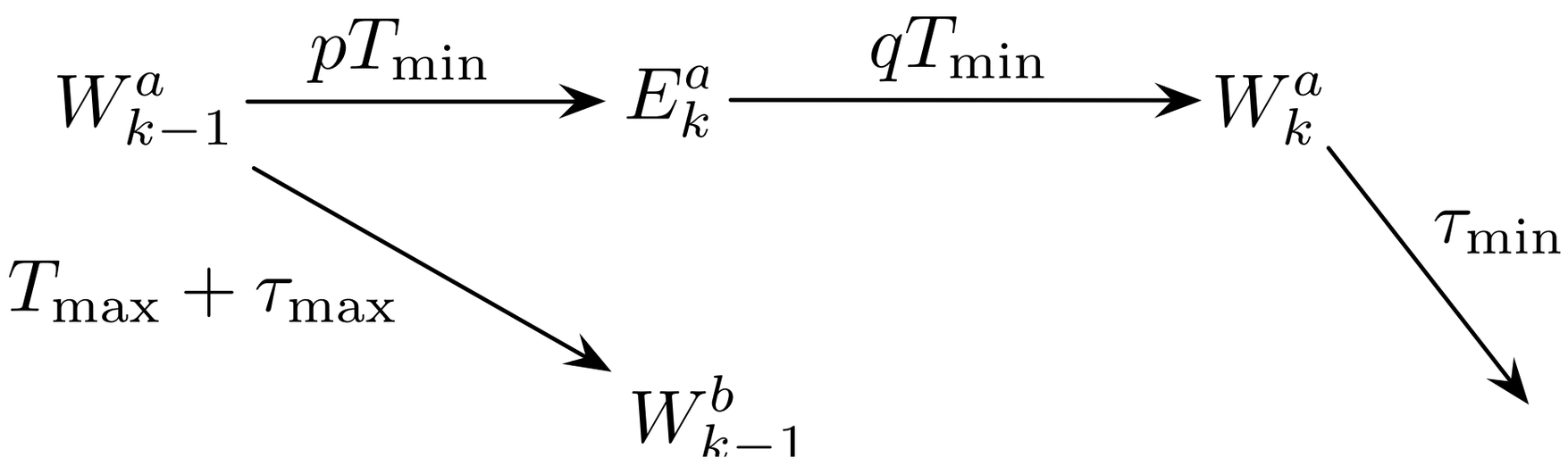
# The Time-Based Protocol



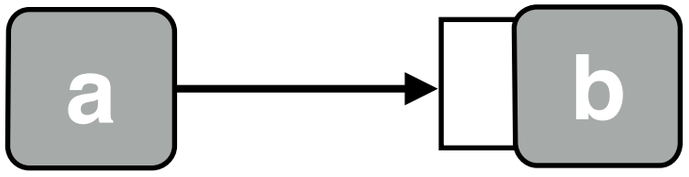

$$
\begin{aligned}
p &> \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} \\
q &> \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)
\end{aligned}
$$

**Property 1:** $W_{k-1}^a \prec E_k^b$

**Property 2:** $E_k^b \prec W_k^a$



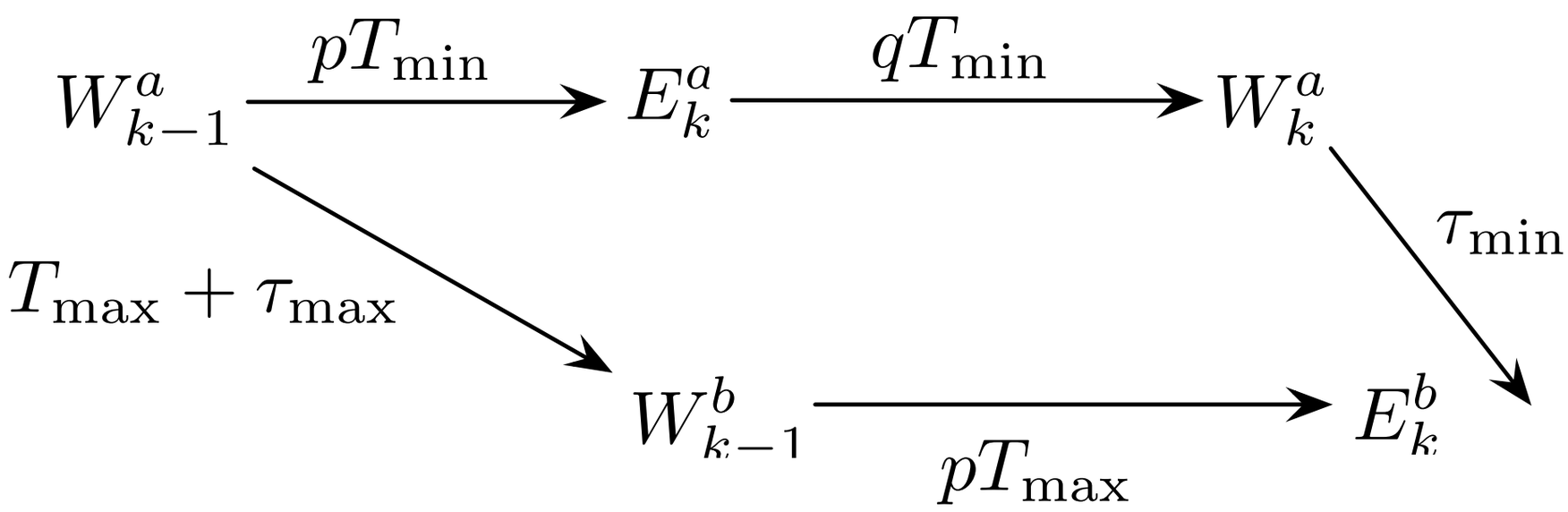
# The Time-Based Protocol




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

**Property 1:** $W_{k-1}^a \prec E_k^b$

**Property 2:** $E_k^b \prec W_k^a$

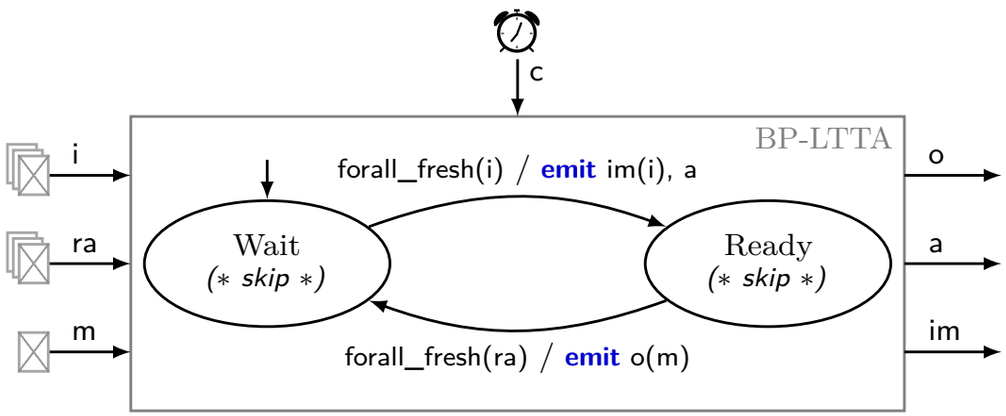


# The Time-Based Protocol

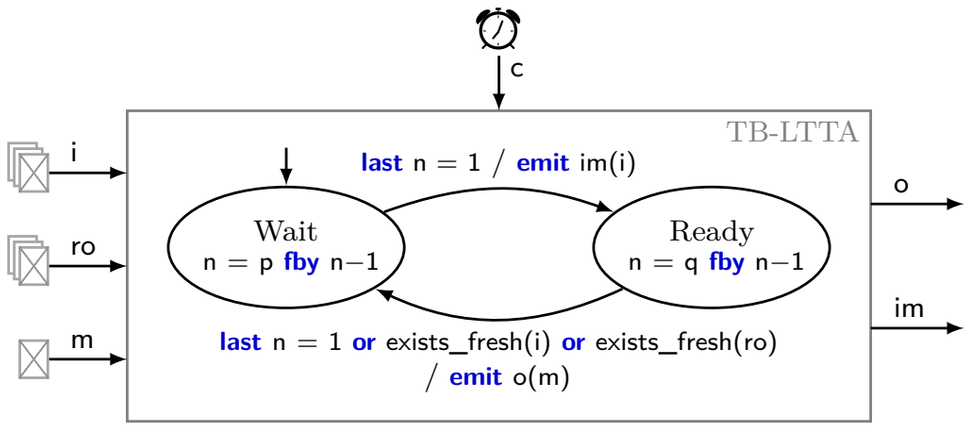
**Corollary:** The protocol ensures alternation between **exec** and **send** phases for each pair of communicating nodes

**Broadcast communication:** ensure clean alternation throughout the entire architecture (idem for back-pressure LTTA)

# Comparison




## Back-pressure




## Time-based

# Comparison

| | Time-Based | Back-Pressure |
|---|---|---|
| **Flexibility** | Require architecture specifications | Very flexible |
| **Robustness** | Can run in a degraded mode | Stuck if a node crash |
| **Fault Tolerance** | Can be programmed in the application | Implemented in the Middleware |
| **Communication** | Any | Any |
| **Pipelining** | Limited | Optimal |

# Comparison

| | Time-Based | Back-Pressure |
|---|---|---|
| **Flexibility** | Require architecture specifications | Very flexible |
| **Robustness** | Can run in a degraded mode | Stuck if a node crash |
| **Fault Tolerance** | Can be programmed in the application | Implemented in the Middleware |
| **Communication** | Any | Any |
| **Pipelining** | Limited | Optimal |

# Outline

1. What is an LTTA?
    1. Quasi-Periodic Architecture
    2. Synchronous Applications
2. General Framework

3. The two protocols
    1. Back-Pressure LTTA
    2. Time-Based LTTA

4. **What About Clock Synchronisation?**

# Global Clock Protocol

Central Master Synchronisation

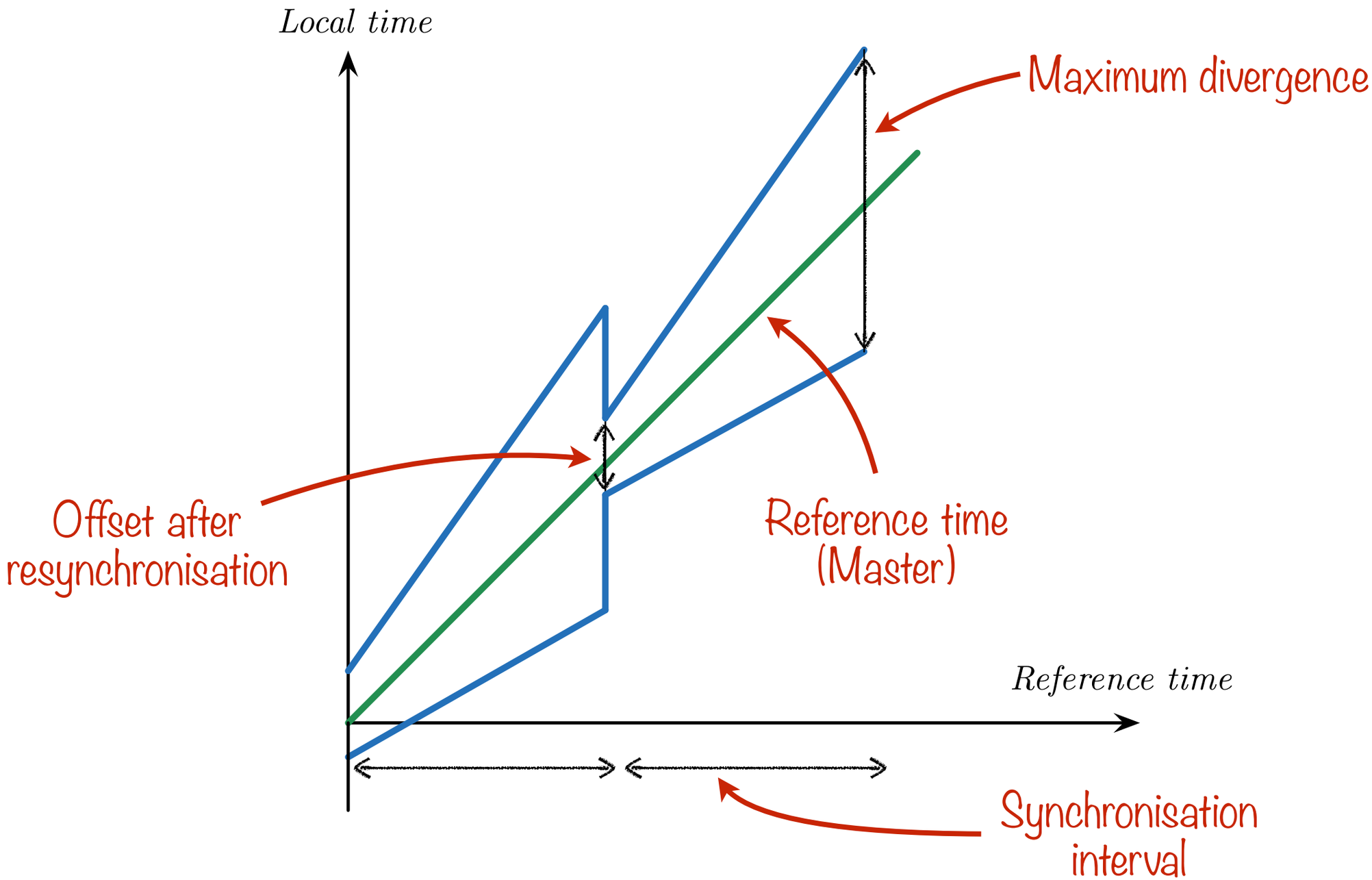
- **Goal:** Implement clock synchronisation on a quasi-periodic architecture

- We use the most efficient protocol for comparison purposes

- One node is used as a time reference for all other nodes: the **central master clock**

# Global Clock Protocol

## The Big Picture



# Global Clock Protocol

## Central Master Synchronisation

| | |
|---|---|
| Synchronisation interval | $R$ |
| Offset after resynchronisation | $\Phi = \tau_{\max} + T_{\max} - \tau_{\min}$ |
| Drift rate | $\rho = \frac{T_{\max}}{T_n} - 1 = \frac{T_{\max} - T_{\min}}{T_{\max} + T_{\min}}$ |
| Maximum divergence | $\Gamma = 2\rho R$ |
| Precision | $\Pi = \Phi + \Gamma$ |

# Global Clock Protocol

Lock-Step Execution

- Given the precision of the synchronisation one can build a global notion of  time or **macrosteps**

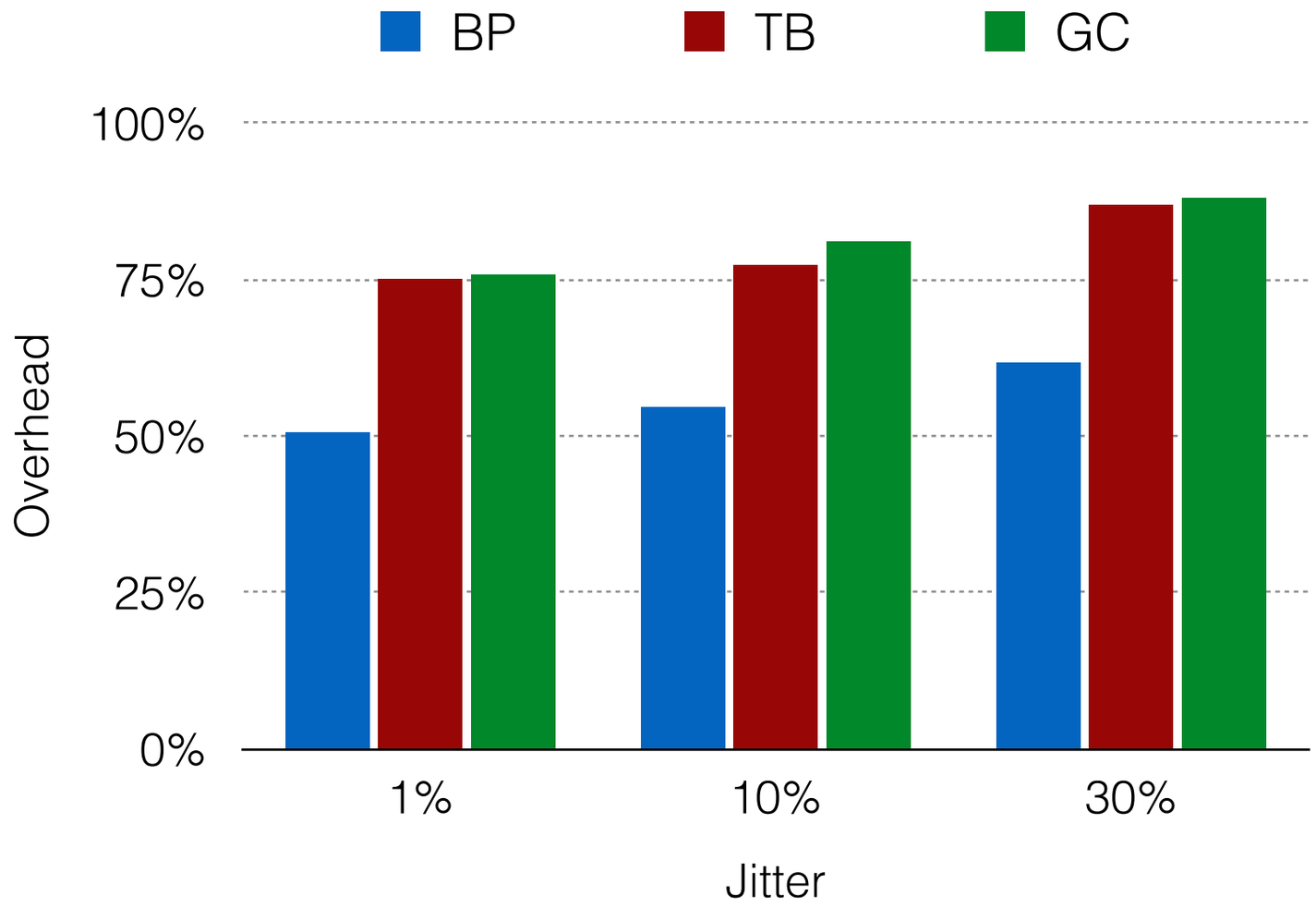
- A lock-step execution can be achieved if nodes execute once over 4 macrosteps [Kopetz 1997].

- We also need to wait for the transmission delay between execution steps (buffers of size 1)

# Overhead Comparison

Compared to synchronous execution*

Node : $10^{-2}$s  Transmission: $10^{-7}$s

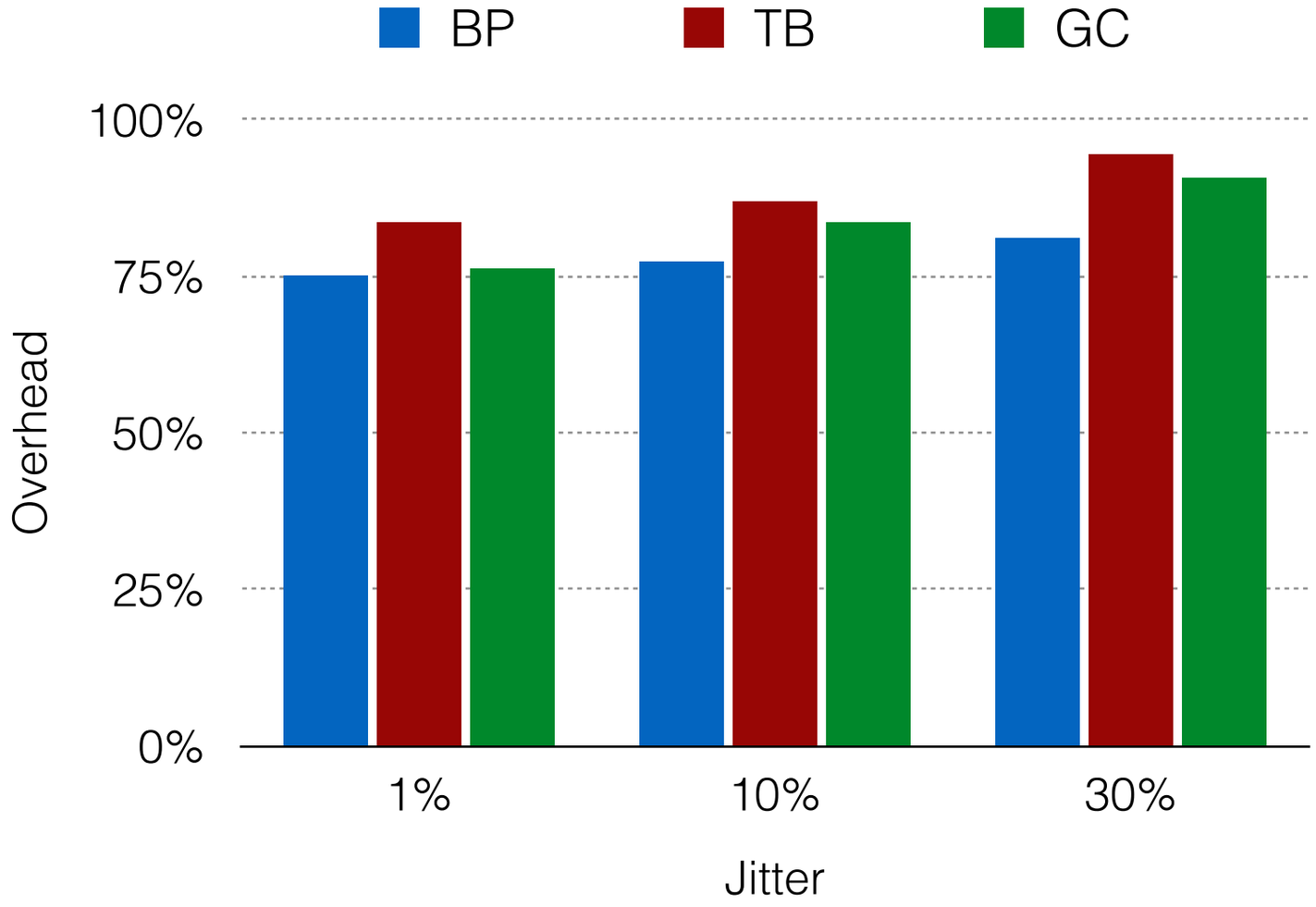



* The smaller, the better.

# Overhead Comparison

Compared to synchronous execution*

Node : $10^{-4}$s  Transmission: $10^{-4}$s



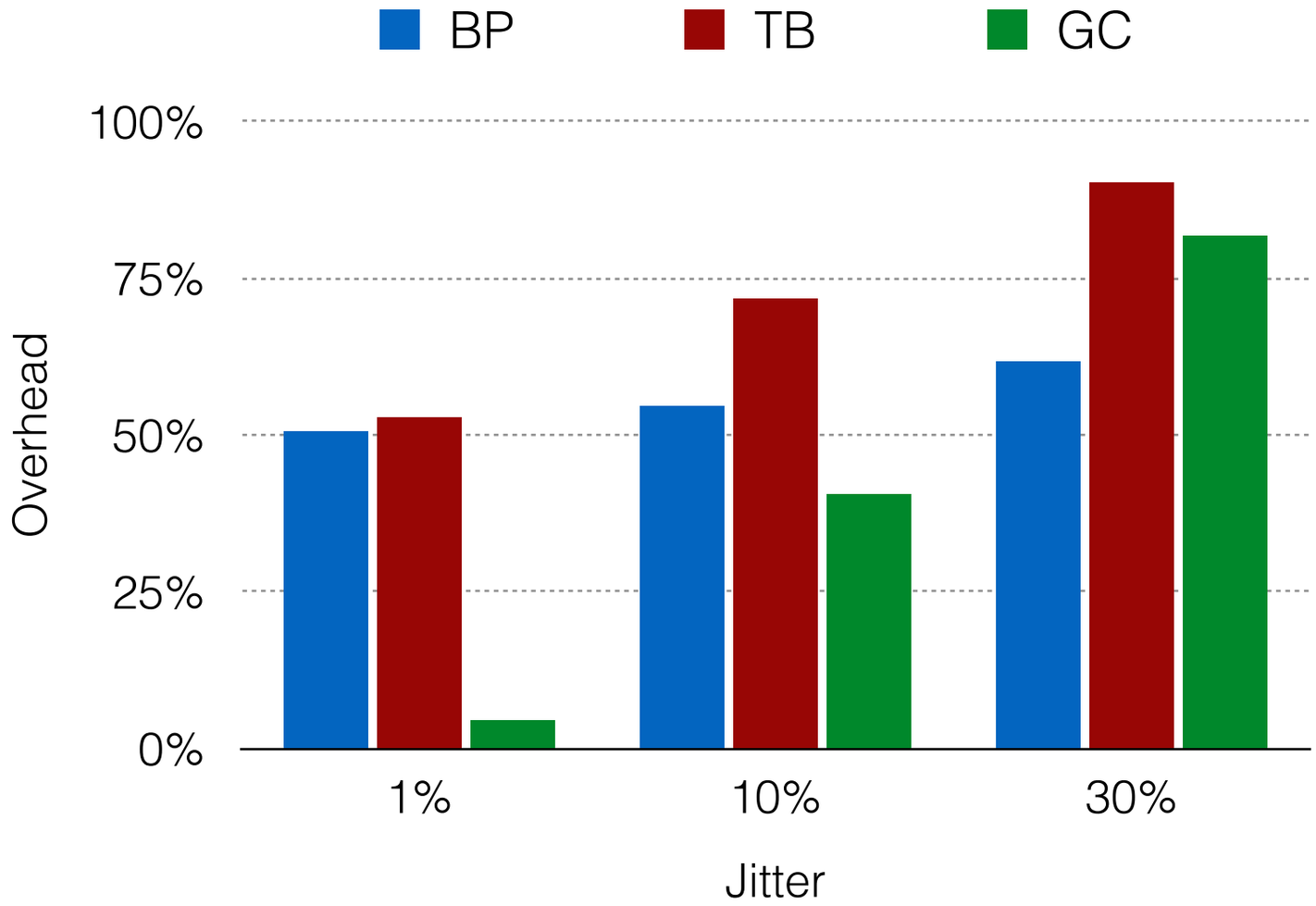



* The smaller, the better.

# Overhead Comparison

Compared to synchronous execution*

Node : $10^{-7}$s  Transmission: $10^{-2}$s






* The smaller, the better.

# Conclusion

- Our new model simplifies and clarifies those of previous papers

- A new proposition for the time-based protocol that does not require broadcast communication and does allow pipelining

- Model and Simulation of the protocols in Zélus
  Discrete model + link with continuous time

- Comparison with clock synchronisation deployed on the same architecture