Synchronous Modeling of Loosely Time-Triggered Architectures

Guillaume Baudart
Albert Benveniste

with many thanks to Timothy Bourke, Adrien Guatto and Marc Pouzet

Synchron'14 **Aussois, 01-12-2014**

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 are LTTAs?
2. Synchronous model
3. The two protocols
4. Comparison

Synchronous Applications

Network of communicating Mealy Machines

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

Semantics

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

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

Synchronous Applications

Network of communicating Mealy Machines

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

Basically, classic synchronous programs without clocks.

Example

```
let node from m = nat where
  rec nat = m -> pre nat + 1
```

Quasi-Periodic Architecture

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

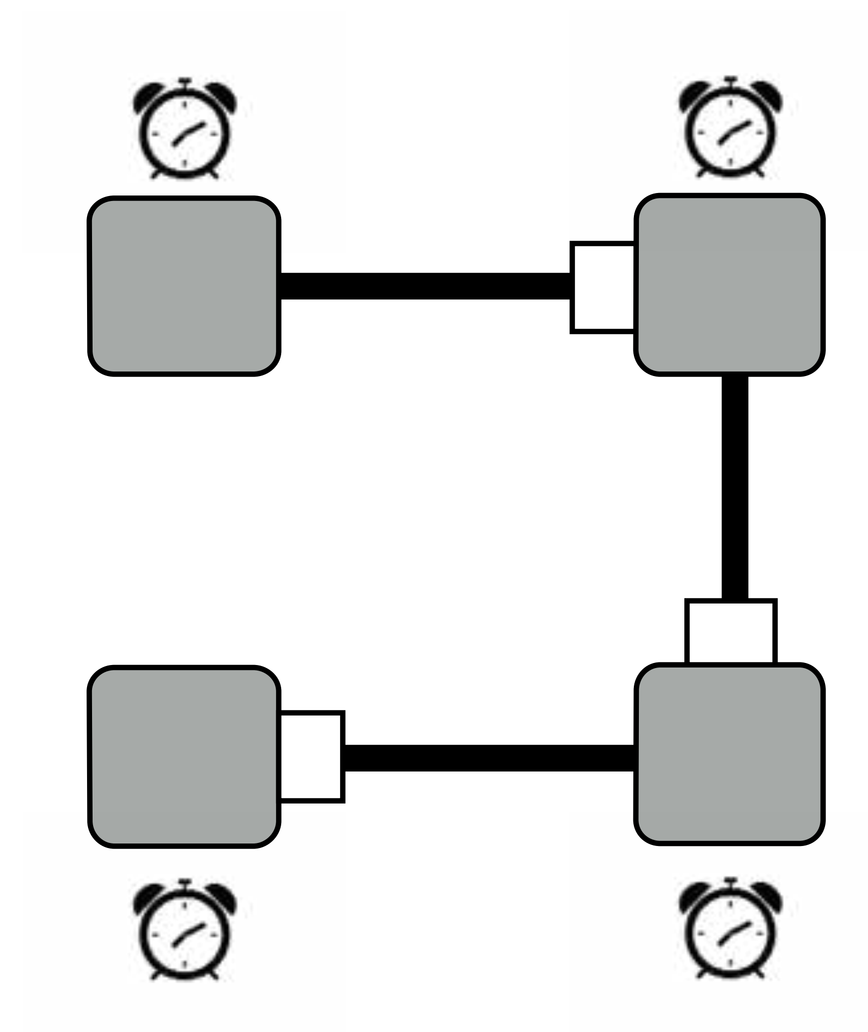
$$0 < T_{\min} \leq T^n \leq T_{\max} \quad \text{or}$$
$$T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon$$

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

- Buffered communication without message inversion or loss

- Bounded communication delay

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



Quasi-Periodic Architecture

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

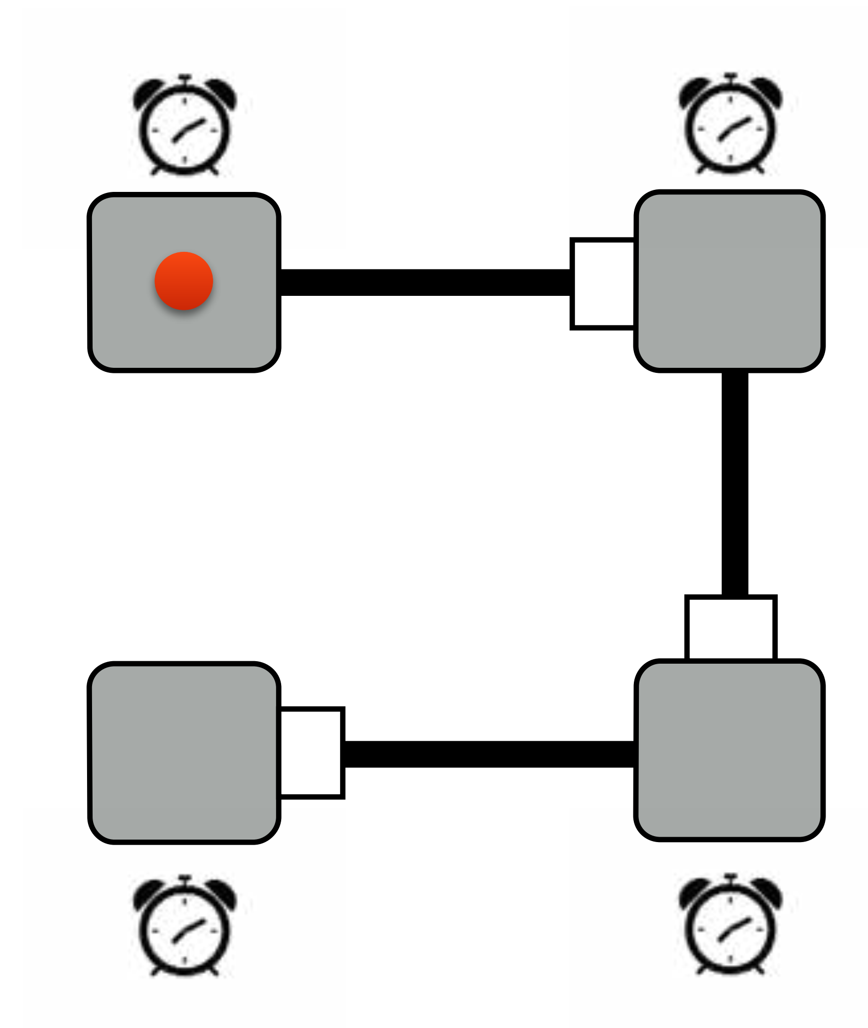
$$0 < T_{\min} \leq T^n \leq T_{\max} \quad \text{or}$$
$$T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon$$

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

- Buffered communication without message inversion or loss

- Bounded communication delay

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



Quasi-Periodic Architecture

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

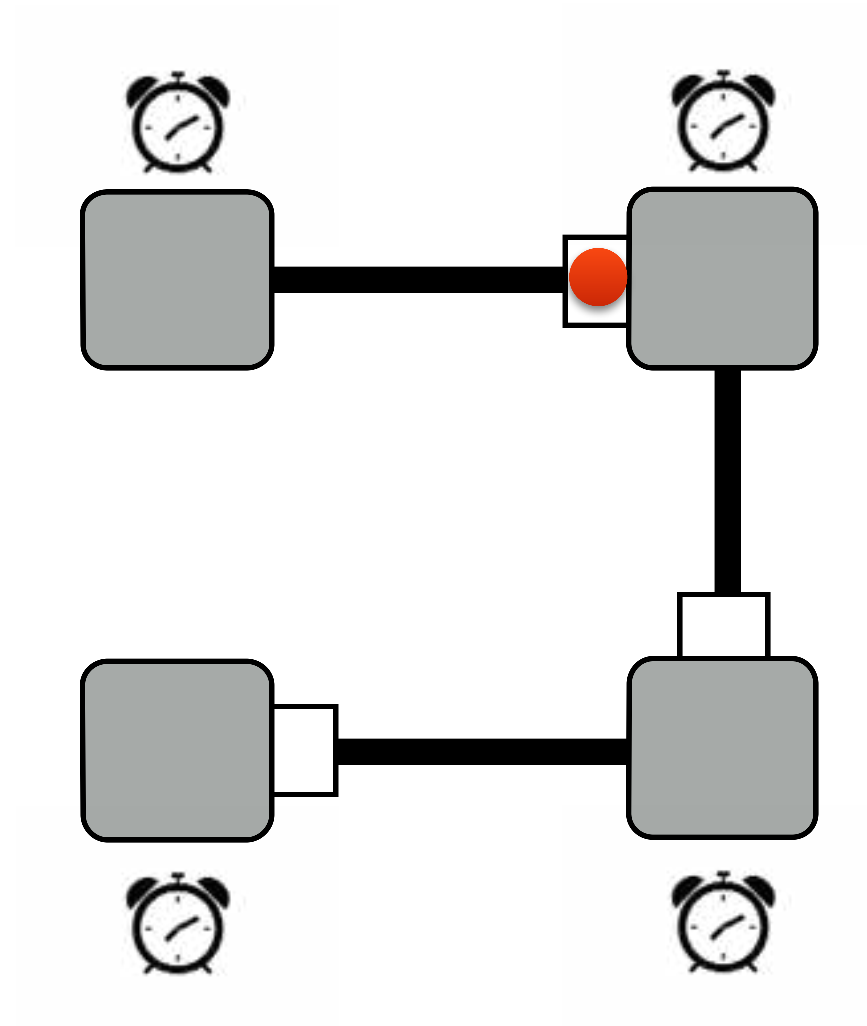
$$0 < T_{\min} \leq T^n \leq T_{\max} \quad \text{or}$$
$$T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon$$

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

- Buffered communication without message inversion or loss

- Bounded communication delay

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



Quasi-Periodic Architecture

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

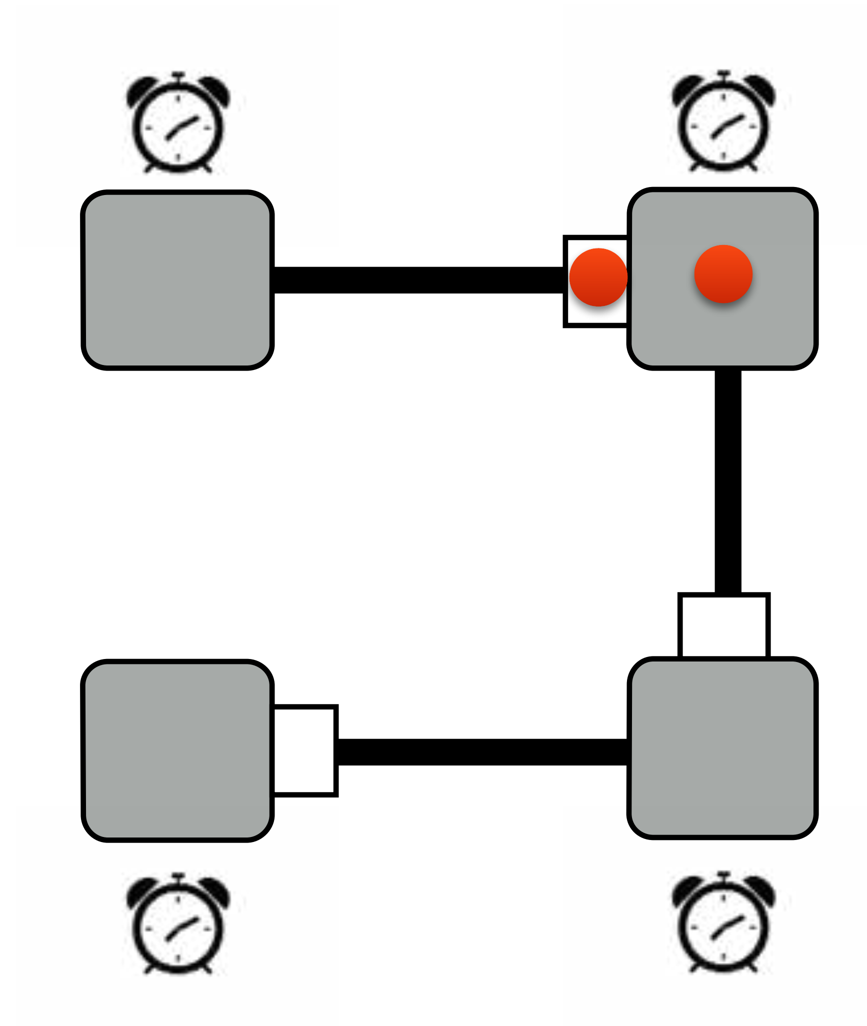
$$0 < T_{\min} \leq T^n \leq T_{\max} \quad \text{or}$$
$$T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon$$

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

- Buffered communication without message inversion or loss

- Bounded communication delay

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



Quasi-Periodic Architecture

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

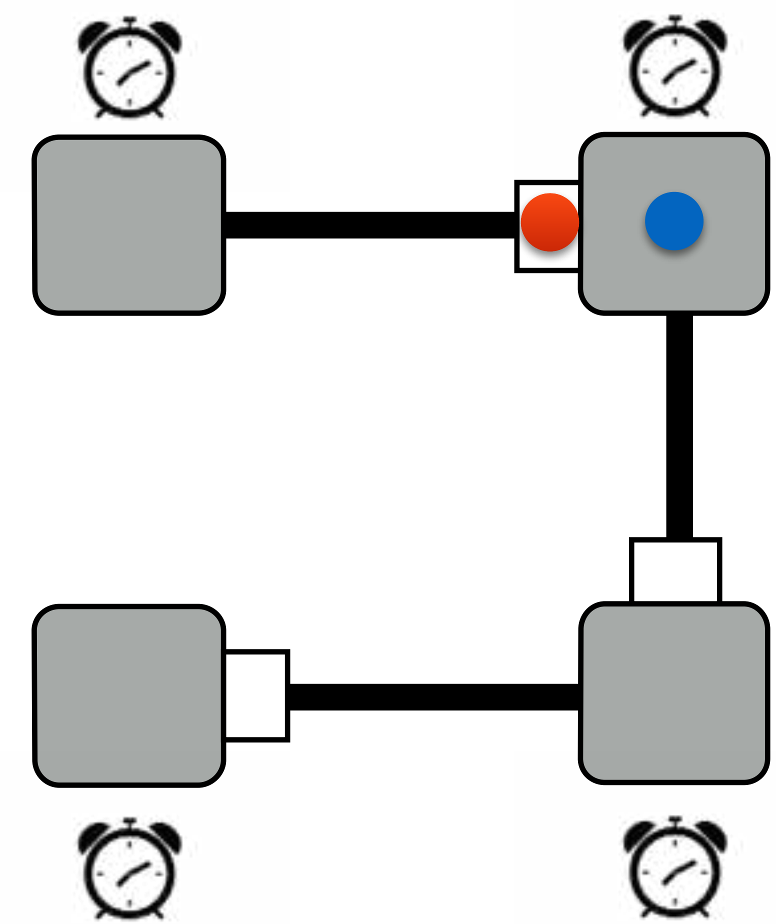
$$0 < T_{\min} \leq T^n \leq T_{\max} \quad \text{or}$$
$$T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon$$

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

- Buffered communication without message inversion or loss

- Bounded communication delay

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



Quasi-Periodic Architecture

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

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

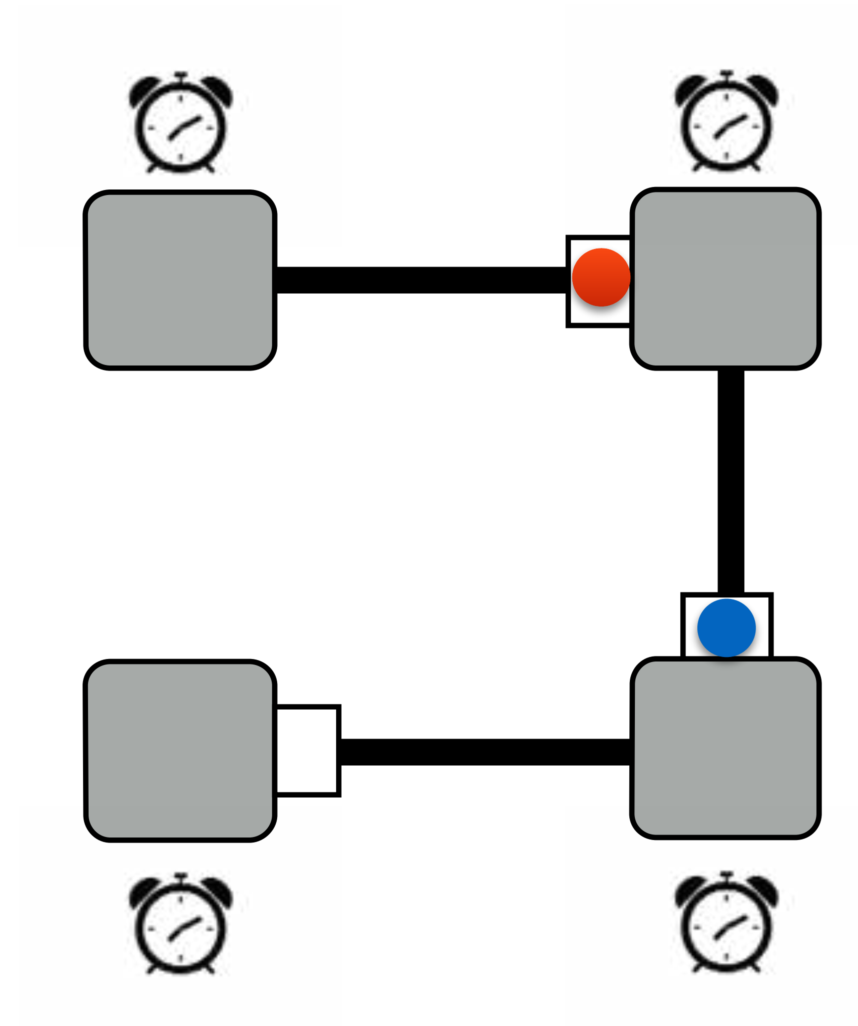
  $T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon$

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

- Buffered communication without message inversion or loss

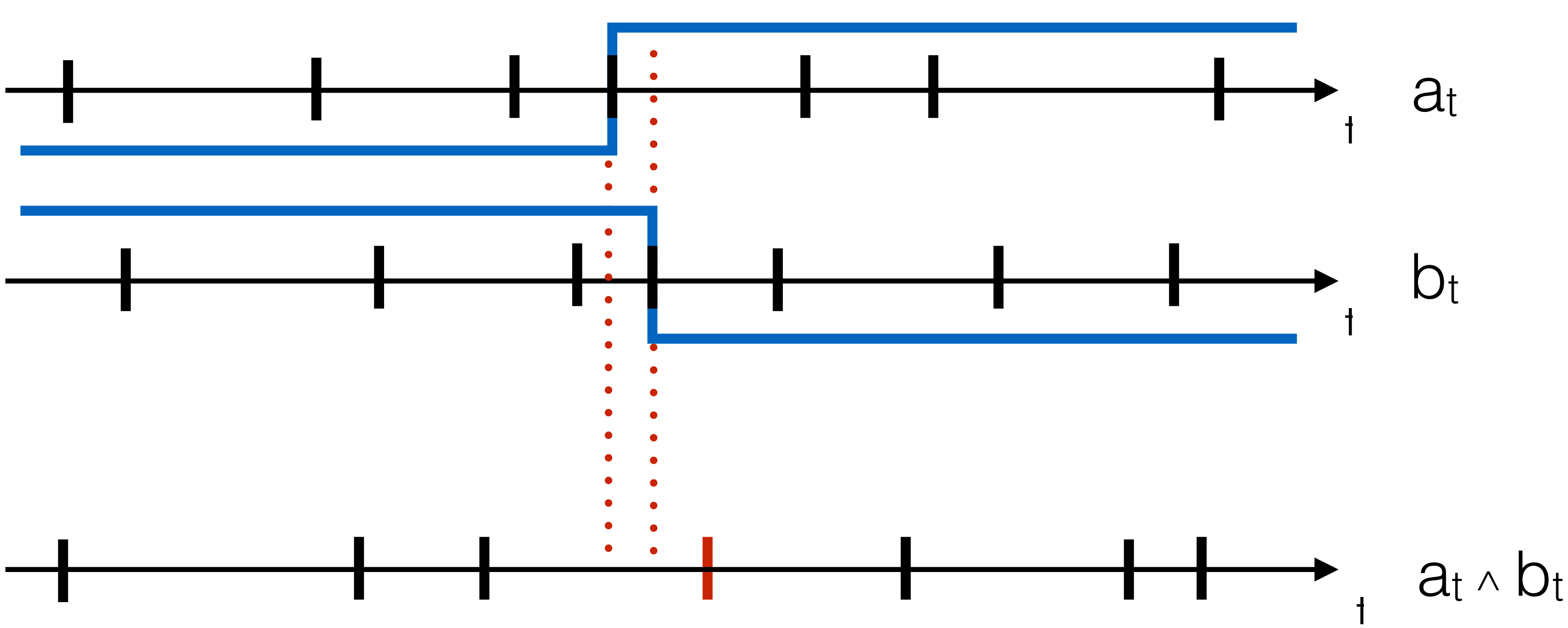
- Bounded communication delay

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



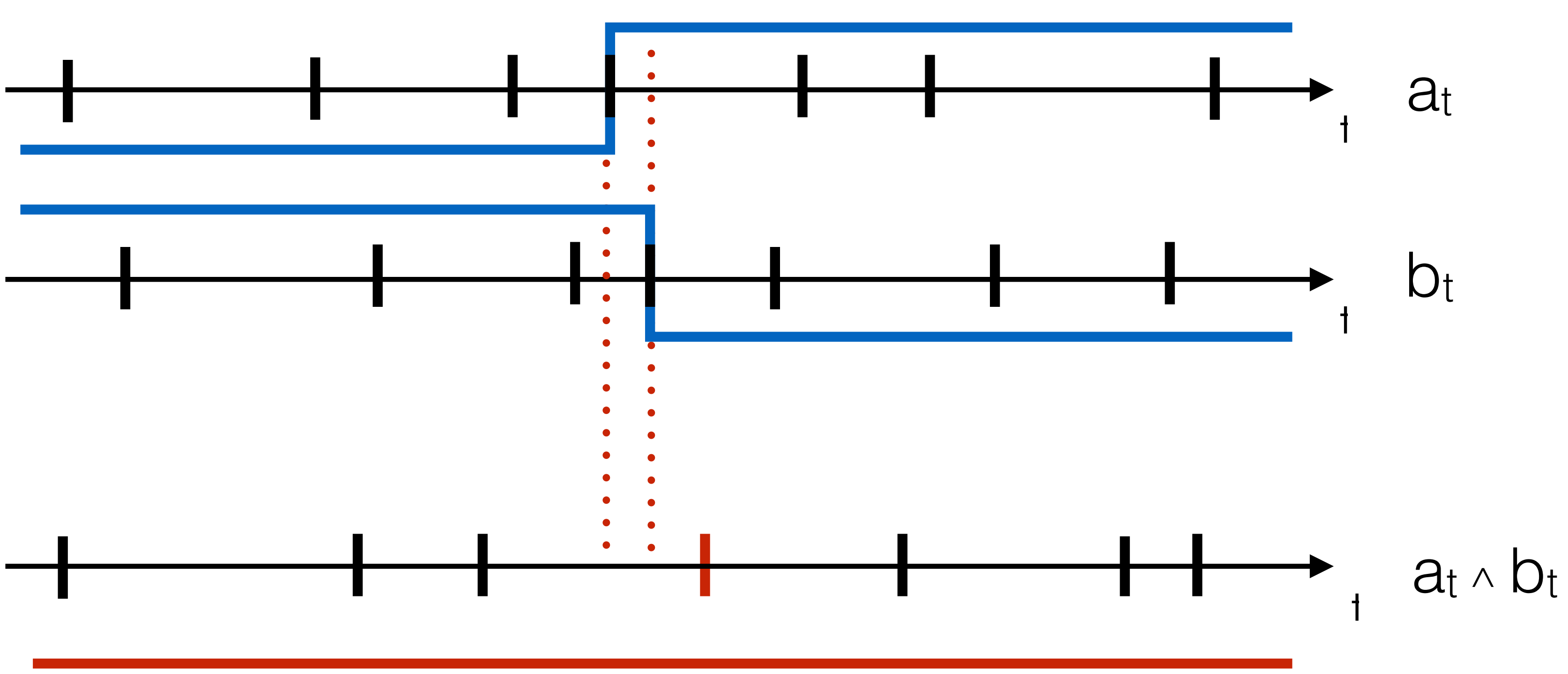
Problems

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



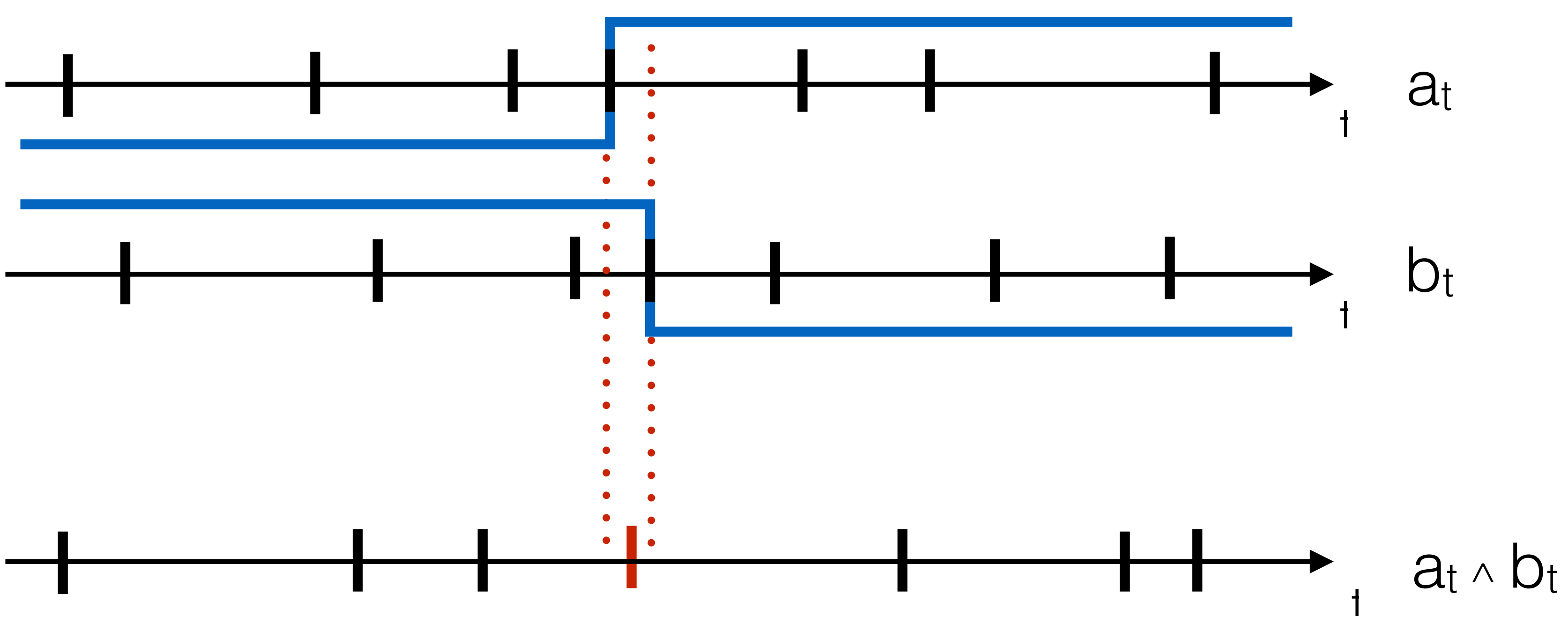
Problems

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



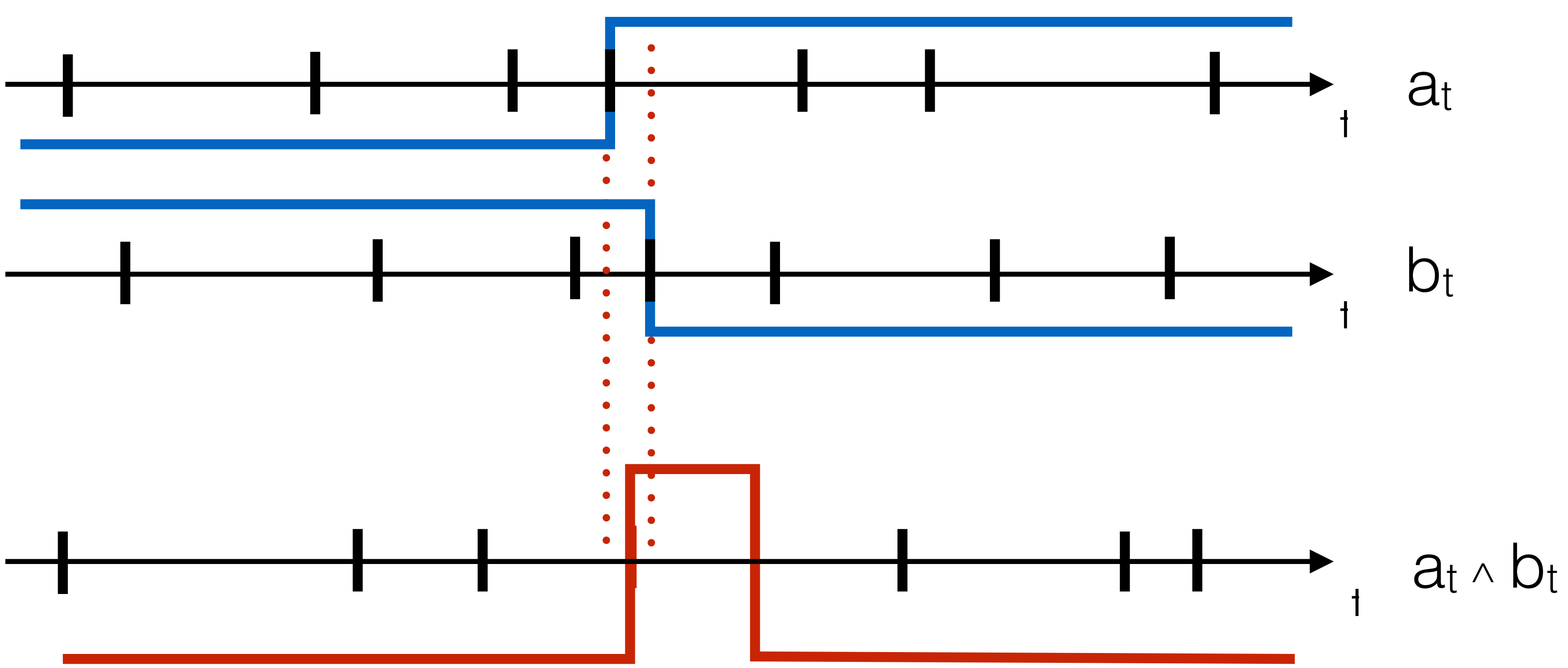
Problems

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



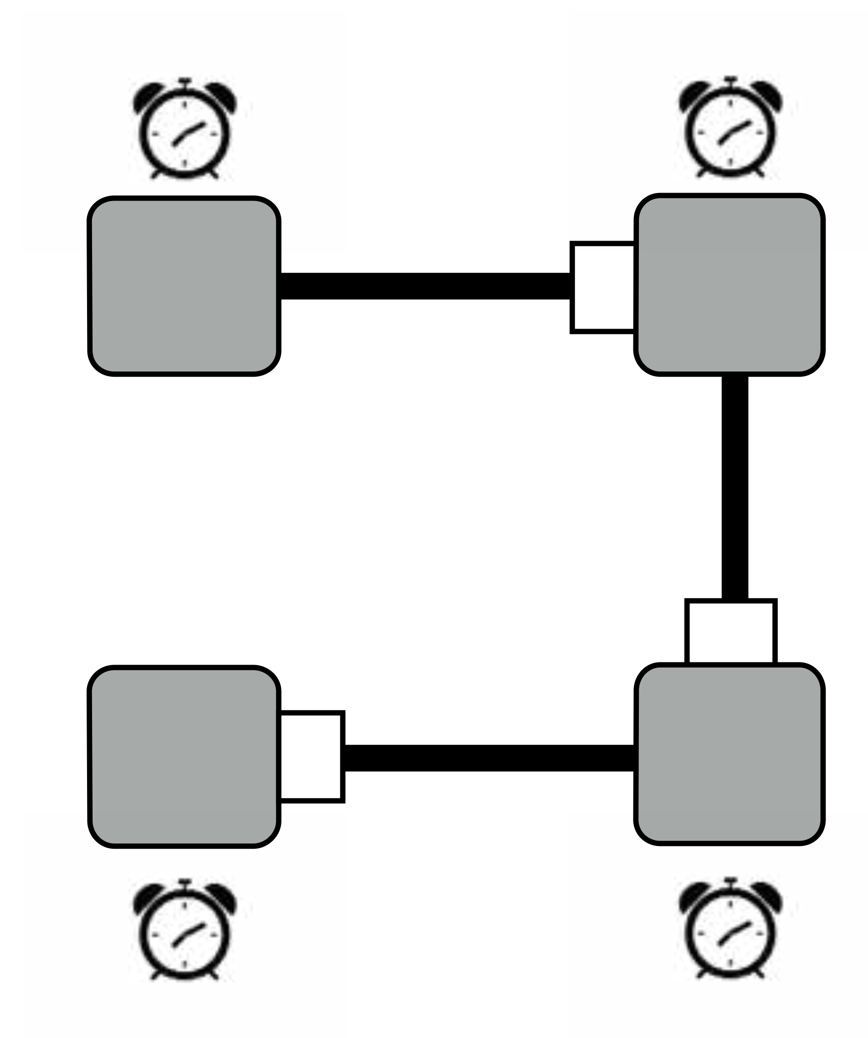
Problems

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



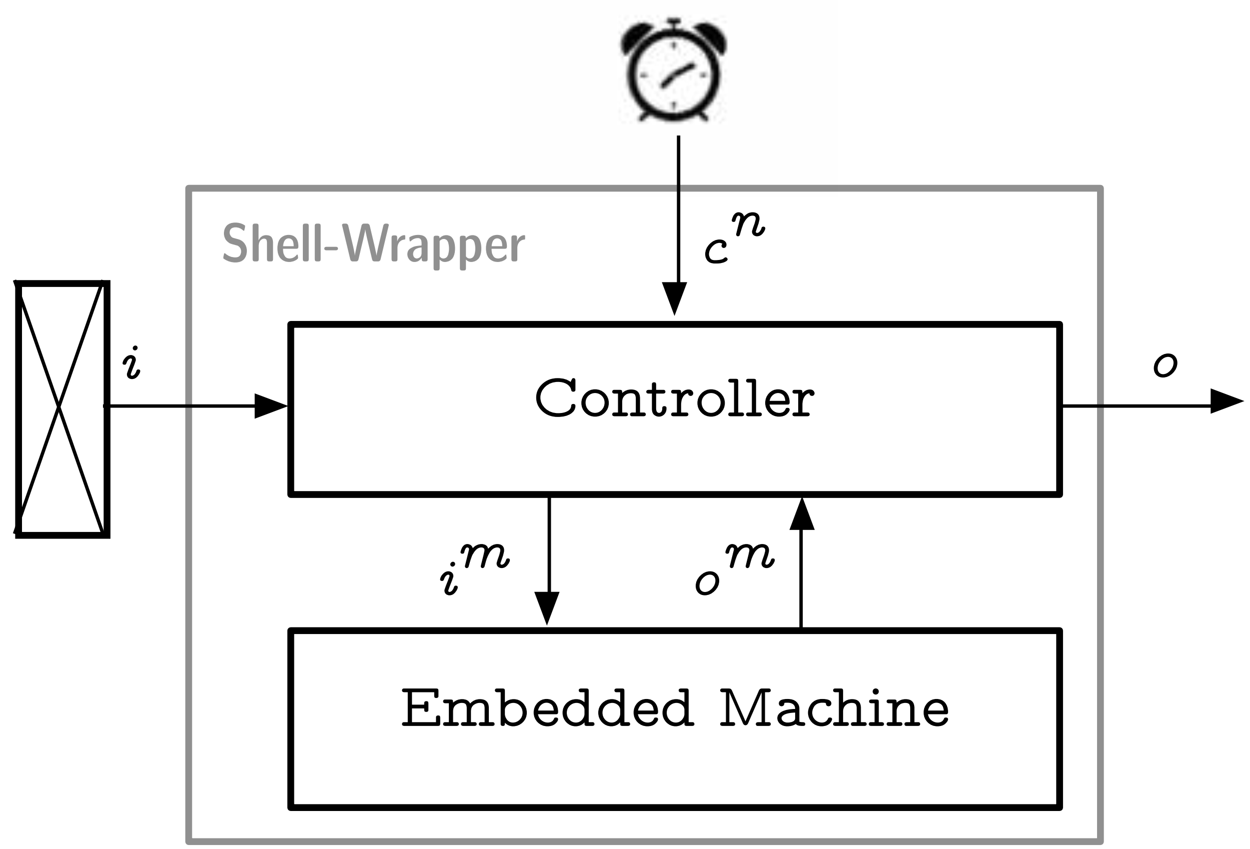
What are LTTAs?

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



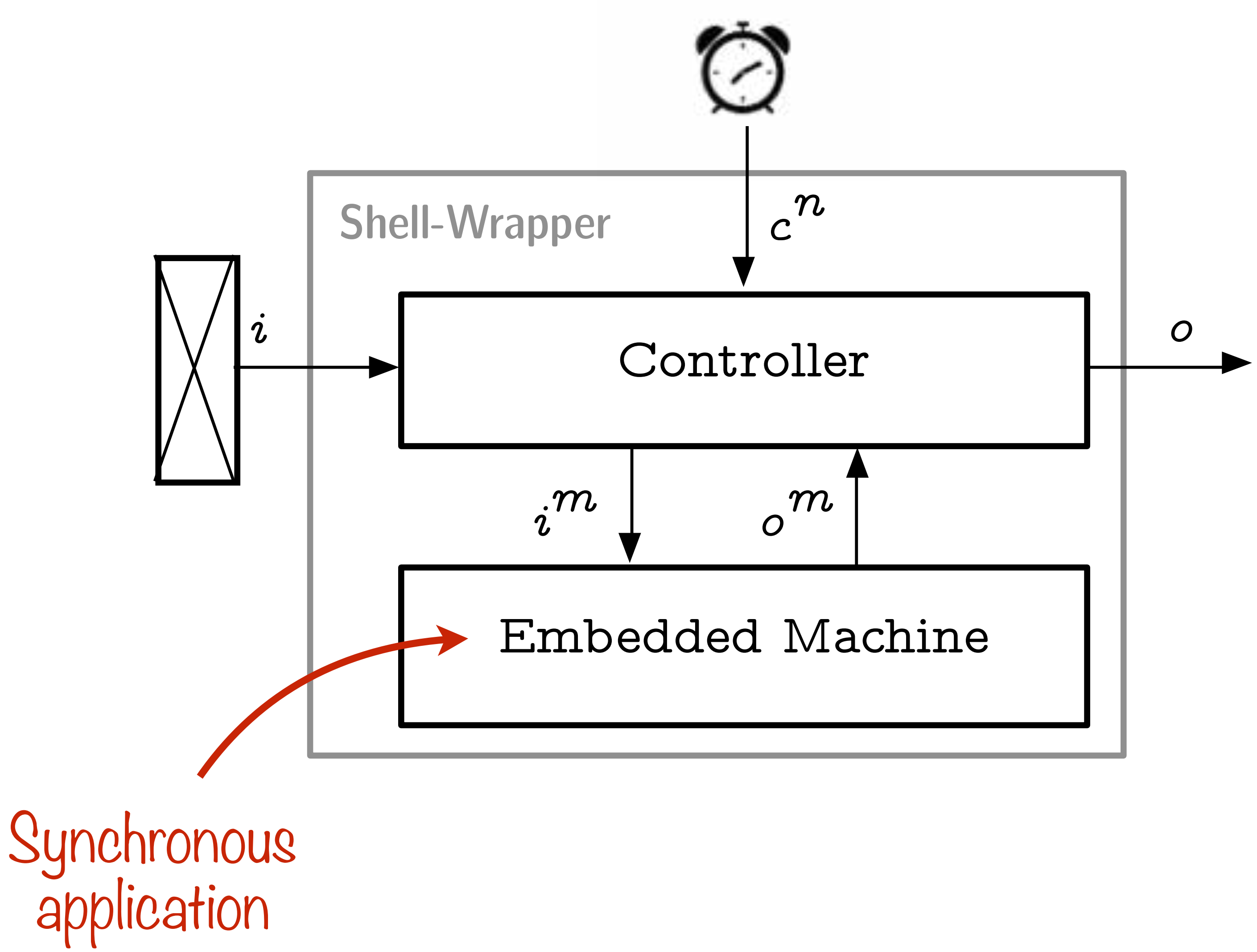
Synchronous Model

Modeling the nodes



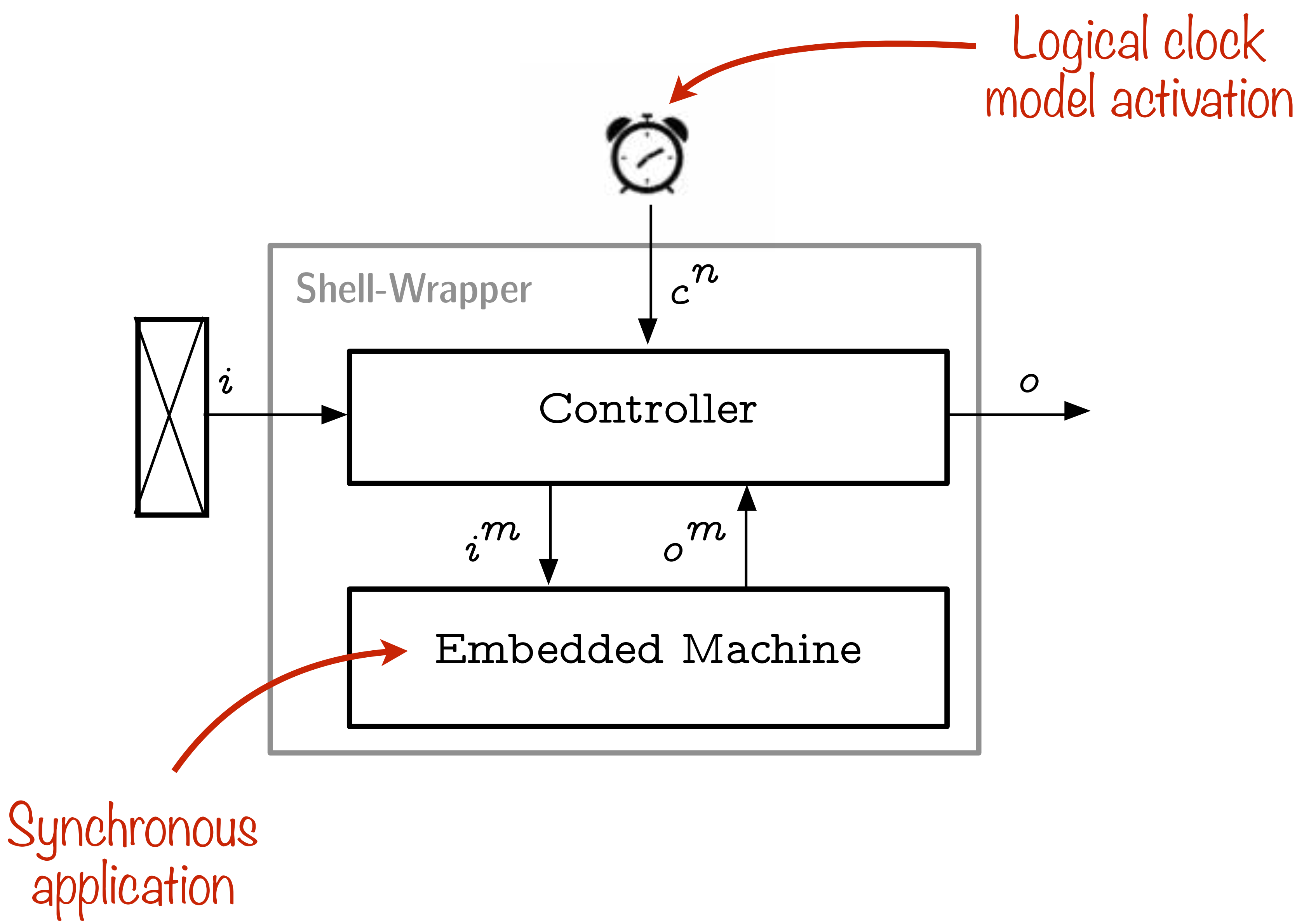
Synchronous Model

Modeling the nodes



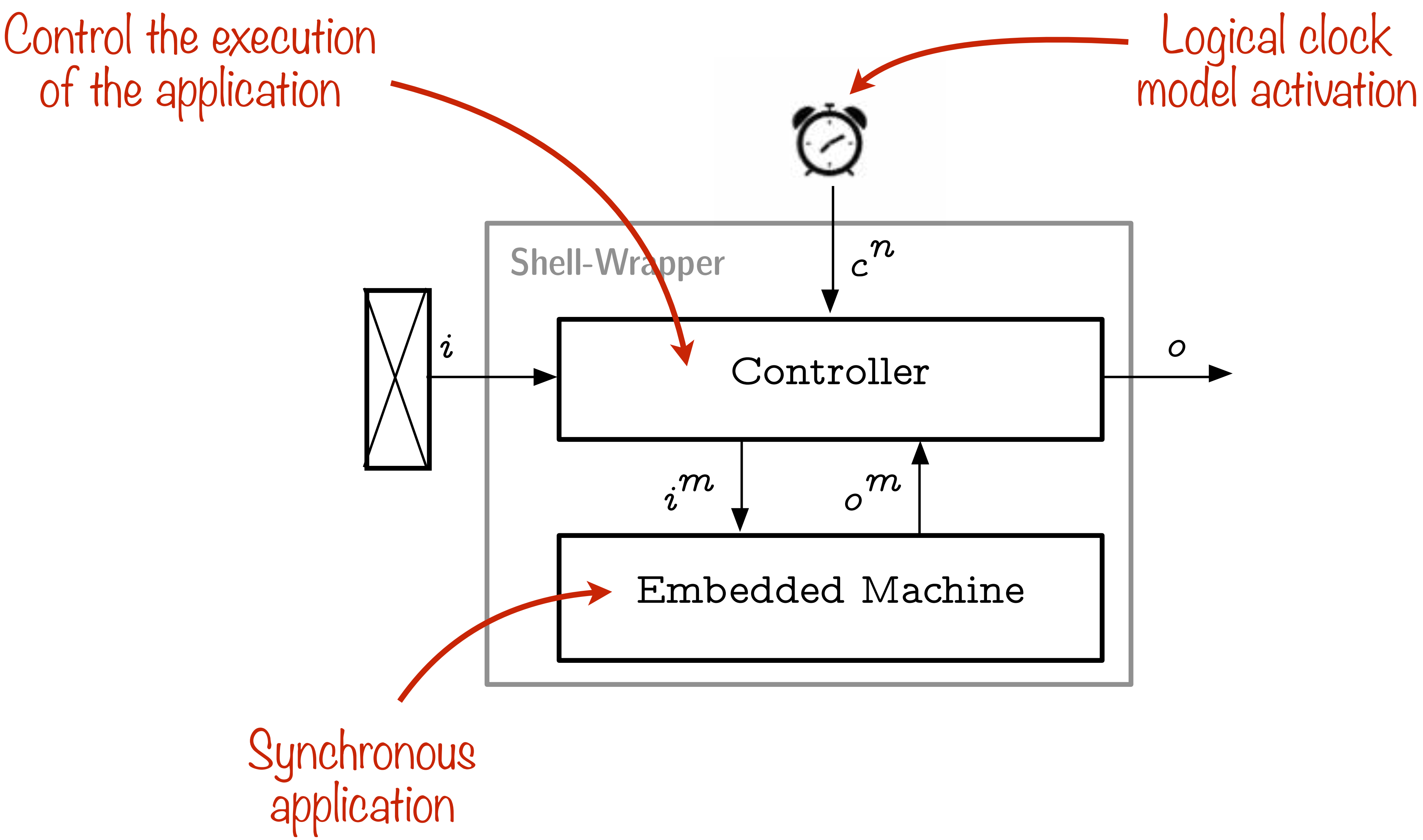
Synchronous Model

Modeling the nodes



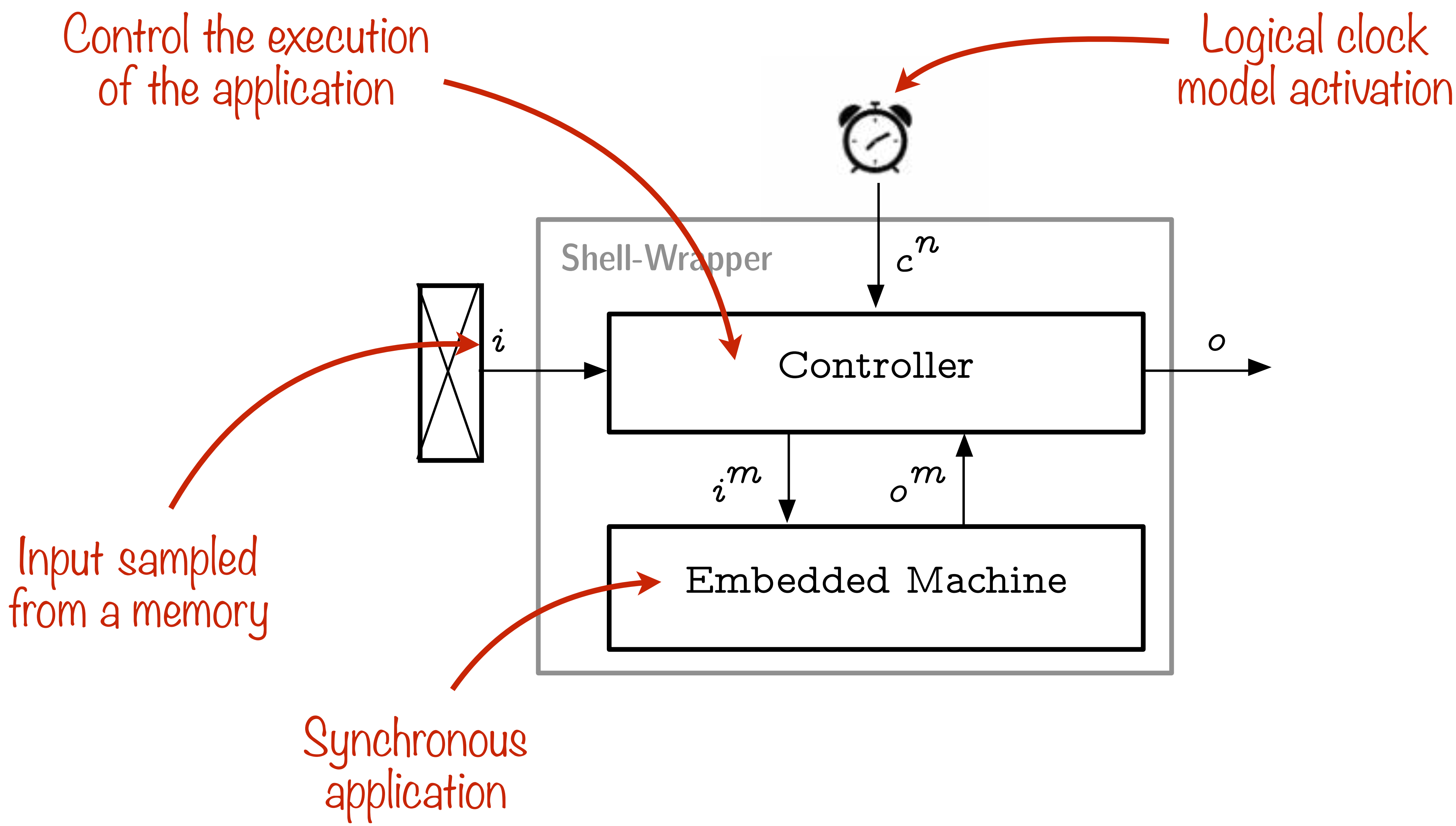
Synchronous Model

Modeling the nodes



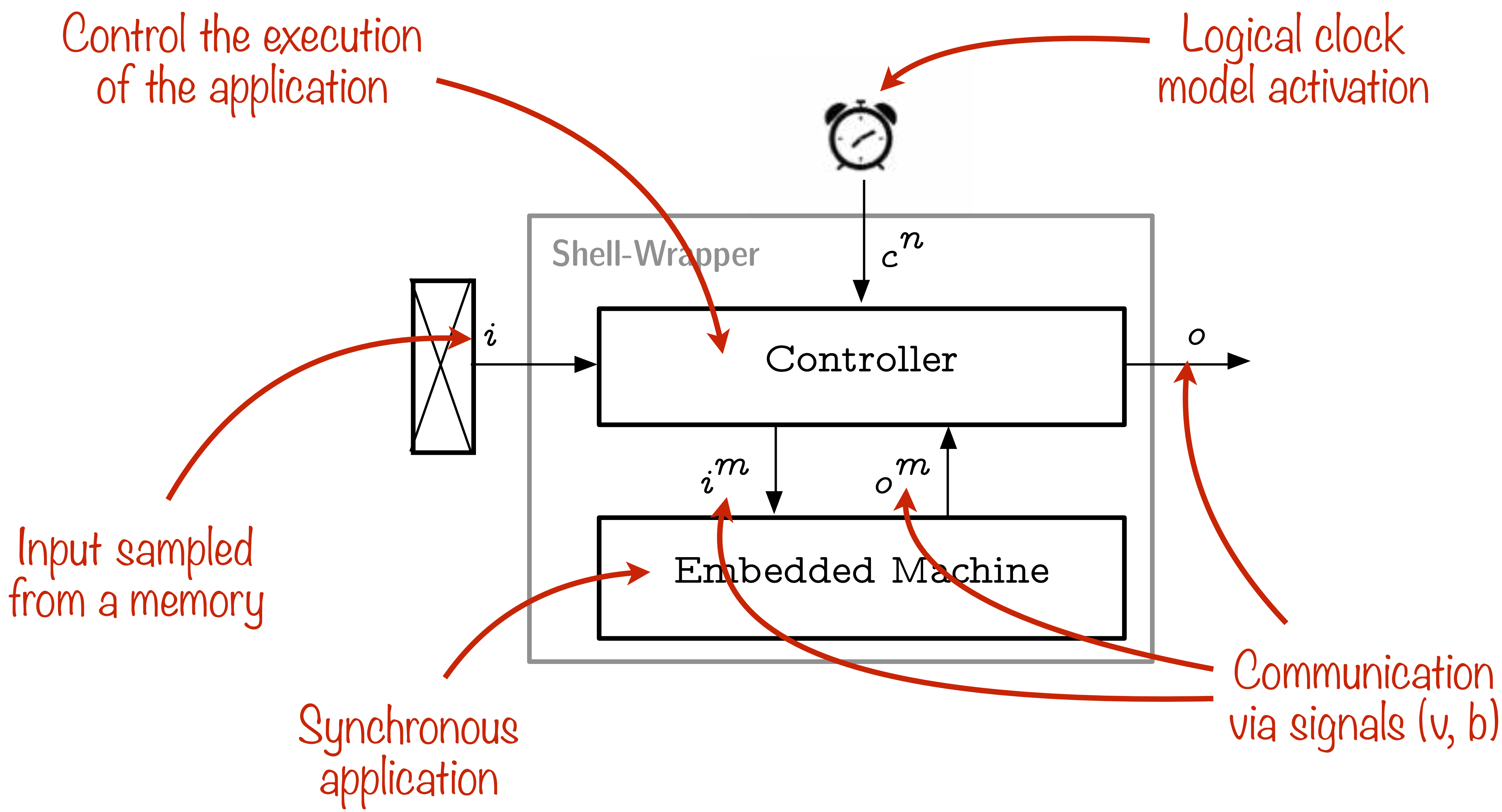
Synchronous Model

Modeling the nodes



Synchronous Model

Modeling the nodes



Synchronous Model

Modeling the nodes

Control the execution
of the application

Logical clock
model activation

```
let node ltta_node (c, i) = o where
  rec (o, im) = controller (c, i, om)
  and present im(v) -> do emit om = em v done
```

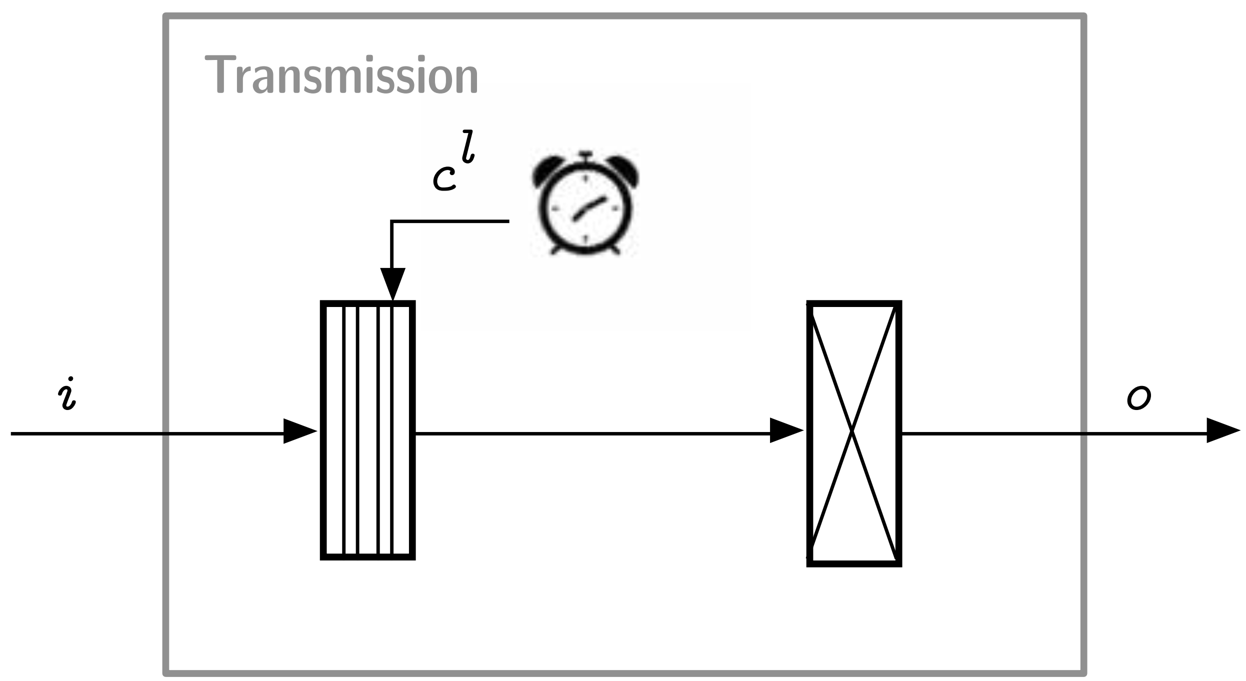
Input sampled
from a memory

Synchronous
application

Communication
via signals (v, b)

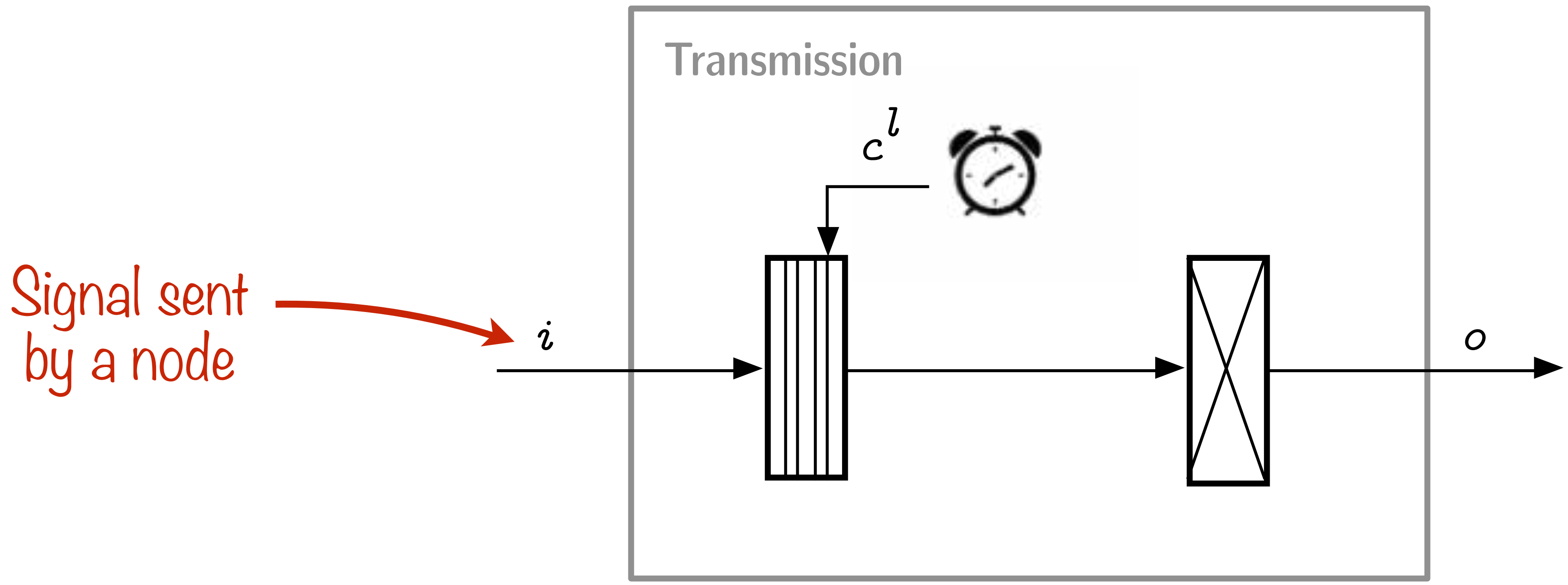
Synchronous Model

Modeling the links



Synchronous Model

Modeling the links

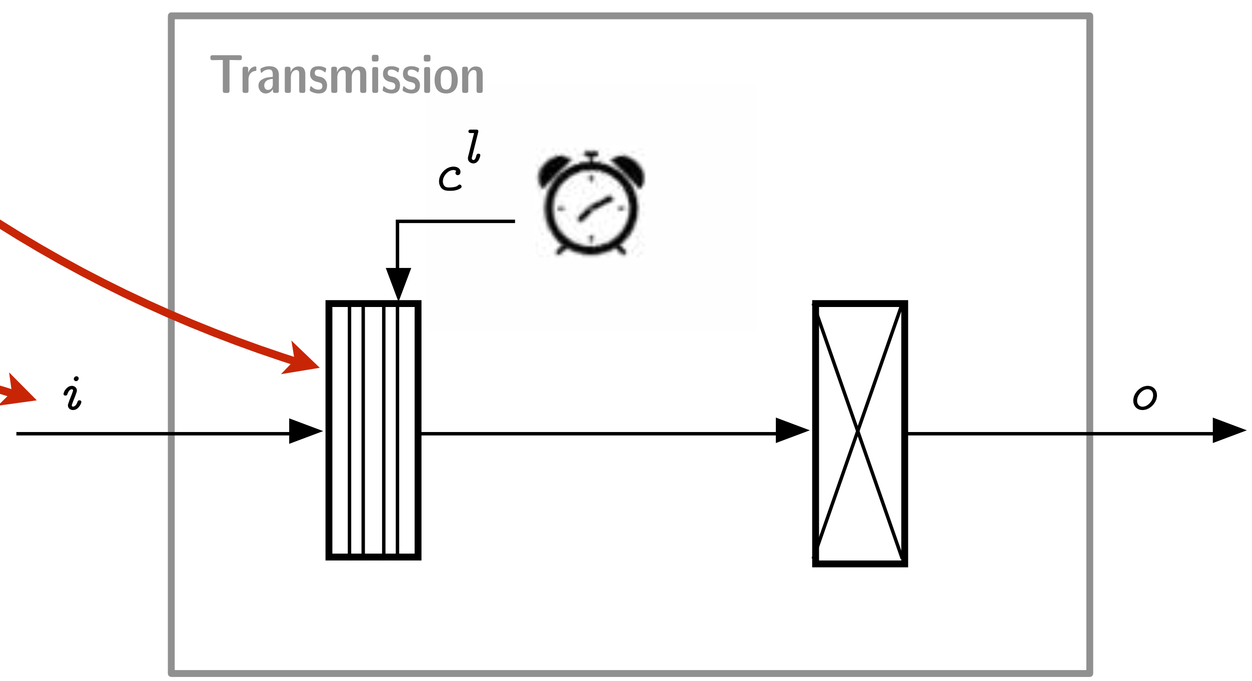


Synchronous Model

Modeling the links

Channel:
delay a signal

Signal sent
by a node



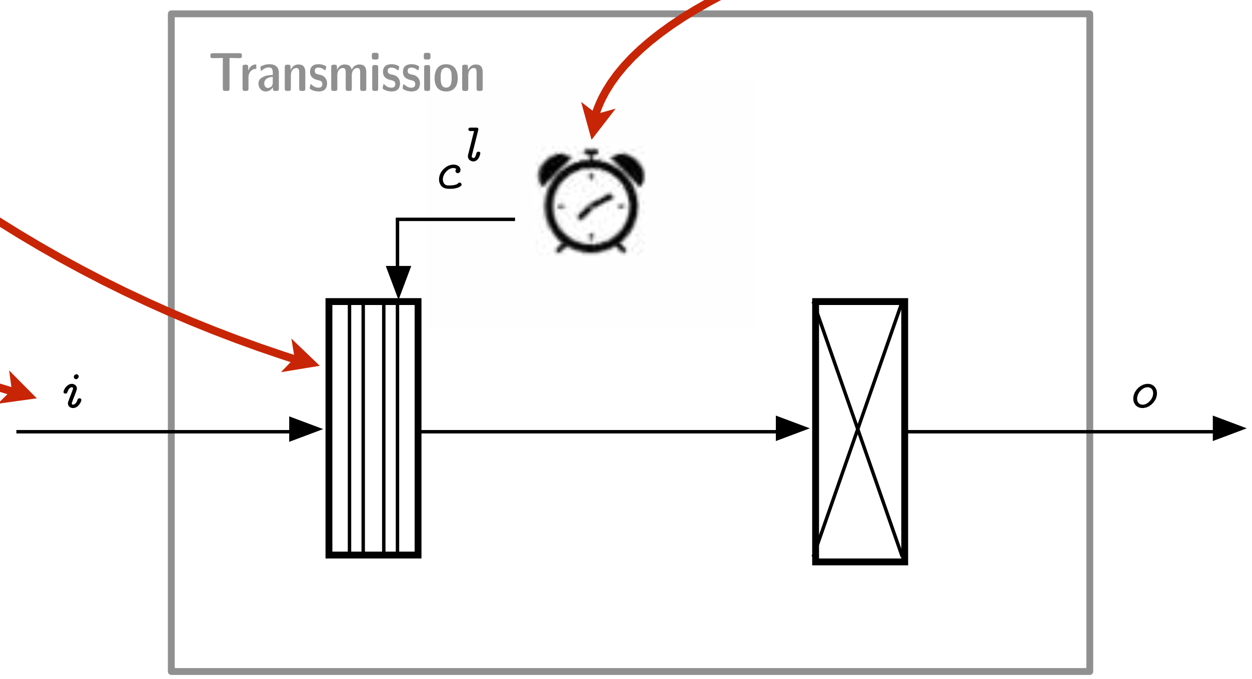
Synchronous Model

Modeling the links

Channel:
delay a signal

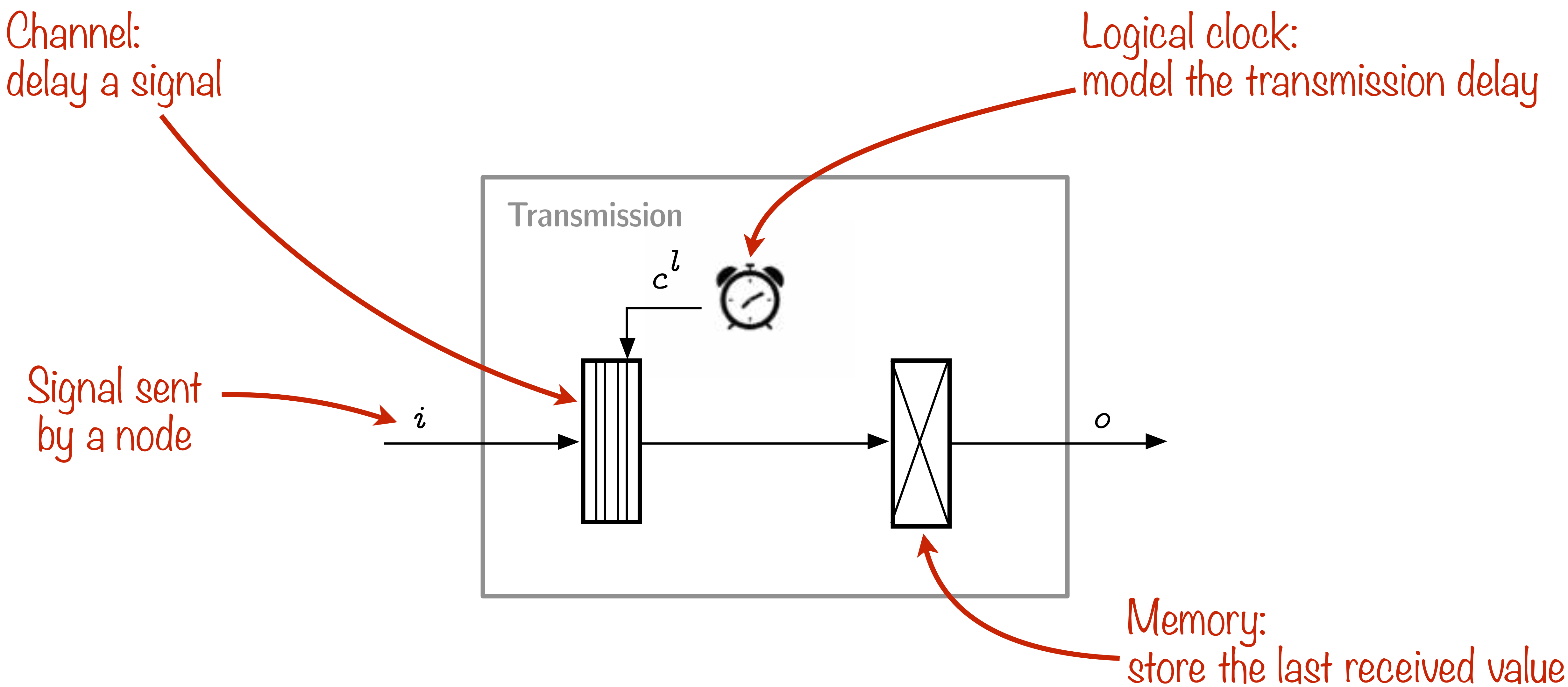
Logical clock:
model the transmission delay

Signal sent
by a node



Synchronous Model

Modeling the links



Synchronous Model

Modeling the links

Channel:
delay a signal

Logical clock:
model the transmission delay

```
let node channel (cl, i) = o where
  rec init mem = empty
  and present i(v) -> do mem = enqueue (last mem, v) done
              | cl   -> do emit o = front (last mem)
                        and mem = dequeue (last mem) done
```

Signal sent
by a node

```
let node mem (i, default) = m where
  rec init m = default
  and present i(v) -> do m = v done
```

Memory:
store the last received value

Synchronous Model

Freshness of values

- **Problem:** Determine if a new value has arrived
- **Idea:** Add an *alternating bit protocol* to the channel

Synchronous Model

Freshness of values

- **Problem:** Determine if a new value has arrived
- **Idea:** Add an *alternating bit protocol* to the channel

Input sampled from a memory

true if the value is fresh

```
let node fresh i = o where
  rec init s = false
  and o = xor (last s, i.alt)
  and present o -> do s = i.alt done
```

Synchronous Model

From discrete to physical time

Timing function: $T : \mathcal{C} \to \mathbb{N} \to \mathbb{R}$
associate a time-tag to the kth activation
of a logical clock

Node

$$\forall i \in \mathbb{N} \quad T_{\min} \leq T(c^n)(i+1) - T(c^n)(i) \leq T_{\max}$$

Link

$$\forall i \in \mathbb{N} \quad T(c^l)(i) = T(c^s)(i) + \tau_i$$
$$\text{with} \quad \tau_{\min} \leq \tau_i \leq \tau_{\max}$$

Synchronous Model

From discrete to physical time

Node

```
let hybrid metro (t_min, t_max) = c where
  rec der t = 1.0 init -. Misc.rand_val (t_min, t_max)
      reset c() -> -. Misc.rand_val (t_min, t_max)
  and present up(last t) -> do emit c = () done
```

Link

```
let hybrid delay (c, tau_min, tau_max) = s where
  rec der t = 1.0 init 0.0
      reset c() -> -. Misc.rand_val (tau_min, tau_max)
  and present up(last t) -> do emit s = () done
```

Synchronous Model

From discrete to physical time

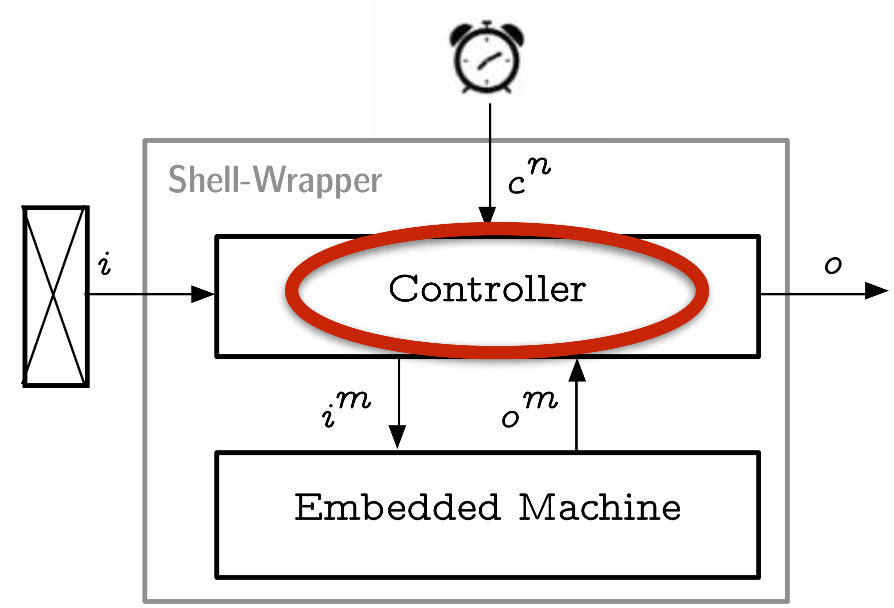
- **Other approaches:** Discrete abstractions of the characteristics of the architecture, e.g., quasi-synchrony
- **Problem:** Does not model the transmission delay (modeled as one tick of the base clock)
State explosion
- **ODE:** Easy simulation, directly relates to the architecture description
But no verification…

What's next?

Design controllers that ensure a synchronous execution of embedded machines

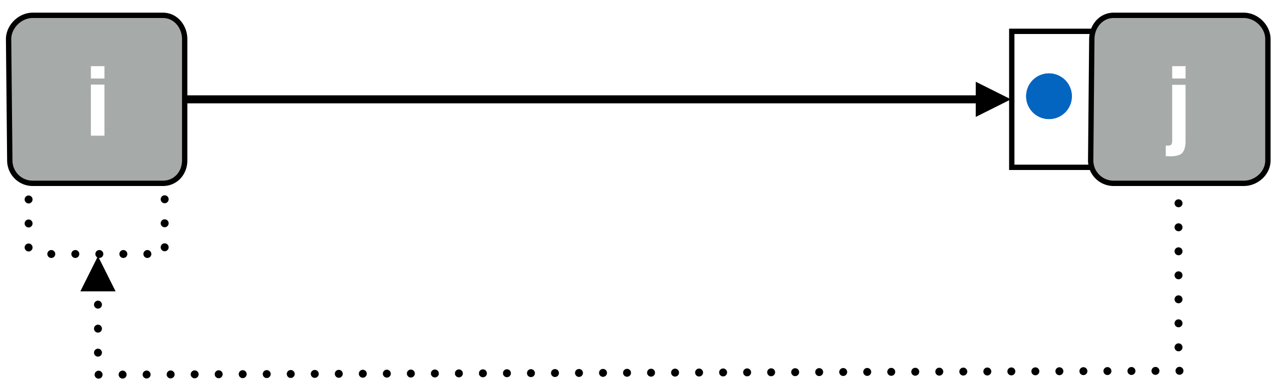
- **Back-Pressure LTTA** [Tripakis et al. 2008]

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



Back-Pressure Kahn Network

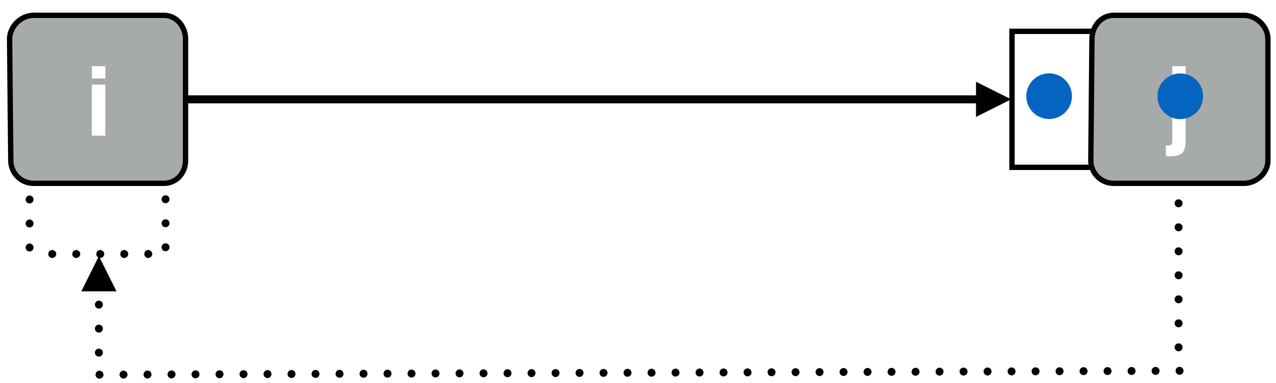
Buffer of size 1



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

Back-Pressure Kahn Network

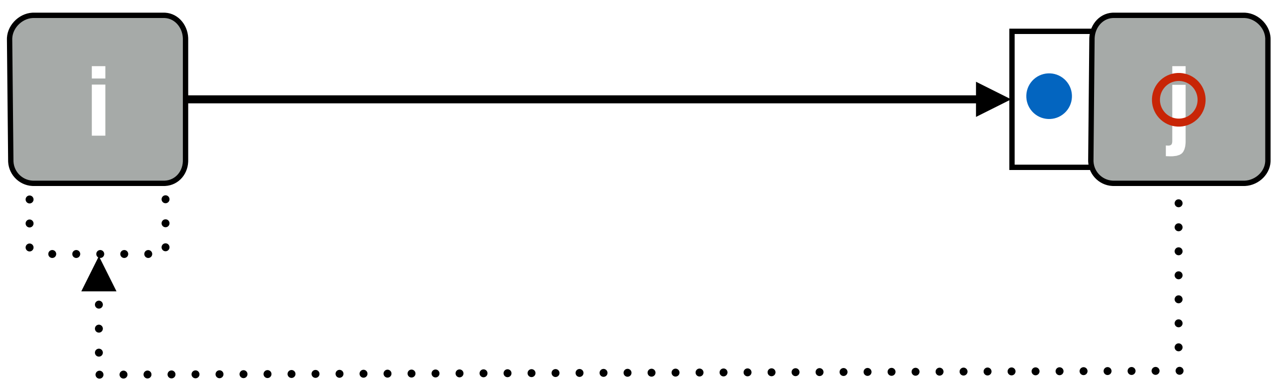
Buffer of size 1



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

Back-Pressure Kahn Network

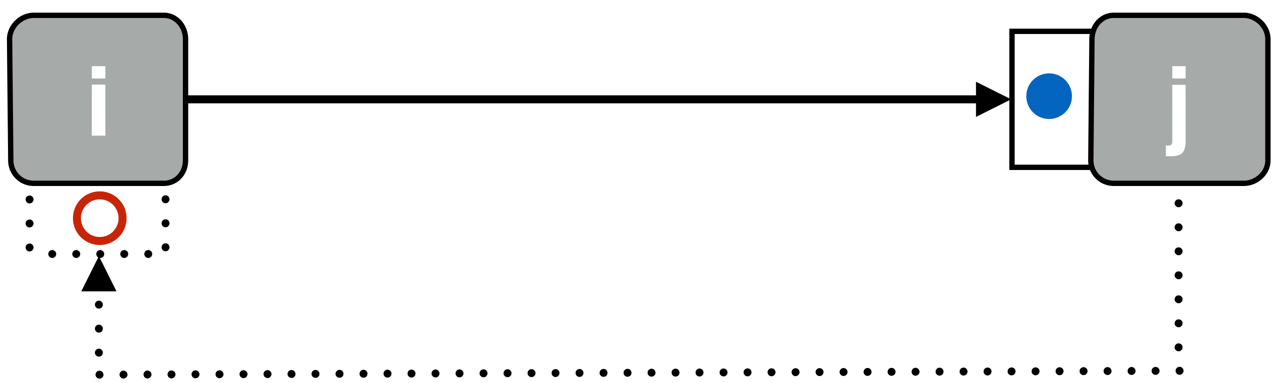
Buffer of size 1



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

Back-Pressure Kahn Network

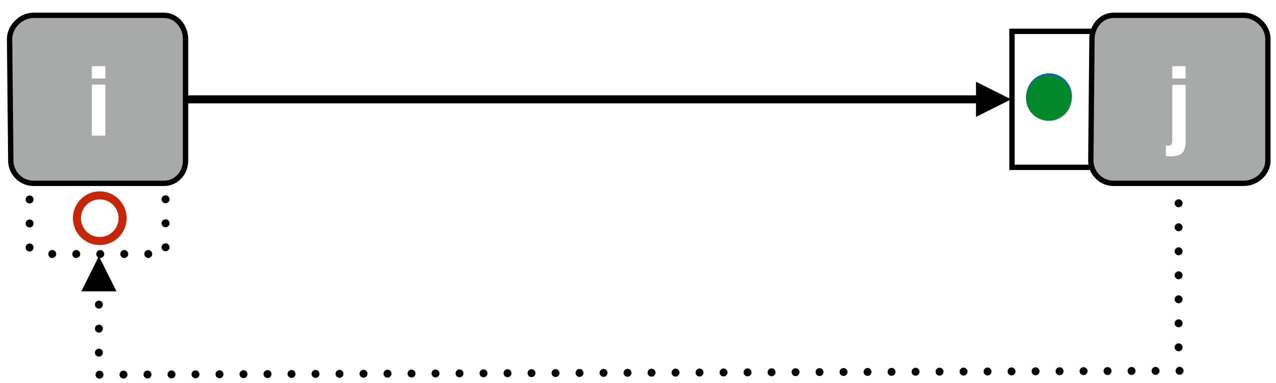
Buffer of size 1



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

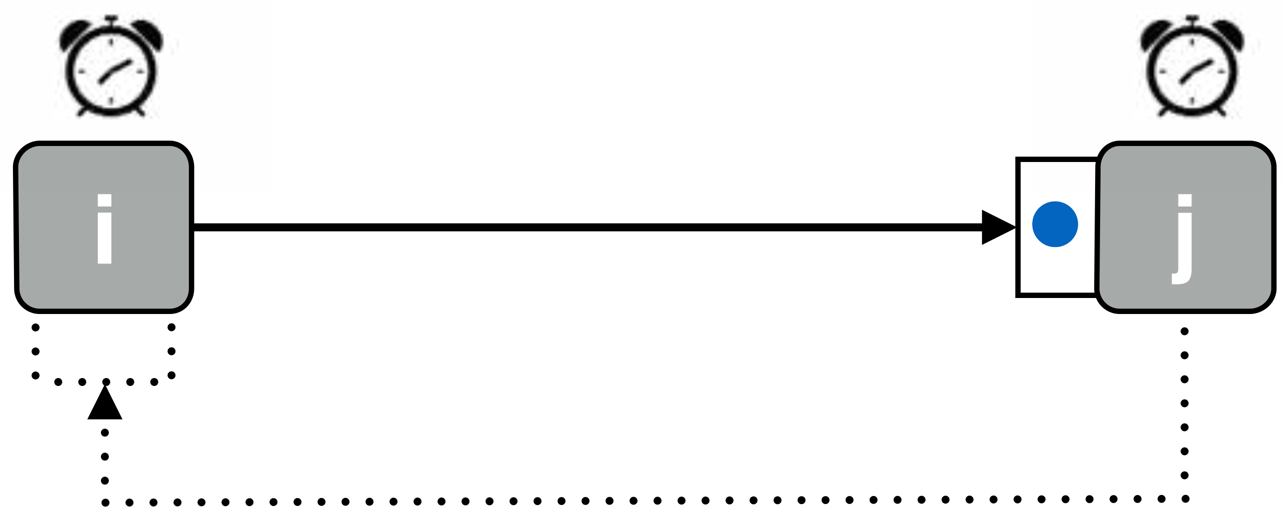
Back-Pressure Kahn Network

Buffer of size 1



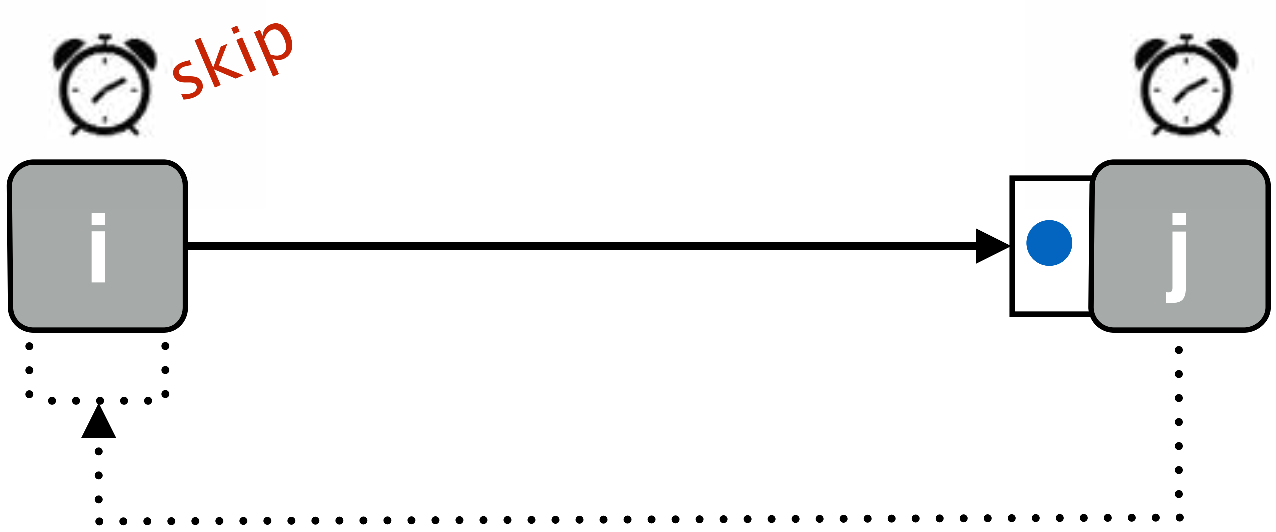
- Reading from a buffer is acknowledged to the writer
- Nodes alternate between **exec** and **write**

Back-Pressure LTTA



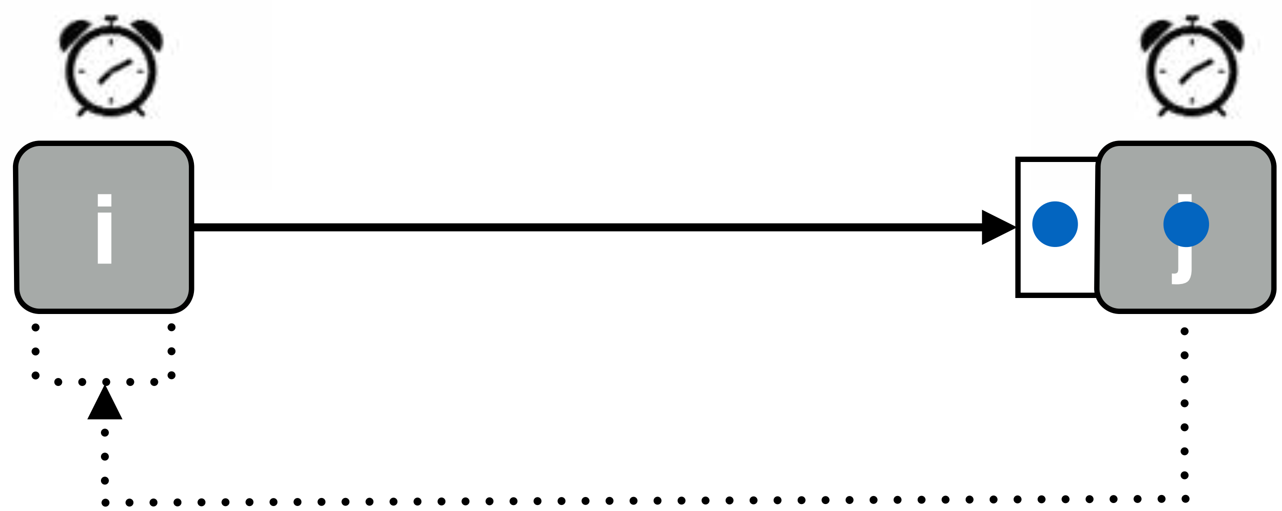
- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism

Back-Pressure LTTA



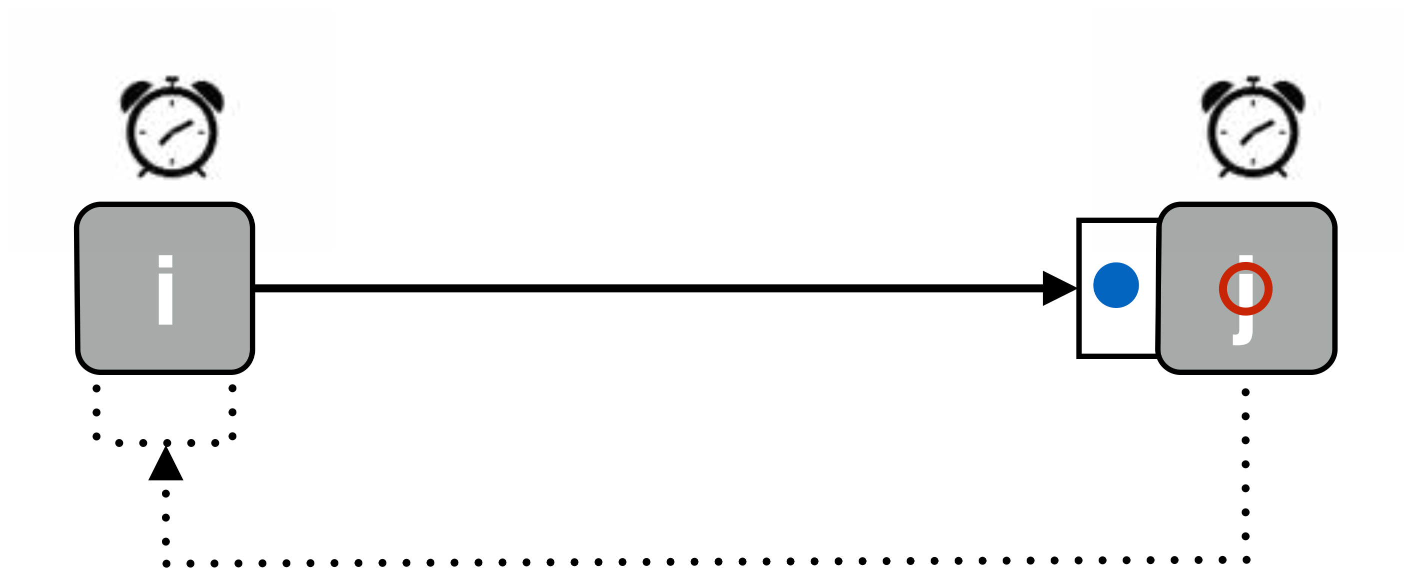
- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism

Back-Pressure LTTA



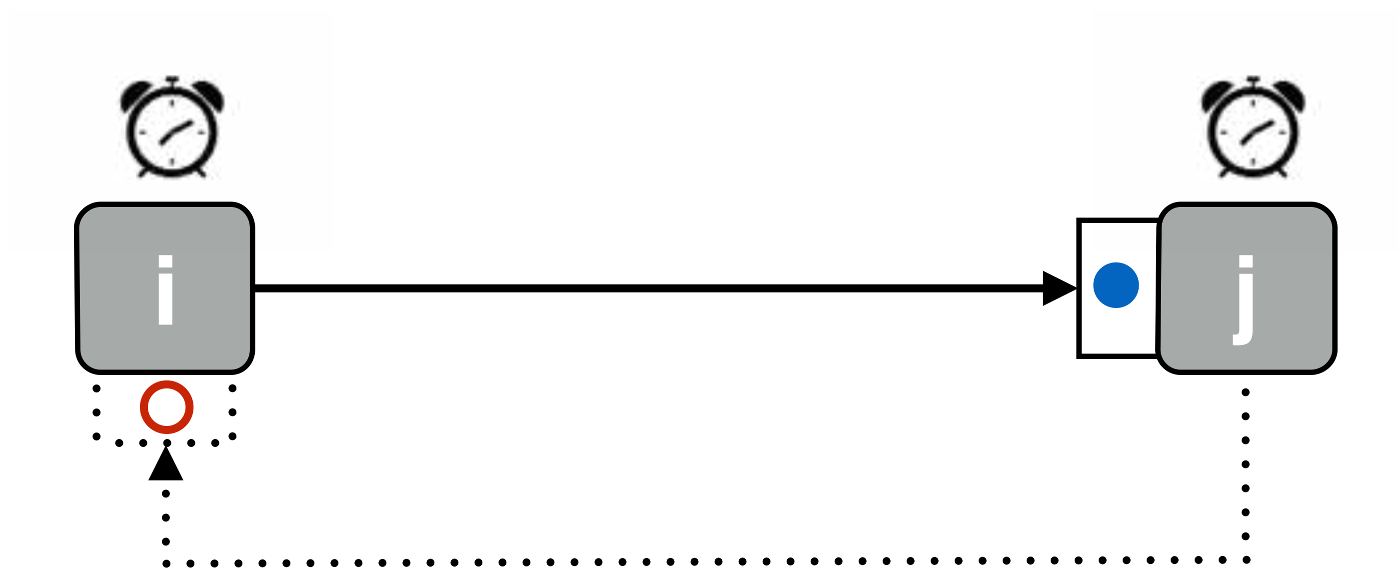
- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism

Back-Pressure LTTA



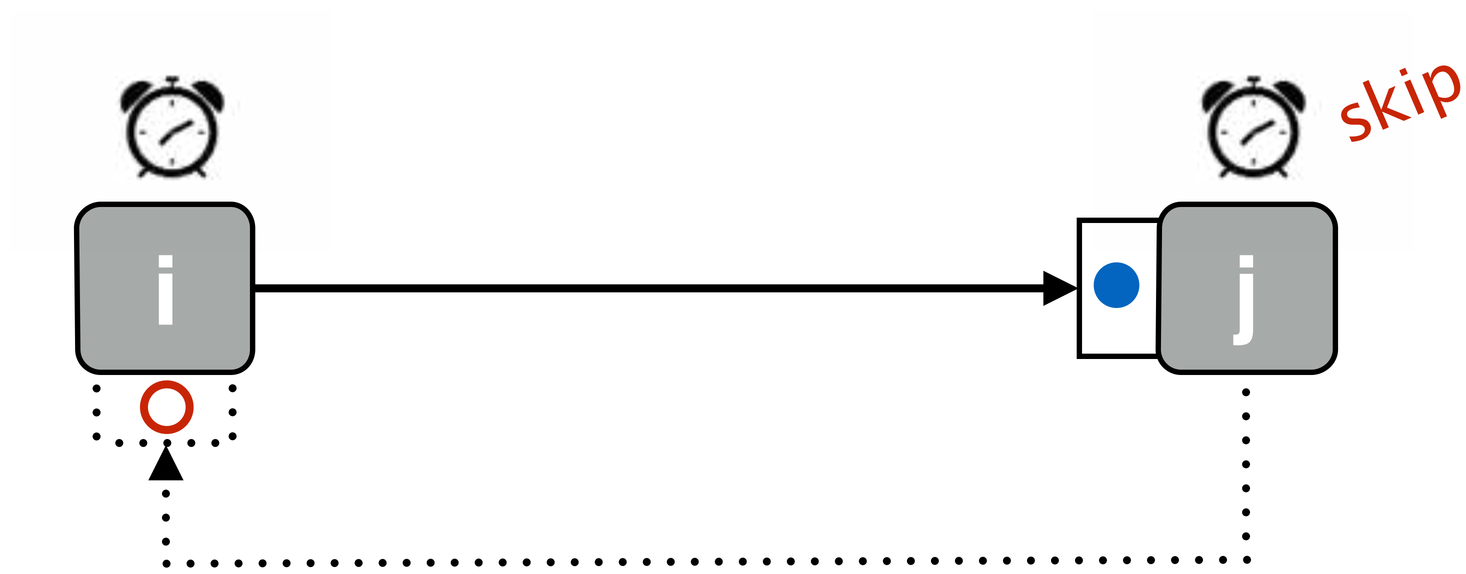
- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism

Back-Pressure LTTA



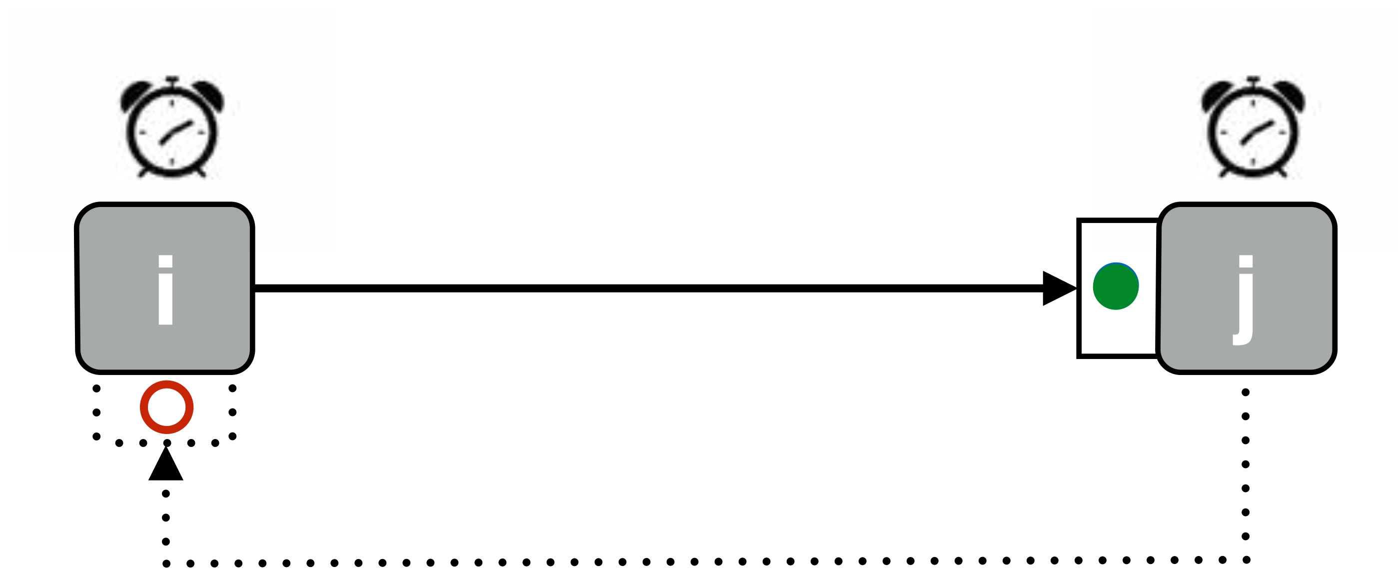
- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism

Back-Pressure LTTA



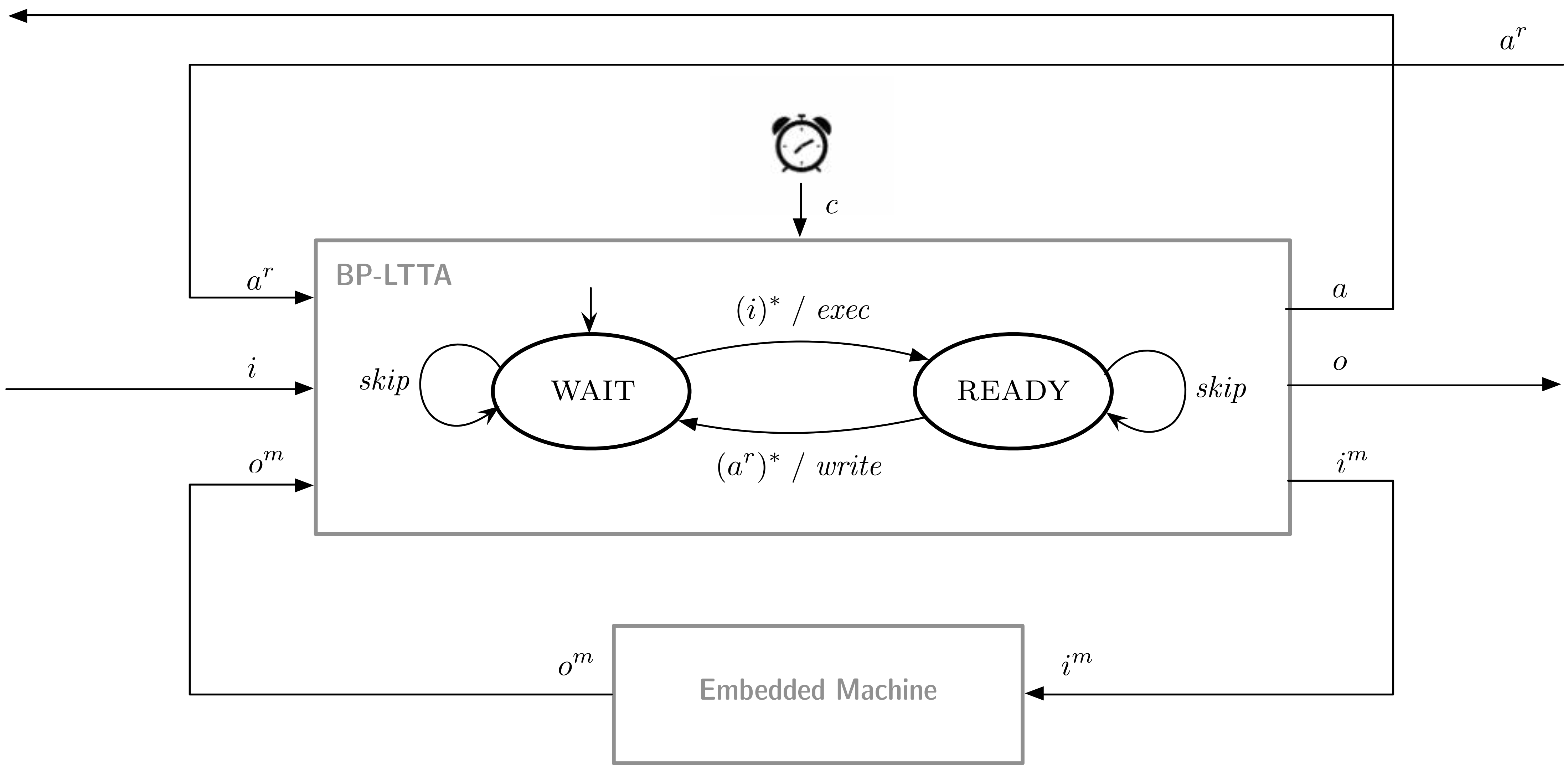
- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism

Back-Pressure LTTA



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

Back-Pressure LTTA



Back-Pressure LTTA

```
let node bp_controller (c, i, ar, om, default) = (a, o, im) where
  rec fi = fresh i
  and far = fresh ar
  and m = mem (om, default)
  and init state = Wait
  and match c with
      | false -> do done
      | true ->
         do match last state, fi, far with
(* exec *)    | Wait, true, _ ->
                do state = Ready
                and emit im = i.data
                and emit a = true done
(* write *)   | Ready, _, true ->
                do state = Wait
                and emit o = m done
(* skip *)    | _ -> do done
         done
```

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:
  $$1/\lambda_{\mathrm{BP}} = 2(T_{\max} + \tau_{\max})$$

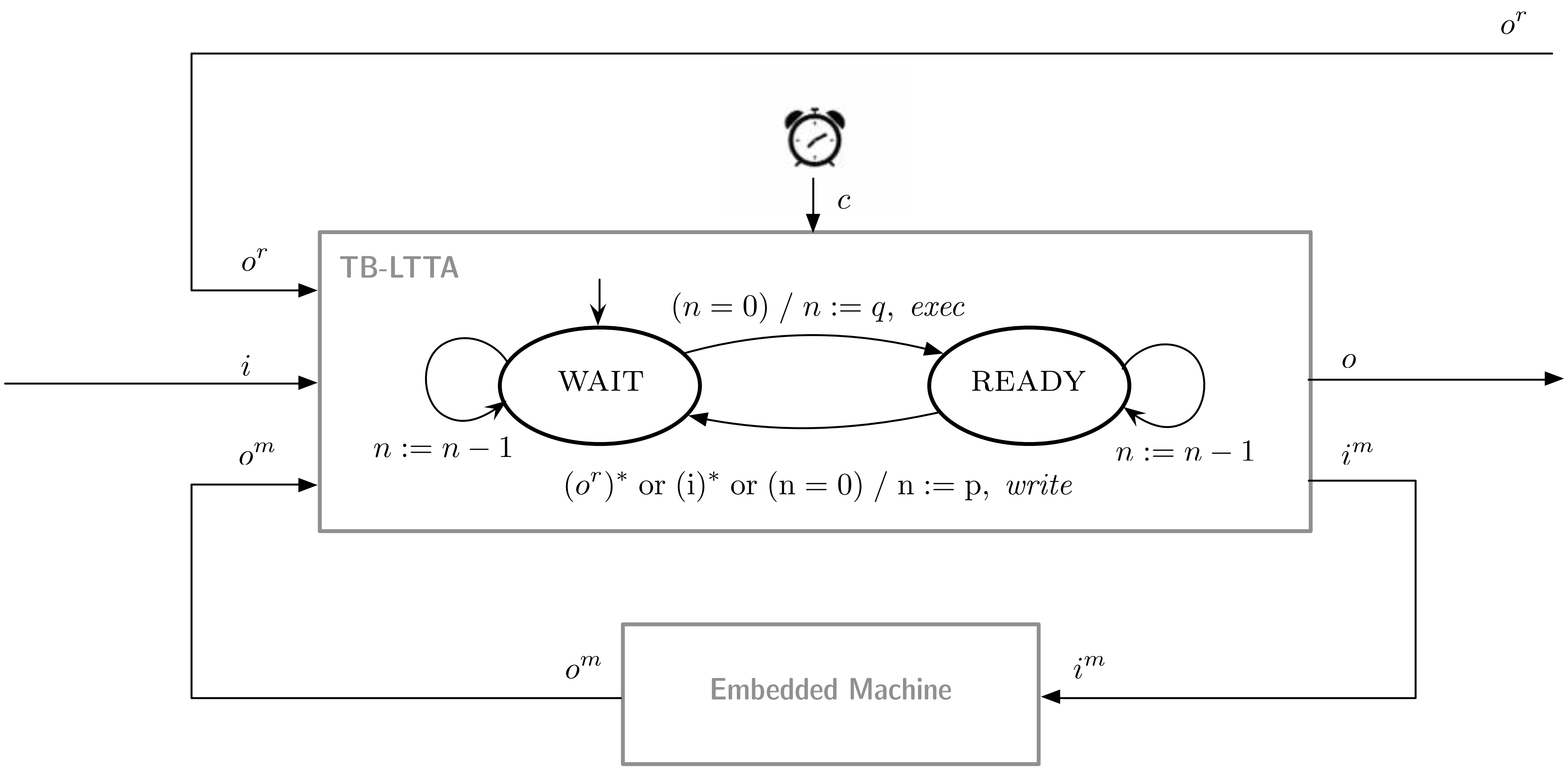
Time-Based LTTA

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

Time-Based LTTA

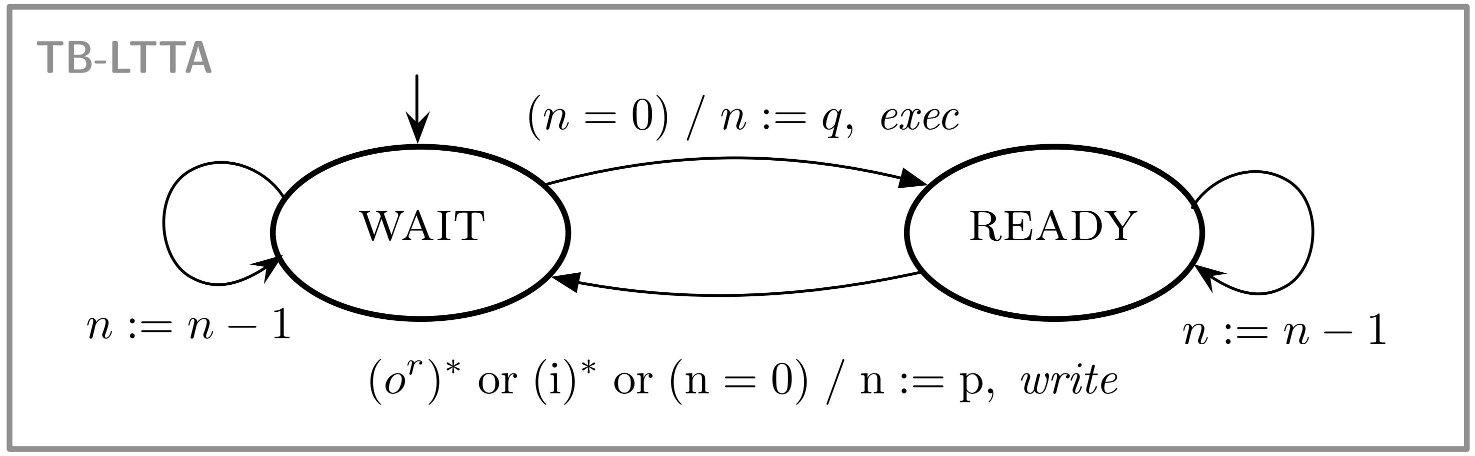
- Nodes alternate between **exec** and **writes**
- Sender sees publication of the receiver
- **Idea:** At some point, a node can be sure that:
    - the last sent data has been read
    - a fresh value is available in the memory

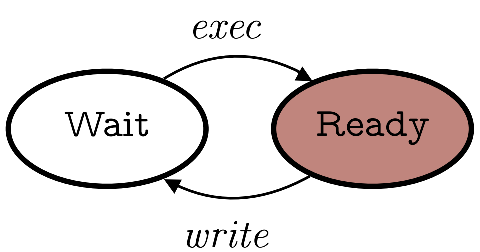
Time-Based LTTA



Time-Based LTTA

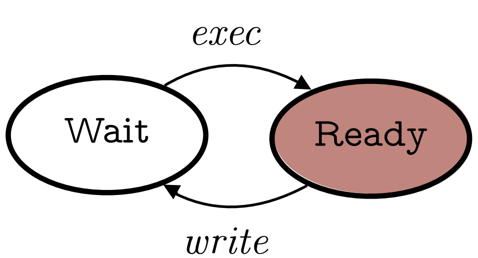
```
let node tb_controller (c, i, ro, om, default) = (o, im) where
  rec fi = fresh i
  and fro = fresh ro
  and init mem = default
  and init n = p
  and init state = Wait
  and match c with
      | false -> do done
      | true ->
        do
          match last state, (last n = 1), (fro or fi)  with
(* exec *)  | Wait, true, _ ->
              do state = Ready and n = q and mem = om
              and emit im = i done
(* write *) | Ready, _, true | Ready, true, _ ->
              do state = Wait and n = p
              and emit o = last mem done
(* wait *)  | _ -> do n = (last n) - 1 done
        done
```





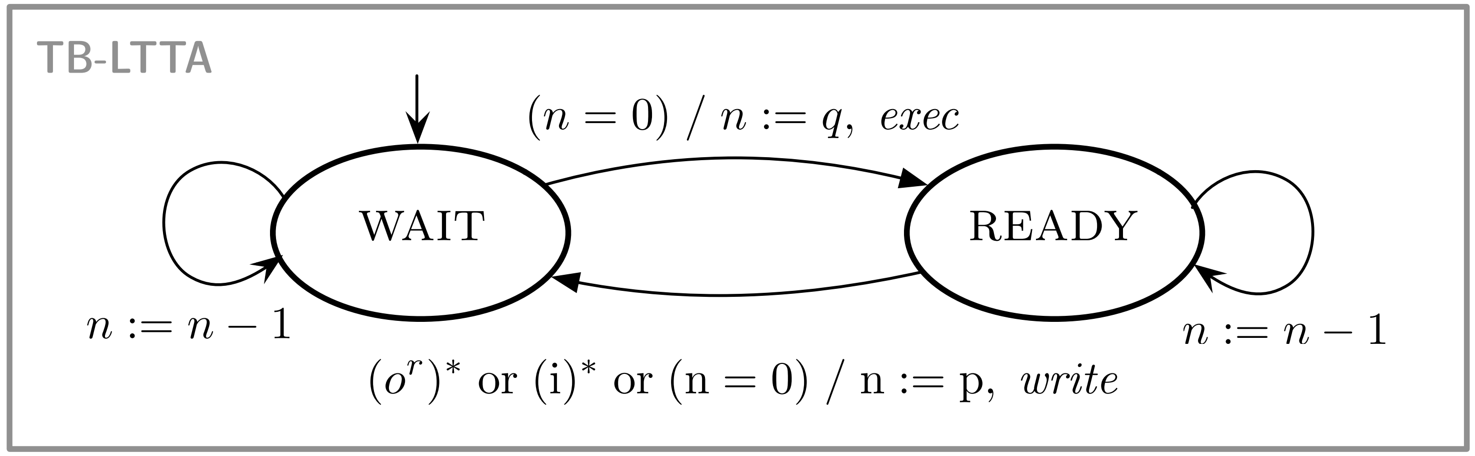
Sender

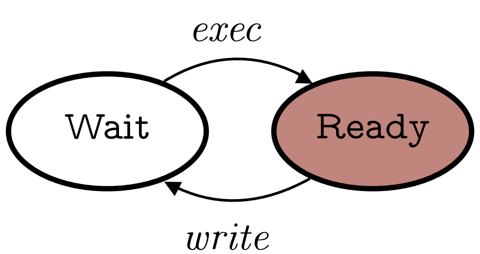
t



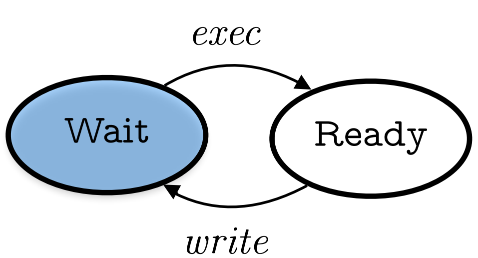
Receiver

t

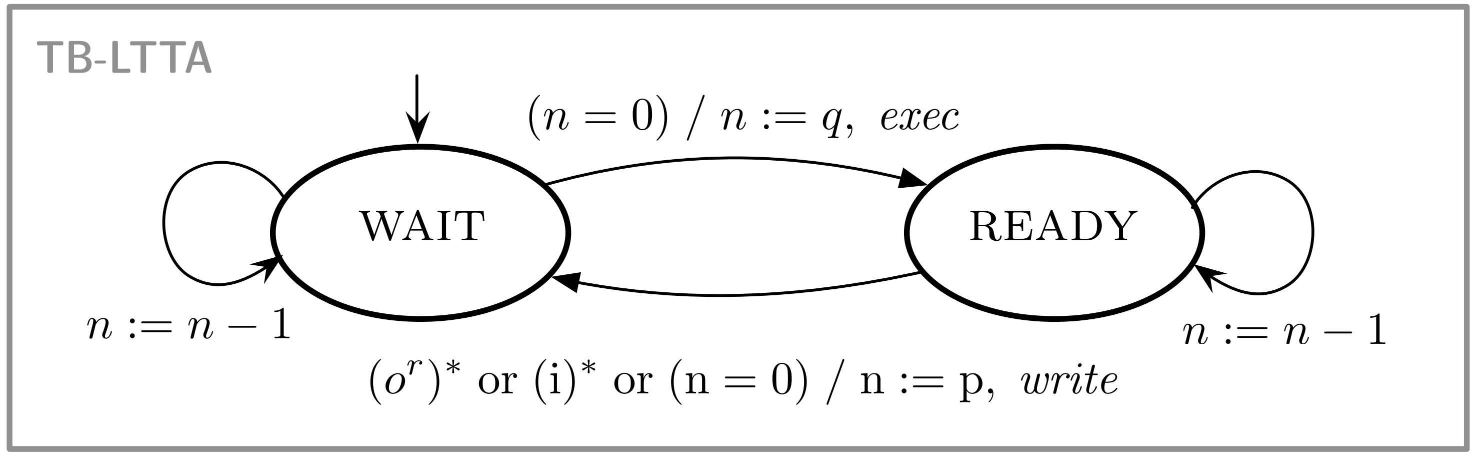




Sender

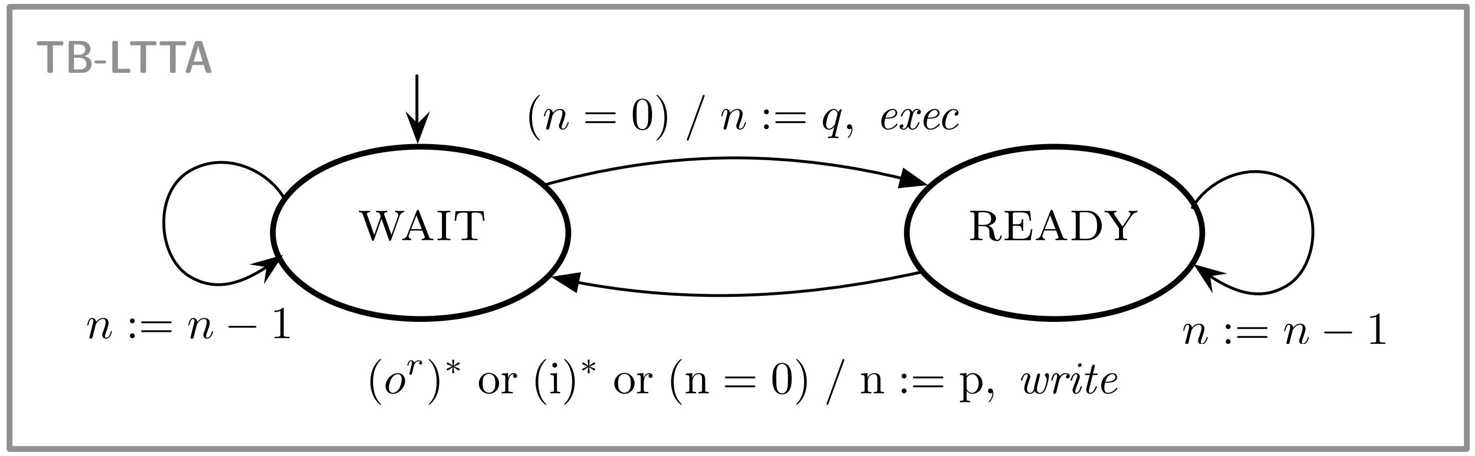


Receiver



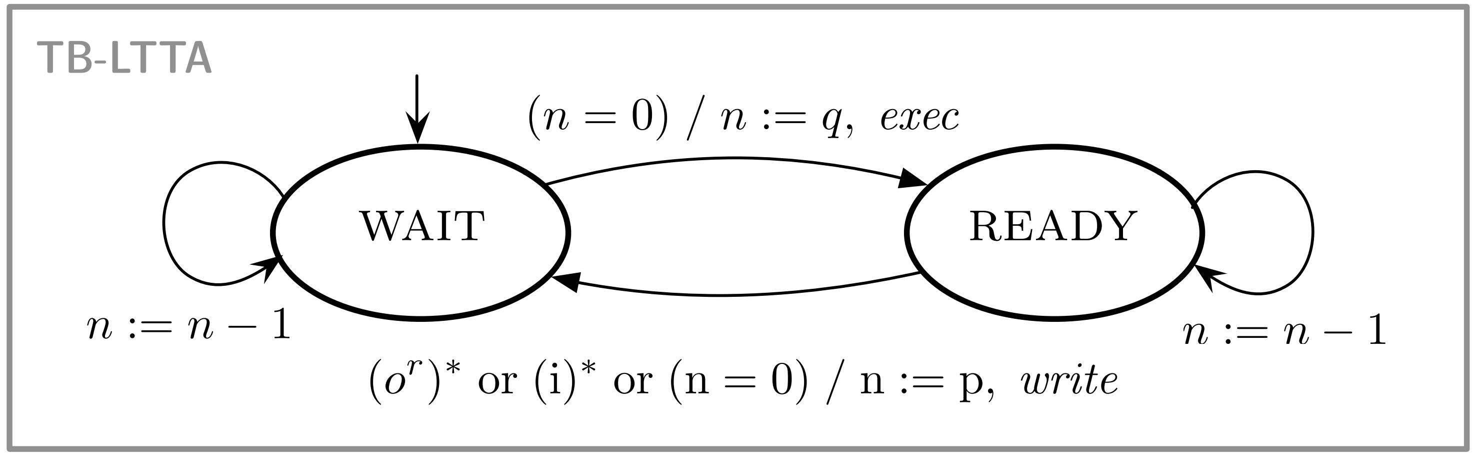
Sender

Receiver


TB-LTTA
(n = 0) / n := q, exec
WAIT
READY
n := n − 1
n := n − 1
(o^r)* or (i)* or (n = 0) / n := p, write

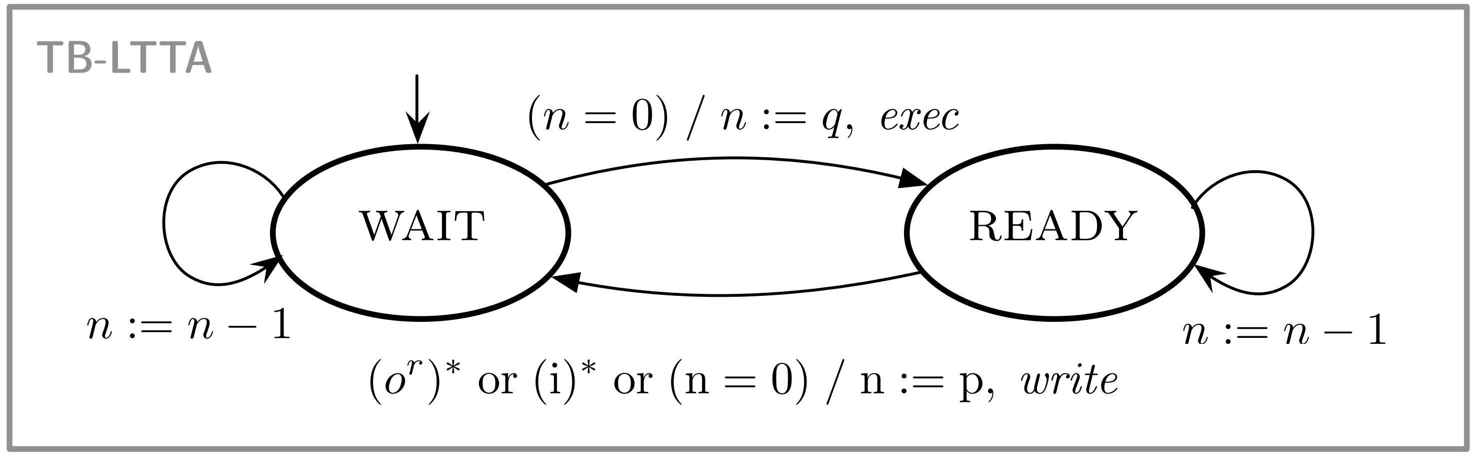

exec
Wait
Ready
write
Sender
write
3
t

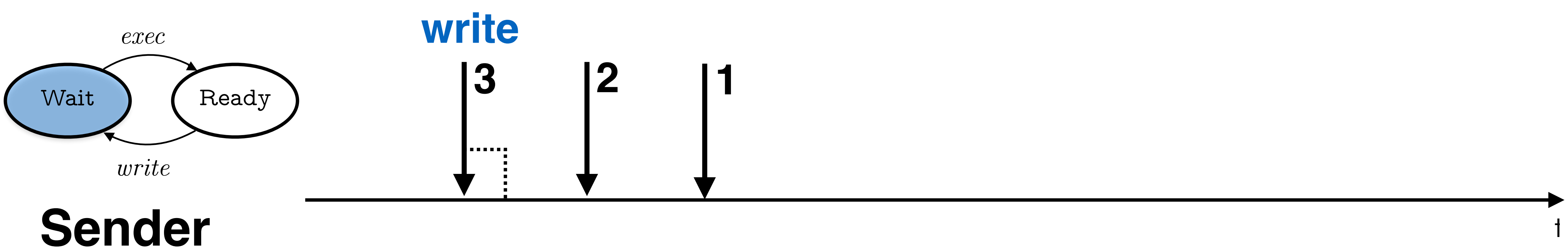

exec
Wait
Ready
write
Receiver
write
0/3
2
t


TB-LTTA
$(n = 0)$ / $n := q$, *exec*
WAIT
READY
$n := n - 1$
$n := n - 1$
$(o^r)^*$ or $(\mathrm{i})^*$ or $(\mathrm{n} = 0)$ / $\mathrm{n} := \mathrm{p}$, *write*

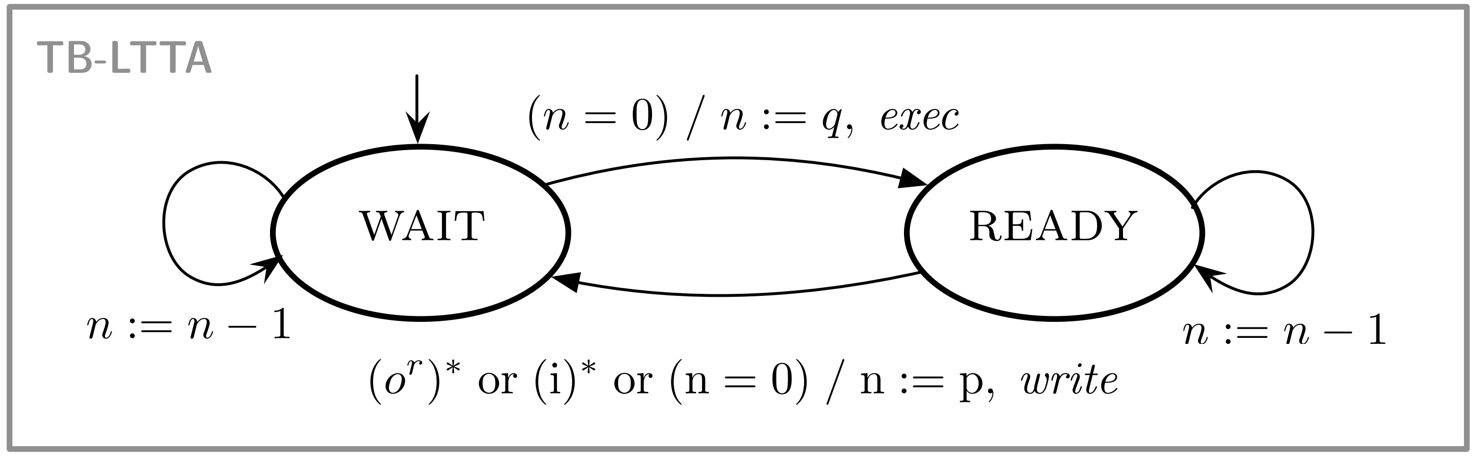

exec
Wait
Ready
write
Sender
write
3
2
t

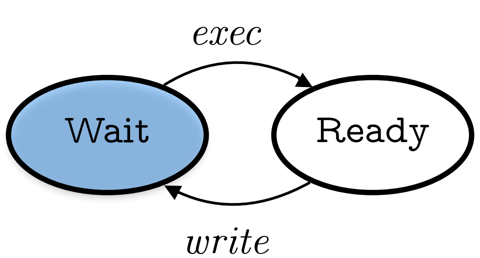

exec
Wait
Ready
write
Receiver
write
0/3
2
t


TB-LTTA
(n = 0) / n := q, exec
WAIT
READY
n := n − 1
n := n − 1
(o^r)* or (i)* or (n = 0) / n := p, write

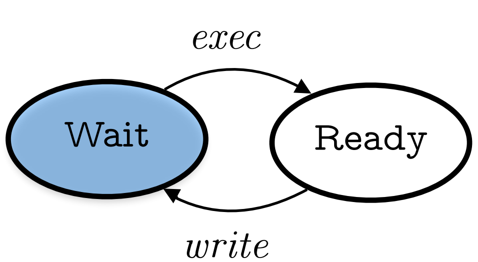

exec
Wait
Ready
write
write
3
2
1
t
Sender

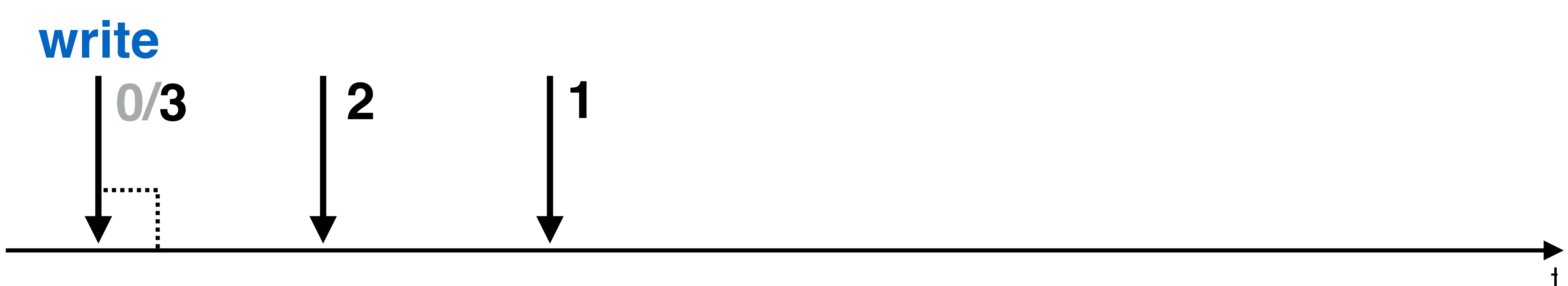

exec
Wait
Ready
write
write
0/3
2
t
Receiver



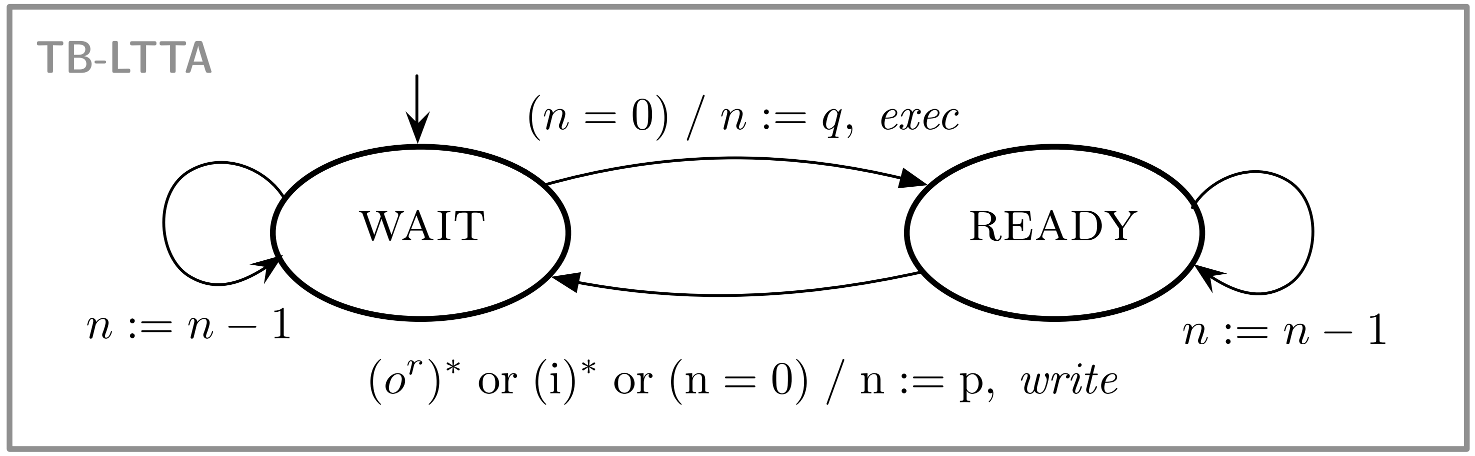


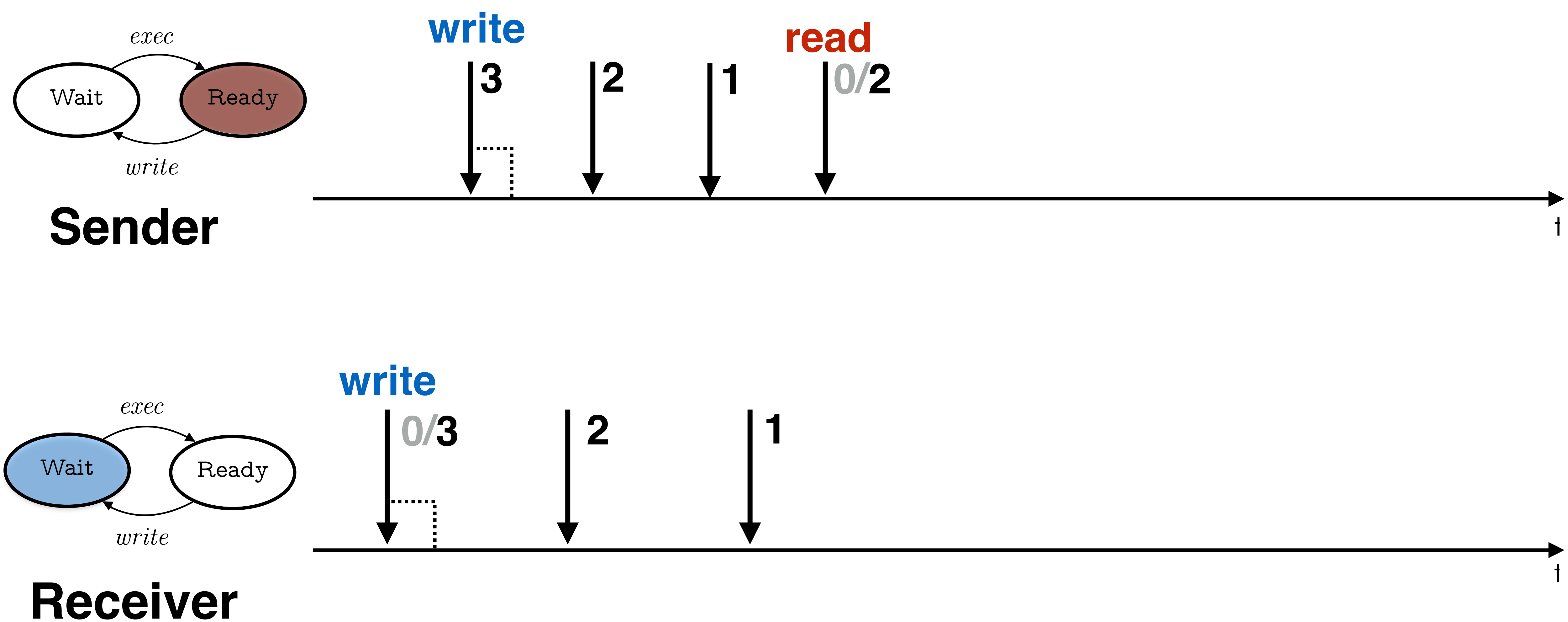
Sender

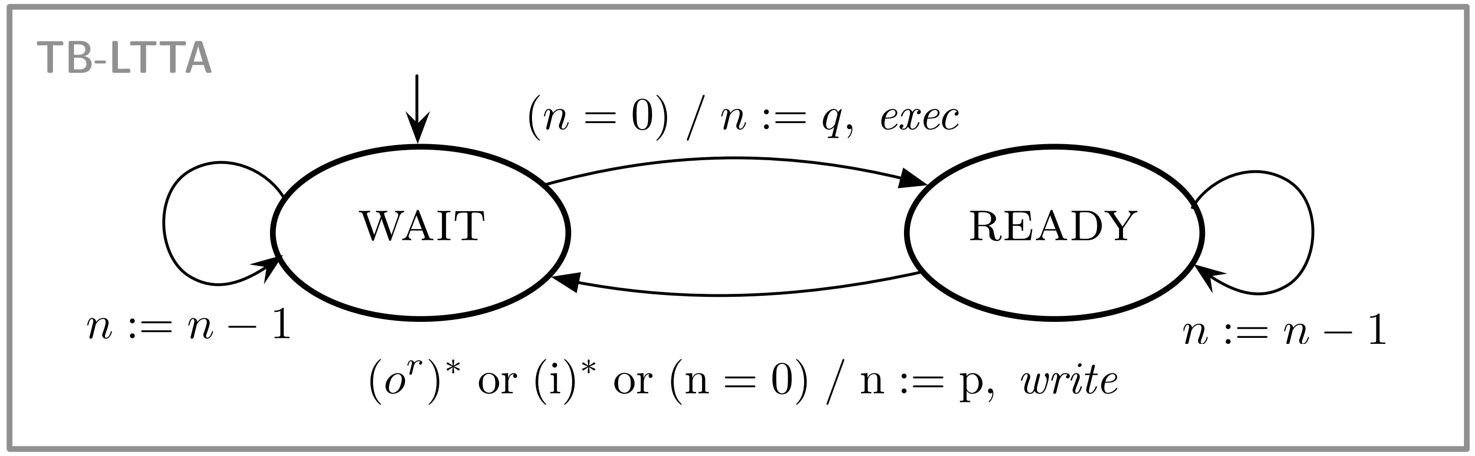


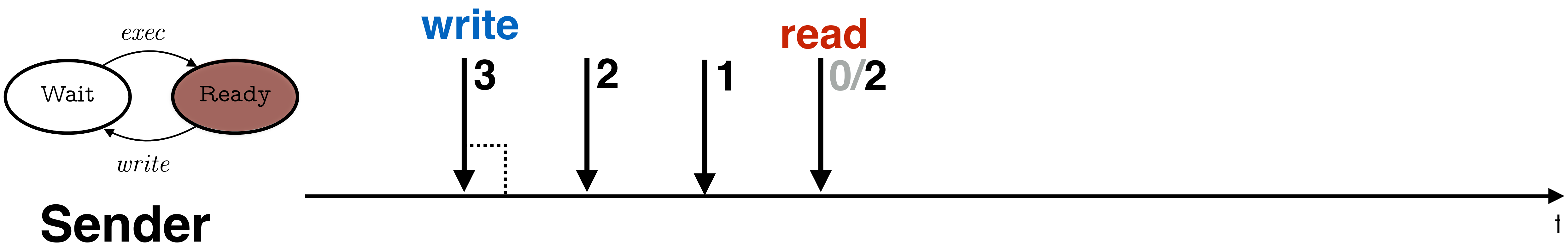


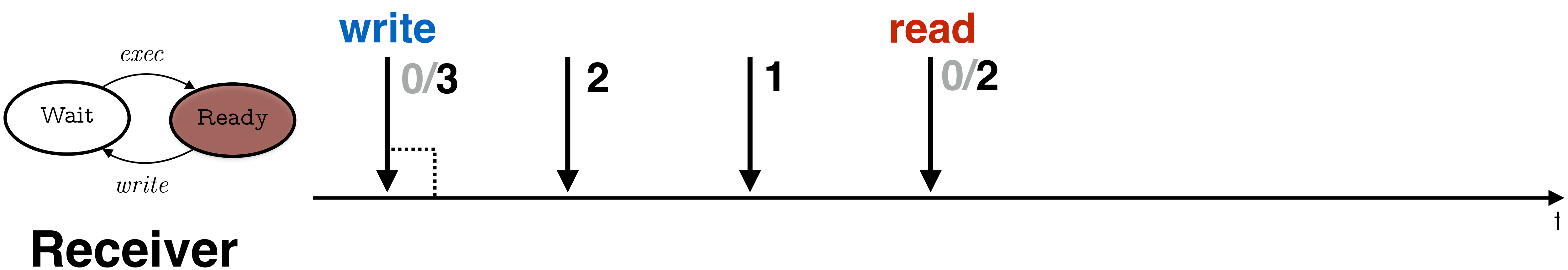
Receiver

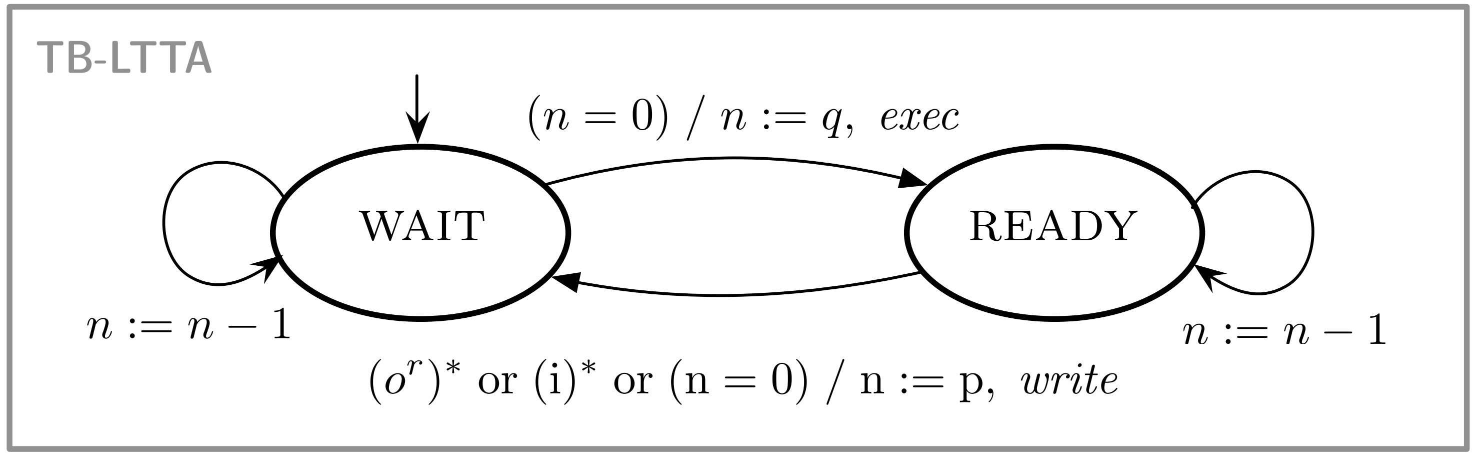

TB-LTTA
WAIT
READY
$(n = 0) \; / \; n := q, \; exec$
$n := n - 1$
$n := n - 1$
$(o^r)^*$ or $(\mathrm{i})^*$ or $(\mathrm{n} = 0) \; / \; \mathrm{n} := \mathrm{p}, \; write$

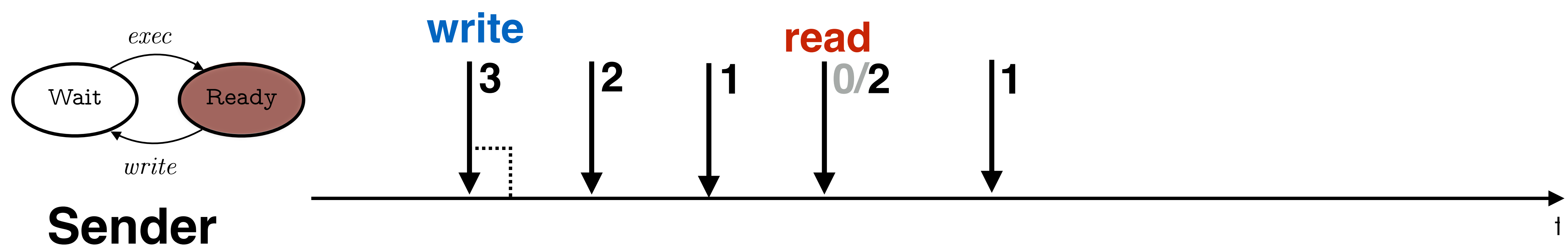

exec
Wait
Ready
write
Sender
write
read
3
2
1
0/2
t
exec
Wait
Ready
write
Receiver
write
0/3
2
1
t

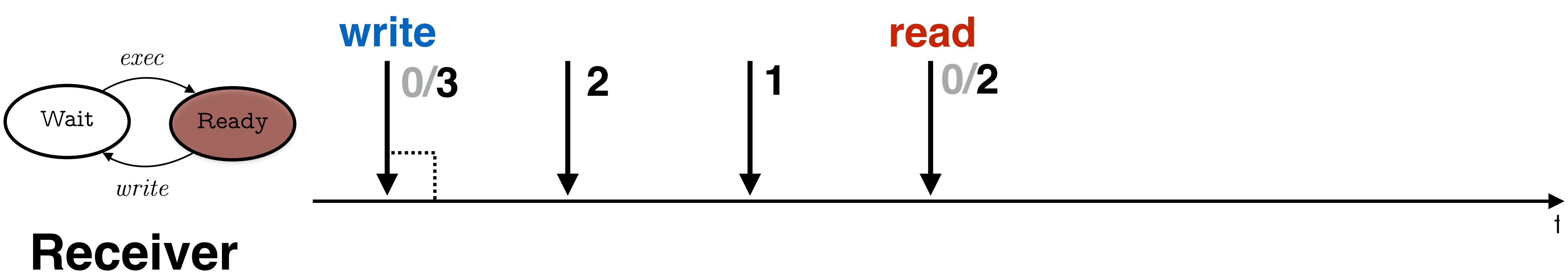

TB-LTTA
$(n = 0)$ / $n := q$, exec
WAIT
READY
$n := n - 1$
$n := n - 1$
$(o^r)^*$ or $(i)^*$ or $(n = 0)$ / $n := p$, write

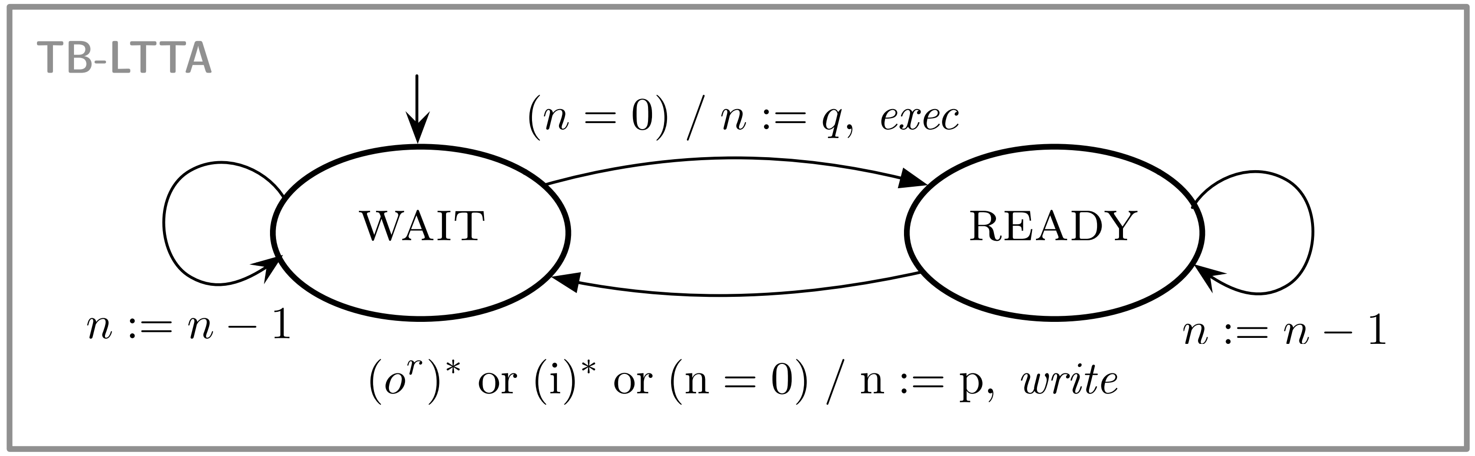

exec
Wait
Ready
write
Sender
write
3
2
1
read
0/2
t

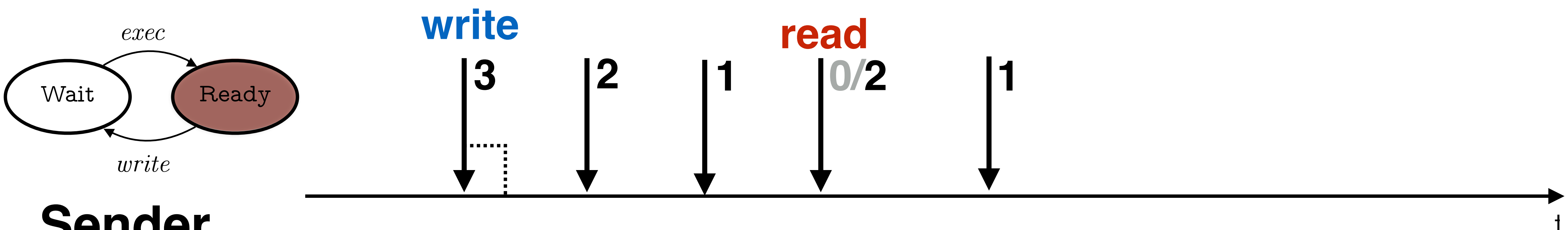

exec
Wait
Ready
write
Receiver
write
0/3
2
1
read
0/2
t


TB-LTTA
(n = 0) / n := q, exec
WAIT
READY
n := n − 1
n := n − 1
(o^r)* or (i)* or (n = 0) / n := p, write

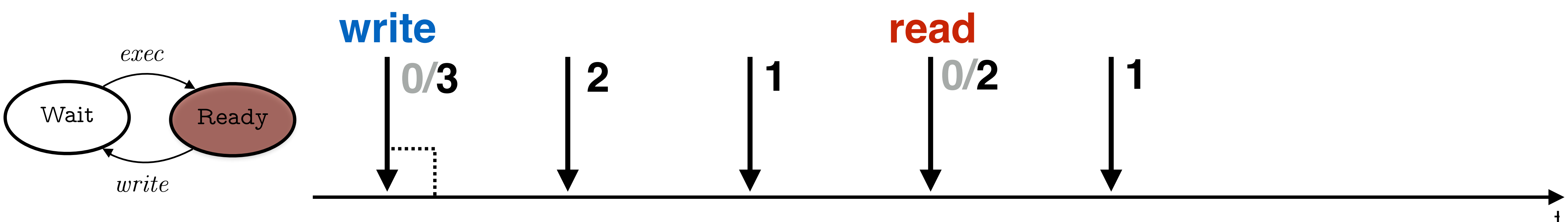

exec
Wait
Ready
write
Sender
write
read
3
2
1
0/2
1
t


exec
Wait
Ready
write
Receiver
write
read
0/3
2
1
0/2
t

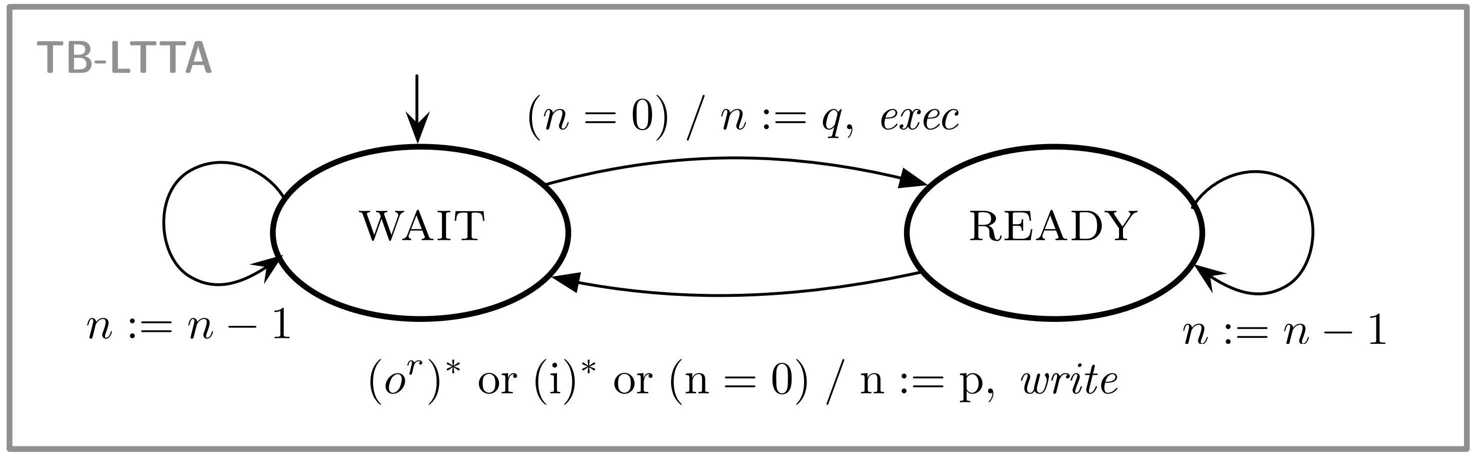


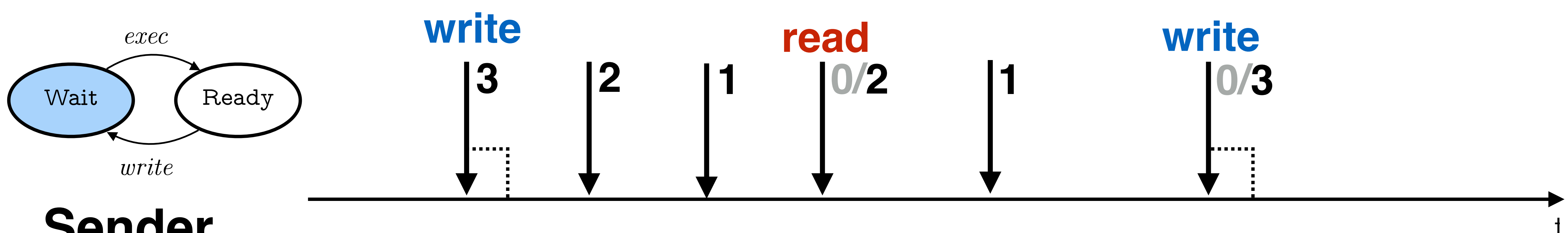


Sender

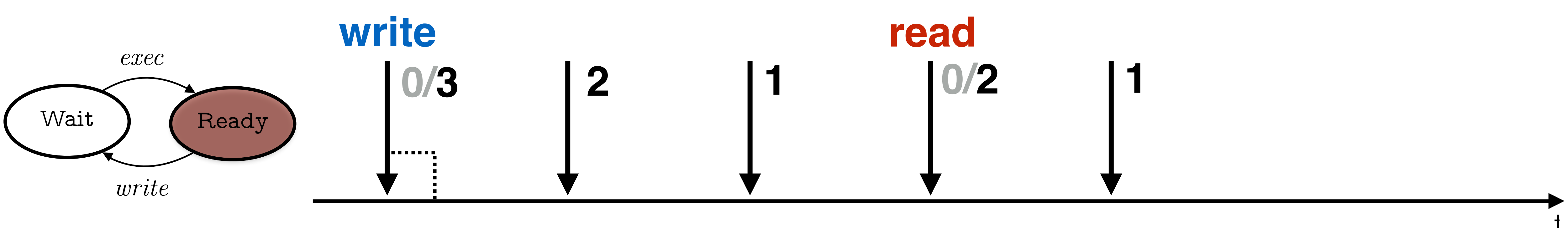


Receiver

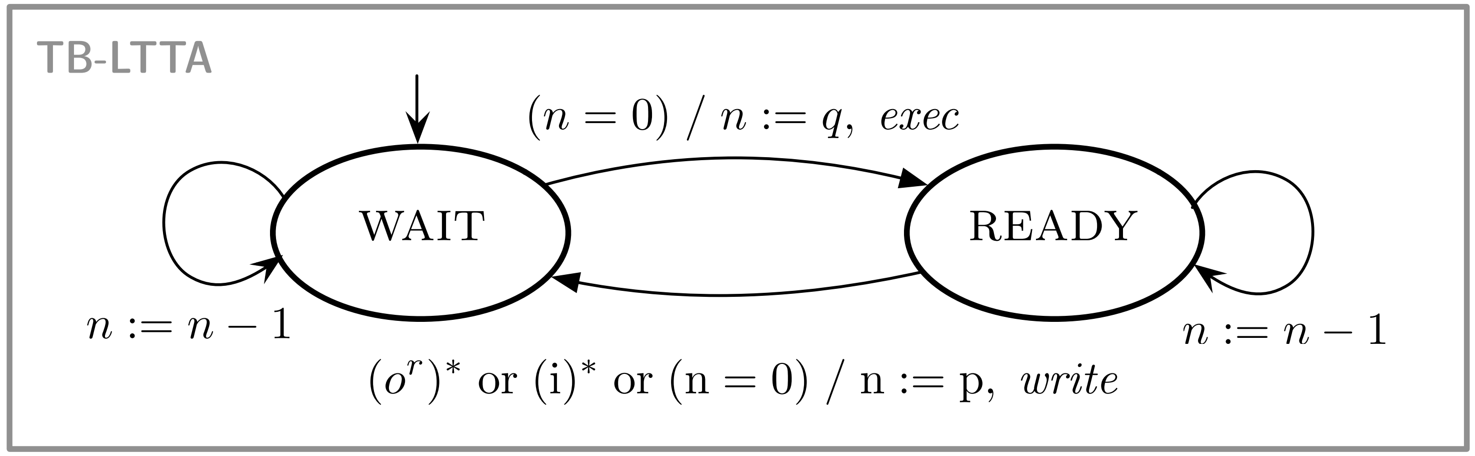


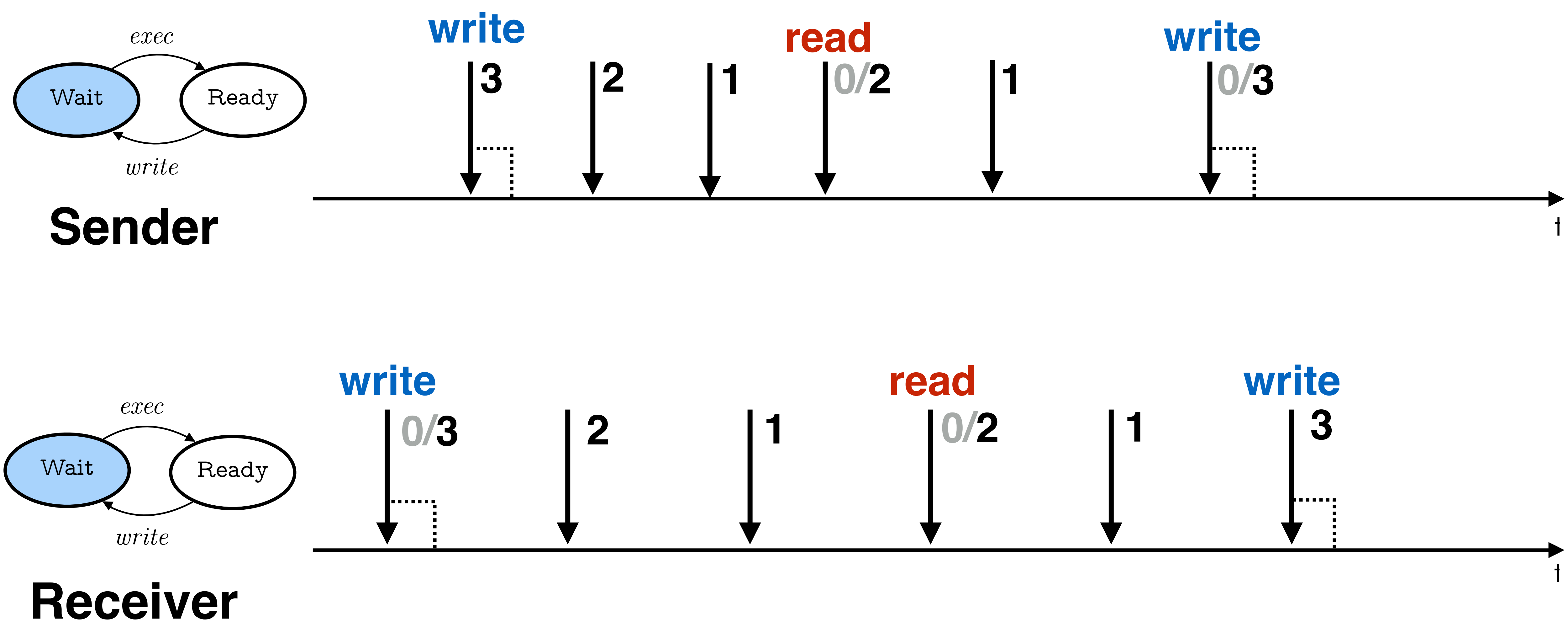


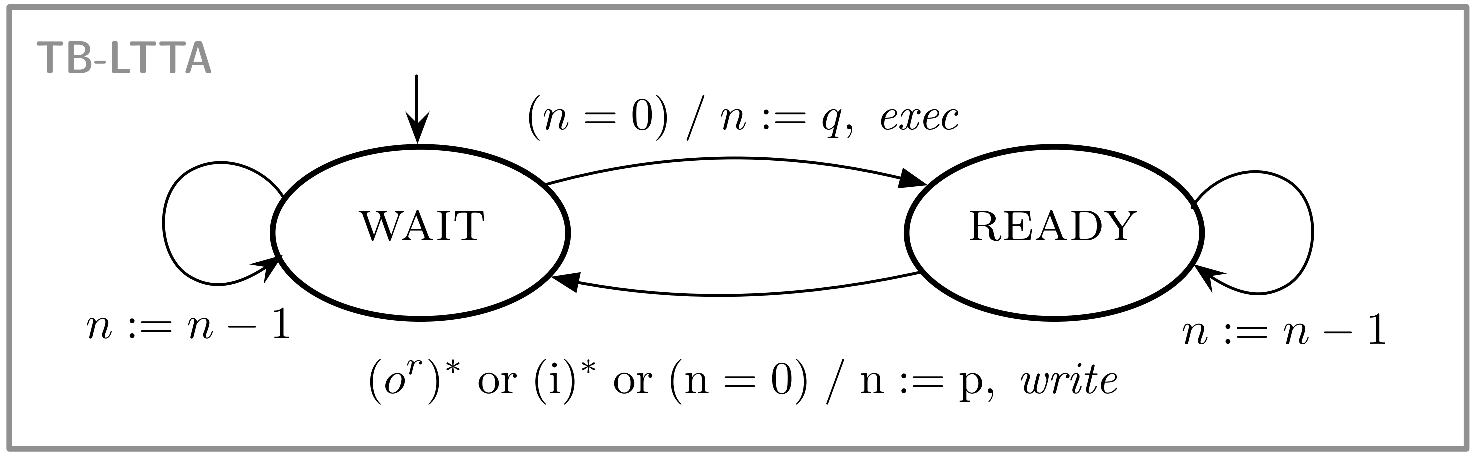
Sender

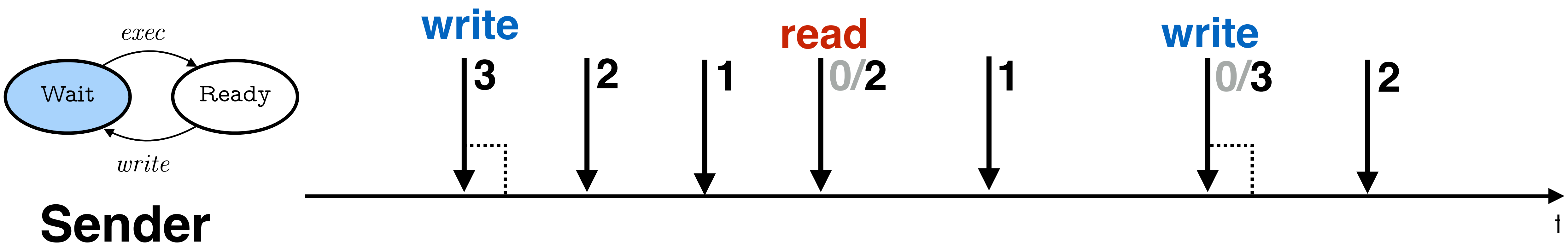


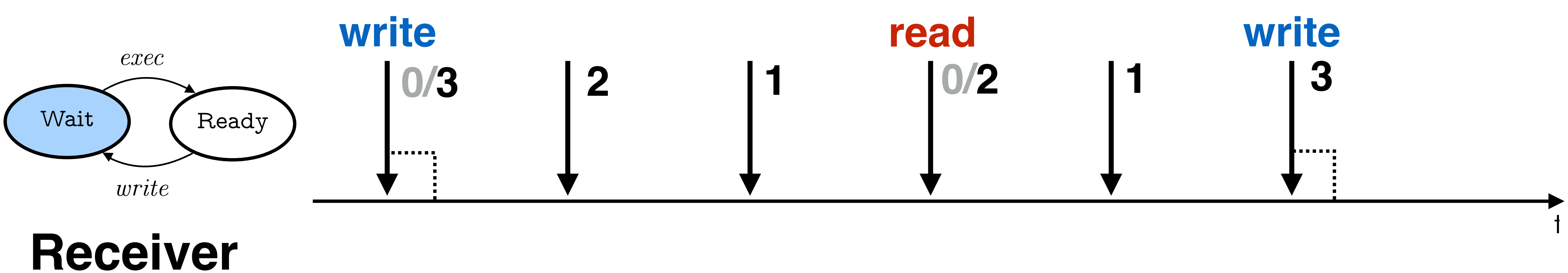
Receiver

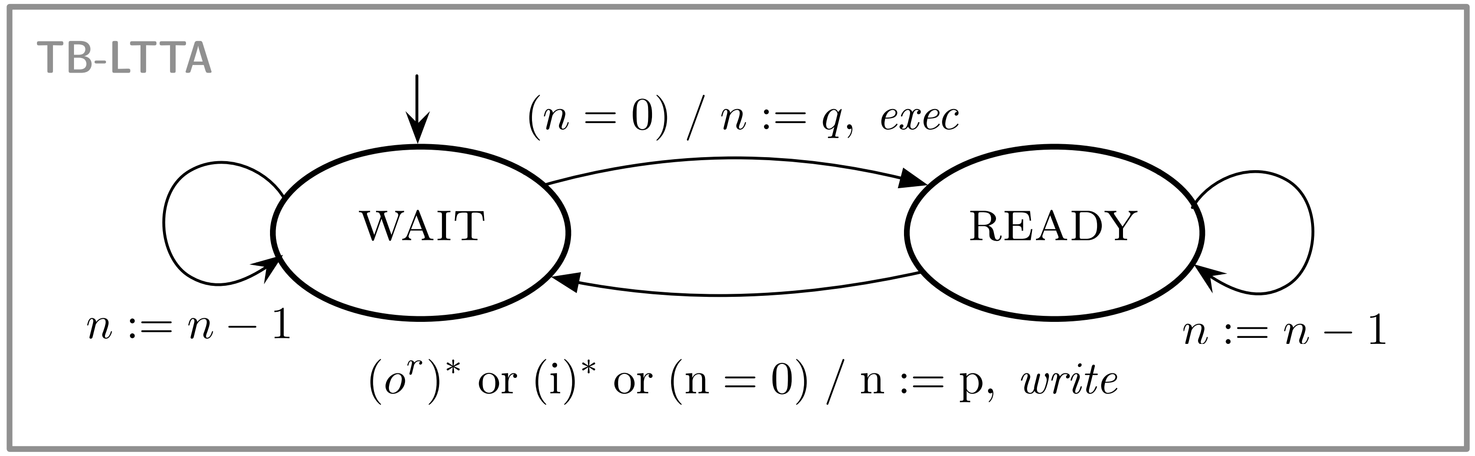

TB-LTTA
WAIT
READY
$(n = 0) \ / \ n := q, \ exec$
$n := n - 1$
$n := n - 1$
$(o^r)^*$ or $(\mathrm{i})^*$ or $(\mathrm{n} = 0) \ / \ \mathrm{n} := \mathrm{p}, \ write$


exec
Wait
Ready
write
Sender
write
3
2
1
read
0/2
1
write
0/3
t
exec
Wait
Ready
write
Receiver
write
0/3
2
1
read
0/2
1
write
3
t

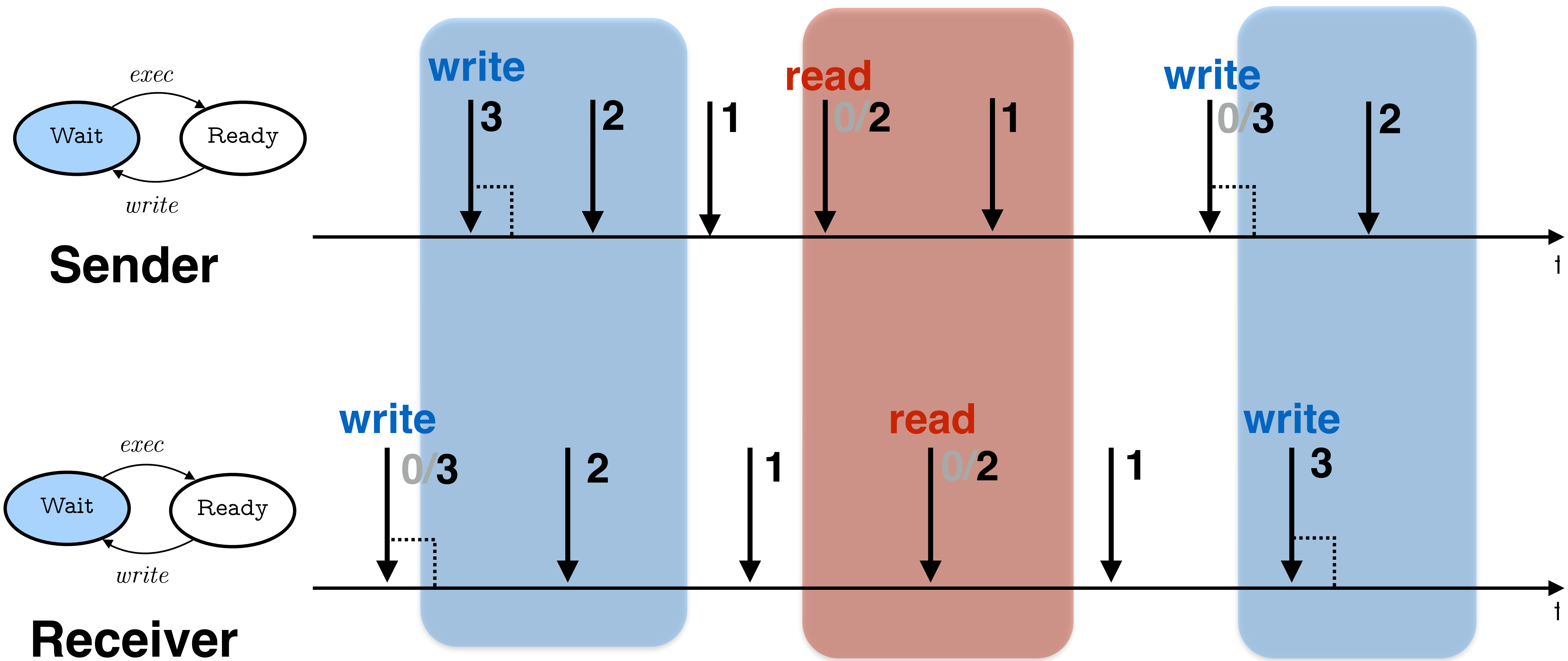

TB-LTTA
WAIT
READY
$(n = 0) \ / \ n := q, \ exec$
$n := n - 1$
$n := n - 1$
$(o^r)^*$ or $(\mathrm{i})^*$ or $(\mathrm{n} = 0) \ / \ \mathrm{n} := \mathrm{p}, \ write$


exec
Wait
Ready
write
Sender
write
3
2
1
read
0/2
1
write
0/3
2
t


exec
Wait
Ready
write
Receiver
write
0/3
2
1
read
0/2
1
write
3
t



TB-LTTA
WAIT
READY
$(n = 0)\ /\ n := q,\ exec$
$n := n - 1$
$n := n - 1$
$(o^r)^*$ or $(\mathrm{i})^*$ or $(\mathrm{n} = 0)\ /\ \mathrm{n} := \mathrm{p},\ write$



exec
Wait
Ready
write
Sender
write
3
2
1
read
0/2
1
write
0/3
2
t
exec
Wait
Ready
write
Receiver
write
0/3
2
1
read
0/2
1
write
3
t

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

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

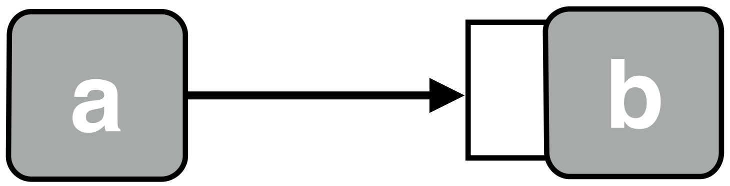
The Time-Based Protocol

Proof sketch

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

□

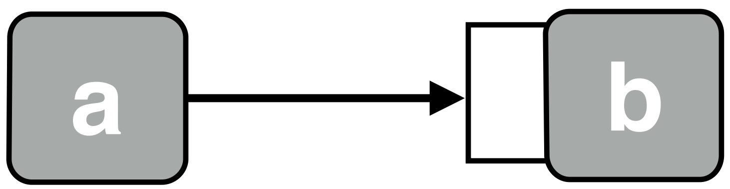
The Time-Based Protocol



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

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

The Time-Based Protocol



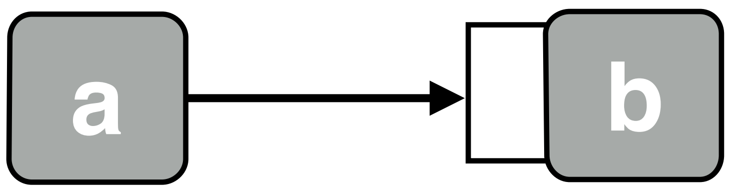
$$p > \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}}$$

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

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

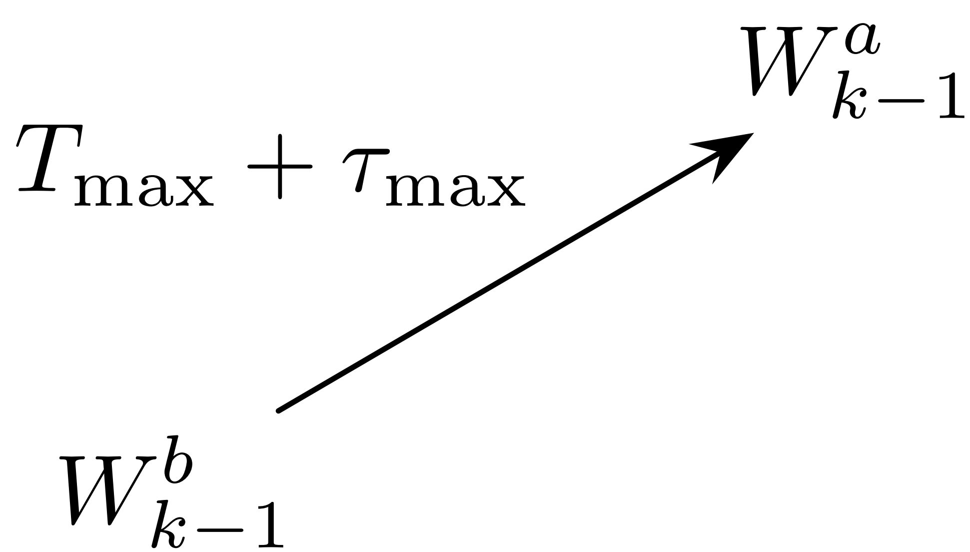
W^b_{k-1}

The Time-Based Protocol

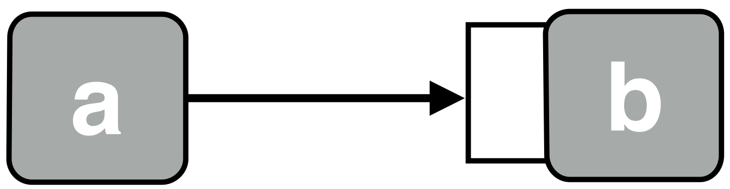


$$
\begin{aligned}
p &> \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} \\
q &> \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)
\end{aligned}
$$

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

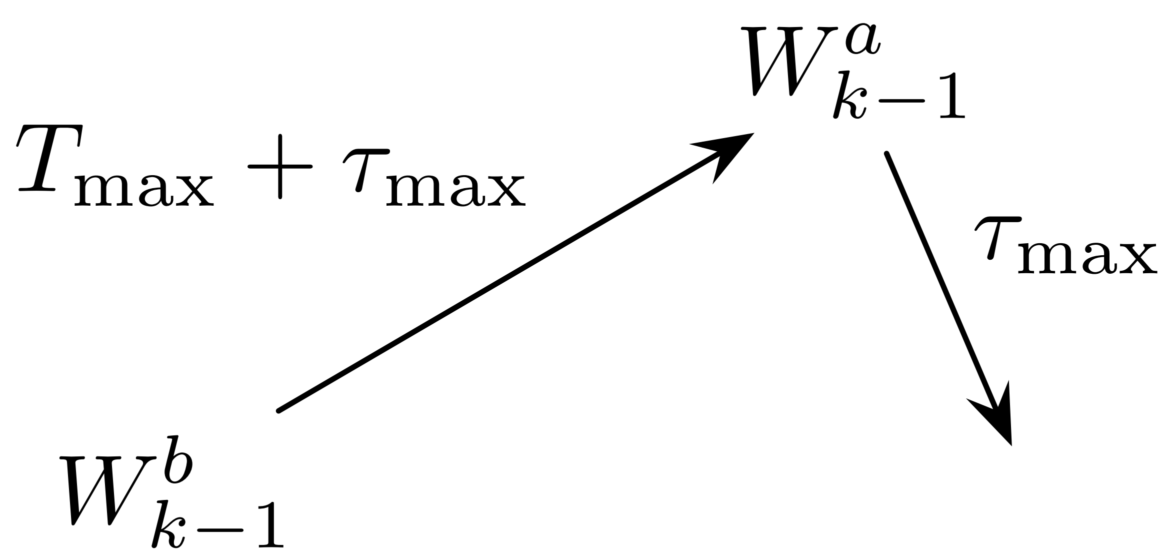


The Time-Based Protocol

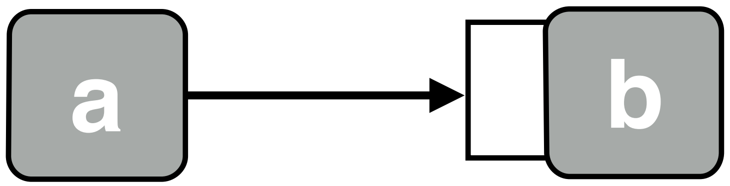


$$
\begin{aligned}
p &> \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} \\
q &> \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)
\end{aligned}
$$

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

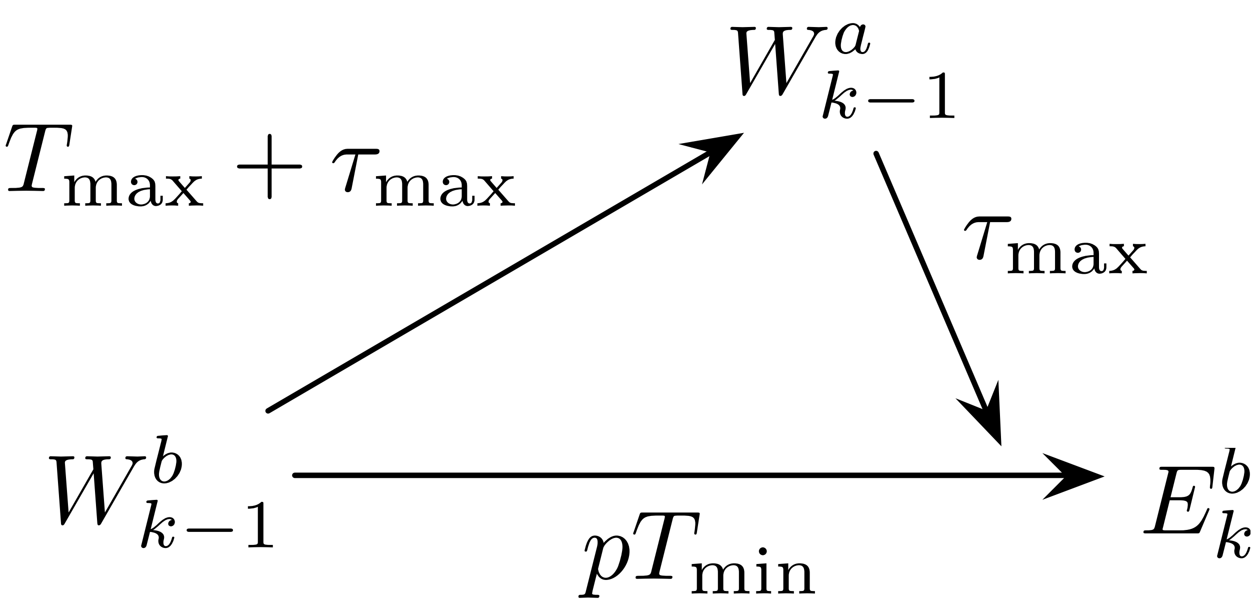


The Time-Based Protocol

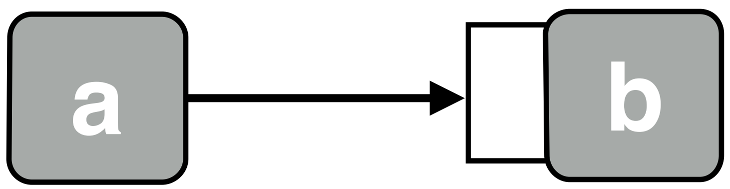


$$
\begin{aligned}
p &> \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} \\
q &> \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)
\end{aligned}
$$

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



The Time-Based Protocol

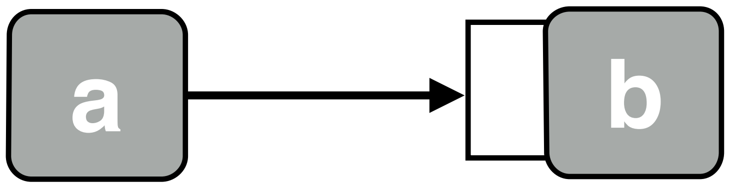


$$p > \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}}$$
$$q > \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)$$

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

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

The Time-Based Protocol



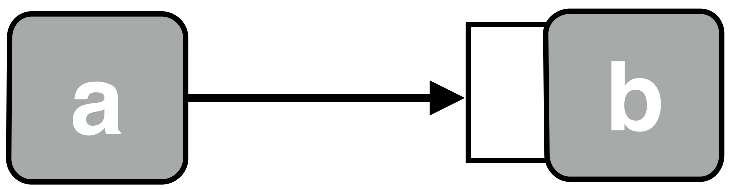
$$p > \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}}$$
$$q > \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)$$

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

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

W^a_{k-1}

The Time-Based Protocol



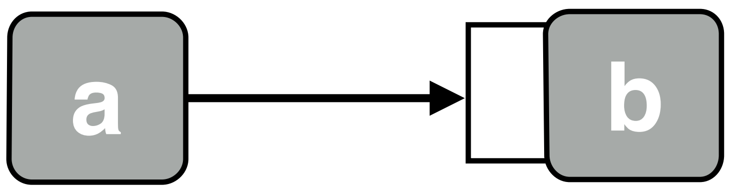
$$
\begin{aligned}
p &> \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} \\
q &> \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)
\end{aligned}
$$

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

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

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

The Time-Based Protocol



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

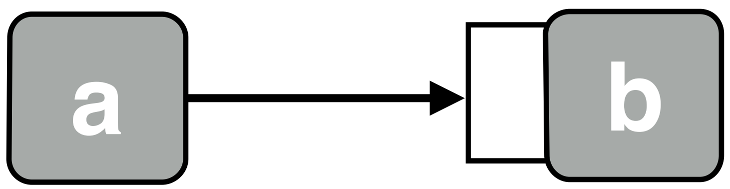
$$q > \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)$$

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



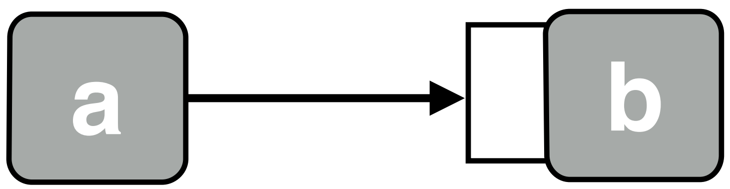
$$
\begin{aligned}
p &> \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} \\
q &> \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)
\end{aligned}
$$

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

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

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

The Time-Based Protocol



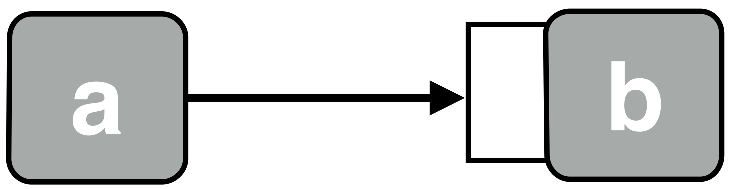
$$
\begin{aligned}
p &> \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} \\
q &> \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right)
\end{aligned}
$$

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

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

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

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: $E^b_k \prec W^a_k$

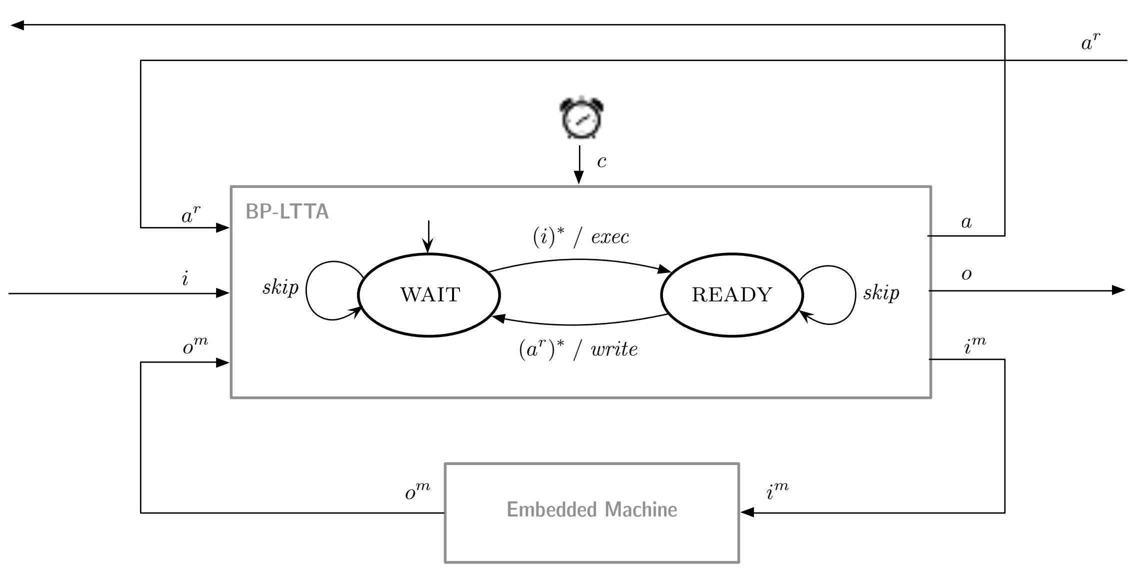
$$
\begin{tikzcd}
W^a_{k-1} \arrow[r, "pT_{\min}"] \arrow[dr, "T_{\max} + \tau_{\max}"'] & E^a_k \arrow[r, "qT_{\min}"] & W^a_k \arrow[d, "\tau_{\min}"] \\
& W^b_{k-1} \arrow[r, "pT_{\max}"'] & E^b_k
\end{tikzcd}
$$

The Time-Based Protocol

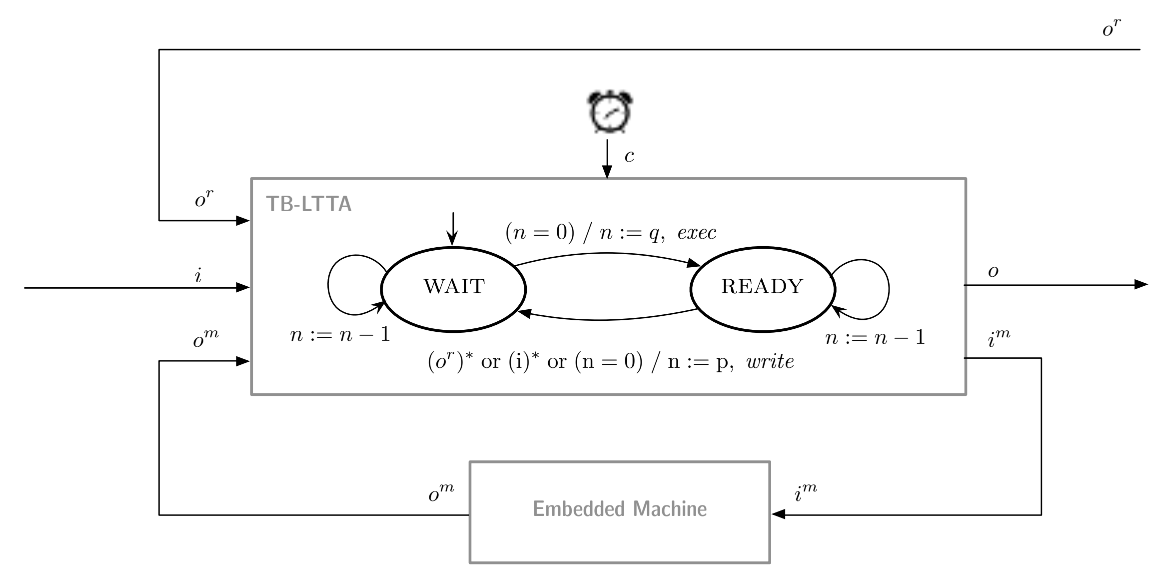
Corollary: The protocol ensures alternation between **exec** and **read** phases for each pair of communicating nodes

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
Pipeline	Yes	Optimal
Throughput (local)	$1/4T_{\max}$	$1/2T_{\max}$
Throughput (distant)	$1/2\tau_{\max}$	$1/2\tau_{\max}$

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
Pipeline	Yes	Optimal
Throughput (local)	$1/4T_{\max}$	$1/2T_{\max}$
Throughput (distant)	$1/2\tau_{\max}$	$1/2\tau_{\max}$

Conclusion

- Synchronous model of both the embedded application and the middleware
- A new proposition for the time-based protocol that does not require broadcast communication and allows pipelining
- Simulation of the protocol in Zélus
  Discrete model + link with continuous time

Next?

- Formal verification of the protocol.
  Problem: parametrised by the number of nodes

- Model the non-determinism of the architecture.
  Discrete abstraction (quasi-synchrony is not enough!)

- Real world experiments with LTTA protocols