

# **Redundancy Management in an Automatic Subway Line: a Protocol based on the Timed Asynchronous Model**

**David Powell**  
dpowell@laas.fr

**Jean Arlat**  
arlat@laas.fr

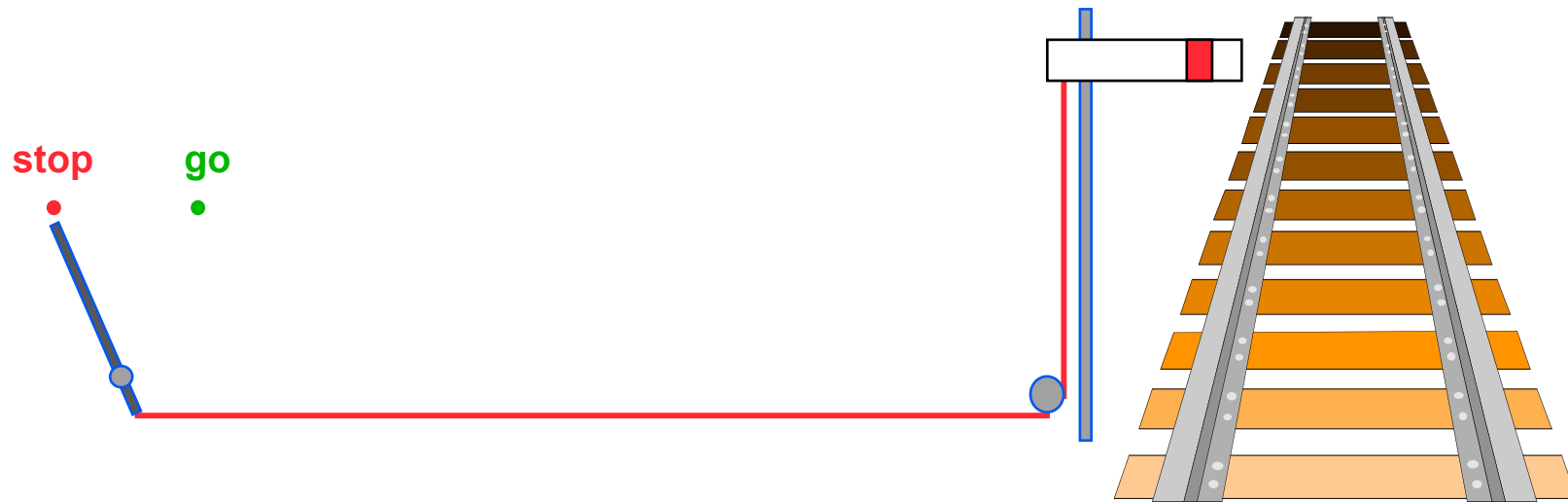
**Didier Essamé**  
didier.essame@siemens.com

**31<sup>st</sup> Spring School on Theoretical Computer  
Science : Distributed Algorithms**

**Porquerolles, France, 4-8 May 2003**

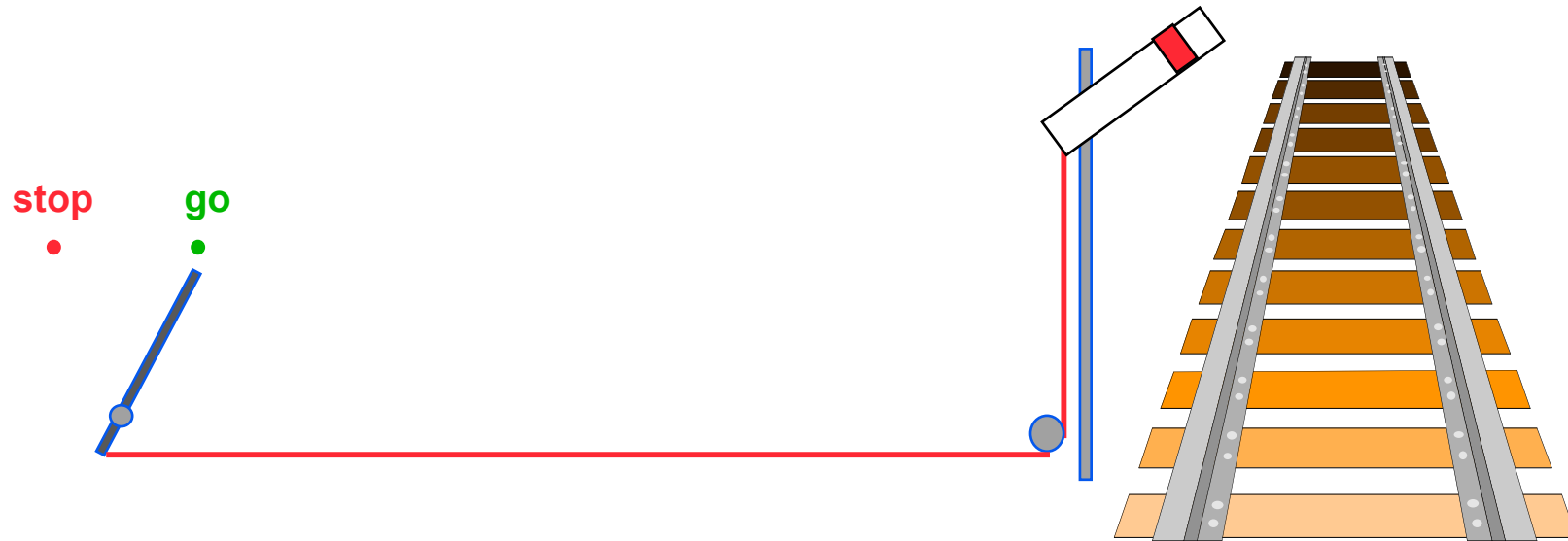
# An Early Example of Fail-Safe Design

---



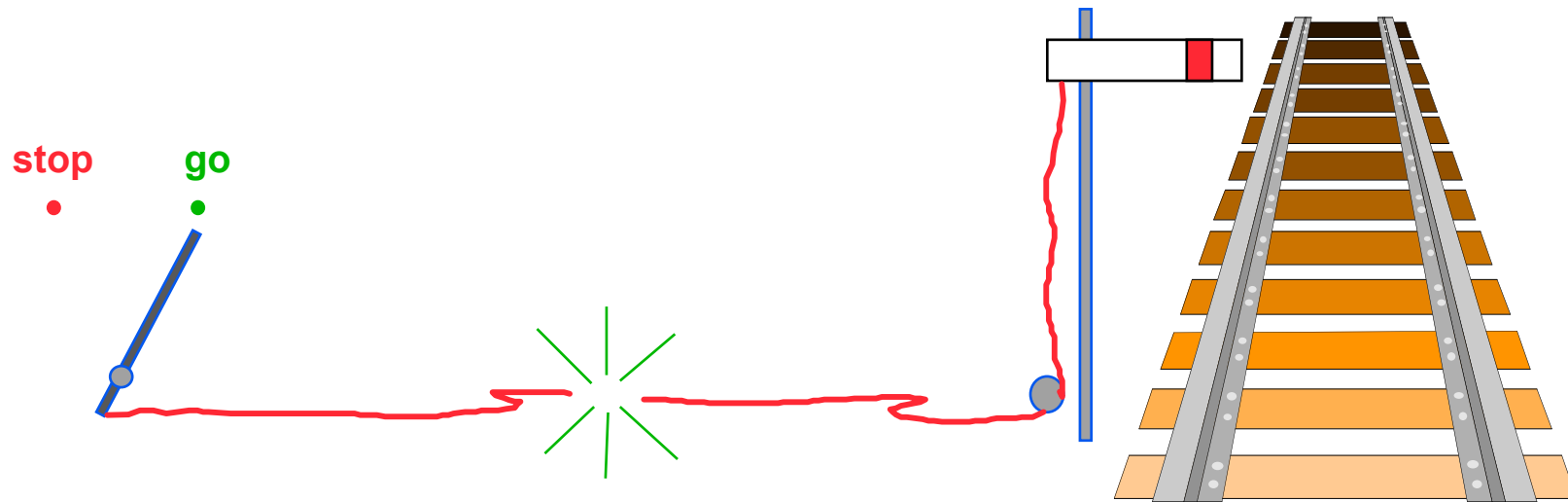
# An Early Example of Fail-Safe Design

---



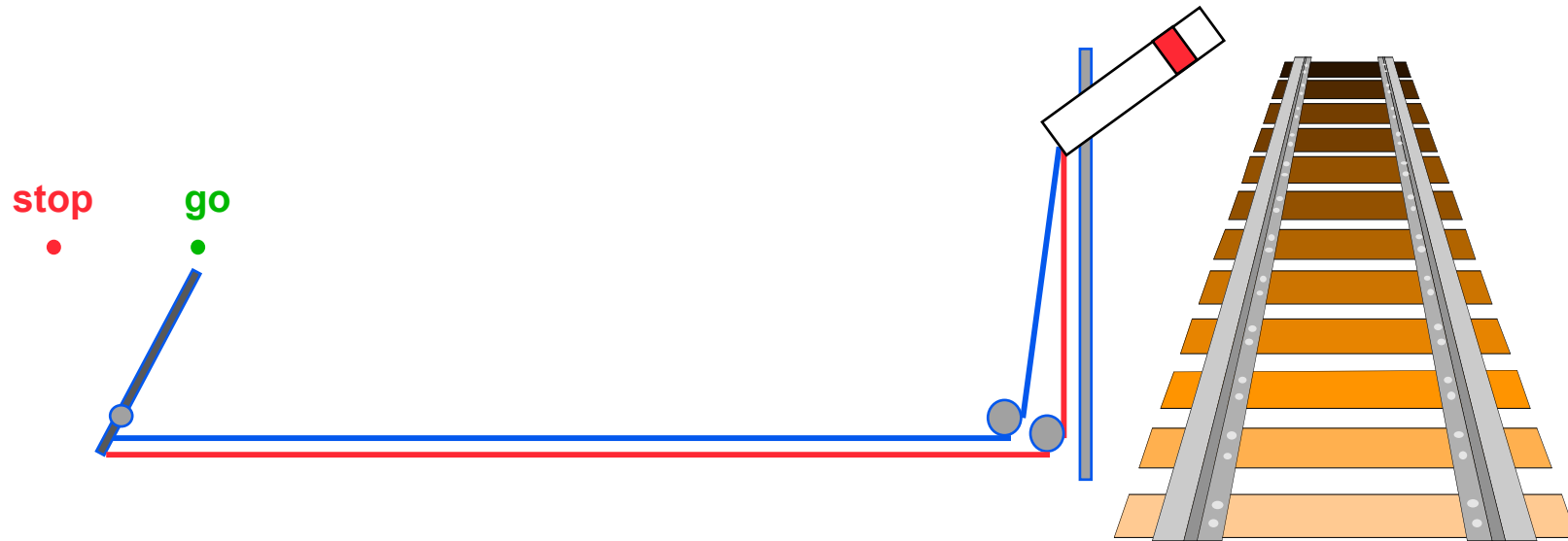
# An Early Example of Fail-Safe Design

---



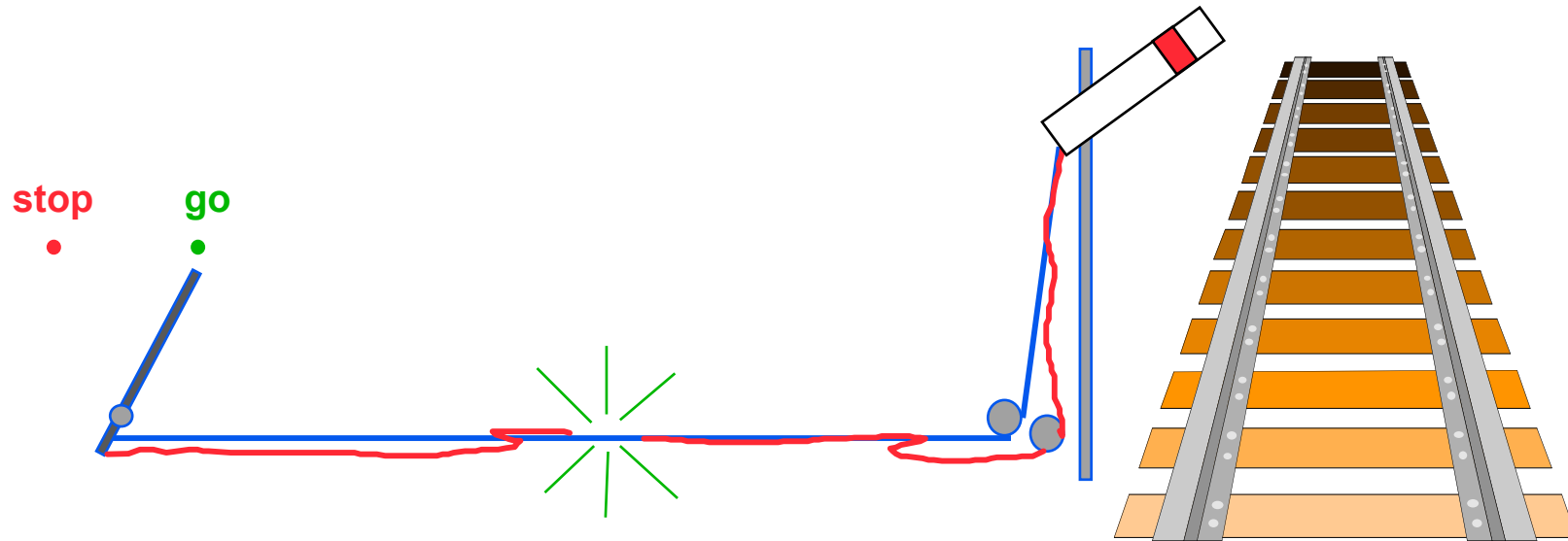
# Fault-Tolerant Fail-Safe Design?

---



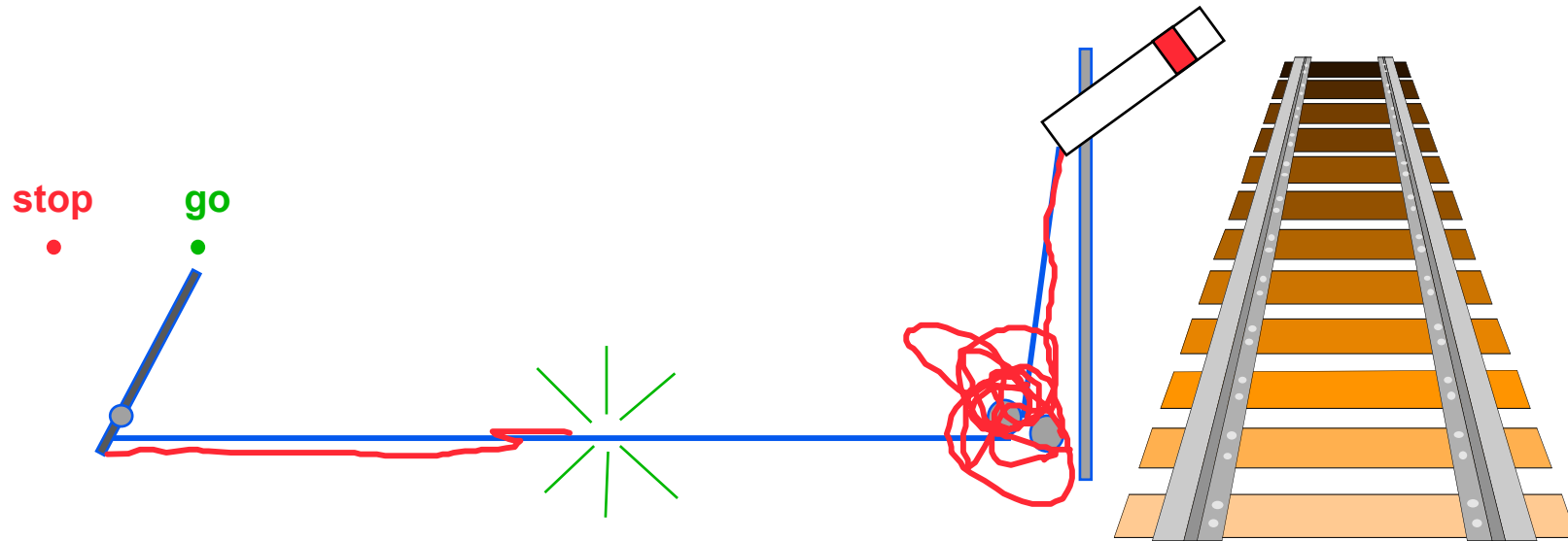
# Fault-Tolerant Fail-Safe Design?

---



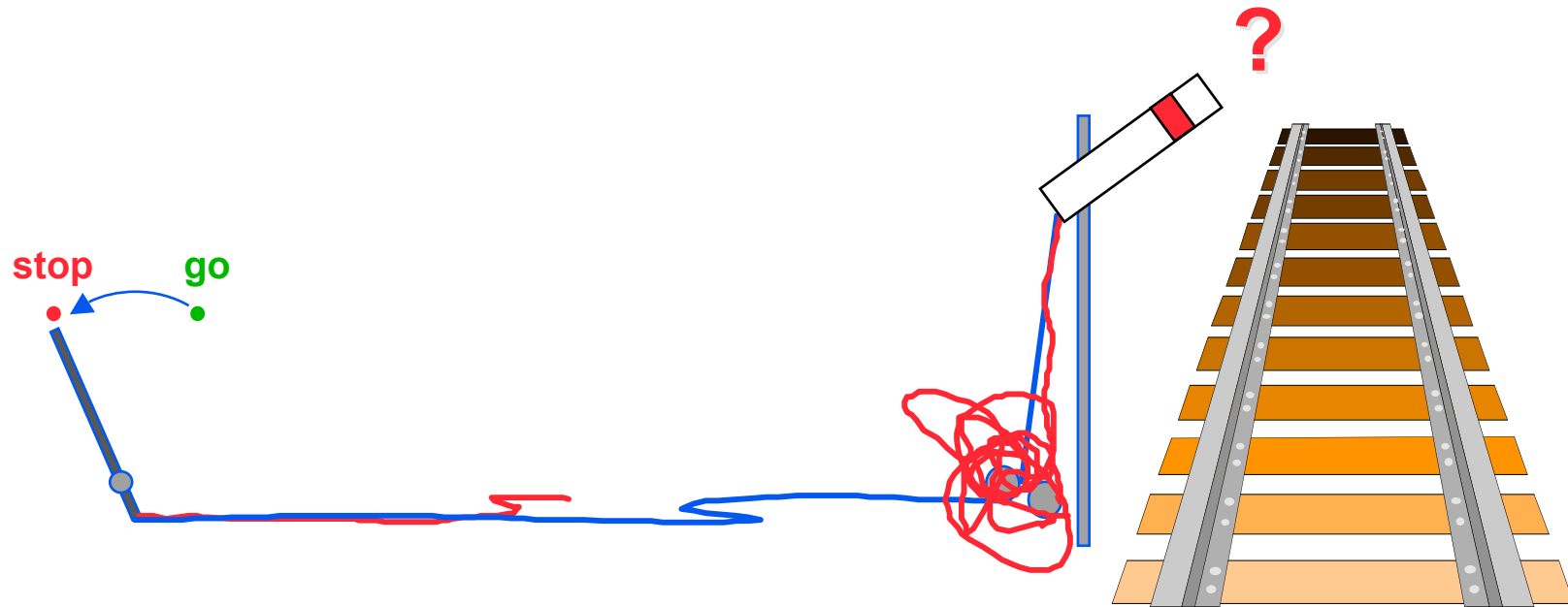
# Fault-Tolerant Fail-Safe Design?

---

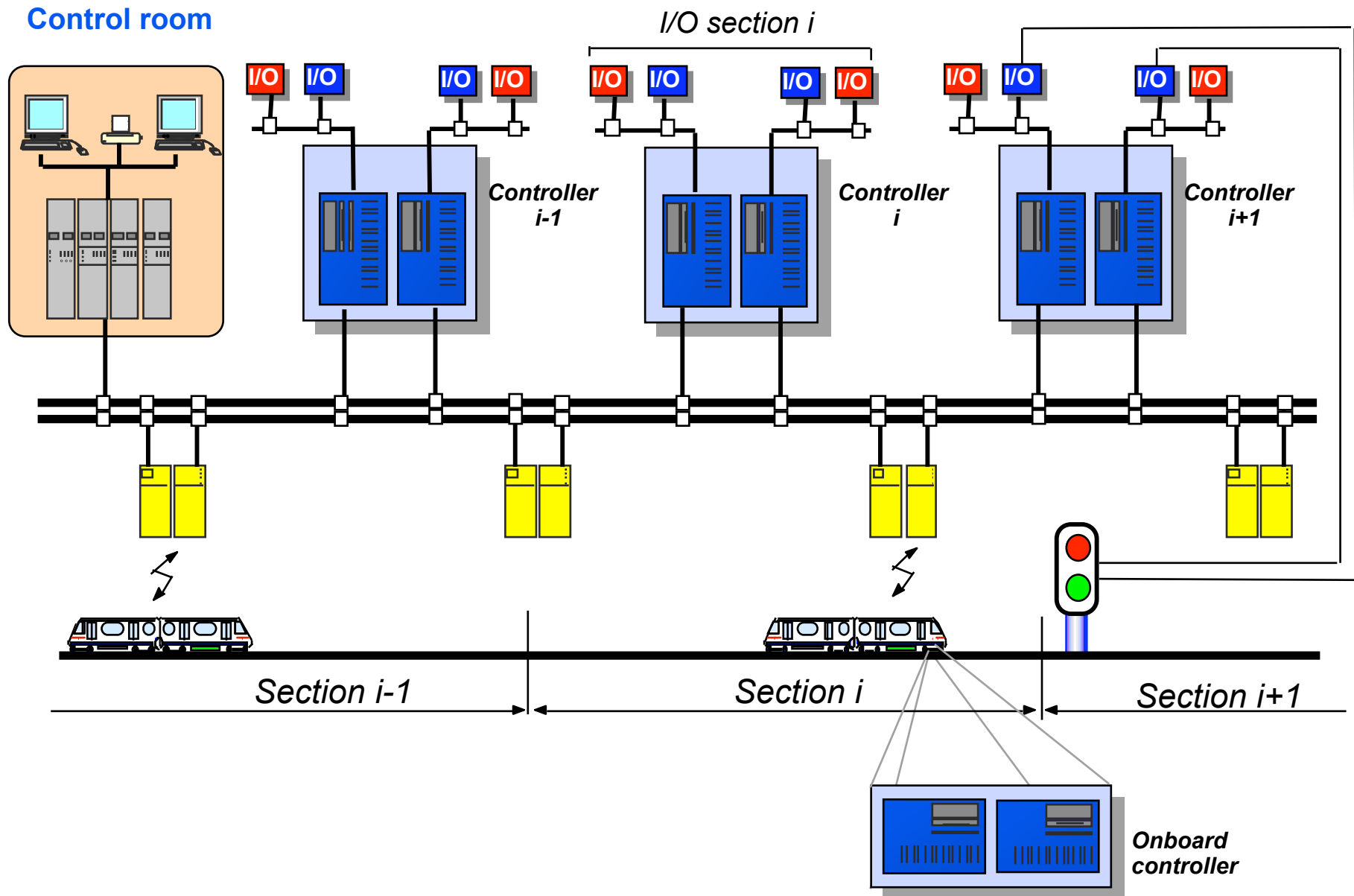


# Fault-Tolerant Fail-Safe Design?

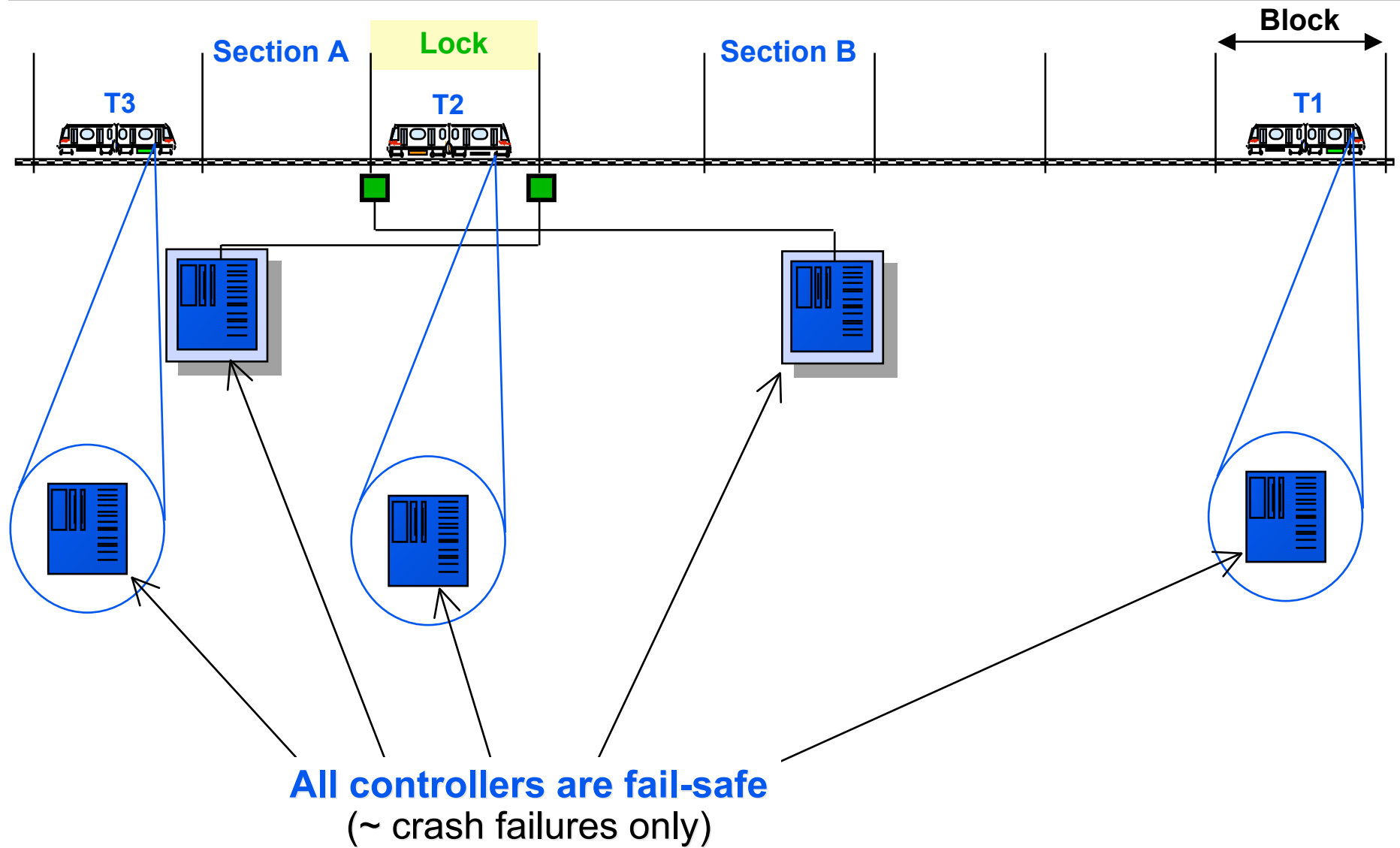
---



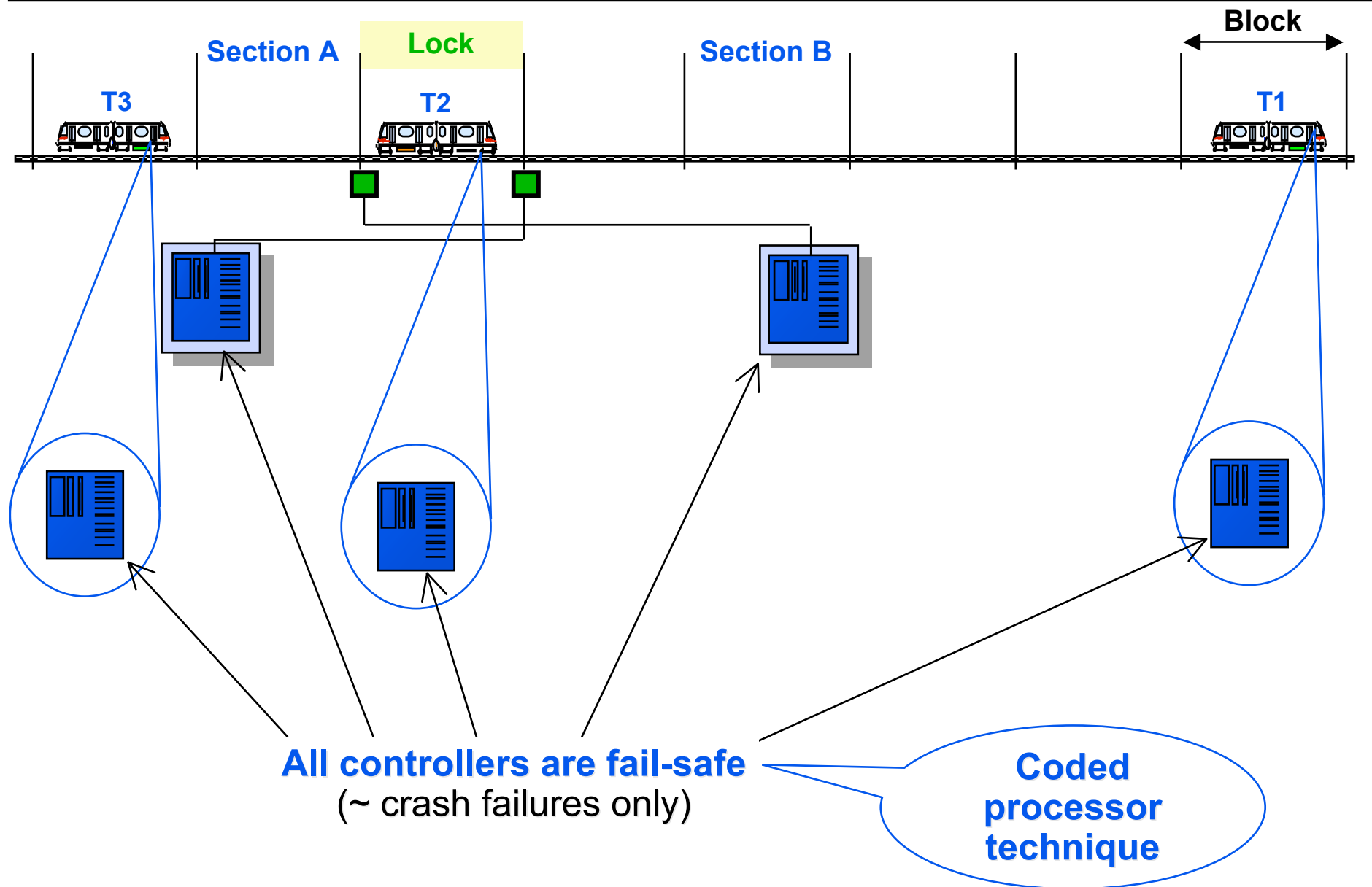
# Automatic Subway System



# Principle of Operation



# Principle of Operation



# Coded Processor Technique (1)

---

## ■ Arithmetic code (principle):

$$\begin{array}{r} 17 \\ + 25 \\ \hline 42 \end{array}$$

# Coded Processor Technique (1)

---

## ■ Arithmetic code (principle):

	17		8 (= 1 + 7)
+	25		7 (= 2 + 5)
<hr/>			
	42		6 (= 4 + 2)

# Coded Processor Technique (1)

---

## ■ Arithmetic code (principle):

$$\begin{array}{r|l} 17 & 8 \text{ (= 1 + 7)} \\ + 25 & 7 \text{ (= 2 + 5)} \\ \hline 42 & 6 \text{ (= 4 + 2)} \end{array}$$



Correct since:  $8 + 7 = 15 \square 6 \text{ (= 1 + 5)}$

# Coded Processor Technique (1)

---

## ■ Arithmetic code (principle):

$$\begin{array}{r|l} 17 & 8 \text{ (= 1 + 7)} \\ + 25 & 7 \text{ (= 2 + 5)} \\ \hline 42 & 6 \text{ (= 4 + 2)} \end{array}$$

↓  
Correct since:  $8 + 7 = 15 \equiv 6 \text{ (= 1 + 5)}$

```
type SAFE_INTEGER is record
```

```
  F: INTEGER; -- functional value
```

```
  C: INTEGER; -- code, i.e., modulo 9 of the functional value
```

```
end record;
```

```
function "+" (x,y : SAFE_INTEGER) return SAFE_INTEGER is
```

```
begin
```

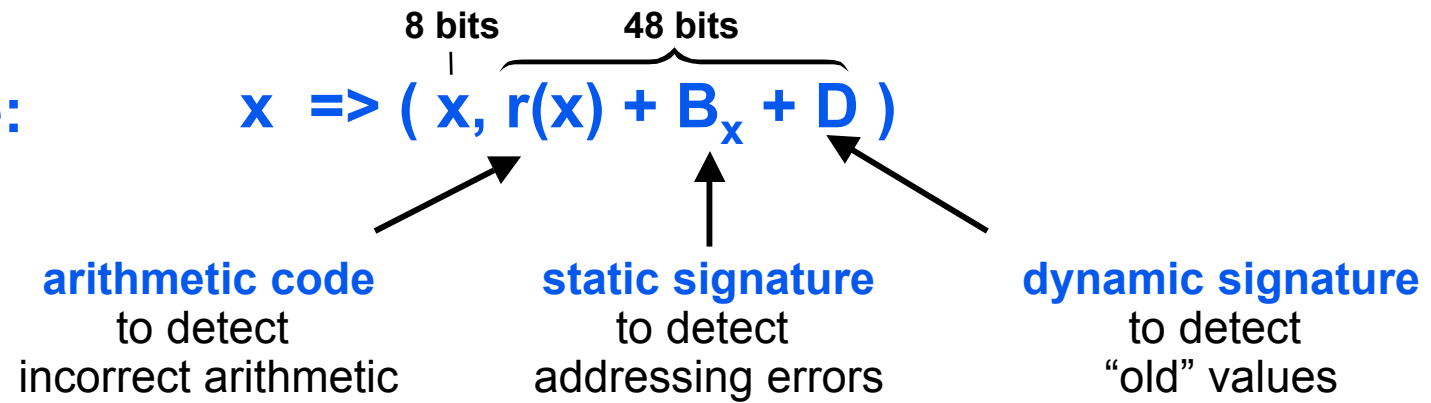
```
  return (x.F + y.F , ( x.C + y.C) mod 9 )
```

```
end "+"
```

# Coded Processor Technique (2)

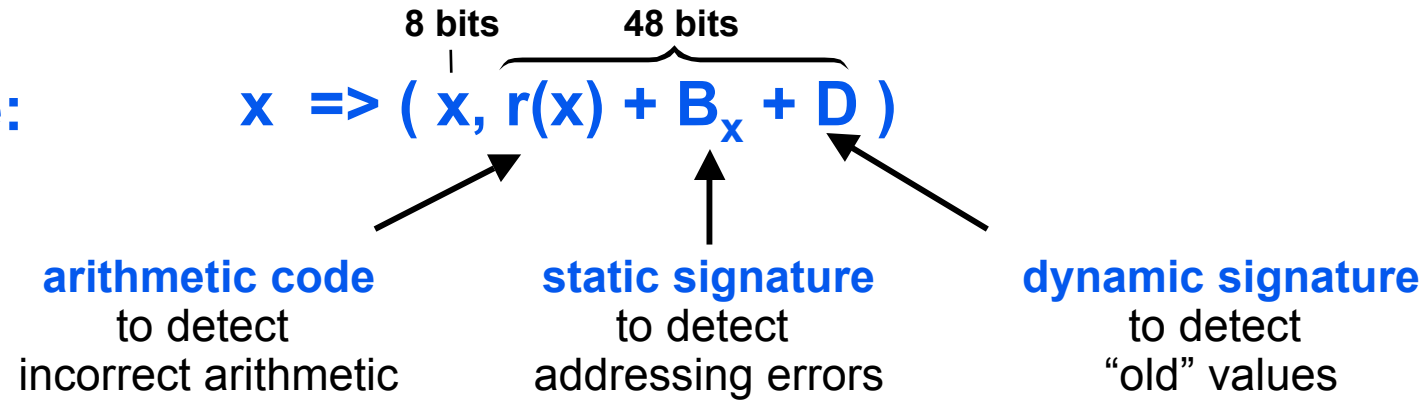
---

## ■ Full code:

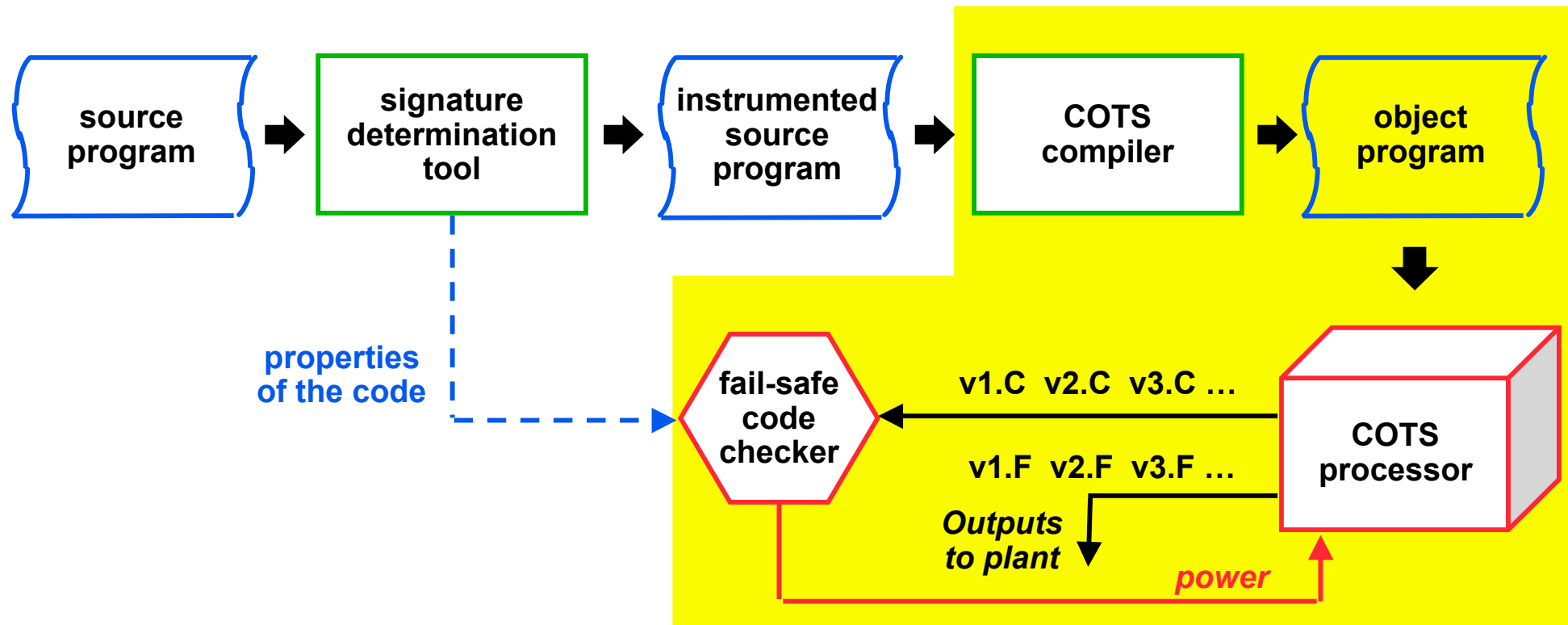


# Coded Processor Technique (2)

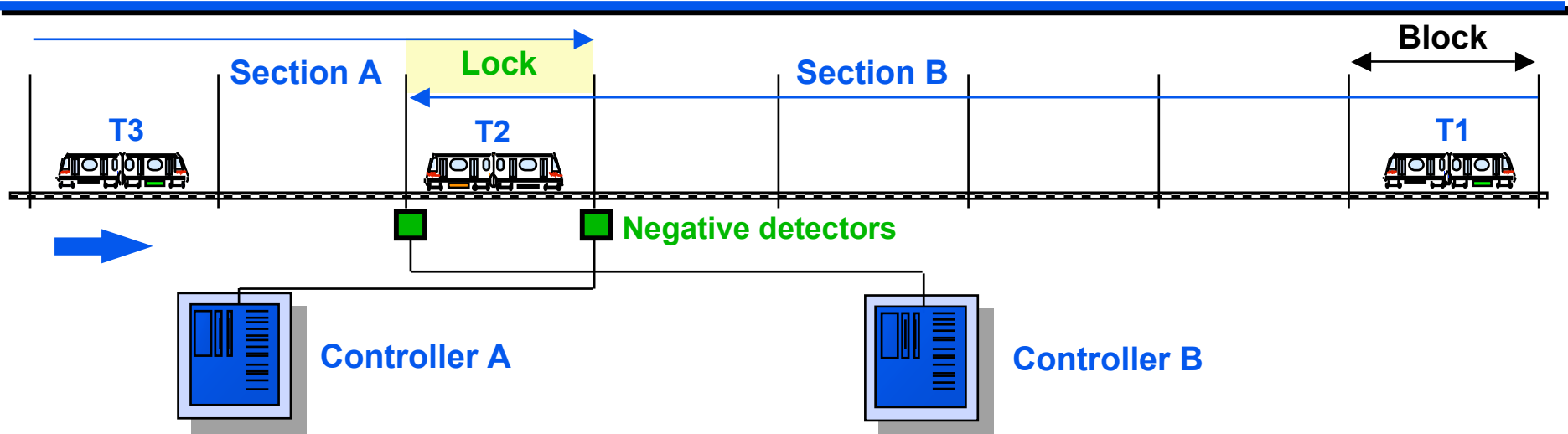
## ■ Full code:



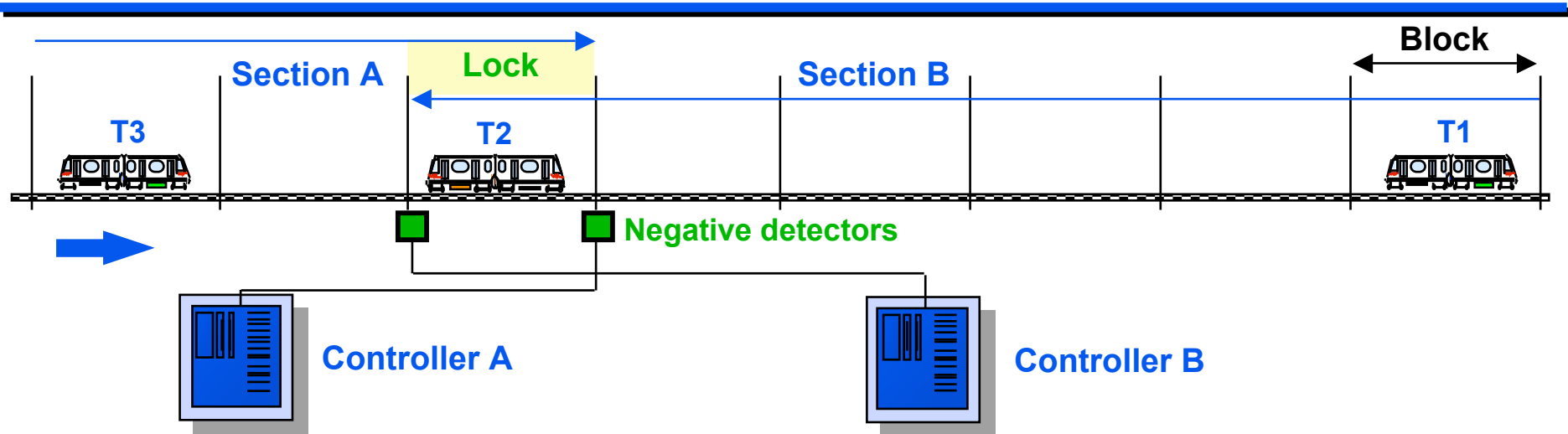
## ■ Source to runtime production chain:



# Principle of Operation



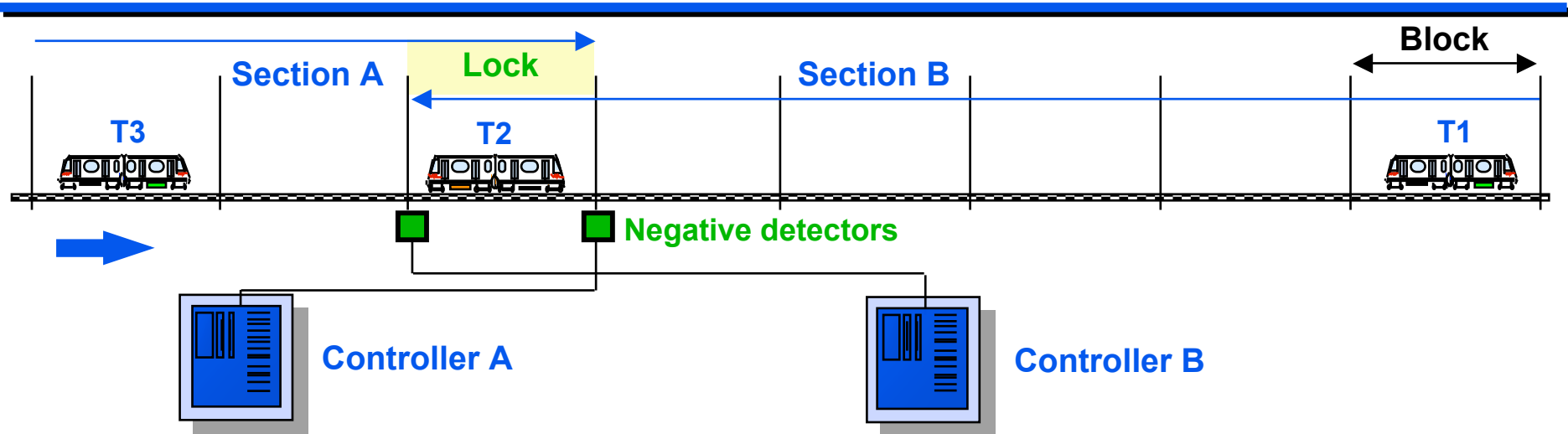
# Principle of Operation



## ■ Target attribution

- Section controllers give each train a “target” - the point up to which it may advance:
  - ➔ *next station*
  - ➔ *block before next train*
  - ➔ *exit point of inter-section Lock*

# Principle of Operation



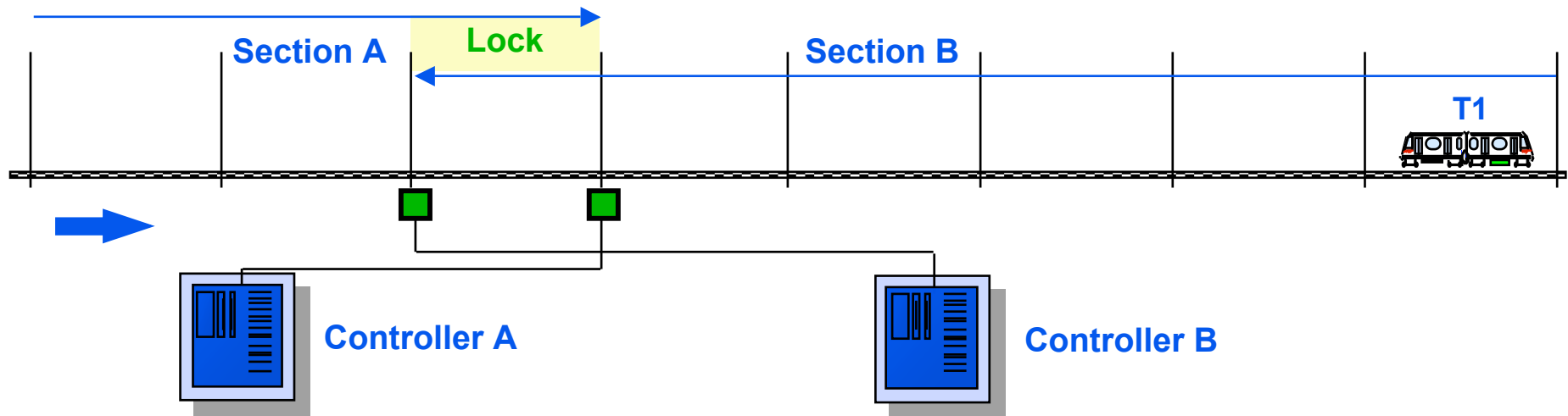
## ■ Target attribution

- Section controllers give each train a “target” - the point up to which it may advance:
  - ➔ *next station*
  - ➔ *block before next train*
  - ➔ *exit point of inter-section Lock*

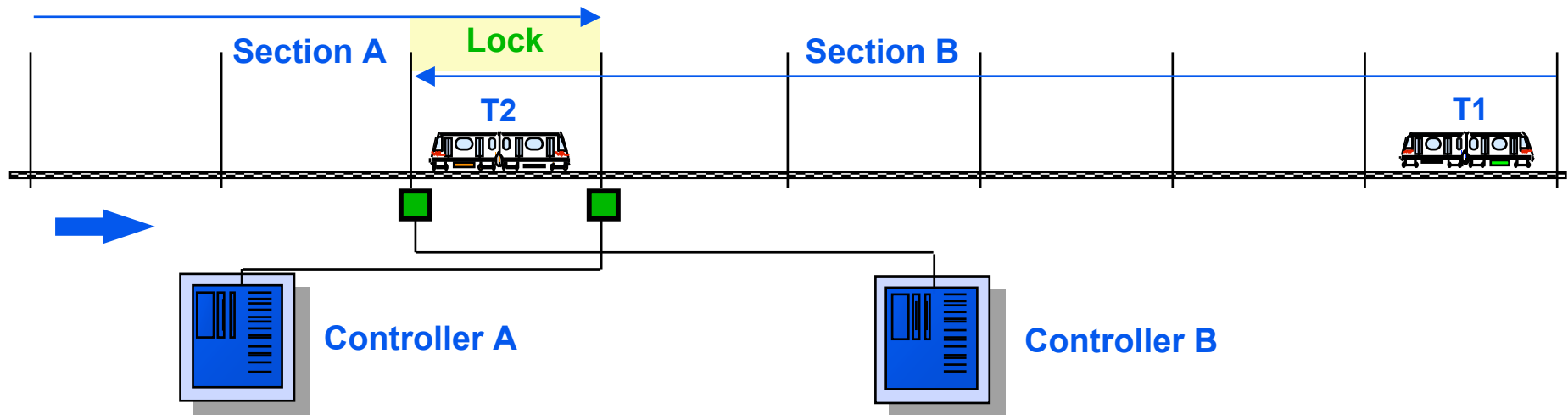
## ■ Train handover from one section to the next (via the Lock)

- Controller B detects, interrogates and registers trains entering the Lock
- Controller A detects and de-registers trains leaving the Lock

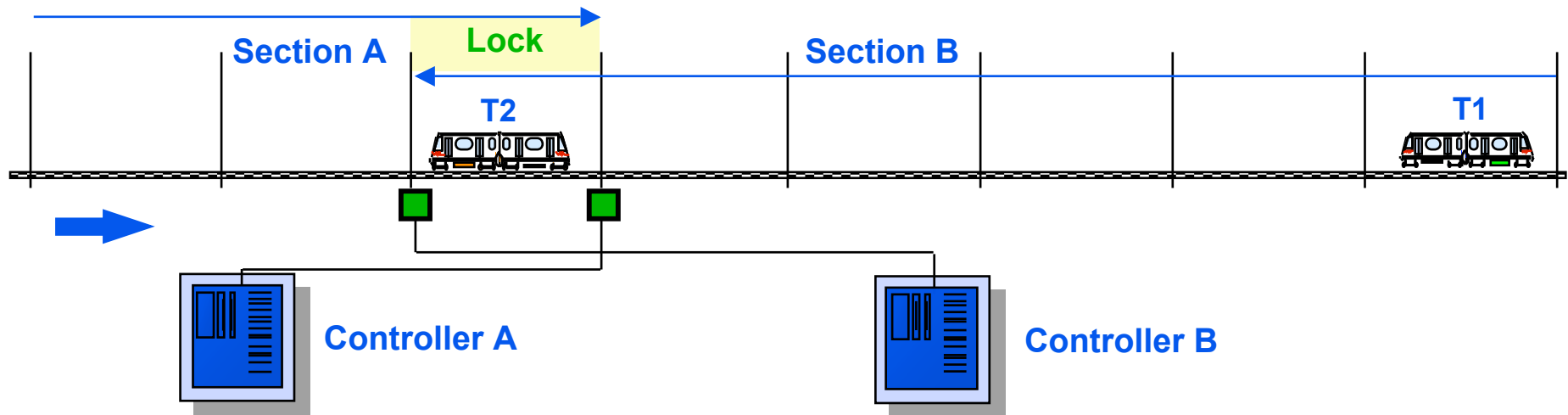
# Handover Scenario without Redundancy



# Handover Scenario without Redundancy

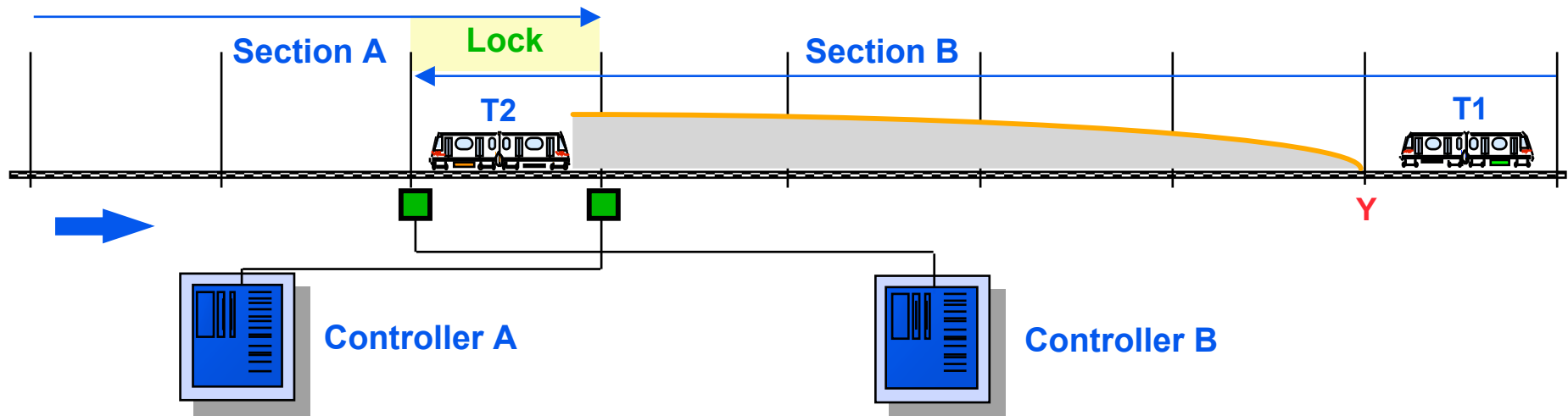


# Handover Scenario without Redundancy



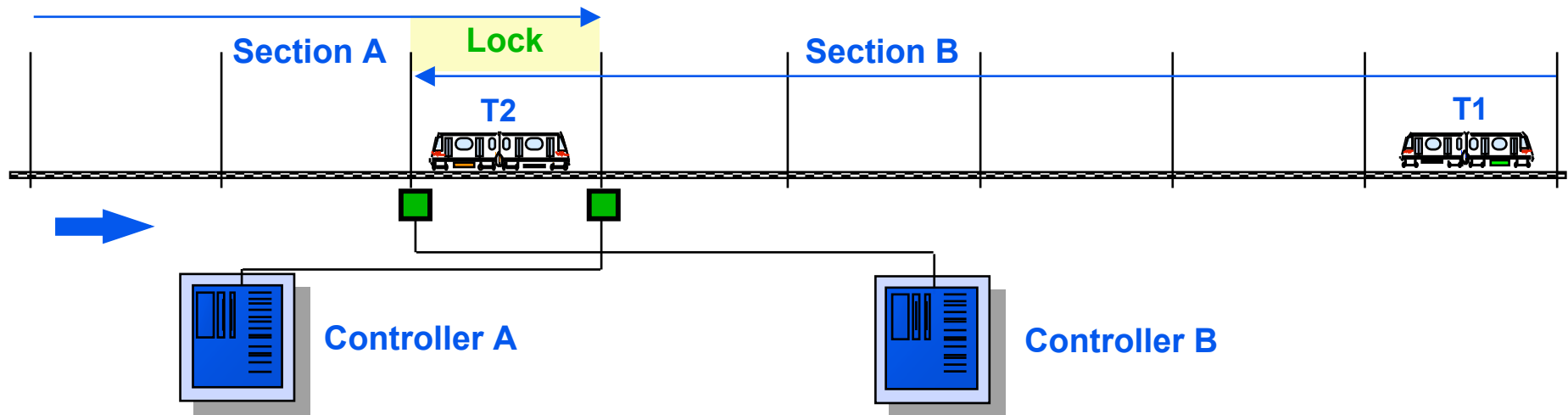
- Controller B registers train T2 and assigns it the target Y

# Handover Scenario without Redundancy



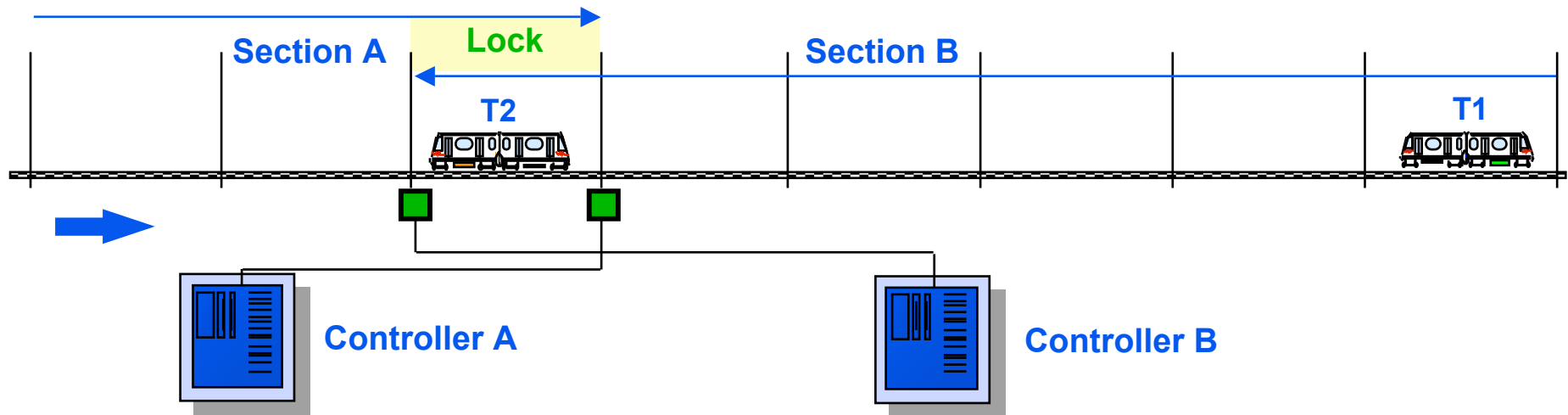
- Controller B registers train T2 and assigns it the target Y

# Handover Scenario without Redundancy



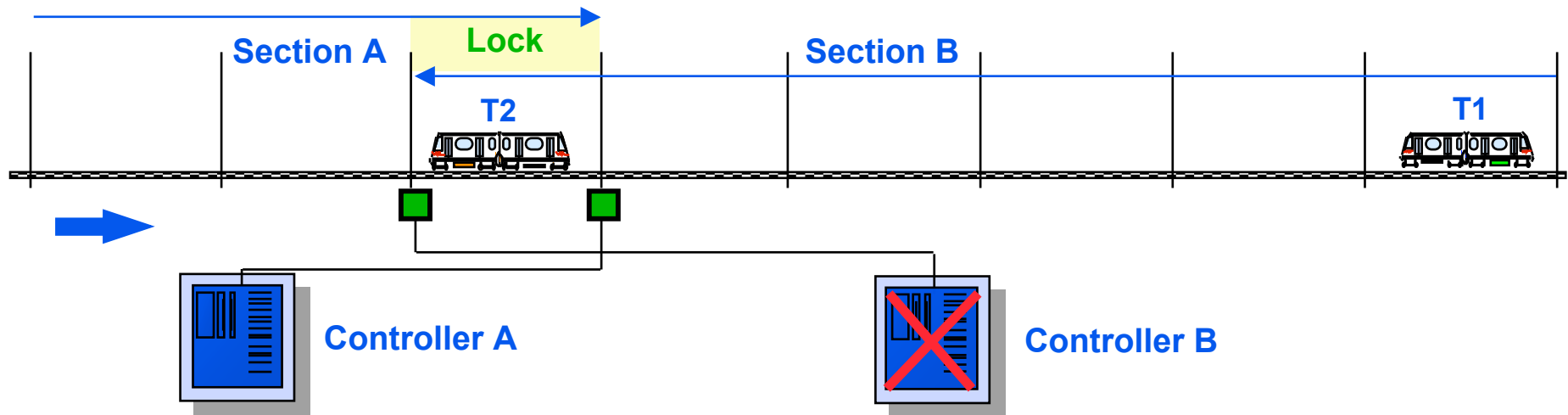
- Controller B registers train T2 and assigns it the target Y

# Handover Scenario without Redundancy



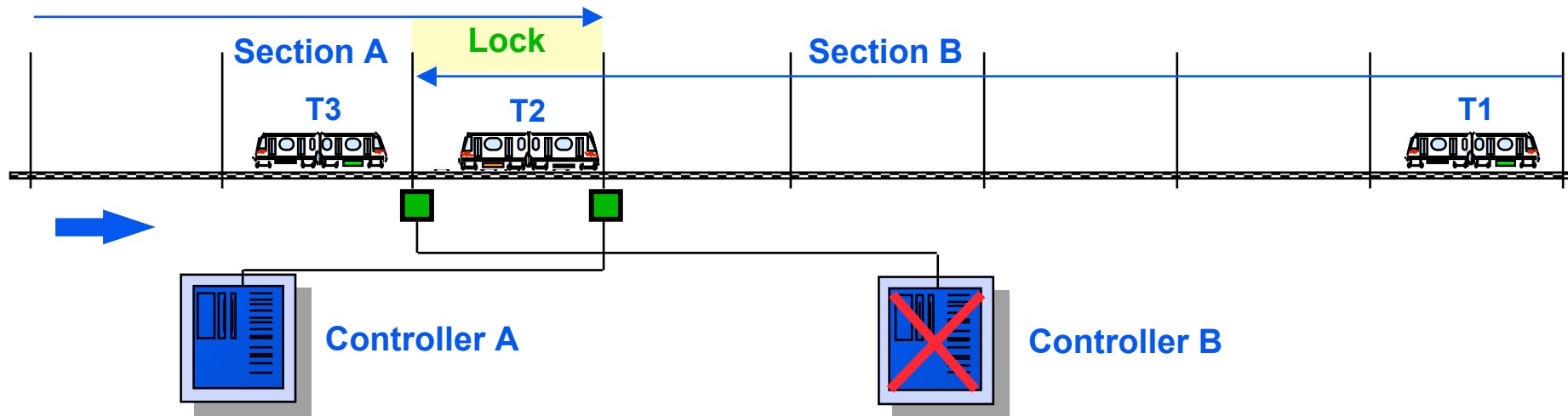
- Controller B registers train T2 and assigns it the target Y  
    ➔ *if controller B does not register T2*

# Handover Scenario without Redundancy



- **Controller B registers train T2 and assigns it the target Y**
  - ➔ *if controller B does not register T2*
  - ➔ *if controller B fails before assigning the target Y*

# Handover Scenario without Redundancy



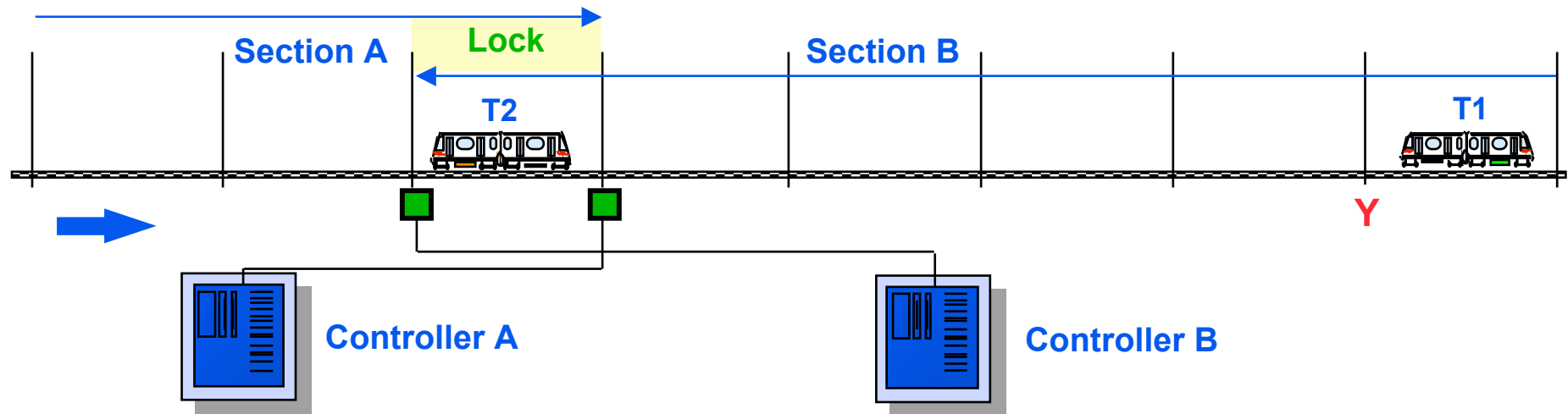
- Controller B registers train T2 and assigns it the target Y

➔ if controller B does not register T2

➔ if controller B fails before assigning the target Y

*T2 stops at exit of Lock  
(its last target)*

# Handover Scenario without Redundancy



- **Controller B registers train T2 and assigns it the target Y**

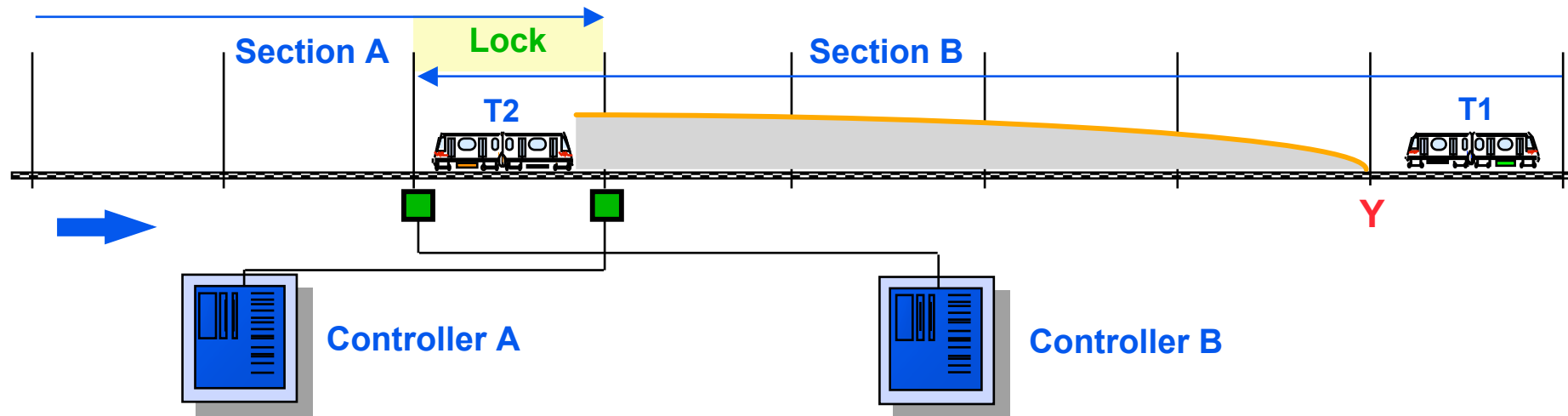
- *if controller B does not register T2*

- *if controller B fails before assigning the target Y*

- *if controller B fails after assigning the target Y*

*T2 stops at exit of Lock  
(its last target)*

# Handover Scenario without Redundancy



- **Controller B registers train T2 and assigns it the target Y**

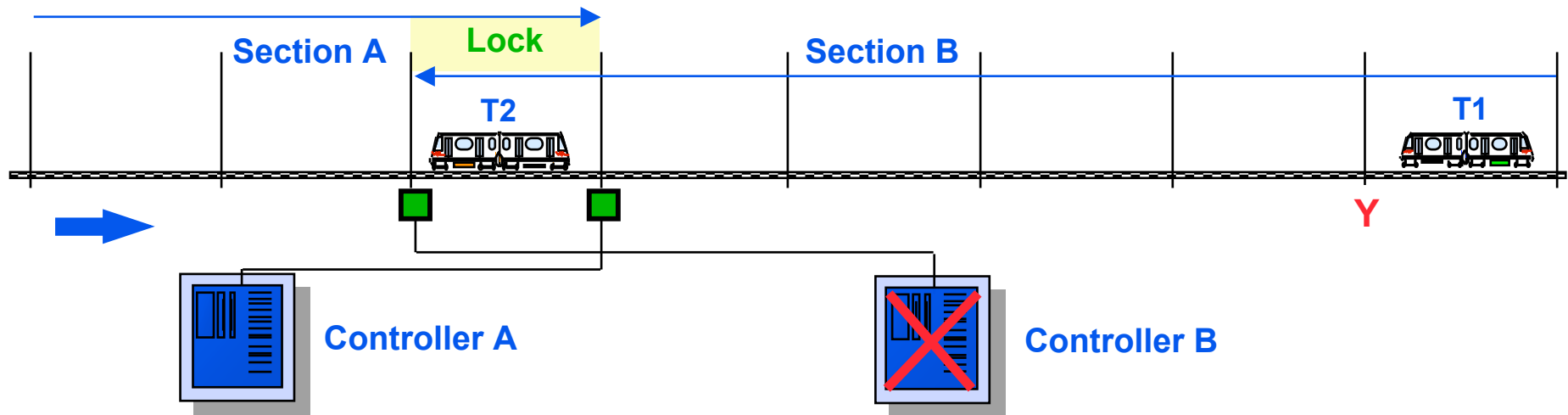
- *if controller B does not register T2*

- *if controller B fails before assigning the target Y*

- *if controller B fails after assigning the target Y*

*T2 stops at exit of Lock  
(its last target)*

# Handover Scenario without Redundancy



- **Controller B registers train T2 and assigns it the target Y**

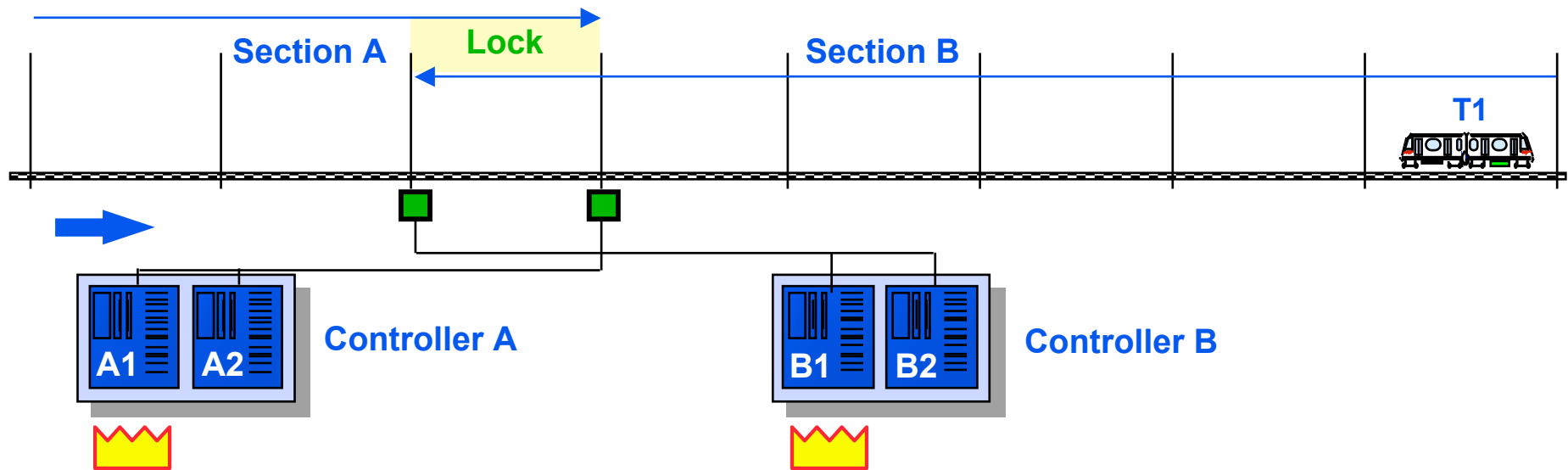
- *if controller B does not register T2*

- *if controller B fails before assigning the target Y*

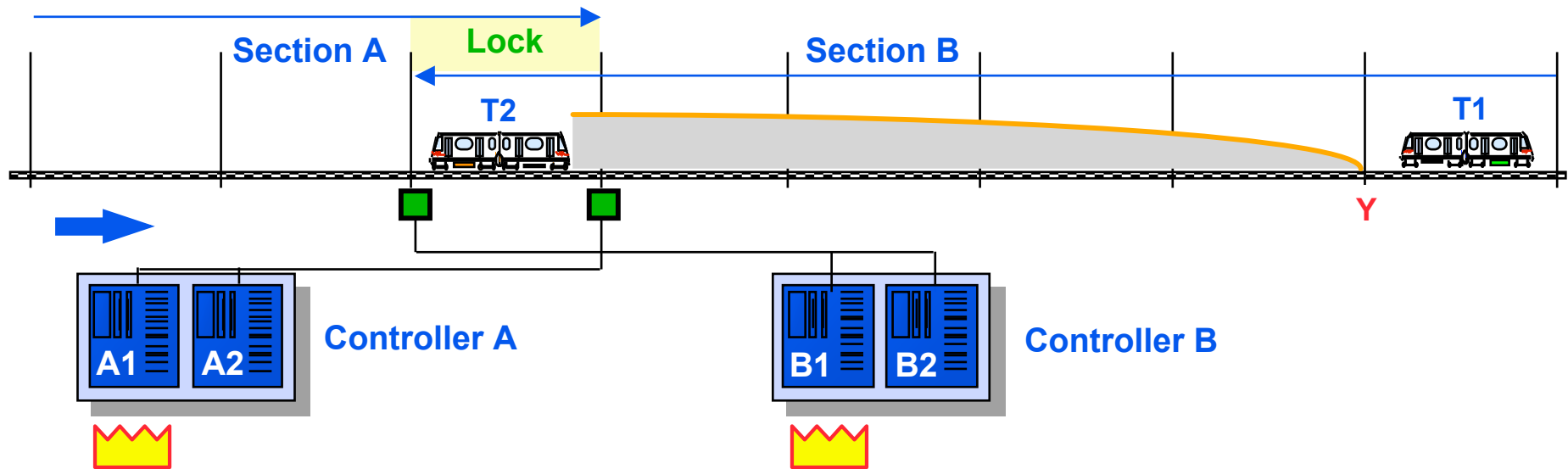
- *if controller B fails after assigning the target Y*

*T2 stops at exit of Lock  
(its last target)*

# Handover Scenario with Redundancy (1)

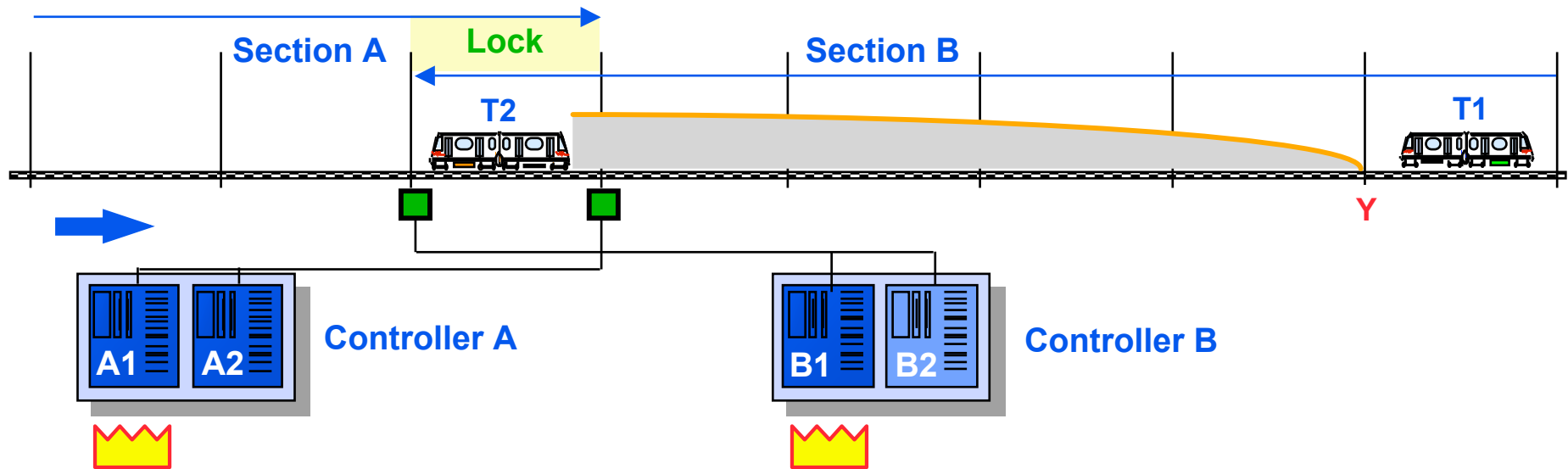


# Handover Scenario with Redundancy (1)



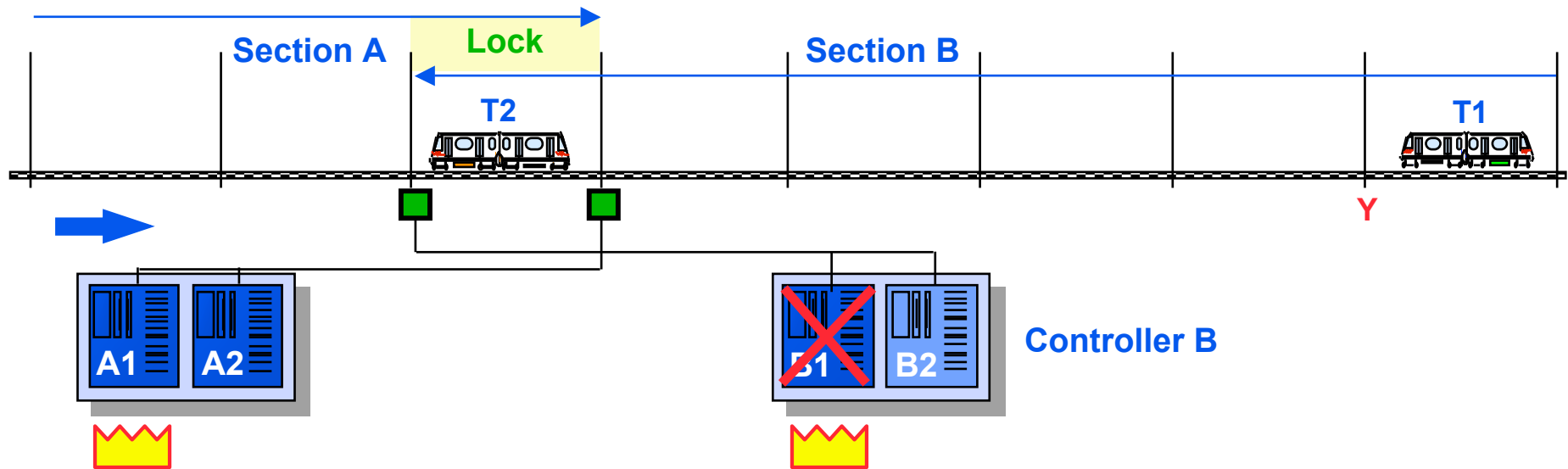
- Unit B1 (primary) registers train T2 and assigns it the target Y

# Handover Scenario with Redundancy (1)



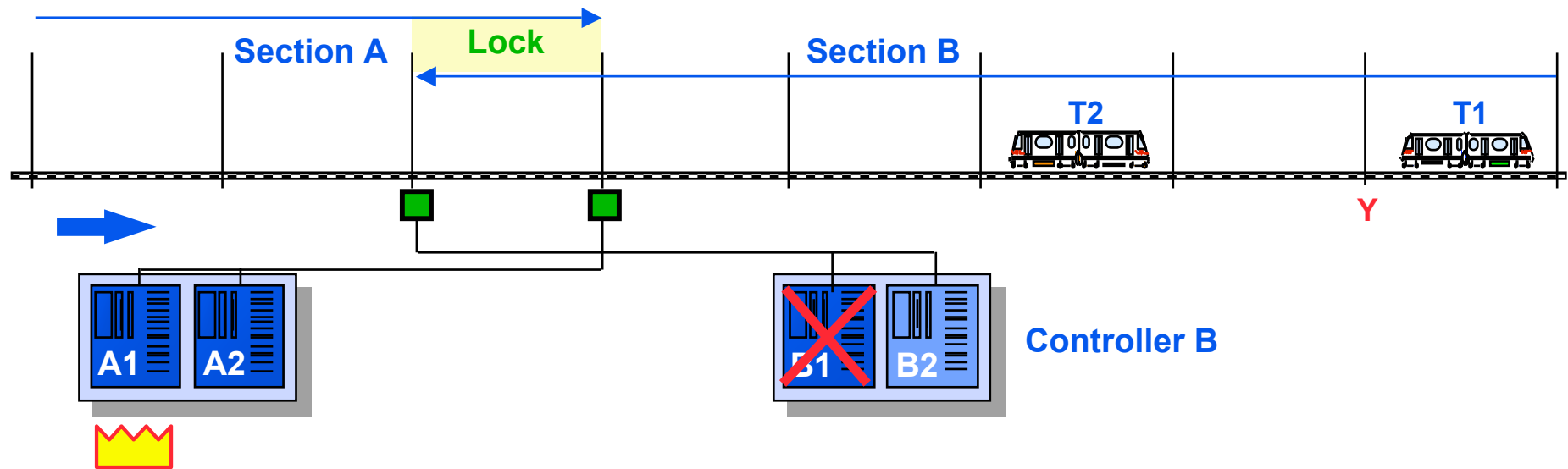
- Unit B1 (primary) registers train T2 and assigns it the target Y
- **Unit B2 (secondary) does not register T2: B1 and B2 have become inconsistent**

# Handover Scenario with Redundancy (2)



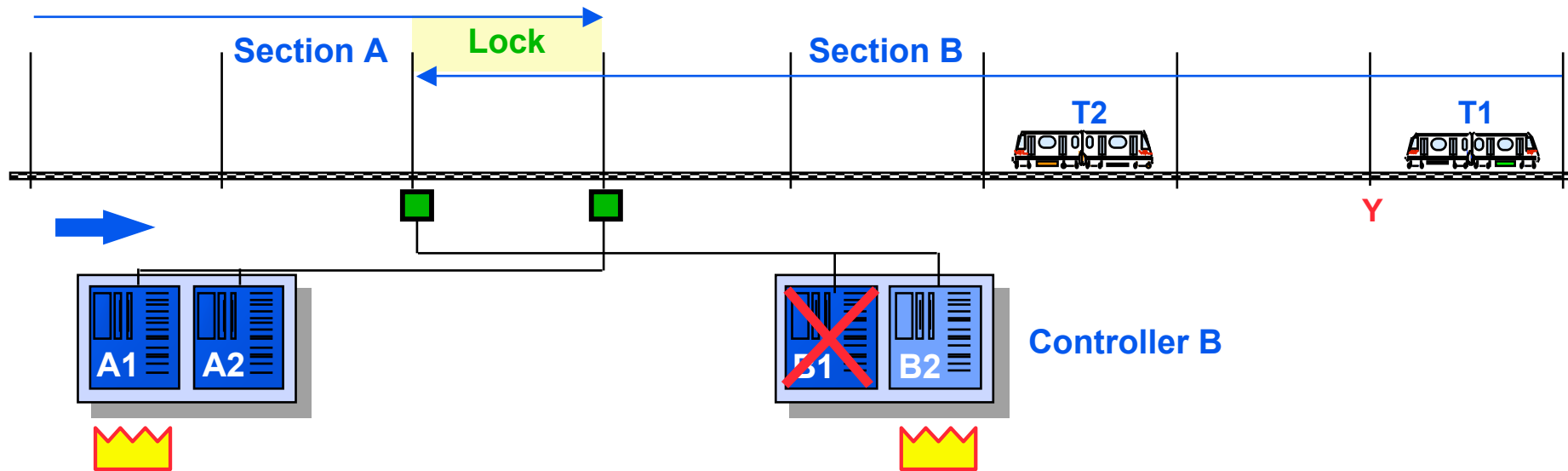
- Unit B1 (primary) registers train T2 and assigns it the target Y
- Unit B2 (secondary) does not register T2: B1 and B2 have become inconsistent
- Unit B1 fails after assigning the target Y

# Handover Scenario with Redundancy (2)



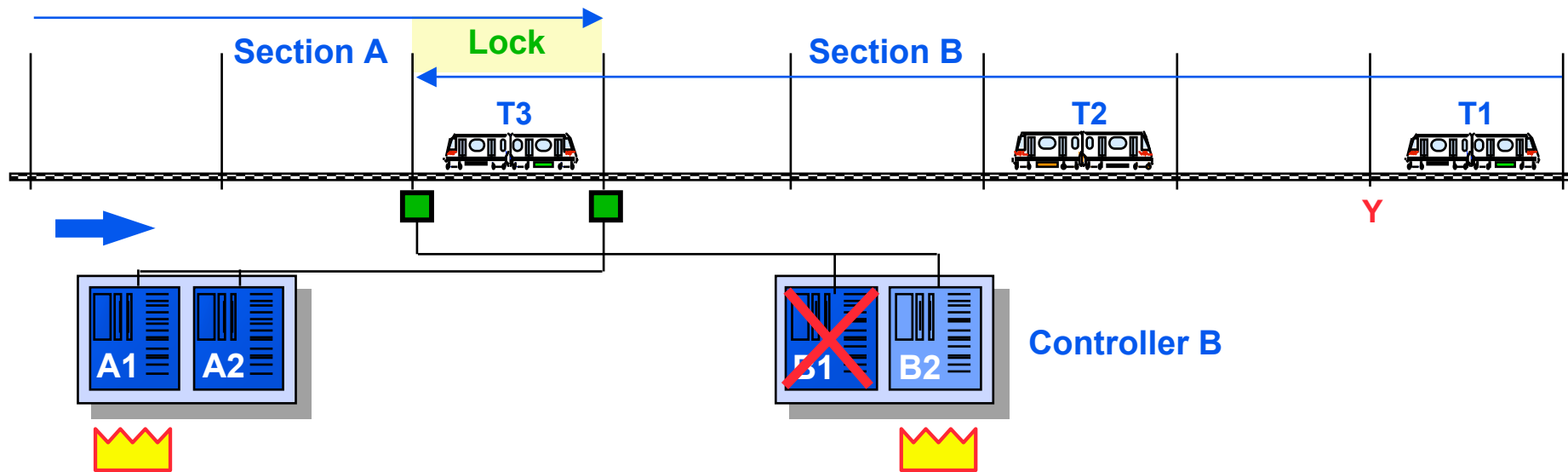
- Unit B1 (primary) registers train T2 and assigns it the target Y
- **Unit B2 (secondary) does not register T2:** B1 and B2 have become inconsistent
- **Unit B1 fails after assigning the target Y**
- **Unit B2 becomes primary**

# Handover Scenario with Redundancy (2)



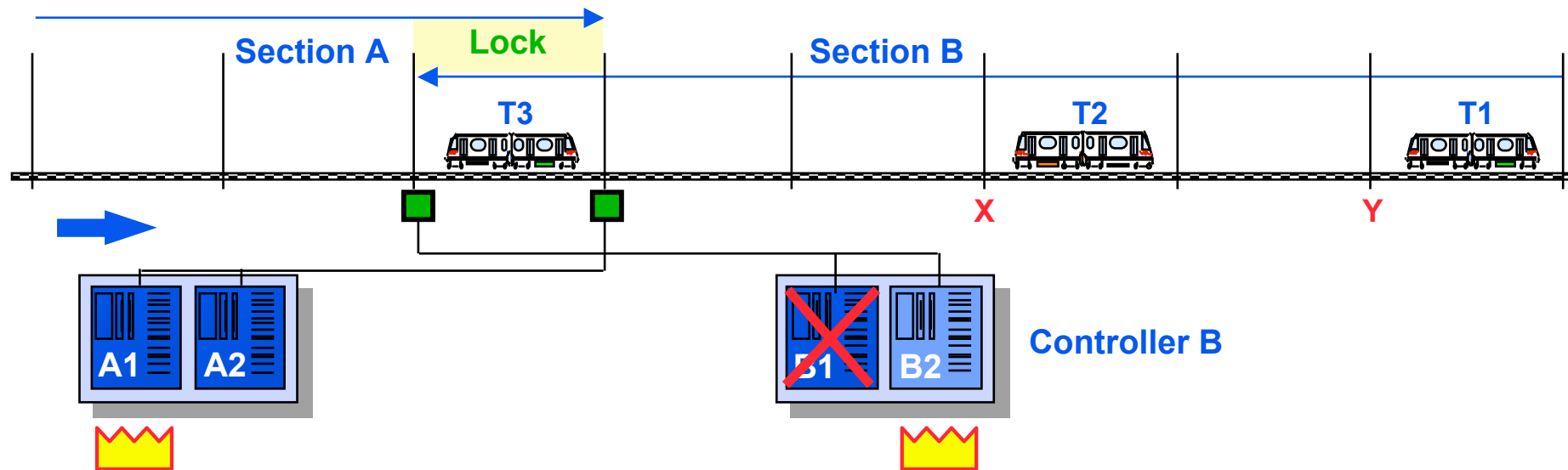
- Unit B1 (primary) registers train T2 and assigns it the target Y
- Unit B2 (secondary) does not register T2: B1 and B2 have become inconsistent
- Unit B1 fails after assigning the target Y
- Unit B2 becomes primary
- Train T3 enters the Lock

# Handover Scenario with Redundancy (2)



- Unit B1 (primary) registers train T2 and assigns it the target Y
- **Unit B2 (secondary) does not register T2:** B1 and B2 have become inconsistent
- **Unit B1 fails after assigning the target Y**
- Unit B2 becomes primary
- Train T3 enters the Lock
- **Unit B2 registers train T3 and assigns it the target Y instead of X because it is not aware of T2**

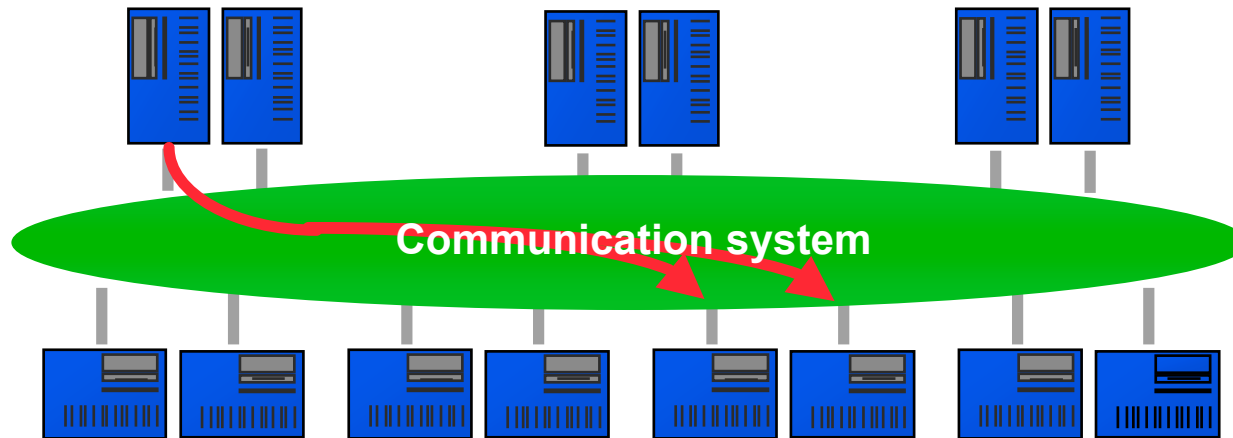
# Handover Scenario with Redundancy (2)



- Unit B1 (primary) registers train T2 and assigns it the target Y
- **Unit B2 (secondary) does not register T2:** B1 and B2 have become inconsistent
- **Unit B1 fails after assigning the target Y**
- Unit B2 becomes primary
- Train T3 enters the Lock
- Unit B2 registers train T3 and assigns it the target Y *instead of X* because it is not aware of T2
- **Train T3 advances towards Y, through point X ...**

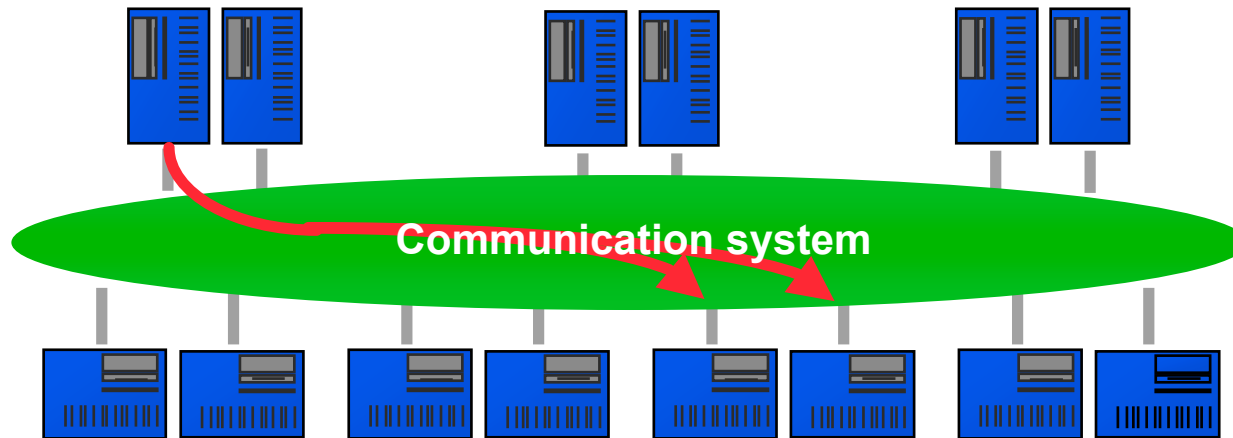
# Problem Statement

---

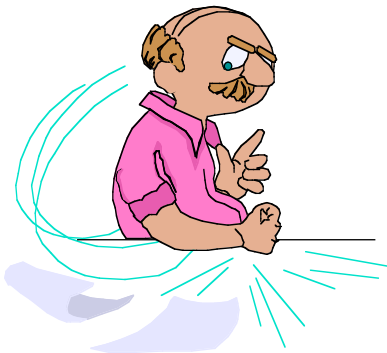


# Problem Statement

---

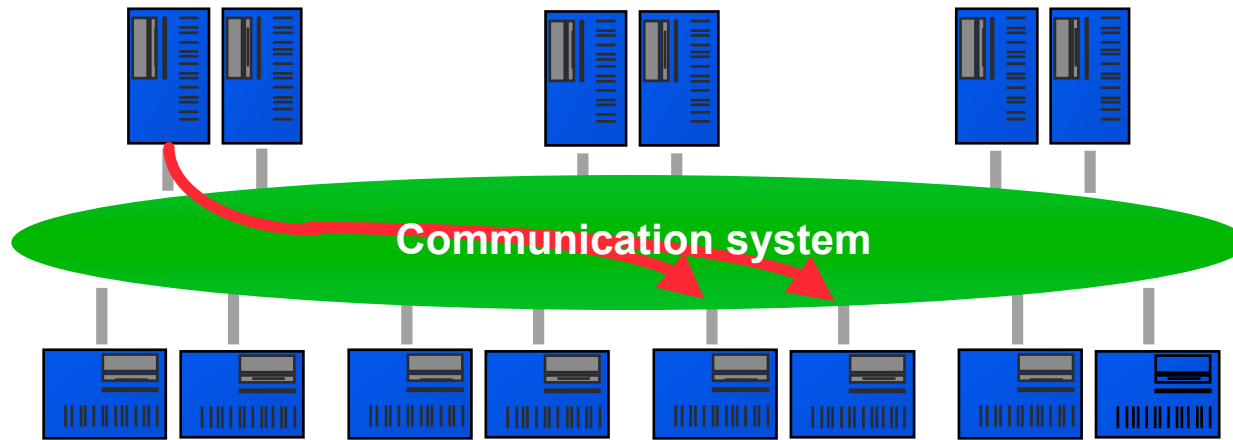


Inconsistency of  
redundant unit states  
causes safety problems

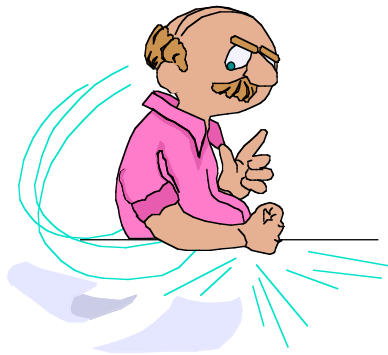


# Problem Statement

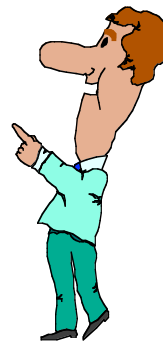
---



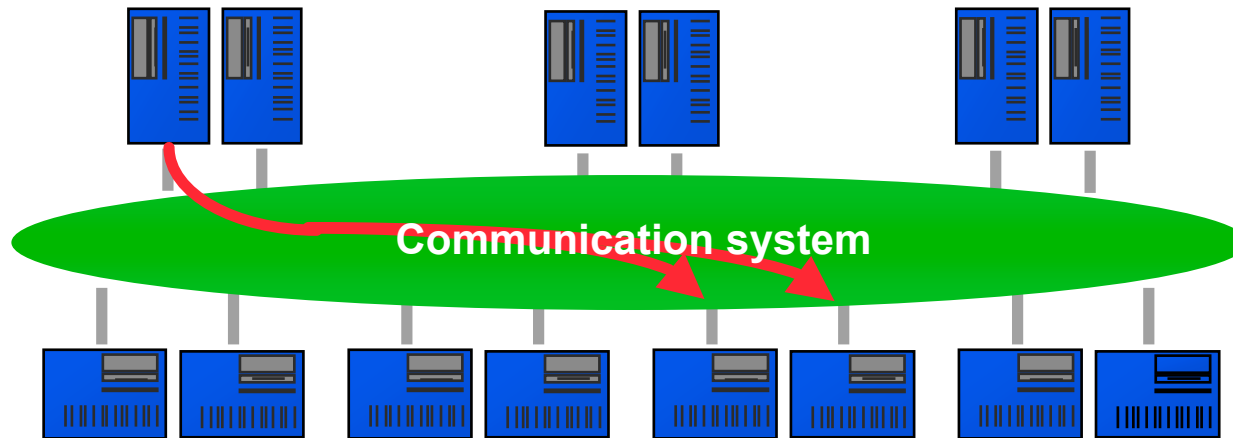
Inconsistency of  
redundant unit states  
causes safety problems



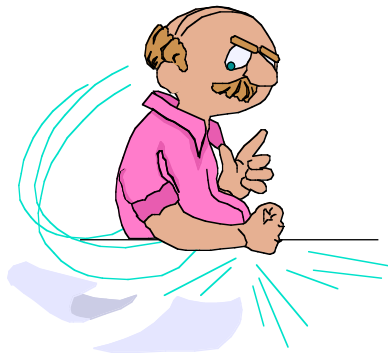
*Use an atomic  
multicast protocol !*



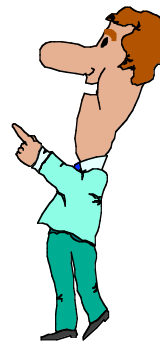
# Problem Statement



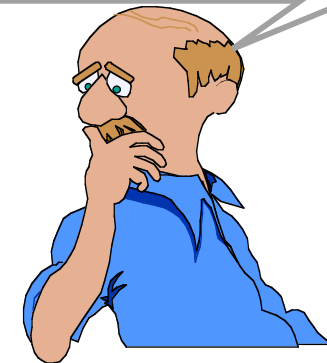
Inconsistency of  
redundant unit states  
causes safety problems



*Use an atomic  
multicast protocol !*

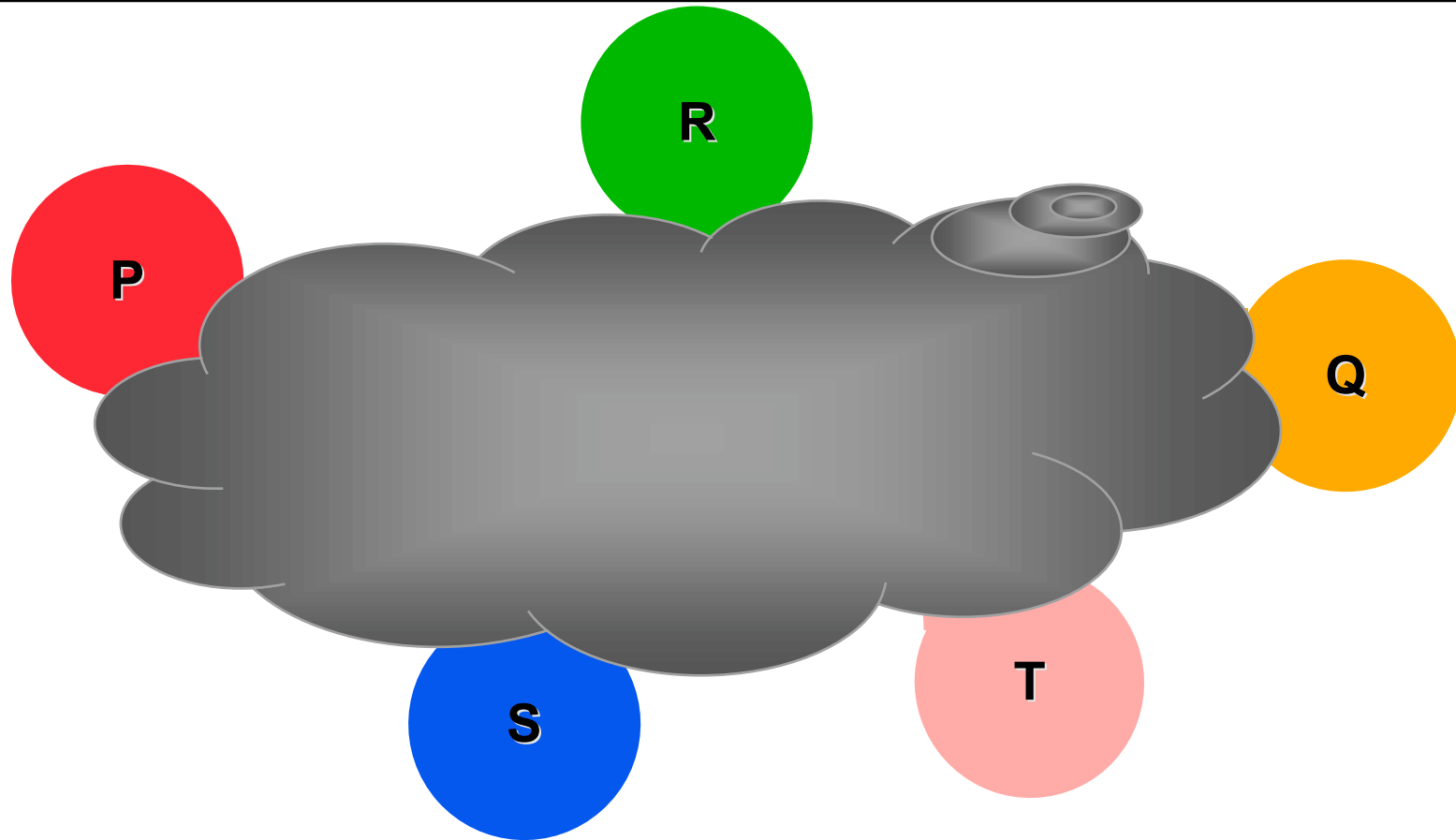


Yes, of course, but what  
distributed system  
model can be *safely*  
assumed?



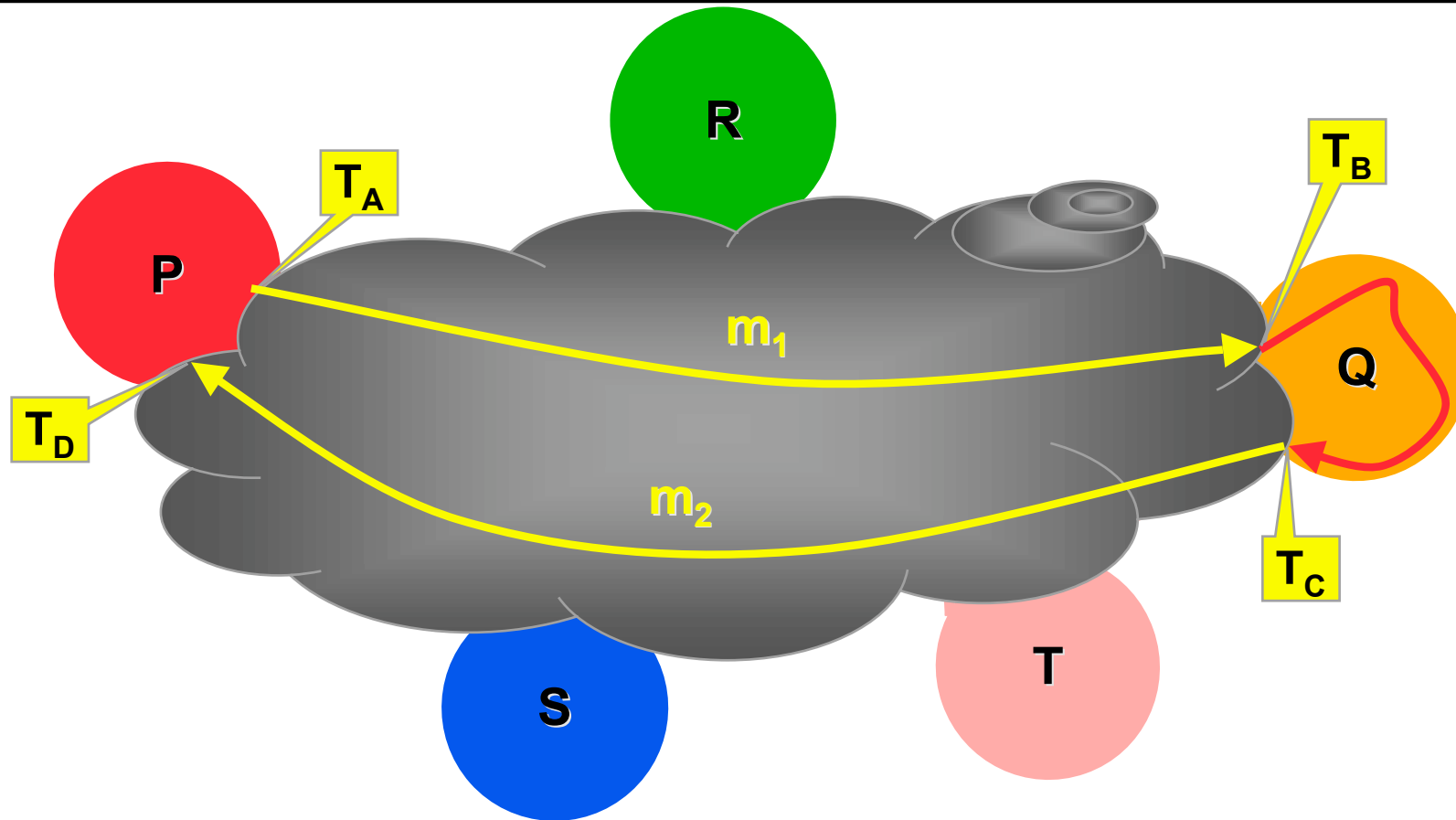
# Assumptions: Timing Models (1)

---



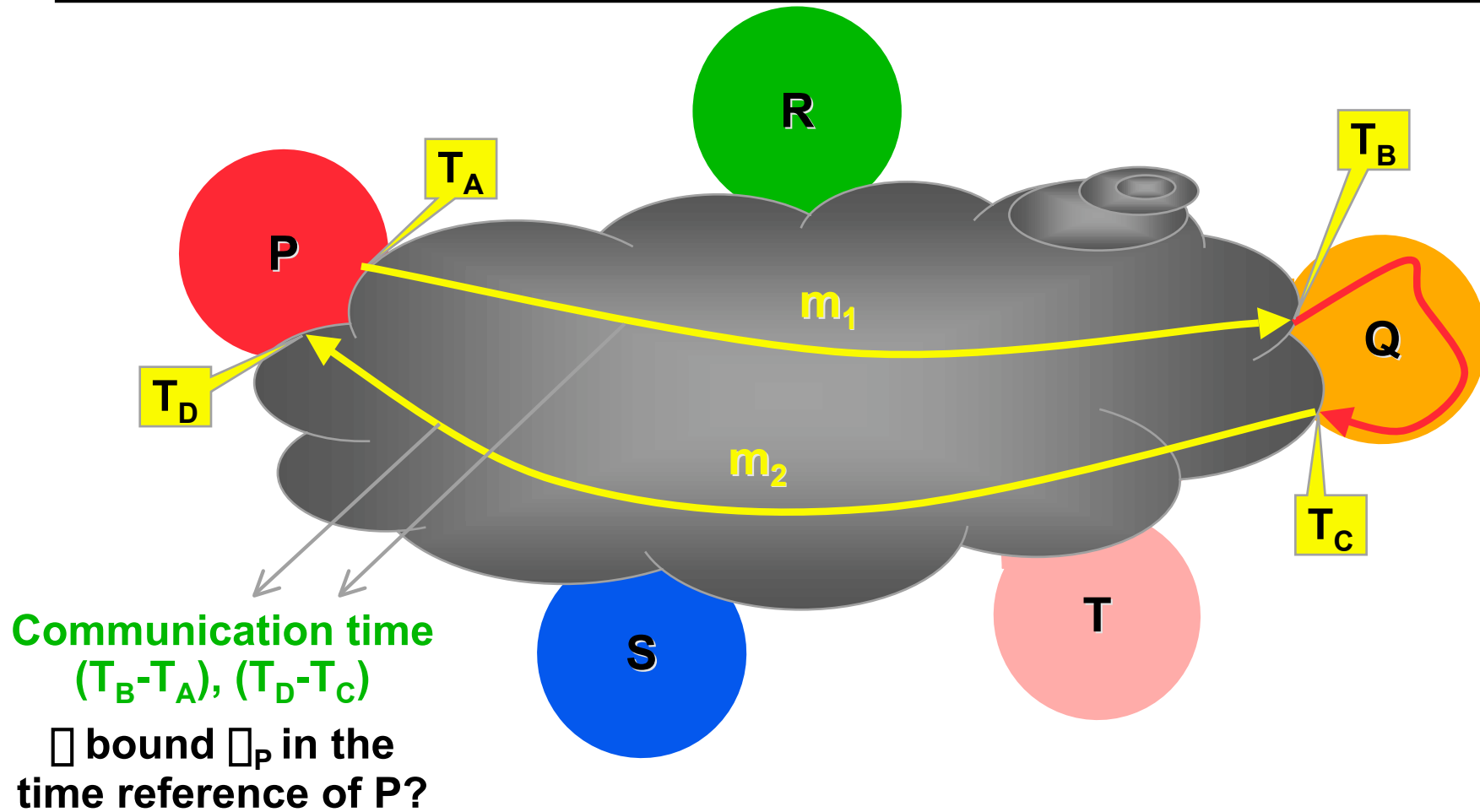
# Assumptions: Timing Models (1)

---

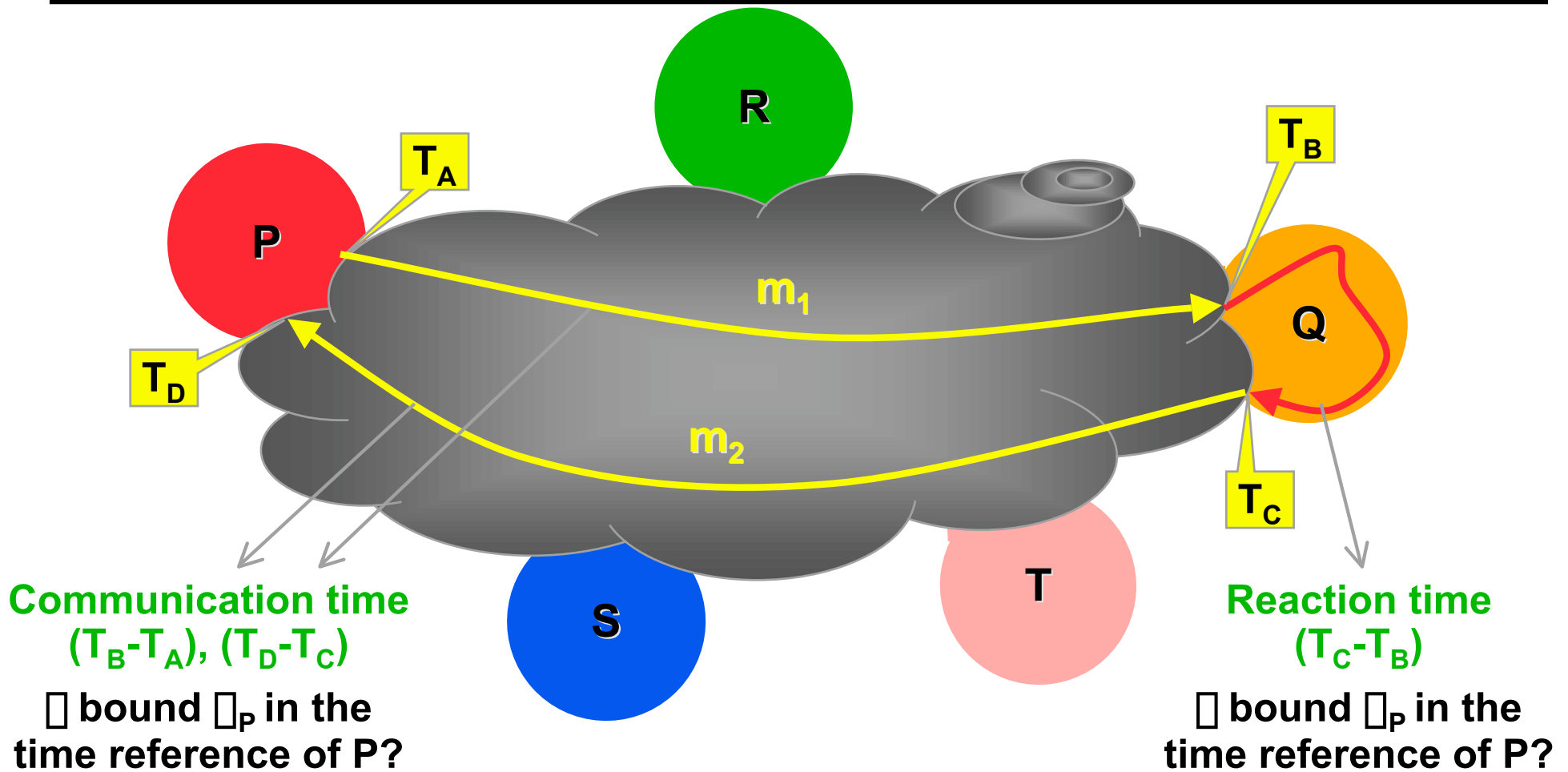


# Assumptions: Timing Models (1)

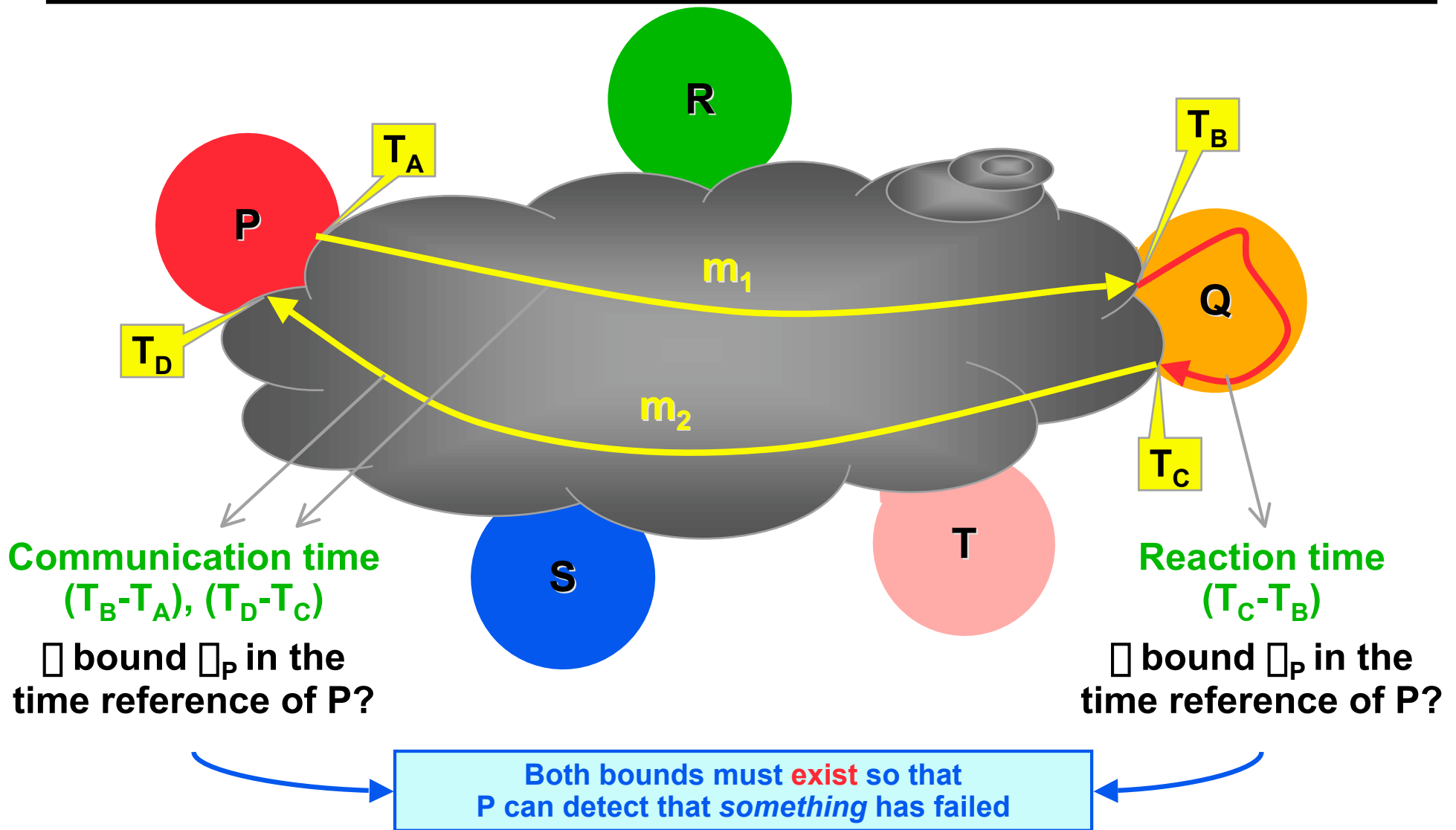
---



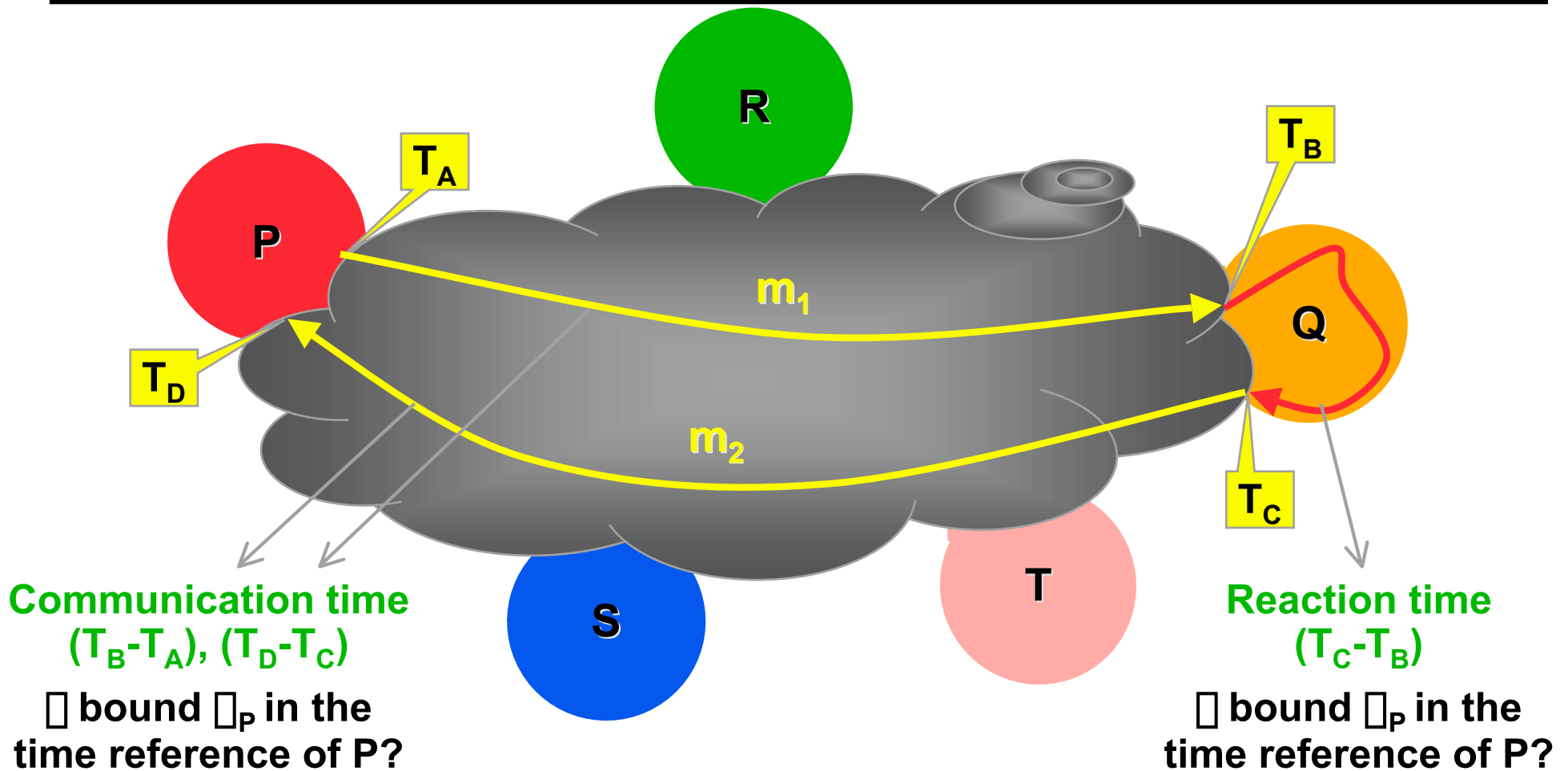
# Assumptions: Timing Models (1)



# Assumptions: Timing Models (1)



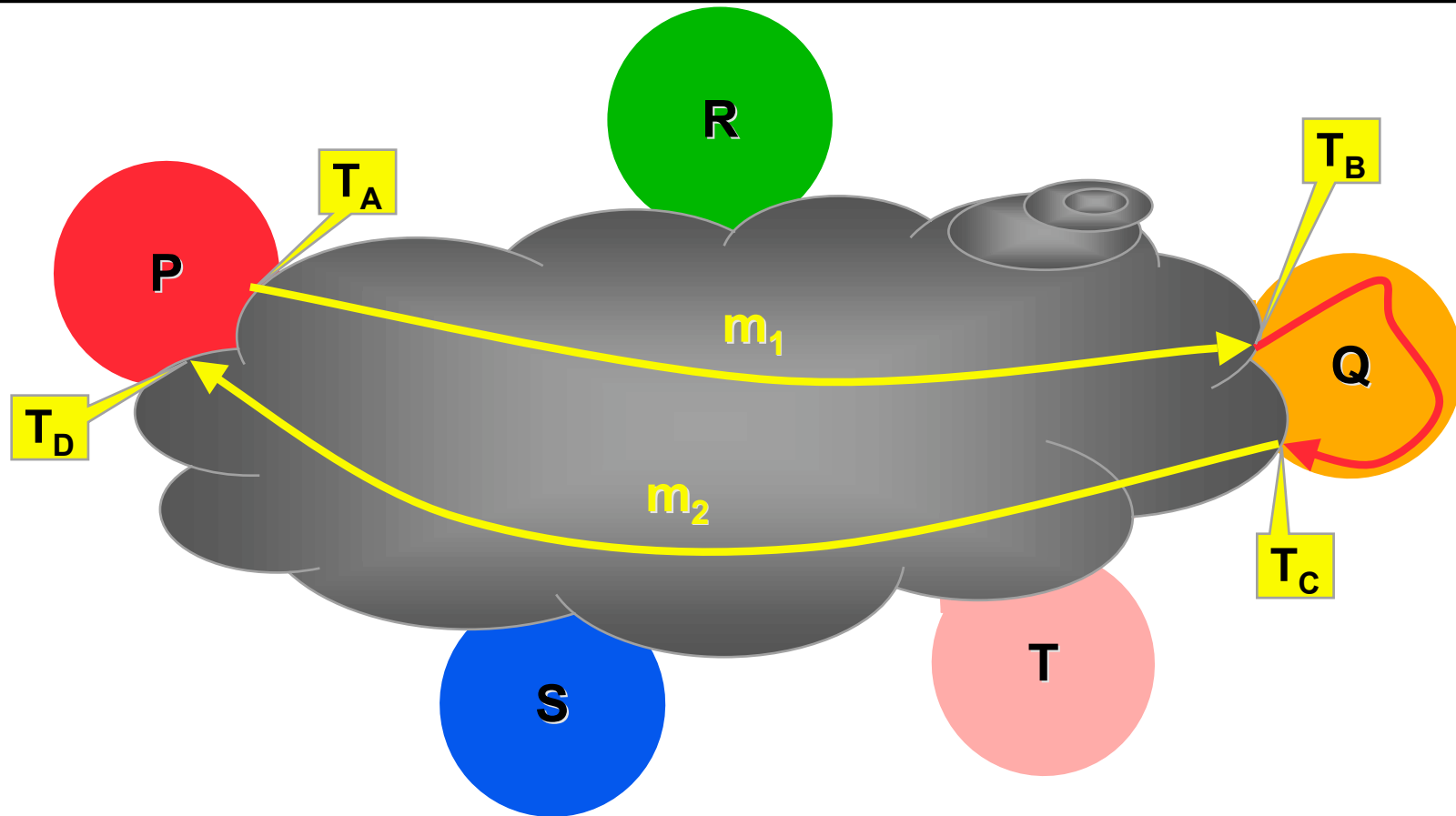
# Assumptions: Timing Models (1)



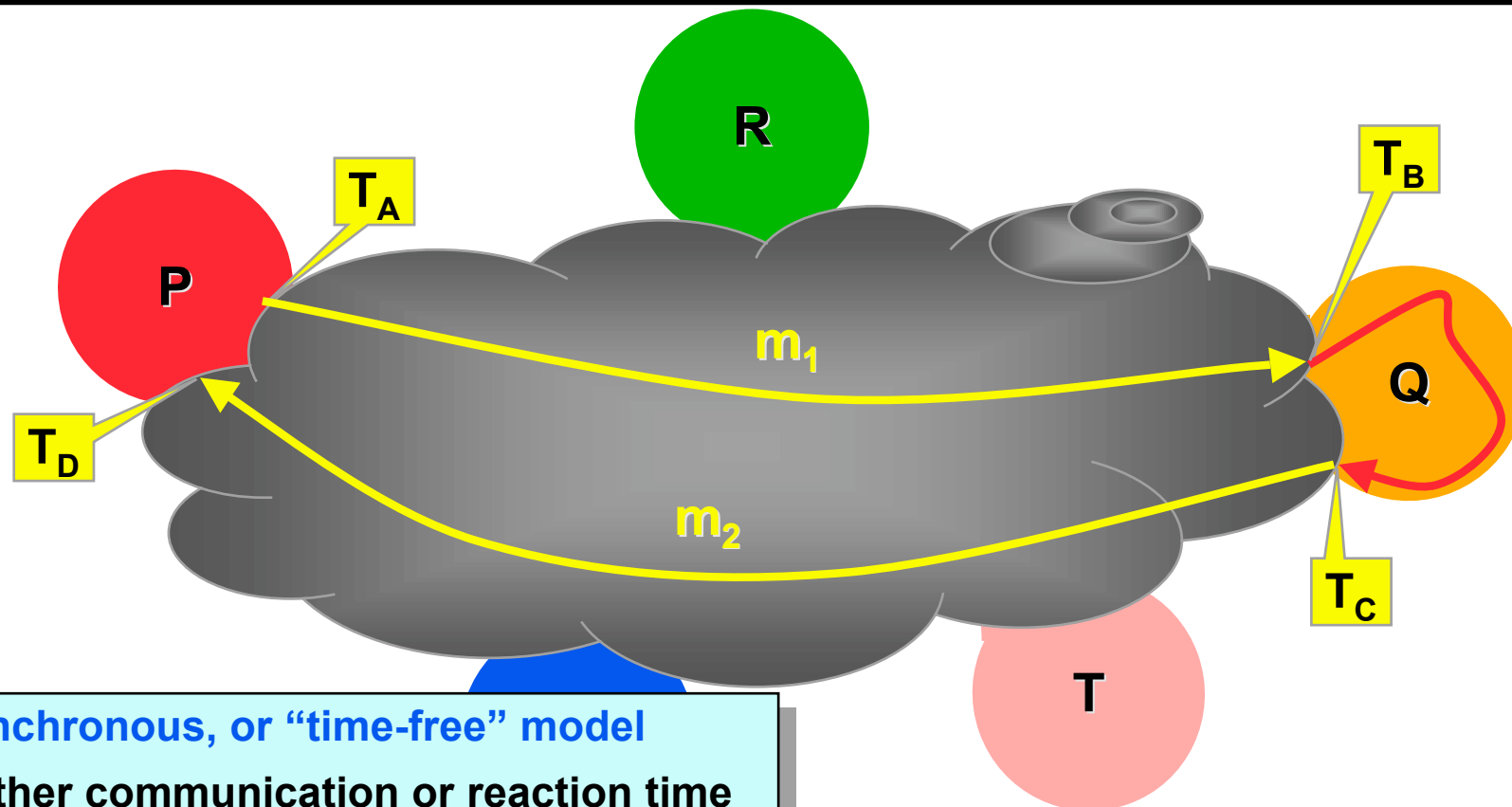
- Both bounds must **exist** so that P can detect that *something* has failed
- One bound must be **guaranteed** so that P can decide *what* has failed

# Assumptions: Timing Models (2)

---



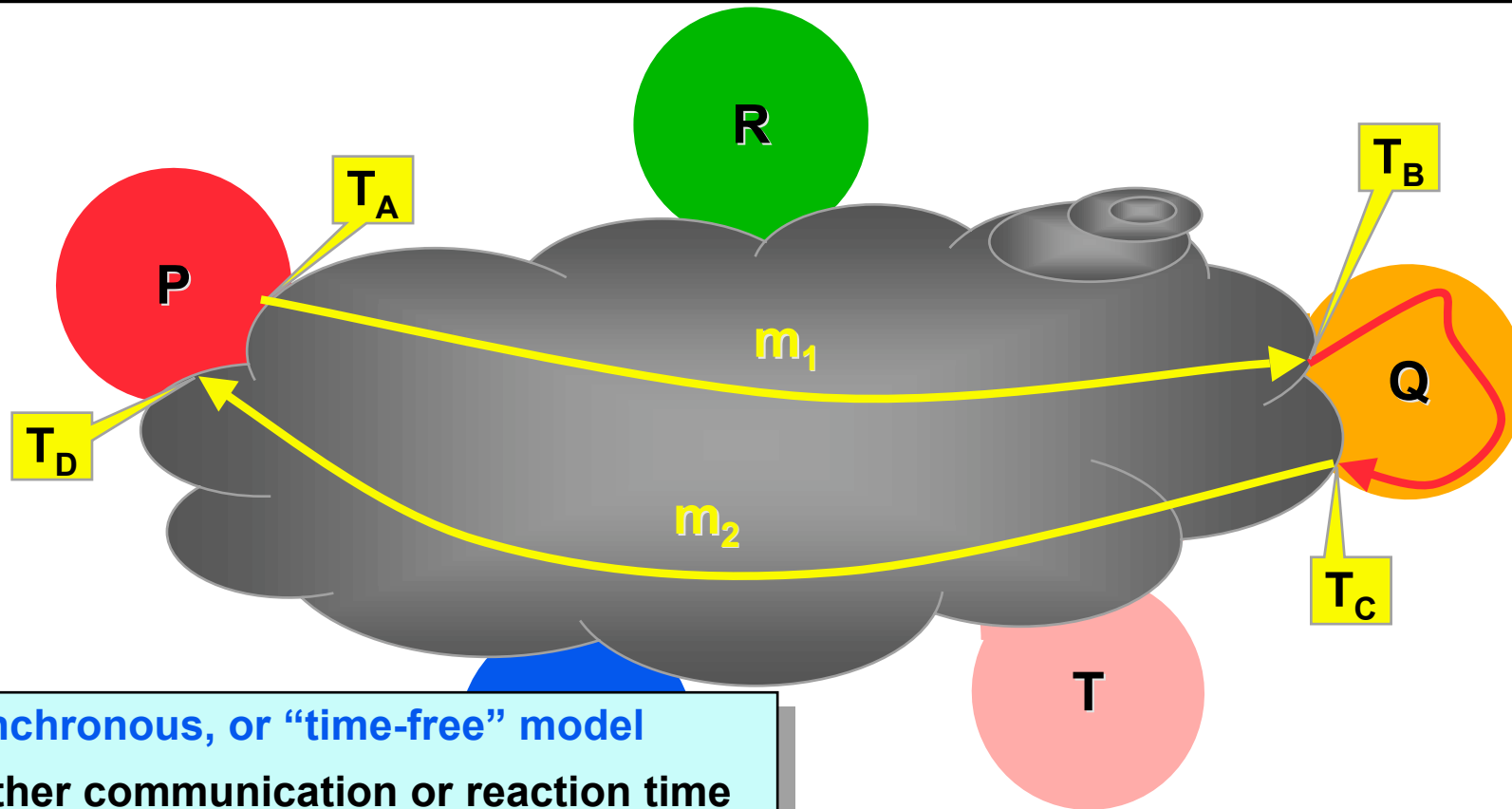
# Assumptions: Timing Models (2)



## Asynchronous, or “time-free” model

- ➔ either communication or reaction time bound does not exist
- ➔  $P$  cannot decide if  $Q$  has stopped, or if  $Q$ ,  $m_1$  or  $m_2$  are very slow

# Assumptions: Timing Models (2)

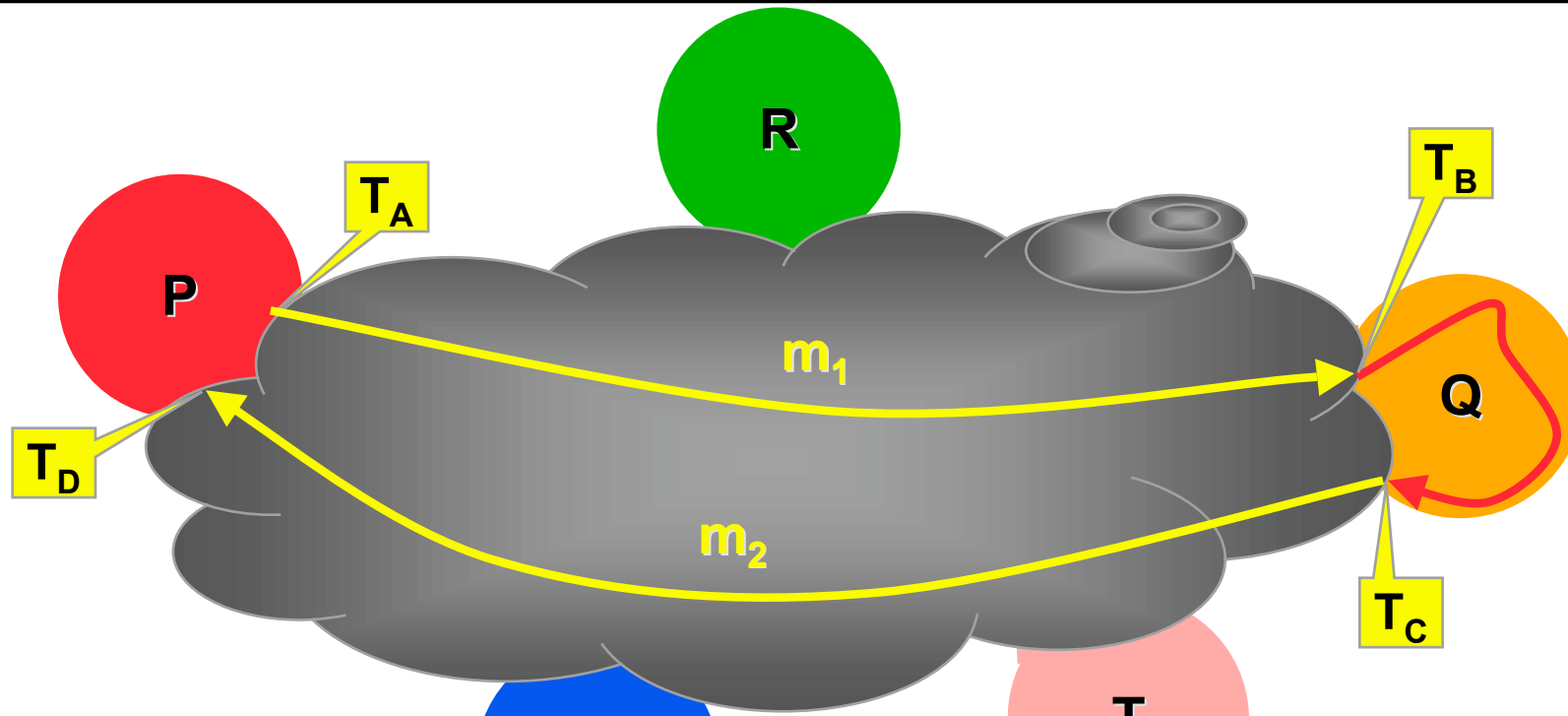


**Asynchronous, or “time-free” model**

- ↳ either communication or reaction time bound does not exist
- ↳ P cannot decide if Q has stopped, or if Q,  $m_1$  or  $m_2$  are very slow

Cannot (deterministically) solve consensus,  
so no atomic multicast

# Assumptions: Timing Models (2)



## Asynchronous, or “time-free” model

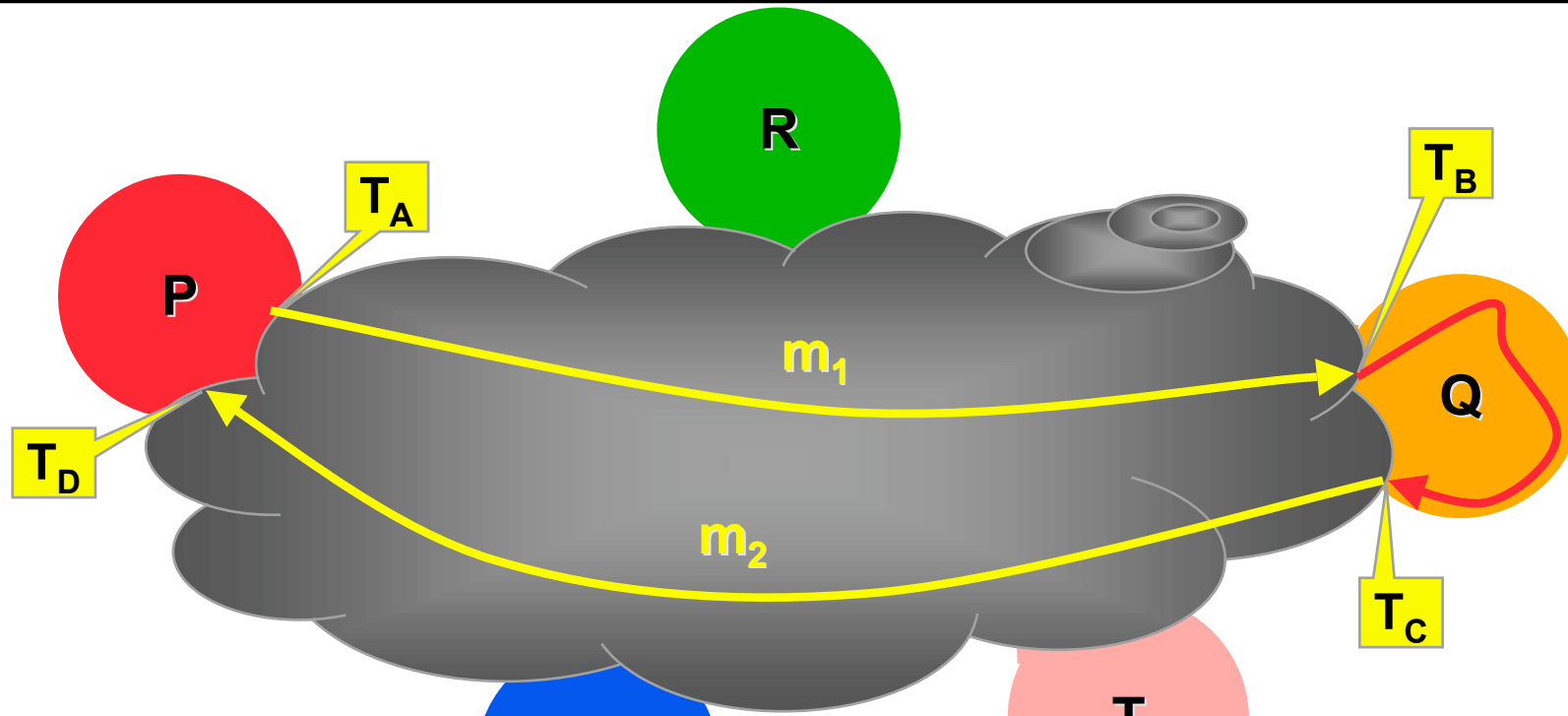
- either communication or reaction time bound does not exist
- P cannot decide if Q has stopped, or if Q, m1 or m2 are very slow

## Synchronous, or “bounded time” model

- communication bound guaranteed (the **network never fails**)
- P can declare that Q has failed if  $T_D - T_A > 2\Delta_P + \Delta_P$

Cannot (deterministically) solve consensus,  
so no atomic multicast

# Assumptions: Timing Models (2)



## Asynchronous, or “time-free” model

- either communication or reaction time bound does not exist
- P cannot decide if Q has stopped, or if Q, m1 or m2 are very slow

Cannot (deterministically) solve consensus,  
so no atomic multicast

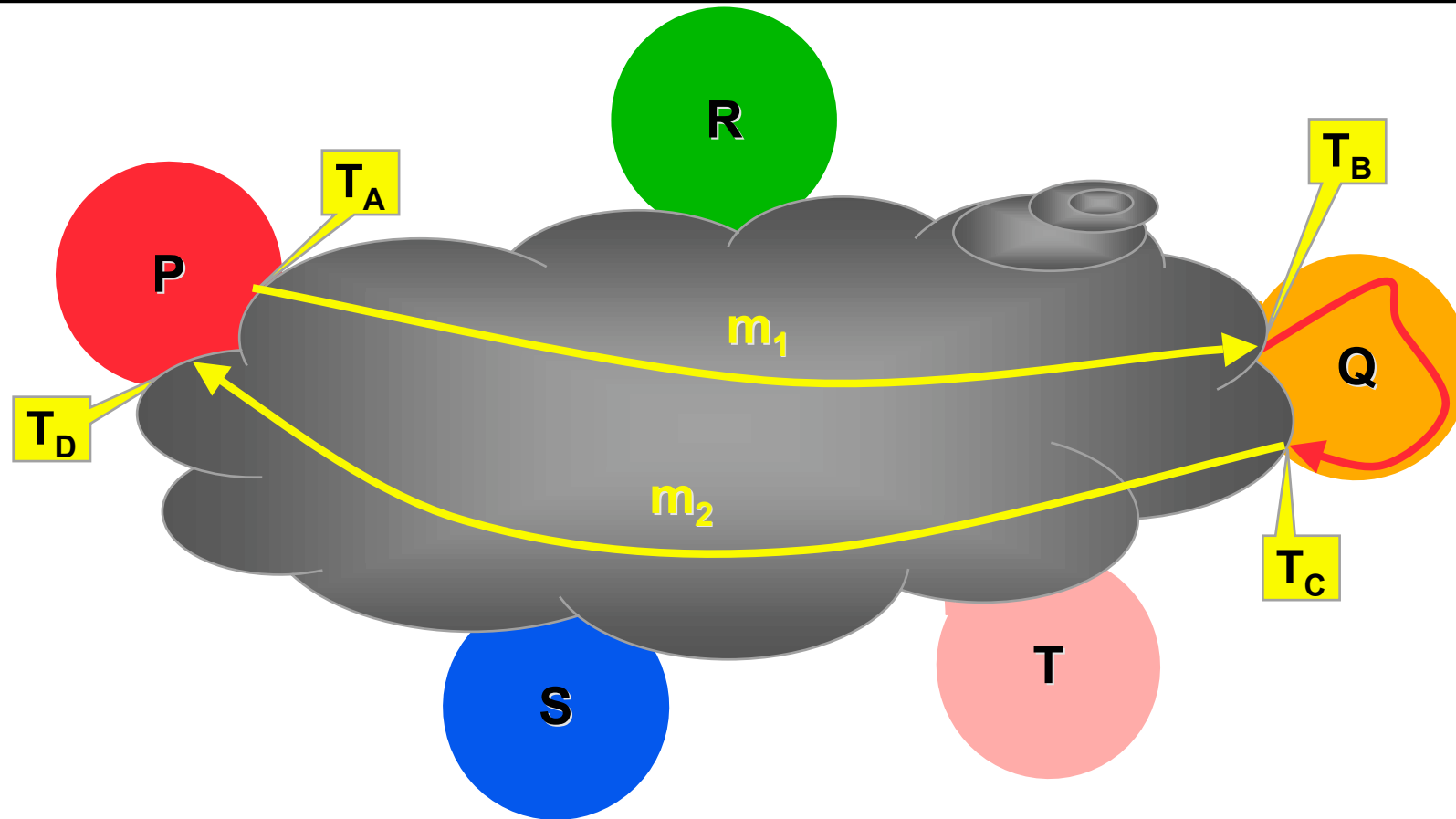
## Synchronous, or “bounded time” model

- communication bound guaranteed (the **network never fails**)
- P can declare that Q has failed if  $T_D - T_A > 2\Delta_P + \Delta_P$

Assumption coverage?

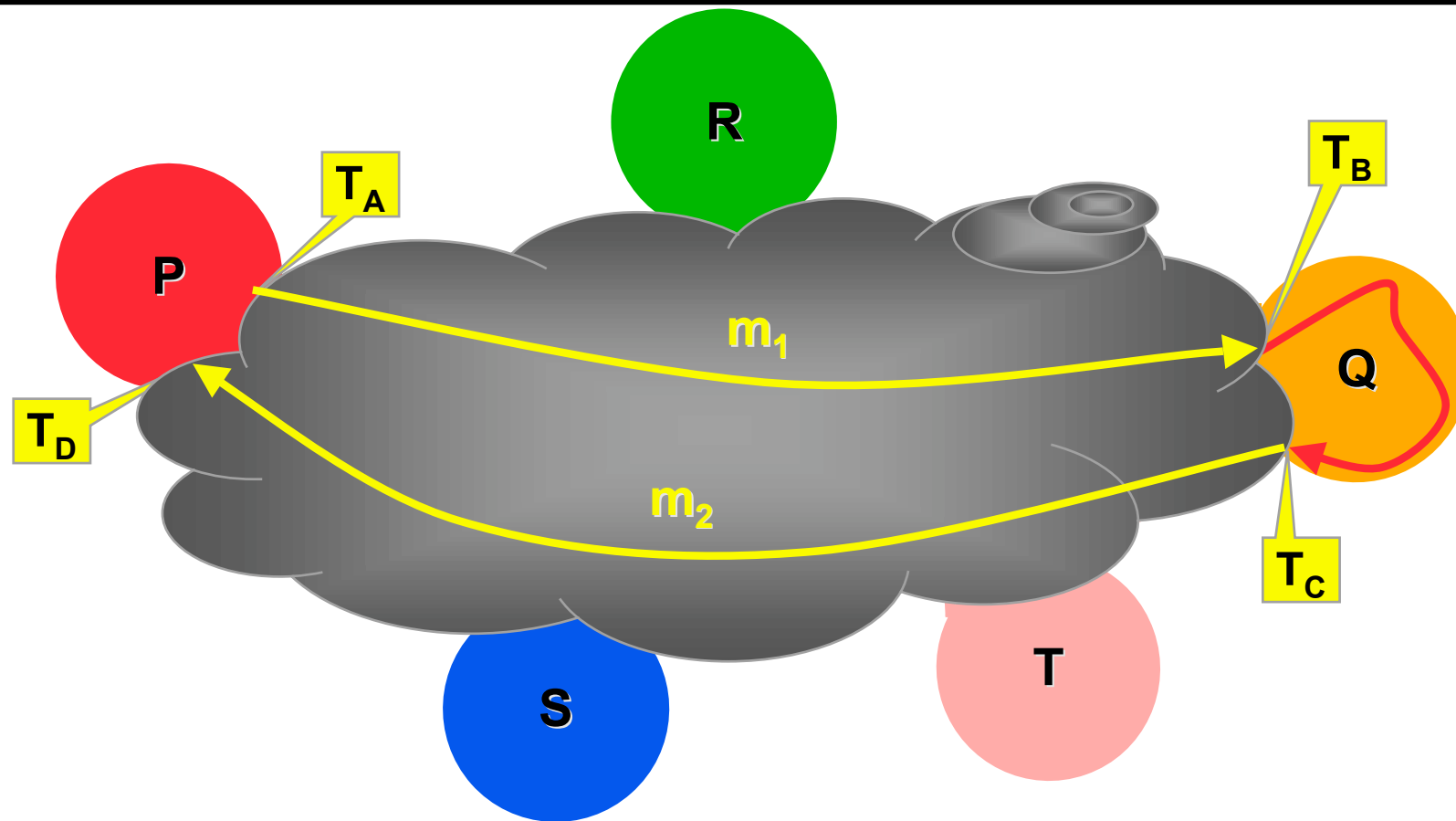
# Assumptions: Timing Models (3)

---



# Assumptions: Timing Models (3)

---

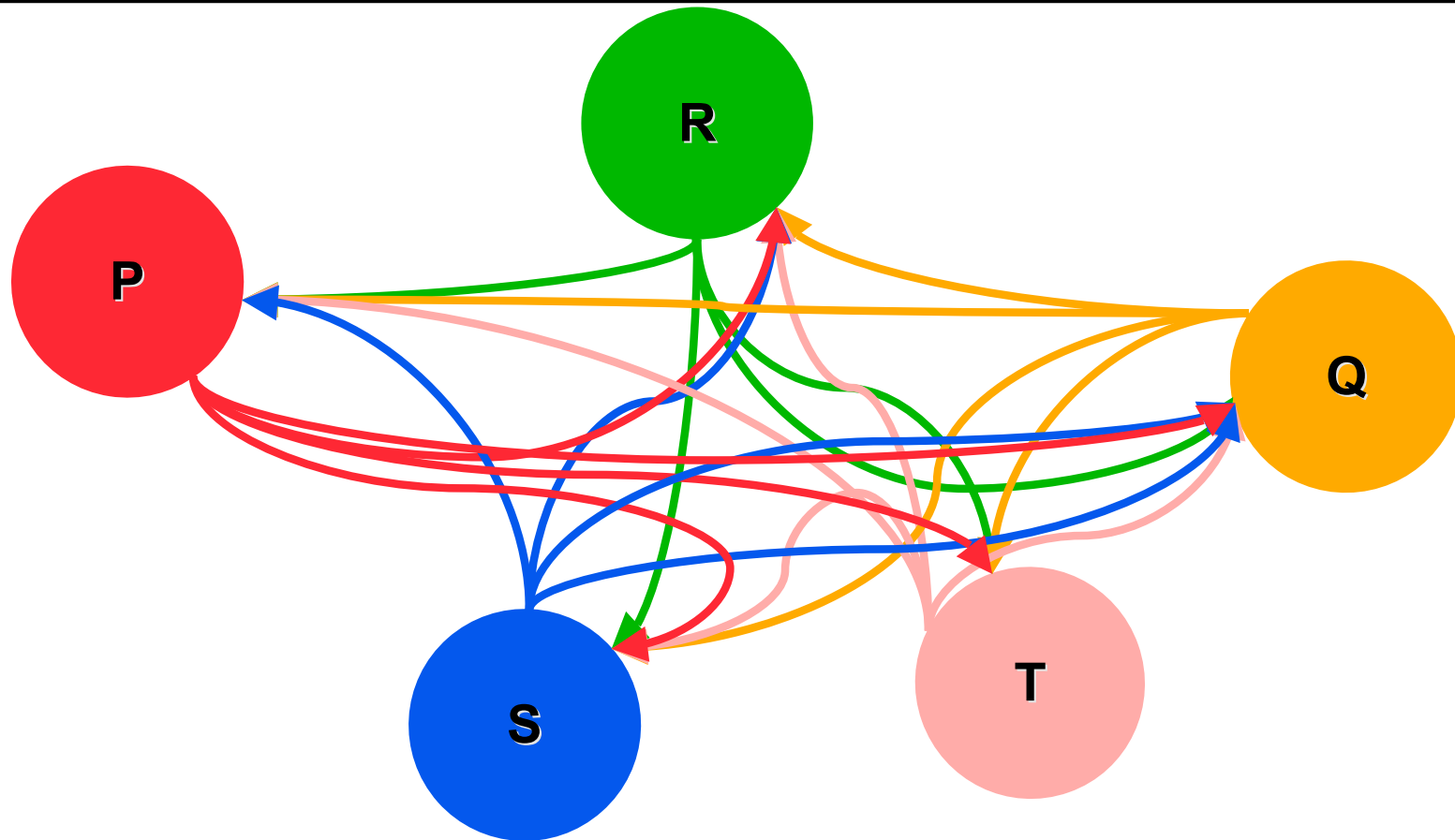


Non-probabilistic justification of synchronous model :

- ↳ *each process has its own private network*  
(a single fault confinement region)

# Assumptions: Timing Models (3)

---

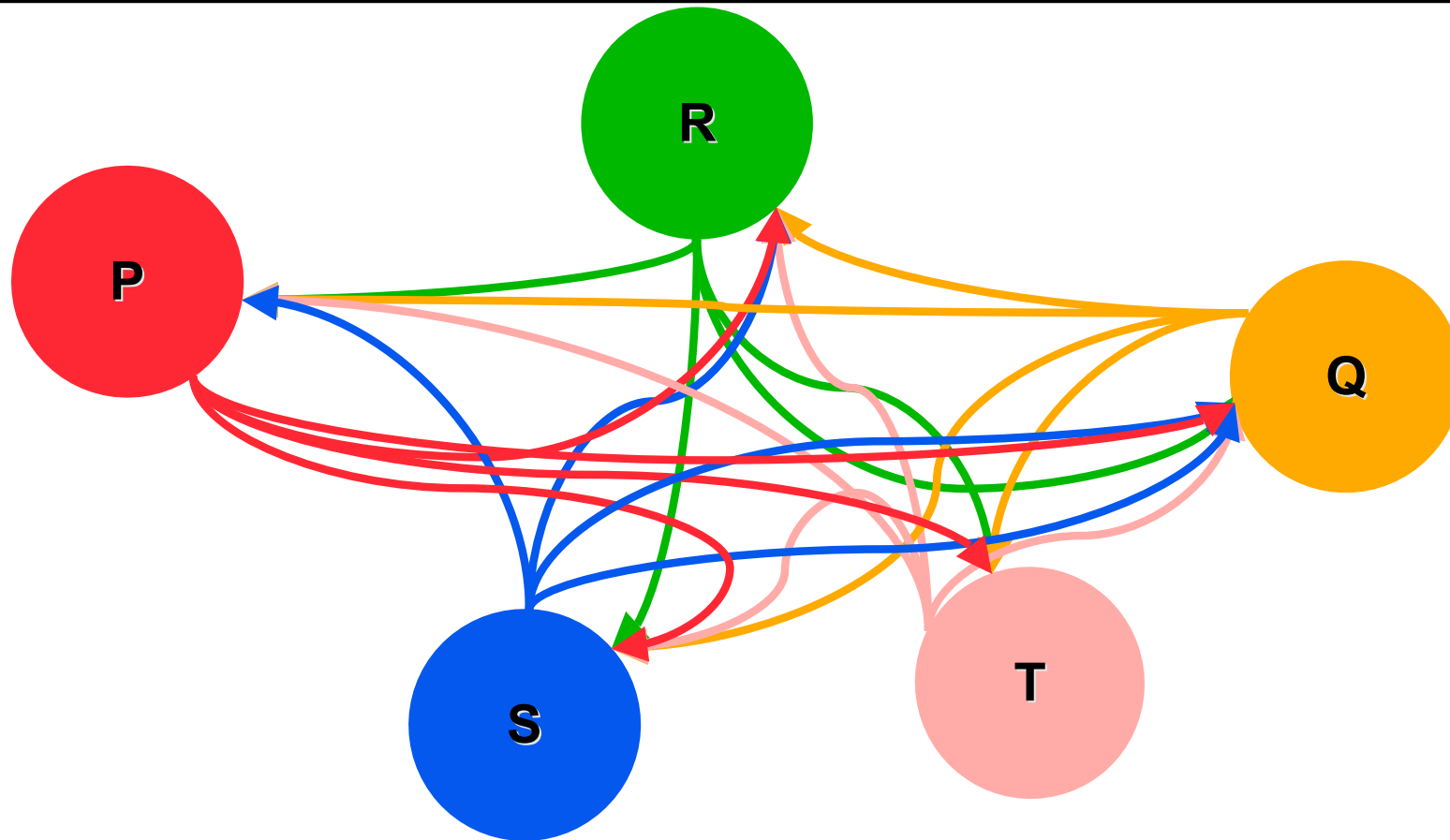


Non-probabilistic justification of synchronous model :

- ↳ *each process has its own private network*  
(a single fault confinement region)

# Assumptions: Timing Models (4)

---

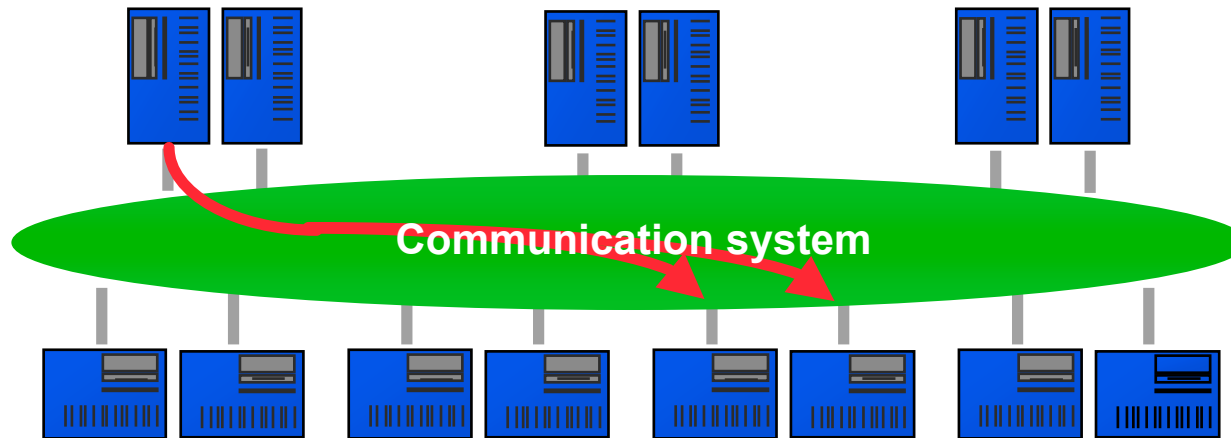


## ■ Practical substantiation of synchronous model:

- set of unidirectional busses (see SIFT, MAFT, GUARDS...)

# The Real System

---



## ■ Assumptions

- human lives are at stake, so must assume that communication is uncertain:
  - ➔ *messages can be lost (omission failures)*
  - ➔ *messages can be delayed (performance failures)*
- fail-safe processing units (coded processor technique)
- table-driven process scheduling
- fail-safe local clocks

## ■ The real system

- human lives are at stake, so must assume that communication is uncertain:
  - ↳ *messages can be lost (omission failures)*
  - ↳ *messages can be delayed (performance failures)*
- fail-safe processing units (coded processor technique)
- table-driven process scheduling
- fail-safe local clocks

## ■ The model

- Datagram service
  - ↳ *Defined upper quantile on transmission delay ( $\square$ )*
  - ↳ *Messages can only suffer omission/performance failures*
- Process management service
  - ↳ *Defined upper quantile on scheduling delay ( $\square$ )*
  - ↳ *Processes can only suffer crash/performance failures*
- Hardware clock service
  - ↳ *Each non-crashed process has access to a hardware clock with a known upper bound on drift rate ( $\square$ ) (NB. clocks are not (cannot be) deterministically synchronized)*

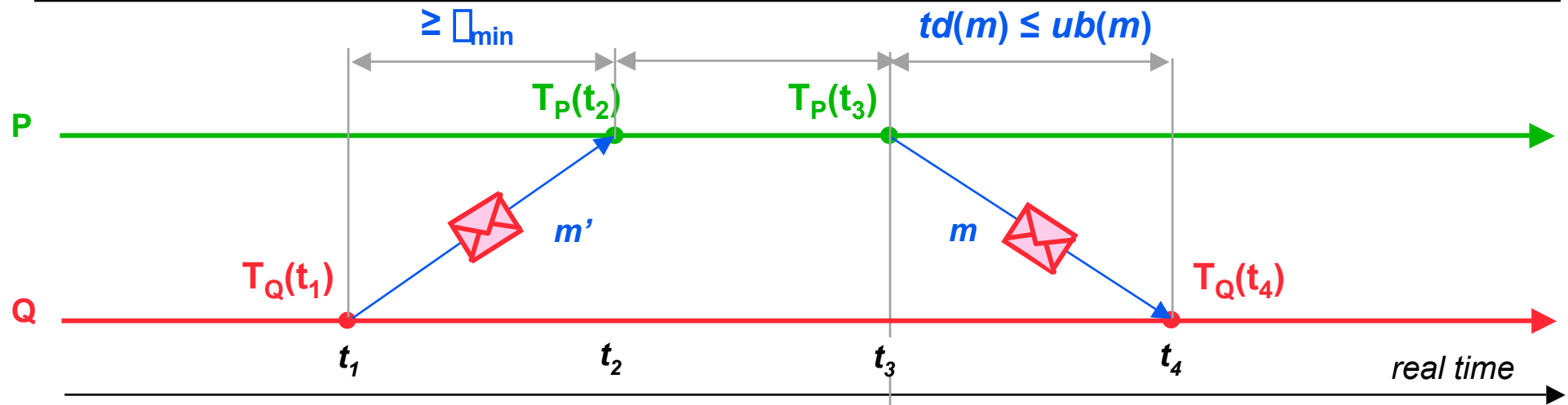
# Fail-Aware Datagram Service

---

[Fetzer & Cristian 1997]

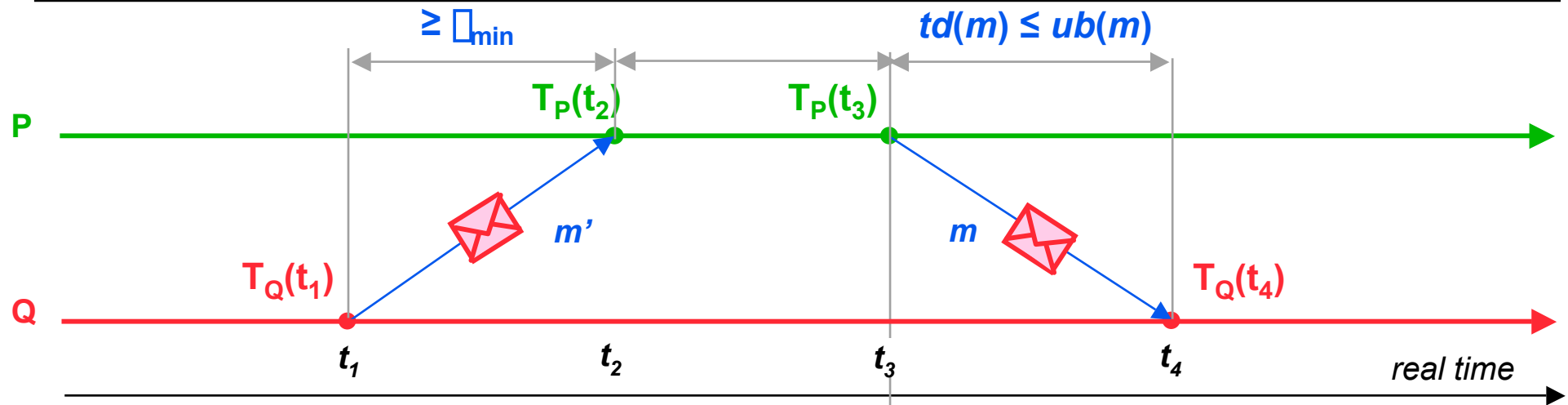
# Fail-Aware Datagram Service

[Fetzer & Cristian 1997]



# Fail-Aware Datagram Service

[Fetzer & Cristian 1997]

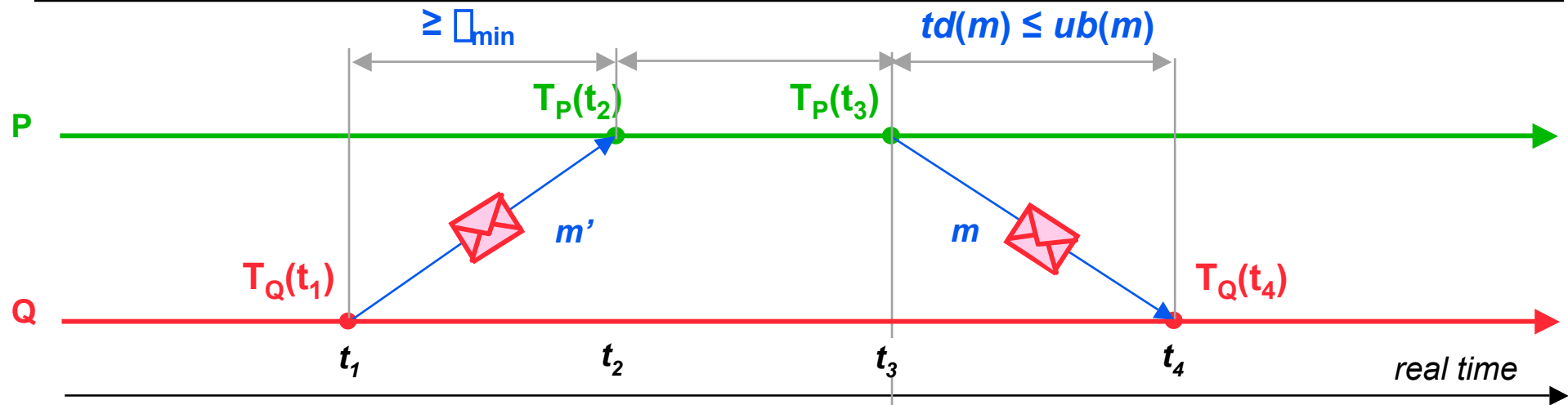


- Let  $td(m)$  be the real delay incurred by a message  $m$

$$td(m) = t_4 - t_3 = (t_4 - t_1) - (t_3 - t_2) - (t_2 - t_1)$$

# Fail-Aware Datagram Service

[Fetzer & Cristian 1997]



- Let  $td(m)$  be the real delay incurred by a message  $m$

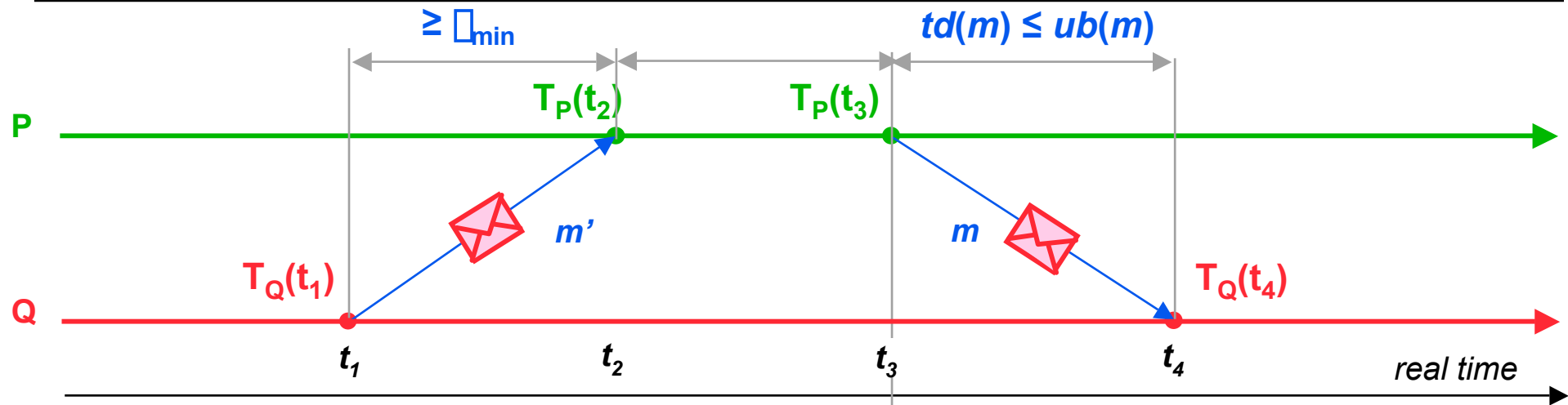
$$td(m) = t_4 - t_3 = (t_4 - t_1) - (t_3 - t_2) - (t_2 - t_1)$$

- Upper bound  $ub(m)$  on  $td(m)$ :

$$ub(m) = (T_Q(t_4) - T_Q(t_1)) \cdot (1 + \square) - (T_P(t_3) - T_P(t_2)) \cdot (1 - \square) - \square_{\min}$$

# Fail-Aware Datagram Service

[Fetzer & Cristian 1997]



- Let  $td(m)$  be the real delay incurred by a message  $m$

$$td(m) = t_4 - t_3 = (t_4 - t_1) - (t_3 - t_2) - (t_2 - t_1)$$

- Upper bound  $ub(m)$  on  $td(m)$ :

$$ub(m) = (T_Q(t_4) - T_Q(t_1)) \cdot (1 + \square) - (T_P(t_3) - T_P(t_2)) \cdot (1 - \square) - \square_{\min}$$

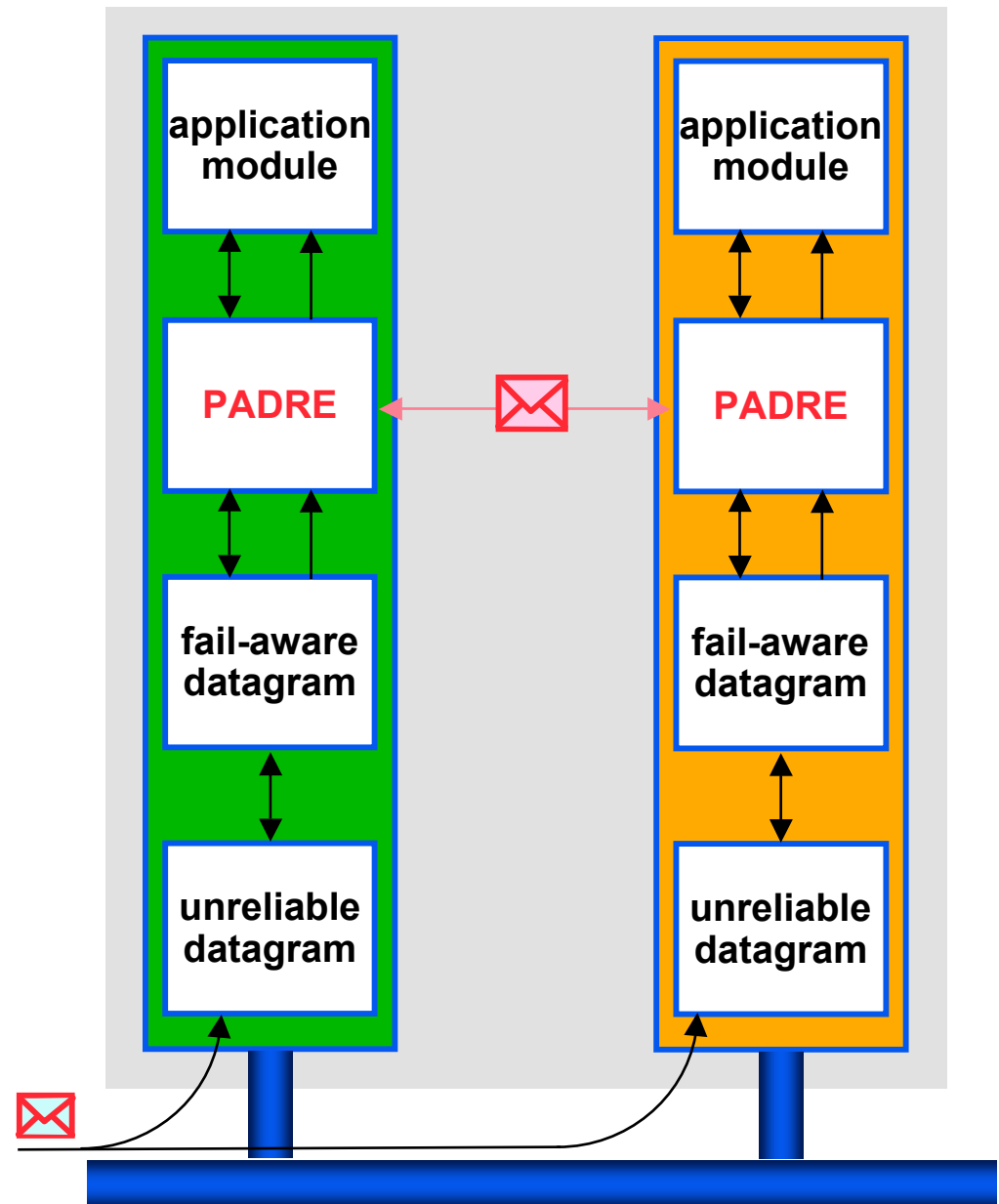
- Choose a constant  $\square$  so that  $m$  can be classified according to:

- if  $ub(m) \leq \square$  message is **fast**
- if  $ub(m) > \square$  message is **slow**

# Protocol for **A**symmetric **D**uplex **RE**dundancy [Essamé et al. 1999]

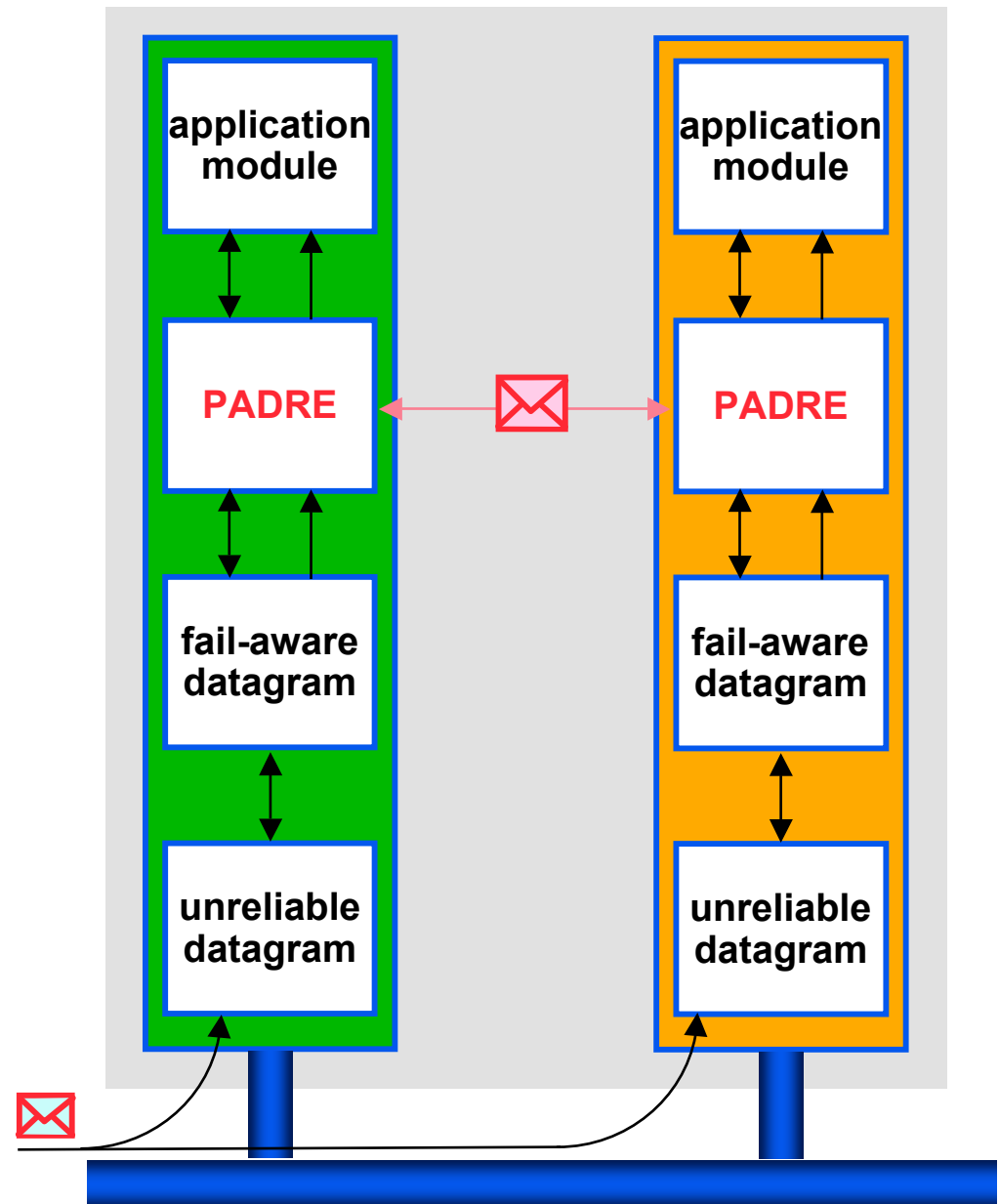
---

# Protocol for Asymmetric Duplex REdundancy [Essamé et al. 1999]



# Protocol for Asymmetric Duplex REdundancy [Essamé et al. 1999]

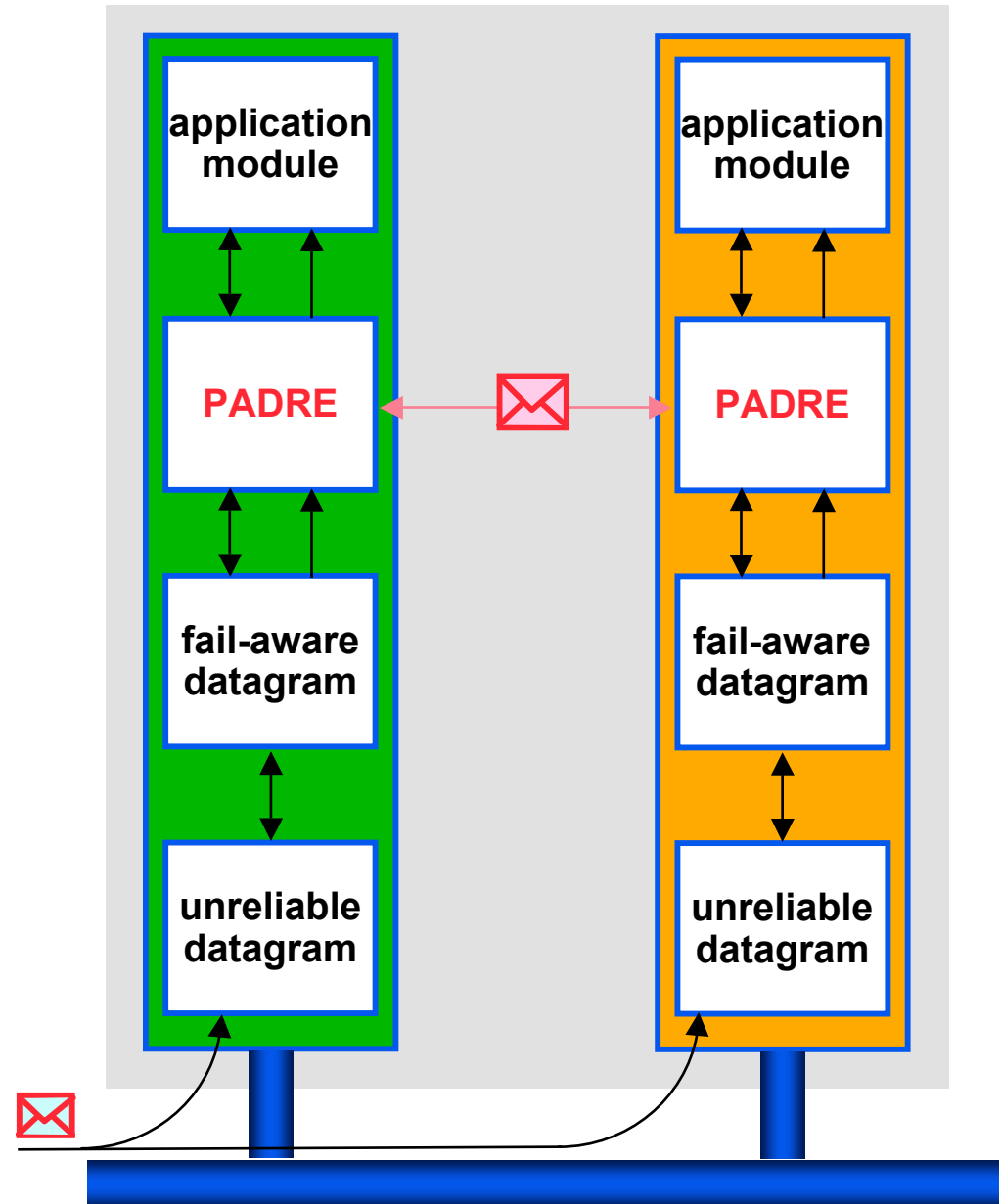
## ■ Idea:



# Protocol for Asymmetric Duplex REdundancy [Essamé et al. 1999]

## ■ Idea:

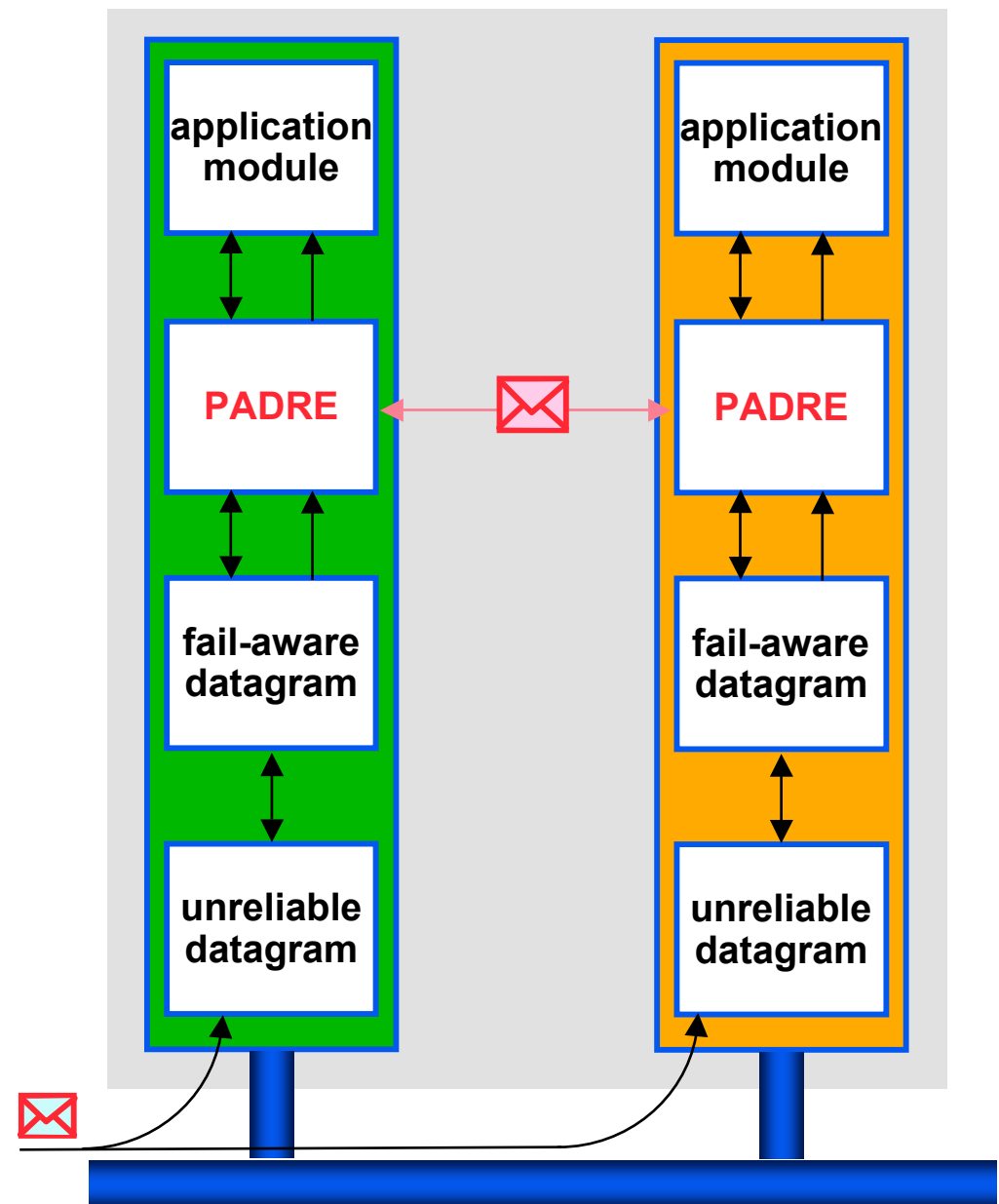
- Cannot guarantee consistency of duplicated units since communication is uncertain



# Protocol for Asymmetric Duplex REdundancy [Essamé et al. 1999]

## ■ Idea:

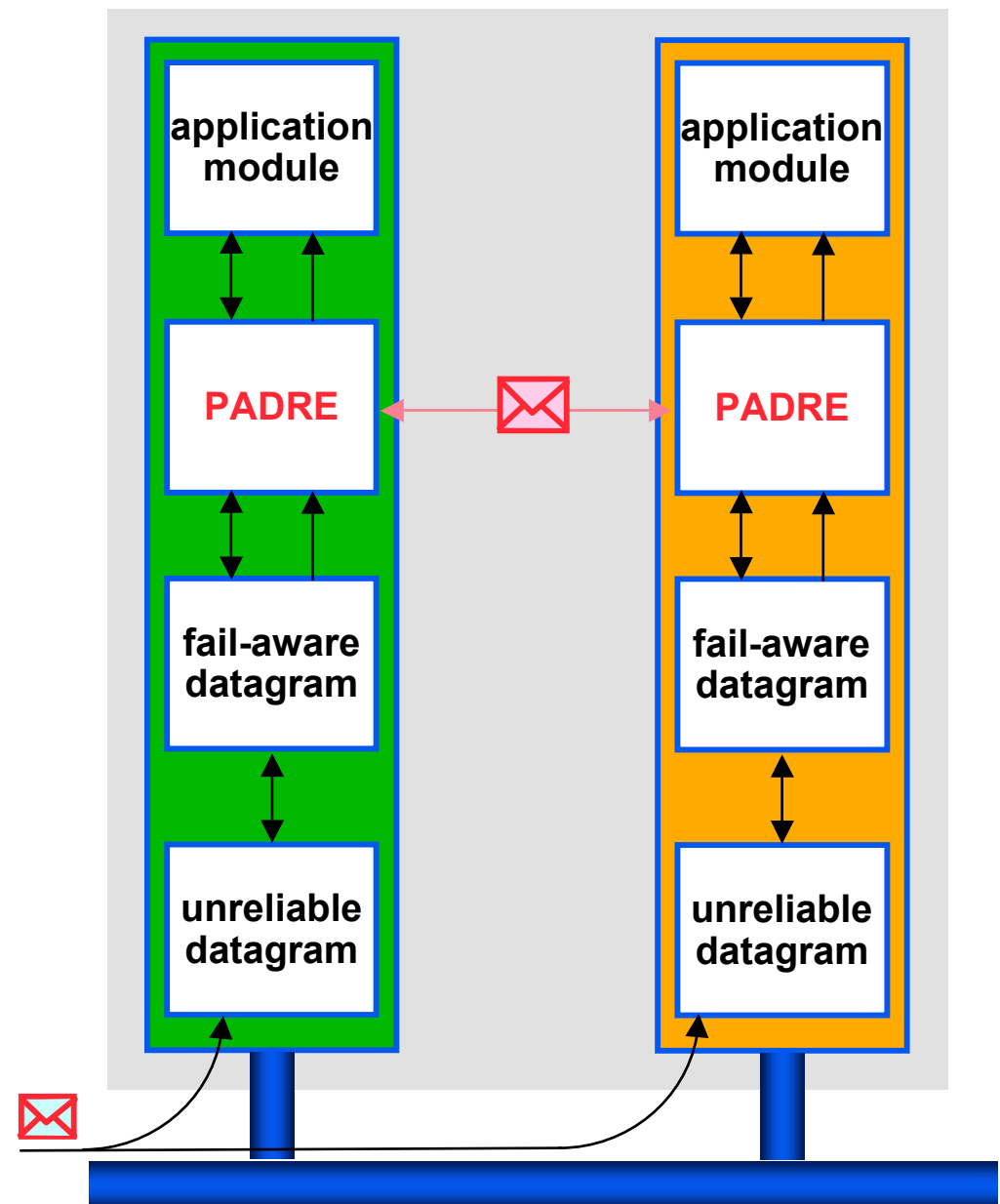
- Cannot guarantee consistency of duplicated units since communication is uncertain
- So, build a *fail-aware* multicast protocol



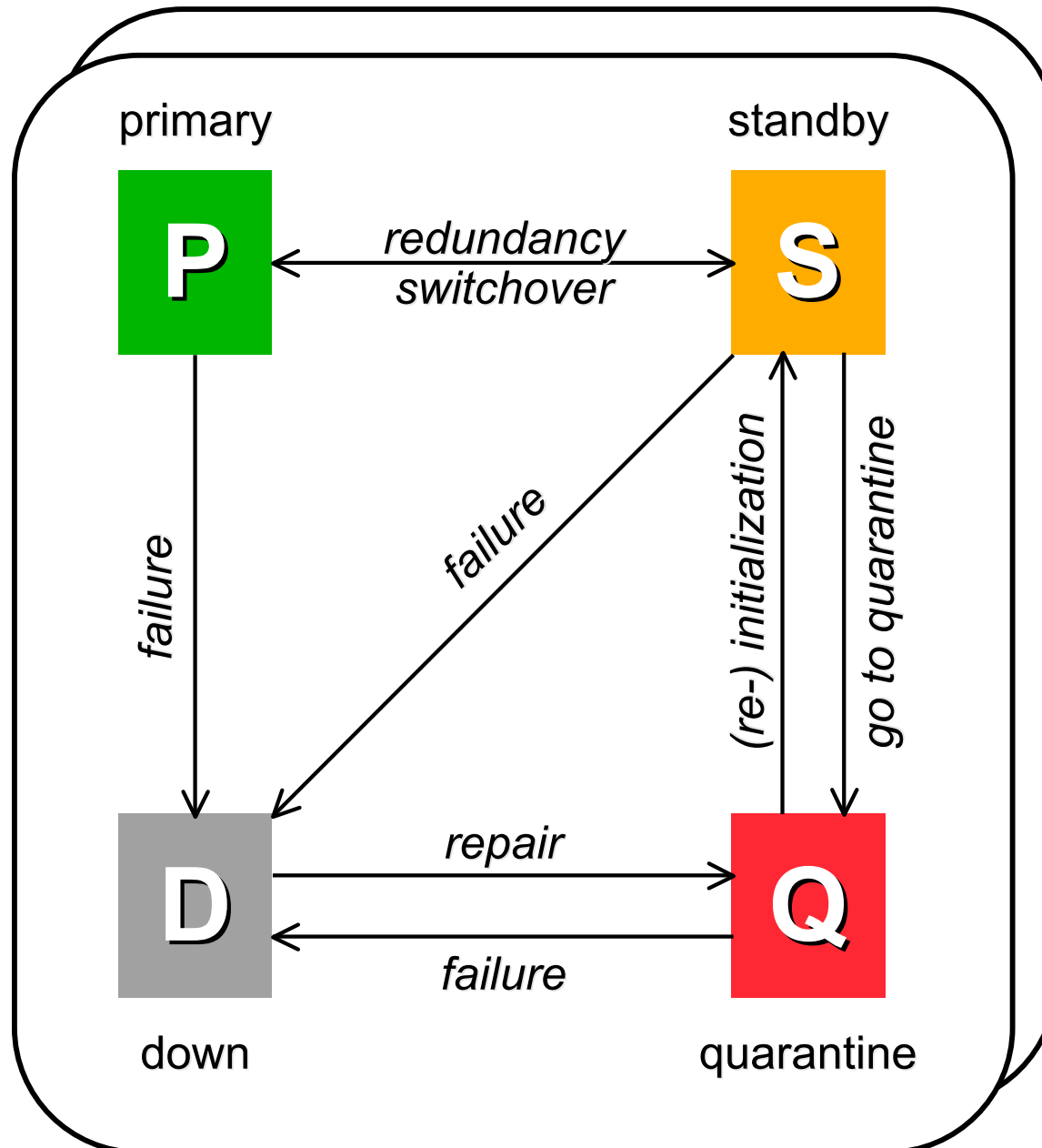
# Protocol for Asymmetric Duplex REdundancy [Essamé et al. 1999]

## ■ Idea:

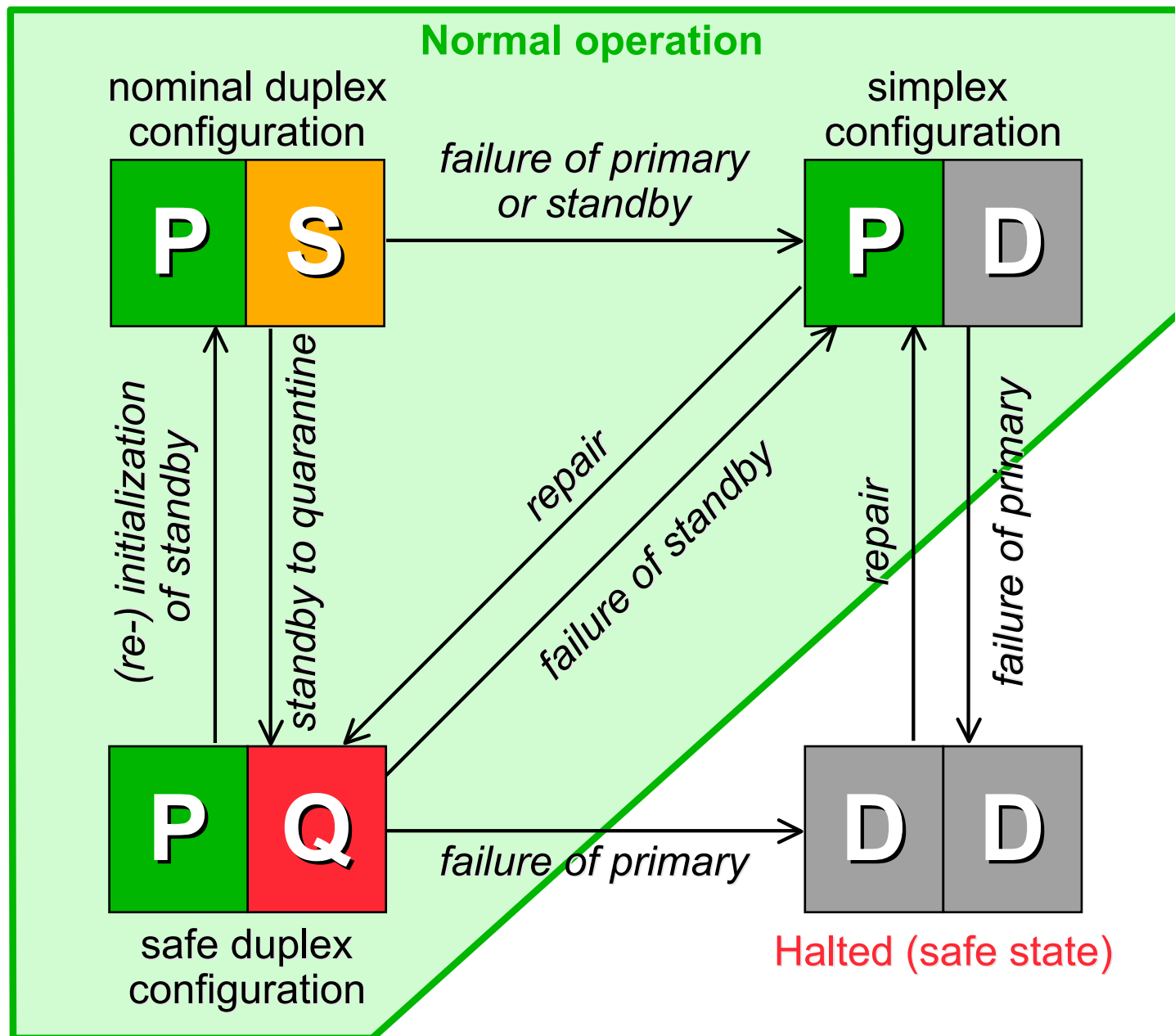
- Cannot guarantee consistency of duplicated units since communication is uncertain
- So, build a *fail-aware* multicast protocol
- Indicator signals when consistency is ensured
  - ↳ *Nominal duplex configuration*
    - primary unit in **primary** mode
    - secondary unit in **standby** mode



# States of Each Unit



# States of Duplex Pair



# Protocol Properties

---

# Protocol Properties

---

## ■ Safety properties

- **Unique Primary (UP):** at any instant, only one unit is in the primary mode
- **Quarantine (MQ):** Secondary must leave standby mode within bounded delay if inconsistent with Primary; return to standby mode only allowed when consistent
- **Prefix of History (PH):** history of Primary must always be a prefix of that of Secondary

# Protocol Properties

---

## ■ Safety properties

- **Unique Primary (UP):** at any instant, only one unit is in the primary mode
- **Quarantine (MQ):** Secondary must leave standby mode within bounded delay if inconsistent with Primary; return to standby mode only allowed when consistent
- **Prefix of History (PH):** history of Primary must always be a prefix of that of Secondary

## ■ Progress properties

- **Agreement (AP):** in the absence of faults, any input accepted by one unit at time  $t$  is accepted by the other unit in the interval  $[ t-\Delta , t+\Delta ]$
- **Limited Quarantine (LQ):** in the absence of faults, a unit in quarantine must eventually switch to standby

# Protocol Principle

---

# Protocol Principle

---

## ■ Primary only accepts an input if the secondary:

- ↳ *has accepted it, or*
- ↳ *has been placed in quarantine, or*
- ↳ *has failed*

# Protocol Principle

---

- **Primary only accepts an input if the secondary:**
  - ↳ *has accepted it, or*
  - ↳ *has been placed in quarantine, or*
  - ↳ *has failed*
  
- **Secondary only accepts messages sent to it from the primary**

# Reception Protocol

---

# Reception Protocol

---

Primary



Secondary



# Reception Protocol

---

Primary

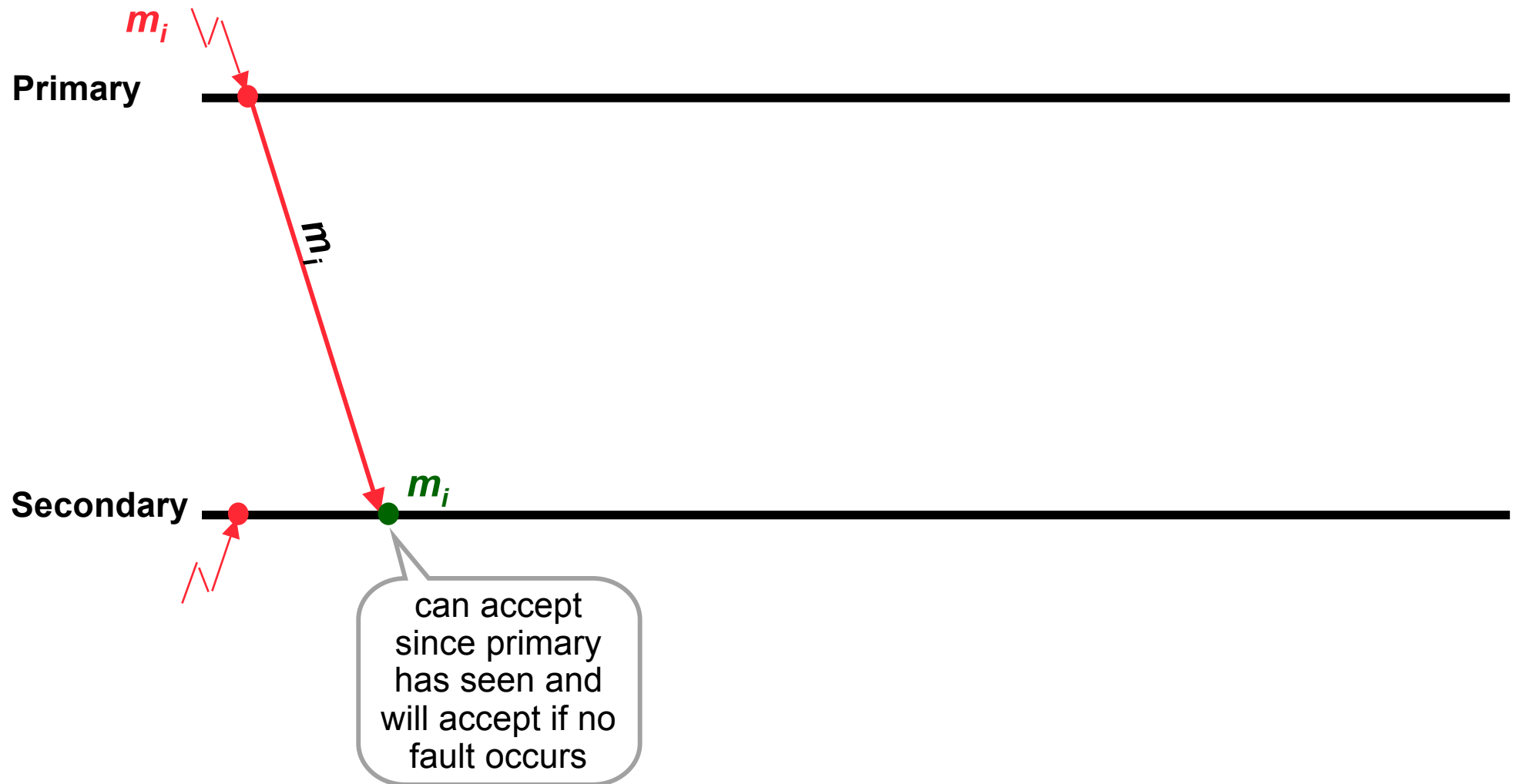


Secondary

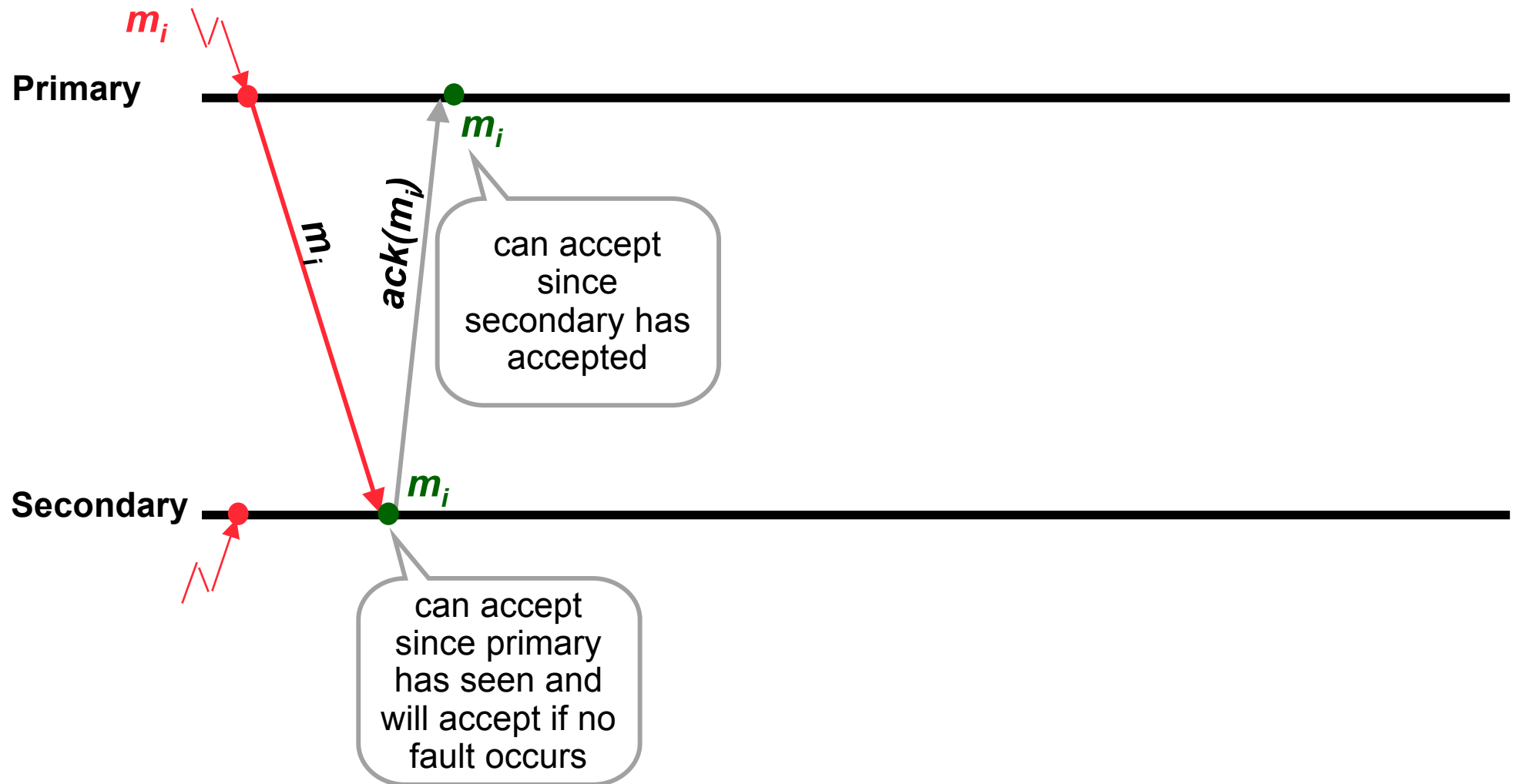


# Reception Protocol

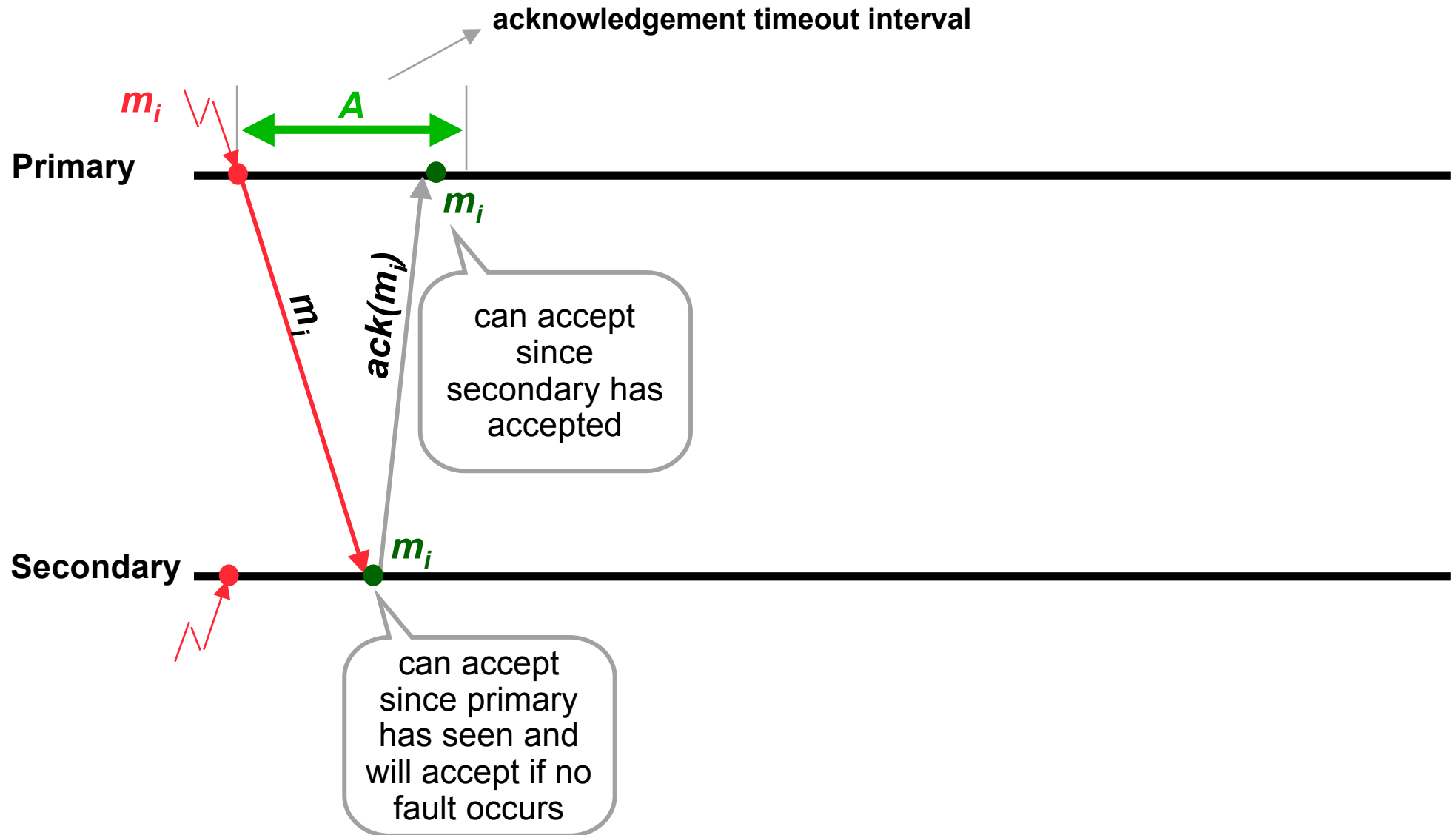
---



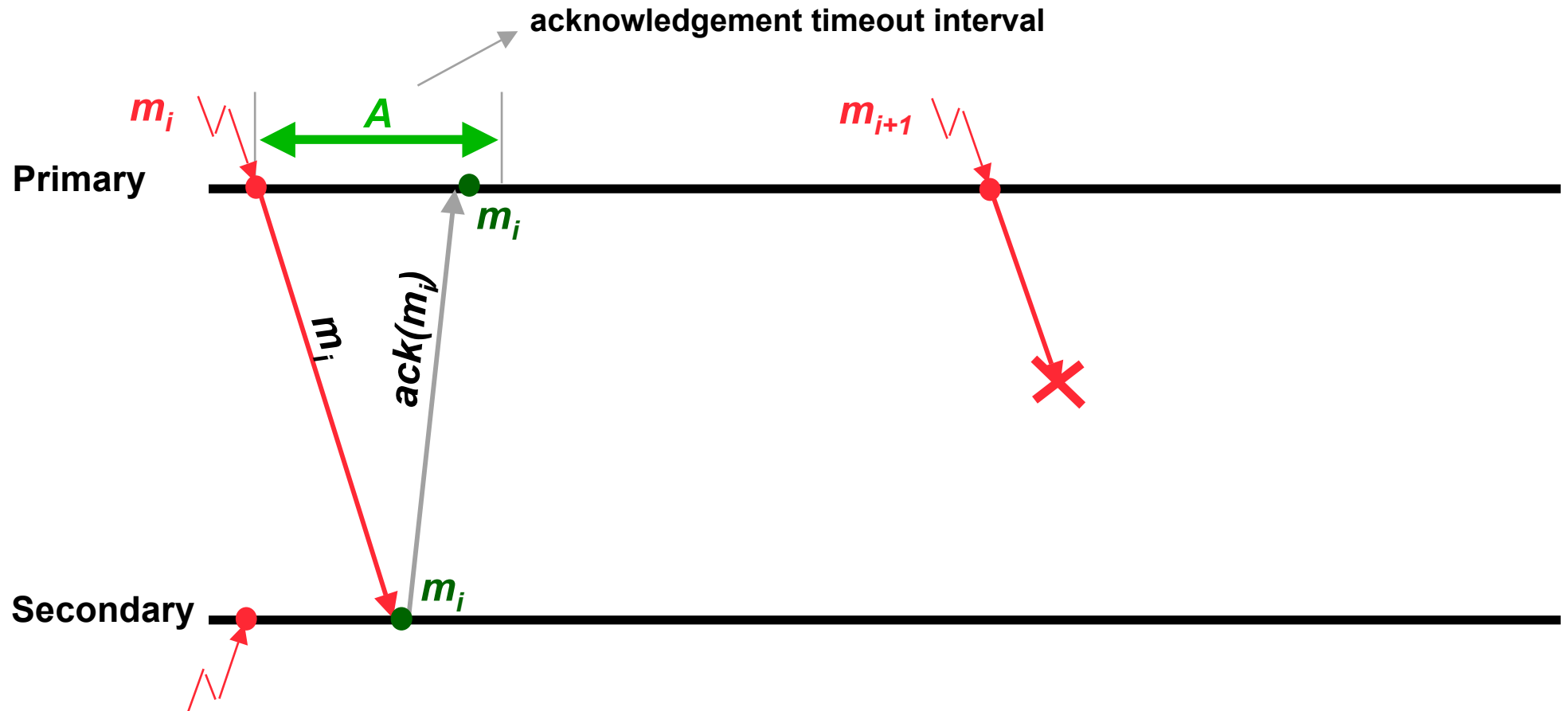
# Reception Protocol



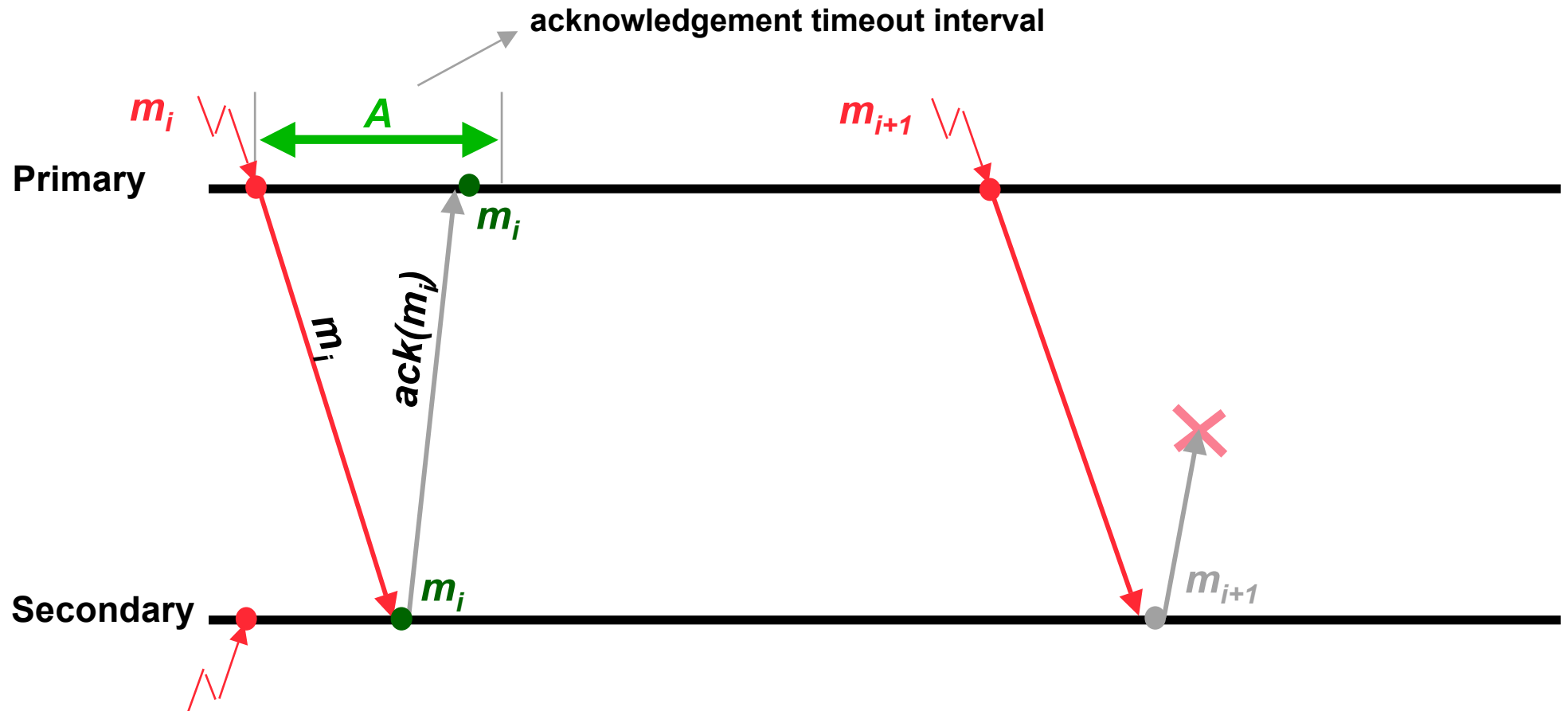
# Reception Protocol



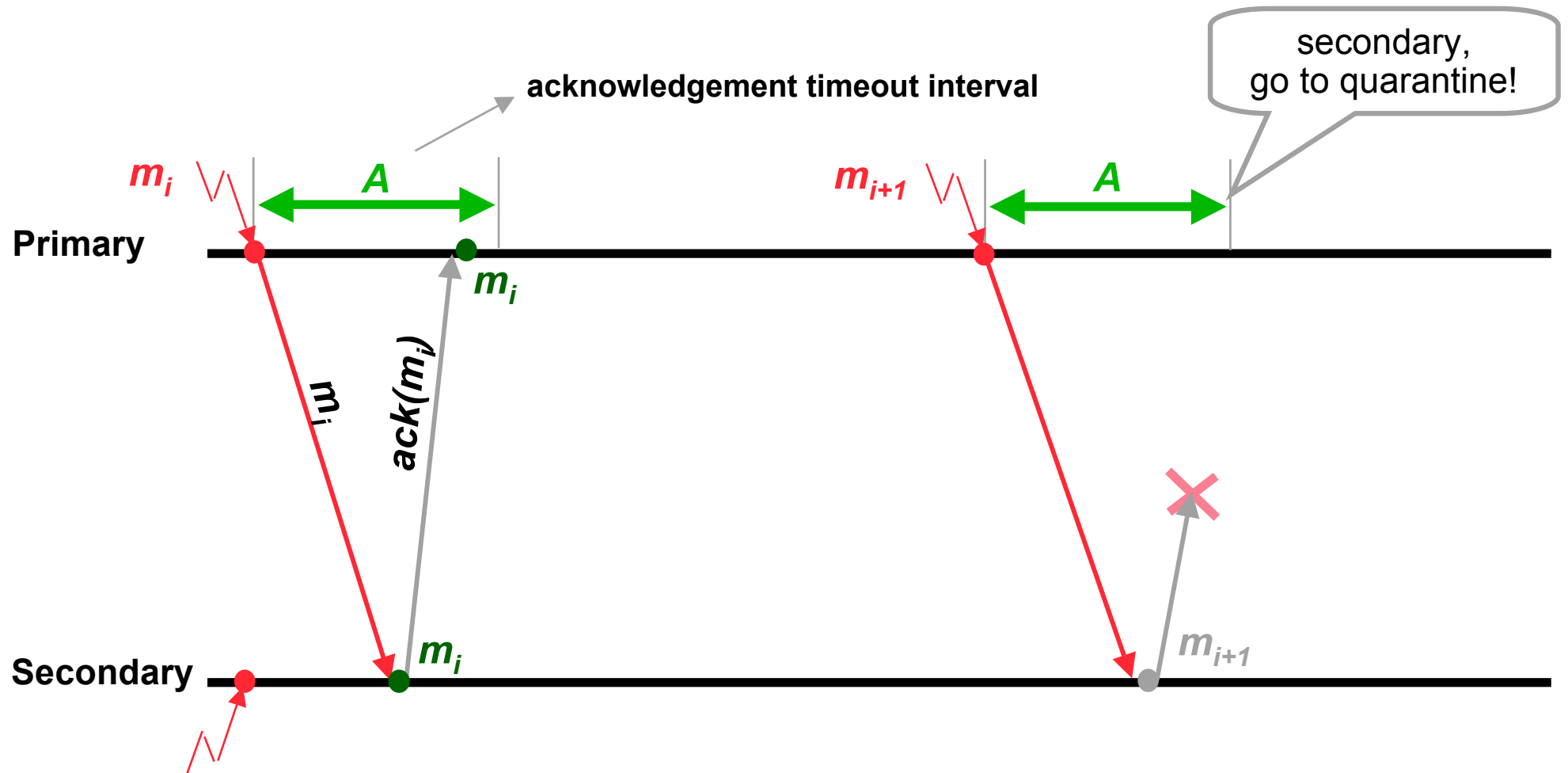
# Reception Protocol



# Reception Protocol



# Reception Protocol

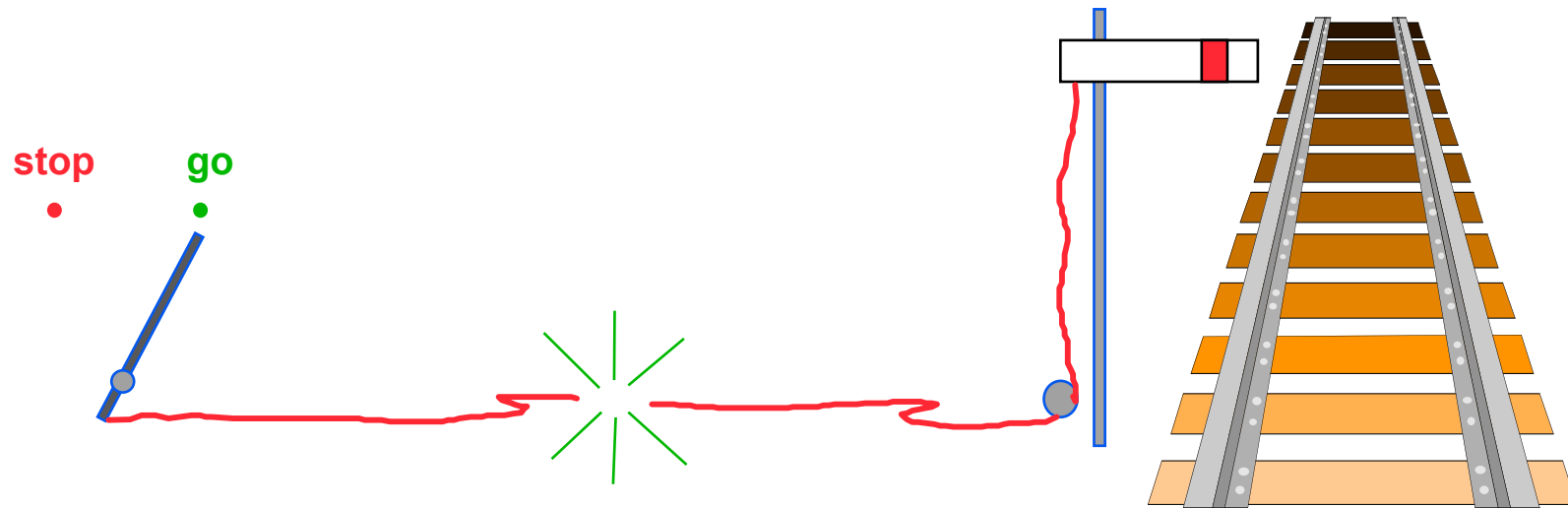


# Quarantine Control Protocol

---

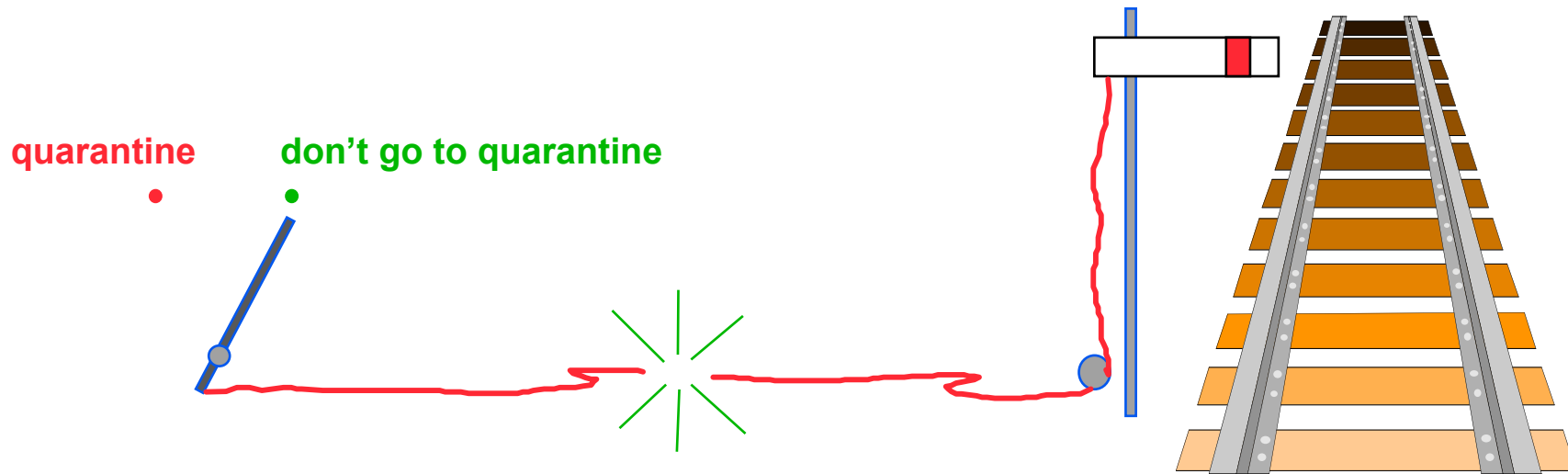
# Quarantine Control Protocol

---

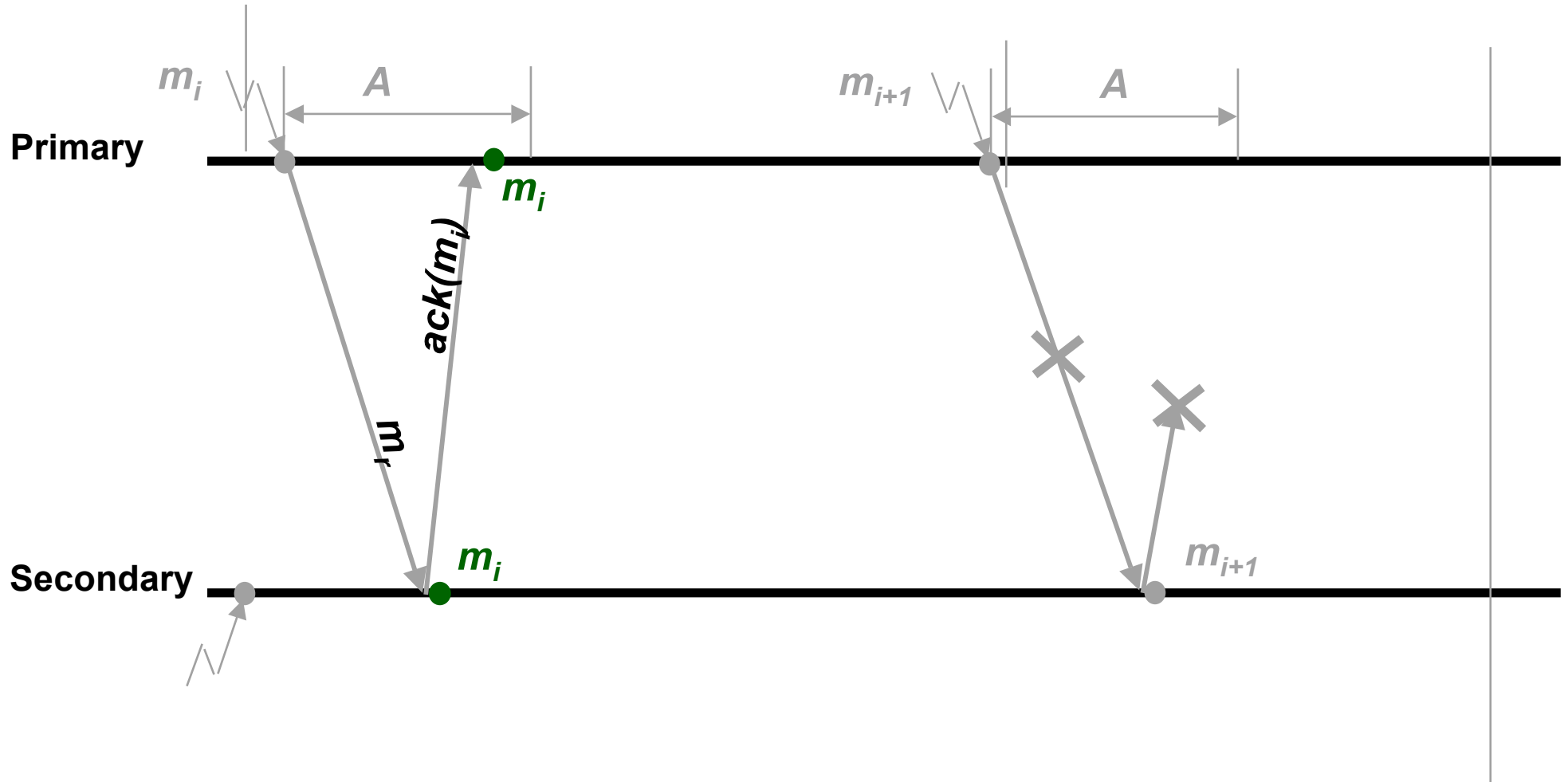


# Quarantine Control Protocol

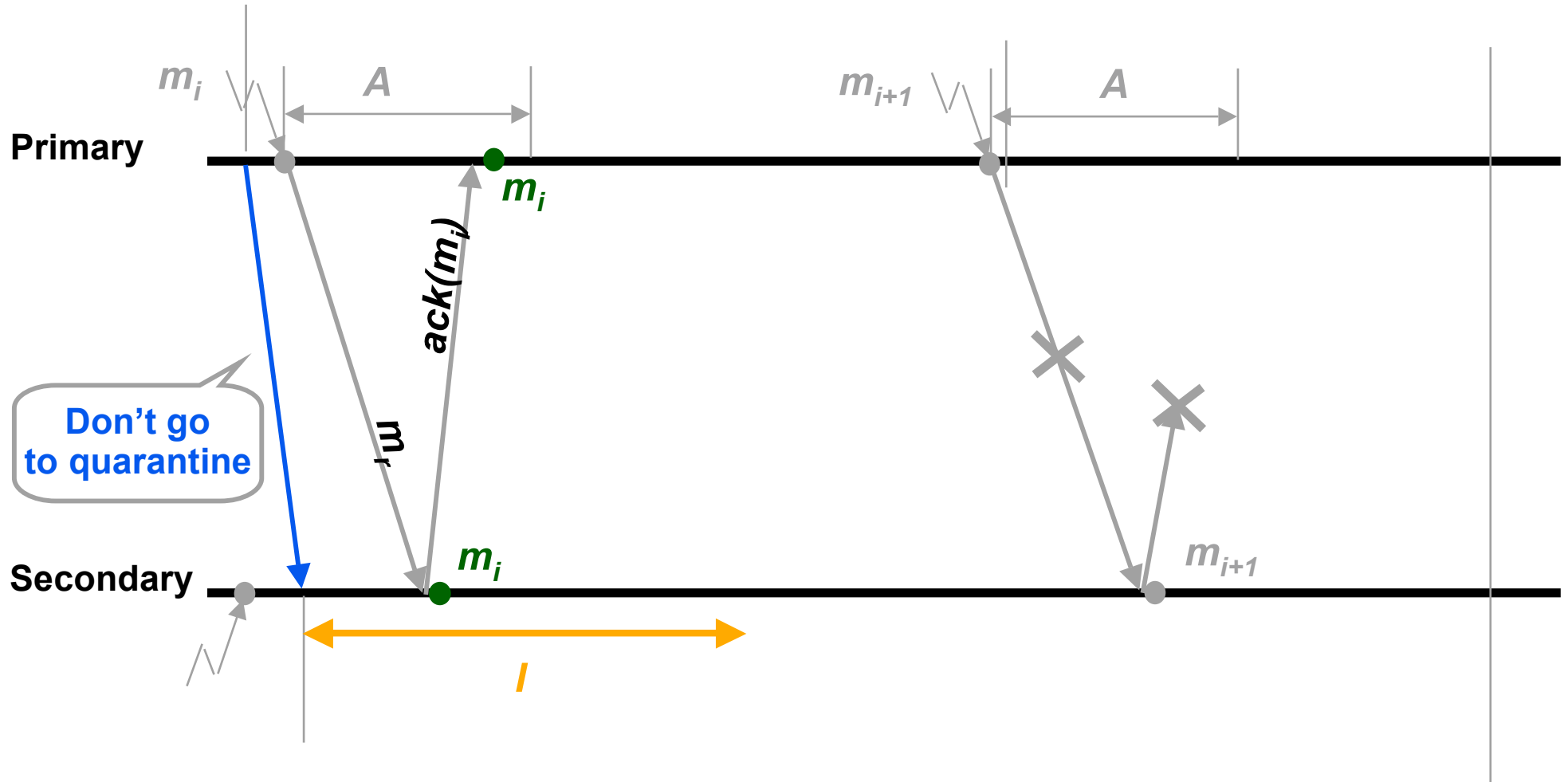
---



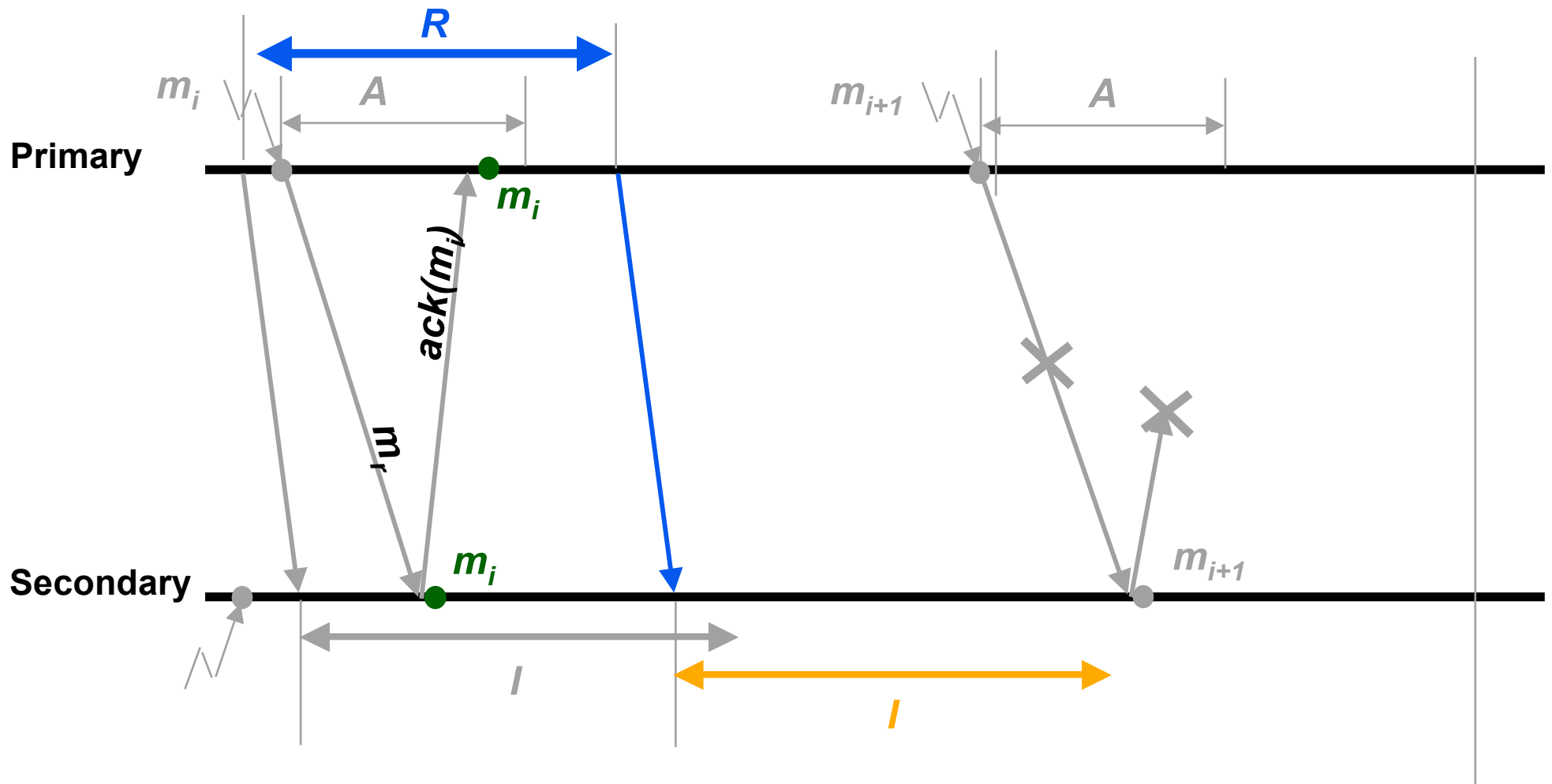
# Quarantine Control Protocol



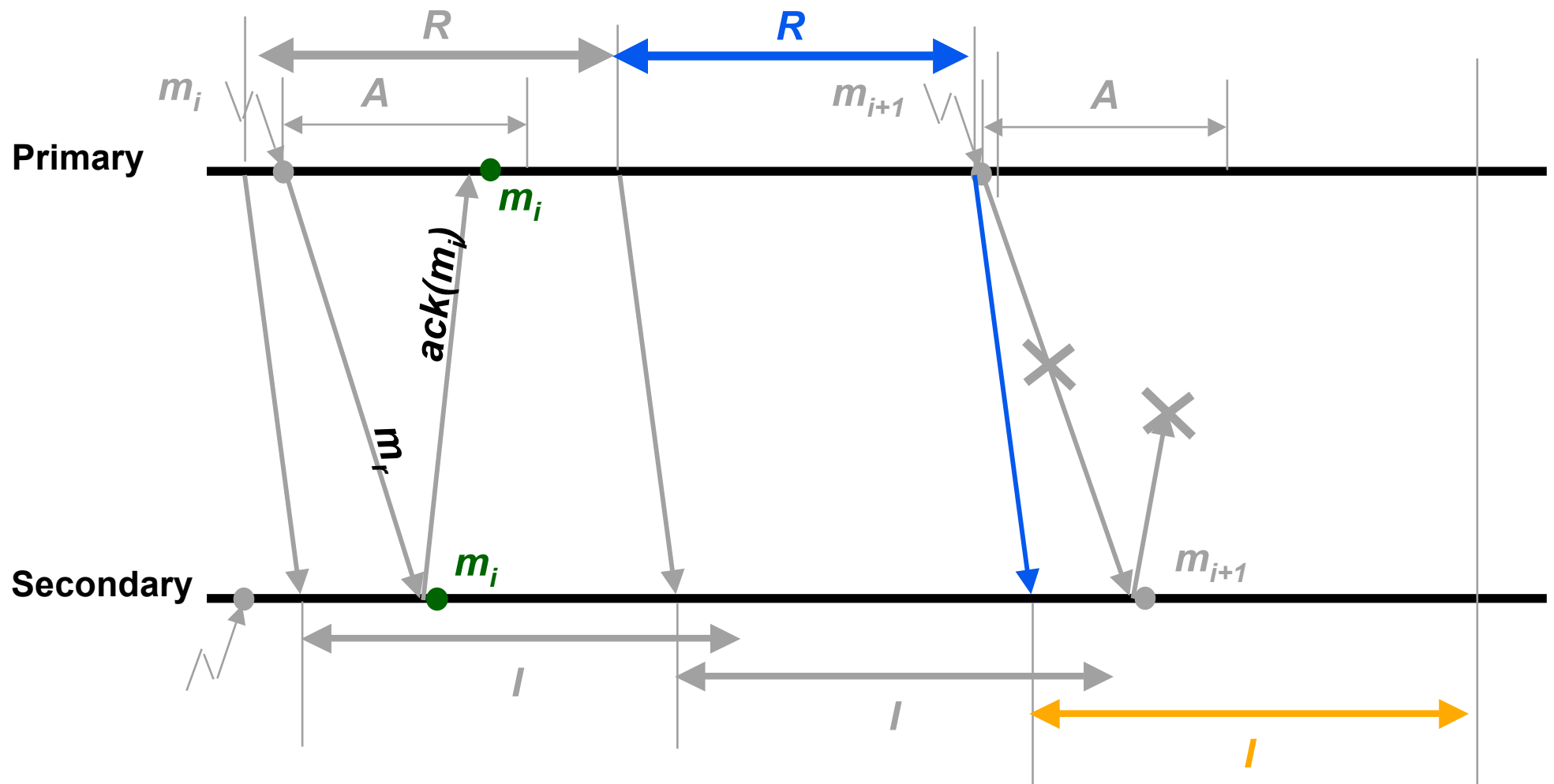
# Quarantine Control Protocol



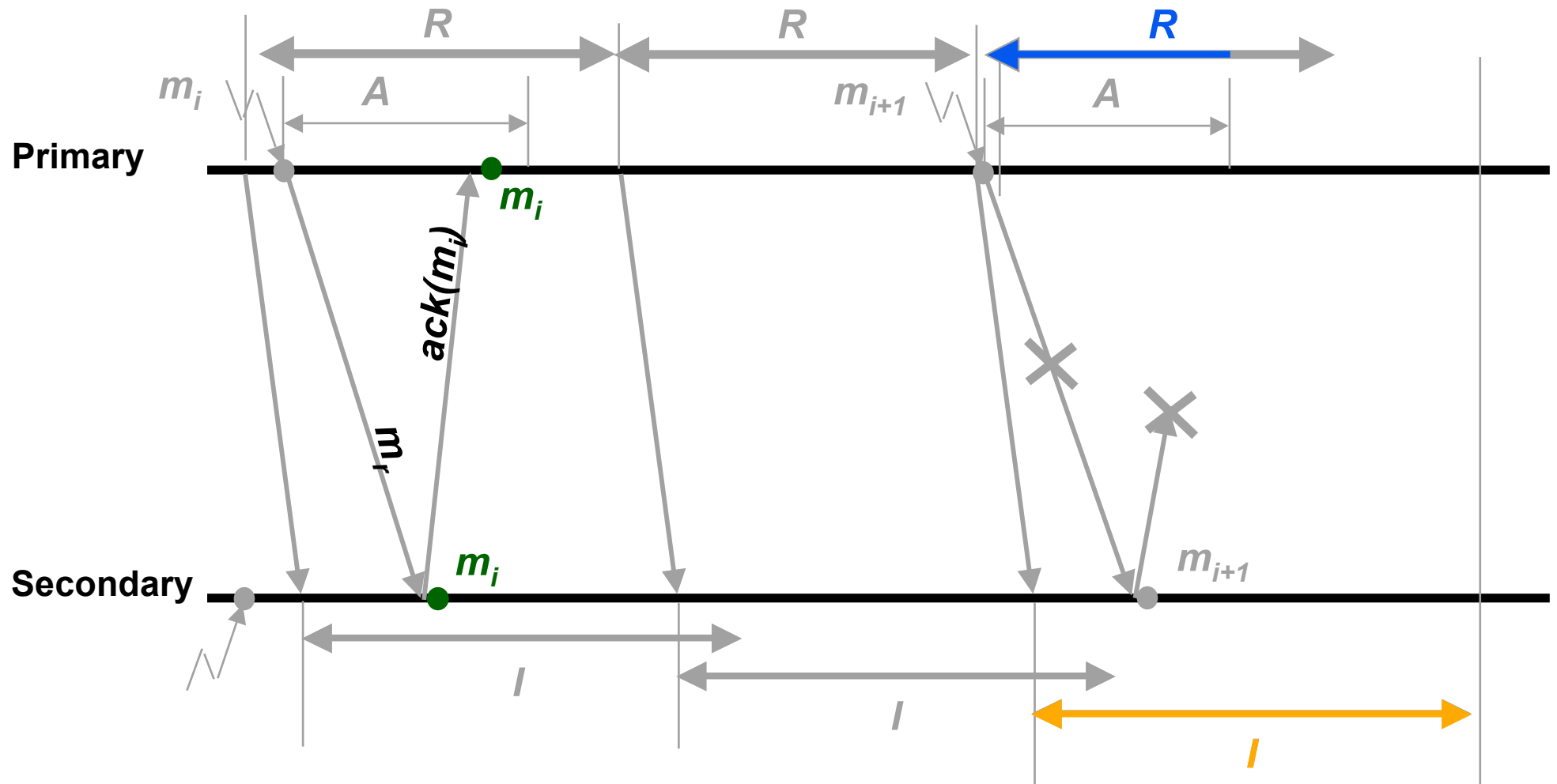
# Quarantine Control Protocol



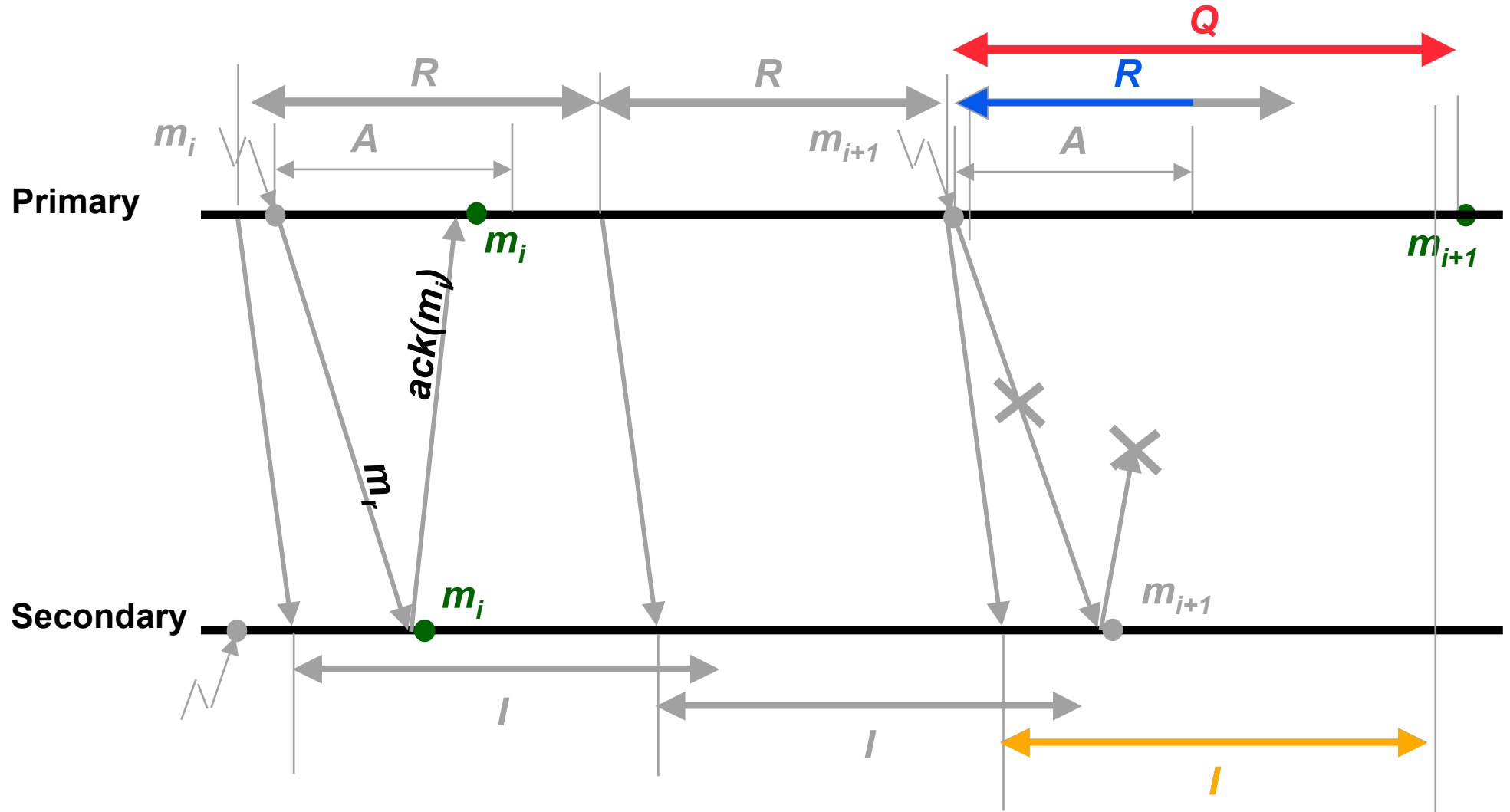
# Quarantine Control Protocol



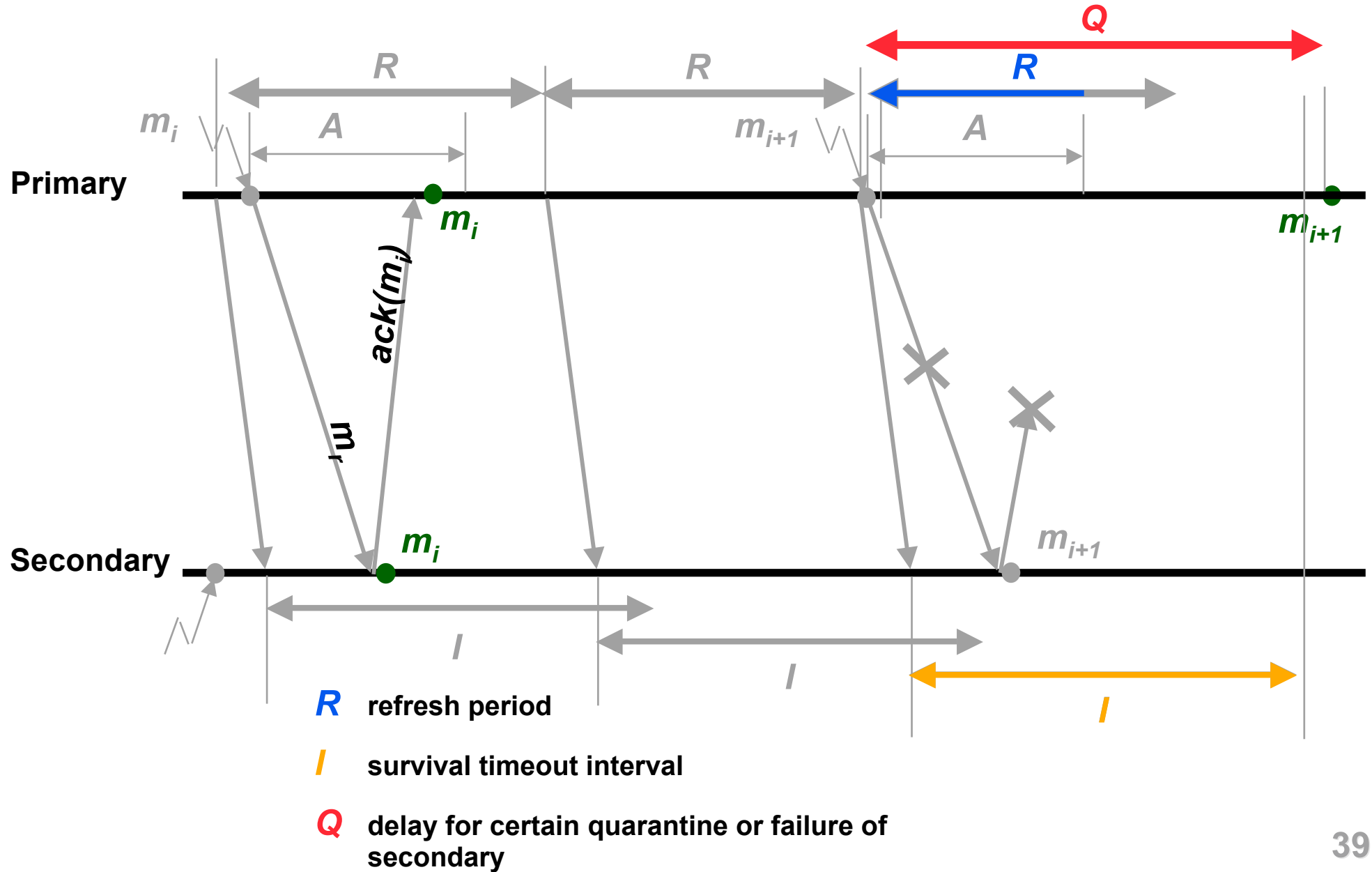
# Quarantine Control Protocol



# Quarantine Control Protocol



# Quarantine Control Protocol



# Choice of Value for $Q$

Need:  $T_P(t_3) < T_P(t_1) + Q$

Equivalently:  $Q \geq T_P(t_3) - T_P(t_1)$

Now:  $(t_3 - t_1) = (t_3 - t_2) + (t_2 - t_1)$

and:

$$(t_3 - t_2) \leq I(1 + \epsilon)$$

$$(t_2 - t_1) \leq \epsilon \quad (\text{fail-aware datagram})$$

so:

$$(t_3 - t_1) \leq \epsilon + I(1 + \epsilon)$$

but:

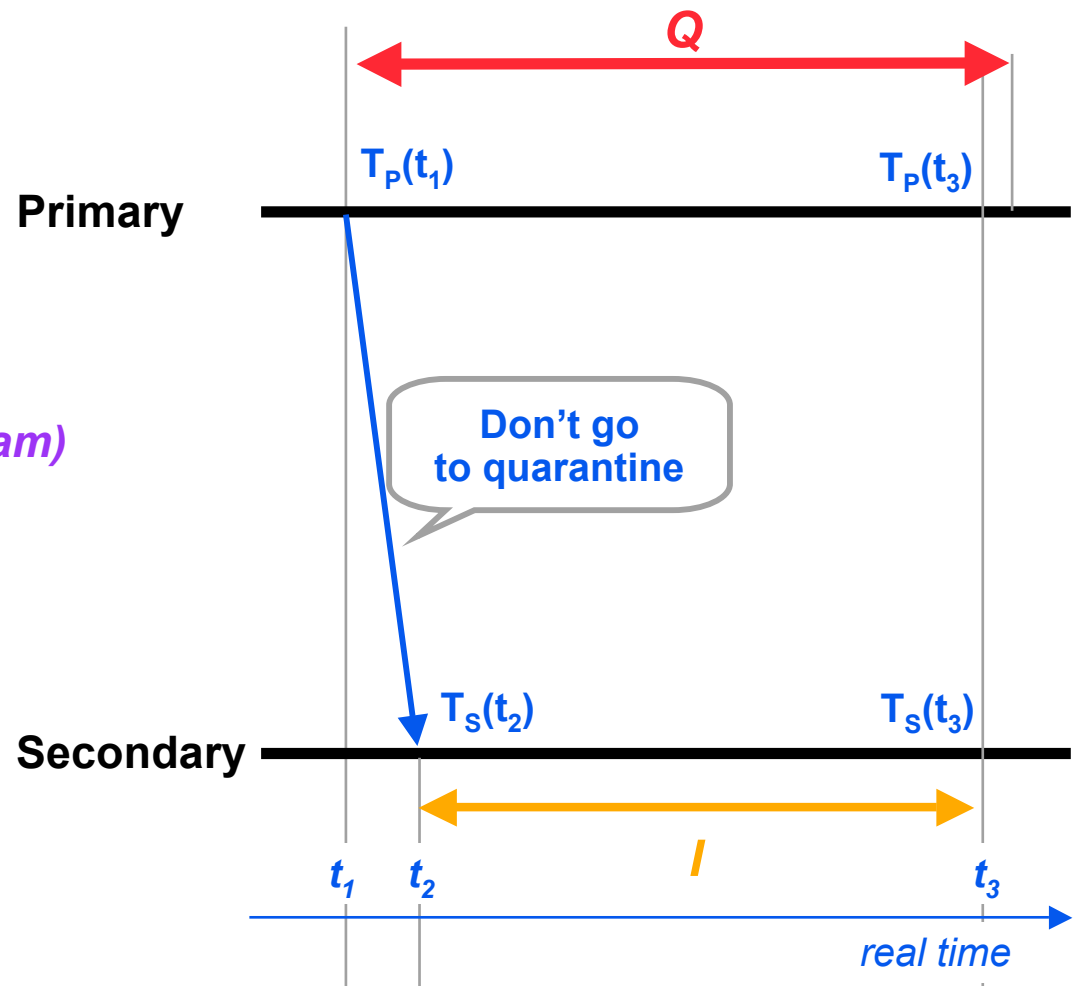
$$T_P(t_3) - T_P(t_1) \leq (t_3 - t_1)(1 + \epsilon)$$

Therefore, must choose  $Q$  such that:

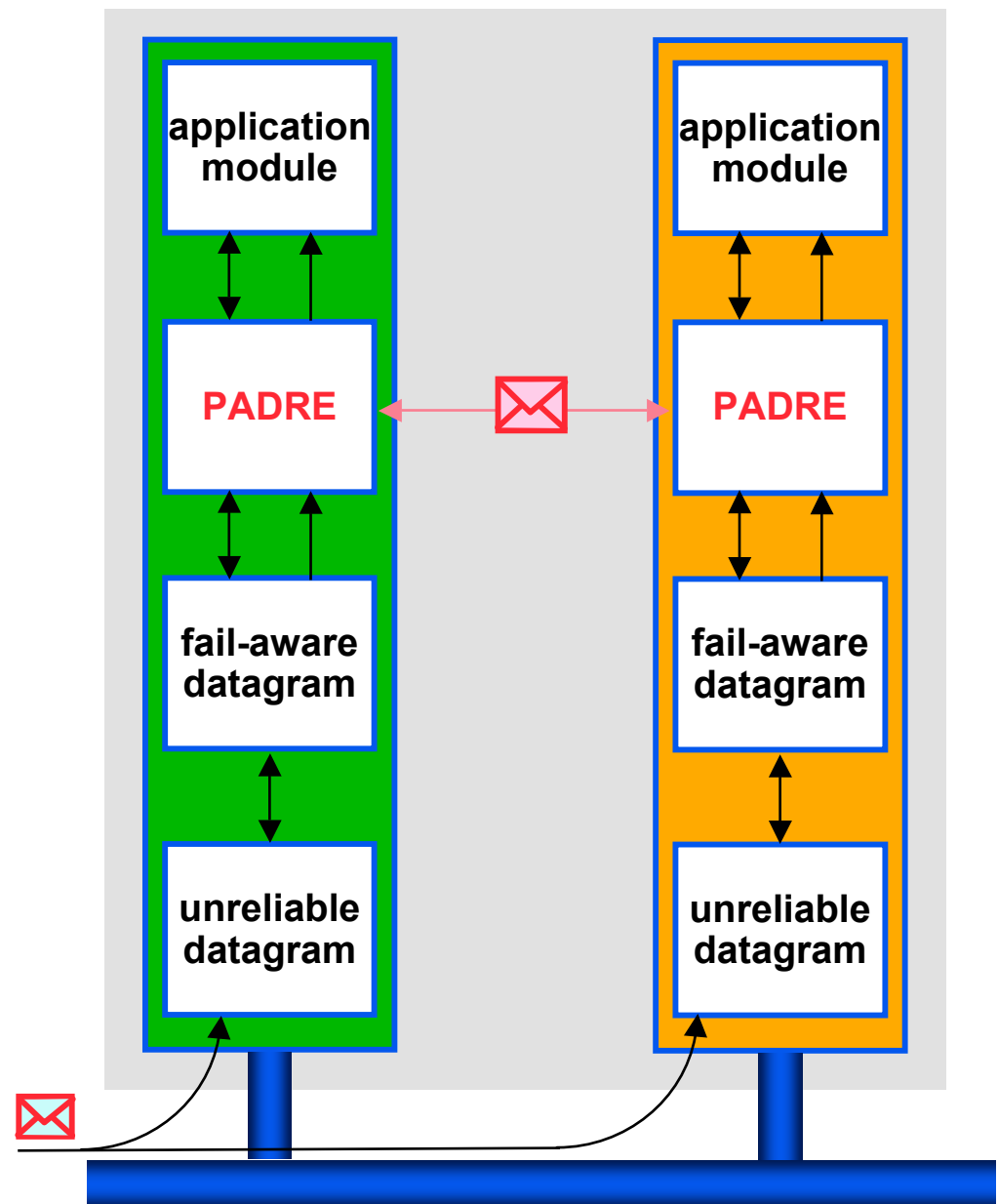
$$Q \geq [\epsilon + I(1 + \epsilon)](1 + \epsilon)$$

or:

$$Q \geq \epsilon(1 + \epsilon) + I(1 + 2\epsilon)$$

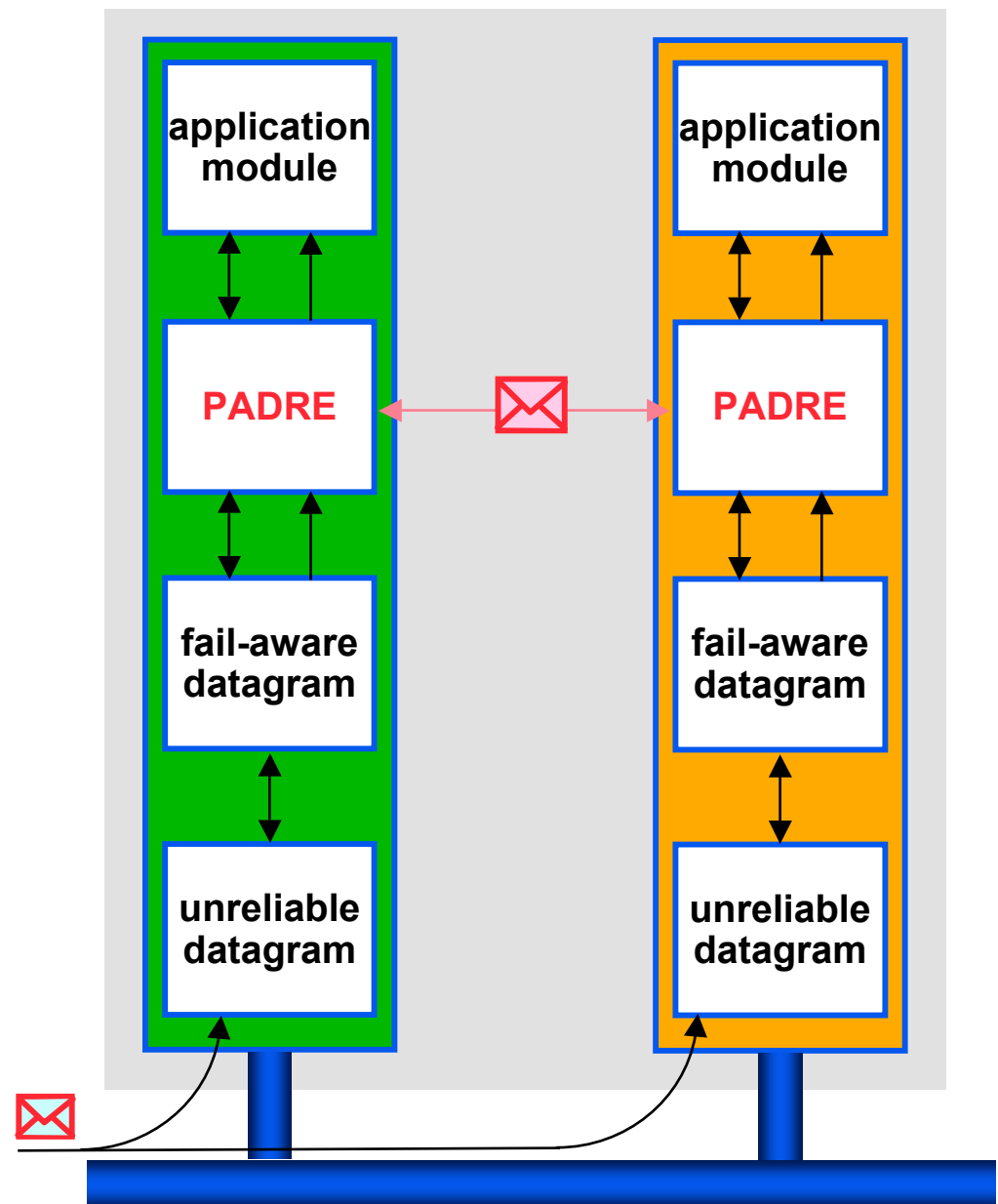


# Unique Primary Property



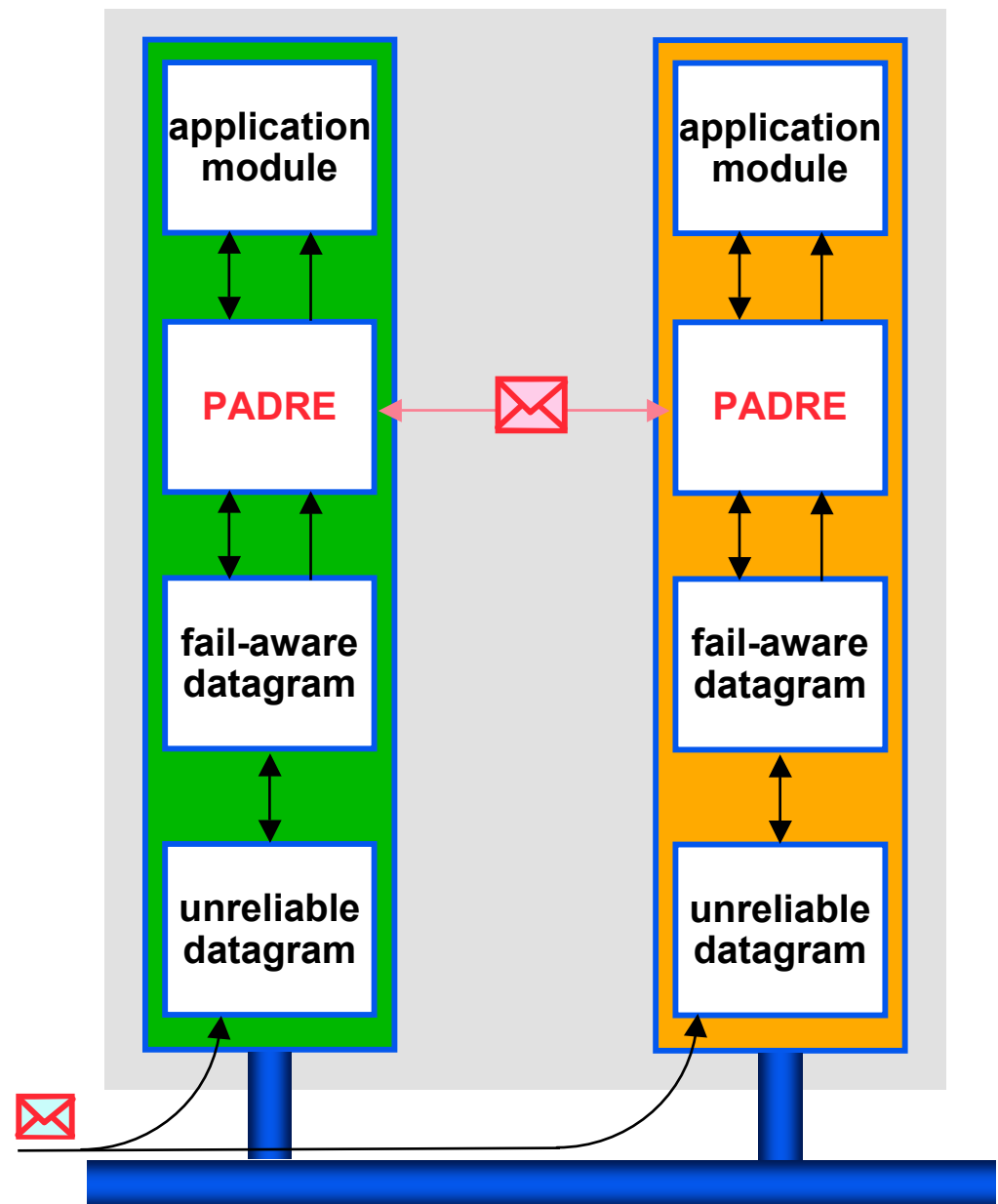
# Unique Primary Property

- Unique Primary (UP): at any instant, only one unit is in the primary mode



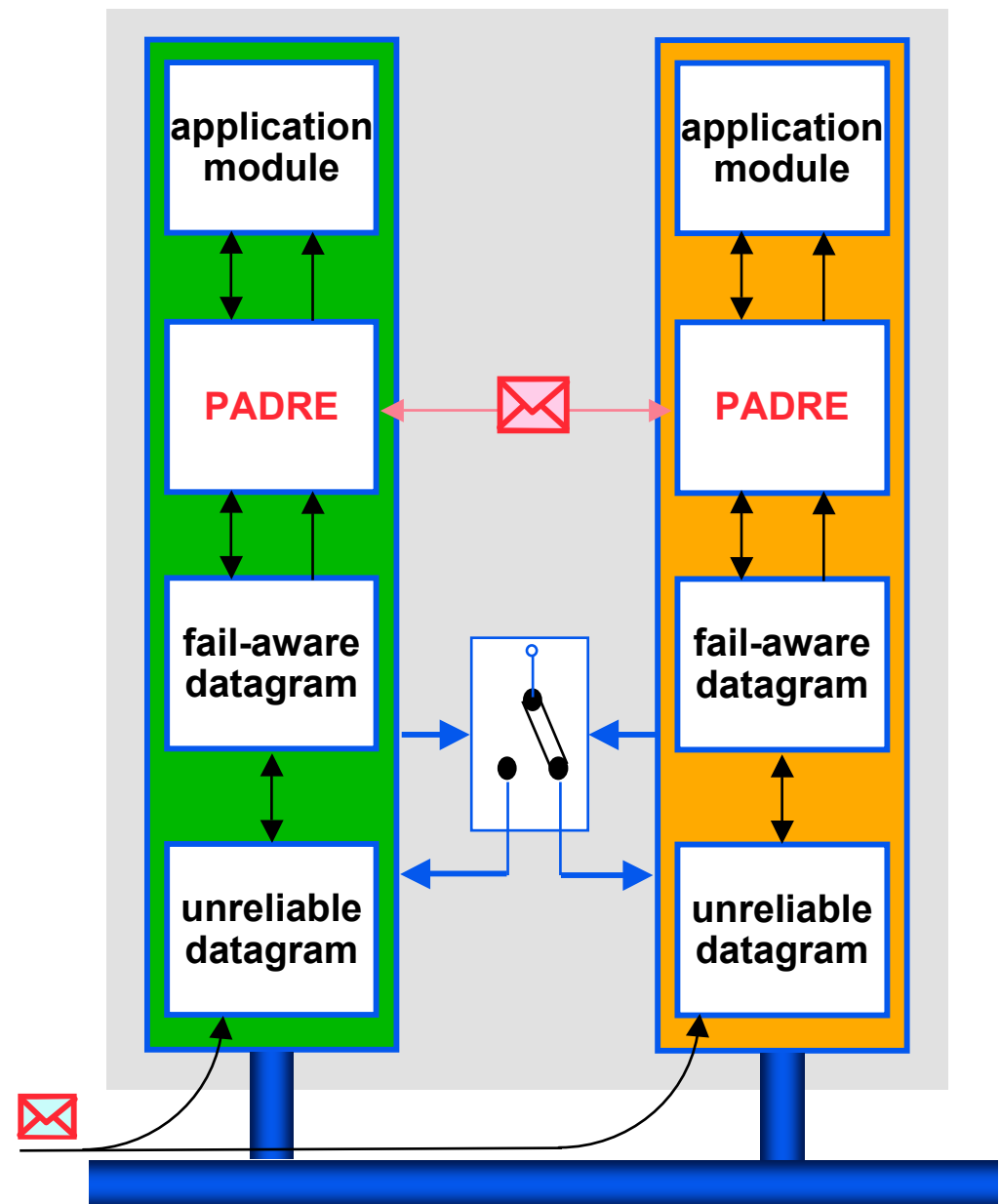
# Unique Primary Property

- **Unique Primary (UP):** at any instant, only one unit is in the primary mode
  - ➔ *Software implementation would require third party to allow majority election of a leader*



# Unique Primary Property

- **Unique Primary (UP):** at any instant, only one unit is in the primary mode
  - ➔ *Software implementation would require third party to allow majority election of a leader*
  - ➔ *Hardware implementation by means of a bistable safety relay*



# De-quarantine Protocol

---

[Bondavalli *et al.* 1998]

**Objective:** secondary to revert from quarantine to standby, to resume its role as a back-up

**Principle:**

- transfer state of primary to secondary
- in general case, state cannot be transferred in single message
- state of primary may be updated while transfer is being carried out

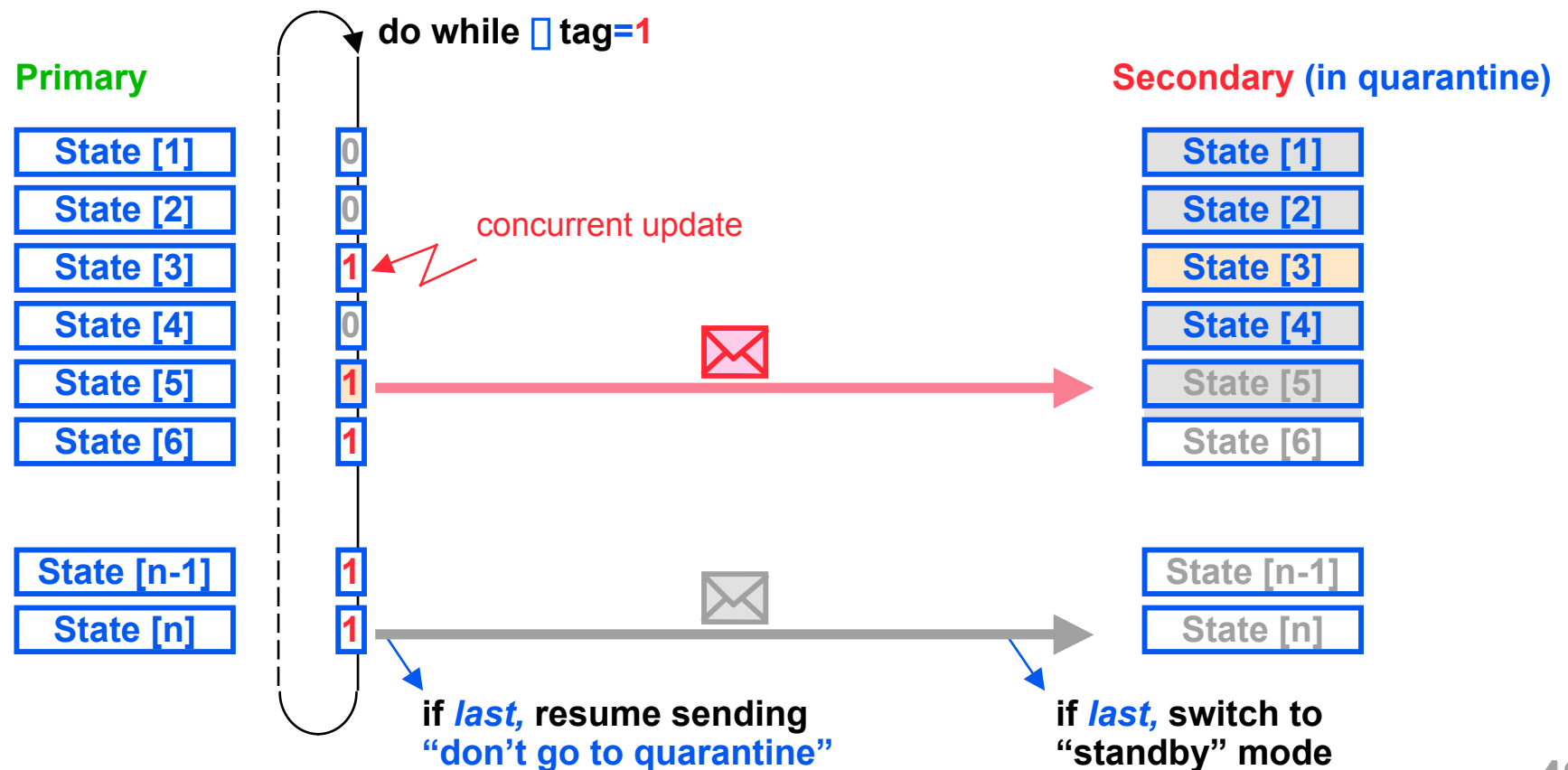
# De-quarantine Protocol

[Bondavalli et al. 1998]

**Objective:** secondary to revert from quarantine to standby, to resume its role as a back-up

**Principle:**

- transfer state of primary to secondary
- in general case, state cannot be transferred in single message
- state of primary may be updated while transfer is being carried out



# Conclusion

---

# Conclusion

---

## ■ Timed asynchronous model

- safety does rely on coverage of synchronous assumptions
- progress can be made when system behaves “as if” it were synchronous
- appropriate model for designing fail-safe distributed systems

# Conclusion

---

## ■ Timed asynchronous model

- safety does rely on coverage of synchronous assumptions
- progress can be made when system behaves “as if” it were synchronous
- appropriate model for designing fail-safe distributed systems

## ■ Asymmetric redundancy management

- tolerance of *potential* inconsistency
- fault-tolerance temporarily sacrificed to guarantee safety

# Conclusion

---

## ■ Timed asynchronous model

- safety does rely on coverage of synchronous assumptions
- progress can be made when system behaves “as if” it were synchronous
- appropriate model for designing fail-safe distributed systems

## ■ Asymmetric redundancy management

- tolerance of *potential* inconsistency
- fault-tolerance temporarily sacrificed to guarantee safety

## ■ Application

- automatic subway (Canarsie Line) in New York, due to start service in 2004

# Bibliography

---

- [Bondavalli et al. 1998] A. Bondavalli, F. D. Giandomenico, F. Grandoni, D. Powell and C. Rabéjac, “State Restoration in a COTS-based N-Modular Architecture”, in *1st Int. Symp. on Object-Oriented Real-Time Distributed Computing (ISORC'98)*, (Kyoto, Japan), pp.174-83, IEEE Computer Society Press, 1998.
- [Cristian & Fetzer 1998] F. Cristian and C. Fetzer, “The Timed Asynchronous System Model”, in *28th Int. Symp. on Fault-Tolerant Computing (FTCS-28)*, (Munich, Germany), pp.140-9, IEEE Computer Society Press, 1998.
- [Essamé et al. 1999] D. Essamé, J. Arlat and D. Powell, “PADRE: a Protocol for Asymmetric Duplex REdundancy”, in *IFIP 7th Working Conf. on Dependable Computing in Critical Applications (DCCA-7)*, (San Jose, CA, USA), pp.213-32, 1999.
- [Fetzer & Cristian 1997] C. Fetzer and F. Cristian, “Fail-Awareness: An Approach to Construct Fail-Safe Applications”, in *27th Int. Symp. on Fault-Tolerant Computing (FTCS-27)*, (Seattle, WA), pp.282-91, IEEE Computer Society Press, 1997.
- [Powell 1992] D. Powell, “Failure Mode Assumptions and Assumption Coverage”, in *22nd Int. Symp. on Fault-Tolerant Computing (FTCS-22)*, (Boston, MA, USA), pp.386-95, IEEE Computer Society Press, 1992.
- [Powell 1994] D. Powell, “Distributed Fault-Tolerance — Lessons from Delta-4”, *IEEE Micro*, 14 (1), pp.36-47, February 1994.
- [Powell et al. 1999] D. Powell, J. Arlat, L. Beus-Dukic, A. Bondavalli, P. Coppola, A. Fantechi, E. Jenn, C. Rabéjac and A. Wellings, “GUARDS: A Generic Upgradable Architecture for Real-time Dependable Systems”, *IEEE Transactions on Parallel and Distributed Systems*, 10 (6), pp.580-99, 1999.