Time

Lakshmi Ganesh
(slides borrowed from
Maya Haridasan,
Michael George)

The Problem

Given a collection of processes that can...

- only communicate with significant latency
- only measure time intervals approximately
- fail in various ways

... we want to construct a shared notion of time

The Problem

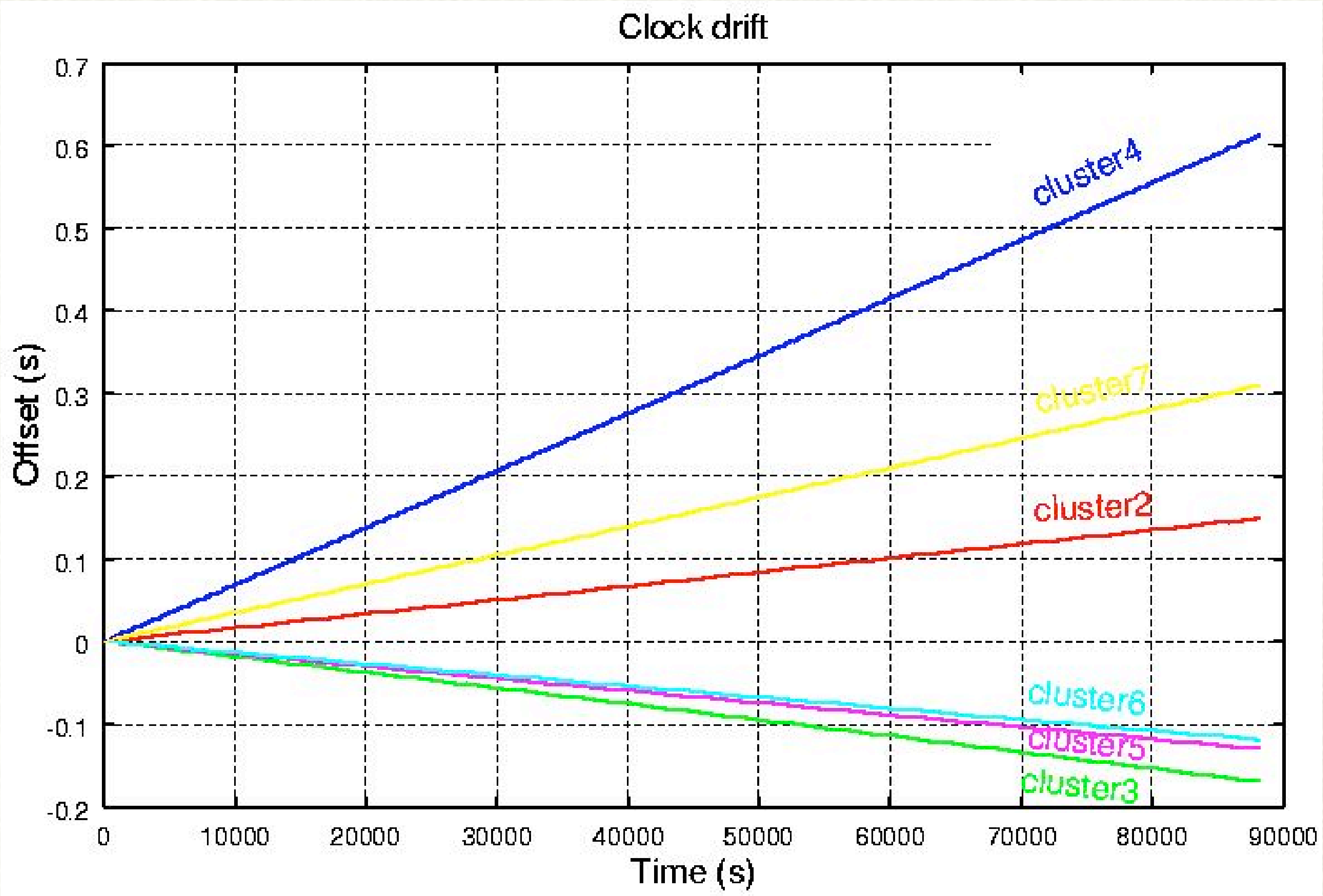
Given a collection of processes that can...

- only communicate with significant latency
- only measure time intervals approximately
- fail in various ways

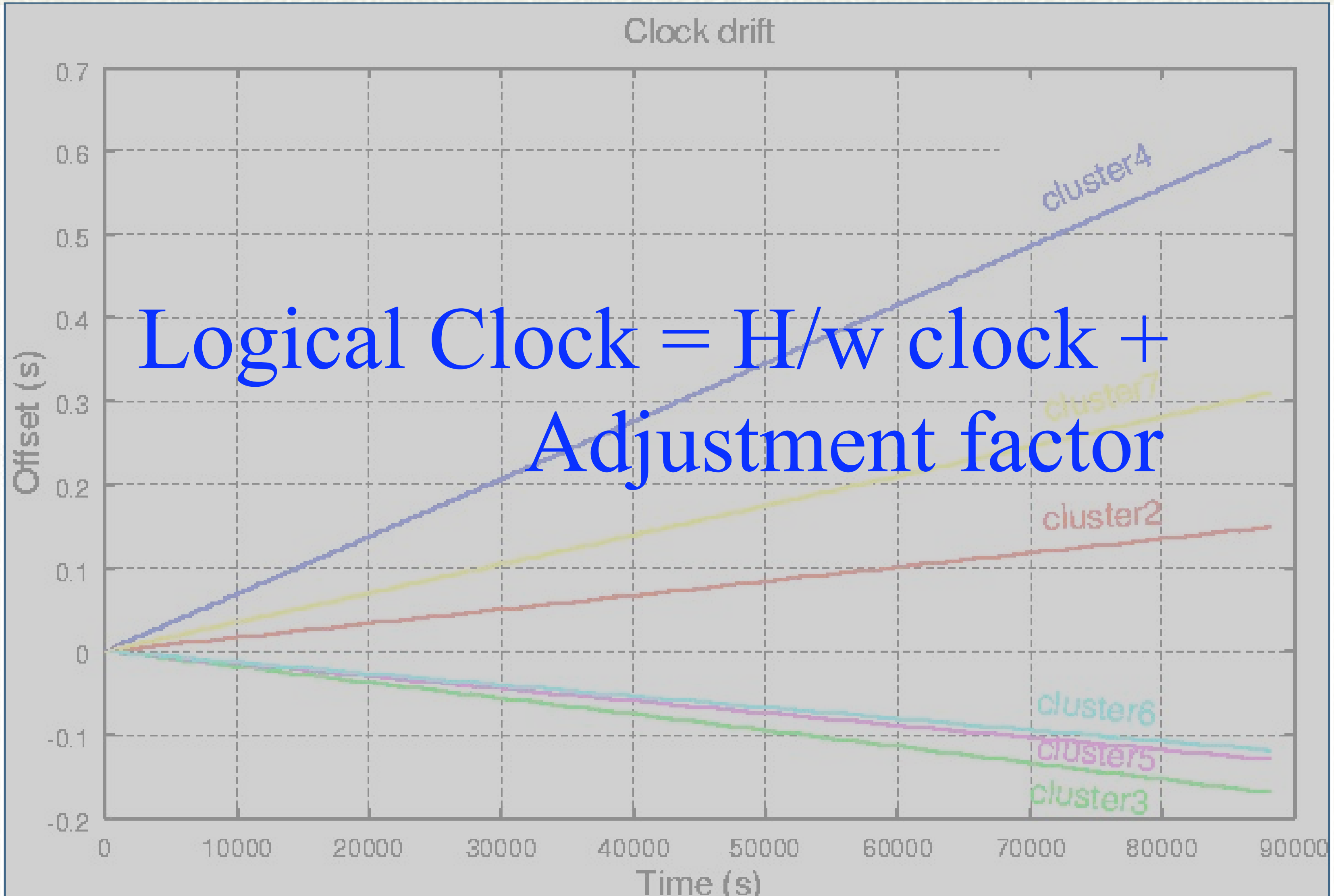
... we want to construct a shared notion of time

But each process has a h/w clock, right??

What's wrong with the clocks?



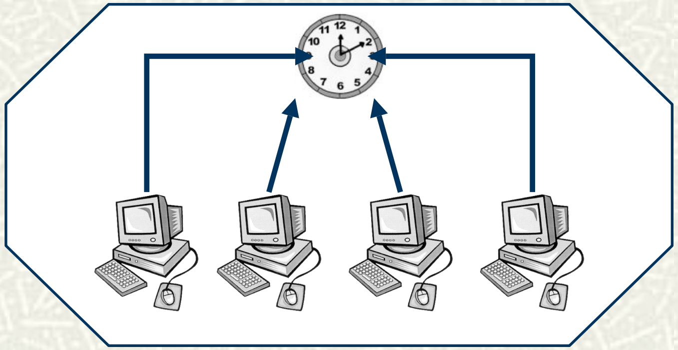
What’s wrong with the clocks?

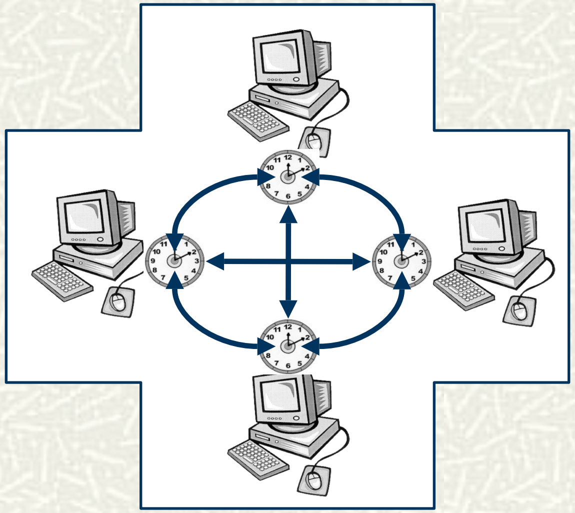


Logical Clock = H/w clock + Adjustment factor

External Vs. Internal Clock Synchronization

- External clock synchronization: 'Adjust' clocks with respect to an external time reference
  - Accuracy: how close logical time is to real time





- Internal clock synchronization (ICS): 'Adjust' clocks among themselves
  - Precision: how close the clocks are to each other

Software Clock Synchronization

1. Deterministic → assumes an upper bound on transmission delays (which bounds accuracy) – guarantees some precision
2. Statistical → expectation and standard deviation of the delay distributions are known
3. Probabilistic → no assumptions about delay distributions (gives better accuracy)

Software Clock Synchronization

1. Deterministic → assumes an upper bound on transmission delays (which bounds accuracy) – guarantees some precision

   Realistic?

2. Statistical → expectation and standard deviation of the delay distributions are known
3. Probabilistic → no assumptions about delay distributions (gives better accuracy)

Software Clock Synchronization

1. Deterministic → assumes an upper bound on transmission delays (which bounds accuracy) – guarantees some precision

   Realistic?

2. Statistical → expectation and standard deviation of the delay distributions are known

   Reliable?

3. Probabilistic → no assumptions about delay distributions (gives better accuracy)

Software Clock Synchronization

1. Deterministic → assumes an upper bound on transmission delays (which bounds accuracy) – guarantees some precision

Realistic?

2. Statistical → expectation and standard deviation of the delay distributions are known

Reliable?

3. Probabilistic → no assumptions about delay distributions (gives better accuracy)

Any guarantees?

Today...

- We will discuss two papers that solve ICS:
  - Optimal Clock Synchronization [Srikanth and Toueg '87]
    - Assume reliable network (deterministic)
    - Provide logical clock with optimal agreement
    - Also optimal with respect to failures
  - Probabilistic Internal Clock Synchronization [Cristian and Fetzer '03]
    - Drop requirements on network (probabilistic)
    - Provide very efficient logical clock
    - Only provide probabilistic guarantees

Paper 1: System Model

We assume...

Clock drift is bounded

$$(1 - \rho)(t - s) \leq H_p(t) - H_p(s) \leq (1 + \rho)(t - s)$$

Communication and processing are reliable

$$t_{recv} - t_{send} \leq t_{del}$$

Authenticated messages

will relax this later...

Paper 1: Our Goals

- Property 1 (Agreement):

  $| L_{pi}(t) - L_{pj}(t) | \leq \delta$,

  (δ is the precision of the clock synchronization algorithm)

- Property 2 (Accuracy):

  $(1 - \rho_v)(t - s) + a \leq L_p(t) - L_p(s) \leq (1 + \rho_v)(t - s) + b$

Paper 1: Our Goals

- Property 1 (Agreement):

  $| L_{pi}(t) - L_{pj}(t) | \leq \delta$,

  (δ is the precision of the clock synchronization algorithm)

- Property 2 (Accuracy):

  $(1 - \rho_v)(t - s) + a \leq L_p(t) - L_p(s) \leq (1 + \rho_v)(t - s) + b$

  $\rho_v \neq \rho$

What is optimal accuracy?

Paper 1: Our Goals

Optimal Accuracy

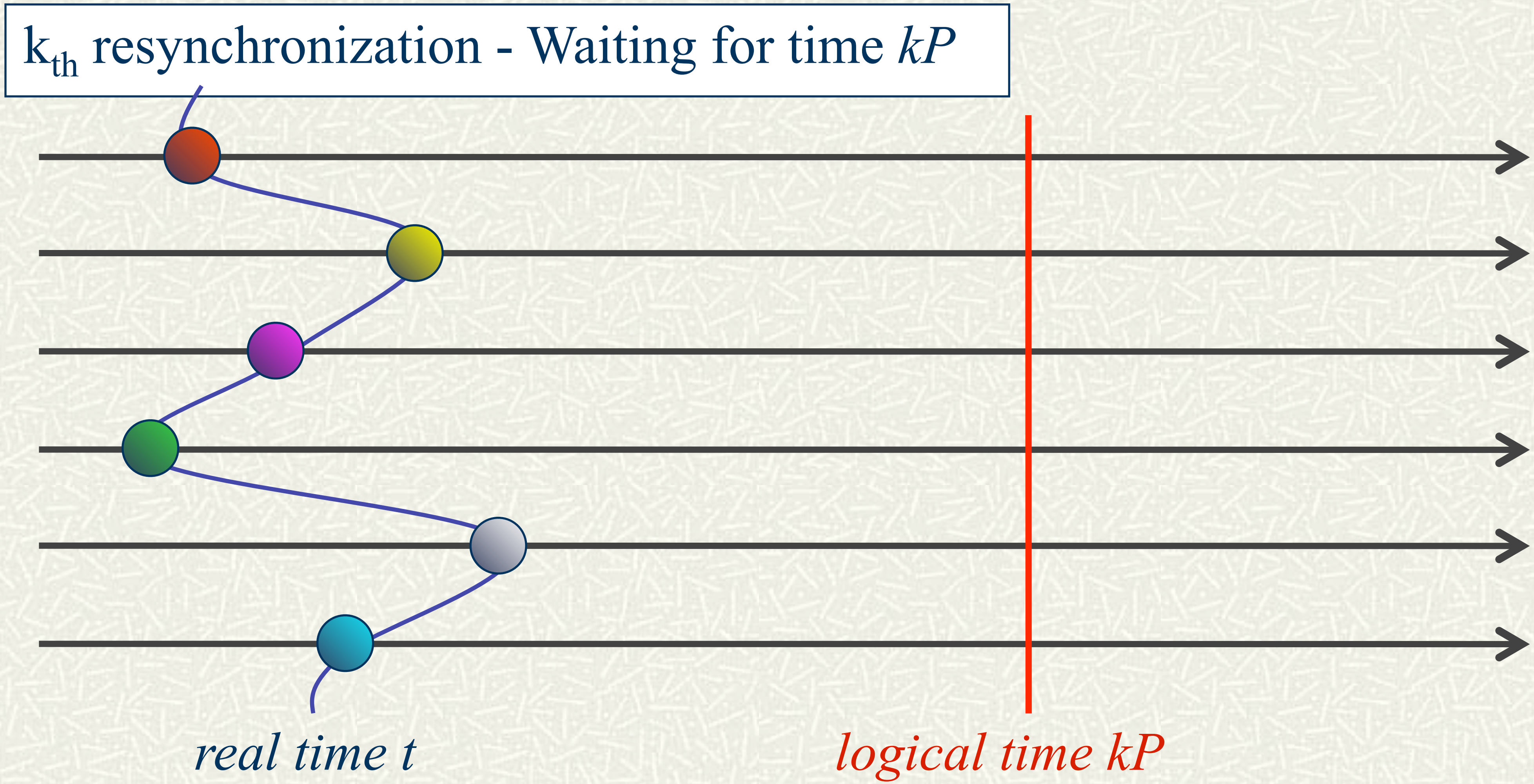
- Drift rate of the synchronized clocks is bounded by the maximum drift rate of correct hardware clocks

$$\rho_v = \rho$$

Fault-tolerant

- Up to f crash failures, performance failures, arbitrary (Byzantine) failures

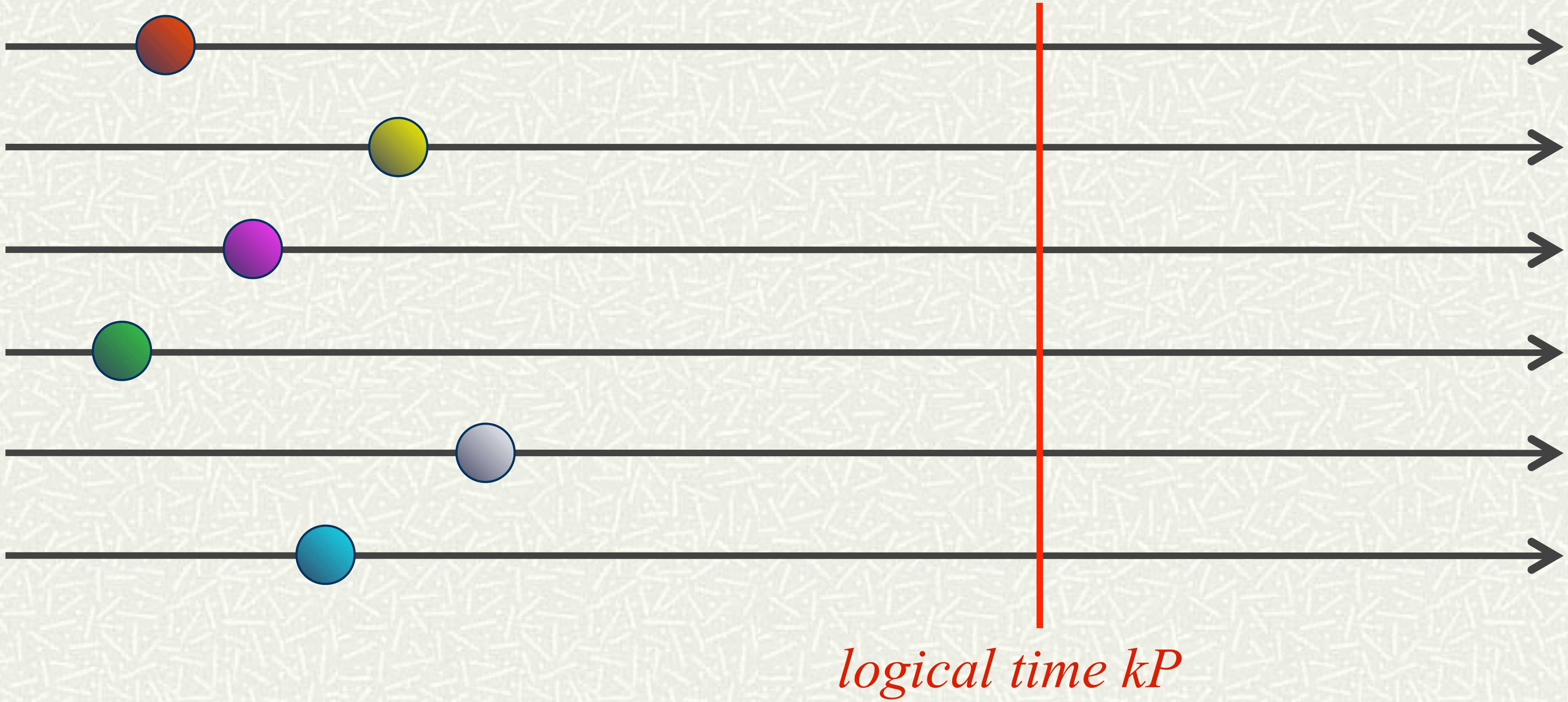
Authenticated Algorithm



P – logical time between resynchronizations

Authenticated Algorithm

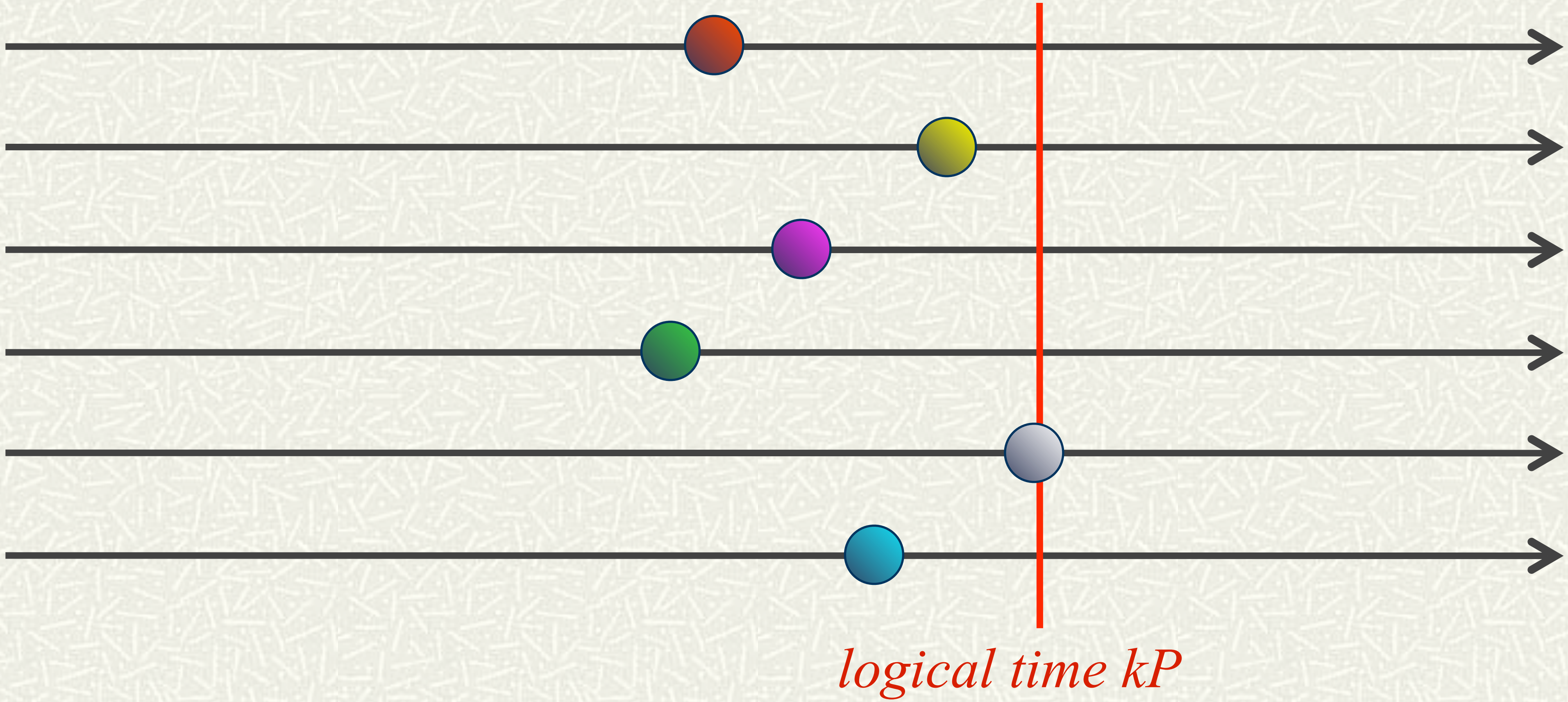
k_{th} resynchronization - Waiting for time *kP*



P – logical time between resynchronizations

Authenticated Algorithm

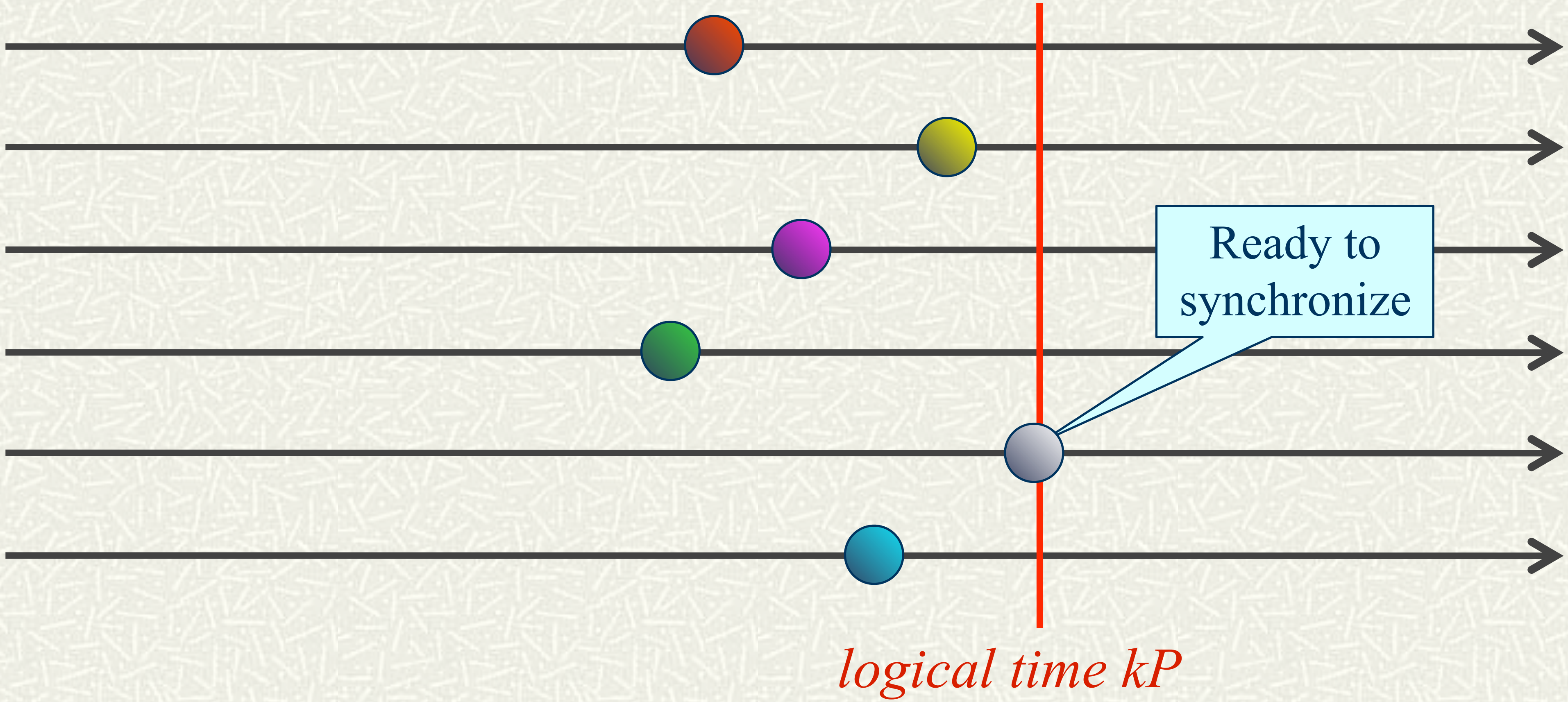
k_{th} resynchronization - Waiting for time *kP*



P – logical time between resynchronizations

Authenticated Algorithm

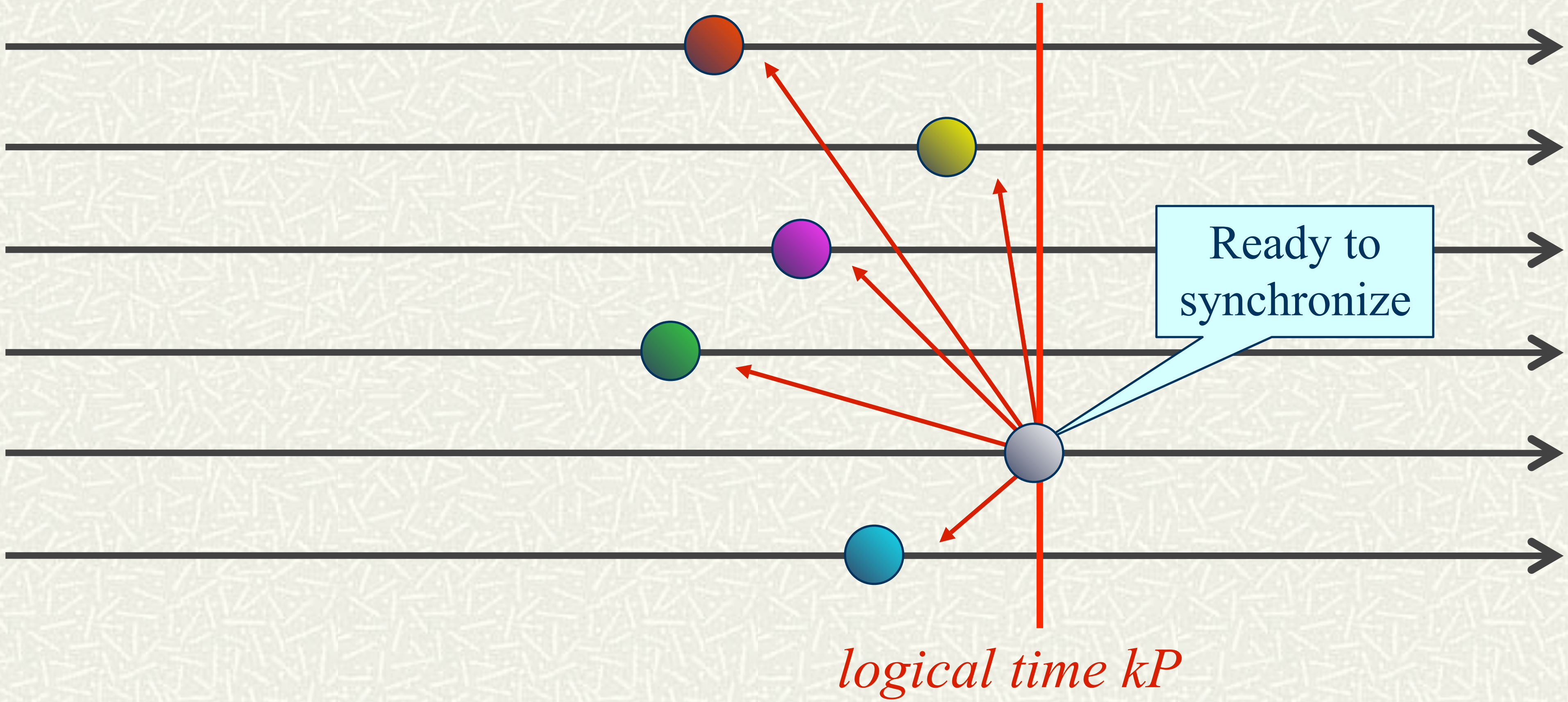
k_{th} resynchronization - Waiting for time *kP*



P – logical time between resynchronizations

Authenticated Algorithm

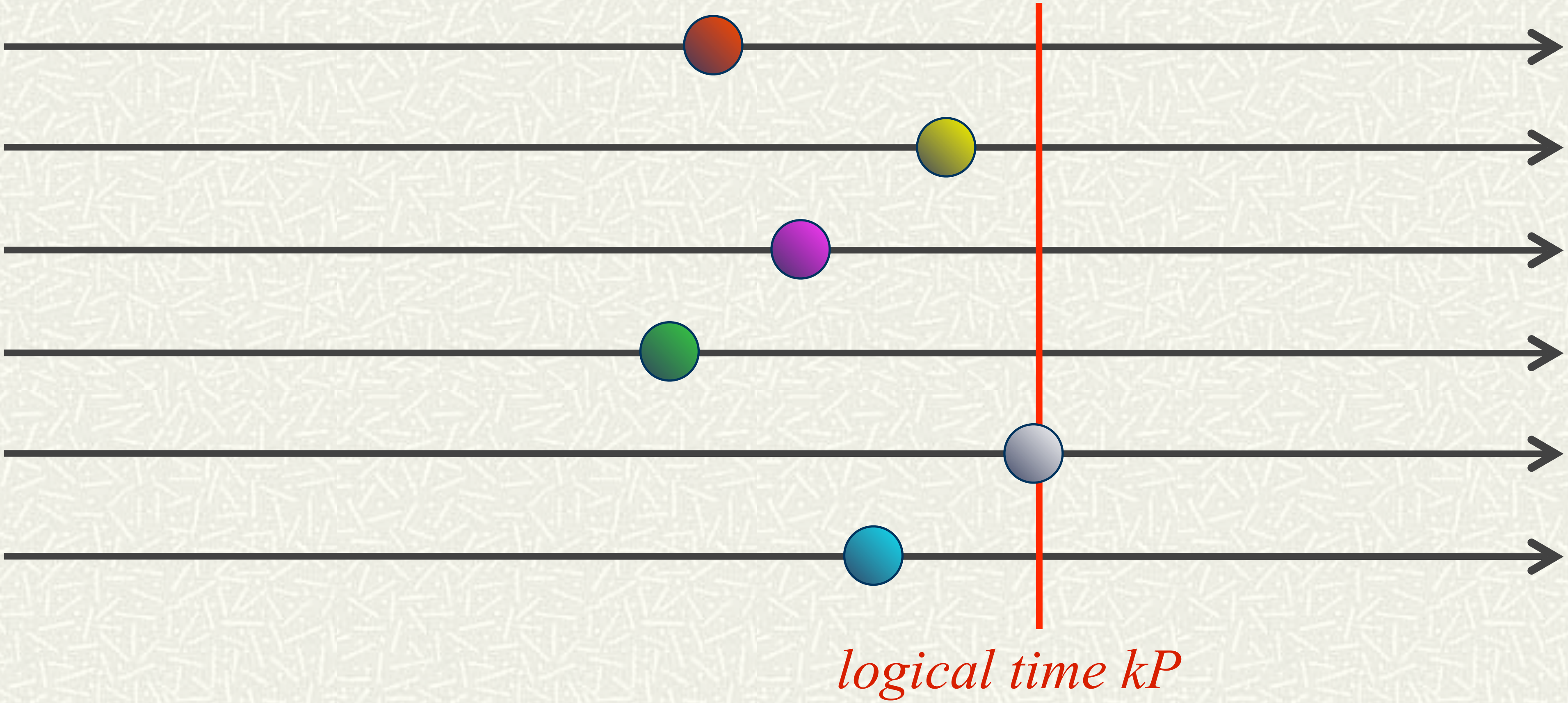
k_{th} resynchronization - Waiting for time *kP*



P – logical time between resynchronizations

Authenticated Algorithm

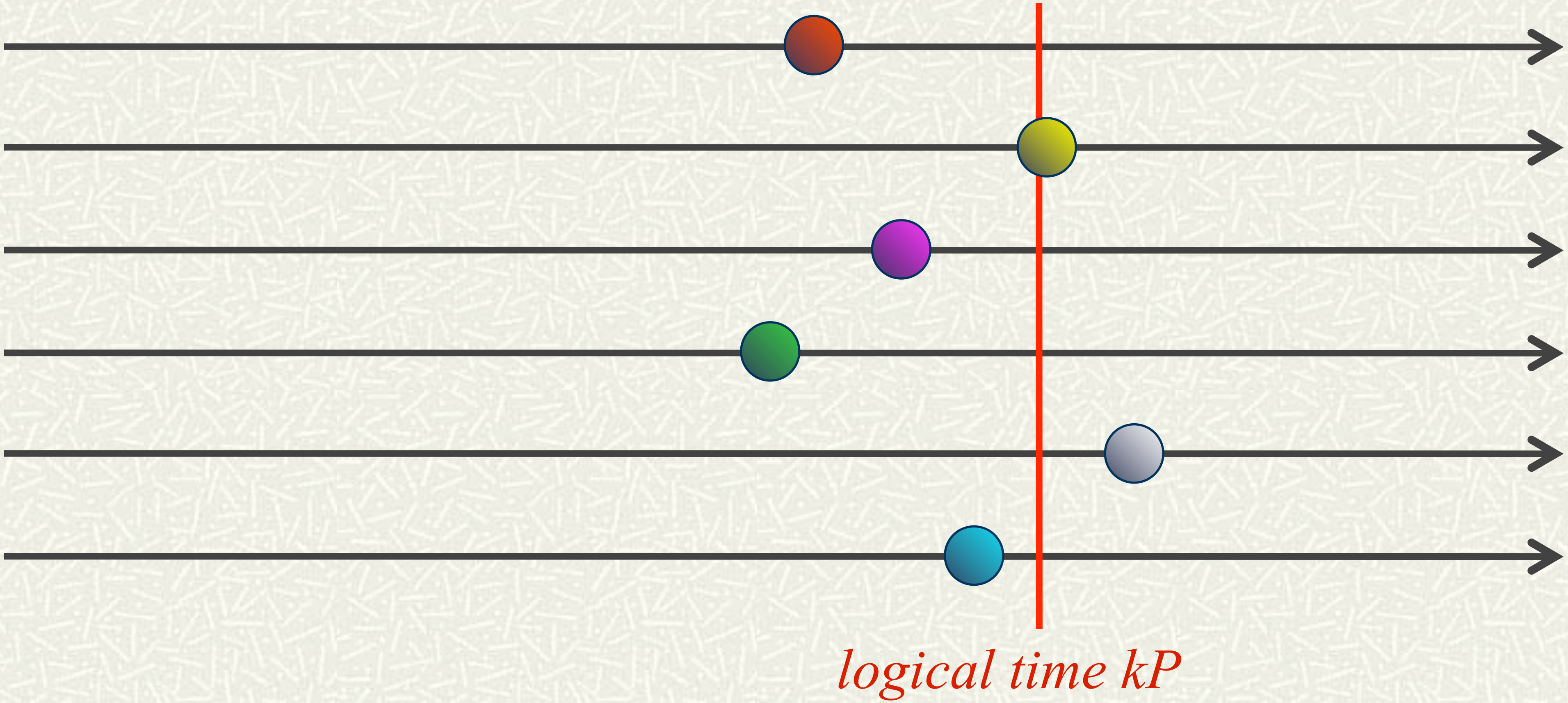
k_{th} resynchronization - Waiting for time *kP*



P – logical time between resynchronizations

Authenticated Algorithm

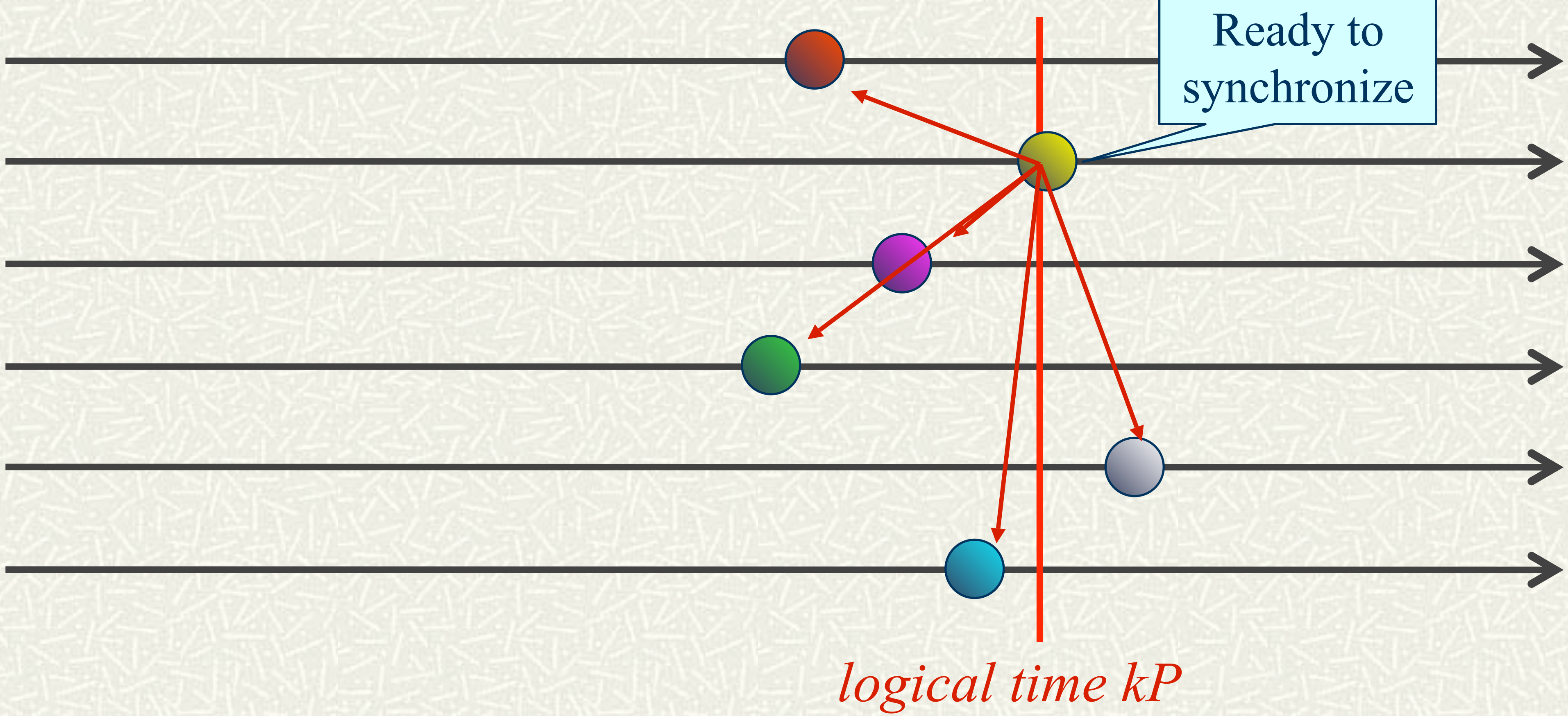
k_{th} resynchronization - Waiting for time *kP*



P – logical time between resynchronizations

Authenticated Algorithm

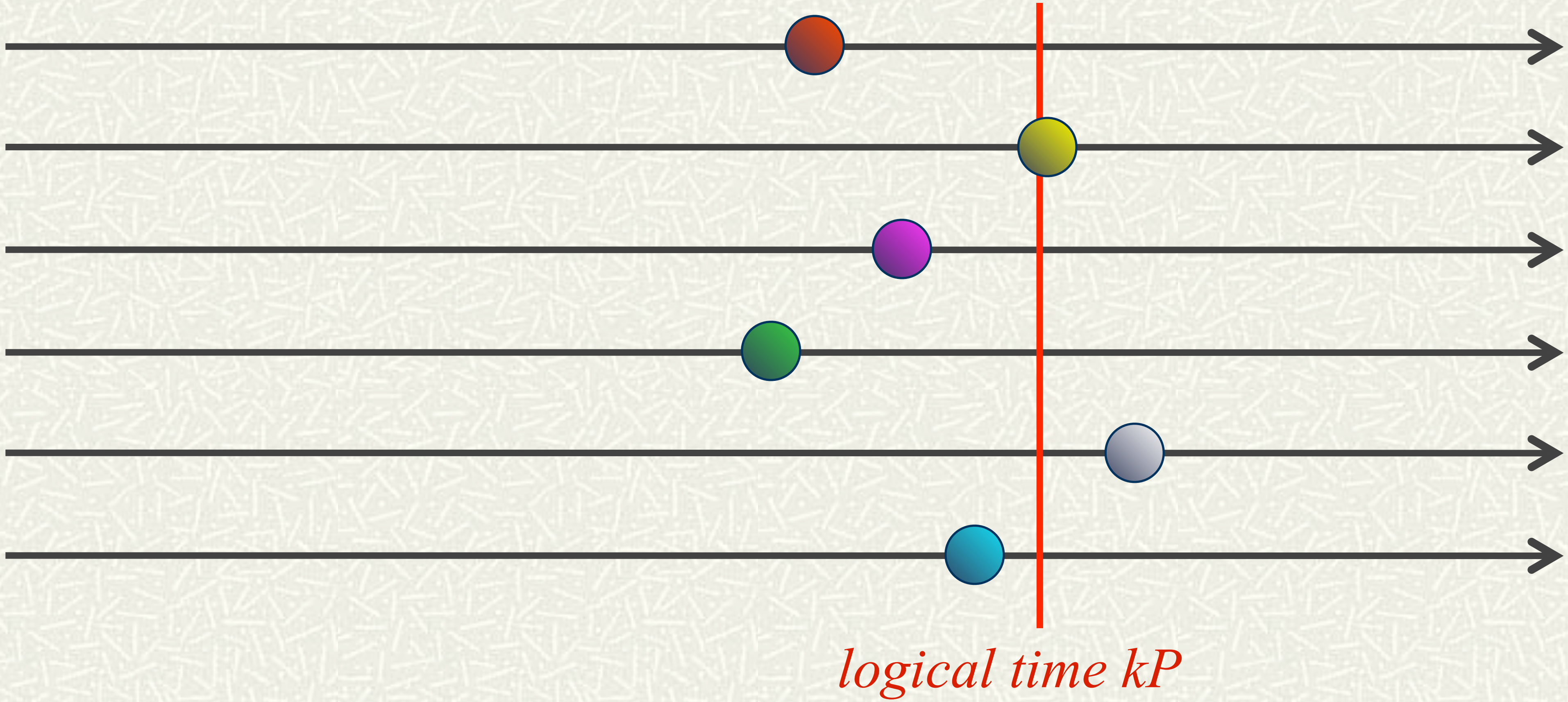
k_{th} resynchronization - Waiting for time *kP*



P – logical time between resynchronizations

Authenticated Algorithm

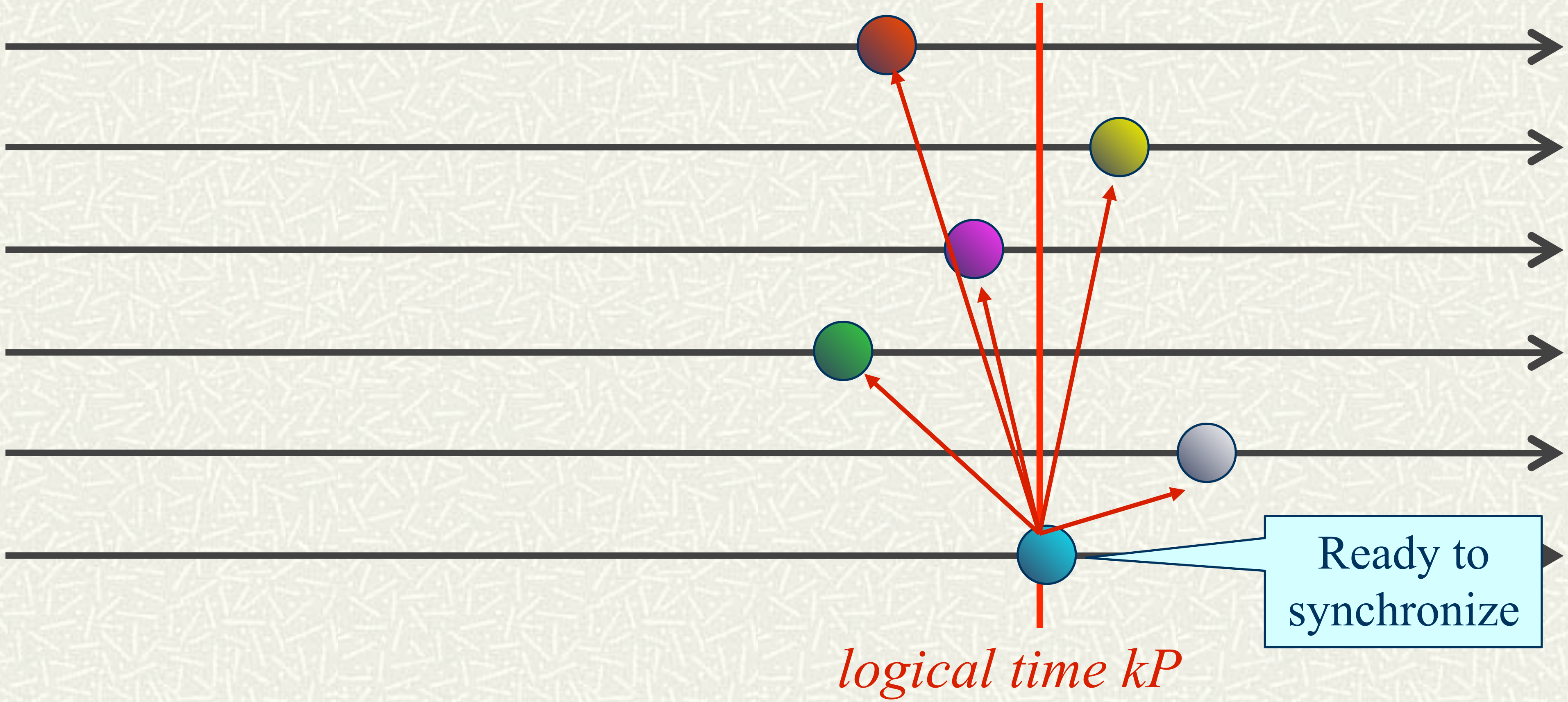
k_{th} resynchronization - Waiting for time *kP*



P – logical time between resynchronizations

Authenticated Algorithm

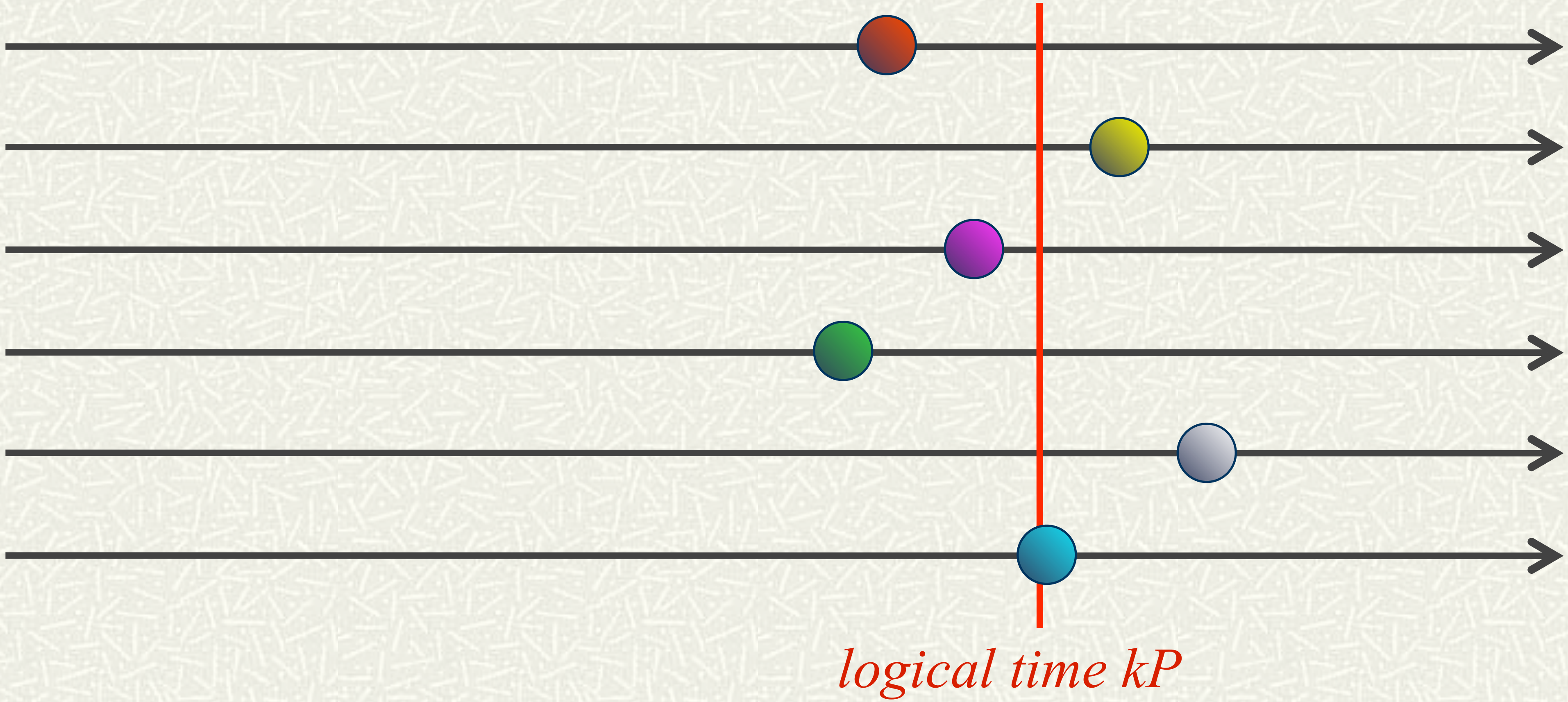
k_{th} resynchronization - Waiting for time *kP*



P – logical time between resynchronizations

Authenticated Algorithm

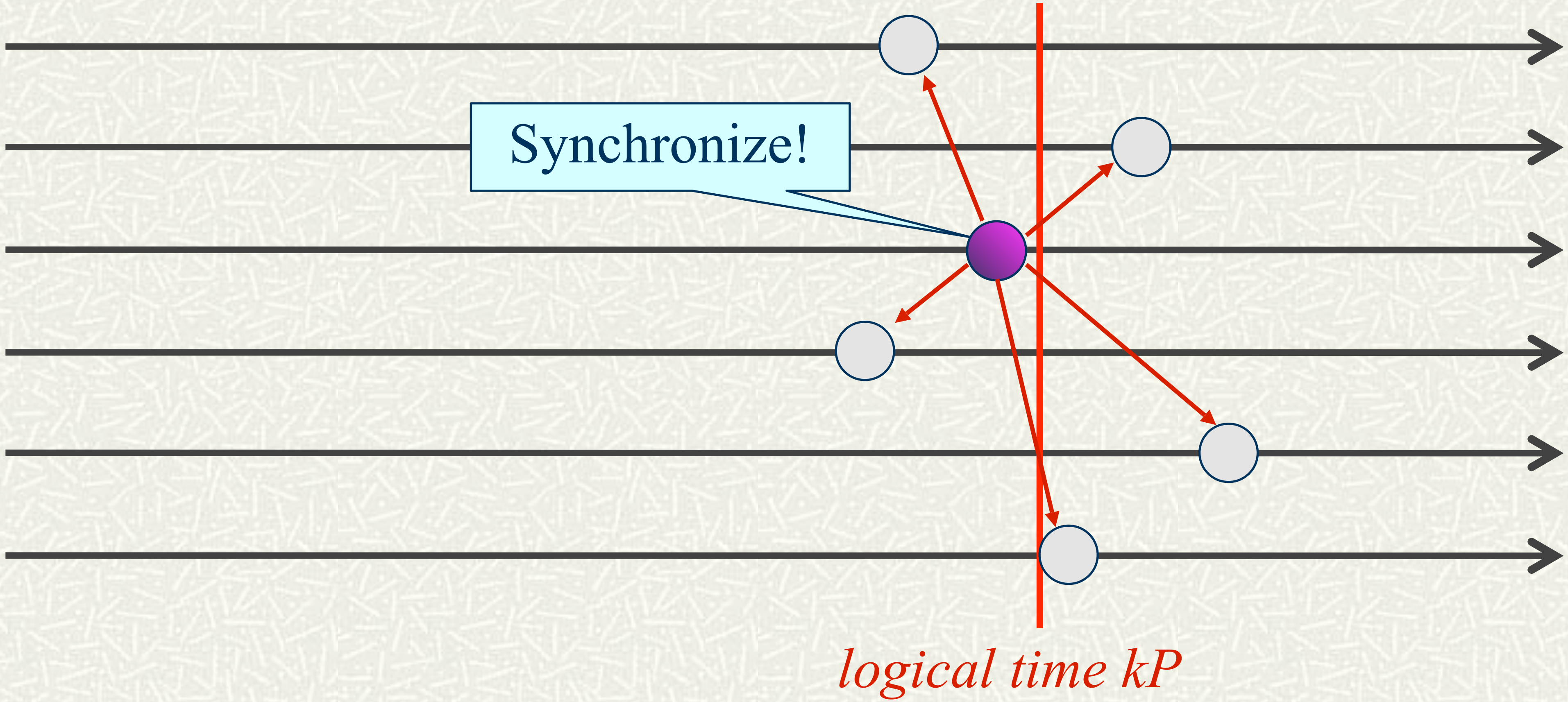
k_{th} resynchronization - Waiting for time *kP*



P – logical time between resynchronizations

Authenticated Algorithm

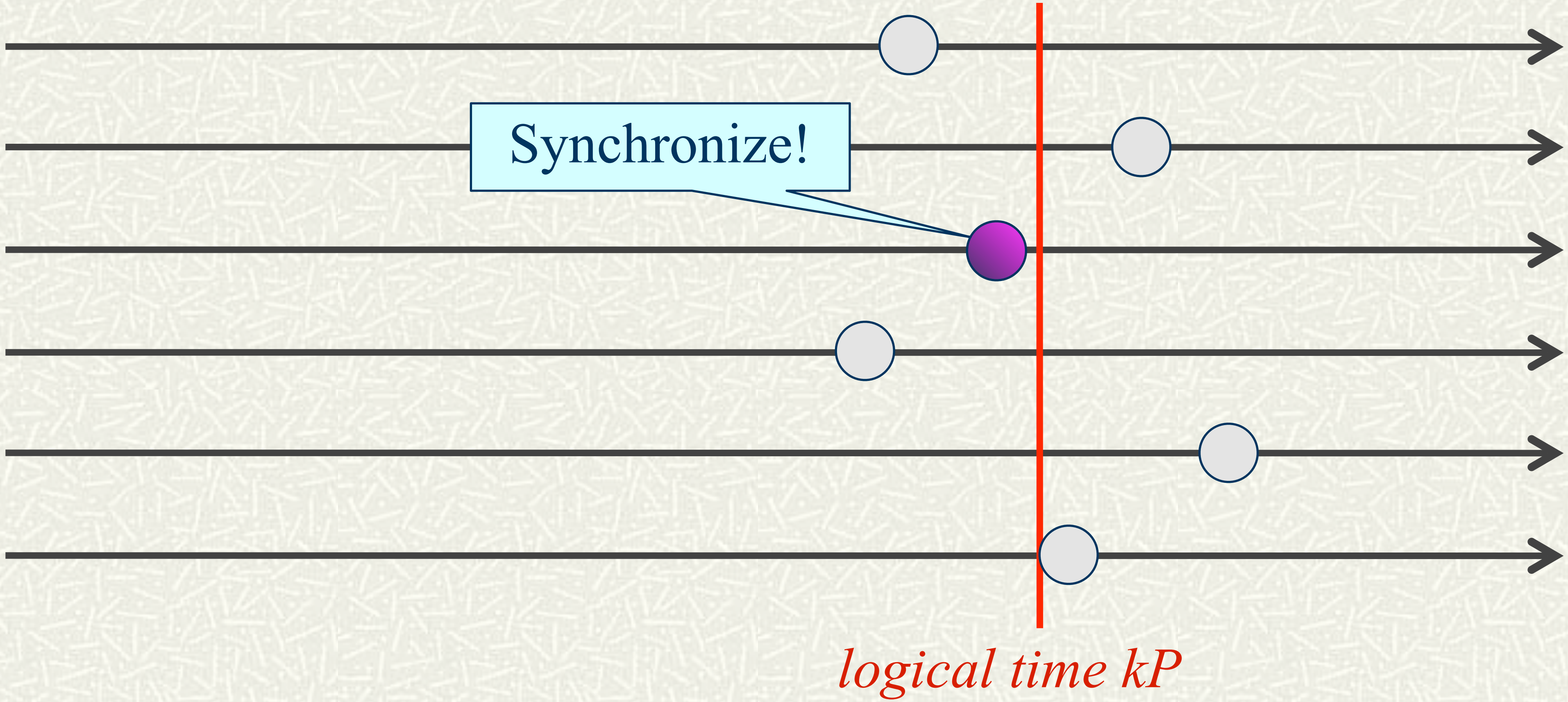
k_{th} resynchronization - Waiting for time *kP*



P – logical time between resynchronizations

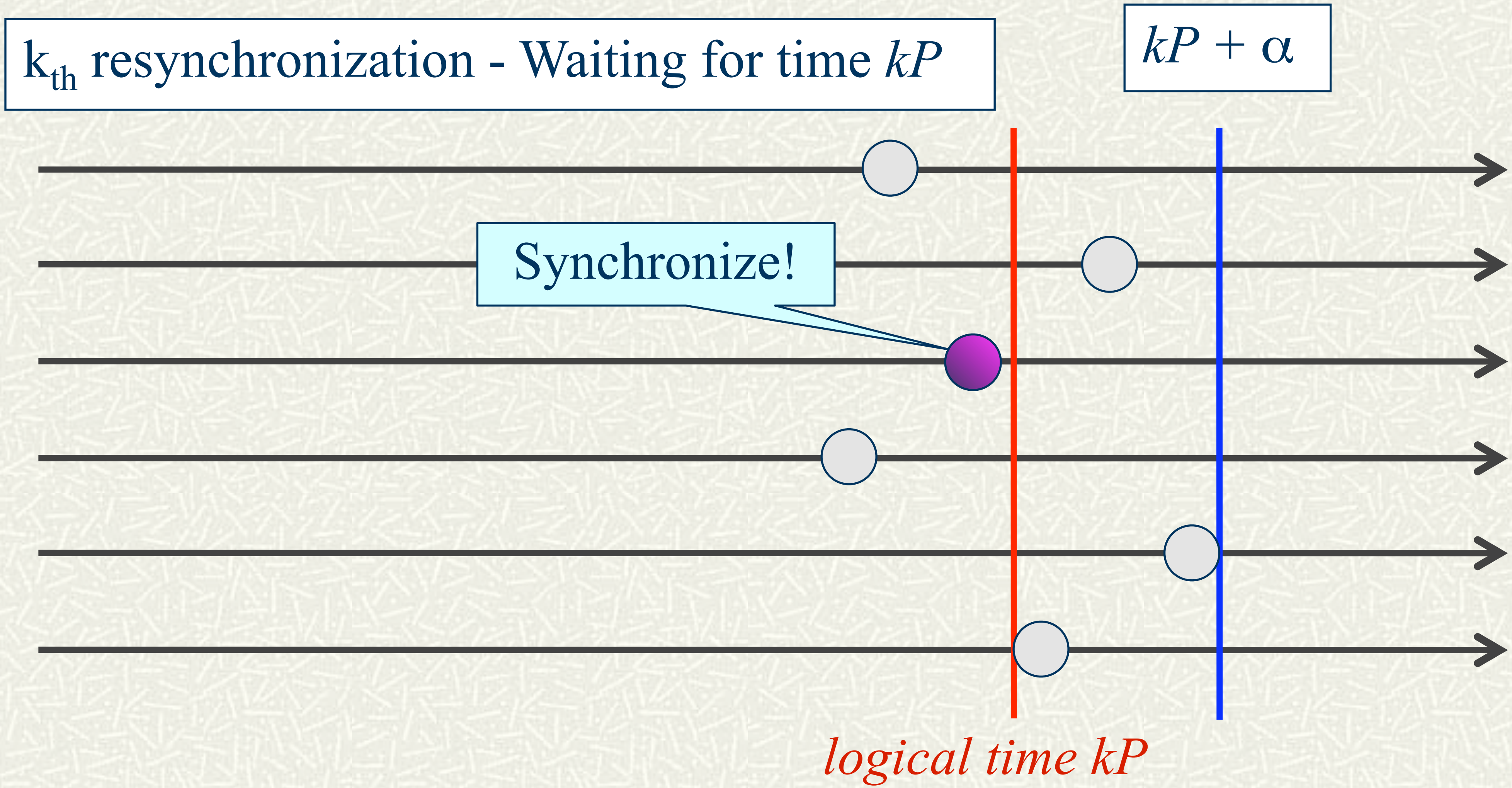
Authenticated Algorithm

k_{th} resynchronization - Waiting for time *kP*



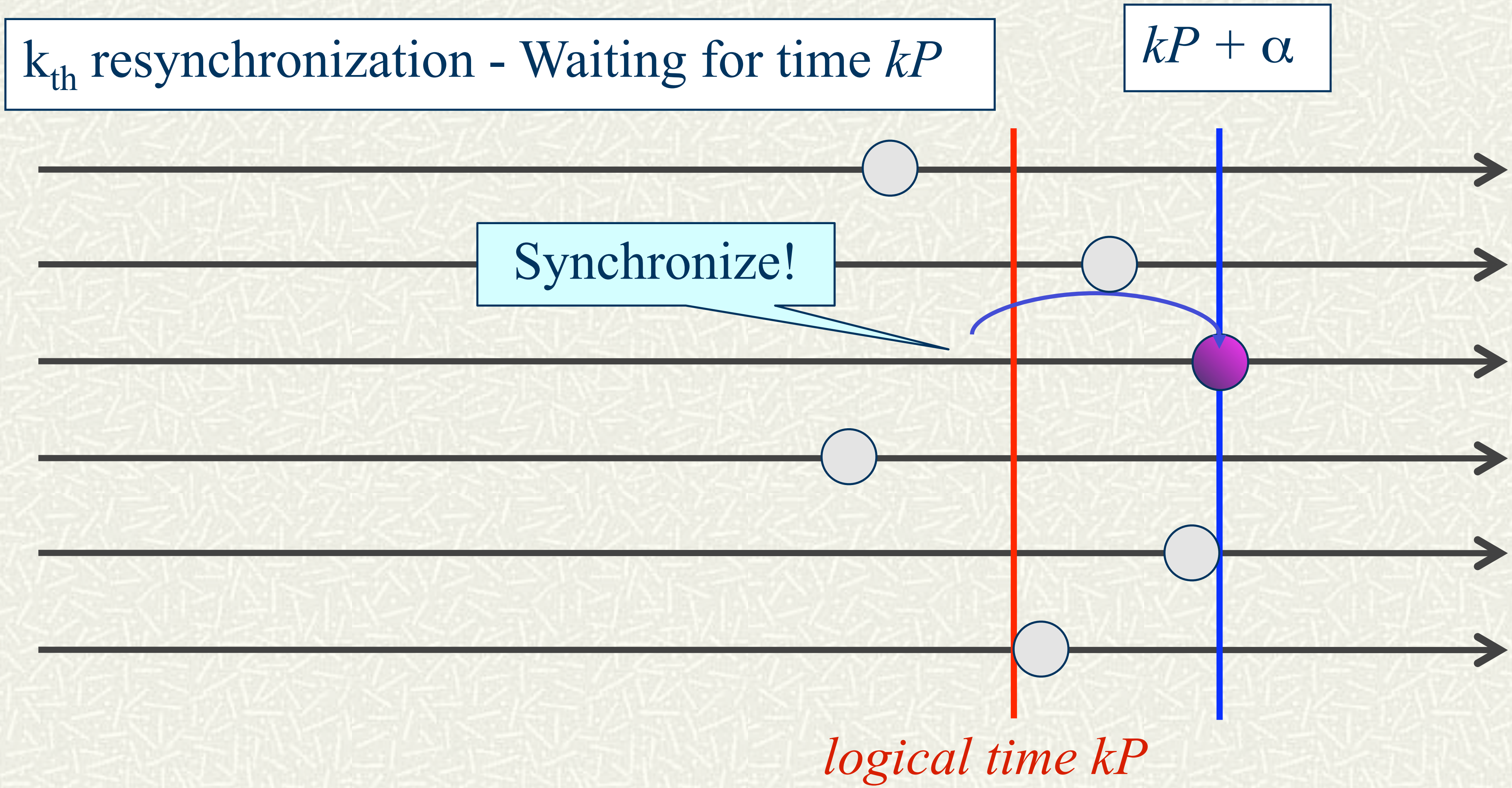
P – logical time between resynchronizations

Authenticated Algorithm



P – logical time between resynchronizations

Authenticated Algorithm



P – logical time between resynchronizations

Achieving Optimal Accuracy

Uncertainty of t_{delay} introduces a difference in the logical time between resynchronizations

→ Reason for non-optimal accuracy

- Solution:
  - Slow down the logical clocks by a factor of

$$\frac{P}{(P - \alpha + \beta)}$$

where $\beta = t_{del} / (2(1 + \rho))$

Authenticated Messages

- **Correctness:**

  If at least $f + 1$ correct processes broadcast messages by time t, then every correct process accepts the message by time $t + t_{del}$

- **Unforgeability:**

  If no correct process broadcasts a message by time t, then no correct process accepts the message by t or earlier

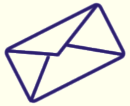
- **Relay:**

  If a correct process accepts the message at time t, then every correct process does so by time $t + t_{del}$

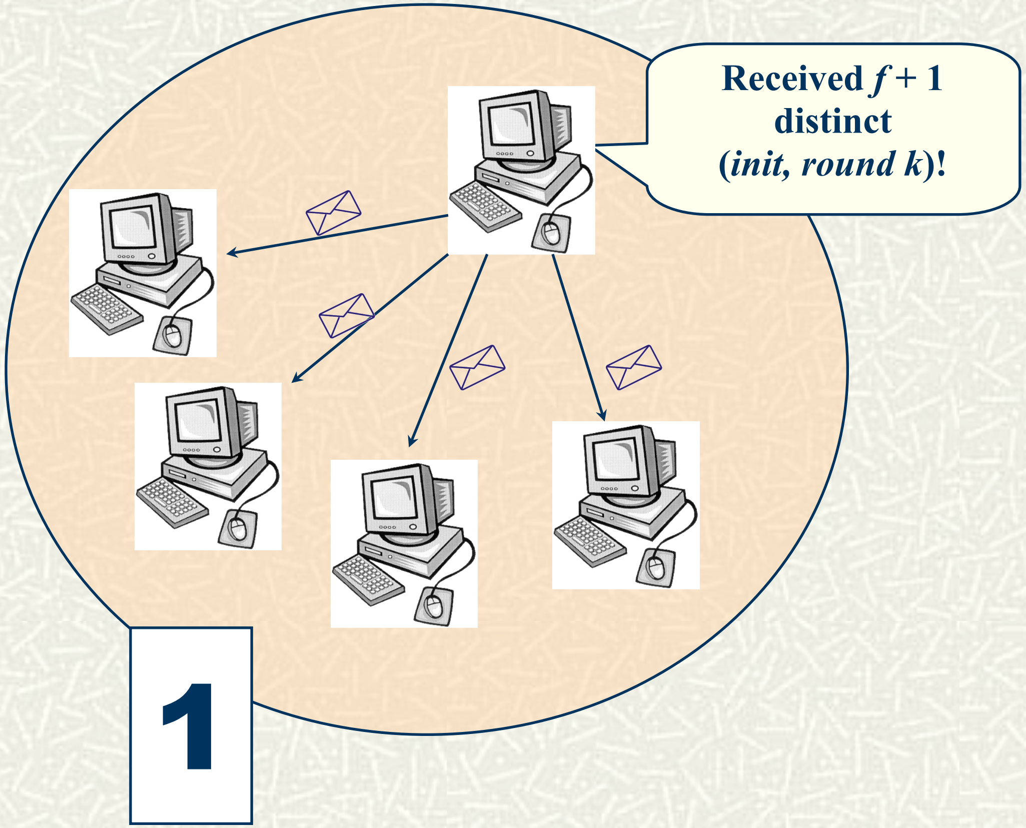
Nonauthenticated Algorithm

- Replace signed communication with a broadcast primitive
  - Primitive relays messages automatically
  - Cost of $O(n^2)$ messages per resynchronization
- New limit on number of faulty processes allowed:
  - $n > 3f$

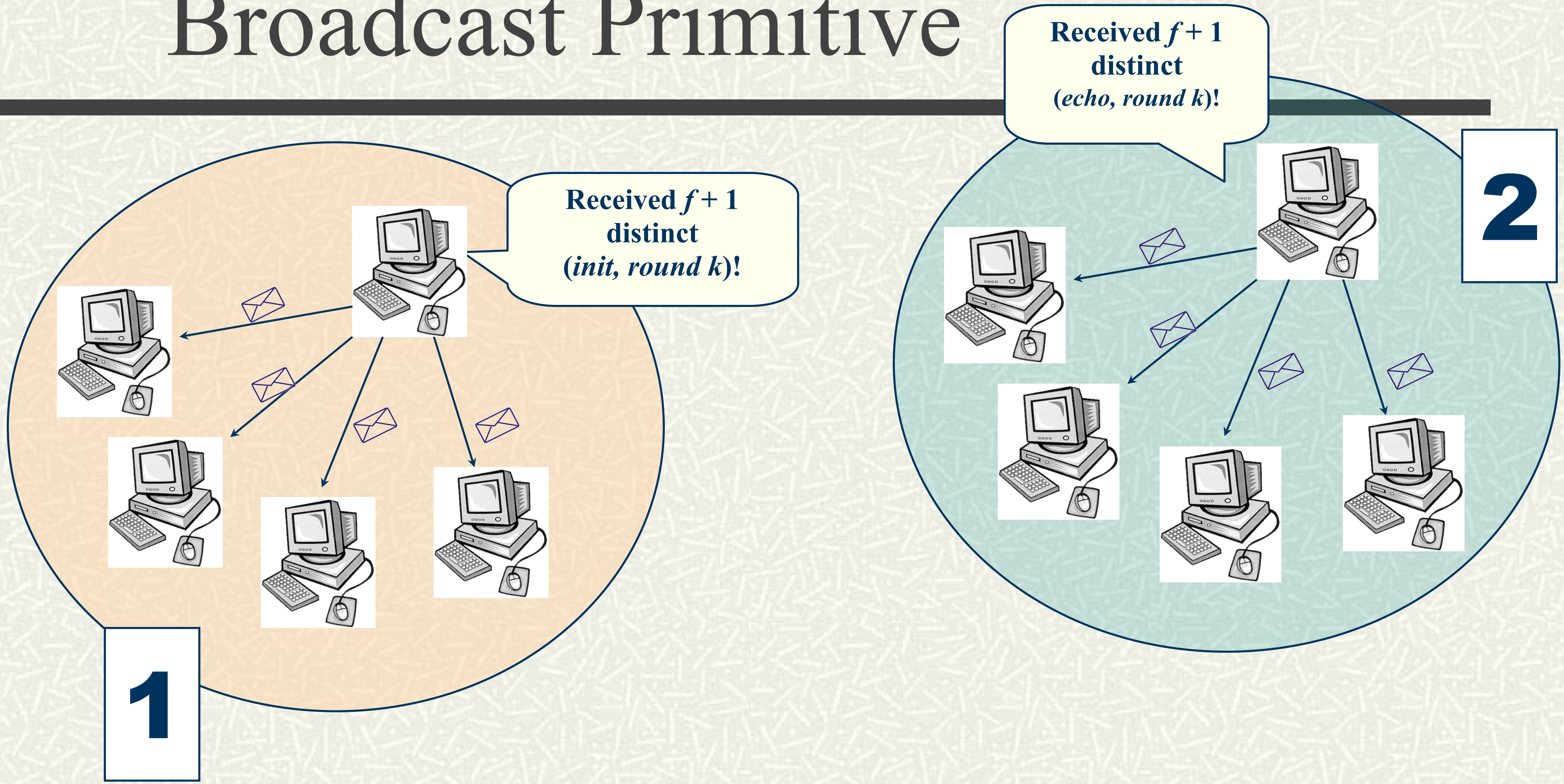
Broadcast Primitive

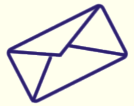
 → (*echo*, *round k*)

Broadcast Primitive

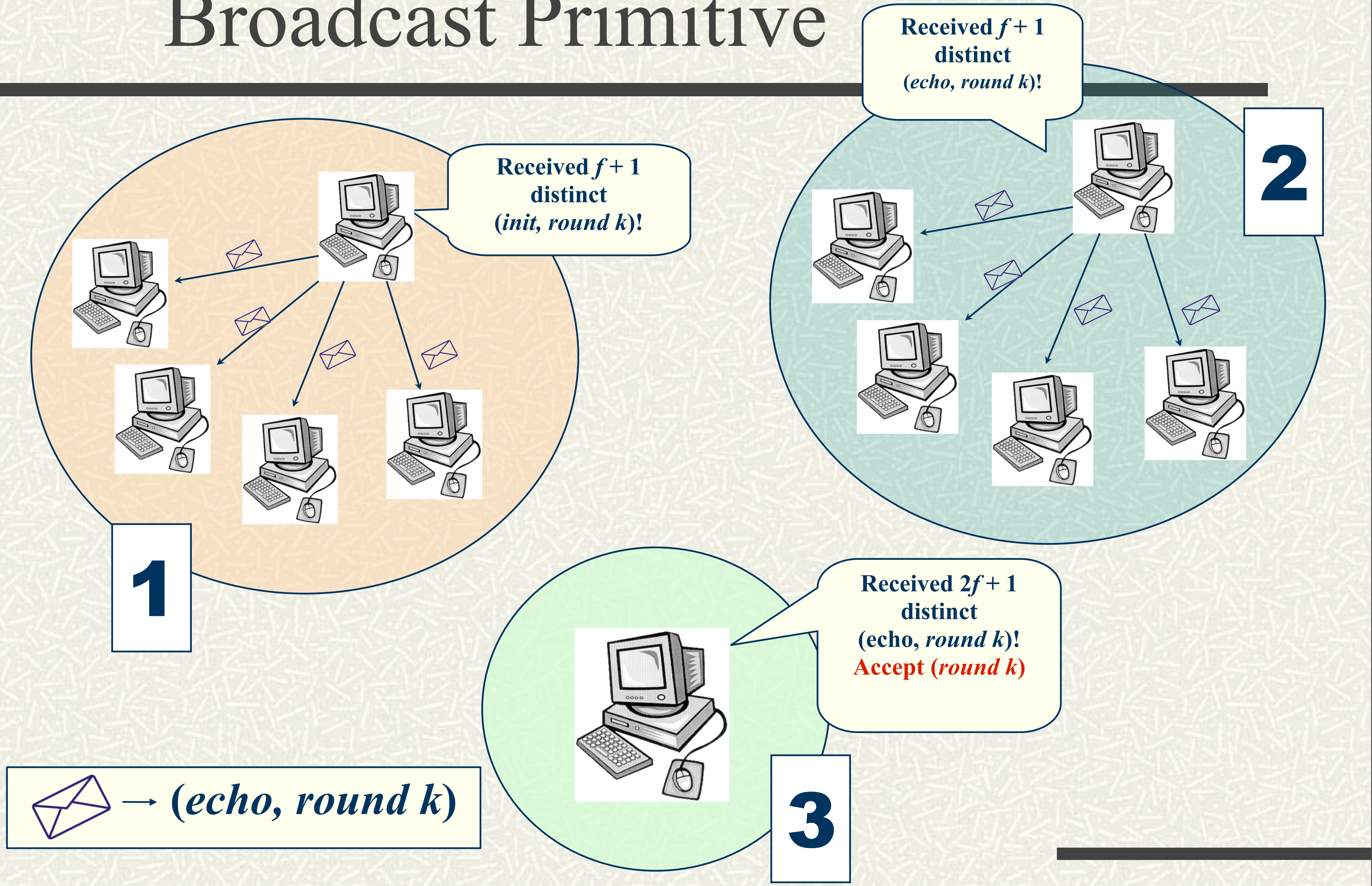


Broadcast Primitive



 → (*echo, round k*)

Broadcast Primitive



Initialization and Integration

- Same algorithms can be used to achieve initial synchronization and integrate new processes into the network
  - A process independently starts clock C^o
  - On accepting a message at real time *t*, it sets $C^0(t) = \alpha$
- "Passive" scheme for integration of new processes

Paper 2: Why try another approach?

- Traditional deterministic fault-tolerant clock synchronization algorithms:
  - Assume bounded communication delays
  - Require the transmission of at least N^2 messages each time N clocks are synchronized
  - Bursty exchange of messages within a narrow re-synchronization real-time interval

Probabilistic ICS

Claims:

- Proposes family of fault-tolerant internal clock synchronization (ICS) protocols
- Probabilistic reading achieves higher precisions than deterministic reading
- Doesn’t assume unbounded communication delays
- Use of convergence function →optimal accuracy

Their approach

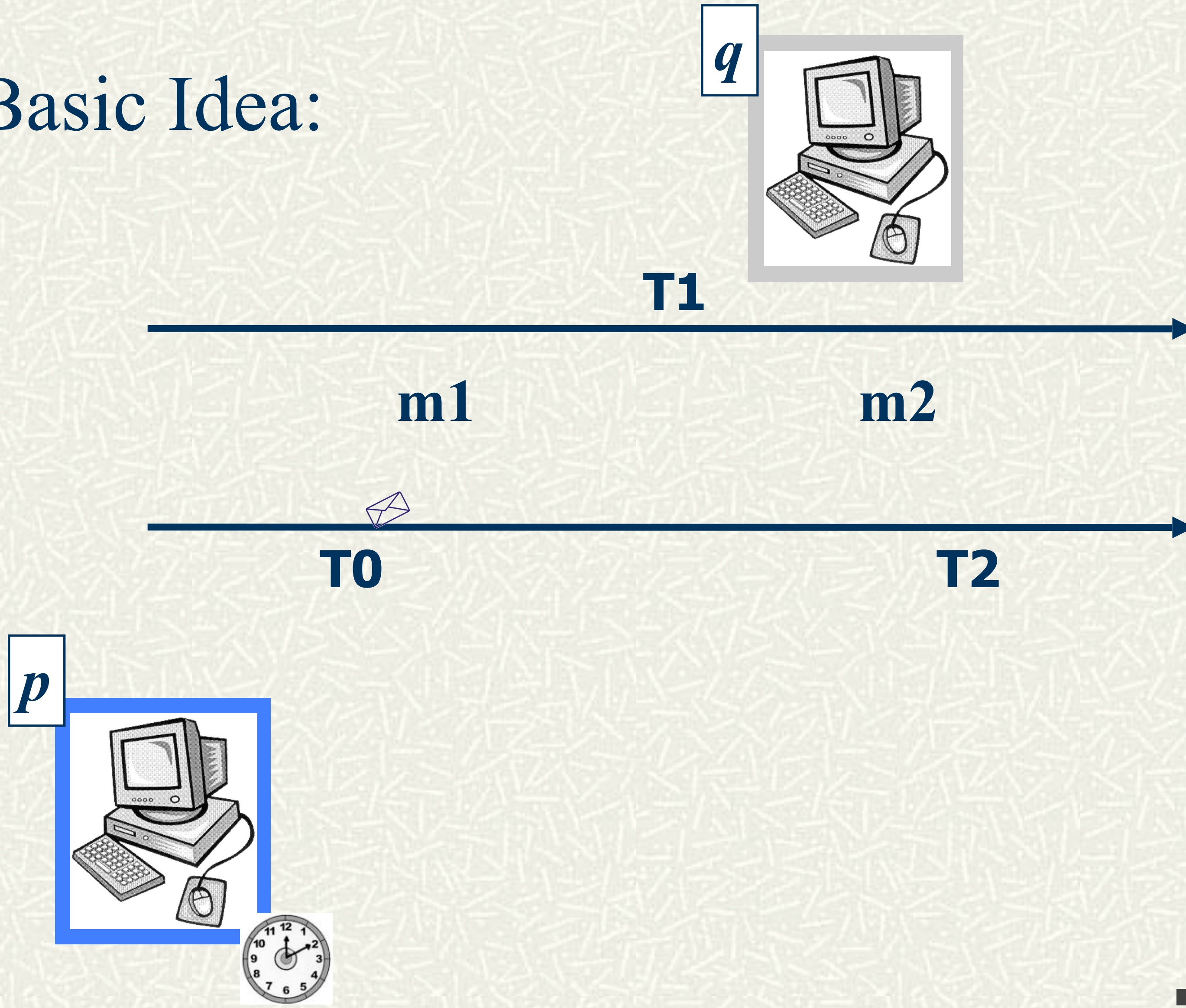
- Only requires to send a number of unreliable broadcast messages
- Staggers the message traffic in time
- Uses a new transitive remote clock reading method

Number of messages in the best case: $N + 1$

(N time server processes)

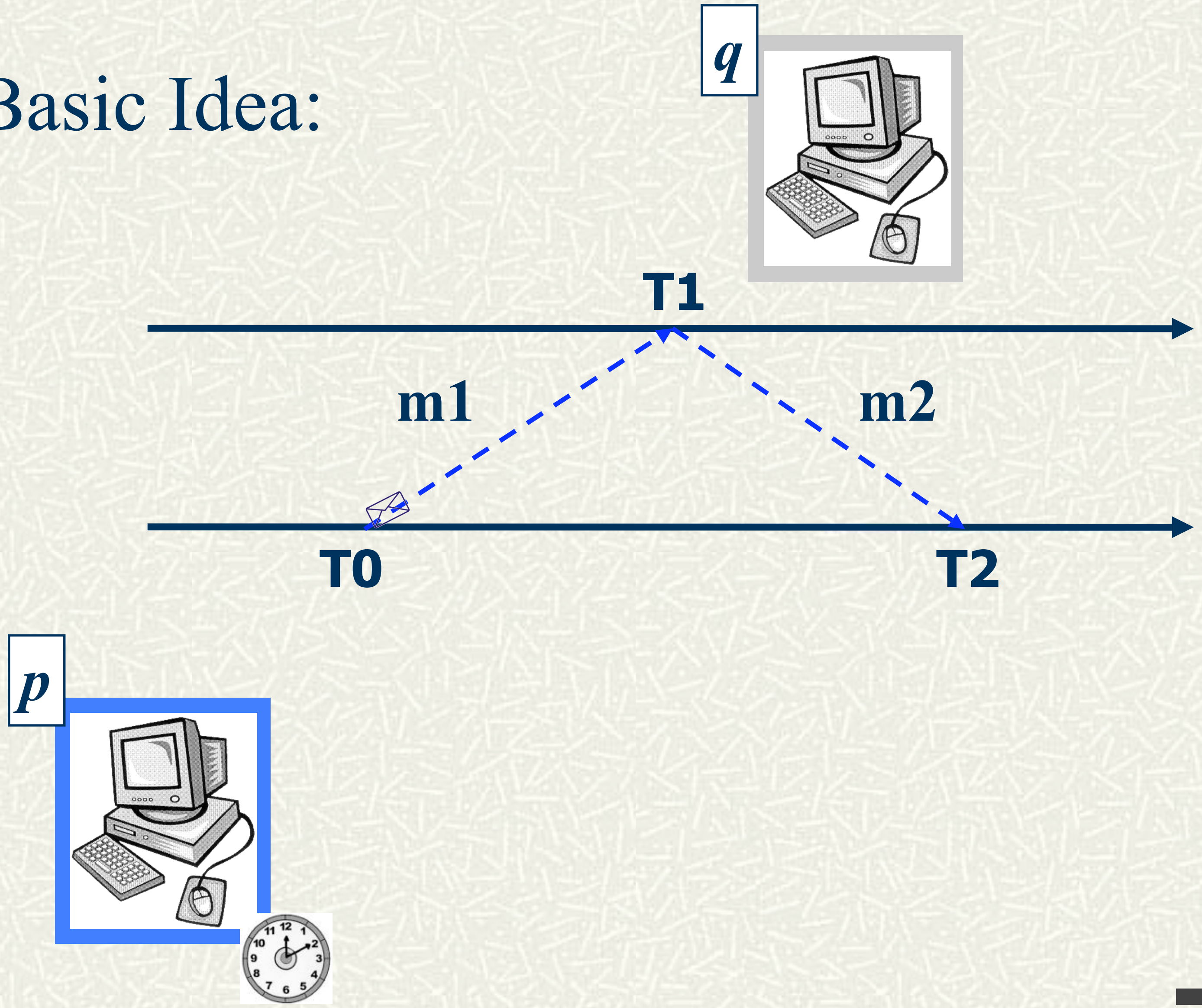
Probabilistic Clock Reading

- Basic Idea:



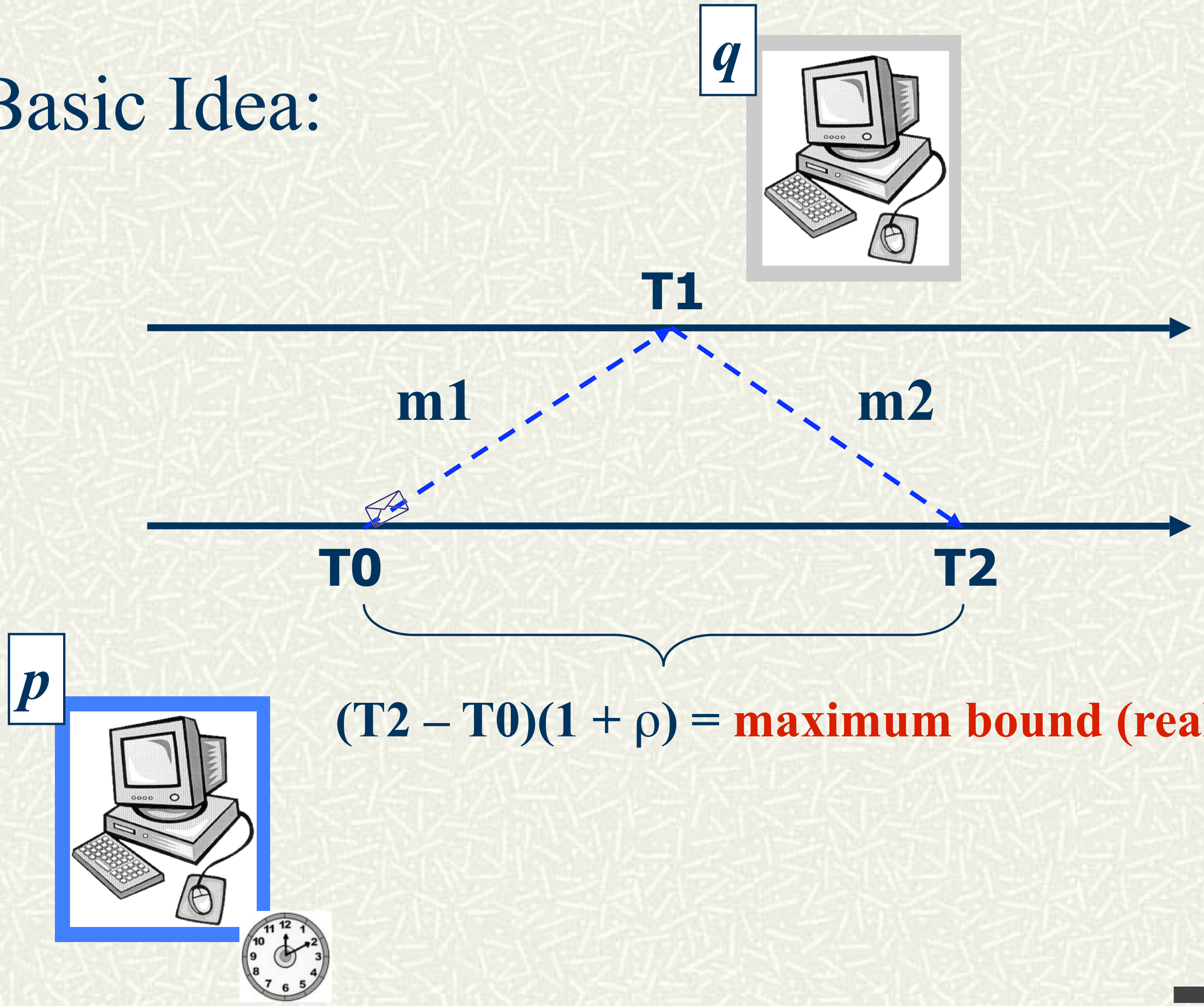
Probabilistic Clock Reading

- Basic Idea:



Probabilistic Clock Reading

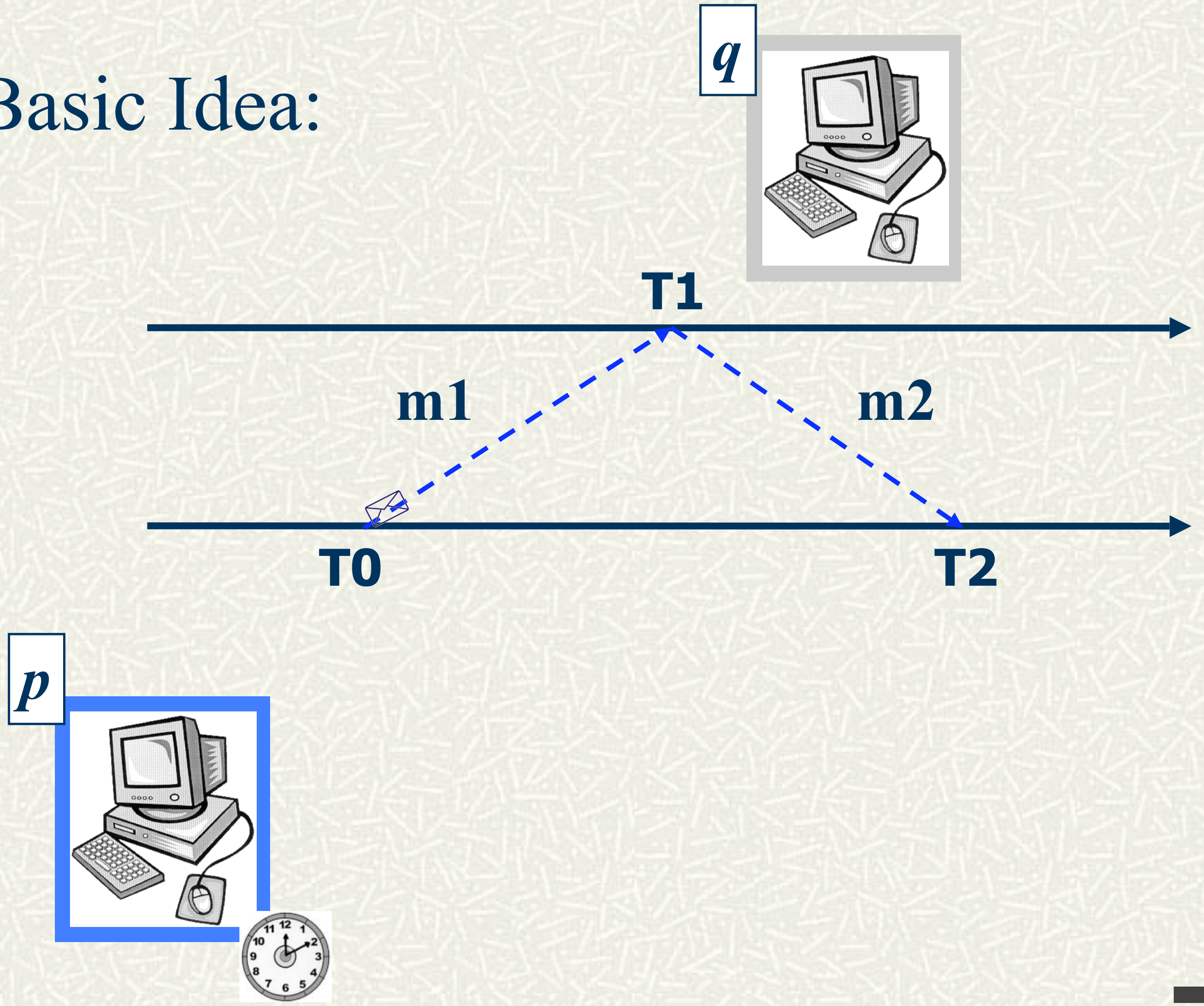
- Basic Idea:



$(T2 - T0)(1 + \rho)$ = **maximum bound (real time)**

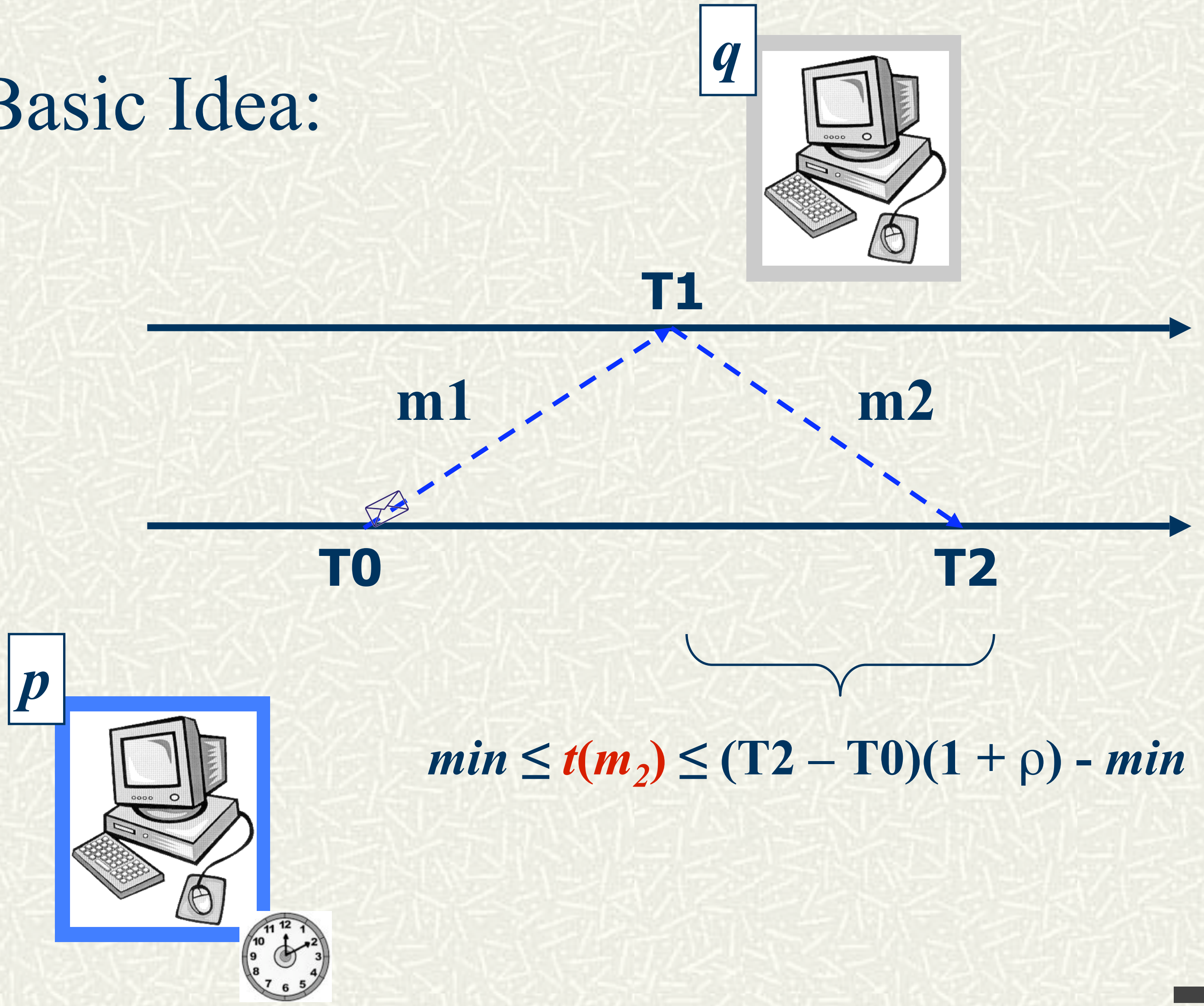
Probabilistic Clock Reading

- Basic Idea:



Probabilistic Clock Reading

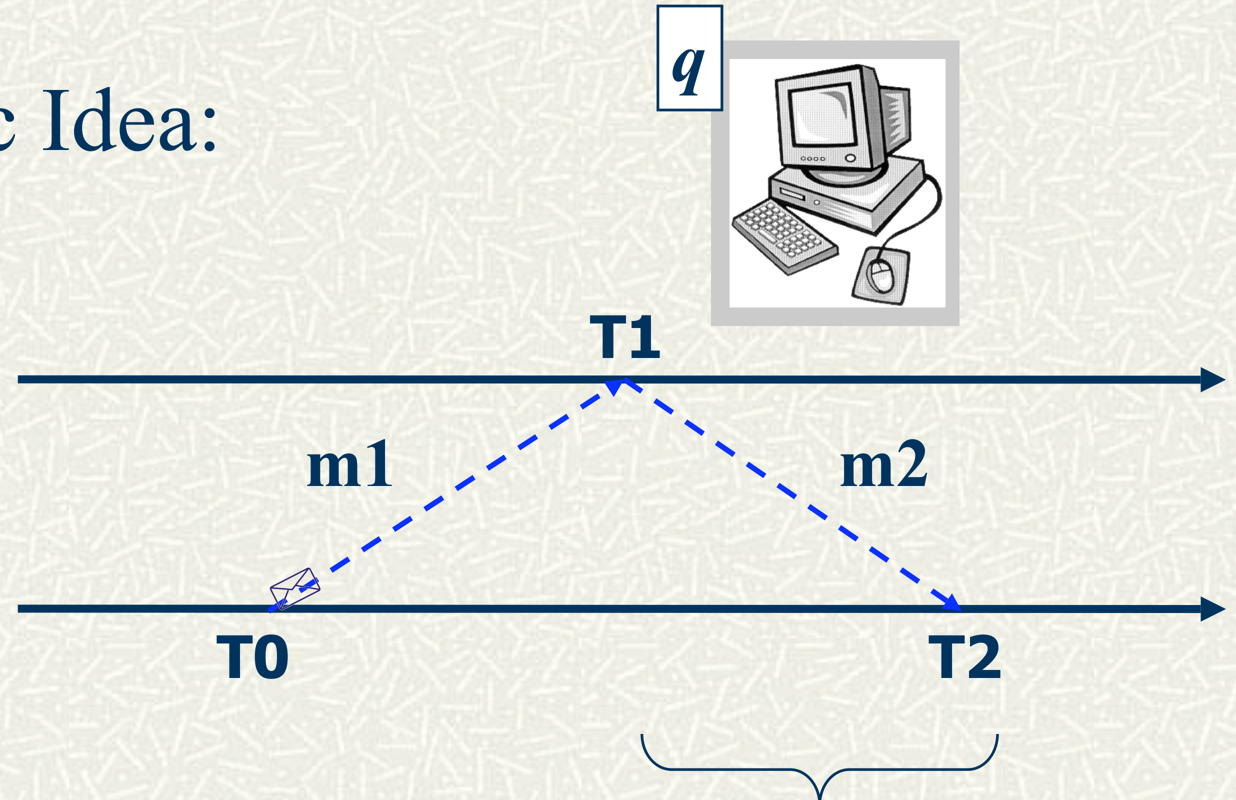
- Basic Idea:



$$min \leq t(m_2) \leq (T2 - T0)(1 + \rho) - min$$

Probabilistic Clock Reading

- Basic Idea:

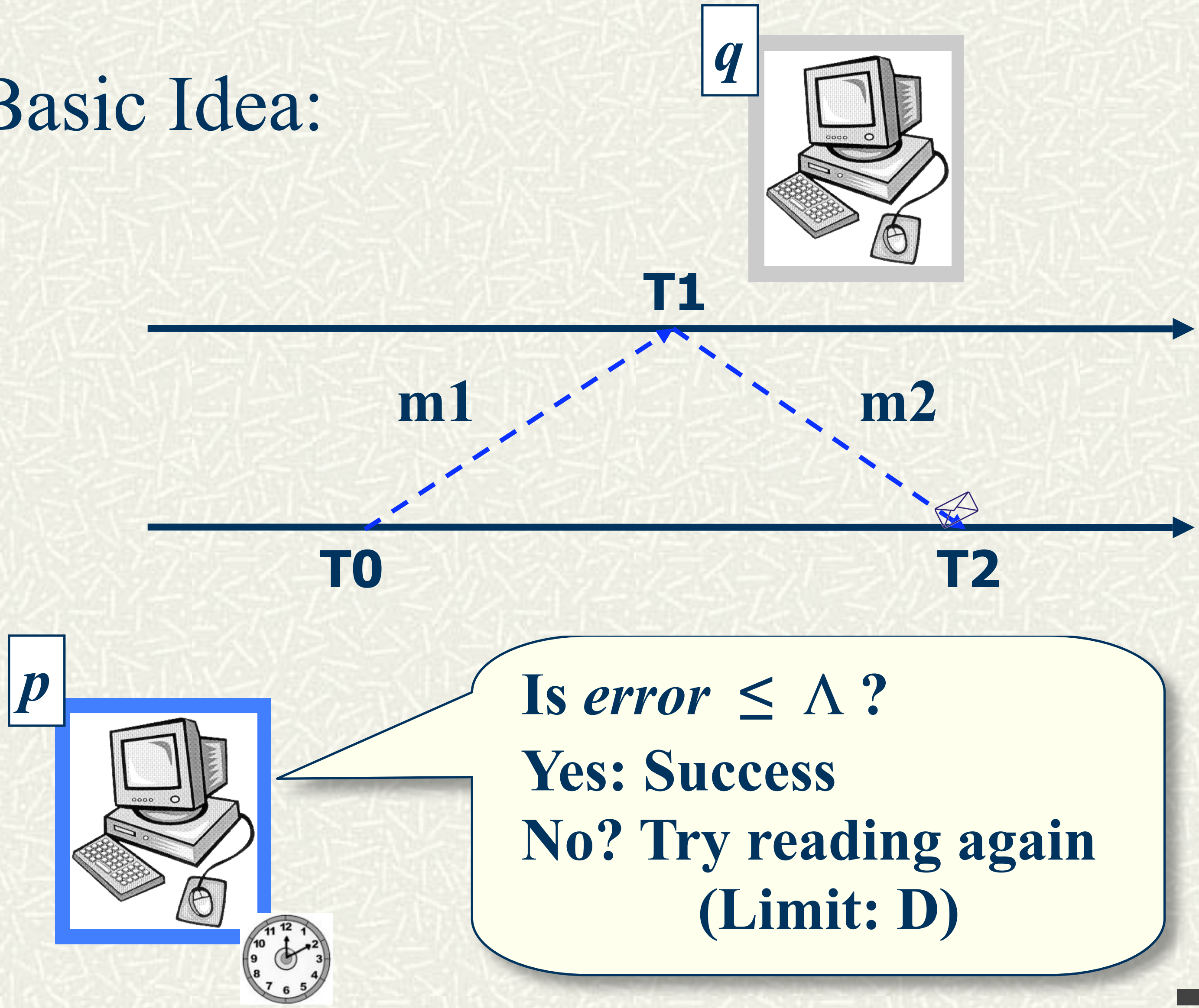


$$\mathit{min} \leq t(m_2) \leq (T2 - T0)(1 + \rho) - \mathit{min}$$

$$C_q = T1 + \frac{\mathit{max}(m_2)(1 + \rho) + \mathit{min}(m_2)(1 - \rho)}{2}$$

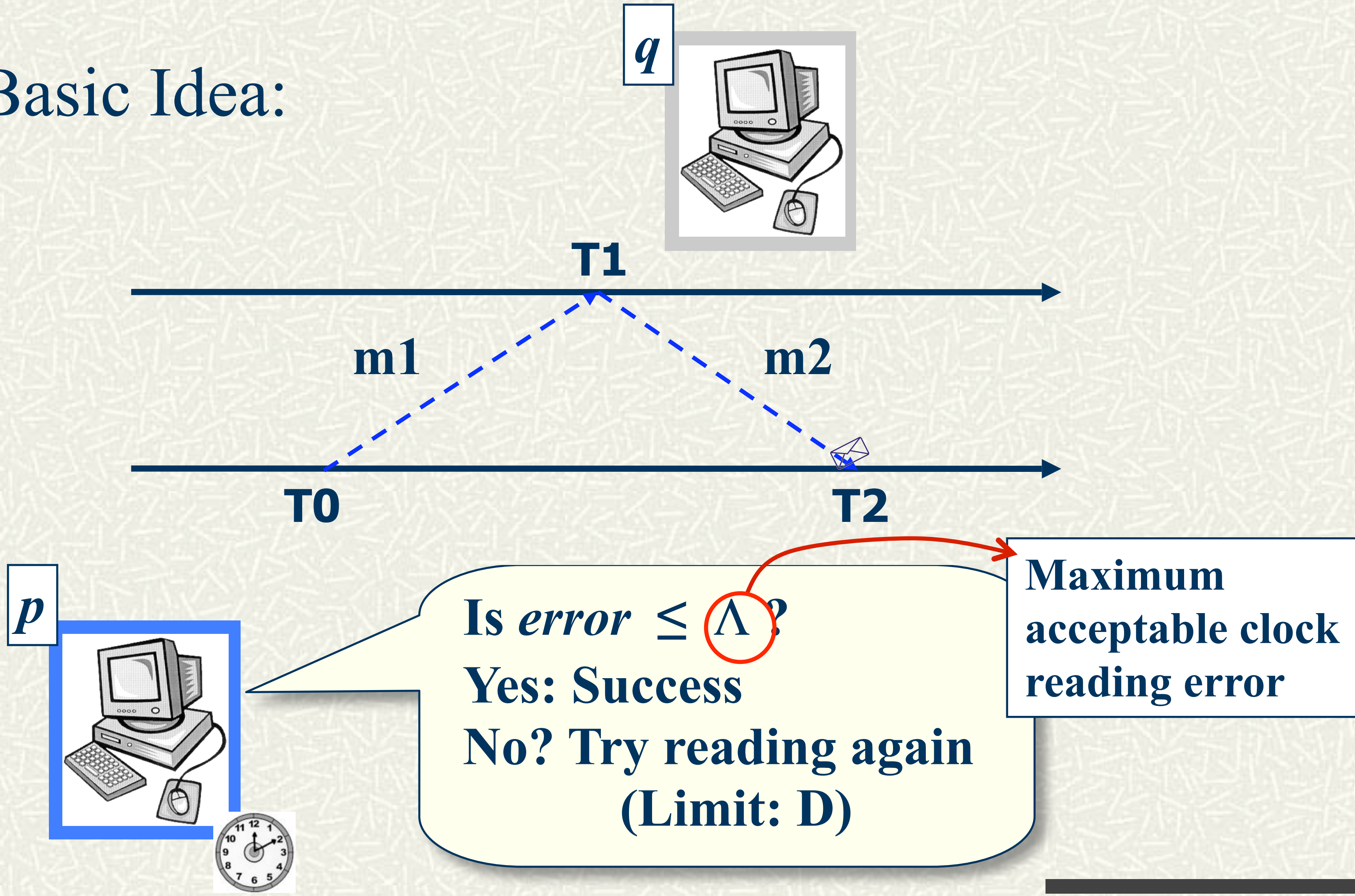
Probabilistic Clock Reading

- Basic Idea:

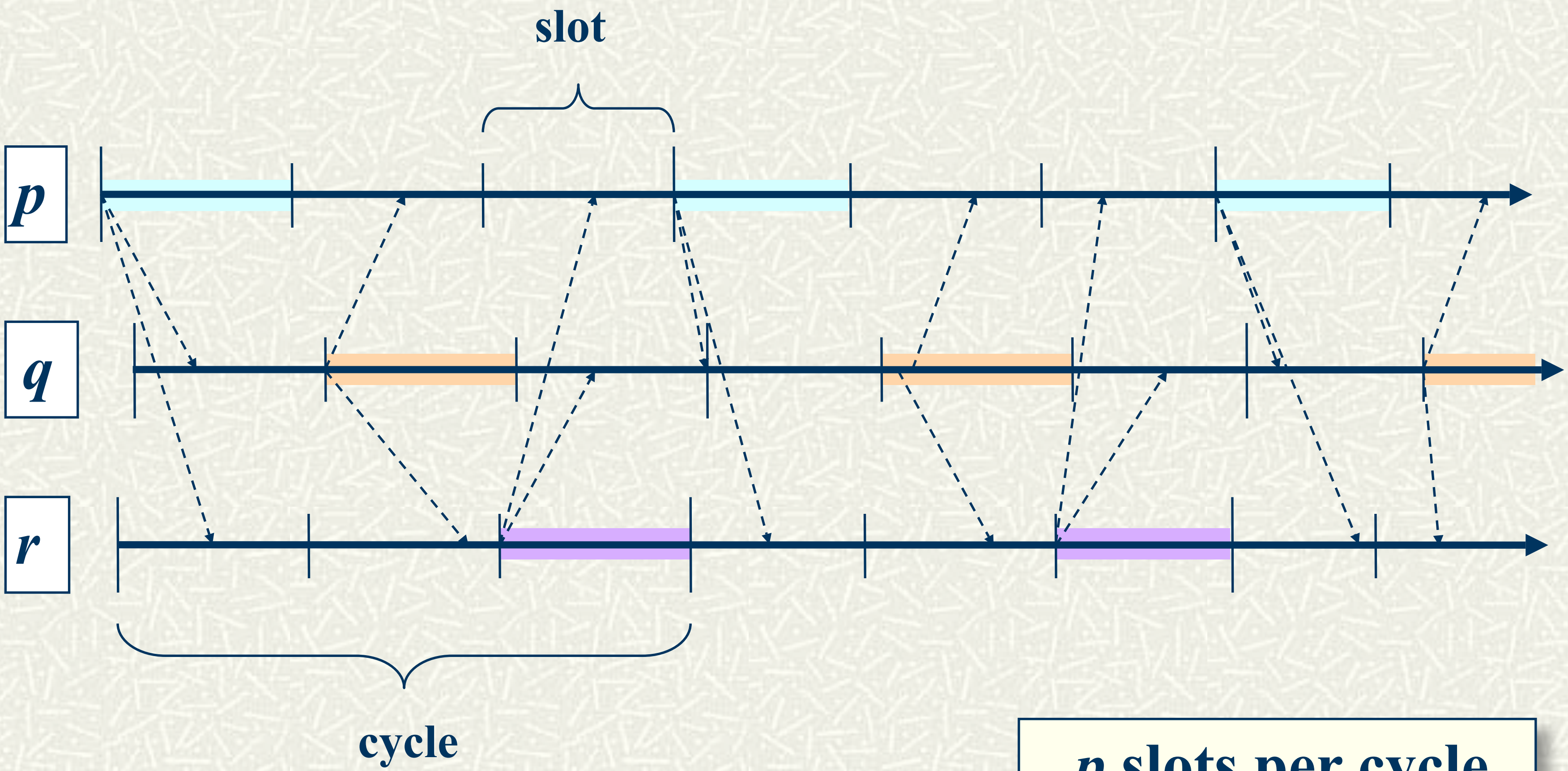


Probabilistic Clock Reading

- Basic Idea:



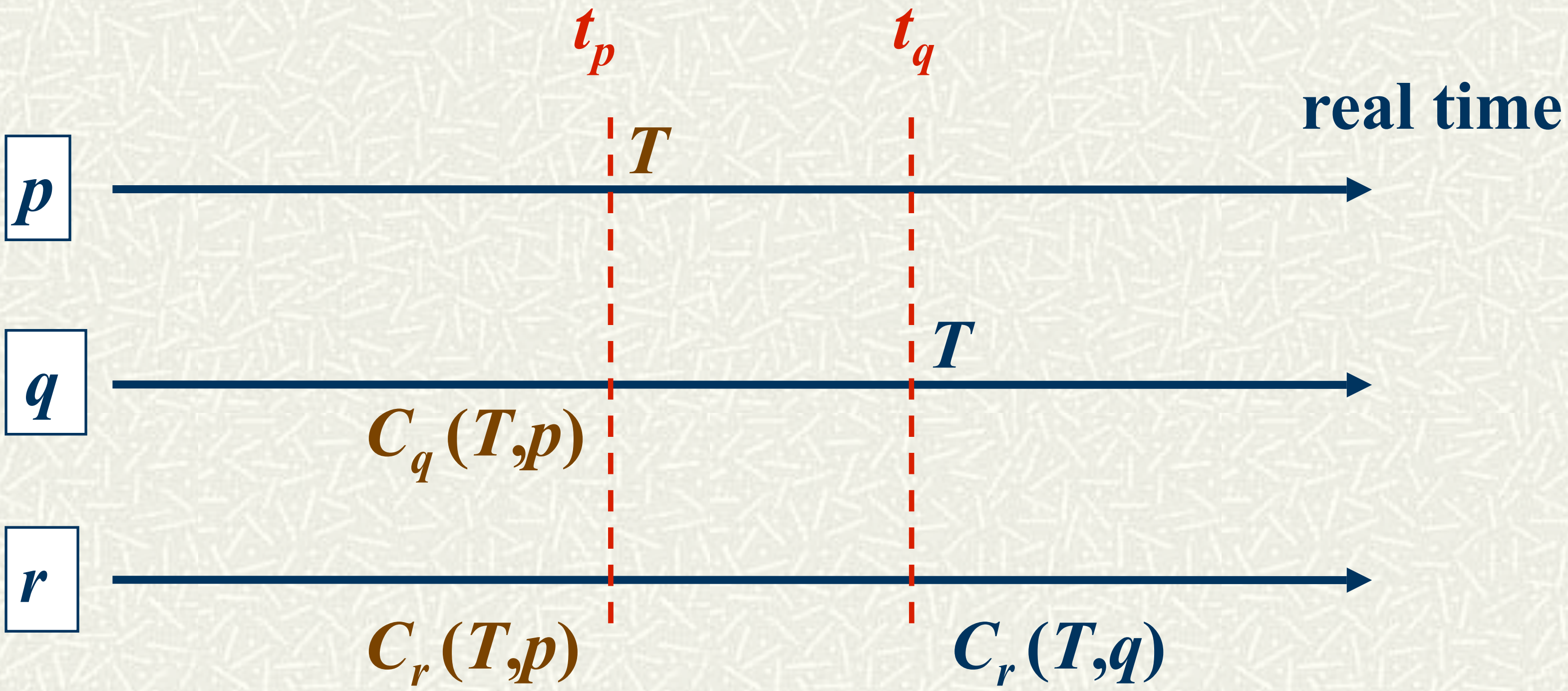
Staggering Messages



***p* slots per cycle**
***k* cycles per round**

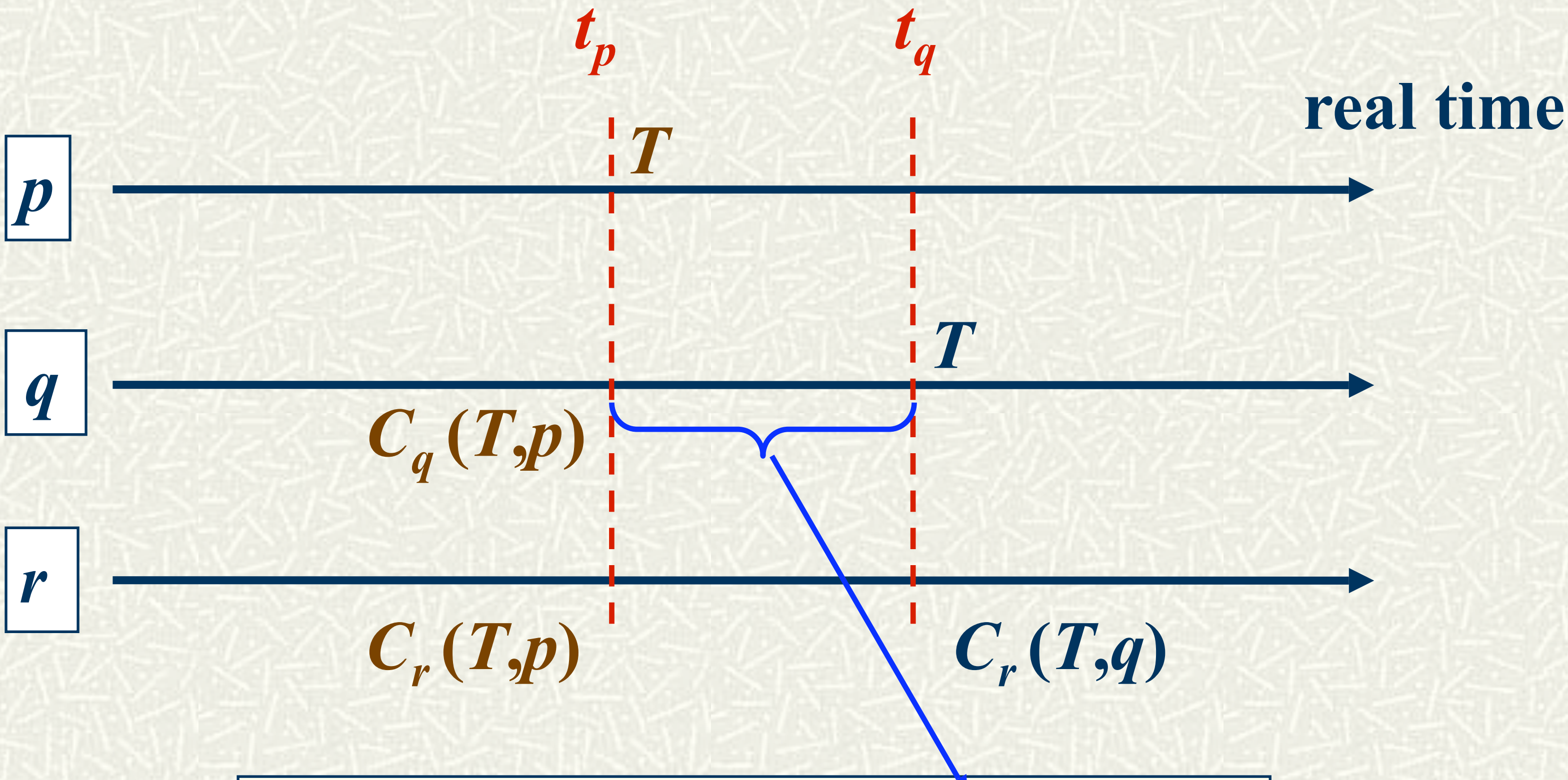
Transitive Remote Clock Reading

- Can reduce the number of messages per round to $N + 1$



Transitive Remote Clock Reading

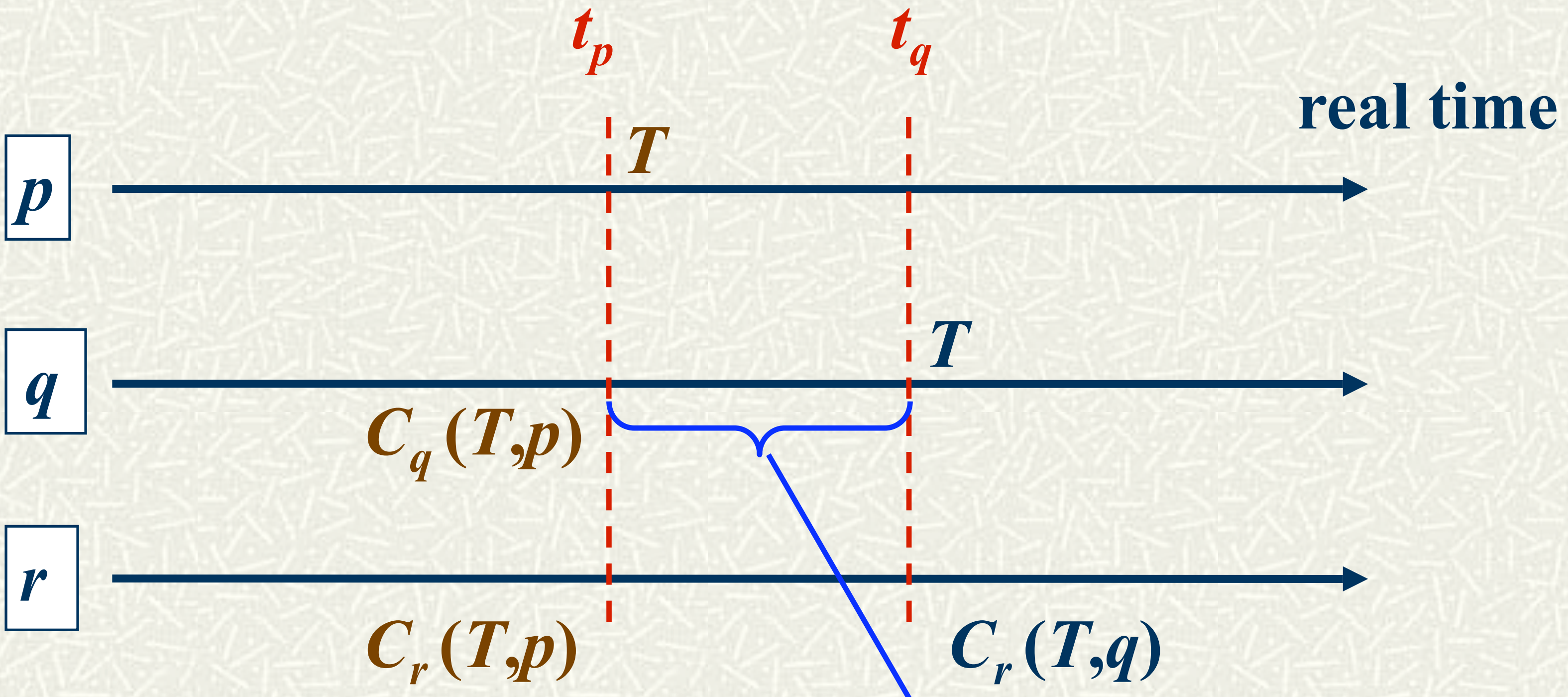
- Can reduce the number of messages per round to $N + 1$



$$C_r\,(T,q) = C_r\,(T,p) + T - C_q\,(T,p)$$

Transitive Remote Clock Reading

- Can reduce the number of messages per round to $N + 1$

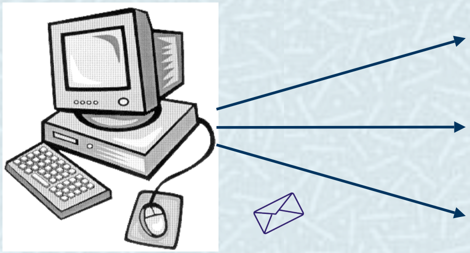


Cannot be used when arbitrary failures can occur!

Round Message Exchange Protocol

Round Message Exchange Protocol

Request Mode



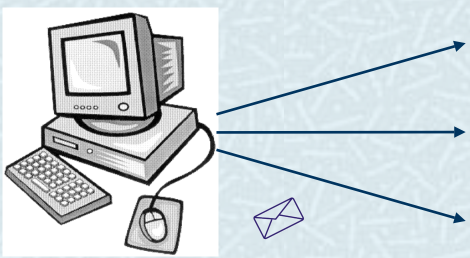
Clock times:

	p	*q*	*r*
t	?	?	?
err	?	?	?

request messages

Round Message Exchange Protocol

Request Mode

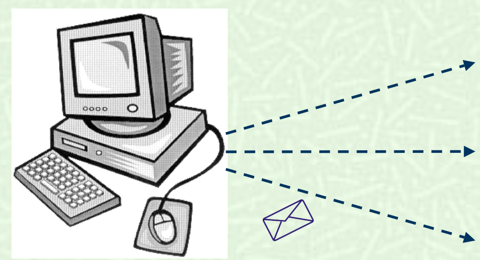


Clock times:

	p	*q*	*r*
t	*?*	*?*	*?*
err	*?*	*?*	*?*

request messages

Reply Mode

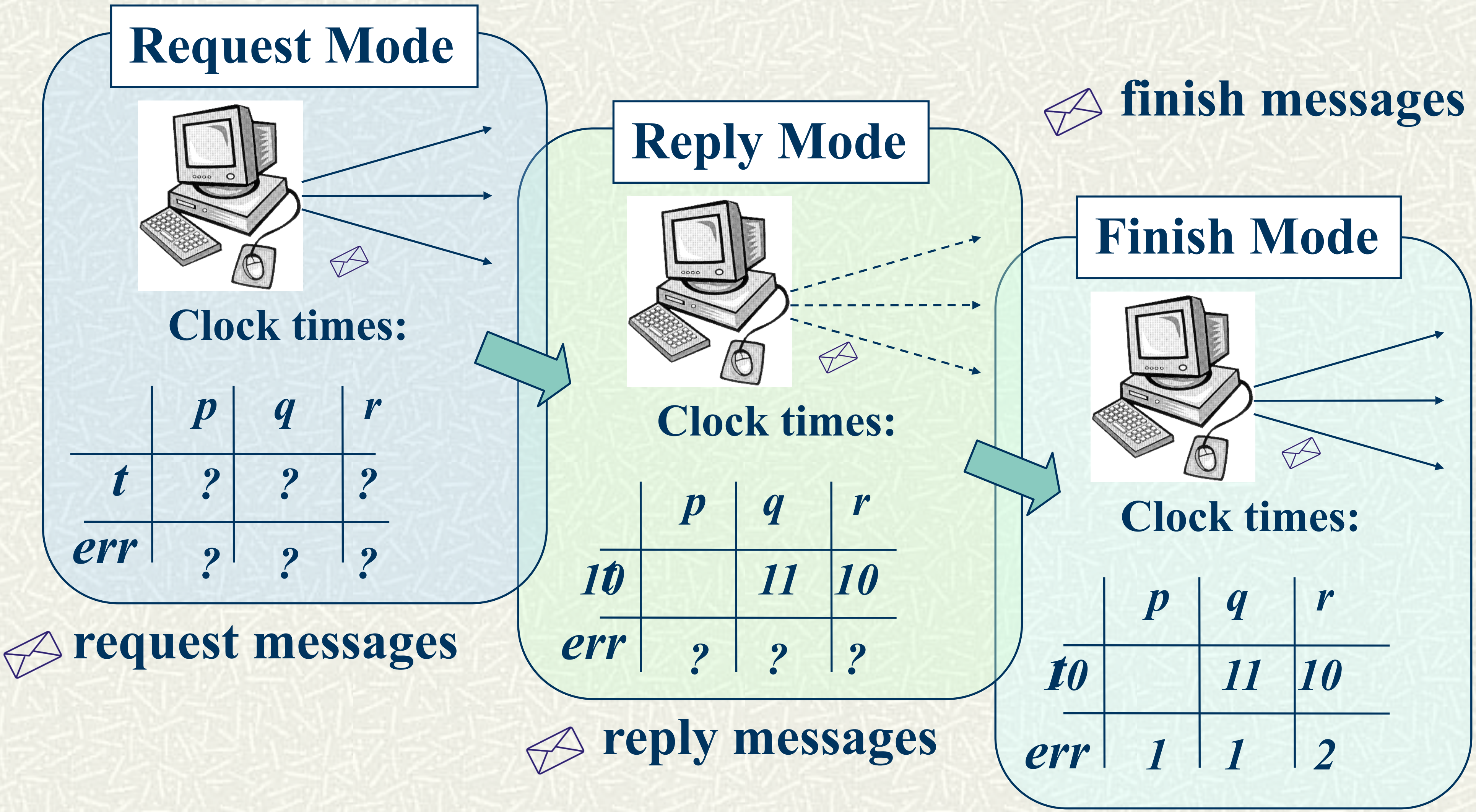


Clock times:

	p	*q*	*r*
10		*11*	*10*
err	*?*	*?*	*?*

reply messages

Round Message Exchange Protocol



Outline of Algorithms

Round clock C_p^k of process *p* for round *k:*

$$C_p^k(t) = H_p(t) + A_p^k$$

```
Void synchronizer() {
   ReadClocks(..)
   A = A + cfn(rank(), Clocks, Errors)
   T = T + P
}
```

Convergence Functions

- Let I(t) = [L, R] be the interval spanned by at t by correct clocks. If all processes would set their virtual clocks at the same time t to the midpoint of I(t), then all correct clocks would be exactly synchronized at that point in time.

Unfortunately, this is not a perfect world!

Convergence Functions

- Each correct process makes an approximation I_p which is guaranteed to be included in a bounded extension of the interval of correct clocks *I*:

$$I_\Lambda^k(t) = [min\{C_s^k\,(\mathrm{t}) - \Lambda\}, max\{C_s^k\,(\mathrm{t}) + \Lambda\}]$$

Deviation of clocks is bounded by δ, so length of $I_\Lambda^k(t)$ is bounded by $\delta + \mathbf{2}\Lambda$

Failure classes

Algorithm	Tolerated Failures	Required Processes	Tolerated types of failures
CSA Crash	F	F + 1	Crash
CSA Read	F	2F + 1	Crash, Reading
CSA Arbitrary	F	3F + 1	Arbitrary, Reading
CSA Hybrid	Fc, Fr, Fa	3Fa + 2Fr + Fc + 1	Crash, Read., Arb.

Conclusions – Which one is better?

- First Paper (deterministic algorithm)
  - Simple algorithm
  - Unified solution for different types of failures
  - Achieves *optimal accuracy*
  - Assumes bounded comunication
  - $O(n^2)$ messages
  - Bursty communication

Conclusions – Which one is better?

- Second Paper (probabilistic algorithm)
  - Takes advantage of the current working conditions, by invoking successive round-trip exchanges, to reach a tight precision)
  - Precision is not guaranteed
  - Achieves *optimal accuracy*
  - $O(n)$ messages

Conclusions – Which one is better?

- Second Paper (probabilistic algorithm)
  - Takes advantage of the current working conditions, by invoking successive round-trip exchanges, to reach a tight precision)
  - Precision is not guaranteed
  - Achieves *optimal accuracy*
  - $O(n)$ messages

If both algorithms achieve optimal accuracy,

Then why is there still work being done?