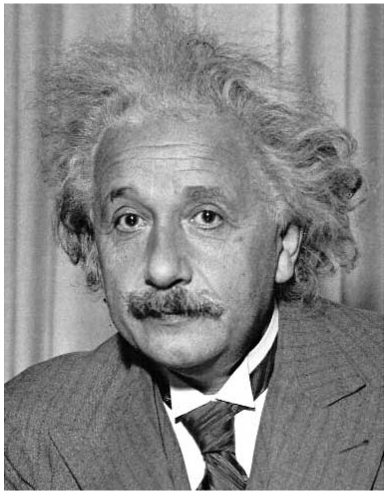
Time

Supriya Vadlamani

Asynchrony v/s Synchrony

- Last class:
  - Asynchrony
    - Event based
    - Lamport's Logical clocks
- Today:
  - Synchrony
    - Use real world clocks
    - But do all the clocks show the same time?

Problem Statement



"The only reason for time is so that everything doesn't happen at once." – Albert Einstein

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.

Why is The Problem Hard?

- Variation of transmission delays
  - Each process cannot have an instantaneous global view
- Presence of drift in clocks.
- Support faulty elements **Hardest !**

Applications that Require Synchronization

- Transaction processing applications
- Process control applications
- Communication protocols
  - require approximately the same view of time

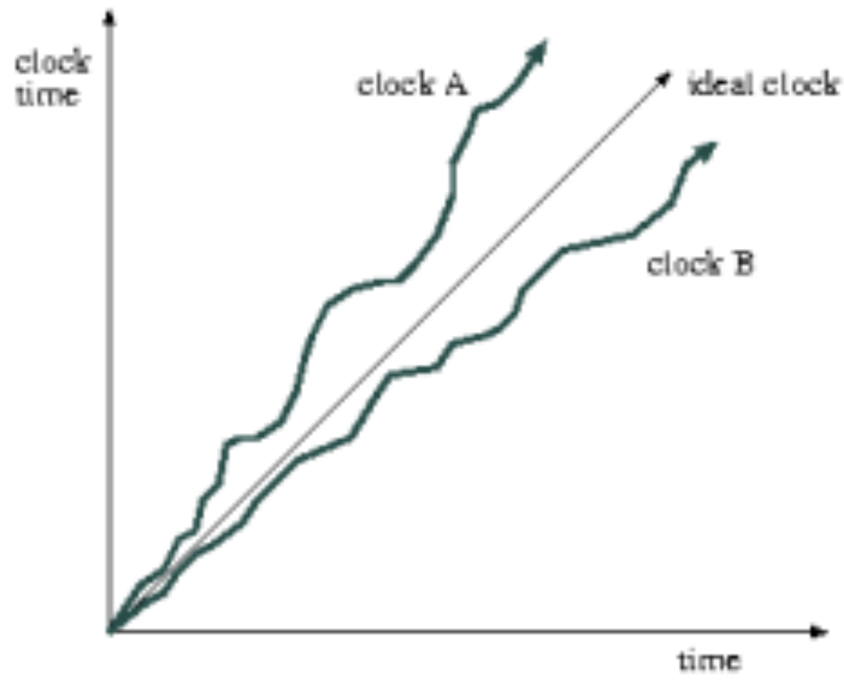
Approaches to Clock Synchronization

- Hardware vs Software clocks
- External vs Internal Clock synchronization

Hardware Clocks

- Each processor has an oscillator
- BUT- oscillators drift!

Logical Clock =
H/w clock + Adjustment Factor

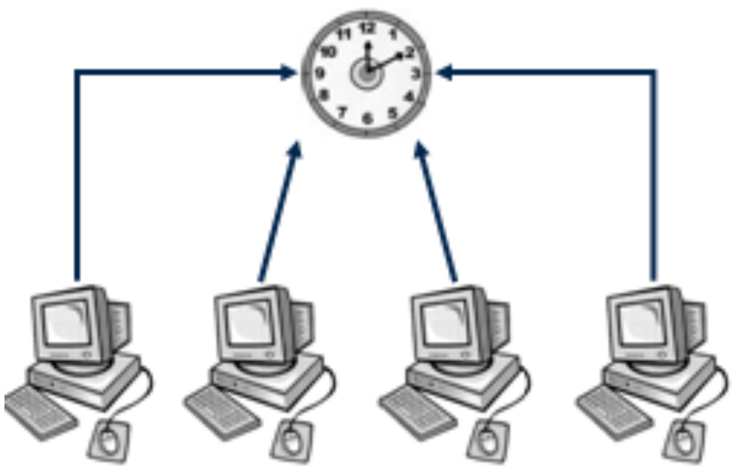


Software Clocks

1. Deterministic → assumes an upper bound on transmission delays – guarantees some precision
   Realistic ?
2. Statistical → expectation and standard deviation of the delay distributions are known
   Reliable ?
3. Probabilistic → no assumptions about delay distributions
   Any Guarantees ?

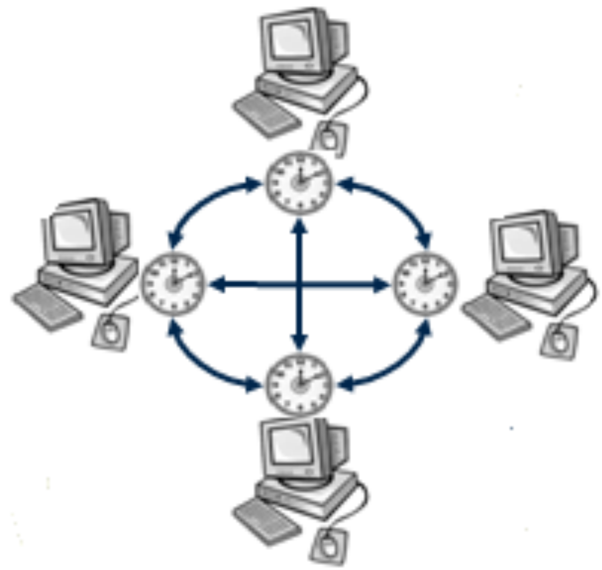
Clock Synchronization

External Clock Synchronization



Synchronize clocks with respect to an external time reference
Example: NTP

Internal Clock Synchronization



Synchronize clocks among themselves

Today…

Optimal Clock Synchronization [Srikanth and Toueg '87]

- Assume reliable network (deterministic)
- Internal clock synchronization
- Also optimal with respect to failures

Authors:

Sam Toueg – Cornell University
-Moved to MIT
T K Srikanth- Cornell University

Types of Failures in a Network

- Up to f processes can fail in the following ways:
  - *Crash Failure:*
    - processor behaves correctly and then stops executing forever
  - *Performance Failure:*
    - processor reacts too slowly to a trigger event(Eg:Clock too slow or fast, Stuck clock bits)
  - *Arbitrary Failure (a.k.a Byzantine):*
    - processor executes uncontrolled computation

System Model

Assumptions:

- Clock drift is bounded

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

- Communication and processing are reliable

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

- Authenticated messages

  will relax this later...

Goals

- Property 1 Agreement:

  Bounded drift btw processes

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

  (δ is the precision of the clock synchronization algorithm)

- Property 2 Accuracy:

  Bounded drift within a process

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

Goals

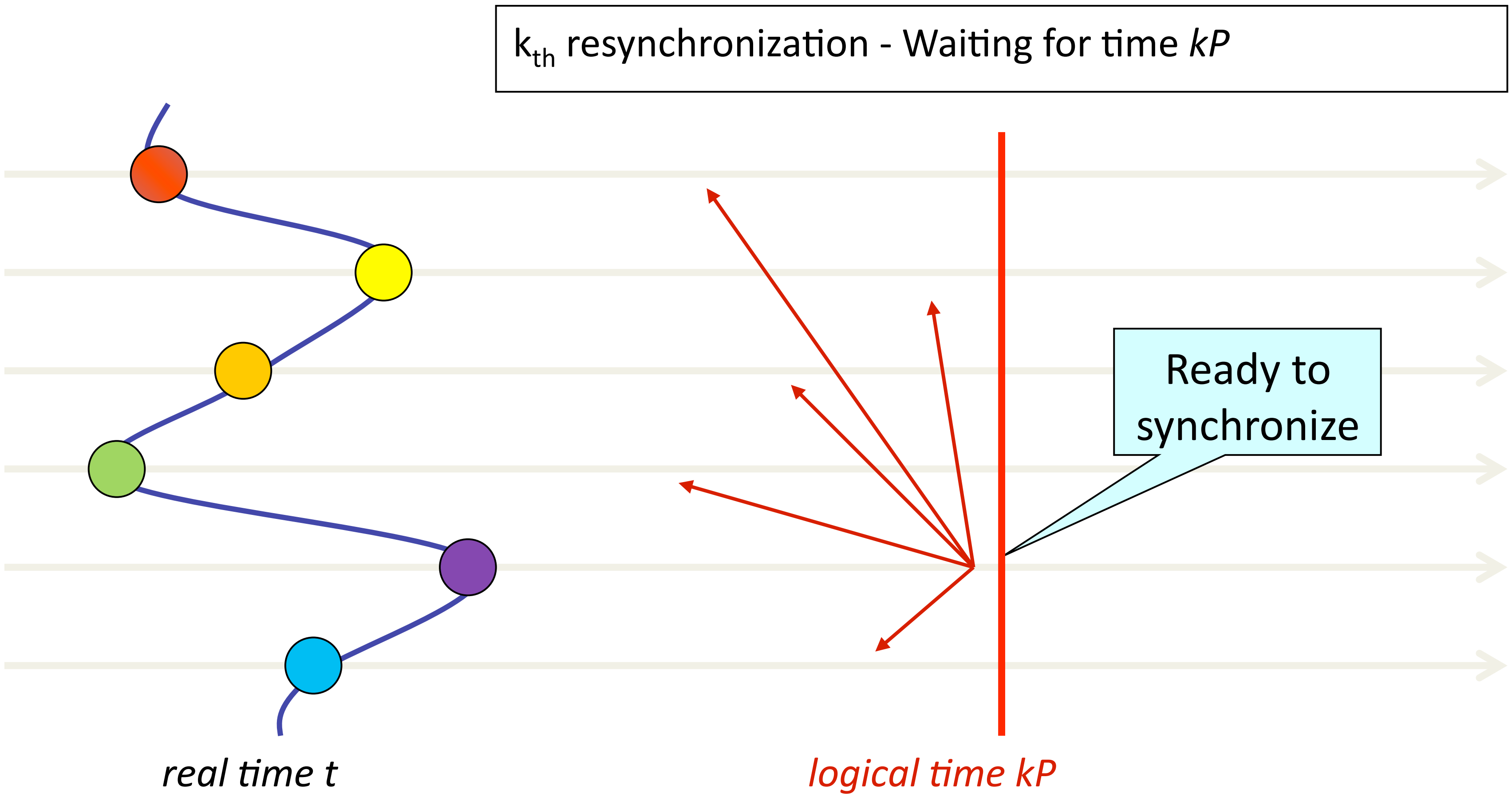
- Optimal Accuracy
  - Drift rate bounded by the maximum drift rate of correct hardware clocks

$$\rho_V = \rho$$

Outline

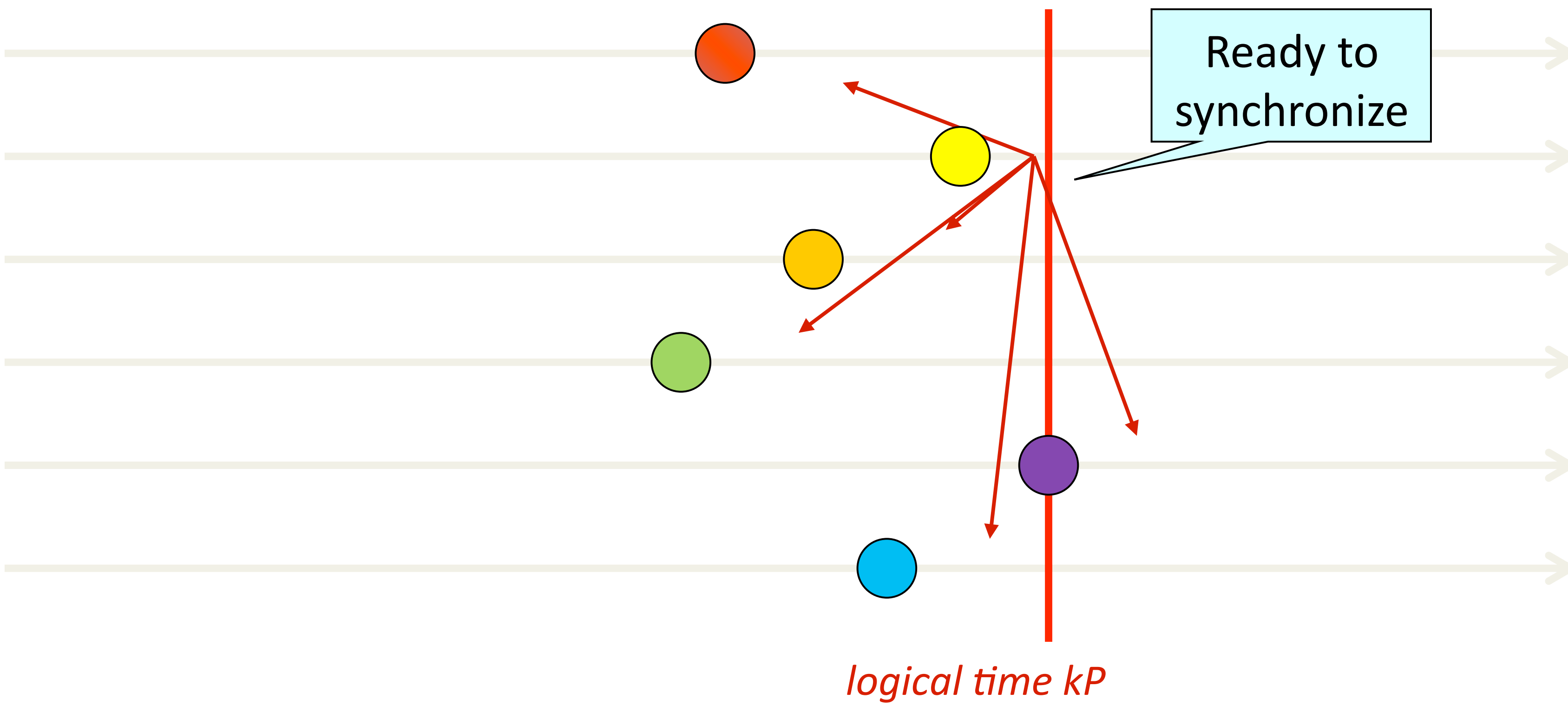
- Synchronization algorithm – authentication
- Optimizing for accuracy
- Properties
- Synchronization algorithm – broadcast
- Initialization and Integration

Synchronization Algorithm (Authenticated Messages)



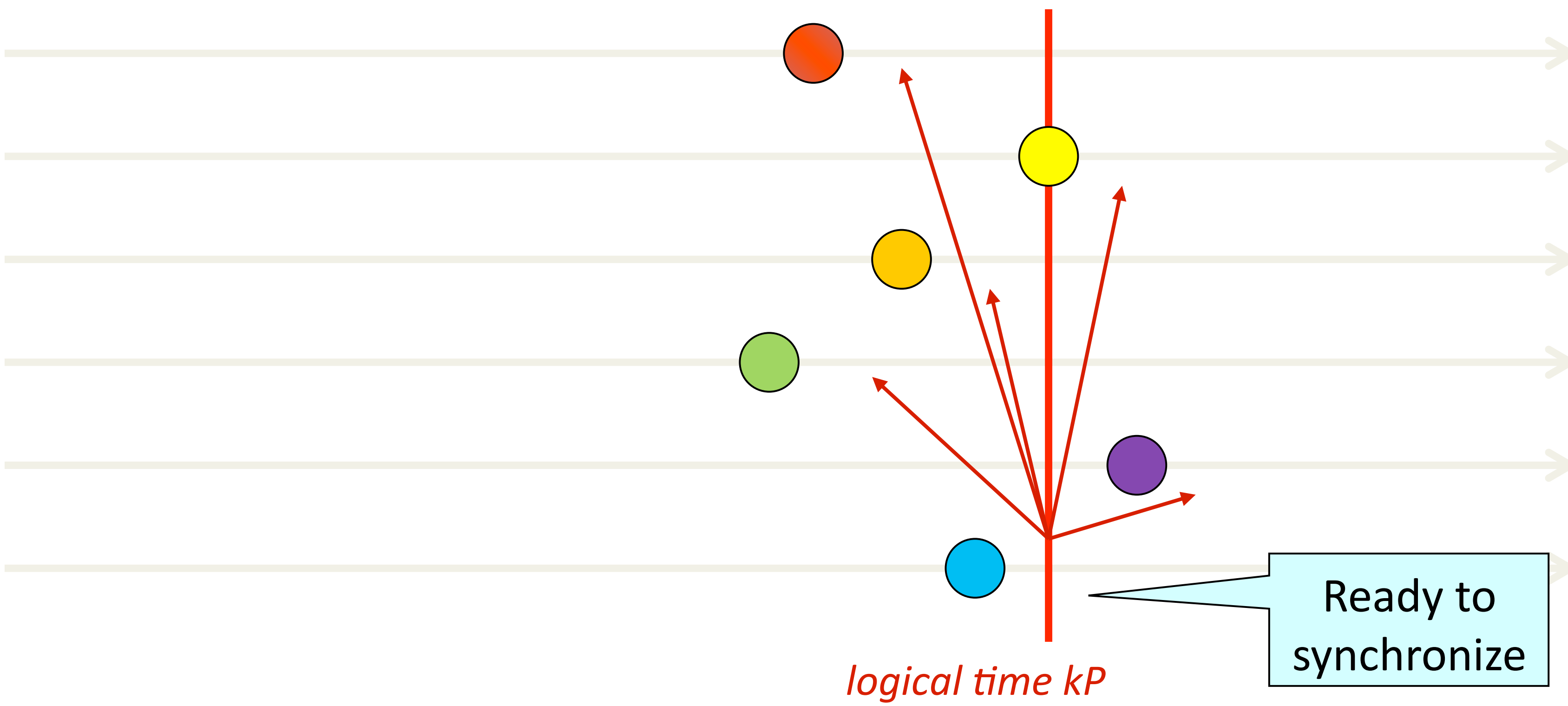
P – logical time between resynchronizations

Synchronization Algorithm (Authenticated Messages)



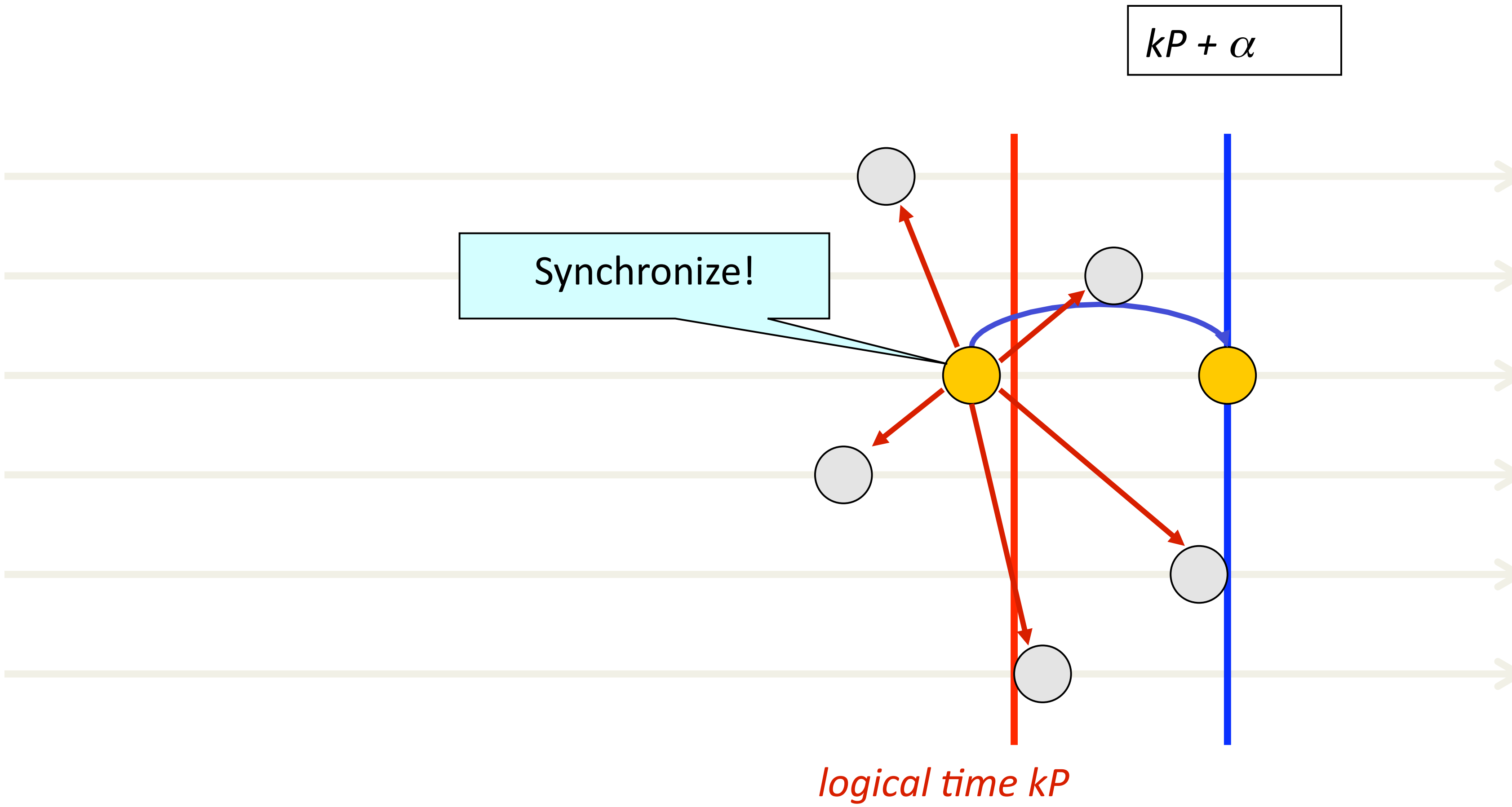
P – logical time between resynchronizations

Synchronization Algorithm (Authenticated Messages)



P – logical time between resynchronizations

Synchronization Algorithm (Authenticated Messages)



P – logical time between resynchronizations

Achieving Optimal Accuracy

Uncertainty of t_{delay}

➔difference in the logical time between resynchronizations

➔Reason for non-optimal accuracy

Solution:

Slow down or speed up the logical clocks.

Slow down to $kP+\alpha$ when C_k^{i-1} reads $\min(T+\beta, kP+\beta)$

Speed up to $kP+\alpha$ when C_k^{i-1} reads $\max(T-\beta, kP+\beta)$

Properties Essential to the Algorithm

Unforgeability:

No process broadcasts → no correct process accepts by t

Relay:

Correct process accepts message at time t, → others do so by time t + tdel

Correctness:

Round k: f+1 broadcast messages → received by t+tdel

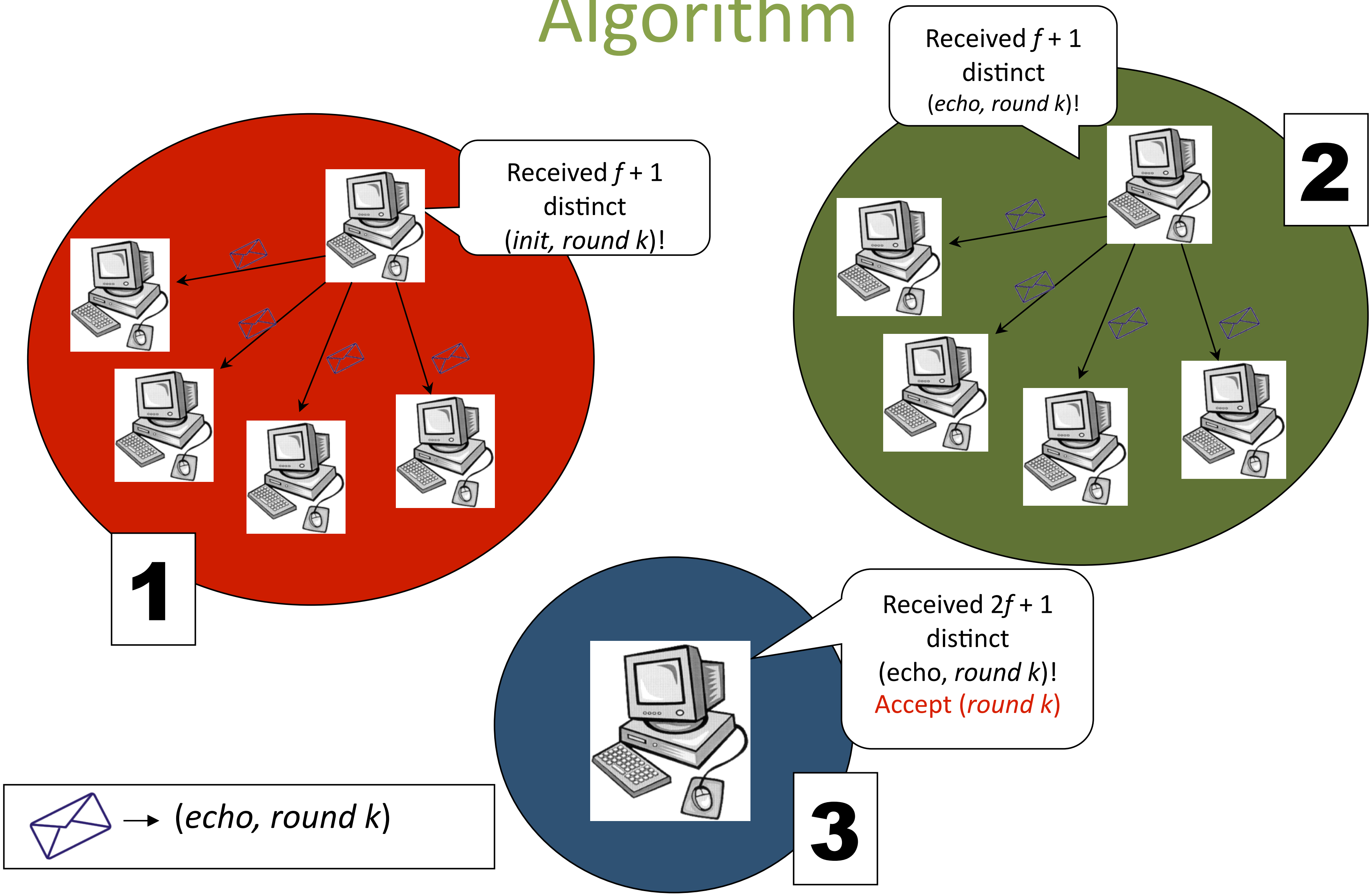
Broadcast Primitive

- Strong authentication is too heavyweight. Only need:
  - Unforgeability
  - Relay
  - Correctness
- Can use a broadcast primitive from the literature.

Synchronization Algorithm (Broadcast Primitives)

- 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 based Synchronization Algorithm



Take Away

- Deterministic algorithm
  - Simple algorithm
  - Unified solution for different types of failures
  - Achieves "*optimal*" *accuracy*
  - $O(n^2)$ messages
  - Bursty communication

Today..

- Probabilistic Internal Clock Synchronization [Cristian and Fetzer '03]
  - Drop requirements on network (probabilistic)
  - Internal Clock synchronization

- Authors

Flaviu Cristian

Christof Fetzer (TU Dresden)

UC San Diego

How is the second paper different ?

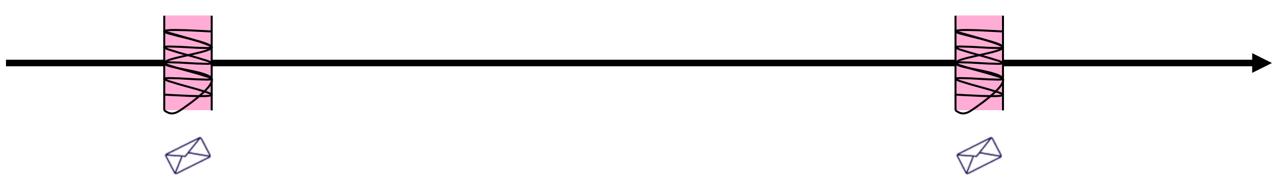
- Deterministic approach
  - ➔ Bound on transmission delays
- Require N^2 messages
- Unified solution for all failures
- Bursty communication

- Probabilistic approach
  - ➔ No upper bound on transmission delays
- Requires N+1 messages

  (best case)
- Caters to every kind of failure
- Staggered communication

Motivation for Probabilistic Synch.

- 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



System Model

- Correct clocks still have bounded drift
- No longer a maximum communication delay
  - delays given by probability distribution
- There is a known minimum message delay t_{min}

Outline

- Probabilistic Clock Reading, (2 processors)
- Optimizing probabilistic clock reading
- Round Message exchange protocol
- Failure algorithm classes

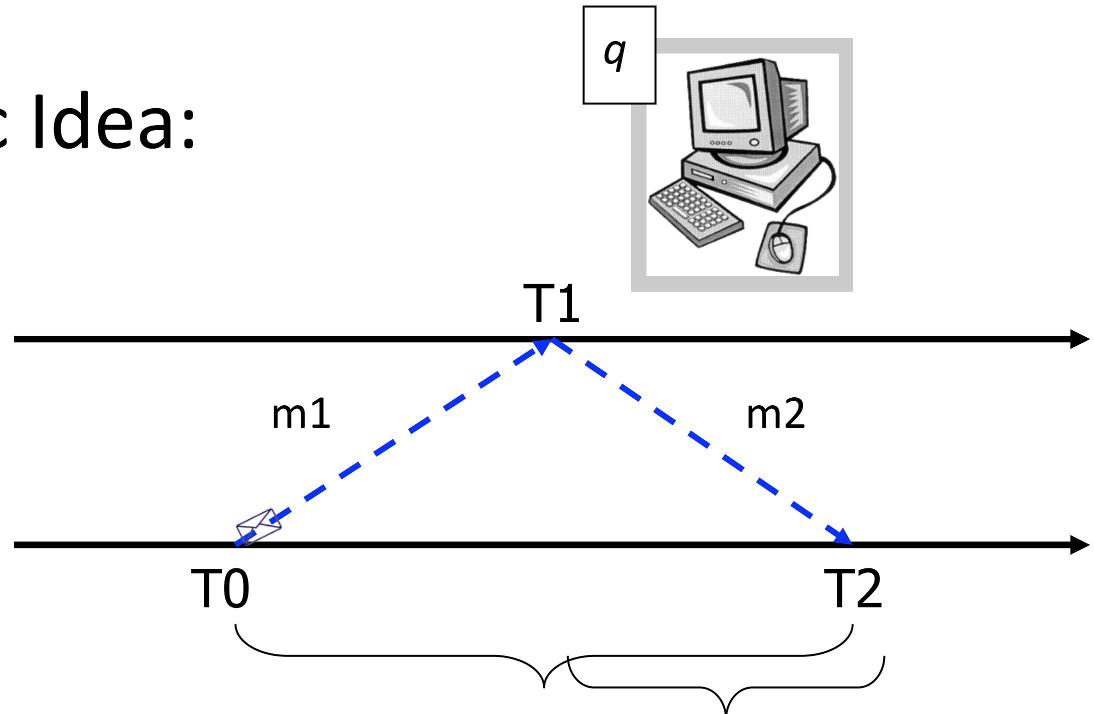
Contents of Exchanged Messages

Message (p->q)
Send time
Estimation of all clocks
Error bounds
Receive time stamps of q

If q trusts p can also use it to approximate other clocks.

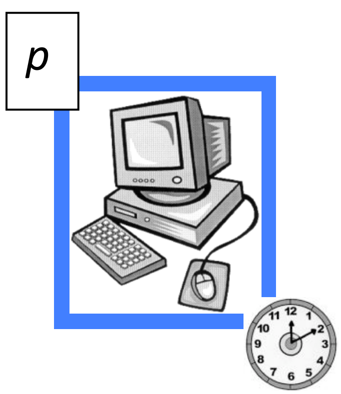
Probabilistic Clock Reading

- Basic Idea:



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

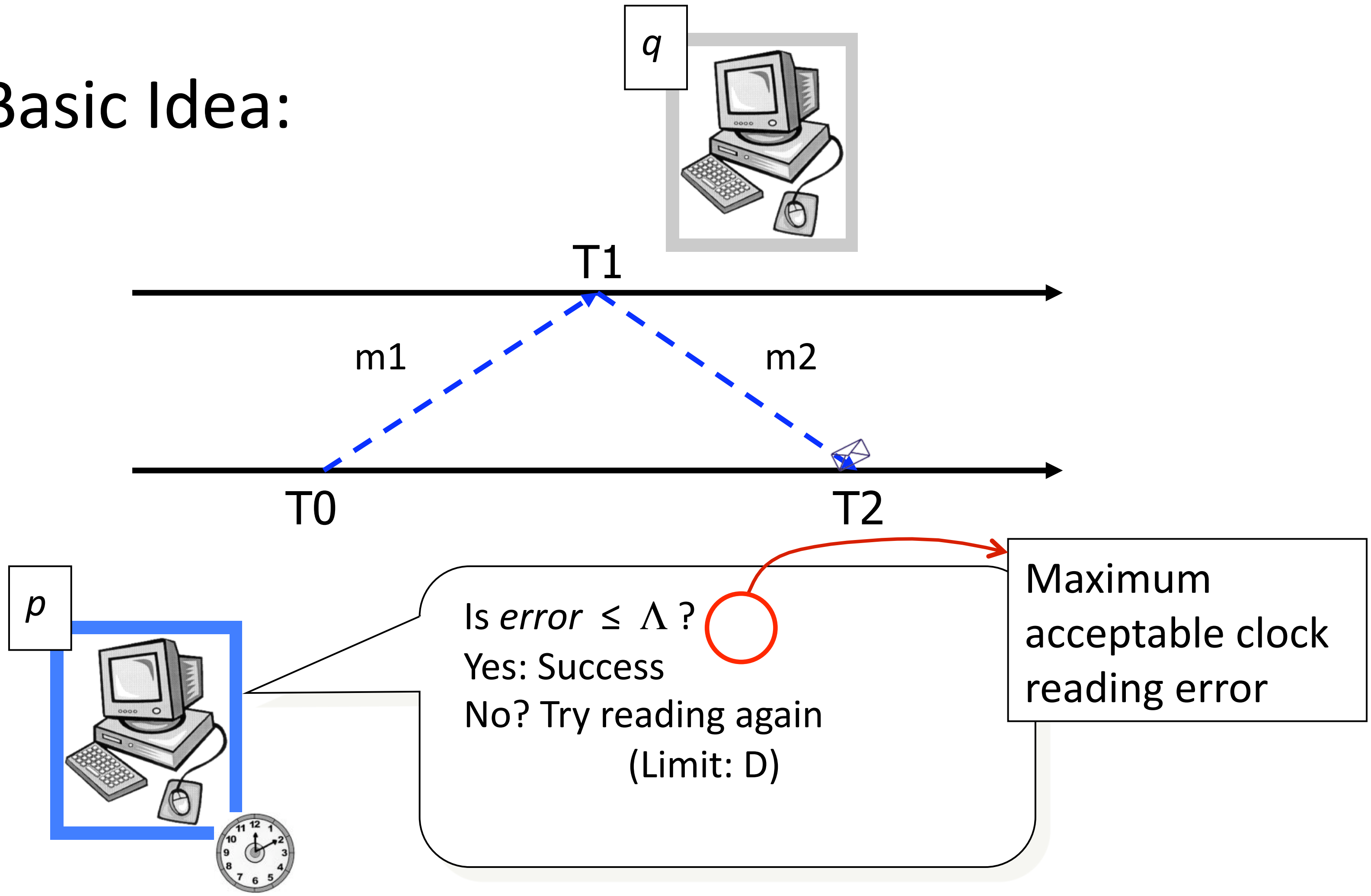
$min \leq t(m_2) \leq (T2 - T0)(1 + \rho) - min$



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

Probabilistic Clock Reading

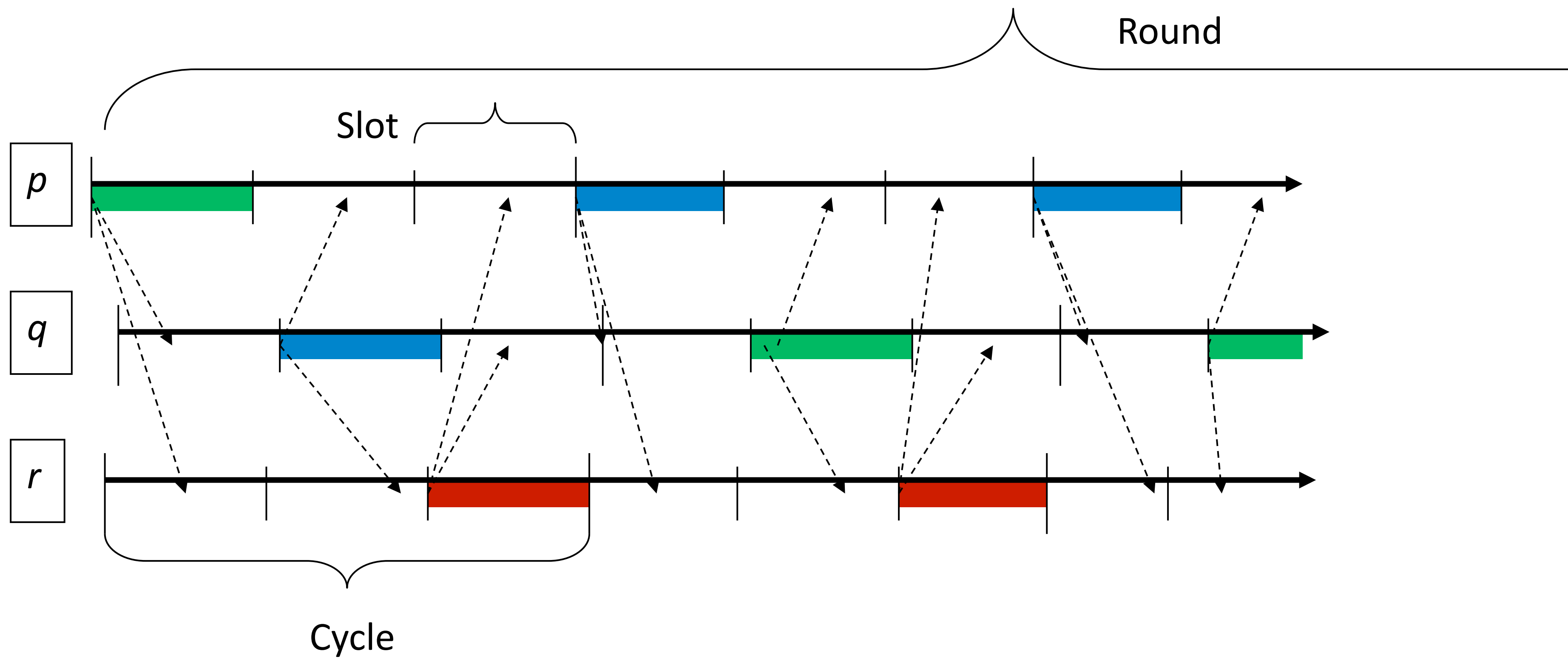
- Basic Idea:



Techniques to Optimize Probabilistic Clock Reading

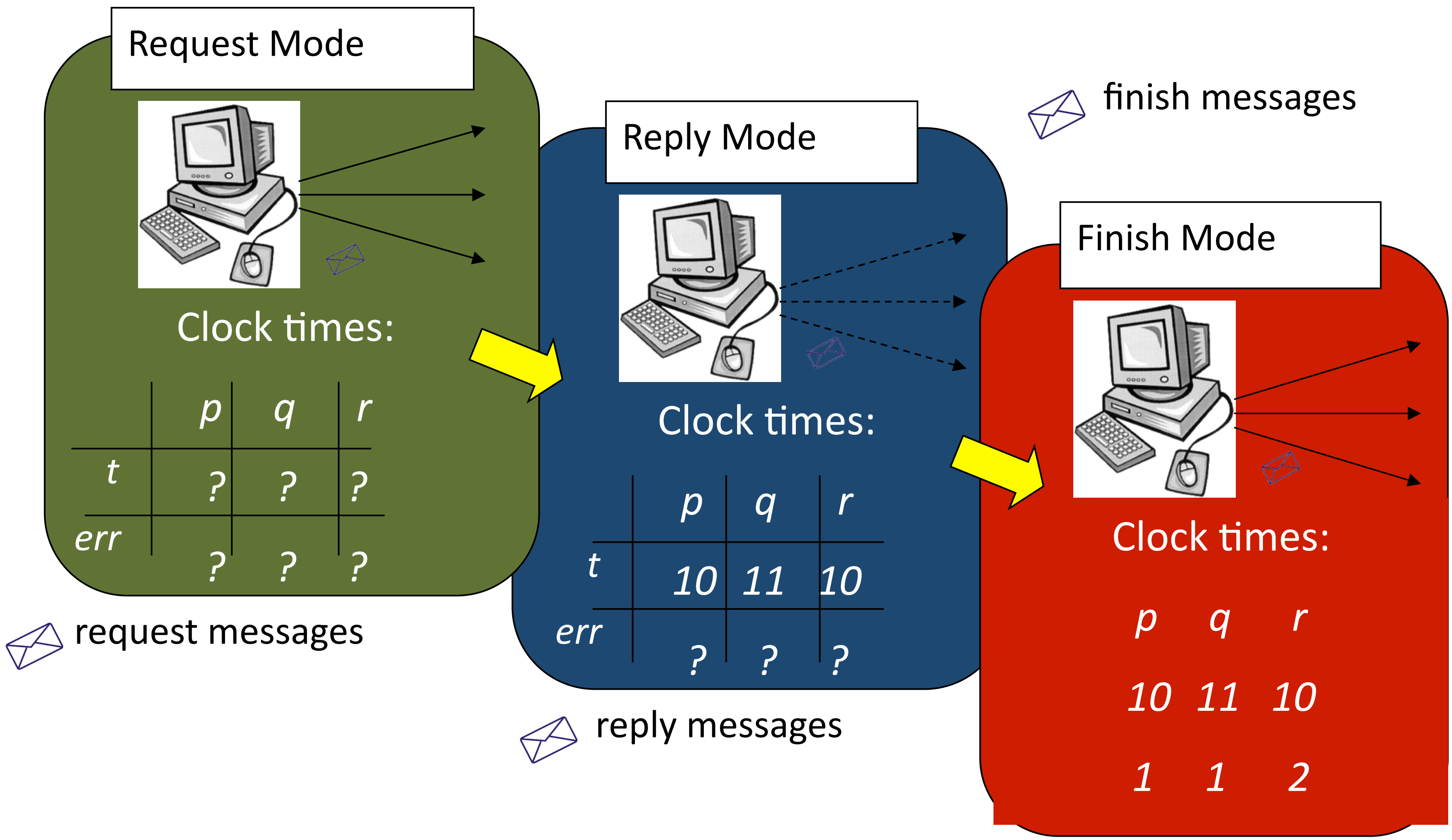
- Use all potentially non-concurrent messages
  - Helpful to approximate the sender's/receiver's clocks
- Stagger the messages
  - Increases no. of non concurrent messages, reduces n/w congestion
- Transitive Remote clock reading
  - Possible when no arbitrary failures

Staggering Messages



A slot is a unit in which a single process gets to send
A cycle is a unit in which all processes get a chance to send
A round is a unit in which all processes must get estimates of other clocks

Round Message Exchange Protocol



Failure Classes

Algo	Tolerated Failures	Required Processes	Tolerated Failures
CSA Crash	F	F+1	Crash
CSA Read	F	2F+1	Crash, Reading
CSA Arbit.	F	3F+1	Arbit, Reading
CSA Hybrid	Fc, Fr, Fa	3Fa+2Fr+Fc+1	Crash, Read, Arb

Take away

- 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?

References

- Thanks to
  - Lakshmi Ganesh
  - Michael George
- Papers:

  - Optimal Clock Synchronization, Srikanth and Toueg. JACM 34(3), July 1987.
  - F. Schneider. Understanding protocols for Byzantine clock synchronization. Technical Report 87-859, Dept of Computer Science, Cornell University, Aug 1987.
  - Using Time Instead of Timeout for Fault-Tolerant Distributed Systems, Lamport. ACM TOPLAS 6:2, 1974.
  - Probabilistic Internal Clock Synchronization, Cristian and Fetzer.

Initialization and Integration

- Same algorithms are used
  - A process independently starts clock C^o
  - On accepting a message at real time *t,* it sets

    $C^0 = \alpha$

- "Passive" scheme for integration of new processes
  - Joining process find out the current round number
  - Prevents a joining process from affecting the correct processes in the system