

# Distributed Systems

Alistair Rendell

# Benefits of Distribution

- Fits an existing physical distribution
  - email system, devices in a large aeroplane etc
- Possible high performance due to potential high degree of parallel computing
- Possible high reliability due to redundancy of the hardware and/or software
- possible scalability
- Integration of a large number of heterogeneous nodes/devices tailored to specific needs

# What to Distribute

- State
  - common method on distributed databases, email
- Function
  - distributed methods on central data
- State and Function
  - client/server clusters
- None of the above
  - pure replication, redundancy

# Common Design Criteria

- Achieve decoupling/high degree of local autonomy
- Cooperation rather than central control
- Consider reliability
- Consider scalability
- Consider performance

# Common Phenomena

- Unpredictable delays (communication)
  - are we done yet?
- Missing or imprecise time-base
  - was there a causal relation?
  - was there a temporal relation?
- Partial failures
  - likelihood of individual failures increases
  - likelihood of complete failure decreases (in case of a good design)

# We Will Consider

- Distributed Mutual Exclusion
  - with synchronized clocks
  - with logical clocks
- Global Properties
  - Termination
  - Snapshot algorithms
- Consensus
  - Including crash resilience
- Transactions
  - Construction of composite atomic actions
  - How does a database management system allow multiple concurrent queries
- References
  - Chapters 10-12 in Ben-Ari
  - Chapters 18-21 in Bacon

# Distributed Mutual Exclusion (and clocks)

# Time

- What is the definition of time?

*“Time has been a major subject of [religion](#), [philosophy](#), and [science](#), but defining it in a non-controversial manner applicable to all fields of study has consistently eluded the greatest scholars.”, wikipedia*

- Physics

- Counting the number of repetitions of some standard event

- Philosophy

- One view: part of the fundamental structure of the universe, time persists like frames in a film strip, we jump from one to the other
- Opposing view: part of a fundamental intellectual structure (with space and number) within which humans sequence and compare events

- Religion

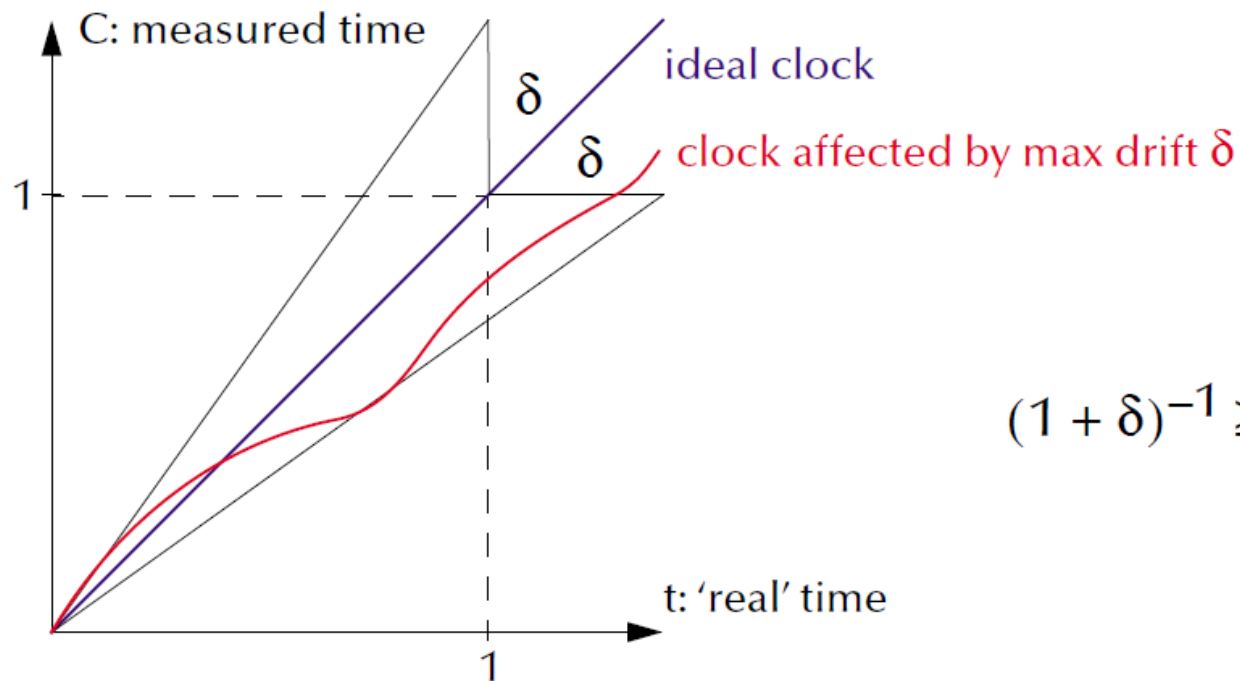
- Traditionally regarded as a medium for the passage of predestined events, e.g. time to give birth a time to die

# Time in Distributed Systems

- Synchronize clocks
  - Physics interpretation, measured in terms of some agreed repeating event
- Virtual time
  - Philosophical interpretation created solely to sequence events

# Real Clocks in Computer Systems

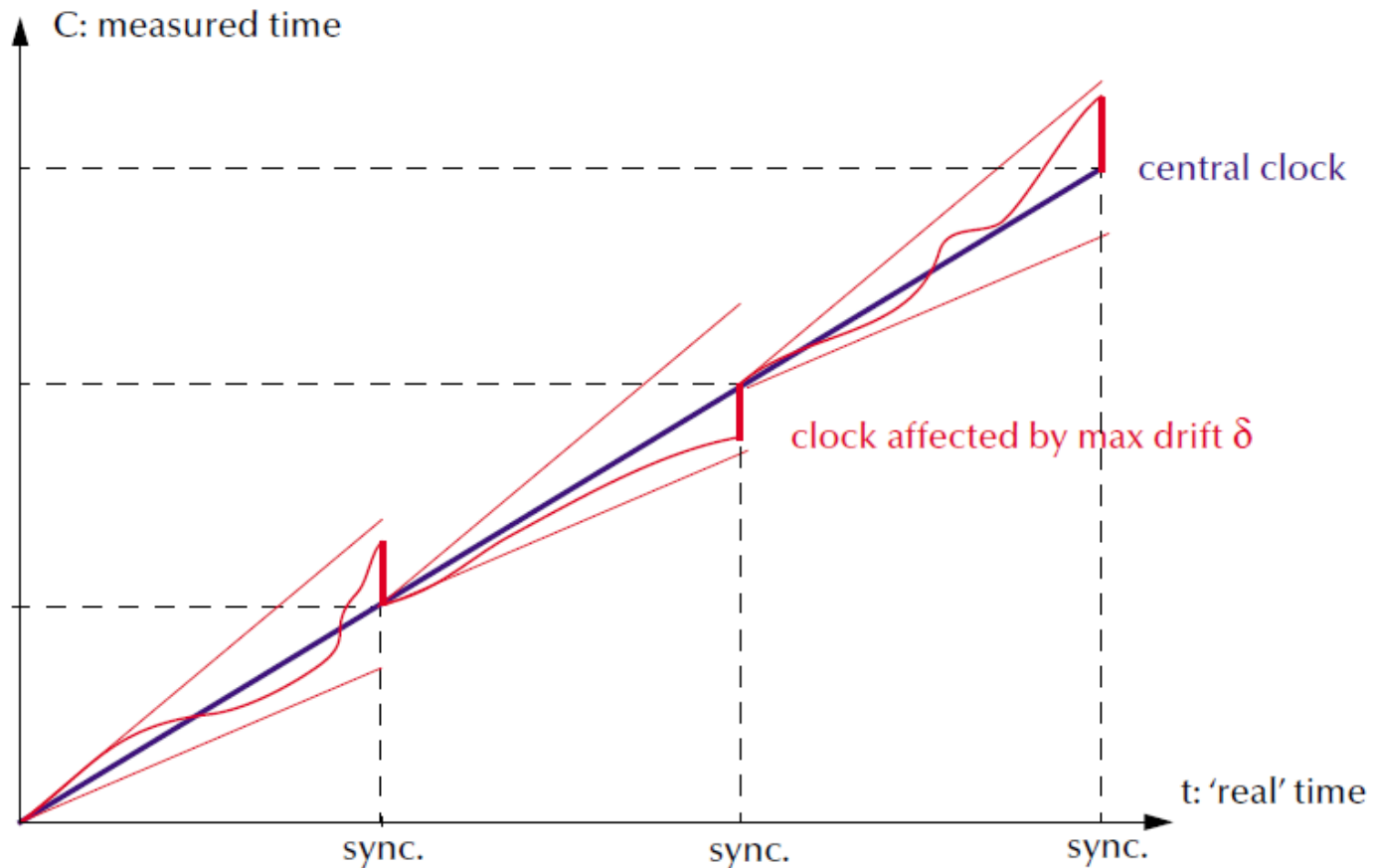
- Discrete, with minimal granularity
- Affected by drift



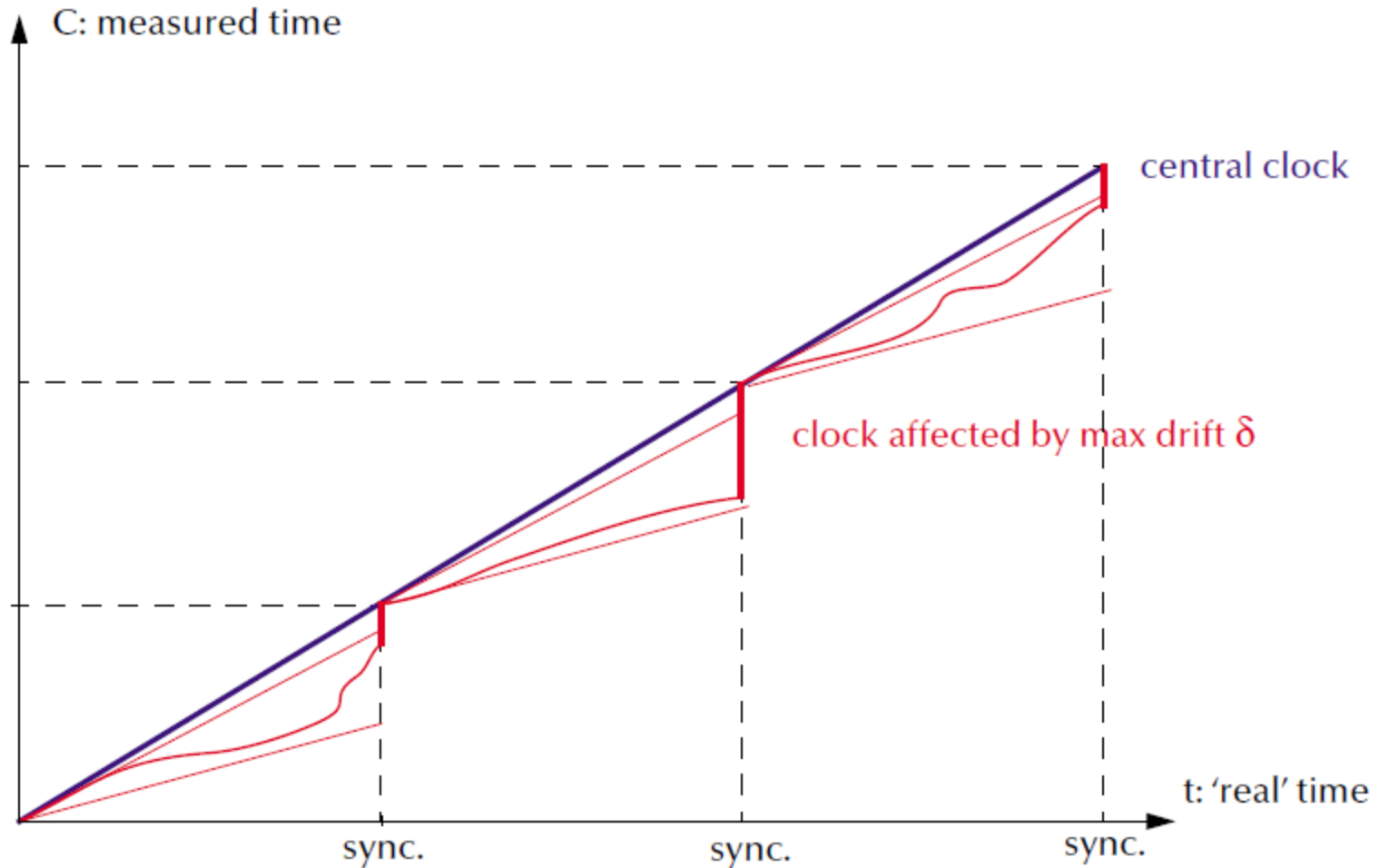
$$(1 + \delta)^{-1} \geq \frac{C(t_2) - C(t_1)}{t_2 - t_1} \geq (1 + \delta)$$

# Clock Synchronization

- Clocks can drift both ways
  - what are potential problems?



# Forward Only Clock Synchronization



# Mutual Exclusion with Synchronized Clocks

1. **Create** `OwnRequest` and **attach** current time-stamp
2. **Add** `OwnRequest` to local `RequestQueue` (ordered by time) and **send** own request to all processes
3. **Delay**  $2L$  ( $L$  being the time it takes for a message to reach all network nodes)
4. **Add** all received `Request` in local `RequestQueue` (ordered by time)
5. **While** `top(RequestQueue) ≠ OwnRequest` **do**
  - a) for all received release messages **delete** corresponding request in local `RequestQueue`
6. **Enter** and **Leave** critical region
7. **Send** `Release` message to all processes

# Analysis

- No deadlock, individual starvation or livelock
- Minimal request delay:  $2L$
- Minimal release delay:  $L$
- Communications requirements per requesting process:  $2(N-1)$  messages
  - can be improved by employing broadcast mechanisms
- Assumptions
  - $L$  is known and constant
  - no messages are lost

# Virtual (logical) Time (Lamport 1978)

- $a \rightarrow b \Rightarrow C(a) < C(b)$ 
  - $a \rightarrow b$  means a causes b
  - $C(a)$ ,  $C(b)$  are virtual times (or clocks) associated with events a and b
- $a \rightarrow b$  holds when
  - a happens earlier than b in the same sequential process
  - a denotes the event of sending a message m, while b denotes the event of receiving message m (in a different processes)
  - there is a transitive causal relation:  $a \rightarrow e_1 \rightarrow e_2 \dots \rightarrow e_n \rightarrow b$
- $a || b \Rightarrow \neg(a \rightarrow b) \wedge \neg(b \rightarrow a)$ 
  - We consider the two events concurrent if the above holds

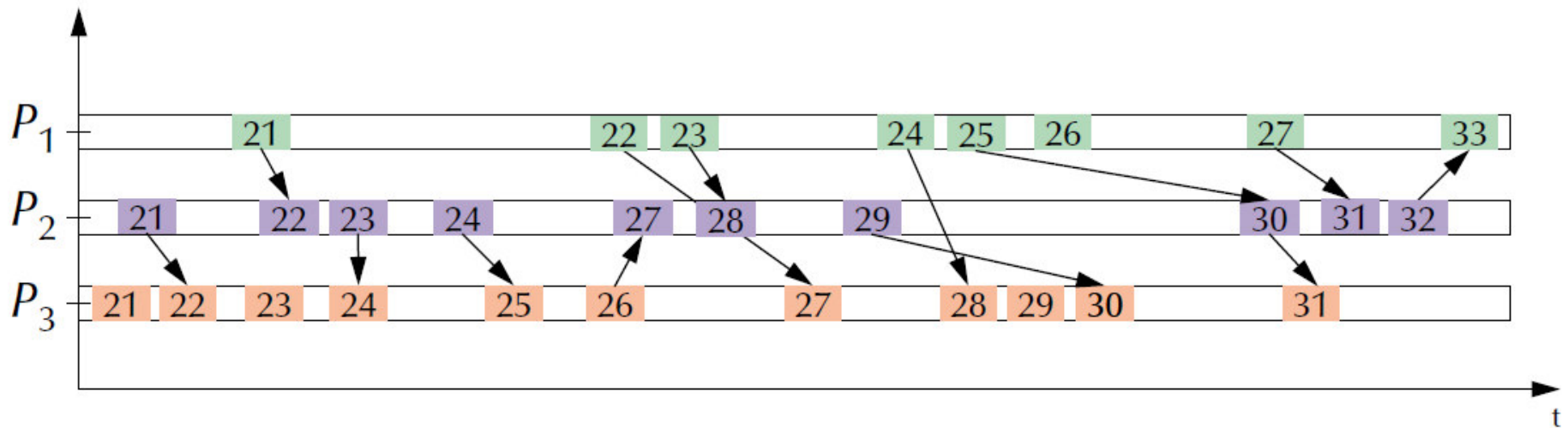
# Reverse Implications

- What can we deduce from relative values of logical clocks

$$a \rightarrow b \Rightarrow C(a) < C(b)$$

- $C(a) < C(b) \Rightarrow (a \rightarrow b) \vee (a \parallel b)$ 
  - either a causes b, or a was concurrent with b
- $C(a) = C(b) \Rightarrow (a \parallel b)$ 
  - events a and b were concurrent

# Graphical Representation



- Note all processes start with same time, but can jump reflecting relation between events
  - time is no longer global but is attached to observable causal relations

# Implementing Virtual/Logical Time

1.  $\forall P_i :$

$$C_i = 0$$

2.  $\forall P_i :$

a)  $\forall$  local events:

$$C_i = C_i + 1$$

b)  $\forall$  send  $m$  operations:

$$C_i = C_i + 1;$$

Send  $(m_i, C_j)$

c)  $\forall$  receive  $m$  operations:

Receive  $(m, C_m);$

$$C_i = \max(C_i, C_m) + 1$$

# Mutual Exclusion with Logical Clocks

## Concurrently

- Request message received
  1. **Add** request in local RequestQueue (ordered in time)
  2. **If** OwnRequest pending **reply** with OwnRequest **else** **reply** with Ack
- Release message received
  1. **Delete** corresponding Request in local RequestQueue
- If access to critical region required
  1. **Create** OwnRequest and attach current time-stamp
  2. **Add** OwnRequest to local RequestQueue (ordered by time) and **Send** OwnRequest to all processes
  3. **Wait** for  $\text{Top}(\text{RequestQueue}) = \text{OwnRequest}$  & no outstanding replies
  4. **Enter** and **leave** critical region
  5. **Send** Release message to all processes

# Analysis

- No deadlock, individual starvation or livelock
- Minimal request delay:  $N-1$  request messages,  $N-1$  reply messages
- Minimal release delay:  $N-1$  release messages
- Total communications requirement per requesting process:  $3(N-1)$  messages
  - can be improved by employing broadcast mechanisms
- Assumptions
  - no messages are lost
- No assumptions about
  - runtime of messages over the communication system

# Mutual Exclusion with Tokens

- Organize all processes in to a ring (physically or logically)
- Pass a token message along the ring
- On receiving the token
  - if the local process wants to enter a critical section it does so, while keeping the token
  - if not it passes the token on
- What happens if token is lost?

# Mutual Exclusion with Central Coordinator

- A central coordinator invalidates the concept of a distributed system, but enables very simple implementation of mutual exclusion
  - pronounce one process the central coordinator
  - provide a means of recovery if central process fails

# Electing a Central Coordinator: The Bully Algorithm

- Any process  $P$  which notices that the central coordinator is down:
  - Sends an `Election` message to all processes with higher process number
  - $P$  waits for response to messages
    - If no one responds after a pre-defined amount of time,  $P$  declares itself the new coordinator and sends out a `Coordinator` message to all
    - If any process responds the election activity for  $P$  is over and  $P$  waits for a `Coordinator` message
- All processes  $P_i$ :
  - If  $P_i$  receives an `Election` message from a process with a lower process number, it responds to the originating process and starts an election process itself (if not running already)