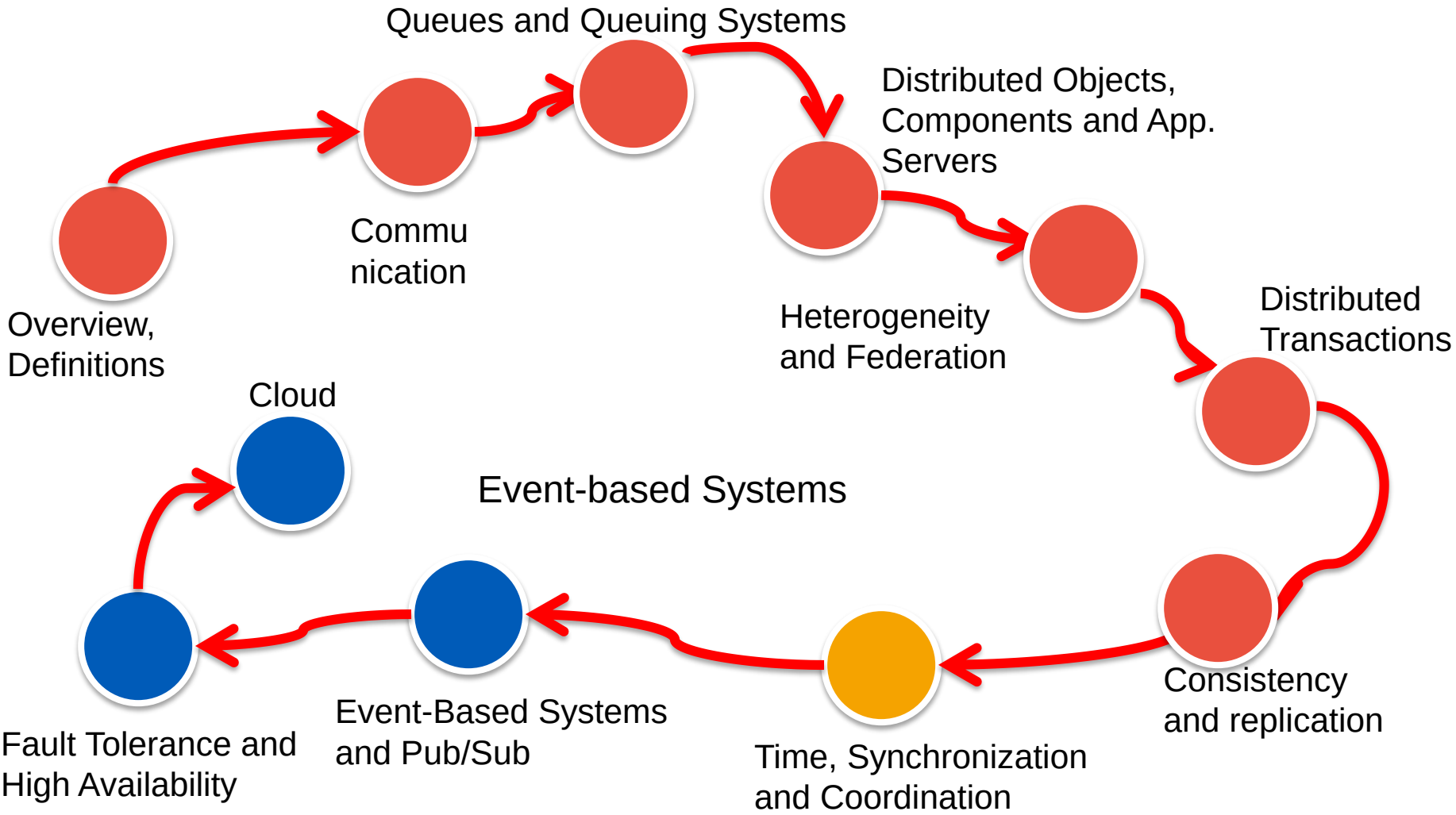
Middleware:
8. Time, Synchronization, Coordination

TECHNISCHE UNIVERSITÄT DARMSTADT

A. Buchmann
Wintersemester 2014/2015

Topics


Queues and Queuing Systems
Distributed Objects, Components and App. Servers
Commu nication
Overview, Definitions
Heterogeneity and Federation
Distributed Transactions
Cloud
Event-based Systems
Consistency and replication
Fault Tolerance and High Availability
Event-Based Systems and Pub/Sub
Time, Synchronization and Coordination

Reading for THIS Lecture

TECHNISCHE UNIVERSITÄT DARMSTADT

- The slides for the lecture are based on material from:
  - Andrew S. Tanenbaum and Maarten Van Steen. 2001. **Distributed Systems: Principles and Paradigms.** Prentice Hall.
    - Chapter 5
  - George Coulouris, Jean Dollimore, and Tim Kindberg. 2005. **Distributed Systems: Concepts and Design.** Addison-Wesley Longman.
    - Chapter 11

Topics

TECHNISCHE UNIVERSITÄT DARMSTADT

- Time is an integral part of the world, models, systems and events

- **Time**
  - Issues in distributed systems
  - Global time vs. local time, physical time vs. logical time
  - Influence on messaging mechanisms or guarantees (ordering, multicast, at-most-once)

- **Synchronization** in middleware and distributed systems

- **Coordination**

Synchronization

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Why do processes synchronize in middleware?**
  - To coordinate access of shared resources
  - To coordinate/synchronize execution
  - To order events

- Example
  - How do processes appoint a coordinator?
  - How can mutual exclusion be implemented?
  - How can events be ordered?

Time and Clocks

TECHNISCHE UNIVERSITÄT DARMSTADT

Created with wordle.net based on:
P. Bernstein. Middleware. CACM, Feb. 1996

DVS

Time, Clocks and Clock Synchronization

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Time**
  - Why is time important in middleware?
  - consider distributed Version Control or the UNIX Make tool
- **Clocks (Timer)**
  - Physical clocks
  - Logical clocks (introduced by Leslie Lamport)
  - Vector clocks (introduced independently by Collin Fidge and Friedemann Mattern)
- **Clock Synchronization**
  - How do we synchronize clocks with real-world time?
  - How do we synchronize clocks with each other?

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

DVS

Physical Clocks

TECHNISCHE UNIVERSITÄT DARMSTADT

Problem: Clock Skew – clocks gradually get out of synch and report different values

Solution: Universal Coordinated Time (UTC, formerly called GMT):

- Based on the number of transitions per second – atomic clock.
  - International Atomic Time (TAI) – std. sec = 9 192 631 770 oscilations Cs^{133}
- Introduces a leap second from time to time to compensate that days are getting longer.

- UTC is **broadcast**
  - Land-based short wave radio → accuracy of +/- 1 msec (MAX +/- 10 msec) and
  - satellite (GEOS) →accuracy of +/- 0.1 msec , GPS +/- 0.5 msec

- **Question:** What problems does this solve/not solve?
  - too coarse for synchronization of many kinds of event (e.g. 1 000 000 ops/sec)

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

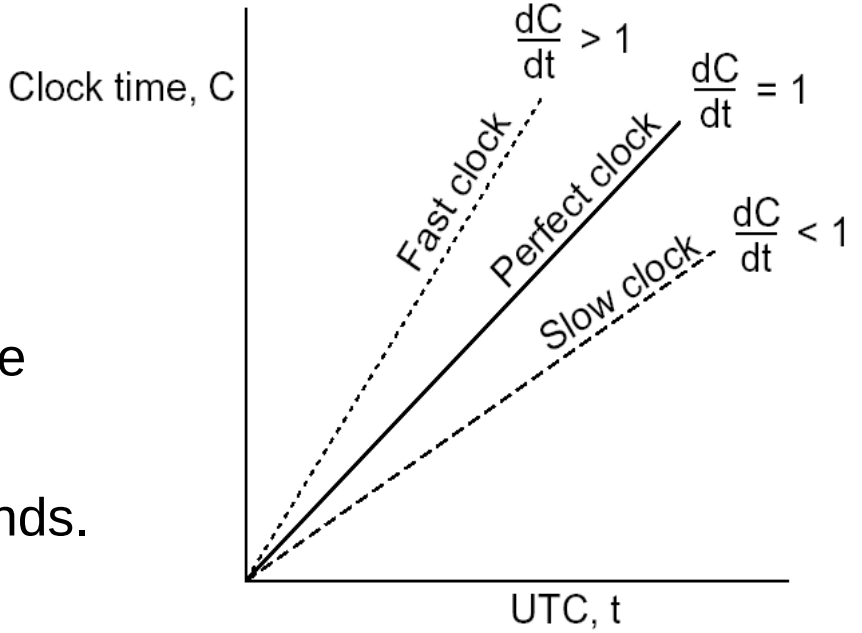
DVS

Physical Clocks – Basic Principle

TECHNISCHE UNIVERSITÄT DARMSTADT

- Every machine has a timer
  - generates an interrupt H times (typically 60) per second
- There is a clock in machine p that ticks on each timer interrupt. Denote the value of that clock by Cp(t) , where t is UTC time.
- Ideally, we have that for each machine p, Cp(t) = t → dC/dt = 1
  - In reality → relative error of 10^{-5}
- If ρ **is the max. drift rate**

$$1-\rho \leq \frac{dC}{dt} \leq 1+\rho.$$



- **Goal:** Never let two clocks differ by more than δ time units
- → synchronize at least every $\delta/2\rho$ seconds.

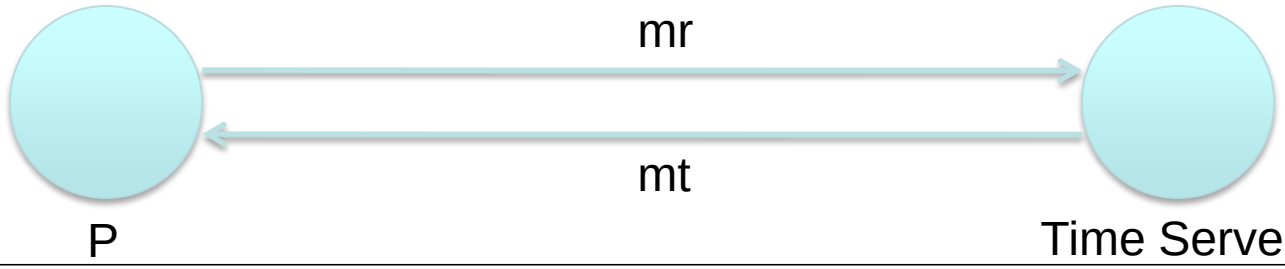
Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

DVS

Clock Synchronization – Cristian's Algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

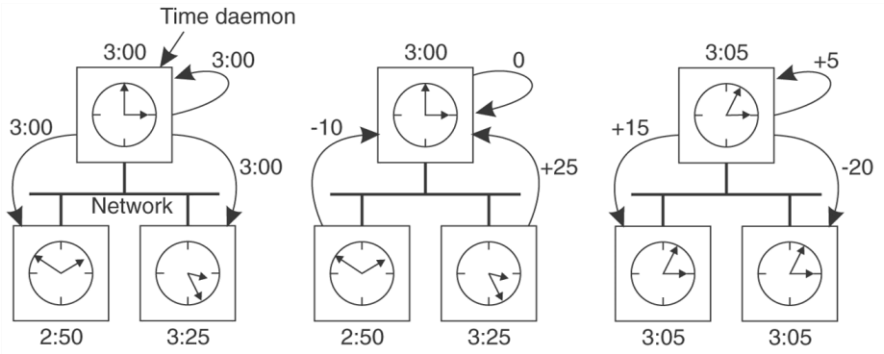
- Time adjustment
  - Slow clock → match UTC
  - Fast clock → slow down clock till match occurs
- Cristian's algorithm → Every machine asks a **time server** for the accurate time at least once every $\delta/2\rho$ seconds
  - Handling round-trip delays?
  - Delays occur because of contention at server and network
    - Ideally: P sends message at t, receives answer at t + T_{round}
    - Really: T_{round} = min + x where x >= 0
    - accuracy = +/- ($T_{round}/2$ – min)



DVS

Clock Synchronization – Berkeley Algorithm

- The Berkeley Algorithm addresses single point of failure and malicious node issues
  - The time server (master) polls slaves periodically → Received times are averaged and each machine is notified about individual compensation
  - Centralized algorithm



- Decentralized Algorithm
  - Every machine broadcasts its time periodically for fixed length synchronization interval
  - Averages the values from all other machines
- Network Time Protocol (NTP) → +/- 50msec
  - the most popular one used by the machines on the Internet
  - uses an algorithm that is a combination of centralized/distributed

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

Logical Clocks

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Problem**: What is **ordering?**
  - Is agreement on time needed (clock sync) or just the relative order of occurrence
- The **happened-before relation** on the set of events in a distributed system is the relation satisfying:
  - If **a** and **b** are two events in the same process, and **a** comes before **b,** then **a → b**. (a happened before b)
  - If **a** is the sending of a message, and **b** is the receipt of the message: **a → b**.
  - If **a → b** and **b → c**, then **a → c**. (transitive relation)

- If two events, **x** and **y,** happen in different processes that do not exchange messages, then they are **concurrent.**

- This mechanism introduces a **partial ordering** of events in a system with concurrently operating processes. Why partial?

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

DVS

Logical Clocks

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Problem:** How do we maintain a global view on the system's behavior that is consistent with the happened-before relation?
- **Solution:** attach a timestamp $C(e)$ to each event e, satisfying the following properties:
- **P1:** If a and b are two events in the same process, and $a \rightarrow b$, then we demand that $C(a) < C(b)$
- **P2:** If a corresponds to sending a message m, and b to the receipt of that message, then also $C(a) < C(b)$
- **Problem:** How do we attach a timestamp to an event when there's no global clock?

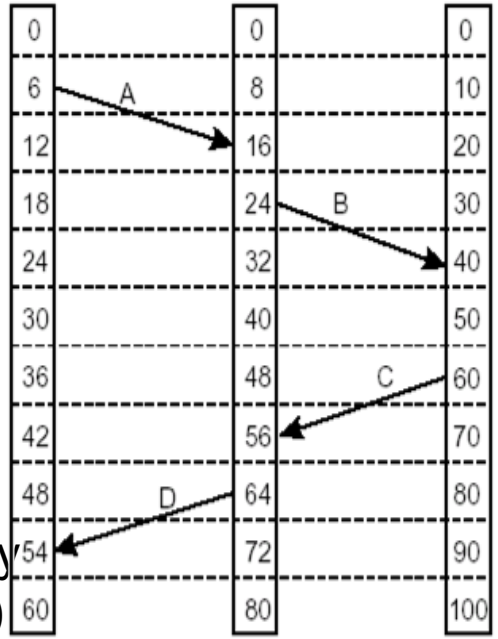
Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

DVS

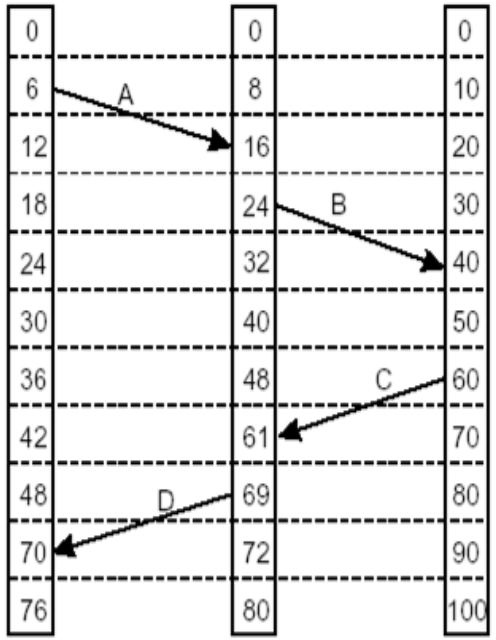
Logical Clocks – Lamport's algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Question**: can it cope with clock drift? Can it be used to correct the drift?
  - Can it implement Global Time?
- Total Ordering
  - Each message carries sender's clock
  - Upon arrival the receiver's clock = sender's clock +1
- Global Time:
  - between every two events the clock ticks at least once
  - events don't occur simultaneously (attach process number to event)



Situation: 3 Clocks + Drift



Solution: Lamport Clocks

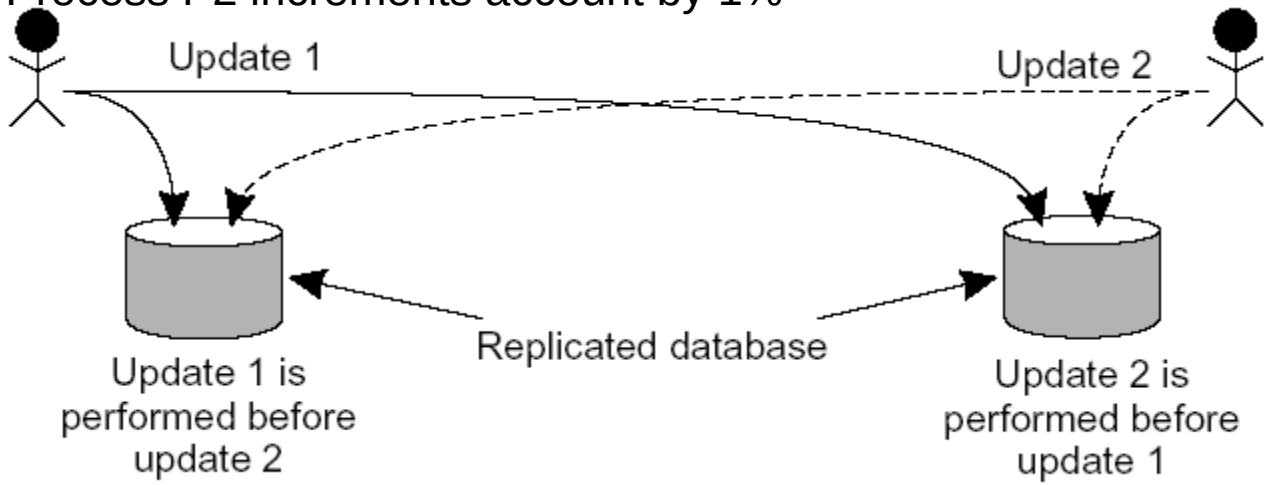
Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

DVS

Example: Totally-Ordered Multicast

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Problem: G**uarantee that concurrent updates on a replicated database are seen in the same order by all replicas. Example with 2 replicas:
  - Update1: Process P1 adds €100 to an account (initial value: €1000)
  - Update2: Process P2 increments account by 1%



- **Result: i**f NO synchronization → Replica1 = €1111, Replica2 = €1110
  - **Solution**: upon receipt of msg → place in queue that is ordered according to timestamp, every recipient (incl. sender) sends an ack. Eventually all queues converge. A message is only executed if it is at the head of the queue and was confirmed by all other recipients.

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

DVS

Vector Timestamps

TECHNISCHE UNIVERSITÄT DARMSTADT

- Lamport's algorithm does not capture causality:
  - If Timestamp(A) < Timestamp(B) → does this imply that A happened before B?
  - Consider posting messages on a forum and replying
    - Is the posting of a new message B a result of message A?
- Causality: Reaction to msg A should always follow the receipt of A
  - If msg A and msg B are independent → order of delivery irrelevant
- Vector Clocks [Fidge, independent solution Mattern]
  - A vector timestamps VT(E), VT(F) for events E,F → VT(E) < VT(F) if causal dept.
  - Each process P has a vector V
    - V[i] – number of events occcured at P
    - If V[j] = K → P knows that K events have occured
    - Important to know how many messages at other nodes/process P had seen when he sent his message → send vectors with message
    - Receiving process knows now for how many events at a node it must wait to be in the same state as the sender

Colin Fidge, "Logical Time in Distributed Computing Systems", *IEEE Computer*, Vol. 24, No. 8, pp. 28-33, 1991

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

Vector Timestamps

TECHNISCHE UNIVERSITÄT DARMSTADT

The Fidge logical clock is maintained as follows:

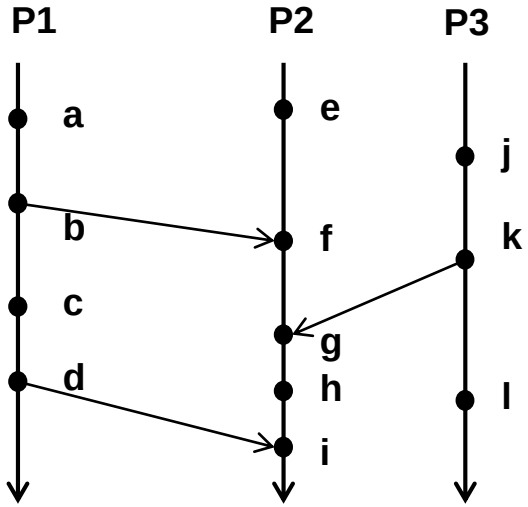
1. Initially all clock values are set to the smallest value.
2. The local clock value is incremented at least once before each primitive event in a process
3. The current value of the entire logical clock vector is delivered to the receiver for every outgoing message.
4. Values in the timestamp vectors are never decremented.
5. Upon receiving a message, the receiver sets the value of each entry in its local timestamp vector to the maximum of the two corresponding values in the local vector and in the remote vector received.
6. The element corresponding to the sender is a special case; it is set to one greater than the value received, but only if the local value is not greater than that received.

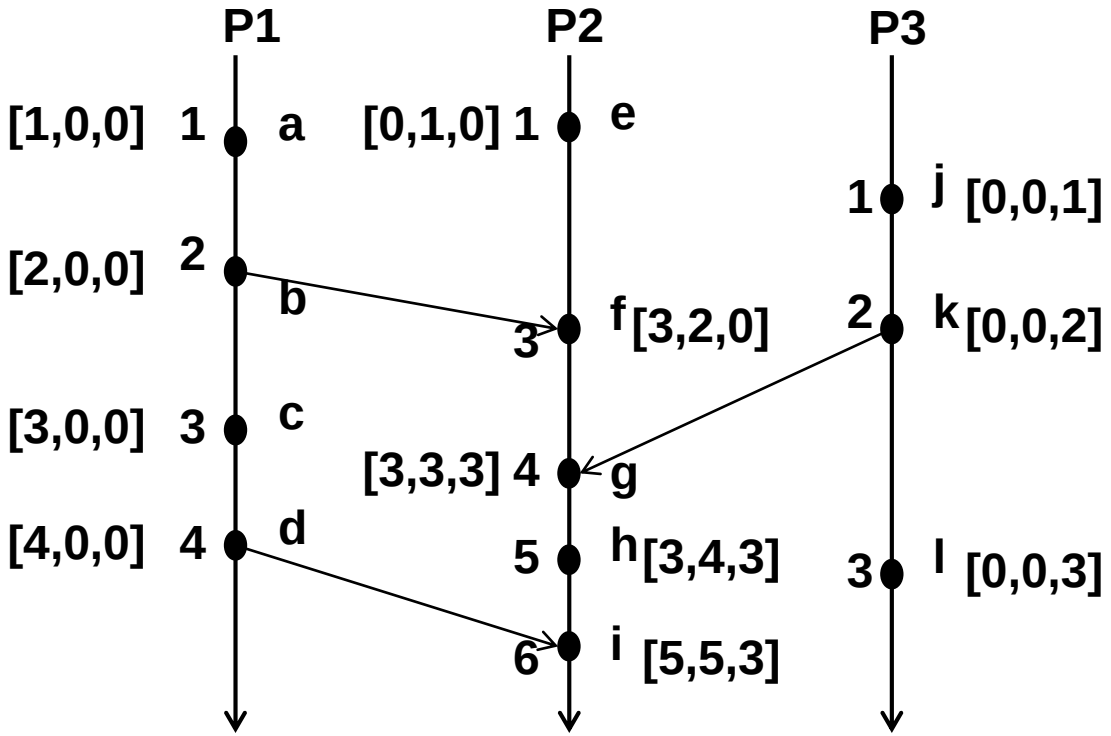
Colin Fidge, "Logical Time in Distributed Computing Systems", *IEEE Computer*, Vol. 24, No. 8, pp. 28-33, 1991

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

Vector Timestamps

- Assign the Lamport and Fidge logical clock values for all the events.
  - The logical clock of each process is initially set to 0





Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

Vector Timestamps

TECHNISCHE UNIVERSITÄT DARMSTADT

- The above diagram shows both Lamport timestamps (an integer value) and Fidge timestamps (a vector of integer values) for each event.

- Lamport clocks:
  - 2 < 5 since b $\rightarrow$ h,
  - 3 < 4 but c <!> g.

- Fidge clocks:
  - f $\rightarrow$ h since 2 < 4 is true,
  - b $\rightarrow$ h since 2 < 3 is true,
  - h <!> a since 4 < 0 is false,
  - c <!> h since (3 < 3) is false and (4 < 0) is false.

Global State

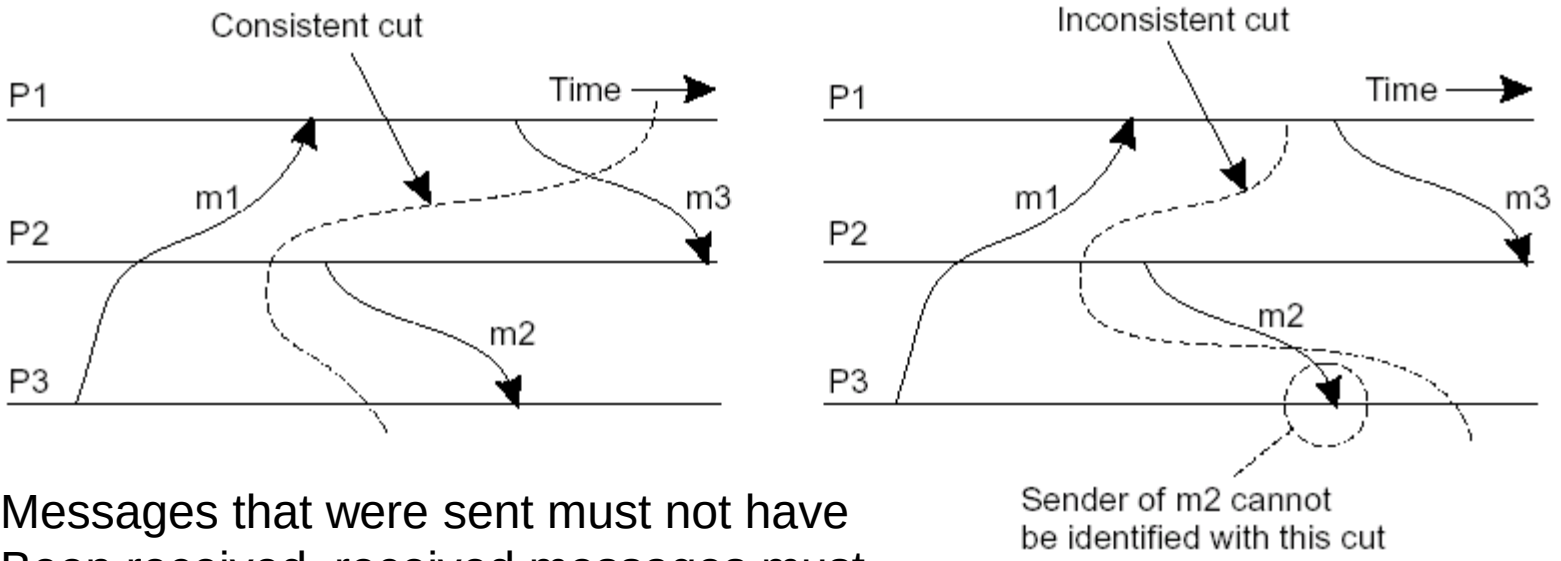
TECHNISCHE UNIVERSITÄT DARMSTADT

Created with wordle.net based on:
P. Bernstein. Middleware. CACM, Feb. 1996

DVS

Global State

- **Basic Idea:** Sometimes you want to collect the current state of a distributed computation, called a **distributed snapshot**. It consists of all local states and messages in transit.
- **Important:** A distributed snapshot should reflect a **consistent** state:



Messages that were sent must not have
Been received, received messages must
Have been recorded as sent!

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

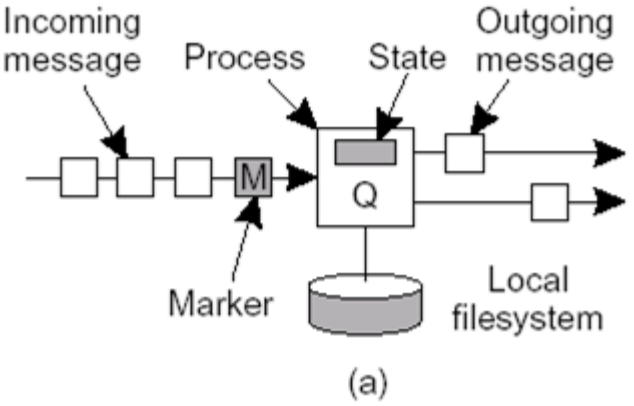
Global State

TECHNISCHE UNIVERSITÄT DARMSTADT

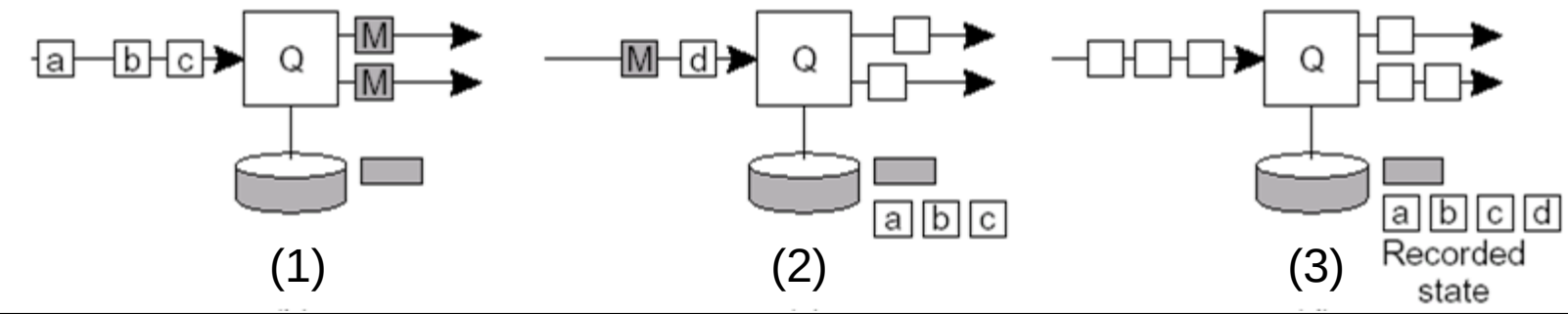
- **Note:** Processes are connected by unidirectional channels, any process P can initiate taking a distributed snapshot
- P starts by recording its own local state and subsequently sends a marker along each of its outgoing channels
- When Q receives a marker through inbound channel C, its action depends on whether it had already recorded its local state:
  - Not yet recorded: it records its local state, and sends the marker along each of its outgoing channels
  - Already recorded: the marker on C indicates that the channel's state should be recorded: all messages received before this marker and the time Q recorded its own state.
- Q is finished when it has received a marker along each of its incoming channels
- Parallel snapshots are possible, therefore markers have initiator ID

Global State

- Organization of a process and channels for a distributed snapshot



1. Process Q receives a marker for the first time and records its local state
2. Q records all incoming messages
3. Q receives a marker for its incoming channel and finishes recording the state of the incoming channel



Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

Election Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

DVS

Election Algorithms

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Principle:** Many distributed algorithms require that some process acts as a coordinator. The question is how to select this special process **dynamically**.

- **Note:** In many systems the coordinator is chosen by hand (e.g., file servers, DNS servers). This leads to centralized solutions => single point of failure.
- **Question:** If a coordinator is chosen dynamically, to what extent can we speak about a centralized or distributed solution?
- **Question:** Is a fully distributed solution, i.e., one without a coordinator, always more robust than any centralized/coordinated solution?

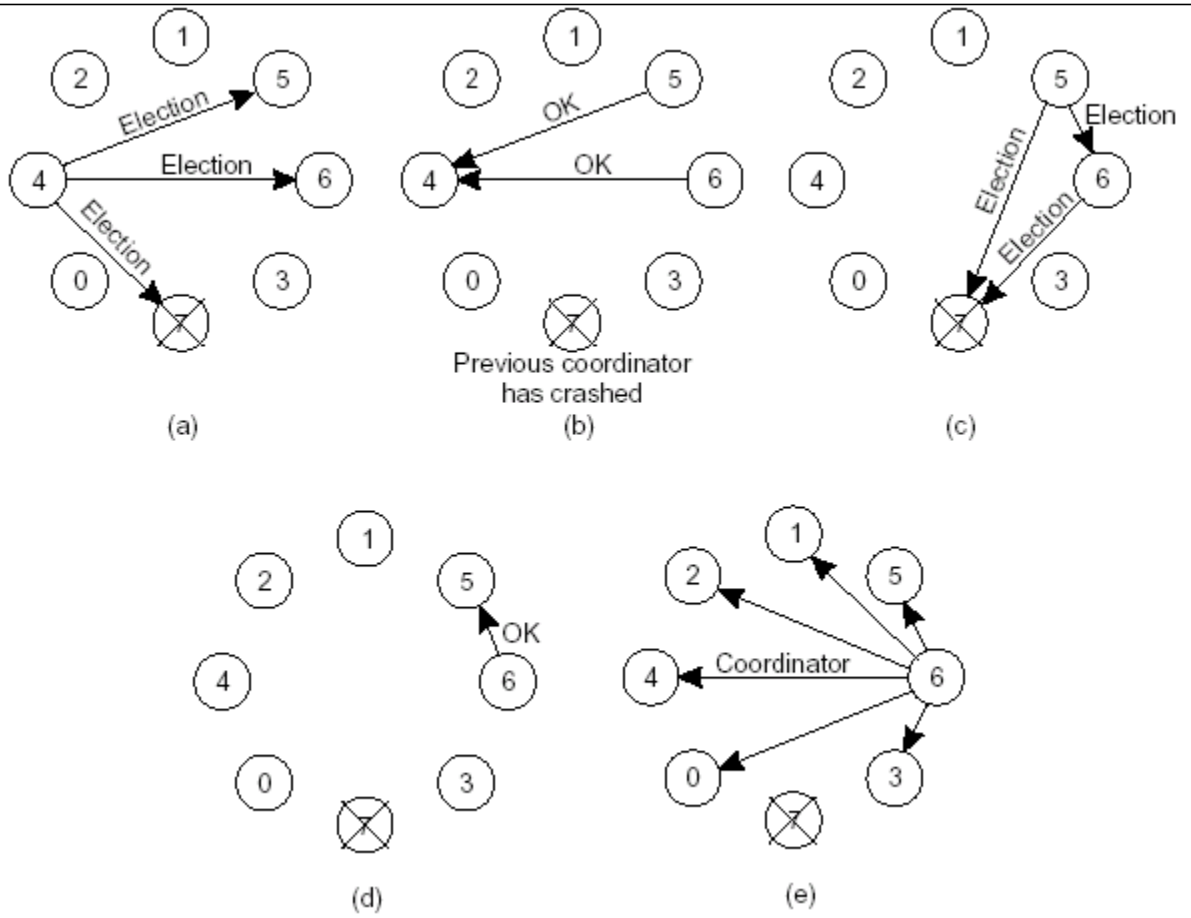
The Bully Algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Principle:** Each process has an associated priority (weight). The process with the highest priority should always be elected as the coordinator.
- **Issue:** How do we find the heaviest process?

- Any process can just start an election by sending an election message to all other processes (assuming you don't know the weights of the others).
- If a process *P*heavy receives an election message from a lighter process *P*light, it sends a take-over message to *P*light. *P*light is out of the race.
- If a process doesn't get a take-over message back, it wins, and sends a victory message to all other processes.

The Bully Algorithm



Question: We're assuming something very important here – what?

Assumption: Each process knows the process number of other processes

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

Election in a Ring

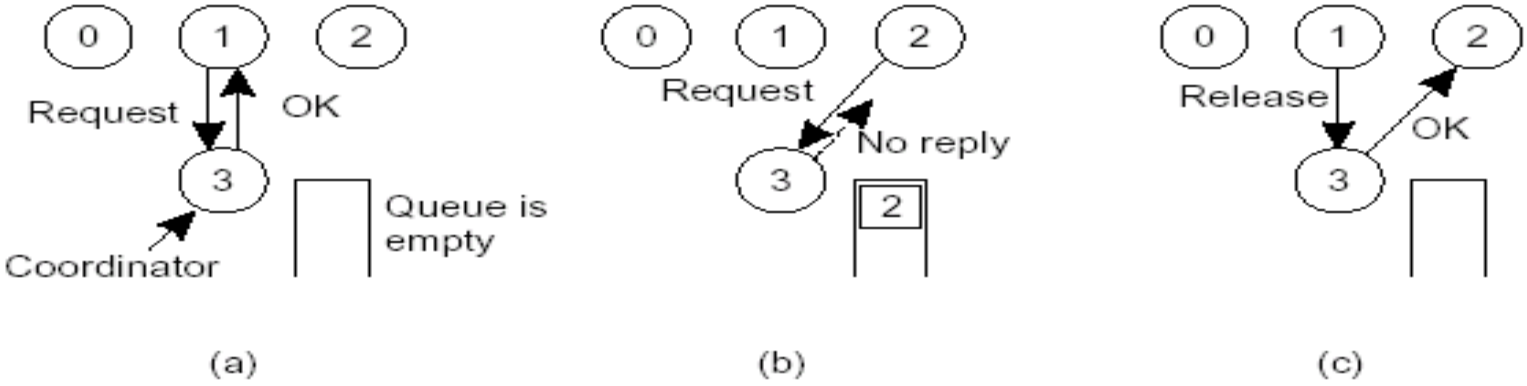
TECHNISCHE UNIVERSITÄT DARMSTADT

- **Principle**: Process priority is obtained by organizing processes into a (logical) ring. Process with the highest priority should be elected as coordinator.
- Any process can start an election by sending an election message to its successor. If a successor is down, the message is passed on to the next successor.
- If a message is passed on, the sender adds itself to the list. When it gets back to the initiator, everyone had a chance to make its presence known.
- The initiator sends a coordinator message around the ring containing a list of all living processes. The one with the highest priority is elected as coordinator.
- **Question**: Does it matter if two processes initiate an election?
- **Question**: What happens if a process crashes during the election?

Mutual Exclusion

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Problem:** A number of processes in a distributed system want exclusive access to some resource.
- **Basic solutions:**
- Via a centralized server.
- Completely distributed, with no topology imposed.
- Completely distributed, making use of a (logical) ring.
- **Centralized:** Really simple:

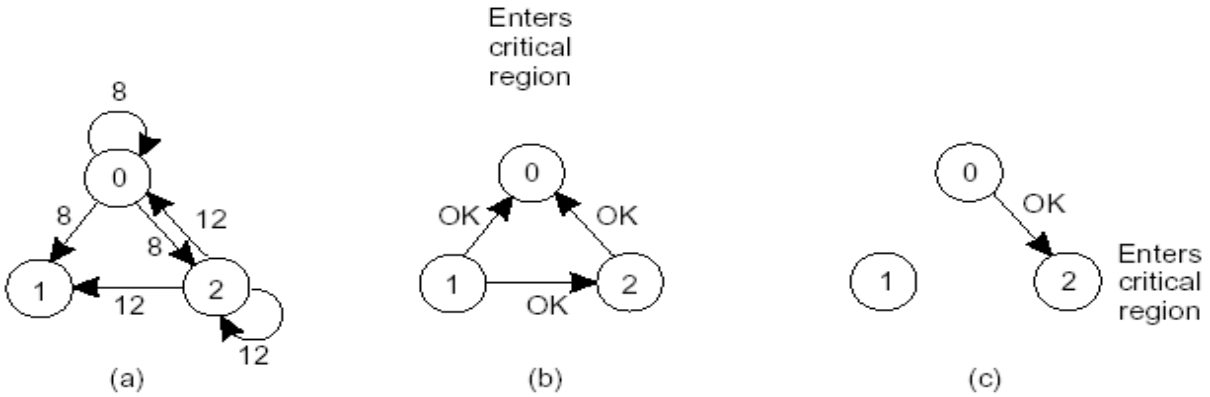


Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

DVS

Mutual Exclusion: Lamport

- A process wanting to enter a critical region
  - Sends a (reliable) message with resource sought, own PID, timestamp
  - Since messages are reliable an ack must be received
  - Upon receipt (depending on recipients status)
    - Not in critical region and not interested → send OK
    - Recipient in critical region → queue incoming message
    - Waiting to enter → compare TS of own message with received message, lowest wins
  - Sending process waits till it gets the OK from all other processes

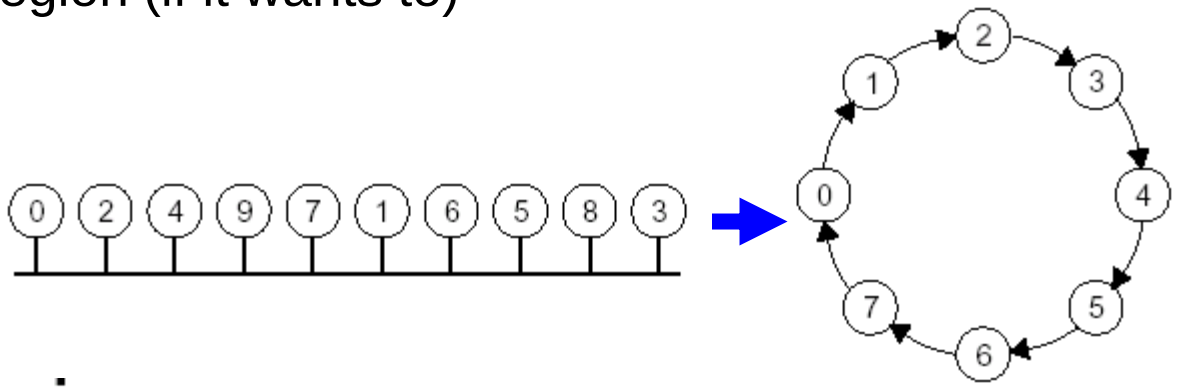


Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

Mutual Exclusion: Token Ring Algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Essence:** Organize processes in a **logical** ring, and let a token be passed between them. The one that holds the token is allowed to enter the critical region (if it wants to)



(a) (b)

Comparison:

Algorithm	# msgs	Delay	Problems
Centralized	3	2	Coordinator crash
Distributed	2 (n – 1)	2 (n – 1)	Crash of any process
Token ring	1 to ∞	0 to n – 1	Lost token, process crash

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms

DVS

Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

- Time issues
- Clocks
  - Physical Clocks
  - Logical Clocks
  - Synchronization
- Global State
- Election algorithms
- Mutual Exclusion

Thank You!

TECHNISCHE UNIVERSITÄT DARMSTADT

Questions?

DVS