Operating Systems

Distributed Systems

Stephan Sigg

Distributed and Ubiquitous Systems
Technische Universität Braunschweig

February 11, 2011

Overview and Structure

- Introduction to operating systems
  - History
  - Architectures
- Processes
  - Processes, Threads, IPC, Scheduling
  - Synchronisation
  - Deadlocks
- Memory management
  - Paging
  - Segmentation
- Filesystems
- Security and Protection
- Distributed systems
- Cryptography

Outline

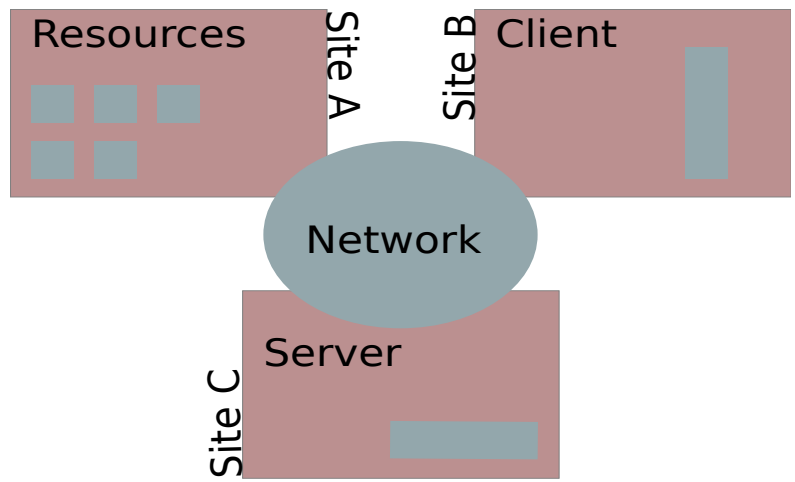
Distribured systems

1. Distributed systems
   - Distributed systems
   - Distributed file systems
   - Distributed synchronisation

Distributed systems

Introduction

- A distributed system is a collection of loosely coupled processors
- Interconnected by a communication network



Distributed systems

Introduction

Reasons for building distributed systems

- Resource sharing
- Computation speed-up
- Reliability
- Communication

Distributed systems

Introduction

Reasons for building distributed systems

Resource sharing :

- Different sites are connected to one another
- A user at one site may be able to use the resources available at another site

Distributed systems

Introduction

Reasons for building distributed systems

Computation speed-up :

- Partition computation into sub-computations can are run concurrently

Distributed systems

Introduction

Reasons for building distributed systems

Reliability :

- If one site fails in a distributed system, the remaining sites can continue operating
- This increases the reliability of the system

Distributed systems

Introduction

Reasons for building distributed systems

Communication :

- When several sites are connected, users at different physical locations can communicate

Distributed systems

Introduction

Types of network-based operating systems

- Network operating systems
  - Remote login
  - Remote file transfer
- Distributed operating systems
  - Data migration
  - Computation migration
  - Process migration

Distributed systems

Introduction

Network operating systems

- The network operating system provides an environment in which users can access remote resources.

Remote login :

- Users log in remotely
- Typically login and password required
- After login, the process used to access acts as a proxy for the user
- The remote user can provide any actions on the remote machine like a local user

Remote file transfer :

- File transfer from one machine to another
- ftp, scp, ...

Distributed systems

Introduction

Distributed operating systems

- In a distributed operating system, users access remote resources in the same way they access local resources
- Data and process migration is under the control of the distributed operating system

Data migration :

- User could work on a local copy of remote files
- Or only small parts of file are transferred and modifications are forwarded to remote host

Distributed systems

Introduction

Distributed operating systems

- In a distributed operating system, users access remote resources in the same way they access local resources
- Data and process migration is under the control of the distributed operating system

Computation migration :

- Transfer computational load rather than data
- Message transfer to trigger remote computation
- Or execute a routine on a remote system via RPC (Method invoked on local system and result is returned to the method from the remote system)

Distributed systems

Introduction

Distributed operating systems

- In a distributed operating system, users access remote resources in the same way they access local resources
- Data and process migration is under the control of the distributed operating system

Process migration :

- Extension of computation migration
- Used for Load balancing, computational speed-up, hardware/software preference, Data access

Outline

Distributed systems

1. Distributed systems
   - Distributed systems
   - Distributed file systems
   - Distributed synchronisation

Distributed systems

Distributed file systems

Definition

A service is a software entity running on one or more machines and providing a particular type of function to clients

Definition

A client is a process can invoke a service using a set of operations that form its client interface

Definition

A server is the service software running on a single machine

Distributed systems

Distributed file systems

- A file system provides file services to clients
- A client interface for a file service is formed by a set of primitive file operations (create, delete, read, write)
- The primary hardware components that a file server controls is a set of local secondary-storage devices on which files are stored
- A distributed file system is a file system whose clients, servers and storage devices are dispersed among the machines of a distributed system

Distributed systems

Distributed file systems

naming

- Naming is a mapping between logical and physical objects
- In a local file system, symbolical file names are translated to a numerical identifier that is in turn mapped to disk blocks
- In a distributed file system, another abstraction layer (network) is added to this naming scheme

Distributed systems

Distributed file systems

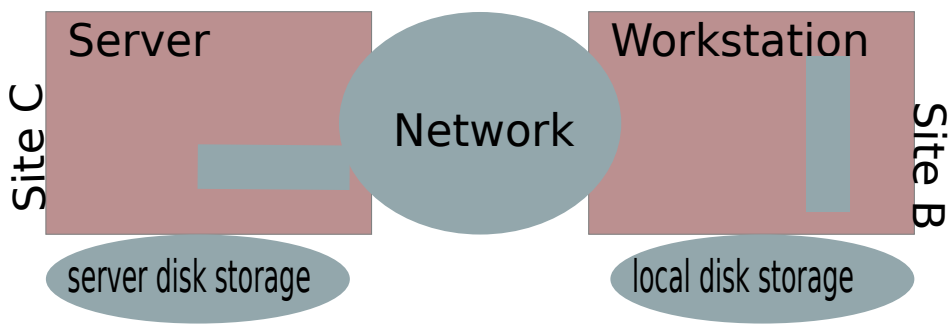
Remote file access

- When accessing a remote file, data from the file has to be transferred

  Basic caching scheme

  Cache update policy

  Consistency



Distributed systems

Distributed file systems

Basic caching scheme

- A copy of the data is brought to the client system
- Access is performed on the copy
- After some time or event the cached file is transferred back to the original location
- The challenge is to keep the cached copies and the master file consistent with reasonable network traffic

Distributed systems

Distributed file systems

Cache update policy

- The policy used to write modified data blocks back to the server's master copy has a critical effect on the system's performance and reliability
- Write through policy
  - Write data as soon as they are placed in any cache
  - Little information is lost when a client system crashes
  - Each write access has to wait until the information is sent to the server
- Delayed write policy(write-back caching)
  - Write data at a later time
  - Write accesses complete more quickly since the do not have to wait for data to be transferred
  - Data may be overwritten before they are written back
  - This reduces the required network traffic

Distributed systems

Distributed file systems

Consistency

- A client machine has to decide whether locally cached copy is consistent with master copy
- Client-initiated approach
  - Client initiates validity check if data is consistent
  - But: when and how often the check shall be done
  - Every access coupled with a validity check is delayed
- Server-initiated approach
  - Server records for each client the files that it caches
  - When the server detects a potential inconsistency, it must react (when two clients in conflicting modes access the file)
  - In UNIX, server and must be notified whenever a file is opened and the intended mode (read or write)
  - In conflicting cases, the server could e.g. disable caching for the particular file

Outline

Distributed systems

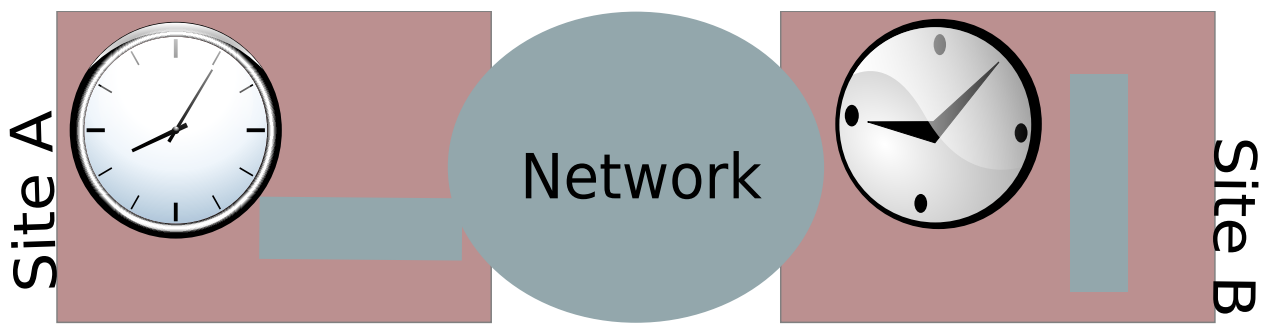
1. Distributed systems
   - Distributed systems
   - Distributed file systems
   - Distributed synchronisation

Distributed systems

Distributed synchronisation

Event ordering

- In a distributed system it might be hard to say for two events, which one occurred first
- The reason is, that no common clock may exist for these two events



Event ordering – The happened-before relation

- All events executed in a single process are totally ordered
- The happened before relation is defined as
  - If A and B are events in the same process and A was executed before B, then $A \rightarrow B$
  - If A is the event of sending a message by one process and B is the event of receiving that message by another process, then $A \rightarrow B$
  - If $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$

Distributed systems

Distributed synchronisation

Event ordering – The happened-before relation

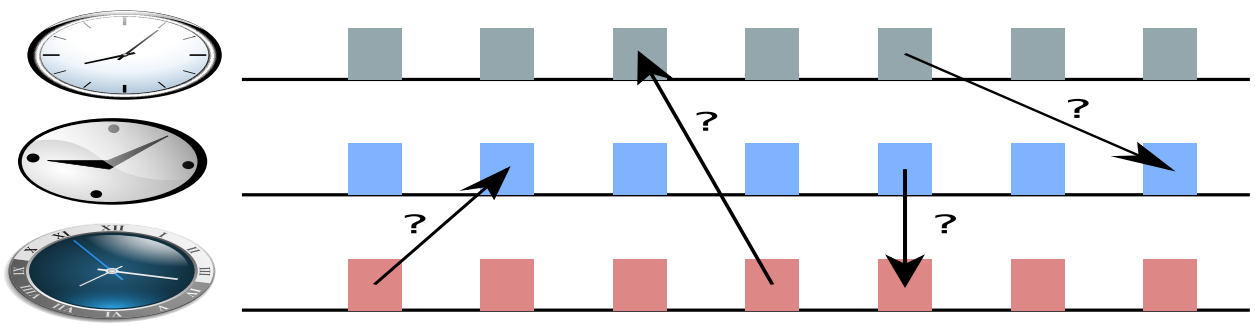
- If two events are not related by the happened before relation, A did not happen before B and B did not happen before A
- These events were executed concurrently
- Neither event can causally affect the other
- If $A \rightarrow B$ then A can affect B

Distributed systems

Distributed synchronisation

Event ordering – The happened-before relation

- We do not know which of the concurrent events happened first.
- Since neither event can affect the other, this is, however, not important
- This is important only for the processes that care about the order of two concurrent events agree on some order

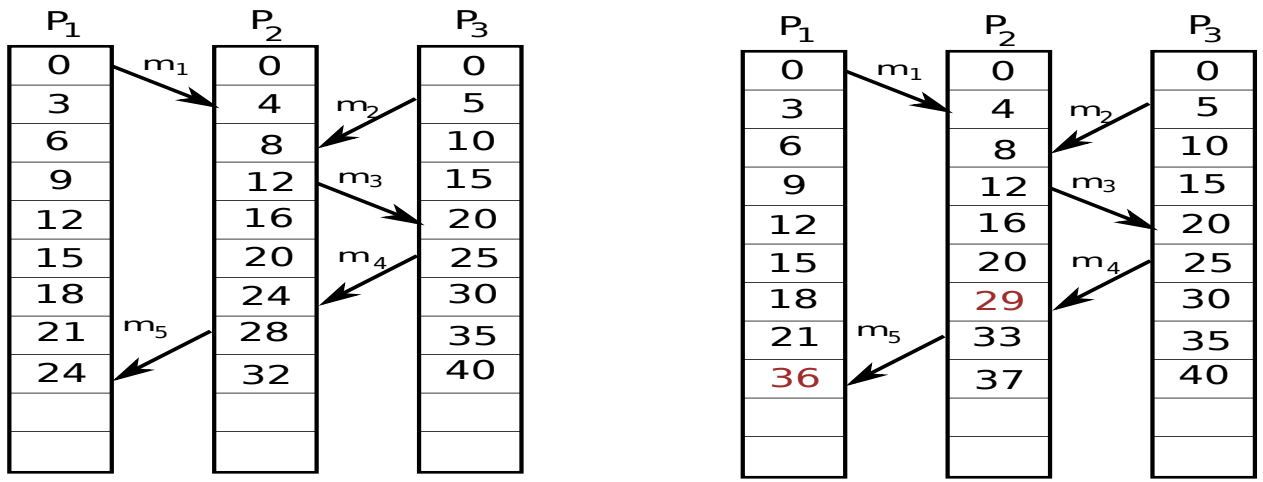


Distributed systems

Distributed synchronisation

Event ordering – Lamport clocks

- To synchronise logical clocks on distributed systems, Lamport used the happened-before relation
- Messages are attached the creation time on their local machine
- Processes synchronised by Lamport clocks alter their clock value whenever an incorrect synchronisation becomes apparent due to message passing

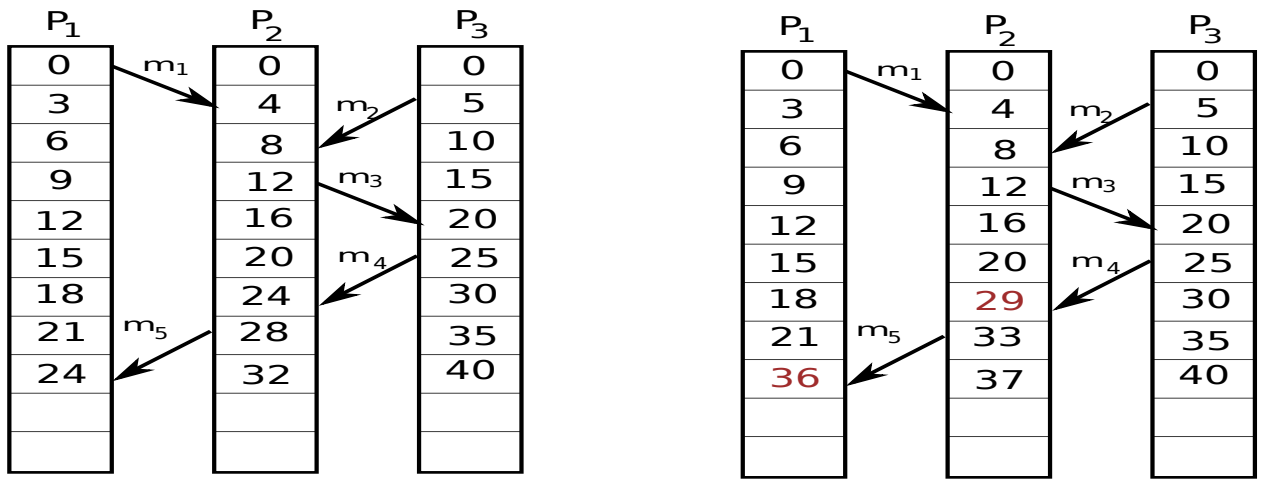


Distributed systems

Distributed synchronisation

Event ordering – Lamport clocks

- With Lamport clocks, a total ordering of processes is possible
- However, nothing can be said about the relationship between two events a and b by merely comparing their time values.
- Example here: m_1 and m_2 – With Lamport clocks can not say, which event happened first

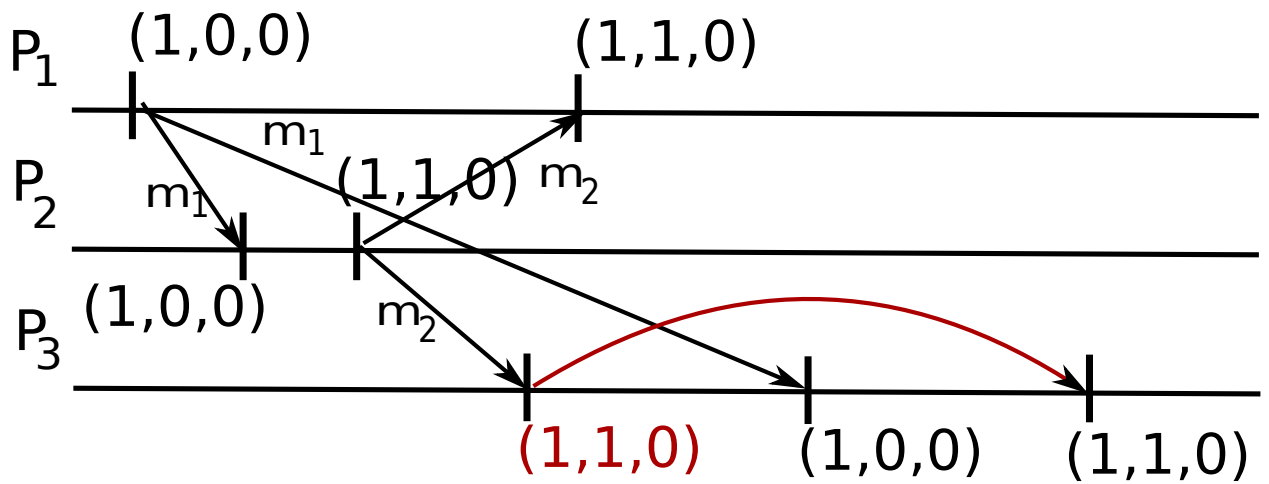


Distributed systems

Distributed synchronisation

Event ordering – Implementation

- Vector clocks are designed to ensure that a message is delivered only if all messages that causally presede it have also been received as well.
- With vector clocks, processes maintain and update an event counter for all events in the system



Distributed systems

Distributed synchronisation

Approaches to implement mutual exclusion

- Assumptions
  - System consists of n processes
  - Each resides at different processor
  - Processes are numbered uniquely from 1 to n
  - Each process has its own processor

Distributed systems

Distributed synchronisation

Approaches to implement mutual exclusion

Centralised approach :

- One of the processes coordinates the entry to the critical section
- Each process that wants to invoke mutual exclusion sends a request message to the coordinator
- The process receives a reply message if request is positively approved
- Process then enters its critical section
- After exiting its critical section, the process sends a release message to the coordinator

Distributed systems

Distributed synchronisation

Approaches to implement mutual exclusion

Fully distributed approach :

- Distribute decision making across entire system
- When process P_i wants to enter critical section
  - Generates a new timestamp TS_i
  - Sends the message $request(P_i, TS_i)$ to all processes in the system
  - On receiving a request message a process may reply immediately or it may defer the reply
  - A process that received a reply message from all other processes enters its critical section
  - After leaving the critical section, process sends a reply message to all its deferred requests

Distributed systems

Distributed synchronisation

Approaches to implement mutual exclusion

Fully distributed approach :

- Positive aspects
  - Mutual exclusion is obtained
  - Freedom from deadlock is ensured
  - Freedom from starvation is ensured
  - The number of messages per critical section entry is $2 \cdot (n - 1)$

Distributed systems

Distributed synchronisation

Approaches to implement mutual exclusion

Fully distributed approach :

- Negative aspects
  - Processes need to know the identity of all other processes in the system
  - If one process fails , the entire scheme collapses
  - Processes that have not entered their critical section must pause frequently, to assure other processes that they intend to enter the critical section

Distributed systems

Distributed synchronisation

Approaches to implement mutual exclusion

Token passing approach :

- Token is circulated among processes
- Only the process with the token is allowed to access the critical section at a time
- Possible failure cases
  - When the token is lost, an election must be called to generate a new token
  - If a process fails a new logical ring must be established

Distributed systems

Distributed synchronisation

Atomicity

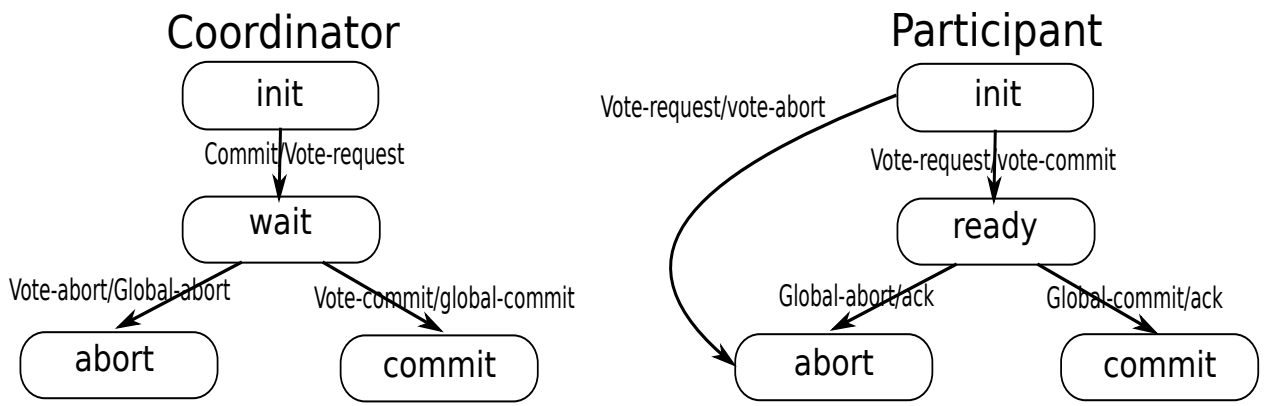
- In distributed systems, also atomic operations are required
- Example: Transaction that has to be executed by either none or all participating nodes
- To ensure atomicity, the following protocols can be implemented
  - Two-phase commit protocol
  - Three-phase commit protocol

Distributed systems

Distributed synchronisation

Atomicity – Two-phase commit

- In 1978, Gray introduced the two-phase commit protocol
- In the protocol, a coordinator asks all participating nodes to commit
- If only one of the nodes does not answer/agree, the commit is aborted
- Otherwise, all nodes commit simultaneously

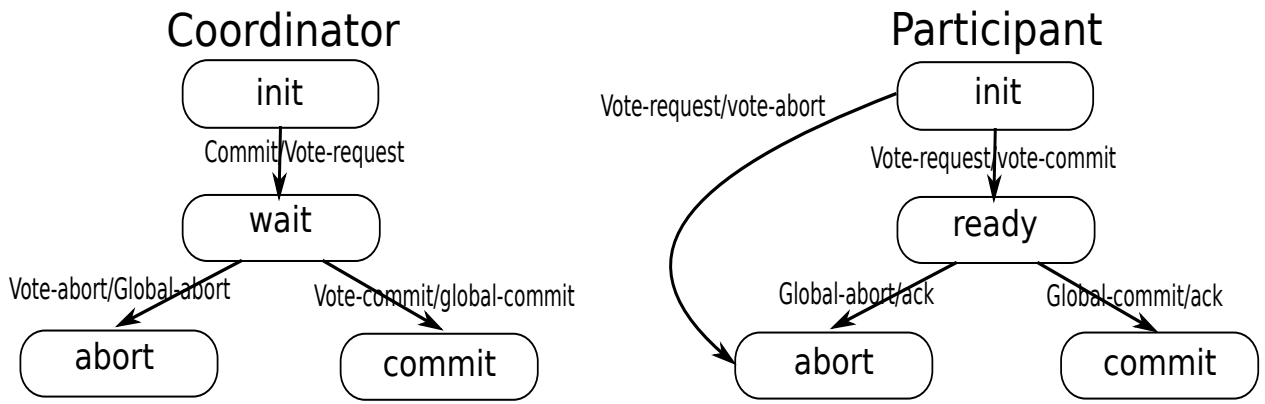


Distributed systems

Distributed synchronisation

Atomicity – Two-phase commit

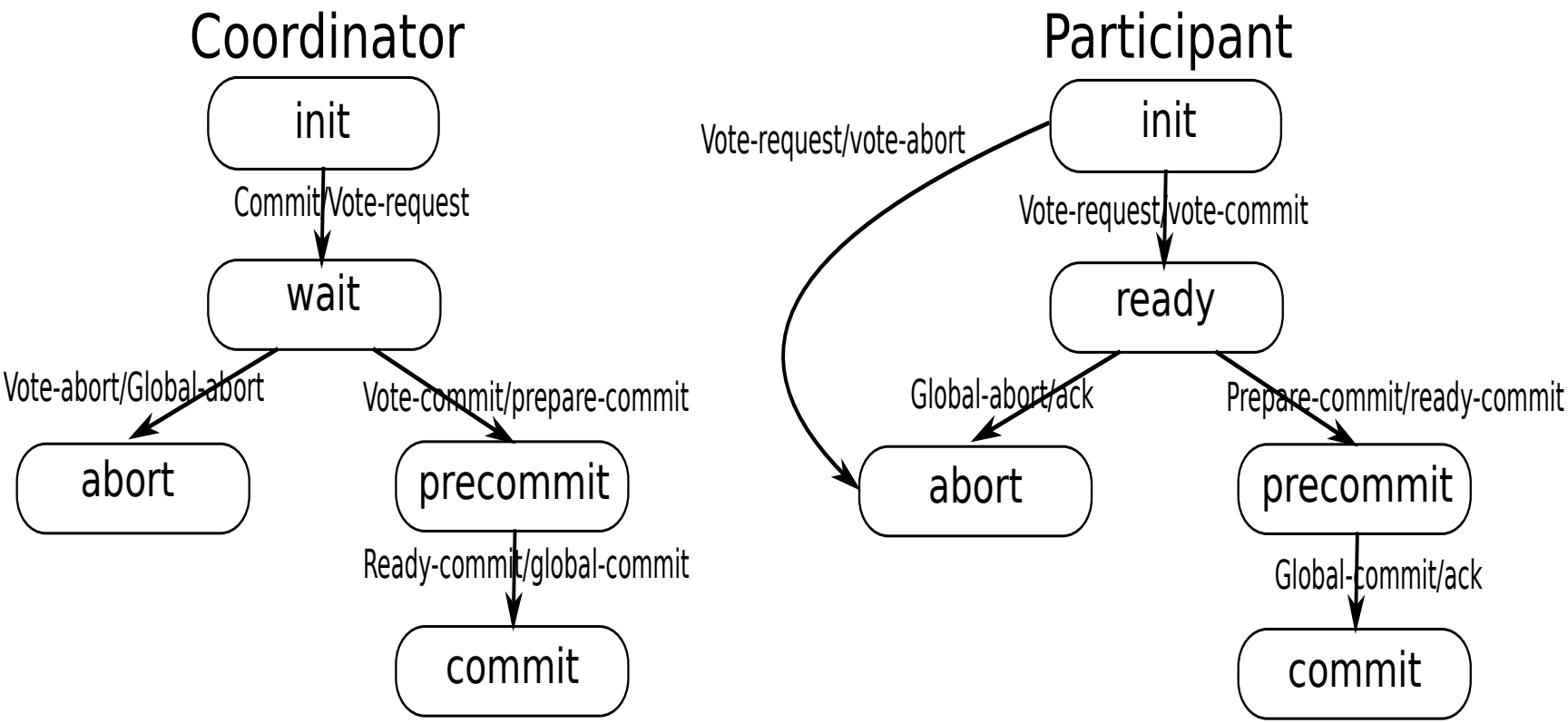
- The two-phase commit protocol has several problems when single nodes fail. This event will frequently result in blocking all participating nodes until the failed node recovers
- When the coordinator has to restart in its Wait state, it might miss some of the answers of the clients



Atomicity – Three-phase commit

- The three-phase commit protocol solves this problem by introducing an intermediate precommit state
- When the coordinator fails in precommit state, the distributed nodes can still take independent decisions
- It is only possible to arrive in precommit, when all nodes already agreed to commit
- Therefore, a node in precommit state can safely commit regardless of the current state of all other nodes

Atomicity – Three-phase commit



Distributed systems

Distributed synchronisation

Deadlock handling

- To handle deadlocks in distributed systems, the same deadlock-prevention and deadlock-avoidance algorithms as for non-distributed system can be applied
- However, some modifications must be applied
  - All resources in the whole distributed system must be assigned unique numbers (for resource ordering)
  - For the bankers algorithm, one process must hold all information necessary to carry out the algorithm

Deadlock handling

- A timestamp based deadlock prevention scheme for distributed systems (preemptive)
  - Each process is assigned a timestamp at creation time
  - Older processes are allowed to pre-empt newer processes

Deadlock handling

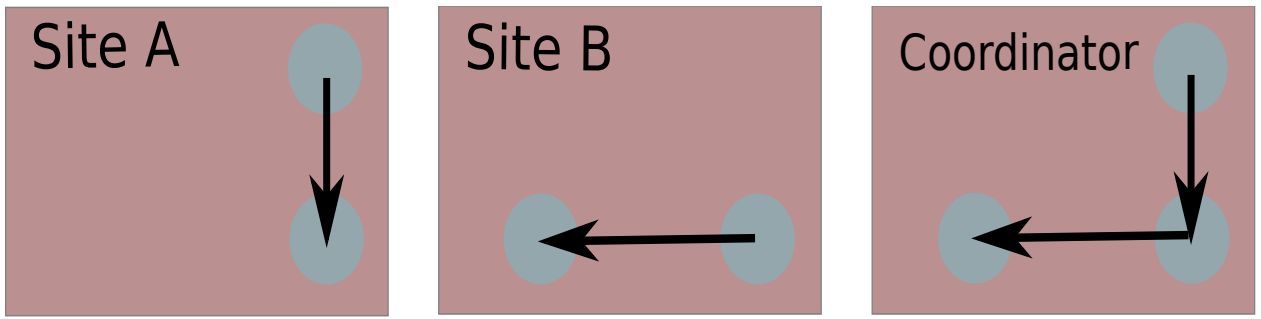
- A timestamp based deadlock prevention scheme for distributed systems (non-preemptive)
  - Each process is assigned a timestamp at creation time
  - Older processes wait for younger processes to finish their tasks
  - Younger processes quit and restart when encountering older processes holding desired resources
  - Processes that quit and restart keep their timestamp

Distributed systems

Distributed synchronisation

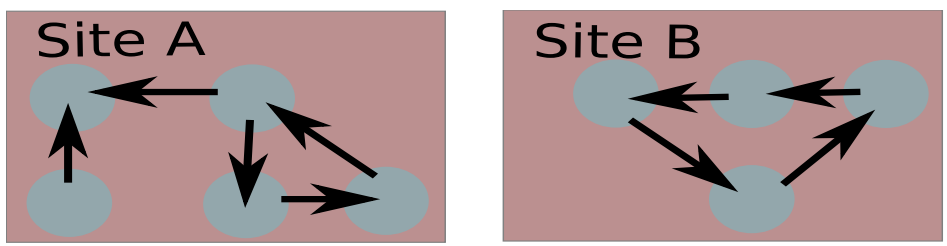
Deadlock detection

- Deadlocks can be detected by creating resource allocation graphs
- The challenge for distributed systems is to decide how to maintain the graph
  - Centralised approach
    - Local resource allocation graphs are merged to a global view
    - Due to delay in the network, the global view may differ temporarily from the actual situation



Deadlock detection – Fully distributed approach

- Fully distributed approach
  - Controllers share equally responsibility for detecting deadlock
  - Every site hosts a resource allocation graph
  - This graph contains an additional node P_{ex}
  - An arc $P_i \rightarrow P_{ex}$ indicates that P_i is waiting for a data item in another site held by any process
  - An arc $P_{ex} \rightarrow P_i$ indicates that any process at another cite waits to acquire P_i
  - Due to delay in the network, the global view may differ temporarily from the actual situation
  - A cycle including P_{ex} does not necessarily mean that the system is in a deadlocked state



Distributed systems

Distributed synchronisation

Election algorithms

- Many distributed algorithms employ a coordinator process
- Especially, the coordinator may fail so that a new coordinator must be elected
- Typically, processes are assigned unique priority numbers
- To choose an appropriate process as the coordinator, several election algorithms can be applied
  - The Bully algorithm
  - The Ring algorithm

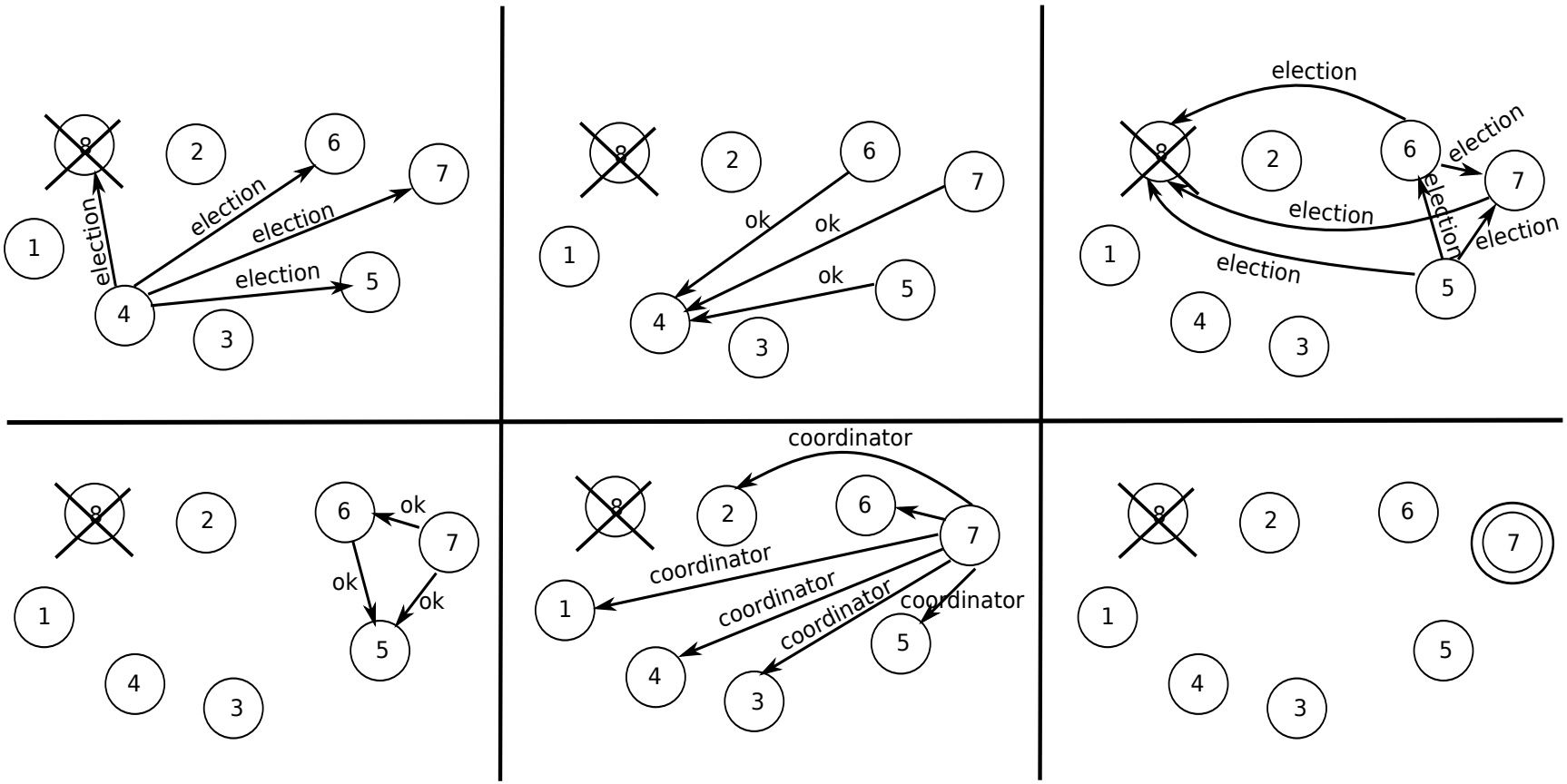
Election algorithms – The Bully algorithm

- A process that notices the absence of a coordinator sends an election message with his priority number to all processes with higher number
- If it does not receive an answer during a time interval T, it becomes the coordinator and informs all active processes
- A process receiving an election message answers only if it has a higher priority (and then starts an election itself)
- When the old coordinator becomes available again it also starts an election

Distributed systems

Distributed synchronisation

Election algorithms – The Bully algorithm



Distributed systems

Distributed synchronisation

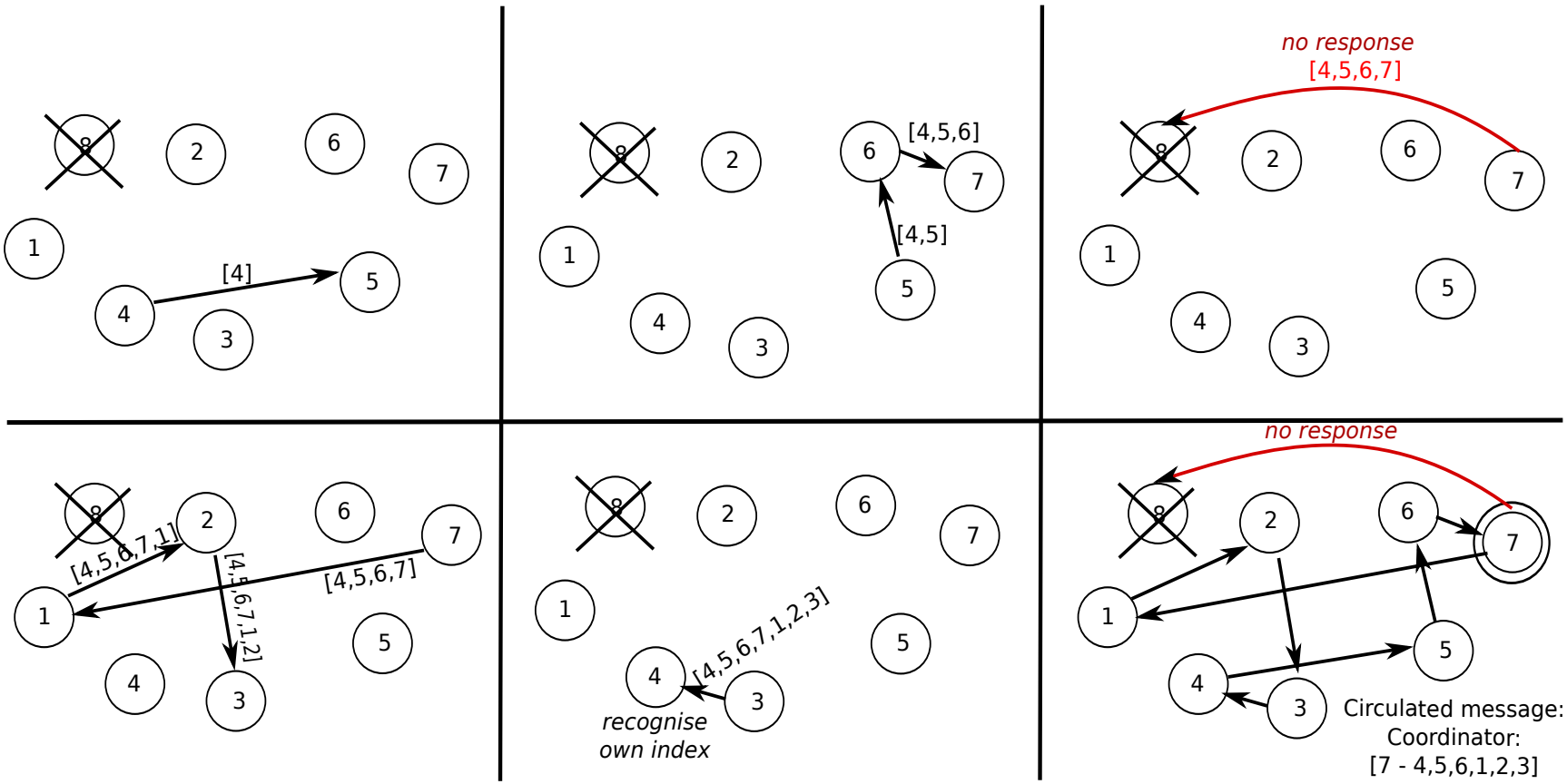
Election algorithms – The Ring algorithm

- The ring algorithm assumes that processes are in an unambiguous order in which messages are sent
- A process that notices the absence of a coordinator creates a list with its priority value as first entry and sends this to its successor
- If the successor is down, the message is sent to the successors successor (and so on)
- A process receiving such a list adds its priority to the end of the list if its own number is not on the list and forwards the list to its successor
- If the own priority is already on the list, the process searches for the process with highest priority
- The message is then circulated a second time as coordinator message to inform all nodes of the new coordinator

Distributed systems

Distributed synchronisation

Election algorithms – The Ring algorithm



Distributed systems

Questions, discussion, remarks

Questions?

Literature

Recommended literature

- A. Tanenbaum, Moderne Betriebssysteme, 2nd edition, Prentice Hall, 2009.
- A. Tanenbaum, Modern operating systems, 3rd edition, Prentice Hall, 2008.
- A. Silberschatz et al. Operating system concepts, Wiley, 2004.
- W. Stallings, Operating systems, 6th edition, Prentice Hall, 2008.