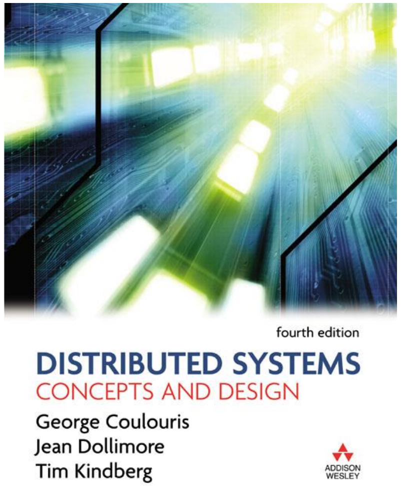


KATHOLIEKE UNIVERSITEIT LEUVEN

Distributed Systems:

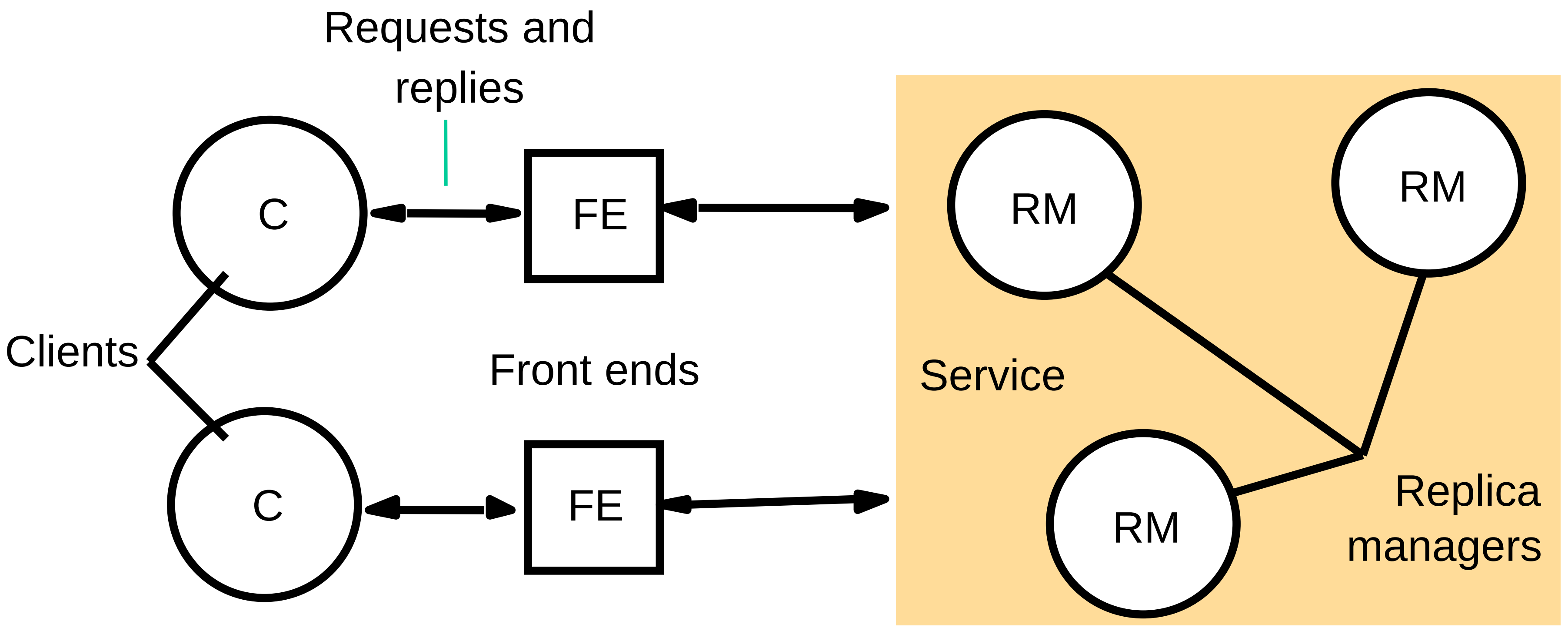
Replicated Data

Overview

- Replication
  - System model and group communication
  - Fault-tolerant services
    - Masking failure
  - Highly available services
    - Maximizing service availability

System model and group communication

- Architectural model



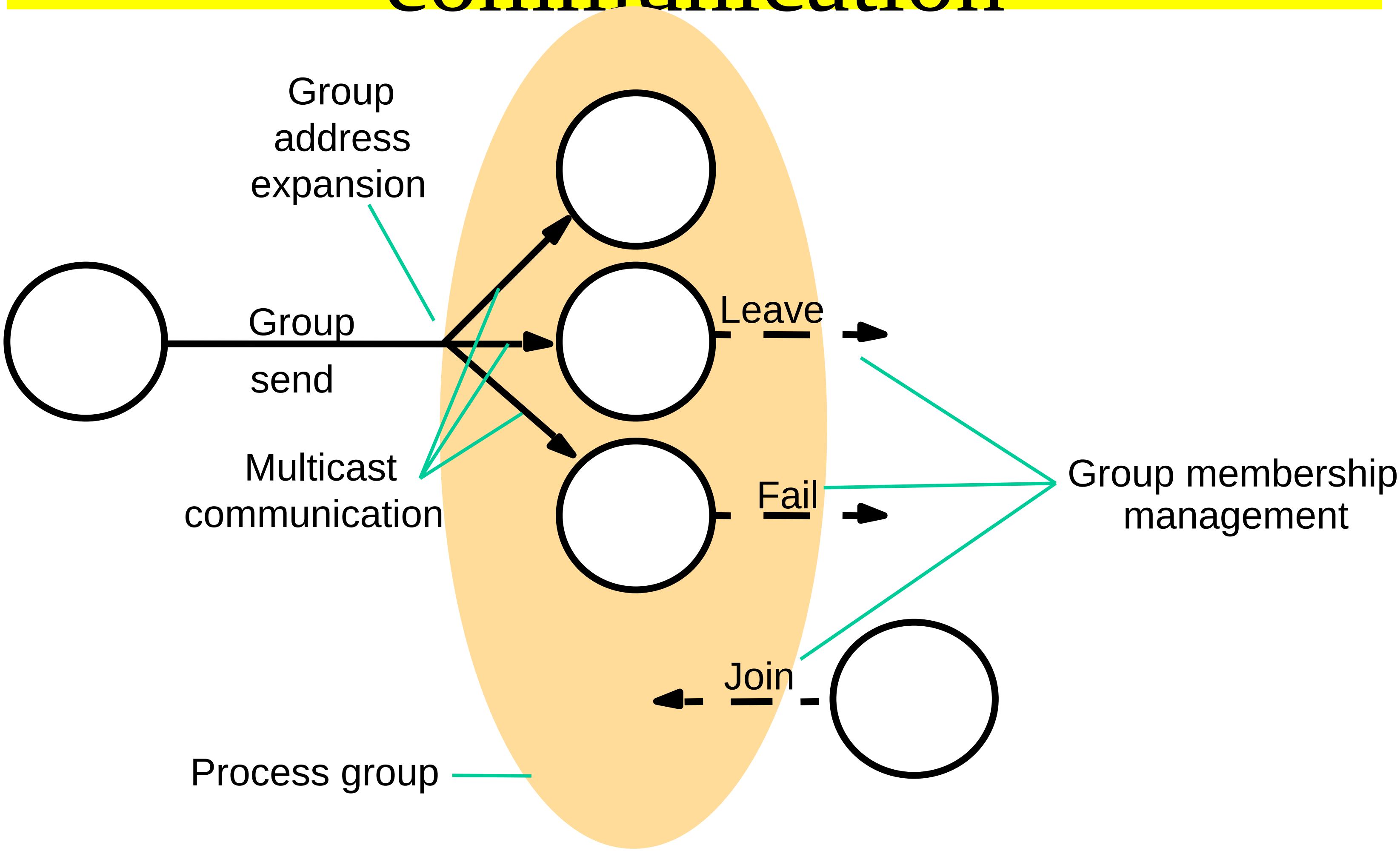
System model and group communication

- 5 phases in the execution of a request:
  - FE issues requests to one or more RMs
  - Coordination: needed to execute requests consistently
    - FIFO
    - Causal
    - Total
  - Execution: by all managers, perhaps tentatively
  - Agreement
  - Response

System model and group communication

- Need for dynamic groups!
- Role of group membership service
  - Interface for group membership changes: create/destroy groups, add process
  - Implementing a failure detector: monitor group members
  - Notifying members of group membership changes
  - Performing group address expansion
- Handling network partitions: group is
  - Reduced: primary-partition
  - Split: partitionable

System model and group communication



System model and group communication

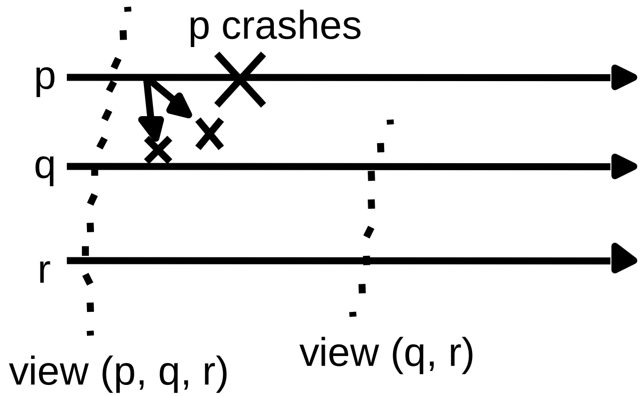
- View delivery
  - To all members when a change in membership occurs
  - <> receive view
- Event occurring in a view v(g) at process p
- Basic requirements for view delivery
  - Order: if process p delivers v(g) and then v(g’) then no process delivers v(g’) before v(g)
  - Integrity: if p delivers v(g) then $p \in v(g)$
  - Non-triviality: if q joins group and remains reachable then eventually $q \in v(g)$ at p

System model and group communication

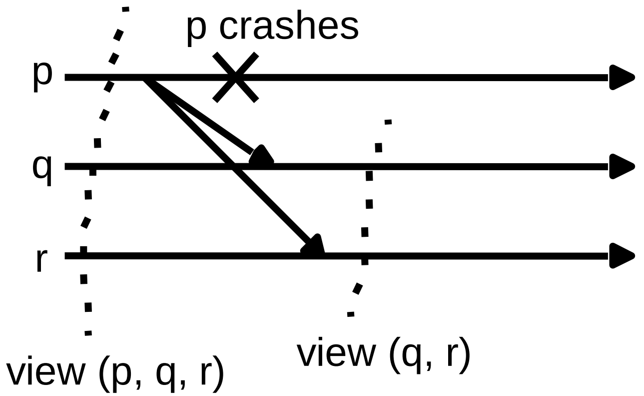
- View-synchronous group communication
  - Reliable multicast + handle changing group views
  - Guarantees
    - Agreement: correct processes deliver the same set of messages in any given view
    - Integrity: if a process delivers m, it will not deliver it again
    - Validity: if the system fails to deliver m to q then other processes will deliver v'(g) (=v(g) –{q}) before delivering m

System model and group communication

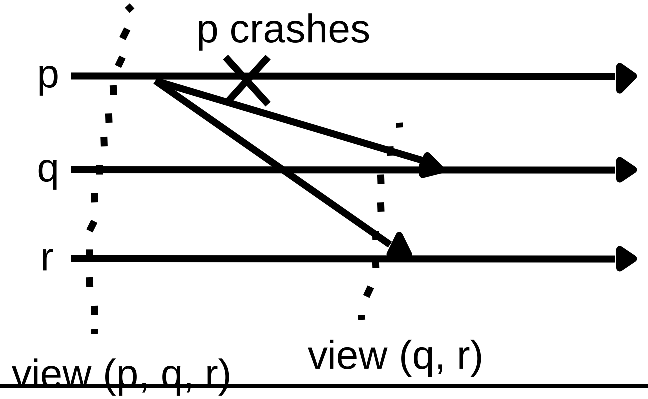
a (allowed).



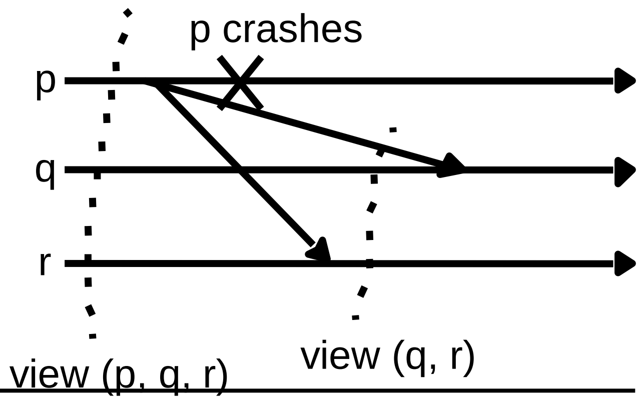
b (allowed).



c (disallowed).



d (disallowed).



Overview

- Replication
  - System model and group communication
  - Fault-tolerant services
  - Highly available services
  - (*Not part of the course 2023-2024*) Transactions with replicated data

Fault-tolerant services

- Basic Goal: provide a service that is correct despite up to f process failures
- Assumptions:
  - Communication reliable
  - No network partitions
- Meaning of correct in case of replication
  - Service keeps responding
  - Clients cannot discover difference with ... (*transparency*)

Fault-tolerant services

- Naiv
  - Cl
    m
  - Cl ilure
  - Re ground
- Example:

Strange behavior:

Client 2 sees 0 on account x and NOT 1

2 on account y

and update of x has been done earlier!!

Client 1	Client 2
$setBalance_B(x,1)$	
$setBalance_A(y,2)$	
	$getBalance_A(y) \rightarrow 2$
	$getBalance_A(x) \rightarrow 0$

Fault-tolerant services

- Correct behaviour? – *Single copy semantics*

- *Two variants:*
  - Linearizability
    - Strong requirement
  - Sequential consistency
    - Weaker requirement

Fault-tolerant services

- Linearizability
  - Terminology:
    - O_{ij}: client i performs operation j
    - Sequence of operations by one client: O_{20}, O_{21}, O_{22},...
    - Virtual interleaving of operations performed by all clients
  - Correctness requirements: ∃ interleaved sequence ...
    - Interleaved sequence of operations meets specification of a (single) copy of the objects
    - Order of operations in the interleaving is consistent with the real times at which the operations occurred
  - Real time?
    - Yes, we prefer up-to-date information
    - Requires clock synchronization: difficult

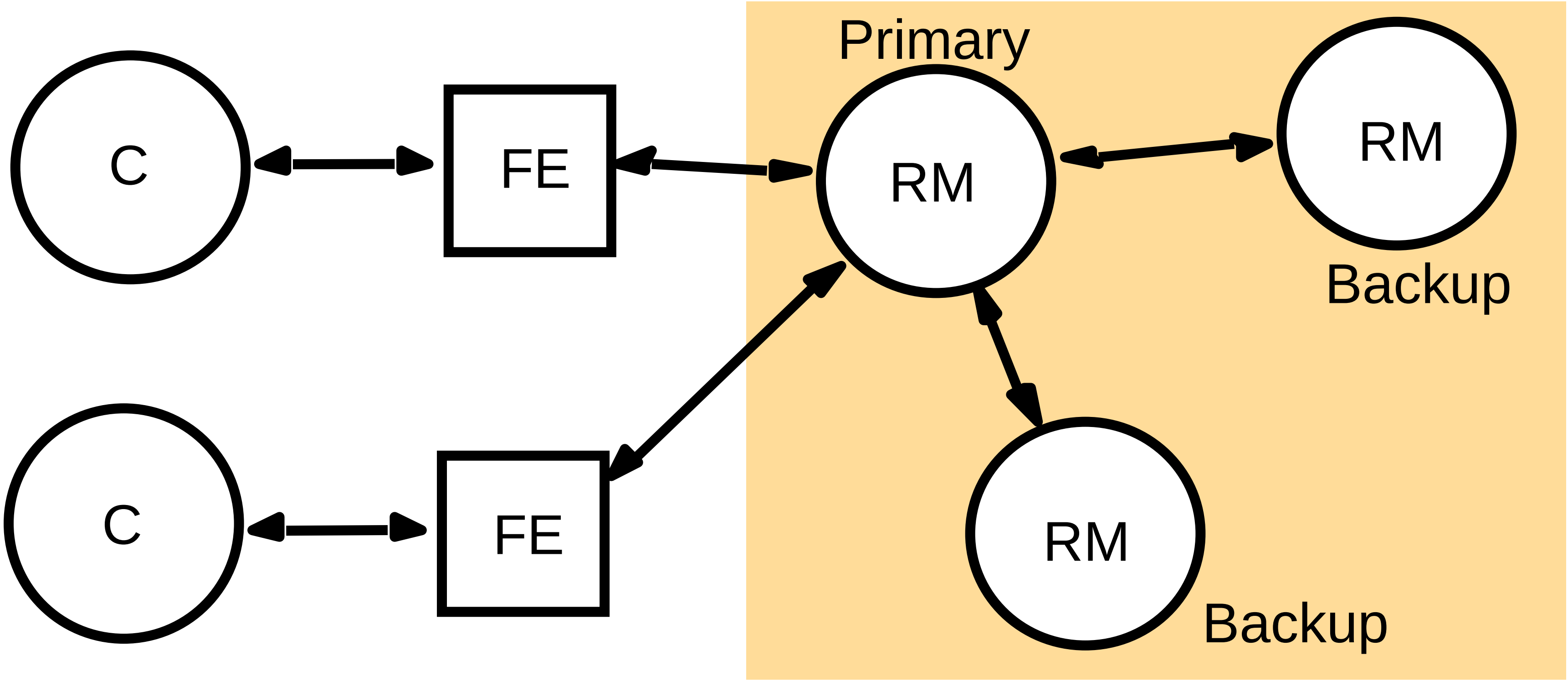
Fault-tolerant services

- Sequential consistency
  - Correctness requirements: ∃ interleaved sequence ... (red = difference!)
    - Interleaved sequence of operations meets specification of a (single) copy of the objects
    - Order of operations in the interleaving is consistent with the program order in which each individual client executed them
  - Example: sequential consistent not linearizable

Client 1	Client 2
$setBalance_B(x,1)$	
	$getBalance_A(y) \rightarrow 0$
	$getBalance_A(x) \rightarrow 0$
$setBalance_A(y,2)$	

Fault-tolerant services

- Passive (primary-backup) replication



Fault-tolerant services

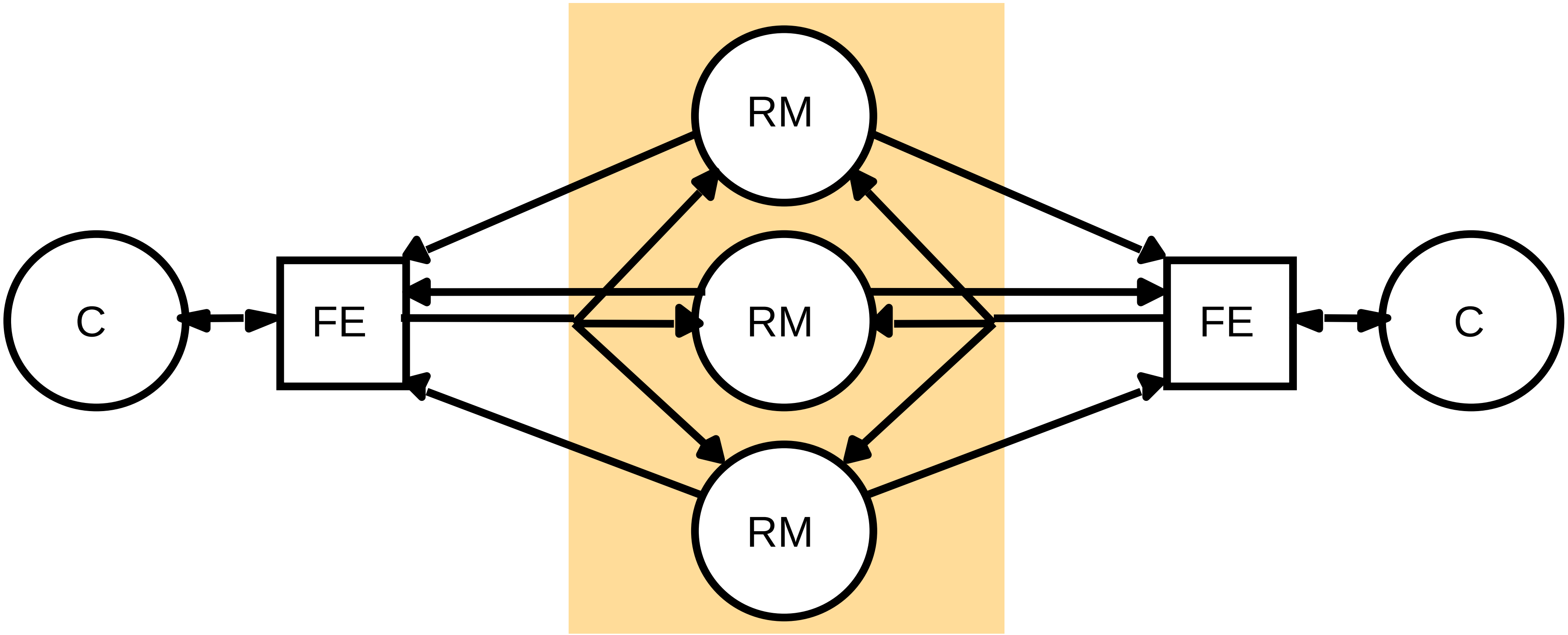
- Passive (primary-backup) replication
  - Sequence of events for handling a client request:
    - Request: FE issues request with unique id to primary
    - Coordination: request handled atomically in order; if request already handled, re-send response
    - Execution: execute request and store response
    - Agreement: primary sends updated state to backups and waits for acks
    - Response: primary responds to FE; FE hands response back to client
  - Correctness: linearizability
  - *Failures*?

Fault-tolerant services

- Passive (primary-backup) replication
  - Failures?
    - Primary uses view-synchronous group communication
    - Linearizability preserved, if
      - Primary replaced by a unique backup
      - Surviving replica managers agree on which operations had been performed at the replacement point
  - Evaluation:
    - Non-deterministic behaviour of primary supported
    - Large overhead: view-synchronous communication required
    - Variation of the model:
      - Read requests handled by backups: linearizability → sequential consistent

Fault-tolerant services

- Active replication



Fault-tolerant services

- Active replication
  - Sequence of events for handling a client request:
    - Request: FE does reliable TO-multicast(g, <m, i>) and waits for reply
    - Coordination: every correct RM gets requests in same order
    - Execution: every correct RM executes the request; all RMs execute all requests in the same order
    - Agreement: not needed
    - Response: every RM returns result to FE; when return result to client?
      - Crash failures: after first response from RM
      - Byzantine failures: after f+1 identical responses from RMs
  - Correctness: sequential consistency, not linearizability

Fault-tolerant services

- Active replication
  - Evaluation
    - Reliable + totally ordered multicast ≡ solving consensus
      - ➔ Synchronous system
      - ➔ Asynchronous + failure detectors
      - – Overhead!
    - More performance
      - – Relax total order in case operations commute: result of $o_1;o_2$ = result $o_2;o_1$
      - – Forward read-only request to a single RM

Overview

- Replication
  - System model and group communication
  - Fault-tolerant services
  - Highly available services

Highly available services

- Goal
  - Provide acceptable level of service
  - Use minimal number of RMs
  - Minimize delay for returning result

  ➔ Weaker consistency ⇔ single-copy semantics
- Overview (text book)
  - Coda
  - Gossip Architecture <*briefly introcuced, mostly SKIPPED 2023-24*>
  - Bayou <*SKIPPED 2023-24*>

Highly available services
Coda

- Aims: constant data availability
  - better performance, e.g. for bulletin boards, databases,…
  - more fault tolerance with increasing scale
  - support mobile and portable computers (disconnected operation)

Approach: AFS + replication

Highly available services
Coda

- Design AFS+
  - file volumes replicated on different servers
  - volume storage group (VSG) per file volume
  - Available Volume Storage group (AVSG) per file volume at a particular instance of time
  - volume disconnected when AVSG is empty; due to
    - network failure, partitioning
    - server failures
    - deliberate disconnection of portable workstation

Highly available services
Coda

- Replication and consistency
  - file version
    - integer number associated with file copy
    - incremented when file is changed
  - Coda version vector (CVV)
    - array of (assumed “version”) numbers stored with file copy on a particular server (holding a volume)
    - one value per volume in VSG

Highly available services
Coda

- Replication and consistency: example 1
  - File F stored at 3 servers: S_1, S_2, S_3
  - Initial values for all CVVs: $CVV_i = [1,1,1]$
  - update by C_1 at S_1 and S_2; S_3 inaccessible

➔ $CVV_1 = [2,2,1]$, $CVV_2 = [2,2,1]$, $CVV_3 = [1,1,1]$

  - network repaired ➔ conflict detected

    file copy at S_3 updated

➔ $CVV_1 = [2,2,2]$, $CVV_2 = [2,2,2]$, $CVV_3 = [2,2,2]$

Highly available services

Coda

- Replication and consistency: example 2
  - File F stored at 3 servers: S_1, S_2, S_3
  - Initial values for all CVVs: $CVV_i = [1,1,1]$
  - update by C_1 at S_1 and S_2; S_3 inaccessible

  ➔ $CVV_1 = [2,2,1]$, $CVV_2 = [2,2,1]$, $CVV_3 = [1,1,1]$
  - update by C_2 at S_3 ; S_1 and S_2 inaccessible

  ➔ $CVV_1 = [2,2,1]$, $CVV_2 = [2,2,1]$, $CVV_3 = [1,1,2]$
  - network repaired ➔ conflict detected
    manual intervention or ….

Highly available services
Coda

- Implementation
  - On open
    - Select one server from AVSG
    - check CCV with all servers in AVSG
    - files in replicated volume remain accessible to a client that can access at least one of the replica
    - load sharing over replicated volumes
  - On close
    - multicast file to AVSG
    - update of CCV
  - manual resolution of conflicts might be necessary

Highly available services
Coda

- Caching: update semantics
  - successful open:

AVSG not empty **and** latest(F, AVSG, 0)

or

AVSG not empty **and** latest(F, AVSG, T) **and** lostcallback(AVSG, T) **and** incache (F)

or

AVSG empty **and** incache (F)

Highly available services
Coda

- Caching: cache coherence
  - relevant events to detect by Venus within T seconds of their occurrence:
    - enlargement of AVSG
    - shrinking of AVSG
    - lost callback event
  - method: probe message to all servers in VSG of any cached file every T seconds

Highly available services
Coda

- Caching: disconnected operation
  - Cache replacement policy: e.g. least-recently used
  - how support long disconnection of portables:
    - Venus can monitor file referencing
    - users can specify a prioritised list of files to retain on local disk
  - reintegration after disconnection
    - priority for files on server
    - client files in conflict are stored on covolumes; client is informed

Highly available services
Coda

- Performance: Coda <> AFS
  - No replication: no significant difference
  - 3-fold replication & load for 5 users:
    load +5%
  - 3-fold replication & load for 50 users
    load + 70% for Coda <> +16% for AFS
    - Difference: replication + tuning?
- Discussion
  - Optimistic approach to achieve high availability
  - Use of semantics free conflict detection
    (except file directories)

Highly available services Gossip

- Goal of Gossip architecture
  - *Framework* for *implementing* highly availanble services
  - Replicate data close to points where groups of clients need it
- Operations:
  - 2 types:
    - Queries: read-only operations
    - Updates: change state (do not read state)
  - FE send operations to any RM

    selection criterium: available + reasonable response time
  - *Guarantees*

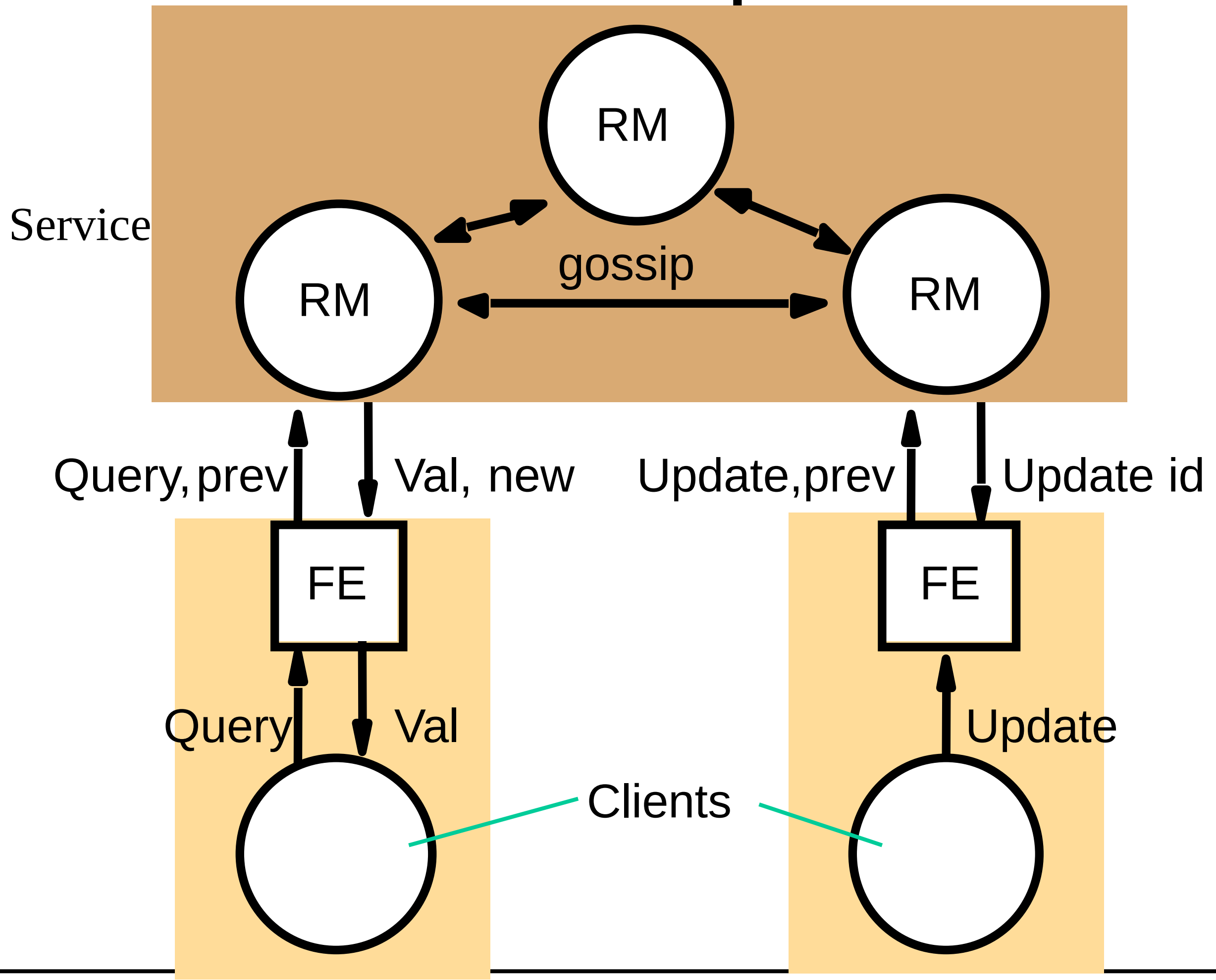
Highly available services
Gossip

- Update ordering:
  - Causal least costly
  - Forced (= total + causal)
  - Immediate
    - Applied in a consistent order relative to any other update at all RMs, independent of order requested for other updates
- Choice
  - Left to application designer!!!
  - Reflects trade-off between consistency and operation cost
  - Implications for users

Highly available services Gossip

- Architecture
  - Clients + FE/client
  - Timestamps added to operations: in next figure
    - Prev: reflects version of latest data values seen by client
    - New: reflects state of responding RM
  - Gossip messages:
    - exchange of operations between RMs

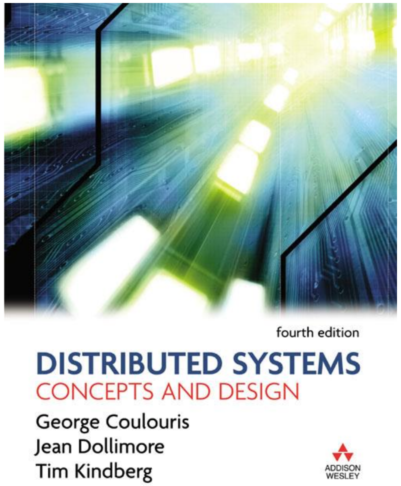
Highly available services
Gossip



Highly available services Gossip

- Discussion of architecture
  + Clients can continue to obtain a service even with network partition
  - Relaxed consistency guarantees
  - Inappropriate for updating replicas in near-real time

- Varying properties dependent on choices made in the framework, and by the application developer
- In any very quite different from active & passive replication!



KATHOLIEKE UNIVERSITEIT LEUVEN

Distributed Systems:

Replicated Data - Part 2

Overview
(recap chapter 16-17-18)

- Transactions

- Distributed transactions

- Replication
  - System model and group communication
  - Fault-tolerant services
  - Highly available services