Today’s Topics - Chapter 15

- Replication 15.1
- Group Communications 15.2
- Fault-tolerant Services 15.3

All figures are in the book “Distributed Systems Concepts and Design” by Couloris, Dollimore and Kindberg

Replication

Motivation: to enhance services.

- Improve its performance.
- Increase its availability.
- Make it fault-tolerant.

Replication - Inhance the performance

- Common: Caching data at clients and servers enhance performance.
- E.g. To share the load, several web servers bind their IP addresses to the same DNS name, say *www.aWebSite.org* and the servers are selected in turn.
- Replication of read-only data is simple, but replication of changing data incurs overheads.

Replication - Increase availability

- Users want that the service always is accessible within resonable response time.
- Suppose there are n servers which each has an independent probability p of failing.
- Availability of an object stored at each of these servers is:

$$1 - p^n$$

- For example, a system of 3 servers each has probability 10% of failure, then availability is $1 - 0.1^3 = 0.999 = 99,9\%$.

Byzantine Faults

Two types of failures:

- **Failure by omission:** The system stops working, it fails to provide some service. You know that the system does not work because it isn't responding.
- **Byzantine Failure:** The system starts producing incorrect output. It is not always easy to distinguish between the system failing and it correctly running.

Fault-tolerant service

- We want services to behave correctly.
  - Highly available data is not necessarily strictly correct data.
  - Data may be out of date; network partitions may induce conflicts that need to be resolved.
- A faul-tolerant service always guarantees correct behaviour despite a number and type of faults.
  - Correctness concerns the freshness of data and the effects of the client's operations upon data.

Fault-tolerant service (cont.)

- The same technique used for high availability - replicating data and functionality - is also used to achieve fault tolerance.
  - If f of f+1 servers crash then 1 remains to supply the service.
  - If f of 2f+1 servers have Byzantine faults then they can supply a correct service.

Requirements for Replicated Data

- Replication transparency clients see *logical objects* (not several physical copies).
  - Clients access one logical item and receive a single result.
- Consistency specified to suit the application.
  - Consisntency - the operations performed upon a collection of replicated objects produce results that meet the specification of correctness for those objects.
  - E.g. when a user of a diary disconnects, their local copy may be inconsistent with the others and will need to be reconciled when they connect again. But connected clients using different copies should get consistent results.

System Model

- Each logical object is implemented by a collection of physical copies called *replicas*.
  - The replicas are not necessarily consistent all the time.
- We assume an:
  - asynchronous system,
  - processes fail only by crashing,
  - network partitions may not occur.

System Model - Replica Managers

- Replicas are held by distinct RM. RM contains the replicas on a given computer and perform operations on them directly.
- RMs apply operations to replicas recoverably i.e. they do not leave inconsistent results if they crash.
- Objects are copied at all RMs unless we state otherwise.
- Static systems are based on a fixed set of RMs. In a dynamic system RMs may join or leave (e.g. when they crash).

State Machine Approach

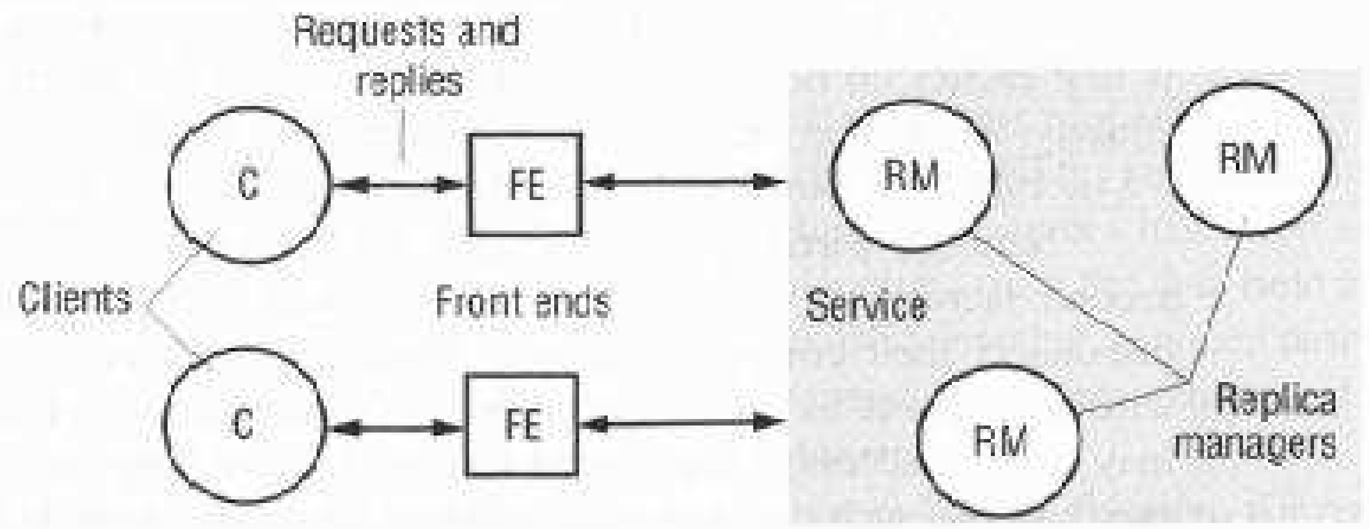
A RM can be a state machine with the following properties:

- Its state is a deterministic function of its initial state and the operations applied.
- All replicas start identical and carry out the same operations.
- Its operations must not be affected by clock readings etc.

Basic Architectural Model

- Clients see a service that gives them access to *logical objects*, which are in fact replicated at the RMs
- Clients request operations: those without updates are called *read-only requests* the others are called *update requests*
- Clients request are handled by *front ends*. A front end makes replication transparent.

A basic architectural model for the management of replicated data



Five Phases in performing a request

- Issue Request: The Front End either:
  - sends the request to a single RM which passes it on to all the others.
  - Multicasts the message to all RM (in the state machine approach).
- Coordination: The RM apply the request; and decide on its ordering relative to other request decide whether to apply the request. (according to FIFO, causal or total ordering)
- Execution: The RMs execute the request (often tentatively).
- Agreement: The RMs *agree* on the effect of the request.

Five Phases Continued

- Response: One or more RMs reply to the FE for:
  - for *high availability* the fastest response is delivered.
  - to tolerate Byzantine faults, take a vote.

Ordering

- Fifo Ordering: If a front end issues request r and then request r' then any correct RM that handles r' handles r before it.
- Causal Ordering: If the issue of request$r$ happend-before the issue of request r', then any correct RM that handles r' handles$r$ before it.
- Total Ordering: If a correct RM handles r before r', then any correct RM that handles r' handles r before it.

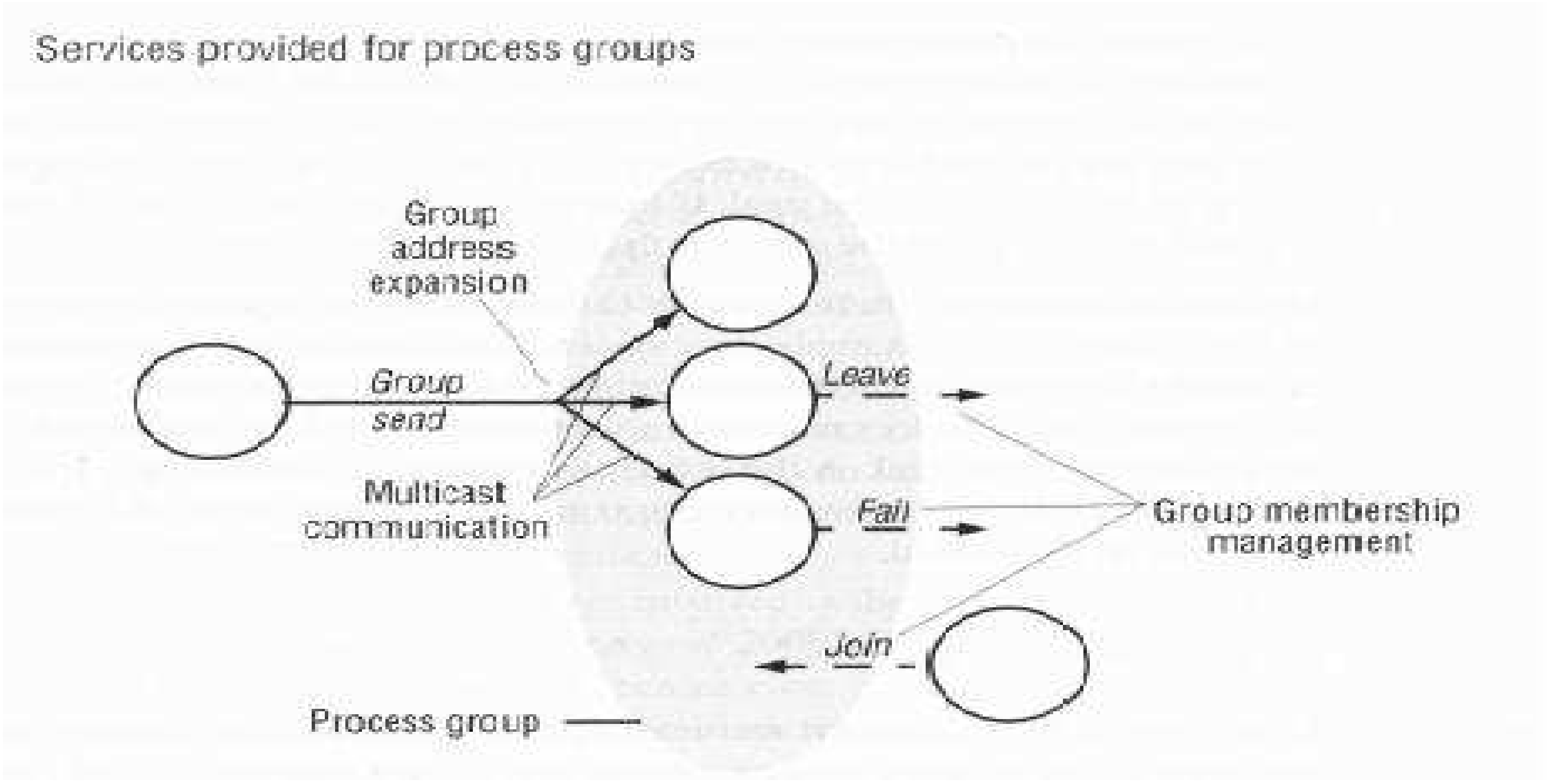
Total Order is too strong. Causal Ordering is desirable, FIFO ordering often implemented.

Group Communication

- The basic idea is that we have a group of processes which participate in the replica.
- If the processes are fixed and no process fails then there is no problem.
- But if we have a number of processes that can join/leave or fail we have to keep track of who belongs to the group.
- The problem is made more complicated, because the might be messages in transit while processes join or leave.

Role of a group membership service

- Provide an interface for group membership changes.
- Implementation of a failure detector.
- Notifying members of group membership changes: The services notifies the group's members when a process is added, or when a process is excluded.
- Performing group address expansion.


Services provided for process groups
Group address expansion
Group send
Multicast communication
Leave
Fail
Join
Group membership management
Process group

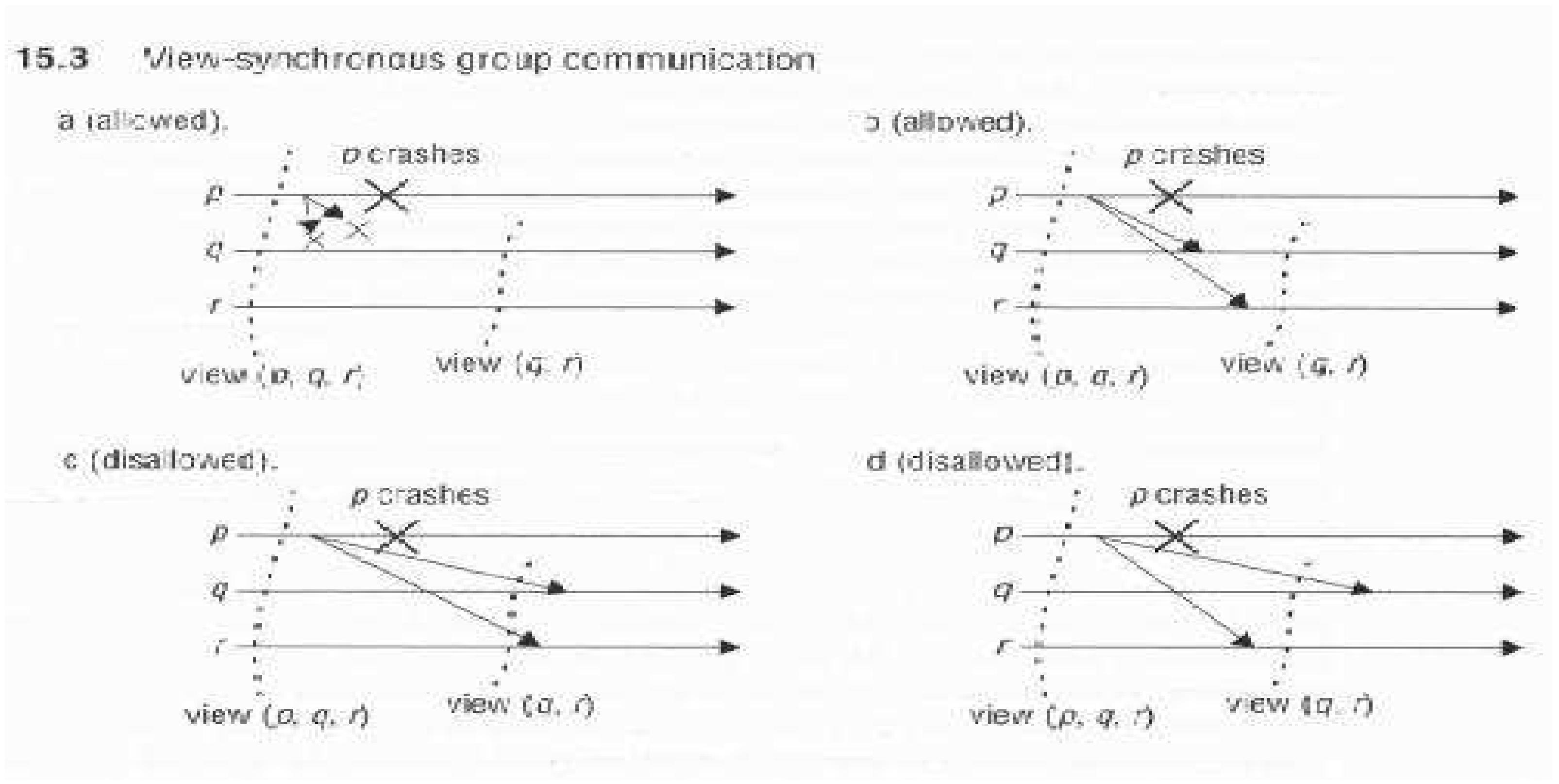
View Delivery

- One way of managing all this is with the idea of a *view*.
- A full group membership service maintains *group views*, which are lists of the current group members.
- The group membership management delivers a series of views to each process.
- We require some consistency requirements with delivery ordering of view notifications w.r.t. the delivery of multicast messages.

View Synchronous group Communication

- *Agreement:* In any given view, correct processes deliver the same set of messages.
- *Integrity:* If a correct process delivers a message, then it it will not deliver that message again.
- *Validity:* Correct process always deliver the messages that they send. If the system fails to deliver a message to any process q, then in the next view q will not be there.

It is essentially a consistency requirement that messages delivered from certain views arrive all before or all after a view change.


15.3 View-synchronous group communication
a (allowed).
p crashes
p
q
r
view (p, q, r)
view (q, r)
b (allowed).
p crashes
p
q
r
view (p, q, r)
view (q, r)
c (disallowed).
p crashes
p
q
r
view (p, q, r)
view (q, r)
d (disallowed).
p crashes
p
q
r
view (p, q, r)
view (q, r)

•

Fault-tolerant Services

- If data is distributed and faults can occur some care has to be taken so that things don't get inconsistent.
- A system is correct if a user can see no difference between one copy and multiple copies.

Bank Account Example

- Consider a naive replication system, in which two RMs at computers A and B each maintain replicas of two bank accounts x and y.
- Clients read and update the accounts at their local RM and the other one in case of failure.
- Replica managers propagate updates to one another in the background after responding to each client.

Bank Account Example

- Client 1 updates the balance of x at its local replica manager B to be 1 Euro and then attempts to update y's balance to be 2 but discovers that B has failed, so Client 1 updates it A instead.
- But Client 2 reads the balance of y to be 2 at A but since B crashed the setting the balance of x did not get through.

Bank Account Example

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

Consistency

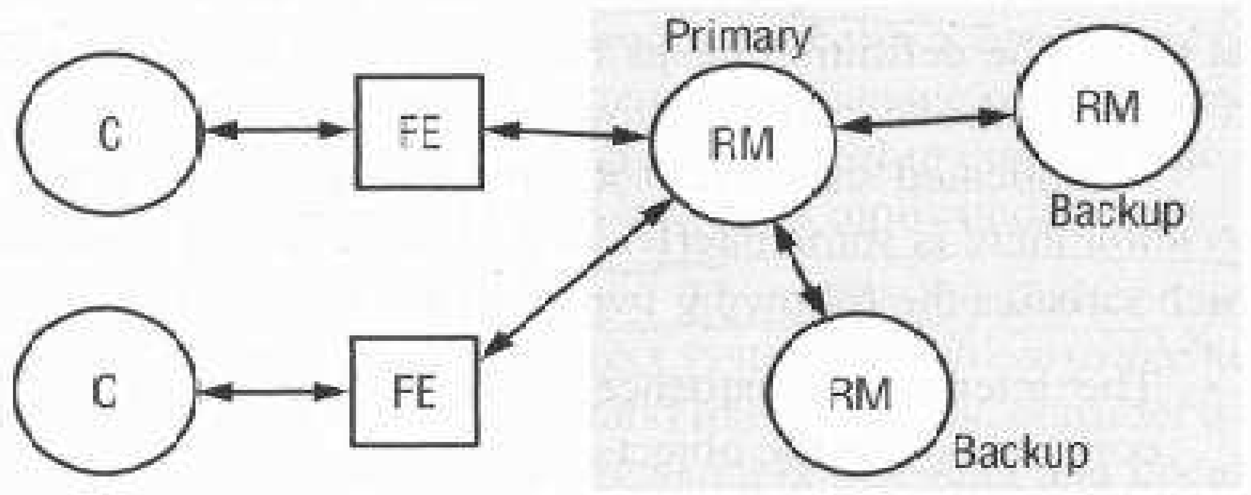
Basic idea.

- We would like some sort of temporal consistency, if s happens before t then on all copies s happens before t. But in the presence of network delays this is not possible.
- So various weaker notions of consistency are introduced.
- One common criterion is sequential consistency. A sequence of operations all allowed there is an interleaving of the individual sequences that produces that interleaving.

Passive Replication for fault tolerance

- Passive model. Single replica manager acts as a primary and one or more as secondary replica managers - backups.
- Front ends communicate only with the primary replica manager.
- The primary replica mangager executes the operations and sends copies of updated data to backups.
- If primary replica manager fails, one of the backups is promoted to primary.

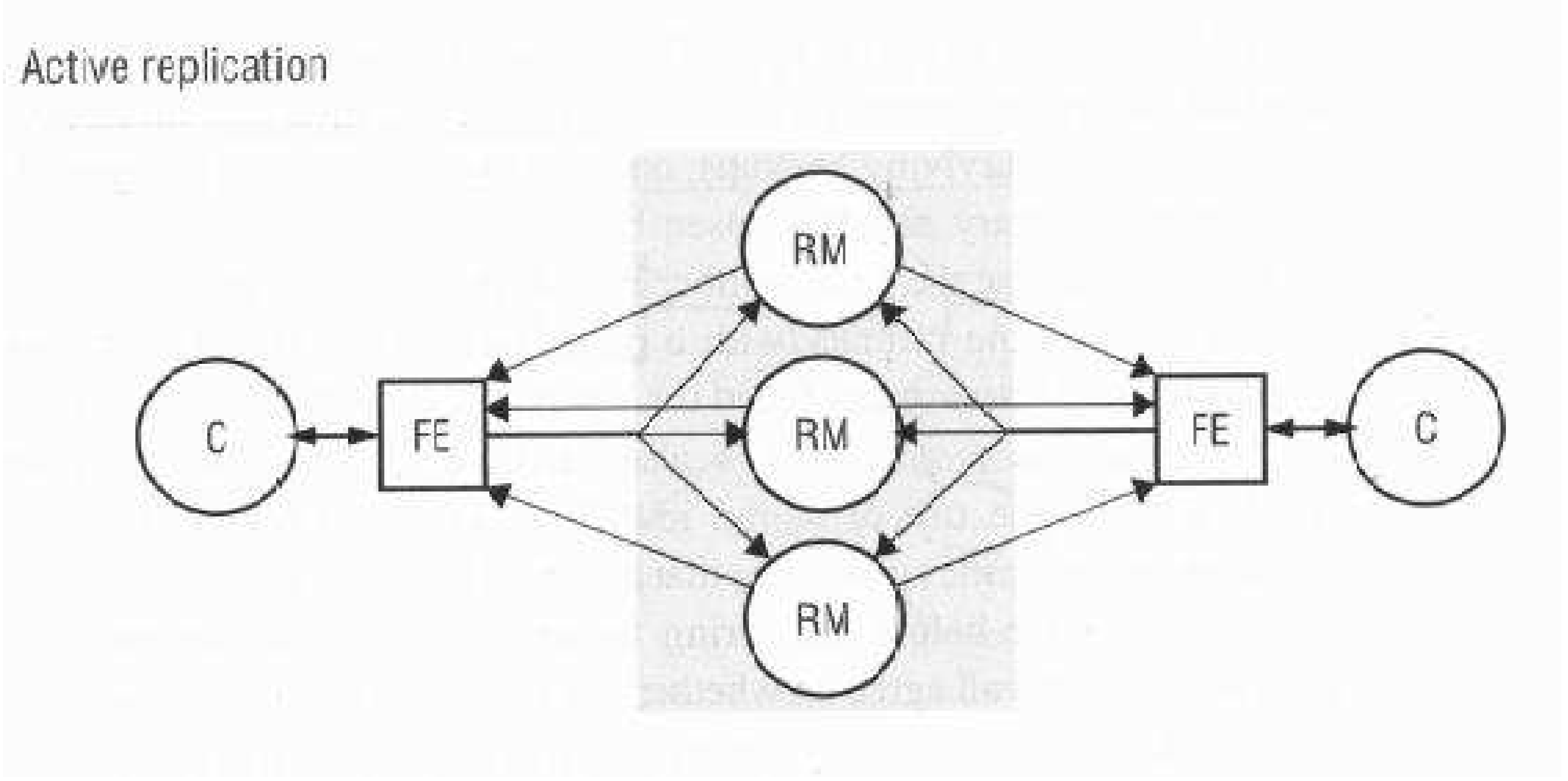
The passive (primary-backup) model for fault tolerance



.

Active Replication for fault tolerance

- Active model. The replica managers are state machines that play equivalent roles.
- Front end sends the same message to every replica manager in the group.
- All replica managers process the request identically and reply.
- If a replica manager fails the other ones still respond in the normal way.


Active replication
RM
RM
RM
C
FE
FE
C