ECEN 757
Replication

Chapter 18

Server-side Focus

- Concurrency Control = how to coordinate multiple concurrent clients executing operations (or transactions) on an object
- P2P = how to store and locate one object in multiple servers

Next:

- Replication Control = how to handle operations (or transactions) when there are objects are stored at multiple servers, with or without replication

Replication: What and Why

- Replication = An object has identical copies, each maintained by a separate server
  - Copies are called "replicas"
- Why replication?
  - Fault-tolerance: With *k* replicas of each object, can tolerate failure of any (*k-1*) servers in the system
  - Load balancing: Spread read/write operations out over the *k* replicas => load lowered by a factor of *k* compared to a single replica
  - Replication => Higher Availability

Availability

- If each server is down a fraction f of the time
  - Server's failure probability
- With no replication, availability of object =

  = Probability that single copy is up

  = $(1 - f)$
- With k replicas, availability of object =

  Probability that at least one replicas is up

  = 1 – Probability that all replicas are down

  = $(1 - f^k)$

Nines Availability

- With no replication, availability of object =
  $= (1 - f)$
- With k replicas, availability of object =
  $= (1 - f^k)$

Availability Table

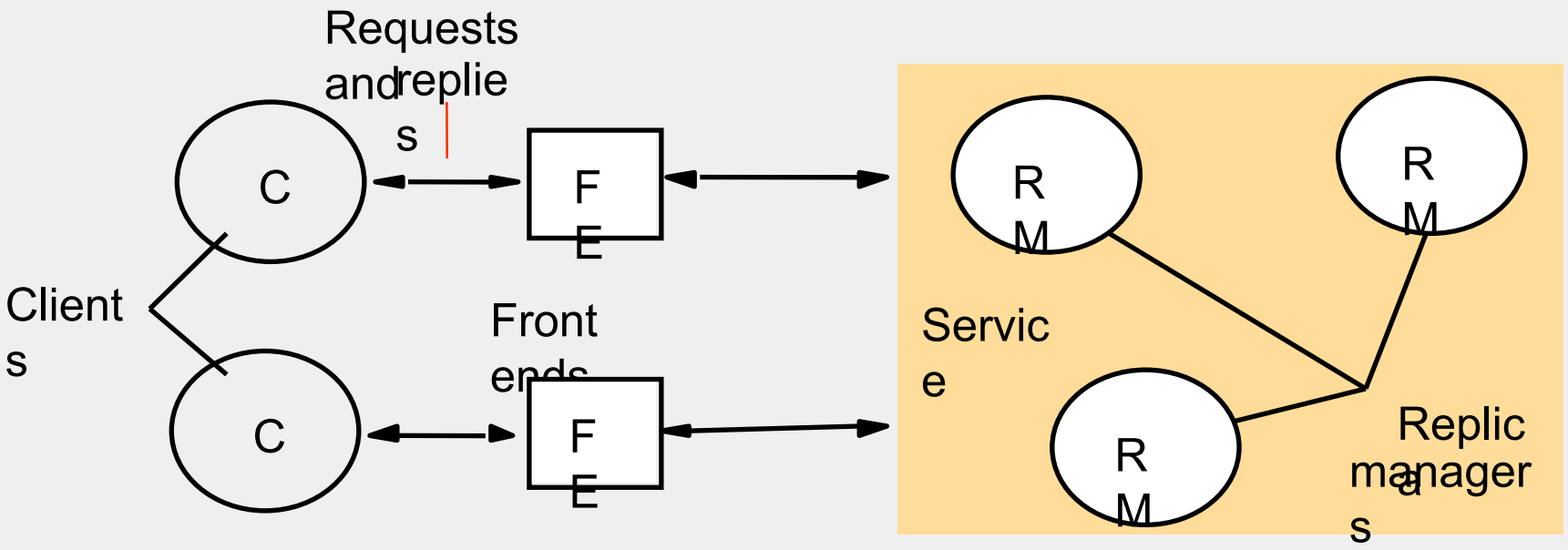
f=failure probability	No replication	k=3 replicas	k=5 replicas
0.1	90%	99.9%	99.999%
0.05	95%	99.9875%	7 Nines
0.01	99%	99.9999%	10 Nines

What's the Catch?

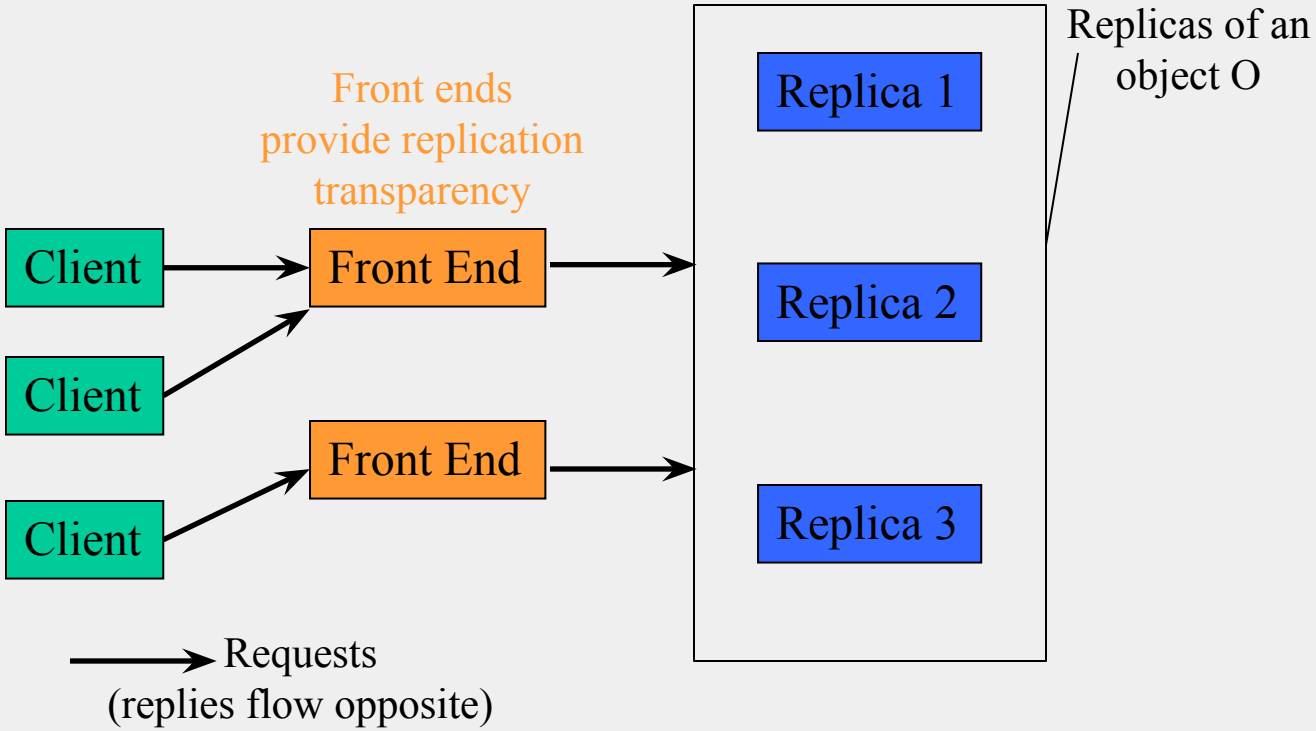
- Challenge is to maintain two properties

1. Replication Transparency
   - A client ought not to be aware of multiple copies of objects existing on the server side
2. Replication Consistency
   - All clients see single consistent copy of data, in spite of replication
   - For transactions, guarantee ACID

System Model



Replication Transparency



How to Handle a Request?

- Five stages are involved in one request:
- Request: The front end issues the request to one or more RM
- Coordination: RMs coordinate in preparation for executing the request. They need to agree on whether to execute the request or not. They also need to agree on the ordering of all requests

Ordering of Requests

- FIFO Ordering: If a front end issues request A and then request B, any correct RM that handles B handles A before it
- Causal Ordering: If the issue of request A happens-before the issue of request B, then any correct RM that handles B handles A before it
- Total Ordering: If a correct RM handles A before B, then any correct RM that handles B handles A before it

How to Handle a Request?

- Five stages are involved in one request:
- Executing: Execute the request, perhaps tentatively in case the transaction aborts later
- Agreement: RMs reach consensus on the effect of the request that will be committed
- Response: One or more RMs respond to the front end

Failures and Group Membership Service

- RMs can only fail by crash
- A fault tolerant system needs to adapt as RMs join, leave, and crash
- A group management service maintains group views, which are lists of the current correct RMs
- A new group view is generated whenever there is a new join/leave/crash
- Group management service is a “protocol,” not a server
  - RMs may coordinate among themselves to maintain group views
  - By using, for example, failure detector+consensus

View Delivery

- The group management service delivers to any RM a series of views $v_0, v_1, v_2, \ldots$
- Ex. $v_0 = \{p\}, v_1 = \{p, q\}, v_2 = \{p\}$
- This means: p joins an empty group, then q joins, and then q leaves/crashes
- Note the difference between deliver and receive
- When a machine receives a view, it may not deliver it to the application immediately
  - For the sake of agreement/ordering/etc.

Basic Requirements of View Delivery

- Order: If a process delivers view v1 and then view v2, then no other process delivers v2 before v1
- Integrity: If process p delivers view v, then p is in v
- Non-triviality: If q and p are indefinitely reachable to each other, then eventually q is in the views that p delivers
- Note: Network partition is possible

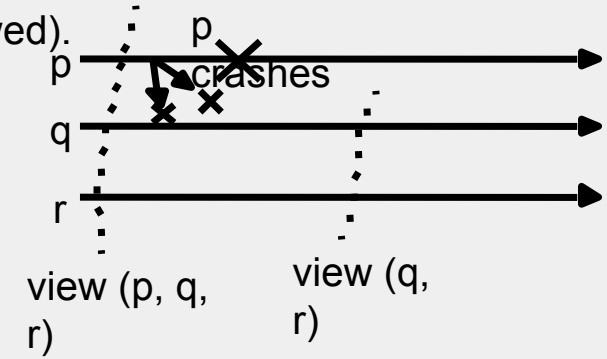
View-synchronous Group Communication

- Some additional requirements about multicast:
- Agreement: Correct processes deliver the same sequence of views, and the same set of messages in any given view
- Integrity: If a correct process delivers message m, it will not deliver m again
- Validity: Correct processes always deliver the messages that they send

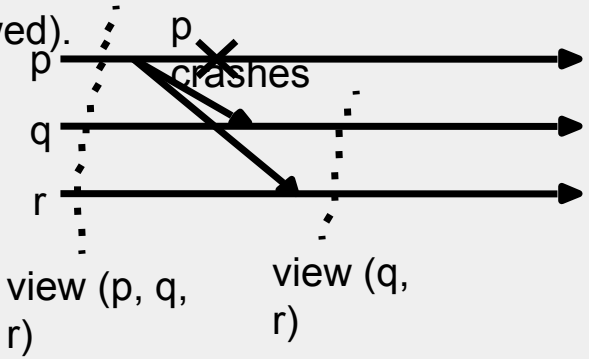
- We assume we use a multicast protocol that achieves these requirements

Some Examples

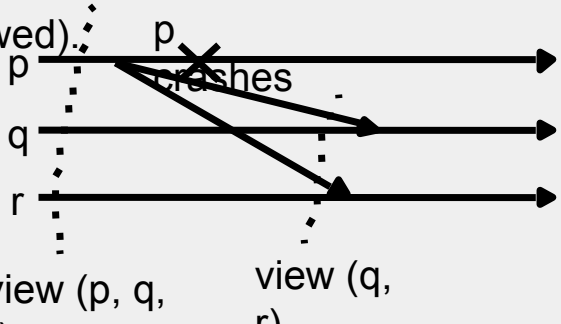
a
(allowed).



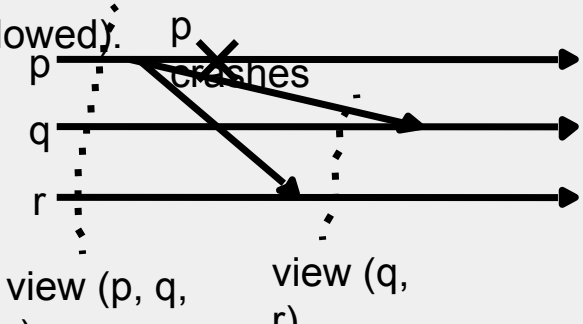
b
(allowed).



c
(disallowed).



d
(disallowed).



Fault-Tolerant Services

- Server B fails after the first step
- Transaction T then performs the second operation on A

Transaction *T*:	**Transaction *U*:**
setBalanceB(x,1)	
setBalanceA(y,2)	
	getBalanceA(y) (= 2)
	getBalanceA(x) (= 0, because B fails to update the value of x to A)

Linearizability

- Each client j has a series of requests: (j,0), (j,1),…
- Without replication, all requests are handled by one server
- The server sees a sequence such as (2,0), (2,1), (1,0), (2,2),…
- Linearizability: For any execution, there exists a sequence such that
- 1. The sequence is possible when there is only one server
- 2. The order of operations in this sequence is consistent with the real times at which the operations occurred in the actual execution

Linearizability != Serial Equivalence

- The following is ok for linearizability
- We still need concurrency control to maintain serial equivalence

Transaction *T*:	**Transaction *U*:**
setBalanceB(x,1)	
	getBalanceA(y) = 0
	getBalanceA(x) = 1
setBalanceA(y,2)	

Sequential Consistent

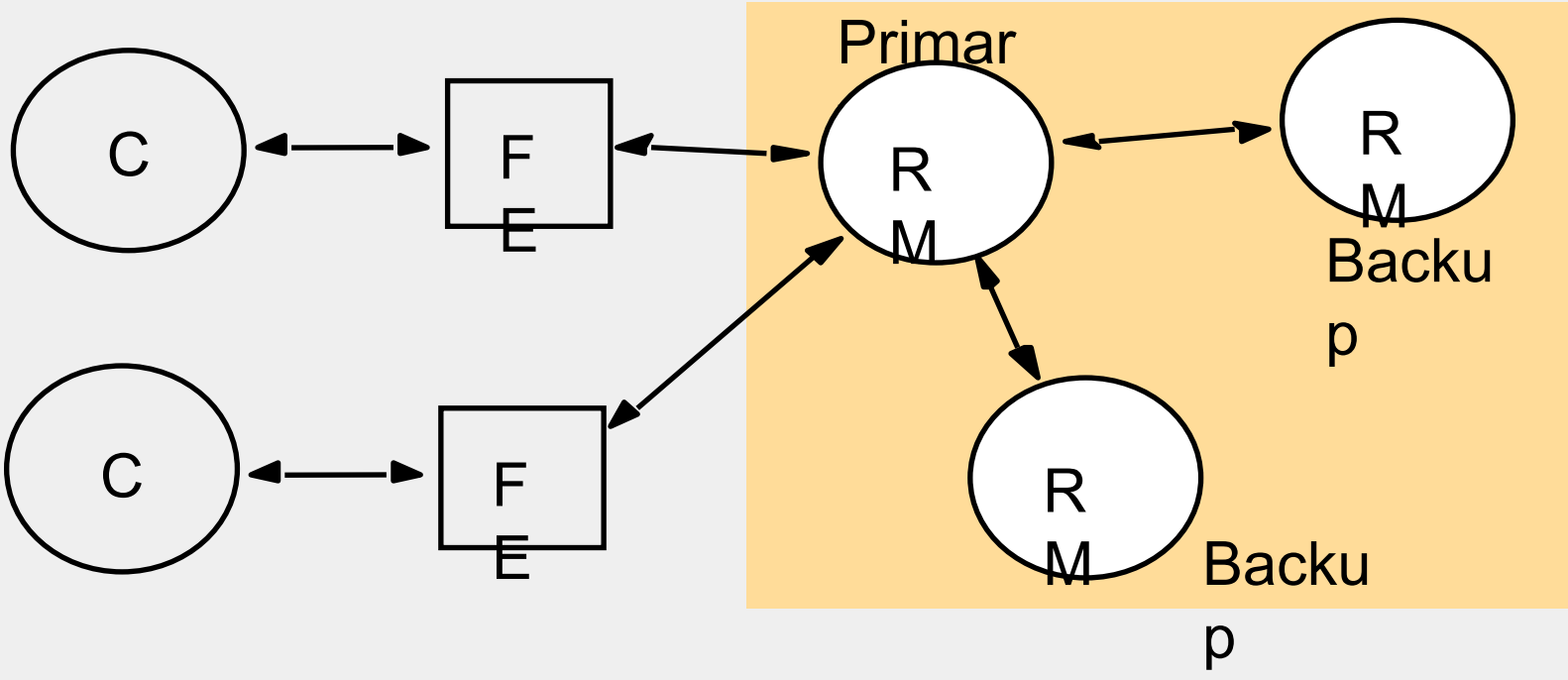
- Linearizability: For any execution, there exists a sequence such that
- 2. … real times…
- “Real times” is hard to guarantee
- Sequential Consistency: For any execution, there exists a sequence such that
- 1. The sequence is possible when there is only one server
- 2. The order of operations in this sequence is consistent with the program order in which each individual client executed them
  - Order between different clients doesn’t matter

Two Types of Replications

- Passive v.s. Active

Passive Replication

- Elect a primary backup



Passive Replication

- Request: The front end issues the request to the primary RM
- Coordination: The primary RM checks the request and determines whether to execute it
- Execution: The primary RM executes the request
- Agreement: The primary RM sends all updates, if any, to all other RMs. The other RMs reply ACKs
- Response: The primary RM sends a response to the front end

- Linearizability is obvious when the primary RM is correct

Passive Replication – Fault Tolerance

- When the primary RM fails, we need:
- The primary is replaced by a unique backup
- The RMs that survive agree on which operations had been performed at the point when the replacement takes over

- Both can be achieved with view-synchronous group communication

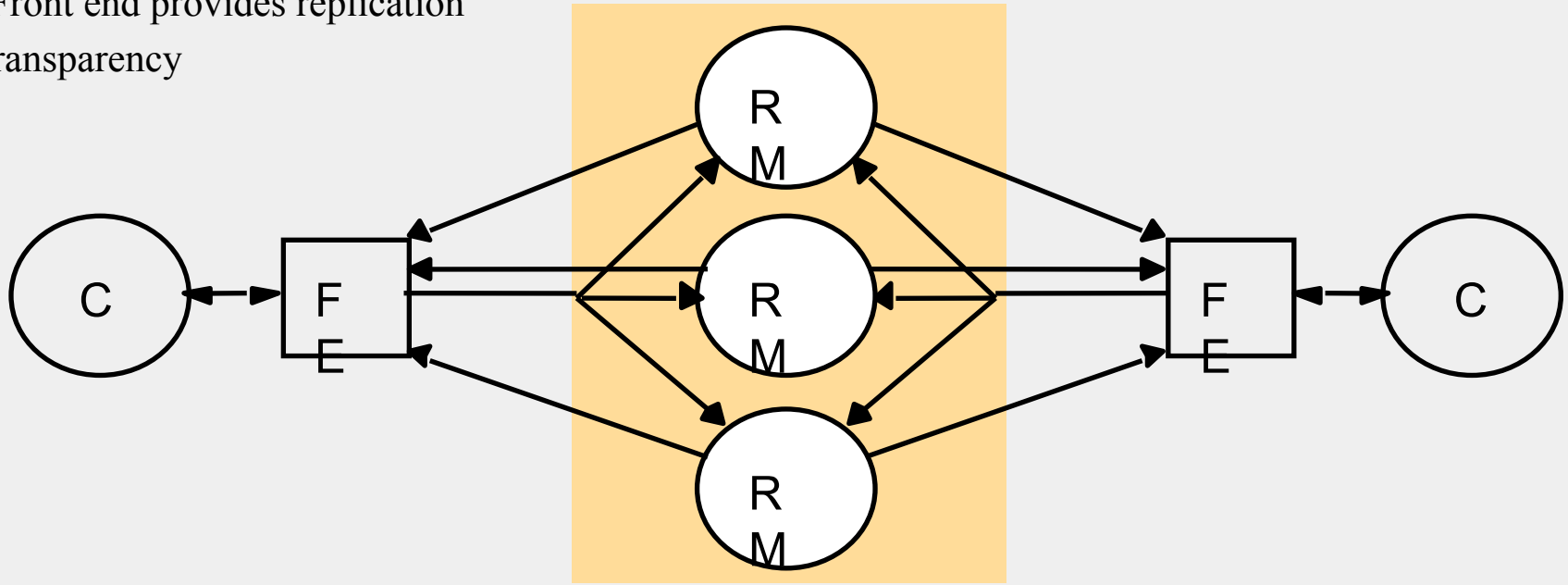
Passive Replication – Fault Tolerance

- The primary is replaced by a unique backup
- Election: with a consistent view, we simply choose the largest id to be the new
- The RMs that survive agree on which operations had been performed at the point when the replacement takes over
- View-synchronous group communications ensure that either all the RMs or none of them will deliver any given update

- Note: View-synchronous group communications is impossible for asynchronous systems. Why?

Active Replication

- Front end talks to all RMs
- Front end provides replication transparency



Active Replication

- Request: The front end issues the request to all RMs, using a totally ordered and reliable multicast
- Coordination: Multicast protocol delivers the request to all correct RMs in the same (total) order
- Execution: All RMs execute the request and reply with response
- Agreement: None.
- Response: The front end may receive multiple response. Drop all duplicate response

- We cannot have linearizability, but we have sequential consistency