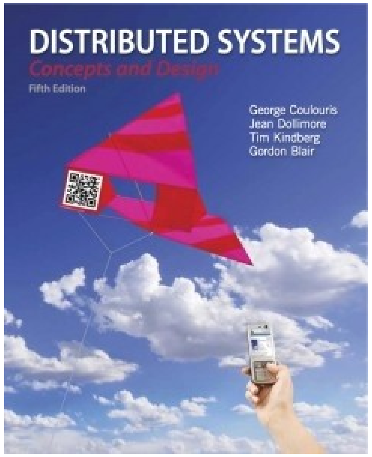
Week 5: Replication and Fault Tolerance



Reference:

Chapter 18

Distributed Systems: Concepts and Design

Coulouris, Dollimore, Kindberg and Blair

Edition 5, © Addison Wesley 2011

Learning Objectives

- Describe multicast and group communication as an important component for replicated service in distributed systems.
  - IP multicast, reliable and ordered multicast
  - Requirements to group communication
- Describe replicated service in terms of:
  - Benefits of using replication
  - Requirements to replication
  - Replication models and operations

Learning Objectives

- Describe fault tolerance in terms of:
  - Requirements to replication for fault tolerance
  - Passive and active replication models for fault tolerance and their features.
  - The role of multicast to satisfy requirements to fault tolerance.

Multicast and Group Communication

- Multicast is an operation that sends a single message from one process to each of the members of a group of processes.

- In general, this is done in such a way that the membership of the group is transparent to the sender.

- A multicast is termed *reliable* if any transmitted message is either received by all members of the group or by none of them.
- A multicast is termed *totally ordered* if all messages transmitted to the group reach all members of the group in the same order.

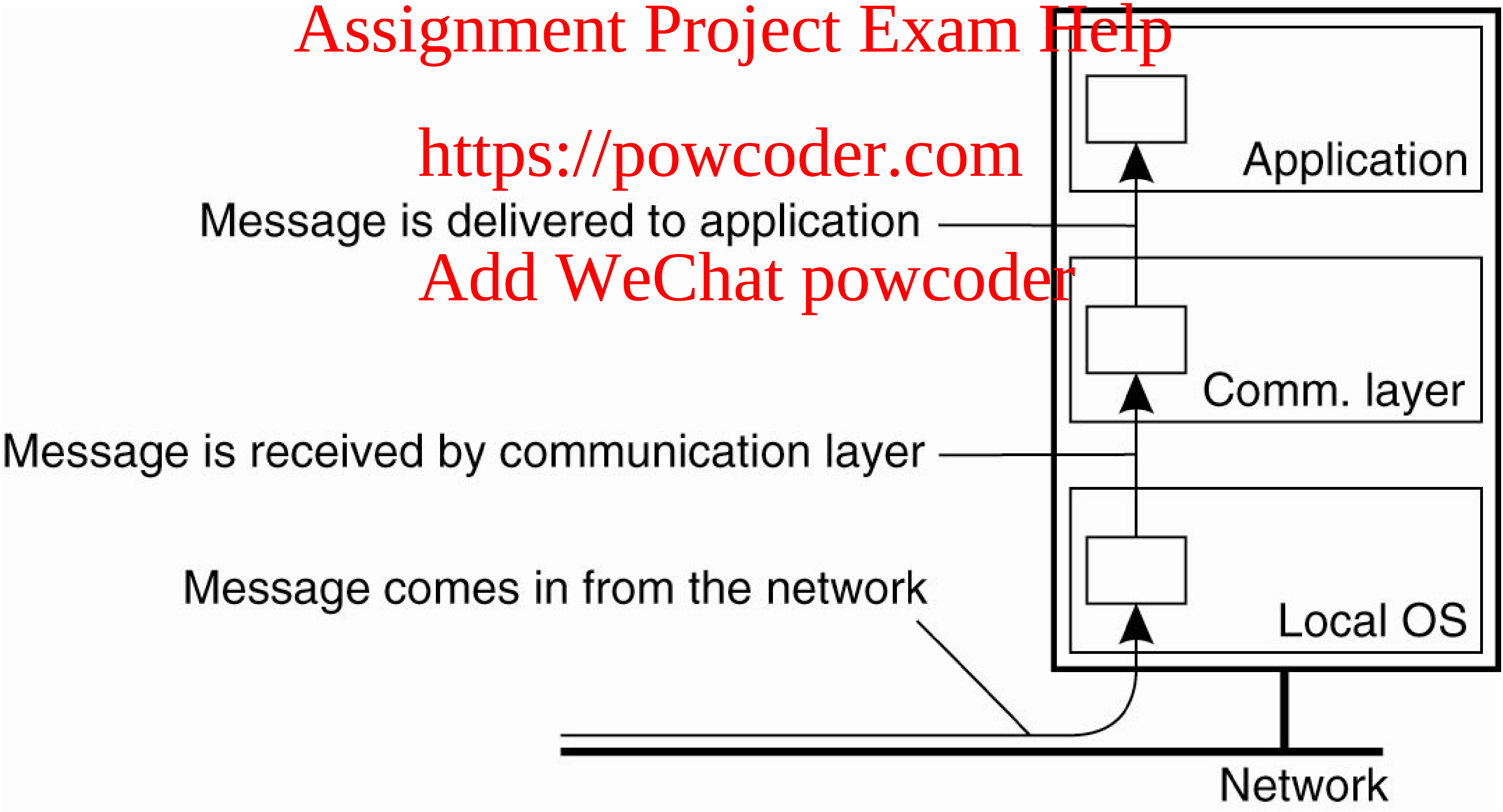
Multicast and Group Communication

- Multicast communication requires coordination and agreement. The aim is for members of a group to receive copies of messages sent to the group.
- Totally ordered and reliable multicast is used in active replication systems to send messages from the *front end* to the *replica managers*.
- In order to achieve a required ordering, a message may not be delivered to the application layer as soon as it is received by a process.
- In other applications other weaker forms of ordering are sufficient.

*

Multicast and Group Communication

- The logical organization of a distributed system to distinguish between message receipt and message delivery


Application
Message is delivered to application
Comm. layer
Message is received by communication layer
Message comes in from the network
Local OS
Network

Multicast and Group Communication

- To discuss group communication
  - The system consists of a collection of processes which can communicate *reliably* over 1 to 1 channels.
  - Processes fail only by crashing (no arbitrary failures).
  - Processes are members of groups, which are the destinations of multicast messages.
  - Multicast message *m* carries the *id* of the sending process *sender*(*m*) and the *id* of the destination group *group*(*m*).

*

Multicast and Group Communication

- The reliable 1 to 1 communication is defined in terms of *validity* and *integrity.*
  - Validity
    - Any message in the outgoing message buffer is eventually delivered to the incoming message buffer.
    - This is achieved by use of acknowledgements and retries.
  - Integrity
    - The message received is identical to one sent, and no messages are delivered twice.
    - This is achieved by use of checksums, rejection of duplicates, e.g. due to retries.

*

Multicast and Group Communication

- Operations used in group communication
  - *multicast*(*g*, *m*) sends message *m* to all members of a process group *g*.
  - *deliver* (*m*) is called to get a multicast message delivered. It is different from *receive* as it may be delayed to allow for ordering or reliability.

IP Multicast

- IP multicast is an implementation of group communication.
    - IP multicast is built on top of IP, allowing the sender to transmit a single datagram to a set of computers that form a multicast group.

    - A multicast group is specified by a class *D* Internet address with *1110* as the first 4 bits (in the range 224.0.0.0 to 239.255.255.255).
    - Being a member of a multicast group allows a computer to receive IP packets sent to the group.
    - The membership of groups is dynamic, allowing computers to join or leave at any time.
    - It is possible to send datagrams to a group without being a member.

Java API to IP Multicast

- Java API provides a datagram interface to IP multicast through the class ***MulticastSocket***.
- A multicast group is specified by a class D IP address and by a standard UDP port number.
- The ***MulticastSocket*** allows sockets to be created to use a specified local port.
- A process can join a multicast group by invoking the ***joinGroup*** method and leave the group by invoking the ***leaveGroup*** method of its multicast socket.
- After joining a group, a process will receive datagrams sent by processes on other computers to that group at that port.

Java Multicast Example

```
import java.net.*;
import java.io.*;
public class MulticastPeer{
  public static void main(String args[]){
            //args give message contents and destination
              multicast group (e.g. "228.5.6.7")
    MulticastSocket s =null;
        try {
          InetAddress group = InetAddress.getByName(args[1]);
          s = new MulticastSocket(6789);
          s.joinGroup(group);
          byte [] m = args[0].getBytes();
          DatagramPacket messageOut =
                       new DatagramPacket(m, m.length, group, 6789);
          s.send(messageOut);
```

☞ This program continues on the next slide

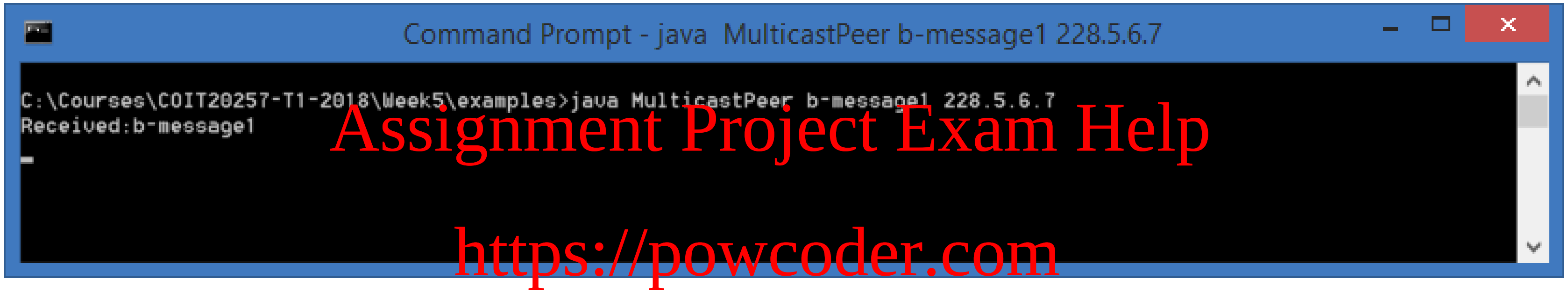
Java Multicast Example

```
       byte[] buffer = new byte[1000];
       for(int i=0; i< 3;i++) {//get messages from others in group
       DatagramPacket messageIn = new DatagramPacket(buffer,
                                     buffer.length);
       s.receive(messageIn);
       System.out.println("Received:" + new
                              String(messageIn.getData()));
       }
       s.leaveGroup(group);
    }catch (SocketException e){
       System.out.println("Socket: " + e.getMessage());
    }catch (IOException e){
       System.out.println("IO: " + e.getMessage());
    }finally {if(s != null) s.close();}
  }
}
```

The end of this program

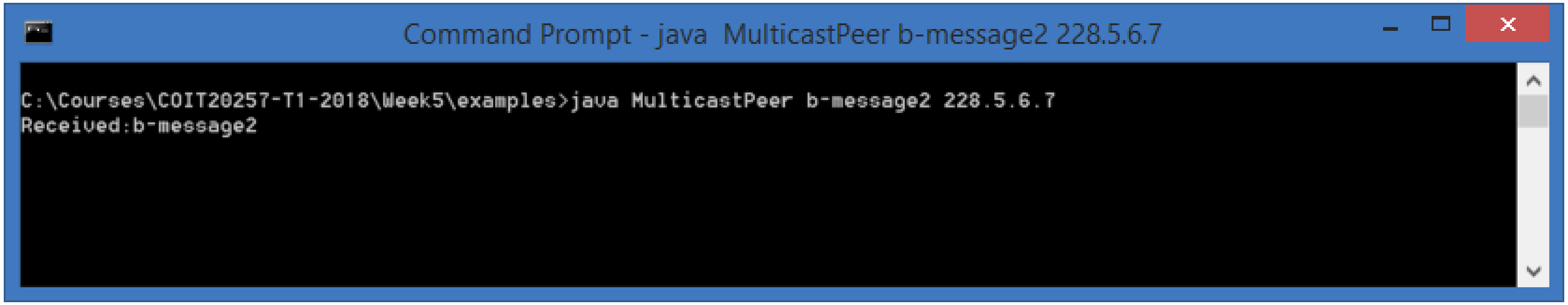
Java IP Multicast Example

- Run ***MulticastPeer*** on one computer (138.77.36.53)



```
C:\Courses\COIT20257-T1-2018\Week5\examples>java MulticastPeer b-message1 228.5.6.7
Received:b-message1
_
```

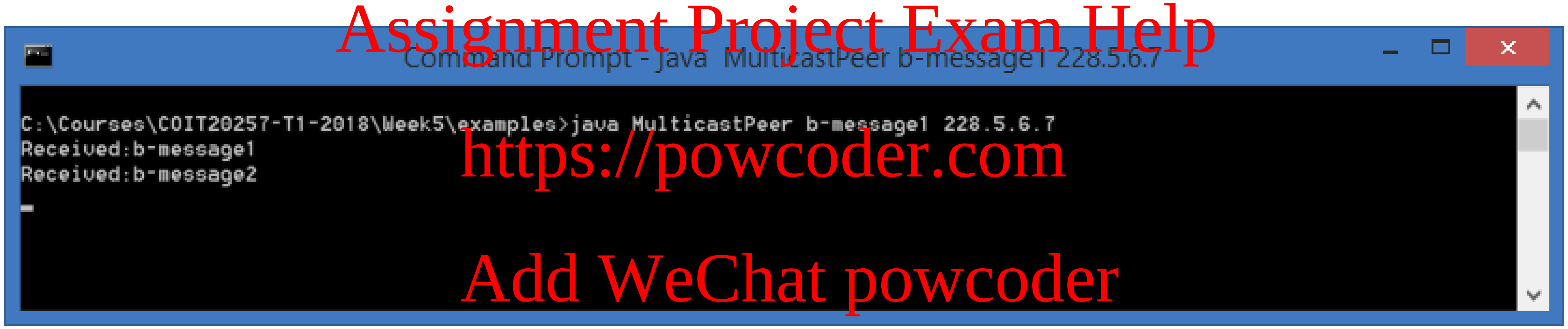
- Run ***MulticastPeer*** on another computer (138.77.36.101)



```
C:\Courses\COIT20257-T1-2018\Week5\examples>java MulticastPeer b-message2 228.5.6.7
Received:b-message2
```

Java IP Multicast Example

- Back to ***MulticastPeer*** on the first computer (138.77.36.53)



```
C:\Courses\COIT20257-T1-2018\Week5\examples>java MulticastPeer b-message1 228.5.6.7
Received:b-message1
Received:b-message2
_
```

IP Multicast

- Datagrams multicast over IP have the same failure characteristic as UDP datagrams – omission failure.
- Messages are not guaranteed to be delivered to any particular group member in the face of even a single omission failure.
  - Datagrams may be dropped by recipients because of full buffer.
  - Dategrams may be lost on the way because of crash of a router.
- Some but not all of the members of the group may receive it – unreliable multicast.

Reliability and Ordering of Multicast

- Fault tolerance based on replicated service
  - A replicated service that consists of the members of a group of servers starts in the same state and always perform the same operation in the same order, so as to remain consistent with one another.
  - This application of multicast requires that either all of the replicas or none of them should receive each request to perform an operation.
  - If one of them misses a request, it will become inconsistent with the others.
  - The replicated service would require that all members receive request messages in the same order as one another.

Reliability and Ordering of Multicast

- Finding the discovery servers in spontaneous networking
  - Any process that wants to locate the discovery servers multicasts requests at periodic intervals for a time after it starts up.
  - A occasional lost request is not an issue when locating a discovery server.
- Propagation of event notifications
  - Multicast to a group may be used to notify processes when something happens.
  - A news system might notify interested users when a new message has been posted.
  - Name or discovery services might announce their existence.
  - The particular application determines the requirements to multicast.

Reliability and Ordering of Multicast

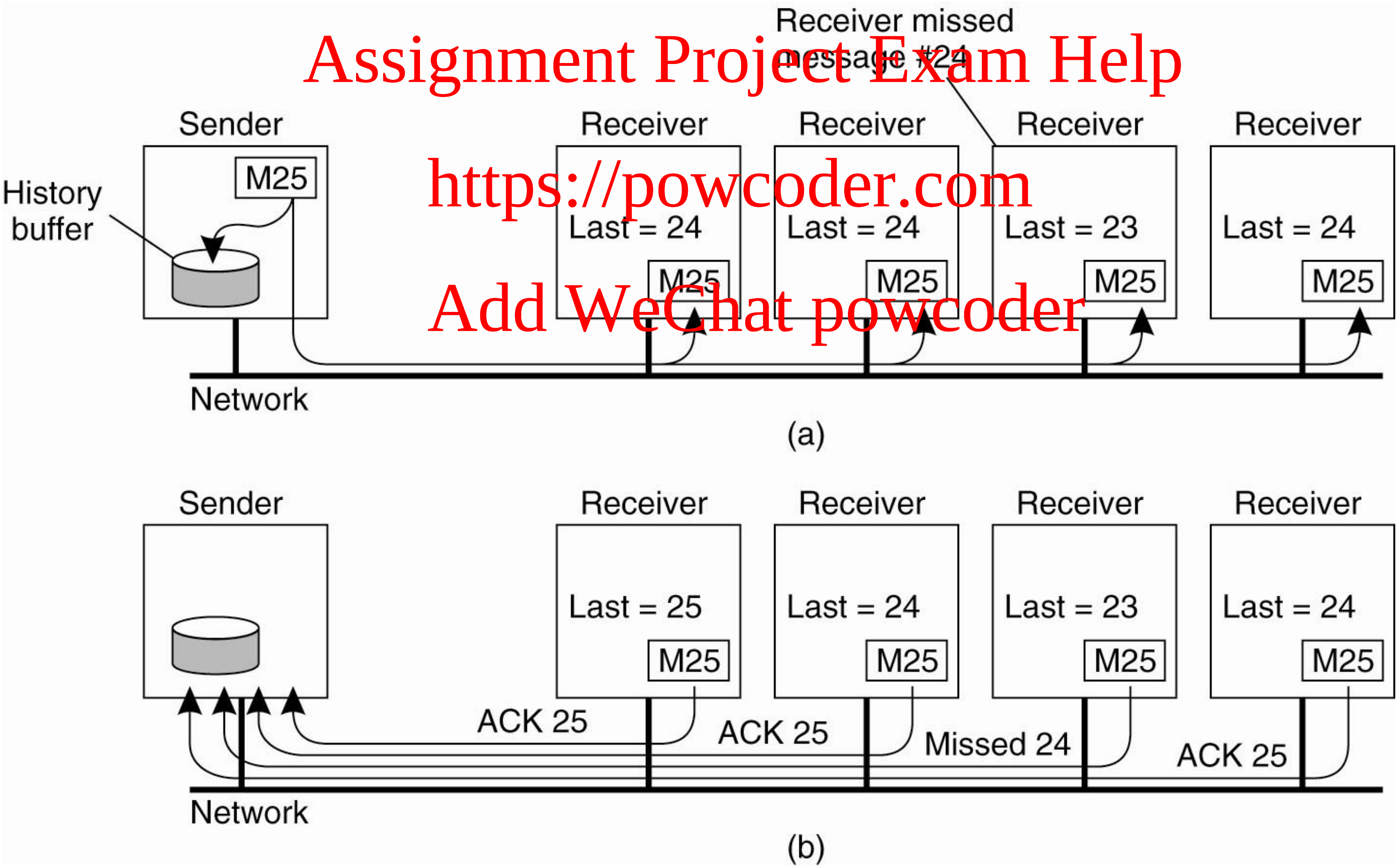
- Better performance through replicated data
  - Data are replicated, for example in users’ caches, to increase performance of a service.
  - Each time the data changes, the new value is multicast to the processes managing the replicas.
  - The effect of lost messages or inconsistent ordering would depend on the importance of all replicas.
  - Requirements to multicast is basically similar to that of fault tolerance based on replicated service.

Reliable Multicast

- A reliable multicast is one that satisfies the following properties:
  - Integrity: A correct process P delivers a message m at most once.
  - Validity: If a correct process P multicasts a message m then P will eventually deliver m.
  - Agreement: If a correct process delivers a message m, then all other correct processes in group(m) will eventually deliver m.
- A number of algorithms are introduced by Section 15.4.1 to 15.4.2 to implement multicast.
  - Some of them are unreliable (B-multicast, IP multicast) etc.
  - Reliable multicast can implemented on unreliable multicast by techniques such as sequencing and acknowledgement etc.

Reliable Multicast

- A simple solution to reliable multicasting when all receivers are known and are assumed not to fail.
- (a) Message transmission (b) Reporting feedback



(a)

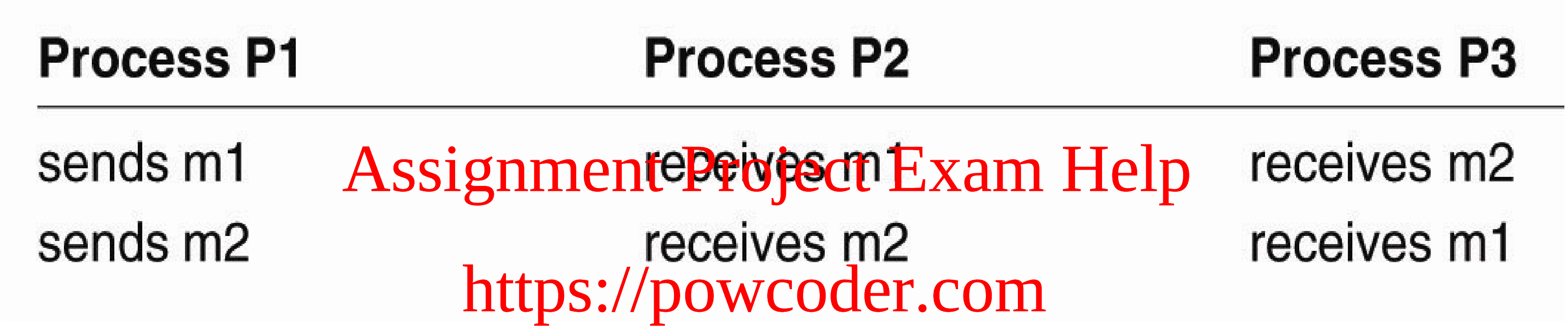
(b)

Ordered Multicast

- The basic multicast algorithm delivers messages to processes in an arbitrary order.
- This is due to arbitrary delays in the underlying one-to-one send operations.
- The lack of an ordering guarantee is not satisfactory for many applications.
    - The replicated service requires ordered multicast.
- A variety of orderings may be implemented, but ordering is expensive in delivery latency and bandwidth consumption.

Ordered Multicast

- Unordered multicast

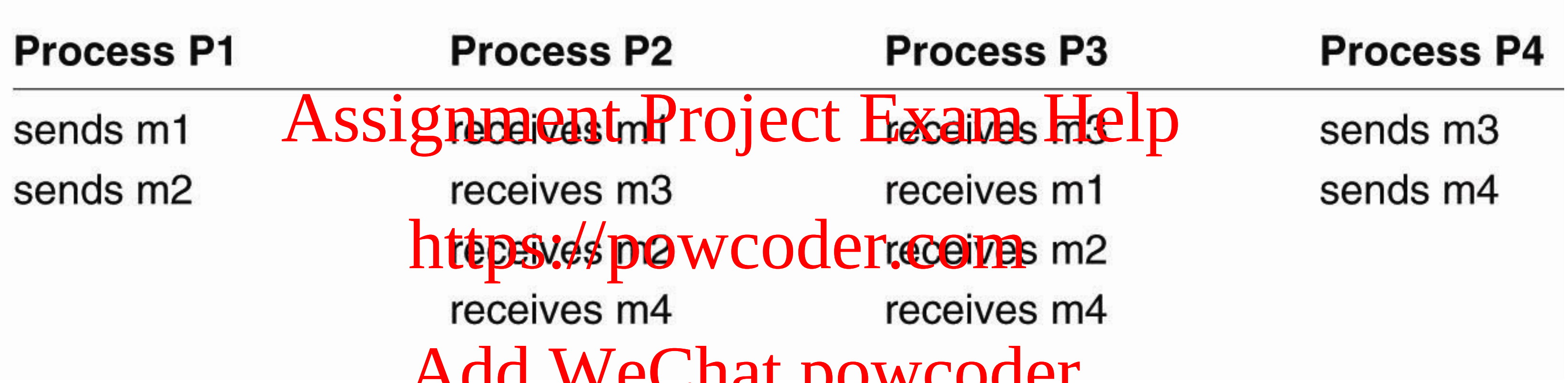


Process P1	Process P2	Process P3
sends m1	receives m1	receives m2
sends m2	receives m2	receives m1

- FIFO ordering
  - If a correct process issues *multicast*(*g*, *m*) and then *multicast*(*g*,*m'*), then every correct process that delivers *m'* will deliver *m* before *m'* .
  - In other words, the communication layer is forced to deliver incoming messages from the same process in the same order as they have sent.

Ordered Multicast

- FIFO ordering delivery



Process P1	Process P2	Process P3	Process P4
sends m1	receives m1	receives m3	sends m3
sends m2	receives m3	receives m1	sends m4
	receives m2	receives m2	
	receives m4	receives m4	

- Causal ordering
  - If *multicast*(*g*, *m*) → *multicast*(*g*, *m'*), where → is the happened-before relation between messages in group *g*, then any correct process that delivers *m'* will deliver *m* before *m'* .

Ordered Multicast

- Causal ordering
  - In other words, if a message *m* causually precedes another message *m’* , regardless of whether they were multicast by the same sender, then communication layer at each receiver will always deliver *m’* after m.
  - The example in last page is not causally ordered.

Ordered Multicast

- Total ordering
  - If a correct process delivers message *m* before it delivers *m'*, then any other correct process that delivers *m'* will deliver *m* before *m'*.
  - In other words, regardless of whether message delivery is unordered, FIFO ordered, or causally ordered, when messages are delivered, they are delivered in the same order to all group members.
  - The example in last page is not totally ordered.
- Atomic multicasting
  - Virtually synchronous reliable multicasting offering totally-ordered delivery of messages

Replication

- Replication is a technique for enhancing services: multiple copies of data are maintained at multiple computers. This can lead to:
  - Performance Enhancement
    - For example, caching data in the web provides improved performance in terms of response time to users and lower utilisation of servers.
    - In some cases the service itself may be replicated.
    - For example, several web servers can have the same DNS name and the servers are selected in turn to share the load.

Replication

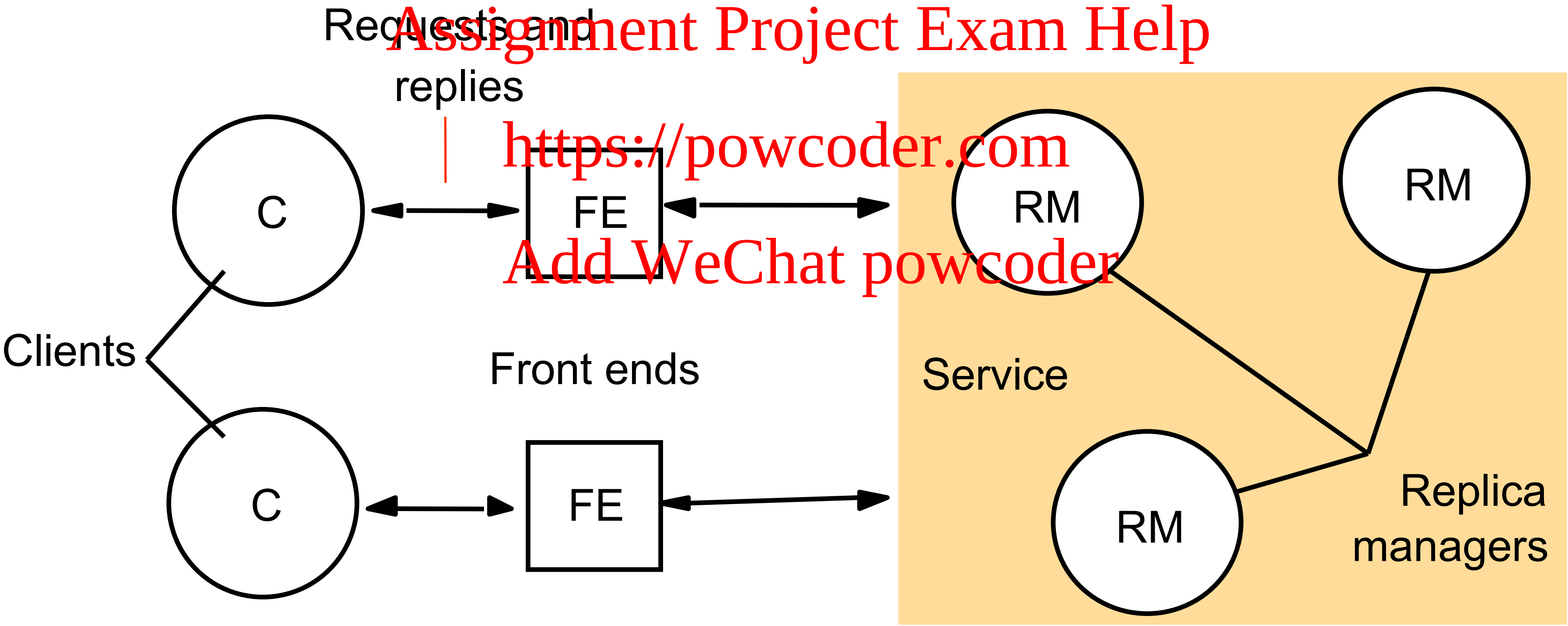
- Increased Availability
  - The availability of a service is the probability that a response is obtained within a reasonable time bound.
  - Delays can be due to service features such as data locking as well as server and communication infrastructure failures.
  - Replicate data at failure-independent servers and when one fails, client may use another.
- Fault Tolerance
  - A fault-tolerant service always guarantees strictly correct behaviour even in the presence of a certain number and type of faults.
  - Correctness here may concern the integrity of the data with respect to client operations, and/or timeliness of the response.
  - If *f* of *f*+1 servers crash then 1 remains to supply the service.
  - If *f* of 2*f*+1 servers have byzantine faults then they can still supply a correct service.

Replication Model

- Replication transparency
  - The client should not normally be aware that multiple physical copies of the data exist, either in terms of submitted requests or returned values.
  - Clients see logical objects rather than several physical copies and they access one logical item and receive a single result.
  - This is achieved by interposing a *Front End* between the client and the service.
- Replication consistency
  - There are also issues of consistency between the replicas although the degree of inconsistency tolerated will depend on the service and the motivation for the replication.

Replication Model

- Basic architectural model of replication



Replication Model

- Each logical object is implemented by a collection of physical copies called replicas.
- A Replica Manager contains replicas on a computer and access them directly. Objects are copied at all RMs.
- RMs apply operations to replicas recoverably and they do not leave inconsistent results if they crash.
- Static systems are based on a fixed set of RMs. In dynamic system, RMs may join or leave (e.g. when they crash).

Replication Model

- An RM can be a *state machine*, which has the following properties:
  - Applies operations atomically.
  - 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.
- In general, five phases are involved in the execution of a single request on a replicated object.

Replication Model

- Initiation
  - The front end sends the request to a single RM that passes it on to the others.
  - The front end multicasts the request to all of the RMs (in state machine approach)
- Coordination
  - The RMs coordinate in preparation for executing the request consistently.
  - They may agree on whether the request should be executed and the ordering of this request relative to others.
- Execution
  - The RMs execute the request, possibly tentatively, i.e. in such a way that they can undo the effects later if necessary.

Replication Model

- Agreement
  - The RMs reach consensus on the effects of the request (if any) that will be committed.
- Response
  - One or more RMs respond to the front end.
  - In some systems, it is the responsibility of the front end to collect responses from a collection of RMs and select or synthesise a response for the client.
    - For high availability, give first response to client.
    - To tolerate byzantine faults, take a vote.

Replication Model

- Depending on the application, different ordering semantics may be appropriate for the handling of requests.
- These are related to the possible orderings in multicast.
  - FIFO ordering: if the front end issues r then r', then any correct RM handles r before r'.
  - Causal ordering: If the issue of request r happened-before the issue of request r', then any correct RM that handles r' handles r before it.
  - Total ordering: If a correct RM handles request r before request r', then any correct RM that handles r' handles r before it.

*

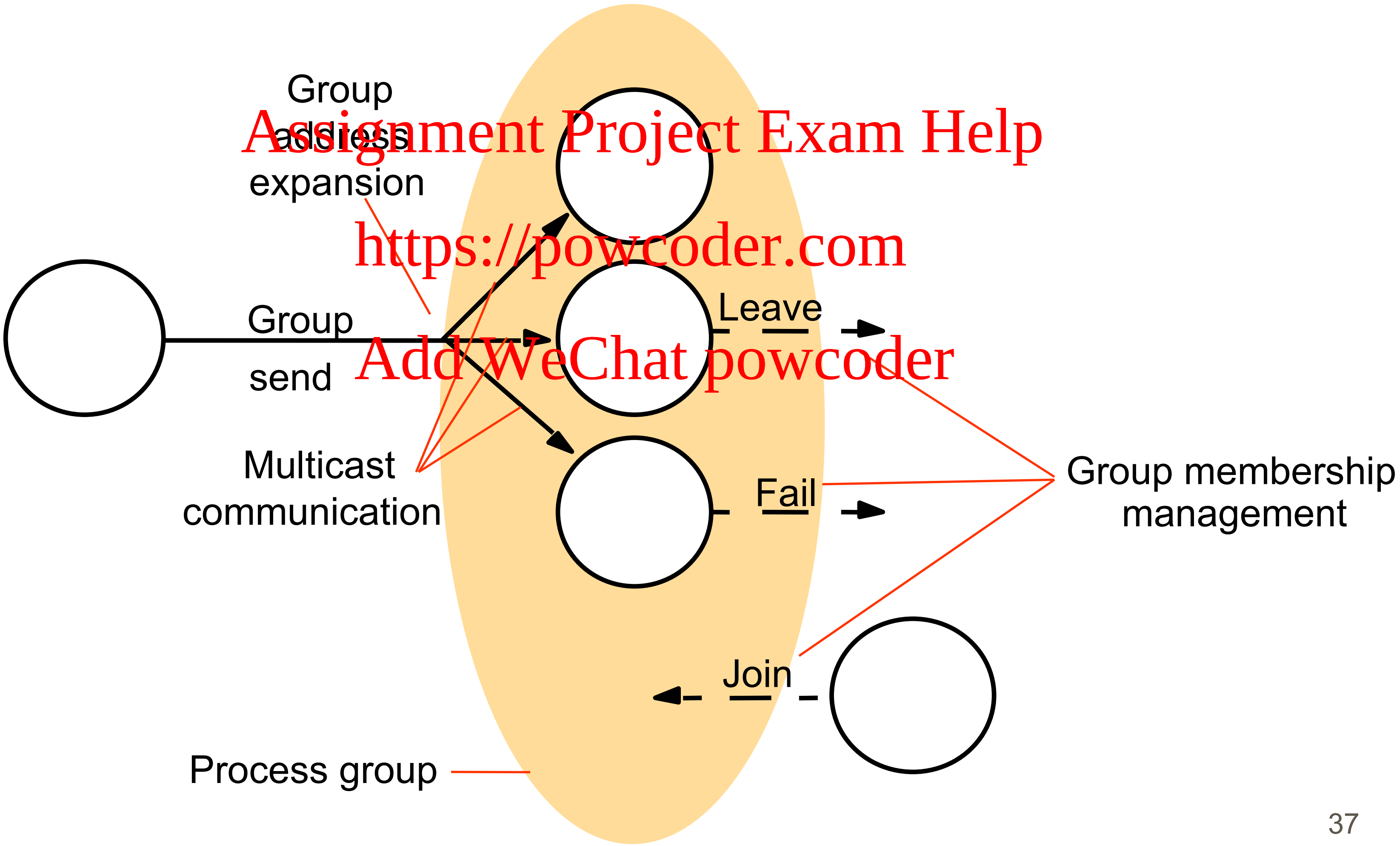
Replication Model

- Process groups are useful for managing replicated data.
- Replication systems need to be able to add/remove RMs and this is achieved by group membership service.
  - Provide interface for adding/removing members.
  - Implements a failure detector to monitor members for failures (crashes/communication) and excludes them when unreachable.
  - Notifies members of changes in membership.
  - Expands group addresses when multicasts addressed to group identifiers.

*

Replication Model

- Group membership service



Fault Tolerance

- An naive replication system operates like this:
  - RMs at *A* and *B* maintain copies of *x* and *y*
  - Clients use local RM when available, otherwise the other one.
  - RMs propagate updates to one another after replying to client.

- Suppose the initial balance of x and y is $0
  - Client 1 updates x at B (local) then finds B has failed, so uses A instead.
  - Client 2 reads balances at A (local)
    - As client 1 updates y after updating x, client 2 should see $1 for x.
    - However, the update to bank account x from B has not arrived since B failed.
  - That is not the behaviour that would occur if A and B were implemented at a single server.

Fault Tolerance

- An naive replication system

time ↓

Client 1:	Client 2:
setBalance$_B$(x,1)	
setBalance$_A$(y,2)	
	getBalance$_A$(*y*) →2
	getBalance$_A$(x) →0

Fault Tolerance

- Systems can be constructed to replicate objects without producing this anomalous behaviour.
- A fault tolerant service based on replication should be able to keep responding despite failures and clients should not be able to tell the difference between the replicated service and one provided by a single correct RM.

- Care is needed to avoid anomalies with respect to the consistency of the data.
- Various notions of correctness have been defined such as Linearizability and Sequential Consistency.

Fault Tolerance

- A replicated object service is *linearizable* if for any execution there is some interleaving of clients’ operations such that:
  - The interleaved sequence of operations meets the specification of a single correct copy of the objects.
  - The order of operations in the interleaving is consistent with the real time at which they occurred.
- A replicated object service is sequentially consistent if for any execution there is some interleaving of clients’ operations such that:
  - The interleaved sequence of operations meets the specification of a single correct copy of the objects
  - The order of operations in the interleaving is consistent with the program order in which each client executed them.

Fault Tolerance

- No-linearizable but sequentially consistent

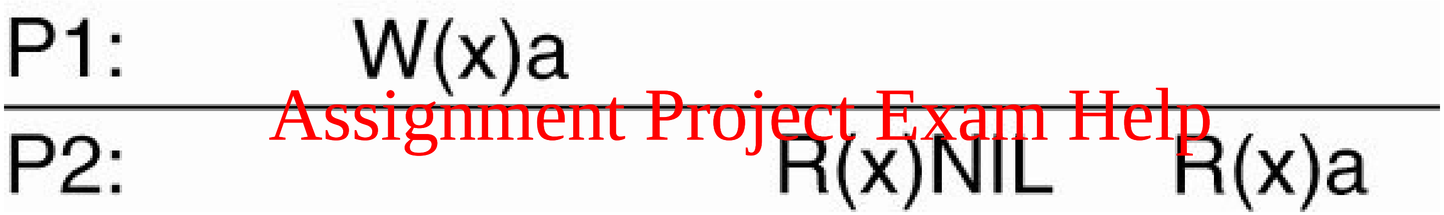
time

Client 1:	Client 2:
$\mathit{setBalance}_B(x,1)$	
	$\mathit{getBalance}_A(y)$ →0
	$\mathit{getBalance}_A(x)$ →0
$\mathit{setBalance}_A(y,2)$	

- This execution is possible under a naive replication strategy, even if neither A or B fails.
- The update at B has not yet been propagated to A when client 2 reads it.

Fault Tolerance

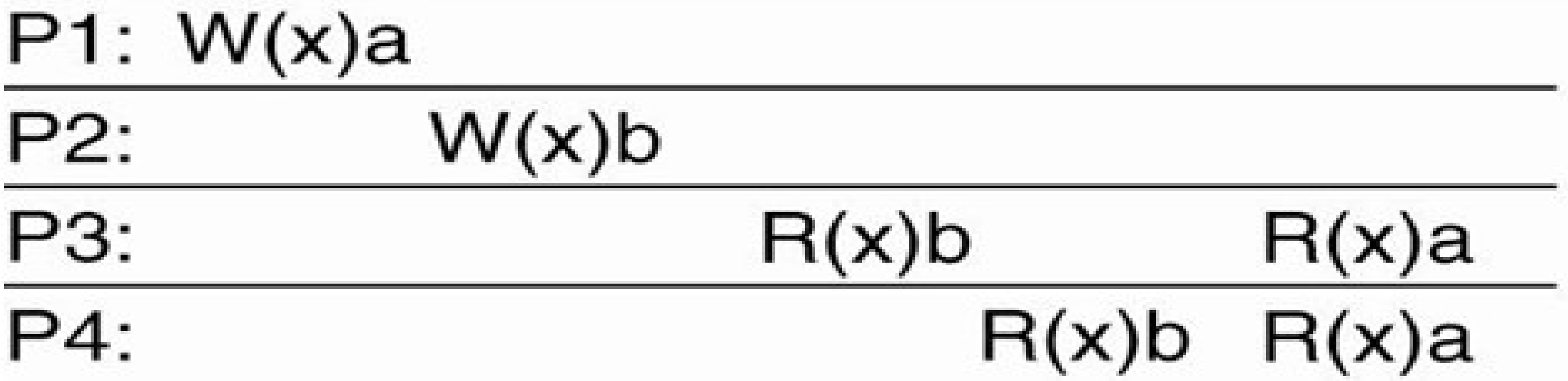
- No-linearizable consistent data store



P1: W(x)a

P2: R(x)NIL R(x)a

- A sequentially consistent data store



P1: W(x)a

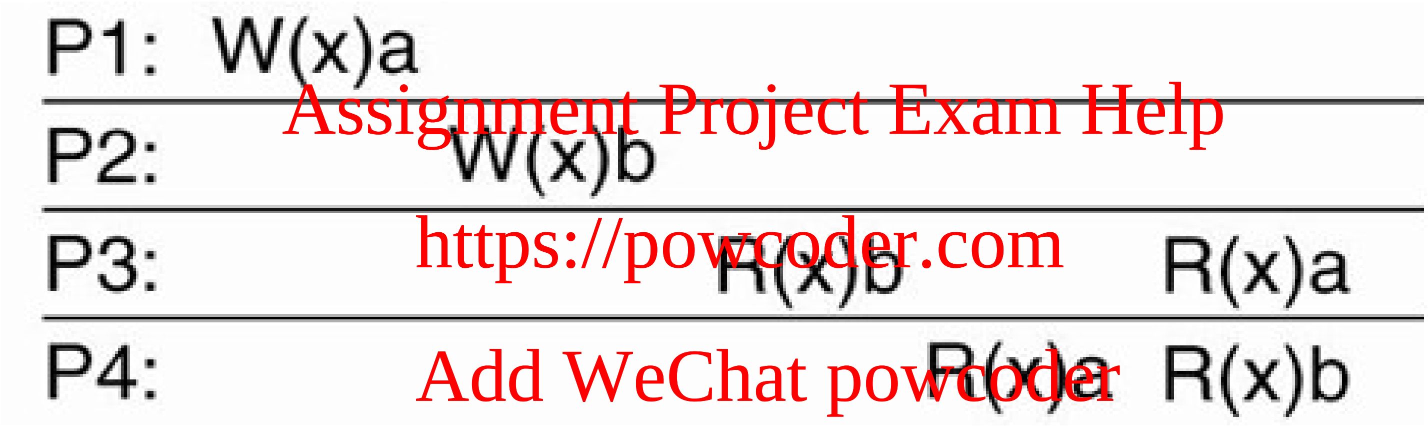
P2: W(x)b

P3: R(x)b R(x)a

P4: R(x)b R(x)a

Fault Tolerance

- A non sequentially consistent data store



P1: W(x)a

P2: W(x)b

P3: R(x)b R(x)a

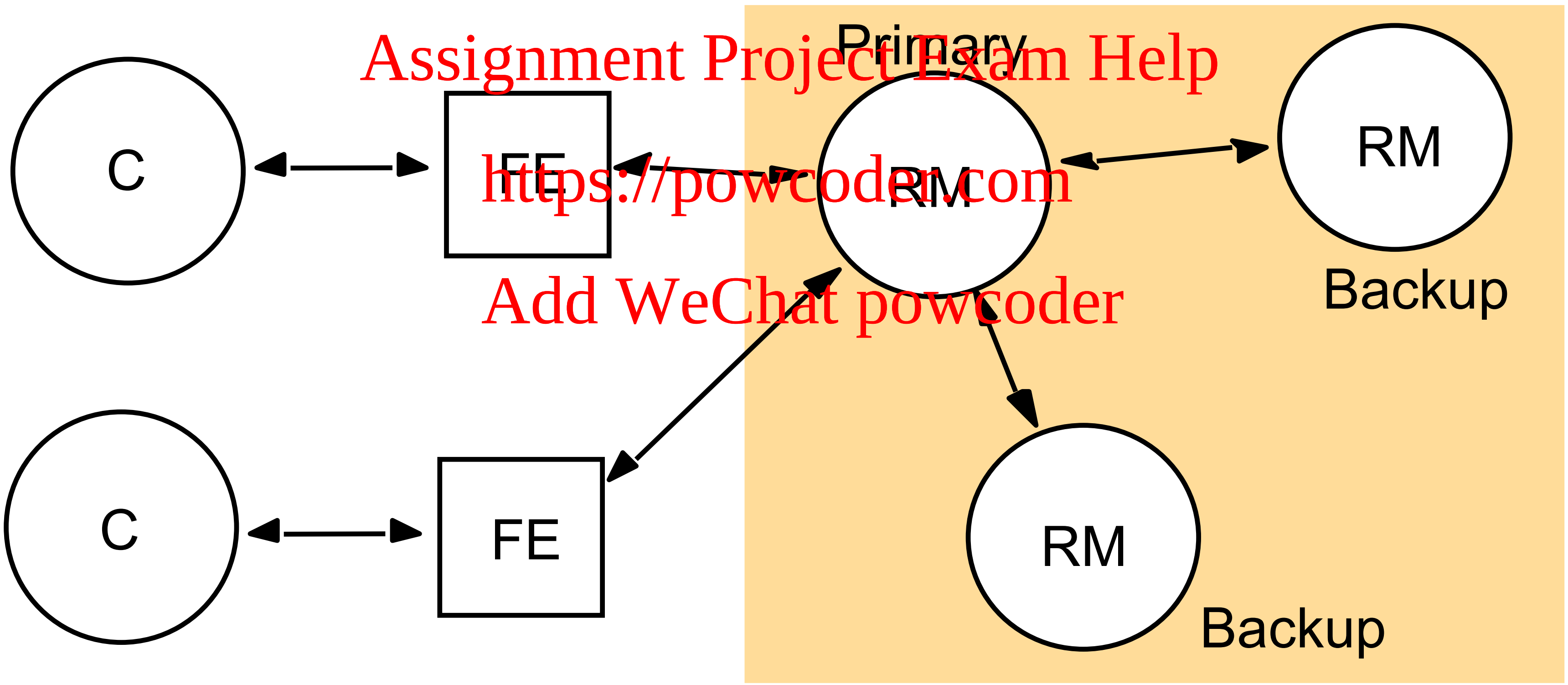
P4: R(x)a R(x)b

Passive Replication

- The passive model of replication for fault tolerance is known as the *primary-backup* model.
- One RM is distinguished as the primary one, which is responsible for:
  - Executing all requests.
  - Updating the other RMs, known as backups or slaves.
- All front ends communicate with the primary RM.
- If the primary RM fails, it is replaced by one of the backups.

Passive Replication

- The primary-backup model for fault tolerance



Passive Replication

- The following sequence is followed if the primary is correct and guarantees linearizability:
  - Request: the front end issues a request to the primary RM. Each request contains a unique identifier.
  - Coordination
    - The primary atomically deals with each request in FIFO order.
    - It checks the unique id. If it has already done the request, it re-sends the response.
  - Execution: the primary executes the request and stores the response.

*

Passive Replication

- Agreement
  - If the request is an update, the primary will propagate it, together with the response and the request id to all the backup RMs.
  - The backups will send an acknowledgement to the primary.
- Response: the primary responds to the front end; the front end responds to the client.

- If the primary RM fails, one of the backup RMs should take its place.

*

Passive Replication

- The system will maintain linearizability
  - If the primary is replaced by a unique backup.
  - If all the surviving RMs agree on which operations had been performed at the point at which replacement occurs.
- These requirements are met
  - If the RMs are organised as a group.
  - If view-synchronous group communication is used to send updates to the backups.

*

Passive Replication

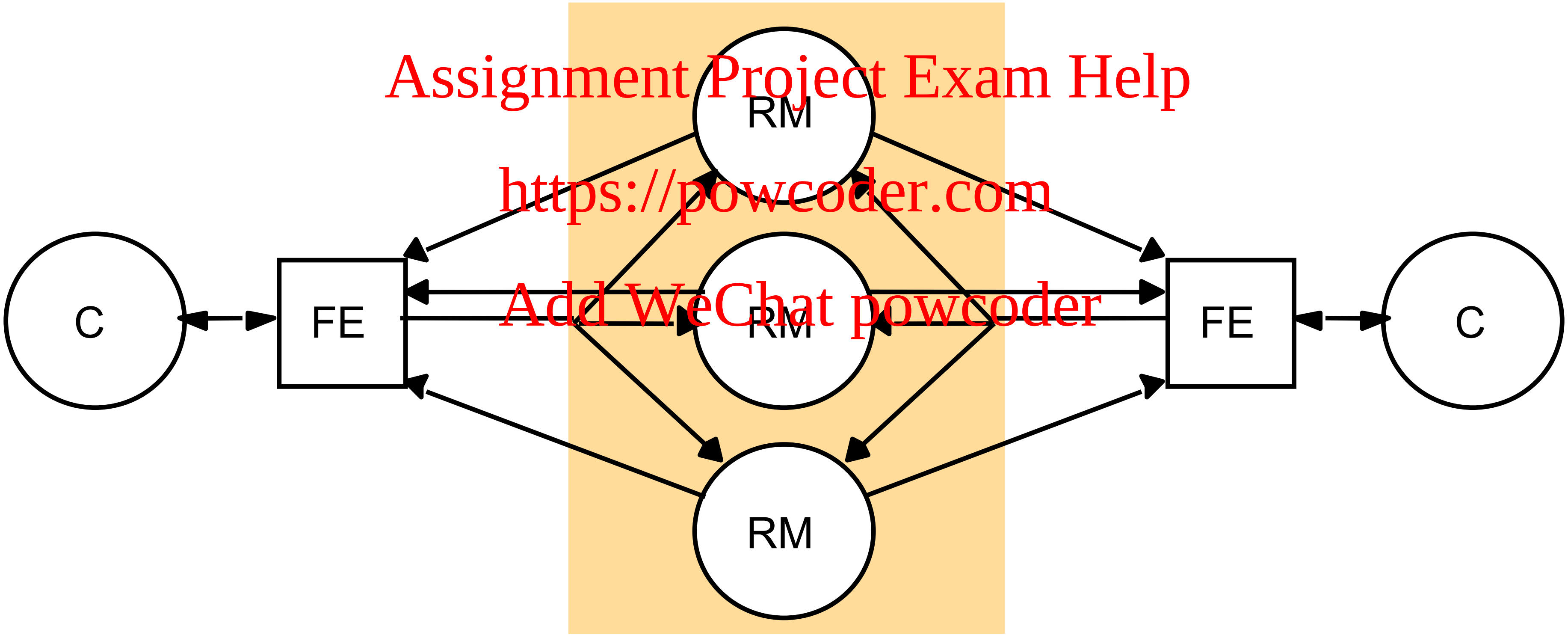
- By view-synchronous group communication
  - The view of the group will be consistent amongst the remaining RMs and will exclude the failed primary.
  - A predefined function will be used to select the new primary from that view and this RM can assume the role.
  - View-synchronous semantics guarantee that either all the backups or none of them will deliver any given update before delivering the new view.
- A passive replication system cannot survive byzantine failures.
- Basing the updates on view-synchronous communication is costly in terms of overhead.

Active Replication

- The RMs are *state machines* all playing the same role and organised as a group.
- All RMs start in the same state and perform the same operations in the same order so that their state remains identical.
- Front ends multicast their requests to the group and each RM processes the request independently but identically and replies.
- If an RM crashes, it has no effect on performance of the service because the others continue as normal.
- It can tolerate byzantine failures because the front end can collect and compare the replies it receives.

Active Replication

- Active replication



Active Replication

- The five phases to perform a client's request:
  - Request: the front end multicasts the request to all RMs, using totally ordered, reliable multicast, after attaching a unique identifier to it.
  - Coordination: the request is delivered to all correct RMs in the same order, depending on properties of the group communication used.
  - Execution
    - Every correct RM executes the request, producing the same result.
    - This is guaranteed since the RMs are all state machines and they each handle requests in the same order.
    - Each response contains the unique identifier.

*

Active Replication

- Agreement: no agreement is required because all RMs execute the same operations in the same order, due to the properties of the totally ordered multicast.

- Response
  - Every RM sends its response to the front end. Differing policies can be enforced here.

  - For example, the front end may respond to the client on the first response, discarding subsequent responses with the same identifier.

  - To tolerate byzantine failures, the frond end can compare the replies it receives.

*

Active Replication

- As RMs are state machines, sequential consistency is maintained.
  - Due to reliable totally ordered multicast, the RMs collectively do the same as a single copy would do.
- The replication scheme is not linearizable
  - Due to the total order is not necessarily the same as real-time order.
- To deal with byzantine failures
  - For up to *f* byzantine failures, use 2*f*+1 RMs.
  - Front end collects f+1 identical responses.
- To improve performance
  - Front ends send read-only requests to just one RM.

Summary

- Reliable and ordered multicast is the fundamental to maintain data consistency for replicated service.
- Replication is a technique for performance enhancement, increased availability, and fault tolerance in distributed systems.
- Replication transparency and consistency are fundamental requirements to replication models.

Summary

- A replication model comprise components such as clients, front ends and replication managers and it performs 5 stage operations for each client’s request to satisfy replication requirements.
- Linearizability and sequential consistency are two types of requirement to replication for fault tolerance.
- Passive replication and active replication are two basic models for fault tolerance. They depend on multicast techniques to implement linearizability and sequential consistency.