

Distributed Systems

Group communication

Alberto Montresor
Università di Trento

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Introduction

Replication: the maintenance of copies of data at multiple machines

◆ Performance enhancement

- ◆ e.g. several web servers can have the same DNS name and the servers are selected in turn, to share the load
- ◆ replication of read-only data is simple, but replication of mutable data incurs overheads in form of protocols

◆ Increased availability

- ◆ Problem: server failures
 - ◆ if each of n servers has probability p of failure, the availability of the system is $1-p^n$
- ◆ Problem: network partitions
 - ◆ Possibility of working in disconnected mode

System model

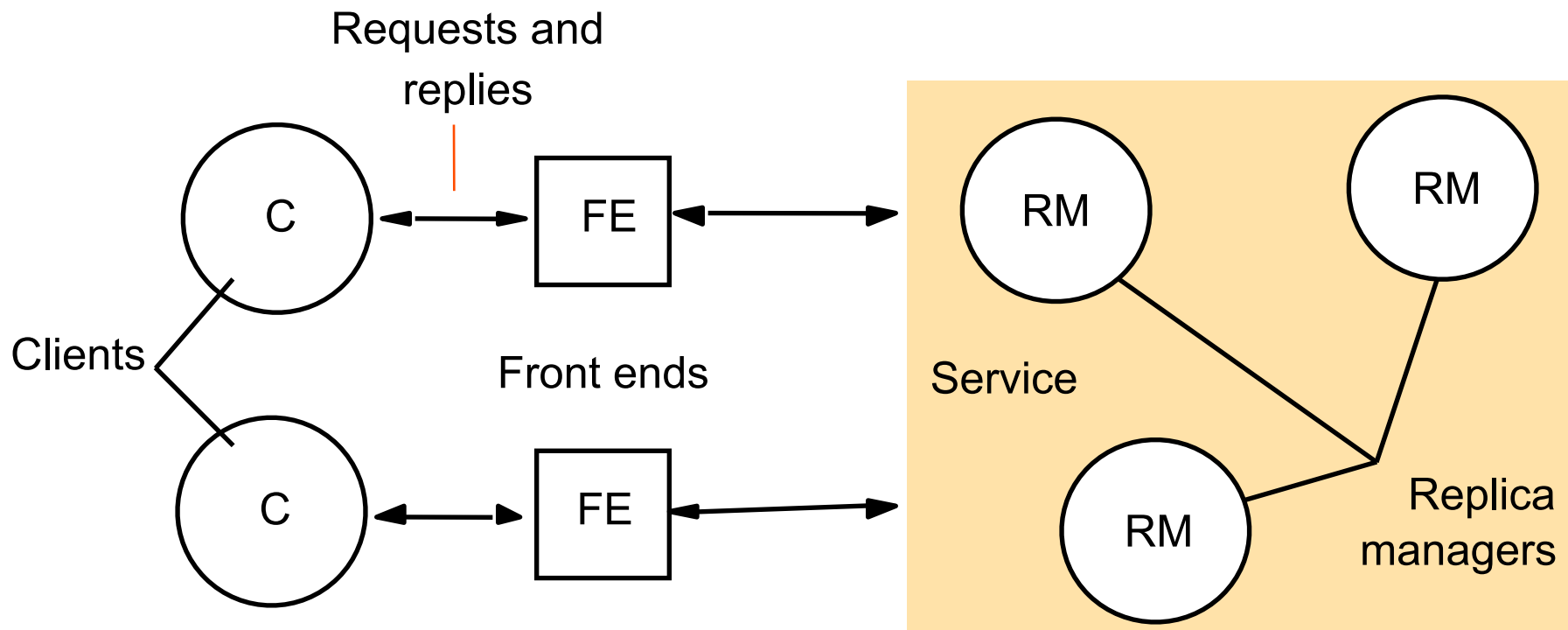
- ◆ **A *logical object* is implemented by a collection of *physical copies* called *replicas***
- ◆ **Replica manager (RM)**
 - ◆ An RM contains replicas on a computer and access them directly
 - ◆ RMs apply operations to replicas recoverably
 - ◆ i.e. they do not leave inconsistent results if they 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)
 - ◆ RM are **state machines**

Requirements for replicated data

- ◆ **Replication transparency**
 - ◆ clients see a single logical object
 - ◆ they are not aware that it is composed of several physical copies
 - ◆ they access one logical item and receive a single result
- ◆ **Consistency**
 - ◆ The consistency model specifies a contract between the replicated system and its clients
 - ◆ The contract describes how operation performed upon a collection of replicated objects produce “correct” results
 - ◆ The concept of correctness depends on the application

A basic architectural model for the management of replicated data

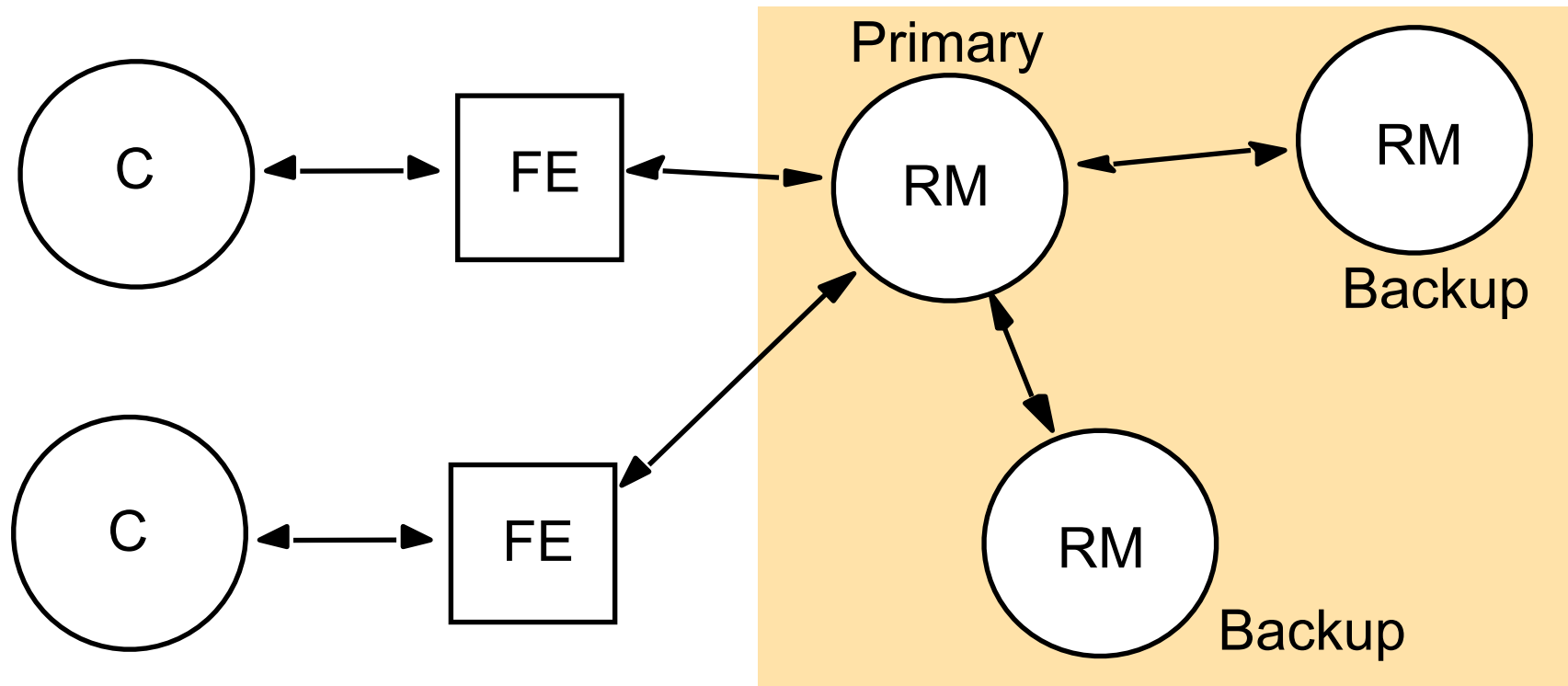
- ◆ **Clients request operations:**
 - ◆ those without updates are called *read-only* requests
 - ◆ the others are called *update* requests (they may include reads)
- ◆ **Clients request are handled by front-ends. A front end makes replication transparent.**



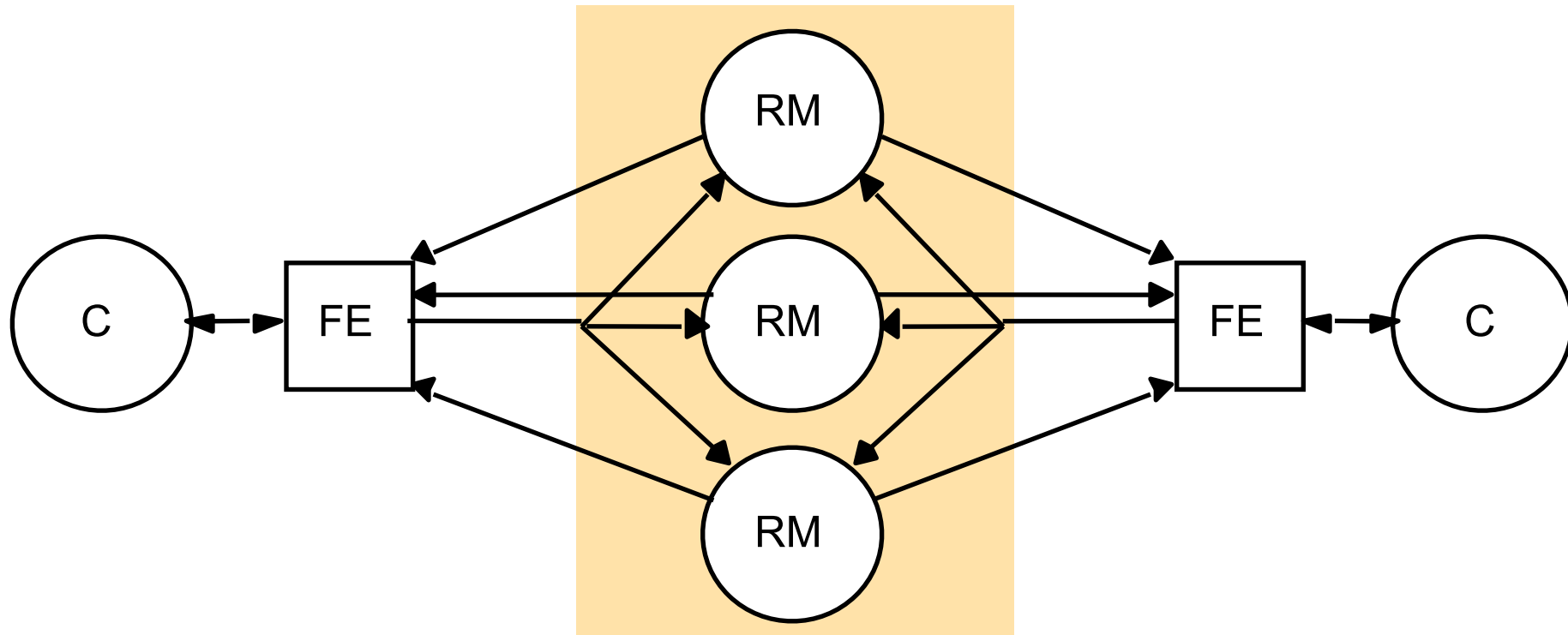
Five steps to issue a request

- ◆ **Issue request; the FE either**
 - ◆ sends the request to a single RM that passes it on to the others
 - ◆ or multicasts the request to all of the RMs (state machine)
- ◆ **Cordination**
 - ◆ RMs decide whether to apply the request; and decide on its ordering relative to other requests (FIFO, causal or total ordering)
- ◆ **Execution**
 - ◆ RMs execute the request (tentatively, i.e., they can undo it)
- ◆ **Agreement**
 - ◆ RMs agree on the effect of the request
- ◆ **Reply**
 - ◆ One or more RMs reply to FE

Passive replication



Active replication

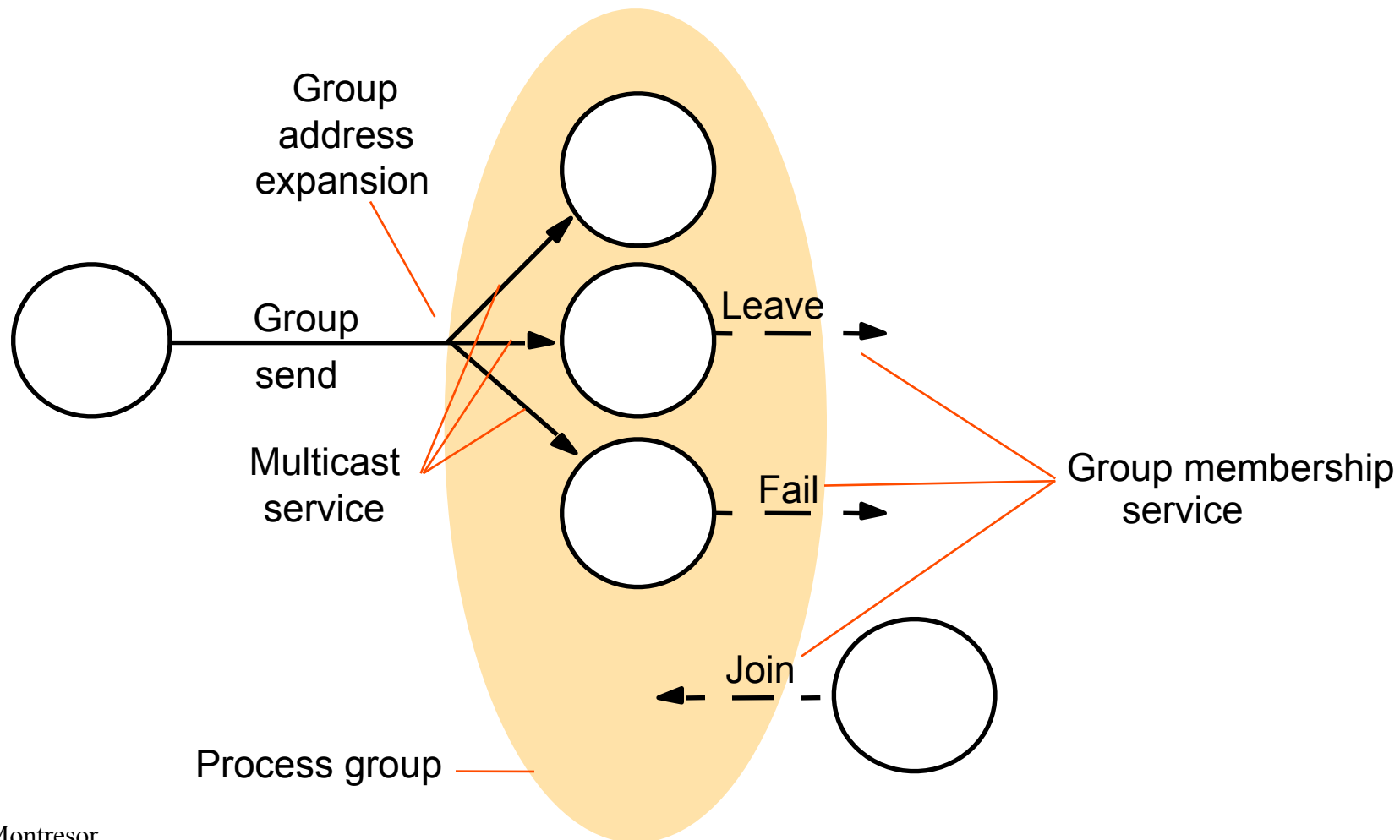


Motivation for group communication

- ◆ **Basic services seen so far (reliable broadcast, consensus):**
 - ◆ are potentially sufficient to implement replicated services...
 - ◆ ... but are not flexible enough
- ◆ **What they lack?**
 - ◆ Management of dynamic groups
 - ◆ Voluntarily join, leave operation
 - ◆ Monitoring of the status of the system
 - ◆ Detection, Reaction
 - ◆ Integration of
 - ◆ reliable communication primitives
 - ◆ agreement protocols

Group communication

- ◆ A group communication service (GCS) is composed by:
 - ◆ A *group membership service*
 - ◆ A *reliable multicast service*



Group communication

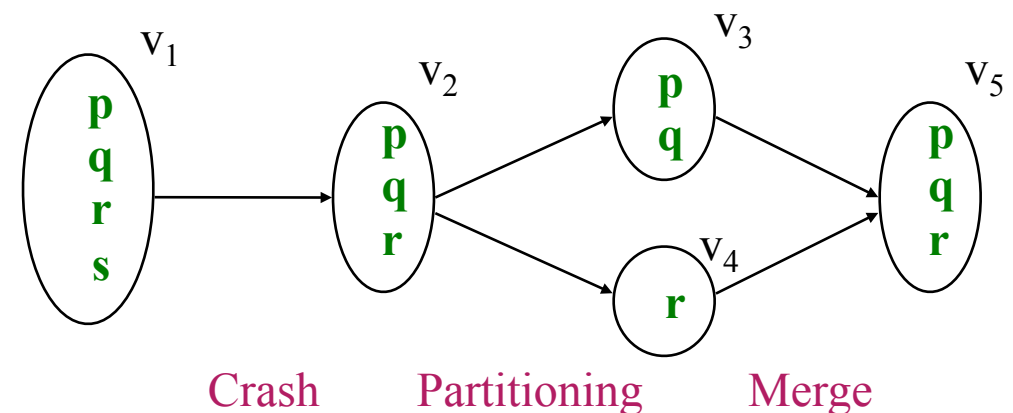
- ◆ **Tasks of a group membership service:**
 - ◆ to enable the **creation** of objects groups
 - ◆ to enable objects to voluntarily **join/leave** a group
 - ◆ to implement a failure detector to **monitor failures** of objects and **partitionings**
 - ◆ to keep members **consistently** informed about changes in the current membership of a group through notification events called **view installations**

Processes take decisions based on the current view; if distinct processes have inconsistent views they take inconsistent decisions

Group membership

- ◆ **A view is composed of**
 - ◆ a set of replica managers
 - ◆ a unique identifier
- ◆ **View installation**
 - ◆ We say that a process p *installs* a view v
 - ◆ v becomes the *current view* of p , until the next view change

- ◆ **A view** represents the perception of the group membership shared between the group members themselves

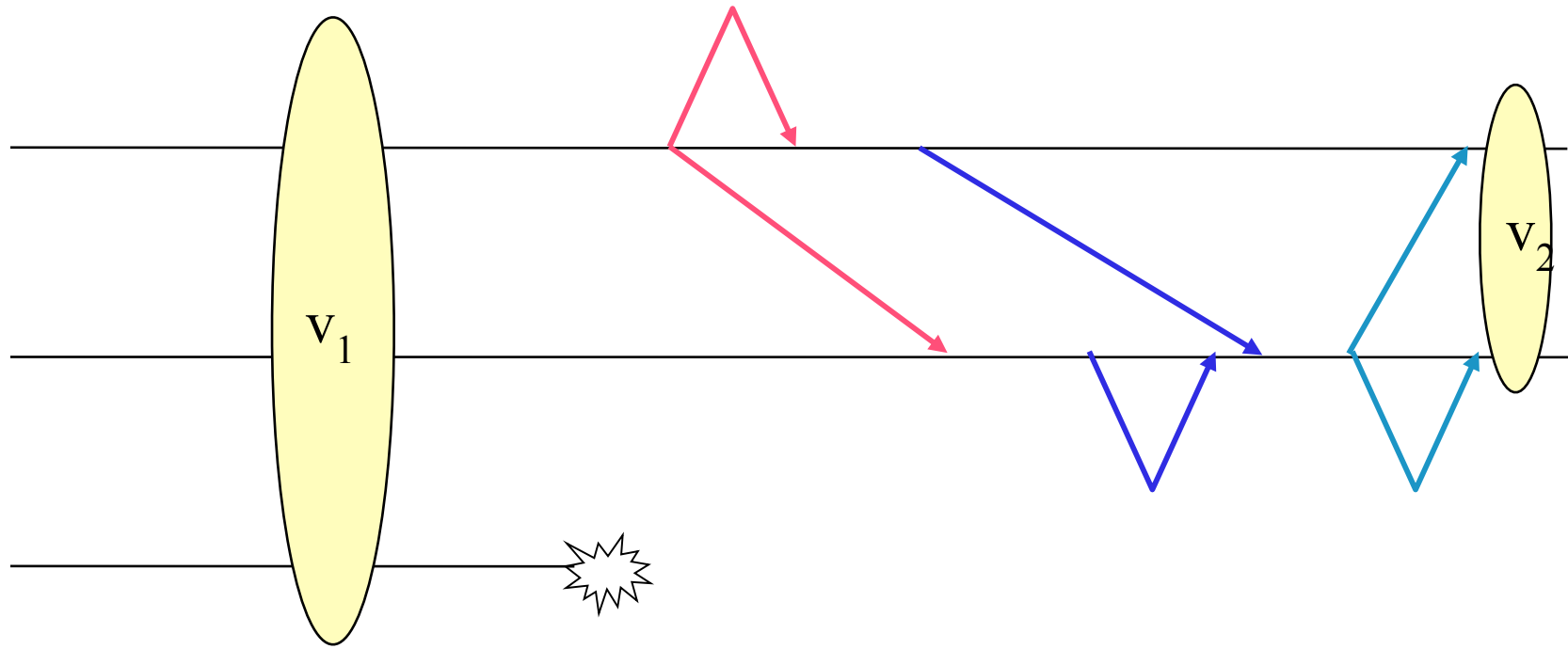


How a view change occurs

◆ The protocol (sketch)

- ◆ One of the replica manager observes a variation in its local perception of the group membership
 - ◆ Join/leave
 - ◆ Failure detector events
- ◆ The replica managers start a consensus protocol, trying to reach an agreement on composition/identifier of the next view
 - ◆ The protocol is normally based on a rotating coordinator
 - ◆ The new view may be the intersection of “local” views
- ◆ When an agreement is reached, the coordinator multicasts (reliably) the view to all members

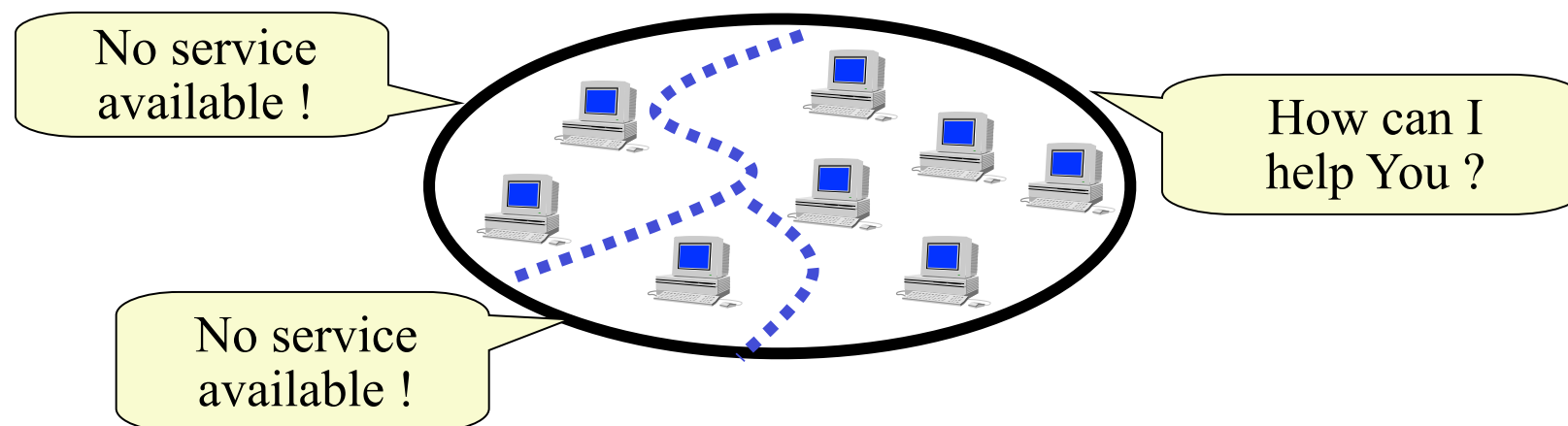
How a view change occurs



Primary Partition vs Partitionable

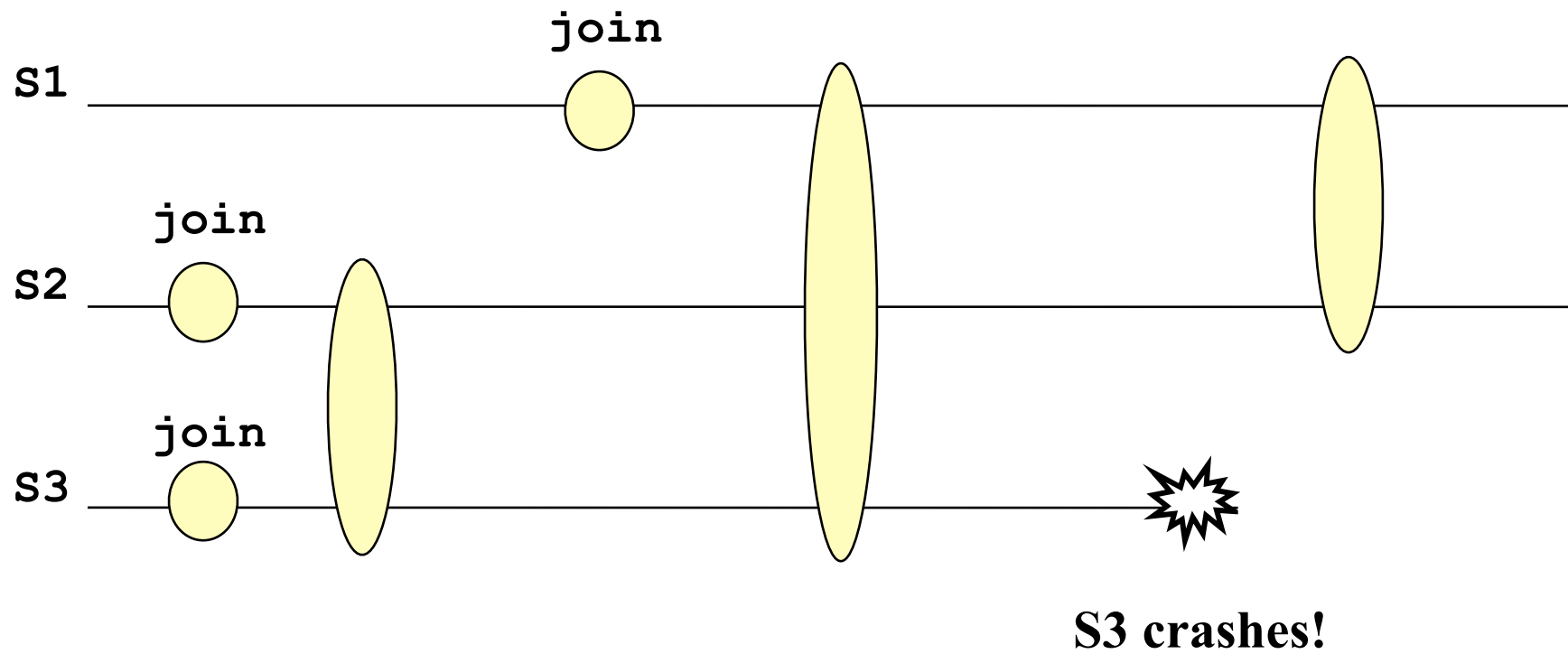
◆ Primary-Partition

- ◆ a **primary-partition** GCS attempts to maintain a single agreed view of the current membership of a group (the **primary** view)
- ◆ normally based on **majority** / **quorums**
- ◆ members outside this view are logically blocked even if they remain perfectly functional
- ◆ the primary partition may be lost temporarily



Primary Partition vs Partitionable

- ◆ A simple primary-partition scenario



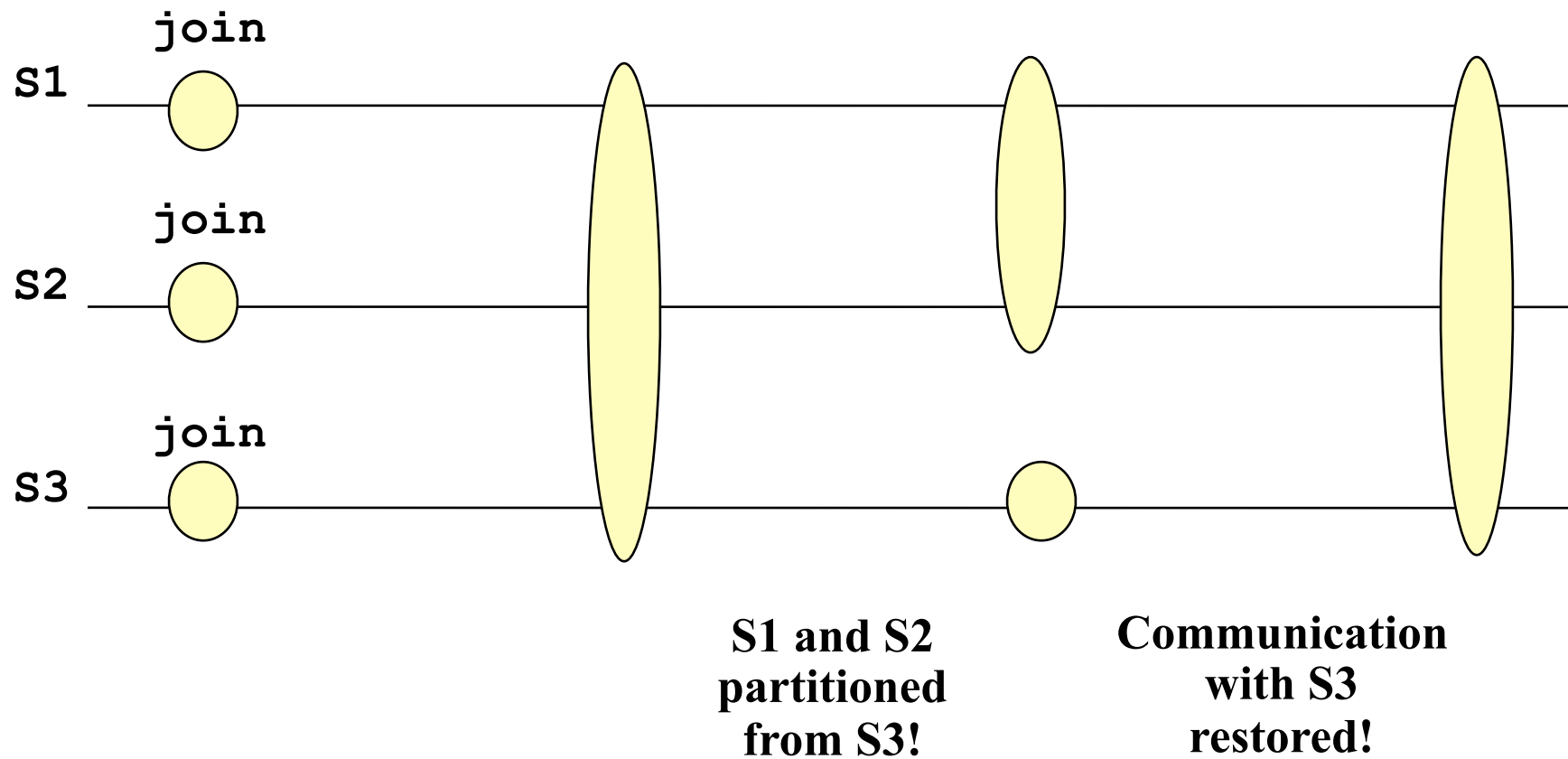
Primary-Partition vs Partitionable

- ◆ **The partitionable approach**

- ◆ A partitionable GCS allows multiple agreed views to co-exist in the system, each of them representing one of the partitions
- ◆ Members inside a view carry on the computation independently from members outside of it
- ◆ When two partitions merge, an application-dependent reconciliation protocol is needed

Primary-Partition vs Partitionable

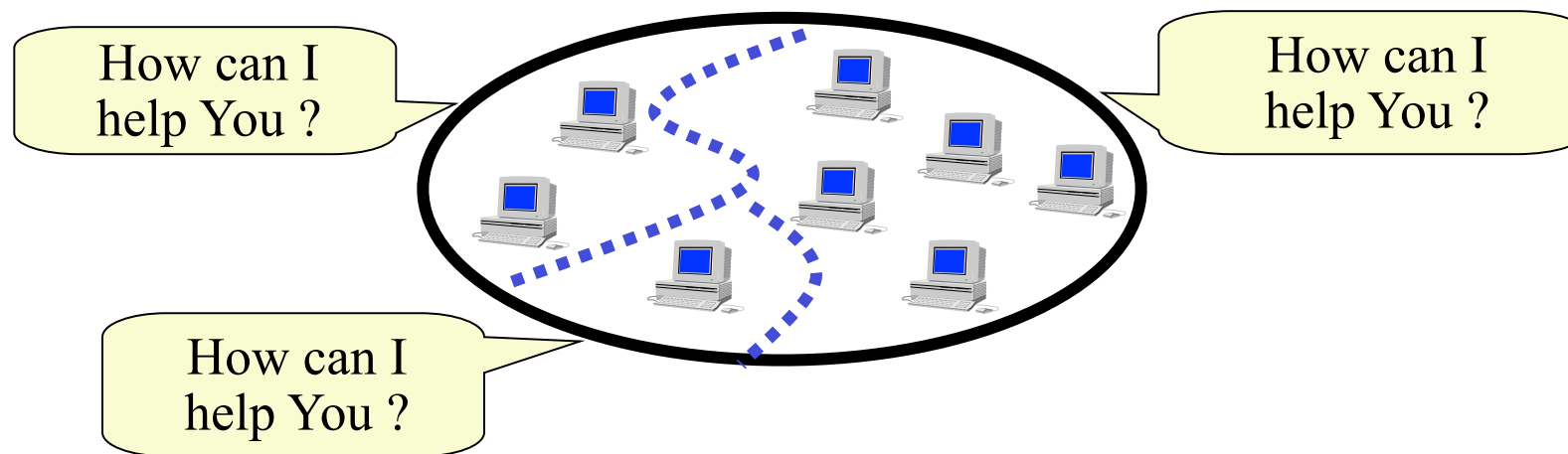
- ◆ A partitionable scenario



Primary Partition vs Partitionable

◆ Partitionable Group Communication

- ◆ Suitable for applications aware of the existence of partitions
- ◆ Partition-aware applications take advantage of their semantics in order to be available also in disconnected mode
- ◆ Application is carried on in all partitions of the system



Group Membership: Formal Specification

◆ Order

- ◆ If process p installs view $V(g)$ and then view $V'(g)$, then no other process $q \neq p$ installs $V'(g)$ before $V(g)$

◆ Integrity

- ◆ If a process p installs view $V(g)$, then $p \in V(g)$

◆ Non-triviality

- ◆ If process q joins the group and it becomes indefinitely reachable from process $p \neq q$, then eventually $q \in V'(g)$, for all $V'(g)$ installed by p
- ◆ If a process q becomes indefinitely unreachable from process p , then eventually $q \notin V'(g)$, for all $V'(g)$ installed by p

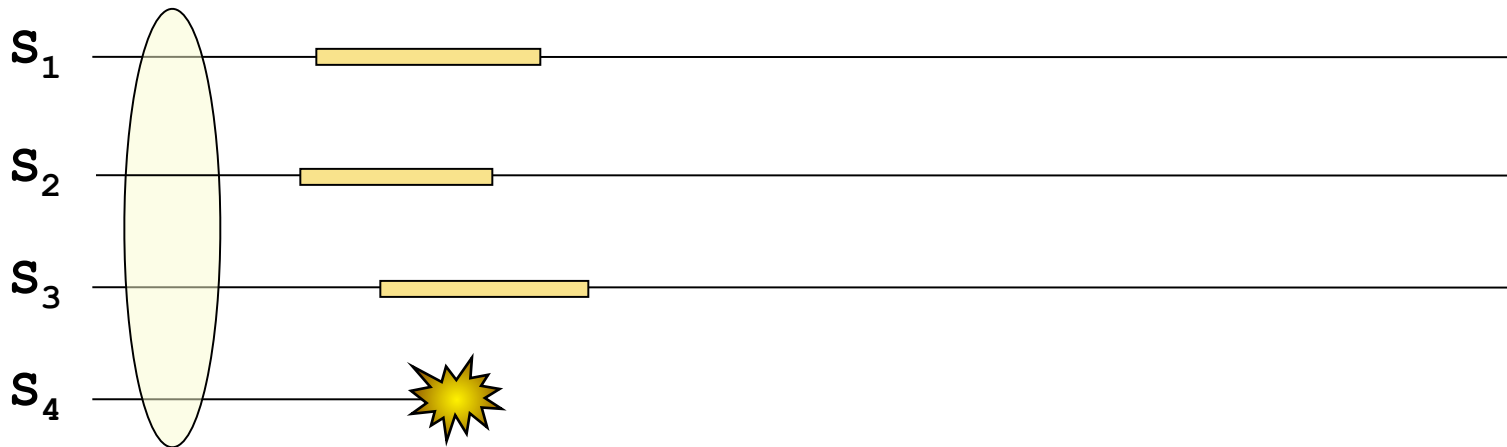
Group communication

◆ Task of a reliable multicast service

- ◆ to enable group members to communicate by msg multicasting
- ◆ to deliver messages to group members following various reliability and ordering semantics
- ◆ to interleave view installations with multicast message deliveries, providing a strong ordering guarantee for both called *view synchrony*

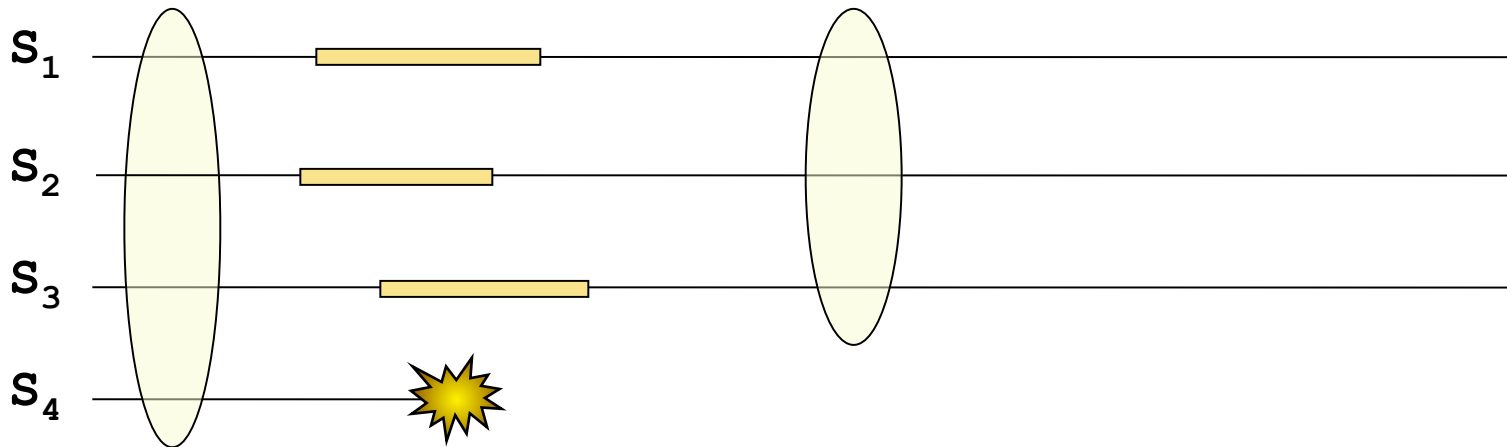
Reliable Multicast: Formal Specification

- ◆ **View synchrony (1)**
 - ◆ If a correct RM p delivers a message m during view $V(g)$, then all RMs within $V(g)$ will also deliver m , or p will install a new view
- ◆ **View synchrony does not admit executions like this:**



Reliable Multicast: Formal Specification

- ◆ **View synchrony (1)**
 - ◆ If a correct RM p delivers a message m during view $V(g)$, then all RMs within the view will also deliver m , or p will install a new view
- ◆ **View synchrony ~~does not admit~~ executions like this:
admits**

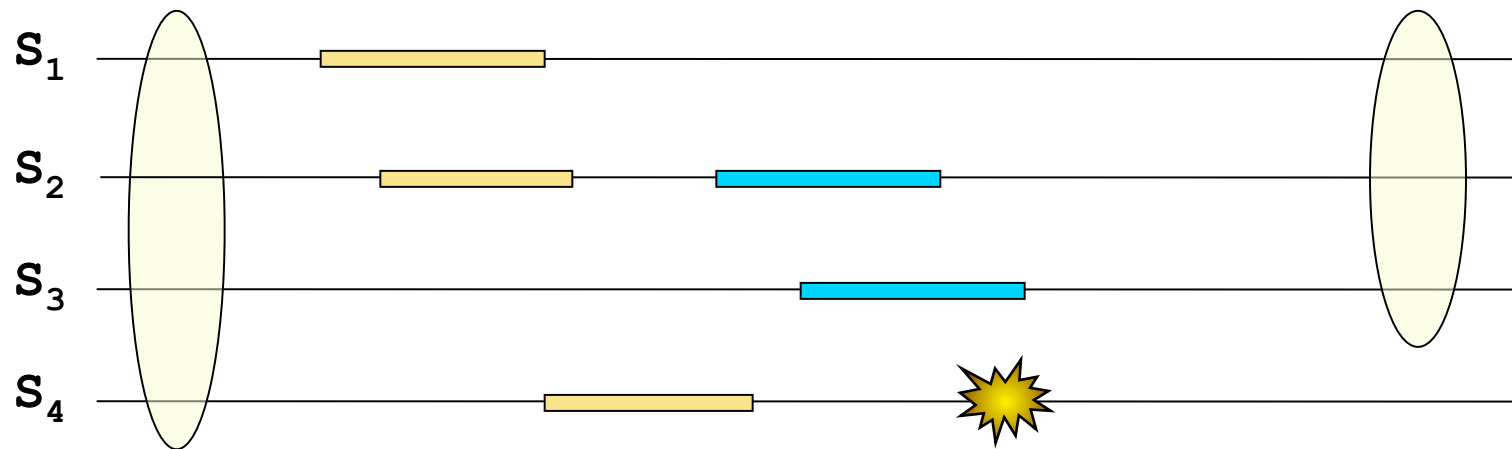


Reliable Multicast: Formal Specification

◆ View Synchrony (2)

- ◆ All servers that survive from one view to the same next view deliver the same set of messages in the original view

◆ View synchrony does not admit executions like this:

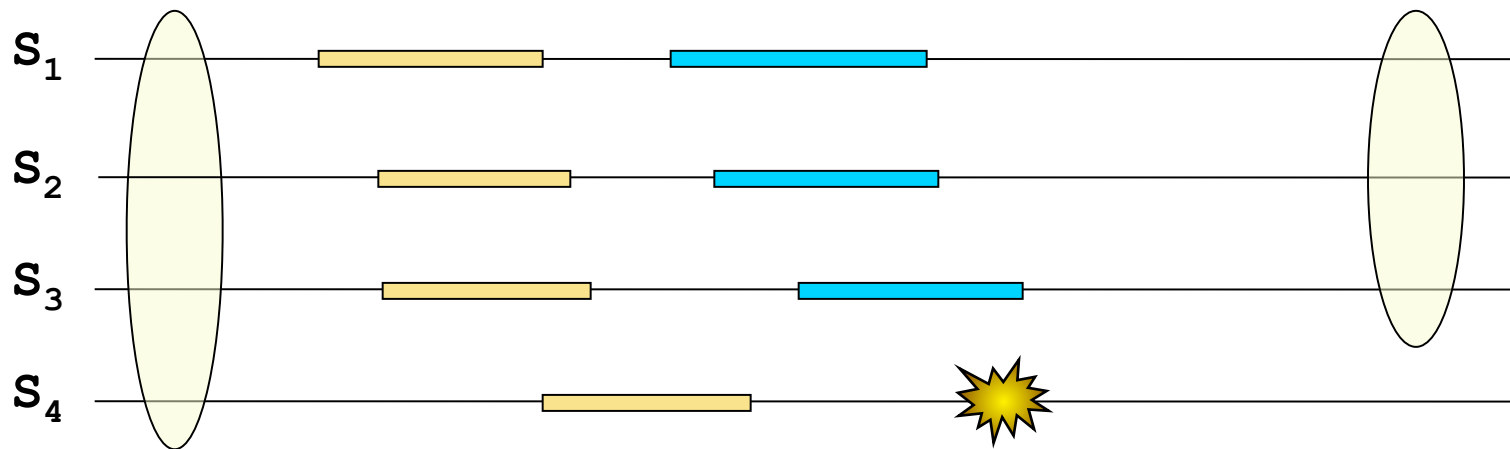


Reliable Multicast: Formal Specification

◆ View Synchrony (2)

- ◆ All servers that survive from one view to the same next view execute the same set of invocations in the original view

- ◆ ~~View synchrony does not admit executions like this:~~
admits



Reliable Multicast: Formal Specification

- ◆ **Integrity**

- ◆ A message m can be delivered by p only once, and only if the current view of p is the same view in which message m has been multicast

- ◆ **Validity**

- ◆ Correct processes always delivers the messages multicast by them

Group Communication: A brief history

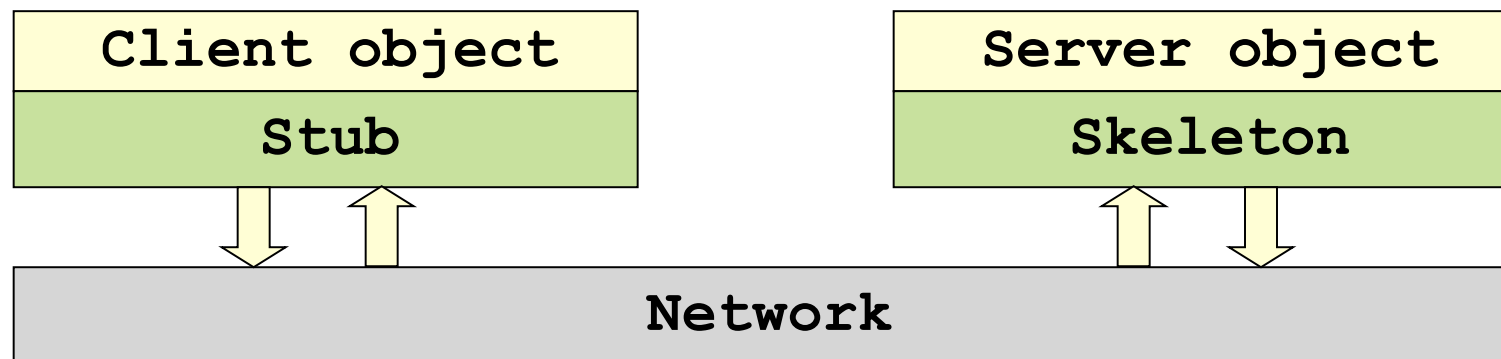
- ◆ **ISIS [Cornell, 1987-1993]**
 - ◆ First to introduce the concept of “*virtual synchrony*”
 - ◆ ISIS was also a company, their product used by Swiss banks
- ◆ **A lot of competition during the 90's**
 - ◆ Electra [Cornell, Zurich], Object Group Service [Lausanne], Eternal [Santa Barbara], Newtop [Newcastle], Filterfresh [Bell Labs], Transis [Hebrew University of Jerusalem], etc.etc.
- ◆ **Still active**
 - ◆ Jgroups [formerly Javagroups,]
 - ◆ Spread [John Hopkins Univ, then Spread Concepts LLC]
 - ◆ **Jgroup [Univ. of Bologna]**

Group Communication: Object Groups

- ◆ **Several programming environments supporting distributed programming have appeared (more or less) recently:**
 - ◆ CORBA (OMG)
 - ◆ Java RMI/J2EE (Sun)
 - ◆ .Net (Microsoft)
- ◆ **Characteristics:**
 - ◆ Object-oriented
 - ◆ Promote modularity, reusability, interoperability, portability
 - ◆ Based on client / server *remote method invocations*

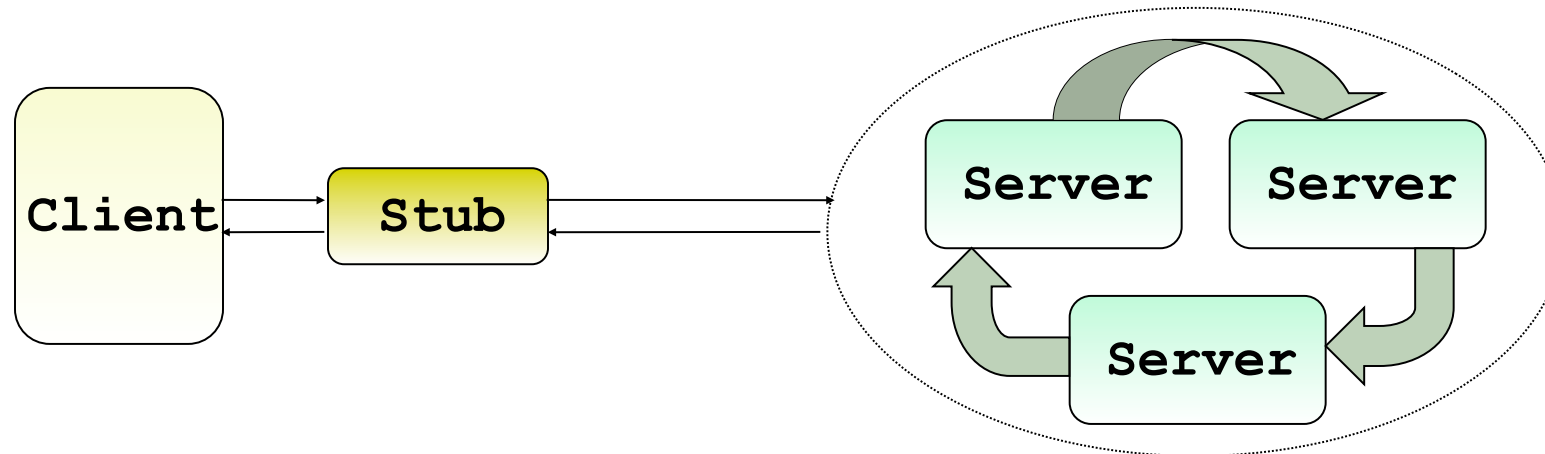
Remote Method Invocations

- ◆ Remote method invocations performed through local objects called *stubs*
 - ◆ stubs present the same interface as their remote counterparts
 - ◆ stubs interact with skeletons that dispatch method invocations to servers
 - ◆ stubs/skeletons handle all low-level details of method invocations



Group Method Invocations

- ◆ **Jgroup: an object-oriented middleware for the development of reliable and high-available applications in partitionable systems**
 - ◆ based on the object group paradigm
 - ◆ extends the Java distributed object model by supporting
 - ◆ group method invocations
 - ◆ is completely written in Java, and thus inherits its portability

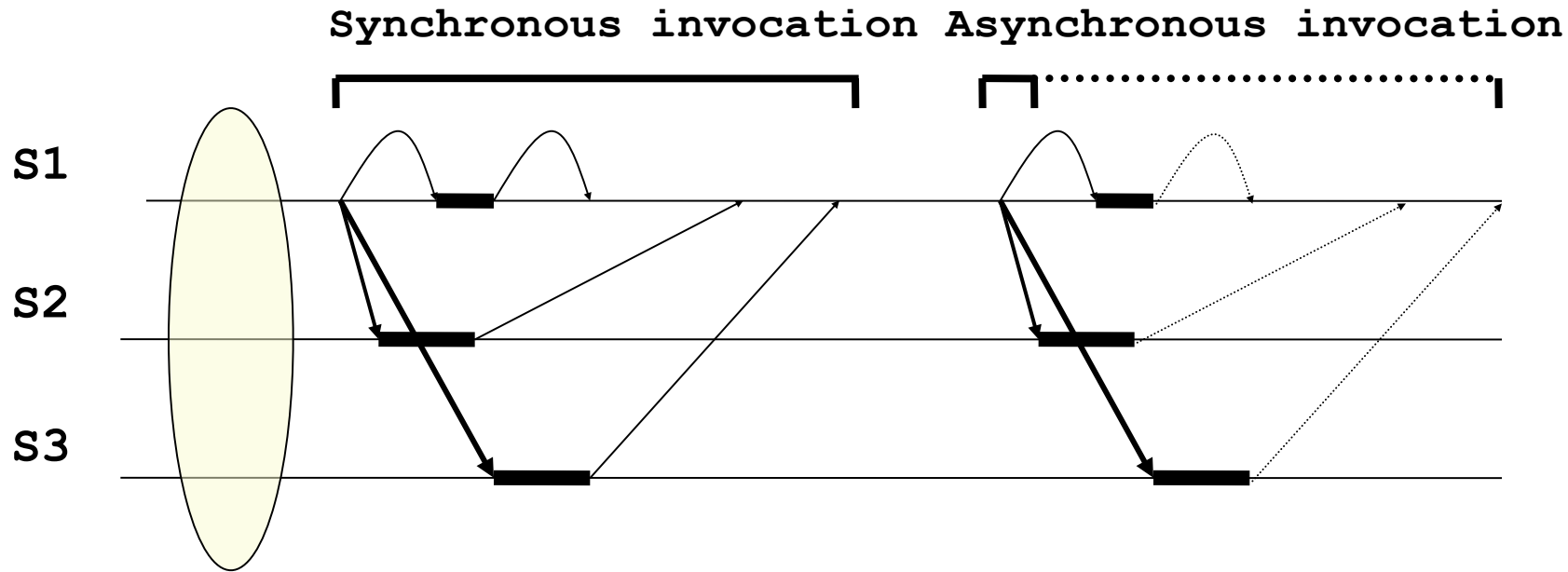


Full Object Orientation

Group method invocations:

- ◆ **Internal group method invocations:**
 - ◆ those performed among group servers
- ◆ **External group method invocations:**
 - ◆ those performed between clients and servers
- ◆ **A uniform, object-oriented model for both service providers and service consumers**

Internal Invocations



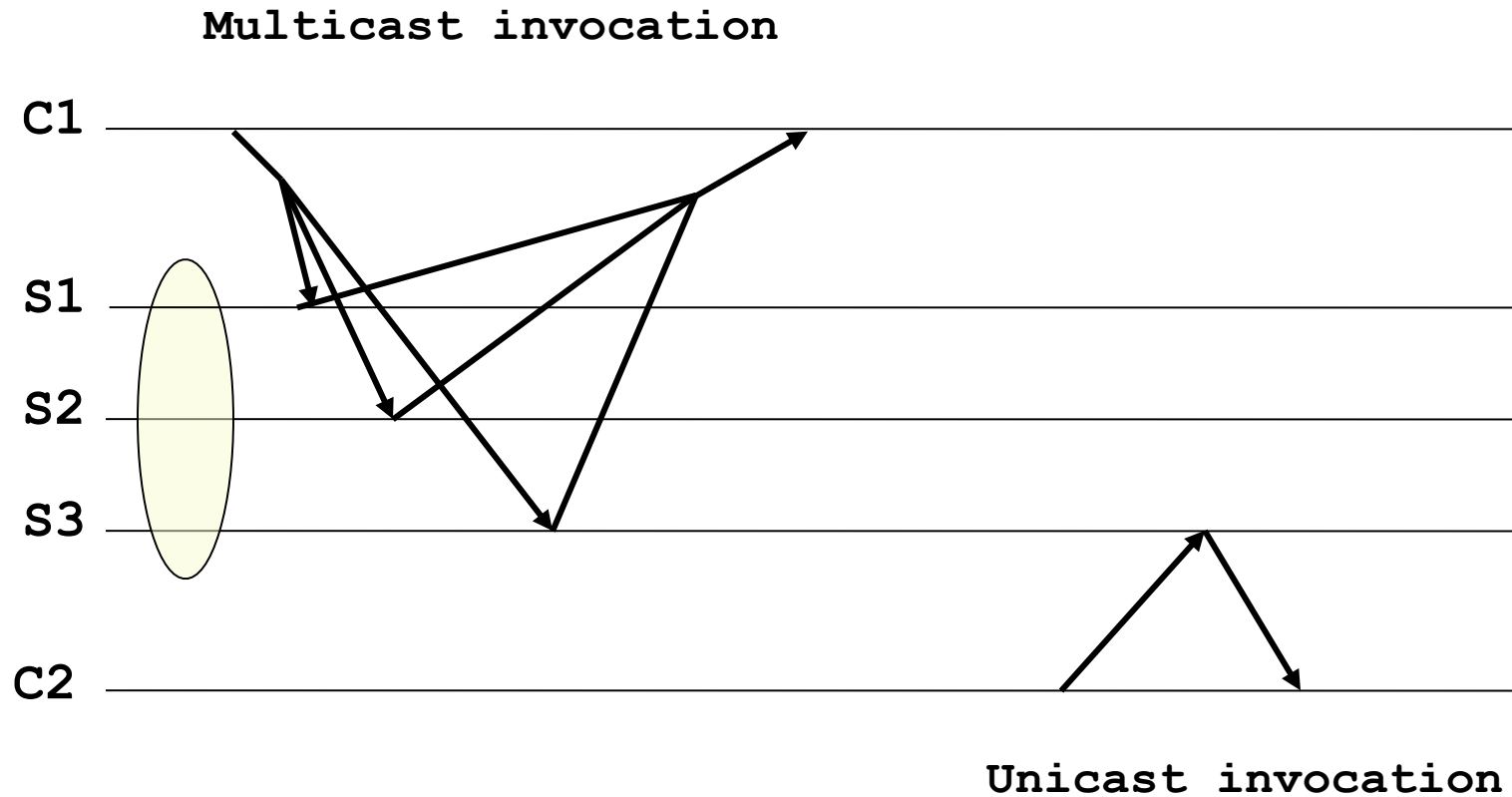
```
int[] array = gm.getValue();  
total = sum(array);
```

```
public int getValue() {  
    return value;  
}
```

```
SumCallback c;  
gm.getValue(c);
```

```
public class SumCallback  
implements Callback {  
    void retValue(int[] array) {  
        total = sum(array);  
    }  
}
```

External Invocations



```
registry.bind("S1", remote); registry.lookup("S1");
```

Passive Replication through Group Communication

- ◆ **Request:**
 - ◆ A front-end issues the request, containing a unique identifier, to the primary RM
- ◆ **Coordination:**
 - ◆ The primary checks the unique id; if already done, it re-sends the response
 - ◆ The primary performs each request atomically, in the order in which it receives it relative to other requests
- ◆ **Execution:**
 - ◆ The primary executes the request and stores the response.

Passive Replication through Group Communication

- ◆ **Agreement:**

- ◆ If the request is an update, the primary sends the updated state, the response and the unique identifier to all the backups. The backups send an acknowledgement.

- ◆ **Response:**

- ◆ The primary responds to the front-end, which hands the response back to the client

Passive Replication through Group Communication

- ◆ **If the primary does not fail, the system easily implements linearizability**
- ◆ **If the primary fails, the system implements linearizability if and only if a single backup takes off exactly where the primary left off**
 - ◆ The primary is replaced by a unique backup
 - ◆ Surviving RMs agree which operations had been performed at take over
- ◆ **Can be done through primary-partition group communication**
 - ◆ Communication primary-backups is done through multicast
 - ◆ The new primary is elected through a deterministic function over the current view

Passive replication: discussion

- ◆ **To survive f process crashes, $f+1$ RMs are required**
- ◆ **To design passive replication that is linearizable**
 - ◆ View synchronous communication has relatively large overheads
 - ◆ Several rounds of messages per multicast
 - ◆ After failure of primary, there is latency due to delivery of group view
- ◆ **Variant in which clients can read from backups**
 - ◆ Which reduces the work for the primary
 - ◆ Get sequential consistency but not linearizability

Active Replication through Group Communication

- ◆ **Request**
 - ◆ FE attaches a unique id and uses totally ordered reliable multicast to send request to RMs.
 - ◆ FE can at worst, crash. It does not issue requests in parallel
- ◆ **Coordination**
 - ◆ The multicast service delivers requests to all the RMs in the same (total) order.
- ◆ **Execution**
 - ◆ Every RM executes the request. They are state machines and receive requests in the same order, so the effects are identical. The id is put in the response

Active Replication through Group Communication

◆ 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

- ◆ FEs collect responses from RMs.
- ◆ FE may just use one or more responses.
- ◆ If it is only trying to tolerate crash failures, it gives the client the first response.

Active Replication through Group Communication

- ◆ **This system implements sequential consistency**
 - ◆ Due to reliable totally ordered multicast, the RMs collectively do the same as a single copy would do
 - ◆ It works in a synchronous system
 - ◆ In an asynchronous system atomic multicast is impossible – but failure detectors can be used to work around this problem.
- ◆ **This replication scheme is not linearizable**
 - ◆ Because total order is not necessarily the same as real-time order
- ◆ **To improve performance**
 - ◆ FEs send read-only requests to just one RM