Group Communication: from practice to theory

André Schiper
EPFL, Lausanne, Switzerland

Outline

- Introduction to fault tolerance
- Replication for fault tolerance
- Group communication for replication
- Implementation of group communication

Practical issues

Theoretical issues

Introduction

- Computer systems become every day more complex
- Probability of faults increases
- Need to develop techniques to achieve fault tolerance

Introduction (2)

Fault tolerant techniques developed over the year

- Transactions (all or nothing property)
- Checkpointing (prevents state loss)
- Replication (masks faults)

Transactions

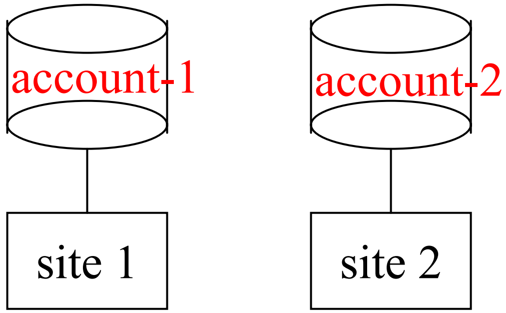
- ACID properties (Atomicity, Consistency, Isolation, Durability)

<u>begin-transaction</u>

remove (100, account-1)

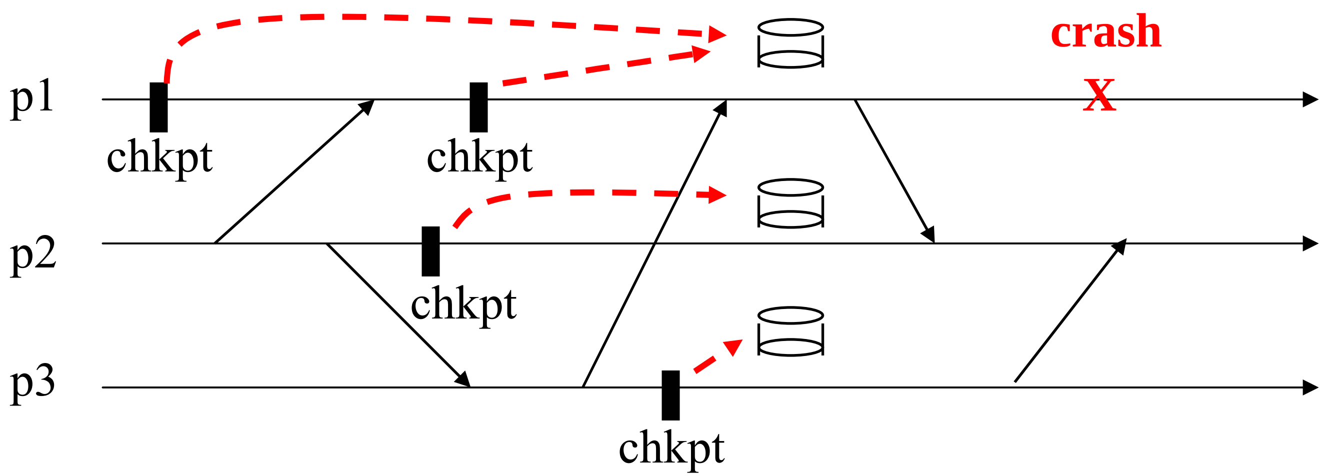
deposit (100, account-2)

<u>end-transaction</u>



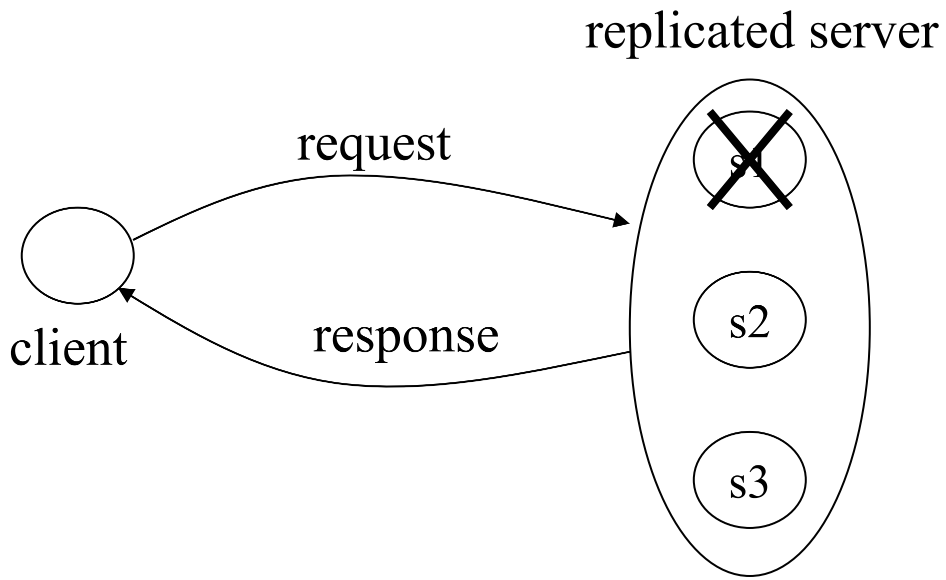
Crash during the transaction: rollback to the state before the beginning of the transaction

Checkpointing



Crash of p1: rollback of p1 to the latest saved state

Replication



- Crash of s1 is masked to the client
- Upon recovery of s1: state transfer to bring s1 up-to-date

Introduction to FT (6)

Comparison of the three techniques

- Only replication masks crashes (i.e., ensures high availability)
- Transactions and checkpointing: progress is only possible after recovery of the crashed process

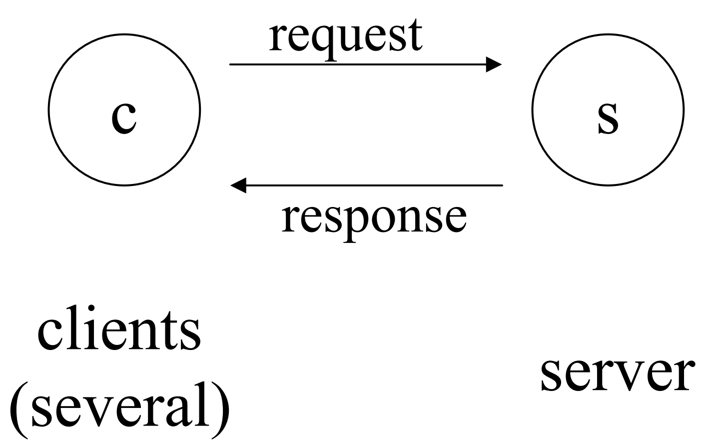
We will concentrate on replication

Outline

- Introduction to fault tolerance
- Replication for fault tolerance
- Group communication for replication
- Implementation of group communication

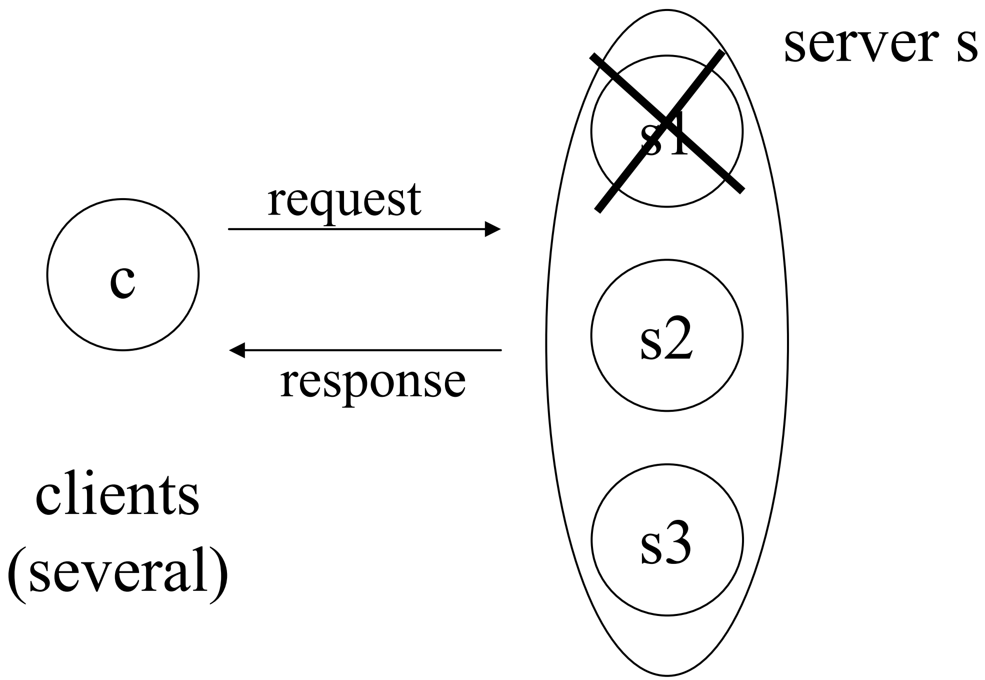
Context

Replication in the general context of client-server interaction:



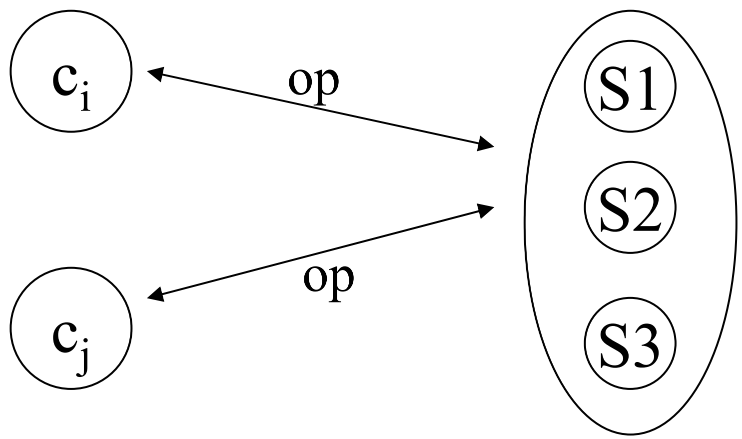
Context (2)

- Fault tolerance by replicating the server



Correctness criterion: linearizability

- Need to keep the replica consistent
- Consistency defined in terms of the operations executed by the clients



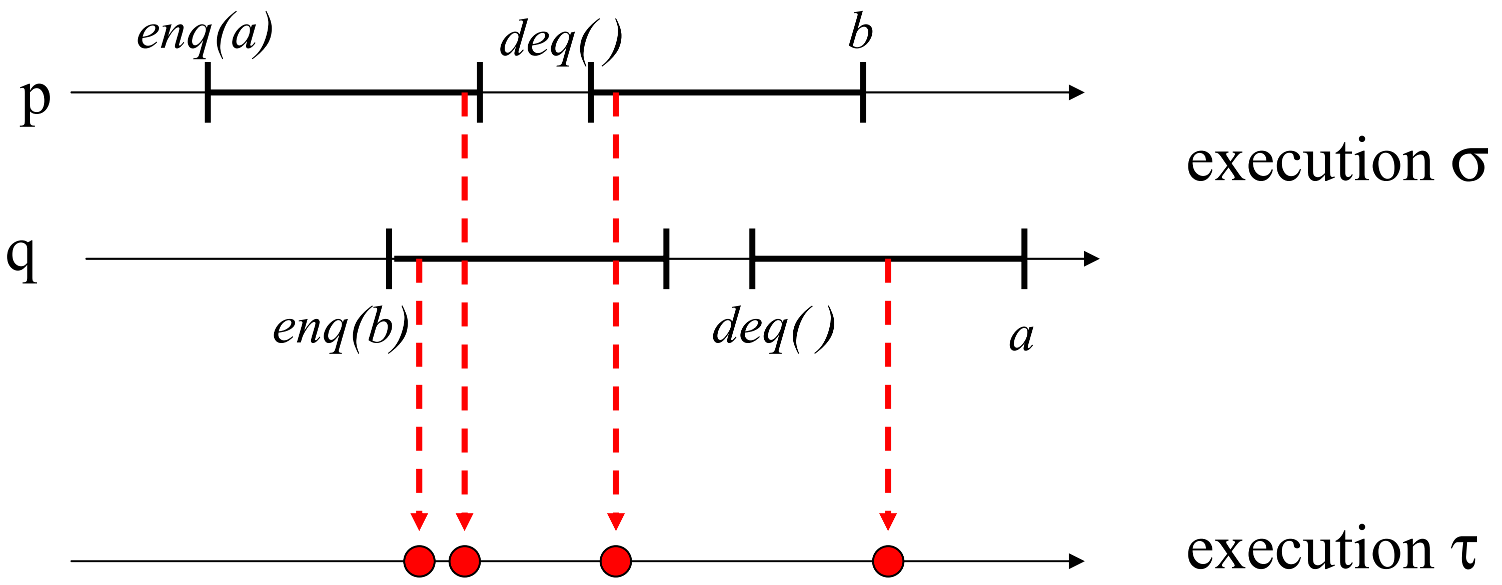
Correctness criterion: linearizability (2)

Example:

- Server: FIFO queue
- Operations: enqueue/dequeue

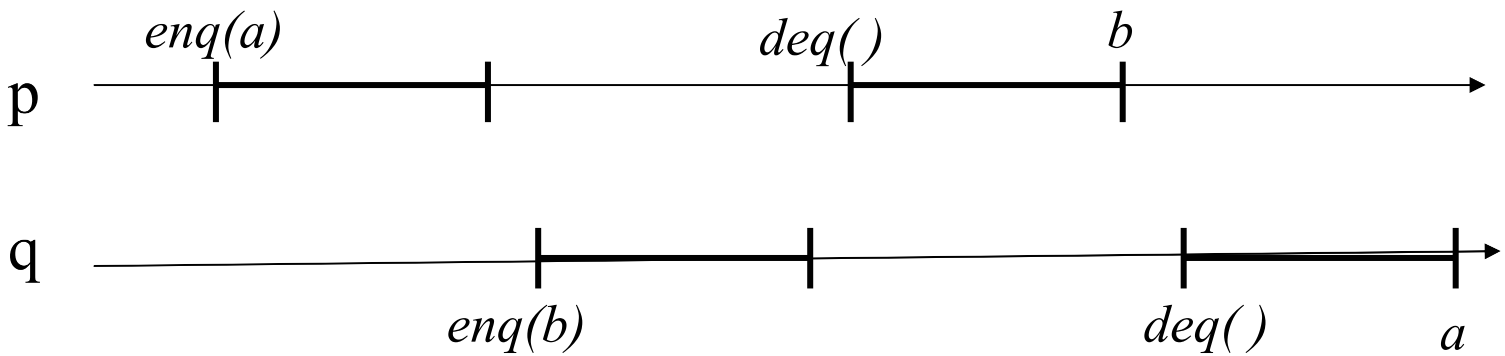
Correctness criterion: linearizability (3)

Example 1 linearizable execution



Correctness criterion: linearizability (4)

Example 2 non linearizable execution

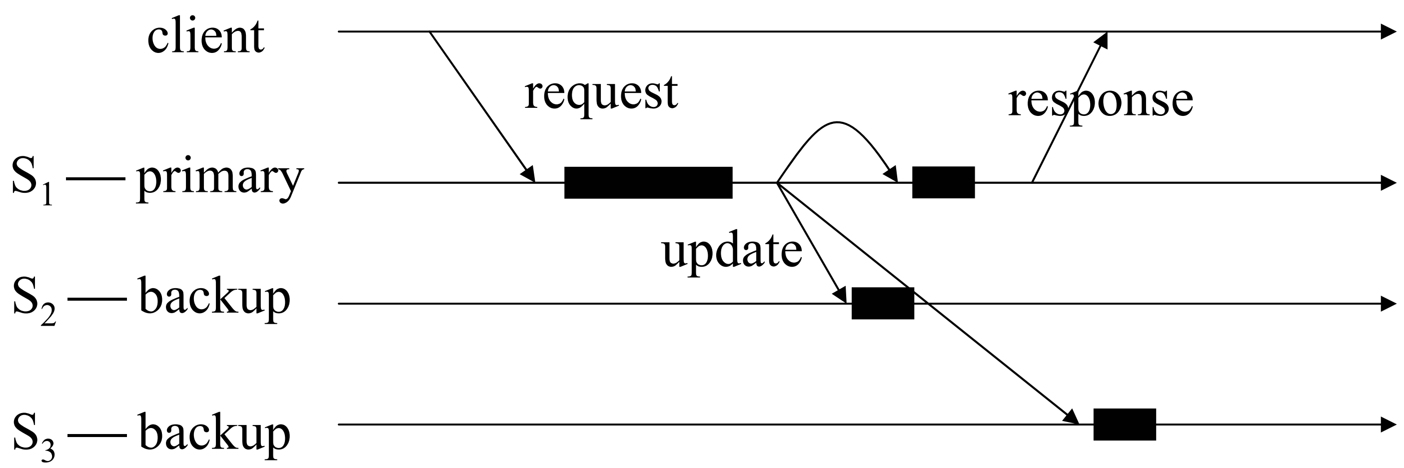


Replication techniques

Two main replication techniques for linearizability

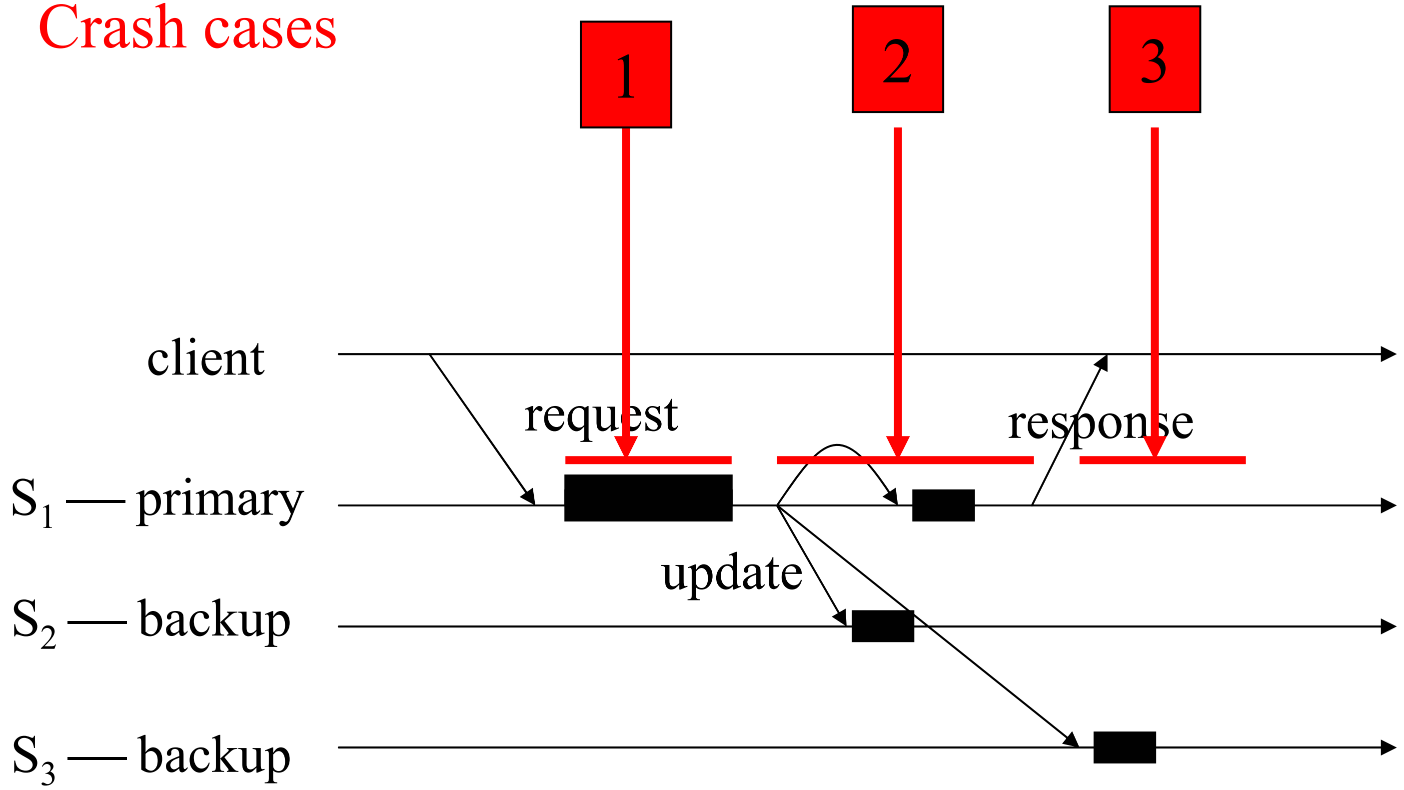
- Primary-backup replication (or *passive replication*)
- Active replication (or *state machine replication*)

Primary-backup replication



Primary-backup replication

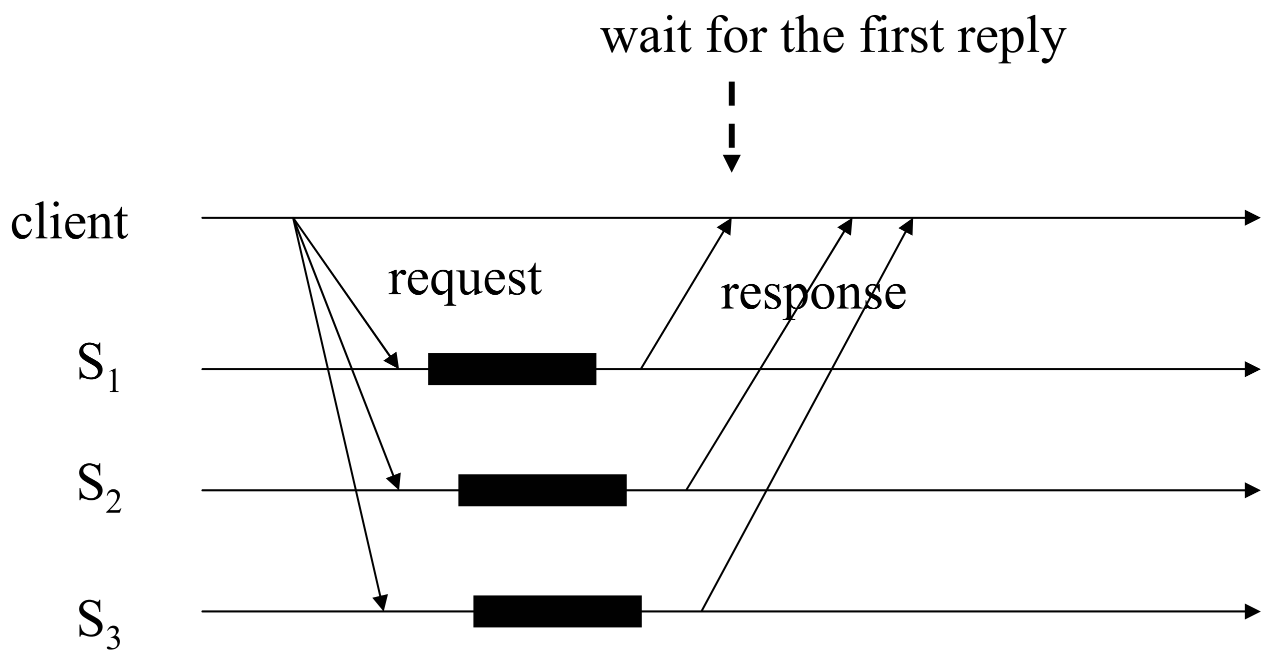
- Crash cases



Primary-backup replication (3)

- Crash detection usually based on time-outs
- Time-outs may lead to incorrectly suspect the crash of a process
- The technique must work correctly even if processes are incorrectly suspected to have crashed

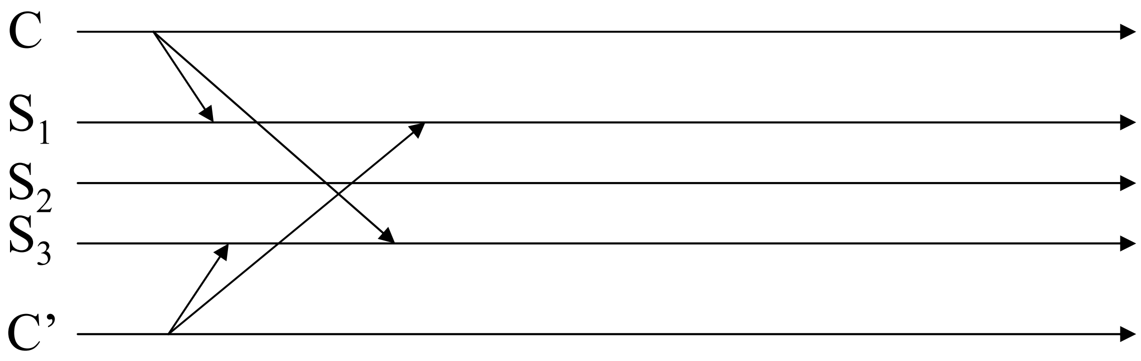
Active replication



Crash of a server replica transparent to the client

Active replication (2)

- If more than one client, the requests must be received by all the replicas in the same order



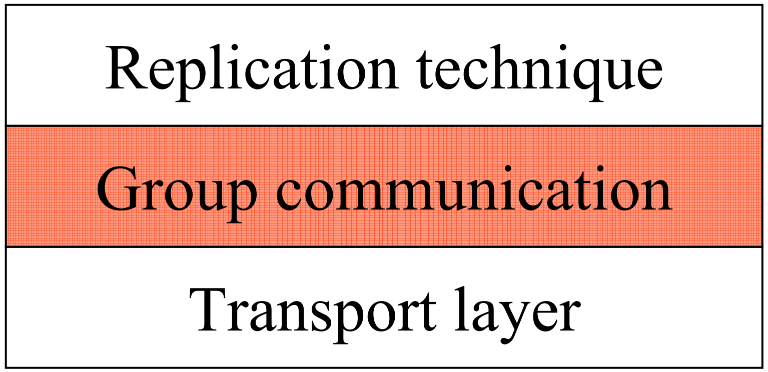
- Ensured by a communication primitive called total order broadcast (or atomic broadcast)
- Complexity of active replication hidden in the implementation this primitive

Outline

- Introduction to fault tolerance
- Replication for fault tolerance
- Group communication for replication
- Implementation of group communication

Role of group communication

- Active replication: requires communication primitive that orders client requests
- Passive replication: have shown the issues to be addressed
- Group communication: communication infrastructure that provides solutions to these problems



Role of group communication (2)

Let g be a group with members p, q, r

- multicast(g, m): allows m to be multicast to g without knowing the membership of g
- IP-multicast (UDP) also provides such a feature
- Group communication provides stronger guarantees (reliability, order, etc.)

Role of group communication (3)

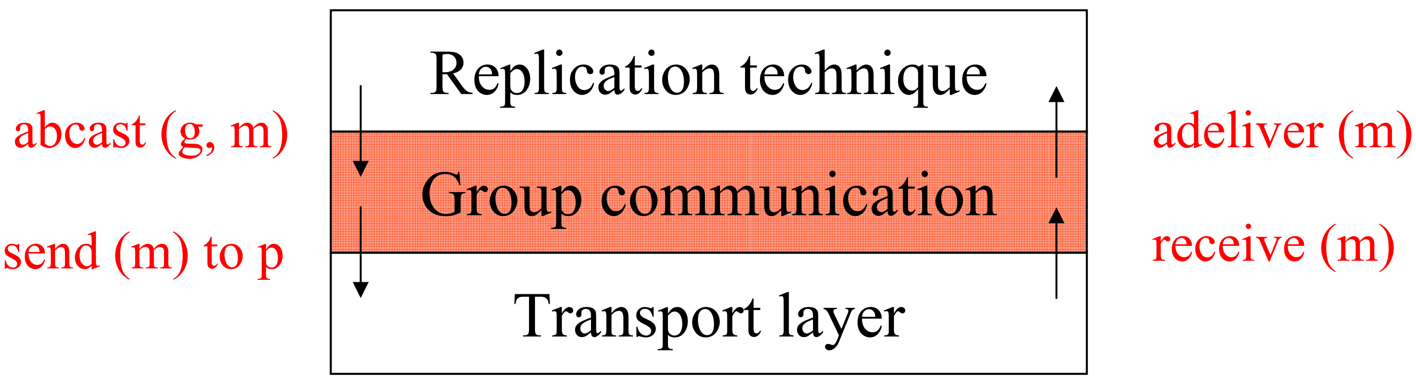
We will discuss:

- Group communication for active replication
- Group communication for passive replication

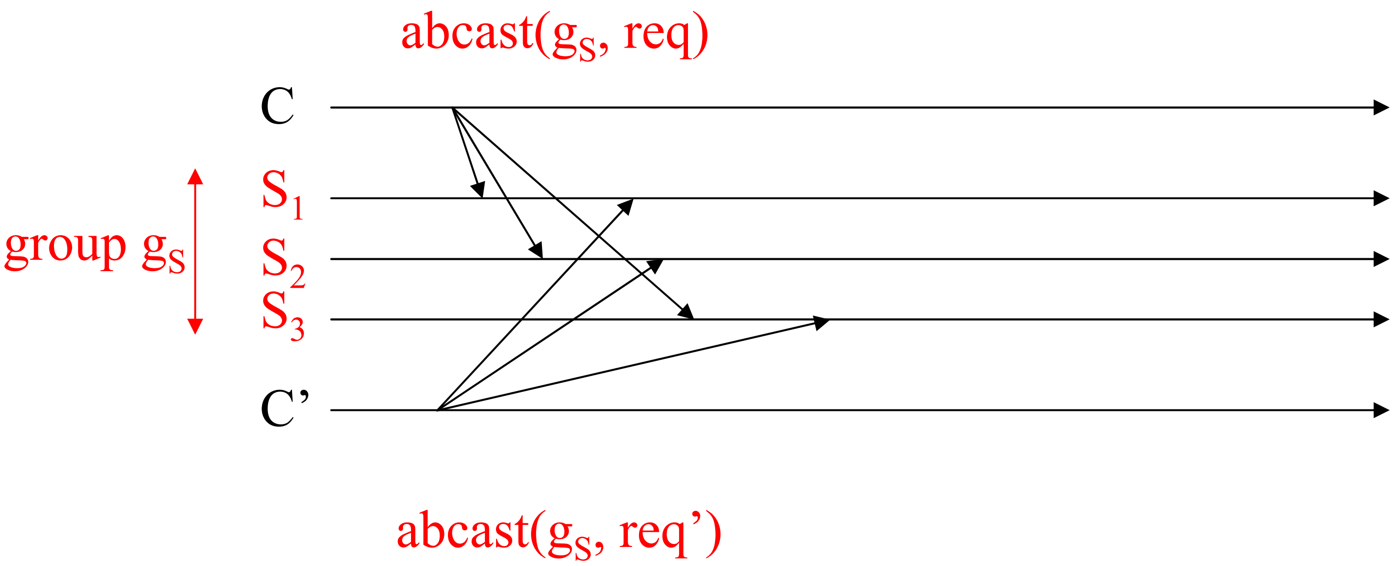
Atomic broadcast for active replication

Group communication primitive for active replication:

- atomic broadcast (denoted sometimes abcast)
- also called total order broadcast



Atomic broadcast for active replication (2)



Atomic broadcast: specification

- If p executes abcast(g, m) and does not crash, then all processes in g eventually adeliver m
- If some process in g adelivers m and does not crash, then all processes in g that do not crash eventually adeliver m
- If p, q in g adeliver m and m', then they adeliver them in the same order

Role of group communication (3)

We will discuss:

- Group communication for active replication
- Group communication for passive replication

Generic broadcast for passive replication

- Atomic broadcast can also be used to implement passive replication
- Better solution: generic broadcast
- Same spec as atomic broadcast, except that not all messages are ordered:
  - Generic broadcast based on a conflict relation on the messages
  - Conflicting messages are ordered, non conflicting messages are not ordered

Generic broadcast for passive replication (2)

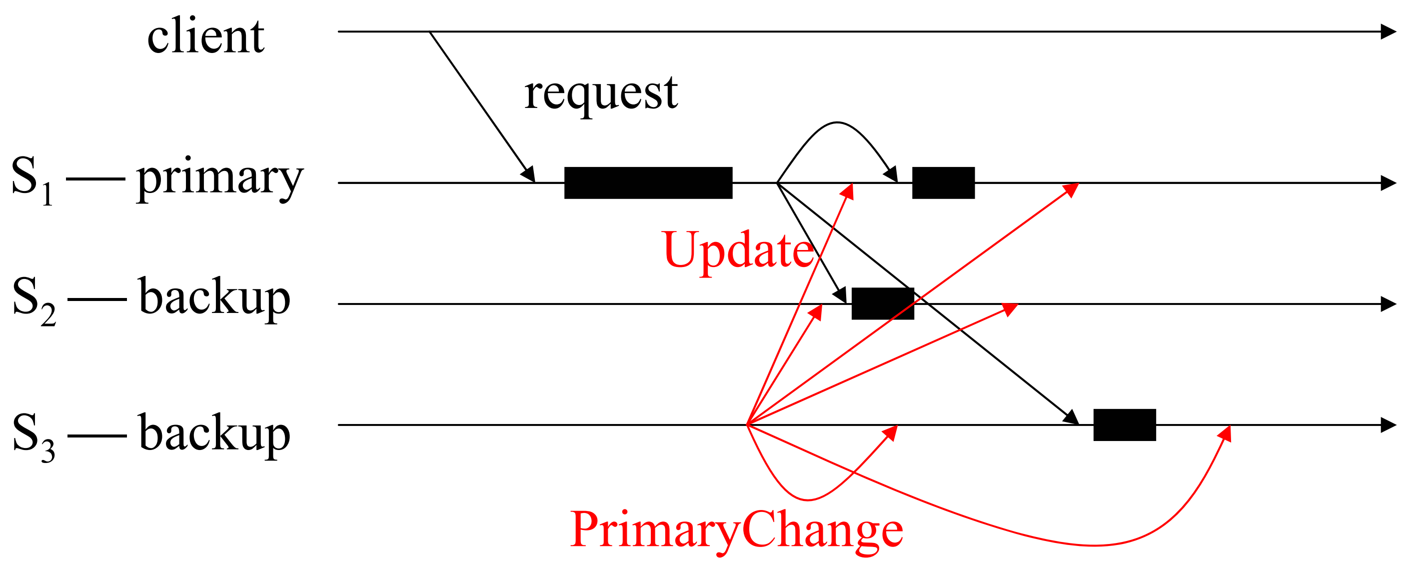
- Two types of messages:
  - *Update*
  - *PrimaryChange*: issued if a process suspects the current primary

    Upon delivery: cyclic permutation of process list

    New primary: process at the head of the process list
- Conflict relation:

	PrimaryChange	*Update*
PrimaryChange	no conflict	conflict
Update	conflict	conflict

Generic broadcast for passive replication (3)



Generic broadcast for passive replication (4)

Another way to view the strategy:

- Replicas numbered 0 .. n-1
- Computation divided into rounds
- In round r, the primary is the process with number r mod n
- When process p suspects the primary of round r, it broadcasts newRound (= primaryChange)

Conflict relation	*newRound*	*Update*
newRound	no conflict	conflict
Update	conflict	conflict

All processes deliver *Update* in the same round

Other group communication issues

To discuss:

- Static *vs.* dynamic group
- Crash-stop *vs.* crash-recovery model

Static *vs*. dynamic groups

- Static group: group whose membership does not change during the lifetime of the system
- A static group is sometimes too limitative from a practical point of view: may want to replace a crashed replica with a new replica
- Dynamic group: group whose membership changes during the lifetime of the system
- Dynamic group requires to address two problems:
  - How to add / remove processes from the group (group membership problem)
  - Semantics of communication primitives

Group membership problem

- Need to distinguish
  - (1) group
  - (2) membership of the group at time t

- New notion: view of group g defined by: (i, s)
  - i identifies the view
  - s: membership of view i (set of processes)
  - membership i also denoted v_i

Group membership problem (2)

Changing the membership of a group:

- add (g, p): for adding p to group g
- remove (g, p): for removing p from group g

Usual requirement: all the members of a group see the same sequence of views:

if (i, s_p) *the view i of p*

and (i, s_q) *the view i of q*

then $s_p = s_q$

Process recovery

- Usual theoretical model: crash-stop
  - processes do not have access to stable storage (disk)
  - if a process crashes, its whole state is lost (i.e. no recovery possible)
- Drawback of the static crash-stop model: if non null crash probability, eventually all processes will have crashed
- Drawback of the dynamic crash-stop model: not tolerant to catastrophic failures (crash of all the processes in a group)
- To tolerate catastrophic failures: crash-recovery model

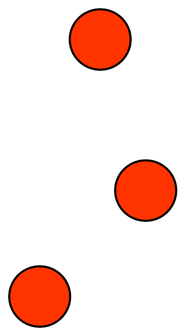
Combining the different GC models

Combining static/dynamic with crash-stop/crash-recovery:

1. Static groups in the crash-stop model
   Considered in most theoretical papers
2. Dynamic groups in the crash-stop model
   Considered in most existing GC systems
3. Static groups in the crash-recovery model
4. Dynamic groups in the crash-recovery model

Quorum systems *vs.* group communication

- Quorum system: context of read/write operations
- Typical situation:
  - Access a majority of replicas to perform a read operation
  - Access a majority of replicas to perform a write operation



Quorum systems *vs.* group communication

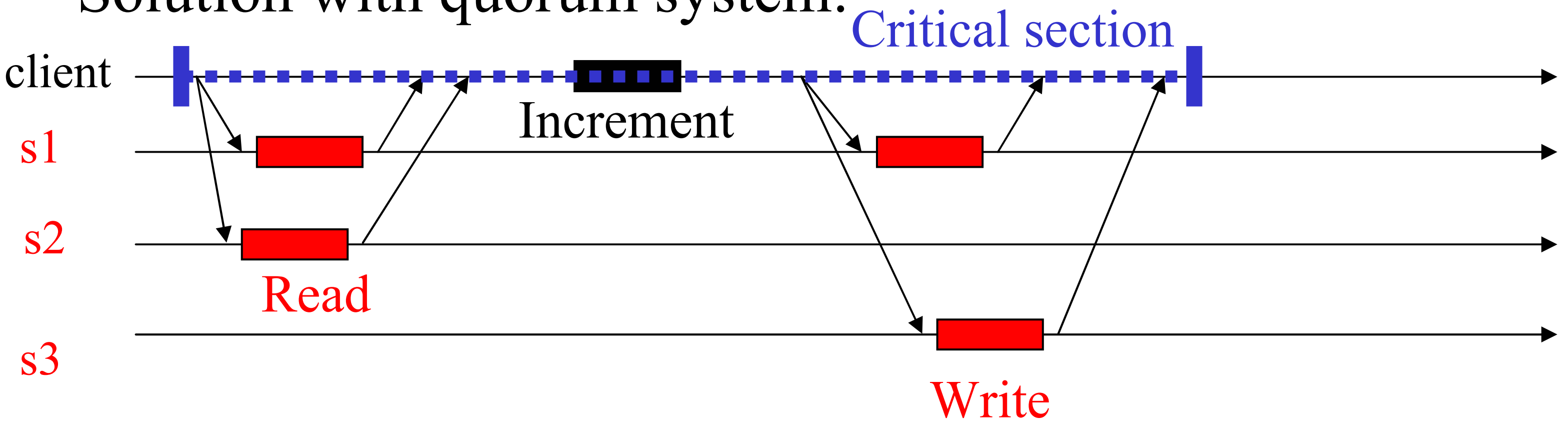
Example:

- a replicated server managing an integer with read/write operations (called register)

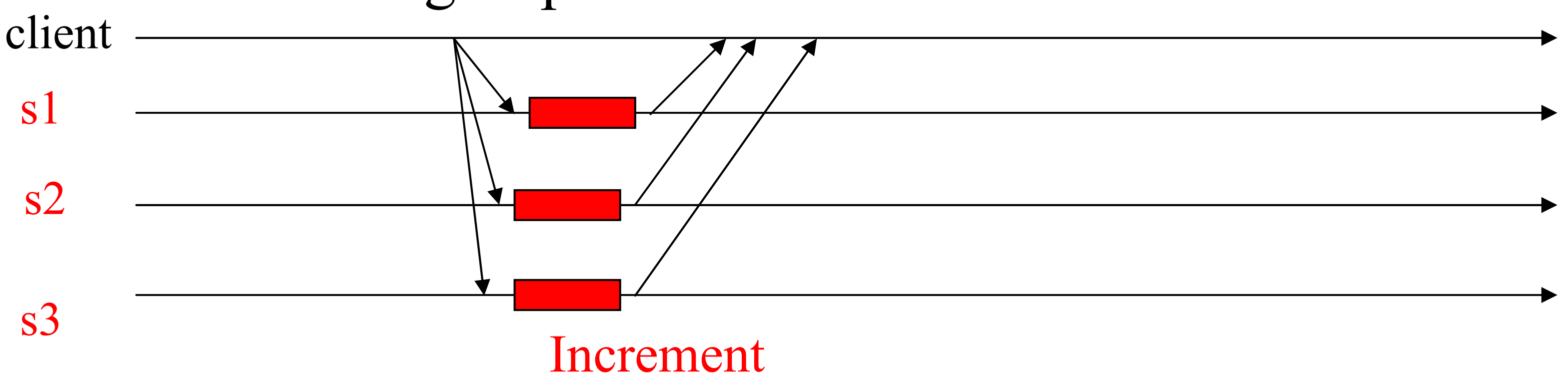
- operation to perform: increment

Quorum systems *vs.* group communication (2)

- Solution with quorum system:



- Solution with group communication



Outline

- Introduction to fault tolerance
- Replication for fault tolerance
- Group communication for replication
- Implementation of group communication

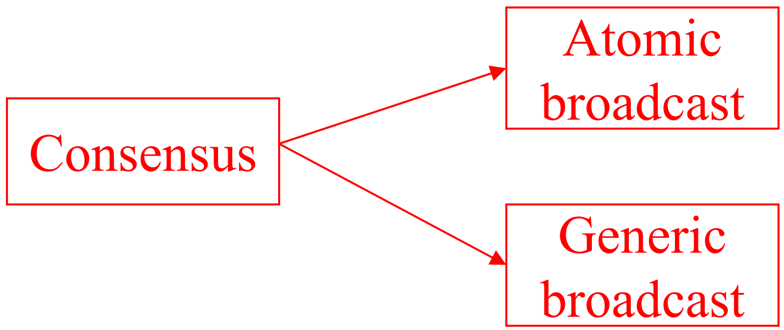
Implementation of group communication

Context:

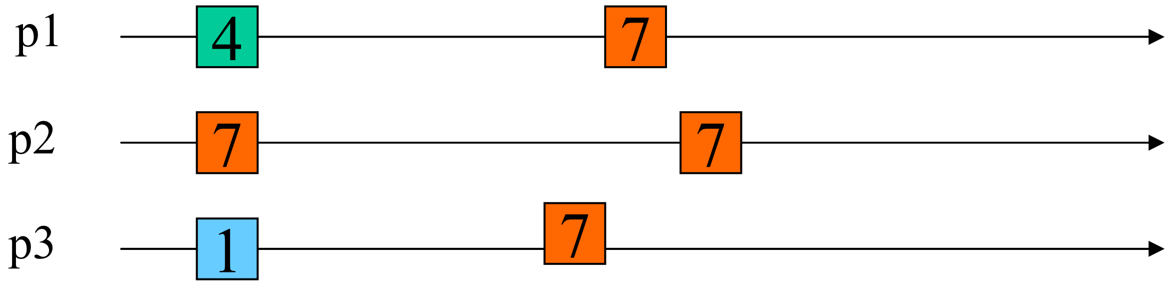
- Static groups
- Crash-stop model
- Non-malicious processes

- Atomic broadcast
- Generic broadcast

↕ *Common denominator:* Consensus



Consensus (informal)



Consensus (formal)

A set Π of processes, each $p_i \in \Pi$ with an initial value v_i

Three properties:

Validity: If a process decides v, then v is the initial value of some process

Agreement: No two processes decide differently

Termination: Every process eventually decides some value

Impossibility result

Asynchronous system:

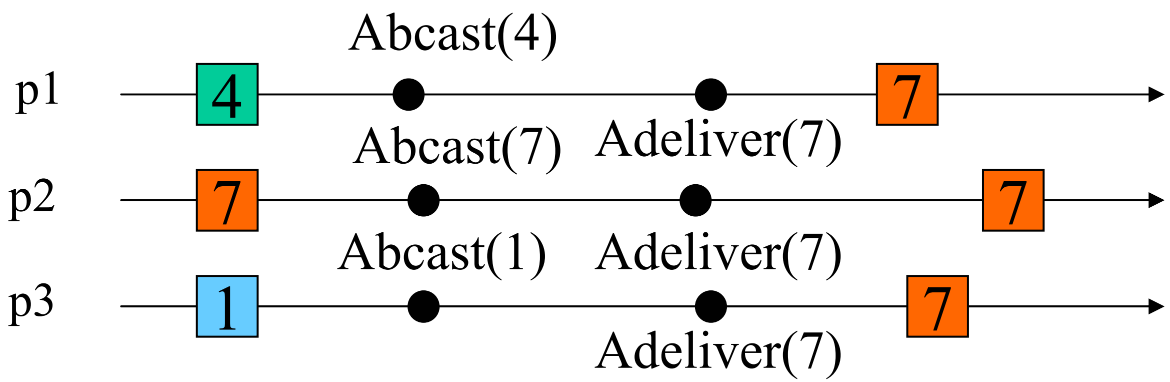
- No bound on the message transmission delay
- No bound on the process relative speeds

Fischer-Lynch-Paterson impossibility result (1985):

Consensus is not solvable in an asynchronous system with a deterministic algorithm and reliable links if one single process may crash

Impossibility of atomic broadcast

- By contradiction



Models for solving consensus

Synchronous system:

- There is a known bound on the transmission delay of messages
- There is a known bound on the process relative speeds

Consensus solvable with up to n-1 faulty processes

Drawback:

- Requires to be pessimistic (large bounds)
- Large bounds lead to a large blackout period in case of a crash

Models for solving consensus (2)

Partially synchronous system (Dwork, Lynch, Stockmeyer, 1988)

Two variants:

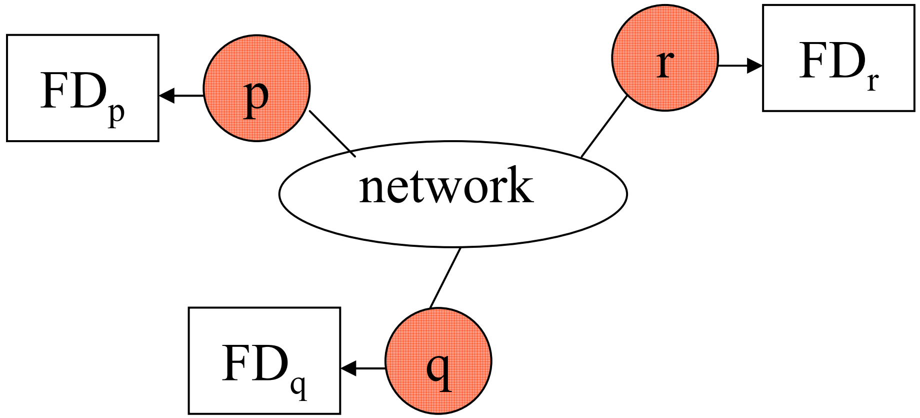
- There is a bound on the transmission delay of messages and on the process relative speed, but these bounds are not known
- There are known bounds on the transmission delay of messages and on the process relative speed, but these bounds hold only from some unknown point on.

Consensus solvable with a majority of correct processes

Models for solving consensus (3)

Failure detector model (Chandra,Toueg, 1995)

- Asynchronous model augmented with an oracle (called failure detector) defined by abstract properties



- A failure detector defined by a completeness property and an accuracy property

Failure detector model (2)

Example: failure detector ⟨⟩S

- Strong completeness: eventually every process that crashes is permanently suspected to have crashed by every correct process
- Eventual weak accuracy: There is a time after which some correct process is never suspected by any correct process

Consensus solvable with ⟨⟩S and a majority of correct processes

Drawback:

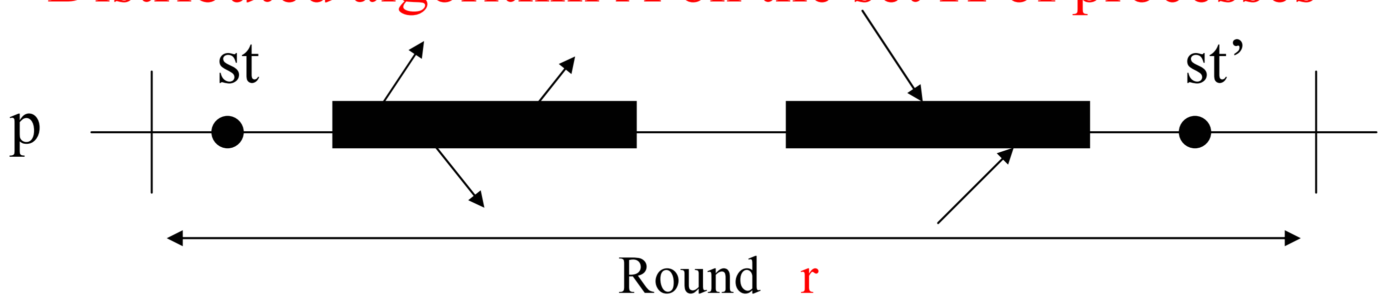
- requires reliable links

RbR model

- Joint work with Bernadette Charron-Bost
- Combinaison of:
  - Gafni 1998, *Round-by-round failure detectors*
    (unifies synchronous model and failure detector model)
  - Santoro, Widmayer, 1989, *Time is not a healer*
    (show that dynamic faults have the same destructive effect on solving consensus as asynchronicity)
- Existing models for solving consensus have put emphasis on static faults
- Our new model handles static faults and dynamic faults in the same way

RbR machine

- Distributed algorithm A on the set Π of processes



In round r, process p receives messages from the set HO(p, r)

- Communication predicate $\mathcal{P}$ on the HO's

  Examples: $\mathcal{P}_1$: $\forall p \in \Pi, \forall r > 0 : \; | HO(p, r) | > n/2$

  $\mathcal{P}_2$: $\exists\, r_0, \exists\, HO \in 2^{\Pi}, \forall p \in \Pi : HO(p, r_0) = HO$

- A problem is solved by a pair (A, $\mathcal{P}$)

Round-by-Round (RbR) model (2)

- Assume $q \notin HO(p, r)$: in the RbR model, no tentative to justify this (e.g, channel failure, crash of q, send-omission failure of q, etc.)
- This removes the difference between static and dynamic faults (static: crash-stop; dynamic: channel fault, crash-recovery)
- In the RbR model, no notion of faulty process

Consensus algorithms

- A lot ….

- Most influential algorithms:
  - Consensus algorithm based on the failure detector ⟨⟩S (Chandra-Toueg, 1995)
  - Paxos (Lamport, 1989-1998)

Consensus algorithms (2)

The CT (Chandra-Toueg) and Paxos algorithms have strong similarities but also important differences:

- CT based on rotating coordinator / Paxos based on a dynamically chosen leader
- CT requires reliable links / Paxos tolerates lossy links
- The condition for liveness clearly defined in CT
- The condition for liveness less formally defined in Paxos (can be formally expressed in the RbR model)

Paxos in the RbR model

Initialization

$x_p := v_p$

$vote_p := V \cup \{?\}$, initially ?

$voteToSend_p$:=false; $decided_p$:=false; ts_p:=0

Round r = 4Φ – 3 :

S^r_p : send $\langle x_p, ts_p \rangle$ to $leader_p(\Phi)$

T^r_p :

if $p = leader_p(\Phi)$ **and** $\#\langle x, ts \rangle$ rcvd $> n/2$ **then**

let $\underline{\theta}$ be the largest θ from $\langle v, \theta \rangle$ received

$vote_p$:= one v such that $\langle v, \underline{\theta} \rangle$ received

$voteToSend_p$:= true

Round r = 4Φ – 2 :

S^r_p : **if** $p = leader_p(\Phi)$ **and** $voteToSend_p$ **then**

send $\langle vote_p \rangle$ to all processes

T^r_p :

if received $\langle v \rangle$ from $leader_p(\Phi)$ **then**

$x_p := v$; $ts_p := \Phi$

Round r = 4Φ – 1 :

S^r_p :

if $ts_p = \Phi$ **then**

send $\langle ack \rangle$ to $leader_p(\Phi)$

T^r_p :

if $p = leader_p(\Phi)$ **and** $\#\langle ack \rangle$ rcvd $> n/2$ **then**

DECIDE ($vote_p$); $decided_p$:= true

Round r = 4Φ

S^r_p :

if $p = coord_p(\Phi)$ **and** $decided_p$ **then**

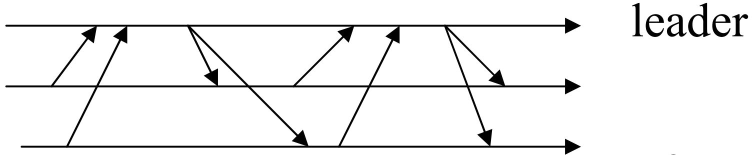
send $\langle vote_p \rangle$ to all processes

T^r_p :

if received $\langle v \rangle$ **and not** $decided_p$ **then**

DECIDE(v)); $decided_p$:= true

$voteToSend_p$:= false



Paxos (2)

- Kernel of round r: $K(r) = \bigcap_{p \in \Pi} HO(p, r)$
- Kernel of a phase Φ : $K(\Phi) = \bigcap_{\forall r \in \Phi} K(r)$

  (*phase* = sequence of rounds)

CONDITION FOR SAFETY	CONDITION FOR LIVENESS
None	$\exists \Phi_0 > 0$: $\forall p : \lvert HO(p, \Phi_0) \rvert > n/2$ **and** $\forall p, q: \; leader_p(\Phi_0) = leader_q(\Phi_0)$ **and** $leader_p(\Phi_0) \in K(\Phi_0)$

Atomic broadcast

- A lot of algorithms published …
- Easy to implement using a sequence of instance of consensus
- Consider atomic broadcast within a static group g:
  - Consensus within g on a set of messages
  - Let consensus #k decide on the set Msg(k)
  - Each process delivers the messages in Msg(k) before those in Msg(k+1)
  - Each process delivers the messages in Msg(k) in a deterministic order (e.g., in the order of the message ids)

Atomic broadcast (2)

Leads to the following algorithm:

- Each process *p* manages a counter *k* and a set *undeliveredMessages*
- **Upon** *abcast(m)* **do** *broadcast(m)*
- **Upon** reception of *m* **do**
  - add *m* to *undeliveredMessages*
  - **if** no consensus algorithm is running **then**

    Msg(k) ← consensus (*undeliveredMessages)*

    deliver messages in *Msg(k)* in some deterministic order

Conclusion

- Necessarily superficial presentation of group communication
- Comments:
  - Static/crash-stop model has reached maturity
  - Maturity not yet reached in the other models (static/crash-recovery, dynamic/crash-stop, …)
  - With the RbR model we hope to bridge the gap between static/crash-stop and static/crash-recovery (ongoing implementation work)
  - More work needed to quantitatively compare the various atomic broadcast algorithms (and other algorithms)