Failures and Consensus

Coordination

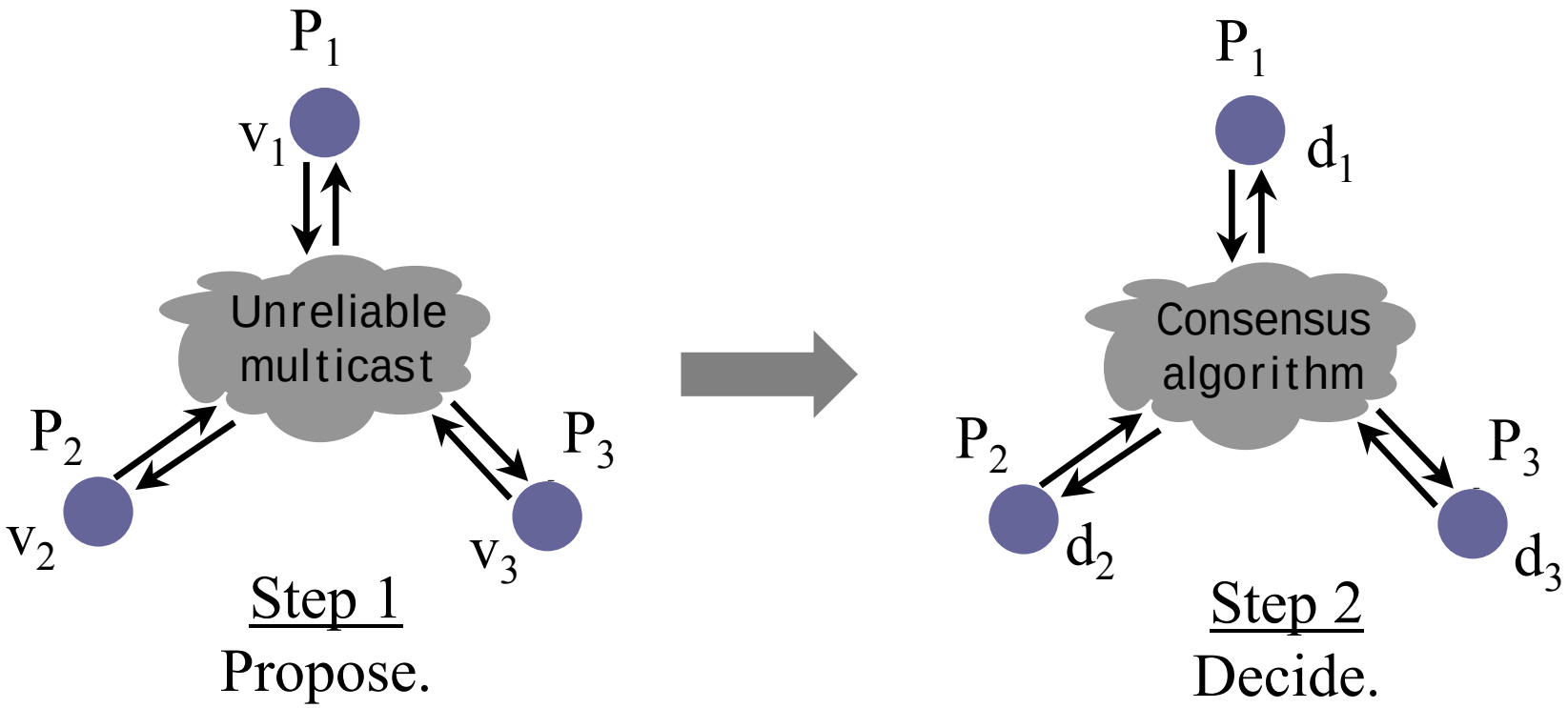
If the solution to availability and scalability is to decentralize and replicate functions and data, how do we coordinate the nodes?

- data consistency
- update propagation
- mutual exclusion
- consistent global states
- group membership
- group communication
- event ordering
- distributed consensus
- quorum consensus

Overview

- The consensus problem and its variants
- Failure models
- Consensus in synchronous model with no failures
- Consensus in synchronous model with fail-stop
- The trouble with byzantine failures
- Impossibility of consensus with too many byzantine failures
- Consensus in synchronous with a few byzantine failures
- Impossibility of consensus in asynchronous with failures
- Consensus in practice anyway
- Recovery and failure detectors

Consensus



Generalizes to N nodes/processes.

Properties for Correct Consensus

Termination: All correct processes *eventually* decide.

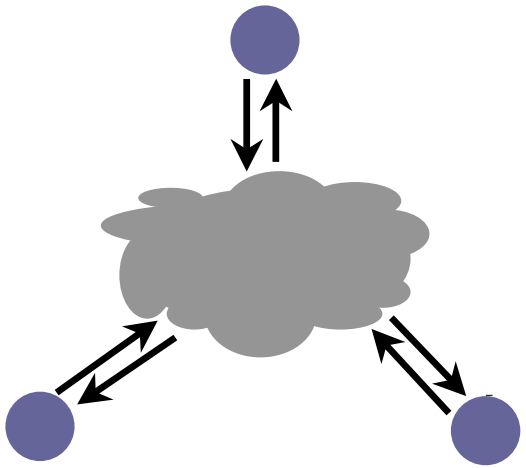
Agreement: All correct processes select the same d_i.

Or…(stronger) all processes that do decide select the same d_i, even if they later fail.

Integrity: All deciding processes select the "right" value.

- As specified for the variants of the consensus problem.

Variant I: Consensus (C)



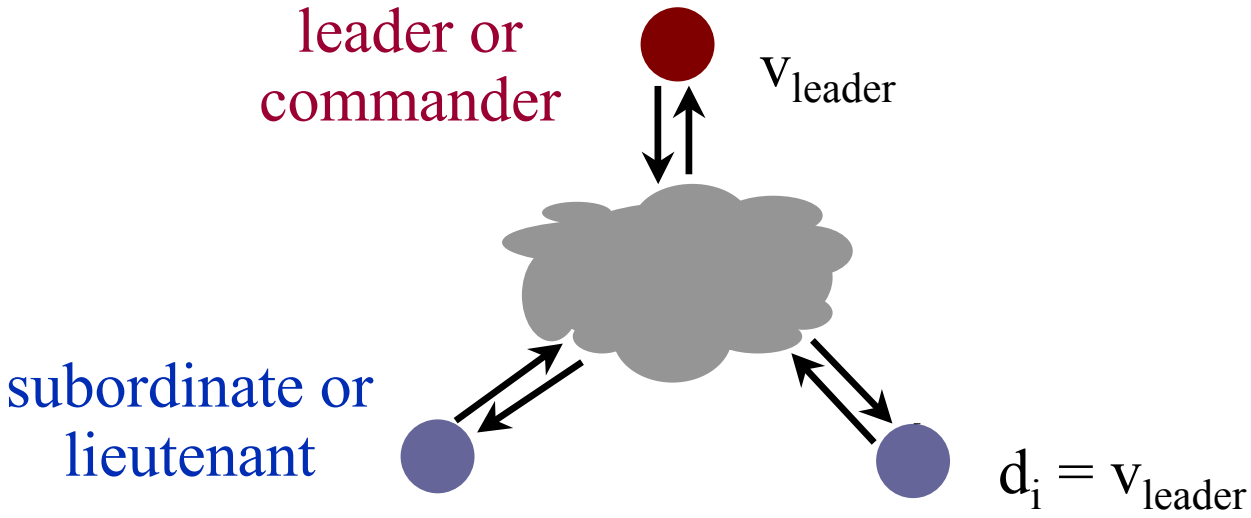
$d_i = v_k$

P_i selects d_i from $\{v_0, \ldots, v_{N-1}\}$.

All P_i select d_i as the same v_k .

If all P_i propose the same v, then $d_i = v$, else d_i is arbitrary.

Variant II: Command Consensus (BG)

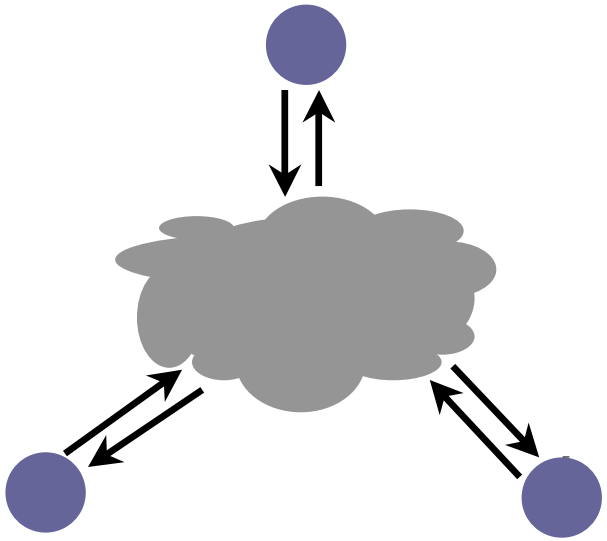


P_i selects $d_i = v_{leader}$ proposed by designated leader node P_{leader} if the leader is correct, else the selected value is arbitrary.

As used in the *Byzantine generals* problem.

Also called *attacking armies*.

Variant III: Interactive Consistency (IC)



$d_i = [v_0, \ldots, v_{N-1}]$

P_i selects $d_i = [v_0, \ldots, v_{N-1}]$ vector reflecting the values proposed by all correct participants.

Equivalence of Consensus Variants

If any of the consensus variants has a solution, then all of them have a solution.

Proof is by *reduction*.

- *IC from BG*. Run BG *N* times, one with each P_i as leader.
- *C from IC*. Run IC, then select from the vector.
- *BG from C*.

  Step 1: leader proposes to all subordinates.

  Step 2: subordinates run C to agree on the proposed value.
- *IC from C*? *BG from IC*? Etc.

Four Dimensions of Failure Models

Reliable vs. unreliable network

Reliable: all messages are eventually delivered exactly once.

Synchronous vs. asynchronous communication

Synchronous: message delays (and process delays) are bounded, enabling communication in *synchronous rounds*.

Byzantine vs. fail-stop

Fail-stop: faulty nodes stop and do not send.

Byzantine: faulty nodes may send arbitrary messages.

Authenticated vs. unauthenticated

Authenticated: the source and content of every message can be verified, even if a Byzantine failure occurs.

Assumptions

For now we assume:

- Nodes/processes communicate only by messages.
- The network may be synchronous or asynchronous.
- The network channels are *reliable*.

  Is this realistic?

There are three kinds of node/process failures:

- Fail-stop
- Authenticated Byzantine ("signed messages")
- Byzantine ("unsigned")

Consensus: synchronous with no failures

The solution is trivial in one round of proposal messages.

Intuition: all processes receive the same values, the values sent by the other processes.

Step 1. Propose.

Step 2. At end of round, each P_i decides from received values.

- *Consensus*: apply any deterministic function to $\{v_0,\ldots, v_{N-1}\}$.
- *Command consensus*: if v_{leader} was received, select it, else apply any deterministic function to $\{v_0,\ldots, v_{N-1}\}$.
- *Interactive consistency*: construct a vector from all received values.

Consensus: synchronous fail-stop

F+1 rounds of exchanges can reach consensus for N processes with up to F processes failing.

In each round, each node says everything that it knows that it hasn't already said in previous rounds.

At most N^2 values are sent.

Intuition: suppose P_i learns a value v from P_j during a round.

- Other correct processes also learned v from P_j during that round, unless P_j failed during the round.
- Other correct processes will learn it from P_i in the next round, unless P_i also fails during that round.
- Adversary must fail one process in each round, after sending its value to one other process…so F+1 rounds are sufficient if at most F failures occur.

Lamport's 1982 Result, Generalized by Pease

The Lamport/Pease result shows that consensus is impossible:

- with byzantine failures,
- if one-third or more processes fail ($N \leq 3F$),

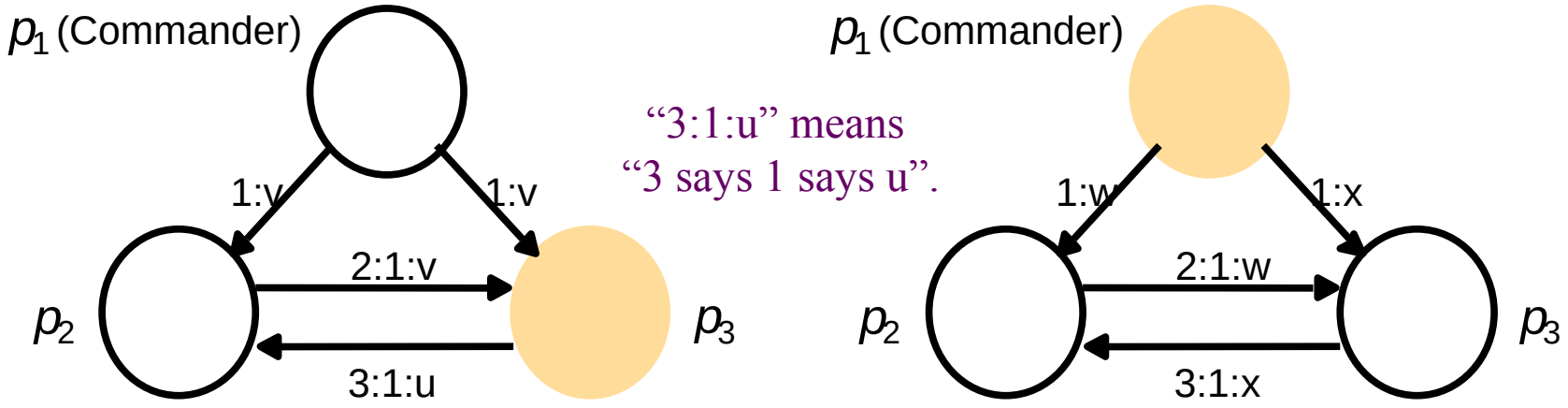
  Lamport shows it for 3 processes, but Pease generalizes to N.

- even with synchronous communication.

Intuition: a node presented with inconsistent information cannot determine which process is faulty.

The good news: consensus can be reached if $N > 3F$, no matter what kinds of node failures occur.

Impossibility with three byzantine generals

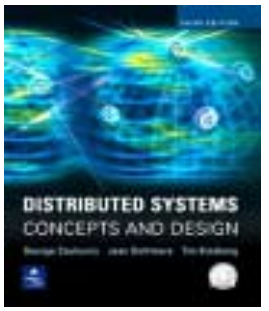


Faulty processes are shown shaded

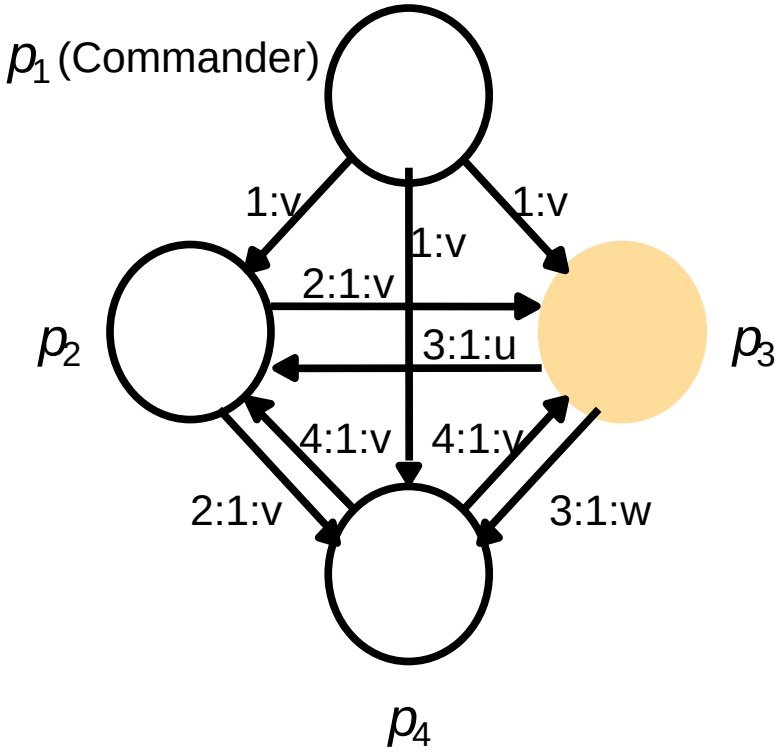
[Lamport82]

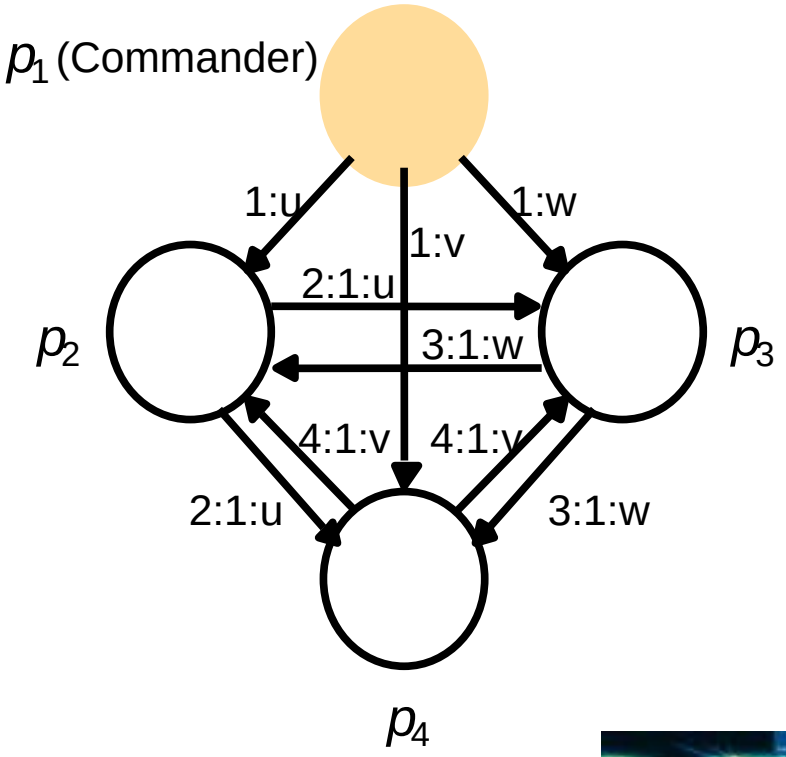
Intuition: subordinates cannot distinguish these cases.

Each must select the commander's value in the first case, but this means they cannot agree in the second case.



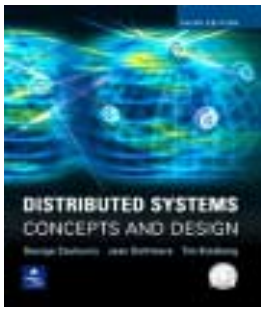
Solution with four byzantine generals





Faulty processes are shown shaded

Intuition: vote.



Summary: Byzantine Failures

A solution exists if less than one-third are faulty (N > 3F).

It works only if communication is synchronous.

Like fail-stop consensus, the algorithm requires F+1 rounds.

The algorithm is very expensive and therefore impractical.

Number of messages is exponential in the number of rounds.

Signed messages make the problem easier (*authenticated byzantine*).

- In general case, the failure bounds (N > 3F) are not affected.
- Practical algorithms exist for N > 3F. [Castro&Liskov]

Fischer-Lynch-Patterson (1985)

No consensus can be guaranteed in an asynchronous communication system in the presence of any failures.

Intuition: a "failed" process may just be slow, and can rise from the dead at exactly the wrong time.

Consensus may occur recognizably on occasion, or often.

e.g., if no inconveniently delayed messages

FLP implies that no agreement can be guaranteed in an asynchronous system with byzantine failures either.

Consensus in Practice I

What do these results mean in an asynchronous world?

- Unfortunately, the Internet is asynchronous, even if we believe that all faults are eventually repaired.
- Synchronized clocks and predictable execution times don't change this essential fact.

Even a single faulty process can prevent consensus.

The FLP impossibility result extends to:

- Reliable ordered multicast communication in groups
- Transaction commit for coordinated atomic updates
- Consistent replication

These are practical necessities, so what are we to do?

Consensus in Practice II

We can use some tricks to apply synchronous algorithms:

- *Fault masking*: assume that failed processes always recover, and define a way to reintegrate them into the group.

  If you haven't heard from a process, just keep waiting…

  A round terminates when every expected message is received.

- *Failure detectors*: construct a failure detector that can determine if a process has failed.

  A round terminates when every expected message is received, or the failure detector reports that its sender has failed.

But: protocols may block in pathological scenarios, and they may misbehave if a failure detector is wrong.

Recovery for Fault Masking

In a distributed system, a recovered node's state must also be consistent with the states of other nodes.

E.g., what if a recovered node has forgotten an important event that others have remembered?

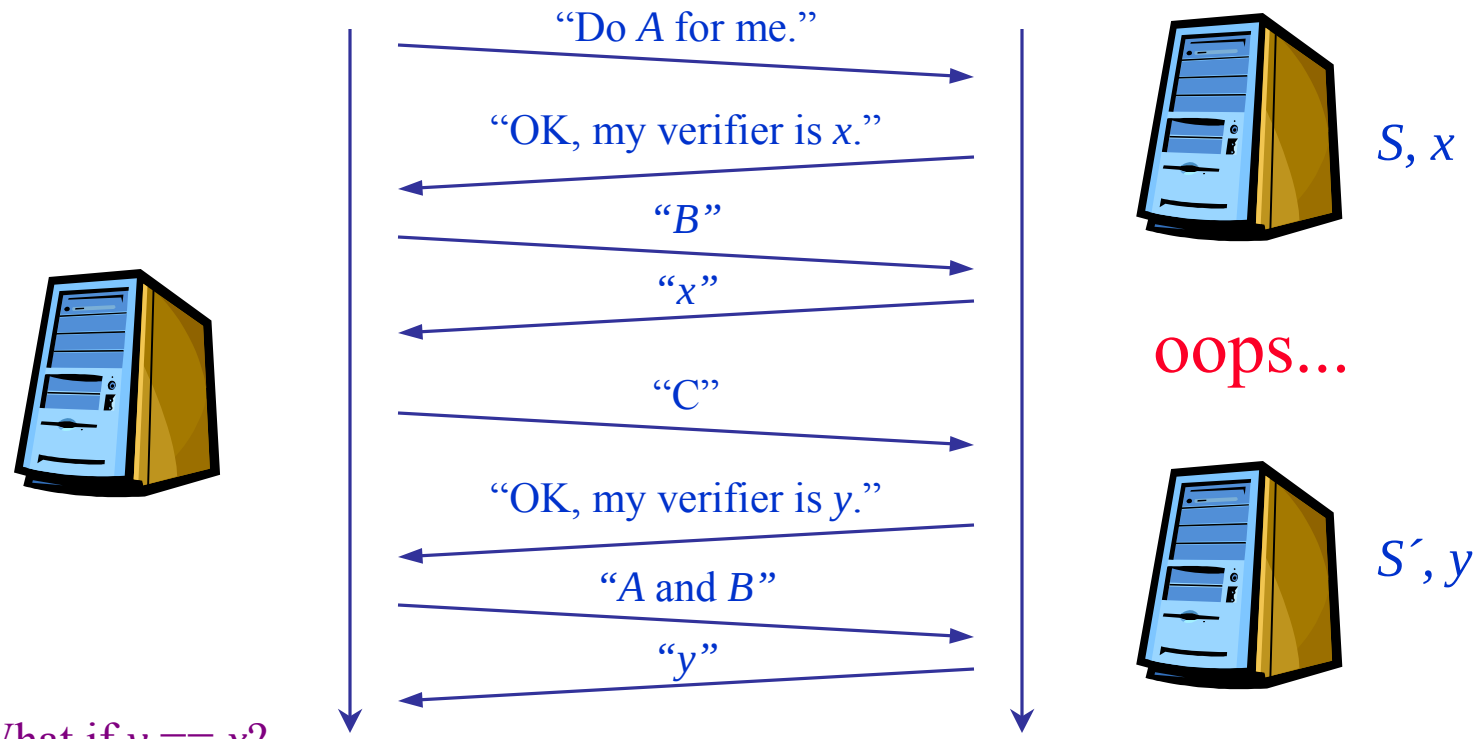
A functioning node may need to respond to a peer's recovery.

- rebuild the state of the recovering node, and/or
- discard local state, and/or
- abort/restart operations/interactions in progress

e.g., *two-phase commit* protocol

How to know if a peer has failed and recovered?

Example: Session Verifier



What if $y == x$?

How to guarantee that $y != x$?

What is the implication of re-executing *A* and *B*, and after *C*?

Some uses: NFS V3 write commitment, RPC sessions, NFS V4 and DAFS (client).

Failure Detectors

First problem: how to detect that a member has failed?

- pings, timeouts, beacons, heartbeats
- recovery notifications

  "I was gone for awhile, but now I'm back."

Is the failure detector *accurate*?

Is the failure detector *live (complete)*?

In an asynchronous system, it is possible for a failure detector to be accurate or live, but not both.

- FLP tells us that it is impossible for an asynchronous system to agree on *anything* with accuracy and liveness!

Failure Detectors in Real Systems

Use a failure detector that is live but not accurate.

- Assume bounded processing delays and delivery times.
- Timeout with multiple retries detects failure accurately with high probability. Tune it to observed latencies.
- If a "failed" site turns out to be alive, then restore it or kill it (*fencing*, *fail-silent*).

Use a recovery detector that is accurate but not live.

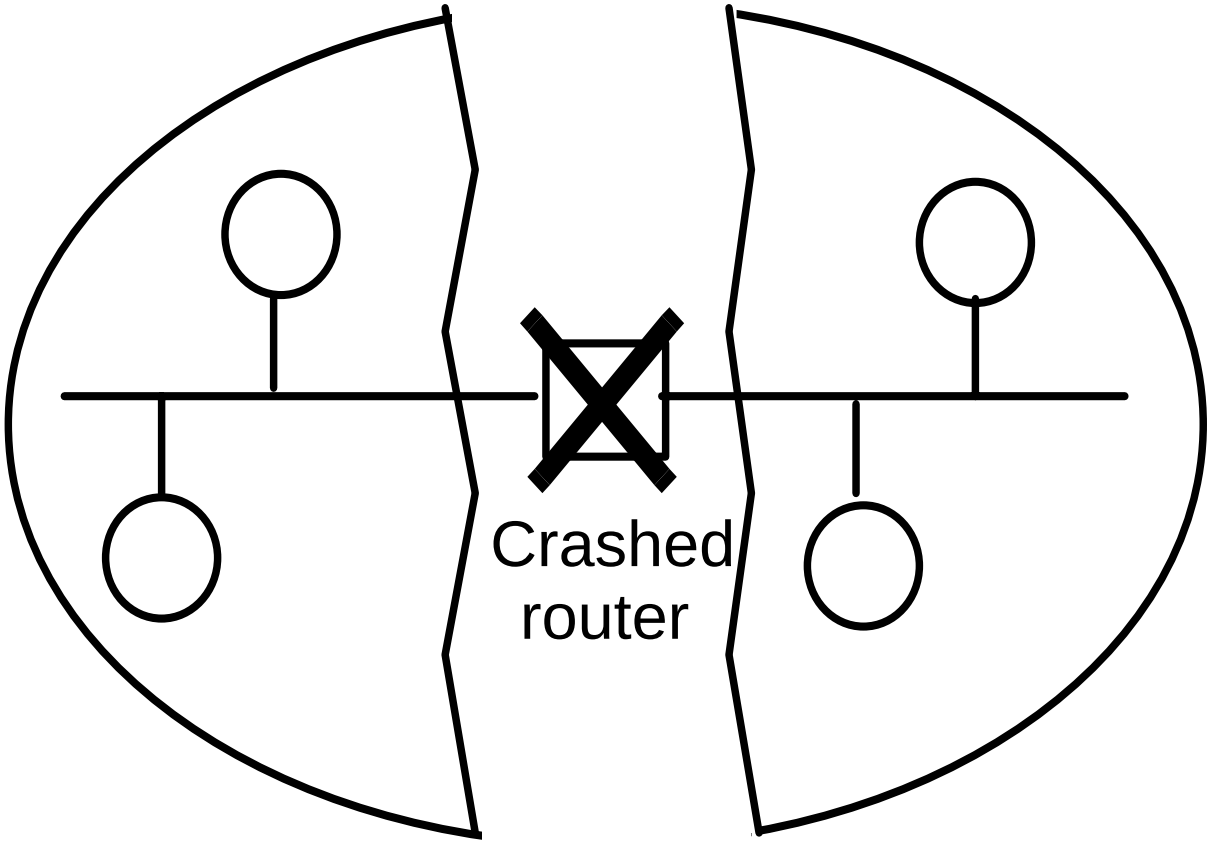
- "I'm back....hey, did anyone hear me?"

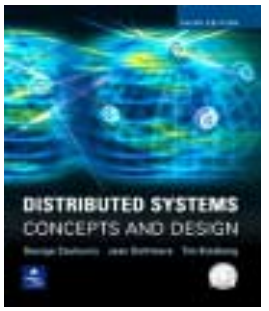
What do we assume about communication failures?

- How much pinging is enough?
- 1-to-N, N-to-N, ring?

What about network partitions?

A network partition





Group Membership Services

Second problem: How to propagate knowledge of failure/recovery events to other nodes?

- Surviving nodes should agree on the new view (*regrouping*).
- Convergence should be rapid.
- The regrouping protocol should itself be tolerant of message drops, message reorderings, and failures.

  liveness and accuracy again

- The regrouping protocol should be scalable.
- The protocol should handle network partitions.

This is another instance of a *consensus* problem.

Failure Detectors

CS 717, Cornell University

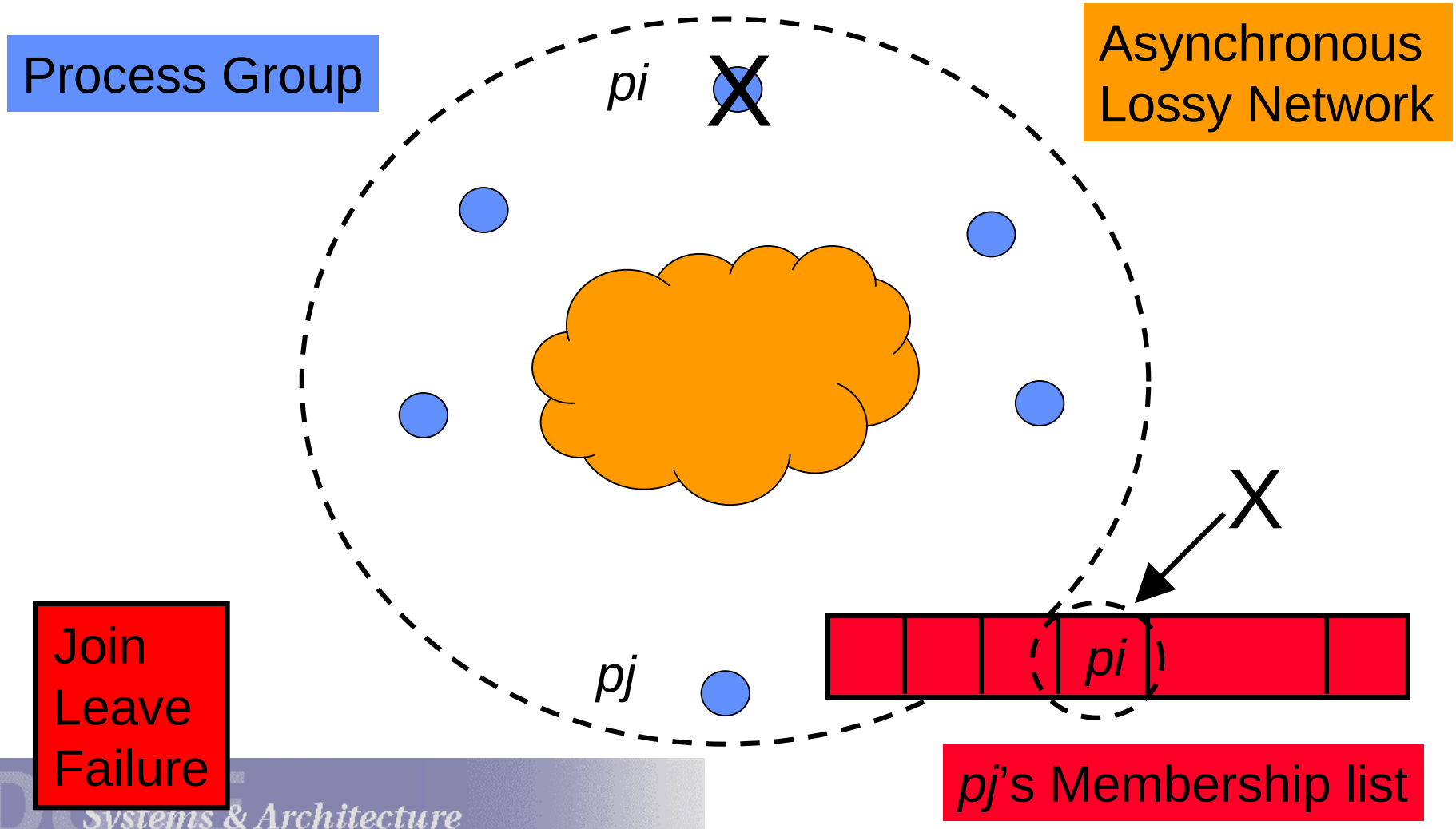
Ashish Motivala

Dec 6th 2001

(Slides borrowed from ppt on Web; permission requested.)

[Motivala]

Group Membership Service



[Motivala]

Chandra/Toueg Failure Detectors

Theoretical Idea

Separate problem into

- The consensus algorithm itself
- A "failure detector:" a form of oracle that announces suspected failure
- But the process can change its decision

Question: what is the weakest oracle for which consensus is always solvable?

[Motivala]

Sample properties

Completeness: detection of every crash

- Strong completeness: Eventually, every process that crashes is permanently suspected by every correct process
- Weak completeness: Eventually, every process that crashes is permanently suspected by *some* correct process

[Motivala]

Sample properties

Accuracy: does it make mistakes?

- Strong accuracy: No process is suspected before it crashes.
- Weak accuracy: Some correct process is never suspected
- Eventual {strong/ weak} accuracy: there is a time after which {strong/weak} accuracy is satisfied.

[Motivala]

A sampling of failure detectors

Completeness	**Accuracy**			
	Strong	Weak	Eventually Strong	Eventually Weak
Strong	*Perfect* P	*Strong* S	*Eventually Perfect* ◊P	*Eventually Strong* ◊ S
Weak	D	*Weak* W	◊ D	*Eventually Weak* ◊ W

[Motivala]

Perfect Detector?

Named *Perfect*, written *P*

Strong completeness and strong accuracy

Immediately detects all failures

Never makes mistakes

[Motivala]

Example of a failure detector

The detector they call W: *"eventually weak"*

More commonly: ◊W: *"diamond-W"*

Defined by two properties:

- There is a time after which every process that crashes is suspected by some correct process {weak completeness}
- There is a time after which some correct process is never suspected by any correct process {weak accuracy}

Eg. we can eventually agree upon a leader. If it crashes, we eventually, accurately detect the crash

[Motivala]

◊W: Weakest failure detector

They show that ◊W is the weakest failure detector for which consensus is guaranteed to be achieved

Algorithm is pretty simple

- Rotate a token around a ring of processes
- Decision can occur once token makes it around once without a change in failure-suspicion status for any process
- Subsequently, as token is passed, each recipient learns the decision outcome

[Motivala]

Building systems with ◊W

Unfortunately, this failure detector is not implementable

This is the weakest failure detector that solves consensus

Using timeouts we can make mistakes at arbitrary times