Distributed Systems (5DV020)

Coordination and agreement

Fall 2013

Processes often need to coordinate their actions and/or reach an agreement

Which process gets to access a shared resource?

Has the master crashed? Elect a new one!

Failure detection – how can we know that a node has failed (e.g., crashed)?

Why not use a master-slave relationship?

Because we want our systems to keep working correctly even if failures occur

We need to avoid single points of failure

Failure detection

Network partition

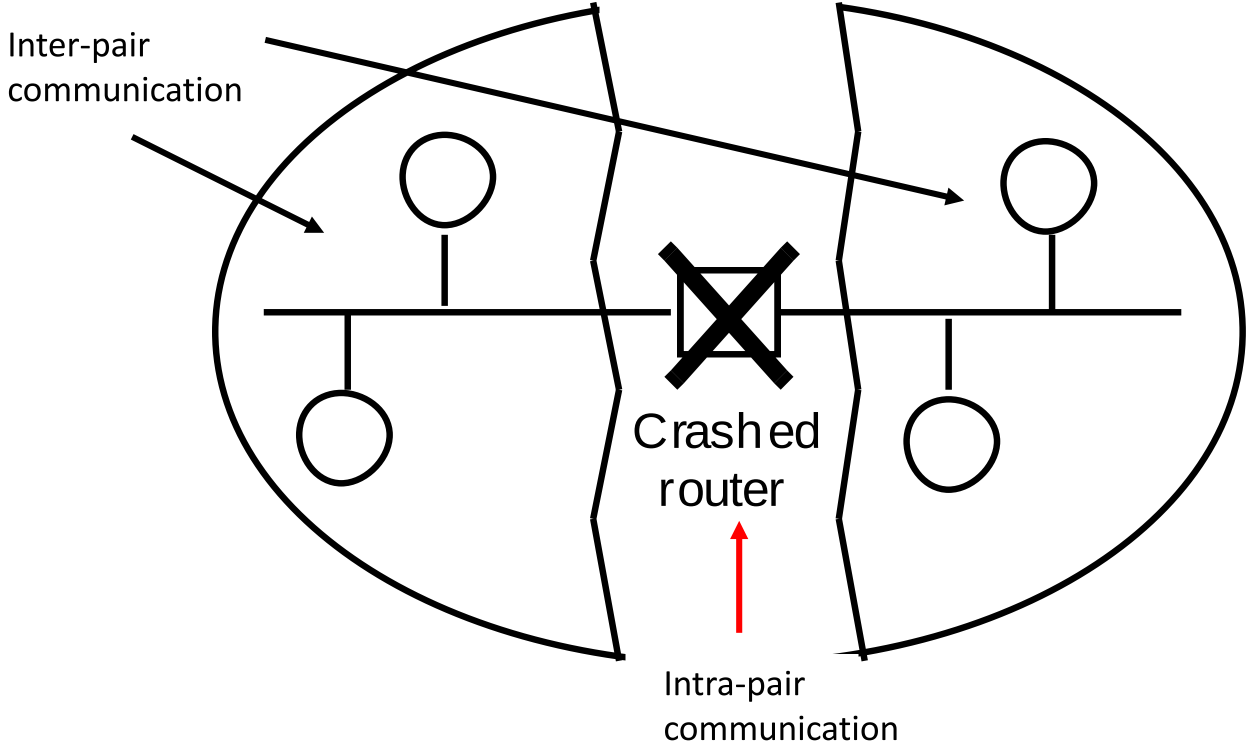


Figure adapted from Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5 © Pearson Education 2012 – **Figure 15.1**

How to determine that a process has crashed?

- Correct process
  - Exhibits no failures at any point
- Failure detector
  - Detects if processes fail
  - Unreliable failure detector
    - Unsuspected or suspected
  - Reliable failure detector
    - Unsuspected or failed

Example of unreliable failure detector

```
max-message-delay = D
processes exchange im-alive messages every T seconds
if (time-since-last-message== T + D)
        if (not receive im-alive message from p_i)
                state-p_i = SUSPECTED
when (receive im-alive message from p_i)
        state-p_i = UN-SUSPECTED
```

Tradeoffs …

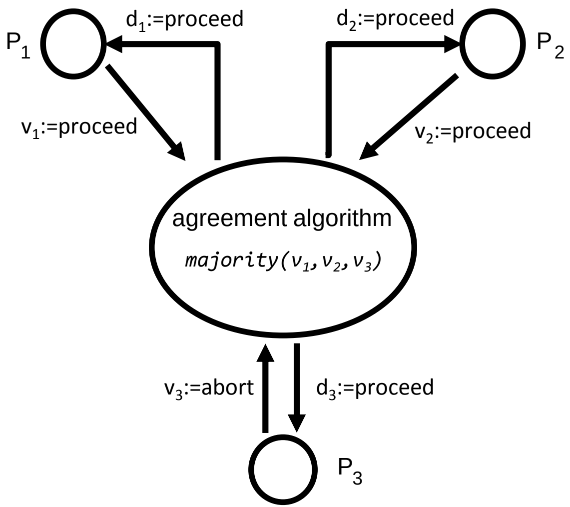
- Small values of **T** and **D**
  - Lots of suspected non-crashed processes
  - Lots of bandwidth due to **im-alive** messages
- Large timeout values
  - Crash processes may be considered unsuspected
- Adapt timeout values (to increase accuracy)
  - According to observed network delays
- Synchronous systems → reliable failure detector
  - **D** is an absolute bound on message transmission

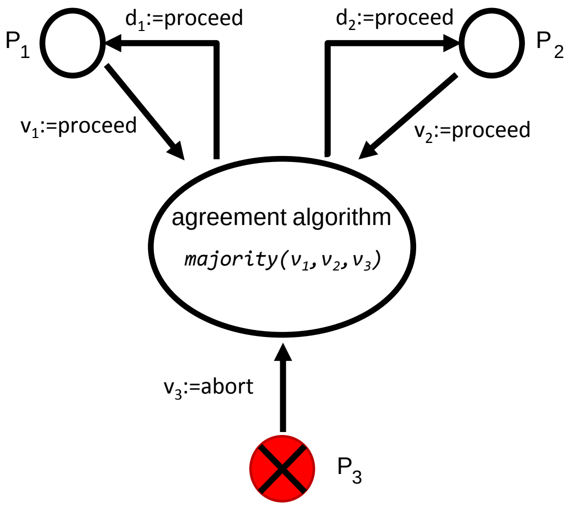
Consensus and related problems

Agreement…

- Mutual exclusion
  - Agreement on which process enter the CS
- Election
  - Agreement on which process is the leader
- Totally ordered multicast
  - Agreement on which messages are delivered and in which order
- Processes need to agree on a value after proposed by one or more processes … even in the presence of faults (crash and arbitrary)
  - Consensus
  - Byzantine Generals Problem (BGP)
  - Interactive consistency

Motivation

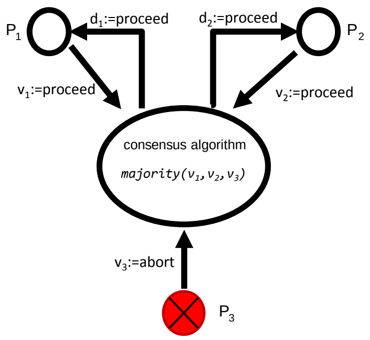

P1
d1:=proceed
d2:=proceed
P2
v1:=proceed
v2:=proceed
agreement algorithm
majority(v1,v2,v3)
v3:=abort
d3:=proceed
P3


P1
d1:=proceed
d2:=proceed
P2
v1:=proceed
v2:=proceed
agreement algorithm
majority(v1,v2,v3)
v3:=abort
P3

Consensus

Processes need to agree on a single value from values proposed by all processes

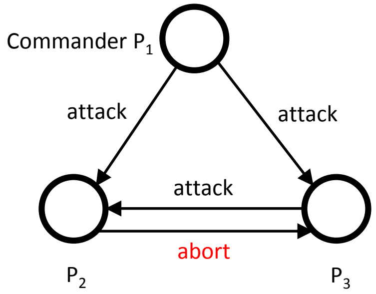
- Every process begins in an *undecided* state
- A process propose one of *D* possible values
- Processes exchange values
- Each process decides on one of the proposed values
  - Once choosing a value, processes enters a *decided* state
  - Processes can't change their chosen value once in a decided state



Byzantine Generals Problem (BGP)

A commander issues an order (attack or retreat), lieutenants need to decide what to do

- One or more generals are treacherous (faulty)
  - Commander issues an order to lieutenants
  - Lieutenants exchange messages with commander's orders
  - Each process decides on the orders to follow



Interactive consistency

Processes need to agree on a value for each process (a *decision vector*)

- For example so that each process knows about each other states

General requirements

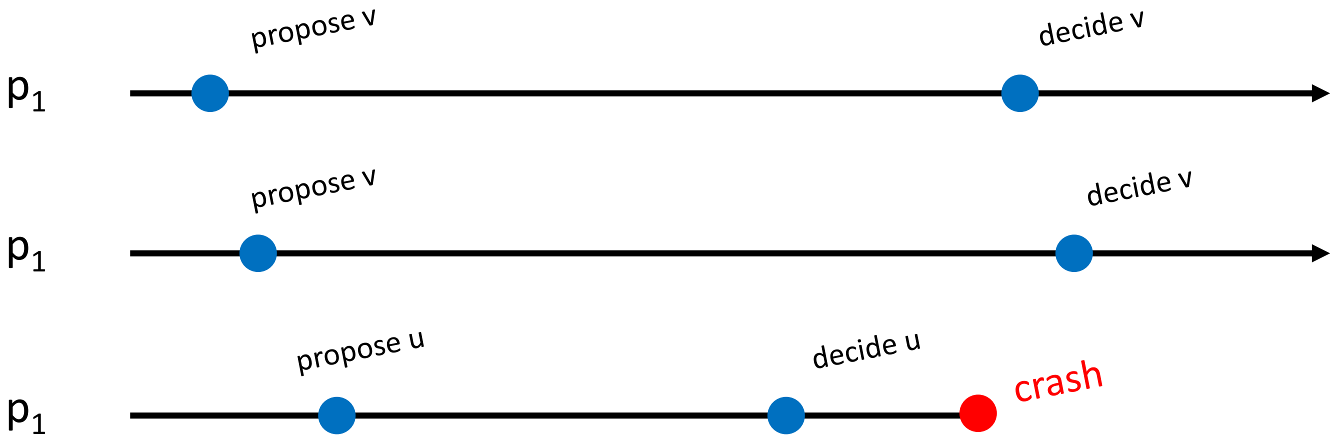
	Termination	Agreement	Integrity
Consensus	Eventually each correct process sets its decision variable.	The decision value of all correct processes is the same (all processes in the *decided* state).	If all correct processes propose the same value, any correct process in the *decided* state has chosen that value.
Byzantine Generals	Eventually each correct process sets its decision variable.	The decision value of all correct processes is the same (all processes in the *decided* state).	If the commander is correct, then all processes decide on the value that the commander proposed.
Interactive Consistency	Eventually each correct process sets its decision variable.	The decision vector of all correct processes is the same.	If p_i is correct, then all correct processes decide on v_i as the *ith* component of their vector.

It is possible to derive a solution to one problem using a solution from another problem!

Simple if processes can't fail

- Collect all processes in a group
- Each process multicast its proposed value to the members of the group
- Each process waits for N messages (including own)
  - Evaluates *majority*$(v_1, v_2, \ldots, v_N)$
  - If no majority exists, majority returns a special value

but if processes can fail…



But still satisfies the definition of consensus

A simple algorithm for synchronous systems (crash failures)

V: set of initial values $\{v_i\}$

For k=1 to f+1 do

    send $\{v \in V | P_i \text{ has not already sent } v\}$ *to all*

    receive S_j *from all processes* P_j, $j \neq i$

    $V = V \cup S_j$

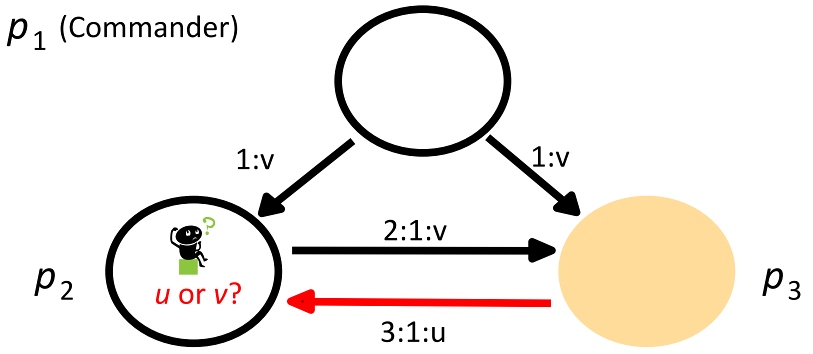
y = min(V)

- *f* is the max number of failed processors
  - Need to know *f*
- Algorithm based on rounds
  - *f+1* rounds

Any algorithm requires at least *f+1* rounds of message exchanges in order to reach consensus despite up to *f* crash failures!

BGP in synchronous systems (3 processes)

2 round of messages, commander to lieutenants and exchange among lieutenants

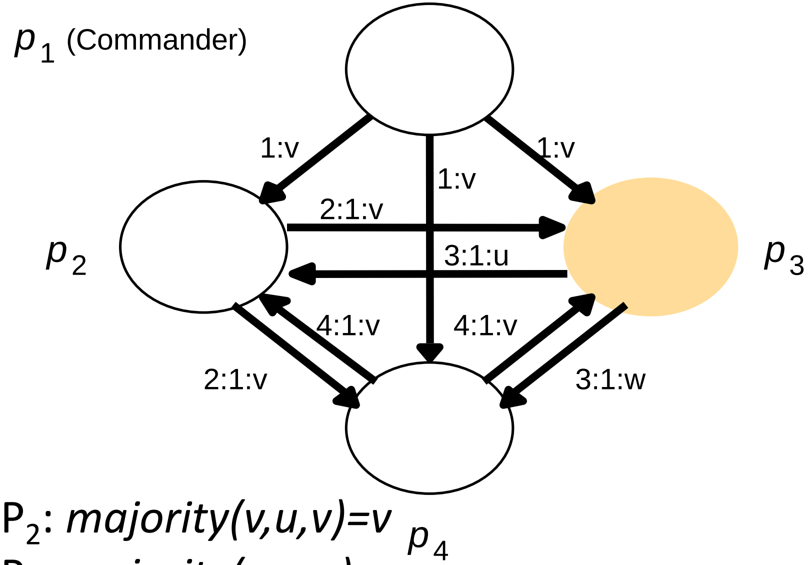


Faulty processes are shown colored

It is impossible to derive a solution if $N \leq 3f$

It is possible to derive a solution if $N \geq 3f + 1$

BGP with 4 processes, 1 faulty, 2 rounds



P_2: *majority(v,u,v)=v*

P_4: *majority(v,v,w)=v*

Faulty processes are shown colored

P_2: P_3:P_4: *majority(u,v, w)=* ⊥

Possible with $N \geq 3f + 1$ processes, where *f* is amount of treacherous ones

Efficiency, according to …

- The number of rounds that it takes
  - Measures how long it takes for the algorithm to terminate
  - At least *f+1* rounds
- The number of messages required
  - $O\left(N^{f+1}\right)$ messages
  - $O(N^2)$ messages using signed messages
- Very expensive, only when necessary

Final notes

Solutions rely on system being synchronous

- Message exchanges take place in rounds

Asynchronous system – bad!

- No timing constraints

Fischer's impossibility result

- Even with just one crashing process, we ***can't*** guarantee to reach consensus in an asynchronous system
  - Can't distinguish between crash process and a slow one
  - No consensus => no BGP, no interactive consistency and no totally ordered and reliable multicast...

Still, we manage to do quite well in practice, how can that be?

How to cope with the impossibility result…

- Mask the faults
  - Use persistent storage and allow process restarts
- Use failure detectors
  - No reliable detectors, but good enough, agree that process is crashed if it takes too long to receive a message (fail silent)
  - Eventually weak failure detector, reaches consensus while allowing suspected processes to behave correctly instead of excluding them
- Randomization
  - Introduces an element of chance that affects the adversary's strategy

- If you want to learn more:

  http://www.ict.kth.se/courses/ID2203/video_lectures.html

- Further reading:

  Leslie Lamport **Paxos Made Simple**
  ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001) 51-58.

  The article is well worth your time…

  http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf

Summary

- Unreliable failure detectors
  - Inaccurate and incomplete
- Reliable failure detectors
  - Require the system to be synchronous
- The problem of agreement is for processes to agree on a value after one or more of the processes has proposed values (even in the presence of faults)
  - Consensus, Byzantine Generals problem, Interactive consistency,…

- Fisher's impossibility result (asynchronous systems)
  - it is impossible to reach consensus even with a single faulty process
- Synchronous systems
  - Impossible for three generals
  - Possible when $N \geq 3f + 1$ processes, with f faulty processes
- Techniques for avoiding Fisher’s result
  - Masking faults
  - Failure detectors
  - Randomization

Next Lecture

Replication and Consistency