CS 425/ECE 428/CSE424
Distributed Systems
(Fall 2009)

Lecture 9

Consensus I

Section 12.5.1-12.5.3

Klara Nahrstedt

Acknowledgement

- The slides during this semester are based on ideas and material from the following sources:
  - Slides prepared by Professors M. Harandi, J. Hou, I. Gupta, N. Vaidya, Y-Ch. Hu, S. Mitra.
  - Slides from Professor S. Gosh's course at University o Iowa.

Administrative

- **MP1 posted September 8, Tuesday**
  - **Deadline, September 25 (Friday), 4-6pm Demonstrations**

Plan for Today

- Failure Models
- Three Problems
  - Consensus
  - Byzantine Generals
  - Interactive Consistency
- Synchronous Setting

Failure Models

- **Crash failure: ceases to execute**
  - **Permanent**
  - **Cause:, e.g., power loss**
  - **Variant: dead for finite period of time then resumes**
- **Omission failure: process or communication channel fails to perform actions that it is supposed to do.**
  - **Communication Omission failure: sender sends a sequence of messages but receiver does not receive some subset of messages**
    - » **Cause: e.g., interference in medium**
  - **Process Omission failures: crash failure**
- **Timing failures**
  - **Messages do not arrive in time, computation takes longer then expected times**
  - **Cause: e.g., congestion, over-loading, garbage-collection**

Failure Models

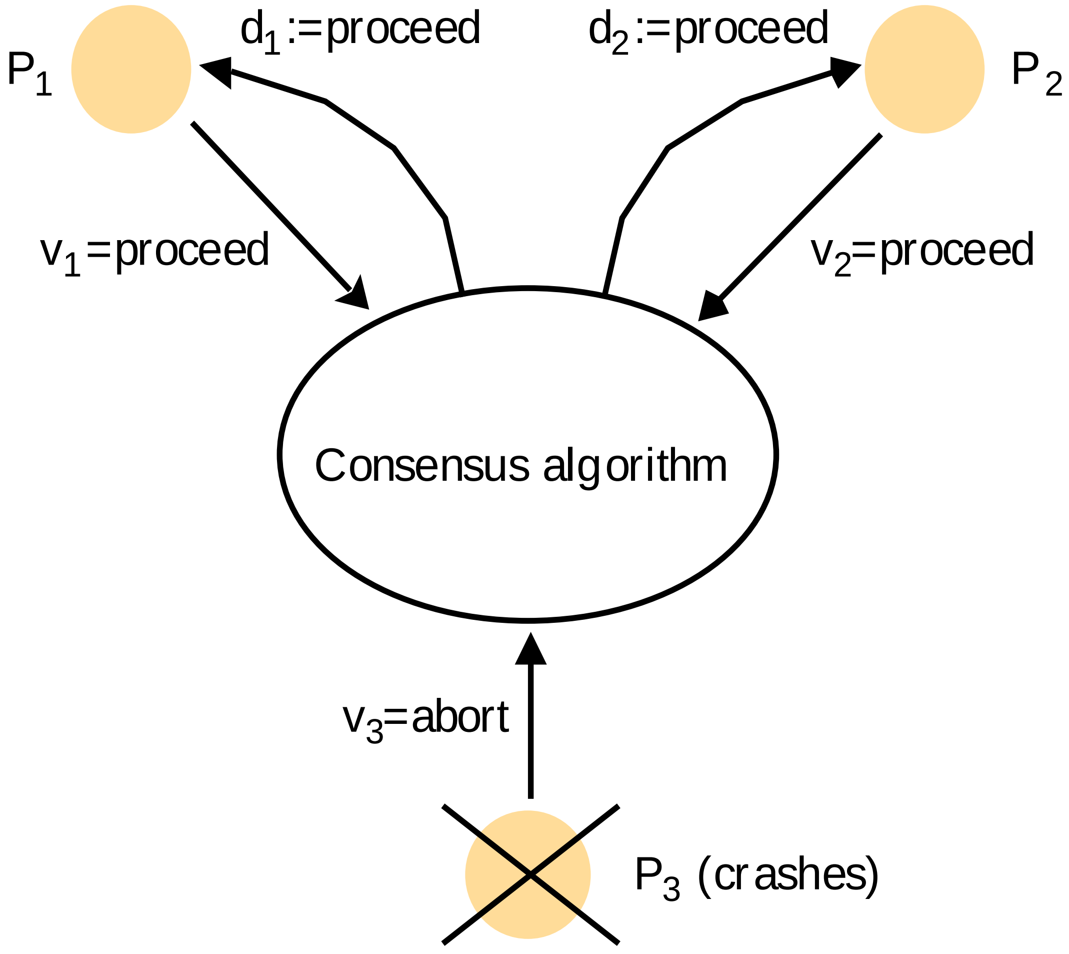
- **Transient failure: process jumps to arbitrary state and resumes normal execution**
  - **Cause: e.g., gamma rays**
- **Byzantine failure: arbitrary messages and transitions**
  - **Cause: e.g., software bugs, malicious attacks**

Definition of Consensus (C) Problem

- N **processes** $\{0,1,2,\ldots, N\text{-}1\}$ **try to agree**
- p_i **begins in undecided state and proposes value**

  $v_i \in D$
- p_i **'s communicate by exchanging values**
- p_i **sets its decision value di and enters decided state**
- **Requirements**
  - **Termination: eventually all correct processes decide**
    - » **i.e., each correct process sets its decision variable**
  - **Agreement : decision value of all correct processes is the same,**
    - » **i.e., if** p_i **and** p_j **are correct and decided , then** $d_i = d_j$
  - **Integrity: if all correct processes proposed** v**, then any correct decided process has** $d_i = v$

Consensus for three processes



Byzantine Generals (BG) Problem

- *N > 2* **generals** *{0,1, 2, … N-1}*
- **One of the generals is *the commander* who issues attack or retreat commands to all the other generals**
- **All generals try to agree about whether to attack or retreat**
- **Some generals (including the commander) may be traitors (byzantine)**

- **Requirements:**
  - **Termination: all correct generals decide**
  - **Agreement: if p_i and p_j are correct and decided then $d_i = d_j$**
  - **Integrity: if commander is correct, then all correct processes decide value issued by commander**
- **If commander is correct, then integrity implies agreement**

Interactive Consistency (IC) Problem

- *N* **processes** *{0,1,2,…, N-1}* **try to agree on vector of values**
- p_i **begins in undecided state and proposes a value** $v_i \in D$
- p_i **sets its decision value di and enters decided state**

- **Requirements:**
  - **Termination:** **all correct processes decide**
  - **Agreement:** **the decision vector for all correct processes is the same**
  - **Integrity:** **if** p_i **is correct, then for any correct process** p_j

$$d_j[i] = v_i$$

C to BG to IC

- **How to solve IC from an algorithm for solving BG?**
  - **Run BG N times once with each process as commander**
- **How to solve C from an algorithm for IC?**
  - **If majority of processes are correct, then solve IC and then apply majority function**
- **How to solve BG using an algorithm for C?**
  - **Commander sends proposed value to itself and the other processes which then run C**
- **How to solve RTO (Reliable Total Ordered) – multicast from C and vice-versa, under crash failures only?**

Solving C with RTO-multicast

- **All processes form a group**
- p_i **performs RTO-multicast (v_i, g)**
- p_i **sets** $d_i = m_i$**, where** m_i **is the first msg delivered by RTO-multicast**
  - **Termination guaranteed by reliable multicast**
  - **Agreement and validity by definitely of TO**

- **Solving consensus using basic multicast in the case where up to f processes may crash**

Consensus in Synchronous Systems

Consensus in a synchronous system Dolev & Strong (1983)

Algorithm for process $p_i \in g$; algorithm proceeds in $f + 1$ rounds

On initialization

$Values_i^1 := \{v_i\}$; $Values_i^0 = \{\}$;

In round r $(1 \le r \le f + 1)$

$B\text{-}multicast(g, Values_i^r - Values_i^{r-1})$; // Send only values that have not been sent

$Values_i^{r+1} := Values_i^r$;

while (in round r)

{

On B-deliver(V_j) *from some* p_j

$Values_i^{r+1} := Values_i^{r+1} \cup V_j$;

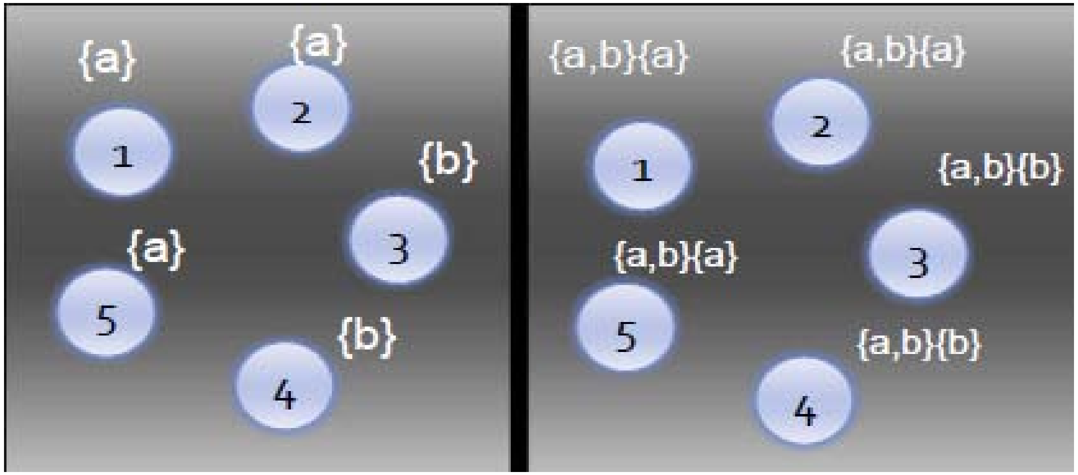
}

After $(f + 1)$ *rounds*

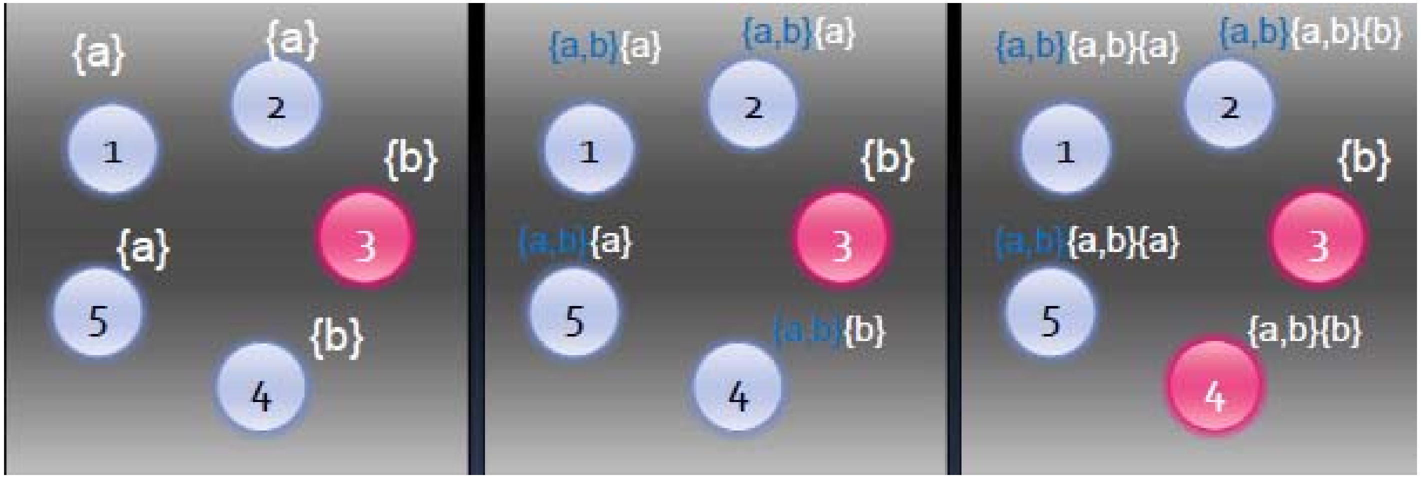
Assign $d_i = minimum(Values_i^{f+1})$;

Examples

Example execution: with No failures (f = 0)



Example execution: with f = 2



Correctness of Dolev & Strong Algorithm

- **Termination: finite number of rounds, finite duration of each round**
- **Agreement and integrity**
  - **We will prove by contradiction that** $V_i[f+1] = V_j[f+1]$
- **Suppose** $V_i[f+1] \neq V_j[f+1]$ *with f crashes*

➔**There is** $v \in V_i[f+1]$, **but** v **is not in** $V_j[f+1]$, **hence there is** p_k **that delivered** v **to** p_i **in round** $f+1$ **but crashed before delivering** v **to** p_j

➔**There is** $v \in V_k[f]$, **but** v **not in** $V_j[f]$, **hence, there is** p_l **that delivered** v **to** p_k **in round** f **but crashed before delivering** v **to** p_j

➔**… all the way back to** $V_j[1]$

➔ **Proceeding in this way, we infer at least one crash in each of the preceding rounds (i.e., which implies f+1 crashes)**

➔ **But we have assumed at most *f* crashes can occur and there are *f+1* rounds ➔ contradiction.**

Byzantine Generals in Synchronous Systems

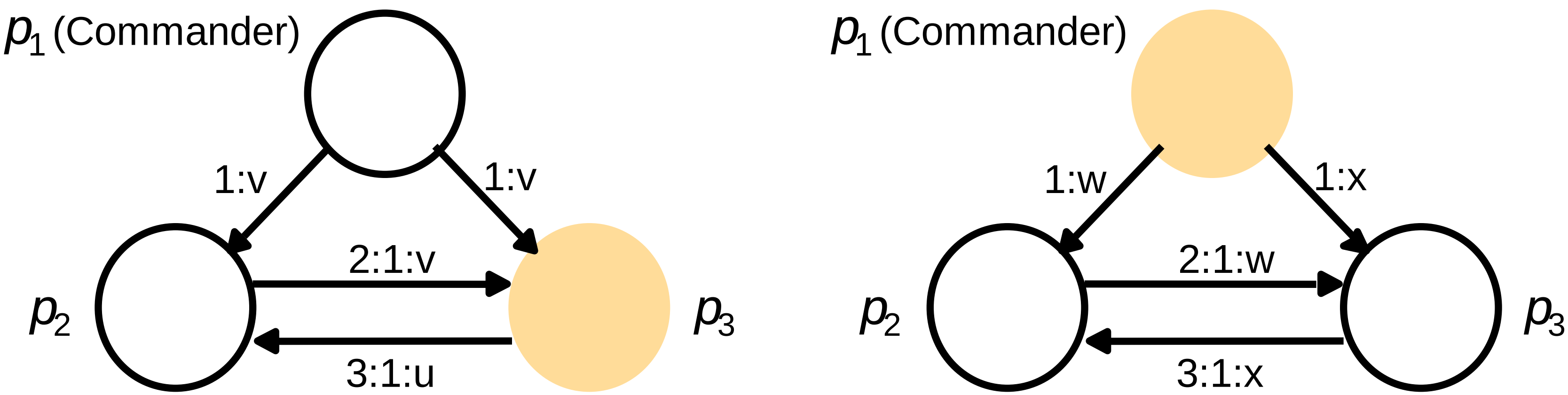
BG in Synchronous System

- **Assumptions**
  - **Up to *f* of *N* processes may be Byzantine**
  - **Synchronous implies**
    - **Correct processes can detect absence of messages with timeout, but cannot conclude that sender has crashed**

- **Is BG solvable?**
  - **For N = 3f ?**
  - **For N = 3, f = 1?**

Impossibility (no solution) with N = 3, f = 1

- **Lamport et al (1982) considered three processes with one Byzantine process**
- **No solution to achieve agreement**
- **Example**
  - **1:v means “1 says v”, 2:1:v means “2 says 1 says v”**
  - **2 different scenarios appear identical to p2**



Faulty processes are shown coloured

Impassibility with N ≤ 3f (outline)

- **Pease et al generalized basic impossibility result**
- **Simulation-based argument**

  - **Impossibility shown by contradiction**
  - **Assume there exists algorithm for N≤3f (e.g. N = 12, f = 4)**

Use algorithm to solve BG for N= 3 and f =1 thus reaching contradiction!

  - **Assume three processes {0,7,1} , each simulate behavior of 4 generals**

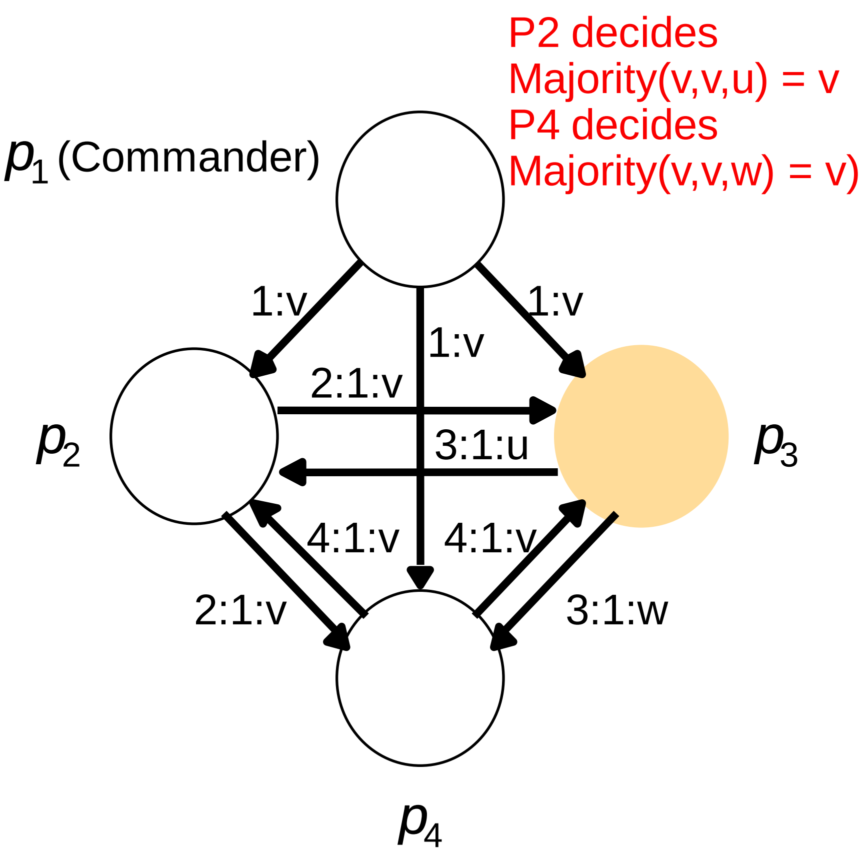
  - **Assume process 0 is faulty, then {0,11,5,6} generals will generate byzantine failures. All other processes are correct**
  - **Correctness of simulated algorithm tells us that algorithm terminates and 1 and 7 satisfy integrity**
  - **2 correct processes {1,7} solve consensus in spite of failure of 0**

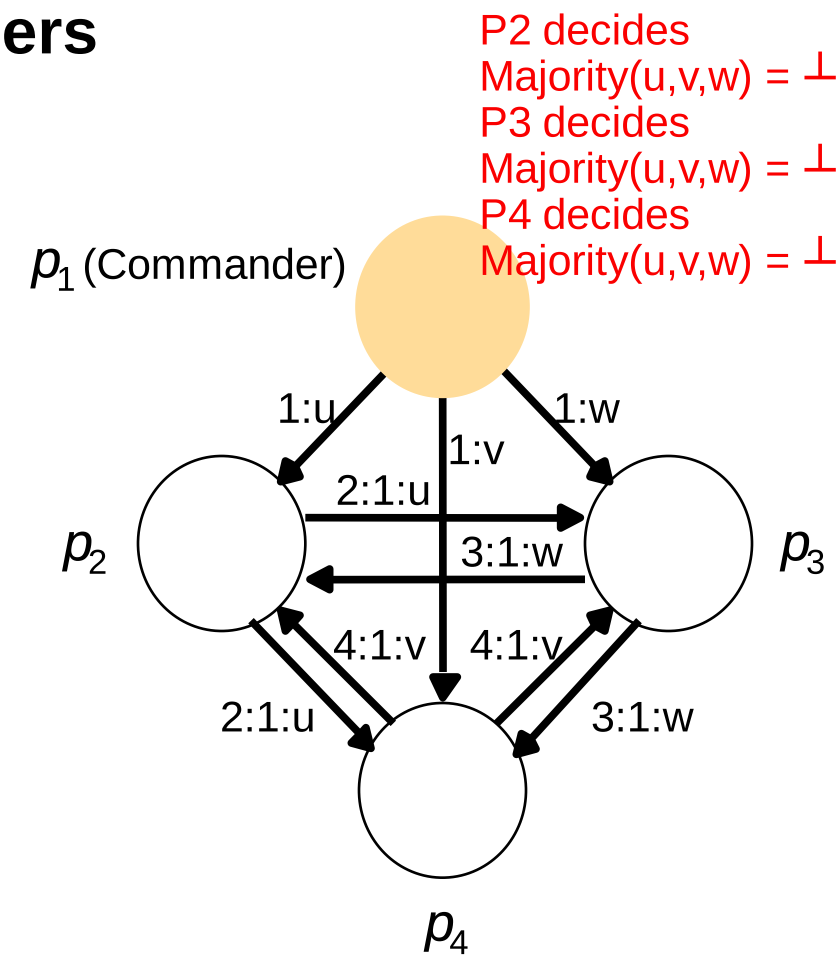
Contradiction (since N= 3, f=1 case is unsolvable)

BG Algorithm for N = 3f + 1

- **2 rounds**

1. **commander sends value to lieutenants**
2. **lieutenants send value to peers**





Faulty processes are shown coloured

Summary

- BG algorithm for N ≥ 3f + 1 by Pease et al
- This algorithm can account for omissions
  - Timeout (synchronous) and assume that the sent value was $\perp$
- We cannot solve BG (synchronous) if more than a third of the generals are byzantine
- We can measure efficiency of agreement algorithms based on the
  - Number of (synchronous) rounds of communication needed
  - Number of messages
- More impossibility results
  - Read paper from FLP (Fischer, Lynch, Patterson), 1983