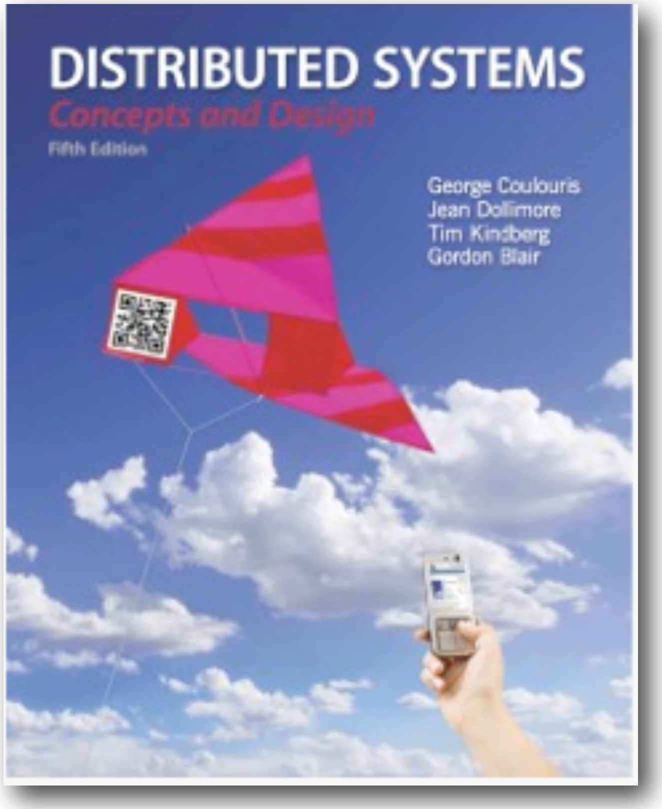
Coordination and Agreement

12.1 Introduction
12.2 Distributed Mutual Exclusion
12.4 Multicast Communication
12.3 Elections
12.5 Consensus and Related Problems



System Model

- Collection of processes p_i (i = 1, 2, ..., N)
- Processes communicate by message passing
- Communication is reliable
- Processes can fail: byzantine (arbitrary) process failures, crash failures

Worst possible failure: any type of error may occur!

Consensus Problem

- Informally: the processes propose values and have to agree on one among these values

- More formally:

  - every process p_i begins in the *undecided* state and *proposes* a single value $v_i \in D$, i = 1, 2, ..., M

  - each process then sets the value of a *decision variable* d_i

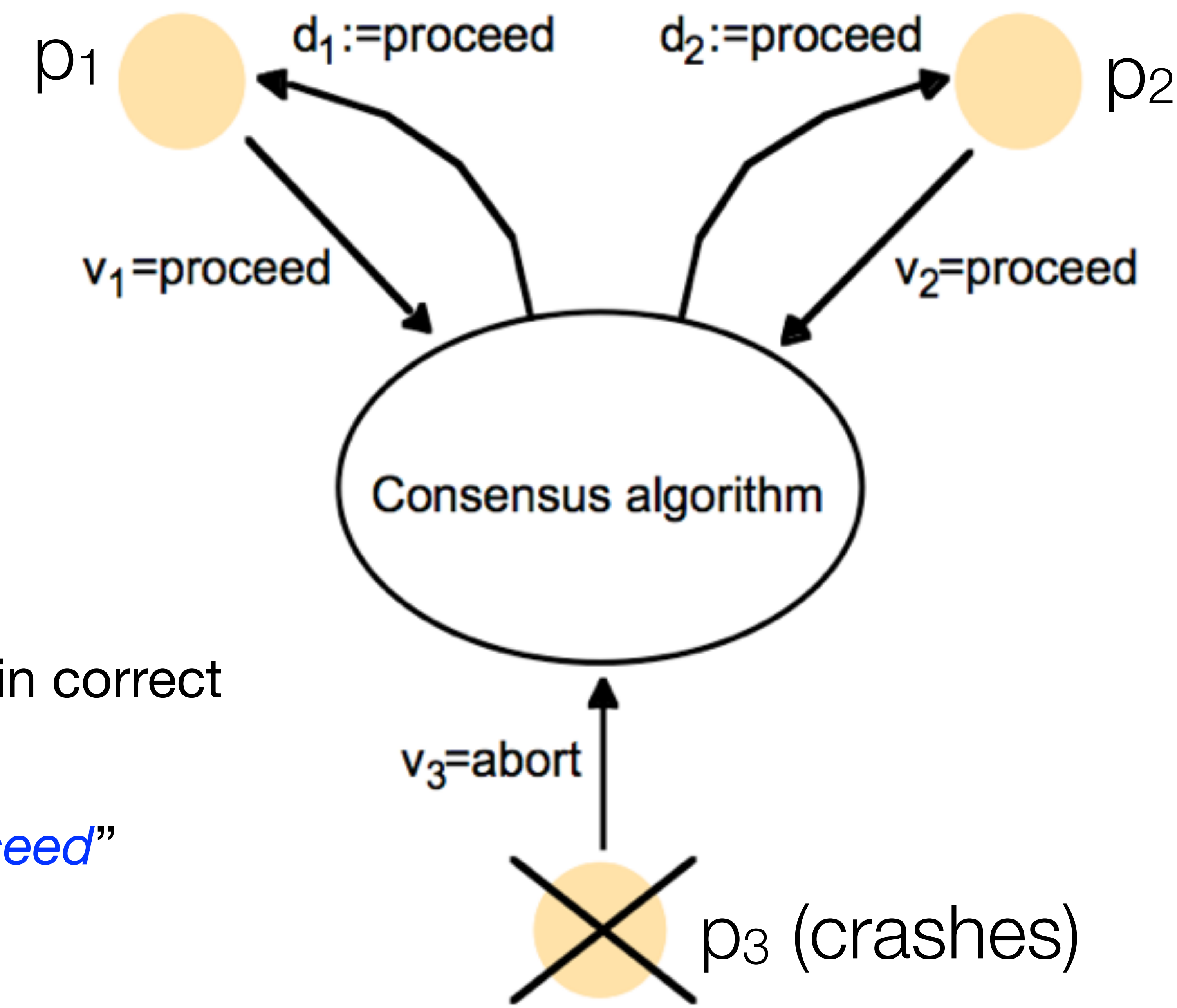
  - in doing so, the process enters the *decided* state, in which it may no longer change d_i

[Consensus] Example

- Consensus for 3 processes
- p_1 and p_2 propose "*proceed*"
- p_3 proposes "*abort*"

  but then crashes
- The two processes that remain correct

  (p_1 and p_2) each decide "*proceed*"



Requirements of a Consensus Algorithm

- The following conditions should hold for every execution of the algorithm:
  - Termination: eventually each correct process sets its decision variable
  - Agreement: the decision value of all correct processes is the same

    IF p_i and p_j are correct and have entered the *decided* state

    THEN $d_i = d_j$ (i, j = 1, 2, ..., N)
  - Integrity: if the correct processes all proposed the same value, then any correct process in the *decided* state has chosen that value

Solving Consensus in **Absence of Failures**

- Consider a system in which processes **cannot** fail
- Straightforward to solve consensus:
    - collect the processes into a group
    - each process *reliably* multicast its proposed value to the group
    - each process waits until it has collected all N values (including its own)
    - it then evaluates the function majority(v_1, v_2, ..., v_N), which returns:
        - the value that occurs most often among its arguments or
        - the special value $\perp \notin D$ if no majority exists

On Conditions

- Termination is guaranteed by the reliability of the multicast operation
- Agreement and integrity are guaranteed by:
  - the definition of majority
  - the integrity property of a reliable multicast (a correct process delivers a message m at most once)

Indeed:

- every process receives the same set of proposed values
- every process evaluates the same function on those values
- therefore they must all agree, and if every process proposed the same value, then they all decide on this value

Solving Consensus in Presence of **Crash Failures**

- The algorithm assumes a **synchronous** system where up to f of the N processes exhibit crash failures

- Idea:

  - each process collects proposed values from the other processes

  - the algorithm proceeds in **f + 1 rounds**, in each of which the correct processes *B-multicast* the values between themselves

  - by assumption, at most f processes may crash ==> at worst, all f crashes occurred during the rounds

  - the algorithm guarantees that at the end of the f + 1 rounds all the correct processes that have survived are in a position to agree

Consensus in a Synchronous System

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

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

$Values^r_i$ holds the set of proposed values known to process p_i at the beginning of round r

In round r $(1 \le r \le f + 1)$
B-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
$Values_i^{r+1} := Values_i^{r+1} \cup V_j$;
}

Assumption: duration of a round limited by setting a timeout based on the maximum time for a correct process to multicast a message

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

On Conditions

- Termination is obvious from the fact that the system is synchronous
- To check the correctness of the algorithm we must show that each process arrives at the same set of values at the end of the final round
- Agreement and integrity will then follow, because the processes apply the *minimum* function to this set
- So we have to prove that the algorithm is correct...

Proof of Correctness

- By contradiction: assume that two processes differ in their final set of values
  - Without loss of generality, some correct process p_i possesses a value v that another process p_j ($i \neq j$) does not possess
  - Situation possible only if a third process, p_k say, that managed to send v to p_i crashed before v could be delivered to p_j
  - In turn, any process sending v in the previous round must have crashed, to explain why p_k possesses v in that round but p_j did not receive it
  - Proceeding in this way, we have to posit at least one crash in each of the preceding rounds
  - BUT we have assumed that at most f crashes can occur, and there are f + 1 rounds! ==> contradiction

Lower Bound

Any algorithm to reach consensus despite up to f crash failures requires at least f + 1 rounds of message exchanges, no matter how it is constructed.

D. Dolev and H. R. Strong
Authenticated Algorithms for Byzantine Agreement
SIAM Journal of Computing 12(4), 656-66, 1983. DOI:10.1137/0212045.

This lower bound also applies in the case of *byzantine* failures.

Variant of Consensus: Byzantine Generals

- Three or more generals must agree to *attack* or *retreat*
  - One general, the commander, issues the order
  - Other generals, the lieutenants, must decide to attack or retreat
- One or more generals may be treacherous:
  - If the commander is treacherous, he proposes attacking to one general and retreating to another
  - If the lieutenant is treacherous, he tells one of his peers that the commander told him to attack and another that they are to retreat
- **Difference from consensus**: *a single process supplies a value that the others are to agree upon (instead of each of them proposing a value)*

[Byzantine Generals] Requirements

- Termination: eventually each correct process sets its decision variable
- Agreement: the decision value of all correct processes is the same
- Integrity: if the commander is correct, then all correct processes decide on the value that the commander proposed
- Further reading:

  L. Lamport, R. Shostak, and M. Pease.
  The Byzantine Generals Problem.
  ACM Transactions on Programming Languages and Systems (TOPLAS), 4(3), 382-401, 1982.

Variant of Consensus: Interactive Consistency

- Every process proposes a single value
- Goal: correct processes agree on a vector of values (decision vector), one for each process
  - Example: each of a set of processes want to obtain the same information about their respective states
- Requirements:
  - Termination: eventually each correct process sets its decision variable
  - Agreement: the decision vector of all correct processes is the same
  - Integrity: if p_i is correct, then all correct processes decide on v_i as the ith component of their vector

Relating Consensus to Other Problems

- All the problems concerned with making decisions in the context of arbitrary or crash failures
- We can sometimes generate solutions for one problem in terms of another
- Very useful property!! Because:
    - it increases our understanding of the problems
    - by reusing solutions we can potentially save on implementation effort and complexity

Suppose There Exists Solution to...

- $C_i(v_1, v_2, ..., v_N)$: returns the decision value of p_i in a run of the **solution to the consensus problem**, where $v_1, v_2, ..., v_N$ are the values that the processes proposed

- $BG_i(j, v)$: returns the decision value of p_i in a run of the **solution to the byzantine generals problem**, where p_j, the commander, proposes the value v

- $IC_i(v_1, v_2, ..., v_N)[\ j\ ]$: returns the jth value in the decision vector of p_i in a run of the **solution to the interactive consistency problem**, where $v_1, v_2, ..., v_N$ are the values that the processes proposed

Linking the Problems: IC from BG

- We can construct a solution to the Interactive Consistency (IC) problem from the Byzantine Generals (BG) problem as follows:
  - we run BG *N* times, once with each process p_i (i = 1, 2, ..., N) as the commander

$$IC_i(v_1, v_2, \ldots, v_N)[\, j \,] = BG_i(j, v_j)$$

(i, j = 1, 2, ..., N)

Linking the Problems: C from IC

- We can construct a solution to the Consensus (C) problem from the Interactive Consistency (IC) problem as follows:
  - we run IC to produce a vector of values at each process
  - then we apply an appropriate function (such as majority) on the vector's values to derive a single value

$$C_i(v_1, v_2, \ldots, v_N) = \mathit{majority}(IC_i(v_1, v_2, \ldots, v_N)[1], \ldots, IC_i(v_1, v_2, \ldots, v_N)[N])$$

$(i = 1, 2, \ldots, N)$

Linking the Problems: BG from C

- Show how it is possible to construct a solution to the Byzantine Generals (BG) problem from the Consensus (C) problem.

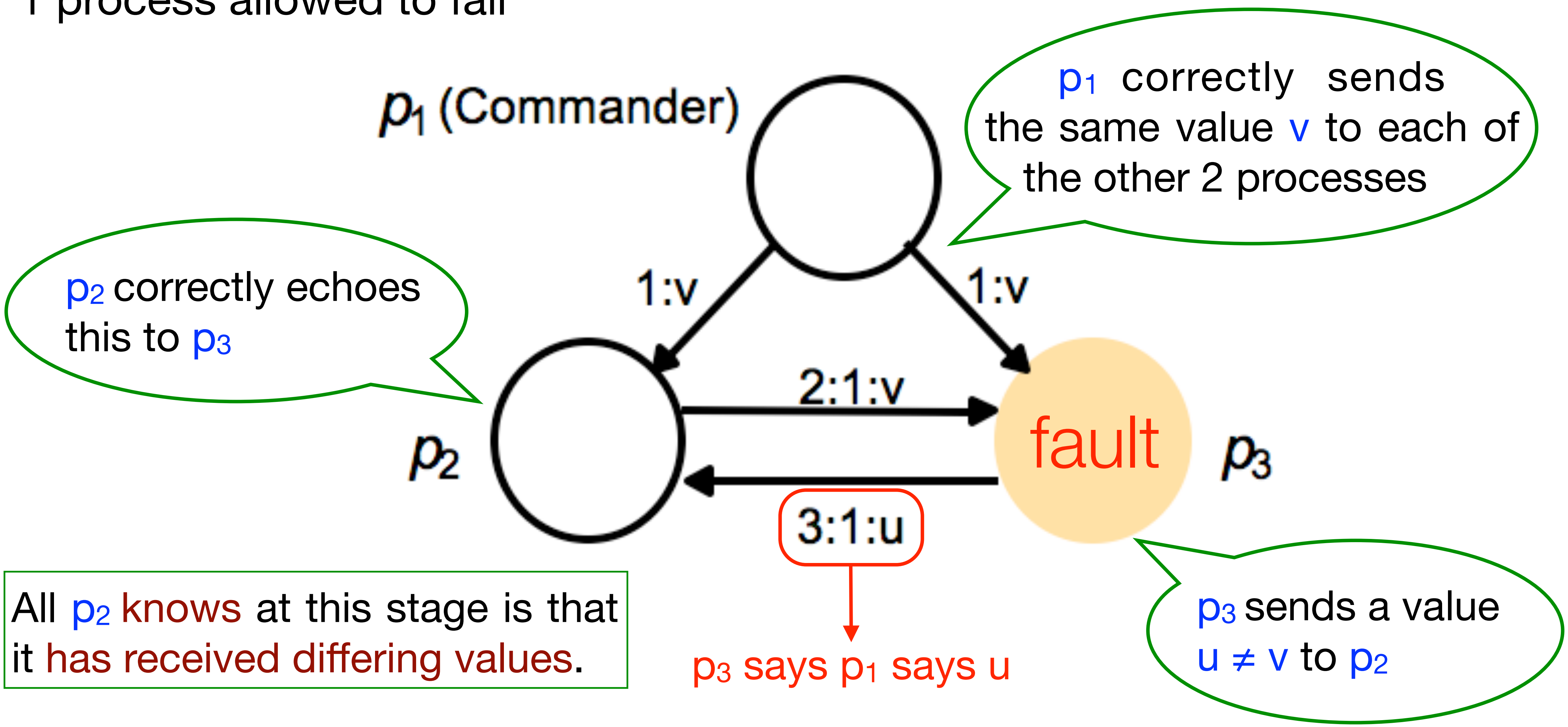
Byzantine Generals Problem in a Sync. System

- Up to f of the N processes can exhibit **arbitrary** (**byzantine**) failures:
  - a faulty process may send any message with any value at any time
  - it may omit to send any message
- Correct processes can detect the absence of a message through a timeout

  BUT they cannot conclude that the sender has crashed, since it may be silent for some time and then send messages again!
- Communication channels between pairs of processes are private and reliable

Impossibility with Three Processes: Scenario 1

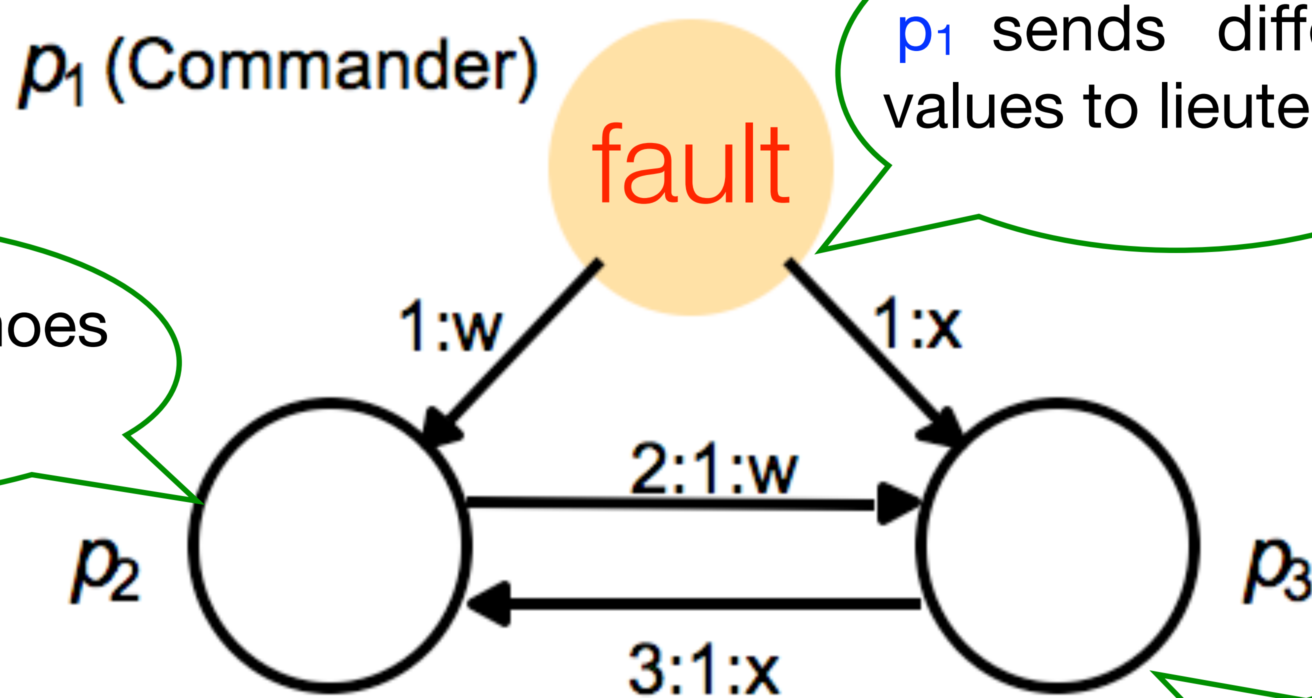
- 3 processes that send messages to one another
- 1 process allowed to fail



Impossibility with Three Processes: Scenario 2

- 3 processes that send messages to one another
- 1 process allowed to fail

p_1 sends different values to lieutenants



p_2 correctly echoes this to p_3

p_3 correctly echoes this to p_2

Both p_1 and p_2 have received two different messages

General Result: Impossibility with $N \leq 3f$

- M. Pease, R. Shostak and L. Lamport.
  Reaching agreement in the presence of faults.
  Journal of the ACM, 27(2), 228-34, 1980.

- They generalized the basic impossibility result for 3 processes, to prove that

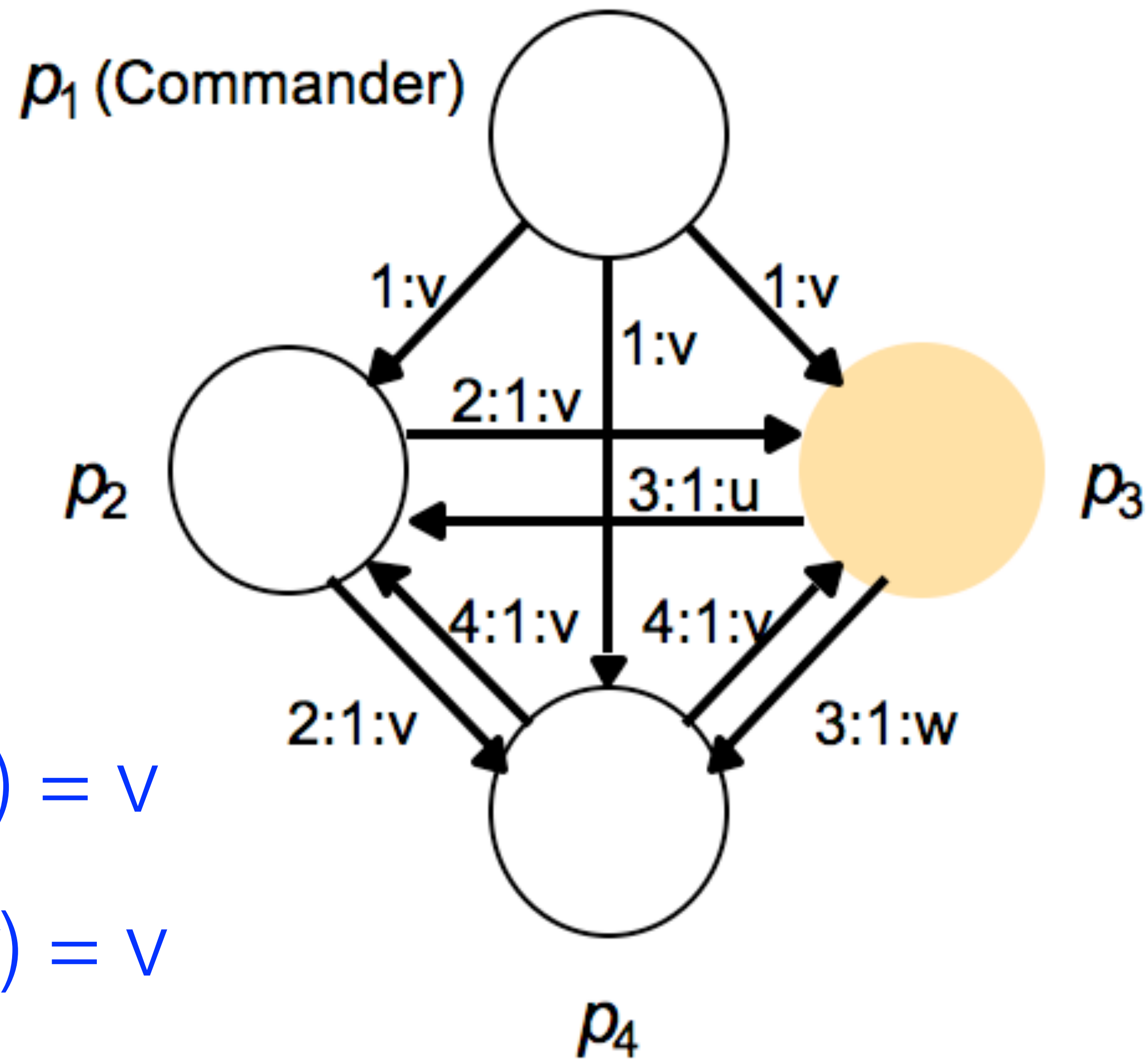
> no solution of the BG problem is possible if the total number of processes (N) is less than three times the number of failures (f), i.e., $N \leq 3f$

Solution with $N \geq 4$ and $f = 1$

- N.B. Have a look at the fully algorithm of Pease et al. that solves the BG problem in a synchronous system with $N \geq 3f + 1$
- In the special case $N \geq 4$ and $f = 1$, the correct generals can reach agreement in 2 rounds of messages:
  - **1st round**: the commander sends a value to each of the lieutenants
  - **2nd round**: each of the lieutenants sends the value it received to its peers
- Agreement is then reached using the function majority

Example: Scenario 1

The two correct lieutenant processes agree, deciding on the commander's value

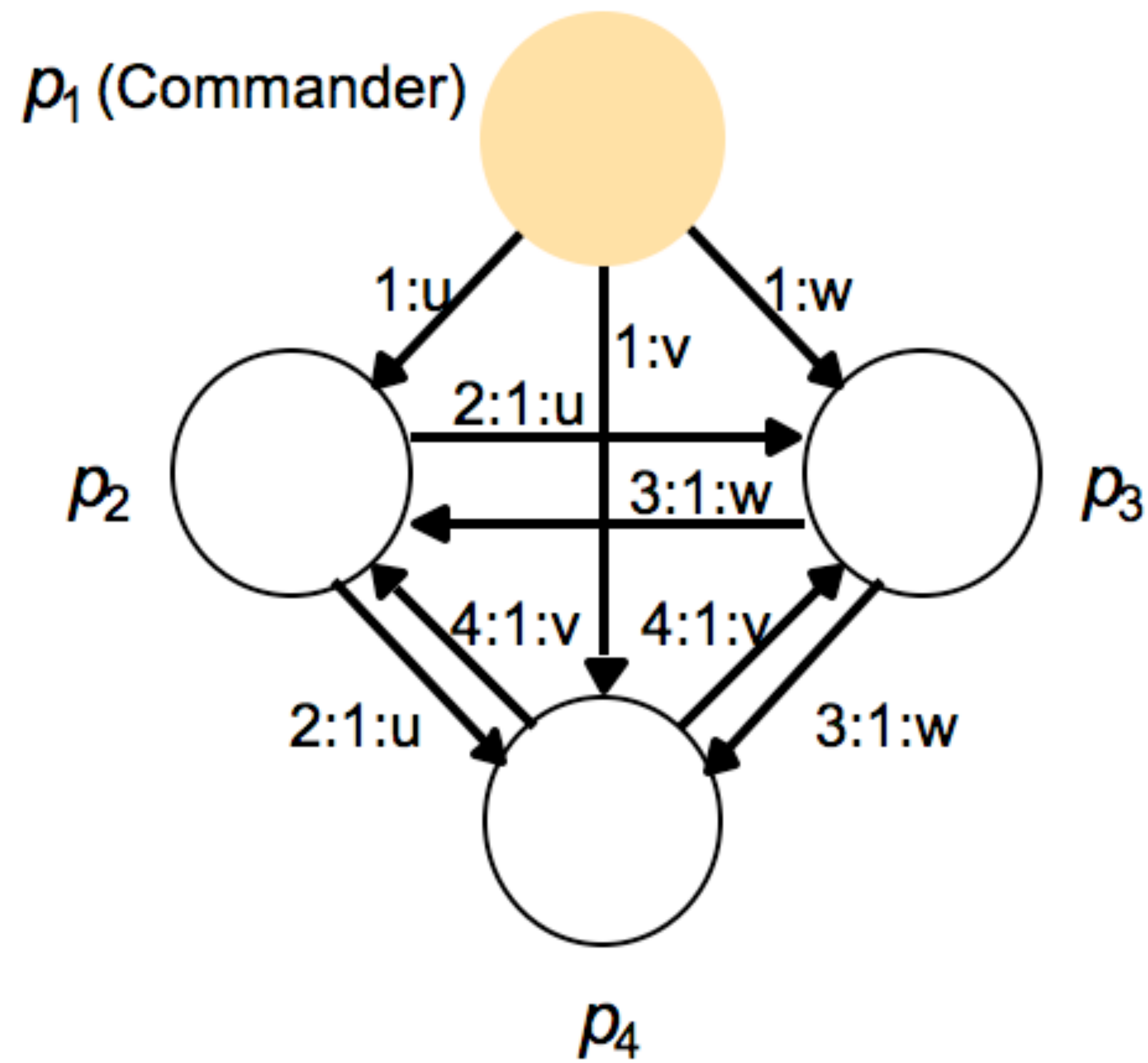


p_2 decides on majority(v, u, v) = v

p_4 decides on majority(v, v, w) = v

Example: Scenario 2

The commander is faulty, but the three correct processes agree



p_2, p_3 and p_4 decides on majority(u, v, w) = $\perp$

Impossibility in Asynchronous Systems

- All solutions we have seen so far are **limited to synchronous systems**

- Fischer et al [1985] proved that **no algorithm can *guarantee* to reach consensus in an asynchronous system, even with one process crash failure**

- Thus we immediately know from this result that there is no guaranteed solution in an asynchronous system to the BG and IC problems

- This impossibility is circumvented by masking faults or using failure detectors