Consensus Problem

Slides are based on the book chapter from Distributed Computing: Principles, Paradigms and Algorithms (Chapter 14) by Kshemkalyani and Singhal

Pradipta De
pradipta.de@sunykorea.ac.kr

What is consensus problem ?

- In a distributed system, reaching agreement is a fundamental problem
  - All processes decide on a common outcome

- Finds application in:
  - Leader Election
  - Mutual Exclusion
  - Commit/Abort in Distributed transactions

Consensus in Fault-free system

- Trivial to reach consensus in a fault free system

- 3-step process can ensure consensus
  - Collect information from all the processes
    - Use all-to-all broadcast
  - Arrive at a decision
    - Compute a common function, like min, max, etc on the collected values
  - Distribute the decision to all other nodes

- Overall ➔ broadcast-convergecast-broadcast

Requirements of Consensus Problem

- Agreement Condition
  - All (non-faulty) processes must agree on the same value

- Validity Condition
  - The value must be the value generated by a source process
  - Rules out trivial solutions
    - Value is a constant

- Termination Condition
  - Each (non-faulty) process must eventually decide on a value

Variants of Consensus Problem

- Agreement
  - Requires a designated process (source process) with an initial value to reach agreement with other processes about its initial value
  - Single source has initial value

- Consensus
  - Each process has an initial value and all the correct processes must agree on a single value

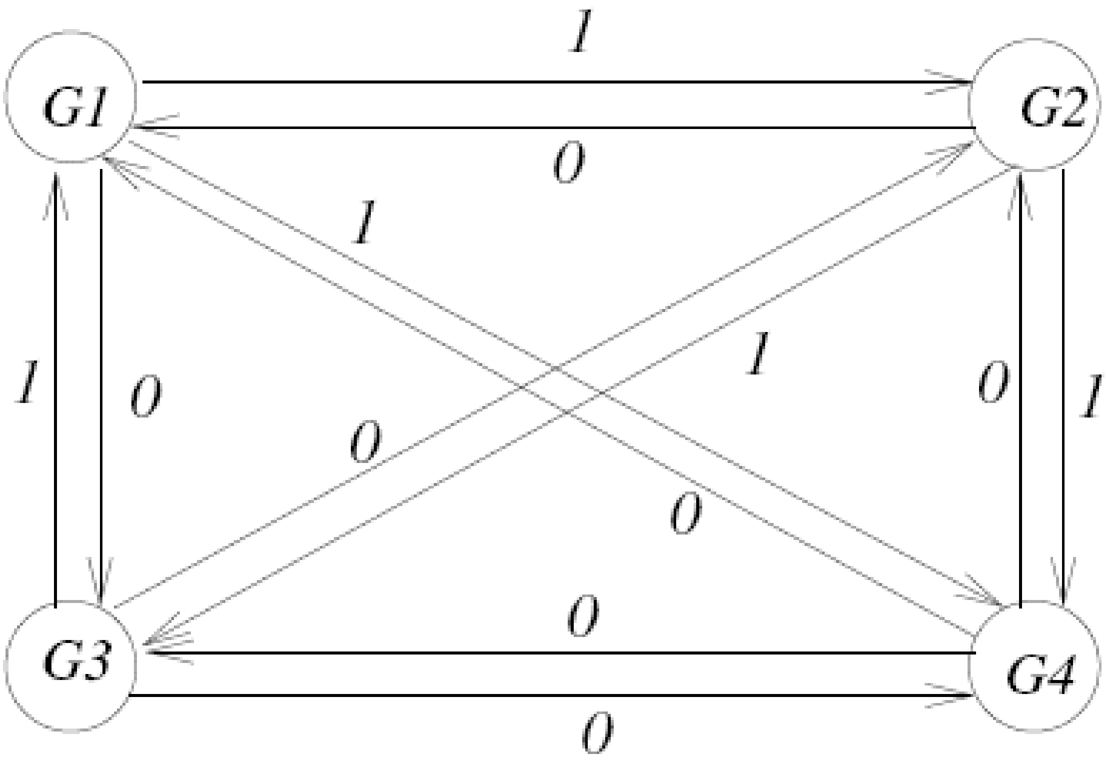
- Interactive Consensus problem
  - Each process has an initial value, and all correct processes must agree on a set of values, one for each process

Failure Models

- **Process failure models**
  - **Fail-stop**: stops execution; other processes learn about failed process
  - **Crash**: stops execution; other processes do NOT learn about failed process
  - **Receive omission**: receives only some of the messages
  - **Send omission**: sends only some of the messages
  - **General omission**: combination of receive and send omission
  - **Byzantine (malicious) failure with authentication**: process misbehaves in any manner, including sending fake messages; can identify source of message
  - **Byzantine (malicious) failure without authentication**: misbehaving process with source not identifiable

- **Link/Communication failure models**
  - **Crash:** links stop carrying messages
  - **Omission:** links drop messages
  - **Byzantine**: links exhibit arbitrary behavior ➔ creates and alters messages

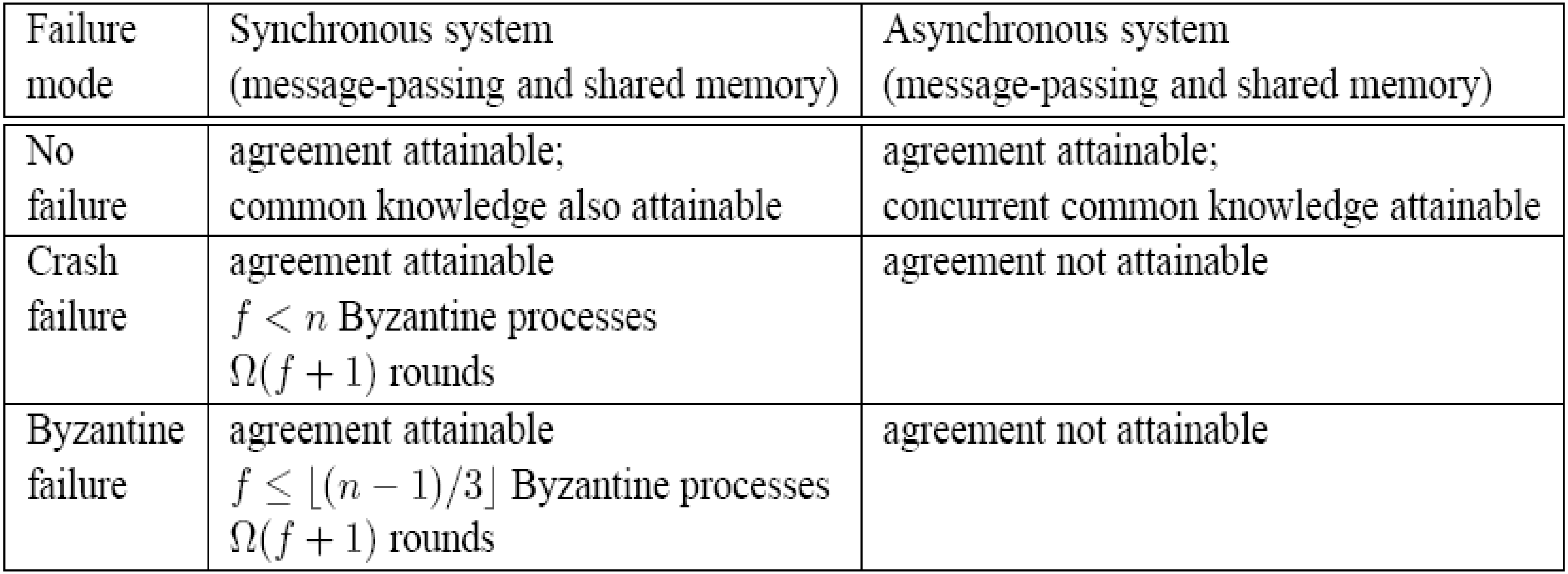
Byzantine General’s Problem



Link Failures: messengers can get lost or captured

Process Failures: generals can be traitors and send incorrect messages (leads to the Byzantine Agreement problem)

Results on Byzantine Agreement



Failure mode	Synchronous system (message-passing and shared memory)	Asynchronous system (message-passing and shared memory)
No failure	agreement attainable; common knowledge also attainable	agreement attainable; concurrent common knowledge attainable
Crash failure	agreement attainable $f < n$ Byzantine processes $\Omega(f+1)$ rounds	agreement not attainable
Byzantine failure	agreement attainable $f \leq \lfloor (n-1)/3 \rfloor$ Byzantine processes $\Omega(f+1)$ rounds	agreement not attainable

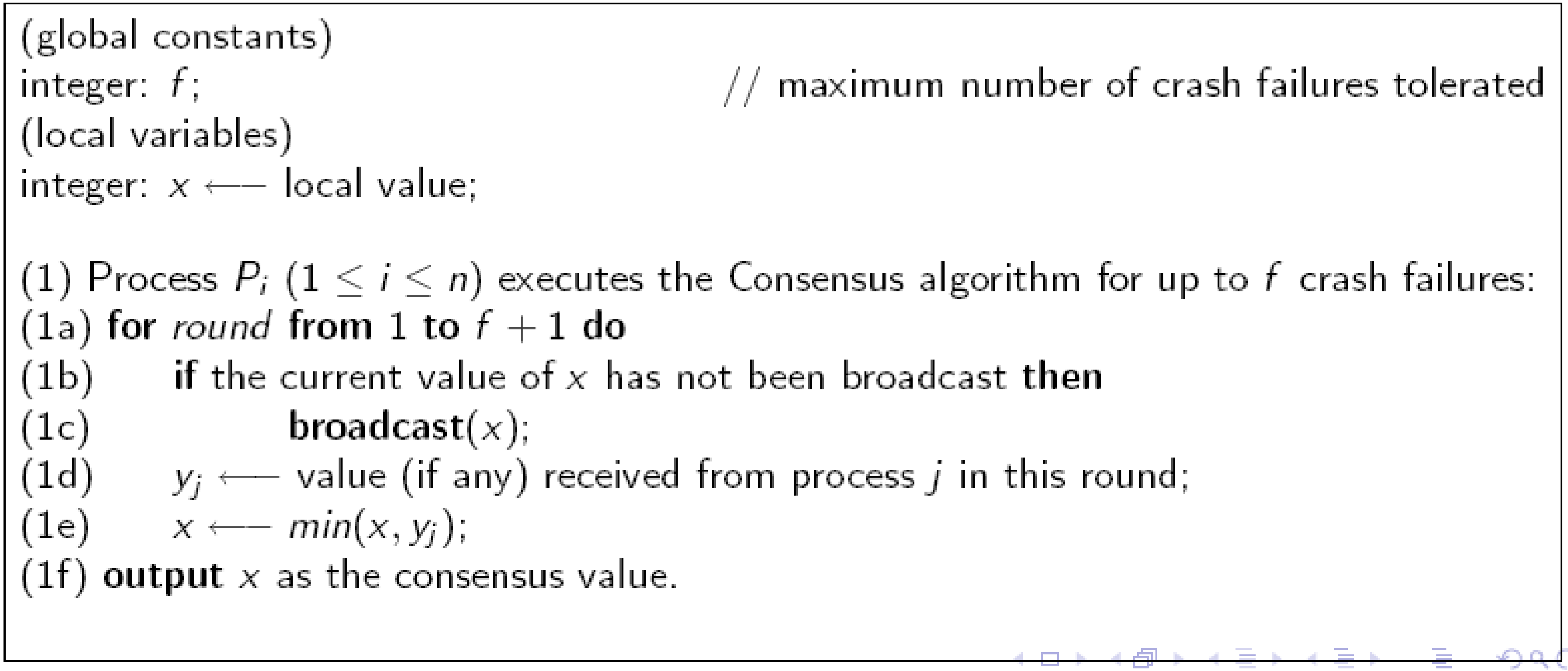
Outline of Key Topics

- Consensus in synchronous systems
  - In presence of crash failures
  - In presence of byzantine failures

- Consensus in asynchronous systems
  - Impossibility result: deterministic solution cannot be reached in asynchronous system even in presence of a single fault (one process crash)

- Protocols to reach consensus
  - 2PC; 3PC; Paxos

Consensus for crash failure (synchronous MP system)



```
(global constants)
integer: $f$;                                // maximum number of crash failures tolerated
(local variables)
integer: $x \longleftarrow$ local value;

Process $P_i$ $(1 \leq i \leq n)$ executes the Consensus algorithm for up to $f$ crash failures:
for round from 1 to $f + 1$ do
    if the current value of $x$ has not been broadcast then
        broadcast($x$);
    $y_j \longleftarrow$ value (if any) received from process $j$ in this round;
    $x \longleftarrow min(x, y_j)$;
output $x$ as the consensus value.
```

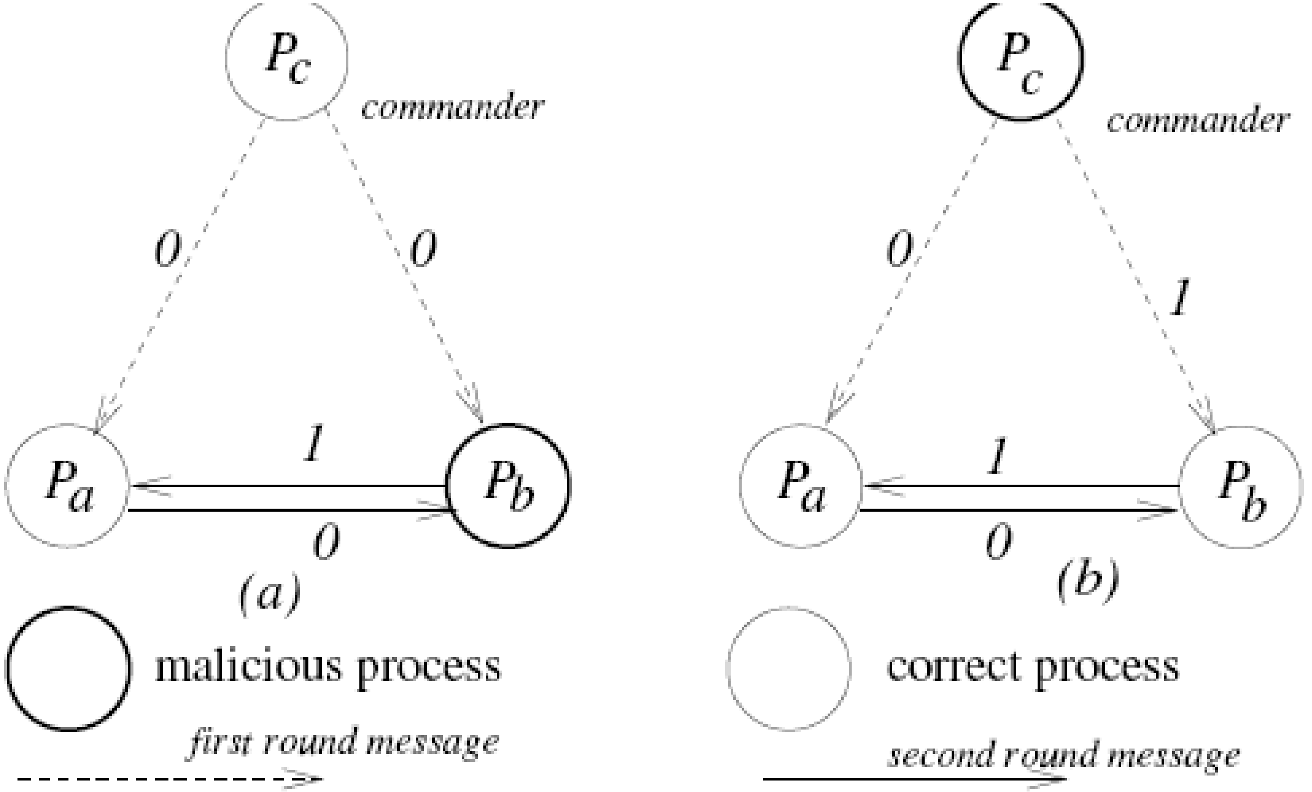
Termination: finishes in f+1 rounds
Validity: processes do not send fictitious values (not Byzantine); if all inputs are same, then that will be only value
Agreement: In (f+1) rounds, at least one round where no process fails

Consensus for crash failure (synchronous MP system)

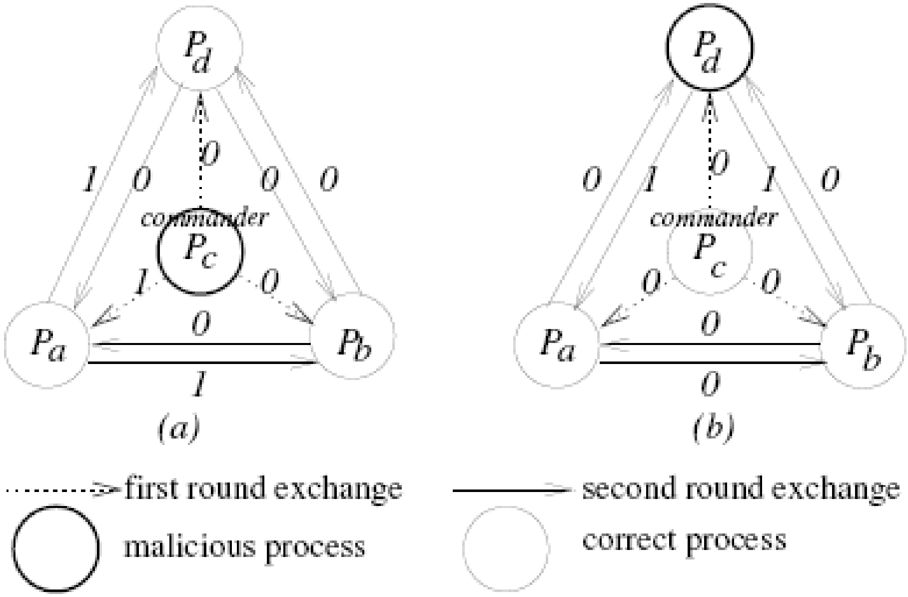
- There are f+1 rounds
- Number of messages is at most $O(n^2)$ in each round
  - Total messages $O((f+1).n^2)$
- Can there be an early stopping algorithm ?
  - If there are f` < f faults, then can terminate in f`+1 rounds

Consensus for Byzantine Failures (synchronous MP system)

- Links are reliable, but processes are malicious
- In a system with 3 processes, if 1 process is byzantine then consensus problem is unsolvable
  - Generalization: no solution if n < (3f + 1), with f Byzantine processes among n processes ➔ n > 3f



Consensus for Byzantine Failures (synchronous MP system)



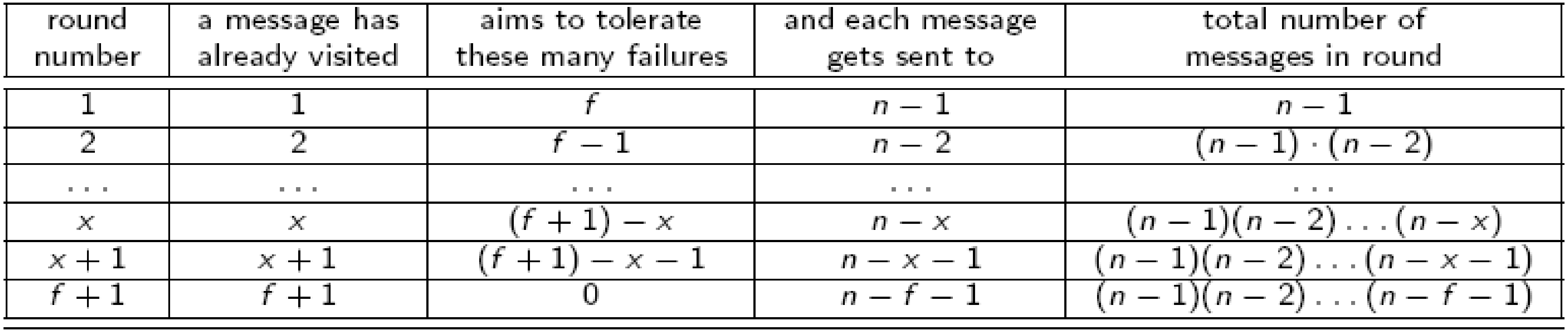
Now, f=1, n=4
A non-malicious process can determine unambiguously what is the correct value.

At the end of 2nd round, a lieutenant takes the majority of the values it received
- Directly from the commander in first round, and
- From the other two lieutenants in the second round

Consensus for Byzantine Failures (synchronous MP system)

- Recursive algorithm, called Oral Messages, OM(k), by Lamport for Byzantine agreement problem
- General Idea:
  - Commander i sends out value v to all lieutenants
  - If m>0, then every lieutenant j ≠ i, after receiving v, acts as a commander, and initiates OM(m-1) with everyone except i
  - Every lieutenant collects (n-1) values to pick majority value
    - (n-2) values from the lieutenants using OM(m-1),
    - One direct value from commander

Consensus for Byzantine Failures (synchronous MP system)



round number	a message has already visited	aims to tolerate these many failures	and each message gets sent to	total number of messages in round
1	1	f	$n-1$	$n-1$
2	2	$f-1$	$n-2$	$(n-1)\cdot(n-2)$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
x	x	$(f+1)-x$	$n-x$	$(n-1)(n-2)\ldots(n-x)$
$x+1$	$x+1$	$(f+1)-x-1$	$n-x-1$	$(n-1)(n-2)\ldots(n-x-1)$
$f+1$	$f+1$	0	$n-f-1$	$(n-1)(n-2)\ldots(n-f-1)$

Number of rounds = $(f+1)$

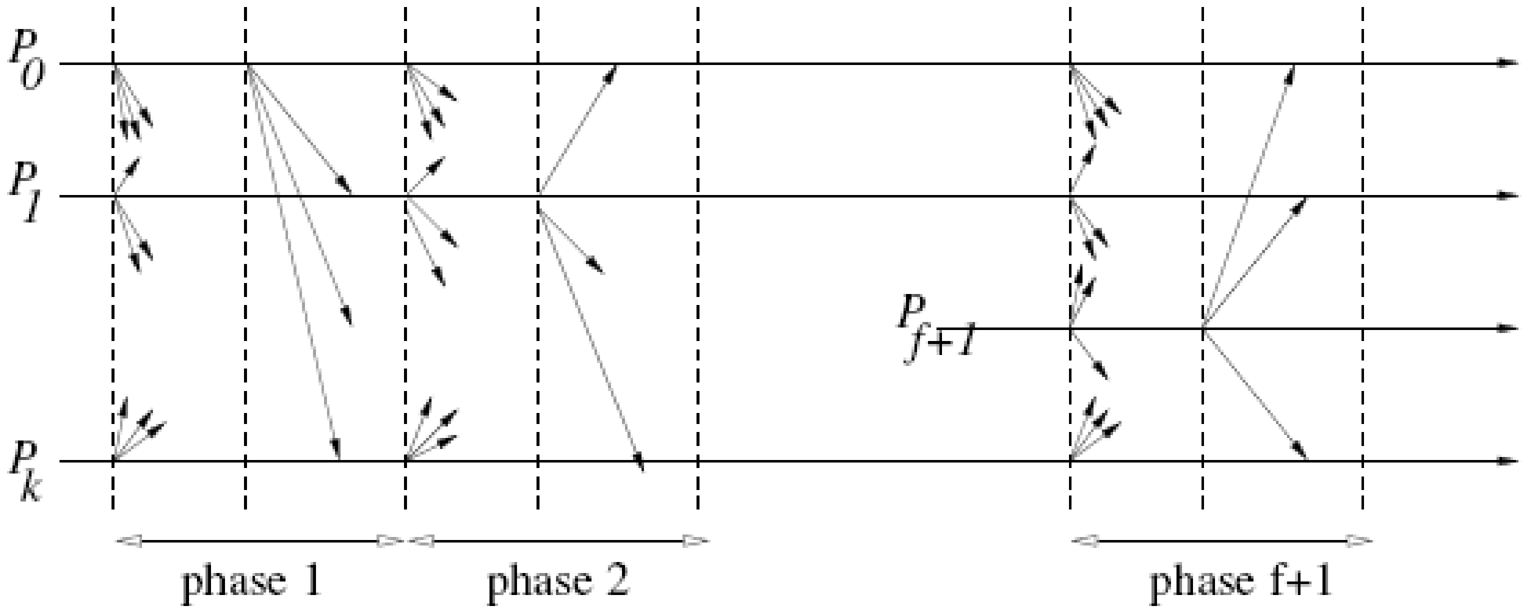
Number of messages exchanges: $O(n^f)$

Byzantine Agreement continued

- Phase-King algorithm improves the message complexity
  - Can tolerate only f < ceil(n/4) faults

- Operates in f+1 phases
  - Each phase has two rounds
  - A unique process plays the role of leader in each round

Byzantine Agreement continued



Round 1: all processes send their estimate to all other processes

messages = n-1

Round 2: Phase-king arrives at an estimate based on Round 1 received values, and broadcasts the estimate to all others

messages = n-1

Total messages: (f+1)n(n-1) + (f+1)(n-1) = f+1[(n-1)(n+1)]

Agreement in Asynchronous Systems

- FLP (Fischer, Lynch, Paterson) Theorem
- There is no deterministic protocol that solves the consensus problem in a message passing (or shared memory) asynchronous system in which at most one process may fail

- In an asynchronous system, a process p cannot tell whether a non-responsive process q has crashed or it is slow
  - P may have to wait forever
  - P may decide, but then q comes up with a different value

FLP Proof: Model

- Asynchronous system with n processes
- Each process has one-bit register {0,1}
- Processes communicate by exchanging messages (p,m), where p is destination process, m is message
- Messages are pushed into a global message buffer
- Two primitives:
  - Send (p,m) : places m for p in the message buffer
  - Receive (p): deletes some message from msg buffer, and returns m, or returns ϕ
- Every message sent will be eventually delivered
- Failure is one failure per execution ➔ n-1 processes must decide without waiting for nth since it may have failed

FLP Proof: Definitions

- Configuration (C) : internal state of each process + state of msg buffer
- Initial configuration: state of process in the beginning + empty msg buffer
- Step: Takes one config to another
  - Two phases: fetches message from buffer; depending on process’ internal state and m, changes state;
- Event: pair (p,m) which determines a step
- Schedule: finite or infinite seq of events
- Run: Associated seq of steps

- A run is unacceptable if every process takes infinitely many steps without deciding

FLP Proof: Core Idea

- Explain a strategy that allows the adversary to steer the execution away from any configuration in which the processes reach agreement.

OR,

- For any agreement protocol, there always exists an unacceptable run

FLP Proof: Classify Configuration

- A decision state is **bivalent**, if starting from that state, there exists two distinct executions leading to two distinct decision values **0** or **1.**
- Otherwise it is **univalent.**
  - 0-valent or 1-valent

FLP Proof
Initial configuration is bivalent

- I_j is the initial config in which first j inputs are 1
  - I_0 is 0-valent and I_n is 1-valent
- 1-crash failure is allowed
- For proof, by contradiction, suppose no bivalent initial configuration exists
- Let k be the smallest index such that I_k is 1-valent
  - I_{k-1} is 0-valent
- If pk crashes before taking any step, then the algorithm reaches decision where there is no step of pk, and still decision is reached.
  - Same argument also holds for I_{k-1} processes
- This leads to contradiction

FLP Proof

- Start with initial bivalent config
- Pick any set of steps that leads to another bivalent config
  - Can a “critical step” exist that takes the config from bivalent to univalent config
- Continue this process ➔ the algorithm cannot decide (an unacceptable run)

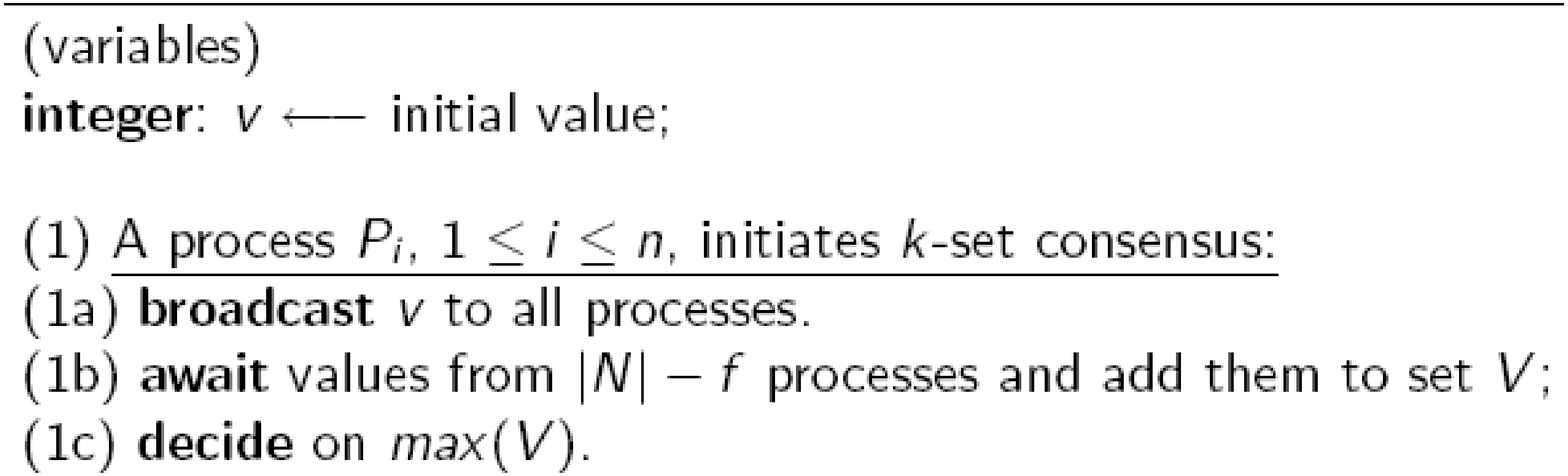
Circumventing FLP

- Weaken termination condition
  - Use randomization to terminate with high probability
  - Guarantee termination only during periods of synchrony
- Weaken agreement
  - K-set agreement
  - Approximate agreement
    - Agreement with real-valued small positive tolerance

- Constrain input values
  - Specify set of input values for which agreement is possible
- Strengthen system model
  - Introduce failure detectors

K-set consensus

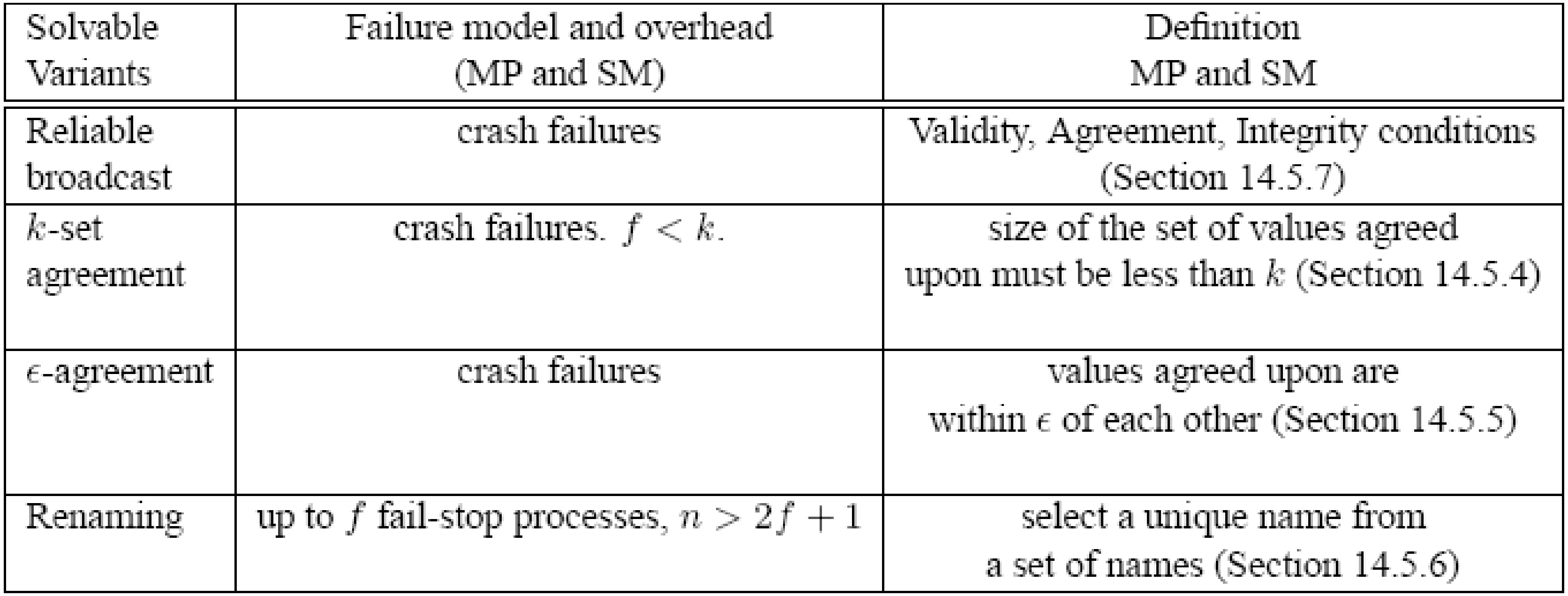
- Specification
  - K-agreement: all non-faulty processes must make a decision, and the set of values decided on can contain up to k values
  - Validity: value must be proposed by some process
  - Termination: non-faulty process must eventually decide



```
(variables)
integer: $v \longleftarrow$ initial value;

(1) A process $P_i$, $1 \leq i \leq n$, initiates $k$-set consensus:
(1a) broadcast $v$ to all processes.
(1b) await values from $|N| - f$ processes and add them to set $V$;
(1c) decide on $max(V)$.
```

Solving Agreement in Asynchronous System



Solvable Variants	Failure model and overhead (MP and SM)	Definition MP and SM
Reliable broadcast	crash failures	Validity, Agreement, Integrity conditions (Section 14.5.7)
k-set agreement	crash failures. $f < k$.	size of the set of values agreed upon must be less than k (Section 14.5.4)
ϵ-agreement	crash failures	values agreed upon are within ϵ of each other (Section 14.5.5)
Renaming	up to f fail-stop processes, $n > 2f + 1$	select a unique name from a set of names (Section 14.5.6)