Byzantine Agreement

Dr. Rajiv Misra
Associate Professor
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

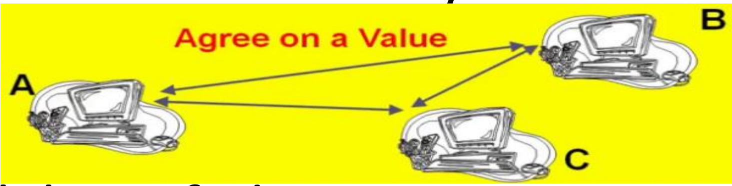
Preface

Content of this Lecture:

- In this lecture, we will discuss about '**Agreement Algorithms for Byzantine processes**'.
- This lecture first covers different forms of the **'consensus problem'** then gives an overview of what forms of consensus are solvable under different failure models and different assumptions on the synchrony/asynchrony.
- Also covers agreement in the category of:
  (i) Synchronous message-passing systems with failures and
  (ii) Asynchronous message-passing systems with failures.

Introduction

- **Agreement among the processes** in a distributed system is a fundamental requirement for a wide range of applications.
- Many forms of coordination require the processes to exchange information to negotiate with one another and eventually reach a common understanding or agreement, before taking application-specific actions.
    - **A classical example is that of the commit decision in database systems**, wherein the processes collectively decide whether to commit or abort a transaction that they participate in.



- In this lecture, we will study the feasibility of designing algorithms to reach agreement under various system models and failure models, and, where possible, examine some representative algorithms to reach agreement.

Classification of Faults: Overview

- **Based on components that failed**

– Program / process

– Processor / machine

– Link

– Storage

- **Based on behavior of faulty component**

– Crash – just halts

–Fail stop – crash with additional conditions

– Omission– fails to perform some steps

– Byzantine– behaves arbitrarily

– Timing – violates timing constraints

Classification of Tolerance: Overview

- **Types of tolerance:**

– **Masking –** System always behaves as per specifications even in presence of faults.
–**Non-masking–** System may violate specifications in presence of faults. Should at least behave in a well-defined manner.

- **Fault tolerant system should specify:**

– Class of faults tolerated
– What tolerance is given from each class

Measuring Reliability and Performance

- **Distributed systems:**
  - Improve performance
  - Improve reliability
- Or do they? Need to measure to know.
- Need a vocabulary

SLIs, SLOs, SLAs, TLAs

- SLI = Service Level Indicator
  ⇒ What you are measuring
- SLO = Service Level Objective
  ⇒ How good should it be?
- SLA = Service Level Agreement
  ⇒ SLO + consequences
- TLA = Three Letter Acronym

Why study SLIs, SLOs, and SLAs?

- If you measure it, you can improve it
- Learn what matters
  - Don't waste time on things that don't matter!
- Reliability promises are part of business

Reading an SLA

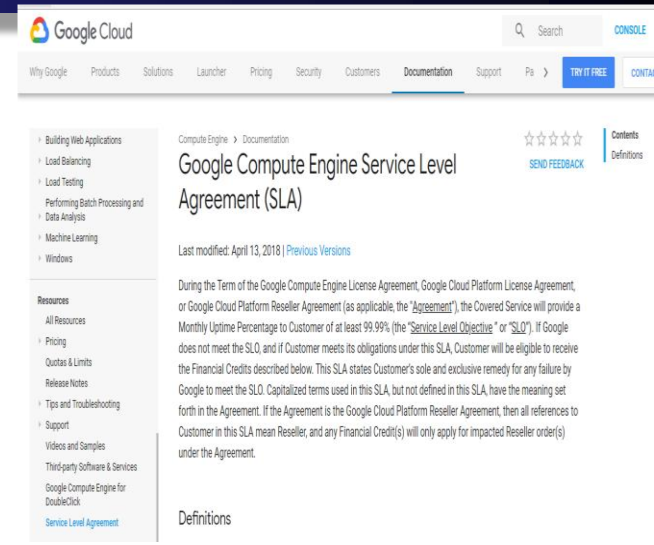
- **"I promise 99% uptime"**
- How often do you check if your system is up?
  - Sampling frequency
- What does it mean to be "up"?
  - Domain of responsibility
- Over what time interval do you promise 99% uptime?
  - Measurement interval

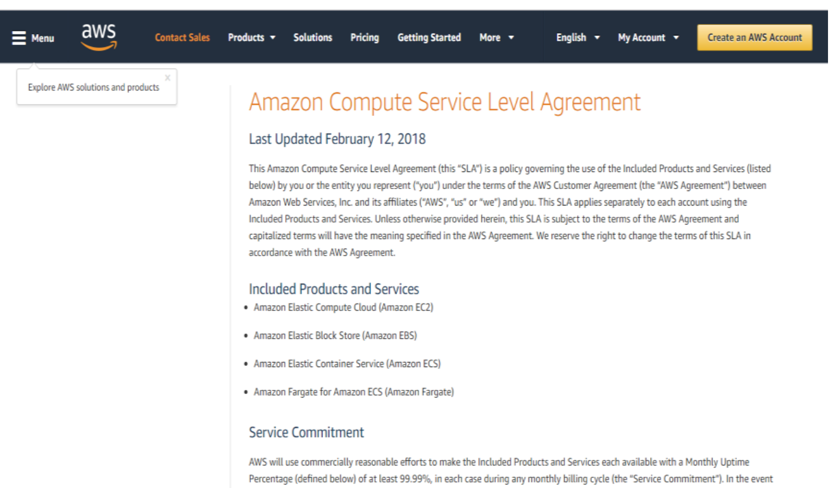
How many nines?

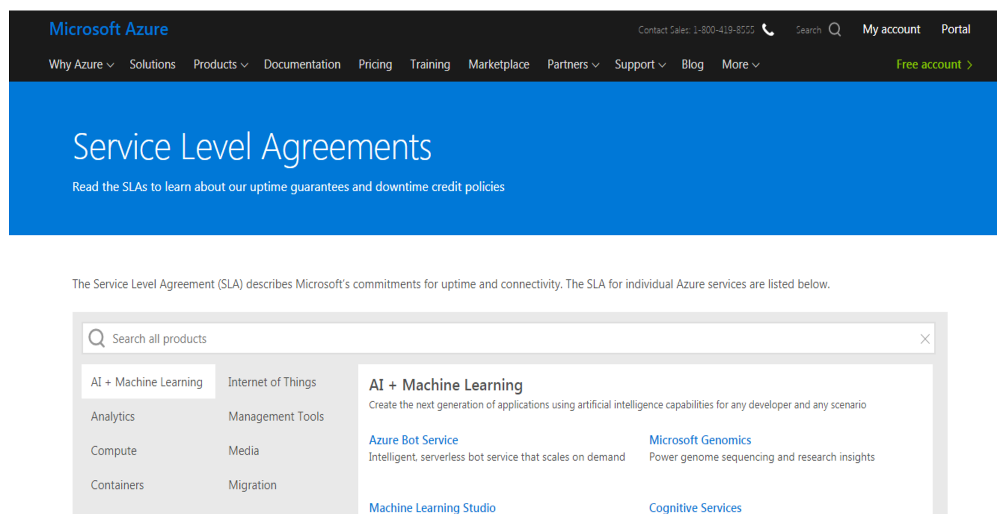
Nines	Uptime	Downtime/month
1	90%	3 days
2	99%	7 hours
3	99.9%	43 minutes
4	99.99%	4 minutes
5	99.999%	25 seconds (5m/year)

Cloud VM providers

- **Consider Microsoft Azure, Amazon EC2, Google GCE (Google Compute Engine)**
- Promise 99.95% uptime (22 minutes downtime/month)
  - Better than my net connection… right?
- 1-minute sampling frequency
  - GCE doesn't count <5 minute outages







What does the SLA imply for provider?

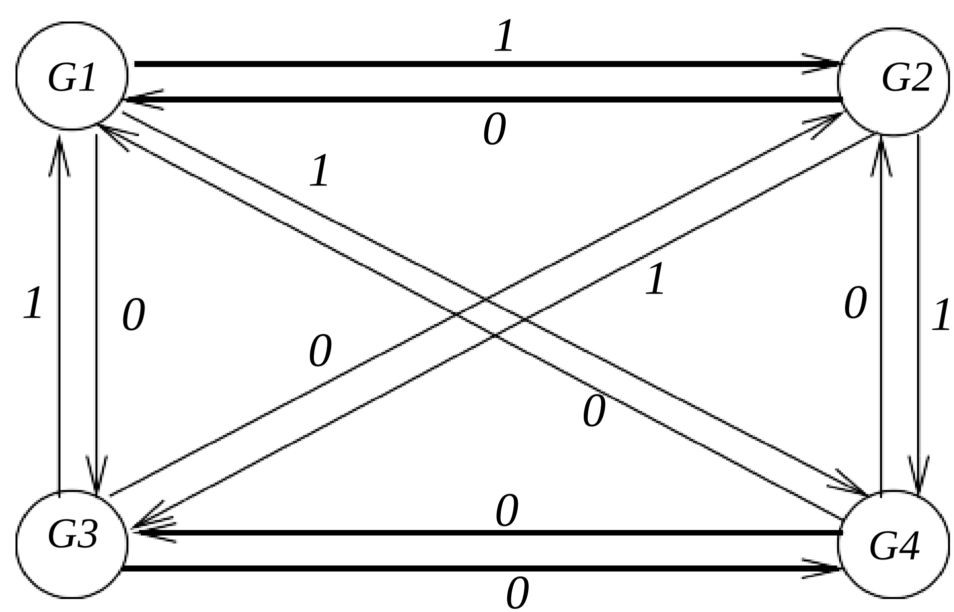
- **99.95% (or 22 minutes/month downtime) means either:**
  - They rarely expect their hardware or software to fail
  - When it fails they think they can fix it quickly

What does the SLA imply for you?

- **SLA requires you to have:**
  - Multiple VMs
  - Over multiple failure domains
  - Automatic failover
  - Monitoring
  - Tolerance of planned outages
  - Automatic machine provisioning (GCE)

Assumptions

1. Failure models
2. Synchronous/ Asynchronous communication
3. Network connectivity
4. Sender identification
5. Channel reliability
6. Authenticated vs. Non-authenticated messages
7. Agreement variable



1) Failure models

- A failure model specifies the manner in which the component(s) of the system may fail.

- There exists a rich class of **well-studied failure models**. The various process failure models are: (i) Fail-stop, (ii) Crash, (iii) Receive omission, (iv) Send omission, (v) General omission, and (vi) Byzantine or malicious failures

- Among the *n* processes in the system, at most *f* processes can be faulty. A faulty process can behave in any manner allowed by the failure model assumed.

Type of Process Failure Models

i. **Fail-stop:** In this model, a properly functioning process may fail by stopping execution from some instant thenceforth. Additionally, other processes can learn that the process has failed.

ii. **Crash:** In this model, a properly functioning process may fail by stopping to function from any instance thenceforth. Unlike the fail-stop model, other processes do not learn of this crash.

iii. **Receive omission:** A properly functioning process may fail by intermittently receiving only some of the messages sent to it, or by crashing.

iv. **Send omission:** A properly functioning process may fail by intermittently sending only some of the messages it is supposed to send, or by crashing.

Contd...

v. **General omission:** A properly functioning process may fail by exhibiting either or both of send omission and receive omission failures.

vi. **Byzantine or malicious failure:** In this model, a process may exhibit any arbitrary behavior and no authentication techniques are applicable to verify any claims made.

2) Synchronous/Asynchronous Computation

-Synchronous Computation:

i.Processes run in lock step manner **[Process receives a message sent to it earlier, performs computation and sends a message to other process.]**
ii.Step of Synchronous computation is called '***round***'

-Asynchronous Computation:

i.Computation does not proceed in lock step.
ii.Process can send receive messages and perform computation at any time.

3) Network connectivity

The system has **full logical connectivity**, i.e., each process can communicate with any other by direct message passing.

4) Sender identification

A process that receives a message always **knows the identity** of the sender process.

5) Channel reliability

The **channels are reliable**, and only the processes may fail (under one of various failure models). This is a simplifying assumption in our study.

6) Authenticated vs. Non-authenticated messages

- In this study, we will be **dealing only with unauthenticated messages**.

- With **unauthenticated messages**, when a faulty process relays a message to other processes, (i) it can forge the message and claim that it was received from another process, and (ii) it can also tamper with the contents of a received message before relaying it.

- Using **authentication** via techniques such as digital signatures, it is easier to solve the agreement problem because, if some process forges a message or tampers with the contents of a received message before relaying it, the recipient can detect the forgery or tampering.

7) Agreement variable

- The agreement variable **may be boolean or multivalued**, and need not be an integer.

- When studying some of the more complex algorithms, we will use a boolean variable.

- This simplifying assumption does not affect the results for other data types, but helps in the abstraction while presenting the algorithms.

Performance Aspects of Agreement Protocols

Few Performance Metrics are as follows:

(i) Time: No of rounds needed to reach an agreement

(ii) Message Traffic: Number of messages exchanged to reach an agreement.

(iii) Storage Overhead: Amount of information that needs to stored at processors during execution of the protocol.

Problem Specifications

1. Byzantine Agreement Problem (single source has an initial value)

Agreement: All non-faulty processes must agree on the same value.

Validity: If the source process is non-faulty, then the agreed upon value by all the non-faulty processes must be the same as the initial value of the source.

Termination: Each non-faulty process must eventually decide on a value.

2. Consensus Problem (all processes have an initial value)

Agreement: All non-faulty processes must agree on the same (single) value.

Validity: If all the non-faulty processes have the same initial value, then the agreed upon value by all the non-faulty processes must be that same value.

Termination: Each non-faulty process must eventually decide on a value.

Contd...

3. Interactive Consistency Problem (all processes have an initial value)

Agreement: All non-faulty processes must agree on the same array of values $A[v_1 \ldots v_n]$.

Validity: If process i is non-faulty and its initial value is v_i, then all non-faulty processes agree on v_i as the i th element of the array A. If process j is faulty, then the non-faulty processes can agree on any value for $A[j]$.

Termination: Each non-faulty process must eventually decide on the array A.

Equivalence of the Problems

- The three problems defined above are equivalent in the sense that a solution to any one of them can be used as a solution to the other two problems. This equivalence can be shown using a reduction of each problem to the other two problems.
- **If problem A is reduced to problem B, then a solution to problem B can be used as a solution to problem A in conjunction with the reduction.**
- Formally, the **difference between the agreement problem and the consensus problem** is that, in the agreement problem, a single process has the initial value, whereas in the consensus problem, all processes have an initial value.
- However, the two terms are used interchangeably in much of the literature and hence we shall also use the terms interchangeably.

Overview of Results

- **Table 10.1** gives an overview of the results and lower bounds on solving the consensus problem under different assumptions.

- It is worth understanding the relation between the consensus problem and the problem of attaining common knowledge of the agreement value. For the **“no failure”** case, consensus is attainable.

- Further, in a synchronous system, common knowledge of the consensus value is also attainable, whereas in the asynchronous case, concurrent common knowledge of the consensus value is attainable.

Overview of Results

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 $f < n$ 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

Table 10.1: Overview of results on agreement. f denotes number of failure-prone processes. n is the total number of processes.

In a failure-free system, consensus can be attained in a straightforward manner

Contd...

- **Consensus is not solvable in asynchronous systems** even if one process can fail by crashing.

- **Figure 10.1** shows further how asynchronous message-passing systems and shared memory systems deal with trying to solve consensus.

Solvable Variants of the Consensus Problem in Asynchronous Systems

Circumventing the impossibility results for consensus in asynchronous systems

Message-passing

k **set consensus**

epsilon− **consensus**

Renaming

Reliable broadcast

Shared memory

k **set consensus**

epsilon− **consensus**

Renaming

using atomic registers and atomic snapshot objects constructed from atomic registers

Consensus

using more powerful objects than atomic registers. This is the study of universal objects and universal constructions.

Weaker Consensus Problems in Asynchronous System

Consensus Problem	Description
Terminating reliable broadcast	It states that a correct process always gets a message even if the sender crashes while sending. If the sender crashes while sending the message, the message may be a null message but it must be delivered to each correct process.
k**-set consensus**	It is solvable as long as the number of crash failures f is less than the parameter k. The parameter k indicates that the non-faulty processes agree on different values, as long as the size of the set of values agreed upon is bounded by k.
Approximate agreement	Like k-set consensus, approximate agreement also assumes the consensus value is from a multi-valued domain. However, rather than restricting the set of consensus values to a set of size k, ϵ-approximate agreement requires that the agreed upon values by the non-faulty processes be within ϵ of each other.
Renaming problem	It requires the processes to agree on necessarily distinct values.
Reliable broadcast	A weaker version of reliable terminating broadcast(RTB), namely reliable broadcast, in which the termination condition is dropped, is solvable under crash failures.

Contd...

- To circumvent the impossibility result, weaker variants of the consensus problem are defined in **Table 10.2**.

- The overheads given in this table are for the algorithms described.

Some Solvable Variants of the Consensus Problem in Asynchronous Systems

Solvable Variants	Failure model and overhead	Definition
Reliable broadcast	crash failures, $n > f$ (MP)	Validity, Agreement, Integrity conditions
k-set consensus	crash failures. $f < k < n$. (MP and SM)	size of the set of values agreed upon must be less than k
ϵ-agreement	crash failures $n \geq 5f + 1$ (MP)	values agreed upon are within s of each other
Renaming	up to f fail-stop processes, $n \geq 2f + 1$ (MP) Crash failures $f \leq n - 1$ (SM)	select a unique name from a set of names

Table 10.2: Some solvable variants of the agreement problem in asynchronous system. The overhead bounds are for the given algorithms, and not necessarily tight bounds for the problem.

Here MP- Message Passing, SM- Shared Memory

Agreement in Synchronous Message-Passing Systems with Failures

Byzantine Failure

- "Not fail stop"

- ***Traitor*** nodes send conflicting messages
  - Which leads to an incorrect result

- **Cause:**
  - Flaky node(s)
  - Malicious node(s)

Why study Byzantine failure?

- **Extreme fault tolerance:**
  - Bitcoin
  - Boeing 777 & 787 flight controls

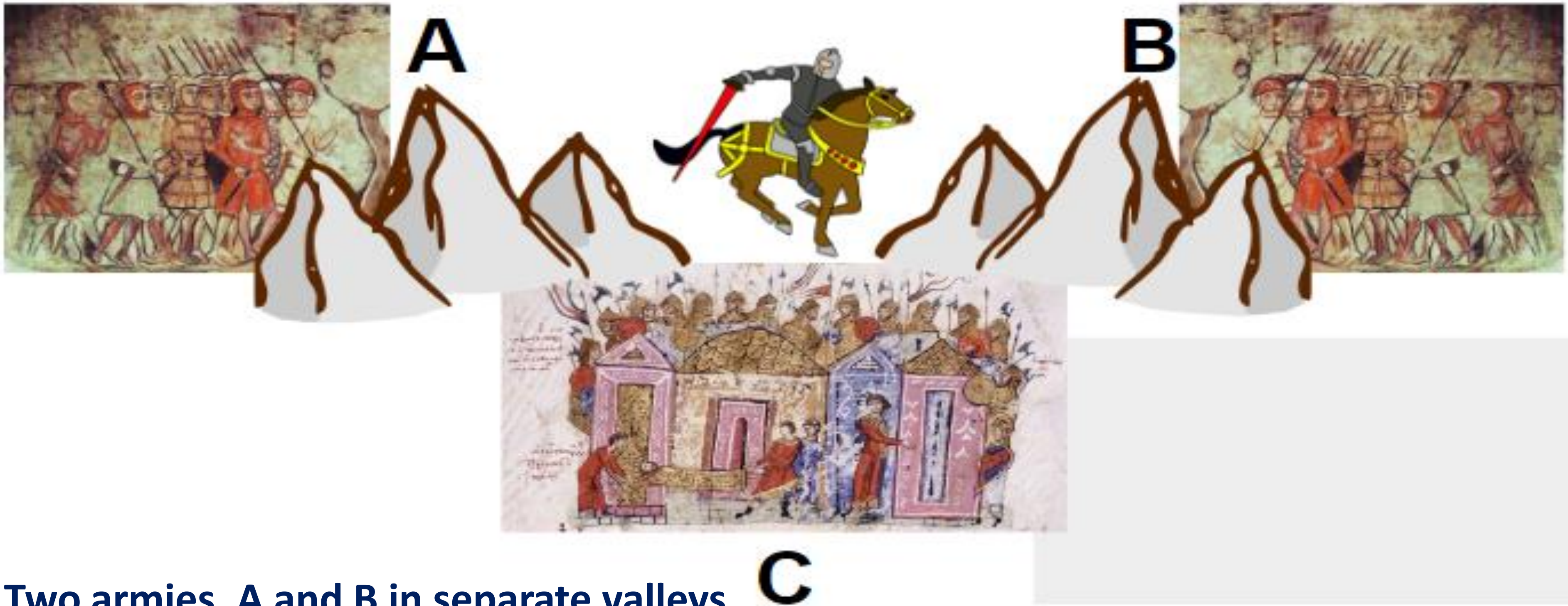
- **Solving this problem is fun!**
  - This reason has really driven a lot of research, since at least the **1980's**

What assumptions are you making?

- Can all nodes see all message? Some? None?
- Do nodes fail? How about the network?
- Finite computation?
- Static or dynamic adversary?
- Bounded communication time?
- Fully connected network?
- Randomized algorithms?
- Quantum or binary computers?

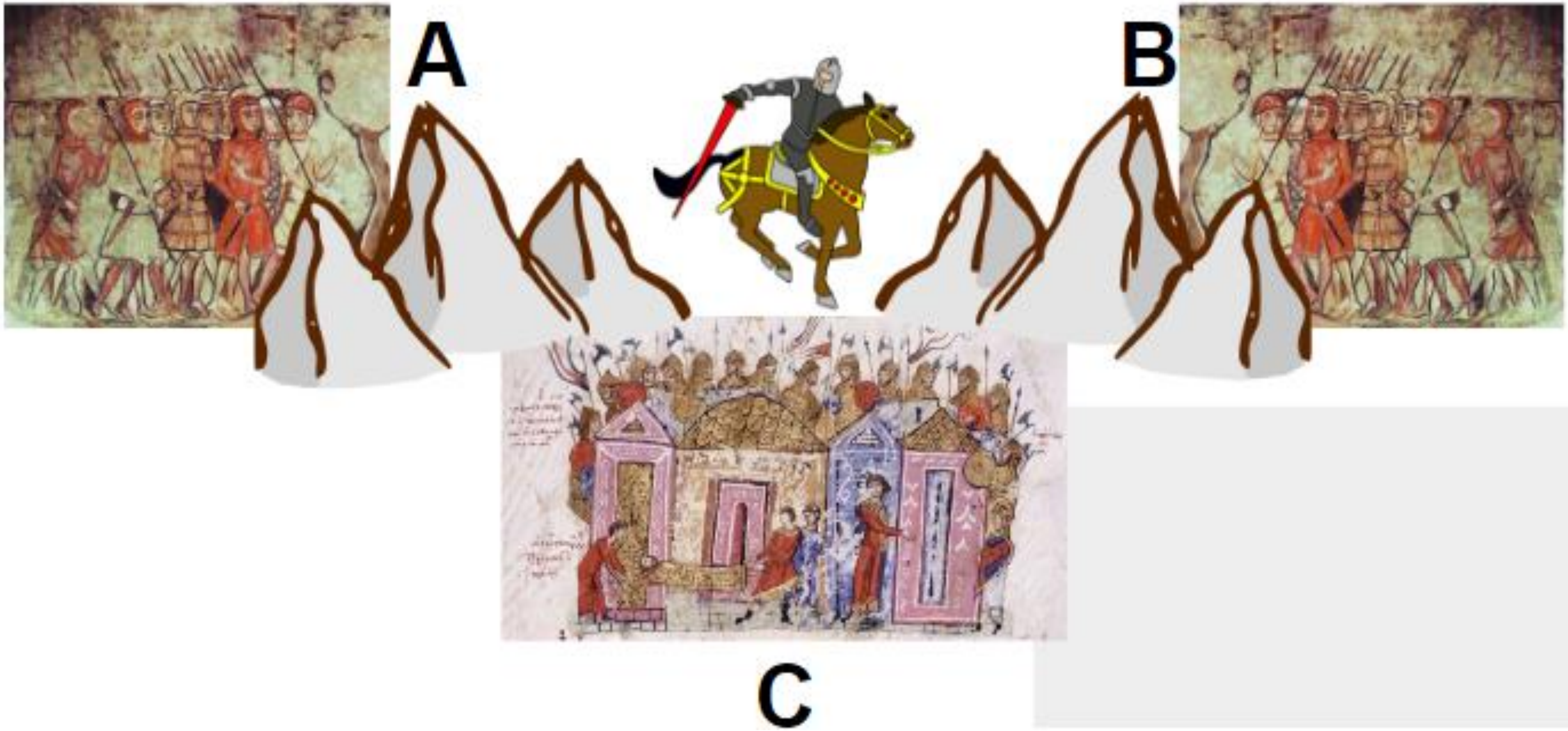
The Two Generals Problem

Consensus: The Two Generals Problem



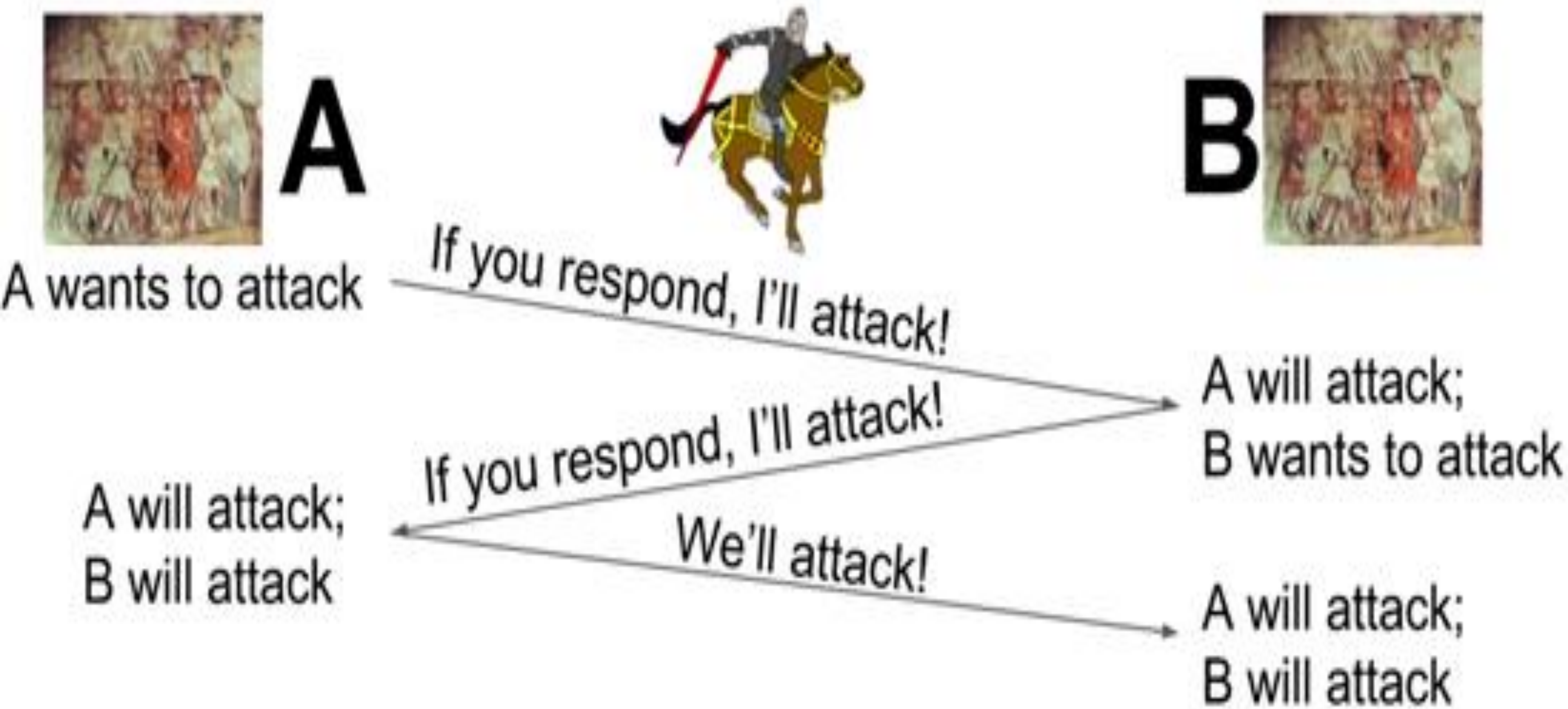
Two armies, A and B in separate valleys.
Want to attack third army, C, in valley between them.
Must decide: attack tomorrow or not?
If they both attack: victory!
If neither attack: survival!
If just one attacks: defeat!
All messages sent by horse -- *through enemy territory*.
Each messenger may or may not make it through.

Consensus: The Two Generals Problem



See if you can figure out a series of messages to solve this problem.

Two Generals Problem: solved ?



There is no perfect solution!

The Byzantine Generals Problem

Byzantine Generals

- **The Byzantine Generals Problem**, Leslie Lamport, Robert Shostack and Mashall Peace. ACM TOPLAS 4.3, 1982.

The Byzantine Generals Problem

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE
SRI International

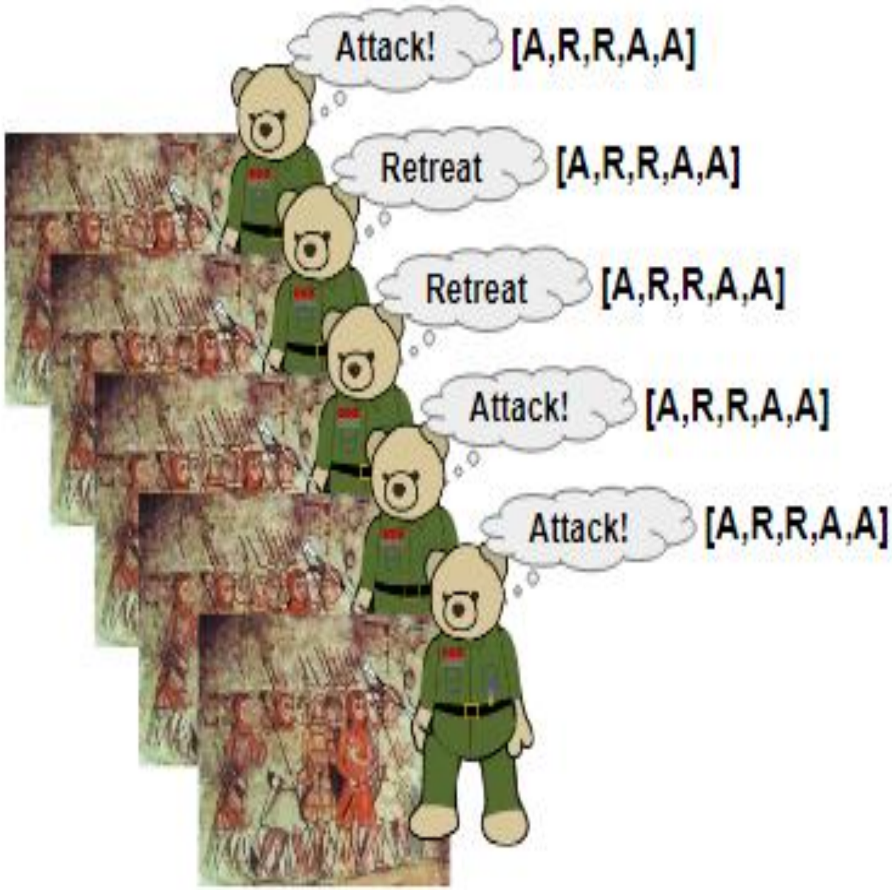
Answers:

- How many byzantine node failures can a system survive?
- How might you build such a system?

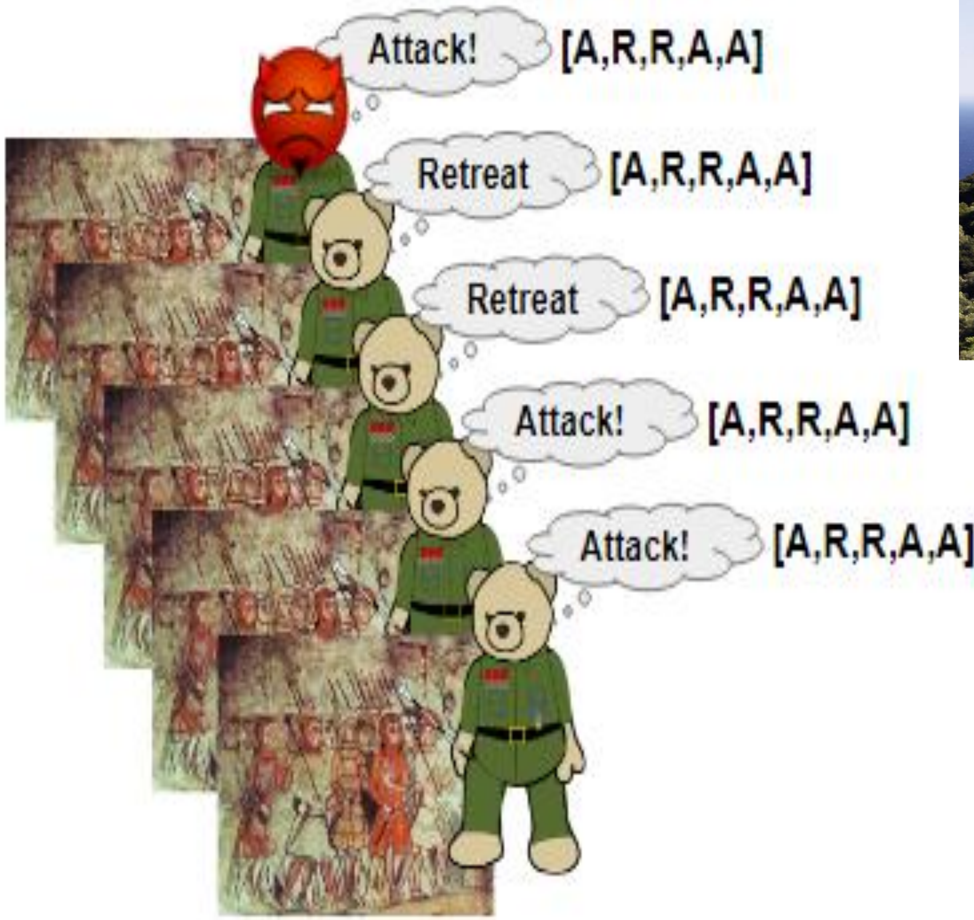
Doesn’t answer:

- Is it worth doing at all?

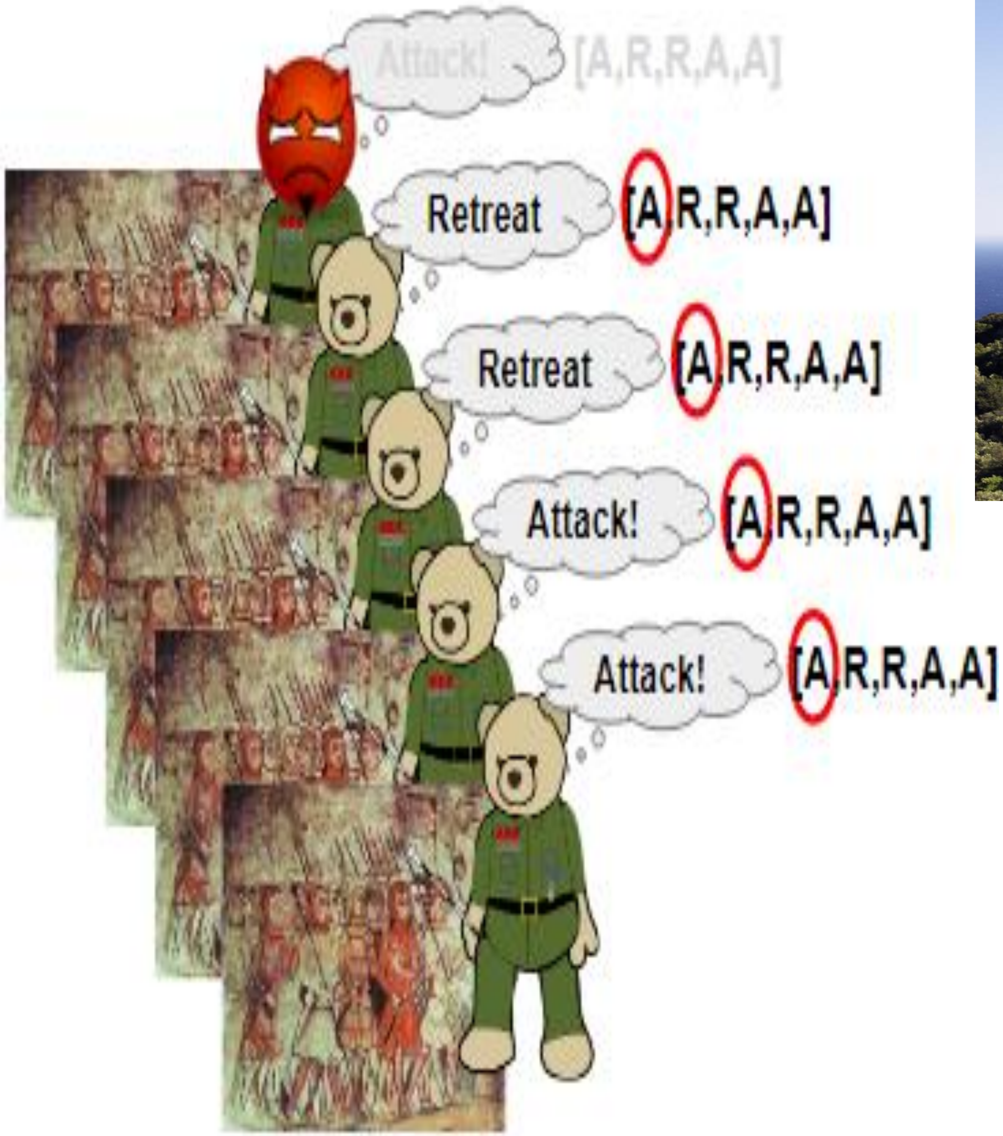
The Problem


Attack!
[A,R,R,A,A]
Retreat
[A,R,R,A,A]
Retreat
[A,R,R,A,A]
Attack!
[A,R,R,A,A]
Attack!
[A,R,R,A,A]

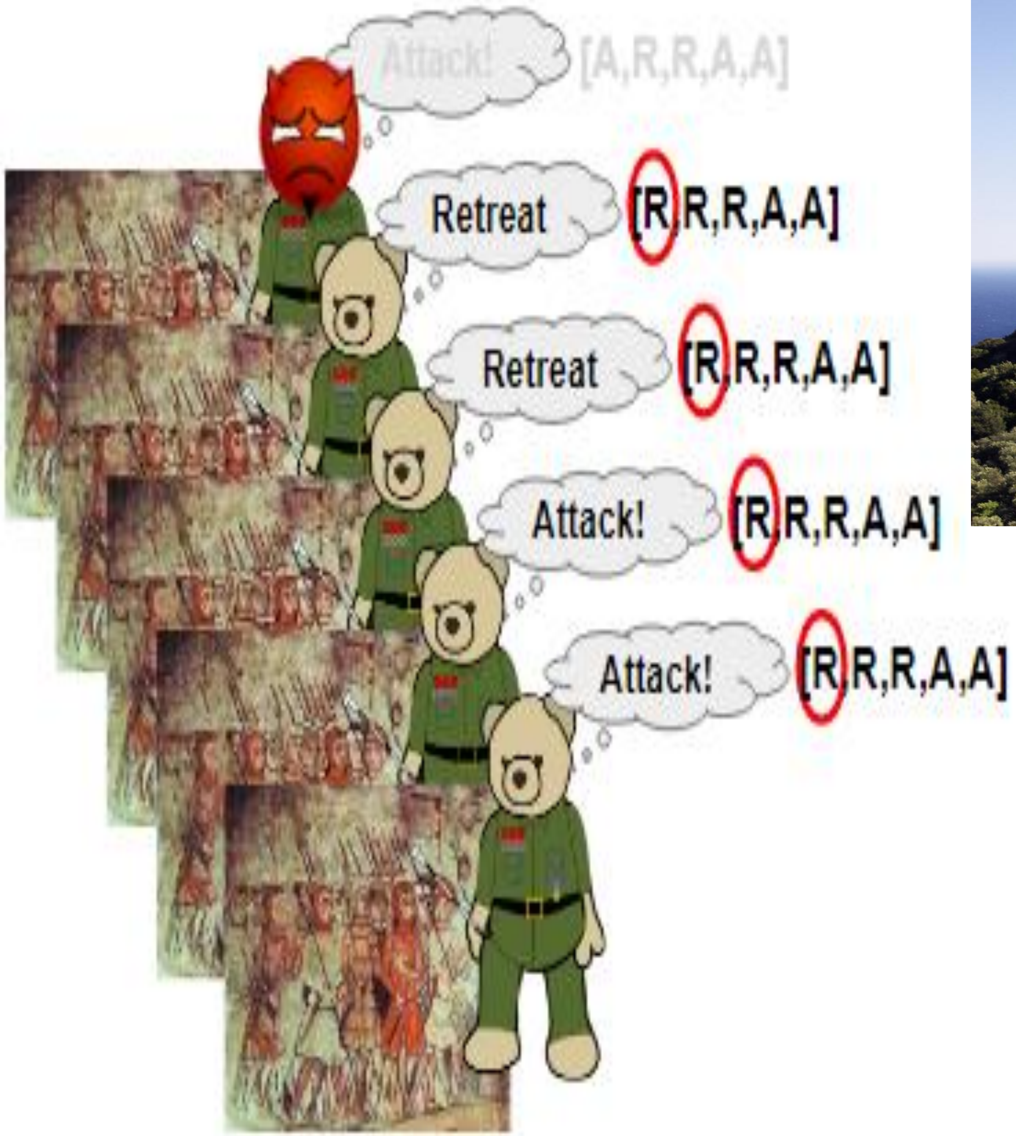
The Problem


Attack!
[A,R,R,A,A]
Retreat
[A,R,R,A,A]
Retreat
[A,R,R,A,A]
Attack!
[A,R,R,A,A]
Attack!
[A,R,R,A,A]

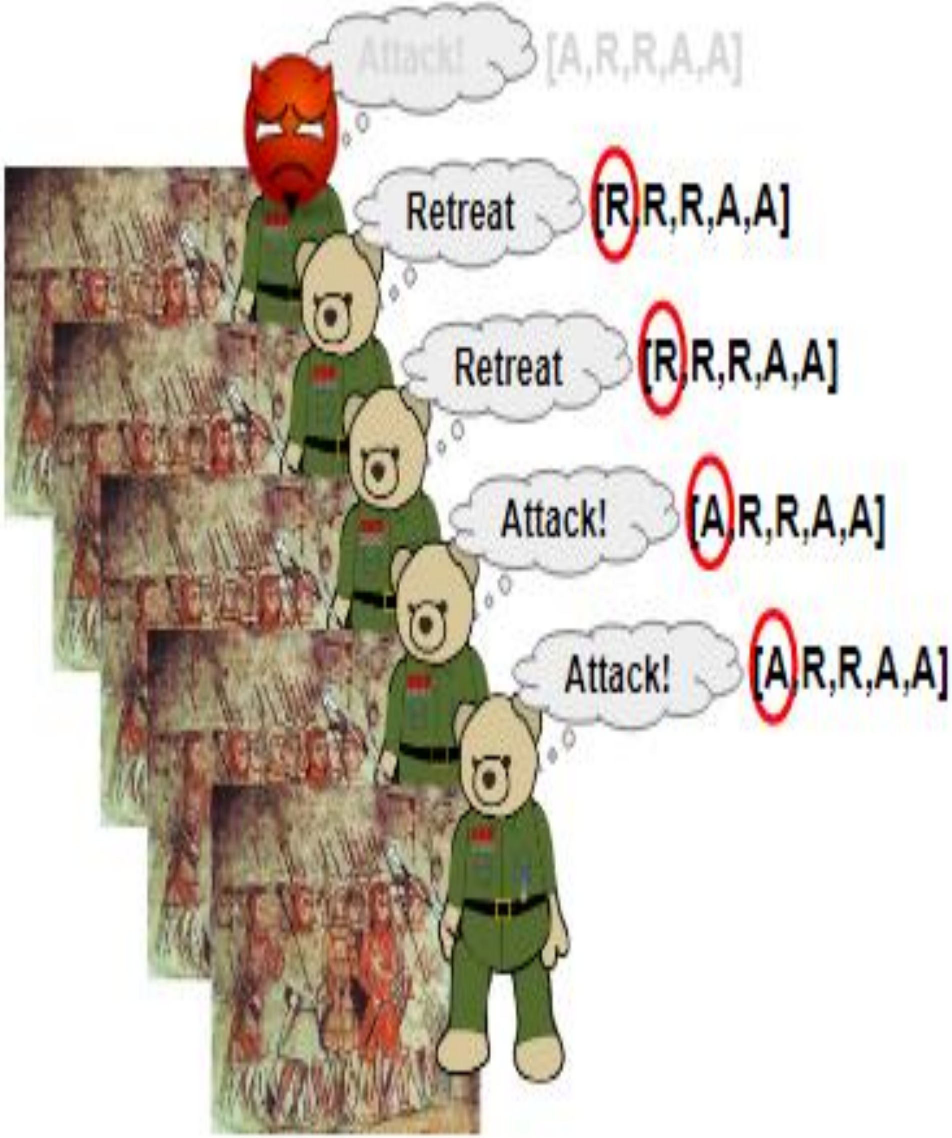
The Problem


Attack!
[A,R,R,A,A]
Retreat
[A,R,R,A,A]
Retreat
[A,R,R,A,A]
Attack!
[A,R,R,A,A]
Attack!
[A,R,R,A,A]

The Problem

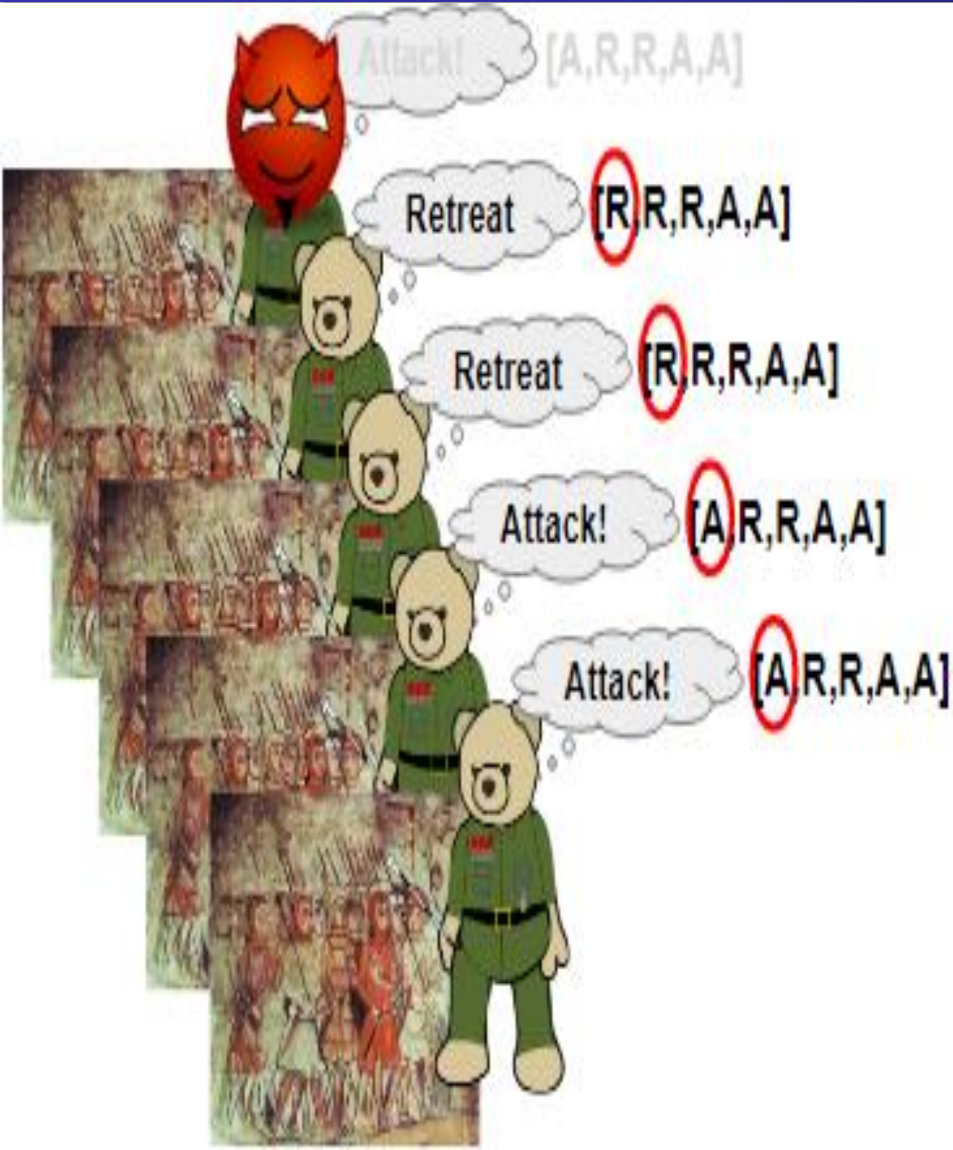

Attack!
[A,R,R,A,A]
Retreat
[R,R,R,A,A]
Retreat
[R,R,R,A,A]
Attack!
[R,R,R,A,A]
Attack!
[R,R,R,A,A]

The Problem



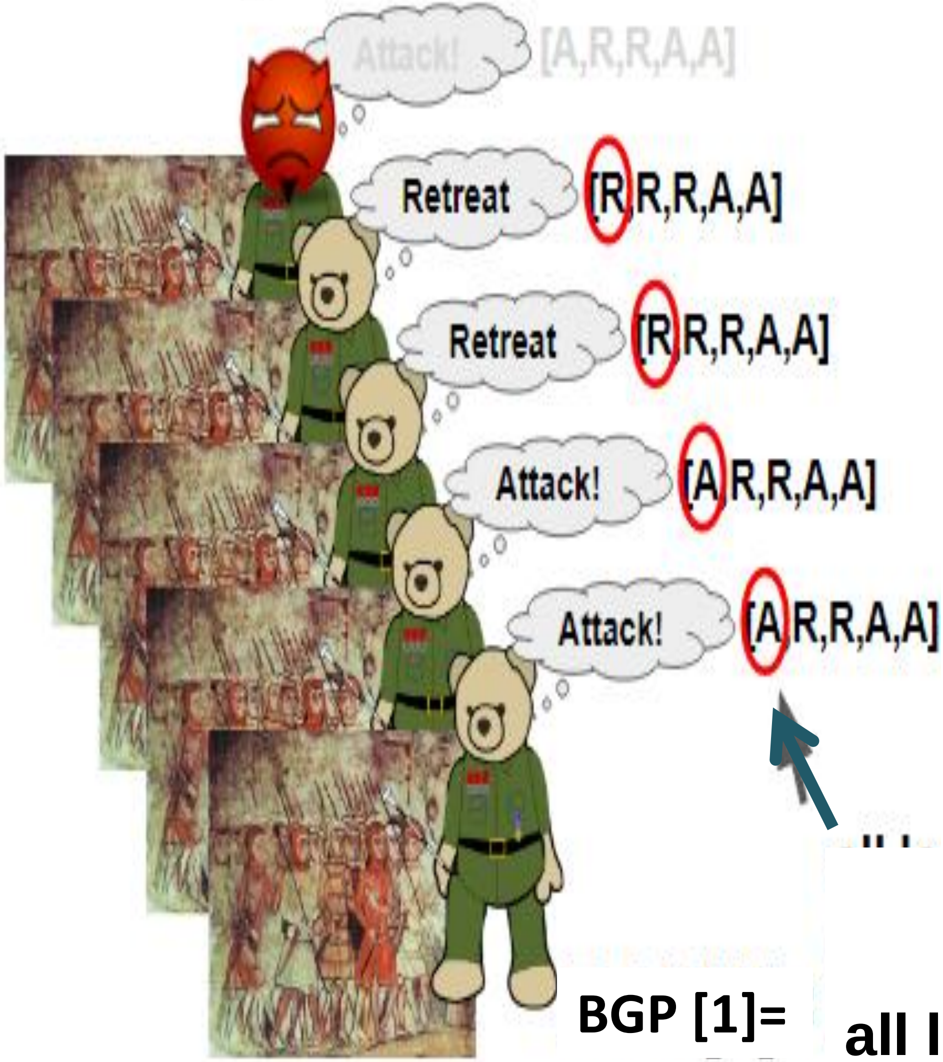
Attack!
[A,R,R,A,A]
Retreat
[R,R,R,A,A]
Retreat
[R,R,R,A,A]
Attack!
[A,R,R,A,A]
Attack!
[A,R,R,A,A]

The Problem



Attack!
[A,R,R,A,A]
Retreat
[R,R,R,A,A]
Retreat
[R,R,R,A,A]
Attack!
[A,R,R,A,A]
Attack!
[A,R,R,A,A]

The Problem



BGP [1]= all loyal generals agree on what 1st general wants

How many traitors can there be?

The question

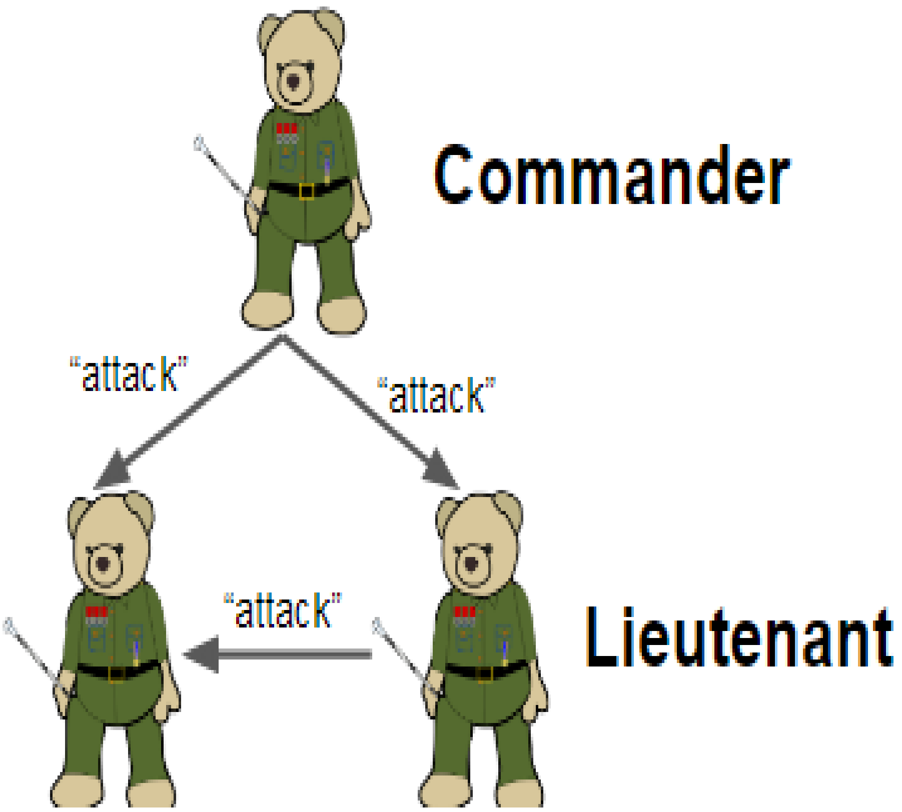
How many traitors can you have and still solve BGP ?

Assuming:
point-to-point ("oral") messages
No crypto

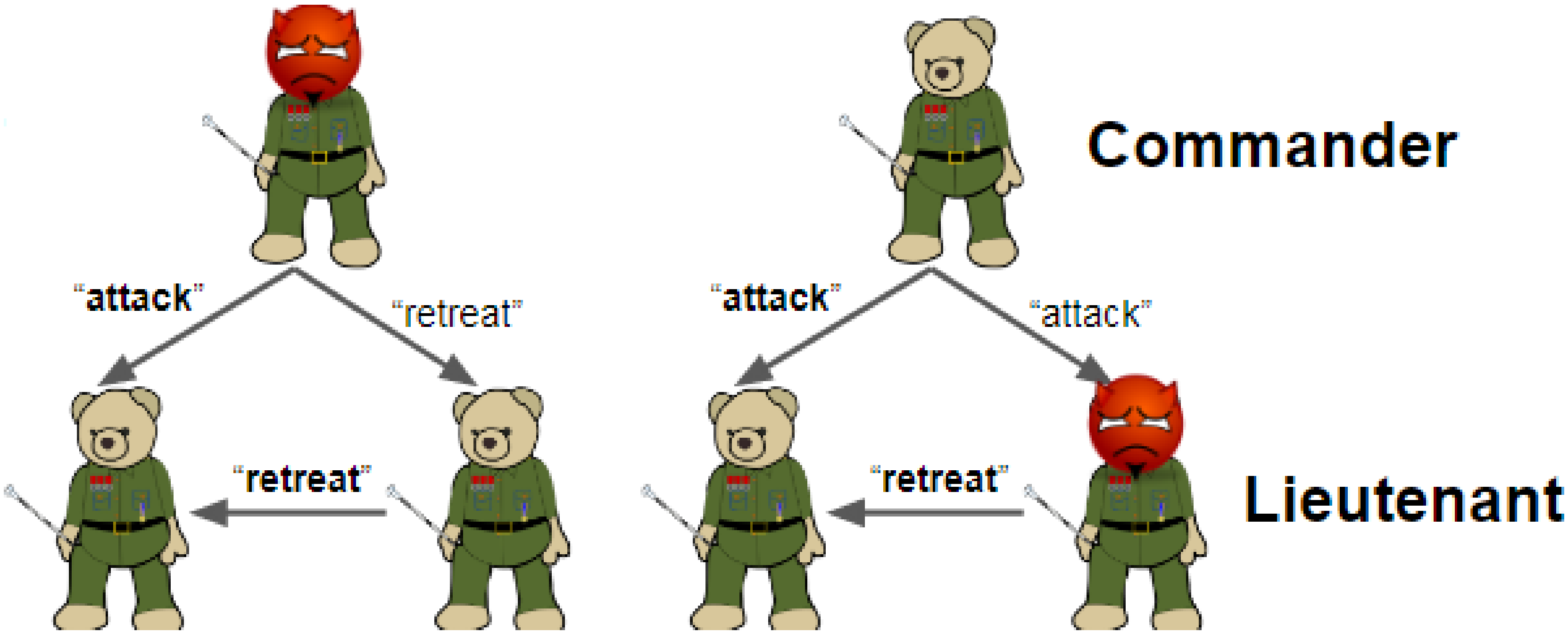
n = 1, or n = 2?

Trivial, no consensus required.

n = 3



n = 3



There is no solution for
3 Generals, 1 Traitor.

How many traitors can be tolerated?

- Lemma: **No** solution for **3m+1** generals with **>m** traitors.

Proof:

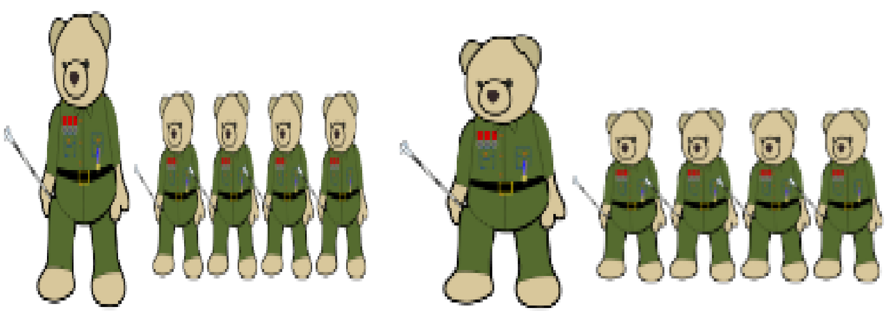
1. Assume solution exists.
2. Use solution to solve **1** traitor **3** generals case.

We know 2 is impossible!

- ⇒ Hence solution must not exist. ⇐

Simulation proof

- **Assume solved:** 4 traitors; 12 generals
- Each general simulates (pretends to be) 4 simulated generals
- Run solution on simulated generals
- Each general chooses value chosen by its simulated generals



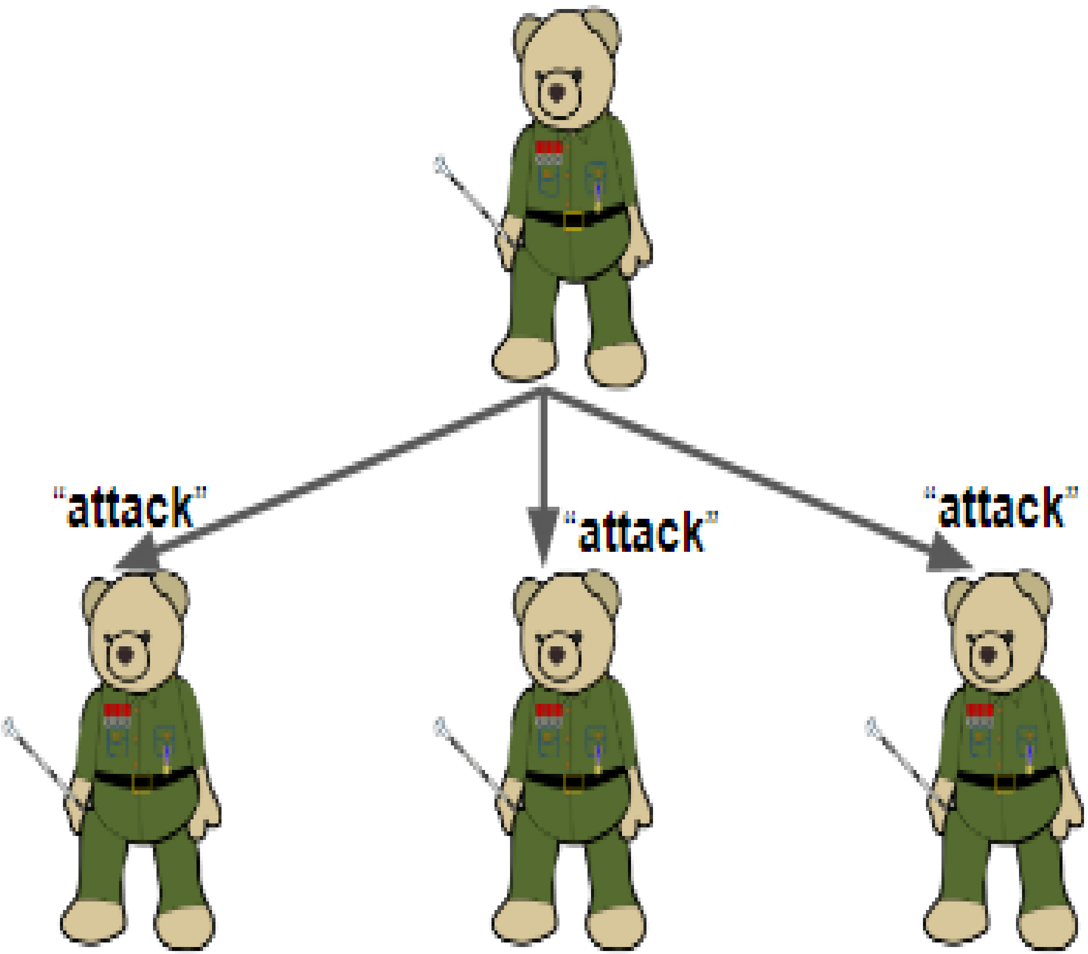
There is no solution for 3m+1 Generals, >m Traitors.

Solving Byzantine Generals

What can you do?

- **Assuming:**
  - Less than ⅓ of generals are traitors
  - Oral messages
  - No crypto
- OM(m): solution to BGP for ≤m traitors

Inductive Solution for Oral Messages

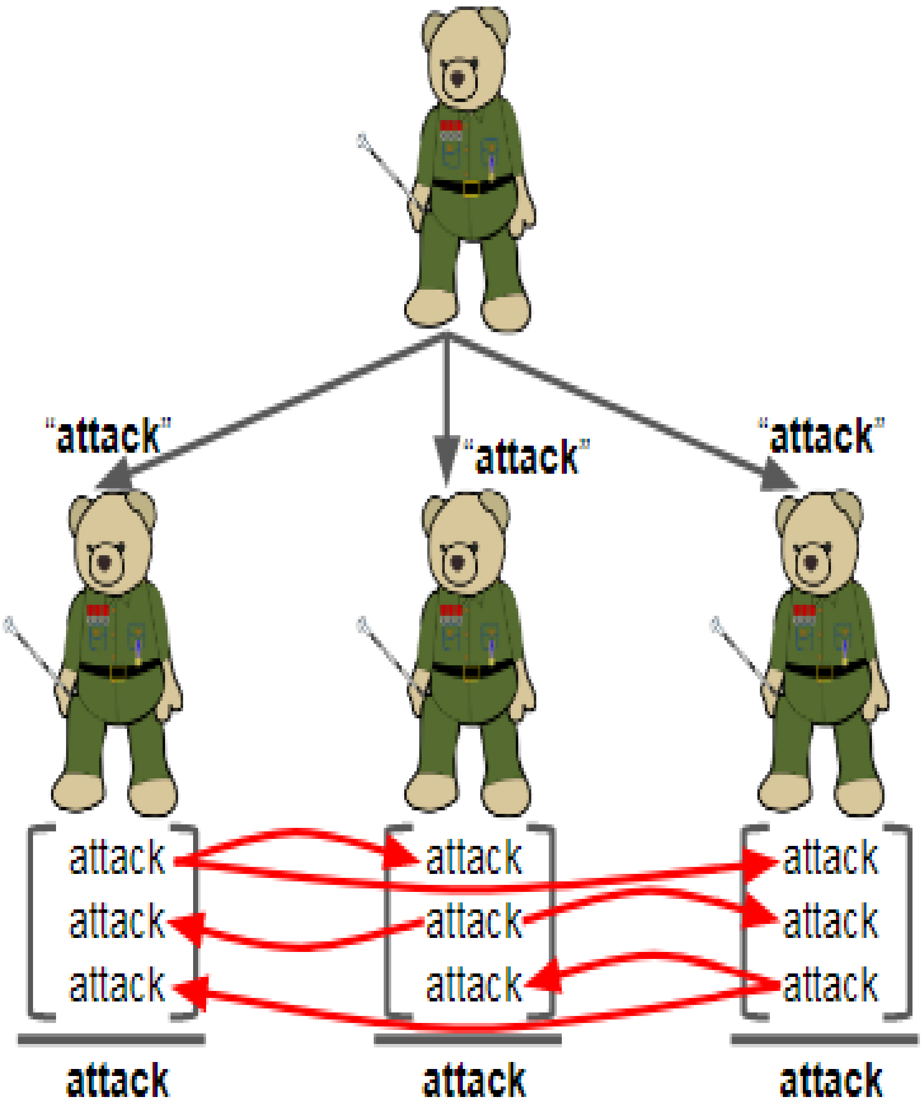


OM(0):

C: Sends order.

L: Follows if received.

Inductive Solution for Oral Messages



OM(m), m>0:

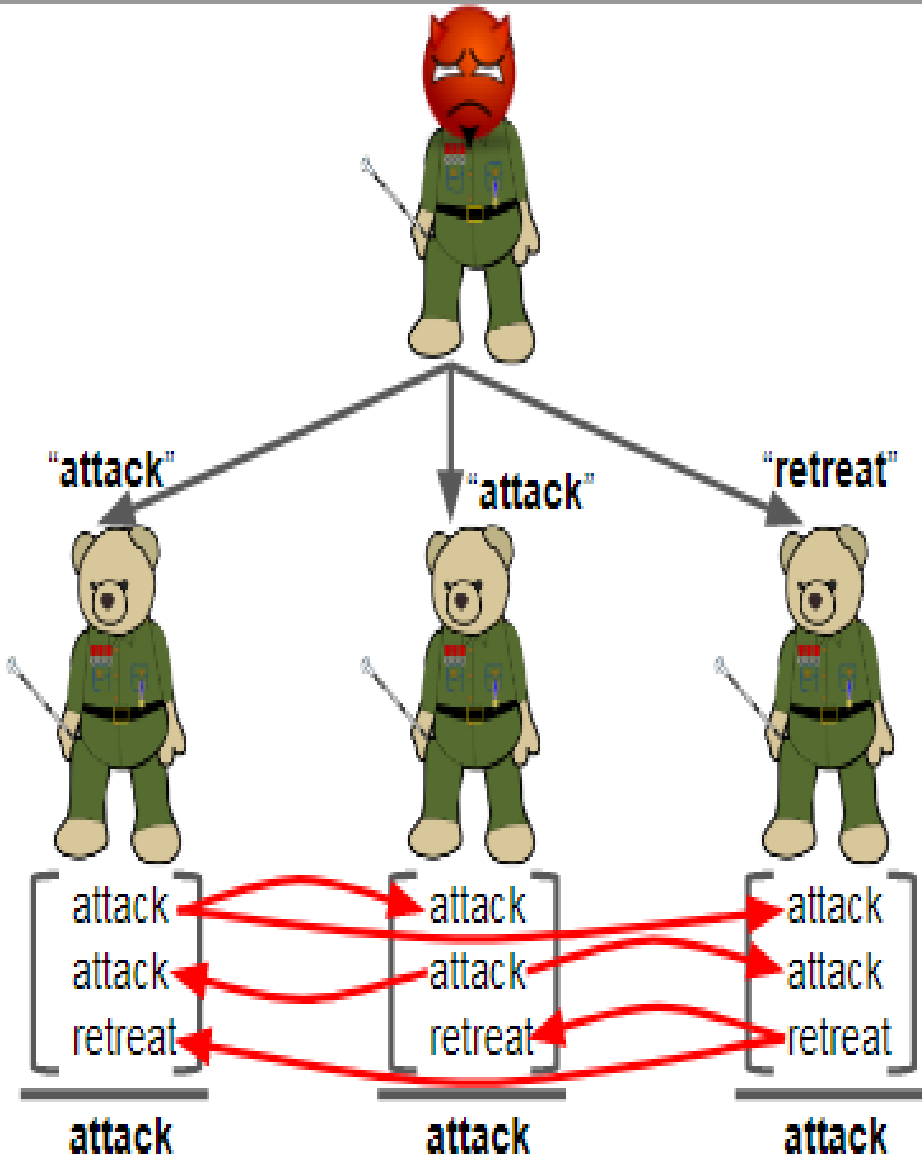
C: Sends order.

L: 1. Records if received.

2. Use OM(m-1) to tell others.

3. Follows majority() order.

How this works with m=1



OM(m), m>0:

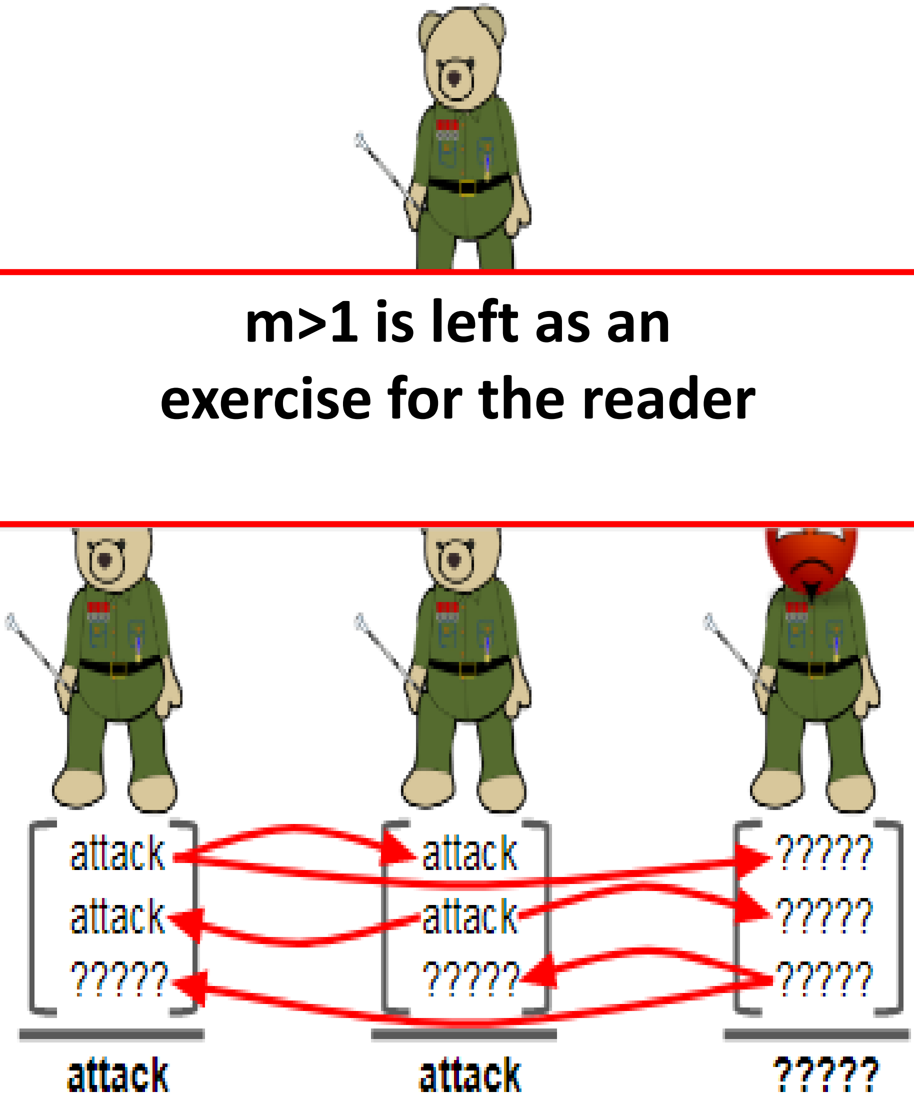
C: Sends order.

L: 1. Records if received.

2. Use OM(m-1) to tell others.

3. Follows majority() order.

How this works with m=1



m>1 is left as an exercise for the reader

OM(m), m>0:

C: Sends order.

L: 1. Records if received.

2. Use OM(m-1) to tell others.

3. Follows majority() order.

Running Time

Expensive

m	Message Sent
0	O(n)
1	O(n^2)
2	O(n^3)
3	O(n^4)

So:

- Don't solve BGP;
- Use someone else's solution; or
- Keep n & m small

2. Consensus Algorithm for Byzantine Failure (Message Passing, Synchronous Systems)

Model :

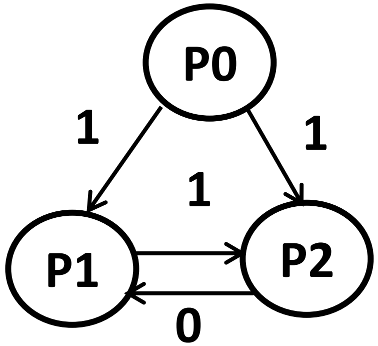
– Total of n processes, at most f of which can be faulty
– Reliable communication medium
– Fully connected
– Receiver always knows the identity of the sender of a message
– Byzantine faults
– **Synchronous system:** In each round, a process receives messages, performs computation, and sends messages.

Solution for Byzantine Agreement Problem

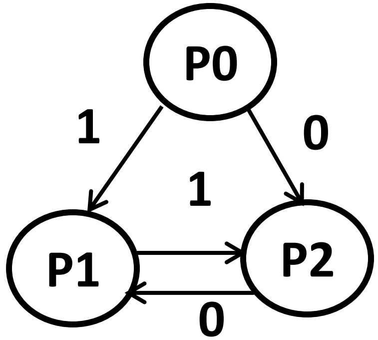
- The solution of Byzantine Agreement Problem is first defined and solved by ***lamport*.**
- ***Pease* showed** that in a fully connected network, **it is impossible to reach an agreement if number of faulty processes *'f'* exceeds *(n-1)/3* where *n* is number of processes**

Byzantine agreement can not be reached among three processes if one process is faulty

P_0 is Non-Faulty



P_0 is Faulty



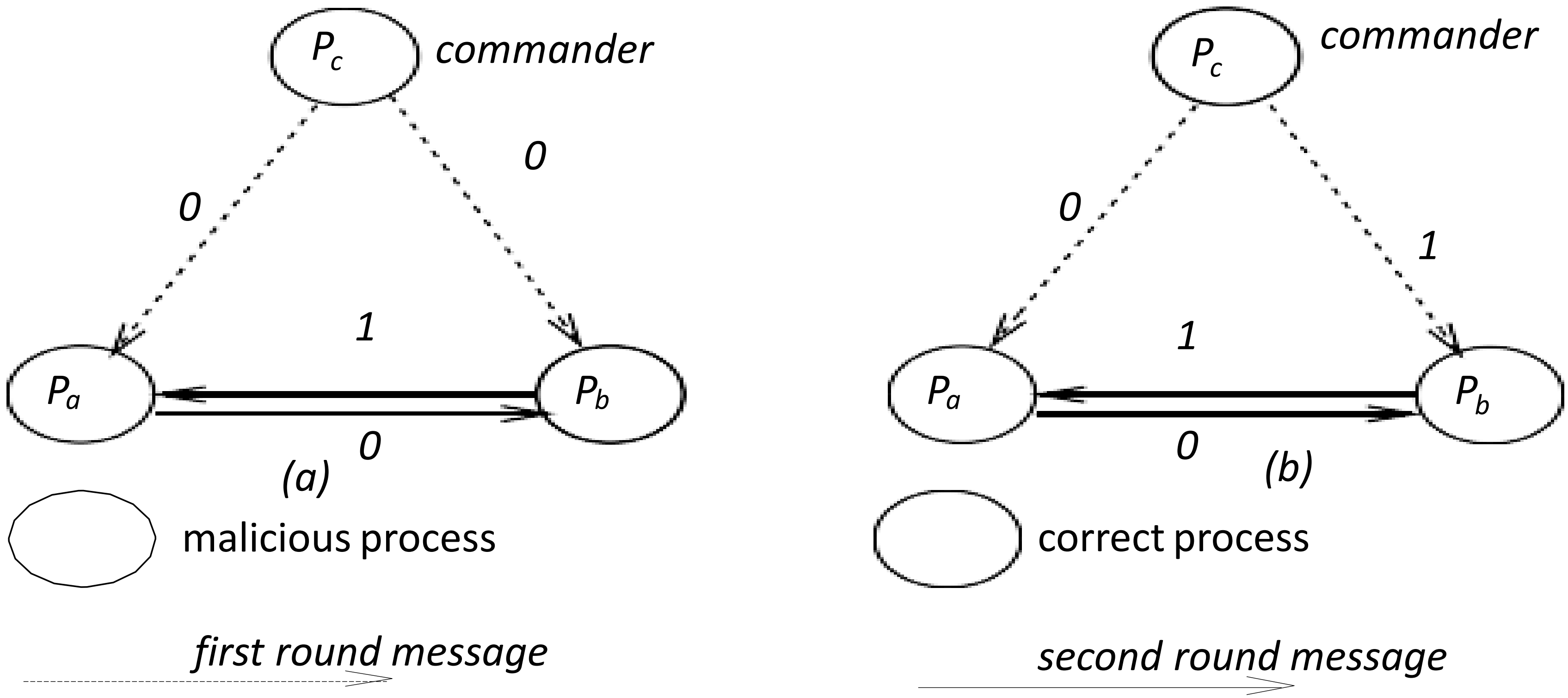
Note: Here P_0 is a source process

Upper bound on Byzantine processes

In a system of n processes, the **Byzantine agreement problem** (as also the other variants of the agreement problem) can be solved in a synchronous only if the number of Byzantine processes f is such that $f \leq \lfloor ((n-1)/3 \rfloor$

Upper Bound on Byzantine Processes

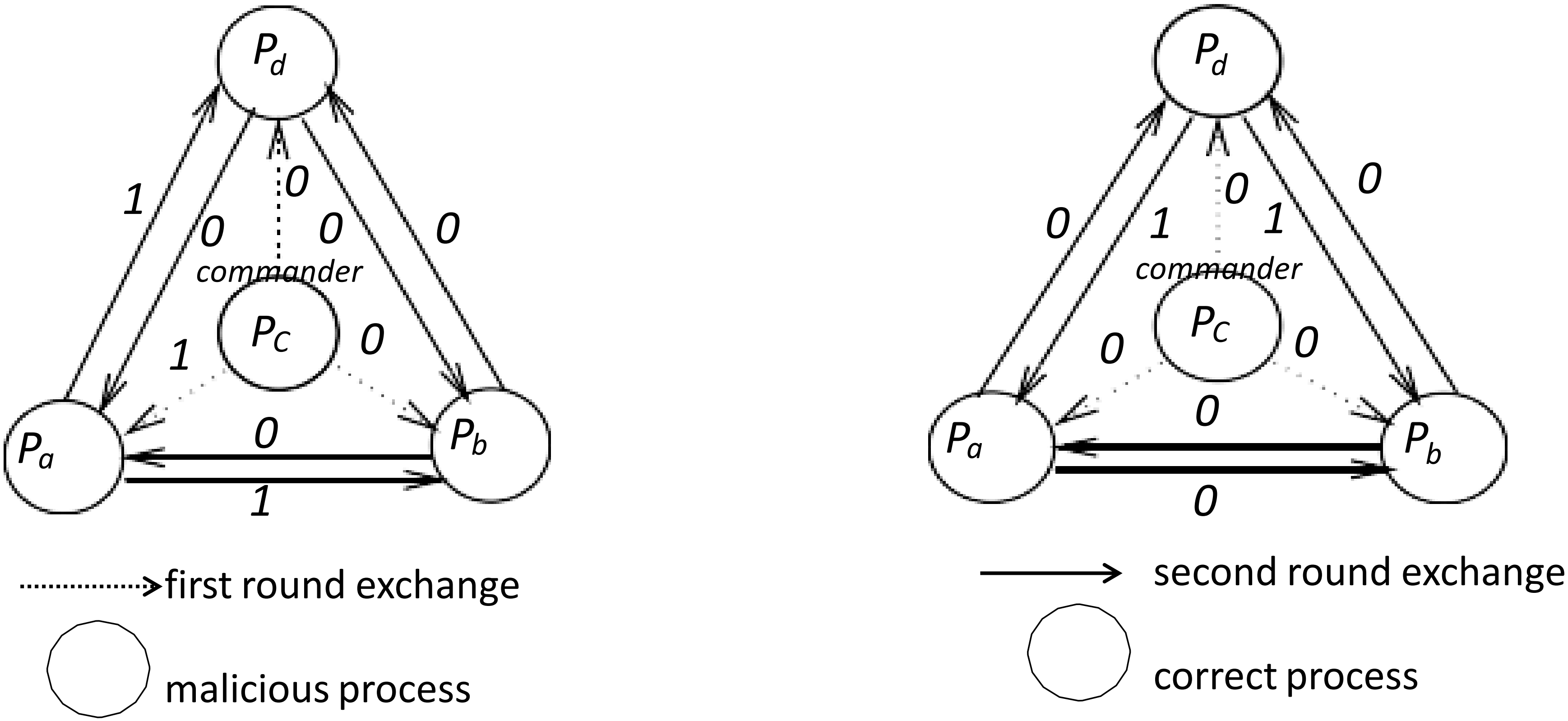
Agreement impossible when *f* = 1, *n* = 3.



Taking simple majority decision does not help because loyal commander P_a cannot distinguish between the possible scenarios (a) and (b); hence does not know which action to take.

Proof using induction that problem solvable if $f \leq \lfloor (n-1)/3 \rfloor$

Consensus Solvable when $f = 1, n = 4$



- There is no ambiguity at any loyal commander, when taking majority decision
- Majority decision is over 2nd round messages, and 1st round message received directly from commander-in-chief process.

Lamport-Shostak-Pease Algorithm

This algorithm also known as ***Oral Message Algorithm OM(f)***
where *f* is the number of faulty processes
'*n*' = Number of processes and ***n >= 3f + 1***

Algorithm is Recursively defined as follows:

Algorithm OM(0)

1. Source process sends its values to each other process
2. Each process uses the value it receives from the source.
   [If no value is received default value 0 is used]

Contd...

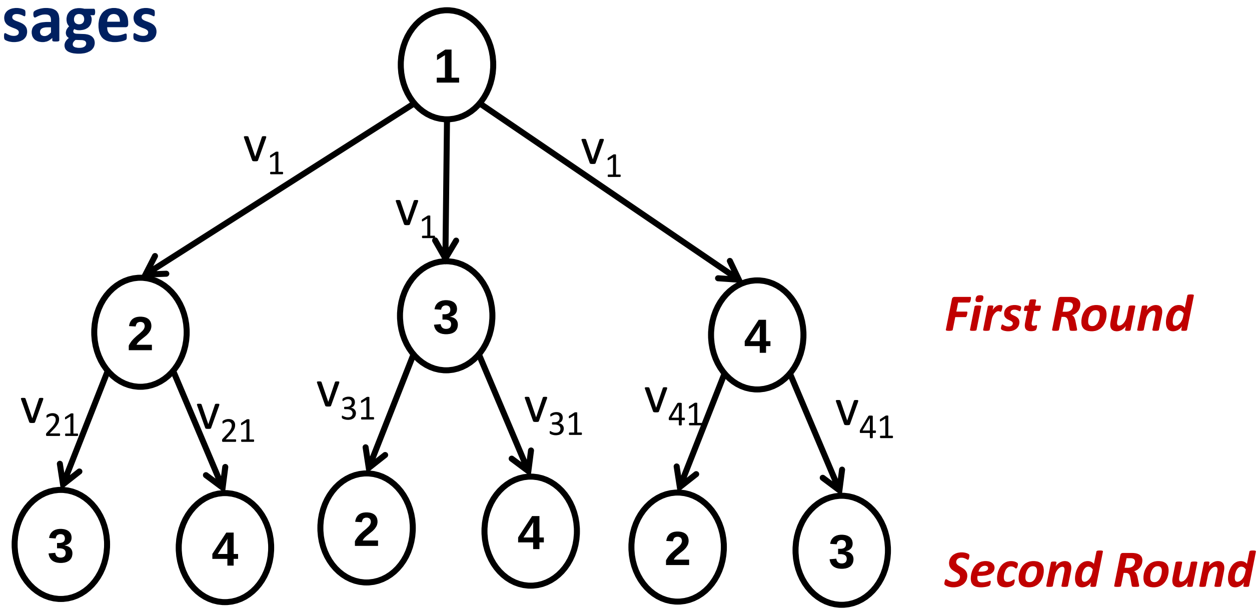
Algorithm OM(*f*), *f* > 0

1. The source process sends its value to each other process.
2. For each *i*, let v_i be the value process *i* receives from source. [Default value 0 if no value received]
3. Process *i* acts as the new source and initiates **Algorithm *OM(f - 1)*** where it sends the value v_i to each of the *n-2* other processes.
4. For each *i* and *j* (not *i*), let v_j be the value process *i* received from process *j* in STEP 3. Process *i* uses the value **majority** $(v_1, v_2 \ldots. v_{n-1})$.

“The function majority($v_1, v_2 \ldots. v_{n-1}$) computes the majority value if exists otherwise it uses default value 0.”

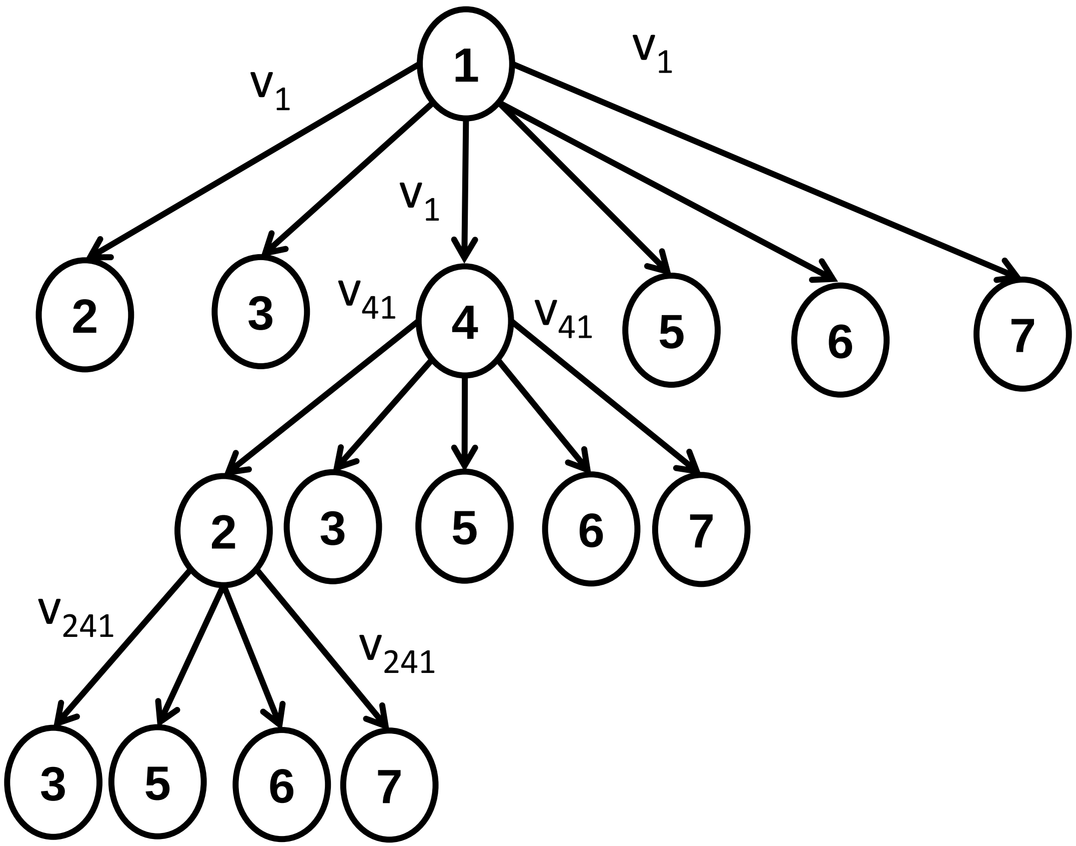
(i) Solving the Byzantine agreement, for *f = 1* and *n = 4*

- **Number of messages for agreement on one value is: 3+2.3 = 9 messages**



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*	*1*	*4 − 1= 3*	*4 − 1 = 3*
2	*2*	*1− 1 = 0*	*4 − 2= 2*	*(4 − 1) · (4 − 2) = 3.2*

(ii) Solving the Byzantine agreement, for *f = 2* and *n = 7*



First Round

Second Round

Third Round

Contd...

- **Number of messages for agreement on one value is: 6+6.5+6.5.4 = 156 messages**

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*	*2*	$7 - 1 = 6$	$7 - 1 = 6$
2	*2*	$2 - 1 = 1$	$7 - 2 = 5$	$(7 - 1) \cdot (7 - 2) = 6.5$
3	*3*	$2 - 2 = 0$	$7 - 3 = 4$	$(7 - 1) \cdot (7 - 2) \cdot (7 - 3) = 6.5.4$

(iii) Solving the Byzantine agreement, for *f = 3* and *n = 10*

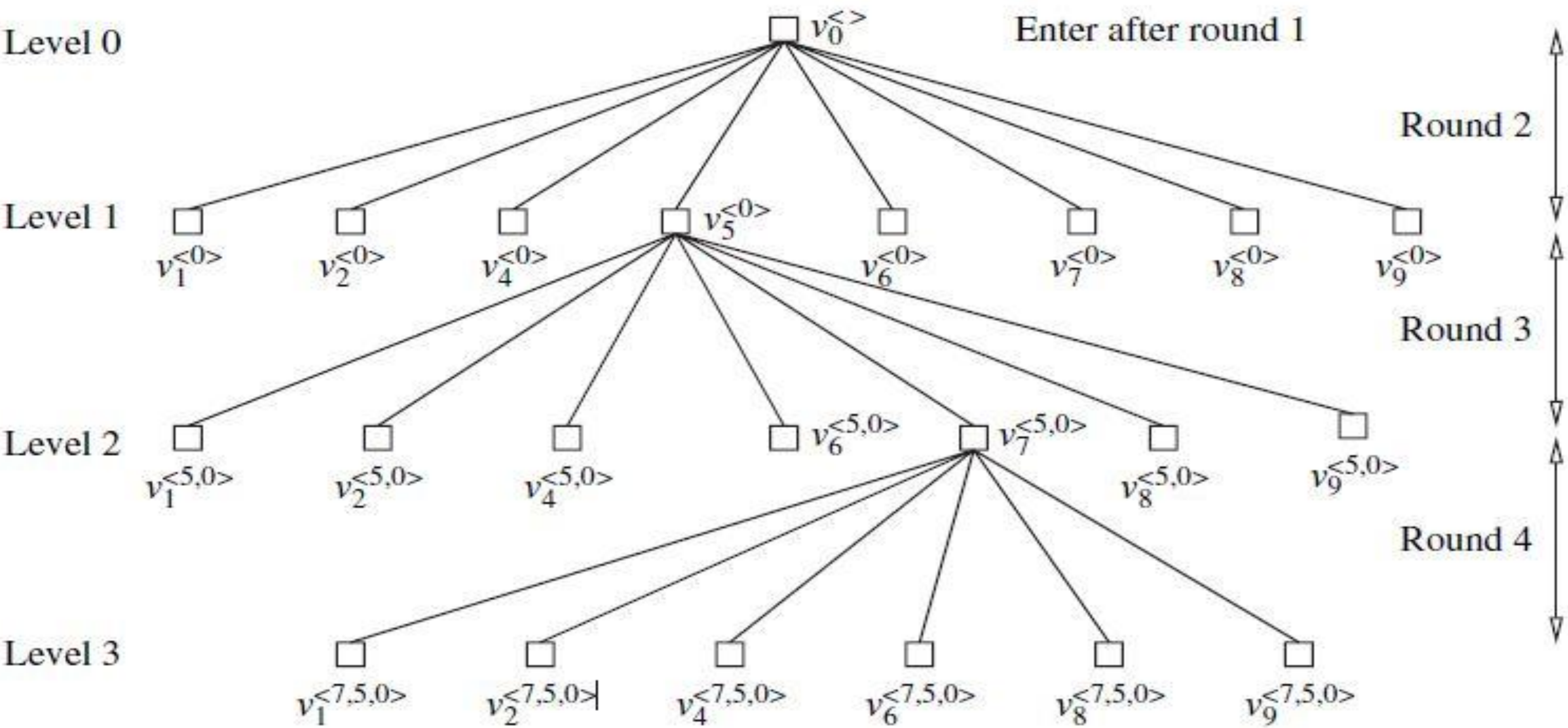


Fig. Local tree at P_3 for solving the Byzantine agreement, for $n = 10$ and $f = 3$, commander is P_0. Only one branch of the tree is shown for simplicity.

Contd...

- **Number of messages for agreement on one value is: 9+9.8+9.8.7+9.8.7.6 = 3609 messages**

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*	*3*	*10 – 1= 9*	*10 – 1 = 9*
2	*2*	*3– 1 = 2*	*10 – 2= 8*	*(10 – 1) · (10 – 2) = 9.8*
3	*3*	*3 - 2 = 1*	*10 – 3 = 7*	*(10 – 1) · (10 – 2) · (10 – 3) = 9.8.7*
4	*4*	*3 – 3 = 0*	*10 – 4 = 6*	*(10 – 1) · (10 – 2) · (10 – 3) · (10 – 4) = 9.8.7.6*

Relationship between # Messages and Rounds

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	$n-f-1$	$(n-1)(n-2)\ldots(n-f-1)$

Table: Relationships between messages and rounds in the Oral Messages algorithm for Byzantine agreement.

Complexity: $f+1$ rounds, exponential amount of space, and $(n-1)+(n-1)(n-2)+\ldots+(n-1)(n-2)..(n-f-1)$ messages

Agreement in Asynchronous Message-Passing Systems with Failures

Impossibility Result for the Consensus Problem

Fischer-Lynch-Paterson (FLP) Impossibility Result
(By M. Fischer, N. Lynch, and M. Paterson, April 1985)

- **Fischer et al.** showed a fundamental result on the impossibility of reaching agreement in an asynchronous (message-passing) system.

- It states that it is "***Impossible to reach consensus in an asynchronous message passing system even if a single process has a crash failure***"

- This result, popularly known as the **FLP impossibility result**, has a significant impact on the field of designing distributed algorithms in a failure-susceptible system.

Weaker Versions of Consensus Problem

Consensus Problem	Description
Terminating reliable broadcast	It states that a correct process always gets a message even if the sender crashes while sending. If the sender crashes while sending the message, the message may be a null message but it must be delivered to each correct process.
k-set consensus	It is solvable as long as the number of crash failures f is less than the parameter k. The parameter k indicates that the non-faulty processes agree on different values, as long as the size of the set of values agreed upon is bounded by k.
Approximate agreement	Like k-set consensus, approximate agreement also assumes the consensus value is from a multi-valued domain. However, rather than restricting the set of consensus values to a set of size k, ϵ-approximate agreement requires that the agreed upon values by the non-faulty processes be within ϵ of each other.
Renaming problem	It requires the processes to agree on necessarily distinct values.
Reliable broadcast	A weaker version of reliable terminating broadcast(RTB), namely reliable broadcast, in which the termination condition is dropped, is solvable under crash failures.

(i) Terminating Reliable Broadcast (TRB) Problem

- A correct process always gets a message, even if sender crashes while sending (in which case the process gets a null message).

- **Validity:** If the sender of a broadcast message *m* is non-faulty, then all correct processes eventually deliver *m*.
- **Agreement:** If a correct process delivers a message *m*, then all correct processes deliver m.
- **Integrity:** Each correct process delivers at most one message. Further, if it delivers a message different from the null message, then the sender must have broadcast m.
- **Termination:** Every correct process eventually delivers some message.

Contd...

- The reduction from consensus to terminating reliable broadcast is as follows:
- A commander process broadcasts its input value using the terminating reliable broadcast. A process decides on a "0" or "1" depending on whether it receives "0" or "1" in the message from this process.
- However, if it receives the null message, it decides on a default value. As the broadcast is done using the terminating reliable broadcast, it can be seen that the conditions of the consensus problem are satisfied.
- **But as consensus is not solvable, an algorithm to implement terminating reliable broadcast cannot exist.**

(ii) Reliable Broadcast Problem

- **Reliable Broadcast (RB)** is RTB without terminating condition.
- RTB requires eventual delivery of messages, even if sender fails before sending. In this case, a null message needs to get sent. In RB, this condition is not there.
- RTB requires recognition of a failure, even if no msg is sent
- **Crux:** RTB is required to distinguish between a failed process and a slow process.
- **RB is solvable under crash failures; $O(n^2)$ messages**

Algorithm: Protocol for reliable broadcast

(1) Process P_0 initiates Reliable Broadcast:

(1a) **broadcast** message M to all processes.

(2) A process P_i, $1 \leq i \leq n$, receives message M:

(2a) **if** M was not received earlier **then**

(2b) **broadcast** M to all processes;

(2c) deliver M to the application.

Applications of Agreement Algorithms

1) Fault-Tolerant Clock Synchronization

- Distributed Systems require physical clocks to synchronized
- Physical clocks have drift problem
- Agreement Protocols may help to reach a common clock value.

2) Atomic Commit in Distributed Database System (DDBS)

- DDBS sites must agree whether to commit or abort the transaction
- Agreement protocols may help to reach a consensus.

Conclusion

- **Consensus problems are fundamental aspects** of distributed computing because they **require inherently distributed processes** to reach agreement.

- This lecture first covers:

  -Different forms of the **consensus problem**,

  -Then gives an overview of what forms of consensus are solvable under different failure models and different assumptions on the synchrony/asynchrony.

Contd...

- Then we have covered agreement in the following categories:

(i) Synchronous message-passing systems with failures:

-Used Fault Models: fail-stop model and the Byzantine model.

(ii) Asynchronous message-passing systems with failures:

-Impossible to reach consensus in this model.

-Hence, several weaker versions of the consensus problem, **i.e. terminating reliable broadcast, reliable broadcast** are considered.