# Byzantine Agreement

Yash Bhagwat

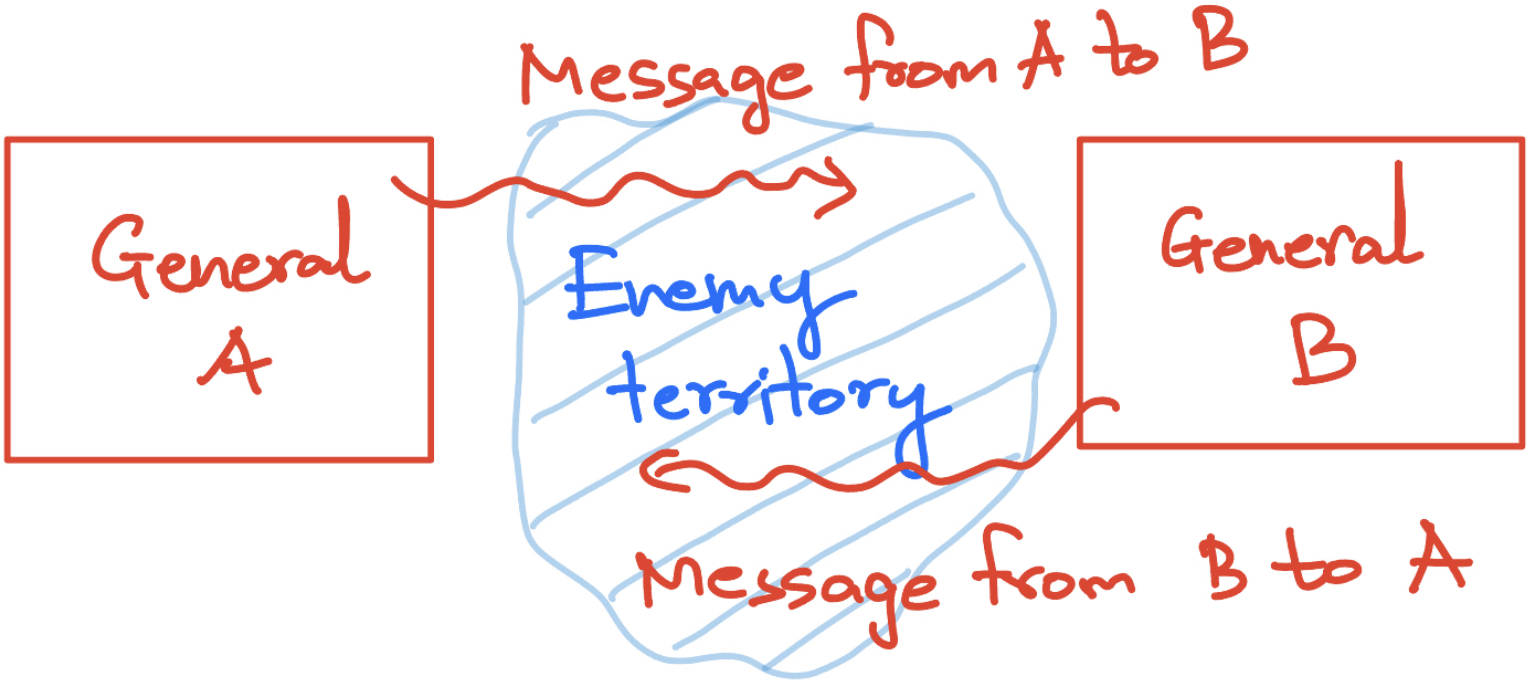
Indian Institute of Technology, Madras

18-07-2023

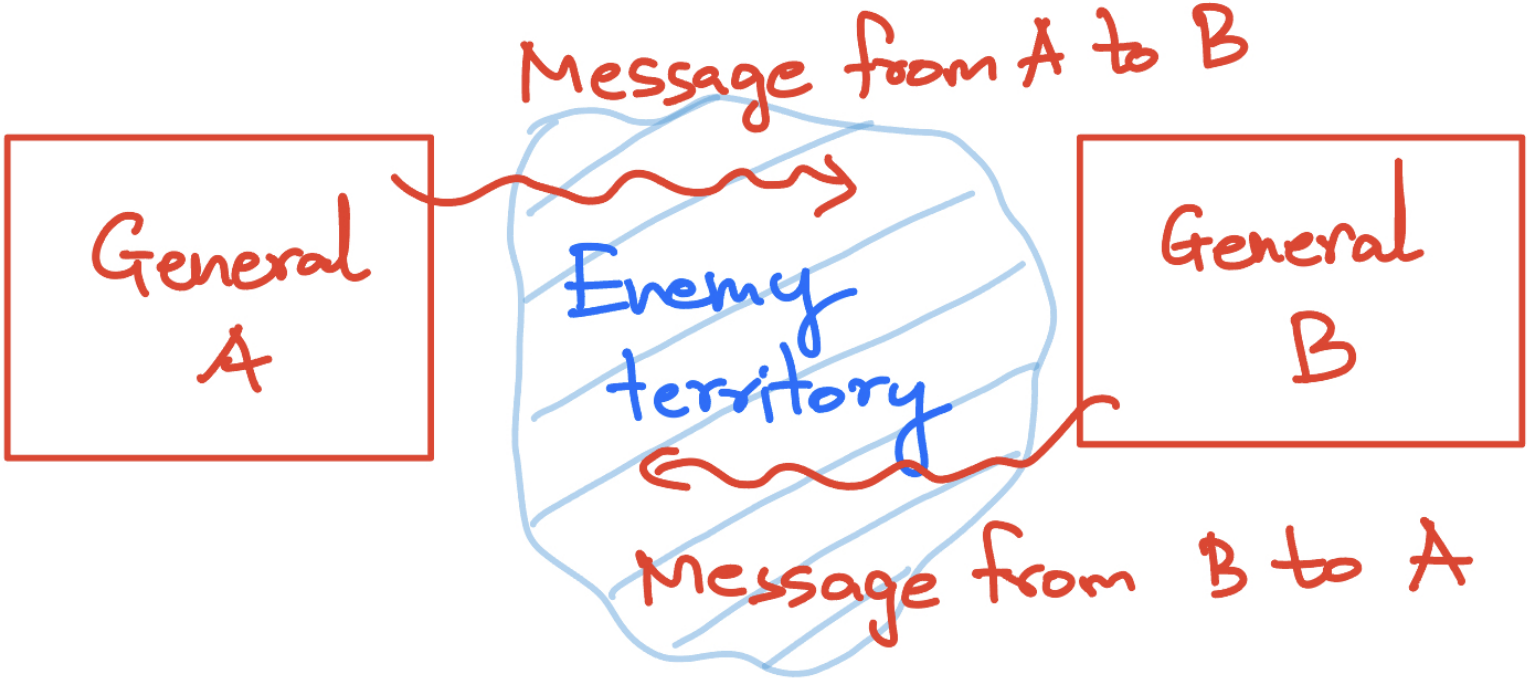
# Overview

- Consensus
  - What is Consensus?
  - Properties of Consensus
- Byzantine Agreement
  - Byzantine
  - Byzantine Agreement
  - Simplest case for Byzantine Agreement
  - King's Algorithm

## Two Generals Problem



**Two Generals Problem**




- Such a protocol cannot terminate **(Why?)**

# Proof of non-termination

- Let's assume that there exists at-least 1 such protocol which leads to agreement.
- Let S be the protocol with the smallest number of messages.
- Since this protocol guarantees agreement even though its last message may be lost, we can just omit and not send the last message.
- But this contradicts the initial assumption that S has smallest number of messages.

# Definition of consensus

There are $n$ nodes, of which at most $f$ might crash, i.e., at least $n - f$ nodes are correct. Node $i$ starts with an input value $v_i$. The nodes must decide for one of those values, satisfying the following properties:

- **Agreement** : All correct nodes decide for the same value
- **Termination** : All correct nodes terminate in finite time
- **Validity** : The decision value must be the input value of some node

# Byzantine Nodes

A node which can have arbitrary behavior is called **byzantine**. This includes "anything imaginable", e.g., not sending any messages at all, or sending different and wrong messages to different neighbors, or lying about the input value.

# Remarks on Byzantine Nodes

- Byzantine nodes can also collude with each other (say they are controlled by the same adversary)
- We assume that any two nodes communicate directly, and that no node can forge an incorrect sender address. **(Why?)**

# Byzantine Agreement

- Finding consensus in a system with byzantine nodes is called **byzantine agreement**. An algorithm is f-resilient if it still works correctly with f byzantine nodes.

- We need agreement, termination and validity to show consensus; agreement and termination are straightforward.

# Validity

What is the issue with the definition of validity discussed before**(?)**
(The decision value must be the input value of some node)

# Any-Input Validity

The decision value must be the input value of some node.

**Remarks**:

- Does this definition still make sense in the presence of byzantine nodes? What if byzantine nodes lie about their inputs?
- Let's look at a validity definition which differentiates between byzantine and correct inputs.

# Correct-Input Validity

The decision value must be the input value of a correct node.

- Can you guess what is the issue with this definition?

# Issue with Correct-Input Validity

The decision value must be the input value of a correct node.

- Consider a Byzantine node which acts exactly as a correct node but just lies about its input value.
- Such a node is indistinguishable from a correct node.

# All-Same Validity

If all correct nodes start with the same input v, the decision value must be v.

- In the case where the decision values are binary, the correct-input validity is induced by the all-same validity.
- But in the case where the decision values are in $\in \mathbb{R}$, all-same validity may turn out to be not that useful.

# Basic Byzantine Agreement

Let us try to consider the most basic Byzantine agreement problem where we tolerate 1 byzantine node.

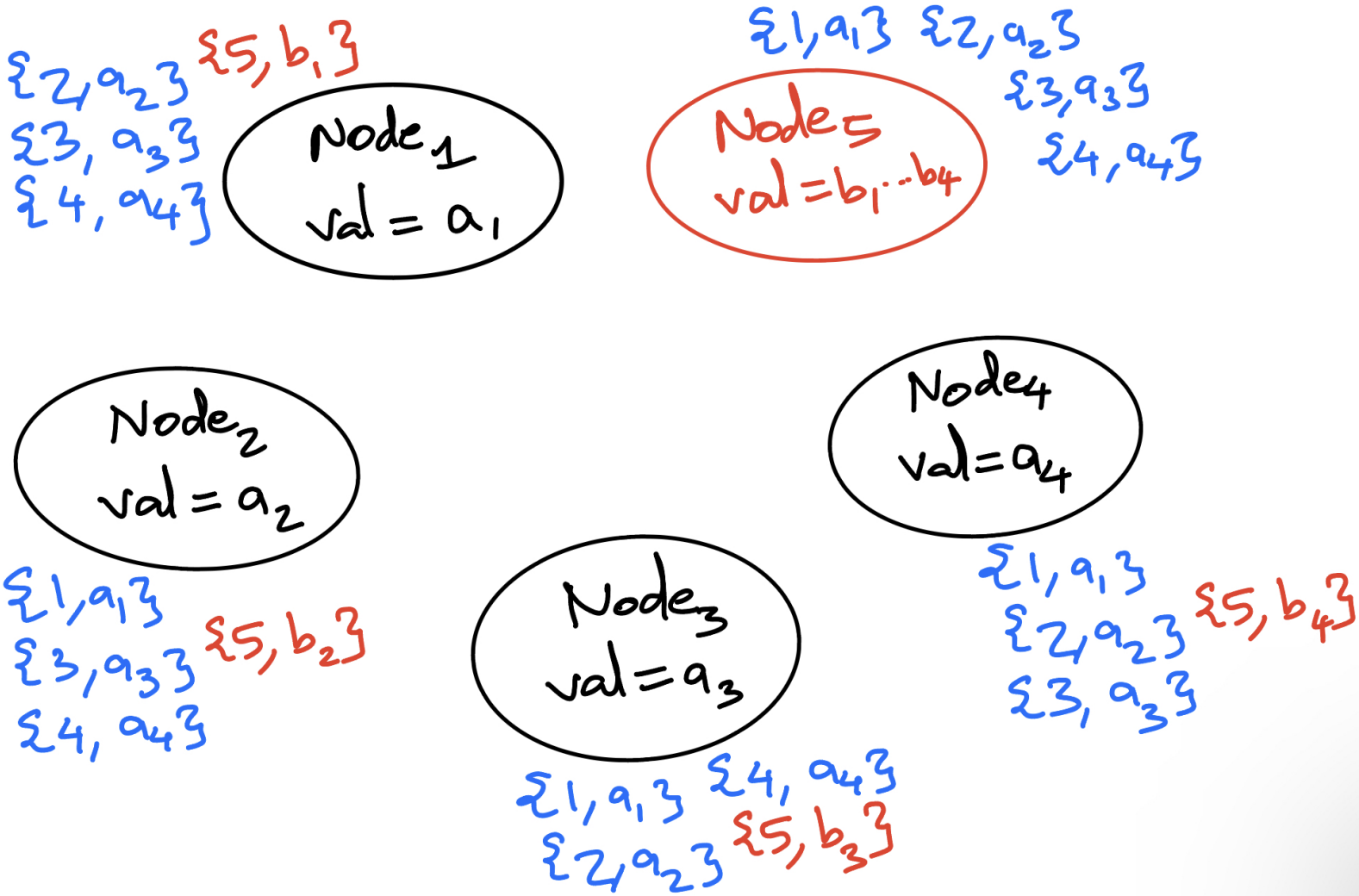
First we restrict ourselves to the synchronous model setting.
**Recap :** In the synchronous model, nodes operate in synchronous rounds. In each round, each node may send a message to the other nodes, receive the messages sent by the other nodes, and do some local computation.

# Protocol for 1 Byzantine node

**Algorithm** Byzantine agreement for f = 1

1: *Code for node u, with input value x* :
2:
3: **Round 1**:
4: Send tuple$(u, x)$ to all other nodes.
5: Receive tuple$(v, y)$ from all other nodes $v$.
6: Store all received tuples in a set $S_u$
7:
8: **Round 2**:
9: Send set $S_u$ to all other nodes
10: Receive sets $S_v$ from all nodes v
11: T = set of tuple$(v, y)$ seen in at least two sets $S_v$, including own $S_u$
12: Let tuple$(v, y) \in T$ be the tuple with the smallest value y
13: Decide on value y

# Example diagram



# Some details about the algorithm

- If nodes send syntactically incorrect messages, we can detect and discard them easily.
- It is worse if byzantine nodes send syntactically correct messages, but with a bogus content, e.g., they send different messages to different nodes.
- Example, if a byzantine node sends different values to different nodes in the first round; such values will be put into $S_u$.

# Some details about the algorithm

- We observe that all nodes only relay other nodes information in Round 2. So, if a byzantine node sends a set $S_v$ which contains a tuple(v, y), this tuple must be removed by u.
- Also we assumed that nodes cannot forge their source address; thus, if a node receives tuple(v, y) in Round 1, it is guaranteed to be sent by v.

# Correctness of algorithm for $n \geq 4$

- With f $= 1$ and $n \geq 4$ we have at least 3 correct nodes. A correct node will see every correct value at least twice, once directly from another correct node, and once from a third correct node. So all correct values are in T.
- If the byzantine node sends the same value to at least 2 other (correct) nodes, all correct nodes will see the value twice, so all add it to set T. If the byzantine node sends all different values to the correct nodes, none of these values will end up in any set T.
- Since all correct nodes have the same set T , they agree on the same minimum value. The nodes agree on a value proposed by some node, so any-input validity holds.
- The algorithm terminates after two rounds.

# Byzantine agreement for 3 nodes

- Let's say we have three nodes u, v, w.
- In order to achieve all-same validity, a correct node must decide on its own value if another node supports that value.
- The third node might disagree, but that node could be byzantine. If correct node u has input 0 and correct node v has input 1, the byzantine node w can fool them by telling u that its value is 0 and telling v that its value is 1.
- This leads to u and v deciding on their own values, which results in violating the agreement condition.
- Even if u talks to v, and they figure out that they have different assumptions about w's value, u cannot distinguish whether w or v is byzantine.

# King's Algorithm

**Algorithm** King Algorithm (for f <n/3)

```
1: x = my input value
2: for phase = 1 to f + 1 do
3: Round 1 :
4:     Broadcast value(x)
5: Round 2 :
6:     if some value(y) at least n − f times then
7:         Broadcast propose(y)
8:     if some propose(z) received more than f times then
9:         x = z
10: Round 3 :
11:    Let node v_i be the predefined king of this phase i
12:    The king v_i broadcasts its current value w
13:    if received strictly less than n − f propose(x) then
14:        x = w
```

# Remarks

**All-Same Validity**

If all correct nodes start with the same value, all correct nodes propose it in Round 2. All correct nodes will receive at least $n - f$ proposals, i.e., all correct nodes will stick with this value, and never change it to the king's value. This holds for all phases.

# Remarks

**If a correct node proposes x, no other correct node proposes y, with $y \neq x$**

- Assume that a correct node proposes value x and another correct node proposes value y.
- Since a good node only proposes a value if it heard at least $n - f$ value messages, both nodes must have received their value from at least $n - 2f$ distinct correct nodes (as at most f nodes can behave byzantine and send x to one node and y to the other one).
- Hence, there must be a total of at least $2(n - 2f) + f = 2n - 3f$ nodes in the system. Using $3f < n$, we have $2n - 3f > n$ nodes, a contradiction.

# Remarks

**After a round with the correct king, the correct nodes will no longer change their value**

- If all correct nodes change their values to the king's value, all correct nodes have the same value.
- If some correct node does not change its value to the king's value, it received a proposal at least $n - f$ times, therefore at least $n - 2f$ correct nodes broadcasted this proposal.
- Thus, all correct nodes received it at least $n - 2f > f$ times (using $n > 3f$), therefore all correct nodes set their value to the proposed value, including the correct king.
- Note that only one value can be proposed more than $f$ times as shown earlier. Now that all the correct nodes have the same value they will not change their value further as in the first remark.

# Conclusion of Proof of King's Algo

- Since we have $f + 1$ phases, atleast 1 phase has a correct king.
- The king algorithm reaches agreement as either all correct nodes start with the same value, or they agree on the same value latest after the phase where a correct node was king as shown earlier.
- We know that once this happens they will stick to that value.
- Termination is guaranteed after $f + 1$ phases and all-same validity was shown earlier.

# References

Principles of Distributed Computing notes by Roger Wattenhofer