Distributed Computer Systems Engineering

CIS 508: Lecture 12
Byzantine Problem

Lecturer: Sid C.K. Chau
Email: ckchau@masdar.ac.ae

Masdar INSTITUTE CIS

Recall: What are Failures

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts

Omission failure
    Receive omission
    Send omission
	A server fails to respond to incoming requests
	A server fails to receive incoming messages
	A server fails to send messages

Timing failure	A server's response lies outside the specified time interval

Response failure
    Value failure
    State transition failure
	The server's response is incorrect
	The value of the response is wrong
	The server deviates from the correct flow of control

Arbitrary failure	**A server may produce arbitrary responses at arbitrary times**

- Fail-stop failure: Failed parties may notify (or detected by) others in advance
- Fail-silent failure: Not certain if a silent response is a failure or not
- *Arbitrary failure*: Byzantine failure, can be malicious and coordinated

Need for Theories

- Designing distributed computer systems is challenging
- *Big questions that consider arbitrary failures*
  - *What can be achieved (theoretically or practically)?*
  - *What cannot be achieved (thus no hope to go beyond that)?*
- Need for precise descriptions and problem formulations
  - Enabling formal theoretical studies and proofs
  - Obtaining mathematical insight to implement practical solution
- A widely studied problem: ***Byzantine agreement***
  - A simplified setting of fault tolerance consensus problem
  - Various applications in distributed computing:
    - Mission critical infrastructure control, aviation, military,

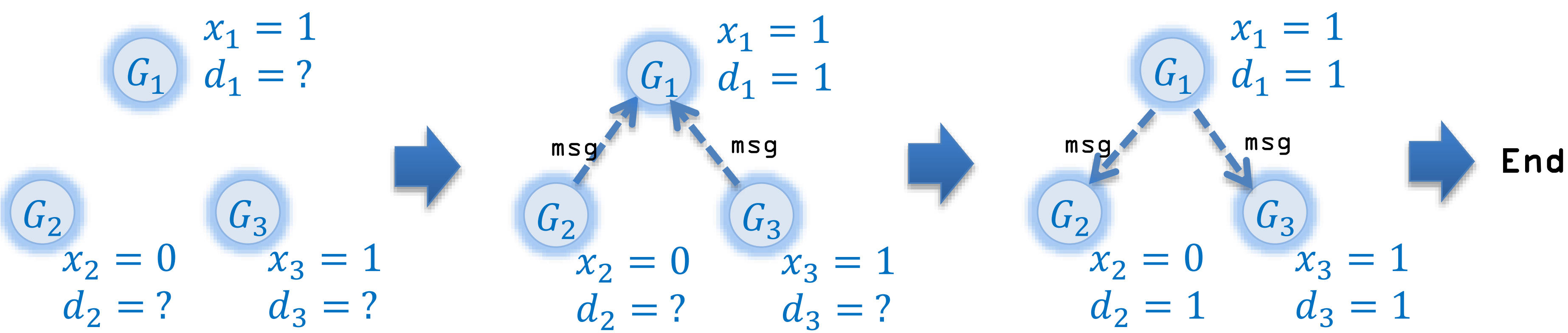
A little back story …

- In 1453 AD, city of Constantinople (controlled by Byzantine Empire) was under siege by troops of Ottoman Empire
- Ottoman battalions camped around the city preparing for attack
- The battalions could communicate with each other by (unreliable) messenger service
- Multiple battalions must attack together in order to win the battle
- Generals of Ottoman battalions must agree on a common time to launch the attack
- However, some (unidentified) generals were corrupted. Aimed at disrupting the attack, they would give false information to confuse the loyal generals
- **How can the Ottoman generals proceed to plan their attack in a distributed manner, in the presence of corrupted generals and unreliable messengers?**

Simple Model

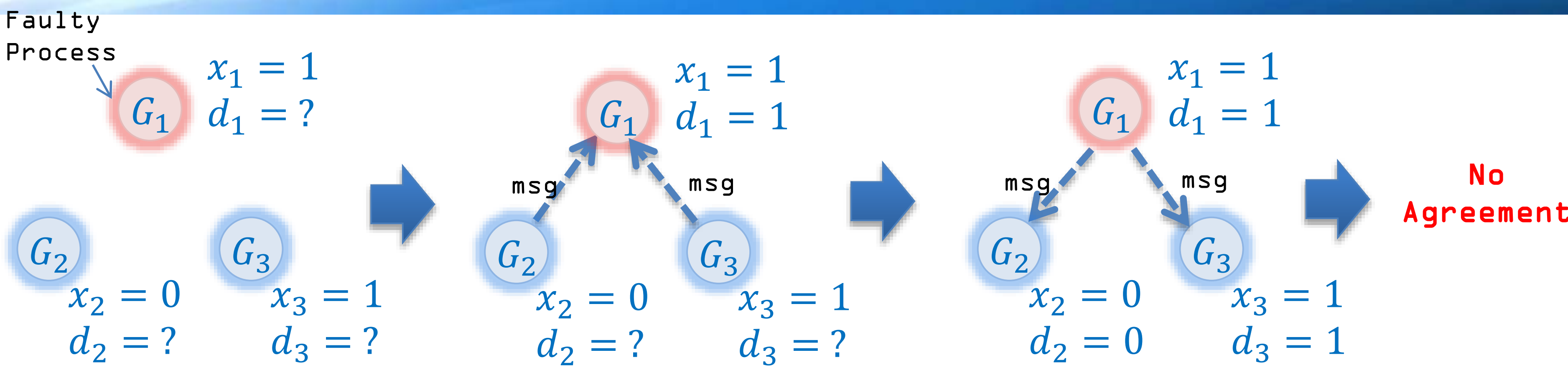
- A set of processes (*generals*): $G = \{G_1, G_2, \ldots, G_n\}$
  - A subset in G are correct processes (*loyal generals*) : $L \subseteq G$
  - A subset in G are faulty processes (*traitorous generals*) : $T \subseteq G$
  - Let $|G| = n$ and $|T| = t$
- Each process $G_i \in L$
  - has an initial *binary* value $x_i \in \{0,1\}$
  - must produce an *binary* output $d_i \in \{0,1\}$ at the end of consensus
- We assume that the communication between processes is
  - *synchronous:* the processes have perfectly synchronized clocks and a message is guaranteed to be delivered in one time unit
  - *reliable:* messages can neither be forged nor corrupted nor lost
  - *authenticated:* the identity of the sender is known to the receiver
  - *point-to-point:* the underlying topology is that of a complete graph

Desirable Protocol



- Processes have a protocol of message exchanges among them
- A desirable protocol shall satisfy:
  - *Non-triviality* : If all processes have the same input value x, then the only possible output of the normal processes is x. More formally,
    - If for all $G_i \in G$ such that $x_i = x$, then $d_i = x$ for all $G_i \in L$
  - *Agreement* : The loyal generals should agree on the decision. That is,
    - For all $G_i, G_j \in L$, we have $d_i = d_j$
  - *Finite running time* : The protocol must terminate
- Note that we **do not** assume voting by majority vote!

Faulty Process



- Remember that any processes can be *faulty*
- Wrong information can be propagated by faulty processes
  - Faulty processes may ignore or send inconsistent messages to others
  - Faulty processes may collude together to disrupt the correct processes
- Identities of faulty processes are unknown
- But we know that there are ***at most*** t faulty processes
  - And there can be no faulty process
- A protocol is ***t-resilient*** if it tolerates up to t faulty processes

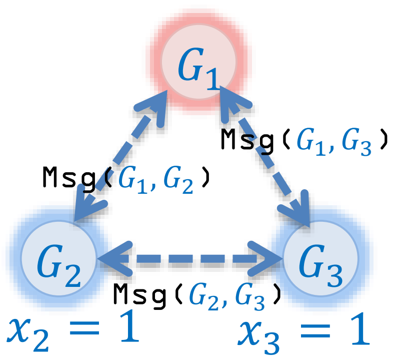
Impossibility Theory

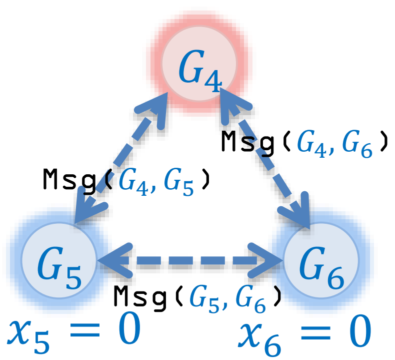
There is **no** t-resilient protocol for Byzantine agreement, when $t \geq n/3$

- Implications:
  - Impossible to design a resilient protocol that can tolerate more than one third of faulty processes
  - Ability to tolerate more faults requiring weaken some of assumptions
- Basic idea of proof
  - Suppose that there exists such a t-resilient protocol
  - Find an instance of execution under the assumption of *non-triviality, agreement, finite running time* that can generate a contradiction

Impossibility Theory

There is **no** t-resilient protocol for Byzantine agreement, when $t \geq n/3$





- We first assume that there are 3 processes ($n = 3$)
- No correct process knows the identity of the faulty process
- Suppose that there exists a 1-resilient protocol satisfying
  - *non-triviality, agreement, finite running time*
- We consider three cases:

1) $G_1 \in T; G_2, G_3 \in L; x_2 = x_3 = 1$:

   Whatever G_1 tells G_2 and G_3, at the end $d_2 = d_3 = 1$

   Because the protocol satisfies non-triviality

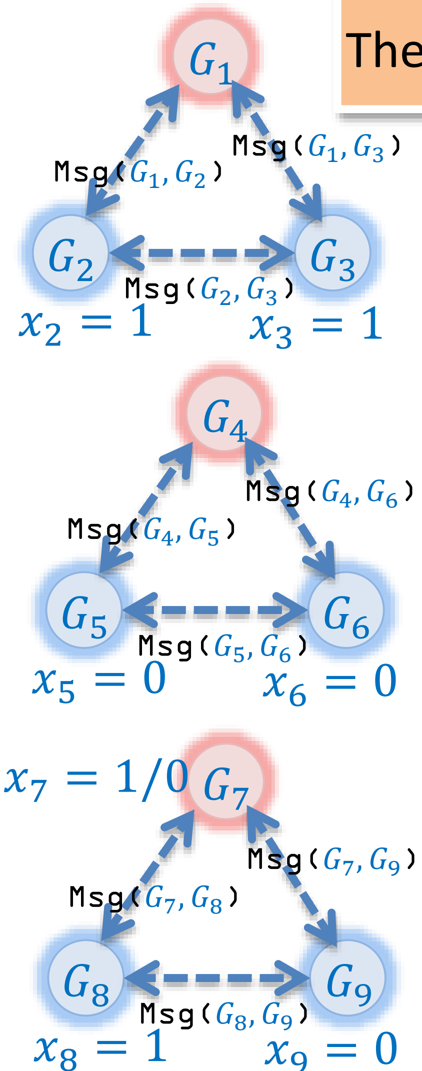
2) $G_4 \in T; G_5, G_6 \in L; x_5 = x_6 = 0$:

   Whatever G_4 tells G_5 and G_6, at the end $d_5 = d_6 = 0$

   Because the protocol satisfies non-triviality

3) …

Impossibility Theory

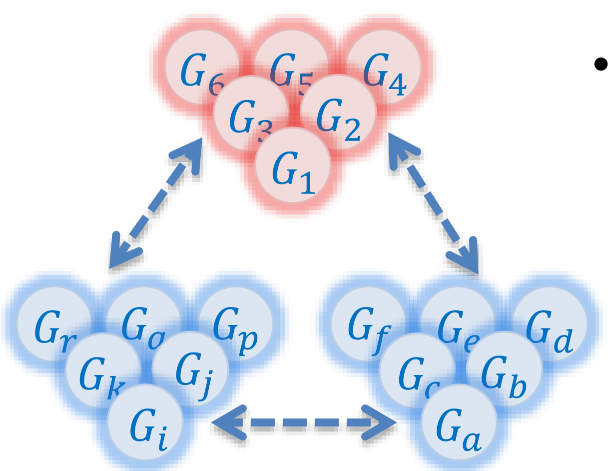


There is **no** t-resilient protocol for Byzantine agreement, when $t \geq n/3$

- We consider three cases:
  1) $G_1 \in T; G_2, G_3 \in L; x_2 = x_3 = 1; d_2 = d_3 = 1$
  2) $G_4 \in T; G_5, G_6 \in L; x_5 = x_6 = 0; d_5 = d_6 = 0$
  3) $G_7 \in T; G_8, G_9 \in L; x_8 = 1; x_9 = 0$:
  - Suppose G_7 tells G_8 in the same manner as G_3 tells G_2 in Case 1). That is, G_7 reacts to G_8 as if $x_7 = 1$
  - Meanwhile, suppose G_7 tells G_9 in the same manner as G_5 tells G_6 in Case 2). That is, G_7 reacts to G_9 as if $x_7 = 0$
  - From Case 1), at the end $d_8 = 1$
  - From Case 2), at the end $d_9 = 0$
  - Contradiction! The protocol cannot satisfy agreement $d_8 \neq d_9$

Impossibility Theory

There is **no** t-resilient protocol for Byzantine agreement, when $t \geq n/3$



- We consider $n = 3t > 3$:
- We can divide the processes into three equal groups
  - All faulty processes into one group
  - Other correct processes into two other groups
  - Pick one leader from each group
  - Suppose all processes in a group have same initial value x_i as the leader
- We seek the agreement among the three leaders
  - Equivalent to Byzantine agreement of 3 processes
- If there exists t-resilient protocol for $n = 3t > 3$, then there exists 1-resilient protocol for $n = 3$
- Contradiction!

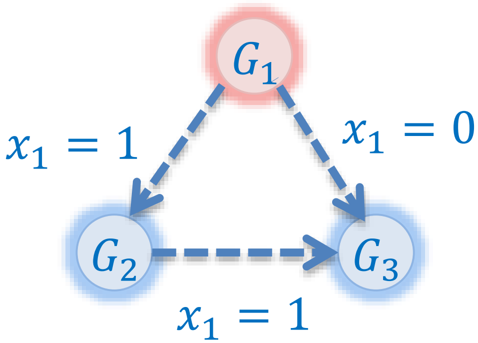
Any solution?

- Impossibility theory implies $t < n/3$ is necessary for Byzantine agreement
- Does there really exist a t-resilient protocol, when $t < n/3$?
- Answer is *Yes*!
- Assume some deterministic tie-breaking rule (e.g. 1 always wins)

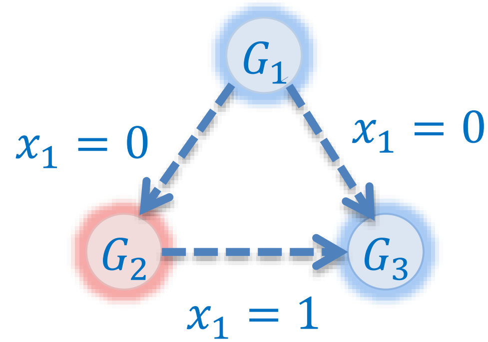
A simple gossiping based solution:

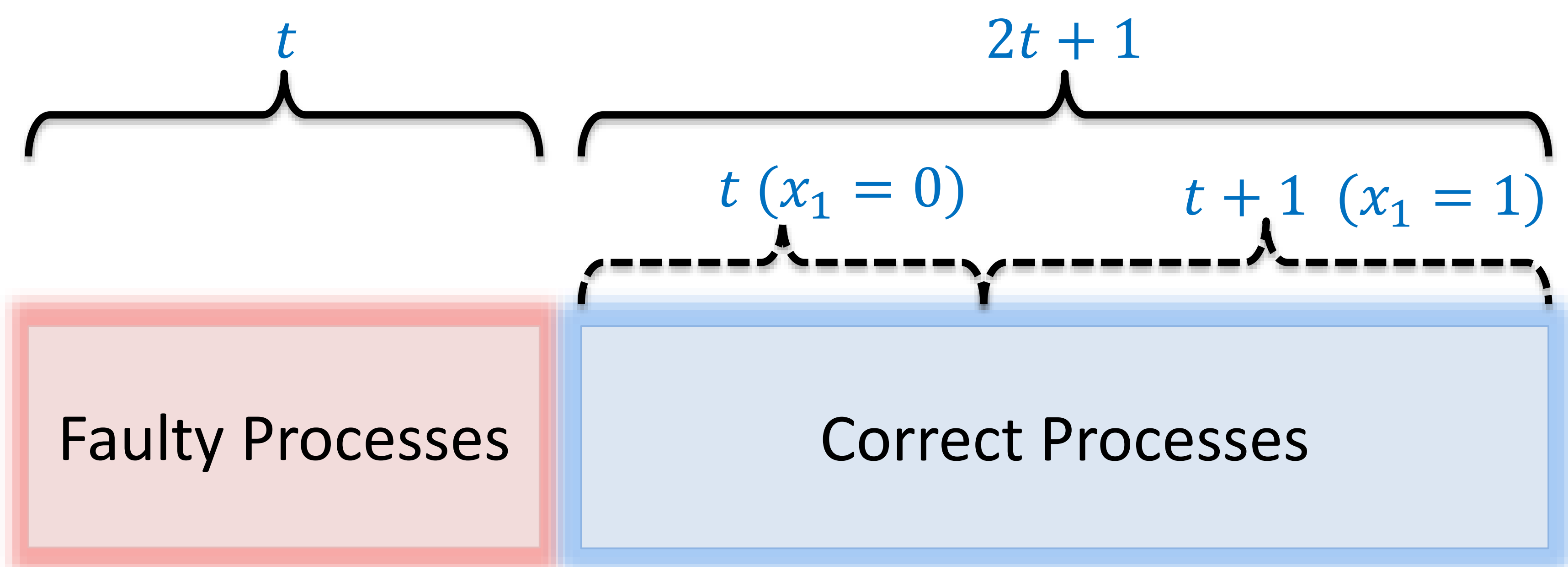
1. Each process will re-announce the initial values that are announced by others
2. Each process will keep track of all the announced and re-announced initial values
3. Majority vote will be used to determine the correct initial values
4. Since the number of faulty processes $t < n/3$, correct processes always win, even if there is an even split of votes among correct process

Intuition



Inconsistency

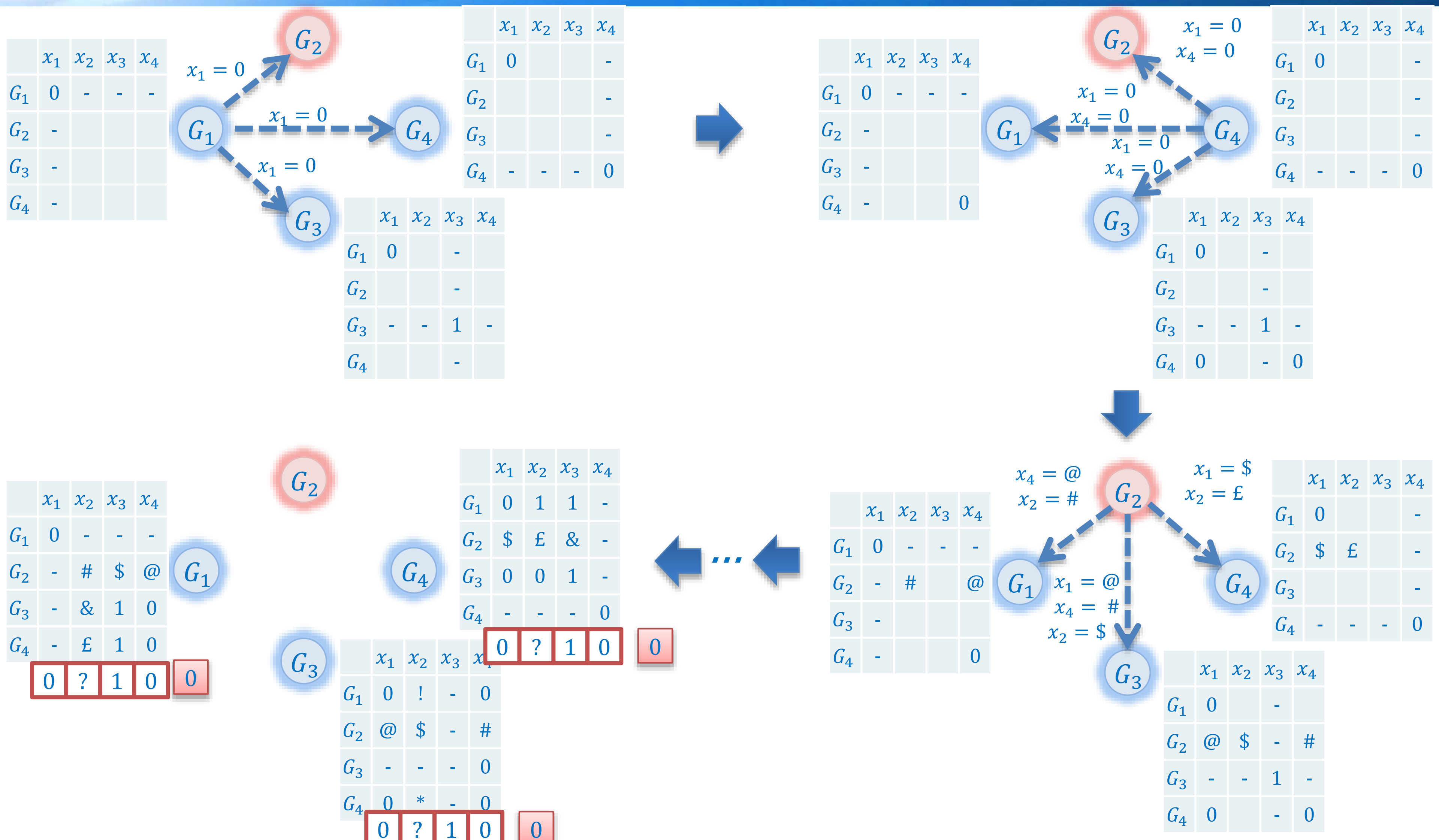




A Simple Solution

1. Each process G_i announces its initial value x_i to all other processes
2. Iteratively, each process G_i announces $x_j(k)$ to other process G_j that is learnt from other process G_k in previous steps
3. Each process G_i constructs a table of all the initial values $x_j(k)$ for other process G_j learnt from other process G_k
4. Each process G_i carries out majority vote to determine the initial value x_j for other process j from the table using $\{x_j(k): G_k \in G \backslash \{G_i, G_j\}\}$
5. A final majority vote is carried out on the all determined initial values of all processes $\{x_j: G_j \in G\}$ to determine output d_i

A simple solution



	x_1	x_2	x_3	x_4
G_1	0	-	-	-
G_2	-			
G_3	-			
G_4	-			

	x_1	x_2	x_3	x_4
G_1	0			-
G_2				-
G_3				-
G_4	-	-	-	0

	x_1	x_2	x_3	x_4
G_1	0		-	
G_2			-	
G_3	-	-	1	-
G_4			-	

	x_1	x_2	x_3	x_4
G_1	0	-	-	-
G_2	-			
G_3	-			
G_4	-			0

	x_1	x_2	x_3	x_4
G_1	0			-
G_2				-
G_3				-
G_4	-	-	-	0

	x_1	x_2	x_3	x_4
G_1	0		-	
G_2			-	
G_3	-	-	1	-
G_4	0		-	0

	x_1	x_2	x_3	x_4
G_1	0	-	-	-
G_2	-	#		@
G_3	-			
G_4	-			0

	x_1	x_2	x_3	x_4
G_1	0			-
G_2	$	£		-
G_3				-
G_4	-	-	-	0

	x_1	x_2	x_3	x_4
G_1	0		-	
G_2	@	$	-	#
G_3	-	-	1	-
G_4	0		-	0

	x_1	x_2	x_3	x_4
G_1	0	-	-	-
G_2	-	#	$	@
G_3	-	&	1	0
G_4	-	£	1	0

0 ? 1 0 → 0

	x_1	x_2	x_3	x_4
G_1	0	1	1	-
G_2	$	£	&	-
G_3	0	0	1	-
G_4	-	-	-	0

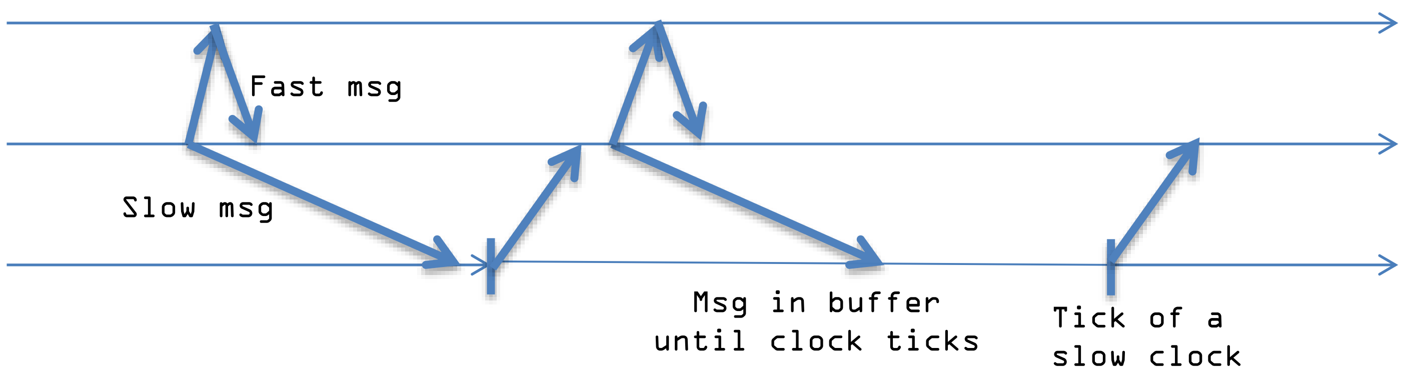
0 ? 1 0 → 0

	x_1	x_2	x_3	x_4
G_1	0	!	-	0
G_2	@	$	-	#
G_3	-	-	-	0
G_4	0	*	-	0

0 ? 1 0 → 0

Asynchronous Communication

- In asynchronous message-passing systems
  - Messages can be delayed in delivery for arbitrarily long time
  - Messages can be re-ordered in arbitrary manner
    - First message can come last
  - Processes can be crashed, or restart with no previous memory
  - Each process runs according to its own clock
    - Clocks not synchronized



Asynchronous Communication

- Intuition of asynchrony:
  - Processes cannot tell whether another process is crashed or just messages delayed
  - If they wait, then they might wait forever
  - If they decide and wait no more, they might find that the other process can come to a different decision if they decide to wait (or their clocks run slower)
  - Handling asynchrony is impossible

There is **no** 1-resilient protocol for Byzantine agreement in the presence of asynchrony, for any n

References

- Reference book
  - Introduction to Distributed Algorithms
    Gerard Tel, Cambridge University Press