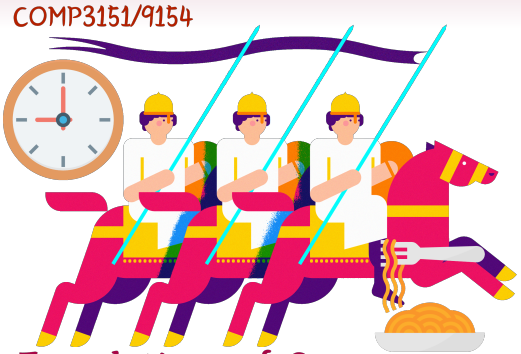


Commitment and Consensus

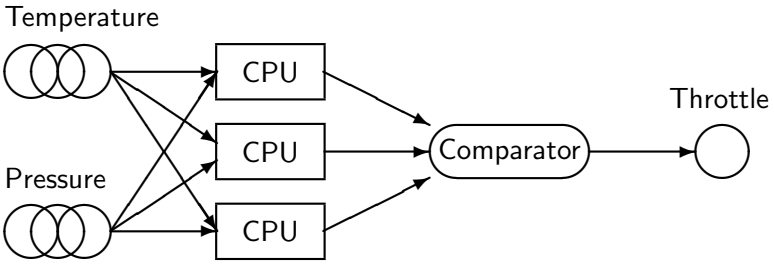
Johannes Åman Pohjola
UNSW
Term 2 2022

Where we're at

Last week, we look at distributed mutual exclusion. We assumed that nodes never crashed or went rogue.

Today, we'll look at *consensus* protocols, where nodes may crash or go rogue.

Architecture for a Fault-Tolerant System



Failure Models

In the literature there are many more, but we focus on two extremes:

Crash failures failed nodes stop sending messages

Byzantine failures failed nodes can send arbitrary messages, even according to a malevolent plan

Commitment Problem

Suppose n agents collaborate on a database transaction.

After each agent has done its share, the agents must reach agreement on whether to commit the transaction or abort it.

Each agent can be assumed to have formed an initial vote, but not yet made a final decision.

We must ensure that no two agents make different decisions.

Commitment Spec

1. All agents that reach a decision reach the same one.
2. If an agent decides to commit, then all agents voted to commit.
3. If there are no failures and all agents voted to commit, then the decision reached is to commit.

Failure model: Only agents can fail. When they do, they crash.

2-Phase Commit Algorithm

One distinguished agent, say, agent 1, collects the other agents' votes. If all votes (including agent 1's) are "commit" then agent 1 tells all other agents to commit, otherwise, to abort. Note that "otherwise" could also mean that some agent crashed before being able to communicate its vote to agent 1.

Theorem

2-Phase Commit solves the commitment problem but may not terminate if processes fail.

2-Phase Commit Algorithm

One distinguished agent, say, agent 1, collects the other agents' votes. If all votes (including agent 1's) are "commit" then agent 1 tells all other agents to commit, otherwise, to abort. Note that "otherwise" could also mean that some agent crashed before being able to communicate its vote to agent 1.

Theorem

2-Phase Commit solves the commitment problem but may not terminate if processes fail.

But how do we know that an agent crashed? What if their message was just slow to arrive?

1st Consensus Problem

A group of Byzantine armies is surrounding an enemy city. The balance of force is such that if they all attack together, they can capture the city; otherwise they must retreat in order to avoid defeat.
The generals of the armies have reliable messengers who successfully deliver any mesage sent from one general to another. However, some of the generals may be traitors endeavouring to bring about defeat of the Byzantine armies.

Bad News

Quoting and adapting from the PODC webpage for the *influential paper award 2001*

Theorem (Fischer, Lynch and Paterson 1995)

It is impossible for a set of nodes in an asynchronous distributed system to agree on a binary value, even if only a single node is subject to an unannounced crash.

Bad News

Quoting and adapting from the PODC webpage for the *influential paper award 2001*

Theorem (Fischer, Lynch and Paterson 1995)

It is impossible for a set of nodes in an asynchronous distributed system to agree on a binary value, even if only a single node is subject to an unannounced crash.

Proof sketch

Initially, either decision, `0` *or* `1`*, is possible. Assume a correct algorithm. At some point in time the system as a whole must commit to one value or the other. That commitment must result from some action of a single process. Suppose that process fails. Then there is no way for the other processes to know the commitment value; hence, they will sometimes make the wrong decision. Contradiction!*

Bad News

Quoting and adapting from the PODC webpage for the *influential paper award 2001*

Theorem (Fischer, Lynch and Paterson 1995)

It is impossible for a set of nodes in an asynchronous distributed system to agree on a binary value, even if only a single node is subject to an unannounced crash.

Proof sketch

Initially, either decision, `0` *or* `1`*, is possible. Assume a correct algorithm. At some point in time the system as a whole must commit to one value or the other. That commitment must result from some action of a single process. Suppose that process fails. Then there is no way for the other processes to know the commitment value; hence, they will sometimes make the wrong decision. Contradiction!*

In order to get anywhere near consensus in the presence of faults, we need a bit of synchrony. With some synchrony, we can do much better and even tolerate some Byzantine faults.

Consider

- *crash* failures (assumed detectable eg by time-out) and
- *Byzantine* failures (detectable at most indirectly).

Task

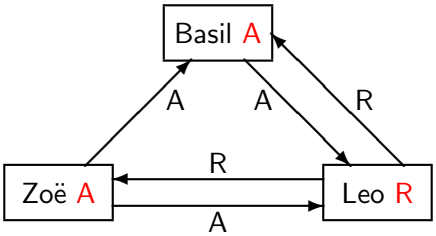
Devise an algorithm so that the *loyal* generals come to a consensus on a plan. The final decision should be almost the same as a majority vote of their initial choices; if the vote is tied, the final decision is to retreat.

planType $= \{A, R\}$ for attack and retreat.

Algorithm 2.1: Consensus—one-round algorithm	
	planType finalPlan planType array[generals] plan
p1:	plan[myID] $\leftarrow$ chooseAttackOrRetreat
p2:	for all *other* generals G
p3:	send(G, myID, plan[myID])
p4:	for all *other* generals G
p5:	receive(G, plan[G])
p6:	finalPlan $\leftarrow$ majority(plan)

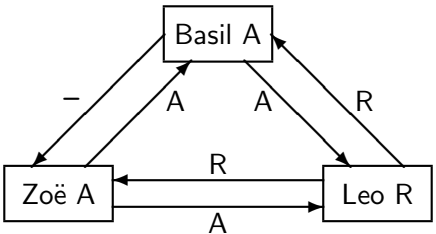
Messages Sent in a One-Round Algorithm

Suppose Basil crashed after sending to Leo but before sending to Zoë.



Messages Sent in a One-Round Algorithm

Suppose Basil crashed after sending to Leo but before sending to Zoë.



Data Structures in a One-Round Algorithm

Leo	
general	plan
Basil	A
Leo	R
Zoë	A
majority	

Zoë	
general	plans
Basil	–
Leo	R
Zoë	A
majority	

Data Structures in a One-Round Algorithm

Leo	
general	plan
Basil	A
Leo	R
Zoë	A
majority	A

Zoë	
general	plans
Basil	–
Leo	R
Zoë	A
majority	R

Thus, the One-Round Algorithm cannot even handle a single crash failure among 3 generals.

Idea

Relay received messages in a further round.

Algorithm 2.2: Consensus—Byzantine Generals algorithm

```
                    planType finalPlan
                    planType array[generals] plan, majorityPlan
                    planType array[generals, generals] reportedPlan
p1:  plan[myID] ← chooseAttackOrRetreat
p2:  for all other generals G                          // First round
p3:     send(G, myID, plan[myID])
p4:  for all other generals G
p5:     receive(G, plan[G])
p6:  for all other generals G                          // Second round
p7:     for all other generals G' except G
p8:        send(G', myID, G, plan[G])
p9:  for all other generals G
p10:    for all other generals G' except G
p11:       receive(G, G', reportedPlan[G, G'])
p12: for all other generals G                          // First vote
p13:    majorityPlan[G] ← majority(plan[G] ∪ reportedPlan[*, G])
p14: majorityPlan[myID] ← plan[myID]                   // Second vote
p15: finalPlan ← majority(majorityPlan)
```

Data Structure for Crash Failure - First Scenario (Leo)

Leo				
general	plan	reported by		majority
		Basil	Zoë	
Basil	A		–	A
Leo	R			R
Zoë	A	–		A
majority				A

Basil sent his first-round vote to Leo, but crashed before sending it to Zoë.

Data Structure for Crash Failure - First Scenario (Zoë)

Zoë				
general	plan	reported by		majority
		Basil	Leo	
Basil	–		**A**	A
Leo	R	–		R
Zoë	A			A
majority				A

Basil sent his first-round vote to Leo, but crashed before sending it to Zoë.

Data Structure for Crash Failure - Second Scenario (Leo)

Leo				
general	plan	reported by		majority
		Basil	Zoë	
Basil	A		A	A
Leo	R			R
Zoë	A	A		A
majority				A

Basil sent his second-round vote to Leo, but crashed before sending it to Zoë.

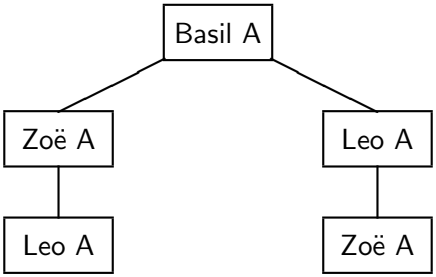
Data Structure for Crash Failure - Second Scenario (Zoë)

Zoë				
general	plan	reported by		majority
		Basil	Leo	
Basil	A		A	A
Leo	R	–		R
Zoë	A			A
majority				A

Basil sent his second-round vote to Leo, but crashed before sending it to Zoë.

Knowledge Tree about Basil for Crash Failure - First Scenario

Suppose Basil has chosen A and sends all messages.

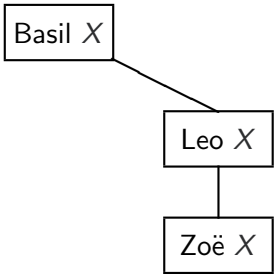


A concise representation of what is known about the root node of the tree; in traditional epistemic logic with a *knowledge modality* K_i and attack predicate A_i for each general i we'd write:

$$A_{\text{Basil}} \wedge K_{\text{Zoë}} A_{\text{Basil}} \wedge K_{\text{Leo}} A_{\text{Basil}} \wedge K_{\text{Leo}} K_{\text{Zoë}} A_{\text{Basil}} \wedge K_{\text{Zoë}} K_{\text{Leo}} A_{\text{Basil}}$$

Knowledge Tree about Basil for Crash Failure - Second Scenario

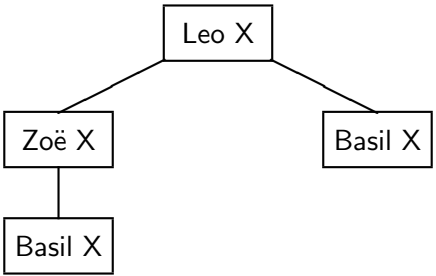
Suppose Basil has chosen $X \in \{A, R\}$ and crashes right before sending the 1st round message to Zoë but after sending it to Leo.



$$X_{\text{Basil}} \wedge K_{\text{Leo}} X_{\text{Basil}} \wedge K_{\text{Zoë}} K_{\text{Leo}} X_{\text{Basil}}$$

Knowledge Tree about Leo for Crash Failure

Suppose Leo has chosen X and Basil crashes before sending the 2nd round message to Zoë.



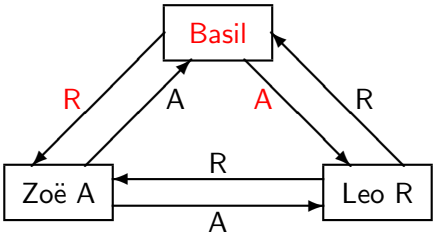
$$X_{\text{Leo}} \wedge K_{\text{Zoë}} X_{\text{Leo}} \wedge K_{\text{Basil}} X_{\text{Leo}} \wedge K_{\text{Basil}} K_{\text{Zoë}} X_{\text{Leo}}$$

Then what?

It seems like the Byzantine Generals protocol is robust to one crash amongst three generals.

What about Byzantine faults?

Messages Sent for Byzantine Failure with Three Generals



Data Stuctures for Leo and Zoë After First Round

Leo	
general	plans
Basil	A
Leo	R
Zoë	A
majority	A

Zoë	
general	plans
Basil	R
Leo	R
Zoë	A
majority	R

Data Stuctures for Leo After Second Round

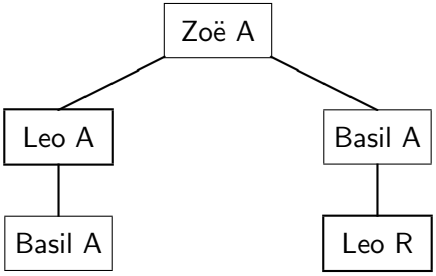
Leo				
general	plans	reported by		majority
		Basil	Zoë	
Basil	A		A	A
Leo	R			R
Zoë	A	**R**		R
majority				R

Data Stuctures for Zoë After Second Round

Zoë				
general	plans	reported by		majority
		Basil	Leo	
Basil	A		A	A
Leo	R	R		R
Zoë	A			A
majority				A

Thus the Byzantine Generals Algorithm is incorrect for 3 generals of which 1 is a traitor.

Knowledge Tree About Zoë



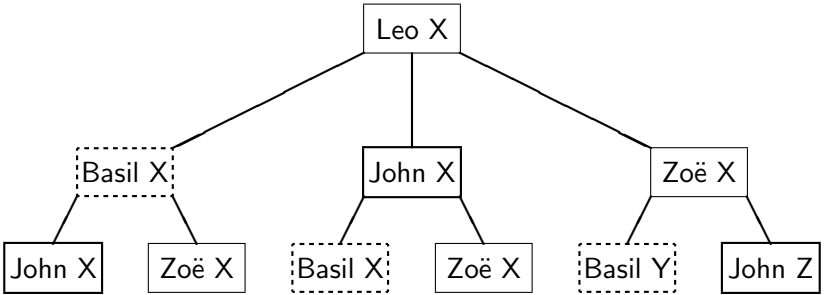
Four Generals: Data Structure of Basil (1)

Basil					
general	plan	reported by			majority
		John	Leo	Zoë	
Basil	A				A
John	**A**		**A**	?	A
Leo	R	R		?	R
Zoë	?	?	?		?
majority					?

Four Generals: Data Structure of Basil (2)

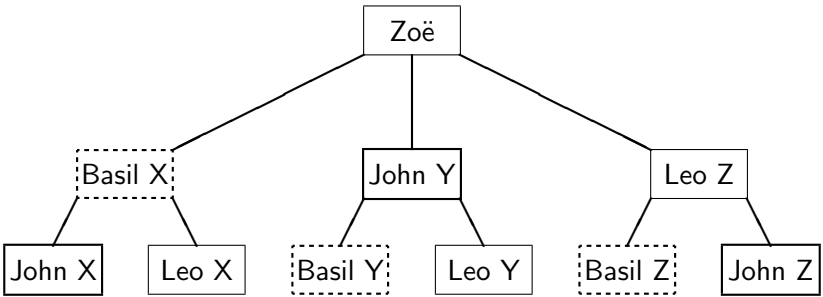
Basil					
general	plans	reported by			majority
		John	Leo	Zoë	
Basil	A				A
John	A		A	?	A
Leo	R	R		?	R
Zoë	**R**	**A**	**R**		R
					R

Knowledge Tree About Loyal General Leo



Note that, with Byzantine failures, trees may represent possibly untrue knowledge (AKA *belief* in the literature).

Knowledge Tree About Traitor Zoë



Complexity of the Byzantine Generals Algorithm

Generalising to any number of generals works as long a round of messages is added for every extra traitor and as long as there are at least $n = 3t + 1$ generals, where t is the maximum number of treacherous generals.
The resulting numbers of messages are $\mathcal{O}(n^4)$.

traitors	generals	messages
1	4	36
2	7	392
3	10	1790
4	13	5408

If we're only worried about crash failures a much simpler algorithm suffices:

Algorithm 2.3: Consensus - flooding algorithm

```
                    planType finalPlan
                    set of planType plan ← { chooseAttackOrRetreat }
                    set of planType receivedPlan
p1: do t + 1 times
p2:     for all other generals G
p3:         send(G, plan)
p4:     for all other generals G
p5:         receive(G, receivedPlan)
p6:         plan ← plan ∪ receivedPlan
p7: finalPlan ← majority(plan)
```

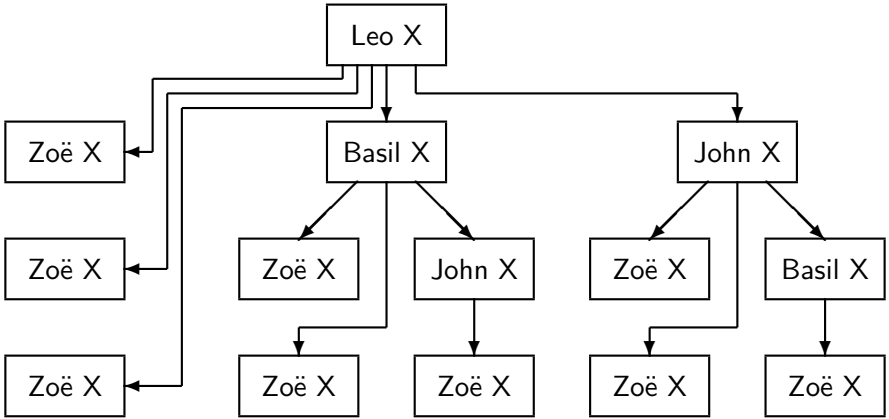
If we're only worried about crash failures a much simpler algorithm suffices:

Algorithm 2.4: Consensus - flooding algorithm

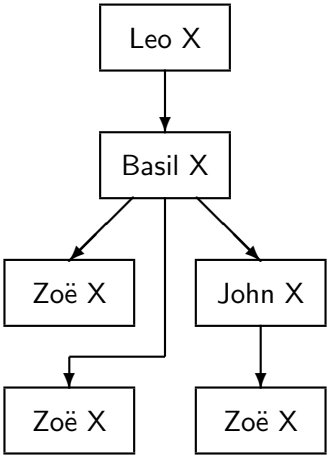
```
                    planType finalPlan
                    set of planType plan ← { chooseAttackOrRetreat }
                    set of planType receivedPlan
p1: do t + 1 times
p2:     for all other generals G
p3:         send(G, plan)
p4:     for all other generals G
p5:         receive(G, receivedPlan)
p6:         plan ← plan ∪ receivedPlan
p7: finalPlan ← majority(plan)
```

t is the number of crashes we can tolerate.

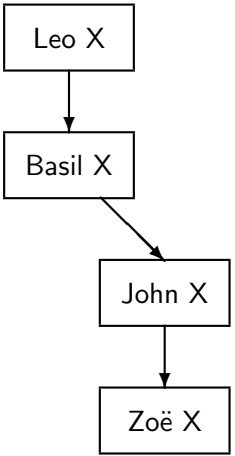
Flooding Algorithm with No Crash: What Zoë Knows about Leo



Flooding Algorithm with Crash: Knowledge Tree About Leo (1)



Flooding Algorithm with Crash: Knowledge Tree About Leo (2)



Algorithm 2.5: Consensus - King algorithm

```
                    planType finalPlan, myMajority, kingPlan
                    planType array[generals] plan
                    integer votesMajority
p1: plan[myID] ← chooseAttackOrRetreat

p2: do two times
p3:     for all other generals G              // First and third rounds
p4:         send(G, myID, plan[myID])
p5:     for all other generals G
p6:         receive(G, plan[G])
p7:     myMajority ← majority(plan)
p8:     votesMajority ← number of votes for myMajority
```

Algorithm 2.5: Consensus - King algorithm (continued)

```
p9:     if my turn to be king                // Second and fourth rounds
p10:       for all other generals G
p11:          send(G, myID, myMajority)
p12:       plan[myID] ← myMajority
        else
p13:       receive(kingID, kingPlan)
p14:       if votesMajority > 3
p15:          plan[myID] ← myMajority
           else
p16:          plan[myID] ← kingPlan

p17: finalPlan ← plan[myID]                  // Final decision
```

Scenario for King Algorithm - First King Loyal General Zoë (1)

Basil							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
A	A	R	R	R	R	3	

John							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
A	A	R	A	R	A	3	

Leo							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
A	A	R	A	R	A	3	

Zoë							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
A	A	R	R	R	R	3	

Scenario for King Algorithm - First King Loyal General Zoë (2)

Basil							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
R							R

John							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
	R						R

Leo							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
		R					R

Zoë							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
				R			

Scenario for King Algorithm - First King Loyal General Zoë (3)

Basil							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
R	R	R	?	R	R	4–5	

John							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
R	R	R	?	R	R	4–5	

Leo							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
R	R	R	?	R	R	4–5	

Zoë							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
R	R	R	?	R	R	4–5	

Scenario 2 for King Algorithm - First King Traitor Irene (1)

Basil							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
R							R

John							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
	A						A

Leo							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
		A					A

Zoë							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
				R			R

Scenario 2 for King Algorithm - First King Traitor Irene (2)

Basil							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
R	A	A	?	R	?	3	

John							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
R	A	A	?	R	?	3	

Leo							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
R	A	A	?	R	?	3	

Zoë							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
R	A	A	?	R	?	3	

Scenario 2 for King Algorithm - First King Traitor Irene (3)

Basil							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
A							A

John							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
	A						A

Leo							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
		A					A

Zoë							
Basil	John	Leo	Irene	Zoë	myMajority	votesMajority	kingPlan
				A			

Complexity of Byzantine Generals and King Algorithms

traitors	BG generals	BG messages	King generals	King messages
1	4	36	5	48
2	7	392	9	240
3	10	1790	13	672
4	13	5408	17	1440

Impossibility Results

The bounds known for (simultaneous) Byzantine agreement are tight.

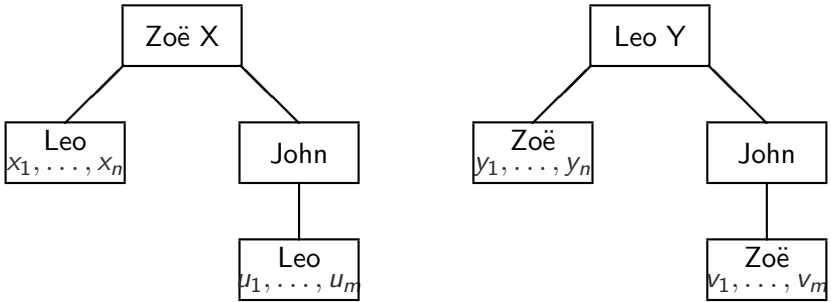
Theorem

Simultaneous Byzantine agreement for n generals of which up to t are traitors requires $n > 3t$ and at least $t + 1$ rounds of communication.

Simultaneous means all generals decide in the same round.

Impossibility with Three Generals (1)

We check the condition $n > 3t$ in the special case of $t = 1$. John is the traitor. Zoë has plan X and Leo has plan Y. Their knowledge trees are:



where the u_i are the messages John sent to Leo about his own and Zoë's plans, etc.

Impossibility with Three Generals (2)

Since the loyal generals are required to correctly infer the plans of other loyal generals (by applying some function f to their state), we have that Leo reasons about Zoë's plan

$$f(x_1, \ldots, x_n, u_1, \ldots, u_m) = X \tag{1}$$

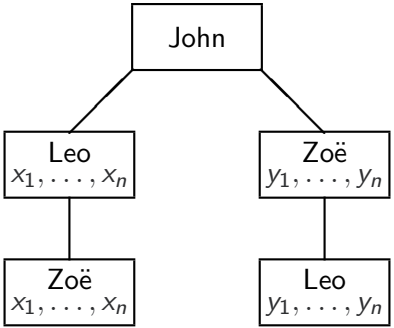
and that Zoë reasons about Leo's plan

$$f(y_1, \ldots, y_n, v_1, \ldots, v_n) = Y \tag{2}$$

If $X \neq Y$, then Zoë and Leo must also come to a consensus on John's plan.

Impossibility with Three Generals (3)

The traitor John can use y_i for u_i and x_i for v_i, which leads to the loyal generals thinking



about John's plan. Using those messages in (1) and (2) gives $X = Y$, contradiction!

One round is not enough

Theorem

Let $n \geq 3$. Then there is no n-process simultaneous crash-failure agreement algorithm that tolerates one fault in which all loyal generals always decide by the end of round 1.

Corollary

Let $n \geq 3$. Then there is no n-process simultaneous Byzantine agreement algorithm that tolerates one traitor in which all loyal generals always decide by the end of round 1.

What now?

Even more distributed algorithms!