



Chapter 6
Consensus
Distributed Systems
SS 2015
Fabian Kuhn

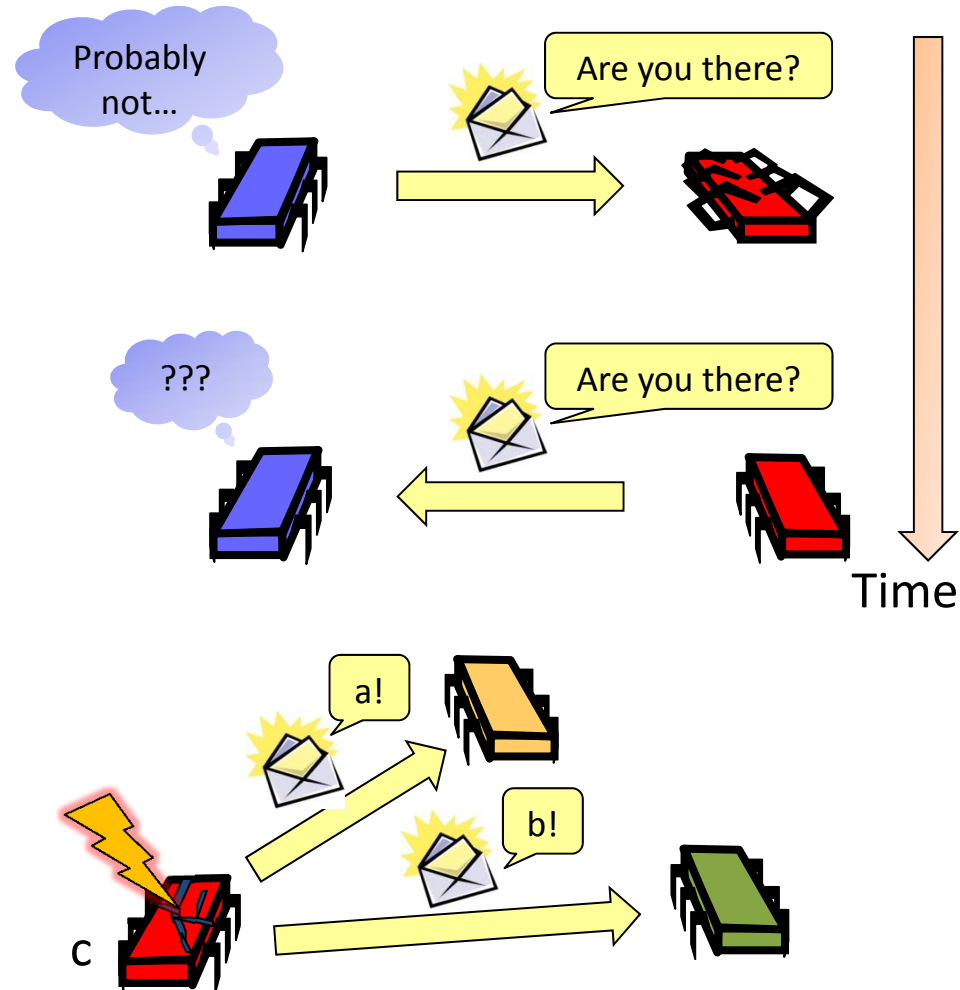
Overview

- Introduction
- Consensus #1: Shared Memory
- Consensus #2: Wait-free Shared Memory
- Consensus #3: Read-Modify-Write Shared Memory
- Consensus #4: Synchronous Systems
- Consensus #5: Byzantine Failures
- Consensus #6: A Simple Algorithm for Byzantine Agreement
- Consensus #7: The Queen Algorithm
- Consensus #8: The King Algorithm
- Consensus #9: Byzantine Agreement Using Authentication
- Consensus #10: A Randomized Algorithm
- Shared Coin

- Slides by R. Wattenhofer (ETHZ)

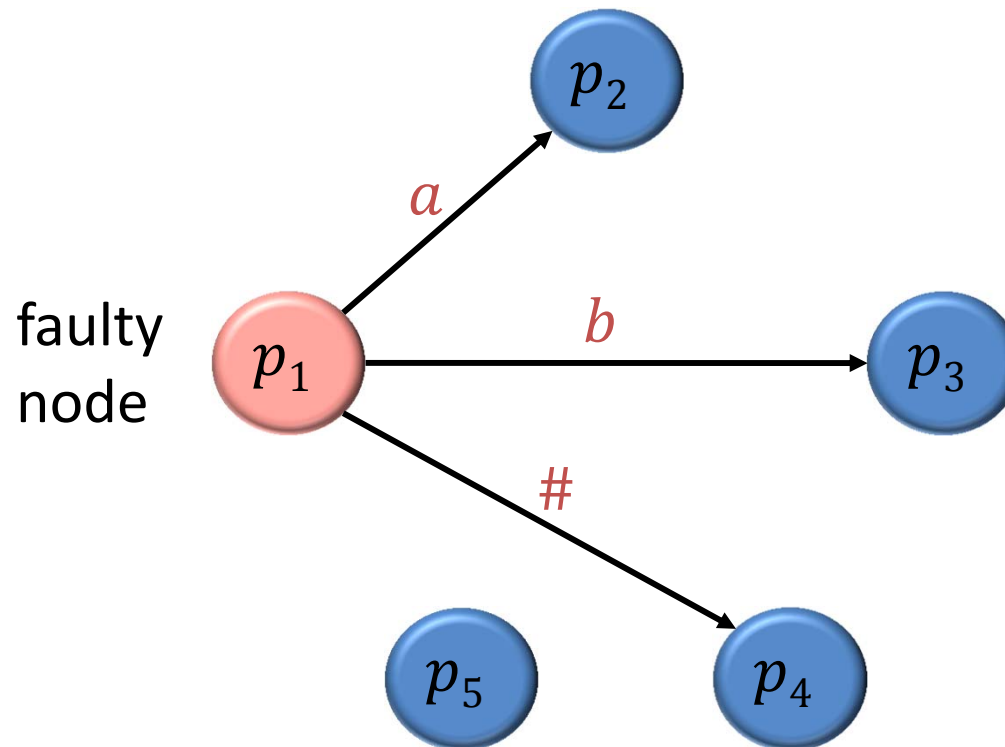
Arbitrary Behavior

- The assumption that processes crash and stop forever is sometimes too optimistic
- Maybe the processes fail and recover:
- Maybe the processes are damaged:

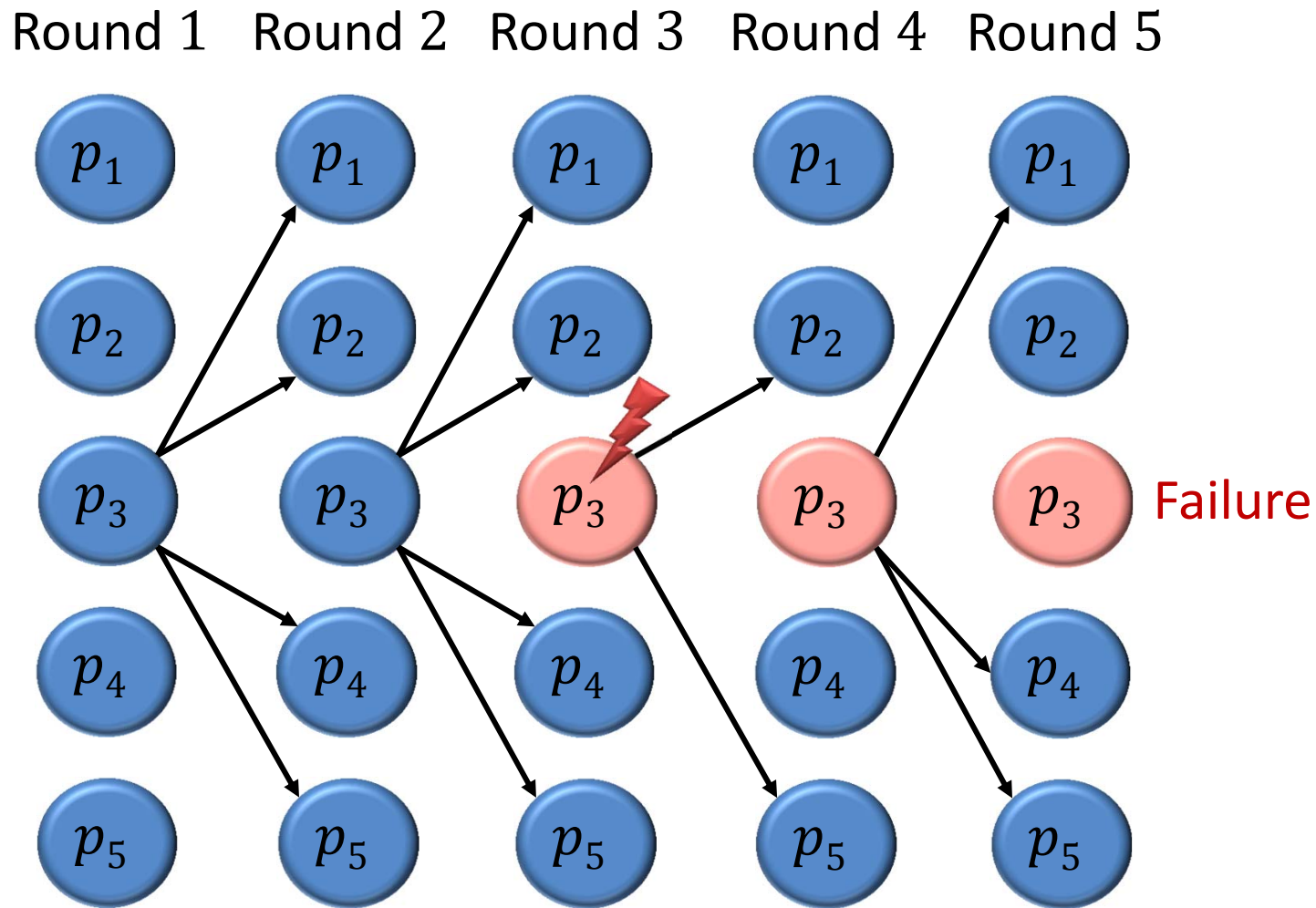


Consensus #5: Byzantine Failures

- Different processes may receive different values
- A Byzantine process can behave like a crash-failed process



After Failure, Node Remains in Network



Consensus with Byzantine Failures

- Again: If an algorithm solves consensus for f failed processes, we say it is an f -resilient consensus algorithm
- **Validity:** If all non-faulty processes start with the same value, then all non-faulty processes decide on that value
 - Note that in general this validity condition does not guarantee that the final value is an input value of a non-Byzantine process
 - However, if the input is binary, then the validity condition ensures that processes decide on a value that at least one non-Byzantine process had initially
- Obviously, any f -resilient consensus algorithm requires at least $f + 1$ rounds (follows from the crash failure lower bound)
- How large can f be...? Can we reach consensus as long as the majority of processes is correct (non-Byzantine)?

Impossibility



Theorem

There is no f -resilient Byzantine consensus algorithm for n nodes for $f \geq n/3$

Proof outline

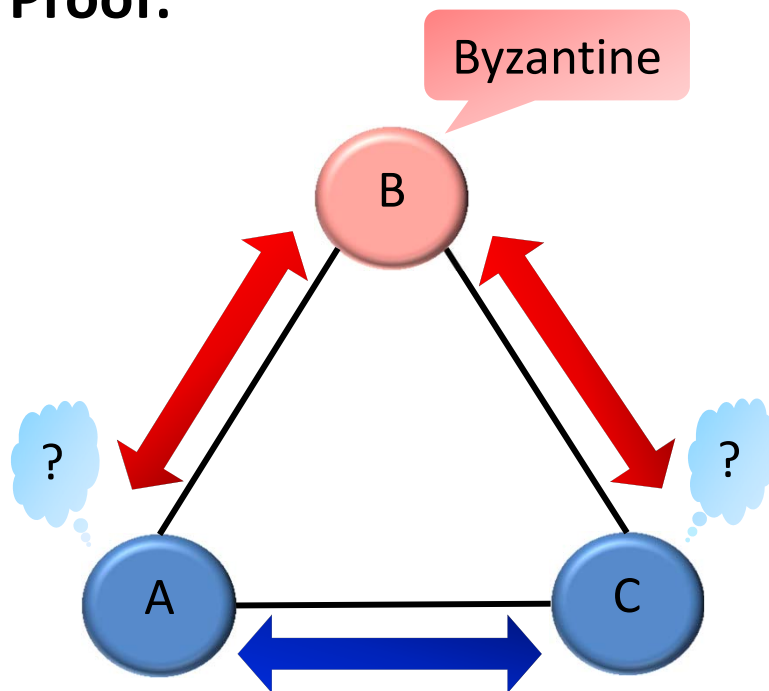
- First, we prove the 3 node case
 - not possible for $f = 1$
- The general case can then be proved by reduction from the 3 node case
 - Given an algorithm for n node and f faults for $f \geq n/3$, we can construct a 1-resilient 3-node algorithm

The 3 Node Case

Lemma

There is no 1-resilient algorithm for 3 nodes

Proof:

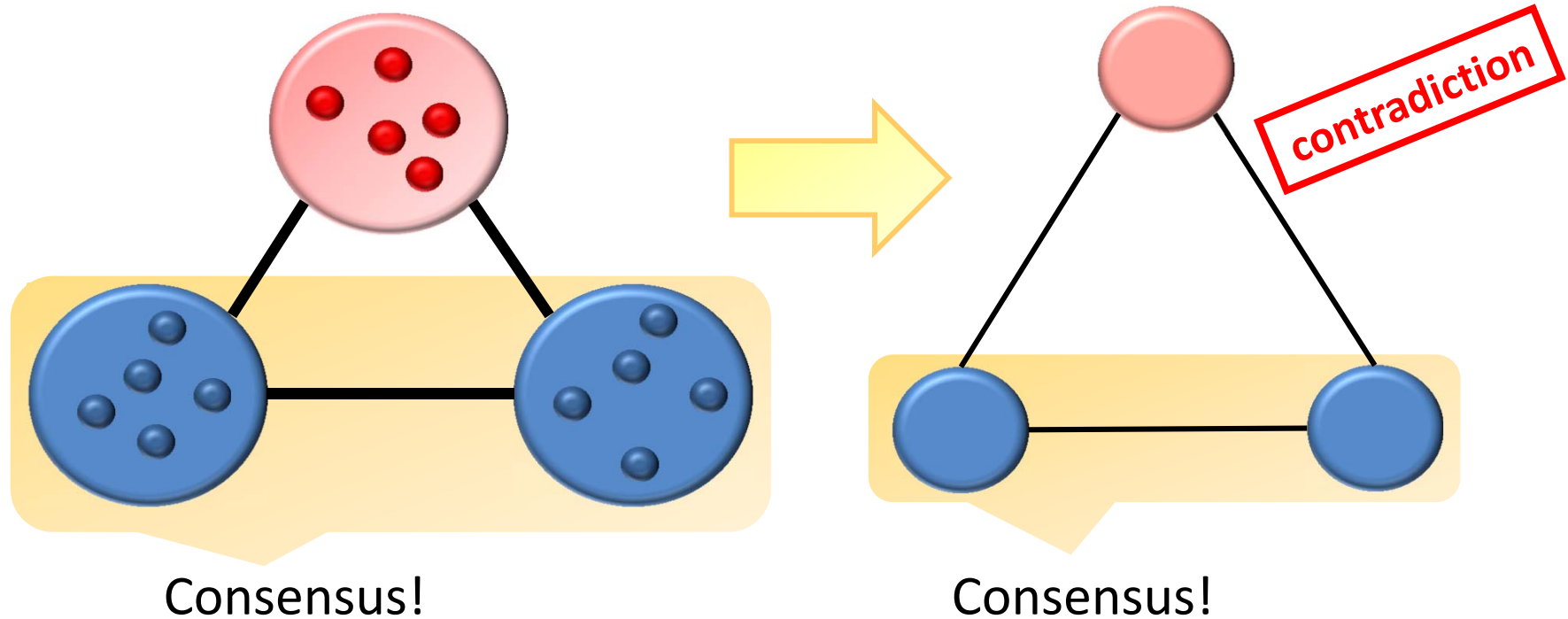


Intuition:

- Node **A** may also receive information from **C** about **B**'s messages to **C**
- Node **A** may receive conflicting information about **B** from **C** and about **C** from **B** (the same for **C**!)
- It is impossible for **A** and **C** to decide which information to base their decision on!

The General Case

- One of the 3 nodes is Byzantine \Rightarrow its $n/3$ simulated nodes may all behave like Byzantine nodes
- Since algorithm A tolerates $n/3$ Byzantine failures, it can still reach consensus
 \Rightarrow We solved the consensus problem for three processes!

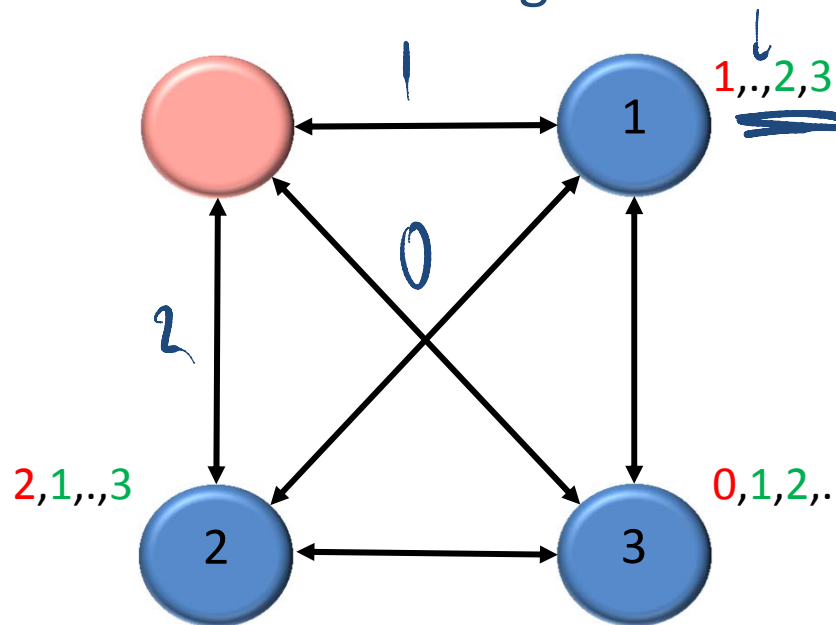


Cons. #6: Simple Byzantine Agreement Alg.

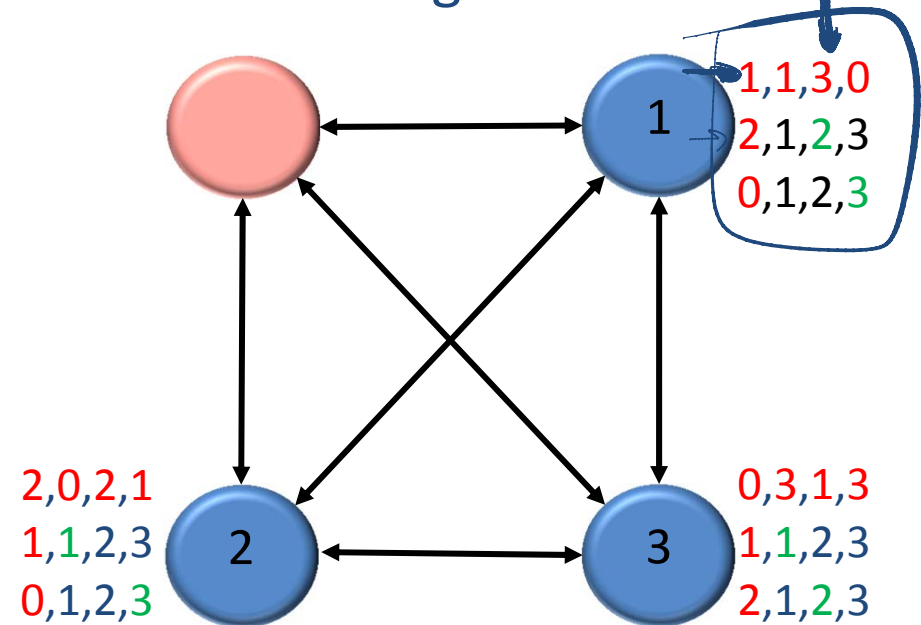


- Can the nodes reach consensus if $n > 3f$?
- A simpler question: What if $n = 4$ and $f = 1$?
- The answer is yes. It takes two rounds:

Round 1: Exchange all values



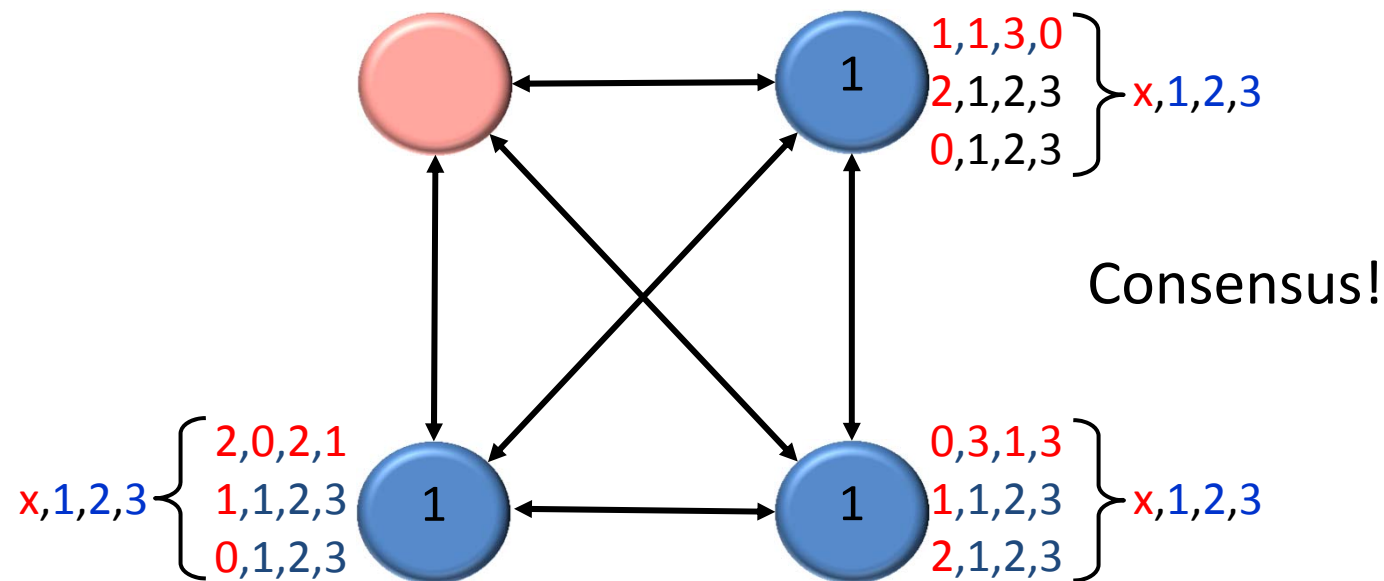
Round 2: Exchange received info



[matrix: one column for each original value, one row for each neighbor]

Simple Byzantine Agreement Algorithm

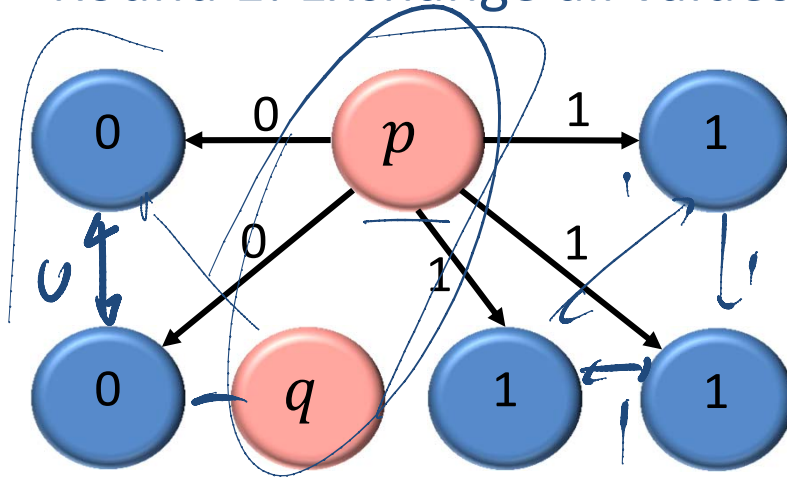
- After round 2, each node has received 12 values, 3 for each of the 4 input values (columns). If at least 2 of the 3 values of a column are equal, this value is accepted, otherwise it is discarded.
 - Values of honest nodes are accepted
 - The value of the Byzantine node is accepted iff it sends the same value to at least two nodes in the first round.
- Decide on most frequently accepted value!



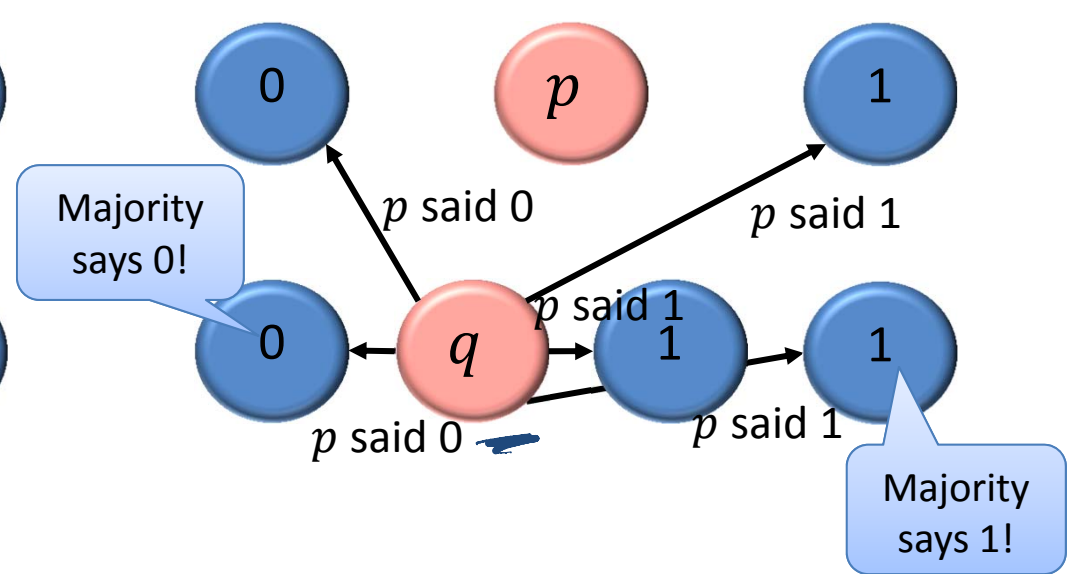
Simple Byzantine Agreement Algorithm

- Does the algorithm still work in general for any f and $n > 3f$?
- The answer is no. Try $f = 2$ and $n = 7$:

Round 1: Exchange all values



Round 2: Exchange received info



- The problem is that q can say different things about what p sent to q
 - What is the solution to this problem?

Simple Byzantine Agreement Algorithm



- The solution is simple: Again exchange all information!
- This way, the nodes can learn that q gave inconsistent information about p
- Hence, q can be excluded, and also p if it also gave inconsistent information (about q).
- If $f = 2$ and $n > 6$, consensus can be reached in 3 rounds!
- In fact, the following “algorithm” solves the problem for any f and any $n > 3f$:

Exchange all information for $f + 1$ rounds
Ignore all nodes that provided inconsistent information
Let all nodes decide based on the same input

Simple Byzantine Agreement Algorithm



The proposed algorithm has several advantages:

- + It works for **any f** and **$n > 3f$** , which is **optimal**
- + It only takes **$f + 1$ rounds**. This is even **optimal** for crash failures!
- + It **works for any input** and not just binary input

However, it has some considerable disadvantages:

- “Ignoring all nodes that provided inconsistent information”
is **not easy to formalize**
- The **size of the messages increases exponentially!**
This is a severe problem. It is therefore worth studying whether
it is possible to solve the problem with small(er) messages

Consensus #7: The Queen Algorithm

- The Queen algorithm is a simple Byzantine agreement algorithm that uses small messages
- The Queen algorithm solves consensus with n nodes and f failures where $f < n/4$ in $f + 1$ phases

A phase consists of 2 rounds

Idea:

- There is a different (a priori known) queen in each phase
- Since there are $f + 1$ phases, in one phase the queen is not Byzantine
- Make sure that in this round all nodes choose the same value and that in future rounds the nodes do not change their values anymore

The Queen Algorithm

$$f < \frac{n}{4}$$



In each phase $i \in \{1, \dots, f + 1\}$:

At the end of phase $f + 1$,
decide on own value

Round 1:

→ Broadcast own value

Also send own
value to oneself

→ Set own value to the value that was received most often

If own value appears $> n/2 + f$ times

support this value

else

do not support any value

If several values have the
same (highest)
frequency, choose any
value, e.g., the smallest

Round 2:

→ The queen broadcasts its value

If not supporting any value

set own value to the queen's value

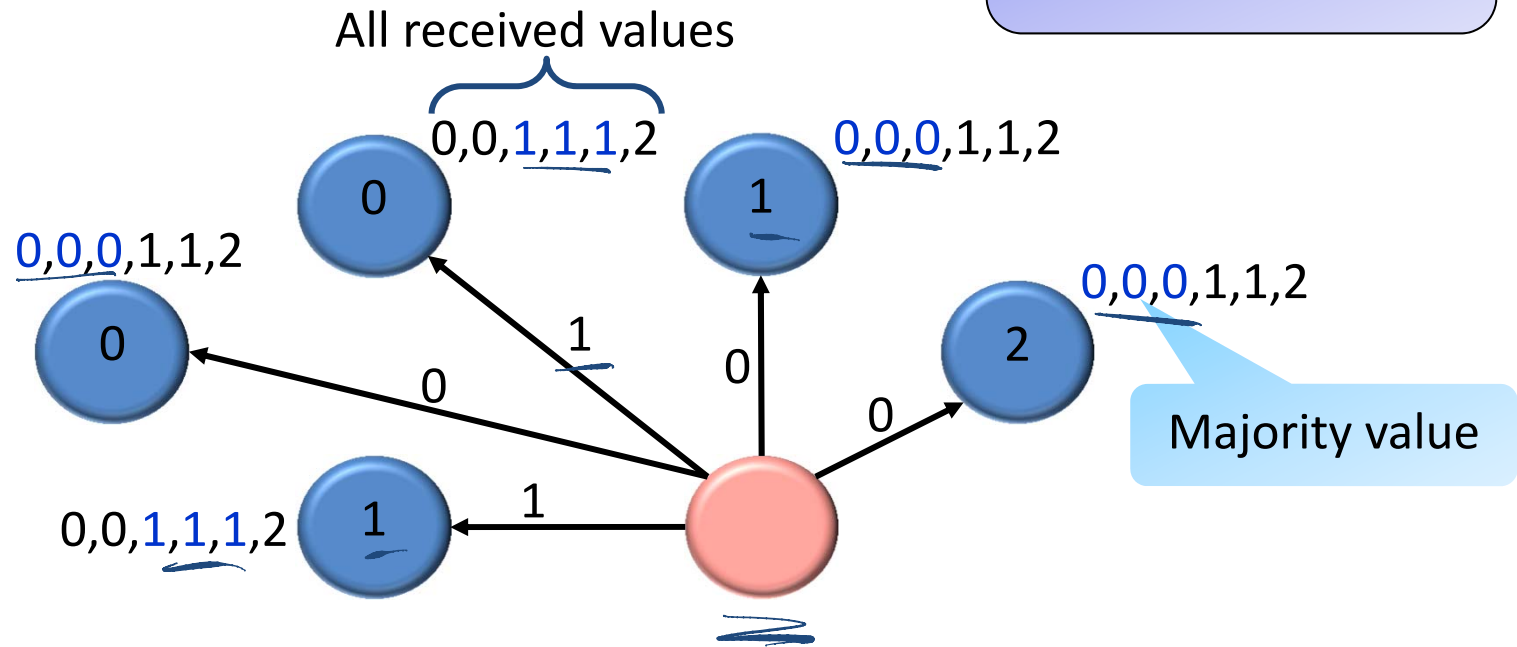
A handwritten diagram consisting of a circle divided into two segments by a vertical line. The larger segment on the left is labeled $n-f$ and the smaller segment on the right is labeled f .

The Queen Algorithm: Example

- Example: $n = 6, f = 1$
- Phase 1, round 1 (all broadcast):

$$\frac{n}{2} + f = 4$$

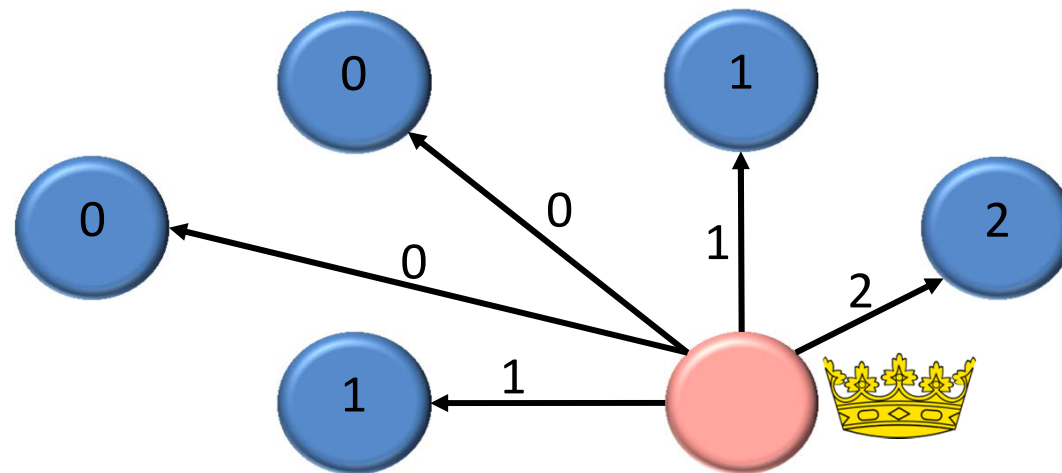
No node supports a value



The Queen Algorithm: Example

- Example: $n = 6, f = 1$
- Phase 1, round 2 (queen broadcasts):

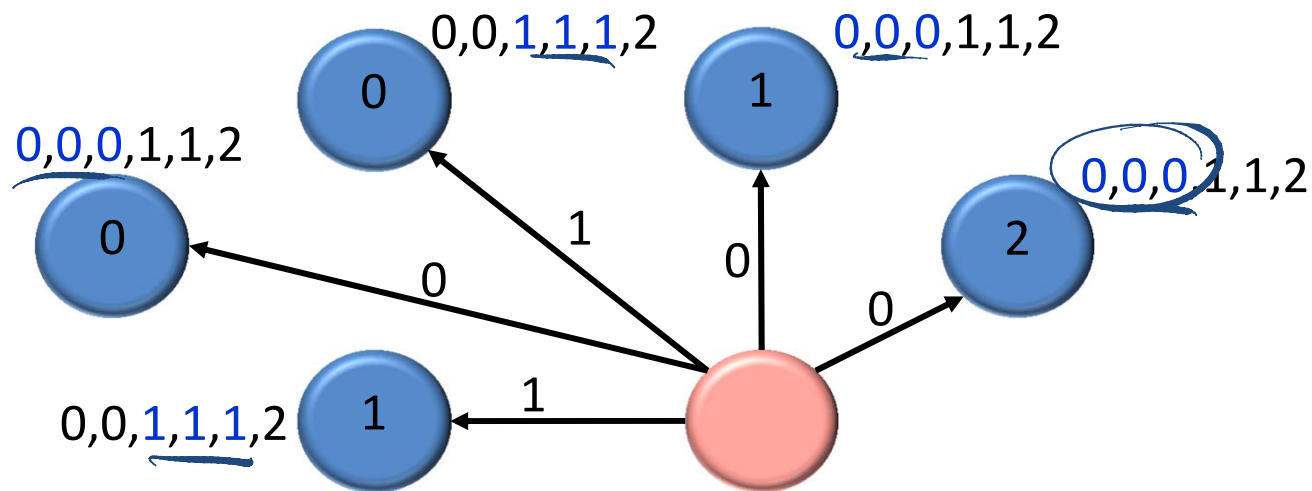
All nodes choose the queen's value



The Queen Algorithm: Example

- Example: $n = 6, f = 1$
- Phase 2, round 1 (all broadcast):

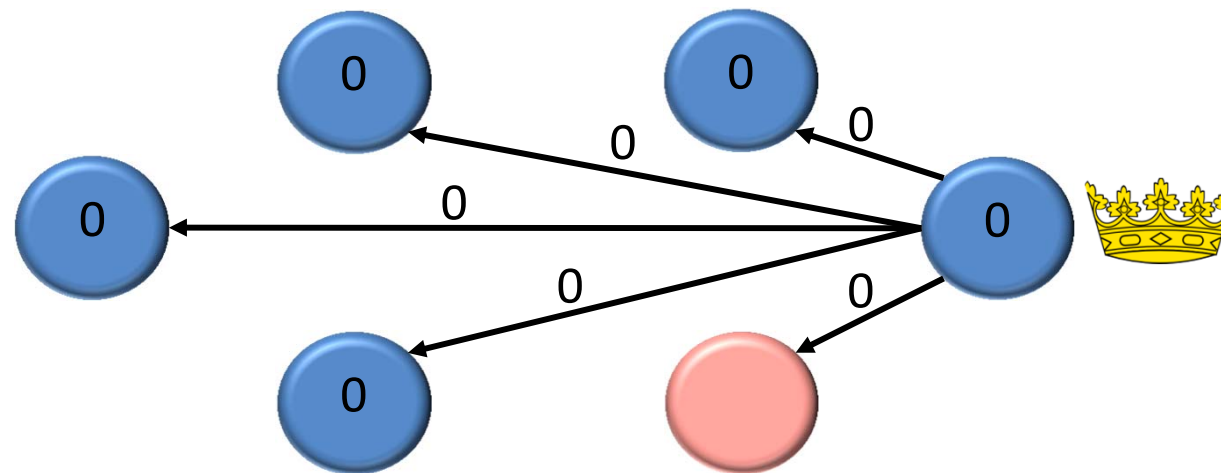
No node supports a value



The Queen Algorithm: Example

- Example: $n = 6, f = 1$
- Phase 2, round 2 (queen broadcasts):

All nodes choose the queen's value



Consensus!

The Queen Algorithm: Analysis



- After the phase where the queen is correct, all correct nodes have the same value
 - If all nodes change their values to the queen's value, obviously all values are the same
 - If some node does not change its value to the queen's value, it received a value $> \underline{n/2 + f}$ times \rightarrow All other correct nodes (including the queen) received this value $> n/2$ times and thus all correct nodes share this value
- In all future phases, no node changes its value
 - In the first round of such a phase, nodes receive their own value from at least $n - f > n/2$ nodes and thus do not change it
 - The nodes do not accept the queen's proposal if it differs from their own value in the second round because the nodes received their own value at least $n - f > n/2 + f$ times. Thus, all correct nodes support the same value

That's why we need
 $f < n/4!$

The Queen Algorithm: Summary

The Queen algorithm has several advantages:

- + The messages are small: nodes only exchange their current values
- + It works for any input and not just binary input

However, it also has some disadvantages:

- The algorithm requires $f + 1$ phases consisting of 2 rounds each ... this is twice as much as an optimal algorithm
- It only works with $f < n/4$ Byzantine nodes!
- Is it possible to get an algorithm that works with $f < n/3$ Byzantine nodes and uses **small messages?**

Consensus #8: The King Algorithm

- The King algorithm is an algorithm that tolerates $f < n/3$ Byzantine failures and uses small messages
- The King algorithm also takes $f + 1$ phases

A phase now consists of 3 rounds

Idea:

- The basic idea is the same as in the Queen algorithm
- There is a different (a priori known) king in each phase
- Since there are $f + 1$ phases, in one phase the king is not Byzantine
- The difference to the Queen algorithm is that the correct nodes only propose a value if many nodes have this value, and a value is only accepted if many nodes propose this value

The King Algorithm

In each phase $i \in \{1 \dots f + 1\}$:

At the end of phase $f + 1$,
decide on own value

Round 1:

Broadcast own value

Also send own value to oneself

Round 2:

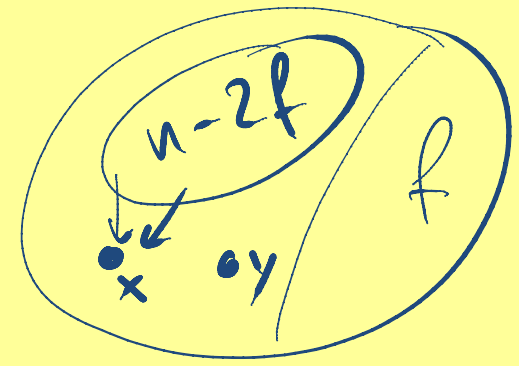
If some value x appears $\geq n - f$ times

Broadcast "Propose x "

$\geq n - f$

If some proposal received $> f$ times

Set own value to this proposal



$$2n - 4f \geq$$

Round 3:

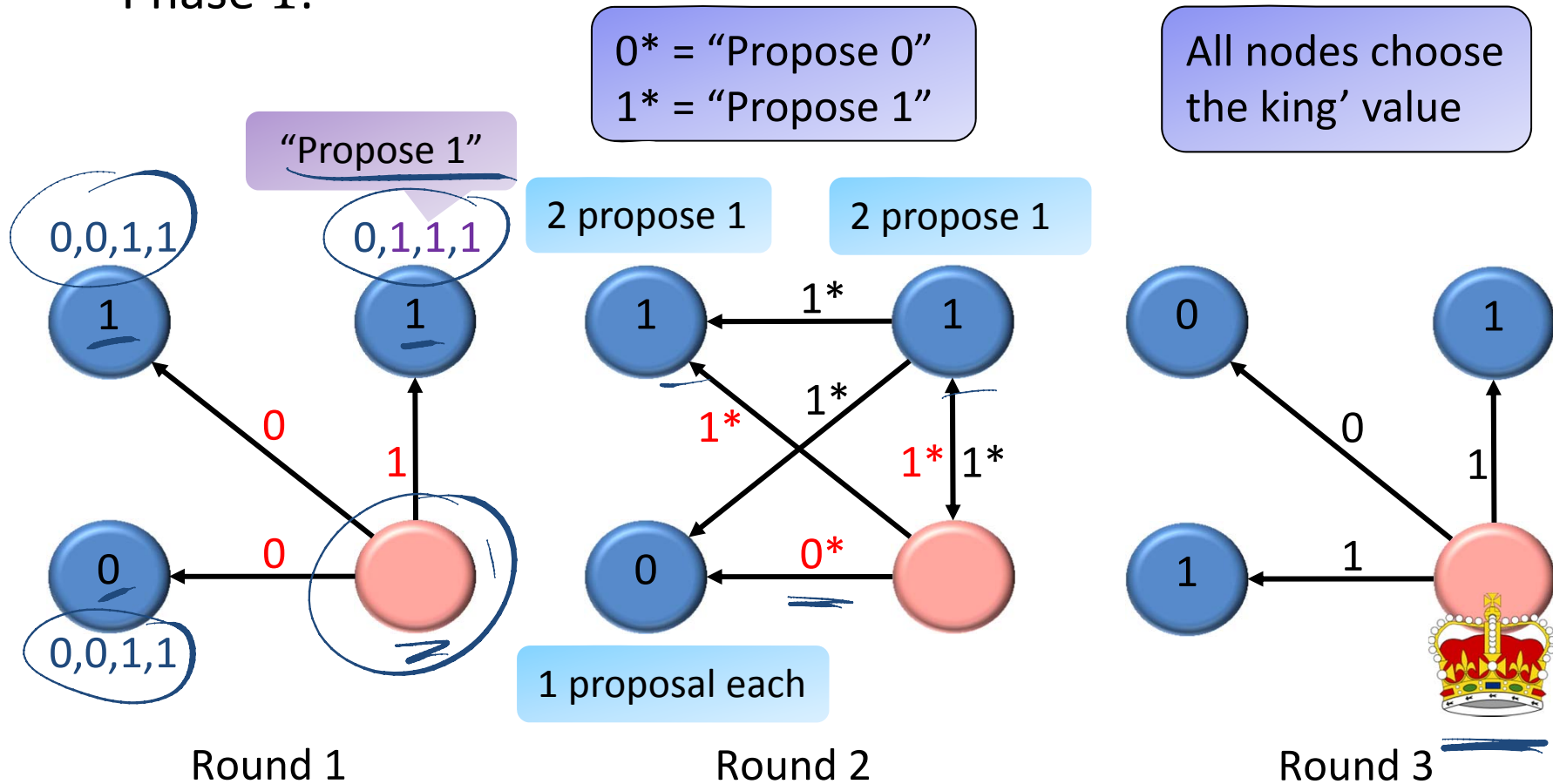
The king broadcasts its value

If own value received $< n - f$ proposals

Set own value to the king's value

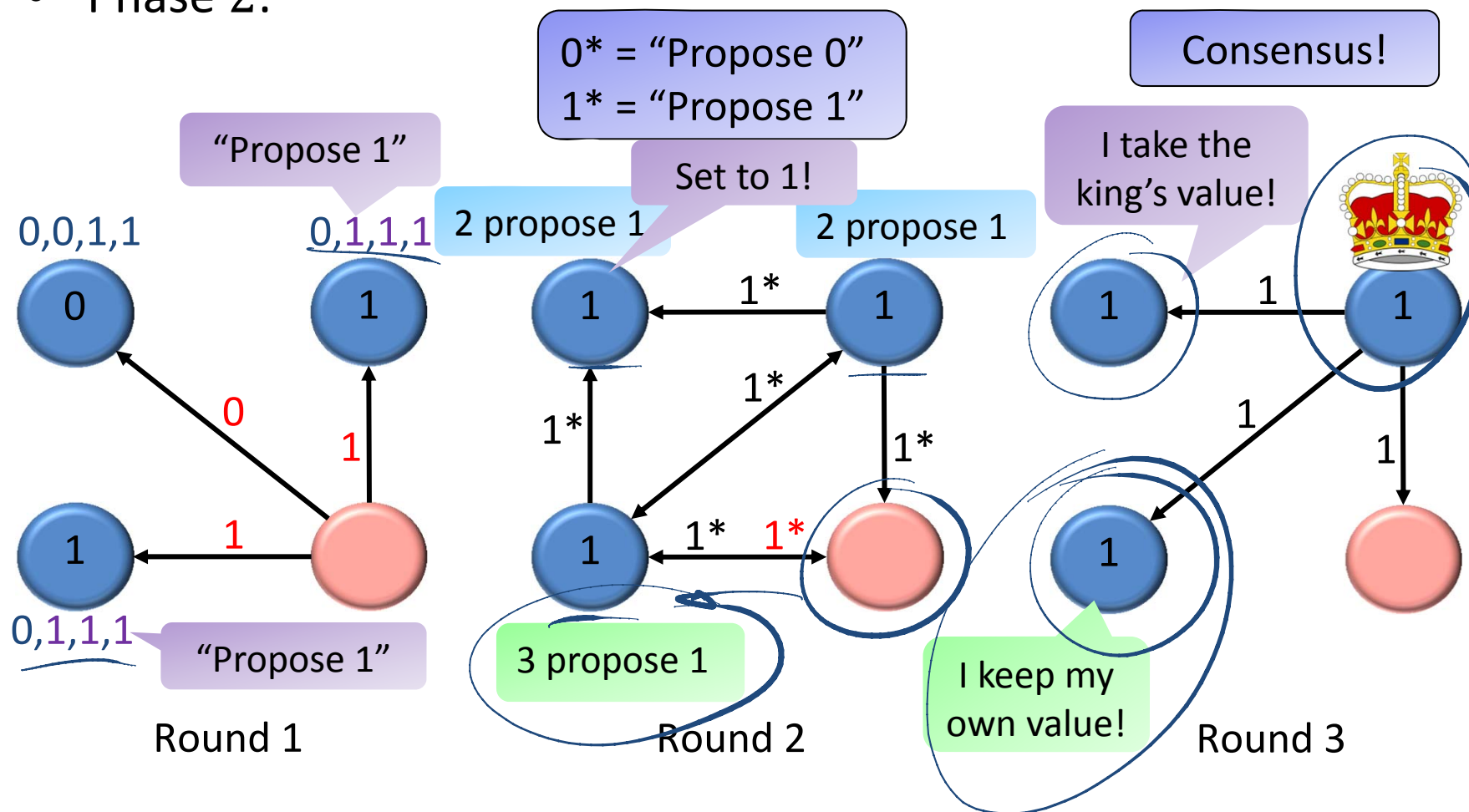
The King Algorithm: Example

- Example: $n = 4, f = 1$
- Phase 1:



The King Algorithm: Example

- Example: $n = 4, f = 1$
- Phase 2:



The King Algorithm: Analysis $2n - 3f > n$



- Observation: If some correct node proposes x , then no other correct node proposes $y \neq x$
 - Both nodes would have to receive $\geq n - f$ times the same value, i.e., both nodes received their value from $\geq n - 2f$ distinct correct nodes
 - In total, there must be $\geq \underline{2(n - 2f)} + \underline{f} > n$ nodes, a contradiction!
- The validity condition is satisfied
 - If all correct nodes start with the same value, all correct nodes receive this value $\geq n - f$ times and propose it
 - All correct nodes receive $\geq n - f$ times proposals, i.e., no correct node will ever change its value to the king's value

We used that
 $f < n/3!$

$$n - f/4$$

The King Algorithm: Analysis

- After the phase where the king is correct, all correct processes have the same value
 - If all processes change their values to the king's value, obviously all values are the same
 - If some process does not change its value to the king's value, it received a proposal $\geq n - f$ times $\rightarrow \geq n - 2f$ correct processes broadcast this proposal and all correct processes receive it $\geq n - 2f > f$ times
 - \rightarrow All correct processes set their value to the proposed value. Note that only one value can be proposed $> f$ times, which follows from the observation on the previous slide
- In all future phases, no process changes its value
 - This follows immediately from the fact that all correct processes have the same value after the phase where the king is correct and the validity condition

The King Algorithm: Summary

The King algorithm has several advantages:

- + It works for any f and $n > 3f$, which is optimal
- + The messages are small: processes only exchange their current values
- + It works for any input and not just binary input

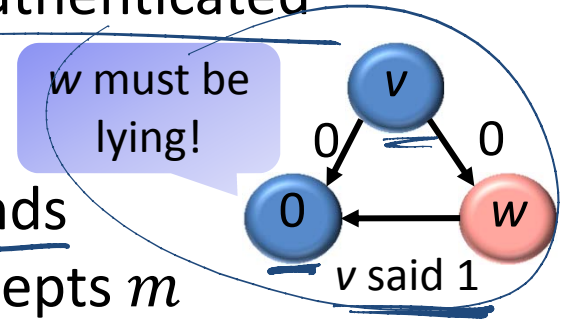
However, it also has a disadvantage:

- The algorithm requires $f + 1$ phases consisting of 3 rounds each
This is three times as much as an optimal algorithm
- Is it possible to get an algorithm that uses **small messages** and requires **fewer rounds of communication**?

Consensus #9: B. A. Using Authentication



- A simple way to reach consensus is to use authenticated messages
- Unforgeability condition: If a node never sends a message m , then no correct node ever accepts m
- Why is this condition helpful?
 - A Byzantine node cannot convince a correct node that some other correct node voted for a certain value if they did not!



Idea:

- There is a designated node v . The goal is to decide on v 's value
- For the sake of simplicity, we assume a binary input. The default value is 0 : if v cannot convince the nodes that v 's input is 1 , everybody chooses 0

Byzantine Agreement Using Authentication



If I am v and own input is 1

value := 1

broadcast "v has 1"

else

value := 0

In each round $r \in \{1, \dots, r + 1\}$:

If value = 0 and accepted r messages "v has 1" in total
including a message from v itself

value := 1

broadcast "v has 1" plus the r accepted messages that
caused the local value to be set to 1

After $r + 1$ rounds:

Decide value

In total $r + 1$ authenticated
"v has 1" messages

Byz. Agr. Using Authentication: Analysis



- **Assume that v is correct**

- v 's input is 1: All correct nodes accept v 's message in round 1 and set value to 1. No node ever changes its value back to 0
- v 's input is 0: v never sends a message " v has 1", thus no correct process ever sets its value to 1

- **Assume that v is Byzantine**

- v tries to convince some correct nodes that its input is 1
- Assume that a correct node u sets its value to 1 in a round $r < f + 1$:
Node u has accepted r messages including the message from v .
Therefore, all other correct nodes accept the same r messages plus u 's message and set their values to 1 as well in round $r + 1$
- Assume that a correct node u sets its value to 1 in round $f + 1$:
In this case, u accepted $f + 1$ messages. At least one of those is sent by a correct node, which must have set its value to 1 in an earlier round.
We are again in the previous case, i.e., all correct nodes decide 1!

Byz. Agr. Using Authentication: Summary



Using authenticated messages has several advantages:

- + It works for any number of Byzantine nodes!
- + It only takes $f + 1$ rounds, which is optimal
- + Small messages: nodes send at most $f + 1$ “short” messages to all other nodes in a single round

sub-exponential length

However, it also has some disadvantages:

- If v is Byzantine, the nodes may agree on a value that is not in the original input *violates validity*
- It only works for binary input
- The algorithm requires authenticated messages...

B. A. Using Authentication: Improvements



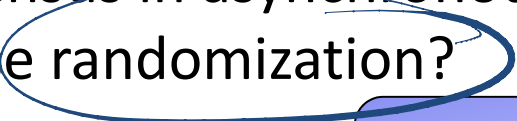
- Can we modify the alg. so that it satisfies the validity condition?
 - Yes! Run the algorithm in parallel for $2f + 1$ “masters” v . Either 0 or 1 occurs at least $f + 1$ times, i.e., at least one correct node had this value. Decide on this value!
 - Alas, this modified protocol only works if $f < n/2$
- Can we modify the algorithm so that it also works with an arbitrary input?
 - Yes! In fact, the algorithm does not have to be changed much
 - We won’t discuss this modification in class
- Can we get rid of the authentication?
 - Yes! Use *consistent broadcast*. This technique is not discussed either
 - This modified protocol works if $f < n/3$, which is optimal
 - However, each round is split into two
 - The total number of rounds is $2f + 2$

Consensus #10: A Randomized Algorithm



- So far we mainly tried to reach consensus in synchronous systems. The reason is that no deterministic algorithm can guarantee consensus in asynchronous systems even if only one process may crash
- Can one solve consensus in asynchronous systems if we allow our algorithms to use randomization?
- The answer is yes!
- The basic idea of the algorithm is to push the initial value. If other nodes do not follow, try to push one of the suggested values randomly
- For the sake of simplicity, we assume that the input is binary and at most $f < n/9$ nodes are Byzantine

Synchronous system: Communication proceeds in synchronous rounds



Asynchronous system: Messages are delayed indefinitely

Randomized Algorithm

$x := \text{own input}; r := 0$
Broadcast proposal(x, r)

$$f < n/9$$



In each round $r = 1, 2, \dots$:

Wait for $n - f$ proposals for $r-1$

If at least $n - 2f$ proposals have some value y

$x := y$; decide on y

else if at least $n - 4f$ proposals have some value y

$x := y$;

else

choose x randomly with $\Pr[x = 0] = \Pr[x = 1] = 1/2$

Broadcast proposal(x, r)

If decided on a value \rightarrow stop

Randomized Algorithm: Analysis

Validity condition (If all have the same input, all choose this value)

- If all correct nodes have the same initial value x , they will receive $n - 2f$ proposals containing x in the first round and they will decide on x

Agreement (if the nodes decide, they agree on the same value)

- Assume that some correct node decides on x . This node must have received x from $n - 3f$ correct nodes. Every other correct node must have received x at least $n - 4f$ times, i.e., all correct nodes set their local value to x , and propose and decide on x in the next round

The processes broadcast at the end of a phase to ensure that the processes that have already decided broadcast their value again!

Randomized Algorithm: Analysis

$$f < n/9$$



Termination (all correct processes eventually decide)

- If some nodes do not set their local value randomly, they set their local value to the same value.

Proof: Assume that some nodes set their value to 0 and some others to 1, i.e., there are $\geq n - 5f$ correct nodes proposing 0 and $\geq n - 5f$ correct processes proposing 1.

Then, in total there are $\geq 2(n - 5f) + f > n$ nodes. Contradiction!

That's why we need $f < n/9$!

- Thus, in the worst case all $n - f$ correct nodes need to choose the same bit randomly, which happens with probability $1/2^{n-f}$
- Hence, all correct processes eventually decide. The expected running time is smaller than 2^n
- The running time is awfully slow. Is there a clever way to speed up the algorithm?
- What about simply setting $x := 1$?! (Why doesn't it work?)

Can we do this faster?! Yes, with a Shared Coin



- A better idea is to replace

choose x randomly with $\Pr[x = 0] = \Pr[x = 1] = 1/2$

with a subroutine in which all the processes compute a so-called **shared** (a.k.a. common, “global”) coin



- A shared coin is a random binary variable that is 0 with constant probability and 1 with constant probability
- For the sake of simplicity, we assume that there are at most $f < n/3$ crash failures (no Byzantine failures!!!)

All correct nodes know the outcome of the shared coin toss after each execution of the subroutine

Shared Coin Algorithm



Code for process i :

Set local coin $c_i := 0$ with probability $1/n$, else $c_i := 1$
Broadcast c_i
Wait for exactly $n - f$ coins and collect all coins in the local coin set s_i
Broadcast s_i
Wait for exactly $n - f$ coin sets
If at least one coin is 0 among all coins in the coin sets
 return 0
else
 return 1

Assume the worst case:
Choose f so that $3f + 1 = n$!

Shared Coin: Analysis

- Termination (of the subroutine)
 - All correct nodes broadcast their coins. It follows that all correct nodes receive at least $n - f$ coins
 - All correct processes broadcast their coin sets. It follows that all correct processes receive at least $n - f$ coin sets and the subroutine terminates

- We will now show that at least 1/3 of all coins are **seen** by everybody

A coin is *seen* if it is in at least one received coin set

- More precisely: We will show that at least $f + 1$ coins are in at least $f + 1$ coin sets
 - Recall that $3f + 1 = n$ and therefore $f + 1 > n/3$
 - Since these coins are in at least $f + 1$ coin sets and all processes receive $n - f$ coin sets, all correct processes see these coins!



Shared Coin: Analysis

- Proof that at least $f + 1$ coins are in at least $f + 1$ coin sets
 - Draw the coin sets and the contained coins as a matrix
 - Example: $n = 7, f = 2$

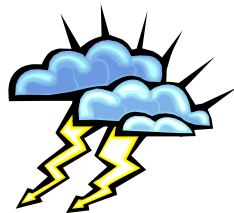
x means coin c_i is in set s_j

	s_1	s_3	s_5	s_6	s_7
c_1	X	X	X	X	X
c_2		X	X		
c_3	X	X	X	X	X
c_4		X	X		X
c_5	X			X	
c_6	X		X	X	X
c_7	X	X		X	X

Shared Coin: Analysis

- At least $f + 1$ rows (coins) have at least $f + 1$ x's (are in at least $f + 1$ coin sets)
 - First, there are exactly $(n - f)^2$ x's in this matrix
 - Assume that the statement is wrong: Then at most f rows may be full and contain $n - f$ x's. And all other rows (at most $n - f$) have at most f x's
 - Thus, in total we have at most $f(n - f) + (n - f)f = 2f(n - f)$ x's
 - But $2f(n - f) < (n - f)^2$ because $2f < n - f$

Here we use $3f < n$



	S_1	S_3	S_5	S_6	S_7
C_1	X	X	X	X	X
C_2		X	X		
C_3	X	X	X	X	X
C_4		X	X		X
C_5	X			X	
C_6	X		X	X	X
C_7	X	X		X	X

Shared Coin: Theorem

$$1 - \frac{1}{n}$$

Theorem

All processes decide 0 with constant probability, and all processes decide 1 with constant probability.

Proof:

- With probability $(1 - 1/n)^n \approx 1/e \approx 0.37$ all nodes choose 1. Thus, all correct nodes return 1
- There are at least $f + 1 \approx n/3$ coins seen by all correct nodes. The probability that at least one of these coins is set to 0 is at least

$$1 - \left(1 - \frac{1}{n}\right)^{\frac{n}{3}} \approx 1 - \left(1 - \frac{1}{e}\right)^{\frac{1}{3}} \approx 0.28$$

Back to Randomized Consensus

- If this shared coin subroutine is used, there is a constant probability that the processes agree on a value
- Some nodes may not want to perform the subroutine because they received the same value x at least $n - 4f$ times. However, there is also a constant probability that the result of the shared coin toss is x !
- Of course, all nodes must take part in the execution of the subroutine
- This randomized algorithm terminates in a constant number of rounds (in expectation)!

Randomized Algorithm: Summary

The randomized algorithm has several advantages:

- + It only takes a constant number of rounds in expectation
- + It can handle crash failures even if communication is asynchronous

However, it also has some disadvantages:

- It works only if there are $f < n/9$ crash failures. It doesn't work if there are Byzantine nodes
- It only works for binary input

There are similar algorithms for the shared memory model

Can it be improved?

- + There is a constant expected time algorithm that tolerates $f < \underline{n/2}$ crash failures
- What about Byzantine failures?

Byzantine and Asynchronous?

- Are there algorithms that can solve Byzantine agreement in an asynchronous environment (faster than simple individual coin flips)?
 - + Yes, there are.
 - However, the solution comes with a cost. The fraction of nodes that can be Byzantine decreases to 1/35715. Also, the expected message complexity of the state of the art algorithm is $O(n^3)$.
 - + Message sizes remain polynomial.
 - + The algorithm does not use a global coin, but tries to detect (biased) local coinflips from Byzantine processes.

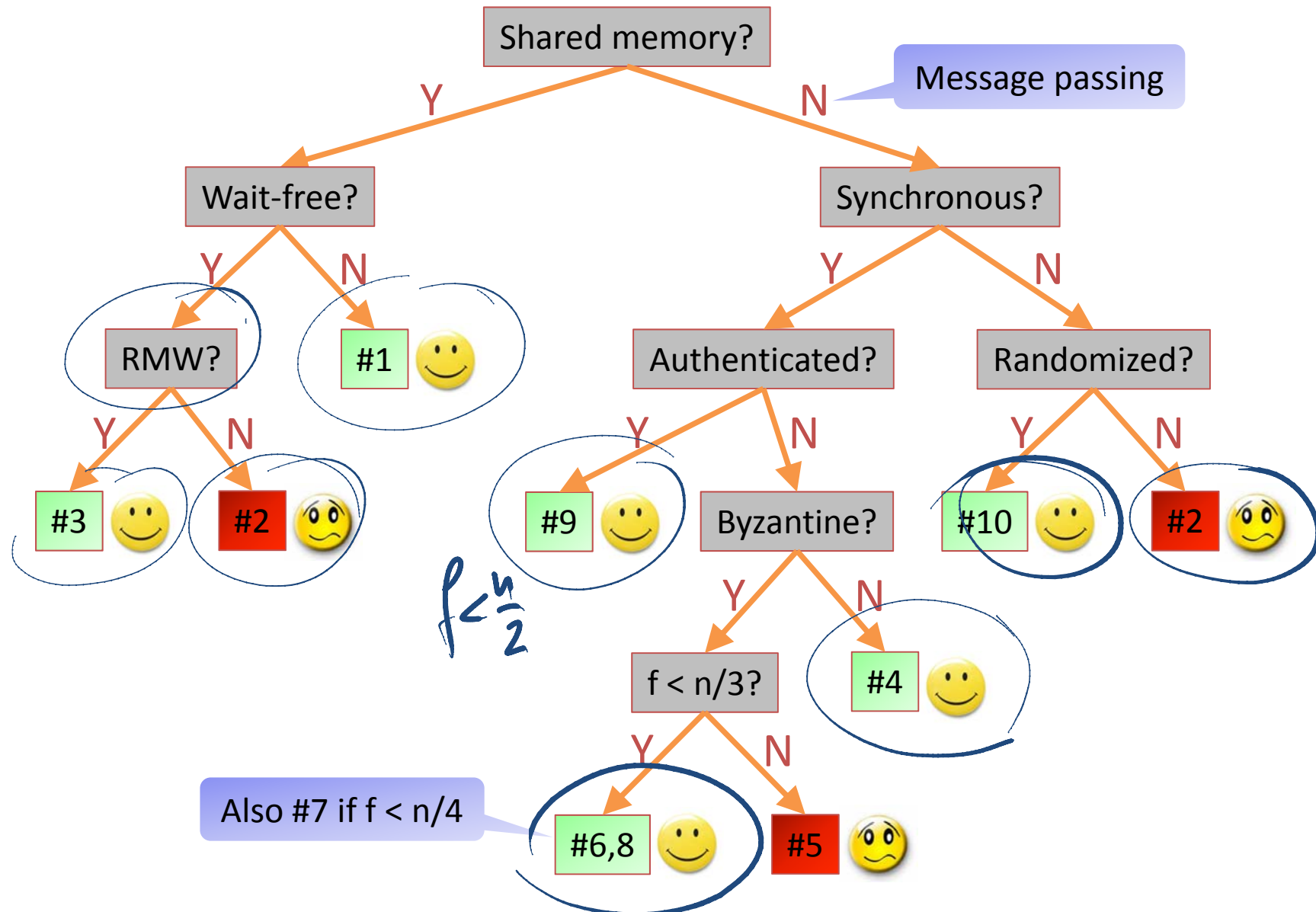
Message Complexity

- In all of the previously discussed message passing algorithms, each (good) node broadcasts. Message complexity is $\Omega(n^2)$.
- It was believed for long that the communication overhead of Byzantine Agreement is inherently large.
- Can the message complexity be improved?
 - + Yes!
 - + Byzantine Agreement can be solved in a synchronous environment with $\tilde{O}(n\sqrt{n})$ messages.
 - Requires polylogarithmic time.

Summary

- We have solved consensus in a variety of models
- In particular we have seen
 - algorithms
 - wrong algorithms
 - lower bounds
 - impossibility results
 - reductions
 - etc.

Consensus: Decision Tree



Credits

- The impossibility result (#2) is from Fischer, Lynch, Patterson, 1985
- The hierarchy (#3) is from Herlihy, 1991.
- The synchronous studies (#4) are from Dolev and Strong, 1983, and others.
- The Byzantine agreement problem (#5) and the simple algorithm (#6) are from Lamport, Shostak, Pease, 1980ff., and others
- The Queen algorithm (#7) and the King algorithm (#8) are from Berman, Garay, and Perry, 1989.
- The algorithm using authentication (#9) is due to Dolev and Strong, 1982.
- The first randomized algorithm (#10) is from Ben-Or, 1983.
- The concept of a shared coin was introduced by Bracha, 1984.
- Byzantine Agreement with fewer than n^2 messages is from King and Saia 2011.
- Byzantine Agreement in an asynchronous setting is from King and Saia 2013.