Computer Systems

Exercise 6

Agenda

- Bonus Task: First Deadline is next Thursday 01.11.18
- Feedback on Course
- Last Weeks Exercise (Scheduling / IO)
- Byzantine Agreement
- Shared Coin - Reliable Broadcast
- This weeks exercise

Byzantine nodes

- Node which has arbitrary behavior
- So it can:
  - Decide not to send messages
  - Sending different messages to different nodes
  - Sending wrong messages
  - Lie about input value
- If an algorithm works with f byzantine nodes, it is f-resilient

Different Validities

- Any-input validity:
  - The decision value must be input of any node
  - That includes byzantine nodes, might not make sense
- Correct-input validity:
  - The decision value must be input of a correct node
  - Difficult because byzantine node could behave like normal one just with different value
- All-same validity:
  - if all correct nodes start with the same value, the decision must be that value
- Median validity:
  - If input values are orderable, byzantine outliers can be prevented by agreeing on a value close to the median value of the correct nodes

Byzantine agreement in the synchronous model

- Assumption: nodes operate in synchronous rounds. In each round, each node may send a message to each other node, receive the message by other nodes and do some computation.
  - -> runtime is easy, since it is only the number of rounds

King Algorithm (synchronous byzantine agreement)

Idea:
If not all correct input nodes have the same value, decide on value of one correct input node. Ensure this by doing f+1 rounds, since there must be at least one correct input node.

Algorithm 11.14 King Algorithm (for $f < n/3$)

x = my input value
for phase $= 1$ to $f + 1$ **do**
  Round 1
  Broadcast `value`(x)
  Round 2
  if some `value`(y) received at least $n - f$ times **then**
    Broadcast `propose`(y)
  end if
  if some `propose`(z) received more than f times **then**
    $x = z$
  end if
  Round 3
  Let node v_i be the predefined king of this phase i
  The king v_i broadcasts its current value w
  if received strictly less than $n - f$ `propose`(y) **then**
    $x = w$
  end if
end for

Do until at least one correct input node

Send out own value

If some value received from all nodes but byzantine ones (or at least ((n – f)- f) correct ones), propose that value

If some value proposed by at least one correct node, set your value to that value

King of this phase broadcasts its value

If didn’t get propose from all nodes but byzantine ones (or at least ((n – f)- f) correct ones), set your value to value of king

King Algorithm (synchronous byzantine agreement)

Why f+1?

- Because there are f byzantine nodes, at least one of the kings will be a correct node

Algorithm 11.14 King Algorithm (for $f < n/3$)

x = my input value
for phase $= 1$ to $f + 1$ **do**
  Round 1
  Broadcast `value`(x)
  Round 2
  if some `value`(y) received at least $n - f$ times **then**
    Broadcast `propose`(y)
  end if
  if some `propose`(z) received more than f times **then**
    $x = z$
  end if
  Round 3
  Let node v_i be the predefined king of this phase i
  The king v_i broadcasts its current value w
  if received strictly less than $n - f$ `propose`(y) **then**
    $x = w$
  end if
end for

King Algorithm (synchronous byzantine agreement)

Algorithm 11.14 King Algorithm (for $f < n/3$)

x = my input value
for phase $= 1$ to $f + 1$ **do**

Round 1

Broadcast `value`(x)

Round 2

if some `value`(y) received at least $n - f$ times **then**
Broadcast `propose`(y)
end if
if some `propose`(z) received more than f times **then**
$x = z$
end if

Round 3

Let node v_i be the predefined king of this phase i
The king v_i broadcasts its current value w
if received strictly less than $n - f$ `propose`(y) **then**
$x = w$
end if
end for

King Algorithm (synchronous byzantine agreement)

Why n-f?

- Because if there are n-f correct nodes, so we can't wait for more. If we wait for less than f + 1 nodes, all the input values could be fake. Because 3f < n, n – f > f.
- Ensures only one proposal: If one node sees n-f values v, then every other node sees at least n-2f times v. Because n-(n-2f) = 2f < n-f, there can be no proposal for another value.
- All same validity ensured here!

Algorithm 11.14 King Algorithm (for $f < n/3$)

x = my input value
for phase = 1 to $f+1$ **do**
  Round 1
  Broadcast value(x)
  Round 2
  if some value(y) received at least $n - f$ times **then**
    Broadcast propose(y)
  end if
  if some propose(z) received more than f times **then**
    $x = z$
  end if
  Round 3
  Let node v_i be the predefined king of this phase i
  The king v_i broadcasts its current value w
  if received strictly less than $n - f$ propose(y) **then**
    $x = w$
  end if
end for

King Algorithm (synchronous byzantine agreement)

Why more than f?

- If we just waited for <= f propose messages, they all could be byzantine.

Algorithm 11.14 King Algorithm (for $f < n/3$)

```
x = my input value
for phase = 1 to f + 1 do
    Round 1
    Broadcast value(x)
    Round 2
    if some value(y) received at least n − f times then
        Broadcast propose(y)
    end if
    if some propose(z) received more than f times then
        x = z
    end if
    Round 3
    Let node v_i be the predefined king of this phase i
    The king v_i broadcasts its current value w
    if received strictly less than n − f propose(y) then
        x = w
    end if
end for
```

King Algorithm (synchronous byzantine agreement)

Why n-f propose messages?

- Similar as for n-f broadcast messages. We can wait for at most n-f ones because those are the correct nodes, and we have to wait for at least f+1 ones.

After a correct king, the correct nodes will not change their values anymore! Why?

- If all of them have less than n-f propose messages, all correct nodes will have the king value and then "all same validity" holds. If one does not adapt, this means that it got n-f propose messages. This means, every other message got at least n-f-f > f propose messages, so it adapted its value to the propose. So the king also adapted it's value and again all nodes have the same value.

Algorithm 11.14 King Algorithm (for $f < n/3$)

x = my input value
for phase = 1 **to** $f+1$ **do**
  Round 1
  Broadcast value(x)
  Round 2
  if some value(y) received at least $n-f$ times **then**
    Broadcast propose(y)
  end if
  if some propose(z) received more than f times **then**
    $x = z$
  end if
  Round 3
  Let node v_i be the predefined king of this phase i
  The king v_i broadcasts its current value w
  if received strictly less than $n-f$ propose(y) **then**
    $x = w$
  end if
end for

King Algorithm (synchronous byzantine agreement)

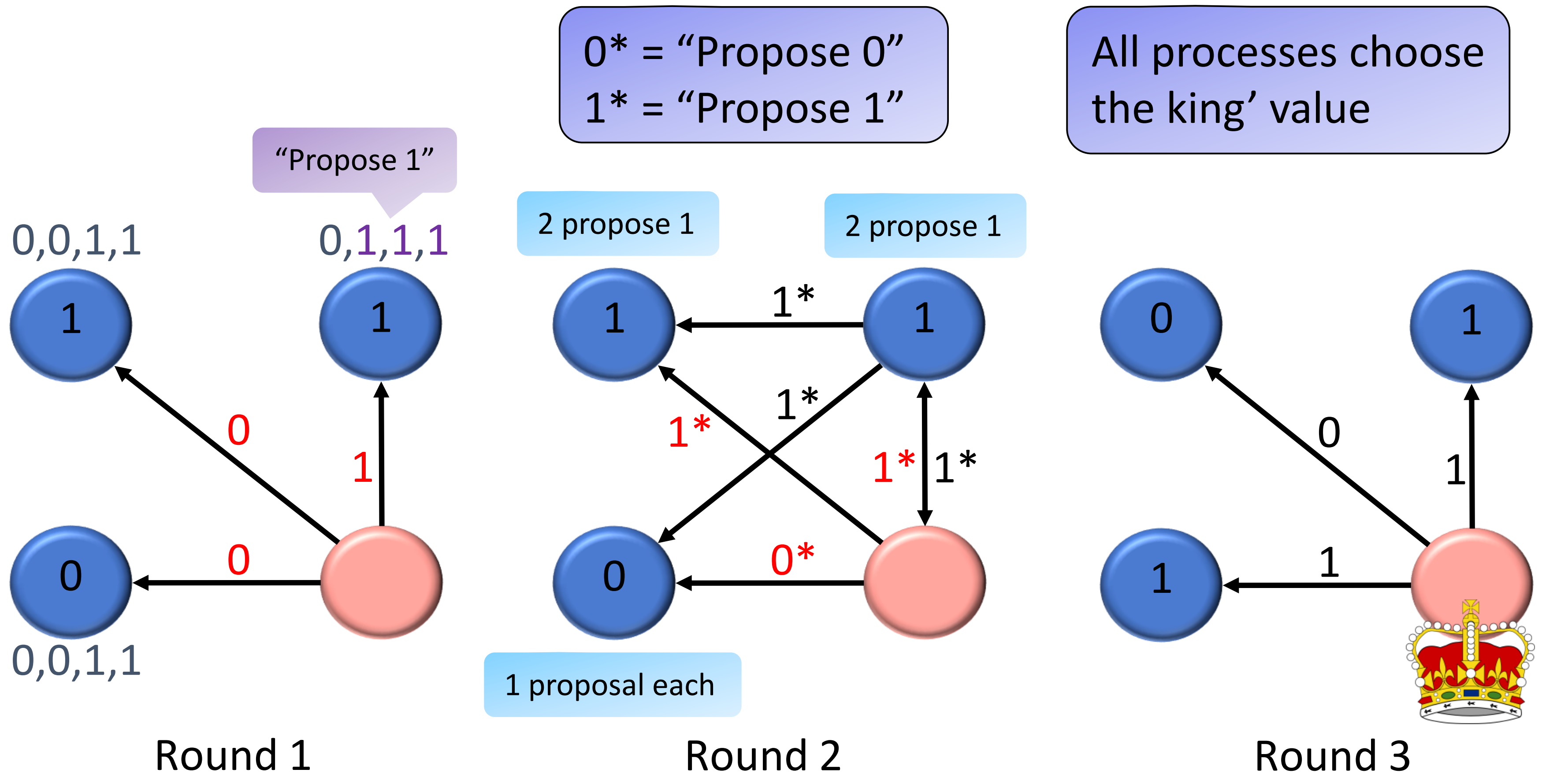
- Does it solve byzantine agreement?
  - Validity: All same validity!
  - Agreement: They agree at least after the first correct king.
  - Termination: After (f+1) phases

Algorithm 11.14 King Algorithm (for $f < n/3$)

x = my input value
for phase $= 1$ to $f+1$ **do**
  Round 1
  Broadcast `value`(x)
  Round 2
  if some `value`(y) received at least $n - f$ times **then**
    Broadcast `propose`(y)
  end if
  if some `propose`(z) received more than f times **then**
    $x = z$
  end if
  Round 3
  Let node v_i be the predefined king of this phase i
  The king v_i broadcasts its current value w
  if received strictly less than $n - f$ `propose`(y) **then**
    $x = w$
  end if
end for

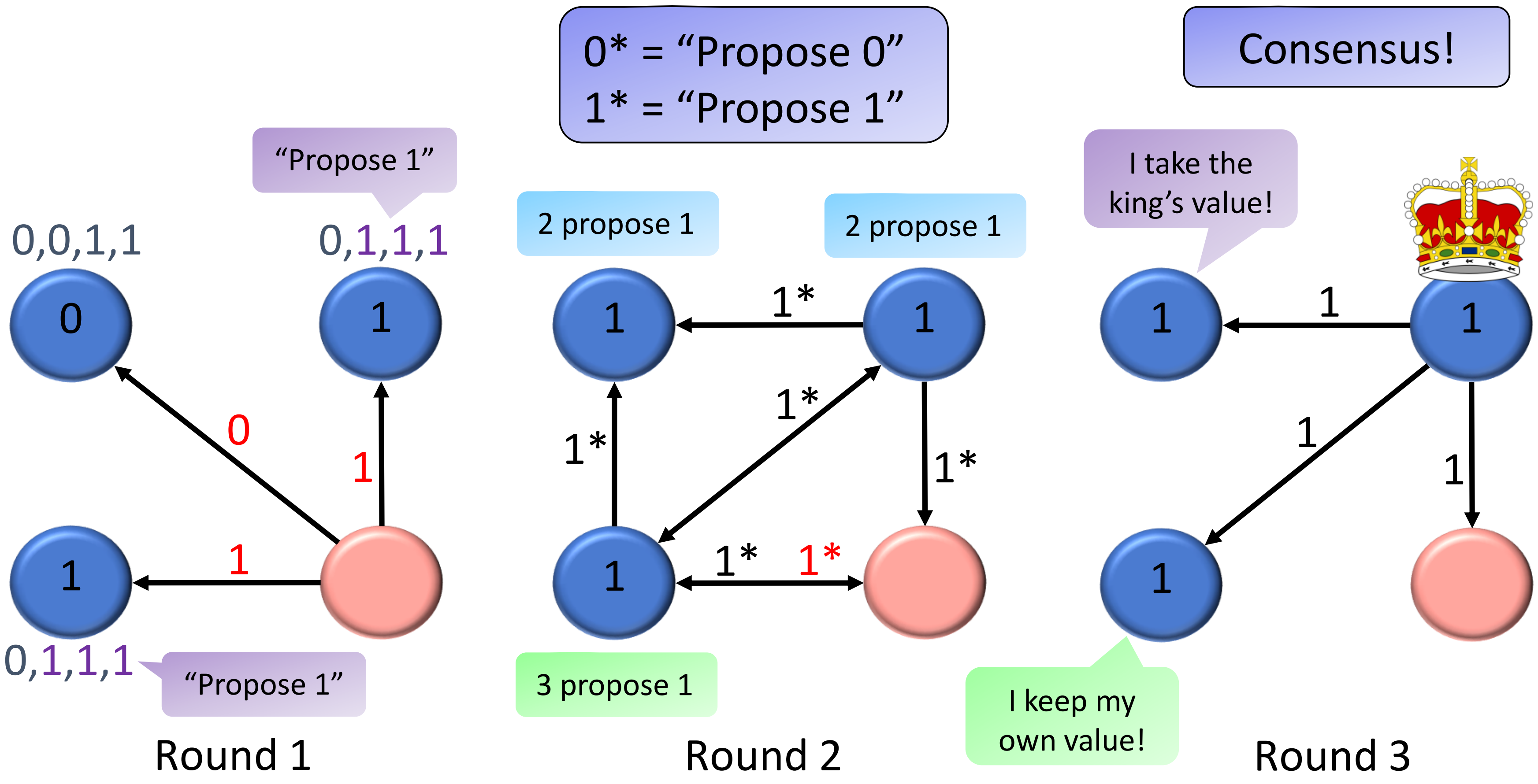
The King Algorithm: Example

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



The King Algorithm: Example

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



Asynchronous Byzantine Agreement

- Assumption: Messages do not need to arrive at the same time anymore. They have variable delays.

-> Also works, but is a lot more complicated.

-> Algorithm in script is proof of concept, so don't worry about it too much.

->Asynchronity changes messages you have to wait for, but not principle

- Problem: slow! (exponential runtime)

Algorithm 11.21 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

1: $x_i \in \{0, 1\}$ ◁ input bit
2: r = 1 ◁ round

Broadcast own value
Do until converged
Wait for enough messages
If big enough amount agrees, decide and terminate
If some agree, adapt your value but don't decide yet
If no popular value, decide randomly
Broadcast own value

16: **until** decided (see Line 8)
17: decision = x_i

Asynchronous Byzantine Agreement with oracle

- Now, if no popular value, all correct nodes will decide on same oracle value.
- Constant runtime
- Problem: oracle does not exist

Algorithm 11.21 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

```
1: x_i ∈ {0,1}            ◁ input bit
2: r = 1                  ◁ round
```

Broadcast own value

Do until converged

Wait for enough messages

If big enough amount agrees, decide and terminate

If some agree, adapt your value but don’t decide yet

If no popular value, ask oracle

Broadcast own value

```
16: until decided (see Line 8)
17: decision = x_i
```

Asynchronous Byzantine Agreement with random bitstring

- New idea: generate a random bitstring and take next value of bitstring instead of asking oracle
- Problem: byzantine nodes know “random” value and can adapt their behavior

Algorithm 11.21 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

$x_i \in \{0, 1\}$ ◁ input bit
r = 1 ◁ round

Broadcast own value

Do until converged

Wait for enough messages

If big enough amount agrees, decide and terminate

If some agree, adapt your value but don’t decide yet

If no popular value, ask look at bitstring

Broadcast own value

until decided (see Line 8)
decision = x_i

Asynchronous Byzantine Agreement with blackboard

- Back to the roots! – shared coin
- Implement it by writing values to a public blackboard, after seeing a certain amount of values nodes decide on coin value
- Constant probability that value is the same for all
- Similar to shared coin but works asynchronously
- Byzantine nodes don't know value of shared coin in advance

Algorithm 11.21 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

1: $x_i \in \{0, 1\}$ $\triangleleft$ input bit
2: r = 1 $\triangleleft$ round

Broadcast own value

Do until converged

Wait for enough messages

If big enough amount agrees, decide and terminate

If some agree, adapt your value but don't decide yet

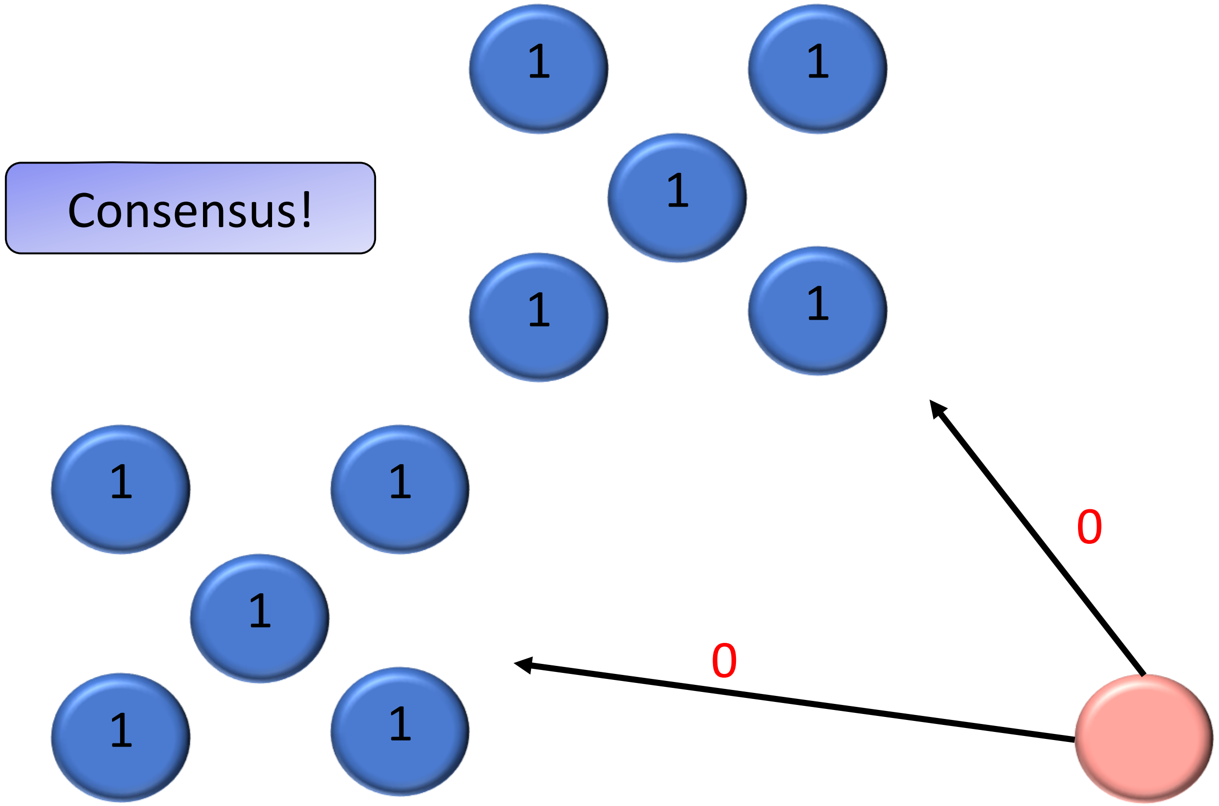
If no popular value, generate shared coin

Broadcast own value

16: **until** decided (see Line 8)
17: decision = x_i

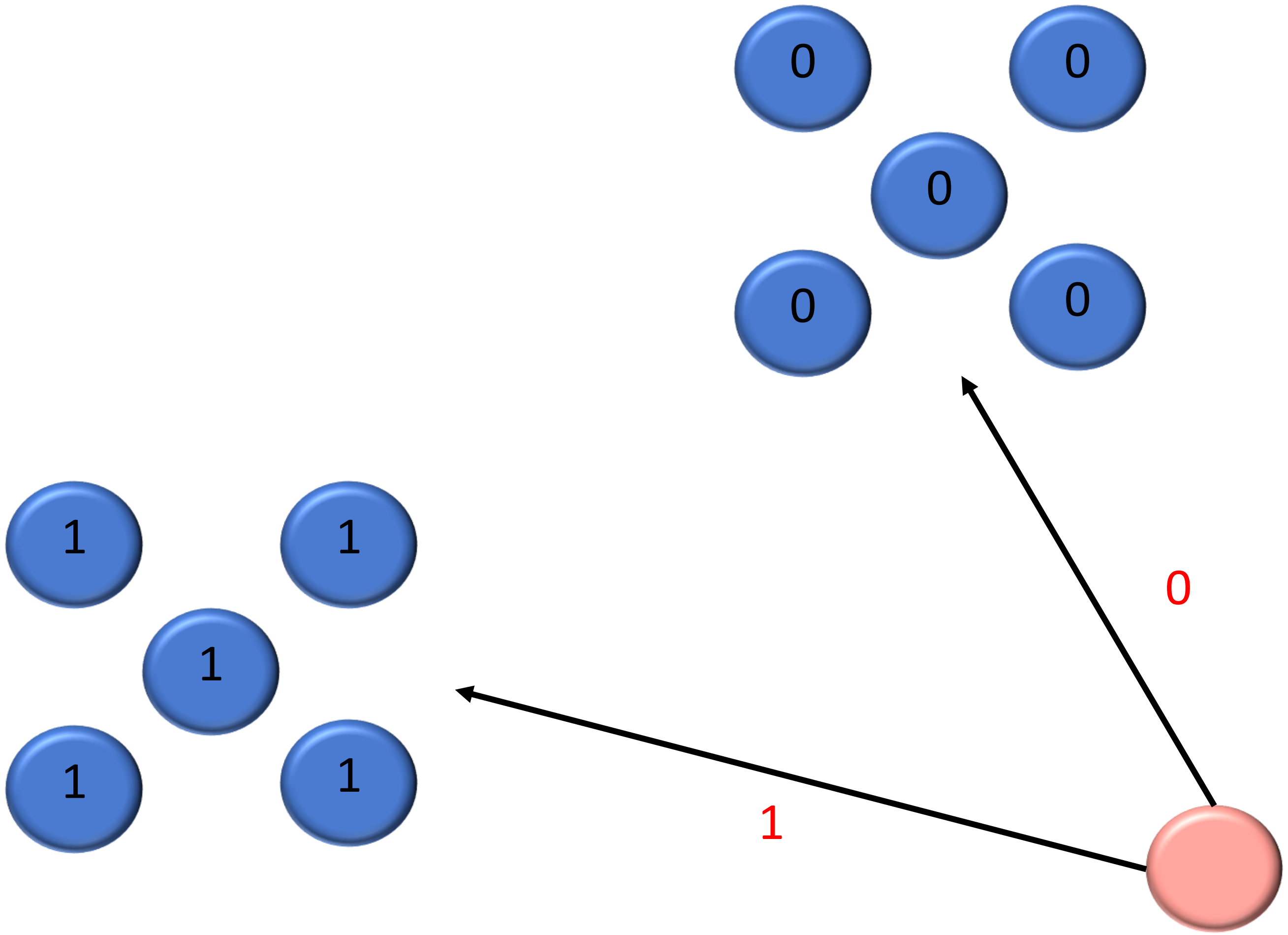
Ben-Or Algorithm: – All-Same Validity

- Example: n = 11, f=1
- Byzantine node has no power



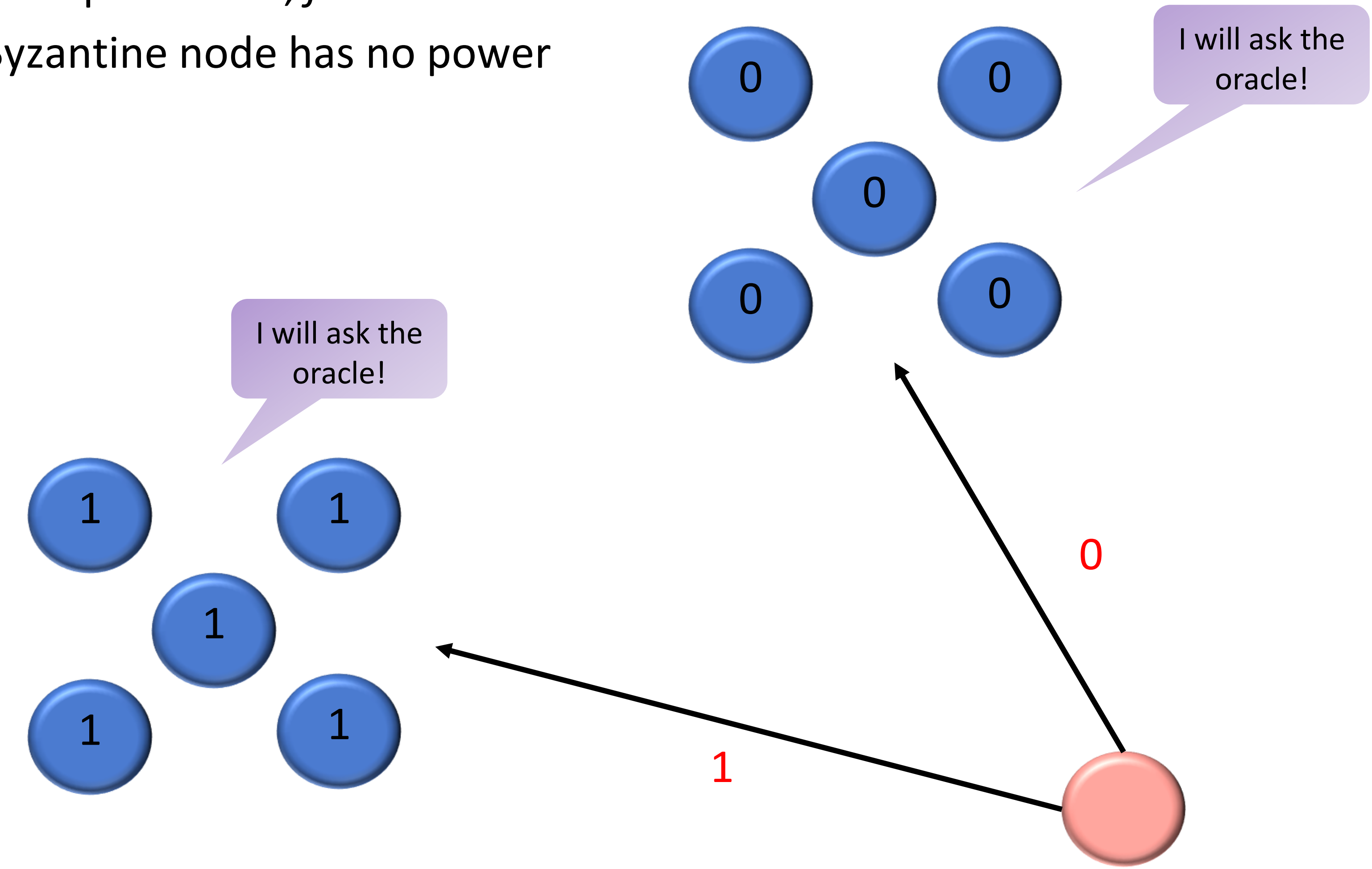
Ben-Or Algorithm: Example – Shared Coin

- Example: n = 11, f=1



Ben-Or Algorithm: Example – Shared Coin

- Example: n = 11, f=1
- Byzantine node has no power



Reliable Broadcast

- **Best effort broadcast**
  - Best effort broadcast ensures that a message that is sent from a correct node v to another correct node w will be received and accepted by w
- **Reliable broadcast**
  - Reliable broadcast ensures that the nodes eventually agree on all accepted messages. That is, if a correct node v considers message m as accepted, then every other node will eventually consider message m as accepted.
- **FIFO (reliable) broadcast**
  - The FIFO (reliable) broadcast defines an order in which the messages are accepted in the system. If a node u broadcasts message m1 before m2, then any node v will accept the message m1 first.
- **Atomic broadcast**
  - Atomic broadcast makes sure that all messages are always received in the same order. So for two random nodes u1 and u2 and two random messages m1 and m2, if u1 sees m1 first, u2 will also see m1 first.

Reliable Broadcast

Algorithm 4.15 Asynchronous Reliable Broadcast (code for node u)

Broadcast own value

If message received from node directly, broadcast it together with your own name

If you do not get message from node directly, but from a reasonable amount of others also broadcast with own name

If you get enough forwarded messages, accept message

Reliable Broadcast

Guarantees:

- If a node broadcasts a message reliably, all correct nodes will eventually accept that value
- If a correct node has not broadcast a message, it will not be accepted by any other correct node
- If a correct node accepts a message from a (byzantine) node, it will be eventually accepted by every correct node

Problem:

- Does not terminate!
- Does only tolerate <= n/5 byzantine nodes
  - This is better if we use the FIFO assumption