Randomized Consensus

Ellis Michael

Randomization Is Useful – Right?

For the most part, we don't know!

BPP (bounded-error probabilistic polynomial time) might equal P, it might not. There are many cases in which we think randomness might help, but few domains in which it has been proven to help.

It does for distributed systems though!

FLP, My Old Friend

Recall the FLP impossibility result.

Theorem: *In an asynchronous environment in which a single process can fail by crashing, there does not exist a protocol which solves binary consensus.*

Paxos doesn't save us. It doesn't guarantee liveness.

However, that result assumed a **deterministic** computation model.

That's So Random

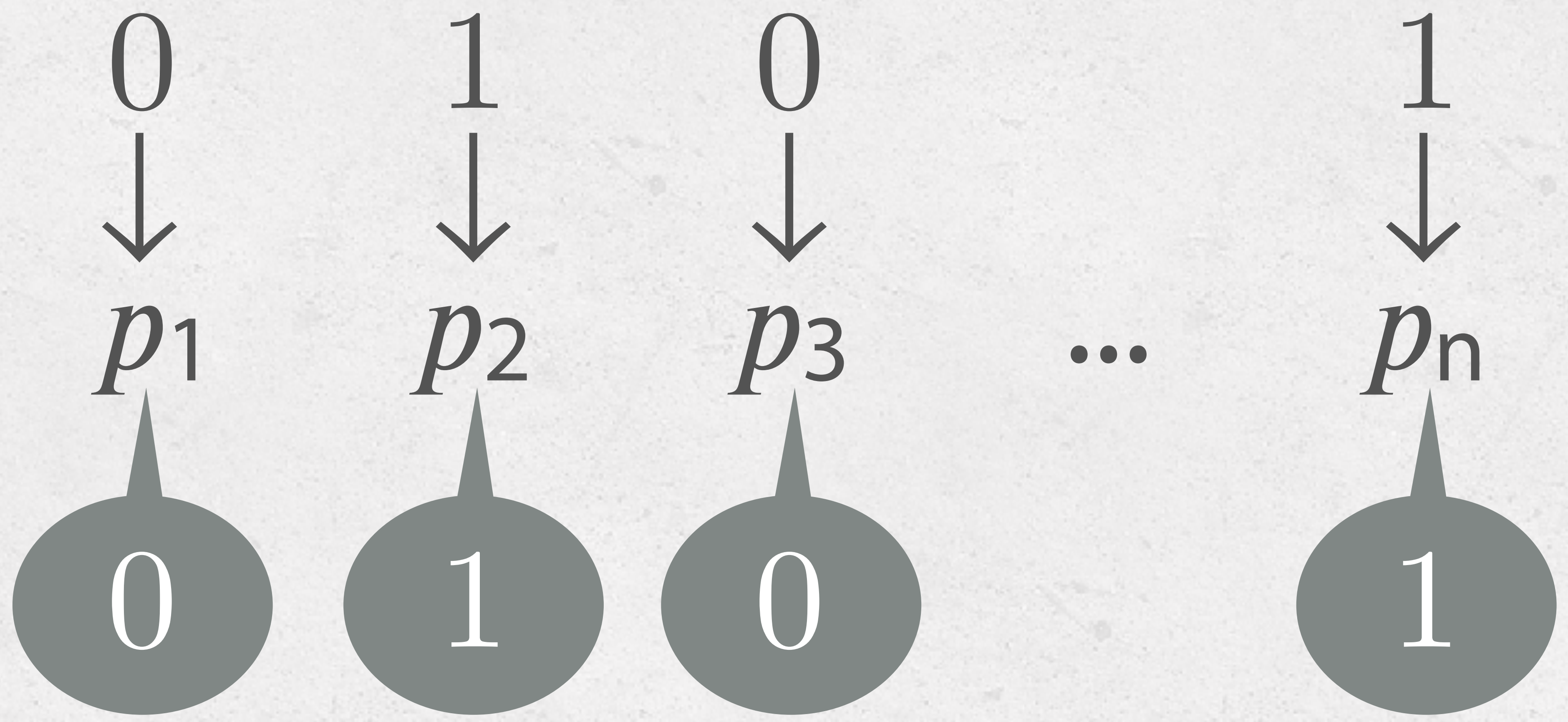
Ben-Or's algorithm uses randomization to ***guarantee consensus**** for crash failures when $f < n/2$.

A variant even works for Byzantine faults!

Intuition

- At first every process proposes their input value.
- After that, they propose random values.
- When enough processes propose the same value, the value is chosen.
- Eventually, that will happen!

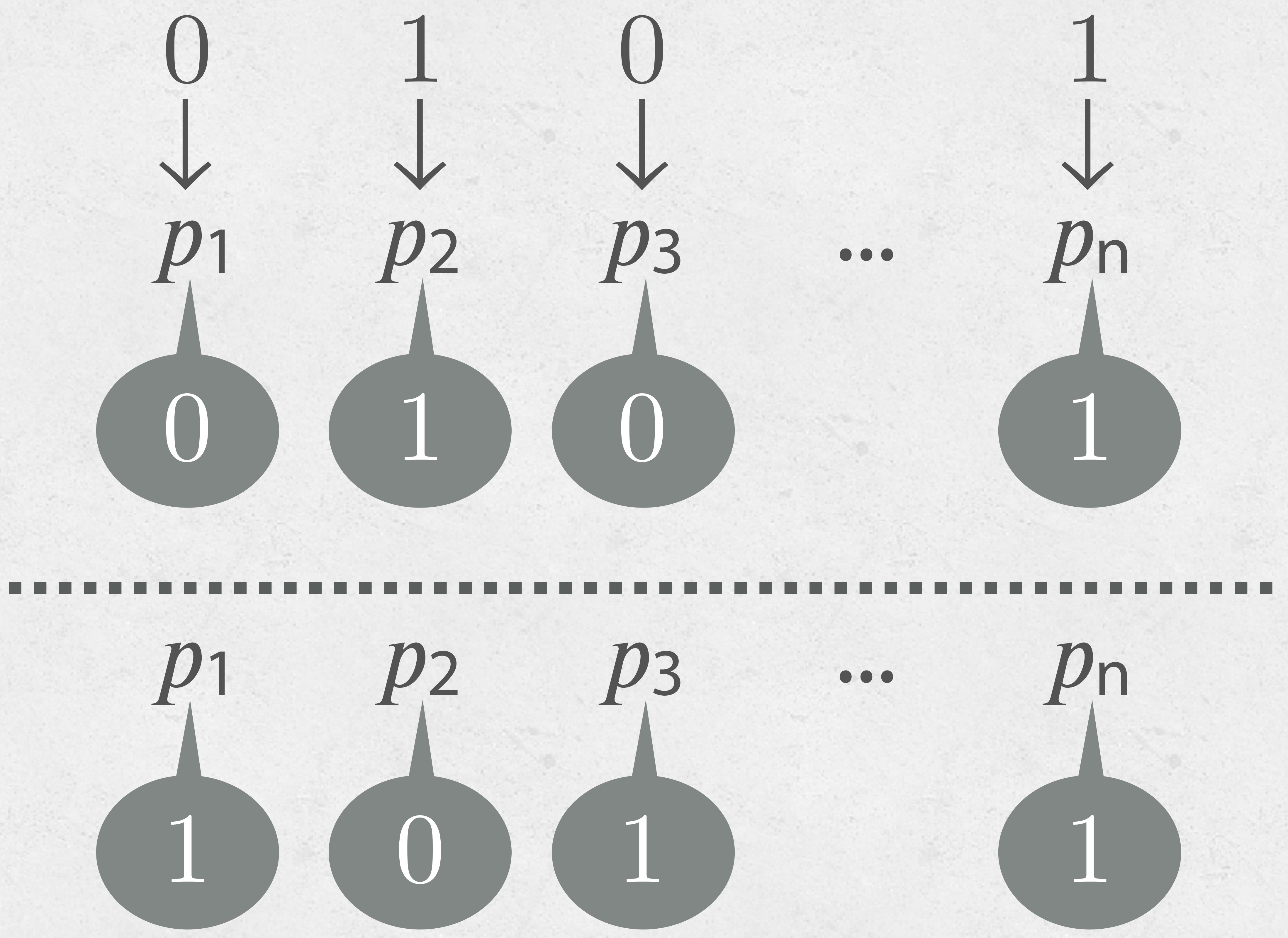
Intuition



- At first every process proposes their input value.
- After that, they propose random values.
- When enough processes propose the same value, the value is chosen.
- Eventually, that will happen!

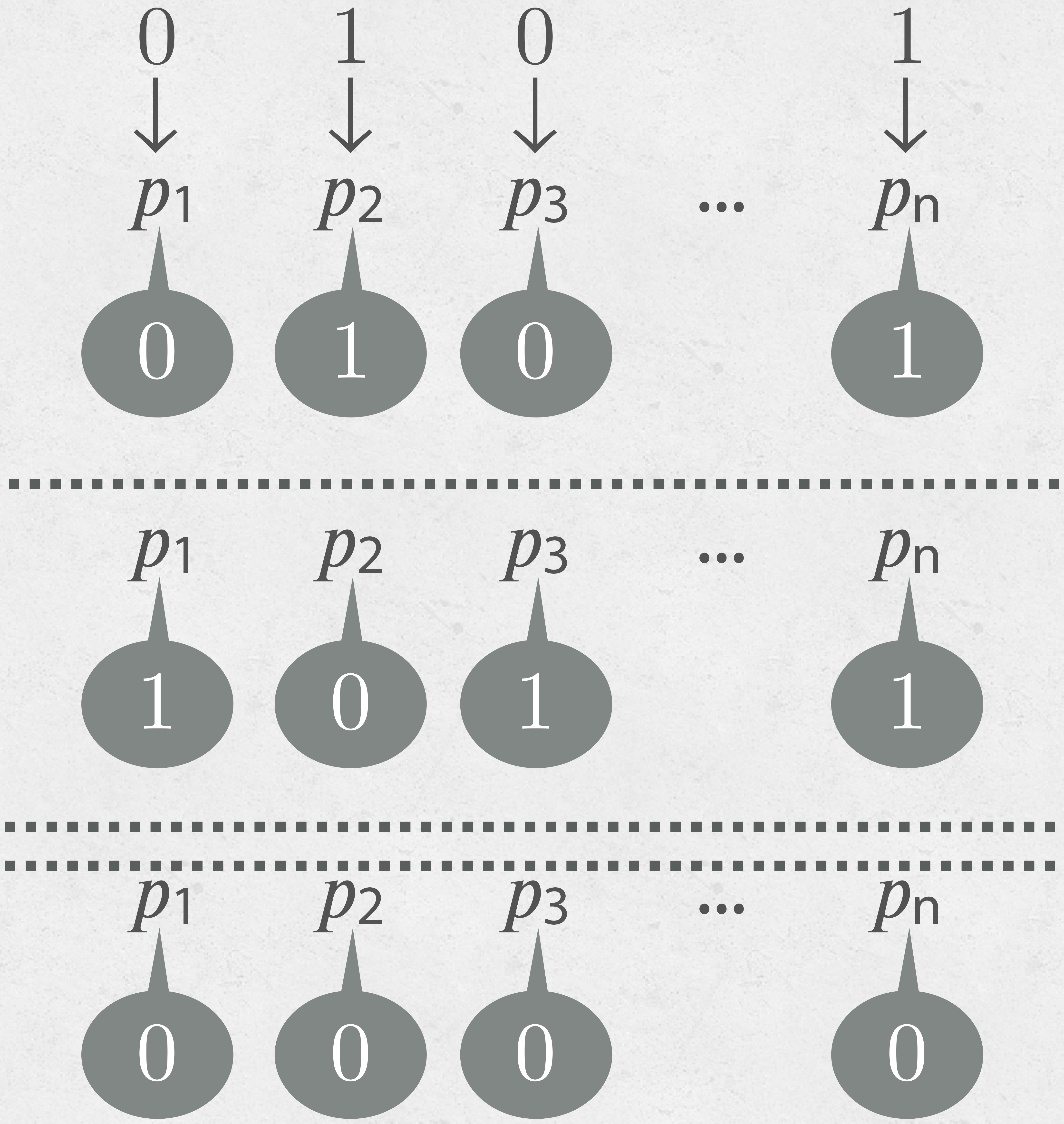
Intuition

- At first every process proposes their input value.
- After that, they propose random values.
- When enough processes propose the same value, the value is chosen.
- Eventually, that will happen!



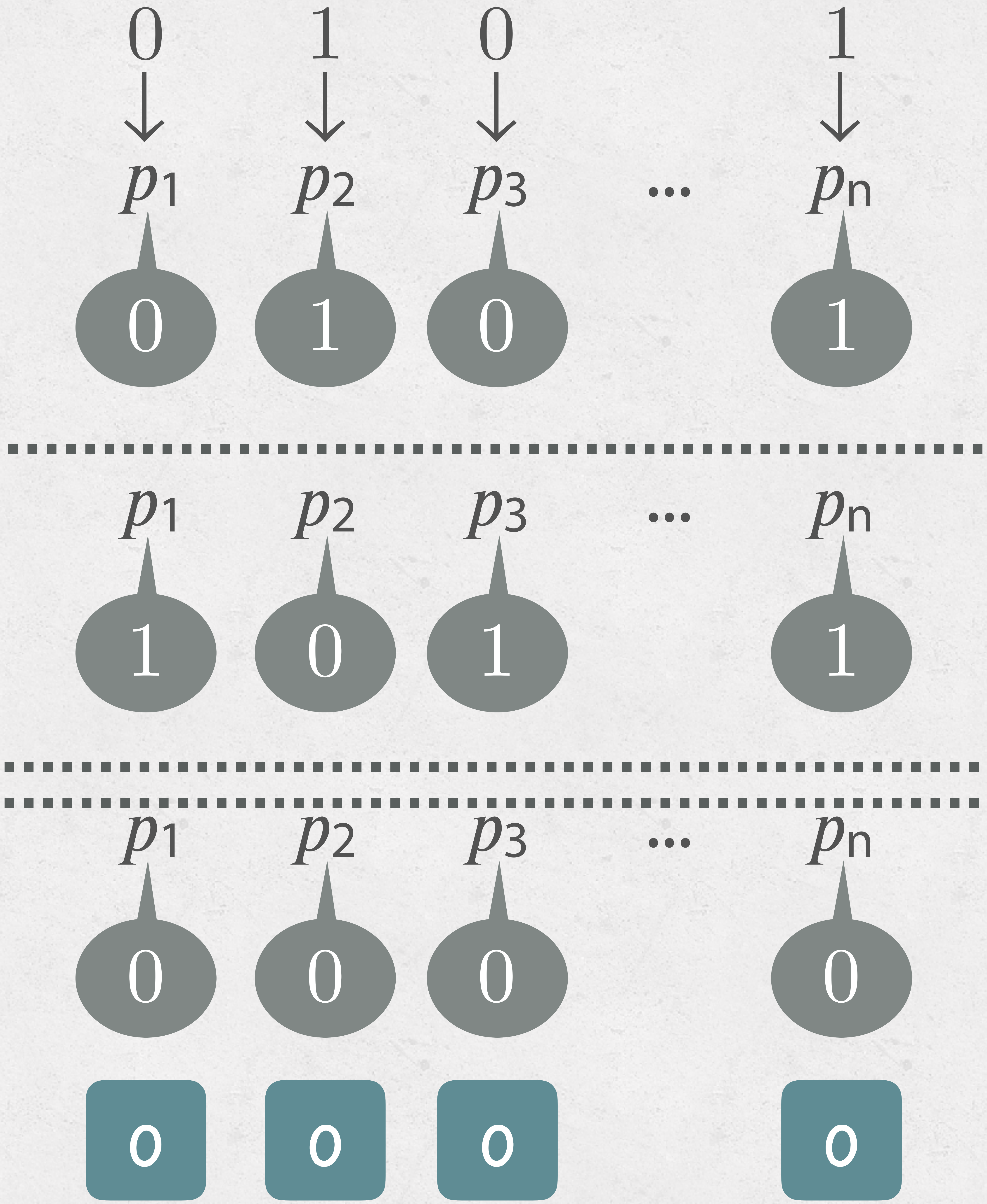
Intuition

- At first every process proposes their input value.
- After that, they propose random values.
- When enough processes propose the same value, the value is chosen.
- Eventually, that will happen!



Intuition

- At first every process proposes their input value.
- After that, they propose random values.
- When enough processes propose the same value, the value is chosen.
- Eventually, that will happen!



SETUP

- Again, we're considering binary consensus.
- Protocol proceeds in **asynchronous rounds**, where each round has two phases.
- For each phase, processes broadcast their input values and wait for $n - f$ messages from the other processes.
- Each message is tagged with the round and phase number. (And messages can be resent to deal with a lossy network. But once a message is sent, that value is locked in for that process for that phase/round.)

Ben-Or Algorithm

Processes send proposals for each phase and then block and wait for the requisite $n - f$ messages (including their own).

During the first phase, processes make a preliminary proposal.

If they receive matching responses from a majority in the first phase, they propose that value in the second phase. Otherwise, they propose $\perp$ (a special null value).

If they get enough non-$\perp$ responses from the second phase, they decide.

```
a←input
loop:
    send_phase1(a)
    A←receive_phase1()
    if (∃a′∈ A : |A_a′| > n/2):
        b←a′
    else:
        b←⊥

    send_phase2(b)
    B←receive_phase2()
    if (∃b′∈ B : b′≠⊥ ∧ |B_b′| > f):
        decide(b′)
    if (∃b′∈ B : b′≠⊥):
        a←b′
    else:
        a←choose_random({0,1})
```

Do We Have Consensus?

- **Agreement:** No two processes decide different values.
- **Integrity:** Every process decides at most one value, and if a process decides a value v, some process had v as its input.
- **Termination:** Every correct process eventually decides a value.

$a \leftarrow$ *input*

loop:

 send_phase1(a)

 $A \leftarrow$ *receive_phase1*()

 if ($\exists a' \in A : |A_{a'}| > n/2$):

  $b \leftarrow a'$

 else:

  $b \leftarrow \perp$

 send_phase2(b)

 $B \leftarrow$ *receive_phase2*()

 if ($\exists b' \in B : b' \neq \perp \wedge |B_{b'}| > f$):

  decide(b')

 if ($\exists b' \in B : b' \neq \perp$):

  $a \leftarrow b'$

 else:

  $a \leftarrow$ *choose_random*({0,1})

Integrity I

If both 0 and 1 are input values to processes, integrity is trivially satisfied.

Suppose all processes have the same input value.

$a \leftarrow$ *input*

loop:

  send_phase1(a)

  $A \leftarrow$ *receive_phase1*$()$

  if $(\exists a' \in A : |A_{a'}| > n/2)$:

    $b \leftarrow a'$

  else:

    $b \leftarrow \perp$

  send_phase2(b)

  $B \leftarrow$ *receive_phase2*$()$

  if $(\exists b' \in B : b' \neq \perp \wedge |B_{b'}| > f)$:

    decide(b')

  if $(\exists b' \in B : b' \neq \perp)$:

    $a \leftarrow b'$

  else:

    $a \leftarrow$ *choose_random({0,1})*

Integrity I

If both 0 and 1 are input values to processes, integrity is trivially satisfied.

Suppose all processes have the same input value.

- Then, they all send the same phase 1 value in round 1.

$a \leftarrow input$

loop:

  $send\_phase1(a)$

  $A \leftarrow receive\_phase1()$

  if $(\exists a' \in A : |A_{a'}| > n/2)$:

    $b \leftarrow a'$

  else:

    $b \leftarrow \perp$

  $send\_phase2(b)$

  $B \leftarrow receive\_phase2()$

  if $(\exists b' \in B : b' \neq \perp \wedge |B_{b'}| > f)$:

    decide(b')

  if $(\exists b' \in B : b' \neq \perp)$:

    $a \leftarrow b'$

  else:

    $a \leftarrow choose\_random(\{0,1\})$

INTEGRITY I

If both 0 and 1 are input values to processes, integrity is trivially satisfied.

Suppose all processes have the same input value.

- Then, they all send the same phase 1 value in round 1.
- So they all send that same value in phase 2.

$a \leftarrow input$

loop:

  $send\_phase1(a)$

  $A \leftarrow receive\_phase1()$

  if $(\exists a' \in A : |A_{a'}| > n/2)$:

    $b \leftarrow a'$

  else:

    $b \leftarrow \bot$

  $send\_phase2(b)$

  $B \leftarrow receive\_phase2()$

  if $(\exists b' \in B : b' \neq \bot \wedge |B_{b'}| > f)$:

    decide(b')

  if $(\exists b' \in B : b' \neq \bot)$:

    $a \leftarrow b'$

  else:

    $a \leftarrow choose\_random(\{0,1\})$

INTEGRITY I

If both 0 and 1 are input values to processes, integrity is trivially satisfied.

Suppose all processes have the same input value.

- Then, they all send the same phase 1 value in round 1.
- So they all send that same value in phase 2.
- So they all decide that value at the end of round 1.

$a \leftarrow$ *input*

loop:

    send_phase1(a)

    $A \leftarrow$ *receive_phase1*()

    if ($\exists a' \in A : |A_{a'}| > n/2$):

        $b \leftarrow a'$

    else:

        $b \leftarrow \perp$

    send_phase2(b)

    $B \leftarrow$ *receive_phase2*()

    if ($\exists b' \in B : b' \neq \perp \wedge |B_{b'}| > f$):

        decide(b')

    if ($\exists b' \in B : b' \neq \perp$):

        $a \leftarrow b'$

    else:

        $a \leftarrow$ *choose_random*({0,1})

FUN FACT

Lemma: *No two processes receive different non-$\perp$ phase 2 values in the same round.*

```
a←input
loop:
    send_phase1(a)
    A←receive_phase1()
    if (∃a′∈ A : |A_a′| > n/2):
        b←a′
    else:
        b←⊥

    send_phase2(b)
    B←receive_phase2()
    if (∃b′∈ B : b′≠⊥ ∧ |B_b′| > f):
        decide(b′)
    if (∃b′∈ B : b′≠⊥):
        a←b′
    else:
        a←choose_random({0,1})
```

Fun Fact

Lemma: *No two processes receive different non-⊥ phase 2 values in the same round.*

Suppose they did. That means that one process received 0s from a majority in phase 1 and another received 1s.

a←*input*

loop:

send_phase1(a)

A←*receive_phase1*()

if ($\exists a' \in A : |A_{a'}| > n/2$):

$b \leftarrow a'$

else:

$b \leftarrow \perp$

send_phase2(b)

B←*receive_phase2*()

if ($\exists b' \in B : b' \neq \perp \wedge |B_{b'}| > f$):

decide(b')

if ($\exists b' \in B : b' \neq \perp$):

$a \leftarrow b'$

else:

a←*choose_random({0,1})*

Fun Fact

Lemma: *No two processes receive different non-⊥ phase 2 values in the same round.*

Suppose they did. That means that one process received 0s from a majority in phase 1 and another received 1s.

But majorities intersect!

a←*input*

loop:

  send_phase1(a)

  A←*receive_phase1*()

  if ($\exists a' \in A : |A_{a'}| > n/2$):

    $b \leftarrow a'$

  else:

    $b \leftarrow \perp$

  send_phase2(b)

  B←*receive_phase2*()

  if ($\exists b' \in B : b' \neq \perp \wedge |B_{b'}| > f$):

    decide(b')

  if ($\exists b' \in B : b' \neq \perp$):

    $a \leftarrow b'$

  else:

    a←*choose_random*({0,1})

Agrement + Integrity II

$a \leftarrow input$

loop:

  $send\_phase1(a)$

  $A \leftarrow receive\_phase1()$

  if $(\exists a' \in A : |A_{a'}| > n/2)$:

    $b \leftarrow a'$

  else:

    $b \leftarrow \perp$

  $send\_phase2(b)$

  $B \leftarrow receive\_phase2()$

  if $(\exists b' \in B : b' \neq \perp \wedge |B_{b'}| > f)$:

    decide(b')

  if $(\exists b' \in B : b' \neq \perp)$:

    $a \leftarrow b'$

  else:

    $a \leftarrow choose\_random(\{0,1\})$

Agrement + Integrity II

Let round r be the first round any process decides a value, 0 w.l.o.g.

$a \leftarrow input$

loop:

  $send\_phase1(a)$

  $A \leftarrow receive\_phase1()$

  if $(\exists a' \in A : |A_{a'}| > n/2)$:

    $b \leftarrow a'$

  else:

    $b \leftarrow \bot$

  $send\_phase2(b)$

  $B \leftarrow receive\_phase2()$

  if $(\exists b' \in B : b' \neq \bot \wedge |B_{b'}| > f)$:

    decide(b')

  if $(\exists b' \in B : b' \neq \bot)$:

    $a \leftarrow b'$

  else:

    $a \leftarrow choose\_random(\{0,1\})$

Agrement + Integrity II

Let round r be the first round any process decides a value, 0 w.l.o.g.

If a process decided a value, it must have received $> f$ 0s in phase 2.

$a \leftarrow$ *input*

loop:

  send_phase1(a)

  $A \leftarrow$ *receive_phase1*()

  if ($\exists a' \in A : |A_{a'}| > n/2$):

    $b \leftarrow a'$

  else:

    $b \leftarrow \bot$

  send_phase2(b)

  $B \leftarrow$ *receive_phase2*()

  if ($\exists b' \in B : b' \neq \bot \wedge |B_{b'}| > f$):

    decide(b')

  if ($\exists b' \in B : b' \neq \bot$):

    $a \leftarrow b'$

  else:

    $a \leftarrow$ *choose_random({0,1})*

AGREMENT + INTEGRITY II

Let round r be the first round any process decides a value, 0 w.l.o.g.

If a process decided a value, it must have received $> f$ 0s in phase 2.

Which means that every process received at least one 0 because they all wait for $n - f$ messages. No process received a 1 by the previous lemma.

$a \leftarrow$ *input*

loop:

    send_phase1(a)

    $A \leftarrow$ *receive_phase1*()

    if $(\exists a' \in A : |A_{a'}| > n/2)$:

        $b \leftarrow a'$

    else:

        $b \leftarrow \perp$

    send_phase2(b)

    $B \leftarrow$ *receive_phase2*()

    if $(\exists b' \in B : b' \neq \perp \wedge |B_{b'}| > f)$:

        decide(b')

    if $(\exists b' \in B : b' \neq \perp)$:

        $a \leftarrow b'$

    else:

        $a \leftarrow$ *choose_random({0,1})*

Agrement + Integrity II

Let round r be the first round any process decides a value, 0 w.l.o.g.

If a process decided a value, it must have received $> f$ 0s in phase 2.

Which means that every process received at least one 0 because they all wait for $n - f$ messages. No process received a 1 by the previous lemma.

Therefore, on round $r + 1$ (and all subsequent rounds), all processes propose 0 and all processes decide 0.

$a \leftarrow$ *input*

loop:

    send_phase1(a)

    $A \leftarrow$ *receive_phase1*()

    if $(\exists a' \in A : |A_{a'}| > n/2)$:

        $b \leftarrow a'$

    else:

        $b \leftarrow \perp$

    send_phase2(b)

    $B \leftarrow$ *receive_phase2*()

    if $(\exists b' \in B : b' \neq \perp \wedge |B_{b'}| > f)$:

        decide(b')

    if $(\exists b' \in B : b' \neq \perp)$:

        $a \leftarrow b'$

    else:

        $a \leftarrow$ *choose_random({0,1})*

Safety ✓

$a \leftarrow input$

loop:

    $send\_phase1(a)$

    $A \leftarrow receive\_phase1()$

    if $(\exists a' \in A : |A_{a'}| > n/2)$:

        $b \leftarrow a'$

    else:

        $b \leftarrow \perp$

    $send\_phase2(b)$

    $B \leftarrow receive\_phase2()$

    if $(\exists b' \in B : b' \neq \perp \wedge |B_{b'}| > f)$:

        decide(b')

    if $(\exists b' \in B : b' \neq \perp)$:

        $a \leftarrow b'$

    else:

        $a \leftarrow choose\_random(\{0,1\})$

Termination

We know that if all processes propose the same value for a round, they all decide that value that round.

At worst, the probability of this happening on any particular round is $1/2^n$.

Why? By the previous lemma, all the non-random values are identical.

Over time, the probability of this happening on **at least one round** converges to 1.

a←*input*

loop:

send_phase1(a)

A←*receive_phase1*()

if ($\exists a' \in A : |A_{a'}| > n/2$):

$b \leftarrow a'$

else:

$b \leftarrow \perp$

send_phase2(b)

B←*receive_phase2*()

if ($\exists b' \in B : b' \neq \perp \wedge |B_{b'}| > f$):

decide(b')

if ($\exists b' \in B : b' \neq \perp$):

$a \leftarrow b'$

else:

a←*choose_random({0,1})*

Liveness ✓???

```
a←input
loop:
    send_phase1(a)
    A←receive_phase1()
    if (∃a′ ∈ A : |A_a′| > n/2):
        b←a′
    else:
        b←⊥
    send_phase2(b)
    B←receive_phase2()
    if (∃b′ ∈ B : b′≠⊥ ∧ |B_b′| > f):
        decide(b′)
    if (∃b′ ∈ B : b′≠⊥):
        a←b′
    else:
        a←choose_random({0,1})
```

Random Termination

Ben-Or's algorithm guarantees termination with probability 1.

Consensus requires termination, i.e., that there does not exist an infinite execution in which a correct process never decides.

Are these the same?

Other Values?

Binary consensus is conceptually simple but not as useful. However, the algorithm can be to support larger domains, even when the processes don't know the domains *a priori* and even when some processes don't receive input values.

$a \leftarrow$ *input*

loop:

    send_phase1(a)

    $A \leftarrow$ *receive_phase1*$()$

    if $(\exists a' \in A : |A_{a'}| > n/2)$:

        $b \leftarrow a'$

    else:

        $b \leftarrow \perp$

    send_phase2(b)

    $B \leftarrow$ *receive_phase2*$()$

    if $(\exists b' \in B : b' \neq \perp \wedge |B_{b'}| > f)$:

        decide(b')

    if $(\exists b' \in B : b' \neq \perp)$:

        $a \leftarrow b'$

    else:

        $a \leftarrow$ *choose_random({0,1})*

OTHER VALUES?

Binary consensus is conceptually simple but not as useful. However, the algorithm can be to support larger domains, even when the processes don't know the domains *a priori* and even when some processes don't receive input values.

- Processes without input values start by proposing $\perp$.

$a \leftarrow input$

loop:

  $send\_phase1(a)$

  $A \leftarrow receive\_phase1()$

  if $(\exists a' \in A : |A_{a'}| > n/2)$:

    $b \leftarrow a'$

  else:

    $b \leftarrow \perp$

  $send\_phase2(b)$

  $B \leftarrow receive\_phase2()$

  if $(\exists b' \in B : b' \neq \perp \wedge |B_{b'}| > f)$:

    decide(b')

  if $(\exists b' \in B : b' \neq \perp)$:

    $a \leftarrow b'$

  else:

    $a \leftarrow choose\_random(\{0,1\})$

Other Values?

Binary consensus is conceptually simple but not as useful. However, the algorithm can be to support larger domains, even when the processes don't know the domains *a priori* and even when some processes don't receive input values.

- Processes without input values start by proposing ⊥.
- Instead of randomly choosing from {0,1}, processes randomly choose from all non-⊥ values they've seen so far (in any message). Only choose ⊥ as a last resort.

$a \leftarrow$ *input*

loop:

  send_phase1(a)

  $A \leftarrow$ *receive_phase1*()

  if ($\exists a' \in A : |A_{a'}| > n/2$):

    $b \leftarrow a'$

  else:

    $b \leftarrow \perp$

  send_phase2(b)

  $B \leftarrow$ *receive_phase2*()

  if ($\exists b' \in B : b' \neq \perp \wedge |B_{b'}| > f$):

    decide(b')

  if ($\exists b' \in B : b' \neq \perp$):

    $a \leftarrow b'$

  else:

    $a \leftarrow$ *choose_random*({0,1})

How Fast Can We Reach Consensus?

- The expected value of a geometric random variable where $p = 1/2^n$ is 2^n. Not great.

- The earlier analysis was not tight, however. When $f << n/2$, we get convergence much quicker.

- More efficient algorithms exist.

Does Anyone Actually Use This?

As far as I know, not really. (Except in proof-of-stake cryptocurrency systems, more on that later.)

Randomized techniques can make termination guarantees but:

Does Anyone Actually Use This?

As far as I know, not really. (Except in proof-of-stake cryptocurrency systems, more on that later.)

Randomized techniques can make termination guarantees but:

- Performance is not predictable.

Does Anyone Actually Use This?

As far as I know, not really. (Except in proof-of-stake cryptocurrency systems, more on that later.)

Randomized techniques can make termination guarantees but:

- Performance is not predictable.
- Convergence can still take multiple rounds.

Does Anyone Actually Use This?

As far as I know, not really. (Except in proof-of-stake cryptocurrency systems, more on that later.)

Randomized techniques can make termination guarantees but:

- Performance is not predictable.
- Convergence can still take multiple rounds.
- Amount of communication needed is potentially high.

Does Anyone Actually Use This?

As far as I know, not really. (Except in proof-of-stake cryptocurrency systems, more on that later.)

Randomized techniques can make termination guarantees but:

- Performance is not predictable.
- Convergence can still take multiple rounds.
- Amount of communication needed is potentially high.

In practice, when there's a stable leader (more on this next) Paxos reaches consensus is a single round of communication. And if your network is not behaving well, you've got bigger problems.

Takeaways

- Randomization can actually solve consensus*! And that's neat!
- You can structure an asynchronous protocol using rounds. It's potentially useful and certainly an interesting way to think about asynchronous computation.