Back to the protocol...

- To broadcast a message in round $r$, $p$ sends $(init, p, m, r)$ to all.
- A confirmation has the form $(echo, p, m, r)$.
- A witness sends $(echo, p, m, r)$ if either:
  - it receives $(init, p, m, r)$ from $p$ directly or
  - it receives confirmations for $(p, m, r)$ from at least $f+1$ processes (at least one correct witness).
- A process accepts $(p, m, r)$ if it has received $n-f$ confirmations (as many as possible...).
- Protocol proceeds in rounds. Each round has 2 phases.

Implementation of broadcast and accept

**Phase $2r-1$**

1. $p$ sends $(init, p, m, r)$ to all

**Phase $2r$**

2. if $q$ received $(init, p, m, r)$ in phase $2r-1$ then
3. $q$ sends $(echo, p, m, r)$ to all /* $q$ becomes a witness */
4. if $q$ receives $(echo, p, m, r)$ from at least $n-f$ distinct processes in phase $2r$ then
5. $q$ accepts $(p, m, r)$

**Phase $j > 2r$**

6. if $q$ has received $(echo, p, m, r)$ from at least $f+1$ distinct processes in phases $(2r, 2r+1, \ldots, j-1)$ then
7. $q$ sends $(echo, p, m, r)$ to all processes /* $q$ becomes a witness */
8. if $q$ has received $(echo, p, m, r)$ from at least $n-f$ processes in phases $(2r, 2r+1, \ldots, j)$ then
9. $q$ accepts $(p, m, r)$

Is termination a problem?

The implementation is correct

**Theorem**

If $n > 3f$, the given implementation of $broadcast(p, m, r)$ and $accept(p, m, r)$ satisfies Unforgeability, Correctness, and Relay.

**Assumption**

Channels are authenticated.

Correctness

If a correct process $p$ executes $broadcast(p, m, r)$ in round $r$, then all correct processes will execute $accept(p, m, r)$ in round $r$. 
Correctness

If a correct process $p$ executes $\text{broadcast}(p, m, r)$ in round $r$, then all correct processes will execute $\text{accept}(p, m, r)$ in round $r$.

If $p$ is correct then
- $p$ sends $(\text{init}, p, m, r)$ to all in round $r$ (phase $2r-1$)
- by Validity of the underlying send and receive, every correct process receives $(\text{init}, p, m, r)$ in phase
- every correct process becomes a witness
- every correct process sends $(\text{echo}, p, m, r)$ in phase $2r$
- Since there are at least $n-f$ correct processes, every correct process receives at least $n-f$ echoes in phase $2r$
- every correct process executes $\text{accept}(p, m, r)$ in phase $2r$ (in round $r$)

Unforgeability - 1

If a correct process $q$ executes $\text{accept}(p, m, r)$ in round $j \geq r$, and $p$ is correct, then $q$ did in fact execute $\text{broadcast}(p, m, r)$ in round $r$

- Suppose $q$ executes $\text{accept}(p, m, r)$ in round $j$
- $q$ received $(\text{echo}, p, m, r)$ from at least $n-f$ distinct processes by phase $k$, where $k = 2j - 1$ or $k = 2j - f$
- Let $k'$ be the earliest phase in which some correct process $q'$ becomes a witness to $(p, m, r)$

Unforgeability - 2

For $q$ to accept, some correct process must become witness.

Earliest correct witness $q'$ becomes so in phase $2r - 1$, and only if $p$ did indeed execute $\text{broadcast}(p, m, r)$

Any correct process that becomes a witness later can only do so if a correct process is already a witness.

For any correct process to become a witness, $p$ must have executed $\text{broadcast}(p, m, r)$
If a correct process \( q \) executes \( \text{accept}(p, m, r) \) in round \( j \geq r \), then all correct processes will execute \( \text{accept}(p, m, r) \) by round \( j + 1 \).

Suppose correct \( q \) executes \( \text{accept}(p, m, r) \) in round \( j \) (phase \( k = 2j - 1 \) or \( k = 2j \)).
- \( q \) received at least \( n - f \) \((\text{echo}, p, m, r)\) from distinct processes by phase \( k \).
- At least \( n - 2f \) of them are correct.
- All correct proc received \((\text{echo}, p, m, r)\) from at least \( n - 2f \) correct processes by phase \( k \).
- From \( n > 3f \), it follows that \( n - 2f \geq f + 1 \).
- Then, all correct processes become witnesses by phase \( k \).
- All correct processes send \((\text{echo}, p, m, r)\) by phase \( k + 1 \).
- Since there are at least \( n - f \) correct processes, all correct processes will accept \((p, m, r)\) by phase \( k + 1 \) (round \( 2j \) or \( 2j + 1 \)).

Taking a step back...

Specified Consensus and TRB
- In the synchronous model:
  - solved Consensus and TRB for General Omission failures
  - proved lower bound on rounds required by TRB
  - solved TRB for AFMA
  - proved lower bound on replication for solving TRB with AF
  - solved TRB with AF
What about the asynchronous model?

**Theorem**

There is no deterministic protocol that solves Consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

The Intuition

In an asynchronous system, a process $p$ cannot tell whether a non-responsive process $q$ has crashed or it is just slow.

- If $p$ waits, it might do so forever.
- If $p$ decides, it may find out later that $q$ came to a different decision.
The Model - 1

- $n$ processes
- a message buffer

Message: $(p, \text{data}, q)$ or $\lambda$

null message

sender

receiver

Message Buffer
The Model – 2

An algorithm \( \mathcal{A} \) is a sequence of steps.

Each step consists of two phases:

- **Receive phase** – some \( p \) removes from buffer \((x, data, p)\) or \( \lambda \).

- **Send phase** – \( p \) changes its state; adds zero or more messages to buffer.

\( p \) can receive \( \lambda \) even if there are messages for \( p \) in the buffer.
Assumptions

Liveness Assumption:
Every message sent will be eventually received if intended receiver tries infinitely often

One-time Assumption:
$p$ sends $m$ to $q$ at most once

WLOG, process $p_i$ can only propose a single bit $b_i$
A configuration $\mathcal{C}$ of $\mathcal{A}$ is a pair $(s, M)$ where:
- $s$ is a function that maps each $p_i$ to its local state
- $M$ is the set of messages in the buffer

A step $e \equiv (p, m, \mathcal{A})$ is applicable to $\mathcal{C} = (s, M)$ if and only if $m \in M \cup \{\lambda\}$. Note: $(p, \lambda, \mathcal{A})$ is always applicable to $\mathcal{C}$

$\mathcal{C}' \equiv e(\mathcal{C})$ is the configuration resulting from applying $e$ to $\mathcal{C}$
Schedules

- A **schedule** $S$ of $A$ is a finite or infinite sequence of steps of $A$

- A schedule $S$ is **applicable** to a configuration $C$ if and only if either
  - $S$ is the empty schedule $S_\perp$ or
  - $S[1]$ is applicable to $C$
  - $S[2]$ is applicable to $S[1](C)$; etc.

- If $S$ is finite, $S(C)$ is the unique configuration obtained by applying $S$ to $C$
A configuration $C'$ is accessible from a configuration $C$ if there exist a schedule $S$ such that $C' = S(C)$.

$C'$ is a configuration of $S'(C)$ if $\exists S'$ prefix of $S$ such that $S'(C) = C'$.
A run of $\mathcal{A}$ is a pair $< I, S >$ where
- $I$ is an initial configuration
- $S$ is an infinite schedule of $\mathcal{A}$ applicable to $I$

A run is partial if $S$ is a finite schedule of $\mathcal{A}$

A run is admissible if every process, except possibly one, takes infinitely many steps in $S$

An admissible run is unacceptable if every process, except possibly one, takes infinitely many steps in $S$ without deciding
Structure of the proof

Show that, for any given consensus algorithm $\mathcal{A}$, there always exists an unacceptable run.

In fact, we will show an unacceptable run in which no process crashes!
Classifying Configurations

**0-valent:** A configuration $C$ is 0-valent if some process has decided 0 in $C$, or if all configurations accessible from $C$ are 0-valent.

**1-valent:** A configuration $C$ is 1-valent if some process has decided 1 in $C$, or if all configurations accessible from $C$ are 1-valent.

**Bivalent:** A configuration $C$ is bivalent if some of the configurations accessible from it are 0-valent while others are 1-valent.
Bivalent initial configurations happen

**Lemma 1**

There exists a bivalent initial configuration
Proof

- Suppose $A$ solves consensus with 1 crash failure
- Let $I_j$ be the initial configuration in which the first $j$ $b_i$'s are 1
- $l_0$ is 0-valent; $l_n$ is 1-valent
- By contradiction, suppose no bivalent
Proof

- Suppose \( \mathcal{A} \) solves consensus with 1 crash failure
- Let \( I_j \) be the initial configuration in which the first \( j \) \( b_i \)'s are 1
- \( I_0 \) is 0-valent; \( I_n \) is 1-valent
- By contradiction, suppose no bivalent
- Let \( k \) be smallest index such that \( I_k \) is 1-valent
- Obviously, \( I_{k-1} \) is 0-valent
- Suppose \( p_k \) crashes before taking any step.
- Since \( \mathcal{A} \) solves consensus even with one crash failure, there is a finite schedule \( S \) applicable to \( I_k \) that has no steps of \( p_k \) and such that some process decides in \( S(I_k) \)
- \( S \) is also applicable to \( I_{k-1} \)

CONTRADICTION
Lemma 2

Let $S_1$ and $S_2$ be schedules applicable to some configuration $C$, and suppose that the set of processes taking steps in $S_1$ is disjoint from the set of processes taking steps in $S_2$.

Then, $S_1; S_2$ and $S_2; S_1$ are both sequences applicable to $C$, and they lead to the same configuration.
Lemma 3

Let $C$ be bivalent, and let $e$ be a step applicable to $C$.

Then, there is a (possibly empty) schedule $S$ not containing $e$ such that $e(S(C))$ is bivalent.
Proof Sketch - 1

By contradiction, assume there is an $e$ for which no such $S$ exists.

Then, $e(C)$ is monovalent; WLOG assume 0-valent.
Proof Sketch - 1

By contradiction, assume there is an $e$ for which no such $S$ exists.

Then, $e(C)$ is monovalent; WLOG assume 0-valent.

Mini Lemma:
There exists an $e$-free schedule $S_0$ such that $D = S_0(C)$ and $e(D)$ is 1-valent.
Proof Sketch - 1

By contradiction, assume there is an $e$ for which no such $S$ exists.

Then, $e(C)$ is monovalent; WLOG assume $0$-valent.

Mini Lemma:
There exists an $e$-free schedule $S_0$ such that $D = S_0(C)$ and $e(D)$ is 1-valent.
Proof Sketch - 1

By contradiction, assume there is an $e$ for which no such $S$ exists

Then, $e(C)$ is monovalent; WLOG assume 0-valent

Mini Lemma:
There exists an $e$-free schedule $S_0$ such that $D = S_0(C)$ and $e(D)$ is 1-valent
Proof Sketch - 1

By contradiction, assume there is an $e$ for which no such $S$ exists.

Then, $e(C)$ is monovalent; WLOG assume 0-valent.

Mini Lemma:
There exists an $e$-free schedule $S_0$ such that $D = S_0(C)$ and $e(D)$ is 1-valent.
Proof Sketch- 2

Proof of mini Lemma.

Since $C$ is bivalent, there exists a schedule $S_1$ such that $E = S_1(C)$ is 1-valent.

Otherwise, let $S_0$ be the largest e-free prefix of $S_1$.

If $S_1$ is e-free, then $D = E$.

Diagram:
- $C$ and $S_1$ connections
- $E = D$ connection
- $S_0$ and $S_1$ connections
- $E$ and $S_1$ connections
Proof Sketch - 3

Consider configuration e(D).

By assumption, e(D) cannot be bivalent (otherwise we would have proved the Procrastination Lemma with $S = S_0$).

Since e(D) is monovalent, E is accessible from e(D), and E is 1-valent, then e(D) is 1-valent.
Proof Sketch - 3

Consider configuration $e(D)$.
By assumption, $e(D)$ cannot be bivalent (otherwise we would have proved the Procrastination Lemma with $S = S_0$).
Since $e(D)$ is monovalent, $E$ is accessible from $e(D)$, and $E$ is 1-valent, then $e(D)$ is 1-valent.

By the mini Lemma, on the "path" from $C$ to $D$ there must be two neighboring configurations $A$ and $B$ and a step $f$ such that:
- $B = f(A)$
- $e(A)$ is 0-valent
- $e(B)$ is 1-valent

---

Diagram:

- Configuration $C$ is connected to $D$.
- $D$ is connected to $E$.
- $S_0$ (e-free)
- $S_1$
- $S_0(e-free)$
- $C$ to $D$ through $E$.

Diagram:

- Configuration $C$ is connected to $D$.
- $D$ is connected to $E$.
- $S_0$ (e-free)
- $S_0(e-free)$
- $C$ to $D$ through $E$.
Proof Sketch - 4

Consider now $A$ and $B = f(A)$.

Claim: The same processes must take steps $e$ and $f$. 
Consider now $A$ and $B = f(A)$

Claim: The same processes $p$ must take steps $e$ and $f$

- Suppose not
- By Commutativity lemma,
  \[ e(B) = e(f(A)) = f(e(A)) \]
Proof Sketch – 4

Consider now $A$ and $B = f(A)$

Claim: The same processes $p$ must take steps $e$ and $f$

- Suppose not
- By Commutativity lemma,
  \[ e(B) = e(f(A)) = f(e(A)) \]
- Impossible since $e(B)$ is 1-valent and $e(A)$ is 0-valent
Since our protocol tolerates a failure, there is a schedule \( \rho \) applicable to \( A \) such that:

- \( R = \rho(A) \)
- Some process decides in \( R \)
- \( p \) does not take any steps in \( \rho \)
Since our protocol tolerates a failure, there is a schedule \( \rho \) applicable to \( A \) such that:

- \( R = \rho(A) \)
- Some process decides in \( R \)
- \( p \) does not take any steps in \( \rho \)

We show that the decision value in \( R \) can be neither 0 nor 1!
Proof Sketch – 6

Cannot be 0:

\[ \square \text{Consider } e(B) = e(f(A)) \]
Proof Sketch - 6

Cannot be 0:

- Consider $e(B) = e(f(A))$
- By Mini Lemma, we know it is 1-valent
Cannot be 0:

- Consider $e(B) = e(f(A))$
- By Mini Lemma, we know it is 1-valent
- Because it contains no steps of $p$, $\rho$ is applicable to $e(B)$
Cannot be 0:

- Consider \( e(B) = e(f(A)) \)
- By Mini Lemma, we know it is 1-valent
- Because it contains no steps of \( \rho \), \( \rho \) is applicable to \( e(B) \)
- The resulting configuration is 1-valent
Proof Sketch – 6

Cannot be 0:

- Consider \( e(B) = e(f(A)) \)
- By Mini Lemma, we know it is 1-valent
- Because it contains no steps of \( \rho \), \( \rho \) is applicable to \( e(B) \)
- The resulting configuration is 1-valent
- By Commutativity Lemma
  \[
  \rho(e(f(A))) = e(f(\rho(A))) = e(f(R))
  \]
Cannot be 0:

☐ Consider $e(B) = e(f(A))$

☐ By Mini Lemma, we know it is 1-valent

☐ Because it contains no steps of $\rho$, $\rho$ is applicable to $e(B)$

☐ The resulting configuration is 1-valent

☐ By Commutativity Lemma

$$\rho(e(f(A))) = e(f(\rho(A))) = e(f(R))$$

☐ Since $\rho(e(B))$ is accessible from $R$, and $\rho(e(B))$ is 1-valent, $R$ cannot be 0-valent
Proof Sketch – 7

Cannot be 1:

Consider $e(A)$
Proof Sketch - 7

Cannot be 1:
☐ Consider $e(A)$
☐ By construction, it is 0-valent
Cannot be 1:
- Consider $e(A)$
- By construction, it is 0-valent
- Because it contains no steps of $\rho$, $\rho$ is applicable to $e(A)$
Cannot be 1:
- Consider $e(A)$
- By construction, it is 0-valent
- Because it contains no steps of $\rho$, $\rho$ is applicable to $e(A)$
- The resulting configuration is 0-valent
Cannot be 1:

- Consider $e(A)$
- By construction, it is 0-valent
- Because it contains no steps of $\rho$, $\rho$ is applicable to $e(A)$
- The resulting configuration is 0-valent
- By Commutativity Lemma

$$\rho(e(A)) = e(\rho(A)) = e(R)$$
Cannot be 1:

☐ Consider $e(A)$

☐ By construction, it is 0-valent

☐ Because it contains no steps of $\rho$, $\rho$ is applicable to $e(A)$

☐ The resulting configuration is 0-valent

☐ By Commutativity Lemma

\[ \rho(e(A)) = e(\rho(A)) = e(R) \]

☐ Since $\rho(e(A))$ is accessible from $R$, and $\rho(e(A))$ is 0-valent, $R$ cannot be 1-valent

Cannot decide in $R$: contradiction
Proving the FLP Impossibility Result

Theorem

There is no deterministic protocol that solves Consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

- By Lemma 1, there exists an initial bivalent configuration $I_{biv}$.
- Consider any ordering $p_{l_1}, \ldots, p_{l_n}$ of $p_1, \ldots, p_n$.
- Pick any applicable step $e_1 = (p_{l_1}, m_1)$.
- Apply Procrastination lemma to obtain another bivalent configuration $C_{biv}^1 = e_1(S_1(I_{biv}))$.
- Pick a step $e_2 = (p_{l_2}, m_2)$ applicable to $C_{biv}^1$.
- Apply Procrastination lemma to obtain another bivalent configuration.
- Continue as before in a round-robin fashion. How do we choose a step?
- We have built an unacceptable run!
How can one get around FLP?

Weaken the problem

Weaken termination
- use randomization to terminate with arbitrarily high probability
- guarantee termination only during periods of synchrony

Weaken agreement
- $\varepsilon$-agreement
  - real-valued inputs and outputs
  - agreement within real-valued small positive tolerance $\varepsilon$
- $k$-set agreement
  - Agreement: In any execution, there is a subset $W$ of the set of input values, $|W|=k$, s.t. all decision values are in $W$
  - Validity: In any execution, any decision value for any process is the input value of some process
How can one get around FLP?

Constrain input values

- Characterize the set of input values for which agreement is possible

Strengthen the system model

- Introduce *failure detectors* to distinguish between crashed processes and very slow processes