Back to the protocol...

- To broadcast a message in round $r$, $p$ sends $(\text{init}, p, m, r)$ to all
- A confirmation has the form $(\text{echo}, p, m, r)$
- A witness sends $(\text{echo}, p, m, r)$ if either:
  - it receives $(\text{init}, p, m, r)$ 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 $\text{broadcast and accept}(p, m, r)$

1. $2r - 1$
   - $p$ sends $(\text{init}, p, m, r)$ to all
2. $2r$:
   - If $q$ received $(\text{init}, p, m, r)$ in phase $2r - 1$ then
   - $q$ sends $(\text{echo}, p, m, r)$ to all
   - $q$ becomes a witness
   - $q$ accepts $(p, m, r)$
3. $j > 2r$
   - If $q$ has received $(\text{echo}, p, m, r)$ from at least $n - f$ distinct processes in phase $(2r, 2r + 1, \ldots, j - 1)$ then
   - $q$ becomes a witness
   - $q$ sends $(\text{echo}, p, m, r)$ to all
   - $q$ accepts $(p, m, r)$

- Is termination a problem?

The Implementation is Correct

**Theorem**

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

**Assumption**

Channels are authenticated

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 $2r - 1$
  - 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 p 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 \(k = 2j\).
- Let \(k'\) be the earliest phase in which some correct process \(q'\) becomes a witness to \((p,m,r)\).

Case 1: \(k' = 2r - 1\)
- \(q'\) received \((\text{init},p,m,r)\) from p.
- Since p is correct, it follows that p did execute \(\text{broadcast}(p,m,r)\) in round \(r\).

Case 2: \(k' > 2r - 1\)
- \(q'\) has become a witness by receiving \(\text{echo}(p,m,r)\) from \(f + 1\) distinct processes.
- At most \(f\) are faulty; one is correct.
- This process was a witness to \((p,m,r)\) before phase \(k'\).
- CONTRADICTIO

The first correct process receives \((\text{init},p,m,r)\) from p!

Unforgeability -2

- For q to accept, some correct processes must become witnesses.
- 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 \(\text{broadcast}(p,m,r)\).

Relay

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.
- Then, all correct processes received \(\text{echo}(p,m,r)\) from at least \(n - 2f\) correct processes by phase \(k\).
- From \(n > 3f\), it follows that \(n - 2f / f + 1\). 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
  - 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
(Fisher, Lynch, and Paterson. Impossibility of distributed consensus with one faulty process. JACM, Vol. 32, no. 2, April 1985, pp. 374-382)

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

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 0 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 \( p \) can only propose a single bit \( b \)

### Definitions - 1

**Configuration** of \( A \)

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

**Schedule** of \( A \) A finite or infinite sequence of steps \( S \) of \( A \)

A schedule \( S \) is applicable to a configuration \( C \) 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.

**Step** \( e = (p, m, \mathcal{A}) \) is applicable to \( C = (s, M) \) if \( m \in M \cup \{\lambda\} \)

**Note:** \((p, \lambda, \mathcal{A})\) is always applicable to \( C \)

**Configuration** \( C' = e(C) \) : configuration that results when \( e \) is applied to \( C \)

### Definitions - 2

- 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 \( S : S'(C) = C' \)

**Run of** \( A \) \( R = < I, S > \)

- \( I \) is an initial configuration
- \( S \) is a finite schedule of \( \mathcal{A} \) applicable to \( I \)

**Partial run** of \( A \) \( R = < I, S > \)

- \( I \) is an initial configuration
- \( S \) is a finite schedule of \( \mathcal{A} \) applicable to \( I \)

**Run** is admissible if every process, except possibly one, takes infinitely many steps in \( S \)

**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 \( A \), there always exists an unacceptable run

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