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
- message: $(p, \text{data}, q)$ or $\lambda$
- null message
- Message Buffer
- sender
- receiver
The Model - 2

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

Each step consists of two phases:

- **Receive phase** - some \( p \) removes from buffer \( (x, \text{data}, p) \) 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 \( p \), can only propose a single bit \( b_i \).

Configurations

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

\( C' \equiv e(C) \) is the configuration resulting from applying \( e \) to \( C \).

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 \( C \).
Schedules and configurations

- 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 $S'(C) = C'$

Runs

- A run of $A$ is a pair $<I, S>$ where $I$ is an initial configuration
- $S$ is an infinite schedule of $A$ applicable to $I$
- A run is partial if $S$ is a finite schedule of $A$
- 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 $A$, 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 $j$ $b$'s are 1
- $I_0$ is 0-valent; $I_n$ is 1-valent
- By contradiction, suppose no bivalent $I_n$

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 $A$ 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 $I_{k-1}$

**Commutativity Lemma**

**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.
Procrastination Lemma

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.

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 $S_0$ is $e$-free, then $D = E$.

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

Proof Sketch - 4

- Claim: The same processes \( p \) must take steps \( e \) and \( f \).

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

Claim: The same processes \( p \) must take steps \( e \) and \( f \).

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

Claim: The same processes \( p \) must take steps \( e \) and \( f \).

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 \( \rho \)

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

Cannot be 0:
- Consider \( e(B) = e(f(A)) \)
- By Mini Lemma, we know it 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 \( p \), \( p \) is applicable to \( e(B) \)
☐ The resulting configuration is 1-valent

By Commutativity Lemma
\[
\rho(e(f(A))) = e(f(\rho(A))) = e(f(R))
\]
Proof Sketch - 7

Cannot be 1:
- Consider $c(A)$
- By construction, it is 0-valent
- Because it contains no steps of $p$, $p$ is applicable to $c(A)$

\[ \text{By construction, it is 0-valent} \]
\[ \text{Because it contains no steps of } p, \ p \text{ is applicable to } c(A) \]
\[ \text{The resulting configuration is 0-valent} \]
Cannot be 1:
- Consider $e(A)$
- By construction, it is 0-valent
- Because it contains no steps of $p$, $p$ is applicable to $e(A)$
- The resulting configuration is 0-valent
- By Commutativity Lemma $\rho(e(A)) = e(\rho(A)) = e(R)$

Cannot decide in $R$: contradiction

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
  - $k$-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

Condition-based Consensus

Is it possible to identify the set of conditions on the input values under which consensus is solvable?

- "all processes propose the same value"

- .... ?