FLP Impossibility of Consensus

Yan Ji      Oct 26, 2017

Slides inspired by Lorenzo Alvisi (CS5414 FA16) slides
and Philip Daian (CS6410 FA16) slides
I think you ought to know I'm feeling very depressed.
I think you ought to know I'm feeling very depressed.

I have a million ideas, but, they all point to certain death...
FLP Impossibility of Consensus

Yan Ji       Oct 26, 2017

Slides inspired by Lorenzo Alvisi (CS5414 FA16) slides and Philip Daian (CS6410 FA16) slides
Impossibility of distributed consensus with one faulty process (1985)
2001 Dijkstra prize for the most influential paper in distributed computing

Michael Fischer, Yale University
Distributed computing, Cryptography

Nancy Lynch, MIT
Distributed computing theory: Algorithms and lower bounds, Modeling and verification, Wireless networks, Biological algorithms

Mike Paterson, University of Warwick
Algorithms, Complexity
FLP Result

It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.
Considering how Paxos works, what is the most difficult part for reaching consensus in an asynchronous distributed system?
When does liveness fail in Paxos

1. P1 receives promises for n1
When does liveness fail in Paxos

1. P1 receives promises for n1
2. P2 receives promises for n2 > n1
3. P1 sends proposal numbered n1, rejected
When does liveness fail in Paxos

1. P1 receives promises for n1
2. P2 receives promises for n2 > n1
3. P1 sends proposal numbered n1, rejected
4. P1 receives promises for n1’ > n2
5. P2 sends proposal numbered n2, rejected
When does liveness fail in Paxos

1. P1 receives promises for n1
2. P2 receives promises for n2 > n1
3. P1 sends proposal numbered n1, rejected
4. P1 receives promises for n1' > n2
5. P2 sends proposal numbered n2, rejected
6. P1 receives promises for n2' > n1'
7. P1 sends proposal numbered n1', rejected
When does liveness fail in Paxos

1. P1 receives promises for n1
2. P2 receives promises for n2 > n1
3. P1 sends proposal numbered n1, rejected
4. P1 receives promises for n1' > n2
5. P2 sends proposal numbered n2, rejected
6. P1 receives promises for n2' > n1'
7. P1 sends proposal numbered n1', rejected
8. ...
The Intuition

Considering how Paxos works, what is the most difficult part for reaching consensus in an asynchronous distributed system?
The Intuition

Considering how Paxos works, what is the most difficult part for reaching consensus in an asynchronous distributed system?

CANNOT DISTINGUISH between processes:

- Crash failure
- Slow (e.g. in processing or network message delivering)
Assumption

It is impossible to have a deterministic protocol that solves consensus in a message-passing \textit{asynchronous} system in which at most one process may fail by crashing.
Assumption

It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

- No assumptions about the relative speeds of processes or about the delay time in delivering a message
- Processes don’t have access to synchronized clocks
- Unable to detect the death of a process
Assumption

It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.
**Assumption**

It is impossible to have a deterministic protocol that solves **consensus** in a message-passing asynchronous system in which at most one process may fail by crashing.

- **Termination**: all non-faulty processes eventually decide on a value
- **Agreement**: all processes that decide do so on the same value
- **Validity**: the value that has been decided must have been proposed by some process
Assumption

It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

For simplicity,

- **Termination**: all non-faulty processes eventually decide on a value in {0, 1}
- **Agreement**: all processes that decide do so on the same value
- **Validity**: both 0 and 1 are possible decision values
Assumption

It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.
Assumption

It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by **crashing**.

No Byzantine, no fail-stop, just CRASH.

All processes follow the protocol except for that at most one might crash.
It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.
Model

It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.
It is impossible to have a deterministic protocol that solves consensus in a \textit{message-passing} asynchronous system in which at most one process may fail by crashing.
It is impossible to have a deterministic protocol that solves consensus in a *message-passing* asynchronous system in which at most one process may fail by crashing.
Model

It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.
It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.
It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.
It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.
It is impossible to have a deterministic protocol that solves consensus in a **message-passing** asynchronous system in which at most one process may fail by crashing.
Model

It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.
Model

It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.
It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.
It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

Deal with Ø
Model

It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

- All messages are delivered correctly and exactly once
- The message buffer acts nondeterministically
- Inserted with arbitrary number of empty messages
Model

It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.
Model

It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

- **Configuration:** $C = (s, M)$
  - $M$: message buffer
  - $s$: internal states of processes
    - Input register
    - Output register
    - Internal storage
    - Program counter
Model

It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

- Configuration: \( C = (s, M) \)
  - \( M \): message buffer
  - \( s \): internal states of processes
    - Input register, value in \( \{0, 1\} \)
    - Output register, value in \( \{b, 0, 1\} \)
    - Internal storage
    - Program counter
It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

- Event: $e = (p, m)$
Model

It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

- Step: $C' = e(C) = (s', M')$

```
C = (s, M)

p_1: 0/1, b/0/1
... 
pi: 0/1, b/0/1
... 
k: 0/1, b/0/1
```

$(p_i, m)$
Model

It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

- Step: $C' = e(C) = (s', M')$
It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

- Step: $C' = e(C) = (s', M')$
It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

- Step: $C' = e(C) = (s', M')$
Model

It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

- Step: $C' = e(C) = (s', M')$

$$
\begin{align*}
C &= (s, M) \\
C' &= e(C) = (s', M') \\
M' &= \{ p_i, m \} \\
\end{align*}
$$
It is impossible to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

Step: $C' = e(C) = (s', M')$

- $C = (s, M)$
- $0/1, b/0/1$
- $p_1, p_2, ..., p_k$
- $M$
- $p_i$
- $0/1, b/0/1$
- $M'$
- $C' = (s', M')$
- $p_i$, $m$
- $s'$
- $s$
It is impossible to have a **deterministic** protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

- **Step:** $C' = e(C) = (s', M')$

$$C = (s, M)$$

$$C' = (s', M')$$
Proof

It is **impossible** to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

How to prove impossibility?
Proof

It is **impossible** to have a deterministic protocol that solves consensus in a message-passing asynchronous system in which at most one process may fail by crashing.

How to prove impossibility?

Assume to the contrary that there exists a consensus protocol $P$ such that...

How to define $P$?
More terms

- A **schedule** $S$ of $P$ is a finite or infinite sequence of events $(e_1, e_2, ..., e_k)$ of $P$, $S(C) = e_k(...)e_2(e_1(C)))...$
More terms

- A **schedule** $S$ of $P$ is a finite or infinite sequence of events $(e_1, e_2, ..., e_k)$ of $P$, $S(C) = e_k(...(e_2(e_1(C))))...$

- A **run** of $P$ is a sequence of steps associating a schedule $S$, in other words, a run is a pair of a configuration $C$ and a schedule $S$, written as $(C, S)$
More terms

- A configuration C’ is **reachable** from a configuration C if there exist a schedule S such that C’ = S(C)
More terms

- A configuration $C'$ is **reachable** from a configuration $C$ if there exist a schedule $S$ such that $C' = S(C)$
- A configuration $C'$ is **accessible** from an initial configuration $C_0$ if $C'$ is reachable from $C_0
More terms

- A configuration $C'$ is **reachable** from a configuration $C$ if there exist a schedule $S$ such that $C' = S(C)$
- A configuration $C'$ is **accessible** from an initial configuration $C_0$ if $C'$ is reachable from $C_0$
More terms

- A configuration \( C \) has decision value \( v \) if some process \( p \) is in a decision state with output \( =v \), which is “write-once”/irreversible.
More terms

● A configuration C has decision value $v$ if some process $p$ is in a decision state with output=$v$, which is “write-once”/irreversible

● A run is a deciding run if some process reaches a decision state.
More terms

- A configuration \( C \) has decision value \( v \) if some process \( p \) is in a decision state with output=\( v \), which is “write-once”/irreversible.
- A run is a deciding run if some process reaches a decision state.
More terms

- A configuration $C$ has **decision value** $v$ if some process $p$ is in a decision state with output=$v$, which is “write-once”/irreversible.
- A run is a **deciding** run if some process reaches a decision state.

$C_0 = (s_0, M_0)$

$C_1 = (s_1, M_1)$

$C_2 = (s_2, M_2)$
A consensus protocol $P$ is **partially correct** if:
- No accessible configuration has more than one decision value (agreement)
- For each $v$ in $\{0, 1\}$, some accessible configuration has decision value $v$ (validity)

A run is **admissible** if every process, except possibly one (faulty process), takes infinitely many steps in $S$
Assume to the contrary that there exists $P$ such that

- $P$ is partially correct
  - Agreement + Validity
Assume to the contrary that there exists $P$ such that

- $P$ is partially correct
  - Agreement + Validity

- Every admissible run of $P$ is a deciding run
  - Termination
Assume to the contrary that there exists $P$ such that

- $P$ is partially correct
  - Agreement + Validity
- Every admissible run of $P$ is a deciding run
  - Termination

- What kind of contradiction should possibly be like?
Categories of configurations

- Univalent, or i-valent (i in \{0, 1\})
  - A configuration C is univalent or i-valent if some process has decided i in C, or if all configurations accessible from C are i-valent
Categories of configurations

- Univalent, or i-valent (i in \{0, 1\})
  - A configuration \(C\) is univalent or i-valent if some process has decided \(i\) in \(C\), or if all configurations accessible from \(C\) are i-valent.
Categories of configurations

- Univalent, or i-valent (i in \{0, 1\})
  - A configuration \( C \) is univalent or i-valent if some process has decided \( i \) in \( C \), or if all configurations accessible from \( C \) are i-valent.
Categories of configurations

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

$C = (s, M)$

$\begin{align*}
0, b \\
\vdots \\
1, b \\
\vdots \\
s
\end{align*}$

$\begin{align*}
S_1 \\
\vdots \\
S_2 \\
\vdots \\
S_3 \\
\vdots
\end{align*}$

Decide on 0

Decide on 1

Decide on 0
Categories of configurations

- Bivalent (see Bivalent, read Undeciding)
  - A configuration $C$ is bivalent if some of the configurations accessible from it are 0-valent while others are 1-valent.

$$C = (s, M)$$

$$p_1$$

0, b

$\cdots$

$\cdots$

$\cdots$

$\cdots$

$\cdots$

$p_k$

1, b

$\cdots$

$S_1$

$S_2$

$S_3$

Decide on 0

Decide on 1

Decide on 0
What kind of contradiction should possibly be like?

**INDISTINGUISHABILITY** between processes:

- Crashed
- Simply slow in processing or having a terrible network condition
What kind of contradiction should possibly be like?

**INDISTINGUISHABILITY** between processes:

- Crashed
- Simply slow in processing or having a terrible network condition

For any protocol, there exists a configuration that is always bivalent.
What kind of contradiction should possibly be like?

**INDISTINGUISHABILITY** between processes:

- Crashed
- Simply slow in processing or having a terrible network condition

For any protocol, there exists a configuration that is always bivalent.

Remaining **UNDECIDED** in the value
What kind of contradiction should possibly be like?

**INDISTINGUISHABILITY** between processes:

- Crashed
- Simply slow in processing or having a terrible network condition

For any protocol, there exists a configuration that is always bivalent.

Remaining **UNDECIDED** in the value
Proof Outline

- For any protocol, there is an initial configuration that is bivalent
- Then there is another bivalent configuration reachable from it after applying some event
- And another reachable bivalent configuration
- ...
- An infinite undeciding run
Most exciting part!!!

- Lemma 1 (commutativity of schedules)
  - Suppose that from some $C$, the schedules $S_1$, $S_2$ lead to $C_1$, $C_2$ respectively. If the steps in $S_1$ and in $S_2$ are disjoint, then $S_2$ can be applied to $C_1$ and $S_1$ can be applied to $C_2$ and both lead to the same $C_3$. 
Most exciting part!!!

- Lemma 1 (commutativity of schedules)
  - Suppose that from some C, the schedules $S_1$, $S_2$ lead to $C_1$, $C_2$ respectively. If the steps in $S_1$ and in $S_2$ are disjoint, then $S_2$ can be applied to $C_1$ and $S_1$ can be applied to $C_2$ and both lead to the same $C_3$. 
Proof Outline

- For any protocol, there is an initial configuration that is bivalent
- Then there is another bivalent configuration reachable from it after applying some event
- And another reachable bivalent configuration
- ...
- An infinite undeciding run
Most exciting part!!

- Lemma 2
  - $P$ has a bivalent initial configuration.
Most exciting part!!!

- Lemma 2
  - $P$ has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.
Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.
Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.
Most exciting part!!!

- Lemma 2
  - $P$ has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.
Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.
Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.
Most exciting part!!!

- Lemma 2
  - $P$ has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.
Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.

Adjacent: differ in the initial state of a single process
Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.
Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.
Most exciting part!!!

- Lemma 2
  - $P$ has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.
Most exciting part!!!

- Lemma 2
  - $P$ has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.

Adjacent: differ in the initial state of a single process $S$ in which $p$ takes no step
Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.
Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.

Adjacent: differ in the initial state of a single process

S in which p takes no step

C_i

C_i+1

C/p

0/1

C_i

C_i+1

S in which p takes no step

C/p

0

0/1
Most exciting part!!!

- Lemma 2
  - $P$ has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.
Most exciting part!!!

- Lemma 2
  - P has a bivalent initial configuration.

Assume all initial configurations are either 0-valent or 1-valent.
Proof Outline

- For any protocol, there is an initial configuration that is bivalent
- Then there is another bivalent configuration reachable from it after applying some event
- And another reachable bivalent configuration
- ...
- An infinite undecided run
Most exciting part!!!

- Lemma 3
  - Let $C$ be a bivalent configuration of $P$, and $e=(p, m)$ be an event that is applicable to $C$. Let $E$ be the set of configurations reachable from $C$ without applying $e$, and let $D=e(E)$, the set of configurations after applying $e$ to all those in $E$. Then, $D$ contains a bivalent configuration.
Most exciting part!!!

- Lemma 3
  - Let $C$ be a bivalent configuration of $P$, and $e=(p, m)$ be an event that is applicable to $C$. Let $E$ be the set of configurations reachable from $C$ without applying $e$, and let $D=e(E)$, the set of configurations after applying $e$ to all those in $E$. Then, $D$ contains a bivalent configuration.
Lemma 3

Let $C$ be a bivalent configuration of $P$, and $e=\langle p, m \rangle$ be an event that is applicable to $C$. Let $E$ be the set of configurations reachable from $C$ without applying $e$, and let $D=e(E)$, the set of configurations after applying $e$ to all those in $E$. Then, $D$ contains a bivalent configuration.
Most exciting part!!!

- Lemma 3
  - Let $C$ be a bivalent configuration of $P$, and $e=(p, m)$ be an event that is applicable to $C$. Let $E$ be the set of configurations reachable from $C$ without applying $e$, and let $D=e(E)$, the set of configurations after applying $e$ to all those in $E$. Then, $D$ contains a bivalent configuration.
Lemma 3

Let $C$ be a bivalent configuration of $P$, and $e=(p, m)$ be an event that is applicable to $C$. Let $E$ be the set of configurations reachable from $C$ without applying $e$, and let $D=e(E)$, the set of configurations after applying $e$ to all those in $E$. Then, $D$ contains a bivalent configuration.
Most exciting part!!!

- Assume all configurations in D are univalent.
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for i in {0, 1} there exists a $C_i$ reachable from C that is i-valent.
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for $i$ in $\{0, 1\}$ there exists a $C_i$ reachable from C that is $i$-valent.
  - Consider $C_0$ without loss of generality,
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for i in {0, 1} there exists a $C_i$ reachable from C that is i-valent.
  - Consider $C_0$ without loss of generality,
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for i in \{0, 1\} there exists a \( C_i \) reachable from C that is i-valent.
  - Consider \( C_0 \) without loss of generality,

A schedule without applying e
Most exciting part!!!

- Assume all configurations in $D$ are univalent.
- There exists both 0-valent configuration and 1-valent configuration in $D$.
  - $C$ is bivalent, so for $i$ in $\{0, 1\}$ there exists a $C_i$ reachable from $C$ that is $i$-valent.
  - Consider $C_0$ without loss of generality,
Most exciting part!!!

- Assume all configurations in $D$ are univalent.
- There exists both 0-valent configuration and 1-valent configuration in $D$.
  - $C$ is bivalent, so for $i$ in $\{0, 1\}$ there exists a $C_i$ reachable from $C$ that is $i$-valent.
  - Consider $C_0$ without loss of generality,

A schedule without applying $e$
Most exciting part!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for $i$ in $\{0, 1\}$ there exists a $C_i$ reachable from C that is $i$-valent.
  - Consider $C_0$ without loss of generality,

A schedule already applied e
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for i in \{0, 1\} there exists a $C_i$ reachable from C that is i-valent.
  - Consider $C_0$ without loss of generality,
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for i in {0, 1} there exists a $C_i$ reachable from C that is i-valent.
  - Consider $C_0$ without loss of generality,
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
  - C is bivalent, so for i in \( \{0, 1\} \) there exists a \( C_i \) reachable from C that is i-valent.
  - Consider \( C_0 \) without loss of generality,
  - There exists \( D_0 \) that is 0-valent.
Most exciting part!!!

- Assume all configurations in $D$ are univalent.
- There exists both 0-valent configuration and 1-valent configuration in $D$.
- Without loss of generality, assume $D_0 = e(C)$ in $D$ is 0-valent
Most exciting part!!!

- Assume all configurations in $D$ are univalent.
- There exists both 0-valent configuration and 1-valent configuration in $D$.
- Without loss of generality, assume $D_0 = e(C)$ in $D$ is 0-valent.

**Neighbor**: one result from the other in a single step.
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
- Without loss of generality, assume $D_0 = e(C)$ in D is 0-valent.

**Neighbor:** one result from the other in a single step

$e = (p, m)$
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
- Without loss of generality, assume $D_0 = e(C)$ in D is 0-valent.
Assume all configurations in D are univalent.

There exists both 0-valent configuration and 1-valent configuration in D.

Without loss of generality, assume $D_0 = e(C)$ in D is 0-valent.

**Neighbor**: one result from the other in a single step.
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
- Without loss of generality, assume $D_0 = e(C)$ in D is 0-valent.
Most exciting part!!!

- Case 1: $p' \neq p$

Neighbor: one result from the other in a single step

$e = (p, m) \quad e' = (p', m')$
Most exciting part!!!

- Case 1: $p' \neq p$
  - Apply Lemma 1

**Neighbor**: one result from the other in a single step
Most exciting part!!!

- Case 1: $p' \neq p$
  - Apply Lemma 1

Neighbor: one result from the other in a single step

\[ e = (p, m) \]
\[ e' = (p', m') \]
Most exciting part!!!

- Case 1: \( p' \neq p \)
  - Apply Lemma 1
Most exciting part!!!

- Case 2: $p' = p$

Neighbor: one result from the other in a single step
Most exciting part!!!

- Case 2: $p' = p$

**Neighbor**: one result from the other in a single step

$e = (p, m)$

$e' = (p', m')$
Most exciting part!!!

- Case 2: $p' = p$
  - Apply Lemma 1

**Neighbor**: one result from the other in a single step

e = (p, m)
e' = (p', m')

deciding $S$ ($p$ takes no steps)
Most exciting part!!!

- Case 2: $p' = p$
  - Apply Lemma 1

Neighbor: one result from the other in a single step

e = (p, m)
e' = (p', m')

A deciding S (p takes no steps)
Most exciting part!!!

- Case 2: $p' = p$
  - Apply Lemma 1

Neighbor: one result from the other in a single step

e = (p, m)
e' = (p', m')

A deciding S (p takes no steps)
Most exciting part!!!

- Case 2: \( p' = p \)
  - Apply Lemma 1

Neighbor: one result from the other in a single step

\( e = (p, m) \)

\( e' = (p', m') \)
Most exciting part!!!

- Case 2: \( p' = p \)
  - Apply Lemma 1

**Neighbor**: one result from the other in a single step.
Most exciting part!!!

- Assume all configurations in D are univalent.
- There exists both 0-valent configuration and 1-valent configuration in D.
- Without loss of generality, assume $D_0 = e(C)$ in D is 0-valent
- Contradiction!
Proof Outline

- For any protocol, there is an initial configuration that is bivalent
- Then there is another bivalent configuration reachable from it after applying some event
- And another reachable bivalent configuration
- ...
- An infinite undeciding run
Most exciting part!!!

- Construct an infinite undeciding admissible run
Most exciting part!!!

- Construct an infinite undeciding admissible run

By Lemma 2
Most exciting part!!!

- Construct an infinite undeciding admissible run

\[ S_1 \text{ applying } e_1 = \text{receive}(p_1) \text{ last} \]

By Lemma 2

\[ C_0 \rightarrow C_1 \]

By Lemma 3

\[ 0/1 \rightarrow 0/1 \]
Most exciting part!!!

- Construct an infinite undeciding admissible run

By Lemma 2

By Lemma 3

By Lemma 3

\[ C_0 \rightarrow C_1 \rightarrow C_2 \]

\[ S_1 \text{ applying } e_1 = \text{receive}(p_1) \text{ last} \]

\[ S_2 \text{ applying } e_2 = \text{receive}(p_2) \text{ last} \]

\[ 0/1 \rightarrow 0/1 \rightarrow 0/1 \]
Most exciting part!!!

- Construct an infinite undeciding admissible run

\[ C_0 \quad C_1 \quad C_2 \quad C_3 \]

- \( S_1 \) applying \( e_1 = \text{receive}(p_1) \) last
- \( S_2 \) applying \( e_2 = \text{receive}(p_2) \) last
- \( S_3 \) applying \( e_3 = \text{receive}(p_3) \) last

By Lemma 2

By Lemma 3

By Lemma 3

By Lemma 3

0/1

0/1

0/1

0/1
Most exciting part!!!

- Construct an infinite undeciding admissible run

\[ C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow C_4 \]

\[ S_1 \text{ applying } e_1 = \text{receive}(p_1) \text{ last} \]
\[ S_2 \text{ applying } e_2 = \text{receive}(p_2) \text{ last} \]
\[ S_3 \text{ applying } e_3 = \text{receive}(p_3) \text{ last} \]
\[ S_4 \text{ applying } e_4 = \text{receive}(p_1) \text{ last} \]

By Lemma 2
By Lemma 3
By Lemma 3
By Lemma 3
By Lemma 3
Most exciting part!!!

- Construct an infinite undeciding admissible run

\[ C_0 \xrightarrow{e_1 = \text{receive}(p_1)} C_1 \xrightarrow{e_2 = \text{receive}(p_2)} C_2 \xrightarrow{e_3 = \text{receive}(p_3)} C_3 \xrightarrow{e_4 = \text{receive}(p_1)} \ldots \]

By Lemma 2

By Lemma 3

By Lemma 3

By Lemma 3

By Lemma 3

\[ S_1 \text{ applying } S_2 \text{ applying } S_3 \text{ applying } S_4 \text{ applying} \]
Most exciting part!!!

- Construct an infinite undeciding admissible run

By Lemma 2

By Lemma 3

By Lemma 3

By Lemma 3

An infinite UNDECIDING run
Assume to the contrary that there exists P such that

- P is partially correct
  - Agreement + Validity

- Every admissible run of P is a deciding run
  - Termination
Assume to the contrary that there exists $P$ such that

- $P$ is partially correct
  - Agreement + Validity

- Every admissible run of $P$ is a deciding run
  - Termination

Contradiction!
Wrap-up

- Computation determinism
  - Deterministic
  - Probabilistic
- Timing assumptions
  - Synchronous
  - Asynchronous
- Failure model
  - Fail-stop
  - Crash
  - Byzantine
  - Permissionless Byzantine
Wrap-up

- Computation determinism
  - Deterministic
  - Probabilistic
- Timing assumptions
  - Synchronous
  - Asynchronous
- Failure model
  - Fail-stop
  - Crash
  - Byzantine
  - Permissionless Byzantine
Wrap-up

- Computation determinism
  - Deterministic
  - Probabilistic

- Timing assumptions
  - Synchronous
  - Asynchronous

- Failure model
  - Fail-stop
  - Crash
  - Byzantine
  - Permissionless Byzantine

Impossibility
Takeaway

You CANNOT guarantee safety and liveness at the same time!
Takeaway

You CANNOT guarantee safety and liveness at the same time!

But you CAN get around FLP:

1. Release the failure model
Release the failure model

Model:
1. The majority are non-faulty
2. No process dies during the execution of the protocol

Two-stage protocol:
1. Listens for messages from L-1 other processes, L=N/2+1 (WHY?), and construct the incoming stream graph G
2. Construct G^+ and make decision upon values from the unique initial clique
Takeaway

You CANNOT guarantee safety and liveness at the same time!

But you CAN get around FLP:

1. Release the failure model
2. Terminate with probability of 1 instead of ALWAYS
Terminate with *probability of 1* instead of ALWAYS

Use randomization to terminate with arbitrarily high probability

M. Ben Or. “Another advantage of free choice: completely asynchronous agreement protocols” (PODC 1983, pp. 27-30)
Takeaway

You CANNOT guarantee safety and liveness at the same time!

But you CAN get around FLP:

1. Release the failure model
2. Terminate with **probability of 1** instead of ALWAYS
3. Use failure detector
Use failure detector

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