The Full Impossibility Result

• In an asynchronous system, consensus cannot be solved when the algorithm has to tolerate even just a single failure
  – n-1 processes cannot take an infinite number of steps without deciding
• Holds for the shared-memory model as well as for the message-passing model
• Describe both proofs in a unified manner using layered executions
  – below, f is the number of processes that may fail, we concentrate on the case f = 1

Layered Schedules

f-layer: sequence of at least n-f different processes
  – Order is sometimes important
f-schedule: sequence of f-layers

- \( p_2 \) is faulty in layer 2, but nonfaulty in layer k
- \( p_n \) crashes in layer 3 – faulty in every layer 3 ≤ r ≤ k

An f-schedule \( \sigma \) and an initial configuration I determine a layered execution \( \alpha(\sigma, I) \)
Similarity

$\alpha_1 \sim \alpha_2$

In execution $\alpha_1$

In execution $\alpha_2$

Connectivity

$\alpha_1 \approx \alpha_m \equiv \alpha_1 \sim \alpha_2 \sim \alpha_3 \sim \ldots \sim \alpha_m \Rightarrow \text{same decision}$
Key Lemma: Crashing a Process

\[ \text{crash}(\sigma, p, r): p \text{ crashes in layer } r \text{ of } \sigma \]

\[ \begin{array}{ccc}
\text{layer 1} & \text{layer 2} & \text{layer } k \\
p_1 & p_2 & \ldots & p_k \\
p_5 & p_2 & \ldots & p_8 \\
p_8 & p_5 & \ldots & p_1 \\
\end{array} \]

**Lemma:** For every input configuration \( I \), f-schedule \( \sigma \), process \( p \) and round \( r \), \( \alpha(\sigma, I) \approx \alpha(\text{crash}(\sigma, p, r), I) \)

---

Deriving the Impossibility Result: Input Connectivity

Consider a sequence of input configurations

\( I_0 = (0,0,\ldots,0) \)

\( I_1 = (1,0,\ldots,0) \)

\( I_2 = (1,1,\ldots,0) \)

\( \ldots \)

\( I_n = (1,1,\ldots,1) \)

**Claim:** \( \alpha(\sigma_F, I_0) \approx \alpha(\sigma_F, I_n) \), where \( \sigma_F \) is the full layered schedule

\( \Rightarrow \) Same decision in \( I_0 \) (all zeroes) and \( I_n \) (all ones)

**Contradiction!**
Getting the Claim from Key Lemma

Proving the Key Lemma

**Lemma:** For every input configuration $I$, process $p$ and round $r$, $\alpha(\sigma, I) \approx \alpha(\text{crash}(\sigma, p, r), I)$

The proof is very model dependent

- Shared memory: read / write (single-writer)
- Message passing

Need to assume bounded executions
\[ \text{swap}(\sigma, \pi_i, r) \]

Process \( \pi_i \) is swapped with the next process (\( \pi_j \)) in layer \( r \)

Both read or write (different registers) \( \Rightarrow \) no process distinguishes. \( \pi_j \) reads and \( \pi_i \) writes \( \Rightarrow \) only \( \pi_j \) distinguishes at the end of layer \( r \)
delay(σ, p_i, r)

The idea is to crash p_i by swapping every appearance of p_i until after the end of the schedule. But cannot swap with another appearance of p_i. Delay p_i by one layer, starting from layer r.

Swaps + Delays $\Rightarrow$ Crash

Now continue swapping…
Algebraically

\[ \text{delay}(\sigma,p,r) = \text{swap}^k(\text{rollover}(\text{swap}^k(\text{delay}(\sigma,p,r+1),p,r+1),p,r),p,r+1) \]

\( k' \) is \( p \)'s distance from the end of layer \( r \)
\( k \) is \( p \)'s distance from the beginning of layer \( r+1 \)

Similarly

\[ \text{crash}(\sigma,p,r) = \text{crash}(\text{delay}(\sigma,p,r),p,r+1) \]

Impossibility Result for Message-Passing Systems

Original context of this result (FLP)
Original proof has a different structure (similar to previous lecture)
Message-passing: Model of Computation

In each step, send messages to all processes
In layered executions, we synchronize the steps

Message-passing

• Crash \( p_i \) by removing it from all layers
  – Incremental \( \Rightarrow \) remove messages from \( p_i \) to \( p_j \)
  – Inductively, crash \( p_j \) in following layers
• Repeat for all layers \( \Rightarrow p_i \) crash
Bounding the Executions

• Why?
  • To have a well-defined base case for the (backwards) induction on the layer number

• How?
  • The proof considers a fixed (and bounded) set of executions from n+1 input configurations

Consensus in Synchronous Systems

"Then we are agreed nine to one that we will say our previous vote was unanimous!"
Synchronous Systems

- Processes take steps in rounds
- In each round, a process
  - sends messages to all (other) processes
  - receives messages from all other processes
  - does some local computation

Crash Failures in Synchronous Systems

- All but at most f faulty processes taken an infinite number of steps (or until everyone decides)
- Once a faulty processor fails to take a step in a round, it takes no more steps
- In the last step of a faulty process, some subset of its outgoing messages are sent
Consensus Algorithm for Crash Failures

- Tolerates $f < n$ crash failures
- Requires $f + 1$ rounds

Each process executes the following code

\[ v = \text{my input} \]
\[ \text{in each round } 1 \text{ through } f+1: \]
\[ \text{if } v \text{ not sent before, send } v \text{ to all} \]
\[ \text{wait to receive messages for this round} \]
\[ v = \min \text{ of received values and current value of } v \]
\[ \text{in round } f+1, \text{ decide on } v \]

- Tolerates $f < n$ crash failures
- Requires $f + 1$ rounds
- A total of $\leq n^2/|V|$ messages each with $\log |V|$ bits, where $V$ is the input set.
An Execution of the Algorithm: $p_i$ with input $v_i$

$v = \text{my input}$
in each round 1 through $f+1$:
  if $v$ not sent before, send $v$ to all
  wait to receive messages for this round
  $v = \min$ of received values and current value of $v$
in round $f+1$, decide on $v$

Correctness of Crash Consensus Algorithm

**Termination:** By the code, finish in round $f+1$.

**Validity:** processes do not create values.
If all inputs are the same, then that is the only value ever sent around (and decided)
Crash Consensus Algorithm: Agreement

Suppose in contradiction $p_j$ decides on a smaller value, $x$, than $p_i$ does

$\Rightarrow x$ was hidden from $p_i$ by a chain of faulty processes (one for each round)

$\Rightarrow$ This chain has $f + 1$ faulty processors, a contradiction

Is this the Best Round Complexity?
Rounds Lower Bound: Initial Lemma

**Lemma:** From some initial configuration, there are two executions $\gamma$ and $\alpha$, in which two different values are decided. $\gamma$ is failure-free, and in $\alpha$, one process crashes before taking any steps, but no other processes fail.

$C_0 = (0,0,\ldots,0,0)$

$\nu_0 = 0$ (by validity)

$C_i = (0,0,\ldots,1,1)$

failure-free

$\nu_i$ is decided

$C_n = (1,1,\ldots,1,1)$

$\nu_n = 1$ (by validity)

---

Rounds Lower Bound: Proof of Initial Lemma

**Lemma:** From some initial configuration, there are two executions $\gamma$ and $\alpha$, in which two different values are decided. $\gamma$ is failure-free, and in $\alpha$, one process crashes before taking any steps, but no other processes fail.

$C_i = (0,0,\ldots,0,1)$

failure-free

$p_j$ crashes at the start

$0$ is decided

$C_{i+1} = (0,0,\ldots,1,1)$

failure-free

$1$ is decided

$\checkmark$
Rounds Lower Bound: Proof of Initial Lemma

**Lemma:** From some initial configuration, there are two executions $\gamma$ and $\alpha$, in which two different values are decided. $\gamma$ is failure-free, and in $\alpha$, one process crashes before taking any steps, but no other processes fail.

$$C_i = (0,0,\ldots,0,1) \quad \text{failure-free} \quad 0 \text{ is decided}$$

$$C_{i+1} = (0,0,\ldots,1,1) \quad \text{failure-free} \quad 1 \text{ is decided}$$

Rounds Lower Bound: Main Lemma

We consider only $f$-round executions such that:

– $f \leq n-2$
– At most one process crashes in each round and at most $f$ processes crash in each execution.
– In the round in which a process crashes, it sends messages to a prefix of processes, ordered by id’s

**Lemma:** For any $f$-round execution $\alpha$, $\alpha \approx \gamma$, where $\gamma$ is the same as $\alpha$ during the first $r$ rounds but has no crashes after round $r$, $0 \leq r \leq f$. 
Rounds Lower Bound: 
Proof of Main Lemma (Base)

By backward induction on $r$

The base case, $r = f$, $\alpha = \gamma$
and the lemma is obvious

**Lemma:** For any $f$-round execution $\alpha$, $\alpha \approx \gamma$, where $\gamma$ is the same as $\alpha$ during the first $r$ rounds but has no crashes after round $r$, $0 \leq r \leq f$.

Rounds Lower Bound: 
Proof of Main Lemma (Inductive Step)

Assume $r < f$ and that the lemma holds for $r+1$. 
Rounds Lower Bound: Proof of Main Lemma (Inductive Step)

Assume $r < f$ and that the lemma holds for $r+1$. Let $\beta$ be the same as $\alpha$ during its first $r+1$ rounds and has no crashes after round $r+1$. By induction, $\alpha \approx \beta$; we need to show $\beta \approx \gamma$.

What happens in $\beta$?
$p$ is the single process that crashes in round $r+1$ of $\beta$ (if none fails then we are done).
$q_1, \ldots, q_t$ are the correct processes to which $p$ does not send a message in round $r+1$ (in order of id’s).
Rounds Lower Bound: Chain of Executions

\( \beta_k \) is the same as \( \beta \) in the first \( r+1 \) rounds, except that \( p \) sends messages to \( q_1, \ldots, q_k \) in round \( r+1 \)

\[ \beta_0 = \beta \]

A correct process does not distinguish \( \beta_t \) from \( \gamma \)

Rounds Lower Bound: \( r = f-1 \)

Some correct process \( \neq q_k \) does not distinguish between \( \beta_k \) and \( \beta_{k-1} \) (there is one since \( f < n-2 \))

\[ \beta \approx \beta_t \approx \gamma \]
Rounds Lower Bound:
\[ \beta_k \approx \beta_{k-1} \text{ for } r < f-1 \]

\( \gamma_k \) is the same as \( \beta_k \) for the first \( r+1 \) rounds, but \( q_k \) crashes in the beginning of round \( r+2 \) (cleanly) and there are no crashes after round \( r+2 \). By induction, \( \beta_k \approx \gamma_k \)

\( \gamma_k' \) is the same as \( \beta_{k-1} \) for the first \( r+1 \) rounds, but \( q_k \) crashes in the beginning of round \( r+2 \) (cleanly) and there are no crashes after round \( r+2 \). By induction, \( \beta_{k-1} \approx \gamma_k' \)

\[ \gamma_k \approx \gamma_k' \Rightarrow \beta_k \approx \beta_{k-1} \]

Theorem: Any consensus algorithm for \( n \geq f+2 \) processes that tolerates \( f \) crashes requires \( \geq f+1 \) rounds

Otherwise, apply initial configuration lemma

There is an initial configuration from which there are two executions \( \alpha \) and \( \gamma \) that decide different values

In \( \alpha \) and \( \gamma \) no processes crashes, except for one process that crashes before the start of \( \gamma \)

By previous lemma, \( \alpha \approx \gamma \)

\[ \Rightarrow \text{ Same value is decided in both} \]