Introduction to fault tolerance
Availability

The availability of a system is the probability that the system will perform its required action.
Availability

The **availability** of a system is the probability that the system will perform its required action.

**Example:**
- one workstation
- crashes once a week
- takes 2 minutes to recover
Availability

The availability of a system is the probability that the system will perform its required action.

Example:
- one workstation
- crashes once a week
- takes 2 minutes to recover

Availability:

\[ 1 - p_{\text{crash}} = 1 - 2 \cdot 10^{-4} = 0.9998 \]
Availability

The **availability** of a system is the probability that the system will perform its required action.

Example:

- One workstation
- Crash once a week
- Take 2 minutes to recover

Availability:
Availability

The availability of a system is the probability that the system will perform its required action.

Example:

- one workstation
- crash once a week
- take 2 minutes to recover

Availability:

\[ (1 - p_{\text{crash}})^{30} \approx 1 - n \cdot p_{\text{crash}} = 0.994 \]
Increasing availability

- Avoid a single point of failure
  - use replication (in time, or space)
  - replicas communicate through narrow interface (e.g. send/receive)
Increasing availability

- Avoid a single point of failure
  - use replication (in time, or space)
  - replicas communicate through narrow interface (e.g. send/receive)

- Example
  - Replicate the workstation 30 times

Availability:
Increasing availability

- Avoid a single point of failure
  - use replication (in **time**, or **space**)
  - replicas communicate through narrow interface (e.g. send/receive)

**Example**
- Replicate the workstation 30 times

**Availability:**

\[ 1 - p_{\text{crash}}^n = 1 - (2 \cdot 10^{-4})^{30} = 1 - 10^{111} \]
Modeling faults

- Mean Time To Failure/ Mean Time To Recover

- close to hardware

- Threshold: $f$ out of $n$

- makes condition for correct operation explicit

- measures fault-tolerance of architecture, not single components

- Set of explicit failure scenarios
A hierarchy of failure models

- Crash
A hierarchy of failure models

Fail-stop -- -- -- -- -- Crash
A hierarchy of failure models

Fail-stop → Crash
  ▼
Send Omission     ▼     Receive Omission
A hierarchy of failure models

- Fail-stop
- Crash
- Send Omission
- Receive Omission
- General Omission

benign failures
A hierarchy of failure models

- Fail-stop
- Crash
- Send Omission
- Receive Omission
- General Omission

Benign failures:
- Arbitrary failures with message authentication
A hierarchy of failure models

- Fail-stop
- Crash
- Send Omission
- Receive Omission
- General Omission
- Arbitrary failures with message authentication
- Arbitrary (Byzantine) failures

benign failures
Replication in space

- Run parallel copies of a unit
- Vote on replica output
- Failures are masked
- High availability, but at high cost
Replication in time

- When a replica fails, restart it (or replace it)
- Failures are detected, not masked
- Lower maintenance, lower availability
- Tolerates only benign failures
- Better than you think...
Non-determinism

An event is non-deterministic if the state that it produces is not uniquely determined by the state in which it is executed.

Handling non-deterministic events at different replicas is challenging.

- Replication in time requires to reproduce during recovery the original outcome of all non-deterministic events.
- Replication in space requires each replica to handle non-deterministic events identically.
Primary-Backup
The Idea

- Clients communicate with a single replica (the primary)
- The primary updates the other replicas (backups)
- Backups detect the failure of the primary using a timeout mechanism,
- Clients fail over to a backup

Note: Non-deterministic events are executed only at the primary
Terminology

- The **failover time** of a primary-backup service is the longest time the service can be without a primary.

- The service has a **server outage** at $t$ if some correct client sends a request at time $t$ to the service, but does not receive a response.

- A **$(k, \Delta)$-bofo service** is one in which all server outages can be grouped into at most $k$ intervals of time, each of at most length $\Delta$. 
**PB: A specification**

*(Budhiraja, Marzullo, Schneider, Toueg)*

**PB1:** There exists a local predicate $Prmys_s$ on the state of each server $s$. At any time, there is at most one server $s$ whose state satisfies $Prmys_s$.

**PB2:** Each client $i$ maintains a server identity $Dest_i$ such that to make a request, client $i$ sends a message to $Dest_i$.

**PB3:** If a client request arrives at a server that is not the current primary, then that request is not enqueued (and therefore is not processed).

**PB4:** There exist fixed values $k$ and $\Delta$ such that the service behaves like a single $(k,\Delta)$-bofo server.
A simple example: system model

- point-to-point communication
- non-faulty channels
- upper bound $\delta$ on message delivery time
- at most one server crashes
A simple example: system model

- point-to-point communication
- non-faulty channels
- upper bound $\delta$ on message delivery time
- at most one server crashes

Two processes:
- the primary $p_1$
- the backup $p_2$
A simple example: $p_1$'s protocol

- On receipt of a client request, process $p_1$
- consumes request and updates its state
- send state update message to $p_2$
- sends response to client without waiting for ack from $p_2$

$\tau$

$\circ$ $p_1$ sends heartbeat message to $p_2$ every $\tau$ seconds
A simple example: $p_2$’s protocol

- Upon receiving a state update from $p_1, p_2$, updates its state.
- If $p_2$ does not receive a heartbeat for $\tau + \delta$ seconds,
  - $p_2$ declares itself primary
  - It informs the clients
  - It begins consuming subsequent requests from clients
It meets the spec...

**PB1:** $Prmy_{p_1} \land Prmy_{p_2} = false$

**Failover:** Time during which $\neg Prmy_{p_1} \land \neg Prmy_{p_2}$

$Prmy_{p_1} \equiv p_1$ has not crashed

$Prmy_{p_2} \equiv p_2$ has not received a message from $p_1$ for $\tau + \delta$ seconds
It meets the spec...

PB1: $Prm_{yp1} \land Prm_{yp2} = false$

Failover: Time during which $\neg Prm_{yp1} \land \neg Prm_{yp2}$

$Prm_{yp1} \equiv p_1$ has not crashed

$Prm_{yp2} \equiv p_2$ has not received a message from $p_1$ for $\tau + \delta$ seconds
It meets the spec...

\[ Prmy_{p_1} \wedge Prmy_{p_2} = false \]

**PB1:** \( Prmy_{p_1} \wedge Prmy_{p_2} = false \)

**Failover:** Time during which

\[ \neg Prmy_{p_1} \wedge \neg Prmy_{p_2} \]

\( Prmy_{p_1} \equiv p_1 \) has not crashed

\( Prmy_{p_2} \equiv p_2 \) has not received a message from \( p_1 \) for \( \tau + \delta \) seconds
It meets the spec...

$P_{rmy}_{p_1} \land P_{rmy}_{p_2} = false$

**PB1:** $P_{rmy}_{p_1} \land P_{rmy}_{p_2} = false$

Failover: Time during which

$\neg P_{rmy}_{p_1} \land \neg P_{rmy}_{p_2}$

$Prmy_{p_1} \equiv p_1$ has not crashed

$Prmy_{p_2} \equiv p_2$ has not received a message from $p_1$ for $\tau + \delta$ seconds
It meets the spec...

PB1: $Prmy_{p_1} \land Prmy_{p_2} = \text{false}$

Failover: Time during which

$\neg Prmy_{p_1} \land \neg Prmy_{p_2}$

$Prmy_{p_1} \equiv p_1$ has not crashed

$Prmy_{p_2} \equiv p_2$ has not received a message from $p_1$ for $\tau + \delta$ seconds
...indeed, it does!

PB2, PB3: Follow immediately from protocol

PB4: Find $k$, $\Delta$ to implement $(k,\Delta)$-bofo server
...indeed, it does!

**PB2, PB3:** Follow immediately from protocol

**PB4:** Find $k, \Delta$ to implement $(k, \Delta)$-bofo server

- $k = 1$ (since at most one crash)
- $\Delta$ = longest interval during which a request elicits no response
  - assume $p_1$ crashes at $t_c$
  - any client request sent to $p_1$ at time $t_c - \delta$ or later may be lost
  - $p_2$ may not become the new primary until $t_c + \tau + 2\delta$
  - client may not learn that $p_2$ is new primary for another $\delta$

$$\Delta = \tau + 4\delta$$
Some like it hot

- **Hot** Backups process information from the primary as soon as they receive it.

- **Cold** Backups log information received from primary, and process it only if primary fails.

- Rollback Recovery implements cold backups cheaply:
  - The primary logs directly to stable storage the information needed by backups.
  - If the primary crashes, a newly initialized process is given content of logs—backups are generated “on demand.”