Welcome to CSCI 7000-001 Lec 5 (Jan 28)!

Recap:

1. Mathematical model of an asynchronous distributed system

   * Set of process/nodes: $P = \{P_1, P_2 \ldots P_n\}$

   * State/Configuration of a process p:

     (x_p, y_p, pc_p, st_p)

     - x_p: Input register. p's proposed value
     - y_p: output register. p's decision value
     - pc_p: p's program counter
     - st_p: Internal storage of p

   * State/Configuration of Entire System:

     $\langle (X, Y, PC, ST), msg\text{-}buf \rangle$

     - (X, Y, PC, ST): Functions mapping each p to its corresponding x/y/pc/st values
       eg: $X[p] = x_p$
     - $msg\text{-}buf$: Set of messages in flight (Captures the state of the network)
       message = (p, m)

   * Execution (Run) of the system (σ) ≅ Sequence of steps, where each step changes the configuration of the Entire system

Step : Config → Config

(P, m)

1. (Some process) P reads a msg m from msg-buf
2. P does internal computation
3. P adds finite number of messages to msg-buf

Represented as event $e = (P, m)$.

Hence an Execution σ of the system =

Series of steps / events $(e_1 . e_2 . \dots e_m)$

m could be ∞

* There can be many different executions of the system starting from an initial configuration depending on the order in which events occur.

Execution$_1$ = σ_1 = < process P_1 runs first, sends msg m_2 to P_2, ... >

Execution$_2$ = σ_2 = < process P_2 runs first, sends msg m_1 to P_2, ... >

* A theorem about the system = a property that is true for every execution starting from every initial config

---

FLP Impossibility

* A theorem about a system running a consensus protocol where one process is allowed to fail.

* What would be legal Executions (σ) of such a system?

Agreement:
1. Two different processes P_1 & P_2 cannot make two different decisions. In any config C that occurs during σ, it cannot be the case that $C.Y[P_1] \neq C.Y[P_2]$

Validity:
2. Every proposable value must be decidable. There must be some Execution σ_0 deciding a value 0, and $\exists \sigma_2$ deciding a value 1.

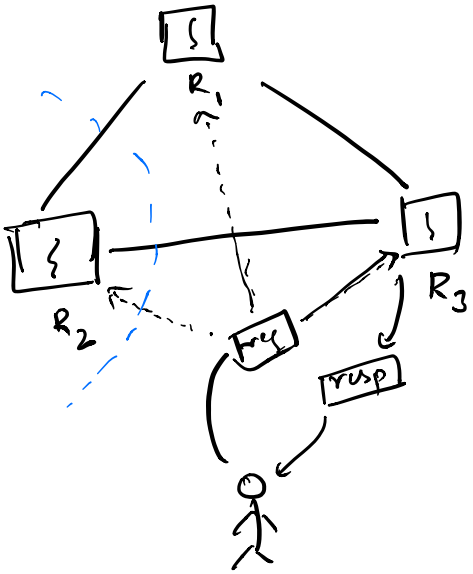
Robustness:
3. No more than one process must "fail".
E.g: If P_2 fails, then we won't see events of form (P_2, m) in σ, but every other process must take steps

FLP Impossibility = Given a legal deciding run (A decision is made) of the system, one can always construct a legal non-deciding run (No decision is made) that is infinitely long.

$\Leftrightarrow$

No consensus algorithm is guarenteed to terminate if we allow even one process to fail.

Today: CAP theorem & its repercussions



Webservice ⇒ Internet for communication

Asynchronous System

1. Each machine runs its own clock
2. No timing guarantees
3. Assumption: All messages are "eventually" delivered.

1. Consistency : Distributed Execution must be similar to a sequential Execution

Each user request must appear to atomically execute

Linearizability.

2. Availability : Every user request must receive a response.

3. Partition Tolerance : well-defined behaviour even when network partitions.

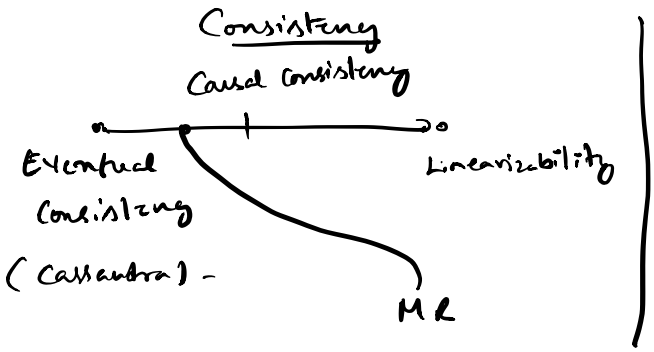
CAP Theorem

In an asynchronous distributed system, it is impossible to Simultaneously Ensure Consistency, Availability, & Partition Tolerance.

* ✓ ✓ : Cassandra, Riak, Dynamo
* C P : VoltDB.
* Ac : ----

→ WAN Systems : Given P : chose b/w A & C.

→



Availability

0% ——→ 100% available

Correctness of Execution

latency of requests

PACELC

PAC, ELC

Latency vs Consistency