CLOUD COMPUTING CONCEPTS

with Indranil Gupta (Indy)

PAXOS

Lecture B

CONSENSUS IN SYNCHRONOUS SYSTEMS

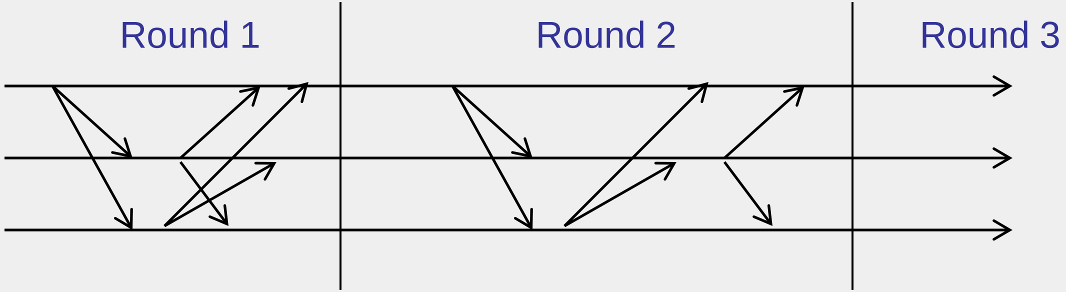
Let's Try to Solve Consensus!

- Uh, what' s the **system model**? (assumptions!)
- Synchronous system: bounds on
  - Message delays
  - Upper bound on clock drift rates
  - Max time for each process step

  e.g., multiprocessor (common clock across processors)
- Processes can fail by stopping (crash-stop or crash failures)

Consensus in Synchronous Systems

- For a system with at most f processes crashing
  - All processes are synchronized and operate in "rounds" of time
  - the algorithm proceeds in $f+1$ rounds (with timeout), using reliable communication to all members
  - $Values^r_i$: the set of proposed values known to p_i at the beginning of round r.



Consensus in Synchronous System

Possible to achieve!

- For a system with at most f processes crashing
  - All processes are synchronized and operate in "rounds" of time
  - The algorithm proceeds in $f+1$ rounds (with timeout), using reliable communication to all members
  - $Values^r_i$: the set of proposed values known to p_i at the beginning of round r.

- Initially $Values^0_i = \{\}$; $Values^1_i = \{v_i\}$

for round = 1 to $f+1$ do

multicast ($Values^r_i - Values^{r-1}_i$) // iterate through processes, send each a message

$Values^{r+1}_i \leftarrow Values^r_i$

for each V_j received

$Values^{r+1}_i = Values^{r+1}_i \cup V_j$

end

end

d_i = minimum($Values^{f+1}_i$)

Why does the Algorithm work?

- After $f+1$ rounds, all non-faulty processes would have received the same set of values. Proof by contradiction.
- Assume that two non-faulty processes, say p_i and p_j, differ in their final set of values (i.e., after $f+1$ rounds)
- Assume that p_i possesses a value v that p_j does not possess.
  - → p_i must have received v in the very last round
    - → Else, p_i would have sent v to p_j in that last round
  - → So, in the last round: a third process, p_k, must have sent v to p_i, but then crashed before sending v to p_j.
  - → Similarly, a fourth process sending v in the last-but-one round must have crashed; otherwise, both p_k and p_j should have received v.
  - → Proceeding in this way, we infer at least one (unique) crash in each of the preceding rounds.
  - → This means a total of $f+1$ crashes, while we have assumed at most f crashes can occur => contradiction.

Next

- Let’s be braver and solve Consensus in the Asynchronous System Model