Formal problem statement

• N processes
• Each process p has
  input variable \( x_p \): initially either 0 or 1
  output variable \( y_p \): initially b (can be changed only once)
• Consensus problem: design a protocol so that at the end, either:
  1. All processes set their output variables to 0 (all-0’s)
  2. Or All processes set their output variables to 1 (all-1’s)
**What is Consensus? (2)**

- Every process contributes a value
- *Goal is to have all processes decide same (some) value*
  - Decision once made can’t be changed
- There might be other constraints
  - Validity = if everyone proposes same value, then that’s what’s decided
  - Integrity = decided value must have been proposed by some process
  - Non-triviality = there is at least one initial system state that leads to each of the all-0’s or all-1’s outcomes
Many problems in distributed systems are equivalent to (or harder than) consensus!
- Perfect Failure Detection
- Leader election (select exactly one leader, and every alive process knows about it)
- Agreement (harder than consensus)

So consensus is a very important problem, and solving it would be really useful!

Consensus is
- Possible to solve in synchronous systems
- Impossible to solve in asynchronous systems
Can't we just solve Consensus?

- Yes, we can!
- (Whut?)
• Paxos algorithm
  – Most popular “consensus-solving” algorithm
  – Does not solve consensus problem (which would be impossible, because we already proved that)
  – But provides safety and eventual liveness
  – A lot of systems use it
    • Zookeeper (Yahoo!), Google Chubby, and many other companies

• Paxos invented by? (take a guess)
Yes we Can!

• Paxos invented by Leslie Lamport

• Paxos provides **safety** and **eventual liveness**
  – **Safety**: Consensus is not violated
  – **Eventual Liveness**: If things go well sometime in the future (messages, failures, etc.), there is a good chance consensus will be reached. But there is no guarantee.

• FLP result still applies: Paxos is not *guaranteed* to reach Consensus (ever, or within any bounded time)
Paxos has **rounds**; each round has a unique ballot id

Rounds are asynchronous

- Time synchronization not required
- If you’re in round $j$ and hear a message from round $j+1$, abort everything and move over to round $j+1$
- Use timeouts; may be pessimistic

Each round itself broken into phases (which are also asynchronous)

- Phase 1: A leader is elected (**Election**)
- Phase 2: Leader proposes a value, processes ack (**Bill**)
- Phase 3: Leader multicasts final value (**Law**)

**Political Science 101, i.e., Paxos Groked**
**Phase 1 – Election**

- Potential leader chooses a unique ballot id, higher than seen anything so far
- Sends to all processes
- Processes wait, respond once to highest ballot id
  - If potential leader sees a higher ballot id, it can’t be a leader
  - Paxos tolerant to multiple leaders, but we’ll only discuss 1 leader case
  - Processes also log received ballot ID on disk
- If a process has in a previous round decided on a value v’, it includes value v’ in its response
- If majority (i.e., quorum) respond OK then you are the leader
  - If no one has majority, start new round
- (If things go right) A round cannot have two leaders (why?)
Phase 2 – Proposal (Bill)

- Leader sends proposed value $v$ to all
  - use $v = v'$ if some process already decided in a previous round and sent you its decided value $v'$
  - If multiple such $v'$ received, use latest one
- Recipient logs on disk; responds OK

Please elect me!

Value $v$ ok?

OK!
Phase 3 - Decision (Law)

- If leader hears a majority of OKs, it lets everyone know of the decision.
- Recipients receive decision, log it on disk.
That is, when is consensus reached in the system
**Which is the point of No-Return?**

- If/when a majority of processes hear proposed value and accept it (i.e., are about to/have respond(edi) with an OK!)
- Processes *may not know it yet*, but a decision has been made for the group
  - Even leader does not know it yet
- What if leader fails after that?
  - Keep having rounds until some round completes

```
Please elect me!

OK!

Value v ok?

OK!

v!
```
Safety

- If some round has a majority (i.e., quorum) hearing proposed value $v'$ and accepting it, then subsequently at each round either: 1) the round chooses $v'$ as decision or 2) the round fails

- Proof:
  - Potential leader waits for majority of OKs in Phase 1
  - At least one will contain $v'$ (because two majorities or quorums always intersect)
  - It will choose to send out $v'$ in Phase 2

- Success requires a majority, and any two majority sets intersect
What could go Wrong?

- Process fails
  - Majority does not include it
  - When process restarts, it uses log to retrieve a past decision (if any) and past-seen ballot ids. Tries to know of past decisions.
- Leader fails
  - Start another round
- Messages dropped
  - If too flaky, just start another round
- Note that anyone can start a round any time
- Protocol may never end – tough luck, buddy!
  - Impossibility result not violated
  - If things go well sometime in the future, consensus reached
A lot more!

This is a highly simplified view of Paxos.

See Lamport’s original paper:
**Summary**

- Paxos protocol: widely used implementation of a safe, eventually-live consensus protocol for asynchronous systems
  - Paxos (or variants) used in Apache Zookeeper, Google’s Chubby system, Active Disk Paxos, and many other cloud computing systems