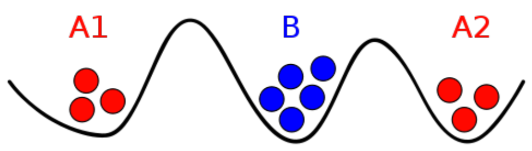
Intro to consensus

A simple form of consensus: the "two-generals problem"

Two armies want to attack a fortified city. Both armies need to attack at the same time in order to succeed. The armies can only communicate through messengers, but unfortunately, the messengers can be captured, so message delivery is not reliable.



A solution must have three properties:

- **Consistency:** both armies decide to attack at the same time
- **Validity:** the time to attack was proposed by one of the armies
- **Termination:** each army decides to attack after a finite number of messages

Unfortunately, this is provably impossible.

Sketch of proof: if there is a finite sequence of messages, there is a final message that causes consensus. But, that message could be lost, and therefore consensus cannot depend on it. (Every message must be acknowledged, but every acknowledgement must be acknowledged, etc.)

Different flavors of consensus

There are a number of problems that you might find yourself trying to solve in a distributed system, all of which are in effect equivalent. (i.e., you can solve one using a solution to another.)

Consensus

- collection of processes, Pi, each begins in an undecided state
- processes propose values Vi, communicate via messages to exchange proposals
- each process sets a decision variable Di and enters a stable "decided state"

Requirements:

- **consistency:** the decision value of all correct processes is the same
- **validity:** if correct processes all propose the same value, then any correct process in the decided state will have decided on that value
- **termination:** eventually all correct processes set their decision variables

Byzantine generals problem

- commander issues an order, lieutenants receive order and must agree on it

Requirements:

- **consistency:** the decision value of all correct processes is the same
- **validity:** if the commander is correct, then any correct lieutenant decides on the order that the commander issues
- **termination:** eventually all correct lieutenants set their decision variables

Leader election

- collection of nodes, none of which is yet the "leader"
- nodes exchange messages, proposing leadership
- nodes set a decision variable, deciding which single node is the leader

Requirements:

- **consistency:** all correct processes choose the same leader
- **validity:** if all correct processes propose the same leader, that leader is elected
- **termination:** eventually all correct processes set their decision variables

Reliable and totally ordered multicast (ABCAST)

- collection of nodes, each of which can multicast a message to all nodes
- multicast messages are delivered to a node in a certain order

Requirements

- **consistency**: all processes deliver all messages in the same order
- **validity**: something about preserving each nodes' transmission order?
- **termination**: eventually all messages are delivered to all processes

Amazingly, each of these problems is equivalent.

e.g., implementing consensus, given ABCAST:

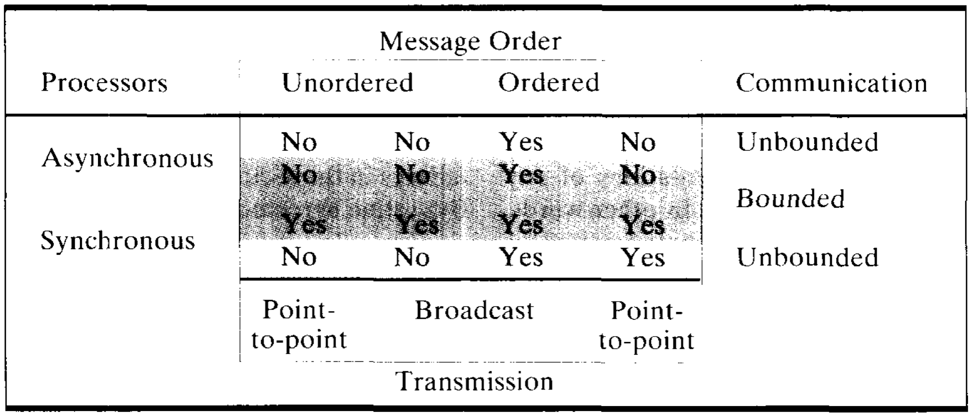
- each processes making a proposal uses ABCAST to send the proposal
- the group chooses the first proposal that is delivered

Pop quiz: how do you implement reliable and totally ordered multicast, given a solution to the consensus problem?

Is consensus possible?

It turns out that consensus is possible in some circumstances, and impossible (not guaranteed to reach all the requirements) in others. There are several axes:

- Processors: synchronous vs. asynchronous
  - o bounded ratio of rate at which processors make forward progress
- Communication delay: bounded vs. unbounded
  - o unbounded means messages have finite but unbounded delivery latency
- Ordered vs. unordered messages
- Broadcast vs. point-to-point messages
- Failures: fail-stop vs. byzantine; we'll focus only on fail-stop for now



Processors	Message Order	Unordered	Unordered	Ordered	Ordered	Communication
	Transmission	Point-to-point	Broadcast	Broadcast	Point-to-point	
Asynchronous		No	No	Yes	No	Unbounded
Asynchronous		No	No	Yes	No	Bounded
Synchronous		Yes	Yes	Yes	Yes	Bounded
Synchronous		No	No	Yes	Yes	Unbounded

Really two cases we will focus on:

- asynchronous system: messages are unordered, communication delay is unbounded, and processors are asynchronous
- synchronous system: messages are unordered, communication delay is bounded, processors are synchronous

The infamous impossibility result: FLP (Fischer, Lynch, Paterson '85)

- **In an asynchronous system, no protocol can guarantee consensus <u>within a finite amount of time</u> if even a single process can fail by stopping**

(Doesn't mean that consensus won't be achieved – just that it's not guaranteed.)