Coordination

Consensus & Agreement

- It is generally important that the processes within a distributed system have some sort of agreement
- Coordination among multiple parties involves agreement among those parties
- Agreement $\iff$ Consensus $\iff$ Consistency
- Agreement is difficult in a dynamic asynchronous system in which processes may fail or join/leave

Impossibility Theorems

- Two fundamental theorems, FLP and CAP, influences the system design choices

- FLP theorem: asynchronicity vs synchronicity

  Consensus is impossible to implement in such a way that it both a) is always correct and b) always terminates if even one machine might fail in an asynchronous system with crash fault failures

- CAP theorem: what happens when network partitions are included in the failure model

  You can't implement consistent storage and respond to all requests if you might drop messages between processes.

FLP

- **Impossibility of Distributed Consensus with One Faulty Process**, by Fischer, Lynch and Paterson (1985)
- **Consensus Problem**: we have a set of processes, each one with a private input; the processes communicate; the processes must agree on on some process's input.

Consensus is important

- With consensus we can implement anything we can imagine:
  - leader decision
  - mutual exclusion
  - transaction commitment
  - much more…
- In some models consensus is possible, in some other models, it is not
- The goal is to learn whether, for a given model, consensus is possible or not… and prove it!

(Wrong) Consensus Protocol

- Model:
  - n > 1 processes
  - shared memory (may be accessed simultaneously by multiple processes)
  - processors can atomically *read* and *write* (not both) a shared memory location
- Protocol:
  - There is a specific memory location *C*
  - Initially *C* is in a special state $\perp$
  - Processor *1* writes its value v_1 into *C*, then decides on v_1
  - Processors $j \neq 1$ read *C* until they read something else than $\perp$ and then decide on that
- Problems with this protocol?

Consensus Properties

1. Agreement: Every correct process must agree on the same value.

2. Integrity: Every correct process decides at most one value, and if it decides some value, then it must have been proposed by some process.

3. Termination: All correct processes eventually reach a decision.

4. Validity: If all correct processes propose the same value V, then all correct processes decide V.

FLP System Model

- Asynchronous communication model, i.e., no upper bound on the amount of time processors may take to receive, process and respond to an incoming message

- Communication links between processors are assumed to be reliable. It is well known that given arbitrarily unreliable links no solution for consensus could be found even in a synchronous model.

- Processors are allowed to fail according to the crash fault model – this simply means that processors that fail do so by ceasing to work correctly. There are more general failure models, such as byzantine failures where processors fail by deviating arbitrarily from the algorithm they are executing.

Notation (I)

- There are *$N > 1$* processors which communicate by sending messages.
- A message is a pair *(p,m)* where *p* is the processor the message is intended for, and *m* is the contents of the message.
- Messages are stored in an abstract data structure called the <u>message buffer</u> which is a multiset – simply a set where more than one of any element is allowed – which supports two operations, *send* and *receive*.
- *send(p,m)* simply places the message *(p,m)* in the message buffer.
- *receive(p)* either returns a (random) message for processor *p* (and removes it from the message buffer) or the special value ∅, which does nothing.

Notation (II)

- Configuration: the internal state of all of the processors – the current step in the algorithm that they are executing and the contents of their memory – together with the contents of the message buffer.
- Step: the system moves from one configuration to the next by a step which consists of a processor *p* performing *receive(p)* and moving to another configuration, i.e.:
    - based on *p* local state and *m*, send an arbitrary but finite number of messages
    - based on *p* local state and *m*, change *p* local state to some new state
- Event: each step is therefore uniquely defined by the message that is received (possibly ∅) and the process *p* that received it. That pair is called an *event* (equivalent to a message)
    - Configurations move from one to another through events.
    - An event *e* can be applied to a configuration *C* if either *m* is ∅ or *(p,m)* is in the message buffer
    - *C' = e(C)* means that if we apply event *e* to configuration *C* we move to configuration *C'*
- Execution: a possibly infinite sequence of events from a specific initial configuration.
    - Since the receive operation is non-deterministic, there are many different possible executions for a given initial configuration.

Notation (III)

- Schedule & Run: a particular execution σ, defined by a possibly infinite sequence of events from a starting configuration, is called a *schedule* and the sequence of steps taken to realize the schedule is a *run*.
  - Non-faulty processes take infinitely many steps in a run (presumably eventually just receiving $\varnothing$ once the algorithm has finished its work) – otherwise a process is considered faulty.
  - σ can be applied to configuration C if the events in σ can be applied to C in order
  - $C' = \sigma(C)$ means that if we apply schedule σ to configuration C we move to configuration C'
- An admissible run is one where at most one process is faulty (capturing the failure requirements of the system model) and every message is eventually delivered (this means that every processor eventually gets chosen to receive infinitely many times).
- We say that a run is a deciding run provided that some process eventually decides according to the properties of consensus, and that a consensus protocol is totally correct if every admissible run is a deciding run.

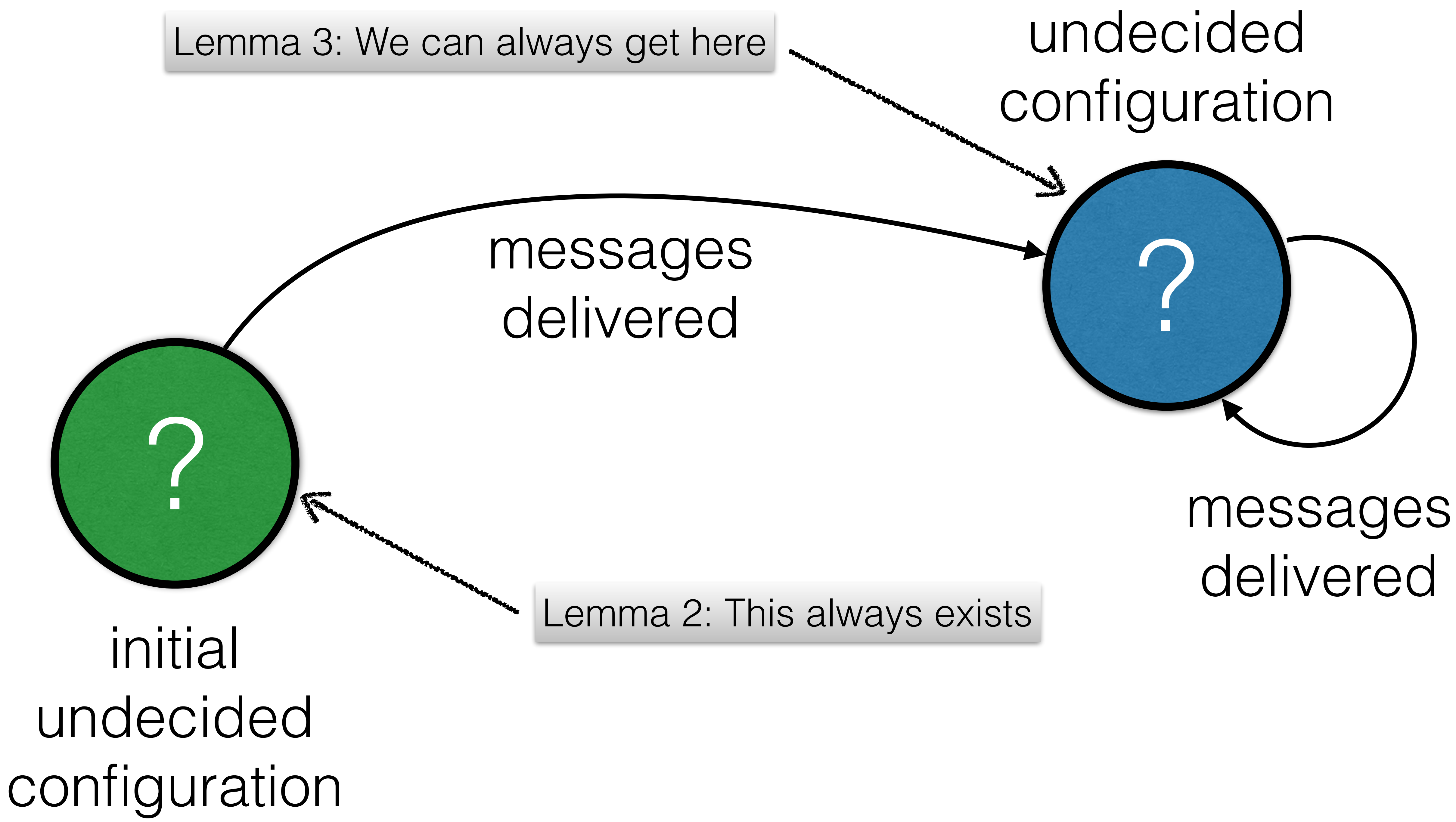
Proof Sketch

- FLP Theorem [1985]. No totally correct consensus algorithm exists (for the given system model).

- The idea behind it is to show that there is *some admissible run* – i.e., one with only one processor failure and eventual delivery of every message – *that is not a deciding run* – i.e., in which no processor eventually decides and the result is a protocol which runs forever (because no processor decides).

- Two processors and binary consensus values

Proof Sketch


Lemma 3: We can always get here
undecided configuration
?
messages delivered
?
messages delivered
Lemma 2: This always exists
initial undecided configuration