Review of Proofs of Paxos and Byzantine Consensus and (time permitting introduction to block chain)
Specification of Paxos

Safety

1. Learners learn at most one value.
   Associated with learner z is a variable learned[z] where learned[z] is either None (representing undefined) or some non-None value.
   For v != None: stable(learned[z] = v)

2. Learners learn only proposed values
   (learned[z] != None) => learned[z] in set of proposals

3. All learners learn the same unchanging value
   For all learners z, z’: (learned[z]=None) or (learned[z’]=None) or (learned[z]=learned[z’])
Proposers, Acceptors, Learners
Algorithm: Phase 1, Step 1
proposers send prepare messages

1. **When triggered**, a proposer sends a $prepare(n)$ message to at least a majority of acceptors where:
   
   
   $$n > \text{latest}_n$$
   
   and makes $\text{latest}_n = n$

   where $\text{latest}_n$ is a local variable of the proposer
Algorithm: Phase 1, Step 1
proposers send prepare messages

1. **When triggered**, a proposer sends a \textit{prepare}(n) message to at least a majority of acceptors where:
   \[ n > \text{latest}_n \]
   and makes \text{latest}_n = n
   where \text{latest}_n is a local variable of the proposer

2. **When an acceptor \text{y} receives a \textit{prepare}(n) message** from a proposer \text{x}:
   if \( n > \text{highest}_\text{prepare}_n \):
   send \text{granted}(n, \text{latest}_\text{accepted}, \text{latest}_\text{accepted}_n) to \text{x}
   \text{highest}_\text{prepare}_n = n

Local variables of acceptor
Phase 2, Step 1
proposers send accept messages

A proposer sends an accept message if it receives a granted message from a majority of acceptors where n in each of these granted messages equals latest_n

Recall format: granted(n, latest_accepted, latest_accepted_n)

An accept message is (latest_n, my_proposal) where my_proposal is as follows:

• **Proposer’s choice**: If each granted message has latest_accepted = None then my_proposal is any value.

• **Acceptor’s choice**: else: my_proposal is the latest_accepted in a granted message with the highest latest_accepted_n.
  
  — **IMPORTANT**: proposer is forced to choose the proposal accepted by some acceptor
Phase 2, Step 2
acceptors send accepted messages to learners

if acceptor receives an
   accept(latest_n, my_proposal) message
from a proposer where:
   latest_n >= latest_prepare_n
then acceptor accepts proposal v and the acceptor sends an
accepted(my_proposal) message to all learners and makes
   latest_accepted = my_proposal
   latest_accepted_n = latest_n
Proof of Safety: Central Idea

If a majority of acceptors have accepted the same proposal then, from then on, that majority does not accept a different proposal.

Lemma: for v != None and any set S of a majority of acceptors:
stable(for all acceptors p in S: latest_accepted = v)
Proof of Safety: 1

Lemma: for \( v \neq None \) and any set \( S \) of a majority of acceptors:

\[
\text{stable(} \forall \text{acceptors } p \in S \colon \text{latest}\_\text{accepted} = v \text{)}
\]

Proof:
Recall what stable means: for a predicate \( Q \), \( \text{stable}(Q) \) holds when
For all actions \( a \): \( \{Q\} a \{Q\} \)

So, we need to prove, for all actions \( a \):

\[
\{\forall \text{acceptors in } S \colon \text{latest}\_\text{accepted} = v\} a \{\forall \text{acceptors in } S \colon \text{latest}\_\text{accepted} = v\}
\]
Proof of Safety: 2

We need to prove, for all actions $a$:

$\{\text{for all acceptors in } S: \text{latest\_accepted} = v\}$

What actions can change latest\_accepted?
Proof of Safety:3

We need to prove, for all actions a:
\[ \{\text{for all acceptors in } S: \text{latest\_accepted} = v\} \]

\[ a \]
\[ \{\text{for all acceptors in } S: \text{latest\_accepted} = v\} \]

What actions can change latest\_accepted? Only this action:
For latest\_n >= highest\_prepare\_n
\[ \{\text{latest\_accepted} = v\} \]
receive accept (latest\_n, my\_proposal)
\[ \{\text{latest\_accepted} = my\_proposal\} \]
Proof of Safety: 4

We need to prove, for all actions \( a \):

\[
\{\text{for all acceptors in } S: \text{latest} \_\text{accepted} = v}\}
\]

\( a \)

\[
\{\text{for all acceptors in } S: \text{latest} \_\text{accepted} = v}\}
\]

What actions can change \( \text{latest} \_\text{accepted} \)? Only this action:

For \( \text{latest} \_n \geq \text{highest} \_\text{prepare} \_n \)

\[
\{\text{latest} \_\text{accepted} = v}\}
\]

receive accept (\( \text{latest} \_n, \text{my} \_\text{proposal} \))

\[
\{\text{latest} \_\text{accepted} = \text{my} \_\text{proposal}\}
\]

Prove:

\[
\text{my} \_\text{proposal} = v
\]
Proof of Safety: 5

For an acceptor p in S if latest_n > highest_prepare_n:

{ (for all acceptors in S: latest_accepted = v) }

receive the message: accept (latest_n, my_proposal)

{ my_proposal = v }

This is why we need “acceptor’s choice”: the proposer is forced to make my_proposal the same as latest_accepted, where latest_accepted is received in the granted message from an acceptor: granted(n, latest_accepted, latest_accepted_n)
Proof of Safety: 6

Proposer sends `accept (latest_n, my_proposal)` only if it received `granted(n, latest_accepted, latest_accepted_n)`, where \( n = \text{latest}_n \), from a majority of acceptors.

At least one of these granted messages must be sent by an acceptor in \( S \) (since \( S \) is a majority). Let’s call this acceptor \( q \).

\( q \) sent `granted(latest_n, latest_accepted, latest_accepted_n)` in reply to a `prepare(latest_n)` message from proposer \( p \).

By acceptor’s choice rule: proposer’s `my_proposal` is equal to acceptor’s `last_accepted`.
Come up with a scenario that does not progress without reading next slide

prepare(1,0)

Like (time, id)

prepare(1,1)
Scenario without progress

1. Proposer 0 sends prepare([1,0])
2. Acceptor replies granted([1,0], -, -)
3. Before proposer 0’s accept message gets to the acceptor it gets prepare([1, 1]) from proposer 1.
4. Acceptor replies granted([1, 1], _, _)
5. Before proposer 1’s accept message gets to the acceptor it gets prepare([2, 0]) from proposer 0
6. Acceptor replies granted([2,0], -, -)
7. ...............
Ideas for Best Effort for Progress

• Can’t guarantee progress.
• But, can reduce the likelihood of proposers interfering with each other for ever.
• Prove progress with a single proposer that makes only one proposal.
• Come up with ideas by which one proposer eventually becomes faster than the others so that other proposers don’t interfere.
Review of Byzantine Consensus: 1

- Only message type: attack.
- Treat retreat and no message as equivalent.

Case non-faulty general sends attack on round 1.
- All non-faulty officers commit to attack on round 1.

Case non-faulty general never sends attack messages
- Non-faulty officers never commit to attack because they only commit to attack if they have an attack message from the general.
Initially a non-faulty officer is uncommitted. A non-faulty officer switches from uncommitted to committed (to attack) on round r if and only if it has:

- Commit messages from at least r officers
- At least one of them is from the general.

When it switches to commit, it sends copies of the commit messages it received plus its own commit message for a total of r+1.
Review of Byzantine Consensus: 3

- Lemma: If any non-faulty officer commits on round $r$ then all non-faulty officers commit on round $r+1$.
- Lemma: If a non-faulty officer has committed by round $t$ then all non-faulty officers commit by round $t+1$.
- Lemma: If no non-faulty officer has committed on round $t$ then no non-faulty officer commits on round $t+1$ (where $t$ is the maximum number of faulty officers).
- Theorem: Consensus among non-faulty officers reached by round $t+1$. 
Introduction to Distributed Ledger

• If Bob gives Alice $10 in cash, then (assuming rationality) Bob and Alice have common knowledge that this transaction occurred, i.e., Bob knows that Alice knows that Bob knows that…. Bob gave Alice $10.

• If Bob gives Alice a check for $10, then after the check clears, Bob’s bank and Alice’s bank have common knowledge that the transaction occurred.

• Develop a distributed algorithm to verify that a transaction occurs without a trusted intermediary (bank) or intermediaries (banks, credit cards….)
A block chain is a protocol that enables peer-to-peer exchanges in a distributed network in a secured way where an exchange cannot be repudiated.

(The bitcoin block chain is also public, but other distributed ledger technologies need not be.)

Why is there so much excitement (hype?) and money invested in startups working on distributed ledgers? Don’t need to pay trusted intermediaries.
To begin with, restrict attention to a single indivisible token that can be transferred among agents.

Requirement: Eventual Consensus – All participants agree about the history of the token: who had it first, who got it next, and ...

Requirement: The history shows that the token is indeed indivisible. Bob can’t pay both Alice and Mindy with the same $10.

Both parties in a transaction agree with the history. If Bob pays Alice $10 then both Bob and Alice agree that this transaction occurred. Alice cannot repudiate.
Block Chain

• Introduction to technologies
  – Public and private keys
  – Fingerprints and one-way functions
  – Blocks

• Next Class: Detailed description of block chain