3-Phase Commit

Coordinator $c$

1. sends VOTE-REQ to all participants

3. if (all votes are Yes) then send Precommit to all else
   $decision_c :=$ Abort
   send Abort to all who voted Yes
   halt

5. collect Ack from all participants
   When all Ack’s have been received:
   $decision_c :=$ Commit
   send Commit to all

Participant $p_i$

2. sends $vote_i$ to Coordinator
   if $vote_i = No$ then
     $decision_i :=$ Abort
     halt

4. if received Precommit then
   send Ack

6. When $p_i$ receives Commit, sets $decision_i :=$ Commit and halts
## Timeout Actions

<table>
<thead>
<tr>
<th>Coordinator  (c)</th>
<th>Participant  (p_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 2: (p_i) is waiting for VOTE-REQ from the coordinator</td>
<td></td>
</tr>
<tr>
<td>Step 3: Coordinator is waiting for vote from participants</td>
<td></td>
</tr>
<tr>
<td>Step 4: (p_i) is waiting for Precommit</td>
<td></td>
</tr>
<tr>
<td>Step 5: Coordinator is waiting for Ack's</td>
<td></td>
</tr>
<tr>
<td>Step 6: (p_i) is waiting for Commit</td>
<td></td>
</tr>
</tbody>
</table>
Timeout actions

Coordinator $c$

Step 3: Coordinator is waiting for vote from participants

Participant $p_i$

Step 2: $p_i$ is waiting for VOTE-REQ from the coordinator

Same as in 2PC

Step 4: $p_i$ is waiting for Precommit

Step 5: Coordinator is waiting for Ack's

Step 6: $p_i$ is waiting for Commit
Timeout actions

Coordinator $c$

Step 2: $c$ is waiting for VOTE-REQ from the coordinator

Same as in 2PC

Step 3: Coordinator is waiting for vote from participants

Same as in 2PC

Step 5: Coordinator is waiting for Ack’s

Participant $p_i$

Step 2: $p_i$ is waiting for VOTE-REQ from the coordinator

Same as in 2PC

Step 4: $p_i$ is waiting for Precommit

Step 6: $p_i$ is waiting for Commit
TIMEOUT ACTIONS

Coordinator \( c \)

Step 3: Coordinator is waiting for vote from participants
Same as in 2PC

Participant \( p_i \)

Step 2: \( p_i \) is waiting for VOTE-REQ from the coordinator
Same as in 2PC

Step 4: \( p_i \) is waiting for Precommit
Run termination protocol

Step 5: Coordinator is waiting for Ack’s

Step 6: \( p_i \) is waiting for Commit
### Timeout Actions

<table>
<thead>
<tr>
<th>Coordinator $c$</th>
<th>Participant $p_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1:</strong> $c$ is waiting for $VOTE-REQ$ from the coordinator.</td>
<td>Same as in 2PC</td>
</tr>
<tr>
<td><strong>Step 2:</strong> $p_i$ is waiting for $VOTE-REQ$ from the coordinator.</td>
<td>Same as in 2PC</td>
</tr>
<tr>
<td><strong>Step 3:</strong> Coordinator is waiting for vote from participants.</td>
<td><strong>Step 4:</strong> $p_i$ is waiting for Precommit</td>
</tr>
<tr>
<td><strong>Step 5:</strong> Coordinator is waiting for Ack’s. Coordinator sends Commit</td>
<td>Run termination protocol</td>
</tr>
<tr>
<td><strong>Step 6:</strong> $p_i$ is waiting for Commit</td>
<td></td>
</tr>
</tbody>
</table>
**TIMEOUT ACTIONS**

**Coordinator** $c$

Step 2: $p_i$ is waiting for VOTE-REQ from the coordinator

Same as in 2PC

Step 3: Coordinator is waiting for vote from participants

Same as in 2PC

Step 5: Coordinator is waiting for Ack's

Coordinator sends **Commit**

**Participant** $p_i$

Step 2: $p_i$ is waiting for VOTE-REQ from the coordinator

Same as in 2PC

Step 4: $p_i$ is waiting for **Precommit**

Run termination protocol

Step 6: $p_i$ is waiting for **Commit**

Run termination protocol
Timeout actions

**Coordinator** $c$

Step 2: $c$ is waiting for VOTE-REQ from the coordinator

Same as in 2PC

Step 3: Coordinator is waiting for vote from participants

Same as in 2PC

Step 5: Coordinator is waiting for Ack's

Participant knows what they will receive…

but the NB property can be violated!

**Participant** $p_i$

Step 2: $p_i$ is waiting for VOTE-REQ from the coordinator

Same as in 2PC

Step 4: $p_i$ is waiting for Precommit

Run termination protocol

Step 6: $p_i$ is waiting for Commit

Run termination protocol
**Termination protocol: Process states**

At any time while running 3PC, each participant can be in exactly one of these four states:

- **Aborted**: Not voted, voted **No**, received **Abort**
- **Uncertain**: Voted **Yes** but not received **Precommit**
- **Pre-committed**: Received **Precommit**, not **Commit**
- **Committed**: Received **Commit**
**Not all states are compatible**

<table>
<thead>
<tr>
<th></th>
<th>Aborted</th>
<th>Uncertain</th>
<th>Pre-committed</th>
<th>Committed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aborted</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Uncertain</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Pre-committed</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Committed</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
When $p_i$ times out, it starts an **election protocol** to elect a new coordinator.

- The new coordinator sends STATE-REQ to all processes that participated in the election.
- The new coordinator collects the states and follows a set of **termination rules**.
The new coordinator collects the states and follows a set of **termination rules**

- **TR1**: if some process decided **Abort**, then decide **Abort**
  - send **Abort** to all
  - halt

- **TR2**: if some process decided **Commit**, then decide **Commit**
  - send **Commit** to all
  - halt

- **TR3**: if all processes that reported state are uncertain, then decide **Abort**
  - send **Abort** to all
  - halt

- **TR4**: if some process is pre-committed, but none committed, then send **Precommit** to uncertain processes
  - wait for **Ack**'s
  - send **Commit** to all
  - halt
Termination Protocol and Failures

Processes can fail while executing the termination protocol

- if $c$ times out on $p$, it can just ignore $p$
- if $c$ fails, a new coordinator is elected and the protocol is restarted (election protocol to follow)
- total failures will need special care
Recovering $p$

- If $p$ fails before sending Yes, decide **Abort**
- If $p$ fails after having decided, follow decision
- If $p$ fails after voting Yes, but before receiving decision value
  - $p$ asks other processes for help
  - 3PC is non-blocking: $p$ will receive a response with the decision
- If $p$ has received **Precommit**
  - still needs to ask other processes (cannot just **Commit**)

No need to log **Precommit!**
(or is there?)
THE ELECTION PROTOCOL

- Processes agree on linear ordering (e.g. by pid)
- Each process $p$ maintains a set $UP_p$ of all processes that it believes to be operational
- When $p$ detects failure of $c$, it removes $c$ from $UP_p$ and chooses smallest $q$ in $UP_p$ to be the new coordinator
- If $p = q$, then $p$ is the new coordinator
- Otherwise, $p$ sends UR-ELECTED to $q$
Total failure

Suppose that $p$ is the first process to recover and that $p$ is uncertain. Can $p$ decide Abort?

Some process could have decided Commit after $p$ crashed!

$p$ is blocked until some process $q$ recovers such that either

- $q$ can recover independently
- $q$ is the last process to fail: then $q$ can simply invoke the termination protocol
Determining the last process to fail

Suppose a set $R$ of processes has recovered

Does $R$ contain the last process to fail?

- the last process to fail is in the $UP$ set of every process
- so the last process to fail must be in

$$\bigcap_{p \in R} UP_p$$

$R$ contains the last process to fail if:

$$\bigcap_{p \in R} UP_p \subseteq R$$
Administrivia

- I will email you homework #1 later today
  - Due next Monday 9/27 before class by email to Tony and me
- Research project
  - Declare your team by Oct 1st (by email to me)
  - Declare your topic by Oct 8th (by email to me)
- Not sure what to do? Come talk to me.