2PC and Blocking

- No AC protocol is non-blocking if communication failures or total failures
- But 2PC can block even with non-total failures!
- Can we do better?

3-Phase Commit

New protocol, 2 versions:

**Version 1**
- Tolerates only site failures (communication failures may produce inconsistencies)
- Non-blocking, unless total failures

**Version 2**
- Tolerates also communication failures
- Blocks even if less than total failure (although less often than 2PC)

- In practice, people use 2PC…

Structure of 3PC

3PC is designed to satisfy the following property:

*Non-blocking Property:* If any operational process is uncertain, then no process has decided COMMIT

- We consider only Version 1 of the protocol.

2PC Revisited

In $U$ can reach both $A$ and $C$!
3PC: The Protocol

- Coordinator c sends VOTE-REQ to all participants.
- When participant p receives a VOTE-REQ, it responds by sending a vote to the coordinator. If vote = NO, then decide = ABORT and p halts.
- Coordinator c collects votes from all. If all votes are YES, then sends PRECOMMIT to all other. Decide = ABORT and sends ABORT to all participants who voted YES. Coordinator c halts.
- If participant p receives PRECOMMIT then sends ACK to coordinator.
- Coordinator collects ACKs from all. When all ACKs have been received, decide = COMMIT; sends COMMIT to all. When participant p receives COMMIT, sets decide = COMMIT and halts.

Wait a minute!

- Messages are known to the receiver before even being sent!
- So, why sending them?
- They inform the recipient of the protocol’s progress!

- When c receives ACK from p, it know p is not uncertain.
- When p receives COMMIT, it knows no participant is uncertain, so it can commit.

Timeout Actions

Processes wait at the beginning of steps 2, 3, 4, 5, and 6.

- Step 2: Participant waits for VOTE-REQ from coordinator.
- Step 3: Coordinator waits for vote from participants.
- Step 4: Participant waits for PRECOMMIT.
- Step 5: Coordinator waits for ACKs.
- Step 6: Participant waits for COMMIT.

Is communication necessary?
- Participant knows what is going to receive...
- Can NB property be violated?
Termination Protocol - I

At any time while running 3 PC, each participant can be in exactly one of these 4 states:

- **Aborted**  Not voted, voted NO, received ABORT
- **Uncertain**  Voted YES, not received PRECOMMIT
- **Committable**  Received PRECOMMIT, not COMMIT
- **Committed**  Received COMMIT

TP-II:
Not all states are compatible…

<table>
<thead>
<tr>
<th>Aborted</th>
<th>Uncertain</th>
<th>Committable</th>
<th>Committed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Aborted</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Uncertain</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Committable</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Committed</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

TP-III

- When participant times out, it starts an election protocol to elect new coordinator
- New coordinator sends STATE-REQ to all processes that participated in election
- Coordinator collects state, and uses
  
  **Termination Rule**

TP-IV

Processes can fail during Termination Protocol…

- If coordinator times out on a participant $p$, then it will ignore $p$
- If coordinator fails, a new coordinator is elected and the protocol is restarted

What about total failures?
**Recovery**

Only for participant (coordinator is similar)
- If $p$ fails before sending YES, decide ABORT
- If $p$ fails after having decided, no problem
- If $p$ fails after voting YES but before receiving decision value
  - ask 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 decide COMMIT)

No need to log PRECOMMIT messages… nothing new!

**Election Protocol**

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

- What if $p'$, which has not detected $c$’s failure, receives a STATE-REQ from $q$?
  - Considers $c$ faulty
  - Removes from $U_{P'}$ every $q' < q$

- What if $p'$, which has changed to $q$ as coordinator, receives a STATE-REQ from $c$?
  - It ignores the request