Let $A_0$ be a set of some $f$ processes

Let $A_1$ be a set of $f + 1$ processes that includes sender

For $b = 0, 1$, a well-formed message received by a process $p \in A_b$ is a message that is delivered to $p$ at phase $\ell =$, has $\ell$ mutually distinct signatures $p_1, \ldots, p_\ell$ such that

1. $p_1$ is sender

2. $p_2, \ldots, p_\ell$ are alternating members of $A_0$ and $A_1$, and none is $p$ itself

3. $p_\ell \in A_{\neg b}$
The Algorithm

- At the first phase, if sender’s value is 1, it signs it and sends it to all processes in $A_0 \cup A_1$

- A process $p \in A_b$ that receives its first well-formed message at phase $k \leq f + 1$, signs is and sends it to $A_{\overline{b}}$ at phase $k + 1$. Processes in $A_1$ save the set of wf msgs they (first) receive

- At phase $f + 3$, all processes in $A_1$ send their saved msgs to all passive processes

- An active process that receives a wf msg by phase $f + 2$ decides 1, and 0 otherwise

- A passive process that receives $f + 1$ signatures on 1, decide 1. Else it decides 0
2-Phase Commit (2PC)

Assume no failures!

- **Round 1:** The coordinator sends **VOTE-REQ**.
- Every process that receives **VOTE-REQ** prepares an answer; if it’s **NO**, it decides **ABORT** and stops.

- **Round 2:** All whose vote is **YES** send it to the coordinator.
- If the coordinator gets (**YES**) votes from all, and its vote is **YES**, it decides **COMMIT**; else it decides **ABORT**.

- **Round 3:** The coordinator sends its decision to all.
- Everyone that receives the coordinator’s decision decides on the same value (and stops.)
Why “2PC”? 

The two phases are:

1. the voting phase (rounds 1-2)
2. the decision phase (round 3, or 2-3 for coordinator)

- processes are uncertain between the end of round 1 and the end of round 3
- coordinator is never uncertain
- AC1–AC4 are trivially satisfied
- As to AC5 . . .
Towards AC5 – Timeouts

Time-outs are possible at:

- **Round 1**: if a process awaiting \texttt{VOTE-REQ} time-outs, can decide \texttt{ABORT}.

- **Round 2**: if the coordinator times-out while awaiting votes, it can just decide \texttt{ABORT} (and send \texttt{ABORT} to every \texttt{YES}-voter.)

- **Round 3**: if a process ($p$) times-out while uncertain, it cannot unilaterally decide; it must invoke a \textsc{termination protocol}. 
The Termination Protocol

- If \( p \) waits until comm. w/ coordinator established
- AC5 is met, but possibly takes too long
- \( p \) can learn decision from \( q \) who knows it
- Then, if \( q \) gets \( p \)’s request, it
  1. sends it decision (if it knows it), OR
  2. if it hasn’t yet voted, decide \textit{ABORT}s, OR
  3. is just not be able to help \( p \)

Worst case is if \( p \) fails to communicate with any “certain” process. Then it just waits for resumption of communication with coordinator.
Towards AC5 – Recovery

- If when \( p \) recovers it remembers its pre-failure state, and this state is not “uncertain”, \( p \) can recover independently.

- Otherwise, \( p \) cannot distinguish between the scenarios:
  1. all others voted \text{YES}, decision is \text{COMMIT};
  2. some voted \text{NO} (or not voted), decision is \text{ABORT}.

- \( p \) is exactly like a process that times-out at round 3.

- To remember its state, \( p \) keeps a DT log in a \text{failure-resistant memory}. 
Coordinator 2PC with DT log

send VOTE-REQ to all *
write START-2PC on DT log *
await YES/NO from all
on timeout {
  \(P_Y := \) set of processes who voted YES
  write ABORT in DT log; *
  send ABORT to every process in \(P_Y\) *
  return }
if all and coordinated voted YES {
  write COMMIT on DT log
  send COMMIT to all }
else {
  \(P_Y := \) set of processes who voted YES
  write ABORT on DT-log *
  send ABORT to every process in \(P_Y\) *
return
wait for VOTE-REQ from coordinator
    on timeout { write abort in DT log; return}
if vote is YES then begin
    write YES on DT log
    send YES to coordinator
    wait for decision
    on timeout run termination protocol
    if decision is COMMIT then write COMMIT on DT-log
else write ABORT on DT-log
end
else begin
    write ABORT on DT-log *
    send NO to coordinator *
end
return
Termination Protocol

Initiator:

start: send DEC-REQ to all
    wait for decision from any process
        on timeout goto start
    if decision is COMMIT then write COMMIT on DT log
        else write ABORT on DT log
    return

Responder:

wait for DEC-REQ from any process
if ABORT in DT-log then send ABORT to initiator
    else if COMMIT in DT-log then send COMMIT to initiator
    else if YES NOT on DT-log then
        { decide NO
            write ABORT on DT-log
            send ABORT to initiator and coordinator }
    else skip;
return
Comments on 2PC with DT

- A process **decides** when it writes **COMMIT/ABORT** on DT log
- When a process decides, the DM can execute the transaction
- For recovery, each process inspects its DT log
- 2PC is resilient to site and communication failures (even to network partitioning)
- 2PC is subject to **blocking**: A process may be blocked if it times out while uncertain
In the absence of failures, 2PC requires three rounds.

In the presence of failure, two additional rounds may be needed (however, blocked processes may take arbitrarily long to invoke the termination protocol).

In absence of failures, 2PC requires $3n$ messages.

In the presence of failures, additional $O(n^2)$ may be required.
Alternative Topologies for 2PC

Decentralized 2PC:

1. Coordinator broadcasts its vote to all (piggybacking VOTE-REQ on its vote)
2. If coordinator’s vote is YES, then everyone that receives it and has a YES vote broadcasts its vote
3. Anyone who receives NO or has a NO vote, decides ABORT
4. Others wait until they receive all votes to decide

- Communication complexity: $O(n^2)$
- Time complexity: 2 rounds
Alternative Topologies for 2PC

Linear 2PC:
Processes are arranged in a line,

\[ P_1, \ldots, P_n \]

1. For \( i = 1, \ldots, n - 1 \), \( P_i \) sends \text{YES} to \( P_{i+1} \) if its vote is \text{YES} and it had received no \text{NO} votes, and \text{NO} otherwise

2. Processes that receive \text{NO} decide \text{abort} (but continue to pass messages)

3. If \( P_n \) receives \text{YES} and its own vote is \text{YES}, it starts sending back \text{commit}.

- Communication complexity: \( O(2n) \)
- Time complexity: \( 2n \) rounds
# Summary of 2PCs

<table>
<thead>
<tr>
<th></th>
<th>Messages</th>
<th>Rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized 2PC</td>
<td>$3n$</td>
<td>3</td>
</tr>
<tr>
<td>Decentralized 2PC</td>
<td>$n^2 + n$</td>
<td>2</td>
</tr>
<tr>
<td>Linear 2PC</td>
<td>$2n$</td>
<td>$2n$</td>
</tr>
</tbody>
</table>

- Note that all entries in the table refer to protocols running without failures.
- Failures cause the **Termination Protocol** to be invoked, costing additional time and messages.
- **Hybrid** approaches are also possible.
- E.g., in Linear 2PC $P_n$ can **broadcast** the decision. Comm. complexity will remain intact, but time complexity will reduced to $n + 1$ rounds.
Towards a 3PC

- 2PC may cause *blocking* even for *non-total* site failure

- 3PC guarantees that *all* non-faulty processes eventually decide

- One version of 3PC tolerates only site failures

- Another version tolerates both communication and site failures, but causes blocking (less frequently than 2PC)

- Almost all “real” DB-systems use 2PC!!
Towards 3PC

Assume only site failures.

- If $p$ times-out while awaiting $q$’s message, it knows that $q$ has failed. It also knows that nobody else can communicate with $q$.
- In 2PC, if all non-failing processes are uncertain, they all block.

Suppose we guarantee:

\[
\text{some process is uncertain} \implies \text{no process has committed}
\]

Then uncertain processes can just decide **ABORT**.
Towards a 3PC

- 2PC: all vote **YES** $\implies$ all **commit** if there are no failures
- 3PC: attempts to guarantee commitment even if there are failures
- 3PC guarantees that no process commits while there are uncertain processes
- accomplished by an extra round, in which the coordinator sends **PRE-COMMIT** to all
- when $p$ receives **PRE-COMMIT**, it knows that
  1. everybody has voted **YES**, and
  2. it will **COMMIT** only when no one is uncertain
3PC: The protocol

- **Round 1:** $C$ sends VOTE-REQ
- **Round 2:** A process that recvs VOTE-REQ sends its vote to $C$
- **Round 3:** If $C$ rcvs YES votes from all, and its vote is YES, it sends PRE-DIR to all, else it sends ABORT to all
- **Round 4:** A process that rcvs PRE-DIR sends ACK, and a process that rcvs ABORT, decides ABORT and stops
- **Round 5:** If $C$ rcvs all ACKs, it decides COMMIT, sends COMMIT to all, and stops
- **Round 6:** A process that rcvs COMMIT, decides COMMIT and stops
Timeouts in 3PC

- In *steps 2 & 3*: just like 2PC
- In *step 5* (all vote *YES*, but some failed processes are uncertain): $C$ can decide *COMMIT* and send to all
- In *step 4*: nobody can decide *ABORT* (violating requirements) or *COMMIT* (w/o violating req. that nobody commits while others are uncertain)
- In *step 6*: since a process (awaiting *COMMIT*) does not know whether others have received *PRE-COMMIT*, it can decide neither *COMMIT* nor *abort*
- timeouts at steps 4 & 6 require running some cooperation protocol among correct processes, each of whom can be: *aborted*, *uncertain*, *committed*, or *pre-committable*
## Timeouts in 3PC

The following states can co-exist:

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

**Termination protocol** can have following rules:

- Some are **aborted** ➔ decision is **ABORT**
- Some are **committed** ➔ decision is **COMMIT**
- All (correct) are **uncertain** ➔ \( C \) can decide **ABORT**
- Some are **committable** but none is **committed** ➔ \( C \) sends **PRE-COMMIT** to all uncertain
Timeouts in 3PC

The following states can co-exist:

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

Termination protocol can have the following rules:

- Some are **aborted** $\implies$ decision is **ABORT**
- Some are **committed** $\implies$ decision is **COMMIT**
- All (correct) are **uncertain** $\implies$ $C$ can decide **ABORT**
- Some are **committable** AND none is **committed** $\implies$ $C$ sends **PRE-COMMIT** to all uncertain

But what to do if coordinator has failed?
Processes that times-out while waiting for `PRE-COMMIT` or `COMMIT` should elect a leader among themselves.

The leader, who is the new coordinator, sends `STATE-REQ` all operational participants; they then follows the termination rules above.

But what if processes fail during the termination protocol?
But What if processes fail during the termination protocol?

**Answer.** Leader can ignore them! (Of course, if the new coordinator fails, the operational processes will time-out and elect a new leader.) Eventually, either a leader will be elected that will be able to finish the protocol, or there will be a total site failure.

**Theorem.** In the absence of total failures, 3PC (and its termination protocol) is non-blocking and correct
Analysis of 3PC

- resilient to site failures
- It is non-blocking unless all fail
- In absence of failures, at most 5 rounds to complete (less if decision is ABORT)
- Each invocation of termination protocol results in additional 5 rounds, plus one round for the election protocol. Thus, total $6f + 5$ rounds. Worse than 2PC, but 2PC may cause blocking.
- In each round, at most $n$ messages are sent, thus, w/o failures, $5n$ messages are sent
At the end of 2PC every participant knows that every participant knows $\phi$, where:

$$\phi: \text{ all votes are YES}$$

Denote by $C$ the set of correct & committed processes. Then, at each point of 2PC we have that for every $i \in C$,

$$\text{COMMIT}_i \rightarrow K_i \phi$$

At the end of 2PC we have . . .
Knowledge in 2PC

At the end of 2PC every participant knows that every participant knows $\phi$, where:

\[ \phi: \text{all votes are YES} \]

Denote by $C$ the set of correct & committed processes. Then, at each point of 2PC we have that for every $i \in C$,

\[ \text{COMMIT}_i \rightarrow K_i\phi \]

At the end of 2PC we have . . .

\[ E^2_C\phi \]
If at the end of 2PC we have:

$E^2_C\phi$

What is the level of knowledge attained in 3PC?
Knowledge in 3PC

If at the end of 2PC we have:

\[ E_C^2 \phi \]

What is the level of knowledge attained in 3PC?

\[ E_C^3 \phi \]
Problems 7.2, 7.11, 7.12 from BGH.

7.2 Give a ACP that also satisfies the converse of condition AC3. That is, if all processes vote YES, then the decision must be COMMIT. Why is it not a good idea to enforce this condition?
In the 2PC protocol the coordinator first decides Commit (by writing a COMMIT record to its site’s DT-log) and then sends COMMIT messages to the participants. Suppose that order of these two steps is reversed. Show that this variation of 2PC is non-blocking if:

1. there are no communication failures and at most one process can fail, or
2. there are no communication failures and processes have the ability to broadcast messages atomically (i.e., so that either all recipients get it or no recipient does.)

How does this variation of 2PC affect the recovery procedures? Give recovery procedures that work for the modified 2pc protocol.
In the 2PC termination protocol each process independently sends a \texttt{DecReq} message, and each process that knows the decision responds to all the \texttt{DecReq}s it receives. An alternative approach would be to elect a new coordinator that will send the decision to all (if a decision can be reached, i.e., if not all operations processes are uncertain.) Develop a termination protocol that uses this approach. The protocol should handle both site and communication failures. Analyze the worst case round and message complexity of the resulting protocol. How do these compare to the round a message complexity of the protocol we studied? (Be sure to include the rounds and messages needed for election in your analysis.)