Types of Failures

- Site failure – assume all or nothing (fail-stop).
- Total failure – all sites fail.
- Partial failure – some sites fail.
- Communication failure:
  - corruption.
  - broken link.
  - network partition.
- Undelivered messages are dropped (assumption).
- Detecting failures via timeouts.
- Timeout failures are possible.
Atomic Commitment

- Model: Coordinator, participants, local DM that uses some recovery scheme.
- Each site has a distributed transaction log – DT.
- ACP: Atomic Commitment Protocol
  - A participant can vote Yes or No.
  - Either all processes commit or all abort.
ACP Requirements

- All Processes **that reach a decision** reach the **same** decision.
- A process cannot reverse a decision once it reached it.
- The Commit decision can only be reached if all processes voted **YES**.
- If there are **no failures** and all processes voted Yes then the decision will be Commit.
- Consider an execution with failures tolerable by the algorithm. If all current failures are repaired and none occurs thereafter, all processes will eventually reach a decision.
Blocking, Uncertainty

- **Uncertainty period** – between voting Yes and being able to know a definite decision.
- **Blocked** – a condition describing a process that cannot proceed to abort or commit until some failures are first repaired.
- **Independent recovery** – the ability of a recovering process to reach a decision concerning a transaction without communicating with other sites.
Impossibility Results

- If communication failures or total failures are possible, then every ACP may cause processes to become blocked. *(An ACP in presence of site failures – but not total site failures- does exist).*

- No ACP can guarantee independent recovery of failed processes.
Two Phase Commit (2PC)

- Voting phase:
  - Coordinator sends vote request to participants.
  - Participants respond with Yes or No.

- Decision phase:
  - If all votes are Yes, the coordinator decides to commit.
  - If any vote is No, the coordinator decides to abort.

- Timeout actions are taken when processes expect messages for ‘too long’.

- These actions may include a termination protocol, designed to conclude (although it may be blocked) the status of a transaction.

- There may be a number of possible communication topologies.
The 2PC Algorithm

- Figure 7-3.
- Figure 7-4. Note the small bug.
Coordinator’s algorithm

send \texttt{VOTE-REQ} to all participants;
write start-2PC record in DT log;
wait for vote messages (YES or NO) from all participants
  on timeout begin
    let $P_Y$ be the processes from which YES was received;
    write abort record in DT log;
    send \texttt{ABORT} to all processes in $P_Y$;
    return
  end;
if all votes were YES and coordinator votes Yes then begin
  write commit record in DT log;
  send \texttt{COMMIT} to all participants
end
else begin
  let $P_Y$ be the processes from which YES was received;
  write abort record in DT log;
  send \texttt{ABORT} to all processes in $P_Y$
end;
return
Participant’s algorithm

wait for vote-req from coordinator
  on timeout begin
    write abort record in DT log;
    return
  end;
if participant votes Yes then begin
  write a yes record in DT log;
  send YES to coordinator;
  wait for decision message (COMMIT or ABORT) from coordinator
    on timeout initiate termination protocol /* cf. Fig. 7-4 */
  if decision message is COMMIT then write commit record in DT log
  else write abort record in DT log
end
else /* participant’s vote is No */ begin
  write abort record in DT log;
  send NO to coordinator
end;
return
Initiator's algorithm

start: send DECISION-REQ to all processes;
    wait for decision message from any process
    on timeout goto start; /* blocked! */
if decision message is COMMIT then
    write commit record in DT log
else /* decision message is ABORT */
    write abort record in DT log;
return

Responder's algorithm

wait for DECISION-REQ from any process p;
if responder has not voted Yes or has decided to Abort (i.e., has an
    abort record in DT log) then send ABORT to p
else if responder has decided to Commit (i.e., has a commit
    record in DT log) then send COMMIT to p
else /* responder is in its uncertainty period */ skip;
return

FIGURE 7-4
Cooperative Termination Protocol for 2PC

There may be a "bug" here, need to write first to DT and in Participant's algorithm, check it.
When a site \( S \) recovers

- DT contains start-2PC, \( S \) is coordinator’s site.
  - Contains \textbf{commit} or \textbf{abort} \( \rightarrow \) decision was reached prior to failure.
  - Otherwise, unilaterally decide Abort (write DT).

- Otherwise, \( S \) was a participant’s \( p \) site.
  - Contains \textbf{commit} or \textbf{abort} \( \rightarrow \) decision was reached prior to failure.
  - Does not contain a \textbf{yes} \( \rightarrow \) either \( p \) failed before voting or voted No (but did not write an \textbf{abort} record); unilaterally decide Abort (write DT).
  - Contains a \textbf{yes} but no \textbf{commit} or \textbf{abort}; \( p \) failed while in its uncertainty period. Can use the termination protocol (next).
Evaluation of 2PC

- Resilient to both site and communication failures.
- Subject to blocking (even if only site failures).
- Three message rounds (without failures). With failures – (possibly overlapped) five rounds.
- Messages: 3*n with no failures. With failures: (n/2) * (3*n +7).
- Counts can change with a different network topology (not covered).
Pictorially (with a few additions)

Coordinator

VOTE-REQ

ANS

Participant

Force-write
YES/Abort
record

Force-write
commit/abort
record

Decision

ACK

Write non-forced
End record

(c) Oded Shmueli 2004
Presumed Abort (with no information)

Coordinator

VOTE-REQ

ANS

Participant

Force-write
YES/Abort record

Abort

Write non-forced Abort record
Presumed Commit (with no information): **Commit** case

**Coordinator**
- Force-write initiation record

**Participant**
- Force-write YES record
- Write non-forced Commit record

- VOTE-REQ
- YES
- Commit
Presumed Commit (with no information): **Abort** case

### Coordinator

- Force-write initiation **record**
  - VOTE-REQ
  - ANS
- Force-erase initiation **record**
  - Abort
  - ACK
- Write **non-forced**
- End **record**

### Participant

- Force-write **YES/Abort** record
- Force-write **Abort** record
3PC – Assumptions and Invariant

- **Major assumption**: no communication failures, only site failures:
  - time out → process is down.
  - all operational processes can communicate with each other.

- **Invariant NB**: If any operational process is uncertain then no process (whether operational or failed) can have decided to commit.
Three Phase Commit (3PC)

- **Version 1**: Non-blocking in presence of site failure but not in presence of communication failures or total site failure.

- **Version 2**: Can handle both communication and site failure, can block. Still, blocking is less frequent. We will not cover this version.

- Both versions are considered “impractical” due to message and round overhead. But, they are interesting and reveal many intricacies.
3PC Stages – No Failures

1) Coordinator sends **VOTE-REQ** to participants.
2) Participant responds with **YES** or **NO** (decides abort and stops).
3) Any **NO** votes or coordinator says **NO** or not all responded → decide abort, send **ABORT** to all **YES** voters. Otherwise – send **PRE-COMMIT** to all.
4) Participants wait for **ABORT** or **PRE-COMMIT**:
   1) **ABORT**: decide abort and stop.
   2) **PRE-COMMIT**: Respond with ACK.
5) When all **ACK** arrive, Coordinator sends **COMMIT**.
6) A participant waits for **COMMIT**. Upon receipt, it decides commit and stops.
Analysis Failures ➔ Timeouts ➔

Actions

2. Coordinator waits for a vote: decide abort and send ABORT to all.
3. Participant waits for PRE-COMMIT or ABORT: Communicate with other processes so as to reach a consistent decision.
4. Coordinator waits for ACKS: Coordinator knows the failed participants voted YES. Coordinator ignores failure, sends COMMITS. So, processes may decide commit when failed processes are uncertain. NB is maintained.
5. Participant p waits for COMMIT:
   1. Communicate with other processes so as to reach a consistent decision.
   2. Why shouldn’t p just commit? **Reason:** some other process might not have received the PRE-COMMIT. It is uncertain. Committing would violate NB.
Timeout
The 3PC Algorithm

- Figure 7-10.
- Figure 7-11.
States

- **Aborted**: not voted, voted NO, or received ABORT. So it either decided ABORT or can unilaterally decide abort.
- **Uncertain**: voted YES, has not received PRE-COMMIT.
- **Committable**: received PRE-COMMIT, has not received COMMIT.
- **Committed**: received COMMIT and decided COMMIT.
## State Compatibility

<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>N</td>
<td>N</td>
</tr>
<tr>
<td>Committable</td>
<td></td>
<td></td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Committed</td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
</tbody>
</table>

(c) Oded Shmueli 2004
Participant Termination Protocol

- Initialize an election protocol to elect a new coordinator.
- Coordinator sends STATE-REQ to all operational processes.
- Participants send their state information.
- Coordinator applies termination rules and decides to commit or abort.
- If new coordinator fails, another will be elected …
- Processes that failed do not participate in the termination protocol when they recover.
Termination Rules

1. If some process is Aborted, decide abort, send ABORT messages.
2. If some process is Committed, decide commit, send COMMIT messages.
3. If all reporting processes are Uncertain, decide abort, send ABORT messages.
4. If some process is committable and none is Committed:
   1. Send PRE-COMMIT to all Uncertain processes.
   2. Wait for ACKS.
   3. After receiving ACKS, decide commit, send COMMIT messages to all processes. Ignore non-responders.

**Note:** Exactly one of 1-4 will apply.
Election Protocol

- Each process maintains a list UP of live processes.
- If a process detects another process is failed, it removes that process from its UP list.
- If a process detects the coordinator failed, it sends URELECTED to smallest numbered UP process.
- The elected coordinator starts requesting states.
- Other processes may deduce the change of guards once they receive STATE-REQ from the new coordinator.
Site Failure (not total)
Recovering from non-total Failure

- A process recovers from failure.
- DT log records significant events.
- Extracts its state from the DT log.
- Can unilaterally recover if never voted YES, has received COMMIT or ABORT.
- Voted YES but has not received ABORT or COMMIT. Asks for help.
  - What if it has received PRE-COMMIT, can it decide unilaterally?
    - No, the failure could have happened so that the operational processes decided to abort …
Total Site Failure
Recovering from Total Failure

- Total failure – all processes.
- Suppose p recovers. If p failed while Uncertain and was not the last to fail, it cannot recover independently.
- So, p must wait for some q to recover that can make this decision. Process q either decided already or starts the termination protocol. Recovered processes will decide this way.
- Without the last process to have failed, inconsistent decisions may be reached.
Determining the Last Process to Fail

- The UP sets are kept in stable storage.
- UPi helps a set of processes, Pi, in determining whether the last process to fail is among them.
- A set R of processes contains the last process to fail iff $R \supseteq \bigcap_{p \in R} UP_p$. 
Correctness and Communications Failures

- See BHG.