Today’s plan

• f+1-round lower bound for stopping agreement, cont’d.
• Various other kinds of consensus problems in synchronous networks:
  – k-agreement
  – Approximate agreement (skip)
  – Distributed commit
• Reading:
  – [Aguilera, Toueg]
  – [Keidar, Rajsbaum]
  – Chapter 7 (skip 7.2)
• Next:
  – Modeling asynchronous systems
  – Chapter 8
Theorem 1: Suppose $n \geq f + 2$. There is no $n$-process $f$-fault stopping agreement algorithm in which nonfaulty processes always decide at the end of round $f$.

Old proof: Suppose $A$ exists.
- Construct a chain of executions, each with at most $f$ failures, where:
  - First has decision value 0, last has decision value 1.
  - Any two consecutive executions are indistinguishable to some process $i$ that is nonfaulty in both.
- So decisions in first and last executions are the same, contradiction.
- Must fail $f$ processes in some executions in the chain, in order to remove all the required messages, at all rounds.
- Construction in book, LTTR.

Newer proof [Aguilera, Toueg]:
- Uses ideas from [Fischer, Lynch, Paterson], impossibility of asynchronous consensus.
By contradiction. Assume A solves stopping agreement for f failures and everyone decides after exactly f rounds.

Consider only executions in which at most one process fails during each round.

Recall failure at a round allows process to miss sending any subset of the messages, or to send all but halt before changing state.

Regard vector of initial values as a 0-round execution.

Defs (adapted from [FLP]): \( \alpha \), an execution that completes some finite number (possibly 0) of rounds, is:

- **0-valent**, if 0 is the only decision that can occur in any execution (of the kind we consider) that extends \( \alpha \).
- **1-valent**, if 1 is...
- **Univalent**, if \( \alpha \) is either 0-valent or 1-valent (essentially decided).
- **Bivalent**, if both decisions occur in some extensions (undecided).
Univalence and Bivalence

0-valent

0 0 0

1-valent

1 1 1

Bivalent

0 1 1

α univalent

α

α

α
Initial bivalence

• **Lemma 1:** There is some 0-round execution (vector of initial values) that is bivalent.

• **Proof (from [FLP]):**
  – Assume for contradiction that all 0-round executions are univalent.
  – 000…0 is 0-valent.
  – 111…1 is 1-valent.
  – So there must be two 0-round executions that differ in the value of just one process, i, such that one is 0-valent and the other is 1-valent.
  – But this is impossible, because if i fails at the start, no one else can distinguish the two 0-round executions.
Bivalence through f-1 rounds

• **Lemma 2:** For every $k$, $0 \leq k \leq f-1$, there is a bivalent $k$-round execution.

• **Proof:** By induction on $k$.
  – **Base:** Lemma 1.
  – **Inductive step:** Assume for $k$, show for $k+1$, where $k < f -1$.
    • Assume bivalent $k$-round execution $\alpha$.
    • Assume for contradiction that every 1-round extension of $\alpha$ (with at most one new failure) is univalent.
    • Let $\alpha^*$ be the 1-round extension of $\alpha$ in which no new failures occur in round $k+1$.
    • By assumption, $\alpha^*$ is univalent, WLOG 1-valent.
    • Since $\alpha$ is bivalent, there must be another 1-round extension of $\alpha$, $\alpha^0$, that is 0-valent.
Bivalence through f-1 rounds

• In $\alpha^0$, some single process, say $i$, fails in round $k+1$, by not sending to some set of processes, say $J = \{j_1, j_2, \ldots, j_m\}$.
• Define a chain of $(k+1)$-round executions, $\alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^m$.
• Each $\alpha^l$ in this sequence is the same as $\alpha^0$ except that $i$ also sends messages to $j_1, j_2, \ldots, j_l$.
  – Adding in messages from $i$, one at a time.
• Each $\alpha^l$ is univalent, by assumption.
• Since $\alpha^0$ is 0-valent, either:
  – At least one of these is 1-valent, or
  – All are 0-valent.
Case 1: At least one $\alpha^l$ is 1-valent

- Then there must be some $l$ such that $\alpha^{l-1}$ is 0-valent and $\alpha^l$ is 1-valent.
- But $\alpha^{l-1}$ and $\alpha^l$ differ after round $k+1$ only in the state of one process, $j_l$.
- We can extend both $\alpha^{l-1}$ and $\alpha^l$ by simply failing $j_l$ at beginning of round $k+2$.
  - There is actually a round $k+2$ because we’ve assumed $k < f-1$, so $k+2 \leq f$.
- And no one left alive can tell the difference!
- Contradiction for Case 1.
Case 2: Every $\alpha^l$ is 0-valent

- Then compare:
  - $\alpha^m$, in which i sends all its round $k+1$ messages and then fails, with
  - $\alpha^*$, in which i sends all its round $k+1$ messages and does not fail.

- No other differences, since only i fails at round $k+1$ in $\alpha^m$.

- $\alpha^m$ is 0-valent and $\alpha^*$ is 1-valent.

- Extend to full f-round executions:
  - $\alpha^m$, by allowing no further failures,
  - $\alpha^*$, by failing i right after round $k+1$ and then allowing no further failures.

- No one can tell the difference.

- Contradiction for Case 2.
Bivalence through f-1 rounds

• So we’ve proved, so far:

• Lemma 2: For every $k$, $0 \leq k \leq f-1$, there is a bivalent $k$-round execution.
Disagreement after f rounds

- **Lemma 3**: There is an f-round execution in which two nonfaulty processes decide differently.

- **Proof**:
  - Use Lemma 2 to get a bivalent (f-1)-round execution $\alpha$ with $\leq f-1$ failures.
  - In every 1-round extension of $\alpha$, everyone who hasn’t failed must decide (and agree).
  - Let $\alpha^*$ be the 1-round extension of $\alpha$ in which no new failures occur in round $f$.
  - Everyone who is still alive decides after $\alpha^*$, and they must decide the same thing. WLOG, say they decide 1.
  - Since $\alpha$ is bivalent, there must be another 1-round extension of $\alpha$, say $\alpha^0$, in which some nonfaulty process (and so, all nonfaulty processes) decide 0.
Disagreement after f rounds

- In $\alpha^0$, some single process $i$ fails in round $f$.
- Let $j$, $k$ be two nonfaulty processes.
- Define a chain of three $f$-round executions, $\alpha^0, \alpha^1, \alpha^*$, where $\alpha^1$ is identical to $\alpha^0$ except that $i$ sends to $j$ in $\alpha^1$ (it might not in $\alpha^0$).
- Then $\alpha^1 \sim^k \alpha^0$.
- Since $k$ decides 0 in $\alpha^0$, $k$ also decides 0 in $\alpha^1$.
- Also, $\alpha^1 \sim^j \alpha^*$.
- Since $j$ decides 1 in $\alpha^*$, $j$ also decides 1 in $\alpha^1$.
- Yields disagreement in $\alpha^1$, contradiction!

- So we’ve proved:
  - **Lemma 3**: There is an $f$-round execution in which two nonfaulty processes decide differently.
  - Which immediately yields the lower bound result.
Early-stopping agreement algorithms

• Tolerate f failures in general, but in executions with $f' < f$ failures, terminate faster.

• [Dolev, Reischuk, Strong 90] Stopping agreement algorithm in which all nonfaulty processes terminate in $\leq \min(f' + 2, f+1)$ rounds.
  – If $f' + 2 \leq f$, decide “early”, within $f' + 2$ rounds; in any case decide within $f+1$ rounds.

• [Keidar, Rajsbaum 02] Lower bound of $f' + 2$ for early-stopping agreement.
  – Not just $f' + 1$. Early stopping requires an extra round.

• **Theorem 2:** Assume $0 \leq f' \leq f - 2$ and $f < n$. Every early-stopping agreement algorithm tolerating $f$ failures has an execution with $f'$ failures in which some nonfaulty process doesn’t decide by the end of round $f' + 1$. 
Just consider special case: $f' = 0$

- **Theorem 3:** Assume $2 \leq f < n$. Every early-stopping agreement algorithm tolerating $f$ failures has a failure-free execution in which some nonfaulty process does not decide by the end of round 1.

- **Definition:** Let $\alpha$ be an execution that completes some finite number (possibly 0) of rounds. Then $\text{val}(\alpha)$ is the unique decision value in the extension of $\alpha$ with no new failures.

- **Proof of Theorem 3:**
  - Assume executions in which at most one process fails per round.
  - Identify 0-round executions with vectors of initial values.
  - Assume, for contradiction, that everyone decides by round 1, in all failure-free executions.
  - $\text{val}(000\ldots0) = 0$, $\text{val}(111\ldots1) = 1$.
  - So there must be two 0-round executions $\alpha^0$ and $\alpha^1$, that differ in the value of just one process $i$, such that $\text{val}(\alpha^0) = 0$ and $\text{val}(\alpha^1) = 1$. 
Special case: $f' = 0$

- 0-round executions $\alpha^0$ and $\alpha^1$, differing only in the initial value of process i, such that $\text{val}(\alpha^0) = 0$ and $\text{val}(\alpha^1) = 1$.

- In failure-free extensions of $\alpha^0$, $\alpha^1$, all processes decide in one round.
- Define:
  - $\beta^0$, 1-round extension of $\alpha^0$, in which process i fails, sends only to j.
  - $\beta^1$, 1-round extension of $\alpha^1$, in which process i fails, sends only to j.
- Then:
  - $\beta^0$ looks to j like ff extension of $\alpha^0$, so j decides 0 in $\beta^0$ after 1 round.
  - $\beta^1$ looks to j like ff extension of $\alpha^1$, so j decides 1 in $\beta^1$ after 1 round.
- $\beta^0$ and $\beta^1$ are indistinguishable to all processes except i, j.
- Define:
  - $\gamma^0$, infinite extension of $\beta^0$, in which process j fails right after round 1.
  - $\gamma^1$, infinite extension of $\beta^1$, in which process j fails right after round 1.
- By agreement, all nonfaulty processes must decide 0 in $\gamma^0$, 1 in $\gamma^1$.
- But $\gamma^0$ and $\gamma^1$ are indistinguishable to all nonfaulty processes, so they can’t decide differently, contradiction.
k-Agreement
k-agreement

- Usually called k-set agreement or k-set consensus.
- Generalizes ordinary stopping agreement by allowing k different decisions instead of just one.
- Motivation:
  - Practical:
    - Allocating shared resources, e.g., agreeing on small number of radio frequencies to use for sending/receiving broadcasts.
  - Mathematical:
    - Natural generalization of ordinary 1-agreement.
    - Elegant theory: Nice topological structure, tight bounds.
The k-agreement problem

• Assume:
  – n-node complete undirected graph
  – Stopping failures only
  – Inputs, decisions in finite totally-ordered set V (appear in state variables).

• Correctness conditions:
  – Agreement:
    • \( \exists W \subseteq V, |W| = k \), all decision values in W.
    • That is, there are at most k different decision values.
  – Validity:
    • Any decision value is some process’ initial value.
    • Like strong validity for 1-agreement.
  – Termination:
    • All nonfaulty processes eventually decide.
FloodMin k-agreement algorithm

• Algorithm:
  – Each process remembers the min value it has seen, initially its own value.
  – At each round, broadcasts its min value.
  – Decide after some generally-agreed-upon number of rounds, on current min value.

• Q: How many rounds are enough?

• 1-agreement: f+1 rounds
  – Argument like those for previous stopping agreement algorithms.

• k-agreement: ⌊f/k⌋ + 1 rounds.

• Allowing k values divides the runtime by k.
FloodMin correctness

• **Theorem 1:** FloodMin, for \( \lfloor f/k \rfloor + 1 \) rounds, solves \( k \)-agreement.

• **Proof:**
  
  • Define \( M(r) = \) set of min values of active (not-yet-failed) processes after \( r \) rounds.
  
  • This set can only decrease over time:

  • **Lemma 1:** \( M(r+1) \subseteq M(r) \) for every \( r, 0 \leq r \leq \lfloor f/k \rfloor \).
    
    • **Proof:** Any min value after \( r+1 \) is someone’s min value after \( r \).
Proof of Theorem 1, cont’d

• **Lemma 2:** If at most $d-1$ processes fail during round $r$, then $|M(r)| \leq d$.

• **E.g., for $d = 1$:** If no one fails during round $r$ then all have the same min value after $r$.

• **Proof:** Show contrapositive.
  – Suppose that $|M(r)| > d$, show at least $d$ processes fail in round $r$.
  – Let $m = \max (M(r))$.
  – Let $m' < m$ be any other element of $M(r)$.
  – Then $m' \in M(r-1)$ by Lemma 1.
  – Let $i$ be a process active after $r-1$ rounds that has $m'$ as its min value after $r-1$ rounds.
  – Claim $i$ fails in round $r$:
    • If not, everyone would receive $m$; in round $r$.
    • Means that no one would choose $m > m'$ as its min, contradiction.
  – But this is true for every $m' < m$ in $M(r)$, so at least $d$ processes fail in round $r$. 
Proof of Theorem 1, cont’d

• **Validity:** Easy
• **Termination:** Obvious
• **Agreement:** By contradiction.
  – Assume an execution with $> k$ different decision values.
  – Then the number of min values for active processes after the full $\lfloor f/k \rfloor + 1$ rounds is $> k$.
  – That is, $|M(\lfloor f/k \rfloor + 1)| > k$.
  – Then by Lemma 1, $|M(r)| > k$ for every $r$, $0 \leq r \leq \lfloor f/k \rfloor + 1$.
  – So by Lemma 2, at least $k$ processes fail in each round.
  – That’s at least $(\lfloor f/k \rfloor + 1) k$ total failures, which is $> f$ failures.
  – Contradiction!
Lower Bound (sketch)

- **Theorem 2**: Any algorithm for k-agreement requires $\geq \left\lfloor \frac{f}{k} \right\rfloor + 1$ rounds.

- Recall old proof for f+1-round lower bound for 1-agreement.
  - Chain of executions for assumed algorithm:
    
    $\alpha_0 \ ----- \alpha_1 \ ----- \ldots \ ----- \alpha_j \ ----- \alpha_{j+1} \ ----- \ldots \ ----- \alpha_m$
    
    - Each execution has a unique decision value.
    - Executions at ends of chain have specified decision values.
    - Two consecutive executions look the same to some nonfaulty process, who (therefore) decides the same in both.

- Argument doesn’t extend immediately to k-agreement:
  - Can’t assume a unique value in each execution.
  - Example: For 2-agreement, could have 3 different values in 2 consecutive executions without violating agreement.

- Instead, use a **k-dimensional generalized chain**.
Lower bound

• Assume, for contradiction:
  – n-process k-agreement algorithm tolerating f failures.
  – All processes decide just after round r, where \( r \leq \lceil f/k \rceil \).
  – All-to-all communication at all rounds.
  – \( n \geq f + k + 1 \) (so each execution we consider has at least \( k+1 \) nonfaulty processes)
  – \( V = \{0,1,\ldots,k\} \), \( k+1 \) values.

• Get contradiction by proving existence of an execution with \( \geq k + 1 \) different decision values.

• Use k-dimensional collection of executions rather than 1-dimensional.
  – \( k = 2 \): Triangle
  – \( k = 3 \): Tetrahedron, etc.
Labeling nodes with executions

- **Bermuda Triangle** \((k = 2)\): Any algorithm vanishes somewhere in the interior.
- **Label nodes with executions:**
  - Corner: No failures, all have same initial value.
  - Boundary edge: Initial values chosen from those of the two endpoints.
  - For \(k > 2\), generalize to boundary faces.
  - Interior: Mixture of inputs.
- **Label so executions “morph gradually” in all directions:**
- **Difference between two adjacent executions along an outer edge:**
  - Remove or add one message, to a process that fails immediately.
  - Fail or recover a process.
  - Change initial value of failed process.
Labeling nodes with process names

• Also label each node with the name of a process that is nonfaulty in the node’s execution.

• **Consistency:** For every tiny triangle (simplex) \( T \), there is a single execution \( \beta \), with at most \( f \) faults, that is “compatible” with the executions and processes labeling the corners of \( T \):
  
  – All the corner processes are nonfaulty in \( \beta \).
  – If \((\alpha',i)\) labels some corner of \( T \), then \( \alpha' \) is indistinguishable by \( i \) from \( \beta \).

• Formalizes the “gradual morphing” property.
• Proof by laborious construction.
• Can recast chain arguments for 1-agreement in this style:

\[
\begin{align*}
\beta \\
\alpha_0 &-\ldots-\alpha_1 &-\ldots-\alpha_j &-\ldots-\alpha_{j+1} &-\ldots-\alpha_m \\
p_0 & &p_1 & &\ldots & &p_j & &p_{j+1} & &\ldots & &p_m
\end{align*}
\]

  – \( \beta \) indistinguishable by \( p_j \) from \( \alpha_j \)
  – \( \beta \) indistinguishable by \( p_{j+1} \) from \( \alpha_{j+1} \)
Bound on rounds

• This labeling construction uses the assumption \( r \leq \left\lfloor \frac{f}{k} \right\rfloor \), that is, \( f \geq r \cdot k \).

• How:
  – We are essentially constructing chains simultaneously in \( k \) directions (2 directions, in the Bermuda Triangle).
  – We need \( r \) failures (one per round) to construct the “chain” in each direction.
  – For \( k \) directions, that’s \( r \cdot k \) total failures.

• Details LTTR (see book, or paper [Chaudhuri, Herlihy, Lynch, Tuttle])
Coloring the nodes

- Now color each node \( v \) with a “color” in \( \{0, 1, \ldots, k\} \):
  - If \( v \) is labeled with \( (\alpha, i) \) then \( \text{color}(v) = i \)’s decision value in \( \alpha \).

- **Properties:**
  - Colors of the major corners are all different.
  - Color of each boundary edge node is the same as one of the endpoint corners.
  - For \( k > 2 \), generalize to boundary faces.

- **Coloring properties follow from Validity,** because of the way the initial values are assigned.
Sperner Colorings

• A coloring with the listed properties (suitably generalized to k dimensions) is called a “Sperner Coloring” (in algebraic topology).

• Sperner’s Lemma: Any Sperner Coloring has some tiny triangle (simplex) whose k+1 corners are colored by all k+1 colors.

• Find one?
Applying Sperner’s Lemma

- Apply Sperner’s Lemma to the coloring we constructed.
- Yields a tiny triangle (simplex) \( T \) with \( k+1 \) different colors on its corners.
- Which means \( k+1 \) different decision values for the executions and processes labeling its corners.
- But consistency for \( T \) yields a single execution \( \beta \), with at most \( f \) faults, that is “compatible” with the executions and processes labeling the corners of \( T \):
  - All the corner processes are nonfaulty in \( \beta \).
  - If \((\alpha',i)\) labels some corner of \( T \), then \( \alpha' \) is indistinguishable by \( i \) from \( \beta \).
- So all the corner processes behave the same in \( \beta \) as they do in their own corner executions, and decide on the same values as in those executions.
- That’s \( k+1 \) different decision values in one execution with at most \( f \) faults.
- Contradicts \( k \)-agreement.
Approximate Agreement
Approximate Agreement problem

- Agreement on real number values:
  - Readings of several altimeters on an aircraft.
  - Values of approximately-synchronized clocks.
- Consider with Byzantine participants, e.g., faulty hardware.
- Abstract problem:
  - Inputs, outputs are reals
  - Agreement: Within $\varepsilon$.
  - Validity: Within range of initial values of nonfaulty processes.
  - Termination: Nonfaulty eventually decide.
- Assumptions: Complete n-node graph, $n > 3f$.
- Could solve by exact BA, using $f+1$ rounds and lots of communication.
- But better algorithms exist:
  - Simpler, cheaper
  - Extend to asynchronous settings, whereas BA is unsolvable in asynchronous networks.
Approximate agreement algorithm
[Dolev, Lynch, Pinter, Stark, Weihl]

• Use convergence strategy, successively narrowing the interval of guesses of the nonfaulty processes.
  – Take an average at each round.
  – Because of Byzantine failures, need fault-tolerant average.

• Maintain val, latest estimate, initially initial value.

• At every round:
  – Broadcast val, collect received values into multiset $W$.
  – Fill in missing entries with any values.
  – Calculate $W' = \text{reduce}(W)$, by discarding $f$ largest and $f$ smallest elements.
  – Calculate $W'' = \text{select}(W')$, by choosing the smallest value in $W'$ and every $f$'th value thereafter.
  – Reset val to $\text{mean}(W'')$. 
Example: \( n = 4, f = 1 \)

- **Initial values:** 1, 2, 3, 4
- **Process 3** faulty, sends:
  - proc 1: 2     proc. 2: 100     proc 3: -100
- **Process 1:**
  - Receives (1, 2, 2, 4), reduces to (2, 2), selects (2, 2), mean = 2.
- **Process 2:**
  - Receives (1, 2, 100, 4), reduces to (2, 4), selects (2, 4), mean = 3.
- **Process 4:**
  - Receives (1, 2, -100, 4), reduces to (1, 2), selects (1, 2), mean = 1.5.
One-round guarantees

- **Lemma 1**: Any nonfaulty process’ val after the round is in the range of nonfaulty processes’ vals before the round.
  - **Proof**: All elements of reduce(W) are in this range, because there are at most f faults, and we discard the top and bottom f values.

- **Lemma 2**: Let d be the range of nonfaulty processes’ vals just before the round. Then the range of nonfaulty processes’ vals after the round is at most \( \frac{d}{\lceil \frac{n - (2f+1)}{f} \rceil + 1} \).
  - That is:
    - If \( n = 3f + 1 \), then the new range is \( \frac{d}{2} \).
    - If \( n = kf + 1, k \geq 3 \), then the new range is \( \frac{d}{k-1} \).
  - **Proof**: Calculations, in book.

- **Example**: \( n = 4, f = 1 \)
  - Initial vals: 1, 2, 3, 4, range is 3.
  - Process 3 faulty, sends 2 to proc 1, 100 to proc 2, -100 to proc 3.
  - New vals of nonfaulty processes: 2, 3, 1.5
  - New range is 1.5.
The complete algorithm

• Just run the 1-round algorithm repeatedly.
• Termination: Add a mechanism, e.g.:
  – Each node individually determines a round by which it knows that the vals of nonfaulty processes are all within $\varepsilon$.
    • Collect first round vals, predict using known convergence rate.
  – After the determined round, decide locally.
  – Thereafter, send the decision value.
    • Upsets the convergence calculation.
    • But that doesn’t matter because the vals are already within $\varepsilon$.

• Remarks:
  – Convergence rate can be improved somewhat by using 2-round blocks [Fekete].
  – Algorithm extends easily to asynchronous case, using an “asynchronous round” structure we’ll see later.
Distributed Commit
Distributed Commit

**Motivation:** Distributed database transaction processing
- A database transaction performs work at several distributed sites.
- Transaction manager (TM) at each site decides whether it would like to "commit" or "abort" the transaction.
  - Based on whether the transaction’s work has been successfully completed at that site, and results made stable.
  - All TMs must agree on whether to commit or abort.

**Assume:**
- Process stopping failures only.
- n-node, complete, undirected graph.

**Require:**
- **Agreement:** No two processes decide differently (faulty or not, uniformity)
- **Validity:**
  - If any process starts with 0 (abort) then 0 is the only allowed decision.
  - If all start with 1 (commit) and there are no faulty processes then 1 is the only allowed decision.
Correctness Conditions for Commit

• **Agreement:** No two processes decide differently.

• **Validity:**
  – If any process starts with 0 then 0 is the only allowed decision.
  – If all start with 1 and there are no faulty processes then 1 is the only allowed decision.
  – Note the asymmetry: Guarantee abort (0) if anyone wants to abort; guarantee commit (1) if everyone wants to commit and no one fails (best case).

• **Termination:**
  – **Weak termination:** If there are no failures then all processes eventually decide.
  – **Strong termination (non-blocking condition):** All nonfaulty processes eventually decide.
2-Phase Commit

- Traditional, blocking algorithm (guarantees weak termination only).
- Assumes distinguished process 1, acts as “coordinator” (leader).
- **Round 1:** All send initial values to process 1, who determines the decision.
- **Round 2:** Process 1 sends out the decision.

Q: When can each process actually decide?
- Anyone with initial value 0 can decide at the beginning.
- Process 1 decides after receiving round 1 messages:
  - If it sees 0, or doesn’t hear from someone, it decides 0; otherwise decides 1.
- Everyone else decides after round 2.
Correctness of 2-Phase Commit

• Agreement:
  – Because decision is centralized (and consistent with any individual initial decisions).

• Validity:
  – Because of how the coordinator decides.

• Weak termination:
  – If no one fails, everyone terminates by end of round 2.

• Strong termination?
  – No: If coordinator fails before sending its round 2 messages, then others with initial value 1 will never terminate.
Add a termination protocol?

- We might try to add a termination protocol: other processes try to detect failure of coordinator and finish agreeing on their own.

- But this can’t always work:
  - If initial values are 0,1,1,1, then by validity, others must decide 0.
  - If initial values are 1,1,1,1 and process 1 fails just after deciding, and before sending out its round 2 messages, then:
    - By validity, process 1 must decide 1.
    - By agreement, others must decide 1.
  - But the other processes can’t distinguish these two situations.
Complexity of 2-phase commit

- **Time:**
  - 2 rounds

- **Communication:**
  - At most 2n messages
3-Phase Commit [Skeen]

- Yields strong termination.
- **Trick:** Introduce intermediate stage, before actually deciding.
- Process states classified into 4 categories:
  - `dec-0`: Already decided 0.
  - `dec-1`: Already decided 1.
  - `ready`: Ready to decide 1 but hasn’t yet.
  - `uncertain`: Otherwise.
- Again, process 1 acts as “coordinator”.
- Communication pattern:
3-Phase Commit

• All processes initially uncertain.

• Round 1:
  – All other processes send their initial values to p1.
  – All with initial value 0 decide 0 (and enter dec-0 state)
  – If p1 receives 1s from everyone and its own initial value is 1, p1 becomes ready, but doesn’t yet decide.
  – If p1 sees 0 or doesn’t hear from someone, p1 decides 0.

• Round 2:
  – If p1 has decided 0, broadcasts “decide 0”, else broadcasts “ready”.
  – Anyone else who receives “decide 0” decides 0.
  – Anyone else who receives “ready” becomes ready.
  – Now p1 decides 1 if it hasn’t already decided.

• Round 3:
  – If p1 has decided 1, bcasts “decide 1”.
  – Anyone else who receives “decide 1” decides 1.
3-Phase Commit

- Key invariants (after 0, 1, 2, or 3 rounds):
  - If any process is in ready or dec-1, then all processes have initial value 1.
  - If any process is in dec-0 then:
    - No process is in dec-1, and no non-failed process is ready.
  - If any process is in dec-1 then:
    - No process is in dec-0, and no non-failed process is uncertain.

- Proof: LTTR.
  - Key step: Third condition is preserved when p1 decides 1 after round 2.
  - In this case, p1 knows that:
    - Everyone’s input is 1.
    - No one decided 0 at the end of round 1.
    - Every other process has either become ready or has failed (without deciding).
  - Implies third condition.

- Note critical use of synchrony here:
  - p1 infers that non-failed processes are ready just because round 2 is completed.
  - Without synchrony, would need positive acknowledgments.
Correctness conditions (so far)

- Agreement and validity follow, for these three rounds.
- Weak termination holds
- Strong termination:
  - Doesn’t hold yet---must add a termination protocol.
  - Allow process 2 to act as coordinator, then 3,…
  - “Rotating coordinator” strategy
3-Phase Commit

• Round 4:
  – All processes send current decision status (dec-0, uncertain, ready, or dec-1) to p2.
  – If p2 receives any dec-0’s and hasn’t already decided, then p2 decides 0.
  – If p2 receives any dec-1’s and hasn’t already decided, then p2 decides 1.
  – If all received values, and its own value, are uncertain, then p2 decides 0.
  – Otherwise (all values are uncertain or ready and at least one is ready), p2 becomes ready, but doesn’t decide yet.

• Round 5 (like round 2):
  – If p1 has (ever) decided 0, broadcasts “decide 0”, and similarly for 1.
  – Else broadcasts “ready”.
  – Any undecided process who receives “decide()” decides accordingly.
  – Any process who receives “ready” becomes ready.
  – Now p2 decides 1 if it hasn’t already decided.

• Round 6 (like round 3):
  – If p2 has decided 1, broadcasts “decide 1”.
  – Anyone else who receives “decide 1” decides 1.

• Continue with subsequent rounds for p3, p4,…
Correctness

- Key invariants still hold:
  - If any process is in *ready* or *dec-1*, then all processes have initial value 1.
  - If any process is in *dec-0* then:
    - No process is in *dec-1*, and no non-failed process is *ready*.
  - If any process is in *dec-1* then:
    - No process is in *dec-0*, and no non-failed process is *uncertain*.

- Imply agreement, validity
- Strong termination:
  - Because eventually some coordinator will finish the job (unless everyone fails).
Complexity

• Time until everyone decides:
  – Normal case 3
  – Worst case 3n

• Messages until everyone decides:
  – Normal case $O(n)$
    • Technicality: When can processes stop sending messages?
  – Worst case $O(n^2)$
Practical issues for 3-phase commit

• Depends on strong assumptions, which may be hard to guarantee in practice:
  – Synchronous model:
    • Could emulate with approximately-synchronized clocks, timeouts.
  – Reliable message delivery:
    • Could emulate with acks and retransmissions.
    • But if retransmissions add too much delay, then we can’t emulate the synchronous model accurately.
    • Leads to unbounded delays, asynchronous model.
  – Accurate diagnosis of process failures:
    • Get this “for free” in the synchronous model.
    • E.g., 3-phase commit algorithm lets process that doesn’t hear from another process i at a round conclude that i must have failed.
    • Very hard to guarantee in practice: In Internet, or even a LAN, how to reliably distinguish failure of a process from lost communication?

• Other consensus algorithms can be used for commit, including some that don’t depend on such strong timing and reliability assumptions.
Paxos consensus algorithm

- A more robust consensus algorithm, could be used for commit.
- Tolerates process stopping and recovery, message losses and delays,…
- Runs in partially synchronous model.
- Based on earlier algorithm [Dwork, Lynch, Stockmeyer].
- Algorithm idea:
  - Processes use unreliable leader election subalgorithm to choose coordinator, who tries to achieve consensus.
  - Coordinator decides based on active support from majority of processes.
  - Does not assume anything based on not receiving a message.
  - Difficulties arise when multiple coordinators are active---must ensure consistency.

- Practical difficulties with fault-tolerance in the synchronous model motivate moving on to study the asynchronous model (next time).
Next time…

• Modeling asynchronous systems
• Reading: Chapter 8