6.852: Distributed Algorithms
Fall, 2015

Lecture 8

Today's plan

- Lower bound on number of rounds for agreement, cont'd.
- Early-stopping agreement.
- Other consensus-type problems:
  - k-agreement
  - Distributed commit
- Reading:
  - [Aguilera, Toueg]
  - [Keidar, Rajsbaum]
  - Chapter 7 (skim 7.2)
- Next:
  - Modeling asynchronous systems
  - I/O automata
- Reading:
  - Chapter 8

Lower bound on number of rounds for agreement

Lower bound on number of rounds

- f+1 rounds are needed in the worst-case, for either Byzantine agreement or just stopping agreement.

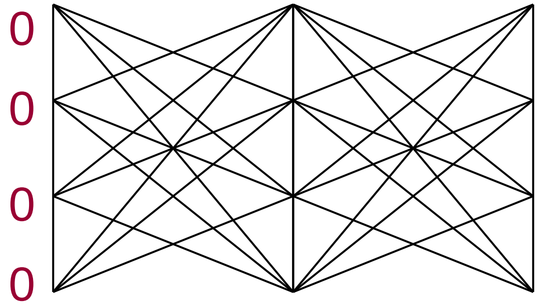
- Assume an f-round stopping agreement algorithm A tolerating f faults, get a contradiction.
- Assume:
  – n-node complete graph.
  – Decisions at end of round f.
  – V = {0,1}
  – All-to-all communication at every round.

Special case: $f = 1$

- Theorem 5: Suppose $n \geq 3$. There is no n-process 1-fault stopping agreement algorithm in which nonfaulty processes always decide at the end of round 1.
- Proof:
  - Construct a chain of executions, each with ≤ 1 failure, such that:
    - First has decision value 0.
    - Last has decision value 1.
    - Any two consecutive executions in the chain are indistinguishable to some process i that is nonfaulty in both. So i must decide the same in both executions, and the two must have the same decision values.
  - So decision values in first and last executions must be the same.
  - Contradiction.

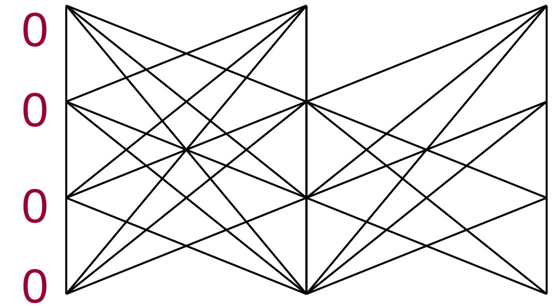
Special case: f = 2

- Theorem 6: Suppose $n \geq 4$. There is no n-process 2-fault stopping agreement algorithm in which nonfaulty processes always decide at the end of round 2.
- Proof:
  - Construct a chain of executions, each with ≤ 2 failures.
  - Start with α_0: All processes have input 0, no failures
  - Work toward α_n, all 1's, no failures.
  - Each consecutive pair is indistinguishable to some nonfaulty process.
  - Use intermediate executions α_i in which:
    - Processes 1,…,i have initial value 1.
    - Processes i+1,…,n have initial value 0.
    - No failures.



Special case: f = 2

- WLOG, show how to connect α_0 and α_1, that is, change p1's initial value from 0 to 1.
- Start with α_0, work toward killing p1 at the beginning, by removing messages.
- Change p1's initial value.
- Then replace messages, working back up to α_1.
- Start by removing p1's round 2 messages, one by one.
- Can't continue by removing p1's round 1 messages, because consecutive executions would not look the same to anyone.
- E.g., removing $1 \rightarrow 2$ at round 1 allows p2 to tell everyone about the failure, at round 2.
- So, use many steps of the chain to remove the round 1 message from p1 to p2.
- In these steps, both p1 and p2 are faulty.



Removing p1's round 1 messages

- Start with execution where p1 sends to everyone at round 1, and to no one at round 2. Only p1 is faulty.
- Remove round 1 message $1 \rightarrow 2$:
  - p2 starts out nonfaulty, so sends all its round 2 messages.
  - Now fail p2, and remove its round 2 messages, one by one, until we reach an execution where $1 \rightarrow 2$ at round 1, but p2 sends no round 2 messages.
  - Now remove the round 1 message $1 \rightarrow 2$.
    - Executions look the same to everyone but p1 and p2.
  - Replace round 2 messages from p2, one by one, until p2 is no longer faulty.
- Repeat to remove p1's round 1 messages to p3, p4,…
- After removing all of p1's round 1 messages, change p1's initial value from 0 to 1.

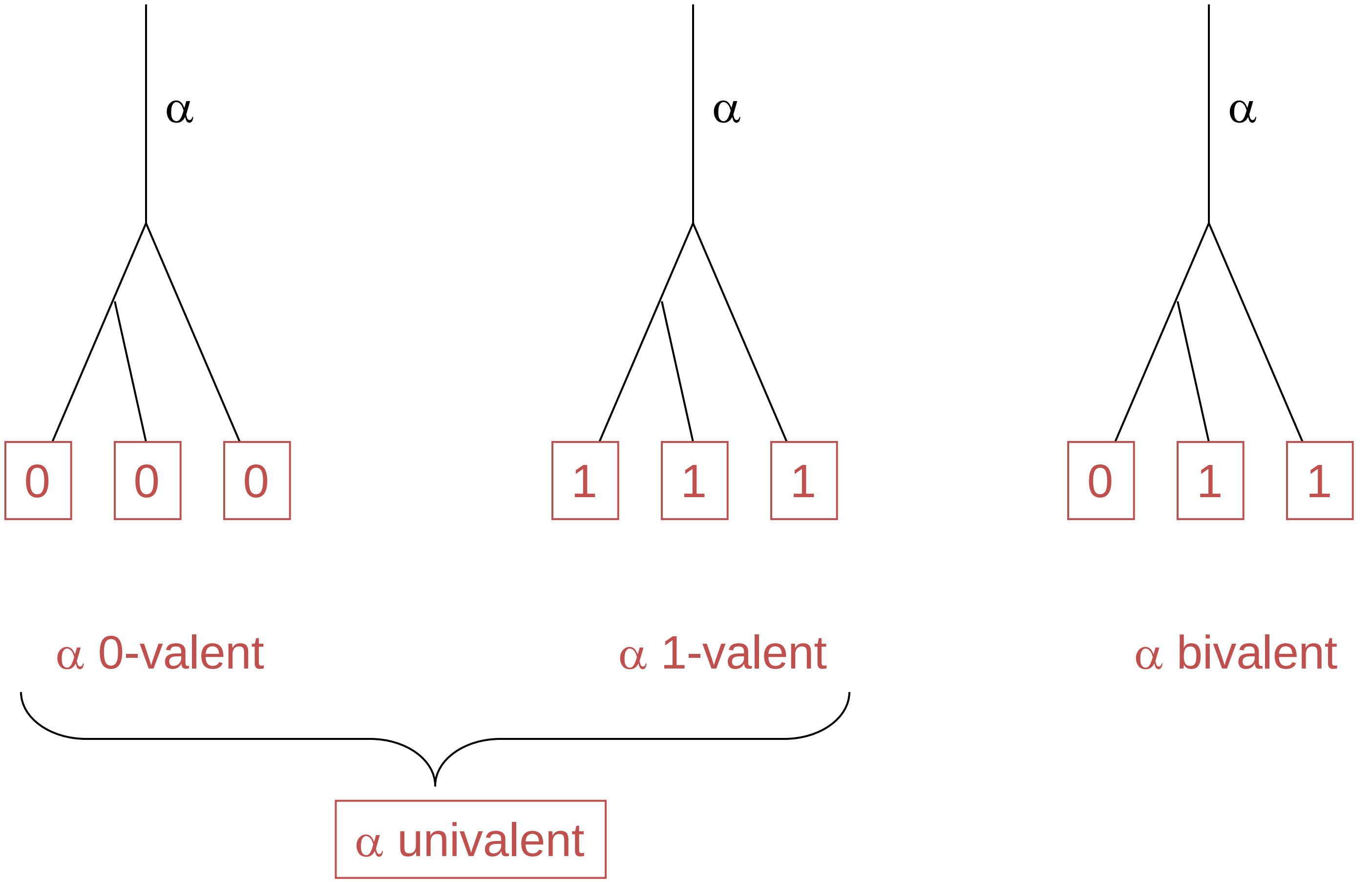
General case: Any f

- Theorem 7: 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.
- Proof:
  - Same ideas, longer chain.
  - 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.
- Alternative proof [Aguilera, Toueg]:
  - Uses ideas from [Fischer, Lynch, Paterson] impossibility of consensus (which you will see later).
  - They assume strong validity, but their proof works for our weaker validity condition also.

[Aguilera,Toueg] lower bound proof

- 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 a process to send any subset of the messages, or to send all but halt before changing state.
- Regard vector of initial values as a 0-round execution.
- Definitions (adapted from [FLP]): α, 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 α.
  - 1-valent, if 1 is the only decision that can occur in…
  - Univalent, if α is either 0-valent or 1-valent (essentially decided).
  - Bivalent, if both decisions occur in some extensions (undecided).

Univalence and Bivalence



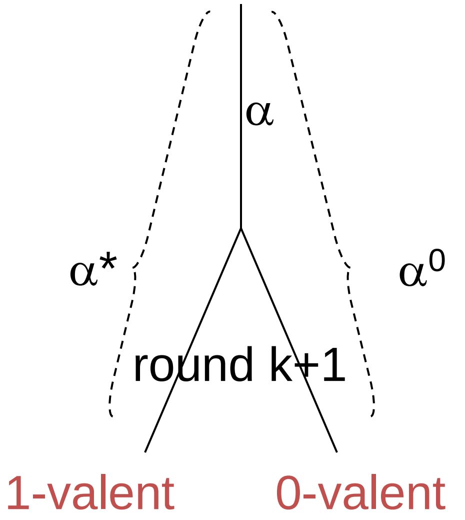
Initial bivalence

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

- Proof (derived 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.

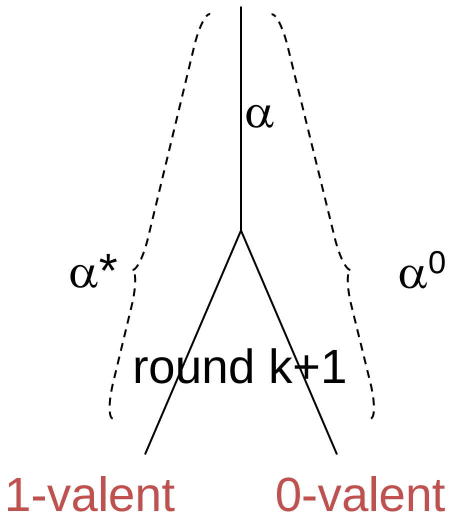
Bivalence through f-1 rounds

- Lemma 2: For every k, $0 \leq k \leq f\text{-}1$, there is a bivalent k-round execution.
- Proof: By induction on k.
  - Base (k=0): Lemma 1.
  - Inductive step: Assume for k, show for k+1, where k < f -1.
    - Assume a bivalent k-round execution α.
    - Assume for contradiction that every 1-round extension of α (with at most one new failure) is univalent.
    - Let α^* be the 1-round extension of α in which no new failures occur in round k+1.
    - By assumption, this is univalent, say WLOG that it's 1-valent.
    - Since α is bivalent, there must be another 1-round extension of α, α^0, that is 0-valent.



Bivalence through f-1 rounds

- In α^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, α^0, α^1, $\alpha^2, \ldots, \alpha^m$.
- Each α^l in this sequence is the same as α^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 α^l is univalent, by assumption.
- Since α^0 is 0-valent, either:
  - At least one of these is 1-valent, or
  - All are 0-valent.

Case 1: At least one α^l is 1-valent

- Then there must be some l such that α^{l-1} is 0-valent and α^l is 1-valent.
- But α^{l-1} and α^l differ after round k+1 only in the state of one process, j_l.
- We can extend both α^{l-1} and α^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 α^l is 0-valent

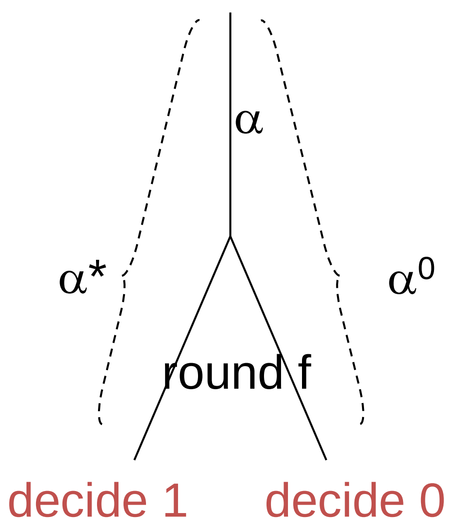
- Then compare:
  - α^m, in which i sends all its round k+1 messages and then fails, with
  - α^* , in which i sends all its round k+1 messages and does not fail.
- No other differences, since only i fails at round k+1 in α^m.
- α^m is 0-valent and α^* is 1-valent.
- Extend to full f-round executions:
  - α^m, by allowing no further failures,
  - α^*, 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.

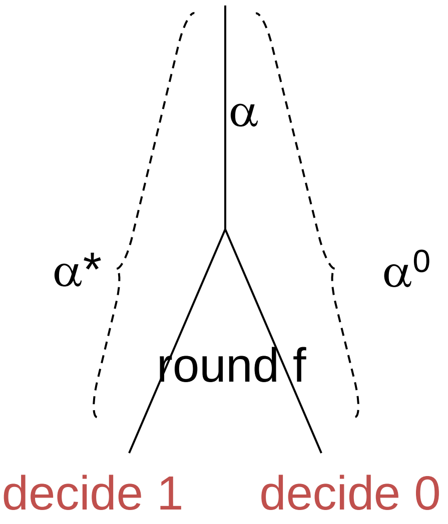
Disagreement after f rounds

- 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 α with $\leq$ f-1 failures.
  - In every 1-round extension of α, everyone who hasn't failed must decide (and agree).
  - Let α^* be the 1-round extension of α in which no new failures occur in round f.
  - Everyone who is still alive decides after α^*, and they must decide the same thing. WLOG, say they decide 1.
  - Since α is bivalent, there must be another 1-round extension of α, say α^0, in which some nonfaulty process (and so, all nonfaulty processes) decide 0.



Disagreement after f rounds

- In α^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 α^1 is identical to α^0 except that i sends to j in α^1 (it might not in α^0).
- Then $\alpha^1 \sim^k \alpha^0$.
- Since k decides 0 in α^0, k also decides 0 in α^1.
- Also, $\alpha^1 \sim^j \alpha^*$.
- Since j decides 1 in α^*, j also decides 1 in α^1.
- Yields disagreement in α^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, but in executions with $f' < f$ failures, terminate correspondingly faster.
- [Dolev, Reischuk, Strong 90] :
- Stopping agreement algorithm in which all nonfaulty processes terminate within $\min(f' + 2, f + 1)$ rounds:
  - Always decide within $f + 1$ rounds.
  - If $f' + 2 \leq f$, decide "early", within $f' + 2$ rounds.

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

- Theorem 1: 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 $f' + 1$.

Special case: $f' = 0$

- Special case Theorem 2: 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 α be an execution that completes some finite number (possibly 0) of rounds. Then $val(\alpha)$ is the unique decision value in the extension of α with no new failures.
  - Different from bivalence definitions from [Aguilera, Toueg] ---now consider value in just one extension.

Special case: $f' = 0$

- Theorem 2: Assume $2 \le f < n$. 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: $val(\alpha)$ is the decision value in the extension of α with no new failures.
- Proof of Theorem 2:
  - Consider 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 the end of round 1, in all failure-free executions.
  - $val(000 \ldots 0) = 0, val(111 \ldots 1) = 1.$
  - So there must be two 0-round executions α^0 and α^1, that differ in the value of just one process i, such that $val(\alpha^0) = 0$ and $val(\alpha^1) = 1$.

Special case: $f' = 0$

- 0-round executions α^0 and α^1, differing only in the initial value of process i, such that val(α^0) = 0 and val(α^1) = 1.
- In the failure-free extensions of α^0 and α^1, all nonfaulty processes decide by the end of round 1.
- Define:
  - β^0, 1-round extension of α^0, in which process i fails, sends only to j.
  - β^1, 1-round extension of α^1, in which process i fails, sends only to j.
- Then:
  - β^0 looks to j like ff extension of α^0, so j decides 0 in β^0 by round 1.
  - β^1 looks to j like ff extension of α^1, so j decides 1 in β^1 by round 1.
- β^0 and β^1 are indistinguishable to all processes except i, j.
- Define:
  - γ^0, infinite extension of β^0, in which process j fails right after round 1.
  - γ^1, infinite extension of β^1, in which process j fails right after round 1.
- By agreement, all nonfaulty processes must decide 0 in γ^0, 1 in γ^1.
- But γ^0 and γ^1 are indistinguishable to all nonfaulty processes, so they can't decide differently, contradiction.

k-Agreement

k-agreement

- Also 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 a 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 minimum 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 (LTTR).
- k-agreement: $\lfloor f/k \rfloor + 1$ 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 round $r+1$ is someone's min value after round 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 round r.
- Proof: Show the contrapositive.
  - Suppose that $|M(r)| > d$, show at least d processes must 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 that is active after $r - 1$ rounds and that has $min = m'$ just after $r - 1$ rounds.
  - Claim i fails during round r:
    - If not, then everyone would receive m' in round r.
    - But then 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!

Rounds for k-agreement

- Theorem 1: $FloodMin$, for $\lfloor f/k \rfloor + 1$ rounds, solves k-agreement.
- This is a tight bound!
- Theorem 2: Any algorithm for k-agreement requires $\geq \lfloor f/k \rfloor + 1$ rounds.

Lower Bound (sketch)

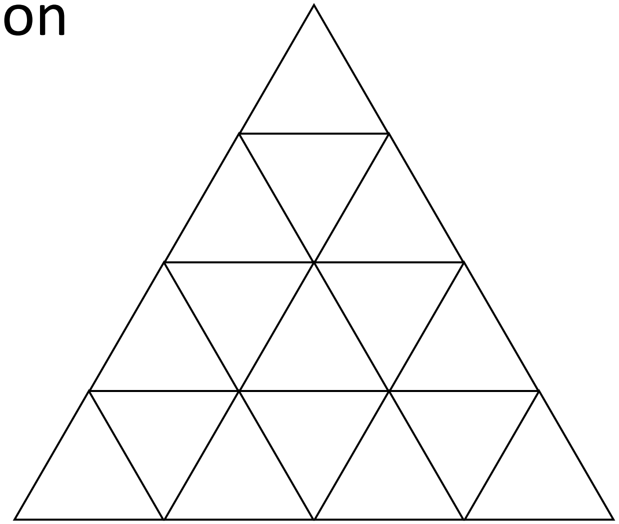
- Theorem 2: Any algorithm for k-agreement requires $\geq \lfloor f/k \rfloor + 1$ rounds.
- Recall old proof for $f + 1$-round lower bound for 1-agreement.
  - Chain of executions for assumed algorithm:

    α_0 ----- α_1 ----- …-----α_j -----α_{j+1}----- …-----α_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.
- This 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

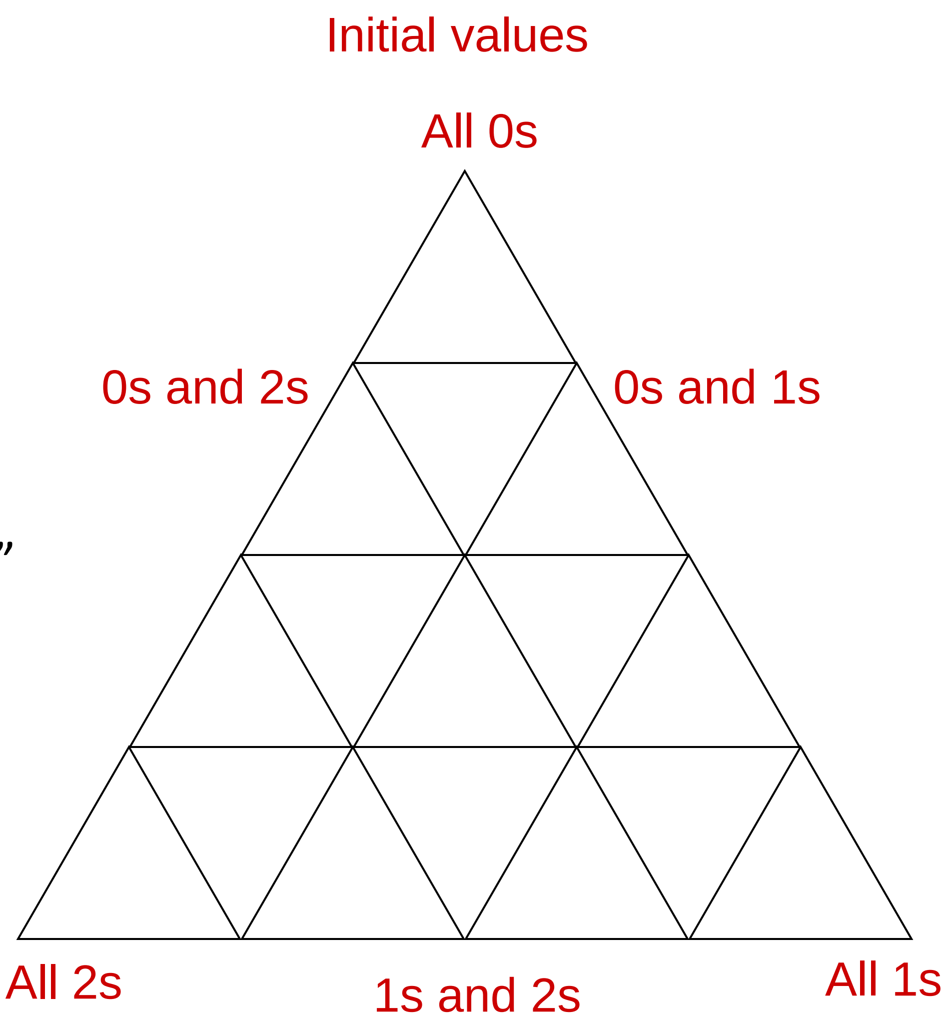
- Assume, for contradiction:
  - An n-process k-agreement algorithm tolerating f failures.
  - $n \geq f + k + 1$ (so each execution we consider has $\geq k + 1$ nonfaulty processes)
  - All-to-all communication at all rounds.
  - $V = \{0,1, \dots, k\}$, $k + 1$ values.
  - All processes decide just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

- Get a contradiction by finding an execution with $k + 1$ different decision values.
- Use a 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 must vanish 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 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; indices chosen for the corners of any tiny triangle (simplex) are distinct.
- Consistency property: For every tiny triangle T, there is a single execution β, with at most f faults, that is "compatible" with the executions and processes labeling the corners of T:
  - All the corner-labeling processes are nonfaulty in β.
  - If (α, i) labels some corner of T, then α is indistinguishable from β by i.
- Formalizes the "gradual morphing" property.
- Proof by laborious, detailed construction.
- Can recast chain arguments for 1-agreement in this style:

$$\begin{array}{cccccccccccccc} & & & & & & & \beta & & & & & & \\ \alpha_0 & \text{-----} & \alpha_1 & \text{-----} & \ldots & \text{-----} & \alpha_j & \text{-----} & \alpha_{j+1} & \text{-----} & \ldots & \text{-----} & \alpha_m \\ p_0 & & p_1 & & \ldots & & p_j & & p_{j+1} & & & & p_m \end{array}$$

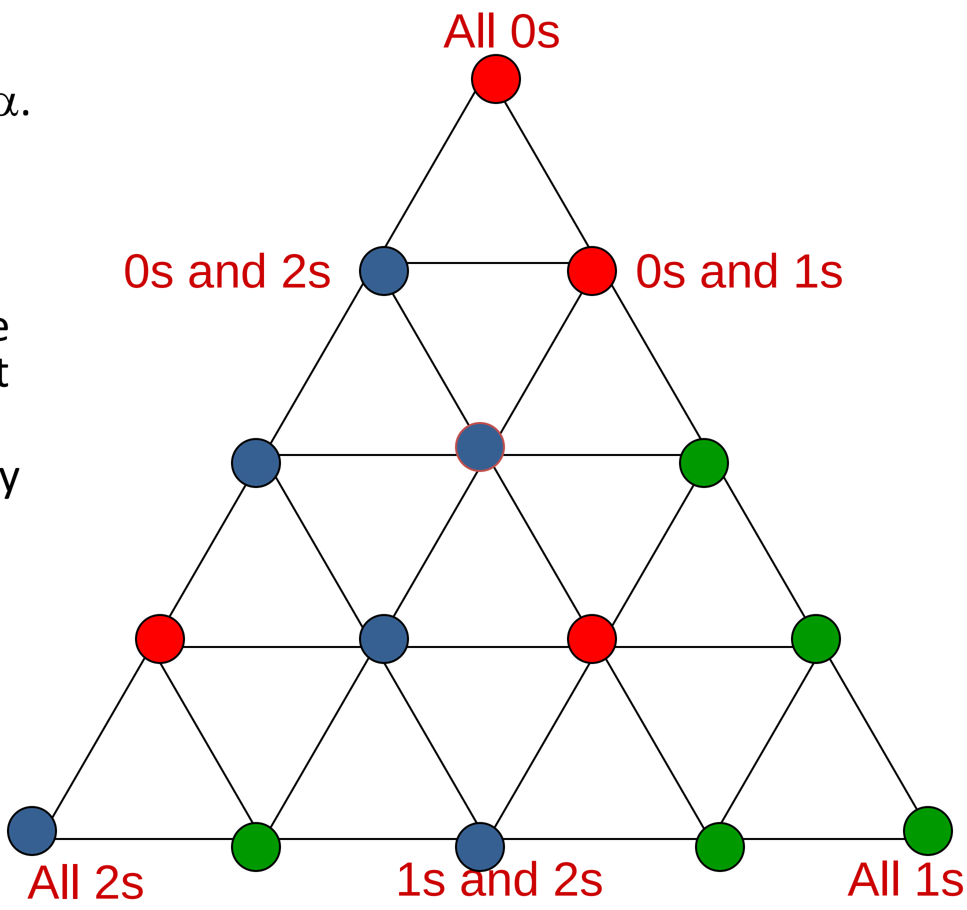
  - β indistinguishable by p_j from α_j
  - β indistinguishable by p_{j+1} from α_{j+1}

Bound on rounds

- This labeling construction uses the assumption that $r \leq \left\lfloor \frac{f}{k} \right\rfloor$, that is, $f \geq r\, k$.
- How:
  - We are essentially constructing chains simultaneously in k directions (2 directions, in the 2-dimensional case).
  - We use r failures (one per round) to construct the "chain" in each direction.
  - For k directions, that's $r\, k$ total failures.
- Details LTTR (see book, or paper [Chaudhuri, Herlihy, Lynch, Tuttle])

Coloring the nodes

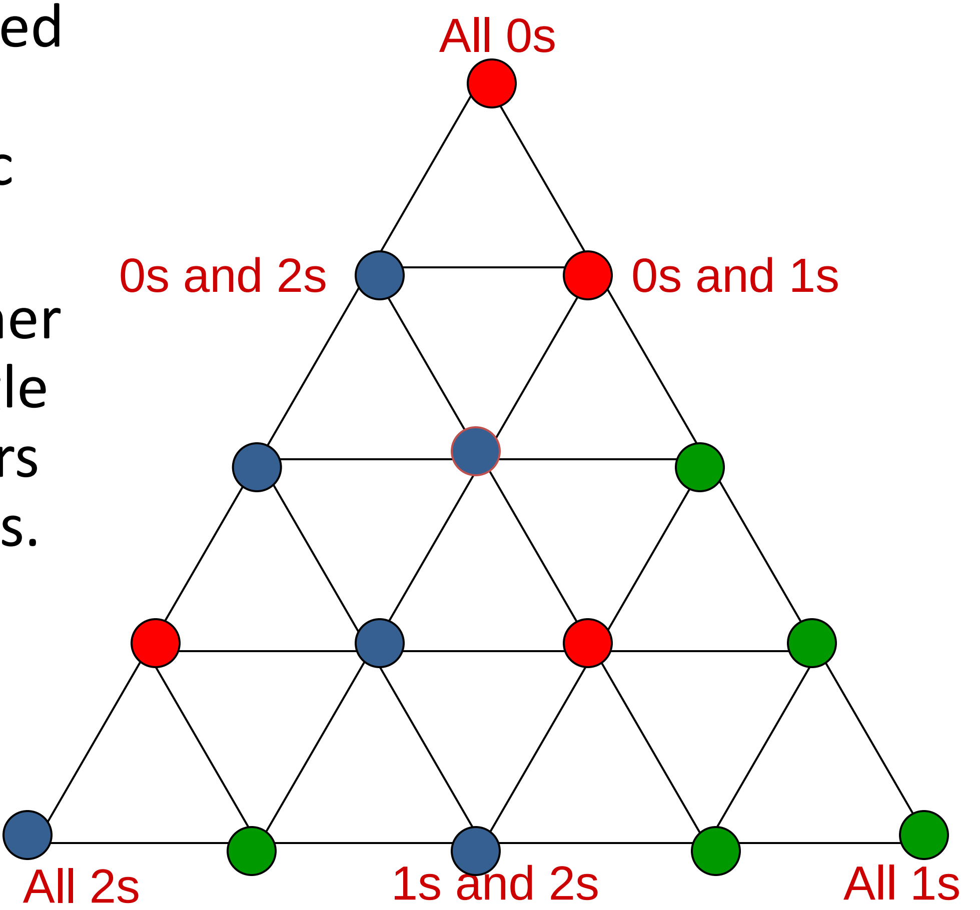
- Now color each node v with a "color" in $\{0, 1, \ldots, k\}$:
  - If v is labeled with (α, i) then $color(v) = i$'s decision value in α.
- Properties:
  - Colors of the major corners are all different.
  - Color of each boundary edge node is the same as one of the endpoint corners.
  - For $k > 2$, 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 recall that there must be a single execution β, 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 β.
  - If (α, i) labels some corner of T, then α is indistinguishable from β by i.
- So all the corner processes behave the same in β 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

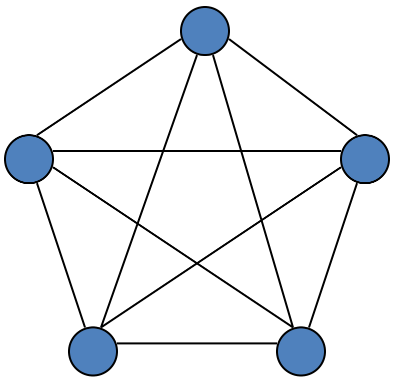
Approximate Agreement problem

- Agreement on real number values, e.g.:
  - Readings of several altimeters on an aircraft.
  - Values of approximately-synchronized clocks.
- Consider Byzantine participants, e.g., faulty hardware.
- Abstract approximate agreement problem:
  - Inputs, outputs are reals
  - Agreement: Within ϵ.
  - Validity: Within range of initial values of nonfaulty processes.
  - Termination: Nonfaulty processes eventually decide.
- Assume: 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
  - Convergence strategy
  - Extend to asynchronous settings, whereas BA is unsolvable in asynchronous networks (as we will see).

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.



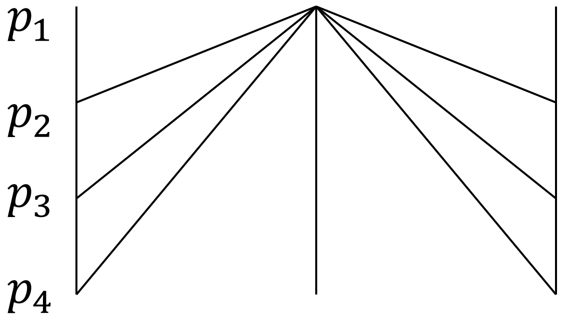
- Assume:
  - Process stopping failures only.
  - n-node, complete, undirected graph.
- Require:
  - Agreement: No two processes decide differently (faulty or not, uniformity)
  - Validity:
    - If any process starts with 0 (abort) then 0 is the only allowed decision.
    - If all start with 1 (commit) 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): (Even if there are failures), all nonfaulty processes eventually decide.

2-Phase Commit

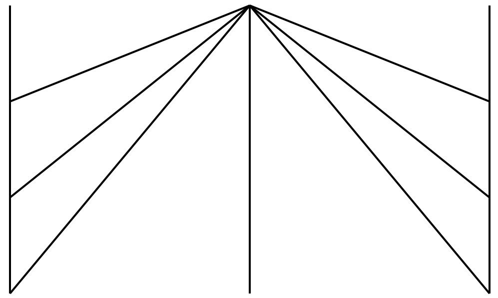
- Traditional, blocking algorithm (guarantees weak termination only).
- Assumes distinguished process 1, acts as “coordinator” (leader).
- Round 1: All send initial values to process 1, who decides.
  - If it sees 0, or doesn’t hear from someone, it decides 0; otherwise it decides 1.
- Round 2: Process 1 sends the decision to everyone else.

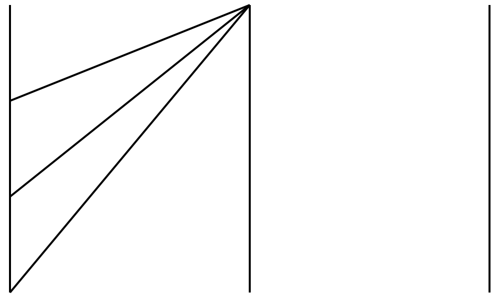


- Q: When can the processes decide?
- Anyone with initial value 0 can decide at the beginning.
- Process 1 decides after receiving round 1 messages.
- Everyone else decides after round 2 (if there are no failures).

Correctness of 2-Phase Commit

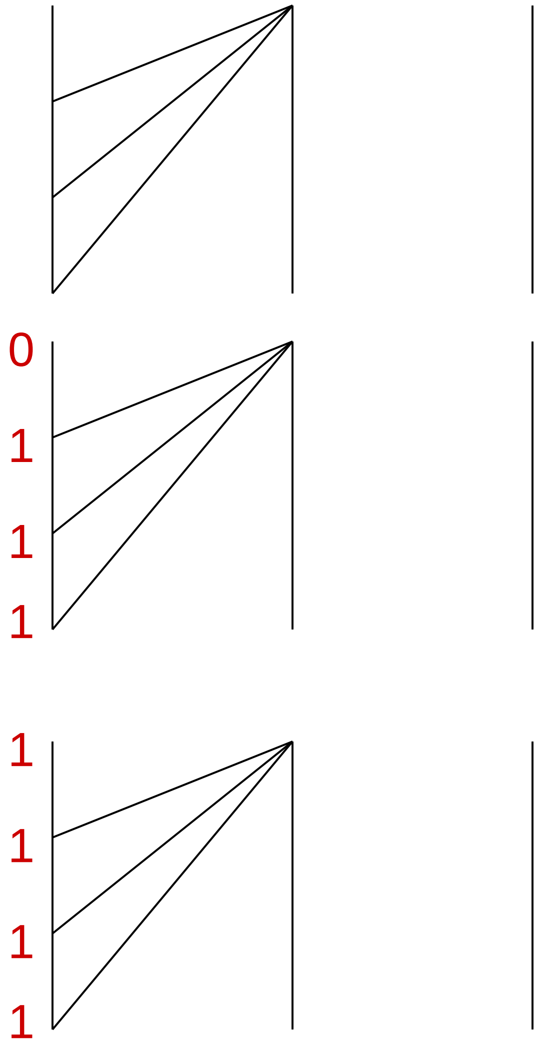
- Agreement:
  - Because decision is centralized (and consistent with any individual initial decisions).
- Validity:
  - Because of how the coordinator decides.
- Weak termination:
  - If no one fails, everyone terminates by end of round 2.
- Strong termination?
  - No: If coordinator fails before sending 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, everyone is required to 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:
    - Process 1 will decide 1.
    - By agreement, others must decide 1.
  - But the other processes can't distinguish these two situations.

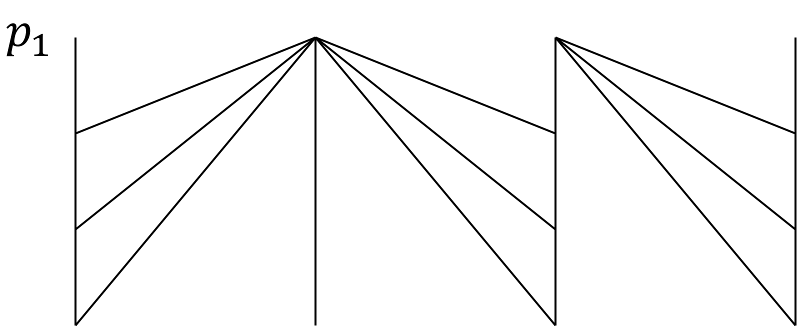


Complexity of 2-phase commit

- Time:
  - 2 rounds
- Communication:
  - At most $2n$ messages

3-Phase Commit [Skeen]

- Yields strong termination.
- Trick: Introduce intermediate stage, before actually deciding.
- Process states are now classified into four categories:
  - $dec0$: Already decided 0.
  - $dec1$: Already decided 1.
  - $ready$: Ready to decide 1 but hasn't yet.
  - $uncertain$: Otherwise.
- Again, process 1 acts as "coordinator".
- Communication pattern:

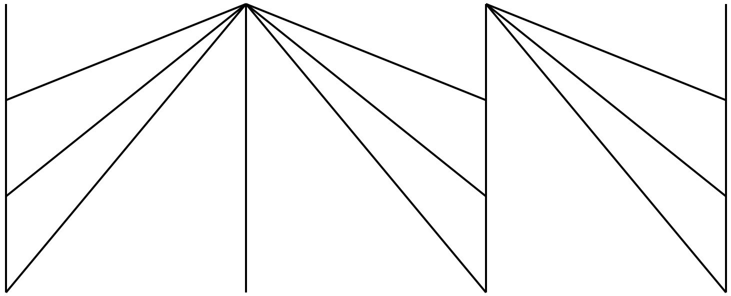


3-Phase Commit

- All processes are initially *uncertain*.
- Round 1:
  - All other processes send their initial values to p_1.
  - All with initial value 0 decide 0 (and enter $dec0$ state)
  - If p_1 receives 1s from everyone and its own initial value is 1, p_1 becomes *ready*, but doesn't yet decide.
  - If p_1 sees 0 or doesn't hear from someone, p_1 decides 0.

- Round 2:
  - If p_1 has decided 0, it broadcasts "decide 0", else it broadcasts "ready".
  - Anyone else who receives "decide 0" decides 0.
  - Anyone else who receives "ready" becomes *ready*.
  - Now p_1 decides 1 if it hasn't already decided.

- Round 3:
  - If p_1 has decided 1, it 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 $dec1$, then all processes have initial value 1.
  - If any process is in $dec0$ then:
    - No process is in $dec1$, and no non-failed process is $ready$.
  - If any process is in $dec1$ then:
    - No process is in $dec0$, and no non-failed process is $uncertain$.

- Proof: LTTR.
  - Key step: Third condition is preserved when p_1 decides 1 after round 2.
  - In this case, p_1 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 the third condition.

- Note critical use of synchrony here:
  - p_1 infers that non-failed processes are $ready$ just because round 2 is completed.
  - Without synchrony, this would require explicit 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 status ($dec0, uncertain, ready, dec1$) to p_2.
  - If p_2 receives any $dec0$'s and hasn't already decided, then p_2 decides 0.
  - If p_2 receives any $dec1$'s and hasn't already decided, then p_2 decides 1.
  - If all received values, and its own value, are $uncertain$, then p_2 decides 0.
  - Otherwise (all values are $uncertain$ or $ready$ and at least one is $ready$), p_2 becomes $ready$, but doesn't decide yet.

- Round 5 (analogous to round 2):
  - If p_2 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 p_2 decides 1 if it hasn't already decided.

- Round 6 (analogous to round 3):
  - If p_2 has decided 1, broadcasts "decide 1".
  - Anyone else who receives "decide 1" decides 1.

- Continue with subsequent rounds for $p_3, p_4, \ldots$

Correctness

- Key invariants still hold:
  - If any process is in $ready$ or $dec1$, then all processes have initial value 1.
  - If any process is in $dec0$ then:
    - No process is in $dec1$, and no non-failed process is $ready$.
  - If any process is in $dec1$ then:
    - No process is in $dec0$, 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

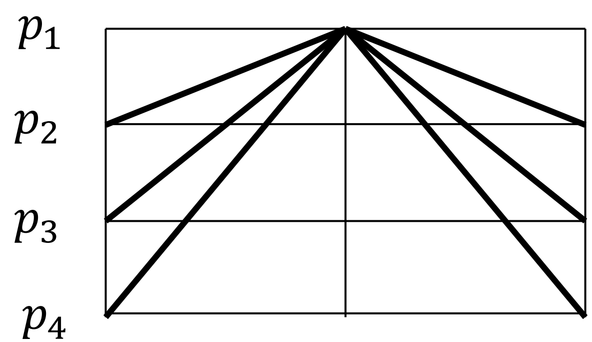
- Depends on strong assumptions, which may be hard to guarantee in practice:
  - Synchronous model:
    - Could emulate with approximately-synchronized clocks, timeouts.
  - Reliable message delivery:
    - Could emulate with acks and retransmissions.
    - But if retransmissions add too much delay, 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 [Lamport]

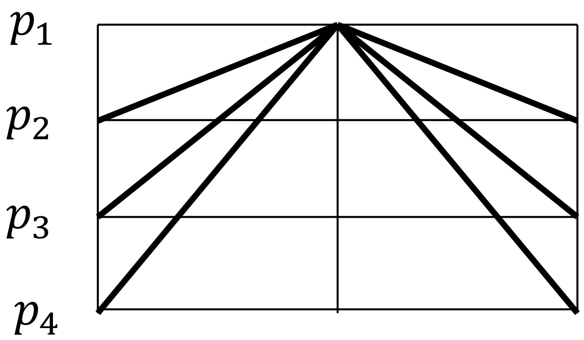
- A more robust consensus algorithm, can be used for commit.
- Tolerates process stopping and recovery, message losses and delays,…
- Runs in partially synchronous model.
- Similar to algorithm by [Dwork, Lynch, Stockmeyer].
- Algorithm idea:
  - Processes use an unreliable leader election subalgorithm to choose a coordinator, who tries to achieve consensus.
  - Coordinator decides based on active support from a majority of the processes.
  - Does not assume anything based on not receiving a message.
  - Subtleties 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 (start this next time).

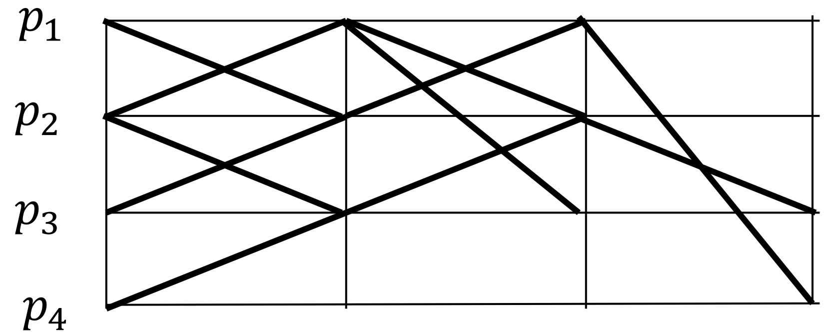
A Lower Bound for Commit

- How many messages are needed to solve the commit problem?
- Theorem [Dwork, Skeen]: Any algorithm that solves the commit problem, even with weak termination, uses at least $2n - 2$ messages in the failure-free execution α in which all inputs are 1.
- Note: That's what 2-phase commit uses, so 2-phase commit is "optimal".
- Proof considers the communication pattern for α:



Information flow in a communication pattern

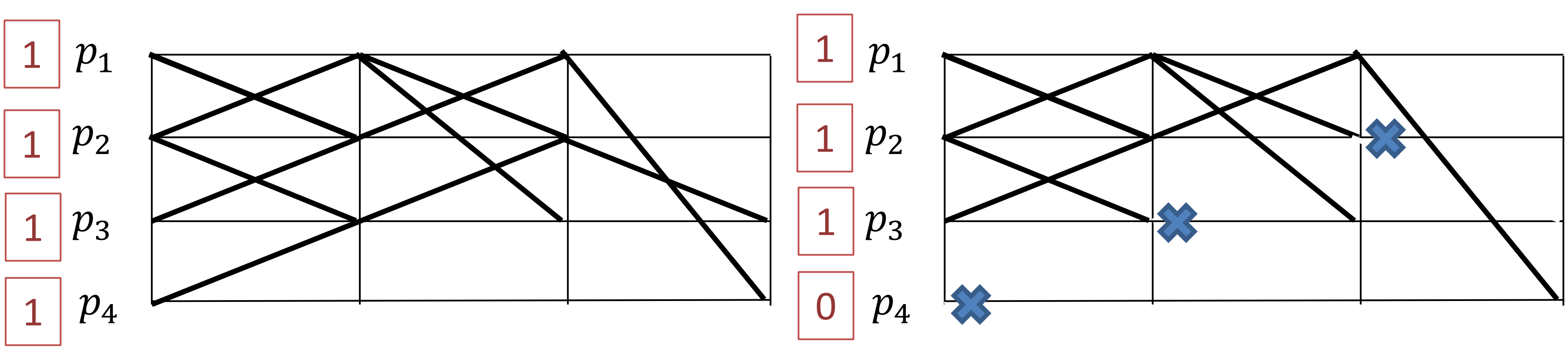




- i affects j in a pattern if there is a path in the pattern from i at time 0 to j at some time.
- In Pattern 1, all processes affect all processes.
- In Pattern 2, 4 does not affect 1.
- Lemma: In the failure-free, all-1-input run α, every i affects every j in the communication pattern of α.
- Corollary: The communication pattern of α has at least $2n - 2$ edges.

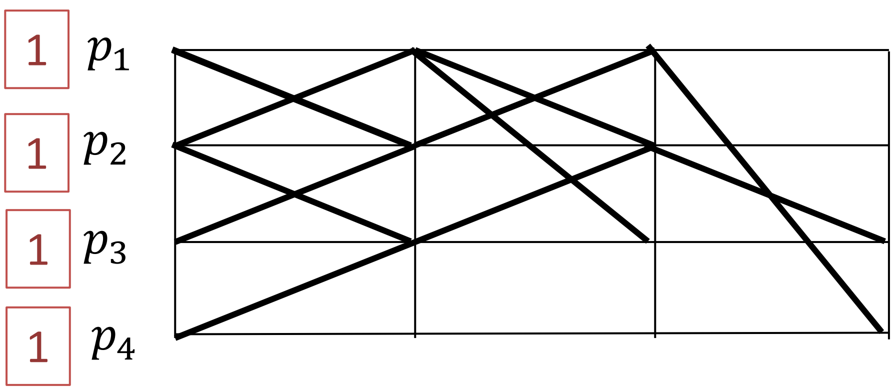
Proof of the Lemma

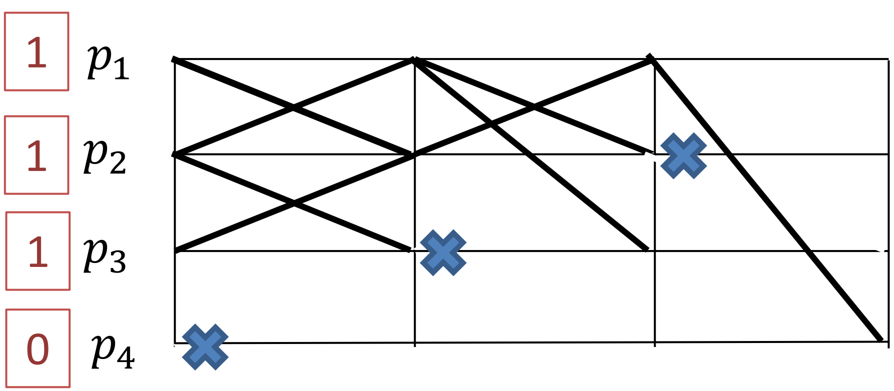
- Lemma: In the failure-free, all-1-input run α, every i affects every j in the communication pattern of α.
- Proof:
  - By contradiction. Suppose i does not affect j (for some particular i, j).
  - Then $i \neq j$.
  - Construct execution α', which is the same as α except that:
    - i's input is 0, and
    - Every process that is affected by process i in α fails just after it first gets affected by process i in α.
- Example: Process 4 does not affect process 1.



Proof of the Lemma

- Lemma: In the failure-free, all-1-input run α, every i affects every j in the communication pattern of α.
- Proof, cont'd:
  – Construct execution α':
    - i's input is 0, and
    - Every process that is affected by process i in α fails just after it first gets affected by process i in α.
  – In α, all processes eventually decide 1.
  – α' is indistinguishable from α to process j.
  – So process j decides 1 in α' , which contradicts the requirements.





Next time…

- Modeling asynchronous systems
- I/O automata
- Reading: Chapter 8