There is no protocol that solves BA without authentication if $f \geq \frac{n}{3}$.

- We first show that there is no BA $n = 3, \ f = 1$
- Based on that, we prove the impossibility claim
- For the $n = 3, \ f = 1$, assume possible values are 0 and 1 and assume a solution
Impossibility for $n = 3, f = 1$

Consider the execution $\alpha$ obtained by:

- For $3(0)$ and $2(0)$, $\alpha$ seems like an execution of the triangle with all starting with $v_i = 0$ and 1 being faulty, thus they decide 0.
- Similarly, $3(1)$ and $2(1)$ decide 1.
- And $1(1)$ and $2(1)$ decide 1.
- And $3(0)$ and $1(1)$ decide ... ??? (They do have to reach agreement!!)
From $n = 3, \ f = 1$ to General Case

**Theorem.** There is no solution to BA for $2 \leq n \leq 3 \cdot f$

**Proof.** Assume a solution to BA with $3 \leq n \leq 3f$ and construct a solution for triangle with one fault. Partition the $n$ process into $P_1, P_2, P_3$, each with $\leq f$ members; $i$ will mimic the behavior of $P_i$ with all members starting with $v^i_j = v_i$:

- $i$’s local state consists of the states of all $P_i$’s members and every message sent among $P_i$’s members
- for every message sent from $P_i$ to $P_j$, $i$ sends $j$ same message (while projecting every “$P_k$” onto “$k$”)
- when some $p \in P_k$ decides $v$, so does $k$

**Claim.** New system correctly solves BA for three processes!
Correctness of New System

Claim. New system correctly solves BA for three processes

Proof. Let:
- \( \alpha \): execution of triangle with \( f \geq 1 \)
- \( \alpha' \): corresponding original system execution (\( f \geq \frac{n}{3} \))

Termination. Let \( j \) be a correct (triangle) proc. \( j \) simulates at least one correct (original) \( j' \) who satisfies termination.

Validity. Similar.

Agreement. Similar.
Bounding $f$ for Authenticated BA

- can attain authenticated BA with \( \text{any } f < n \)
- no solution \( f \geq \frac{n}{3} \)

Contradiction??
Weak and Strict BA

- In *Strict* BA, if *correct* processes start with $v$ then $v$ is only possible decision
- In *Weak* BA, if *all* start with $v$ then $v$ is the only possible decision

Use strict notion for possibility, and weak notion for impossibility!

- We showed impossibility results for the strict notion (easier)
- For possibility, we focused on the weak notion (usually protocols can be easily made to attain strict BA; authenticated BA protocol is an exception)
**Bounding Number of Rounds**

**Assumptions**

- A full communication graph.
- Possible values are 0 and 1
- Decisions are made only at round $f$
- Stopping failures only
- At every round, every process sends a message to every other process

---

*a* simple BA, harder impossibility:
Impossibility of $1$-round ($n > 2$)

**Idea.**

- Consider a sequence of executions $\alpha_0, \ldots, \alpha_k$, $\alpha_0$ has no faults and all start with 0, and $\alpha_k$ has no faults and all start with 1.
- $\alpha_{\ell+1}$ is obtained from $\alpha_{\ell}$ by a removal/addition of a single message.
- Only processes who distinguish between $\alpha_{\ell}$ and $\alpha_{\ell+1}$ are sender/receiver of message.
- Correct processes reach same decision in $\alpha_{\ell}$ and $\alpha_{\ell+1}$.
- Obviously, this leads to a contradiction.
Obtaining the Sequence

- In $\alpha_0$ only decision is 0
- To obtain $\alpha_1$, remove message from 1 to 2
- There is some $j$ s.t. $\alpha_0 \sim j \alpha_1$
- Continue removing 1’s messages (until $\alpha_{n-1}$ is obtained)
- Change 1’s input to 1
- Add 1’s messages, one by one until $\alpha_{2(n-1)}$ is obtained which has no faults
- All correct processes still decide the same value
- Repeat above for processes $2 \ldots, n$
And for 2-rounds . . .

- **Intuition.** Similar to the previous one, only harder. E.g., removing \( p \)'s first round message to \( q \) may impact \( q \)'s messages in second round

- After \( p \)'s round 2 messages are removed, we **remove \( q \)'s round 2 messages** (Okay since \( f = 2 \)).

- Then, remove \( p \)'s round 1 message to \( q \)

- Then, restore \( q \)'s round 2 messages

- And only then remove \( p \)'s round 1 message to \( q \)

- Once done for every \( q \), \( p \)'s corruption is accomplished
To show there is not BA with $\leq f$ rounds for any $1 \leq f \leq n – 2$, assume such a solution, and show how to obtain sequences of executions as above

- We actually showed first two steps of general claim (for last two rounds)

- We need to leave (at least) two processes correct and intact at each step, so we can argue they cannot distinguish between executions (and therefore reach same decision)

- Since proof assumes stopping failures only, the impossibility proof applies for both strict and weak BA
Concurrency Control: coordinating the actions of processes operating in parallel, access *shared data*, and potentially *interfere* with each other. Closely related to

Recovery: ensuring that software and hardware failure do not corrupt *persistent data*

Both arise in the design of hardware and operating, real time, communication, and database systems

transaction: an execution of a program that accesses a shared data
Goal of concurrency control

- guarantee that transactions execute atomically
- a transaction either commits, and then its effects become permanent, or aborts and has no effects
A database: set of named data items with values

Example: To transfer \( x \) in from \( from \) into \( to \) that reside on different machines, DBS can execute:

1. \( t_1 := \text{Read}(\text{from}) \quad \%\% \text{check} \quad t_1 \geq x \)
2. \( t_2 := \text{Read}(\text{to}) \)
3. \( \text{Write} (\text{from}, t_1 - x) \)
4. \( \text{Write} (\text{to}, t_2 + x) \)

- 1 and 2, and 3 and 4 can be performed concurrently
- 1/2 should be performed before 3/4 respectively
- \{1,2,3,4\} should be performed as a single atomic instruction, w/o interference of other transactions
- any transaction accessing \( from \) and \( to \) should either see both as before, or after, the transfer of \( x \), but not in the midst
Scheduler is in charge of deciding which site performs which operations in which order. Its goal is to ensure:

- **Serializability** The effect of each execution on the DBS is as if transactions are performed serially.

- **Recoverability** DBS contains effects of all committed transactions and none of aborted transactions.

- **Cascadelessness** Each transaction reads only values written by committed transactions OR

- **Strictness** Each read and write operation are performed only when all previously issued transactions that write (to same variables) commit or abort. (Guarantees both recoverability and cascadelessness)
Distributed Recovery

Assume

- No replication of data
- A unique DM and scheduler in charge of each data item
- Each distributed transaction $T$ has a home site where it is issued
- $T$ submits the operations to the TM at the home site
- TM forwards operations to other sites: each read and write of $x$ are forwarded to the site where $x$ is stored, and processed by the DM at that site as if $T$ were local
Distributed Recovery

The case of commit/abort is different:

- TM at $T$’s home site should submit the commit/abort to all sites where $T$ accessed data, which is considerably more difficult!
- While the TM at the home site may send commit, the scheduler may decide to abort
- some (but not all) sites involved in the transaction may fail, and the transaction should abort
Atomic Commitment

- Assume $T$ involves $S_1, S_2, \ldots, S_n$ with $S_1$ at $T$’s home site

- Before $S_1$’s TM sends $\text{commit}(T)$ to all sites, it must make sure that scheduler and all DMs are ready (and willing) to commit

- (A scheduler should be ready to commit if $T$ satisfies recoverability at the site. A DM should be ready to commit if all values written by $T$ are on stable storage; it’s not consulted if it only issues reads in $T$.)

- Atomic Commit Protocol (ACP) is an algorithm for coordinator (home site) and participants at whose termination they all commit or abort. Initially, they each have a vote, yes or no.
ACP: Requirements

- AC1 (Agreement) all processes that decide, reach same decision
- AC2 Decisions are irreversible
- AC3 Commit possible only if all vote yes
- AC4 Commit only decision if all processes vote yes and no failures
- AC5 (Termination) if at some point all failures are repaired and no failure occurs for sufficiently long, all processes eventually decide
Note on ACP

- AC5 deals with blocking that is possible under some failure patterns.
- It is possible that all processes vote yes and decide to abort.
- Processes that vote no can trivially decide to abort.
- A process that votes yes is uncertain until it finds out what the decision is. For example, if it is disconnected from all other processes. (In this case, it is “blocked” until faults are repaired.)
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 own 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 VOTE-REQ time-outs, can decide ABORT.

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

- **Round 3**: if a process \( p \) times-out while uncertain, it cannot unilaterally decide; it must invoke a TERMINATION PROTOCOL.
2-Phase Commit (2PC)

Assume no failures!

- **Round 1:** The coordinator send VOTE-REQ
- Every process that receives VOTE-REQ prepares answer; if it’s NO, it decides ABORT and stops
- **Round 2:** All whose vote is YES send it to coordinator
- If 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 VOTE-REQ time-outs, can decide **ABORT**.

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

- **Round 3**: if a process \( p \) times-out while uncertain, it cannot unilaterally decide; it must invoke a **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 ABORTs, 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 YES, decision is COMMIT;
  2. some voted NO (or not voted), decision is ABORT

- $p$ is exactly like a process that times-out at round 3.

- To remember its state, $p$ keeps a DT log in a failure-resistant memory.