How Many Processes can Solve Consensus with One Byzantine Failure?

**Validity:** If all nonfaulty processes have input \( v \), decide \( v \)

- **Two processes?**
  - If \( p_0 \) has input 0 and \( p_1 \) has 1, someone has to change, but not both.

  - **What if one processor is faulty?**
  - **How can the other one know?**

- **Three processes?**
  - If \( p_0 \) has input 0, \( p_1 \) has input 1, and \( p_2 \) is faulty, then a tie-breaker is needed, but \( p_2 \) can act maliciously.
# Processes Lower Bound for $f = 1$

**Theorem:** Any consensus algorithm for one Byzantine failure must have at least four processes.

Suppose in contradiction there is a consensus algorithm for 3 processes and 1 Byzantine failure.

Get two copies:

Rewire the copies.
# Processes Lower Bound for $f = 1$

Rewire the copies and assign inputs
This execution does not have to solve consensus
But it can **specify the behavior of faulty processes**

---

**# Processes Lower Bound for $f = 1$**

acts as if it has 1
acts as if it has 0

$p_0$ and $p_1$ must decide 1
# Processes Lower Bound for $f = 1$

$p_0$ and $p_1$ must decide 1

$p_0$ and $p_1$ decide 1

$p_1$ acts as if it has 1

$p_1$ and $p_2$ must decide 0

$p_2$ acts as if it has 0

$p_0$ and $p_2$ must agree!

$p_0$ and $p_1$ must decide 1
n > 3f for arbitrary f

**Theorem:** Any consensus algorithm for $f$ Byzantine failures must have at least $3f+1$ processes.

**Proof:** By reduction to the 3:1 case.

- Suppose in contradiction there is an algorithm $A$ for $f > 1$ failures and $n = 3f$ total processes
- Use $A$ to construct an algorithm for 1 failure and 3 processors, a contradiction

---

**The Reduction**

Partition the $n \leq 3f$ processes into three sets, $Q_0$, $Q_1$, and $Q_2$, each of size at most $f$

- $p_0$ simulates $Q_0$
- $p_1$ simulates $Q_1$
- $p_2$ simulates $Q_2$

If one process is faulty in the $n = 3$ system, then at most $f$ processes are faulty in the simulated system

$\Rightarrow$ The simulated system is correct.

The processes in the $n = 3$ system decide as the simulated processes

$\Rightarrow$ Their decisions are correct
Tree Algorithm

- This algorithm uses
  - $f + 1$ rounds (optimal)
  - $n = 3f + 1$ processors (optimal)
  - exponential size messages (very bad)
- Each process keeps a local tree data structure
- Values are filled in the tree during the $f + 1$ rounds
- Then, the decision is calculated from the tree values

Local Tree Data Structure

Each node is labeled with a sequence of unique process identifiers
- Root's label is the empty sequence $\lambda$; its level is 0
- Root has $n$ children, labeled 0 .. $n - 1$

$n = 4$, $f = 1$
Local Tree Data Structure

Child node labeled $i$ has $n - 1$ children, labeled $i : 0 .. i : n-1$ (skipping $i : i$)

Node at level $d$ labeled $v$ has $n - d$ children, labeled $v : 0 .. v : n-1$ (skipping any index in $v$)

Nodes at level $f + 1$ are leaves

$n = 4$, $f = 1$

Filling in the Tree Nodes

• Initially store your input in the root (level 0)
• Round 1:
  – send level 0 of your tree to all
  – store value $x$ received from each $pj$ in tree node labeled $j$ (level 1); use a default if necessary
  – "pj told me that pj 's input was x"
• Round 2:
  – send level 1 of your tree to all
  – store value $x$ received from each $pj$ for each tree node $k$ in tree node labeled $k : j$ (level 2); use a default if necessary
  – "pj told me that pk told pj that pk's input was x"
• Continue for $f + 1$ rounds
Calculating the Decision

- In round $f + 1$, each process uses the values in its tree to compute its decision.
- Recursively compute $\text{resolve}(\lambda)$ for the root, based on the "resolved" values for its children.
Calculating the Decision

\[
\text{resolve}(\pi) = \begin{cases} 
\text{value in tree node labeled } \pi \text{ if it is a leaf} \\
\text{majority}\{\text{resolve}(\pi') : \pi' \text{ is a child of } \pi\} \\
0 \text{ if no majority (use a default if tied)}
\end{cases}
\]

n = 4, f = 1

Resolved Values are Consistent

**Lemma:** If \( p_i \) and \( p_j \) are nonfaulty, then \( p_i \)'s resolved value for tree node labeled \( \pi_j \) (what \( p_j \) tells \( p_i \) for node \( \pi \)) equals what \( p_j \) stores in its node \( \pi \).

Proof by induction \( \pi \)'s height

By inductive hypothesis, resolved values for \( \pi \) children corresponding to nonfaulty processes are consistent.

Since \( n > 3f \) and \( \pi \) has \( \geq n - f \) children majority of children correspond to nonfaulty processes.
Resolved Values are Valid

- Suppose all inputs are \( v \).
- Nonfaulty process \( p_i \) decides \( \text{resolve}(\lambda) \), which is the majority among \( \text{resolve}(j) \), \( 0 \leq j \leq n-1 \), based on \( p_i \)'s tree.
- Since resolved values are consistent, \( \text{resolve}(j) \) (at \( p_i \)) is the value stored at the root of \( p_j \)'s tree, which is \( p_j \)'s input value if \( p_j \) is nonfaulty.
- Since there is a majority of nonfaulty processes, \( p_i \) decides \( v \).

Common Nodes

A tree node \( \pi \) is **common** if all nonfaulty processes compute the same value of \( \text{resolve}(\pi) \).
Common Frontiers

A tree node $\pi$ has a common frontier if there is a common node on every path from $\pi$ to a leaf.

Proof by induction on height of $\pi$, since resolve uses majority.

Implies agreement:
- On each root-leaf path there is at least one node corresponding to a nonfaulty process
  - The nodes on the path correspond to $f + 1$ different processes
  - There are at most $f$ faulty processes
- This node is common (by consistency of resolved values)
- The root has a common frontier
- The root is common

Lemma: If $\pi$ has a common frontier, then $\pi$ is common
Complexities of the Tree Algorithm

- $n > 3f$ processors
- $f + 1$ rounds
- exponential size messages:
  - each message in round $r$ contains $n(n-1)(n-2)...(n-(r-2))$ values
  - When $r = f + 1$, this is exponential if $f$ is more than a constant relative to $n$

A More Efficient Algorithm?

Better message complexity by increasing the number of rounds and ratio of nonfaulty processes
- $n > 4t$, $2(f + 1)$ rounds

Aside: there are algorithms with
- Polynomial number of message bits
- $f+1$ rounds
- $n > 3t$
Phase King Algorithm
(n > 4t, 2(f+1) rounds)

Code for process $p_i$

```
pref = my input

first round of phase $k$, $1 \leq k \leq f+1$:
    send pref to all
    receive prefs of others
    let maj be value that occurs > $n/2$ times // default 0
    let mult be number of times maj occurs

second round of phase $k$:
    if $i = k$ then send maj to all // I am the phase king
    receive tie-breaker from $p_k$ // default 0
    if mult > $n/2 + f$ then
        pref := maj
    else
        pref := tie-breaker
    if $k = f + 1$ then decide pref
```

Unanimous Phase Lemma

**Lemma:** If all nonfaulty processes prefer $v$ at start of phase $k$, then all prefer $v$ at end of phase $k$

Since $n > 4f$, it follows that $n - f > n/2 + f$

Therefore, if all nonfaulty processes have input $v$

- At start of phase 1, all nonfaulty processes prefer $v$.
- At end of phase 1, all nonfaulty processes prefer $v$.
- At start of phase 2, all nonfaulty processes prefer $v$.
- At end of phase 2, all nonfaulty processes prefer $v$.
- ...  
- At end of phase $f+1$, all nonfaulty processes prefer $v$ and decide $v$
Nonfaulty King Lemma

**Lemma:** If $p_k$ is nonfaulty, then all nonfaulty processes have same preference at end of phase $k$

**Proof:** If two nonfaulty processes $p_i$ and $p_j$ use $p_k$'s tie-breaker, they have same preference.
If $p_j$ uses a majority value $v$ and $p_j$ uses $p_k$'s tie-breaker then $p_k$ majority value is also $v$.
If both $p_i$ and $p_j$ use their majority value, then it must be the same value.

Agreement in Phase King Algorithm

$f + 1$ iterations $\Rightarrow$ at least one with a nonfaulty king.
Nonfaulty King Lemma $\Rightarrow$ at the end of that phase, all nonfaulty processes have same preference.
Unanimous Phase Lemma $\Rightarrow$ from that phase on, all nonfaulty processes have same preference.
$\Rightarrow$ All nonfaulty processes decide on the same value.
Phase King Algorithm
(n > 3t, 3(f+1) rounds)

Code for process $p_i$

$pref = \text{my input}$

**first round of phase $k$, $1 \leq k \leq f+1$:**
- send $pref$ to all
- receive $pref$’s of other processes
- $pref = \text{abort}$
  - if some value $v$ appears > $n-t$ times then $pref = v$

**second round of phase $k$, $1 \leq k \leq f+1$:**
- send $pref$ to all
- receive $pref$’s of others
  - if some value $v$ appears $\text{mult} > t$ times then
    - $pref = \text{smallest such } v$ // abort is largest

**third round of phase $k$, $1 \leq k \leq f+1$:**
- if $i = k$ then send $pref$ // I am phase king
- receive $\text{vking}$ from $k$
  - if ($pref = \text{abort}$ or $\text{mult} < n-t$) and ($\text{vking} \neq \text{abort}$)
    - then $pref = \min(1, \text{vking})$ // abort is largest

Unanimous Phase Lemma

**Lemma:** If all nonfaulty processes prefer $v$ at start of phase $k$, then all prefer $v$ at end of phase $k$

For each phase $k$:
- At the end of the first round, the value of $pref$ for all nonfaulty processes, is $v$ or abort, for some $v \in \{0,1\}$
- At the end of the second round, the value of $pref$ for all nonfaulty processes, is $v$ or abort, for the same $v$
Nonfaulty King Lemma

**Lemma:** If $p_k$ is nonfaulty, then all nonfaulty processes have same preference at end of phase $k$

- All nonfaulty processes accept the phase king’s message
- Some nonfaulty process ignores the king since $\text{mult} \geq n-t$. Then $\text{mult} > f$ for every nonfaulty process, and its pref is the same.

After this phase, Unanimous Phase Lemma ensures agreement is maintained until the algorithm terminates.

Randomized Consensus

- Weakening the termination condition and measuring the expected time to decide
- Agreement and validity remain the same
- Allow to overcome the asynchronous impossibility and the synchronous lower bound (we’ll see only the first)
Two Sources of Nondeterminism

• In a randomized algorithm, processes flip coins to determine their next steps
  — Several possible executions
• But even when the algorithm is deterministic, it has several possible executions (from fixed inputs)
  — Due to asynchrony, failures

• Separate out variation under the control of an adversary
  — Determines the next event to occur after an execution prefix
  — Obeys admissibility conditions of the relevant model
  — May have other limitations (what information it can observe, how much computational power it has)

Evaluating a Distributed Randomized Algorithm

• An execution of a specific algorithm, \( \text{exec}(C_0, R, A) \), is uniquely determined by
  — an initial configuration \( C_0 \)
  — a set of random numbers \( R \)
  — an adversary \( A \)
• Given a predicate \( \text{Pred} \) on executions and a fixed adversary \( A \) and initial config \( C_0 \),
  \[ \Pr[\text{Pred}] = \text{Prob}\{R : \text{exec}(C_0, R, A) \text{ satisfies } \text{Pred}\} \]
• Let \( T \) be a random variable (time).
  \[ \exp(T, A, C_0) = \sum_t t \Pr[T = t] \]
Expected Time Complexity of a Randomized Distributed Algorithm

The **expected time complexity** is the max over all admissible adversaries $A$ and initial configurations $C_0$, of the expected time for that particular $A$ and $C_0$. I.e.,

$$\max_{\text{adversary } A, \text{ initial configuration } C_0} \exp(T(\text{Alg}, A, C_0))$$

**Worst-case average**: for the worst adversary (asynchrony and failures) and initial configuration, average over the random choices of the algorithm.

Extend naturally to other measures (like RMRs)

---

Structure of the Algorithm

The algorithm has two components.

- **Phase-based voting scheme** using individual processors’ preferences to reach agreement (when possible)
  
  We use extended adopt-commit

- A **shared coin procedure** used to break ties among these preferences
Shared Coin

A shared coin with agreement probability \( \rho \) (with no input) returns a binary output, s.t.

- For every \( v \in \{0,1\} \), all nonfaulty processors executing the procedure output \( v \) with probability at least \( \rho \)

A simple and very resilient shared coin with \( \rho = 1/2^n \) bias is when each process output a (uniform) random bit.

There are more sophisticated constructions.

Recall: Adopt-Commit

The decision is \((\text{grade}, y_i)\), where grade is either adopt or commit, such that

- **Graded agreement**: if a process decides \((\text{commit}, y_i)\) then all processes decide \((\text{adopt}, y_i)\) or \((\text{commit}, y_i)\)
- **Validity**: \( y_i \) was proposed by some process
- **Convergence**: If only \( y_i \) is proposed before \( p \) outputs \((\text{grade}, y_i)\) then grade = commit
- **Termination**: A process returns within a finite number of steps

\[(\text{commit}, 0) \quad (\text{adopt}, 0) \quad (\text{adopt}, 1) \quad (\text{commit}, 1)\]
Extended Adopt-Commit

The decision is either \( \bot \) (\text{abort}) or (\text{grade}, y_i), where grade is either \text{adopt} or \text{commit}, such that

- **Graded agreement**: if a process decides (\text{commit},v) then all processes decide (\text{adopt},v) or (\text{commit},v) and if a process decides (\text{adopt},v) then no process adopts a different value

- **Validity**: if all nonfaulty processes propose v then all nonfaulty processes return (\text{commit},v)

- **Termination**: A process returns within a finite number of steps

Implementing Extended Adopt-Commit w/ Byzantine Failures

- Assumes \( n > 3f \)
- 2 rounds

send v to all
receive values from others
let maj be value that occurs > n/2 times (0 if none)
let mult be number of times maj occurs
if mult \( \geq n-f \) then send maj to all;

receive values from others
let maj’ be value that occurs most times
let mult’ be number of times maj’ occurs
if mult’ \( \geq n-f \) return (commit,maj’)
else if mult’ \( \geq f+1 \) return (adopt,maj’)
else return abort
Randomized Consensus w/ Extended Adopt-Commit

Assume we have a shared coin algorithm with agreement probability $\rho$ and time complexity $T_{\text{coin}}$.

```
pref = my input
Phase k
(grade, v) = Extended-adopt-commit(pref) 
flip = Shared-coin()
if grade == abort
    pref = flip
if grade == adopt
    pref = v
else // grade == commit
    decide v // but continue to echo
```

Time complexity of a phase is $(T_{\text{EAC}} + T_{\text{coin}})$

Validity

**Unanimous Phase Lemma:** If all nonfaulty processes prefer $v$ at start of phase $k$, then all do at end of phase $k$.

If all processes have input $v \Rightarrow$ all prefer $v$ in phase 1. By the lemma (and graded agreement), all nonfaulty processes decide $v$ in phase 1.
Agreement

**Lemma:** If $p_i$ decides $v$ in phase $r$, then all nonfaulty processes decide $v$ in phase $r+1$.

**Proof:** Let $r$ be the earliest phase in which a process (say, $p_i$) decides (say, on $v$) $p_i$ got $(\text{commit},v)$ in round $r$.
All other process got $(\text{adopt},v)$ in round $r$, so they prefer $v$ in round $r+1$ and by previous lemma, decide $v$.

Termination

**Lemma:** The probability that all nonfaulty processes decide in a phase is at least $\rho$.

**Proof:** If all nonfaulty processes set their preference in phase $r$ using Shared-coin
  – with probability $2\rho$, they all get the same value ($\rho$ for 0 and $\rho$ for 1); lemma follows from unanimous phase lemma
If some processes do not set their preference using Shared-coin
  – All of them have the same value $v$ as phase $r$ preference
  – With probability $\geq \rho$, all processes get $v$ from Shared-coin.
Expected Number of Phases

Probability of all deciding in any given phase ≥ ρ
⇒ Probability of terminating after i phases is (1−ρ)i−1ρ
⇒ Number of phases until termination is a geometric random variable whose expected value is 1/ρ

The time complexity of the algorithm is ρ−1(T_{EAC}+T_{coin}),
T_{EAC} is the time complexity of Extended Adopt-Commit
T_{coin} is the time complexity of Shared-coin

Better Shared Coin

Back to shared memory and crash failures...
• constant agreement probability ρ
• polynomial total number of steps T_{coin}

```
Shared SumCoins[i], NumFlips[i], initially 0
while ()
    c = random(-1,+1)
    SumCoins[i] += c  // written only by i, atomic
    NumFlips[i]++     // written only by i, atomic
read NumFlips[0,...,n-1]
if Σ NumFlips[0,...,n-1] > n²
    read SumCoins[0,...,n-1]
return( sign( Σ SumCoins[0,...,n-1] ) )
```

simpler than (0,1)
Step Complexity

- Number of coins flipped (= iterations of the while loop) < $n^2 + n$
- $O(n)$ steps per iteration $\Rightarrow O(n^3)$ total work

```
Shared SumCoins[i], NumFlips[i], initially 0
while ()
    c = random(-1,+1)
    SumCoins[i] += c  // written only by i, atomic
    NumFlips[i]++  // written only by i, atomic
    read NumFlips[0,...,n-1]
    if $\Sigma$ NumFlips[0,...,n-1] > $n^2$
        read SumCoins[0,...,n-1]
        return( sign( $\Sigma$ SumCoins[0,...,n-1] ))
```

Agreement Parameter

- Among $t^2+t$ independent unbiased coins, the minority is less than $t^2/2$ with probability $> \frac{1}{2}$
- Probability all processes get same value $> \frac{1}{4}$

```
Shared SumCoins[i], NumFlips[i], initially 0
while ()
    c = random(-1,+1)
    SumCoins[i] += c  // written only by i, atomic
    NumFlips[i]++  // written only by i, atomic
    read NumFlips[0,...,n-1]
    if $\Sigma$ NumFlips[0,...,n-1] > $n^2$
        read SumCoins[0,...,n-1]
        return( sign( $\Sigma$ SumCoins[0,...,n-1] ))
```