Implementing Causal Ordering

- Uses vector clocks to keep causal ordering (piggybacked to messages)
- Vector clock $V^g_i[j]$ counts all multicast messages of process $i$ in group $g$
- Hold-back queue reflects vector clocks

Algorithm for group member $p_i$ ($i = 1, 2, \ldots, N$)

On initialization
$$V^g_{i}[j] := 0 \ (j = 1, 2, \ldots, N);$$

To CO-multicast message $m$ to group $g$
$$V^g_{i}[i] := V^g_{i}[i] + 1;$$
$$B\text{-multicast}(g, <V^g_{i}, m>);$$

On $B\text{-deliver(<}V^g_{j}, m>)$ from $p_j$, with $g = \text{group}(m)$
place $<V^g_{j}, m>$ in hold-back queue;
wait until $V^g_{j}[j] = V^g_{i}[j] + 1$ and $V^g_{j}[k] \leq V^g_{i}[k] \ (k \neq j)$;
$CO\text{-deliver } m;$ // after removing it from the hold-back queue
$$V^g_{i}[j] := V^g_{i}[j] + 1;$$
4.5: Consensus

- **n** processes $p_1, \ldots, p_n$
- At most $f$ processes have arbitrary (Byzantine) failures
- Every process starts in the *undecided* state and *proposes* a value $v_i$
- Eventually all correct processes $p_i$
  - Choose the *decided* state
  - And choose the same value $d_i \in \{v_1, \ldots, v_n\}$
  - And stay in this state
Consensus Problem

- **Termination**: Eventually each correct process \( p_i \) is *decided* by setting variable \( d_i \).
- **Agreement**: The decision value \( d_i \) of all correct processes is the same.
- **Integrity**: If all correct process proposed the same value \( v \), then \( d_i = v \) for all correct \( p_i \).

- Possible decision functions: *majority*, *minimum*, *maximum*, ... 
- Byzantine failures can cause irritating and adversarial messages.
- System crashes may not be detected.

![Consensus Algorithm Diagram](image)
Byzantine Generals Problem

- $n$ generals have to agree on attack or retreat
- one of them is the commander and issues the order
- at most $f$ generals are traitors (possibly also the commander) and have adversarial behavior
- all correct generals have eventually to agree on the commander decision if he acts correctly

Consensus Problem

- **Termination**: Eventually each correct process $p_i$ is decided by setting variable $d_i$
- **Agreement**: The decision value $d_i$ of all correct processes is the same
- **Integrity**: If the commander is correct then all correct processes choose the commander's proposal
Interactive Consistency

- $n$ processes need to agree on a vector of values
- Each process proposes a value $v_i$
- A correct processes eventually decide on a vector $d_i = \{d_{i,1}, \ldots, d_{i,n}\}$ where

$$d_{i,j} = v_j \text{ if } p_j \text{ is correct}$$

Interactive Consistency

- **Termination**: Eventually each correct process $p_i$ is decided by setting variable $d_i$
- **Agreement**: The decision value $d_i$ of all correct processes is the same
- **Integrity**: If the $p_j$ is correct then all correct processes $p_i$ set $d_{i,j} = v_j$
The Relationship between Consensus Problems

Assume solutions to Consensus (C), Byzantine generals (BG), interactive consistency (IC)

\[
C_i(v_1, \ldots, v_n) = \text{consensus decision value of } p_i \text{ for proposals } v_i \\
BG_i(j, v) = \text{BG decision value of } p_i \text{ for commander } p_j \text{ proposal } v_j \\
IC_i(v_1, \ldots, v_n)[j] = j\text{-th position of interactive consistency decision vector of } p_i \text{ for proposals } v_i
\]

Solving IC from BG

- In parallel \( n \) Byzantine generals problems are solved
- each process \( p_j \) acts as commander once

\[
IC_i(v_1, \ldots, v_n)[j] = BG_i(j, v)
\]
The Relationship between Consensus Problems

Solving $C$ from $IC$

- $\text{majority}$ returns the most often parameter or $\bot$ if no such value exists
- for all $i = 1, \ldots, n$

$$C_i(v_1, \ldots, v_n) = \text{majority}(IC_i(v_1, \ldots, v_n)[1], \ldots, IC_i(v_1, \ldots, v_n)[n])$$

Solving $BG$ from $C$

- The commander $p_j$ sends its proposed value to itself and each other process
- All processes run consensus with the values $v_1, \ldots, v_n$ received from the commander
- for all $i = 1, \ldots, n$

$$BG_i(j, v) = C_i(v_1, \ldots, v_n)$$
Consensus in a Synchronous System

- Assume that there are no arbitrary (Byzantine) errors
- Given a synchronous distributed systems (fail-stop model)
- Use basic multicast for $f + 1$ rounds
- Multicast all known values of all participants
- $Values^r_i$ denotes the set of proposed variables at the beginning of round $r$
- Reduce communication overhead by multicasting only freshly arrived variables $Values^r_i - Values^{r-1}_i$
- Choose the minimum of all known values as final value
Consensus in a Synchronous System

Algorithm for process $p_i \in g$; algorithm proceeds in $f + 1$ rounds

On initialization

$$\text{Values}_i^1 := \{v_i\}; \text{ Values}_i^0 = \{\};$$

In round $r$ ($1 \leq r \leq f + 1$)

$B$-multicast($g$, $\text{Values}_i^r - \text{Values}_i^{r-1}$); // Send only values that have not been sent

$\text{Values}_i^{r+1} := \text{Values}_i^r$;

while (in round $r$)

$$\{$$

On $B$-deliver($V_j$) from some $p_i$

$\text{Values}_i^{r+1} := \text{Values}_i^{r+1} \cup V_j$;

$$\}$$

After $(f + 1)$ rounds

Assign $d_i = \text{minimum}(\text{Values}_i^{f+1})$;
Consensus in a Synchronous System

- There are no arbitrary errors only processes that crash and are correctly detected
- Given a synchronous distributed systems (fail-stop model)
- Correctness
  - Assume that two processes $p_i$ and $p_j$ have different values at round $r$
  - Then, in round $r - 1$ at least one process $p_k$ has sent different values to $p_i$ and $p_j$
  - Then, $p_k$ has crashed in this round
  - Since the number of crashes is limited to $f$ there are not enough crashes to cover each of the $f + 1$ rounds
Byzantine Generals Problem in a Synchronous System

- Assume that there are Byzantine errors
- Given a synchronous distributed system
  - crashes are detected
  - other wrong behavior cannot be detected, e.g. strange messages
- messages are not (digitally) signed
- at most $f$ faulty processes

Impossibility of a solution of the Byzantine generals problem
[Lamport, Shostak, Pease 1982]

- The byzantine generals problem cannot be solved for $n = 3$ and $f = 1$.
- The byzantine generals problem cannot be solved for $n \leq 3f$. 
Impossibility of a solution of the Byzantine generals problem for $n = 3$

- The Byzantine generals problem with arbitrary failures cannot be solved for $n = 3$ and $f = 1$ in a synchronous system.
  - A faulty commander sending different values to his generals
  - Cannot be distinguished from a faulty general forwarding wrong values
Solution of the Byzantine Generals Problem

- Assume that there are **Byzantine** errors
- Given a synchronous distributed system
- messages are not (digitally) signed
- at most $f$ faulty processes

Solution of the Byzantine generals problem [Pease, Shostak, Lamport 1980]

- The byzantine generals problem can be solved for $n = 4$ and $f = 1$.
- The byzantine generals problem can be solved for $n \geq 3f + 1$. 
Solution for Four Generals and One Faulty Process

- The byzantine generals problem can be solved for $n \geq 4$ and $f = 1$.

Algorithm of Pease et al.

1. The commander sends a value to all other generals (lieutenants)
2. All lieutenants send the received value to all other lieutenants
3. The commander chooses its value; the lieutenants compute the majority of all received values

- Since $n \geq 4$ the majority function always can be computed if at most one process is faulty
- If the commander crashes very early then all lieutenants agree on $\perp$
More About the Byzantine Generals Problems

For $f > 1$ the algorithm can be used recursively
- Complexity: $f + 1$ rounds and $O(n^{f+1})$ messages
- The time complexity of $f + 1$ rounds is optimal

With the help of signed messages
- any number of faulty generals $f < n$ can be dealt with
- with signed messages the Byzantine Generals problem can be solved in $f + 1$ rounds with $O(n^2)$ messages [Dolev & Strong 1983]

For asynchronous systems with crash failures
- No algorithm can reach consensus even if only one processor is faulty [Fischer, Lynch, Paterson 1985]
- Each algorithm that tries to reach consensus can be confronted with a faulty process which influences the result if it continues (instead of crashing)
End of Section 4