Lecture 18: Distributed Agreement

CSC 469H1F / CSC 2208H1F

Fall 2010

Angela Demke Brown

A rose by any other name…

- Distributed Consensus has many names (depending on the assumptions and application)
  - Reliable multicast
  - Interactive consistency
  - Atomic broadcast
  - Byzantine Generals Problem

"This has resulted in a voluminous literature which, unfortunately, is not distinguished for its coherence. The differences in notation and the haphazard nature of the assumptions obfuscates the close relationship among these problems"

– Hadzilacos & Toueg, Distributed Systems.

Review: Distributed Algorithms

- System model from last lecture.
- Distributed system is composed of *n* processes
- A process executes a sequence of events
  - Local computation
  - Sending a message *m*
  - Receiving a message *m*
- A distributed algorithm is an algorithm that runs on more than one process.
  - Safety – some bad thing never happens
  - Liveness – some good thing eventually happens

Review: Timing / Failure Models

- Timing assumptions:
  - Synchronous – shared clock, known bounds on message delivery
  - Asynchronous – no global clock, no time bounds on message delivery
  - Partial Synchrony – clocks synchronized within some bound, timeout to manage bounds on message delivery
- Failure assumptions:
  - Fail-stop – process is correct until it stops entirely
  - Byzantine – failed process behaves arbitrarily

Setup of Distributed Consensus

- N processes have to agree on a single value.
  - Example applications of consensus:
    - Performing a commit in a replicated/distributed database.
    - Collecting multiple sensor readings and deciding on an action
- Each process begins with a value
- Each process can irrevocably *decide* on a value
- Up to $f < n$ processes may be faulty
  - How do you reach consensus if no failures?

Properties of Distributed Consensus

- *Agreement*
  - If *any correct* process believes that *V* is the consensus value, then *all correct* processes believe *V* is the consensus value.
- *Validity*
  - If *V* is the consensus value, then some process proposed *V*.
- *Termination*
  - Each process decides some value *V*.
- *Agreement* and *Validity* are **Safety** Properties
- *Termination* is a **Liveness** property.

Synchronous Fail-stop Consensus

- FloodSet algorithm run at each process *i*
  - Remember, we want to tolerate up to f failures

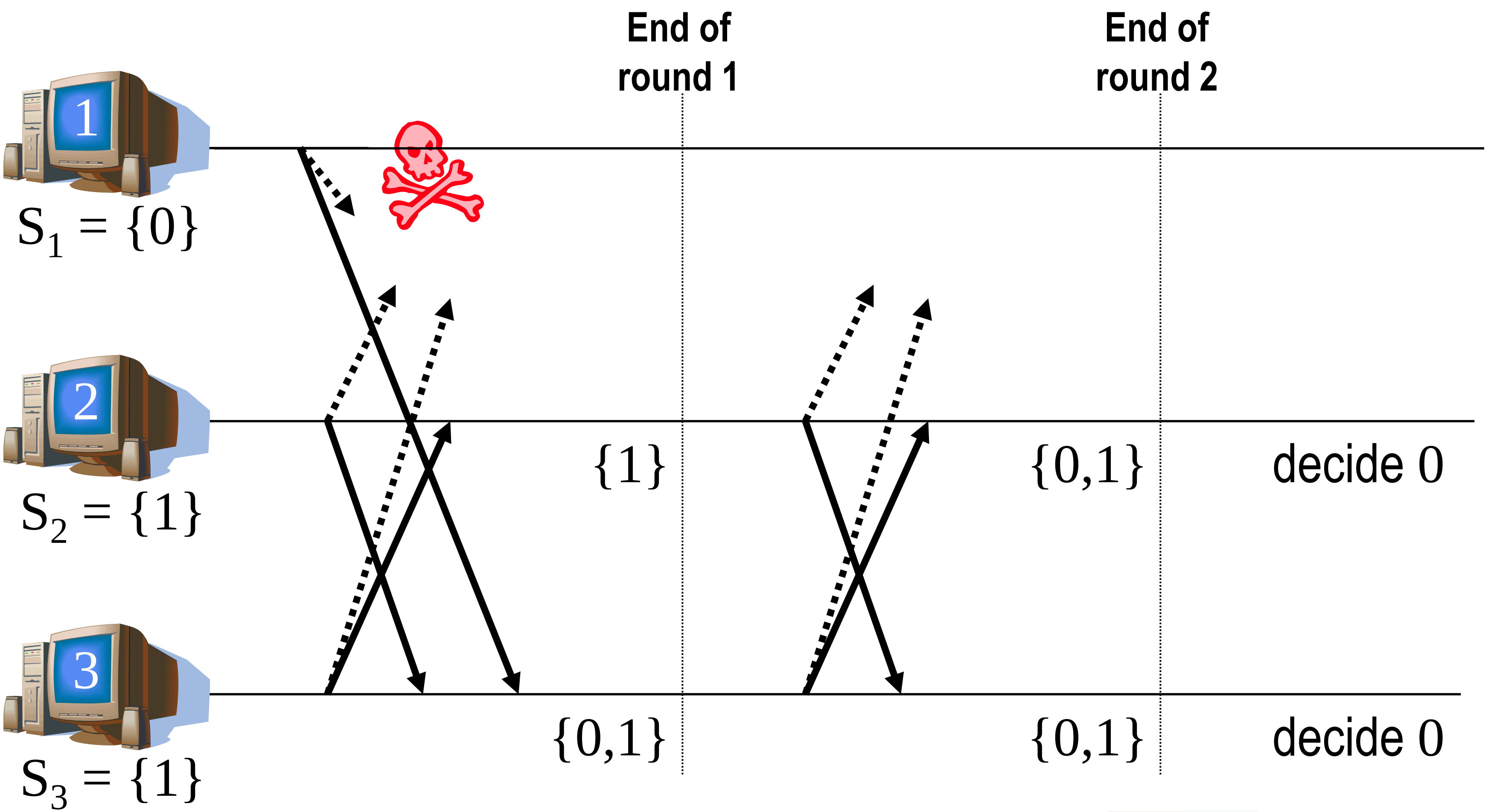
```
Si ← {initial value}
for k = 1 to f+1
    send Si to all processes
    receive Sj from all j != i
        Si ← Si ∪ Sj  (for all j)
end for
Decide(Si)
```

- S is a set of values
- Decide(x) can be various functions
  - E.g. min(x), max(x), majority(x), or some default
- Assumes nodes are connected and links do not fail

Analysis of FloodSet

- Requires *f+1* rounds because process can fail at any time, in particular, during send
  - Must guarantee 1 round in which no failure occurs
- *Agreement*: Since at most *f* failures, then after *f+1* rounds all correct processes will evaluate `Decide`(S_i) the same.
- *Validity*: `Decide` results in a proposed value (or default value)
- *Termination*: After *f+1* rounds the algorithm completes

Example with $f = 1$, Decide() = min()



Synchronous/Byzantine Consensus

- Faulty processes can behave arbitrarily
  - May actively try to trick other processes
- Algorithm described by Lamport, Shostak, & Pease in terms of Byzantine generals agreeing whether to attack or retreat.   Simple requirements:
  - All loyal generals decide on the same plan of action
    - Implies that all loyal generals obtain the same information
  - A small number of traitors cannot cause the loyal generals to adopt a bad plan
  - Decide() in this case is a majority vote, default action is “Retreat”

Byzantine Generals

- Use v(i) to denote value sent by i^{th} general
- traitor could send different values to different generals, so can't use v(i) obtained from i directly. New conditions:
  - Any two loyal generals use the same value v(i), regardless of whether i is loyal or not
  - If the i^{th} general is loyal, then the value that she sends must be used by every loyal general as the value of v(i).
- Re-phrase original problem as *reliable broadcast:*
  - General must send an order ("Use v as my value") to lieutenants
  - Each process takes a turn as General, sending its value to the others as lieutenants
  - After all values are reliably exchanged, Decide()

Synchronous Byzantine Model

Theorem: There is no algorithm to solve consensus if only oral messages are used, unless *more than two thirds* of the generals are loyal.

- In other words, impossible if $n \le 3f$ for *n* processes, *f* of which are faulty
- *Oral messages* are under control of the sender
  - sender can alter a message that it received before forwarding it
- Let's look at examples for special case of n=3, f=1

Case 1

- Traitor lieutenant tries to foil consensus by refusing to participate

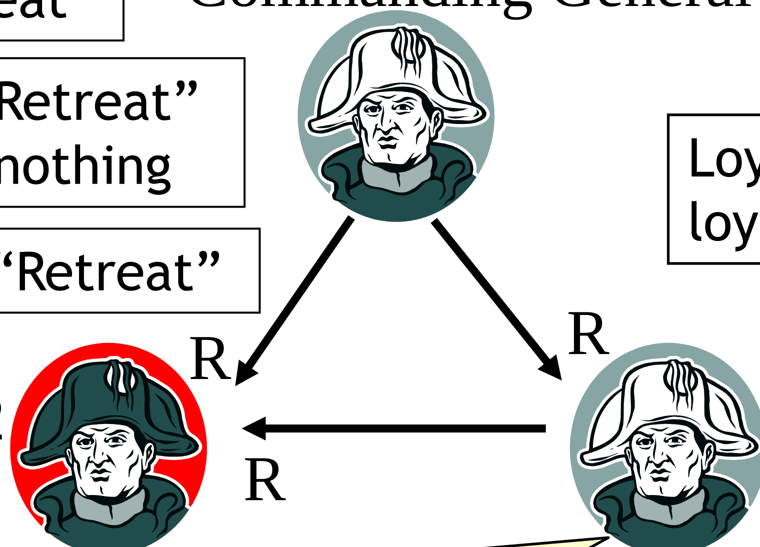
“white hats” == loyal or “good guys”
“black hats” == traitor or “bad guys”

Round 1: Commanding General sends “Retreat”

Round 2: L3 sends “Retreat” to L2, but L2 sends nothing

Decide: L3 decides “Retreat”

Loyal lieutenant obeys loyal commander. (good)



Remember, synchronous timing assumption, so L3 knows when to stop waiting for message from L2

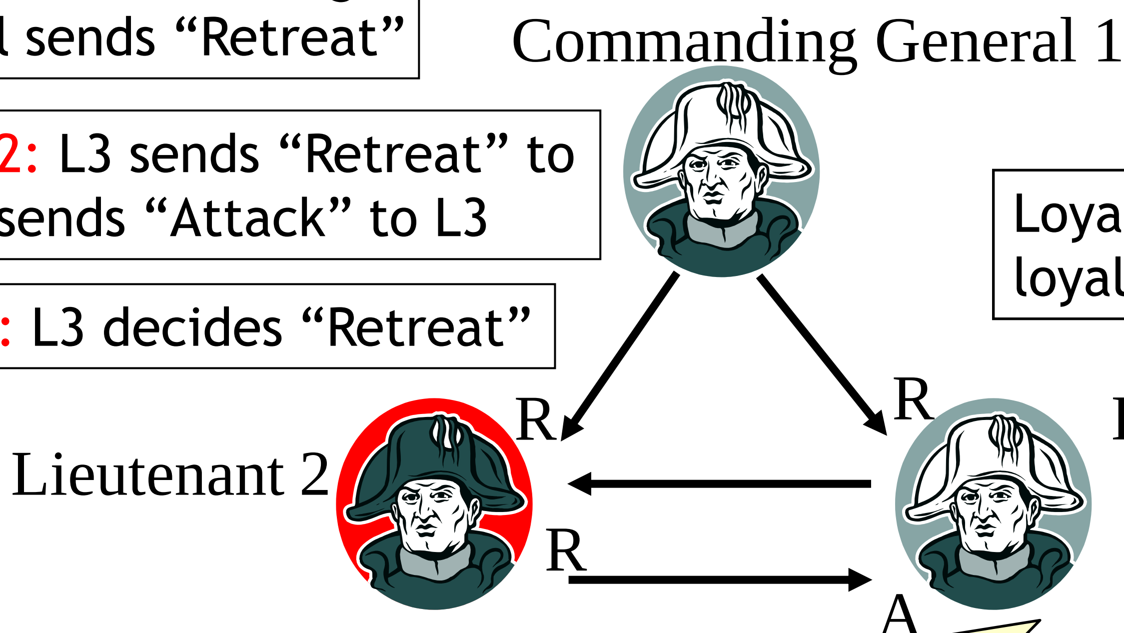
Case 2a

- Traitor lieutenant tries to foil consensus by lying about order sent by general

Round 1: Commanding General sends “Retreat”

Round 2: L3 sends “Retreat” to L2; L2 sends “Attack” to L3

Decide: L3 decides “Retreat”



Lieutenant 3

Loyal lieutenant obeys loyal commander. (good)

decides to retreat (default action)

L3 has no clear majority. Default: Obey General? Always retreat? Same in this case.

Case 2b

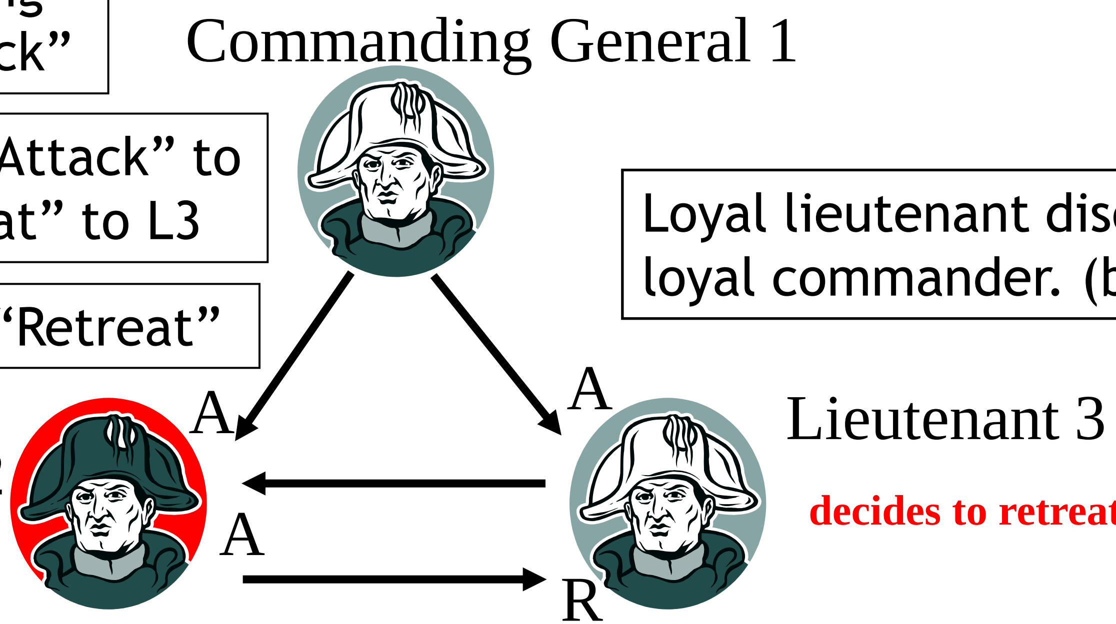
- Traitor lieutenant tries to foil consensus by lying about order sent by general

Round 1: Commanding General sends "Attack"

Round 2: L3 sends "Attack" to L2; L2 sends "Retreat" to L3

Decide: L3 decides "Retreat"

Loyal lieutenant disobeys loyal commander. (bad)



L3 again has no majority. Default action of "retreat" leads to bad outcome. Should default be "obey order"?

Case 3

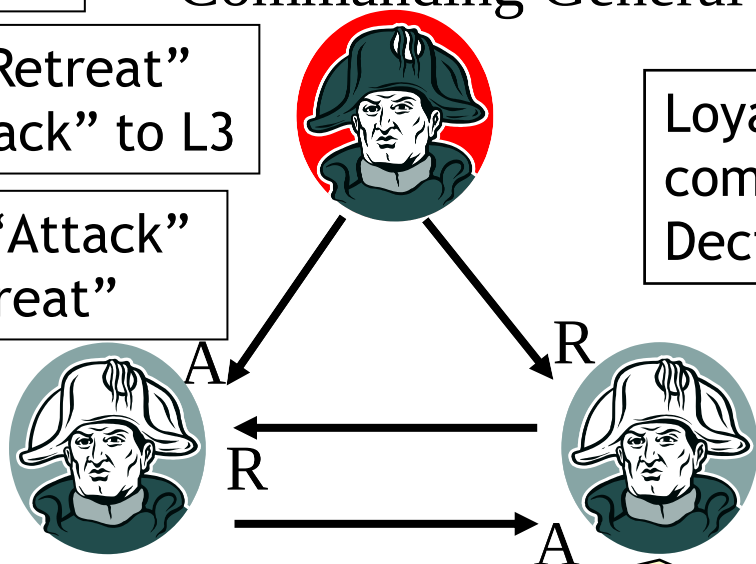
- Traitor General tries to foil consensus by sending different orders to loyal lieutenants

Round 1: General sends "Attack" to L2 and "Retreat" to L3

Round 2: L3 sends "Retreat" to L2; L2 sends "Attack" to L3

Decide: L2 decides "Attack" and L3 decides "Retreat"

Loyal lieutenants obey commander. (good?) Decide differently (bad)



Neither L2 or L3 has majority. Default action of "obey order" violates consensus. Default "retreat" was bad in Case 2. Indistinguishable from perspective of L3.

Byzantine Consensus: $n > 3f$

- Oral Messages algorithm, OM(f)
- Consists of $f+1$ "phases"
- Algorithm OM(0) is the "base case" (no faults)
  1) Commander sends value to every lieutenant
  2) Each lieutenant uses value received from commander, or default "retreat" if no value was received
- Recursive algorithm handles up to f faults

OM(f): Recursive Algorithm

1) Commander sends value to every lieutenant
2) For each lieutenant *i*, let v_i be the value *i* received from commander , or “retreat” if no value was received. Lieutenant *i* acts as commander in Alg. OM(f-1) to send v_i to each of the *n-2* other lieutenants
3) For each *i*, and each *j* not equal to *i*, let v_j be the value Lieutenant *i* received from Lieutenant *j* in step (2) (using Alg. OM(f-1)), or else “retreat” if no such value was received. Lieutenant *i* uses the value *majority*$(v_1, \dots, v_{n-1})$.

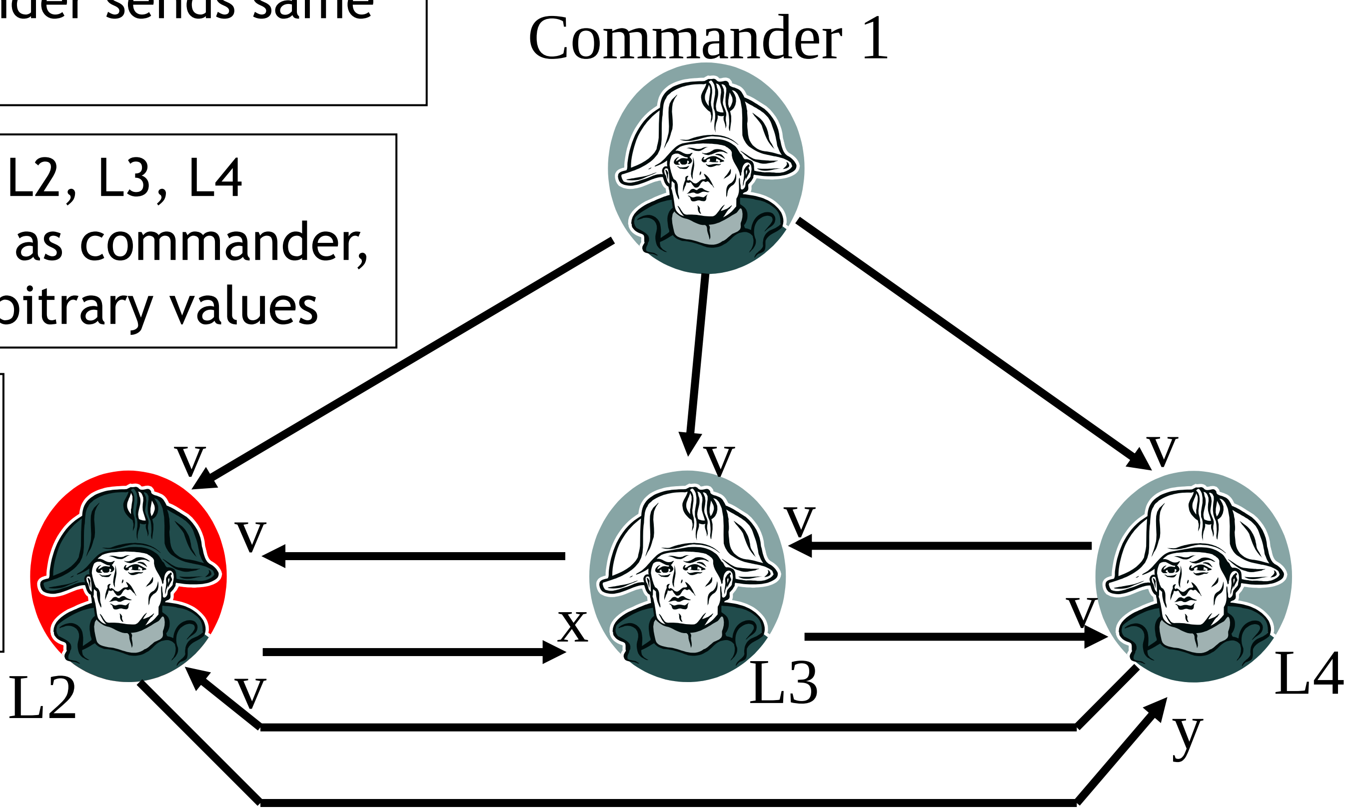
Example: f = 1, n = 4

- Loyal General, 1 traitor lieutenant

Step 1: Commander sends same value, v, to all

Step 2: Each of L2, L3, L4 executes OM(0) as commander, but L2 sends arbitrary values

Step 3: Decide
L3 has {v,v,x},
L4 has {v,v,y},
Both choose v.



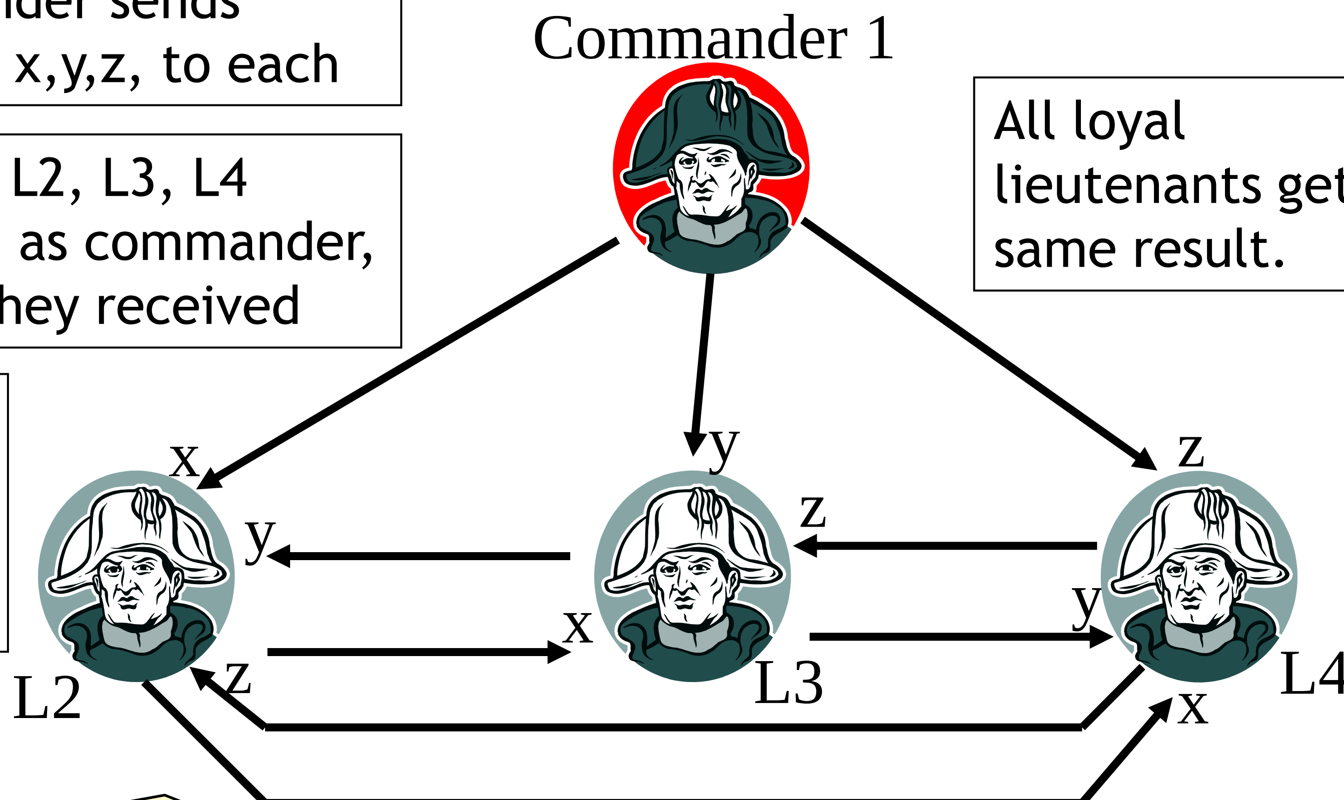
Example: f = 1, n = 4

- Traitor General, all lieutenants loyal

Step 1: Commander sends different value, x,y,z, to each

Step 2: Each of L2, L3, L4 executes OM(0) as commander, sending value they received

Step 3: Decide
L2 has {x,y,z}
L3 has {x,y,z},
L4 has {x,y,x},



All loyal lieutenants get same result.

If Commander sends same value (say, x) to 2 of 3 lieutenants, all still finish with same set and decide same value (x). Consensus still satisfied.

Example: OM(2), f=2, n=7

- General sends value v to all six lieutenants
- Now run OM(1) six times
  - L_i takes turn as general to send value received from original general to others
    - All receivers run OM(0) to exchange values
  - At end of each OM(1), all lieutenants agree on the value to use for L_i
- All lieutenants are now using the same set of values to reach overall decision

Problem

- Lots of messages required to handle even 1 faulty process
- Need minimum 4 processes to handle 1 fault, 7 to handle 2 faults, etc.
  - But as system gets larger, probability of a fault also increases
- If we use *signed messages*, instead of oral messages, can handle f faults with 2f+1 processes
  - Simple majority requirement
  - Still lots of messages sent though, plus cost of signing

Asynch. Distributed Consensus

- Fail-Stop/Byzantine $\rightarrow$ IMPOSSIBLE!
- Fischer, Lynch and Patterson (FLP) impossibility result
  - Asynchronous assumption makes it impossible to differentiate between failed and slow processes.
  - Therefore *termination* (**liveness**) cannot be guaranteed.
  - If an algorithm terminates it may violate *agreement* (**safety**).
    - A slow process may decide differently than other processes thus violating the agreement property

Castro: Practical Byz. Fault Tolerance

- Uses various optimizations to combine messages, reduce total communication
- Relies on partially synchronous assumption to guarantee **liveness**.
- Therefore attacks on system can only slow it down – **safety** is guaranteed.
- Assumes that an attack on **liveness** can be dealt with in a reasonable amount of time.
- Suitable for wide area deployment (e.g., internet)
- Being used in Microsoft Research's *Farsite* distributed file system

Partially Synchronous Consensus Algs

- Relies on a Fault-Detector
- Synchronous/Fail-stop distributed consensus algorithms (e.g. FloodSet) can be transformed to run in the partially synchronous environment
- Byzantine is still a problem though…
  - DoS attacks on correct processes result in the identification of correct processes as failed, reducing the number of processes that must be compromised to breach the *safety* property (i.e. attackers can manipulate *f* which is **not** cool)