Byzantine Algorithms

Diego Puppin

Why Byzantine Algorithms?

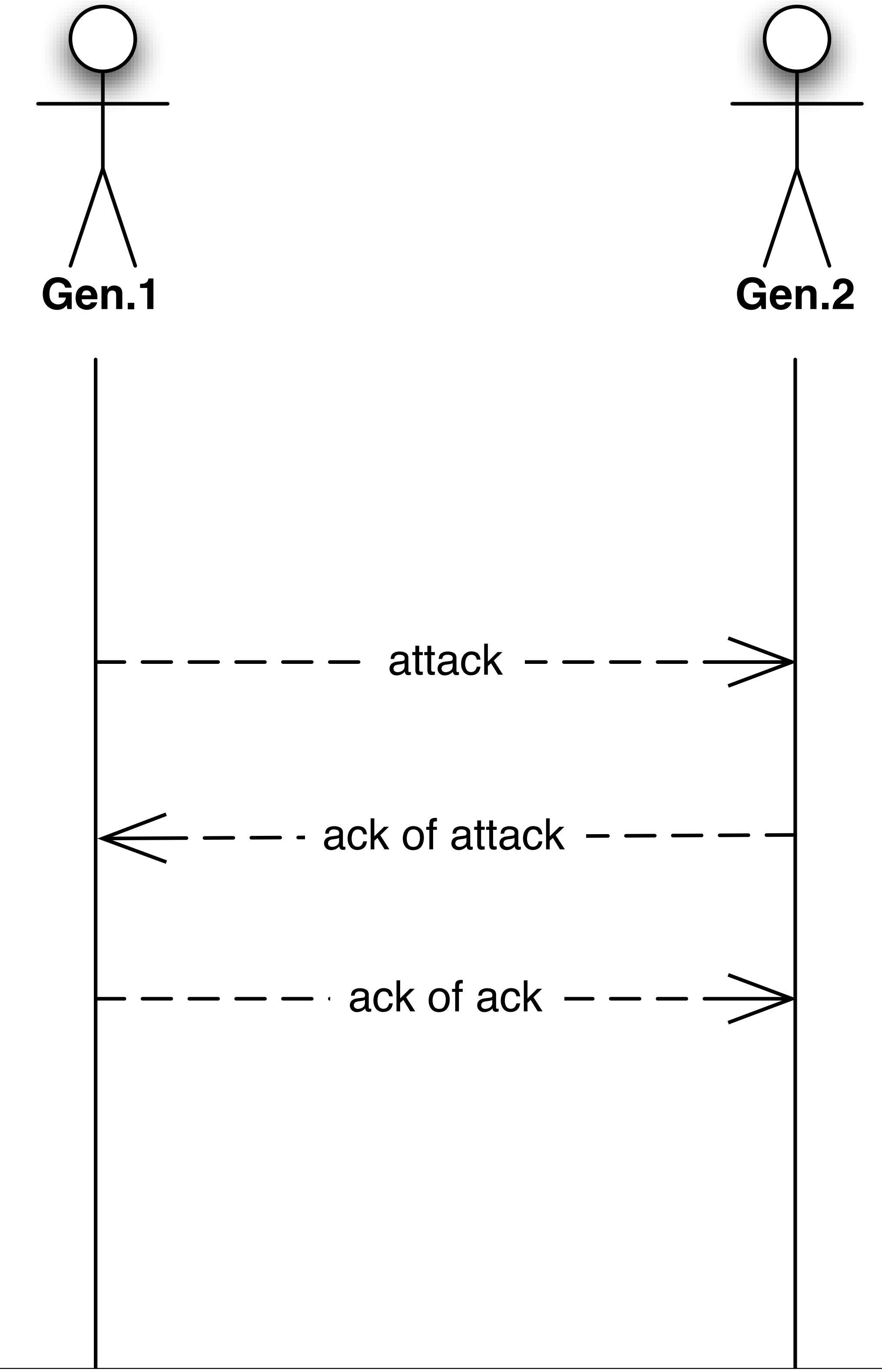
- In a distributed systems, messages can be lost, sent incorrectly
- Some agents may behave maliciously
- Agents may coordinate to attack
- The system must agree on a choice anyway!

Examples

- Replicated archives
- File systems
- Replicated web services

Two Generals Problem

- Two generals, on two sides of a hill, have to agree on a common plan
- They have messengers, running forth and back
- Messengers can be intercepted ...
- *It's impossible to be sure of acknowledgment from the other side!*


Gen.1
Gen.2
attack
ack of attack
ack of ack

WHY?

- Suppose N messages (forth and back) are enough
- The N-1 th messages is acknowledged by the N-th message
- The last sender does not know if acknowledgment was received
- He wouldn't attack

Byzantine Algorithms

- Any agent in the system can send wrong or not send at all
- We want that loyal (non-faulty, non-malicious) agents agree on a value
- On a GOOD value
  - A traitor must not bring the system to a wrong choice

Conditions

- A: All loyal generals agree
- B: A small number of traitors cannot cause the loyal generals to adopt a bad plan

A Simple Plan

- The only we know of [Lamport, 1982]
- If the generals communicate their opinion to each other, they go for the majority, so...
- 1. Every loyal general must obtain the same information v(1)...v(n)
- 2. Every loyal general i send same v(i) to all other generals
- 1′. Two loyal generals use the same v(i)

Restricted Problem

- IC1. All loyal lieutenants obey the same order
- IC2. If the general is loyal, every loyal lieutenant obeys the order

BYZANTINE ALGORITHMS

ALGORITHMS WHICH ALLOW FOR A (FINITE) NUMBER OF ARBITRARY FAULTS, INCLUDING FALSE MESSAGES.

- BASIC IDEA: Use redundant messages and majority voting.
- NOTE: If arbitrary faults can occur, messages which in principle should be identical may in fact differ!
- BASIC PROBLEM: To exchange sufficient information between sufficiently many participants, so that all CORRECTLY OPERATING systems build up the SAME picture of which message(s) have been sent.

THE BYZANTINE GENERALS

- A typical example of a Byzantine problem: to achieve so-called INTERACTIVE CONSISTENCY.
- Usually formulated as a "military" problem with n "generals" (One Commander and (n-1) Lieutenants):

> A Commander must send a value to all his (n-1) Lieutenants, such that:
>
> IC1: All loyal Lieutenants agree on the SAME value.
>
> IC2: If the Commander is loyal, then all loyal Lieutenants agree on the COMMANDER'S value.

- NOTE. In a computer system:

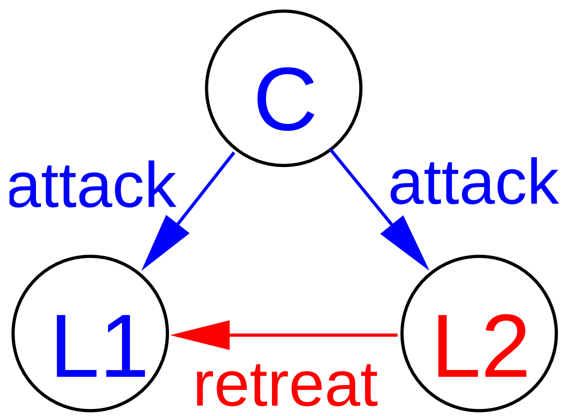
  LOYAL generals are computers which work correctly, and
  ILLOYAL generals are those which may fail in an arbitrary way.
  (This includes sending arbitrary incorrect messages.)

ORAL MESSAGES

- DEF.: messages whose content is 100% determined by the sender.
- This means that sender can give them *ARBITRARY CONTENT* without the receiver being able to see this.
- How much redundancy is required to solve the BG problem?
  First idea: Majority voting among 3 parties, with 1 illoyal.

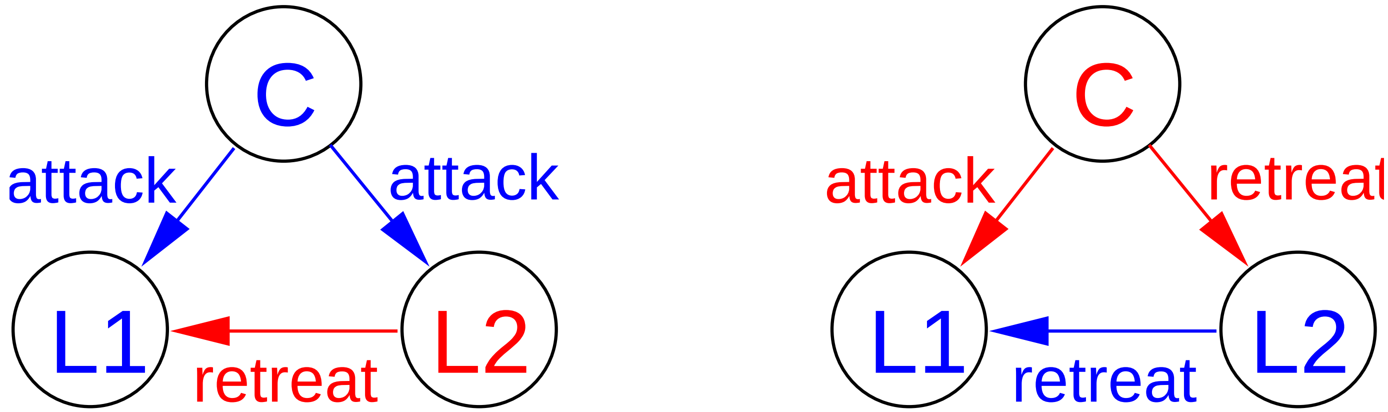
ORAL MESSAGES

- DEF.: messages whose content is 100% determined by the sender.
- This means that sender can give them *ARBITRARY CONTENT* without the receiver being able to see this.
- How much redundancy is required to solve the BG problem?
  First idea: Majority voting among 3 parties, with 1 illoyal.
  HOWEVER:



ORAL MESSAGES

- DEF.: messages whose content is 100% determined by the sender.
- This means that sender can give them *ARBITRARY CONTENT* without the receiver being able to see this.
- How much redundancy is required to solve the BG problem?
  First idea: Majority voting among 3 parties, with 1 illoyal.
  HOWEVER:



  - These situations look identical, seen from L1.
  - But if Commander is loyal, L1 must attack
    and if Commander is illoyal, L1 should (maybe) retreat.

ORAL MESSAGES

- DEF.: messages whose content is 100% determined by the sender.
- This means that sender can give them *ARBITRARY CONTENT* without the receiver being able to see this.
- How much redundancy is required to solve the BG problem?
  First idea: Majority voting among 3 parties, with 1 illoyal.

> GENERAL RESULT:
>
> No solution with ORAL MESSAGES for t illoyal participants, if there are fewer than (3t+1) participants in total.

NOTE: This is not because we require exact agreement! It is also true for approximate agreement.

SOLUTION for ORAL MESSAGES

ASSUMPTIONS:

A1. FAULT-TOLERANCE: At most t unreliable/illoyal participants.

A2. NETWORK: Every message sent arrives; receiver knows who sent it.

A3. TIMING: The absence of a message can be detected.

ALGORITHM OM(n,t): $t > 0$

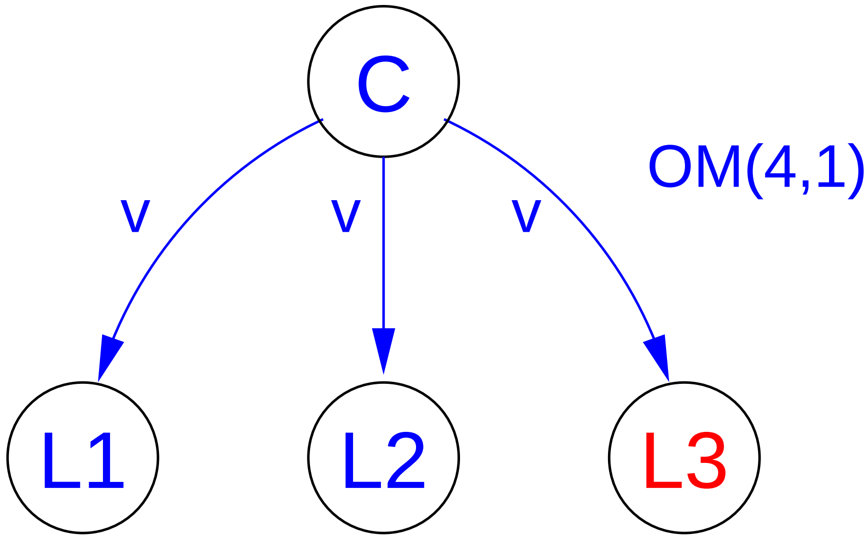
1. Commander sends his value to all (n-1) Lieutenants.
2. For Lieutenant i, let v_i be the value received from his Commander (or v_{def} if no value was received).
   Lieutenant i then acts as Commander in algorithm OM(n-1,t-1), in which he sends v_i to the (n-2) other participants.
3. For each i and each $j \neq i$, let v_j be the value which Lieutenant i received from j in step 2 during OM(n-1,t-1).
   Then i chooses the value majority($v_1, \ldots, v_{n-1}$).

ALGORITHM OM(n,0):

1. Commander sends his value to all (n-1) Lieutenants.
2. Each Lieutenant uses the value received from his Commander (or v_{def} if no value was received).

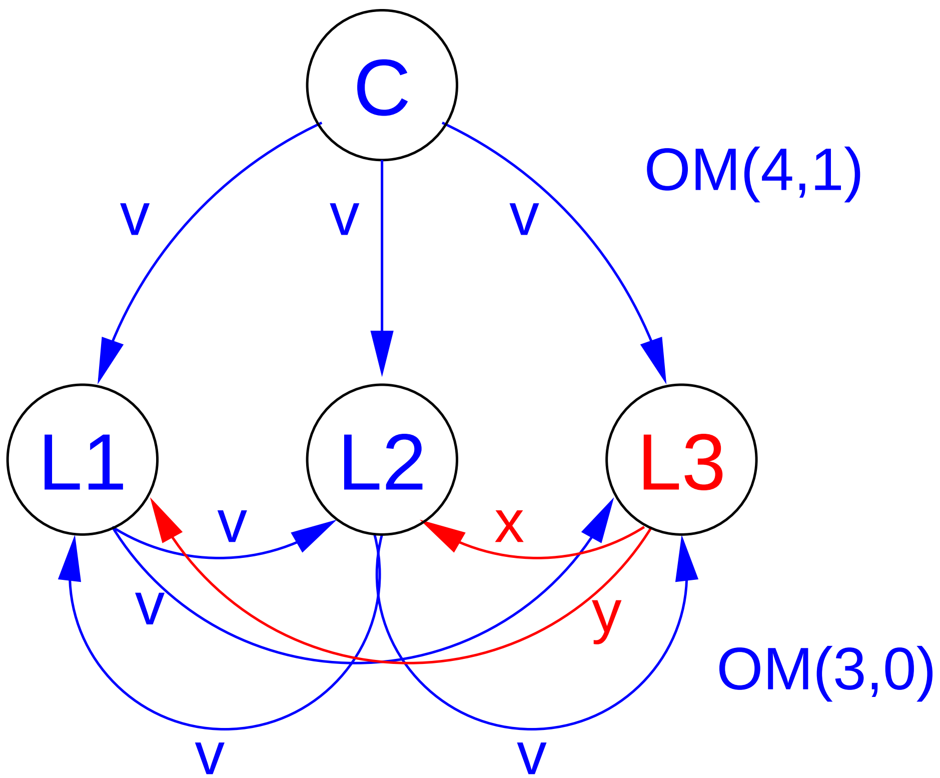
ALGORITHM OM IN ACTION

CASE 1: LOYAL COMMANDER.



ALGORITHM OM IN ACTION

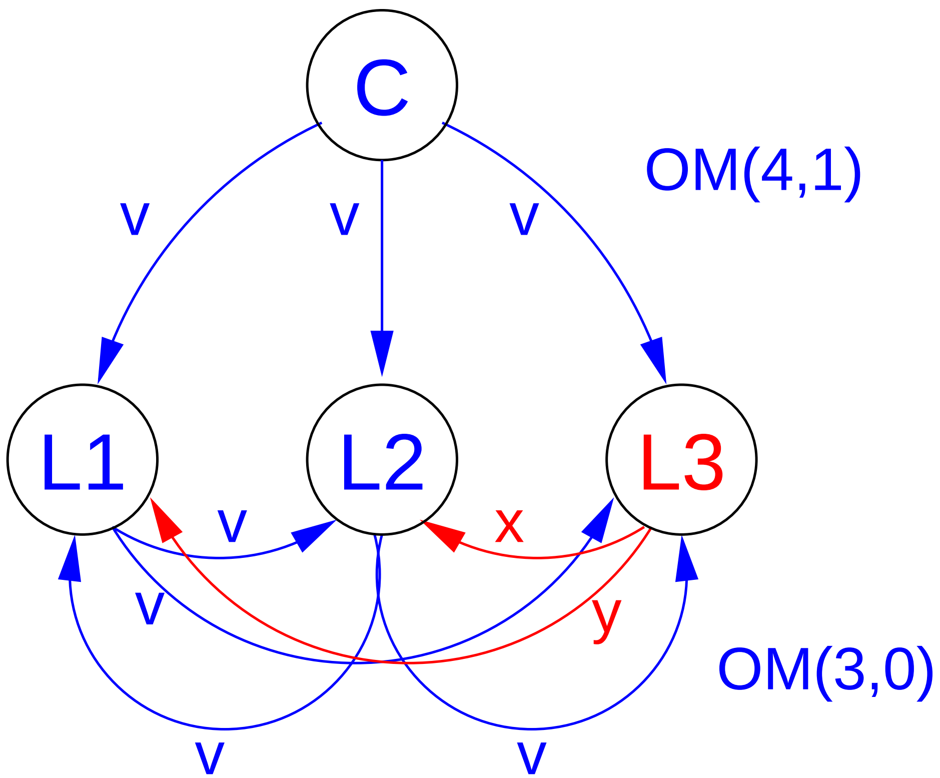
CASE 1: LOYAL COMMANDER.



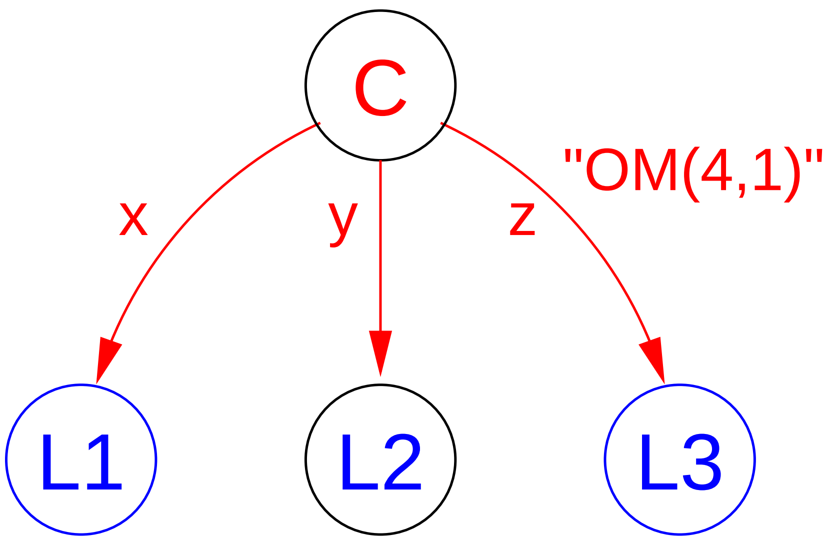
L1, L2 use majority(v,v,*)

ALGORITHM OM IN ACTION

CASE 1: LOYAL COMMANDER. CASE 2: ILLOYAL COMMANDER.

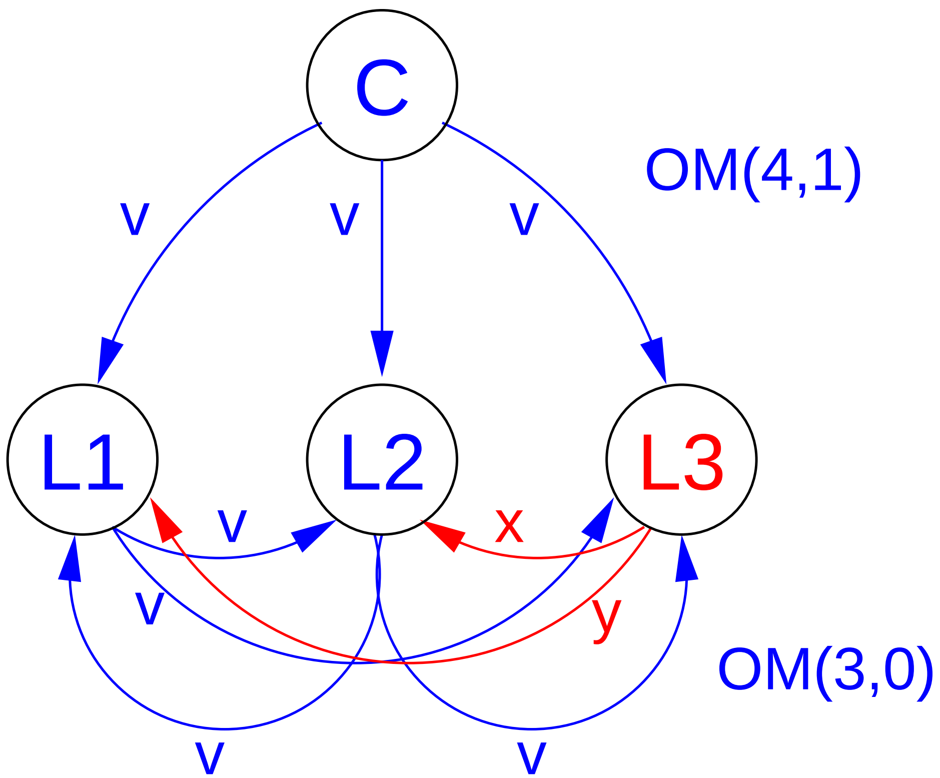


L1, L2 use majority(v,v,*)



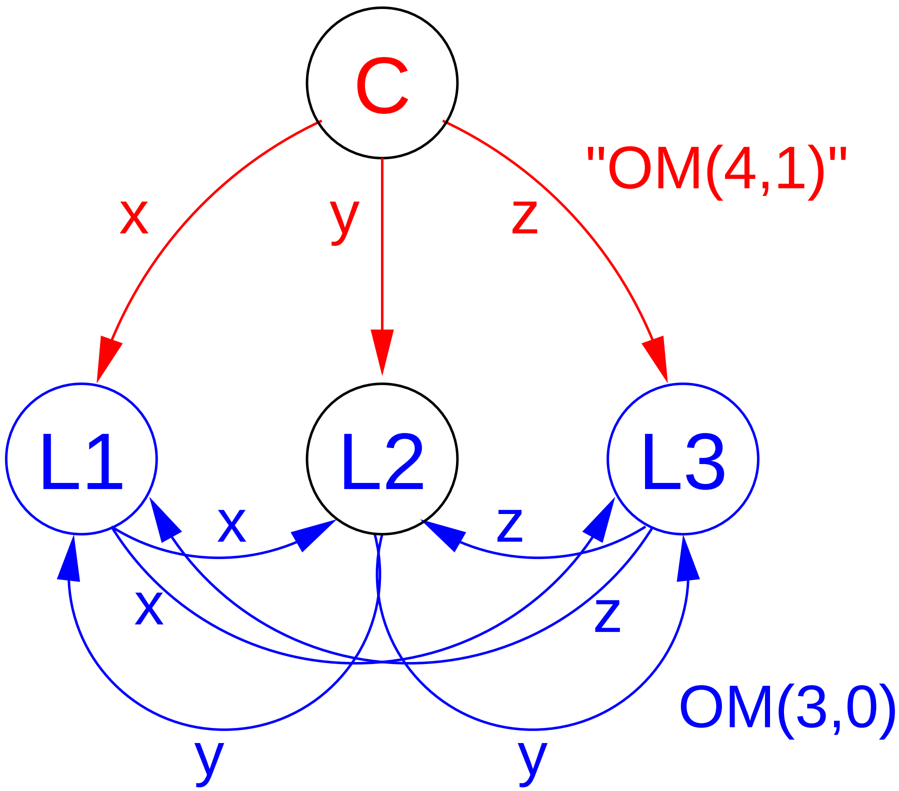
ALGORITHM OM IN ACTION

CASE 1: LOYAL COMMANDER.



L1, L2 use majority(v,v,*)

CASE 2: ILLOYAL COMMANDER.



L1, L2, L3 all use majority(x,y,z)

PROOF of ALGORITHM OM

LEMMA: For arbitrary m, k, OM(n,m) fulfils IC2 if there are:
$> 2k + m$ participants, and $\leq k$ illoyal ones.

PROOF of ALGORITHM OM

LEMMA: For arbitrary m, k, OM(n,m) fulfils IC2 if there are:
$> 2k + m$ participants, and $\leq k$ illoyal ones.

PROOF BY INDUCTION:

1. Trivially true for OM(n,0).
2. Assume true for $(m-1)$, $m > 0$. Then:
   - In step 1, loyal C sends v to all L's.
   - In step 2, each loyal L uses OM(n-1,m-1) with (n-1) generals.
   - Now $(n-1) > 2k + (m-1)$.
   - It then follows from Assumption that each loyal general receives $v_j = v$ from each loyal general j.
   - But $(n-1) > 2k + (m-1) \geq 2k$.
   - So the majority of the (n-1) are loyal!
     Thus all loyal generals use majority($v, v, v, \ldots$), where $> 50\%$ of the values are v's.
   - Thus all loyal generals use value v.

PROOF of ALGORITHM OM

LEMMA: For arbitrary m, k, OM(n,m) fulfils IC2 if there are:
$> 2k + m$ participants, and $\leq k$ illoyal ones.

Corresponding proof for:

THEOREM: For arbitrary m, OM(n,m) fulfils IC1 and IC2 if there are:
$> 3m$ participants, and $\leq m$ illoyal ones.

See Lamport, Shostak & Pease's paper for details!

ANALYSIS of ALGORITHM OM

- In a system with up to t illoyal generals, OM proceeds in (t+1) rounds:
  1. Algorithm OM(n,t) is executed once: C sends to (n-1) lieutenants.

ANALYSIS of ALGORITHM OM

- In a system with up to t illoyal generals, OM proceeds in (t+1) rounds:
  1. Algorithm OM(n,t) is executed once: C sends to (n-1) lieutenants.
  2. Each lieutenant acts as commander for OM(n-1,t-1):
     Each of the $(n-1)$ lieutenants sends to $(n-2)$ generals.
     Thus each general receives $(n-2)$ messages.

ANALYSIS of ALGORITHM OM

- In a system with up to t illoyal generals, OM proceeds in (t+1) rounds:
  1. Algorithm OM(n,t) is executed once: C sends to (n-1) lieutenants.
  2. Each lieutenant acts as commander for OM(n-1,t-1):
     Each of the $(n-1)$ lieutenants sends to $(n-2)$ generals.
     Thus each general receives $(n-2)$ messages.
  3. For each message received in (2), the receiver acts as commander for OM(n-2,t-2):
     Each general sends $(n-3)(n-2)$ messages in total.
     Thus each general receives $(n-3)(n-2)$ messages.
  4. etc., etc.

ANALYSIS of ALGORITHM OM

- In a system with up to t illoyal generals, OM proceeds in (t+1) rounds:
  1. Algorithm OM(n,t) is executed once: C sends to (n-1) lieutenants.
  2. Each lieutenant acts as commander for OM(n-1,t-1):
     Each of the $(n-1)$ lieutenants sends to $(n-2)$ generals.
     Thus each general receives $(n-2)$ messages.
  3. For each message received in (2), the receiver acts as commander for OM(n-2,t-2):
     Each general sends $(n-3)(n-2)$ messages in total.
     Thus each general receives $(n-3)(n-2)$ messages.
  4. etc., etc.
- Thus OM(n,t) is executed once.
  OM(n-p,t-p) is exec. $(n-1)\cdots(n-p)$ times, for successive $p \in \{1..t\}$.

ANALYSIS of ALGORITHM OM

- In a system with up to t illoyal generals, OM proceeds in (t+1) rounds:
  1. Algorithm OM(n,t) is executed once: C sends to (n-1) lieutenants.
  2. Each lieutenant acts as commander for OM(n-1,t-1):
     Each of the $(n-1)$ lieutenants sends to $(n-2)$ generals.
     Thus each general receives $(n-2)$ messages.
  3. For each message received in (2), the receiver acts as commander for OM(n-2,t-2):
     Each general sends $(n-3)(n-2)$ messages in total.
     Thus each general receives $(n-3)(n-2)$ messages.
  4. etc., etc.
- Thus OM(n,t) is executed once.
  OM(n-p,t-p) is exec. $(n-1)\cdots(n-p)$ times, for successive $p \in \{1..t\}$.
- So total number of messages sent is:
  $$s = (n-1) + (n-1)(n-2) + \ldots + (n-1)(n-2)\cdots(n-t-1)$$
  Thus s is $O(n^t)$, i.e. it is exponential in t.

ANALYSIS of ALGORITHM OM

- In a system with up to t illoyal generals, OM proceeds in (t+1) rounds:
  1. Algorithm OM(n,t) is executed once: C sends to (n-1) lieutenants.
  2. Each lieutenant acts as commander for OM(n-1,t-1):
     Each of the $(n-1)$ lieutenants sends to $(n-2)$ generals.
     Thus each general receives $(n-2)$ messages.
  3. For each message received in (2), the receiver acts as commander for OM(n-2,t-2):
     Each general sends $(n-3)(n-2)$ messages in total.
     Thus each general receives $(n-3)(n-2)$ messages.
  4. etc., etc.
- Thus OM(n,t) is executed once.
  OM(n-p,t-p) is exec. $(n-1)\cdots(n-p)$ times, for successive $p \in \{1..t\}$.
- So total number of messages sent is:

$$s = (n-1) + (n-1)(n-2) + \ldots + (n-1)(n-2)\cdots(n-t-1)$$

Thus s is $O(n^t)$, i.e. it is exponential in t.

An expensive algorithm, but resistant to (up to) t faults.

SIGNED MESSAGES

REVISED ASSUMPTIONS:

A1. FAULT-TOLERANCE: At most t unreliable/illoyal participants.

A2. NETWORK: Every message sent is delivered, and receiver knows who sent it.

A3. TIMING: The absence of a message can be detected.

A4. SIGNATURES: A signature cannot be forged, and any changes to a signed message can be seen. Anybody can verify a signature.

Assumption A4 implies that only possible misbehaviour is to OMIT TO PASS ON a message.

With these assumptions, IC1 and IC2 can be fulfilled for arbitrary number of faults, i.e. $n > (t + 1)$.

Algorithm SM(m).

Initially $V_i = \emptyset$.

(1) The commander signs and sends his value to every lieutenant.
(2) For each i:

 (A) If Lieutenant i receives a message of the form $v:0$ from the commander and he has not yet received any order, then

  (i) he lets V_i equal $\{v\}$;
  (ii) he sends the message $v:0:i$ to every other lieutenant.

 (B) If Lieutenant i receives a message of the form $v:0:j_1:\cdots:j_k$ and v is not in the set V_i, then

  (i) he adds v to V_i;
  (ii) if $k < m$, then he sends the message $v:0:j_1:\cdots:j_k:i$ to every lieutenant other than $j_1, \ldots, j_k$.

(3) For each i: When Lieutenant i will receive no more messages, he obeys the order *choice*(V_i).

PROOF of ALGORITHM SM(n,t)

THEOREM 2: For arbitrary $t \geq 0$, algorithm SM(n,t) solves BG problem for at most t illoyal generals.

PROOF of ALGORITHM SM(n,t)

THEOREM 2: For arbitrary $t \geq 0$, algorithm SM(n,t) solves BG problem for at most t illoyal generals.

PROOF FOR IC2: (Assuming loyal Commander)

- Loyal Commander $\Rightarrow$ all generals get $(v, \{0\})$ in first round.
- In second round, all loyal $P[i, ...]$ send $(v, \{0, i\})$ to everyone except $0, i$.
  Thus everyone gets two copies of v.
  Thus everyone terminates with $V = \{v\}$

PROOF of ALGORITHM SM(n,t)

THEOREM 2: For arbitrary $t \geq 0$, algorithm SM(n,t) solves BG problem for at most t illoyal generals.

PROOF FOR IC1: (Only relevant for illoyal Commander)

- Loyal generals must terminate with same V.
- Assume $P[i, ...]$ receives (v, ss) in round k, where $v \notin V$. Afterwards, $v \in V$ in i. There are then two cases:
  1. $j \in ss$: Then j's V must already contain v.
  2. $j \notin ss$:
     (a) card $ss < (t+1)$ $\Rightarrow$ i sends v to j.
     (b) card $ss = (t+1)$ $\Rightarrow$ no more rounds.
     BUT at least 1 of the $(t+1)$ must be loyal, and so must have sent v to j when it first received v.

  CONCLUSION: If $v \in V$ in i, then $v \in V$ in j. So both terminate with the same V.

Atomic Transactions

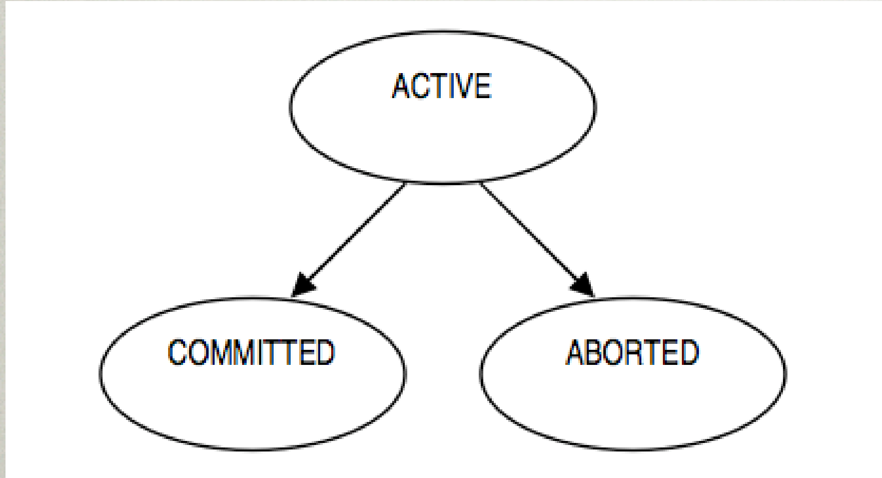
- Atomic agreement: either ALL or NONE should happen
- Storage is R/W
- Processes can change status, send/rcv messages, r/w storage

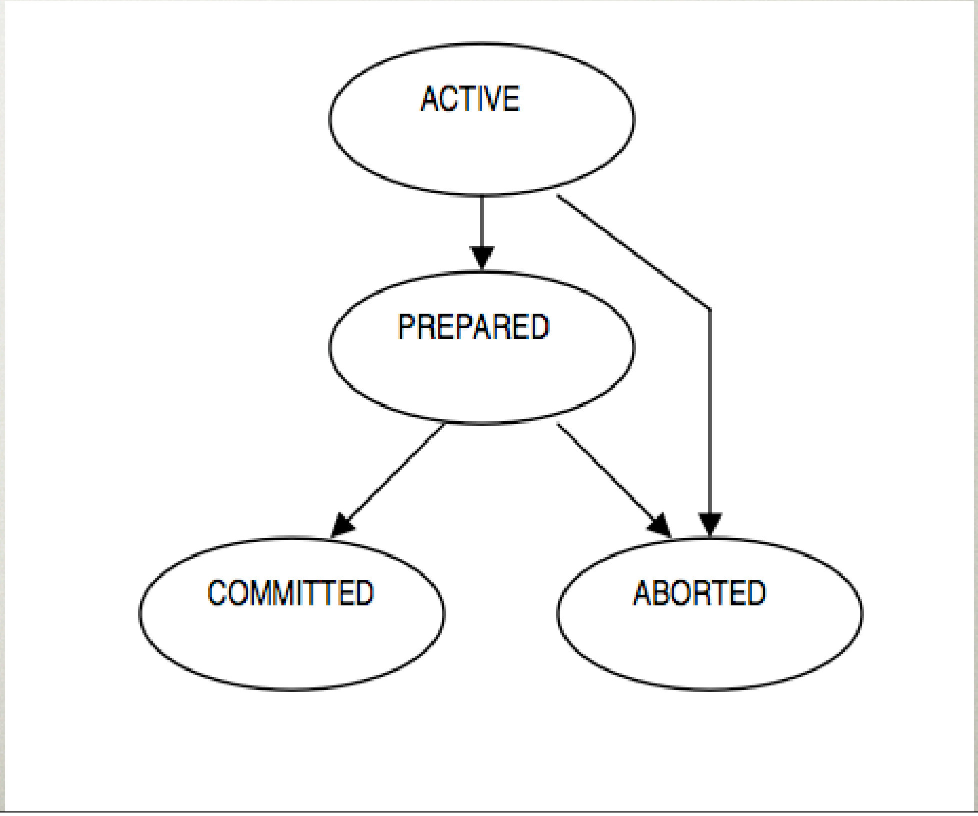
Faults (all detectable)

- Storage write may fail/corrupt/decay
- Processes may lose state
- Messages may be delayed/corrupted/lost

COMMIT Problem

- Given N stable processes, find an algorithm which forces all processes to COMMIT or ABORT





Simple Solution

- Store a record of intentions
- When abort is no more possible...
- keep sending a “please, commit” message until it is acknowledged
- No guarantee on the worst case

Byzantine vs. Commit

- Accept N/3 faults
- Some agree
- Unknown answer if too many faults
- Bounded time
- Redundant proc. and messages

- Accept N faults
- All agree
- Fail-fast
- Unbounded time
- No redundancy

Example

- A Byzantine ATM system could have an incoherent status, but it responds in bounded time
- A Commit ATM can be delayed, but it is always coherent
- Commit is good if failure is rare