The Byzantine Generals Problem

(Lamport-Shostak-Pease)

The Byzantine Generals Problem

(Lamport-Shostak-Pease)

"I have long felt that, because it was posed as a cute problem about philosophers seated around a table, Dijkstra's dining philosopher's problem received much more attention than it deserves.

…

The popularity of the dining philosophers problem taught me that the best way to attract attention to a problem is to present things in terms of a story."

— Lamport

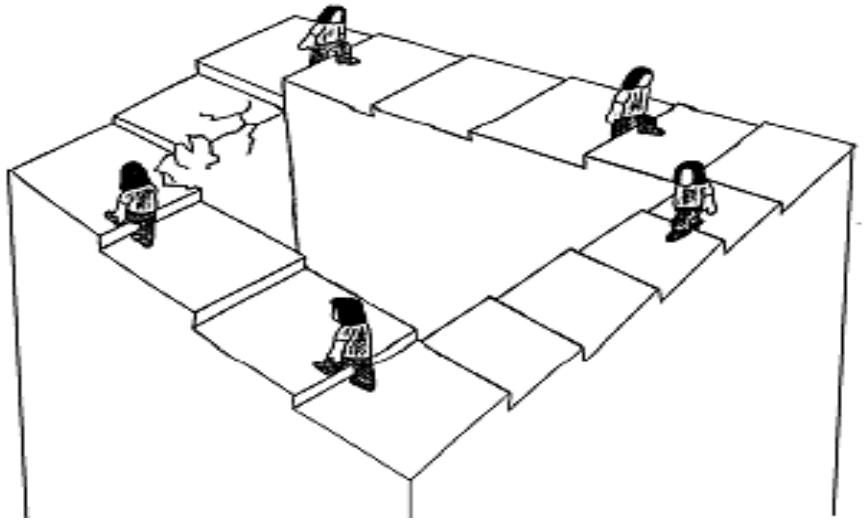
The Byzantine Generals Problem

A general must send an order to his $n - 1$ lieutenants such that the following Interactive Consistency conditions hold:

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

Note:
- IC1 follows from IC2 when the general is loyal.
- IC2 is vacuous when the general is a traitor

2. Impossibility Results



Our army

Traitor

Loyal

General

Impossibility Results

Definition:

An oral message is one whose contents are completely under the control of the sender

(e.g. authenticity of information forwarded by traitors cannot be verified).

We make the following assumptions:

A1. Every message that is sent is delivered correctly.

A2. The receiver of a message knows who sent it.

A3. The absence of a message can be detected.

Such a message corresponds to the type of message that computers normally send to one another.

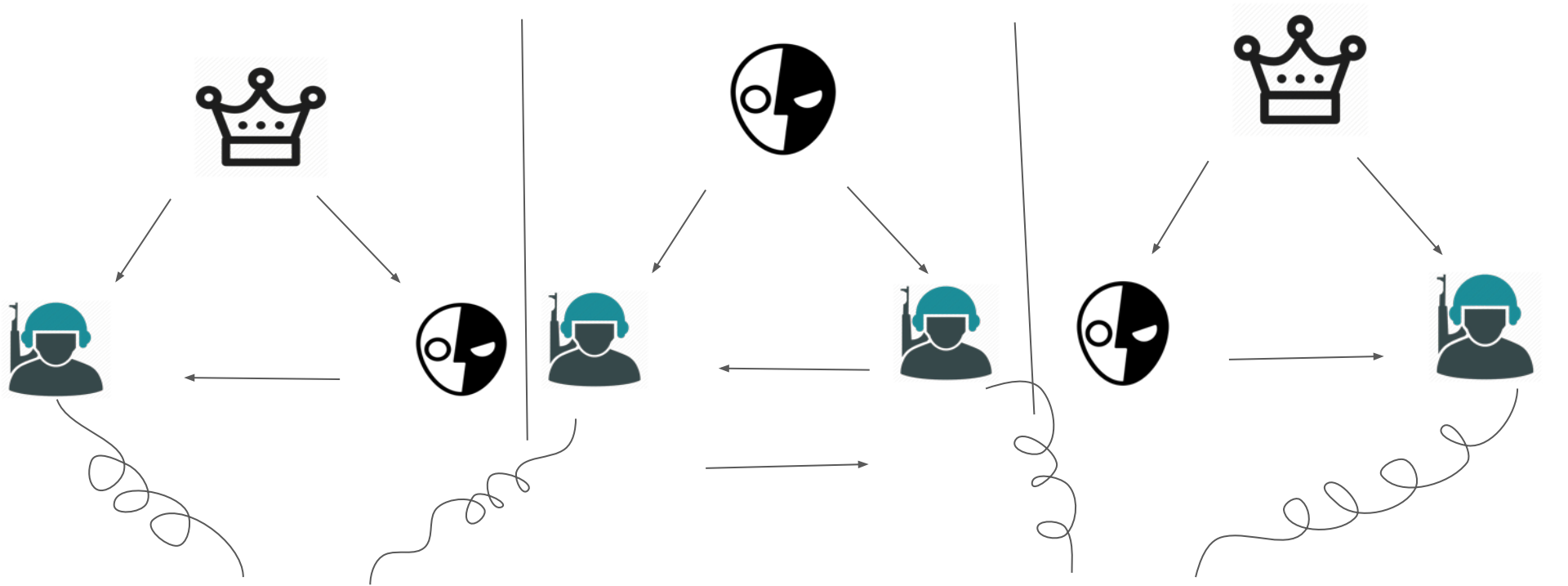
Impossibility Results (cont)

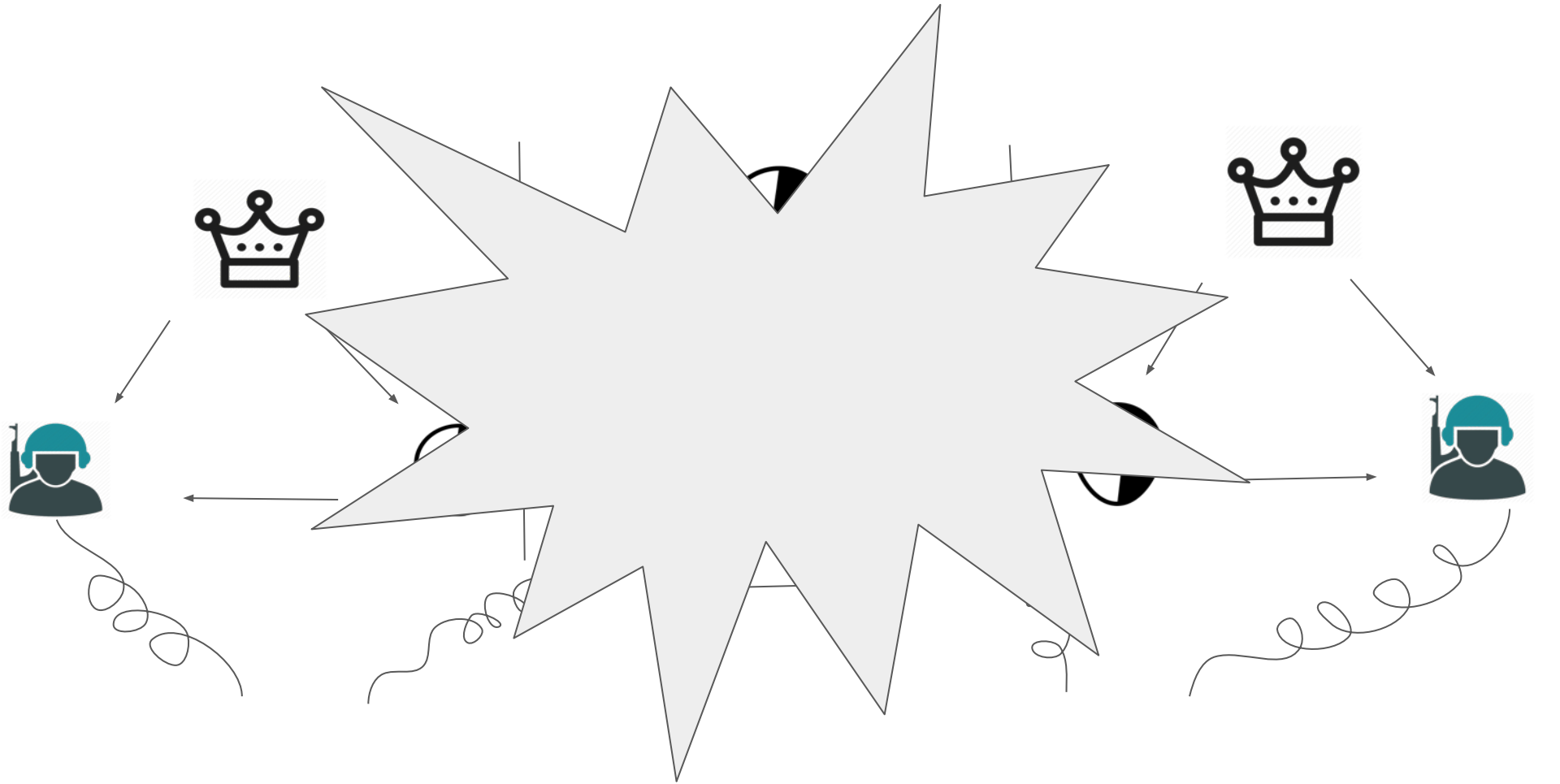
Claim:

If only oral messages are used,

then with only 1 general and 2 lieutenants,

no protocol can work in the presence of a single traitor.





Proof (cont.)

- In the first case, the loyal lieutenant (2) receives two messages:
  1. Attack from the general.
  2. Retreat from another lieutenant.
- Since the general is loyal, he must attack in order to fulfill IC2.
- In the second case, the loyal lieutenant (2) receives the same two messages than in the previous case, so he has to attack.
- But by symmetry, lieutenant (1) has to retreat → ~~IC1~~

Impossibility Results

Claim:

If only oral messages are used,

then no protocol will work unless more than two-thirds

of the nodes are loyal.

Proof:

By a reduction.

Assume there is a solution for *3k* nodes to cope with *k* traitors.

We can build from this assumption a solution for 3 nodes to cope with a single traitor.

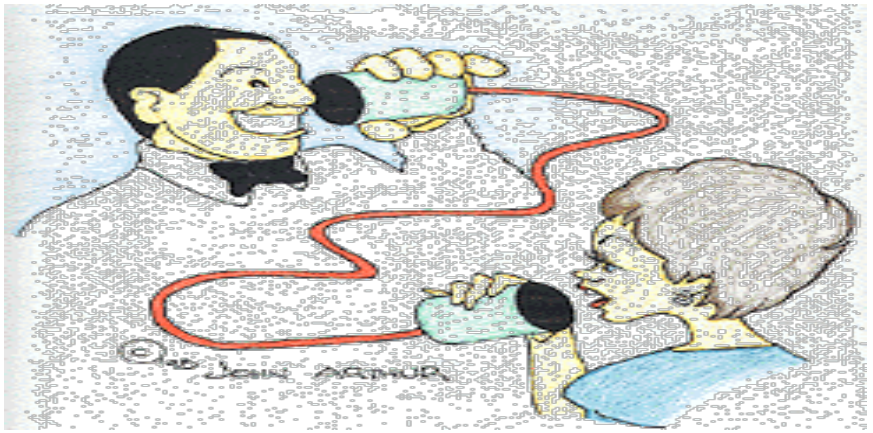
The reduction

- We want to build a solution for 3 nodes (say *A, B* and *C)* in the presence of a single traitor.
- Each one of *A, B* and *C* subdivides in *k* sub-nodes, according to his loyalty. Since one of them is a traitor, there will be *k* traitors in the new problem (each one of *A, B* and *C* knows if he is a traitor or not).

- We get *3k* nodes, including *k* traitors.
- From reduction's assumption, there is a solution to this new problem.

- *A, B* and *C* run this solution and take the same decision that all their sub-nodes take.
- Since IC1, IC2 hold in the extended problem, they also hold in the starting problem (to prove as exercise)
  → we get a solution to an unsolvable problem.

3. A solution with Oral Messages



Oral Messages - Definition

Definition:

An oral message is one whose contents are completely under the control of the sender

(e.g. authenticity of information forwarded by traitors cannot be verified).

We make the following assumptions:

A1. Every message that is sent is delivered correctly.
A2. The receiver of a message knows who sent it.
A3. The absence of a message can be detected.

Additional assumption:

The communication graph of the nodes is **complete**.

Algorithm OM - Intuition

- Solution without traitors (k=0): any ideas ?
  - ➔ Believe what the general tells you…

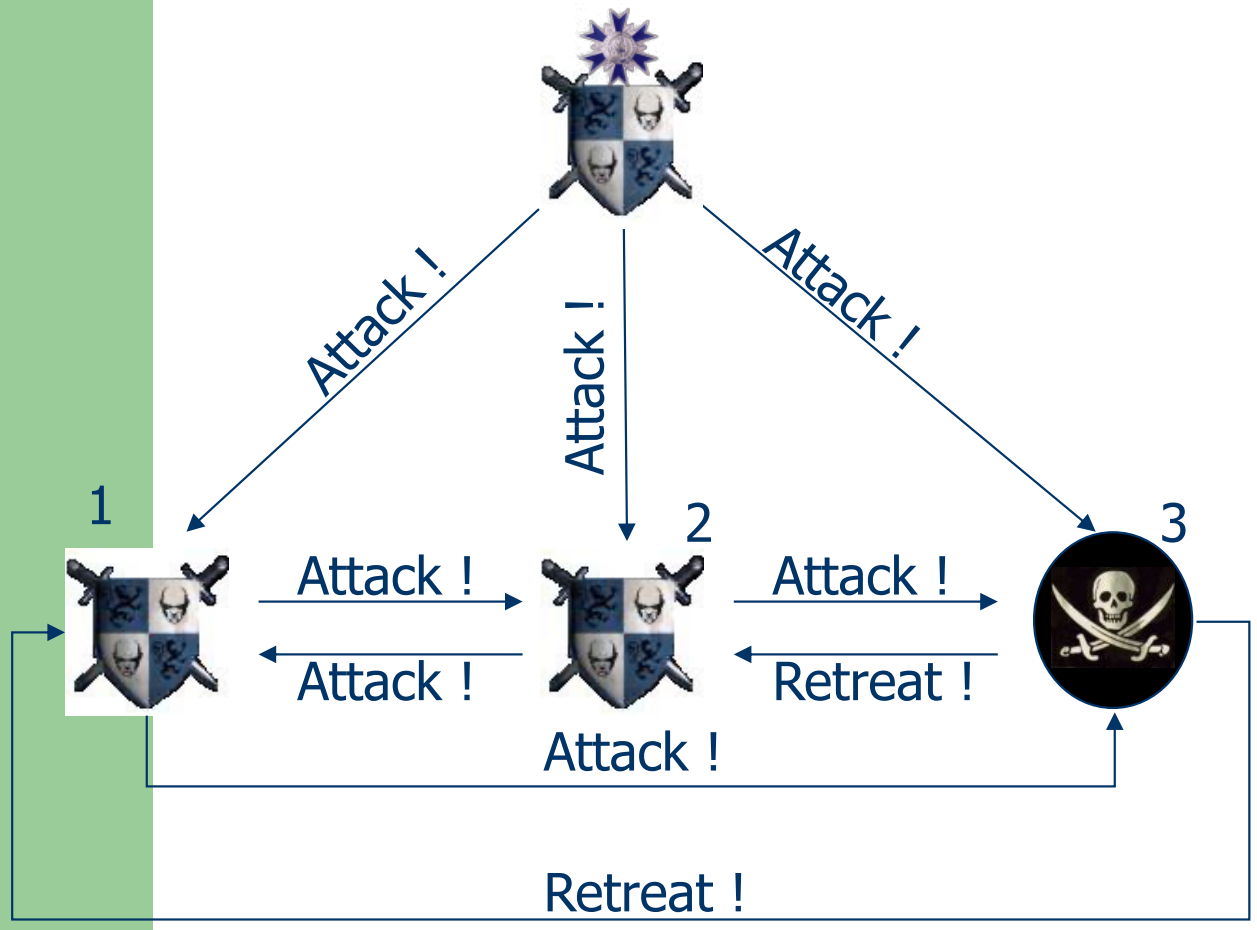
- Solution with one traitor (k=1)…
  - ➔ If the traitor is a lieutenant, no need to change the algorithm
  - ➔ If the traitor is the general, our algorithm fails…

    In this case, we have to restore validity of IC1.

    If each lieutenant knows all the values that the general sent, the problem is solved (majority).

    This solution holds also for the case above (k=0).

Example: n = 4 and k = 1



Lieutenant 1 received:

- Attack from the general and 2
- Retreat from the lieutenant 3

He chooses Attack

Lieutenant 2 received:

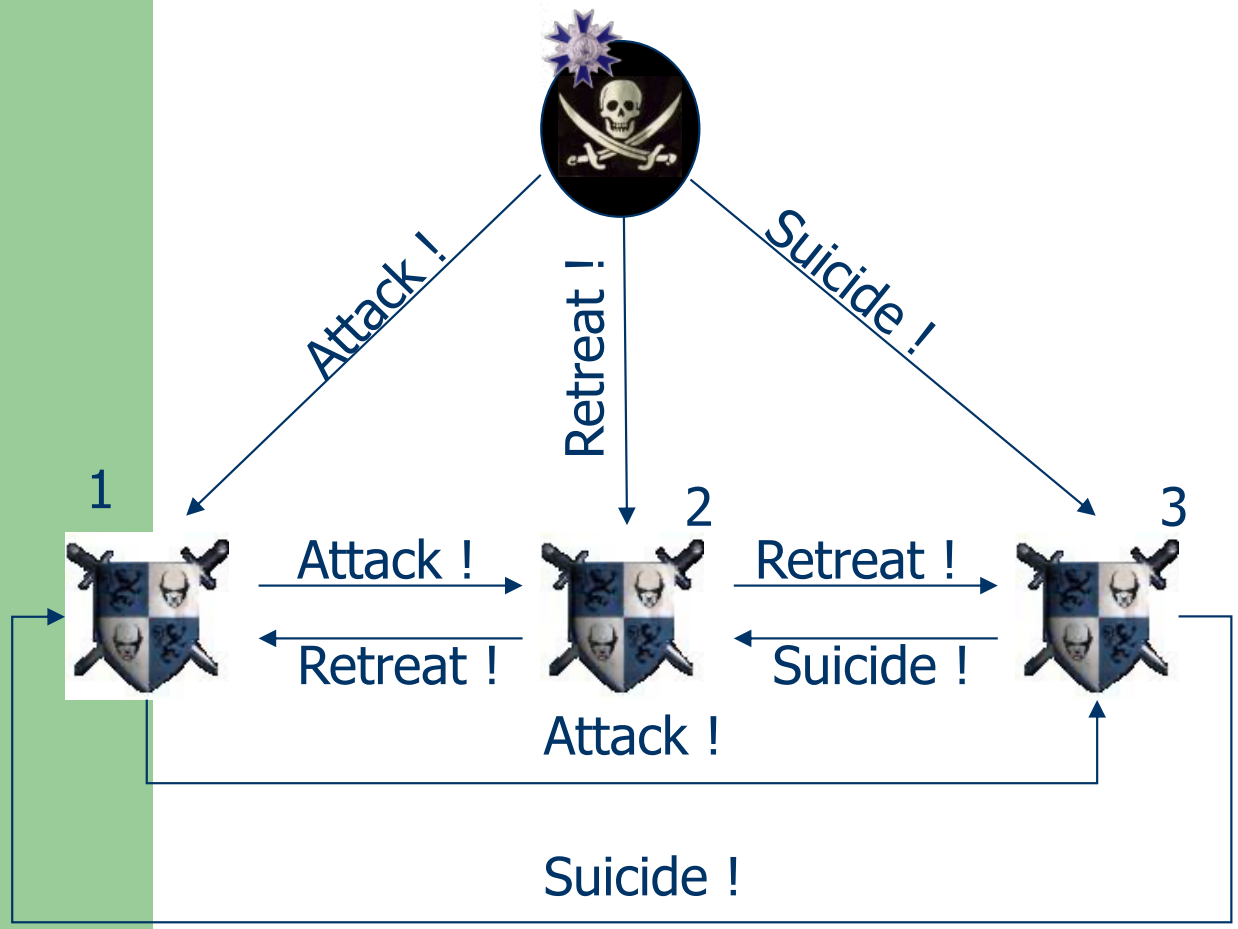
- Attack from the general and 1
- Retreat from the lieutenant 3

He chooses Attack

IC1 ☑

IC2 ☑

Example: n = 4 and m = 1



Lieutenant 1 chooses

maj (*Attack*, *Retreat*, *Suicide*)

Lieutenant 2 chooses

maj (*Attack*, *Retreat*, *Suicide*)

Lieutenant 3 chooses

maj (*Attack*, *Retreat*, *Suicide*)

IC1 ☑

IC2 ☑ - vacuous

OM - Intuition two traitors

- If the two traitors are lieutenants the aforementioned algorithm works: take the majority of what all the lieutenants received.

- If the general and a lieutenant are traitors, not so easy…
  - ➔ Each loyal lieutenant may compute a different majority.
  - ➔ In order to satisfy IC1, the loyal lieutenants have to
    1. agree on the value the traitor lieutenant sent.
    2. compute the majority with the value computed in 1.

OM(*k*) - The algorithm

(a.k.a. EIG: Exponential Information Gathering)

- *k* = number of traitors.
- OM(0)
  1. The general sends his value to every lieutenant.
  2. Each lieutenant decides with the value he received from the general, or a default value if no message is received.
- OM(*k*), $k > 0$
  1. The general sends his value to every lieutenant.
  2. For each lieutenant *i*:
     Let v_i be the value *i* received from the general.
     Now, *i* acts as general and runs Algorithm OM(*k-1*), sending *v* to all other lieutenants (old general is excluded).
  3. For each pair of different lieutenants *i*,*j*,
     Let v_i be the value *j* received from *i* in step 2.
     Now, *j* decides with value majority(v_1, v_2, …).

OM(*k*) - Remarks

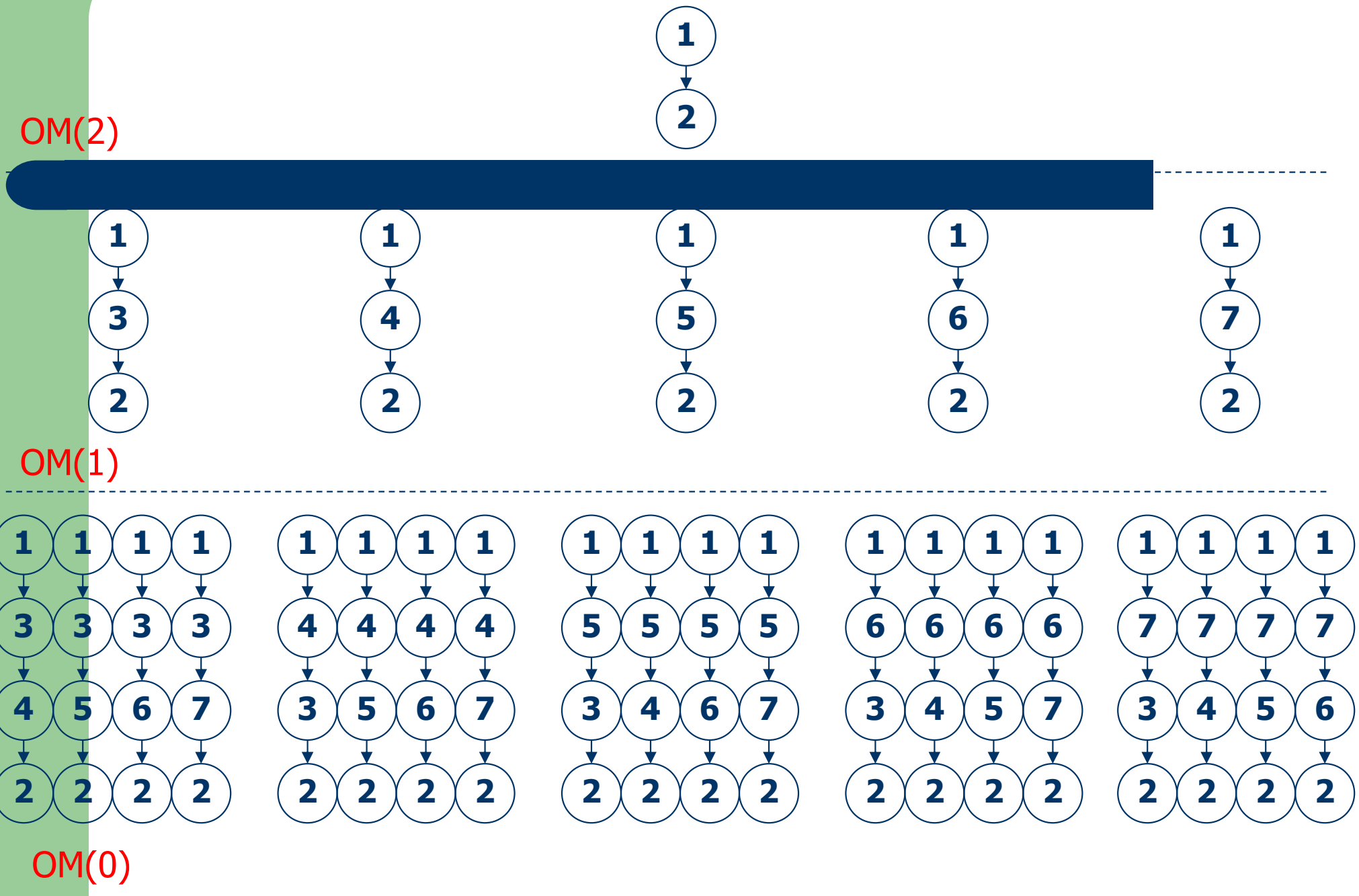
- Since each lieutenant sends many separate messages to each other lieutenant, there must be some way to distinguish among these different messages

  → each lieutenant *i* prefixes the number *i* to the value *v* he sends in step 2.

- In step 2 lieutenant *i* runs OM(*k-1*) to send *v* to each lieutenant that does not appear in the list of senders of *v* .

Messages received by the lieutenant 2

In OM(2), OM(1), OM(0).



OM(k) - Message complexity

- OM(k) invokes $n - 1$ separate executions of OM($k - 1$).
- Each execution of OM($k - 1$) invokes $n - 2$ separate executions of OM($k - 2$) etc…

  ➔ $(n - 1)*(n - 2)*(n - 3)*\ldots*(n - k) = O(n^k)$ messages are sent.

- For $n = 3k + 1$, the algorithm is exponential in n.

OM(*k*) - Proof of Correctness

Lemma 1.

For any *h* and *k*, algorithm OM(*h*) satisfies IC2 if there are more than $2k + h$ nodes and at most *k* traitors.

Note.

If the general is a traitor, lemma 1 is true (why ?),
so we will only prove the case in which the general is loyal.

Proof: By induction on *h*.

h = 0: Trivial.

hypothesis: Lemma 1 holds for $h - 1$, $h > 0$.
We will prove it for *h*.

OM(*k*) - Proof of Correctness

Proof (cont).

In step 1, the loyal general sends a value *v* to all *n* – 1 lieutenants.

In step 2, each loyal lieutenant *j* applies OM(*h* – 1) behaving as general, with the *n* – 2 other lieutenants acting as *its* lieutenants.

1. $n > 2k + h$, thus $(n - 1) > 2k + (h - 1) \geq 2k$
2. Each running of OM(*h* – 1) includes at most *k* traitors.

From hypothesis, each lieutenant *j* gets *v* from each loyal *i*.

Since there is a majority of loyal lieutenants, the majority function will give the same result: *v*.

OM(*k*) - Proof of Correctness

Theorem 1.

For any *k*, algorithm OM(*k*) satisfies conditions IC1 and IC2 if there are at least 3*k* lieutenants and at most *k* traitors.

Proof: By induction on *k*.

k = 0: Trivial.

hypothesis: Theorem 1 holds for $k - 1$, $k > 0$.

We will prove it for *k*.

OM(*k*) - Proof of Correctness

Note.

If the general is loyal,
Theorem 1 follows from Lemma 1 with $k = m$:
IC2 holds and IC1 is implied by IC2.
So we focus on the case in which the general is a traitor.

Proof (cont).

In step 2, each loyal lieutenant *j* applies OM(k – 1) with n – 2 other lieutenants acting as *its* lieutenants.

1. $n > 3k$, thus $(n - 1) > 3(k - 1)$.
2. Each running of OM(k – 1) includes at most k - *1* traitors lieutenants.

OM(*k*) - Proof of Correctness

Proof (cont).

From hypothesis (IC1 holds for each running of OM(k – 1)), we get that each loyal lieutenant gets the same value for each lieutenant, and therefore obtain the same value in the majority computation of step 3, proving IC1.

4. A Solution with Signed Messages

Signed messages - Definition

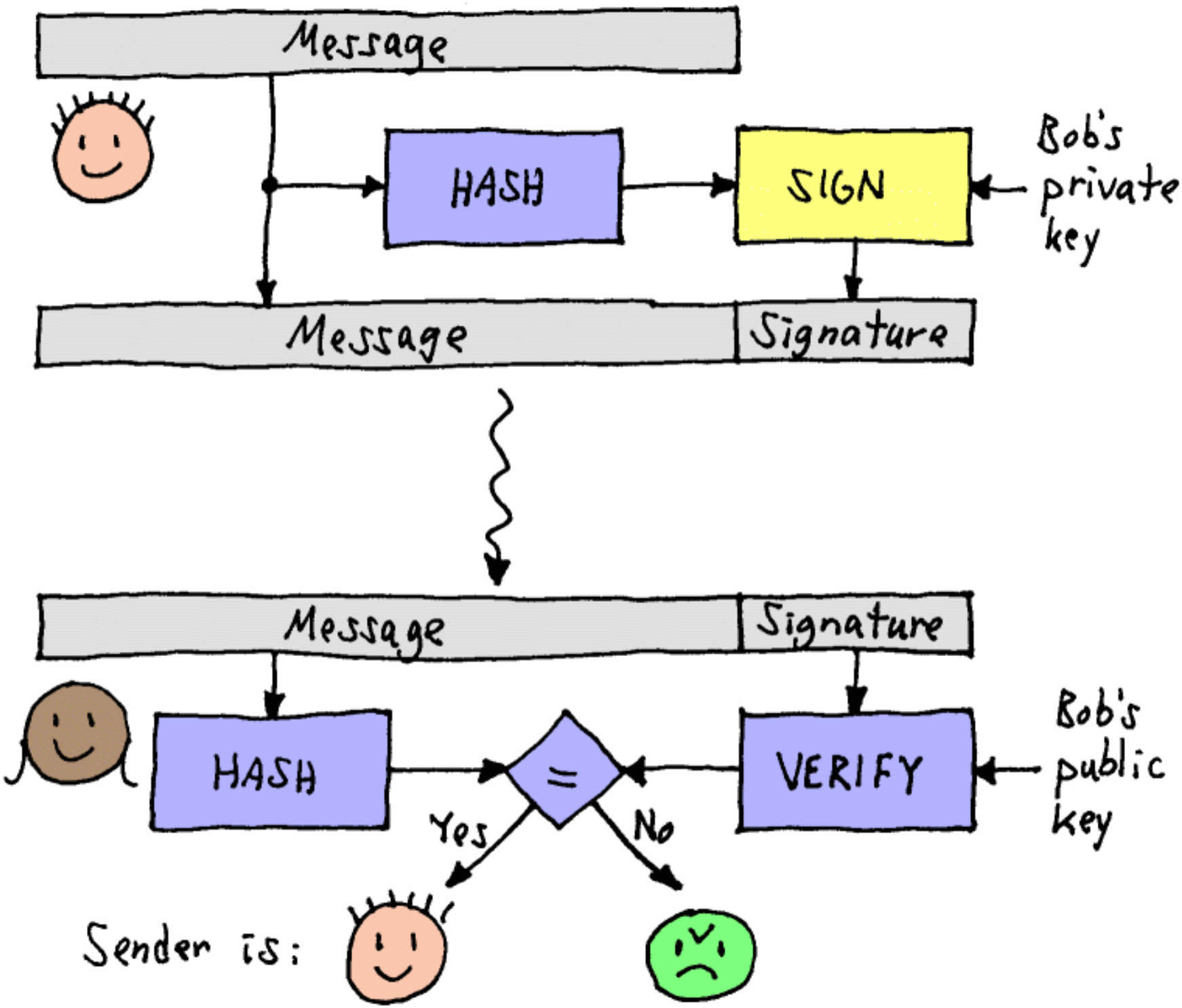
- It is the traitor's ability to lie that makes the Byzantine Broadcast Problem so difficult.

- One way to restrict that ability is to allow nodes to send *unforgeable* signed messages.

- When the messages are signed, we add the following assumption to A1, A2 and A3:

  A4. (a) A loyal node's signature cannot be forged, and any alteration of the contents of his signed messages can be detected.

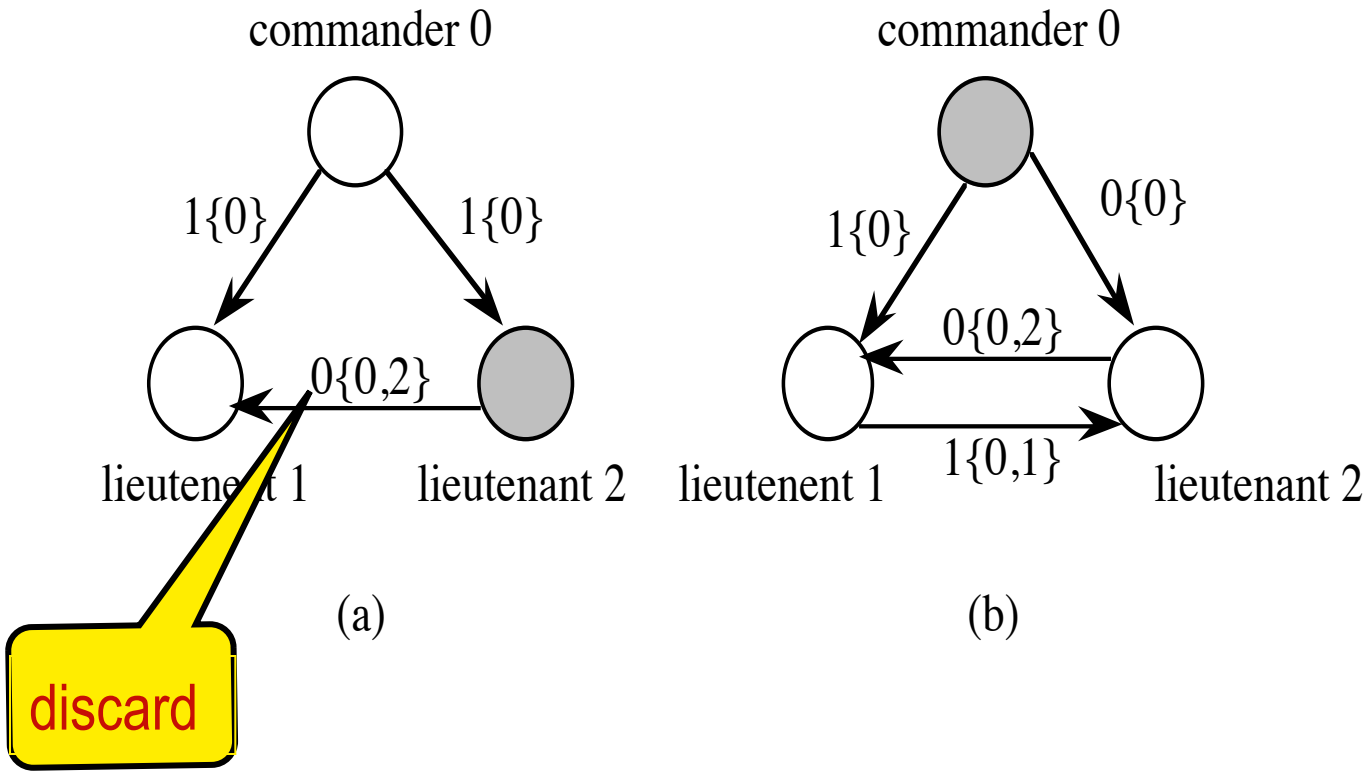
  (b) Anyone can verify the authenticity of a node's signature.


Message
HASH
SIGN
Bob's private key
Message
Signature
Message
Signature
HASH
=
VERIFY
Bob's public key
Yes
No
Sender is:

Signed messages - Definition

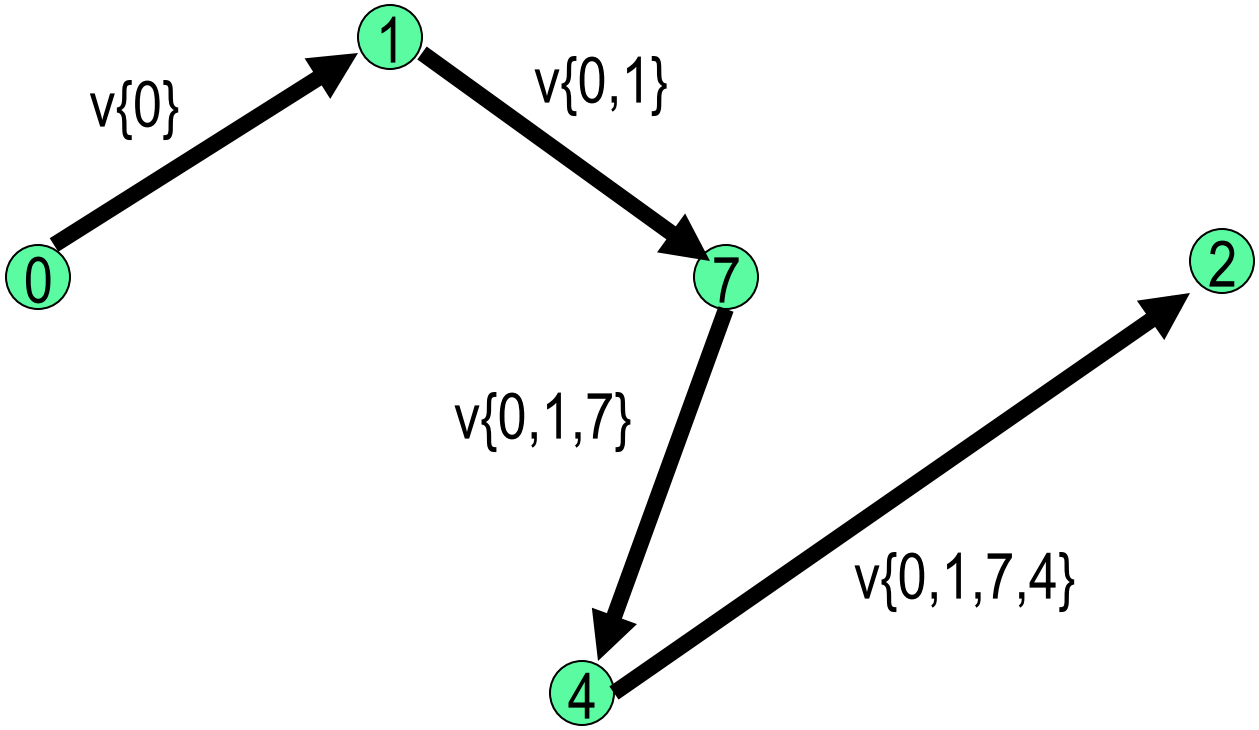
- Note that we made no assumption about a traitor's signature → we permit collusion among the traitors!
- There is a solution to cope with *k* traitors for any number n of nodes (the problem becomes vacuous for $n < k$).
- We assume the existence of a common *choice* function which maps a set of orders to an order:
  1. V = {v} → *choice*(V) = v
  2. *choice*(empty) = RETREAT

Example



Using signed messages, byzantine consensus ***is feasible*** *with 3 generals and 1 traitor. In (b) the the loyal lieutenants compute the consensus value by applying some choice function on the set of values*

Signature list



Byzantine consensus:
The signed message algorithms SM(m)

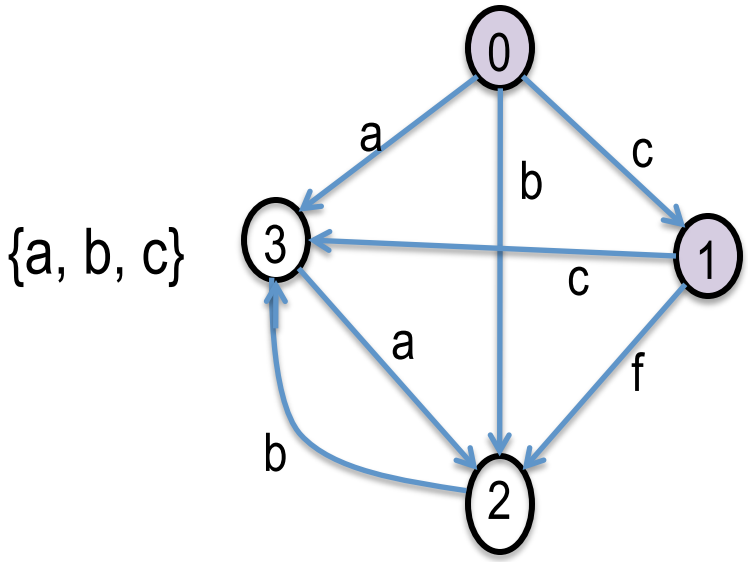
Commander **i** sends out a signed message **v{i}** to each lieutenant j ≠ i

Lieutenant **j**, after receiving a message **v{S}**, appends it to a set **V.j,** only if
(i) it is not forged, and (ii) it has not been received before.

If the *length* of **S** is less than **m+1**, then lieutenant **j**
(i) appends his own signature to **S**, and
(ii) sends out the signed message to every other lieutenant
whose signature does not appear in **S**.

Lieutenant **j** applies a choice function on **V.j** to make the final decision.

Example



3 accepts c,
but 2 rejects f

With m=2 and a signature list of length 2, the loyal generals may not receive the same order from the commander who is a traitor.
When the length of the signature list grows to 3, the problem is resolved

Theorem of signed messages

If **n ≥ m + 2**, where **m** is the maximum number of traitors, then **SM(m)** satisfies both **IC1** and **IC2**.

Proof.

Case 1. Commander is loyal. The bag of each process will contain exactly one message, that was sent by the commander.

(Try to visualize this)

Proof of signed message theorem

Case 2. Commander is traitor.

- The signature list has a size (m+1), and there are m traitors, so at least one lieutenant signing the message must be loyal.

- Every loyal lieutenant i will receive every other loyal lieutenant's message. So, every message accepted by **j** is also accepted by **i** and vice versa. So **V.i = V.j**.

Concluding remarks

- The signed message version tolerates a larger number (n-2) of faults.

- Message complexity however is the same in both cases.

  Message complexity = (n-1)(n-2) … (n-m+1)

Exercise:

Reach consensus (only lieutenants, no commanding general -> full democracy!) with faulty/traitors nodes, using a solution to the Byzantine Generals Problem…

Bracha-Toueg Byzantine consensus algorithm

Let $k < \frac{N}{3}$.

Again, in every round, each correct process:

- broadcasts its value,
- waits for $N - k$ incoming messages, and
- changes its value to the majority of votes in the round.

(No weights are needed.)

A correct process decides b if it receives $> \frac{N+k}{2}$ b-votes in one round.

Bracha-Toueg Byzantine consensus algorithm

Let $k < \frac{N}{3}$.

Again, in every round, each correct process:

- broadcasts its value,
- waits for $N - k$ incoming messages, and
- changes its value to the majority of votes in the round.

(No weights are needed.)

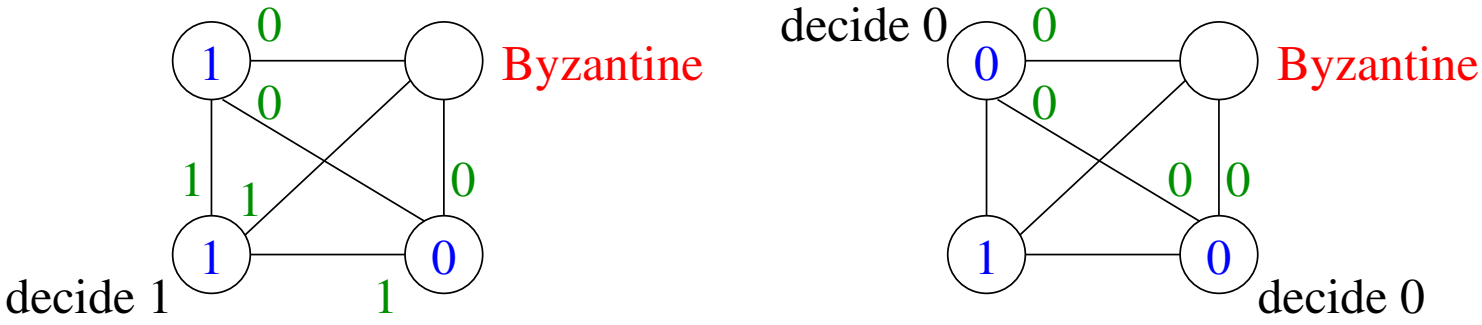
A correct process decides b if it receives $> \frac{N+k}{2}$ b-votes in one round.

The other processes interpret <decide b> as a b-vote for all rounds to come (otherwise the process who decided may have been Byzantine…)

Echo mechanism

Complication: A Byzantine process may send different votes to different processes.

Example: Let $N = 4$ and $k = 1$. Each round, a correct process waits for three votes, and needs three b-votes to decide b.



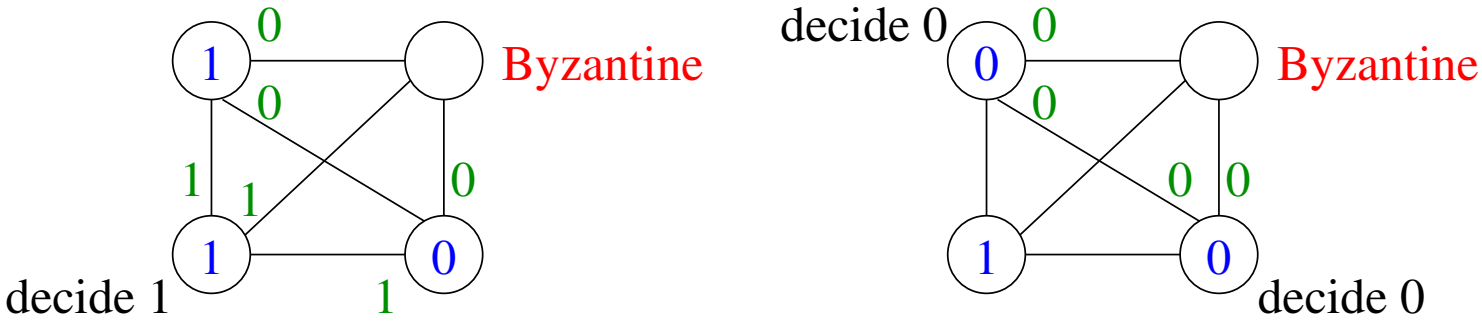
Solution: Each incoming vote is verified using an echo mechanism.

A vote is *accepted* after $> \frac{N+k}{2}$ confirming echos.

Echo mechanism

Complication: A Byzantine process may send different votes to different processes.

Example: Let $N = 4$ and $k = 1$. Each round, a correct process waits for three votes, and needs three b-votes to decide b.



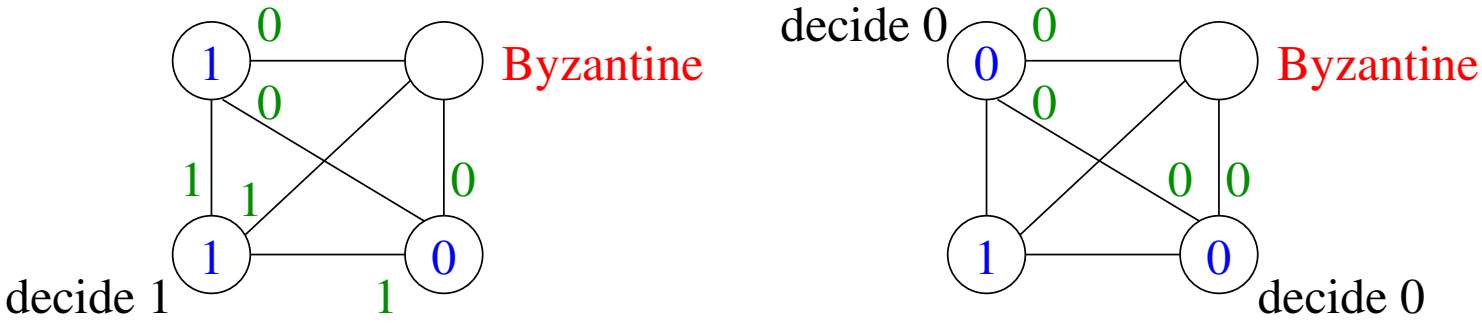
Solution: Each incoming vote is verified using an echo mechanism.

A vote is *accepted* after $> \frac{N+k}{2}$ confirming echos. **Why?**

Echo mechanism

Complication: A Byzantine process may send different votes to different processes.

Example: Let $N = 4$ and $k = 1$. Each round, a correct process waits for three votes, and needs three b-votes to decide b.



Solution: Each incoming vote is verified using an echo mechanism.

A vote is *accepted* after $> \frac{N+k}{2}$ confirming echos. **Why?**

Because if b, b' were confirmed by processes p, p', then >N+k confirmations messages would exist, so >k processes would have confirmed both b and b', but only k among them can be Byzantine...

Bracha-Toueg Byzantine consensus algorithm

Initially, each correct process randomly selects 0 or 1.

In round n, at each correct, undecided p:

- p sends $\langle \textbf{vote}, n, value_p \rangle$ to all processes (including itself).
- If p receives $\langle \textbf{vote}, m, b \rangle$ from q, it sends $\langle \textbf{echo}, q, m, b \rangle$ to all processes (including itself).
- p counts incoming $\langle \textbf{echo}, q, n, b \rangle$ messages for each q, b.

  When $> \frac{N+k}{2}$ such messages arrived, p *accepts* q's b-vote.
- The round is completed when p has accepted $N - k$ votes.

  If most votes are for 0, then $value_p \leftarrow 0$. Else, $value_p \leftarrow 1$.

Bracha-Toueg Byzantine consensus algorithm

Processes dismiss/store messages from earlier/future rounds.

If multiple messages $\langle \textbf{vote}, m, \_ \rangle$ or $\langle \textbf{echo}, q, m, \_ \rangle$ arrive via the same channel, only the first one is taken into account.

If $> \frac{N+k}{2}$ of the *accepted* votes are for b, then p decides b.

When p decides b, it broadcasts $\langle \textbf{decide}, b \rangle$ and terminates.

The other processes interpret $\langle \textbf{decide}, b \rangle$ as a b-vote by p, and a b-echo by p for each q, for all rounds to come.

Bracha-Toueg Byzantine consensus algorithm

Processes dismiss/store messages from earlier/future rounds.

If multiple messages $\langle \textbf{vote}, m, \_ \rangle$ or $\langle \textbf{echo}, q, m, \_ \rangle$ arrive via the same channel, only the first one is taken into account.

If $> \frac{N+k}{2}$ of the *accepted* votes are for b, then p decides b.

When p decides b, it broadcasts $\langle \textbf{decide}, b \rangle$ and terminates.

The other processes interpret $\langle \textbf{decide}, b \rangle$ as a b-vote by p, and a b-echo by p for each q, for all rounds to come.

Questions

If an undecided process receives $\langle \textbf{decide}, b \rangle$, why can it in general not immediately decide b?

What happens if all k Byzantine processes keep silent?

Questions

If an undecided process receives $\langle \textbf{decide}, b \rangle$, why can it in general not immediately decide b?

Answer: The message may originate from a Byzantine process.

What happens if all k Byzantine processes keep silent?

Questions

If an undecided process receives $\langle \textbf{decide}, b \rangle$, why can it in general not immediately decide b?

Answer: The message may originate from a Byzantine process.

What happens if all k Byzantine processes keep silent?

Questions

If an undecided process receives $\langle \textbf{decide}, b \rangle$, why can it in general not immediately decide b?

Answer: The message may originate from a Byzantine process.

What happens if all k Byzantine processes keep silent?

Answer: The $N - k$ correct processes reach consensus in two rounds.

Bracha-Toueg Byzantine consensus alg. - Example

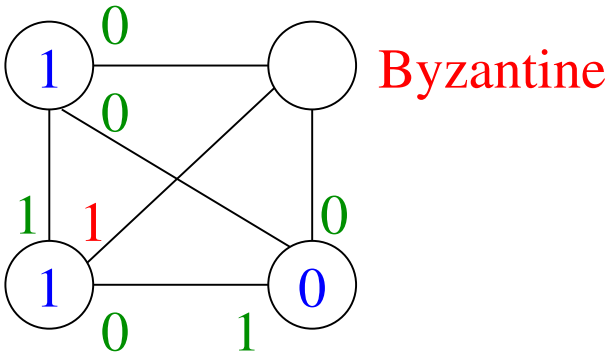
We study the previous example again, now with verification of votes.

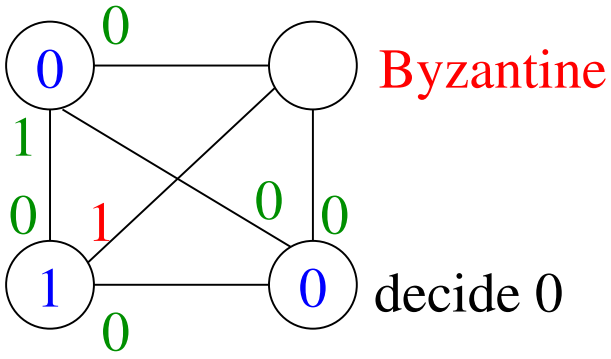
$N = 4$ and $k = 1$, so each round a correct process needs:

- $> \frac{N+k}{2}$, i.e. three, confirmations to accept a vote;
- $N - k$, i.e. three, accepted votes to determine a value; and
- $> \frac{N+k}{2}$, i.e. three, accepted b-votes to decide b.

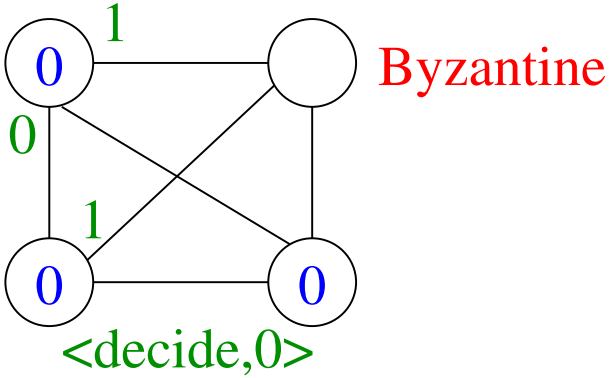
Only relevant **vote** messages are depicted (without their round number).

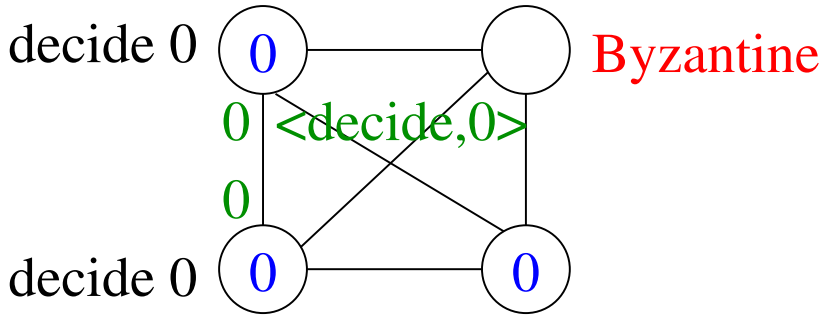
Bracha-Toueg Byzantine consensus alg. - Example





In round zero, the left bottom process doesn't accept vote 1 by the Byzantine process, since none of the other two correct processes confirm this vote. So it waits for (and accepts) vote 0 by the right bottom process, and thus doesn't decide 1 in round zero.





Bracha-Toueg Byzantine consensus alg. - Correctness

Theorem: Let $k < \frac{N}{3}$. The Bracha-Toueg k-Byzantine consensus algorithm is a Las Vegas algorithm that terminates with probability 1.

Proof: Each round, the correct processes eventually accept $N - k$ votes, since there are $\geq N - k$ correct processes. *(Note that* $N - k > \frac{N+k}{2}$*.)*

In round n, let correct processes p and q accept votes for b and b', respectively, from a process r.

Then they received $> \frac{N+k}{2}$ messages $\langle \textbf{echo}, r, n, b \rangle$ resp. $\langle \textbf{echo}, r, n, b' \rangle$.

$> k$ processes, so at least one correct process, sent such messages to both p and q.

So $b = b'$.

Bracha-Toueg Byzantine consensus alg. - Correctness

Theorem: Let $k < \frac{N}{3}$. The Bracha-Toueg k-Byzantine consensus algorithm is a Las Vegas algorithm that terminates with probability 1.

Proof: Each round, the correct processes eventually accept $N - k$ votes, since there are $\geq N - k$ correct processes. *(Note that $N - k > \frac{N+k}{2}$.)*

In round n, let correct processes p and q accept votes for b and b', respectively, from a process r.

Then they received $> \frac{N+k}{2}$ messages $\langle \textbf{echo}, r, n, b \rangle$ resp. $\langle \textbf{echo}, r, n, b' \rangle$.

$> k$ processes, so at least one correct process, sent such messages to both p and q.

So $b = b'$.

Bracha-Toueg Byzantine consensus alg. - Correctness

Theorem: Let $k < \frac{N}{3}$. The Bracha-Toueg k-Byzantine consensus algorithm is a Las Vegas algorithm that terminates with probability 1.

Proof: Each round, the correct processes eventually accept $N - k$ votes, since there are $\geq N - k$ correct processes. *(Note that* $N - k > \frac{N+k}{2}$*.)*

In round n, let correct processes p and q accept votes for b and b', respectively, from a process r.

Then they received $> \frac{N+k}{2}$ messages $\langle \textbf{echo}, r, n, b \rangle$ resp. $\langle \textbf{echo}, r, n, b' \rangle$.

$> k$ processes, so at least one correct process, sent such messages to both p and q.

So $b = b'$.

Bracha-Toueg Byzantine consensus alg. - Correctness

Suppose a correct process decides b in round n.

In this round it accepts $> \frac{N+k}{2}$ b-votes.

So in round n, correct processes accept $> \frac{N+k}{2} - k = \frac{N-k}{2}$ b-votes.

Hence, after round n, $value_q = b$ for each correct q.

So correct processes will vote b in all rounds $m > n$.

Because they will accept $\geq N - 2k > \frac{N-k}{2}$ b-votes.

Bracha-Toueg Byzantine consensus alg. - Correctness

Let S be a set of $N - k$ correct processes.

Assuming fair scheduling, there is a chance $\rho > 0$ that in a round each process in S accepts $N - k$ votes from the processes in S.

With chance ρ^2 this happens in consecutive rounds $n, n + 1$.

After round n, all processes in S have the same value b.

After round $n + 1$, all processes in S have decided b.

Impossibility of $\lceil \frac{N}{3} \rceil$-Byzantine consensus

Theorem: Let $k \geq \frac{N}{3}$. There is no Las Vegas algorithm for k-Byzantine consensus.

Proof: Suppose, toward a contradiction, there is such an algorithm.

Since $k \geq \frac{N}{3}$, we can choose sets S and T of processes with $|S| = |T| = N - k$ and $|S \cap T| \leq k$.

Suppose all processes in S select 0 and all processes in T select 1.

Suppose that messages between processes in $S \setminus T$ and in $T \setminus S$ are very slow.

Suppose all processes that are in $S \cap T$ are Byzantine.

The processes in S can then, with the help of the Byzantine processes, decide 0.

Likewise the processes in T can decide 1.

Note:
Bracha-Toueg crash / byzantine consensus algorithms work on a clique.

Quiz:
Can you find variants of these algorithms that work on arbitrary graphs?
(feel free to strenghten some assumptions...)