# REACHING AGREEMENT IN THE PRESENCE OF FAULTS

M. PEASE, R, SHOSTAK, AND L. LAMPORT

**Ashay Katkar**

**Sameeksha Mahesh**

**Sarvesh Halbe**

**Ritesh Patro**

# ABSTRACT


- Each nonfaulty processor has a private value of the information that must be communicated to each other nonfaulty processor.
- Nonfaulty processors always communicate honestly, whereas faulty processors may lie.

- The problem is to devise an algorithm in which processors communicate their own values and relay values received from others that allows each nonfaulty processor to refer a value for each other processor

# INTERACTIVE CONSISTENCY

- Given m, n >=0, is it possible to devise an algorithm based on an exchange of messages that will allow each nonfaulty processor ‘p’ to compute a vector of values with an element for each of the n processors, such that:
  1. the non-faulty processors compute exactly the same vector;
  2. the element of this vector corresponding to a given non-faulty processor is the private value of the processor.

- The computed vector is called an interactive consistency (i.c) vector.

# SINGLE-FAULT CASE

- Round 1 (direct receives)
- A receives: A:24, B:24, C:24, D:30
- B receives: A:24, B:24, C:24, D:18
- C receives: A:24, B:24, C:24, D:100

# SINGLE-FAULT CASE

- Round 1 (direct receives)
- A receives: A:24, B:24, C:24, D:30
- B receives: A:24, B:24, C:24, D:18
- C receives: A:24, B:24, C:24, D:100

- Each processor forwards the values it heard in Round-1.
- A forwards to everyone: “B told me 24”, “C told me 24”, “D told me 30”
- B forwards to everyone: “A told me 24”, “C told me 24”, “D told me 18”
- C forwards to everyone: “A told me 24”, “B told me 24”, “D told me 100”

# SINGLE-FAULT CASE

A’s reports:

- For A: Private value (24)

# SINGLE-FAULT CASE

A’s reports:

- For A: Private value (24)
- For B: [24 (direct from B), 24 (from C), 34 (from D)] → majority 24

# SINGLE-FAULT CASE

A's reports:

- For A: Private value (24)
- For B: [24 (direct from B), 24 (from C), 34 (from D)] → majority 24
- For C: [24 (direct from C), 24 (from B), 21 (from D)] → majority 24
- For D: [30 (direct from D), 18 (forward from B), 100 (forward from C)] → no majority → NIL

# SINGLE-FAULT CASE

A's reports:

- For A: Private value (24)
- For B: [24 (direct from B), 24 (from C), 34 (from D)] → majority 24
- For C: [24 (direct from C), 24 (from B), 21 (from D)] → majority 24
- For D: [30 (direct from D), 18 (forward from B), 100 (forward from C)] → no majority → NIL

Final IC vectors (all non-faulty record the same vector)

- A final: [A:24, B:24, C:24, D:NIL]
- B final: [A:24, B:24, C:24, D:NIL]
- C final: [A:24, B:24, C:24, D:NIL]
- IC vector of faulty processor not relevant.

# N >= 3M+1

- Message rounds:

In the general case of m faults, m + 1 rounds are required in order to describe the algorithm.

Each extra round gives honest processors a chance to verify and cross-check the information that might have been tampered with by faulty ones.

# N >= 3M+1

- Message rounds:

In the general case of m faults, m + 1 rounds are required in order to describe the algorithm.

Each extra round gives honest processors a chance to verify and cross-check the information that might have been tampered with by faulty ones.

- m = 1, round 1: A → B, C, D

n-1

# N >= 3M+1

- Message rounds:

In the general case of m faults, m + 1 rounds are required in order to describe the algorithm.

Each extra round gives honest processors a chance to verify and cross-check the information that might have been tampered with by faulty ones.

- m = 1, round 1: A → B, C, D (other 3 processors do the same)

(n * n-1)

# N >= 3M+1

- Message rounds:

In the general case of m faults, m + 1 rounds are required in order to describe the algorithm.

Each extra round gives honest processors a chance to verify and cross-check the information that might have been tampered with by faulty ones.

- m = 1, round 1: n * n-1

  round 2: A → forwards to B what it heard from C, D

  → forwards to C what it heard from B, D

  → forwards to D what it heard from B, C

  n-1 * n-2

# N >= 3M+1

- Message rounds:

In the general case of m faults, m + 1 rounds are required in order to describe the algorithm.

Each extra round gives honest processors a chance to verify and cross-check the information that might have been tampered with by faulty ones.

- m = 1, round 1: n * n-1

  round 2: A → forwards to B what it heard from C, D

  → forwards to C what it heard from B, D

  → forwards to D what it heard from B, C

  n-1 * n-2

  (every processor does this)

  n * n-1 * n-2

# N >= 3M+1

- Message rounds:

In the general case of m faults, m + 1 rounds are required in order to describe the algorithm.

Each extra round gives honest processors a chance to verify and cross-check the information that might have been tampered with by faulty ones.

- m = 1, round 1: n * n-1

  round 2: A → forwards to B what it heard from C, D

  → forwards to C what it heard from B, D

  → forwards to D what it heard from B, C

  (every processor does this)

  (n * n-1) + (n * n-1 * n-2) → order of $n^3$

# N >= 3M+1

- Message rounds:

In the general case of m faults, m + 1 rounds are required in order to describe the algorithm.

Each extra round gives honest processors a chance to verify and cross-check the information that might have been tampered with by faulty ones.

- m = 1, round 1: n * n-1
  round 2: (n * n-1) + (n * n-1 * n-2) → order of $n^3$

- m = 2, round 1: n * n-1
- round 2: (n * n-1) + (n * n-1 * n-2)
- round 3: ((n * n-1) + (n * n-1 * n-2)) +(n * n-1 * n-2 * n-3)→ order of $n^4$

# N >= 3M+1

- Message rounds:

In the general case of m faults, m + 1 rounds are required in order to describe the algorithm.

Each extra round gives honest processors a chance to verify and cross-check the information that might have been tampered with by faulty ones.

- m = 1, round 1:  n * n-1
  round 2: (n * n-1) + (n * n-1 * n-2) → order of $n^3$

- m = 2, round 1: n * n-1
- round 2: (n * n-1) + (n * n-1 * n-2)
- round 3: ((n * n-1) + (n * n-1 * n-2)) +(n * n-1 * n-2 * n-3)→ order of $n^4$

$$M_{IC}(n, m) = \Theta(n^{m+2})$$

# N >= 3M+1

- Message rounds:

In the general case of m faults, m + 1 rounds are required in order to describe the algorithm.

Each extra round gives honest processors a chance to verify and cross-check the information that might have been tampered with by faulty ones.

- Number of processors:

For consensus to hold, you need more than twice as many honest nodes as faulty ones.

Total = honest + faulty = (2m+1) + m = 3m + 1

(smallest number of honest processors that can still form a majority against any set of m liars)

# N >= 3M+1

Threshold:  Q >(n+m)/2

- p and p' are both nonfaulty processors

# N >= 3M+1

Threshold:  Q >(n+m)/2

- p and p' are both nonfaulty processors

p found set with more than (n+m)/2 members, all agreeing on some value for processor q

Vector p → {_ _ _ q( v ) _ _ _}

Voted by greater than (n+m)/2 other processors.

# N >= 3M+1

Threshold: Q >(n+m)/2

- p' found set with more than (n+m)/2 members, all agreeing on some value for processor q

Vector p' → {_ _ _ q( v' ) _ _ _}

Voted by greater than (n+m)/2 other processors.

Vector p → {_ _ _ q( v ) _ _ _}

Vector p' → {_ _ _ q( v' ) _ _ _}

# N >= 3M+1

Threshold: Q >(n+m)/2

- p' found set with more than (n+m)/2 members, all agreeing on some value for processor q

Vector p' → {_ _ _ q( v' ) _ _ _}

Voted by greater than (n+m)/2 other processors.

Vector p → {_ _ _ q( v ) _ _ _}

Vector p' → {_ _ _ q( v' ) _ _ _}

(n+m)/2 + (n+m)/2 = n+m

# N >= 3M+1

Threshold: Q >(n+m)/2

- p’ found set with more than (n+m)/2 members, all agreeing on some value for processor q

Vector p’ → {_ _ _ q( v’ ) _ _ _}

Voted by greater than (n+m)/2 other processors.

Vector p → {_ _ _ q( v ) _ _ _}

Vector p’ → {_ _ _ q( v’ ) _ _ _}

(n+m)/2 + (n+m)/2 = n+m (only n processors!)

# N >= 3M+1

Threshold: Q >(n+m)/2

q (v) + q(v')

> (n+m)/2 + > (n+m)/2

> (n+m)

Remove n processor to find overlapping processors.

> m

- Number of faulty procesors is 'm'.

# N >= 3M+1

Threshold: Q >(n+m)/2

q (v) + q(v')

> (n+m)/2 + > (n+m)/2

> (n+m)

Remove n processor to find overlapping processors.

> m

- Number of faulty procesors is 'm'.
- Therefore, at least one processor in the overlap must be nonfaulty.

# N >= 3M+1

Threshold: Q >(n+m)/2

q (v) + q(v')

> (n+m)/2 + > (n+m)/2

> (n+m)

Remove n processor to find overlapping processors.

> m

- Number of faulty procesors is 'm'.
- Therefore, at least one processor in the overlap must be nonfaulty.
- The two sets must give rise to the same value v.

# N < 3M+1 (IMPOSSIBILITY)


- Group C sends private (two-party) messages to each group.
  To Group A, it claims its value is “v”.
  To Group B, it claims its value is “w”.

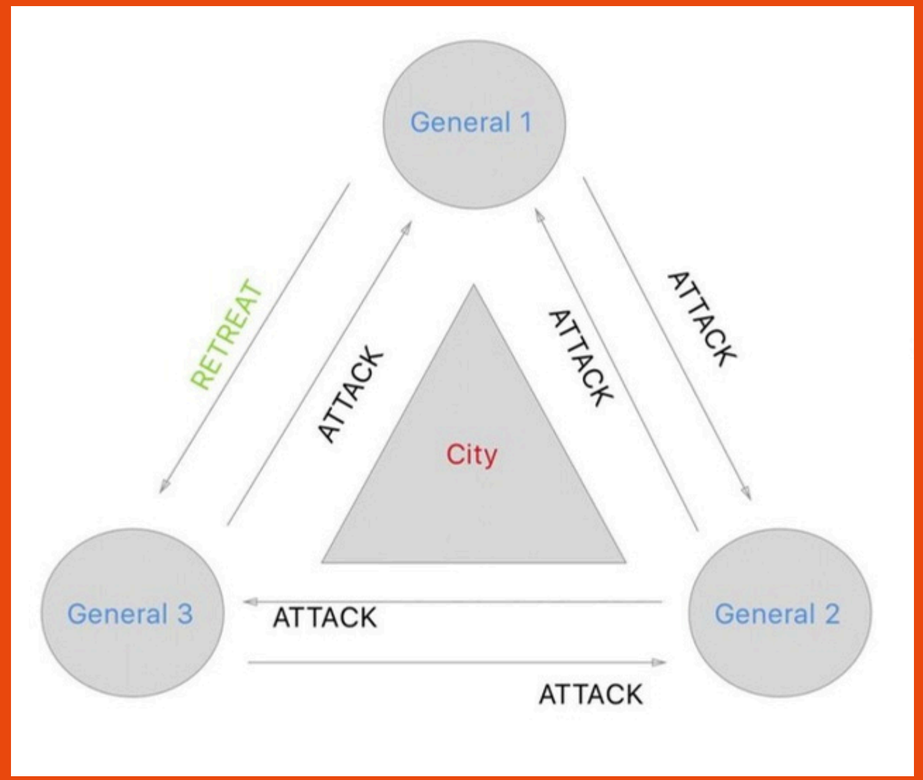
Since A and B have conflicting entries for C’s value (v vs. w),
there is no common majority among the nonfaulty processors.

# THE BYZANTINE GENERALS PROBLEM

LESLIE LAMPORT, ROBERT SHOSTAK, AND MARSHALL PEASE

# INTRODUCTION

- All loyal generals decide upon the same plan of action.
- A small number of traitors cannot cause the loyal generals to adopt a bad plan.



# INTRODUCTION

- Each general sends their value v(i) using a Byzantine solution.
- Other generals act as lieutenants, receiving and forwarding messages.

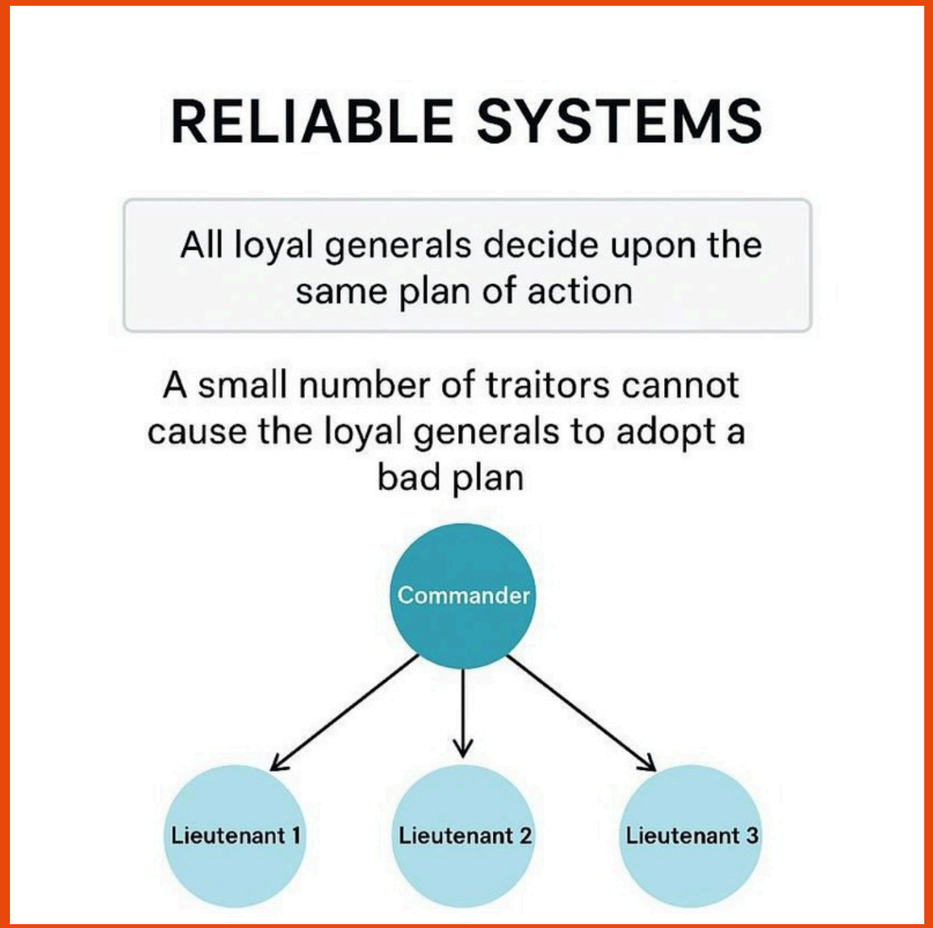
- v(i) be the information communicated by the ith general.
    a. Every loyal general must obtain the same information v(1)...v(n).
    b. If the ith general is loyal, then the value that he sends must be used by every loyal general as the value of v(i).

# INTRODUCTION

- A commanding general sends an order to n-1 lieutenants.

- Interactive Consistency Conditions:

1. IC1 → All loyal lieutenants obey the same order.
2. IC2 → If the commander is loyal, all loyal lieutenants obey his order.

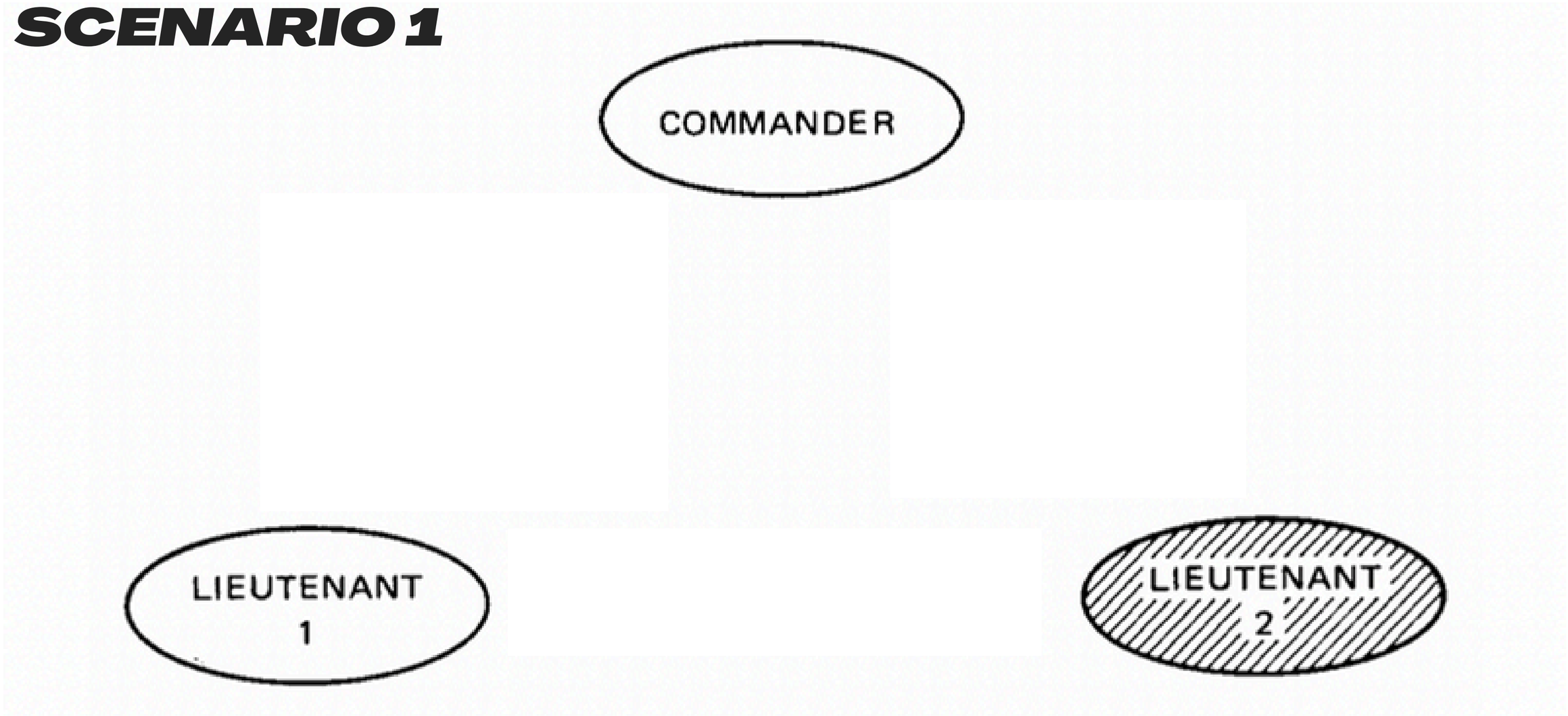


# IMPOSSIBILITY RESULTS

- Prove that any problem using oral messages with fewer than 3m+1 generals cannot handle m traitors
- First prove with only three generals that single traitor makes problem unsolvable, then extrapolate

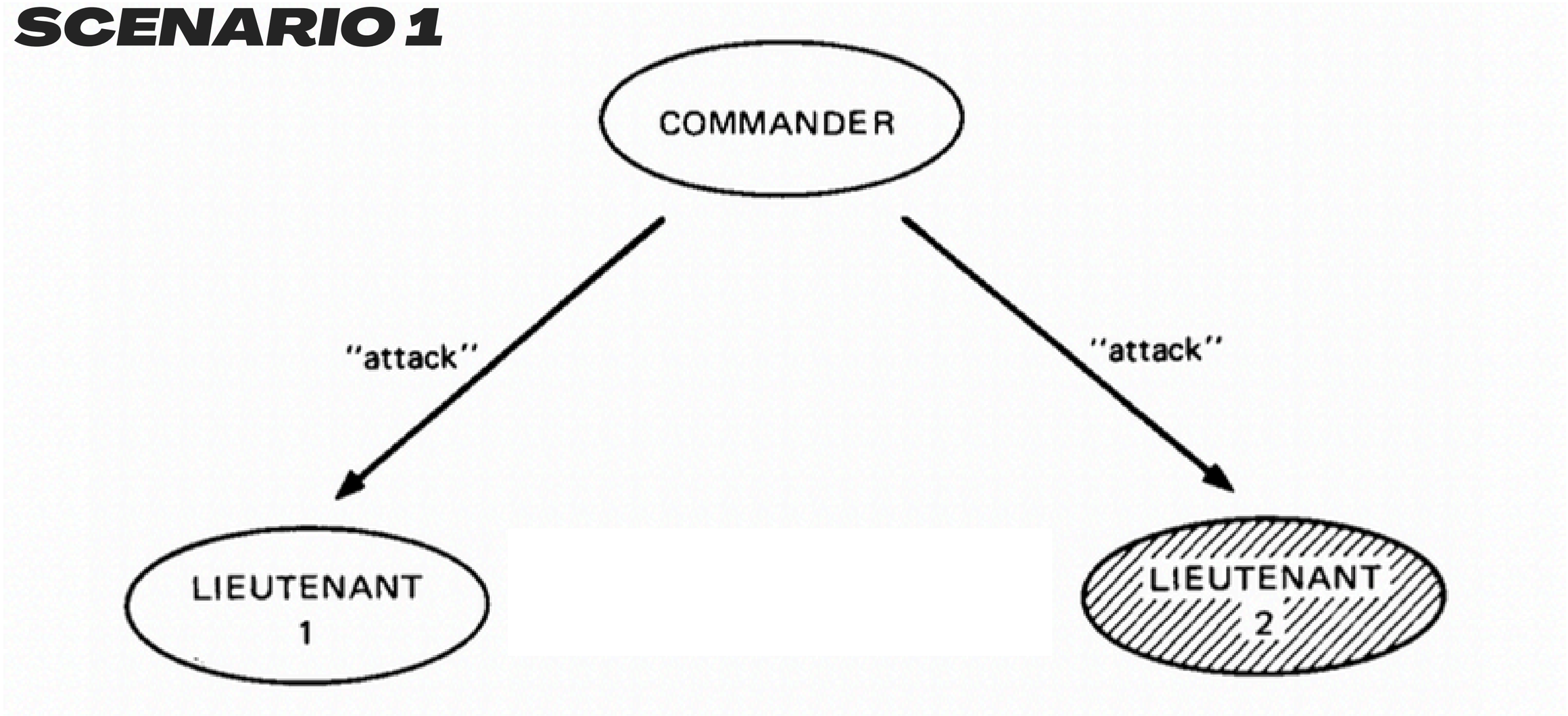
# IMPOSSIBILITY RESULTS

## *SCENARIO 1*



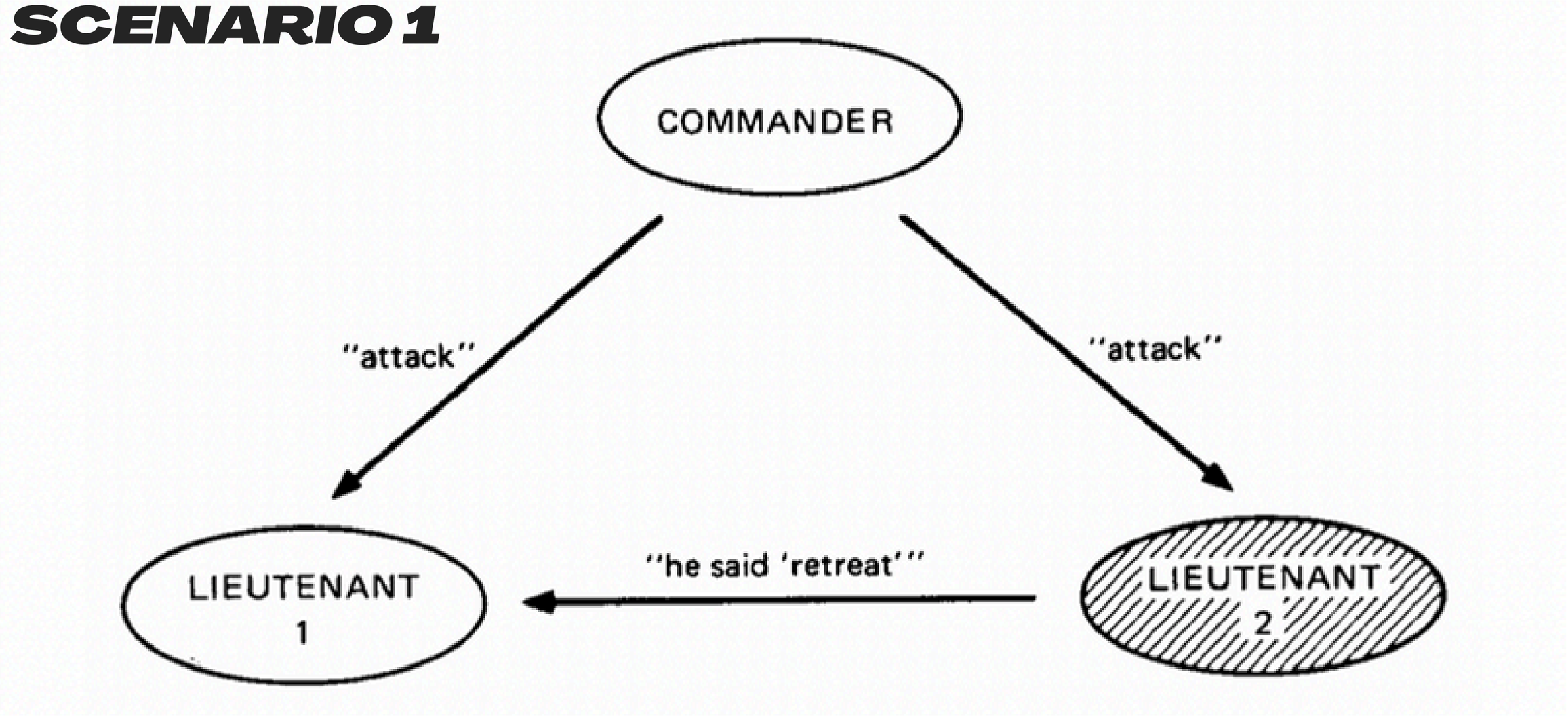
# IMPOSSIBILITY RESULTS

## *SCENARIO 1*



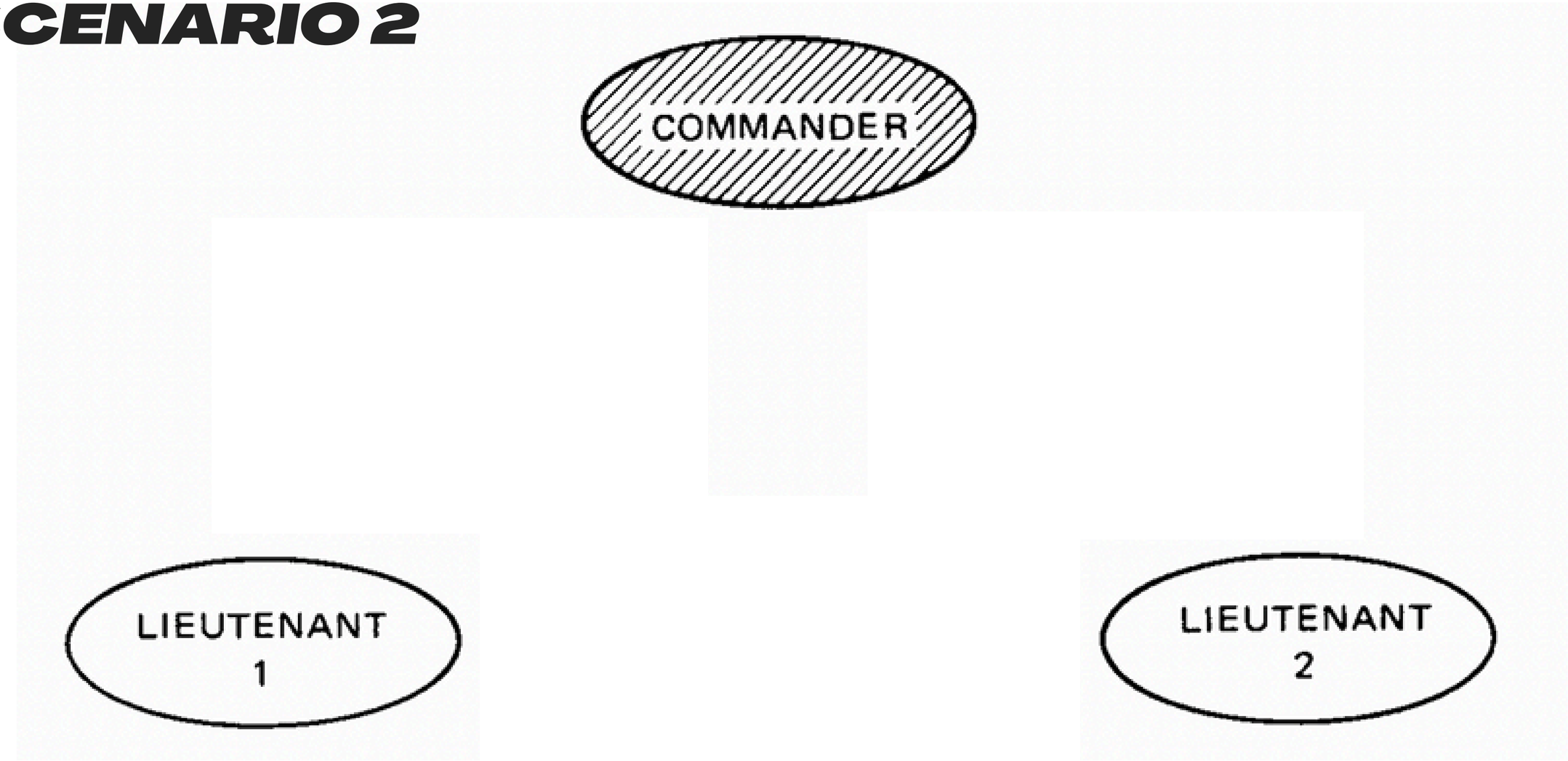
# IMPOSSIBILITY RESULTS

## SCENARIO 1



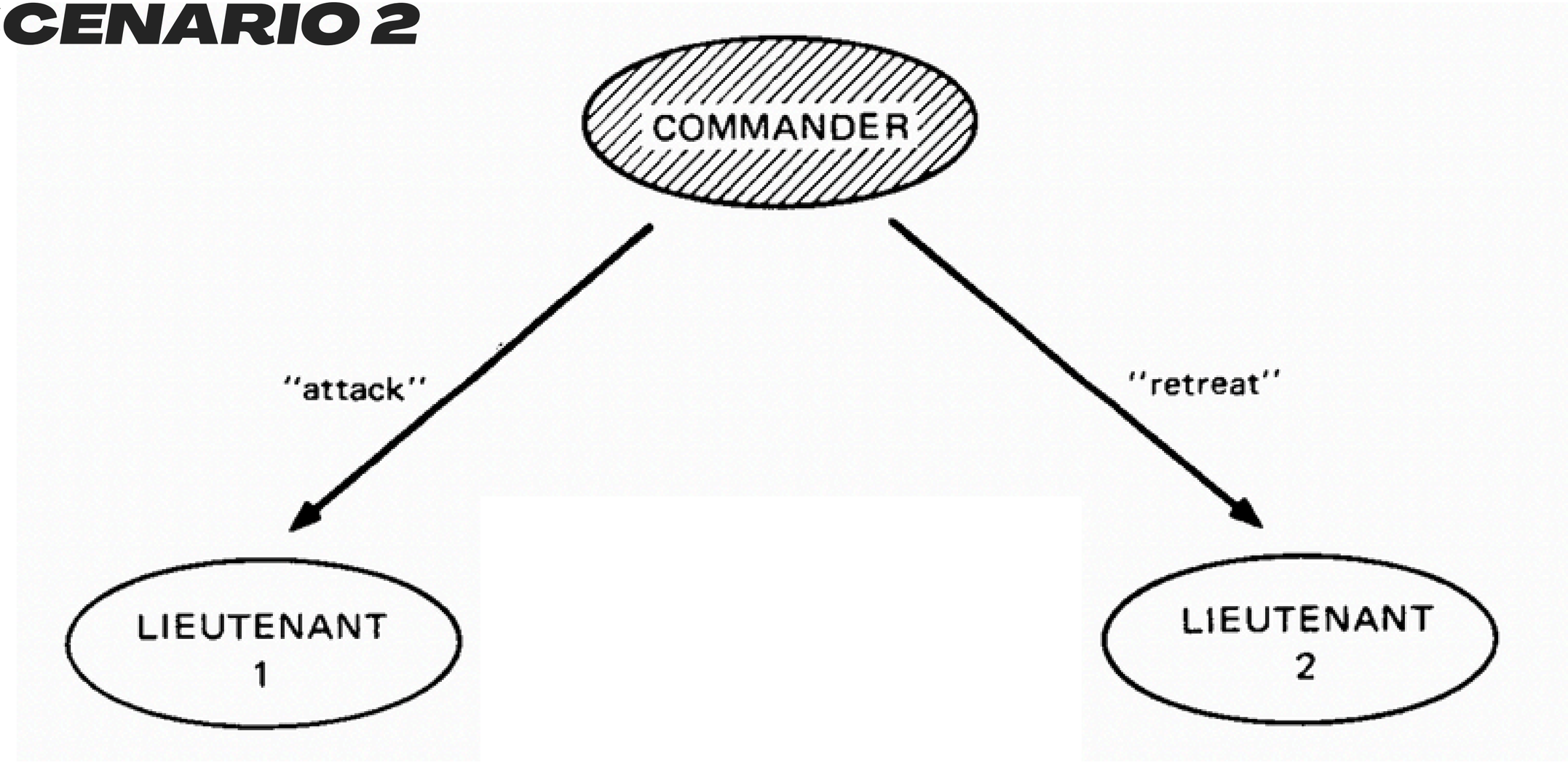
# IMPOSSIBILITY RESULTS

## SCENARIO 2



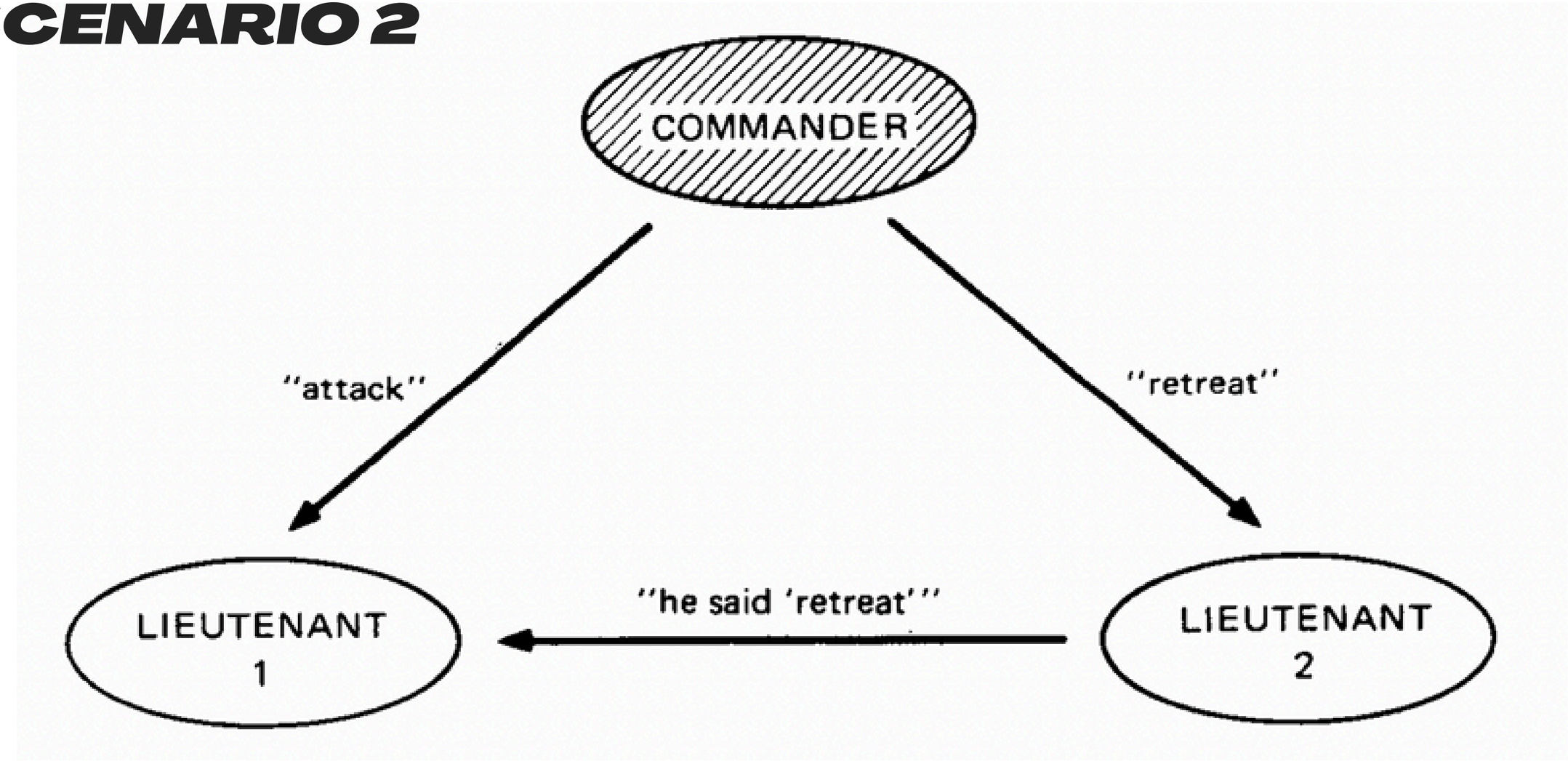
# IMPOSSIBILITY RESULTS

## SCENARIO 2



# IMPOSSIBILITY RESULTS

## SCENARIO 2



# SOLUTION WITH ORAL MESSAGES

## *MESSAGE SYSTEM 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.

Majority function assumptions:
Majority value among inputs if it exists(takes median value if from ordered set), otherwise the value RETREAT

# SOLUTION WITH ORAL MESSAGES

## *ALGORITHM OM(m)*

m = 0:

1. The commander sends his value to every lieutenant.
2. Each lieutenant uses the value he receives from the commander, or uses the value RETREAT if he receives no value.

# SOLUTION WITH ORAL MESSAGES

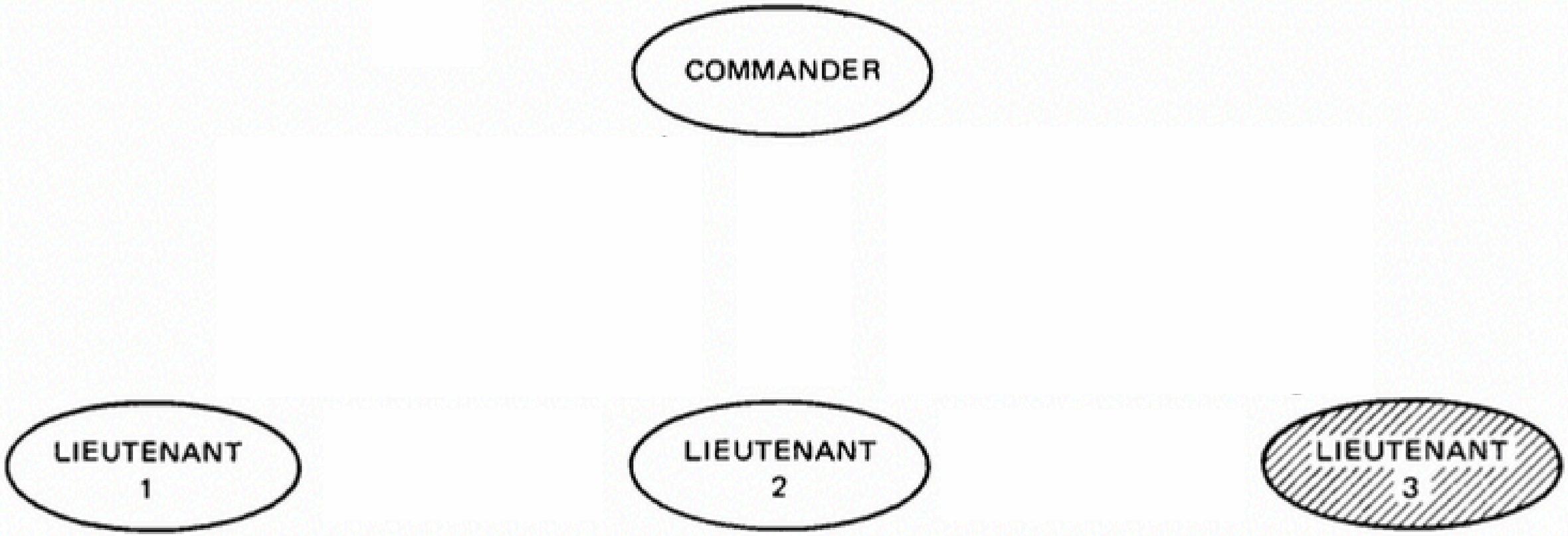
## *ALGORITHM OM(m)*

m > 0:

1. The commander sends his value to every lieutenant.
2. For each i, let $v_i$ be the value Lieutenant i receives from the commander, or else be RETREAT if he receives no value. Lieutenant i acts as the commander in Algorithm OM(m - 1) to send the value $v_i$ to each of the n - 2 other lieutenants.
3. For each i, and each j ≠ i, let $v_j$ be the value Lieutenant i received from Lieutenant j in step (2) (using Algorithm OM(m - 1)), or else RETREAT if he received no such value. Lieutenant i uses the value majority (vl ..... v,-1 ).

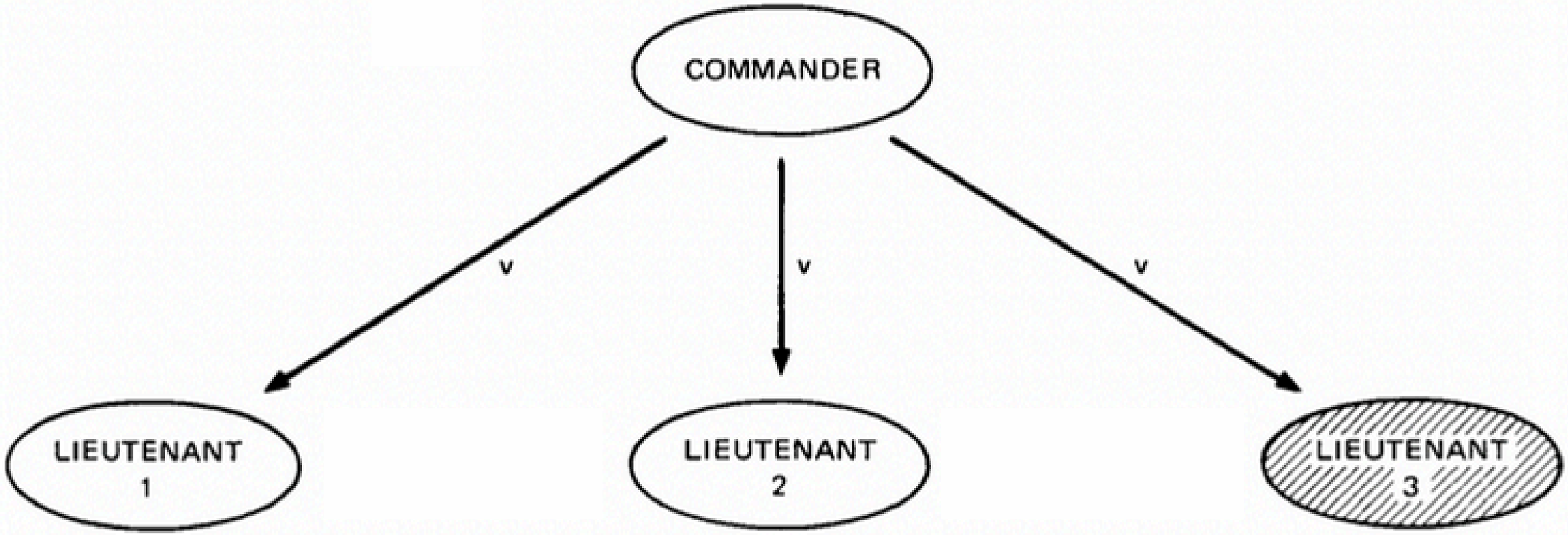
# SOLUTION WITH ORAL MESSAGES

## *SCENARIO 1*



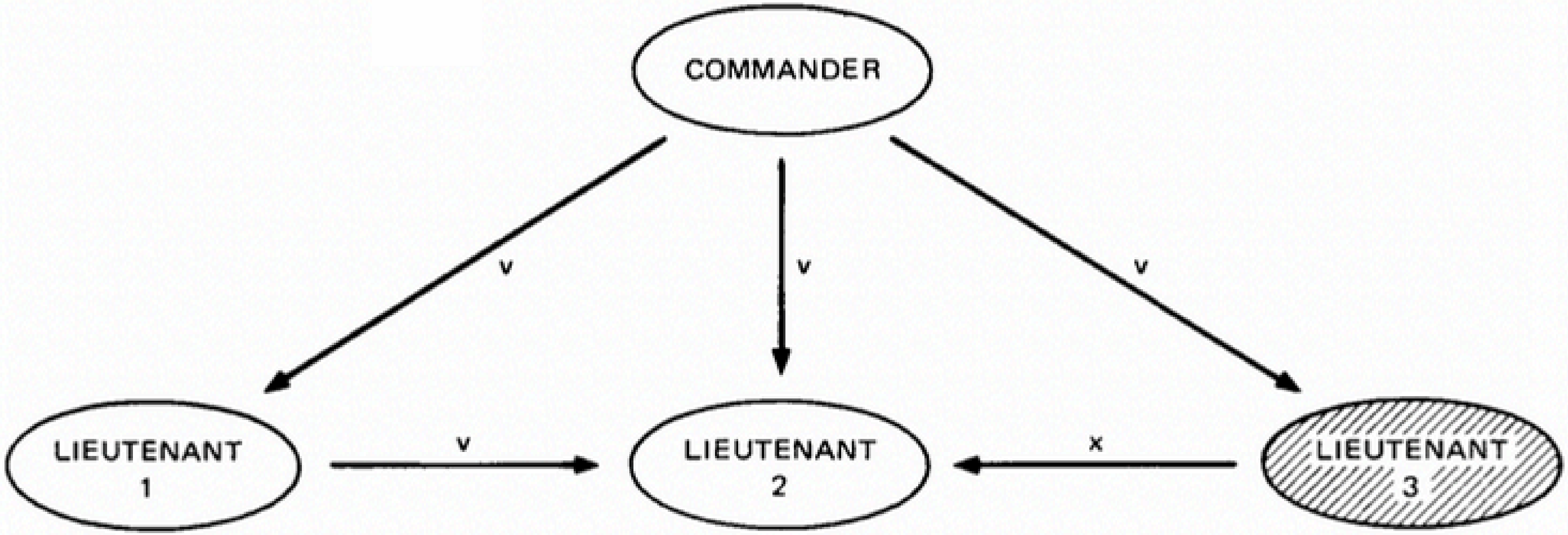
# SOLUTION WITH ORAL MESSAGES

## *SCENARIO 1*



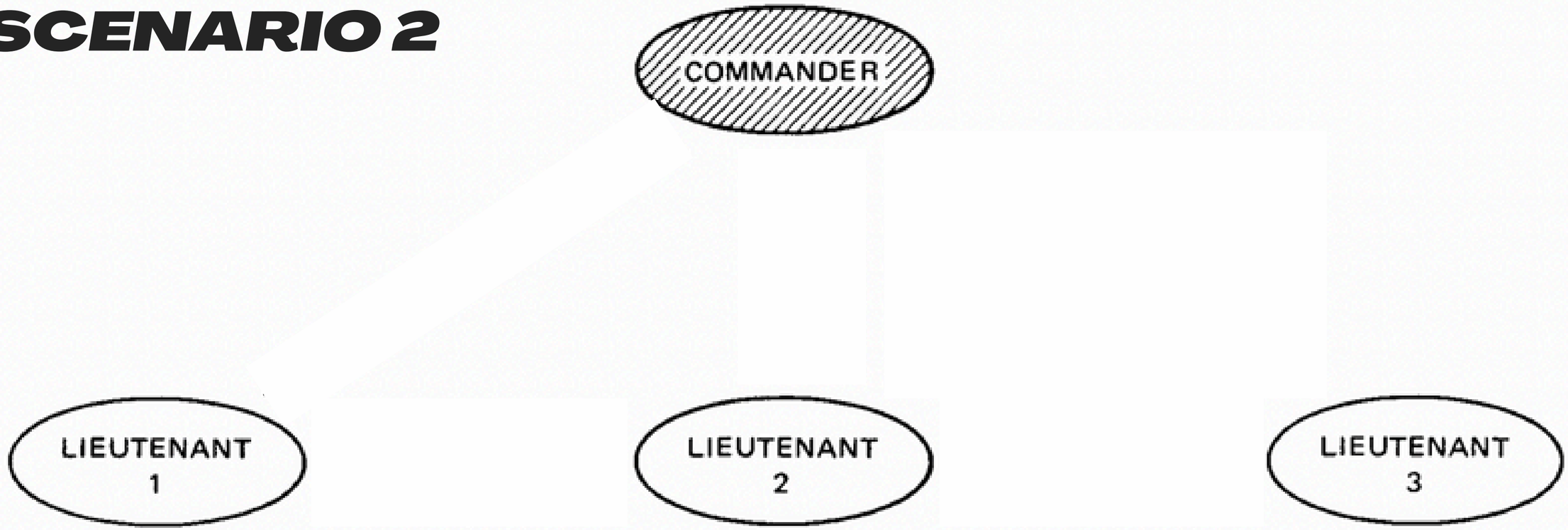
# SOLUTION WITH ORAL MESSAGES

## *SCENARIO 1*



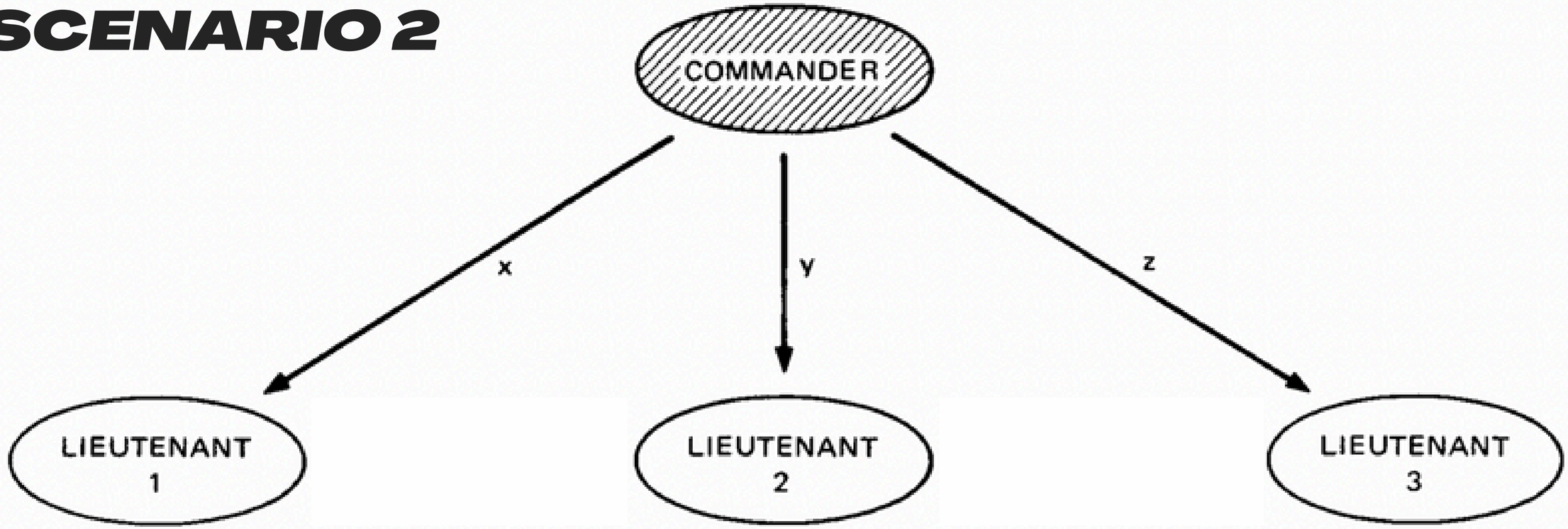
# SOLUTION WITH ORAL MESSAGES

## *SCENARIO 2*



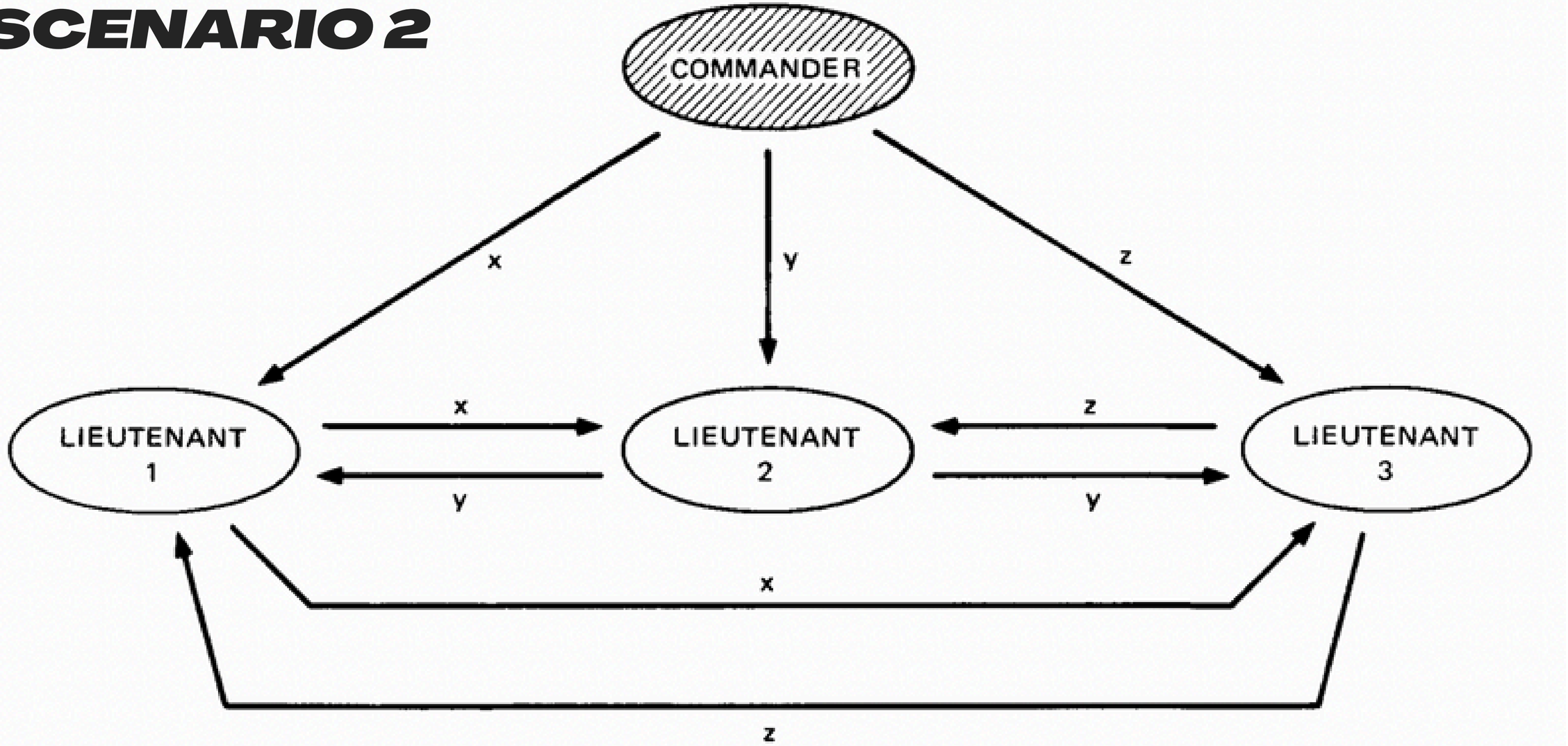
# SOLUTION WITH ORAL MESSAGES

## *SCENARIO 2*



# SOLUTION WITH ORAL MESSAGES

## SCENARIO 2



# SOLUTION WITH ORAL MESSAGES

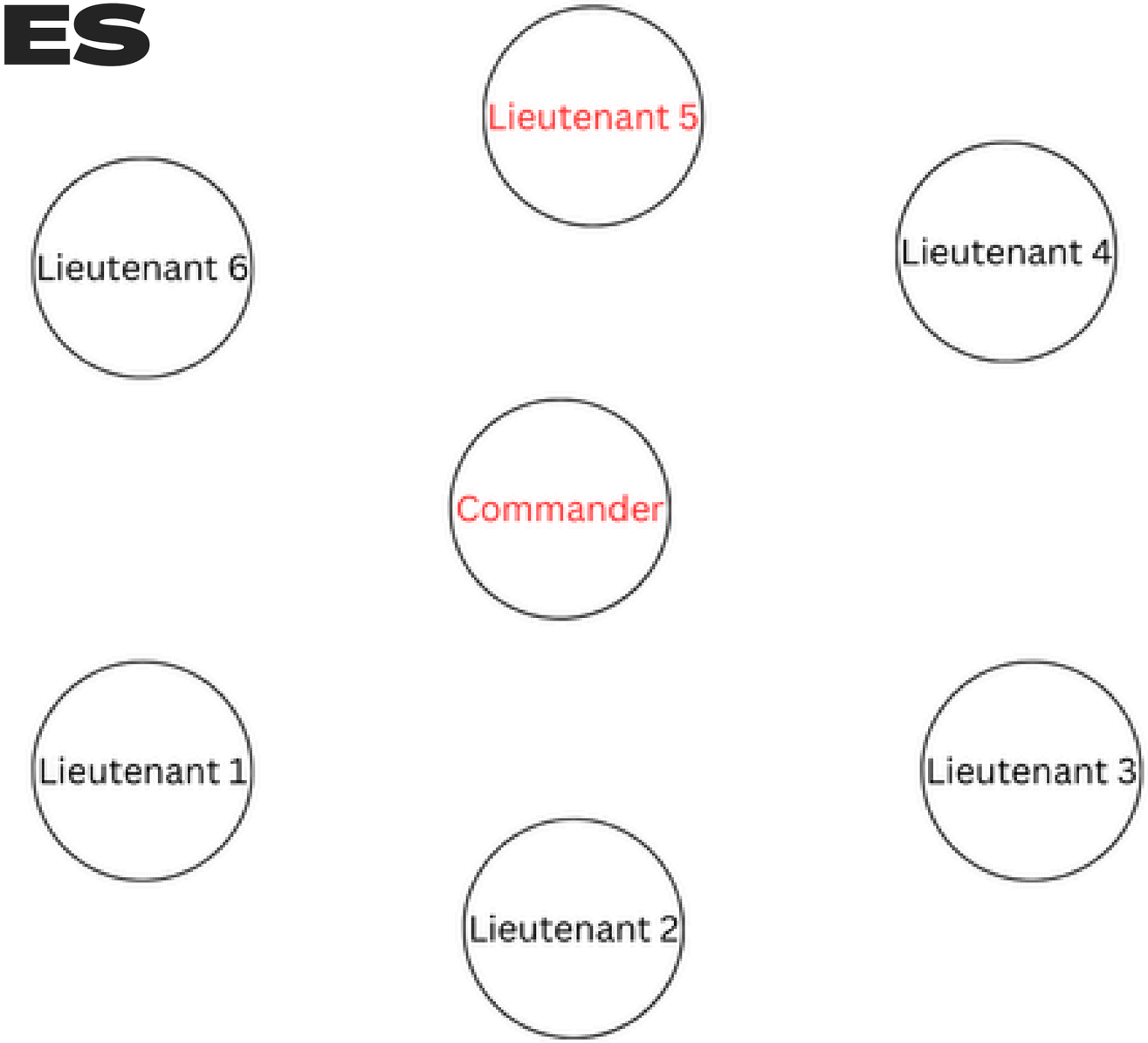
## THEOREM 1

For any m, Algorithm OM(m) satisfies conditions IC1 and IC2 if more than 3m generals and at most m traitors.

Proof by induction:

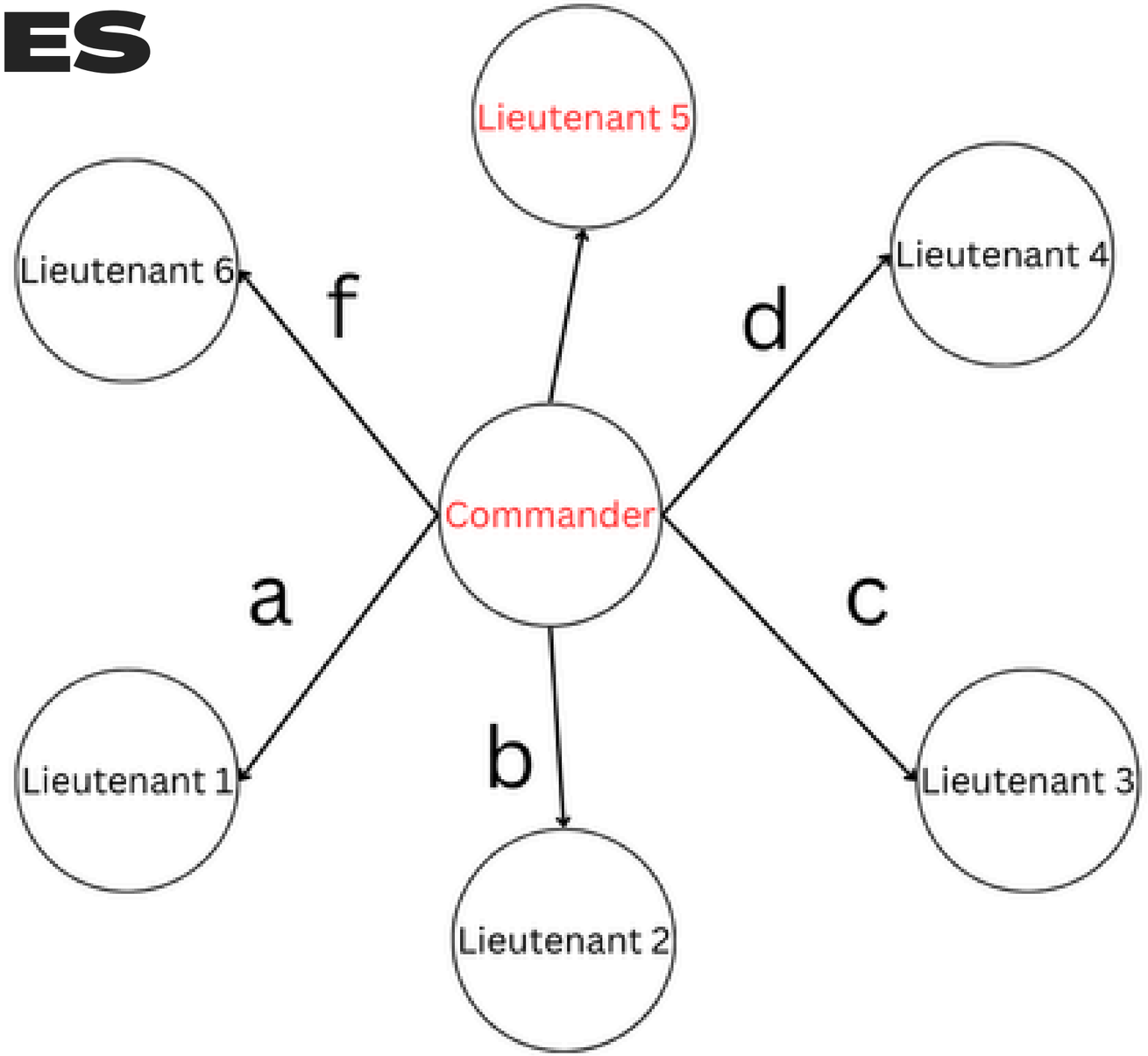
- OM(0) obviously satisfies IC1 and IC2
- OM(m) satisfies IC2(majority of lieutenants will be loyal every level down until m=0)
- IC1 obviously follows IC2 if commander is loyal, so just need to prove IC1 if commander isn't loyal
- Using induction, can assume OM(m-1) true, meaning every loyal lieutenant gets same vector of values, proving IC1

# SOLUTION WITH ORAL MESSAGES



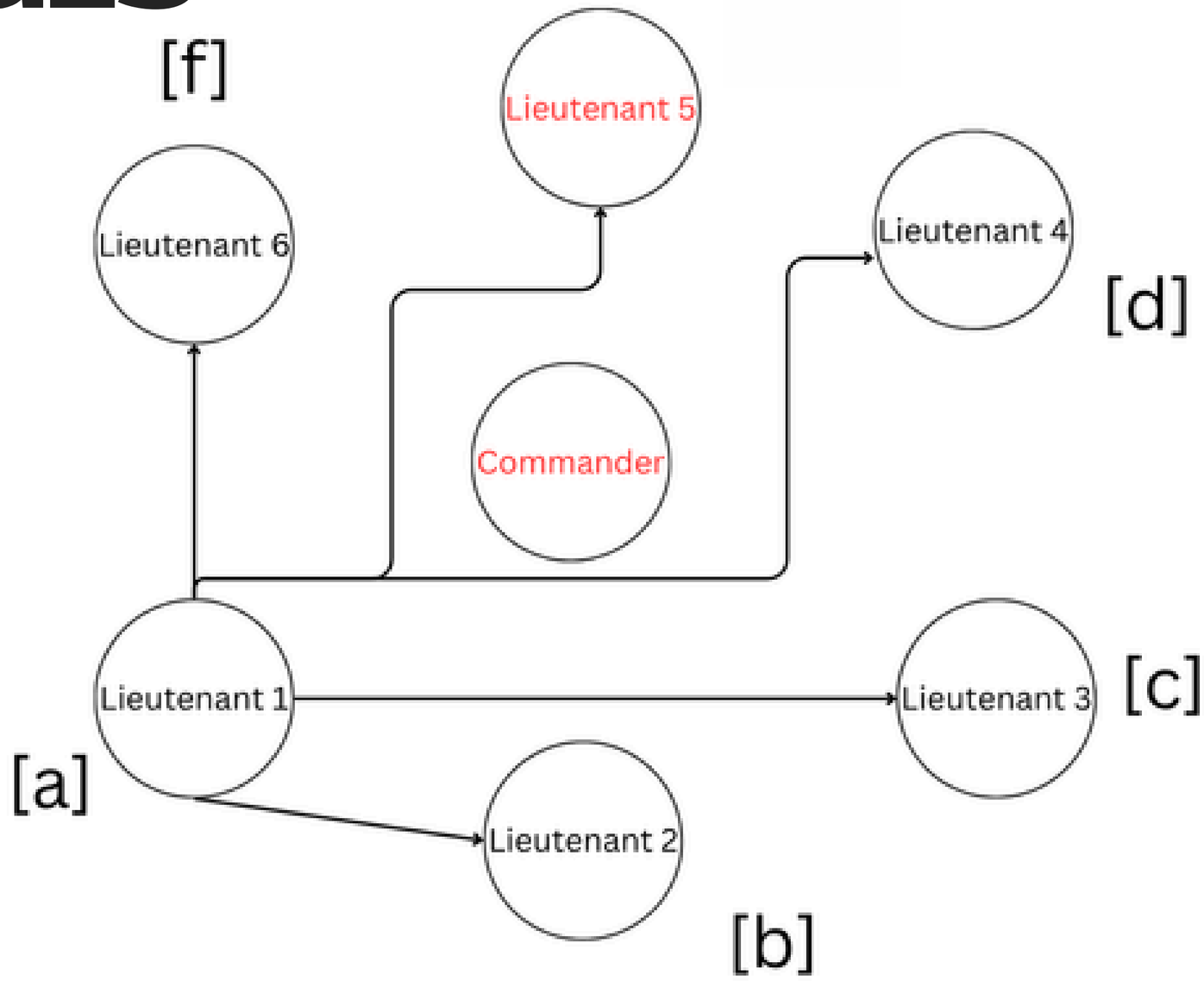
# SOLUTION WITH ORAL MESSAGES

## OM(2)



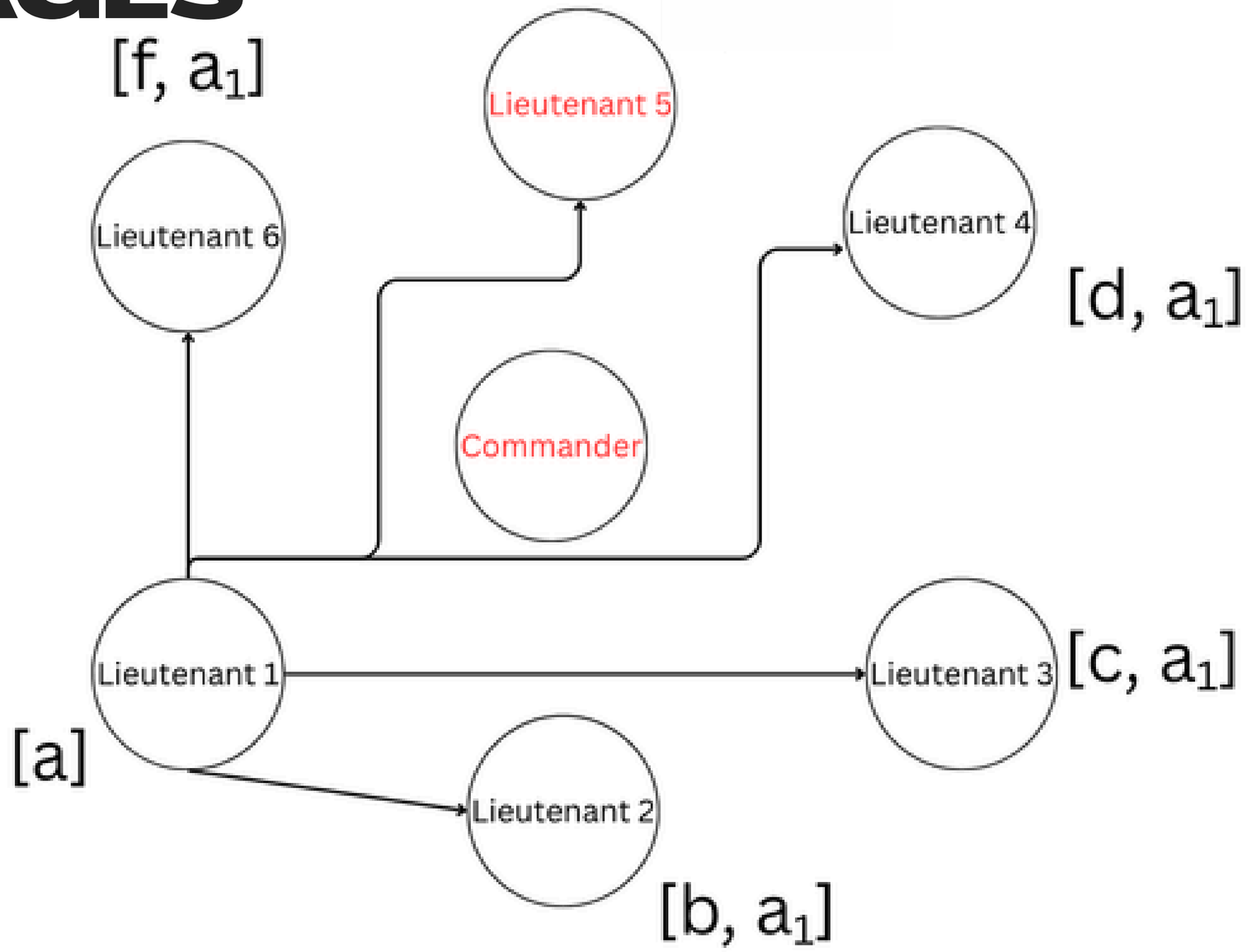
# SOLUTION WITH ORAL MESSAGES

## OM(1)



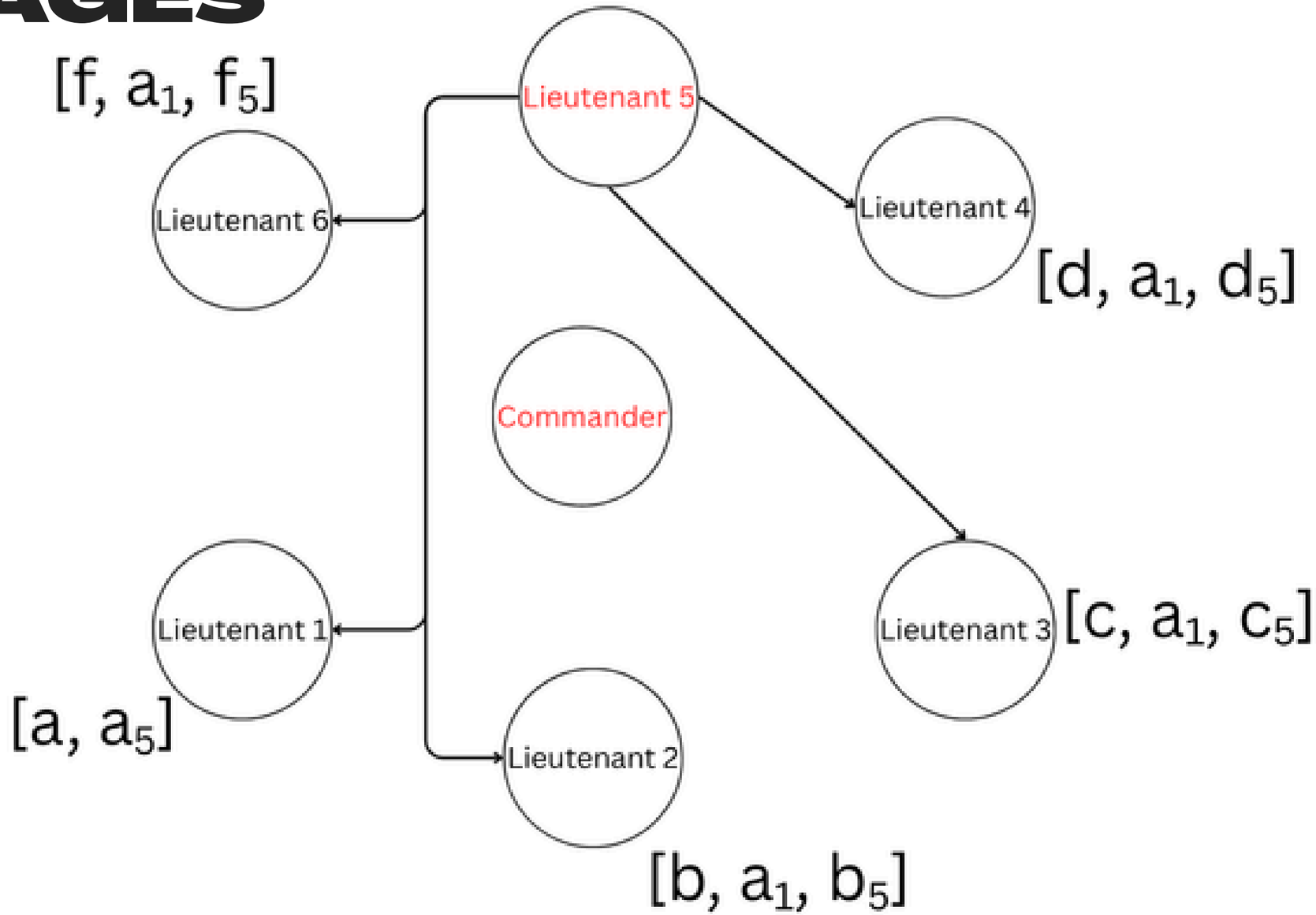
# SOLUTION WITH ORAL MESSAGES

## OM(1)



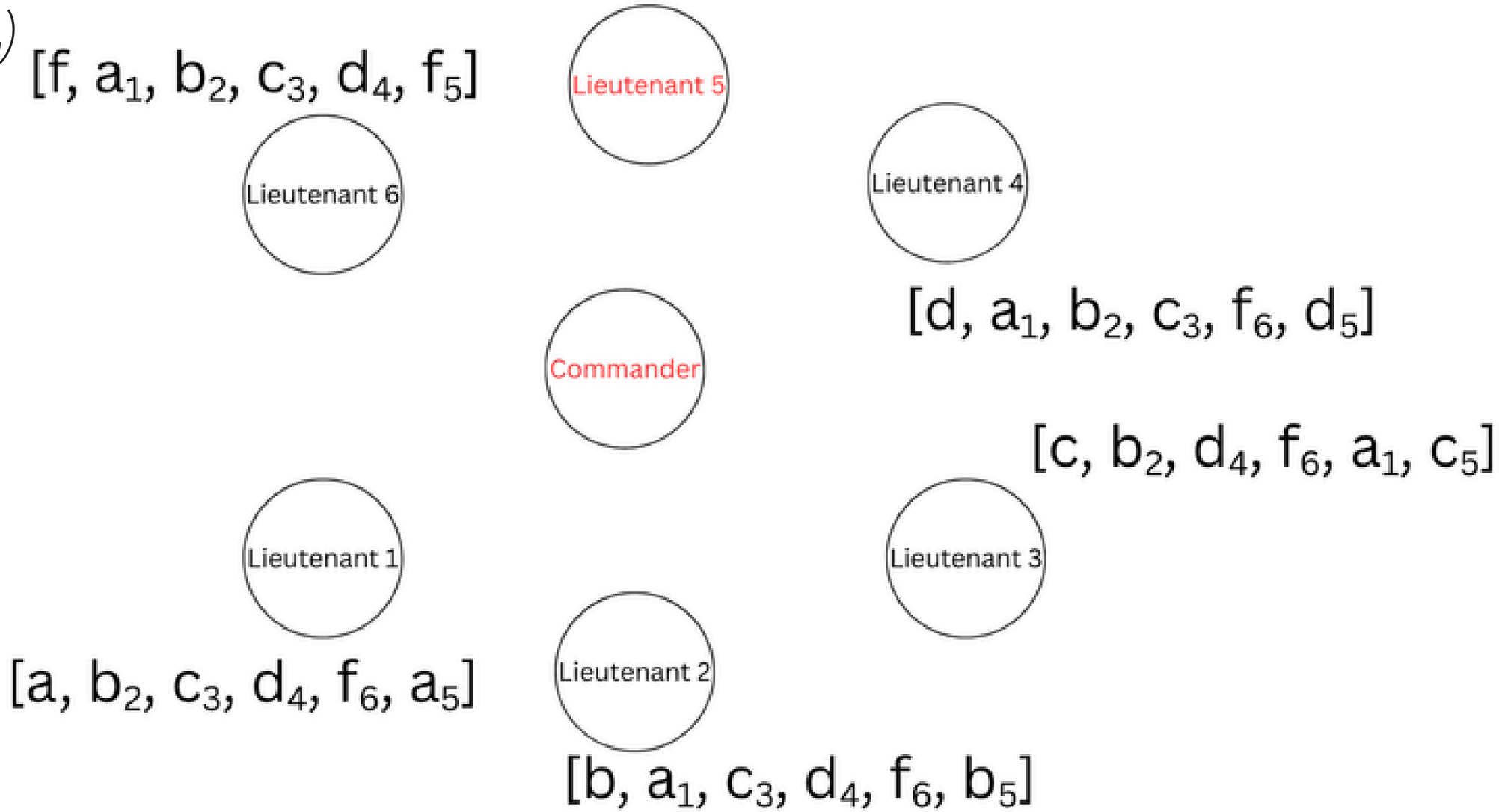
# SOLUTION WITH ORAL MESSAGES

## OM(1)



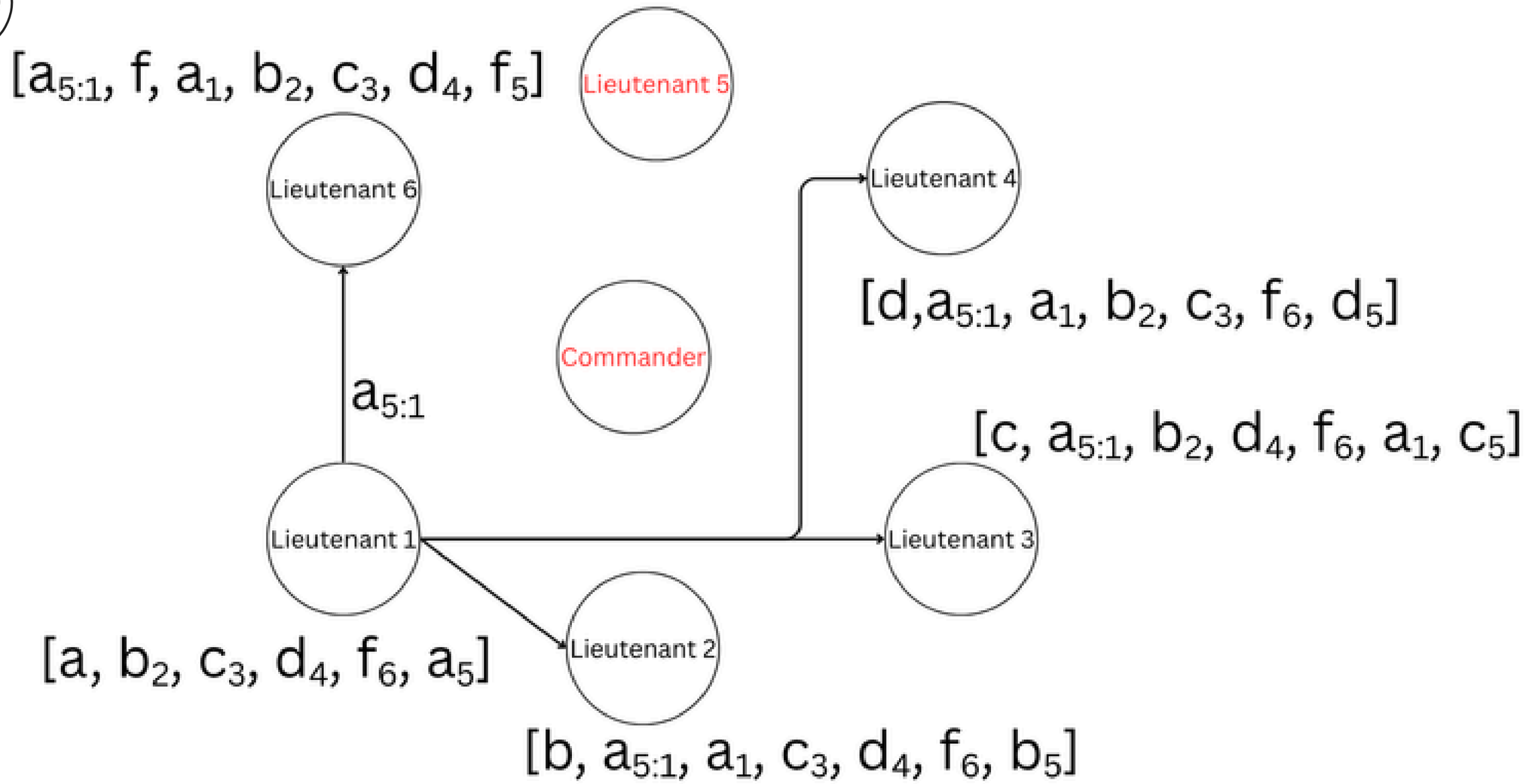
# SOLUTION WITH ORAL MESSAGES

## OM(1)



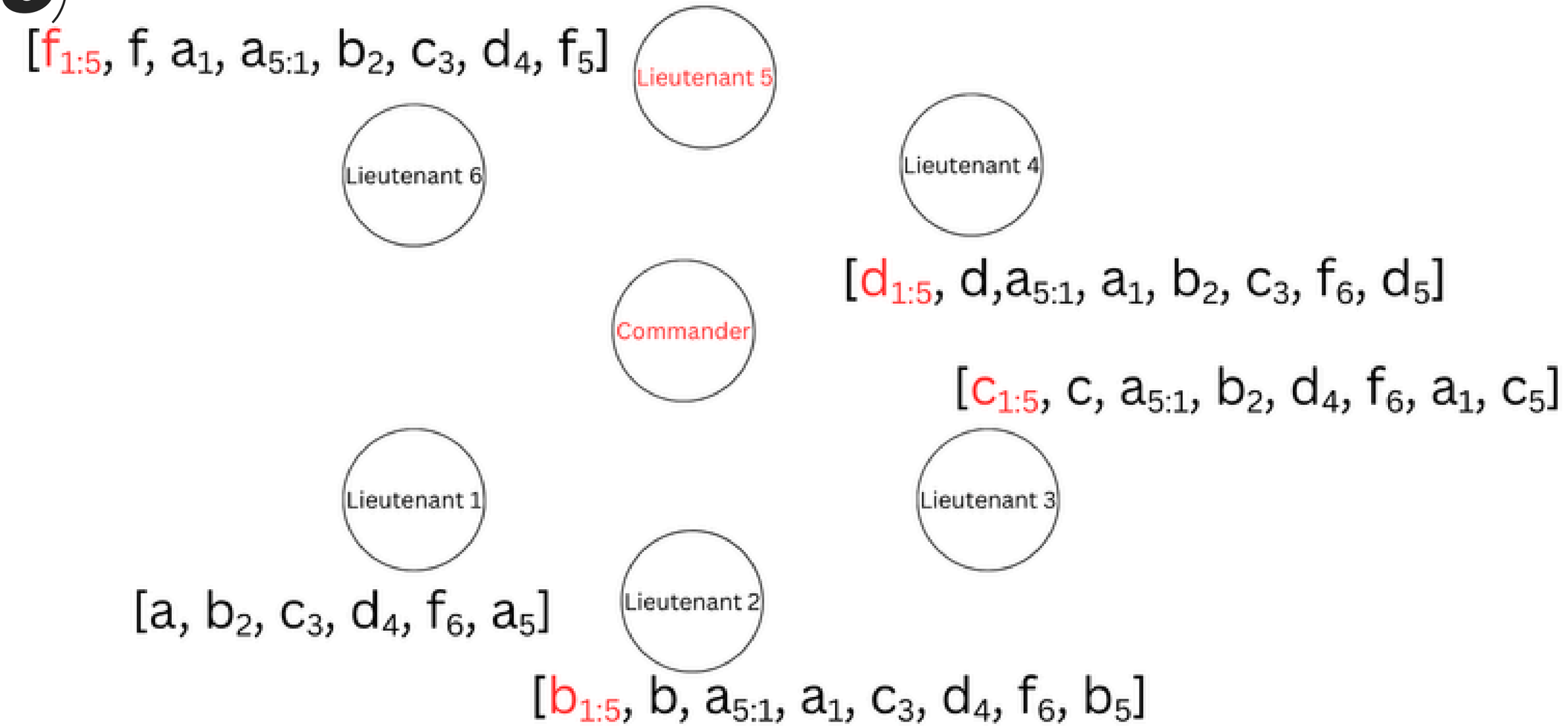
# SOLUTION WITH ORAL MESSAGES

## OM(0)



# SOLUTION WITH ORAL MESSAGES

## OM(0)



# SOLUTION WITH ORAL MESSAGES

## OM(0)

| Lieutenant 1 | Lieutenant 2 | Lieutenant 3 | Lieutenant 4 | Lieutenant 6 |
|---|---|---|---|---|
| 5a, 5b, 5c, 5d, 5f | 5a, 5b, 5c, 5d, 5f | 5a, 5b, 5c, 5d, 5f | 5a, 5b, 5c, 5d, 5f | 5a, 5b, 5c, 5d, 5f |

# SOLUTION WITH ORAL MESSAGES

## *PROBLEM RECAP: OM(M) LIMITATIONS*

- OM(m) works only if n ≥ 3m + 1 (needs many generals for few traitors).
- Traitors can send conflicting orders to different lieutenants.
- No authentication → messages can be forged or modified.
- Works only under strong assumptions like complete connectivity (everyone talks to everyone).

# SOLUTION WITH SIGNED MESSAGES

## *MAIN IDEA:*

*"The traitors' ability to lie makes the Byzantine Problem difficult"*

To ease the problem, the lying ability of traitors need to be restricted.

**A4: Assumptions of Unforgeable Signatures:**
(a) A loyal general's signature cannot be forged and message tampering is detectable.
(b) Anyone can verify the authenticity of a signature.

Traitors' signatures can be forged by other traitors → allows collusion.

# SOLUTION WITH SIGNED MESSAGES

## KEY CONCEPTS

What Each Lieutenant Maintains

| Concept | Meaning |
|---|---|
| **x : i**<br>**Attack:0:1** | **“x” signed by General i**<br>0 → 1 → 2 |
| **$V_i$** | **Set of properly signed orders received so far bu Lieutenant i** |
| **Choice ($V_i$)** | **decision function applied by lieutenant i, used to decide the final action.** |

# SOLUTION WITH SIGNED MESSAGES

## RULES OF HANDLING MESSAGES:

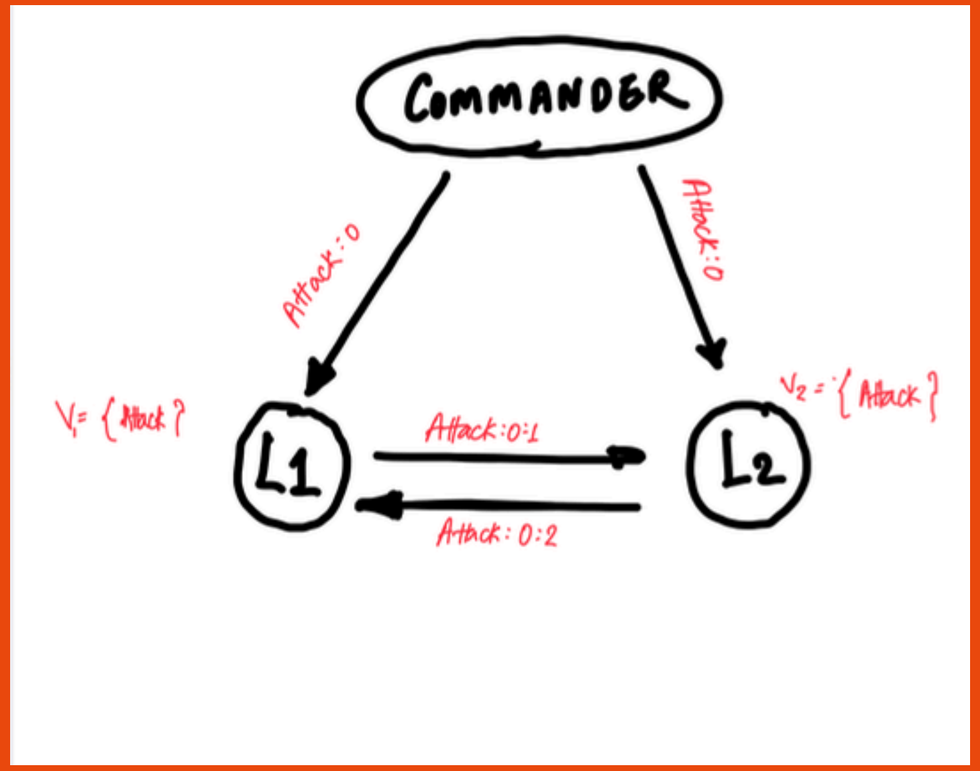
| Case | Example Message | What Happens | Explanation |
|---|---|---|---|
| ✅ Keep unique, valid messages | Attack:0:1 | ✅ Keep | Message has valid signatures and hasn't been seen before. |
| ❌ Duplicate message | Two copies of Attack:0:1 | ❌ Ignore | Already received the same signed message; no need to store twice. |
| ❌ Invalid signature | Attack:1 (forged or tampered) | ❌ Discard | Signature check fails → message not trustworthy. |
| ❌ Same order already known | Retreat:0 already in $V_3$, then Retreat:0:2 arrives | ❌ Ignore | Order "Retreat" already exists; no need to store forwarded copies. |
| ✅ Enough signatures → stop forwarding | Attack:0:2 in SM(1) | ✅ Stop sending further | Message already verified by commander + one lieutenant (m+1 signatures reached). |

- **Ignore duplicates or improperly signed messages:**
- **Ignore any message where the order already exists in $V_i$**
- **If an order is signed by m lieutenants (SM(m)), stop forwarding (enough verification)**

# SOLUTION WITH SIGNED MESSAGES

## *LIEUTENANTS IN ACTION IN SM(M)*

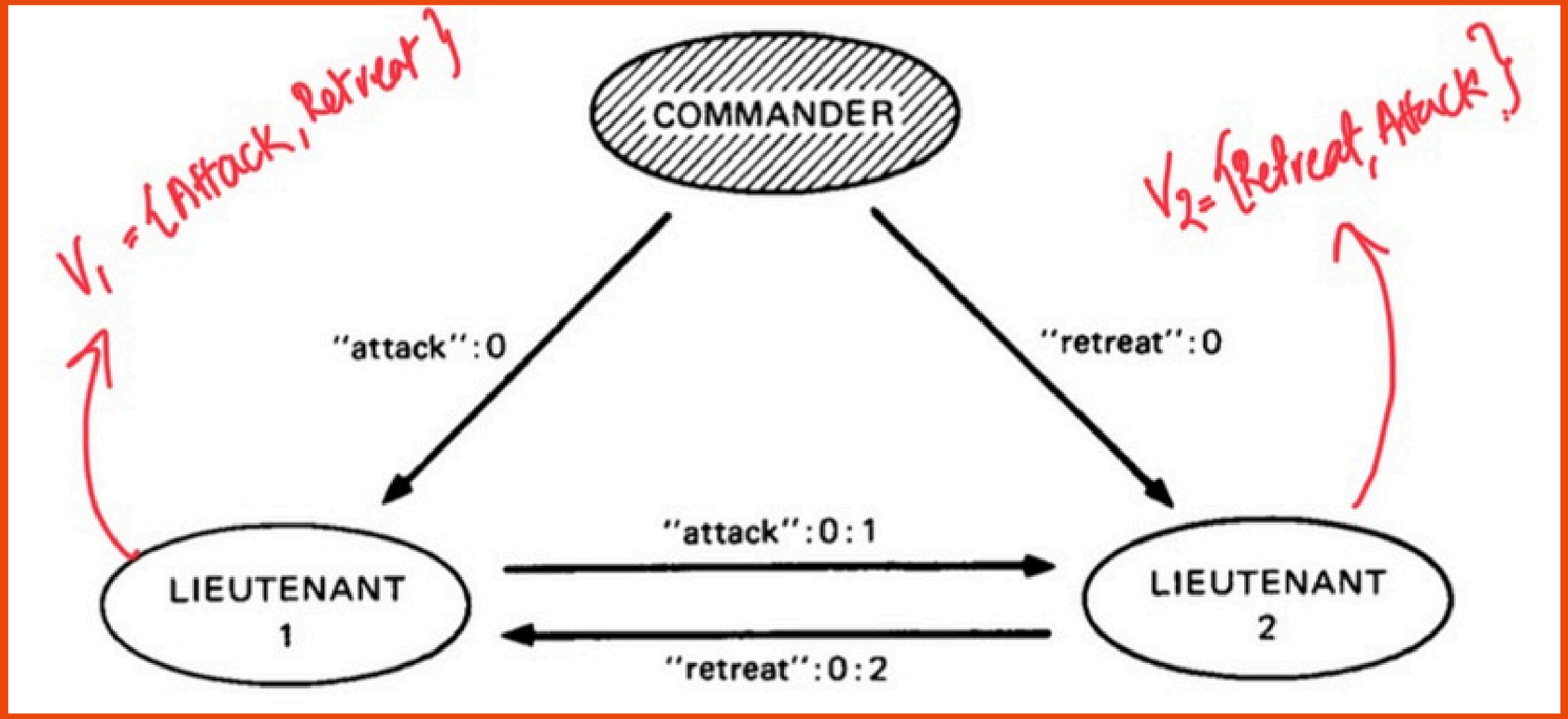
1. Commander signs & sends his order (e.g., “Attack:0”) to all lieutenants.
2. Each lieutenant who receives a new signed order:
   - Adds the order to his set $\mathbf{V_i}$.
   - Adds his own signature → forwards it to all other lieutenants.
3. When no more messages are incoming, each lieutenant executes **choice**($\mathbf{V_i}$) to decide final action.



# SOLUTION WITH SIGNED MESSAGES

## *EXAMPLE: SM(1) WITH 3 GENERALS*

*(COMMANDER IS A TRAITOR)*



# SOLUTION WITH SIGNED MESSAGES

## *WHY SM(M) ALWAYS WORKS*

Theorem 2:
For any m, Algorithm SM(m) solves the Byzantine Generals Problem if ≤ m traitors.
Proof Sketch:

- IC2: *(If commander is loyal)* → all loyal lieutenants receive and obey the same signed order.
- IC1: *(If commander is traitor)* → all loyal lieutenants still receive the same set of signed orders, so they make the same final choice.
- Recursive signature propagation ensures every loyal lieutenant sees the same set of valid orders.

# THANK YOU !

# RELIABLE SYSTEM

- Redundant processors compute the same output; majority voting selects the correct result.

- CONDITIONS FOR RELIABILITY:
  - All nonfaulty processors use the same input.
  - If the input unit is nonfaulty, all nonfaulty processors must use its value as input.

- Analogy to Byzantine Generals Problem:
  - Commander → Input device
  - Lieutenants → Processors
  - Loyal → Nonfaulty
  - Traitor → Faulty

# RELIABLE SYSTEM

- COMMUNICATION ASSUMPTIONS:
  - A1: Messages delivered correctly
  - A2: Sender can be identified
  - A3: Missing messages detectable (timeouts + synchronized clocks)
  - A4: Messages can be signed to prevent forgery

- Timeouts to work , two things are needed
  - A known maximum delay for message transmission
  - Synchronized clocks between sender and reciever

- If P1's and P2's clocks are not perfectly synchronized:
  - P1 might think it sent the message on time
  - P2 might think the message is late- even when it's not