

SCC311

Challenges of Distributed Systems



**THEY DON'T KNOW, THAT WE
KNOW, THAT THEY KNOW!**

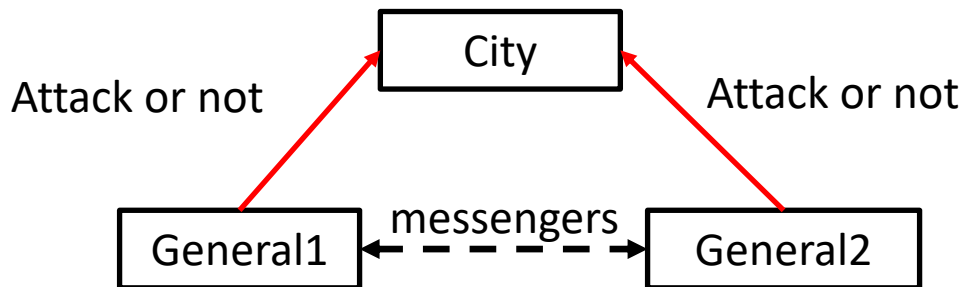
Overview of the Session

- Two Generals problem
- Byzantine Generals problem
- Fallacies of distributed computing
- Middleware

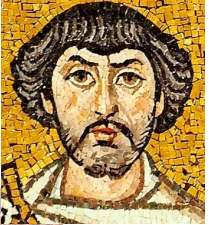
Two Generals Problem

■ Complication in distributed consensus

- Two generals, each leading an army, want to capture a city. The attack is only successful if both armies attack together
- Coordination is vital
- Generals need to reach consensus in order to succeed in conquering the city.



Belisarius



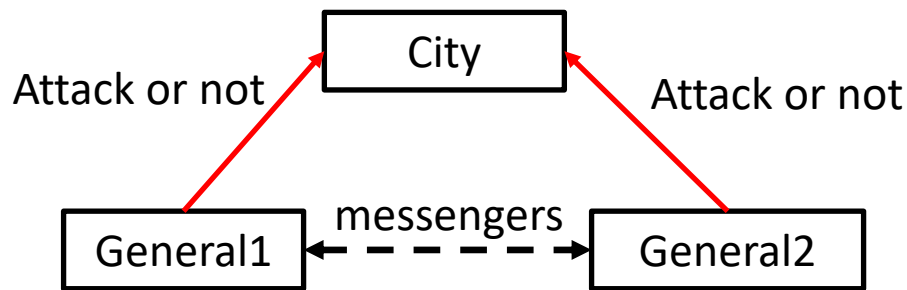
Narses



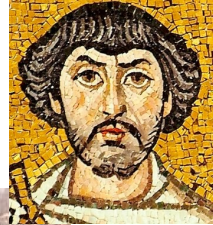
Two Generals Problem

■ Complication in distributed consensus

- Two generals, each leading an army, want to capture a city. The attack is only successful if both armies attack together



Belisarius



Narses

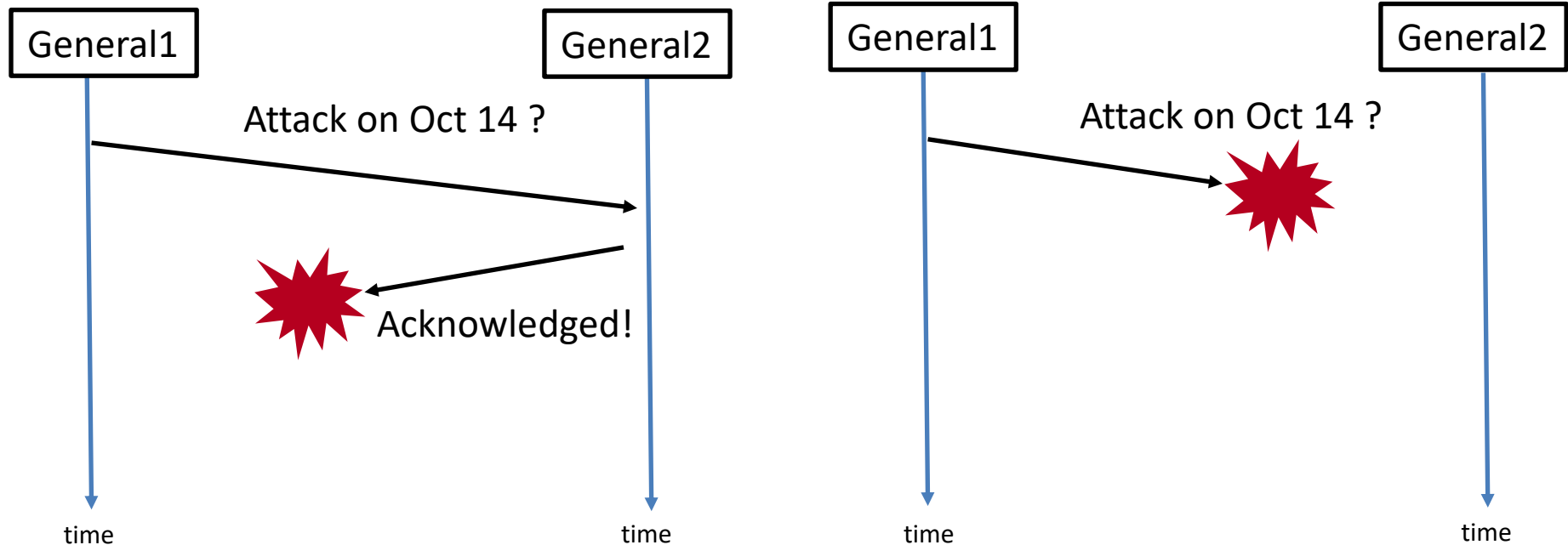


Army1	Army2	Result
No Attack	No attack	Nothing happens
No Attack	Attack	Army2 defeated
Attack	No attack	Army1 defeated
Attack	Attack	City captured

Two Generals Problem

■ Important challenges

- Faulty network
- Receipt of Acknowledgement is necessary



How the generals should decide?

- Is it possible to come up with an algorithm that works all the time ?
 - Answer is no.
- Possible approaches (none of them fully works):
- Approach 1
 - General1 sends a lot of messages at once, and then attacks
 - Problem: if all the messengers are captured, army1 is defeated
- Approach 2
 - General1 attacks only if an ACK is received from General 2
 - Now general2 will be captured if its response is intercepted or delayed
 - Problem: General2 has no idea if his ACK is received by General1
- In the two generals problem, we assume a faulty network but processes (i.e., generals) are not faulty.

FLP Impossibility Result

- **FLP impossibility result**

- Fischer, Lynch, and Patterson, [Impossibility of Distributed Consensus with One Faulty Process](#), 1985

- Asynchronous systems:

- No bounds on **message delivery time** and **process execution time**
 - **Clock drifts** are also unbounded

- When there is no upper bound on the time a process takes to complete its work and respond, it's *impossible* to make the distinction between a process that is crashed and one that is working (but taking very long to respond).
- FLP shows that there is no guarantee for distributed processes to reach consensus in an asynchronous environment, where it's possible for at least one process to crash.
- Equivalently, it's not always possible to detect failure in an asynchronous system with nodes crashing.
- So let's consider the case where messages are delivered reliably (i.e., network is not faulty) next.

Byzantine Generals Problem

We imagine that several divisions of the Byzantine army are camped outside an enemy city, each division has its own general. One commanding general (on behalf of the emperor) sends an order to the lieutenant generals. However, some of the generals, including the commanding general, may be traitors. The generals must have an algorithm to guarantee that:

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

L. Lamport, R. Shostak, and M. Pease, “The Byzantine Generals Problem”, 1982

Byzantine Generals Problem

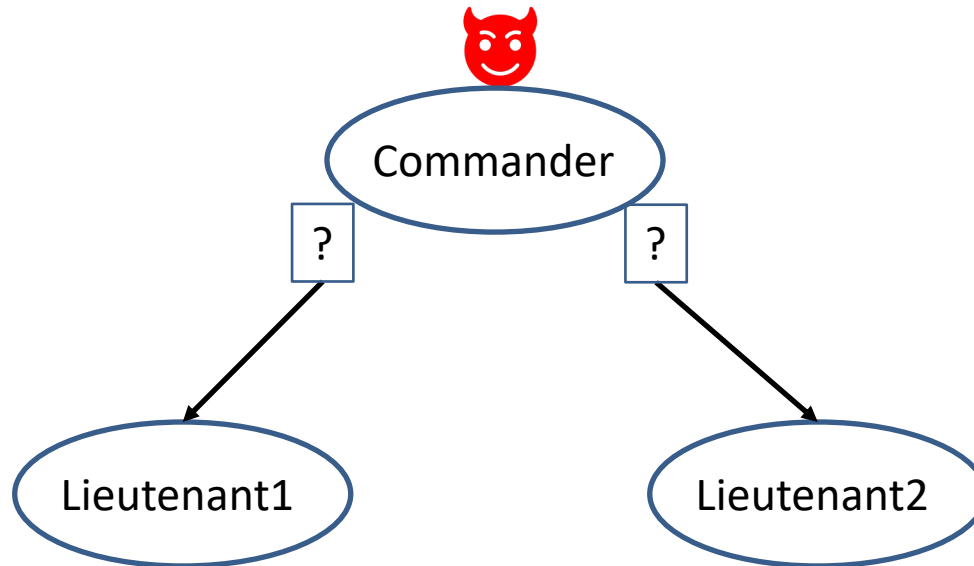
The Byzantine Generals Problem

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE
SRI International

Reliable computer systems must handle malfunctioning components that give conflicting information to different parts of the system. This situation can be expressed abstractly in terms of a group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement. It is shown that, using only oral messages, this problem is solvable if and only if more than two-thirds of the generals are loyal; so a single traitor can confound

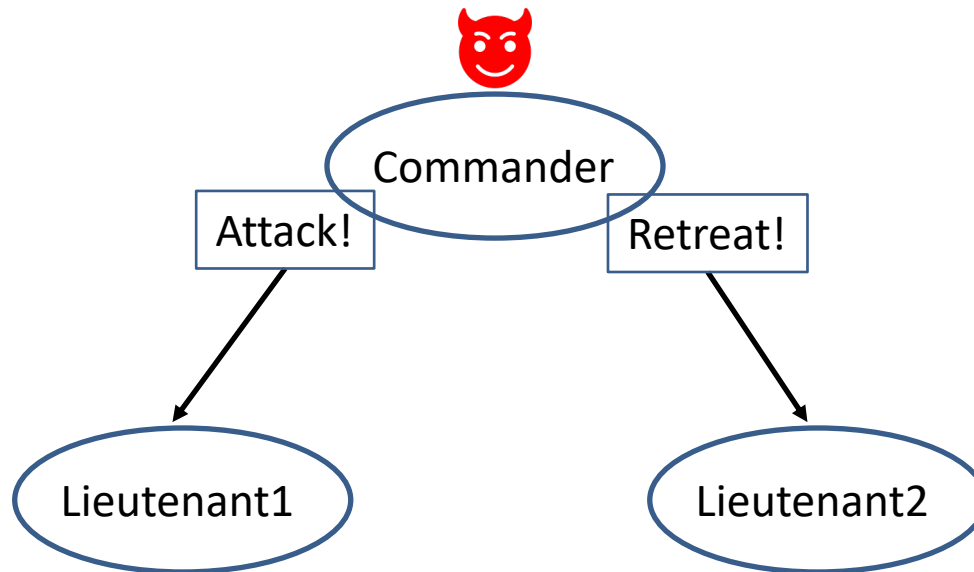
Byzantine Generals Problem

- 3 processes, commanding general is disloyal



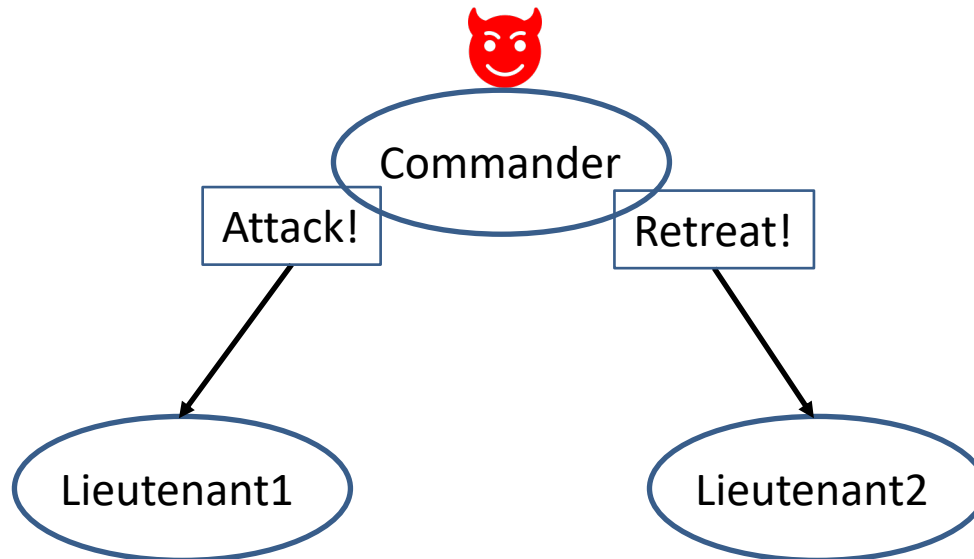
Byzantine Generals Problem

- 3 processes, commanding general is disloyal



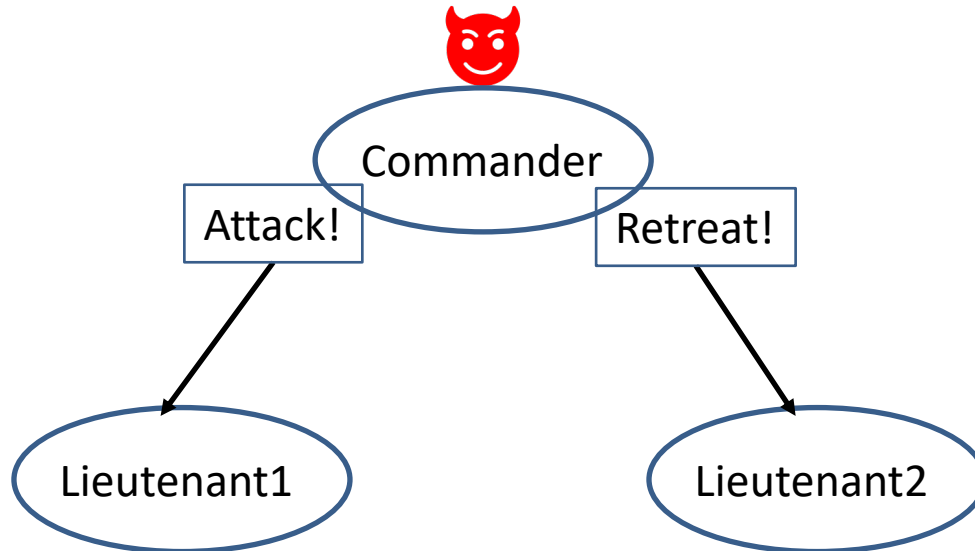
Byzantine Generals Problem

- What should the loyal lieutenants do to follow a unified course of action?



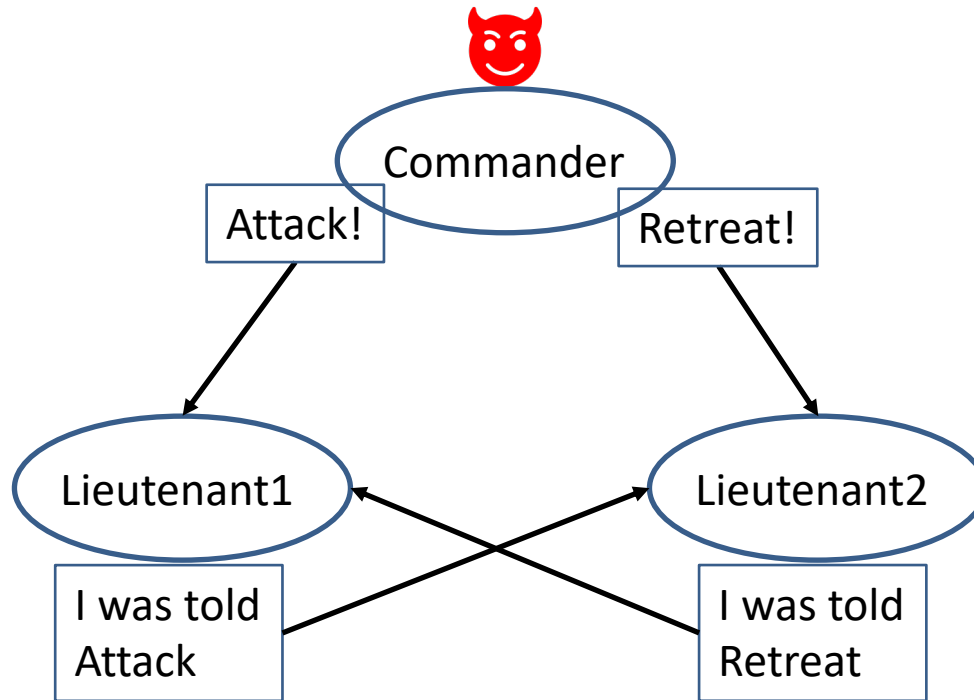
Byzantine Generals Problem

- What should lieutenants do ?
 - Lieutenants can communicate with one another and compare notes



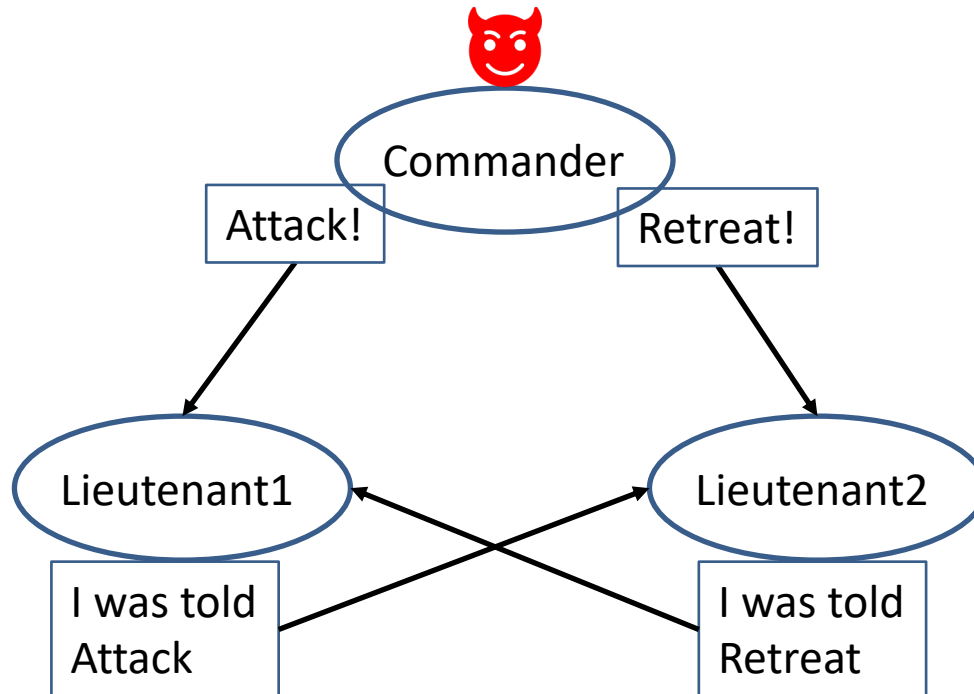
Byzantine Generals Problem

- Comparing notes (by sending messages to each other)
 - Remember this time network is not faulty



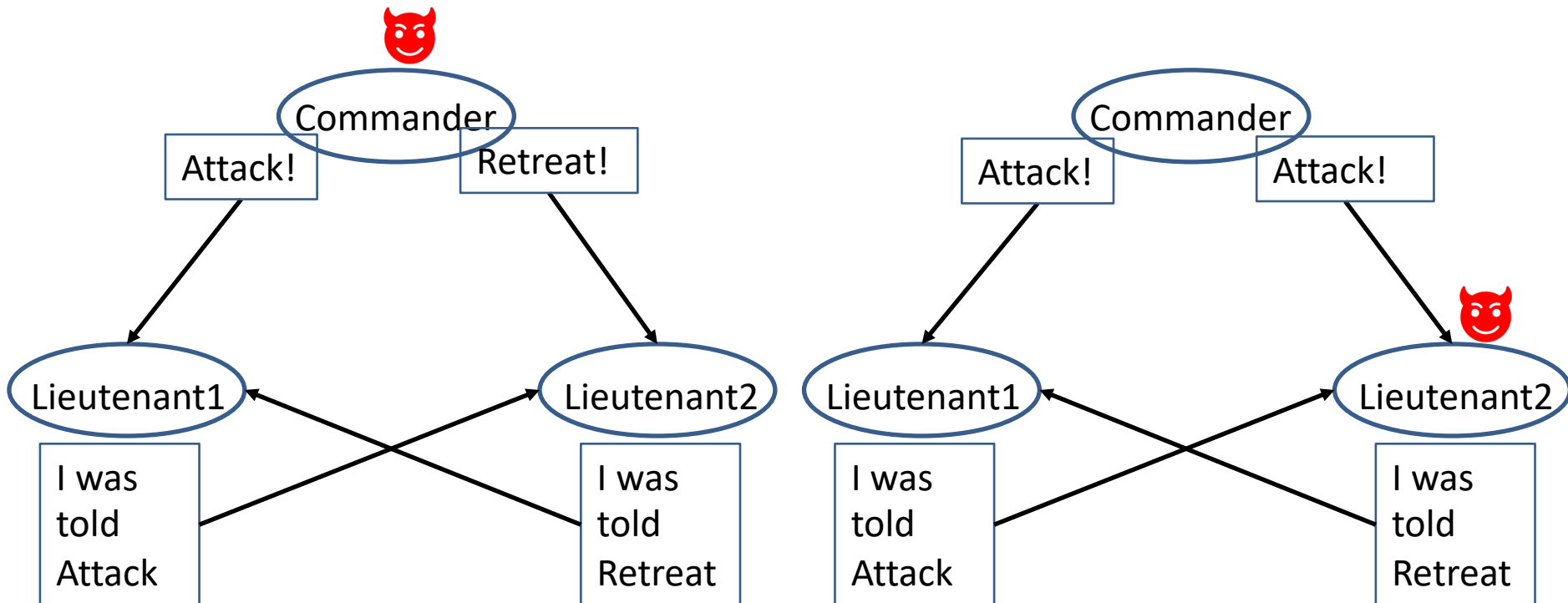
Byzantine Generals Problem

- Lieutenants can't agree on the commander's order so they do the default action which is retreat.



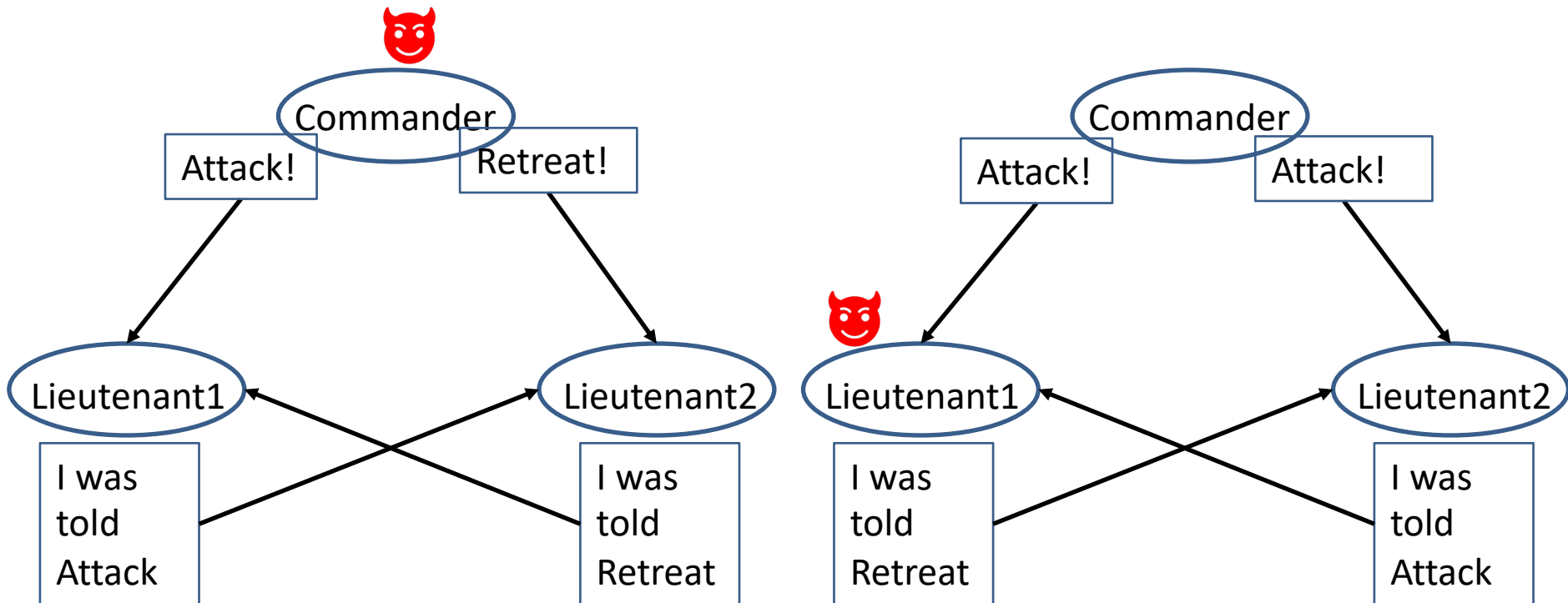
Byzantine Generals Problem

- Both scenarios are equivalent from Lieutenant1's point of view



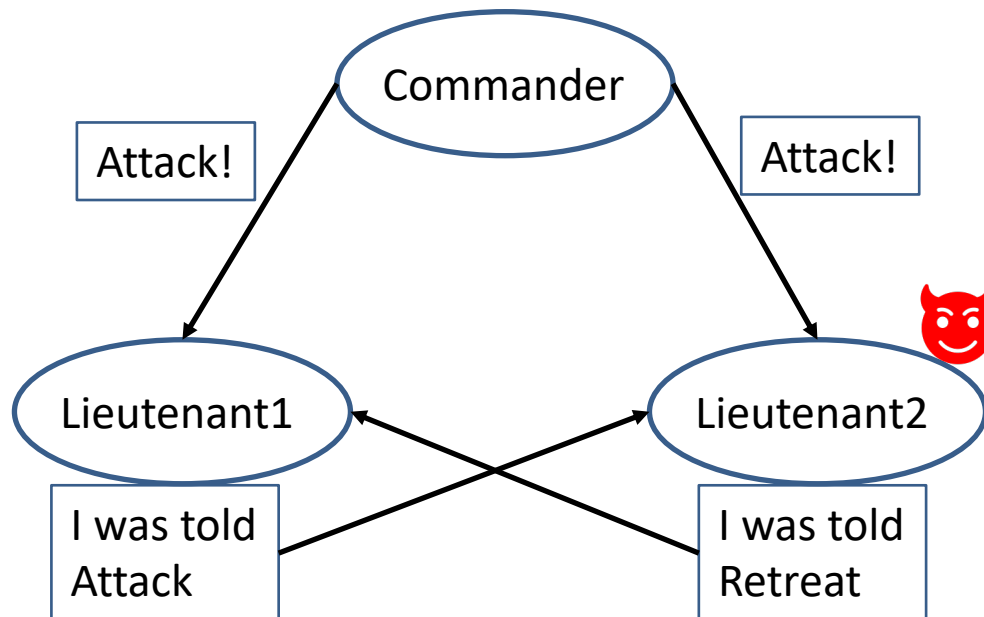
Byzantine Generals Problem

- Both scenarios are equivalent from Lieutenant2's point of view



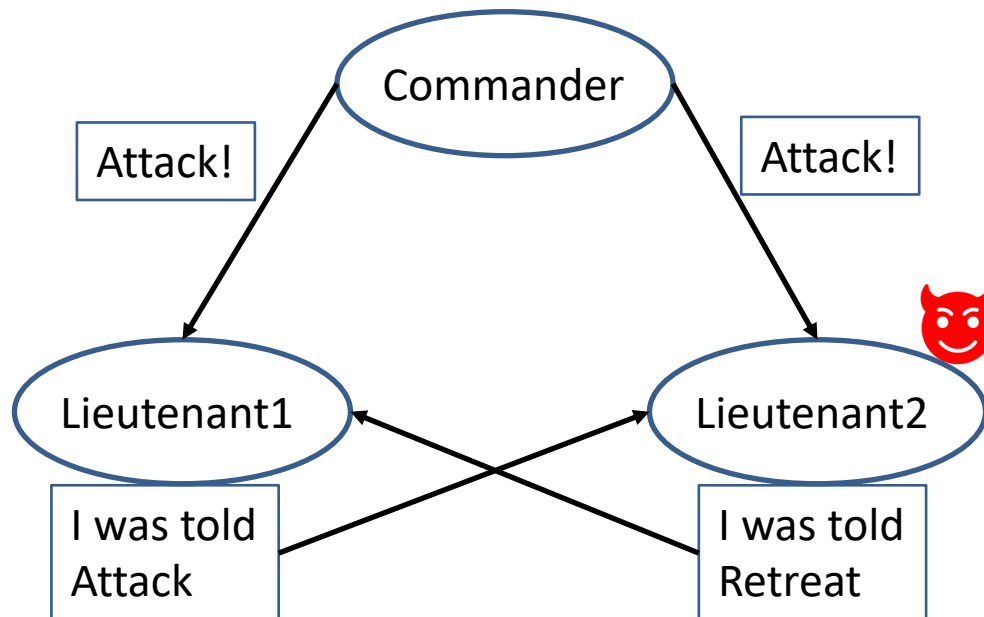
Byzantine Generals Problem

- With $d = 1$ traitor, the Byzantine Generals problem is not solvable with 3 generals
 - Consensus is not possible
- You need at least 4 nodes! (see next slide)



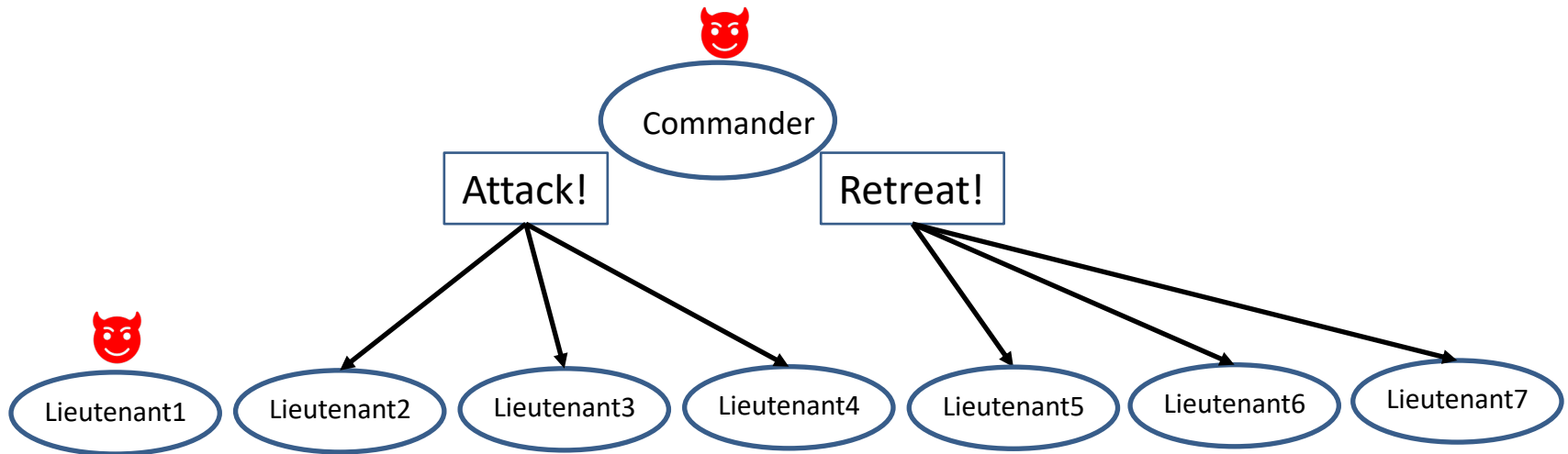
Byzantine Generals Problem

- With more than 3 participants, the strategy is still the same
 - All lieutenants compare notes with each other
 - **Then follow the majority “vote”**
 - If the majority was told “Attack”, then attack
 - If the majority was told “Retreat!”, then retreat



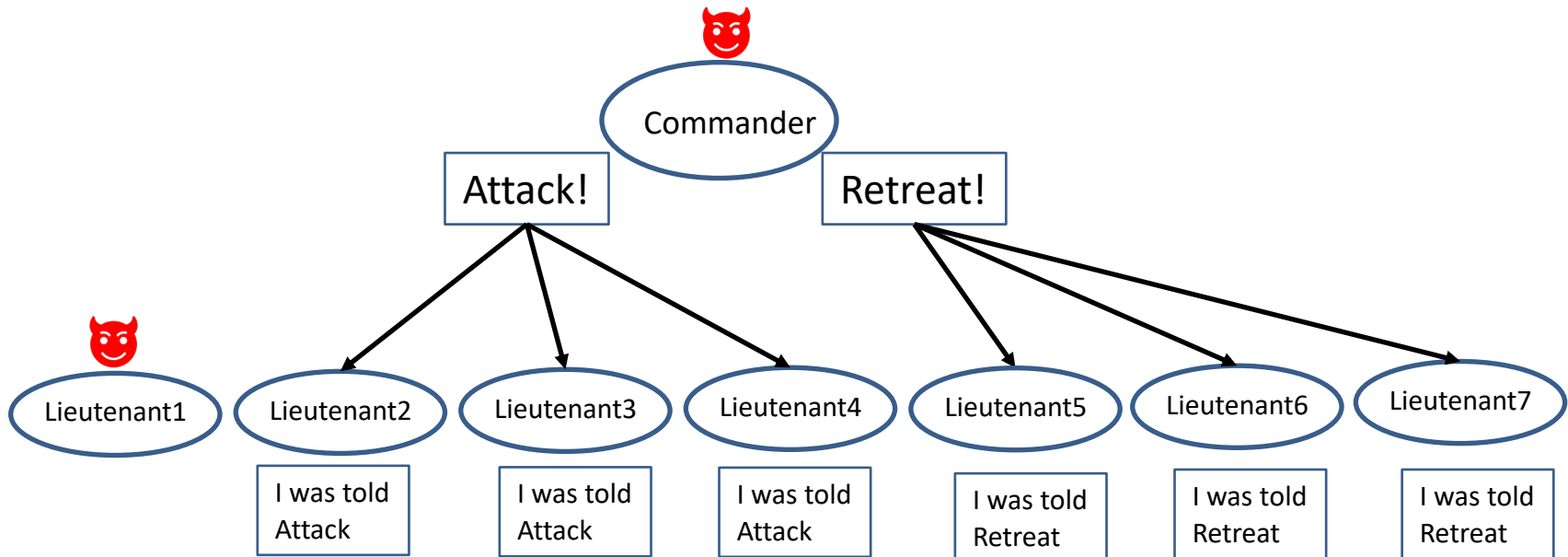
Byzantine Generals Problem

- Communicating notes only work with one traitor



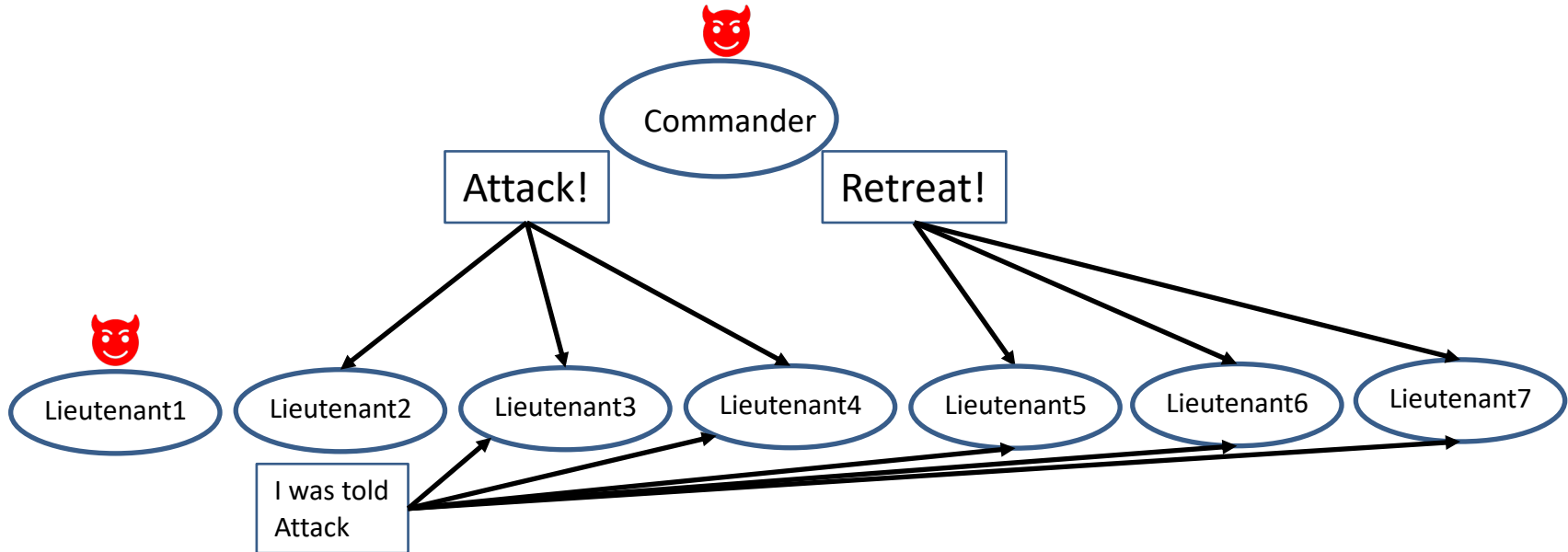
Byzantine Generals Problem

- Let's say all lieutenants compare notes as before
- Each lieutenant sends a message to every other lieutenant and indicates the order it received



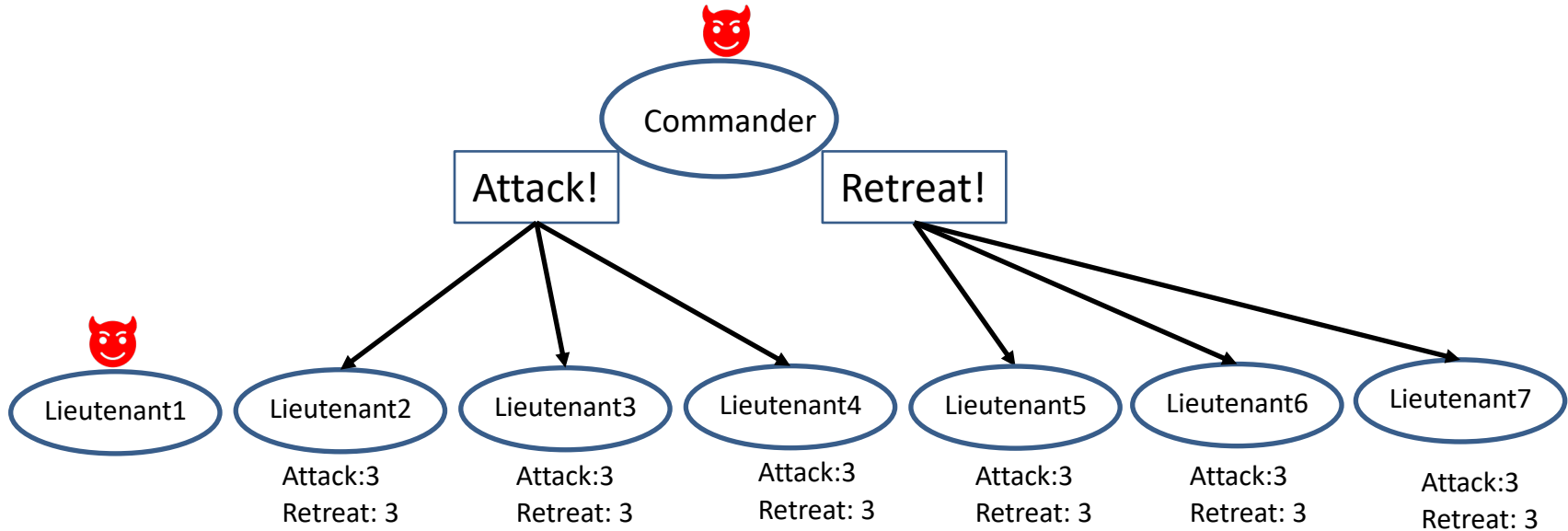
Byzantine Generals Problem

- Let's say all lieutenants compare notes as before
- Each lieutenant sends a message to every other lieutenant and indicates the order it received



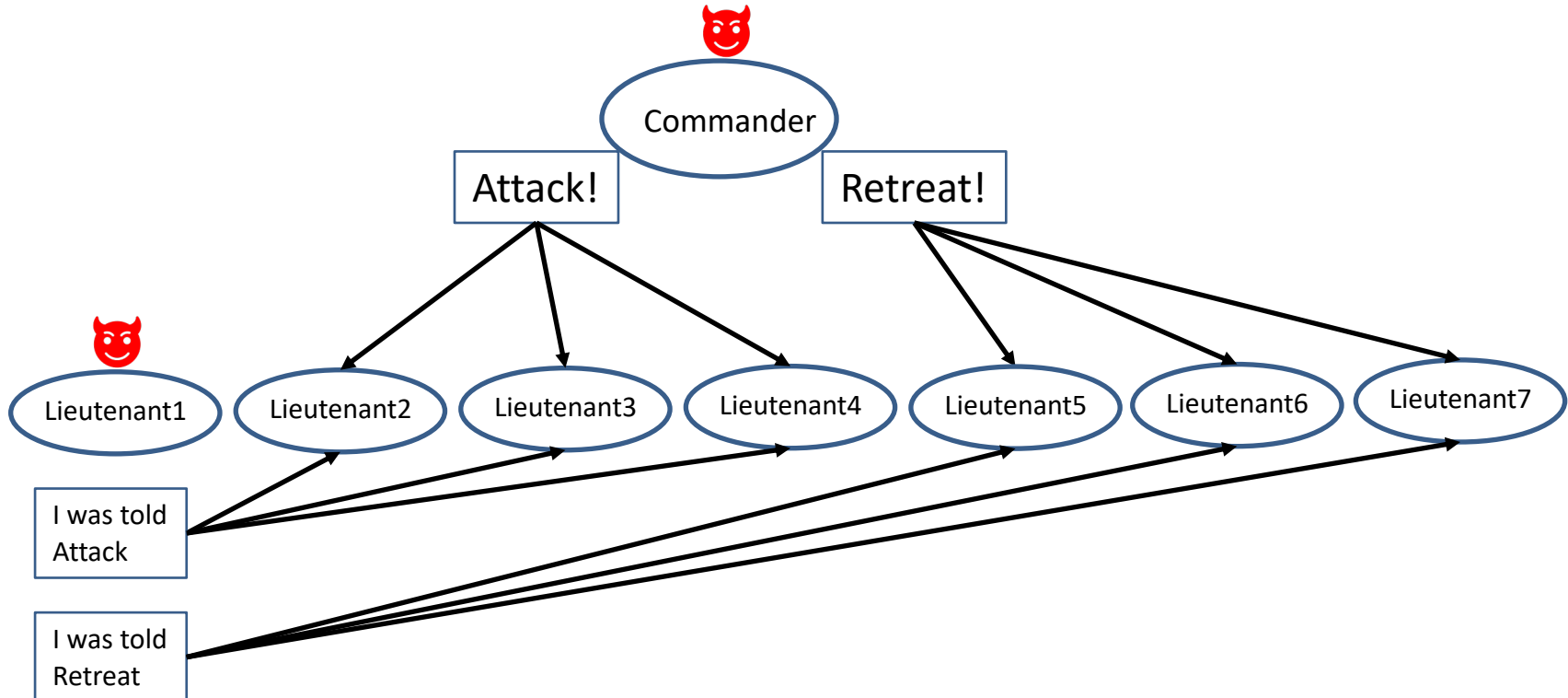
Byzantine Generals Problem

- How can the traitor lieutenant disrupt the majority vote and cause chaos ?



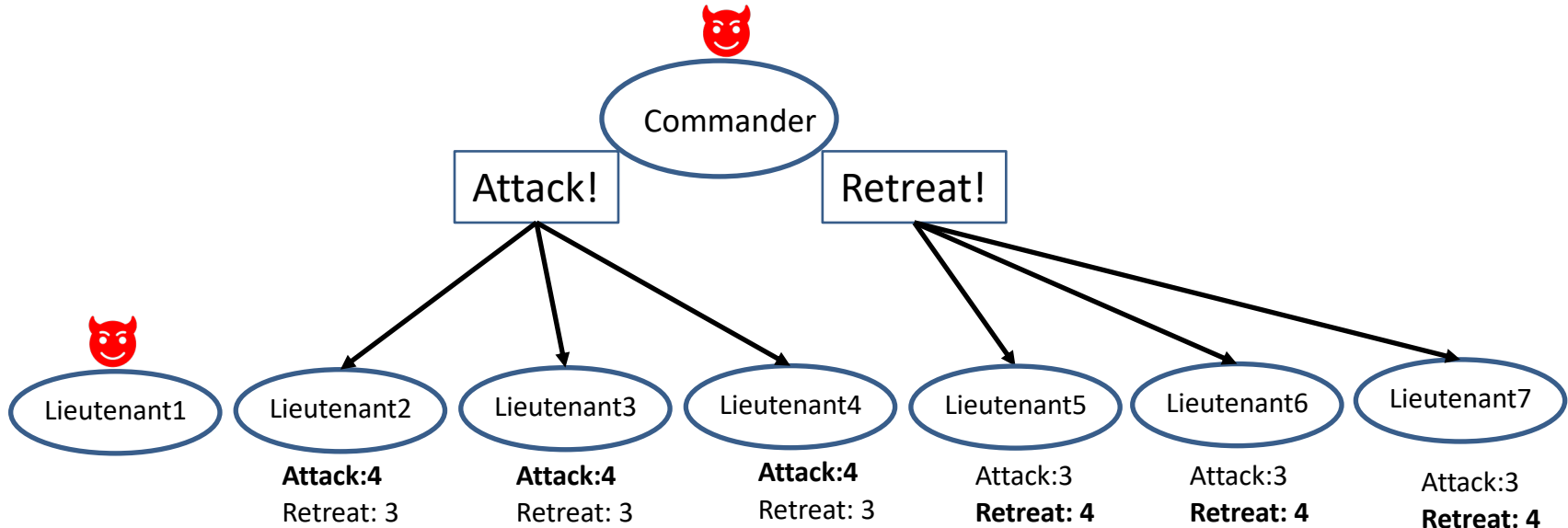
Byzantine Generals Problem

- How can the traitor lieutenant disrupt the majority vote and cause chaos ?



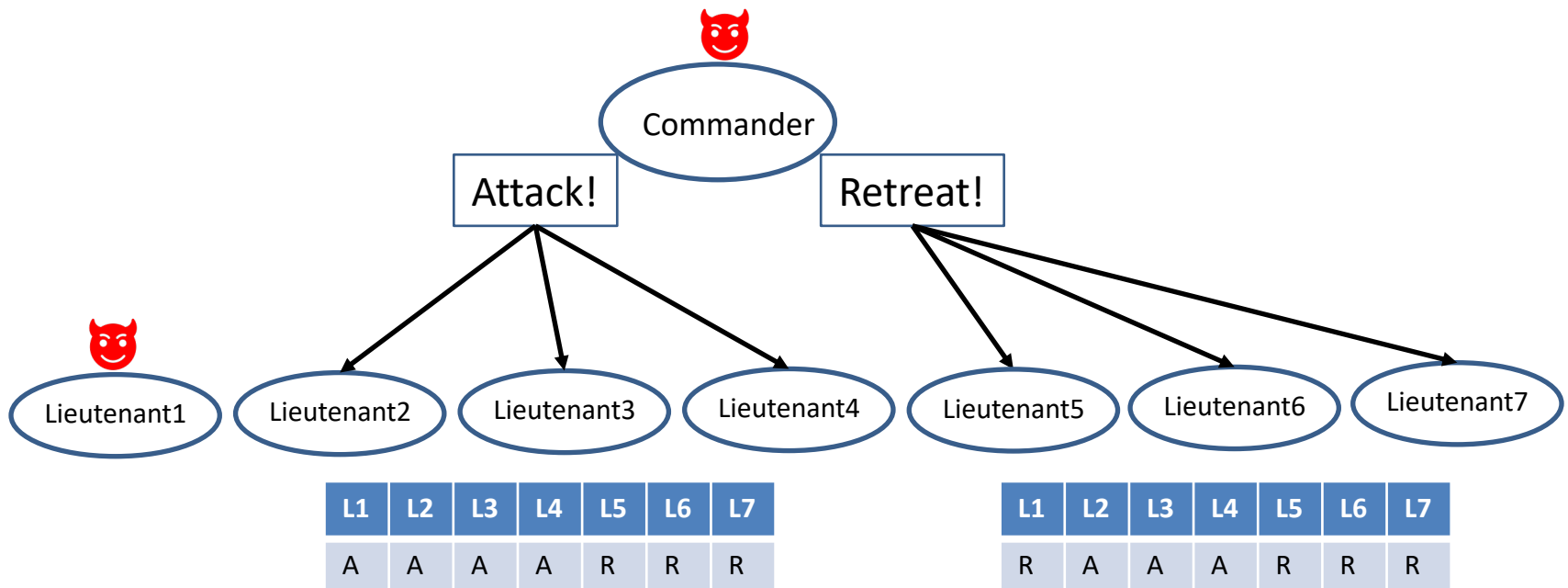
Byzantine Generals Problem

- How can the traitor lieutenant disrupt the majority vote and cause chaos ?



Byzantine Generals Problem

- The solution to this problem is more rounds of communication between lieutenants!
- In addition to what they heard from the commander, lieutenants also share what they heard from each other.



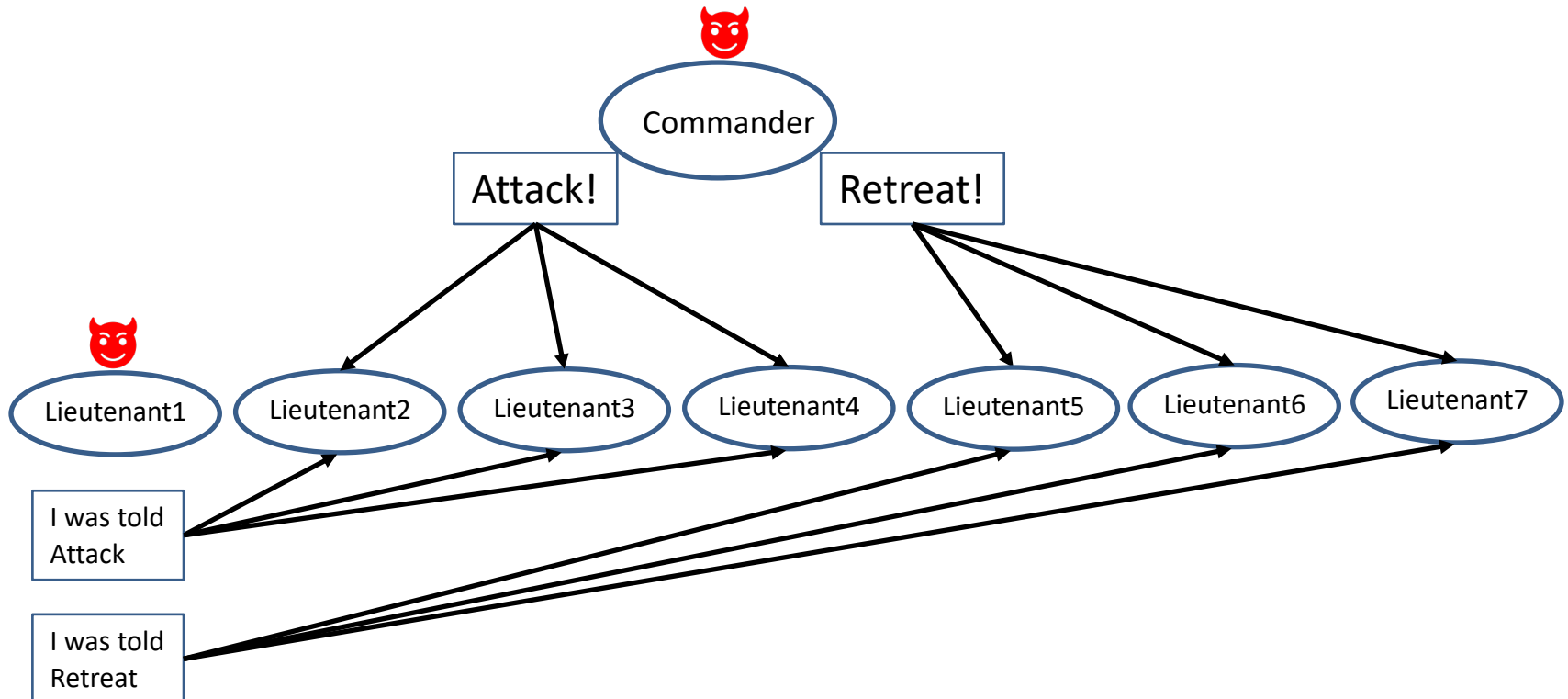
Byzantine Generals Problem

■ For lieutenant2:

L1		L3	L4	L5	L6	L7
A		A	A	R	R	R

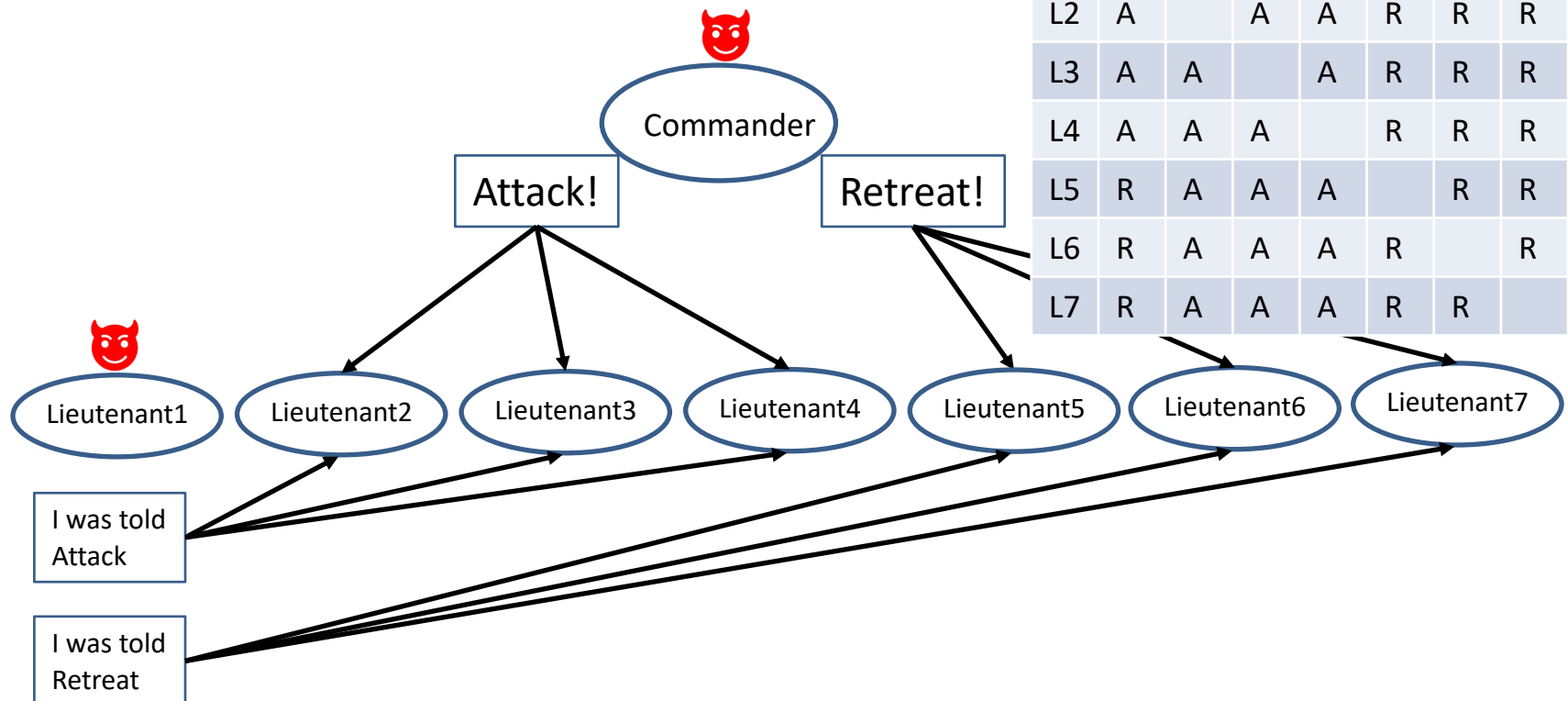
■ For lieutenant5:

L1	L2	L3	L4		L6	L7
R	A	A	A		R	R



Byzantine Generals Problem

- Then, the lieutenants exchange their vectors with each other
 - Each compose a table, where each row is a vector that is exchanged in the previous round



Byzantine Generals Problem

- **Assumption:** there is a default action in case loyal lieutenants can not agree on an action
 - Default = Retreat
- Then, the lieutenants exchange their vectors with each other
 - Lieutenant5:

	L1	L2	L3	L4	L5	L6	L7
L1		X	X	X	X	X	X
L2	A		A	A	R	R	R
L3	A	A		A	R	R	R
L4	A	A	A		R	R	R
L5	R	A	A	A		R	R
L6	R	A	A	A	R		R
L7	R	A	A	A	R	R	

Apply majority() to each column

?	A	A	A	R	R	R
---	---	---	---	---	---	---

Summary of Lamport's Algorithm

- For d traitors, there has to be more than $3d$ generals
- For d faulty nodes, the algorithm requires a total of $d+1$ rounds of communication (d rounds of comparing notes + order from the commander)
- Lamport's algorithm provides **Byzantine Fault tolerance, given that you have more than $3d$ generals (processes)**
 - Byzantine fault model: a process can behave arbitrarily (even maliciously as in the Byzantine Generals problem)
 - There are other fault models such as crash-stop (i.e., a process can stop responding), which are less challenging to deal with than the Byzantine fault model
 - **We will cover fault tolerance in detail during week 5**
- Can we somehow make sure lieutenants don't forge orders that they didn't receive?
 - Public key cryptography can help with this – signed messages
 - However, traitors can collaborate and share their keys with each other
 - We will cover public key cryptography next week

Applications of Byzantine Fault Tolerance

- Space flight
 - Space shuttle had 4 computers for decision making
 - They used a majoritarian system in case of disagreements
- Cryptocurrency: there are financial incentives to act maliciously
- Problems stem from having multiple control systems
 - e.g. air traffic control, airline bookings, critical manufacturing processes, stock exchange, etc.
- Lamport's algorithm is expensive! – $O(n^2)$ messaging overhead each round.
- Alternative: accept uncertainty, just work around it
 - e.g. send a predetermined number of messages and assume at least one gets to the destination
 - Assume at most $d=1$ or 2 simultaneous bad actors

Further complications

■ Heterogeneity

- Access
- Platform
- Format
- Administration

■ Changing nature

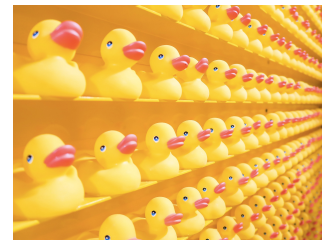
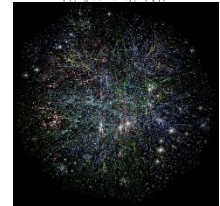
- Increase/decrease in scale
- Churn
- Relocation
- Failure



Fallacies of distributed computing

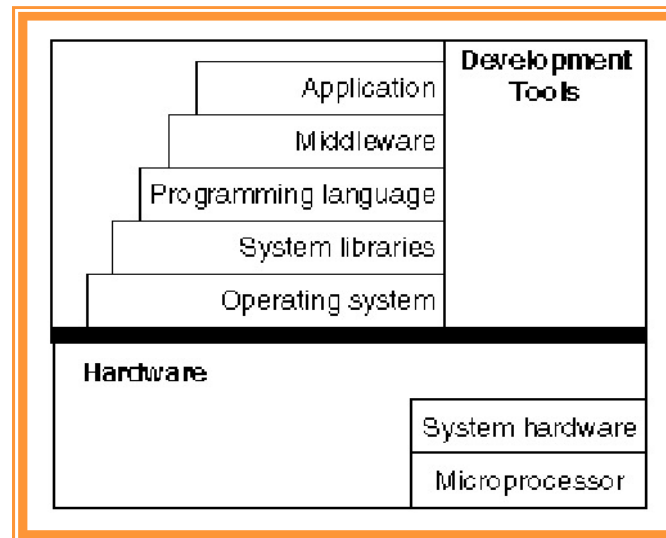
■ Asserted by Peter Deutsch (Sun Microsystems)

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. There is one administrator
6. Transport cost is zero
7. The network is homogeneous
8. Topology doesn't change



The Take-away Message?

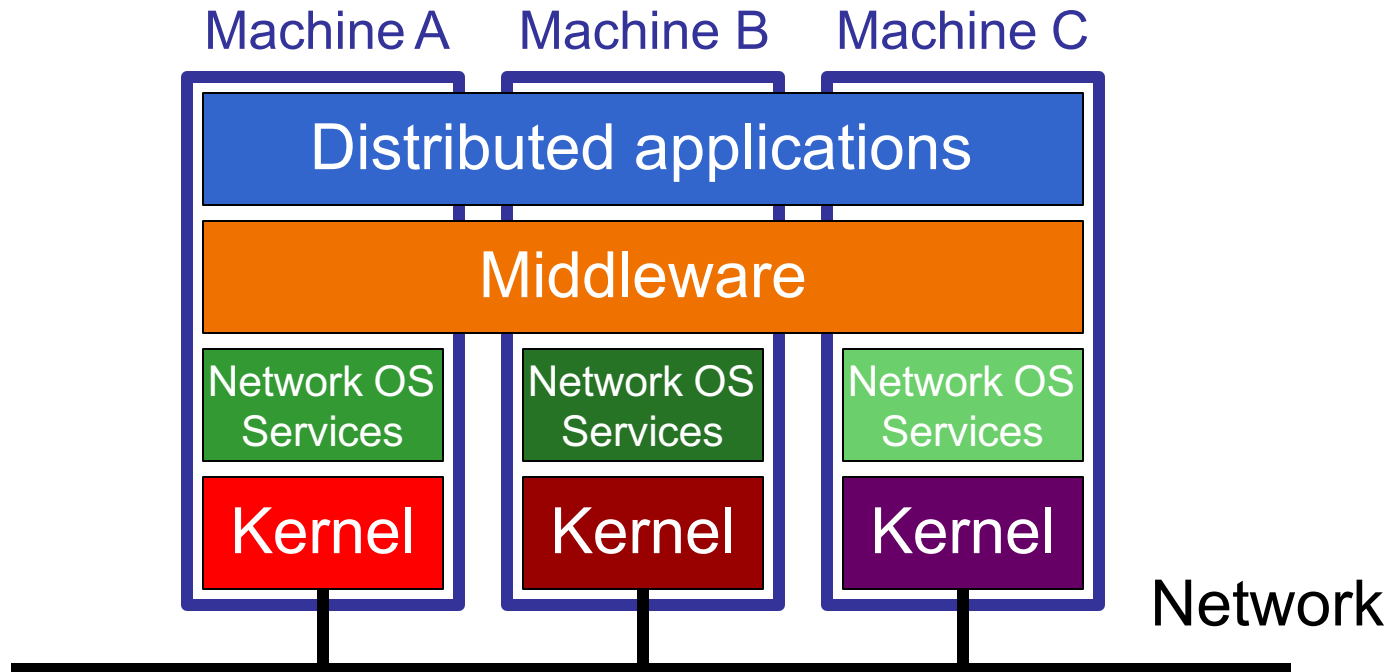
- Challenging
 - Developing good software is difficult
 - Developing good distributed software is even harder
- To do this you need help!
 - Very hard to build such systems on bare-bones devices
 - Strong need for software **platforms** -> **Middleware**



Middleware

■ Rationale

- Provide a high level programming **abstraction**
- Hide the **complexity** associated with distributed systems (including the underlying forms of **heterogeneity**)



Goals of a Middleware Platform

Middleware provides protocols to support:

■ Resource sharing

- The ability to **access and share resources** in a distributed environment
- Distributed locking

■ Transparency

- The ability to view a distributed system as if it were a single computer
- Varying **dimensions of transparency** incl. location, access, migration, etc.

■ Openness

- The offering of services according to standard rules (syntax and semantics)
- Openness provides support for the key properties of:
 - **Portability**: ability to run across different platforms
 - **Interoperability**: having separate parts harmoniously work together

■ Extensibility

- The ability to be able to introduce **new or modified functionality**

Goals of a Middleware Platform

- Scalability: the ability to grow/shrink with respect to
 - Size
 - e.g. support massive growth in the number of users of a service
 - Geography
 - e.g. supporting systems across continents (dealing with latencies, etc.)
 - Administration
 - e.g. supporting systems spanning many different administrative organisations
- Dependability/ Quality of Service
 - Security
 - Providing secure and authenticated channels, access control, key management, etc.
 - Fault tolerance
 - Providing **highly available** and **resilient** distributed applications and services

Styles of Middleware

- **Client-server** platforms

- e.g. DCE (Distributed Computing Environment) – first!

- **Distributed object** technology

- e.g. CORBA, Java RMI

- **Component-based** programming

- e.g. Fractal, Enterprise Java Beans, OpenCOM

- **Microservice** architecture

- Loosely coupled, testable services using CI/CD

- **Others** styles

- Resource discovery platforms (e.g. Jini)
 - Group communication services (e.g. JGroups)
 - Publish-subscribe systems (e.g. JMS)
 - Distributed file systems, distributed transaction services, distributed document-based systems, agent-based systems, message-oriented middleware, P2P technologies, etc.

Expected Learning Outcomes

At the end of this session, you should:

- Have an intuition of why realising good distributed systems is difficult
 - Byzantine generals problem as an example
- Understand what the **role of middleware** in supporting the development of distributed applications and services
 - Goals of middleware systems
 - Some different styles of middleware

Additional Reading

- **CDKB, chapter 1 section 1.5**
→ also chapter 3 for revision
- TvS, chapter 1, and sections 8.1-8.2
- Lamport, Shostak, Pease, “The Byzantine Generals Problem”, ACM Transactions on Programming Languages and Systems, July 1982. <http://dl.acm.org/citation.cfm?id=357176>
- Fischer, Lynch, Patterson, “Impossibility of Distributed Consensus with One Faulty Process”, Journal of ACM, April 1985.
<https://groups.csail.mit.edu/tds/papers/Lynch/jacm85.pdf>
- Lamport, “Paxos made simple”,
<http://www.cs.utexas.edu/users/lorenzo/corsi/cs380d/past/03F/notes/paxos-simple.pdf>
- Rotem-Gal-Oz, “Fallacies of Distributed Computing Explained”,
<http://www.rgoarchitects.com/Files/fallacies.pdf> (2006).