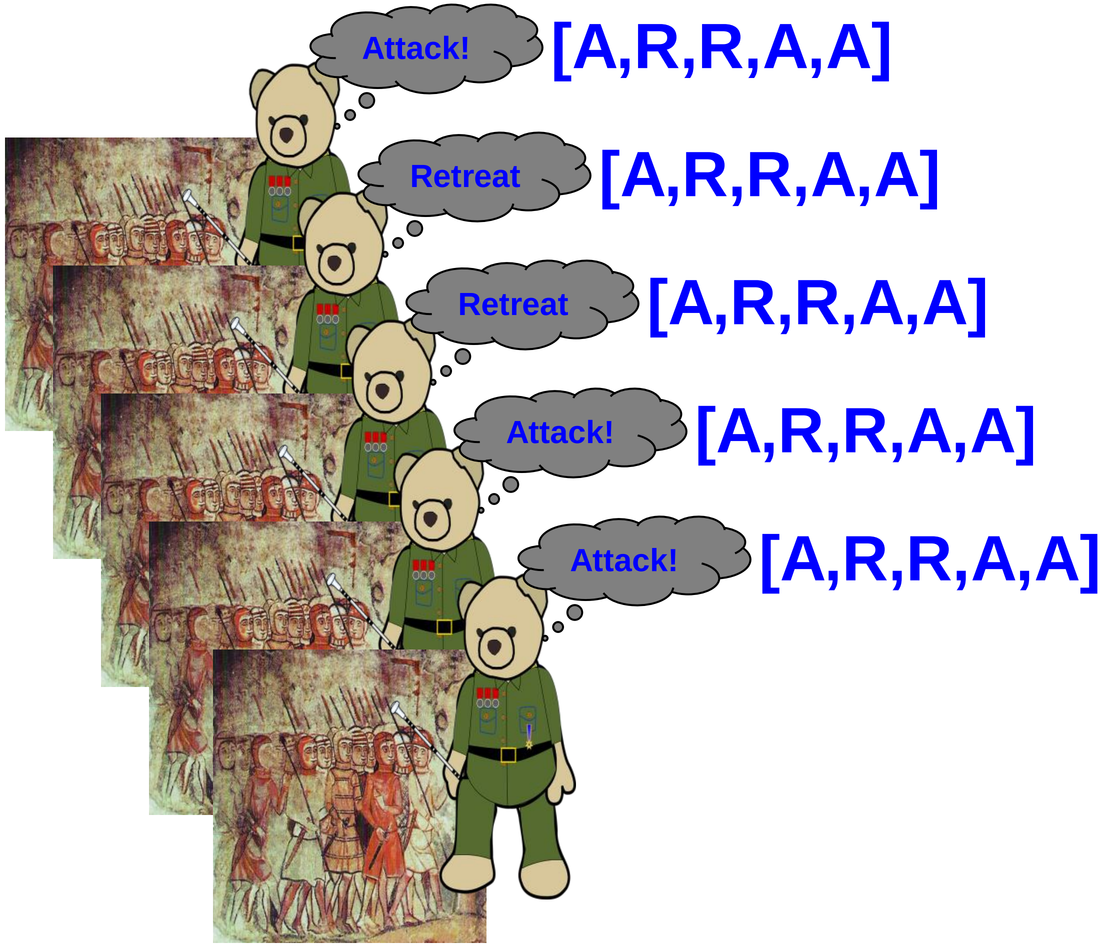
Byzantine Failures

- The network includes faulty processes that do not terminate but continue to participate in the execution of the algorithm.
- The behavior of the processes may be completely unpredictable.
- The internal state of a faulty process may change during the execution of a round arbitrarily, without receiving any message.
- A faulty process may send a message with any content (i.e., fake messages), independently of the instructions of the algorithm.
- We call such kind of failures as Byzantine failures.
- We use byzantine failures to model malicious behavior (e.g. cyber-security attacks).

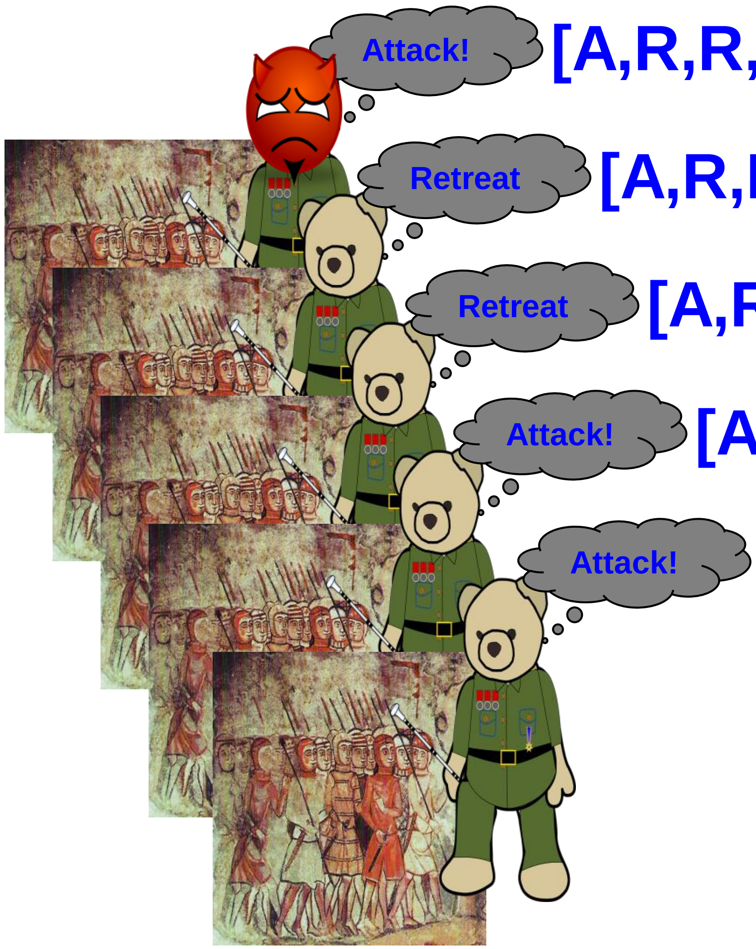
Why study Byzantine Fault Tolerance?

- Does this happen in the real world?
  The "one in a million" case.
  - Malfunctioning hardware,
  - Buggy software,
  - Compromised system due to hackers.

The problem


Attack!
[A,R,R,A,A]
Retreat
[A,R,R,A,A]
Retreat
[A,R,R,A,A]
Attack!
[A,R,R,A,A]
Attack!
[A,R,R,A,A]

The problem



[A,R,R,A,A]

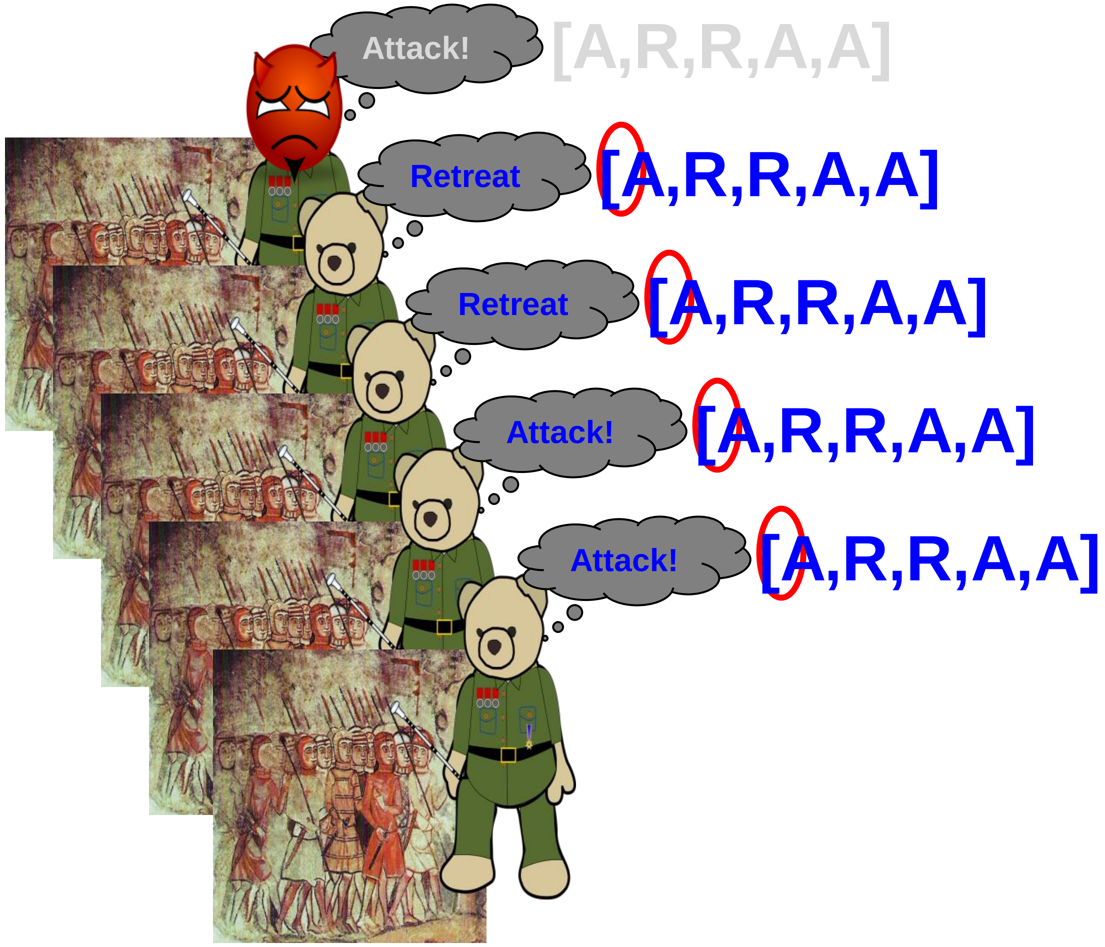
[A,R,R,A,A]

[A,R,R,A,A]

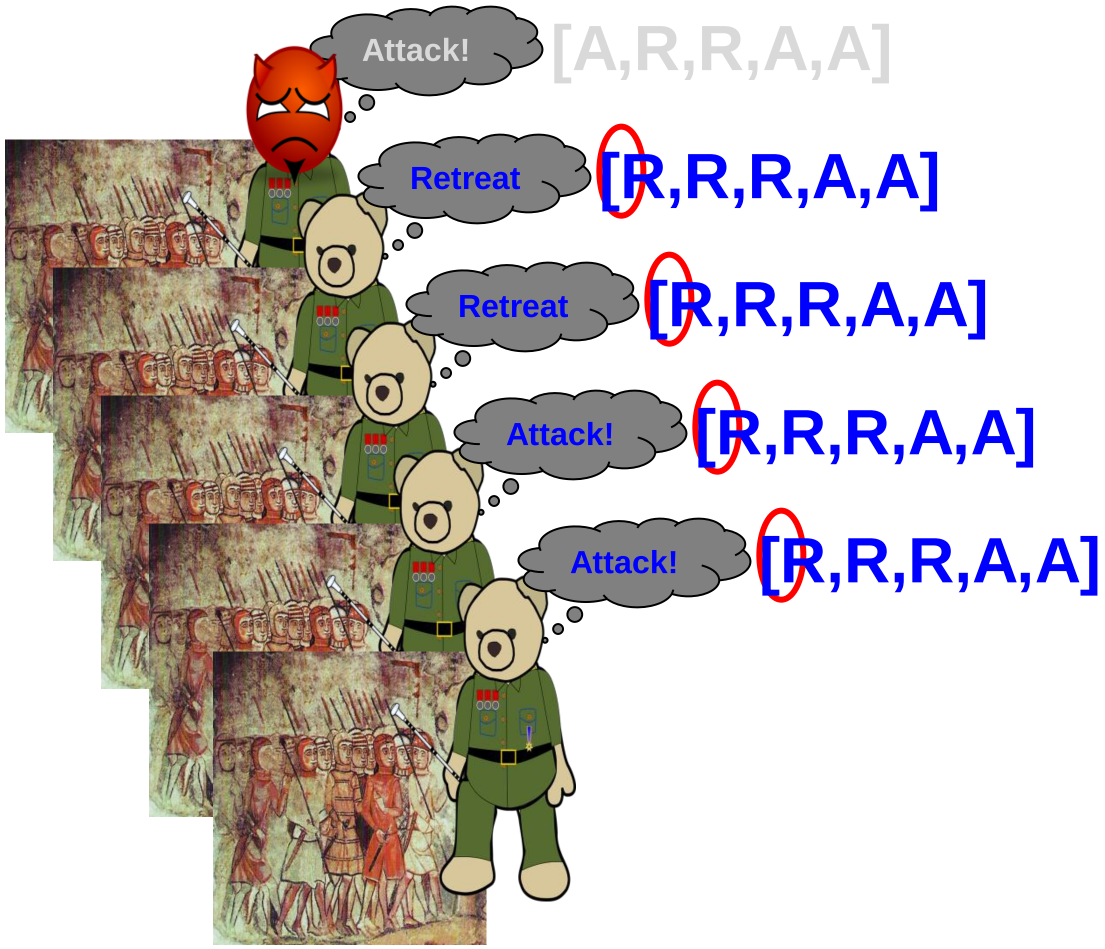
[A,R,R,A,A]

[A,R,R,A,A]

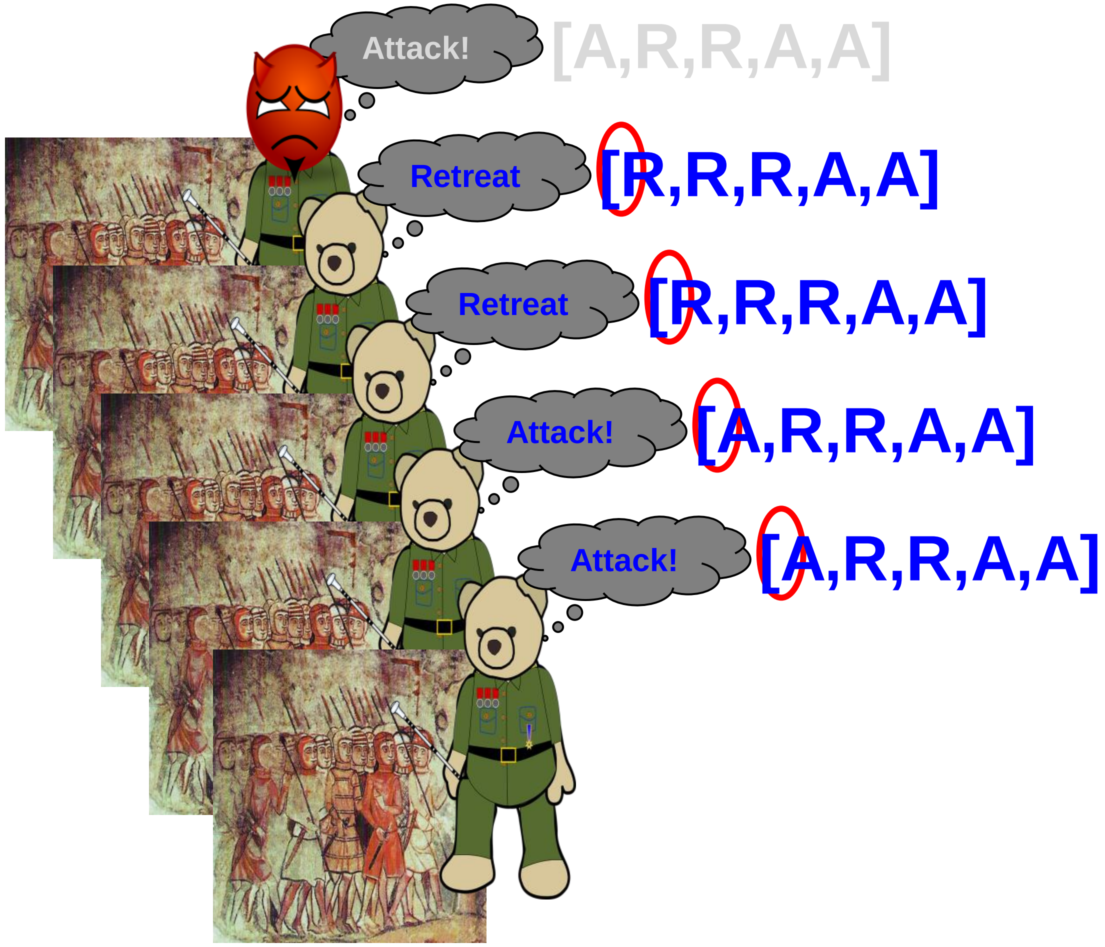
The problem


Attack!
[A,R,R,A,A]
Retreat
[A,R,R,A,A]
Retreat
[A,R,R,A,A]
Attack!
[A,R,R,A,A]
Attack!
[A,R,R,A,A]

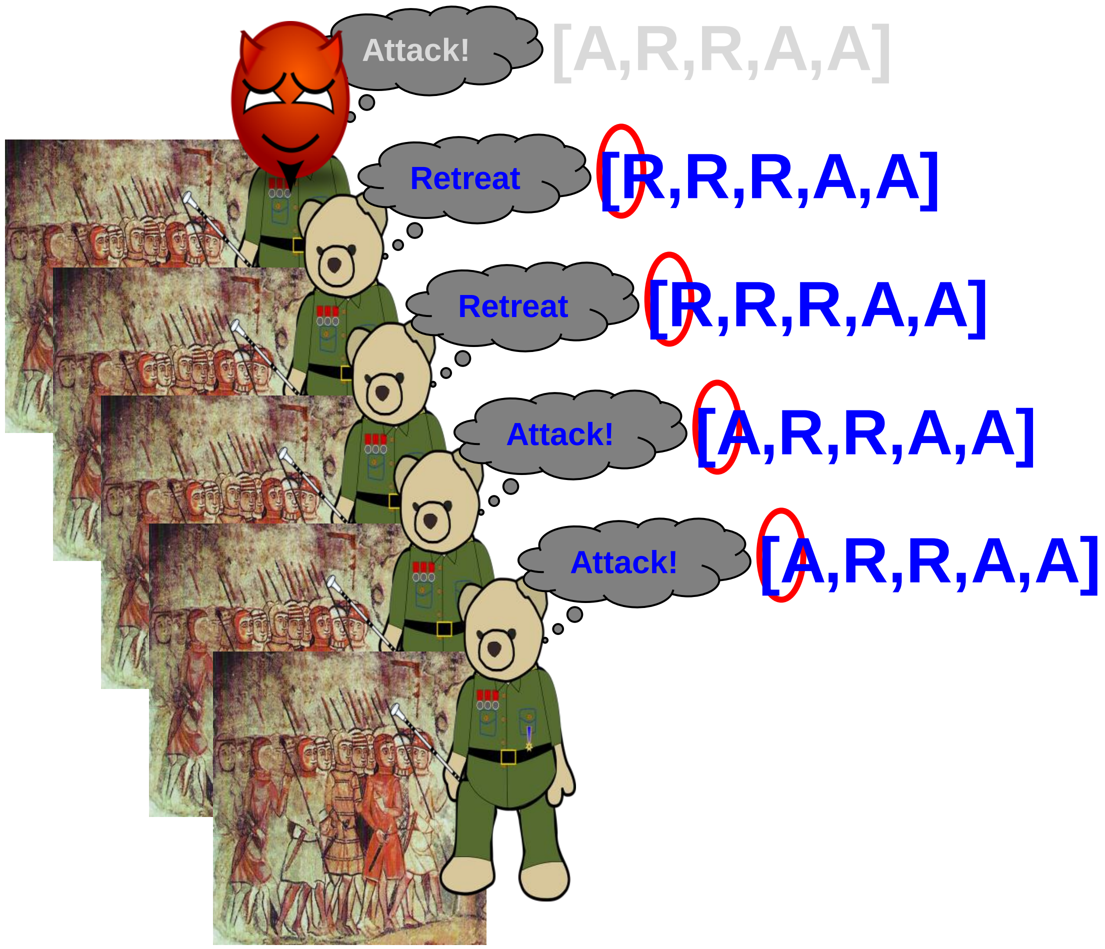
The problem


Attack!
[A,R,R,A,A]
Retreat
[R,R,R,A,A]
Retreat
[R,R,R,A,A]
Attack!
[R,R,R,A,A]
Attack!
[R,R,R,A,A]

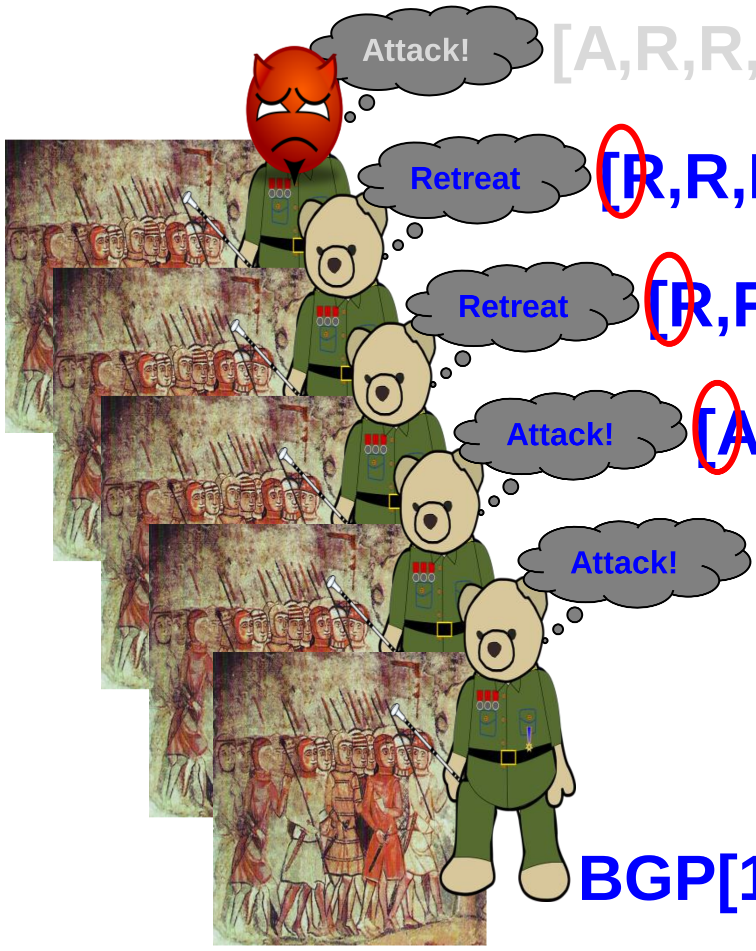
The problem


Attack!
[A,R,R,A,A]
Retreat
[R,R,R,A,A]
Retreat
[R,R,R,A,A]
Attack!
[A,R,R,A,A]
Attack!
[A,R,R,A,A]

The problem


Attack!
[A,R,R,A,A]
Retreat
[R,R,R,A,A]
Retreat
[R,R,R,A,A]
Attack!
[A,R,R,A,A]
Attack!
[A,R,R,A,A]

The problem



[A,R,R,A,A]

[R,R,R,A,A]

[R,R,R,A,A]

[A,R,R,A,A]

[A,R,R,A,A]

BGP[1]= all loyal generals agree on what 1st general wants

How many traitors can there be?

Solving Byzantine Generals

Lamport, Shostak and Pease Algorithm

L. Lamport, R. Shostak, M. Pease: "The Byzantine Generals Problem", ACM Transactions on Programming Languages and Systems, 4(3): pp 382-401, 1982.

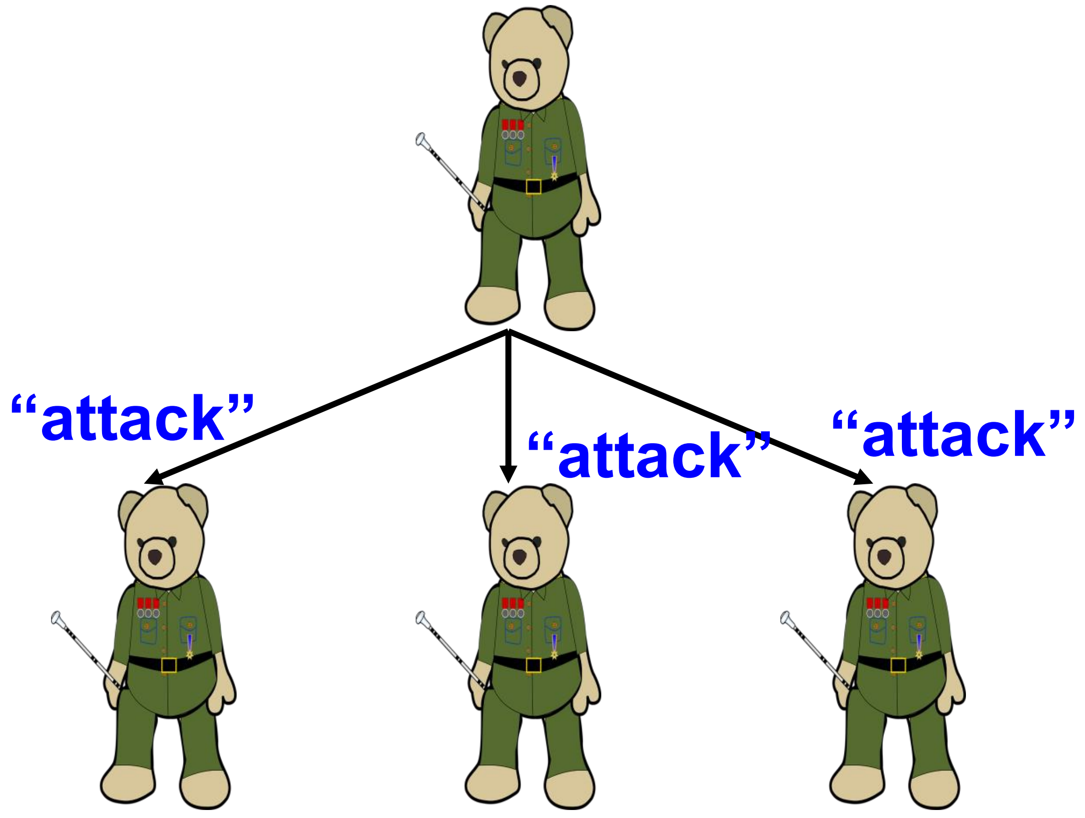
- The algorithm makes three assumptions regarding communication:
  1. All message transmissions are delivered correctly.
  2. The receivers knows the identity of the sender.
  3. The absence of a message can be detected.
- The 1st and 2nd assumptions limit the traitor from interfering with the transmissions of the other generals.
- The 3rd assumptions prevents the traitor to delay the attack by not sending any message.
- In computer networks conditions 1 and 2 assume that the processors are directly connected and communication failures are counted as part of the β failures.

What *can* you do?

- Assuming:
    - Less than ⅓ of generals are traitors
    - Oral messages
    - No crypto

UM(m): solution to BGP for ≤m traitors

Inductive Solution for Oral Messages

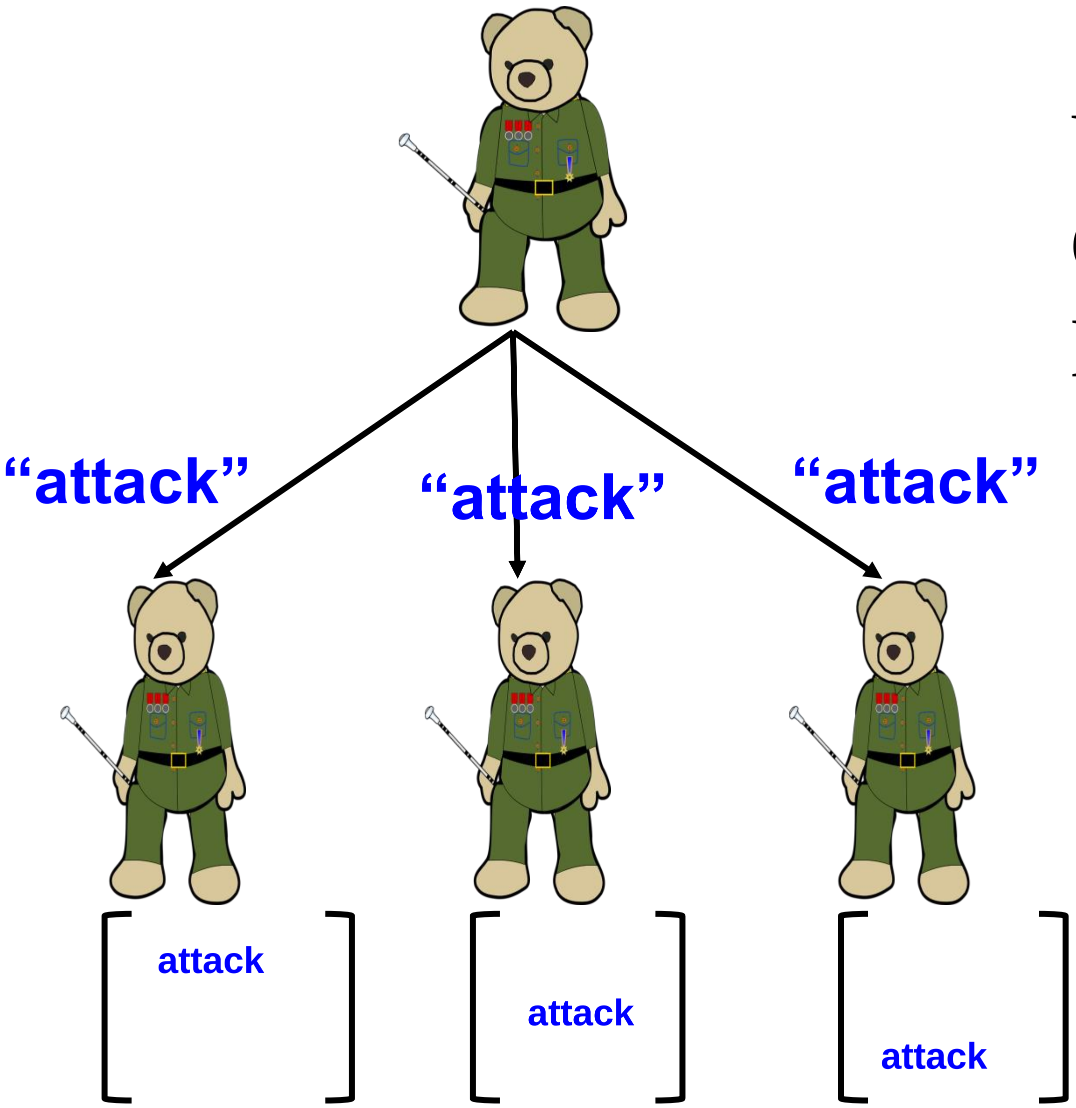


UM(4,0):

C: Sends order.

L: Follows if received.

Inductive Solution for Oral Messages



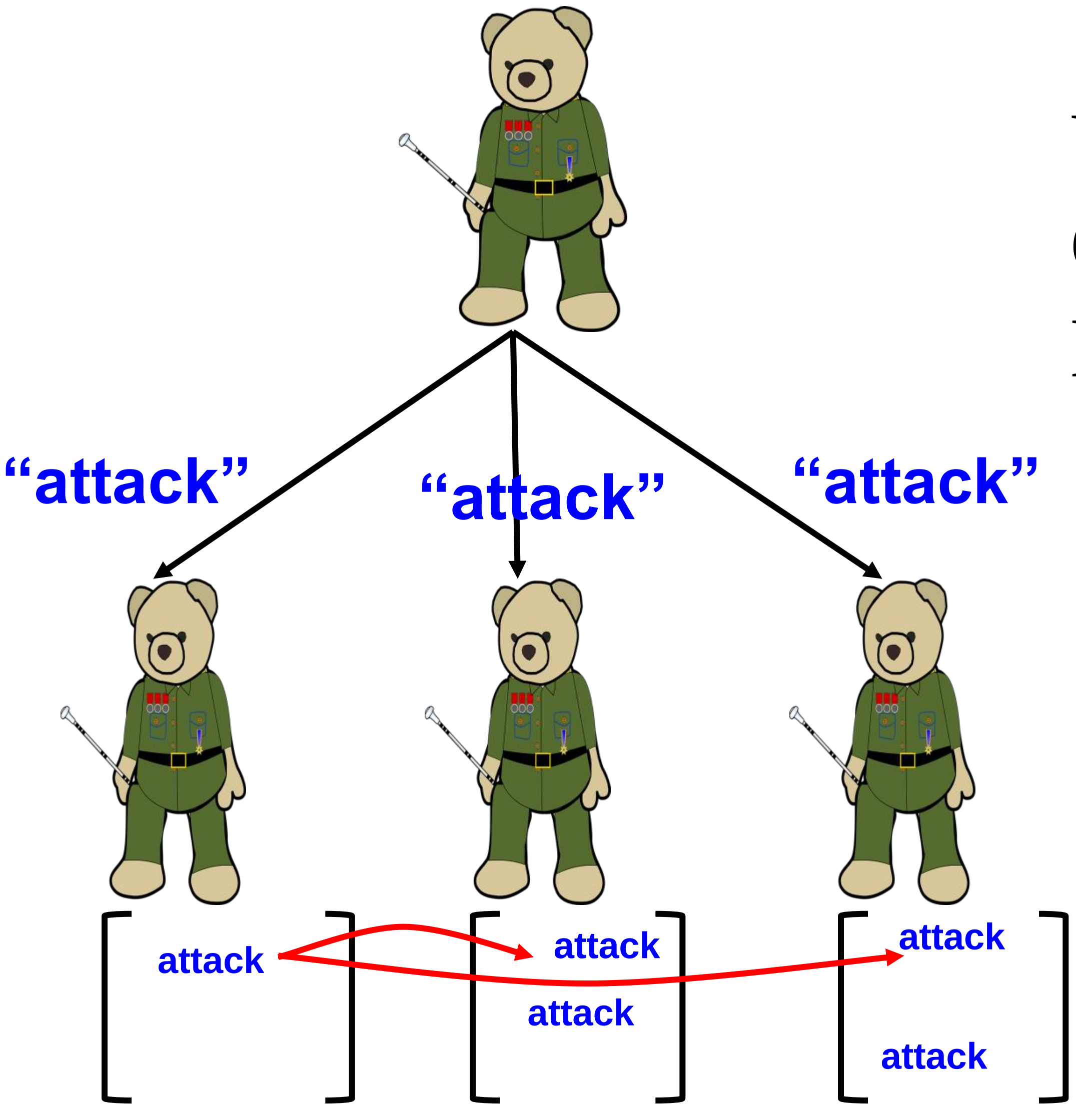
UM(4,m), m>0:

C: Sends order.

L:

1. **Records if received.**
2. **Use UM(3,m-1) to tell others.**
3. **Follows majority() order.**

Inductive Solution for Oral Messages



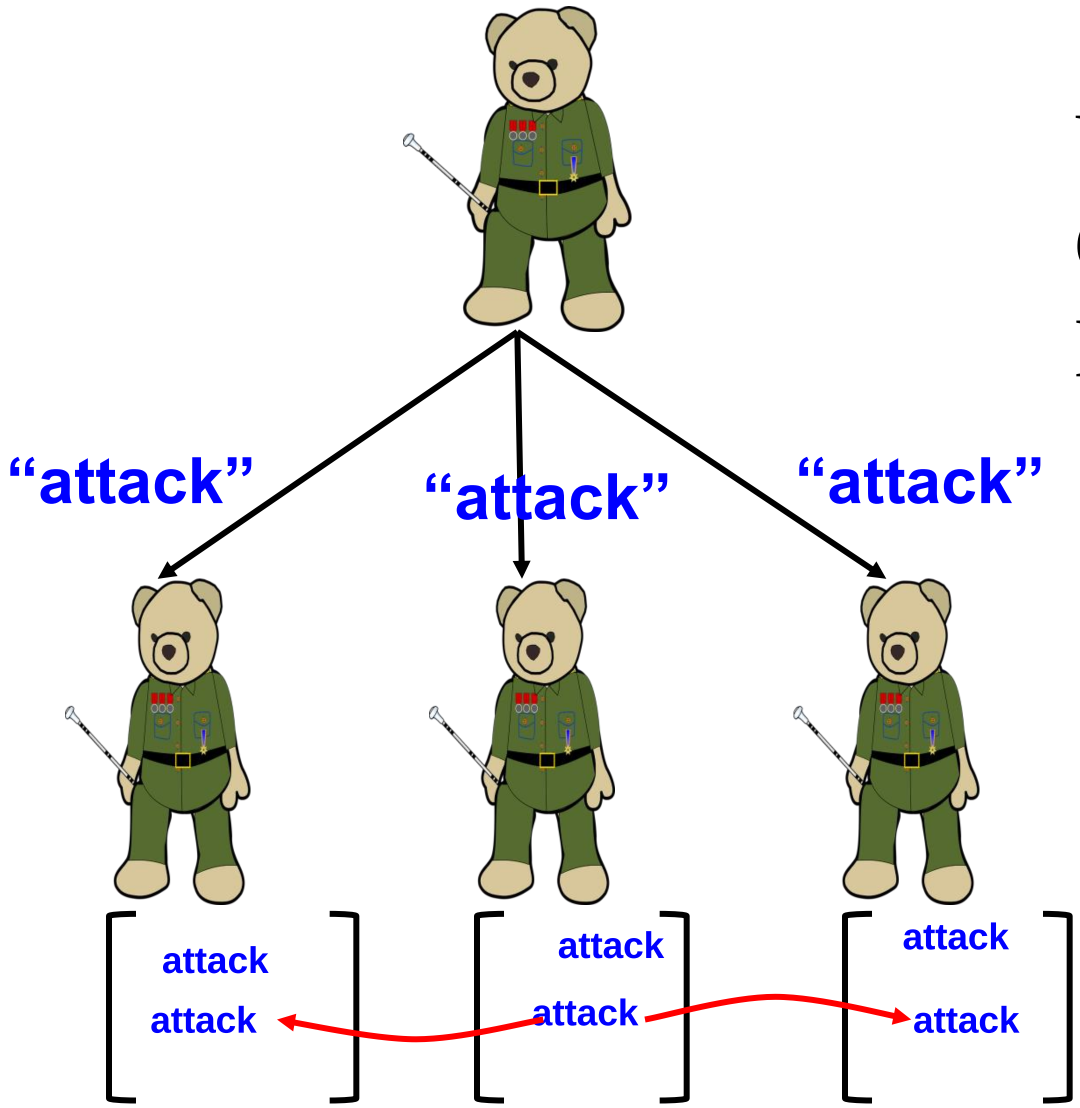
UM(4,m), m>0:

C: Sends order.

L:

1. **Records if received.**
2. **Use UM(3,m-1) to tell others.**
3. **Follows majority() order.**

Inductive Solution for Oral Messages

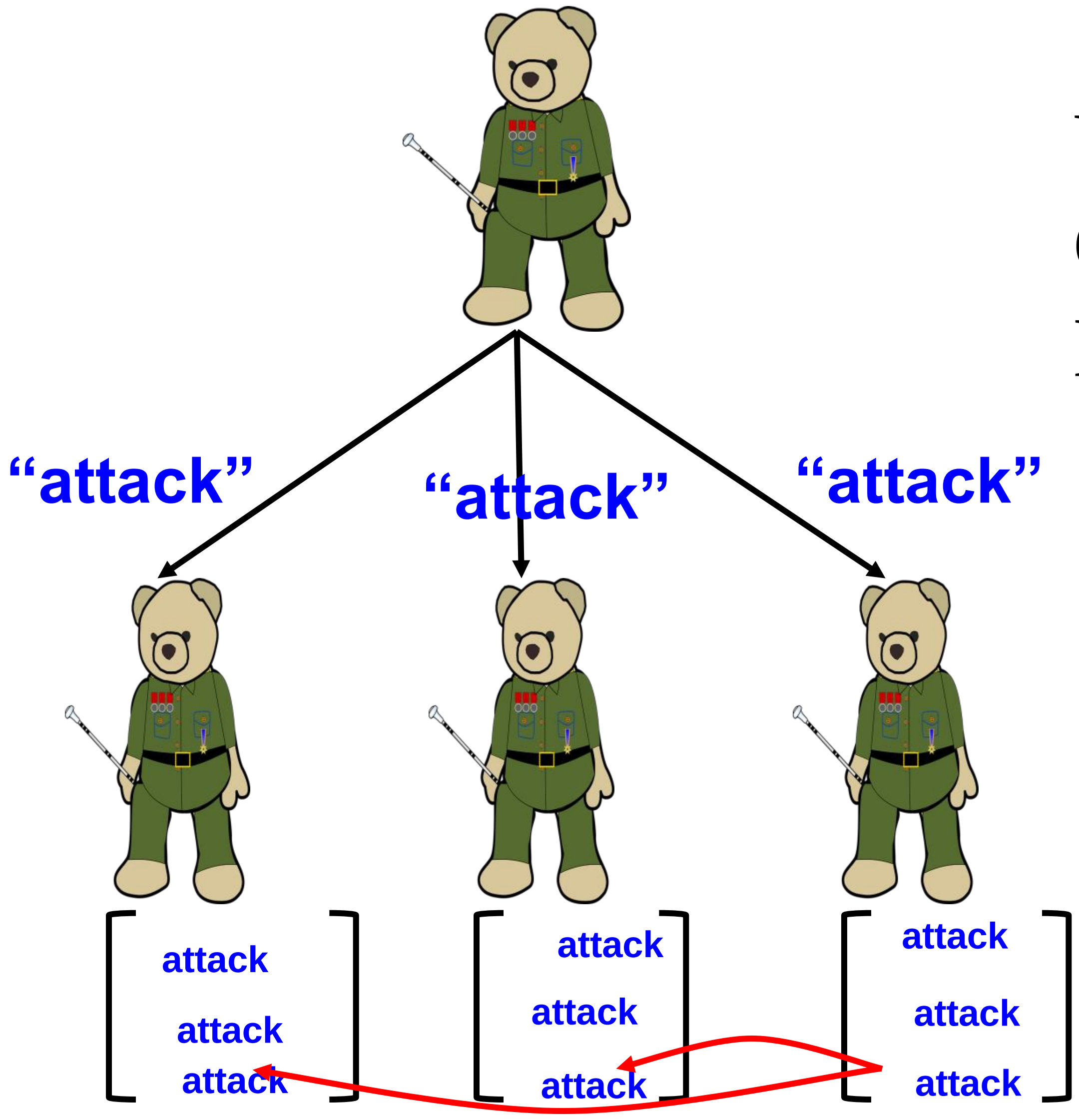


UM(m), m>0:

C: Sends order.

L:
1. Records if received.
2. Use UM(m-1) to tell others.
3. Follows majority() order.

Inductive Solution for Oral Messages



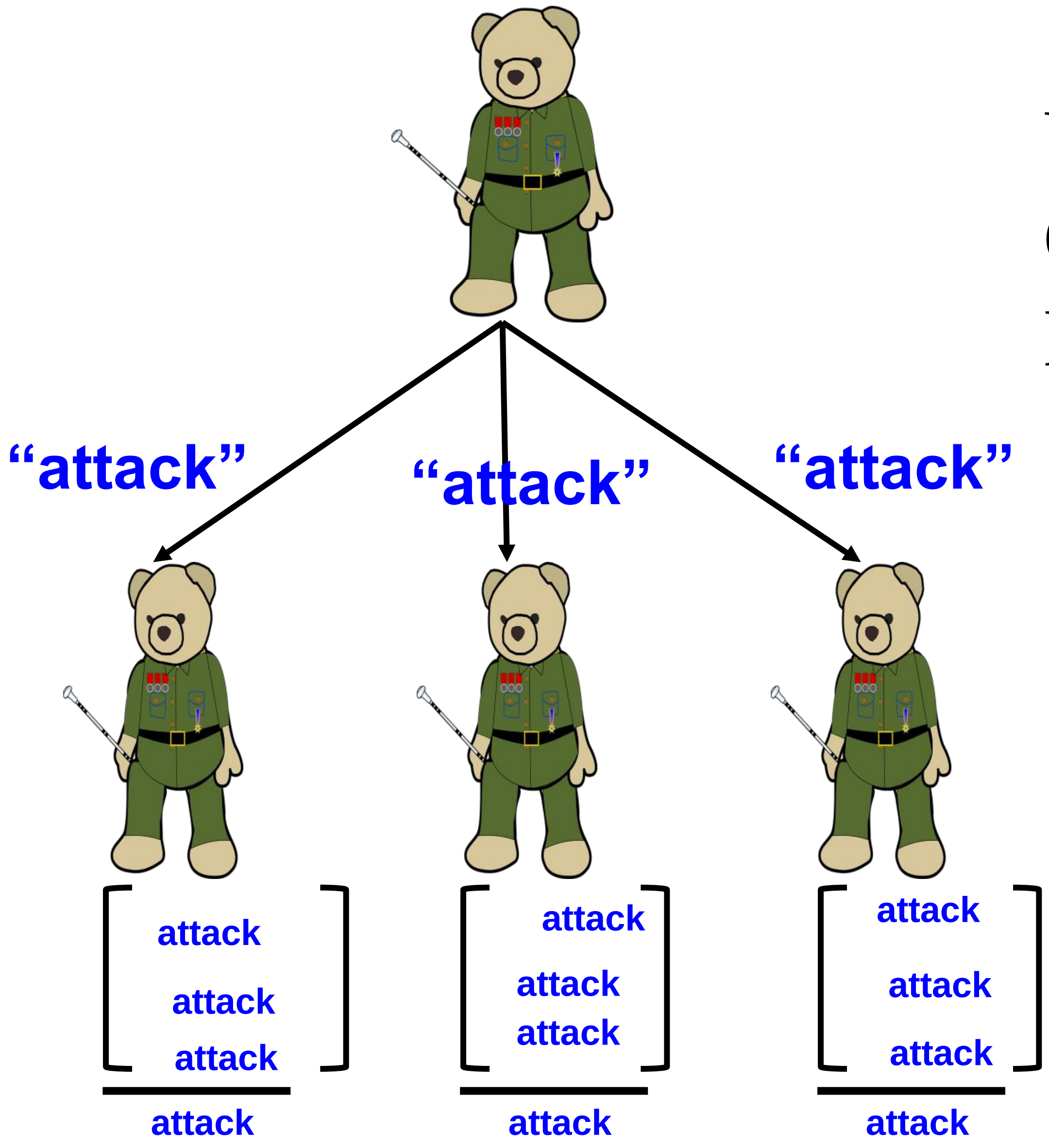
UM(m), m>0:

C: Sends order.

L:

1. **Records if received.**
2. **Use UM(m-1) to tell others.**
3. **Follows majority() order.**

Inductive Solution for Oral Messages

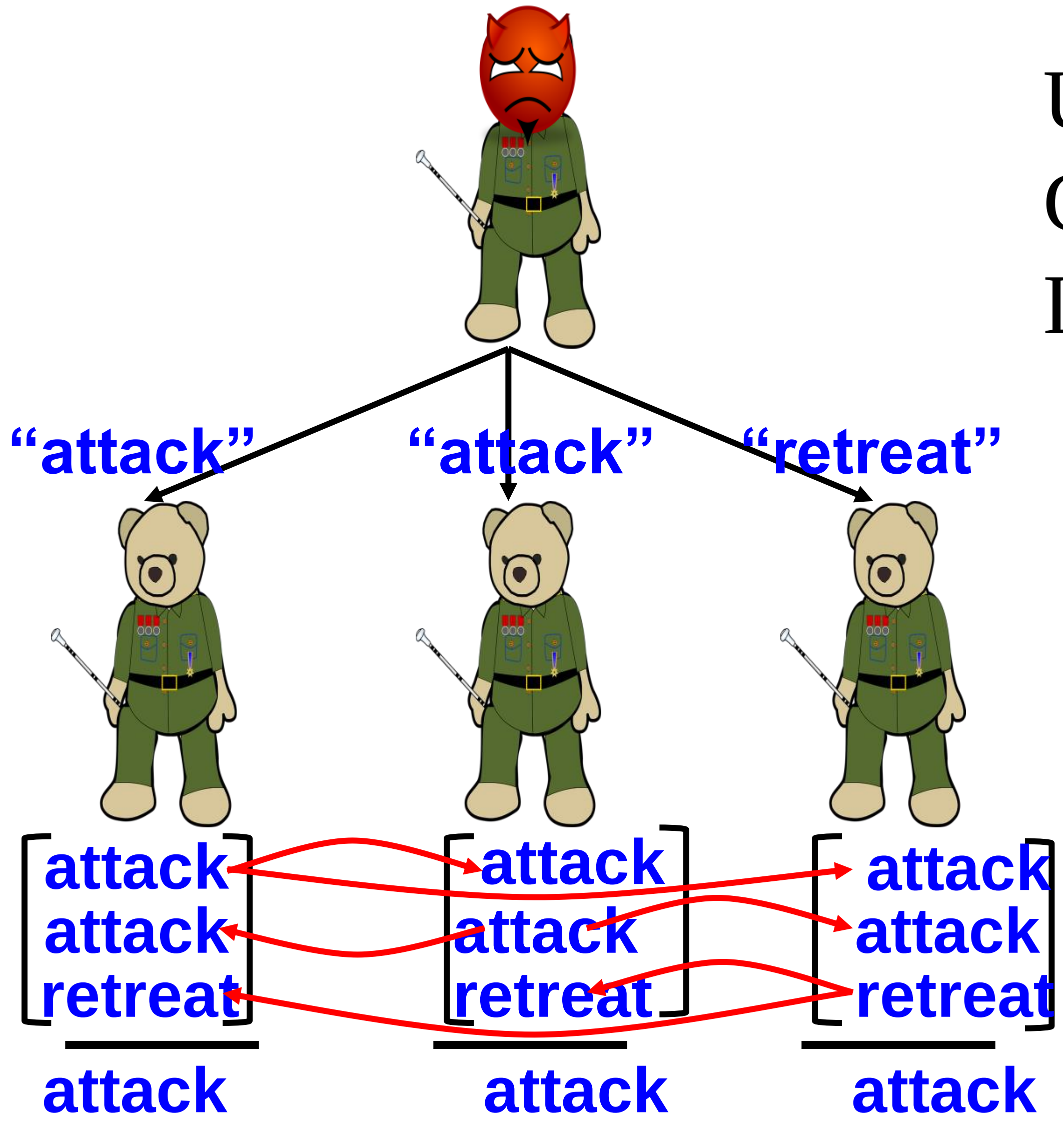


UM(m), m>0:

C: Sends order.

L:
1. **Records if received.**
2. **Use UM(m-1) to tell others.**
3. **Follows majority() order.**

How this works with m=1

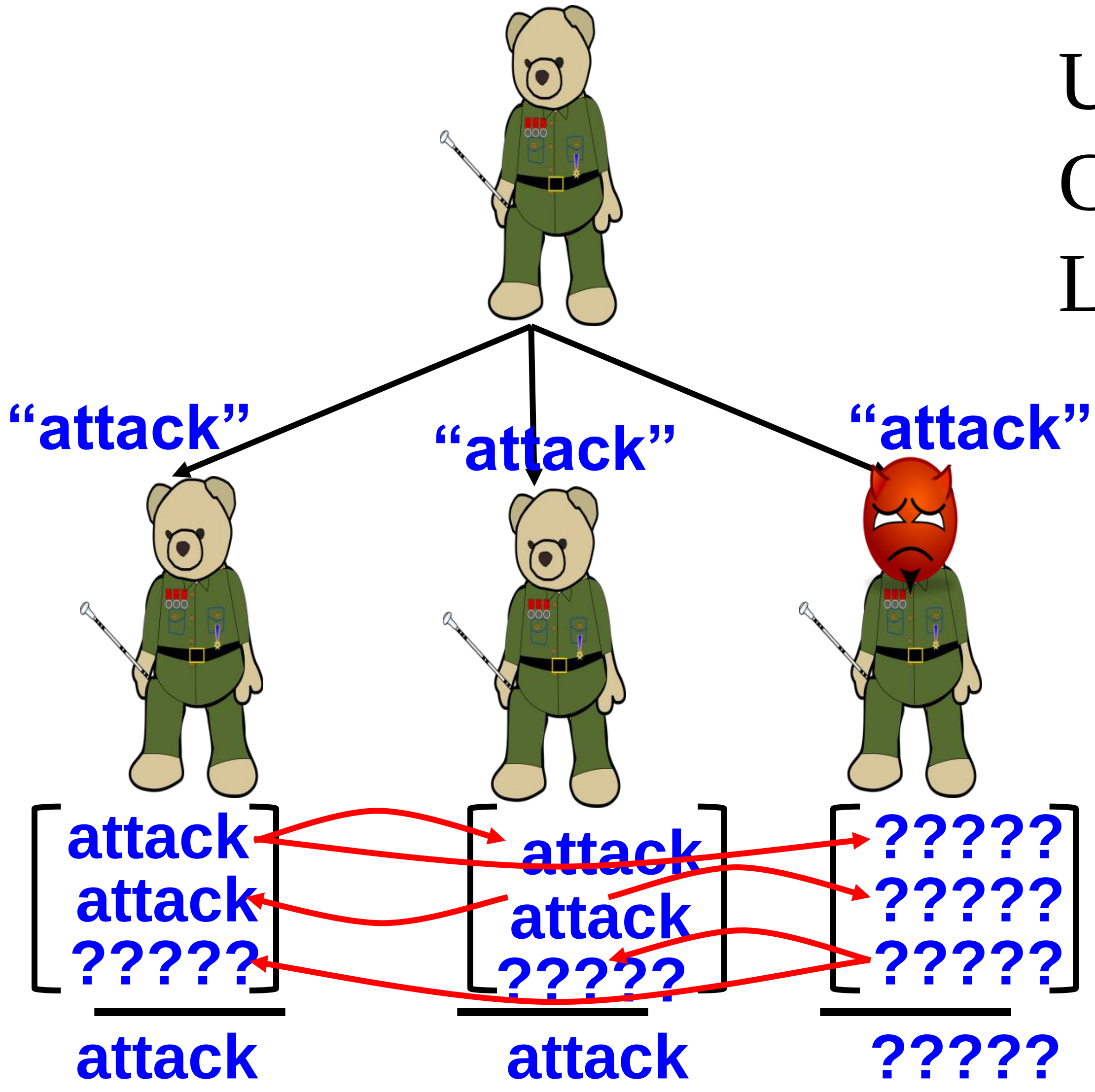


UM(4,m), m>0:

C: Sends order.

L:
1. **Records if received.**
2. **Use UM(3,m-1) to tell others.**
3. **Follows majority() order.**

How this works with m=1

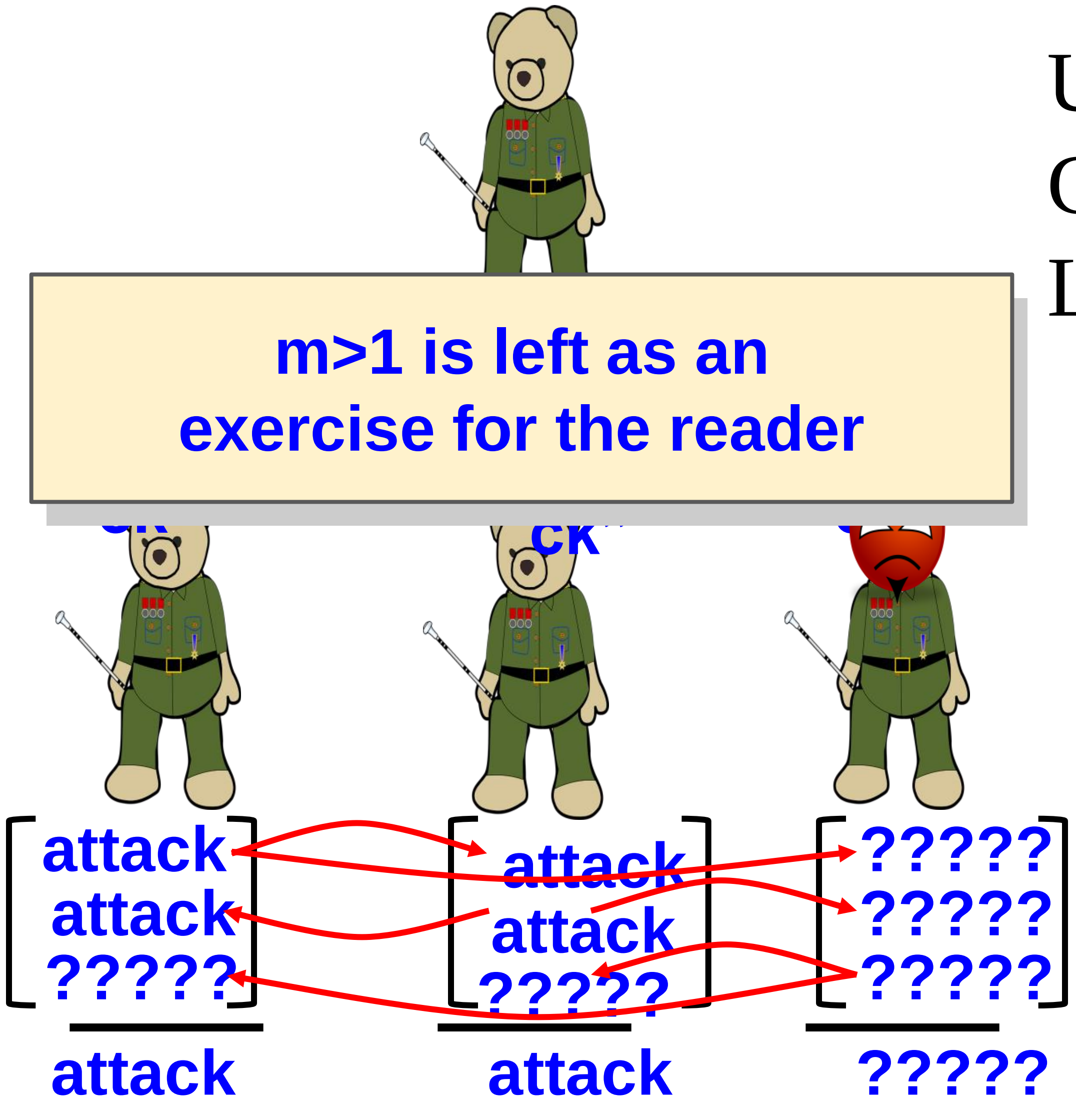


UM(4,m), m>0:

C: Sends order.

L:
1. Records if received.
2. Use UM(3,m-1) to tell others.
3. Follows majority() order.

How this works with m=1



m>1 is left as an exercise for the reader

UM(4,m), m>0:

C: Sends order.

L:
1. **Records if received.**
2. **Use UM(3,m-1) to tell others.**
3. **Follows majority() order.**

Lamport, Shostak and Pease Algorithm

- Let n processes and β failures.
- Processes have a predefined decision o_{def} that is used when the Chief of Staff is a traitor (e.g., retreat).
- We define function majority$(o_1, \dots, o_{n-1}) = o$ that computes the majority of decisions $o_u = o$

Algorithm UM(n,0) (for 0 traitors)

1. The Chief of Staff transmits decision o to all generals.
2. All generals decide o or if they do not receive a message, they decide o_{def}.

Lamport, Shostak and Pease Algorithm

Algorithm UM(n, m) (for m traitors)

1. The Chief of Staff transmits decision o to all generals.
2. For each general u
   - Set o_u to the value received, or if no message received, set to o_{def}.
   - Send the value o_u to the $n-2$ generals by invoking UM$(n-1, m-1)$.
3. For each general u and each $v \neq u$
   - Set o_v to the value received from u at step 2, or if no message received set to o_{def}.
   - Decide on value majority$(o_1, \ldots, o_{n-1})$.

The final theorem

Theorem. If **n > 3m** where **m** is the maximum number of traitors, then **OM(m)** satisfies both **IC1** and **IC2**.

Fischer-Lynch-Paterson [FLP'85] Impossibility Result

- It is impossible for a set of processors in an asynchronous system to agree on a binary value, even if only a single process is subject to an unannounced failure
  - We won't show any proof here – it's too complicated
- The core of the problem is asynchrony
  - It makes it impossible to tell whether or not a machine has crashed (and therefore it will launch recovery and coordinate with you safely) or you just can't reach it now (and therefore it's running separately from you, potentially doing stuff in disagreement with you)
- For synchronous systems, 3PC can be made to guarantee both safety and liveness!
  - When you know the upper bound of message delays, you can infer when something has crashed with certainty