The life of a Bitcoin Miner

Bitcoin Mining

- It is a mechanism to generate hash of the block
- Mining involves creating a hash of a block of transactions that cannot be easily forged, protecting the integrity of the entire blockchain without the need for a central system
- **Bitcoin mining:**

  H(k)= Hash(H(k-1) || T || Nonce)

Here

H(k-1)= Previous block hash

Nonce= miners find this nonce as per the complexity (number of zeros at the prefix)

Cont…

- The number of leading zeros required is the difficulty
- 00091dc94afbd7f451122fa0d646f8b7f004774134c88fb5ab4e93dd1a11393a
- Miners attempt to generate new blocks by solving the complex mathematical equations
- The first miner who is successful in finding a new block is entitled to the block reward and writes the first transaction on the new block they found
- In order for the blockchain network, the newly found block gets added as the new 'unit' on the blockchain
- It is possible that two miners produce a block at a similar time. This situation occurs because the acceptance of the blocks into the blockchain by the nodes of the blockchain network does not happen instantaneously

Cont…

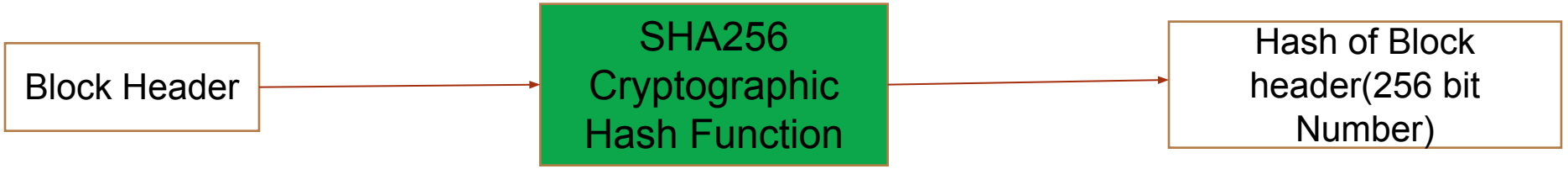
- This time lag in accepting a block may lead to another miner solving for the same exact block
- It leads to a temporary mix-up on the blockchain network, as the nodes try to decide which block of the two newly identified blocks it wants to accept
- In such a situation, the block with the larger share of proof of work (POW) (longest chain of blocks) gets accepted into the blockchain
- The other block, with a smaller proof of work(smaller chain of blocks), is discarded from getting added to the blockchain and is termed as an orphan block
- Such blocks are essentially valid and verified blocks, but due to the network's working mechanism and the lag time leading to delayed acceptance, one of the blocks is rejected, or orphaned

Mining Difficulty

Mining Difficulty

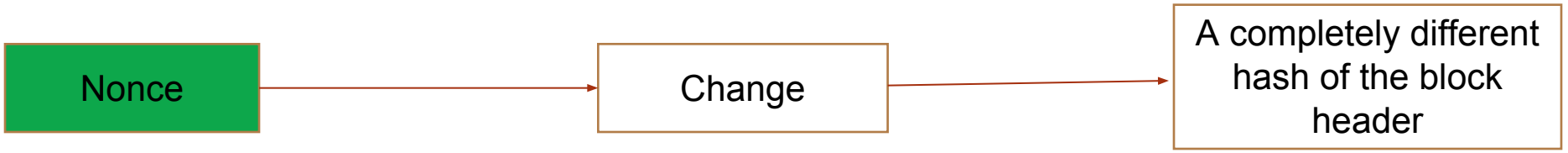
- Block time defines the time it takes to mine a block, in bitcoin the expected block time is 10 minutes
- The expected block time is set at a constant value to make sure, miners cannot impact the security of the network by adding more computational power
- The average block time of the network is evaluated after **n** number of blocks, and if it is greater than the expected block time, then the difficulty level of the proof of work algorithm will be reduced, and if it is less than the expected block time then the difficulty level will be increased
- New_difficulty = old_difficulty x (2016 blocks x 10 minutes) / (the time took in minutes to mine the last 2016 blocks)
- If the average speed of mining the last 2016 blocks is 8 minutes — then the new difficulty factor will be greater than one, so the current difficulty level will be increased
- In case — the average is above 10 minutes, then the factor will be less than 1 and the difficulty level will be decreased for the next 2016 blocks
- The difficulty level is reevaluated after every 2016 blocks, that's roughly after every 2 weeks.

Block Confirmation: Proof of Work



- Current target (T_{cur}) :Bits" field
  - Maximum target(T_{max}) :
    0x00000000FFFF0000000000000000000000000000000000000000000000000000
- **Condition of Block confirmation**
  - Hash of block header T_{cur}
- Block Difficulty (D): $D = T_{max} / T_{cur}$ (2016 Blocks/ every 2 week)

- Which hash validate the block?



- SHA 256 choose any 256-bit number from 0 to 2^{256}

Block Validation Probability

- 0x00000000FFFF0000000000000000000000000000000000000000000000000000
  16 bits 208 bits
  T_{max}
- The offset for difficulty 1 is $(2^{16}-1)2^{208}$ and for difficulty D is $\frac{(2^{16}-1)2^{208}}{D}$
- The expected number of hashes we need to calculate to find a block with difficulty D is
  $2^{256} = \frac{2^{256}}{\frac{(2^{16}-1)2^{208}}{D}} = \frac{2^{48}D}{(2^{16}-1)2^{208}} = \frac{2^{32}D}{(2^{16}-1)}$
- Every hash has a probability of $\frac{1}{2^{32}D}$ to validate a block

Bitcoin Mining

- It h is a hash rate and t is a mine time, on average the number of found blocks is

  $N = (t*h)/ 2^{32}D$

- D=difficulty, h=miner's hash rate

- Example- Amar buys a mining computer with h=1Ghash/s= 10^9 hash/s.If he mines for a day(86,400s) when D=1690906 and B=50BTC

- Found blocks= $(t*h)/ 2^{32}D$ =0.0119 blocks=0.119*B=0.595 BTC

- Classification of mining
  - Solo Mining: Mining alone
  - Pooled Mining: The mining pool means a state of miners coming together collectively and they are trying to mine a new block

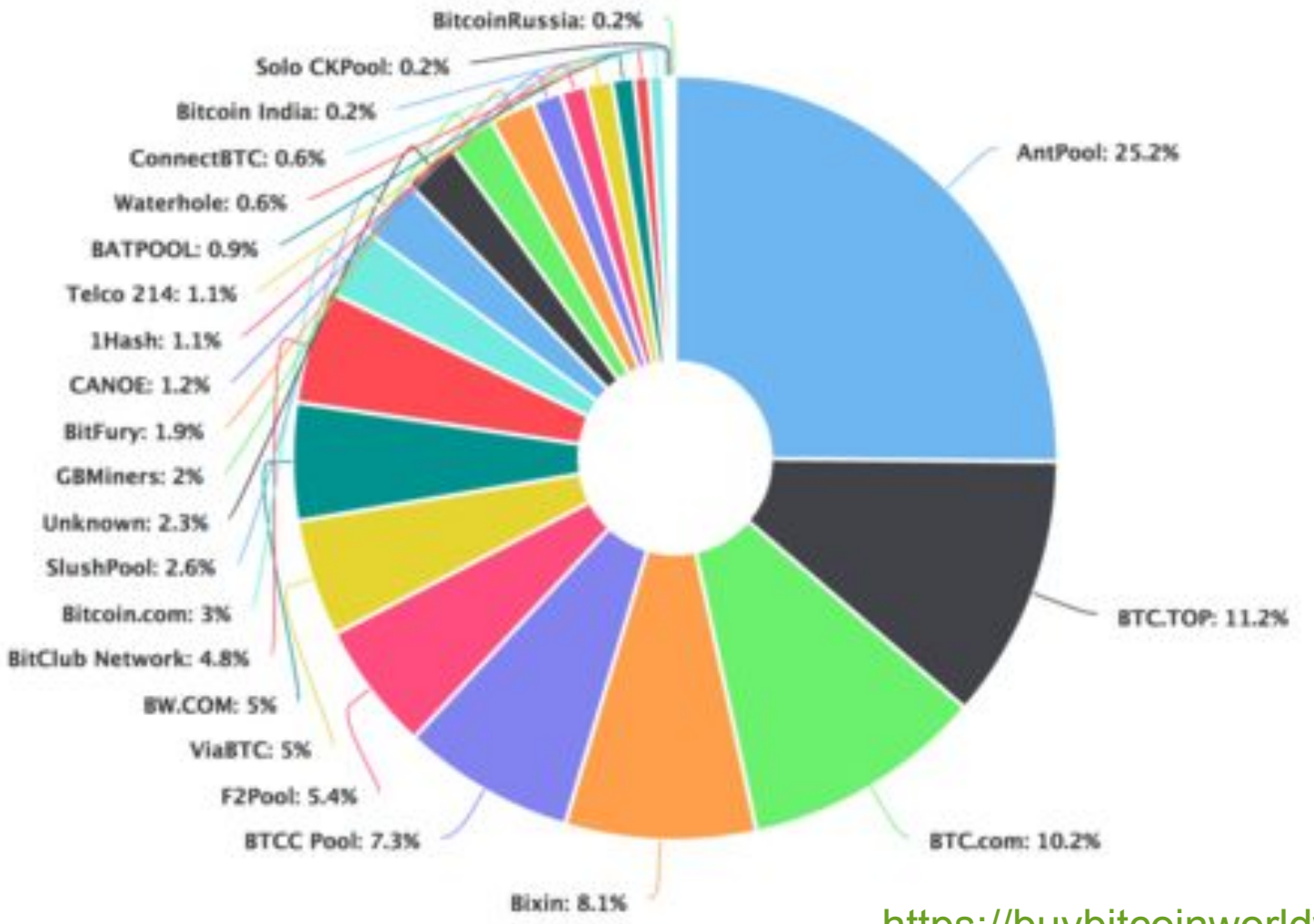
Mining Pool

Mining Pool

- Mining pools are groups of cooperating miners who agree to share block rewards in proportion to their contributed mining hash power
- The mining pools are operated centrally or are designed in a p2p way
- Some of the mining pools charge fees for their services
- The expected revenue from pooled mining is slightly lower than the expected revenue from solo mining

E.g. if the operator got 25 BTC from mining then he will share 25 BTC-fee among them (and keep the fee to himself)

Popular Mining Pools



https://buybitcoinworldwide.com/mining/pools/
03-Aug-2022

How to design a mining pool?

This includes a coinbase transaction transferring the reward to pk

Miner

Mining pool operator

Tries to find nonce such that H(nonce, H(Bi), Ti) starts with n zeros

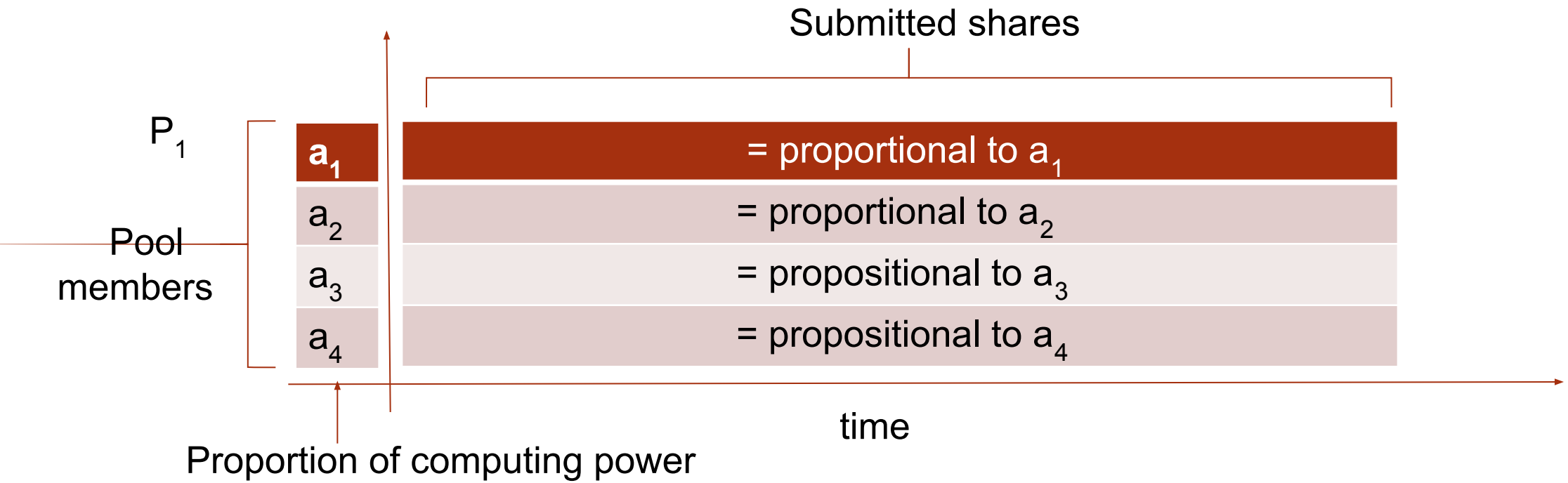
Current hardness parameter

Problem

How to verify how much work a miner really did?

Once nonce is found by some of the pool members each of them is rewarded proportionally to his work

Works if the miners don't change the pools



Probability of that this pool wins: $a_1+a_2+a_3+a_4$

Reward for P1 in case it wins: BTC 25* $a_1/(a_1+a_2+a_3+a_4)$ → Expected reward For P_1:BTC25 * a_1

Pooled Mining

- Joint effort and reward distribution
- H= Total hash rate of all miners
- Single miner's hash rate h=qH (0<q<1)
- $E[P_p]$= Total average payout of the pool

$$E[P_p] = \frac{H*t*B}{2^{32}D}$$

- $E[P_s]$= Single miner's payout in pooled mining

$$E[P_s] = q*(H*t*B/\ 2^{32}D) = (h/H)*(H*t*B/\ 2^{32}D) = (h*t*B/\ 2^{32}D)$$

- $V[P_s]$= Single miner's variance in pooled mining

$$V[P_s] = q^2 * H *(t*B/\ 2^{32}D\ ) = q*(h*t*B\ /\ 2^{32}D\ )$$

Pooled Mining

- f= Fee/Block, B=Block reward
  - Operator's fee for a block= f*B
- Actual reward for the pool miners=B-f*B=(1-f)B
- In a pool
  - Each miner submits shares into the pool
  - **Share:** Hash of a block header calculated by a miner which is less than T_{cur} assuming D=1 (e.g. $T_{cur} = T_{max}$)
- Each hash has a probability of $1/2^{32}$ to be a share in the pool
- Each share has a probability p= 1/D to validate a block
- For a single share, a miner's
- **Expected payout= Expected contribution to total reward= pB**

Byzantine General's Problem

Faults in a Distributed System

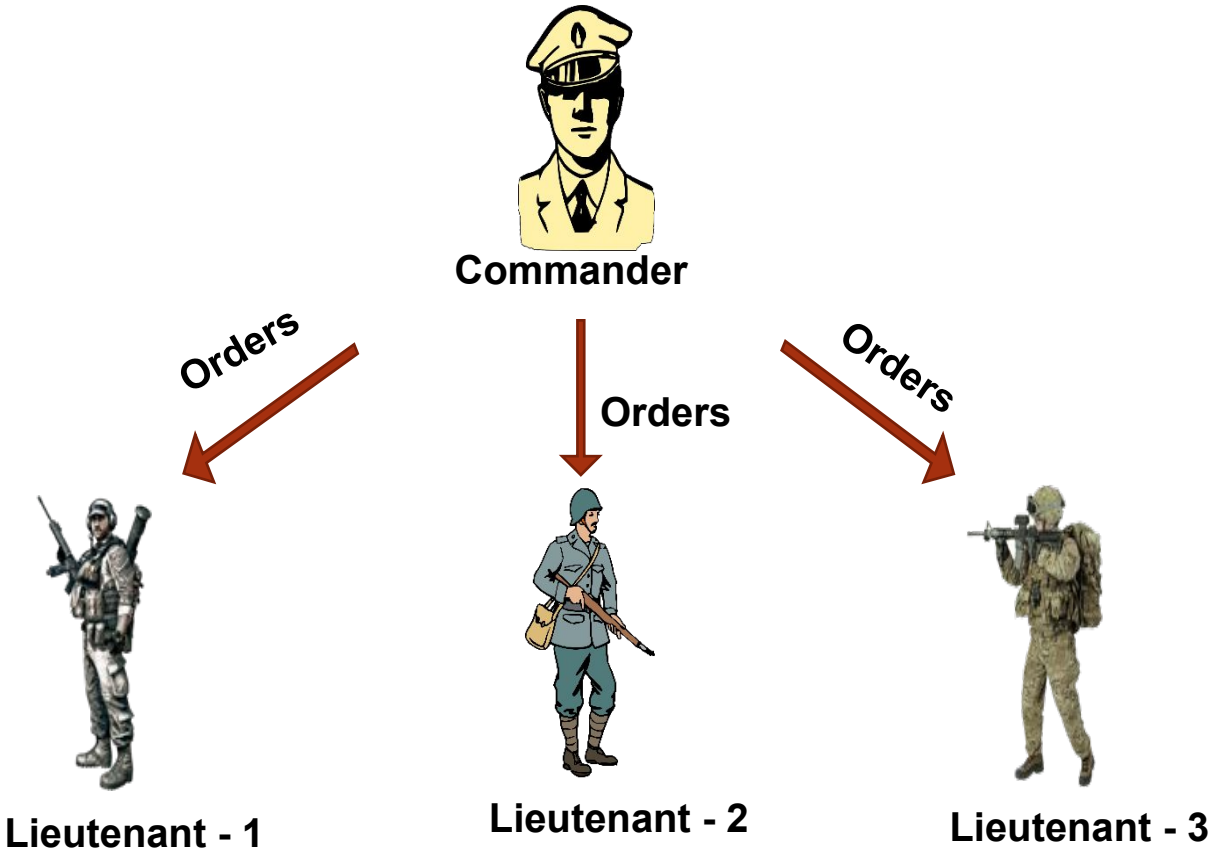
- **Crash Faults:**
  - The node stops operating
    - Hardware or software faults
  - In an asynchronous system
    - We do not know whether messages have been delayed or the node is not responding
  - Rely on majority voting
    - Progress as and when we have received the confirmation from the majority
  - Propagation of the consensus information
    - Nodes on a slow network will receive it eventually

Faults in a Distributed System

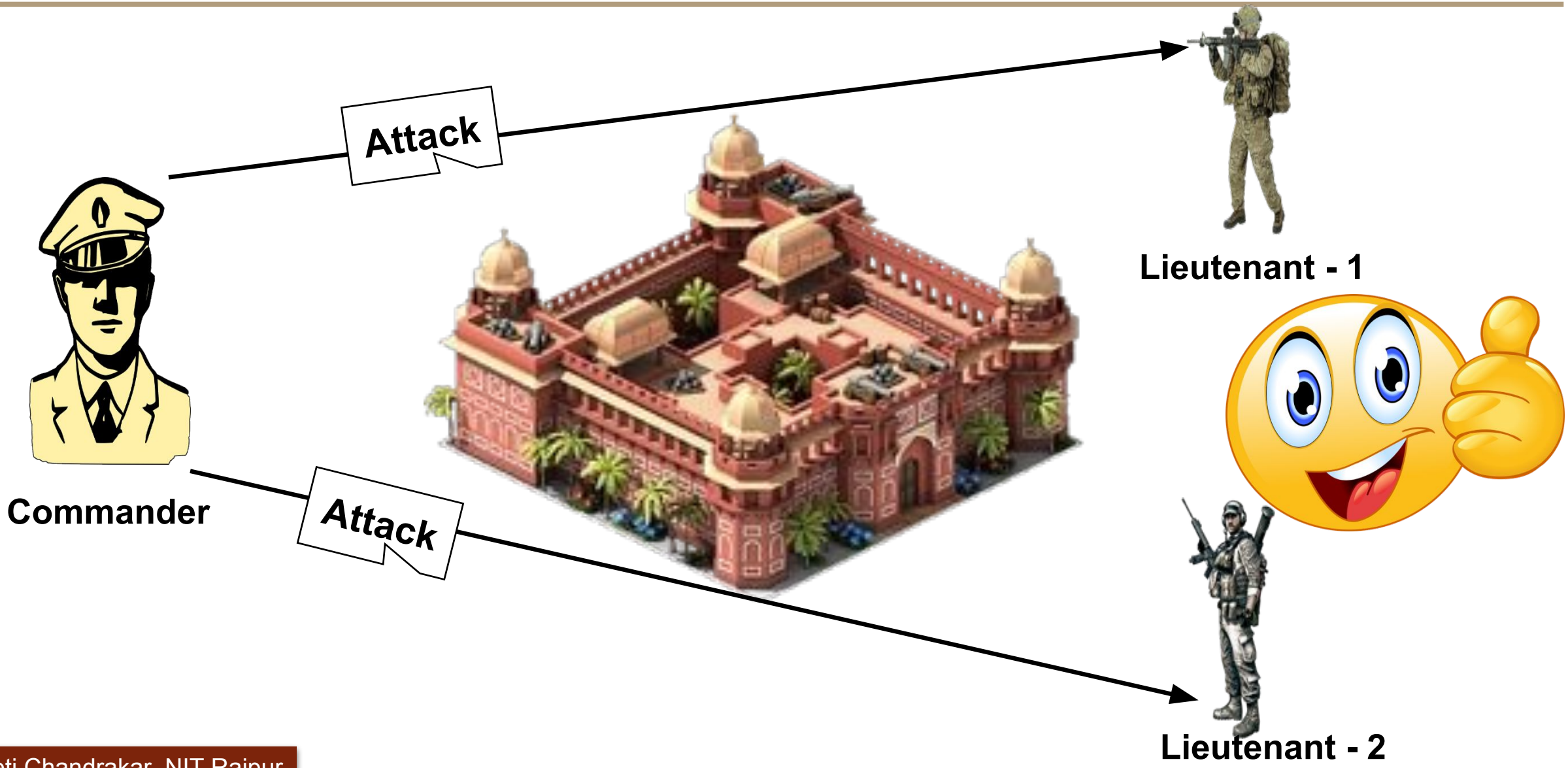
- **Byzantine Faults**
  - Everything is fair for nodes with Byzantine faults(lying, colluding, etc.)
  - They can
    - Show up as crash failures
    - Fake and forge messages
    - Even get together with other faulty nodes and create confusion
  - Nodes misbehave- send different information to different peers(partition the network)
  - More difficult to handle
  - More suitable for blockchains

Byzantine General's Problem

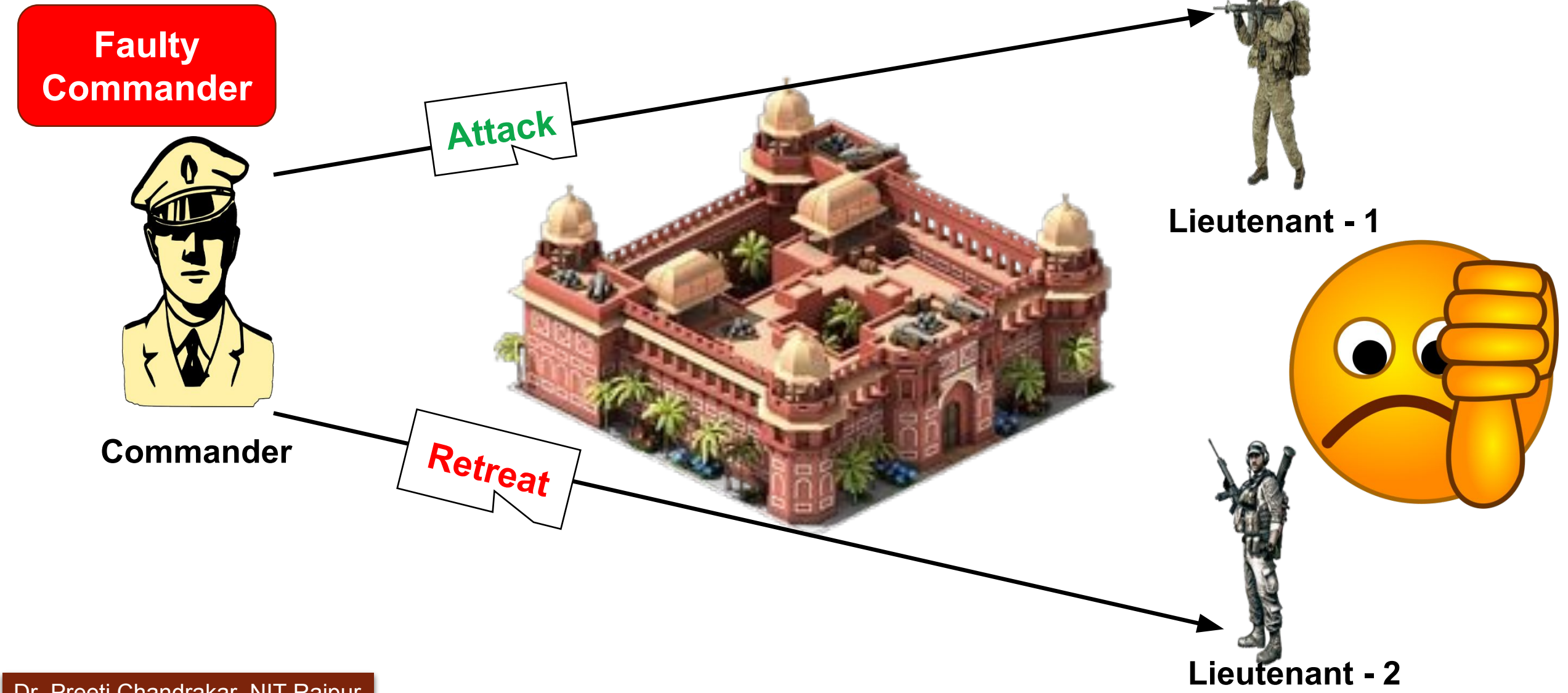
- The commander or any of the lieutenant generals can be faulty/ disloyal
- Can have Byzantine failures
- We cannot trust them at all
- The commander sends an order to n-1 lieutenant generals
- Condition 1: All loyal lieutenant generals obey the same order
- Condition 2: If the commander is loyal, every loyal general obeys the order that the commander issues



Byzantine General's Problem



Byzantine General's Problem



Byzantine General's Problem

Faulty Commander

Attack

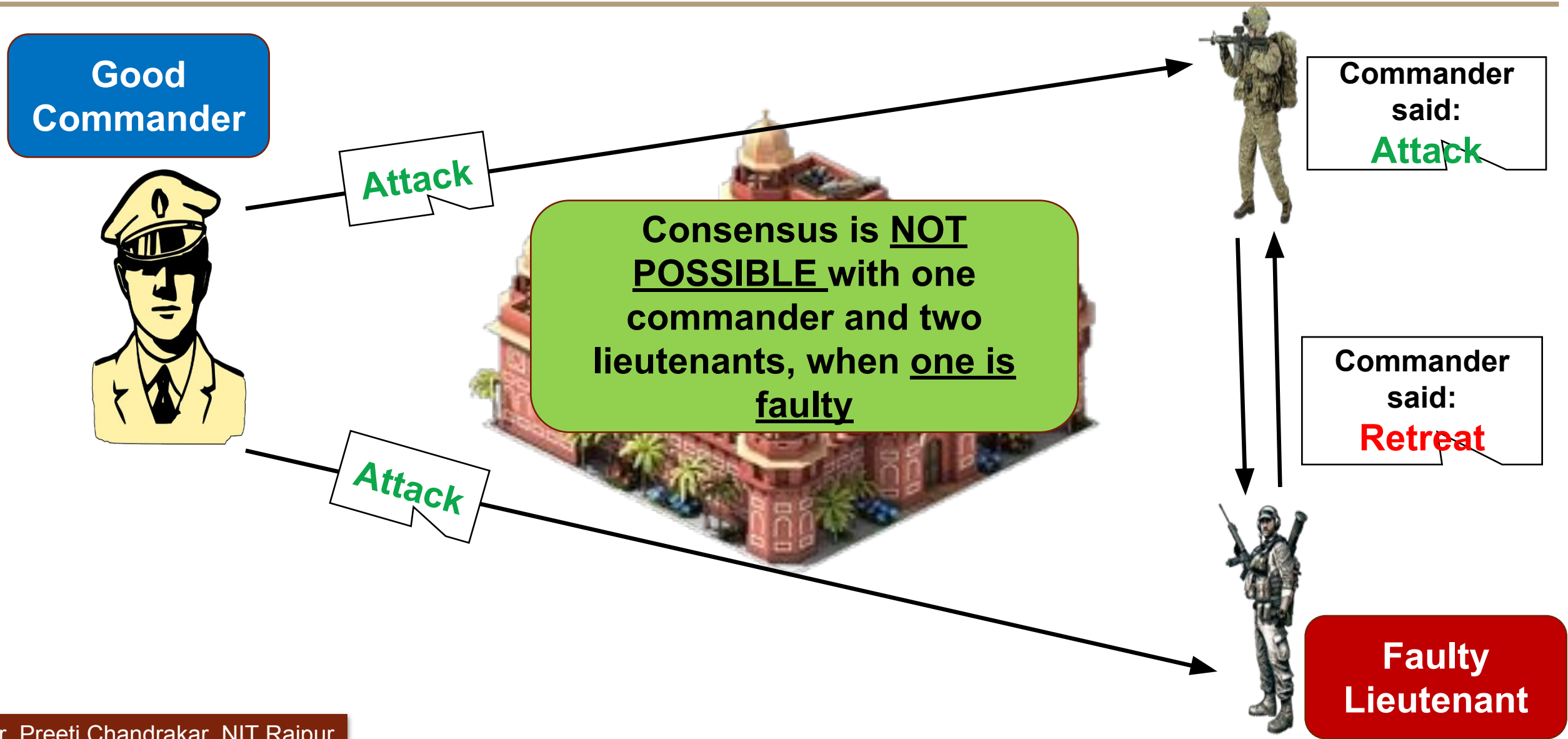
Retreat

Commander said: Attack

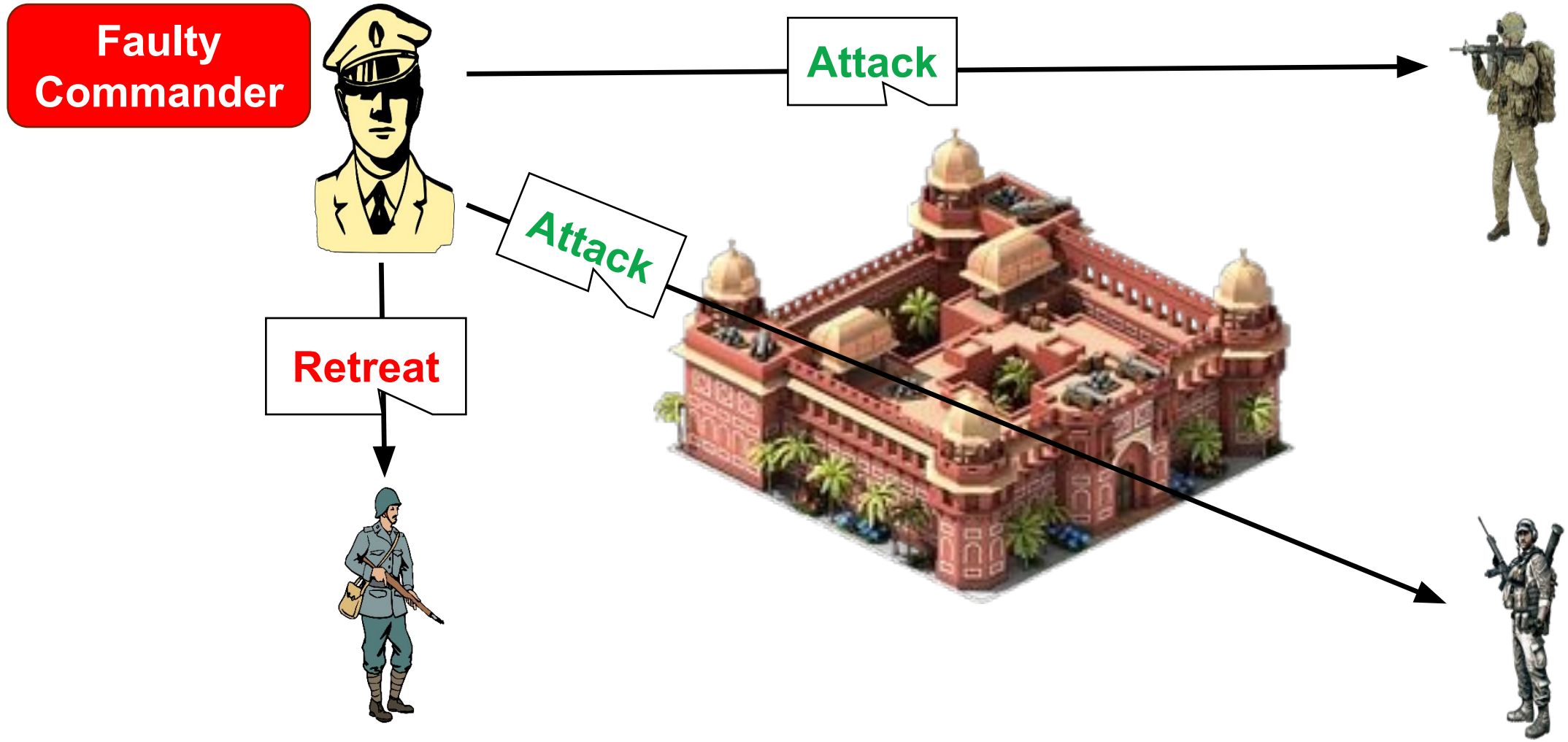
Can we reach to consensus ?

Commander said: Retreat

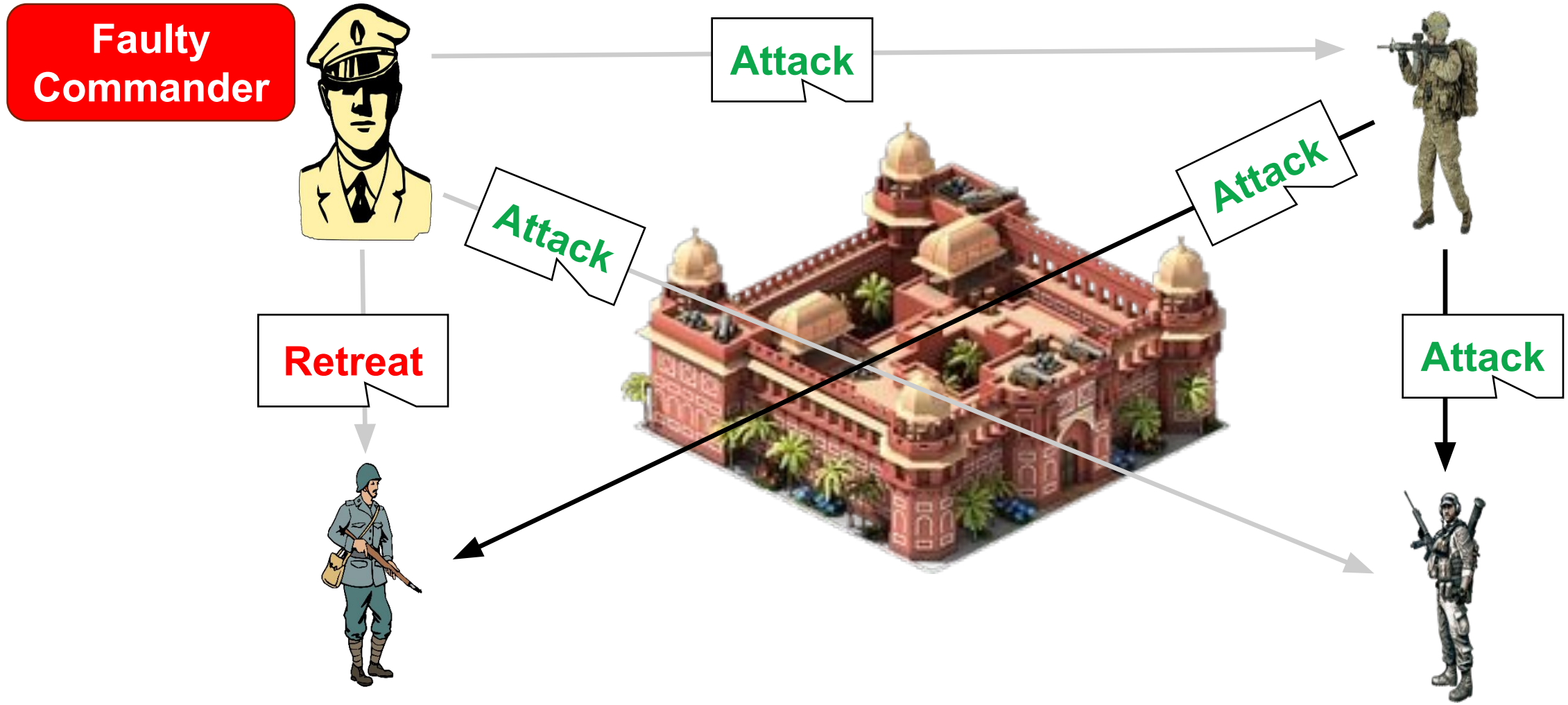
Byzantine General's Problem



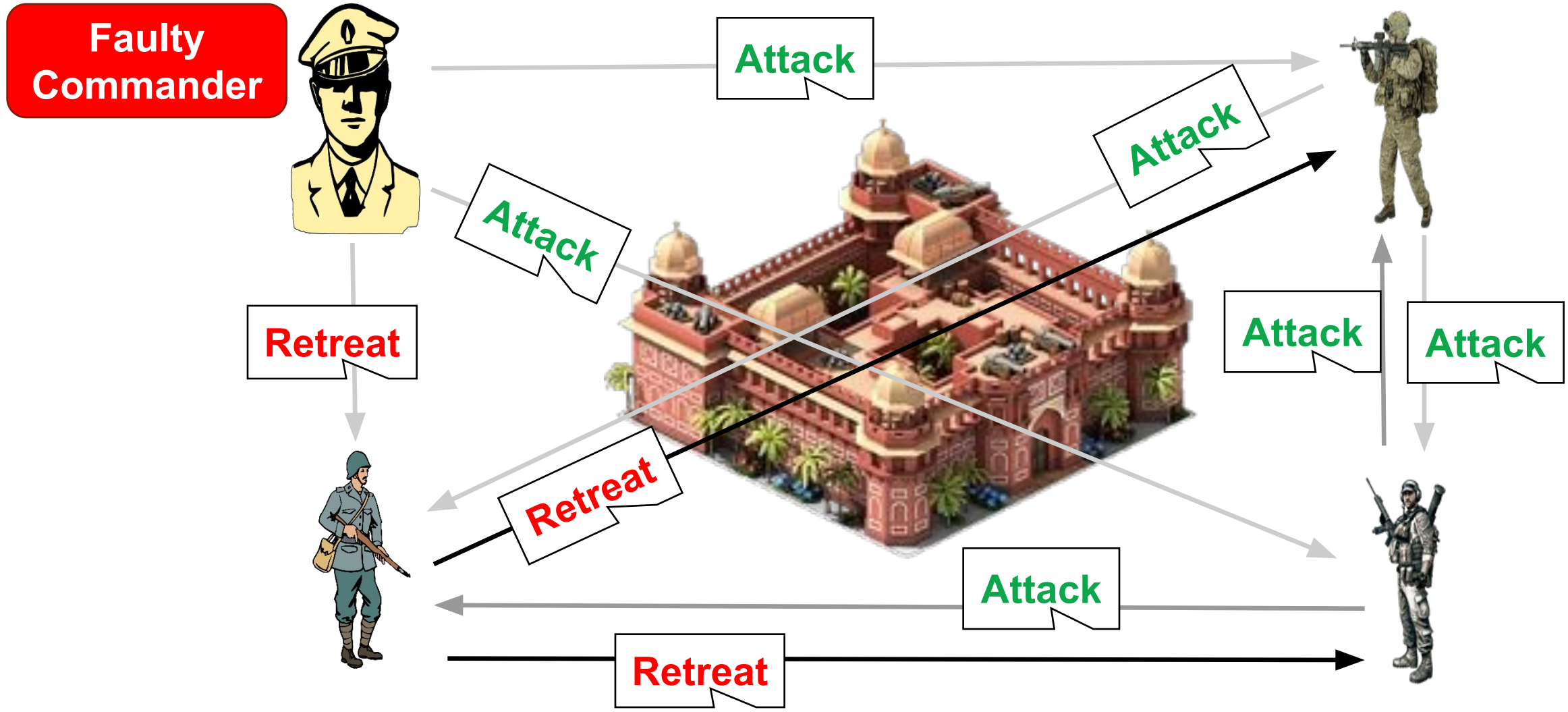
Byzantine General's Problem- Three Lieutenants



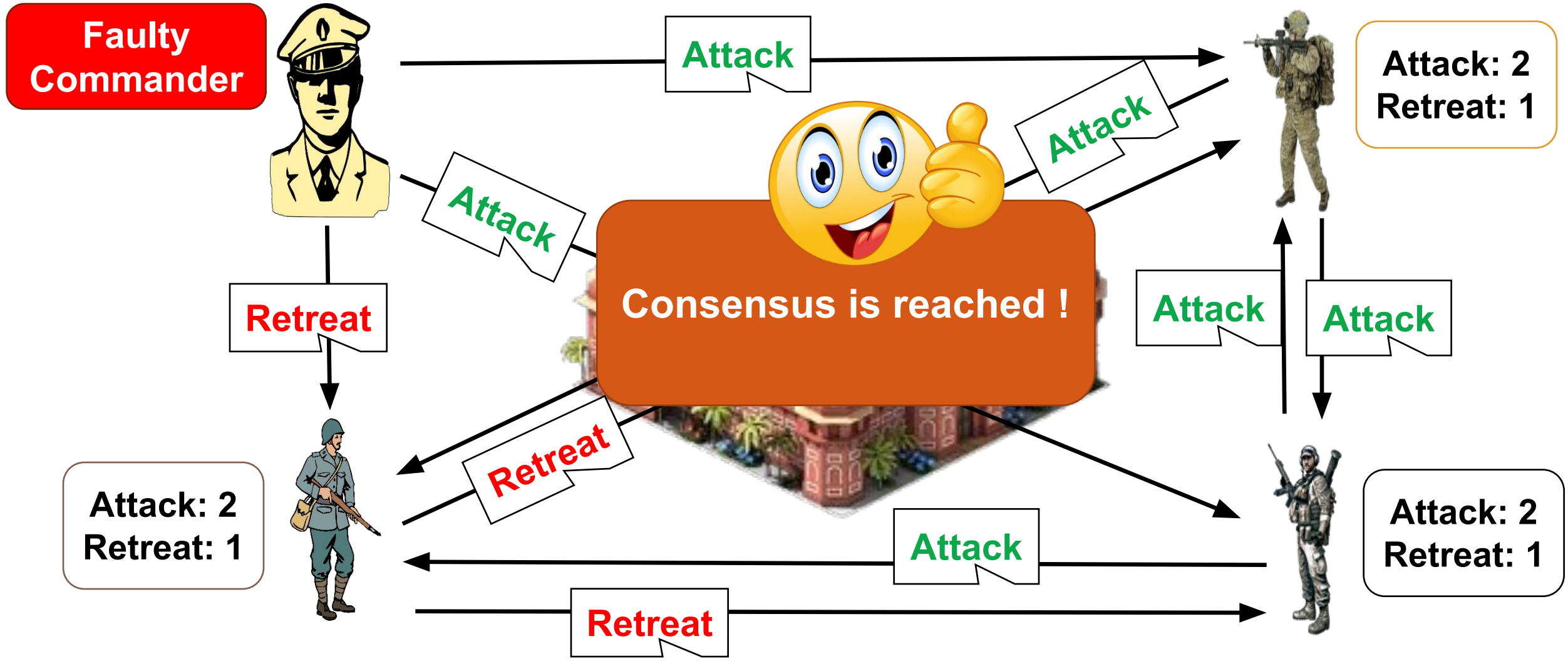
Byzantine General's Problem- Three Lieutenants



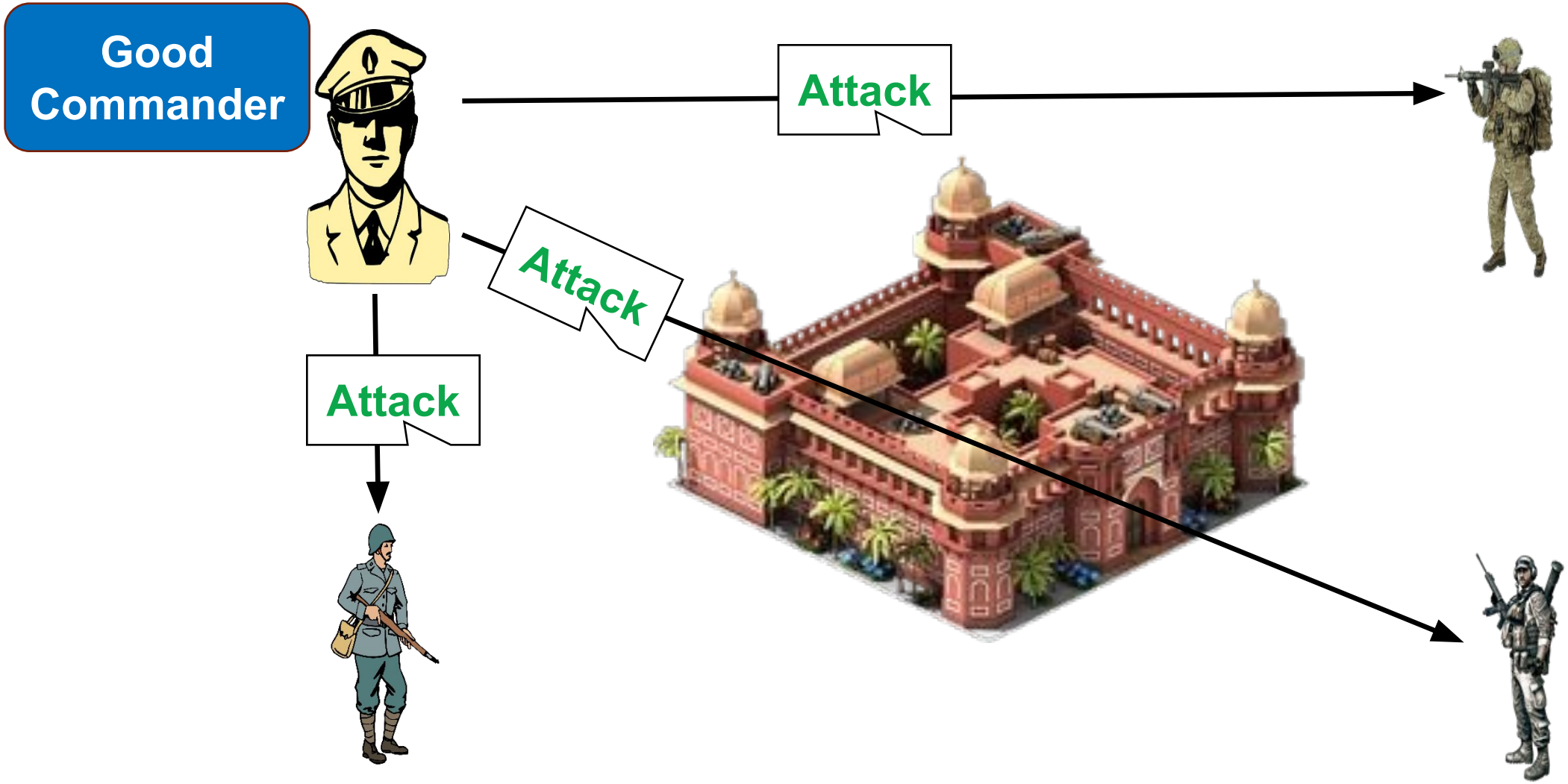
Byzantine General's Problem- Three Lieutenants



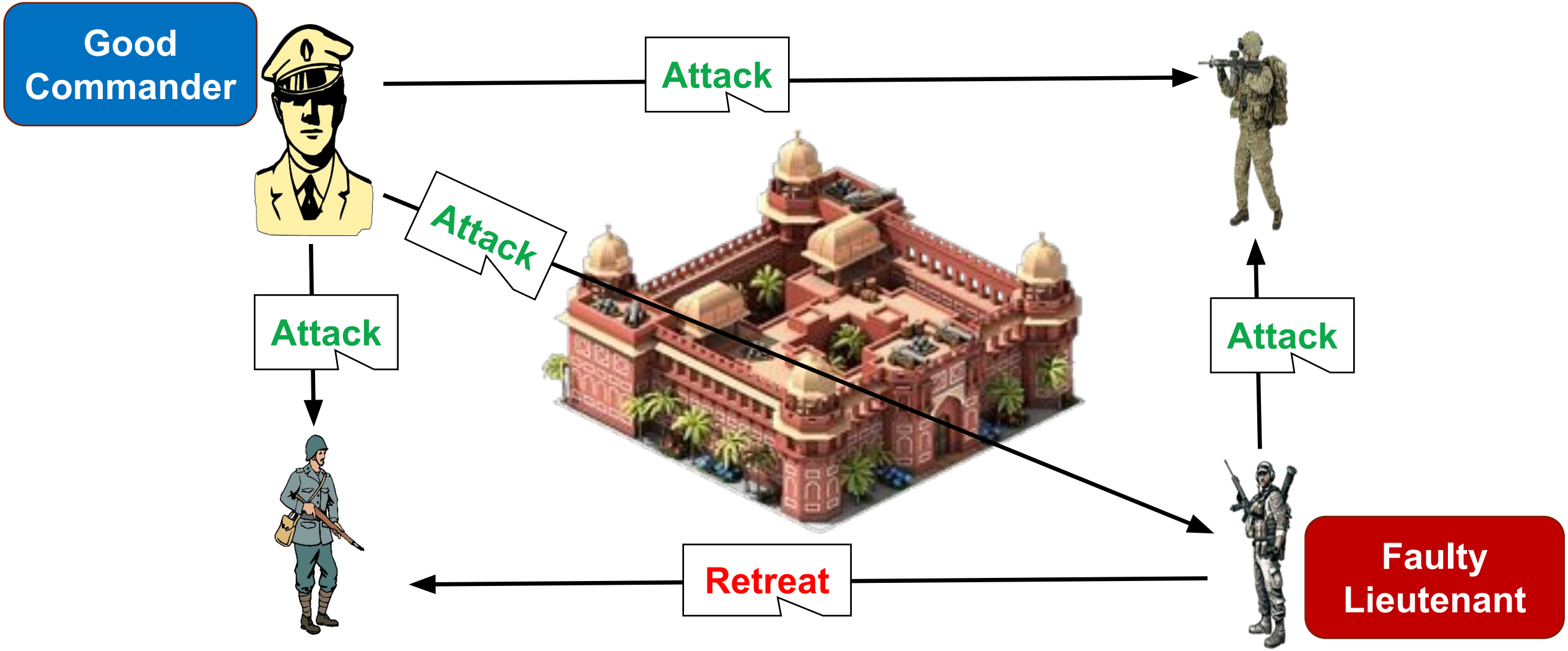
Byzantine General's Problem- Three Lieutenants



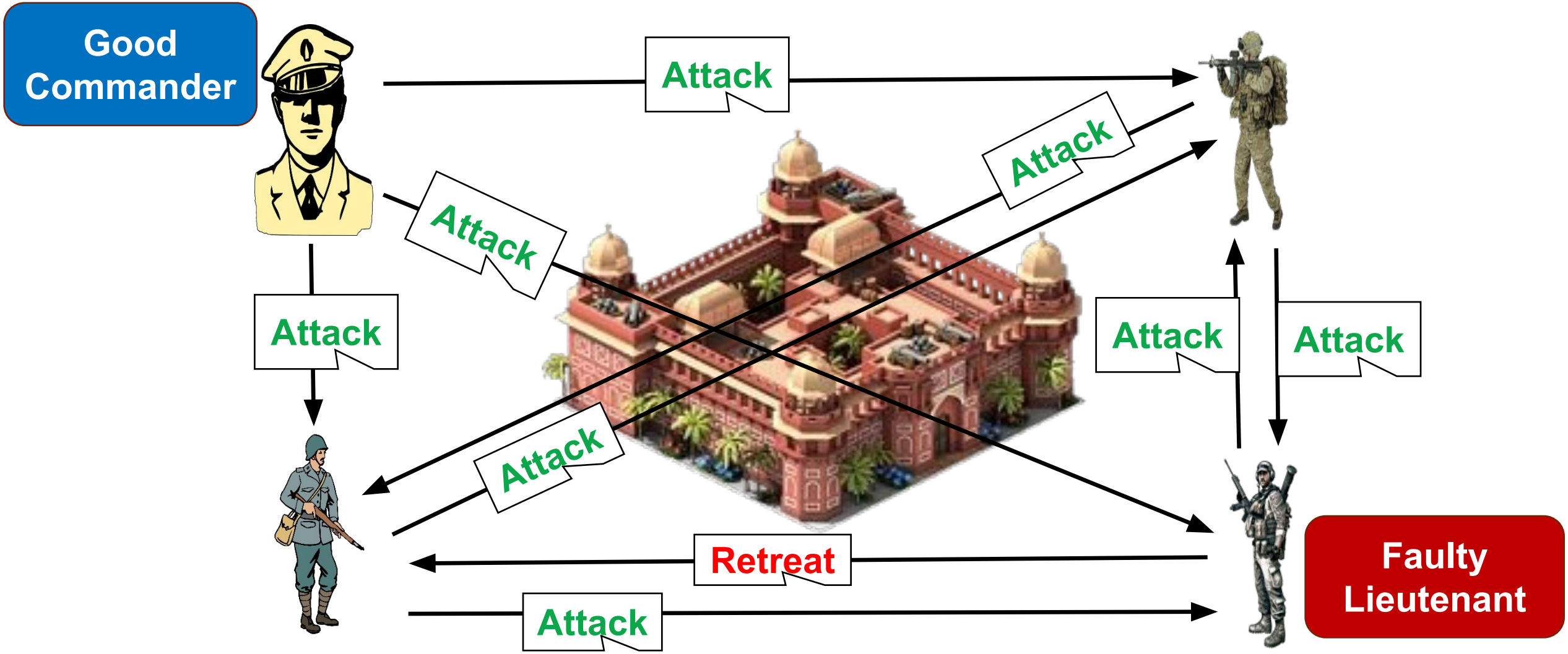
Byzantine General's Problem- Three Lieutenants



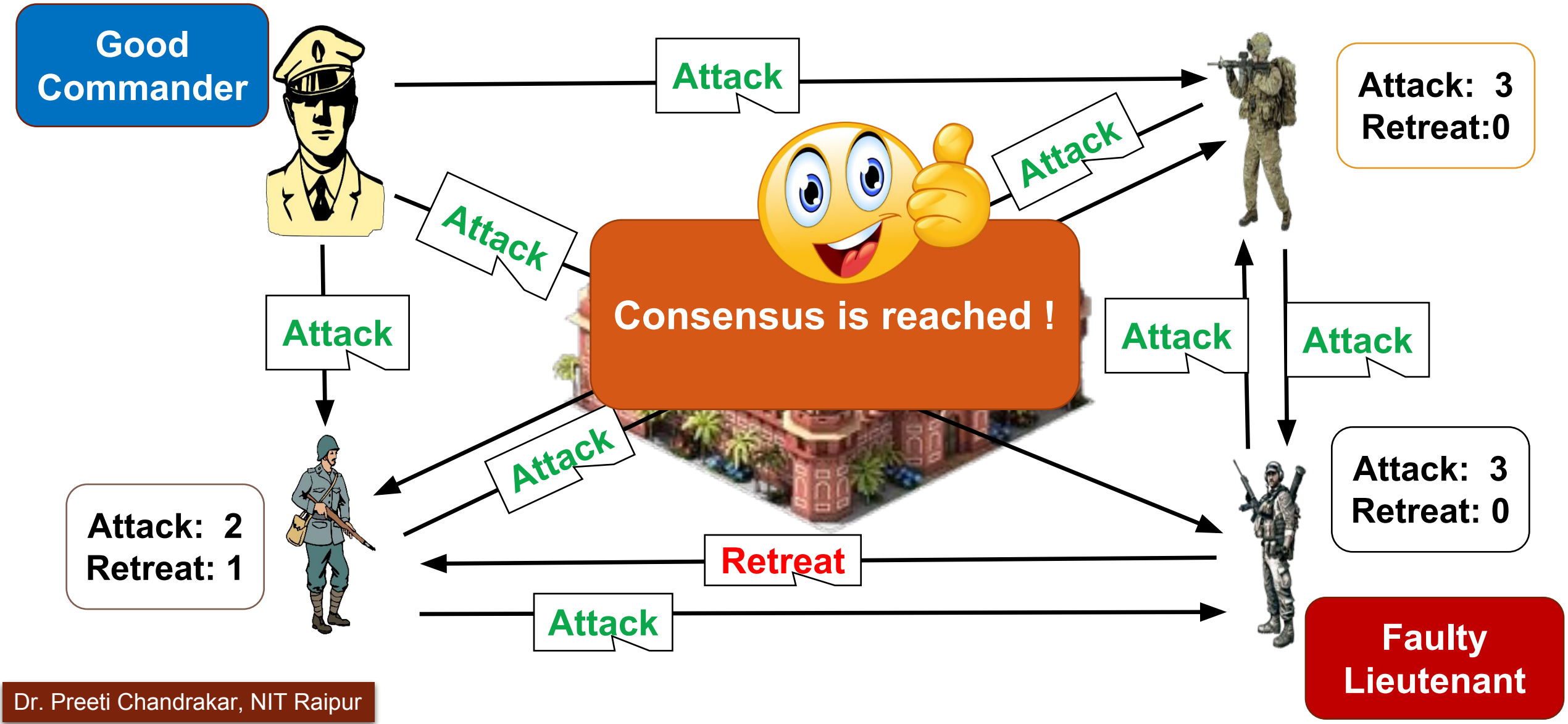
Byzantine General's Problem- Three Lieutenants



Byzantine General's Problem- Three Lieutenants



Byzantine General's Problem- Three Lieutenants



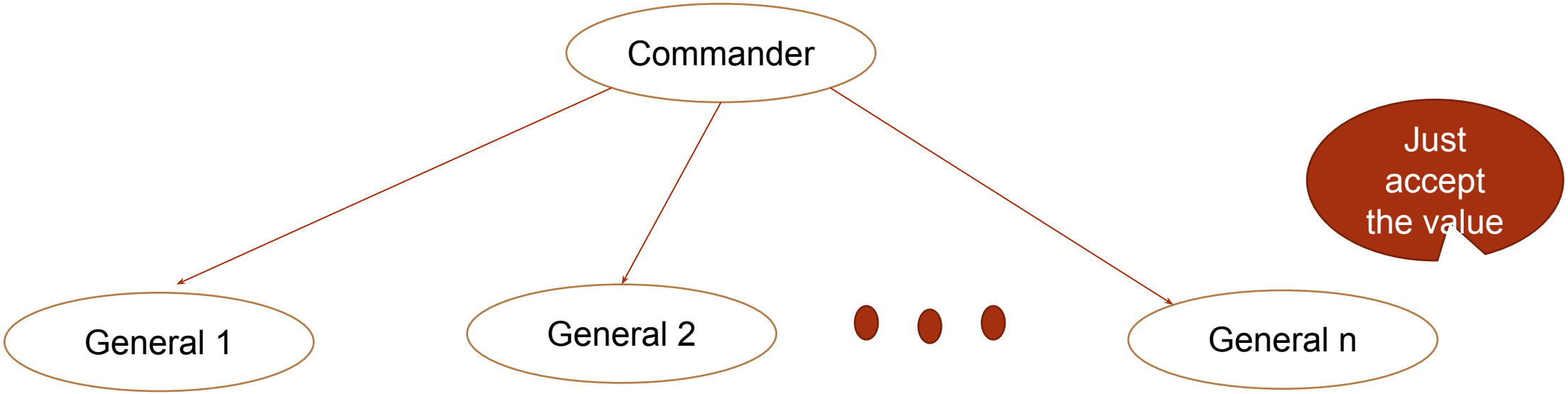
Byzantine Agreement Algorithm

Assumptions

- **A1:** Every message is delivered correctly. Message integrity is maintained
- **A2:** The receiver of a message knows the identy of the sender
- **A3:** The absence of a message can be detected
- **A4:** A majority function(V_1, V_2……V_n) that returns the majority value. If the majority does not exist, return RETREAT

Algorithm: Step 1

- **OM(0)**
- The commander sends its value to each lieutenant
- Each lieutenant accepts the value or accepts RETREAT if no message was sent



Note: OM name of algorithm
OM(0): means there is no faulty member

Algorithm: Step 2

- **OM(m), m>0, total n generals including the commander**
  - The commander sends his value to every lieutenant
  - Let lieutenant i, receive value v_i from the commander
    - How does it know that the same value was sent to other lieutenants?
    - It acts as a commander in an algorithm with the rest of the n-2 lieutenants OM(m-1): Sends them value v_i, and tries to arrive at an agreement
    - The loyal generals in OM(m-1) come to an agreement about the value received by lieutenant i
    - It does not matter if lieutenant i is loyal or not. The rest of the loyal generals agree on the value that i got from its commander after OM(m-1)
- (n-1) lieutenants run their instance of OM(m-1)

Algorithm: Step 3

- **In step** OM(m-1), general i receives a total of (n-1) values
  - **From (n-2)** lieutenant generals (result of their OM(m-1) after Byzantine agreement)
  - The commander sends his value to every lieutenant
  - One value from its commander
  - It computes the majority of the (n-1) values(=v)
- It uses this value v as the output of OM(m-1)

With f faulty node, we need at least(3f+1) nodes to reach consensus

- Notes:
  - This is a recursive algorithm
  - Since generals don't trust their commander, they do not accept an order at face value
  - They inquire with the rest of the generals, and ask them what they have gotten
  - Each general collects(n-2) response from other lieutenants, and 1 from its commander
  - It can be proven that all loyal generals will compute the same majority value subsequently

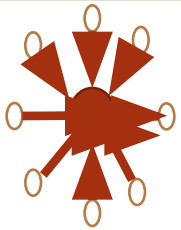
Hybrid Consensus

Hybrid Consensus

- The core of blockchain technology, the consensus algorithm plays an important role in determining the security, data consistency, and efficiency of blockchain systems
- A single mainstream consensus algorithm is often difficult to fully satisfy the needs of real-world scenarios (e.g., security, decentralization, and efficiency).
- Each consensus algorithm has some limitations
- Therefore, we design a hybrid consensus algorithm to make it have the advantages of the mainstream consensus algorithms and make up for their shortcomings as much as possible
- A hybrid consensus algorithm based on the modified consensus algorithms make up for the shortcomings of single consensus algorithms
- Like PoW+ PoS
- PoW+PoS+PBFT
- PoW

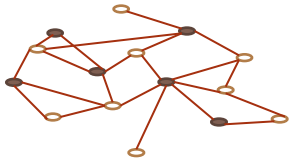
Blockchain and Future World of Web 3.0

Blockchain and Future World of Web 3.0



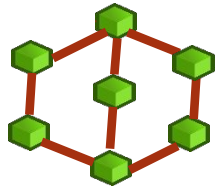
Web 1.0

- Basic Web pages
- Html
- Ecommerce
- Java & JavaScript



Web 2.0

- Social Media
- User Generated Content
- Mobile Access
- Apps
- High-speed Communication
- Global Internet Access



Web 3.0

- Semantic Web
- dApps
- Permissionless Blockchain
- NFTs
- Users Monetize their data
- AI & ML
- Interoperability

Blockchain and Future World of Web 3.0

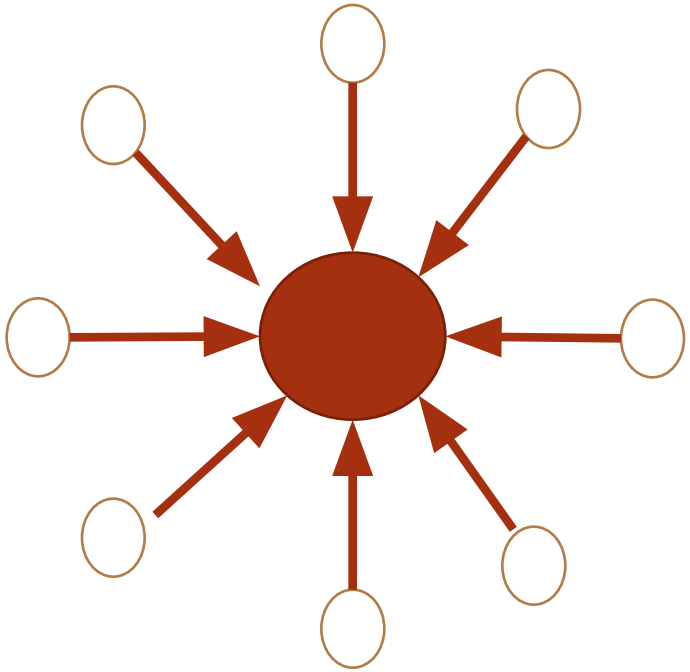
- The third generation of the evolution of web technologies
- It is anticipated that Web 3.0 will be:
  - **Decentralized -** Web 3.0, information would be found based on its content, it could be stored in multiple locations simultaneously and hence be decentralized.
  - **Trustless and Permissionless-** Web 3.0 will be trustless , the network will allow participants to interact directly without going through a trusted intermediary and permissionless , meaning that anyone can participate without authorization from a governing body
  - **Artificial Intelligence and machine learning-** In Web 3.0, computers will be able to understand information similarly to humans, through technologies based upon Semantic Web concepts and natural language processing
  - **Connectivity and ubiquity-** With Web 3.0, information and content are more connected and ubiquitous, accessed by multiple applications and with an increasing number of everyday devices connected to the web

Blockchain and Future World of Web 3.0

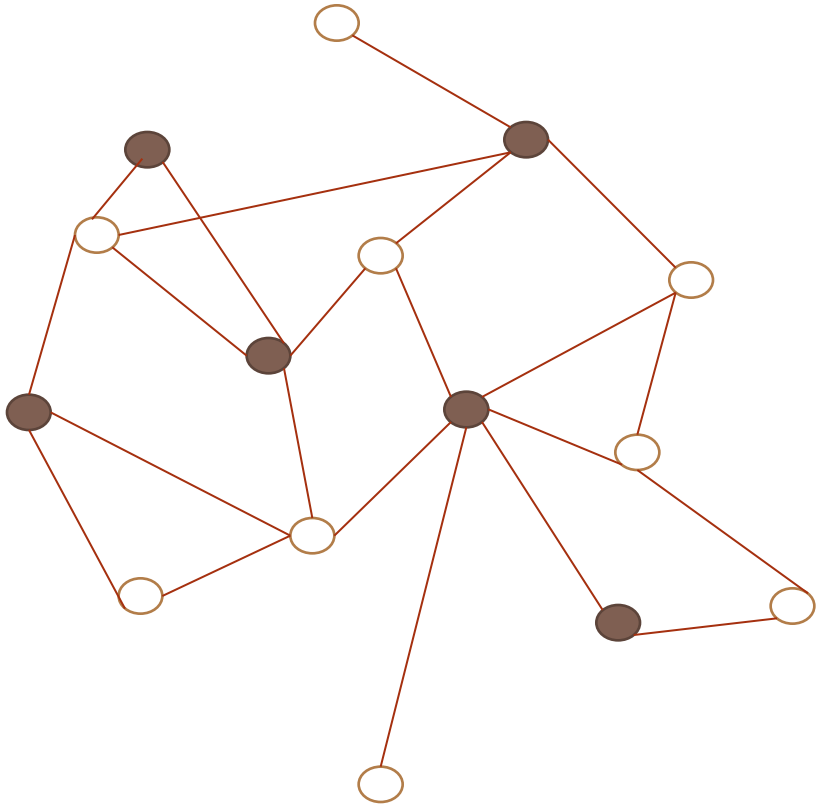
- Blockchain technology will make it possible for users to communicate directly with one another throughout the next stage of the internet
- Users will communicate by becoming a part of Decentralized Autonomous Organization (DAO) a group that is run and owned by its community
- Data belonging to the user will be protected via a network of openly available smart contracts
- These contracts will be stored in a blockchain, which a decentralized network that nodes will control.

Blockchain and Future World of Web 3.0

Web 1.0



Web 2.0



Web 3.0