香港城市大學
City University of Hong Kong

CS 6290

Privacy-enhancing Technologies

Department of Computer Science

Slides credit in part from E. Ben-Sasson, D. Dziembowski, I. Eyal, R. Geambasu, and F. Greenspan, A. Judmayer, A. Juels, A. Miller, J. Poon, V. Shmatikov, D. Song, and F. Zhang

Lecture 2 – Consensus

Prof. Cong WANG
CS Department
City University of Hong Kong

Recall: Nakamoto Consensus

- Securely reaching <u>**distributed consensus**</u> on the same ledger across the globe
  - Through mining process (i.e., PoW)

What is Consensus?

First, let me ask you all a simple question:

In your opinion, which flavour for Douhua is the best?

Well… I hear many different answers:
savoury, spicy, sweet, …

But, sadly, we only got one medal …

How can we jointly decide the winner?

It is Easy!

- We can do a voting!
- I can ask each of your answer, count the result, and finally pick the most voted one

Analysis

- Everyone needs to trust me for honestly conducting the voting process!
  - I am the trusted third party

- Can the previous voting process be conducted correctly without me?
  - I.e., with no central coordinator
  - Only through talking to each other

The Consensus Problem

- <u>Distributed consensus:</u>

  All "correct" nodes decide on the **same** value

  This value must have been proposed by some **correct** node

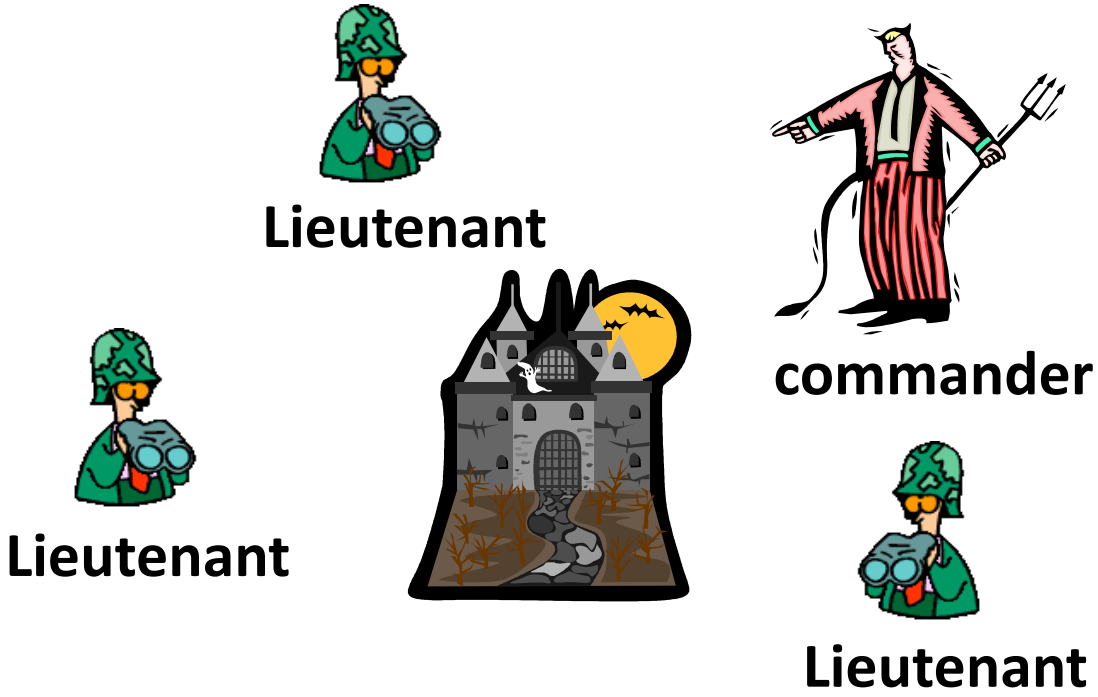
  The protocol terminates

The medal must be given to a **valid** option, not some incorrect ones (e.g., crystal shrimp dumping)

- Actually, it's an old problem

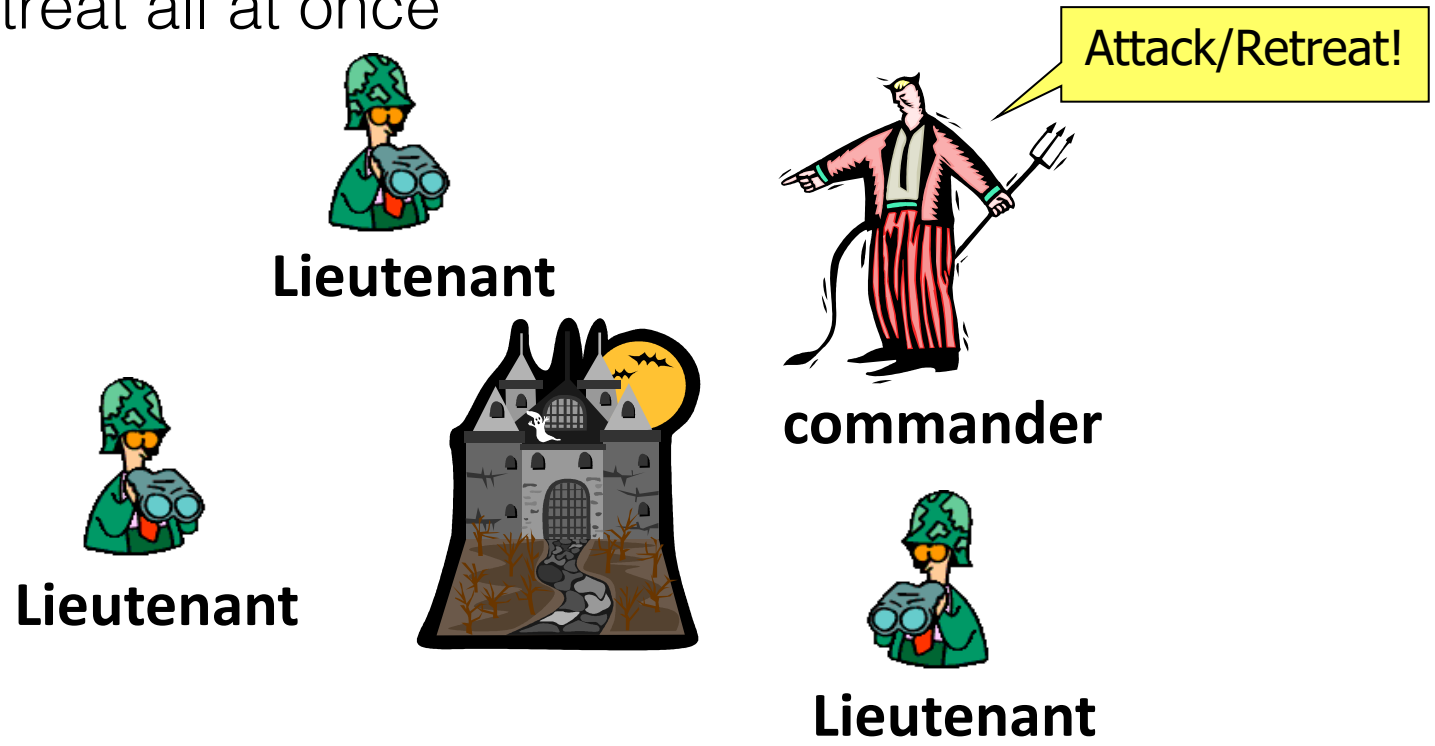
The Byzantine Generals Problem

- The Byzantine army has surrounded a city and decided to capture it
  - Lieutenant generals communicate between each other and to the commander general **only through messengers**



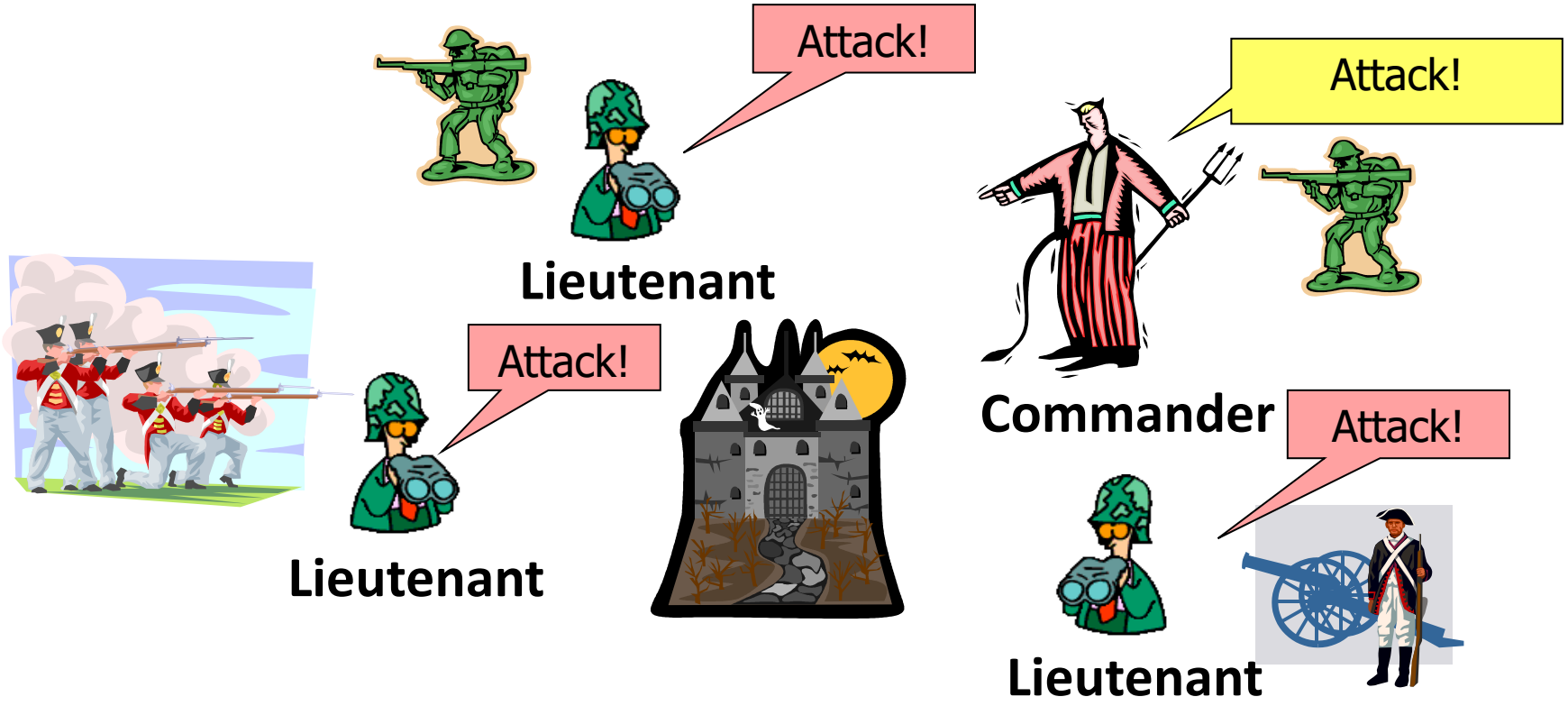
The Byzantine Generals Problem

- They have to <u>agree upon one of the two plans of actions</u>
  - Exact time to attack all at once
  - Or, if faced by fierce resistance then the time to retreat all at once



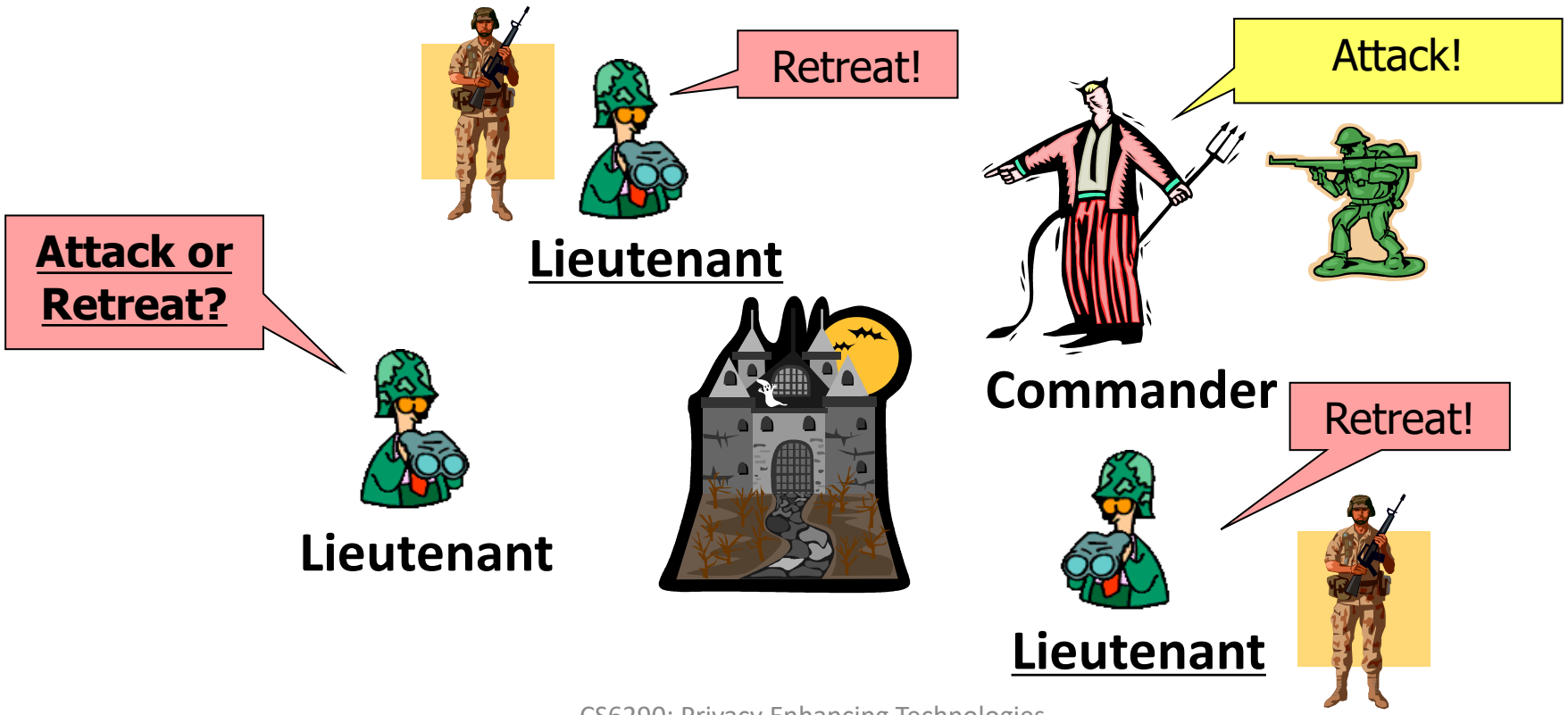
Simple Case

- If the commander and all lieutenants are <u>honest</u>



However, Things can Get Difficult

- What if some of the lieutenants/commander are <u>traitors</u>
  - Will not follow orders or pass on incorrect message



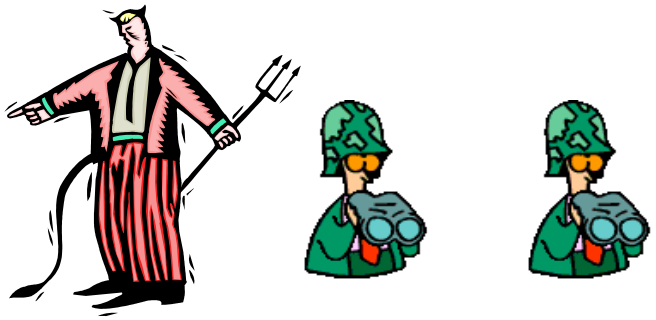
Problem Formalization

- <u>Byzantine Generals Problem (BGP):</u>
  - A commander must send an order to his $n-1$ lieutenants such that
    a) All honest/loyal lieutenants obey the same order
    b) If the commander is honest, then every honest/loyal lieutenant obeys the order he sends

- Note: both the lieutenants and the commander can be dishonest

Problem Demonstration

- Scenario:
  - One commander and two lieutenants



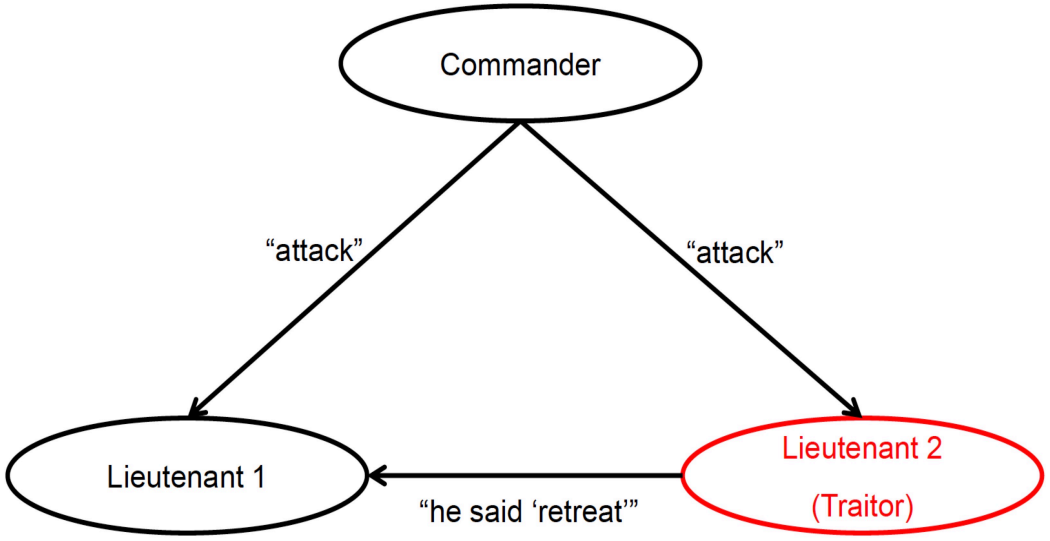
With **one traitor**, can non-traitors

<u>decide on a common plan</u> ?

Situation 1(assuming traitor is known)

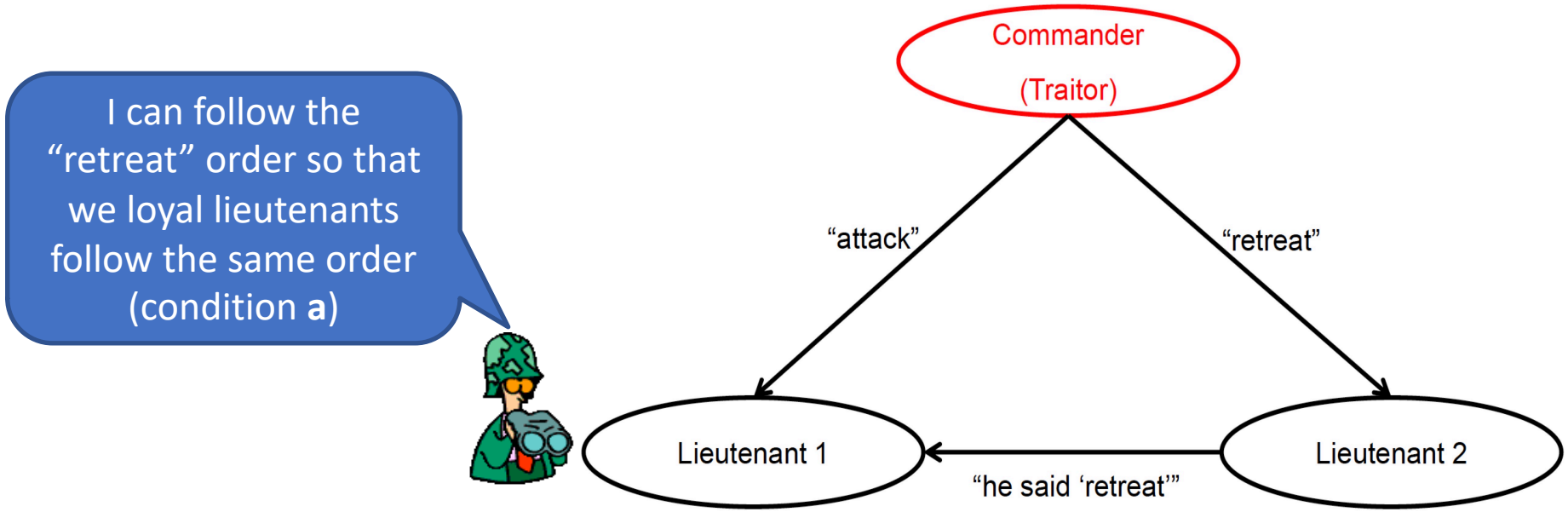
- Commander is honest. Lieutenant 2 is the traitor and sends the fake "retreat" order to lieutenant 1

To satisfy **Condition b)** in the problem, I must obey the commander's order to attack



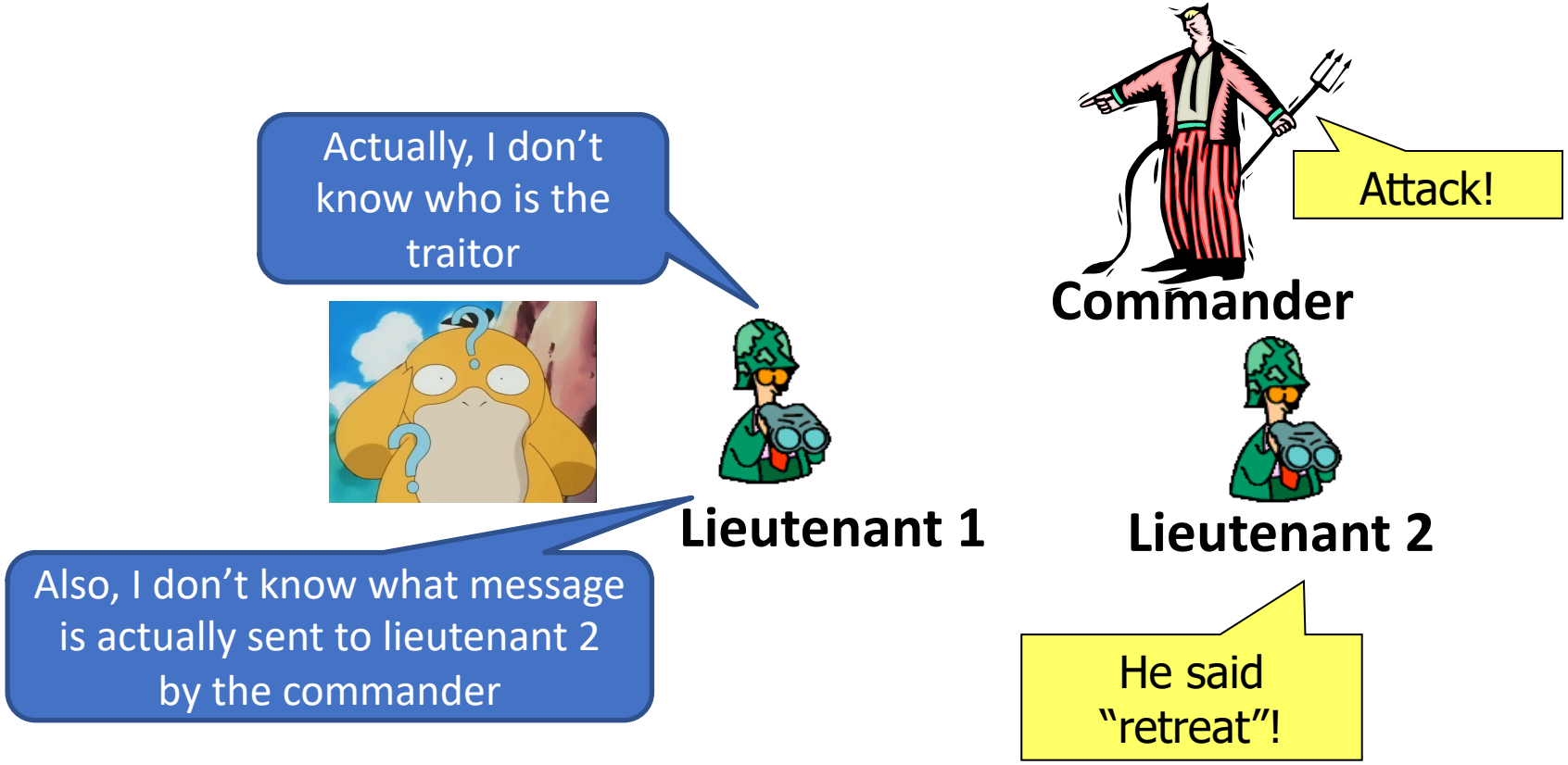
Situation 2 (assuming traitor is known)

- **Commander is the traitor** and sends an "attack" order to Lieutenant 1 and a "retreat" order to lieutenant 2



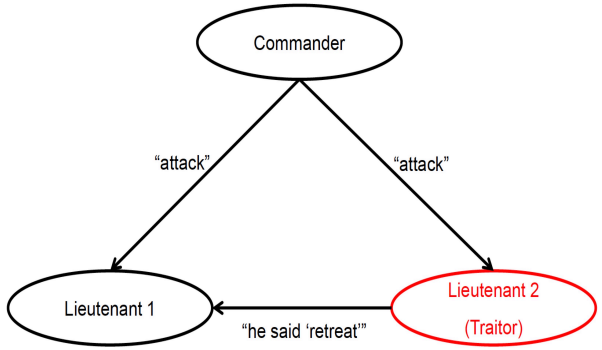
However, Traitor will Hide in the Dark

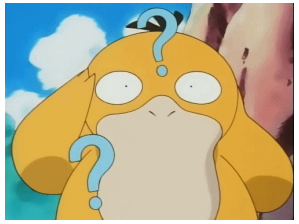
- Lieutenant 1's actual view of conflicting information from the commander and the other lieutenant



Analysis

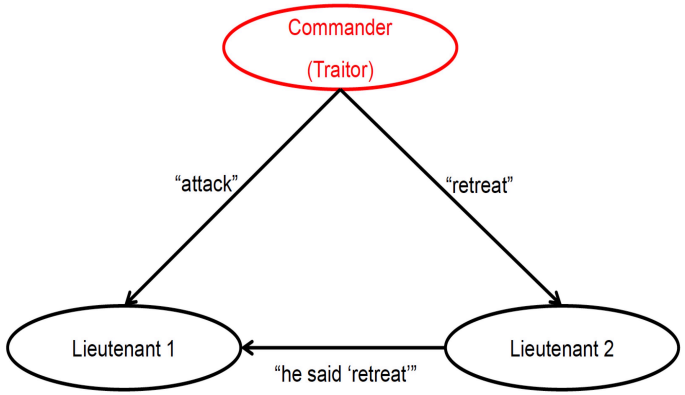
Situation 1





Lieutenant 1

Situation 2

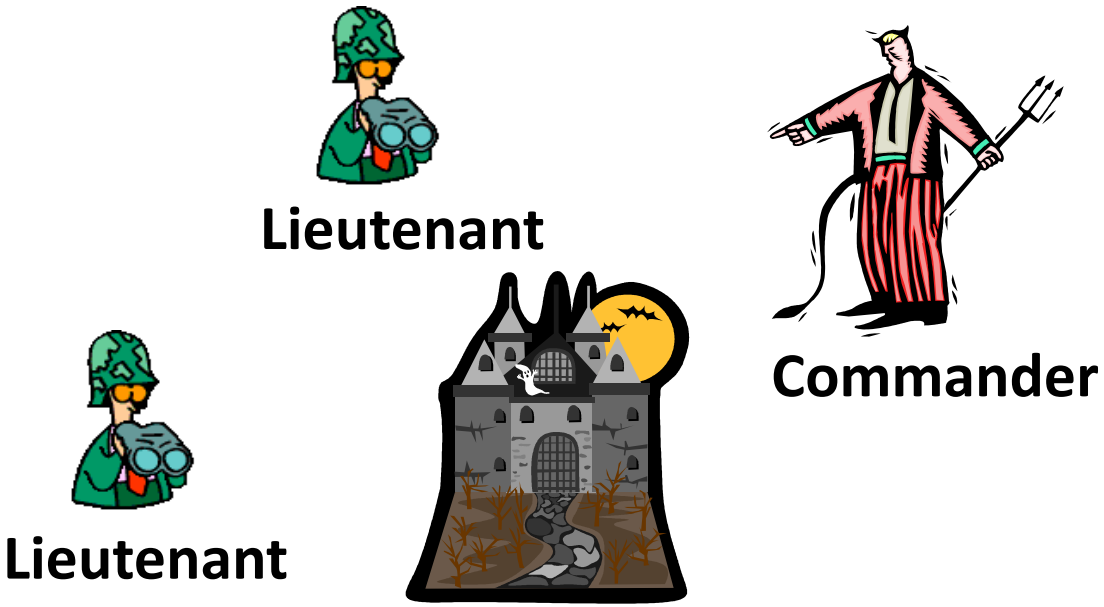


Lieutenant 1 **can never distinguish between these two situations** when he got conflicting information from the commander and the other lieutenant

=> Generally speaking: no solution can guarantee that the BGP problem **<u>can always be solved</u>** (better than random guess)

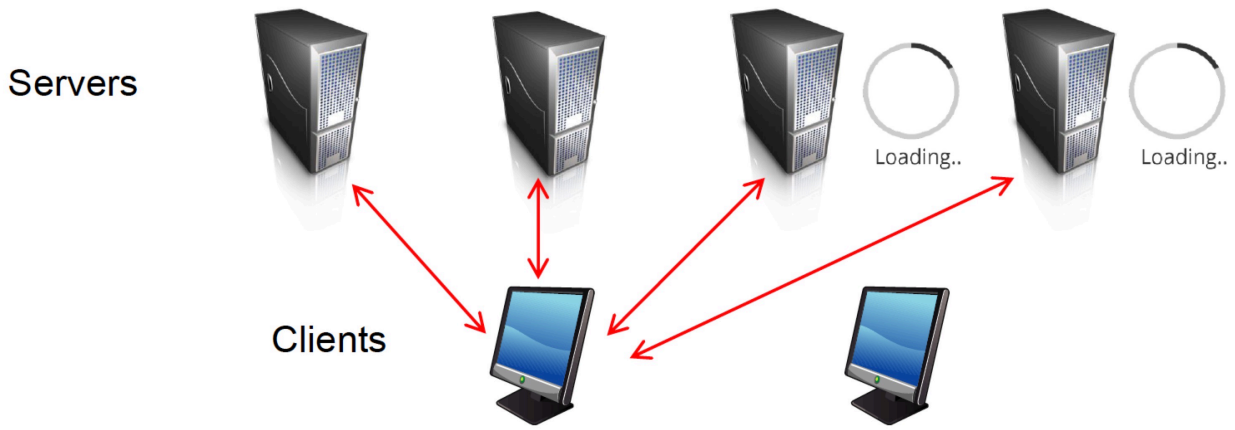
Impossibility Results

- One traitor makes it impossible with three generals (to solve the BGP problem)
- Or more generally, a set of 2f nodes can be deadlocked between the non-faulty (f nodes) and faulty (f nodes)



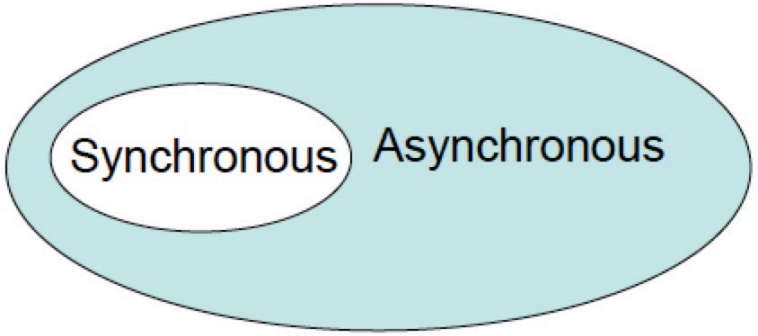
More Practical Setting

- In the previous setting, we assume that the message delays between generals are **bounded and the bound is known**
  - i.e., in the synchronous model
- However, in real systems, e.g., **replicated web servers**, some servers can just be very slow



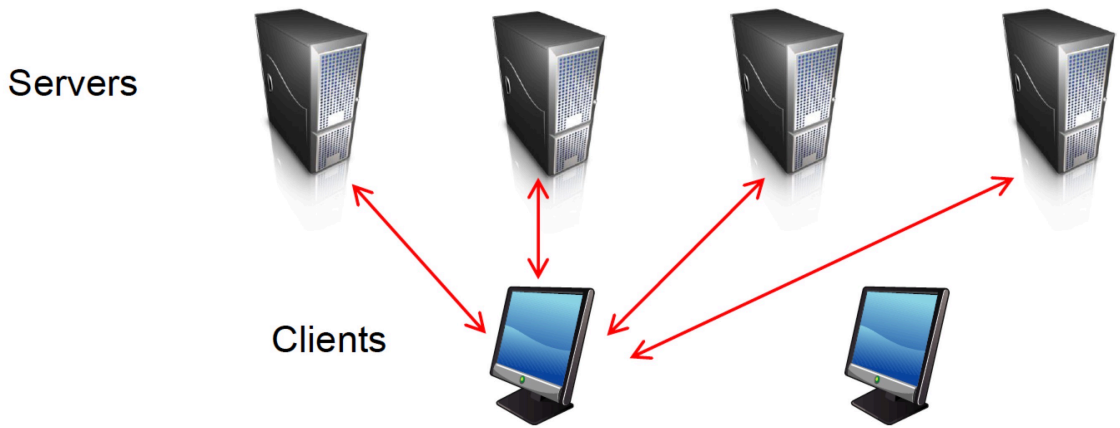
Asynchronous Model

- Message delays are finite, but <u>unbounded or unknown</u>
- More realistic/general than synchronous model
- Strictly harder/weaker than synchronous model
  - Consensus is not always possible



Case Study

- <u>Replicated Web Servers</u>
  - Multiple servers running the same state machine
  - For example, a client asks a question and each server replies with an answer (yes/no)
  - Goal: the client determines what the correct answer is based on the replies



Threat Assumptions

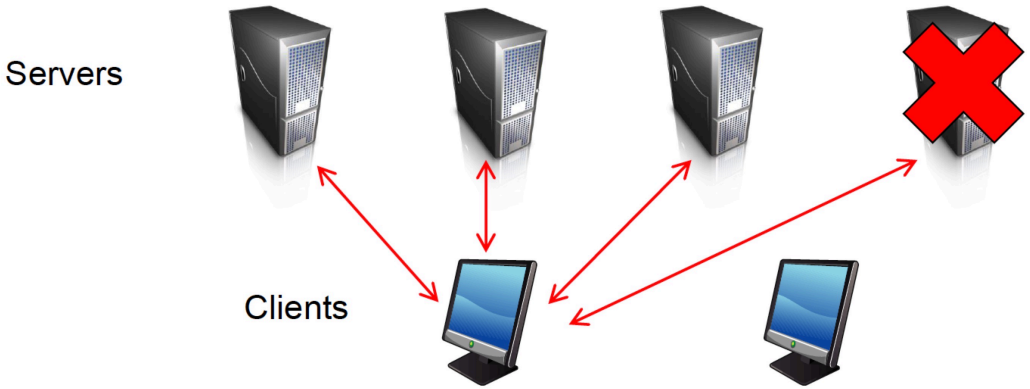
- f Byzantine failures
  - At any point of time, there can be up to f failures
- Many possibilities for a failure
  - A crashed process, a message loss, a malicious behaviour (e.g., a lie), etc
  - A client cannot tell which one it is
  - But in total, the maximum # of failures is bounded by f

Byzantine Fault Tolerance (BFT) Question

- Given f, how many nodes do we need to tolerate f Byzantine failures?
  - f failures can be any mix of malicious servers, crashed servers, message losses, etc
  - Malicious servers can do anything, e.g., they can lie (**If yes, say no; if no, say yes)**

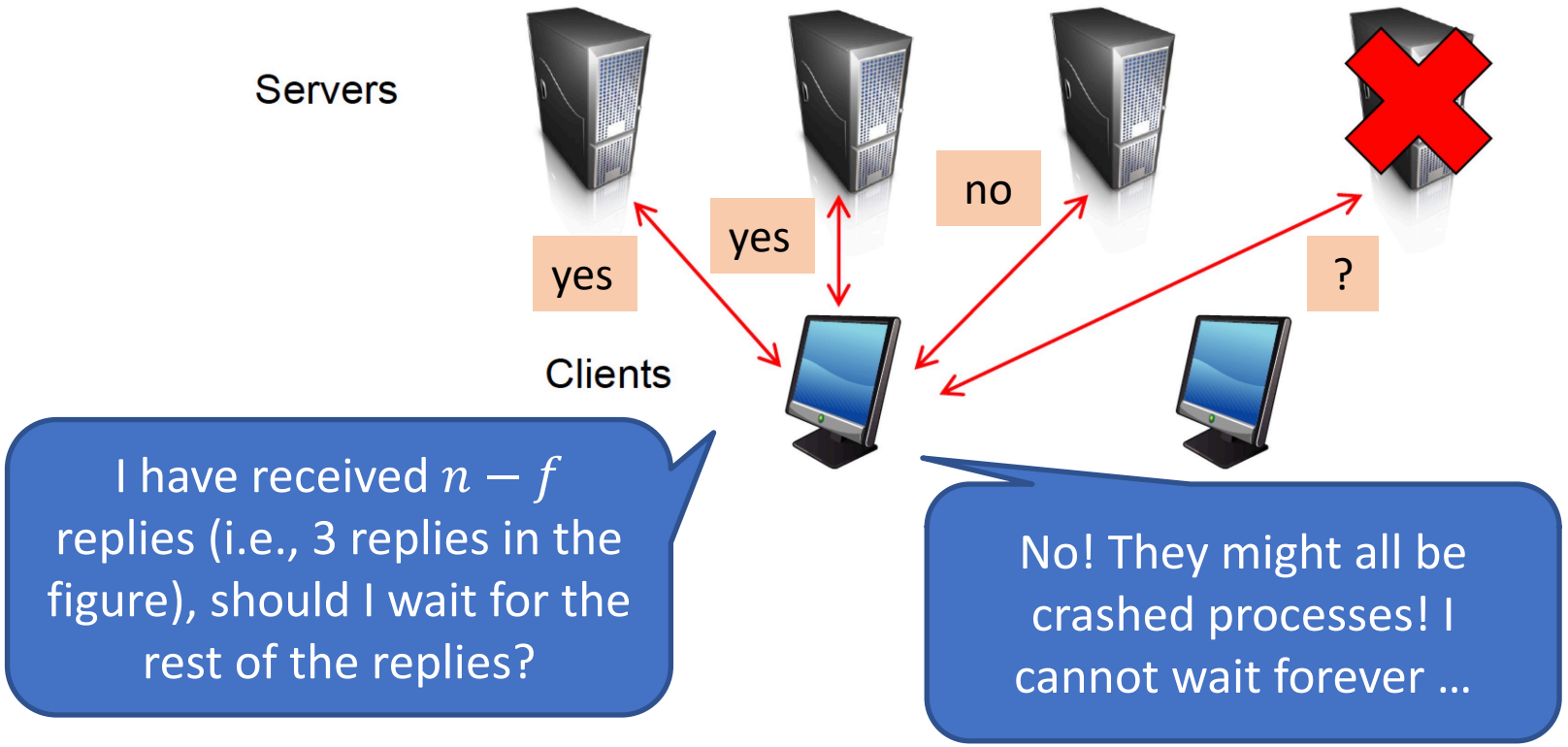
Intuition for the Result

- Let's assume we have n servers, and maximum f Byzantine failures
- What is the minimum # of replies that you are always guaranteed to get?
  - $n - f$
  - when f failures are all crashed/unresponsive processes



Intuition for the Result (Cont.)

- The problem is that a client does not know what kinds of those f failures are



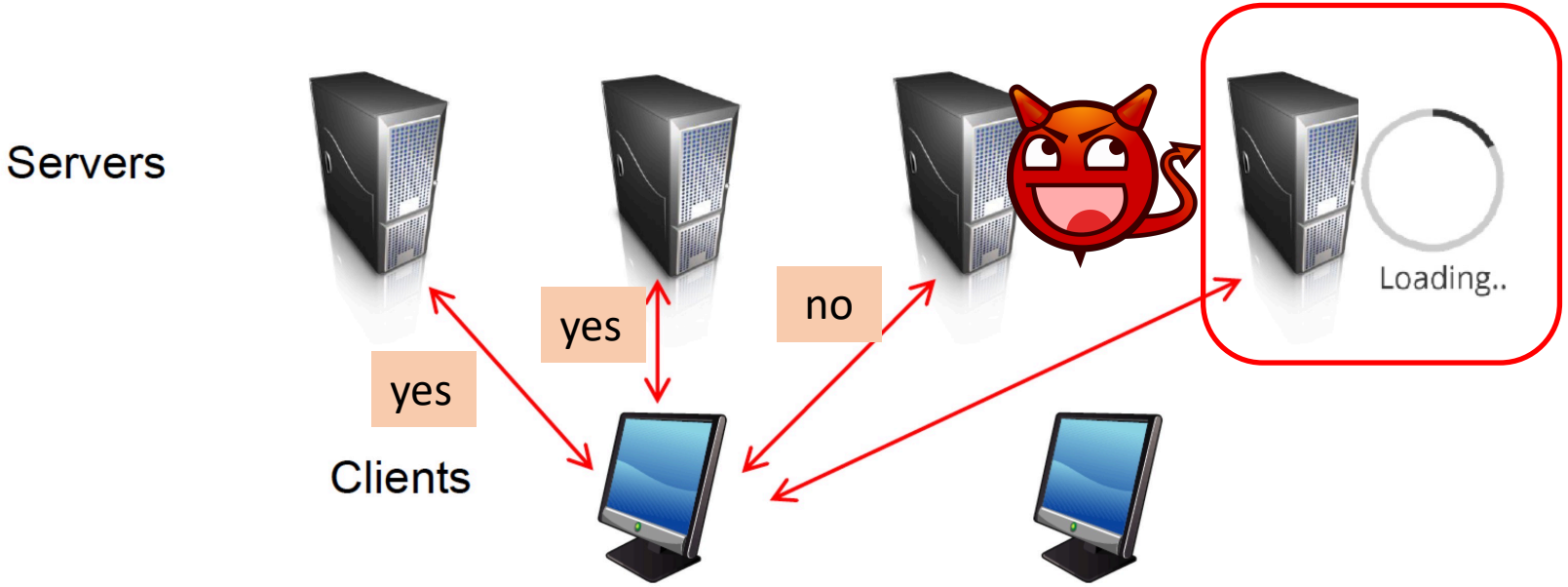
Intuition for the Result (Cont.)

- This means that if a client receives $n - f$ replies, the client needs to determine what the correct answer is at that time
  - because the rest of the replies might never come
- However, upon receiving $n - f$ replies, how many replies can come from malicious servers (i.e., lies)?

Still f ,why?

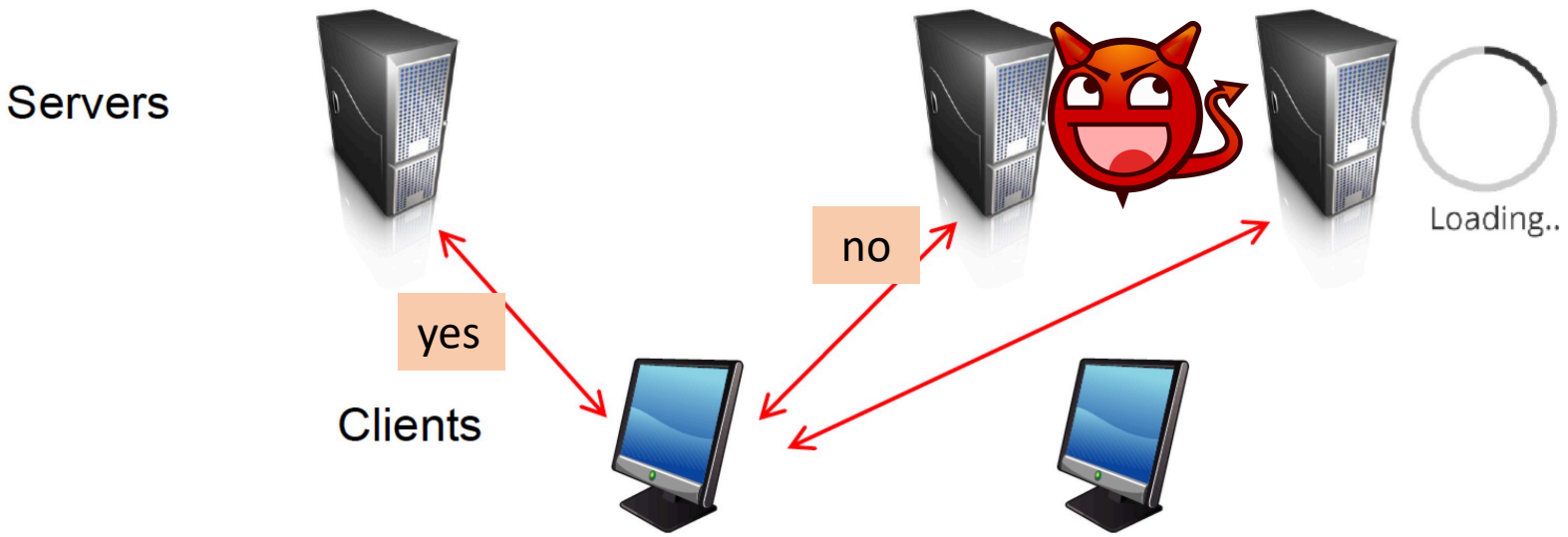
Intuition for the Result (Cont.)

- Since some servers can just be really **slow**!



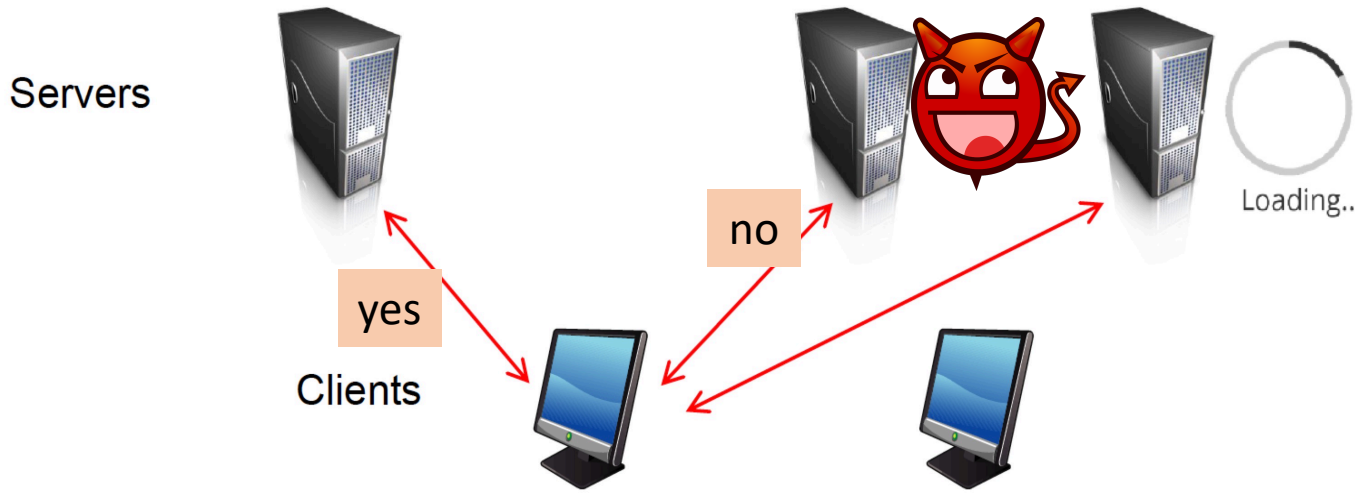
Intuition for the Result (Cont.)

- What can be the minimum n to determine the correct answer? What if $n == 2f + 1$? (where f = 1)
  - $n - f = 3 - 1 = 2$
  - Obviously, it doesn't work



Result for the BFT Question

- If $n - f$ replies always contain <u>more replies from honest nodes than Byzantine nodes</u>, we're safe!
  - We set $\boldsymbol{n == 3f + 1}$
  - We can always obtain $n - f$, i.e., $2f + 1$ votes. Then we have at least $f + 1$ votes from honest nodes
    - <u>one more</u> than the number of potential faulty nodes

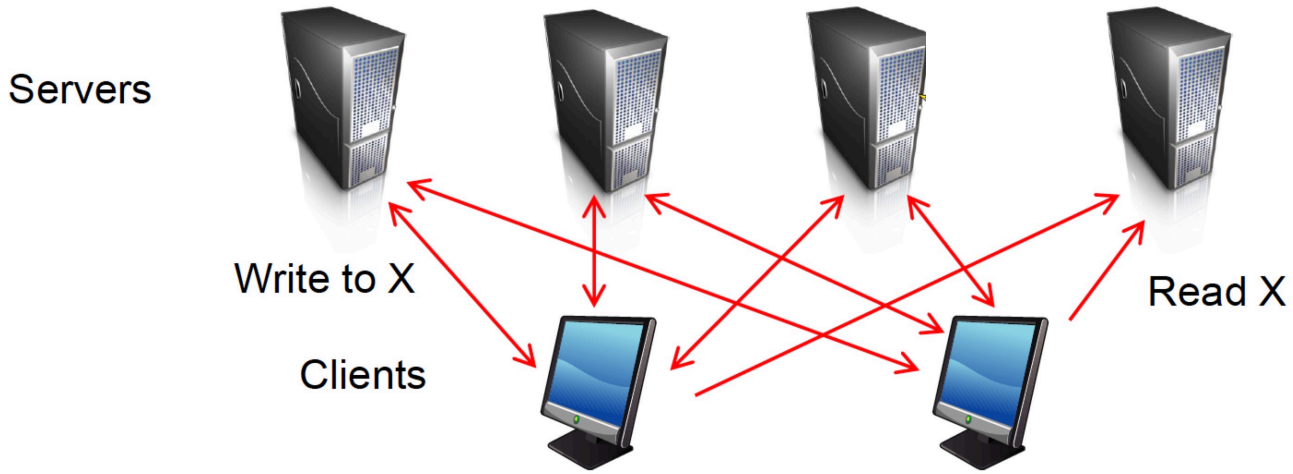


Practical Byzantine Fault Tolerance (PBFT),
M. Castro and B. Liskov,
in Proc. of OSDI, 1999

(Optional, will not include in the final exam)

PBFT in a Nutshell

- Designed for <u>replicated state machines</u>
  - Arbitrary service accessed by operations, e.g.,
    - E.g., read and write operations in a file system
  - Tolerates f Byzantine failures with $3f + 1$ replicas in the asynchronous model



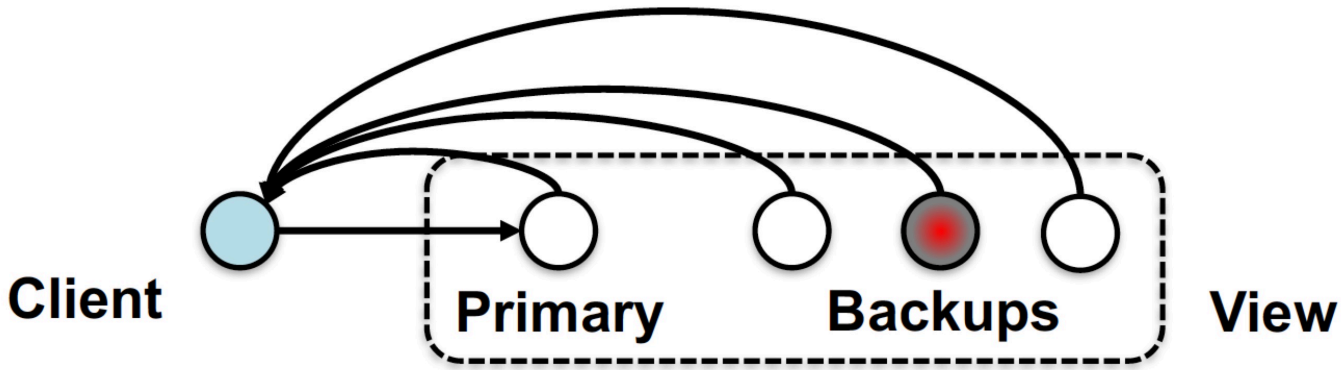
Correctness in Replicated State Machine

- Assume
  - Operations are **deterministic**
  - Replicas start in same state

- Then if replicas execute the <u>**same requests**</u> in the <u>**same order**</u>:
  - Correct replicas will produce identical results

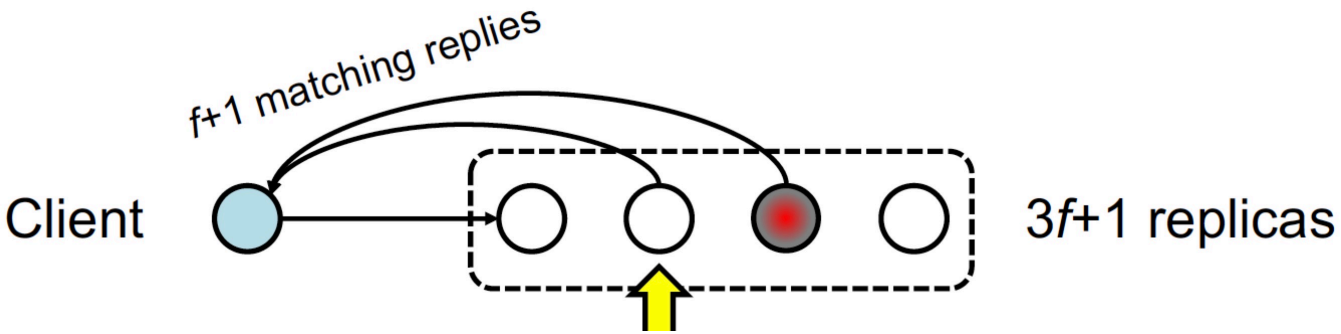
PBFT System Setting

- Primary replica + backup replicas
  - Primary performs operations
  - Backups monitor the primary and do a view change (will explain later) if they detect a primary failure
- All messages are <u>**authenticated**</u> using crypto-based techniques



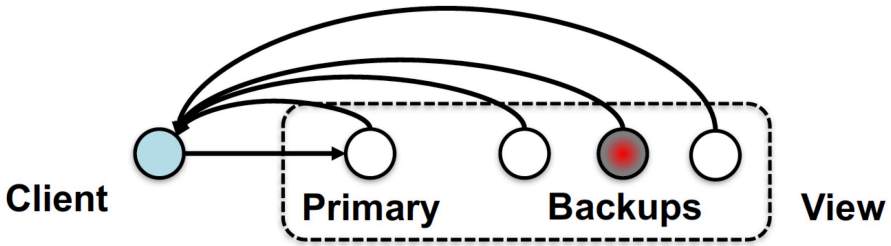
10,000ft View

1. Client sends request to primary
2. Primary sends request to all backups
3. Replicas execute the request and send the reply to the client
4. Client waits for $f + 1$ responses with the same result



Views

- System goes through a series of views
- In view v, replica $(v \ mod \ (3f + 1))$ is designated primary
  - Responsible for selecting the **order of operations**
  - Assigns an increasing **<u>sequence number</u>** to each operation
- Tentative order subject to replicas accepting
  - May get rejected if
    - A new view is established
    - Or the order is inconsistent with prior operations



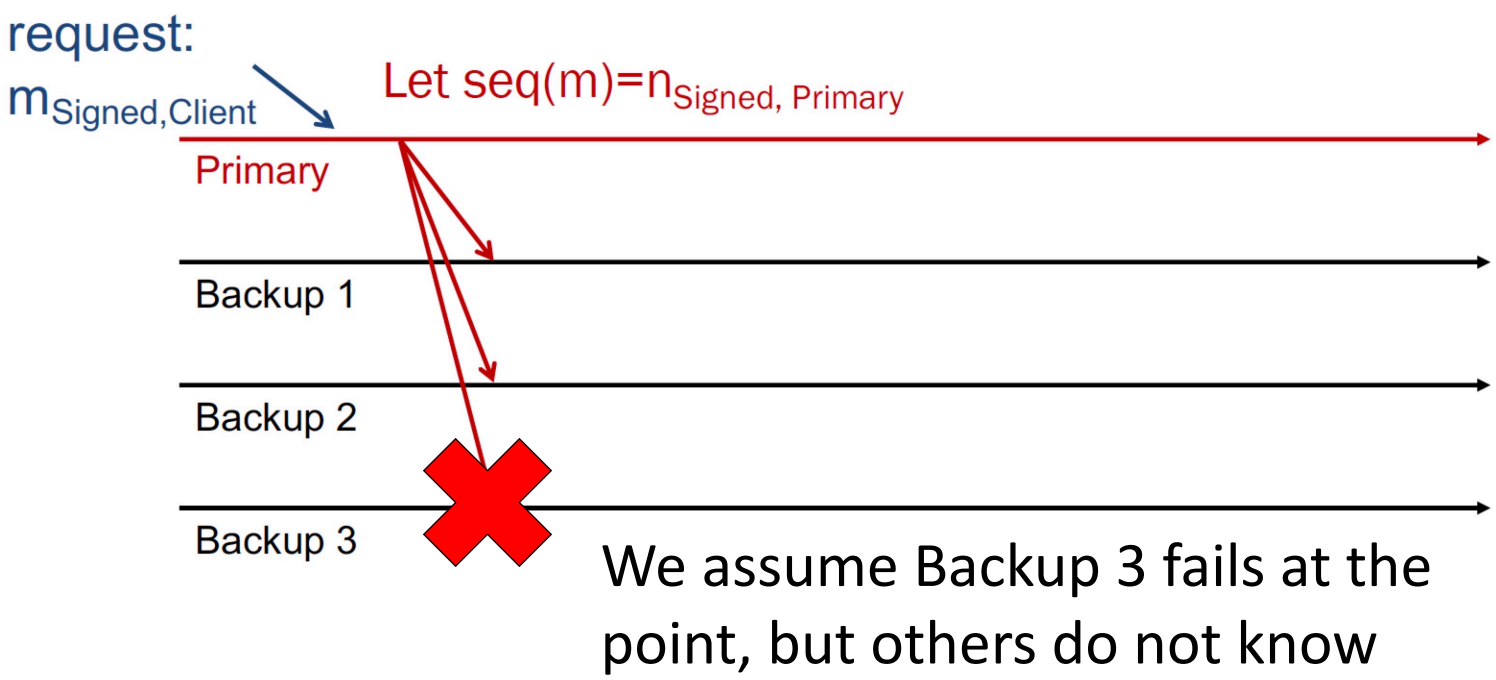
Replace the primary with another replica

Request Handling Phases

- In normal-case operation (the primary is not faulty), use two-phase protocol for request r:
- Phase 1 (pre-prepare, prepare) goal:
  - Ensure at least $f + 1$ honest replicas agree that "If request r executes in view v, will execute with sequence number (seqn) n"
- Phase 2 (prepare, commit) goal:
  - Ensure at least $f + 1$ honest replicas agree that "Request r has executed in view v with seqn n"

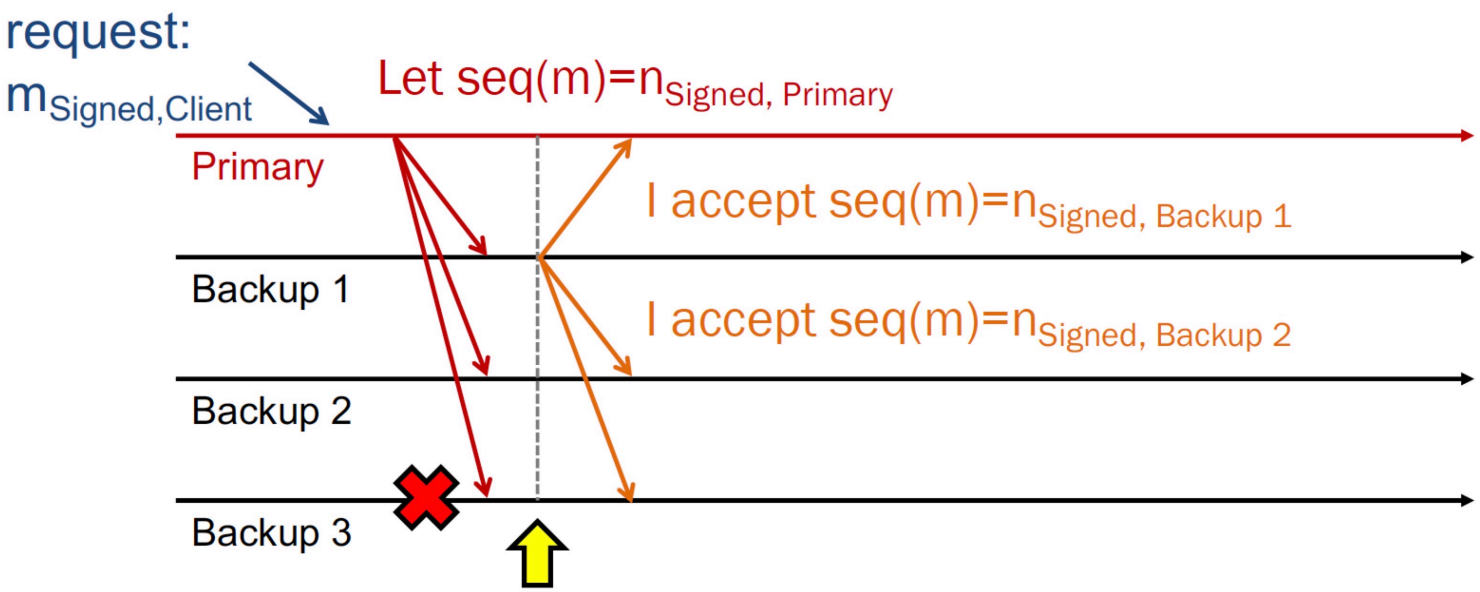
Phase 1: Ordering Requests

- Client send request to the primary
- Primary picks the request's seqn (<u>n</u>) and multicast the request to all backups



Phase 1: Checking the Primary's Message

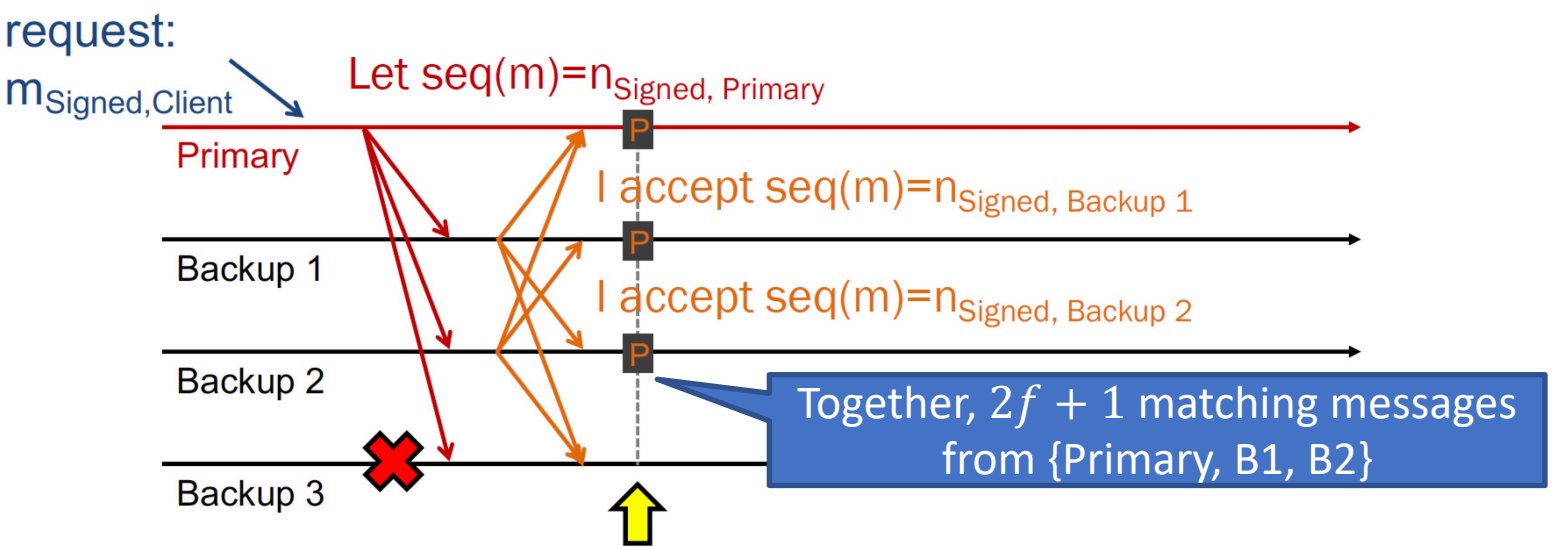
- Backups **locally** verify they've seen ≤ **one** client request for seqn <u>n</u>
  - If local check passes, replica broadcasts accept message
  - Each replica makes this decision independently



Phase 1: Collecting a Prepared Certificate

(f = 1)

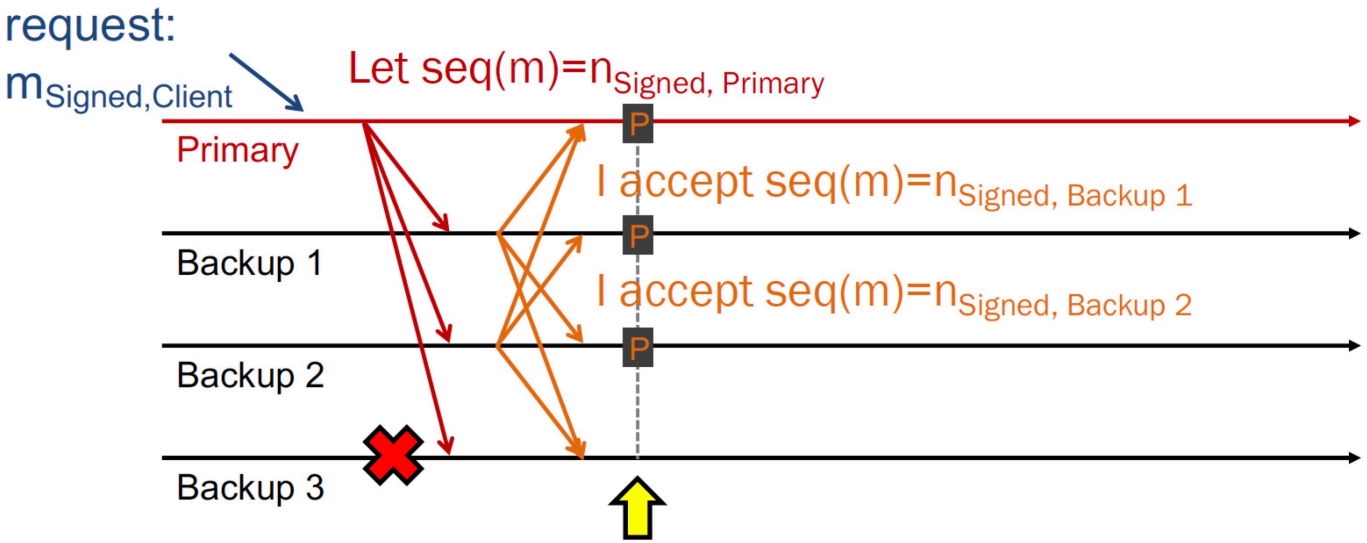
- Backups wait to collect a **prepared** quorum certificate
- Message is prepared (P)/**TRUE** at a replica when it has:
  - A message from the primary proposing the seqn
  - $2f$ messages from itself and others accepting the seqn



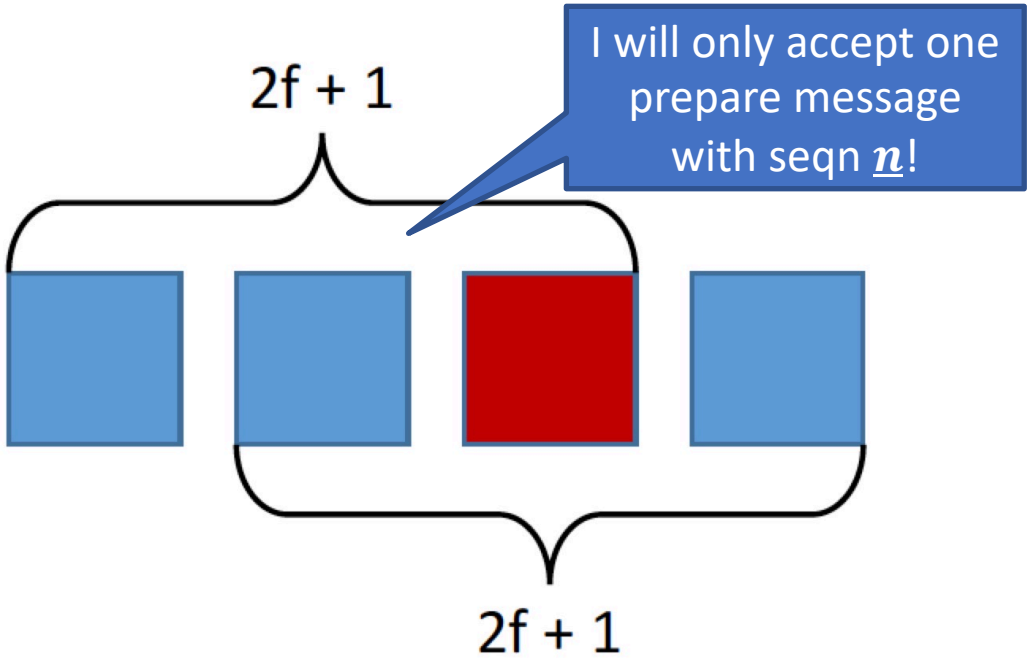
Result of Phase 1

(f = 1)

- If prepared (m, v, n, i) is **TRUE** for honest replica r_i then prepared (m', v, n, j) where $m' \neq m$ **FALSE** for any honest r_j
  - <u>So no other operation can execute with view $\boldsymbol{v}$ sequence number $\boldsymbol{n}$</u>

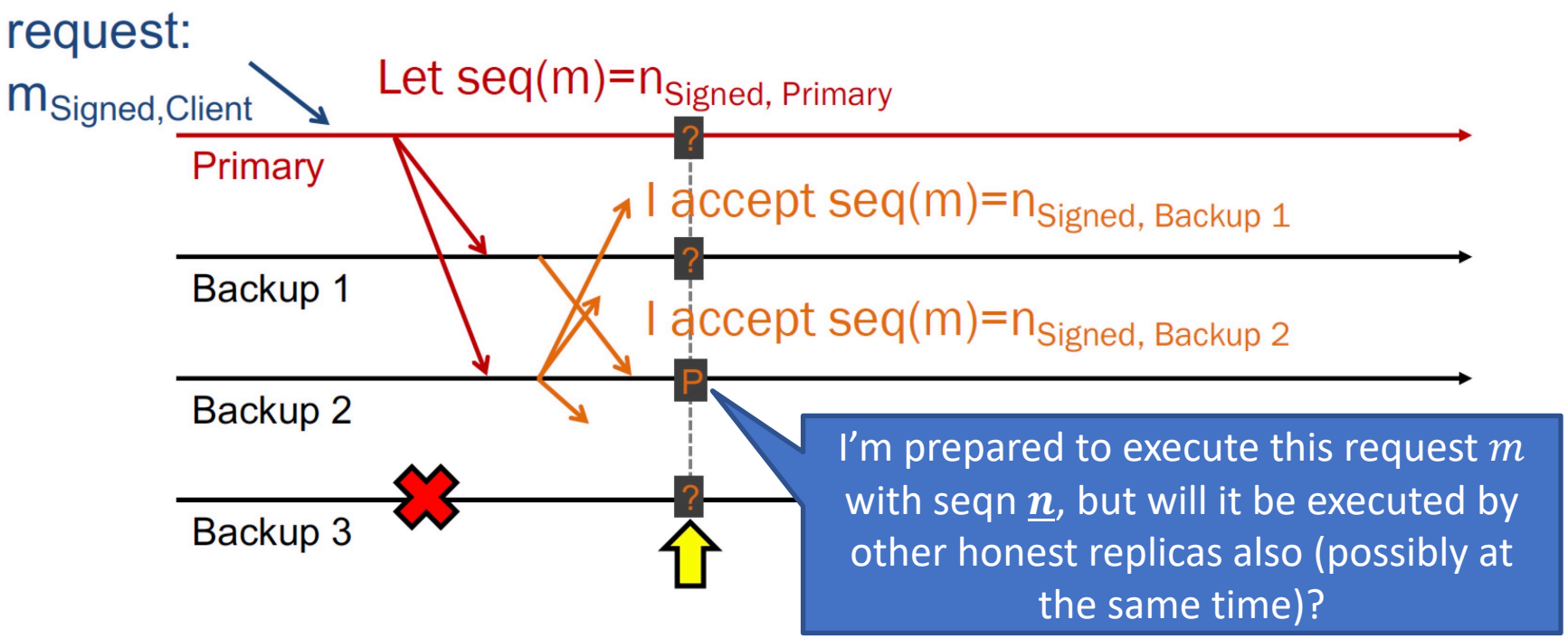


Why No Double Prepares?



- Honest intersection of maximally disjoint $2f + 1$ sets is non-empty

Phase 2: Problem



Extreme Case

- In view v, Backup 2 has prepared for message (n, m) and assumes the other honest replicas do the same thing also
- It executes the message m and replies to the client (backed by f Byzantine replicas), convincing the client that (n, m) was committed
- However, Backup 1 has never prepared (i.e., never see all $2f + 1$ prepare certificate) for message (n, m). Neither did the other honest replicas
  - Might due to network delays

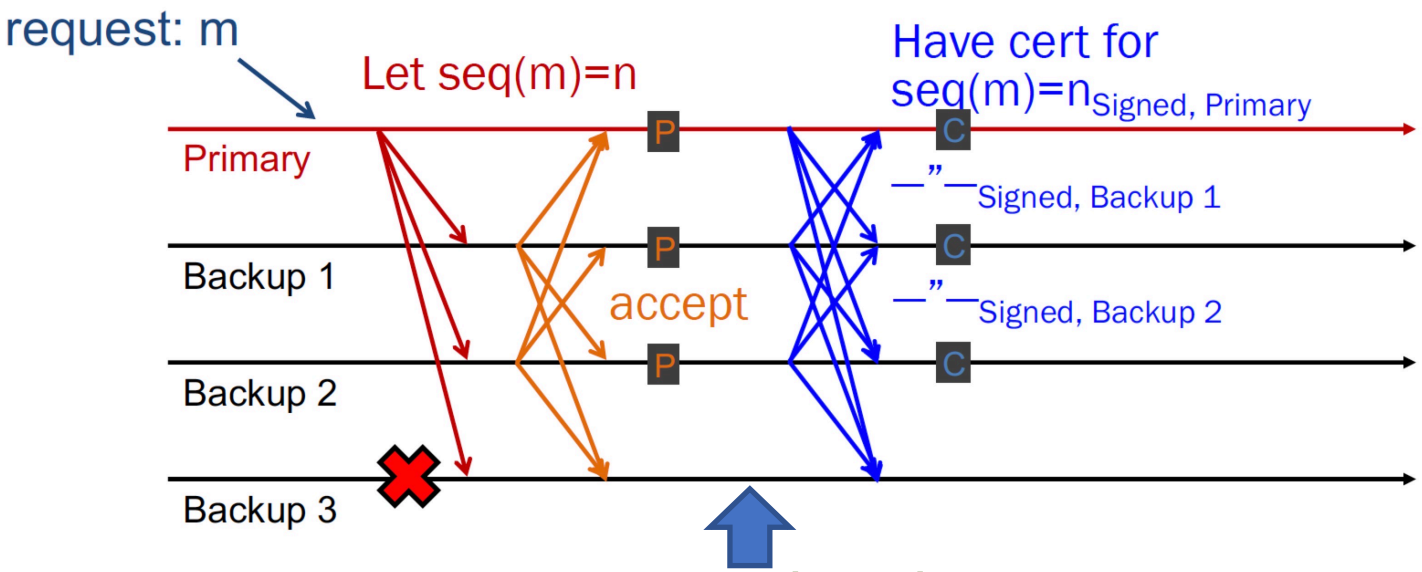
Extreme Case (Cont.)

- Hence, if there is a view change
  - <u>Say, now Backup 1 is the primary</u>
  - In this view $v + 1$, Backup 1 might assign (n, m') for $m' \neq m$, and eventually commits it to the client (with other honest replicas)
- Client now might see two conflicting results with seqn n, i.e., (n, m) committed in view v, and (n, m') committed in view $v + 1$
  - In short, consistency across views is not guaranteed

Ensuring Consistency Across Views

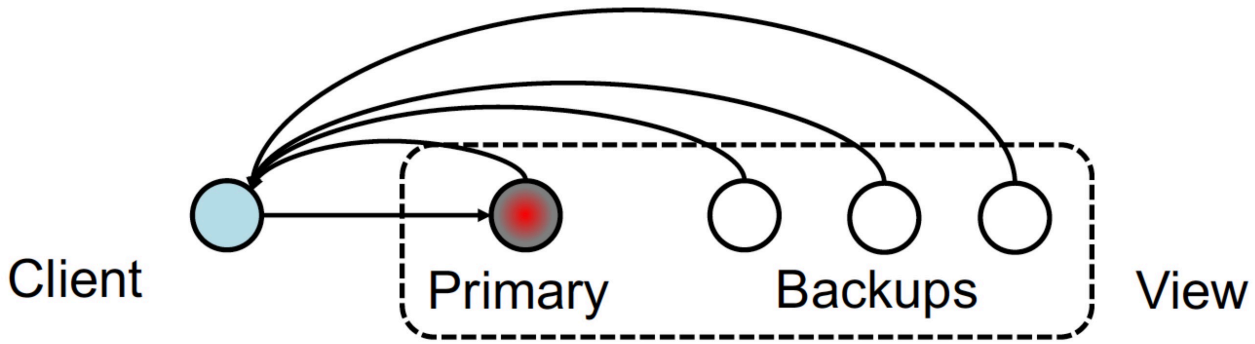
- r_i makes sure a request doesn't execute until prepared (m, v, n, i) is TRUE for $f + 1$ honest replicas
  - We say $committed(m, v, n)$ is TRUE when this property holds
- How does replica know $committed(m, v, n)$ holds?
  - Add one more message for multicasting the COMMIT message ("I'm prepared to execute (m, v, n) ")
  - Once $2f + 1$ COMMITs at a node, then execute the request and respond to client

Phase 2: Collecting a Committed Certificate



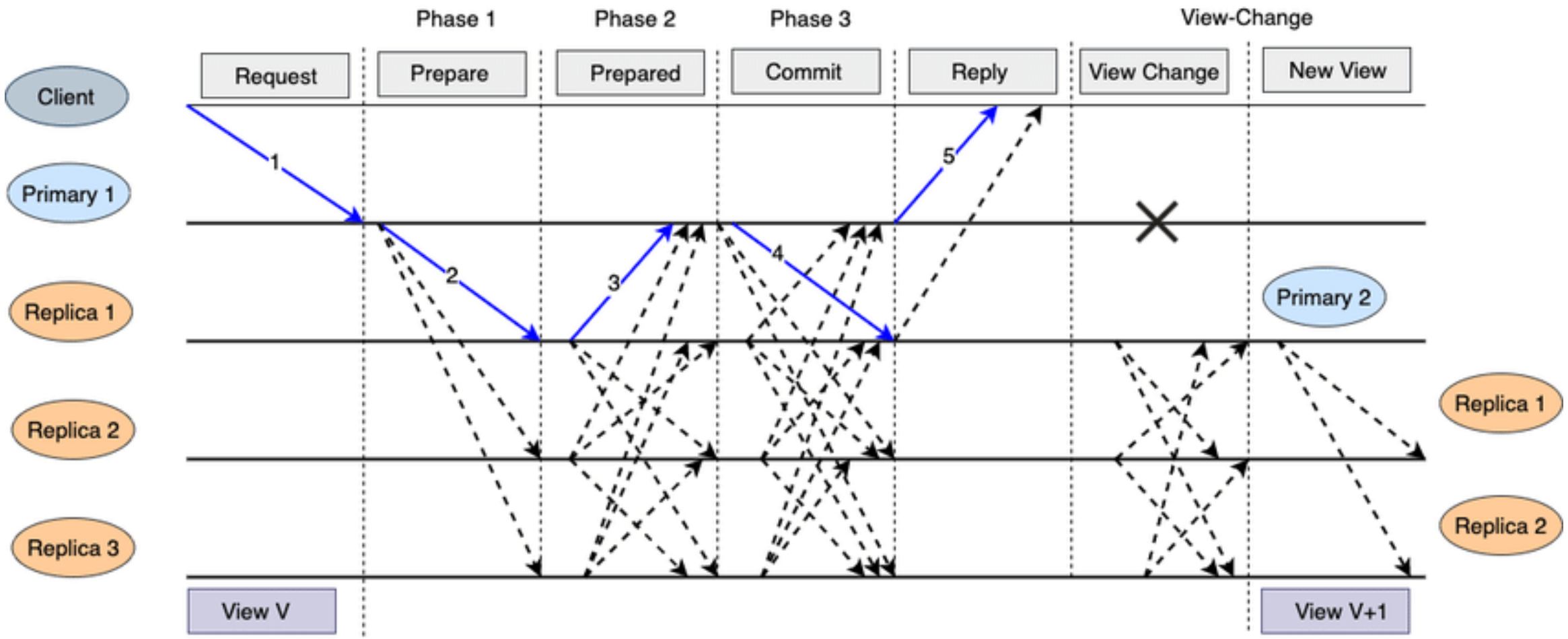
- Prepared replicas announce: they know a quorum accepts
- Replicas wait for a <u>**committed**</u> quorum certificate C: $2f + 1$ different statements that a replica is prepared, and execute the request

View Change



- If a replica suspects the primary is faulty, it requests a view change
  - Sends a **viewchange** request to all replicas
    - Everyone acks the view change request
- New primary collects a quorum $(2f + 1)$ of responses
  - Sends a new-view message with this certificate

Putting it toghther



PBFT Benefits

- Efficient
  - Add very little latency compared to an unreplicated system
  - Achieve consensus without requiring energy intensive computations
- The consensus state is final
  - Transactions can be agreed upon and finalized without needing multiple confirmations

Applications

- Aircraft systems
  - E.g., the Boeing 777 Aircraft Information Management System, and the Boeing 777 and 787 flight control system
  - The SpaceX Dragon flight system

Applications (Cont.)

- Can support private blockchains
  - Where there are only a small number of validators
  - Reaching consensus while tolerating Byzantine nodes

HYPERLEDGER

ripple

STELLAR

Zilliqa
(ZIL)

PBFT for Cryptocurrency?

- PBFT cannot tolerate a large network of nodes
  - Due to the communication overhead of multicasting
  - Tested only up to $n \leq 20$

- Obviously, it doesn't work on an open cryptocurrency system where <u>thousands of nodes</u> can be expected

Consensus in the Era of Cryptocurrency Systems

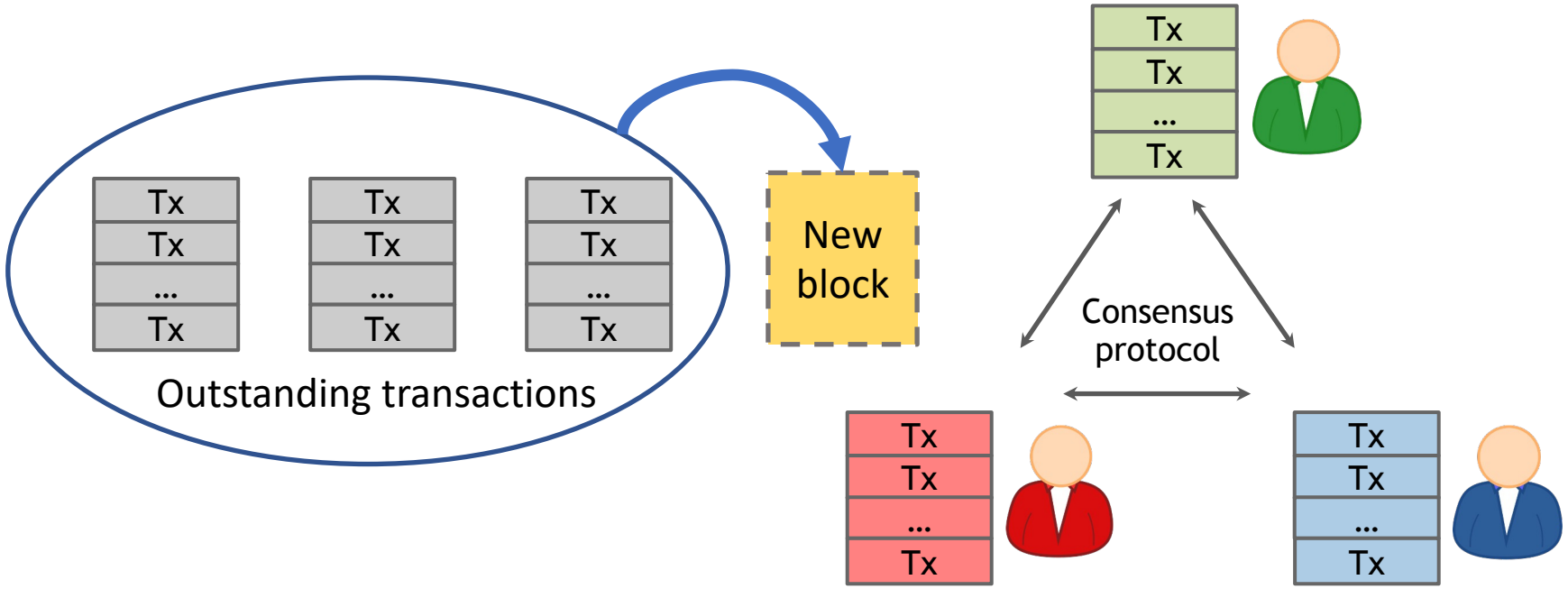
(will include in the final exam)

Nakamoto Consensus

Bitcoin uses **Nakamoto consensus**:

- **Fork-choice / proposal rule:** At any given time, each honest miner attempts to extend (i.e., mines on the tip of) the longest (heaviest) chain *held* in its view (Ties broken adversarially).
- **Confirmation rule:** Each miner confirms the block (along with its prefix) that is *k*-deep within the longest chain in its view.
  - In practice, $k = 6$.
  - Miners and clients accept the transactions in the latest confirmed block and its prefix <u>as their log</u>.
- **Leader selection rule:** Proof-of-Work.

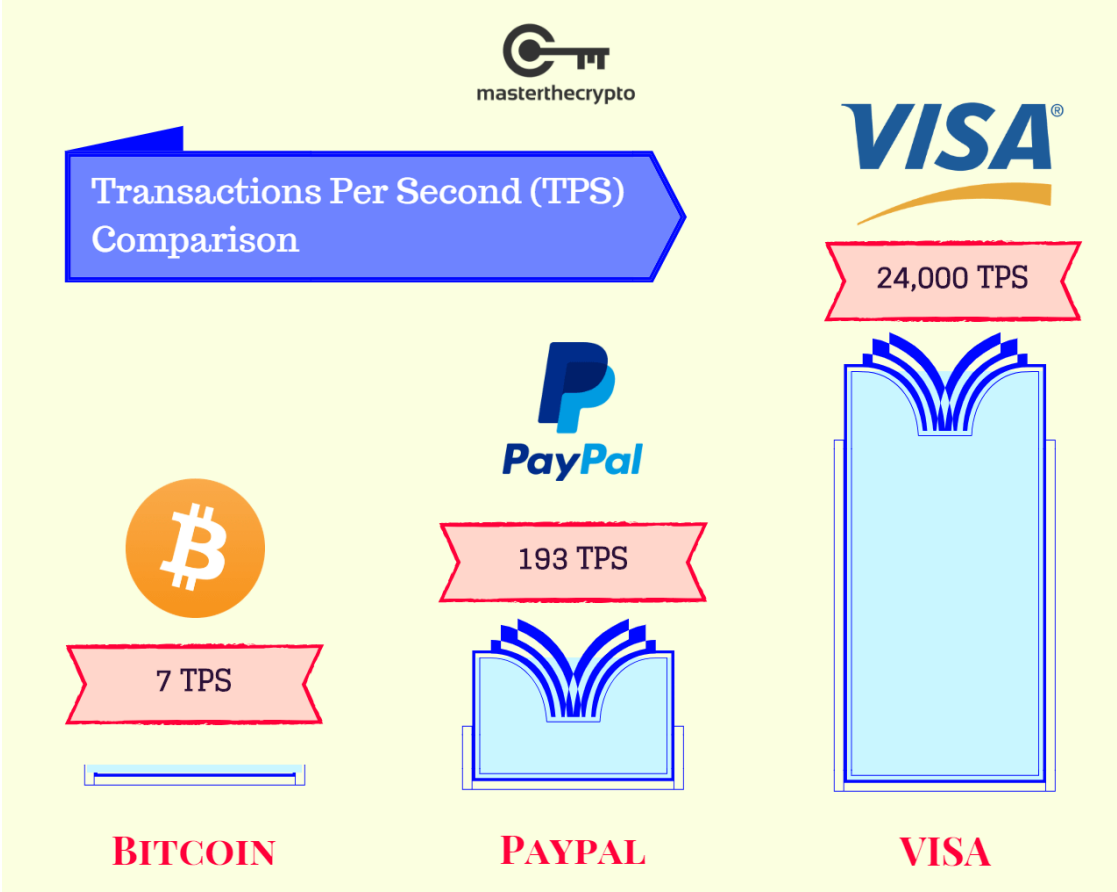
Nakamoto Consensus



- Maintain a chain of blocks where each block is filled with transactions
  - New block is added via conducting PoW minings
- If conflicts, choose the <u>longest chain</u>

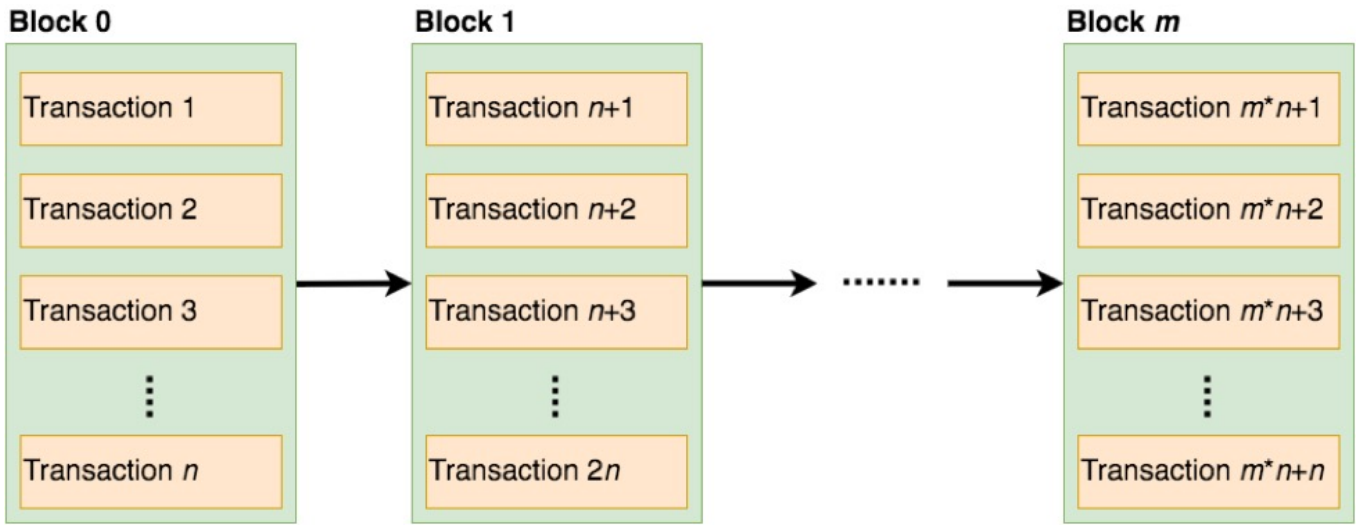
Nakamoto Consensus Scalability

- 3~7 transactions/ sec. throughput
- ~ 1 hour for confirmation
  - Average 10 mins/block, ~6 blocks ahead to confirm a transaction
- Blocks are at most 1MB in size
  - Transaction about 500 Bytes on average
  - At most 2200 transactions per block



Can Bitcoin do better?

How to scale Bitcoin

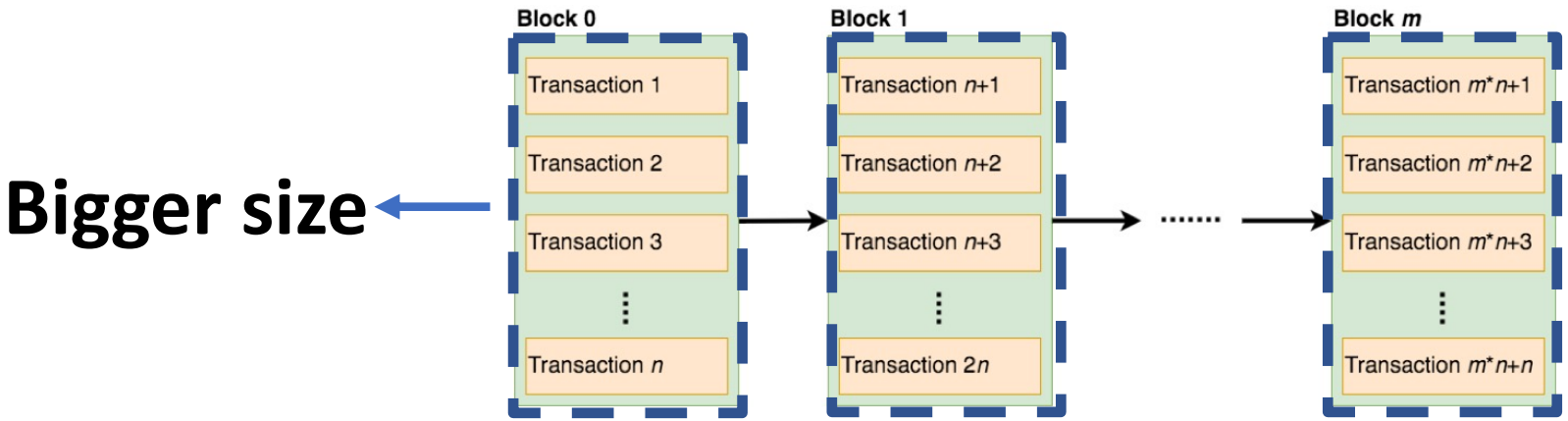


- Increase the block size?
- Increase the mining rate?
- Do something else? (new algorithms)

Big controversy!!

Attempt 1: Increase the block size

- E.g., 1 MB -> 8 MB per block, as in the Bitcoin Cash system (will introduce later)
  - Pro: <u>more transactions per sec.</u>
  - Con: <u>Require sufficient consensus to do a hard fork</u>
    - Might be catastrophic for the community



Bitcoin “tried” many times

- SegWit2x
- Time: November 2017
- Goal: increase the block size to 2 MB
- Method: hard fork
- Result: the hard fork planned for around November 16, 2017 was **canceled** for the time being due to a *lack of consensus*.

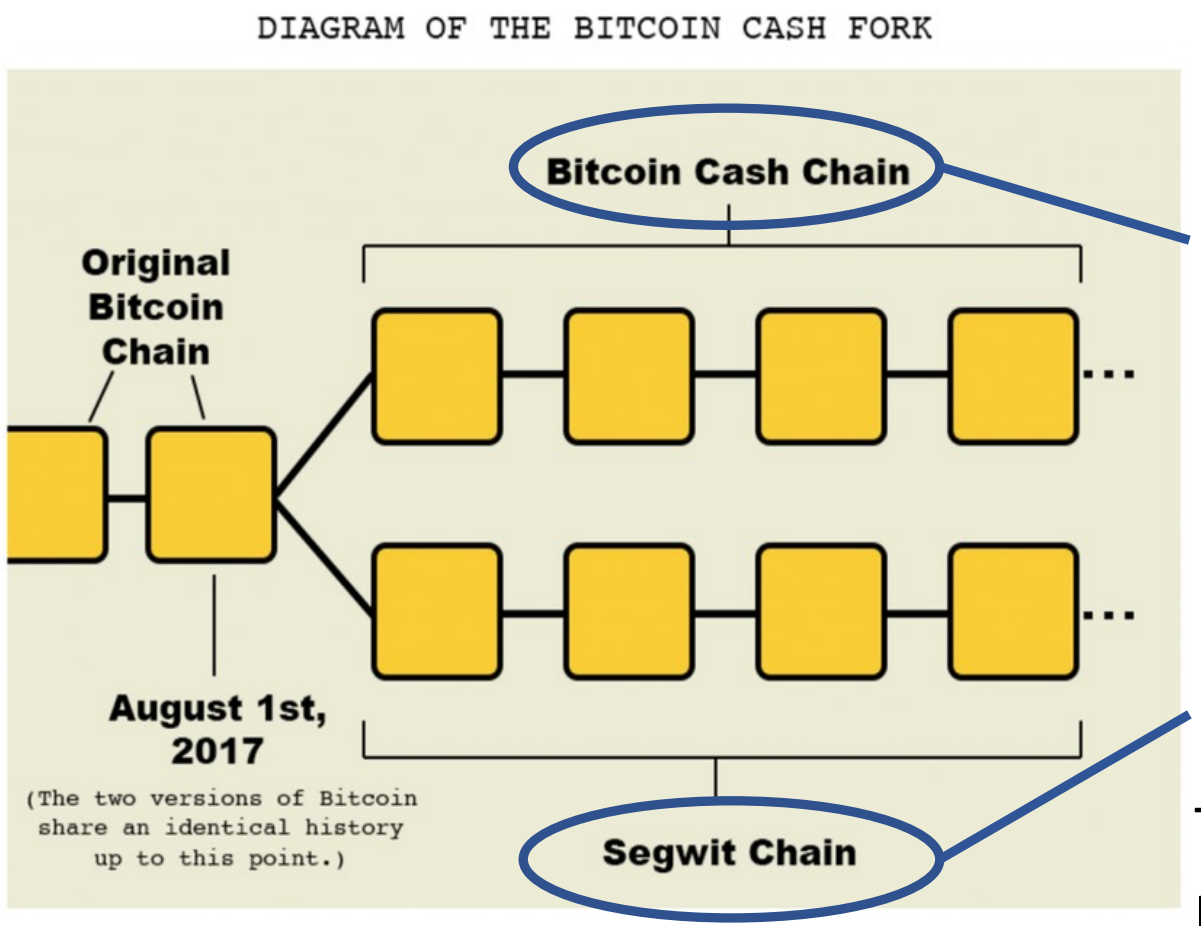
- Bitcoin XT was proposed in 2015 to increase th
- Bitcoin Classic was proposed in 2016 to increas
- "The Hong Kong Agreement" was a 2016 agree proposal established in December 2015 by Bitc
- SegWit2x was a proposed hard fork of the cryp block size by SegWit compromised with those v On November 8, 2017 the developers of SegW
- Bitcoin Unlimited advocates for miner flexibility different from Bitcoin Core in that the block size Unlimited proposal argue that from an ideologic

Case study: Bitcoin Cash

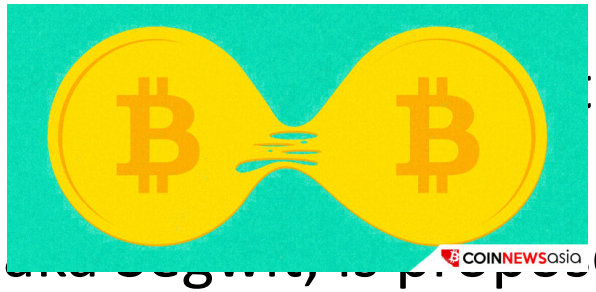
- Bitcoin Cash is a hard fork of the bitcoin blockchain that was born on 1 August 2017 (since block 478559)
  - Bitcoin holders own *equal amounts* of both bitcoin (BTC) and Bitcoin Cash (BCH) at the time they split
  - Bitcoin Cash increased block size from 1MB to 8MB
- However, it was supported by <u>only a limited number of miners</u>

Bitcoin Cash
Peer-to-Peer Electronic Cash

Bitcoin Cash Fork: Under the hood



Unsatisfied with BIP 91, some members of the Bitcoin community plan to create an ideal one. (Supported by several Chinese mining pools).



aka Segwit, is proposed on July 20, 2017

The overall Bitcoin mining power is divided

Figure by Coin News Asia

Bitcoin holders are happy!

Assessed on Jan 20 2020 on https://coinmarketcap.com/currencies/bitcoin-cash

Attempt 2:
Increase the mining rate

- Decrease the mining difficulty
- Much faster than 10 mins / block
- Will it cause problems ?

Wait...
why we target the 10 mins block time?

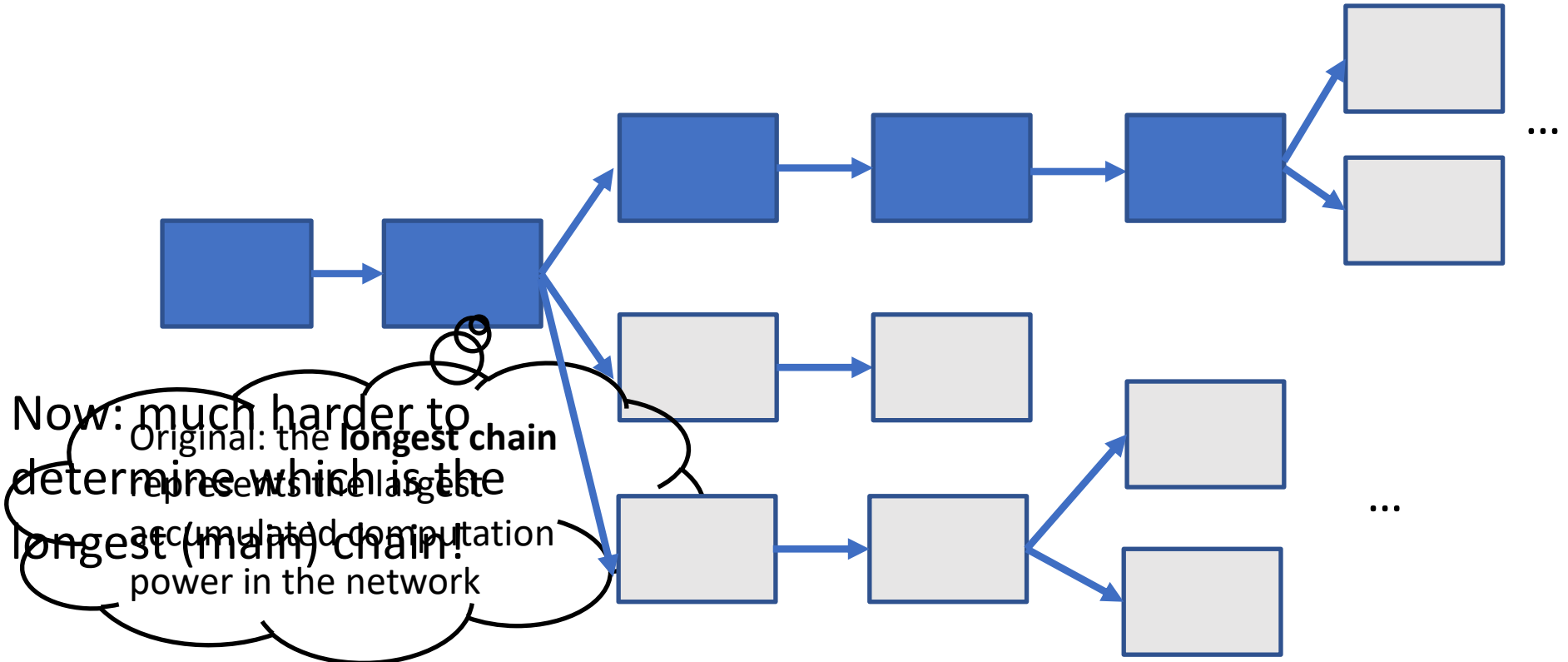
Satoshi Nakamoto said so

- It was introduced by Satoshi Nakamoto in the original Bitcoin paper
  - Not a concrete suggestion, but is the only reference existed
  - It works out fine, so people accept this block time parameter

- Recall: the mining difficult adjusts every 2016 blocks s.t. the ***average*** block time target is 10 mins

Potential problems with shorter block time

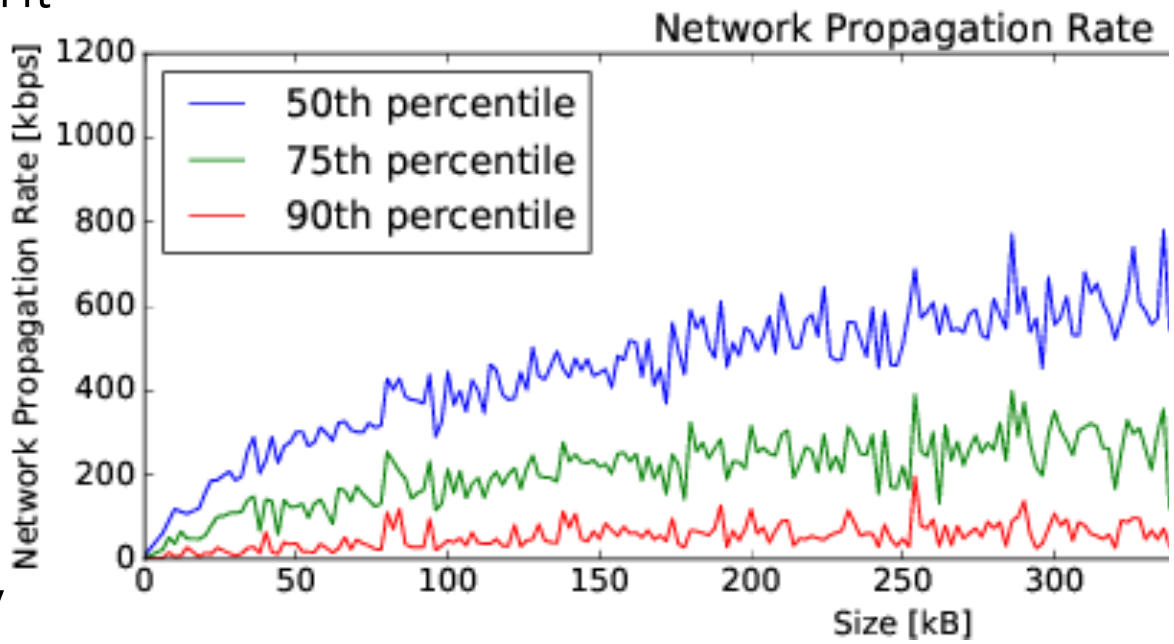
- There will be more and longer *forks* in a short time period



More …

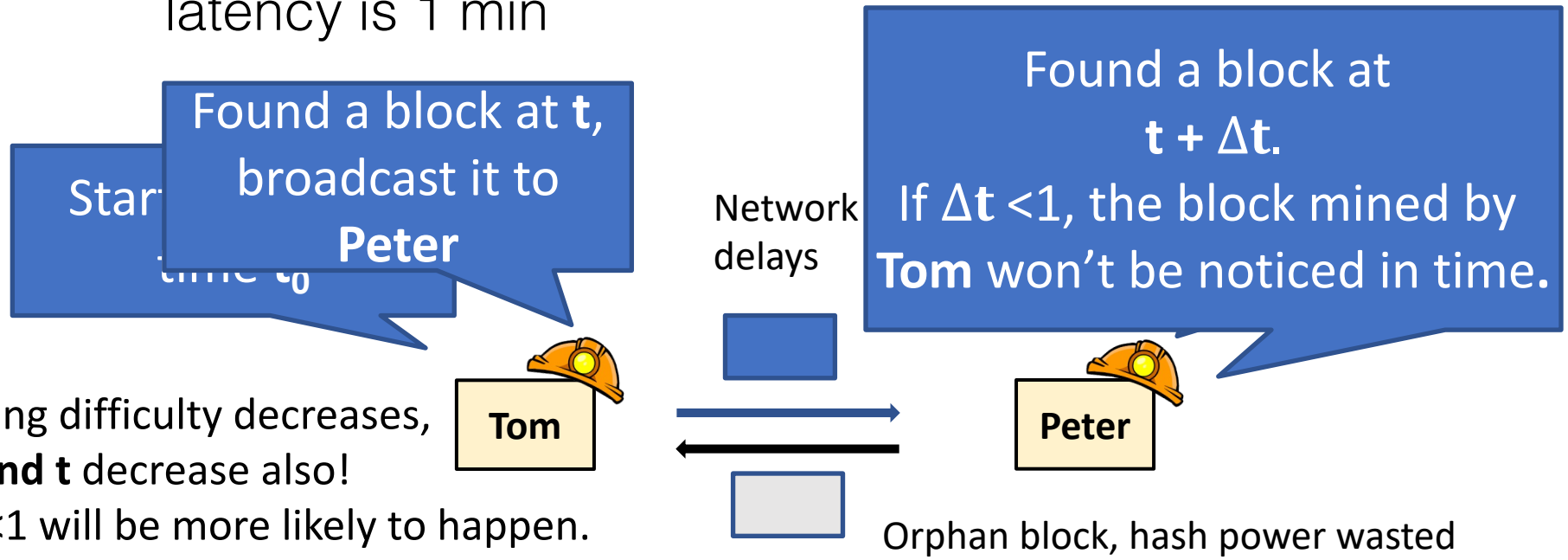
- Requires increased bandwidth

Previous measurement (2014-5) indicates that the 10%, median, and *90% block propagation* times are 0.8 seconds, 8.7 seconds, and 79 seconds respectively (540k block size)



More …

- A greater portion of the raw hash power will be wasted
  - Consider generating *orphan blocks* as the waste of hashes
  - Suppose the block interval is **t** (mins) and network latency is 1 min



It seems that both the two immediate solutions for Bitcoin have (security) problems

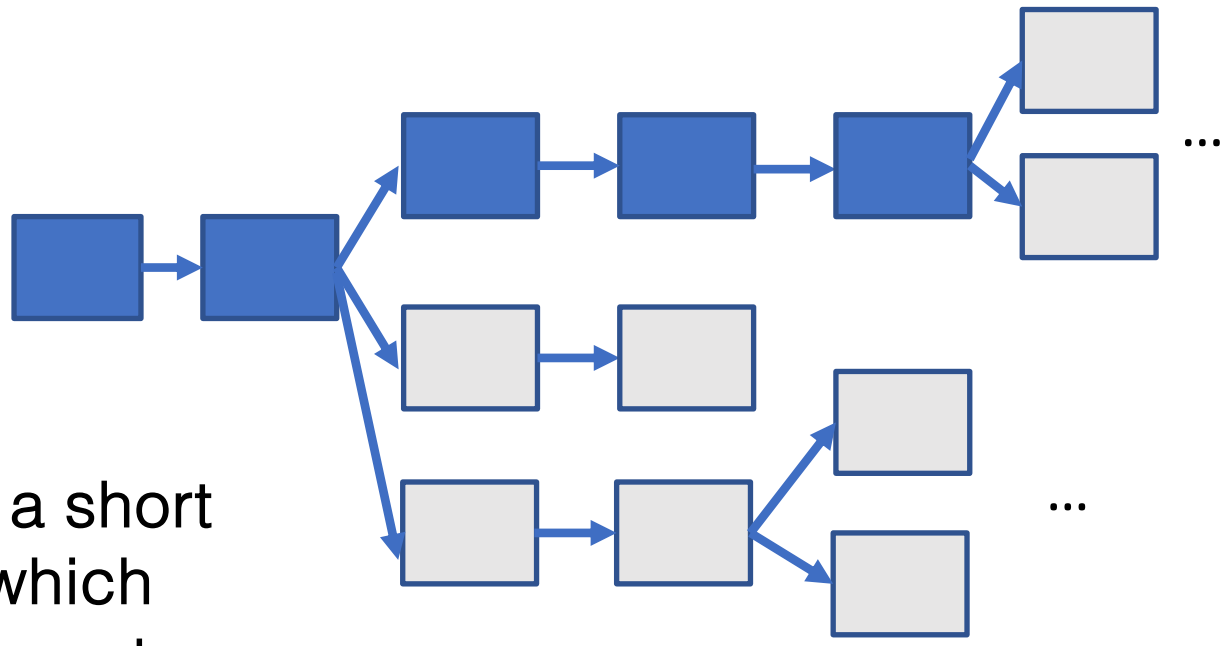
New consensus protocols are required

- **Incompatible** with the original Bitcoin design
  - Deployment needs hard fork
- New consensus strategies
  - GHOST (adopted by Ethereum)
  - Proof-of-Stake
- New structure for parallel blockchain processing
  - Sharding

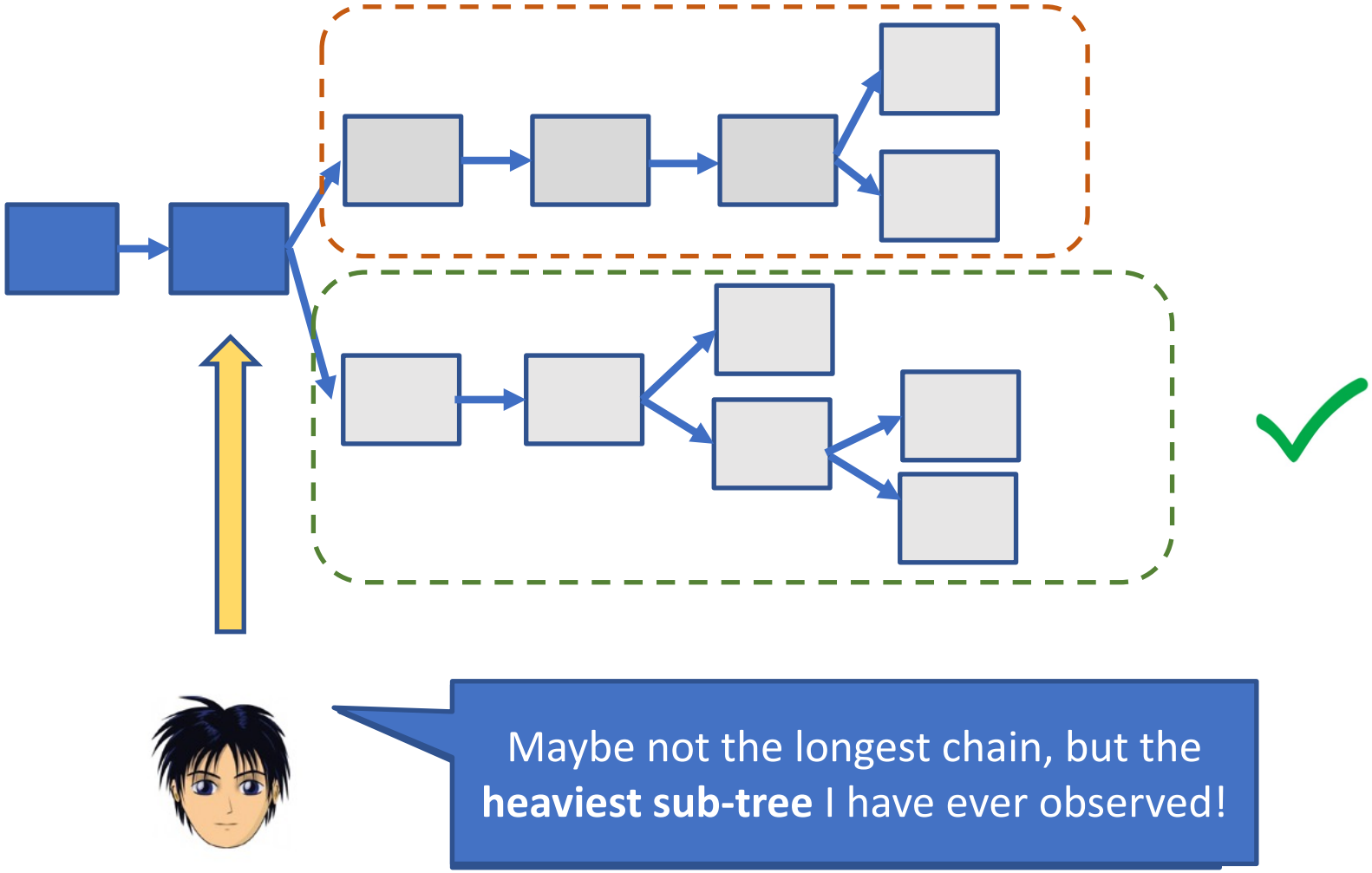
Secure High-Rate Transaction Processing in Bitcoin,
Yonatan Sompolinsky and Aviv Zohar,
in Proc. of FC, *2015*

Recall

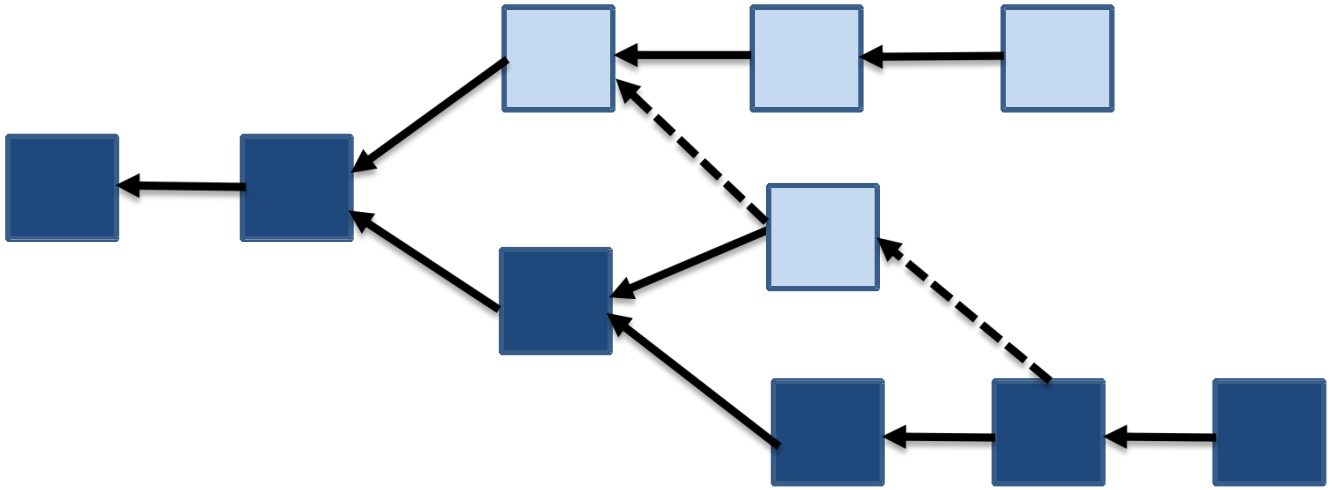
- What might happen if we increase the block mining rate?



More *forks* in a short time period, which makes the longest chain hard to find!


Maybe not the longest chain, but the **heaviest sub-tree** I have ever observed!

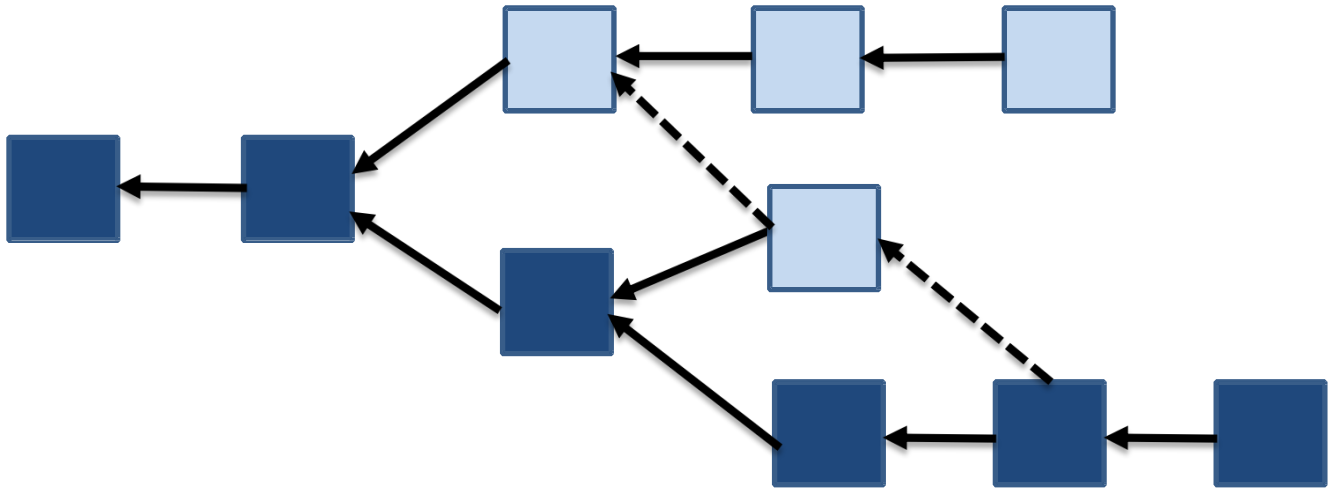
The GHOST Protocol



Greedy Heaviest Observed Sub-Tree: Allow frequent forks

- Blocks do not only reference their direct parent, but also *uncle* blocks
- Orphan blocks' transactions are not applied to the blockchain state, but their mining power still counts towards the "heaviest" chain

Uncle Blocks



An uncle is any block that is

- A child of a direct ancestor
- Not a direct ancestor itself
- Not already marked as an uncle of a direct ancestor

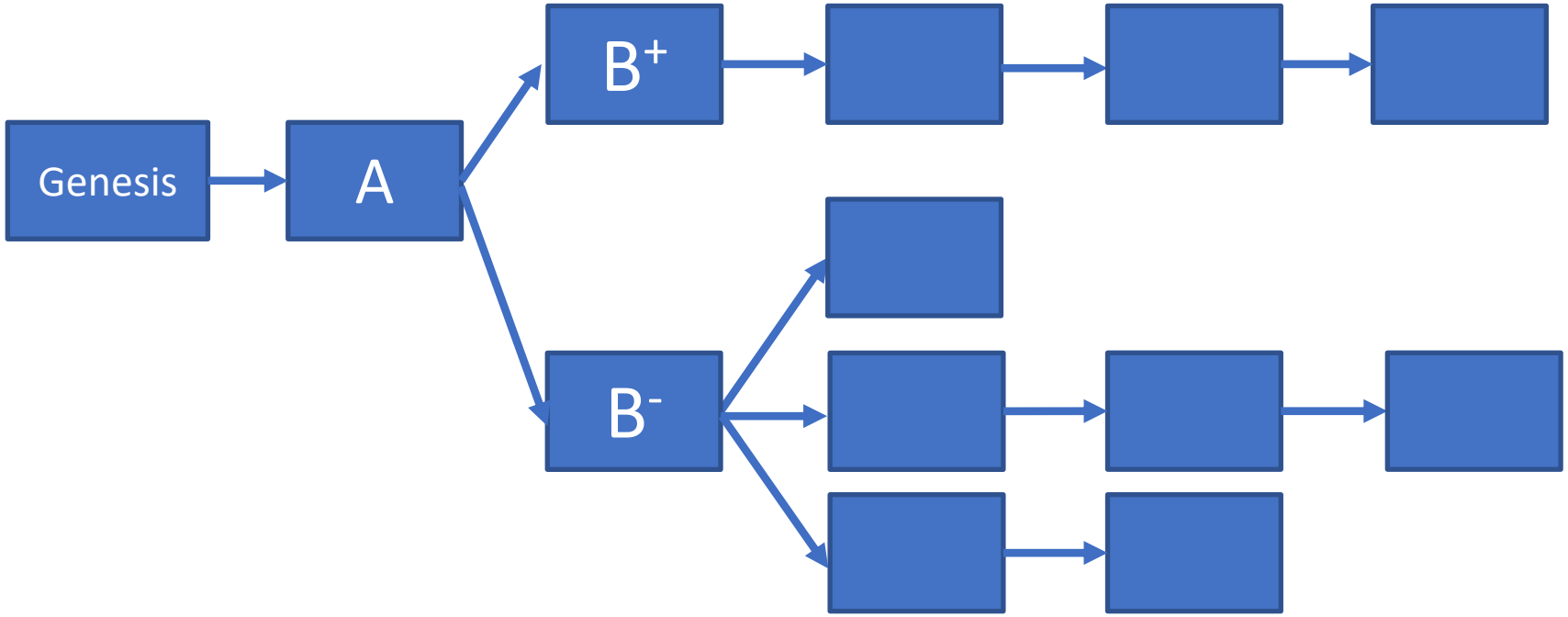
Uncle blocks get some of the **block reward** (but not transaction fees)

Subtle Changes in the Incentive Mechanism (compared to Bitcoin)

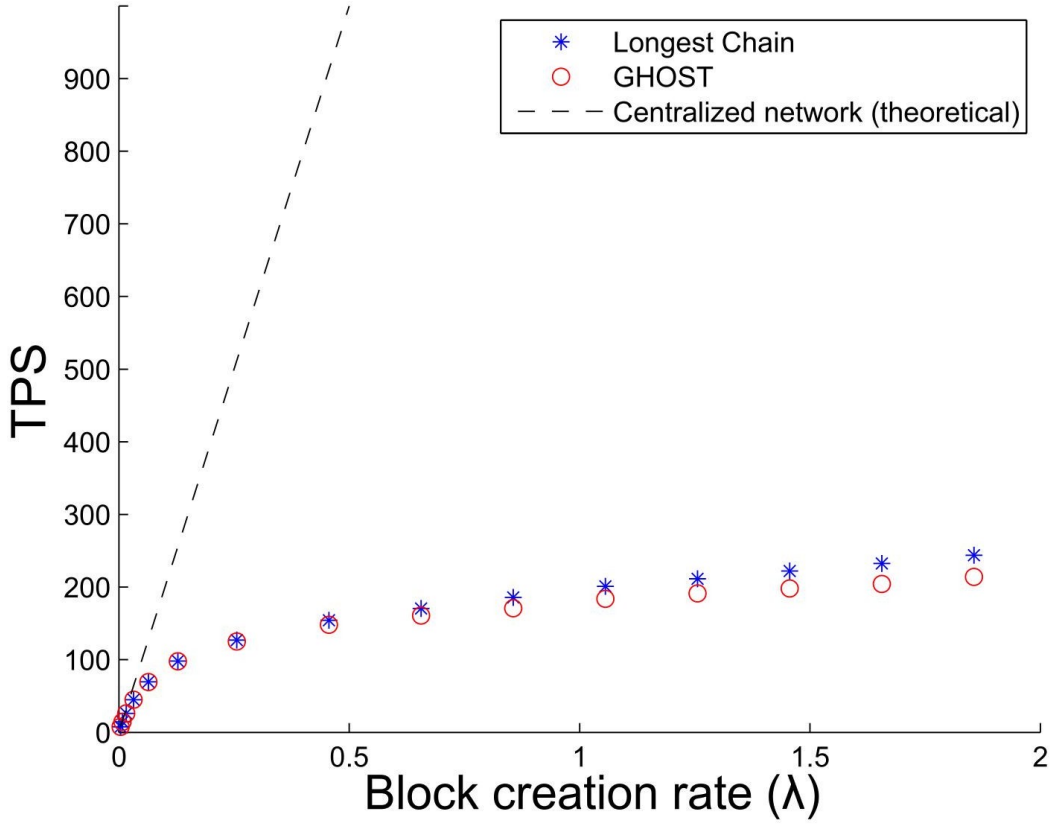
- Recall: Bitcoin only rewards the new block mined **in the main chain**
  - Orphan blocks in Bitcoin do not contribute to longest chain rule-based consensus
- But in GHOST, orphan blocks ("uncles") are counted when determining the heaviest sub-tree
  - Incentivizing their **honest-but-not-so-lucky** works
  - In Ethereum, each uncle block is rewarded with 7/8 of the static block reward = 2.625 ETH
    - A maximum of 2 uncles are allowed per block

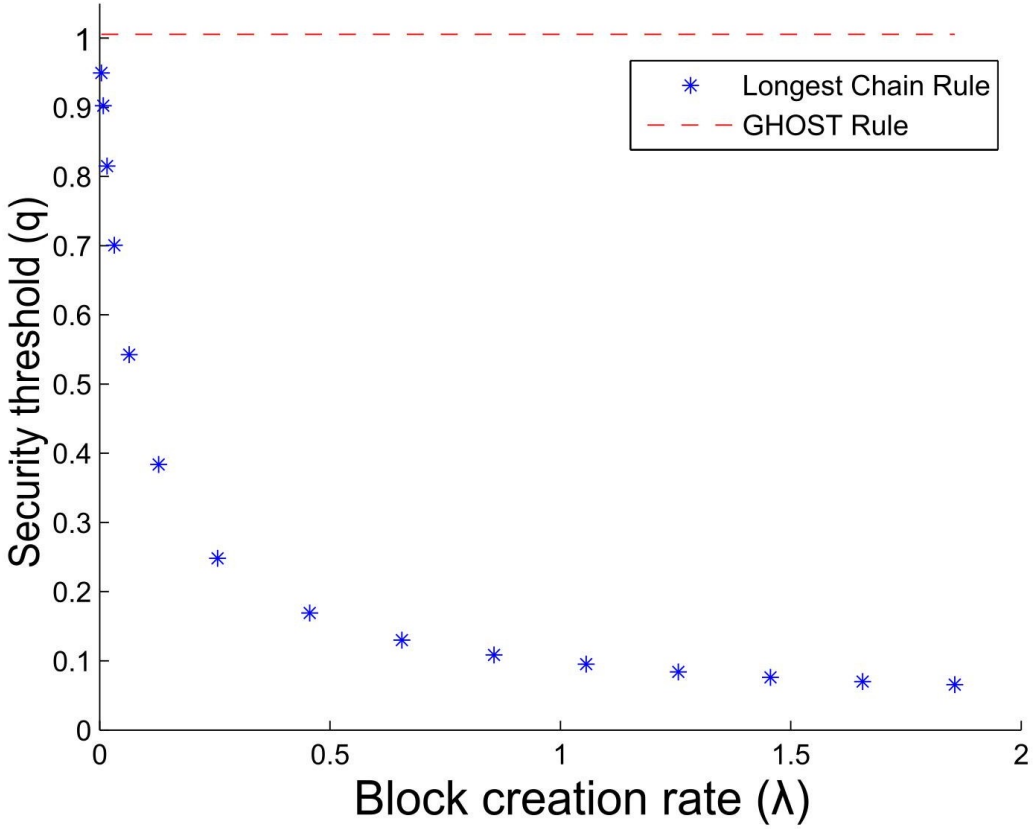
GHOST Protocol Demo

- Begin at the “genesis block”
- At every split, pick the *heaviest sub-tree*



Ghost: Formal Results





Loss in efficiency of network resources caused by following the GHOST rule is relatively small

Strength: Security does not degrade with higher block frequency

Selfish Mining

- Demonstrates that the economic incentives in the original Bitcoin paper are not sound
- "Majority is not Enough: Bitcoin Mining is Vulnerable", published in 2013 by Ittay Eyal and Emin Gün Sirer
- In 2020, "re-examined" in "Selfish Mining Re-Examined" by Kevin Negy, Peter Rizun, and Emin Gün Sirer

Ittay Eyal (Technion and IC3)

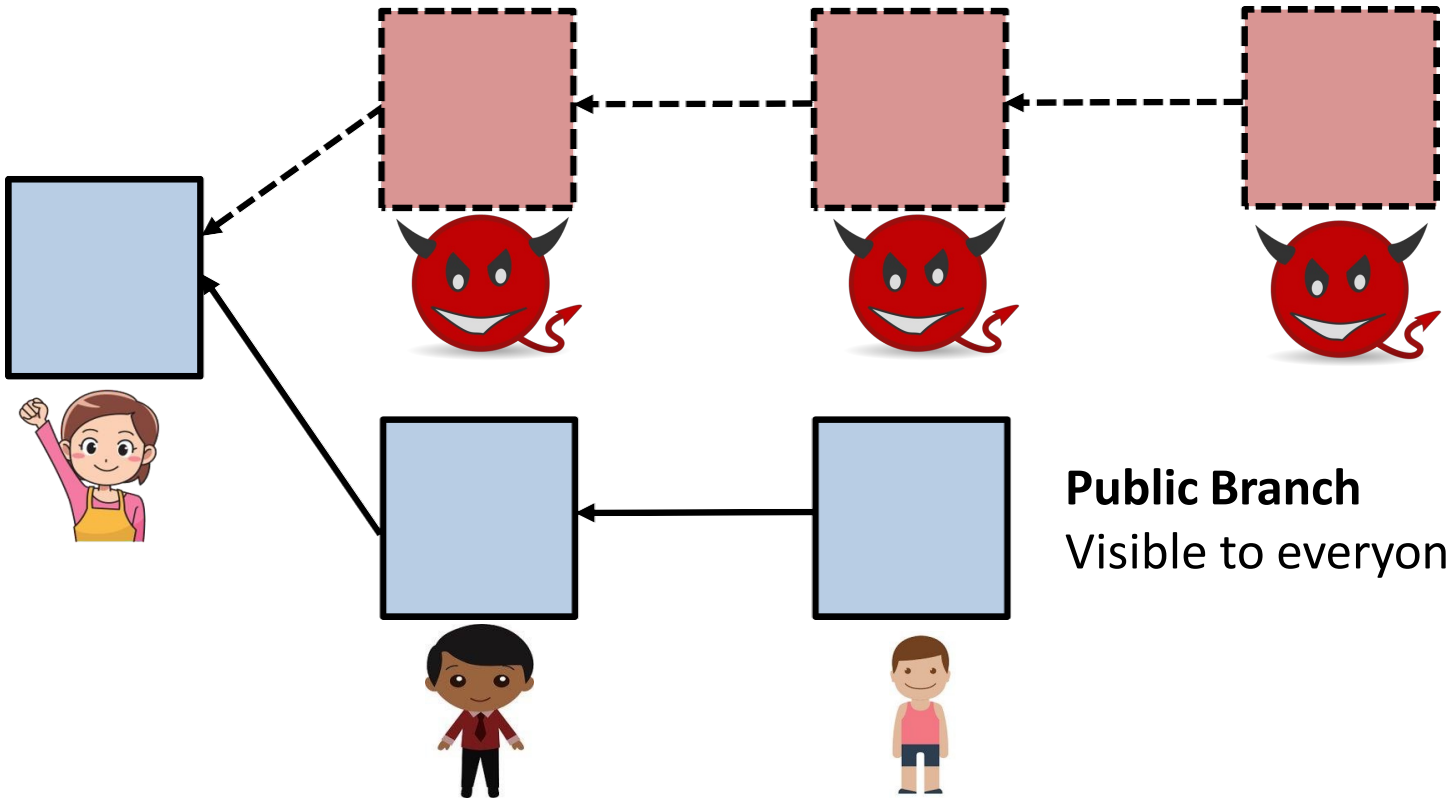
Kevin Negy (Cornell and IC3)

Emin Gün Sirer (Cornell, Ava Labs, and IC3)

Peter Rizun (Bitcoin Unlimited)

Selfish Mining

Idea: Selectively hide blocks you mine to gain an advantage



Private Branch Only visible to selfish miner(s)

Public Branch
Visible to everyone

Selfish Mining: Intuition

Goal 1: Keep mined blocks hidden as long as possible

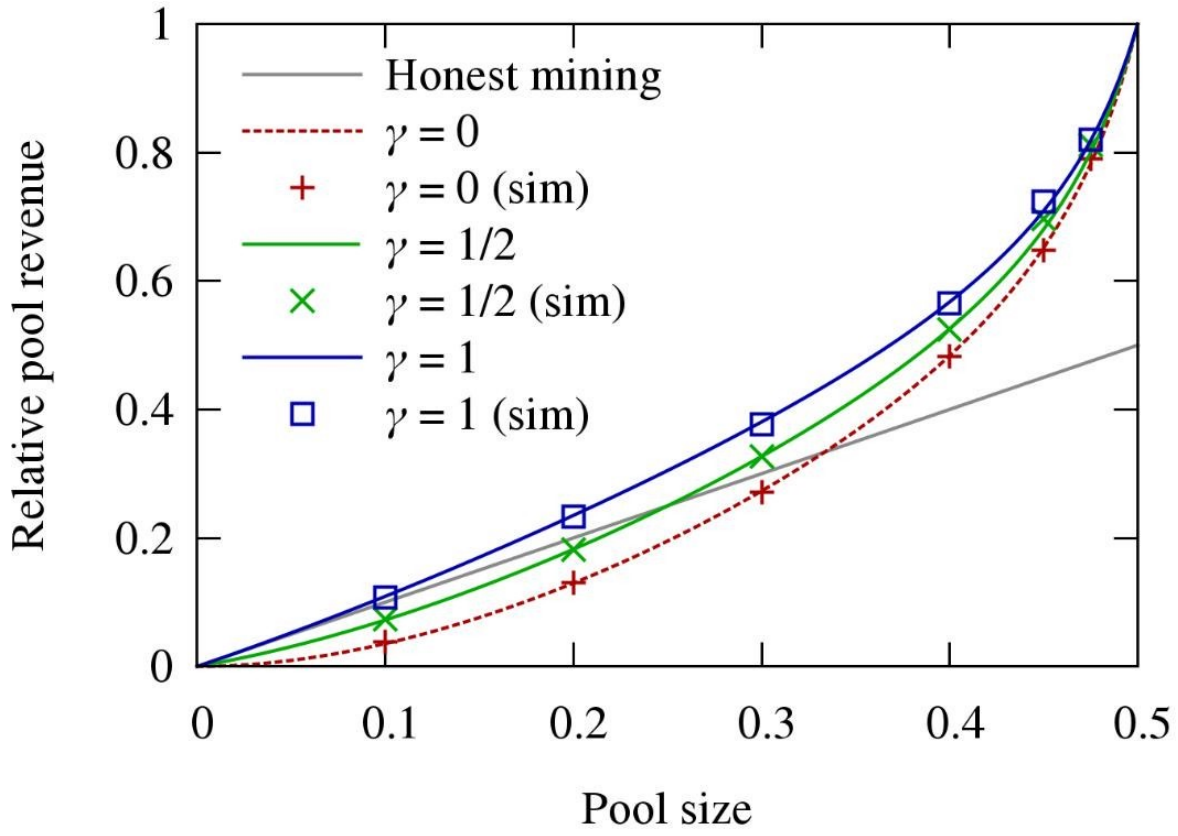
- Use "information asymmetry" to your advantage
- Other miners cannot mine on hidden blocks

Goal 2: Try to have two competing branches exist for as long as possible

- If there are multiple competing branches, honest mining power might be split between them

Approach: When honest miners publish a block, try to reveal one (or more) blocks from the private branch to "confuse" honest majority

Selfish Mining Revenue



- The paper presents both theoretical and simulated results
- γ is the fraction of honest miners that choose to mine on the selfish chain
- If all miners pick the selfish chain, selfish-mining is always profitable!

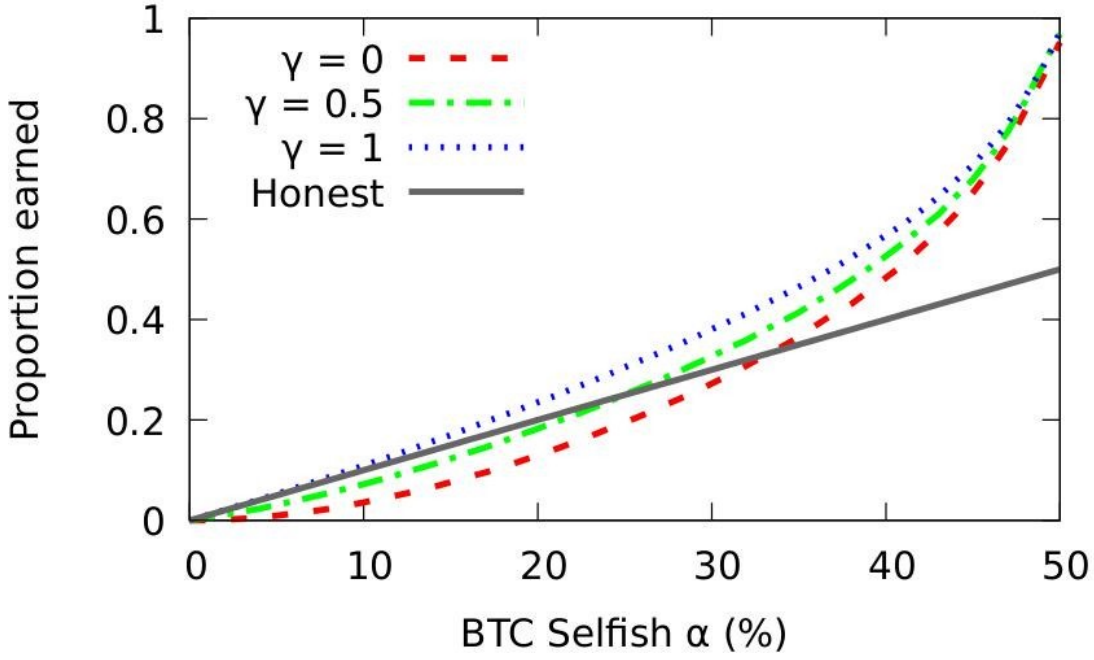
Criticism of Selfish Mining

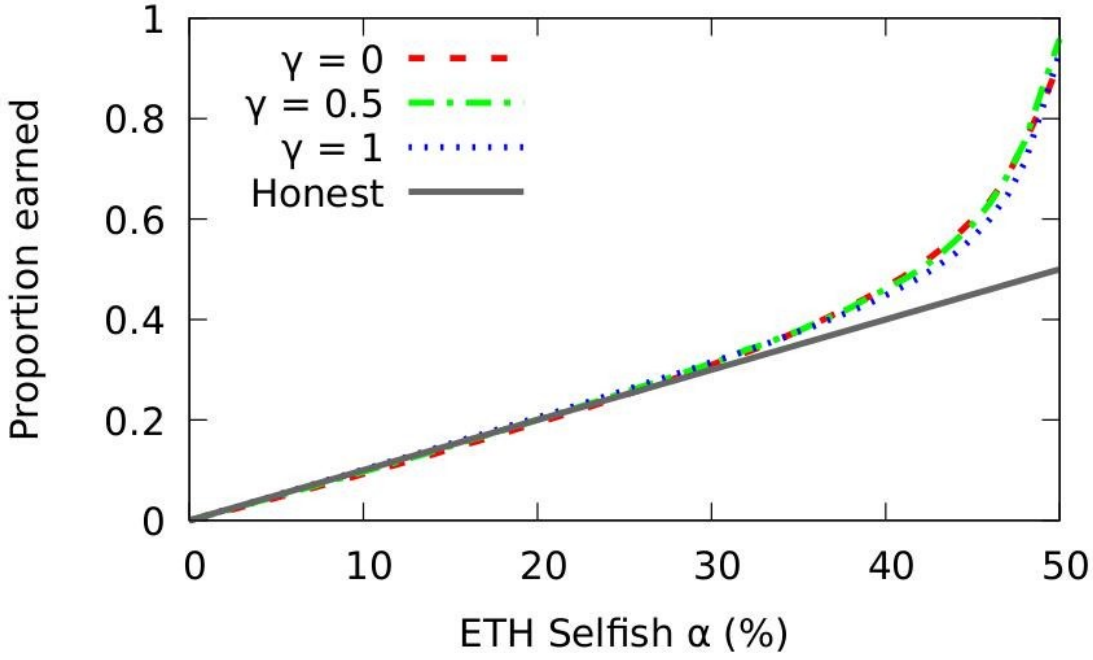
1. While selfish miners may win more blocks than expected, overall revenue could still be lower because of high orphan rate
   - If more blocks are orphaned, less blocks receive block rewards
   - Difficulty adjustment will resolve this somewhat, original analysis does not fully take difficulty adjustment into account

2. High sunk costs at first

   Needs to maintain selfish mining for weeks before it becomes profitable

Selfish Mining and GHOST





GHOST almost nullifies the downsides of selfish mining, because even orphan blocks get some reward!

Related References

- Leslie Lamport, Robert Shostak, and Marshall Pease. “The Byzantine Generals Problem”, ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3, Pages 382-40, 1982.
- Miguel Castro and Barbara Liskov. “Practical Byzantine Fault Tolerance”, in Proc. of OSDI’ 99.
- Kyle Croman, Christian Decker, Ittay Eyal et al. “On Scaling Decentralized Blockchains (A Position Paper)”, in Proc. of BITCOIN'16.
- Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. “Bitcoin-NG: A Scalable Blockchain Protocol”, in Proc. of USENIX NSDI ’16.
- Yonatan Sompolinsky and Aviv Zohar. “Secure High-Rate Transaction Processing in Bitcoin”, in Proc. of FC’ 15.