MONASH University

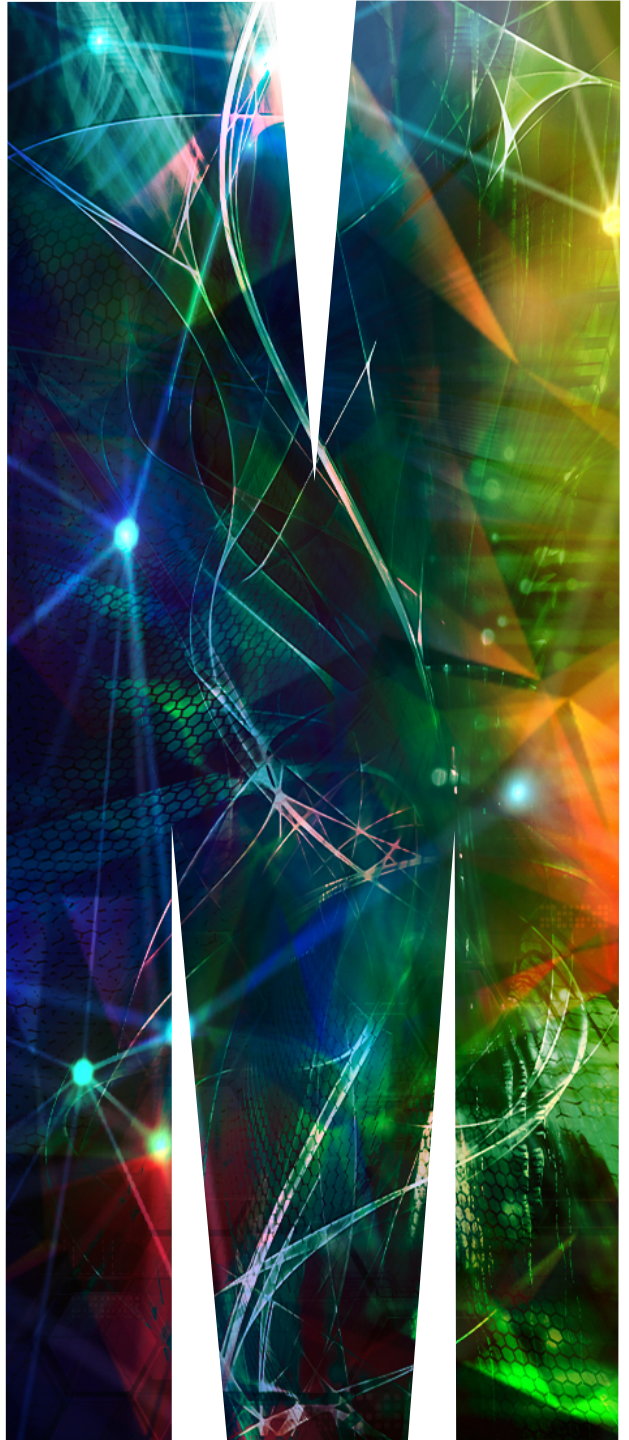
FIT5214: Blockchain

Lecture 9: Byzantine Agreement

Lecturer: Rafael Dowsley

rafael.dowsley@monash.edu

https://dowsley.net



Unit Structure

- Lecture 1: Introduction to Blockchain
- Lecture 2: Bitcoin
- Lecture 3: Ethereum and Smart Contracts
- Lecture 4: Proof-of-Work (PoW)
- Lecture 5: Attacks on Blockchains
- Lecture 6: Class Test/Alternatives to PoW
- Lecture 7: Proof-of-Stake (PoS)
- Lecture 8: Privacy
- Lecture 9: Byzantine Agreement
- Lecture 10: Blockchain Network
- Lecture 11: Payment Channels
- Lecture 12: Guest Lecture

Unit Structure

- Lecture 1: Introduction to Blockchain
- Lecture 2: Bitcoin
- Lecture 3: Ethereum and Smart Contracts
- Lecture 4: Proof-of-Work (PoW)
- Lecture 5: Attacks on Blockchains
- Lecture 6: Class Test/Alternatives to PoW
- Lecture 7: Proof-of-Stake (PoS)
- Lecture 8: Privacy
- Lecture 9: Byzantine Agreement
- Lecture 10: Blockchain Network
- Lecture 11: Payment Channels
- Lecture 12: Guest Lecture

Learning outcome:

Have basic understandings on BFT protocols.

Recap: consensus

Definition 1. (Consensus.) There are n nodes, of which at most f can be malicious, i.e., at least $n - f$ nodes are correct. For all $i \in [1, n]$, node i starts with an input value v_i. The nodes must decide for one of those values, satisfying the following properties:

- **Agreement:** All correct nodes decide for the same value.

- **Termination:** All correct nodes terminate in finite time.

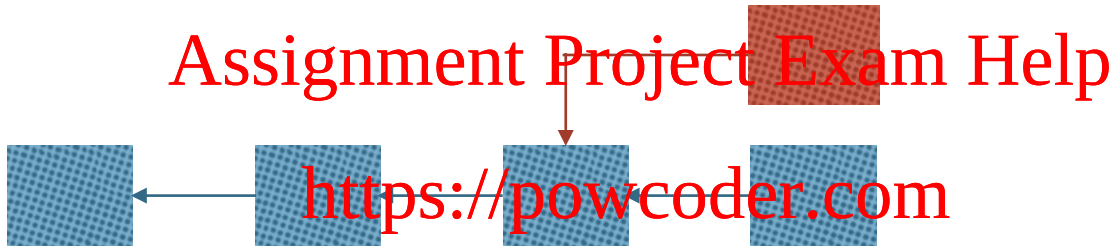
- **Validity:** The decision value must be the input value of a node.

In Bitcoin, the number of nodes does not matter, what matters is the computing power.

Recap: safety and liveness

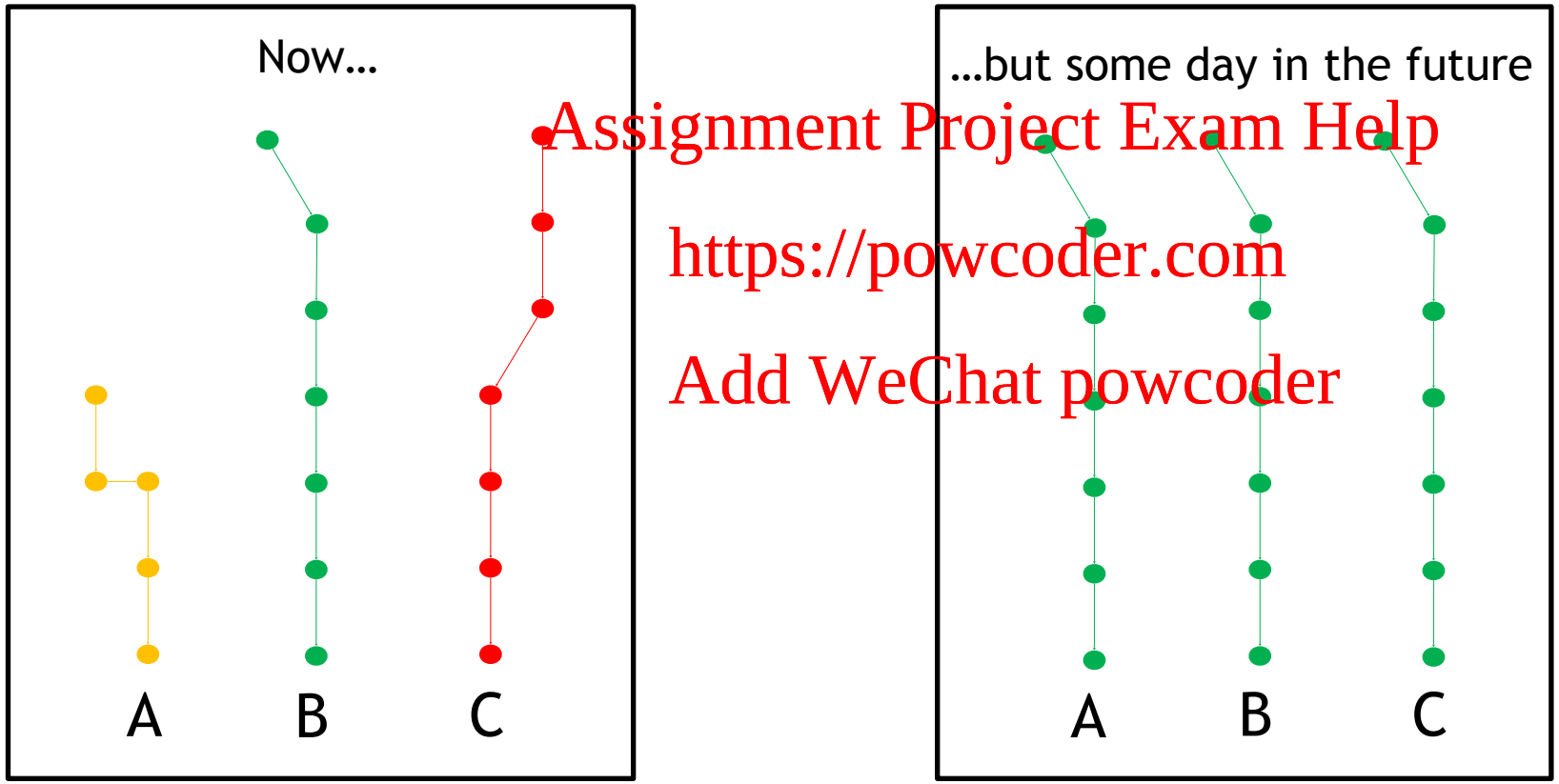
- The **safety** of the global state is ensured by the consensus
  - The entire *valid* transaction history can be *agreed by all* correct nodes. Agreed values cannot be changed later on. (Agreement & Validity)
- The **liveness** of the system is ensured by the consensus
  - All nodes can *terminate* their process and reach a conclusion. (Termination)

Recap: Agreement cannot be reached

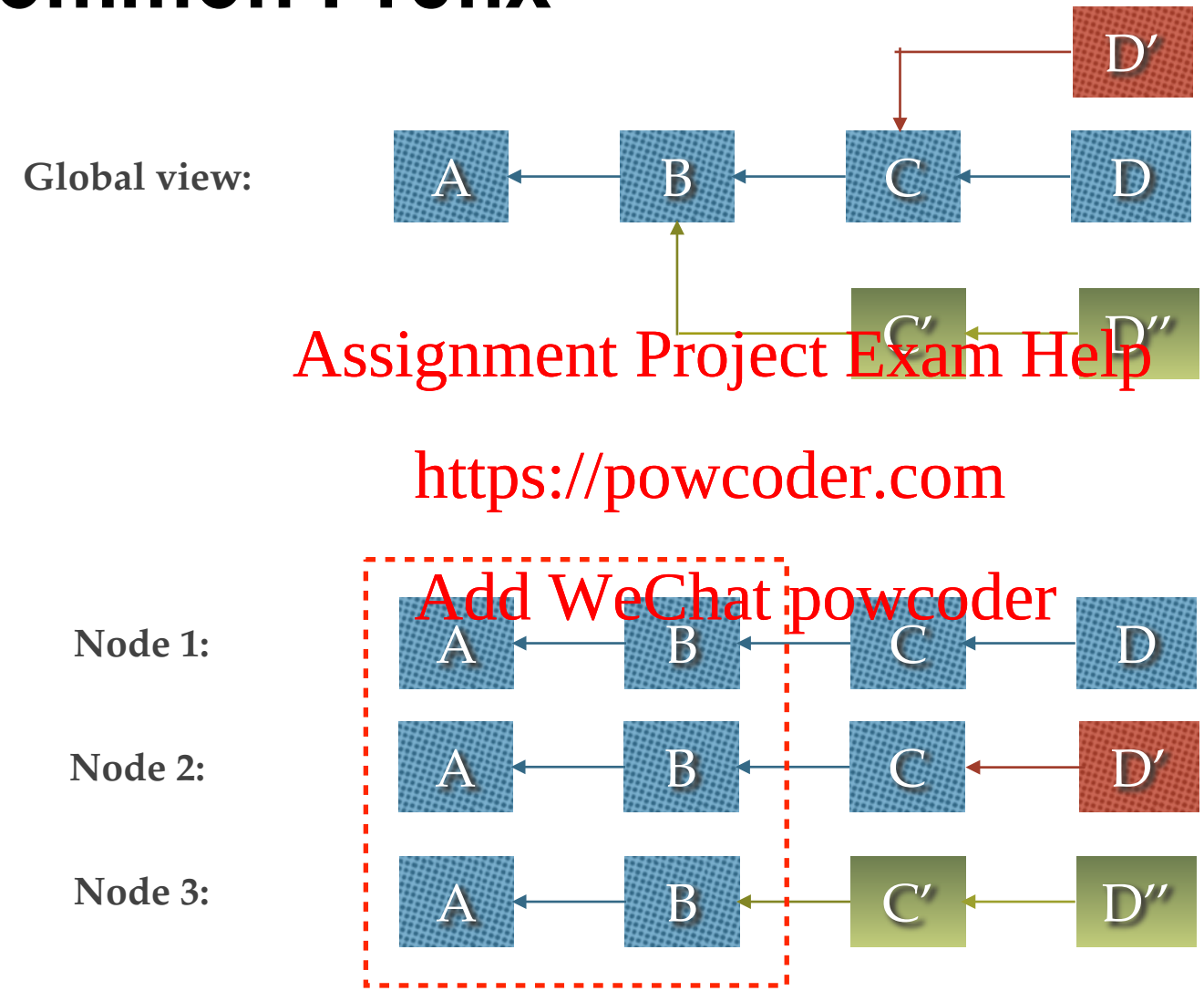


Recap: Eventual Consistency

Eventually, all the nodes will agree on the same blockchain



Recap: Common Prefix

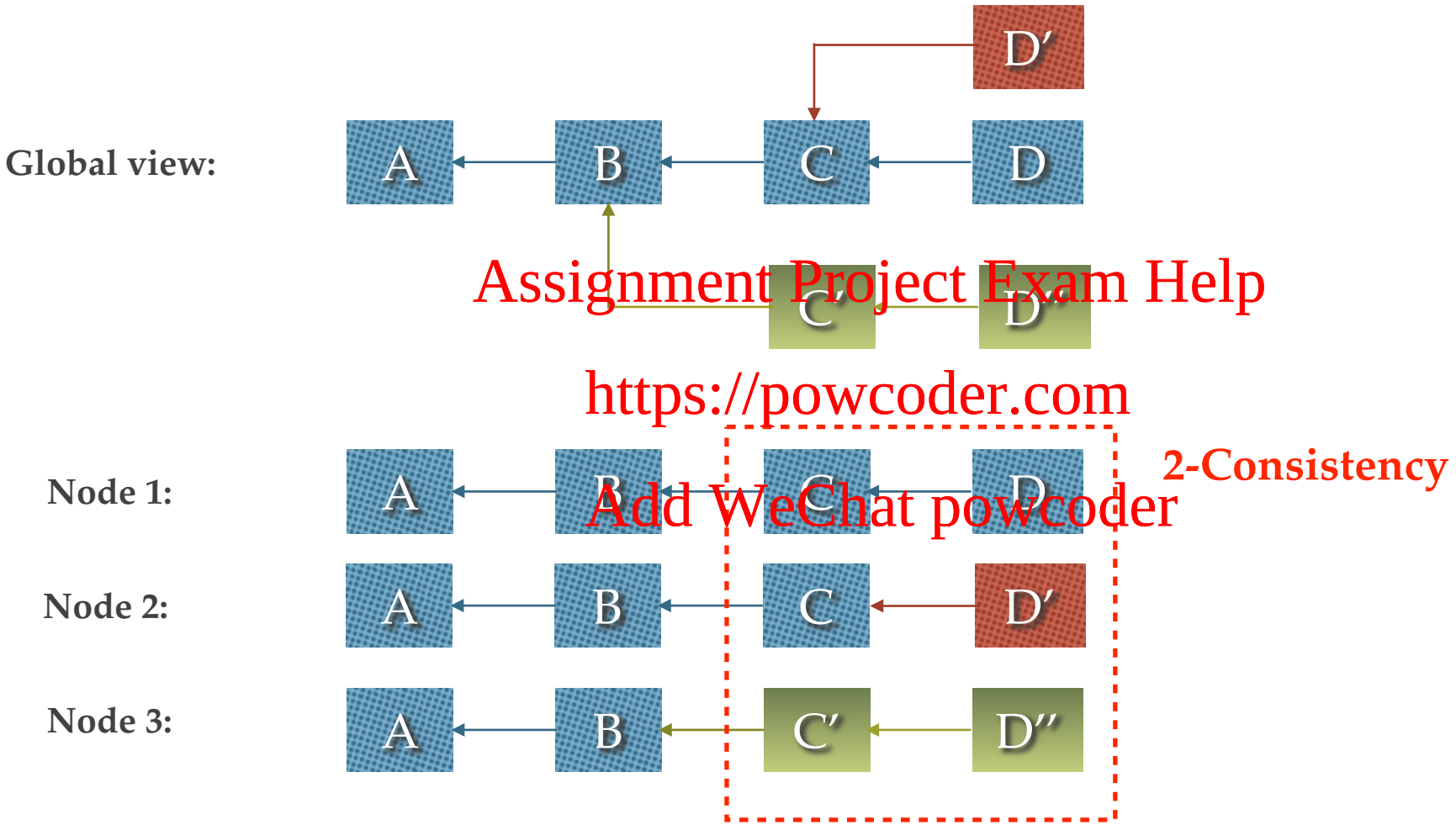


Recap: PoW Consensus

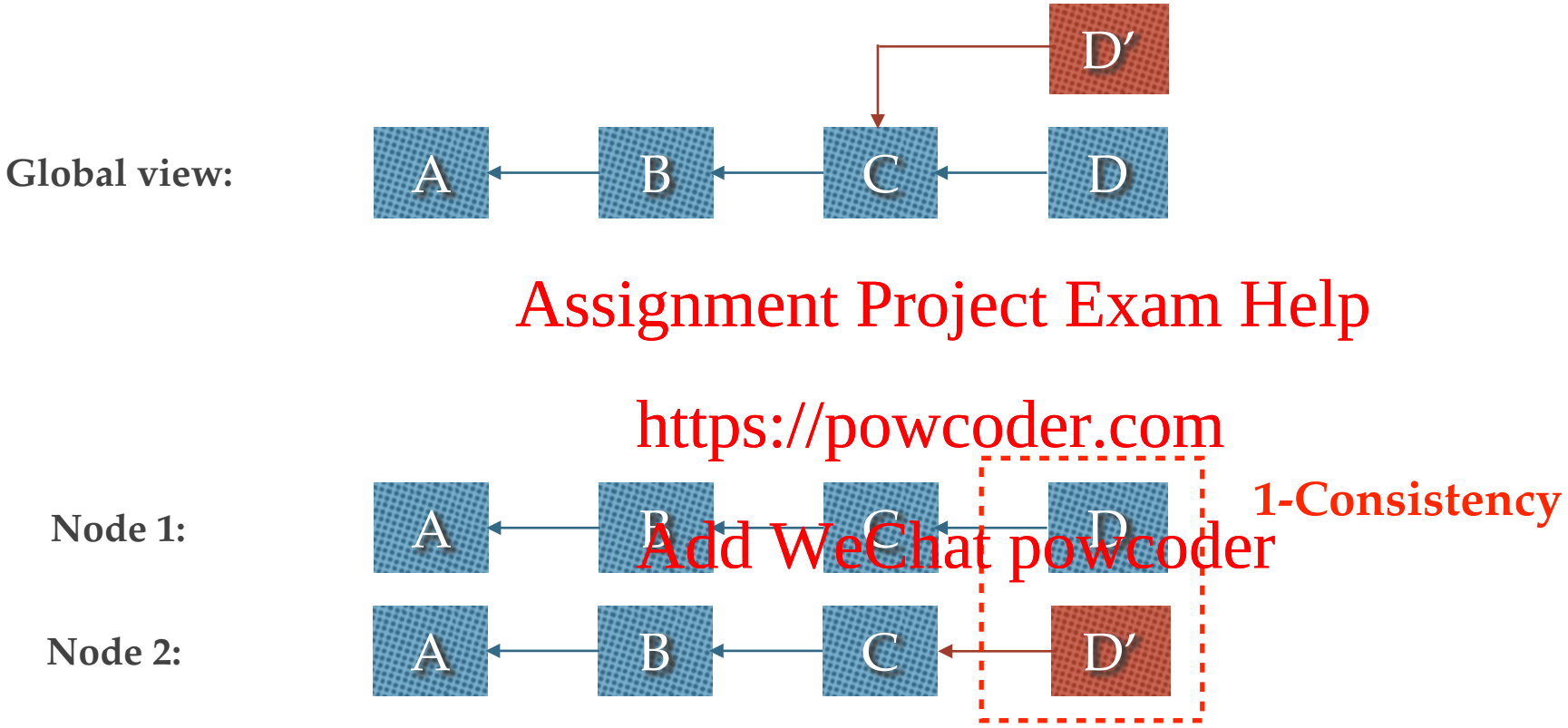
Assuming a synchronous network, PoW guarantees eventual consistency, or more actually, nodes in PoW agrees on a *Common Prefix* of the blockchain.

T-consistency **is used to quantify the quality of the blockchain consensus — after cutting down the last T blocks, all nodes should agree on the rest of the blockchain.**

Recap: T-Consistency



Recap: T-Consistency



The best consistency is 0-consistency, which can be provided by traditional consensus protocols (Week 9)

Problem

Computer systems provide crucial services, **but they may fail!!!!**


Nuclear power plant

- software errors
- power outage
- natural disasters
- hardware failures
- …


boeing 777

Replication

Replication is a technique used for tolerating faults. Each replica is a state machine, which is a deterministic program that:

- receives an input;
- changes its state according to the input;
- produces an output.

By replicating machines with a shared state, we can tolerate faults when some replicas are not available. This is called crash fault tolerance **(CFT)** — systems tolerate faulty nodes that are somehow not available.

Replication can be passive or active.

Passive replication

Passive replication is also called Primary-Backup (PB) or master-slave:

- Clients talk with the primary server, that sends the operations and checkpoints to the backups.
- If the primary crashes, one of the backups take over.

Passive replication

An example:

This is replication to tolerate a potential fault!

Active replication

Active replication is also called **State Machine Replication** (SMR):

- All servers execute the same sequence of operations, so they are always synchronised.

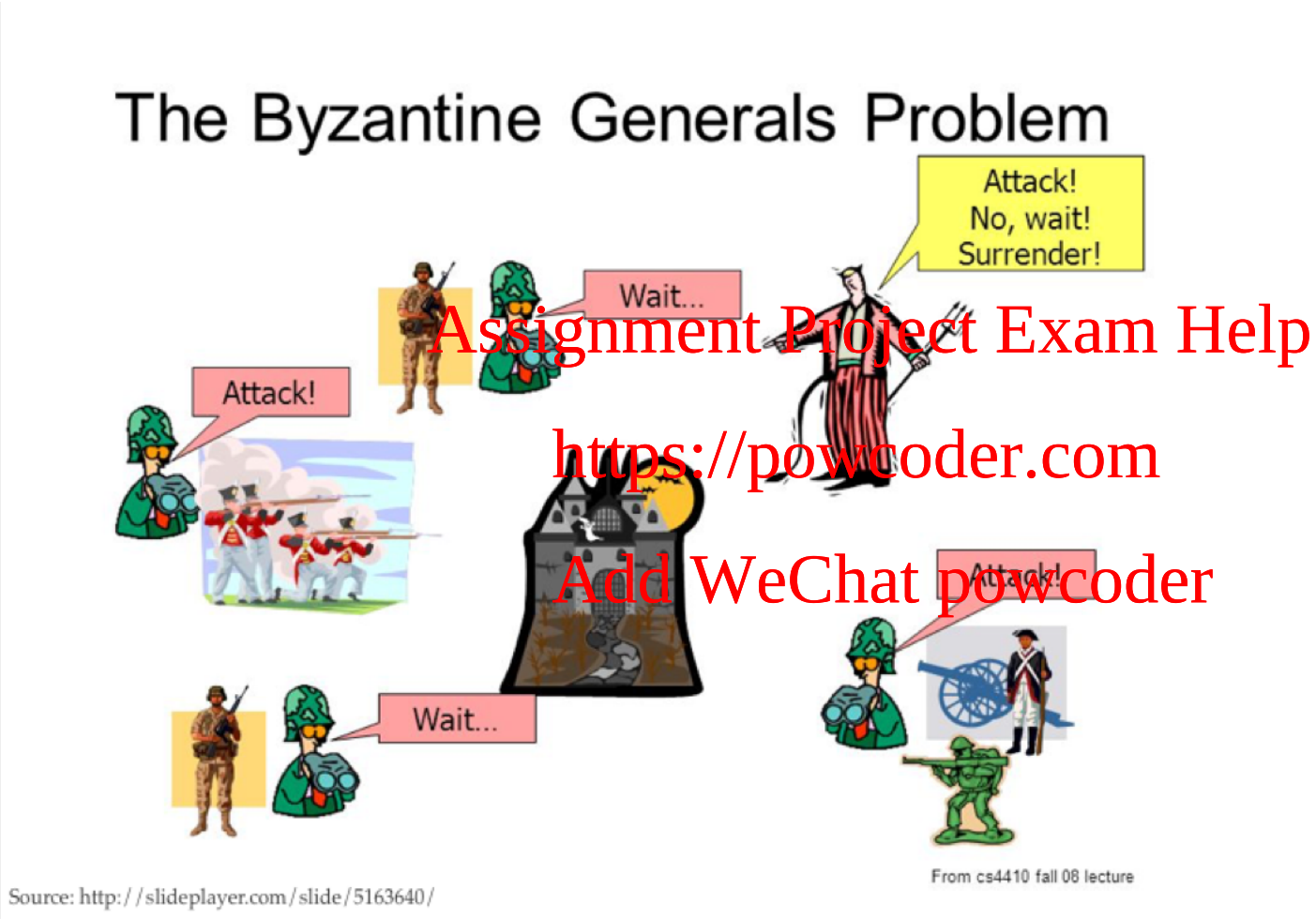
Replication with malicious server

An example:

I will just play at home…

What if we have someone malicious?

Byzantine Fault Tolerance

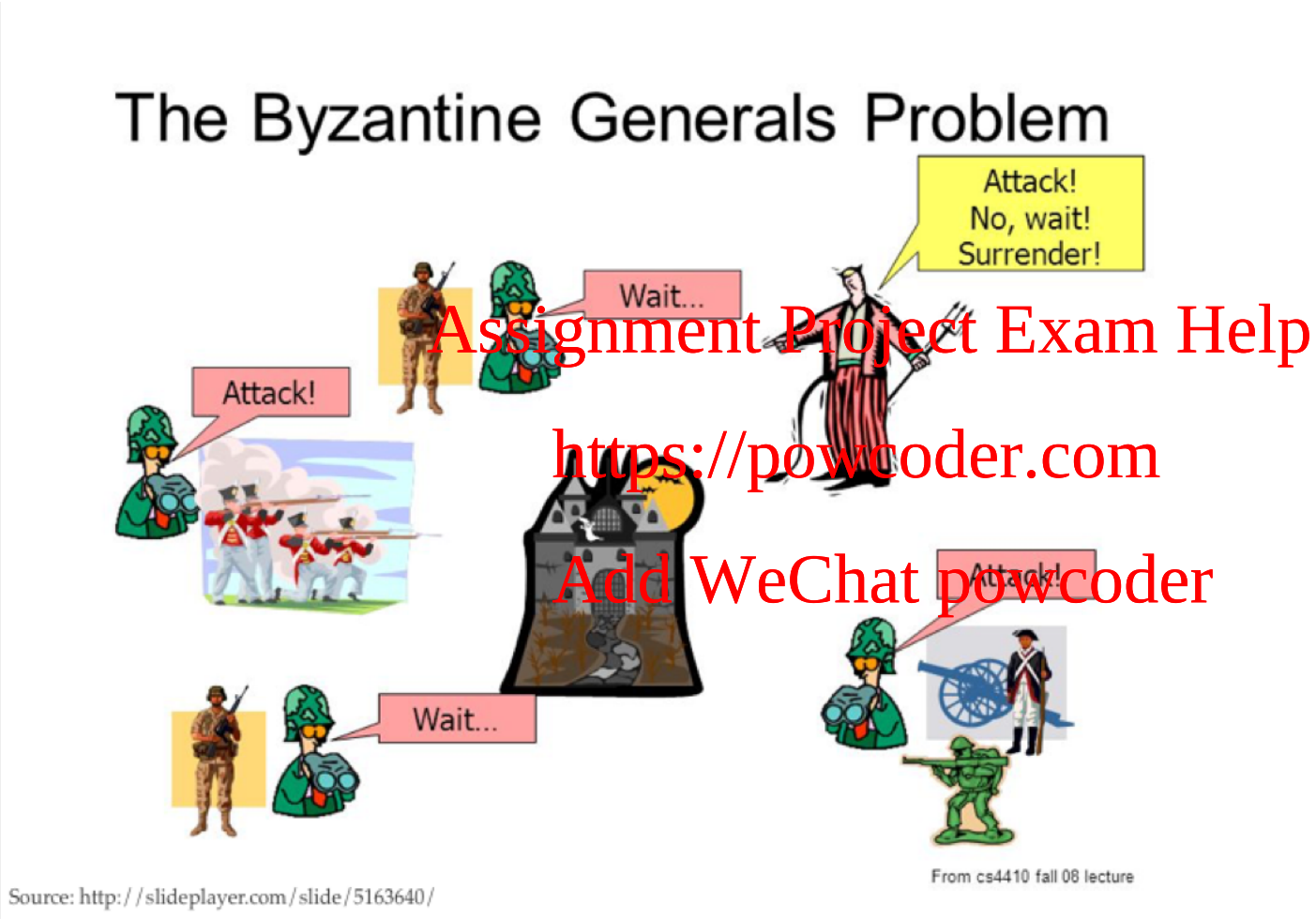


Source: http://slideplayer.com/slide/5163640/

From cs4410 fall 08 lecture

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

Byzantine Fault Tolerance



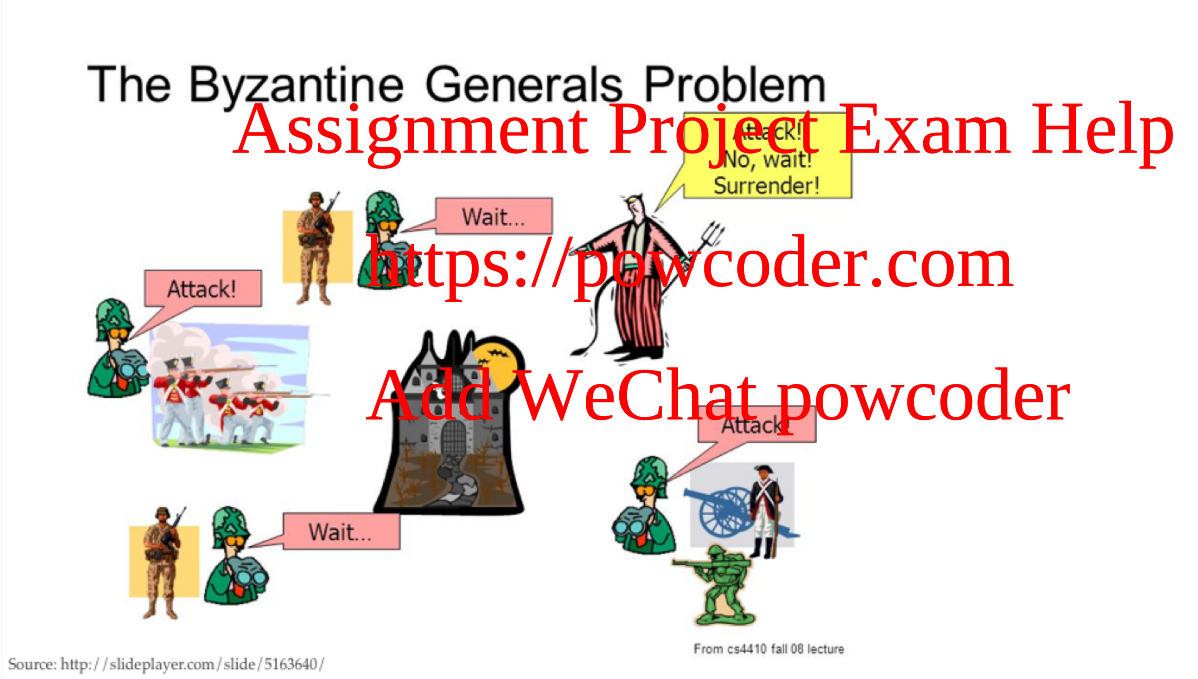
Group discussion:

If majority wins, can all honest parties reach the same agreement to attack or wait?

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

Question:

If there are *f* malicious (or Byzantine) nodes, what is the minimum number of total nodes *n* to ensure the safety and liveness?



Group discussion:

Can you try to work out the problem?

Answer: it depends …

Network synchrony

Asynchronous network:

There are no bounds on the amount of time a node might take to complete its work and then respond with a message. Therefore it's not possible to say whether a processor has crashed or is simply taking a long time to respond.

Answer: Byzantine agreement CANNOT be solved if f > 0 under asynchrony.

Michael J. **Fischer**, Nancy A. **Lynch**, and Mike **Paterson**. Impossibility of distributed consensus with one faulty process. J. ACM, 32(2):374–382, 1985.

FLP impossibility

Fischer, Lynch, Paterson (FLP) result:

There is no **deterministic** algorithm which **always solves** the Byzantine Generals' Problem in the **asynchronous model**, with $f > 0$.

Michael J. **Fischer**, Nancy A. **Lynch**, and Mike **Paterson**. Impossibility of distributed consensus with one faulty process. J. ACM, 32(2):374–382, 1985.

Network synchrony

Partially synchronous network:

Type A:
An upper bound Δ on message delivery time exists, but we do not know what it is a priori.

Type B:
We know Δ, but the message system is sometimes unreliable, delivering messages late or not at all.

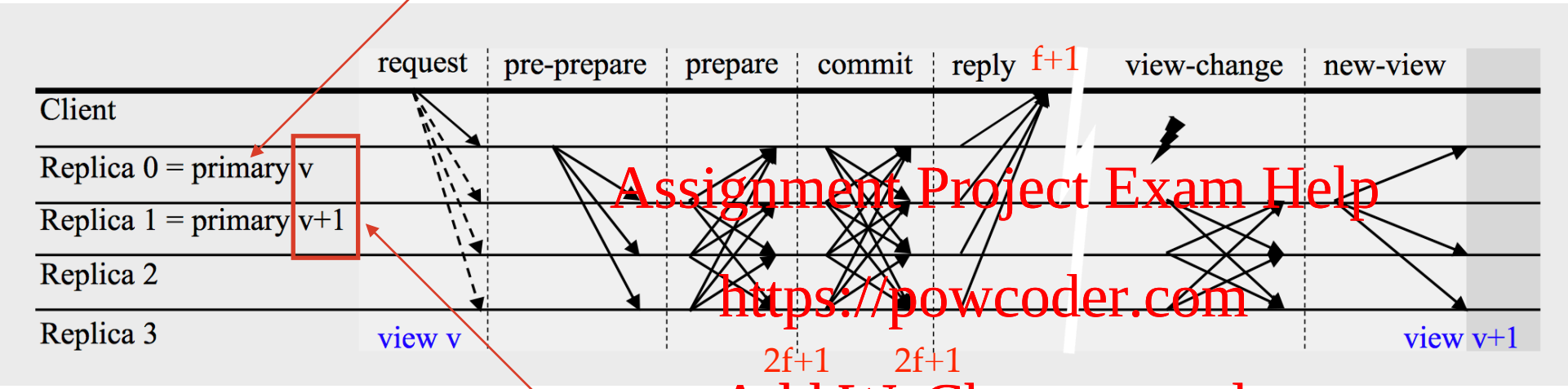
Answer: Byzantine agreement can be solved with $f < n/3$ under partial synchrony.

Byzantine agreement (BA)

Byzantine agreement (BA) systems, also known as Byzantine fault tolerant (BFT) systems, provide solutions to the Byzantine generals problem.

PBFT: n=3f+1

A primary node, is also called a leader.
PBFT is a leader based BFT protocol.

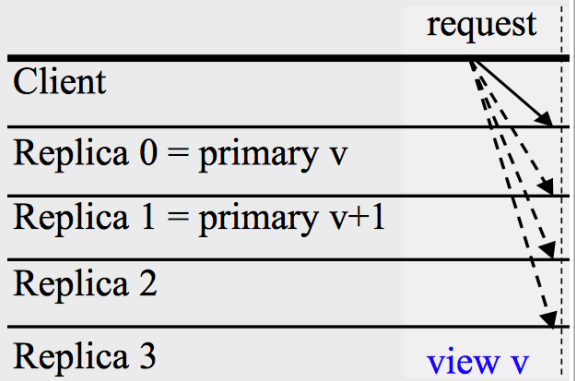


View represents the current state of the system.

- Partial synchrony.
- Deterministic termination.
- Deterministic agreement.
- **Not scalable.**

Miguel Castro, Barbara Liskov: Practical byzantine fault tolerance. OSDI 1999, pp. 173-186.
Miguel Castro, Barbara Liskov: Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. 20(4): 398-461 (2002)

PBFT: n=3f+1



Request:

To initiate agreement, a client c sends a request of the form $< REQUEST, o, t, c >_{\sigma_c}$ to the primary, but is also prepared to broadcast it to all replicas if replies are late or primaries change.

$< REQUEST, o, t, c >_{\sigma_c}$ specifies the operation to execute o and a timestamp t that orders requests of the same client. Replicas will not re-execute requests with a lower timestamp than the last one processed for this client, but are prepared to resend recent replies.

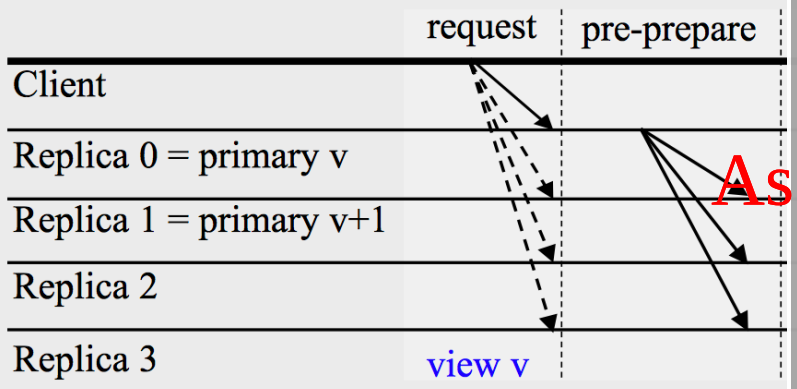
Operation

Client signature

Client ID

timestamp

PBFT: n=3f+1



Pre-prepare:

The primary of view v puts the pending requests in a total order and initiates agreement by sending $<$ PRE-PREPARE, $v, n, m >_{\sigma_p}$ to all the backups, where m should be the n-th executed request.

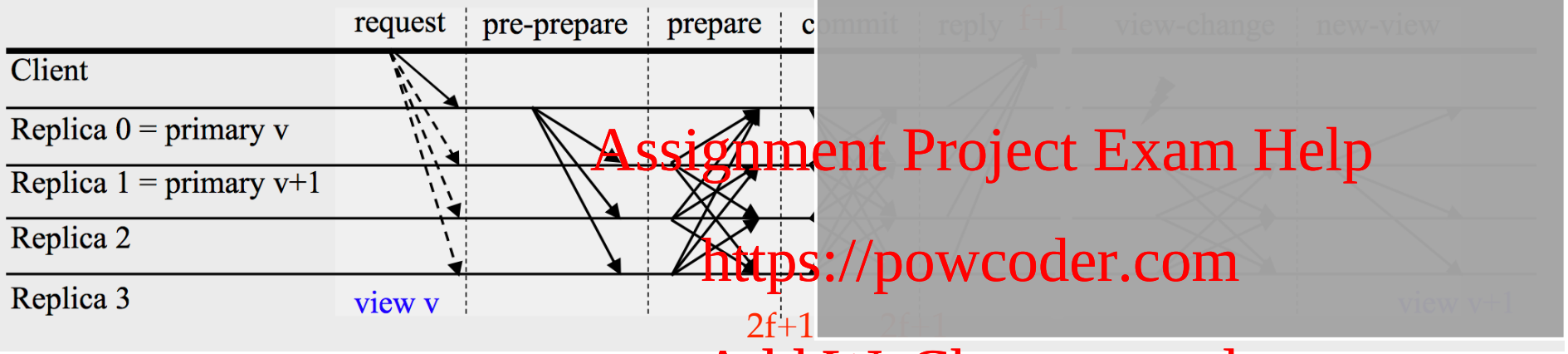
Client message

Sequence number

The strictly monotonically increasing and contiguous sequence number n ensures preservation of this order despite message reordering.

Miguel Castro, Barbara Liskov: Practical byzantine fault tolerance. OSDI 1999, pp. 173-186.
Miguel Castro, Barbara Liskov: Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. 20(4): 398-461 (2002)

PBFT: n=3f+1

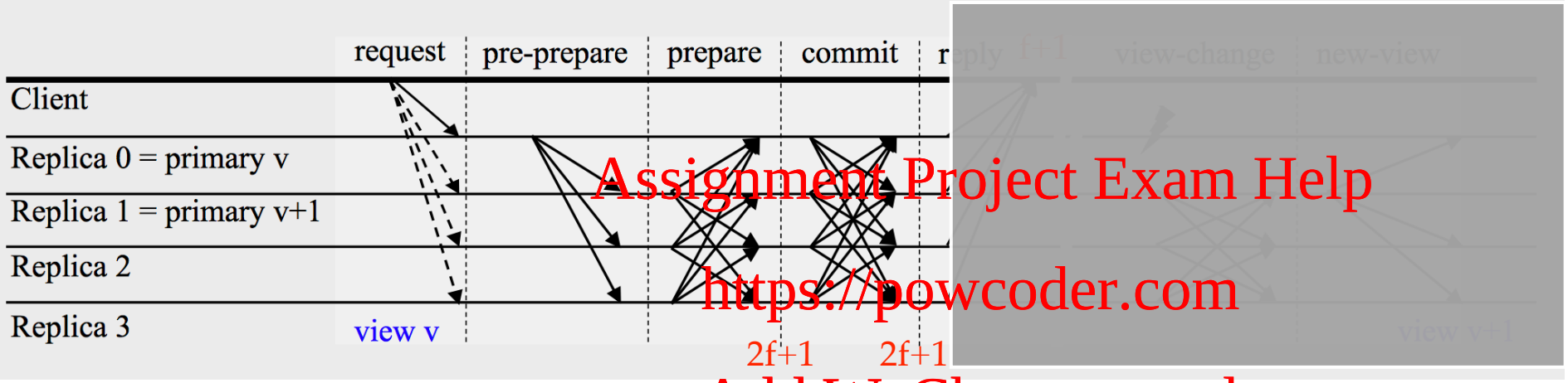


Prepare:

Node i acknowledges the receipt of a pre-prepare message by sending the digest d of the client's request in $< PREPARE, v, n, d, i >_{\sigma_i}$ to all replicas.

Miguel Castro, Barbara Liskov: Practical byzantine fault tolerance. OSDI 1999, pp. 173-186.
Miguel Castro, Barbara Liskov: Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. 20(4): 398-461 (2002)

PBFT: n=3f+1



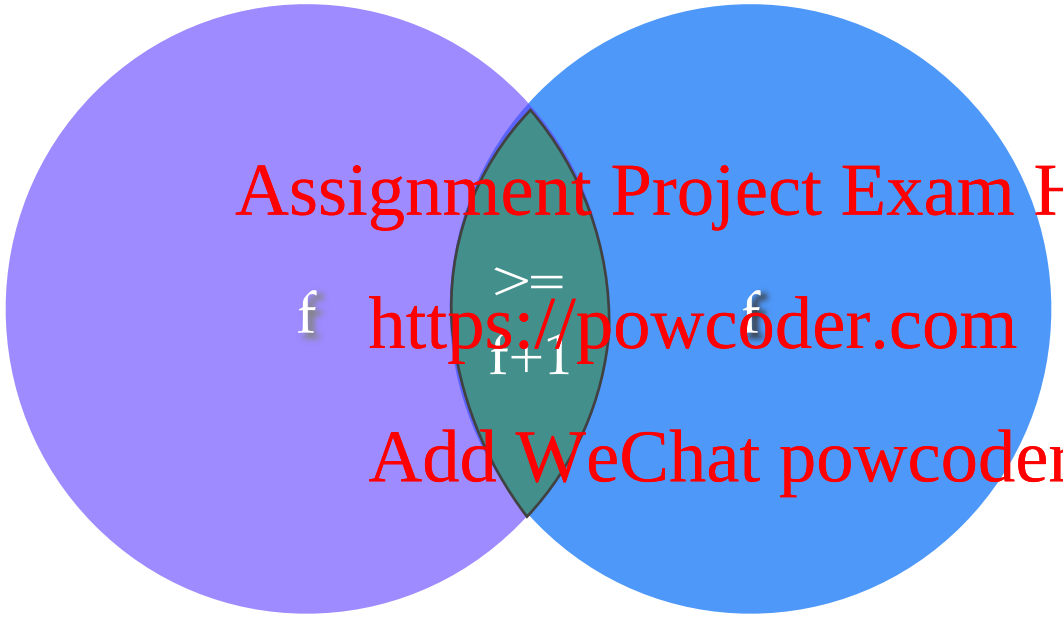
Commit:

Node i acknowledges the reception of $2f$ prepares matching a valid pre-prepare by broadcasting $< COMMIT, v, n, d, i >_{\sigma_i}$. In this case, we say that the message is prepared at i.

Miguel Castro, Barbara Liskov: Practical byzantine fault tolerance. OSDI 1999, pp. 173-186.
Miguel Castro, Barbara Liskov: Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. 20(4): 398-461 (2002)

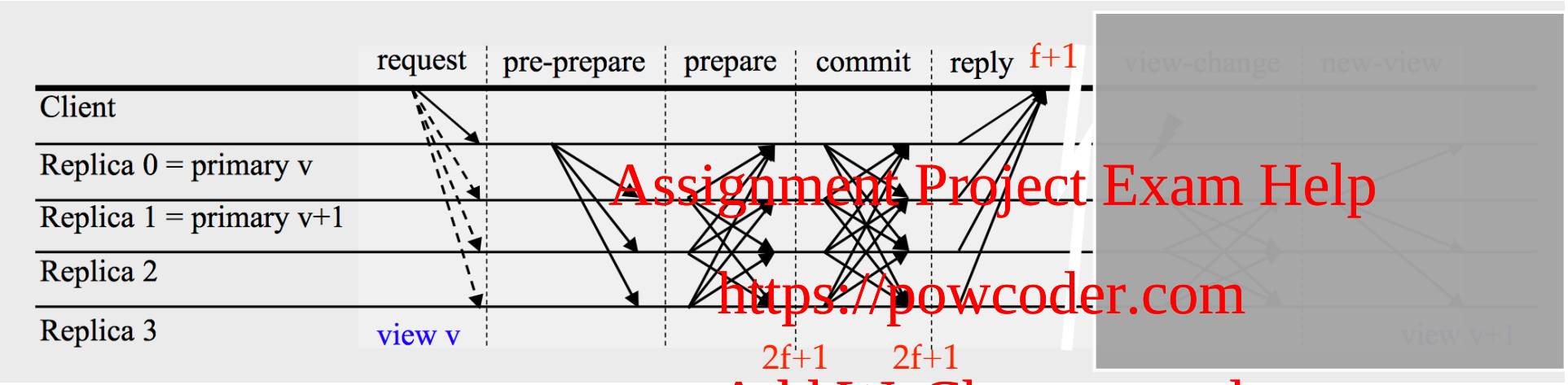
Quorum: at least n-f votes

Why 2f+1?



n>=3f+1

PBFT: n=3f+1



Execution & Reply

Nodes execute client operations after receiving $2f + 1$ matching commits, and follow the order of sequence numbers for this execution. Once node i has executed the operation o requested by client c, it sends $< REPLY, v, t, c, i, r >_{\sigma_i}$ to c, where r is the result of applying o to the service state. Client c accepts r if it receives $f + 1$ matching replies from different replicas.

Miguel Castro, Barbara Liskov: Practical byzantine fault tolerance. OSDI 1999, pp. 173-186.
Miguel Castro, Barbara Liskov: Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. 20(4): 398-461 (2002)

Why f+1?

Definition 1. (Consensus.) There are n nodes, of which at most f can be malicious, i.e., at least $n - f$ nodes are correct. For all $i \in [1, n]$, node i starts with an input value v_i. The nodes must decide for one of those values, satisfying the following properties:

- **Agreement:** All correct nodes decide for the same value.
- **Termination:** All correct nodes terminate in finite time.
- **Validity:** The decision value must be the input value of a node.

If f+1 nodes send back the replies of the same decision, then that means at least one honest node has agreed to this decision, thus all honest nodes have agreed to this decision.

View Change

The view-change protocol provides liveness by allowing the system to make progress when the primary fails.

If the client does not receive replies soon enough, it broadcasts the request to all replicas (If the request has already been processed, the replicas simply re-send the reply; replicas remember the last reply message they sent to each client. Otherwise, if the replica is not the primary, it relays the request to the primary).

View changes are triggered by timeouts that prevent backups from waiting indefinitely for requests to execute (a backup starts a timer when it receives a request).

If the timer of backup expires in view *v*, the backup starts a view change to move the system to view *v*+1. It stops accepting messages (other than checkpoint, view-change, and new-view messages) and multicasts a VIEW-CHANGE message to all replicas.

When the primary of view *v*+1 receives 2*f* valid VIEW-CHANGE messages for view *v*+1 from other replicas, it multicasts a NEW-VIEW message to all other replicas.

Synchrony in PBFT

The PBFT algorithm does not rely on synchrony to provide safety.

Therefore, it must rely on synchrony to provide liveness.

The synchrony assumption of PBFT is rather weak (high-level idea: message delays **do not** grow faster than the timeout period indefinitely).

Why are the 3 phases important?

The pre-prepare and prepare phases are used to totally order requests sent in the same view even when the primary, which proposes the ordering of requests, is faulty (i.e., non-faulty replicas agree on a total order for the requests within a view).

The prepare and commit phases are used to ensure that requests that commit are totally ordered across views.

If a message is prepared at a non-faulty node and that node received $2f+1$ commits for that message (including its own), then it is true that the message is prepared in $f+1$ non-faulty nodes (even if there are view changes). This ensures that any request that commits locally at a non-faulty replica will commit at $f+1$ or more non-faulty replicas eventually.

A brief summary

	PBFT	PoW of BTC
Openess	A pre-fixed committee for voting	Open to everyone
Non-malicious participants	Honest	Honest or rational
Assumption	$f \leq \left\lfloor \frac{n-1}{3} \right\rfloor$	f < 50% mining power
# voters	Small	Large
# players	n total; f faulty	?
Agreement	Deterministic	Eventual consistency

Reading

Practical Byzantine Fault Tolerance

Next lecture: Blockchain Network

Enjoy the break!