Computer Systems

Exercise Session

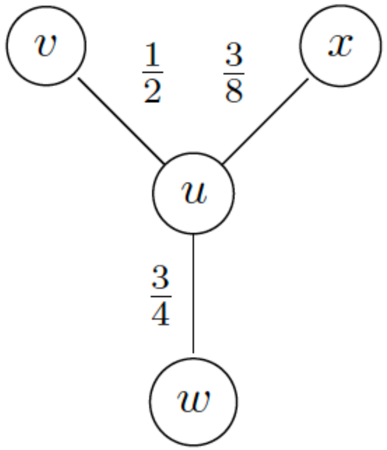
Last Exercise

Assignment 11

Assignment 1.2a

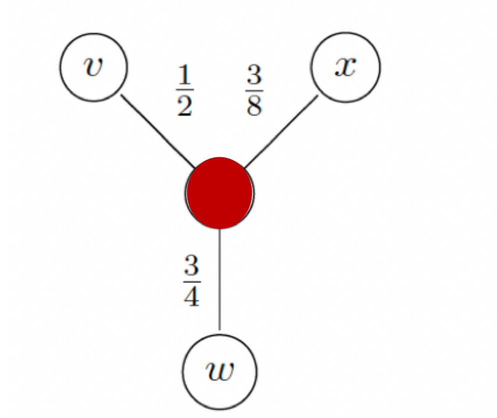
For each of the following caching networks, compute the social optimum, the pure Nash equilibria, the price of anarchy (PoA) as well as the optimistic price of anarchy$(OPoA)$:

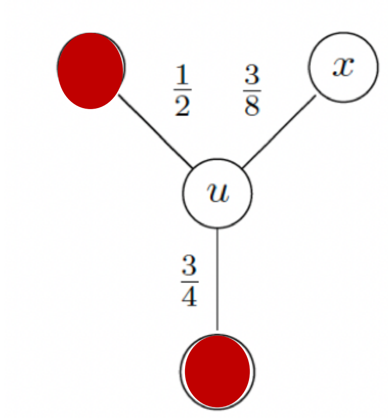
i. $d_u = d_v = d_w = d_x = 1$

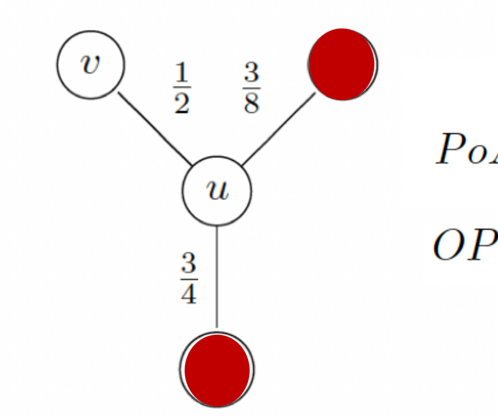


Solutions:

Nash Equilibria





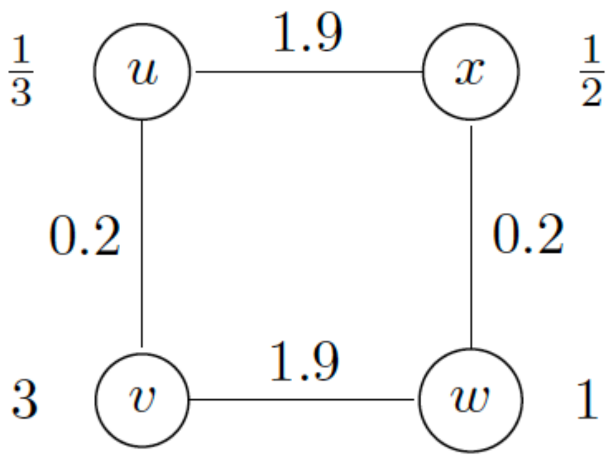


$$PoA = \frac{cost(0110)}{OPT} = \frac{\frac{1}{2}+1+1+\frac{7}{8}}{\frac{21}{8}} = \frac{9}{7} \approx 1.286$$

$$OPoA = 1.$$

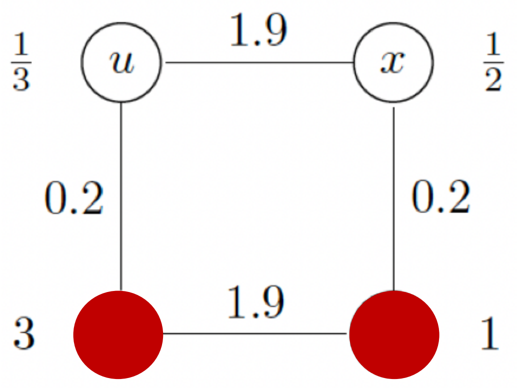
Assignment 1.2b

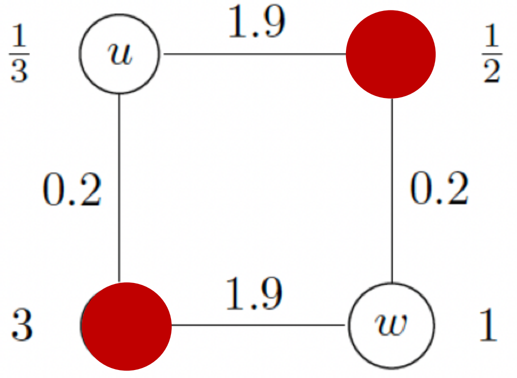
ii. The demand is written next to a node.



Solutions:

Nash Equilibria





$$PoA = \frac{cost(0101)}{OPT} = \frac{1/3 \cdot 0.2 + 1 + 0.2 + 1}{2.1\overline{6}} = \frac{68}{65} \approx 1.046$$

$$OPoA = 1.$$

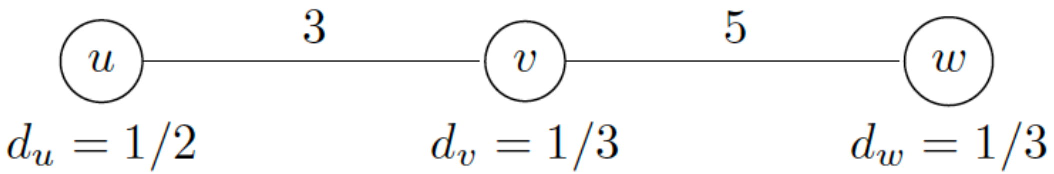
Assignment 1.3

The selfish caching model introduced in the lecture assumed that every peer incurs the same caching cost. However, this is a simplification of the reality. A peer with little storage space could experience a much higher caching cost than a peer who has terabytes of free disc space available. In this exercise, we omit the simplifying assumption and allow variable caching costs α_i for node i.

What are the Nash Equilibria in the following caching networks given that

i. $\alpha_u = 1$, $\alpha_v = 2$, $\alpha_w = 2$,

ii. $\alpha_u = 3$, $\alpha_v = 3/2$, $\alpha_w = 3$?



Does any of the above instances have a dominant strategy profile? What is the PoA of each instance?

Assignment 1.3

i. We have NE = {(101)} and PoA = 1 since social optimum SO={(101)}

ii. Nash equilibriums NE={(100), (010)} PoA = $\frac{40}{28} = 1.43$

Dominant strategy profile: None

Assignment 1.5 – PCA classes

The PoA of a class $\mathcal{C}$ is defined as the maximum PoA over all instances in $\mathcal{C}$. Let

- $\mathcal{A}^n_{[a,b]}$ be the class of caching networks with n peers, $a \leq \alpha_i \leq b$, $d_i = 1$, and each edge has weight 1,
- $\mathcal{W}^n_{[a,b]}$ be the class of networks with n peers, $a \leq d_i \leq b$, $\alpha_i = 1$, and each edge has weight 1.

Show that $PoA(\mathcal{A}^n_{[a,b]}) \leq \frac{b}{a} \cdot PoA(\mathcal{W}^n_{[\frac{1}{b},\frac{1}{a}]})$ for all $n > 0$.

Assignment 1.5 – PCA classes

Let I^n be an instance of $\mathcal{A}^n_{[a,b]}$ that maximizes the PoA

$x, y \in X$ two strategies in I_n s.t. PoA$(I^n) = \frac{cost(y)}{cost(x)}$

We construct $\hat{I}^n \in \mathcal{W}^n_{[\frac{1}{b},\frac{1}{a}]}$ by setting $d_i = \frac{1}{\alpha_i}$ for α_i from I^n

We have the same NE in I^n and $\hat{I}^n$. This is because the cover sets D_i (nodes for which we do not cache if these cache already) stay the same.

For I^n a peer j is in D_i iff $c_{i \leftarrow j} < \alpha_i$

For $\hat{I}^n$ a peer j is in D_i iff $c_{i \leftarrow j}/\alpha_i < 1$.

Assignment 1.5 – PCA classes

$$
\begin{aligned}
PoA(\hat{I}^n) &\geq \frac{c\hat{o}st(y)}{c\hat{o}st(x)} = \frac{\sum_{i=1}^n \left(y_i + (1-y_i)\frac{c_i(y)}{\alpha_i}\right)}{\sum_{i=1}^n \left(x_i + (1-x_i)\frac{c_i(x)}{\alpha_i}\right)} && (1)\\
&= \frac{b \cdot a \sum_{i=1}^n \left(y_i + (1-y_i)\frac{c_i(y)}{\alpha_i}\right)}{b \cdot a \sum_{i=1}^n \left(x_i + (1-x_i)\frac{c_i(x)}{\alpha_i}\right)} && (2)\\
&\geq \frac{a \sum_{i=1}^n (y_i\alpha_i + (1-y_i)c_i(y))}{b \sum_{i=1}^n (x_i\alpha_i + (1-x_i)c_i(x))} && (3)\\
&= \frac{a \cdot cost(y)}{b \cdot cost(x)} = \frac{a}{b} PoA(I^n) && (4)
\end{aligned}
$$

Assignment 2.3

Consistent hashing relies on having k hashing functions $\{h_0, \ldots, h_{k-1}\}$ that map a node's unique name and the object ids to hashes. There are several constructions for these hash functions, the most common being iterative hashing and salted hashing. In iterative hashing we use a hash function h and apply it iteratively so that the hashes of an object id o is defined as

$$h_i(o) = \begin{cases} h(o) & \text{if} \quad i = 0 \\ h(h_{i-1}(o)) & \text{otherwise.} \end{cases}$$

With salted hashing the object id is concatenated with the hash function index i resulting in the following definition

$$h_i(o) = h(o|i).$$

Which hashing function derivation is better and why?

=> Iterative hashing is computationally more expensive

Assignment 2.4 – Multiple Skiplists

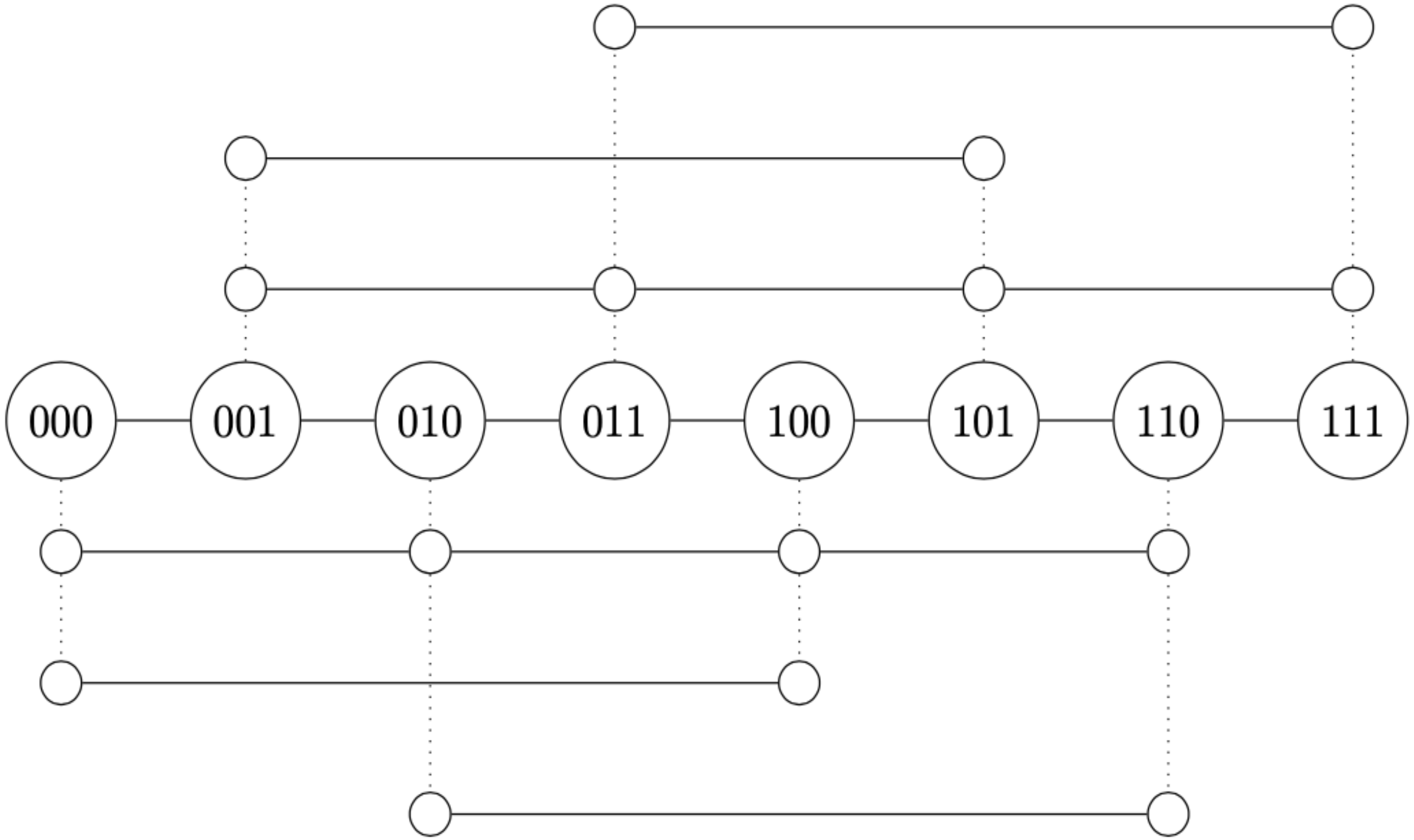
In the lecture we have seen the simple skip list in which a node is in the root level and promoted with probability 1/2. We now redefine the promotion so that a node is promoted to a list s if s is a suffix of the binary representation of the node's id. At each level l we now have multiple lists, each defined by a suffix s of length l. The root level is defined as the empty suffix with l. The first level has two lists $p \in \{0, 1\}$, the second level has four lists $p = \{00, 01, 10, 11\}$ and so on. We call the resulting network a multi-skiplist.

a) Assuming we have an 8 node network, with ids $\{000, \ldots, 111\}$, draw the multi-skiplist graph.

b) What is the minimum degree of a node in the multi-skiplist if we have d levels?

c) What is the maximum number of hops a lookup has to perform?

Assignment 2.4 – Multiple Skiplists



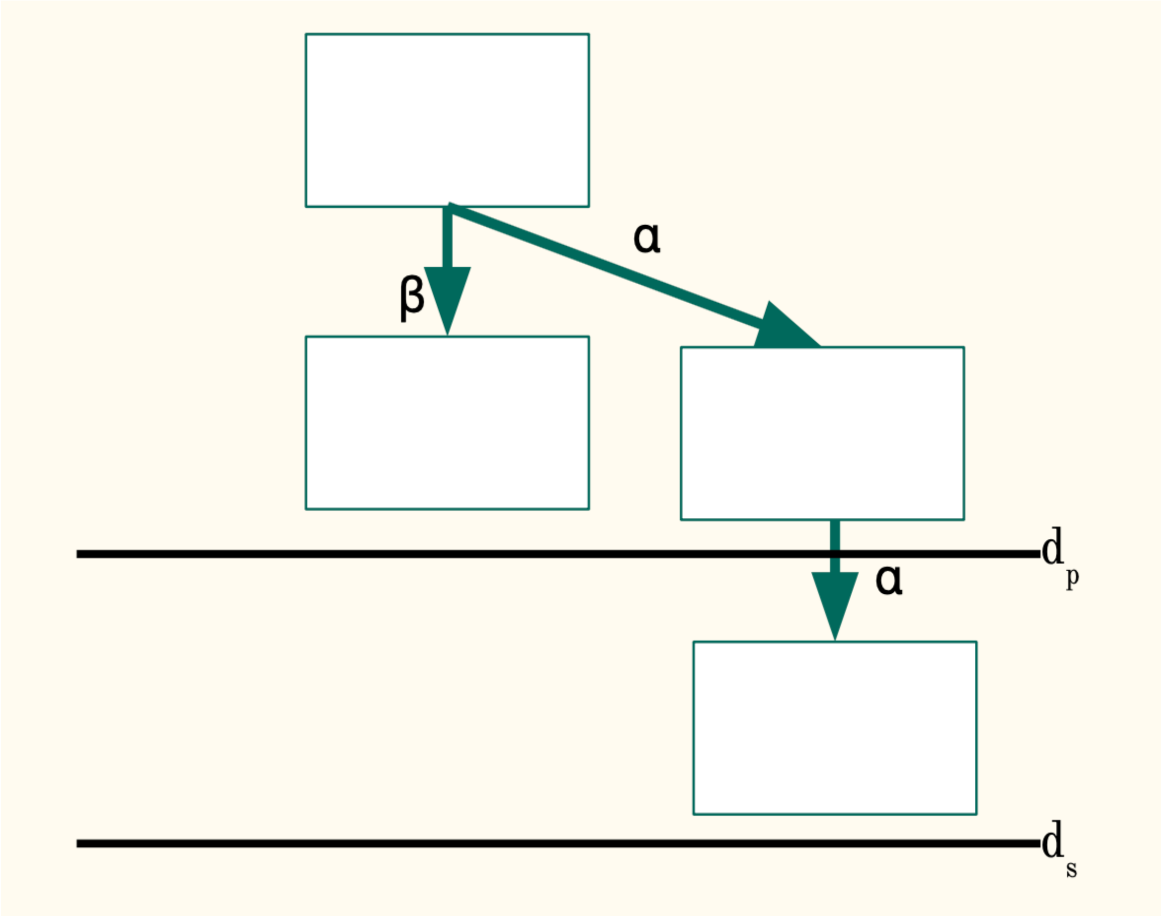
- Nodes have a constant degree of 2 * (d + 1)
- Maximum number of hops for lookup O(log(n))

Chapter 25

Advanced Blockchain

Selfish Mining

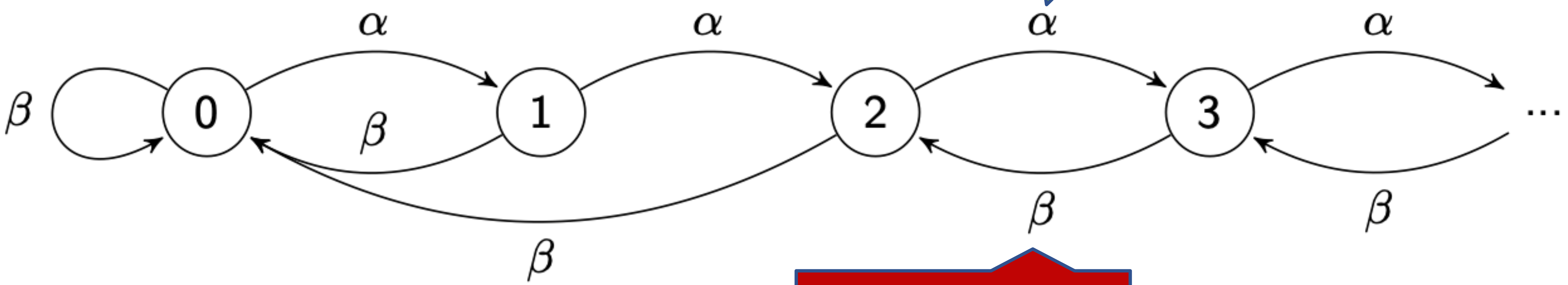
Should we always directly publish a mined block?



- Selfish miners don’t release their blocks immediately
- Instead keeps secret and continues to mine next block
- **Goal**: Hoping to get more reward (multiple block)
- **Problem**: sometimes useless work on grandchildren

Selfish Mining

Probability that selfish miner finds new block (share of computing power)



When selfish miner is two blocks in advance and a new public block is published, the selfish miner releases both its two blocks, so they do not become worthless

Probability that altruistic miner finds new block

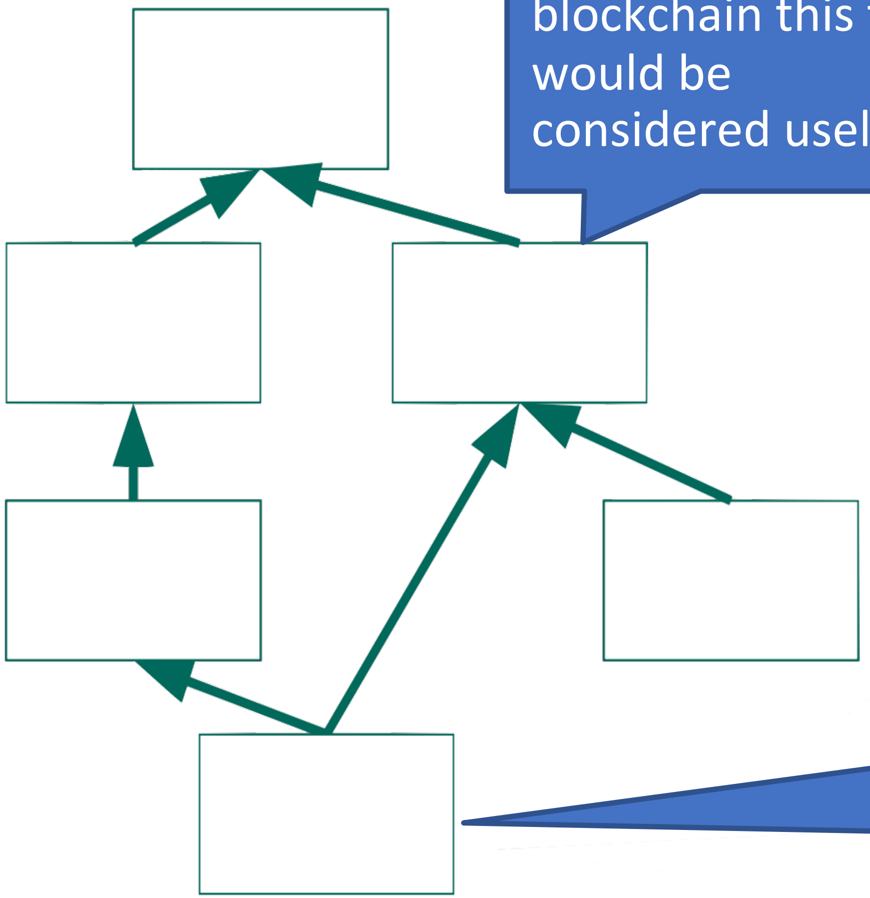
Selfish Mining

- α = ratio of computing power of a selfish miner
- γ = share of the altruistic mining power the selfish miner can reach when seeing a new block

<u>**Selfish Miner Reward:**</u>

$$\frac{\alpha(1-\alpha)^2(4\alpha+\gamma(1-2\alpha))-\alpha^3}{1-\alpha(1+(2-\alpha)\alpha)}$$

DAG-Blockchain

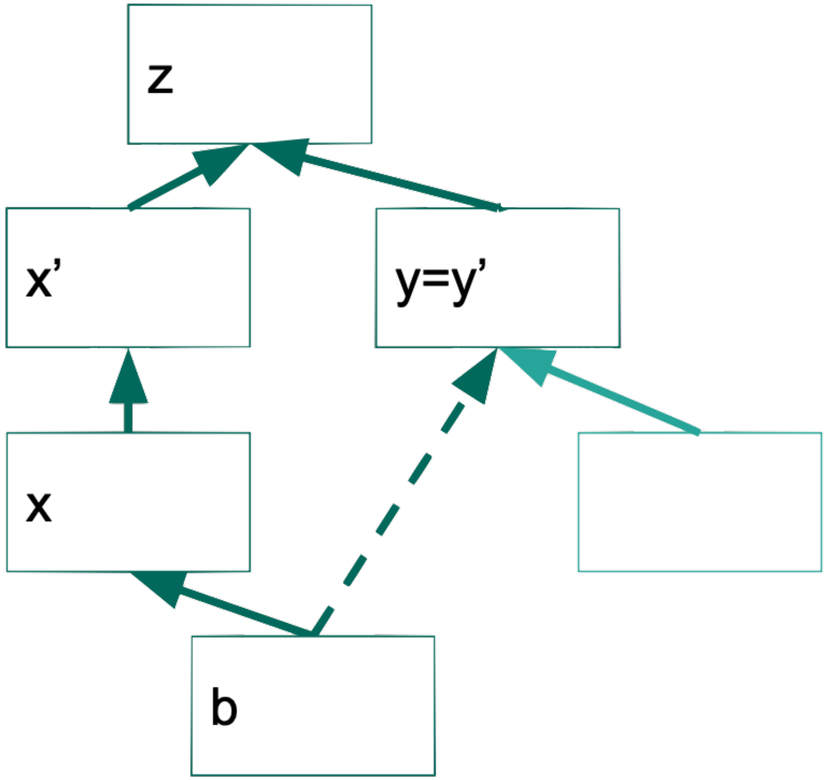


In a regular blockchain this fork would be considered useless

- Solve problem of wasted resources on fork
- Instead of one predecessor, a block can reference multiple parent blocks (hashes)
- Cycles are not possible, as that would require referencing a block whose hash is not yet known.

In a DAG-blockchain this block can reference a fork (we introduce an ordering to ensure conflicting transactions don’t violate our constraints)

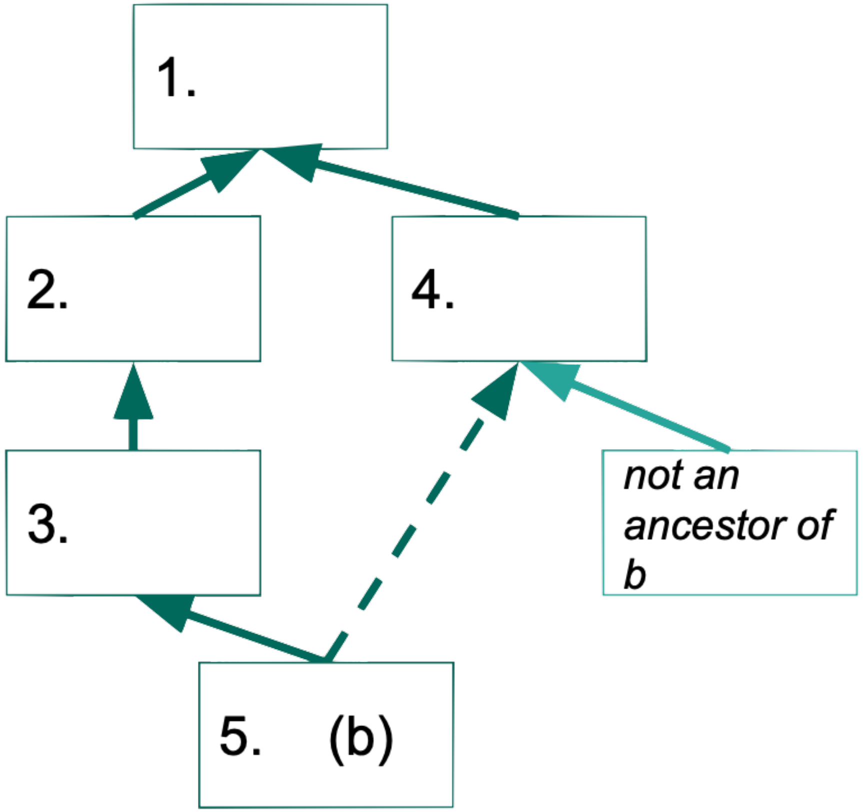
DAG-Blockchain



- Blocks now can have **multiple parents**. To aid ordering the blocks, a **tree** (solid lines), i.e. one designated parent per block, is marked.

- **Parent Order of b**, having DAG-parents *x, y*: x < y ⟺ “Starting from common ancestor *z* of *x* and *y*, the subtree starting just below *z* that contains *x*, restricted to ancestors of *b*, is larger than the analogously defined subtree for *y*.”

DAG-Blockchain



Algorithm 25.12 DAG-Blockchain Ordering

We totally order all dag-ancestors of block b as $<_b$ as follows:
Initialize $<_b$ as empty
for all dag-parents p of b, in their parent order **do**
  Compute $<_p$ (recursively)
  Remove from $<_p$ any blocks already included in $<_b$
  Append $<_p$ at the end of $<_b$
end for
Append block b at the end of $<_b$

Ethereum

- Allows to run arbitrary computer programs in the block chain

- Different types of transactions:
  1. Simple transaction (Send some ETH from Alice to Bob)
  2. Smart Contract creation transaction (employing smart contract into the Ethereum blockchain)
  3. Smart Contract execution transaction (executing specific functions of the smart contract)

Proof-of-Stake

Awarding block rewards proportionally to the economic stake in the system

1. Chain-based proof of stake
    1. Lottery where accounts are selected with probability according to their stake
    2. Winner of the lottery can extend the blockchain by one block and earn a reward

2. BFT based proof of stake
    1. Lottery where accounts are selected with probability according to their stake
    2. Winner suggests block but a committee votes whether to accept the block into the chain

Chapter 26

Advanced Agreement

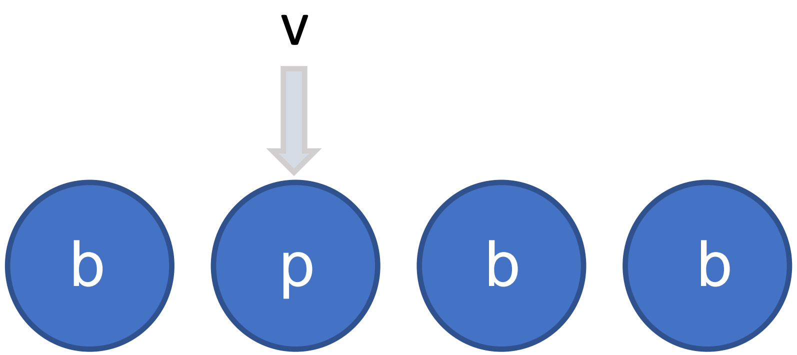
Prerequisite: Signatures

- Nodes can sign messages such that they **can not be forged**
- Even byzantine nodes can not forge signatures
- This enables us to “trust” forwarded messages, i.e. even if a Byzantine node forwards messages by a correct node, it cannot change the contents of the correct node’s message.
- For information on how this is implemented, look up **public-key cryptography**.

Notation: msg_n: Message signed by *n*.

System Model

Goal: state replications. Clients send request r, and servers execute them all in the same order



n = 3f + 1

- **Primary p:** Node that currently acts as serializer, ordering the commands
- **Backup b:** Node that currently is not a primary
- **View v:** A node in view v considers v mod n to be primary
- **Sequence number s: s**: Unique number assigned to commands by which they are ordered.

PBFT (Practical Byzantine Fault Tolerance)

Agreement Protocol

Ensures agreement.

Uses serializing primary to reach agreement on command order. Always ensures agreement, but only ensures progress if primary is correct.

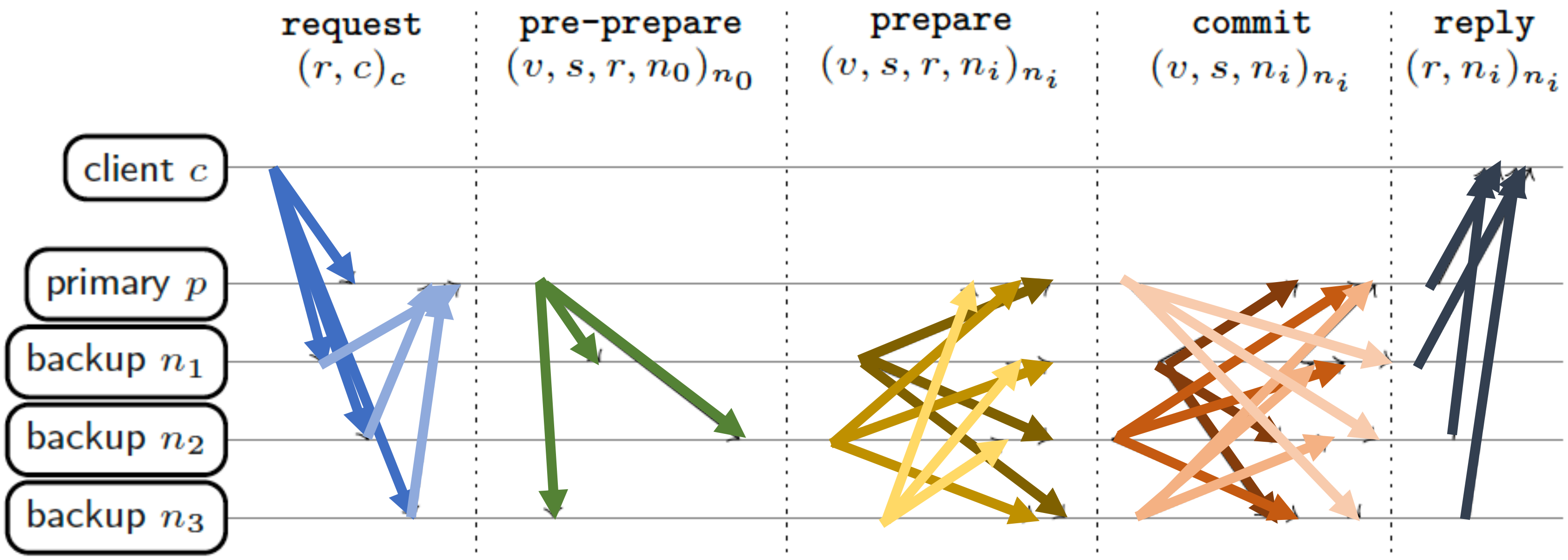
\+

View Change Protocol

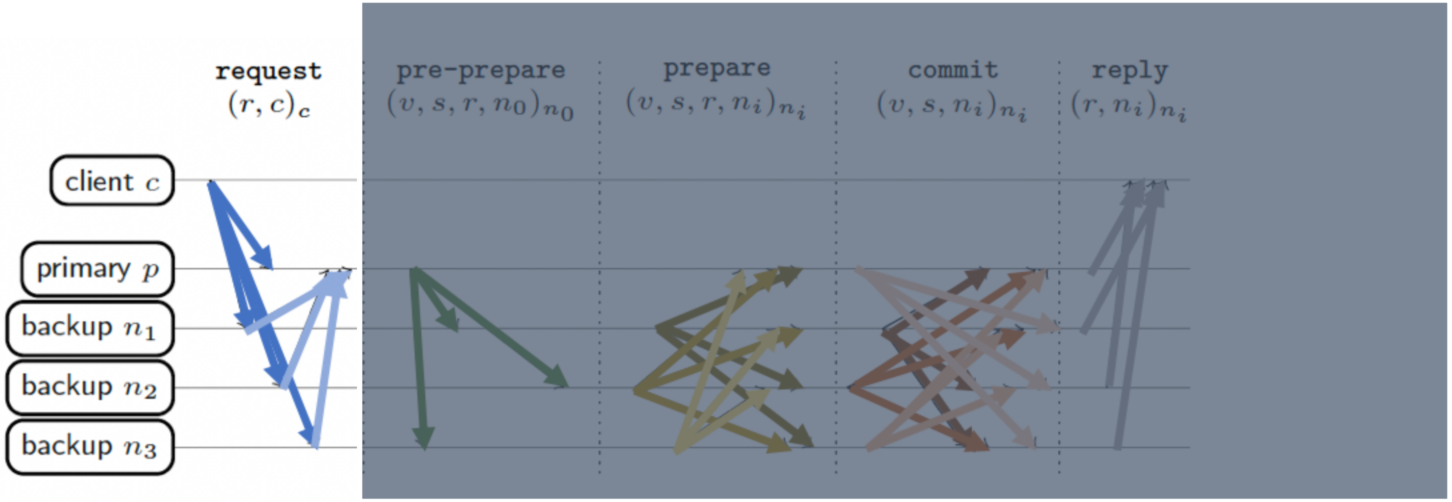
Ensures progress.

Completes a change of view, i.e. selecting a new primary, if the current one is suspected to be faulty, while ensuring the state remains synchronized.

PBFT – Agreement Protocol

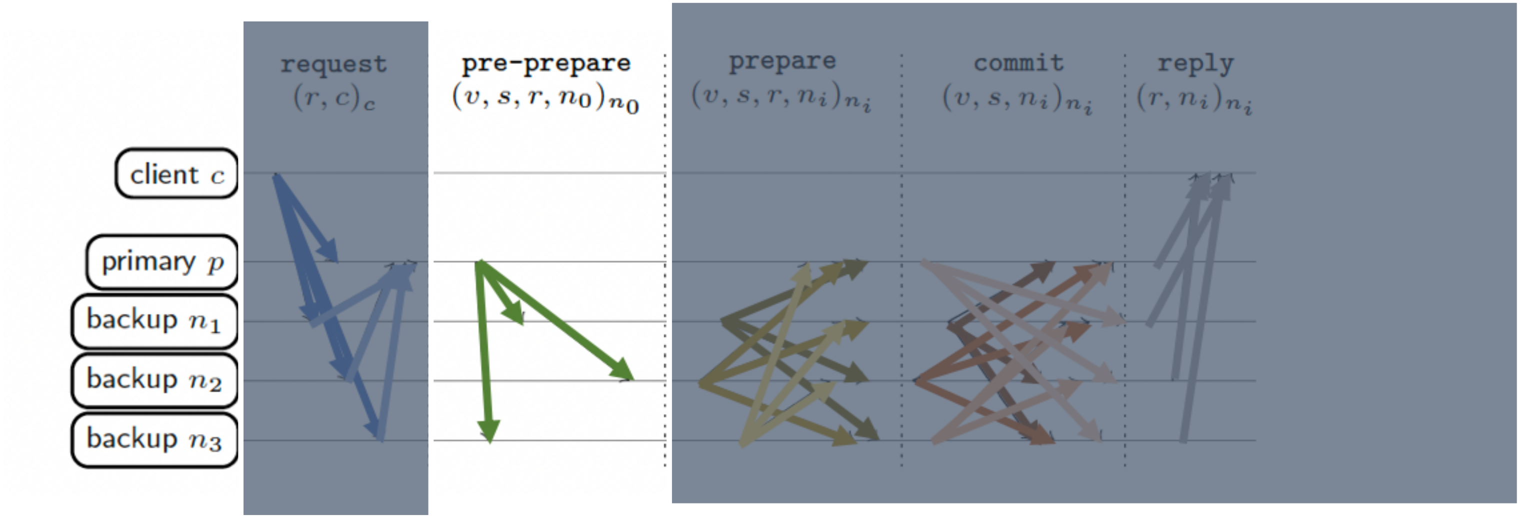


PBFT – request



- Why do backups confirm request?
  - otherwise client could only send request to backups and not primary. Then backups would trigger a view change without the primary being faulty

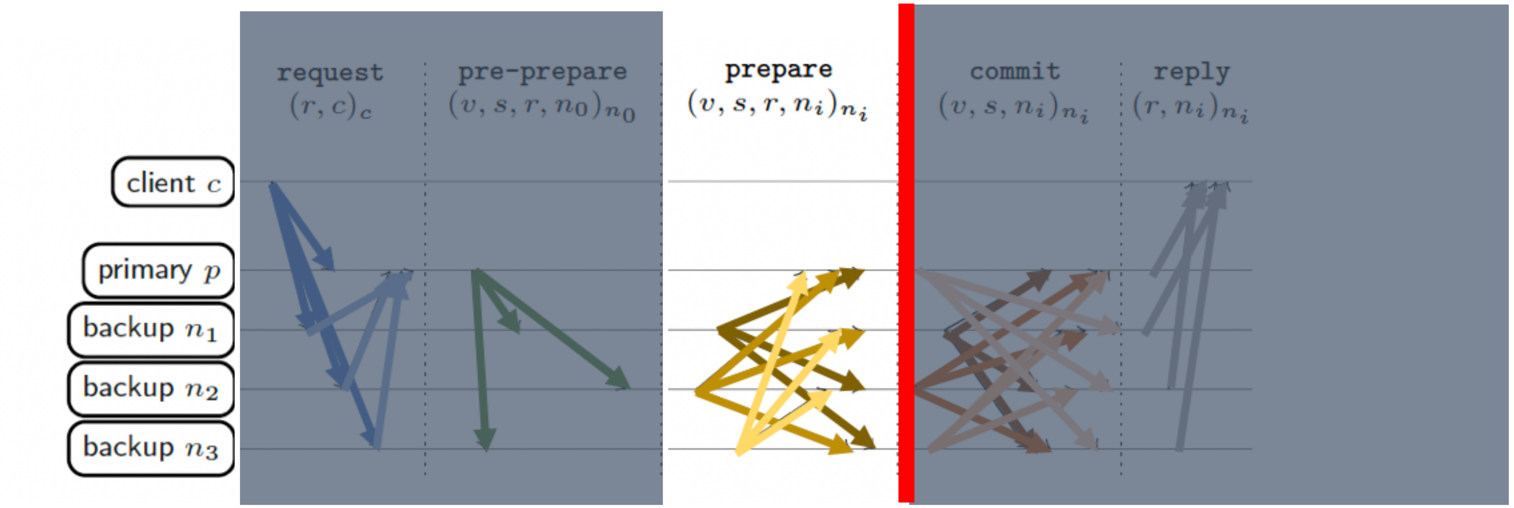
PBFT – pre-prepare



Why pre-prepare necessary?

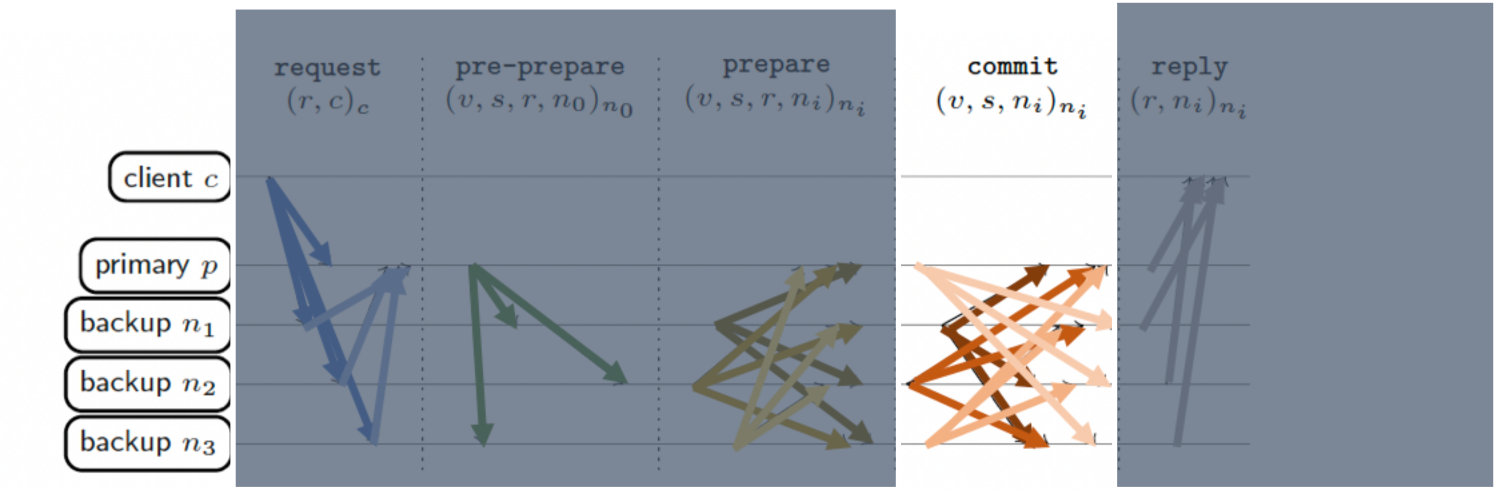
- all backups get sequence number and current view identifier.

PBFT – prepare



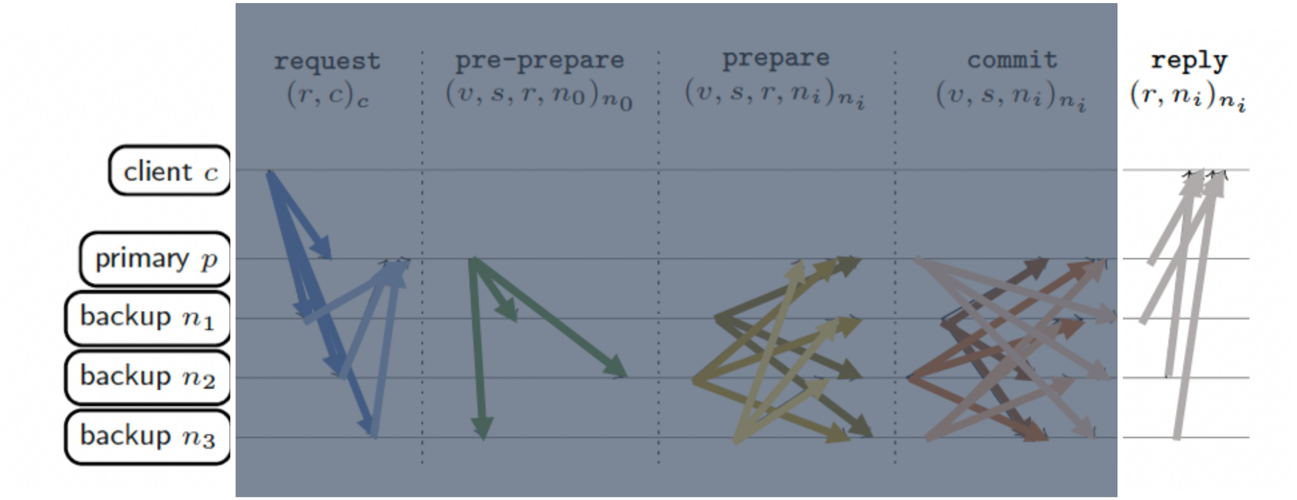
- Why prepare messages necessary?
  - after each node got enough prepare messages, it knows that the request has been propagated through the system and no other request with that sequence number can ever be executed
- Why exactly 2f prepare messages needed before commit?
  - because then, 2f+1 messages from different nodes have been received in total (pre-prepare message from primary and 2f prepare messages from backups). If two nodes have such a prepared certificate for a command, this means that in those two sets of 2f+1 messages/nodes, at least one correct message/node must overlap. Thus, the two commands must be the same.

PBFT – commit



- Why commit necessary?
  - we have to know that enough nodes have the prepared certificate and will also execute the command. So we check that by issuing commit messages.
- Why do we wait for 2f+1 commit messages?
  - Same as before, we want 2f+1 nodes so we have a correct node in every intersection

PBFT – reply



- Why reply necessary?
  - client needs to know that command got executed
- Why does client have to wait for f+1?
  - because then at least one correct node executed command

PBFT – view change

- what if the primary turns out to be byzantine?
  - Idea: if the faulty timers in nodes expire, nodes start a view change, to switch to the new primary
  - new primary needs to know which requests have been executed
    - gather from 2f+1 nodes the prepared certificates

- What does new primary do?

PBFT – view change backups

Algorithm 25.22 PBFT View Change Protocol: View Change Phase

Code for backup b in view v whose faulty-timer has expired:

1: stop accepting `pre-prepare`/`prepare`/`commit`-messages for v
2: let $\mathcal{P}_b$ be the set of all prepared-certificates that b has collected since the system was started
3: send $\texttt{view-change}(v+1, \mathcal{P}_b, b)_b$ to all nodes

if faulty timer expires, they send view-change message (v+1, set of all prepared certificates, own name) to all nodes and stop accepting messages for view v

PBFT – view change primary

Algorithm 25.23 PBFT View Change Protocol: New View Phase - Primary

Code for primary p of view $v+1$:

1: accept $2f+1$ `view-change`-messages (including possibly p's own) in a set $\mathcal{V}$ (this is the *new-view-certificate*)
2: let $\mathcal{O}$ be a set of $\texttt{pre-prepare}(v+1, s, r, p)_p$ for all pairs (s, r) where at least one prepared-certificate for (s, r) exists in $\mathcal{V}$
3: let $s_{max}^{\mathcal{V}}$ be the highest sequence number for which $\mathcal{O}$ contains a `pre-prepare`-message
4: add to $\mathcal{O}$ a message $\texttt{pre-prepare}(v+1, s', \texttt{null}, p)_p$ for every sequence number $s' < s_{max}^{\mathcal{V}}$ for which $\mathcal{O}$ does not contain a `pre-prepare`-message
5: send $\texttt{new-view}(v+1, \mathcal{V}, \mathcal{O}, p)_p$ to all nodes
6: start processing requests for view $v+1$ according to Algorithm 25.12 starting from sequence number $s_{max}^{\mathcal{V}}+1$

- accept 2f+1 view change messages
- add a null message for every sequence number that has not been used in the set of prepare messages, so that we have a complete set to continue with
- send new-view message that contains all the prepare message so that all nodes are on the same page
- start working as primary

PBFT – view change

- what if new primary is also byzantine?

Algorithm 25.24 PBFT View Change Protocol: New View Phase - Backup

Code for backup b of view $v+1$ if b's local view is $v' < v+1$:

accept `new-view`$(v+1, \mathcal{V}, \mathcal{O}, p)_p$
stop accepting `pre-prepare`-/`prepare`-/`commit`-messages for v`// in case b has not run Algorithm 25.22 for v+1 yet`
set local view to $v+1$
if p is primary of $v+1$ **then**
  if $\mathcal{O}$ was correctly constructed from $\mathcal{V}$ according to Algorithm 25.23 Lines 2 and 4 **then**
    respond to all `pre-prepare`-messages in $\mathcal{O}$ as in the agreement protocol, starting from Algorithm 25.15
    start accepting messages for view $v+1$
  else
    trigger view change to $v+2$ using Algorithm 25.22
  end if
end if

Backups check that O is constructed correctly from V and also time how long the new primary takes for the view change. If anything goes wrong, another view change is triggered.