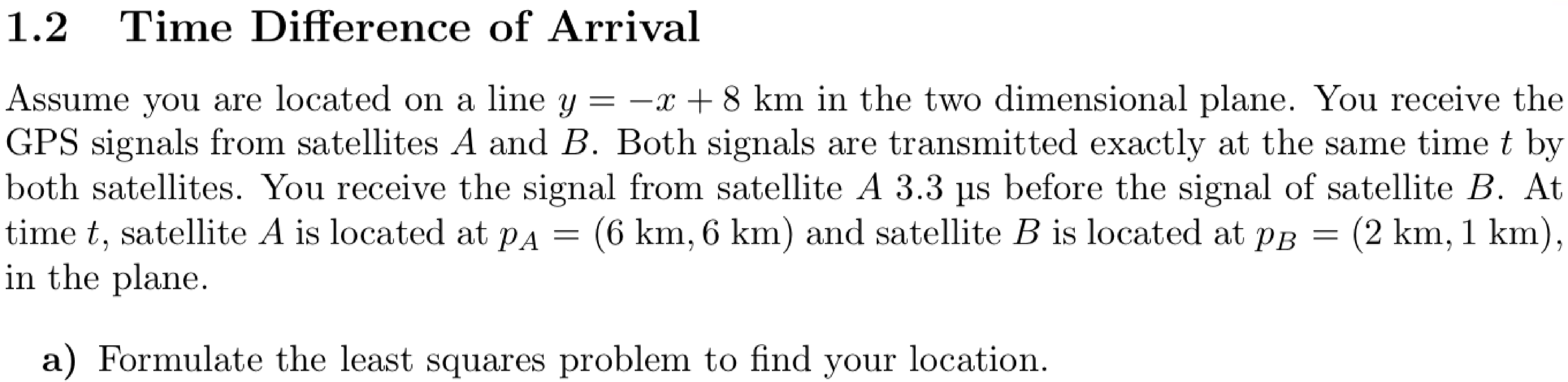
Computer Systems

Exercise Session

Last Exercise

Assignment 9

Last Exercise



1.2 Time Difference of Arrival

Assume you are located on a line $y = -x + 8$ km in the two dimensional plane. You receive the GPS signals from satellites A and B. Both signals are transmitted exactly at the same time t by both satellites. You receive the signal from satellite A 3.3 µs before the signal of satellite B. At time t, satellite A is located at $p_A = (6\text{ km}, 6\text{ km})$ and satellite B is located at $p_B = (2\text{ km}, 1\text{ km})$, in the plane.

a) Formulate the least squares problem to find your location.

Location p

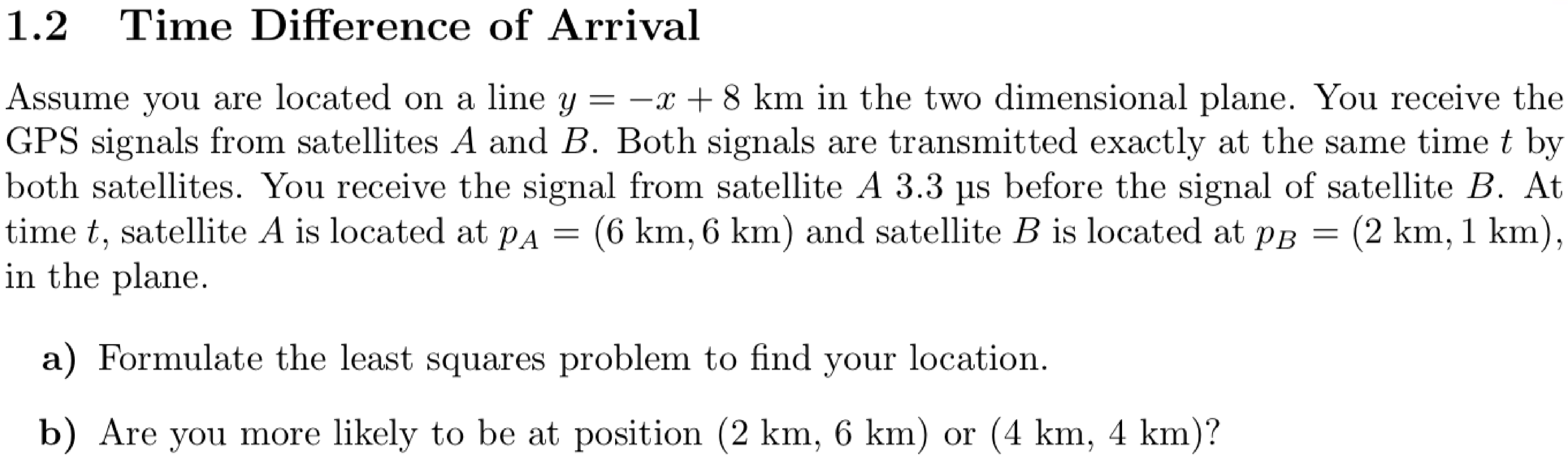
Distance to A: $|| p_A – p ||$

Distance to B: $|| p_B – p ||$

Time difference 3.3 µs => distance = 3.3 µs · 3 · 10^8 m/s ≈ 1km

Residual r = $|| p_B − p || − || p_A − p || − 1$ km

Last Exercise



1.2 Time Difference of Arrival

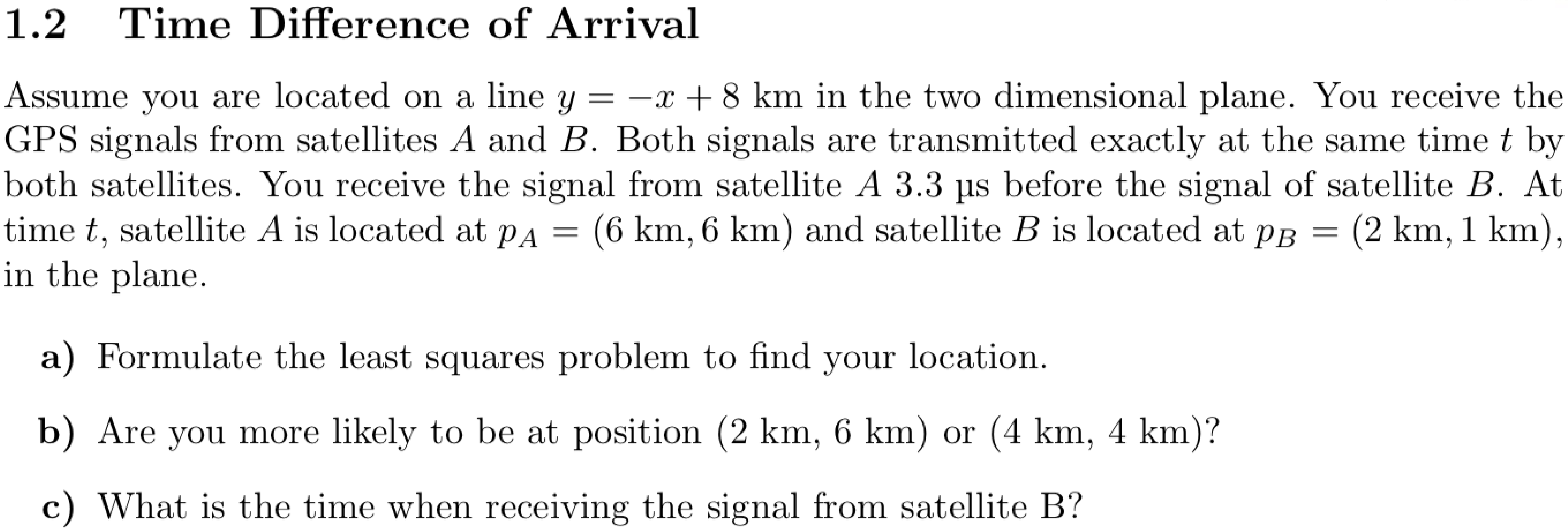
Assume you are located on a line $y = -x + 8$ km in the two dimensional plane. You receive the GPS signals from satellites A and B. Both signals are transmitted exactly at the same time t by both satellites. You receive the signal from satellite A 3.3 µs before the signal of satellite B. At time t, satellite A is located at $p_A = (6\text{ km}, 6\text{ km})$ and satellite B is located at $p_B = (2\text{ km}, 1\text{ km})$, in the plane.

a) Formulate the least squares problem to find your location.

b) Are you more likely to be at position (2 km, 6 km) or (4 km, 4 km)?

Residual at (2,6): abs(||(2,1)-(2,6)||-||(6,6)-(2,6)||-1) = abs(5 – 4 – 1) = 0
Residual at (4,4): abs(||(2,1)-(4,4)||-||(6,6)-(4,4)||-1) = abs(3.6 – 2.8 – 1) = 0.2

Last Exercise



1.2 Time Difference of Arrival

Assume you are located on a line $y = -x + 8$ km in the two dimensional plane. You receive the GPS signals from satellites A and B. Both signals are transmitted exactly at the same time t by both satellites. You receive the signal from satellite A 3.3 µs before the signal of satellite B. At time t, satellite A is located at $p_A = (6\text{ km}, 6\text{ km})$ and satellite B is located at $p_B = (2\text{ km}, 1\text{ km})$, in the plane.

a) Formulate the least squares problem to find your location.

b) Are you more likely to be at position (2 km, 6 km) or (4 km, 4 km)?

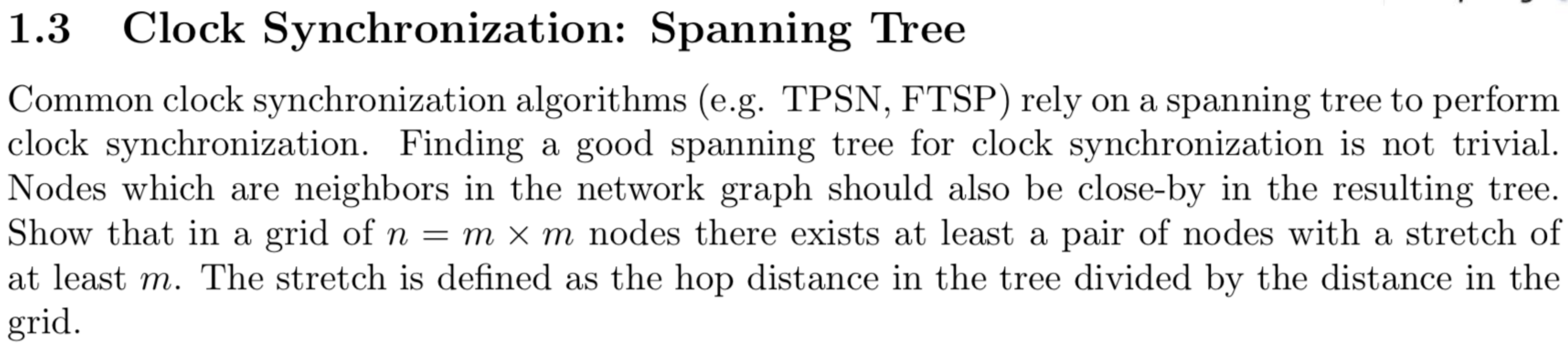
c) What is the time when receiving the signal from satellite B?

Distance from (2,6) to (2,1) is: 5 km

Divide by speed of light: $\frac{5km}{3*10^8 m/s} = 16.7\ microseconds$

Time message is received: t + 16.7 microseconds

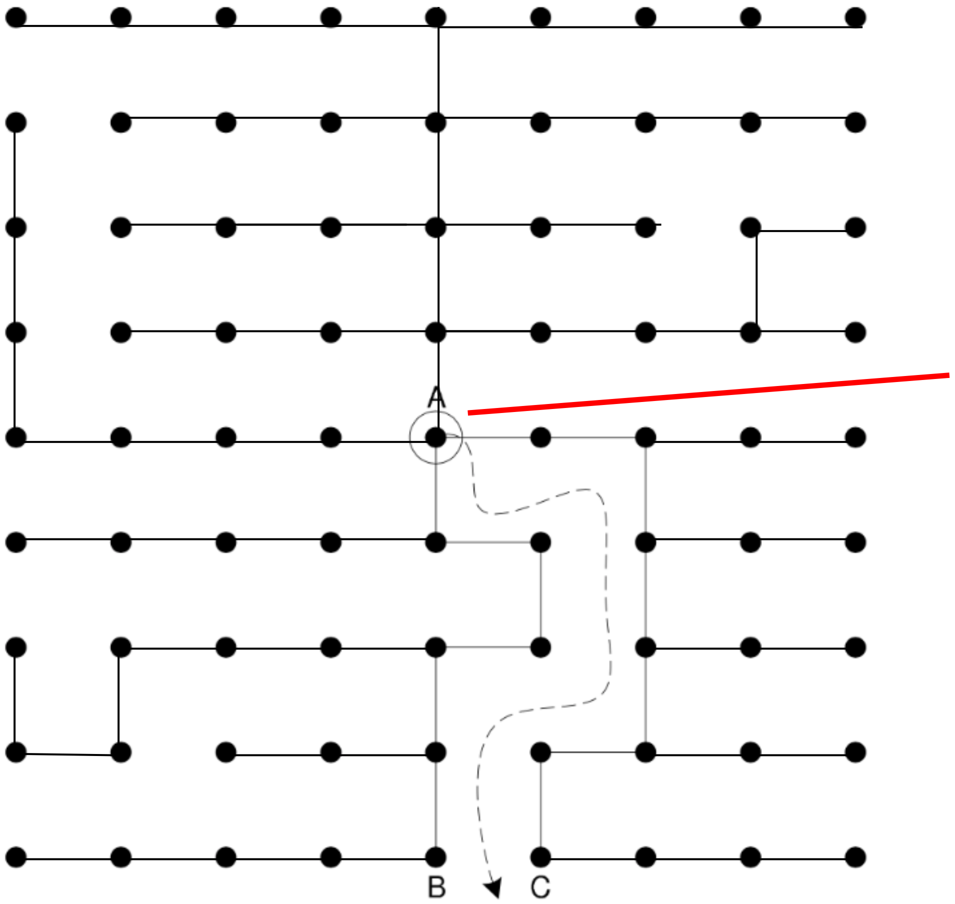
Last Exercise



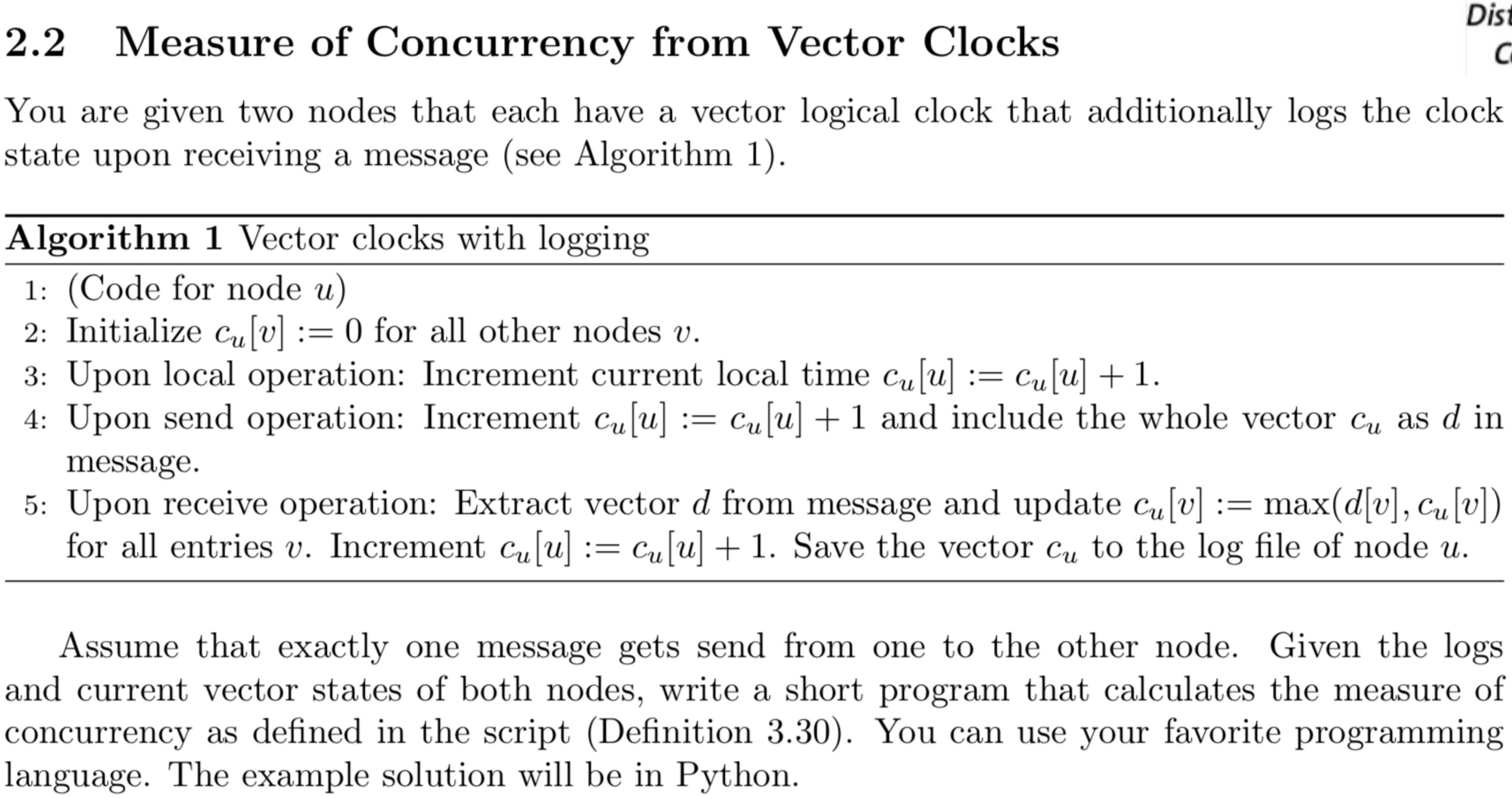
1.3 Clock Synchronization: Spanning Tree

Common clock synchronization algorithms (e.g. TPSN, FTSP) rely on a spanning tree to perform clock synchronization. Finding a good spanning tree for clock synchronization is not trivial. Nodes which are neighbors in the network graph should also be close-by in the resulting tree. Show that in a grid of $n = m \times m$ nodes there exists at least a pair of nodes with a stretch of at least m. The stretch is defined as the hop distance in the tree divided by the distance in the grid.

Last Exercise 1.3



Last Exercise



2.2 Measure of Concurrency from Vector Clocks

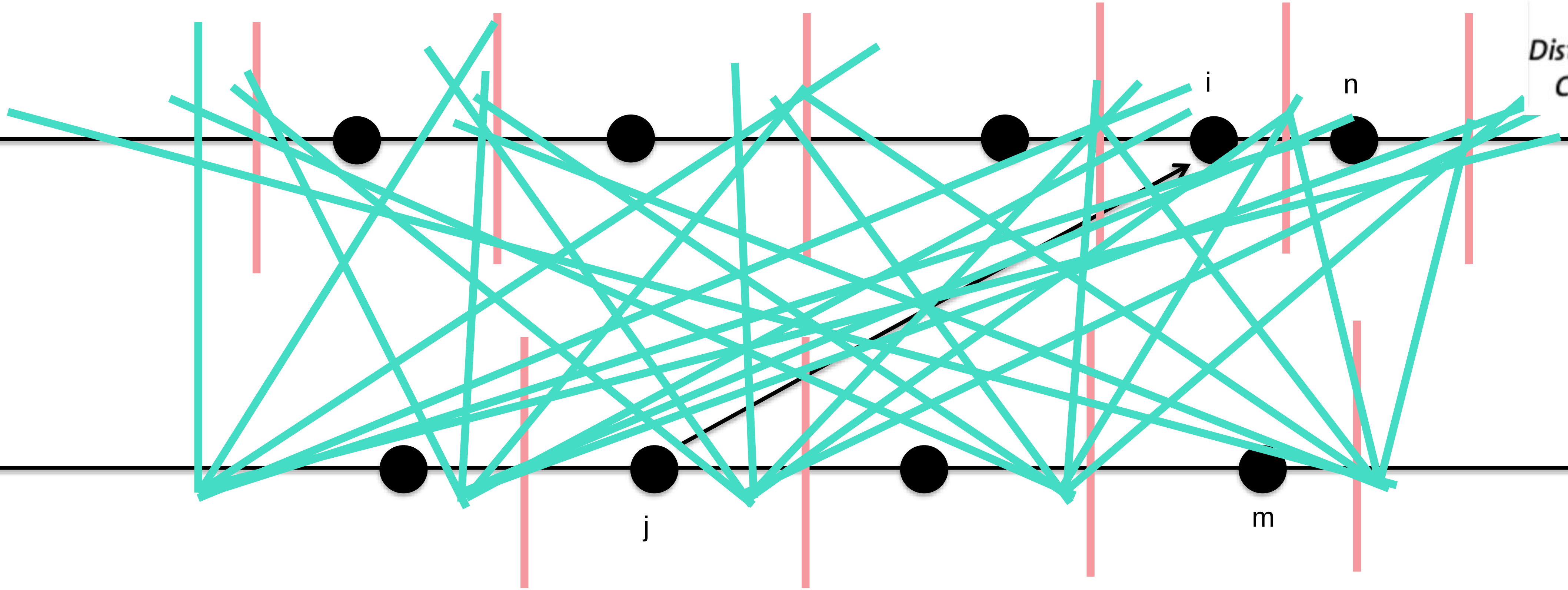
You are given two nodes that each have a vector logical clock that additionally logs the clock state upon receiving a message (see Algorithm 1).

Algorithm 1 Vector clocks with logging

```
(Code for node $u$)
Initialize $c_u[v] := 0$ for all other nodes $v$.
Upon local operation: Increment current local time $c_u[u] := c_u[u] + 1$.
Upon send operation: Increment $c_u[u] := c_u[u] + 1$ and include the whole vector $c_u$ as $d$ in
    message.
Upon receive operation: Extract vector $d$ from message and update $c_u[v] := \max(d[v], c_u[v])$
    for all entries $v$. Increment $c_u[u] := c_u[u] + 1$. Save the vector $c_u$ to the log file of node $u$.
```

Assume that exactly one message gets send from one to the other node. Given the logs and current vector states of both nodes, write a short program that calculates the measure of concurrency as defined in the script (Definition 3.30). You can use your favorite programming language. The example solution will be in Python.

Last Exercise 2.2



Measure of concurrency:

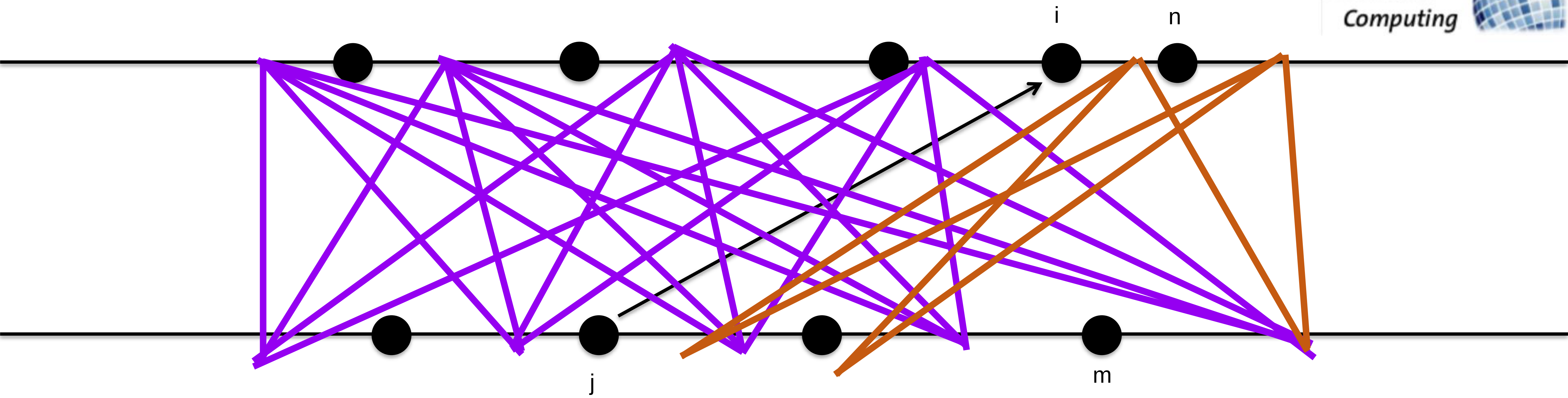
$$\frac{M_u - M_s}{M_c - M_s}$$

M_s = Nr. sequential snapshots $= n + m + 1$

M_c = Nr. concurrent snapshots $= (m+1)(n+1)$

M_u = Nr. snapshots in our system =
$\big(i * (m+1)\big) + \big((n+1-i)(m+1-j)\big)$

Last Exercise 2.2



M_s = Nr. sequential snapshots $= n + m + 1$

M_c = Nr. concurrent snapshots $=(n+1)(m+1)$

M_u = Nr. snapshots in our system $=$
$\left(i * (m+1)\right) + \left((n+1-i)(m+1-j)\right)$

Measure of concurrency: $\frac{M_u - M_s}{M_c - M_s}$

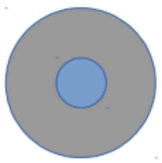
Chapter 21

Quorum Systems

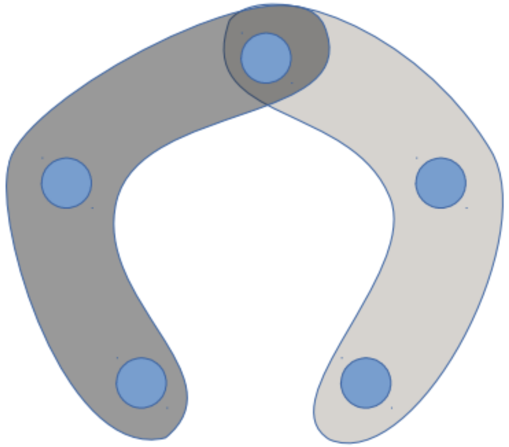
Quorum Systems

- Quorum
  - Subset of nodes
- Quorum System
  - Set of quorums such that every two quorums intersect
    - *Majority quorum*: every quorum has $\left\lfloor \frac{n}{2} \right\rfloor + 1$ nodes
- Idea:
  - When accessing a lock from all members of one quorum there will not be possible for another node to do the same for any quorum

Singleton and Majority



Singleton

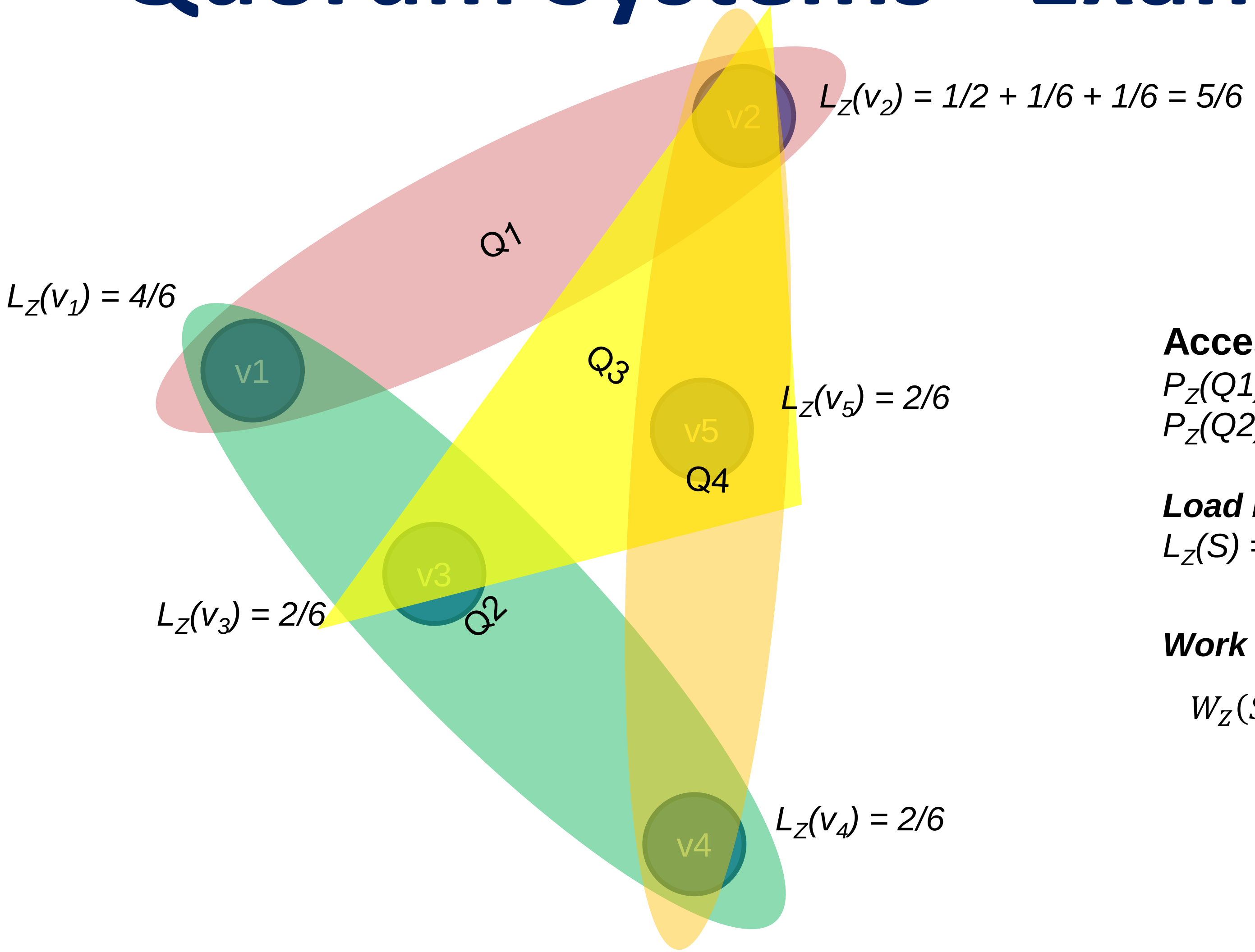


Majority quorum system
(every set of $\left\lfloor \frac{n}{2} \right\rfloor + 1$ nodes)

Load and work

- Load ~ probability
  - **Load of access strategy on node**: Probability it gets accessed
  - **Load on quorum system induced by access strategy**: load of node with maximal load
  - **Load of quorum system**: load induced by access strategy with best access strategy
- Work ~ count
  - **Work of quorum:** number of nodes
  - **Work induced by access strategy:** expected number of nodes accessed
  - **Work of quorum system:** work induced by access strategy with best access strategy

Quorum Systems - Example



$L_Z(v_2)$ = 1/2 + 1/6 + 1/6 = 5/6

$L_Z(v_1)$ = 4/6

$L_Z(v_5)$ = 2/6

$L_Z(v_3)$ = 2/6

$L_Z(v_4)$ = 2/6

Access Strategy *Z*:
$P_Z(Q1)$ = 1/2
$P_Z(Q2) = P_Z(Q3) = P_Z(Q4)$ = 1/6

Load *induced by Z on quorum system S:*
$L_Z(S) = \max_{v_i \in S} L_Z(v_i)$ = 5/6

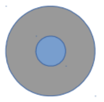
Work *induced by Z on quorum system S:*

$$W_Z(S) = \sum_{Q \in S} P_Z(Q) * W(Q) = \frac{1}{2} * 2 + \frac{1}{6} * 3 + \frac{1}{6} * 3 + \frac{1}{6} * 3 = \frac{15}{6}$$

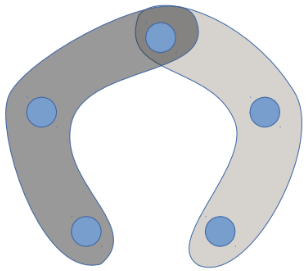
Fault Tolerance

- f-resilient
  - any f nodes can fail and at least one quorum still exists
  - resilience: largest such f

Load, work and resilience



Singleton



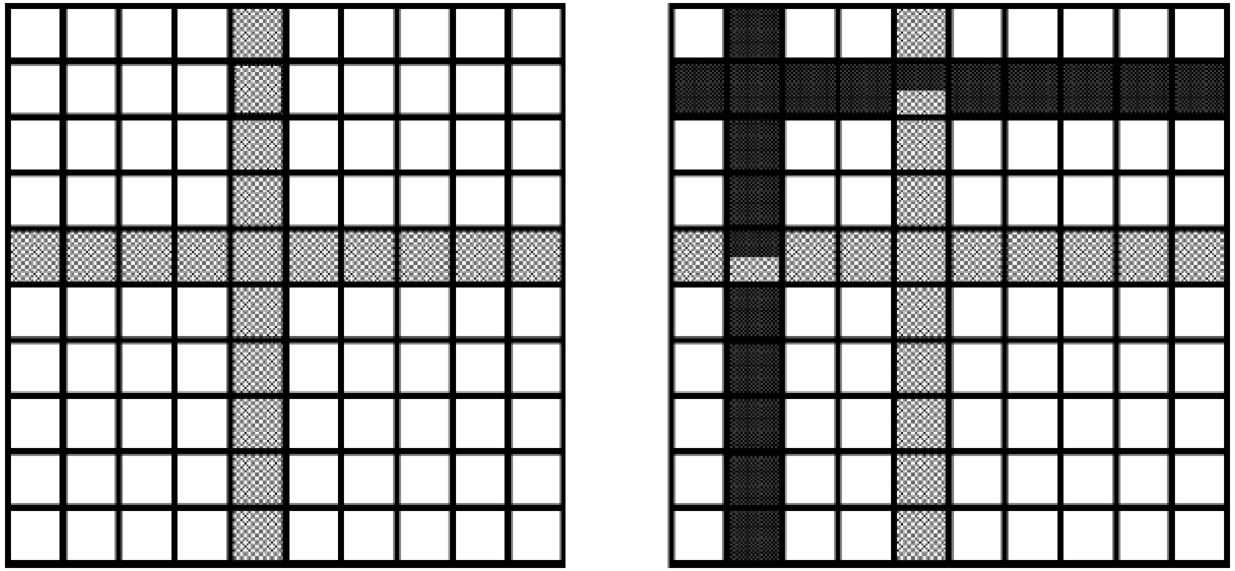
Majority quorum system
(every set of $\left\lfloor \frac{n}{2} \right\rfloor + 1$ nodes)

	Singleton	Majority
How many servers need to be contacted? **(Work)**	1	> n / 2
What's the load of the busiest server? **(Load)**	100%	≈ 50%
How many server failures can be tolerated? **(Resilience)**	0	< n / 2

Grid Quorum System – Basic Grid

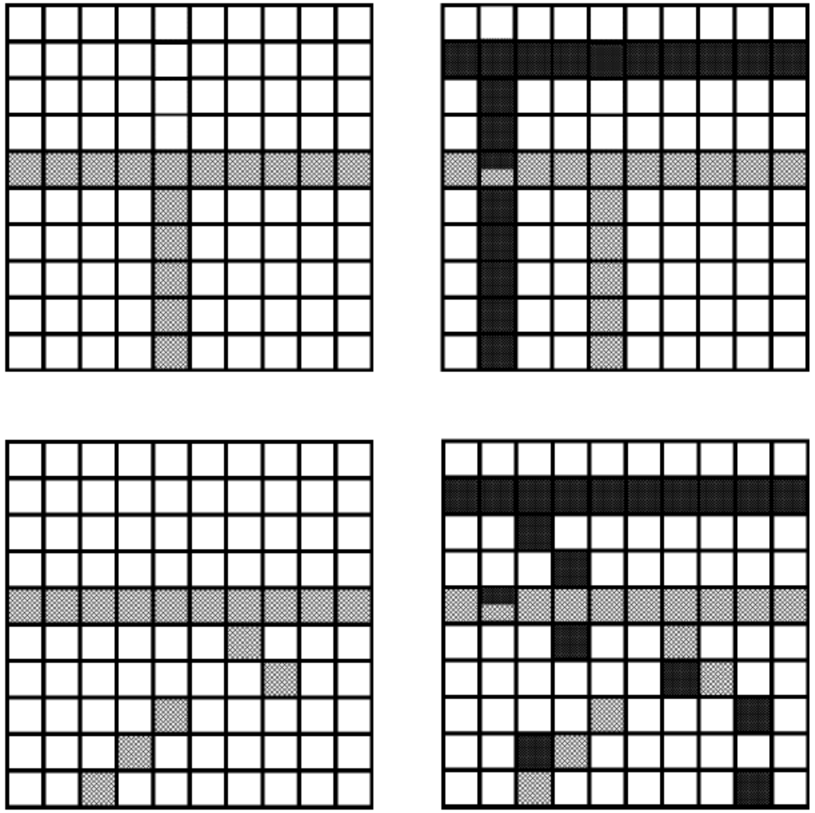
Problem:

- <span>2 quorums intersect in two nodes -> deadlock</span>



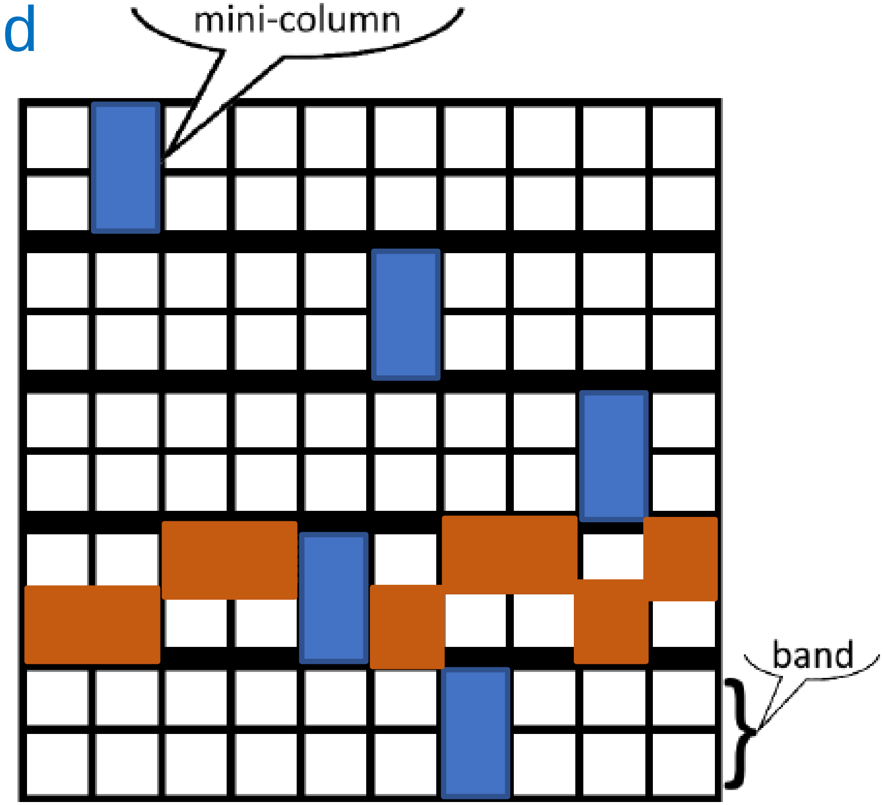
Grid Quorum System – Another Grid

Solution:
Try to get all locks in order (by id), if one is locked release all and start over.
-> at least one quorum will always make progress (the one with highest identifier locked currently)



B-Grid Quorum System

- Mini columns: one mini column in every band
- One band with at least one element per mini-column
- r = rows in a band, h = number of bands, d = count columns
- size of each quorum: h*r + d -1
- Has ideal properties:
  - work: $\theta(\sqrt{n})$
  - load: $\theta(\frac{1}{\sqrt{n}})$
  - asymptotic failure probability: 0



Byzantine Quorum System

- **f-disseminating**

  1) if the intersection of two quorums always contains f+1 nodes

  2) for any set of f byzantine nodes, there always is a quorum without byzantine nodes

  - *good model if data is self-authenticating, if not we need a stronger one*

- **f-masking**

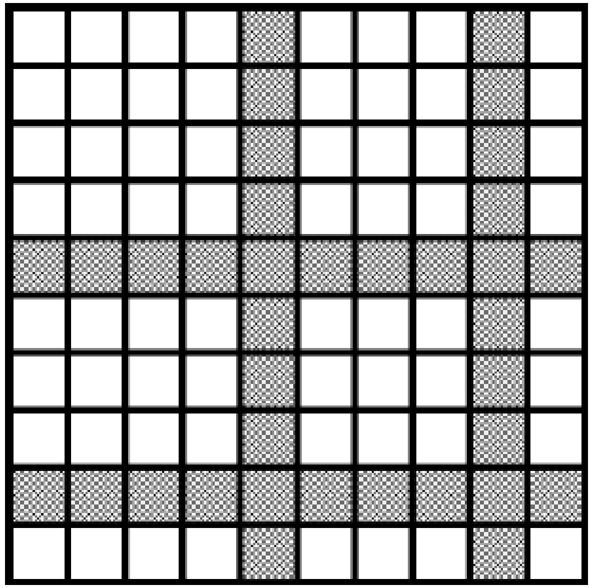
  1) if the intersection of two quorums always contains 2f+1 nodes

  2) for any set of f byzantine nodes, there always is a quorum without byzantine nodes

  - *correct nodes will always be in majority*

Byzantine Quorum System – M Grid

- $\sqrt{f+1}$ rows and $\sqrt{f+1}$ columns in each Quorum
  - $2 * \sqrt{f+1} * \sqrt{f+1} = 2\mathrm{f} + 2$ intersections
  - -> f-masking quorum systems
  - Example: f = 3



Chapter 22

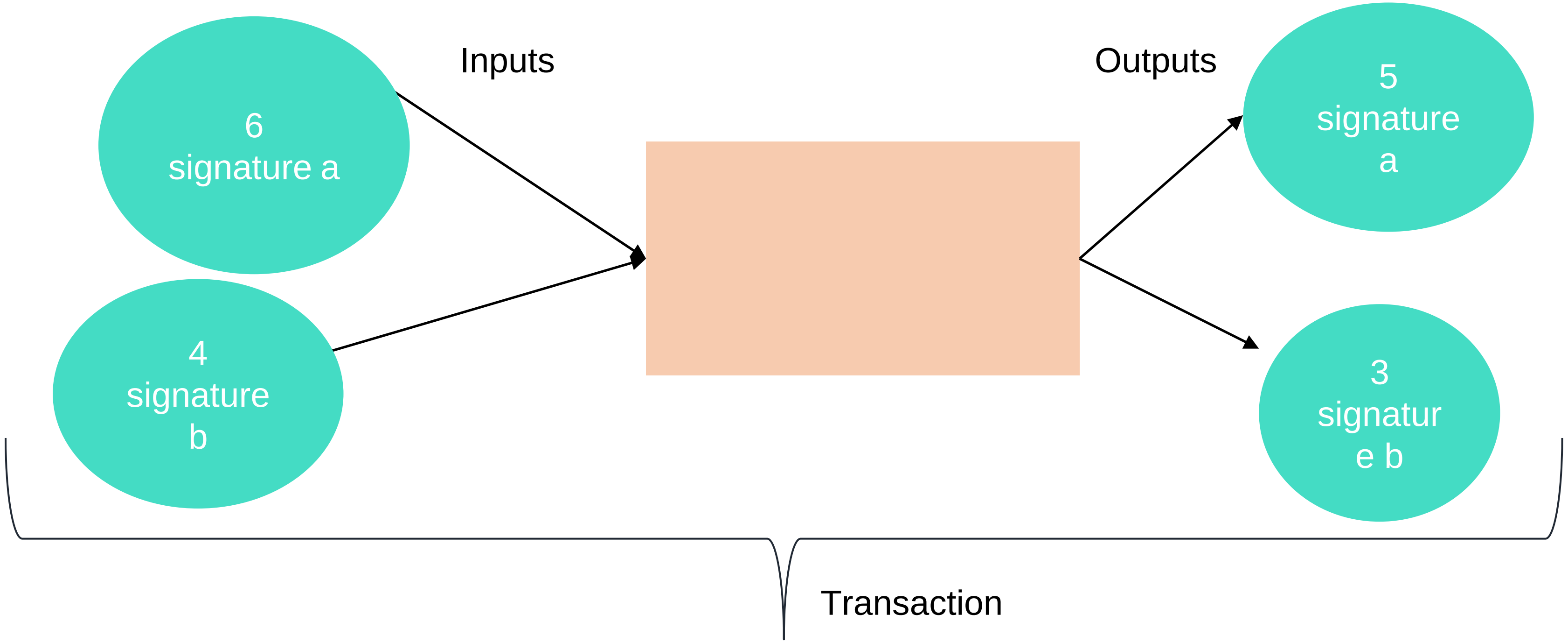
Eventual Consistency & Bitcoin

Consistency, Availability, and Partition Tolerance

- Consistency:
  - All nodes agree on the current state of the system
- Availability:
  - The system is operational and instantly processing incoming requests
- Partition tolerance:
  - Still works correctly if a network partition happens
- Good news:
  - achieving any two is very easy
- Bad news:
  - achieving three is impossible (CAP theorem)
- => Eventual Consistency:
  - Guarantees that the state is eventually agreed upon, but the nodes may disagree temporarily

Bitcoin

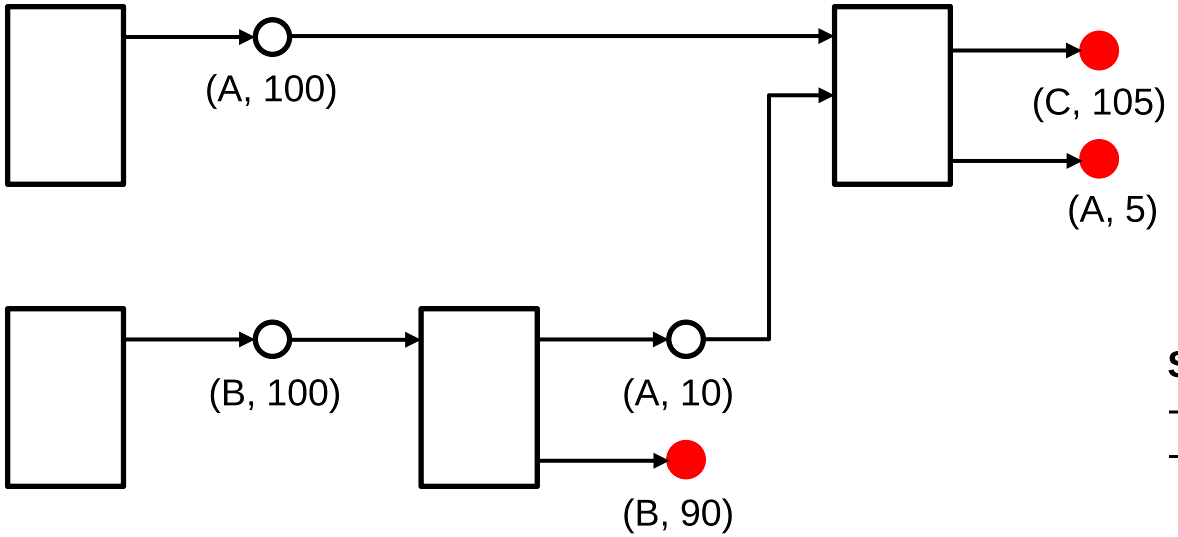
- decentralized network consisting of nodes
- users generate private/public key pair
  - address is generated from public key
  - it is difficult to get users “real” identity from public key



Bitcoin Transactions

- Conditions:
  - Sum of inputs must always be at least the sum of outputs
    - unused part is used as transaction fee, gets paid to miner of block
  - An input must always be some whole output, no splitting allowed!
  - Money that a user “has” is defined as sum of unspent outputs

Bitcoin Transactions

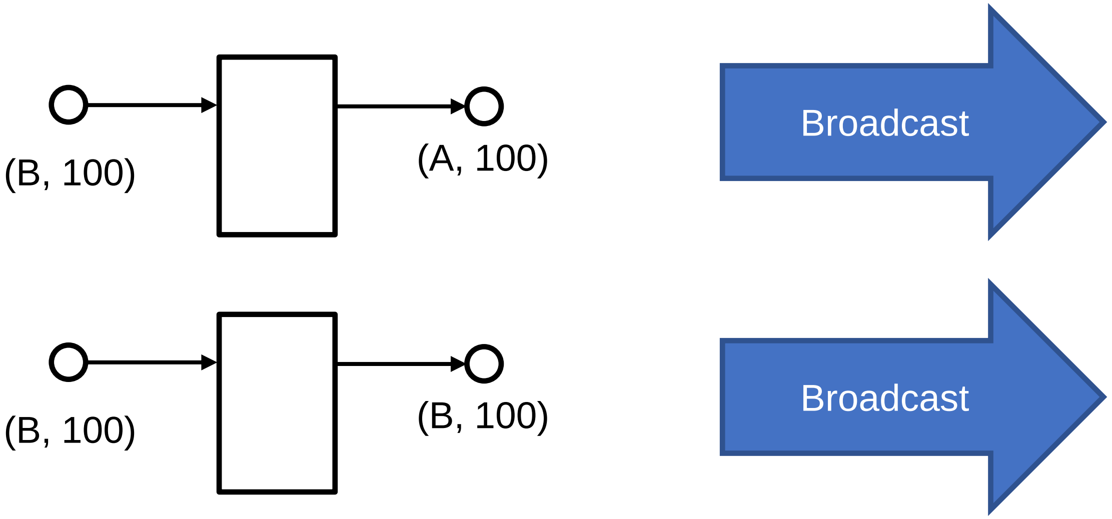


Set of unspent transaction outputs (UTXOs):
- This set is the shared state of Bitcoin
- The red outputs

Doublespend Attack

- Multiple transactions attempt to spend the same output

- Ex: In a transaction, an attacker pretends to transfer an output to a victim, only to doublespend the same amount in another transaction back to itself.

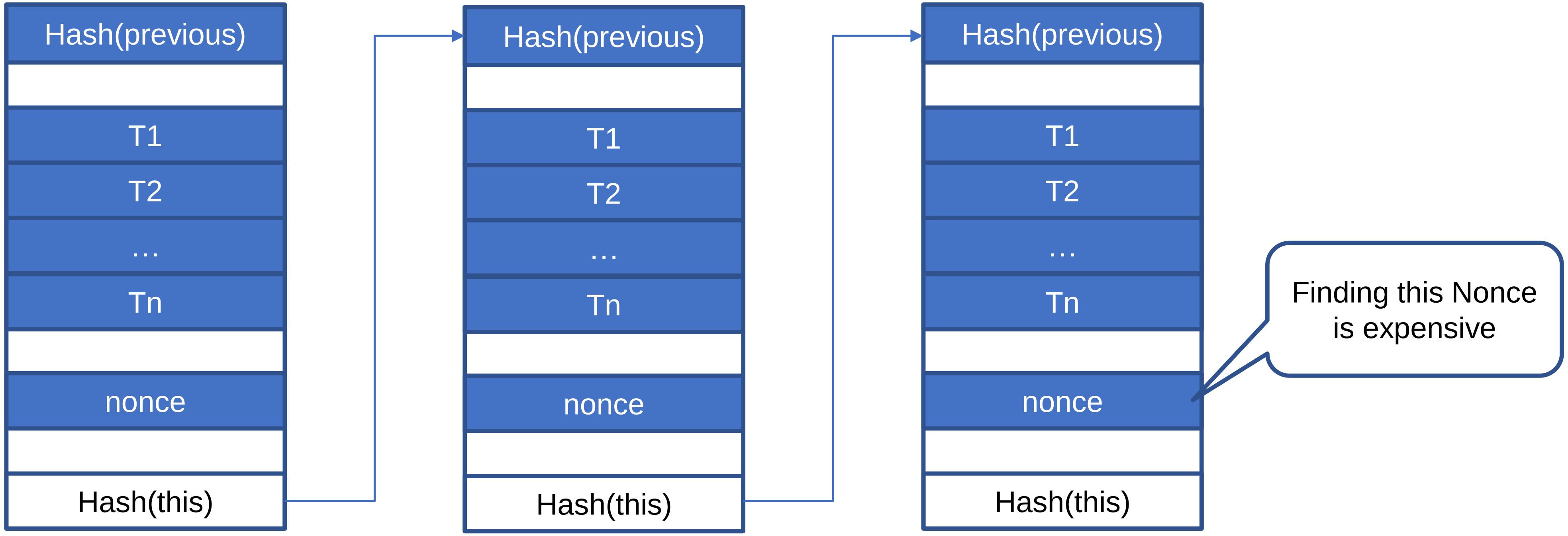


Proof-of-Work

- Right now we have infinitely growing memory pool and we can’t be sure that other nodes have the same pool
- Solution: Propagate memory pool through network and make sure everybody else will have same state
- Problem: How to avoid that everybody wants to propagate its own memory pool?
- Solution: Proof-of-Work
  - proof that you put a certain amount of work into propagating your memory pool

Block

- Data structure holding transactions reference to previous blocks and a nonce.
- Miner creates blocks with transactions from the memory pool



Proof-of-Work

- Mining Blocks requires to proof that a certain amount of computational resources has been utilized

$$F_d(c,x) \rightarrow \{true, false\}$$

  - d: difficulty (is adapted all 24h)
  - c: challenge (the transactions and the hash of the previous block)
  - x: nonce (has to be found)

  - For fixed parameters d and c, finding x such that the function

Bitcoin PoW: $$\mathcal{F}_d(c,x) \rightarrow \mathrm{SHA256}(\mathrm{SHA256}(c|x)) < \frac{2^{224}}{d}.$$

Bitcoin chooses the difficulty such that a block is created all ~10 min

Mining

- Why should someone mine blocks?
  - You get a reward for each block you mine
  - You get the fee in the transactions

Bitcoin:

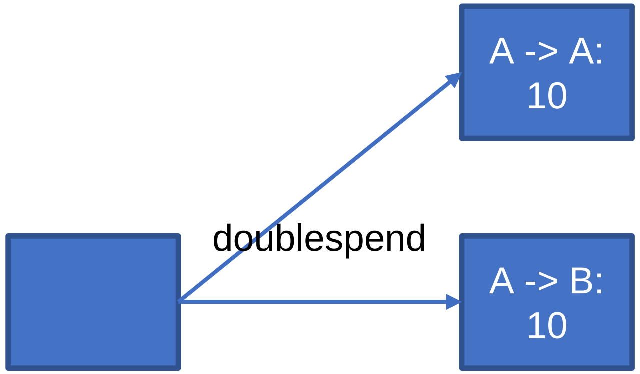
- Reward started at 50B and it is being halved every 210,000 blocks or 4 years in expectation
- This bounds the total number of Bitcoins to 21 million
- What will happen after that?

- Fee is the positive difference of input-output
- Problem: Miner go for transactions which have a high fee.

- Problem: More miners -> more blocks are mined -> higher difficulty -> more Power needed

How does this prevent double spending?

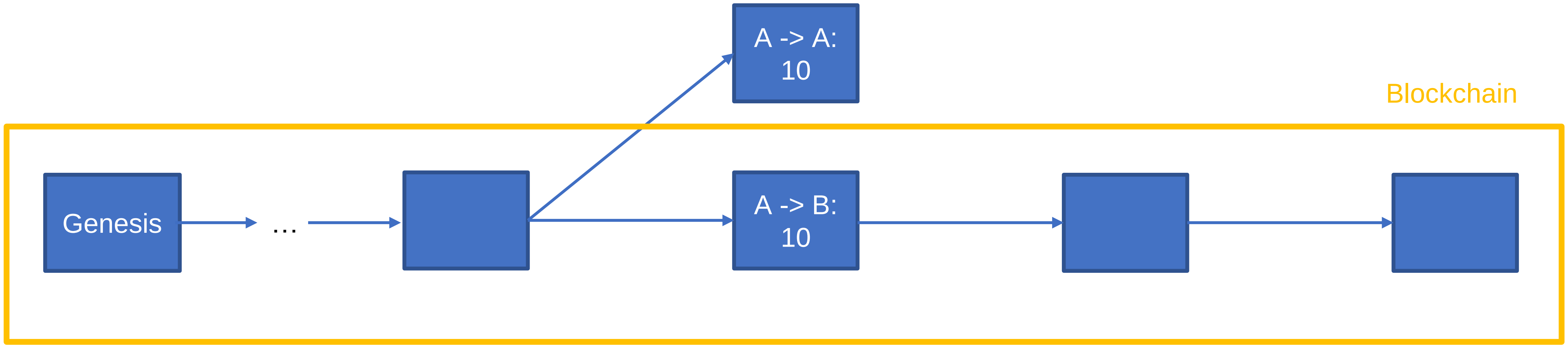
- An intruder needs to have 50% of computation power to be faster in mining than all other together



The goal of Alice is now to make the branch where she spends the money to herself growing faster.

Blockchain

- Starts with the genesis blog and is the longest path from this genesis block to a leaf.
- Consistent transaction history on which all nodes eventually agree

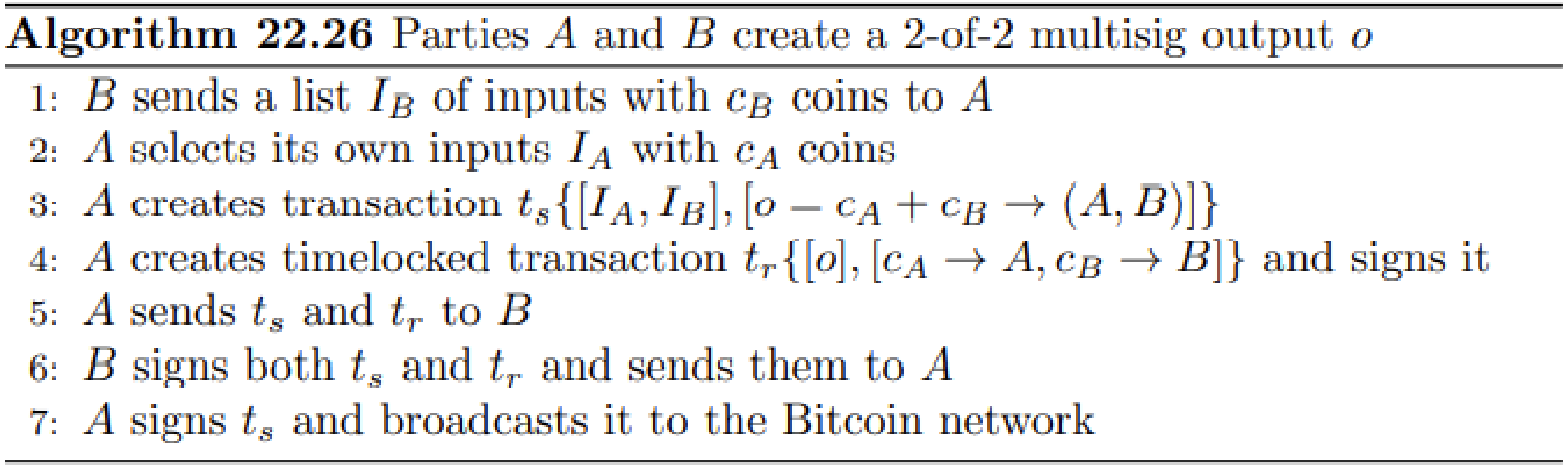


Note: To ensure that you'll get the money you should wait 5-10 further blocks

Smart Contracts

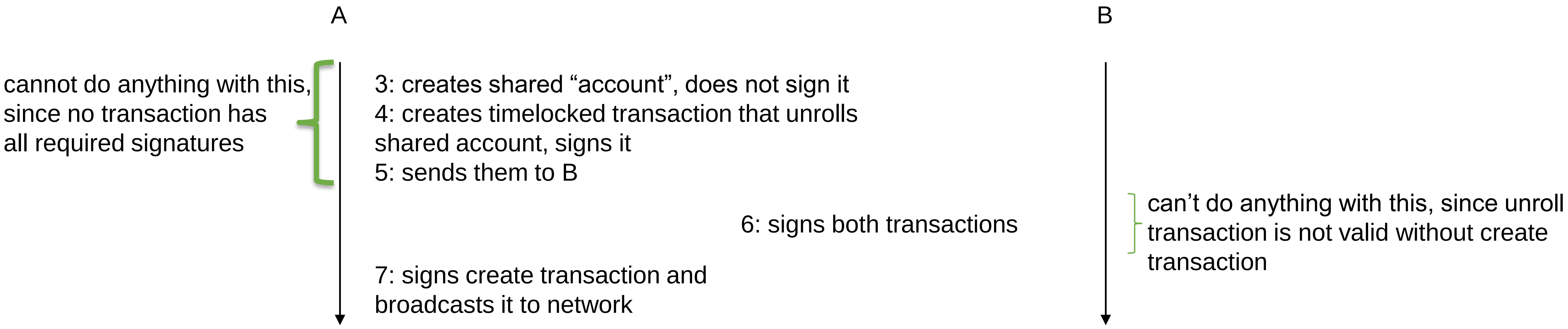
- Contract between two or more parties, encoded in such a way that correct execution is guaranteed by blockchain
  - Timelock: transaction will only get added to memory pool after some time has expired
    - Micropayment channel:
      - Idea: Two parties want to do multiple small transactions but want to avoid fees. So they only submit first and last transaction to blockchain and privately do everything inbetween

Micropayment Channel Setup Transaction



Algorithm 22.26 Parties A and B create a 2-of-2 multisig output o

```
1: $B$ sends a list $I_B$ of inputs with $c_B$ coins to $A$
2: $A$ selects its own inputs $I_A$ with $c_A$ coins
3: $A$ creates transaction $t_s\{[I_A, I_B], [o - c_A + c_B \rightarrow (A, B)]\}$
4: $A$ creates timelocked transaction $t_r\{[o], [c_A \rightarrow A, c_B \rightarrow B]\}$ and signs it
5: $A$ sends $t_s$ and $t_r$ to $B$
6: $B$ signs both $t_s$ and $t_r$ and sends them to $A$
7: $A$ signs $t_s$ and broadcasts it to the Bitcoin network
```



cannot do anything with this, since no transaction has all required signatures

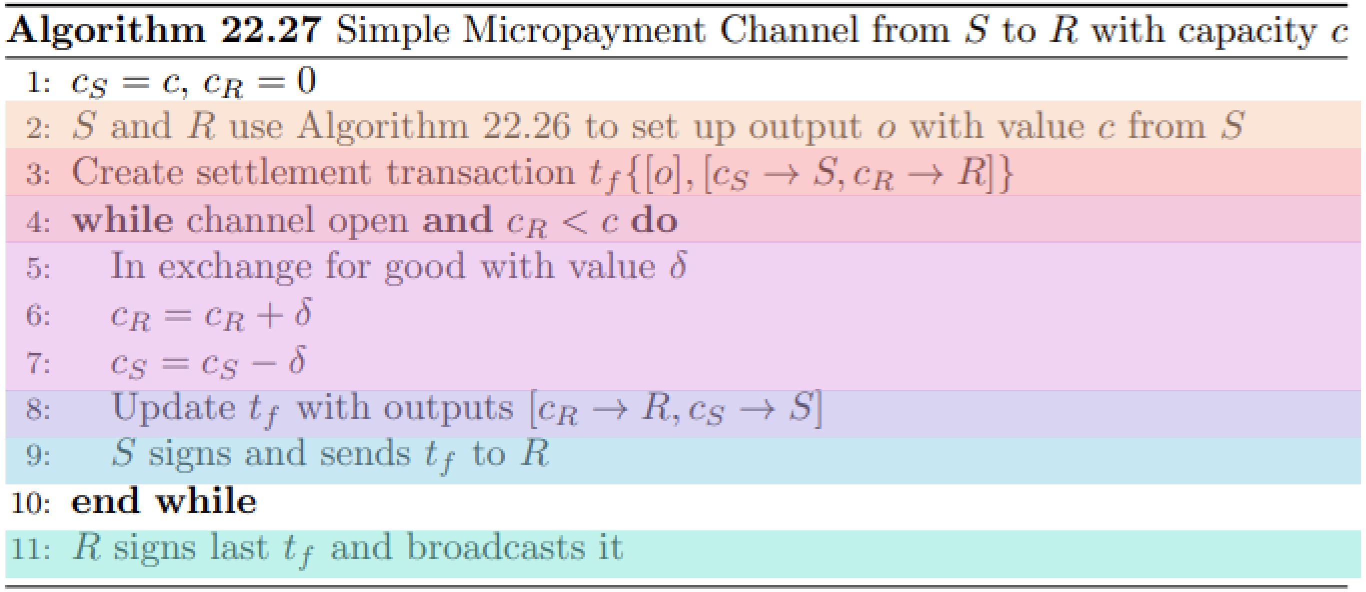
3: creates shared “account”, does not sign it
4: creates timelocked transaction that unrolls shared account, signs it
5: sends them to B

6: signs both transactions

can’t do anything with this, since unroll transaction is not valid without create transaction

7: signs create transaction and broadcasts it to network

Micropayment Channel



Algorithm 22.27 Simple Micropayment Channel from S to R with capacity c

```
$c_S = c$, $c_R = 0$
$S$ and $R$ use Algorithm 22.26 to set up output $o$ with value $c$ from $S$
Create settlement transaction $t_f\{[o], [c_S \rightarrow S, c_R \rightarrow R]\}$
while channel open and $c_R < c$ do
    In exchange for good with value $\delta$
    $c_R = c_R + \delta$
    $c_S = c_S - \delta$
    Update $t_f$ with outputs $[c_R \rightarrow R, c_S \rightarrow S]$
    $S$ signs and sends $t_f$ to $R$
end while
$R$ signs last $t_f$ and broadcasts it
```

set up shared account and unrolling

create settlement transaction

while sender still has money and timelock not expired

exchange goods and adapt money

update settlement transactions with new values

S signs transaction and sends it to R

R signs last transaction and broadcasts it
before timelock expires

Why does s sign it?

- like this, R always holds all fully signed transactions and can choose the last one (where he gets the most money)
- S cannot submit any transaction, so S cannot get the goods and later submit a transaction where S did not pay the money for it

Quiz - Quorums

a) Does a quorum system exist, which can tolerate that all nodes of a specific quorum fail? Give an example or prove its nonexistence.

- No such quorum system exists.

b) Consider the nearly all quorum system, which is made up of n different quorums, each containing n − 1 servers. What is the resilience of this quorum system?

- Just 1 - as soon as 2 servers fail, no quorum survives.

c) Can you think of a quorum system that contains as many quorums as possible? Note: the quorum system does not have to be minimal.

- 2^{n-1} quorums. All quorums overlap exactly in one single node. Each element of the powerset of the remaining n − 1 nodes joined with this special node is a quorum.