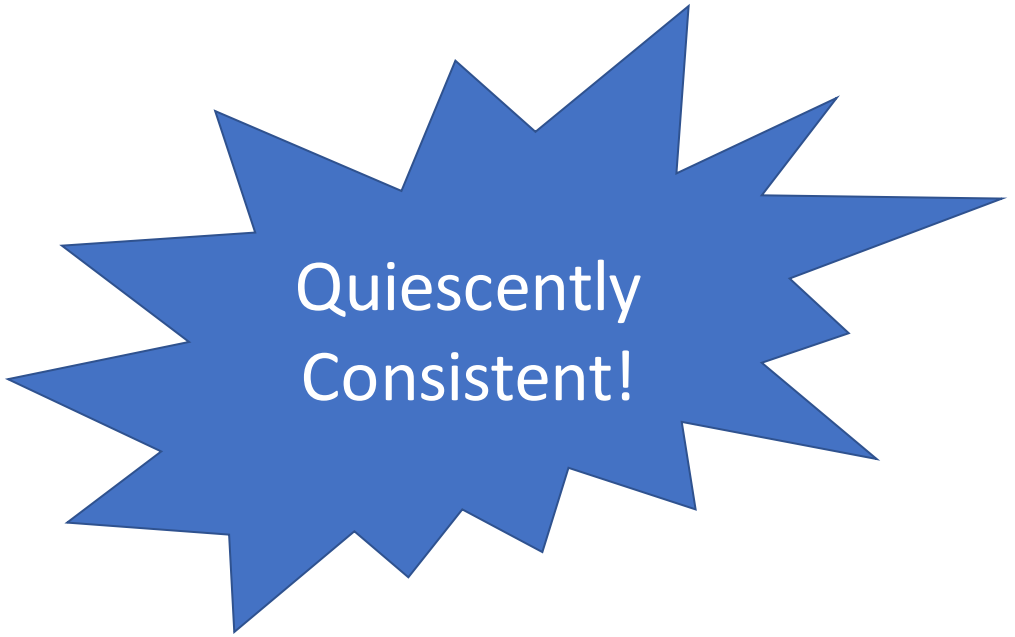
# Computer Systems

Exercise 7



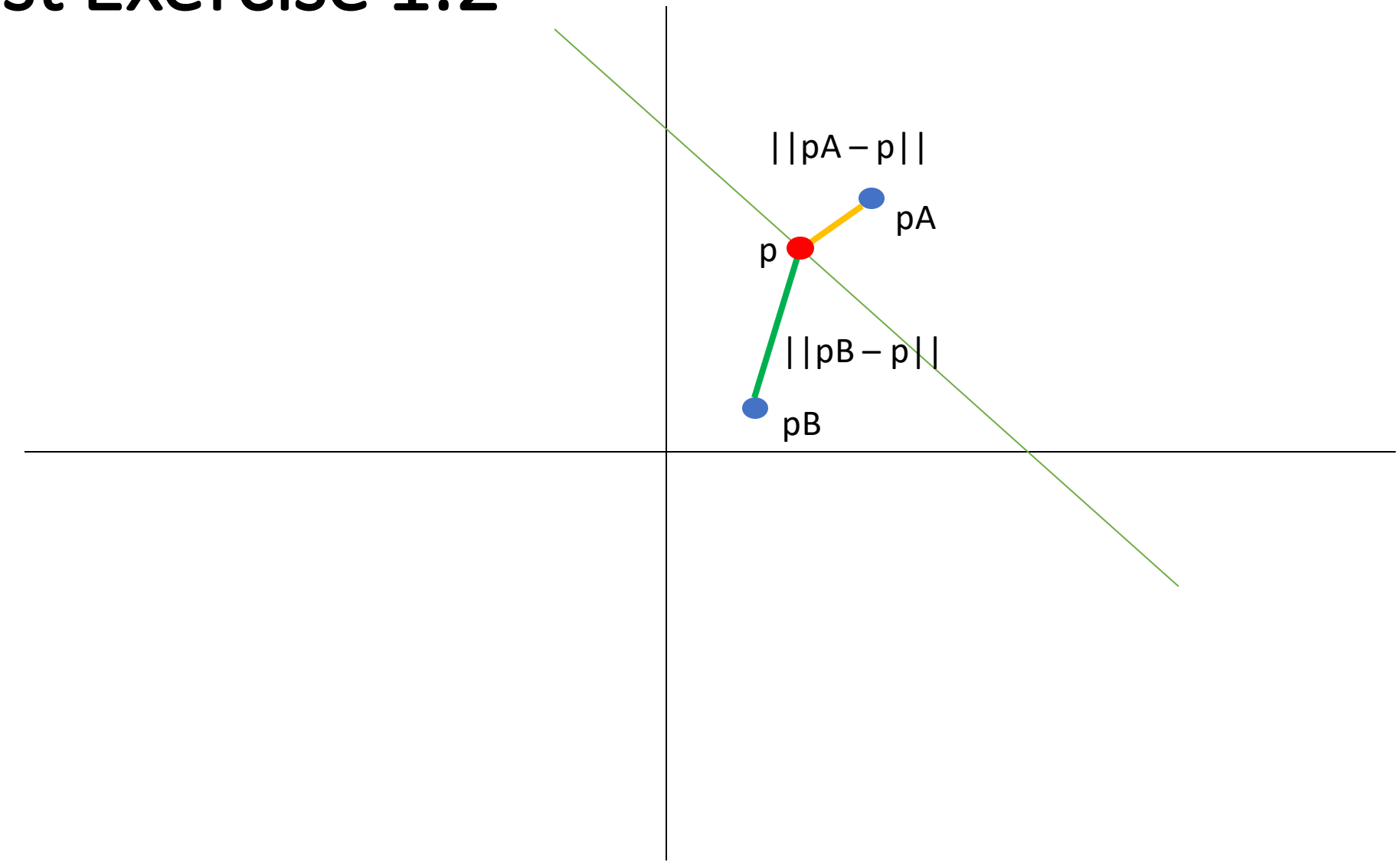
# Last Exercise

## 1.2 Time Difference of Arrival

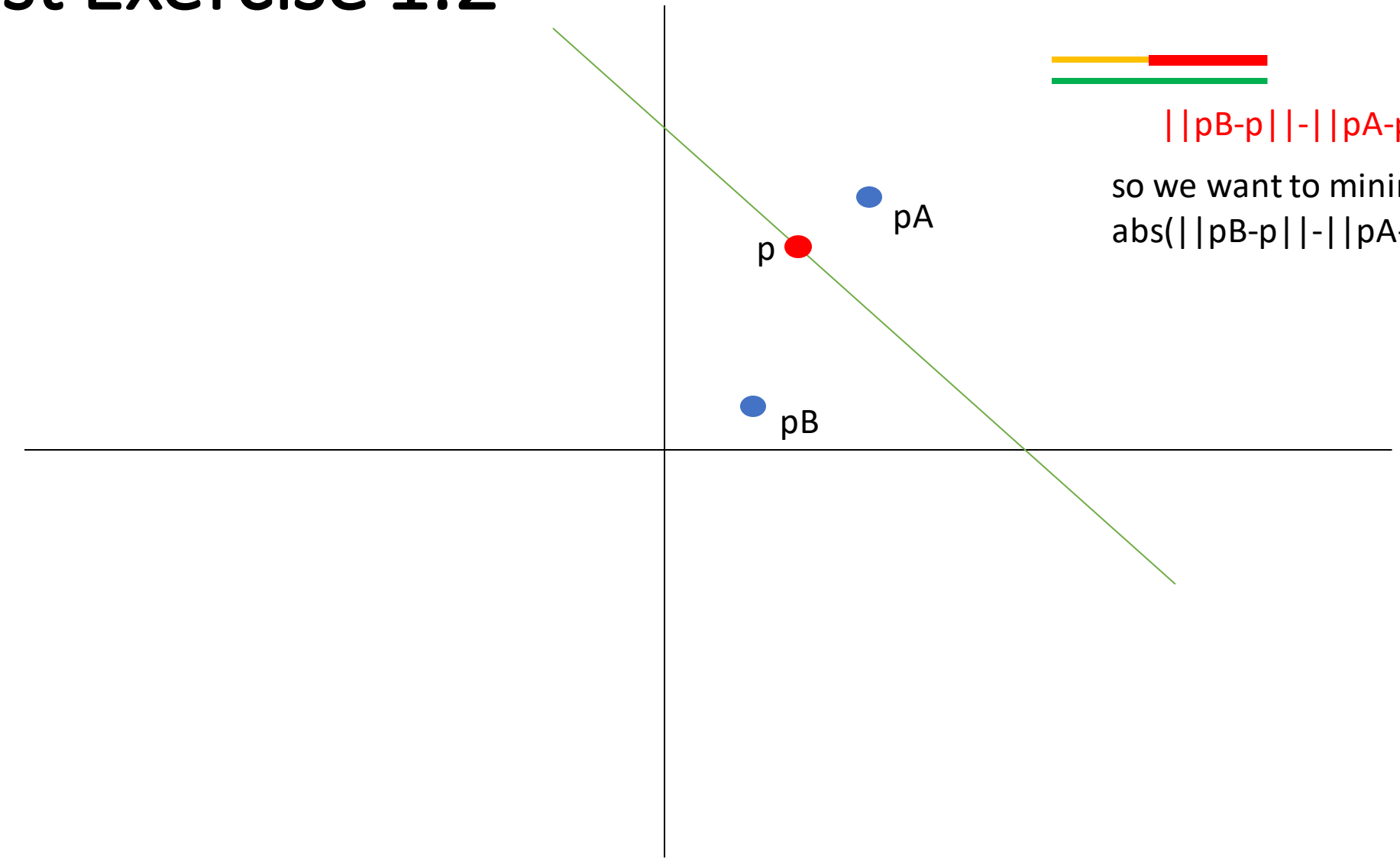
Assume you are located on a line $y = -x + 8$ km in the two dimensional plane. You receive the GPS signals from satellites $A$ and $B$. Both signals are transmitted exactly at the same time $t$ by both satellites. You receive the signal from satellite $A$ 3.3 µs before the signal of satellite $B$. At 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.

# Last Exercise 1.2



# Last Exercise 1.2






||pB-p||-||pA-p|| = 1km

so we want to minimize:
abs(||pB-p||-||pA-p|| - 1)

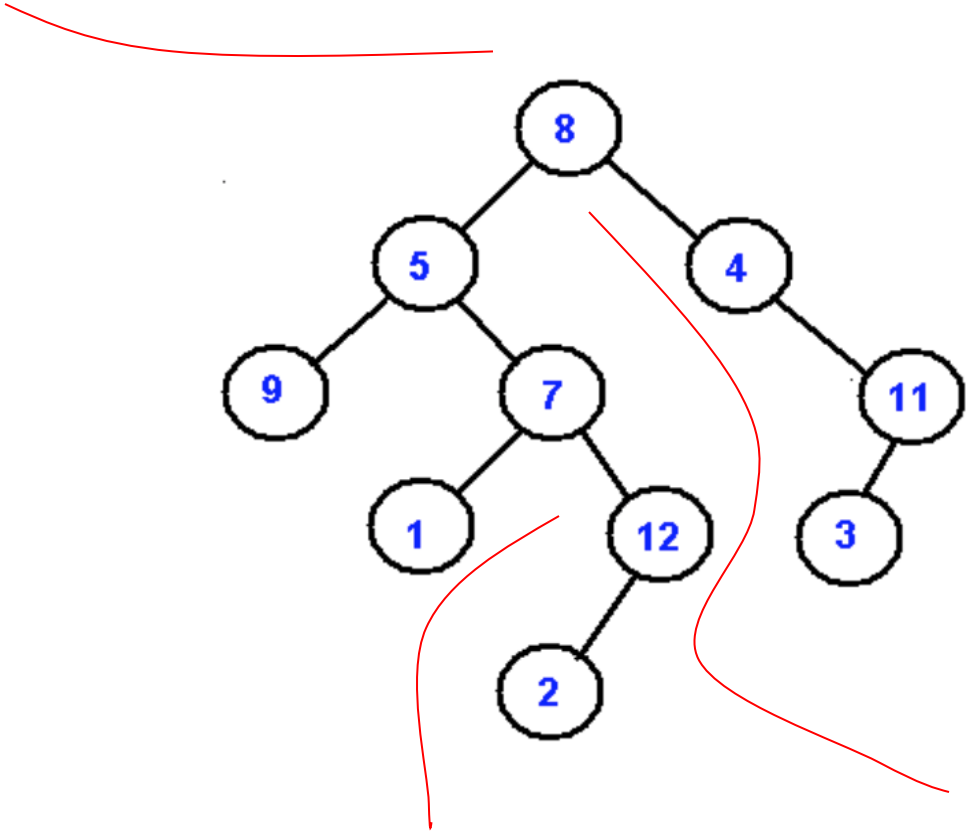
# Last Exercise

## 1.3 Clock Synchronization: Spanning Tree

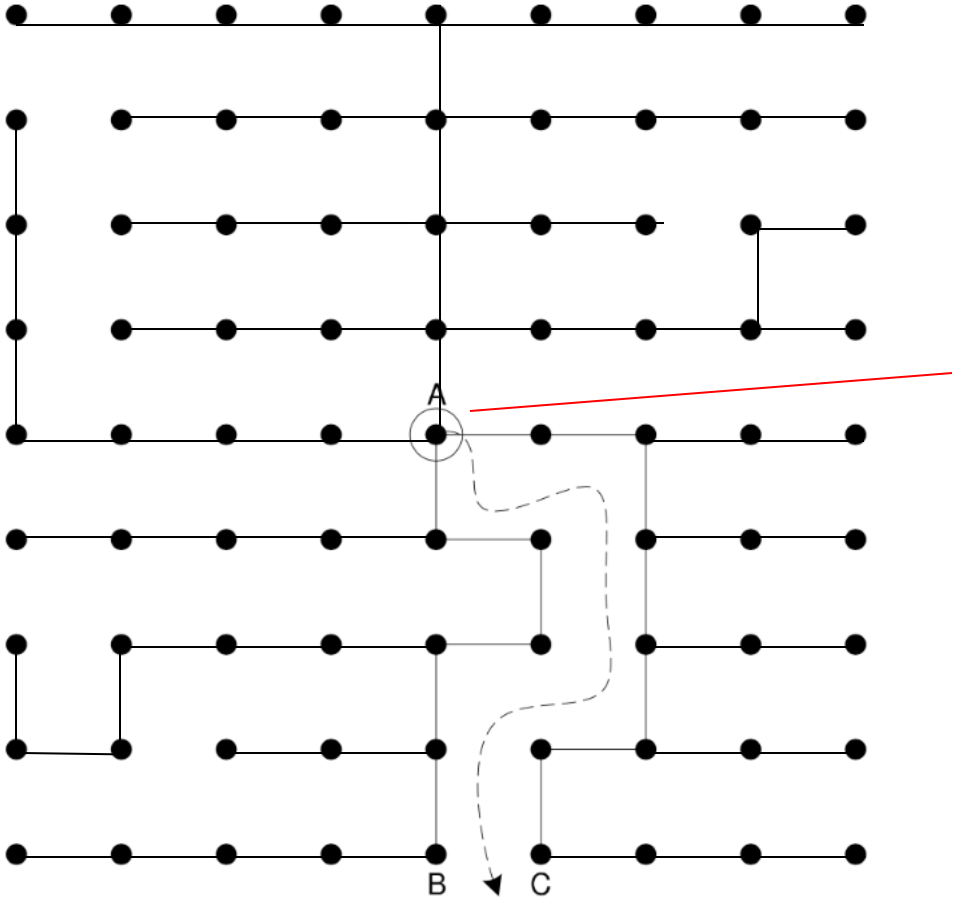
Common clock synchronization algorithms (e.g. TPSN, FTSP) rely on a spanning tree to perform clock synchronization. Finding a good spanning tree for clock synchronization is not trivial. Nodes which are neighbors in the network graph should also be close-by in the resulting tree. Show that in a grid 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

General layout of spanning tree:



# Last Exercise 1.3



# Quorum Systems - Resilience

The largest f such that:

f nodes fail

and there is still a quorum without failed nodes.

# Quorum Systems - Load

- of access strategy on node: Probability the node is accessed by this strategy
- of access strategy on quorum system induced by access strategy: maximal load on a single node
- of a quorum system: load induced on the system by the best access strategy

# Quorum Systems - Work

- of a quorum: The size of that quorum (I.e. how many nodes will process a request to this quorum)
- induced by an access strategy on a quorum system: Expected number of nodes accessed by that strategy
- of a quorum system: The work induced by the best access strategy

# Load and Work

- Must be measured together!
  - I.e. with the same strategy

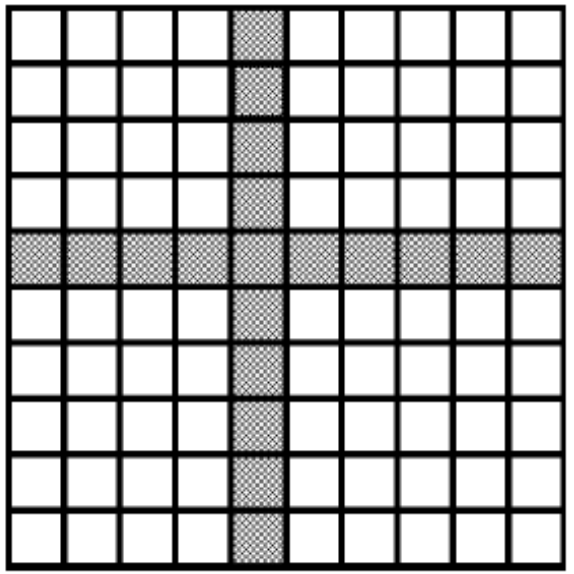
- Load: Measure of the processing a node has to perform
- Work: Measure of the processing the entire system performs

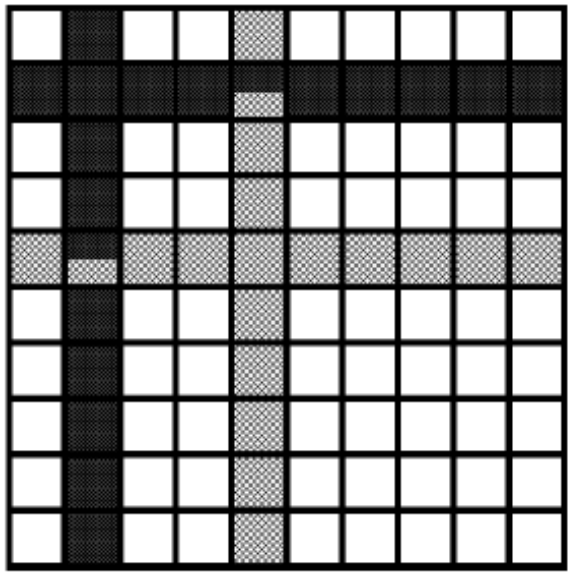
# Grid quorum system – Basic Grid

Problem:

- 2 quorums intersect

in two nodes -> deadlock





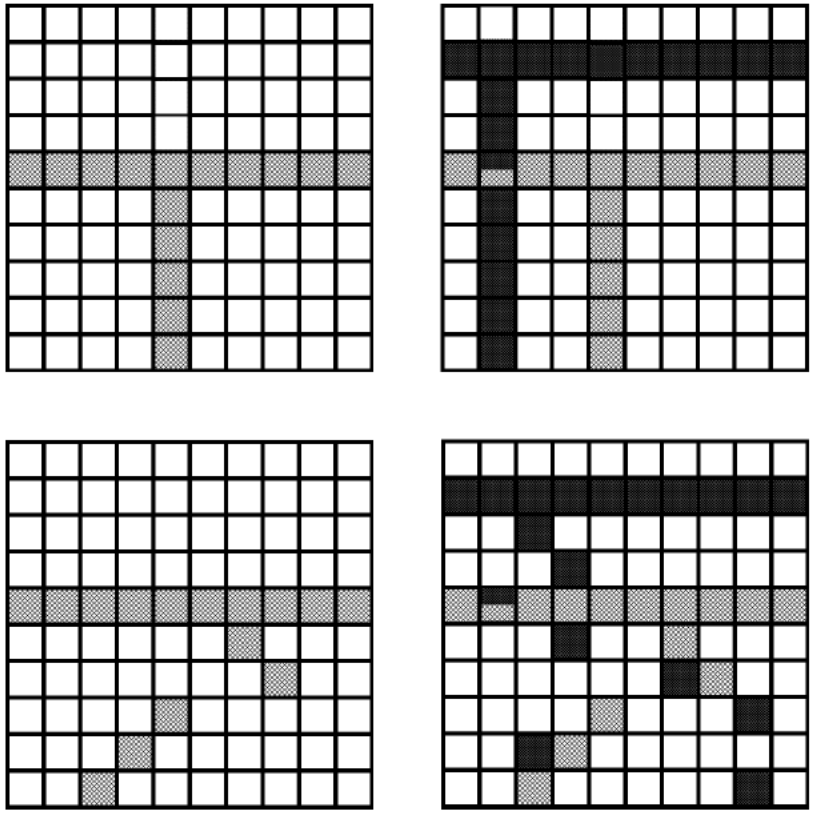
# Grid quorum system – Another Grid

Better:

- Two quorums won't run into a deadlock anymore

Problem:

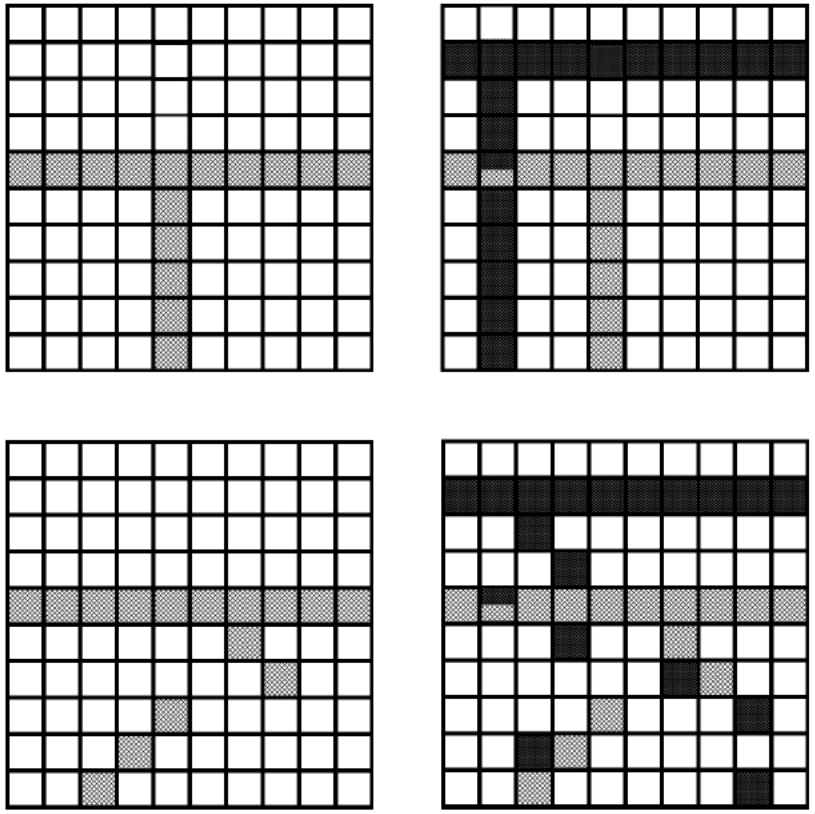
- It can still happen with three or more threads



# Grid quorum system – Better 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)

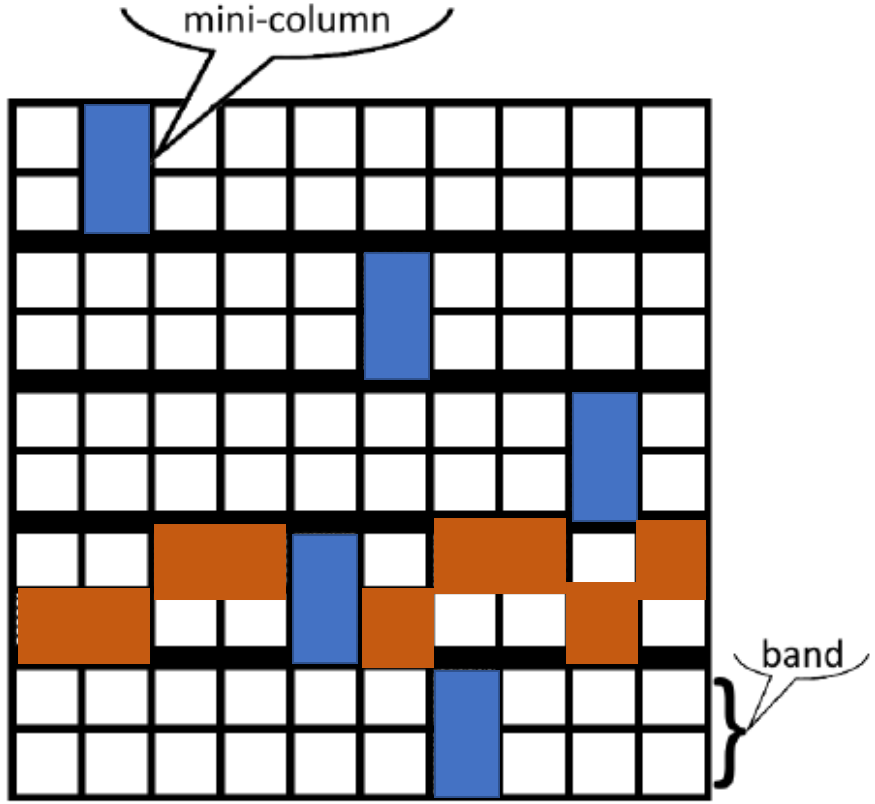


# Fault tolerance

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

# B-grid quorum system

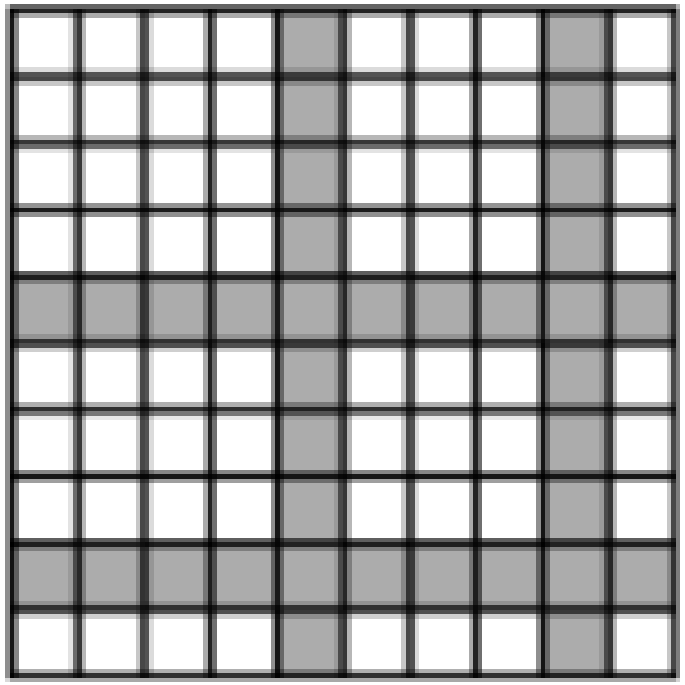
- Mini columns: one mini column in every band
- One band with at least one element per mini-column
- r = rows in a band, h = number of bands, d = count columns
- size of each quorum: h*r + d -1
- Has ideal properties:
  - work: $\theta(\sqrt{n})$
  - load: $\theta(\frac{1}{\sqrt{n}})$
  - asymptotic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



# Byzantine quorum systems

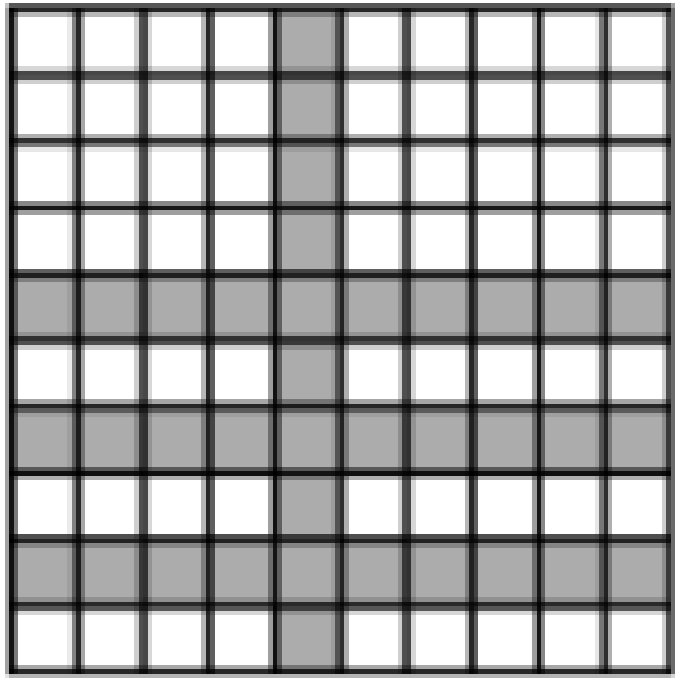
- **f-disseminating**
  - if the intersection of two quorums always contains f+1 nodes
  - for any set of f byzantine nodes, there always is a quorum without byzantine nodes
  - *good model if data is self-authenticating, if not we need a stronger one*
- **f-masking**
  - if the intersection of two quorums always contains 2f+1 nodes
  - for any set of f byzantine nodes, there always is a quorum without byzantine nodes
  - *correct nodes will always be in majority*

# M-Grid



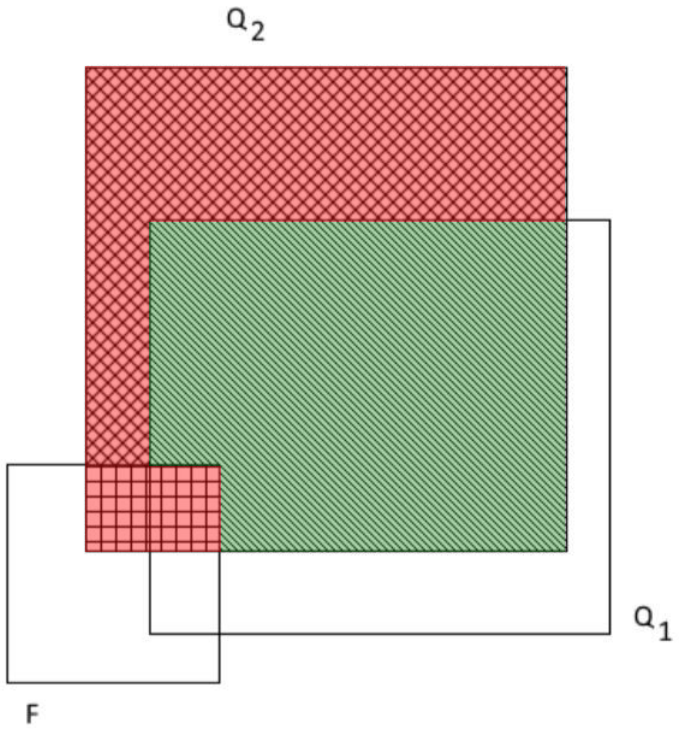
**Definition 15.30** ($M$-Grid quorum system)**.** *The* $M$***-Grid*** *quorum system is constructed as the grid quorum as well, but each quorum contains* $\sqrt{f+1}$ *rows and* $\sqrt{f+1}$ *columns of nodes, with* $f \leq \frac{\sqrt{n}-1}{2}$.

# f-Masking Grid



**Definition 15.28** ($f$-masking Grid quorum system)**.** *A* $f$***-masking Grid*** *quorum system is constructed as the grid quorum system, but each quorum contains one full column and* $f+1$ *rows of nodes, with* $2f+1 \leq \sqrt{n}$.

# Opaque quorum systems




Also we want at least one Quorum that contains no byzantine nodes

Figure 15.34: Intersection properties of an opaque quorum system. Equation (15.33.1) ensures that the set of non-byzantine nodes in the intersection of $Q_1, Q_2$ is larger than the set of out of date nodes, even if the byzantine nodes "team up" with those nodes. Thus, the correct up to date value can always be recognized by a majority voting.

## Quiz

### 1.1 The Resilience of a Quorum System

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

**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?

**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.*

# 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)

# Side Note: Availability vs. Reliability

- Availability: The system is operating and can be used at any given time

- Reliability: The system will eventually work and not lose any previous work.

- E.g. my old phone:
- Reliable (as long as the flash memory doesn't give out).
- But not available when it spontaneously decides to freeze for a whole minute (Seriously, WHY??)

# Side Note: Availability vs. Reliability

- Another example: Let's say I promised someone a rubber duck
- But I didn't have it right now
  - => My rubber duck delivery is not available
- But if I eventually delivered one
  - => It's reliable

# Really overstretching the concept

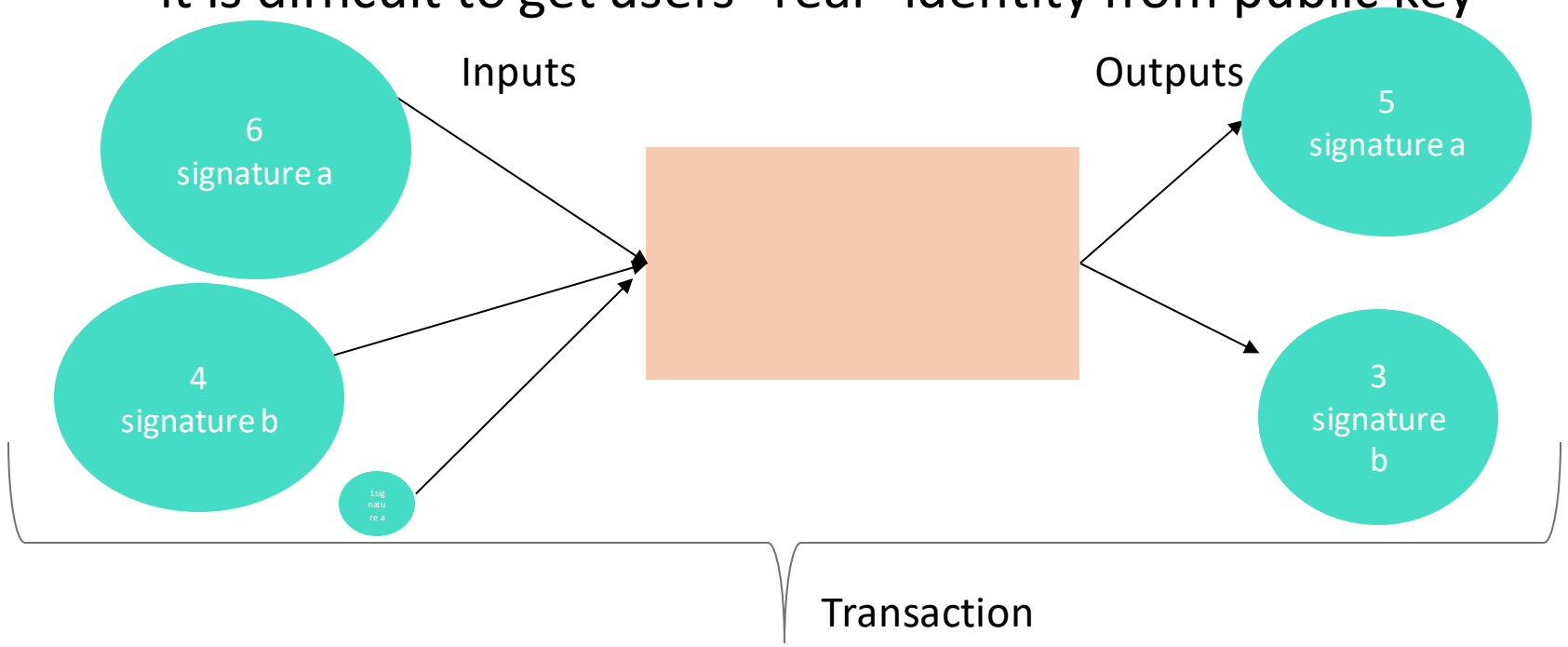
The following information is available in the script.
But is it reliable?

## 16.2 Bitcoin

**Definition 16.8** (Bitcoin Network). *The Bitcoin network is a randomly connected overlay network of a few thousand* ***nodes****, controlled by a variety of owners. All nodes perform the same operations, i.e., it is a homogenous network and without central control.*

# 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

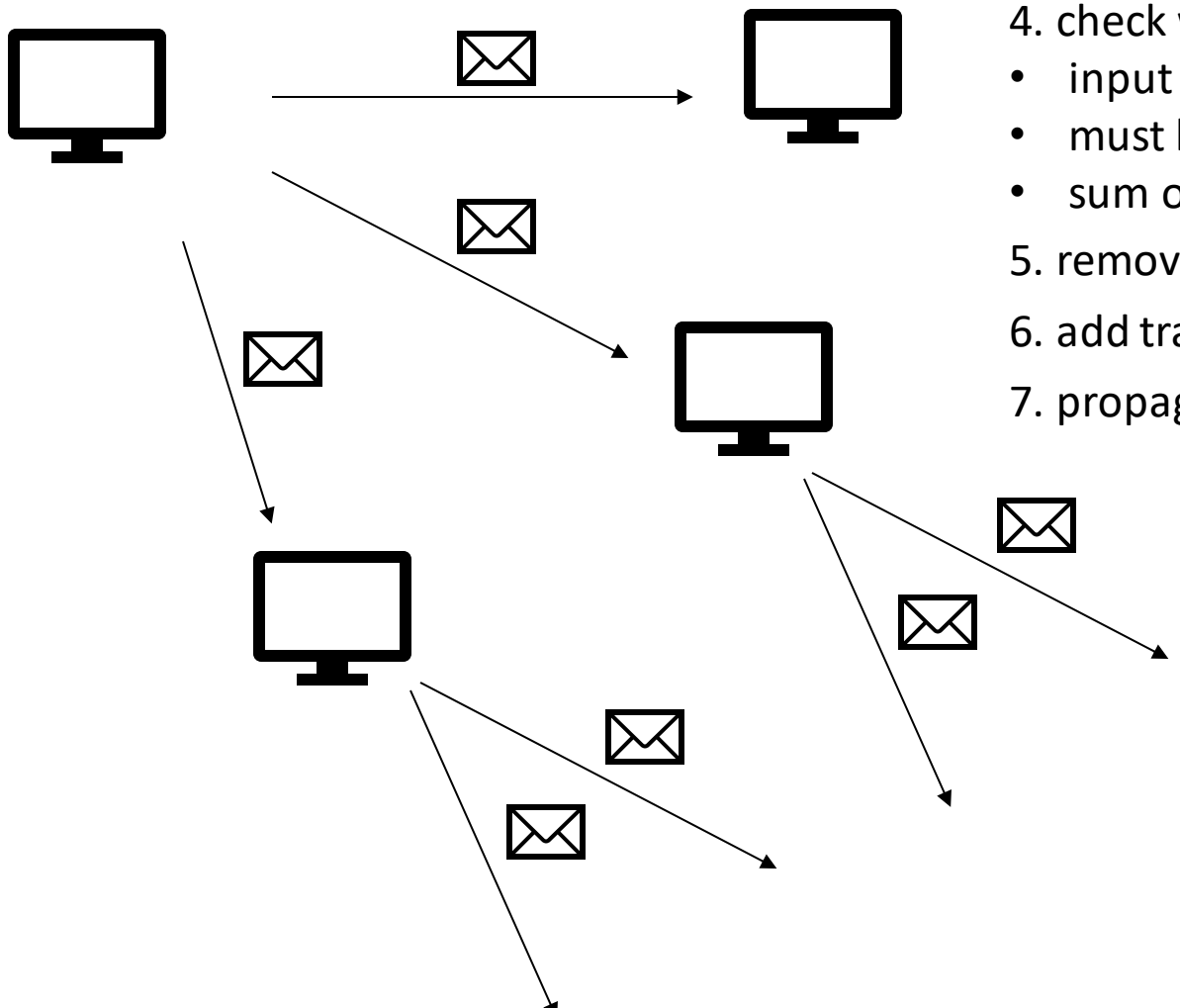


# Bitcoin transactions

- Conditions:
  - Sum of inputs must always be >= 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

# Blockchain

1. issue transaction
2. add transaction to local history
3. send transaction to other nodes in network



4. check whether transaction is valid
   - input of transaction must be in local UTXO
   - must have valid signature
   - sum of inputs >= sum of outputs
5. remove any input of transaction from local UTXO
6. add transaction to local history
7. propagate transaction further

# Blockchain

- 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 it’s own memory pool?
- Solution: Proof-of-Work
  - proof that you put a certain amount of work into propagating your memory pool

# In Bitcoin

- A node is allowed to propagate block as soon as it has calculated a hash with some special property
  - no better method than iterating through different input values until the right hash appears
- Problem: What happens if two nodes calculate a correct hash at the same time? Some nodes will accept one block then, the other ones another block
- Solution: Nodes will always accept the block with the longest chain, so in order to keep up the split, the nodes would have to always calculate two hashes at the same time -> probability goes to 0

# Bitcoin

- What does a node actually do when it receives a block:
  - node has current block $B_{max}$ with height $h_{max}$
  - it connects received block in tree as child of its parent
  - if height of new block is bigger than $h_{max}$ then
    - set $B_{max}$ and $h_{max}$
    - compute new UTXO
    - clean up memory pool
- Result: all nodes that accept block will see same history of transactions

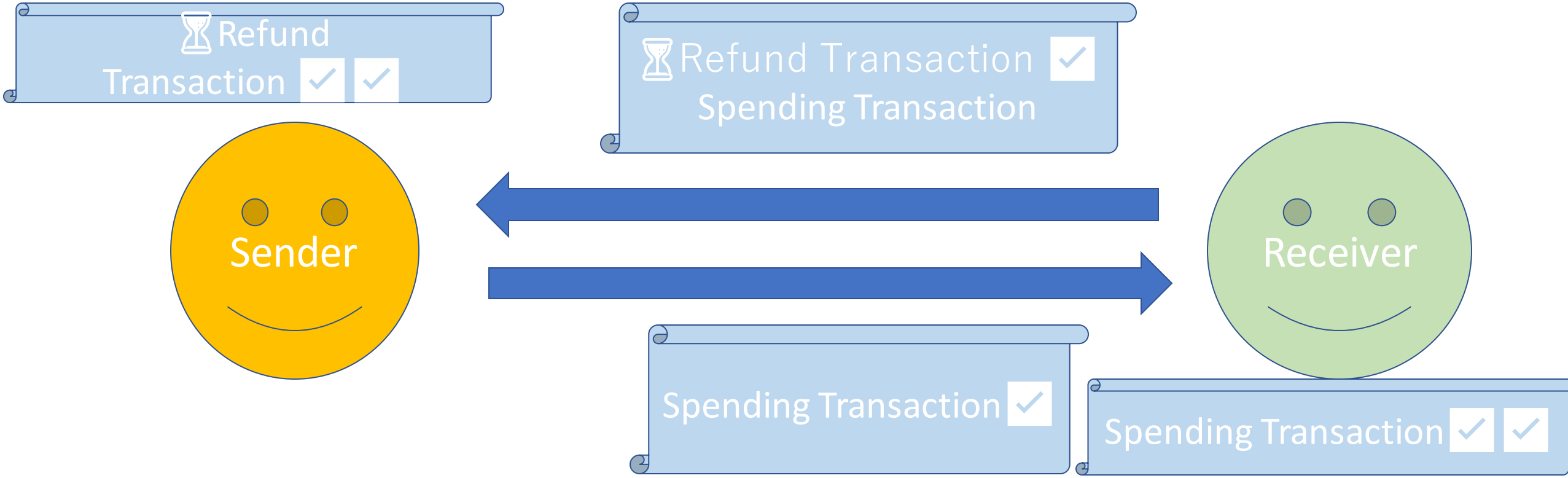
# Smart Contracts

- 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

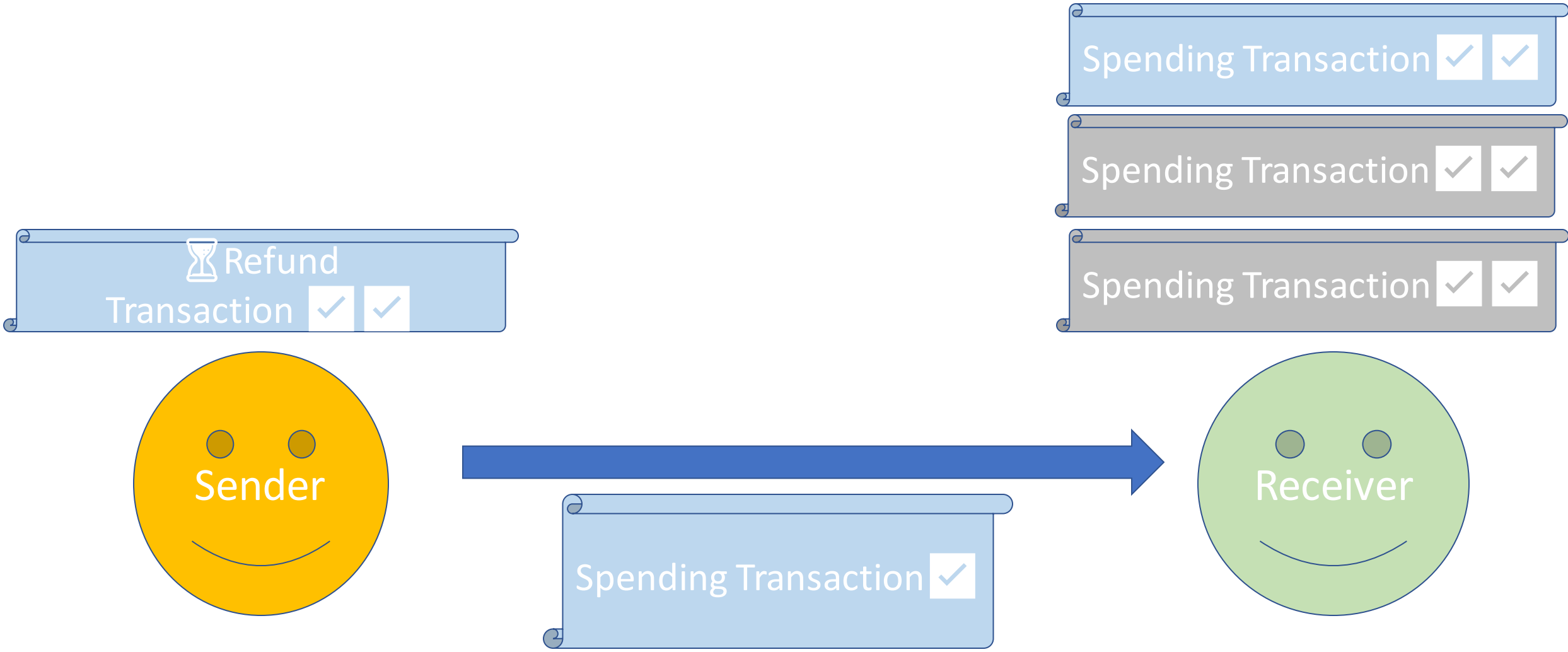
# Smart Contracts – Micropayment Channel

- Idea: Two parties want to do multiple small transactions
- But they want to avoid the fees and transaction times of the Bitcoin network.
- So they only submit first and last transaction to the blockchain and privately do transactions between themselves

# Micropayment Channel - Setup



# Micropayment Channel - Payment



# Micropayment Channel - Settlement

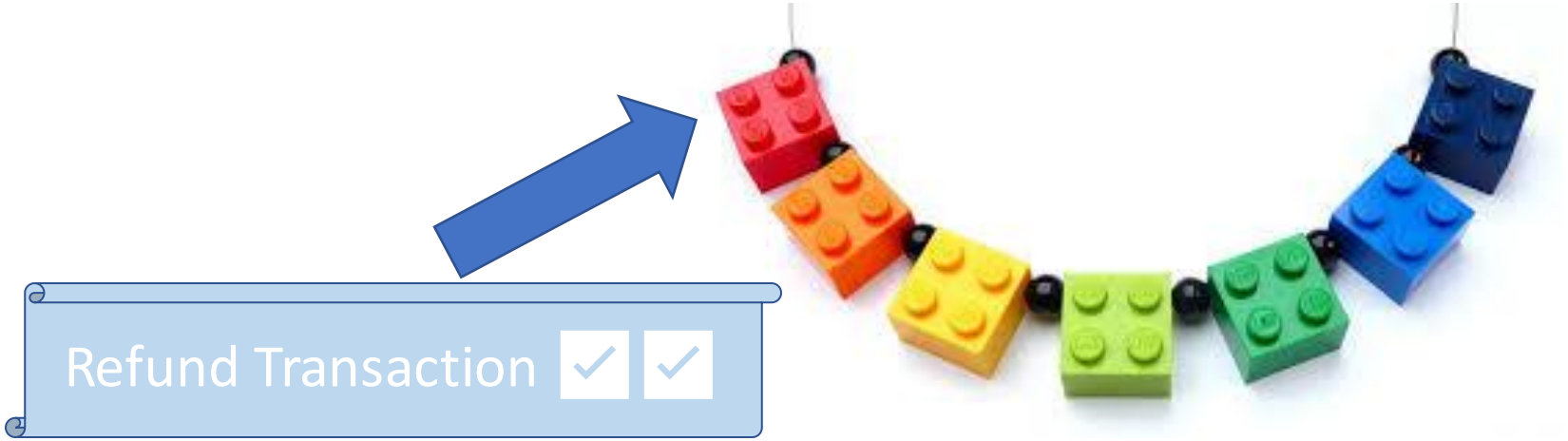
Refund Transaction

Spending Transaction

Sender

Receiver

# Micropayment Channel - Settlement




Sender

Receiver

## Quiz

### 2.1 Delayed Bitcoin

In the lecture we have seen that Bitcoin only has eventual consistency guarantees. The state of nodes may temporarily diverge as they accept different transactions and consistency will be re-estalished eventually by blocks confirming transactions. If, however, we consider a delayed state, i.e., the state as it was a given number $\Delta$ of blocks ago, then we can say that all nodes are consistent with high probability.

**a)** Can we say that the $\Delta$-delayed state is strongly consistent for sufficiently large $\Delta$?

**b)** Reward transactions make use of the increased consistency by allowing reward outputs to be spent after *maturing* for 100 blocks. What are the advantages of this maturation period?