Distributed Systems

Exercise Session 4

Selma Steinhoff: selmas@ethz.ch

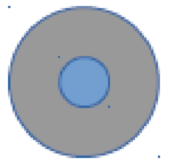
Disclaimer: these are not official slides, there might be mistakes, treat them this way

Quorum Systems:

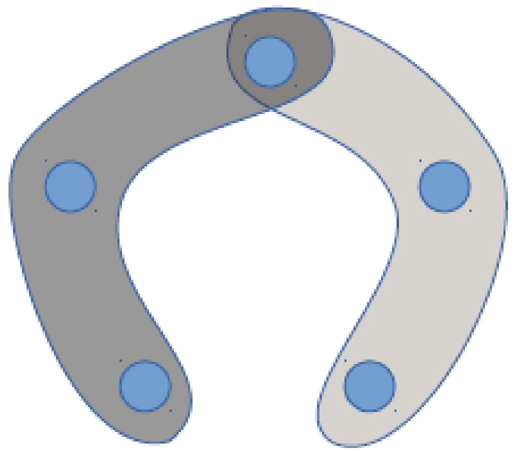
High-level functionality:

1. Client selects a free quorum
2. Locks all nodes of the quorum
3. Client releases all locks

Quorum Systems: Singleton & Majority



Singleton



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

Quorum Systems: Load and Work

	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

Quorum Systems: Load and Work

- An access strategy Z defines the probability $P_Z(Q)$ of accessing a quorum $Q \in S$ such that $\sum_{Q \in S} P_Z(Q) = 1$

Quorum Systems: Load and Work

- **Load** of access strategy Z on a node v_i $L_Z(v_i) = \sum_{Q \in S; v_i \in Q} P_Z(Q)$
- **Load** induced by Z on quorum system S $L_Z(S) = \max_{v_i \in S} L_Z(v_i)$
- **Load** of quorum system S $L(S) = \min_Z L_Z(S)$

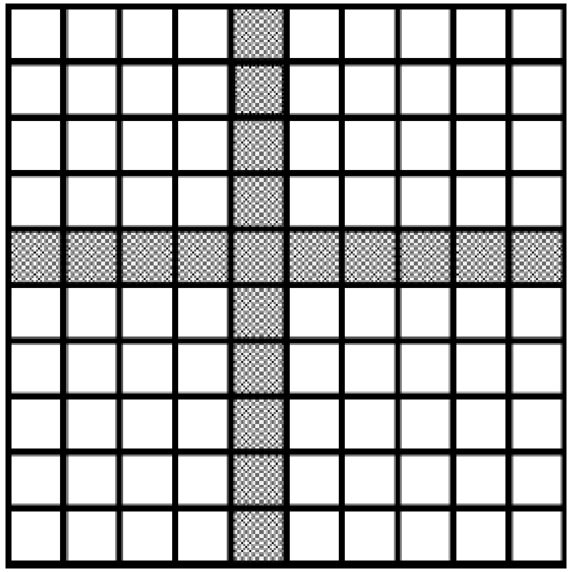
- **Work** of quorum Q $W(Q) = |Q|$
- **Work** induced by Z on quorum system S $W_Z(S) = \sum_{Q \in S} P_Z(Q) \cdot W(Q)$
- **Work** of quorum system S $W(S) = \min_Z W_Z(S)$

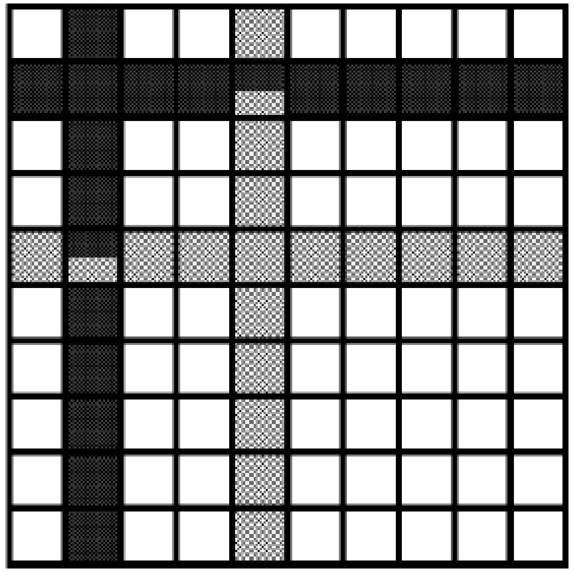
Grid quorum system – Basic Grid

- n nodes arranged in a square matrix
- Each quorum contains the full row i and the full column i

Problem:

- 2 quorums intersect in two nodes -> deadlock





Fault tolerance

- f-resilient
  - any f nodes can fail and at least one quorum still exists

B-grid quorum system

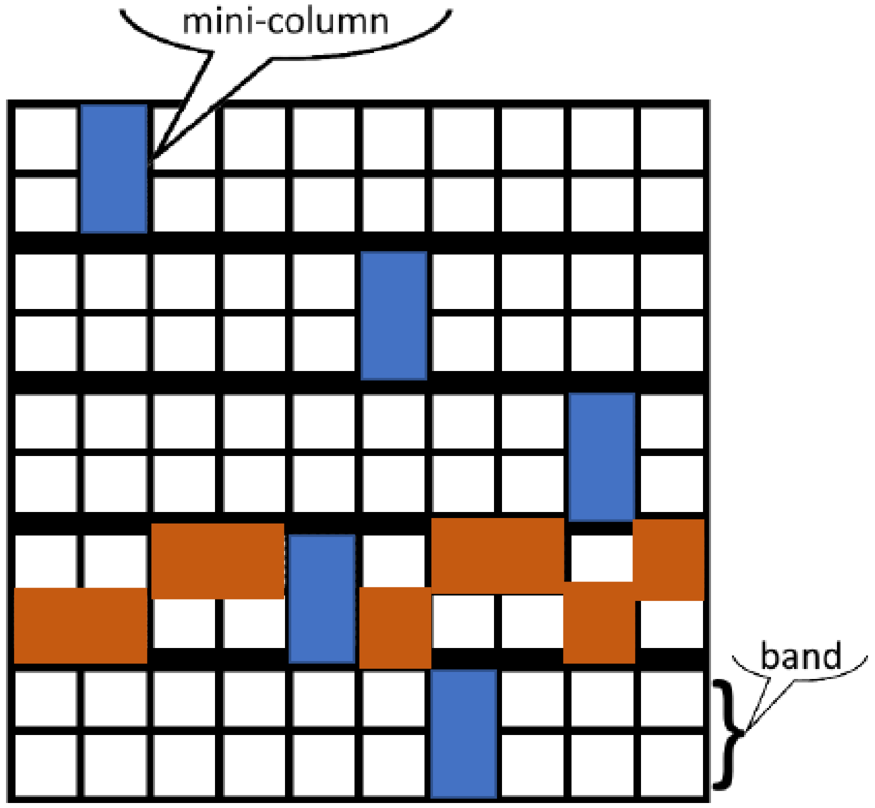
- Mini columns: one mini column in every band
- One band with at least one element per mini-column

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*

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
- Bad news:
    - achieving three is impossible (CAP theorem)

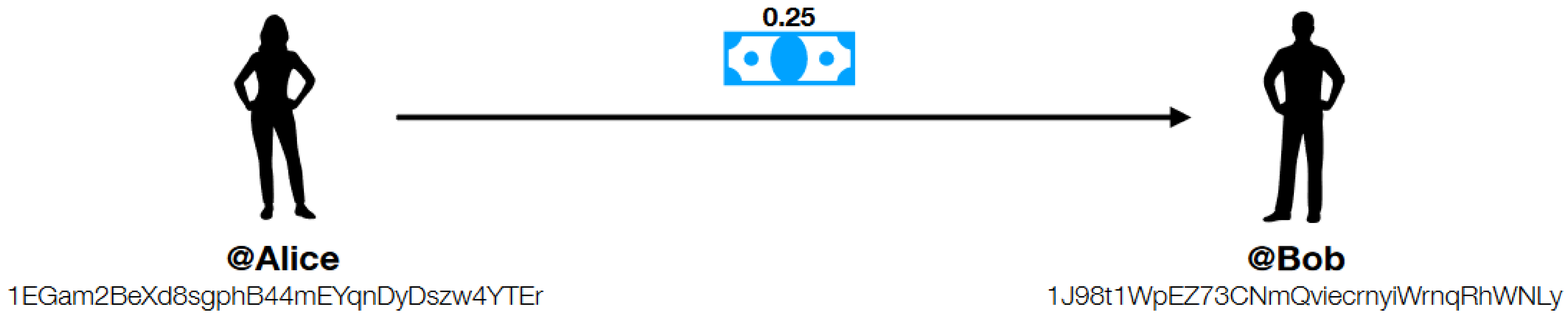
Bitcoin

- decentralized network consisting of nodes

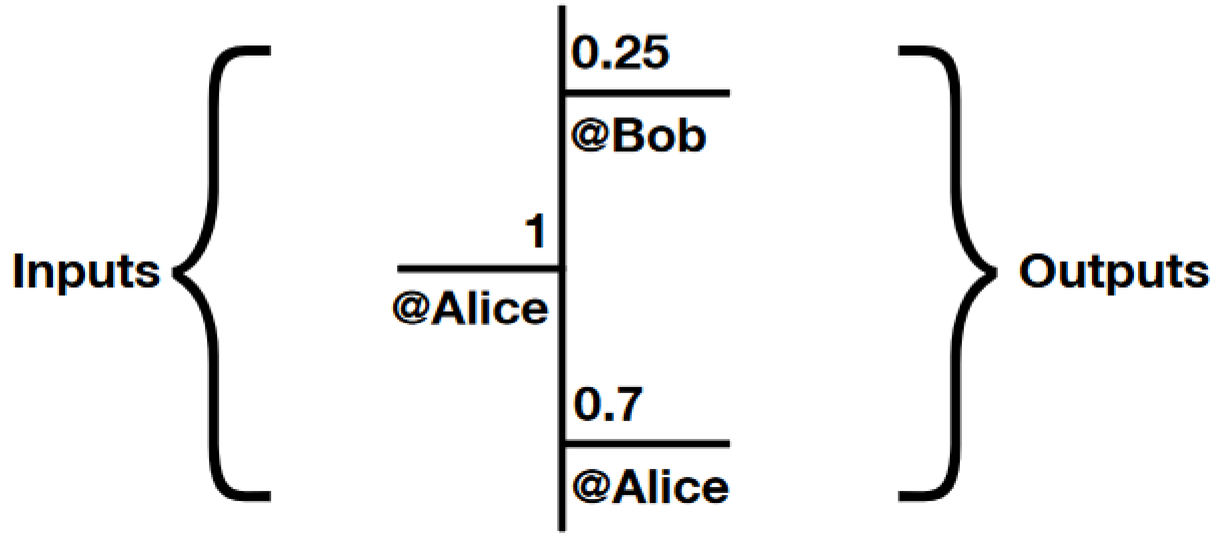
Bitcoin: Addresses

- unique identifier
- hash of public key
    - base 58 encoding
- multiple per person
    - pseudonymous

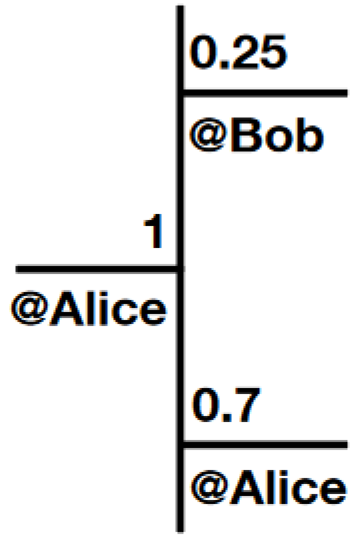
Bitcoin transaction


0.25
@Alice
1EGam2BeXd8sgphB44mEYqnDyDszw4YTEr
@Bob
1J98t1WpEZ73CNmQviecrnyiWrnqRhWNLy

Bitcoin transaction



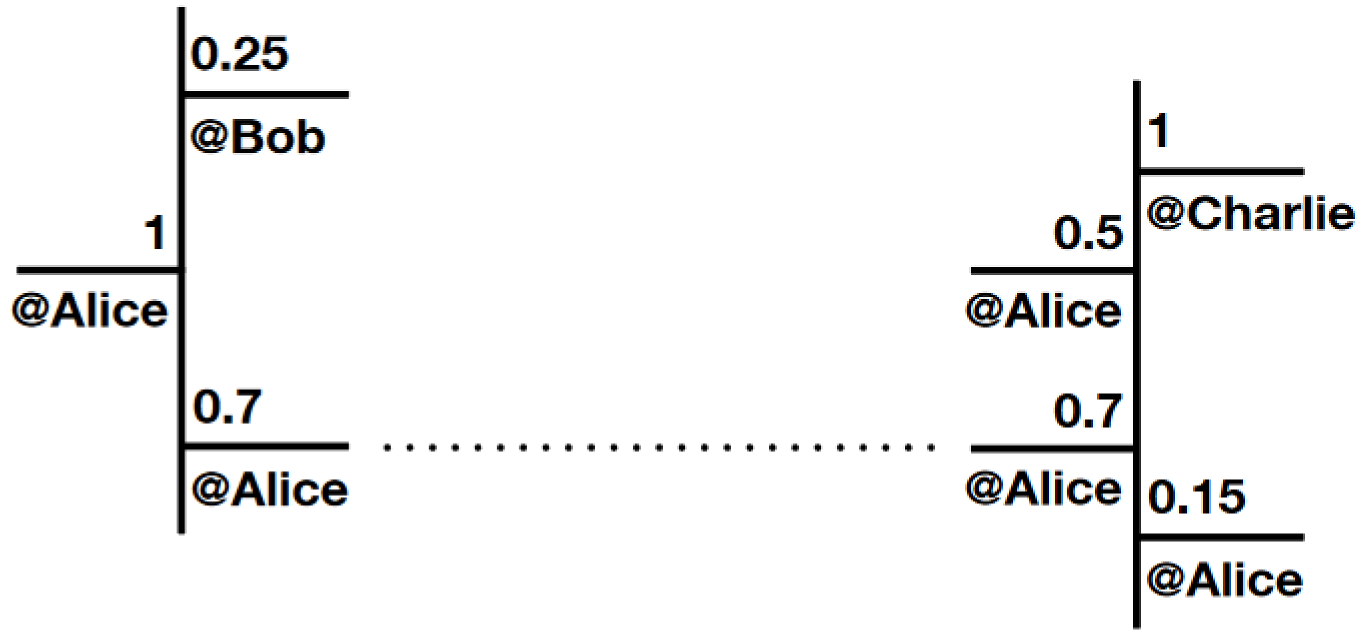
Bitcoin transaction



Transaction Fees 0.05

1 Satoshi per byte

Bitcoin transaction



Bitcoin: Local Transaction Acceptance

Sanity check for each input in transaction

- if referenced output is not in local UTXO or signature invalid, drop it and stop
- if sum of values of inputs < sum of values of new outputs, drop it and stop

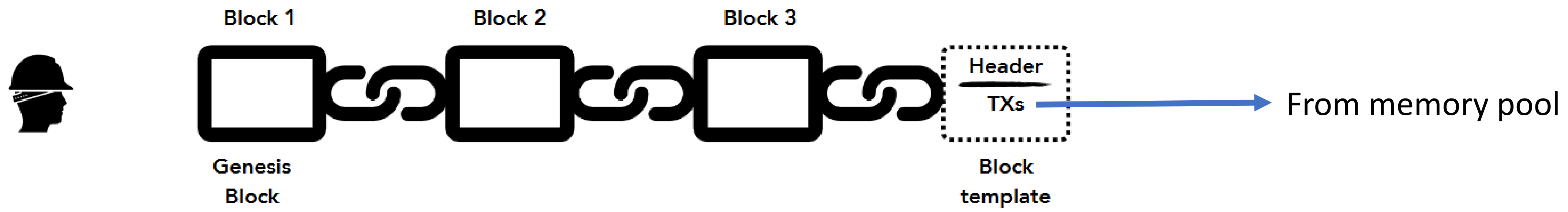
$\Rightarrow$ Remove referrenced output from local UTXO

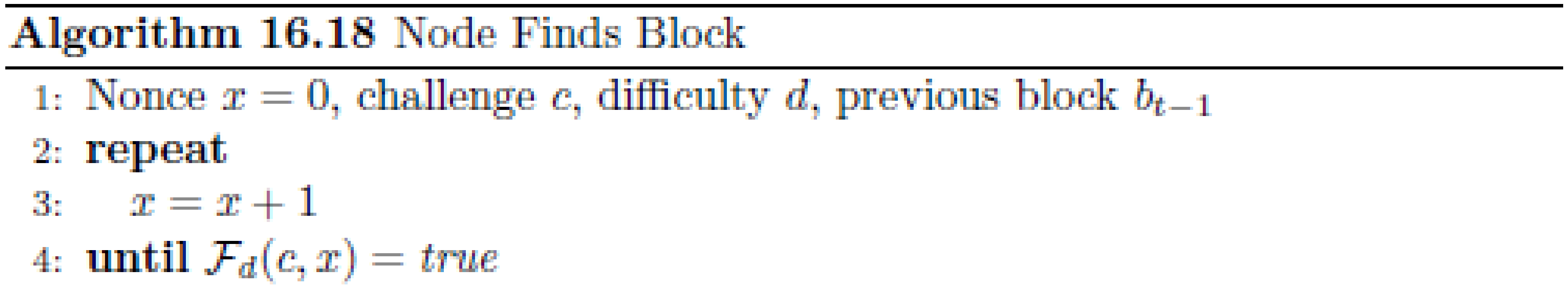
$\Rightarrow$ Add transaction to memory pool

$\Rightarrow$ Doublespending = multiple trx spending same coin

Bitcoin: Conflict Resolution Mechanism

- Transactions are either confirmed or unconfirmed



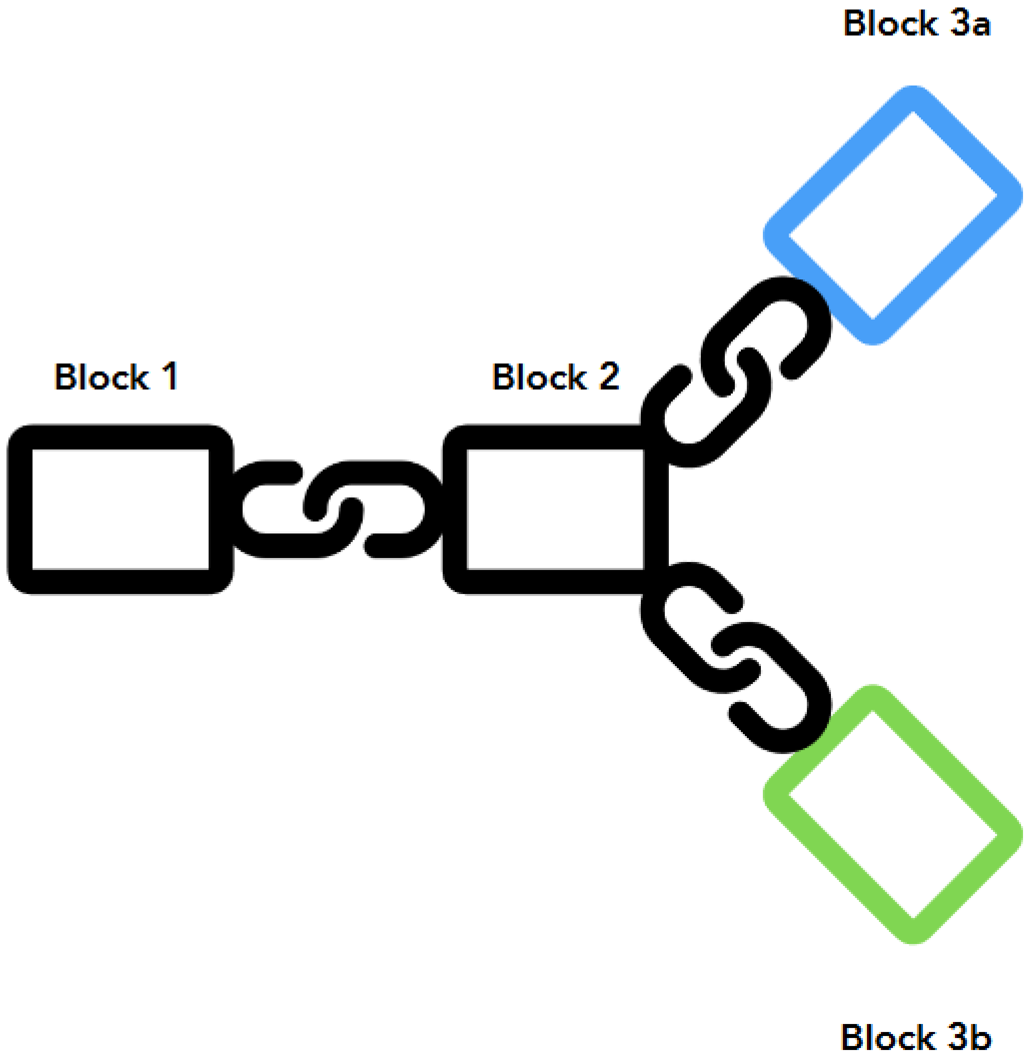


Algorithm 16.18 Node Finds Block

```
1: Nonce $x = 0$, challenge $c$, difficulty $d$, previous block $b_{t-1}$
2: repeat
3:     $x = x + 1$
4: until $\mathcal{F}_d(c, x) = true$
```

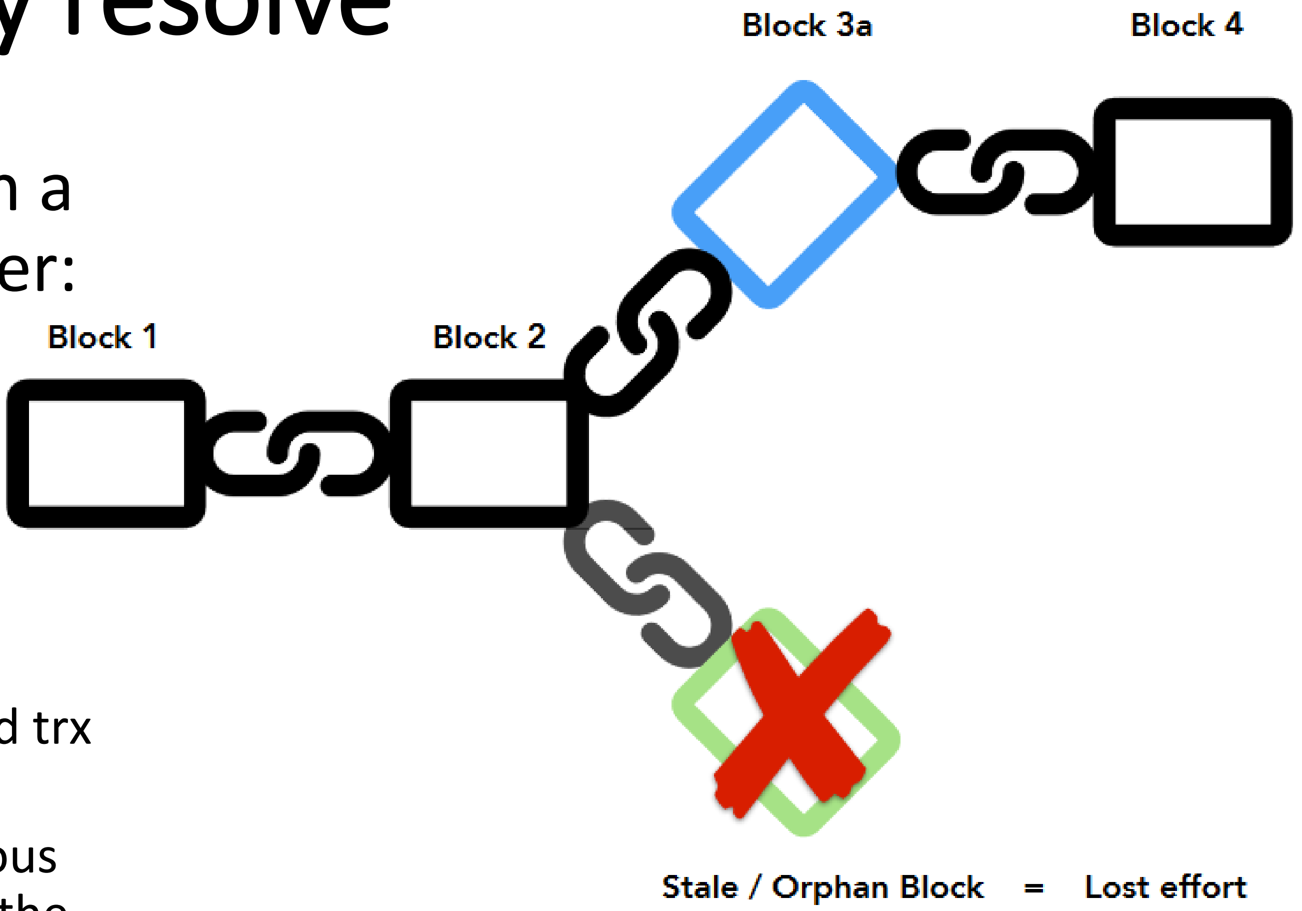
Bitcoin: Fork

- Blocks get found at same time
- Network state temporarily diverges



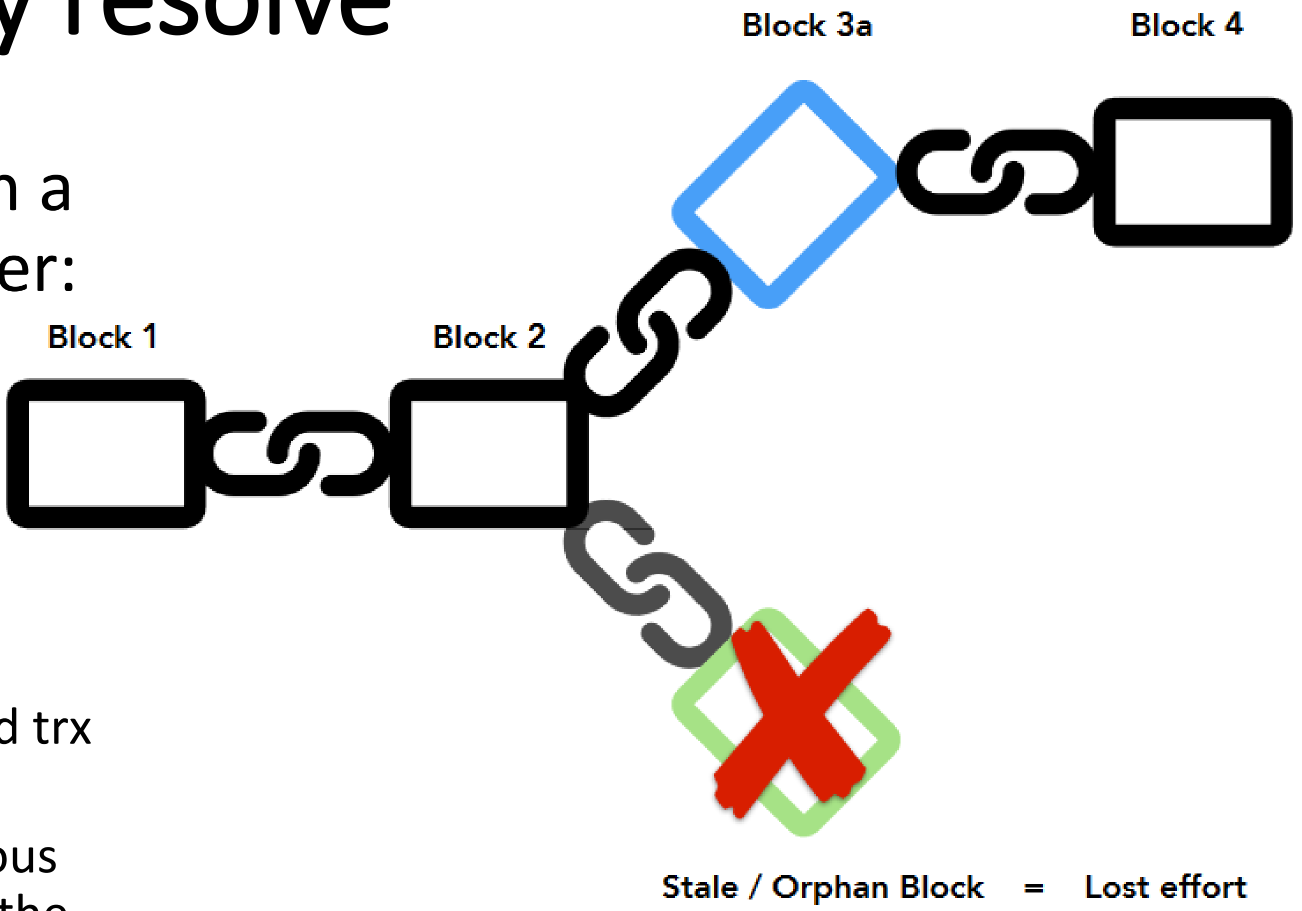
Bitcoin: Fork eventually resolve

- Node hears about a new block on a different branch that is now longer:
  - Discard Block 3b
  - Move to longest chain
  - Clean-up memory pool:
    - remove trx that are confirmed in the current path
    - remove trxt in conflict with confirmed trx ("Doublespending")
    - add trx that were confirmed in previous path, but are no longer confirmed in the current path



Bitcoin: Fork eventually resolve

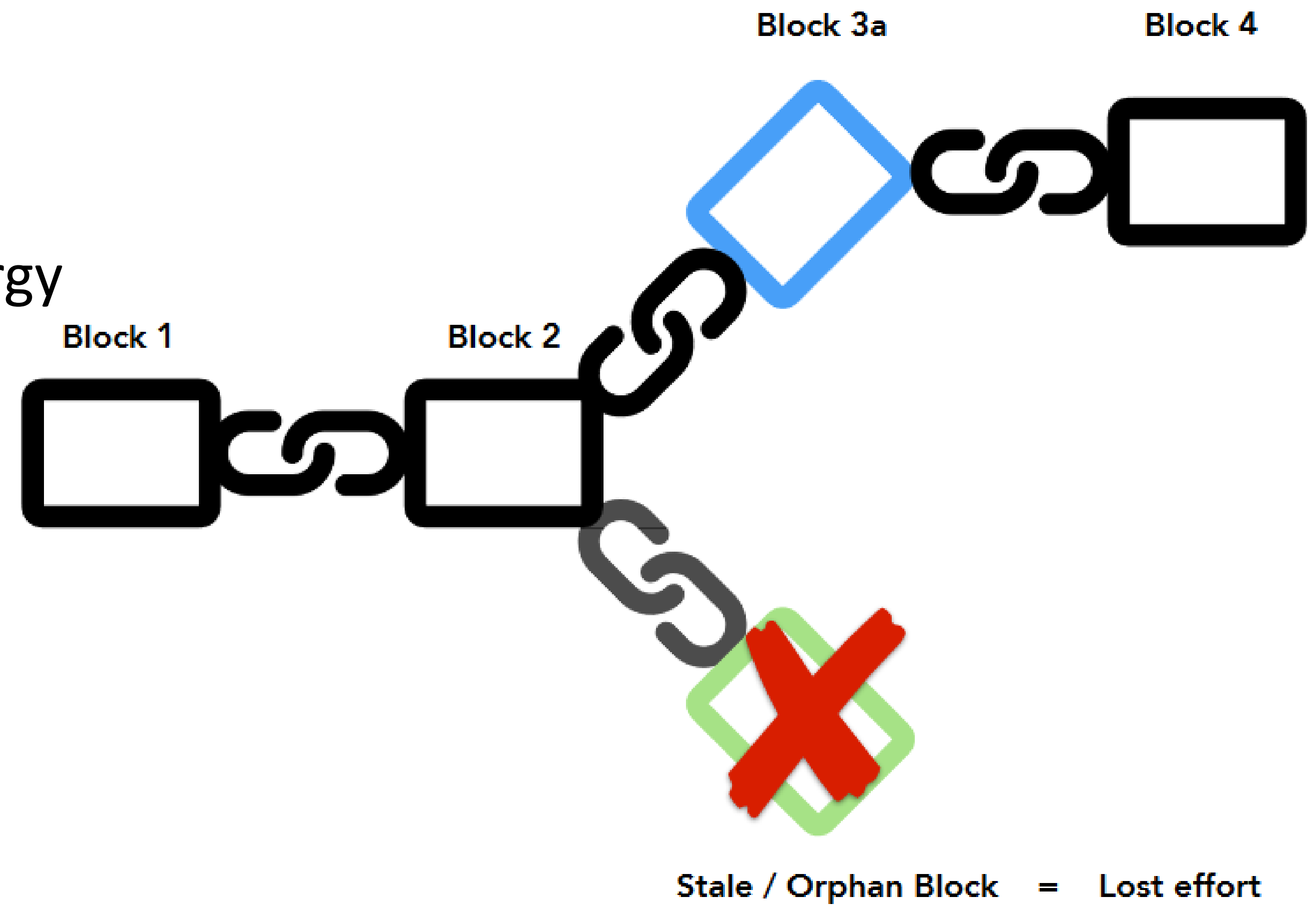
- Node hears about a new block on a different branch that is now longer:
  - Discard Block 3b
  - Move to longest chain
  - Clean-up memory pool:
    - remove trx that are confirmed in the current path
    - remove trxt in conflict with confirmed trx ("Doublespending")
    - add trx that were confirmed in previous path, but are no longer confirmed in the current path



Bitcoin: Incentives

- Why would you mine?

  (= spend lots of money on high energy consuming computations)
- Money!
  - Block rewards
  - Transaction fees



Smart Contracts

- Contract between two or more parties, encoded in such a way that correct execution is guaranteed by blockchain
- Multisig outputs:
  - specifies set of m public keys and requires k-of-m (with $k <= m$) valid signatures from distinct matching public keys from that set to be valid

Smart Contracts: Timelock

- transaction will only get added to memory pool after some time has expired

Smart Contracts: Micropayment channel

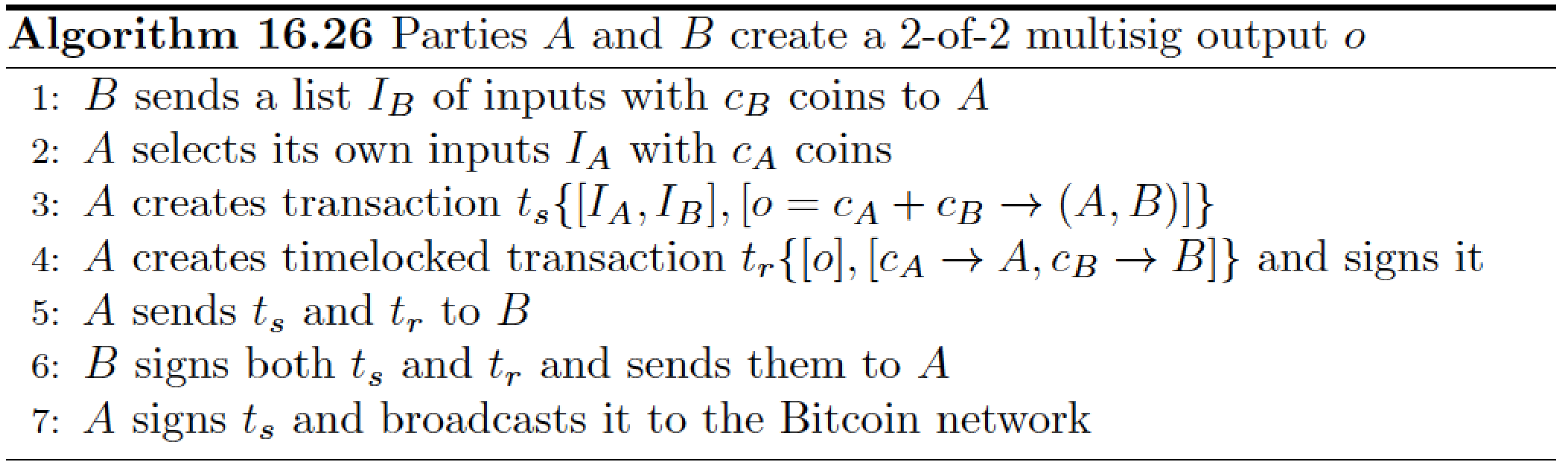
Idea:

- Two parties want to do multiple small transactions, but avoid fees.

Solution:

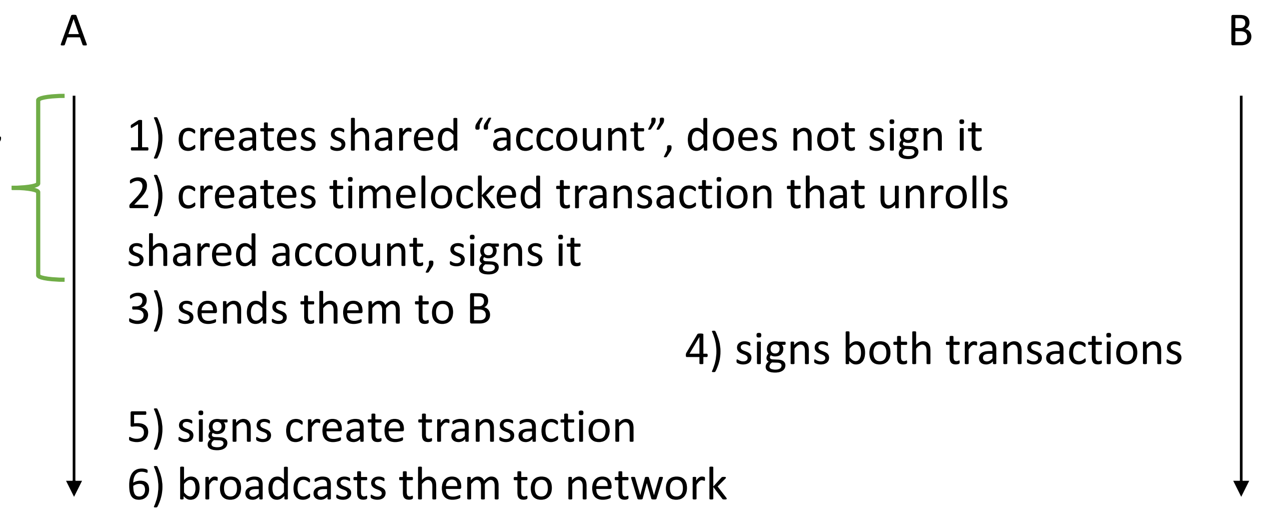
- they only submit first and last transaction to blockchain and privately do everything in between.

Micropayment channel – setup transaction



Algorithm 16.26 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

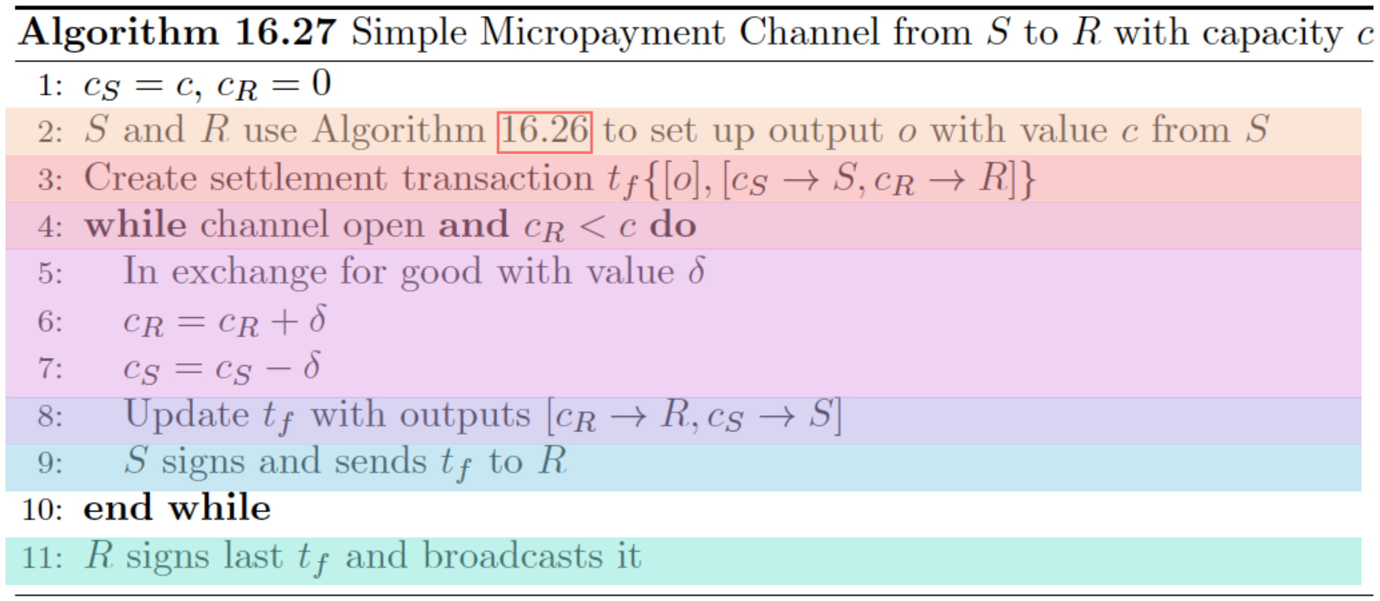
1) creates shared “account”, does not sign it
2) creates timelocked transaction that unrolls shared account, signs it
3) sends them to B

4) signs both transactions

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

5) signs create transaction
6) broadcasts them to network

Micropayment channel



Algorithm 16.27 Simple Micropayment Channel from S to R with capacity c

```
$c_S = c$, $c_R = 0$
$S$ and $R$ use Algorithm 16.26 to set up output $o$ with value $c$ from $S$
Create settlement transaction $t_f\{[o], [c_S \to S, c_R \to R]\}$
**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 $[c_R \to R, c_S \to S]$
    $S$ 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
while buyer 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

- What is the maximum number of nodes that can fail so that the majority quorum system still works?
  - n/2-1
- Are effects of transactions on a node deterministic?
  - yes
- Can a micropayment channel be used bidirectionally?
  - No, because then both parties would hold fully signed transactions and then submit the ones that are best for them

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.

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 Δ of blocks ago, then we can say that all nodes are consistent with high probability.

a) Can we say that the Δ-delayed state is strongly consistent for sufficiently large Δ?

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?