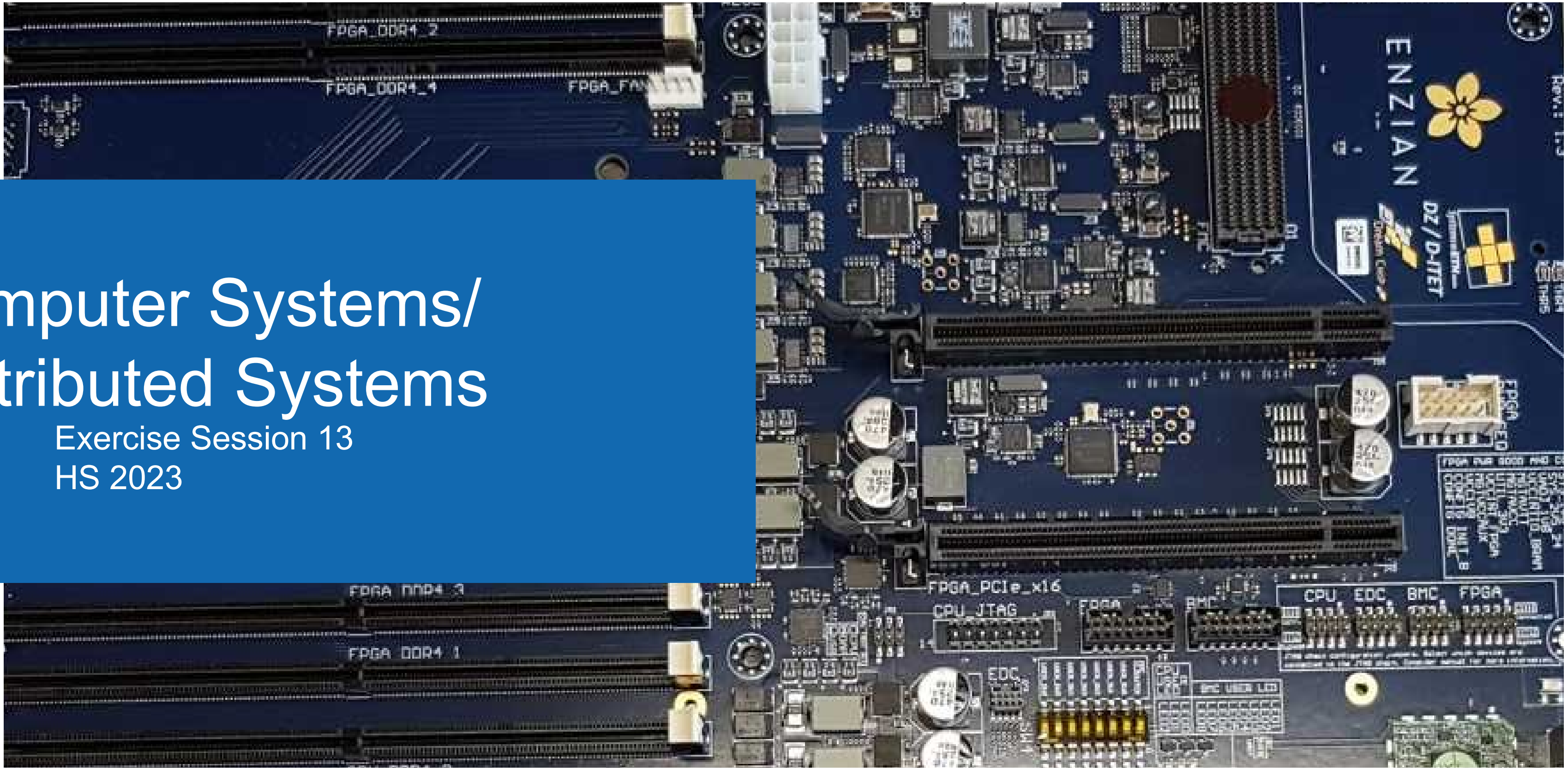
Slides last updated:
19.12.2023



Computer Systems/ Distributed Systems

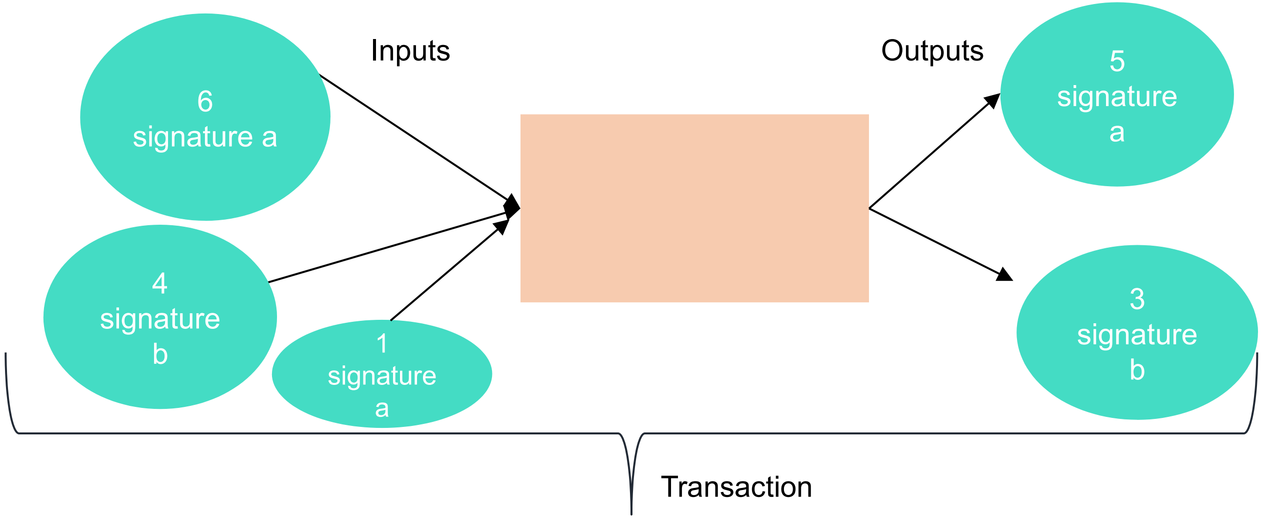
Exercise Session 13
HS 2023

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

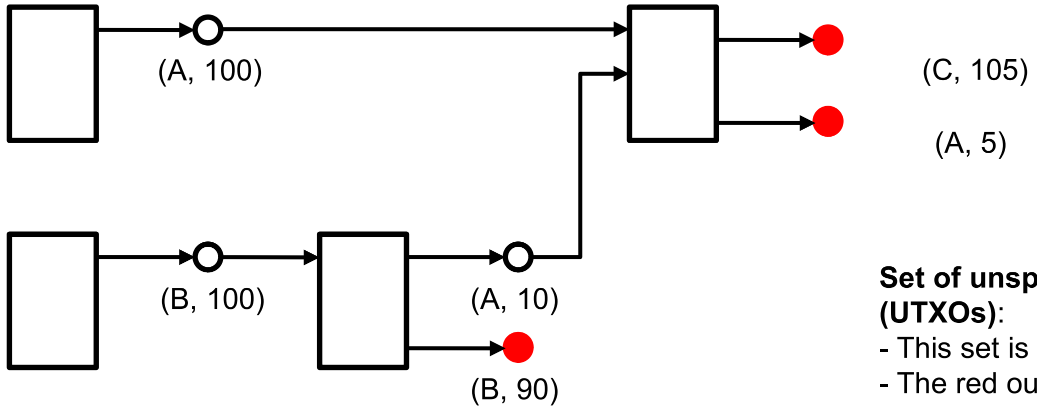
- Decentralized network consisting of nodes
- Users generate private/public key pair
  - Address is generated from public key
  - It is difficult to get users "real" identity from public key



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

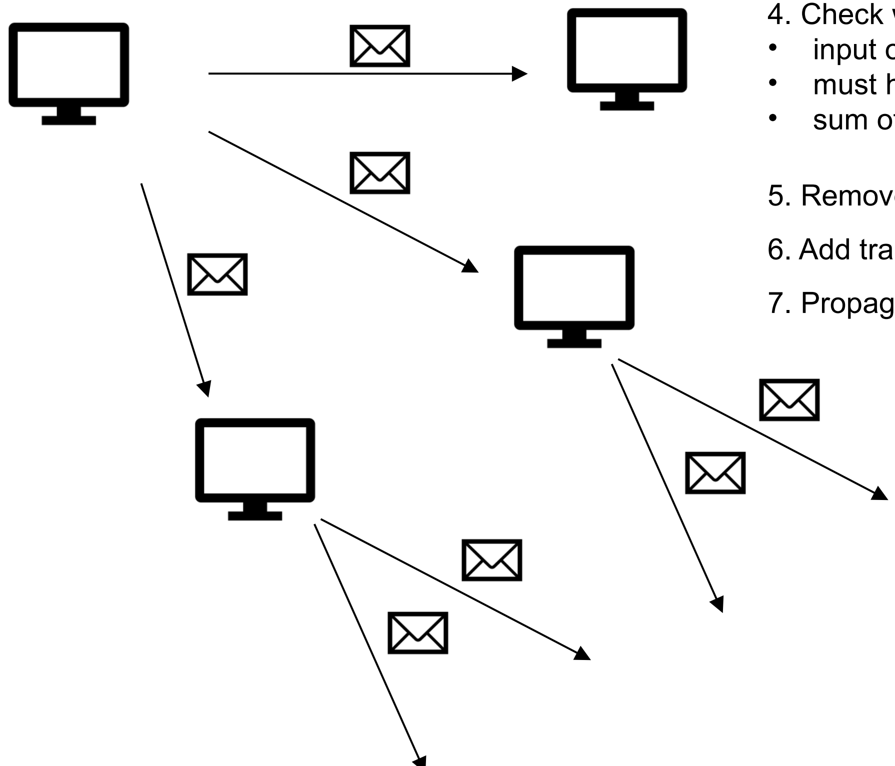


Set of unspent transaction outputs (UTXOs):
- This set is the shared state of Bitcoin
- The red outputs

Transaction Broadcast

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

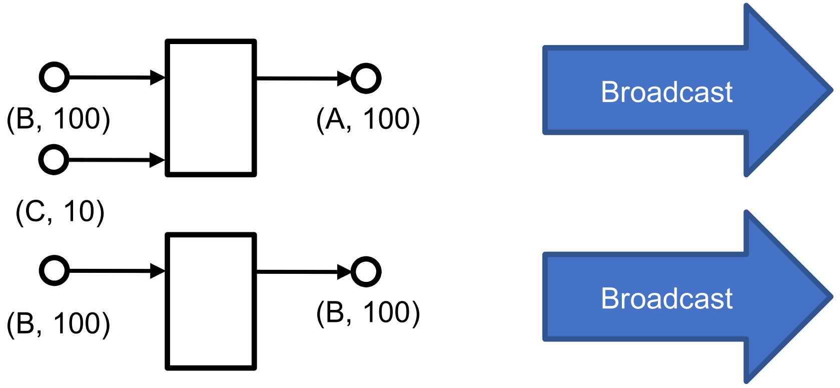
6. Add transaction to local history

7. Propagate transaction further

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

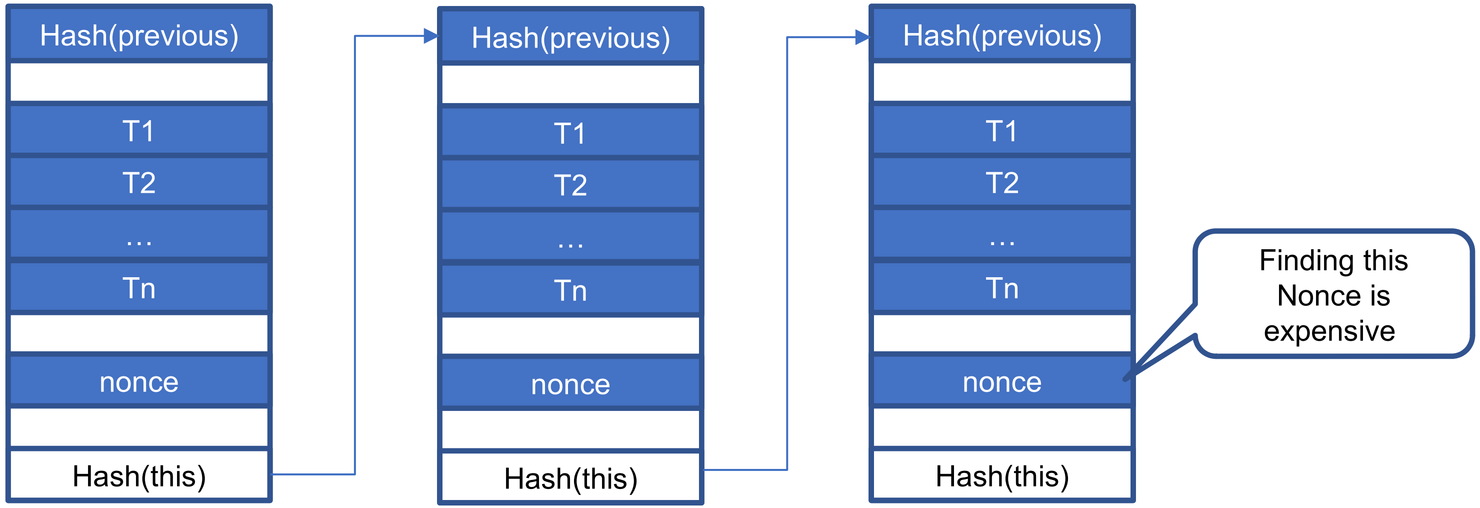
Proof-of-Work

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

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

Block

- Data structure holding transactions reference to previous blocks and a nonce.
  - Header also contains more fields, such as a timestamp, the difficulty, network version, etc.
- Miner creates blocks with transactions from its memory pool



Mining

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

Bitcoin:

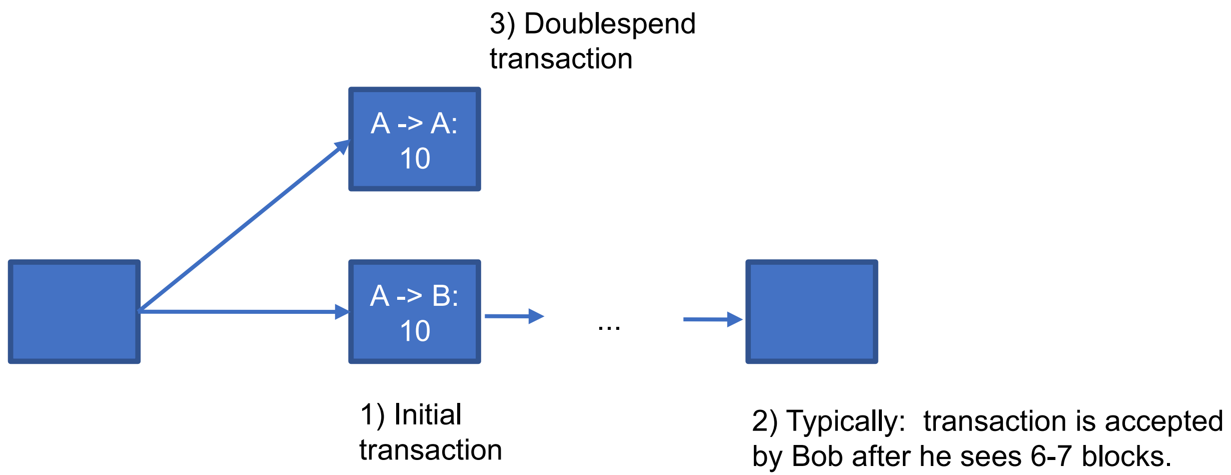
- Reward started at 50B and it is being halved every 210,000 blocks or 4 years in expectation
- This bounds the total number of Bitcoins to 21 million
- What will happen after that?

- Fee is the positive difference of input-output
- Miner include transactions which have a high fee.

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

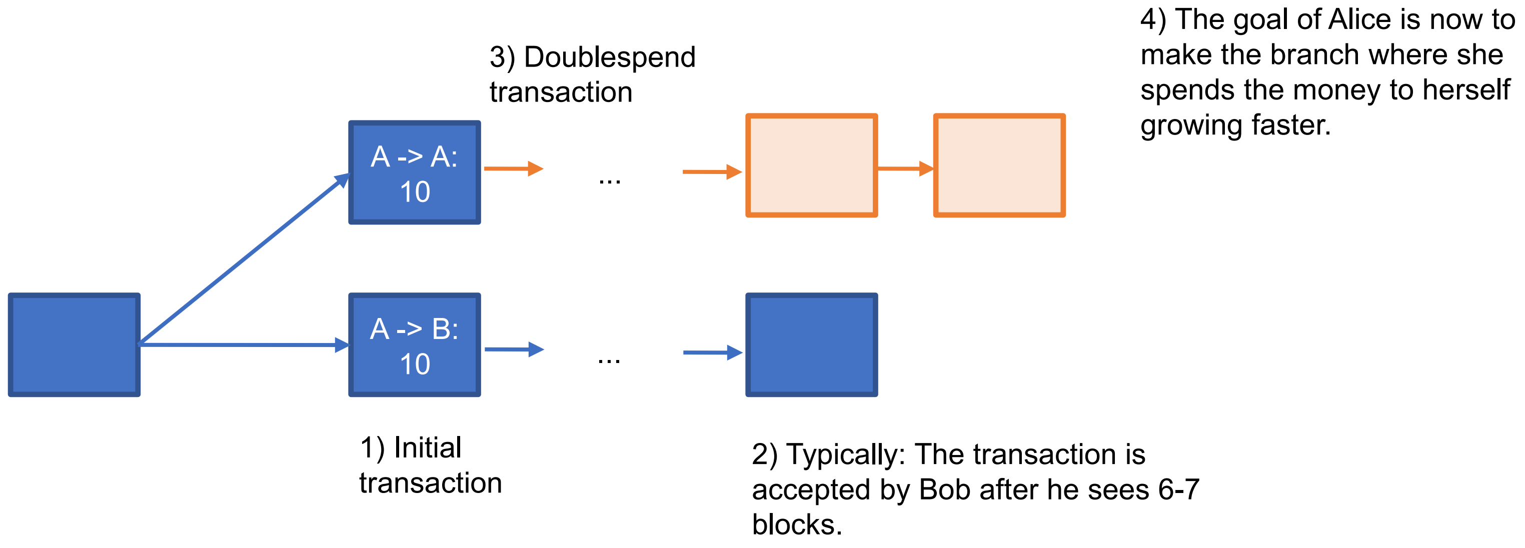
How does this prevent double spending?

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


3) Doublespend transaction
A -> A: 10
A -> B: 10
...
1) Initial transaction
2) Typically: transaction is accepted by Bob after he sees 6-7 blocks.

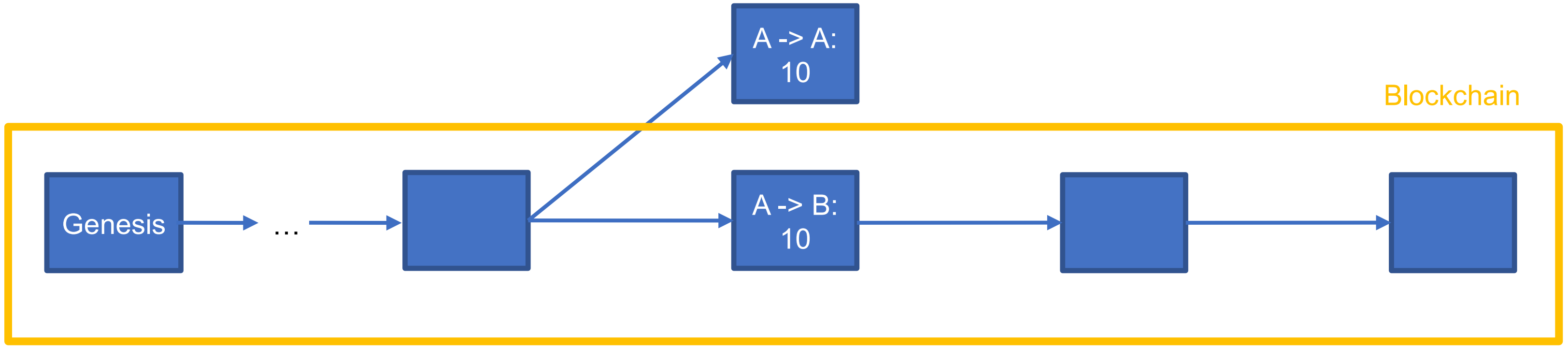
How does this prevent double spending?

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


3) Doublespend transaction
4) The goal of Alice is now to make the branch where she spends the money to herself growing faster.
A -> A: 10
...
A -> B: 10
...
1) Initial transaction
2) Typically: The transaction is accepted by Bob after he sees 6-7 blocks.

Blockchain

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



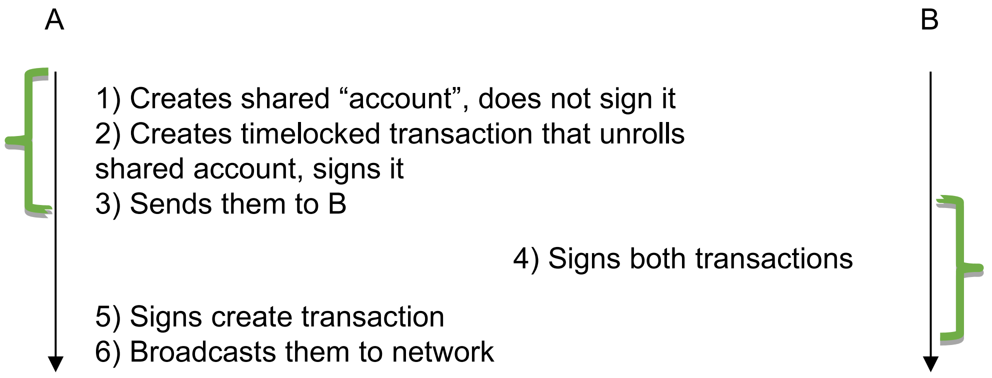
Note: To ensure that you'll get the money you should wait 5-10 further blocks

Smart Contracts

- Contract between two or more parties, encoded in such a way that correct execution is guaranteed by blockchain
  - Timelock transaction: Tx 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 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
```



A can't do anything with this, since no transaction has all required signatures

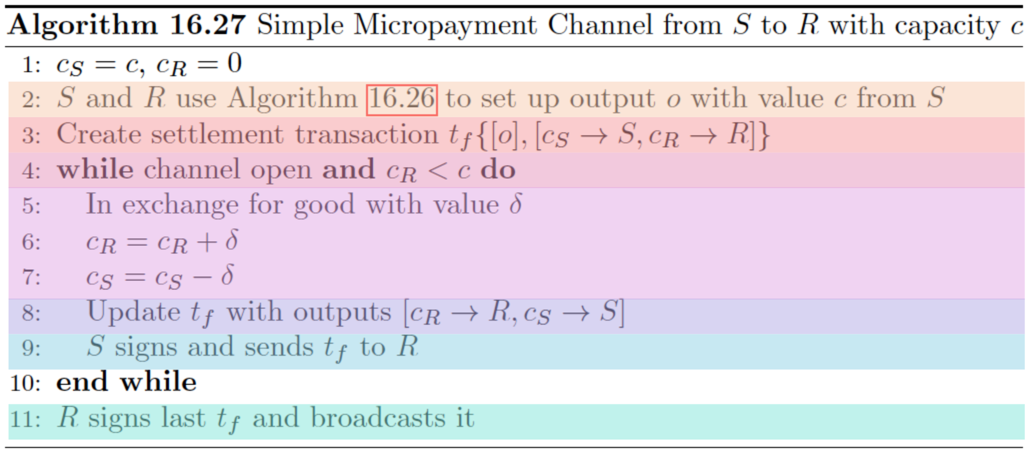
1) Creates shared “account”, does not sign it
2) Creates timelocked transaction that unrolls shared account, signs it
3) Sends them to B

4) Signs both transactions

5) Signs create transaction
6) Broadcasts them to network

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

Micropayment Channel



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

```
1: $c_S = c$, $c_R = 0$
2: $S$ and $R$ use Algorithm 16.26 to set up output $o$ with value $c$ from $S$
3: Create settlement transaction $t_f\{[o], [c_S \rightarrow S, c_R \rightarrow R]\}$
4: while channel open and $c_R < c$ do
5:     In exchange for good with value $\delta$
6:     $c_R = c_R + \delta$
7:     $c_S = c_S - \delta$
8:     Update $t_f$ with outputs $[c_R \rightarrow R, c_S \rightarrow S]$
9:     $S$ signs and sends $t_f$ to $R$
10: end while
11: $R$ signs last $t_f$ and broadcasts it
```

Set up shared account and unrolling

Create settlement transaction

While buyer 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

1.1 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?

1.1 Delayed Bitcoin

a) It is true that naturally occurring forks of length l decrease exponentially with l, however this covers naturally occuring blockchain forks only. As there is no information how much calculation power exists in total, it is always possible a large blockchain fork exists. This may be the result of a network partition or an attacker secretly running a large mining operation.

This is a general problem with all "open-membership" consensus systems, where the number of existing consensus nodes is unknown and new nodes may join at any time. As it is always possible a much larger unknown part of the network exists, it is impossible to have strong consistency.

In the Bitcoin world an attack where an attacker is secretly mining a second blockchain to later revert many blocks is called a 51% attack, because it was thought necessary to have a majority of the mining power to do so. However later research showed that by using other weaknesses in Bitcoin it is possible to do such attacks already with about a third of the mining power.

b) The delay in this case prevents coins from completely vanishing in the case of a fork. Newly mined coins only exist in the fork containing the block that created them. In case of a blockchain fork the coins would disappear and transactions spending them would become invalid as well. It would therefore be possible to taint any number of transactions that are valid in one fork and not valid in another. Waiting for maturation ensures that it is very improbable that the coins will later disappear accidentially.

Note that this is however only a protection against someone accidentially sending you money that disappears with a discontinued fork. The same thing can still happen, if someone with evil intent double spends the same coins on the other side of the fork. You will not be able to replay a transaction of a discontinued fork on the new active chain if the old owner spent them in a different transaction in the meantime. To prevent theft by such an attacker you need to wait enough time to regard the chance of forks continuing to exist to be small enough. A common value used is about one hour after a transaction entered a block (~6 blocks).

Quiz

2.2 Double Spending

Figure 1 represents the topology of a small Bitcoin network. Further assume that the two transactions T and T' of a doublespend are released simultaneously at the two nodes in the network and that forwarding is synchronous, i.e., after t rounds a transaction was forwarded t hops.

a) Once the transactions have fully propagated, which nodes know about which transactions?

b) Assuming that all nodes have the same computational power, i.e., same chances of finding a block, what is the probability that T will be confirmed?

c) Assuming the rightmost node, which sees T' first, has 20% of the computational power and all nodes have equal parts of the remaining 80%, what is the probability that T' will be confirmed?

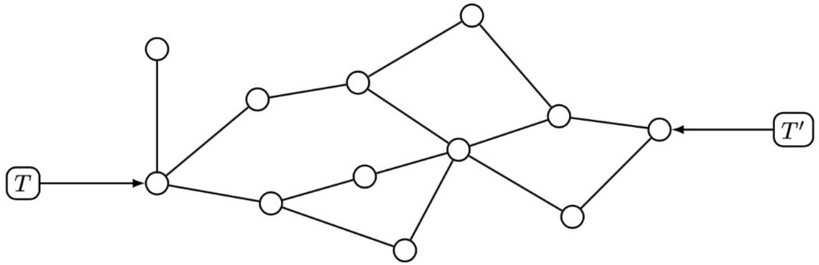


Figure 1: Random Bitcoin network

Quiz

Systems@ETH Zürich

Distributed Computing

2.2 Double Spending

Figure 1 represents the topology of a small Bitcoin network. Further assume that the two transactions T and T' of a doublespend are released simultaneously at the two nodes in the network and that forwarding is synchronous, i.e., after t rounds a transaction was forwarded t hops.

a) Once the transactions have fully propagated, which nodes know about which transactions?

b) Assuming that all nodes have the same computational power, i.e., same chances of finding a block, what is the probability that T will be confirmed?

$$\frac{7}{12}$$

c) Assuming the rightmost node, which sees T' first, has 20% of the computational power and all nodes have equal parts of the remaining 80%, what is the probability that T' will be confirmed?

$$20+4*\frac{80}{11} = 49\%$$

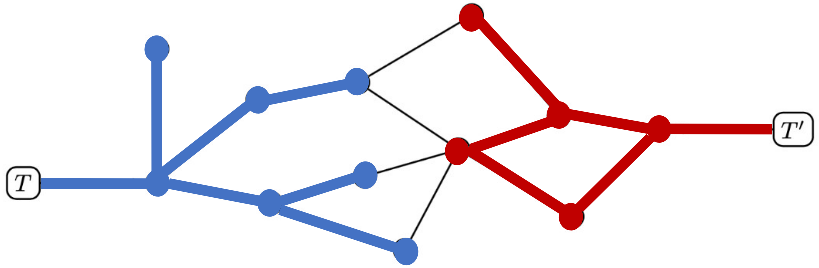
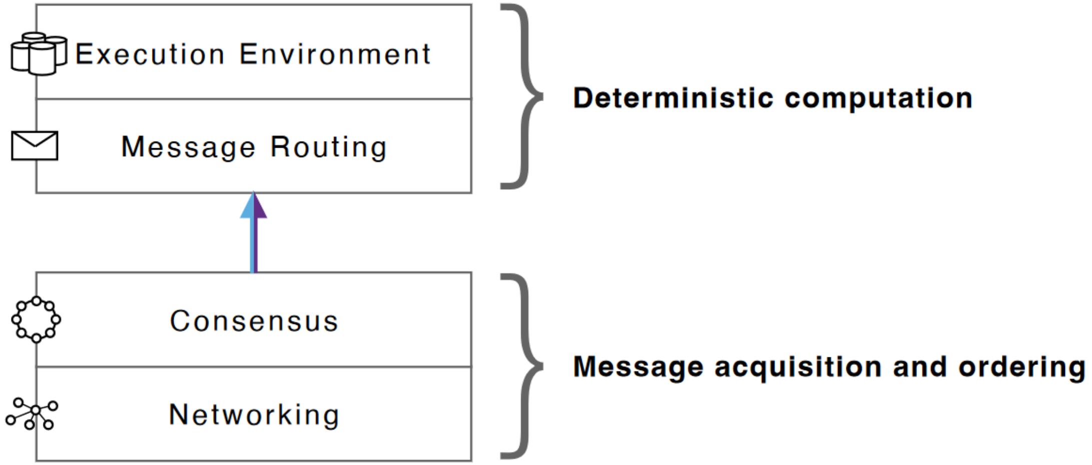


Figure 1: Random Bitcoin network

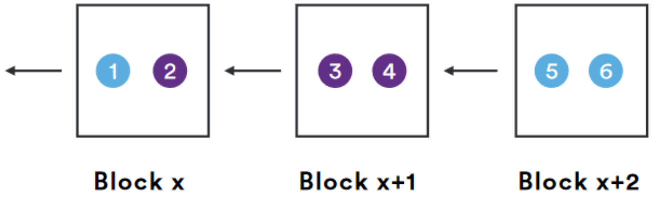
Internet Computer

- The Layers of the Internet Computer Protocol



Internet Computer

Messages are placed in **blocks**. We reach agreement using a blockchain.



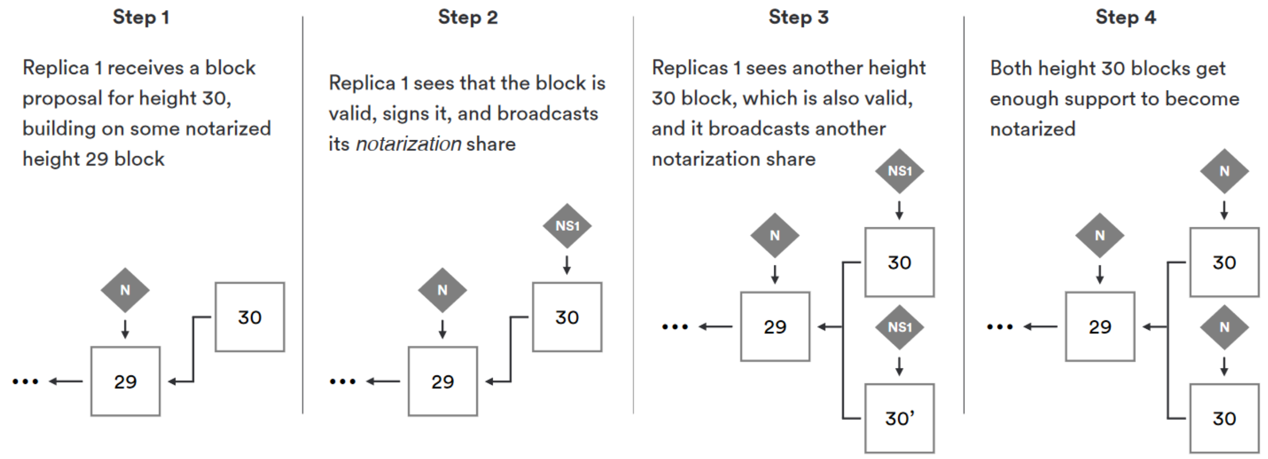
We use *n* = 4, *f* = 1 in examples

The following properties must hold even if up to *f* < n/3 nodes misbehave

- **Agreement**: For any i, If two (honest) nodes think that the *i*-th block is agreed upon, they must have the same block
- **Termination**: For any i, at some point every (honest) node will think that the *i*-th block is agreed upon
- **Validity**: all agreed upon blocks are valid

Internet Computer - Notarization

- The notarization process ensures that a valid block proposal is published for every block height
- Replicas may notary-sign multiple blocks to ensure that at least one block becomes fully notarized
- Multiple notarized blocks may exist at the same block height



Step 1

Replica 1 receives a block proposal for height 30, building on some notarized height 29 block

Step 2

Replica 1 sees that the block is valid, signs it, and broadcasts its *notarization* share

Step 3

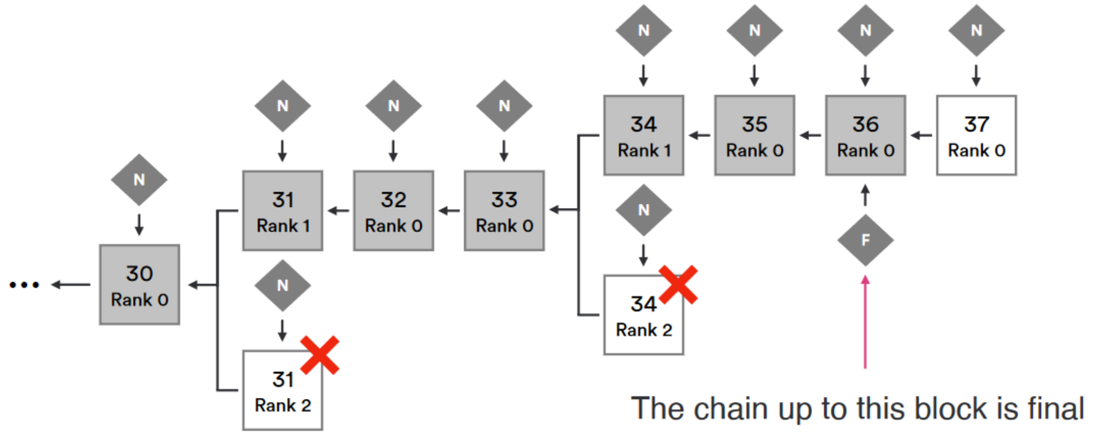
Replicas 1 sees another height 30 block, which is also valid, and it broadcasts another notarization share

Step 4

Both height 30 blocks get enough support to become notarized

Internet Computer - Finalization

- Replicas create finalization shares if they did not sign any other block at that height
- Finalization on block b at height h = Proof that no other block is notarized at height h



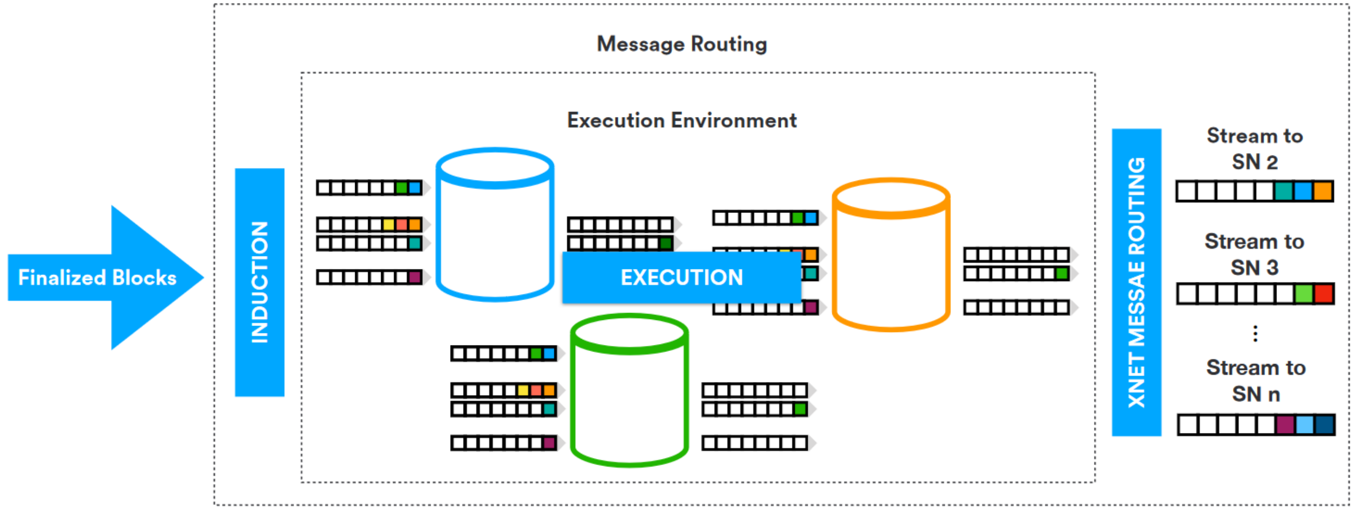
Internet Computer

- Agreement
  - If block b at height h is finalized, then there is no notarized block b’ ≠ b at height h.

- Termination
  - For every block height h, at least one block is notarized.
  - If communication is synchronous for 3 time units at the beginning of the execution for block height h and the rank-0 block maker is honest, then its block will be finalized.

Internet Computer

- High-Level Intuition of a Deterministic Execution Cycle

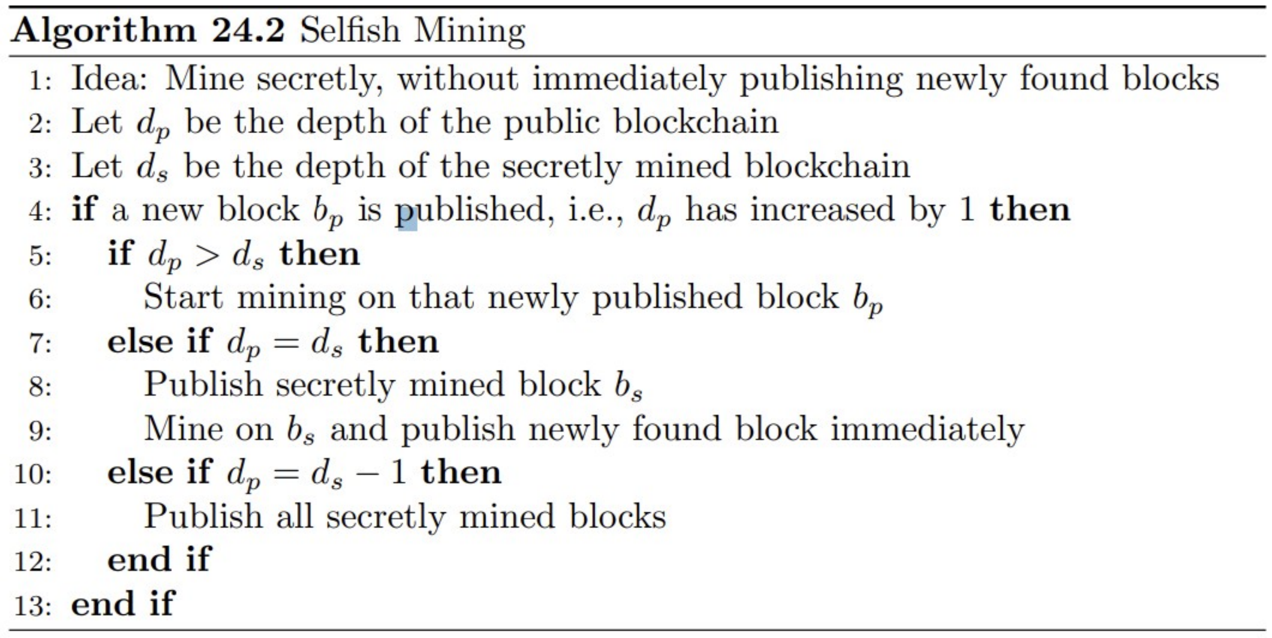


Take messages from finalized blocks and put them into input/ingress queues of the respective canisters.

Take messages from input/ingress queues, execute them and put produced messages in output queues

Take messages from output queues, route them in cross subnet streams (respecting ordering)

Selfish Mining



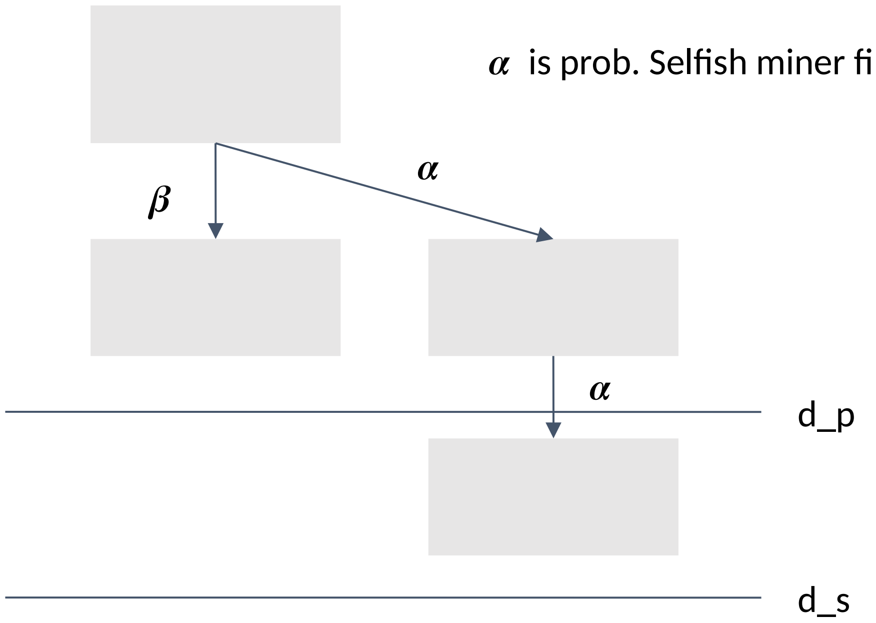
Algorithm 24.2 Selfish Mining

```
1: Idea: Mine secretly, without immediately publishing newly found blocks
2: Let $d_p$ be the depth of the public blockchain
3: Let $d_s$ be the depth of the secretly mined blockchain
4: if a new block $b_p$ is published, i.e., $d_p$ has increased by 1 then
5:     if $d_p > d_s$ then
6:         Start mining on that newly published block $b_p$
7:     else if $d_p = d_s$ then
8:         Publish secretly mined block $b_s$
9:         Mine on $b_s$ and publish newly found block immediately
10:    else if $d_p = d_s - 1$ then
11:        Publish all secretly mined blocks
12:    end if
13: end if
```

Selfish Mining

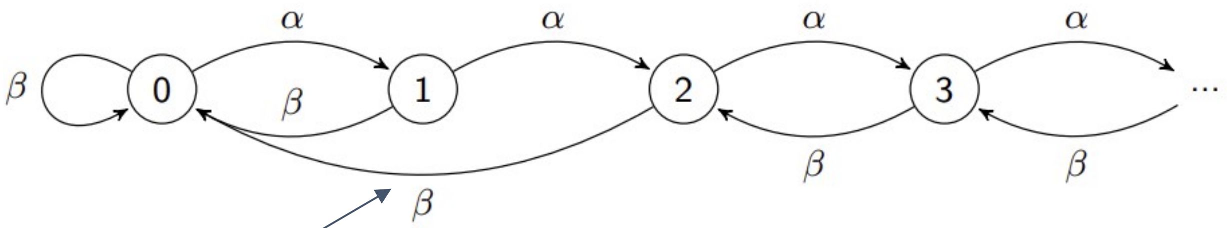
β is prob. others find block

α is prob. Selfish miner finds block



- Selfish miner does not release its block immediately, but keeps secret and works on "grandchildren" (secret)
- **Advantage**: selfish miner can work on next-next block, while others still work on next block.
- **Disadvantage** : Work on blocks (and rewards) potentially rendered useless when public chain gets longer.

Selfish Mining



Selfish minor is two block ahead and new public block -> release all secret blocks

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

Ratio of mining power

Share of reachable nodes if selfish miner publishes

Ethereum smart contracts

- Smart contract creation: A transaction with recipient with address 0 deploys a new smart contract
- Smart Contract Execution Transaction: A transaction with a smart contract address in its recipient field and code to execute a function of that contract in its data field
- Gas: The unit of an atomic computation, i.e ADDing two numbers costs 3 Gas