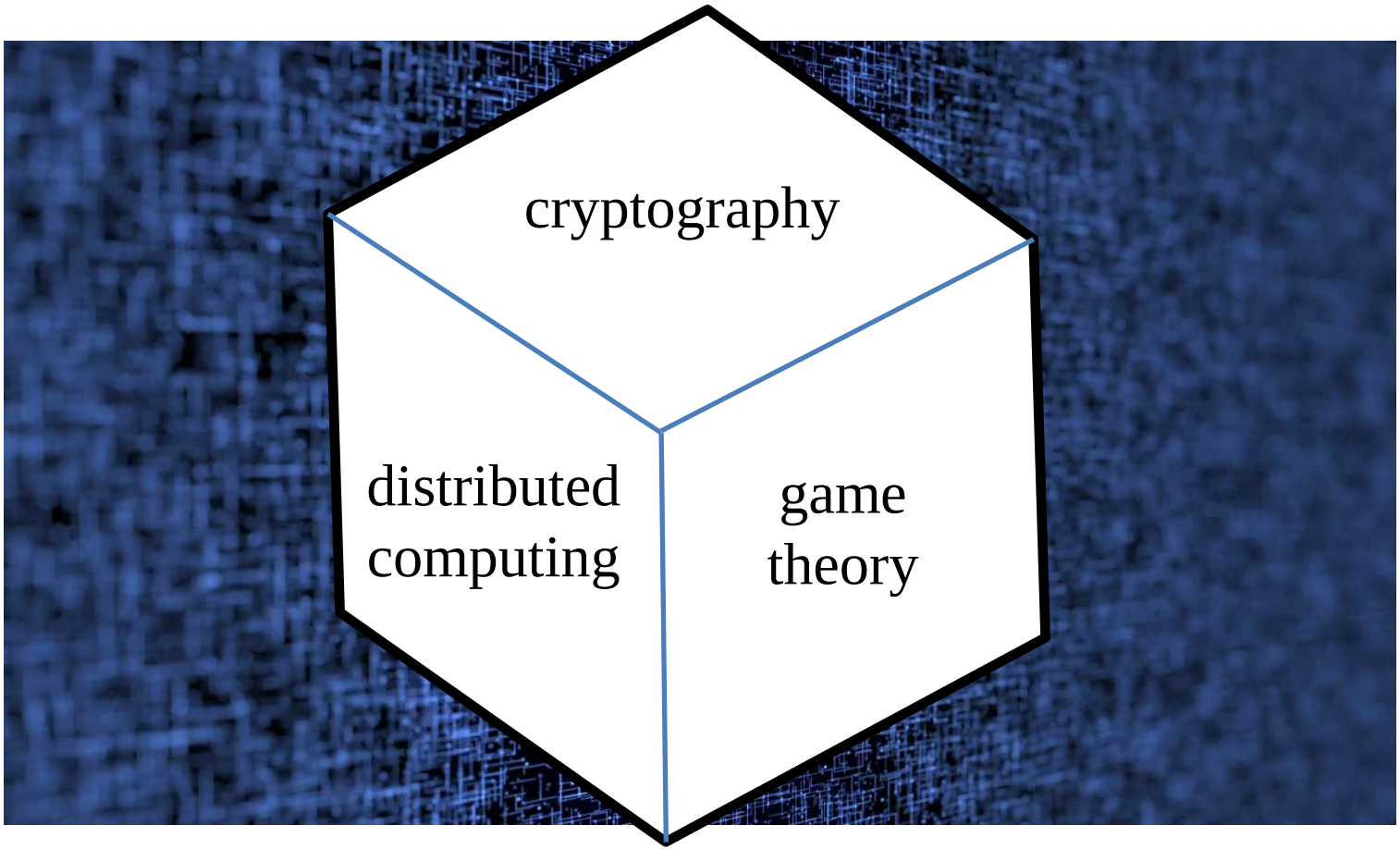
Consensus & fault tolerance: distributed and strategic aspects of the Blockchain technology

Luciano Gualà

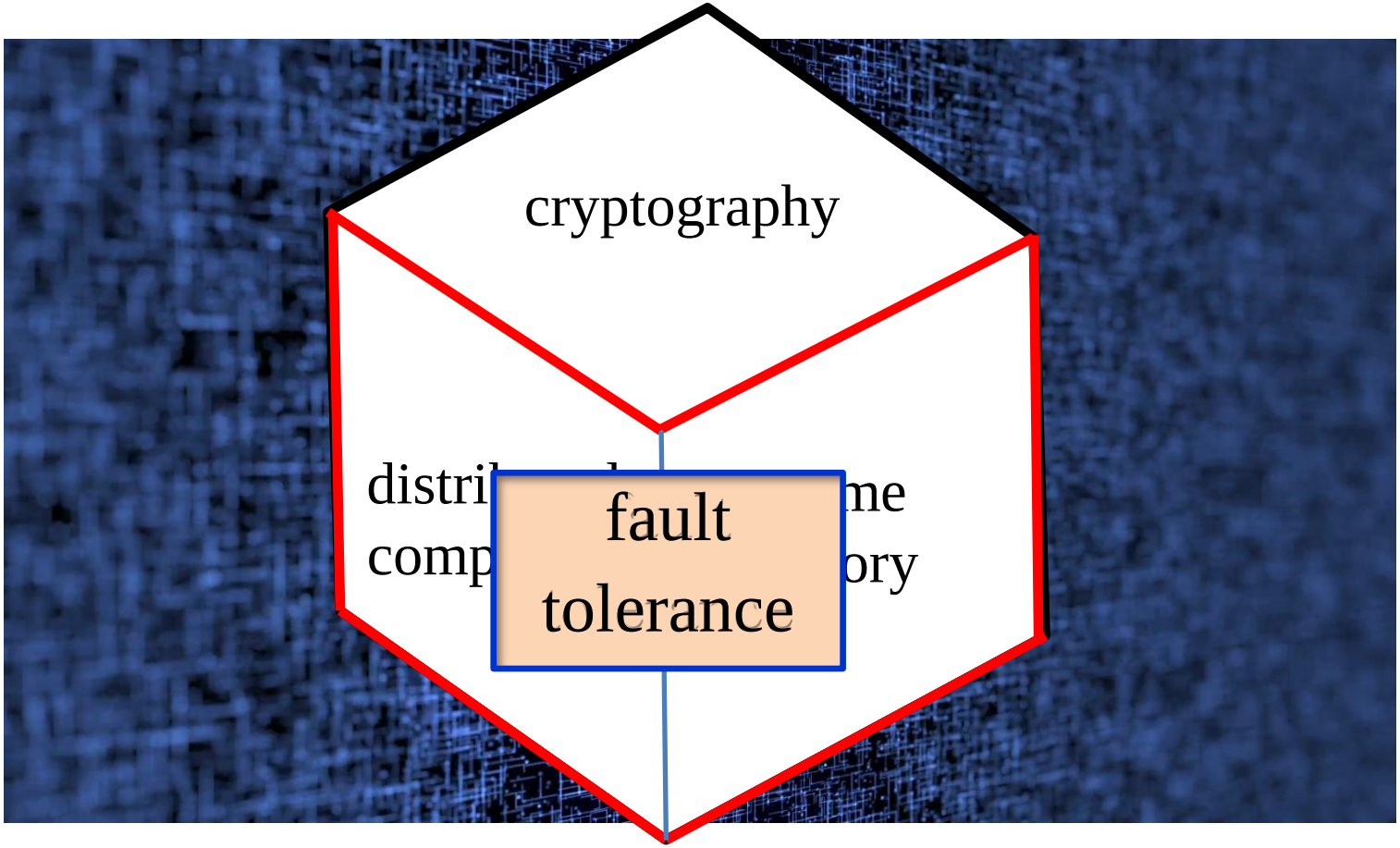
University of Rome

“Tor Vergata”

Understanding Blockchain technology



Understanding Blockchain technology



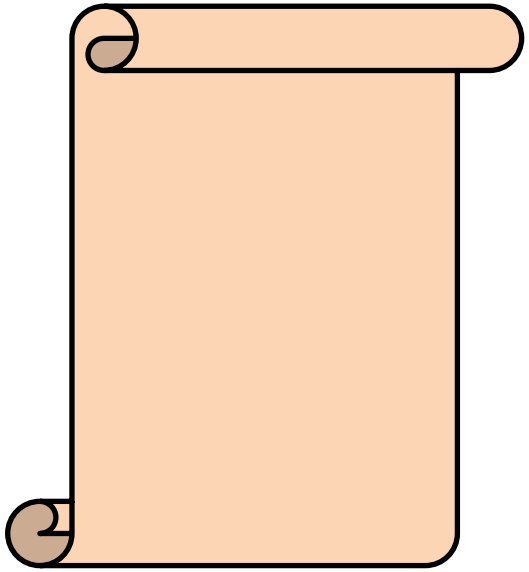
the distributed ledger problem

Roger Wattenhofer,
Distributed Ledger Technology – The science of the Blockchain

problem:

maintain a ledger containing a sequence of commands such that:

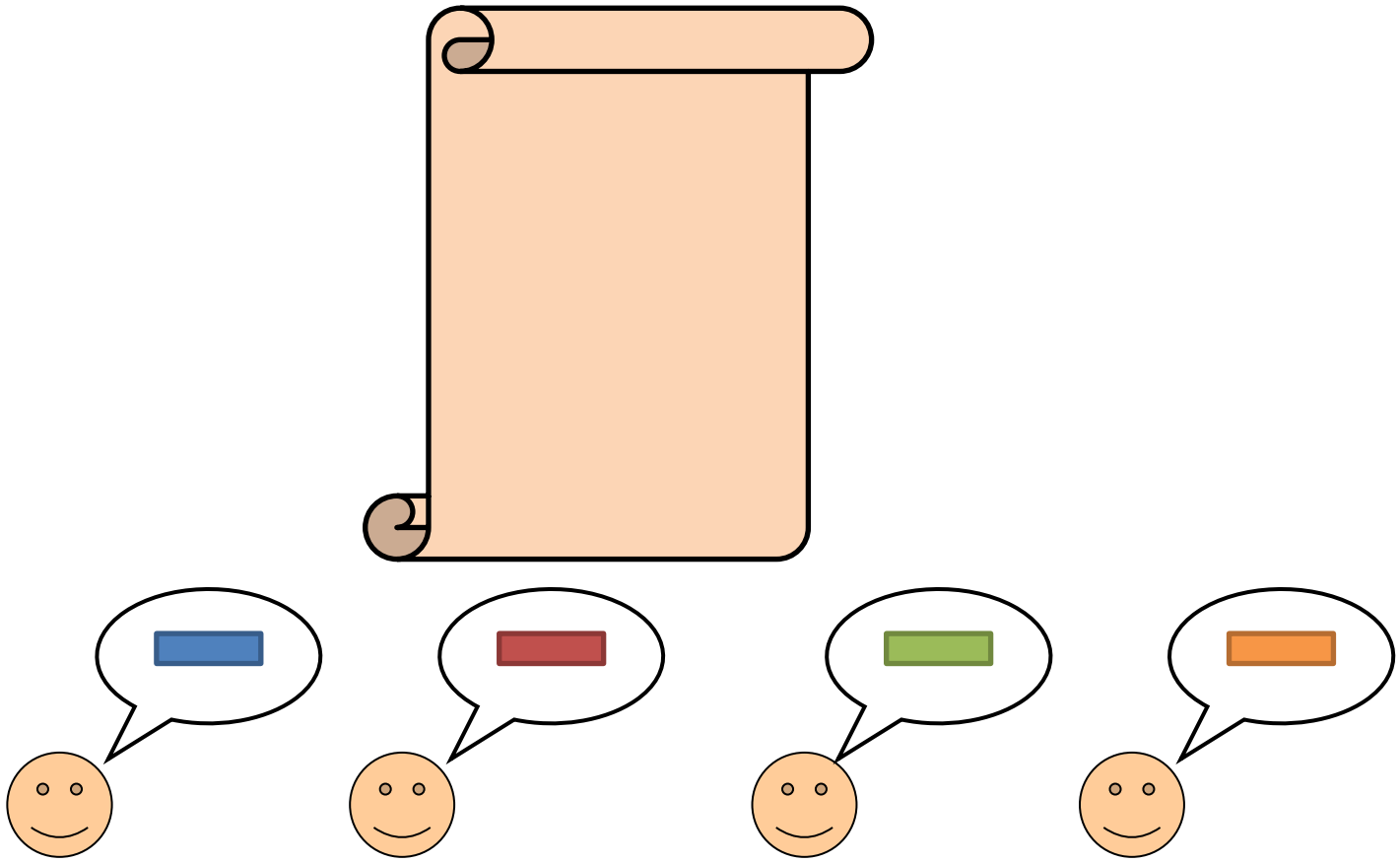
- all agents agree on the content of the ledger
- every agent can fairly write its commands



problem:

maintain a ledger containing a sequence of commands such that:

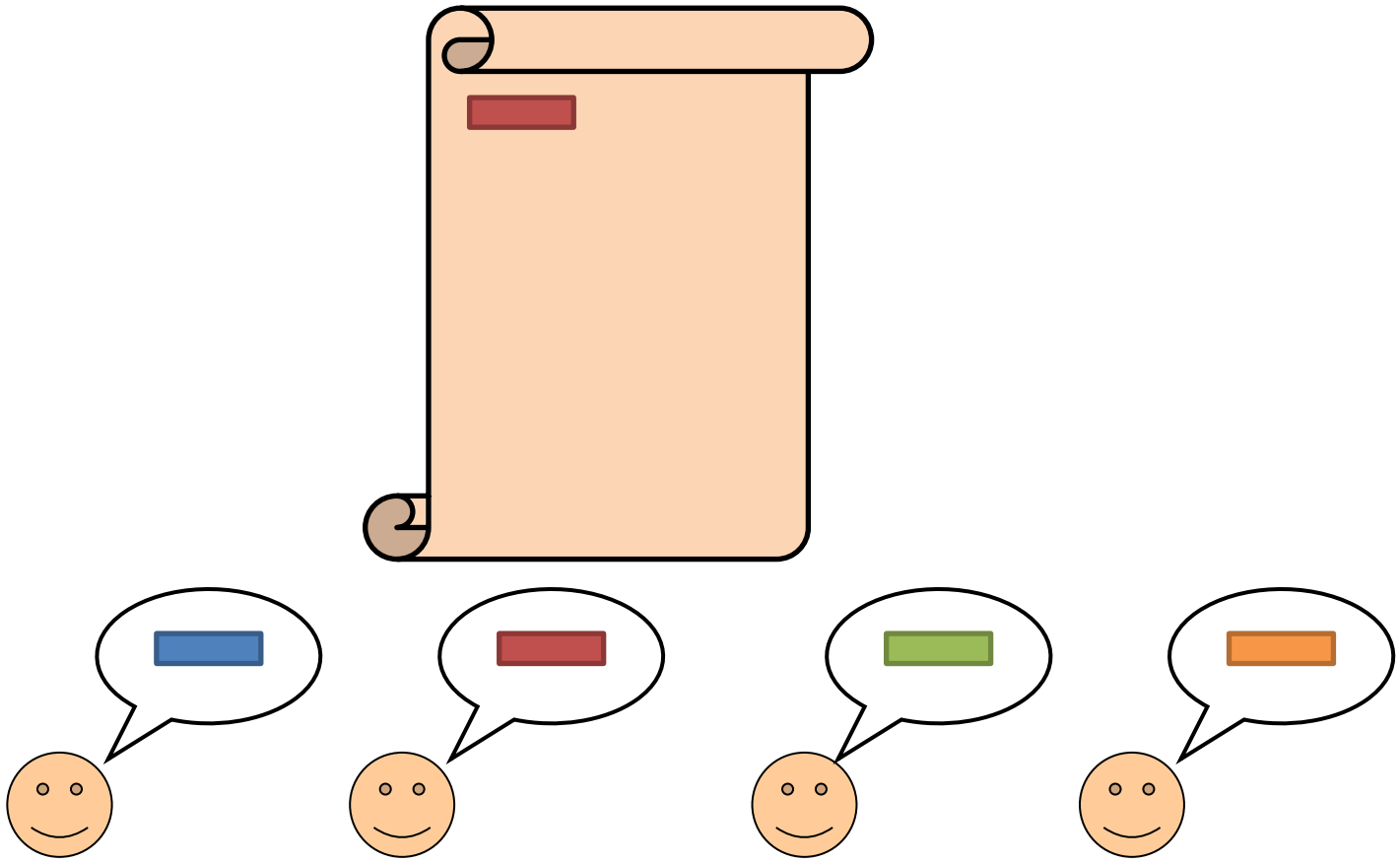
- all agents agree on the content of the ledger
- every agent can fairly write its commands



problem:

maintain a ledger containing a sequence of commands such that:

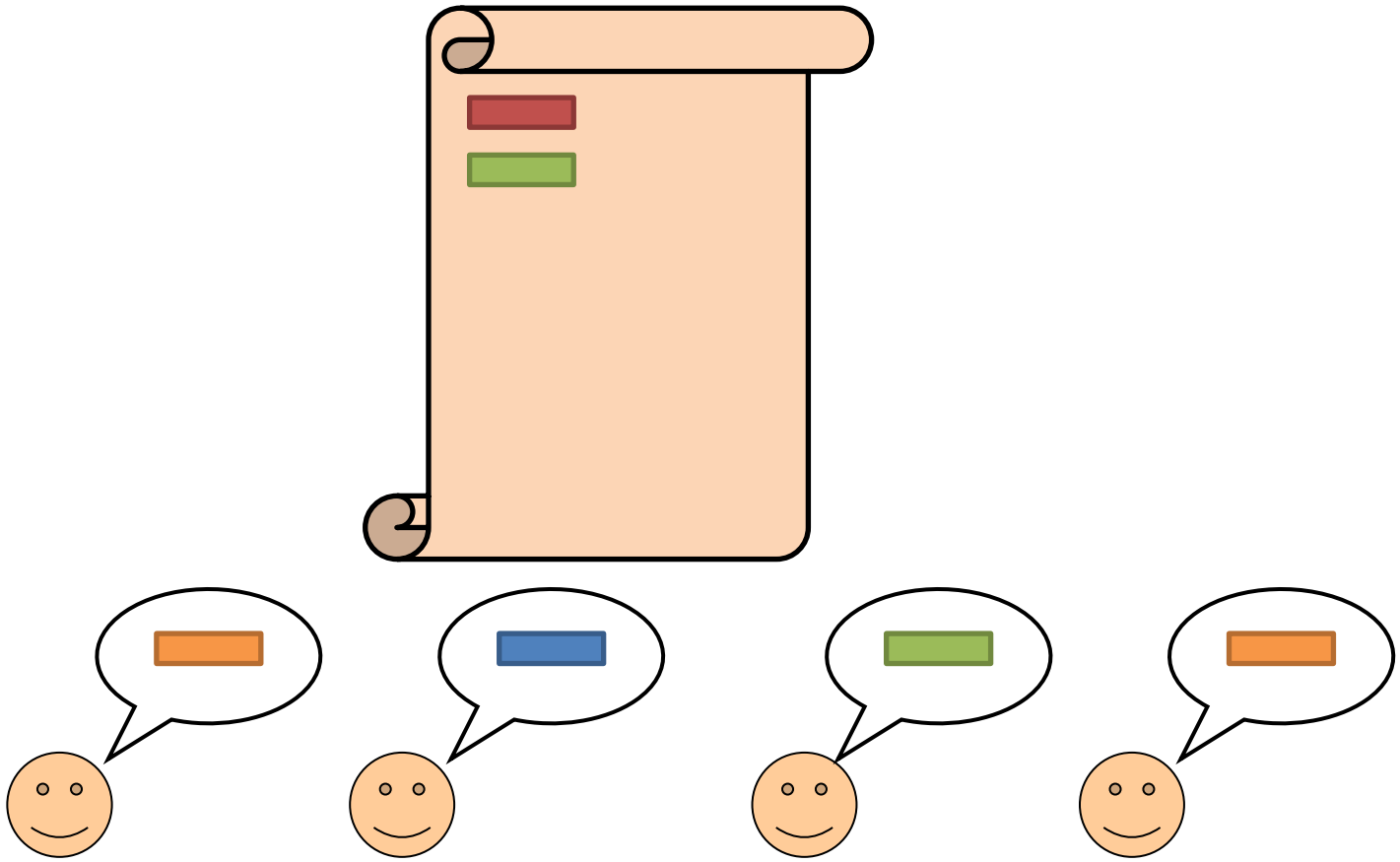
- all agents agree on the content of the ledger
- every agent can fairly write its commands



problem:

maintain a ledger containing a sequence of commands such that:

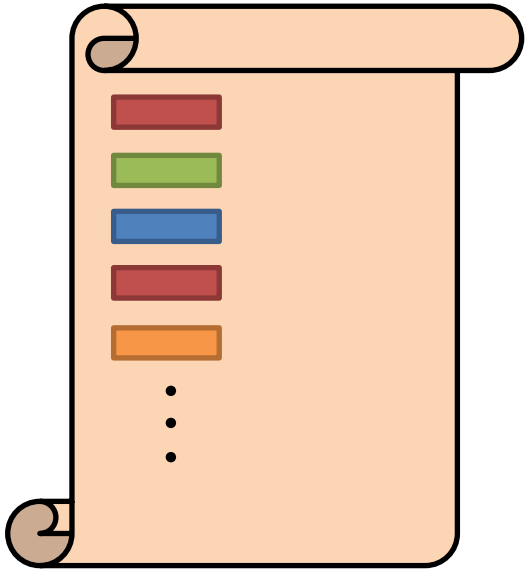
- all agents agree on the content of the ledger
- every agent can fairly write its commands



problem:

maintain a ledger containing a sequence of commands such that:

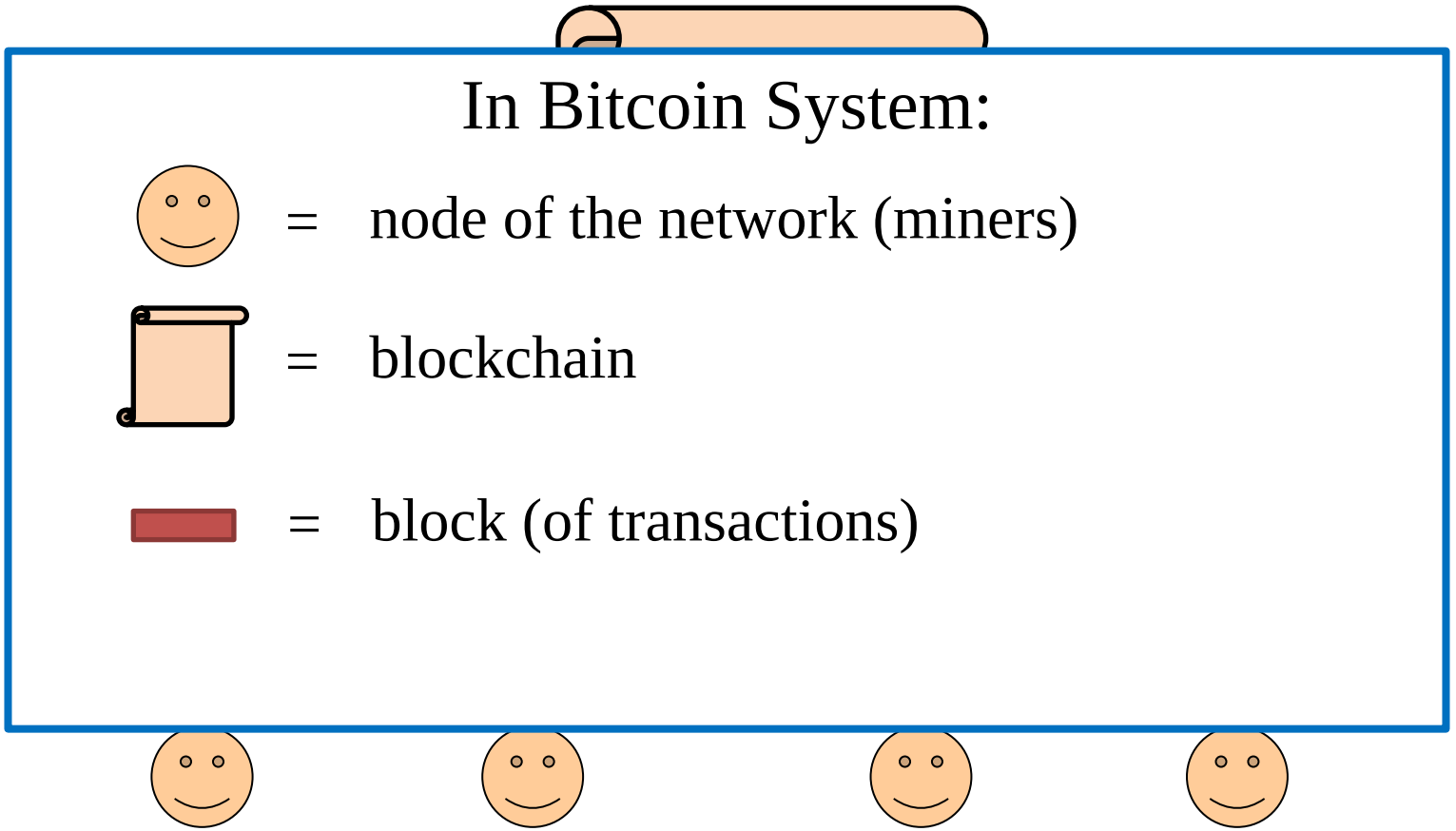
- all agents agree on the content of the ledger
- every agent can fairly write its commands



problem:

maintain a ledger containing a sequence of commands such that:

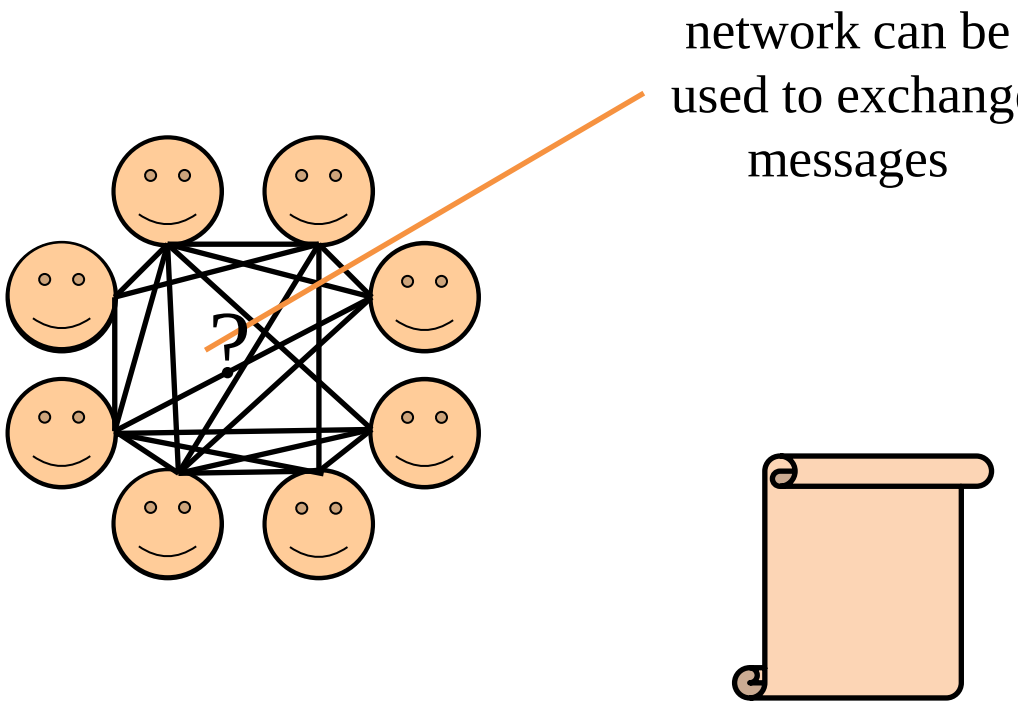
- all agents agree on the content of the ledger
- every agent can fairly write its commands



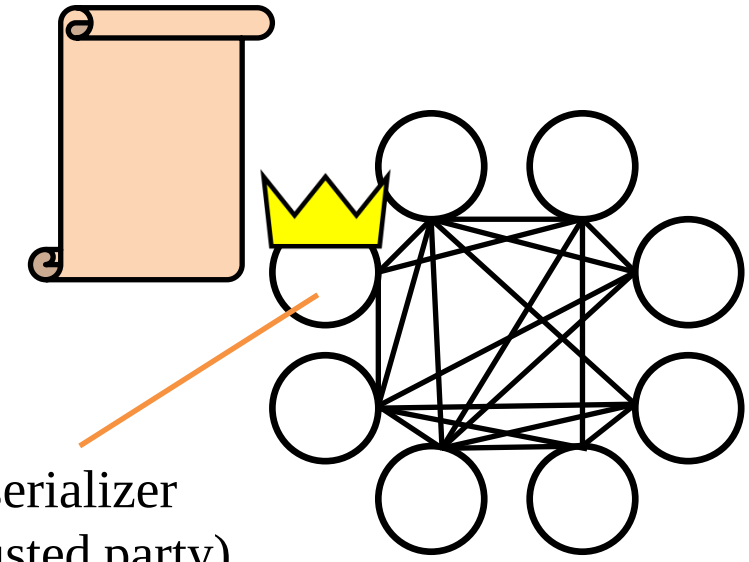
problem:

maintain a distributed ledger containing a seq. of commands such that:

- all agents agree on the content of the ledger
- every agent can fairly write its commands

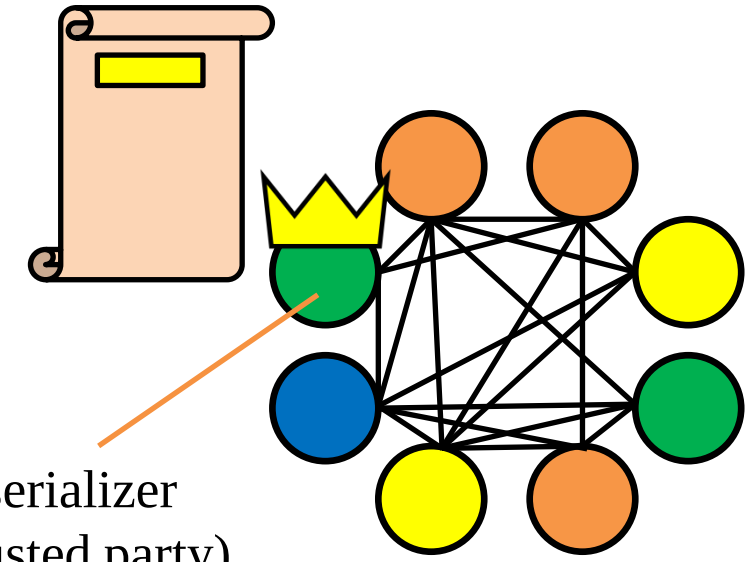


a simple solution



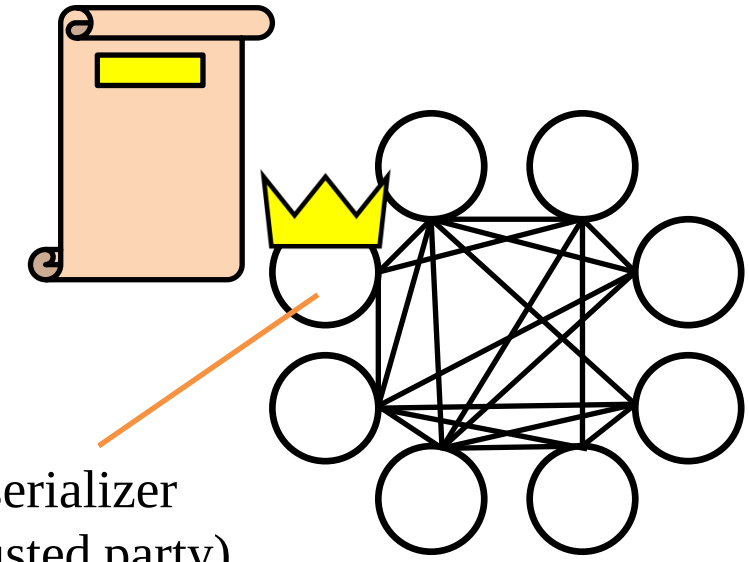
serializer
(trusted party)

a simple solution



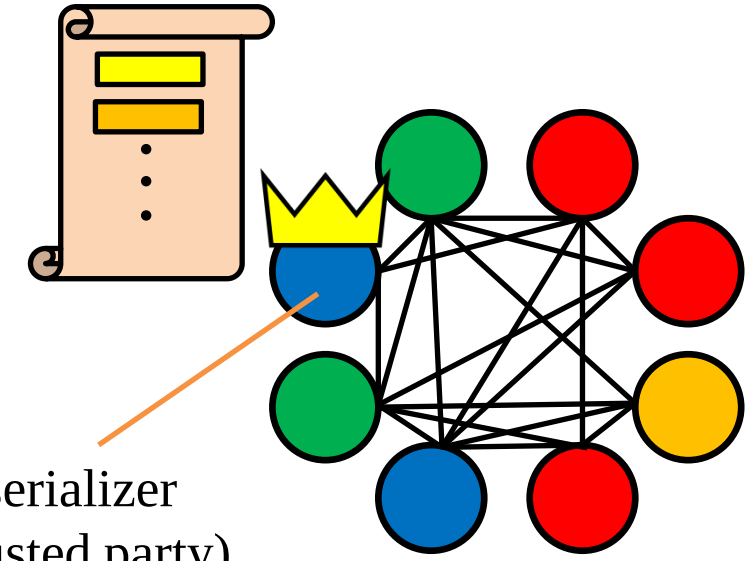
serializer
(trusted party)

a simple solution

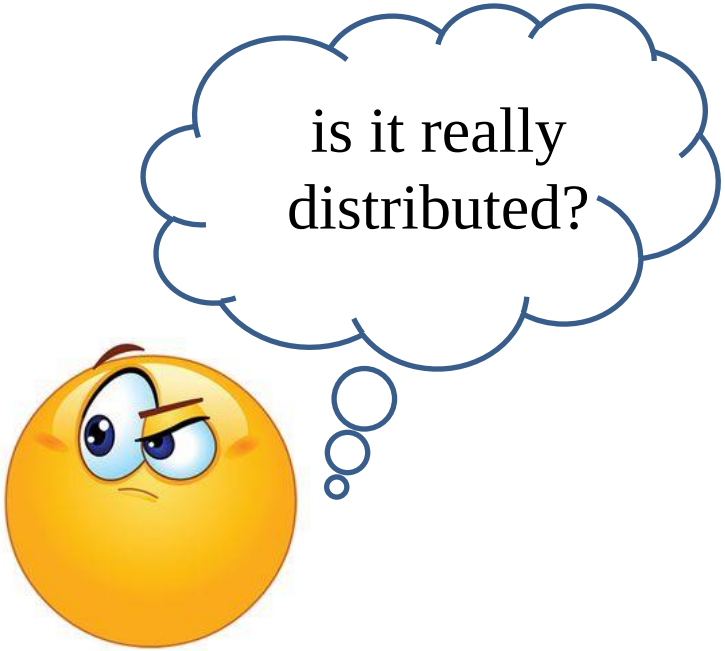


serializer
(trusted party)

a simple solution



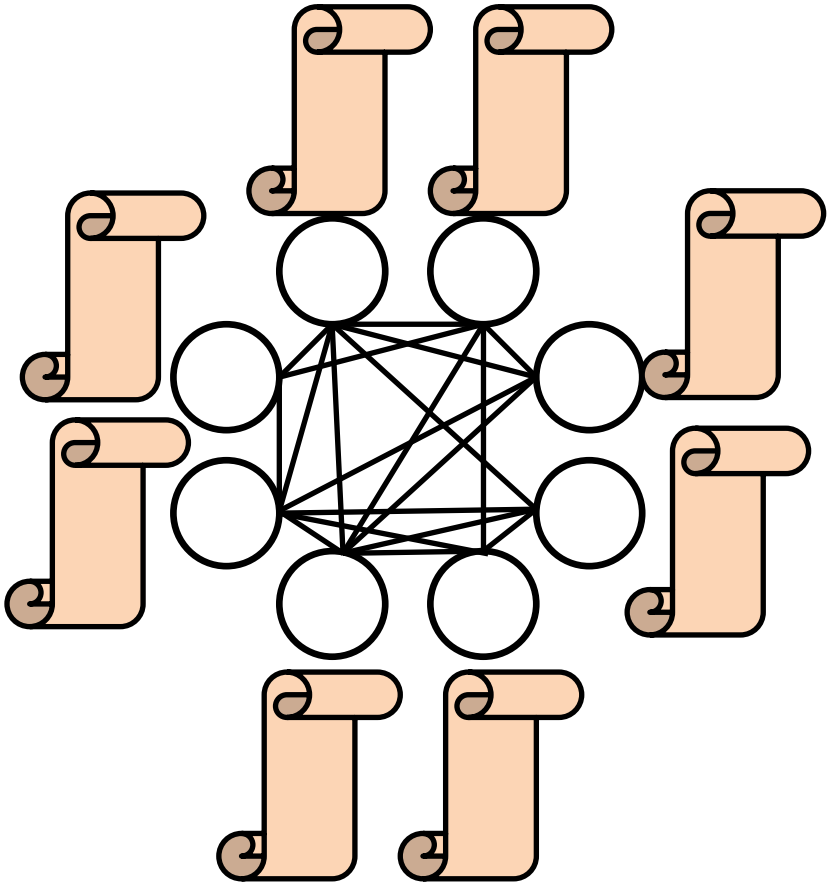
serializer
(trusted party)



- what if the serializer fails?
- what if the serializer is not honest?

fault tolerance

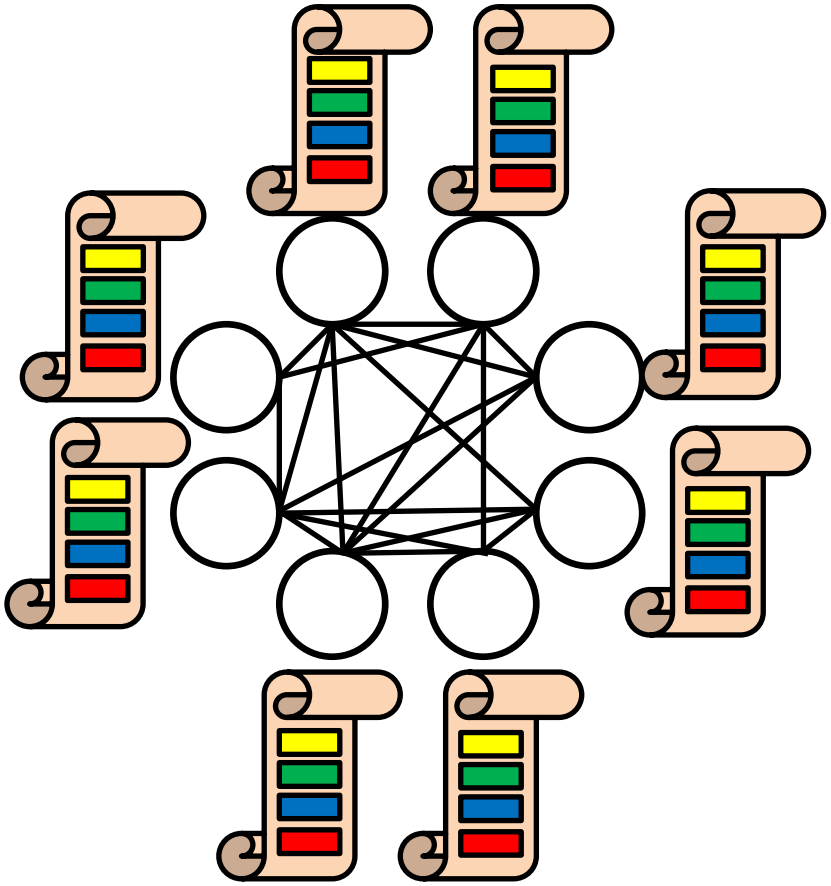
a better solution



Consistency:
all nodes always agree on the current state of the ledger

Eventual consistency:
all nodes eventually agree on the current state of the ledger (if no new updates are issued)

a better solution

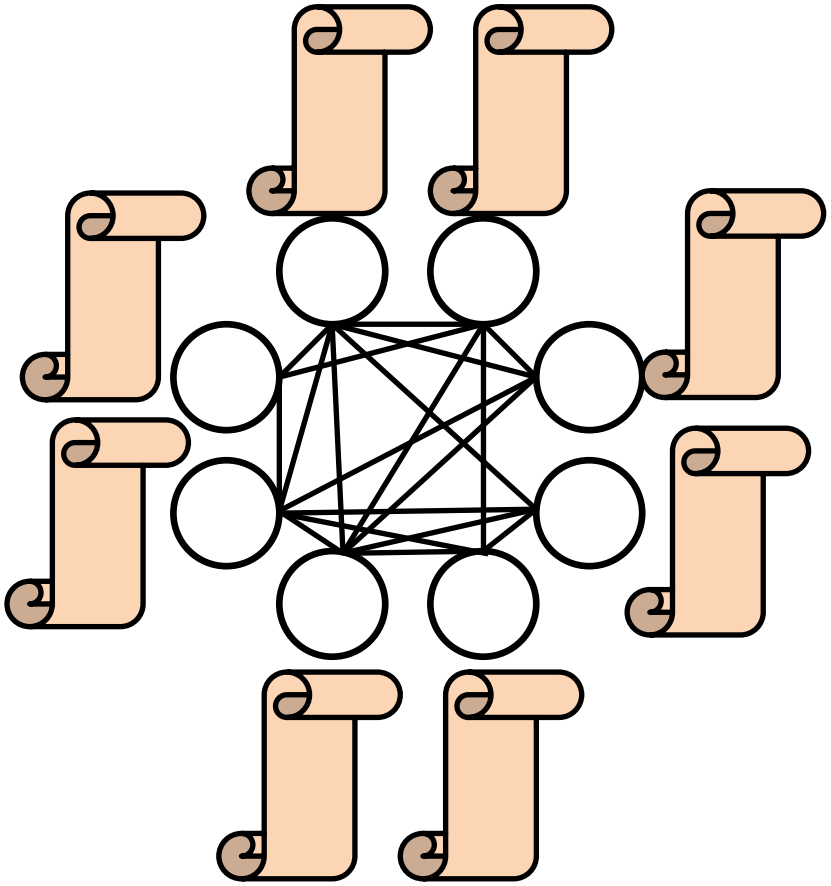


Consistency:
all nodes always agree on the current state of the ledger

Eventual consistency:
all nodes eventually agree on the current state of the ledger (if no new updates are issued)

how to solve the distributed ledger problem

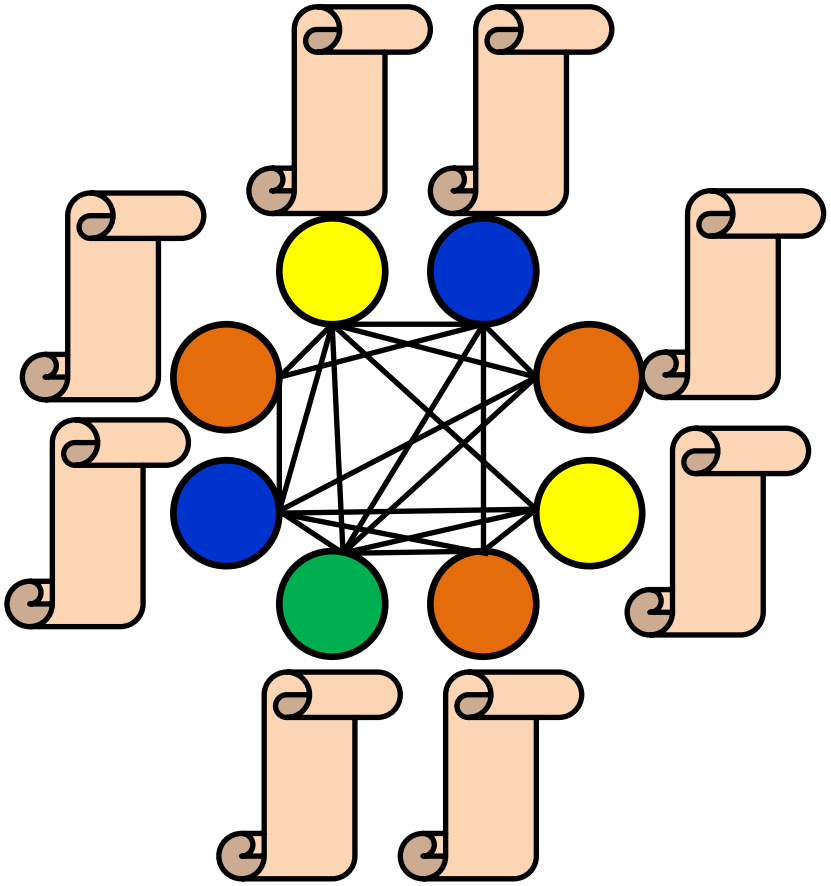
distributed ledger via repeated consensus



repeat:

- each node supports its command
- exchange messages to get an agreement on the winning command
- every node updates its (local) ledger with the winning command

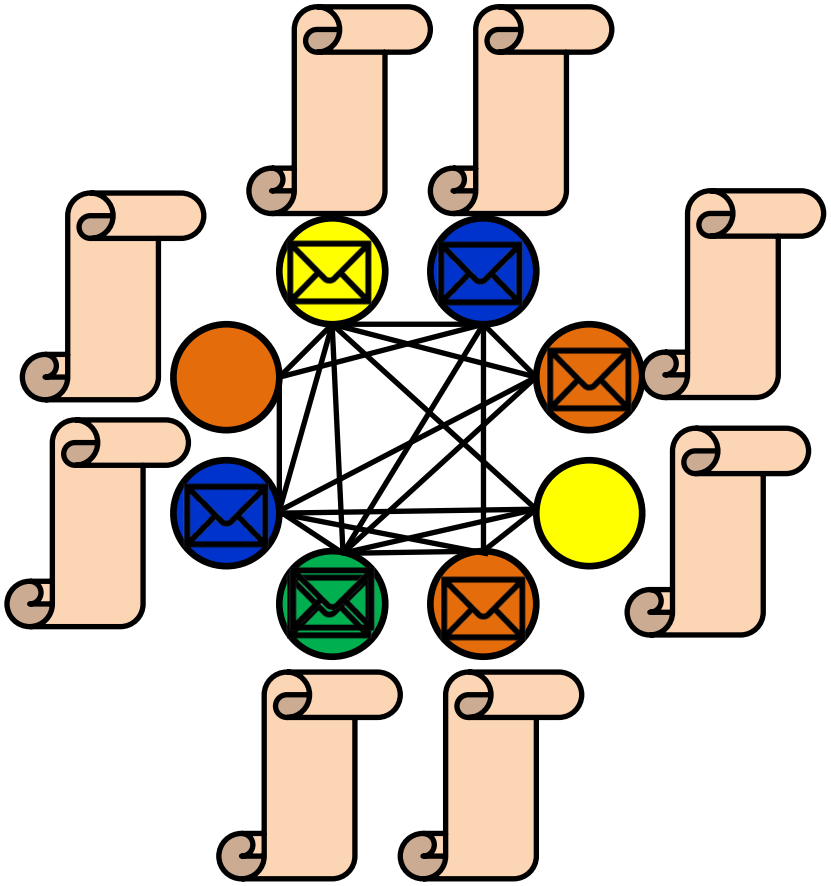
distributed ledger via repeated consensus



repeat:

- each node supports its command
- exchange messages to get an agreement on the winning command
- every node updates its (local) ledger with the winning command

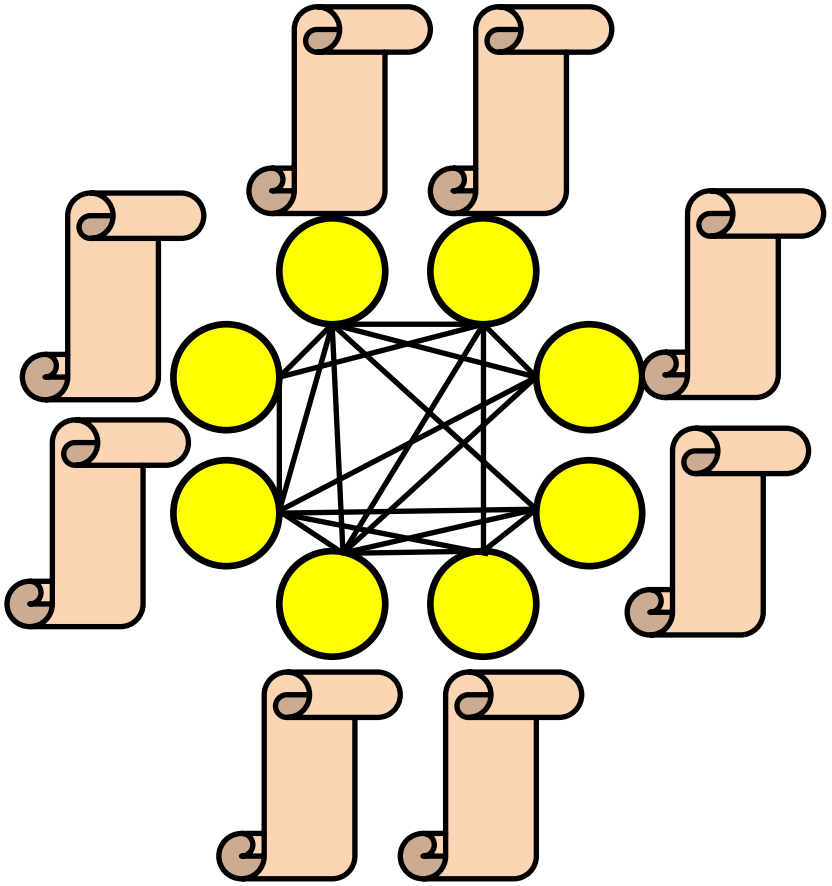
distributed ledger via repeated consensus



repeat:

- each node supports its command
- exchange messages to get an agreement on the winning command
- every node updates its (local) ledger with the winning command

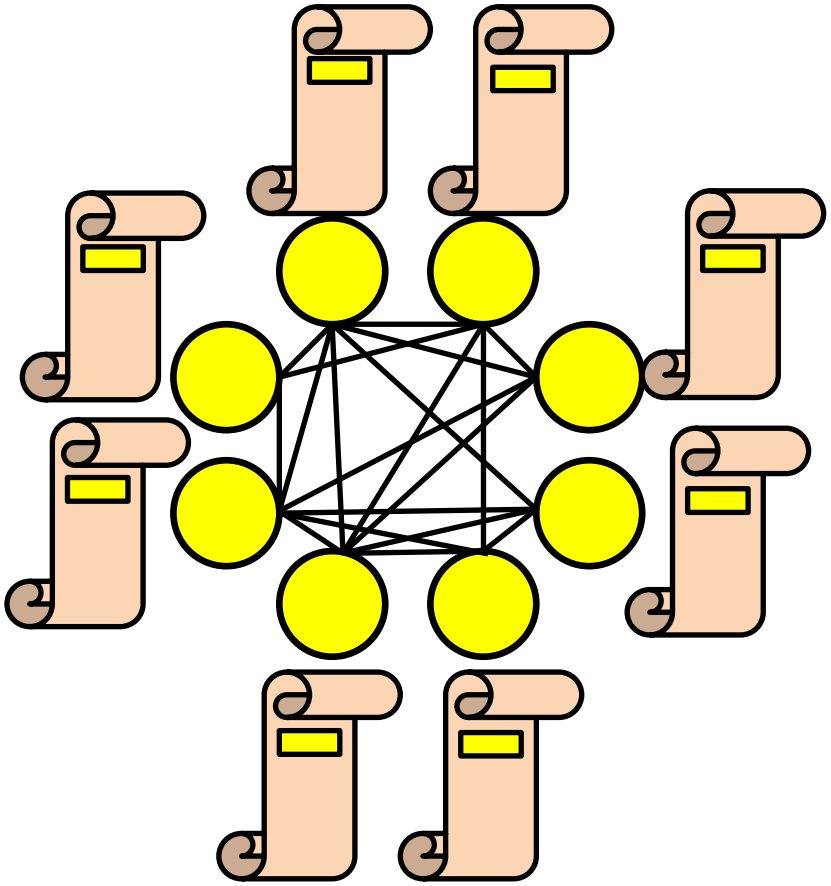
distributed ledger via repeated consensus



repeat:

- each node supports its command
- exchange messages to get an agreement on the winning command
- every node updates its (local) ledger with the winning command

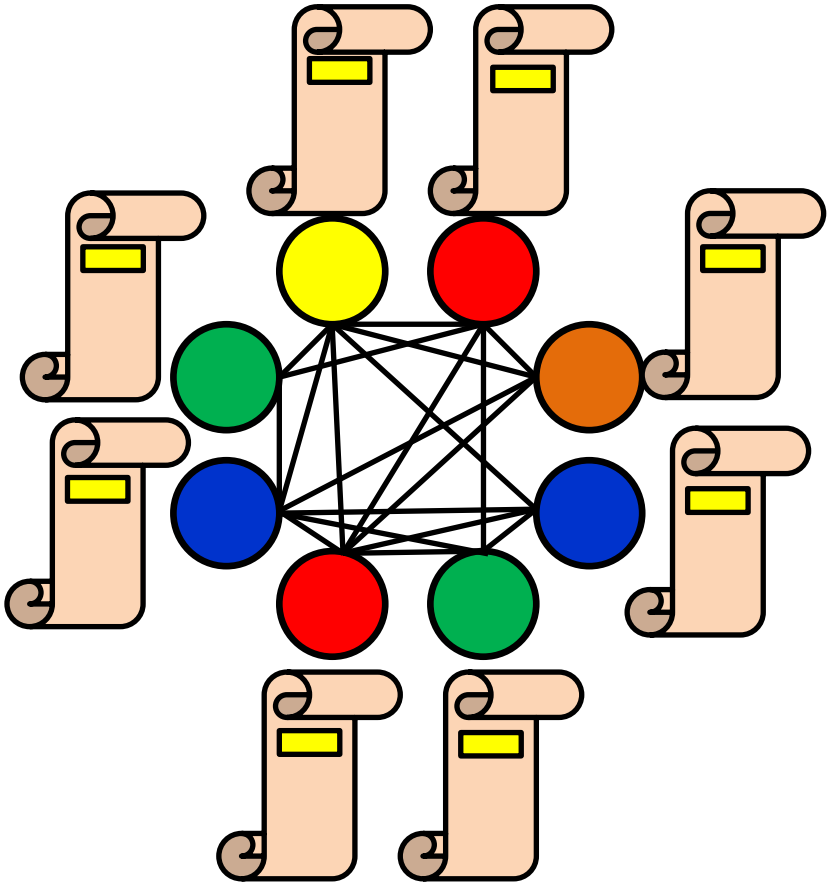
distributed ledger via repeated consensus



repeat:

- each node supports its command
- exchange messages to get an agreement on the winning command
- every node updates its (local) ledger with the winning command

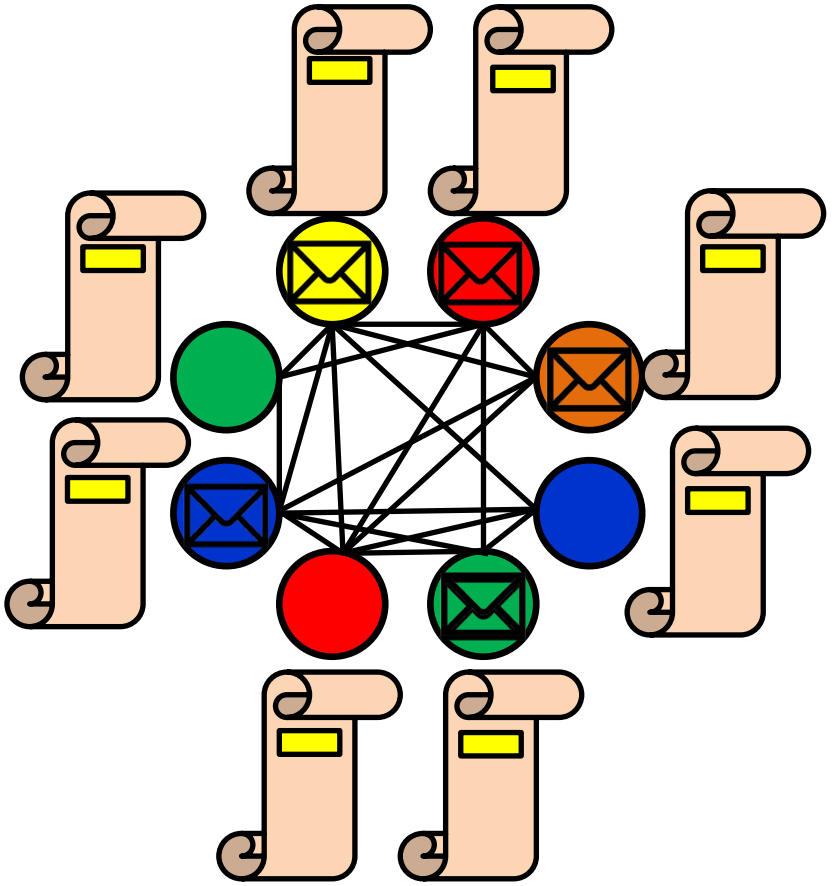
distributed ledger via repeated consensus



repeat:

- each node supports its command
- exchange messages to get an agreement on the winning command
- every node updates its (local) ledger with the winning command

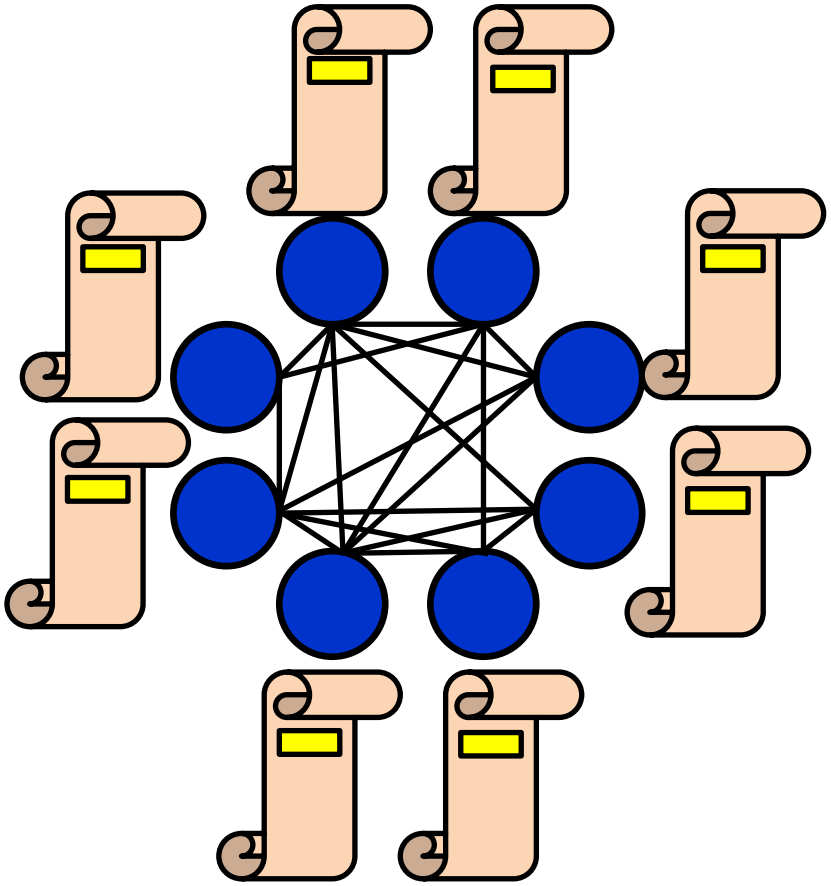
distributed ledger via repeated consensus



repeat:

- each node supports its command
- exchange messages to get an agreement on the winning command
- every node updates its (local) ledger with the winning command

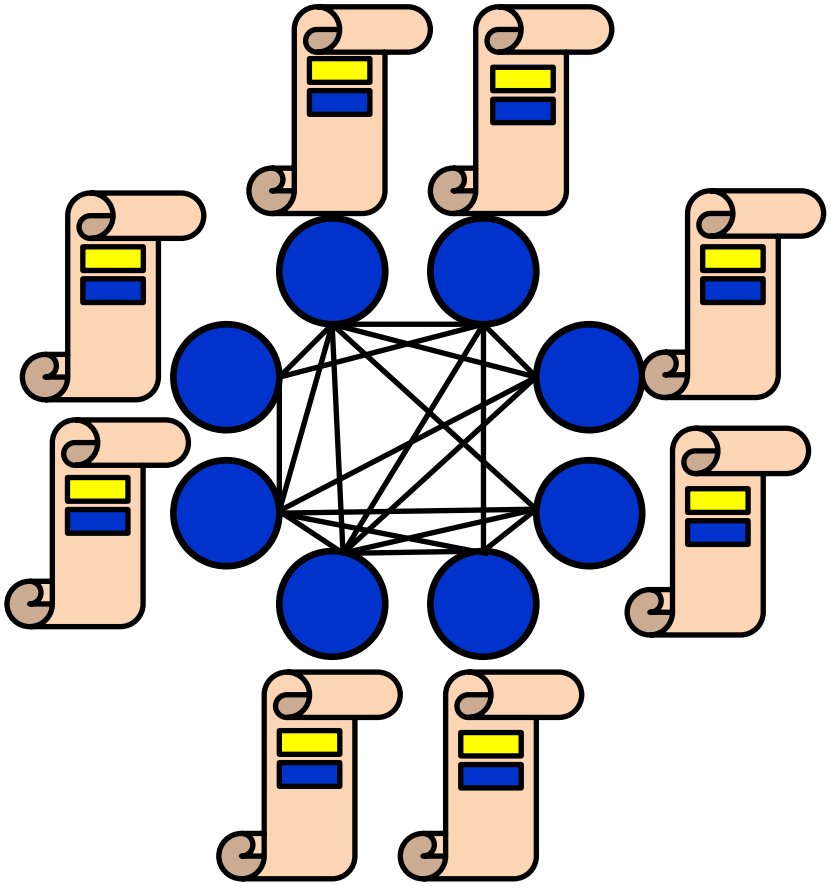
distributed ledger via repeated consensus



repeat:

- each node supports its command
- exchange messages to get an agreement on the winning command
- every node updates its (local) ledger with the winning command

distributed ledger via repeated consensus



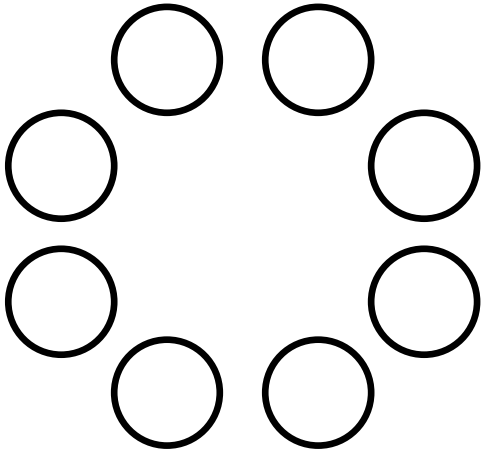
repeat:

- each node supports its command
- exchange messages to get an agreement on the winning command
- every node updates its (local) ledger with the winning command

the consensus problem

Consensus Problem

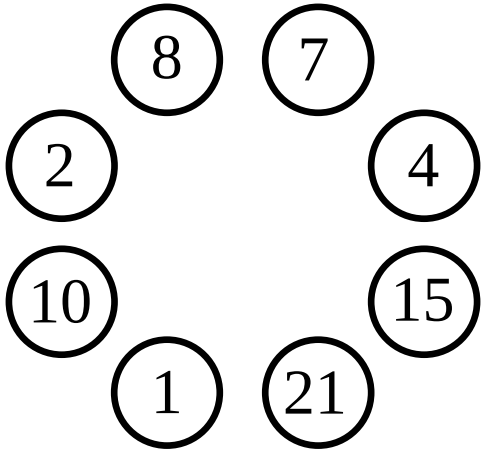
a set of *n* nodes



Consensus Problem

a set of *n* nodes

each node has:
  unique ID



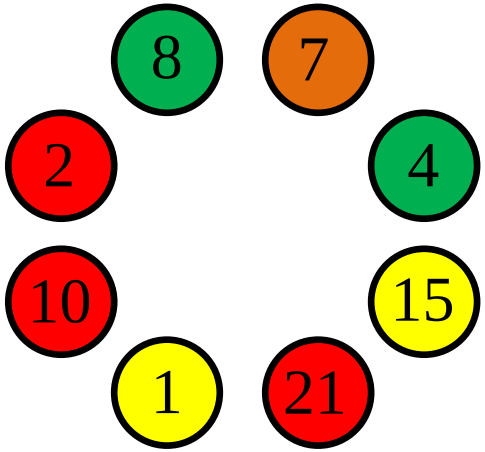
Consensus Problem

a set of *n* nodes

each node has:

unique ID

a color in {●●●…●}



Consensus Problem

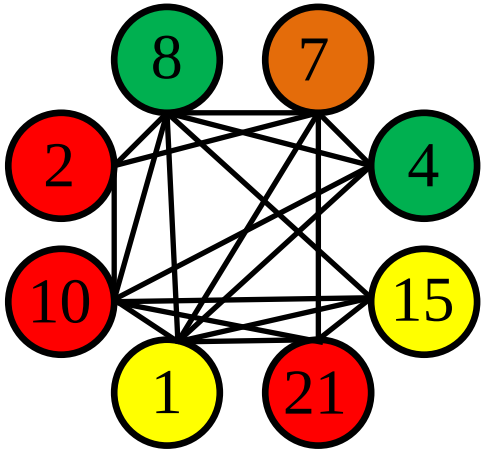
a set of *n* nodes

each node has:

unique ID

a color in {●●●…●}

an underlying communication graph *G*



Consensus Problem

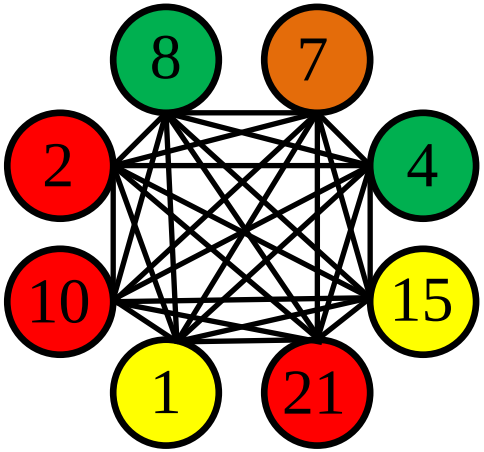
a set of n nodes

each node has:

unique ID

a color in {●●●…●}

an underlying communication graph G



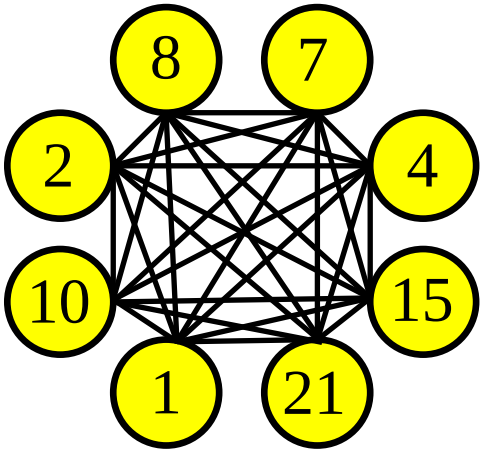
Goal: a *distributed protocol* guaranteeing

Termination (protocol eventually ends)

Agreement (monocromatic final configuration)

Validity

the *winning color* is initially supported by some node

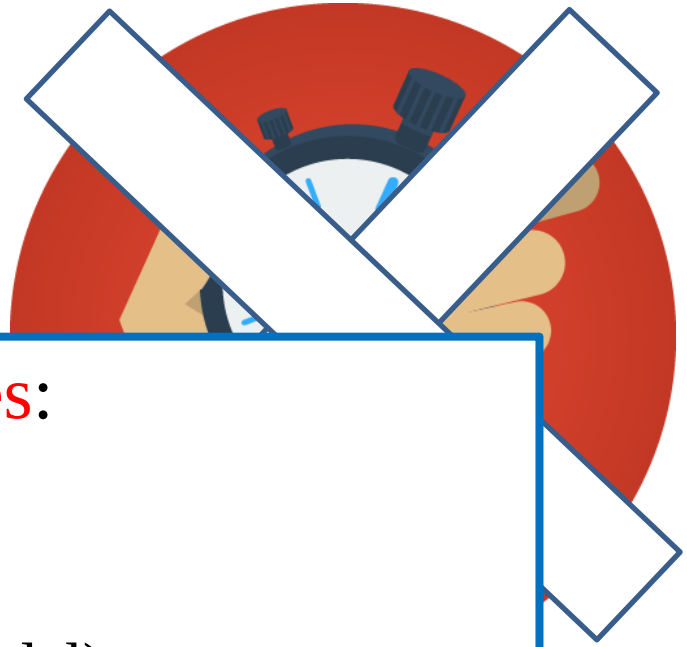


Models of computation

- shared clock
-computation proceeds in *rounds*
- messages sent in a round arrive in the next round
- in a round, a node receives mgs & computes & sends msgs

- no shared clock
- no rounds
- messages arrive in the finite but unbounded time

Models of computation



Quality measures:

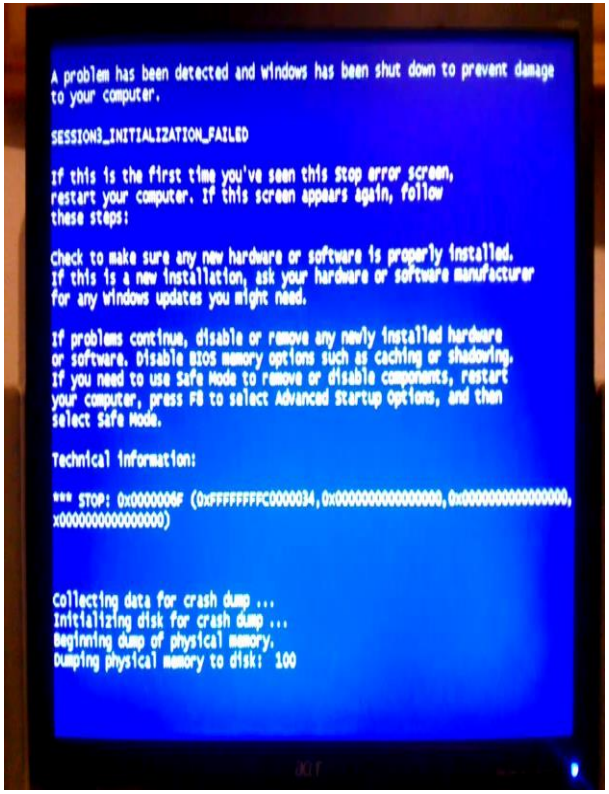
- # of messages
- size of the messages
- # of rounds (sync. model)

Fault tolerance

- must work when some nodes fail

- sl
-co
- m n the
the ed time
- in a round, a node receives msgs &
computes & sends msgs

Type of failures

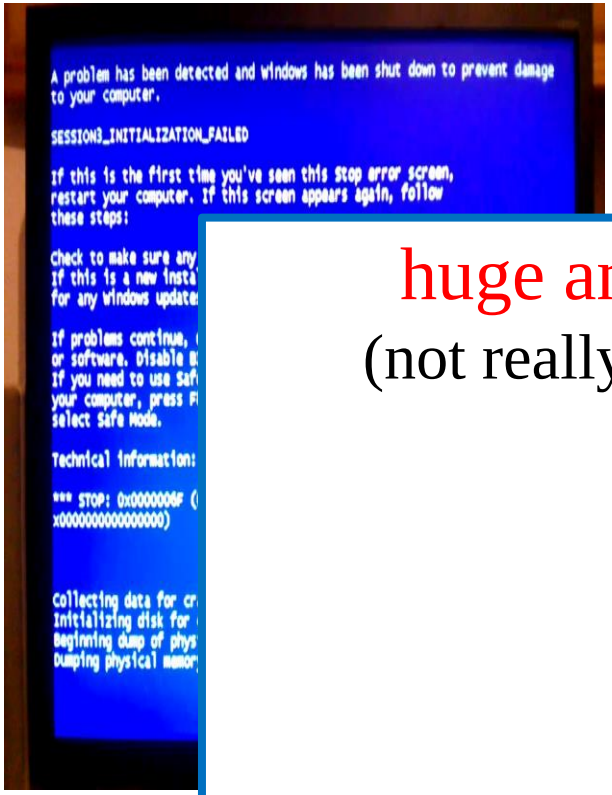


Crash failures

Byzantine failures

rational selfish failures

Type of failures



huge amount of work in the literature

(not really useful in the context for solving the distributed ledge problem)

Crash failures

Byzantine failures

rational selfish failures

Bitcoin system (in a nutshell)

Tim Roughgarden, Incentives in Computer Science
Lecture #9: Incentives in Bitcoin Mining,
http://timroughgarden.org/f16/l/l9.pdf

Transactions

A Bitcoin transaction:

1. One or more senders.
2. One or more receivers.
3. The amount of BTC (Bitcoins) transferred from each sender to each receiver.
4. A proof of ownership of the coins being transferred, in the form of a pointer back to most recent transactions involving the transferred coins.
5. A transaction fee, paid by the sender to the authorizer of the transaction.

Transactions

A transaction is valid if:

1. It has been cryptographically signed by all of the senders.
2. The senders really do own the coins being transferred.

This can be verified using the senders' public keys.

This can be verified as follows:

-transactions are broadcast to all other users (through a peer-to-peer network);
-all users keep track of all transactions that have ever been authorized;
-thus, everyone knows everyone's current balance

the ledger: the record of all the authorized transactions.

Transactions

Two important questions:

1. How do transactions get authorized and added to the ledger?
   (Traditionally, this would done by a centralized entity like a bank.)

2. How do Bitcoins get created in the first place?
   (Traditionally, money is printed by the government.)

Blocks

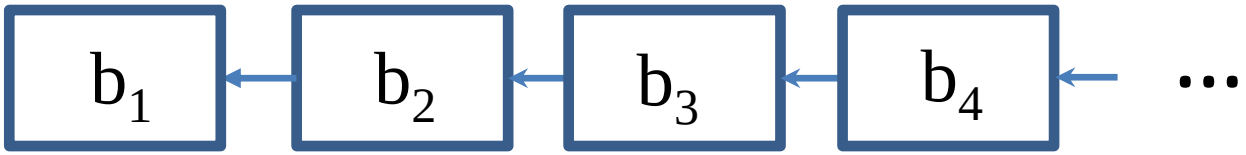
Transactions are added to the ledger in groups, known as blocks.

A block contains:

1. One or more transactions.
2. A hash of the previous block.
3. A nonce. (I.e., a bunch of bits that can be set arbitrarily.)

This imposes a natural linked list-type structure on the ledger:
-the predecessor of a block b_2 being the block b_1 whose hash matches the hash stored in b_2.

Blockchain



Blockchain

Some issues:

-How do new blocks get added to the blockchain?

-Who can do it?

-Why should they bother?

-How can we make sure that everybody agrees on the contents of the blockchain?

Two key ingredients:

1. Any user can authorize a block. Bitcoin incentivizes users to do authorizations through explicit monetary rewards (in BTC, naturally).
2. Authorizing a new block of transactions involves a proof of work, meaning that the authorizer has to solve a computationally difficult puzzle.

Computational difficult puzzle

A block *b* is valid if *h*(*b*) is sufficiently close to 0.

the leading *l* bits of *h*(*b*) should all be 0, where *l* is a parameter

h: pre-agreed upon hash function
(currently, SHA-256)

a block contains:
transactions, the hash of the previous block, the nonce

has to be set properly set in order to make the block valid

parameter *l* chosen to keep the rate of valid block creation roughly every ten minutes

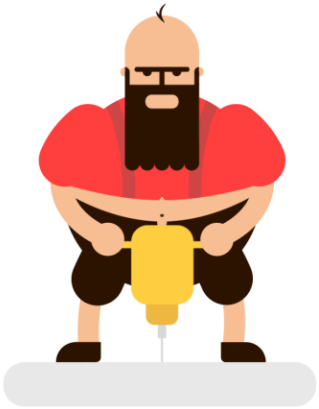
Block Rewards and Bitcoin Mining

Bitcoin mining: the process of finding new valid blocks.

A *miner*:
-chooses a subset of the transactions;
-inserts the hash of the current last block;
-arbitrarily set the bits in the nonce (and hope that the resulting block is valid).

h is a cryptographic hash function

the accepted belief is that there is no algorithm for finding a valid block that is smarter or faster than random guessing or exhaustive search



cryptographic hash function

$h:\{0,1\}^* \longrightarrow \{0,1\}^k$

robust to (at least) the following attacks:

Preimage attack:
give $x \in \{0,1\}^k$, find a such that $h(a)=x$.

Second preimage attack:
given a, find $b \neq a$ such that $h(a)=h(b)$.

Birthday attack:
find a and $b \neq a$ such that $h(a)=h(b)$.

Block Rewards and Bitcoin Mining

The reward that a miner gets for adding a new (valid) block to the blockchain has two ingredients:

1. A flat reward that does not depend on the contents of the block
   (When Bitcoin debuted this reward was 50 BTC, but the protocol dictates that this amount gets cut in half every four years. Currently, it is 6.25.)

2. The sum of the transaction fees of the transactions in the block
   (Currently, transaction fees are non-zero but typically constitute only a few percent of the overall reward.)

remark: create a new block is the only way that new money gets printed

the miner gets the new mined BTCs as special transaction inserted into the mined block

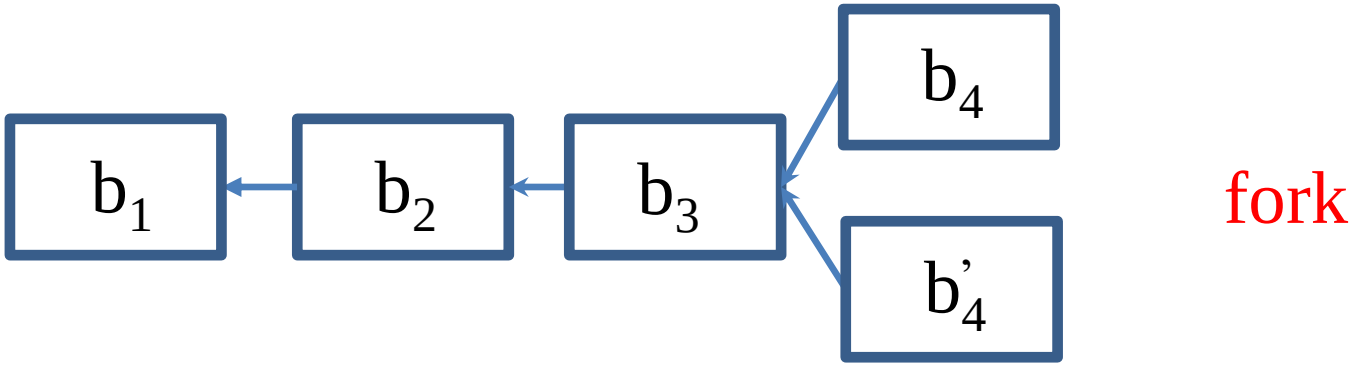
Forks

When a new valid block has been found:

-the miner is supposed to immediately broadcast it across the entire network, so that it gets appended to the blockchain;

-If someone else announces a new valid block first, then the miner restarts this procedure, now using only transactions not already authorized by the new block, and using the hash of the new block.

when two miners solve a block at roughly the same time:



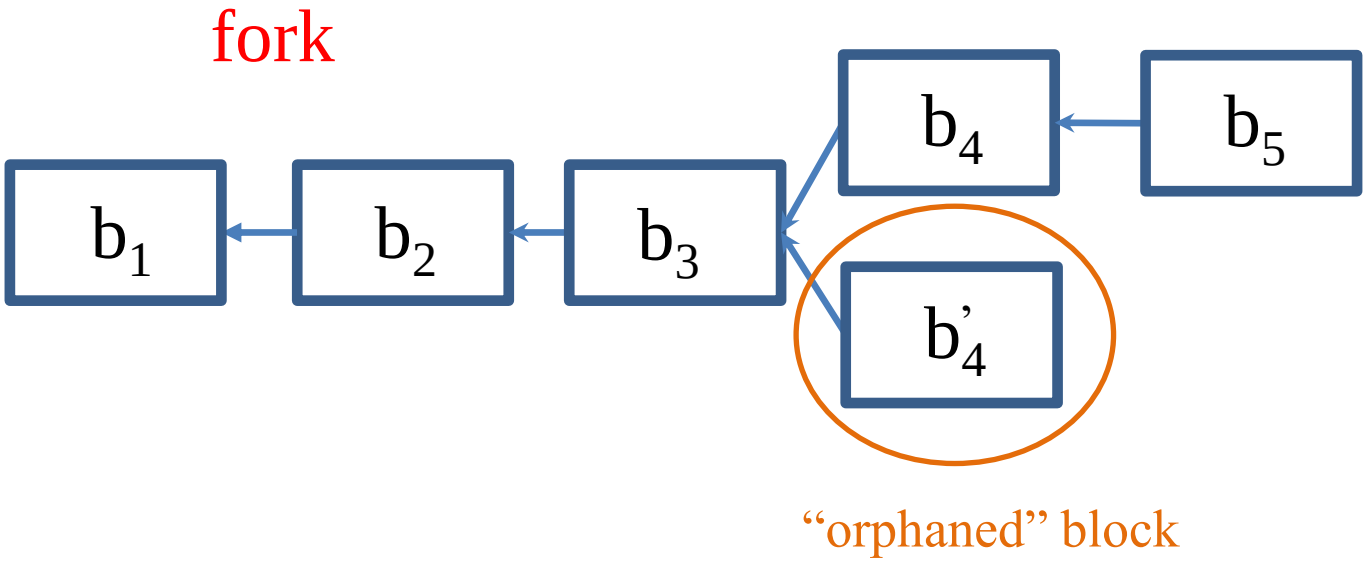
fork

Forks

Intended behavior when there is a fork:

-a user should regard the longest branch as the valid one;

-break ties according to the block that it heard about first.



robustness

Bitcoin Mining Protocol:

- work on the next block to be added to the longest chain
- announce the solved block as soon as you get it

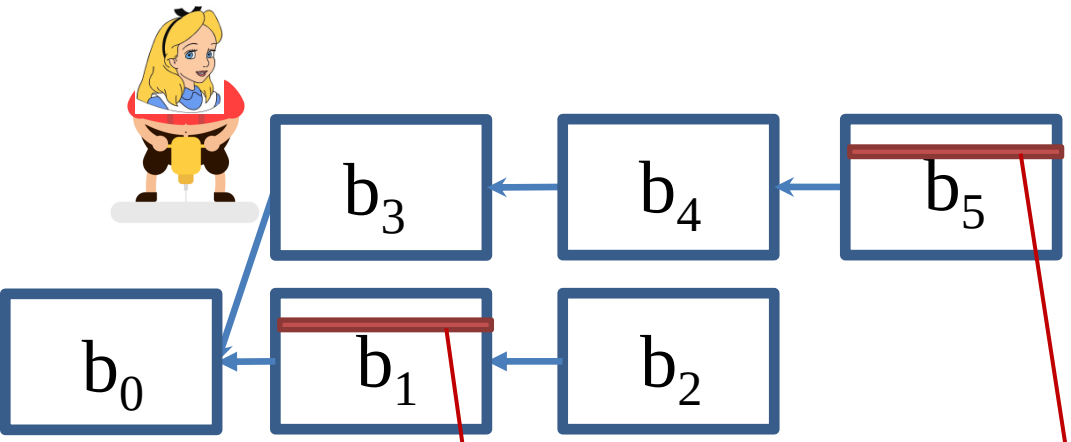
Does a miner have convenience to follow the protocol?

The Double-Spend Attack

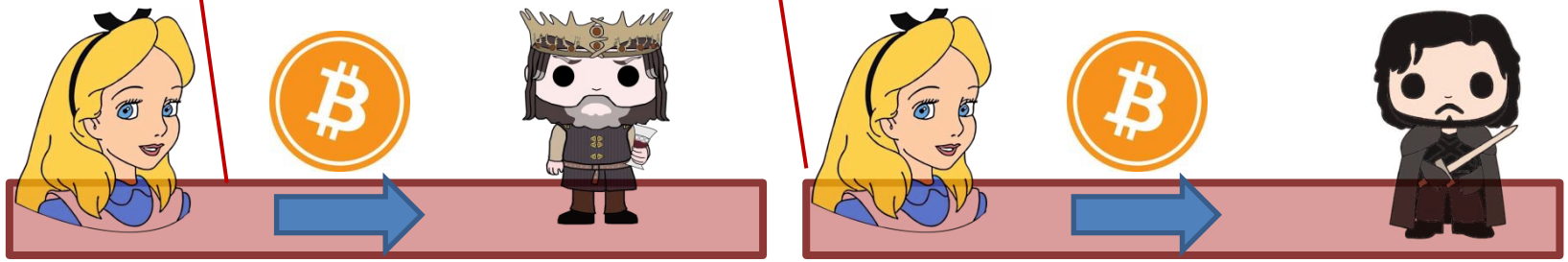
Idea: miners deliberately create forks.

Assumption: Bob only ships the purchased goods to Alice once another block b_2 has been appended to b_1.

α: fraction of the computational power controlled by Alice



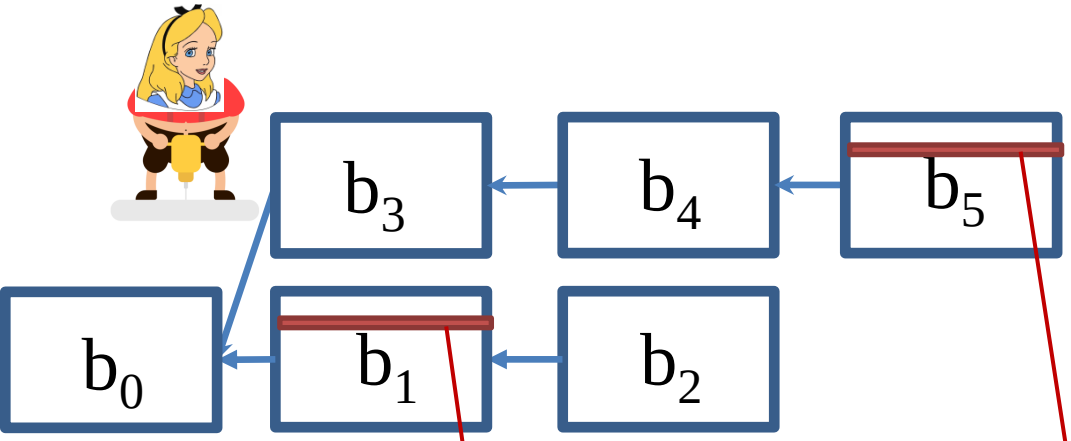
probability of mining 3 blocks in a row: α^3



The Double-Spend Attack

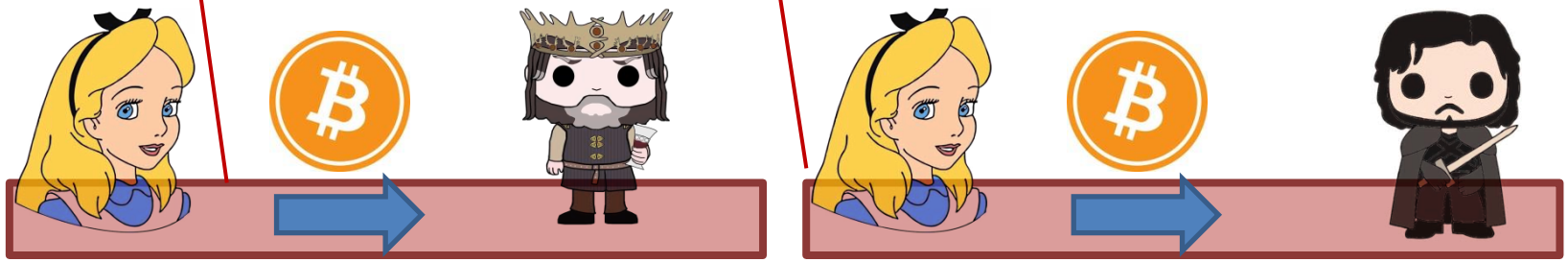
Idea: miners deliberately create forks.

Assumption: Bob only ships the purchased goods to Alice once k other blocks have been appended to b_1.



α: fraction of the computational power controlled by Alice

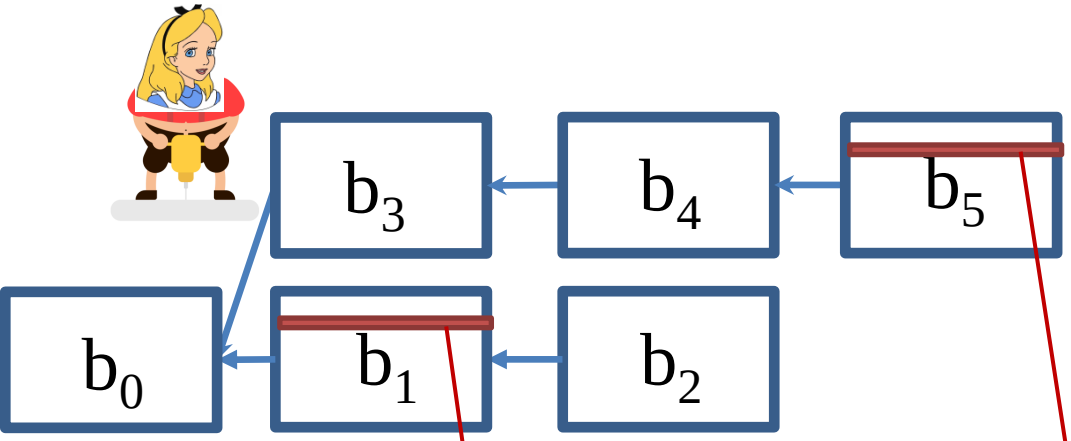
probability of mining k+2 blocks in a row : α^{k+2}



The Double-Spend Attack

Idea: miners deliberately create forks.

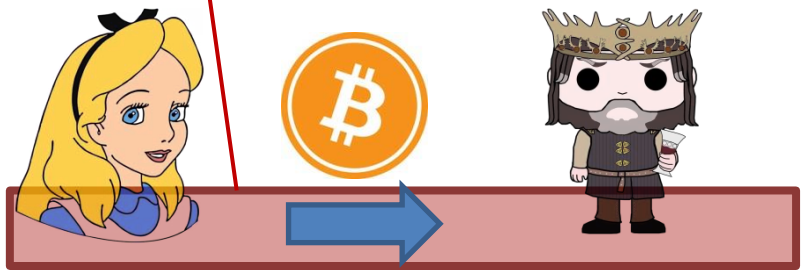
Assumption: Bob only ships the purchased goods to Alice once k other blocks have been appended to b_1.

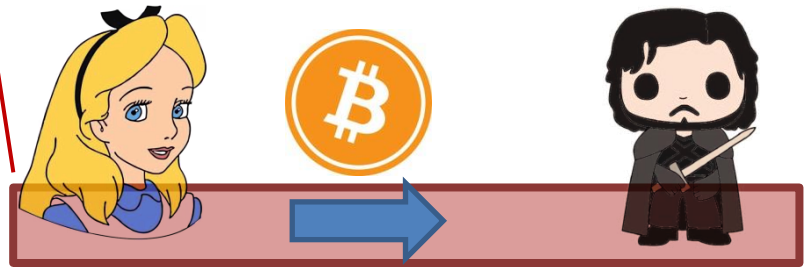


α: fraction of the computational power controlled by Alice

if Alice keeps mining until her branch (hopefully) becomes longer:

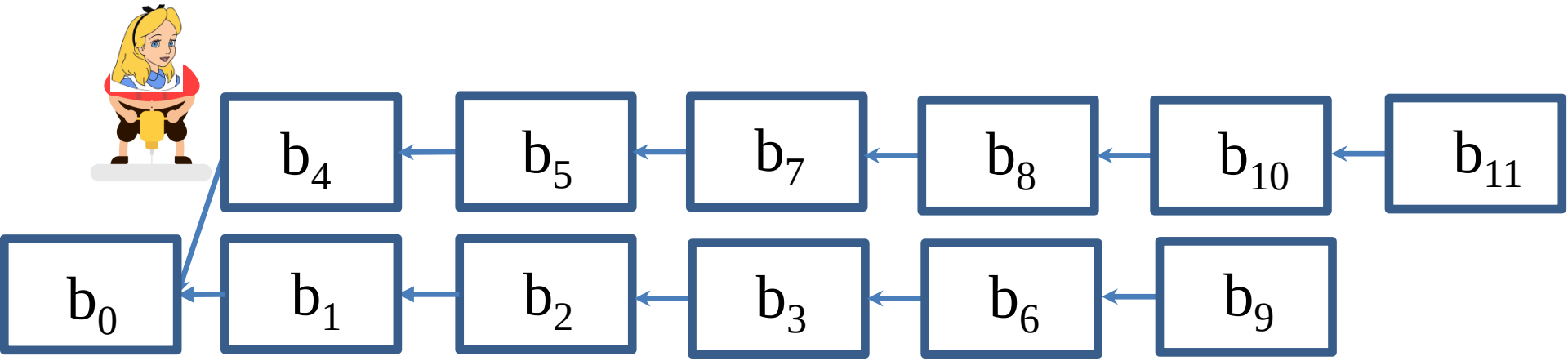
probability of success:
$[\alpha/(1-\alpha)]^{k+2}$





The 51% Attack

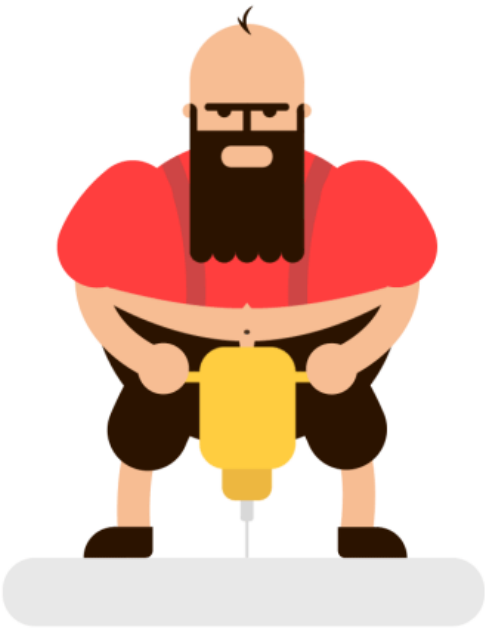
if Alice controls > 50% of the computational power



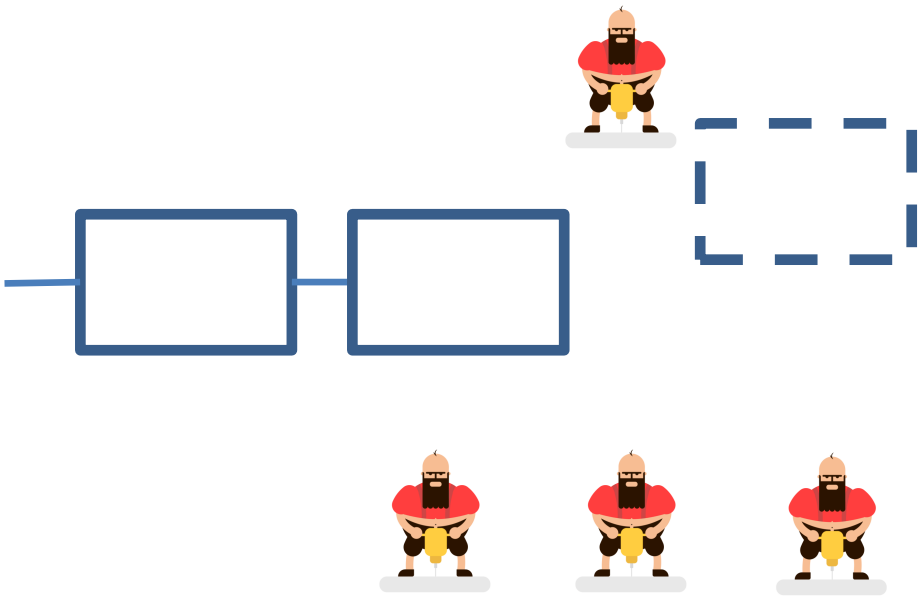
remark:

Bitcoin is not intended to function when a single entity controls more than 50% of the computational power

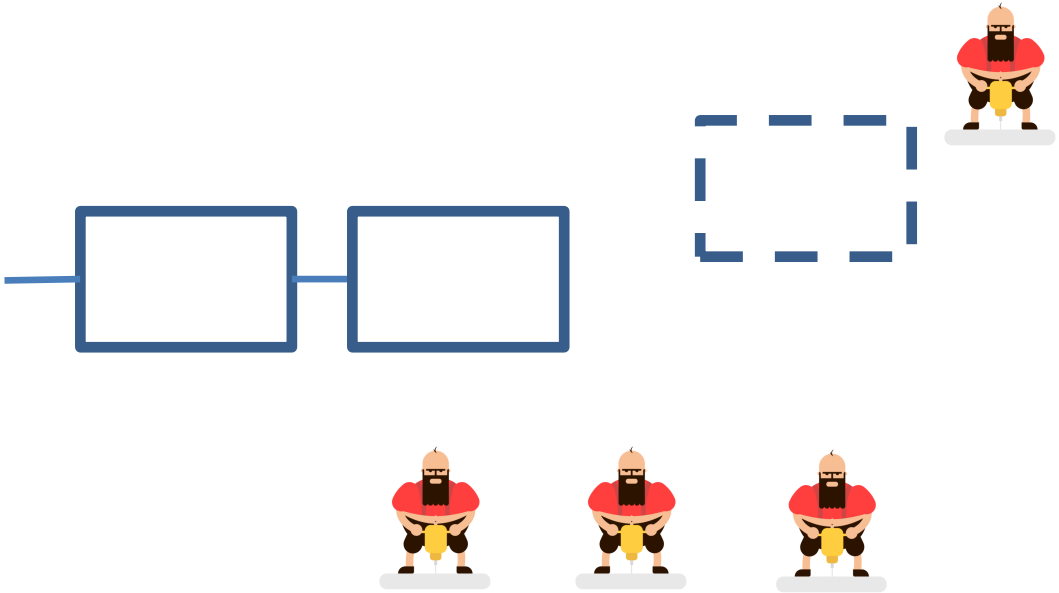
selfish mining



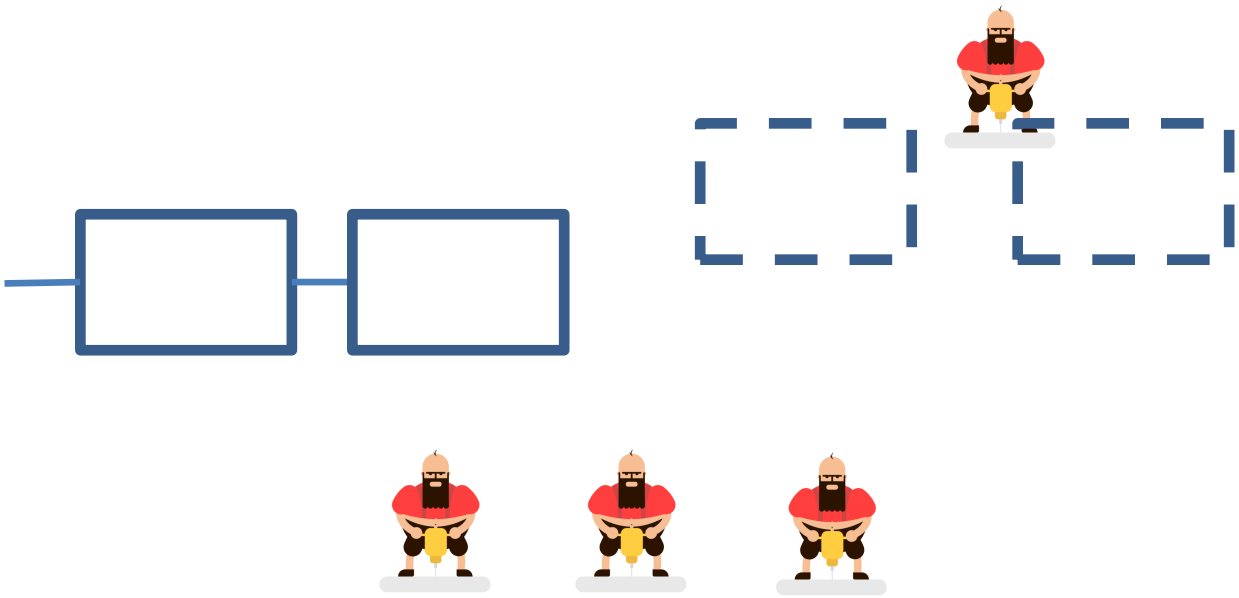
I. Eyal and E. Gun Sirer, Majority is not enough: Bitcoin mining is vulnerable,
Financial Cryptography 2014



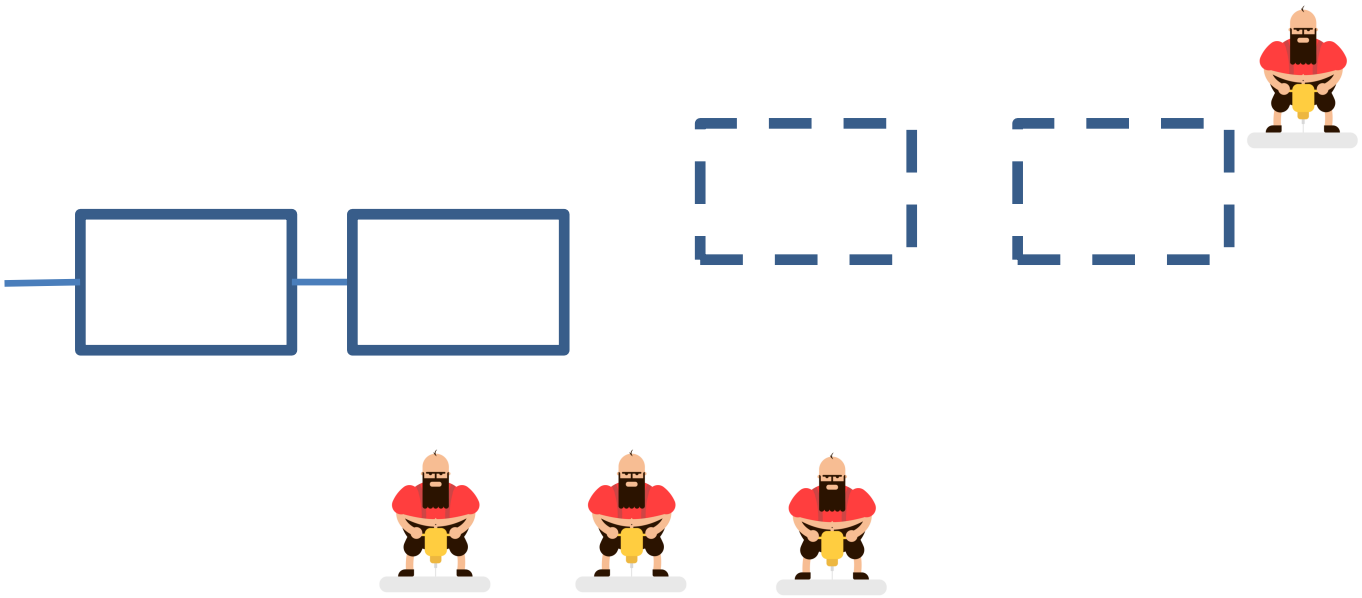
idea: selfishly decide when to announce solved blocks



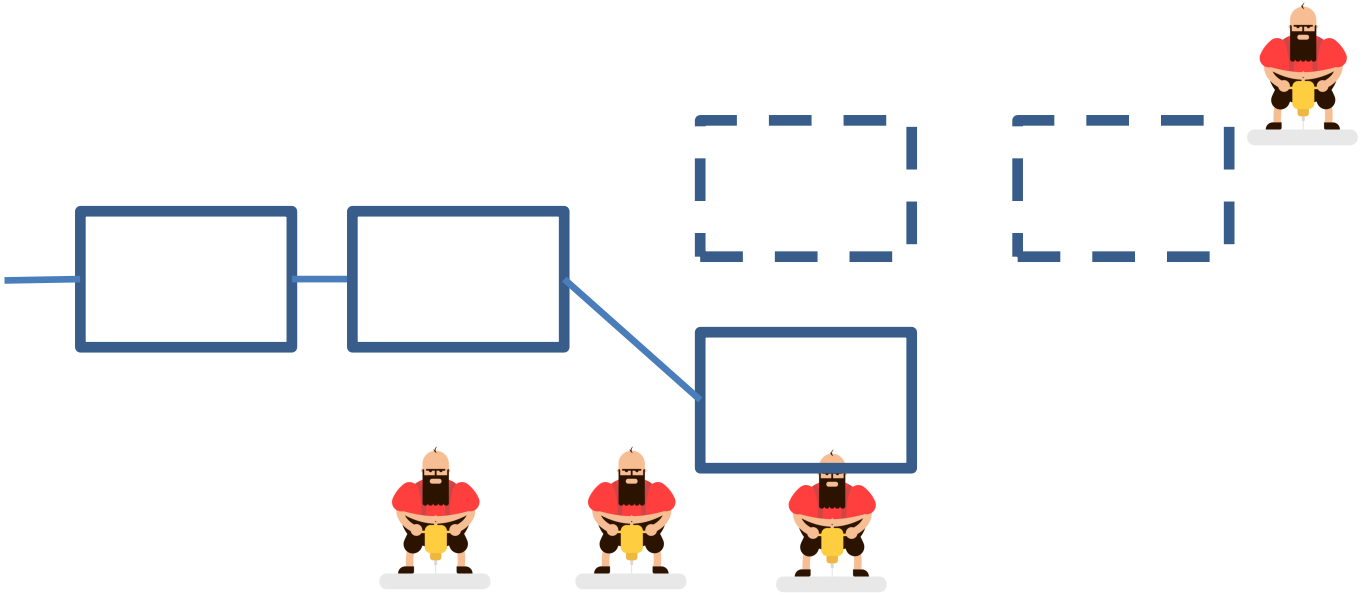
idea: selfishly decide when to announce solved blocks



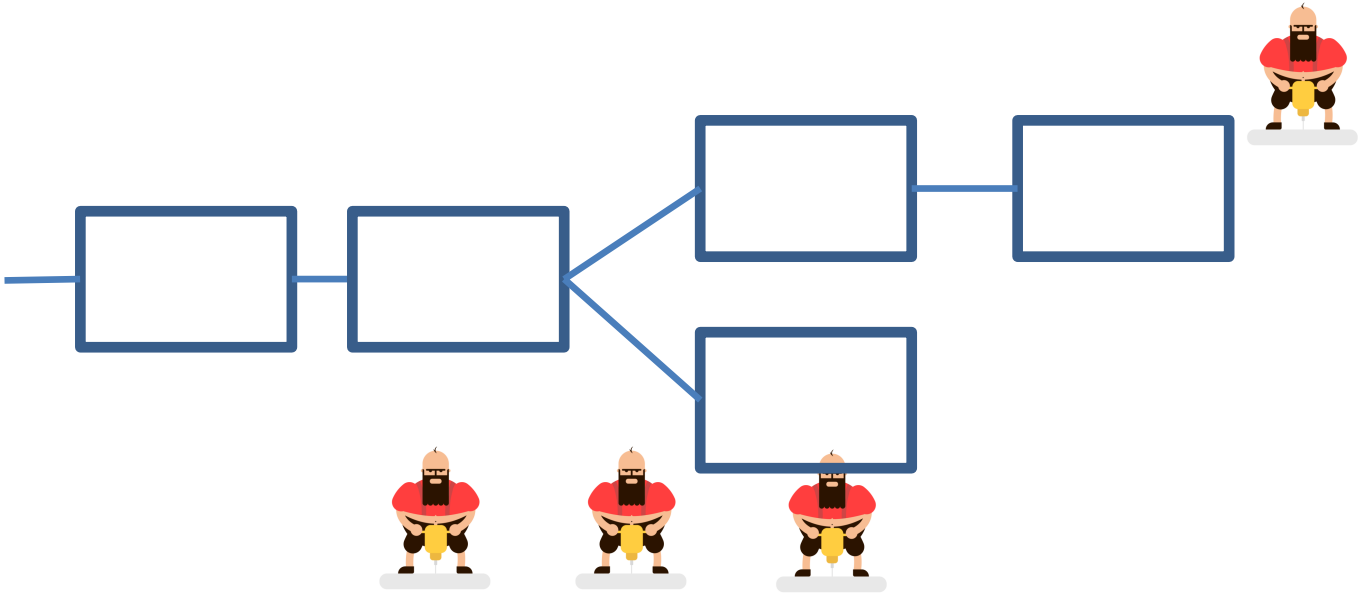
idea: selfishly decide when to announce solved blocks



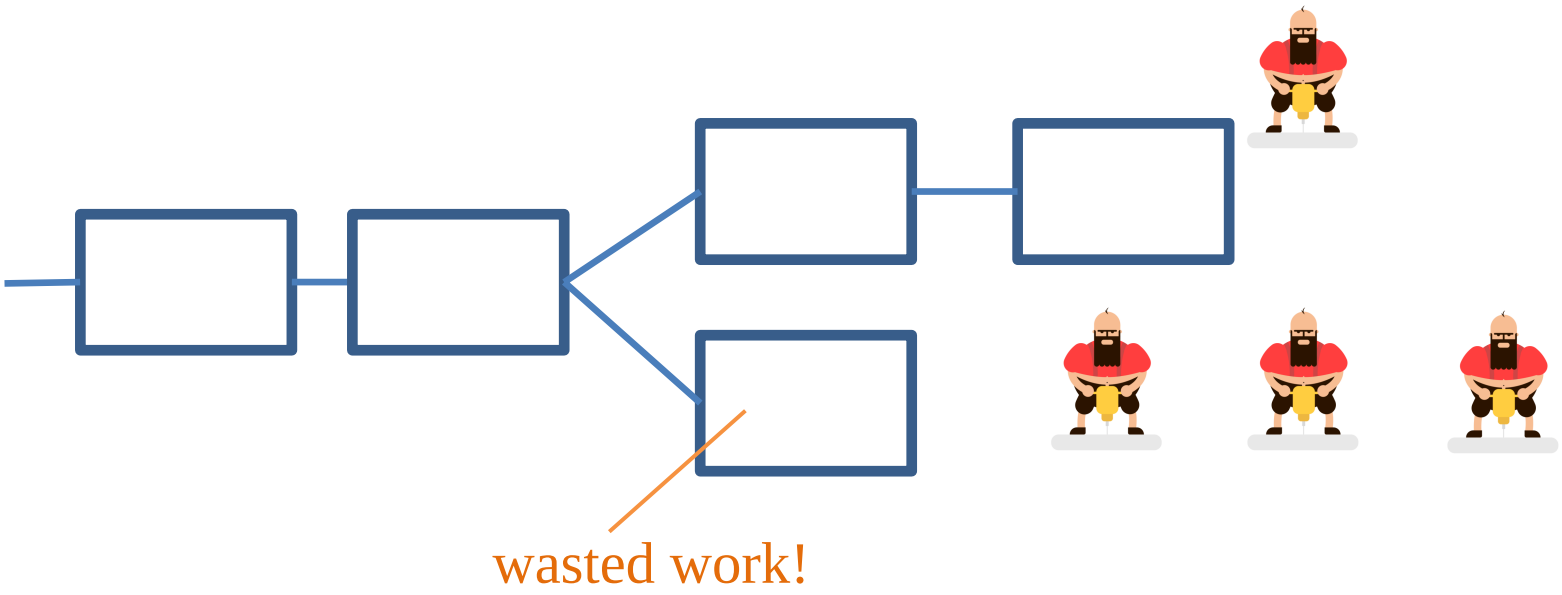
idea: selfishly decide when to announce solved blocks



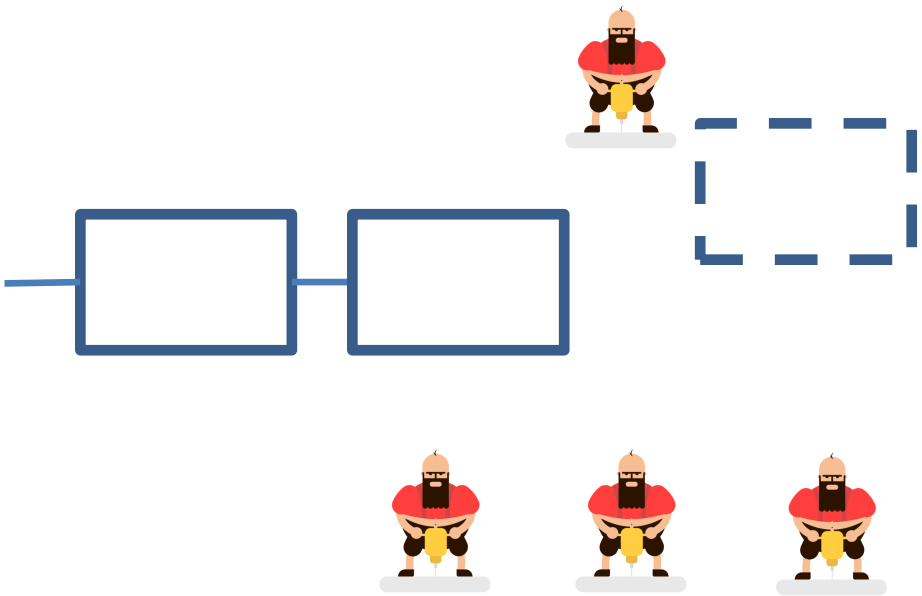
idea: selfishly decide when to announce solved blocks



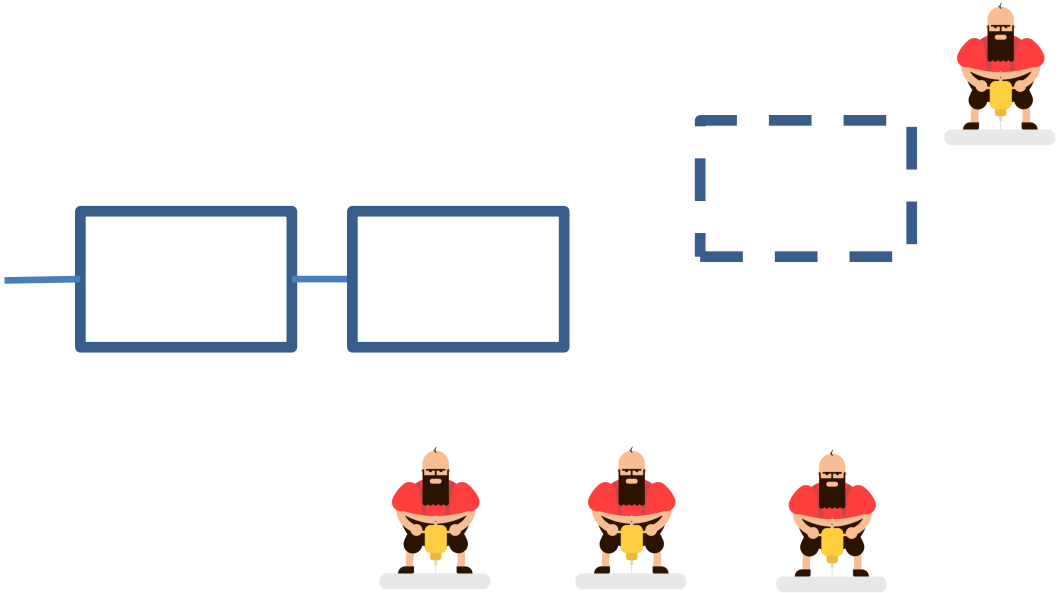
idea: selfishly decide when to announce solved blocks



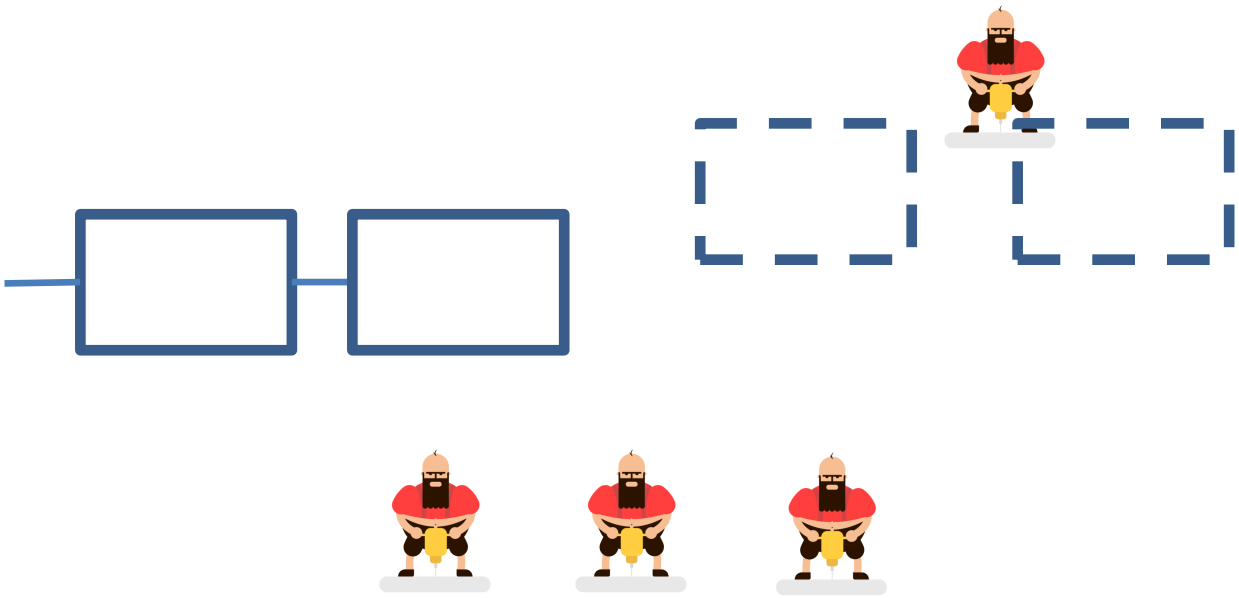
idea: selfishly decide when to announce solved blocks



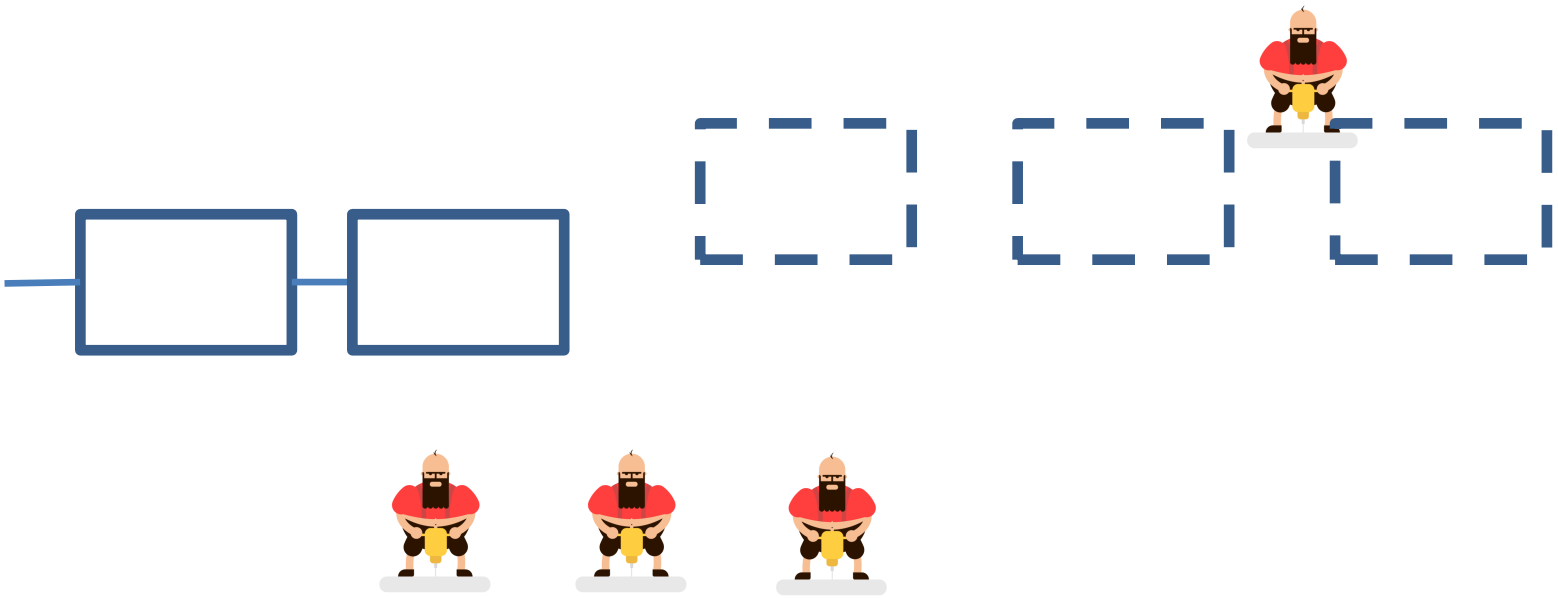
idea: selfishly decide when to announce solved blocks



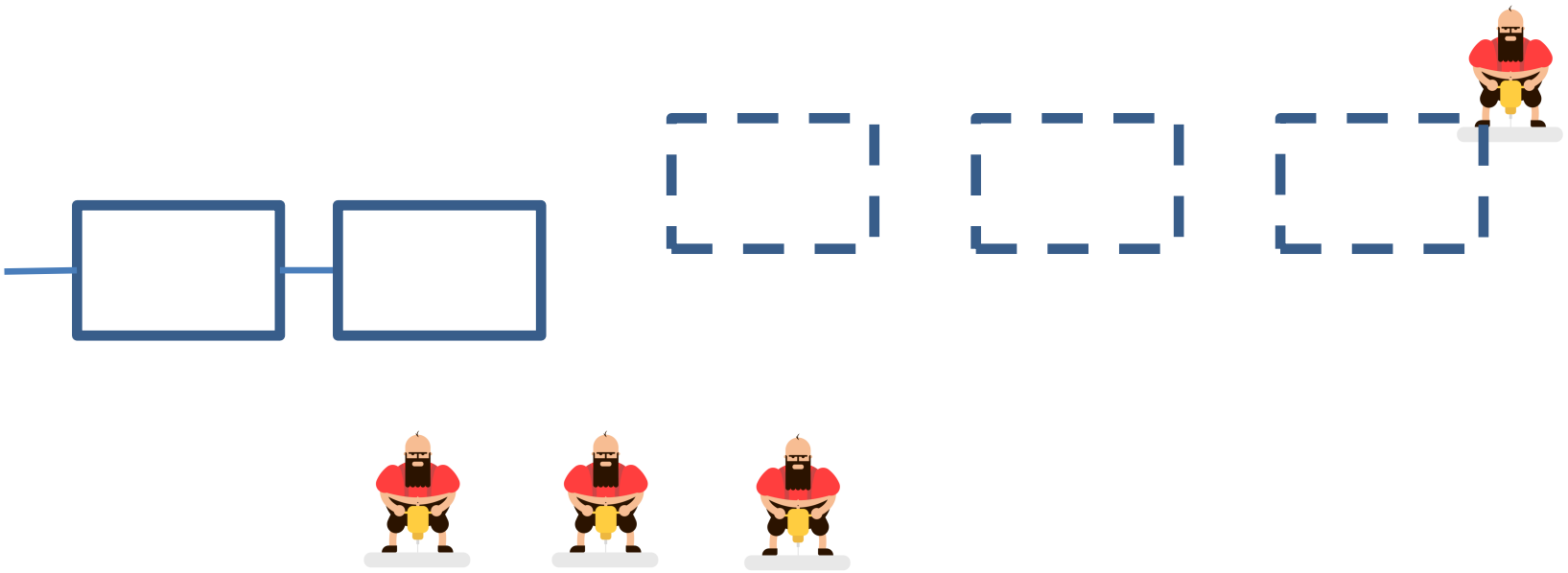
idea: selfishly decide when to announce solved blocks



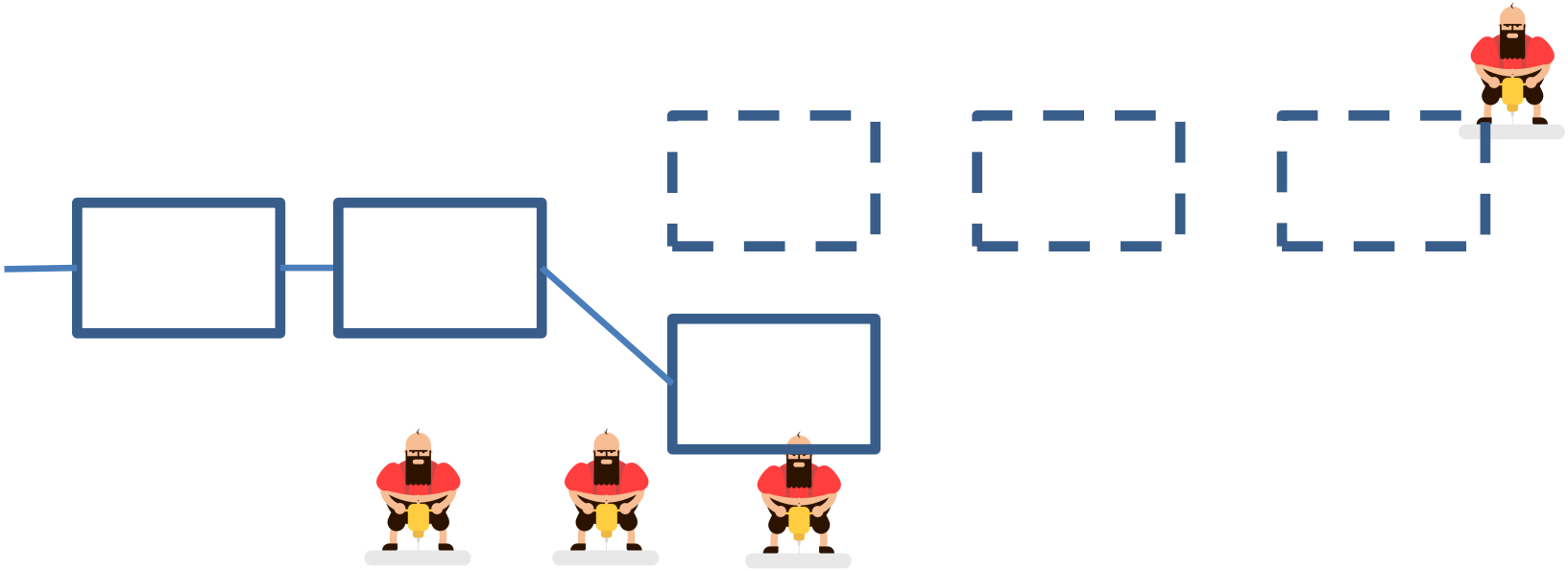
idea: selfishly decide when to announce solved blocks



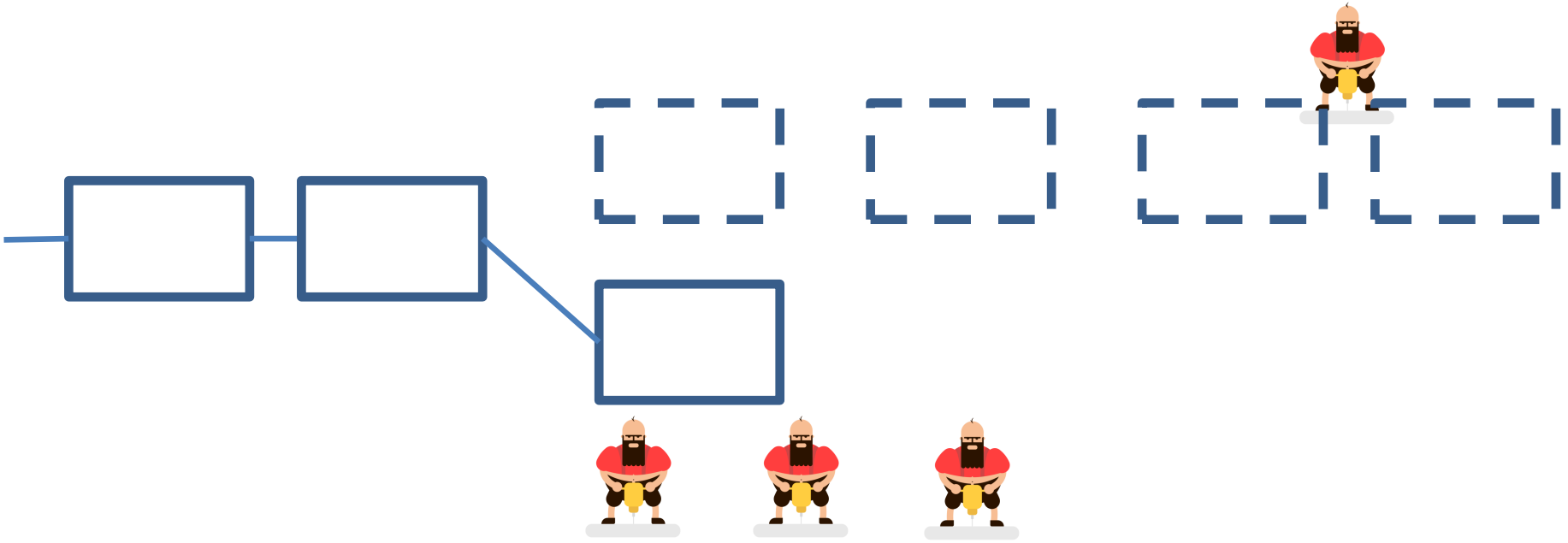
idea: selfishly decide when to announce solved blocks



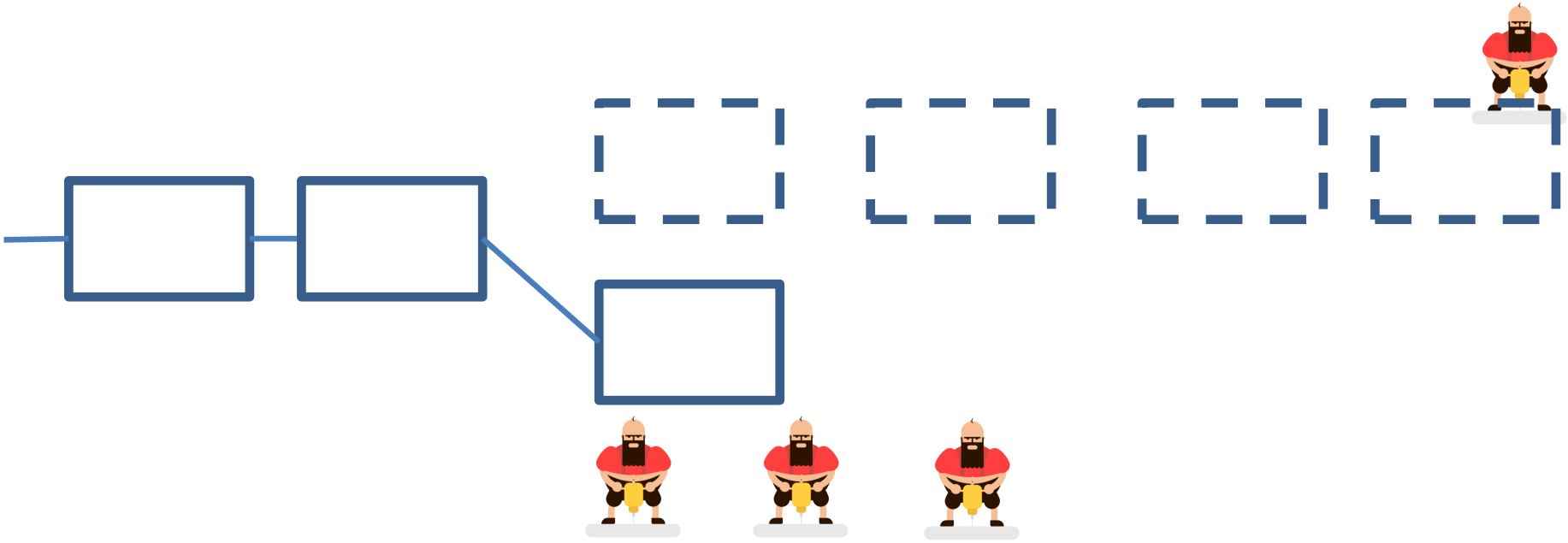
idea: selfishly decide when to announce solved blocks



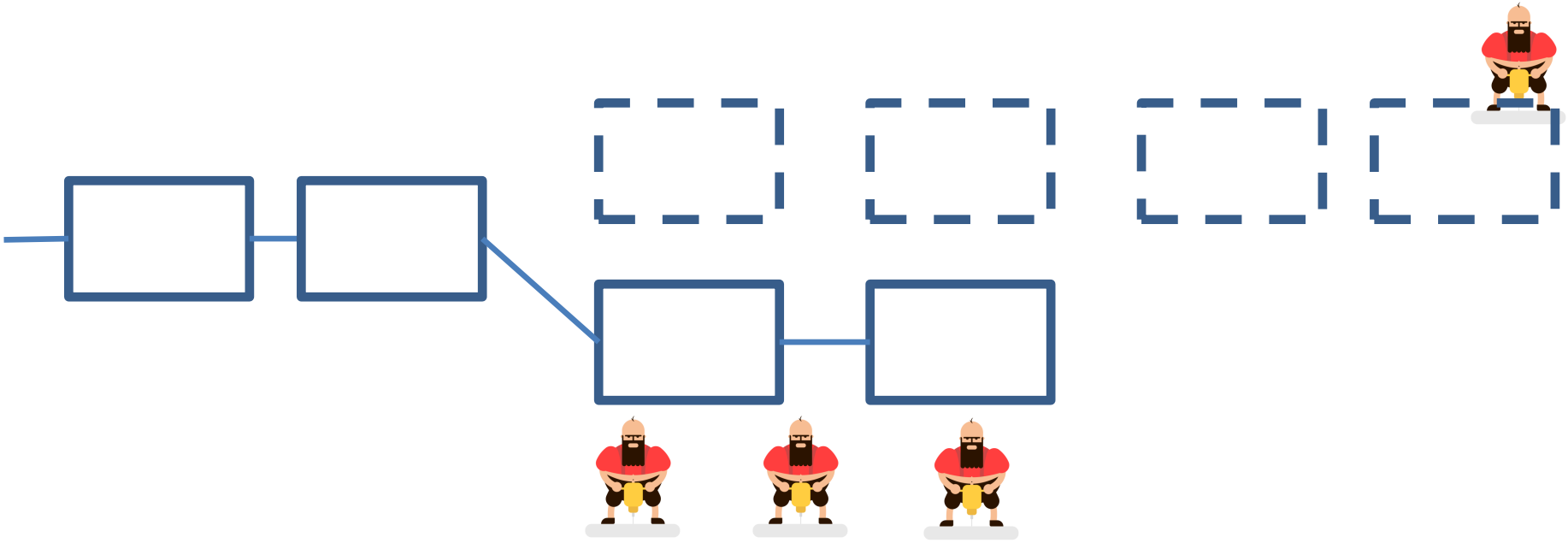
idea: selfishly decide when to announce solved blocks



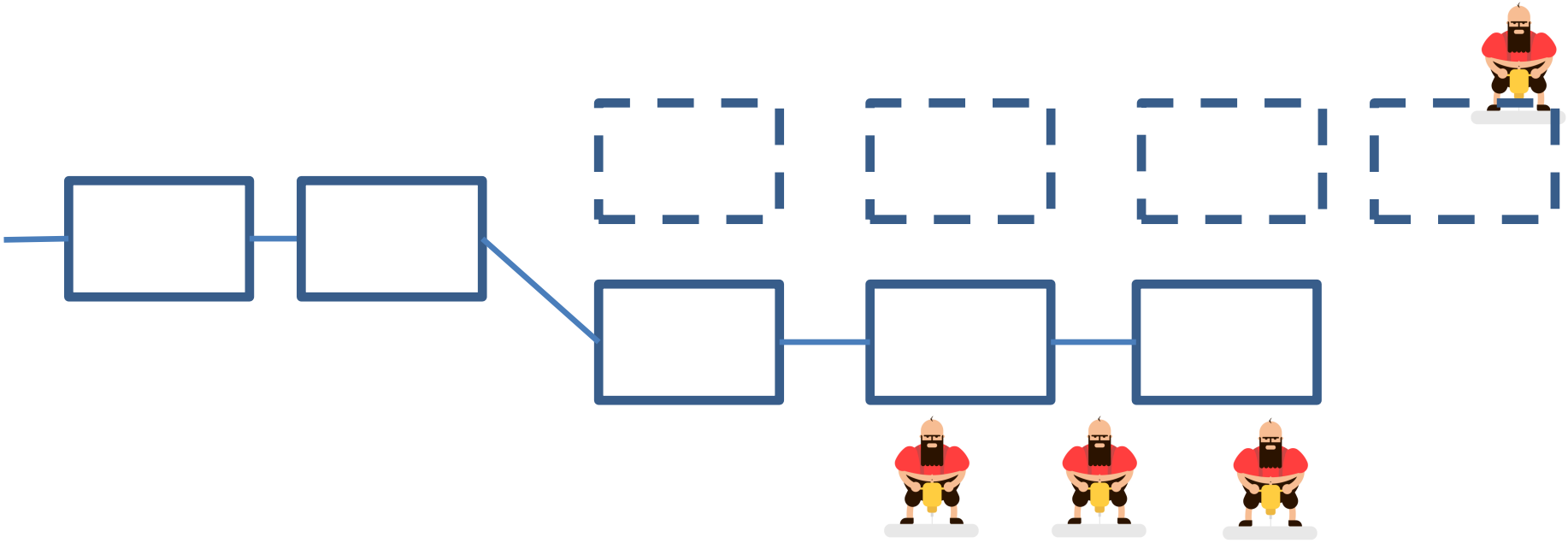
idea: selfishly decide when to announce solved blocks



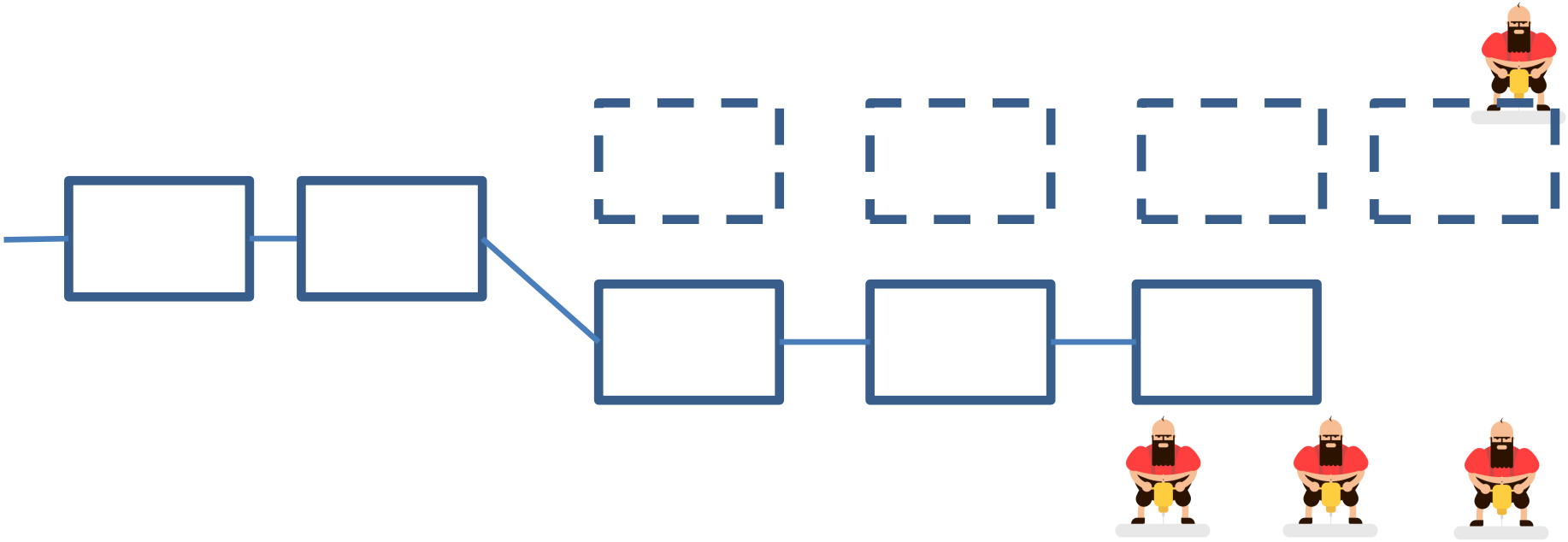
idea: selfishly decide when to announce solved blocks



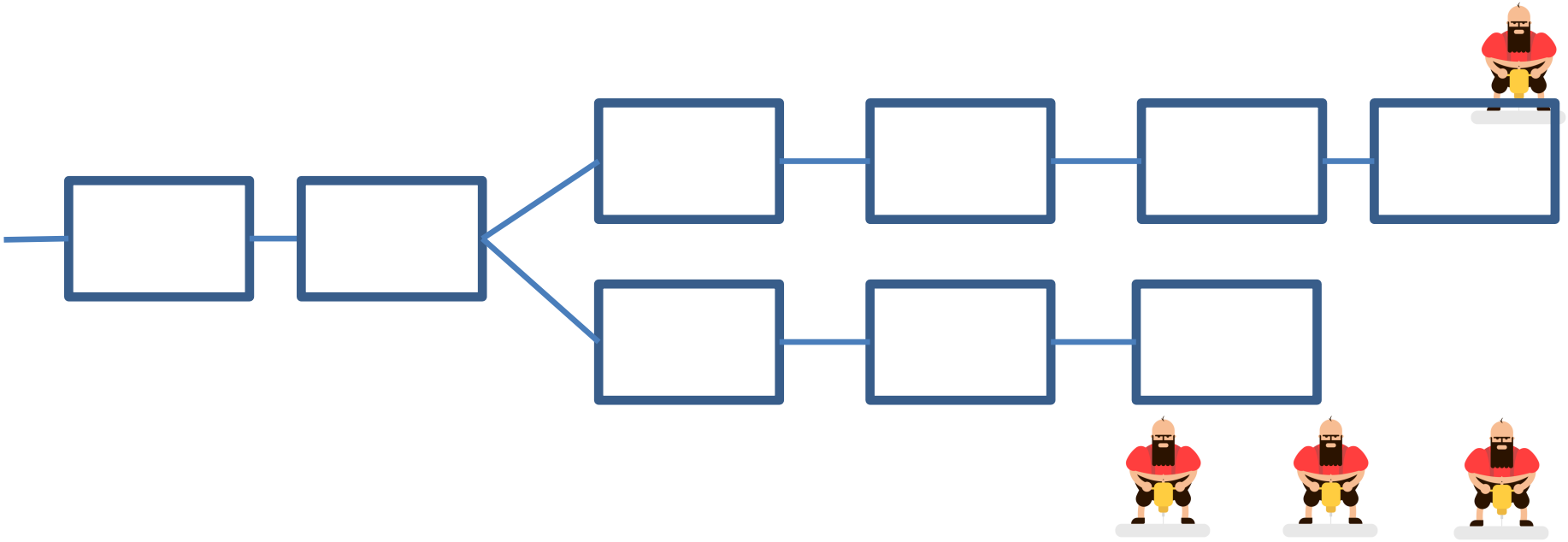
idea: selfishly decide when to announce solved blocks



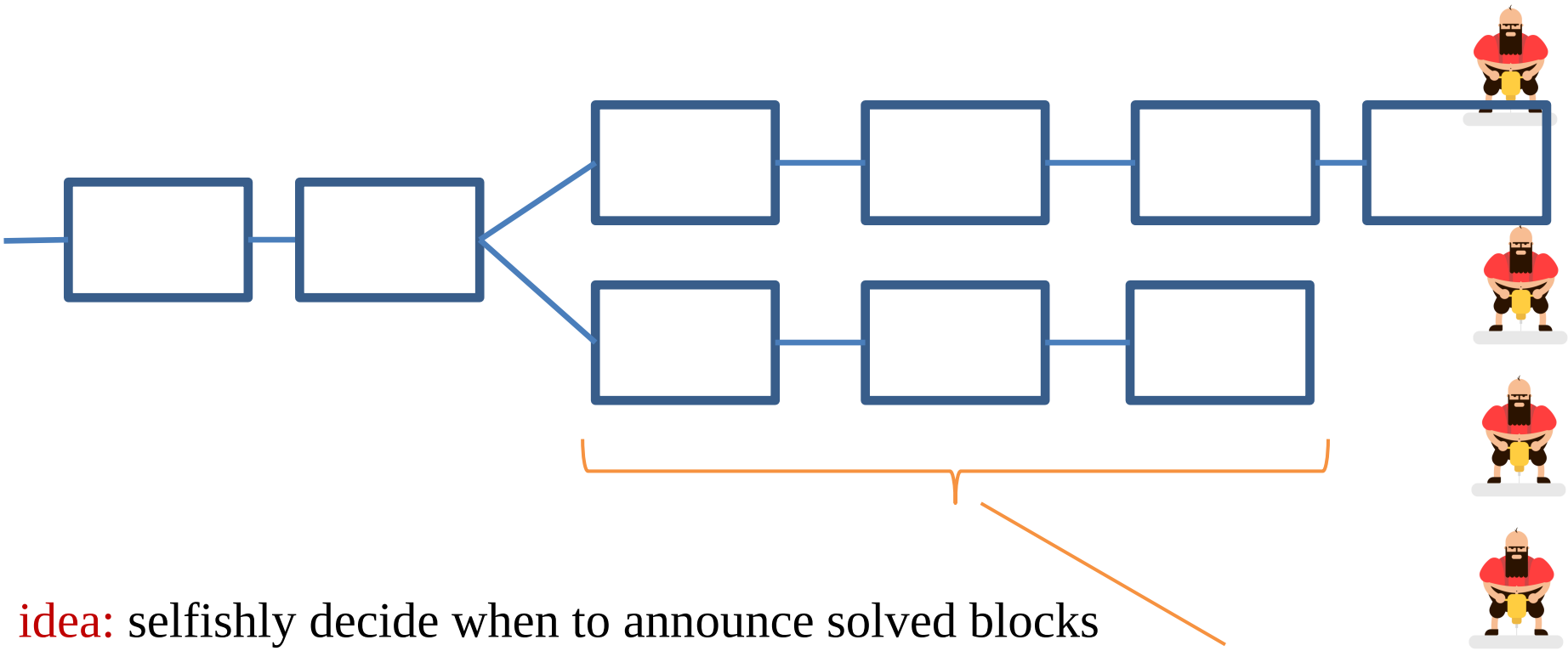
idea: selfishly decide when to announce solved blocks



idea: selfishly decide when to announce solved blocks

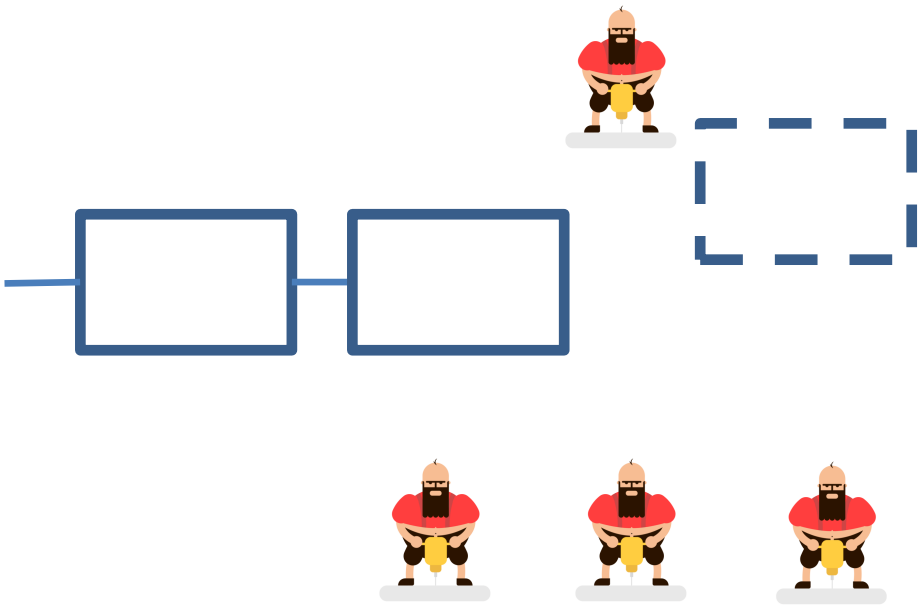


idea: selfishly decide when to announce solved blocks

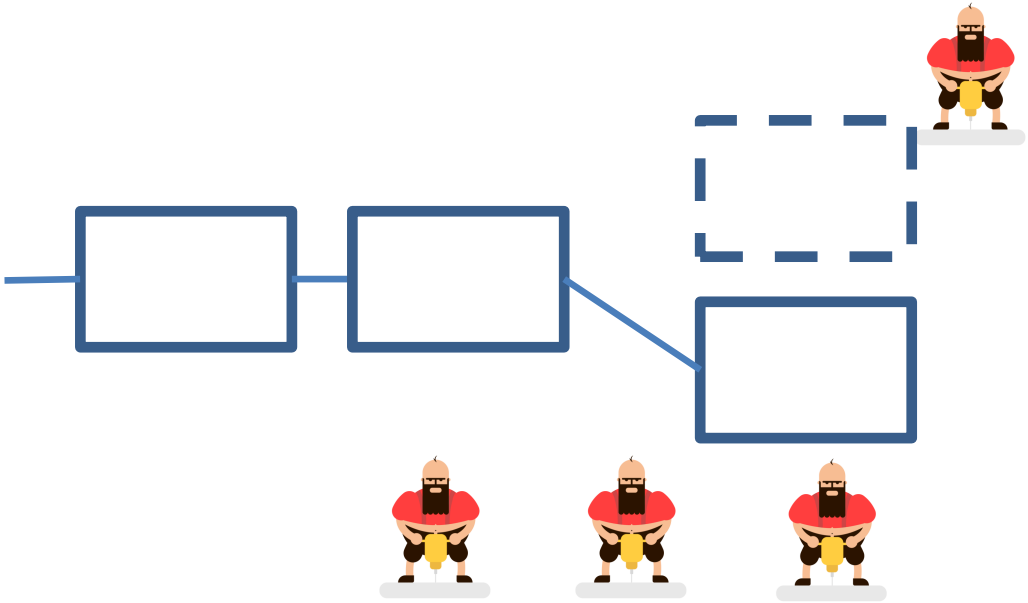


idea: selfishly decide when to announce solved blocks

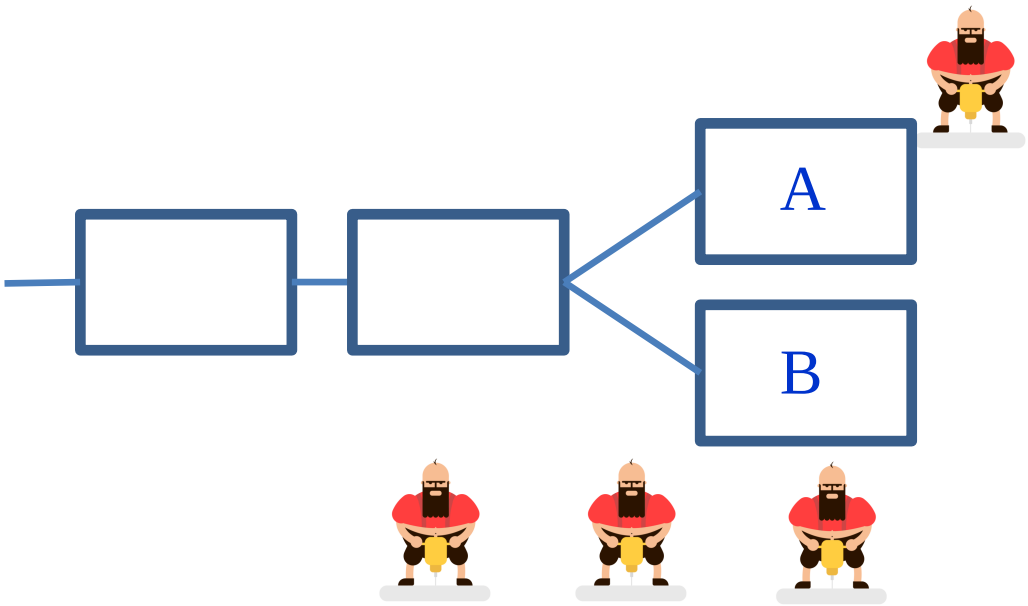
wasted work!



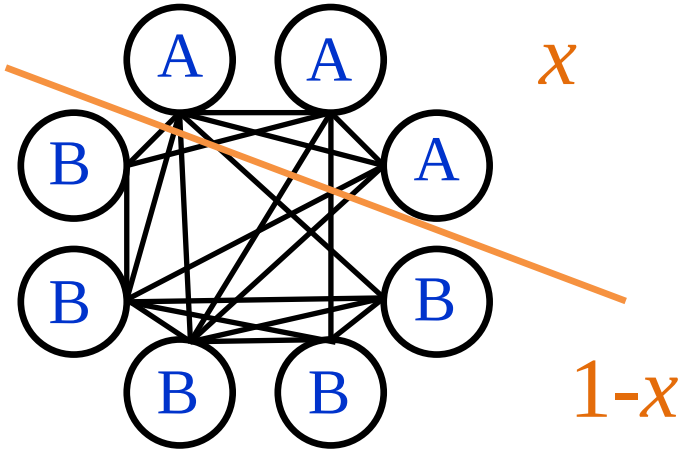
idea: selfishly decide when to announce solved blocks

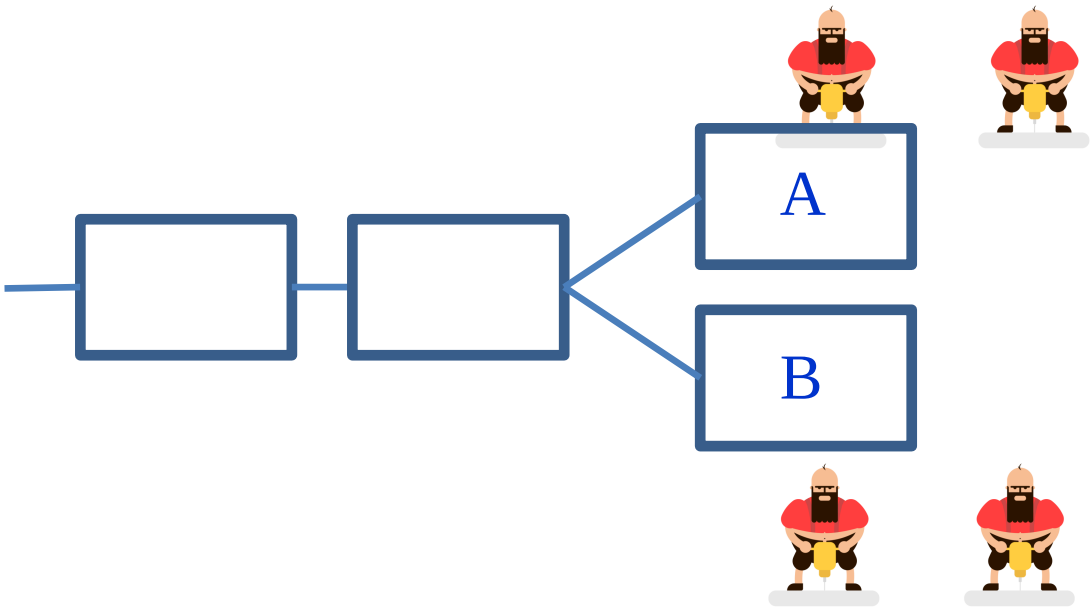


idea: selfishly decide when to announce solved blocks

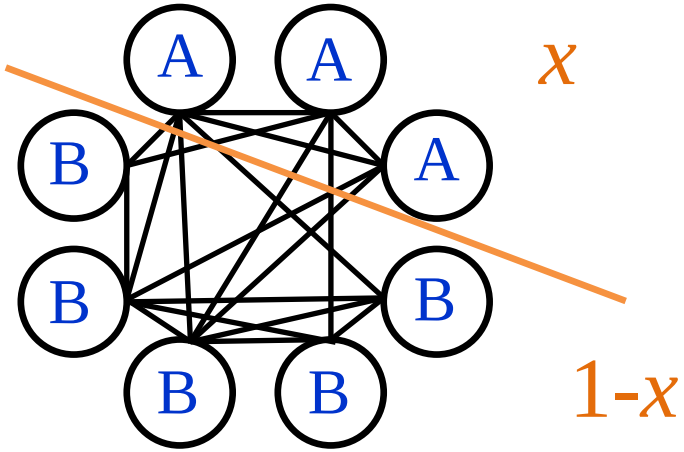


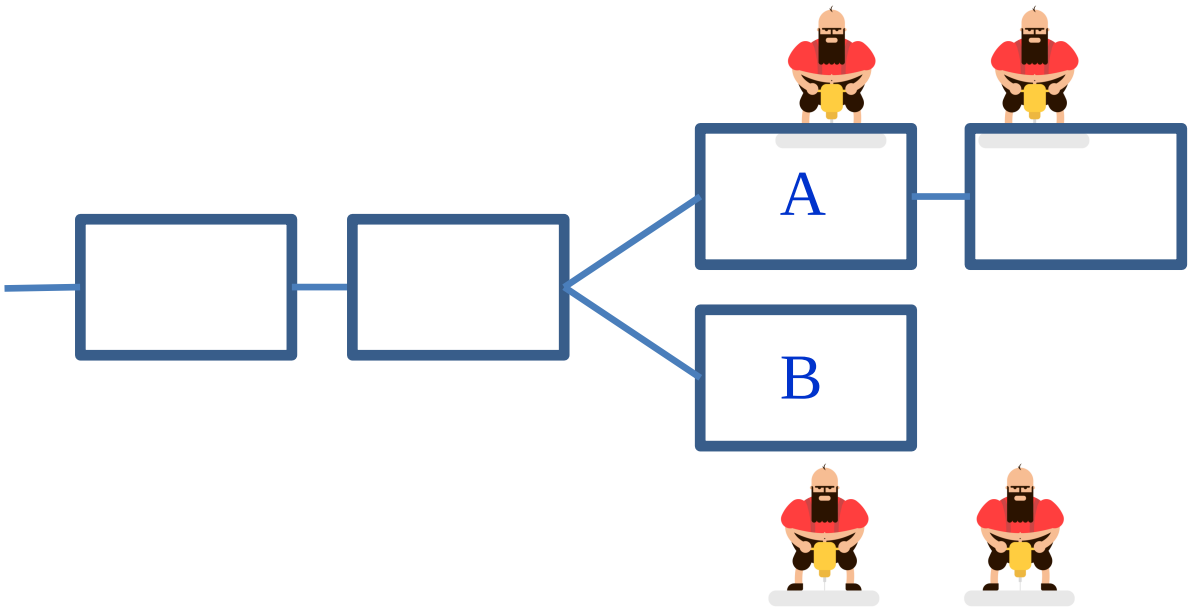
idea: selfishly decide when to announce solved blocks



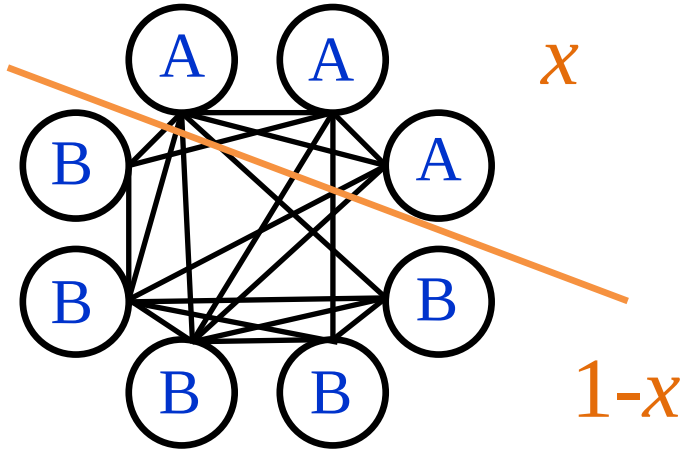


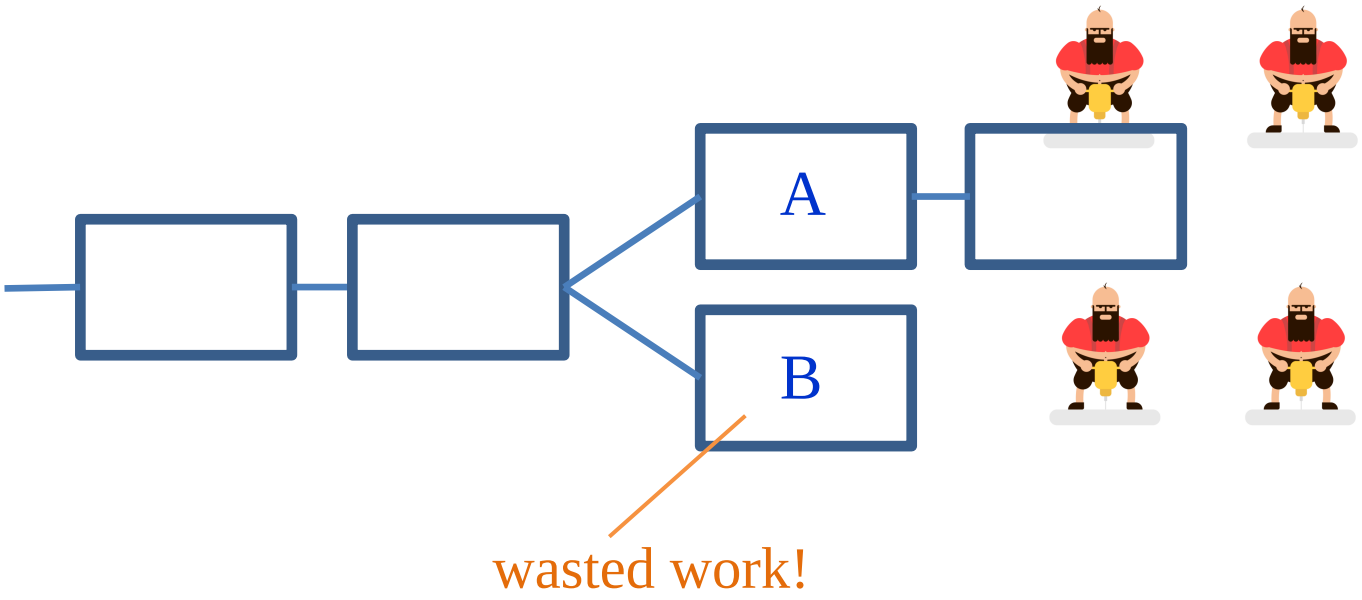
idea: selfishly decide when to announce solved blocks



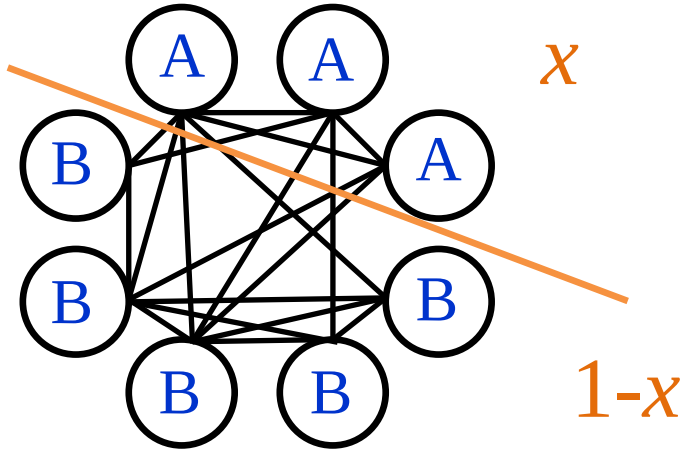


idea: selfishly decide when to announce solved blocks

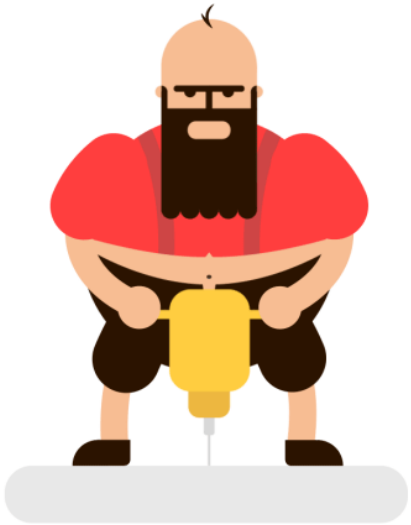




idea: selfishly decide when to announce solved blocks



idea:



Convenient if:

- ≥1/3 of the total computational power
- x≥1/2 & 1/4 of the total computational power

B

B

B

B

1-*x*


Thank you for
your attention!