Kaj sestavlja Blockchain?

FIN-TECH Risk management delavnica

Matjaž Krnc

FINTECH RISK MANAGEMENT
www.fintech-ho2020.eu

Università del Litorale Univerza na Primorskem

famnit

Fakulteta za informacijske študije
Faculty of information studies

Ingredient #1:

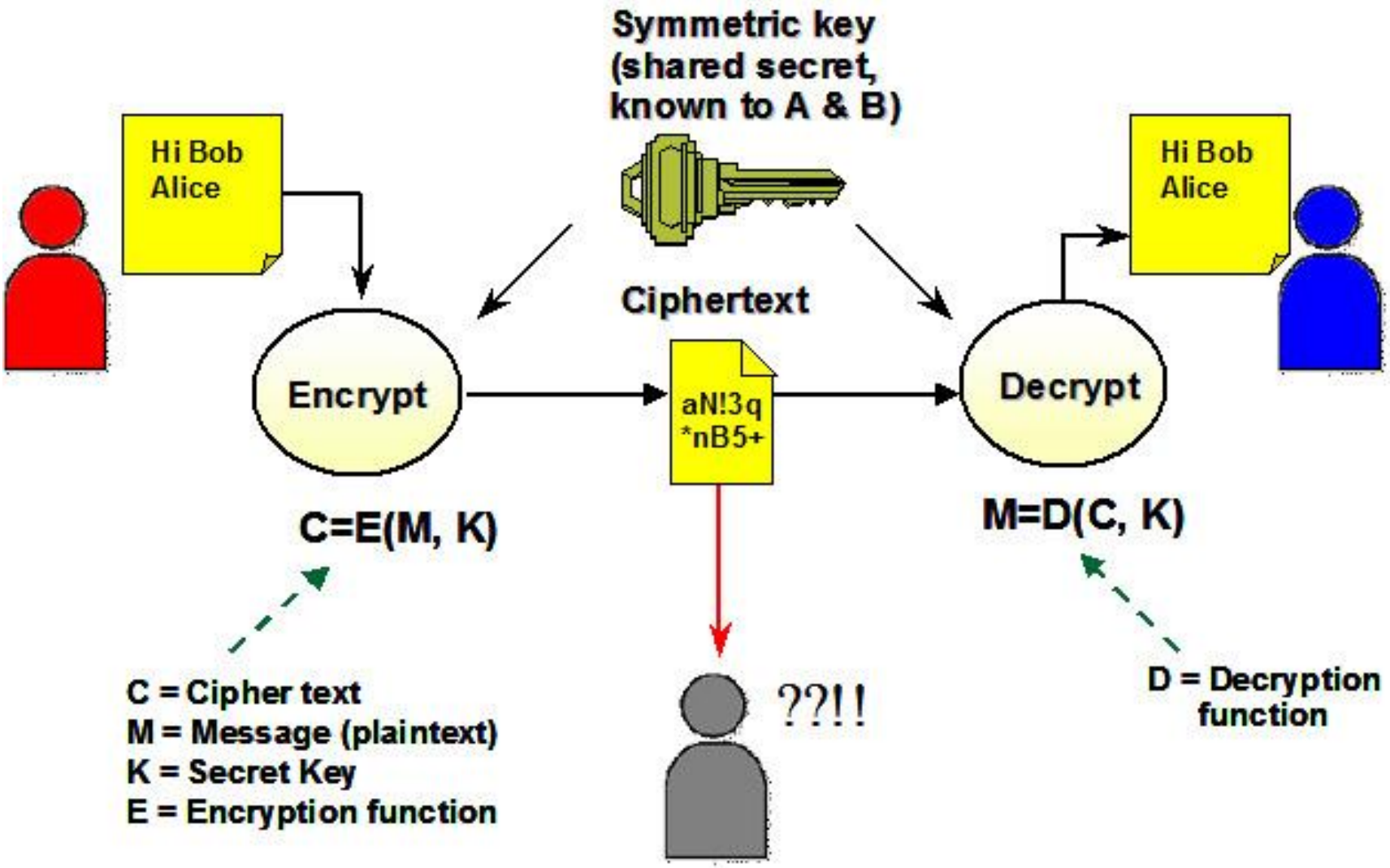
Cryptography 101

What is Cryptography?

- From Greek, meaning “secret writing”
- Confidentiality: encrypt data to hide content
- Include “signature” or “message authentication code”
  - Integrity: Message has not been modified
  - Authentication: Identify source of message

- Modern encryption:
  - *Algorithm* public, *key* secret and provides security
  - Symmetric (shared secret) or asymmetric (public-private key)

Symmetric Cipher Model



Symmetric (Secret Key) Crypto

- Sender and recipient share common key
- Provides dual use:
  - Confidentiality (encryption)
  - Message authentication + integrity
- 1000x more computationally efficient than asymmetric
- **Main challenge: How to distribute the key?**

Public-Key Cryptography

- **Each party has (public key, private key)**

- **Alice's public key PK**
  - Known by anybody
  - Bob uses PK to encrypt messages *to* Alice
  - Bob uses PK to verify signatures *from* Alice

- **Alice's private/secret key: sk**
  - Known only by Alice
  - Alice uses sk to decrypt ciphertexts sent to her
  - Alice uses sk to generate new signatures on messages

Public-Key Cryptography

- (PK, sk) = generateKey(keysize)

- **Encryption API**
  - ciphertext = encrypt (message, PK)
  - message = decrypt (ciphertext, sk)

- **Digital signatures API**
  - Signature = sign (message, sk)
  - isValid = verify (signature, message, PK)

(Simple) RSA Algorithm

(Simple) RSA Algorithm

- Generating a key:
  - Generate composite **n = p * q**, where p and q are secret primes
  - Pick public exponent **e**
  - Solve for secret exponent **d** in $d \cdot e \equiv 1 \pmod{(p-1)(q-1)}$
  - Public key = (e, n), private key = d

- Encrypting message m: $c = m^e \bmod n$
- Decrypting ciphertext c: $m = c^d \bmod n$

- **Security** due to cost of factoring large numbers
  - For an b-bit value **n**, finding **(p,q)** takes $O\left(\exp \sqrt[3]{\frac{64}{9} b (\log b)^2}\right).$
  - We choose n to be 2048 or 4096 bits long.

Ingredient #2:

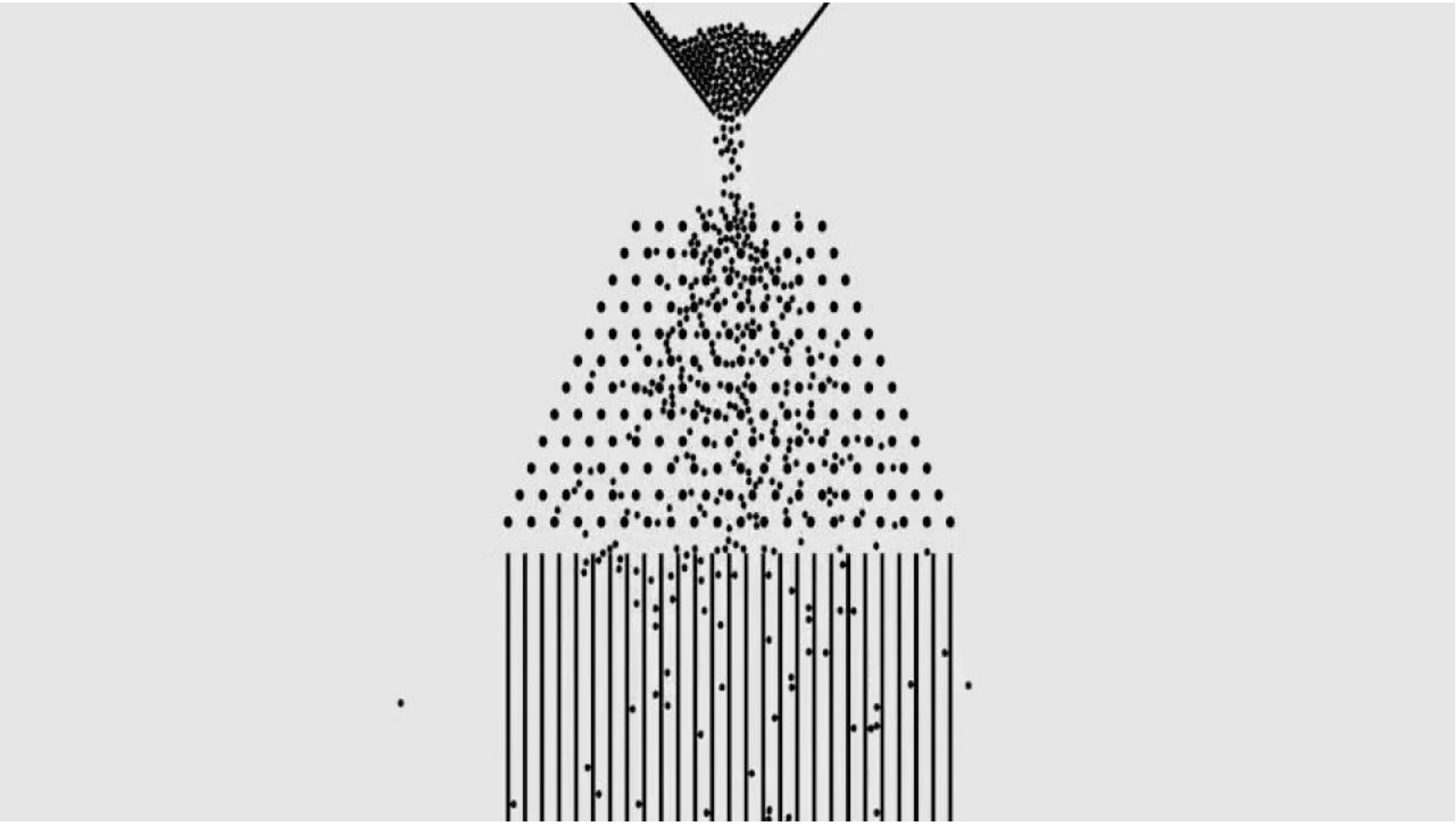
Cryptographic hash function

Cryptography Hash Functions I

- Take message *m* of arbitrary length and produces fixed-size (short) number *H(m)*

- One-way function
  - Efficient: Easy to compute *H(m)*
  - **Hiding property:** Hard to find an *m*, given *H(m)*
    - Assumes “m” has sufficient entropy, not just {“heads”, “tails”}
  - **Random:** Often assumes for output to “look” random

Cryptography Hash Functions I



Cryptography Hash Functions II

- Collisions exist: | possible inputs | >> | possible outputs |

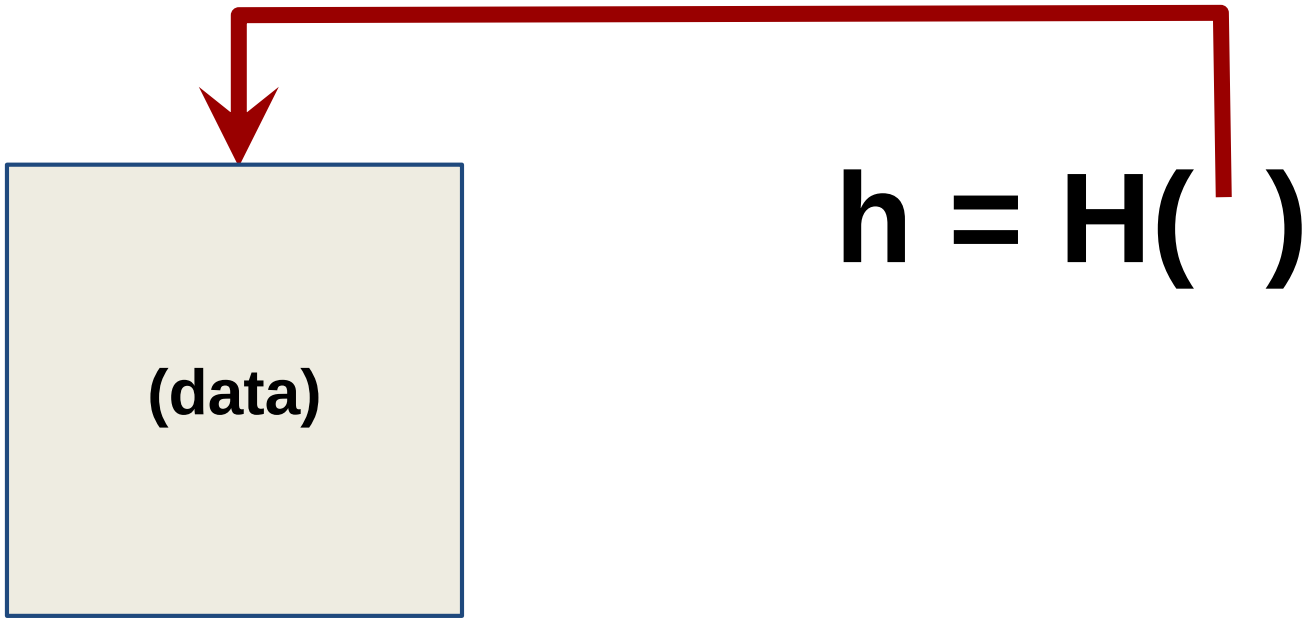
  … but hard to find

- Collision resistance:
  - Find any m != m' such that H(m) == H(m')
  - (harder) Given m, find m' such that H(m) == H(m')
  - For 160-bit hash (SHA-1)
    - Finding any collision is birthday paradox: 2^{160/2} = 2^80
    - Finding specific collision requires 2^160

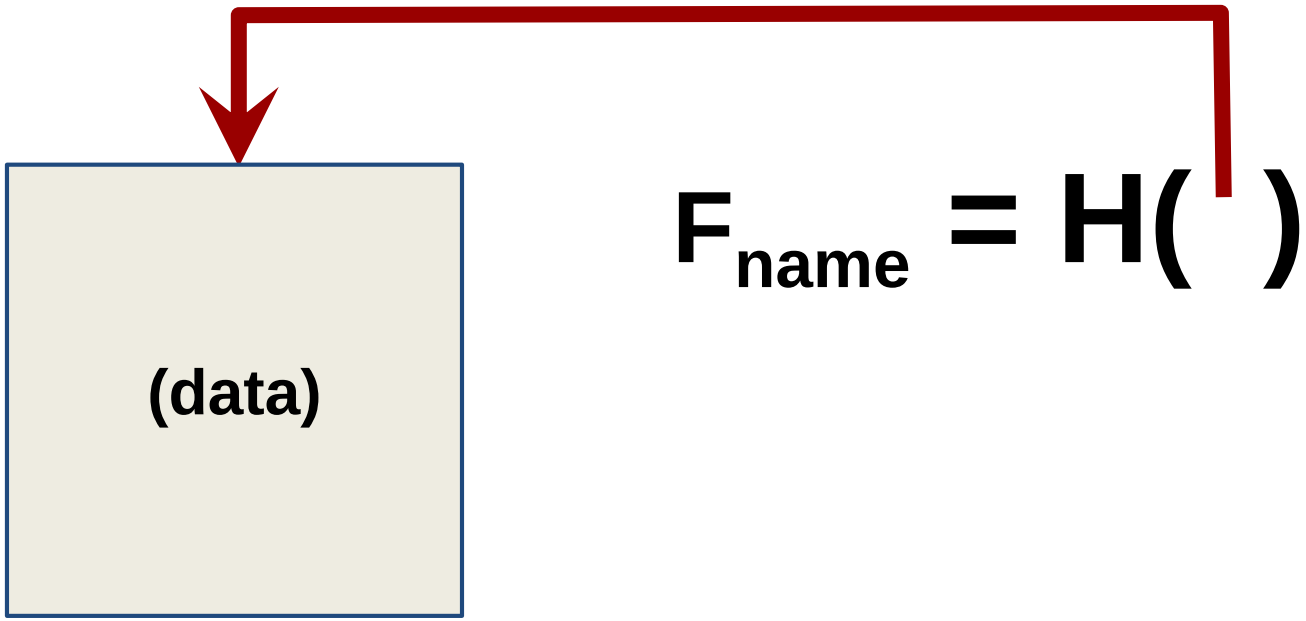
Example use: Passwords

- Can't store passwords in a file that could be read
  - Concerned with insider attacks / break-ins
- Must compare typed passwords to stored passwords
  - Does H (input) == H (password) ?
- Memory cheap: build table of all likely password hashes?
  - Use "salt" to compute h = H (password || salt)
  - Store salt as plaintext in password file, not a secret
  - Then check whether H (input, salt) == h

Example use: Hash Pointers

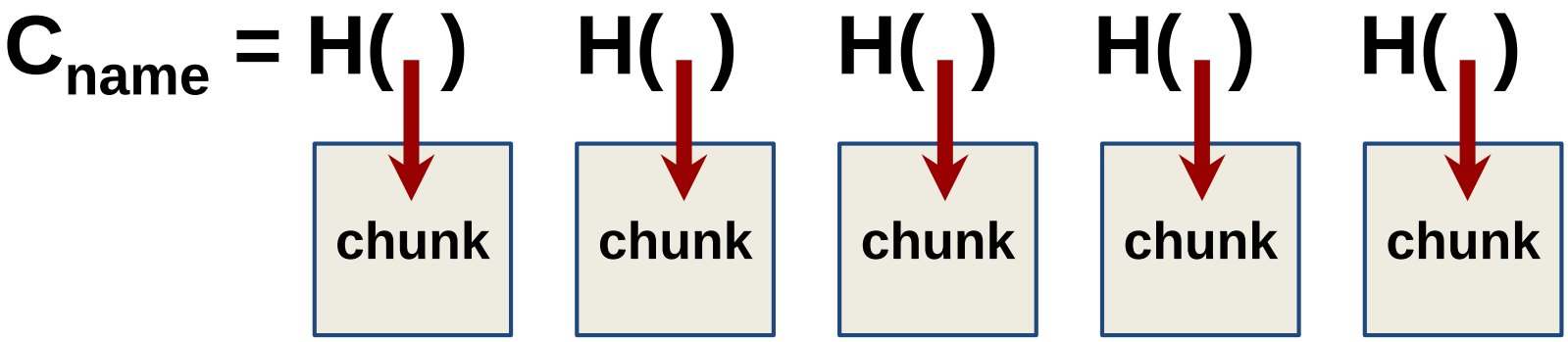


Example Use: Self-certifying names



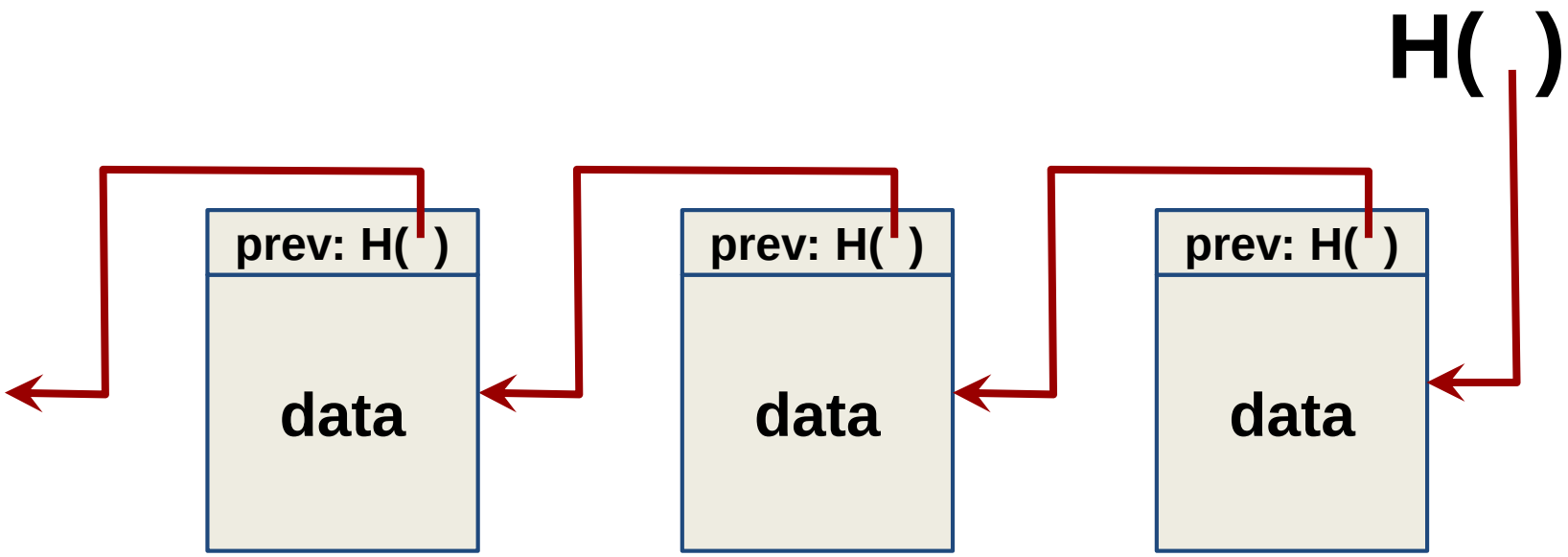
- P2P file sharing software (e.g., Limewire)
  - File named by $F_{name} = H(data)$
  - Participants verify that $H(downloaded) == F_{name}$

Self-certifying names



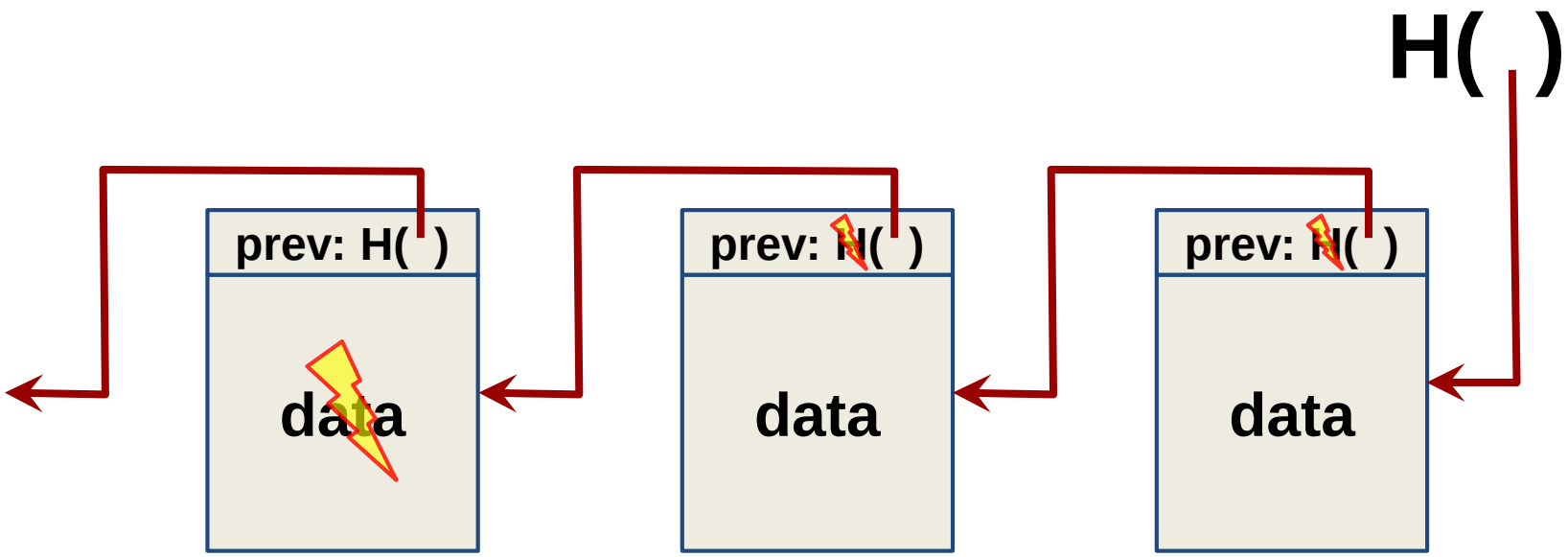
- **Another example**: BitTorrent
  - Large file split into smaller chunks (~256KB each)
  - Torrent file specifies the name/hash of each chunk
  - Participants verify that H (downloaded) == C_{name}
  - Security relies on getting torrent file from trustworthy source

Main Example: Hash chains



Creates a “tamper-evident” log of data

Hash chains



If data changes, all subsequent hash pointers change

Otherwise, found a hash collision!

Hash chain application – Blockchain

- New bitcoins are “created” every ~10 min, owned by “miner” (more on this later)
- Thereafter, just keep record of transfers
  - e.g., Alice pays Bob 1 BTC
- Basic protocol:
  - Alice signs transaction: $txn = Sign_{Alice}(BTC, PK_{Bob})$
  - Alice shows transaction to others…

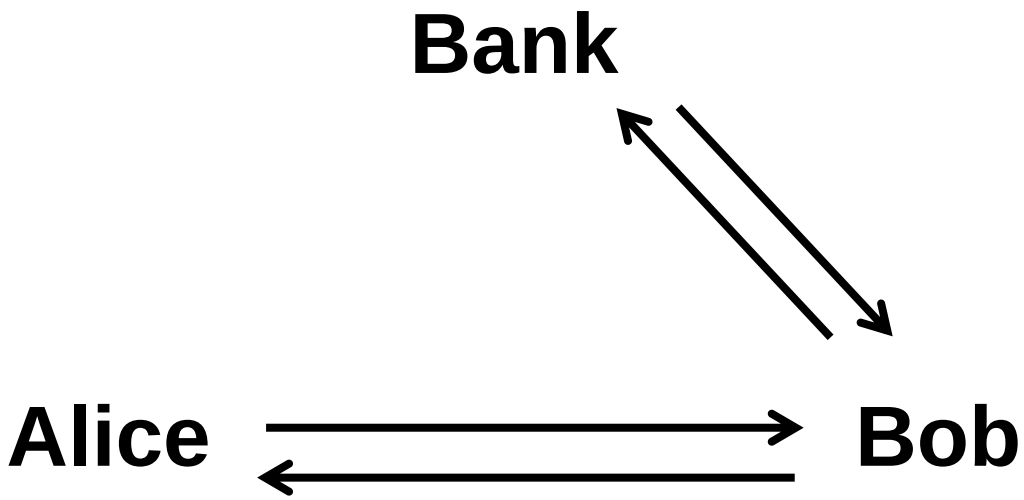
Ingredient #3:

Avoiding Equivocation!

Can Alice “pay” both Bob and Charlie with same bitcoin ?

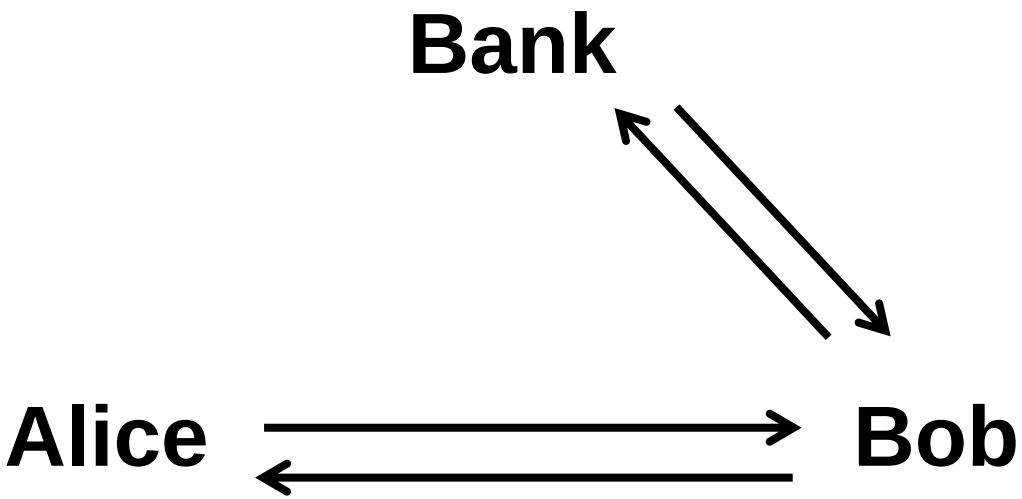
(Known as “double spending”)

How traditional e-cash handled problem



- When Alice pays Bob with a coin, Bob validates that coin hasn't been spend with trusted third party
- Introduced "blind signatures" and "zero-knowledge protocols" so bank can't link withdrawals and deposits

How traditional e-cash handled problem



- When Alice pays Bob with a coin, Bob validates that coin hasn't been spend with trusted third party

Bank maintains linearizable log of transactions

Ingredient #3:

Avoiding Equivocation!

Goal: No double-spending in decentralized environment

Main idea: Make transaction log

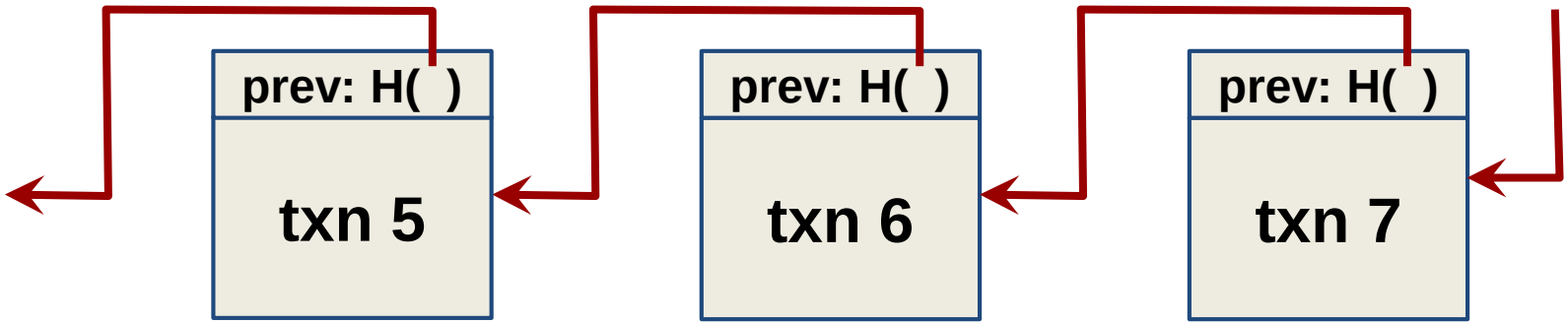
1. public
2. append-only
3. strongly consistent

Bitcoin (10,000 foot view)

- Public
  - Transactions are signed: $txn = Sign_{Alice}(BTC, PK_{Bob})$
  - All transactions are sent to all network participants

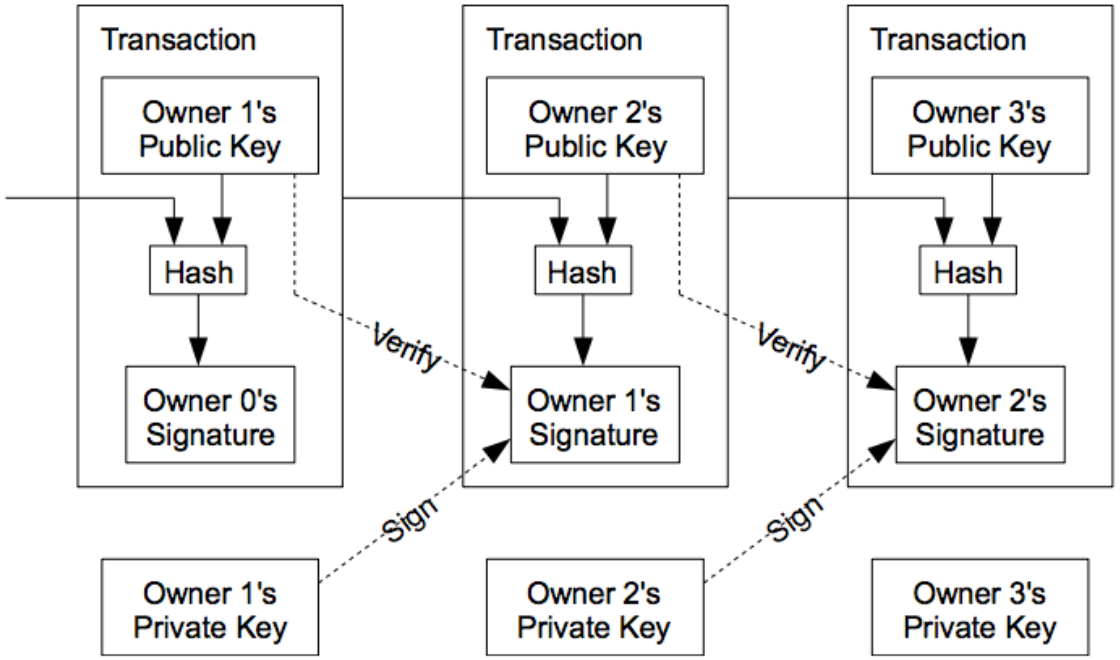
- No equivocation: Log append-only and consistent
  - All transactions part of a hash chain
  - Consensus on set/order of operations in hash chain

Blockchain: Append-only hash chain



- Recall: hash chain creates “tamper-evident” log of txns
- Security based on collision-resistance of hash function
  - Given m and h = hash(m), difficult to find m’ such that h = hash(m’) and m != m’

Blockchain: Append-only hash chain

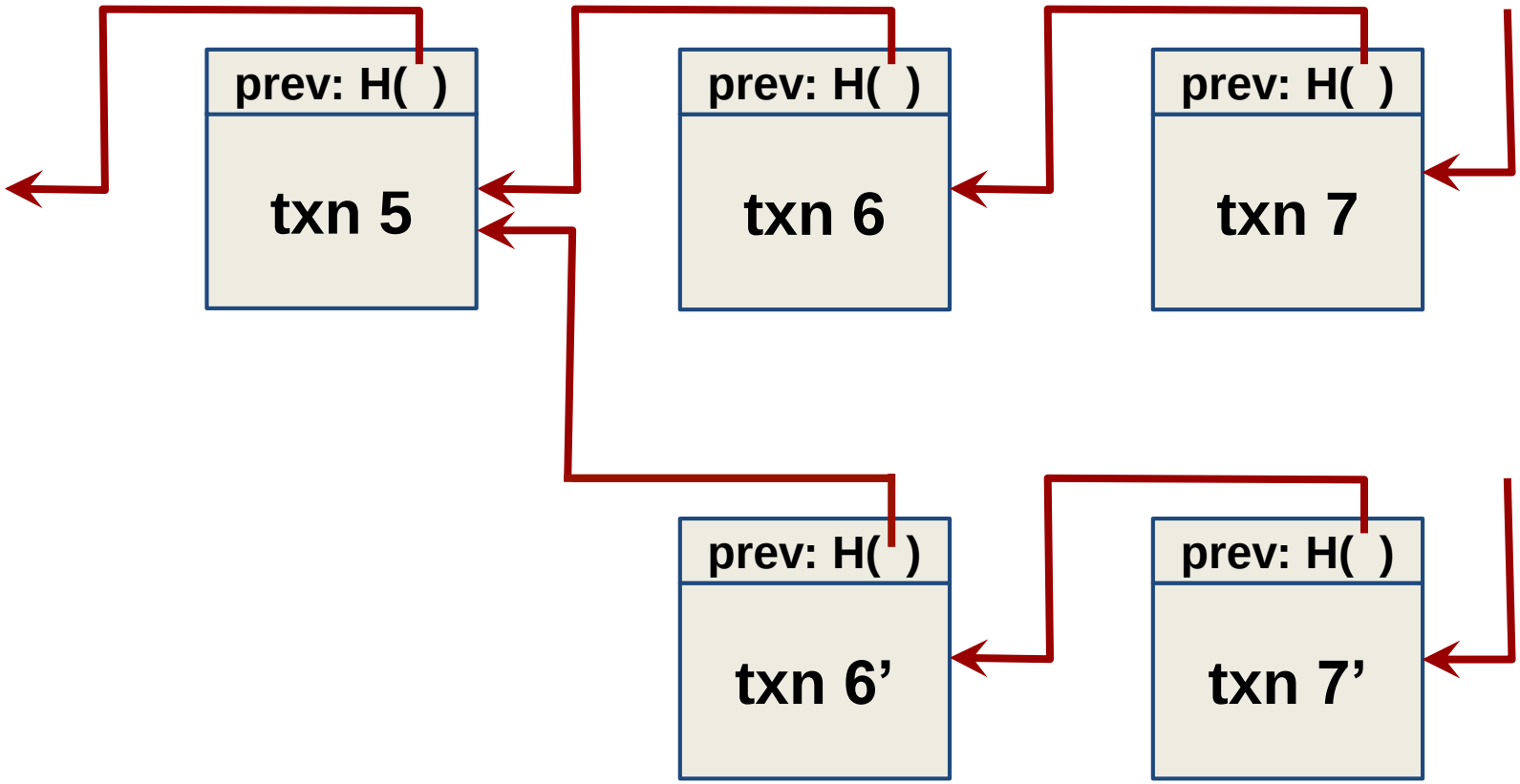


Bitcoin: A Peer-to-Peer Electronic Cash System

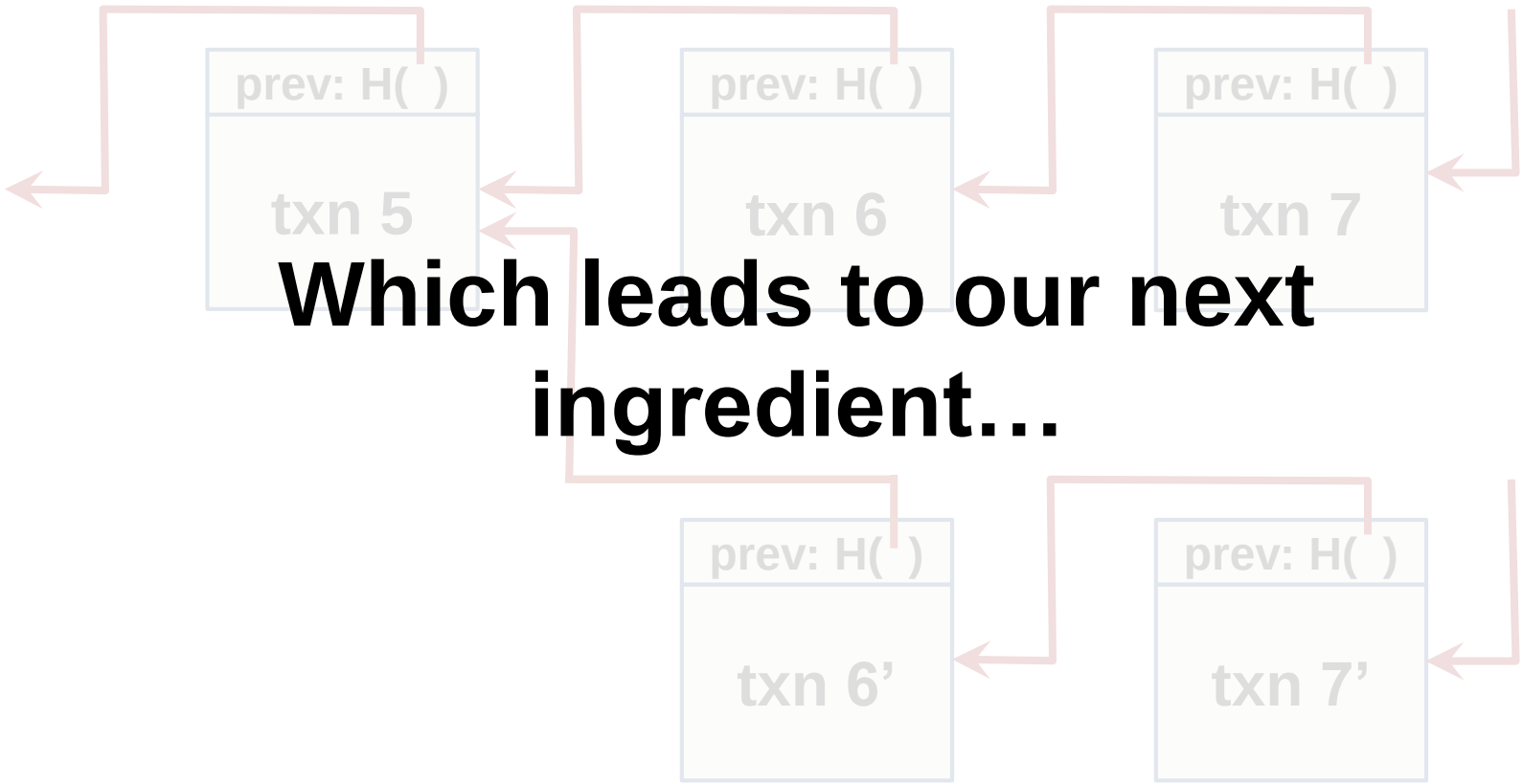
Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main

Problem remains: forking



Problem remains: forking



Which leads to our next ingredient…

Ingredient #4:

The Consensus Problem

Consensus doesn't happen by magic... You have to drive to it.

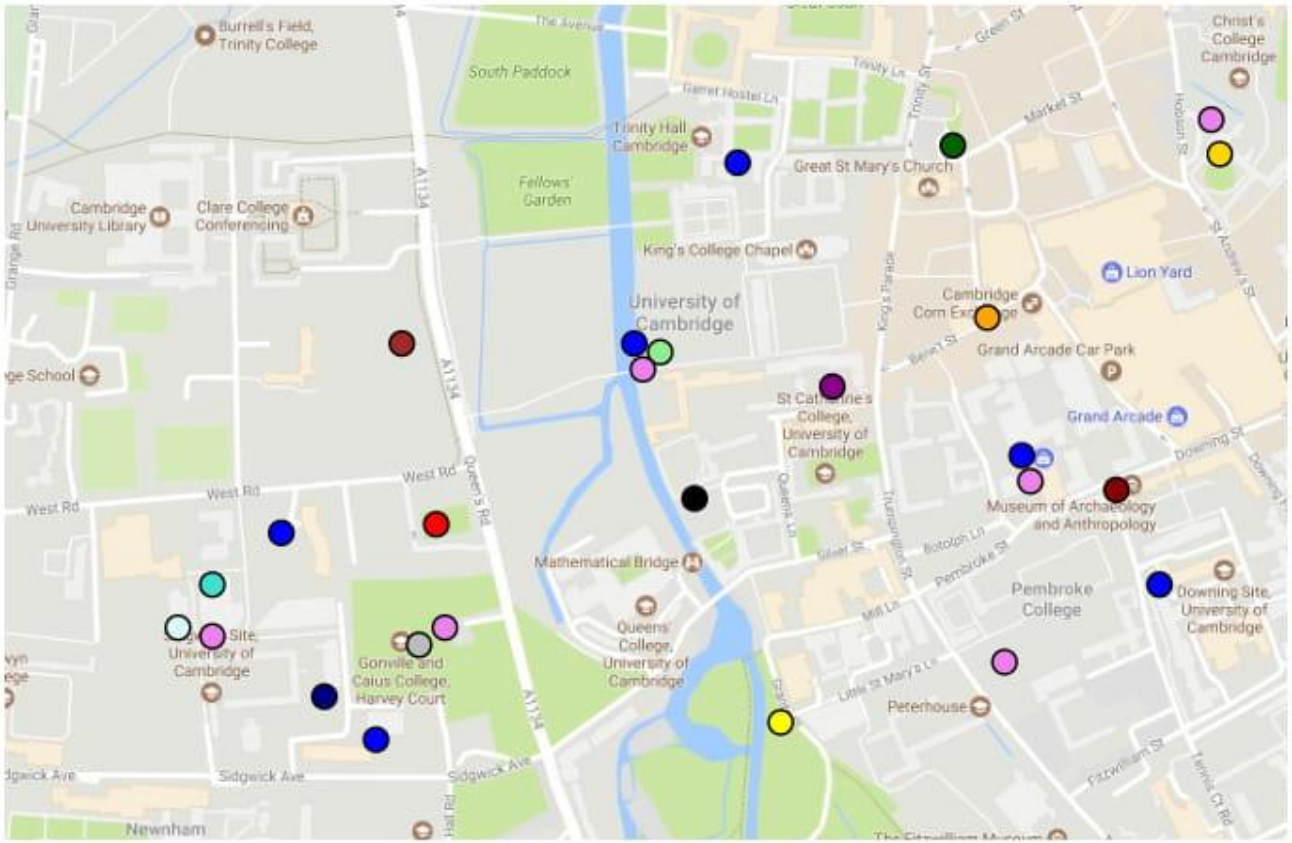
Christine Quinn Read

The Random Bar Problem

- A particular group of friends always hangs out together on Friday evenings.
- Each have an initial preference on what to do, and a phone.
- On Friday 7pm, they all start randomly calling each other, exchanging opinions.

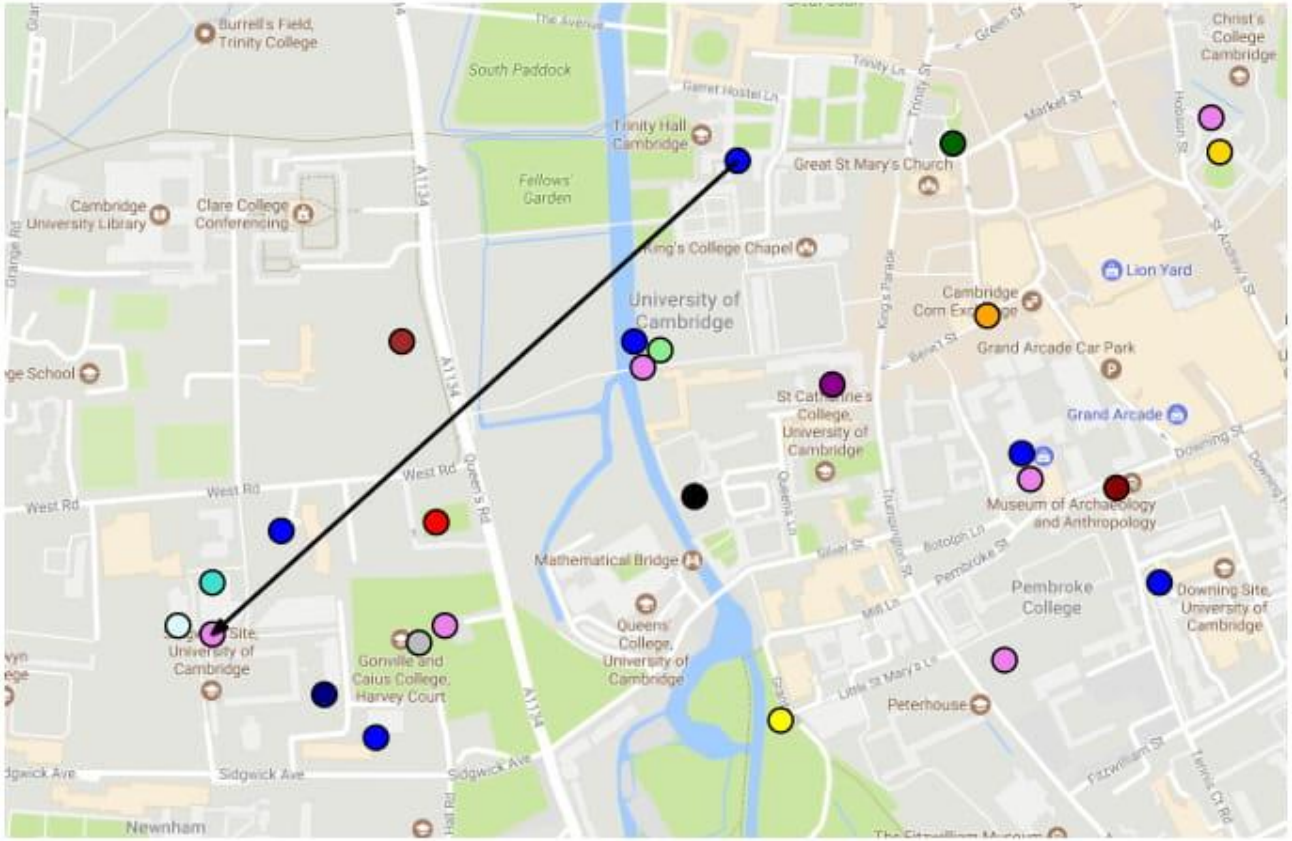
The Random Bar Problem

- A particular group of friends always hangs out together on Friday evenings.
- Each have an initial preference on what to do, and a phone.
- On Friday 7pm, they all start randomly calling each other, exchanging opinions.



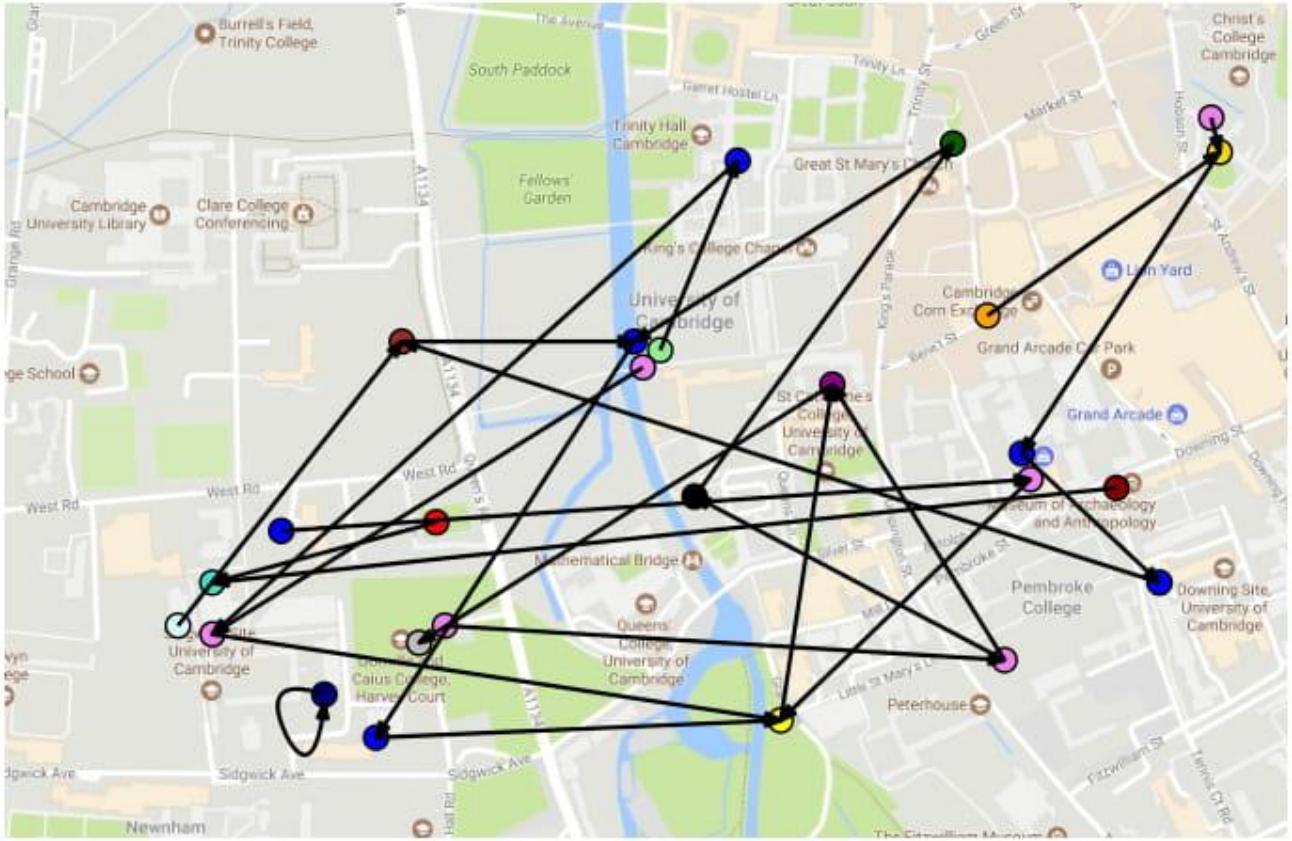
The Random Bar Problem

- A particular group of friends always hangs out together on Friday evenings.
- Each have an initial preference on what to do, and a phone.
- On Friday 7pm, they all start randomly calling each other, exchanging opinions.



The Random Bar Problem

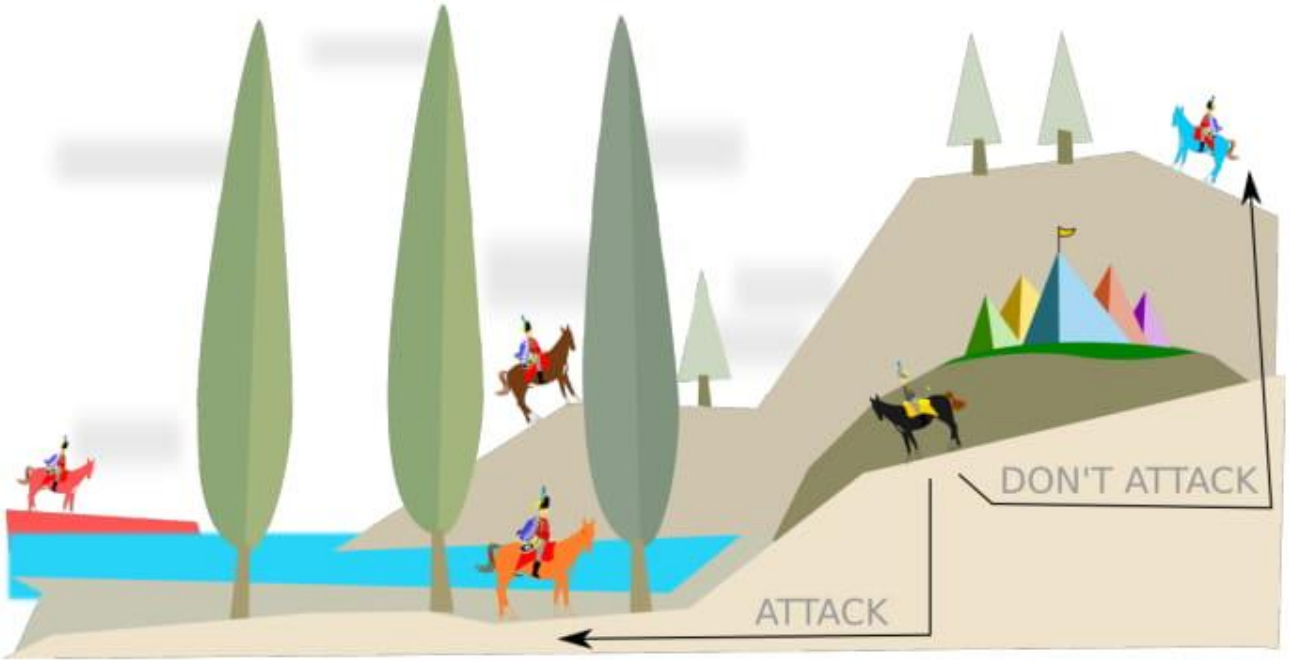
- A particular group of friends always hangs out together on Friday evenings.
- Each have an initial preference on what to do, and a phone.
- On Friday 7pm, they all start randomly calling each other, exchanging opinions.



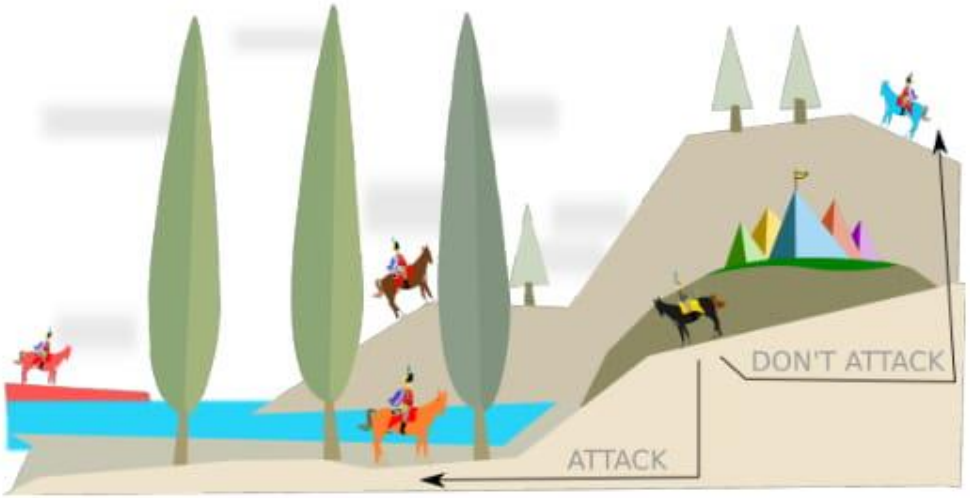
Outline

The Byzantine Generals Problem

- Each division of Byzantine army is directed by its own general.
- There are n generals, some of which are traitors.
- All armies are camped outside enemy castle, observing enemy.
- Communicate with each other by messengers.



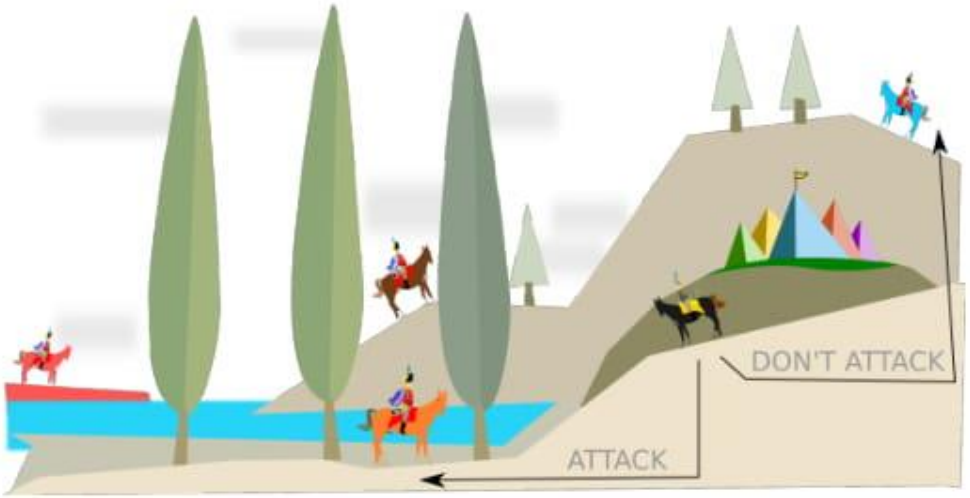
The Byzantine Generals Problem



Requirements:

- All loyal generals decide upon the same plan of action.
- The consensus is reached in reasonable time.
- A small number of traitors cannot cause the loyal generals to adopt a bad plan.

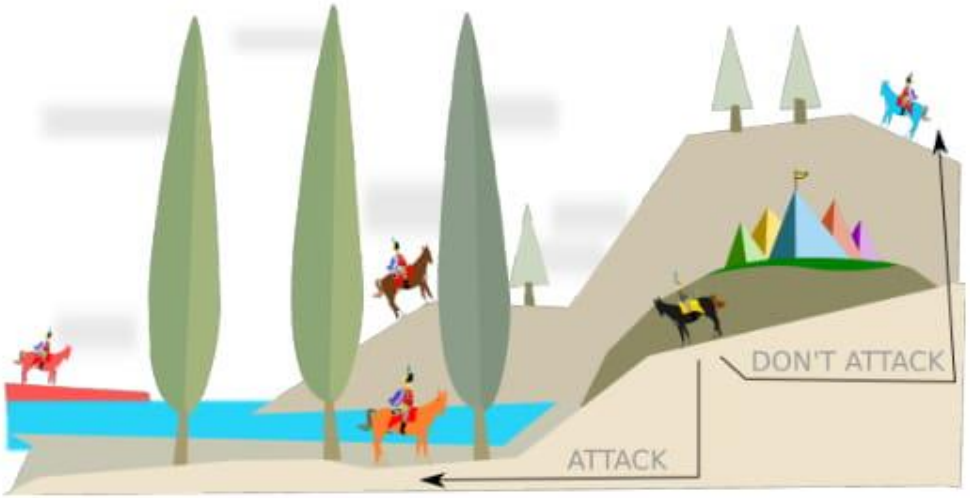
The Byzantine Generals Problem



Requirements:

- All loyal generals decide upon the same plan of action.
- The consensus is reached in reasonable time.
- A small number of traitors cannot cause the loyal generals to adopt a bad plan.

The Byzantine Generals Problem



Requirements:

- All loyal generals decide upon the same plan of action.
- The consensus is reached in reasonable time.
- A small number of traitors cannot cause the loyal generals to adopt a bad plan.

Outline

Objectives

- Find a protocol to synchronize people on the same activity!
- The consensus should be reached fast.
- The ending consensus should be equal to initially dominating preference.
- If a couple of participants are not in condition to follow the protocol, they cannot mislead the others.

Plurality consensus problem appears in several fields!

- clock synchronization
- e-voting
- PageRank
- load balancing

Objectives

- Find a protocol to synchronize people on the same activity!
- The consensus should be reached fast.
- The ending consensus should be equal to initially dominating preference.
- If a couple of participants are not in condition to follow the protocol, they cannot mislead the others.

Plurality consensus problem appears in several fields!

- clock synvhronization
- e-voting
- PageRank
- load balancing

Model Description

A Model for a Synchronous System

Random Phone-Call Model (Demers et al., 1987; Karp et al., 2000)

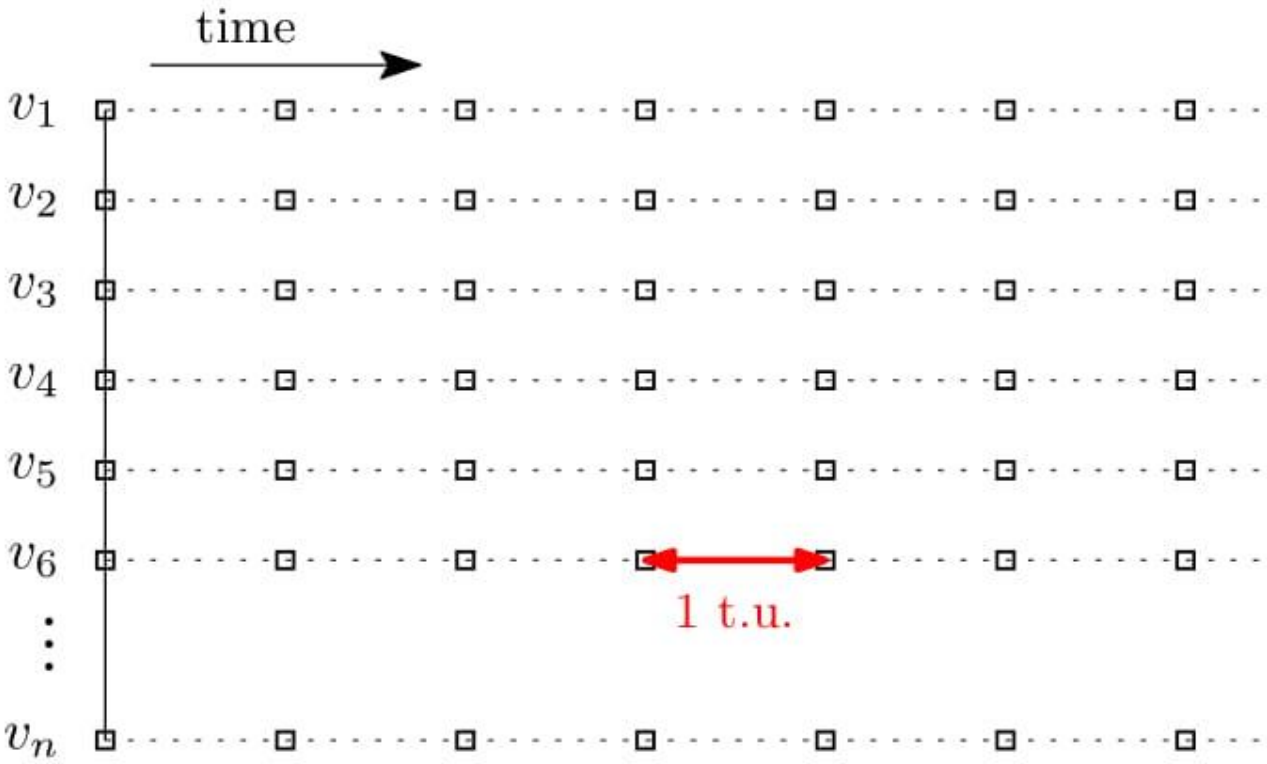
- Underlying graph K_n of possible communication links,
- algorithms operate in synchronous rounds, in parallel,
- communication with one (or a constant number) of neighbors per round,
- any new communication is established *u.a.r.*

Additional assumptions:

- Nodes are aware of n.
- Nodes have a small memory.
- Nodes are aware of the initial bias.

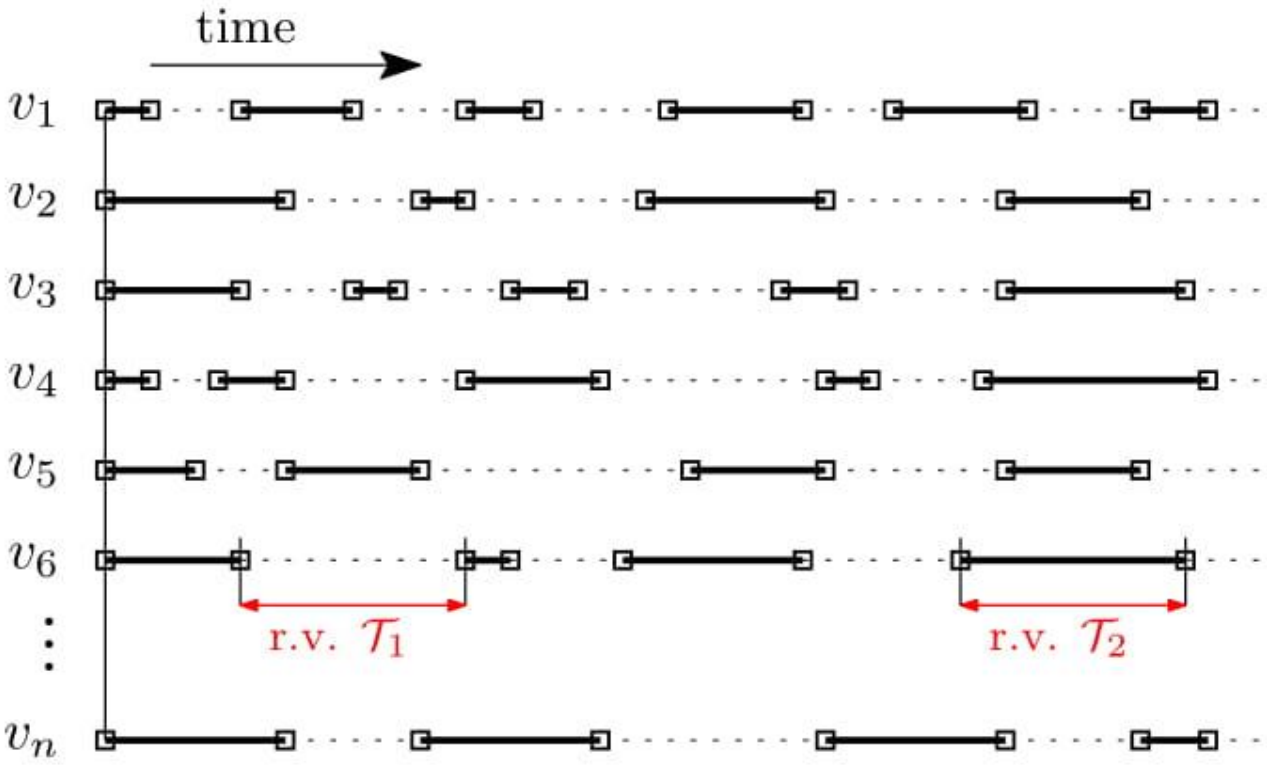
Model Description

A Synchronous Model



Model Description

An Asynchronous Model with Latencies



Model Description

An Asynchronous Model

1. Definitions of $\mathscr{T}_1$ (delay) and $\mathscr{T}_2$ (*latency*), with finite mean and variance (image).
2. A communication is established u.a.r, or to one (unique) vertex samples in the past (the address is known).
3. Restriction of requests per time unit – $\log n$
4. *Skill* to throw a biased coin.

Single-Voter Approach

PULL algorithm:

In each step, every node chooses a neighbor uniformly at random, and adopts the opinion of this node.

- Requires $\Omega(n)$ rounds to converge.
- The problem is equivalent to Coalescing Random-Walks.
- Bad success guarantee for dominant color.

Studied by

- Nakata, Imahayashi, and Yamashita (1999),
- Hassin and Peleg (2001),
- Cooper, Elsässer, Ono, and Radzik (2013),
- Berenbrink, Giakkoupis, Kermarrec and Mallmann-Trenn (2016),
- Varun, Mallmann-Trenn and Sauerwald (2017).

Single-Voter Approach

PULL algorithm:

In each step, every node chooses a neighbor uniformly at random, and adopts the opinion of this node.

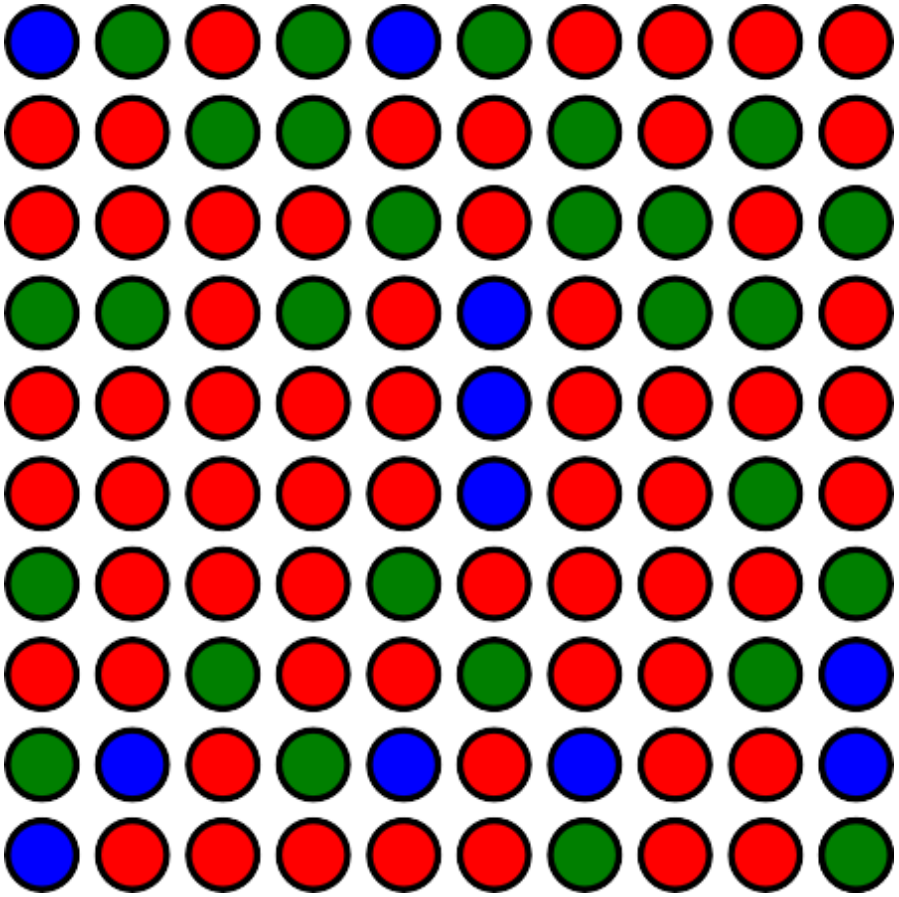
- Requires $\Omega(n)$ rounds to converge.
- The problem is equivalent to Coalescing Random-Walks.
- Bad success guarantee for dominant color.

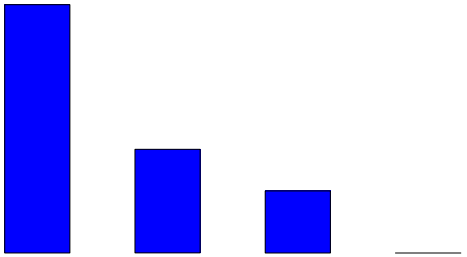
Studied by

- Nakata, Imahayashi, and Yamashita (1999),
- Hassin and Peleg (2001),
- Cooper, Elsässer, Ono, and Radzik (2013),
- Berenbrink, Giakkoupis, Kermarrec and Mallmann-Trenn (2016),
- Varun, Mallmann-Trenn and Sauerwald (2017).

PULL approach (0)

Agree with anything!

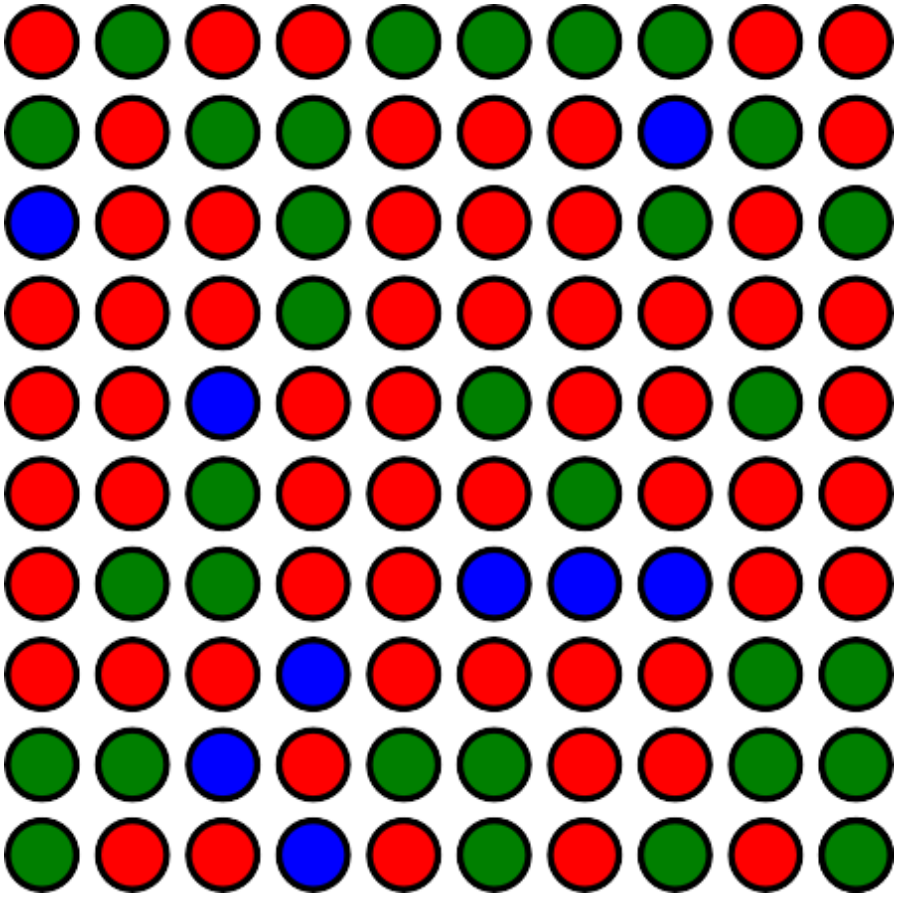


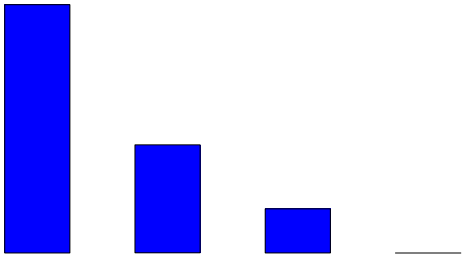


Initial distribution: (r, g, b) ~ (0.6, 0.25, 0.15)

PULL approach (1)

Agree with anything!

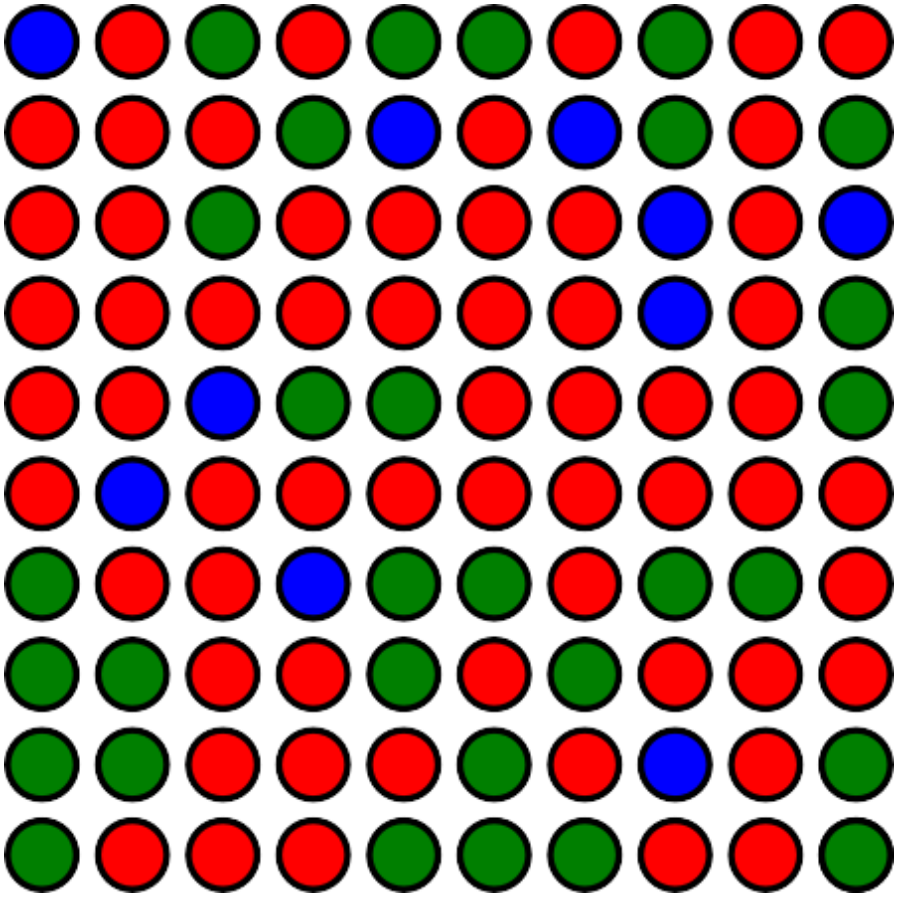


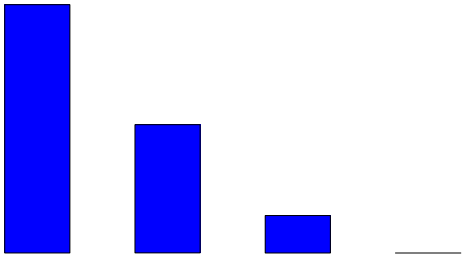


Initial distribution: (r, g, b) ~ (0.6, 0.25, 0.15)

PULL approach (2)

Agree with anything!

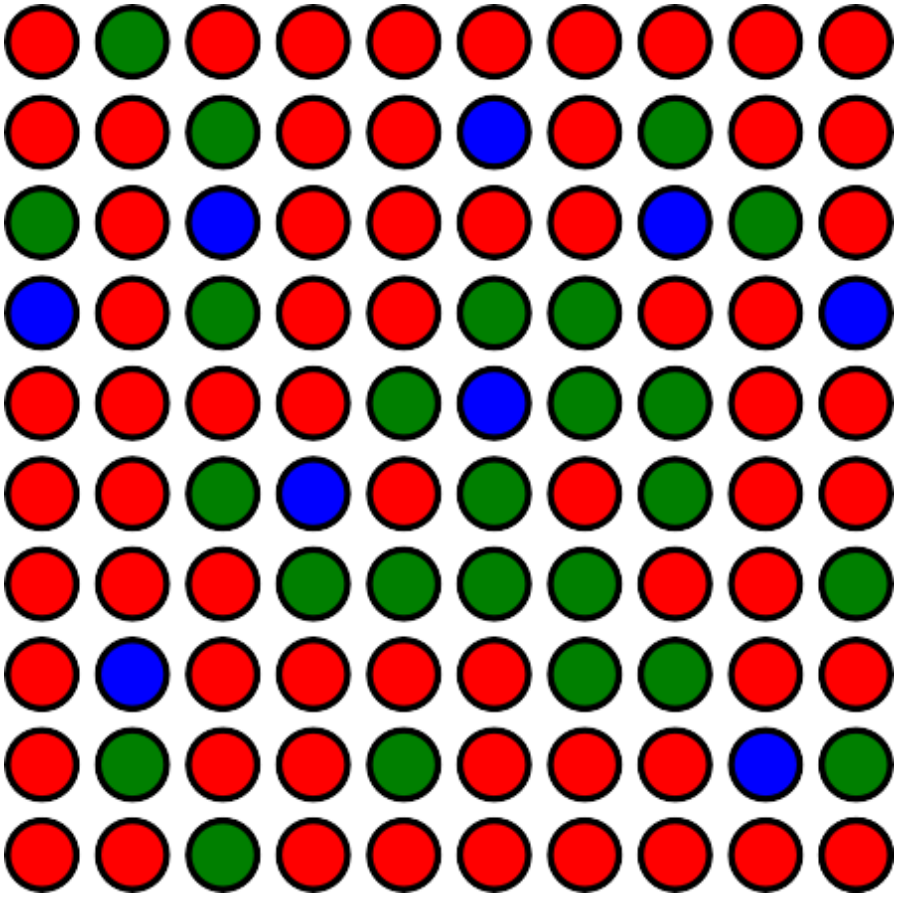


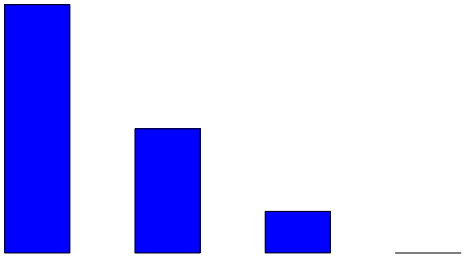


Initial distribution: (r, g, b) ~ (0.6, 0.25, 0.15)

PULL approach (3)

Agree with anything!

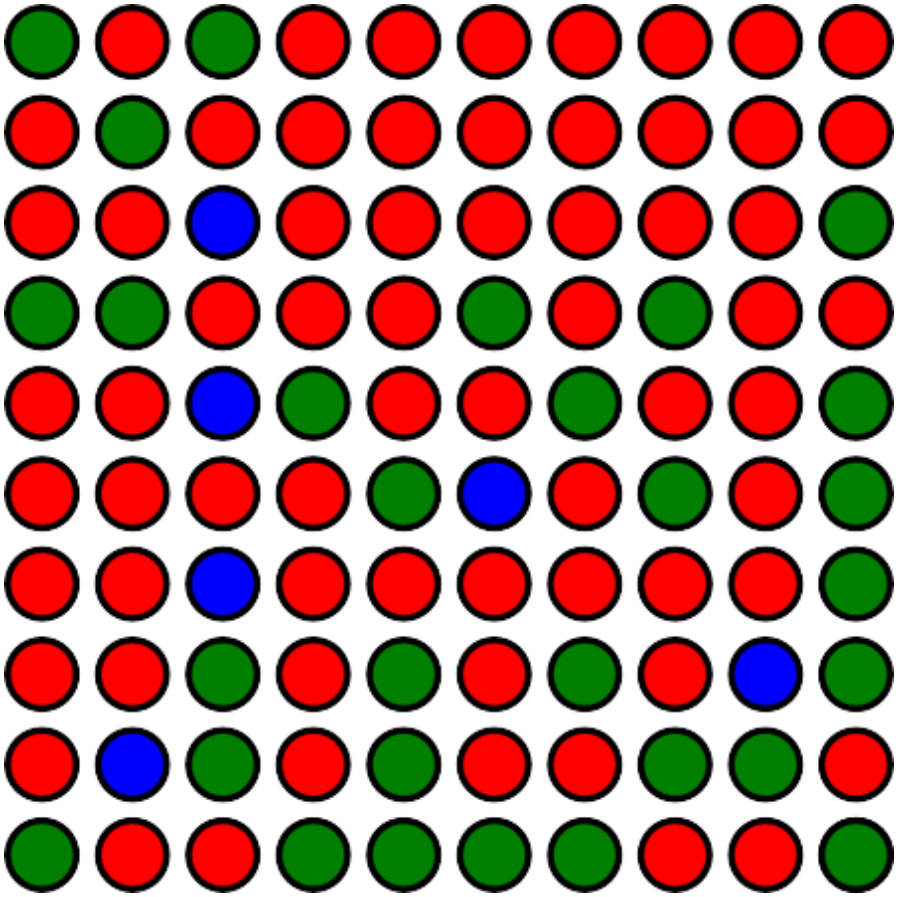


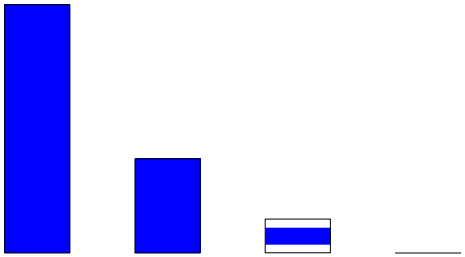


Initial distribution: (r, g, b) ~ (0.6, 0.25, 0.15)

PULL approach (4)

Agree with anything!

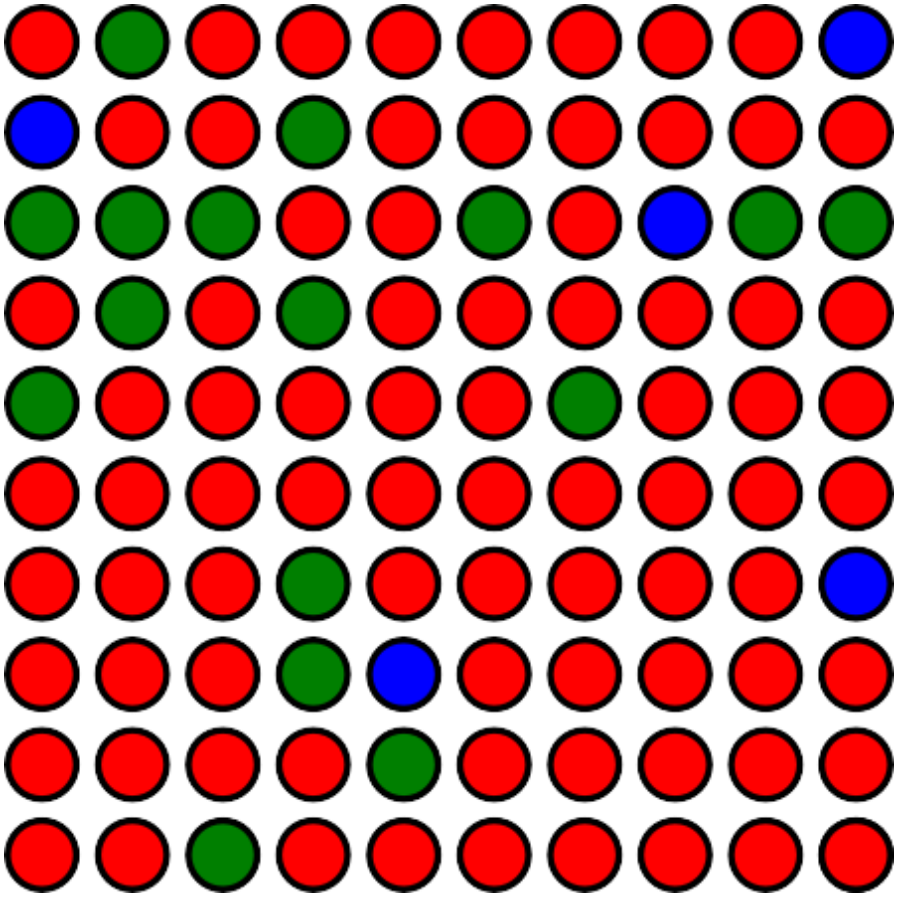


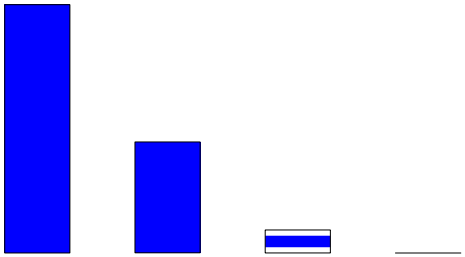


Initial distribution: (r, g, b) ~ (0.6, 0.25, 0.15)

PULL approach (5)

Agree with anything!

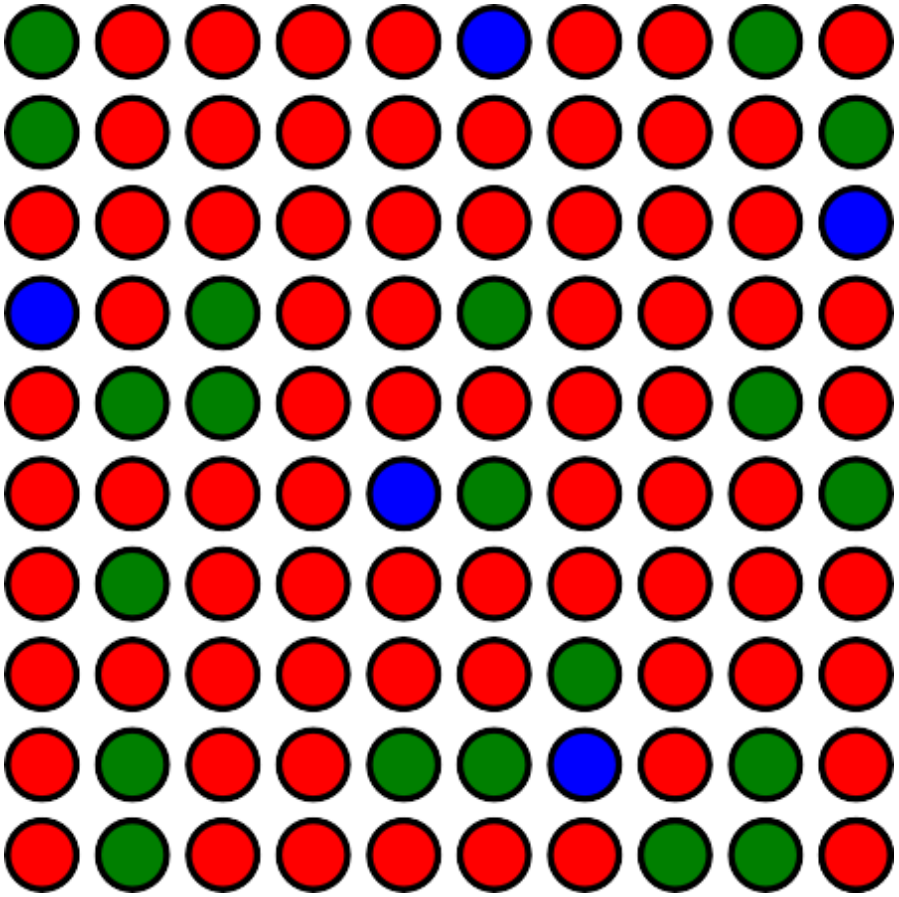


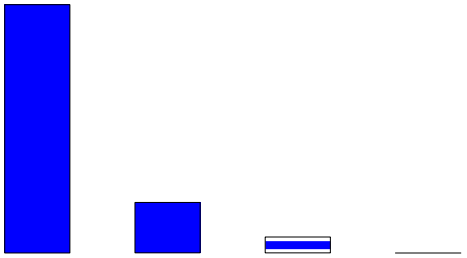


Initial distribution: (r, g, b) ~ (0.6, 0.25, 0.15)

PULL approach (6)

Agree with anything!

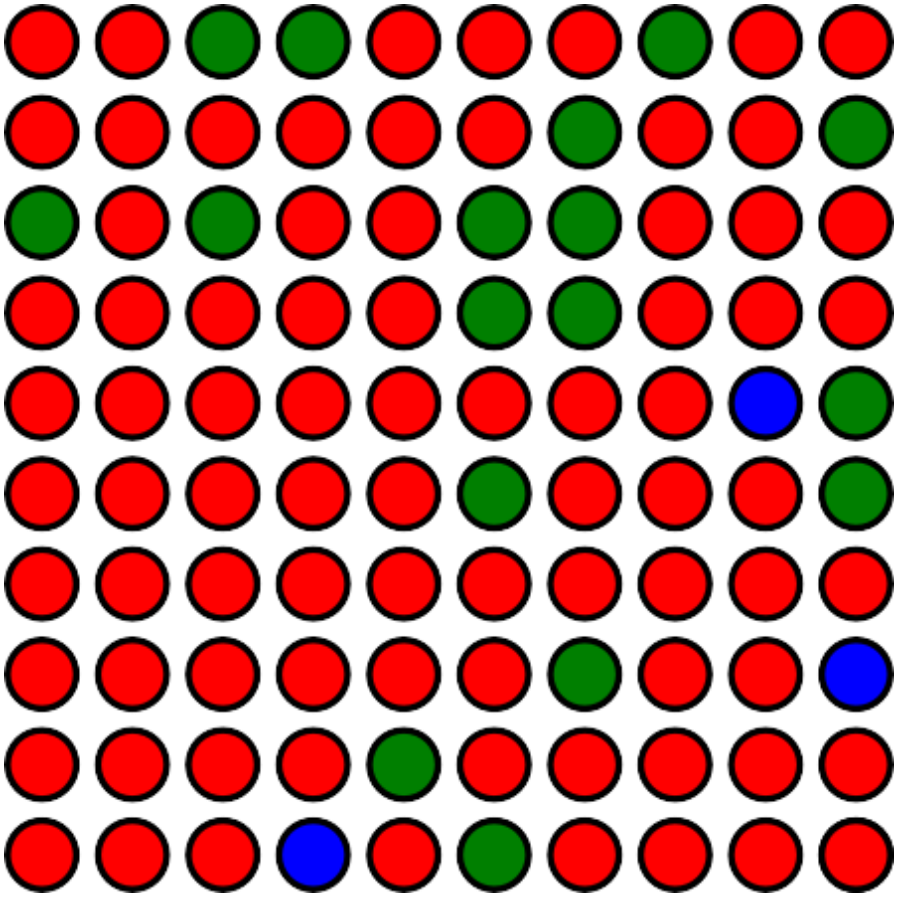


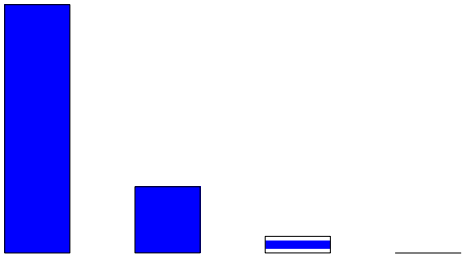


Initial distribution: (r, g, b) ~ (0.6, 0.25, 0.15)

PULL approach (7)

Agree with anything!

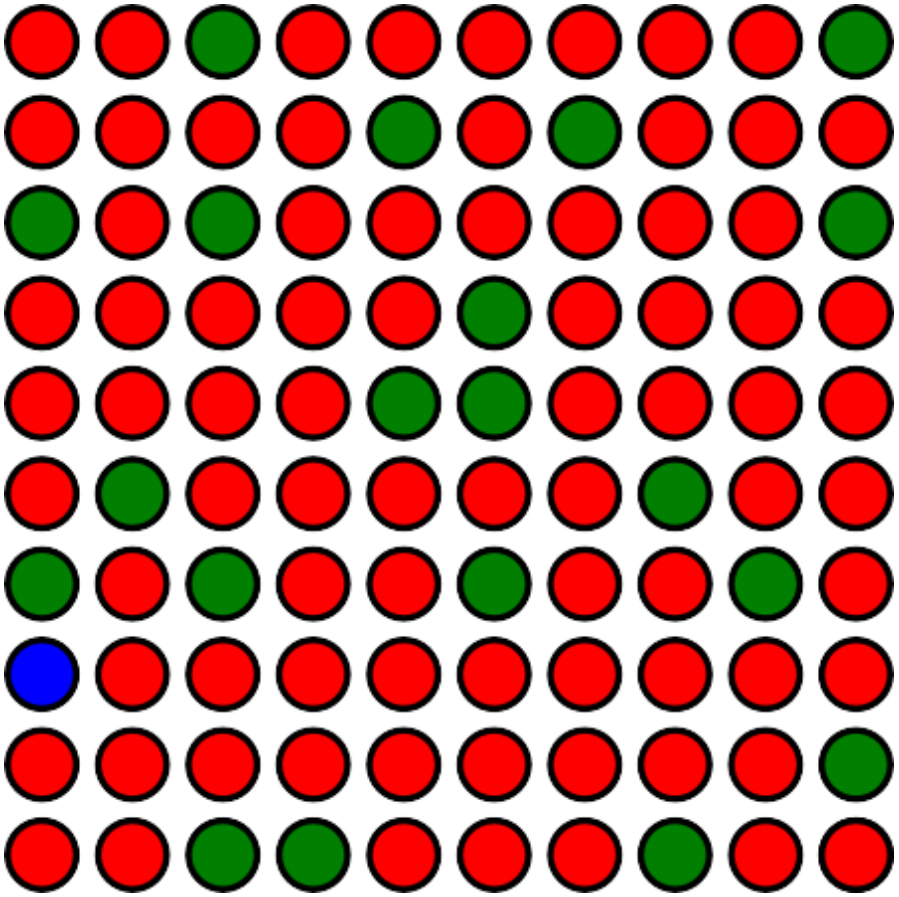


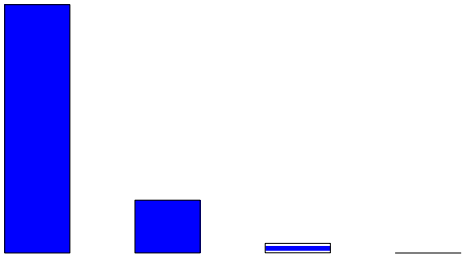


Initial distribution: (r, g, b) ~ (0.6, 0.25, 0.15)

PULL approach (8)

Agree with anything!

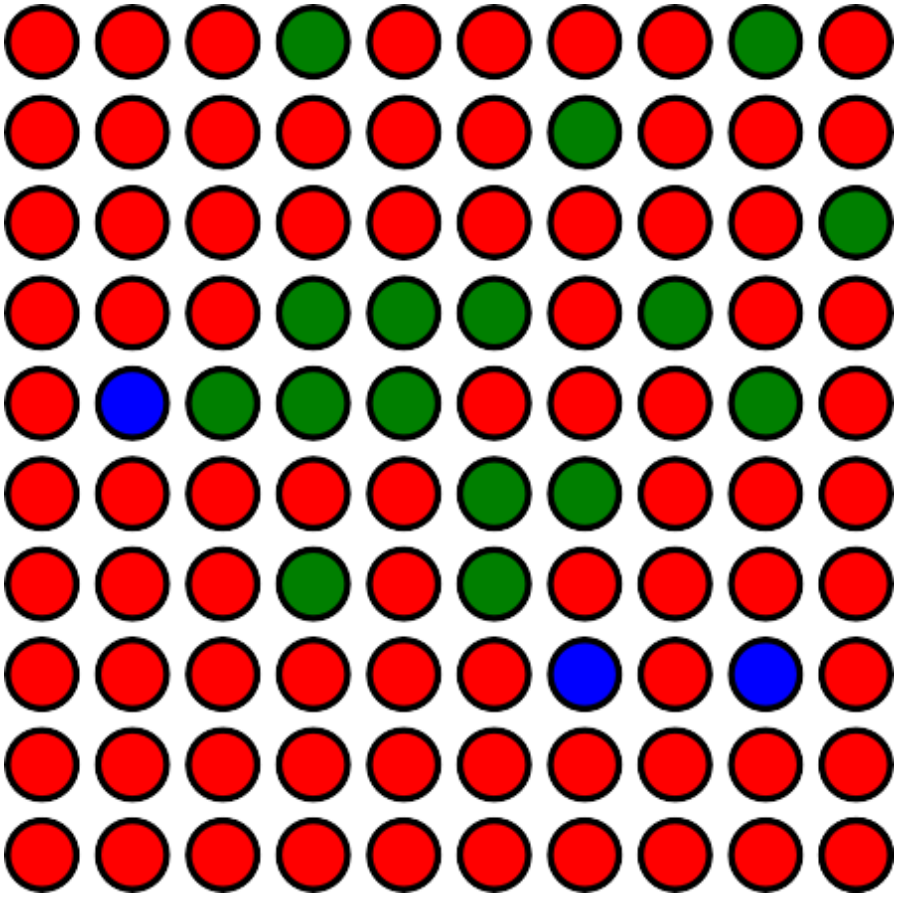


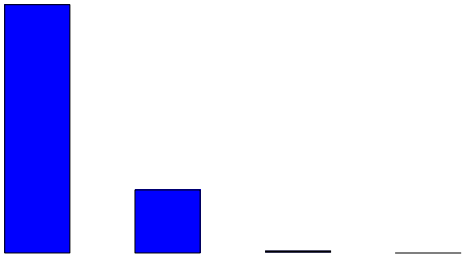


Initial distribution: (r, g, b) ~ (0.6, 0.25, 0.15)

PULL approach (9)

Agree with anything!





Initial distribution: (r, g, b) ~ (0.6, 0.25, 0.15)

2-Voter Approach

"If you hear it twice, it must be true"

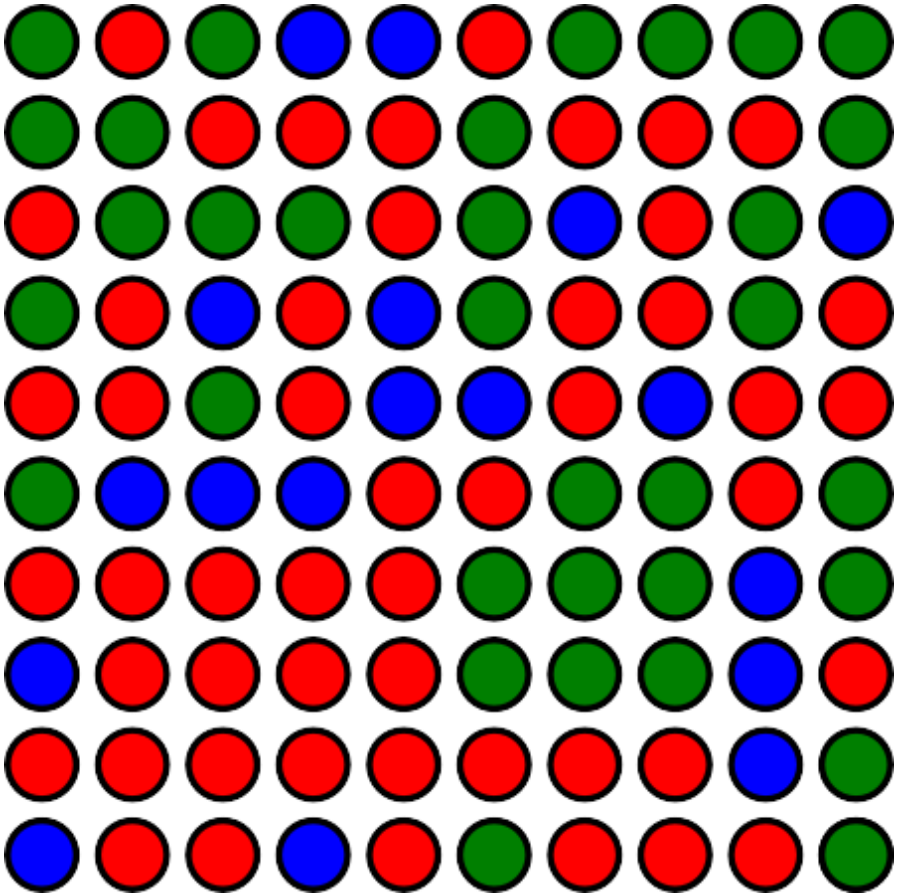
Two-Sample Voting:

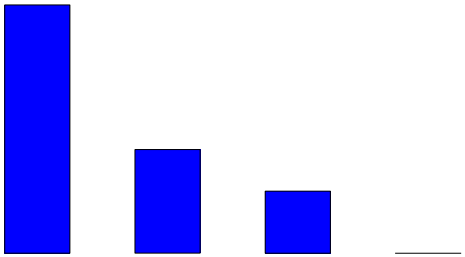
Sample two nodes and if their colors coincide, adopt it.

- We may guarantee the convergence of the initially dominant opinion whip.
- Requires a bias of $\Omega\left(\sqrt{n \log n}\right)$ for the initially dominating color to win whip.,
- however, only after $\Omega(k)$ rounds.

2-Voter Approach (0)

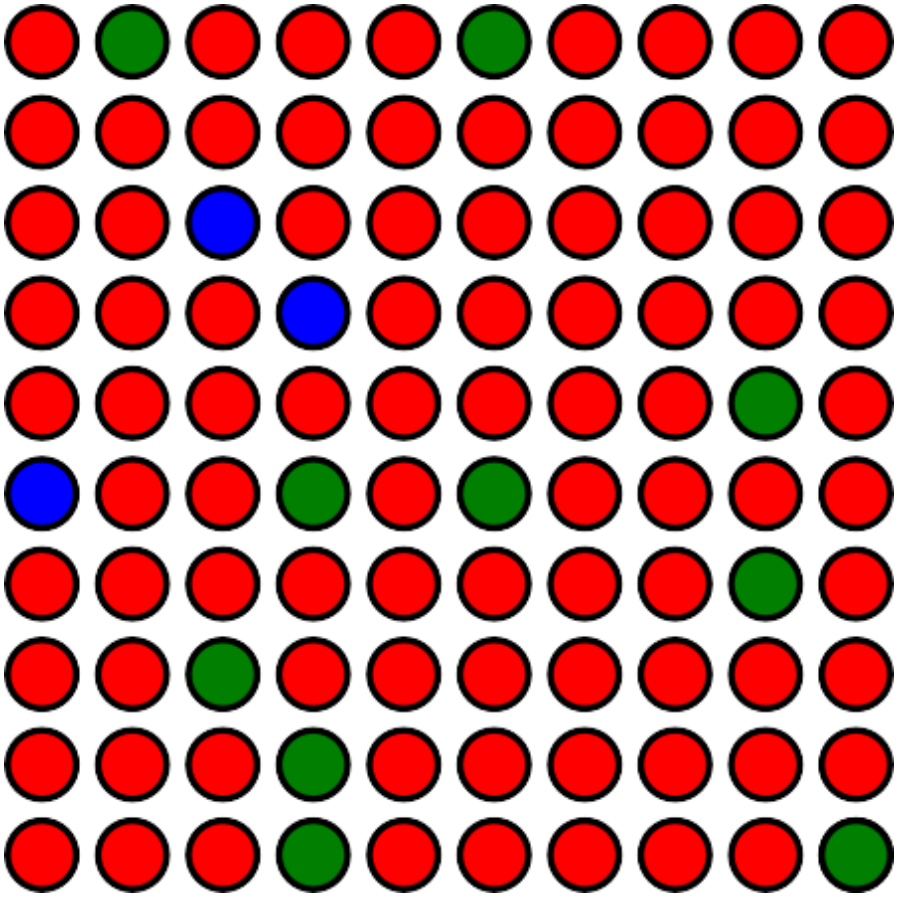
Example 1 $(r, g, b) \sim (0.6, 0.25, 0.15)$

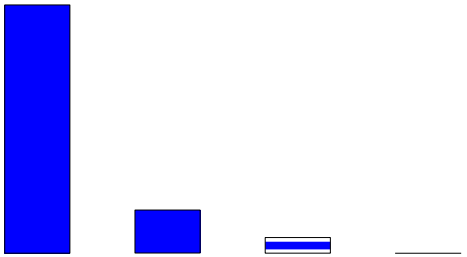




2-Voter Approach (1)

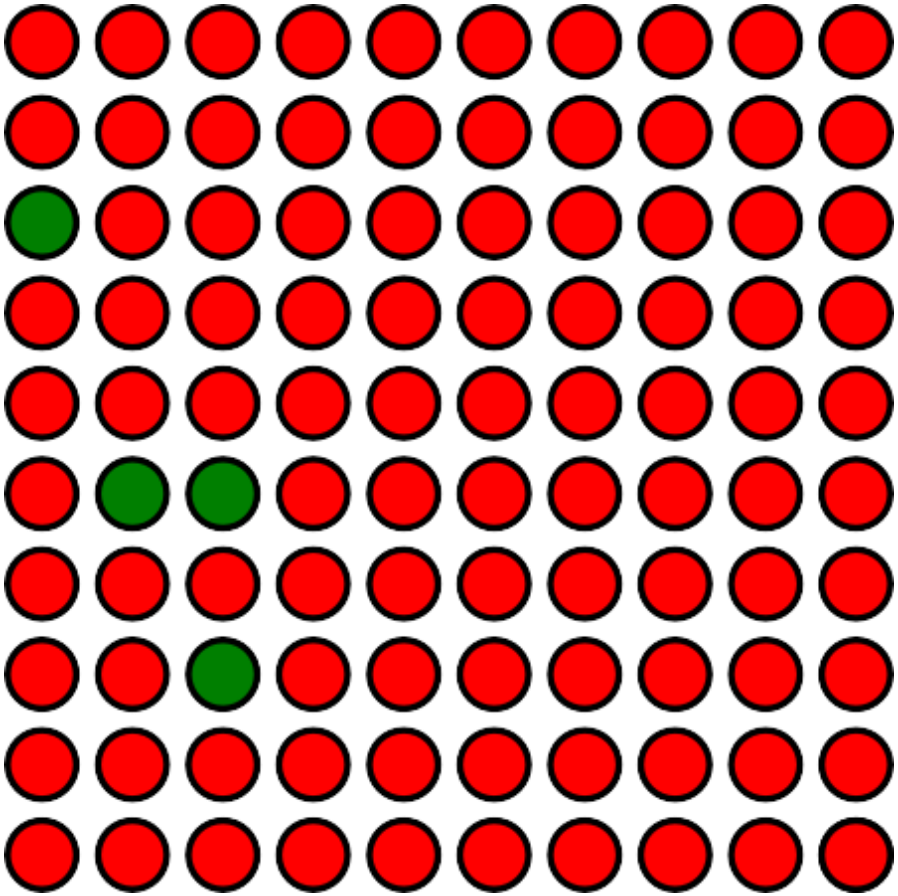
Example 1

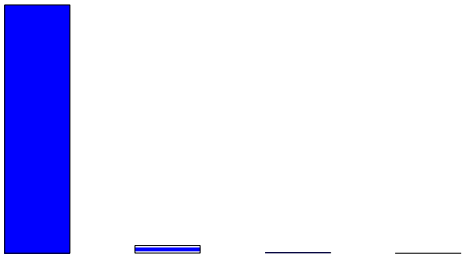




2-Voter Approach (2)

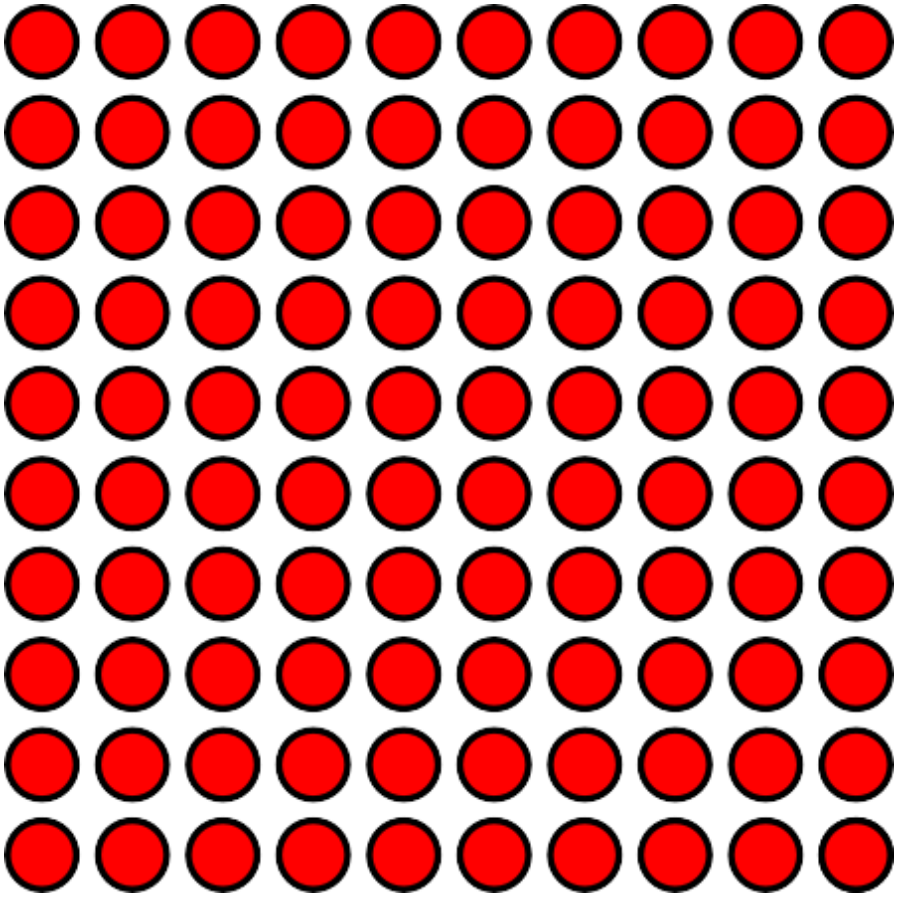
Example 1

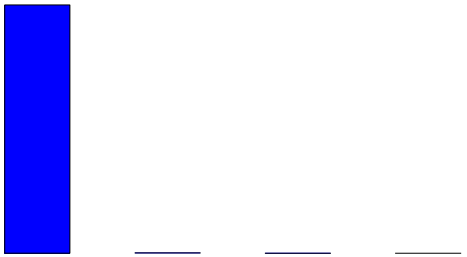




2-Voter Approach (3)

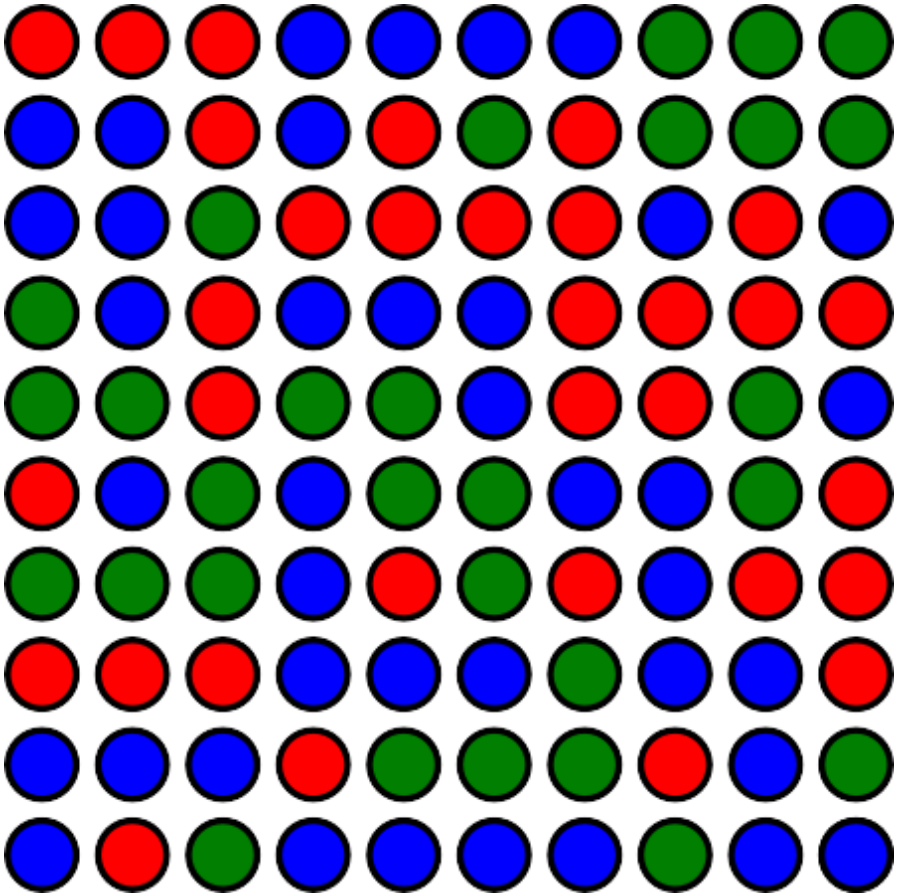
Example 1

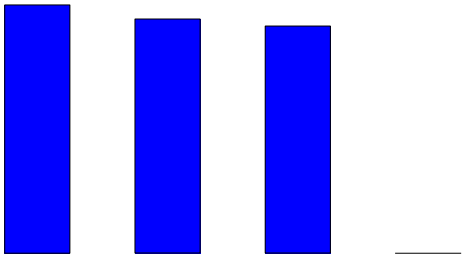




2-Voter Approach (0)

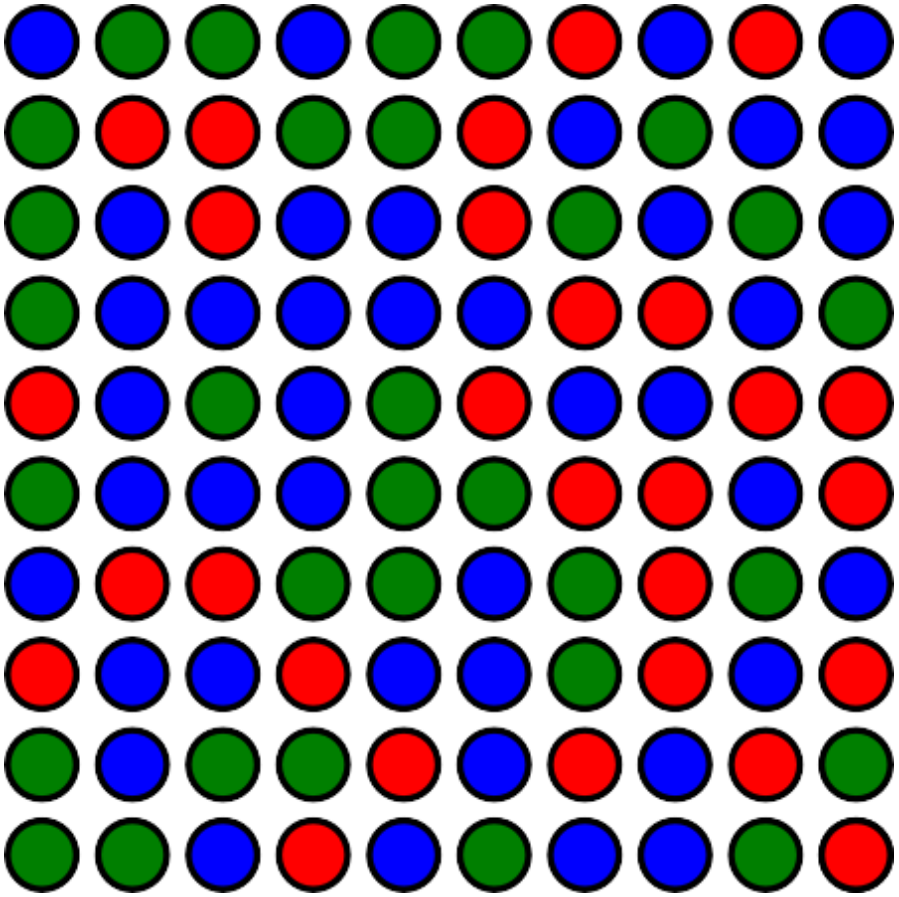
Example 2: distribution $(r, g, b) \sim (0.35, 0.33, 0.32)$

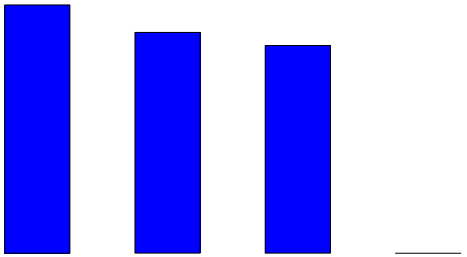




2-Voter Approach (1)

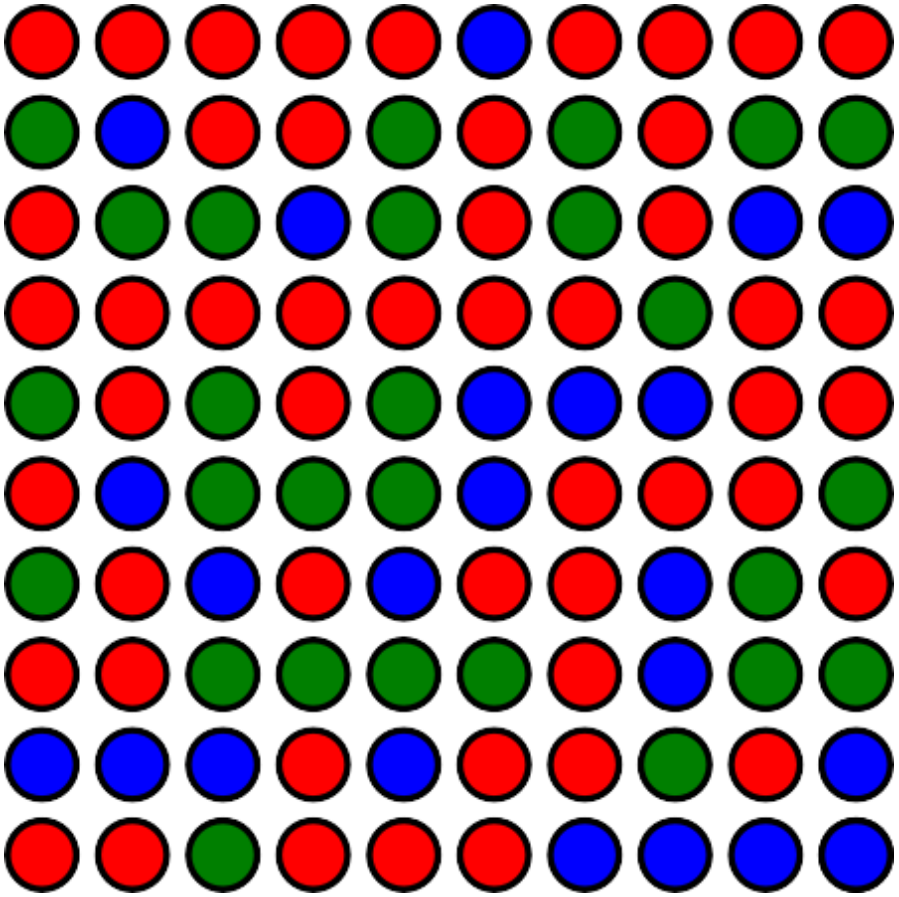
Example 2: distribution (r,g,b) ~ (0.37,0.33,0.30)

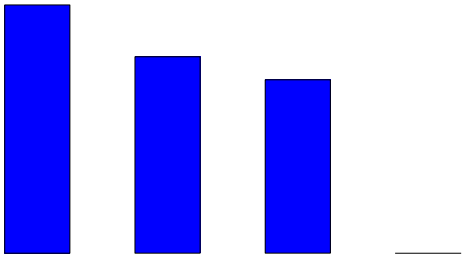




2-Voter Approach (2)

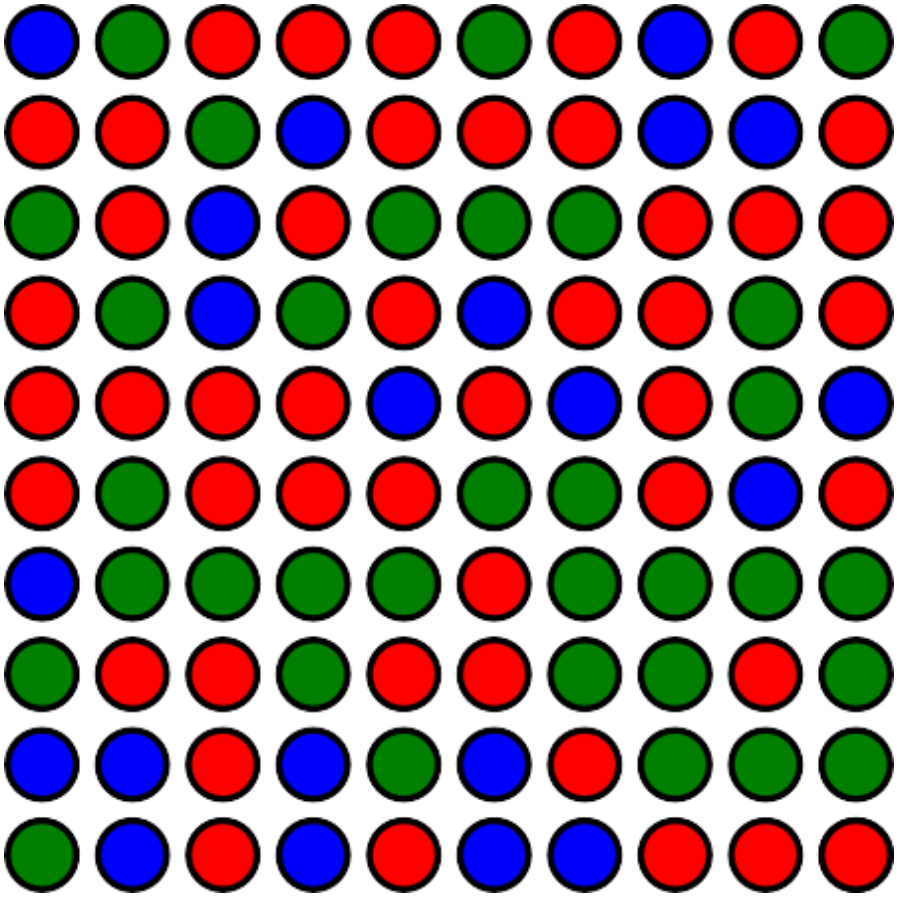
Example 2: distribution (r,g,b) ~ (0.40,0.32,0.28)

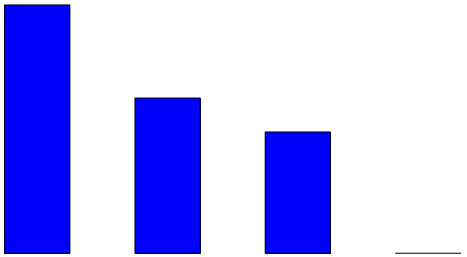




2-Voter Approach (3)

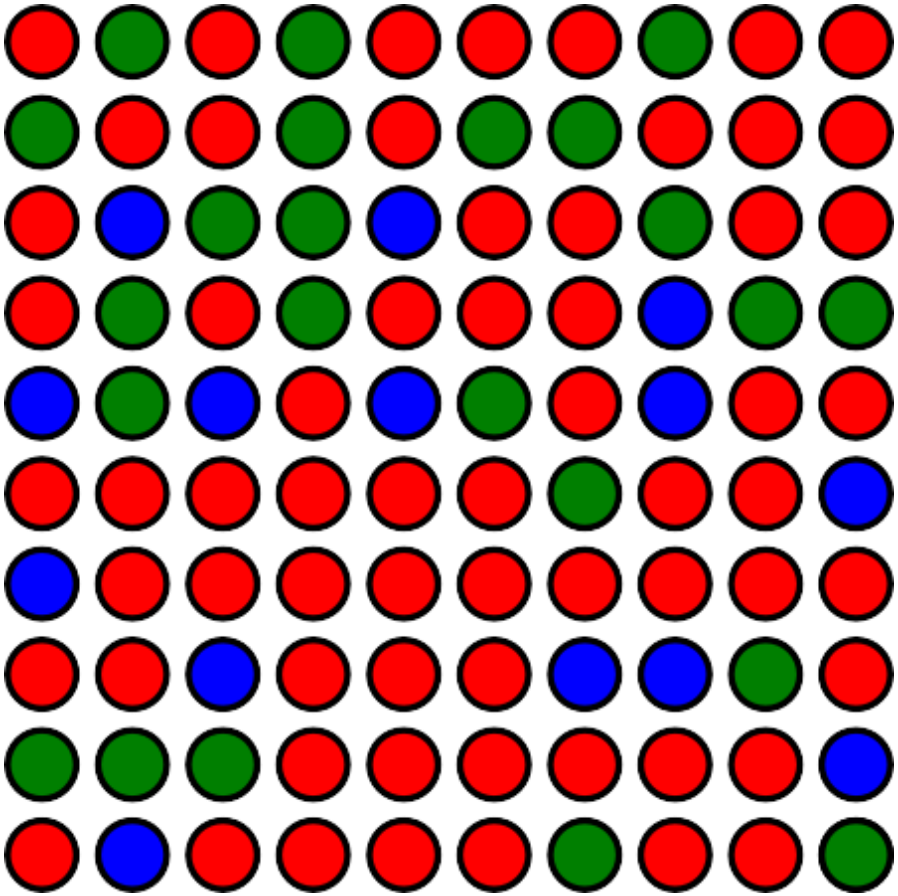
Example 2: distribution (r,g,b) ~ (0.47,0.30,0.23)

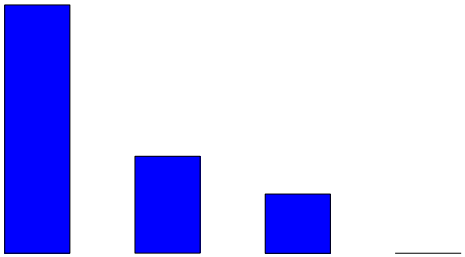




2-Voter Approach (4)

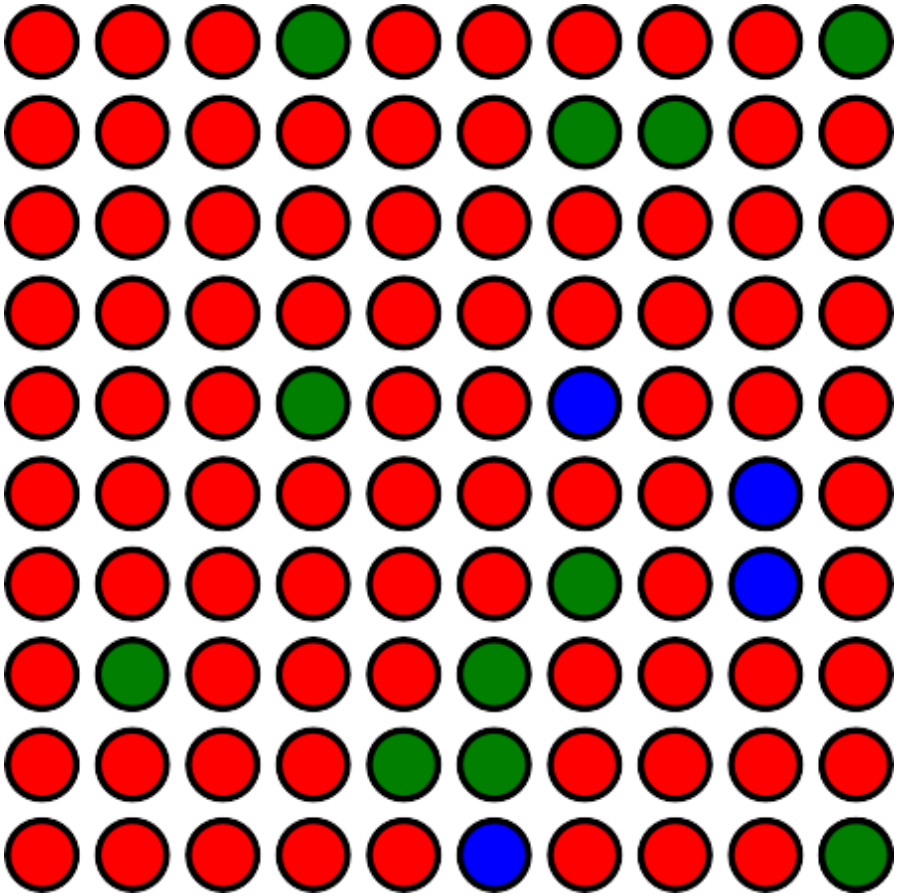
Example 2: distribution $(r, g, b) \sim (0.61, 0.24, 0.15)$

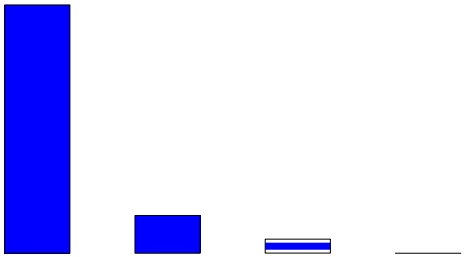




2-Voter Approach (5)

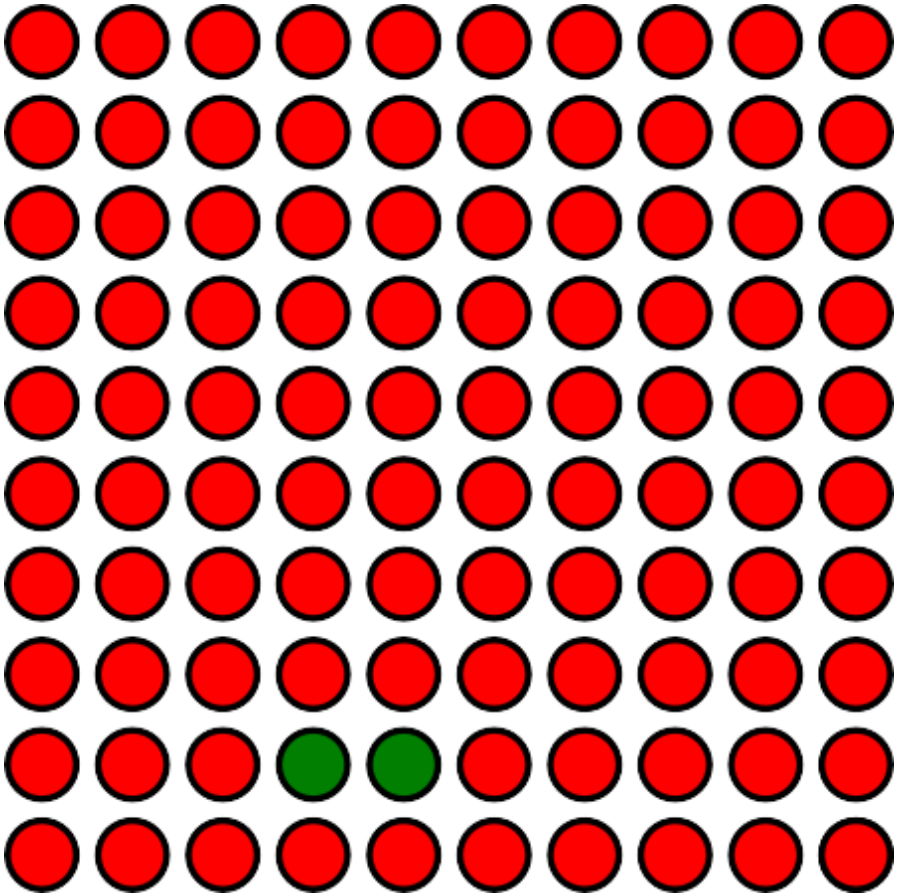
Example 2: distribution $(r, g, b) \sim (0.83, 0.13, 0.05)$

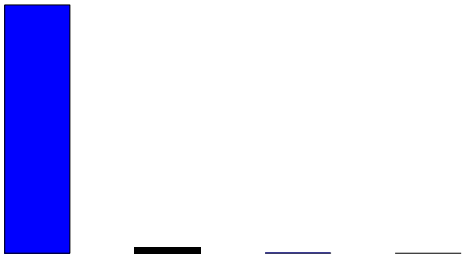




2-Voter Approach (6)

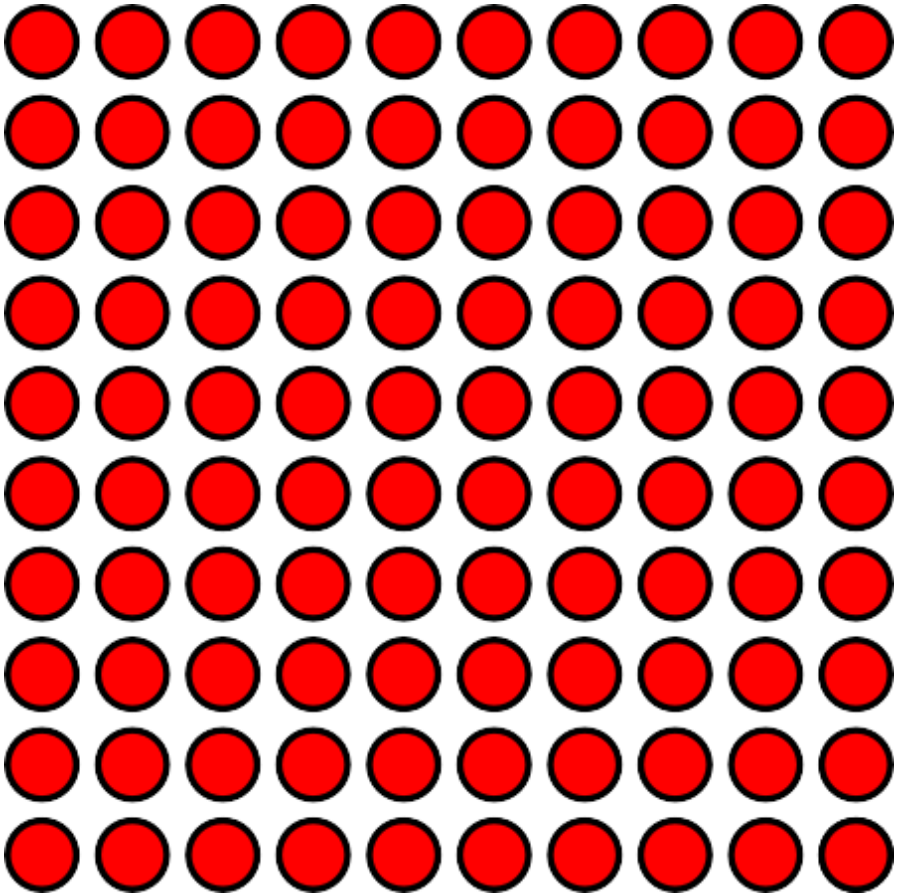
Example 2: distribution $(r, g, b) \sim (0.98, 0.02, 0)$

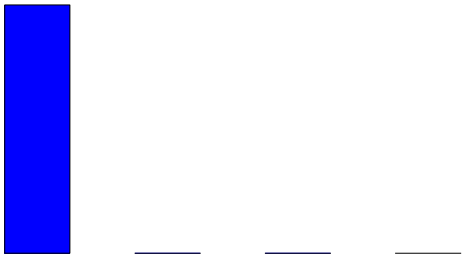




2-Voter Approach (7)

Example 2





2-Voter Approach

Two-Sample Voting - related work

- Hassin-Peleg, Nakata et al.: general graphs
- Cruise-Ghanesh: complete graph, traditional model
- Doerr et al.: complete graph, median rule
- Mossel et al.: expander, majority consensus
- Abdullah-Draief: random graph, five-sample voting
- Becchetti et al. or Angluin et al.: 3-majority protocol, 3-state protocol
- Two-Party Two-Sample Voting:
  - Cooper, Elsässer and Radzik (2014)
  - Cooper, Elsässer, Radzik, Rivera and Shiraga (2015)

The speed can be improved by using gossip algorithms.

Random gossiping

short overview

Gossip algorithms

- spread information by vertices repeatedly forwarding information to a few random neighbors
- algorithm is distributed and robust (fault-tolerant)
- can be very fast and *message-efficient*
- for K_n, it takes $O(\log n)$ time for the information to spread from $O(1)$ to $O(n)$ vertices
- examples: PULL or PUSH protocols

Our Results

For two specific restrictions on α and k

Bound on k:	Bound on α:	Time Complexity:
/	$\alpha \geq k^{\varepsilon}, \varepsilon > 0$	$O(\log k + \log\log n)$
$k \leq \exp\left(\frac{\log\log n}{\log\log\log n}\right)$	$\alpha \geq 1 + \frac{1}{\log\log\log n - 1}$	$O(\log\log n)$
/	$\alpha \geq 1 + \frac{1}{\text{polylog}(n)}$	$O(\log\log n \cdot \log k)$

Memory overhead is of order $O(\log\log k + \log\log\log n)$. In the asynchronous setting with $(1-\varepsilon)$-convergence, the results are the same, and the memory overhead is of order $O(\log\log n)$.

Consensus susceptible to Sybils

- All consensus protocols based on membership…
  - … assume independent failures …
  - … which implies strong notion of identity
- "Sybil attack" (p2p literature ~2002)
  - Idea: one entity can create many "identities" in system
  - Typical defense: 1 IP address = 1 identity
  - Problem: IP addresses aren't difficult / expensive to get, esp. in world of botnets & cloud services

Ingredient #5:

The Proof of Work

Consensus based on “work”

- Rather than “count” IP addresses, bitcoin “counts” the amount of CPU time / electricity that is expended

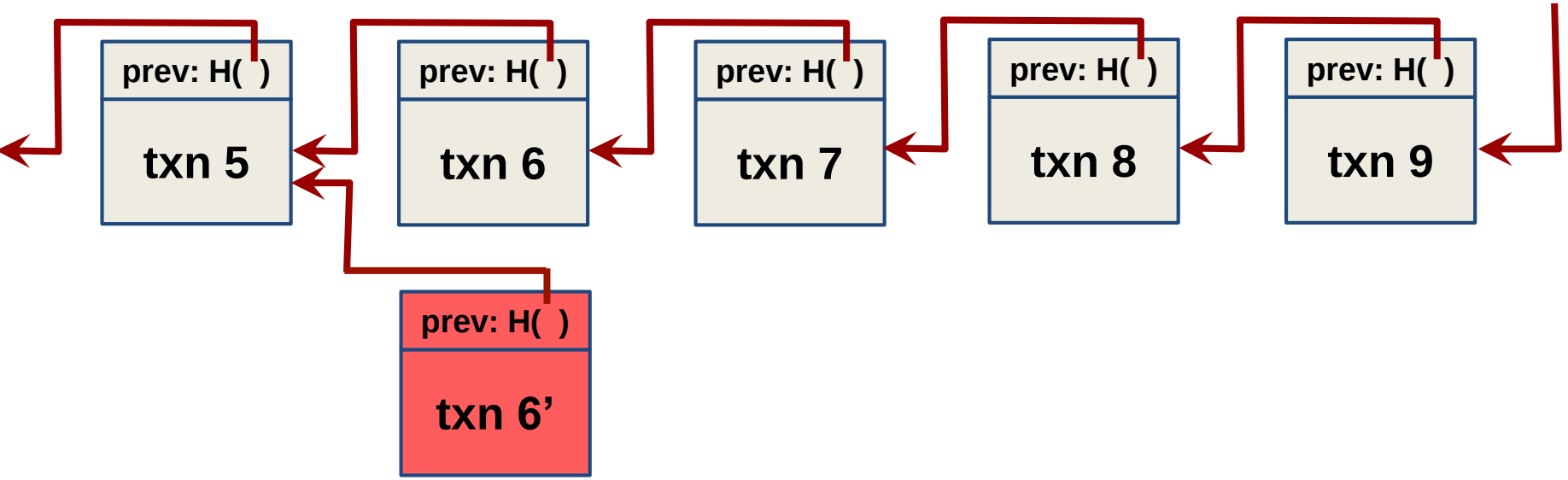
> **“The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.”**
>
> **- Satoshi Nakamoto**

- Proof-of-work: Cryptographic “proof” that certain amount of CPU work was performed

Form of randomized leader election

- After enough work is performed:
  - New leader elected for past ~10 min
  - Leader elected randomly, probability of selection proportional to leader's % of global hashing power
  - Leader decides which transactions comprise block

Key idea: Chain length requires work



- Generating a new block requires "proof of work"
- "Correct" nodes accept longest chain
- Creating fork requires rate of malicious work >> rate of correct
  - So, the older the block, the "safer" it is from being deleted

Use hashing to determine work!

- Recall hash functions are one-way / collision resistant
  - Given *h*, hard to find *m* such that *h* = *hash(m)*
- But what about finding partial collision?
  - *m* whose hash has most significant bit = 0?
  - *m* whose hash has most significant bits = 00?
  - Assuming output is randomly distributed, complexity grows exponentially with # bits to match

The Proof of Work

Find **nonce** such that

hash (**nonce** || prev_hash || block data) < target

i.e., hash has certain number of leading 0's

Miner's job description:

- Pick a set of transactions for block.
- Build "block header": prevhash, version, timestamp, txns, …
- Try to find nounce till hash < target OR till another node wins!

The Proof of Work

Find **nonce** such that

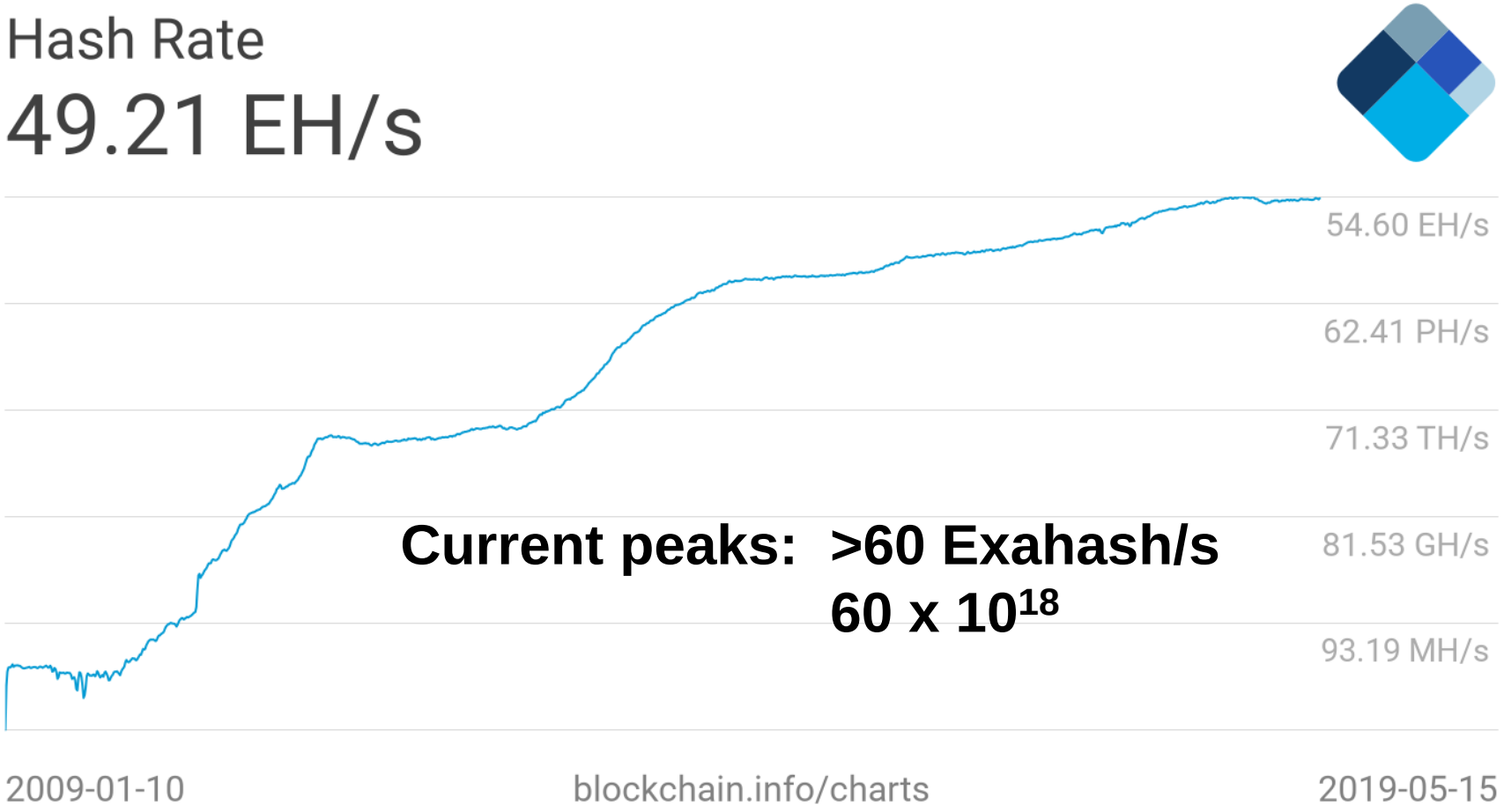
hash (**nonce** || prev_hash || block data) < target

i.e., hash has certain number of leading 0's

What about changes in total system hashing rate?

- Target is recalculated every 2 weeks
- Goal: One new block every 10 minutes

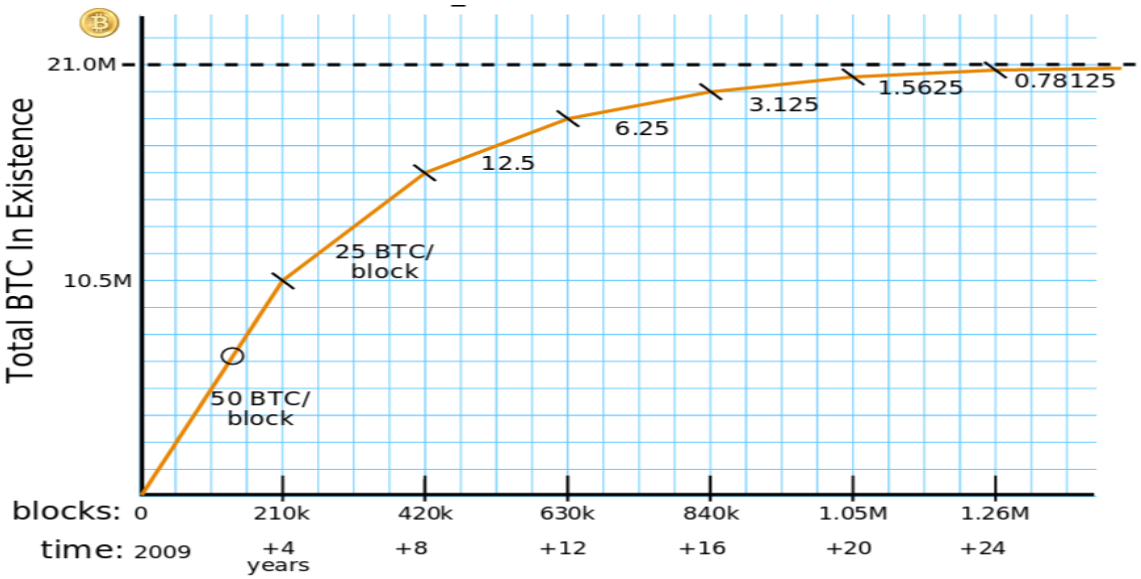
Historical hash rate trends of bitcoin



Current peaks: >60 Exahash/s
60×10^{18}

Tech: CPU → GPU → ASICs

Why consume all this energy?



- Creating a new block creates bitcoin!
  - Initially 50 BTC, decreases over time, currently 12.5
  - New bitcoin assigned to party named in new block
  - Called “mining” as you search for gold/coins

Incentivizing correct behavior?

- Race to find nonce and claim block reward, at which time race starts again for next block

  hash (**nonce** || prev_hash || block data)

  – As solution has prev_hash, corresponds to particular chain

- Correct behavior is to accept longest chain

  – "Length" determined by aggregate work, not # blocks

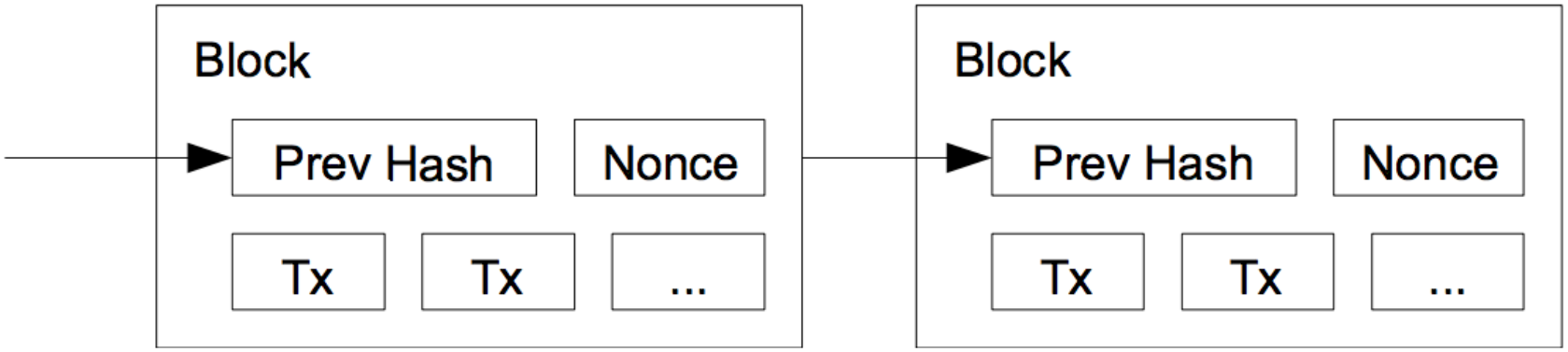
  – So miners incentivized only to work on longest chain, as otherwise solution not accepted

  – Remember blocks on other forks still "create" bitcoin, but only matters if chain in collective conscious (majority)

Ingredient #6:

The Block Structure

One block = many transactions



- Each miner picks a set of transactions for block;
- Builds “block header”: prevhash, version, timestamp, txns, …
- Tries to find nounce s.t. hash < target OR another node wins:
  - Pick nonce for header, compute hash = SHA256(SHA256(header))

Transaction format:

Create 12.5 coins, credit to Alice	
Transfer 3 coins from Alice to Bob	**SIGNED(Alice)**
Transfer 8 coins from Bob to Carol	**SIGNED(Bob)**
Transfer 1 coins from Carol to Alice	**SIGNED(Carol)**

How do you determine if Alice has balance?

Scan backwards to time 0 ?

Transaction format

Inputs:	Ø *// Coinbase reward*
Outputs:	25.0→PK_Alice
Inputs:	*H*(prevtxn, 0) *// 25 BTC from Alice*
Outputs:	25.0→PK_Bob **SIGNED(Alice)**
Inputs:	*H* (prevtxn, 0) *// 25 BTC From Alice*
Outputs:	5.0→PK_Bob, 20.0 →PK_Alice2 **SIGNED(Alice)**
Inputs:	*H* (prevtxn1, 1), *H*(prevtxn2, 0) // 10+5 BTC
Outputs:	14.9→PK_Bob **SIGNED(Alice)**

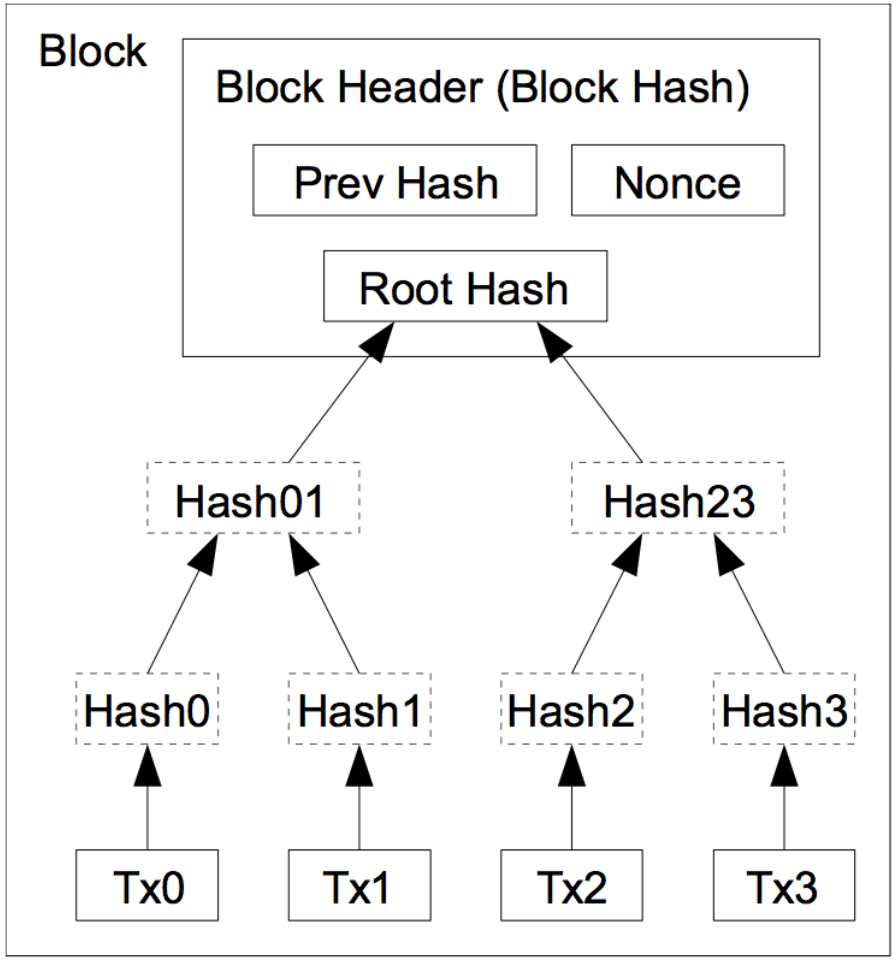
- Transaction typically has 1+ inputs, 1+ outputs
- Making change: 1st output payee, 2nd output self
- Output can appear in single later input (avoids scan back)

Transaction format

Inputs:	Ø	*// Coinbase reward*
Outputs:	25.0→PK_Alice	
Inputs:	*H*(prevtxn, 0)	*// 25 BTC from Alice*
Outputs:	25.0→PK_Bob	**SIGNED(Alice)**
Inputs:	*H* (prevtxn, 0)	*// 25 BTC From Alice*
Outputs:	5.0→PK_Bob, 20.0 →PK_Alice	**SIGNED(Alice)**
Inputs:	*H* (prevtxn1, 1), *H*(prevtxn2, 0)	// 10+5 BTC
Outputs:	14.9→PK_Bob	**SIGNED(Alice)**

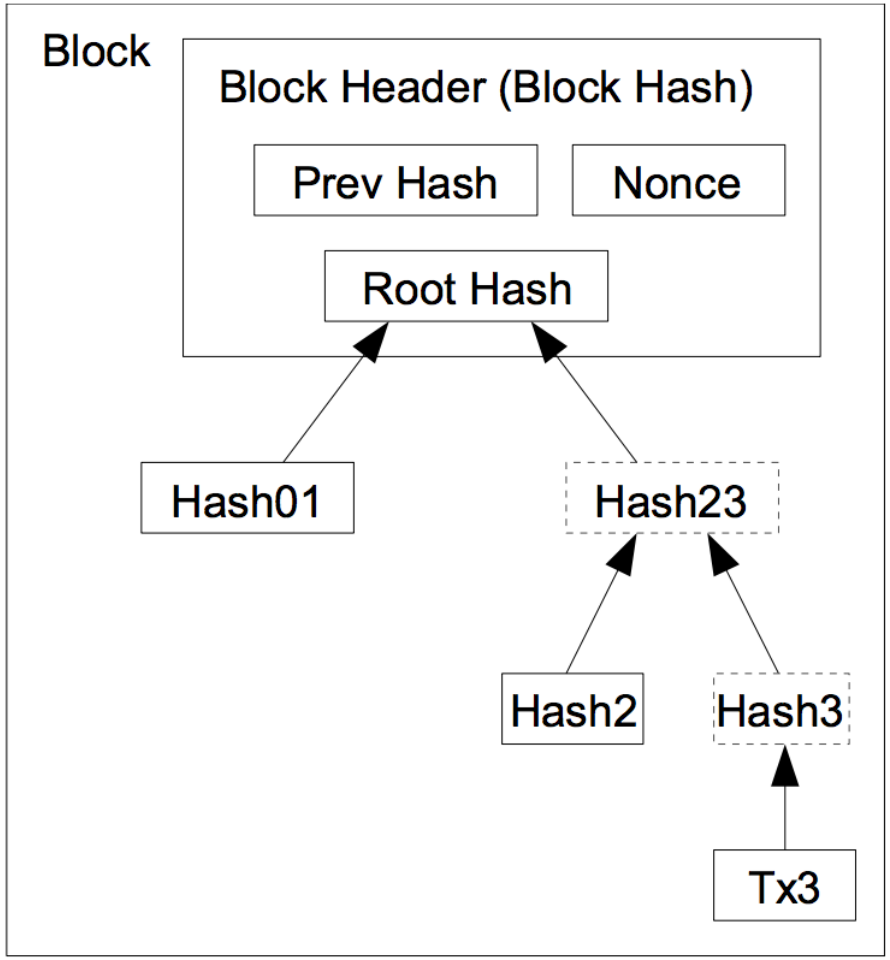
- Unspent portion of inputs is “transaction fee” to miner
- In fact, “outputs” are stack-based scripts
- 1 Block = 1MB max

Storage / verification efficiency



- Merkle tree
  - Binary tree of hashes
  - Root hash “binds” leaves given collision resistance
- Using a root hash
  - Block header now constant size for hashing
  - Can prune tree to reduce storage needs over time

Storage / verification efficiency



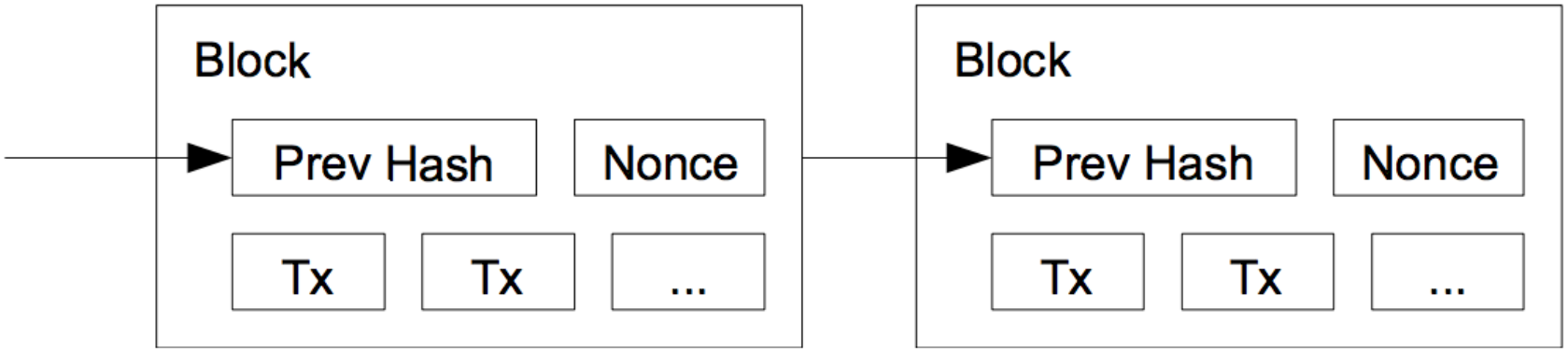
- Merkle tree
  - Binary tree of hashes
  - Root hash “binds” leaves given collision resistance
- Using a root hash
  - Block header now constant size for hashing
  - Can prune tree to reduce storage needs over time
    - Can prune when all txn outputs are spent
    - Now: 80GB pruned, 300GB unpruned

Ingredient ##:

Hard questions?

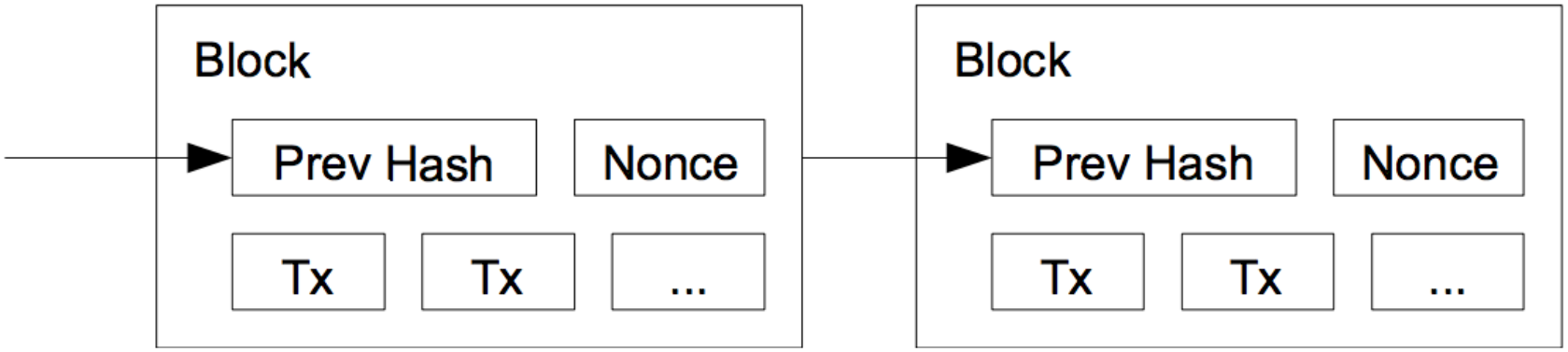
+random details

How long does one transaction take?



- At some time *T*, block header constructed
- Those transactions had been received *[T – 10 min, T]*
- Block will be generated at time T + 10 min (on average)
- So transactions are from 10 - 20 min before block creation
- Can be much longer if “backlog” of transactions is long

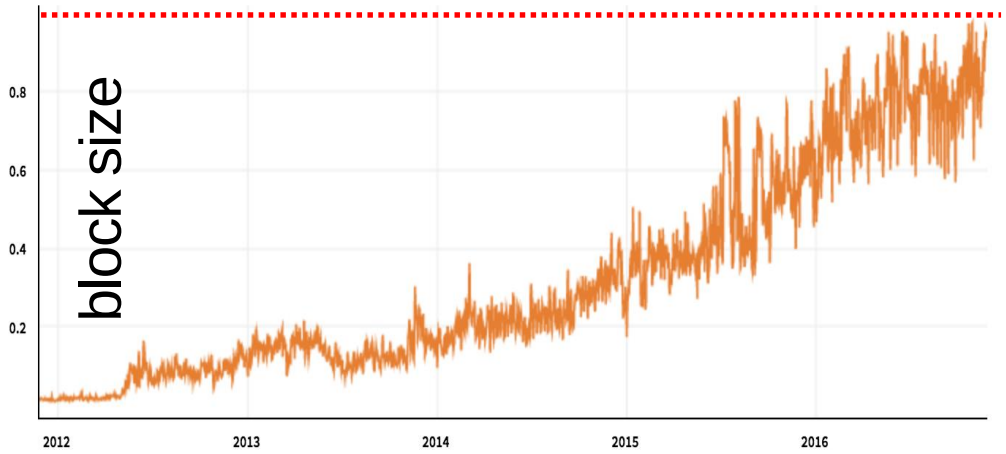
When do you trust a transaction?



- We trust:
  - After we know it is “stable” on the hash chain
  - Recall that the longer the chain, the hard to “revert”
- Common practice: transaction “committed” when 6 blocks deep
  - i.e., Takes another ~1 hour for txn to become committed

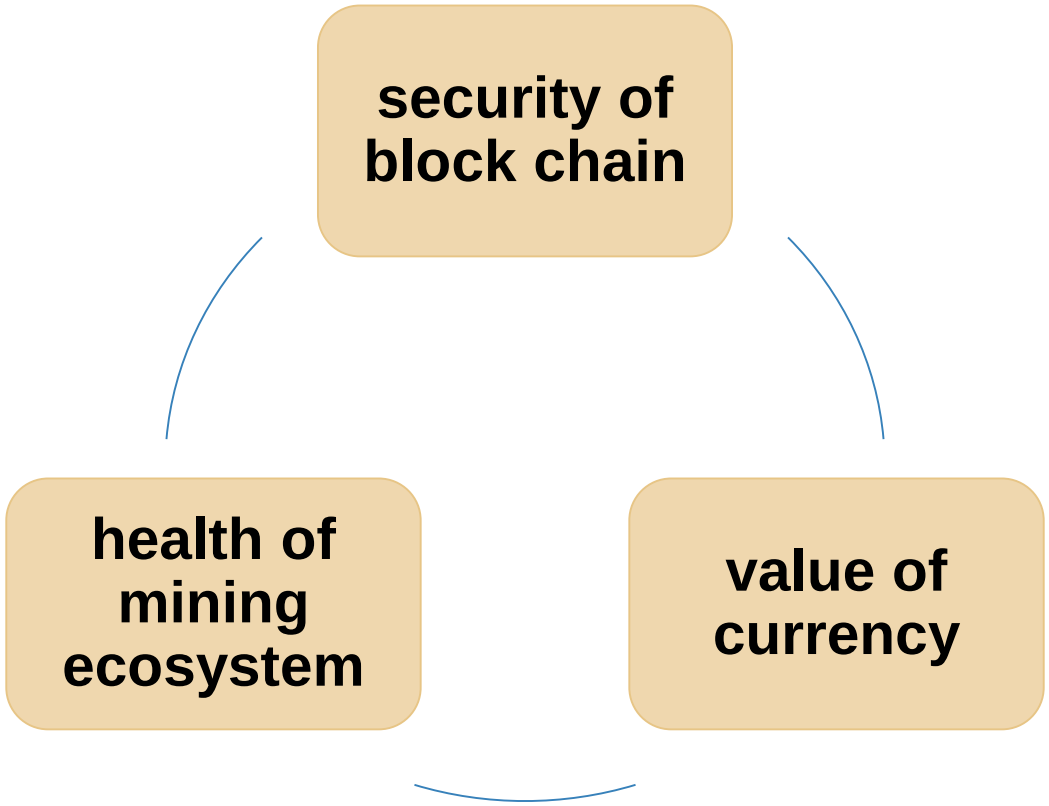
Does it scale?

- Scaling limitations
  - 1 block = 1 MB max
  - 1 block ~ 2000 txns
  - 1 block ~ 10 min
  - So, 3-4 txns / sec
  - Log grows linearly, joining requires full dload and verification



- Visa peak load comparison
  - Typically 2,000 txns / sec
  - Peak load in 2013: 47,000 txns / sec

Bitcoin & blockchain intrinsically linked



Rich ecosystem: Mining pools

health of mining ecosystem

- Mining == gambling:
  - Electricity costs $, huge payout, low probability of winning
- Development of mining pools to ***amortize risk***
  - Pool computational resources, participants "paid" to mine e.g., rewards "split" as a fraction of work, etc
  - Verification? Demonstrate "easier" proofs of work to admins
  - Prevent theft? Block header (coinbase txn) given by pool

More than just a currency…

BLOCKTECH in FINANCIAL SERVICES VIRTUALscape

by William Mougayar

APPLICATIONS & SOLUTIONS

Brokerage
coinbase, BITPagos, Unocoin, BTCC, BITFINEX, CIRCLE, COINJAR, QUADRIGACX, bitFlyer, safello, volabit, coinfloor, coins.ph

Exchanges
BTer.com, coinbase, kraken, HUOBI.com, BITSTAMP, POLONIEX, BTC-e, bitcoin.de, GEMINI, mexbt, CAMP BX, BITSO, Coinffeine, PAYMIUM, CEX.IO, BitOasis, SHAPESHIFT, BTC express, coinsecure, coinsetter, Bits of Gold

Soft Wallets
BLOCKCHAIN, airBitz, coinbase, ARMORY, ELECTRUM, xapo, bread wallet, Coinkite, Mycelium, MultiBit HD, coinprism

Hard Wallets
TREZOR, case, Ledger Wallet, keepkey

Microtransactions
BitMesh, BitWall, ChangeTip, ProTip, Strawpay

Investments
Grayscale, magnr, Loanbase, string, Yuanbao, KOIBANX, Bitbond, WeiFund, WEALTHCOIN, lighthouse, BSAVE.IO, dangpu.com, BTCjam, CHROMA.FUND

Merchants
bitpay, Bitnet, Coinkite, Coinify, PEY, CoinPayments, coinsnap, coinbase, CoinSimple, BITPagos

Compliance
ELLIPTIC, thirdkey solutions, PROTUS, CHAINALYSIS, Sig3, BLOCKSEER, CryptoCorp, IdentityMind GLOBAL, Tradle, vogogo, COINALYTICS, BLOCKVERIFY, Merkle Tree

Trading Platforms
COINIGY, HEDGY, OrderBook, tradewave, COINUT, AltOptions, MAKER, BITNOMIAL, TERA EXCHANGE, BitMEX, Mirror, CRYEX, 1Broker, TABTRADER, DXmarkets, AlphaPoint, NOBLE MARKETS, HitFin

Capital Markets
Chain, symbiont, NASDAQ Private Market, Digital Asset Holdings, clearmatics, itBit, TradeBlock, t0, epiphyte

Money Services
CRYPTOPAY, cashila, ABRA, Fuzo, tether, BitwalaBTC, coins.ph, Simplex, ATLAS, BITX, coinx, REBIT, uphold, SecuraCoin, DUO MONEY, BITNEXO, CoinPip, LocalBitcoins.com, BitPesa, BlinkTrade, COINAPULT, MELOTIC, Glidera, bridge21

Financial Data
bitcoinity, CoinMarketCap, CryptoCoin, BRAVE NEW COIN, BlockJockey, CRYPTRADER, BitcoinWisdom, TradeBlock, CoinGecko, Coinhills

Payments
Align Commerce, GOCOIN, About Payments, BLADE, GAZEBO.IO, GemPay, cuber, SETL.io, safecash

ATMs
LocalBitcoins.com, Robocoin, bitxatm, bitaccess, Project Skyhook, btcpoint, SERY, LAMASSU BITCOIN VENTURES, GB, COINOUTLET, genesiscoin, Modenero Concierge

Trade Finance
GAZEBO.IO, everledger, CHRONICLED, WAVE, digix, skuchain, PROVENANCE, thingchain

Payroll & Insurance
paybits, bitWAGE, DYNAMIS

Banks
BBVA, UBS, LHV, London Stock Exchange, secco, BNY MELLON, BARCLAYS, fidor BANK, citibank, moni

MIDDLEWARE & SERVICES

Services
CRYPTONOMEX, B9, CONSENSYS, SolidX, appliedblockchain, RUBIX

Software Development
chainscript, HydraChain, Blockstack.io, PEERNOVA, openchain, CREDITS, eris INDUSTRIES, MultiChain, Manifold Technology, Blockstream

General APIs
BitGo, neuroware, coinbase, bitcore, Gem, BLOCKCYPHER, Coinkite

Special APIs
TIERION, Open Assets, bitbind.io, COLOREDCOINS, colu, factom, ChromaWay

Platforms
Counterparty, Monetas, Omni, Mercoin, blockstack, HYPERLEDGER, Tendermint, BLOCKAPPS, appliedblockchain

Smart Contracts
SmartContract, ETH BaaS, CoinSpark, ROOTSTOCK, bitShares, Tembusu Systems

INFRASTRUCTURE & BASE PROTOCOLS

Public
bitcoin, bitShares, ethereum

Special
ripple, stellar

Payment
Amiko Pay, MONERO, Lightning Network

Miners
ANTPOOL, BitFury, 21 INC, BTCC, BITCOINCZ

Source: http://startupmanagement.org/blog