CS251 Fall 2020

(cs251.stanford.edu)

Large Scale Consensus:

Availability/Finality, Randomness Beacons, VDFs

Benedikt Bünz

Blockchain Consensus

Consistency (Safety)

For all honest nodes $i, j \in [n]$ and times t, t':

Either list $L_i(t)$ is a prefix of $L_j(t')$ or vice versa

Δ −Liveness

There exists function T such that:

If any honest node receives tx at time t then $\forall i \;\; tx \in L_i\big(t + \mathrm{T}(\Delta, \mathrm{n})\big)$. At time $t + \mathrm{T}(\Delta, \mathrm{n})$ tx is *finalized*

$\Delta = maximum\ network\ delay$

Two additional features

Finality

Anyone can verify that a transaction is *finalized.*

-> There are no deep forks

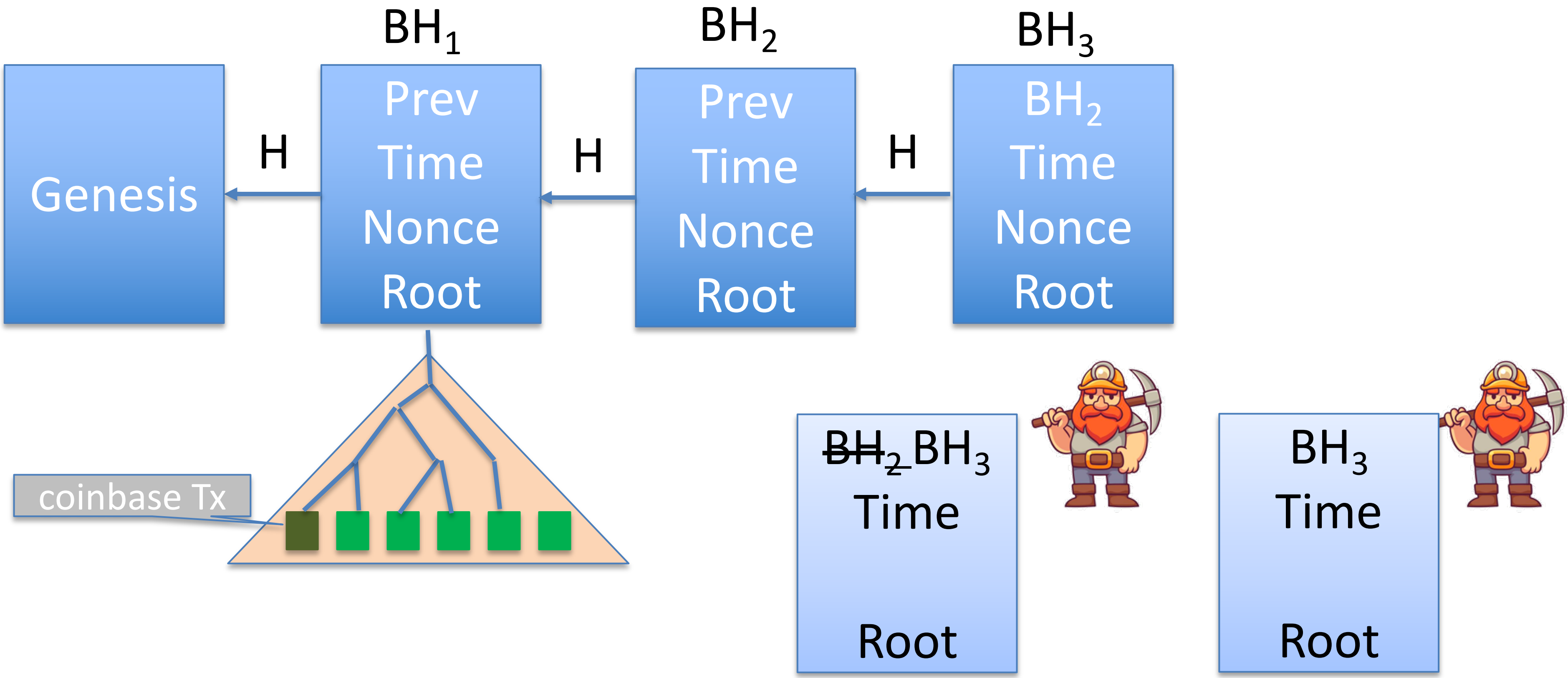
Dynamic –Availability

Chain makes progress even under network partitions.

->The chain keeps growing even if it forks

->Nodes can leave and join the network

Recap: Nakamoto Consensus



BH_1 BH_2 BH_3

Genesis

H

Prev
Time
Nonce
Root

H

Prev
Time
Nonce
Root

H

BH_2
Time
Nonce
Root

coinbase Tx

~~BH_2~~ BH_3
Time
Root

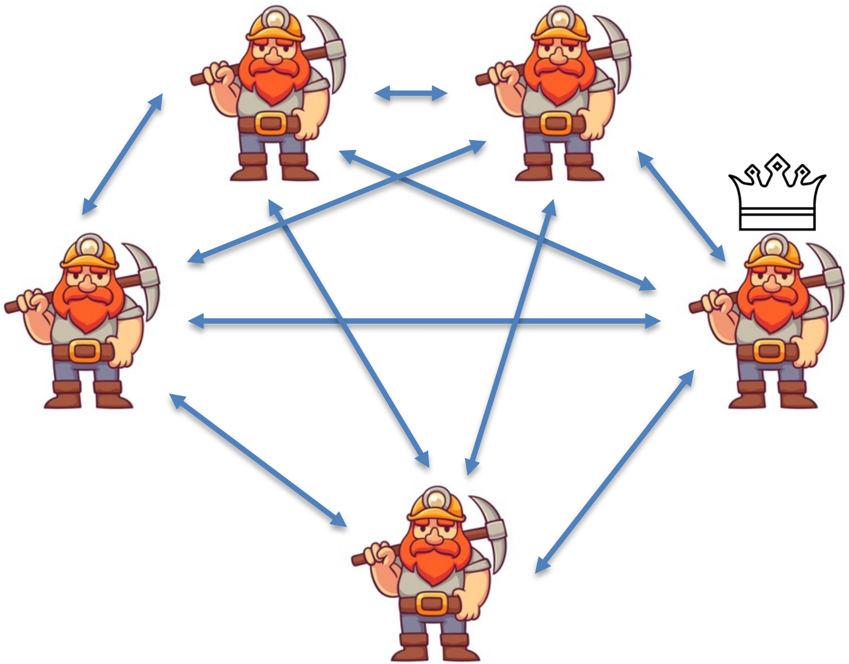
BH_3
Time
Root

Nakamoto Properties

- Anonymous participation
- Nodes can join/leave
  - Very scalable
  - *Dynamic availability*
- Leader not known beforehand
  - Makes bribing harder
- Up to ½ corruptions

- Slow
  - Even when everyone is honest
- Resource intensive
  - PoS based possible
- Long forks possible
- No guarantees under long delays
- *No finality*

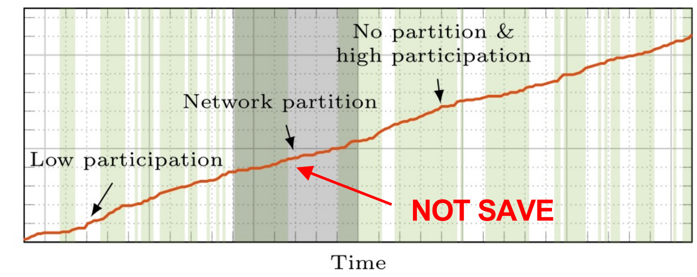
Recap Byzantine Consensus



- Fast
- Partially Synchronous
- Halts under network partition
- Provides *finality*
- Known committee
    - (must communicate)
- Large committee
    - Large communication
- Predictable Leader
    - Bribing 💸

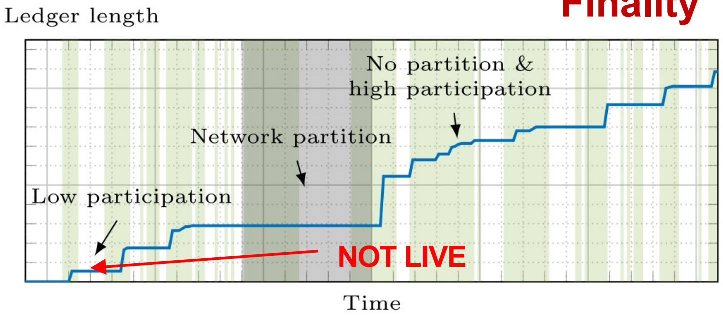
Nakamoto vs BFT under network outage

Dynamic availability



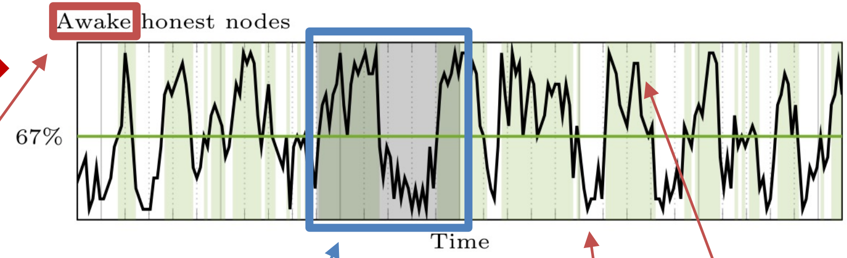
NOT SAVE

Finality



NOT LIVE

Nakamoto Consensus



BFT protocol

DYNAMIC PARTICIPATION

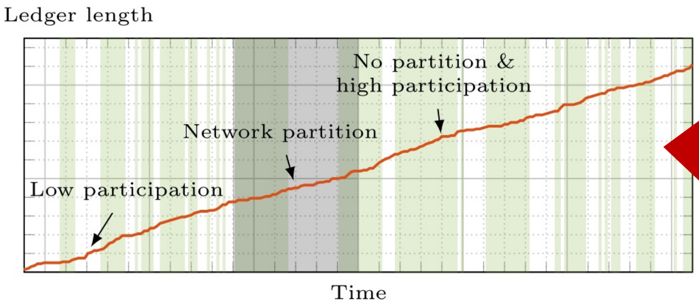
NETWORK PARTITIONS

LOW PARTICIPATION

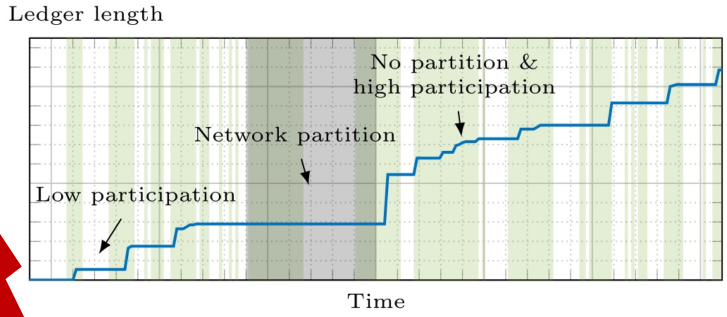
HIGH PARTICIPATION

From [NTT21]

Availability and Finality [Gilbert, Lynch ’02,Lewis-Pye, Roughgarden ‘20]



Dynamic availability

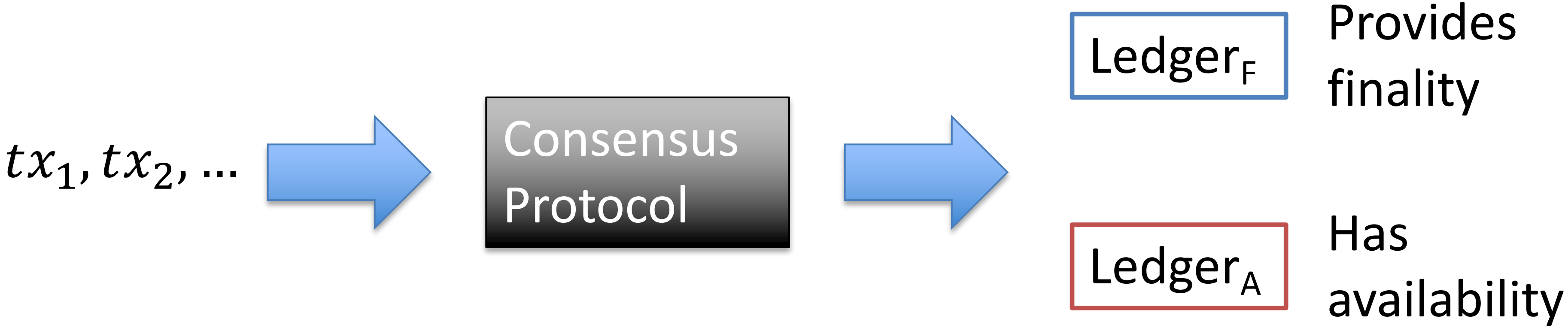


Finality

NO!

Is there a consensus protocol that provides **both** availability and finality?

Resolving the dilemma



$tx_1, tx_2, \ldots$

Consensus Protocol

Ledger$_F$

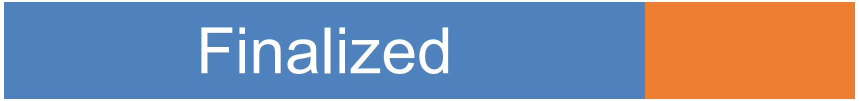
Provides finality

Ledger$_A$

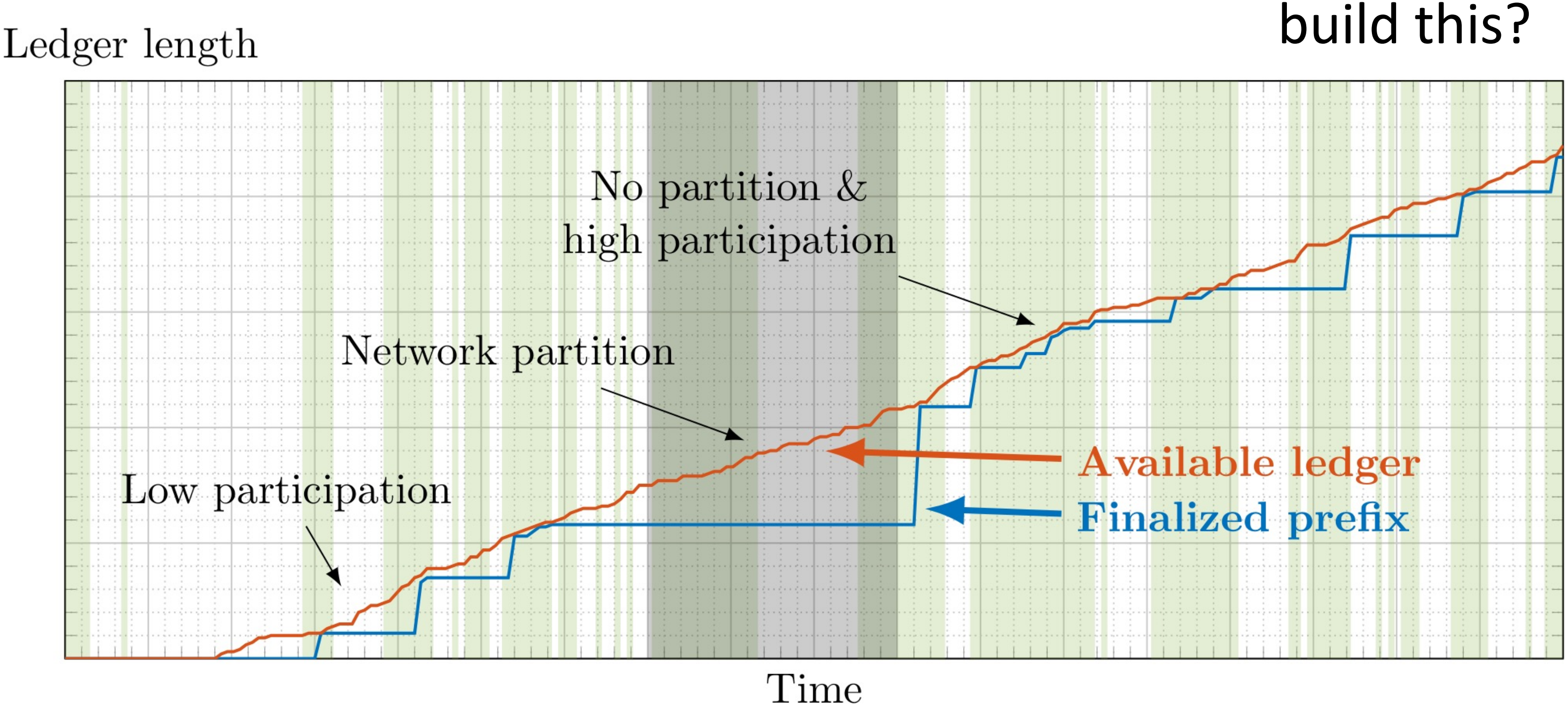
Has availability

Prefix condition: $\boxed{\text{Ledger}_F} < \boxed{\text{Ledger}_A}$

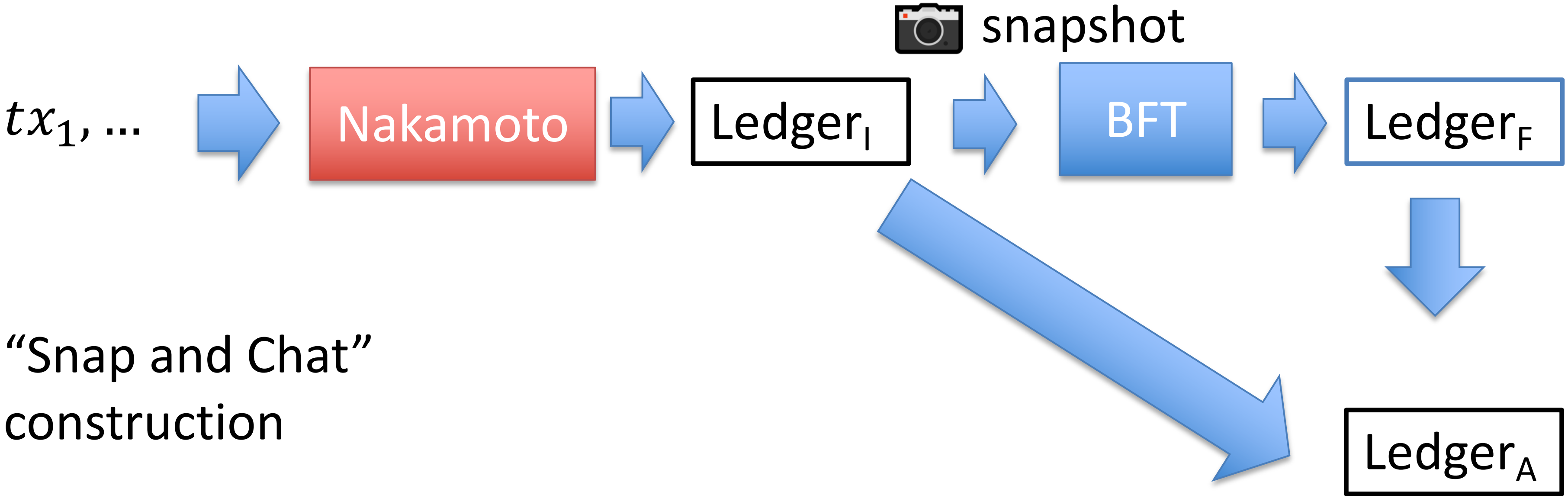
Ebb and Flow protocol [NTT21]



How do we build this?



Building Ebb and Flow [NTT21]



“Snap and Chat” construction

“Sanitized” availability ledger

Ensures prefix

Ethereum 2.0

Ethereum currently uses PoW Nakamoto Consensus
Since last year there exists a separate PoS chain
The two chains will merge and PoW will be deactivated

PoS chain uses a snap and chat style protocol

- 12s block time
- 1 epoch is 32 blocks (6.4 minutes)
- Finalization in 2 epochs (~13 minutes)

ethereum 2.0

Proof of Stake

Replace Sybill resistance of PoW with money

Stakes coins (through transaction)

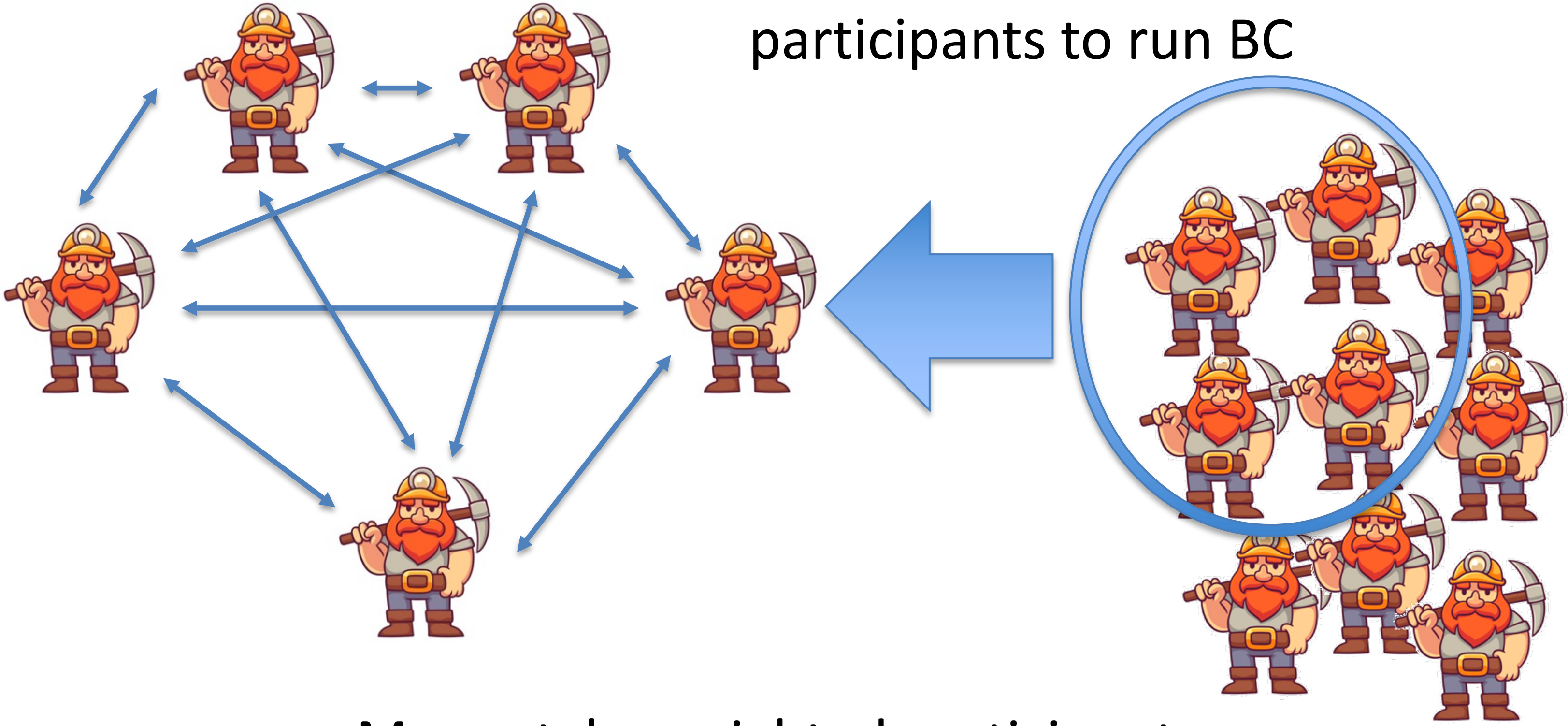
Can’t use staked coins for anything else!

Staking pool

Incentives: Get’s rewards/fees. Can use punishments/slashing

Voting Power: Proportional to relative stake

Scaling Byzantine Consensus

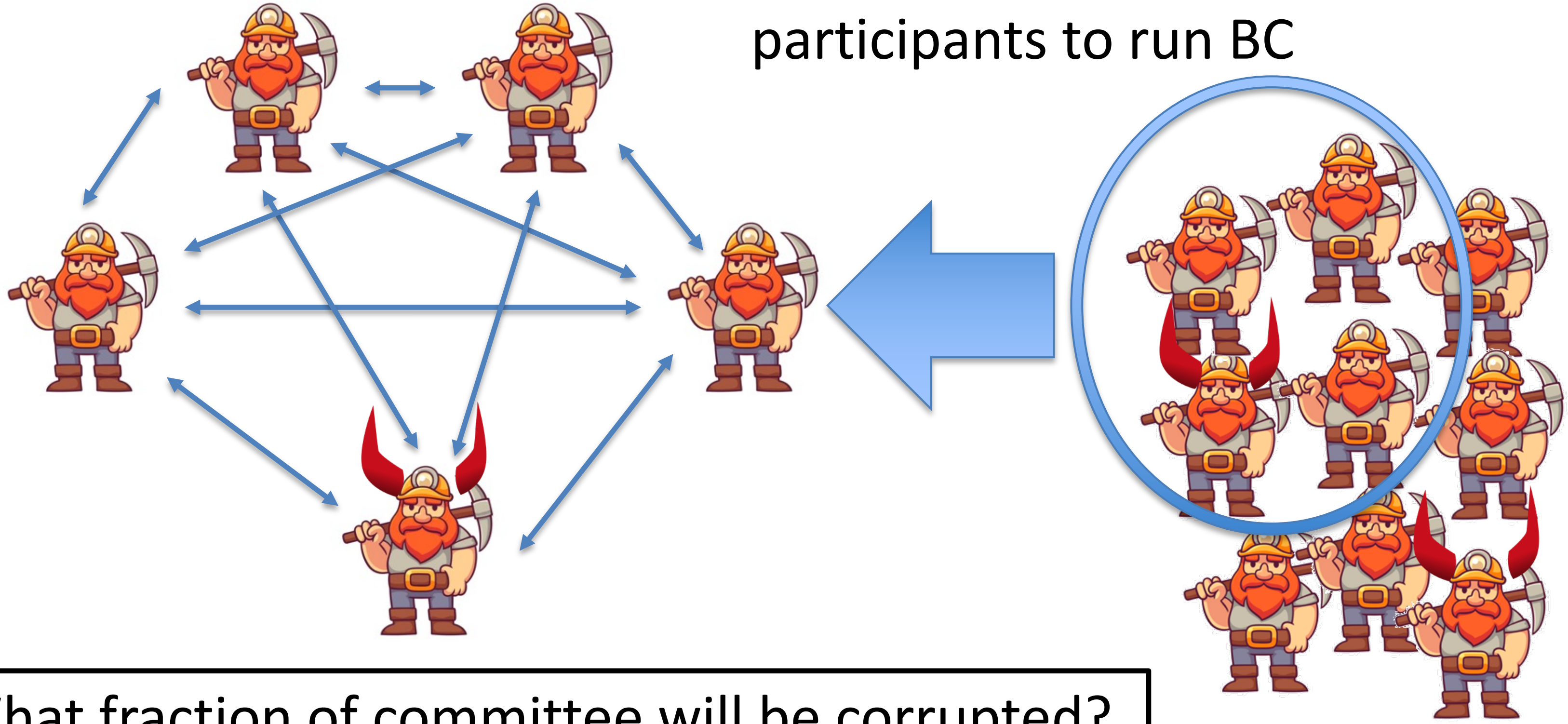


Sub select a set of participants to run BC

Many stake weighted participants

Committee selection

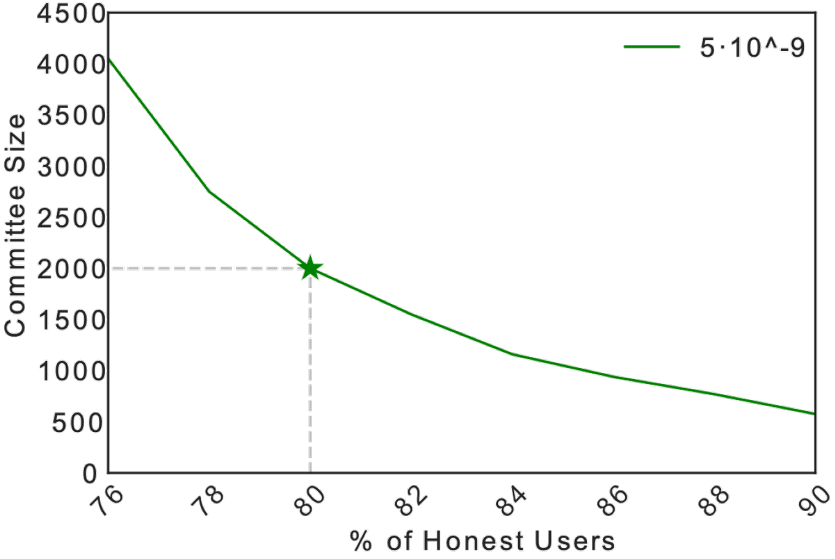
Sub select a set of participants to run BC

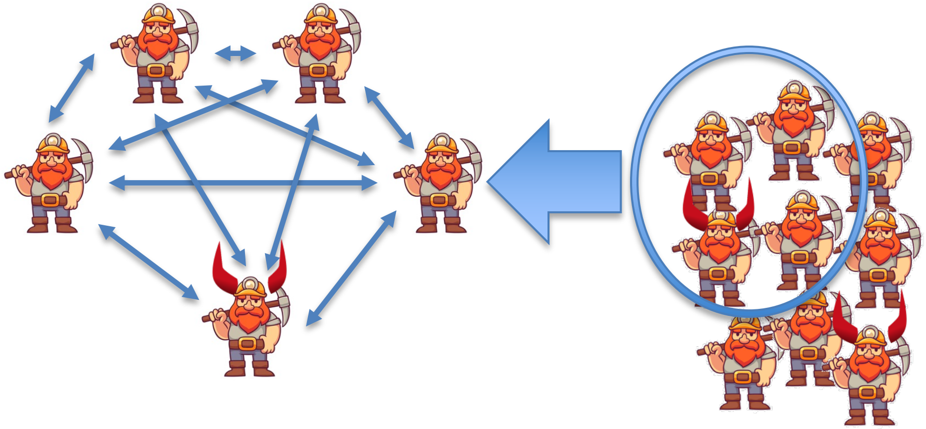


What fraction of committee will be corrupted?

Committee selection

Sub committee roughly looks like general population





100s of nodes
>67% Honest

>1000s of nodes
80% Honest

Random Selection

How to choose committee?

Proposal:

- Each staker computes H(block number,PK)
- If H(block number,PK)< target
    - Become part of committee for round
- If BC succeeds add Block to chain
- Target such that ~1000 nodes win

Broken! Attacker can choose PK such that they win

Randomness beacon

An ideal service that regularly publishes random value which no party can ***predict*** *or* ***manipulate***

01010001 01101011 10101000 11110000

Random Selection with Beacon

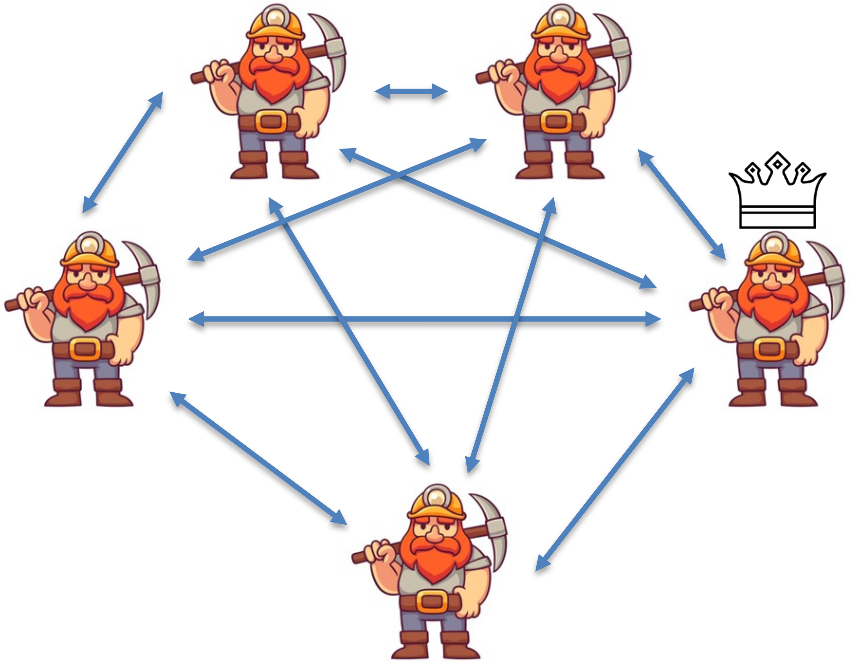
How to choose committee?

- Each Block wait for beacon randomness
- Each staker computes H(~~block number~~ beacon, PK)
- If H(~~block number~~ beacon,PK)< target
  - Become part of committee for round
- If BC succeeds add Block to chain

Beacon unpredictable so can’t choose PK

Even better: Compute deterministic (BLS) signature on Beacon and use as ticket (prevents others from seeing who won) VRF

Leader Selection



We can also make leader election random with a beacon!

Can make BC resilient vs. adversary that corrupts *adaptively* (Bribing)

See Algorand reading

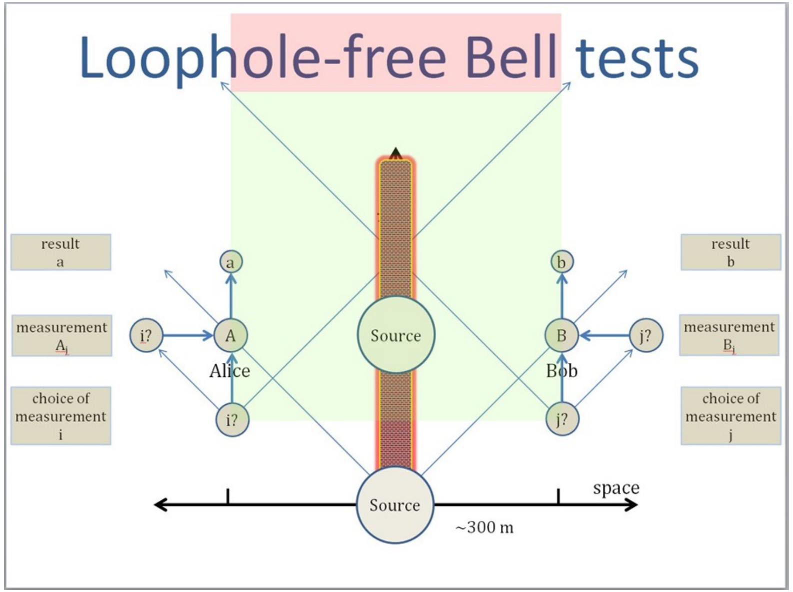
Lotteries

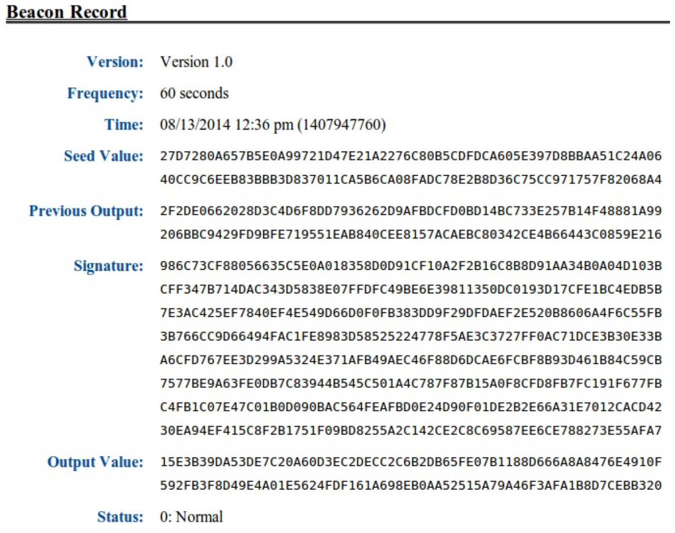
``Public displays" can be corrupted

A beacon can be used to run a fair lottery

How to build a Beacon?

NIST (NSA) Beacon

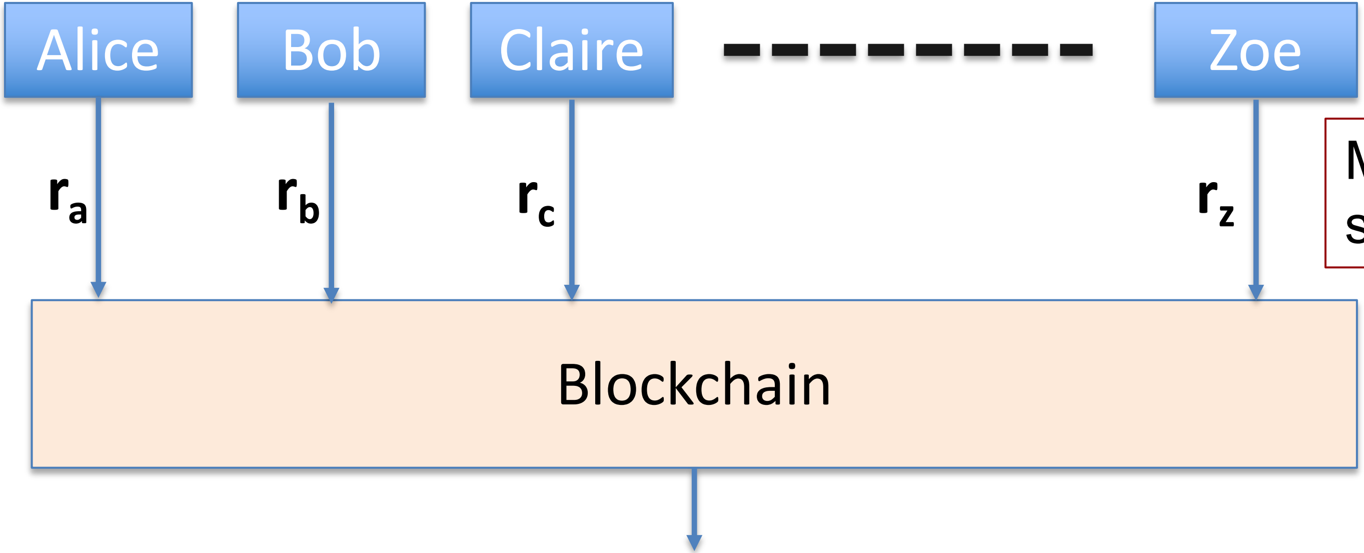




Beacon Record

Version:	Version 1.0
Frequency:	60 seconds
Time:	08/13/2014 12:36 pm (1407947760)
Seed Value:	27D7280A657B5E0A99721D47E21A2276C80B5CDFDCA605E397D8BBAA51C24A06 40CC9C6EEB83BBB3D837011CA5B6CA08FADC78E2B8D36C75CC971757F82068A4
Previous Output:	2F2DE0662028D3C4D6F8DD7936262D9AFBDCFD0BD14BC733E257B14F48881A99 206BBC9429FD9BFE719551EAB840CEE8157ACAEBC80342CE4B66443C0859E216
Signature:	986C73CF88056635C5E0A018358D0D91CF10A2F2B16C8B8D91AA34B0A04D103B CFF347B714DAC343D5838E07FFDFC49BE6E39811350DC0193D17CFE1BC4EDB5B 7E3AC425EF7840EF4E549D66D0F0FB383DD9F29DFDAEF2E520B8606A4F6C55FB 3B766CC9D66494FAC1FE8983D58525224778F5AE3C3727FF0AC71DCE3B30E33B A6CFD767EE3D299A5324E371AFB49AEC46F88D6DCAE6FCBF8B93D461B84C59CB 7577BE9A63FE0DB7C83944B545C501A4C787F87B15A0F8CFD8FB7FC191F677FB C4FB1C07E47C01B0D090BAC564FEAFBD0E24D90F01DE2B2E66A31E7012CACD42 30EA94EF415C8F2B1751F09BD8255A2C142CE2C8C69587EE6CE788273E55AFA7
Output Value:	15E3B39DA53DE7C20A60D3EC2DECC2C6B2DB65FE07B1188D666A8A8476E4910F 592FB3F8D49E4A01E5624FDF161A698EB0AA52515A79A46F3AFA1B8D7CEBB320
Status:	0: Normal

Collect randomness approach

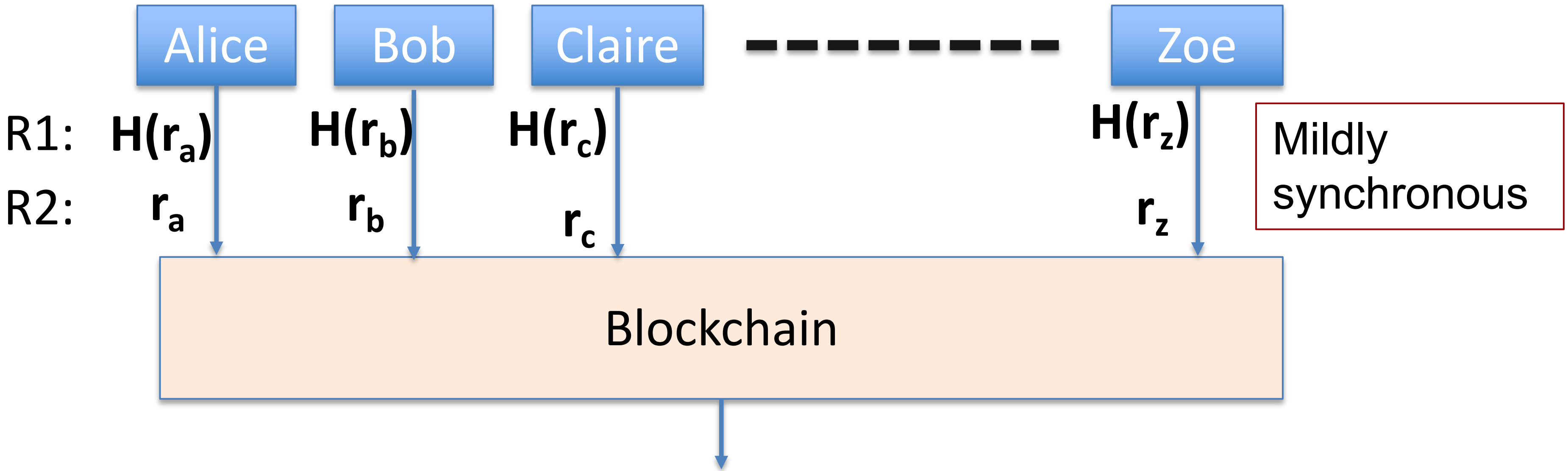


Mildly synchronous

output $\textbf{beacon = Hash}(r_a \,||\, r_b \,||\, \cdots \,||\, r_z) \in \{0,1\}^{256}$

Problem: Zoe controls the final seed !!

Commit and Reveal

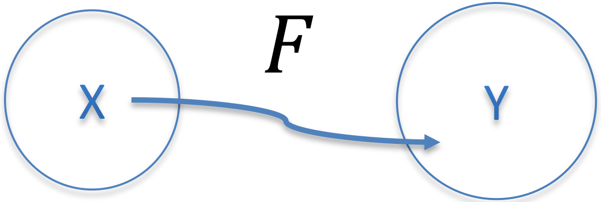


output **beacon = Hash(r_a || r_b || ⋯ || r_z)** $\in \{0,1\}^{256}$

Problem: Beacon can be biased by not opening!!
K parties, k bits of influence

Verifiable Delay Function (VDF)

- **Function** – unique output for every input



- **Delay –** can be evaluated in time T

  cannot be evaluated in time (1-ϵ)T on parallel machine

- **Verifiable** – correctness of output can be verified efficiently

x, y, π
$F(x)$
✓
Verifier

Security Properties (Informal)

- Setup(λ, T) ⟶ public parameters *pp*
- Eval(*pp*, ***x***) ⟶ output ***y***, proof $\boldsymbol{\pi}$ (requires *T* steps)
- Verify(*pp*, ***x***, ***y***, $\boldsymbol{\pi}$) ⟶ { *yes*, *no* }

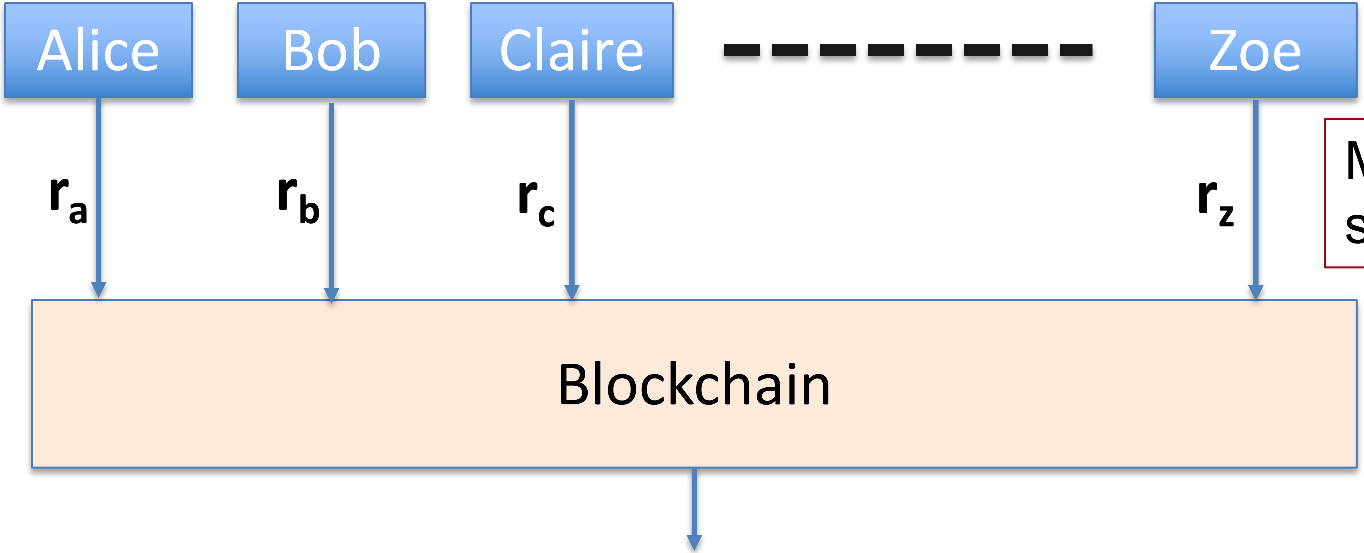
"**Soundness**": if Verify(*pp*, *x*, ***y***, $\boldsymbol{\pi}$) = Verify(*pp*, *x*, ***y'***, $\boldsymbol{\pi'}$) = yes

then ***y* = *y'***

"σ-**Sequentiality**": if A is a PRAM algorithm, time(*A*) $\leq \sigma(T)$,

e.g. $\sigma(T) = (1-\epsilon)T$ then Pr[***A*(*pp*, *x*) = *y***] < negligible(λ)

Collect randomness approach

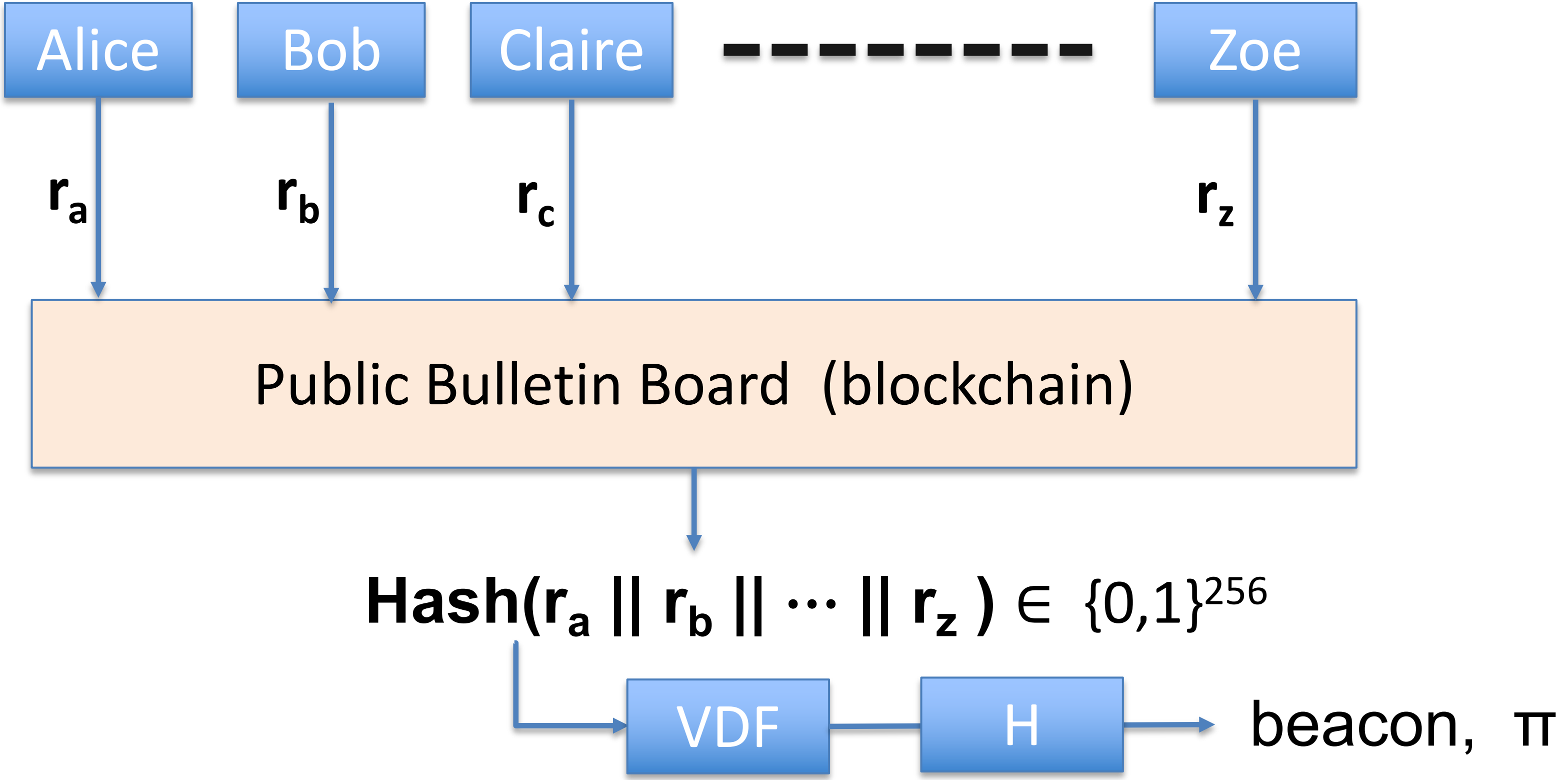


Mildly synchronous

output $\textbf{beacon} = \textbf{Hash}(r_a \,||\, r_b \,||\, \cdots \,||\, r_z) \in \{0,1\}^{256}$

Problem: Zoe controls the final seed !!

Solution: slow things down with a VDF [LW'15]



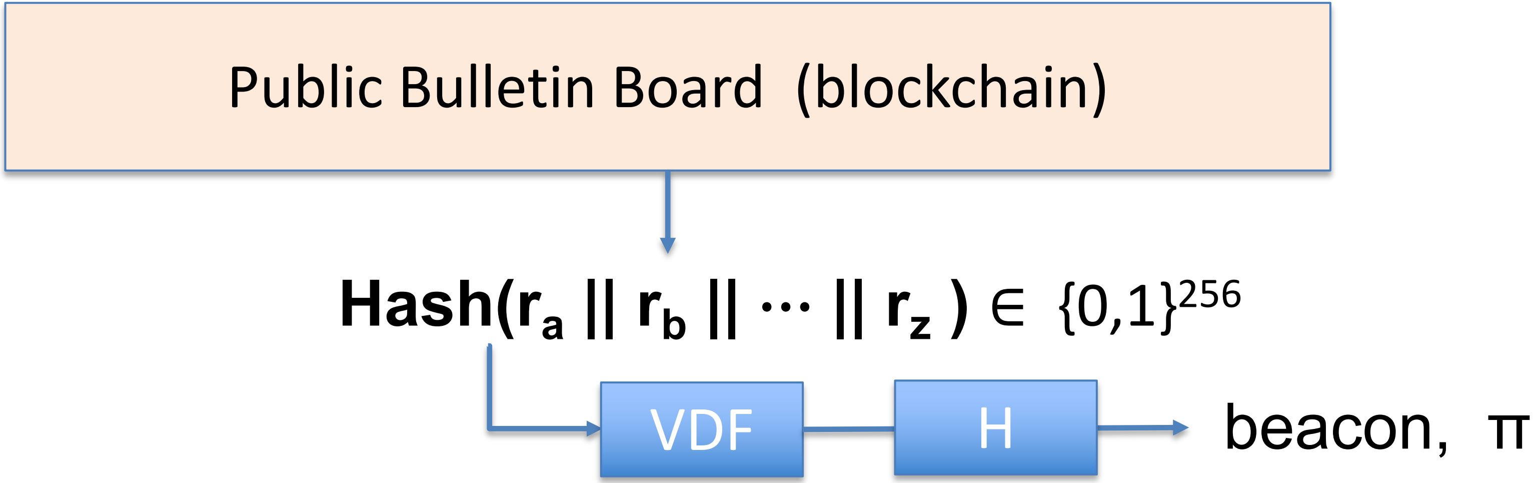
Hash($r_a \,||\, r_b \,||\, \cdots \,||\, r_z$) $\in \{0,1\}^{256}$

beacon, π

Solution: slow things down with a VDF [LW'15]

VDF delay ≫ max-Δ-time(Alice ⟶ Zoe)

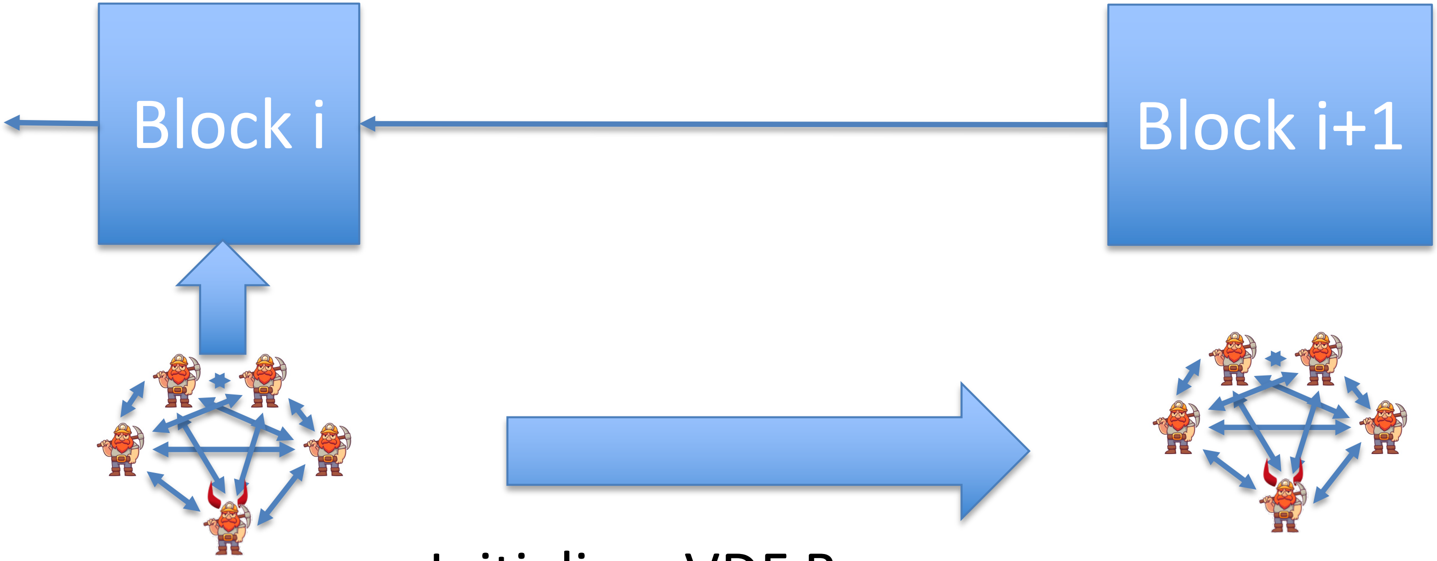
Uniqueness: ensures no ambiguity about output



Public Bulletin Board (blockchain)

$$\mathbf{Hash}(\mathbf{r}_a \,||\, \mathbf{r}_b \,||\, \cdots \,||\, \mathbf{r}_z\,) \in \{0,1\}^{256}$$

VDF Beacon in a blockchain



Block i

Block i+1

Committee i

Initializes VDF Beacon

Committee i+1

How to build a VDF

Choose a “Group of unknown order”:

- Pick two primes p,q, Let $N = p \cdot q$
- Computing $g^{2^T}\ mod\ N$ takes T repeated squarings
  - Can’t be parallelized
  - Unless factorization of N is known
- Let H be a hash function that maps to $[0, N-1]$

Eval(pp, x): output $H(x)^{2^T}$

How to verify?

VDF Proof

Efficiency: Bob runs in time O(log(T))

Security: If Bob accepts then $y = H(x)^{2^T}$

Computes y $= H(x)^{2^T}$

Produces a small proof π

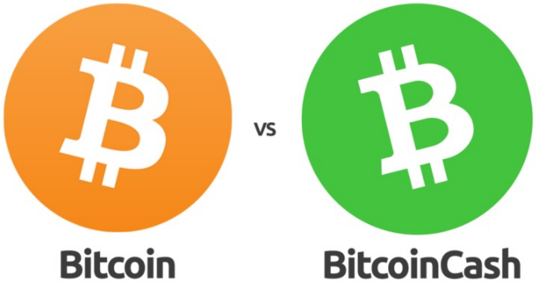
Sends y, π to Bob

Takes as input x, y, π

Outputs “accept” or “reject”

Changing the rules/Governance

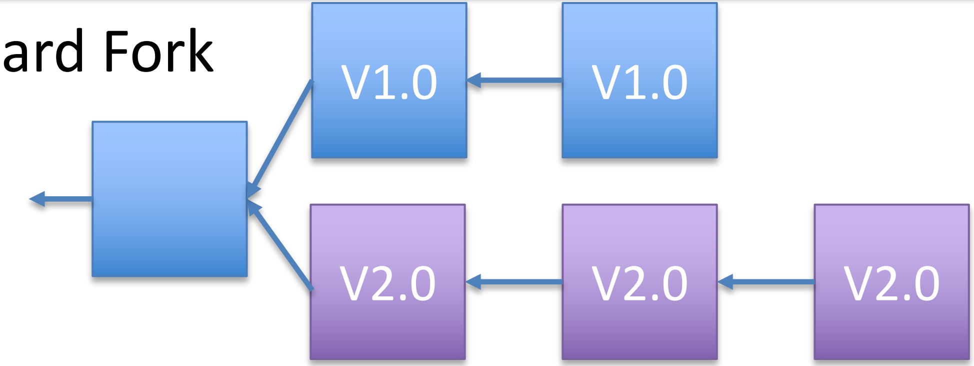
- Protocol upgrades
  - New Transaction types (Add Smart Contracts)
  - New Consensus (Switch from PoW to PoS)
  - Increase Blocksize (1MB) Bitcoin/Bitcoin Cash



- How do we reach consensus on these things

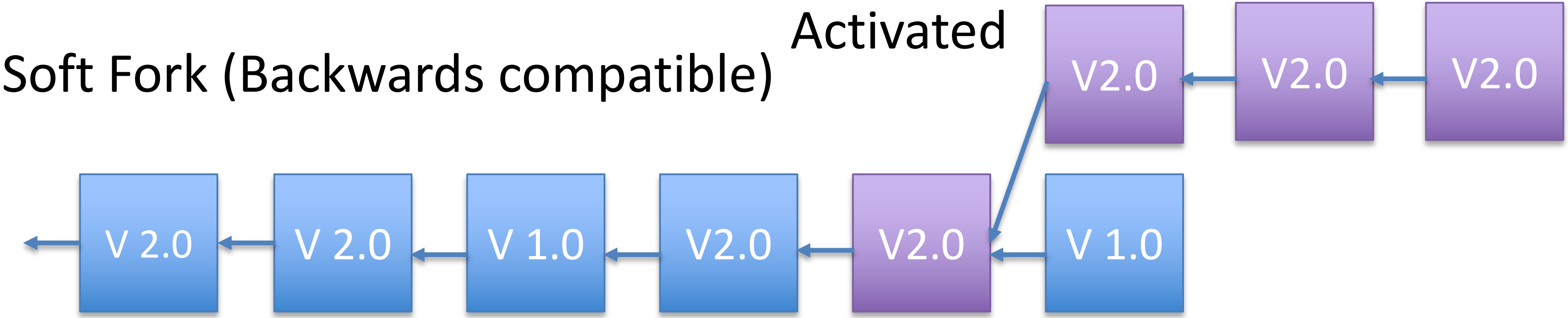
Soft/Hard Fork Activiation

Hard Fork



Soft Fork (Backwards compatible)

Activated



Hard Forks

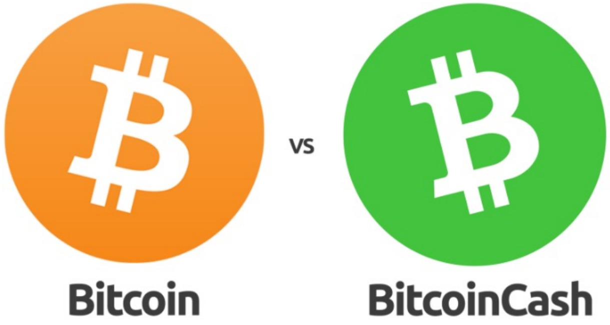
- Technically the simplest
- New protocol version (new software)
- Everyone upgrades
- New protocol incompatible with old protocol
- Everyone needs to upgrade
- Ethereum/Zcash/Monero do this semi regularly
- *Contentious* Hard Fork: Both versions exists
  - Need to worry about replay attacks

Ethereum Ethereum Classic

Soft Forks

- Rules become more restrictive
  - Disabling old OP_CODES
  - Further specifying signatures (ECDSA)
- Old clients still work but their transactions may get rejected
- If >50% upgrade then new rules enforced
- Segregated Witness was a contentious soft fork

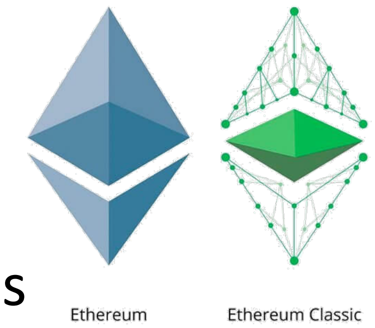
Case Study: Bitcoin vs Bitcoin Cash



- Bitcoin Blocks are limited to 1MB
- ~Roughly 7 tx/s
- Proposal to increase block size
- Opinion 1: “Larger blocks increase network delay, decreases security, transactions should be moved off the chain.”
- Opinion 2: “Bitcoin can support more transactions we should increase block size.”
- Split in 2017: Every Bitcoin user got same amount of Bitcoin Cash (sum is less than sum of parts).

Case Study: Ethereum vs. Classic

- Ethereum had a smart contract called *DAO*
- Smart contract had a bug
- July 2016, $50 Million USD of Ether stolen
- Proposal to hard fork Ethereum and return funds
- Stake vote was held
    - 87% in favor but only 5.5% participated
    - 4 days later Ethereum forked
    - “Classic” is the old version including stolen funds
- Ethereum Foundation owns trademark and branded Fork Ethereum
- Later more divergence: Ethereum will move to PoS, Classic stay on PoW



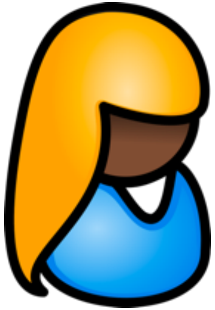
END OF LECTURE

Next lecture:

Ethereum and Smart Contracts

VDF Proof [Wesolowski'18]

$(x, y, T) : x^{2^T} = y$



y

Random λ bit prime l

$\pi = x^q$

Computes
q,r s.t.
$2^T = q \cdot l + r$
and $0 \leq r < l$

Computes
$r = 2^T \ mod \ l$
Checks:

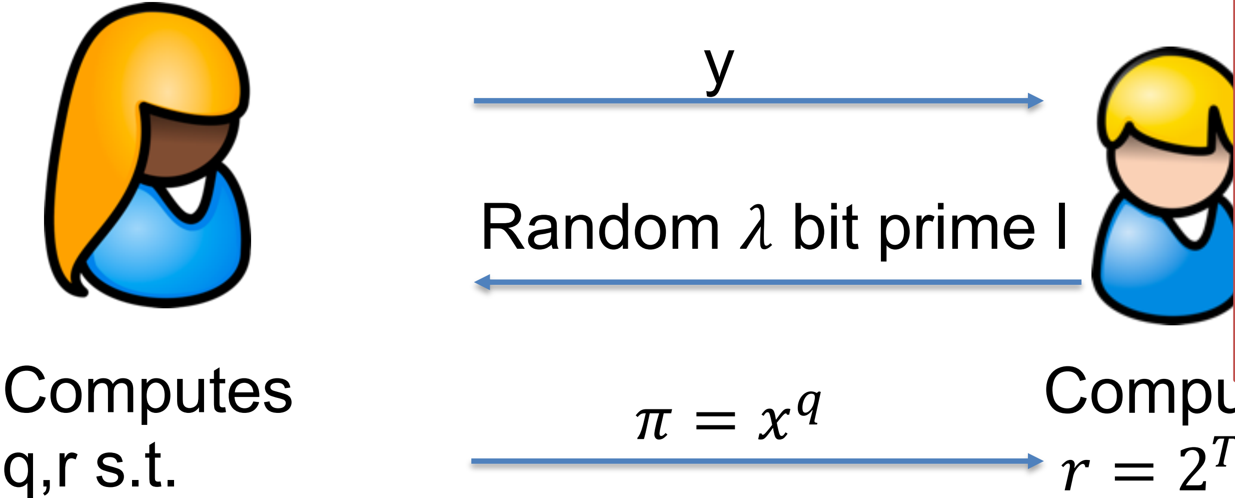
$$\pi^l x^r = y$$

$$x^{q \cdot l} x^r = x^{2^T}$$

$\log(T)$steps

Security intuition

$(x, y, T): x^{2^T} = y$



y

Random λ bit prime l

$\pi = x^q$

Must compute π

s.t. $\pi = \left(\frac{y}{x^r}\right)^{\frac{1}{l}}$

Taking roots is hard

See reading

Computes

q,r s.t.

$2^T = q \cdot l + r$

and $0 \leq r < l$

Computes

$r = 2^T \bmod l$

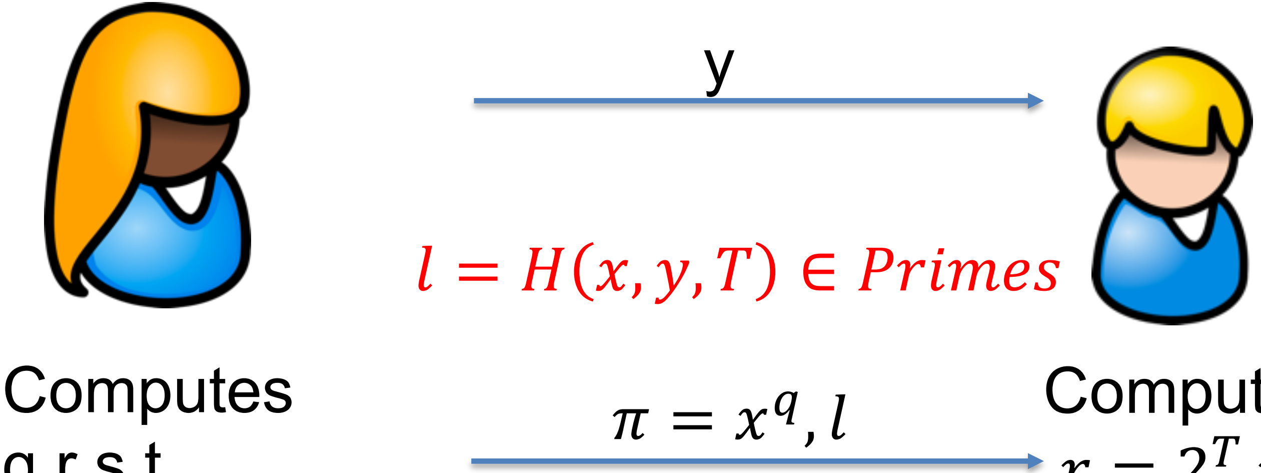
Checks:

$$\pi^l x^r = y$$

$$x^{q \cdot l} x^r = x^{2^T}$$

VDF Proof [Wesolowski'18]

$(x, y, T): x^{2^T} = y$



y

$l = H(x, y, T) \in Primes$

$\pi = x^q, l$

Computes
q,r s.t.
$2^T = q \cdot l + r$
and $0 \le r < l$

Computes
$r = 2^T \bmod l$
Checks: $l = H(x, y, T)$
$\pi^l x^r = y$
$x^{q \cdot l} x^r = x^{2^T}$