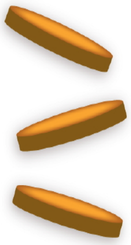
Byzantine agreement in the Clear

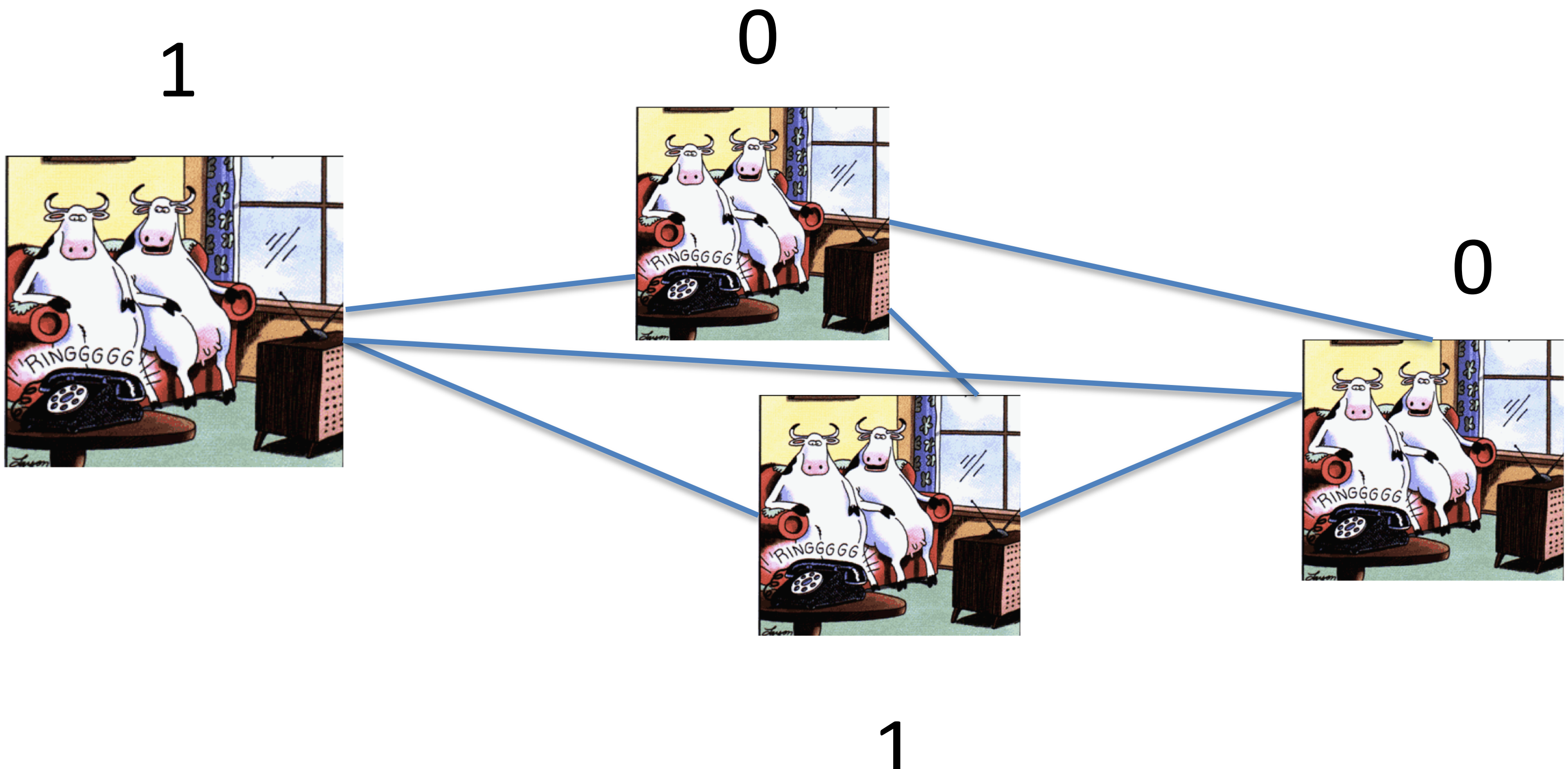


Valerie King

University of Victoria

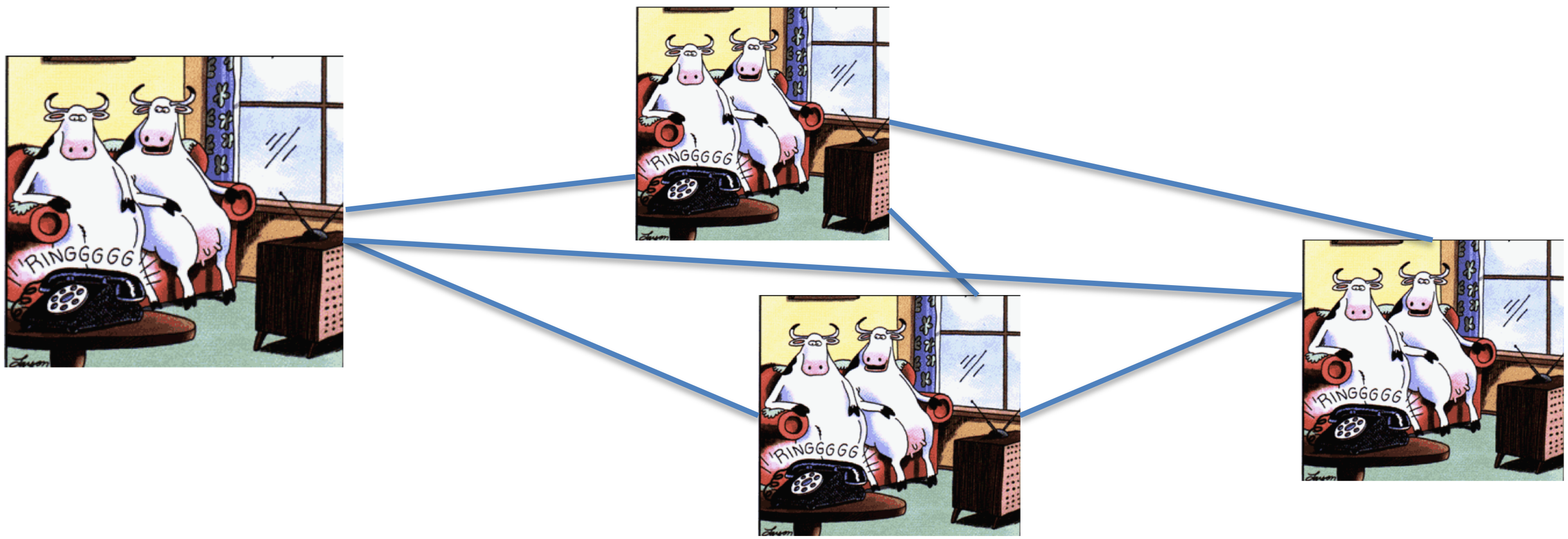
Victoria, Canada

Byzantine Agreement



Start with initial bits; exchanges messages, then output same bit. If all start with the same bit, must output that bit

Byzantine Agreement



To model worst case faults in processors which communicate via point-to-point links and worst case delays in message delivery

Today: Need for decentralized agreement over the internet with untrusted players

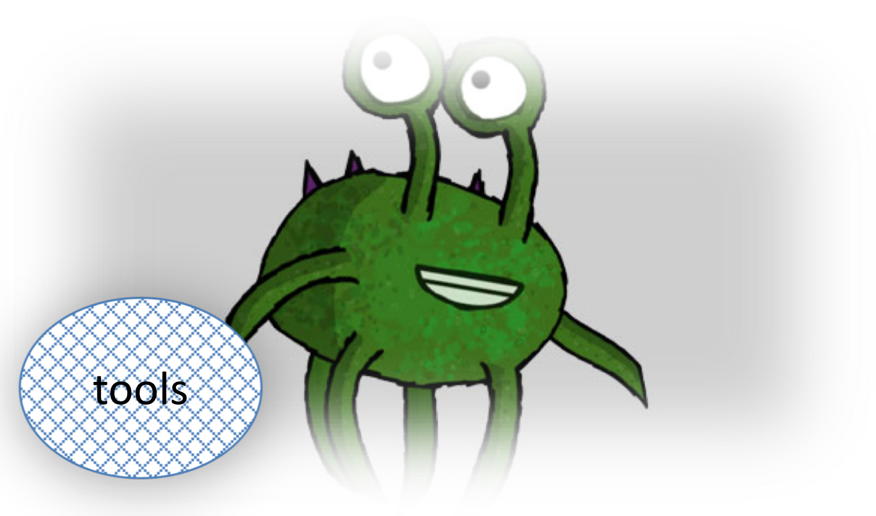
Distributed ledger:

- Digital currency
- Smart contracts

"*Bitcoins*? Do you take me for a fool - I want magic beans."

Goal of this talk

Byzantine agreement



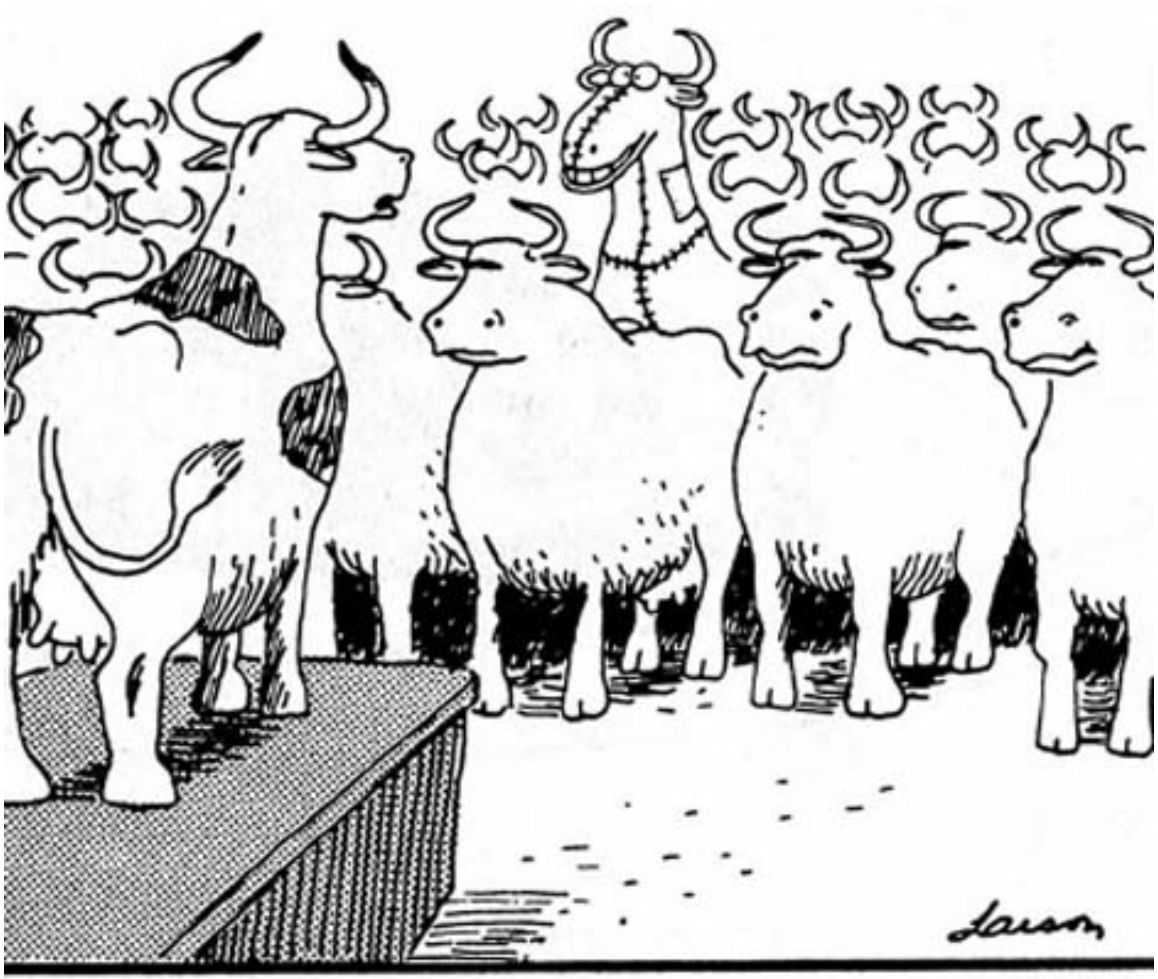
Decentralized ledger

Byzantine adversary

n nodes

t <n/3 bad behave arbitrarily

Worst case input



"The revolution has been postponed . . . We've discovered a leak."

Asynchronous Communication

Adversary schedules message delivery, no global clock, no known delay bounds

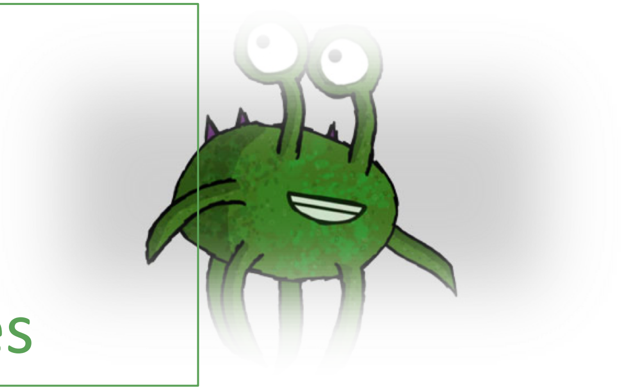
→ *Can't wait to hear from >n-t before taking next action*

Asynchronous Communication

Adversary schedules message delivery, no global clock

→ *Can't wait to hear from >n-t before taking next action*

Do we care about this?
If we assume this, can't use computation power to bound adversary's ability to solve puzzles



Asynchronous Communication

Adversary schedules message delivery, no global clock

→ *Can't wait to hear from >n-t before taking next action*

Do we care about this?
If we assume this, can't use computation power to bound adversary's ability to solve puzzles

How about assuming bound on Energy (Independent of time)?

Impossibility result

One worst case crash fault makes (deterministic) agreement impossible with asynchrony.

(1982: Fischer, Lynch and Patterson)

There are fast solutions in some cases

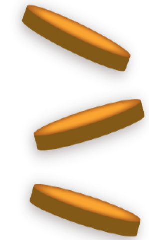
Reliable broadcast:

If a player broadcasts the same transaction
To all players, then all decide in 3 steps
Else possibly no decision

With randomness

- If there's a global coin.
- If there's secret communication between good nodes, e.g. with crypto
- If t is O($\sqrt{n}$)

What kind of randomness?



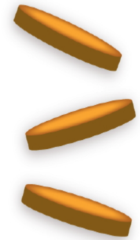
- Global coin

doesn't exist

- Global random oracle:

truly random hash function known to every node, returns a consistent answer.

What kind of randomness?



- Global coin

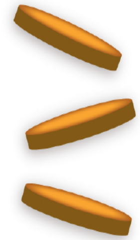
doesn't exist

- Global random oracle:

truly random hash function known to every node, returns a consistent answer.

doesn't exist either

What kind of randomness?



- Global coin

doesn’t exist

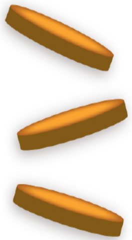
- Global random oracle:

truly random hash function known to every node, returns a consistent answer.

doesn’t exist either

Usual assumption
for setting puzzles,
creating a common coin,

What kind of randomness?



- Global coin

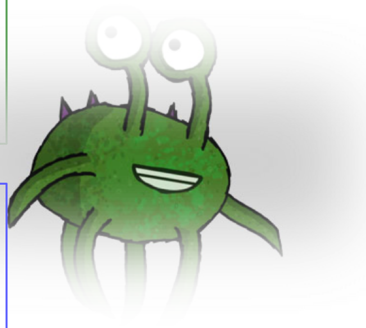
doesn't exist

- Global random oracle:

truly random hash function known to every node, returns a consistent answer.

doesn't exist either

usual assumption
for setting puzzles,
creating a common coin



- **Here**, weaker assumption: private coins

Rest of talk: **In the Clear**

- Adversary can view state of players.
- Randomness: private random bits only
- No cryptographic assumptions, no random oracle, no public key system, “plain model”

But what if we can’t pass messages directly?

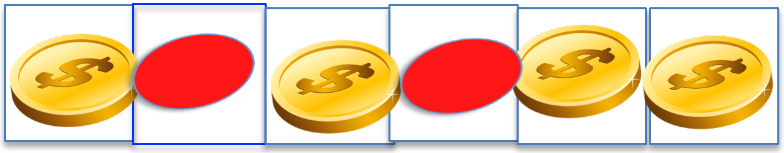
Rest of talk: 2 different ideas

1 The value of a short common string from a bit-fixing source

2 Solving Byzantine agreement in a fully asynchronous environment

Robust to “adaptive adversary”.

Using a O(log n) bit common string



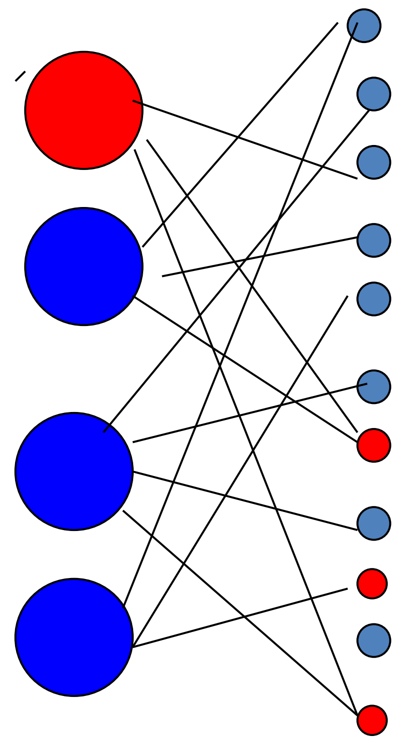
To create a set of n small committees, one for each node, ALL of which are representative, w.h.p.

Used for

- load balancing
- a communication network or distributed hash table with reliable supernodes and
- maintain these over changes to the network by repeatedly choosing strings

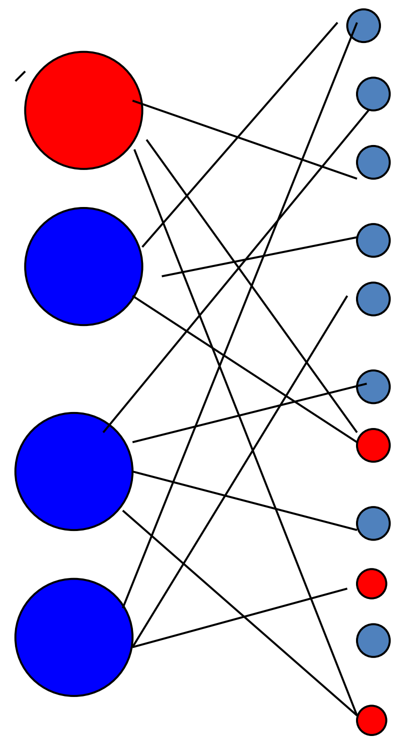
To go from Common String to many, a committee for each node

Create Deterministic Sampler



To go from Common String to many, a committee for each node

Create Deterministic Sampler

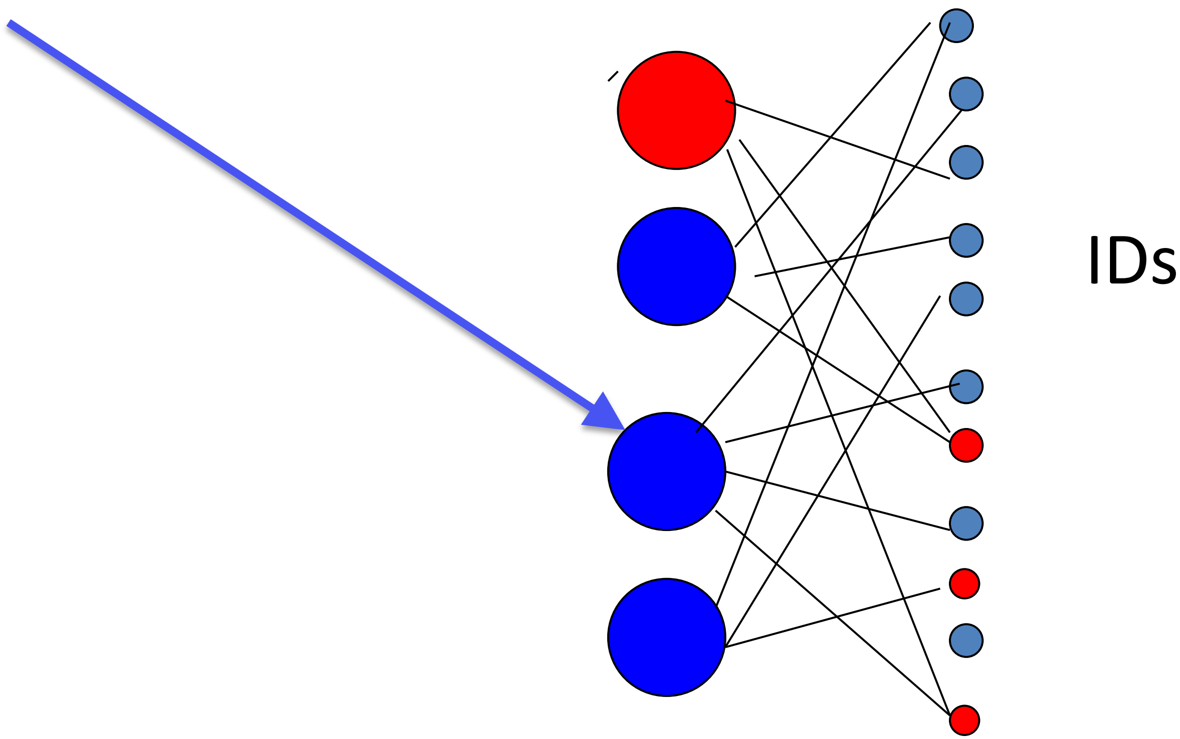


Is this constructive? Can each node determine its neighbors quickly?

To go from short Common String to a committee for each node:

Committee is indexed by (Common String, node ID)

Create Deterministic Sampler

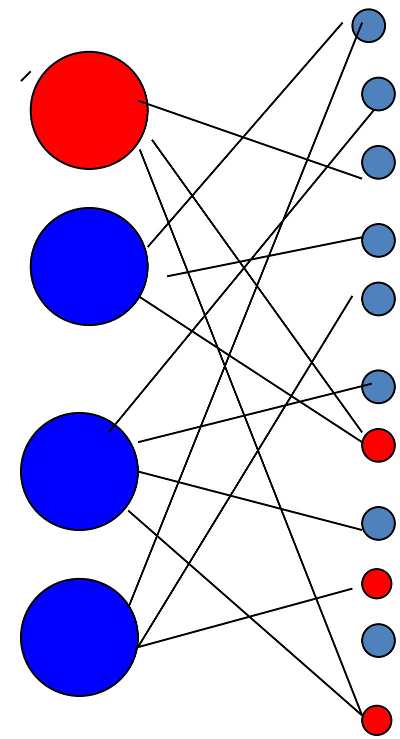


To go from short Common String to a committee for each node:

Committee is indexed by

(Common String, node ID)

Create Deterministic Sampler



Since almost all committees are good,

it suffices if a small constant fraction of bits in Common string are random

To go from Common String to a committee for each node:

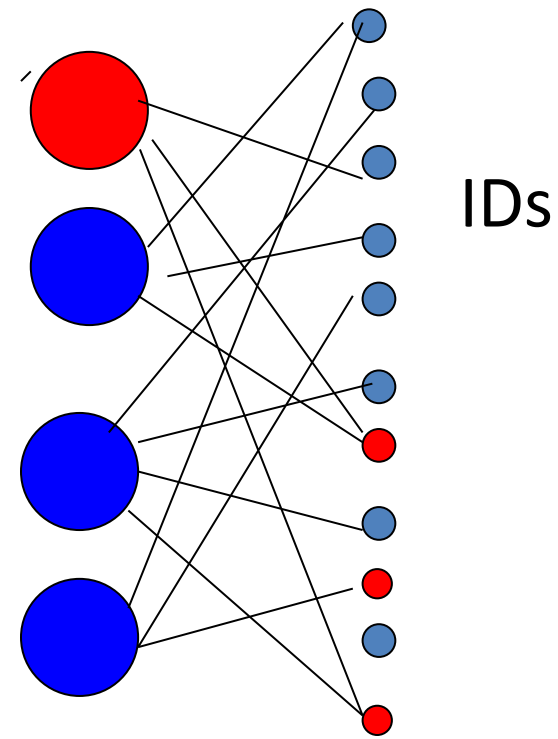
Committee is indexed by

(Common String, node ID)

It works even if:

- adversary sets its bits after seeing good bits,
- adversary controls more than half the bits,
- there are bits hidden by delays from asynchrony

Create Deterministic Sampler



To go from Common String to a committee for each node:

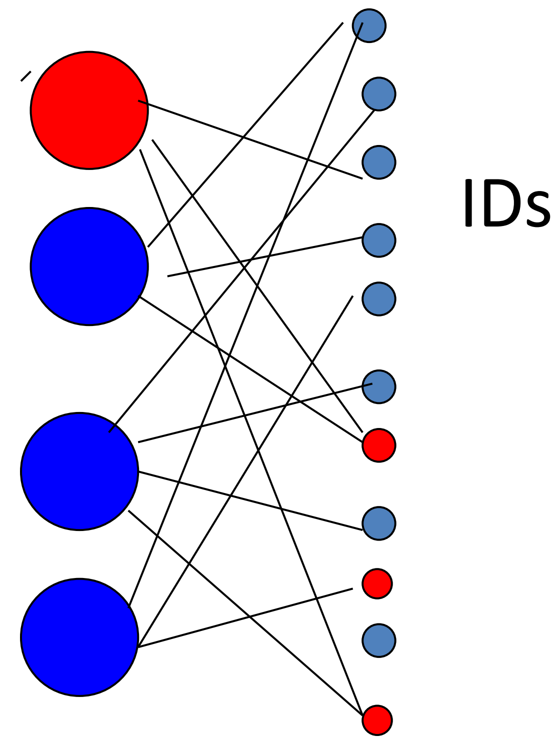
Committee is indexed by

(Common String, node ID)

It works even if:

- adversary sets its bits after seeing good bits,
- adversary controls more than half the bits,
- there are bits hidden by delays from asynchrony
- Even if the ID space is unknown and poly(n)

Create Deterministic Sampler



To go from Common String to a committee for each node:

Committee is indexed by

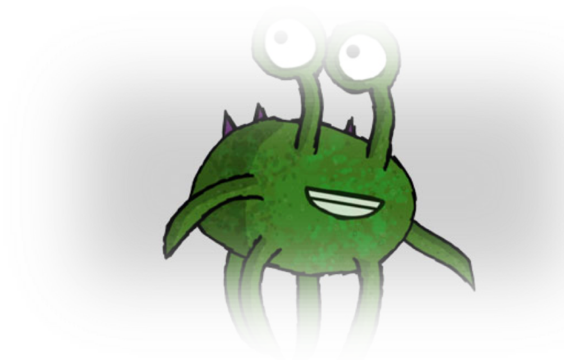
(Common String, node ID)

It works even if:

- adversary sets its bits after seeing good bits,
- adversary controls more than half the bits,
- there are bits hidden by delays from asynchrony
- Even if the ID space is unknown and poly(n)(?)

Create Deterministic Sampler

Is this function polytime constructable?



One small representative committee can:

- Run BA in less time and communication and then tell other nodes the result.

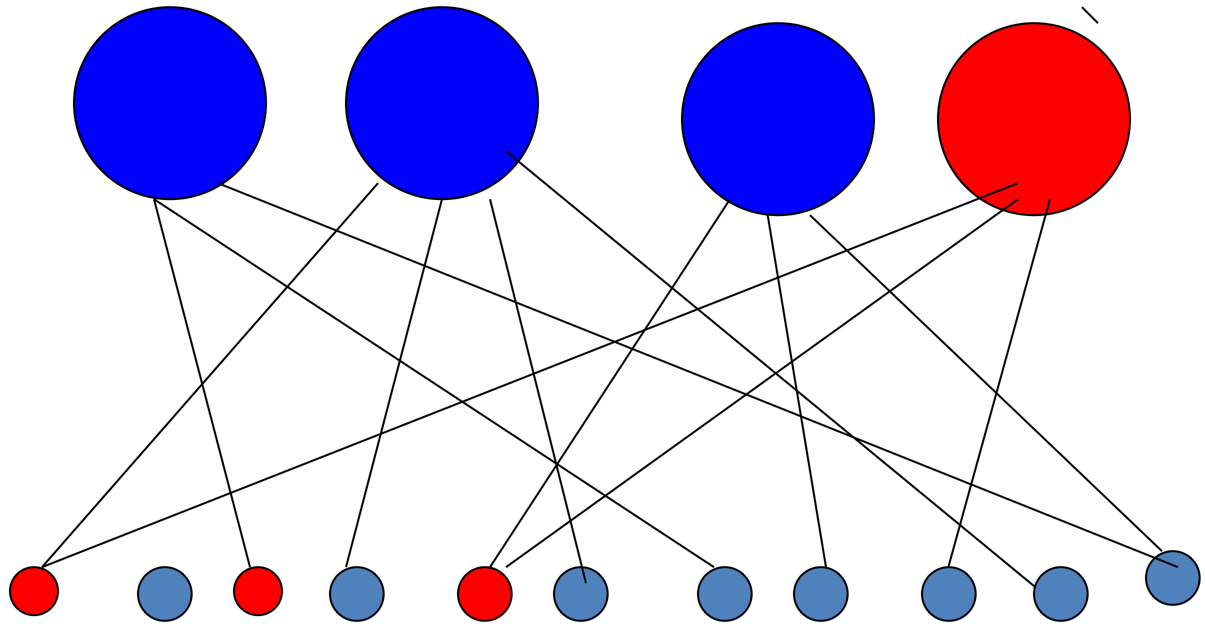
- Produce a O(log n) bit common string

  of fair coins interspersed with ~t/n fraction of adversary set bits

"Bit fixing random source"

A set of mostly representative committees can be built deterministically and efficiently

1-1/log n fraction of committees have close to representative membership, for ANY subset of BAD nodes

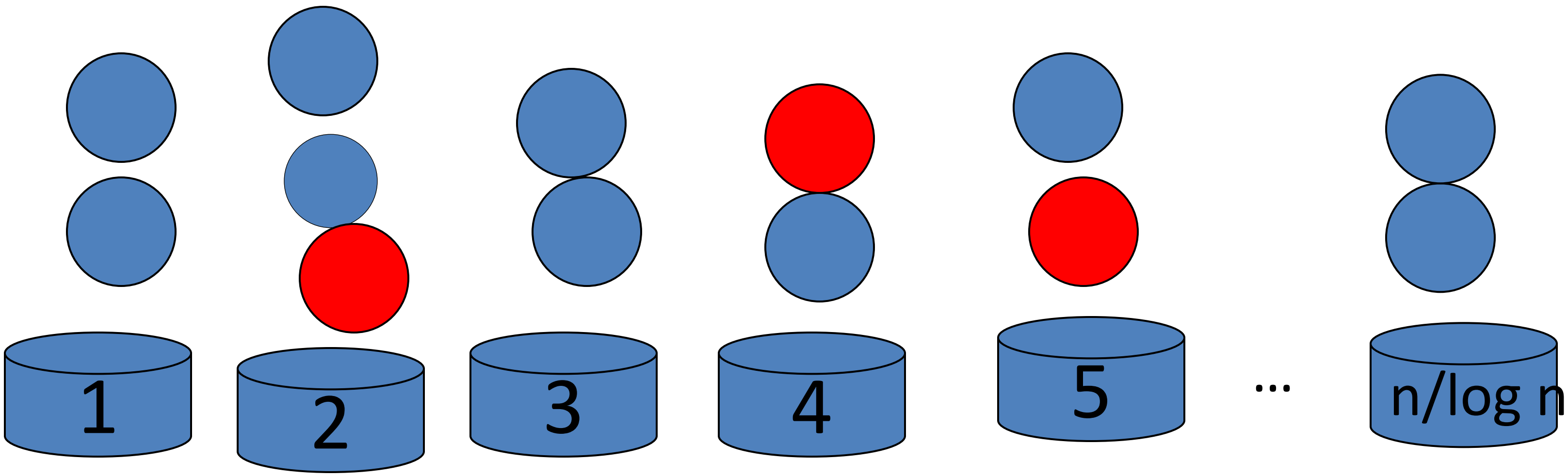


But requires an agreed upon mapping of nodes to the graph nodes !!

To elect a single small committee, adapt Feige's O(log*n) (broadcast) method for leader election

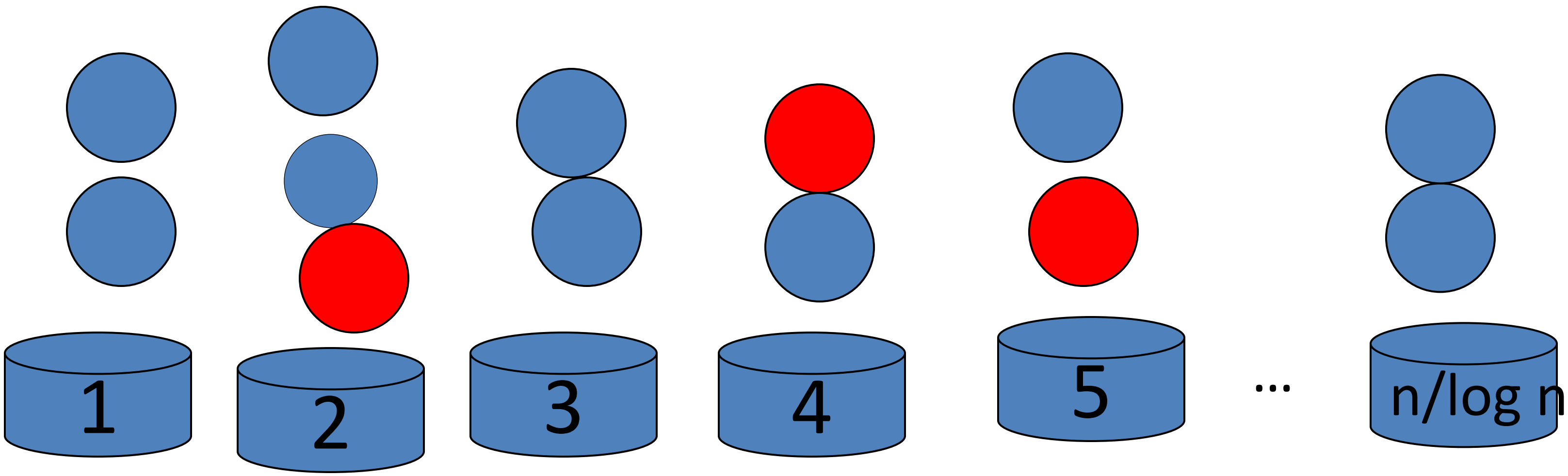
Each candidate randomly picks a bin;

remaining candidates =lightest bin's contents



To elect a single small committee, adapt Feige’s O(log*n) (broadcast) method for leader election

Each candidate randomly picks a bin;
remaining candidates =lightest bin’s contents

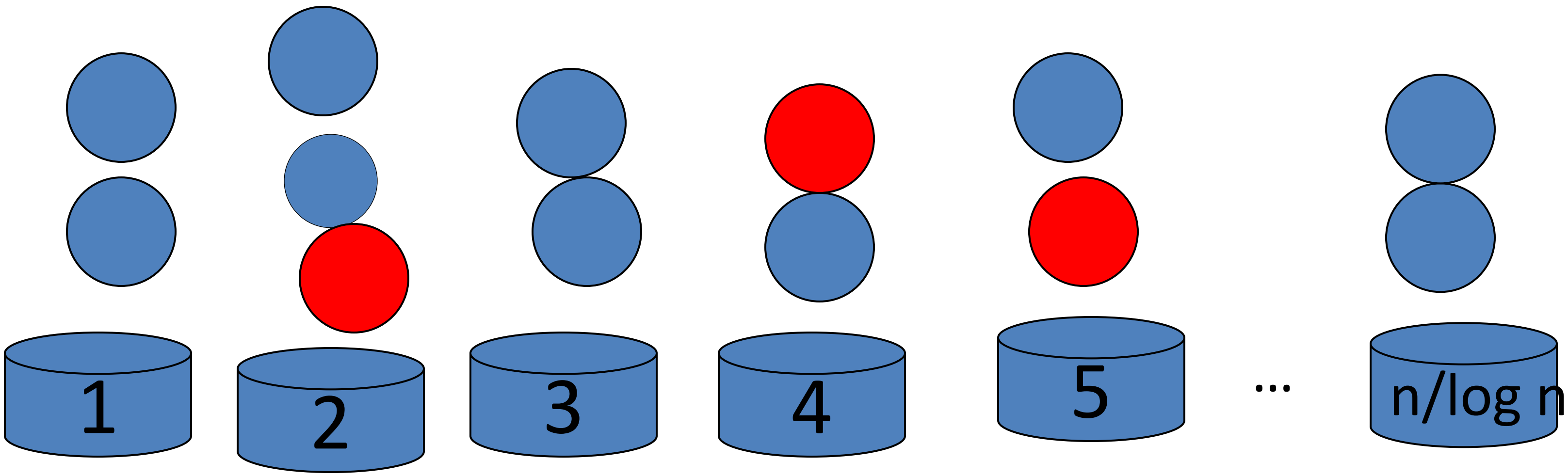


Even if bad ones see the choices first, lightest bin will be representative

In one round: #candidates → O(log n) whp

To elect a single small committee, adapt Feige's O(log*n) (broadcast) method for leader election

Each candidate randomly picks a bin;
remaining candidates =lightest bin's contents



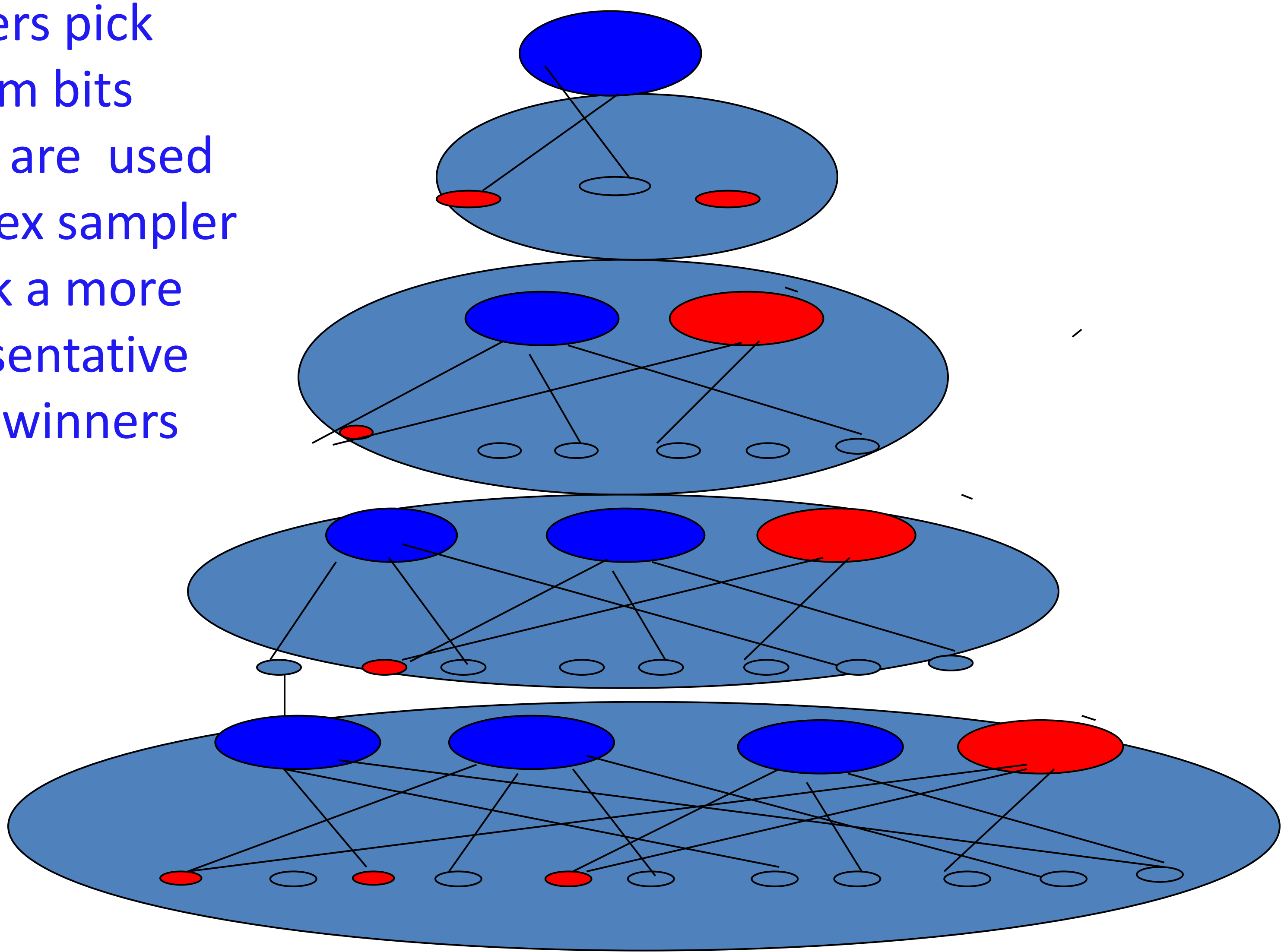
Even if bad ones see the choices first, lightest bin will be representative

In one round: #candidates → O(log n) whp

- Can be made to work even with asynchrony with polylog messages in $O(\log^c n)$ time

Use sampler to map winners to new committees

Winners pick random bits which are used to index sampler to pick a more representative set of winners



Static vs Adaptive adversary

- Note: A technique which elects a small committee is subject to the adaptive adversary which takes over the committee before it acts.

Do we care about this??

Byzantine agreement with an adaptive adversary and asynchrony

BA with asynchrony and adaptive adversary

- Ben-Or, $t<n/5$ 1983 expected exponential time
- Bracha $t<n/3$ 1984 expected exponential time
- K, Saia $t<cn$ 2013-6, expected $O(n^{2.5})$,$O(n^3)$ time, c very small constant

BA with asynchrony and adaptive adversary

- Ben-Or, $t<n/5$ 1983 expected exponential time
- Bracha $t<n/3$ 1984 expected exponential time
- K, Saia $t<cn$ 2013-6, expected $O(n^{2.5})$,$O(n^3)$ time, c very small constant

Not practical!

BA with asynchrony and adaptive adversary

- Ben-Or, t<n/5 1983 expected exponential time
- Bracha t<n/3 1984 expected exponential time
- K, Saia t <cn 2013-6, expected $O(n^{2.5})$,$O(n^3)$ time, c very small constant

Not practical!

Not yet

Review: Ben-Or's BA Alg 1983, $t<n/5$

While not decided each p repeats:

do Broadcast of vote b_p

$v \leftarrow$ majority value

tally $\leftarrow$ size of majority

CASE: tally

A) $> (n+t)/2$ then Decides on v

B) $> t$ then $b_p \leftarrow v$

C) else $b_p \leftarrow$ personal coinflip

We modify Ben-Or

While not decided each p repeats:

do <u>Broadcast</u> of vote b_p

$v \leftarrow$ majority value

tally $\leftarrow$ size of majority

CASE: tally

A) $> (n+t)/2$ then Decides on v

B) $> t$ then $b_p \leftarrow v$

C) else $b_p \leftarrow$ ~~personal coinflip~~

compute a **collective coin**

Decision results if **collective coin** agrees with v ("**good direction**")

Recall:

Ben-Or's iterations can be repeated while **collective coin** is not agreed on or not fair.

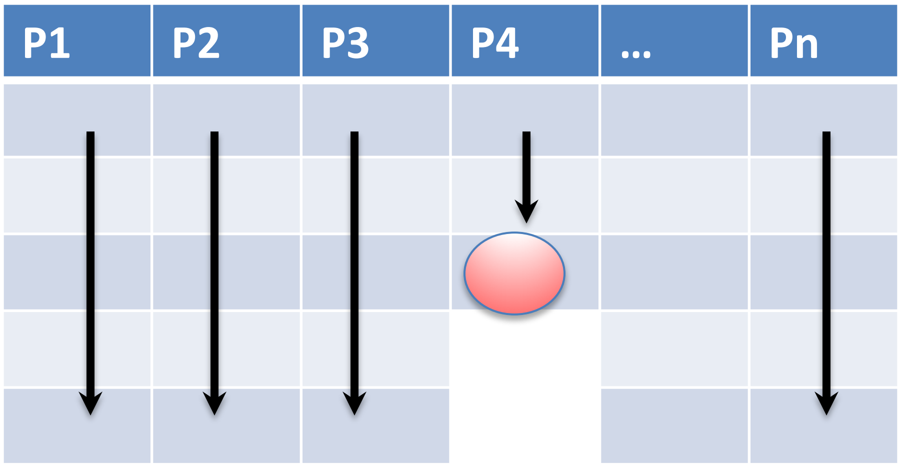
Ends when 4n/5 good processors hold the same value

collective coin flipping

- Idea: nodes communicate their coinflips and take a vote

Must be robust to up to t (good) coins missing in any step.

m-sync: adaption of multicast



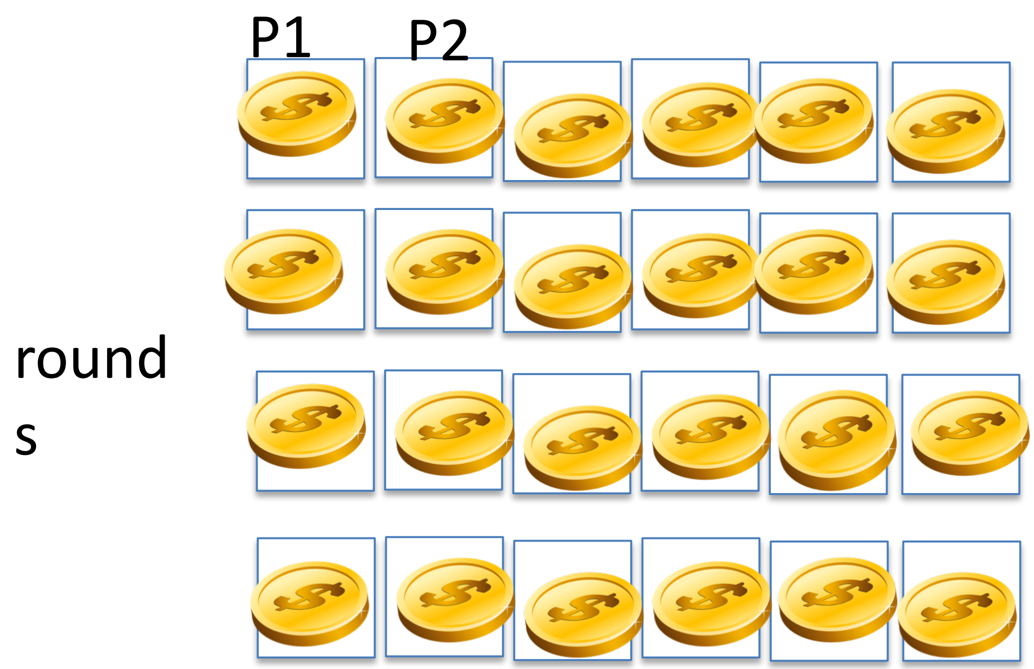
Each node "posts" messages to a column from top to bottom

All but t columns are full and agreed upon by all good nodes

For up to t columns, the adversary may stop the node early and the last value written may be ambiguous.

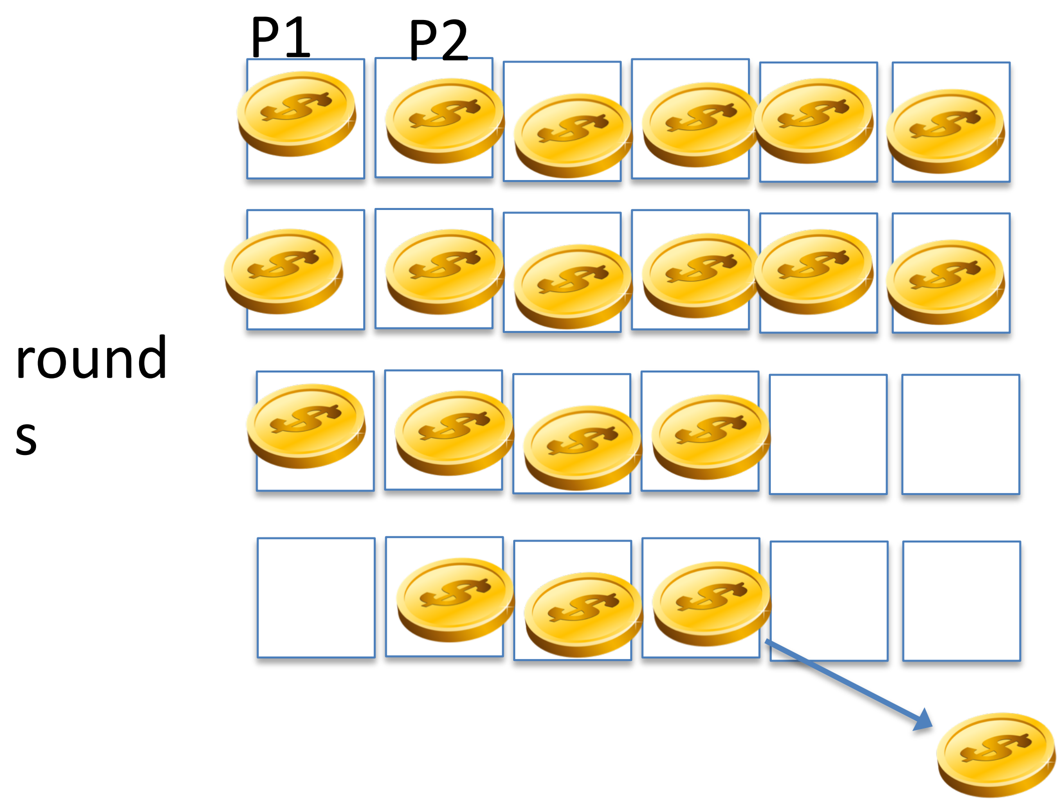
Use the m-sync: m rounds of coinflips generated by each node, m~=n to create "blackboard"

- *If all nm coins are flipped and fair*, then with constant prob they have deviation $\sigma > \sqrt{nm} > ct$ if m = n, c constant

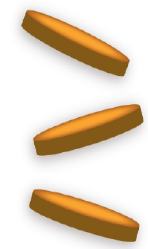


Adversary can

1. Stop t columns early
2. Hide the last coin tossed
   in each of up to t columns



1. Effect of stopping coins

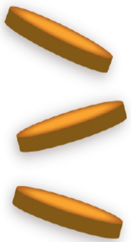


There are n(n-2t) fair coins plus
a number chosen by the adversary
between 0 and tn.

Suppose we let
the adversary sees all the n(n-2t) fair coins first

It will choose to stop the remaining coins so as to minimize the deviation of the sum

Random walk of n steps



Each step is +1, -1 with prob ½

Let n be the number of steps

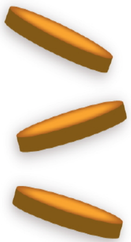
Let S(n) be the sum after n steps

Let M(n) be the minimum sum achieved during a walk

Lemma : Pr (M(n) ≥k) < 2 Pr(S(n) >k)

Adversary can do no better than to stop the stream of nt coins at the lowest point in the walk, i.e, M(nt)

With both effects

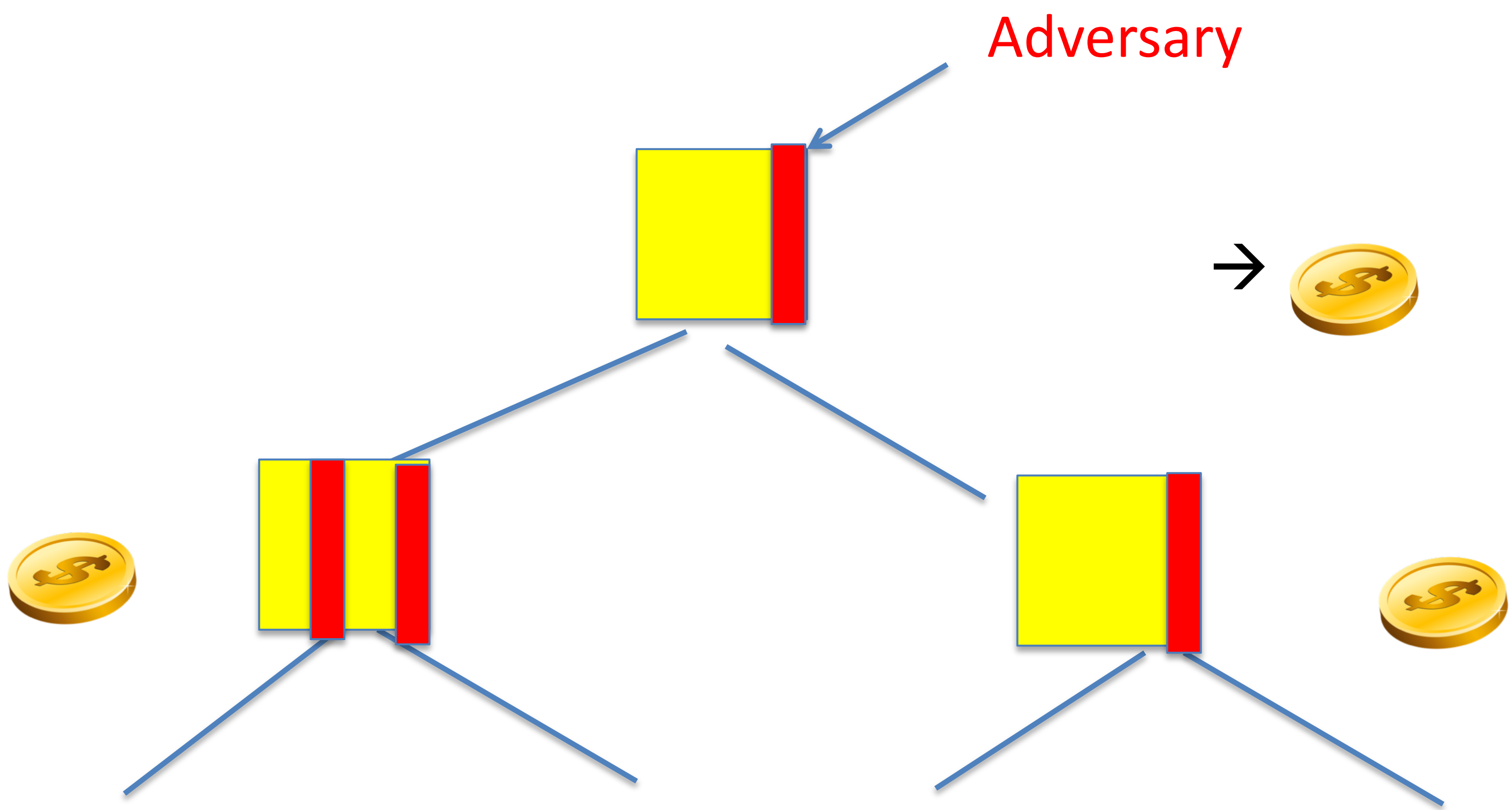


Pr(Fair coin is given by the sum of entries in blackboard)=

Pr(S(n(n-2t)) >M(tn) (for the stopping) + t (for the hidden coins)]

> Pr(S(n(n-2t)) >2S(tn) + t (for the hidden coins)]
=constant for sufficiently small t

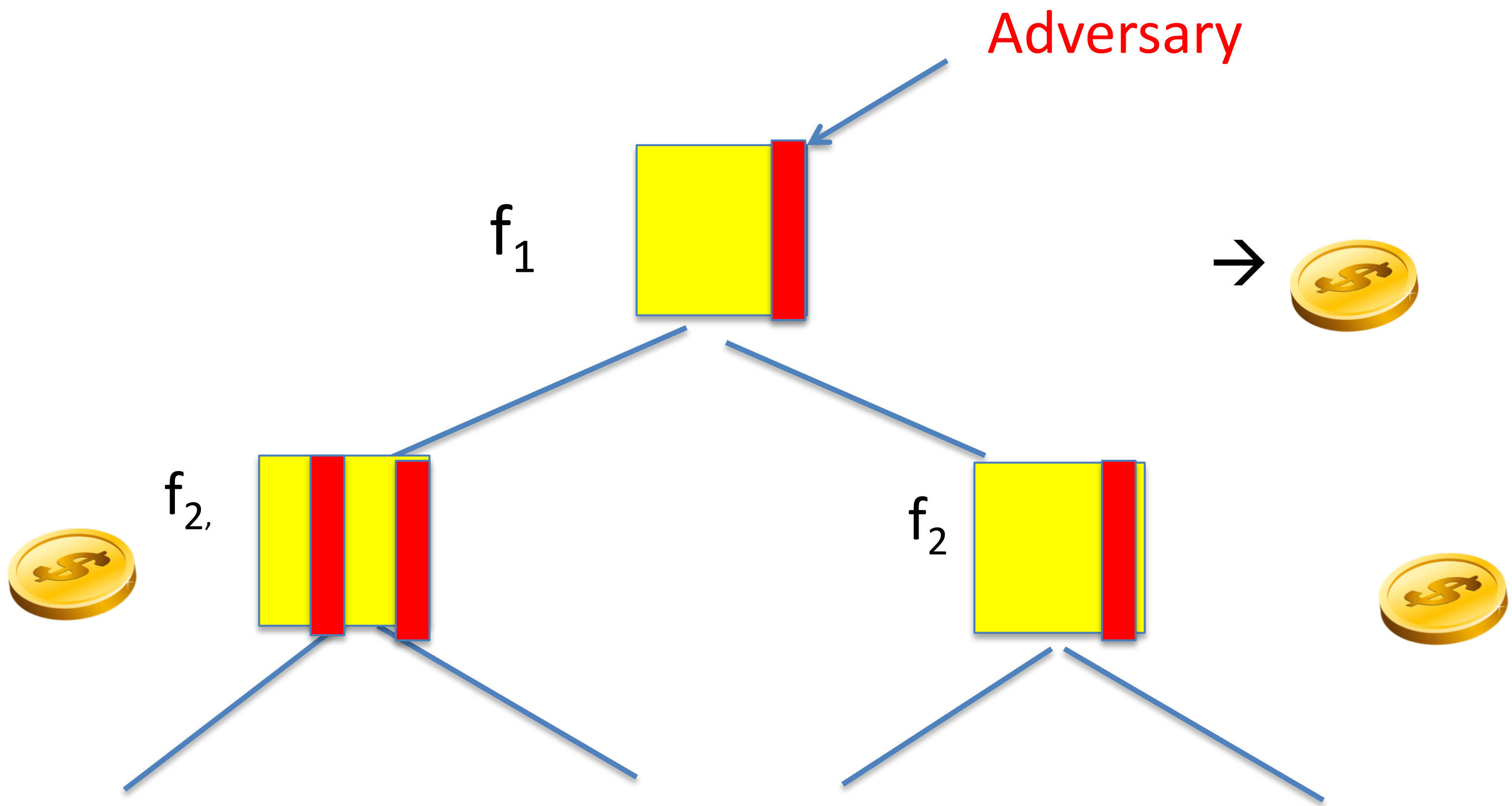
The adversary takes over nodes adaptively and set values in t columns



Basic step is n-sync

How many iterations are needed to generate a fair coin sometimes?

Goal is to design a function $F=f_{1}, f_{2}, \ldots$



Basic step is n-sync

How to design an F?

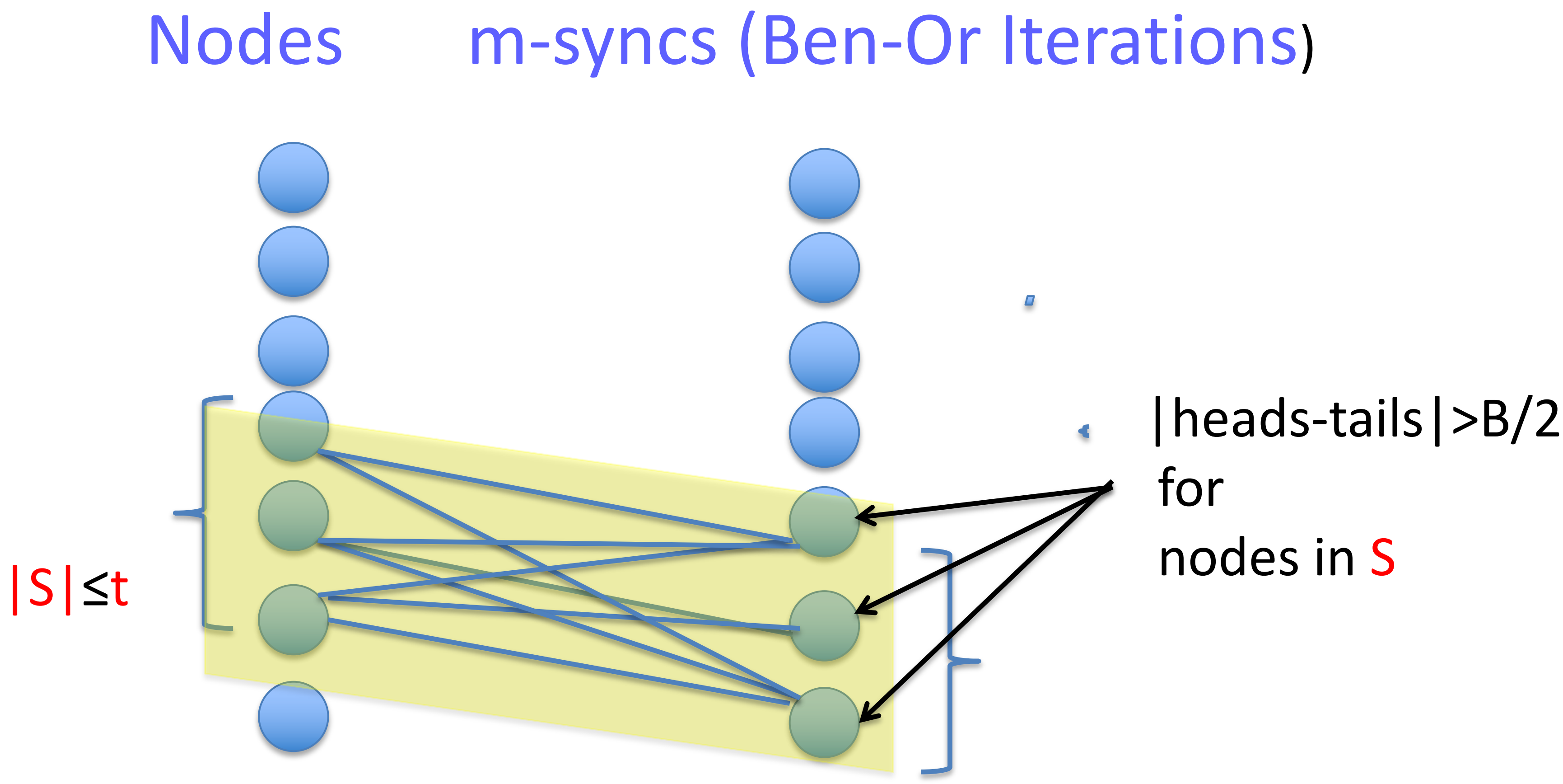
IDEA: If majority does not yield a fair coin sometimes then

Adversarially controlled columns show a suspect pattern of Biased coinflips over time, from the view of a constant fraction of nodes.

Each node individually detects unusual bias and individually eliminates suspicious nodes

Detection of suspicious nodes: finding "planted heavy-weighted clique"

Find a set of ≤t suspect nodes S



collective coin

Initially, V_p={1,2,…,n} set of columns

p outputs 1 if #heads-#tails from nodes in V_p >0 else 0

Every s iterations, determines S_p suspicious nodes

Sets $V_p \leftarrow V_p \setminus S_p$

Once all bad nodes are excluded by all good nodes, a O(1) expected iterations of Ben-Or suffice to produce a fair coin

Cow Tools

Constructing a polynomial time F

How to find suspicious columns

For each group of 2n iterations, construct matrix M_p

$M_p(i,j)$= #heads-#tails in m-sync i in column j

DEF: 2-norm of vector v is $|v|_2 = (\Sigma v_i^2)^{1/2}$

2-norm of matrix M is $|M|_2 = \max |Mu|_2$ for all u,

where $|u|_2=1$

Maintain badness score bad(j) for each column j, initially 0.

Each p removes suspicious nodes (after m iterations):

If $|M_p|_2 >$ Threshold

- $r \leftarrow$ top right singular vector of M_p,
- for all j, increase bad(j) by r_j^2
- if bad(j) ≥ 1 remove node j from V_p

To summarize:

Ben-Or's iterations are repeated until it stops

- m-sync allows all nodes to view nearly the same coinflips
- Each node p sets its coinflip in Ben-Or to the majority of the votes in the n-sync cast by nodes in unsuspected node set V_p (collective coin)
- If agreement doesn't occur, many nodes p detect bias and make progress towards removing bad nodes from V_p
- Eventually, the bad nodes are removed by enough nodes p and agreement occurs in constant expected time.

Larger lesson

Either nodes are cooperative and agreement happens. Or we can detect them.

Don’t need global hash functions, assumption of synchrony, solving puzzles(?). Gives an incentive to act according to protocol.

Larger lesson

Either nodes are cooperative and agreement happens. Or we can detect them.

Don't need global hash functions, assumption of synchrony, solving puzzles(?). Gives an incentive to act according to protocol.

What about changing nodes and Sybil attacks?

Larger lesson

Either nodes are cooperative and agreement happens. Or we can detect them.

Don't need global hash functions, assumption of synchrony, solving puzzles(?). Incentive to act according to protocol or be excluded.

What about changing nodes and Sybil attacks?

Identities can be interchangeable but the set of identities controlled by bad nodes must be stable enough to accumulate badness

Larger lesson

Either nodes are cooperative and agreement happens. Or we can detect them.

Don't need global hash functions, assumption of synchrony, solving puzzles(?). Incentive to act according to protocol or be excluded.

What about changing nodes and Sybil attacks?

Identities can be interchangeable but the set of identities controlled by good nodes must be stable enough to accumulate goodness??

References

- Samplers, construction, randomness extraction (David Zuckerman). Applications to reducing messages (K, Saia, esp ICDCN 2011, Braud-Santoni PODC 2013)
- On reducing message complexity with the use of public key crypto and/or random oracles (See Abraham, et al 2018 arxiv, Katz, Koo STOC 2006)
- $o(n^2)$ messages with adapative adversary, if private channels, no other crypto assumptions (K, Saia JACM 2011)
- Use of representative sets, e.g., for blockchain (NUS paper on ELASTICO, CCS 2016, Luu et al.), for DHT (Awerbuch and Scheidler)
- Byzantine agreement with adaptive adversary (K, Saia JACM 2016+ correction for stopping effect Dec 2018 arxiv)
- Using Feige's to do leader election with asynchrony in the static model (Kapron,etal. SODA 2008)

Thank you
(and thanks to Gary Larsen)

Questions?

Cow philosophy