Conflict on Large Networks

Jared Saia

Collaborators: Bruce Kapron, David Kempe, Valerie King, Amitabh Trehan, Vishal Sanwalani and Maxwell Young

Research funding:

NSF

SCIENCE TO SHAPE THE FUTURE OF THE ARMY
ARMY RESEARCH OFFICE
1951

Sandia
National
Laboratories

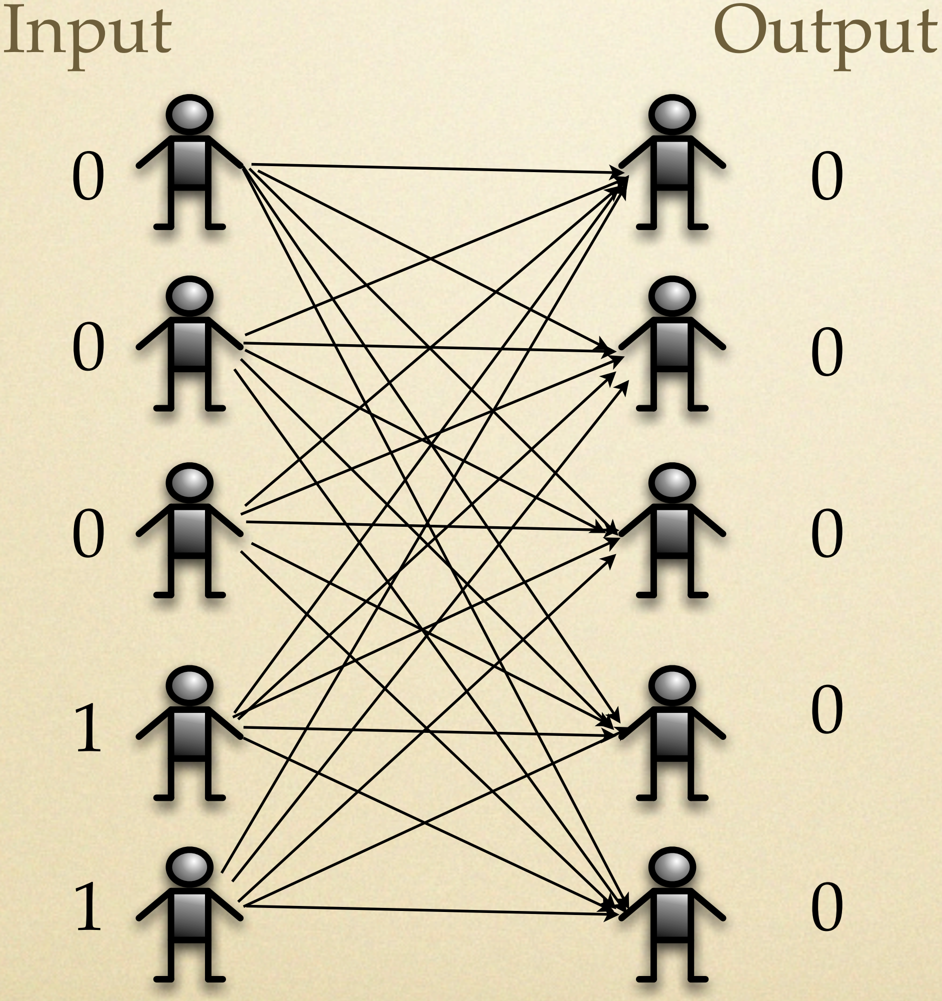
IARPA

Components Fail, Group Functions

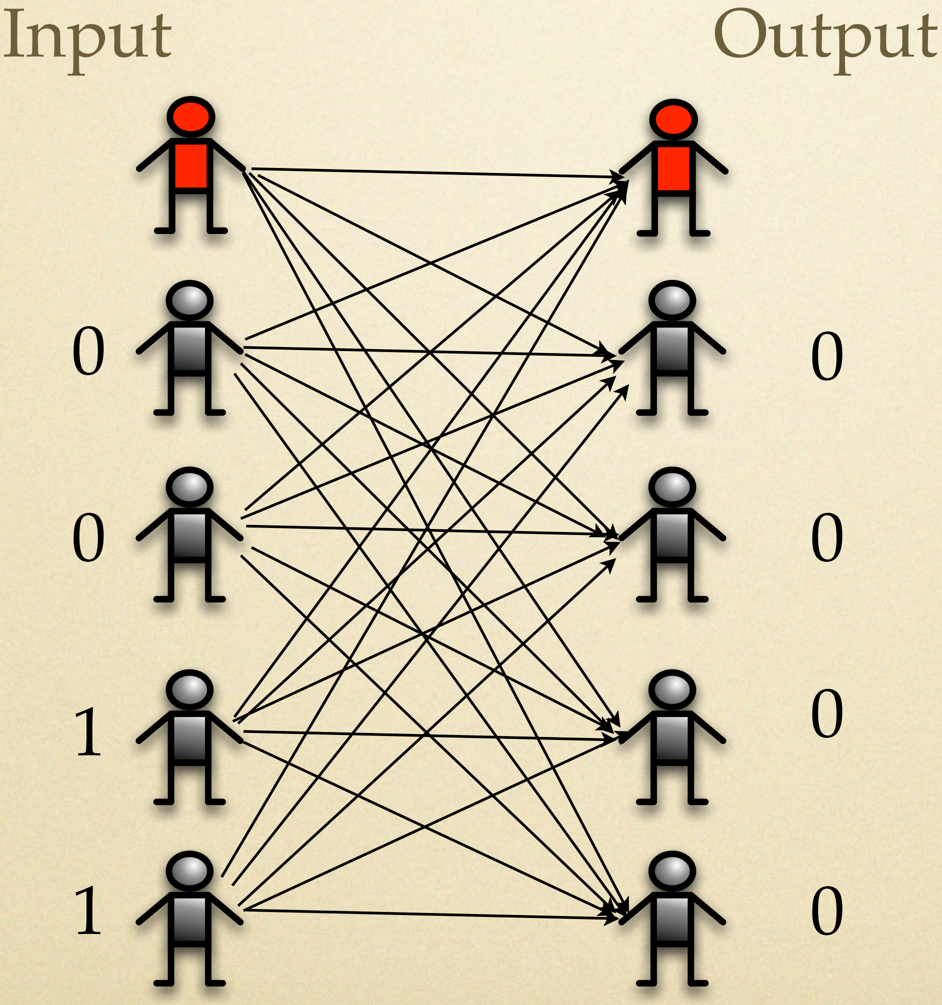
Group Decisions

- Periodically, components unite in a decision
- Idea: components vote. Problem: Who counts the votes?

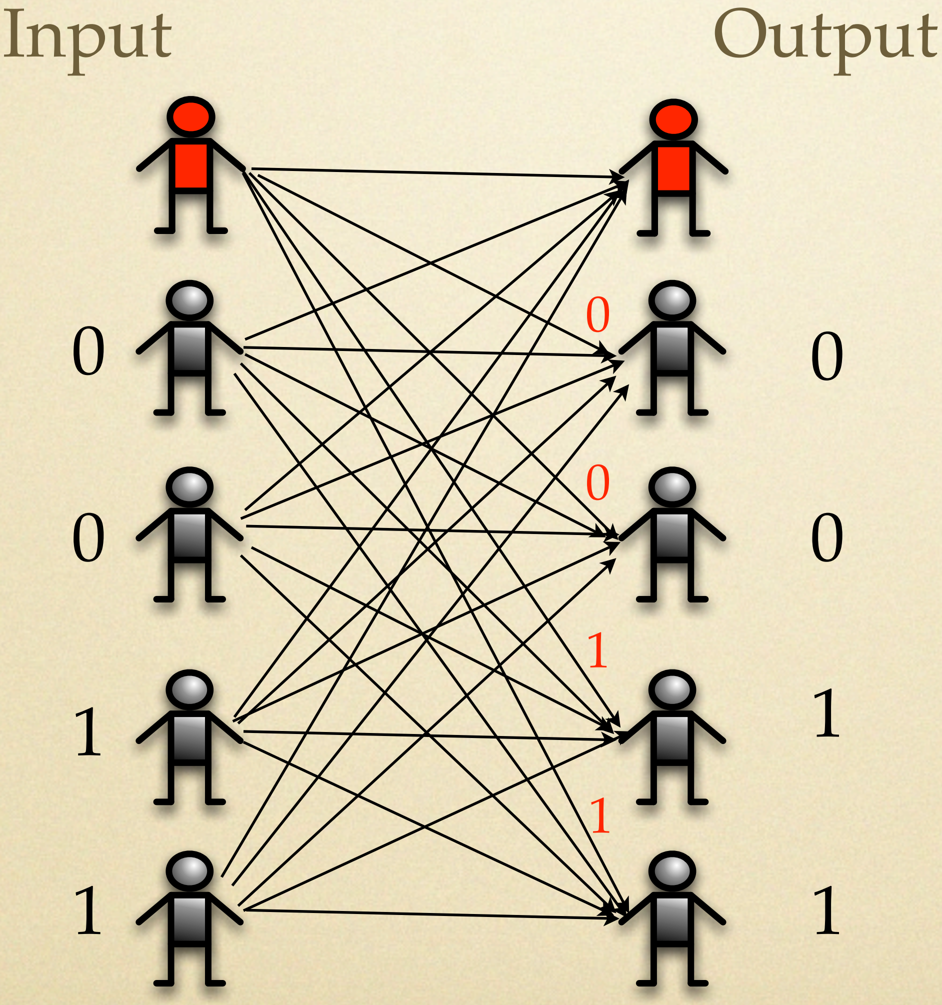
Idea: Majority Filtering



Idea: Majority Filtering



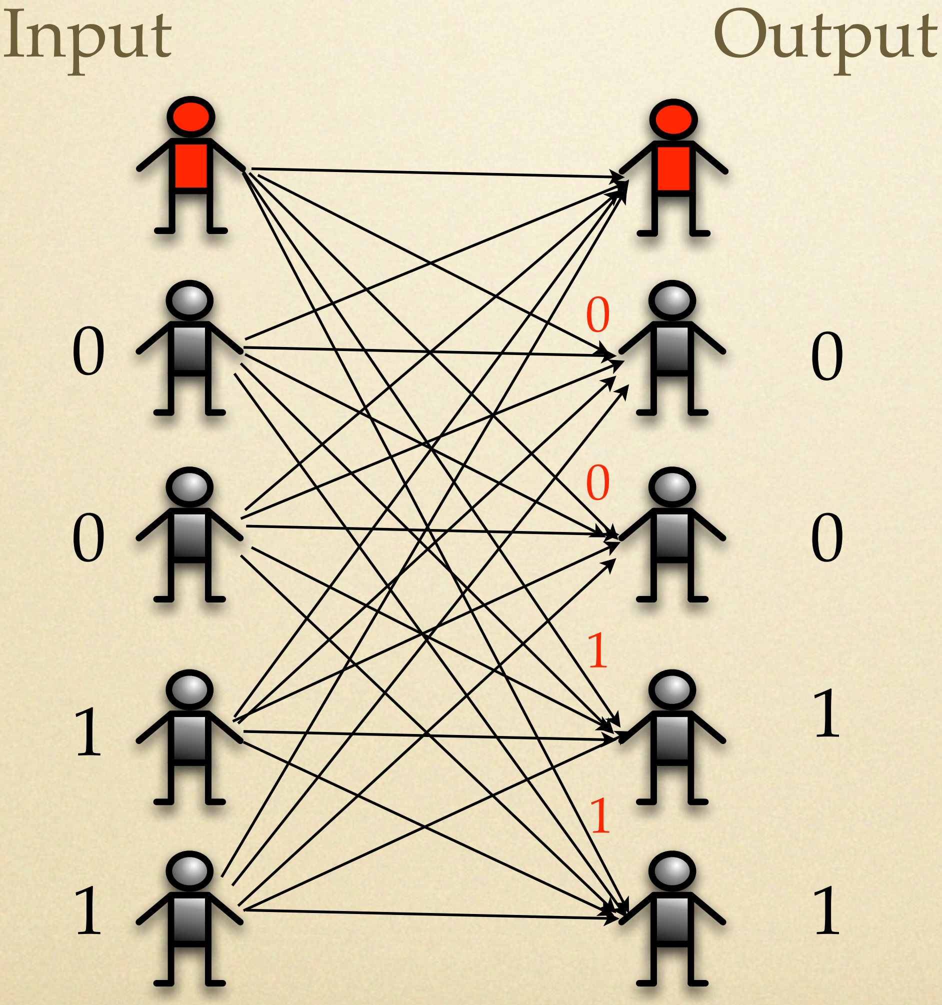
Problem



Byzantine Agreement

- Each processor starts with a bit
- Goal: 1) all good procs output the same bit; and 2) this bit equals an input bit of a good proc
- t = # bad procs controlled by an adversary

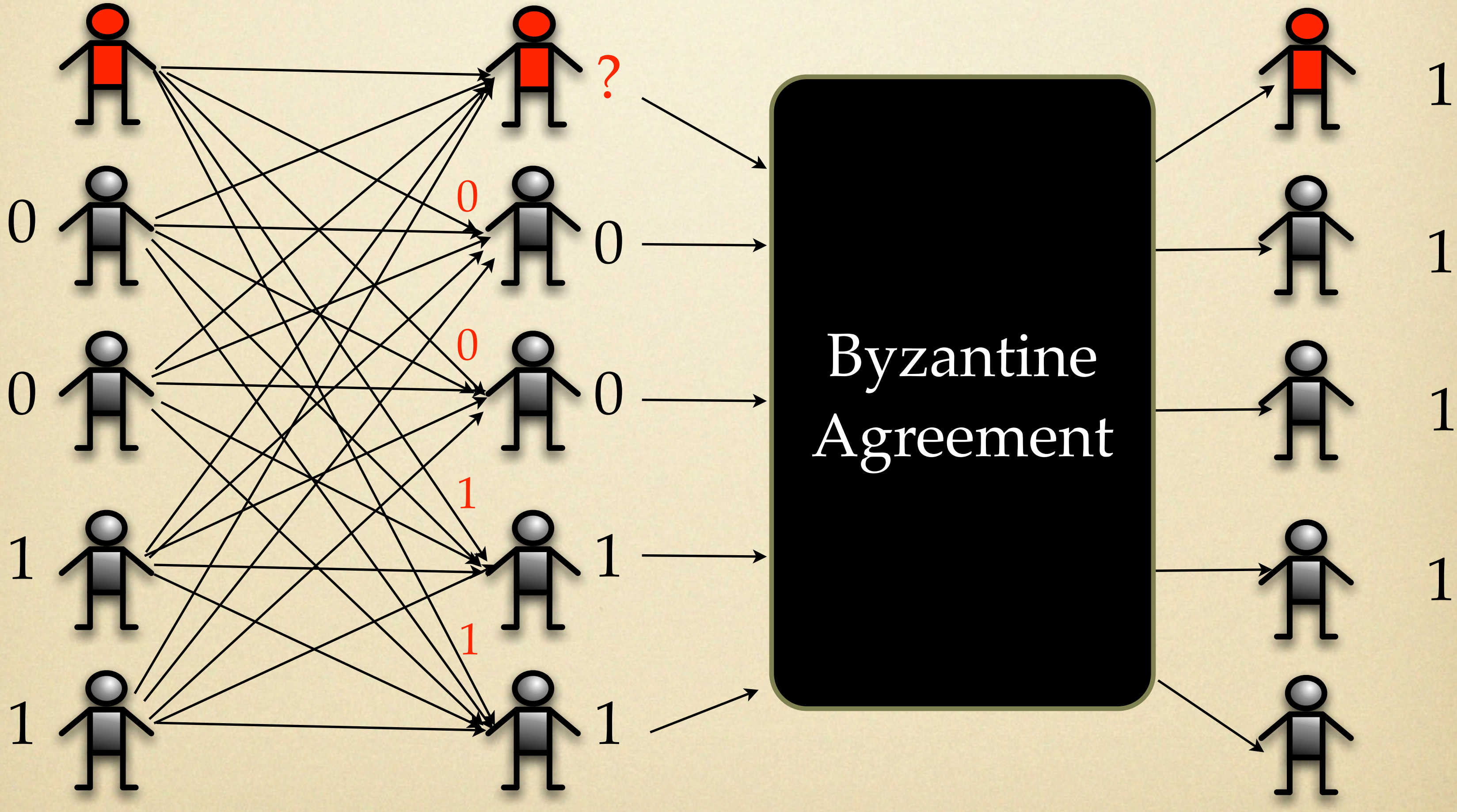
Problem



Idea

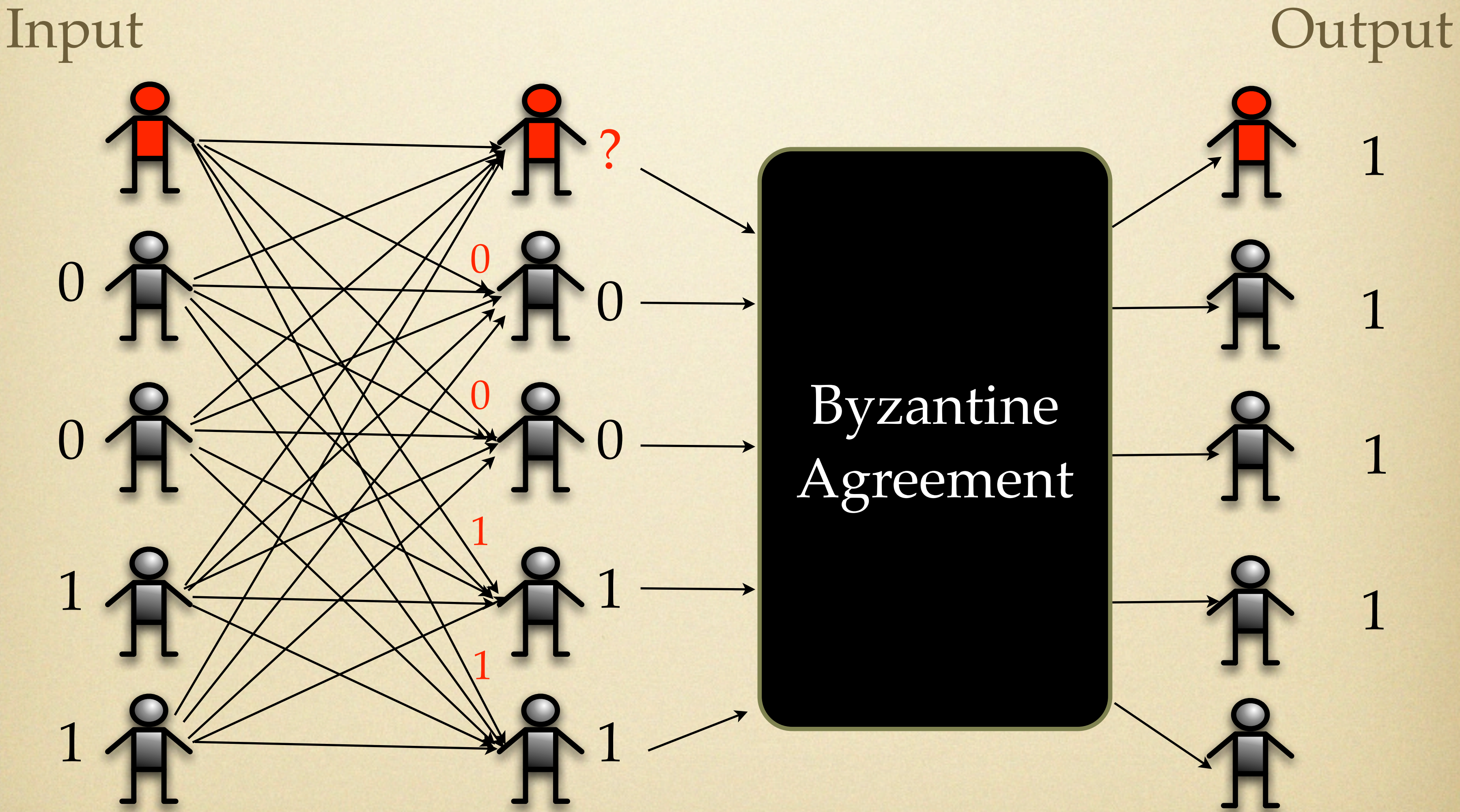
Input

Output

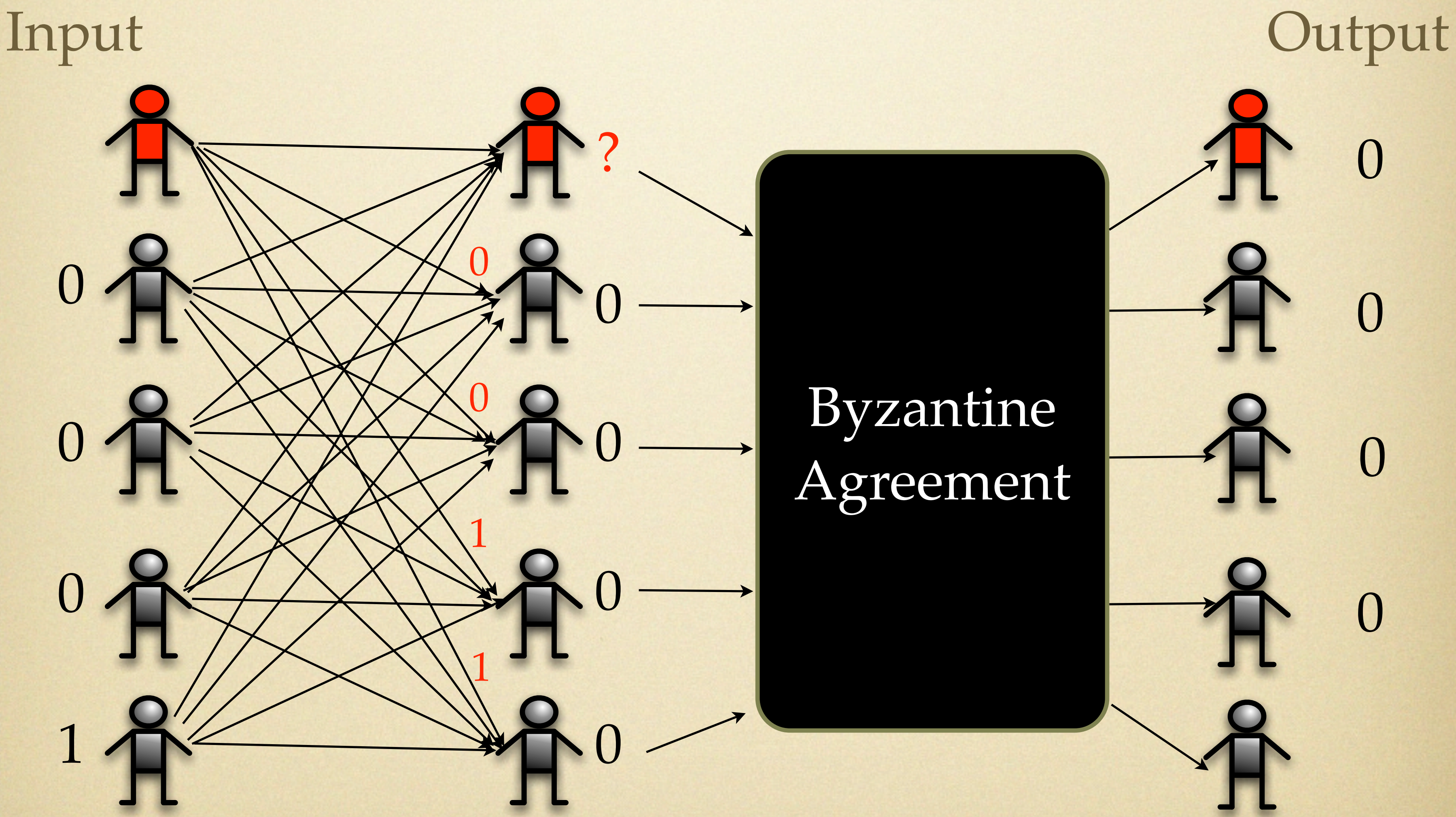


Byzantine Agreement

All good procs always output same bit



If majority bit held by >= 3 good procs, then all procs will output majority bit



Impossibility Result

- 1982: FLP show that 1 fault makes deterministic BA impossible in asynch model
- 2007: Nancy Lynch wins Knuth Prize for this result, called “fundamental in all of Computer Science”

Applications

- Peer-to-peer networks

  *“These replicas cooperate with one another in a **Byzantine agreement** protocol to choose the final commit order for updates.” [KBCCEGGRWWWZ ’00]*

- Rule Enforcement

  *“... requiring the manager set to perform a **Byzantine agreement protocol”** [NWD ’03]*

- Game Theory (Mediators)

  *“deep connections between implementing mediators and various agreement problems, such as **Byzantine agreement**” [ADH ’08]*

Applications

- Peer-to-peer networks

  "These replicas cooperate with one another in a ***Byzantine agreement*** *protocol to choose the final commit order for updates." [KBCCEGGRWWWZ '00]*

- Rule Enforcement

  "... requiring the manager set to perform a ***Byzantine agreement protocol"*** *[NWD '03]*

- Game Theory (Mediators)

  "deep connections between implementing mediators and various agreement problems, such as ***Byzantine agreement****" [ADH '08]*

- Also: Databases, Sensor Networks, Cloud Computing, Control systems, etc.

Scalability

- *"Unfortunately, Byzantine agreement requires a **number of messages quadratic** in the number of participants, so it is infeasible for use in synchronizing a large number of replicas"* [REGZK '03]

- *"Eventually batching cannot compensate for the **quadratic number of messages** [of Practical Byzantine Fault Tolerance (PBFT)]"* [CMLRS '05]

- *"The **communication overhead** of Byzantine Agreement is **inherently large**"* [CWL '09]

Impossibility

- Any BA (randomized) protocol which **always** uses less than n^2 messages will fail with non-zero probability. Implication of [DR ’85]

Impossibility

- Any BA (randomized) protocol which **always** uses less than n^2 messages will fail with non-zero probability. Implication of [DR '85]

- To do better than n^2 messages, we will need to fail with non-zero probability

Our Model

- Private channels
- Synchronous w/ rushing adversary
- Unlimited messages for bad procs
- Adaptive adversary

Our Model

- Private channels
- Synchronous w/ rushing adversary
- Unlimited messages for bad procs
- Adaptive adversary

  Adv. takes over procs at any time, up to t total

Our results

Theorem 1 (BA): For any constants c, ε, there is a constant d and a protocol which solves BA, for t <= (1/3- ε)n, with prob. 1-1/n^c , using

$$O(\sqrt{n}\log^3 n) \text{ bits per processor and } O(\log^d n) \text{ rounds}$$

Also

Theorem 2: (a.e.BA) For any constants. c, ε, there is a constant d and a protocol which for t<=(1/3- ε) brings

1-O(1/log n) fraction of good procs to agreement with prob. 1-1/n^c using

Polylogarithmic bits per processor and $O(\log^d n)$ rounds

Previous work

- Constant rounds in expectation is possible [FM '88]
- However, all previously known protocols use all-to-all communication

KEY IDEA: S

- $S = s_1\ s_2\ \dots\ s_k$ is a stream of mostly random numbers.
- Some a.e. globally known random numbers, some numbers fixed by an adversary which can see the preceding stream when choosing.

Algorithm Outline

I: Using S to get a.e. BA

II: Using S to go from a.e. BA to BA

III: Implementing S

BA with Global Coin, GC

Rabin's Algorithm

Send your vote to everyone

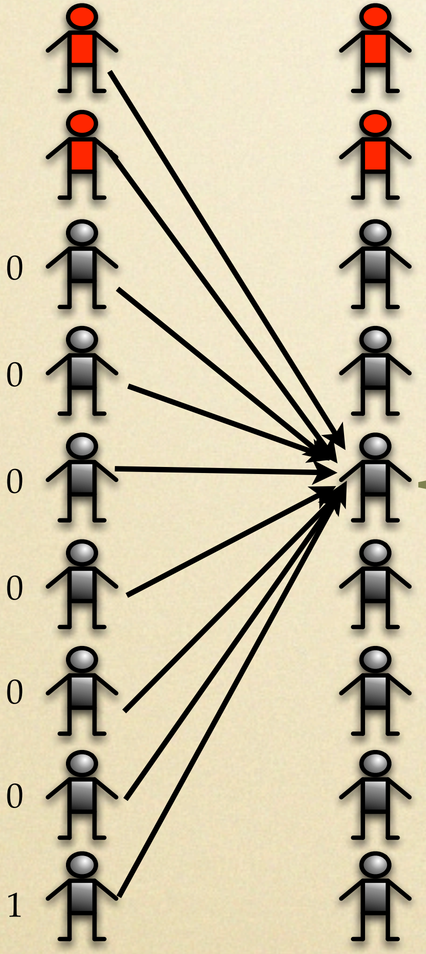
Let *fraction* be fraction of votes for majority bit

If *fraction* >= 2/3, set vote to majority bit; else set vote to GC

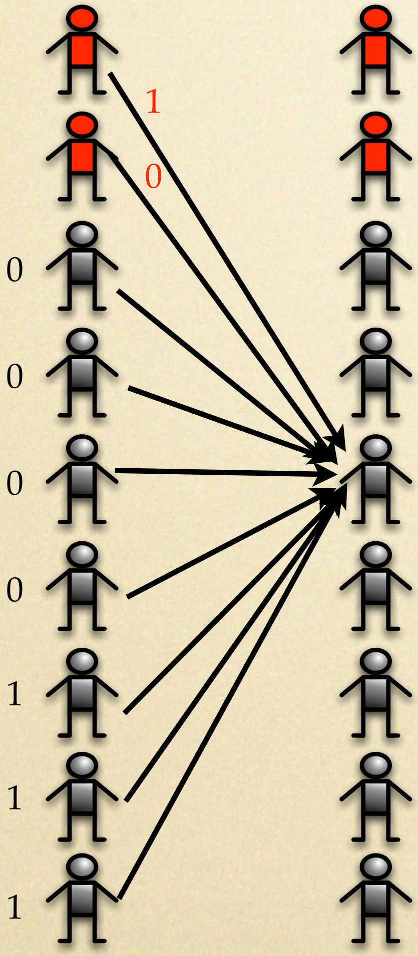
BA with Global Coin, GC

Rabin's Algorithm

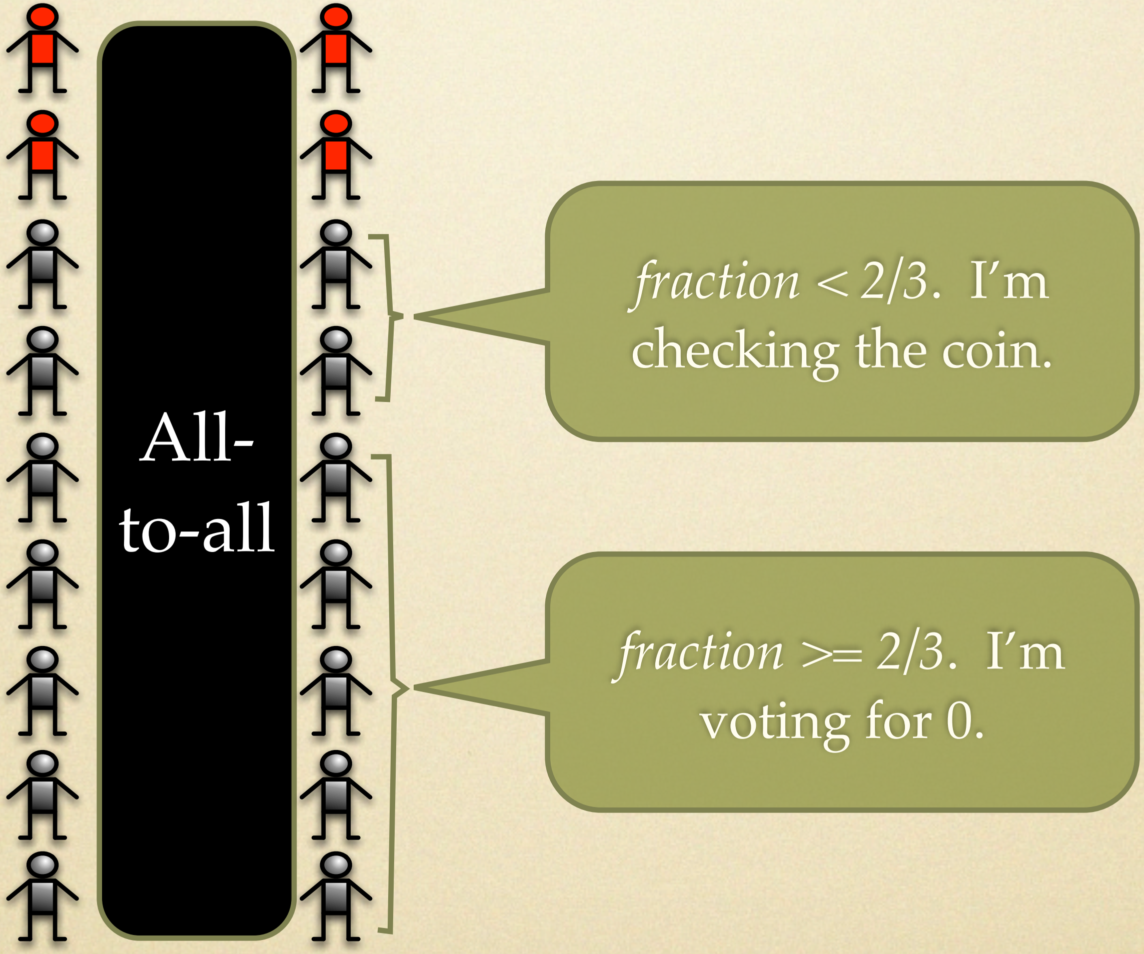
```
Set your vote to input bit
Repeat clogn times:
    Send your vote to everyone
    Let fraction be fraction of votes for majority bit
    If fraction >= 2/3, set vote to majority bit;  else set
        vote to GC
Output your vote
```



fraction >= 2/3. I'm voting for 0.



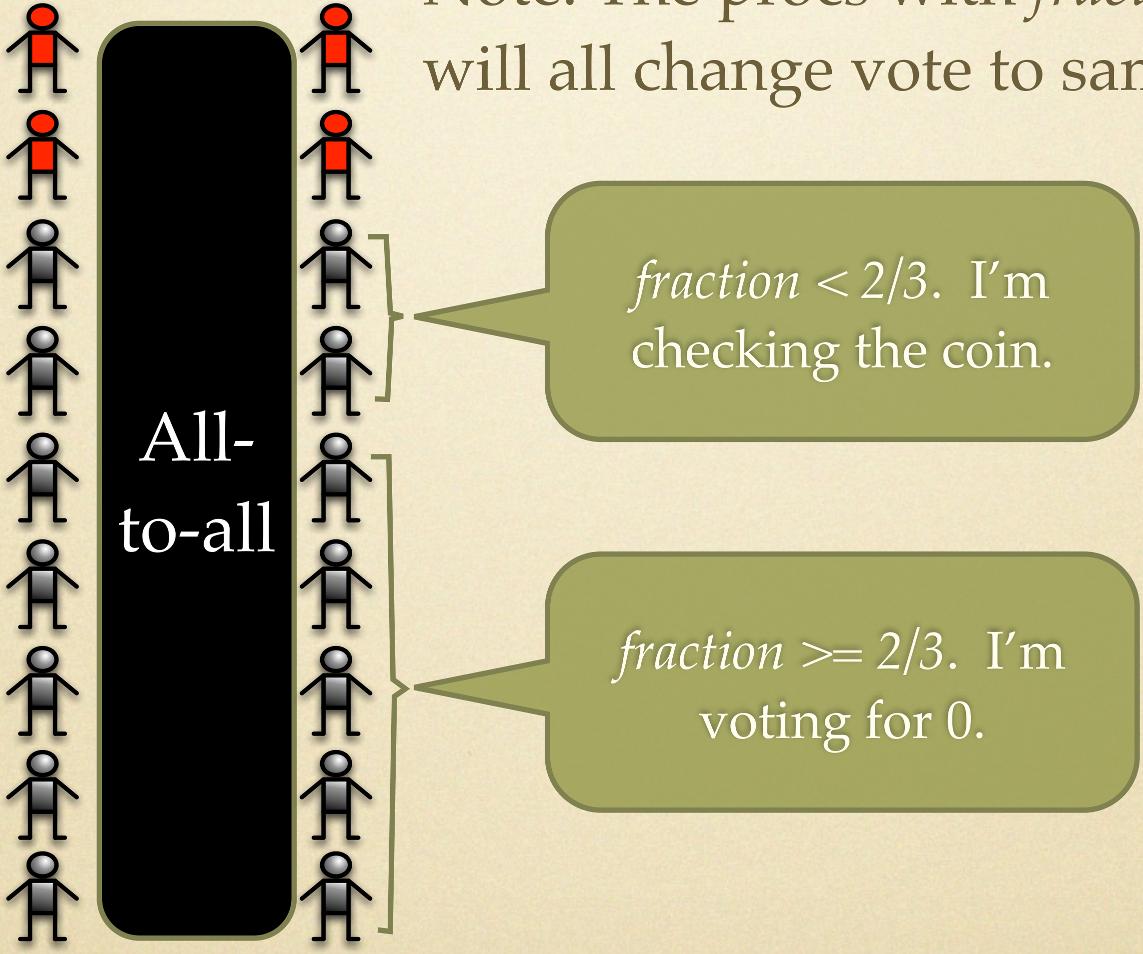
fraction < 2/3. I'm checking the coin.



fraction < 2/3. I'm checking the coin.

fraction >= 2/3. I'm voting for 0.

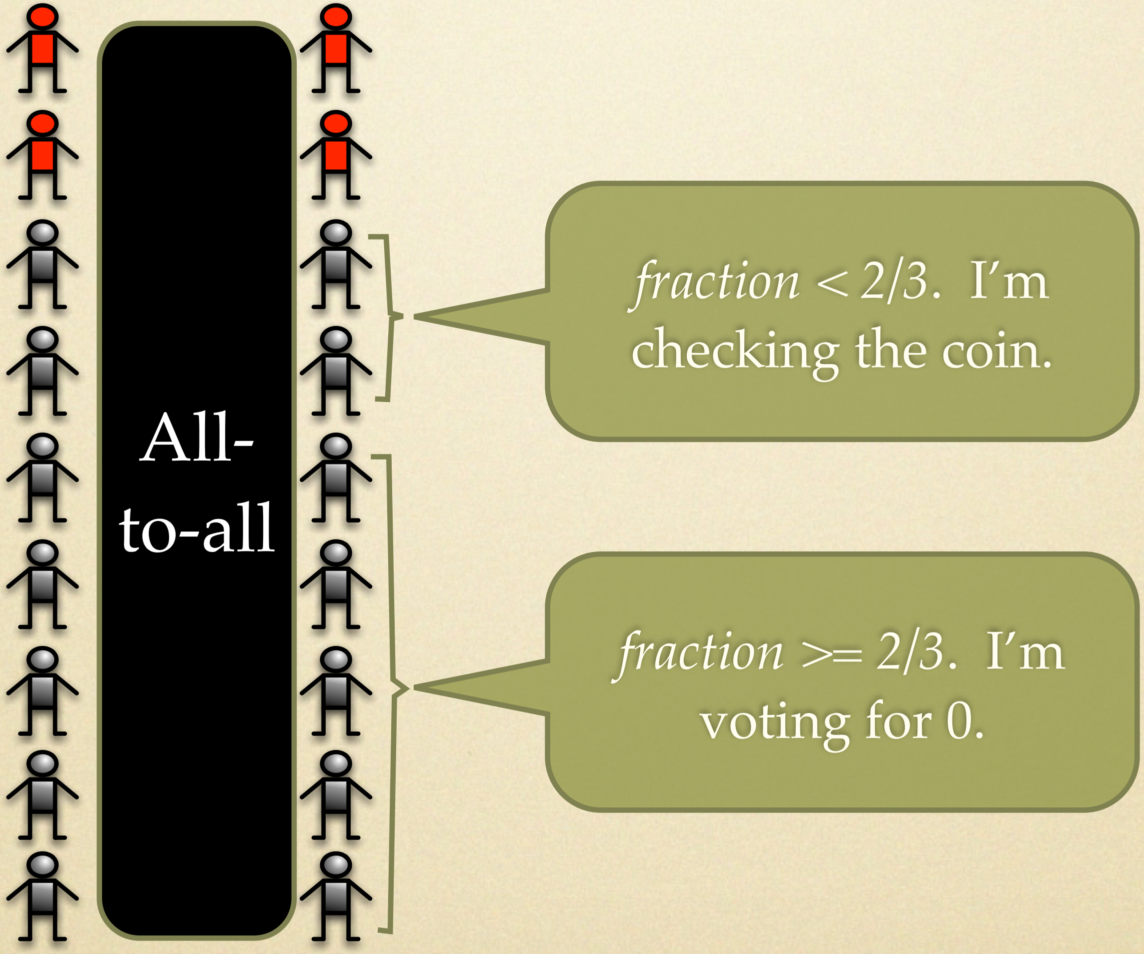
Note: The procs with *fraction* >= 2/3 will all change vote to same value



All-to-all

fraction < 2/3. I’m checking the coin.

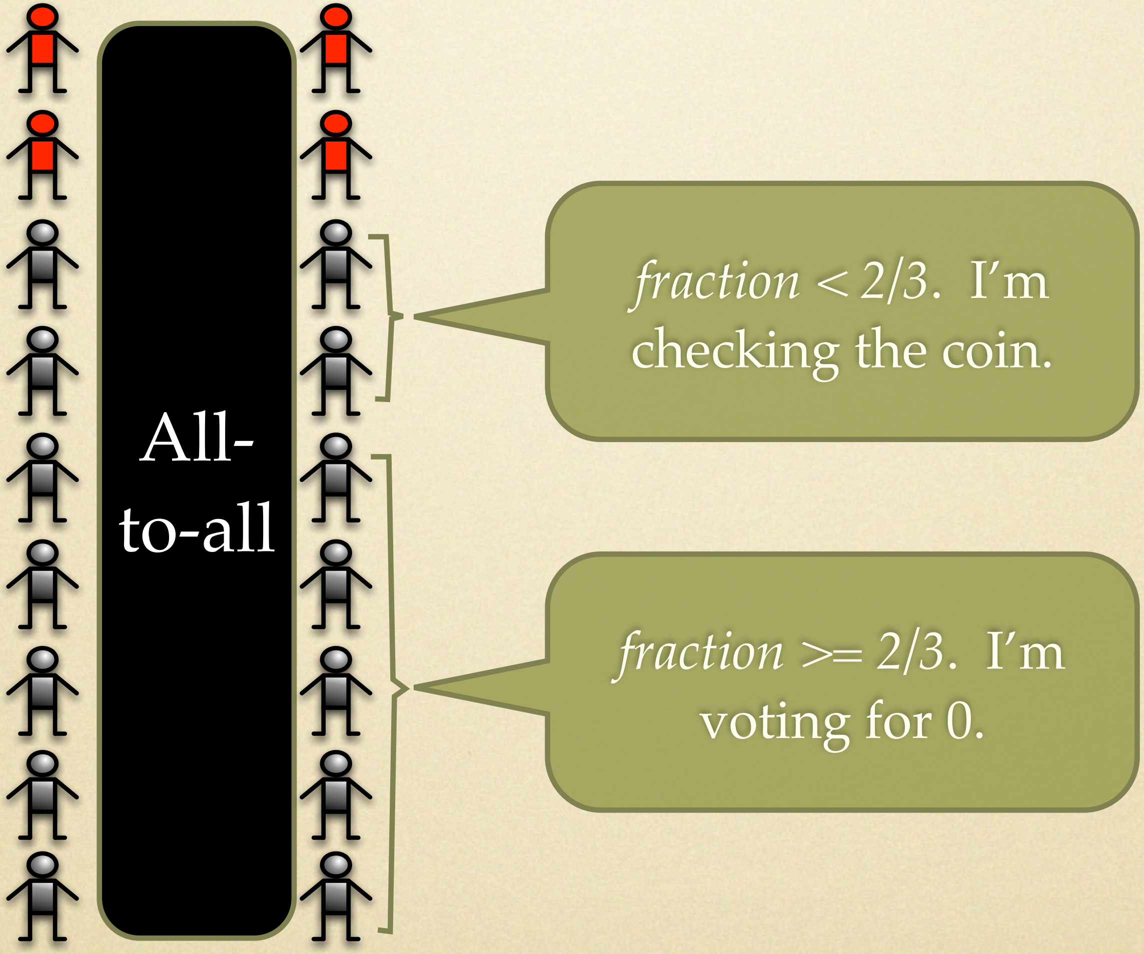
fraction >= 2/3. I’m voting for 0.



fraction < 2/3. I'm checking the coin.

fraction >= 2/3. I'm voting for 0.

Probability 1/2 that both groups change vote to the same value



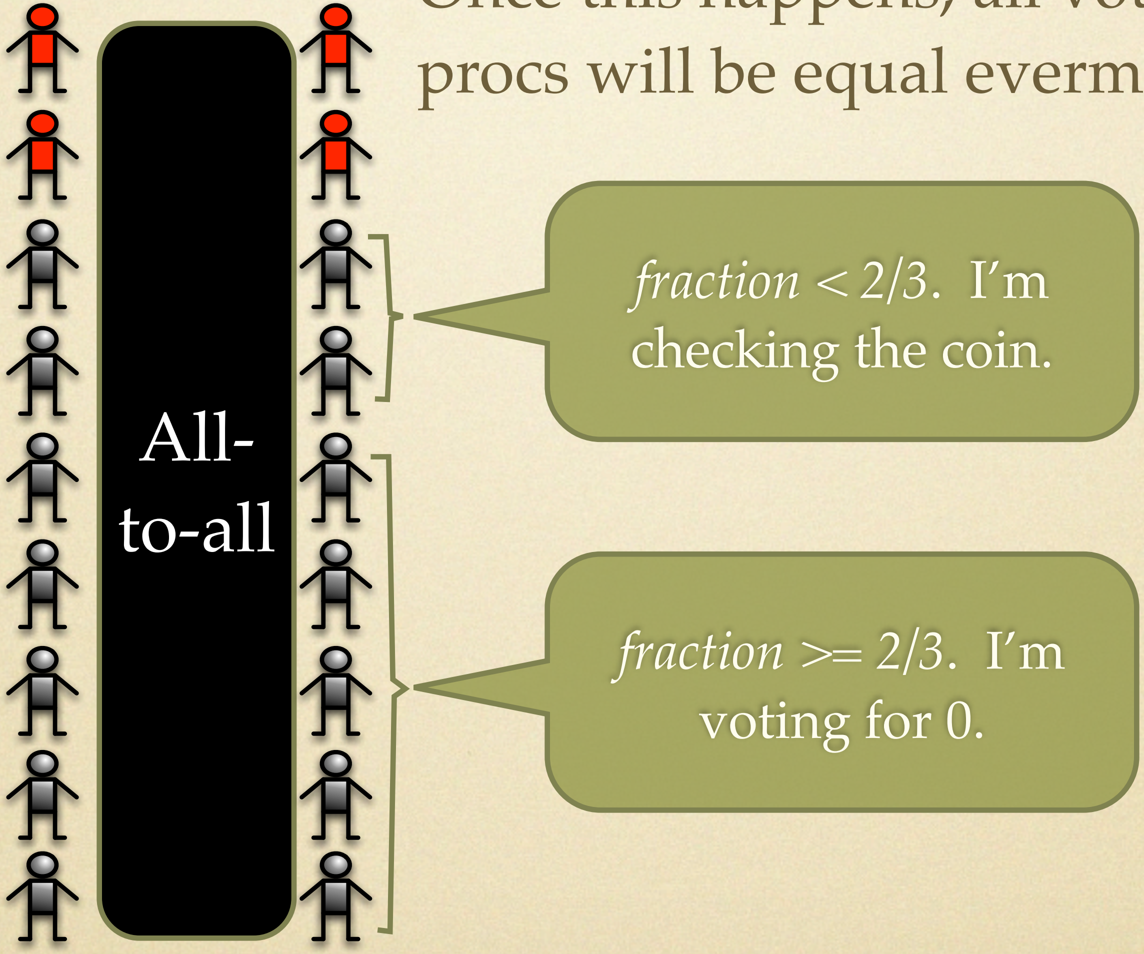
All-to-all

fraction < 2/3. I'm checking the coin.

fraction >= 2/3. I'm voting for 0.

Probability 1/2 that both groups change vote to the same value

Once this happens, all votes of good procs will be equal evermore

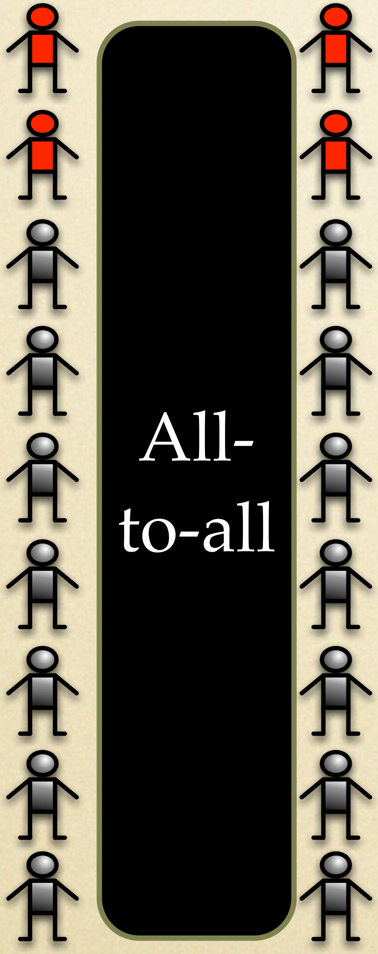


All-to-all

fraction < 2/3. I'm checking the coin.

fraction >= 2/3. I'm voting for 0.

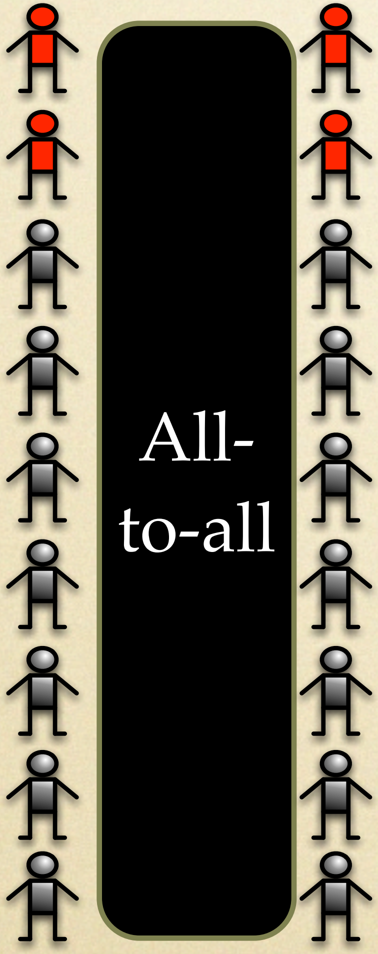
Probability 1/2 that both groups change vote to the same value



Once this happens, all votes of good procs will be equal evermore

Prob of failure $= (1/2)^{c \log n}$

Probability 1/2 that both groups change vote to the same value

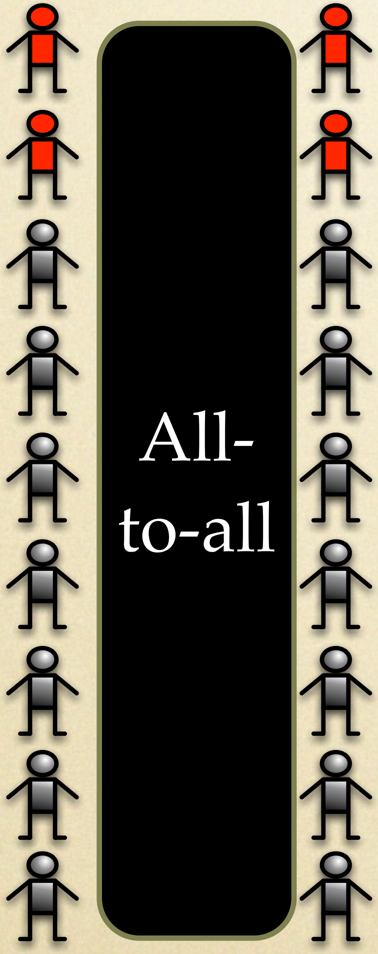


Once this happens, all votes of good procs will be equal evermore

$$\begin{aligned}\text{Prob of failure} &= (1/2)^{clogn}\\ &= 1/n^c\end{aligned}$$

Probability 1/2 that both groups change vote to the same value

Once this happens, all votes of good procs will be equal evermore



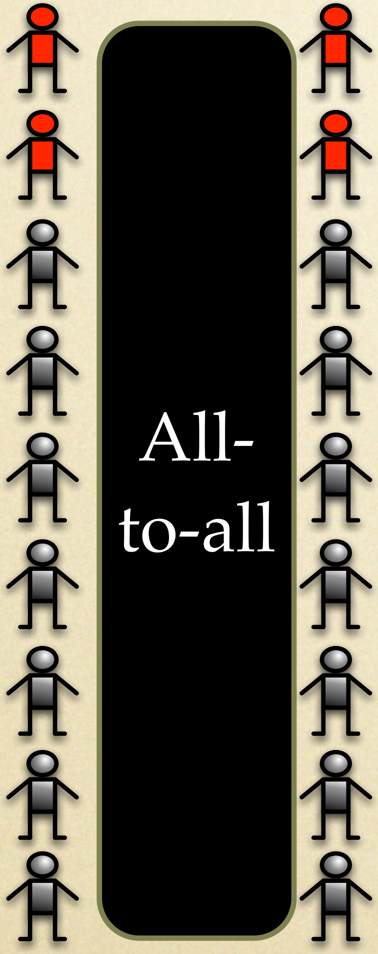
All-to-all

$$\text{Prob of failure} = (1/2)^{clogn}$$

$$= 1/n^c$$

$$\text{Prob of success} \ = 1 - 1/n^c$$

Probability 1/2 that both groups change vote to the same value



Once this happens, all votes of good procs will be equal evermore

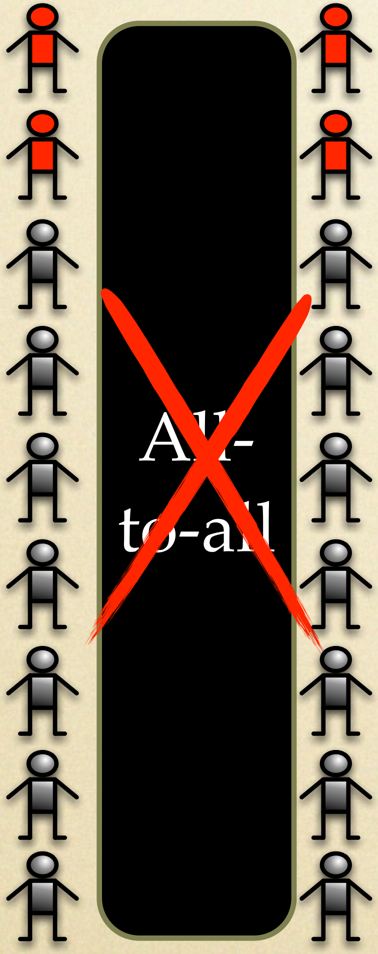
$$\text{Prob of failure} = (1/2)^{clogn}$$

$$= 1/n^c$$

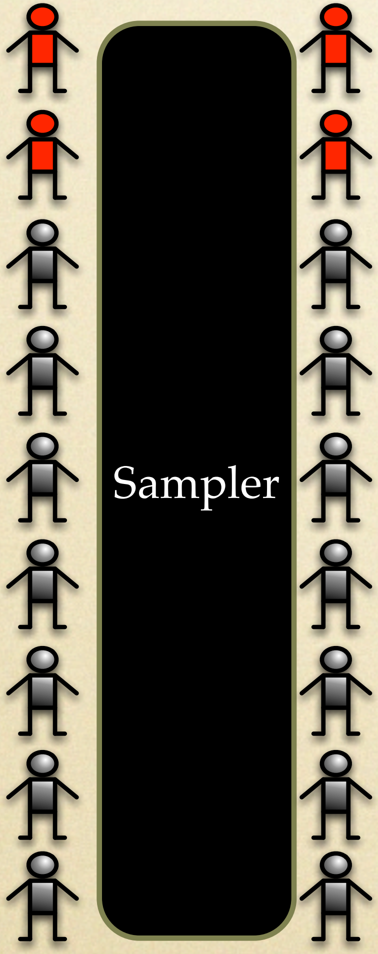
$$\text{Prob of success} = \boxed{1 - 1/n^c}$$

whp

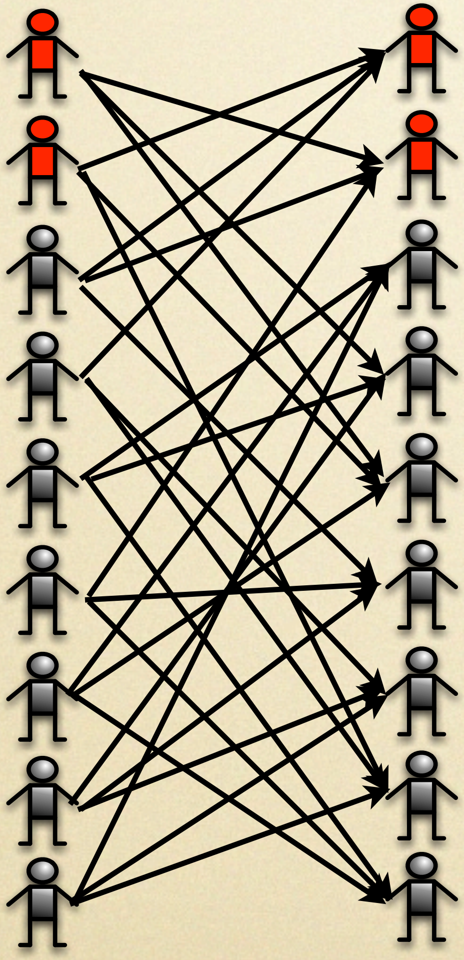
Scalable a.e.BA w/ GC



Scalable a.e.BA w/ GC

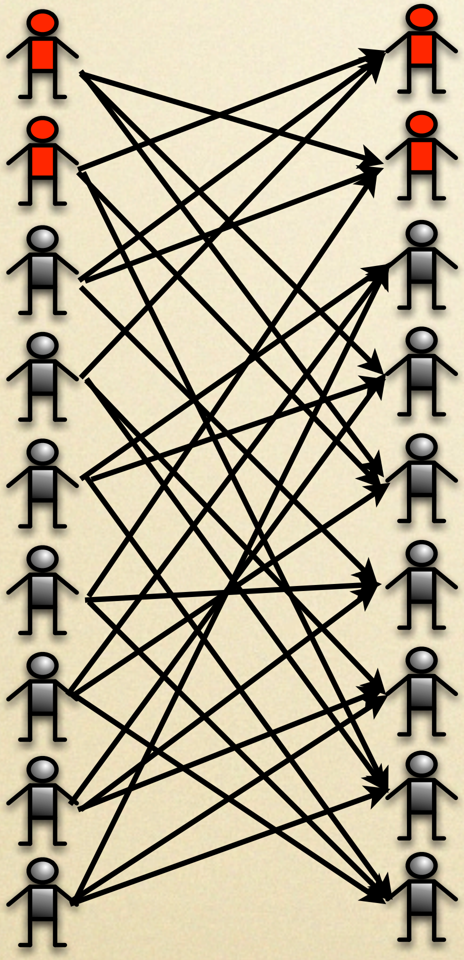


Scalable a.e.BA w/ GC



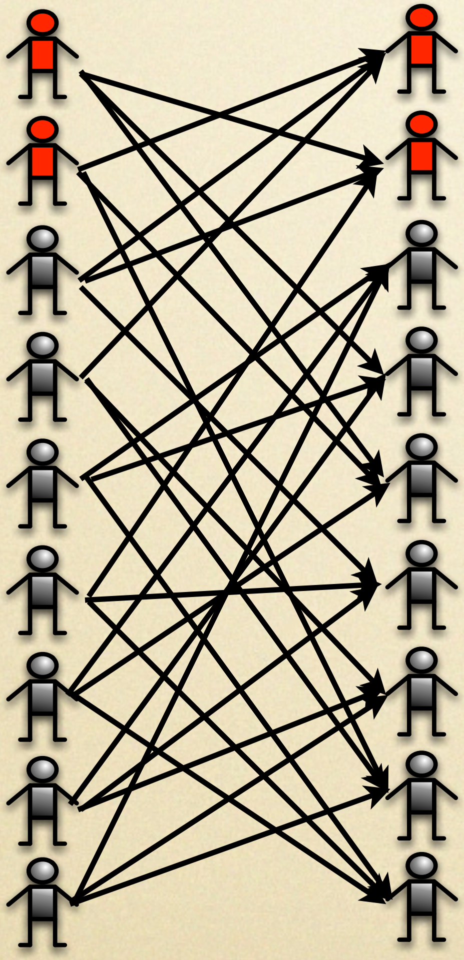
A **sampler** is a sparse graph ensuring that almost everyone on right has a fraction of bad neighbors ~ t/n

Scalable a.e.BA w/ GC



A **sampler** is a sparse graph ensuring that $\geq 1 - \delta$ fraction on right has a fraction of bad neighbors $\leq t/n + \theta$

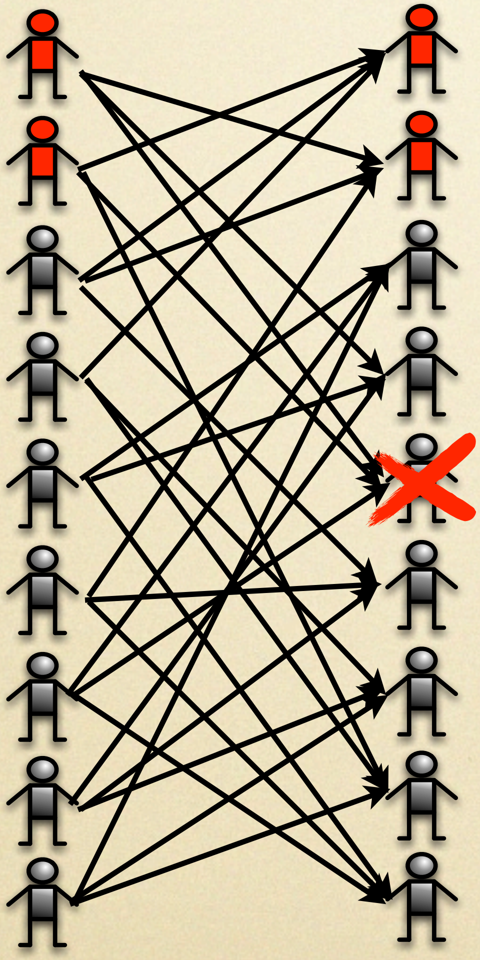
Scalable a.e.BA w/ GC



A **sampler** is a sparse graph ensuring that $\geq 1 - \delta$ fraction on right has a fraction of bad neighbors $\leq t/n + \theta$

No matter which subset is bad!

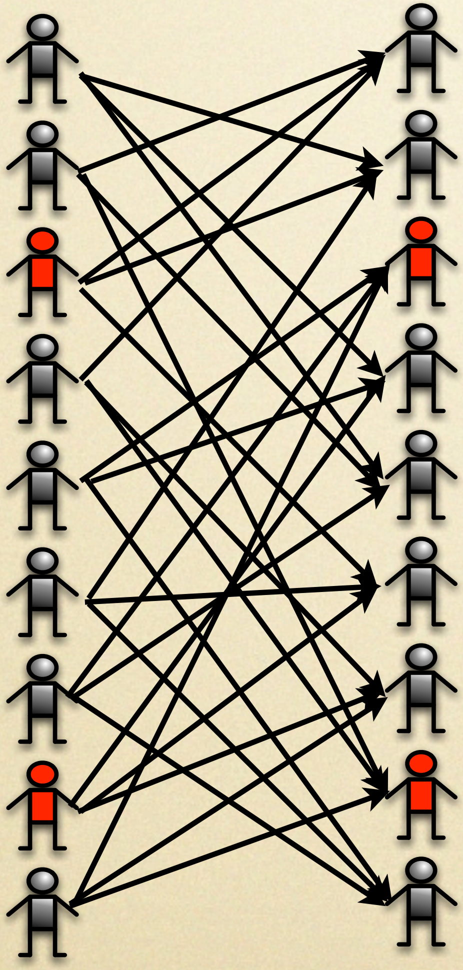
Scalable a.e.BA w/ GC



A **sampler** is a sparse graph ensuring that $\geq 1 - \delta$ fraction on right has a fraction of bad neighbors $\leq t/n + \theta$

No matter which subset is bad!

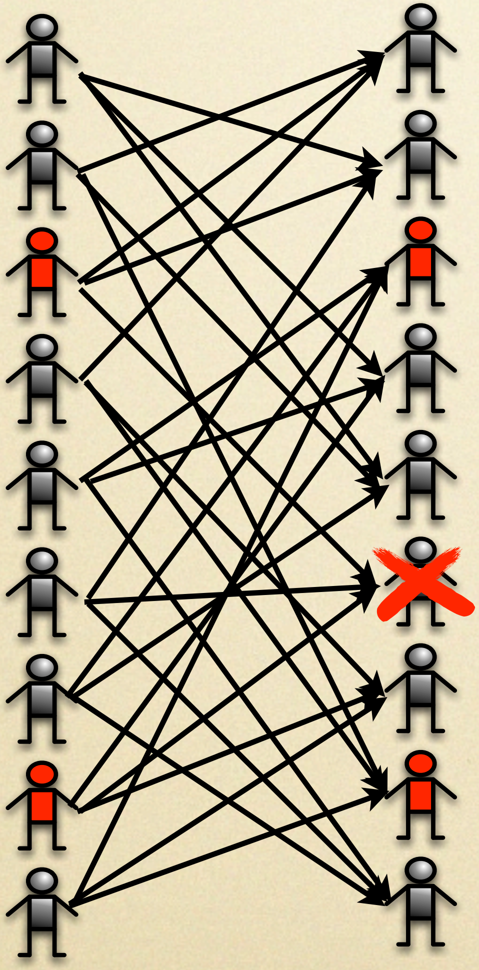
Scalable a.e.BA w/ GC



A **sampler** is a sparse graph ensuring that $\geq 1 - \delta$ fraction on right has a fraction of bad neighbors $\leq t/n + \theta$

No matter which subset is bad!

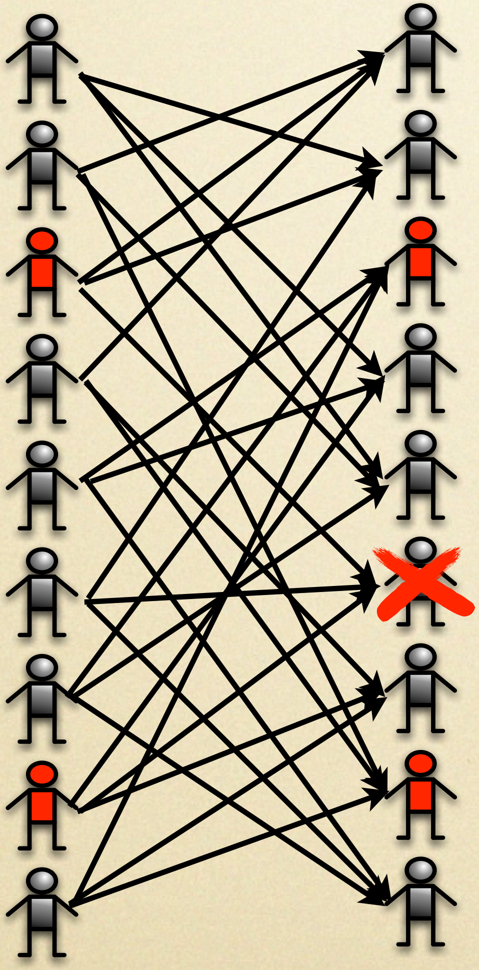
Scalable a.e.BA w/ GC



A **sampler** is a sparse graph ensuring that $\geq 1-\delta$ fraction on right has a fraction of bad neighbors $\leq t/n+\theta$

No matter which subset is bad!

Scalable a.e.BA w/ GC



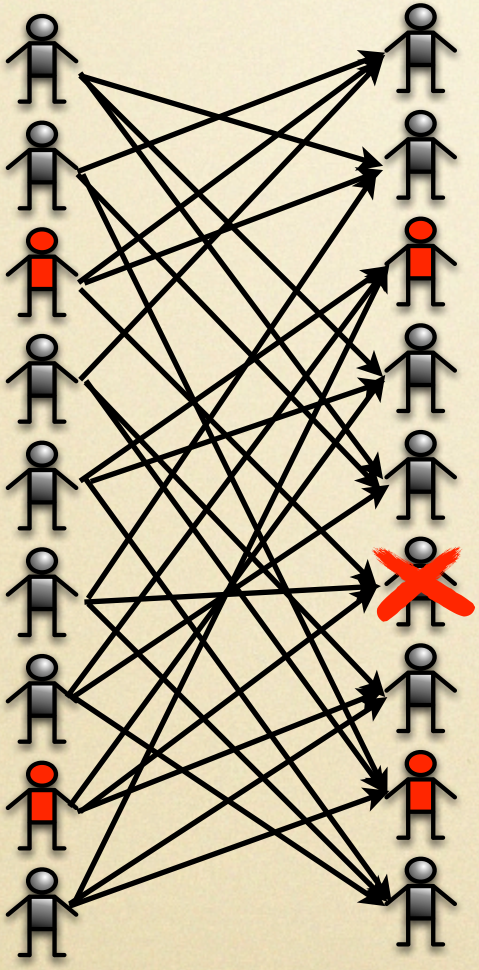
A **sampler** is a sparse graph ensuring that $\geq 1 - \delta$ fraction on right has a fraction of bad neighbors $\leq t/n + \theta$

and the degree is just:

$$\frac{2-\delta}{\theta^2\delta \cdot 2\log_2 e}$$

No matter which subset is bad!

Scalable a.e.BA w/ GC



A **sampler** is a sparse graph ensuring that $\geq 1-\delta$ fraction on right has a fraction of bad neighbors $\leq t/n+\theta$

and the degree is just:

$$\frac{2-\delta}{\theta^2\delta \cdot 2\log_2 e}$$

$= O(\log n)$ if $\delta = 1/\log n$ and $\theta = O(1)$

No matter which subset is bad!

BA with Global Coin, GC

Rabin's Algorithm

```
Set your vote to input bit
Repeat clogn times:
    Send your vote to everyone
    Let fraction be fraction of votes for majority bit
    If fraction >= 2/3, set vote to majority bit;  else set
        vote to GC
Output your vote
```

BA with Global Coin, GC

Rabin's Algorithm

Set your vote to input bit

Repeat clogn times:

neighbors in sampler

Send your vote to ~~everyone~~

Let *fraction* be fraction of votes for majority bit

If *fraction* >= 2/3, set vote to majority bit; else set vote to GC

Output your vote

BA with Global Coin, GC

Rabin's Algorithm

```
Set your vote to input bit

Repeat clogn times:
                                neighbors in sampler
    Send your vote to ev~~eryo~~ne

    Let *fraction* be fraction of votes for majority bit

    If *fraction* >= 2/3, set vote to majority bit;  else set
        vote to GC ⟵ ?

Output your vote
```

BA with Global Coin, GC

Rabin's Algorithm

```
Set your vote to input bit

Repeat clogn times:

    Send your vote to ~~everyone~~ neighbors in sampler

    Let *fraction* be fraction of votes for majority bit

    If *fraction* >= 2/3, set vote to majority bit;  else set
        vote to ~~GC~~

Output your vote
```

BA with Global Coin, GC

Rabin's Algorithm

Set your vote to input bit

Repeat clogn times:

- Send your vote to ~~everyone~~ neighbors in sampler
- Let *fraction* be fraction of votes for majority bit
- If *fraction* >= 2/3, set vote to majority bit; else set vote to ~~GC~~ s_i

Output your vote

BA with Global Coin, GC

Rabin's Algorithm

```
Set your vote to input bit

Repeat clogn times:

    Send your vote to ~~everyone~~ neighbors in sampler

    Let *fraction* be fraction of votes for majority bit

    If *fraction* >= 2/3, set vote to majority bit;  else set
        vote to ~~GC~~ $S_i$

Output your vote
```

Suffices that O(log n) of the S_i are random and known a.e.

Algorithm Outline

I: Using S to get a.e. BA

II: Using S to go from a.e. BA to BA

III: Implementing S

Algorithm Outline

I: Using S to get a.e. BA ✓

II: Using S to go from a.e. BA to BA

III: Implementing S

Flooding!

- Idea: Query random set of procs to ask bit - take majority
- Problem: In our model, the adversary can flood all procs with queries!
- Idea: Use S to decide which queries to answer.

Flooding!

- Idea: Query random set of procs to ask bit - take majority
- Problem: In our model, the adversary can flood all procs with queries!
- Idea: Use S to decide which queries to answer.
- Each query will have a tag between 1 and $\sqrt{n}$
- The elements of S will now be numbers between 1 and $\sqrt{n}$

a.e. BA to BA

For i = 1 to to c log n:

- Each proc. p picks $k\sqrt{n}\log n$ random queries <proc,tag> and sends tag to proc.
- q answers only if tag = s_i (and not overloaded)
- if 2/3 majority of p's queries with the same tag are returned and agree on b, then p decides b.

a.e. BA to BA

For i = 1 to to c log n:

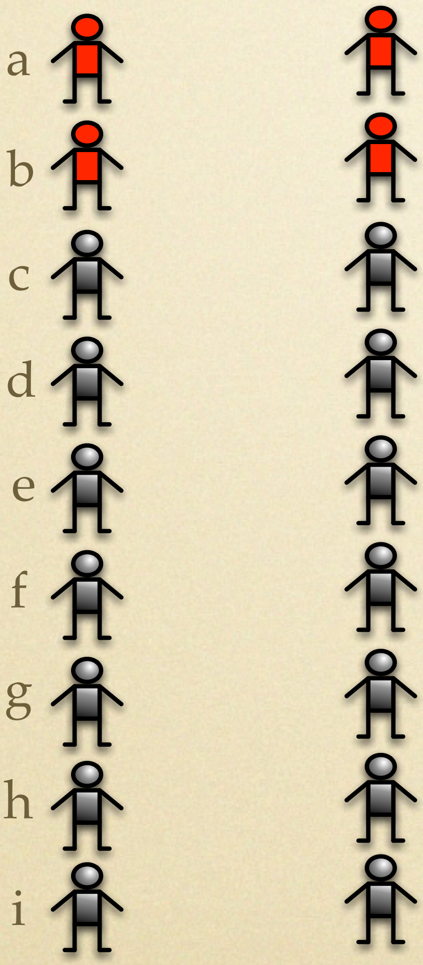
- Each proc. p picks $k\sqrt{n}\log n$ random queries <proc,tag> and sends tag to proc.
- q answers only if tag = s_i (and q received $\sqrt{n}\log n$ queries with this tag)
- if 2/3 majority of p's queries with the same tag are returned and agree on b, then p decides b.

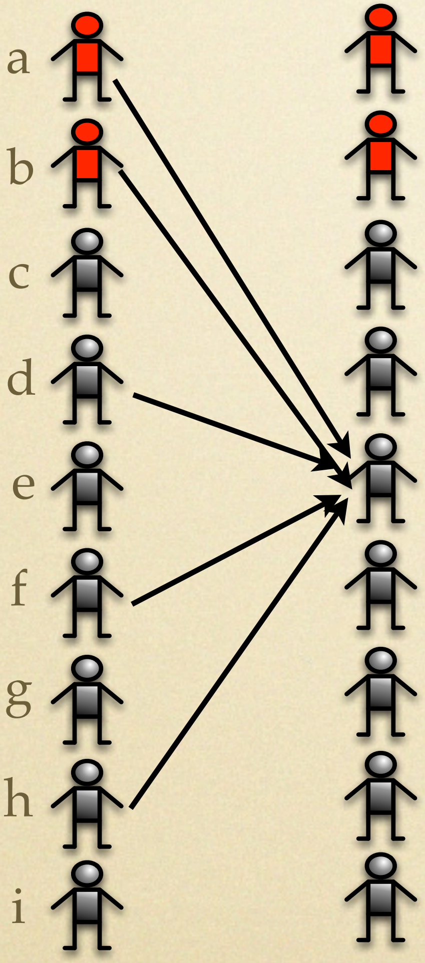
a.e. BA to BA

without replacement with replacement

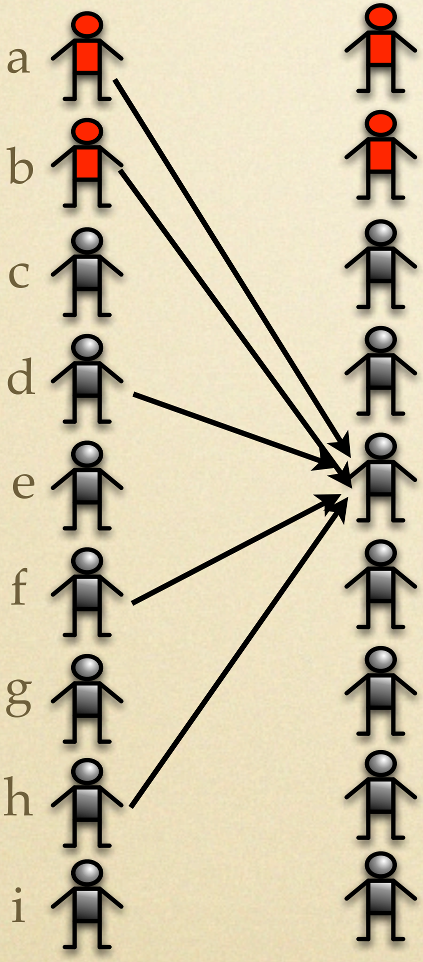
For i = 1 to to c log n:

- Each proc. p picks $k\sqrt{n}\log n$ random queries <proc,tag> and sends tag to proc.
- q answers only if tag = s_i (and q received $\sqrt{n}\log n$ queries with this tag)
- if 2/3 majority of p's queries with the same tag are returned and agree on b, then p decides b.


a
b
c
d
e
f
g
h
i

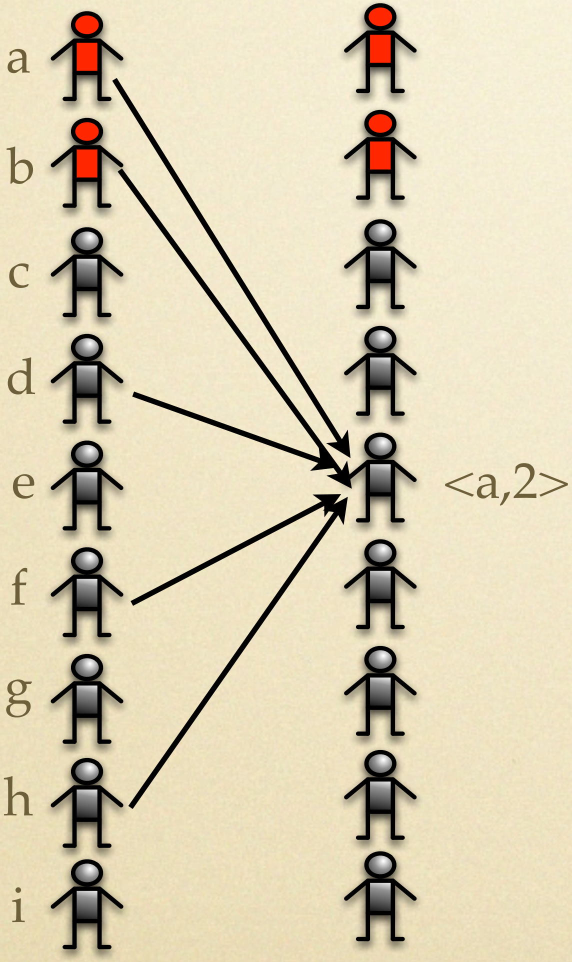

a
b
c
d
e
f
g
h
i

<a,2> <b,1> <d,1> <f,2> <h,3>



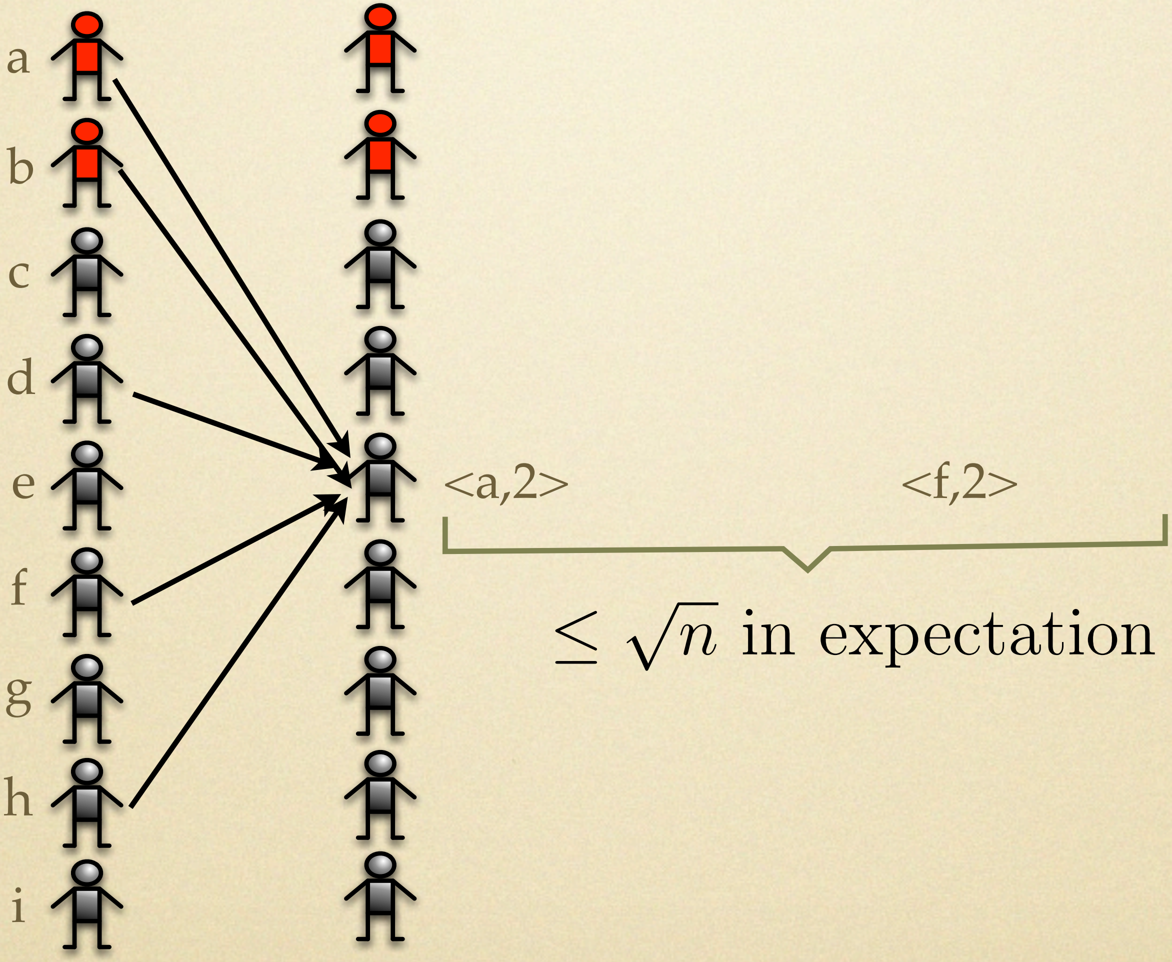
$$\underbrace{\langle a,2\rangle \quad \langle b,1\rangle \quad \langle d,1\rangle \quad \langle f,2\rangle \quad \langle h,3\rangle}_{\leq n}$$

$S_i = 2$



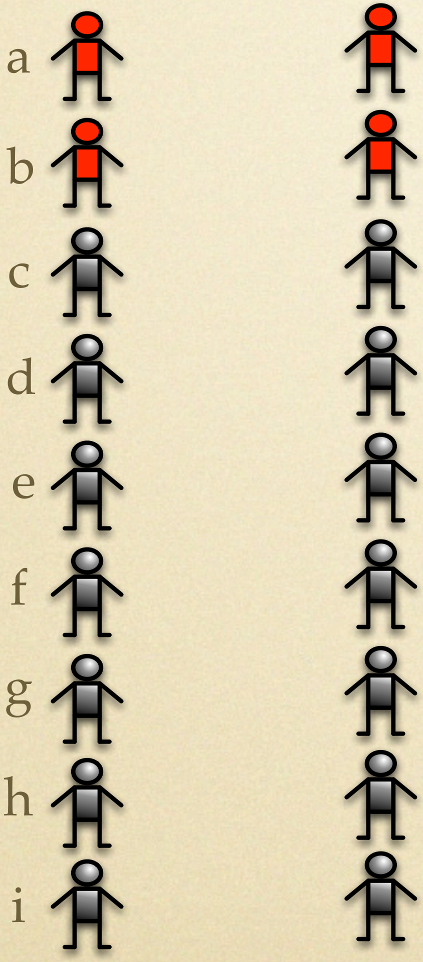
<f,2>

$S_i = 2$



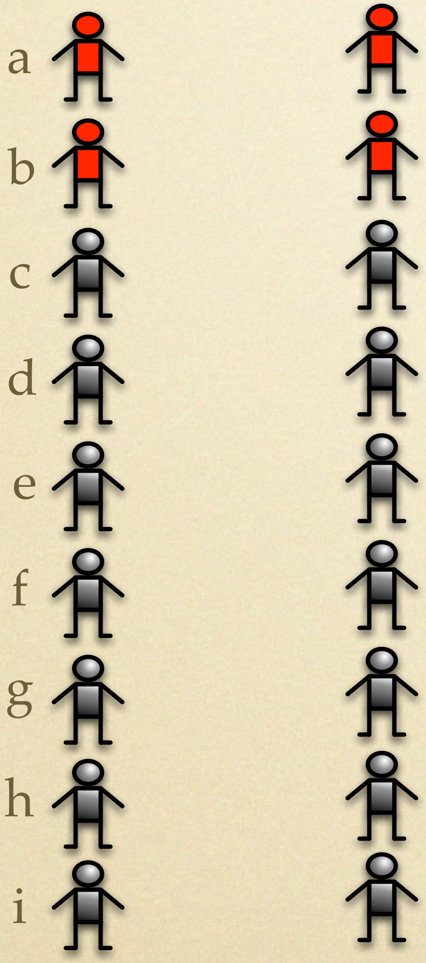
$\leq \sqrt{n}$ in expectation

Analysis



Each proc receives $\leq n$ requests

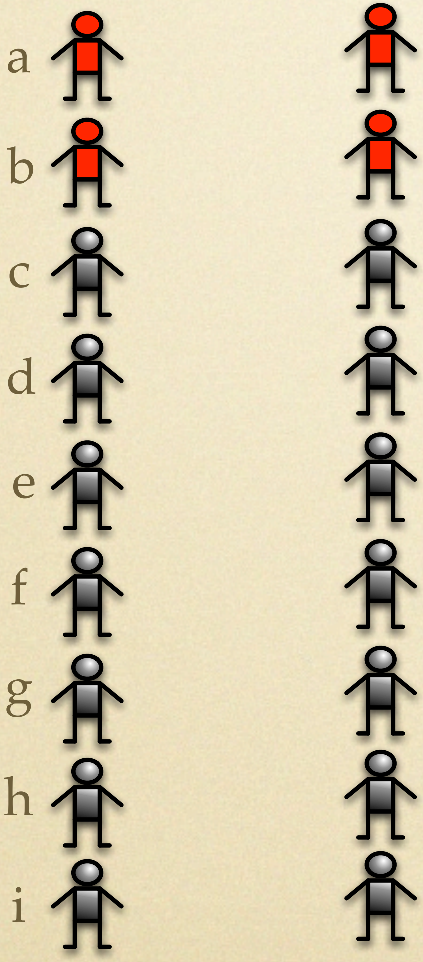
Analysis



Each proc receives $\leq n$ requests

So expected # requests with tags that match $\textcolor{blue}{s_i}$ is $\leq \sqrt{n}$

Analysis

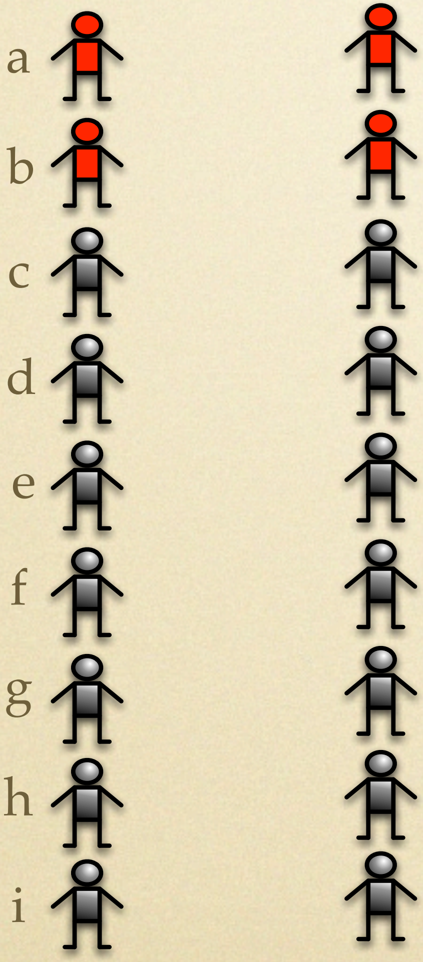


Each proc receives $\leq n$ requests

So expected # requests with tags that match s_i is $\leq \sqrt{n}$

So in any loop, w/ prob >=1/2, $\leq \epsilon$ fraction of procs overloaded

Analysis



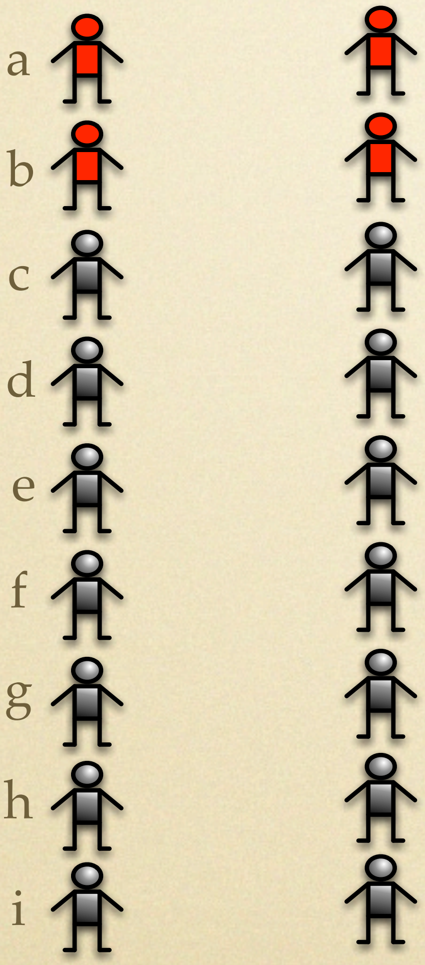
Each proc receives $\leq n$ requests

So expected # requests with tags that match $\color{blue}{\mathrm{s_i}}$ is $\leq \sqrt{n}$

So in any loop, w/ prob >=1/2, $\leq \epsilon$ fraction of procs overloaded

whp, some loop iteration is “good”: $\leq \epsilon$ fraction of overloaded procs

Analysis



Each proc receives $\leq n$ requests

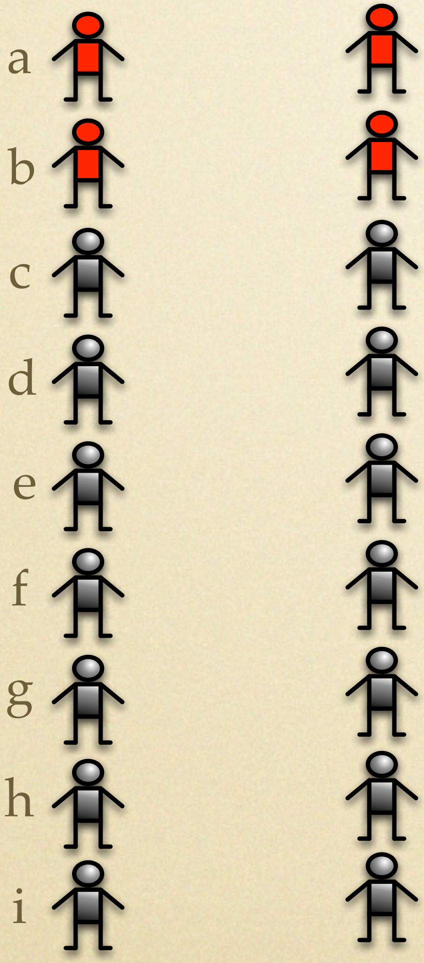
So expected # requests with tags that match s_i is $\leq \sqrt{n}$

So in any loop, w/ prob >=1/2, $\leq \epsilon$ fraction of procs overloaded

(By Linearity & Markov's inequality)

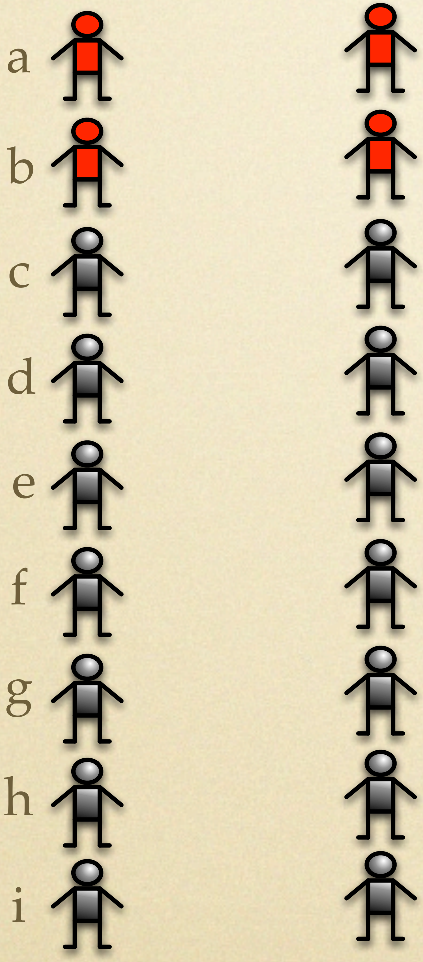
whp, some loop iteration is "good": $\leq \epsilon$ fraction of overloaded procs

Analysis



whp, some loop iteration is “good”:
$\leq \epsilon$ fraction of overloaded procs

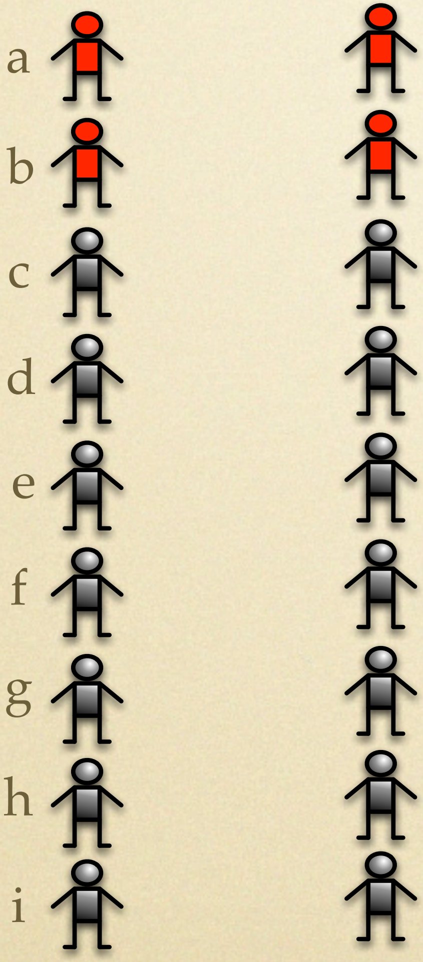
Analysis



whp, some loop iteration is “good”: $\leq \epsilon$ fraction of overloaded procs

Each good proc. sends $k\sqrt{n}\log n$ queries

Analysis

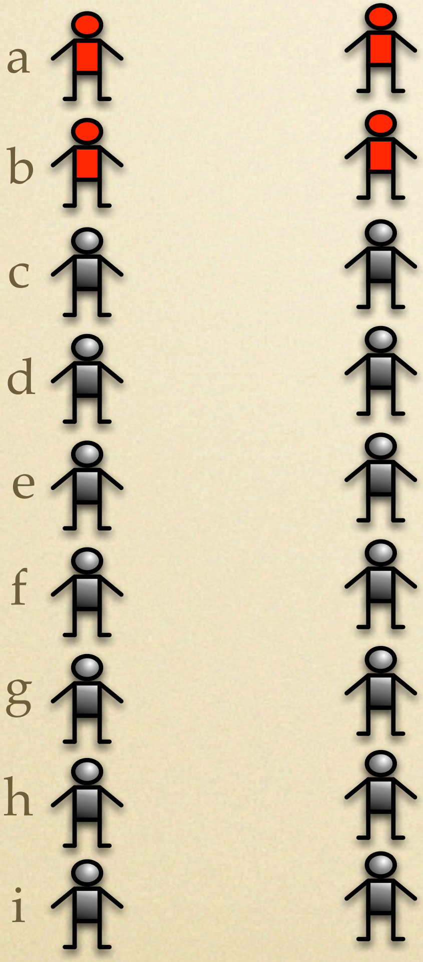


whp, some loop iteration is "good": $\leq \epsilon$ fraction of overloaded procs

Each good proc. sends $k\sqrt{n}\log n$ queries

whp O(log n) of these have tag S_i

Analysis



whp, some loop iteration is “good”: $\leq \epsilon$ fraction of overloaded procs

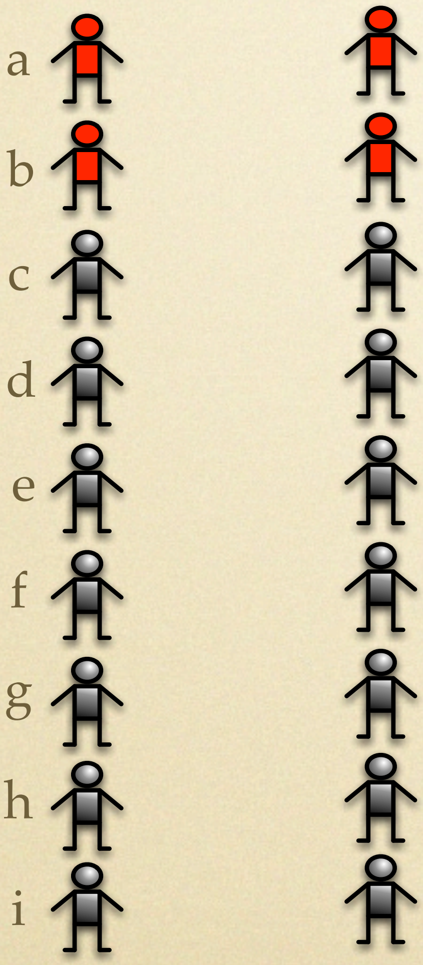
Each good proc. sends $k\sqrt{n}\log n$ queries

whp O(log n) of these have tag s_i

In a “good” iteration, a majority of queries are sent to good procs who are not overloaded

Each good proc. decides on correct bit

Analysis



whp, some loop iteration is “good”: $\leq \epsilon$ fraction of overloaded procs

Each good proc. sends $k\sqrt{n}\log n$ queries

whp O(log n) of these have tag $\mathrm{S_i}$

(by Linearity and Chernoff bounds)

In a “good” iteration, a majority of queries are sent to good procs who are not overloaded

Each good proc. decides on correct bit

Algorithm Outline

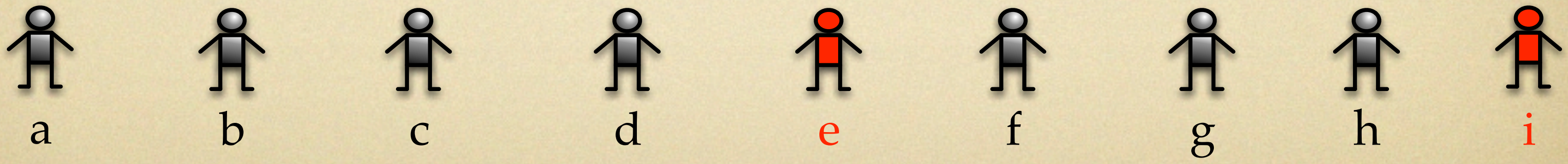
I: Using S to get a.e. BA ✓

II: Using S to go from a.e. BA to BA ✓

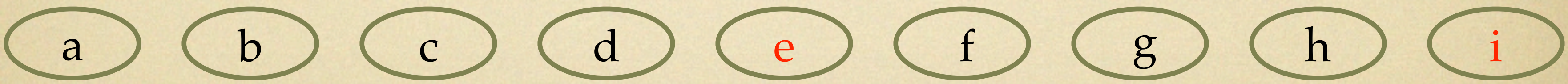
III: Implementing S

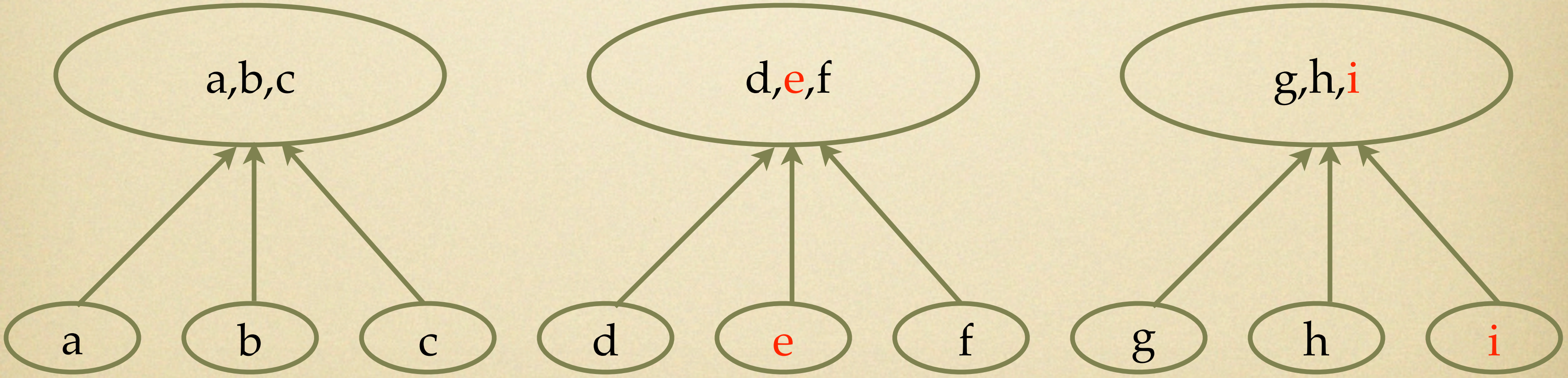
III: Implementing S

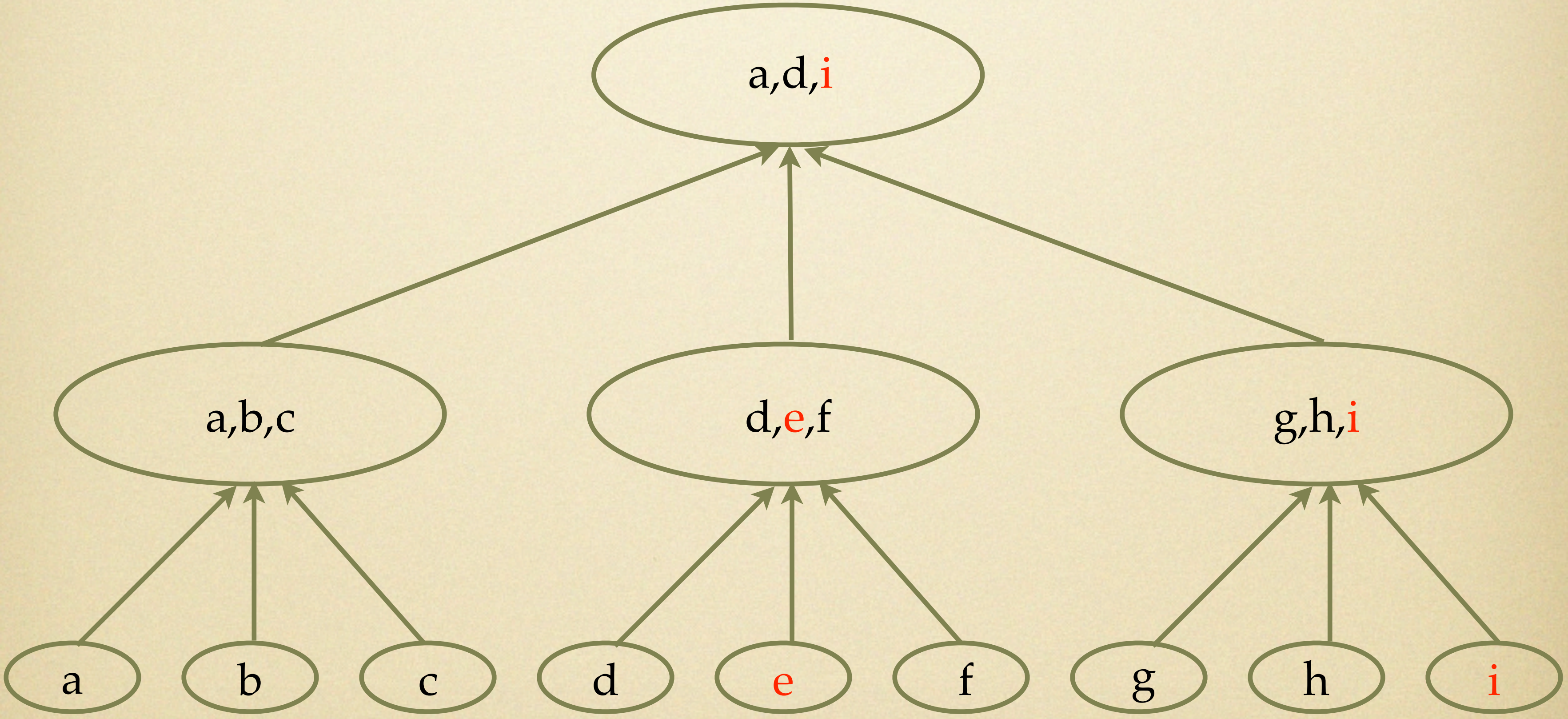
Idea: Tournament


a
b
c
d
e
f
g
h
i

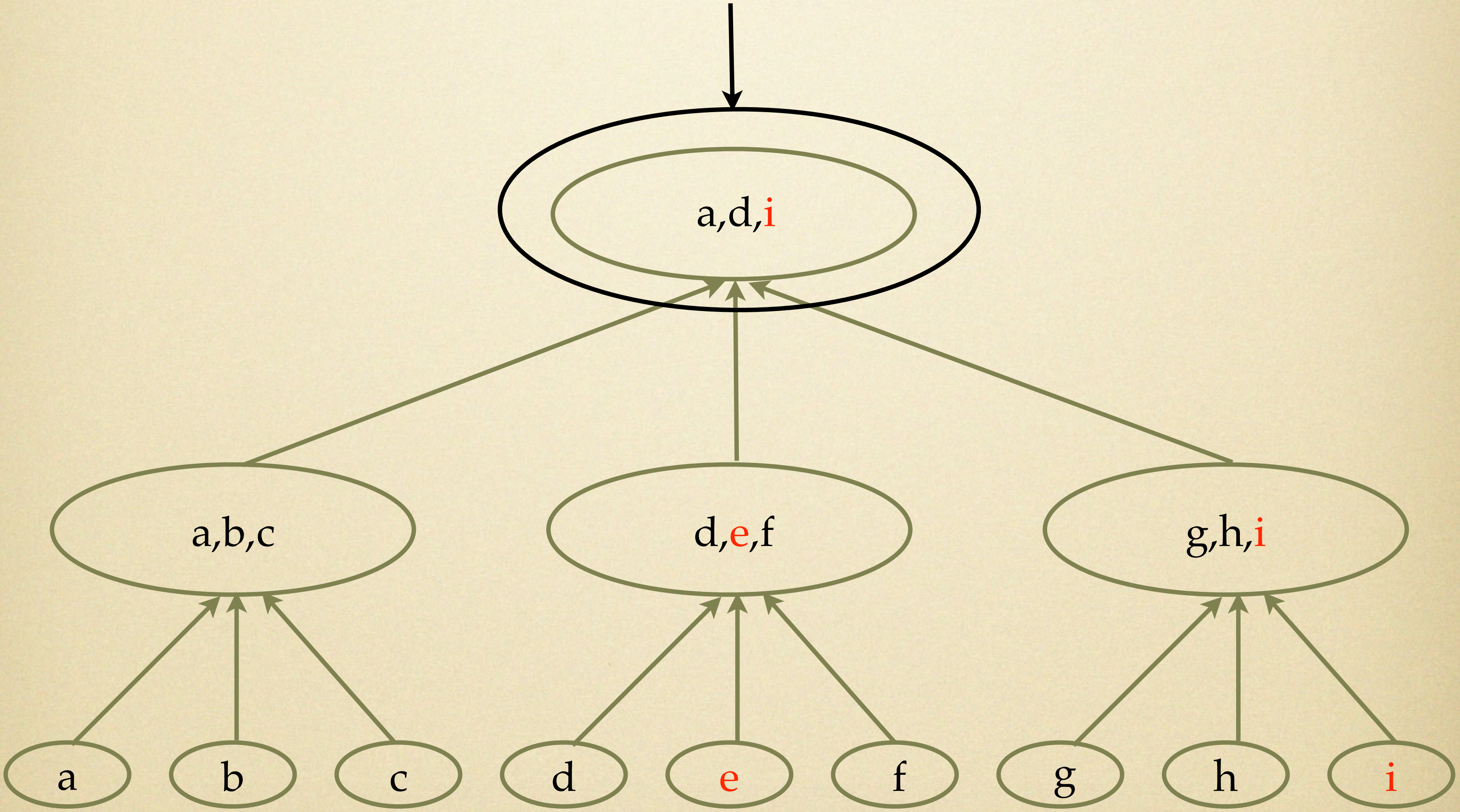
a b c d e f g h i


a
b
c
d
e
f
g
h
i

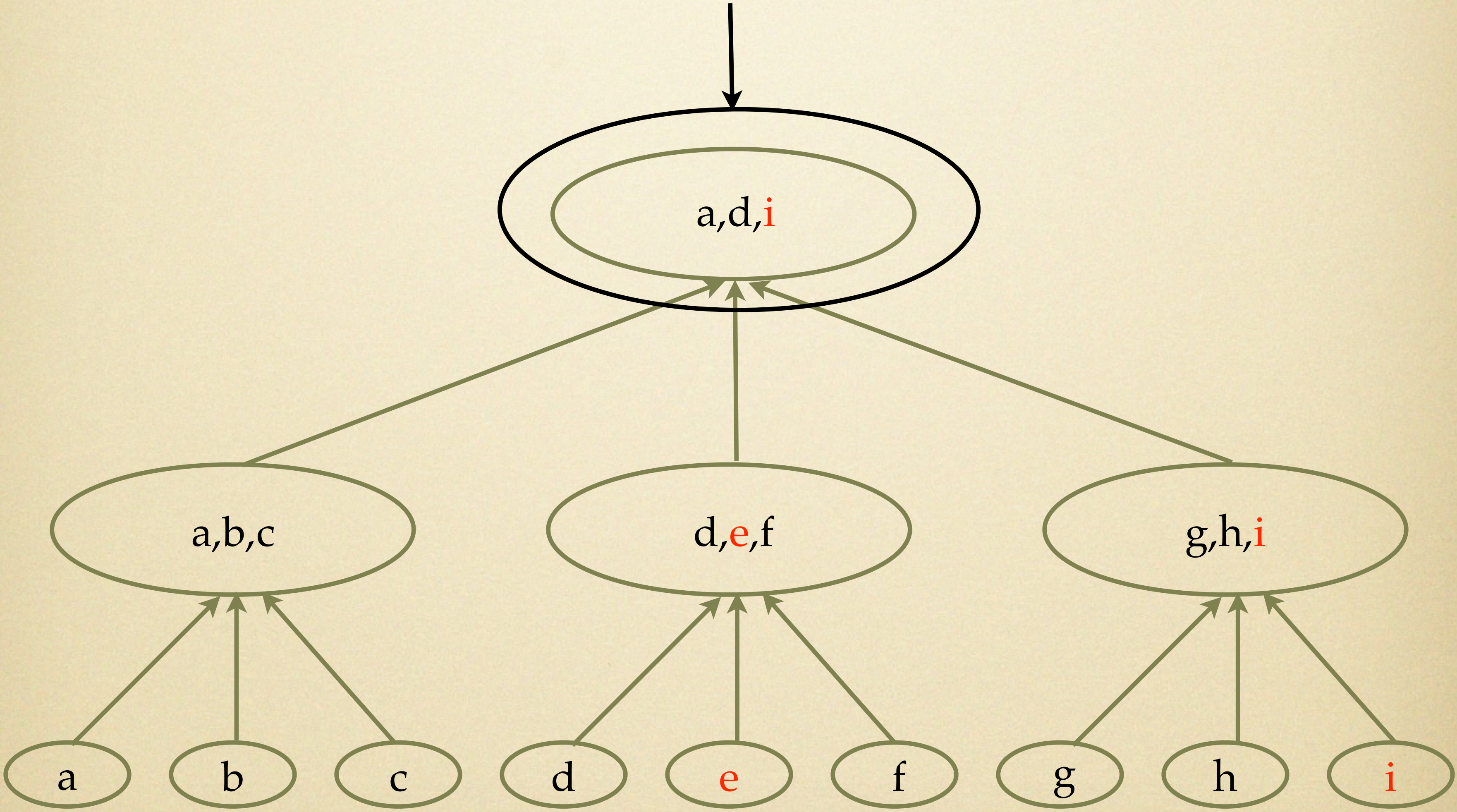

a,b,c
d,e,f
g,h,i
a
b
c
d
e
f
g
h
i


a,d,i
a,b,c
d,e,f
g,h,i
a
b
c
d
e
f
g
h
i

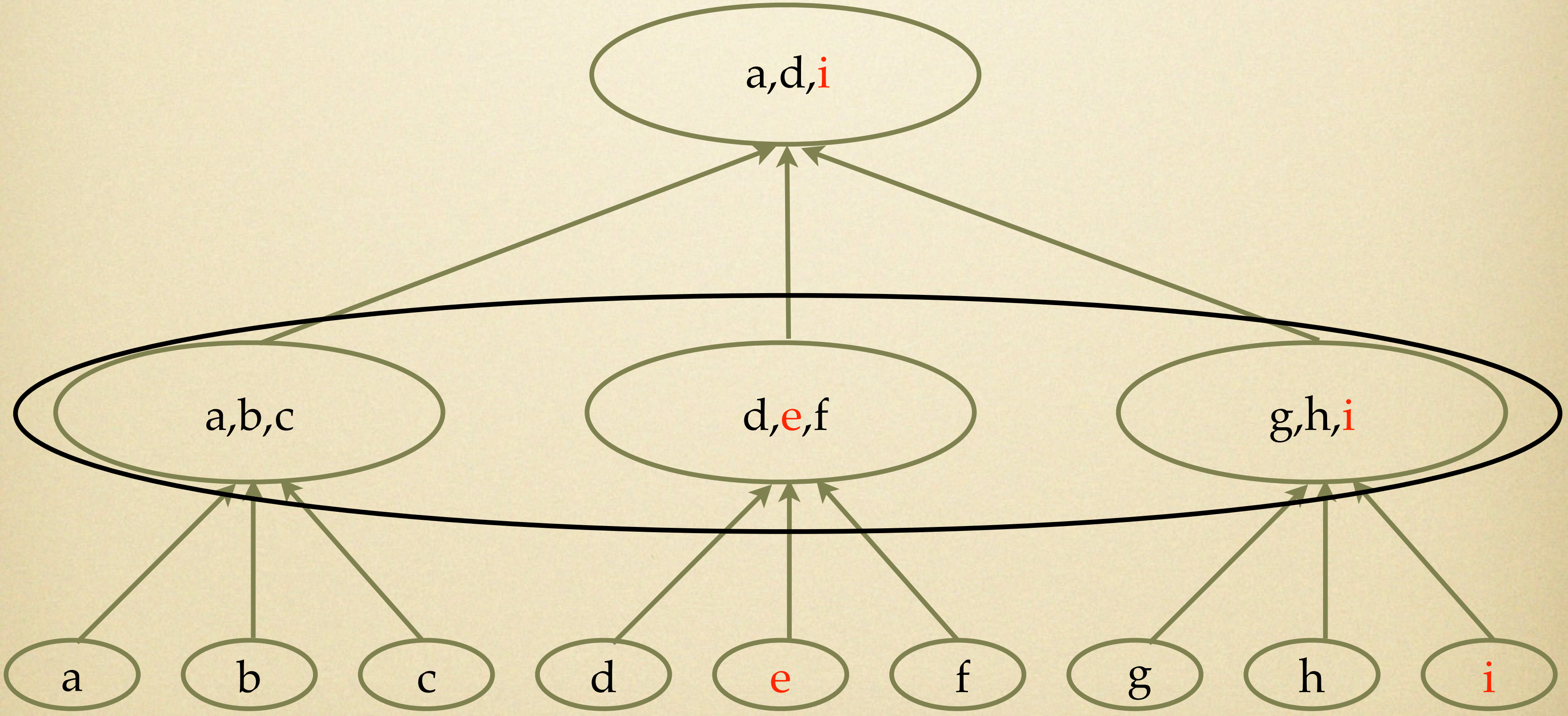
Goal: Fraction of bad procs at top supernode is not much more than t/n



Then the procs at the top super node can implement S



Problem: How to hold local elections?



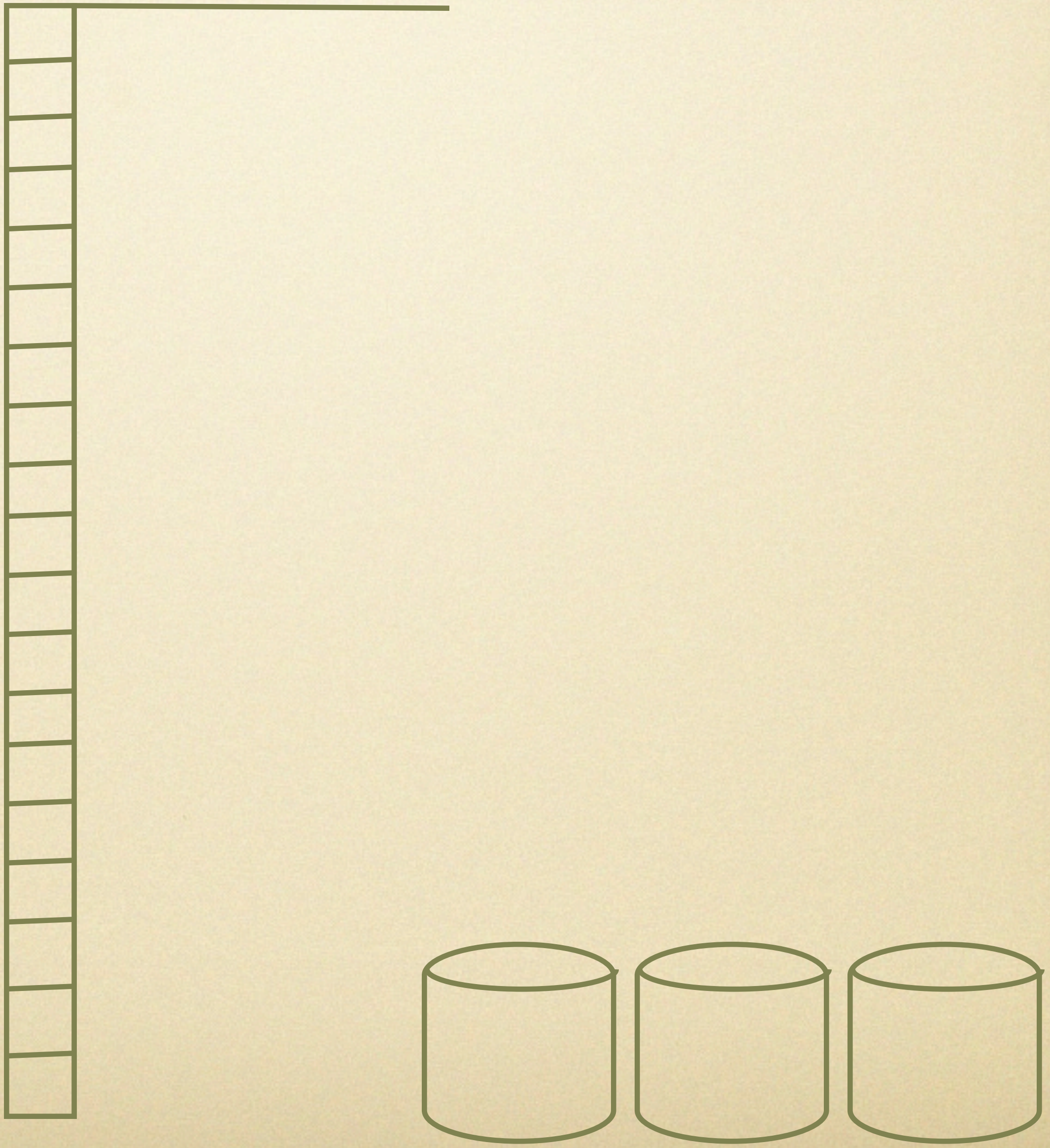
Idea: Lightest Bin Algorithm

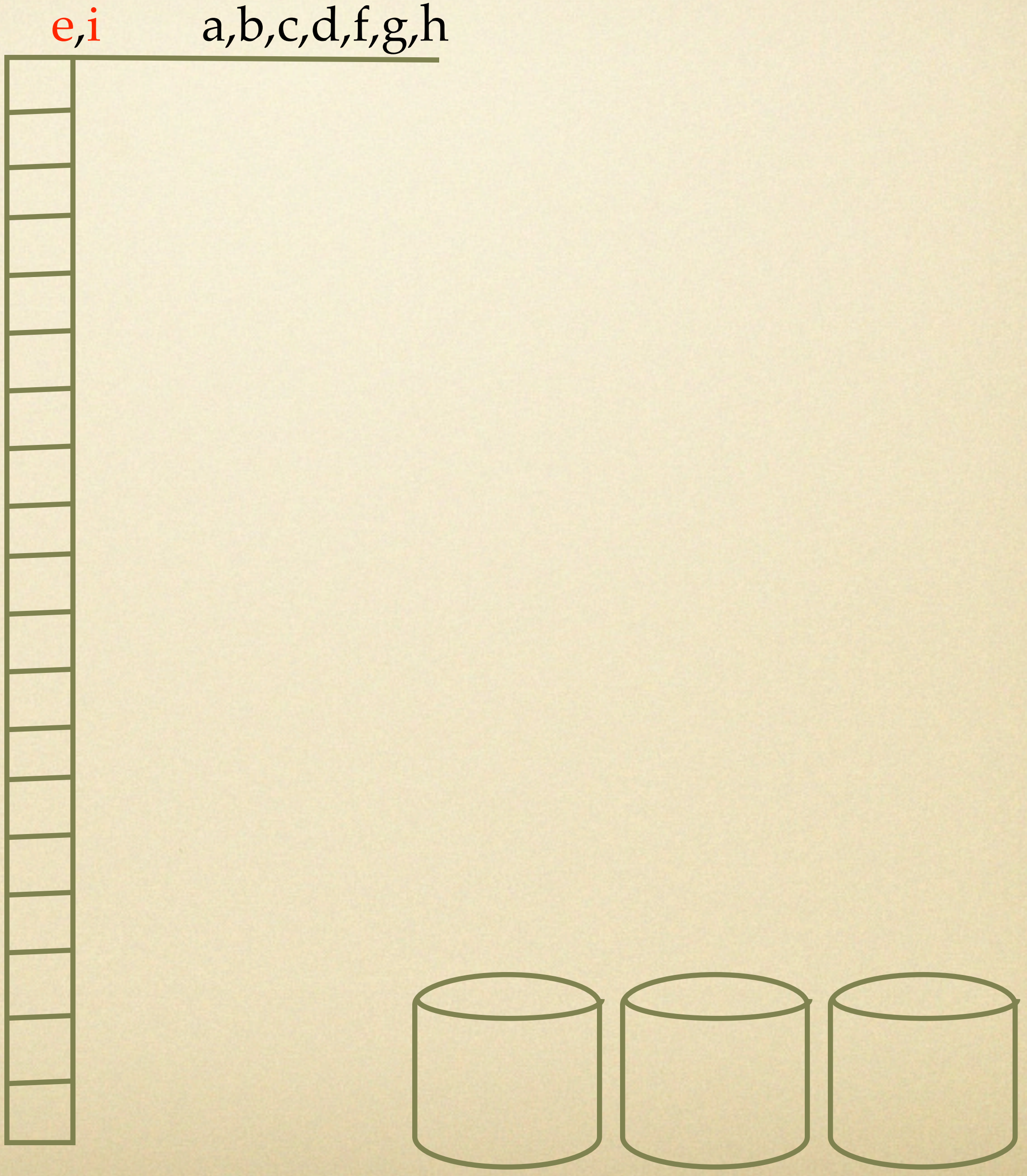
1. Each proc. picks a bin uniformly at random

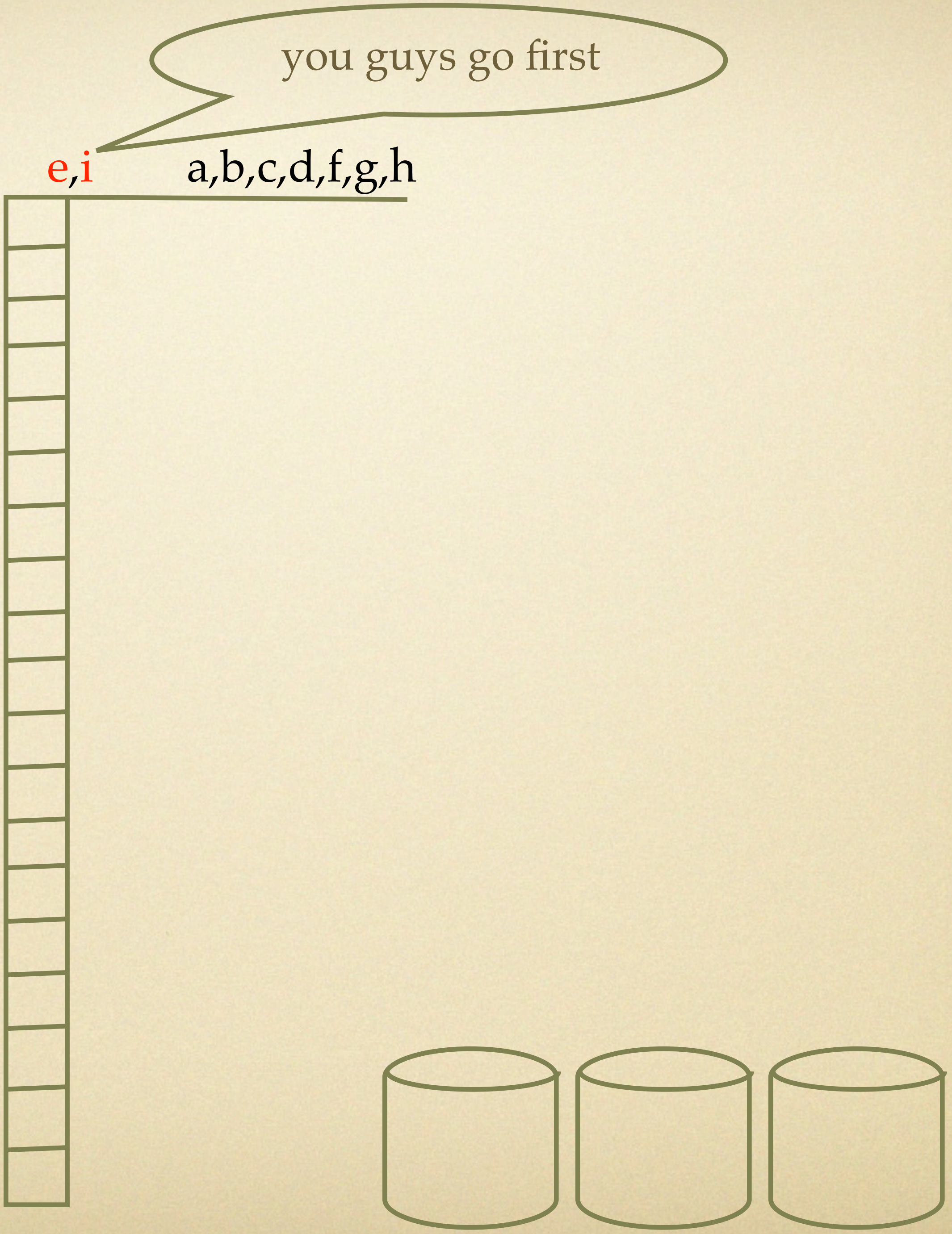
2. Winners are candidates in lightest bin

Feige

a,b,c,d,e,f,g,h,i



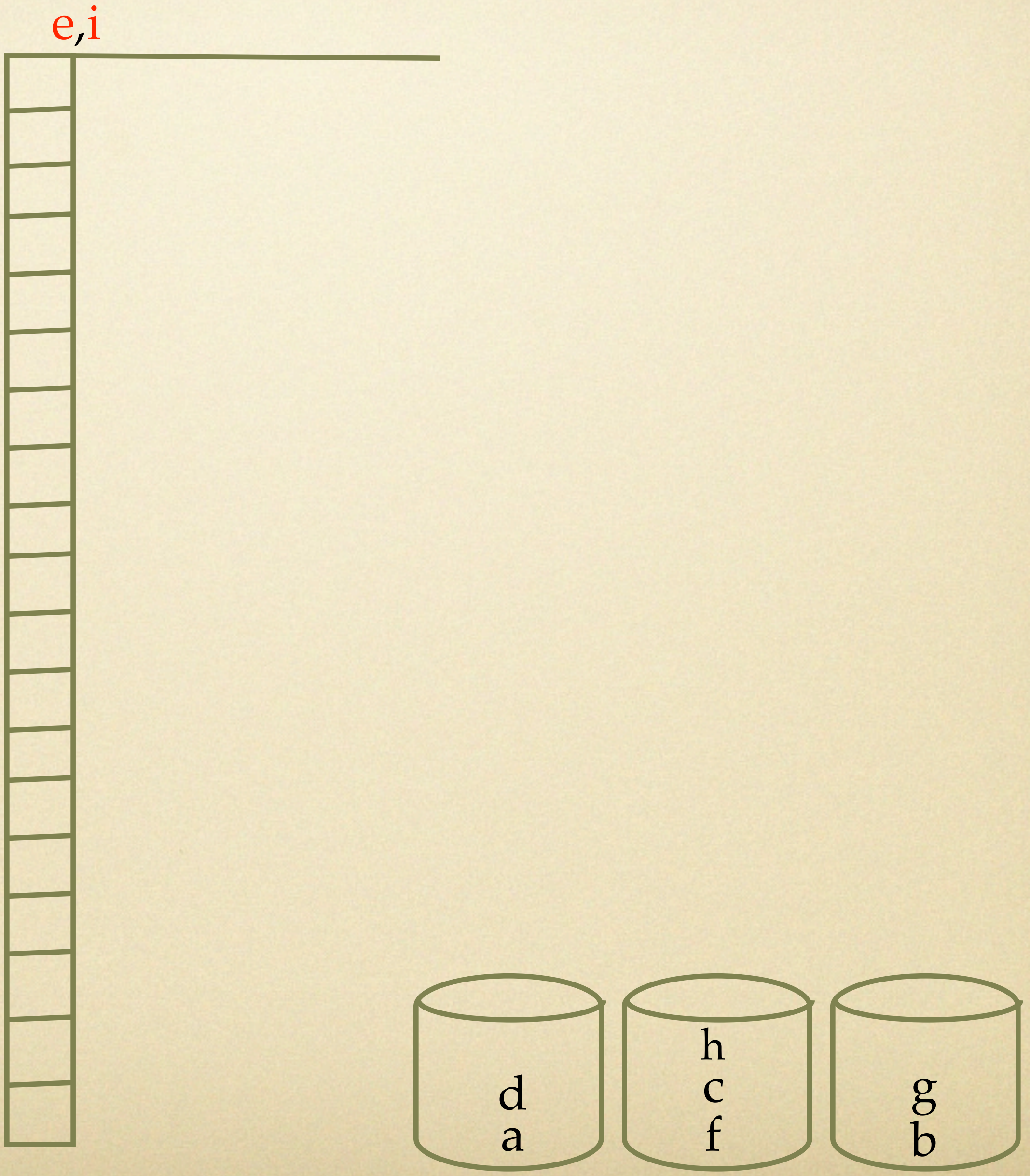

e,i
a,b,c,d,f,g,h

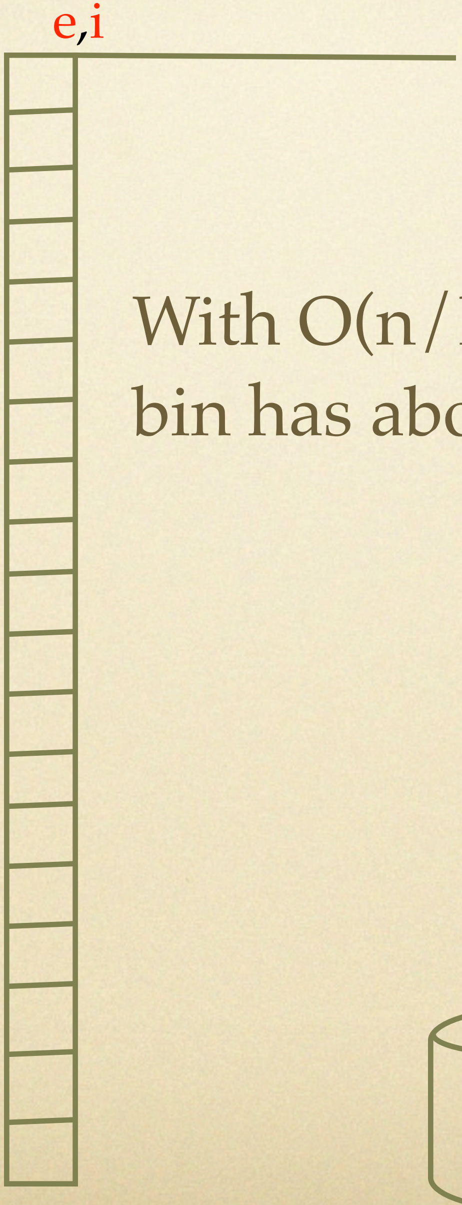


you guys go first

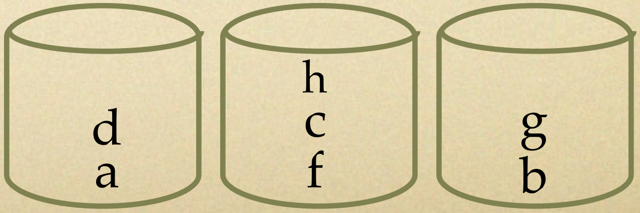
e,i

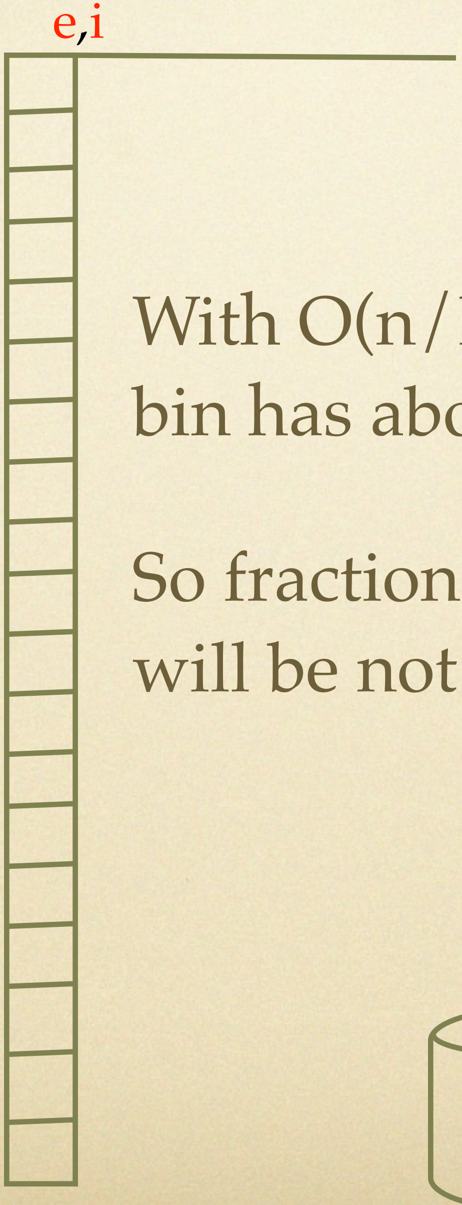
a,b,c,d,f,g,h


e,i
d
a
h
c
f
g
b



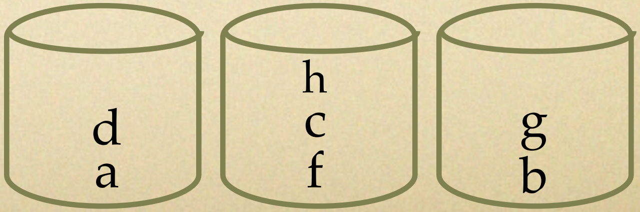
With O(n/log n) bins, whp, each bin has about same # of good procs

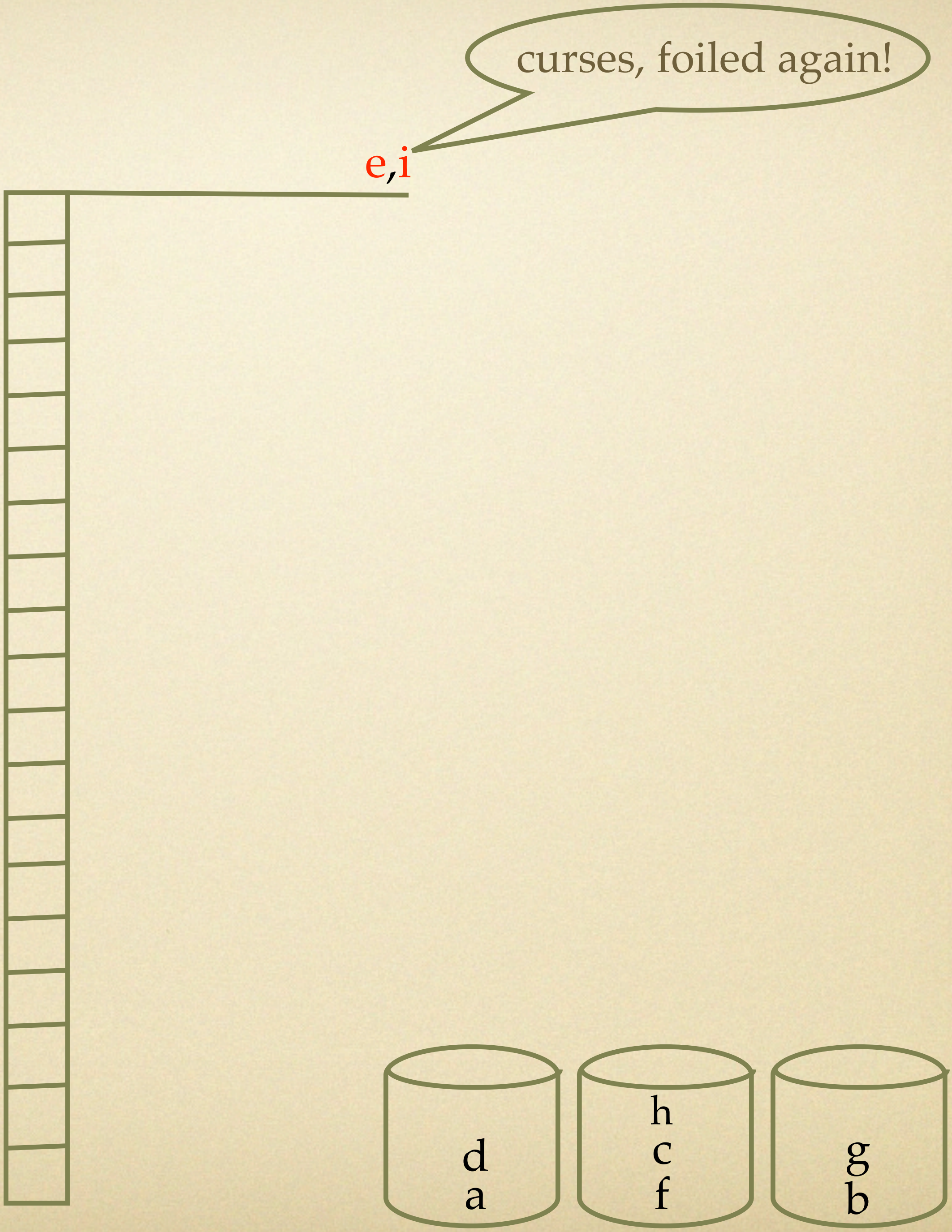




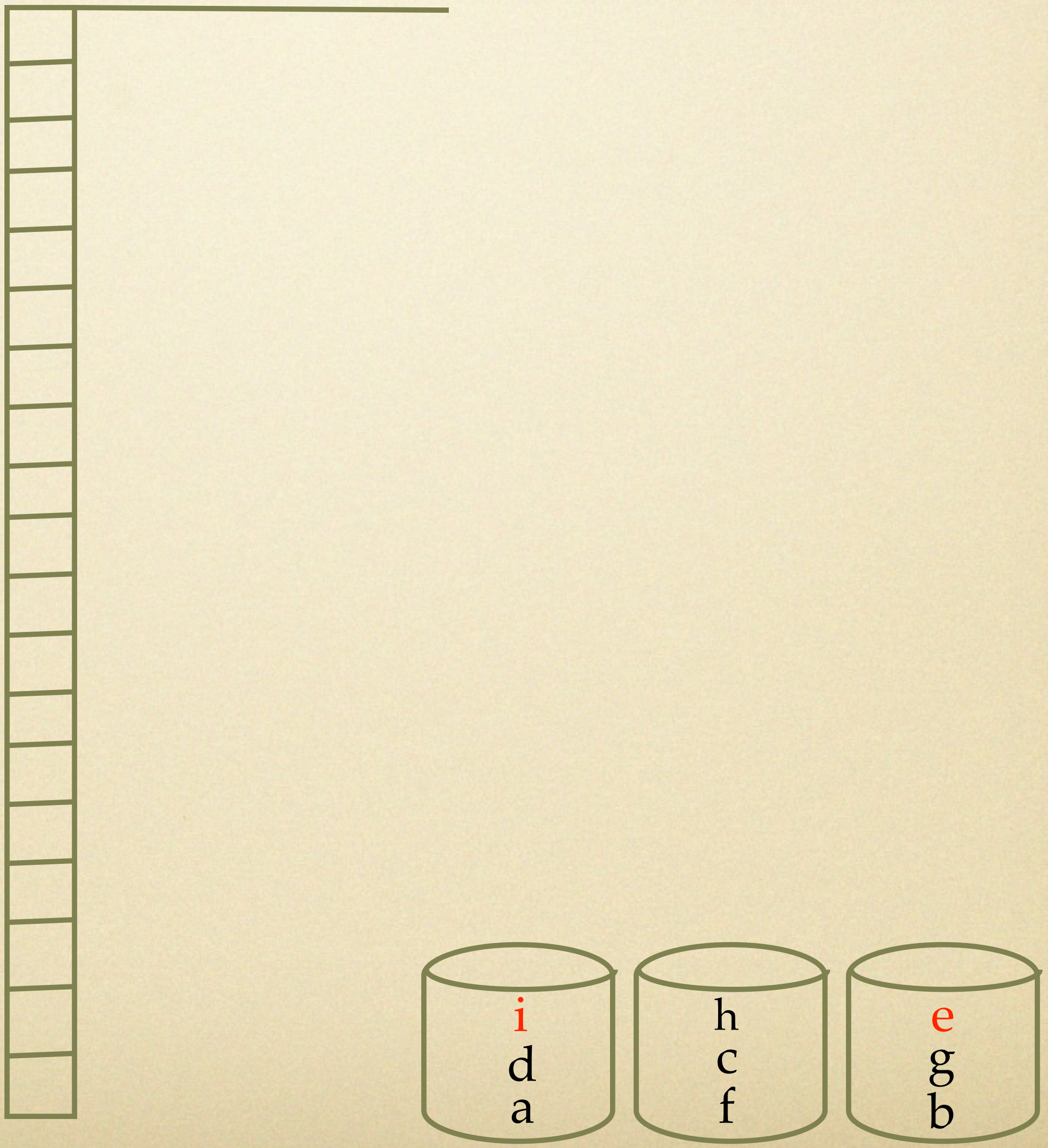
With O(n/log n) bins, whp, each bin has about same # of good procs

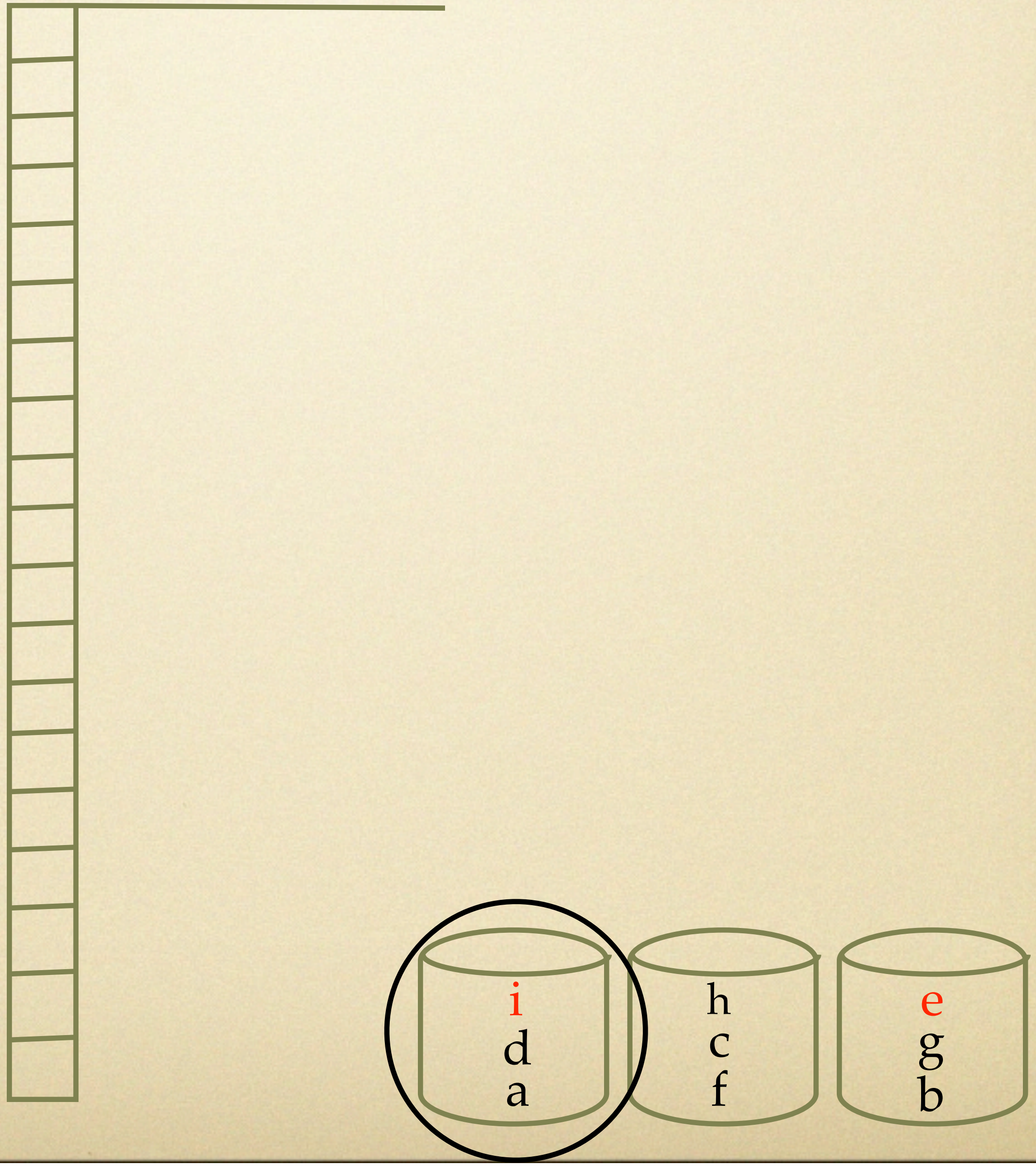
So fraction of bad in lightest bin will be not increase by much



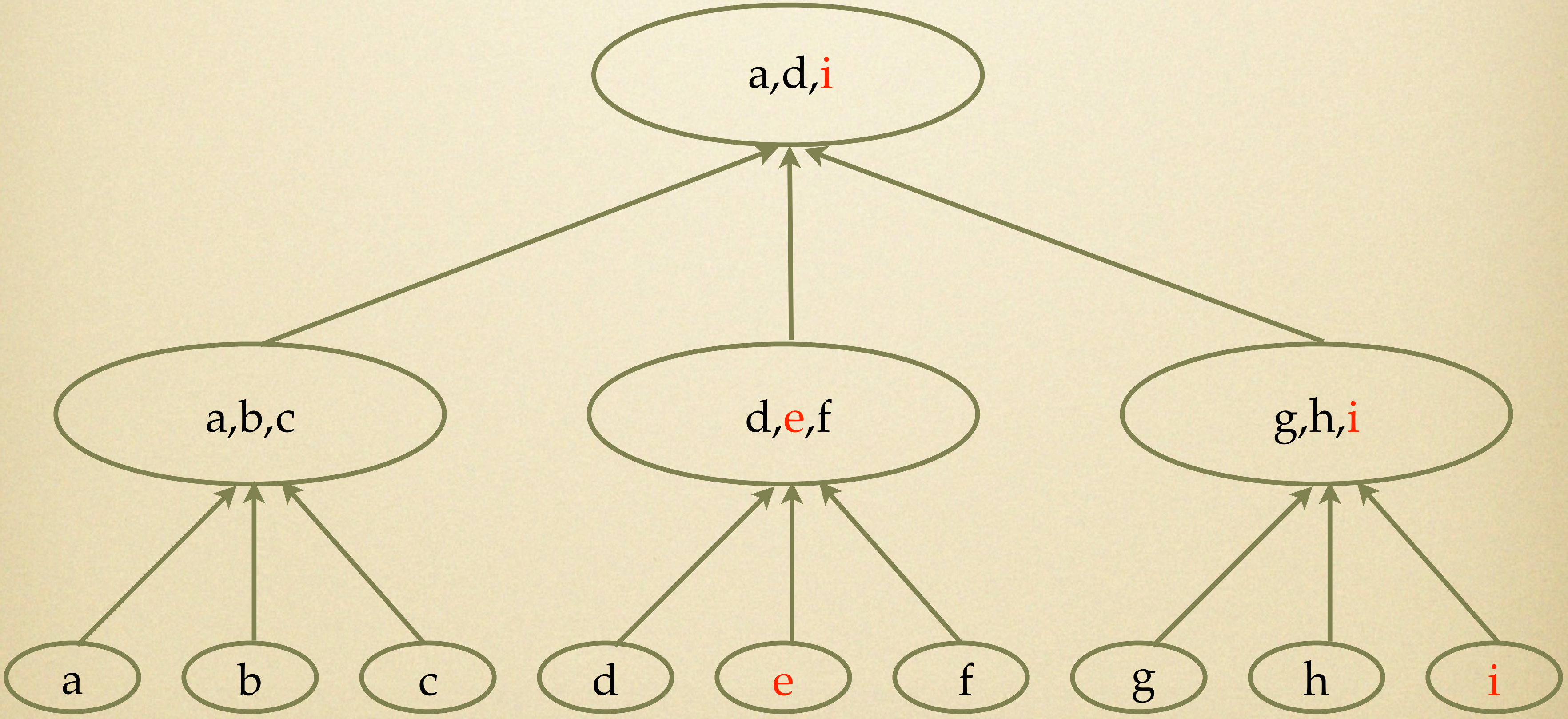


Monday, March 7, 2011


i
d
a
h
c
f
e
g
b


i
d
a
h
c
f
e
g
b

Problems:

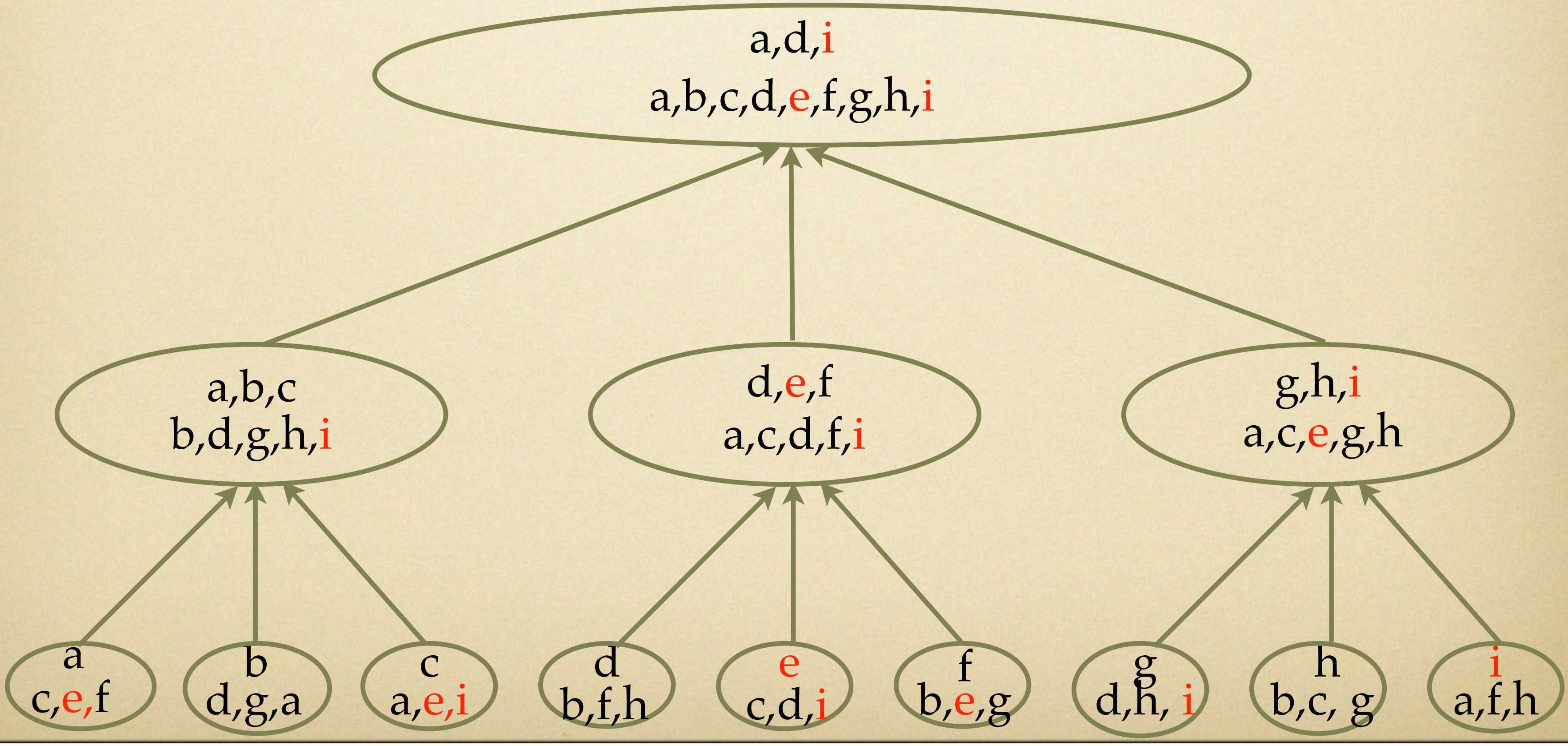


Problem 1: Bad procs may be inconsistent in bin choice

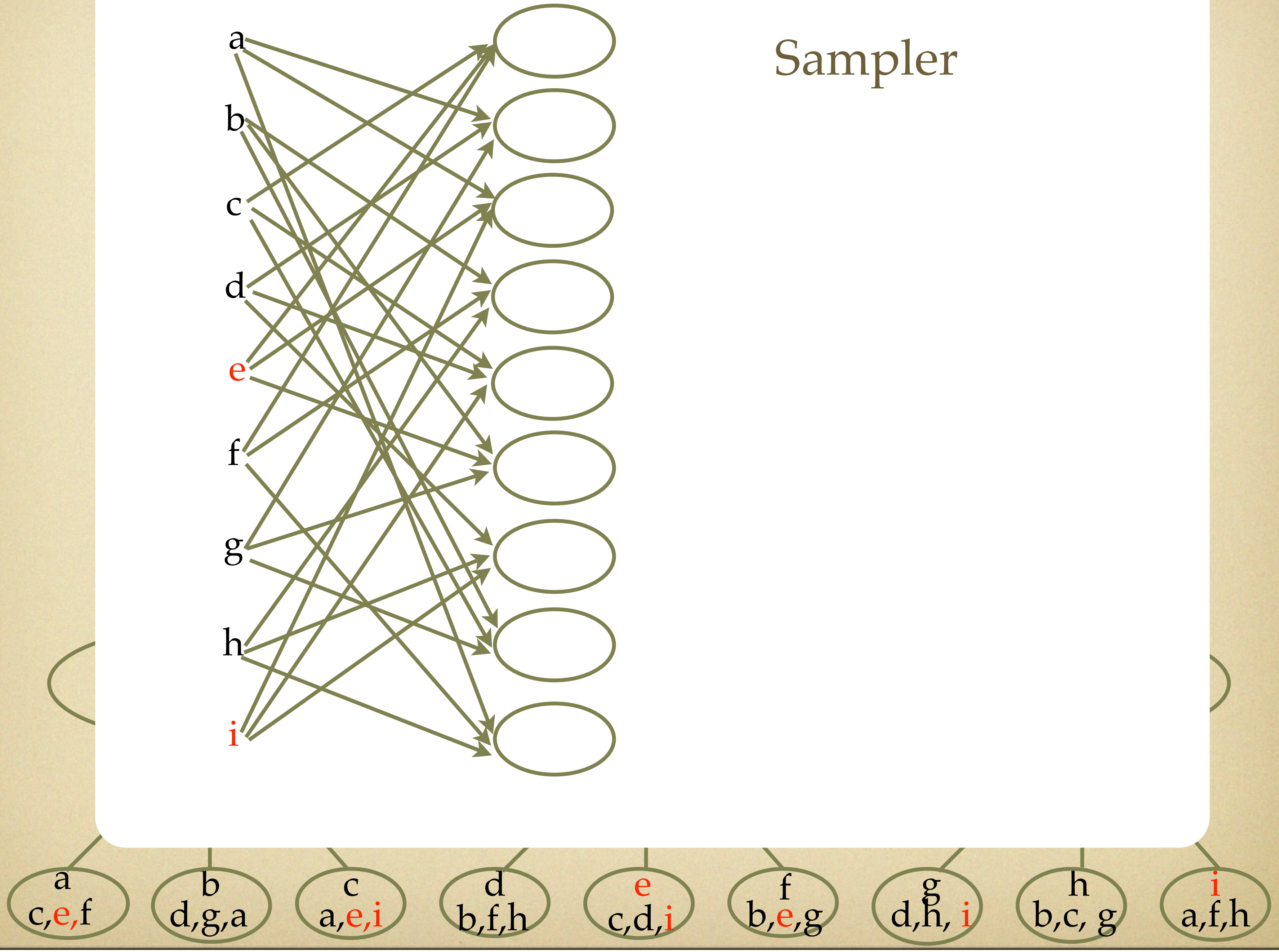
Solution:

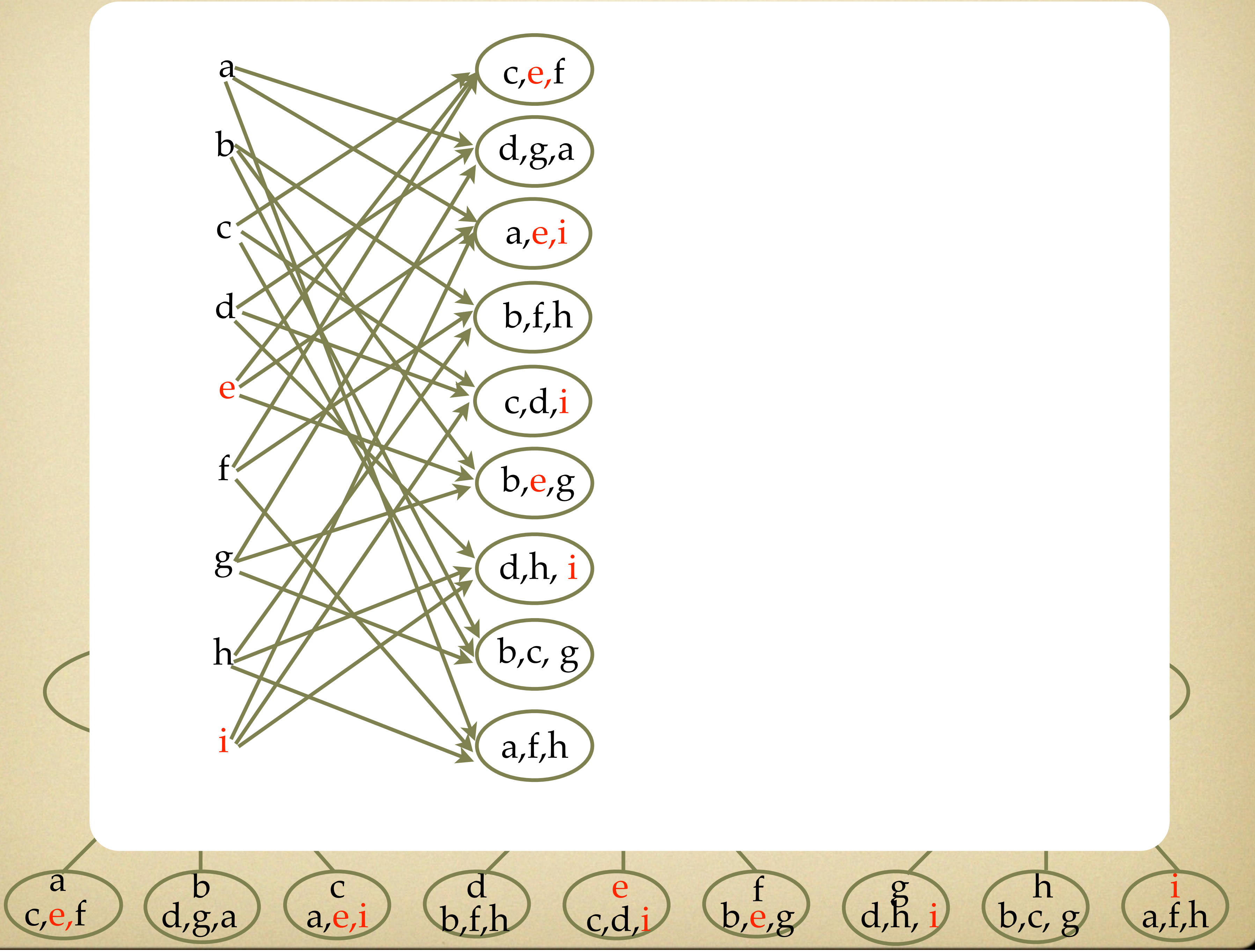
- Set of “enforcers” at each supernode who will run the election
- Higher supernodes have more enforcers
- Samplers map between procs and enforcer sets

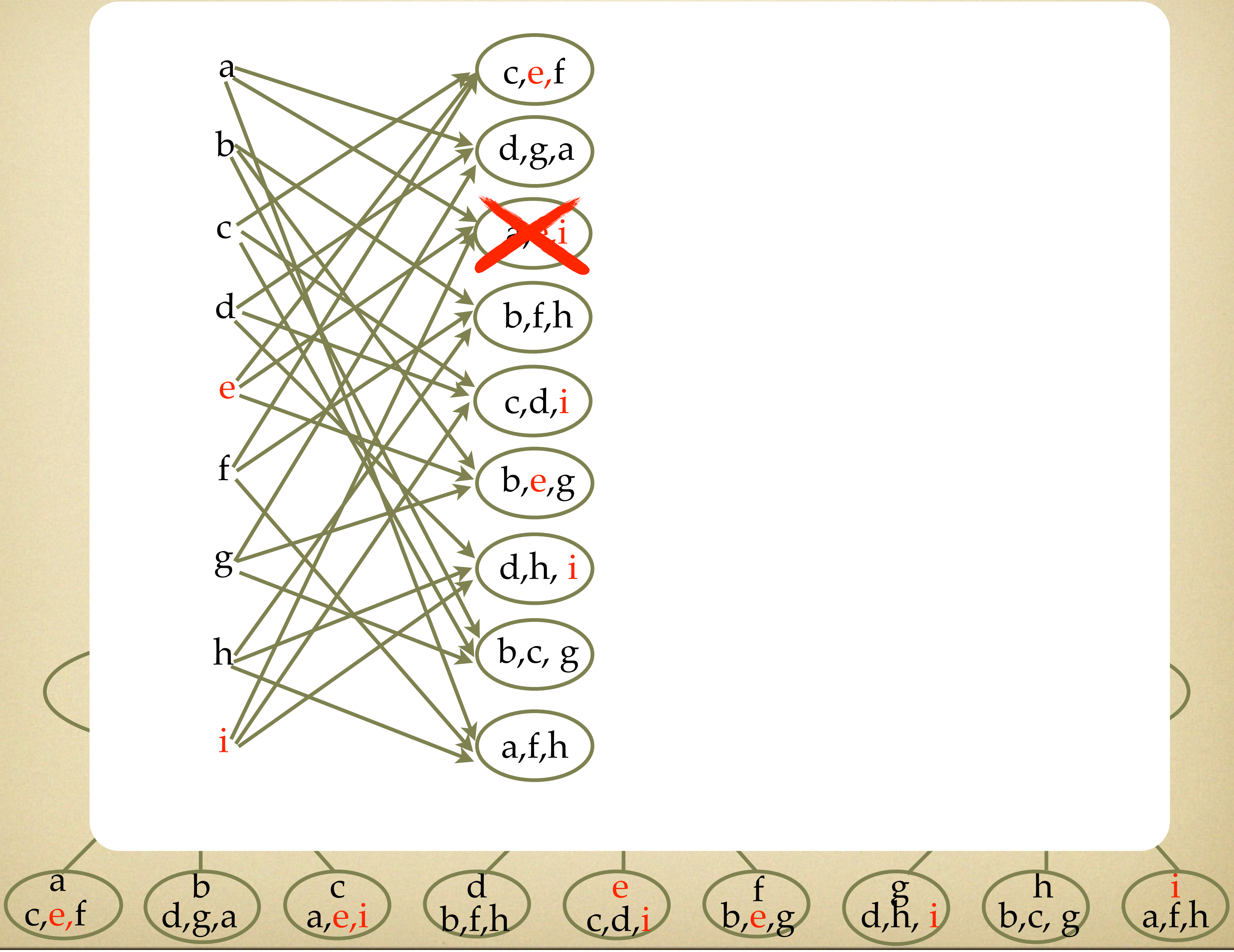
Enforcers

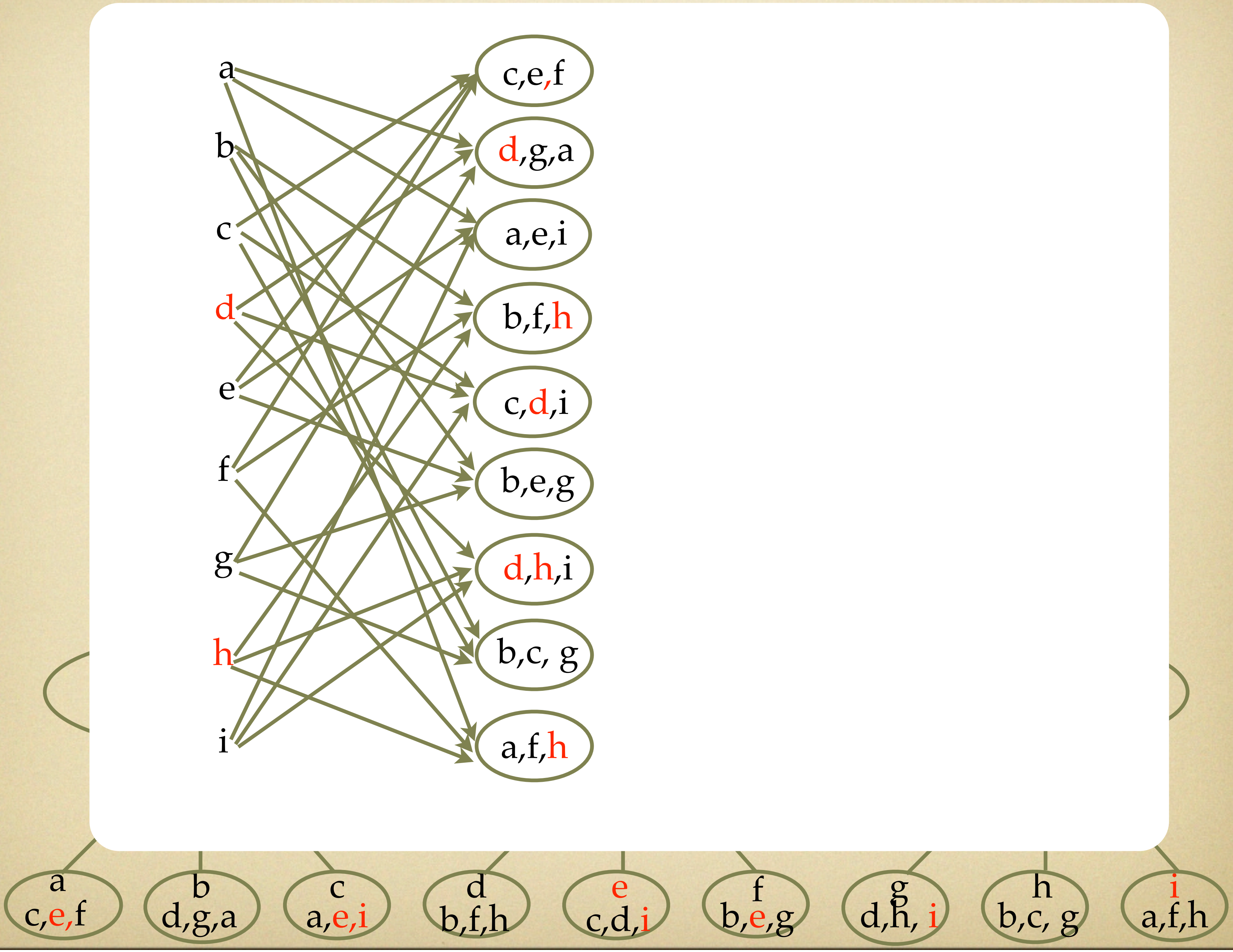

a,d,i
a,b,c,d,e,f,g,h,i
a,b,c
b,d,g,h,i
d,e,f
a,c,d,f,i
g,h,i
a,c,e,g,h
a
c,e,f
b
d,g,a
c
a,e,i
d
b,f,h
e
c,d,i
f
b,e,g
g
d,h, i
h
b,c, g
i
a,f,h

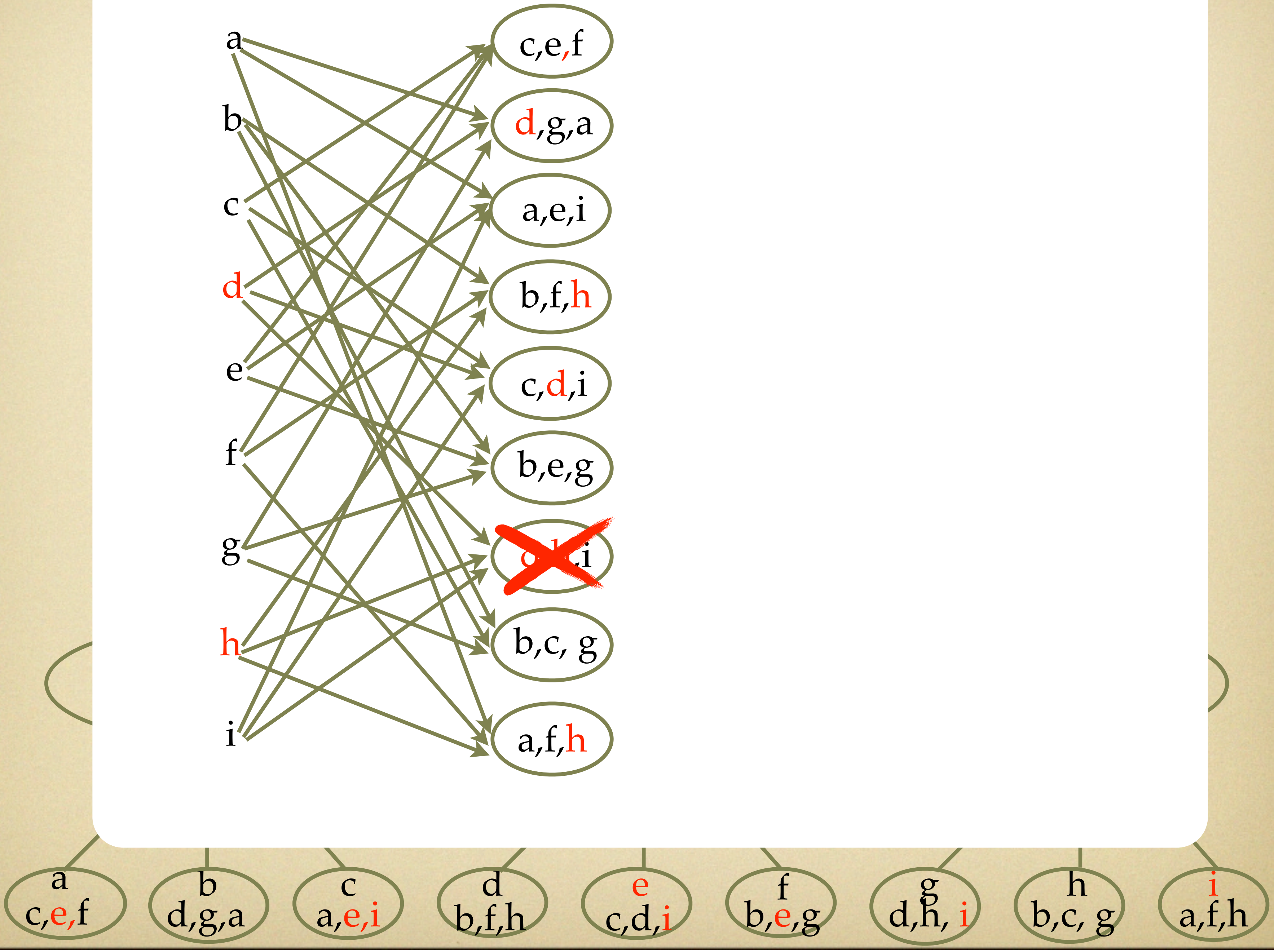
Sampler

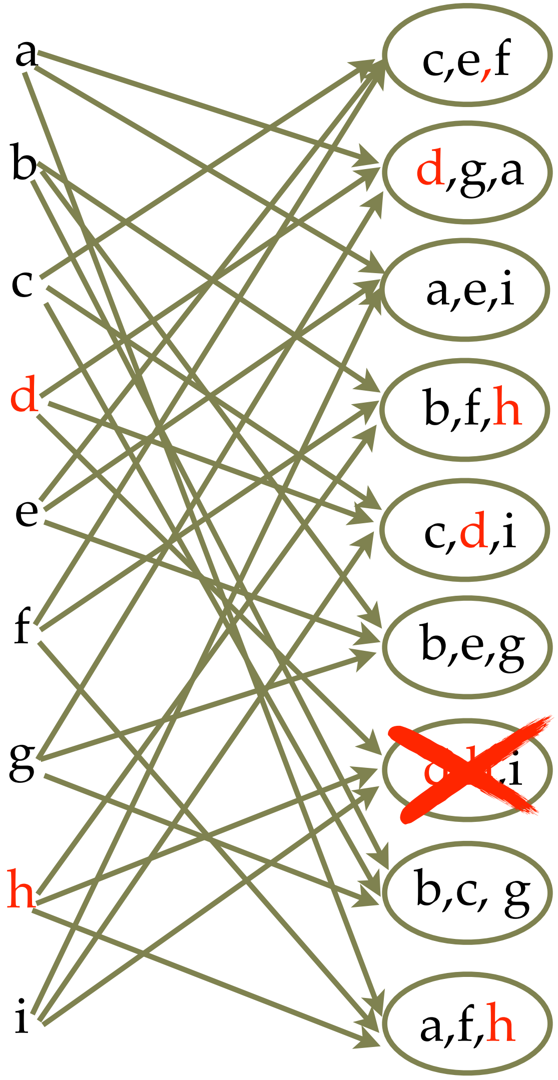



a
b
c
d
e
f
g
h
i
c,e,f
d,g,a
a,e,i
b,f,h
c,d,i
b,e,g
d,h, i
b,c, g
a,f,h
a
c,e,f
b
d,g,a
c
a,e,i
d
b,f,h
e
c,d,i
f
b,e,g
g
d,h, i
h
b,c, g
i
a,f,h


a
b
c
d
e
f
g
h
i
c,e,f
d,g,a
~~a,e,i~~
b,f,h
c,d,i
b,e,g
d,h, i
b,c, g
a,f,h
a c,e,f
b d,g,a
c a,e,i
d b,f,h
e c,d,i
f b,e,g
g d,h, i
h b,c, g
i a,f,h


a
b
c
d
e
f
g
h
i
c,e,f
d,g,a
a,e,i
b,f,h
c,d,i
b,e,g
d,h,i
b,c, g
a,f,h
a
c,e,f
b
d,g,a
c
a,e,i
d
b,f,h
e
c,d,i
f
b,e,g
g
d,h, i
h
b,c, g
i
a,f,h


a
b
c
d
e
f
g
h
i
c,e,f
d,g,a
a,e,i
b,f,h
c,d,i
b,e,g
~~d,h,i~~
b,c, g
a,f,h
a
c,e,f
b
d,g,a
c
a,e,i
d
b,f,h
e
c,d,i
f
b,e,g
g
d,h, i
h
b,c, g
i
a,f,h

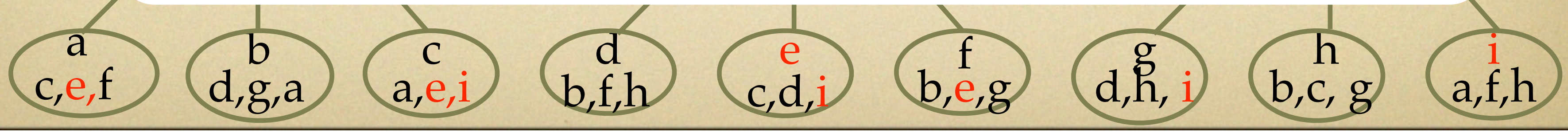


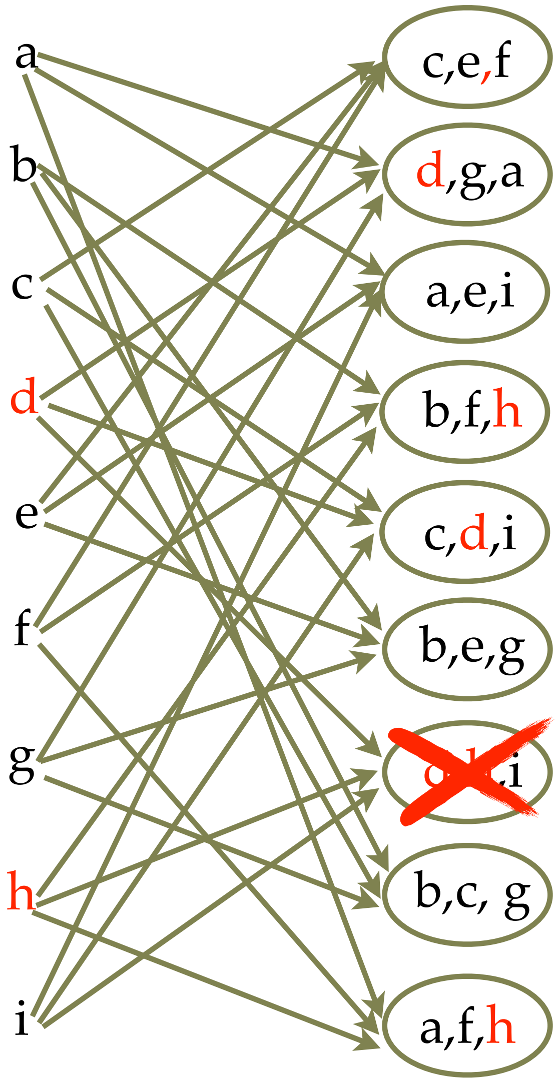
A **sampler** ensures that $\geq 1 - \delta$ fraction on right has a fraction of bad neighbors $\leq t/n + \theta$

and the degree of the graph is just:

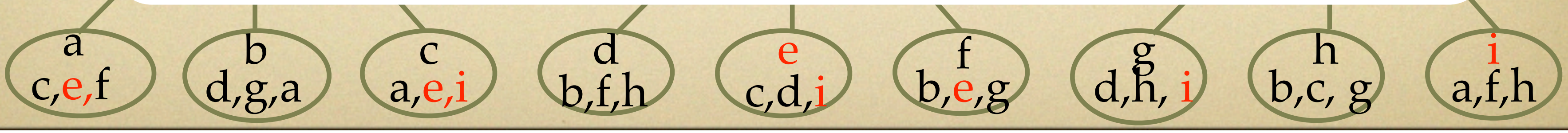
$$\frac{2-\delta}{\theta^2 \delta \cdot 2 \log_2 e}$$

No matter which subset is bad!

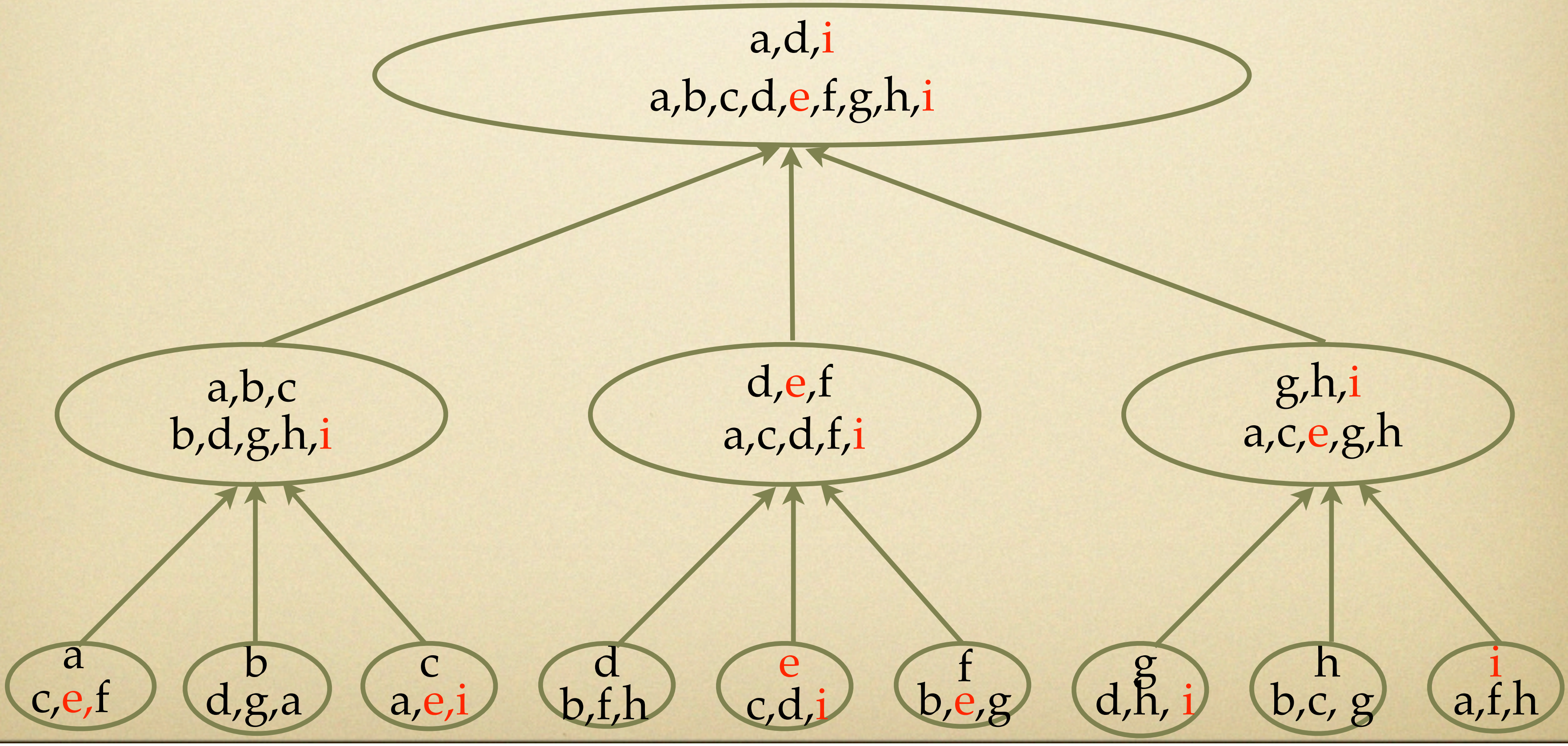




Almost all enforcer sets have >= 2/3 fraction of good procs

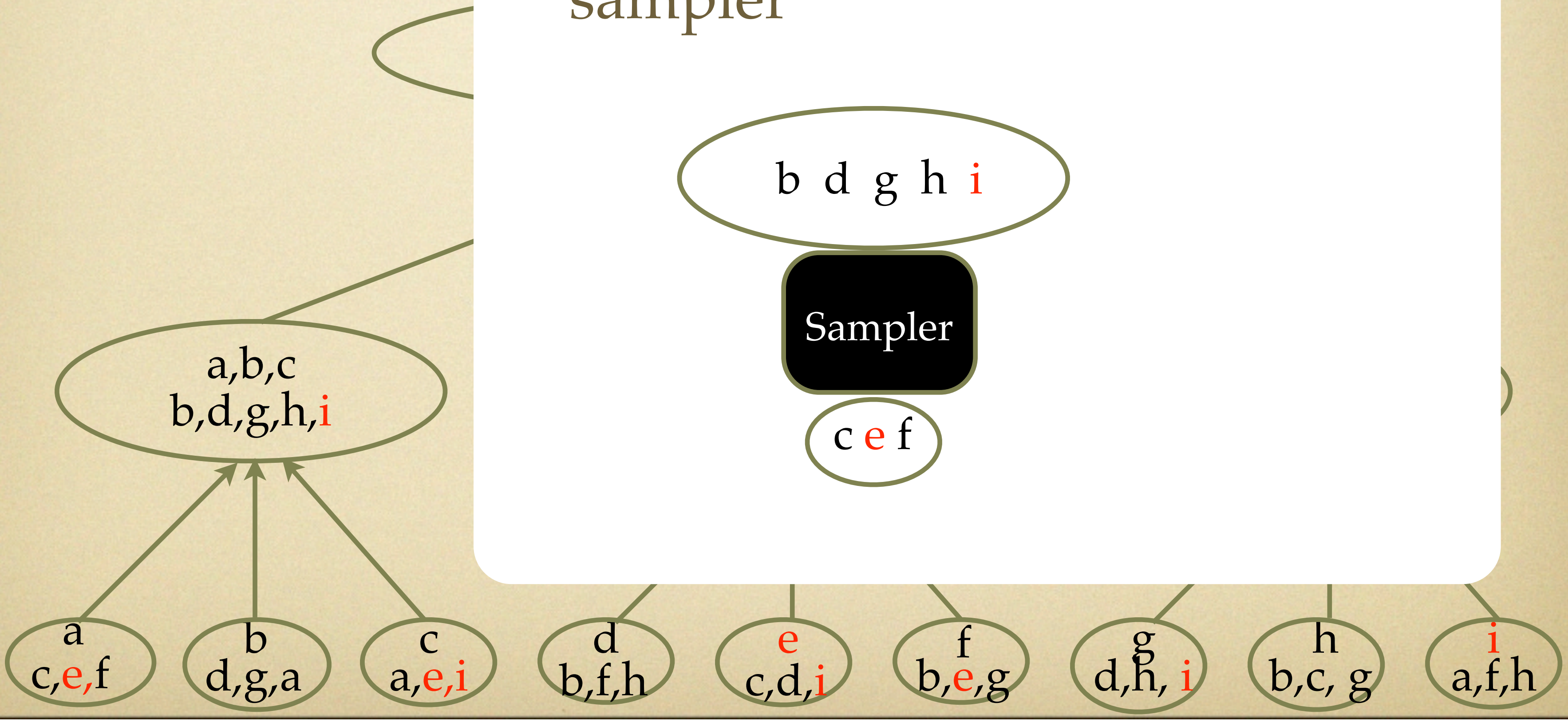


Enforcers

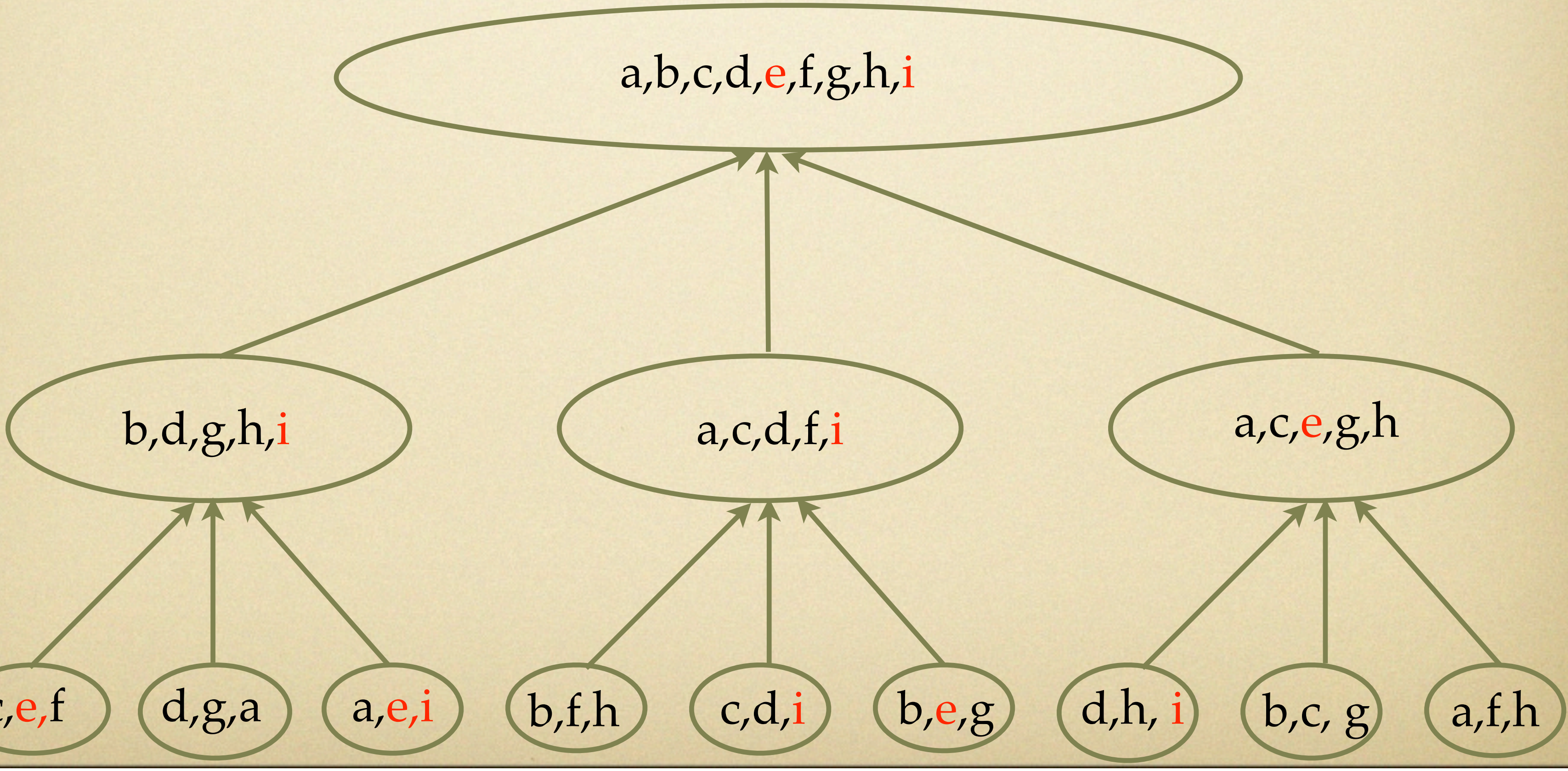

a,d,i
a,b,c,d,e,f,g,h,i
a,b,c
b,d,g,h,i
d,e,f
a,c,d,f,i
g,h,i
a,c,e,g,h
a
c,e,f
b
d,g,a
c
a,e,i
d
b,f,h
e
c,d,i
f
b,e,g
g
d,h, i
h
b,c, g
i
a,f,h

Enforcers

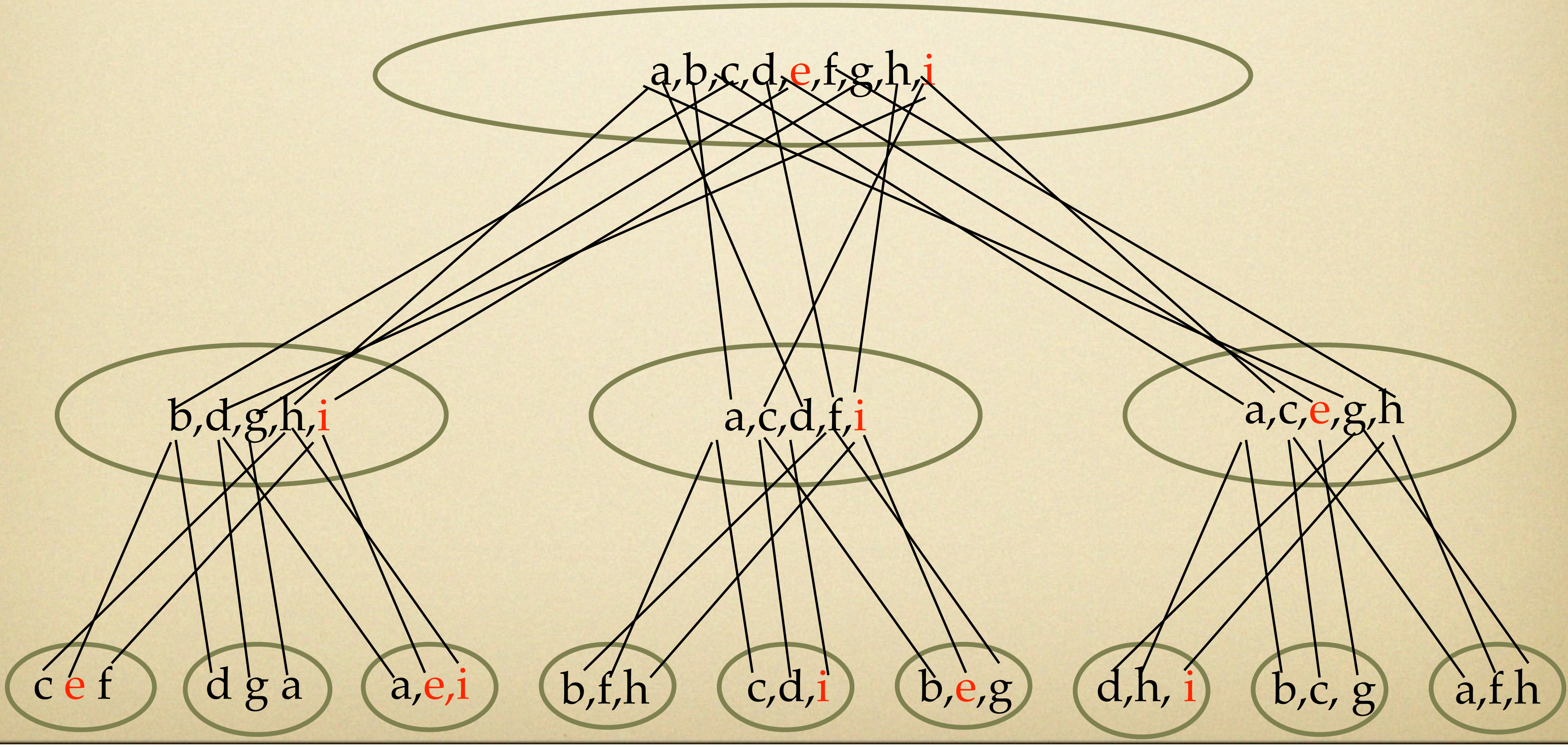
Connections between enforcers in parent and children supernodes also given by a sampler



Samplers


a,b,c,d,e,f,g,h,i
b,d,g,h,i
a,c,d,f,i
a,c,e,g,h
c,e,f
d,g,a
a,e,i
b,f,h
c,d,i
b,e,g
d,h, i
b,c, g
a,f,h

Samplers



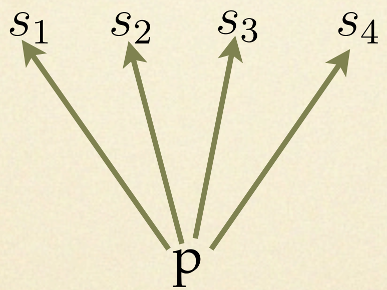
Monday, March 7, 2011

Problem 2: Adaptive adversary can wait and take over all procs at the top supernode

Solution:

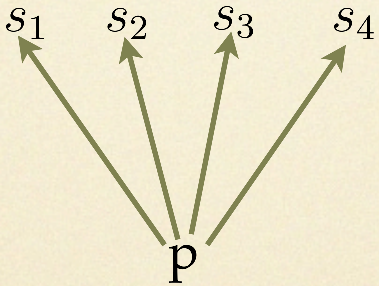
- Each proc p generates array A_p of random numbers and **secret shares** it with its leaf node
- Numbers are revealed as needed to elect which arrays will be passed on to parent node
- As winning array moves up, secret shares are split up among more and more procs on higher levels

Secret Sharing



- p's secret is f(0), where f is a polynomial of degree 3
- The shares are f() evaluated at different points

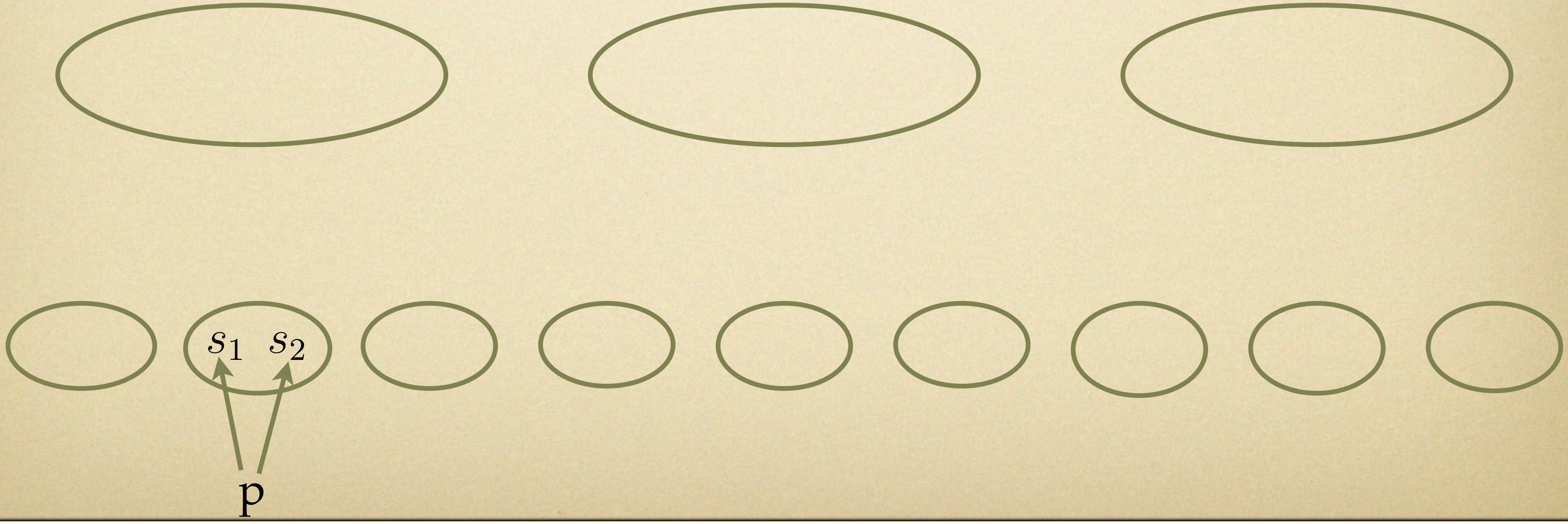
Secret Sharing



- p's secret is f(0), where f is a polynomial of degree 3
- The shares are f() evaluated at different points
- We use secret sharing schemes where just a 2/3 fraction of the shares are needed to reconstruct

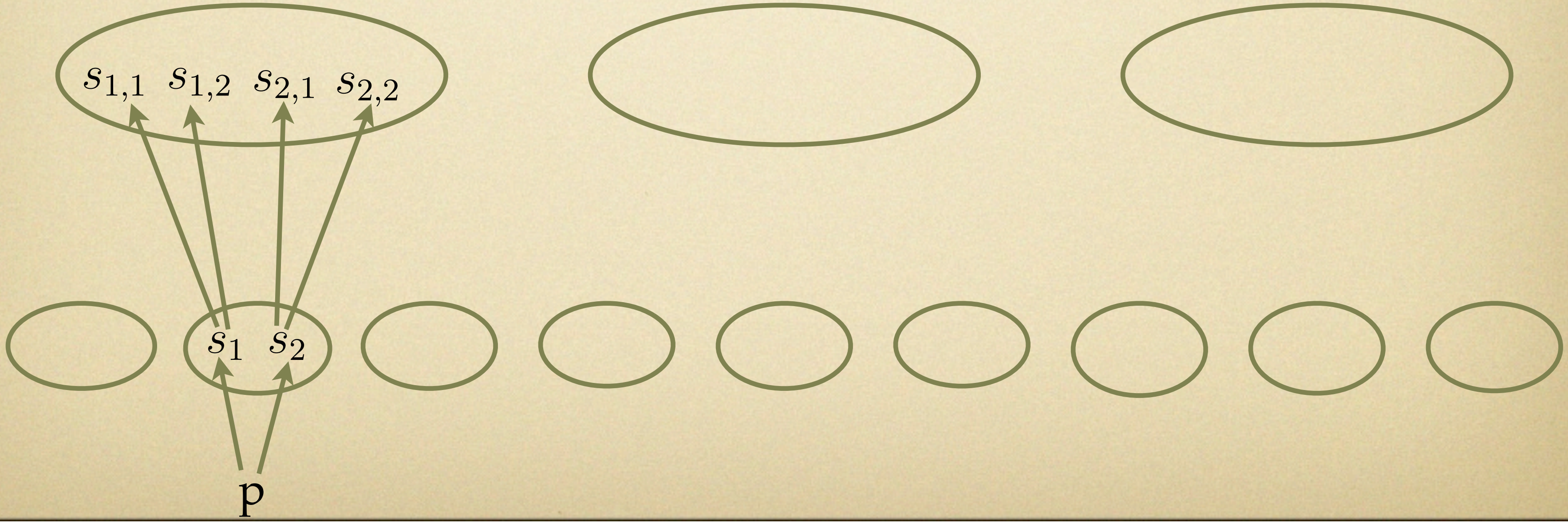
Splitting Secrets

As winning array moves up, secret shares are split up among more and more procs on higher levels and erased from children



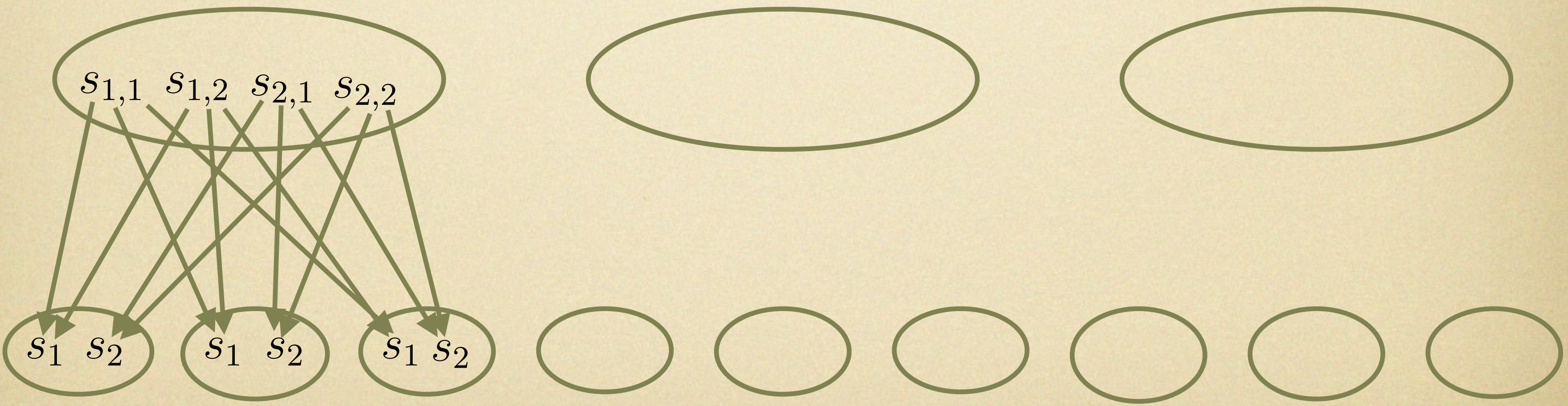
Splitting Secrets

As winning array moves up, secret shares are split up among more and more procs on higher levels and erased from children



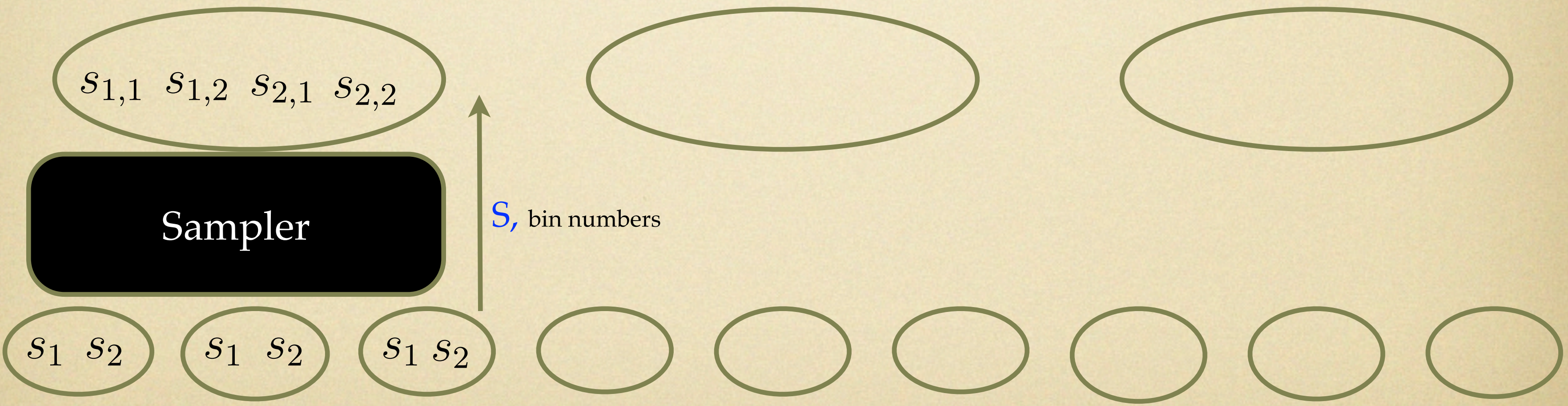
Revealing Secrets

- Secrets revealed as needed: by reversing communication downward, reassembling shares at subtrees and leaves
- Thus, adversary can't prevent secret from being exposed by blocking a single path

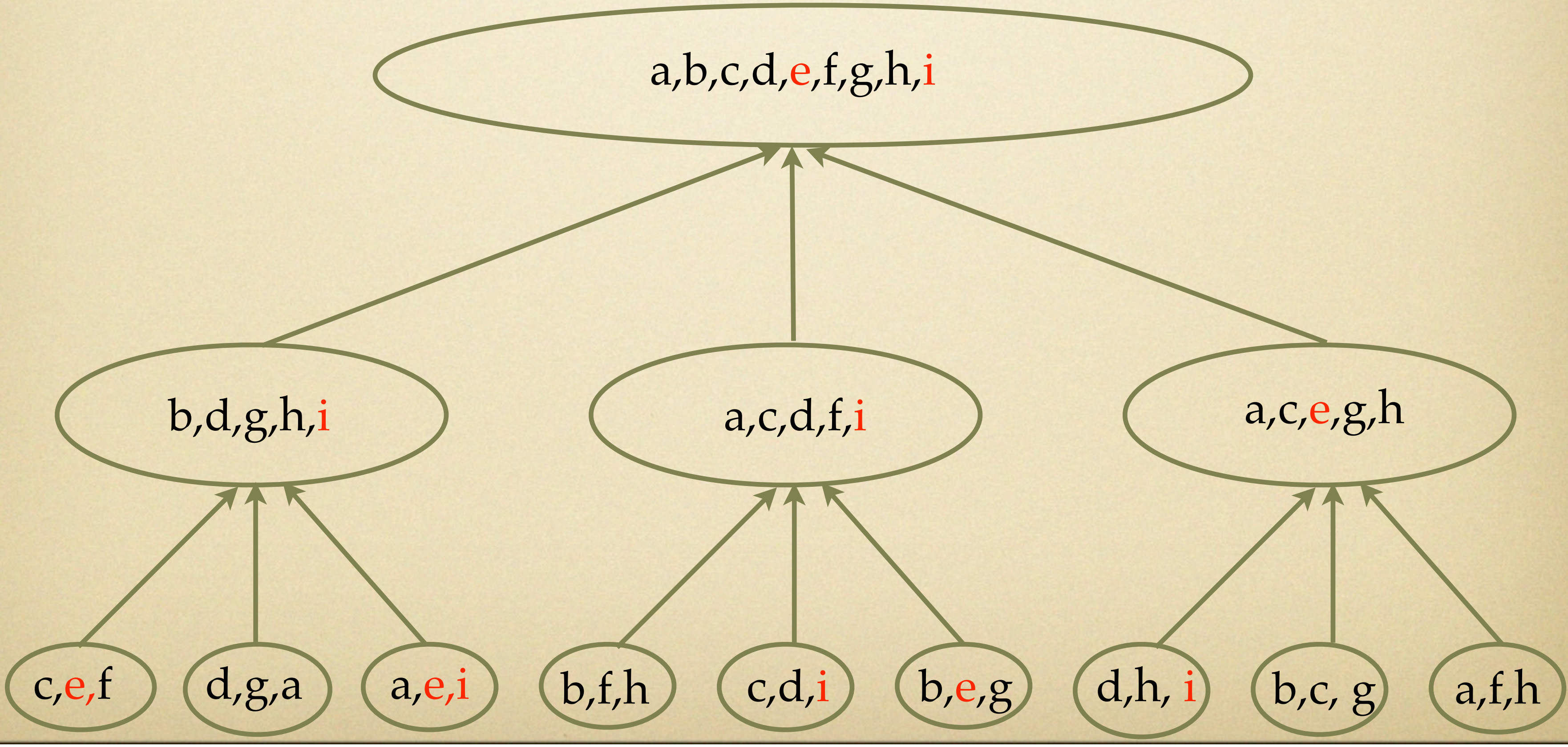


Revealing Secrets

- Leaves are sampled deterministically by procs in subtree root in order to learn the secret value

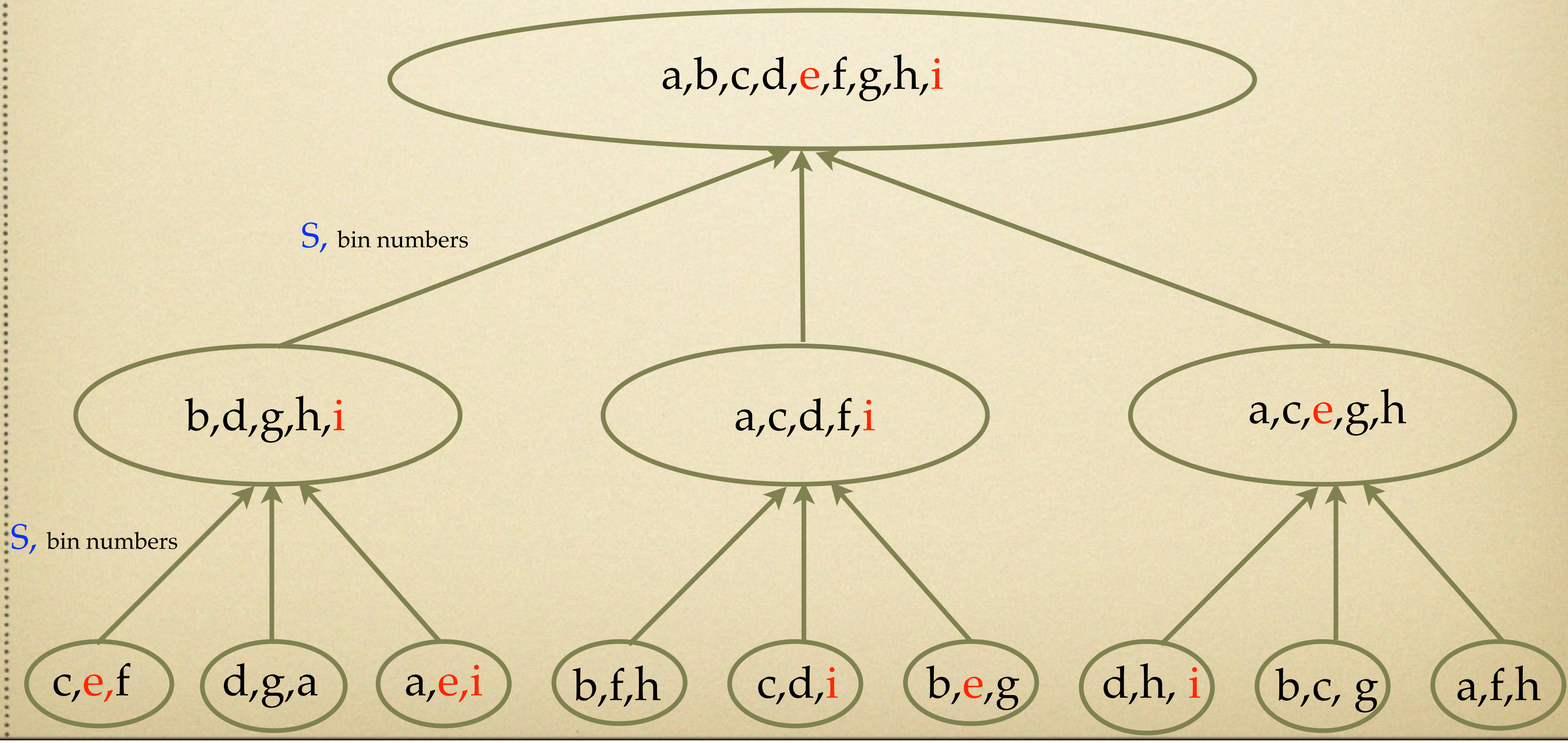


Implementing S



Implementing S

S and bin numbers are given by winning arrays of children supernodes through secret sharing



Monday, March 7, 2011

Algorithm Outline

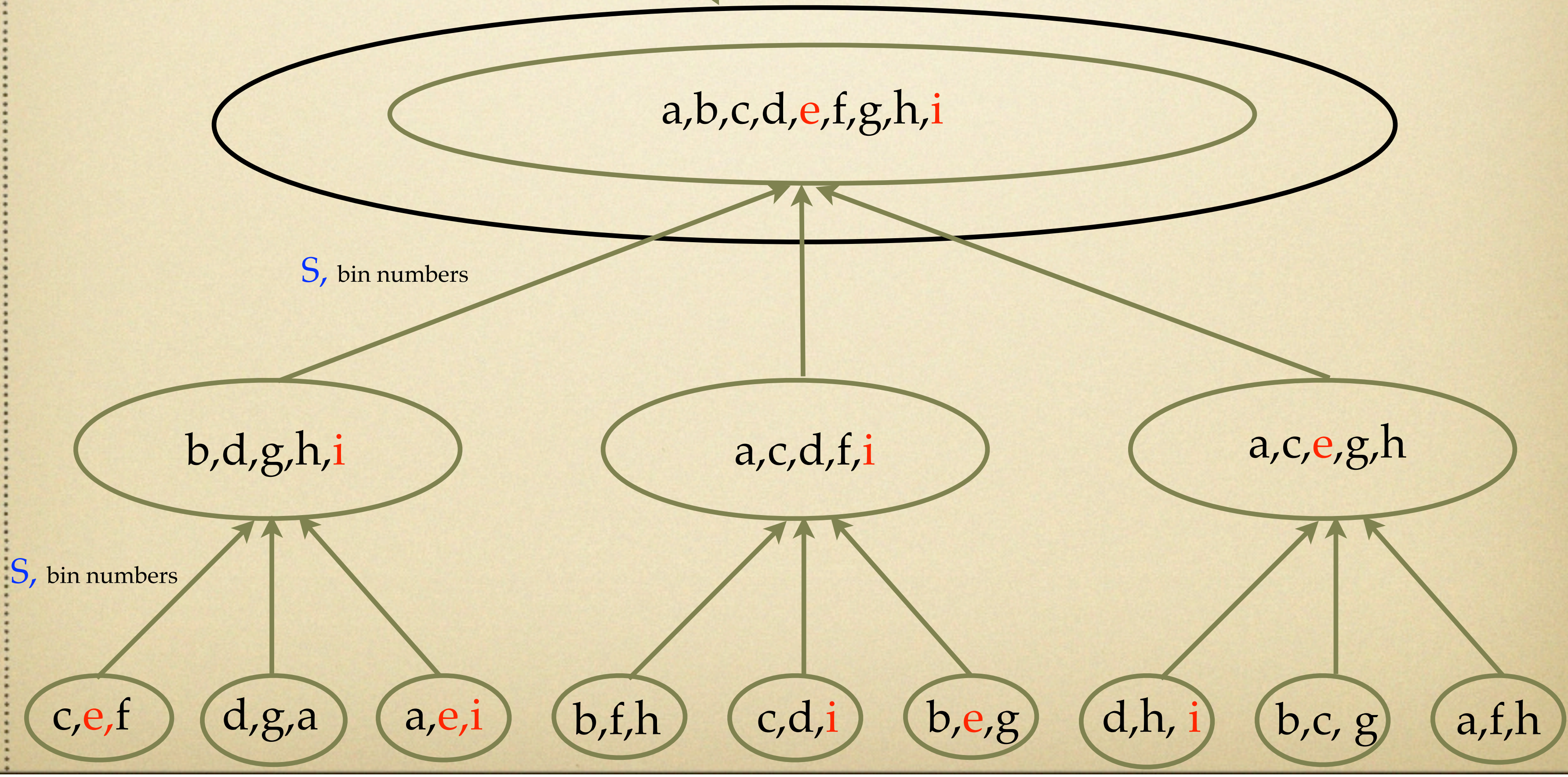
I: Using S to get a.e. BA ✓

II: Using S to go from a.e. BA to BA ✓

III: Implementing S ✓

Recap

Once S is known here, aeBA can be performed among the enforcers at top supernode (i.e. all procs)



Models where we can implement S

- Secret channels, adaptive adversary

Breaking the O(n^2) Bit Barrier: Scalable Byzantine agreement with an Adaptive Adversary" by Valerie King and Jared Saia, *Published in Principles of Distributed Computing (PODC)*, 2010. **Best Paper award.**

- Open channels, nonadaptive adversary

"Fast, scalable Byzantine agreement in the full information model with a Nonadaptive adversary" by Valerie King and Jared Saia *International Symposium on Distributed Computing (DISC)*, 2009.

- Asynchronous, nonadaptive adversary

"Fast Asynchronous Byzantine Agreement and Leader Election with Full Information" by Bruce Kapron, David Kempe, Valerie King, Jared Saia and Vishal Sanwalani. *In Symposium on Discrete Algorithms (SODA), 2008* (pdf) **Invited submission to "Transactions on Algorithms" best papers of SODA 2008**.

Uses of S

- Scalable BA
- Scalable Leader election, Global Coin, etc. (non-adaptive adversary)
- Can specify a set of n small (O(log n) size) and balanced (no proc in more than O(log n)) quorums which are all good w.h.p

``Load balanced Scalable Byzantine Agreement through Quorum Building, with Full Information'' by Valerie King, Steve Lonargan, Jared Saia and Amitabh Trehan. *In the International Conference on Distributed Computing and Networking(ICDCN)*, 2010.

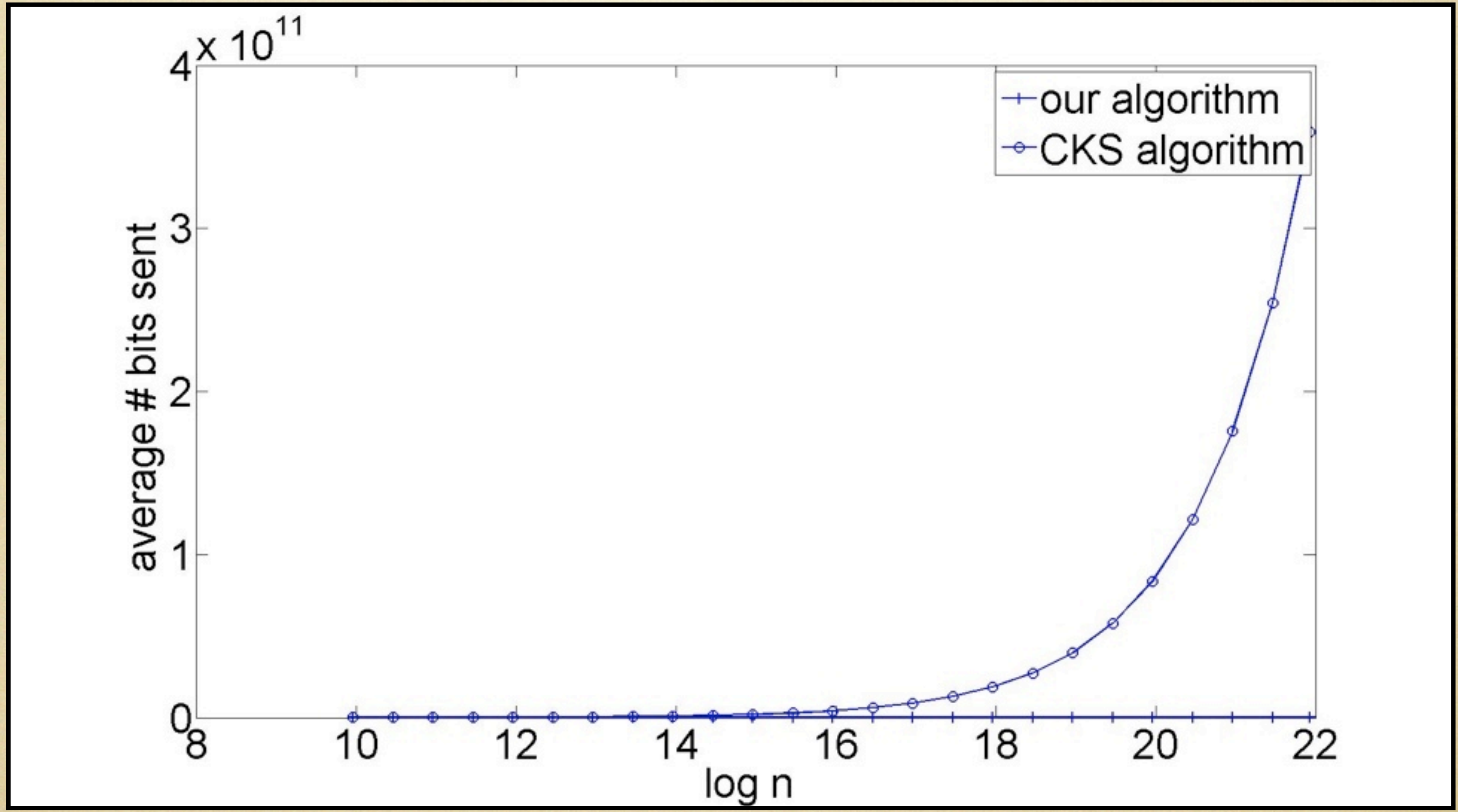
Uses of S

- Scalable BA
- Scalable Leader election, Global Coin, etc. (non-adaptive adversary)
- Can specify a set of n small (O(log n) size) and balanced (no proc in more than O(log n)) quorums which are all good w.h.p

``Load balanced Scalable Byzantine Agreement through Quorum Building, with Full Information'' by Valerie King, Steve Lonargan, Jared Saia and Amitabh Trehan. *In the International Conference on Distributed Computing and Networking(ICDCN)*, 2010.

Robust Multiparty Computation

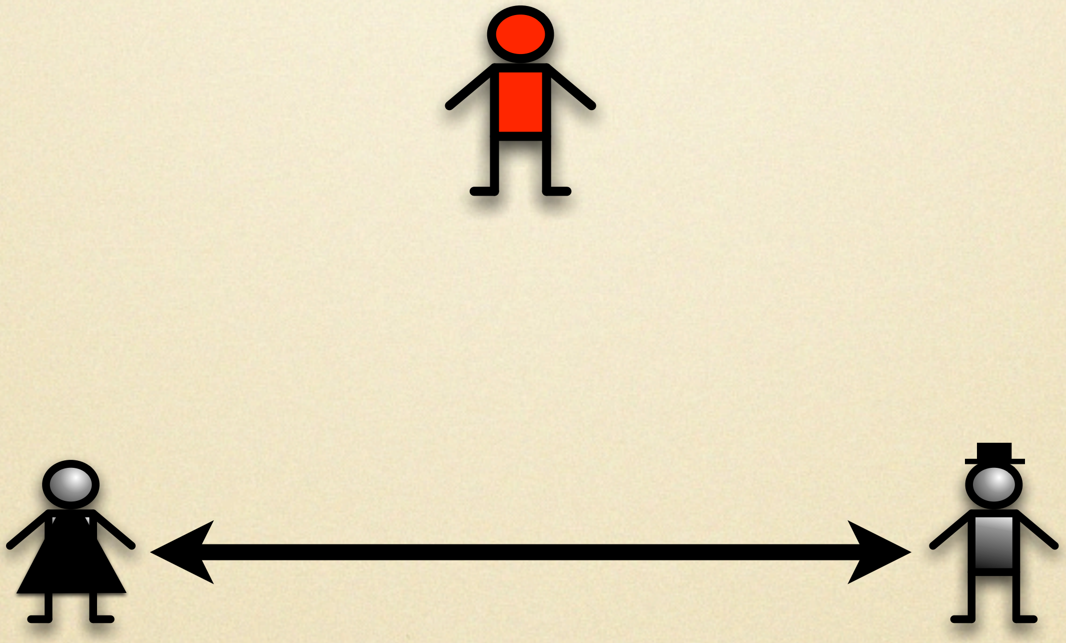
Simulations

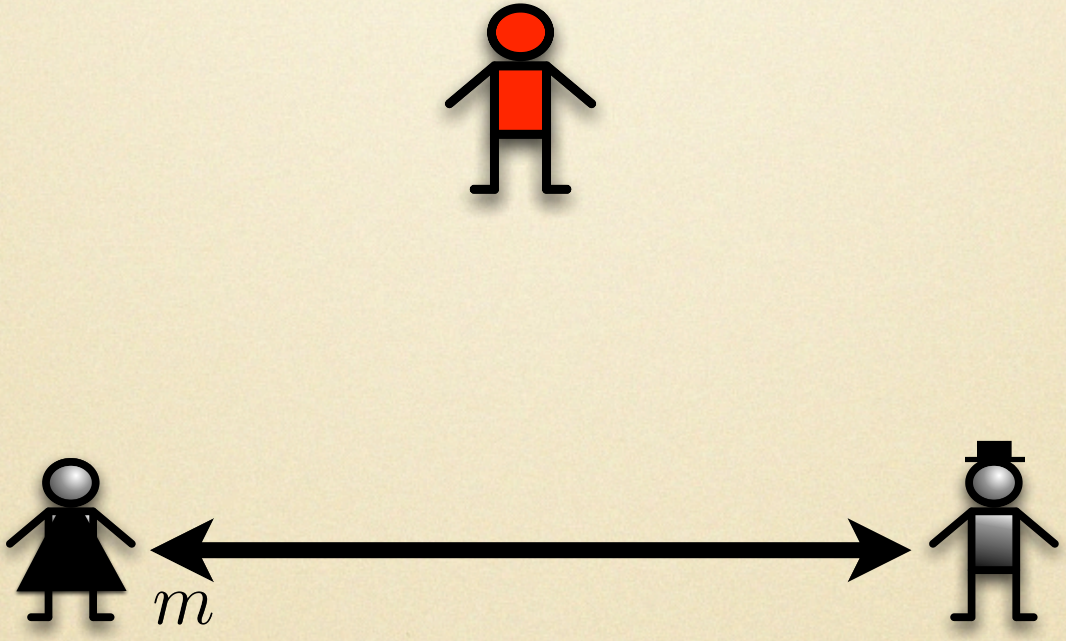


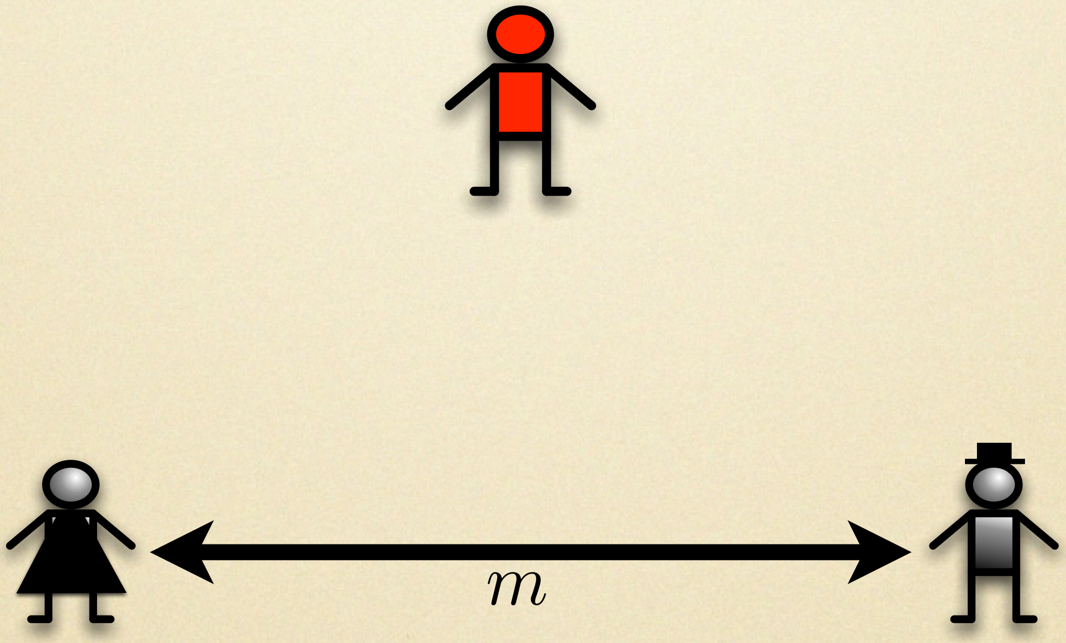
Rest of Talk: Sketch of Other Results

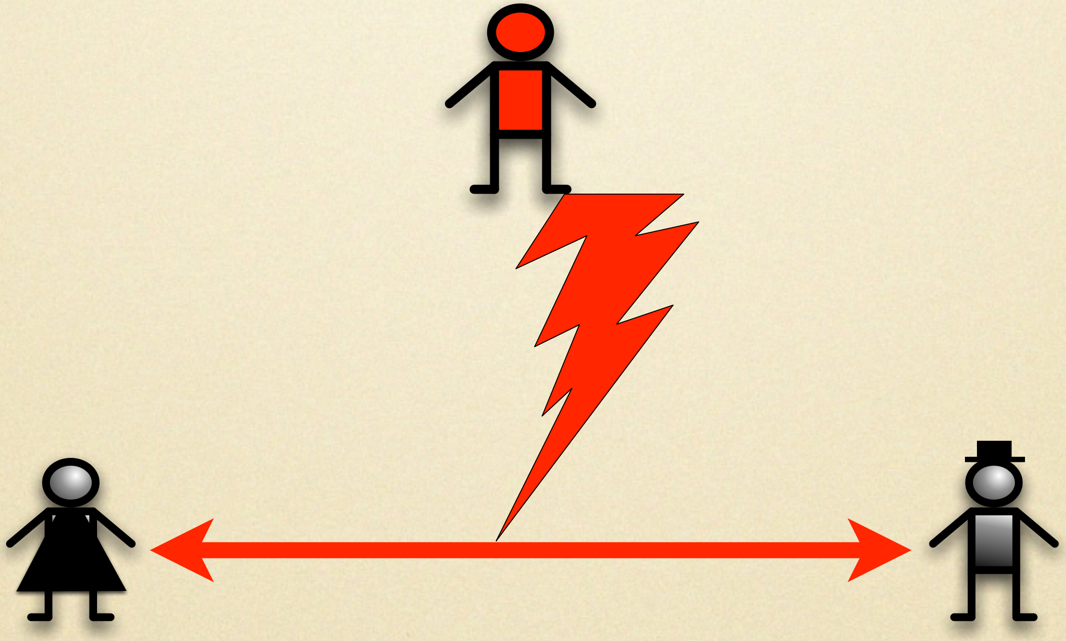
1) Conflict on a Communication Channel

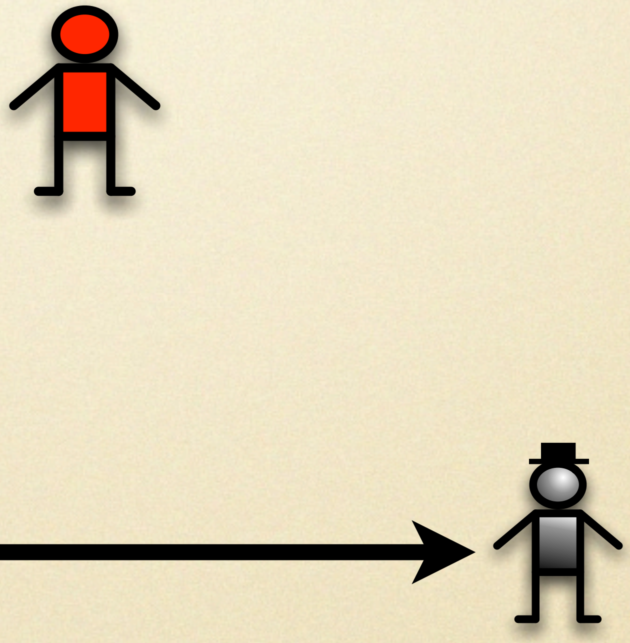
2) Self-Healing Networks

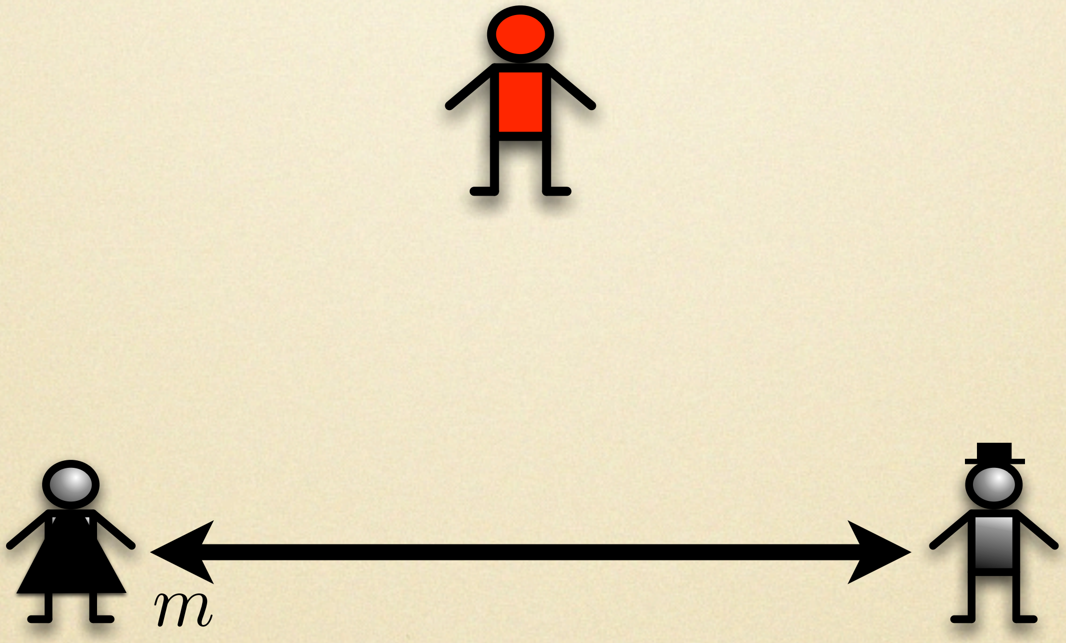


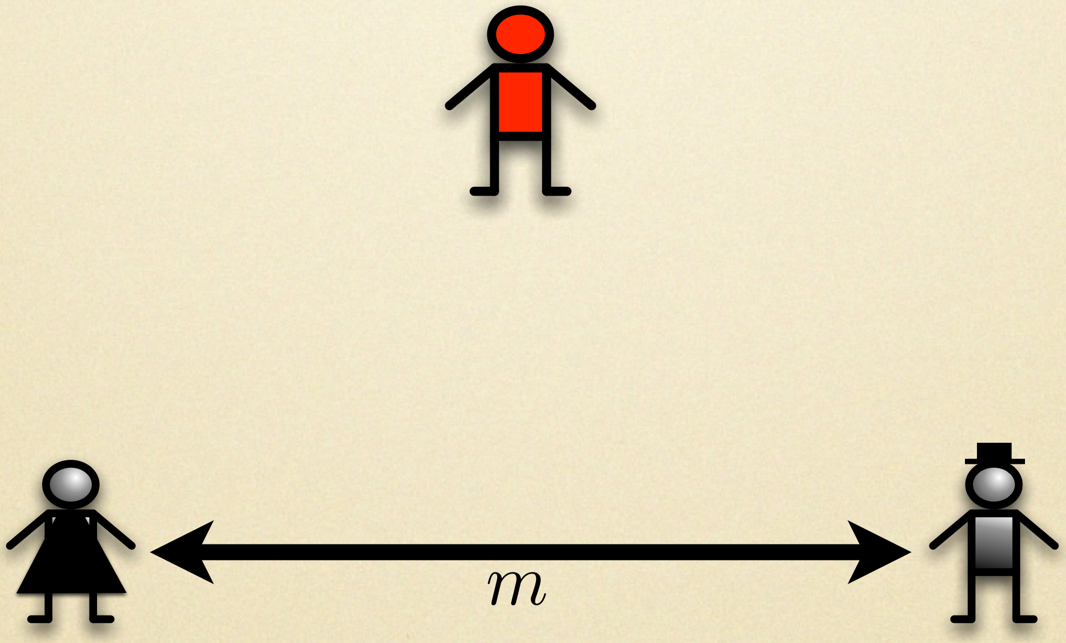

m

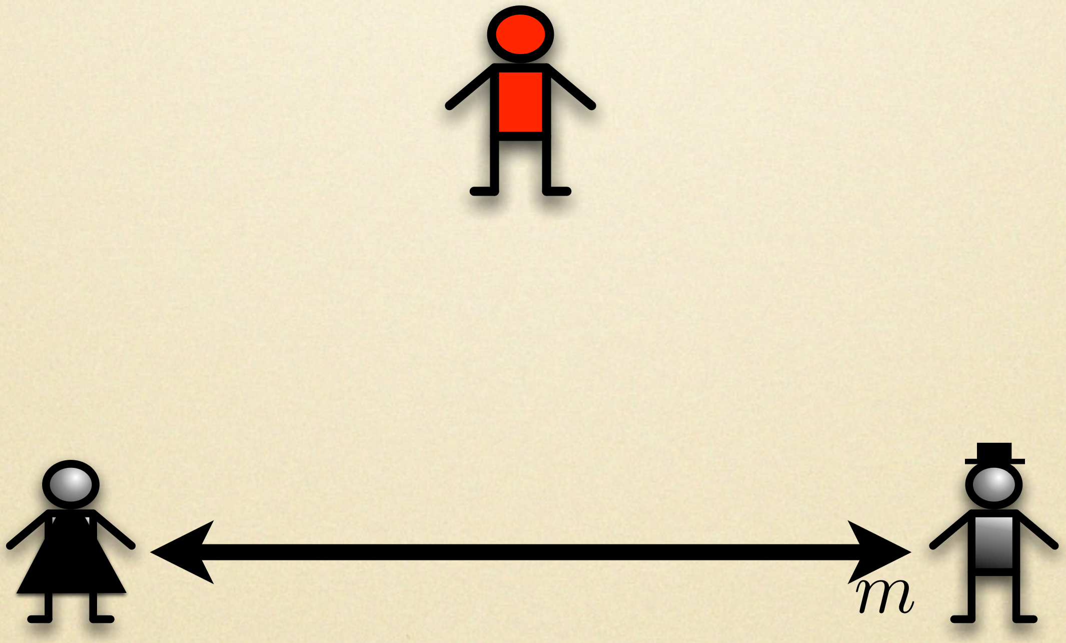

m

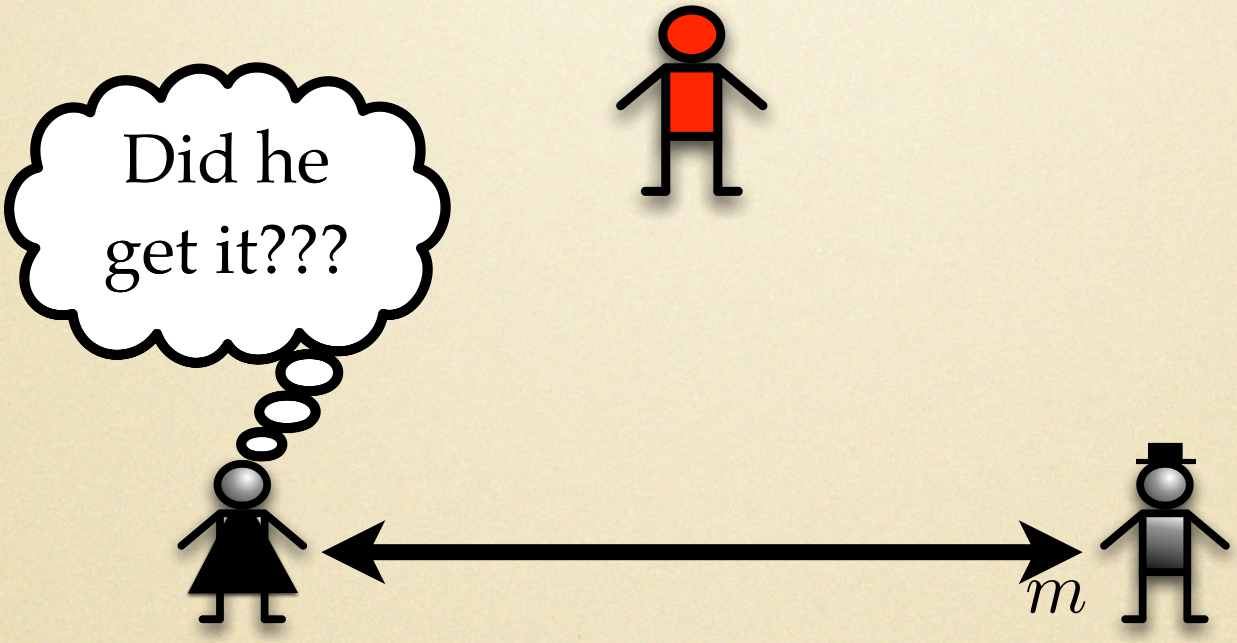





m


m


m


Did he get it???
m

3 Player Game

- Alice wants to send a message to Bob
- Adversary wants to block the message
- There is a communication channel between Alice and Bob, but Adv. can block it

Costs

- Costs $S to send on channel
- Costs $L to listen on channel
- Costs $J to block channel
- Adv. spends $B

Costs - Sensors

Costs $S to send on channel	38mW
Costs $L to listen on channel	35mW
Costs $J to block channel	>1mW
Adv. spends $B	>5,000mW

Costs - Sensors

Costs \$S to send on channel	38mW
Costs \$L to listen on channel	35mW
Costs \$J to block channel	>1mW
Adv. spends \$B	>5,000mW

We assume S, L and J are O(1)

Key Assumptions

- If Alice or Bob listen on channel when Adv. jams it, they can detect a “collision”
- Adv. can successfully imitate Bob but not Alice

An Idea

- A round consists of n slots
- Alice sends w/ prob $c/\sqrt{n}$
- Bob listens w/ prob $c/\sqrt{n}$

An Idea

- A round consists of n slots
- Alice sends w/ prob $c/\sqrt{n}$
- Bob listens w/ prob $c/\sqrt{n}$

Assume Adv. blocks w/ prob 1/2.

Then prob. a given slot is one where Alice sends and there is no jam is $\frac{c}{2\sqrt{n}}$

An Idea

- A round consists of n slots
- Alice sends w/ prob $c/\sqrt{n}$
- Bob listens w/ prob $c/\sqrt{n}$

Assume Adv. blocks w/ prob 1/2.

Then prob. a given slot is one

where Alice sends and there is no

jam is $\frac{c}{2\sqrt{n}}$

$$\text{Prob(Bob fails to get message)} \sim \left(1 - \frac{c}{2\sqrt{n}}\right)^{c\sqrt{n}}$$

$$\leq e^{-c^2/2}$$

An Idea

- A round consists of n slots
- Alice sends w/ prob $c/\sqrt{n}$
- Bob listens w/ prob $c/\sqrt{n}$

Assume Adv. blocks w/ prob 1/2.

Then prob. a given slot is one where Alice sends and there is no jam is $\frac{c}{2\sqrt{n}}$

$$\text{Prob(Bob fails to get message)} \sim \left(1 - \frac{c}{2\sqrt{n}}\right)^{c\sqrt{n}}$$

$$\leq e^{-c^2/2}$$

Bob can request a resend if necessary

An Idea

- A round consists of n slots
- Alice sends w/ prob $c/\sqrt{n}$
- Bob listens w/ prob $c/\sqrt{n}$

Assume Adv. blocks w/ prob 1/2.

Then prob. a given slot is one where Alice sends and there is no jam is $\frac{c}{2\sqrt{n}}$

$$\text{Prob(Bob fails to get message)} \sim \left(1 - \frac{c}{2\sqrt{n}}\right)^{c\sqrt{n}}$$

$$\leq e^{-c^2/2}$$

Bob can request a resend if necessary

After each failed round, n can double in size

Problem

- Adv. can imitate Bob and keep sending fake requests and thereby bankrupt Alice
- Idea: Impose a larger cost to trigger a resend, to mitigate increased cost to Alice

Our Algorithm: Round i

Send Phase: For 2^{ci} slots do

- Alice sends with prob. $2/2^i$
- Bob listens with prob. $2/2^{(c-1)i}$

Req Phase: For 2^i slots do

- If Bob has not received m, Bob sends **req** message
- Alice listens with prob. $4/2^i$

If Alice listened in Req phase and detected no **req** message or collision then algorithm terminates

Our Algorithm: Round i

Send Phase: For 2^{ci} slots do

- Alice sends with prob. $2/2^i$
- Bob listens with prob. $2/2^{(c-1)i}$

Req Phase: For 2^i slots do

- If Bob has not received m, Bob sends **req** message
- Alice listens with prob. $4/2^i$

If Alice listened in Req phase and detected no **req** message or collision then algorithm terminates

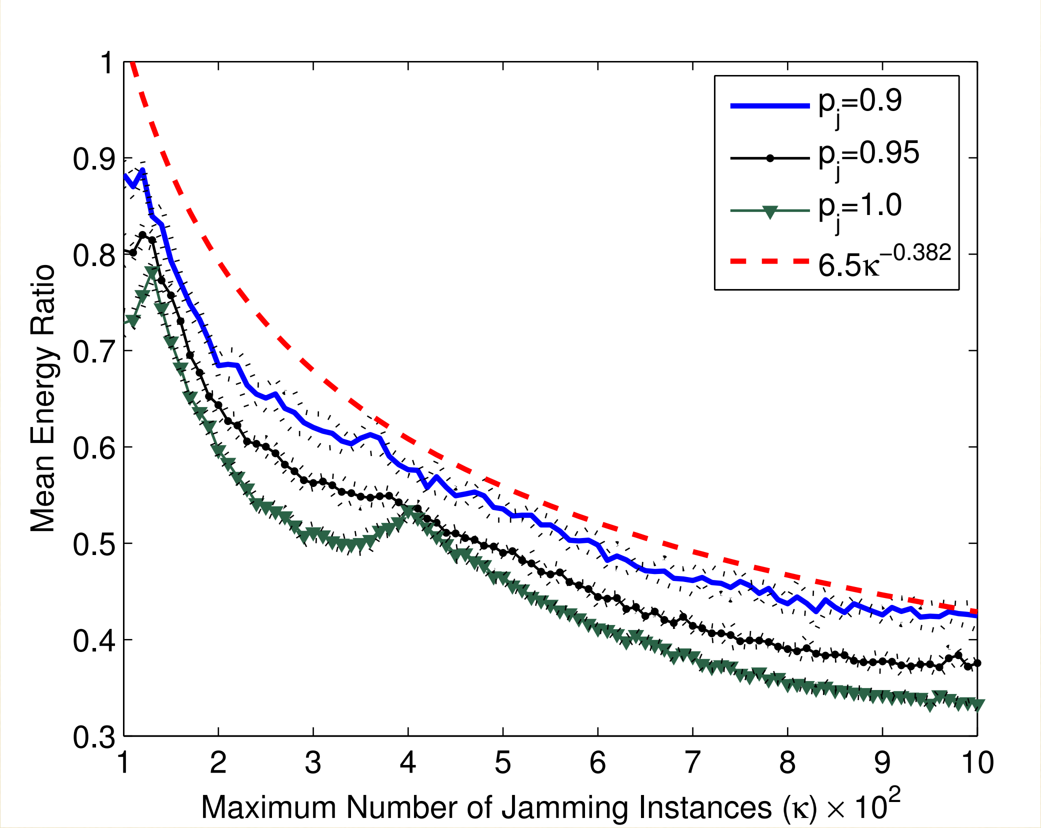
Analysis shows it's best to set $c = \varphi$

Result

Theorem: Our algorithm has the following properties:

- The expected cost to Alice and Bob is $O(B^{\varphi-1}+1) = O(B^{0.62}+1)$.
- Alice and Bob terminate within $O(B^{\varphi})$ slots in expectation.

Simulations

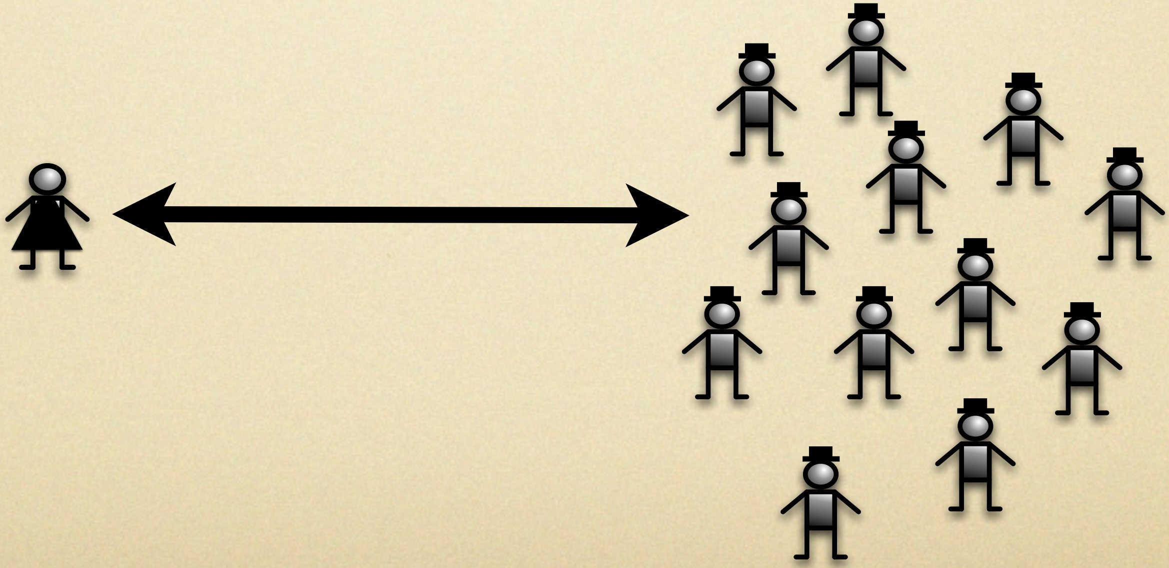


p_j is probability adversary jams a slot

Many Receivers

Theorem: There exists an algorithm for one sender and n receivers that ensures the message is delivered to all receivers and has the following costs:

- The sender's expected cost is $O(B^{\varphi-1}\log n + \log^{\varphi} n)$
- The expected cost to any receiver is $O(B^{\varphi-1} + \log n)$
- The worst case number of slots used is $O((B + \log^{\varphi-1})^{\varphi+1})$

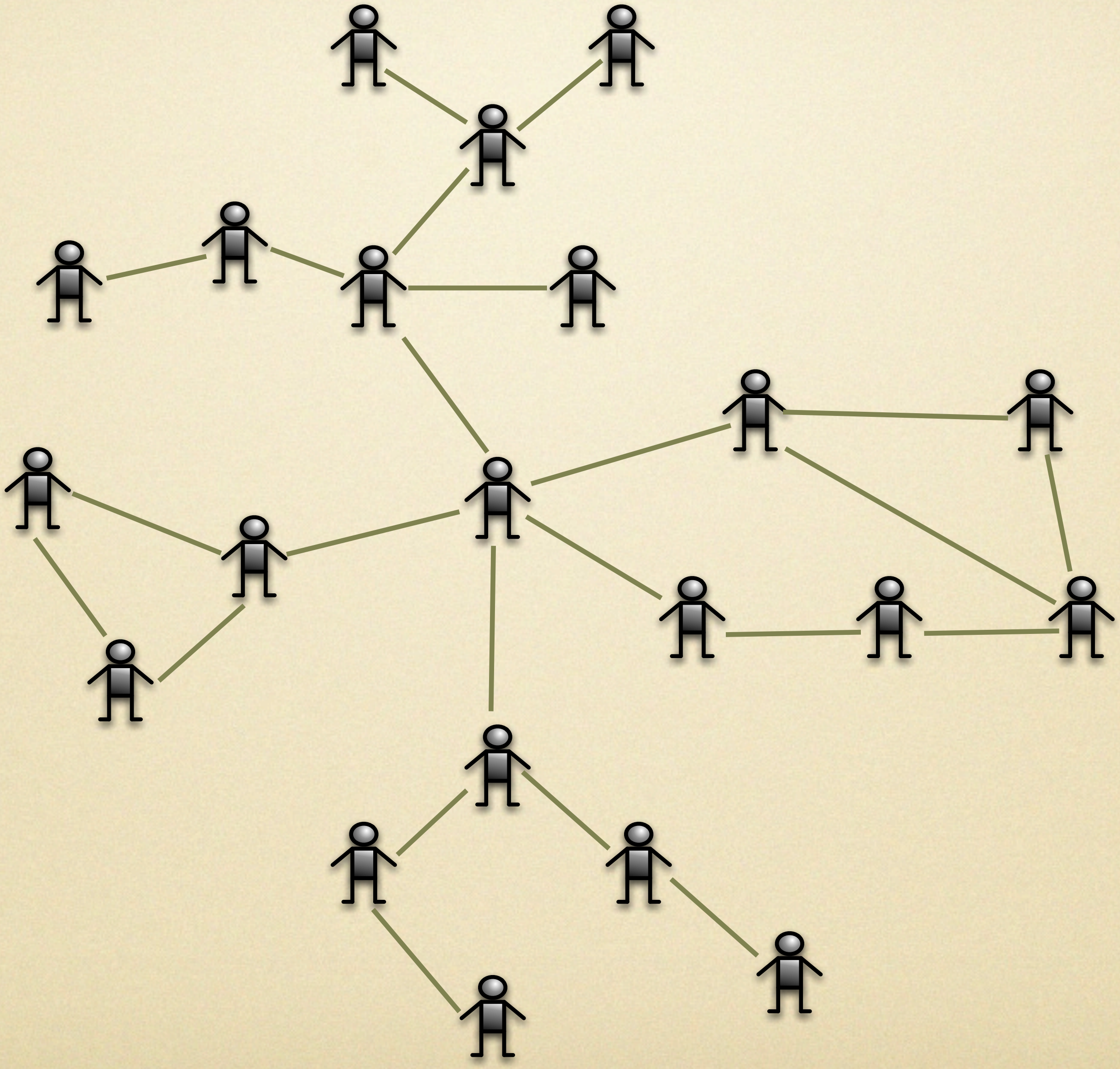


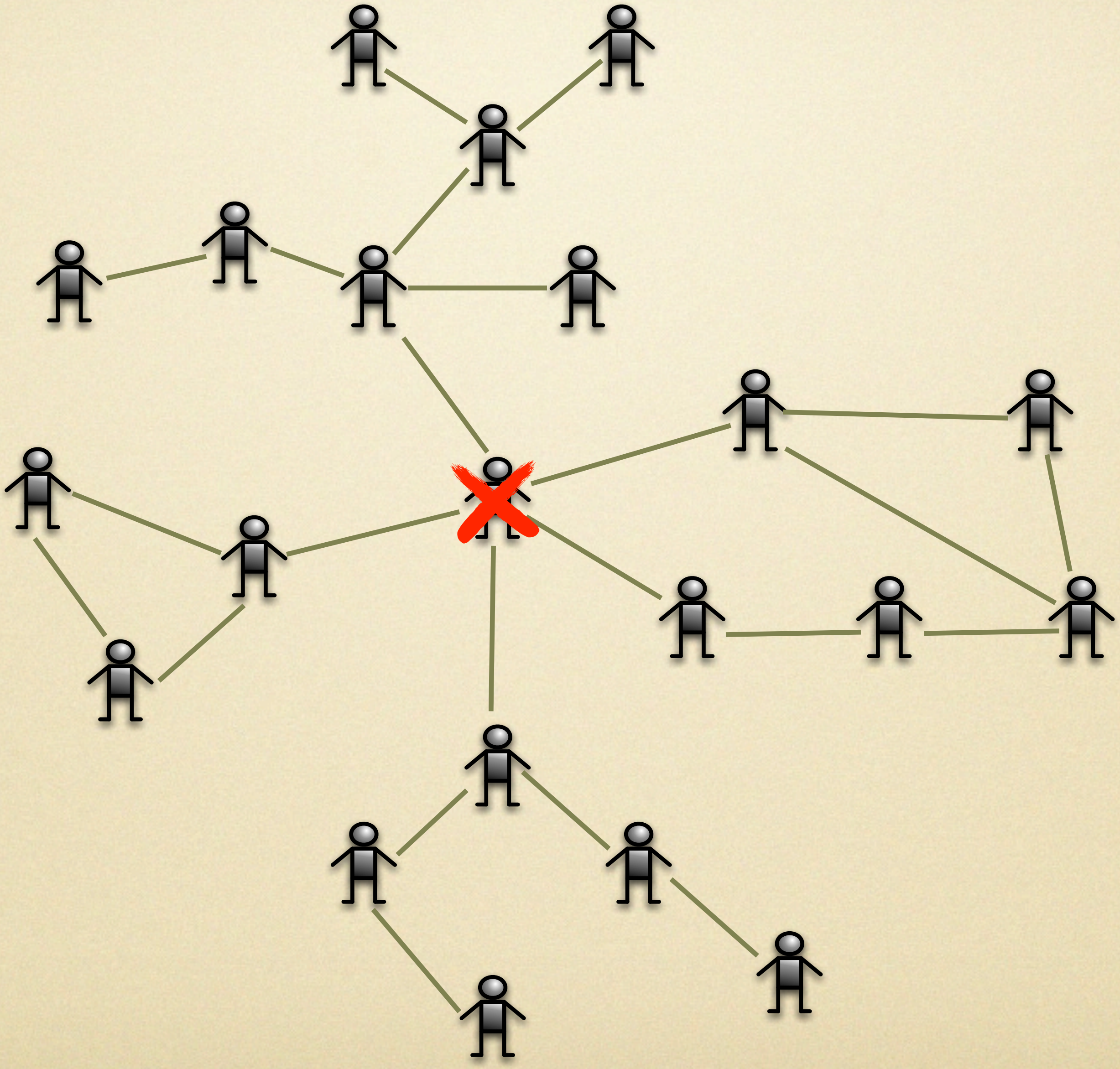
Many Players

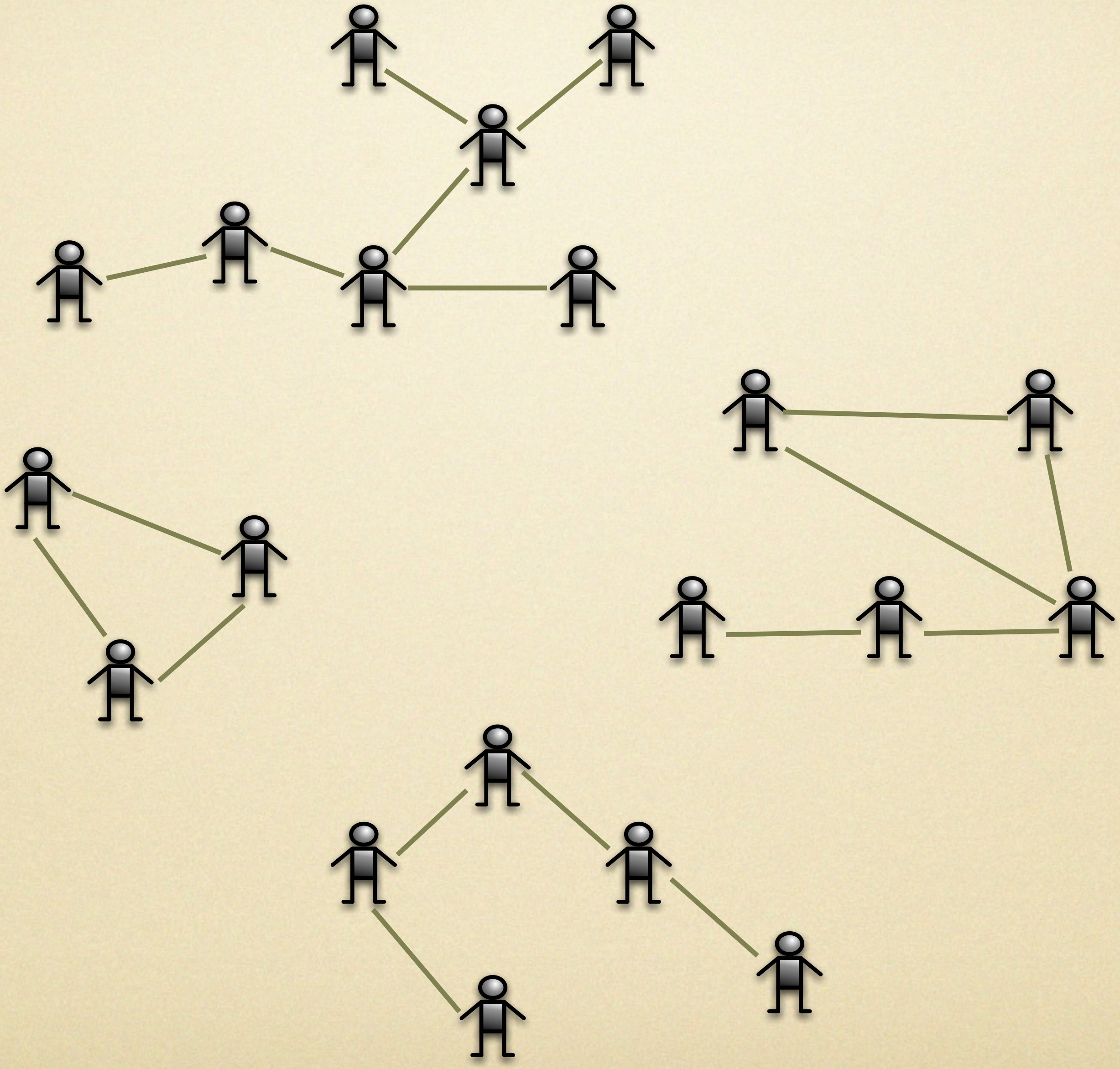
- One player (dealer) wants to transmit a message to all other players in an arbitrary graph (graph and dealer location known to all)
- Assume in any broadcast neighborhood, that the fraction of adversarial players is small enough to achieve broadcast
- Then can achieve broadcast, and adversary can force good players to expend only $o(B)$ additional energy

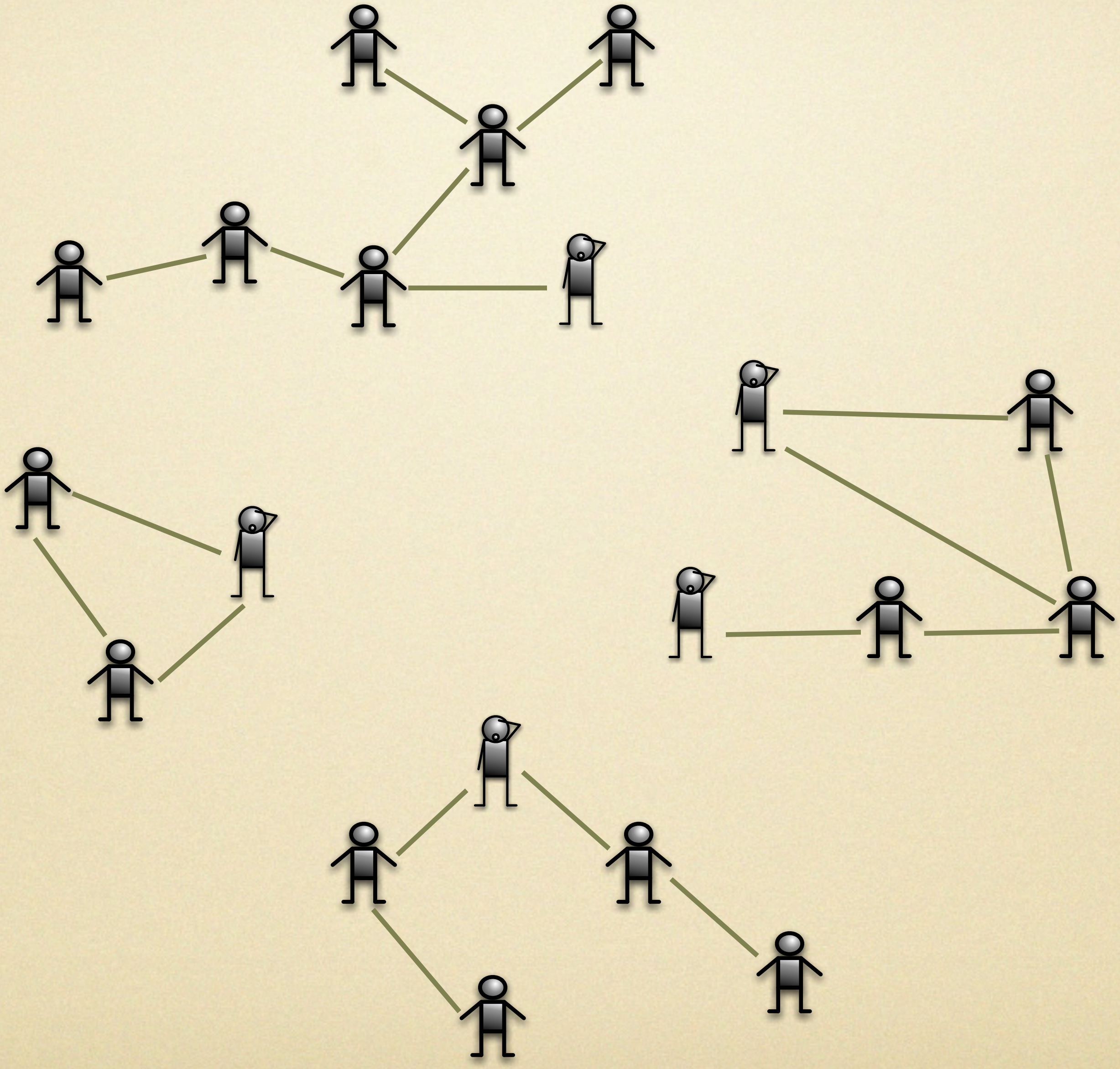
Self-Healing Network

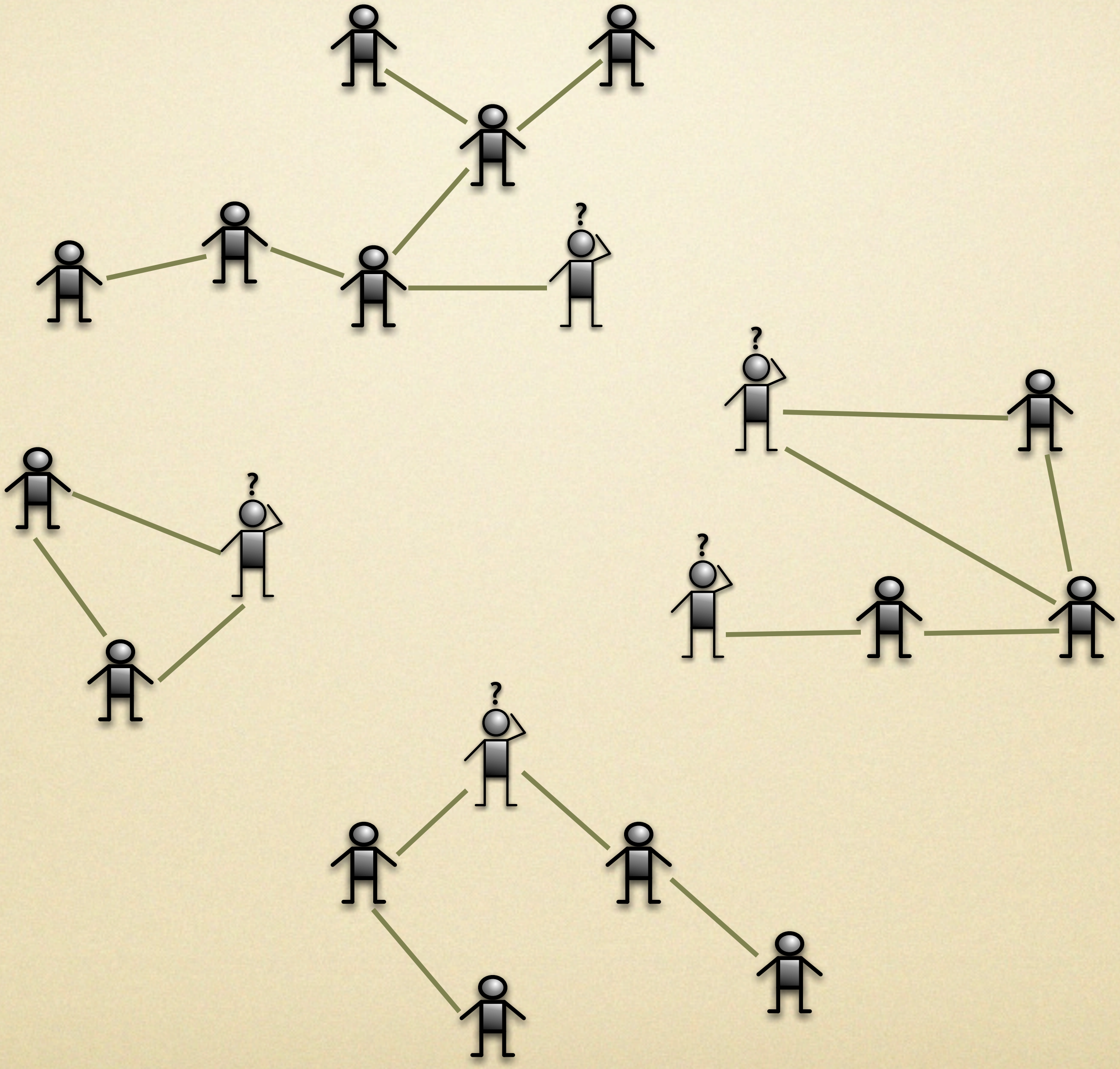
Original Network



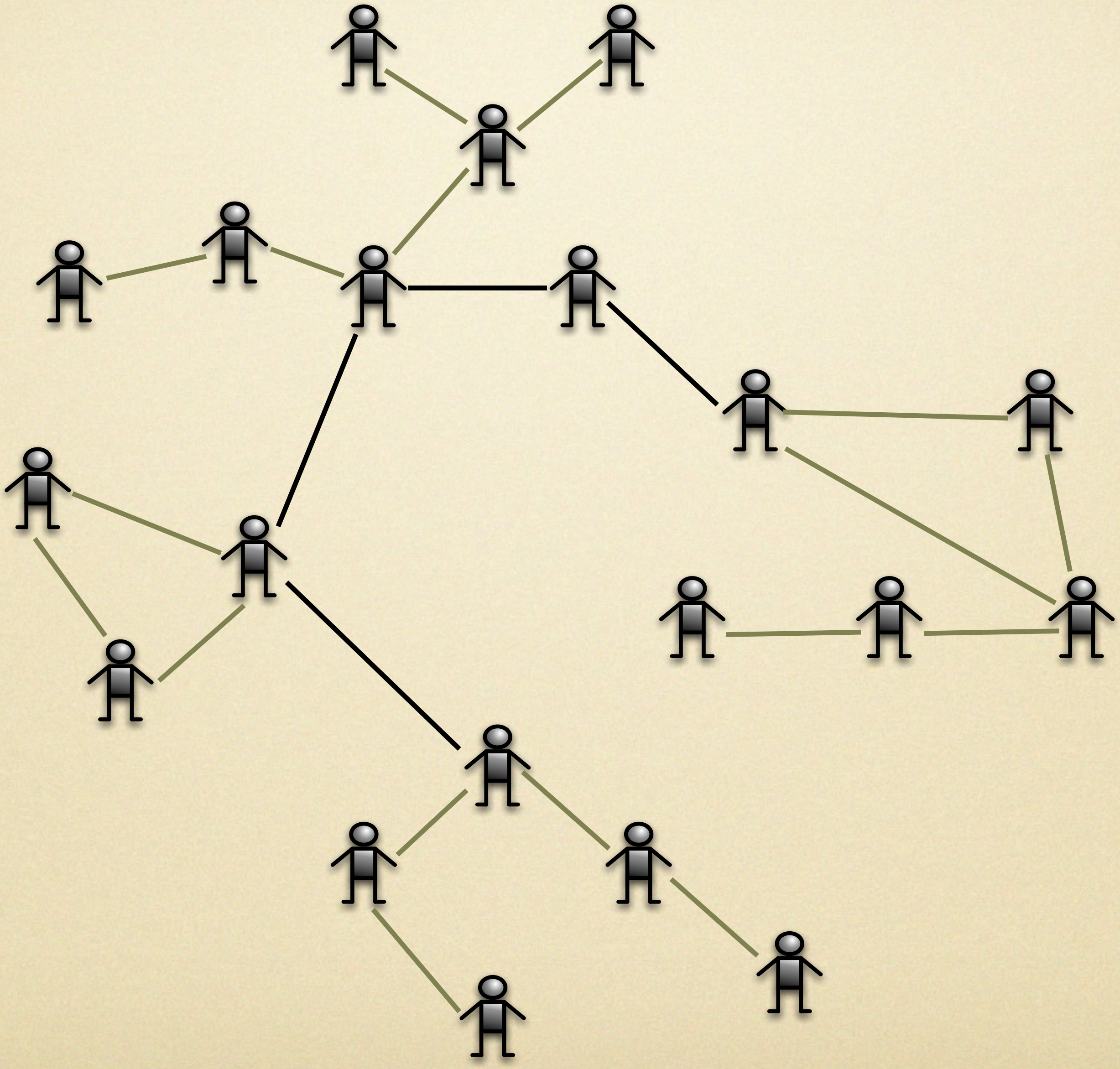






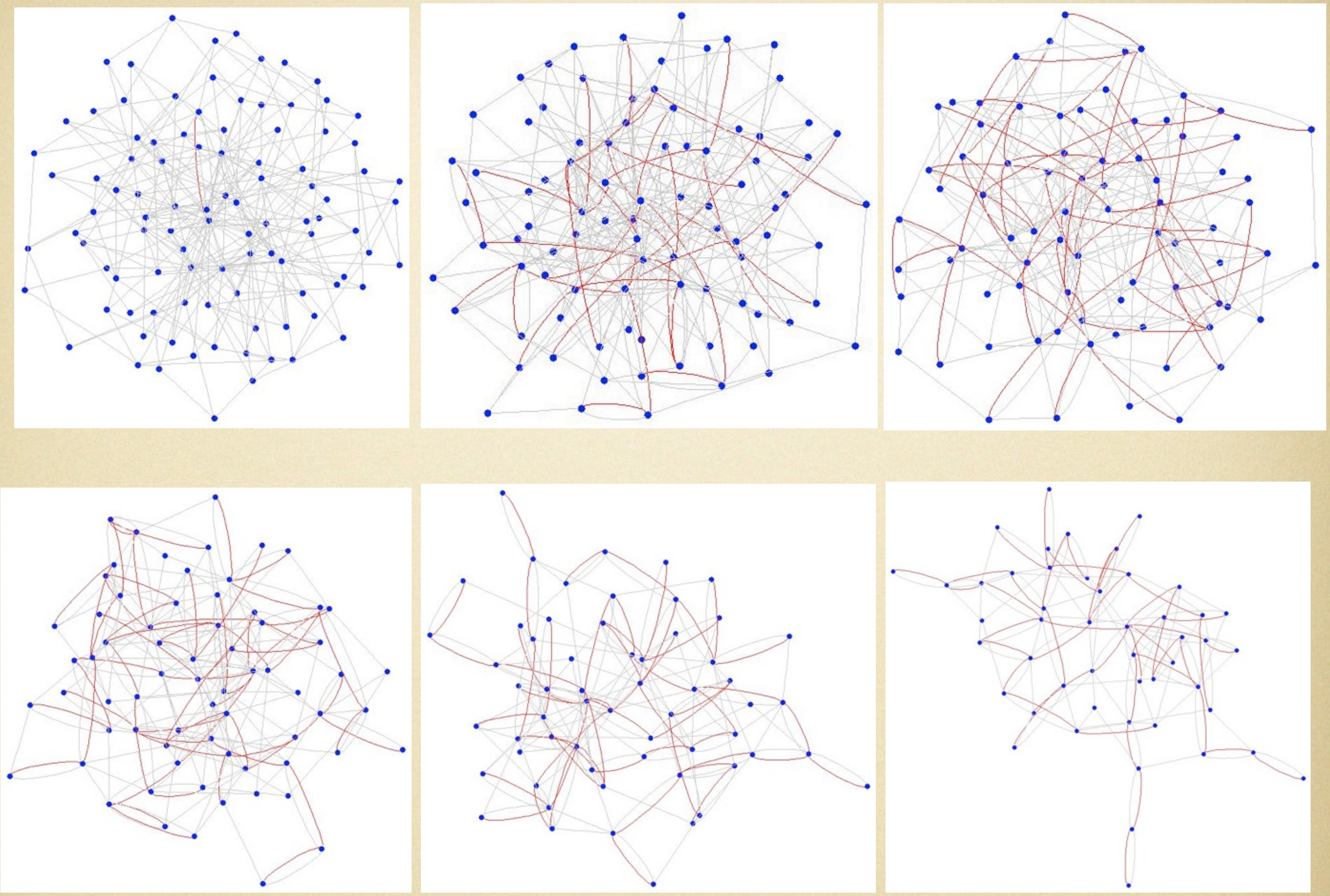


Recovered Network



Problem

- Game between adversary and algorithm on a graph
- Adversary deletes nodes
- Algorithm adds edges
- Goal of algorithm: Keep distances “small” while ensuring no node gets overloaded with edges



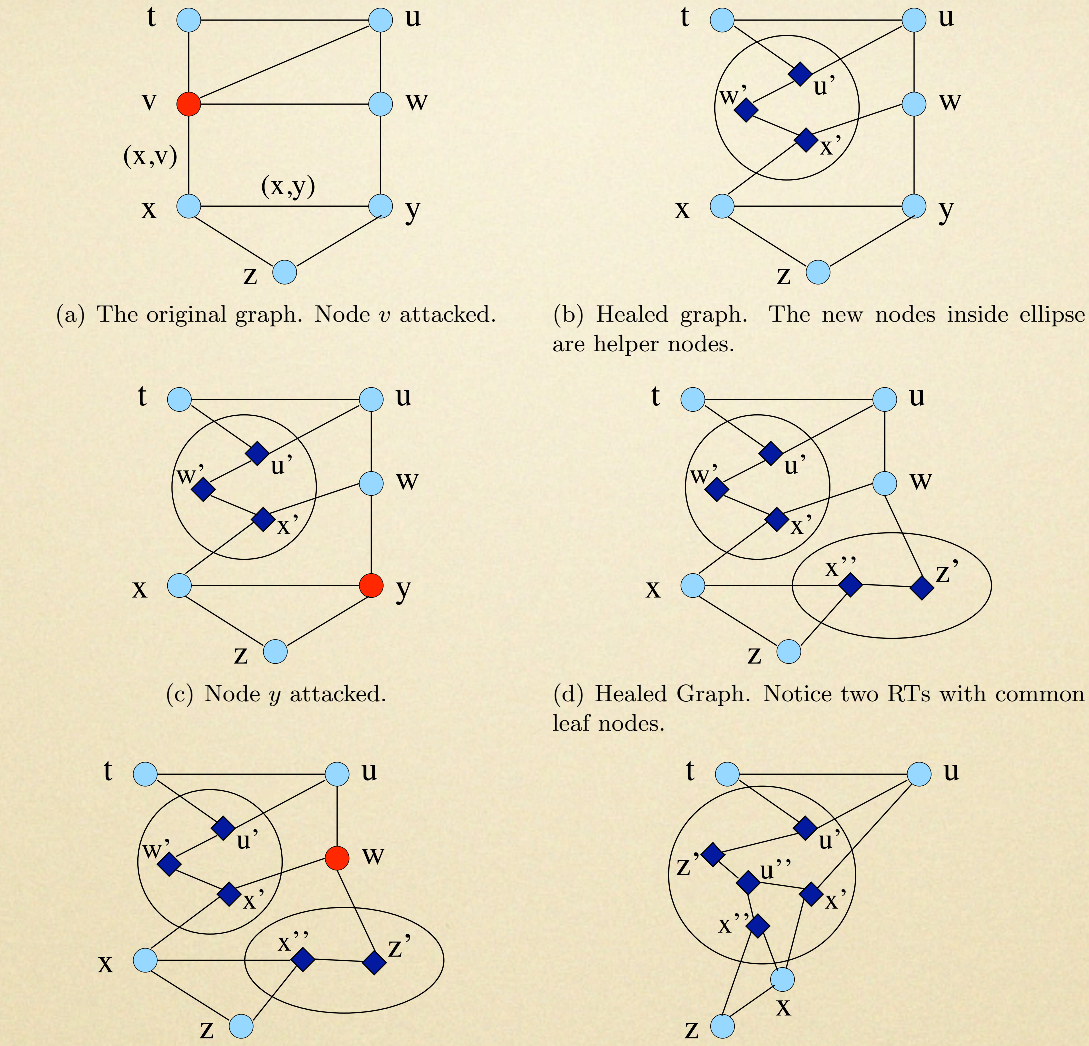
Result

Our algorithm ensures:

- Shortest path between any pair of nodes increases by no more than log n mult. factor
- Each node increases degree by no more than mult. factor of 3
- Each “healing” requires latency and messages per proc. that is logarithmic

Idea

- Maintain a collection of distributed data structures called RT’s
- These RT’s give information on what new links should be maintained
- When a node is deleted, quickly update the RT’s



(a) The original graph. Node v attacked.

(b) Healed graph. The new nodes inside ellipse are helper nodes.

(c) Node y attacked.

(d) Healed Graph. Notice two RTs with common leaf nodes.

(e) Node w attacked: notice w is a common leaf of both RTs

(f) Healed Graph. The RTs have merged. Some of the leaf nodes (x's, u's) are identical (so the picture no longer shows the RT resembling a haft. However, refer figure 10).

Outcomes

- ## Keep Shortest Paths Small

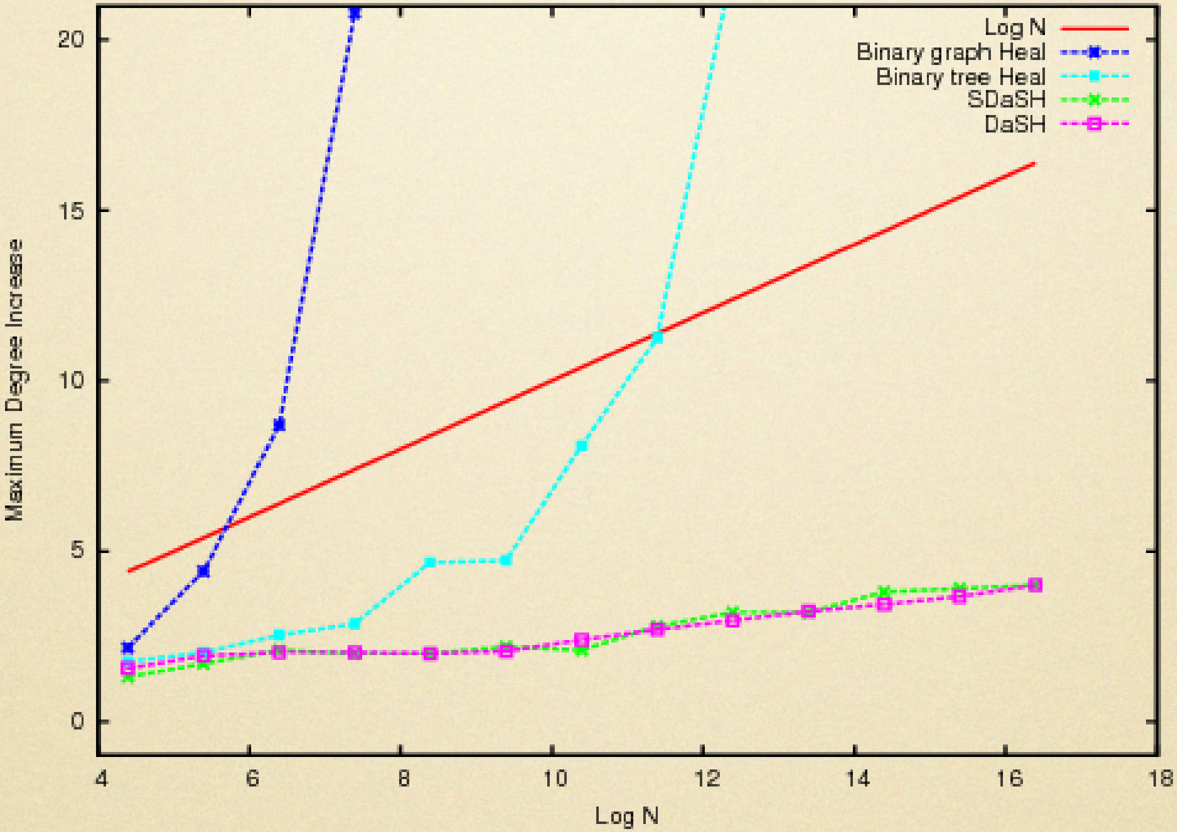
"The Forgiving Graph: A Distributed Data Structure for Maintaining Low Stretch under Adversarial Attack" by Tom Hayes, Jared Saia and Amitabh Trehan, Principles of Distributed Computing(PODC), 2009.

- ## Keep Diameter Small

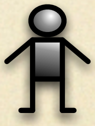
"The Forgiving Tree: A Self-Healing Distributed Data Structure" by Tom Hayes and Navin Rustagi and Jared Saia and Amitabh Trehan, Principles of Distributed Computing(PODC), 2008.

- ## Maintain Connectivity

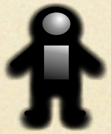
"Picking up the Pieces: Self-Healing in Reconfigurable Networks" by Jared Saia and Amitabh Trehan In IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2008


Log N
Binary graph Heal
Binary tree Heal
SDaSH
DaSH
Maximum Degree Increase
0
5
10
15
20
4
6
8
10
12
14
16
18
Log N

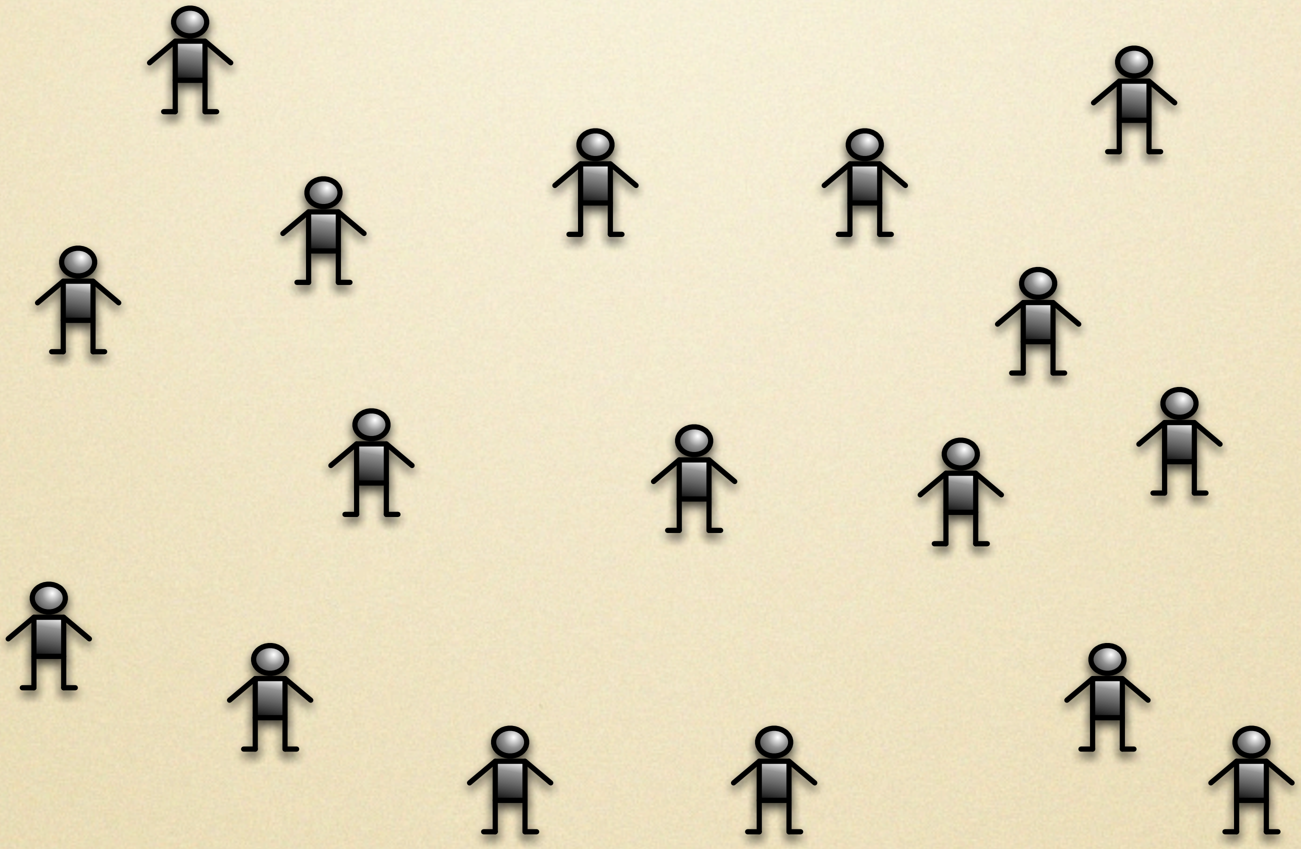
Defense



Vision?



Vision



Vision

- Many small, interchangeable components
- Simple, decentralized algorithms
- Security through obscurity? Yes! But obscurity encapsulated in random bits

Vision

- Provably maintain **invariants** under **attack**
  - **Invariants**: 1) consensus; 2) communication; 3) short paths
  - **Attack**: 1) control of procs; 2) jamming channels; 3) deletion of procs

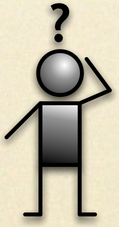
Future Work

- Practical Byzantine agreement; Scalable Distributed Computation: e.g. MapReduce without a master
- Web Censorship: Can we obtain an asymptotic economic analysis, like for jamming?
- Social networks: Self-healing and conflict around information diffusion

Future Work

- Amortized Robustness: “Fool me once, shame on you. Fool me $\omega(\log n)$ times, shame on me.”
- Can we enable enforcement of a “distributed treaty” in systems like Bittorrent?

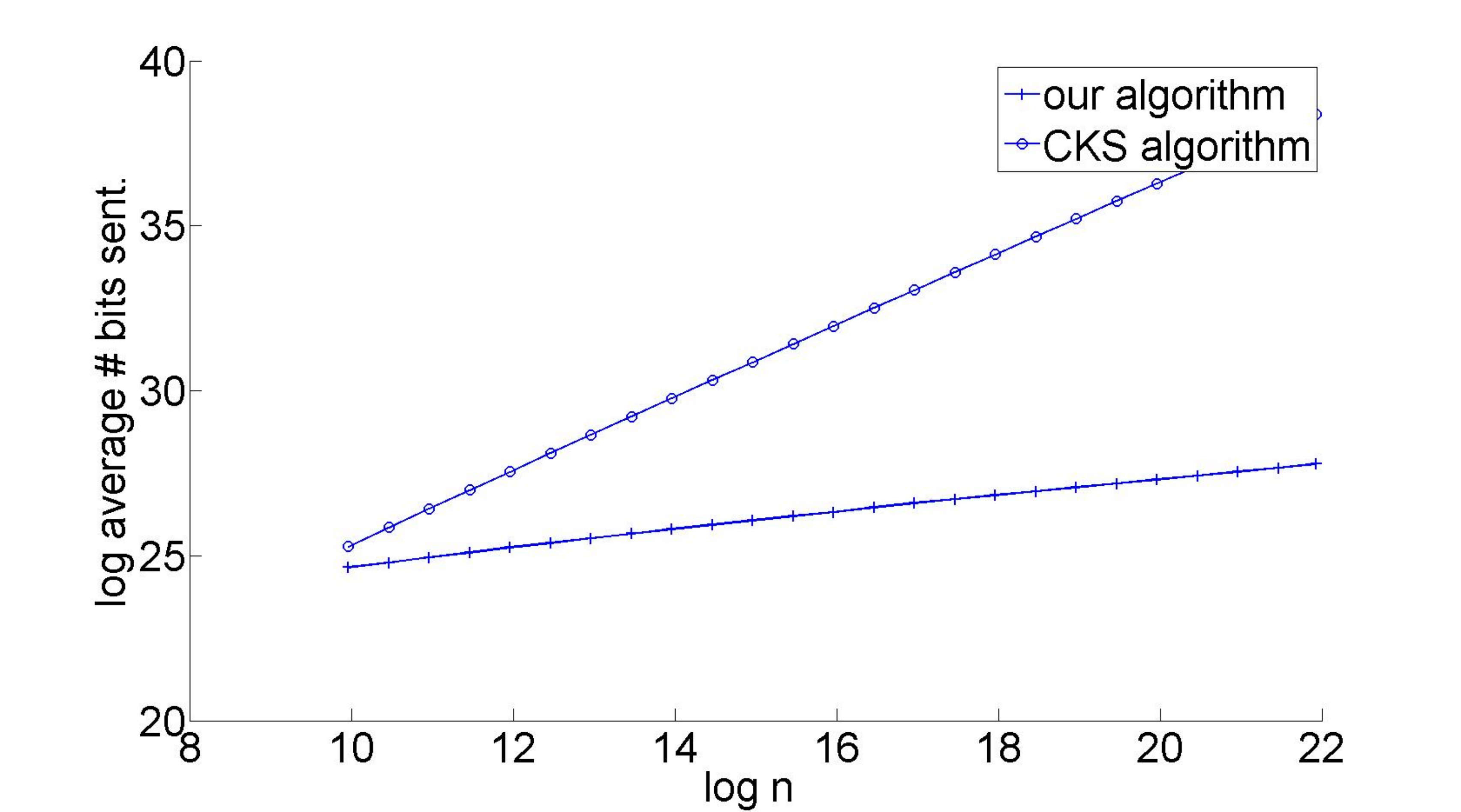
Questions



Lessons Learned

- 1) Don’t trust a processor to run its own code! Instead share state of a processor over more of the network as that processor gets more important.
- 2) Don’t let bad guys group together! Use samplers to spread them out.
- 3) More efficient to render cheating ineffective than to create infrastructure to catch cheaters

Bits vs n (log-log)



Reactive Jammer

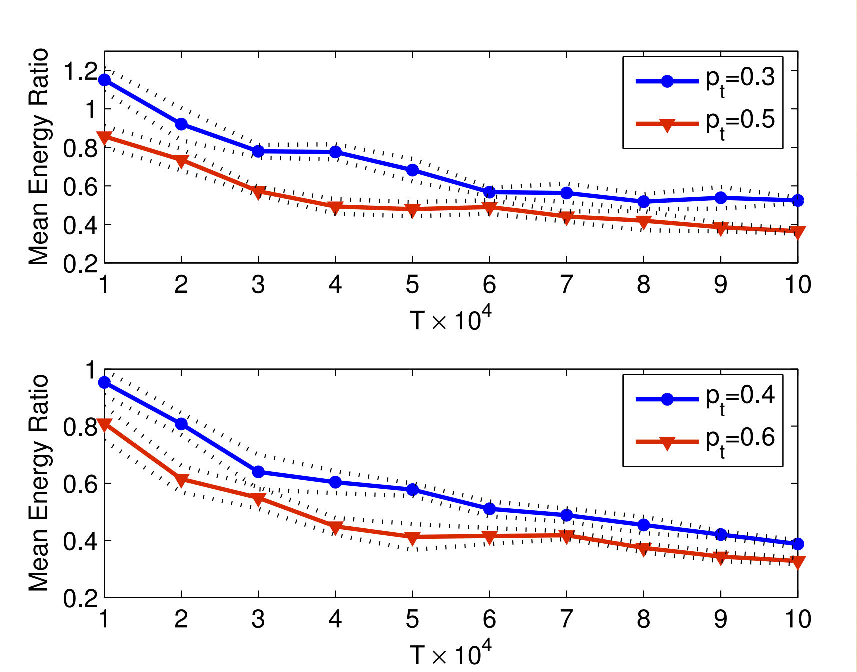


Fig. 4. Mean energy ratio (maximum of either player) for a reactive jammer with $p_t = 0.3, 0.4, 0.5$ and 0.6, separated for clarity. Dotted lines signify 95% confidence intervals.