Scalable and Secure Computation Among Strangers: Message-Competitive Byzantine Protocols

Jared Saia

Joint with: John Augustine, Valerie King, Anisur Rahaman Molla, and Gopal Pandurangan

Permissionless Networks

Permissionless networks are large; nodes join and leave at will

Nodes are known by self-generated IDs

Adversarial (Byzantine) IDs common

Goal: Solve coordination problems with sub-quadratic messages

Scalability

Recent work* ensures solutions to Byzantine agreement/leader election where good nodes send $\tilde{O}(n)$ messages total

Assume KT_1 *model:* Each good node knows IDs of all neighbors.

How can we extend this result to churn?

One step is to extend it to KT_0 model

Scalability

Recent work* ensures solutions to Byzantine agreement/leader election where good nodes send $\tilde{O}(n)$ messages total

Assume KT_1 *model:* Each good node knows IDs of all neighbors.

How can we extend this result to churn?

One step is to extend it to KT_0 model

*Braud-Santoni, et al. PODC ‘13

KT_0 Model

Nodes don't know their neighbors a priori

But they do learn a neighbors ID upon receiving a message from it

Can convert KT_0 to KT_1:

- Initial step where each node communicates with all neighbors solely to learn IDs
- But this is $\Theta(n^2)$ messages

Can we design Byzantine agreement/leader election protocols that require sub-quadratic messages in KT_0?

Our Model

Adversary is *static*, *rushing* and computationally-unbounded.

n nodes have distinct IDs in $[1, n^k]$. Byzantine nodes choose their IDs.

Synchronous, fully-connected KT_0 model

Upper Bound

Theorem: Our algorithm solves Byzantine agreement, leader and committee election in KT_0 with:

$O(\text{polylog}(n))$ latency

$O((T+n)\log n)$ expected messages

$T = \min(n^2, \text{\# bits sent by adversary})$

Upper Bound

Theorem: Our algorithm solves Byzantine agreement, leader and committee election in KT_0 with:

$O(\text{polylog}(n))$ latency

$O((T+n)\log n)$ expected messages

$T = \min(n^2, \text{\# bits sent by adversary})$

Handles $< 1/4$ fraction of Byzantine faults

Succeeds with probability $1 - \frac{1}{n^c}$ for any fixed, positive c

Upper Bound

Theorem: Our algorithm solves Byzantine agreement, leader and committee election in KT_0 with:

$O(\text{polylog}(n))$ latency $\longleftarrow$ Holds even in CONGEST

$O((T+n)\log n)$ expected messages

$T = \min(n^2, \text{\# bits sent by adversary})$

Handles $< 1/4$ fraction of Byzantine faults

Succeeds with probability $1 - \dfrac{1}{n^c}$ for any fixed, positive c

Talking to Strangers

In fact, our algorithm only writes to unknown IDs (strangers) via two primitives:

(1) Random stranger

(2) All strangers

LB for polylog(n) rounds

LB for polylog(n) rounds

$\Omega(nt)$ messages needed when $t \leq n$ bad nodes
[Hadzilacos and Halpern, 91]

LB for polylog(n) rounds

$\Omega(nt)$ messages needed when $t \leq n$ bad nodes [Hadzilacos and Halpern, 91]

In CONGEST, $\tilde{O}(n)$ bits can be sent by Byzantine nodes in poylog(n) rounds

LB for polylog(n) rounds

$\Omega(nt)$ messages needed when $t \leq n$ bad nodes [Hadzilacos and Halpern, 91]

In CONGEST, $\tilde{O}(n)$ bits can be sent by Byzantine nodes in poylog(n) rounds

Thus, $T = \tilde{O}(tn)$

LB for polylog(n) rounds

$\Omega(nt)$ messages needed when $t \leq n$ bad nodes [Hadzilacos and Halpern, 91]

In CONGEST, $\tilde{O}(n)$ bits can be sent by Byzantine nodes in poylog(n) rounds

Thus, $T = \tilde{O}(tn)$

So $\tilde{\Omega}(T)$ messages needed for polylog(n) round algorithms in CONGEST

LB for polylog(n) rounds

$\Omega(nt)$ messages needed when $t \leq n$ bad nodes
[Hadzilacos and Halpern, 91]

In CONGEST, $\tilde{O}(n)$ bits can be sent by Byzantine nodes in poylog(n) rounds

Thus, $T = \tilde{O}(tn)$

So $\tilde{\Omega}(T)$ messages needed for polylog(n) round algorithms in CONGEST

Even for algs succeeding whp

LB for polylog(n) rounds

$\Omega(nt)$ messages needed when $t \leq n$ bad nodes
[Hadzilacos and Halpern, 91]

In CONGEST, $\tilde{O}(n)$ bits can be sent by
Byzantine nodes in poylog(n) rounds

Thus, $T = \tilde{O}(tn)$

So $\tilde{\Omega}(T)$ messages needed for polylog(n) round
algorithms in CONGEST

Even for algs succeeding whp

Our alg optimal up to log terms

LB for Deterministic

LB for Deterministic

Theorem: For $T = O(n^2)$ bits sent by Byzantine nodes, any *deterministic* algorithm sends $\Omega(T)$ total bits. (also for KT$_1$)

LB for Deterministic

Theorem: For $T = O(n^2)$ bits sent by Byzantine nodes, any *deterministic* algorithm sends $\Omega(T)$ total bits. (also for KT$_1$)

Also show that if $T = n^{1+\alpha}$ for $\alpha \in (0,1]$, then any Las Vegas algorithm sends $\Omega(n^{1+\alpha/2})$ bits in expectation

Upper Bound

Upper Bound

This talk: Byzantine agreement only
Paper: Leader and committee election

Our Algorithm

$p \leftarrow (\log n)/n$

1. Each ID becomes active with probability p
2. Active IDs try to solve Byzantine agreement
3. If fail, then $p \leftarrow 2p$, goto step 1

Our Algorithm

$p \leftarrow (\log n)/n$

1. Each ID becomes active with probability p
2. Active IDs try to solve Byzantine agreement
3. If fail, then $p \leftarrow 2p$, goto step 1

Our Algorithm

$p \leftarrow (\log n)/n$

1. Each ID becomes active with probability p

2. Active IDs try to solve Byzantine agreement

3. If fail, then $p \leftarrow 2p$, goto step 1

Implicit Agreement
Success: $> t/n$ fraction of good IDs decide on correct bit, and remaining good do not decide

Failure: no good IDs decide

Our Algorithm

$p \leftarrow (\log n)/n$

1. Each ID becomes active with probability p
2. Active IDs try to solve Byzantine agreement
3. If fail, then $p \leftarrow 2p$, goto step 1

Implicit Agreement
Success: $> t/n$ fraction of good IDs decide on correct bit, and remaining good do not decide

Failure: no good IDs decide

Promise Agreement
Implicit Agreement output:

Success →all IDs decide correctly
Failure → no IDs decide

Our Algorithm

$p \leftarrow (C \log n)/n$

Implicit Agreement (Effort = p)
Success: $> t/n$ fraction of good IDs decide on correct bit, and remaining good do not decide

Failure: no good IDs decide

Our Algorithm

$p \leftarrow (C \log n)/n$

Implicit Agreement (Effort = p)
Success: $> t/n$ fraction of good IDs decide on correct bit, and remaining good do not decide

Failure: no good IDs decide

Promise Agreement
Implicit Agreement output:

Success→all IDs decide correctly
Failure → no IDs decide

Our Algorithm

$p \leftarrow (C \log n)/n$

Implicit Agreement (Effort = p)
Success: $> t/n$ fraction of good IDs decide on correct bit, and remaining good do not decide

Failure: no good IDs decide

Promise Agreement
Implicit Agreement output:

Success→all IDs decide correctly
Failure → no IDs decide

If fail:
$p \leftarrow 2p$

$\boxed{p < \frac{1}{C \log n}?}$

If Success:
DONE

Our Algorithm

$p \leftarrow (C \log n)/n$

Implicit Agreement (Effort = p)
Success: $> t/n$ fraction of good IDs decide on correct bit, and remaining good do not decide

Failure: no good IDs decide

Promise Agreement
Implicit Agreement output:

Success→all IDs decide correctly
Failure → no IDs decide

If fail:
$p \leftarrow 2p$

$p < \dfrac{1}{C \log n}$?

Y

N

Heavy-weight BA

If Success:
DONE

Our Algorithm

If adversary sends $\leq pn^2$ messages, then Implicit Agreement succeeds

$p \leftarrow (C \log n)/n$

Implicit Agreement (Effort = p)
Success: $> t/n$ fraction of good IDs decide on correct bit, and remaining good do not decide

Failure: no good IDs decide

Y

$p < \dfrac{1}{C \log n}$?

If fail:
$p \leftarrow 2p$

Promise Agreement
Implicit Agreement output:

Success→all IDs decide correctly
Failure → no IDs decide

N

Heavy-weight BA

If Success:
DONE

Implicit Agreement

Each ID is *active* with probability p; broadcasts its ID

Each active ID, x, sets $S_x \leftarrow$ IDs received

Use LARGE-CORE-BA among the active IDs

Implicit Agreement

Each ID is *active* with probability p; broadcasts its ID

Each active ID, x, sets $S_x \leftarrow$IDs received

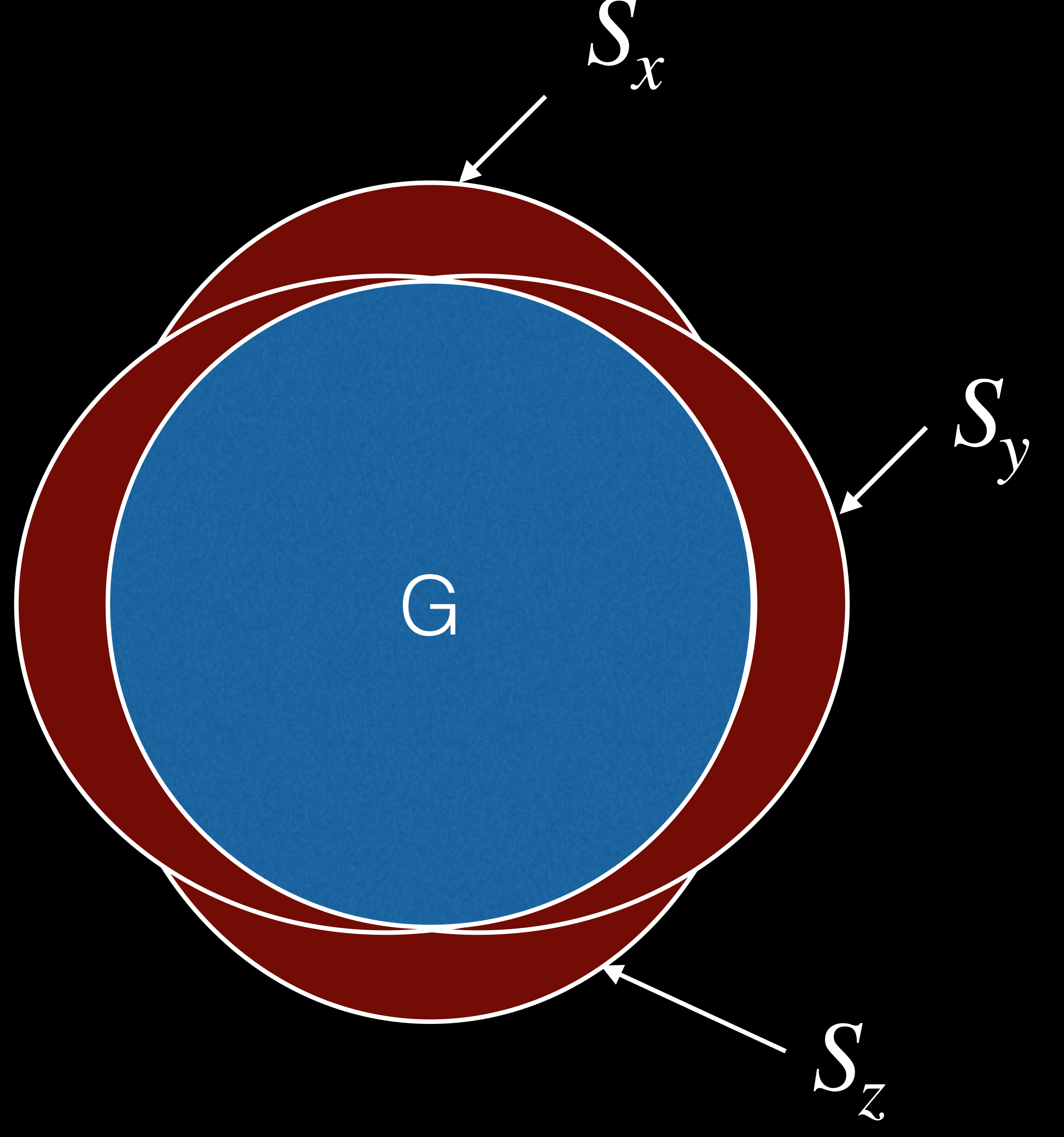
Use LARGE-CORE-BA among the active IDs

LARGE-CORE-BA:

Ensures agreement among nodes whose views “mostly” overlap

Requires IDs in range $[0,n^k]$, for fixed k

LARGE-CORE-BA (LCBA)



Lemma 1. Assume:

G = good IDs, B = bad IDs, i.e. $\cup_{x \in G} S_x$

$G \subseteq_x S_x$; B/G bounded away from 1/2

Then all but $1/\log n$ fraction reach agreement in:

Latency and per ID message cost polylog(|G|+|B|)

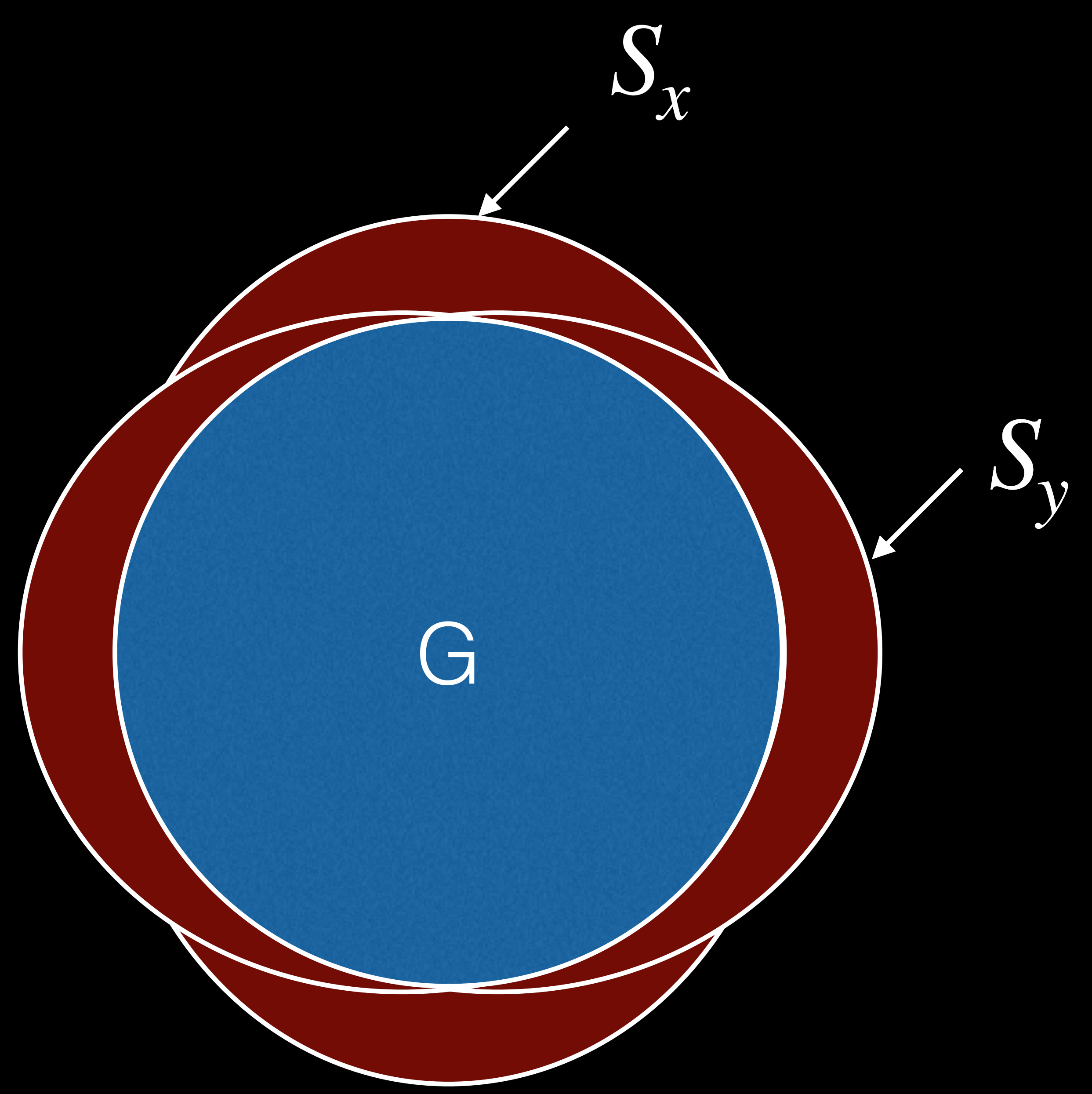
With high probability in |G| + |B|

Two problems

Problem 1: How does each active ID maintain a set S_x to meet LCBA requirements?

Problem 2: How can active IDs agree on whether conditions are favorable for agreement?

Problem 1: Meeting LCBA requirements

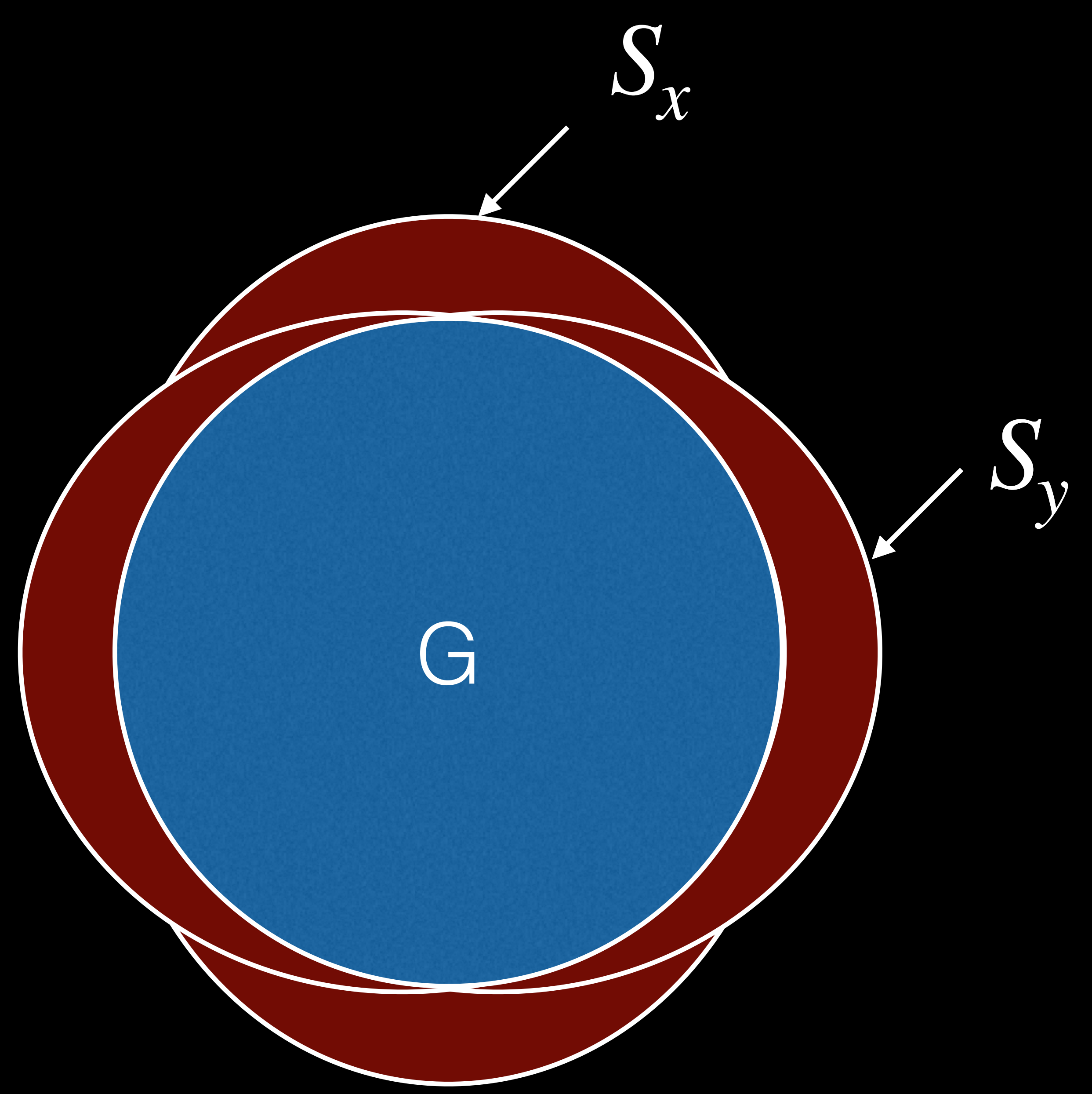


Need |B|/|G| < 1/2 unless adversary sends $\Omega(nA)$ messages, where $A = pn$ is # active nodes

Naive: ID x adds to S_x all IDs that it receives message from

But: Adversary can make |B| = A^2, by sending only A^2 messages

Problem 1: Meeting LCBA requirements

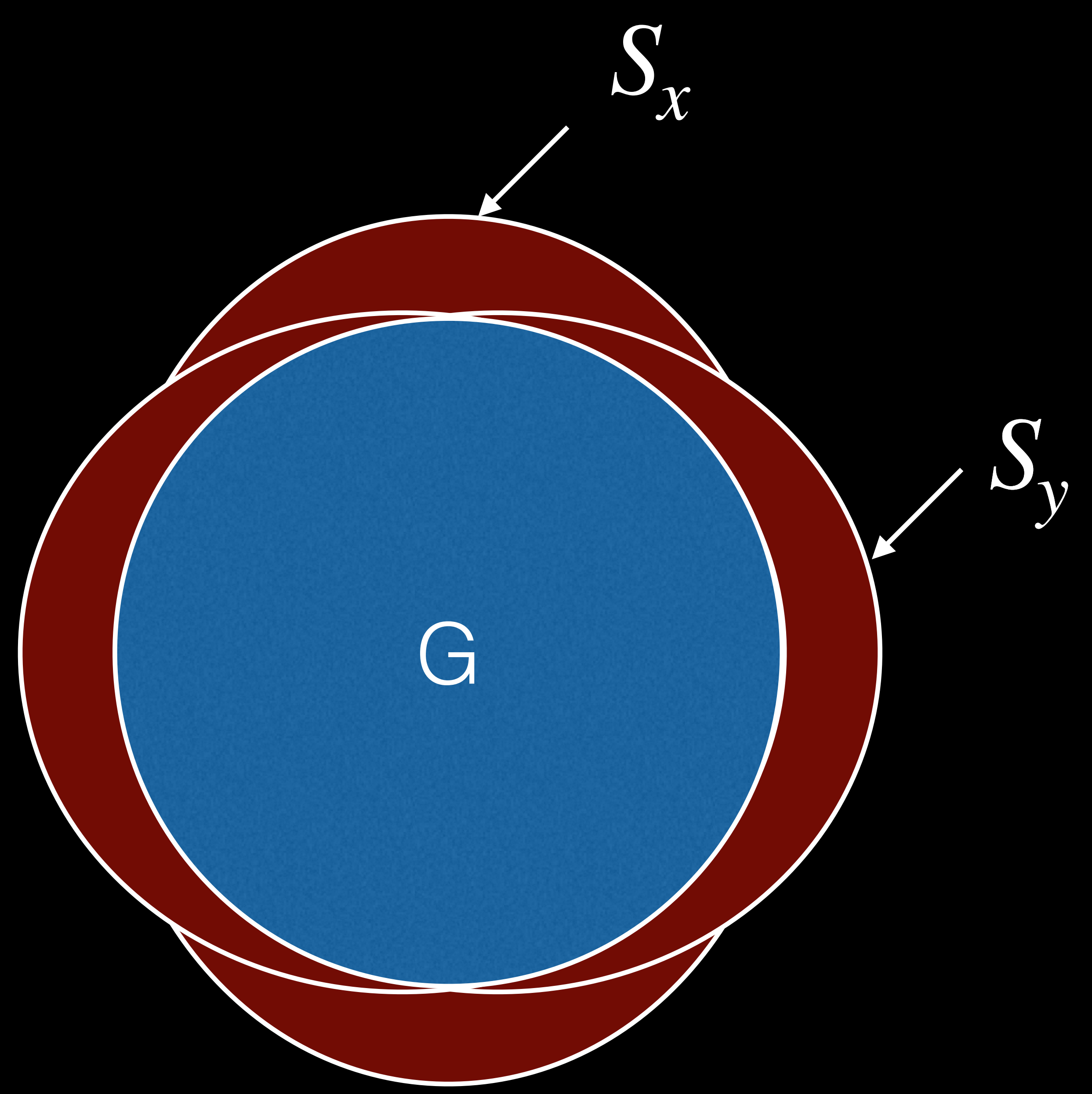


Need |B|/|G| < 1/2 unless adversary sends $\Omega(nA)$ messages, where $A = pn$ is # active nodes

Naive: ID x adds to S_x all IDs that it receives message from

But: Adversary can make |B| = A^2, by sending only A^2 messages

Problem 1: Meeting LCBA requirements



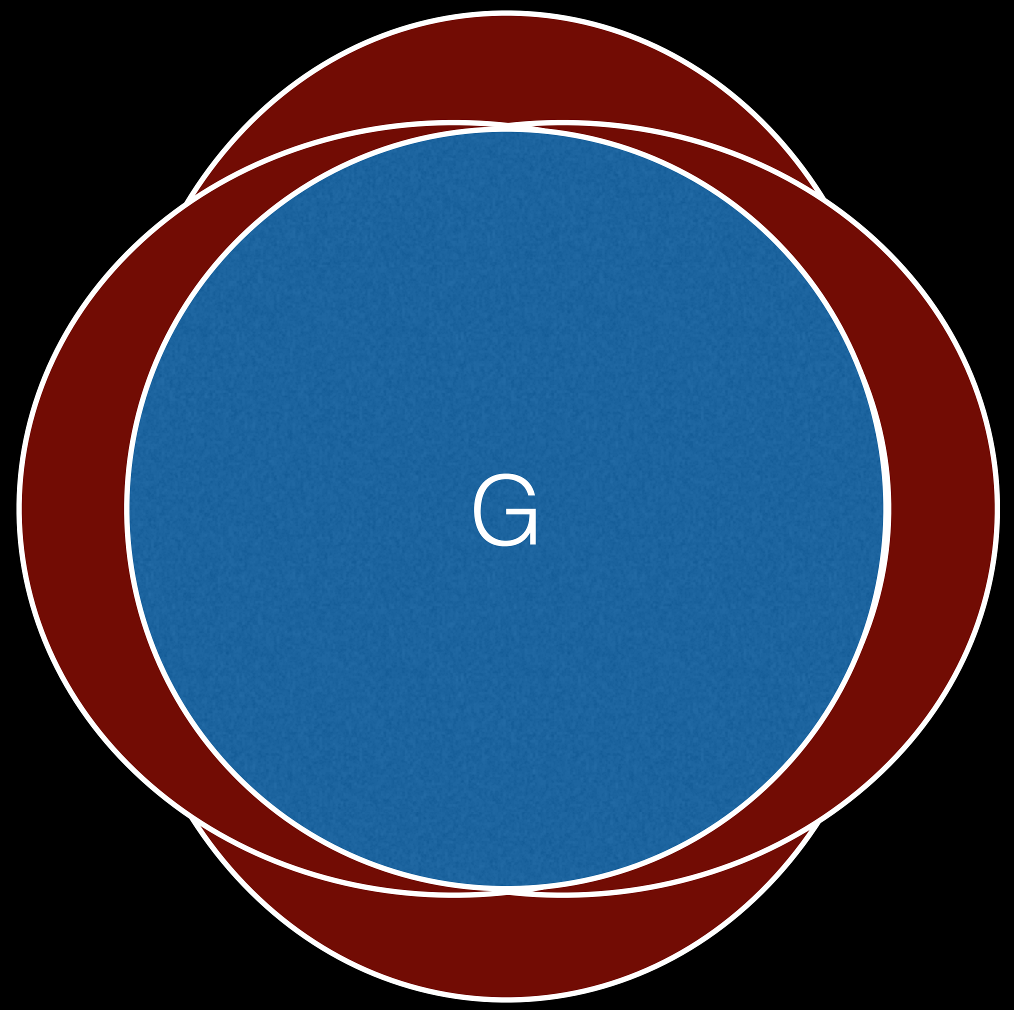
Need |B|/|G| < 1/2 unless adversary sends $\Omega(nA)$ messages, where $A = pn$ is # active nodes

Naive: ID x adds to S_x all IDs that it receives message from

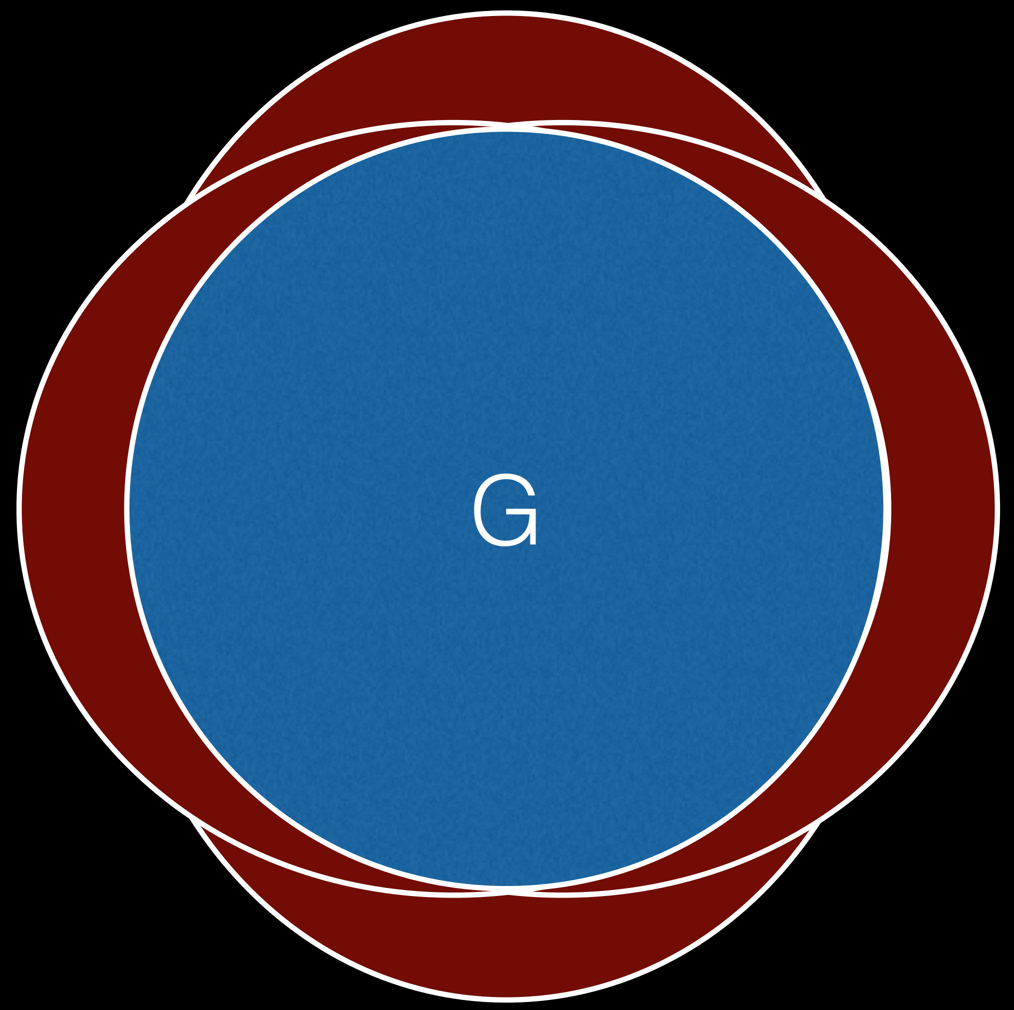
But: Adversary can make |B| = A^2, by sending only A^2 messages

Instead: Use non-active IDs to help out.

Need |B|/|G| < 1/2, unless adversary sends $\Omega(n^2p)$ messages

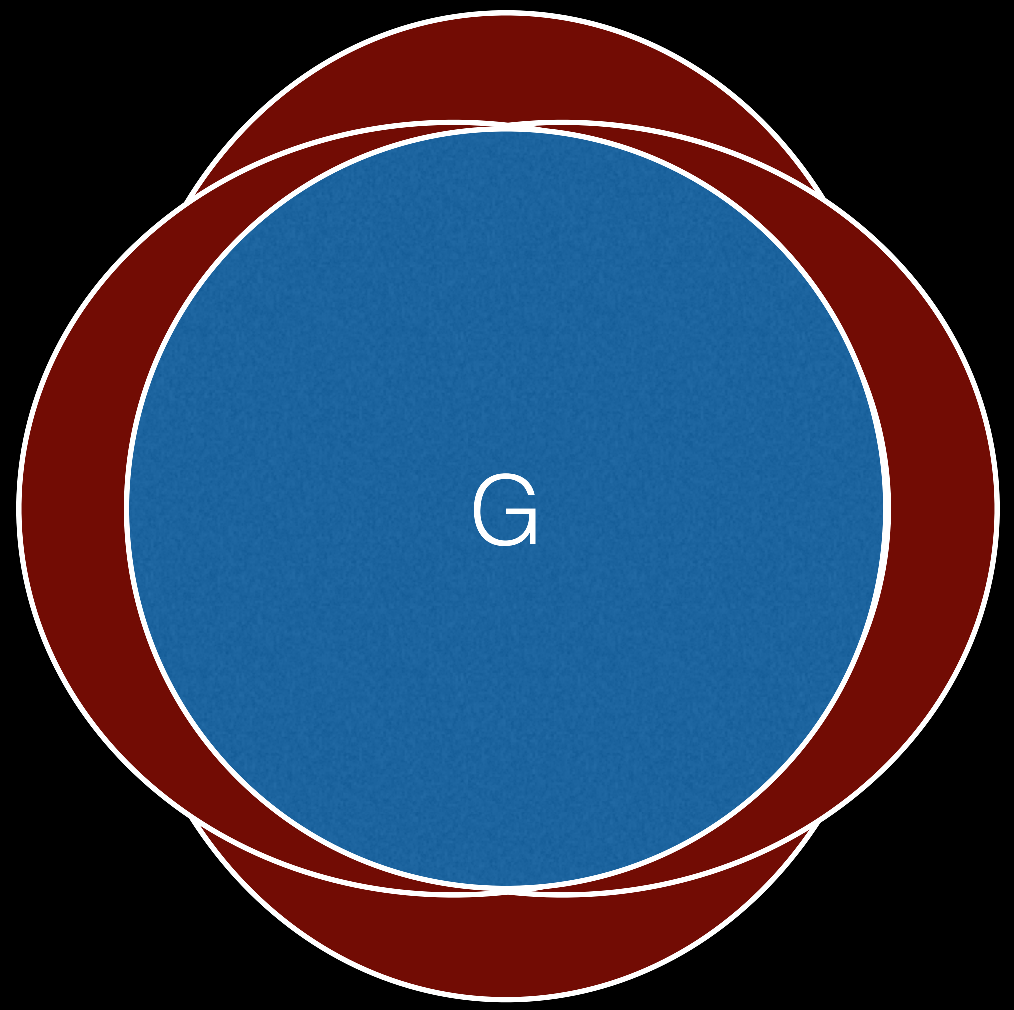


Need |B|/|G| < 1/2, unless adversary sends $\Omega(n^2 p)$ messages



Good ID is *light* if received ~np messages

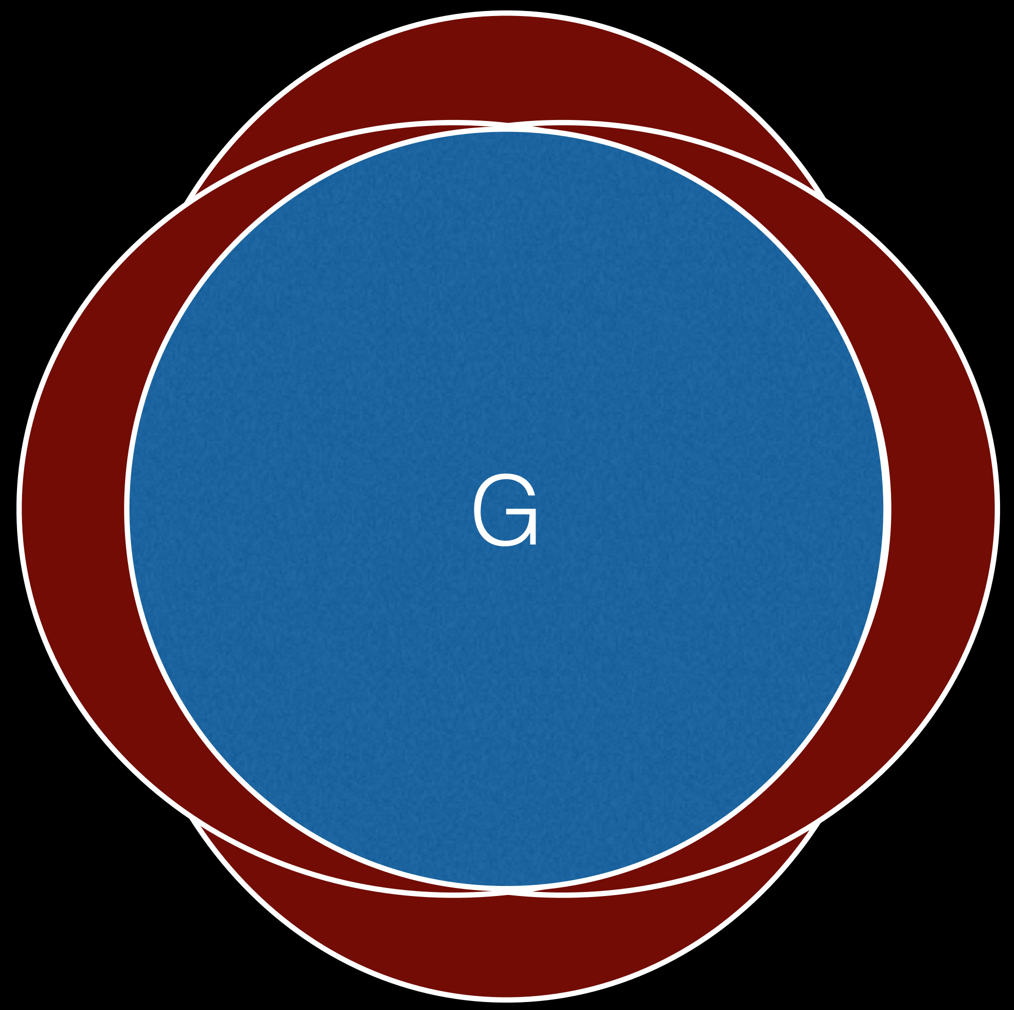
Need |B|/|G| < 1/2, unless adversary sends $\Omega(n^2 p)$ messages



Good ID is *light* if received ~np messages

Light IDs send info about S_x sets to IDs in S_x

Need |B|/|G| < 1/2, unless adversary sends $\Omega(n^2 p)$ messages

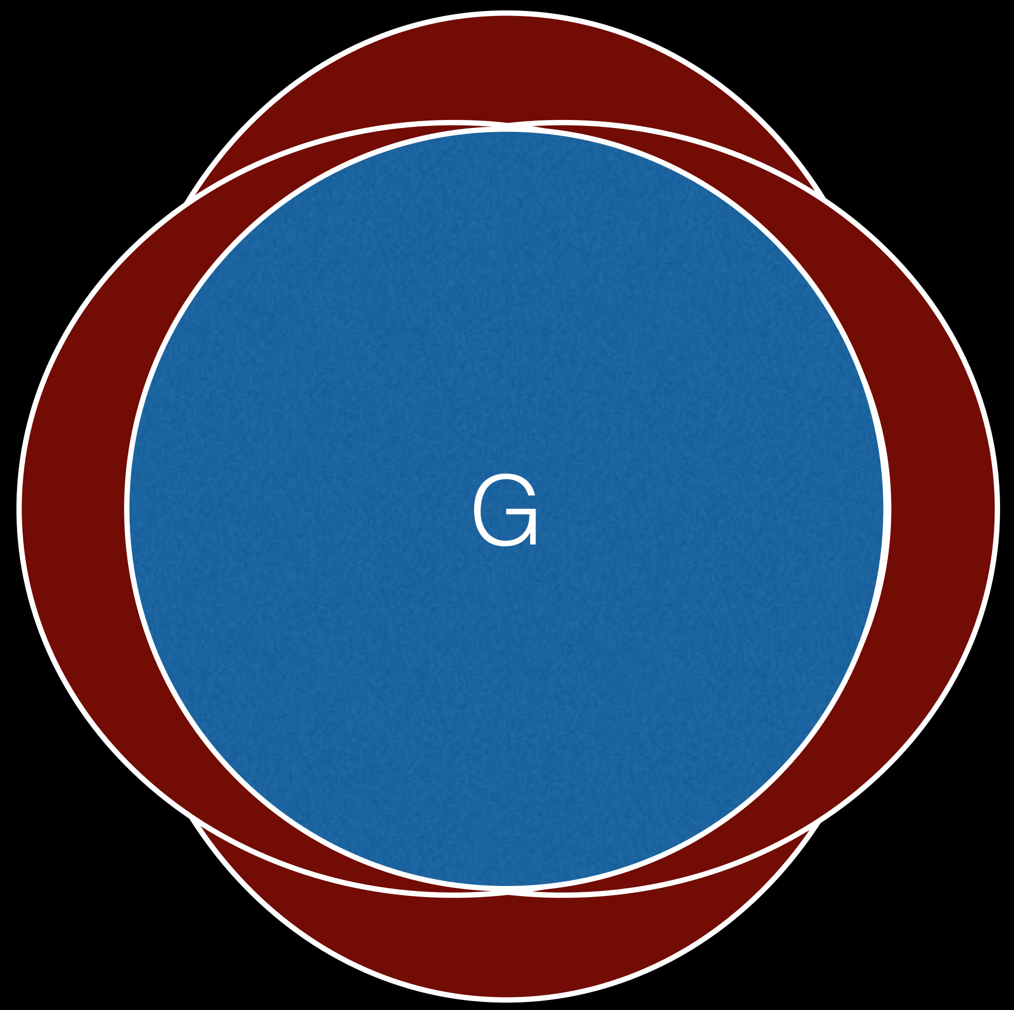


Good ID is *light* if received ~np messages

Light IDs send info about S_x sets to IDs in S_x

Can't send all IDs in S_x; Instead send a sample

Need |B|/|G| < 1/2, unless adversary sends $\Omega(n^2 p)$ messages



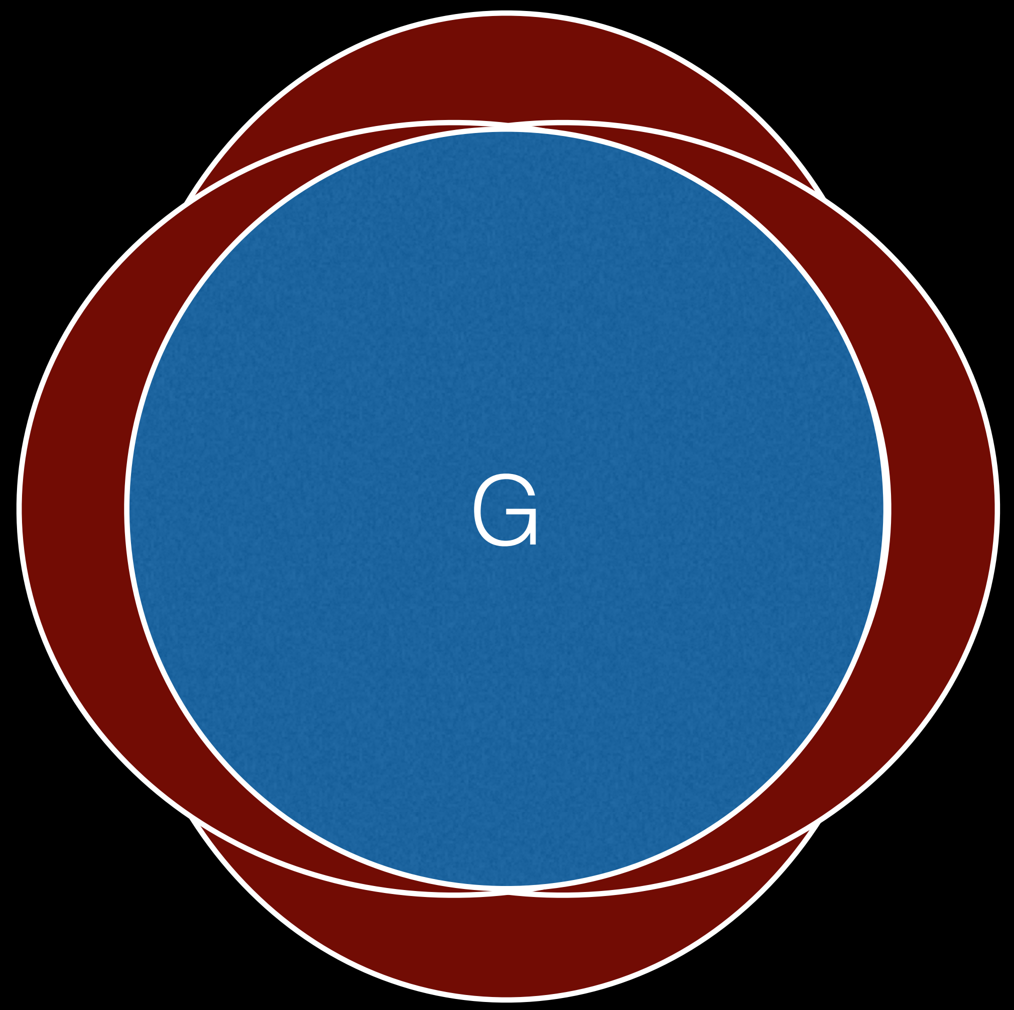
Good ID is *light* if received ~np messages

Light IDs send info about S_x sets to IDs in S_x

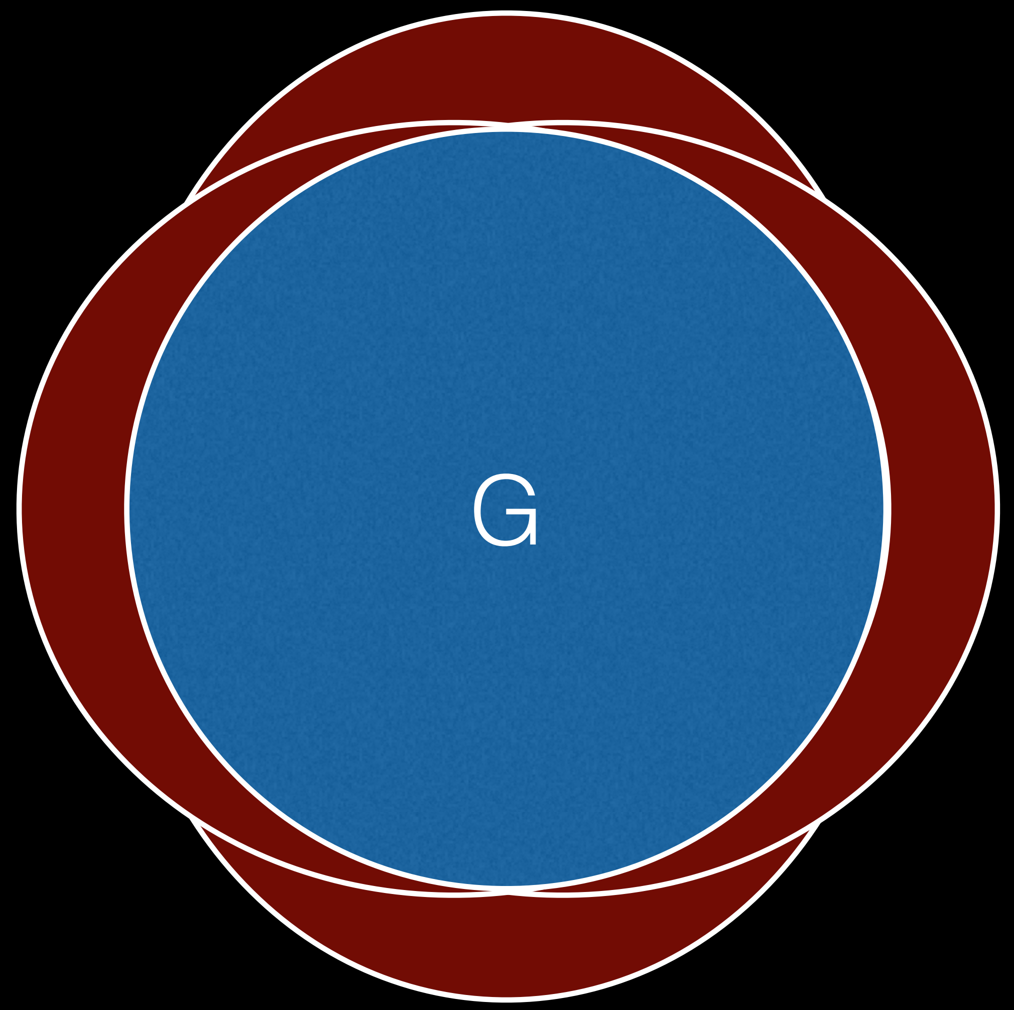
Can't send all IDs in S_x; Instead send a sample

Problem: Adversary can still easily make set B (all bad nodes in S_x sets) too large

Need |B|/|G| < 1/2, unless adversary sends $\Omega(n^2 p)$ messages

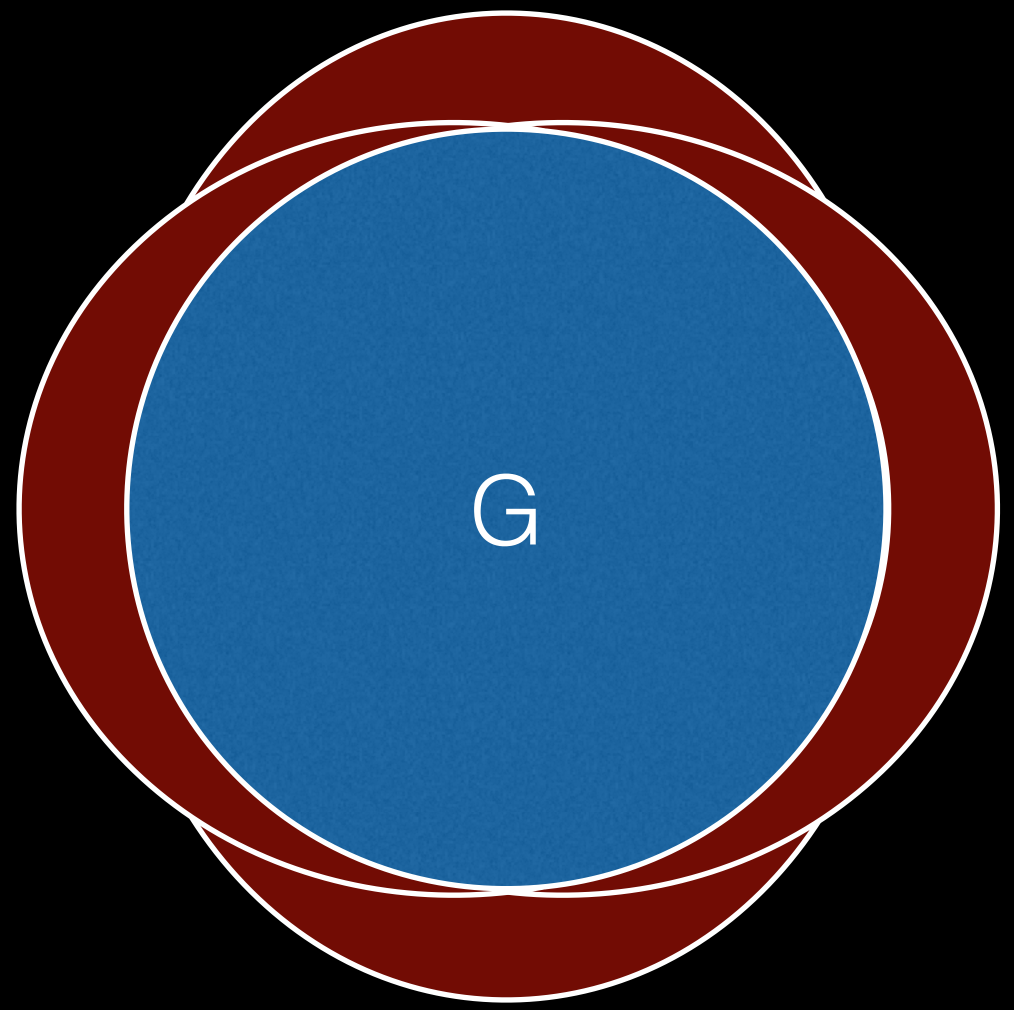


Need |B|/|G| < 1/2, unless adversary sends $\Omega(n^2 p)$ messages



Problem: Adversary can still easily make set B (all bad nodes in S_x sets) too large

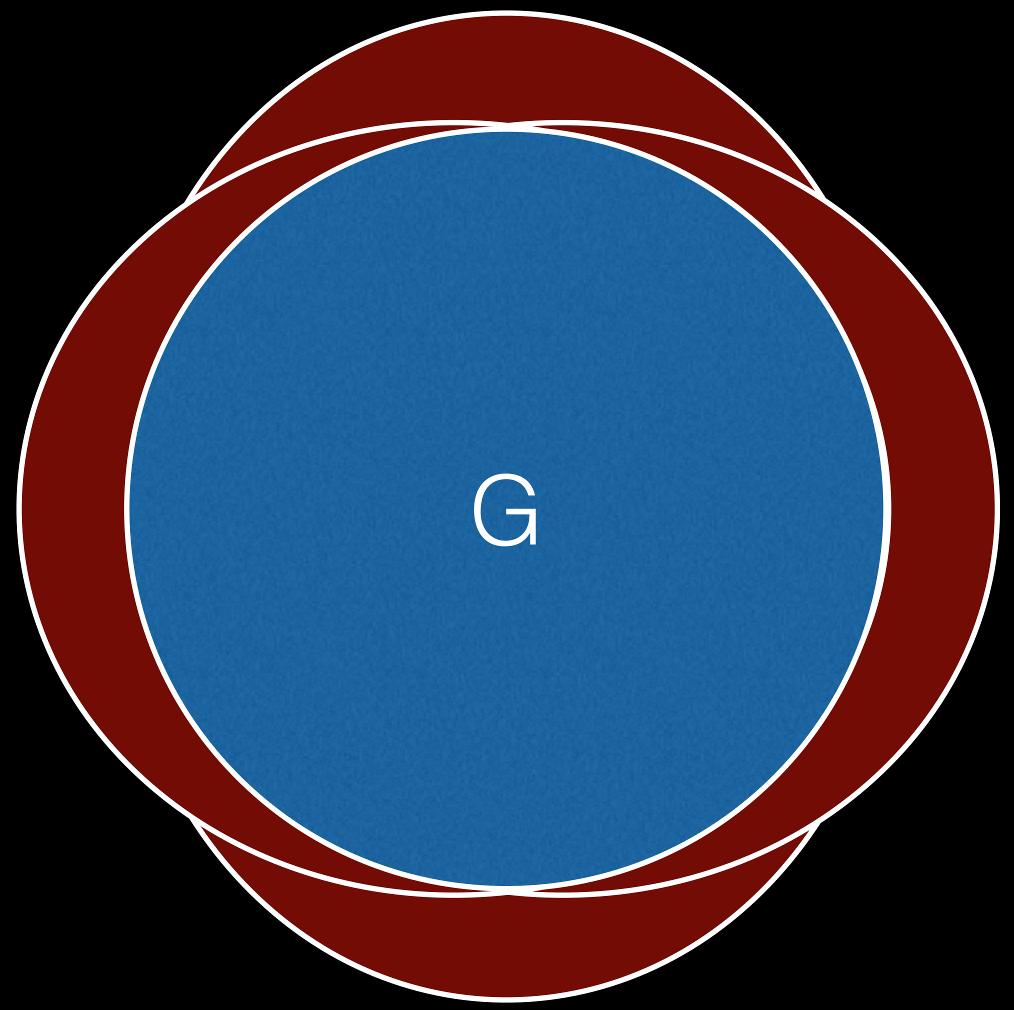
Need |B|/|G| < 1/2, unless adversary sends $\Omega(n^2p)$ messages



Problem: Adversary can still easily make set B (all bad nodes in S_x sets) too large

Validation step: each active ID x, for each $y \in S_x$ queries $\Theta(\log n)$ random IDs to see if they have y in their own S set

Need |B|/|G| < 1/2, unless adversary sends $\Omega(n^2 p)$ messages



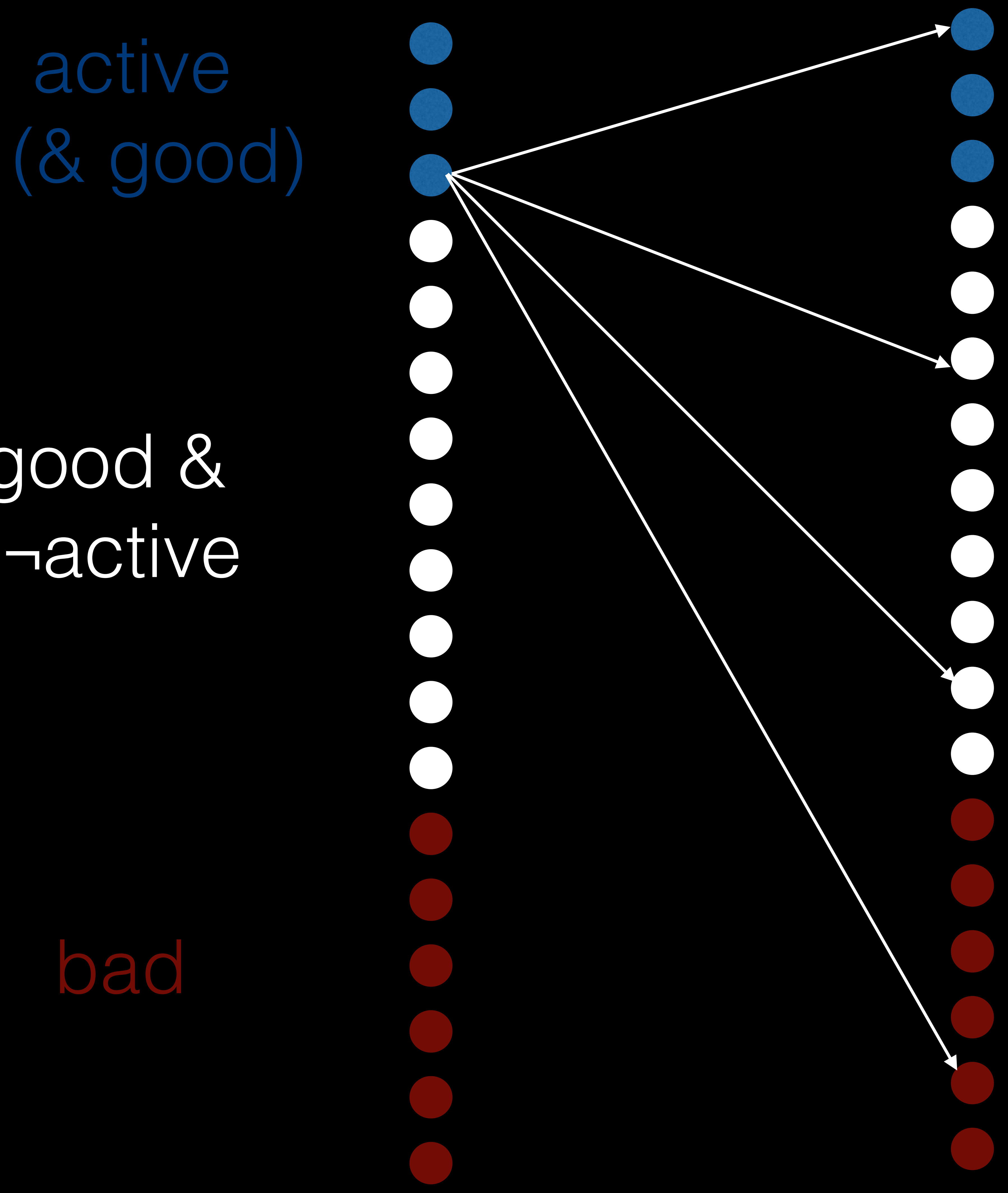
Problem: Adversary can still easily make set B (all bad nodes in S_x sets) too large

Validation step: each active ID x, for each $y \in S_x$ queries $\Theta(\log n)$ random IDs to see if they have y in their own S set

Filter out IDs from S_x that are not in enough samples

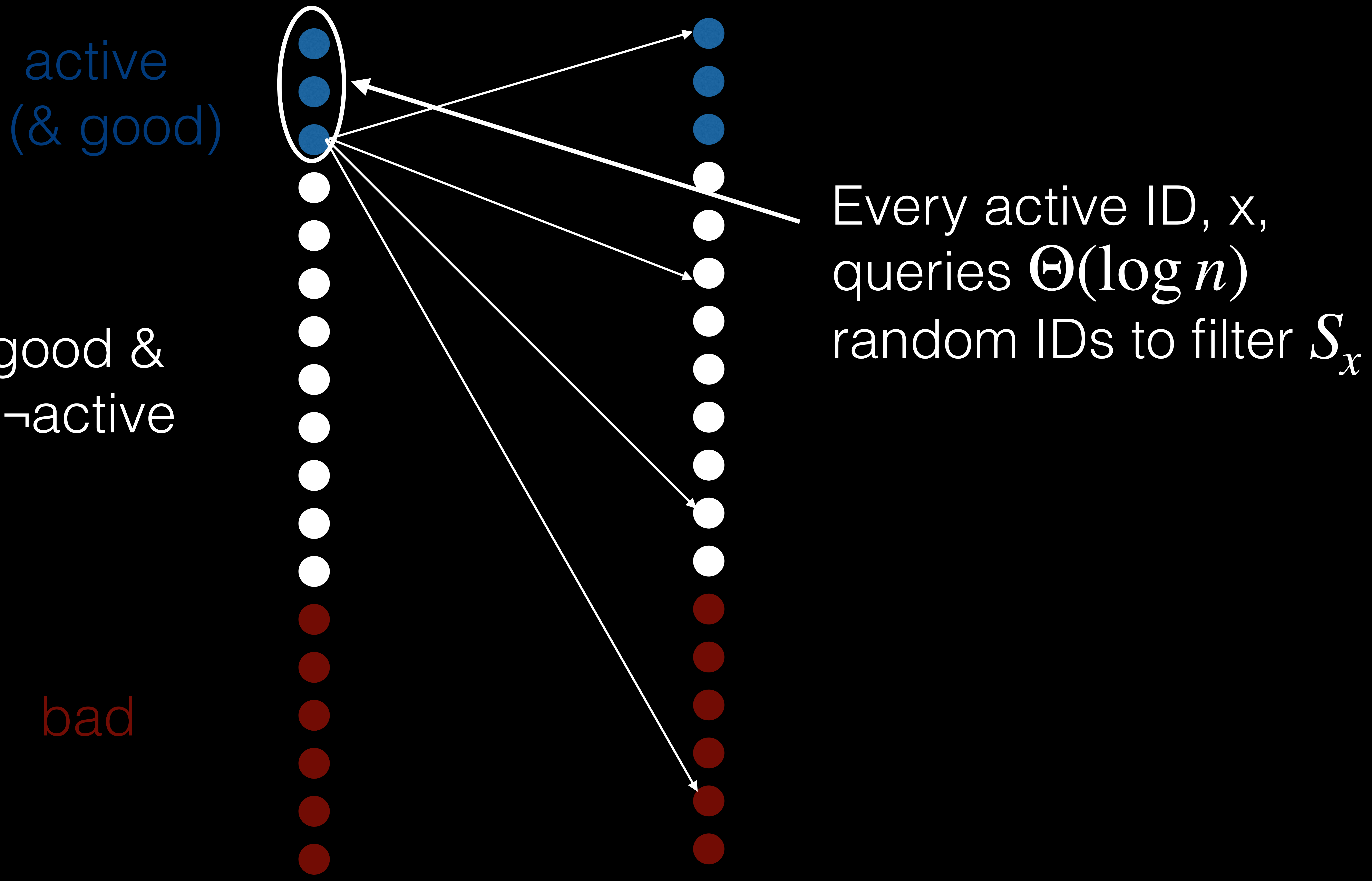
Validation step: each active ID x, for each $y \in S_x$ queries $\Theta(\log n)$ random IDs to see if they have y in their own S set

Filter out IDs from S_x that not in enough samples



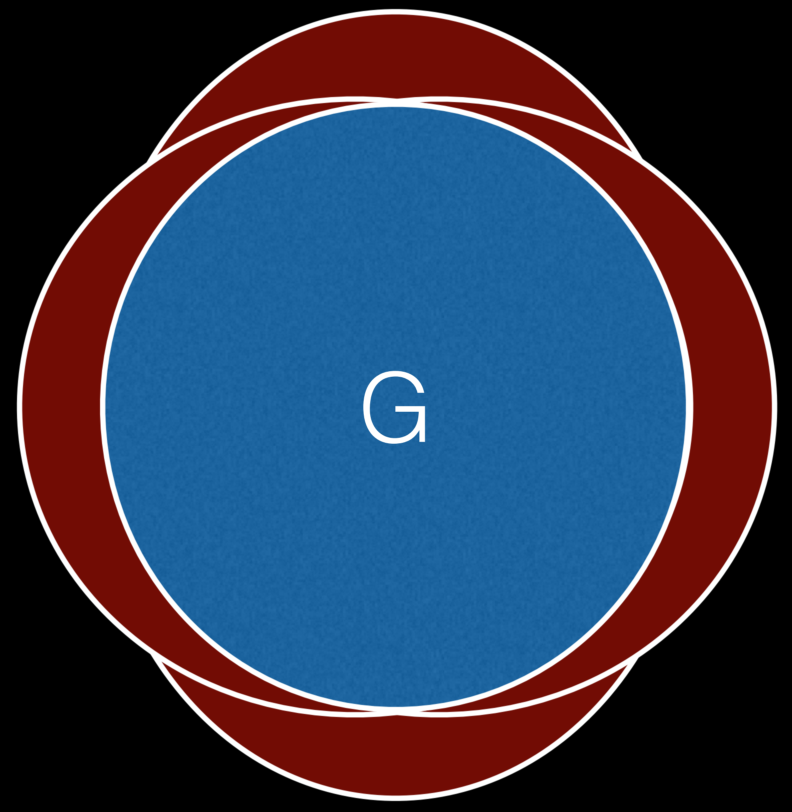
Validation step: each active ID x, for each $y \in S_x$ queries $\Theta(\log n)$ random IDs to see if they have y in their own S set

Filter out IDs from S_x that not in enough samples



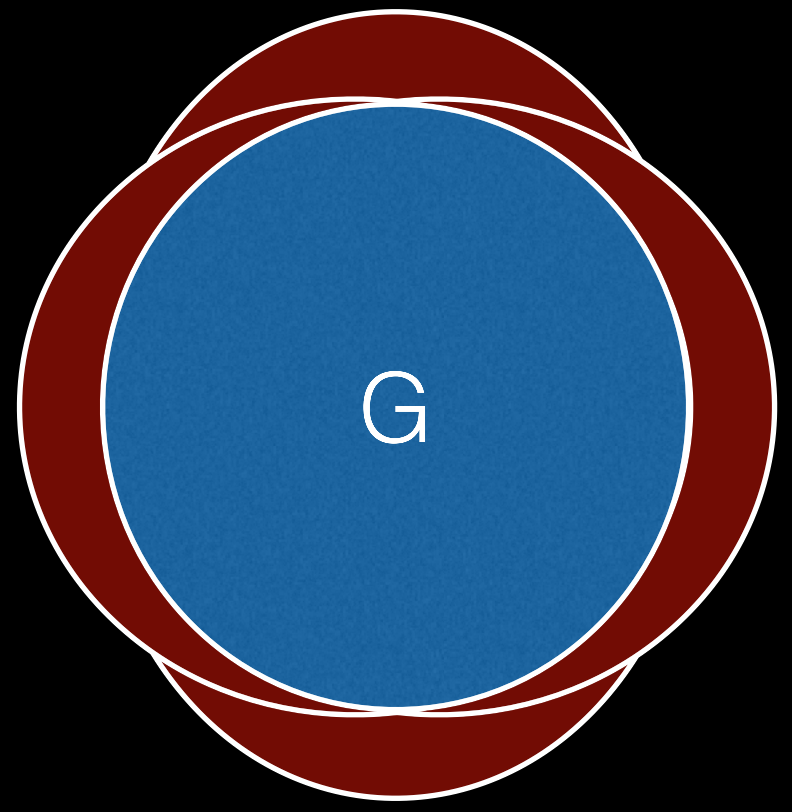
Problem 2: Ready? - How do active IDs agree on whether conditions favorable for agreement?

Problem 2: Ready?

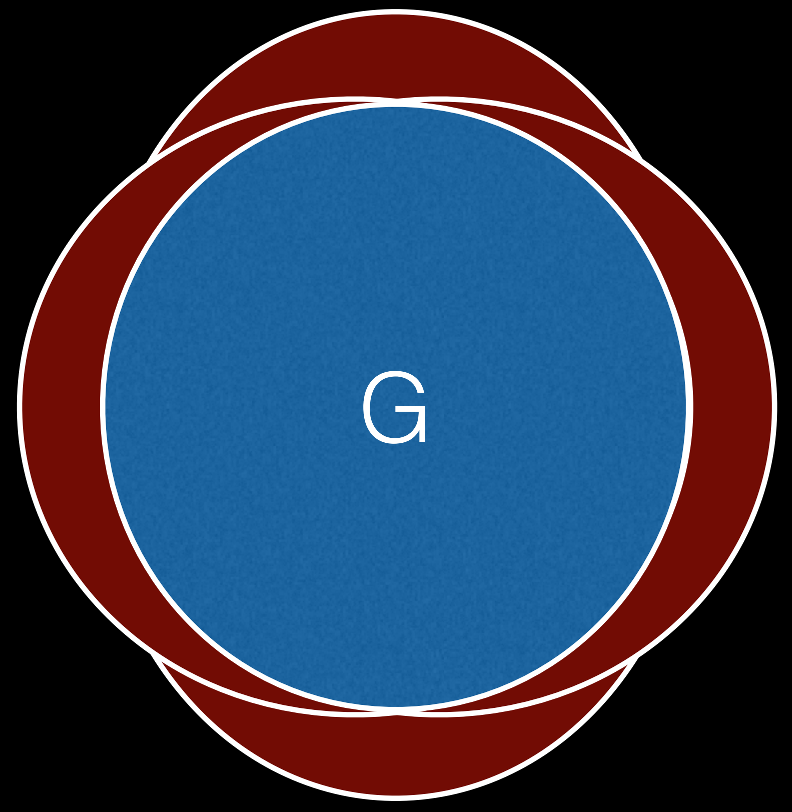


Problem 2: Ready?

Naive: use LCBA to determine Ready?



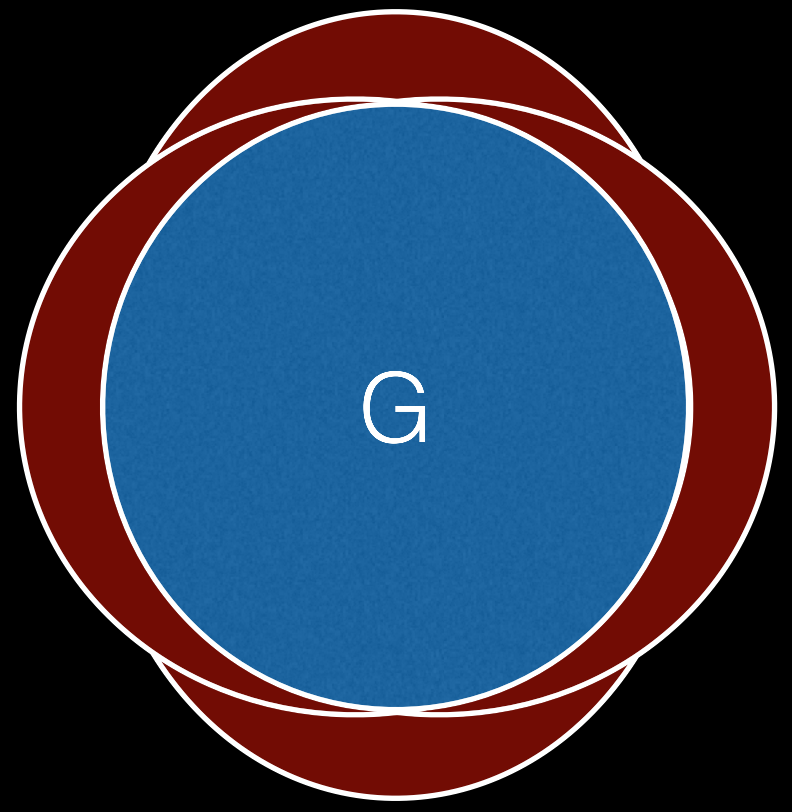
Problem 2: Ready?



Naive: use LCBA to determine Ready?

Problem: Some active IDs run LCBA, but others don't have small enough S_x to run it

Problem 2: Ready?

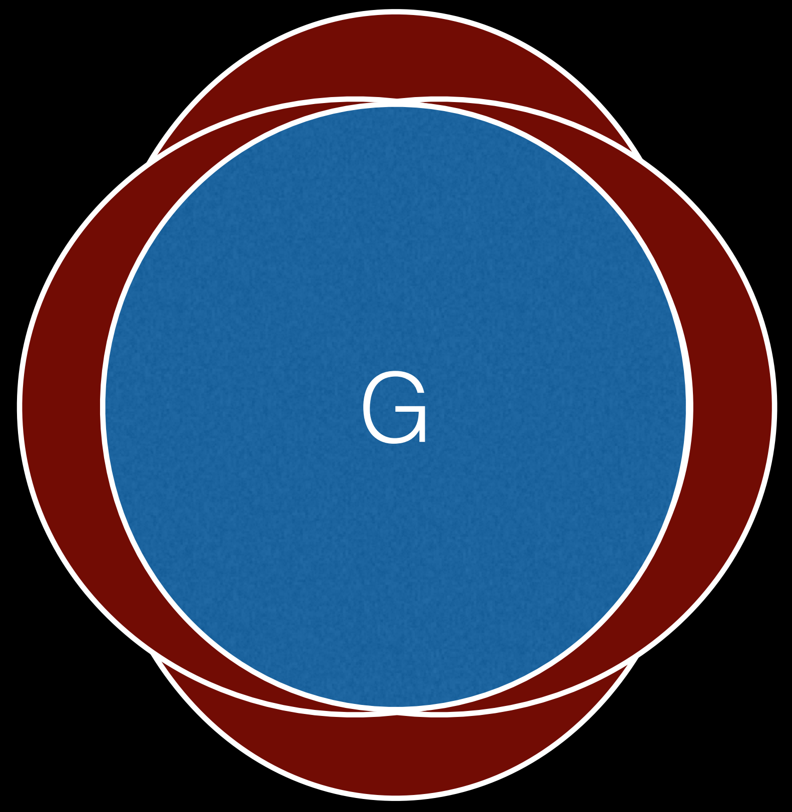


Naive: use LCBA to determine Ready?

Problem: Some active IDs run LCBA, but others don't have small enough S_x to run it

Solution: Need careful decisions about: (1) input; (2) whether will run LCBA; (3) whether will trust LCBA

Problem 2: Ready?



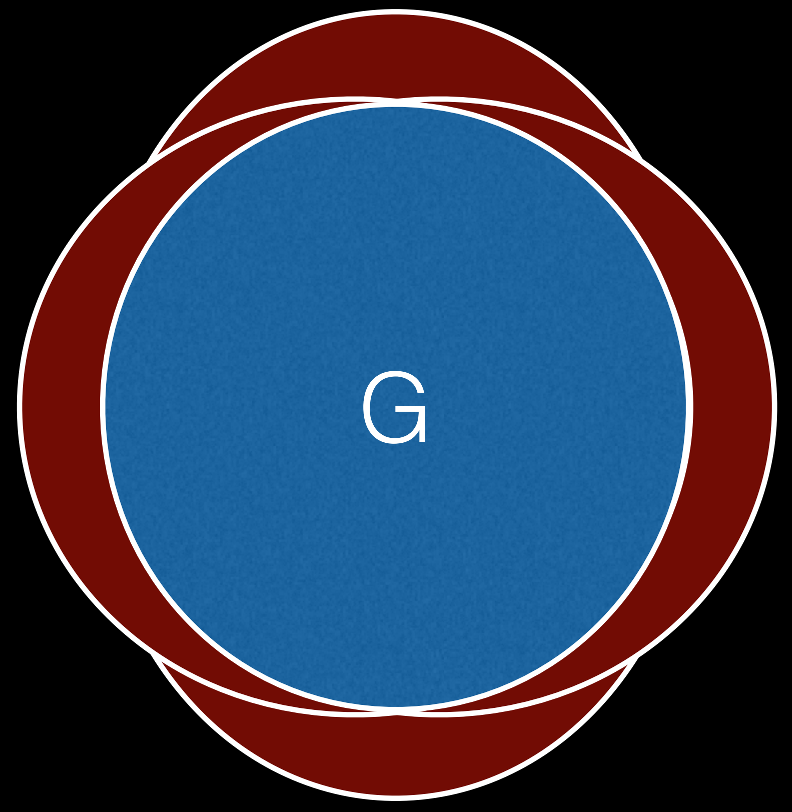
Naive: use LCBA to determine Ready?

Problem: Some active IDs run LCBA, but others don't have small enough S_x to run it

Solution: Need careful decisions about: (1) input; (2) whether will run LCBA; (3) whether will trust LCBA

IDs sometimes run LCBA, even when they plan to ignore output, since other IDs counting on them

Problem 2: Ready?



Naive: use LCBA to determine Ready?

Problem: Some active IDs run LCBA, but others don't have small enough S_x to run it

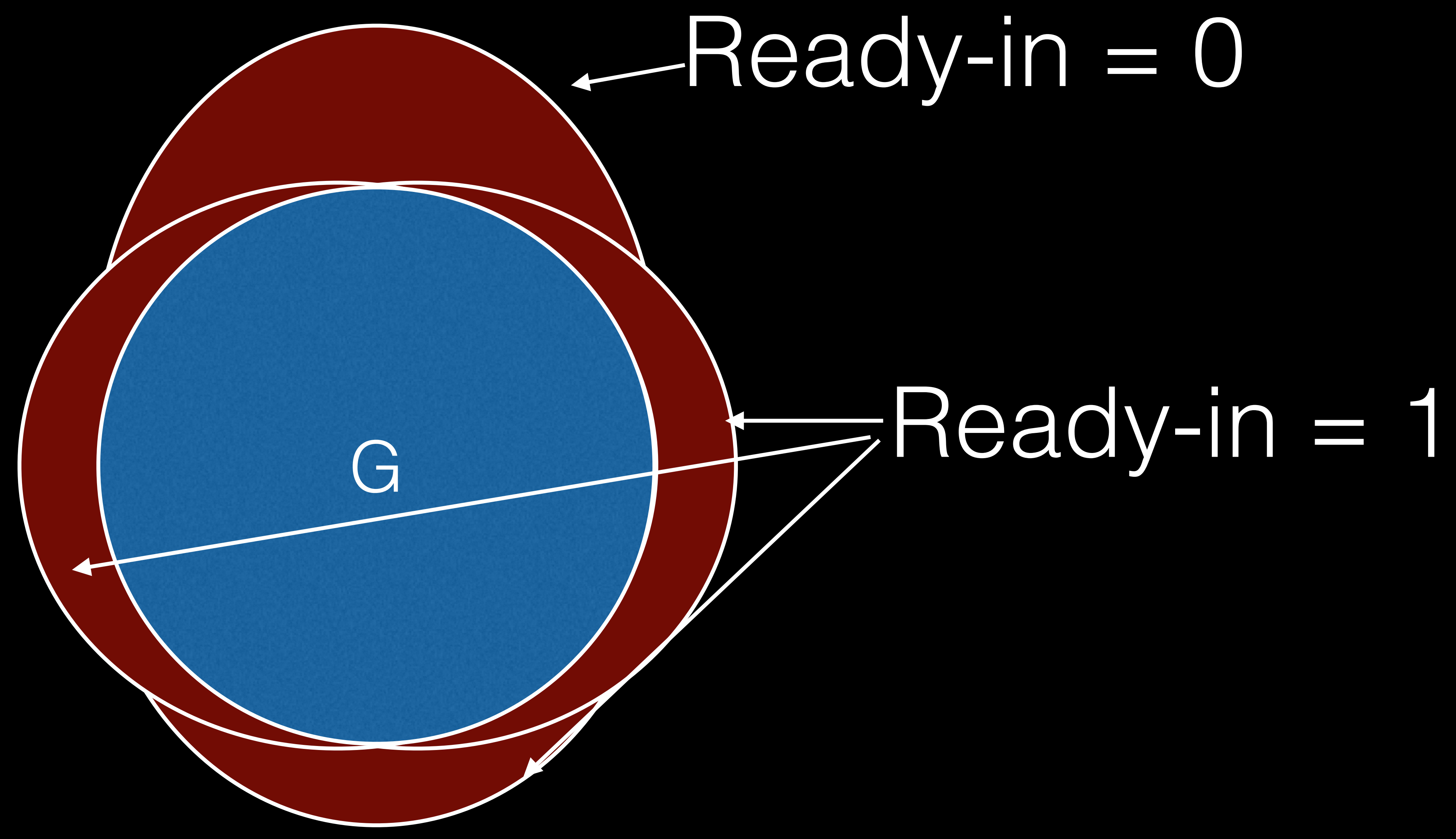
Solution: Need careful decisions about: (1) input; (2) whether will run LCBA; (3) whether will trust LCBA

IDs sometimes run LCBA, even when they plan to ignore output, since other IDs counting on them

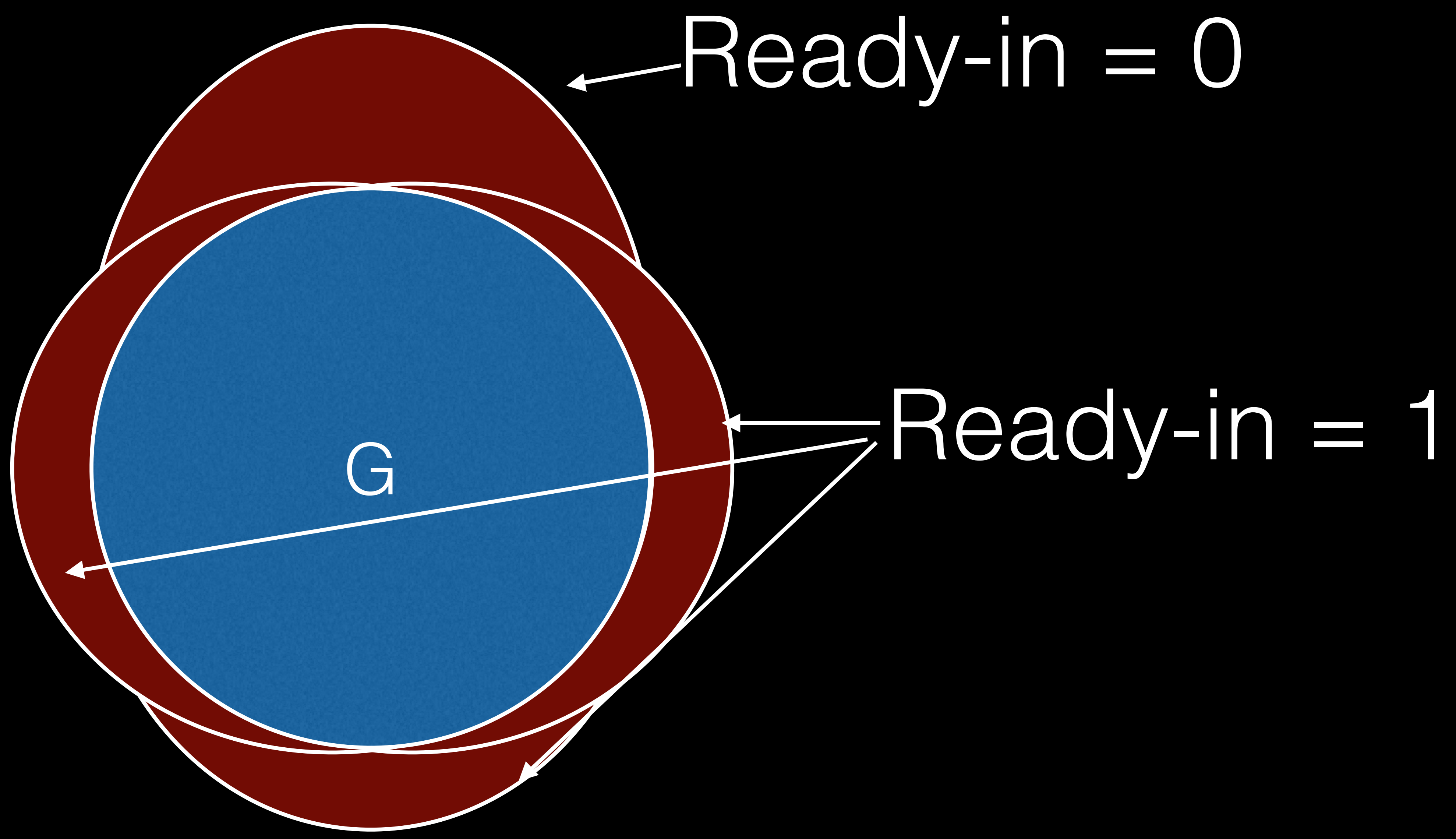
If "Ready?" output = yes, IDs run LCBA again for agreement

All active IDs then broadcast Ready? bit and Agreement bit

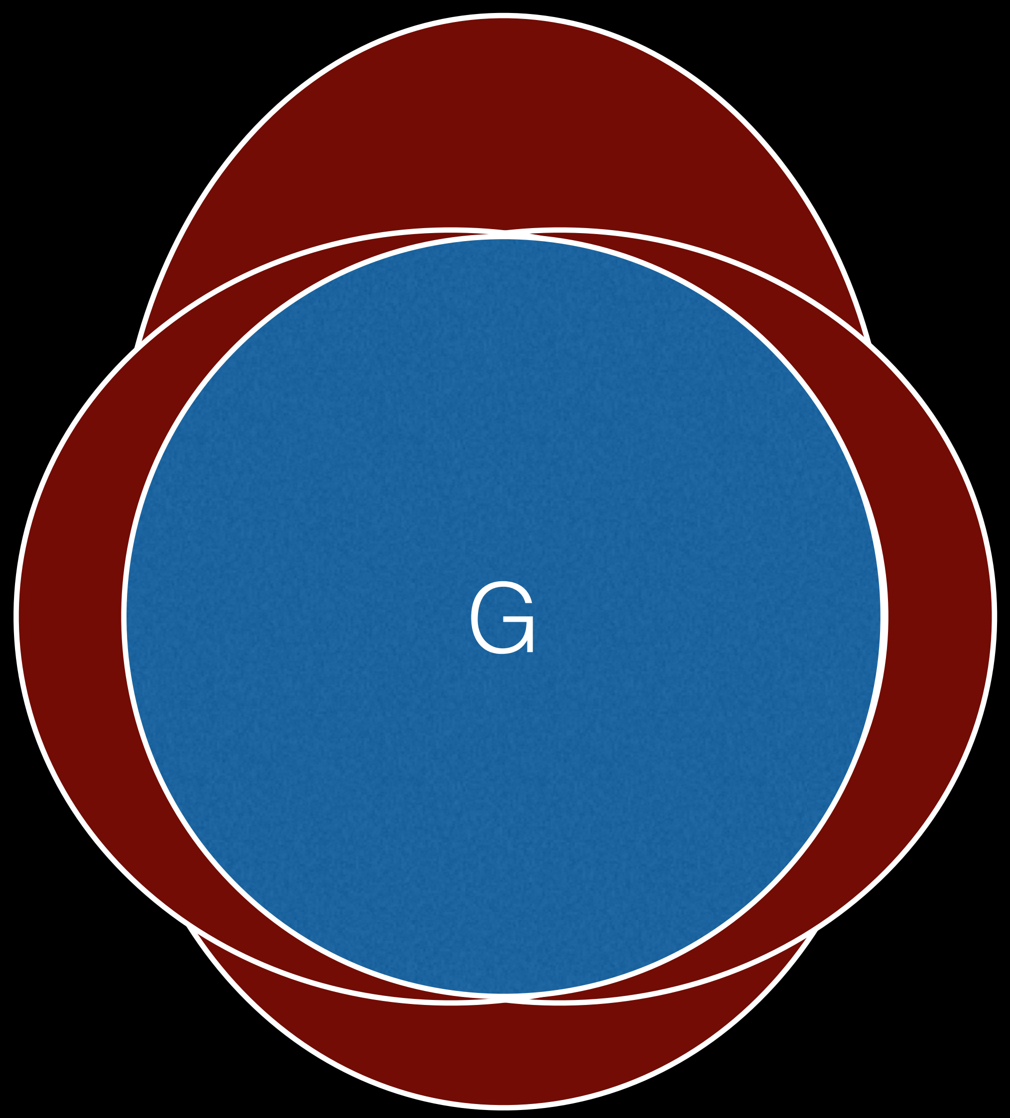
LCBA for
“Ready-out” bit



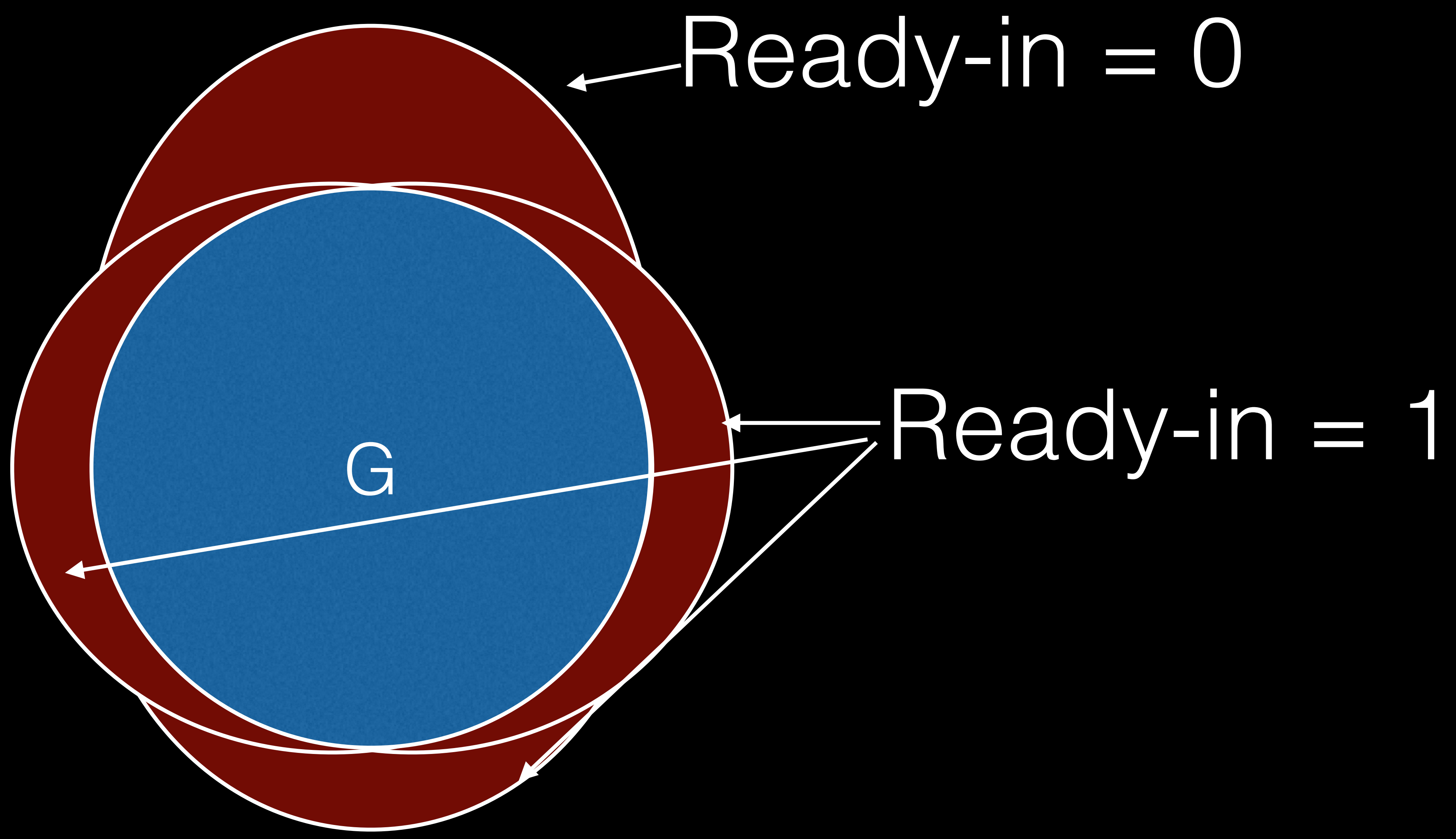
LCBA for
“Ready-out” bit



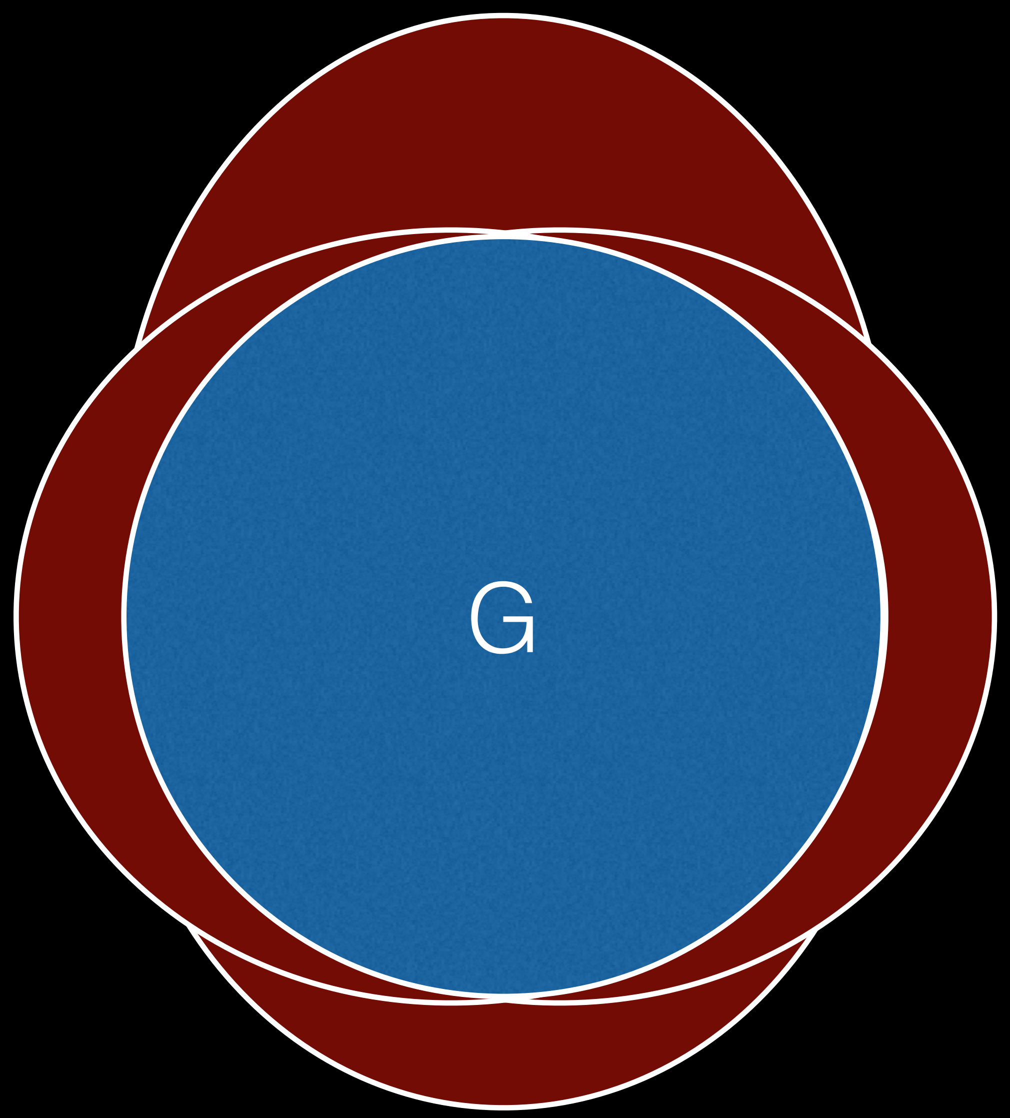
LCBA for agreement
“value” bit



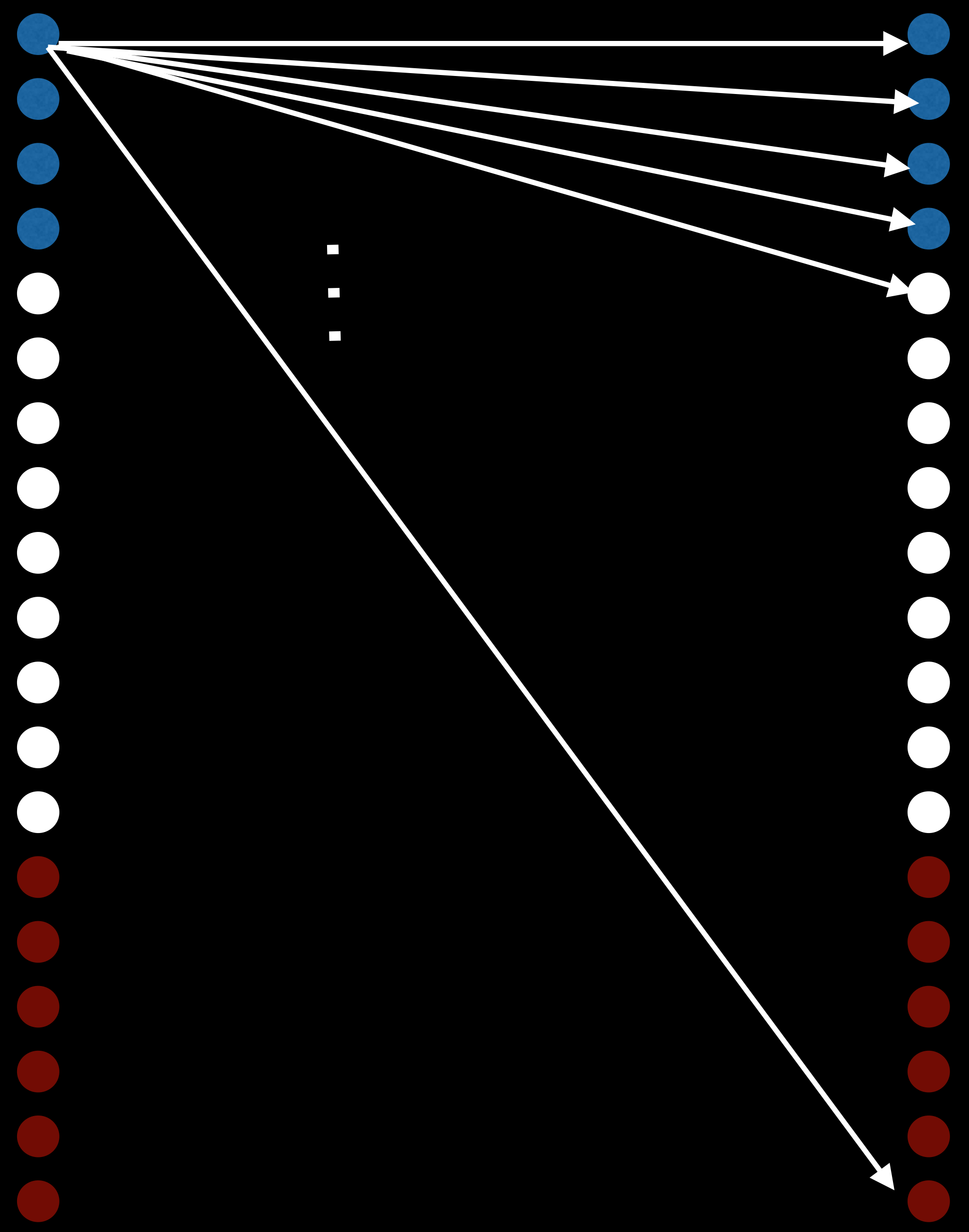
LCBA for
“Ready-out” bit



LCBA for agreement
“value” bit

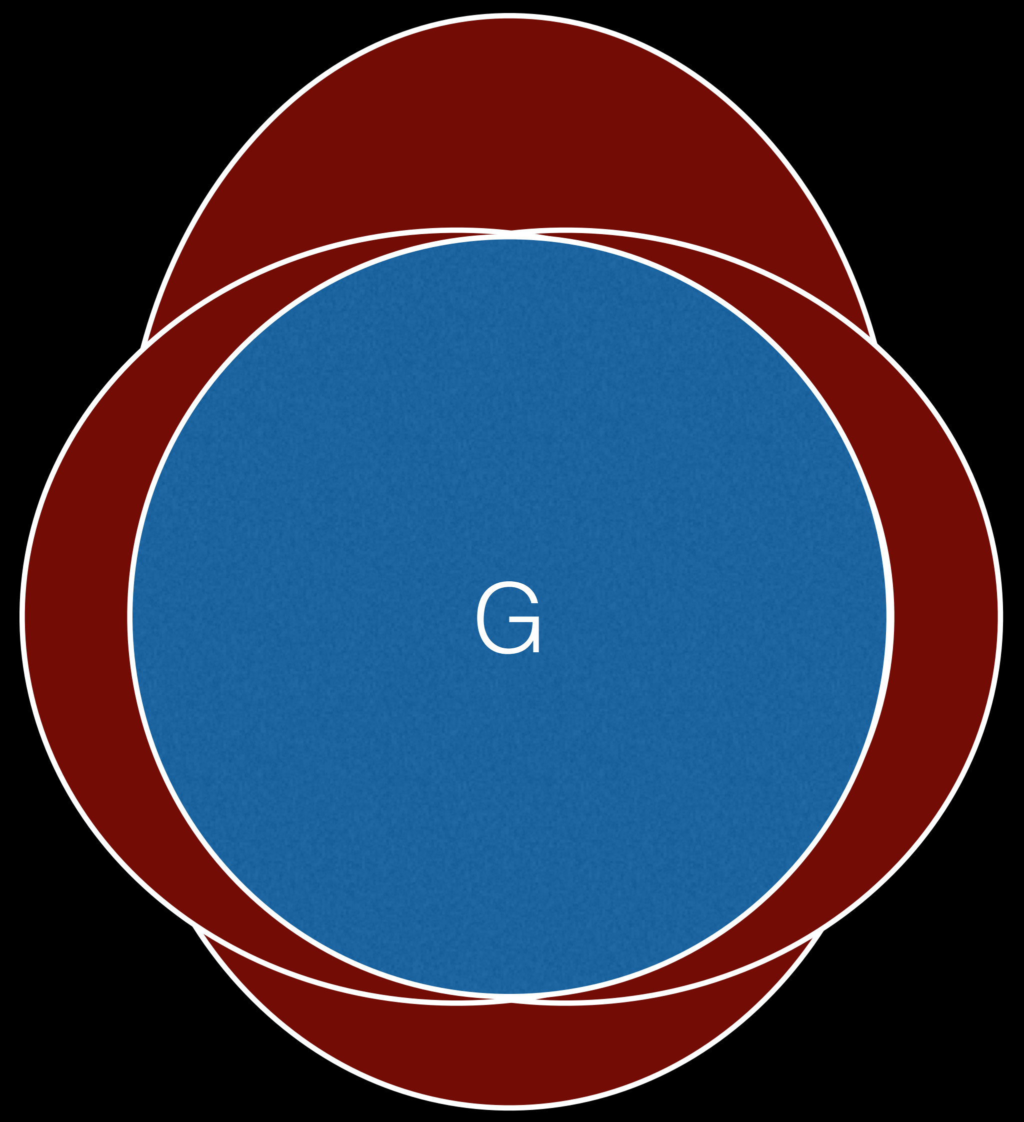


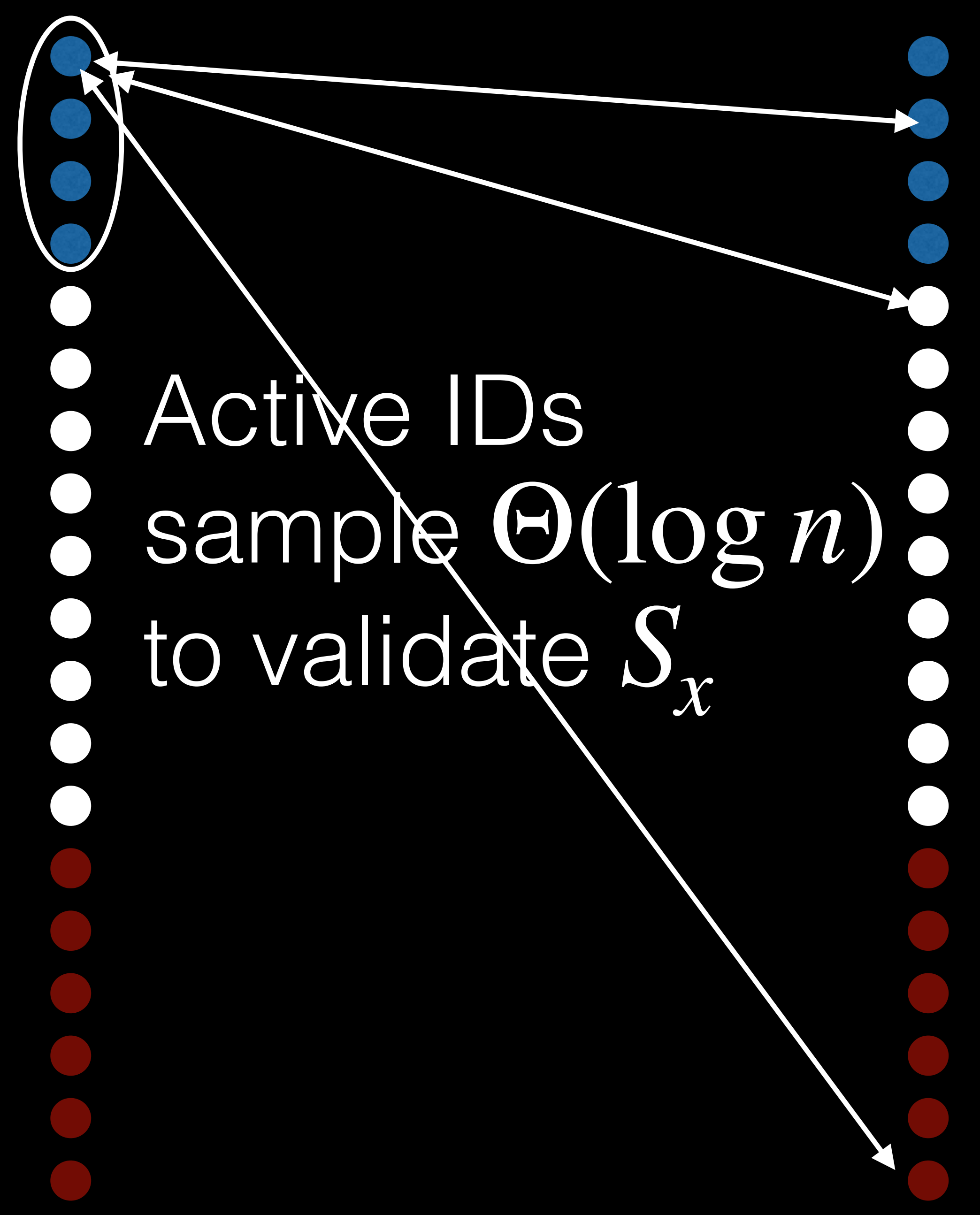
Every active ID
broadcasts its
(ready-out,value)
bits



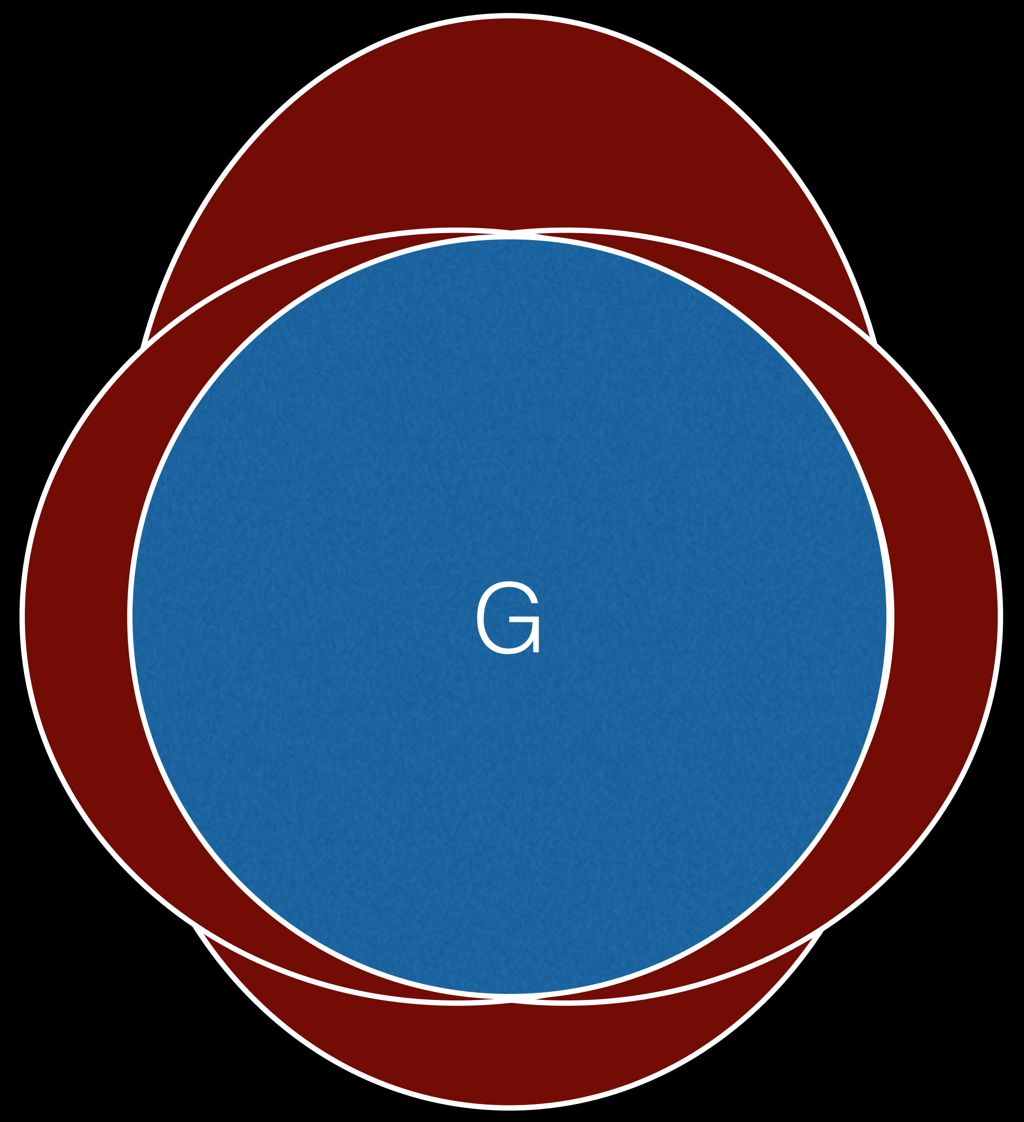

Active IDs
sample $\Theta(\log n)$
to validate S_x

LCBA for “Ready-out” bit

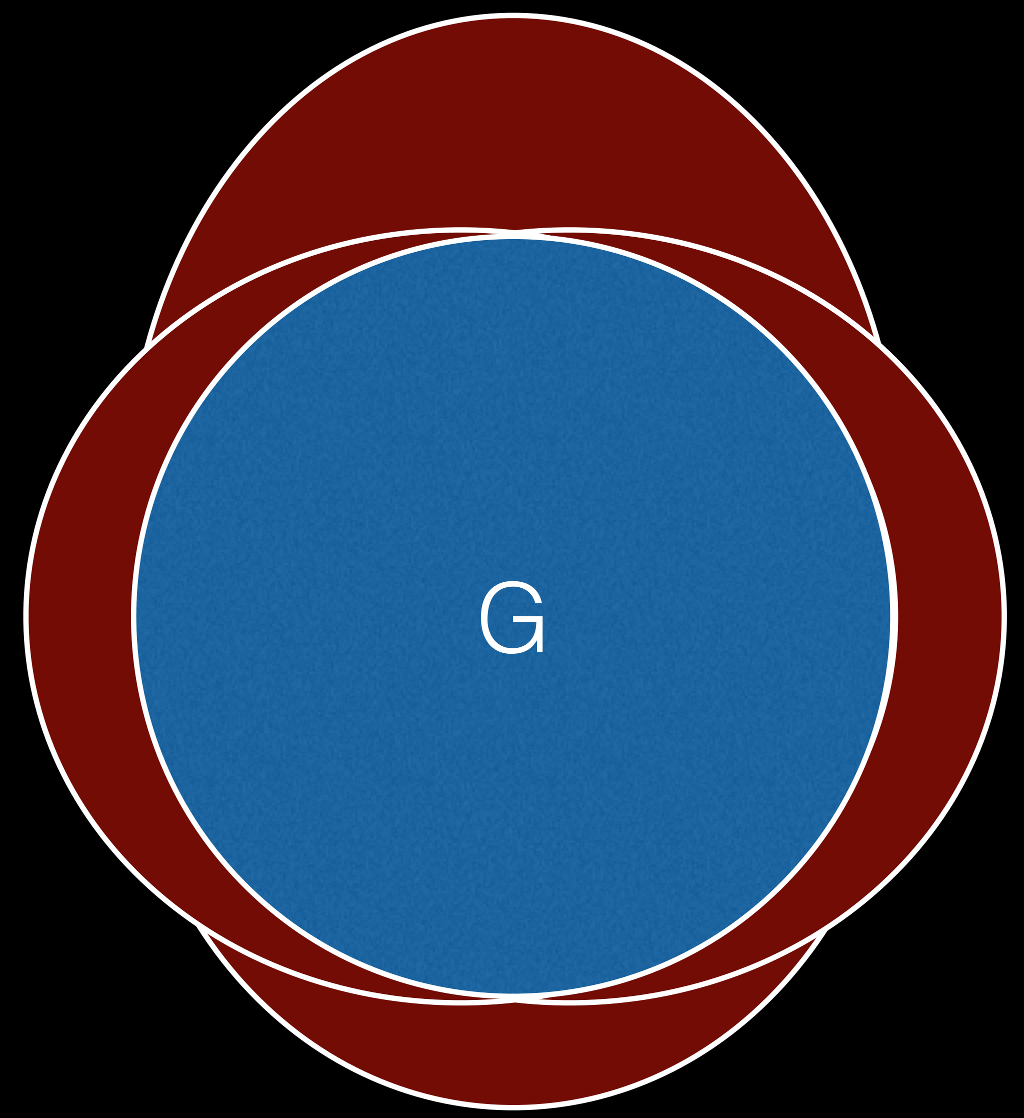


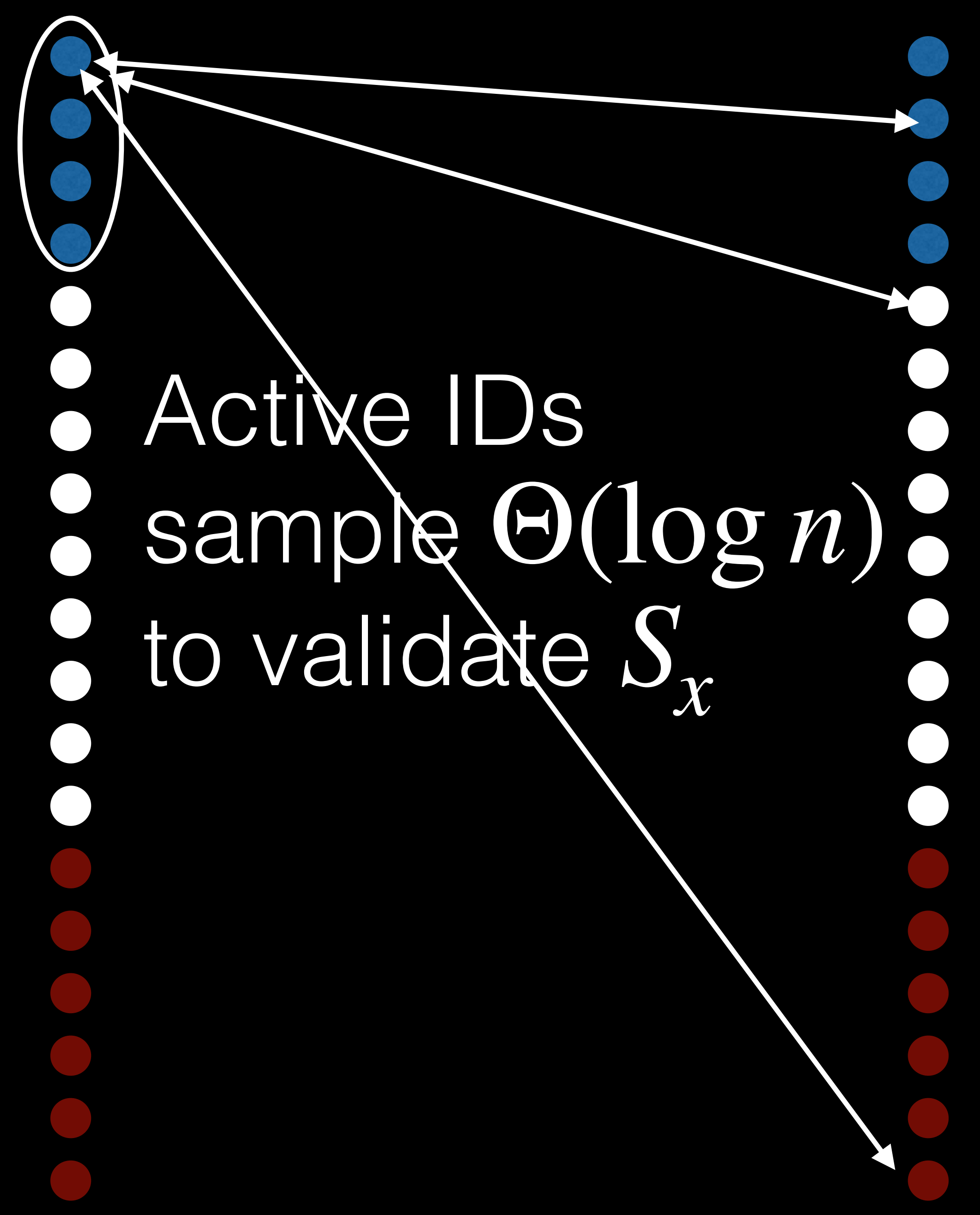


LCBA for “Ready-out” bit

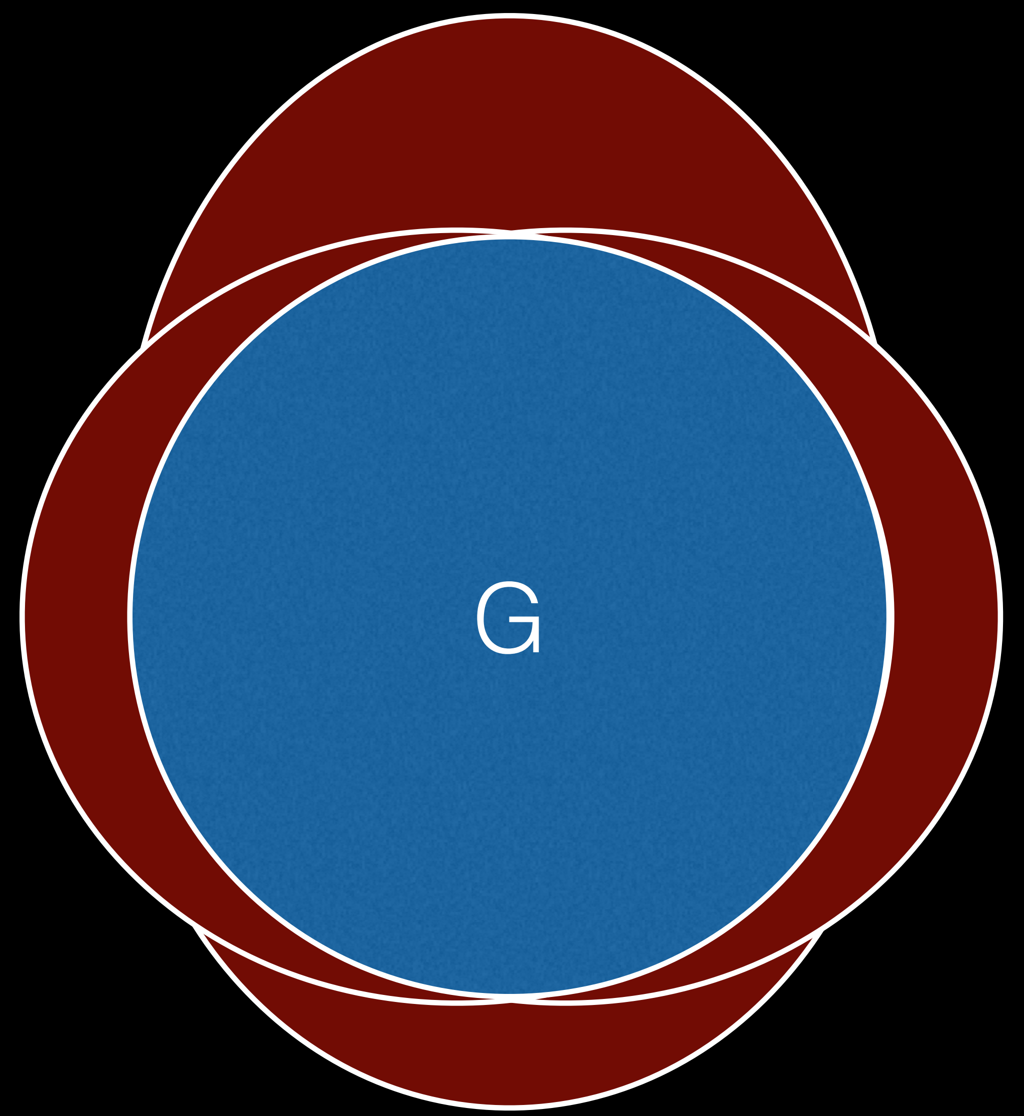


LCBA for “value” bit

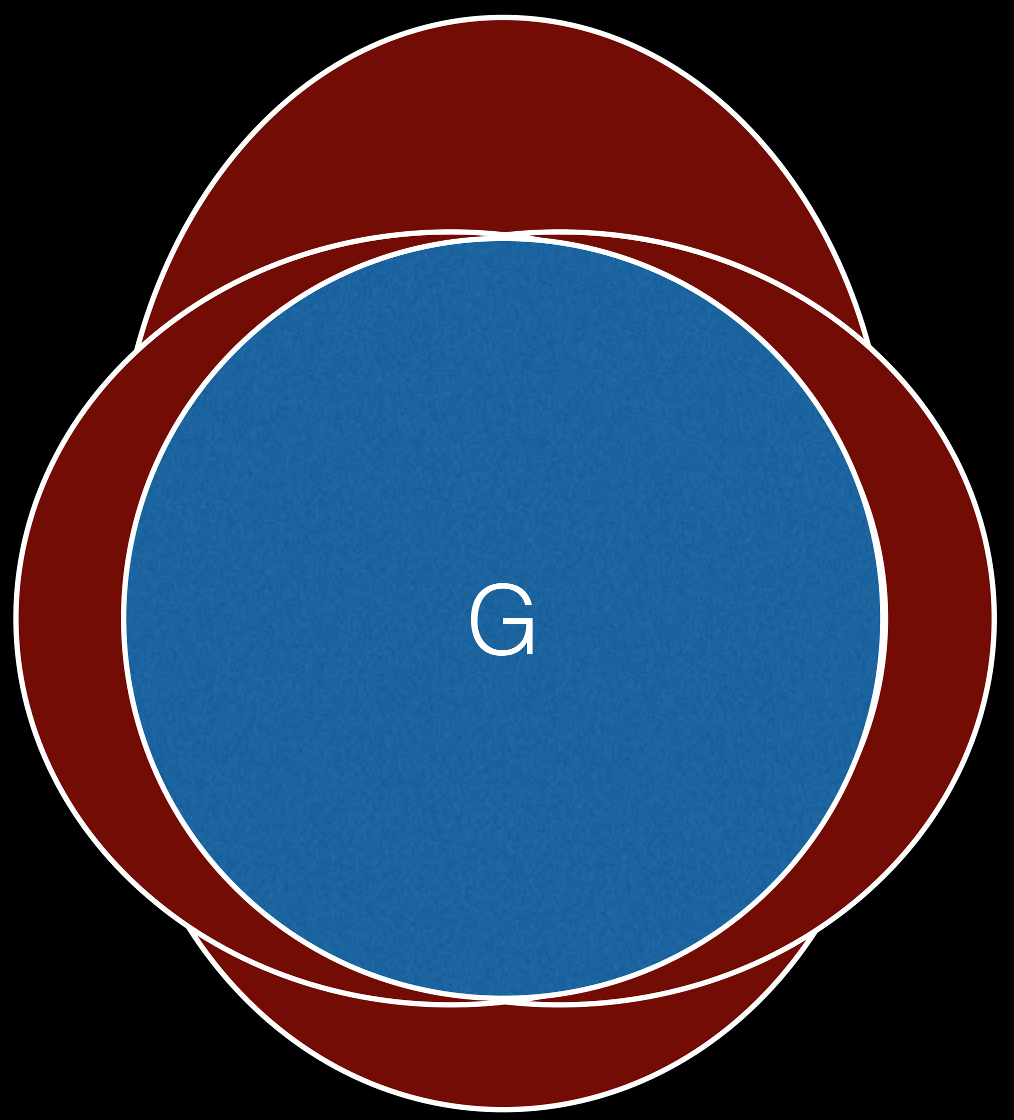


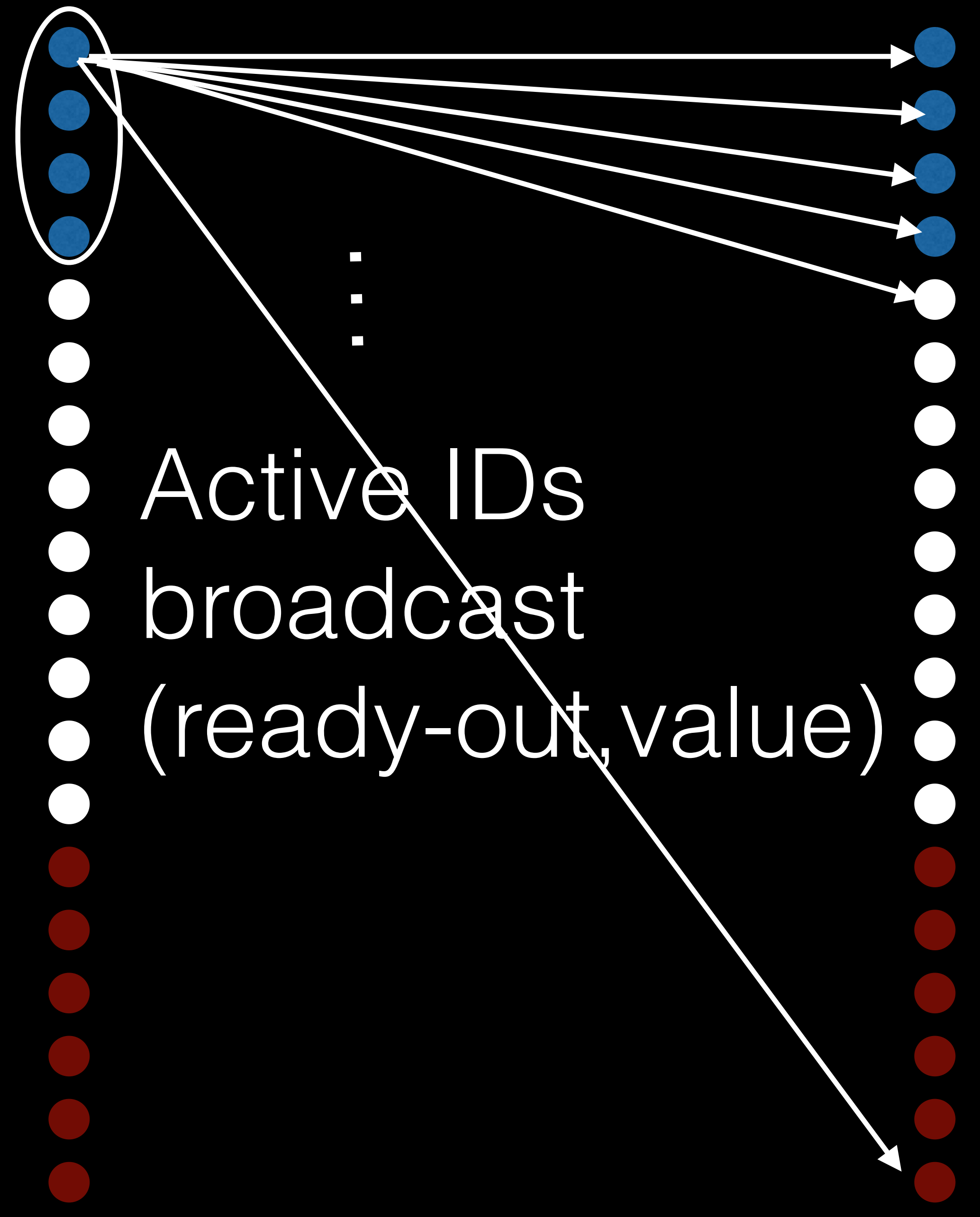


LCBA for “Ready-out” bit



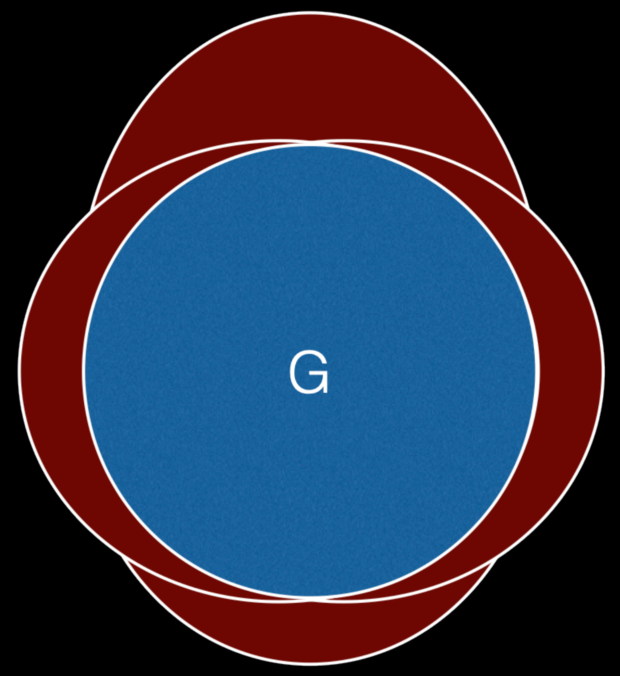
LCBA for “value” bit



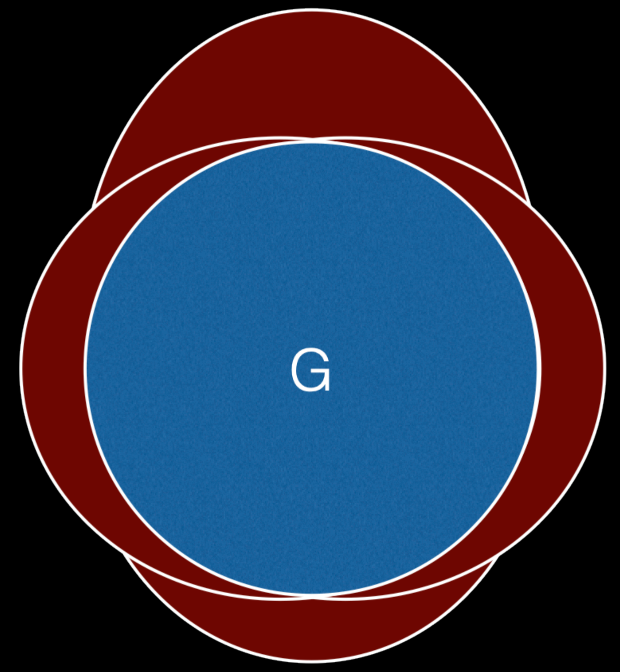


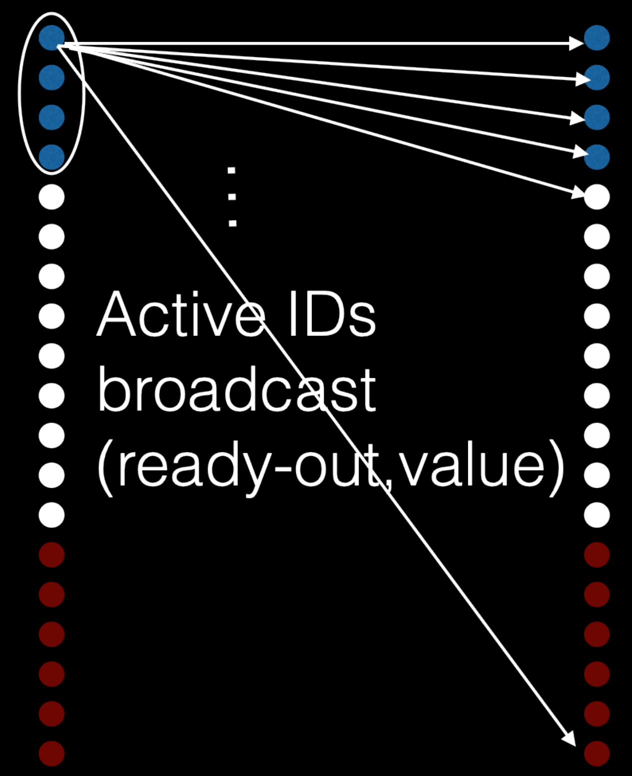
Implicit Agreement

LCBA for “Ready-out” bit



LCBA for “value” bit



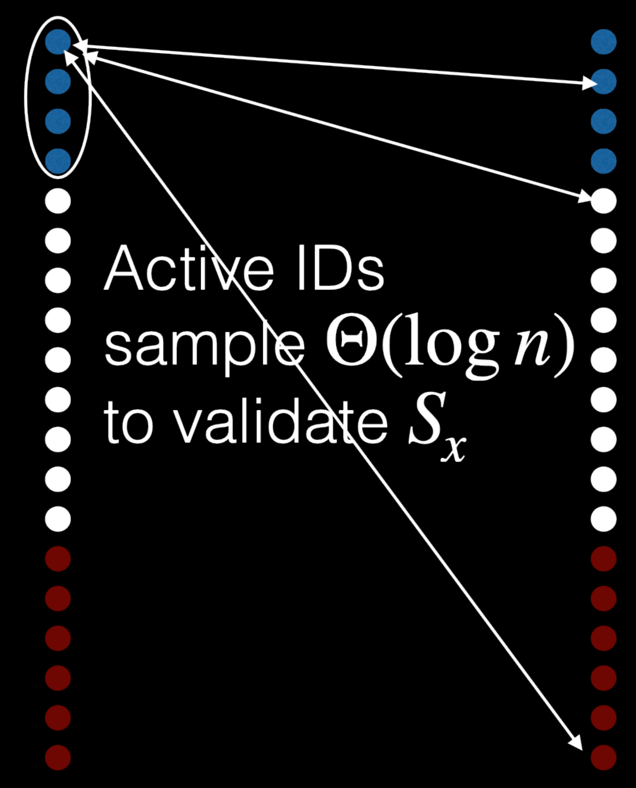


Either:

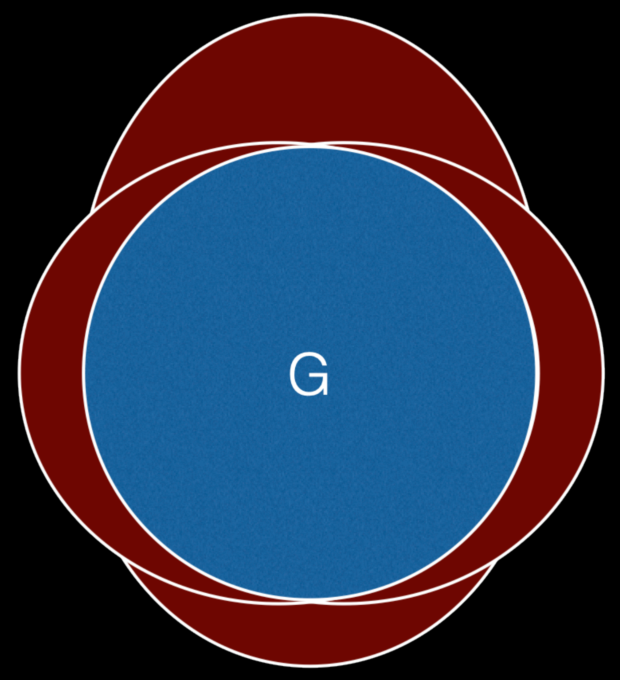
(1) > t/n good IDs decide; or

(2) no good IDs decide

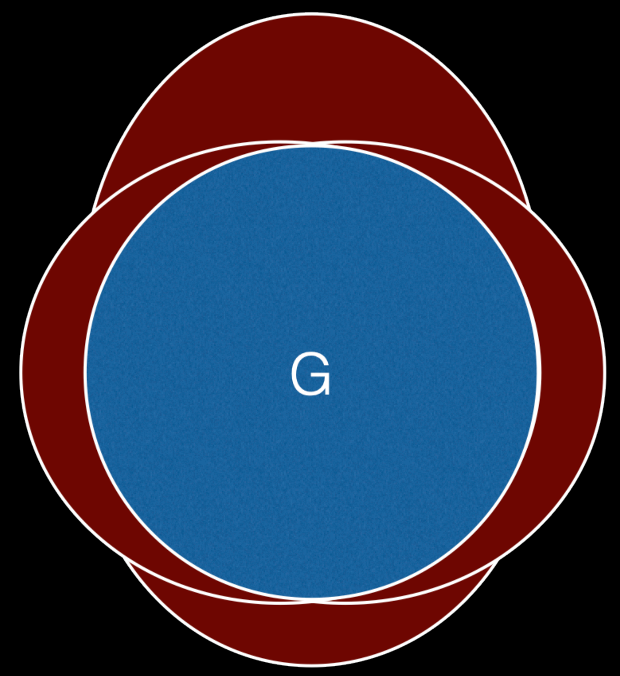
Implicit Agreement

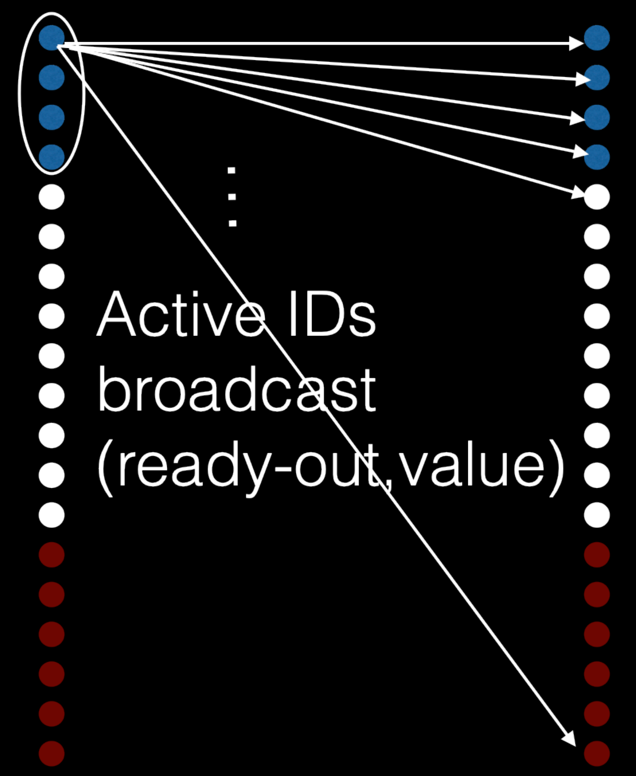


LCBA for “Ready-out” bit



LCBA for “value” bit





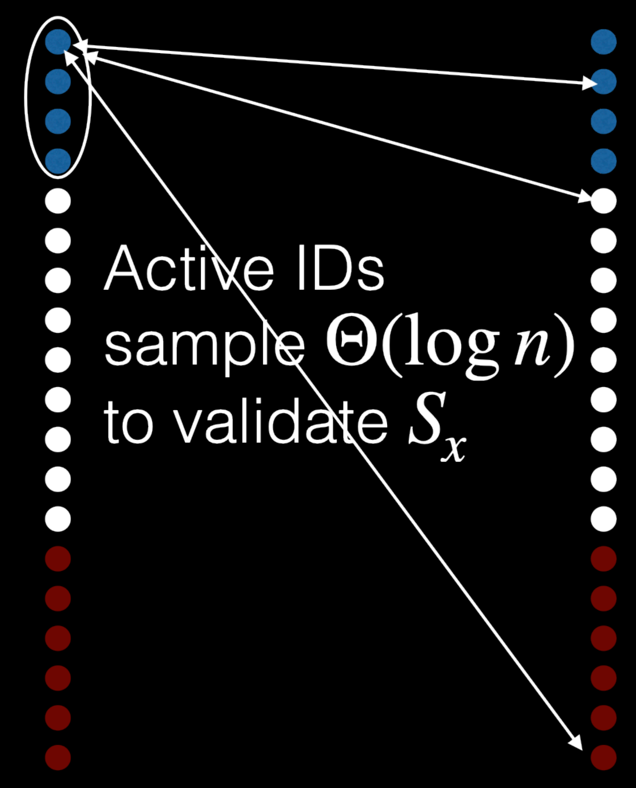
What next?

Either:

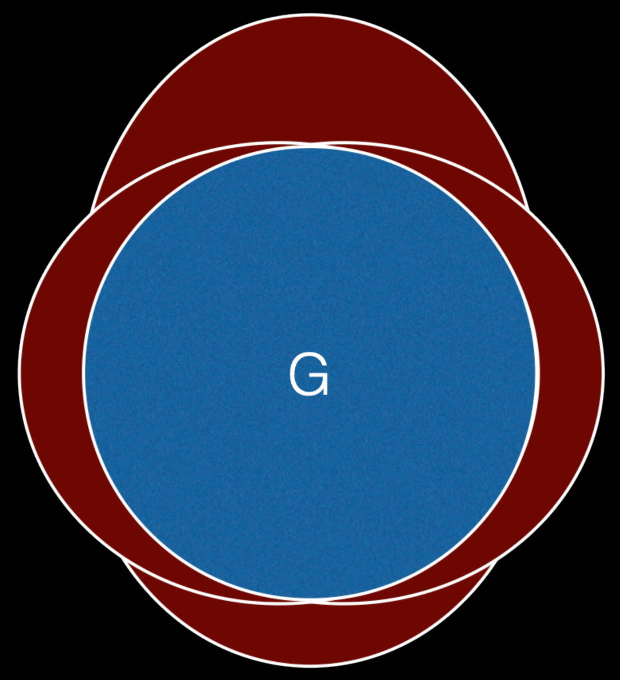
(1) > t/n good IDs decide; or

(2) no good IDs decide

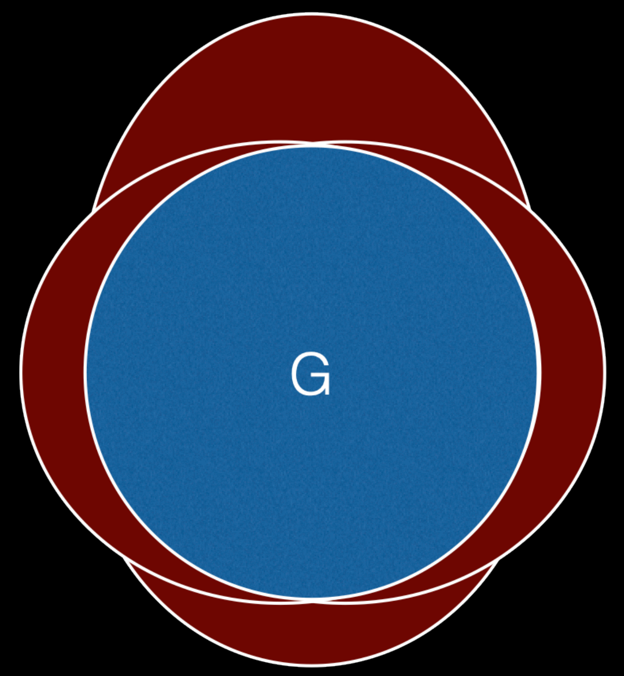
Implicit Agreement

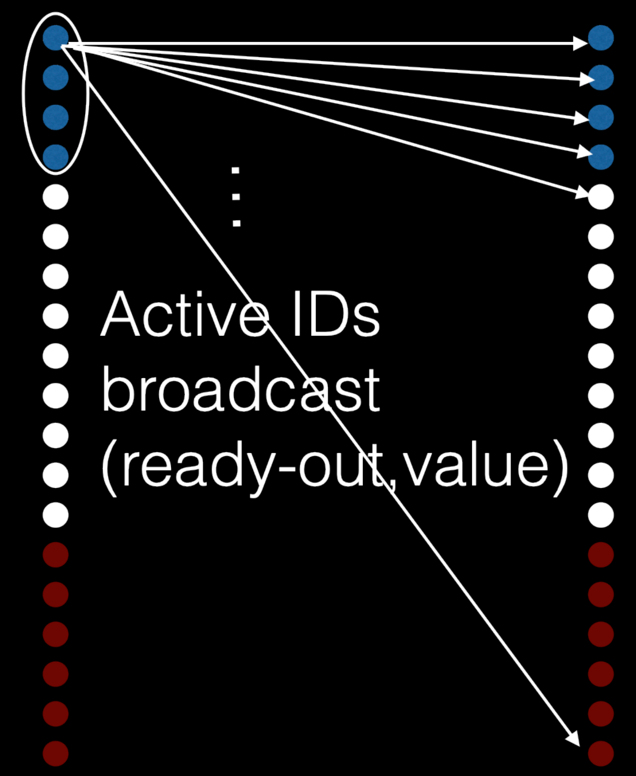


LCBA for “Ready-out” bit



LCBA for “value” bit



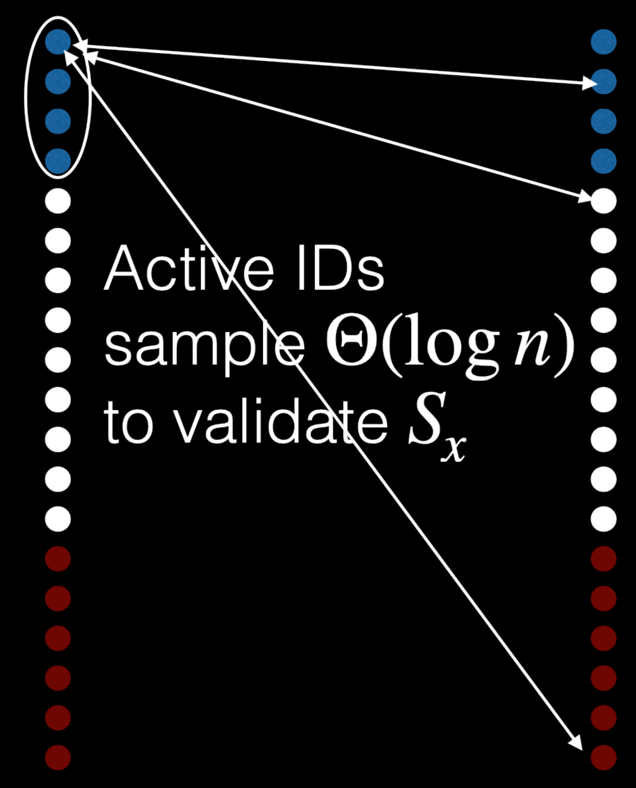


Either:

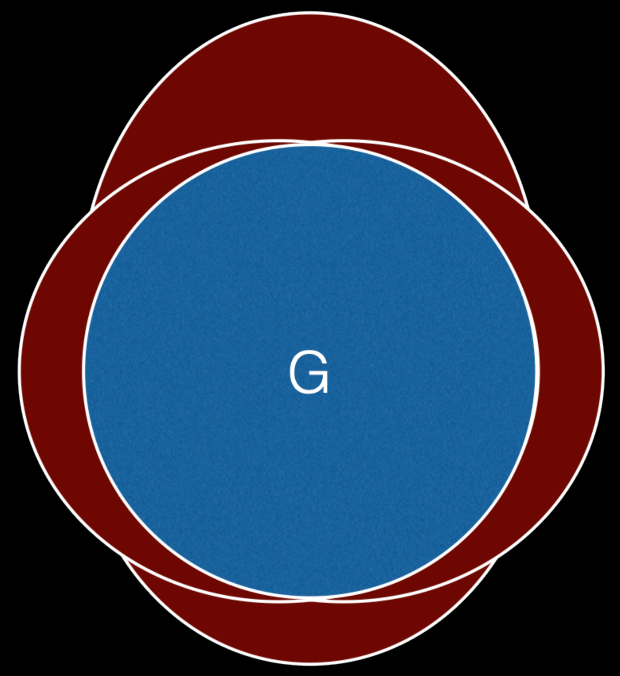
(1) > t/n good IDs decide; or

(2) no good IDs decide

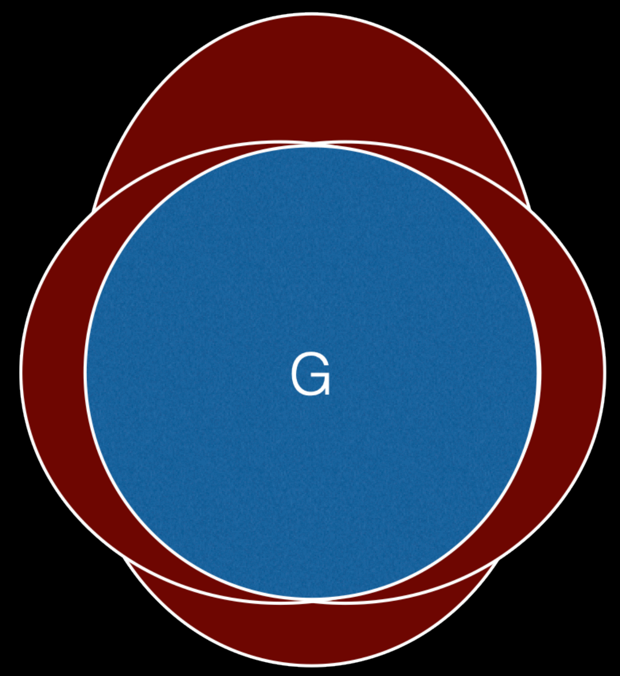
Implicit Agreement

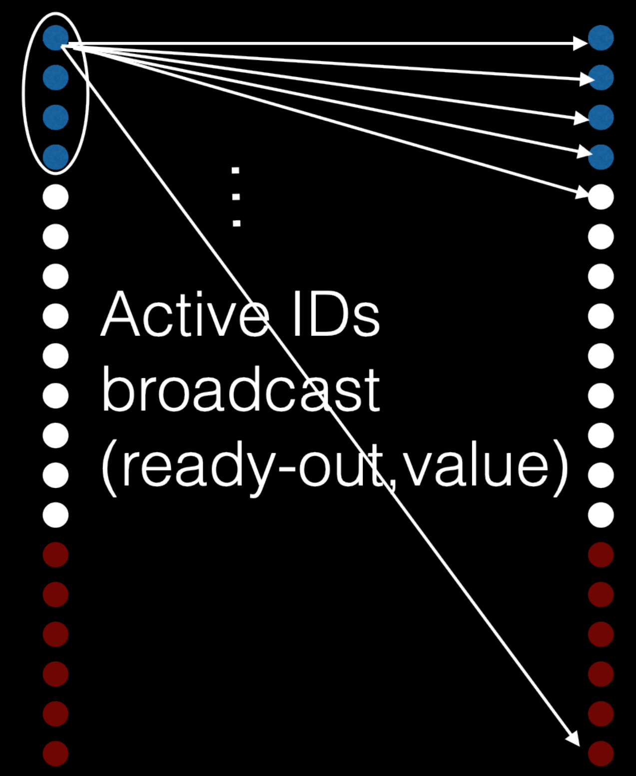


LCBA for "Ready-out" bit



LCBA for "value" bit





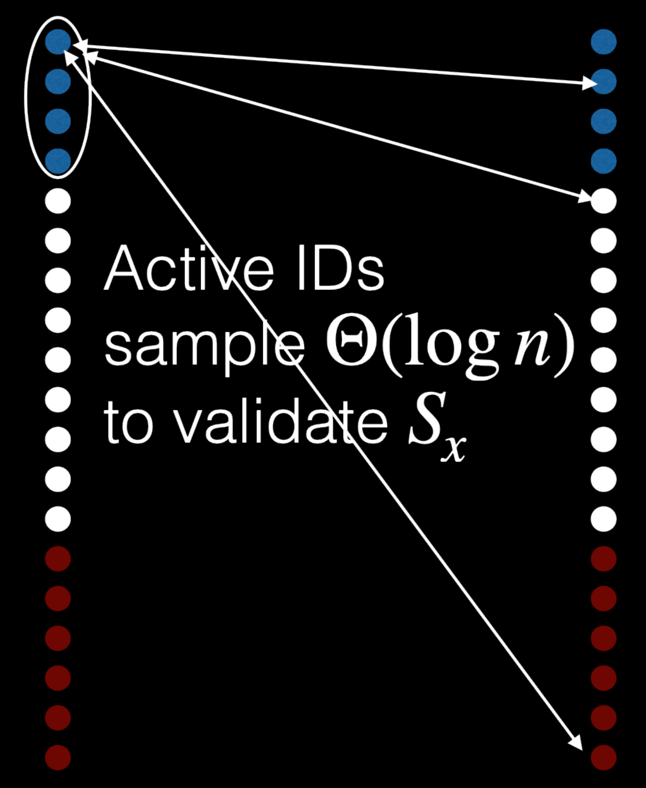
Either:

(1) > t/n good IDs decide; or

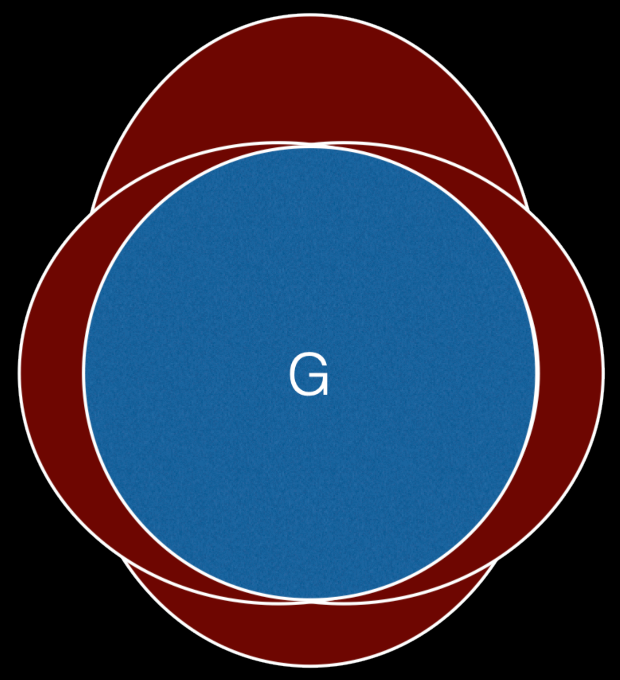
(2) no good IDs decide

Promise Agreement

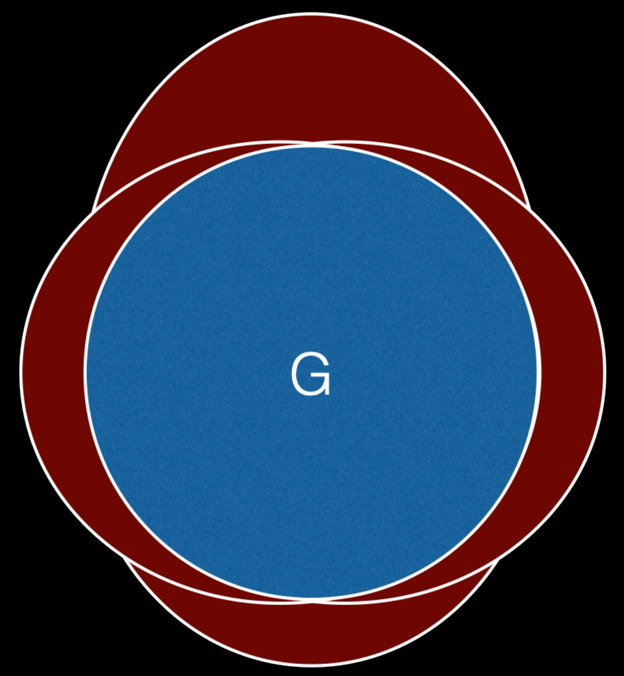
Implicit Agreement

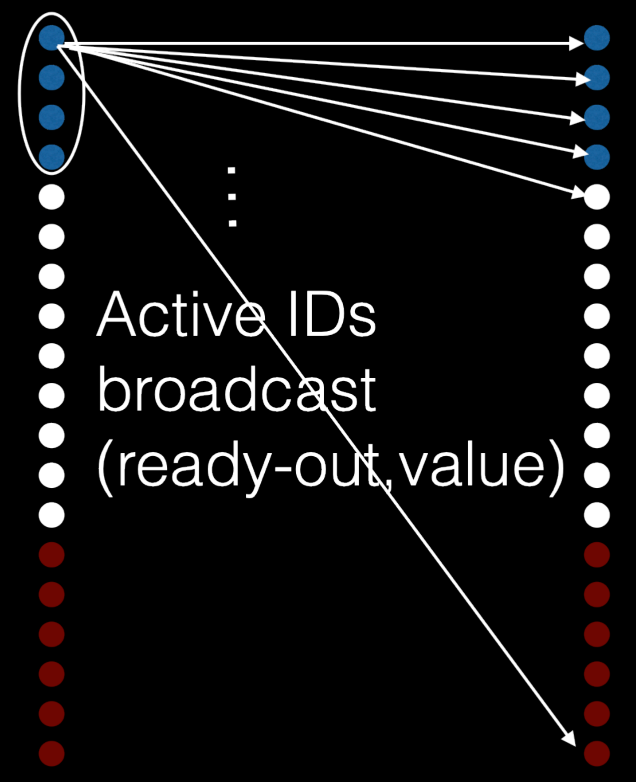


LCBA for “Ready-out” bit



LCBA for “value” bit





Either:

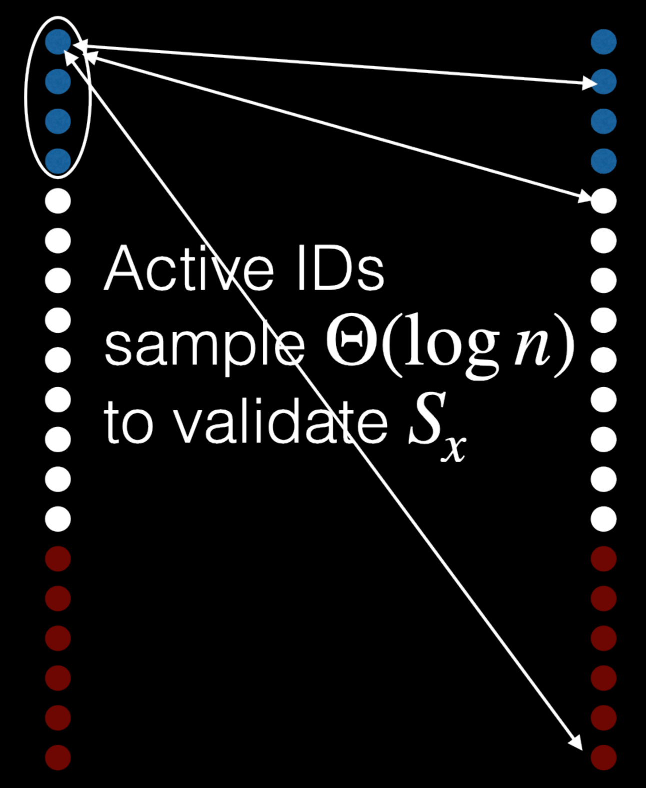
(1) > t/n good IDs decide; or

(2) no good IDs decide

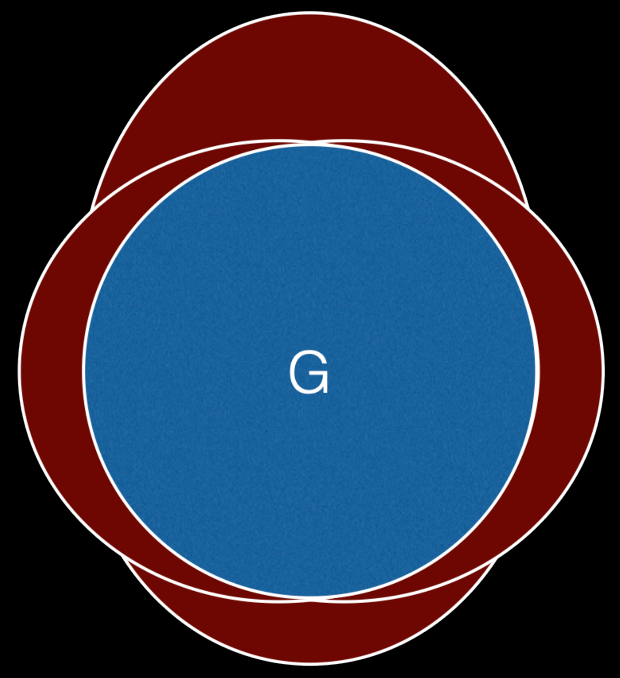
Promise Agreement

Everyone samples

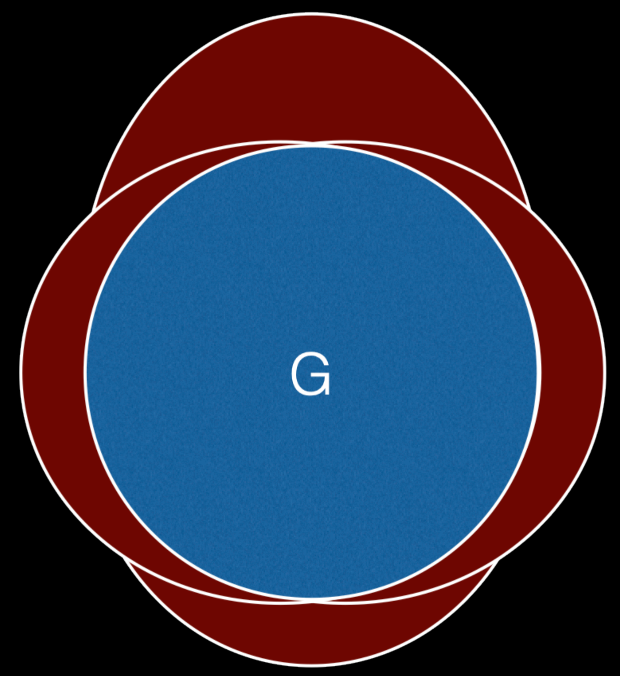
Implicit Agreement

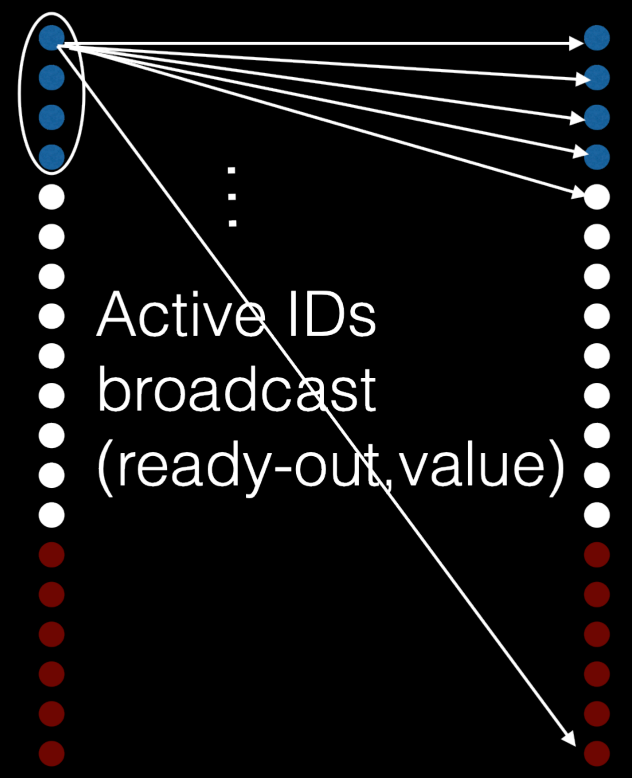


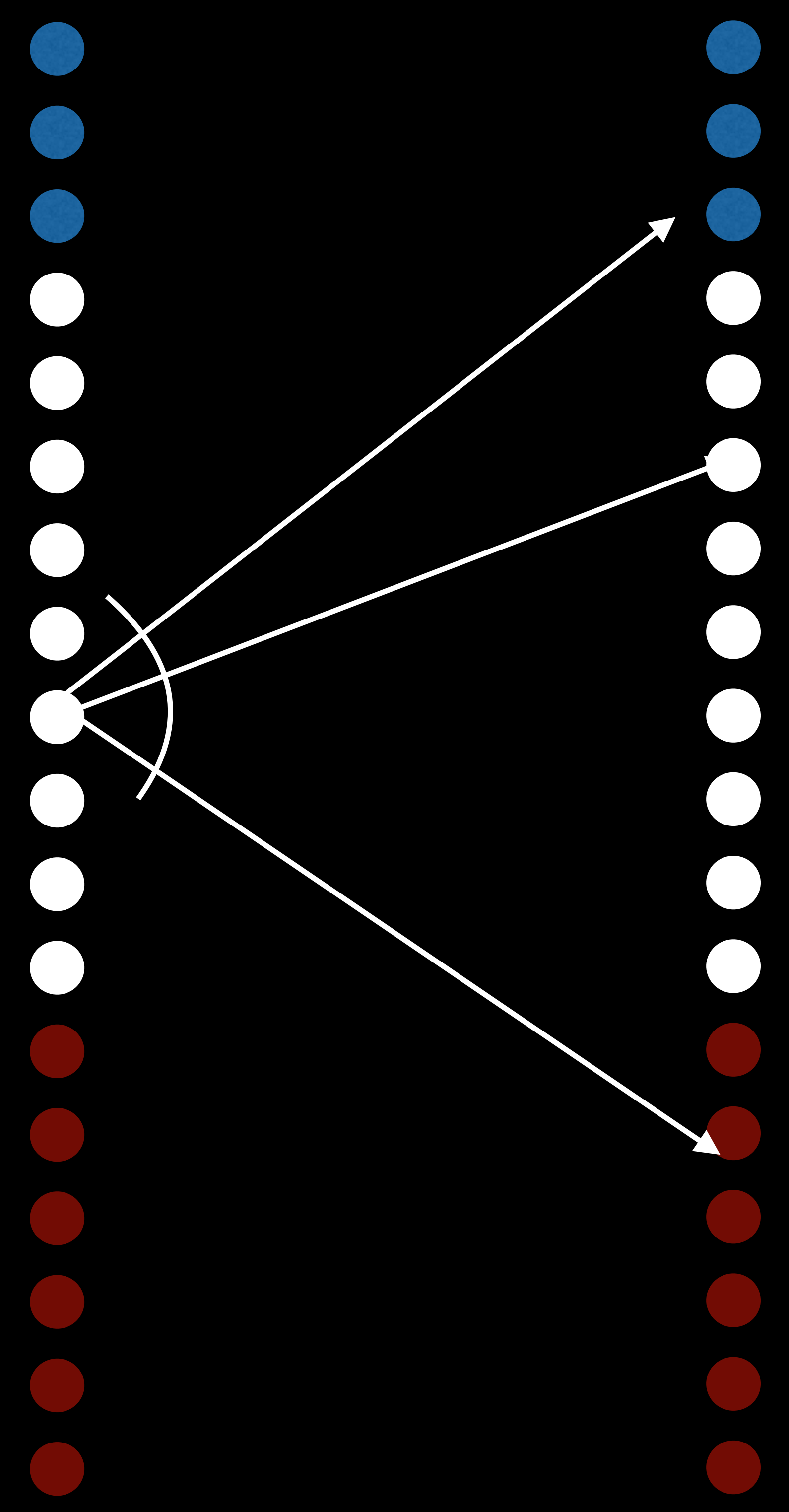
LCBA for "Ready-out" bit



LCBA for "value" bit



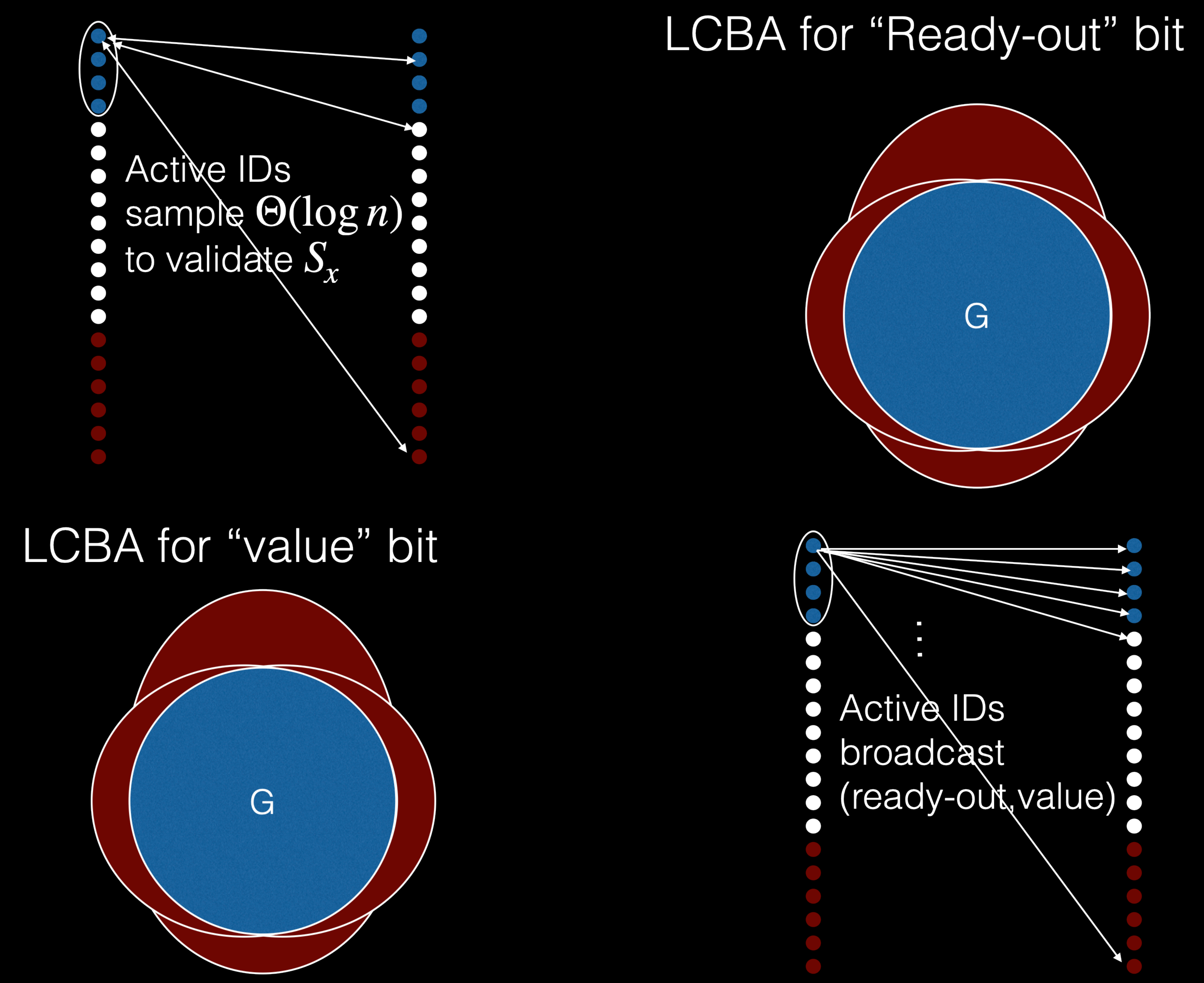




Either:
(1) > t/n good IDs decide; or
(2) no good IDs decide

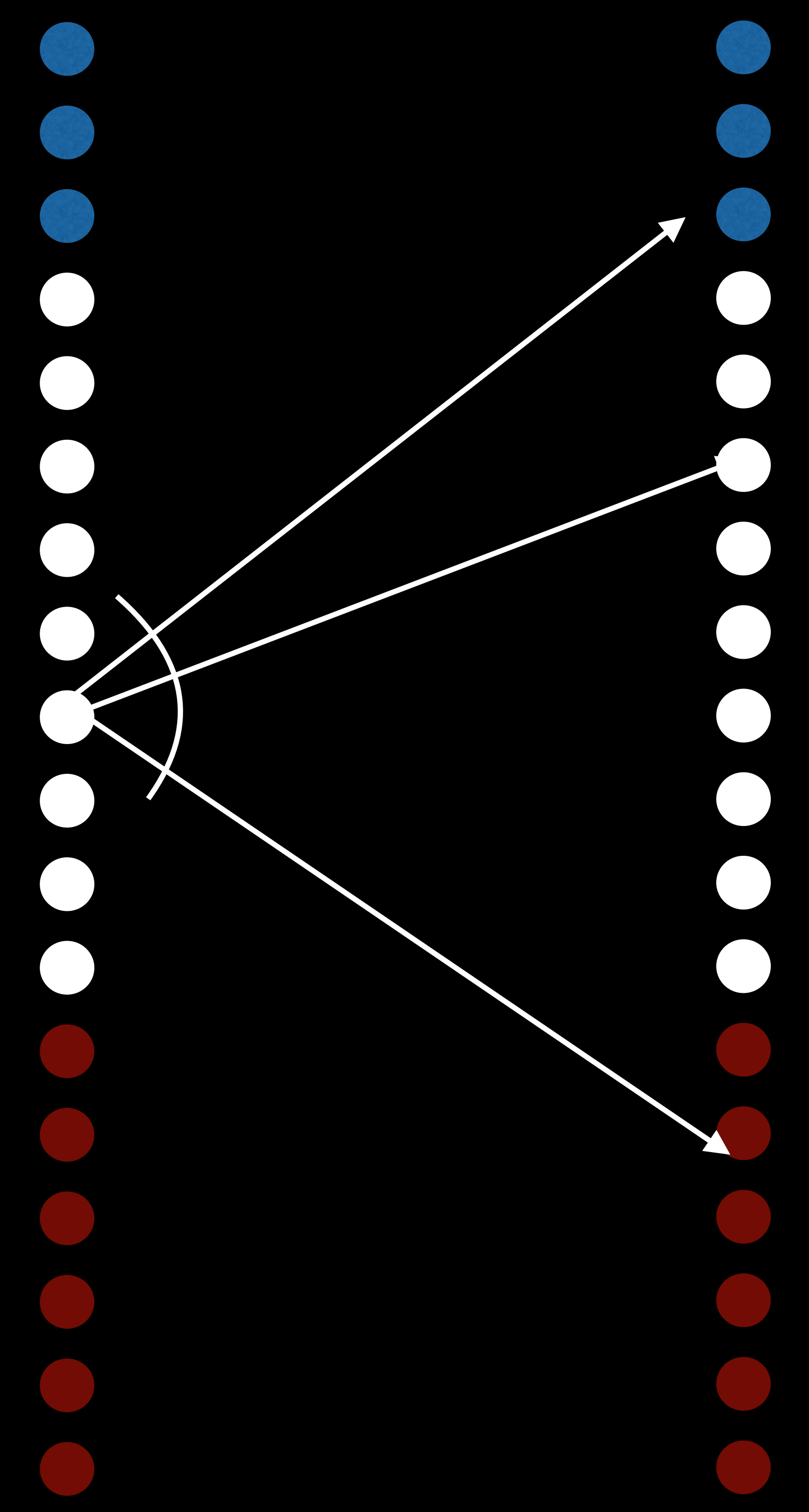
Every good ID samples $\Theta(\log n)$ IDs for their (ready-out, value) bits

Implicit Agreement



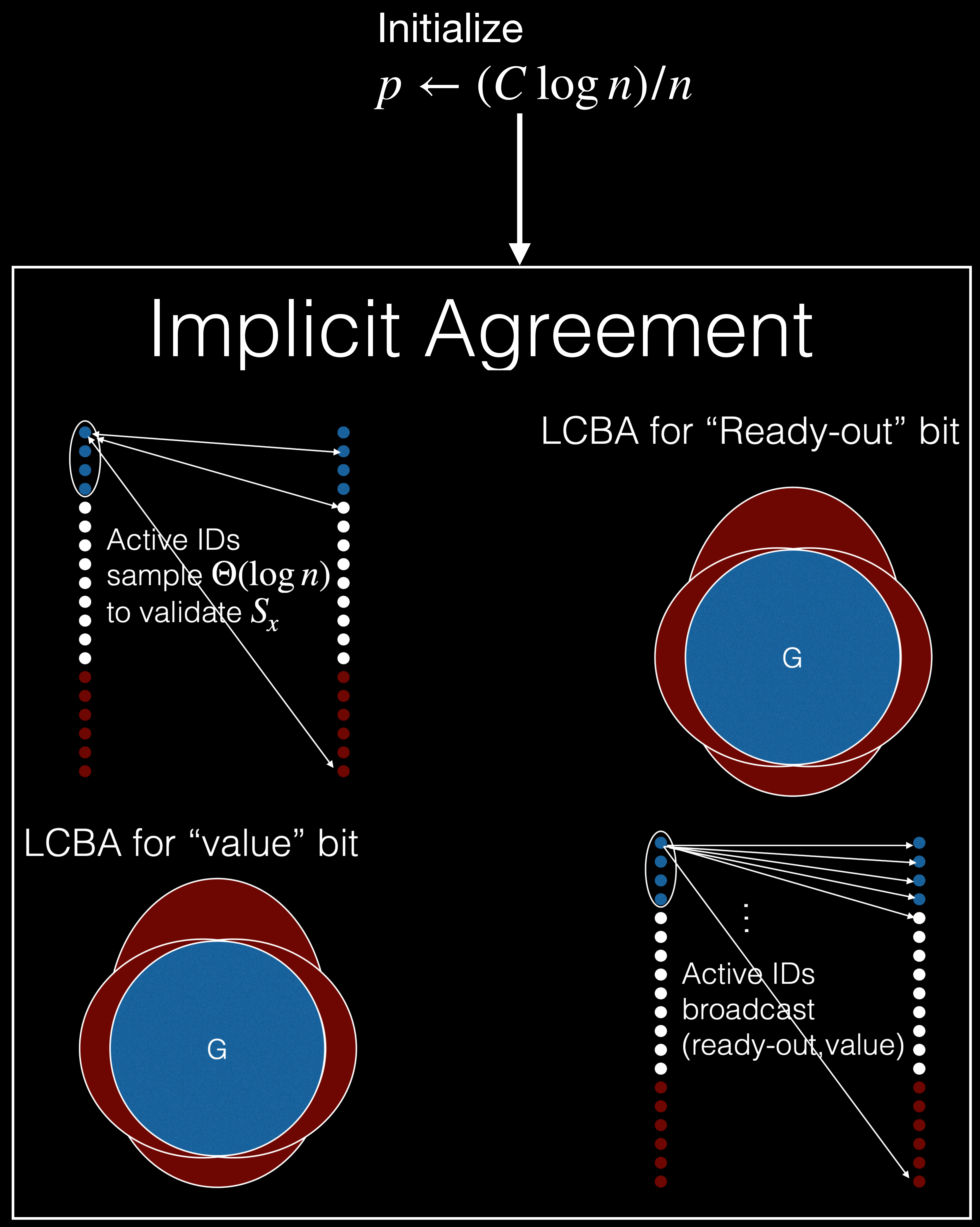
Either:
(1) > t/n good IDs decide; or
(2) no good IDs decide

Promise Agreement



Every good ID samples $\Theta(\log n)$ IDs for their (ready-out, value) bits

Entire Algorithm



Initialize
$p \leftarrow (C \log n)/n$
Implicit Agreement
Active IDs
sample $\Theta(\log n)$
to validate S_x
LCBA for "Ready-out" bit
G
LCBA for "value" bit
G
Active IDs
broadcast
(ready-out,value)

Initialize
$p \leftarrow (C \log n)/n$
Implicit Agreement
LCBA for “Ready-out” bit
Active IDs
sample $\Theta(\log n)$
to validate S_x
G
LCBA for “value” bit
G
Active IDs
broadcast
(ready-out,value)
Promise Agreement
Sample $\Theta(\log n)$
for (ready-out, value)

Initialize
$p \leftarrow (C \log n)/n$
Implicit Agreement
Active IDs
sample $\Theta(\log n)$
to validate S_x
LCBA for “Ready-out” bit
G
LCBA for “value” bit
G
Active IDs
broadcast
(ready-out,value)
Promise Agreement
Sample $\Theta(\log n)$
for (ready-out, value)
If fail:
$p \leftarrow 2p$
If success:
DONE!

Initialize
$p \leftarrow (C \log n)/n$
Implicit Agreement
Active IDs
sample $\Theta(\log n)$
to validate S_x
LCBA for "Ready-out" bit
G
LCBA for "value" bit
G
Active IDs
broadcast
(ready-out,value)
Promise Agreement
Sample $\Theta(\log n)$
for (ready-out, value)
If fail:
$p \leftarrow 2p$
$p < \frac{1}{C \log n}$?
Y
N
Heavy-weight
Byzantine agreement
If success:
DONE!

Conclusion

Can solve Byzantine agreement in KT_0 with:

$O(\text{polylog}(n))$ latency

$O((T + n)\log n)$ expected messages

$T = \min(n^2,$ # bits sent by adversary$)$

Can solve Byzantine agreement in KT_0 with:

$O(\text{polylog}(n))$ latency

$O((T+n)\log n)$ expected messages

$T = \min(n^2, \text{\# bits sent by adversary})$

Communication with strangers only occurs via:

(1) broadcast to all strangers; (2) writing to random stranger

Can solve Byzantine agreement in KT_0 with:

$O(\text{polylog}(n))$ latency

$O((T + n)\log n)$ expected messages

$T = \min(n^2, \text{\# bits sent by adversary})$

Communication with strangers only occurs via:

(1) broadcast to all strangers; (2) writing to random stranger

Almost matching lower bounds for polylog(n) round algs

Non-trivial lower-bound for all Las Vegas algorithms

Future Work

(1) Closing gap between upper and lower bounds for randomized algorithms:

Know: If $T = n^{1+\alpha}$ for $\alpha \in (0,1]$, then any Las Vegas algorithm sends $\Omega(n^{1+\alpha/2})$ bits in expectation

But, our algorithm sends $O(n^{1+\alpha})$ bits in this case

(1) Closing gap between upper and lower bounds for randomized algorithms:

Know: If $T = n^{1+\alpha}$ for $\alpha \in (0,1]$, then any Las Vegas algorithm sends $\Omega(n^{1+\alpha/2})$ bits in expectation

But, our algorithm sends $O(n^{1+\alpha})$ bits in this case

(2) Can we adapt our algorithm to better handle churn in permissionless networks?

Need a good model of churn

Thanks!