Riposte: an Anonymous Messaging System that 'Hides the Metadata'

Henry Corrigan-Gibbs

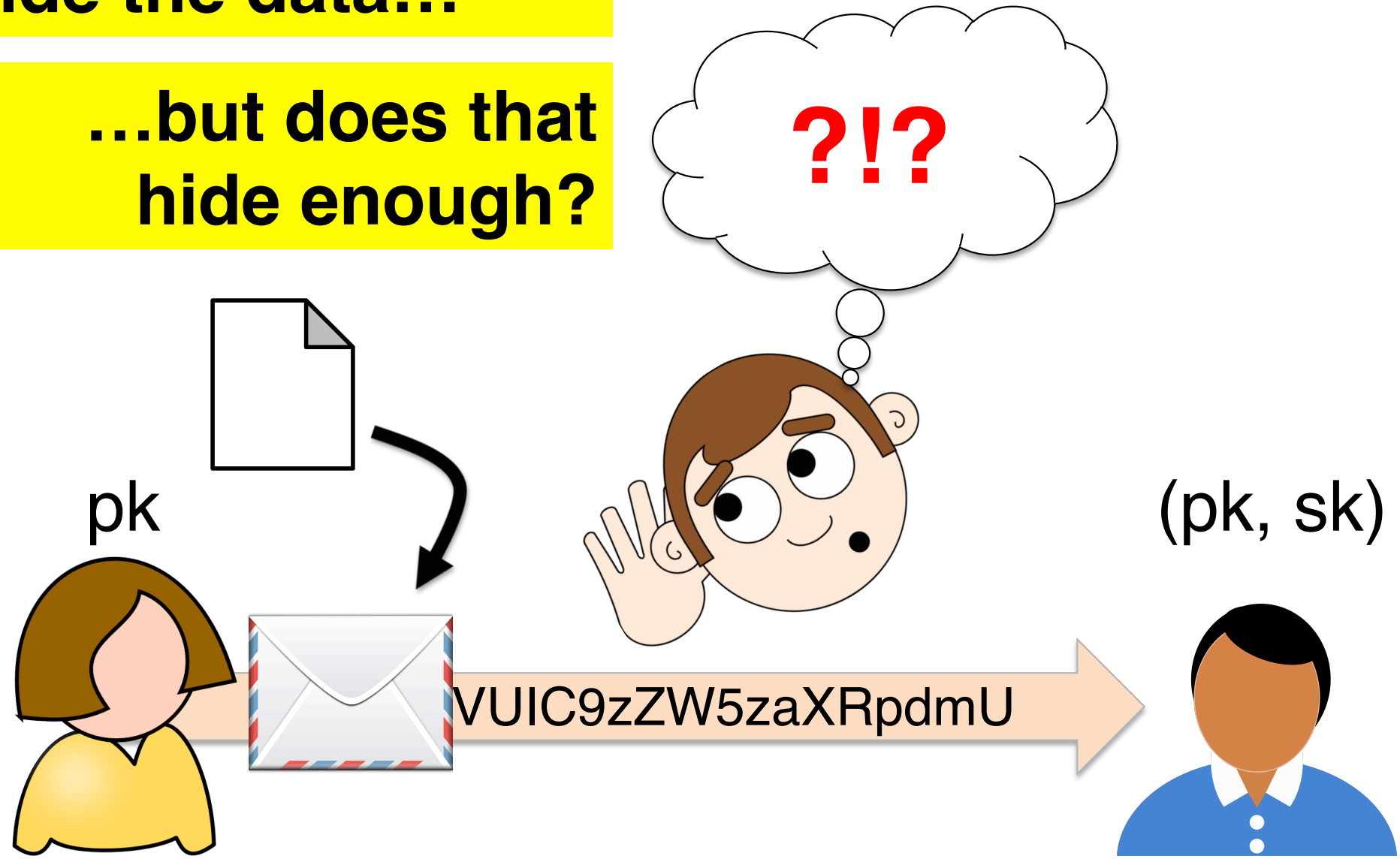
Joint work with Dan Boneh and David Mazières

To appear at IEEE Security and Privacy 2015

Charles River Crypto Day
20 February 2015

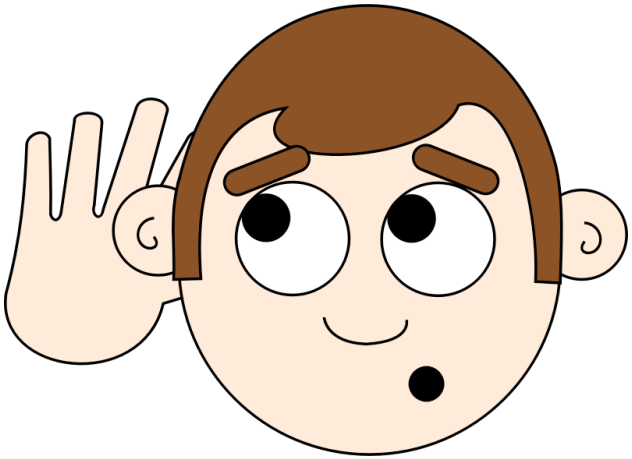
With PKE, we can hide the data…

…but does that hide enough?



Time	From	To	Size
10:12	Alice	Bob	2543 B
10:27	Carol	Alice	567 B
10:32	Alice	Bob	450 B
10:35	Bob	Alice	9382 B

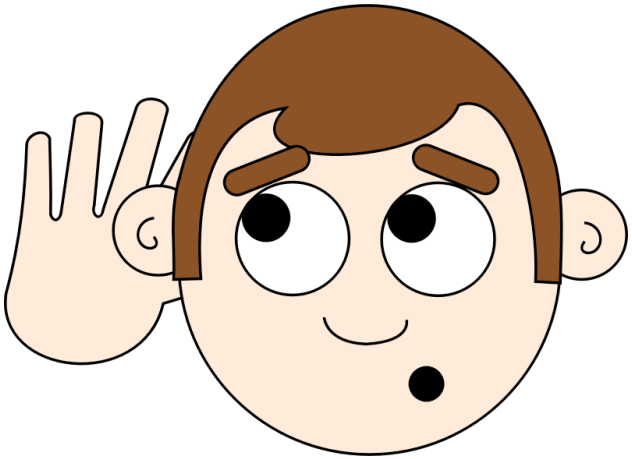
⋮



Time	From	To	Size
10:12	Alice	**taxfraud@stanford.edu**	2543 B
10:27	Carol	Alice	567 B
10:32	Alice	Bob	450 B
10:35	Bob	Alice	B

⋮

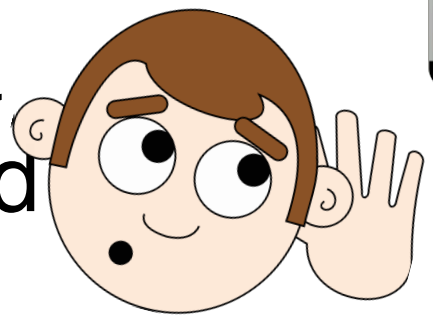
Hiding the data is necessary, but not sufficient

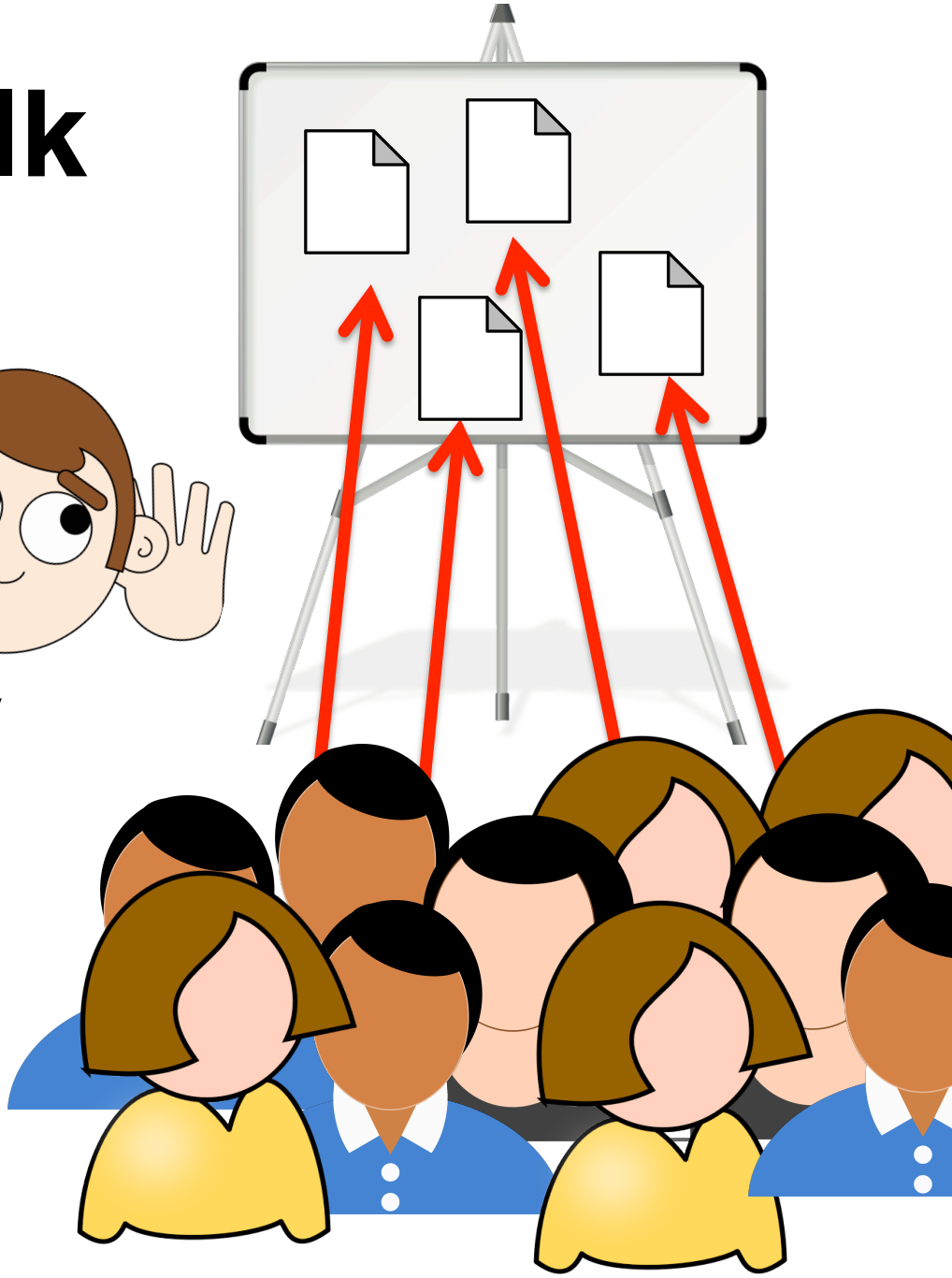


[cf. Ed Felten's testimony before the House Judiciary Committee, 2 Oct 2013]

Focus of this talk

Goal: post "anonymously" to a public bulletin board

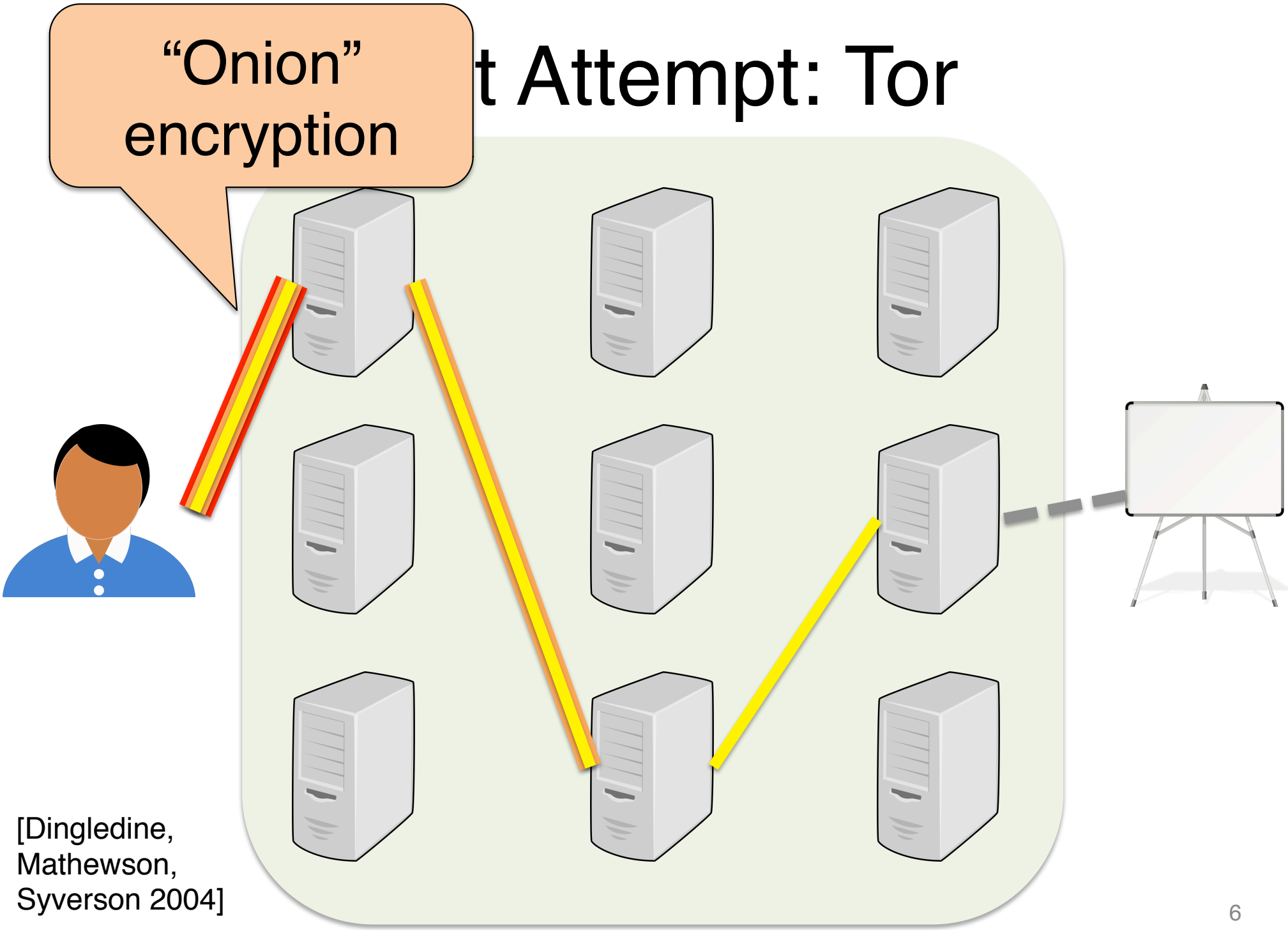




Building block for many problems related to "hiding the metadata"

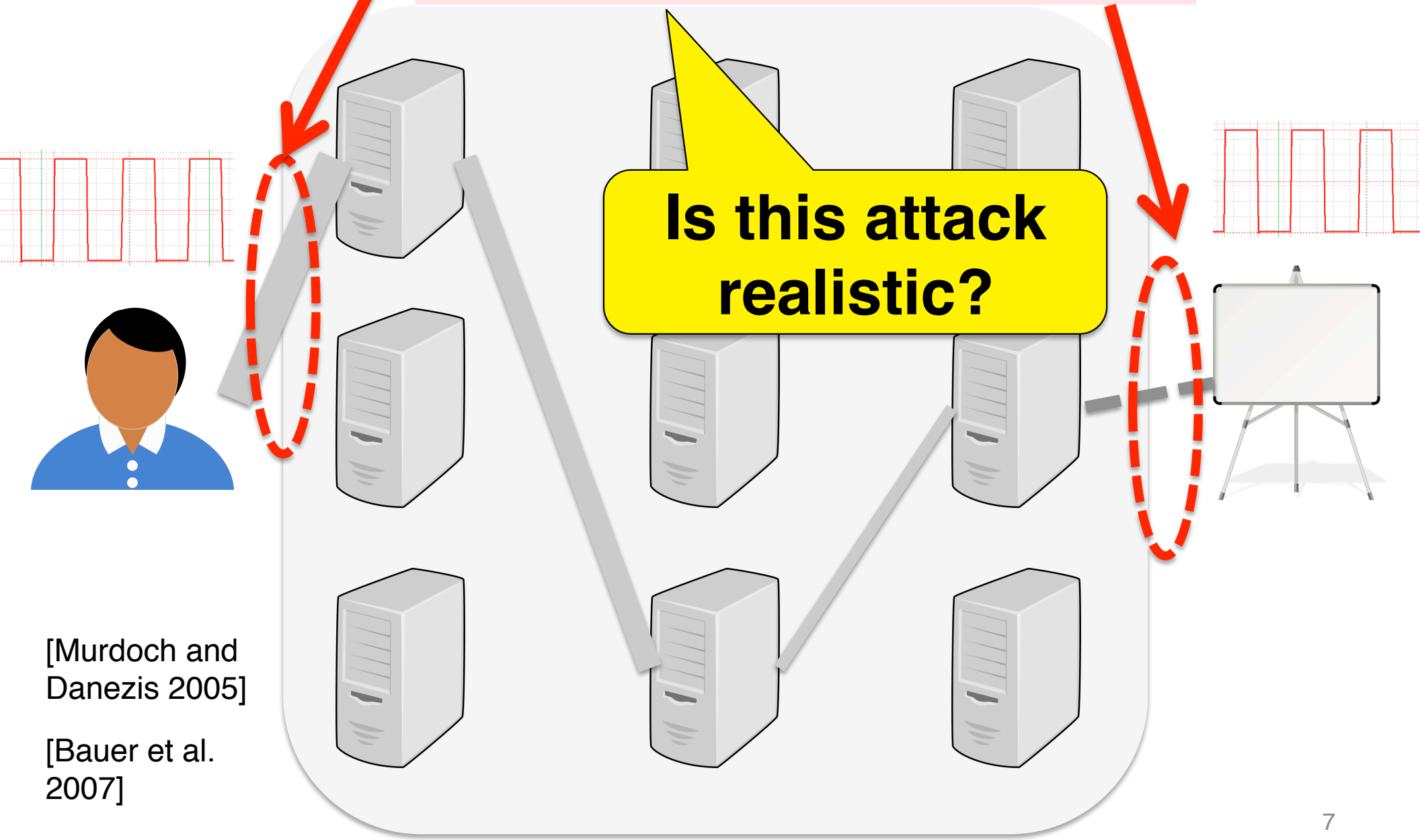
→ E-voting
→ Anonymous surveys
→ Private messaging, etc.

t Attempt: Tor



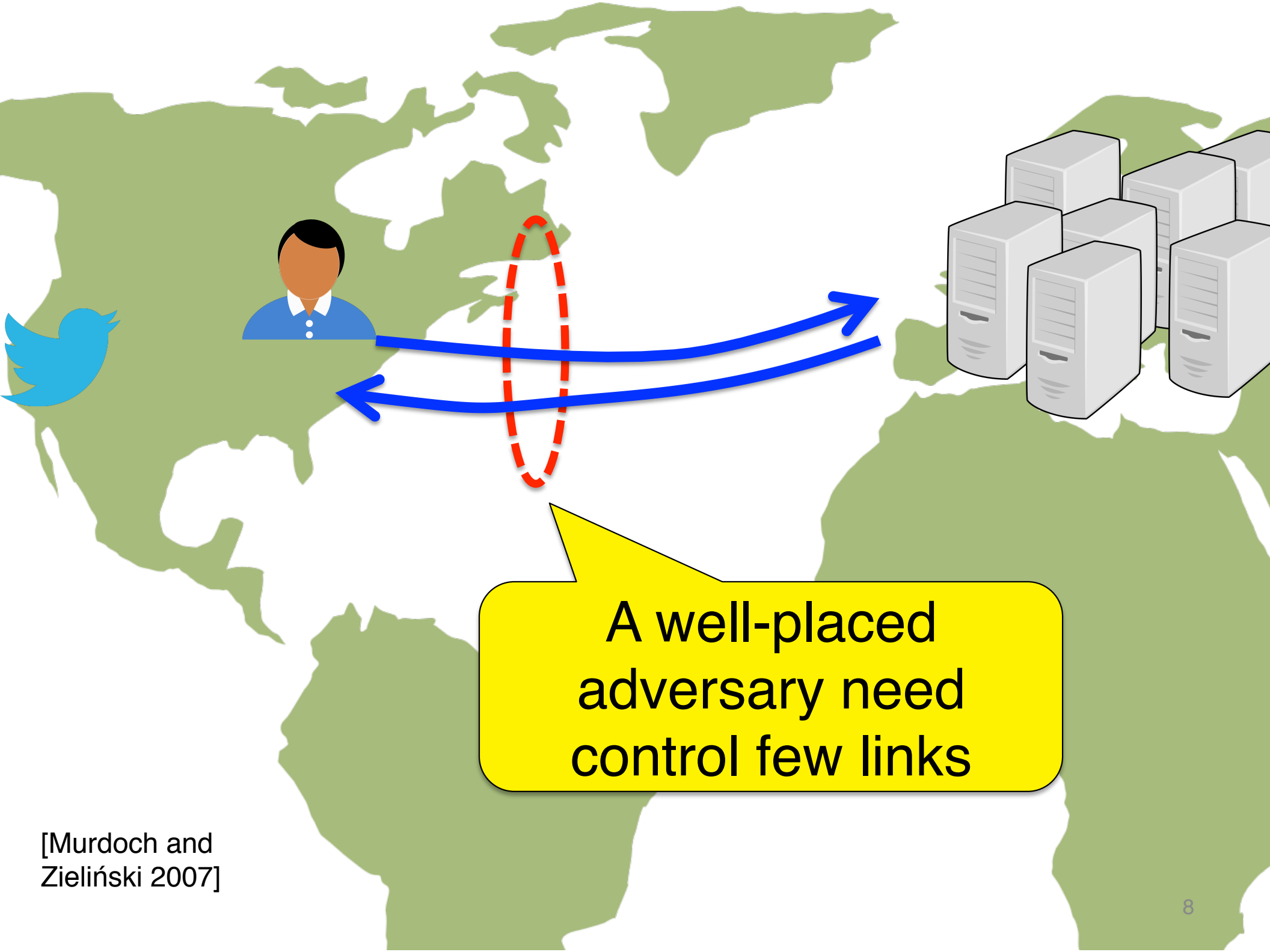
[Dingledine, Mathewson, Syverson 2004]

Passive network adversary can correlate flows!



[Murdoch and Danezis 2005]

[Bauer et al. 2007]



[Murdoch and Zieliński 2007]

Tor is **practical** at Internet scale

… but its security properties are unclear

We design an anonymous messaging system that:

1) satisfies clear security goals,

2) handles millions of users in an "anonymous Twitter" system.

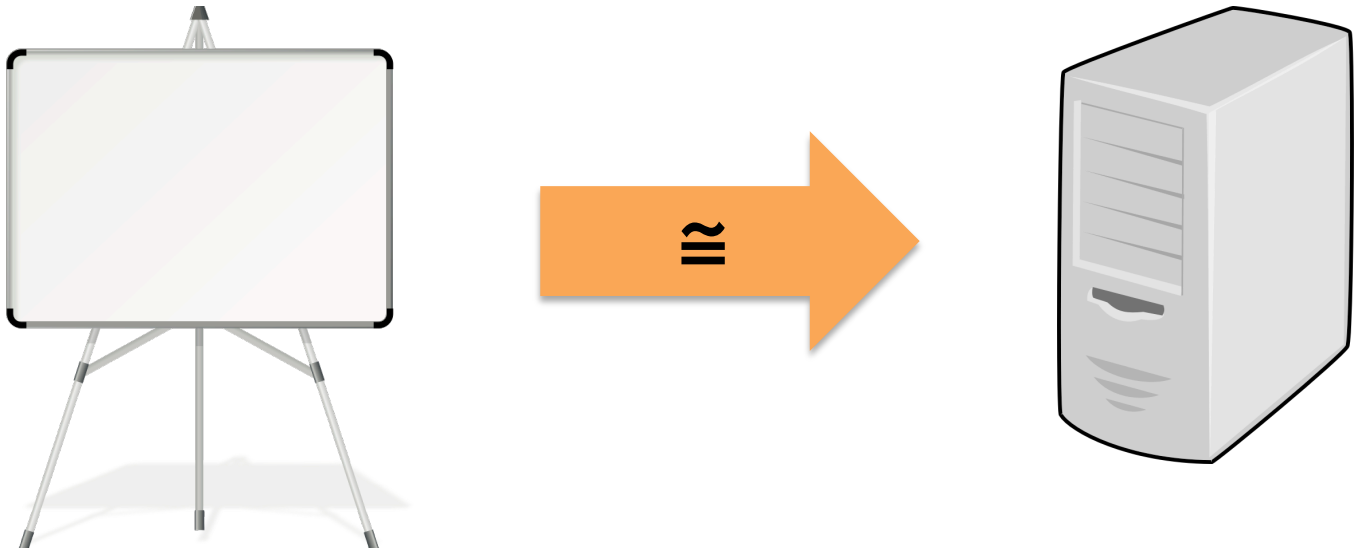
Outline

- **Motivation**
- Definitions and a “Straw man” scheme
- Technical challenges
- Evaluation
- Conclusions

Outline

- Motivation
- **Definitions and a “Straw man” scheme**
- Technical challenges
- Evaluation
- Conclusions

Reframe the Problem

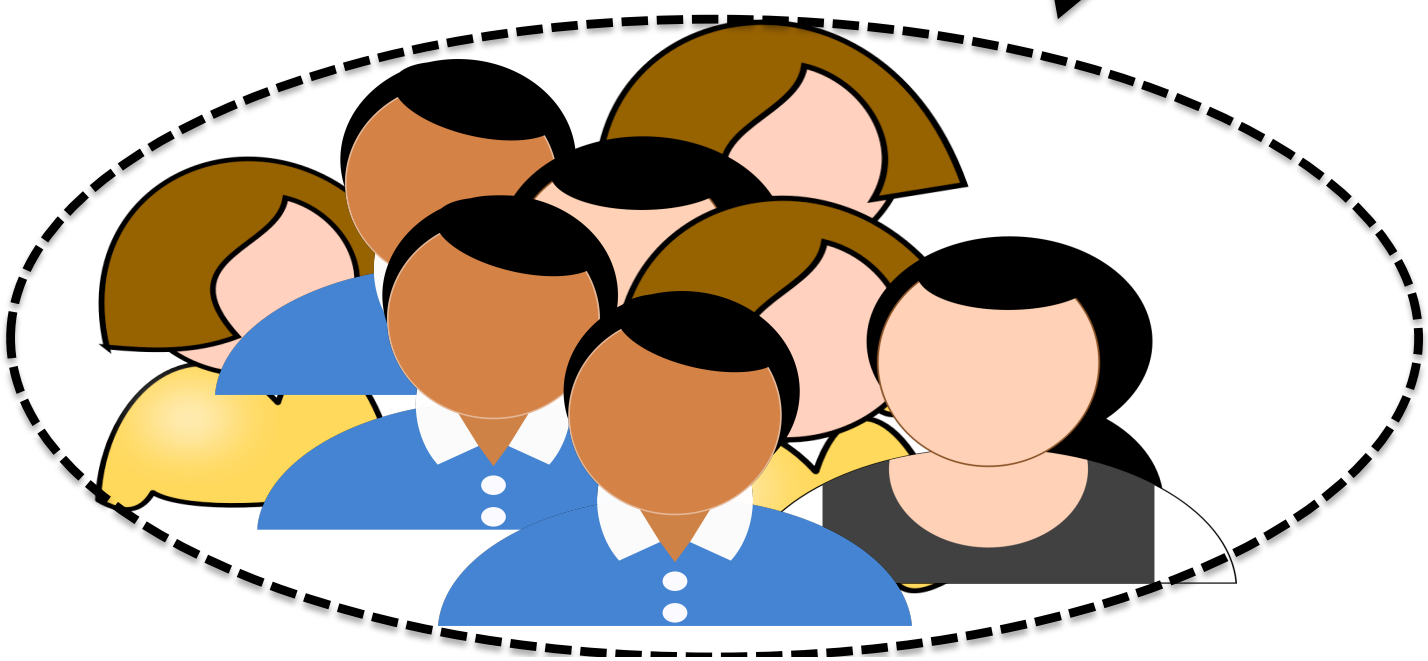


Posting anonymously to a bulletin board

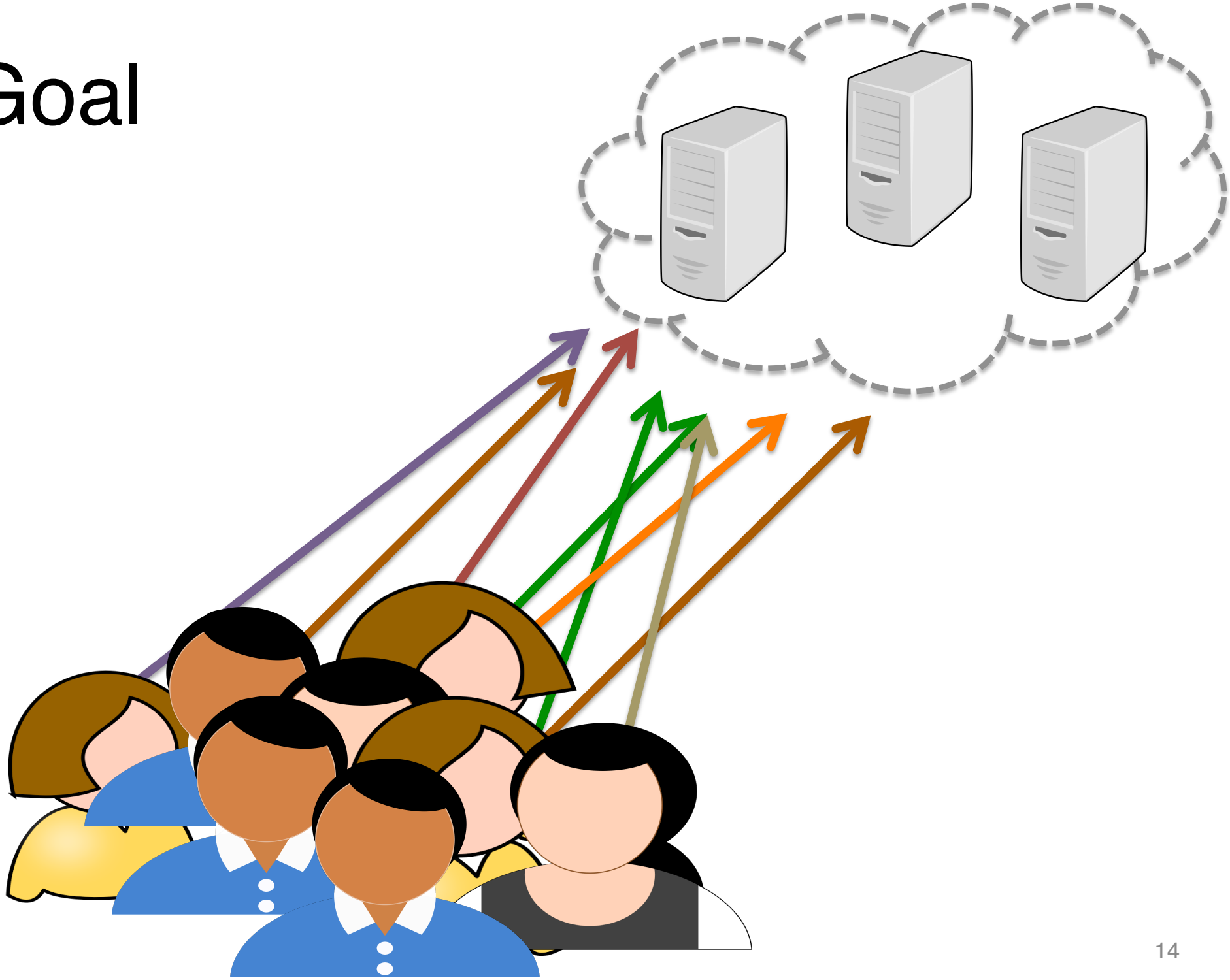
Writing “privately” to a database

Goal

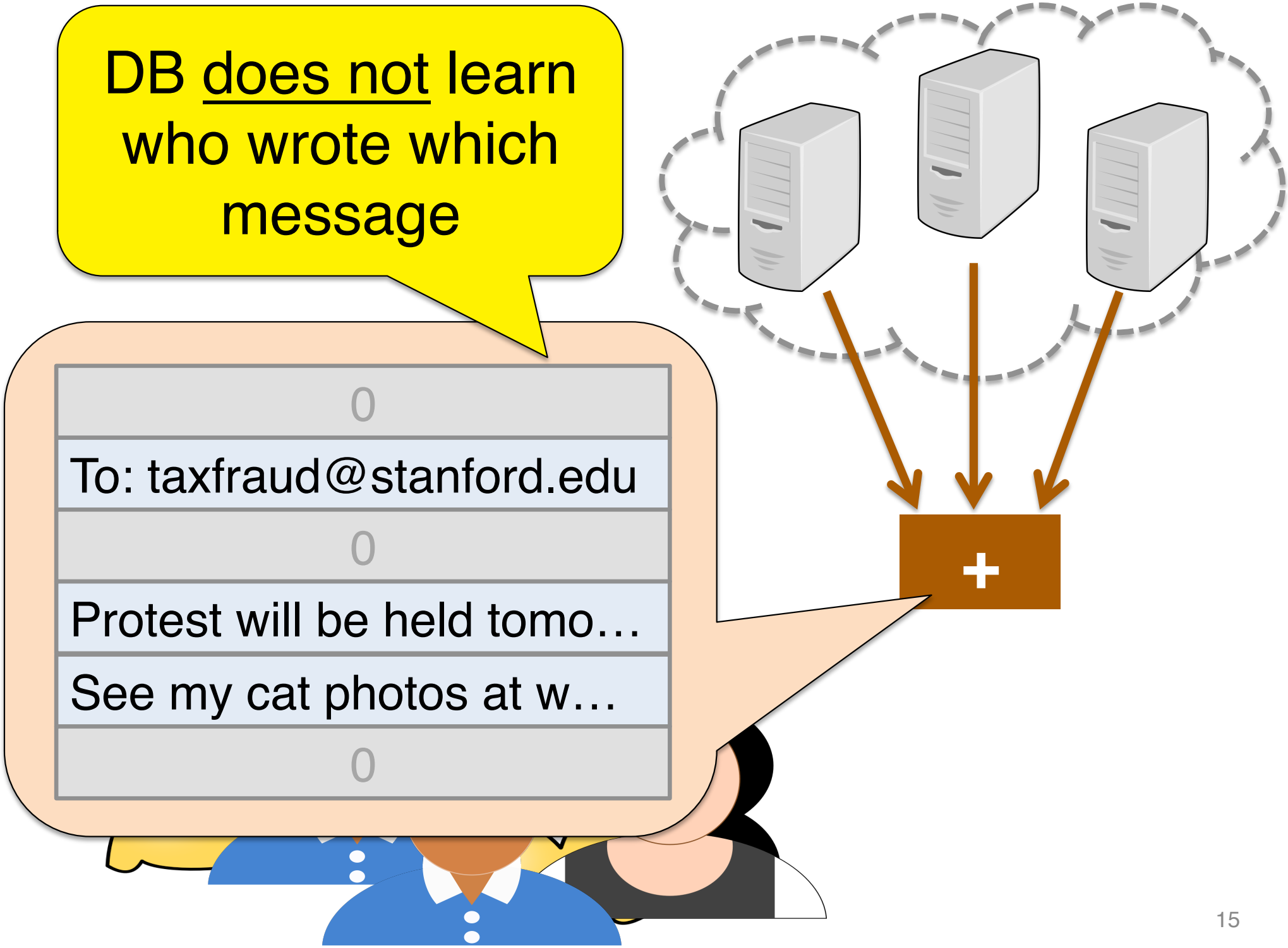
The “Anonymity Set”



Goal

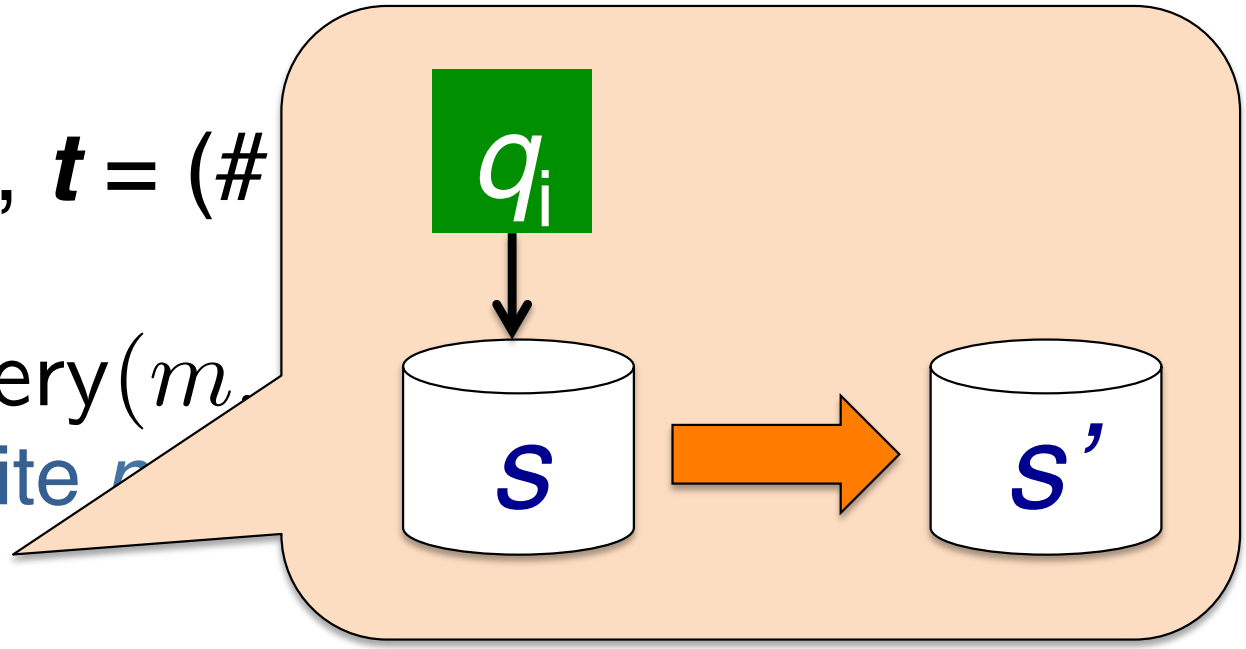


DB <u>does not</u> learn who wrote which message



(k,t)-Write-Anonymous DB Scheme

k = (# of servers), ***t*** = (#



$(q_1, \ldots, q_k) \leftarrow \mathsf{Query}(m,$

[Gen query to write m

$s' \leftarrow \mathsf{Update}(s, q_i)$

[Apply query to state of server i]

$DB \leftarrow \mathsf{Reveal}(s_1, \ldots, s_k)$

[Combine server states to reveal plaintext DB]

Goals

1. Correctness
2. Write-anonymity
3. Disruption resistance

Goal 1: Correctness

Informal: Output DB should be result of applying queries to DB state.

If queries are: $(m_1, \ell_1), \ldots, (m_n, \ell_n)$

then result of Reveal() is:

$$\sum_{i=1}^{n} m_i e_{\ell_i}$$

Goal 2: Write-Anonymity

Informal: coalition of ***t*** malicious servers and any number of malicious clients should not learn <u>who wrote what</u> to the DB.

I will present the [simplified] two-server definition with one malicious server.

Challenger

Let **n** = number of clients total

Adversary

$H \subseteq [n]$ s.t. $|H| \geq 2$

$\{(m_i, \ell_i) \mid i \in H\}$

For $i \in H$:

$(q_i, \hat{q}_i) \leftarrow$ Query(m_i, ℓ_i)

Choose on π elements of H

$\{q_{\pi(i)} \mid i \in H\}$

Challenger

Let **n** = number of clients total

Adversary

$H \subseteq [n]$ s.t. $|H| \geq 2$

$\{(m_i, \ell_i) \mid i \in H\}$

For $i \in H$:

$(q_i, \hat{q}_i) \leftarrow$ Query(m_i, ℓ_i)

Choose perm π on elements of H

$\{q_{\pi(i)} \mid i \in H\}$

$\{\hat{q}_i \mid i \notin H\}$

Let n – number of clients total

$H \subseteq [n]$ s.t. $|H| \geq 2$

$\{(m_i, \ell_i) \mid i \in H\}$

For $i \in H$:

$(q_i, \hat{q}_i) \leftarrow$ Query(m_i, ℓ_i)

Choose perm π on elements of H

$\{q_{\pi(i)} \mid i \in H\}$

$\{\hat{q}_i \mid i \notin H\}$

$s \leftarrow$ Update

$H \subseteq [n]$ s.t. $|H| \geq 2$

$\{(m_i, \ell_i) \mid i \in H\}$

For $i \in H$:

$(q_i, \hat{q}_i) \leftarrow$ Query(m_i, ℓ_i)

Choose perm π on elements of H

$\{q_{\pi(i)} \mid i \in H\}$

$\{\hat{q}_i \mid i \notin H\}$

$s \leftarrow$ Update $(\hat{q}_1, \ldots, \hat{q}_n)$

For $i \in H$:

$(q_i, \hat{q}_i) \leftarrow$ Query(m_i, ℓ_i)

$\{(m_i, \ell_i) \mid i \in H\}$

Choose

on eler

$i \in H\}$

Queries in H updated according to permutation π

$\{\hat{q}_i \mid i \notin H\}$

$s \leftarrow$ Update $(\hat{q}_1, \ldots, \hat{q}_n)$

s, π

For $i \in H$:

$(q_i, \hat{q}_i) \leftarrow$ Query(m_i, ℓ_i)

Choose perm π on elements of H

Choose perm π^* on elements of H

$s \leftarrow$ Update $(\hat{q}_1, \ldots, \hat{q}_n)$

$\{(m_i, \ell_i) \mid i \in H\}$

$\{q_{\pi(i)}$

$\{\hat{q}_i \mid i \in$

s, π^*

Intuition: The scheme hides "who wrote what" (which query corresponds to which message)

Goal 3: Disruption Resistance

Intuition: each query should change at most one DB row—prevent disruption

Informal: An adversary cannot generate N "valid" queries that affect > N rows

[We defer the definition a "valid" query for now…]

Privacy-Preserving DB Schemes

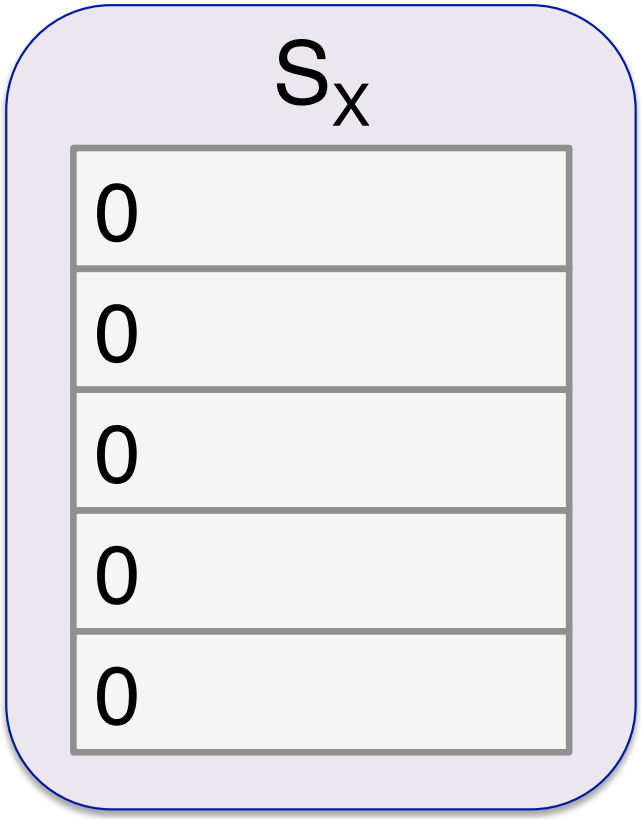
ORAM [GO'96] / Group ORAM [GOMT'11]
– CPU(s) writing to RAM

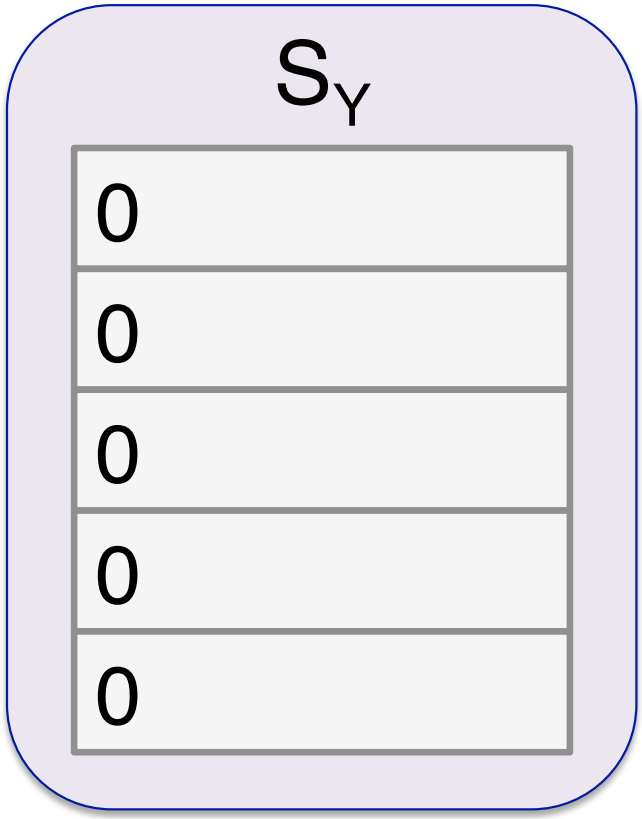
Private Info Retrieval (PIR) [CGKS'97]
– Client reading from DB

Private Info Storage [C
– Client writing to DB sh

Ideally: for all **k**, tolerate compromise of **k-1** servers

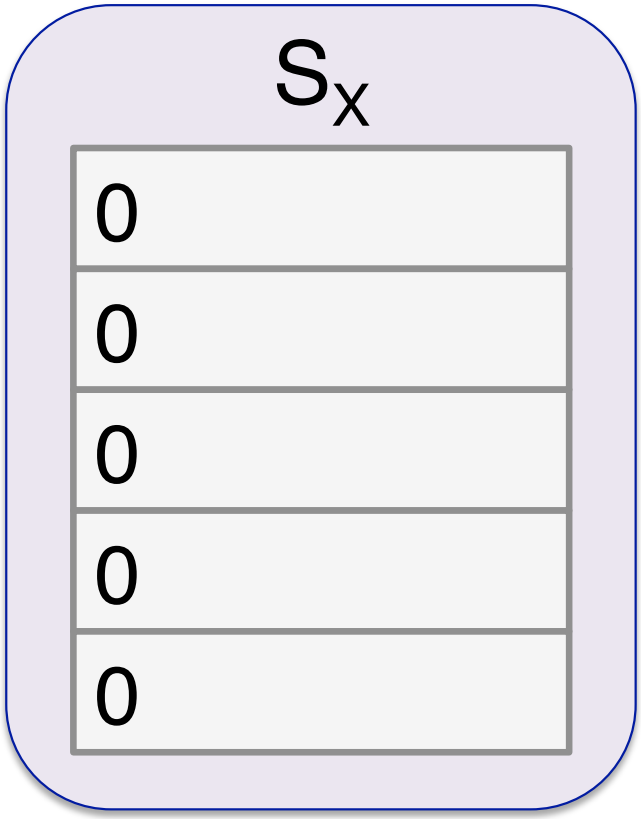
This work: Many clients (incl malicious ones) writing to DB shared across servers

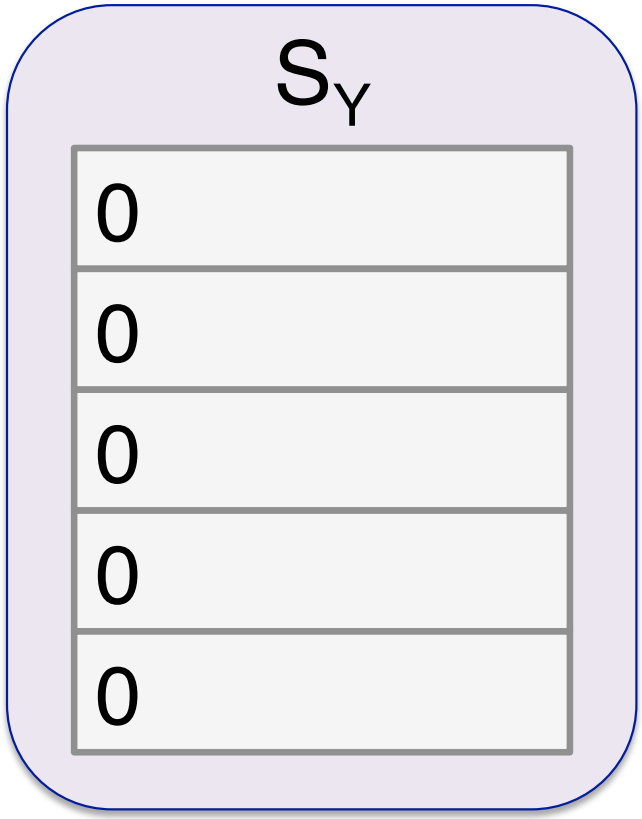




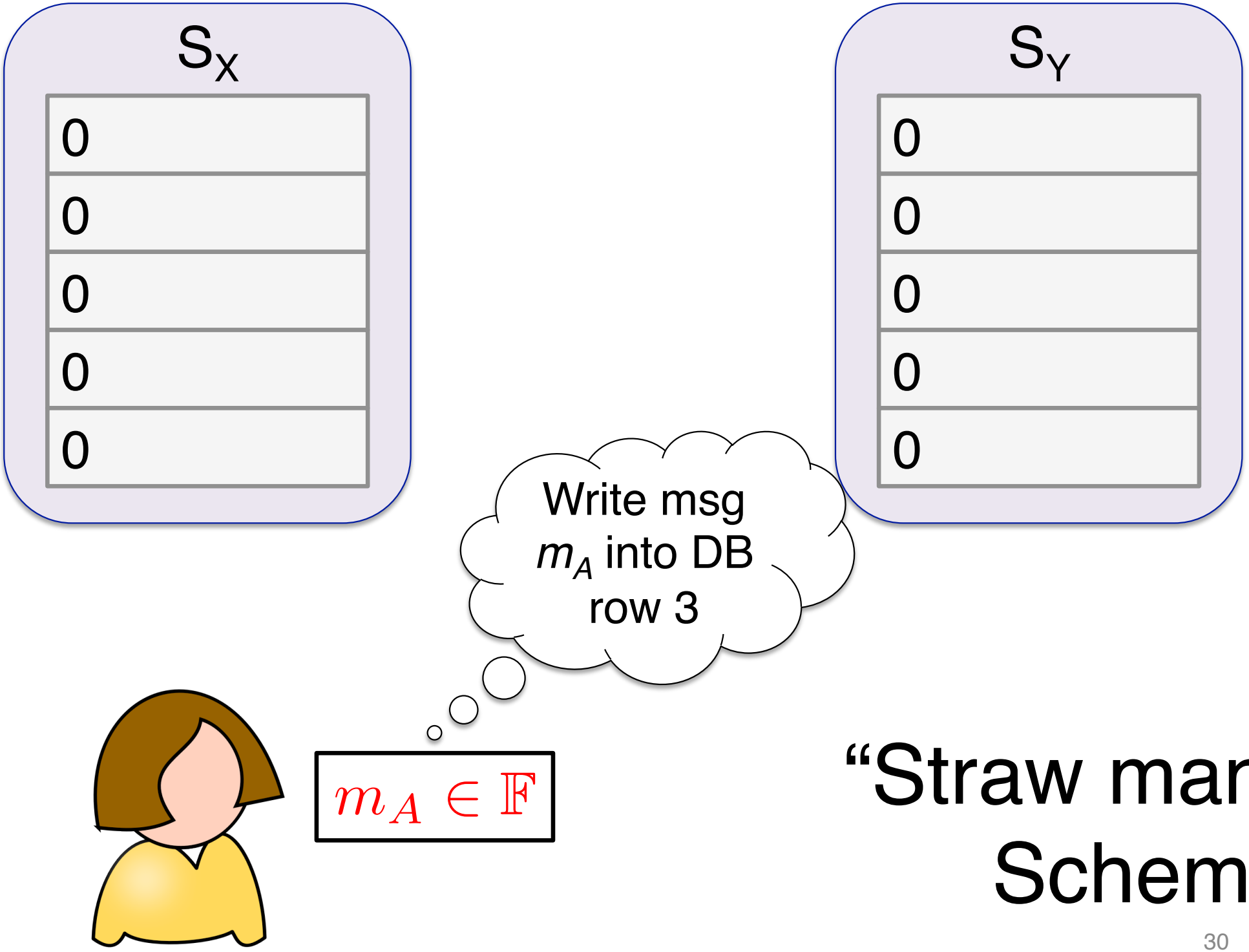
(2,1)-Private "Straw man" Scheme

[Chaum '88]

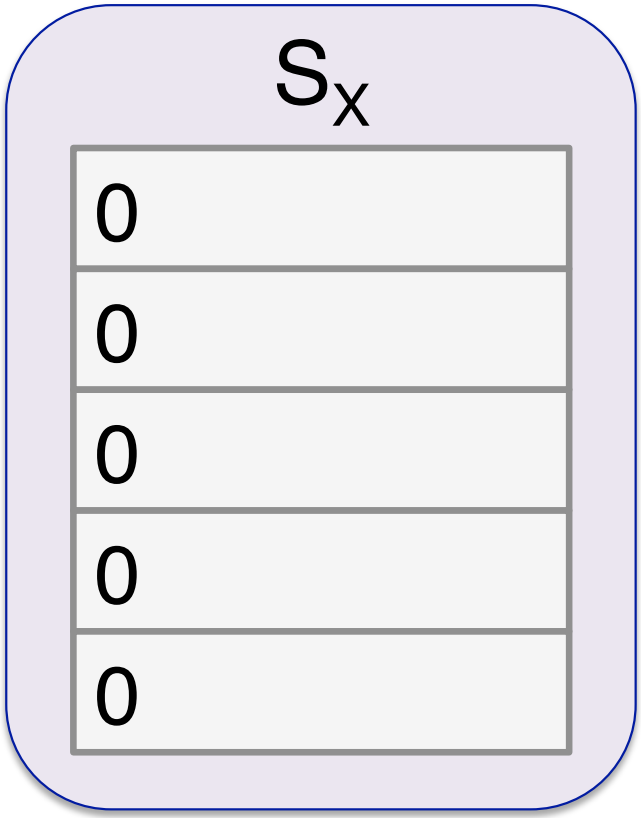


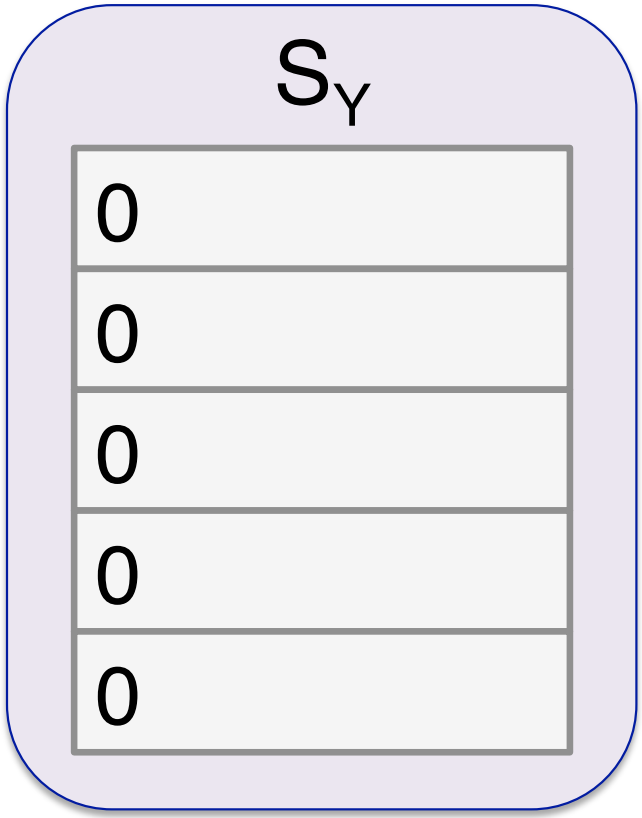


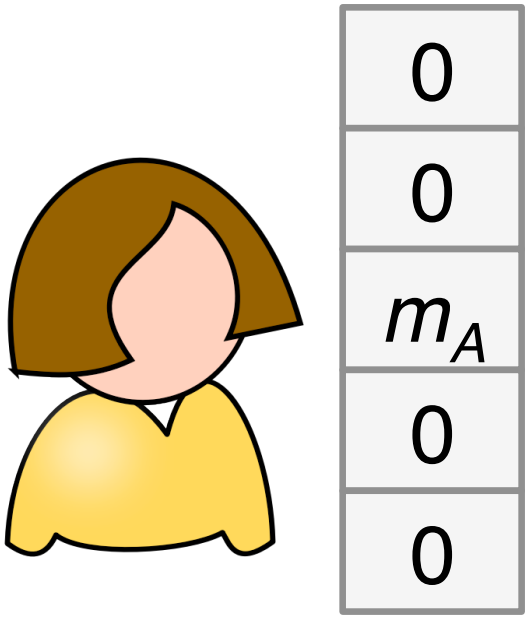
“Straw man” Scheme



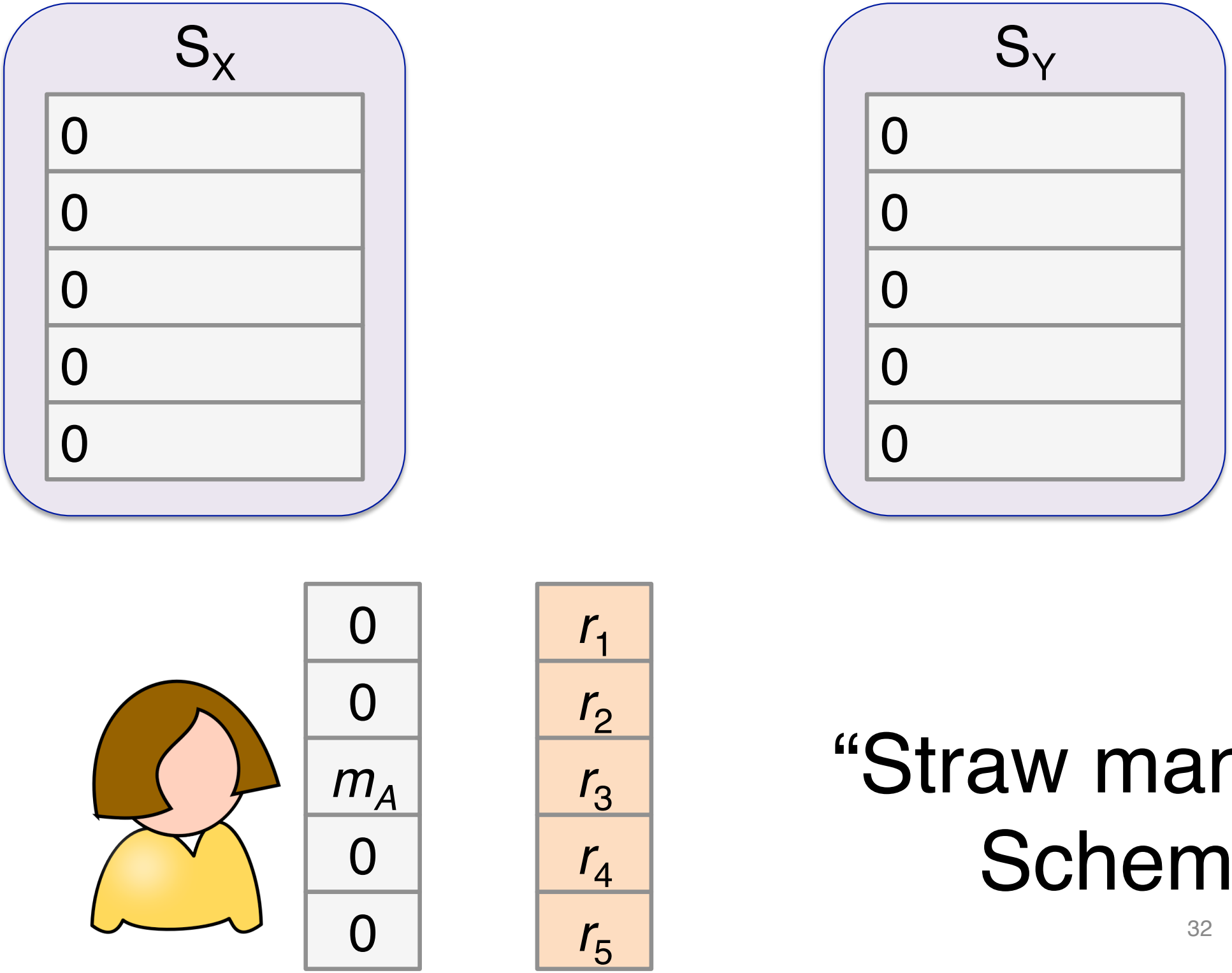
"Straw man" Scheme



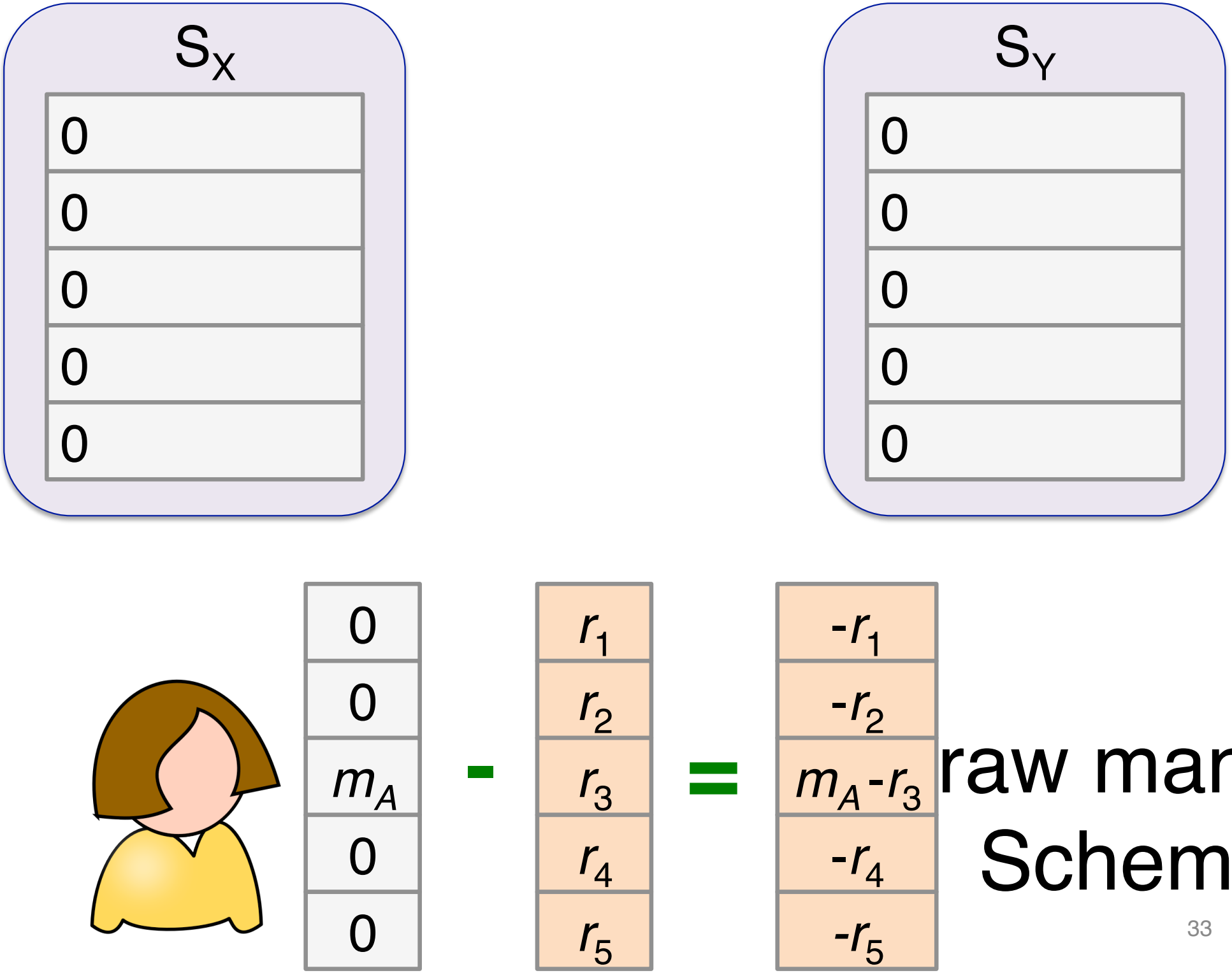


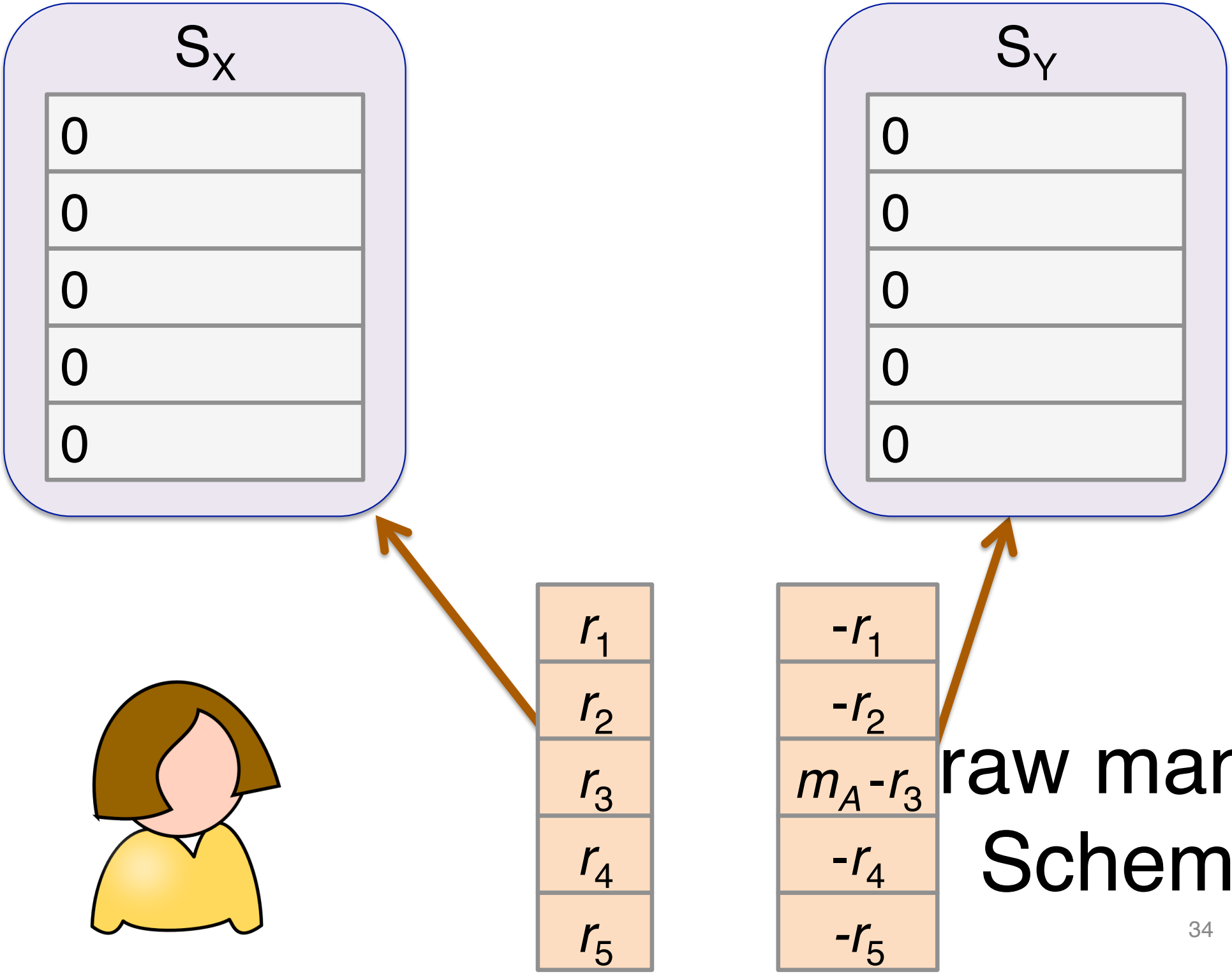


"Straw man" Scheme

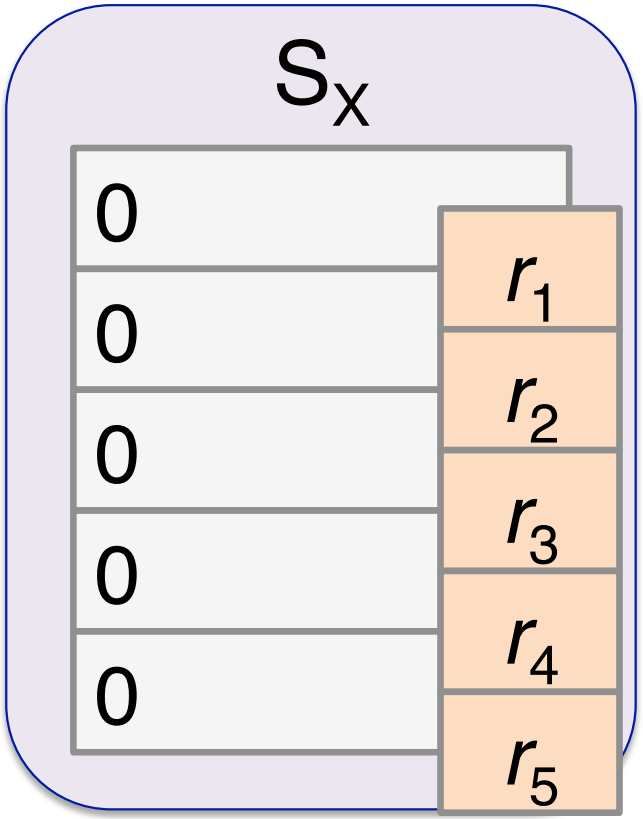


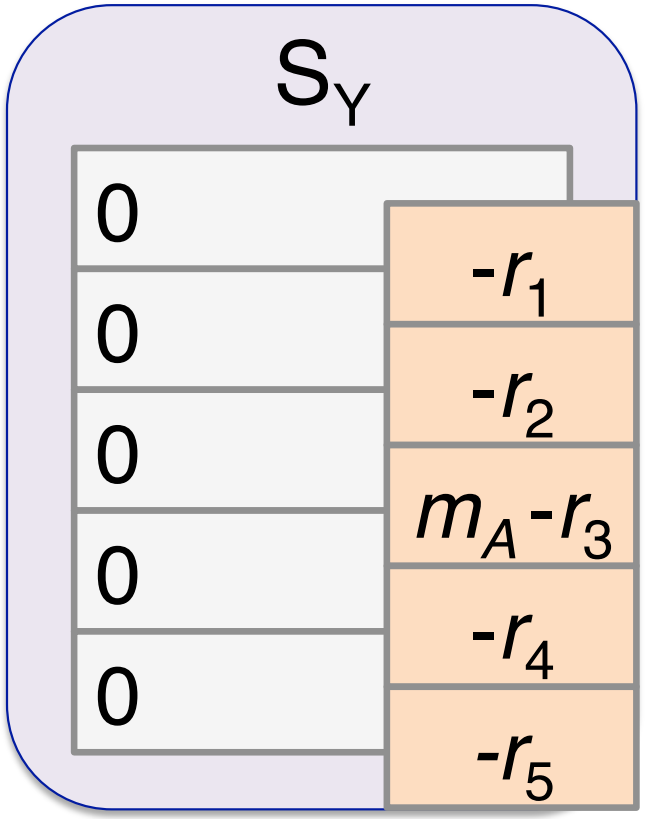
"Straw man" Scheme

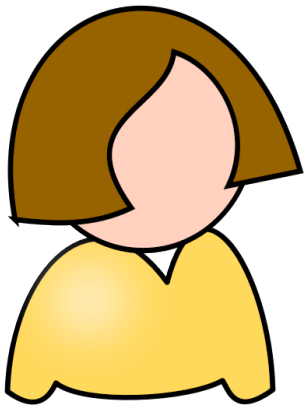

S_X
0
0
0
0
0
S_Y
0
0
0
0
0
0
0
m_A
0
0
-
r_1
r_2
r_3
r_4
r_5
=
$-r_1$
$-r_2$
m_A-r_3
$-r_4$
$-r_5$
raw man”
Scheme



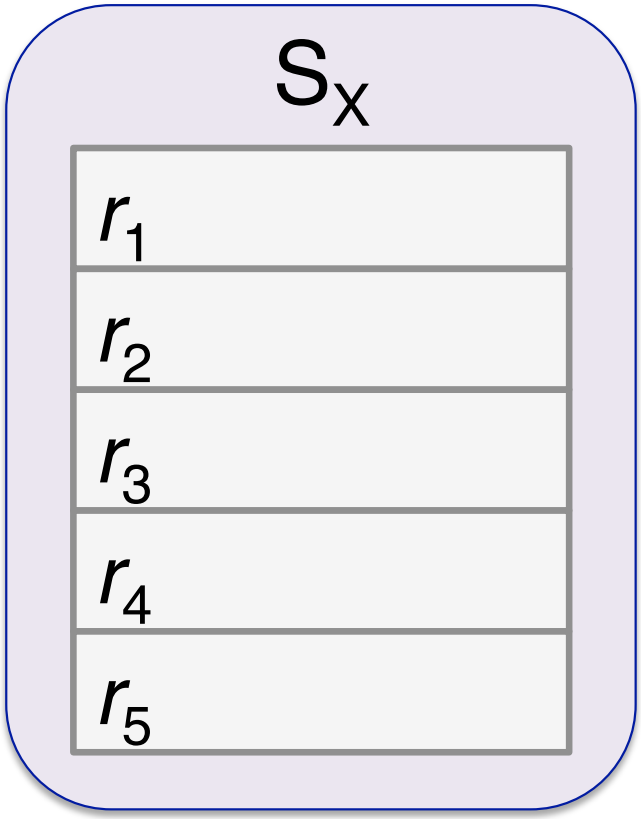
raw man"
Scheme

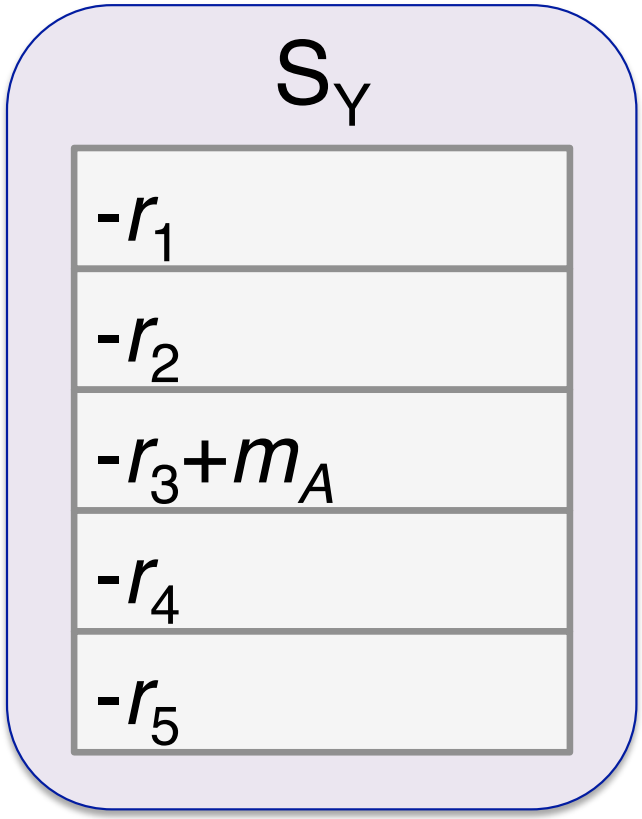






"Straw man" Scheme





"Straw man" Scheme

S_X
r_1
r_2
r_3
r_4
r_5

S_Y
$-r_1$
$-r_2$
$-r_3+m_A$
$-r_4$
$-r_5$

0
0
0
0
m_B

"Straw man" Scheme

S_X
r_1
r_2
r_3
r_4
r_5

S_Y
$-r_1$
$-r_2$
$-r_3+m_A$
$-r_4$
$-r_5$

0
0
0
0
m_B

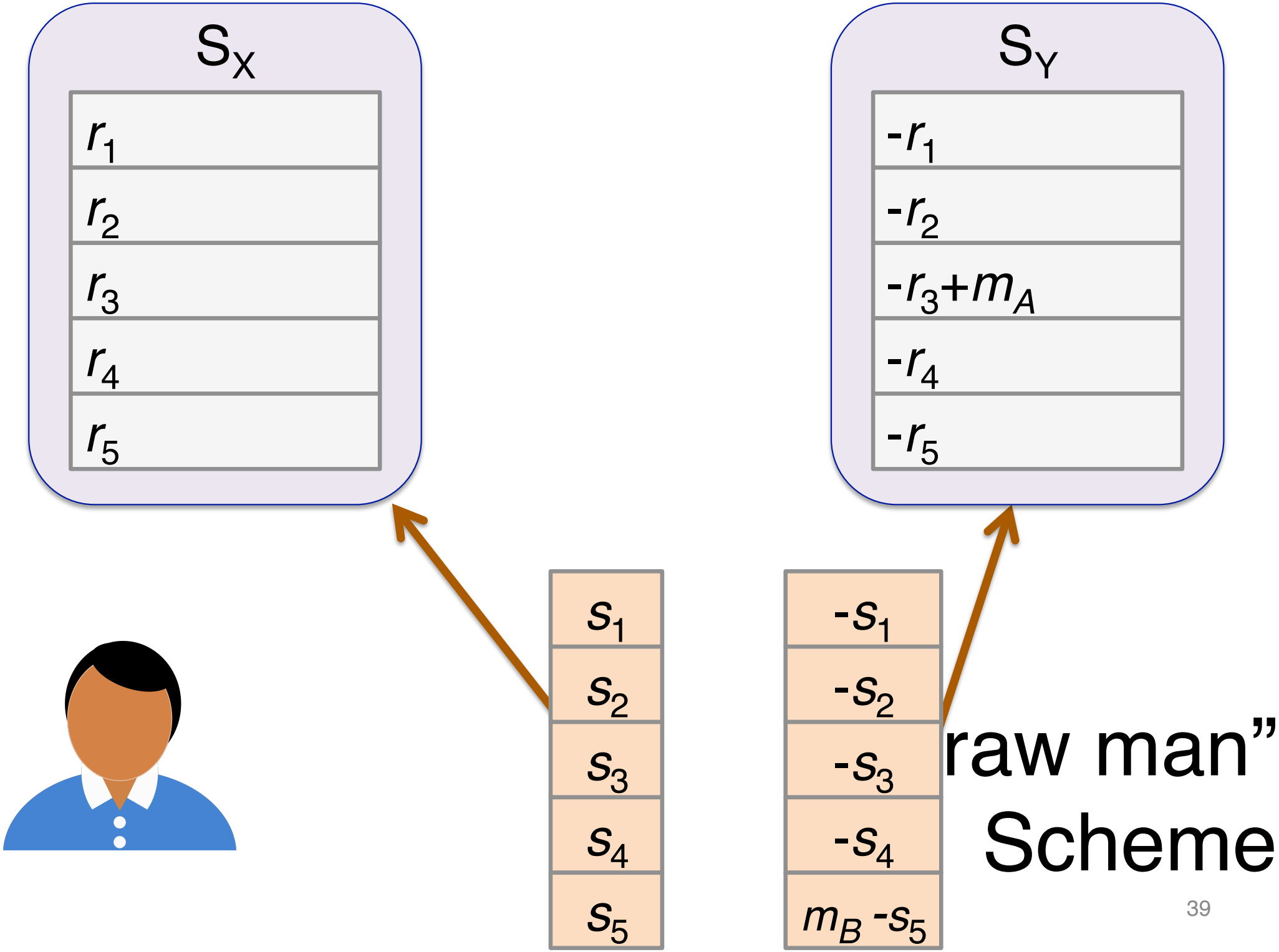
$-$

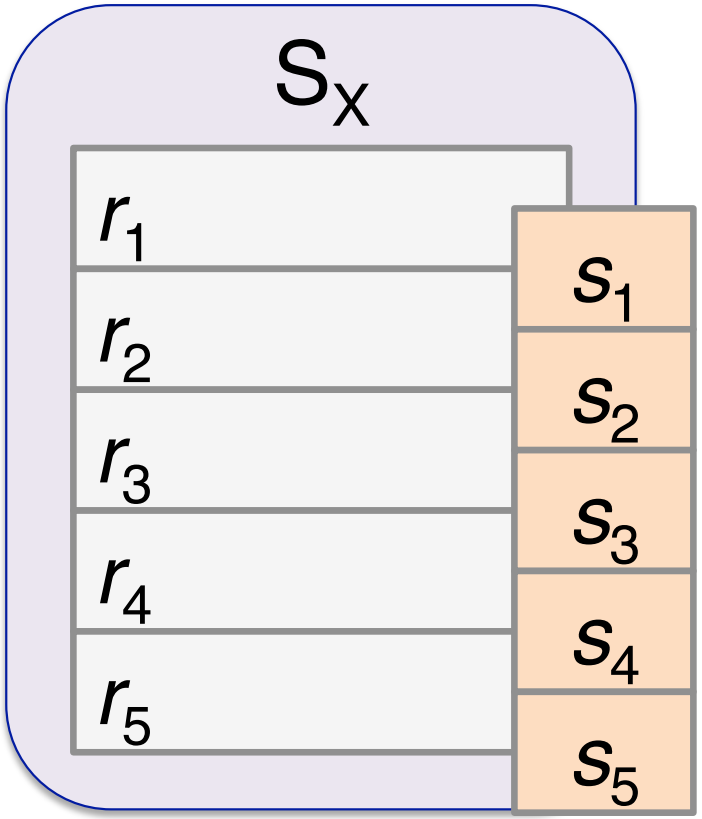
s_1
s_2
s_3
s_4
s_5

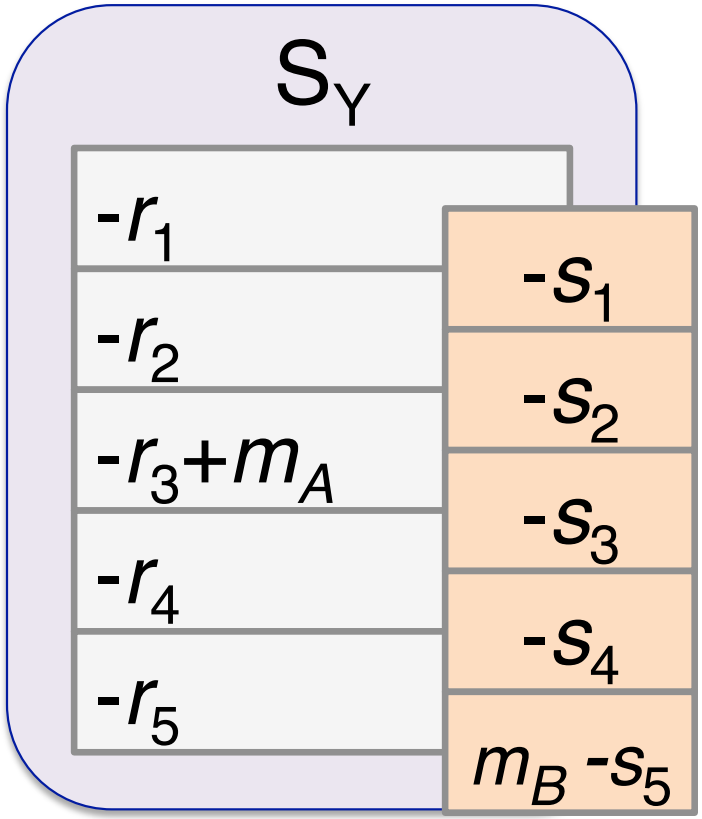
$=$

$-s_1$
$-s_2$
$-s_3$
$-s_4$
m_B-s_5

raw man”
Scheme


S_X
r_1
r_2
r_3
r_4
r_5
S_Y
$-r_1$
$-r_2$
$-r_3+m_A$
$-r_4$
$-r_5$
s_1
s_2
s_3
s_4
s_5
$-s_1$
$-s_2$
$-s_3$
$-s_4$
m_B-s_5
raw man”
Scheme





"Straw man" Scheme

S_X
$r_1 + s_1$
$r_2 + s_2$
$r_3 + s_3$
$r_4 + s_4$
$r_5 + s_5$

S_Y
$-r_1 - s_1$
$-r_2 - s_2$
$-r_3 - s_3 + m_A$
$-r_4 - s_4$
$-r_5 - s_5 - m_B$

"Straw man" Scheme

S_X
$r_1 + s_1$
$r_2 + s_2$
$r_3 + s_3$
$r_4 + s_4$
$r_5 + s_5$

S_Y
$-r_1 - s_1$
$-r_2 - s_2$
$-r_3 - s_3 + m_A$
$-r_4 - s_4$
$-r_5 - s_5 - m_B$

"Straw man" Scheme

S_X
$r_1 + s_1$
$r_2 + s_2$
$r_3 + s_3$
$r_4 + s_4$
$r_5 + s_5$

S_Y
$-r_1 - s_1$
$-r_2 - s_2$
$-r_3 - s_3 + m_A$
$-r_4 - s_4$
$-r_5 - s_5 - m_B$

"Straw man" Scheme

S_X
$r_1 + s_1$
$r_2 + s_2$
$r_3 + s_3$
$r_4 + s_4$
$r_5 + s_5$

S_Y
$-r_1 - s_1$
$-r_2 - s_2$
$-r_3 - s_3 + m_A$
$-r_4 - s_4$
$-r_5 - s_5 - m_B$

"Straw man" Scheme

S_X
$r_1 + s_1$
$r_2 + s_2$
$r_3 + s_3$
$r_4 + s_4$
$r_5 + s_5$

+

S_Y
$-r_1 - s_1$
$-r_2 - s_2$
$-r_3 - s_3 + m_A$
$-r_4 - s_4$
$-r_5 - s_5 - m_B$

=

0
0
m_A
0
m_B

At the end of the day, servers combine DBs to reveal plaintext

"Straw man" Scheme

First-Attempt Scheme: Properties

Correctness

— By construction

Write-Anonymity

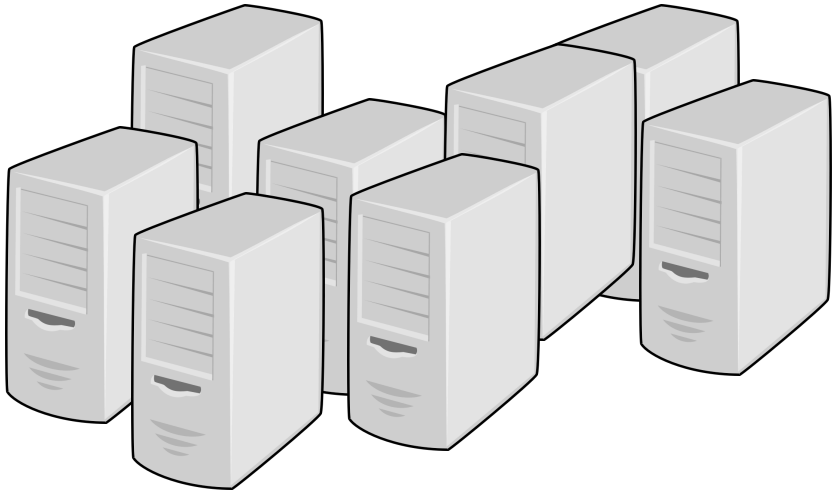
— Given output vector, servers can simulate their view of the protocol run

Practical Efficiency

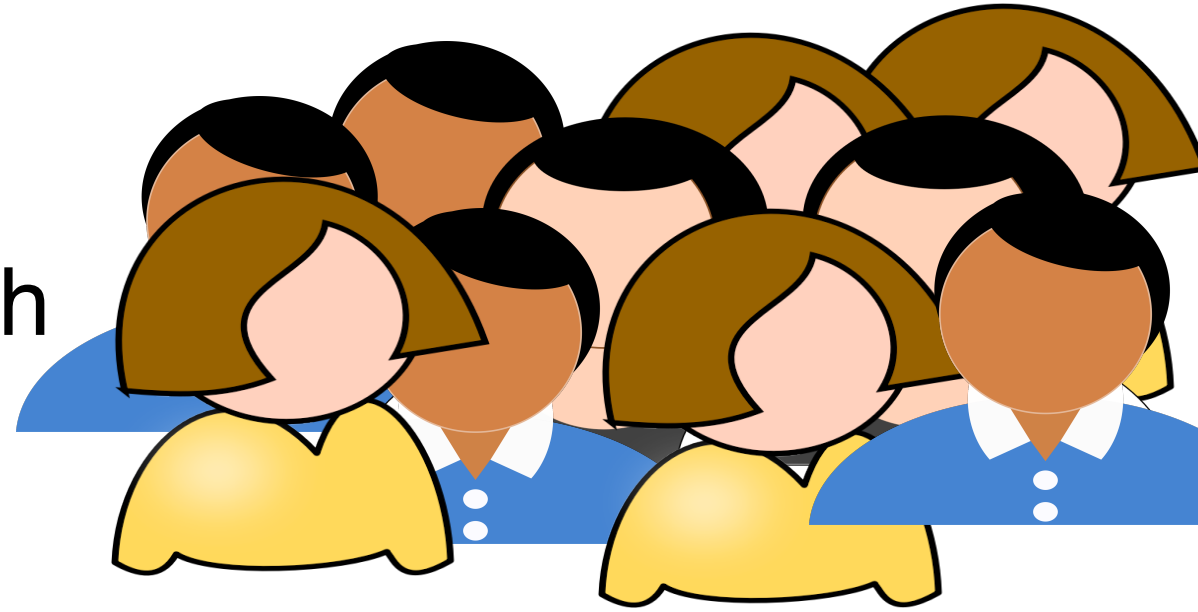
— Almost no “heavy” computation involved

Extensions

Use k > 2 servers
→ secure against
k-1 evil servers



Use a large-
characteristic field
→ e.g., email-length
rows



Outline

- Motivation
- Definitions and a “Straw man” scheme
- **Technical challenges**
- Evaluation
- Conclusions

Limitations of the “Straw man”

1. O(L) communication cost
2. Collisions
3. Malicious clients

Limitations of the “Straw man”

1. **O(*L*) communication cost**
2. Collisions
3. Malicious clients

Challenge 1: Bandwidth Efficiency

In “straw man” design, client sends DB-sized vector to each server

Idea: run PIR protocol *in reverse* to write into DB while sending fewer bits

PIR-in-reverse used in Ostrovsky-Shoup ’97 in single-client context

We extend their results to a many-client context (with malicious clients)

(k,t)-Distributed Point Functions

- We use a generalization of "DPFs" defined by Gilboa and Ishai (2014)
- Many one-round-trip PIR protocols
  citly

Goal: $|q_i| \ll |\mathbf{x}_i|$ for all i

$m \in \mathbb{F}\ ;\ \ell \in [L]$

$$(q_1, \ldots, q_k) \leftarrow \mathsf{KeyGen}(m, \ell)$$

$$\mathbf{x}_i \leftarrow \mathsf{Eval}(q_i) \qquad \in \mathbb{F}^L$$

(k,t)-Distributed Point Functions

Correctness:

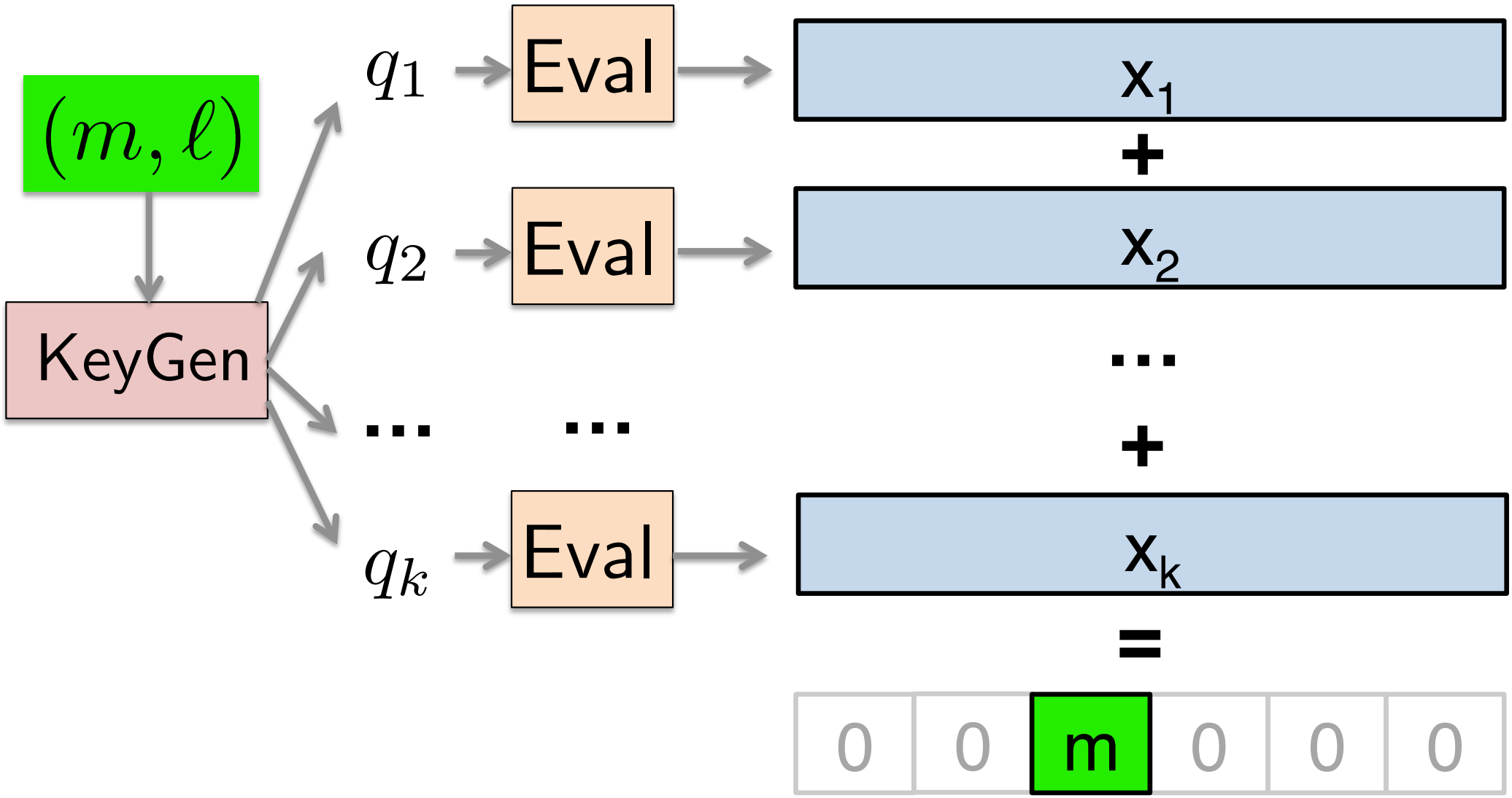
$$(q_1, \ldots, q_k) \leftarrow \mathsf{Ke}$$

Sum of the Eval() outputs will be zero everywhere, except at position ℓ

$$m \cdot \mathbf{e}_\ell \leftarrow \sum_{i=1}^{k} \mathsf{Eval}(q_i)$$

(k,t)-Privacy: [In a minute]

DPF Correctness



(k,t)-Distributed Point Functions

(k,t)-Privacy: Can simulate the distribution of any subset *S* of at most *t* DPF keys

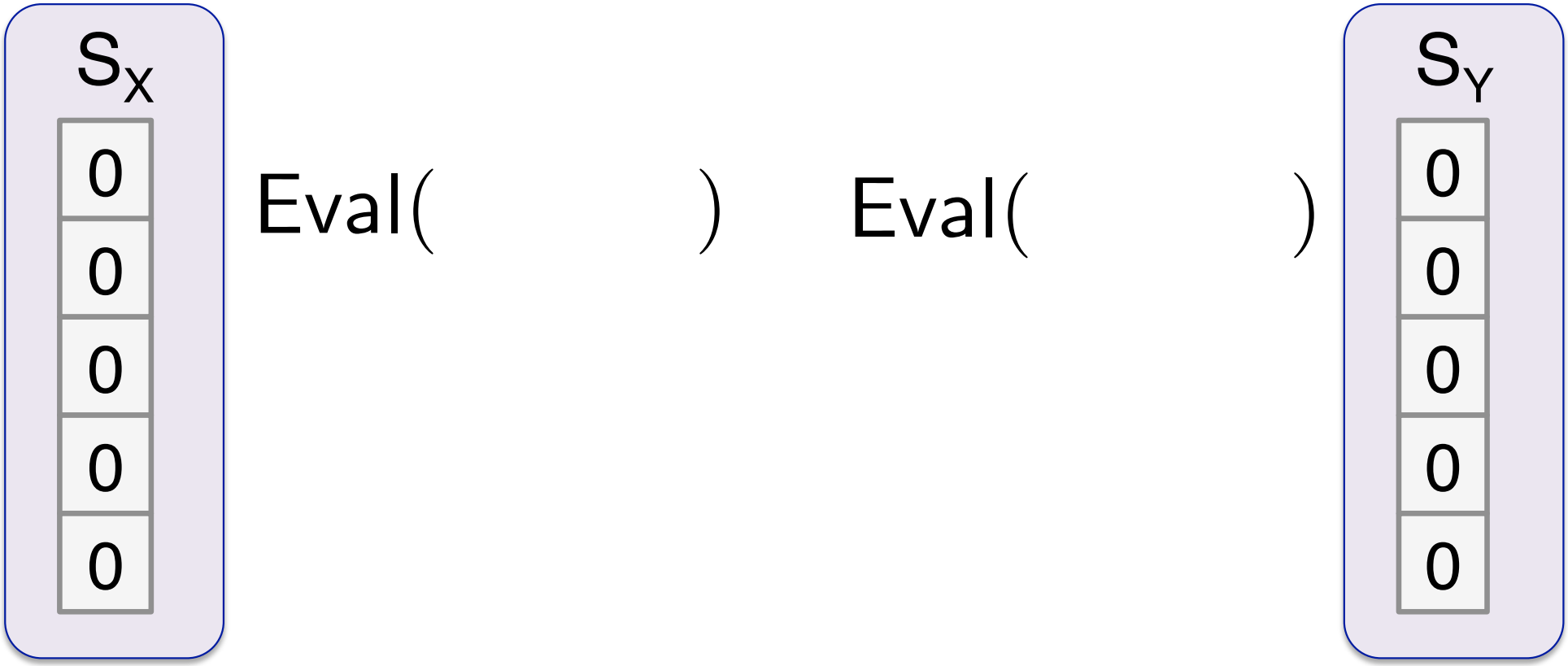
$$(q_1, \ldots, q_k) \leftarrow \mathsf{KeyGen}(m, \ell)$$

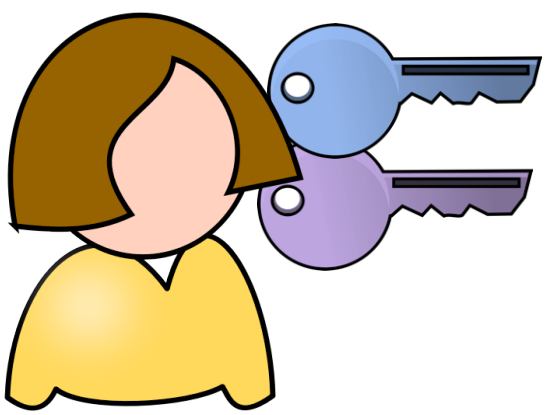
$$\{q_i\}_{i \in S} \approx_c \mathsf{Sim}(S)$$

$\forall S \subset [k]$

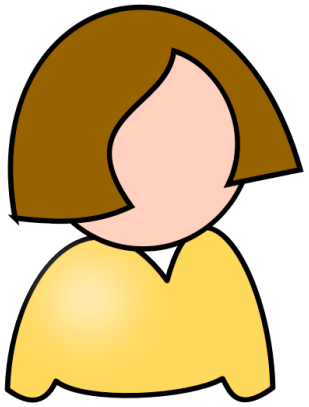
s.t. $|S| \leq t$

[**Intuition**: *t* keys leak nothing about *m* or ℓ]

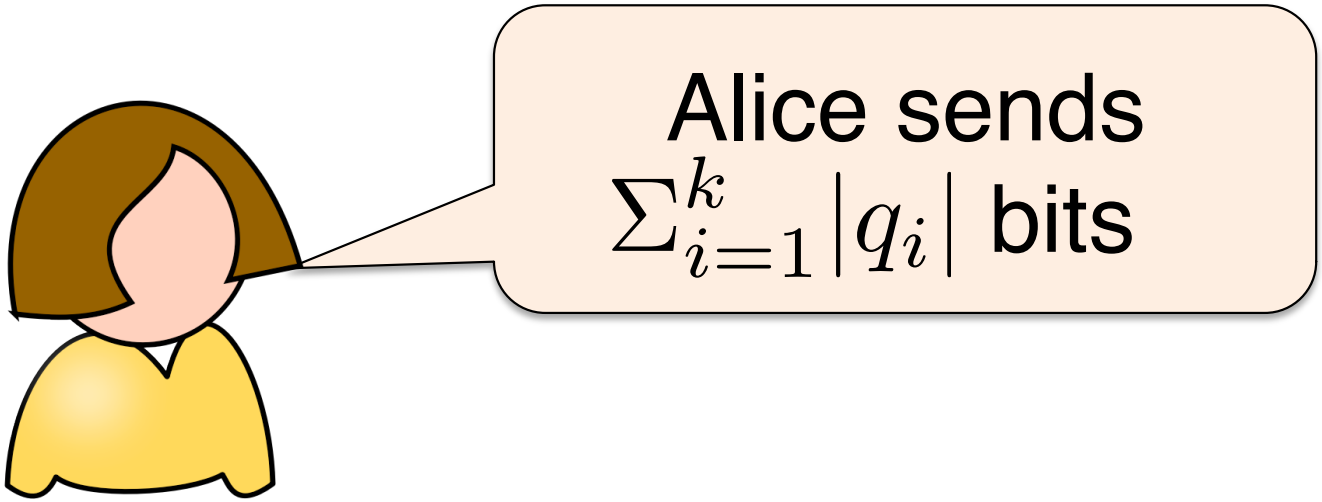




DPFs Reduce Bandwidth Cost



DPFs Reduce Bandwidth Cost


Alice sends
$\Sigma_{i=1}^{k}|q_i|$ bits

Challenge 1: Bandwidth Efficiency

We show: a (***k***, ***t***)-private DPF

yields

a ***k***-server write-anonymous DB scheme tolerating up to ***t*** malicious servers

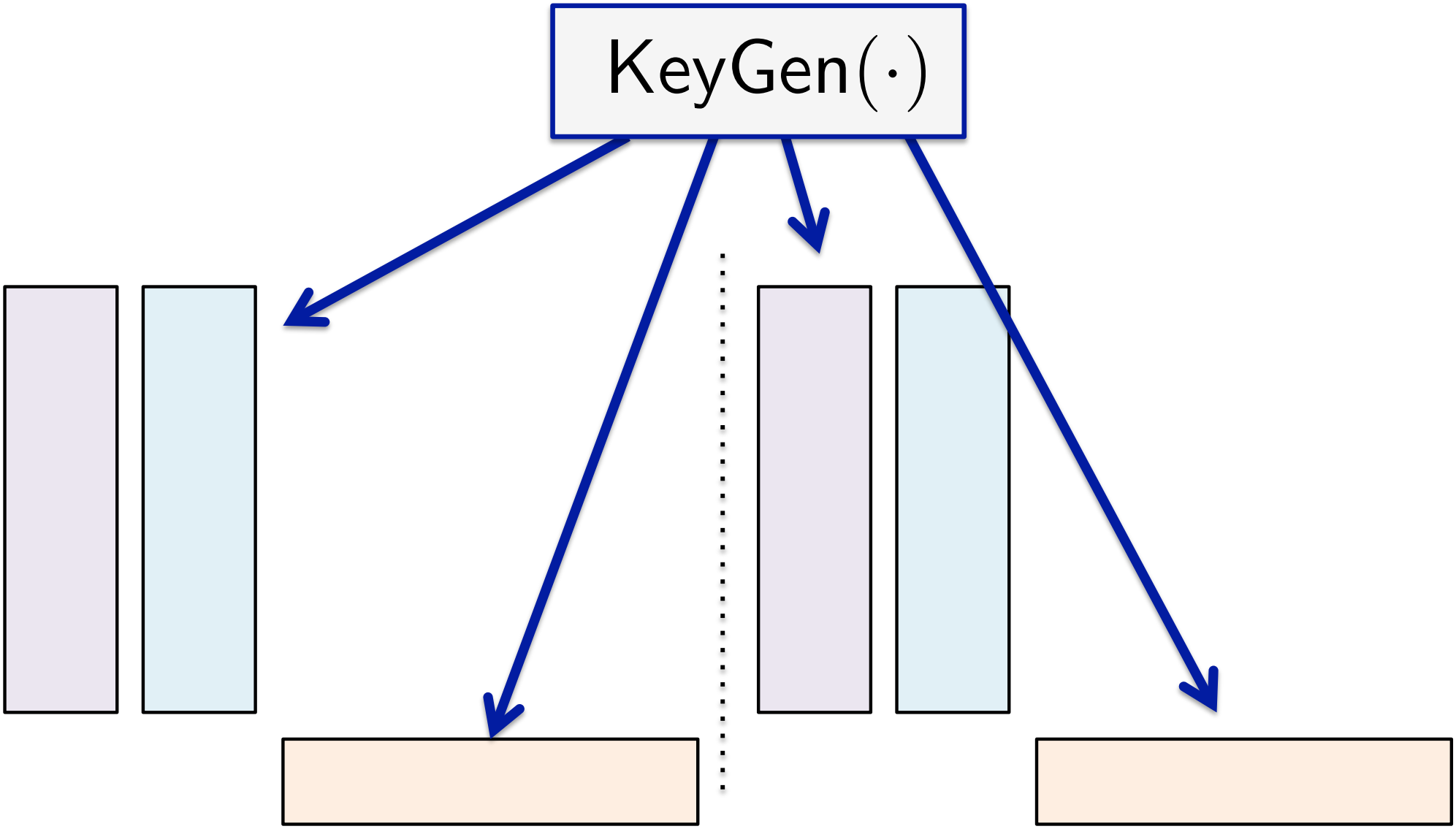
I will present a (2,1)-DPF with $O(L^{1/2})$-length keys based on PIR of Chor and Gilboa ('97)

(2,1)-DPF Construction

Idea: – Represent Eval() output as a matrix

– Keys can be length of side

(2,1)-DPF Construction

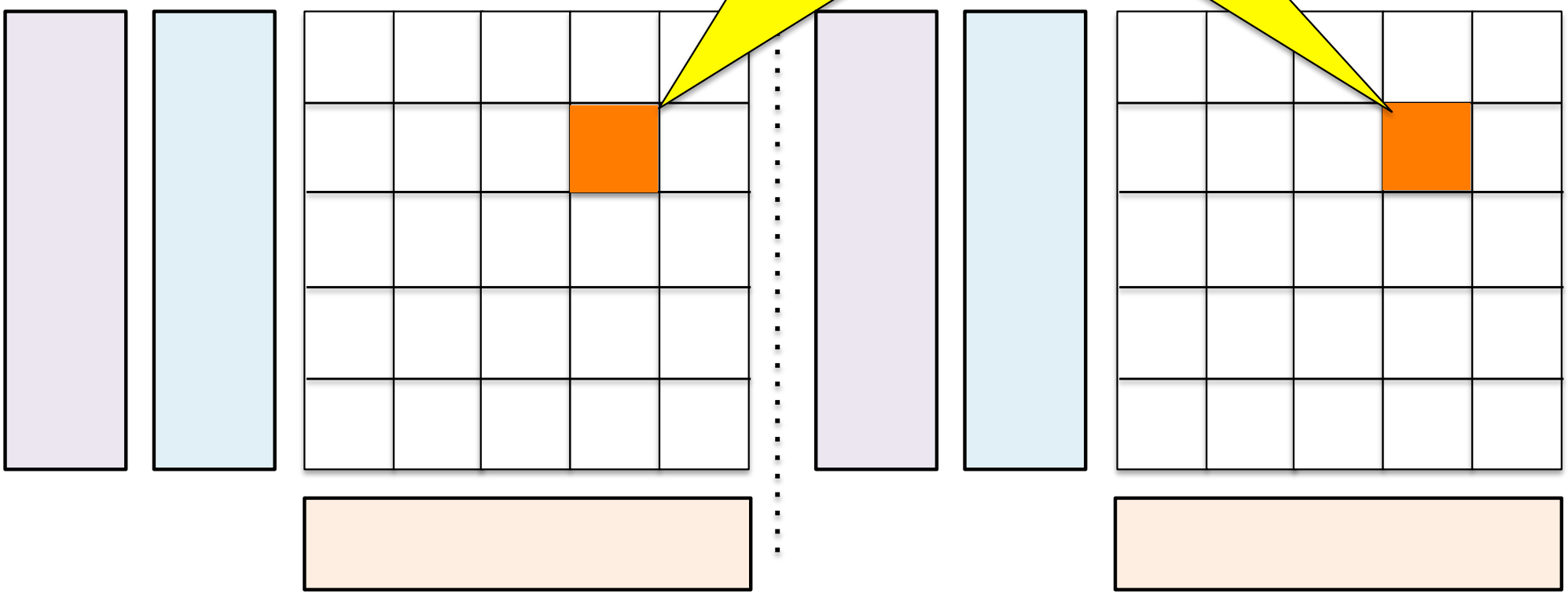


(2,1)-DPF Construction

Idea: – Represent rix

– Keys can b

Output will sum to ***m*** at $\ell = (i, j)$

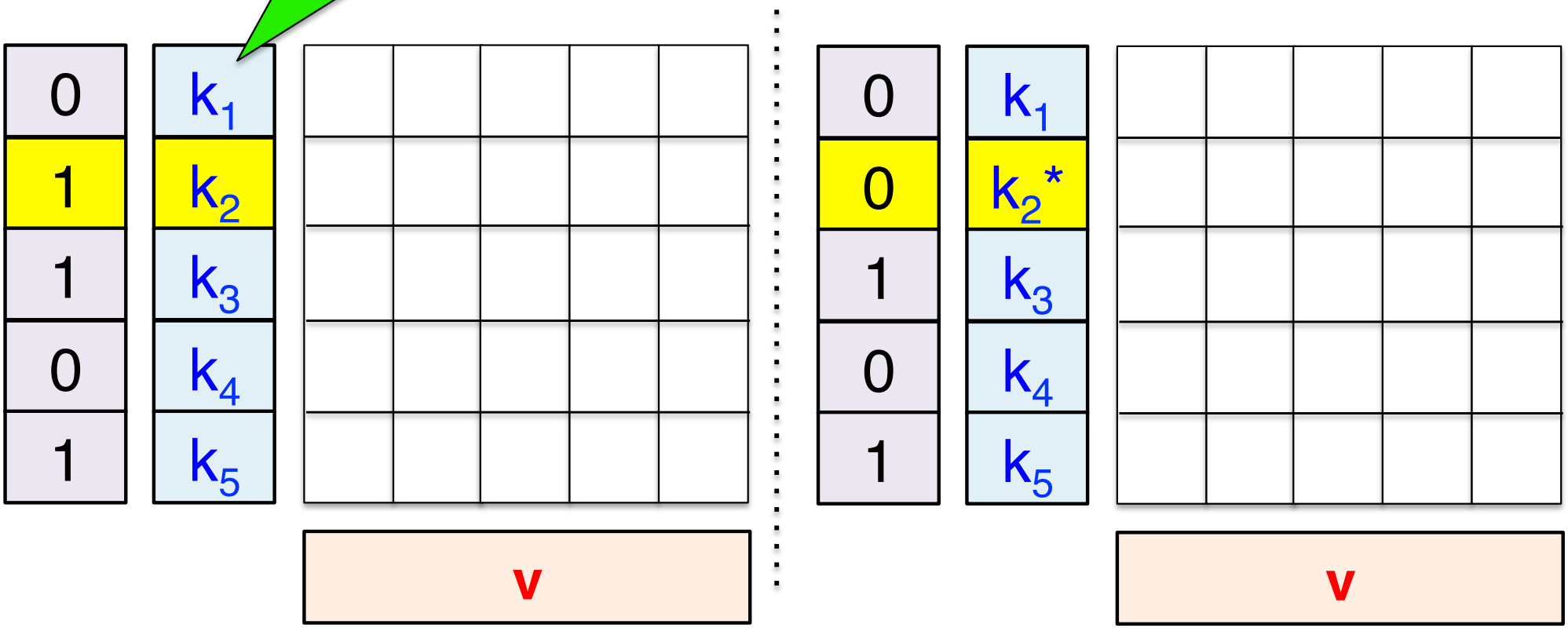


(2,1)-DPF Construction

Sampled at random

Using

Key = ( ), where each has length $O(\sqrt{L})$



(2,1)-DPF Construction

G() is a PRG mapping keys ***k*** to $L^{1/2}$ bits

0	k_1	$G(k_1)$
1	k_2	$G(k_2)$
1	k_3	$G(k_3)$
0	k_4	$G(k_4)$
1	k_5	$G(k_5)$
		v

0	k_1	$G(k_1)$
0	k_2^*	$G(k_2^*)$
1	k_3	$G(k_3)$
0	k_4	$G(k_4)$
1	k_5	$G(k_5)$
		v

(2,1)-DPF Construction

0	k_1	$G(k_1)$
1	k_2	$G(k_2) + v$
1	k_3	$G(k_3) + v$
0	k_4	$G(k_4)$
1	k_5	$G(k_5) + v$

v

0	k_1	$G(k_1)$
0	k_2^*	$G(k_2^*)$
1	k_3	$G(k_3) + v$
0	k_4	$G(k_4)$
1	k_5	$G(k_5) + v$

v

(2,1)-DPF Construction

Outputs are equal everywhere except at row 2

0	k_1	$G(k_1)$
1	k_2	$G(k_2) + v$
1	k_3	$G(k_3) + v$
0	k_4	$G(k_4)$
1	k_5	$G(k_5) + v$
		v

0	k_1	$G(k_1)$
0	k_2^*	$G(k_2^*)$
1	k_3	$G(k_3) + v$
0	k_4	$G(k_4)$
1	k_5	$G(k_5) + v$
		v

(2,1)-DPF Construction

Outputs sum to zero everywhere except at row 2

0	k_1	$G(k_1)$
1	k_2	$G(k_2) + \mathbf{v}$
1	k_3	$G(k_3) + \mathbf{v}$
0	k_4	$G(k_4)$
1	k_5	$G(k_5) + \mathbf{v}$
		$\mathbf{v}$

0	k_1	$G(k_1)$
0	k_2^*	$G(k_2^*)$
1	k_3	$G(k_3) + \mathbf{v}$
0	k_4	$G(k_4)$
1	k_5	$G(k_5) + \mathbf{v}$
		$\mathbf{v}$

(2,1)-DPF Construction

Construct **v** as:

$$\mathbf{v} = G(k_2) + G(k_2^*) + \mathbf{m} \bullet \mathbf{e}_j$$

0	k_1	$G(k_1)$
1	k_2	$G(k_2) + \mathbf{v}$
1	k_3	$G(k_3) + \mathbf{v}$
0	k_4	$G(k_4)$
1	k_5	$G(k_5) + \mathbf{v}$
		v

0	k_1	$G(k_1)$
0	k_2^*	$G(k_2^*)$
1	k_3	$G(k_3) + \mathbf{v}$
0	k_4	$G(k_4)$
1	k_5	$G(k_5) + \mathbf{v}$
		v

$G(k_1)$
$G(k_2) + v$
$G(k_3) + v$
$G(k_4)$
$G(k_5) + v$

+

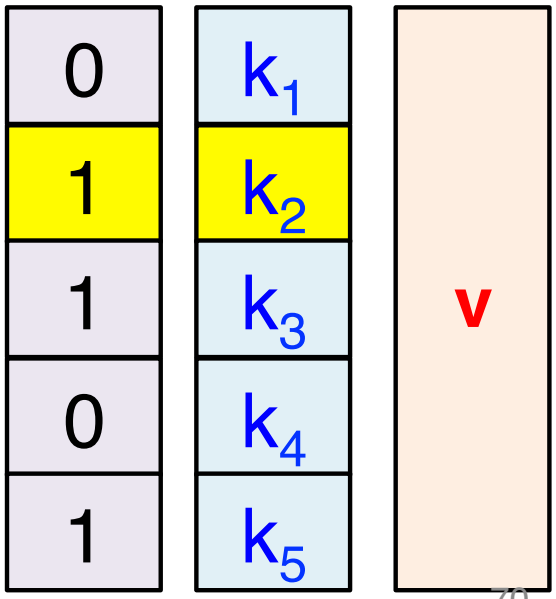
$G(k_1)$
$G(k_2^*)$
$G(k_3) + v$
$G(k_4)$
$G(k_5) + v$

=

00000…00000
0000000m000
00000…00000
00000…00000
00000…00000

Challenge 1: Bandwidth Efficiency

- Brings comm cost down to $O(L^{1/2})$
  - Just requires PRG — fast!
- Recursive application of the same trick
  - Key size down to polylog(L) [GI'14]



New DPF Construction

Given a seed-homomorphic PRG

$$G(\mathbf{s_1}) + G(\mathbf{s_2}) = G(\mathbf{s_1 + s_2})$$

we build a (k, k-1)-private DPF

[NPR'99] [BLMR'13] [BP'14] [BV'15]

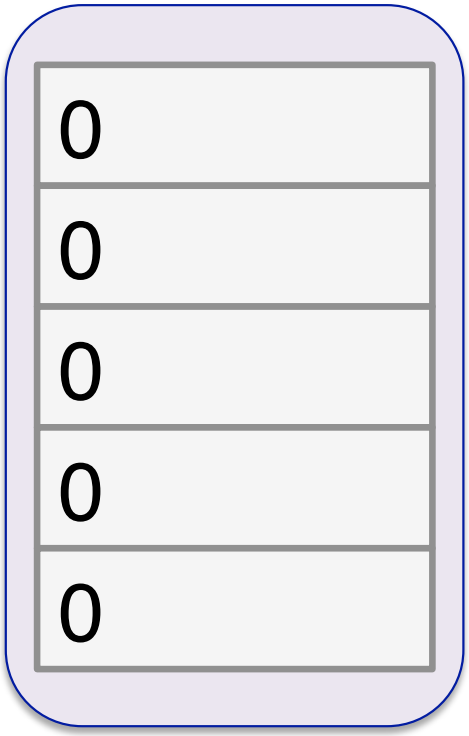
→ Privacy holds even if all but one server is adversarial

Limitations of the “Straw man”

1. O(L) communication cost
2. **Collisions**
3. Malicious clients

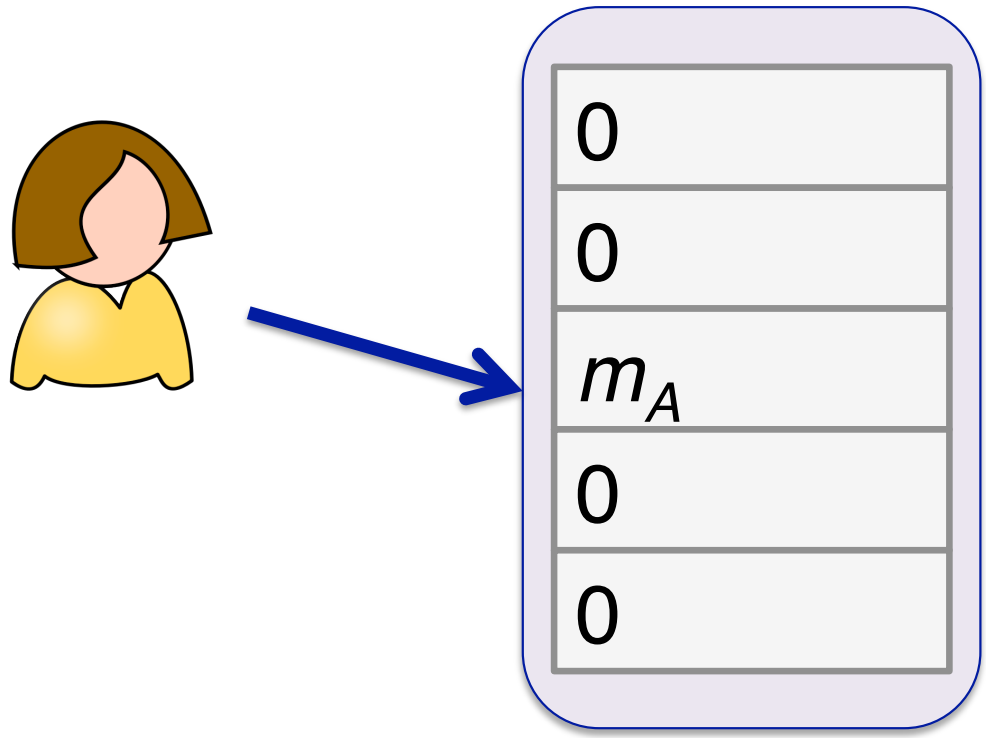
Challenge 2: Collisions

- Clients pick write location ℓ at random
- Two honest clients may write into the same location ℓ



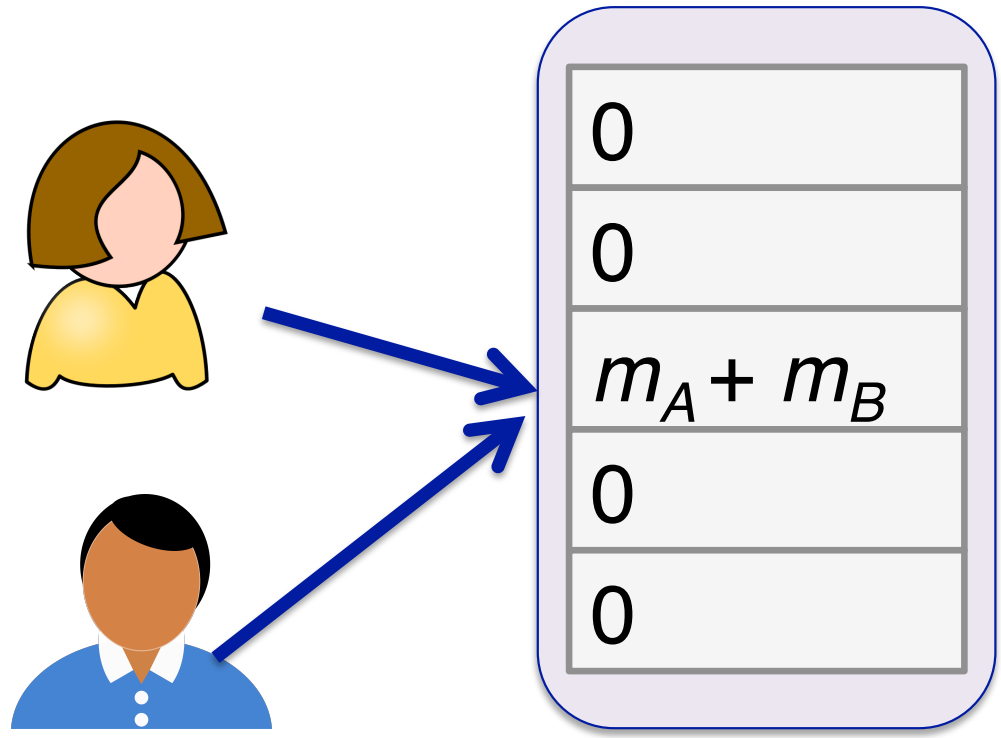
Challenge 2: Collisions

- Clients pick write location ℓ at random
- Two honest clients may write into the same location ℓ



Challenge 2: Collisions

- Clients pick write location ℓ at random
- Two honest clients may write into the same location ℓ

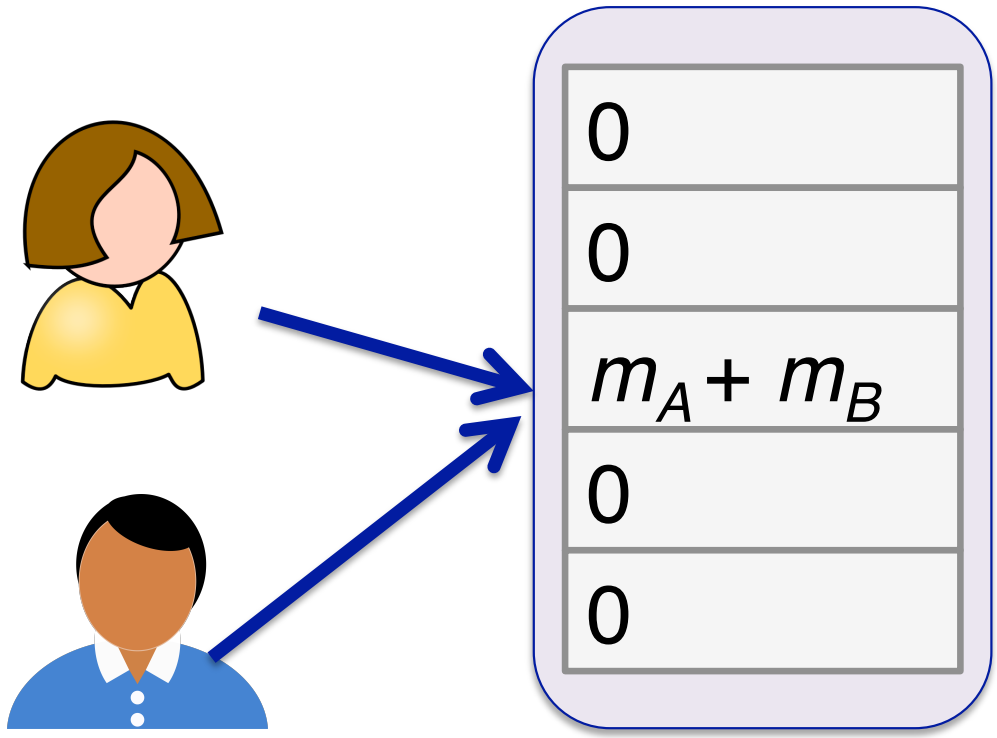


Challenge 2: Collisions

- Clients pick write location ℓ at random
- Two honest clients may write into the same location ℓ

Instead of getting m_A, m_B, get the sum

$$m_A + m_B$$



Challenge 2: Collisions

Straightforward solutio

Make DB table large to avoid collisions

Better solution:

Use coding techniques up to ***d***-way collisions

Key idea: even after a collision, learn the <u>sum</u> of colliding writes $c = m_1 + m_2$

Challenge 2: Collisions

Idea: To handle 2-collisions, can code message ***m*** as: **(m, m^2)**

[Let $\mathrm{char}(\mathbb{F}) > 2$]

After a 2-collision, DBs recover the values:

$$c_1 = m_1 + m_2$$

$$c_2 = m_1^2 + m_2^2$$

Given c_1 and c_2 can recover m_1 and m_2

Challenge 2: Collisions

Using coding technique, can tolerate ***d***-collisions for any ***d***

Reduces table size by 93%

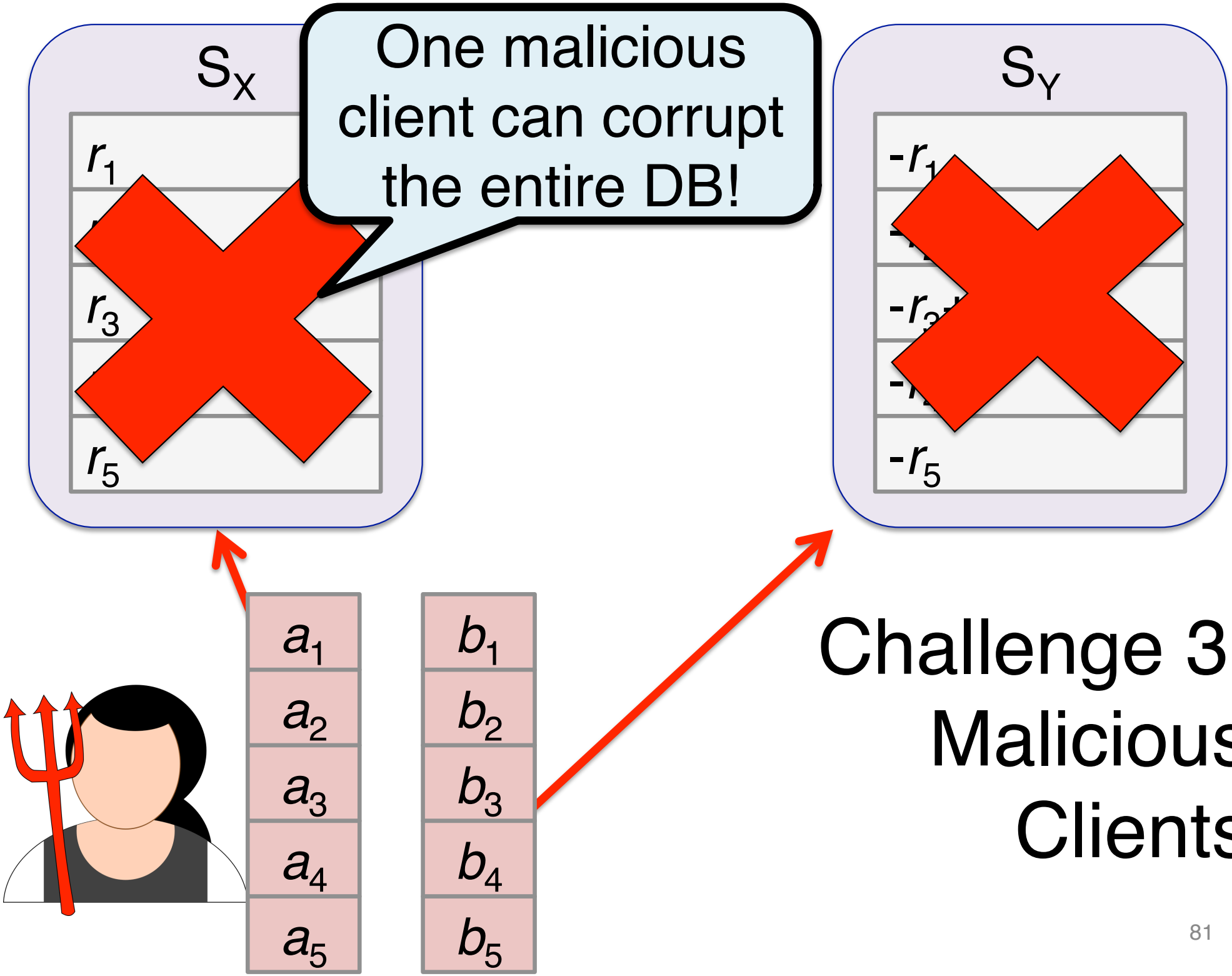
For 1% loss rate, 1k users:

Naive method: 100k cells

Coding method: 6.9k cells

Limitations of the “Straw man”

1. O(L) communication cost
2. Collisions
3. **Malicious clients**



Challenge 3: Malicious Clients

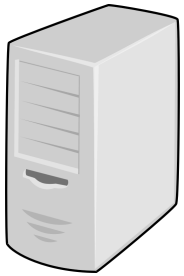
Challenge 3: Malicious Client

Goal: Prevent evil client from destroying DB

- One way to solve this is with NIZKs
  - Expensive public-key crypto [Golle Juels '04]
- More efficient solution:
  - Add a third non-colluding "audit" server to get honest majority
  - Fast, info-theoretic MPC techniques

    [GMW'87], [CCD'88], [FNW'96]

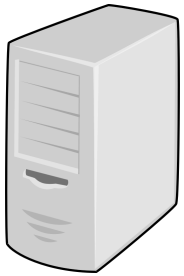
Auditor

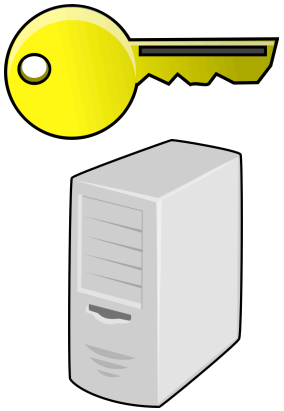


DB Server X

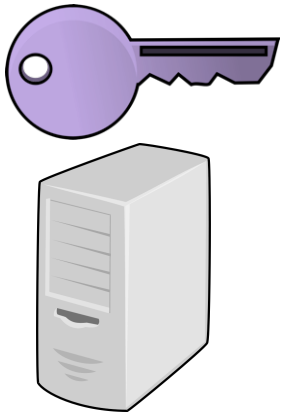
DB Server Y

Auditor



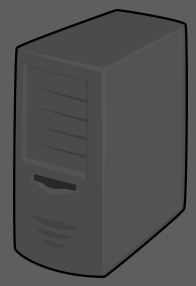


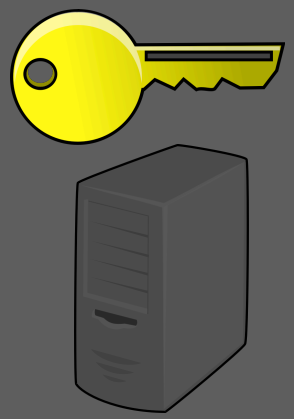
DB Server X



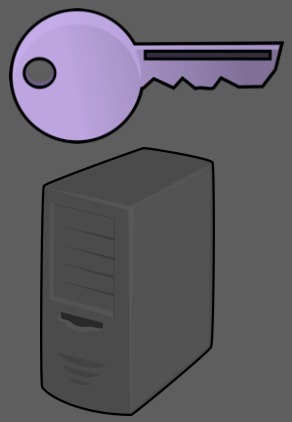
DB Server Y

Auditor

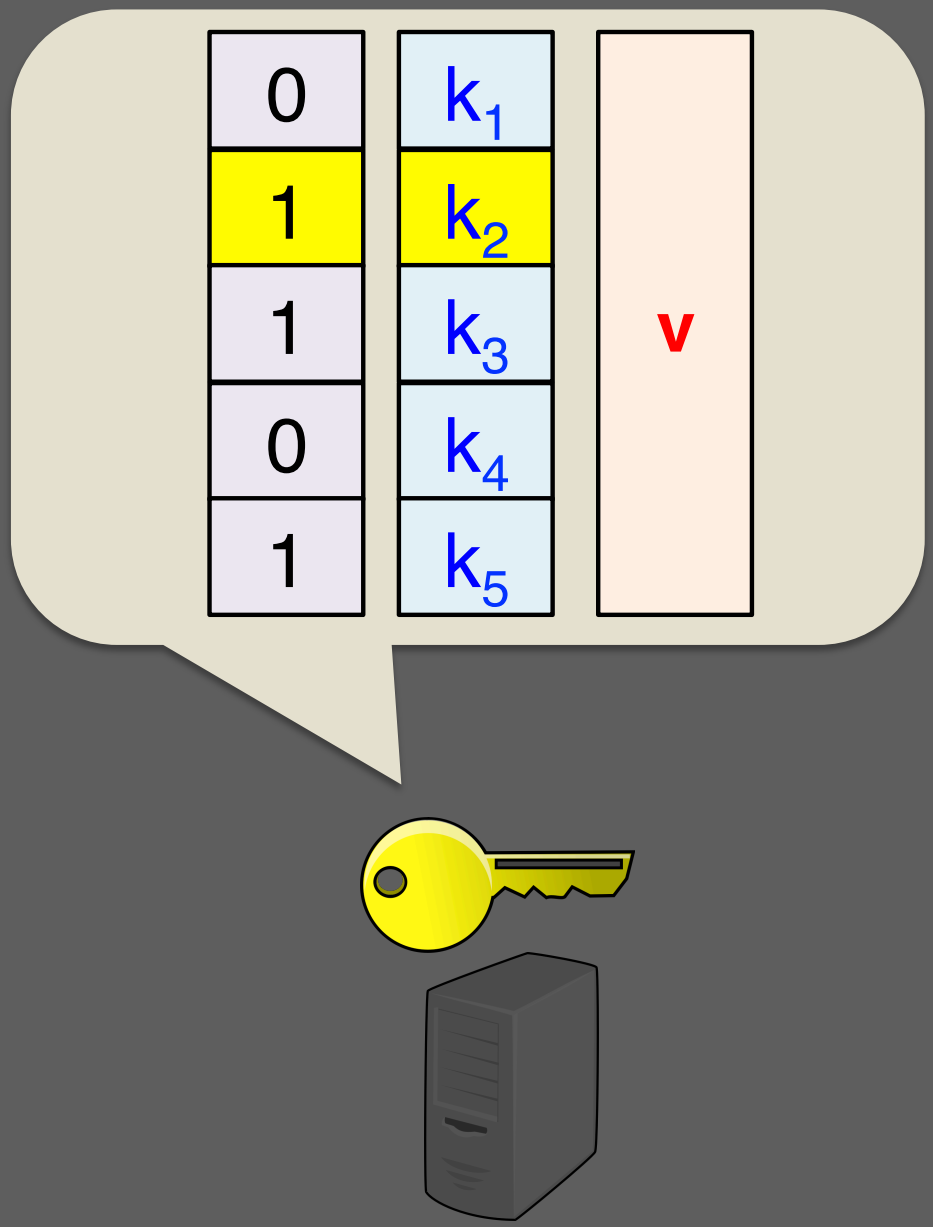


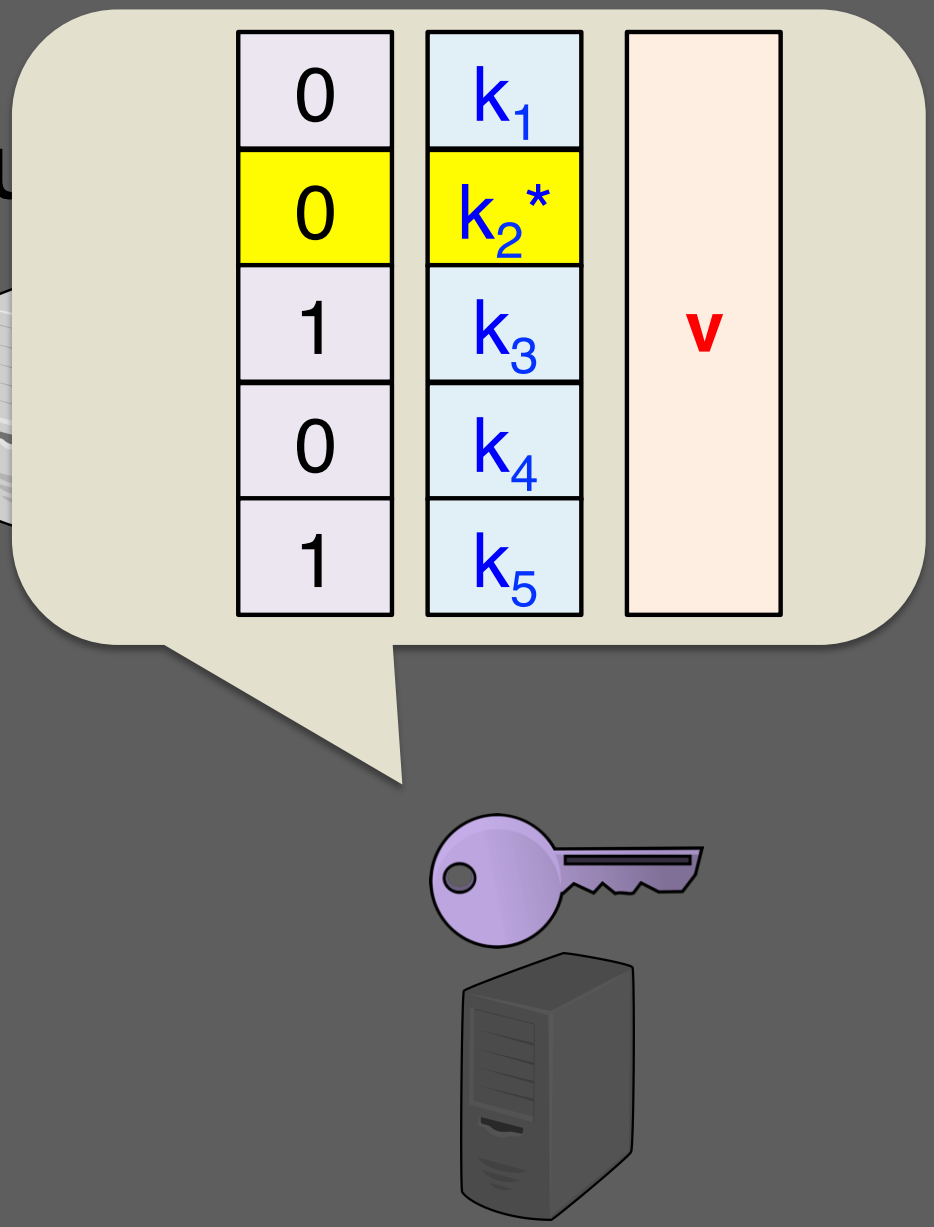


DB Server X



DB Server Y





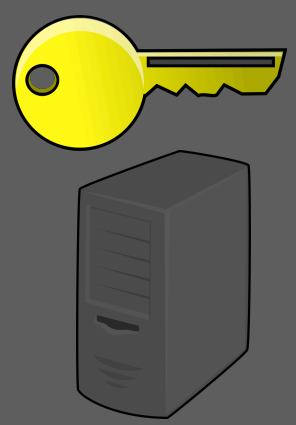
DB Server X

DB Server Y

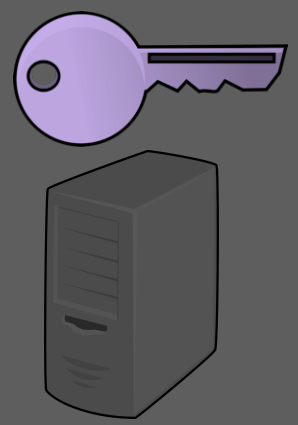
0	k_1
1	k_2
1	k_3
0	k_4
1	k_5

Au

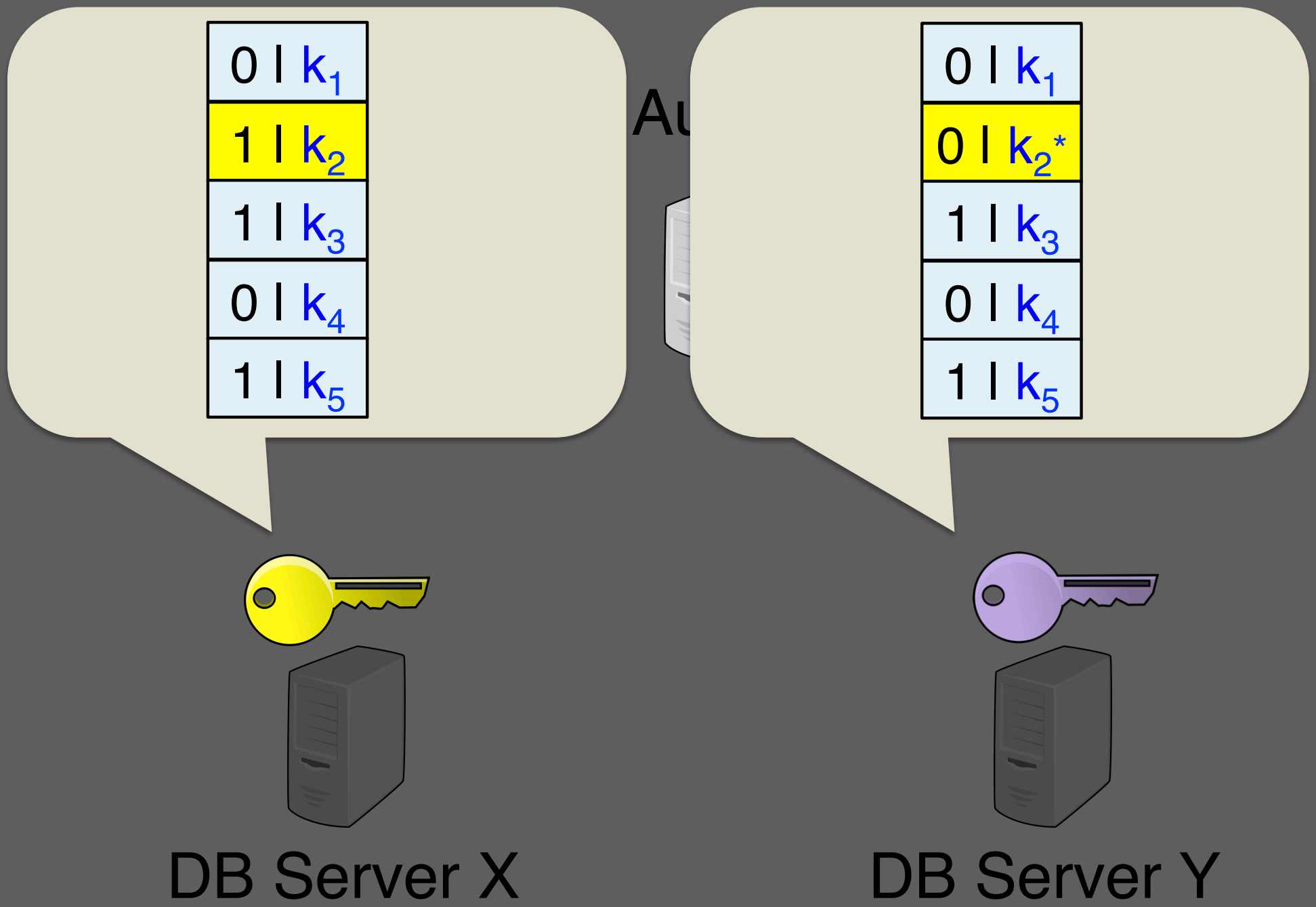
0	k_1
0	k_2*
1	k_3
0	k_4
1	k_5



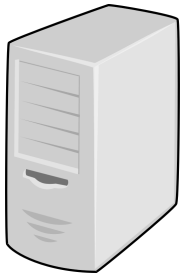
DB Server X



DB Server Y


0 | k_1
1 | k_2
1 | k_3
0 | k_4
1 | k_5
Au
0 | k_1
0 | k_2*
1 | k_3
0 | k_4
1 | k_5
DB Server X
DB Server Y

Auditor

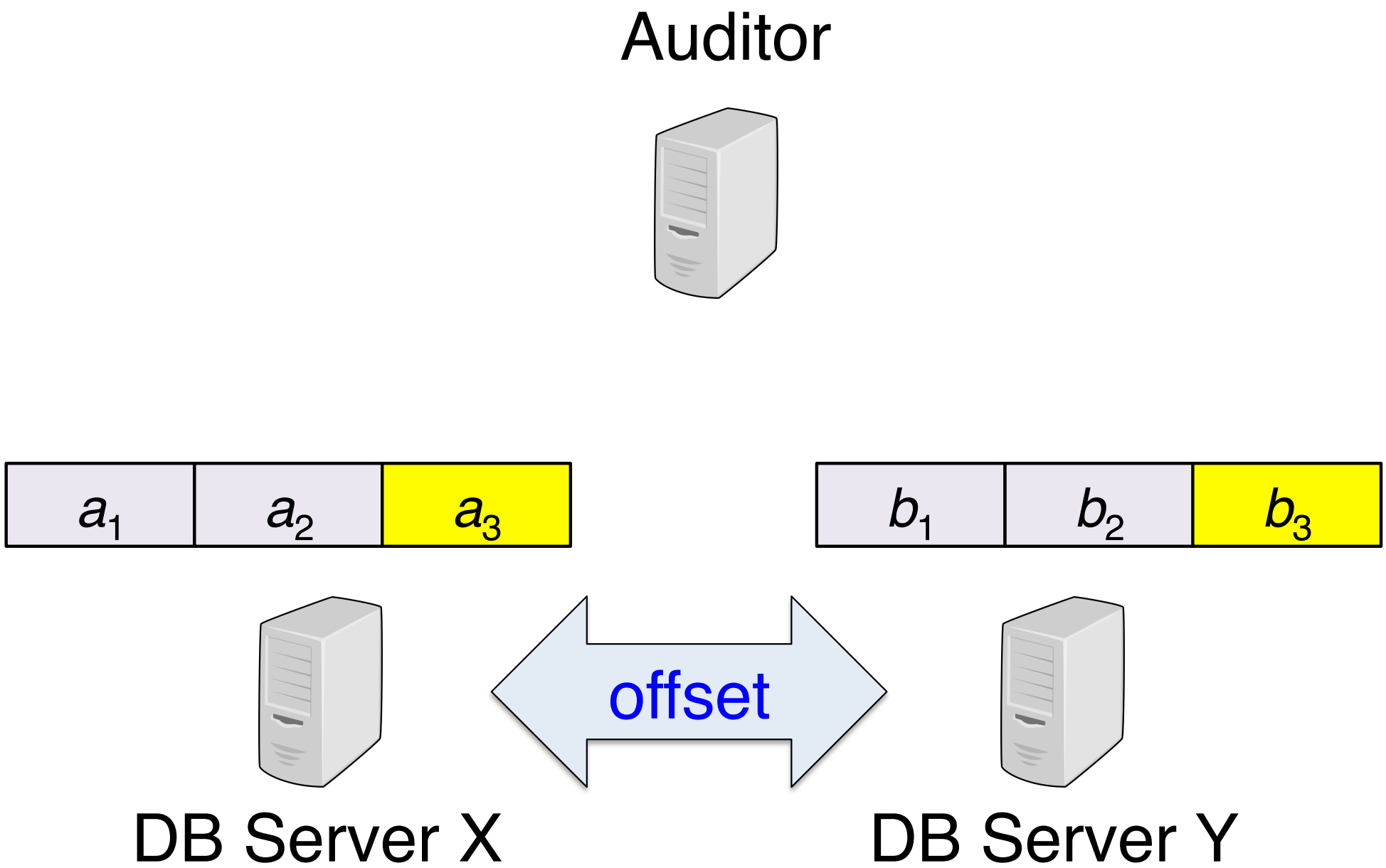


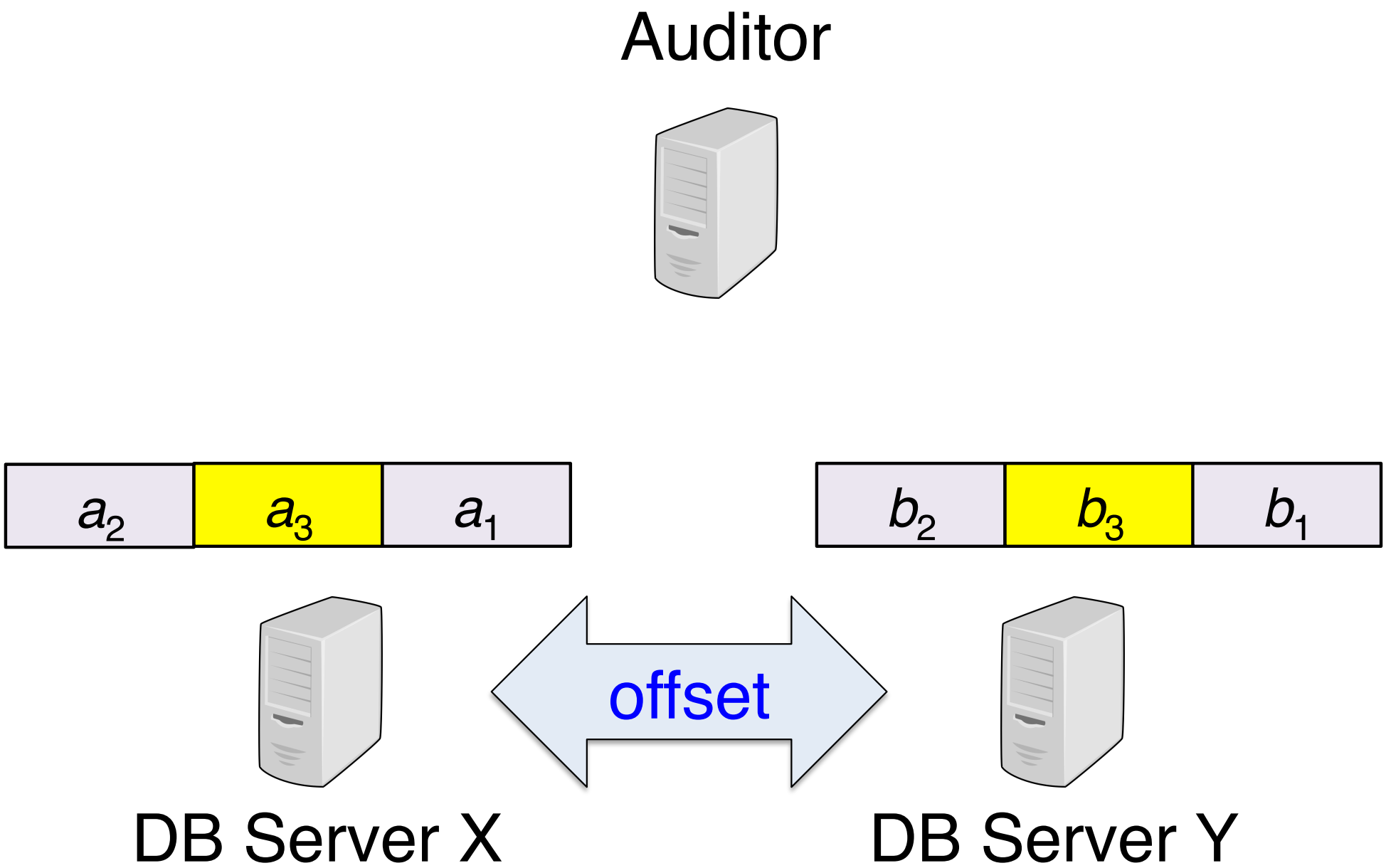
a_1	a_2	a_3

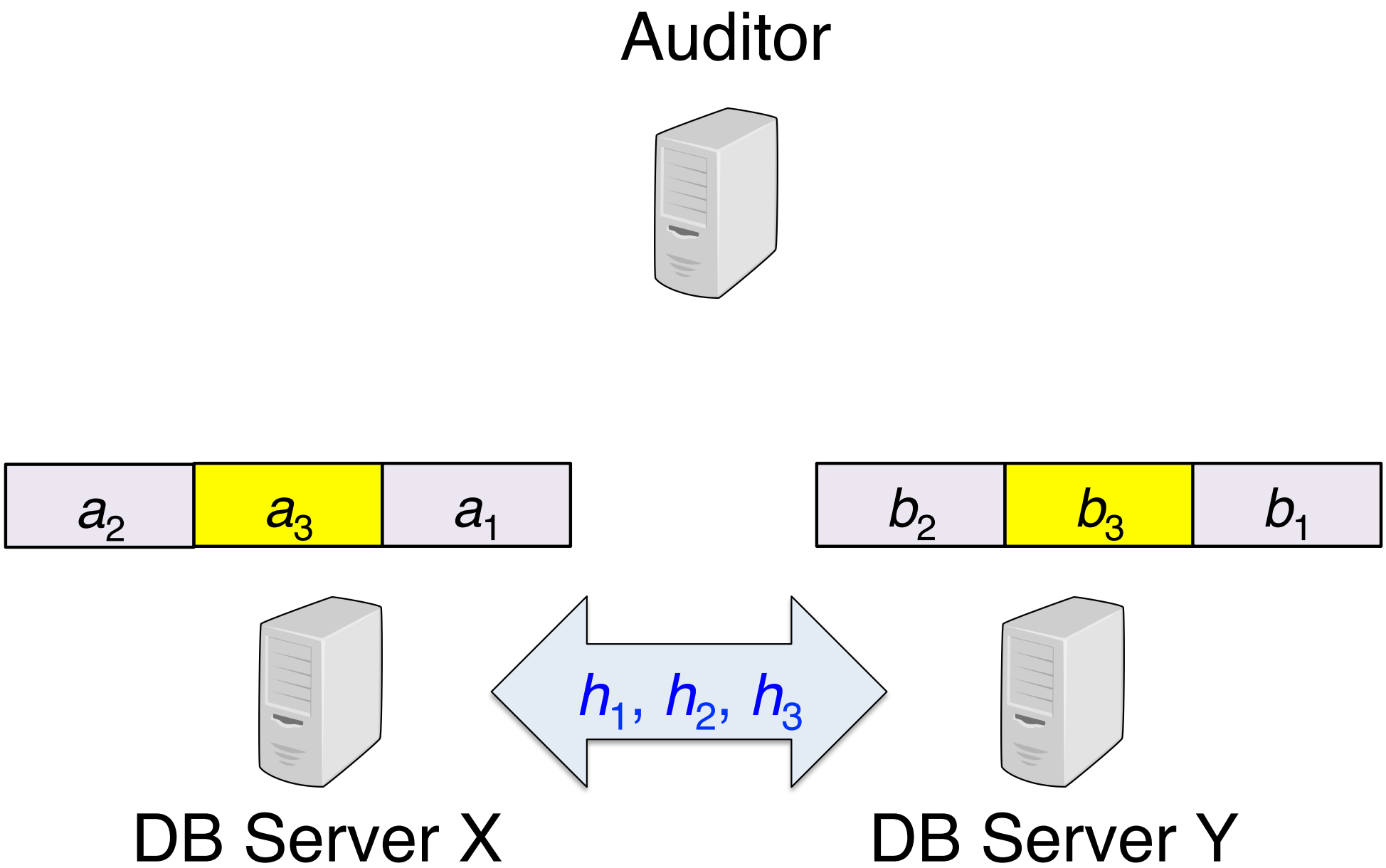
DB Server X

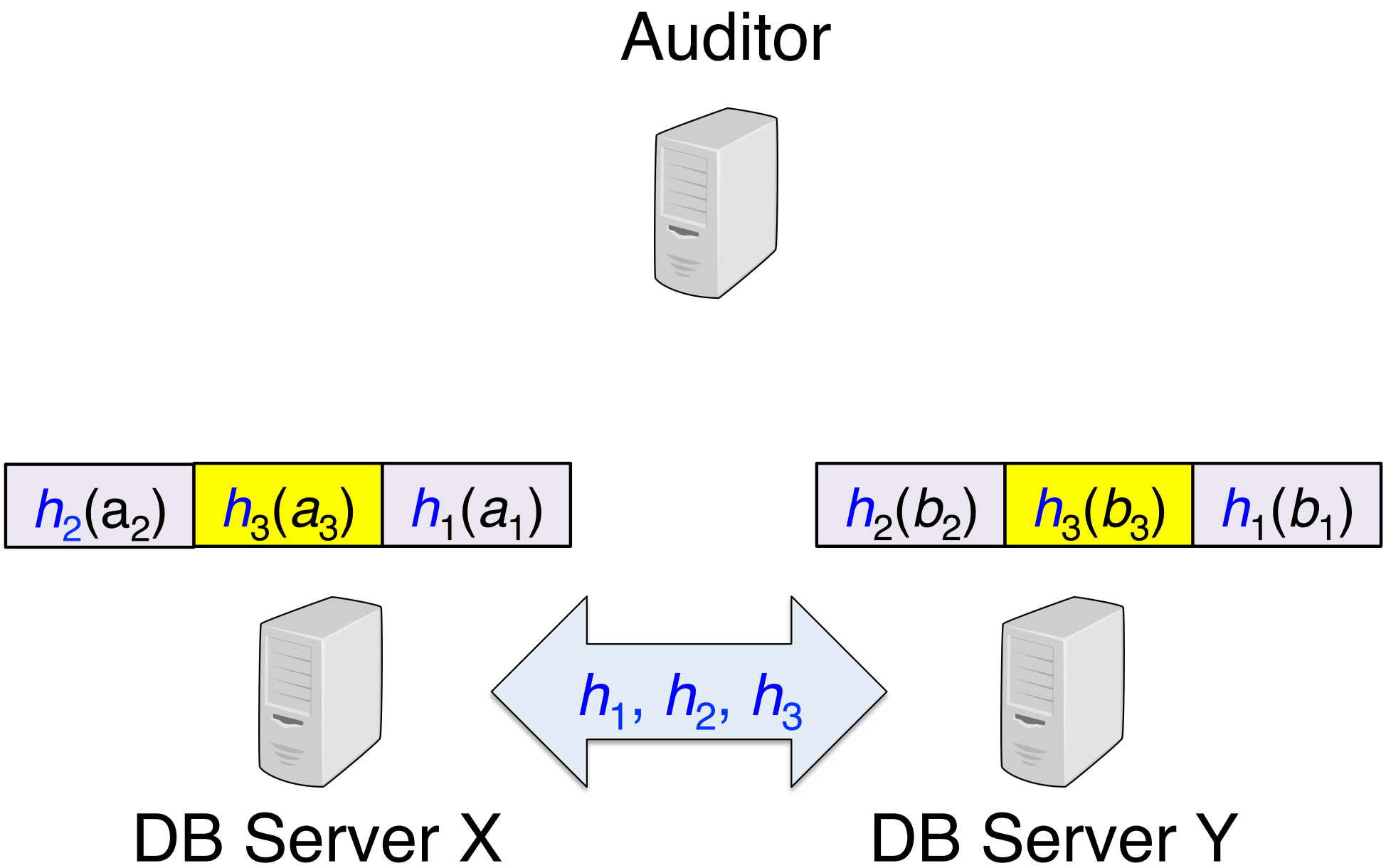
b_1	b_2	b_3

DB Server Y

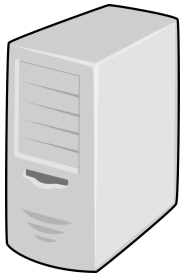

Auditor
a_1
a_2
a_3
b_1
b_2
b_3
offset
DB Server X
DB Server Y


Auditor
a_2
a_3
a_1
b_2
b_3
b_1
offset
DB Server X
DB Server Y


Auditor
a_2
a_3
a_1
b_2
b_3
b_1
h_1, h_2, h_3
DB Server X
DB Server Y


Auditor
$h_2(a_2)$
$h_3(a_3)$
$h_1(a_1)$
$h_2(b_2)$
$h_3(b_3)$
$h_1(b_1)$
h_1, h_2, h_3
DB Server X
DB Server Y

Auditor

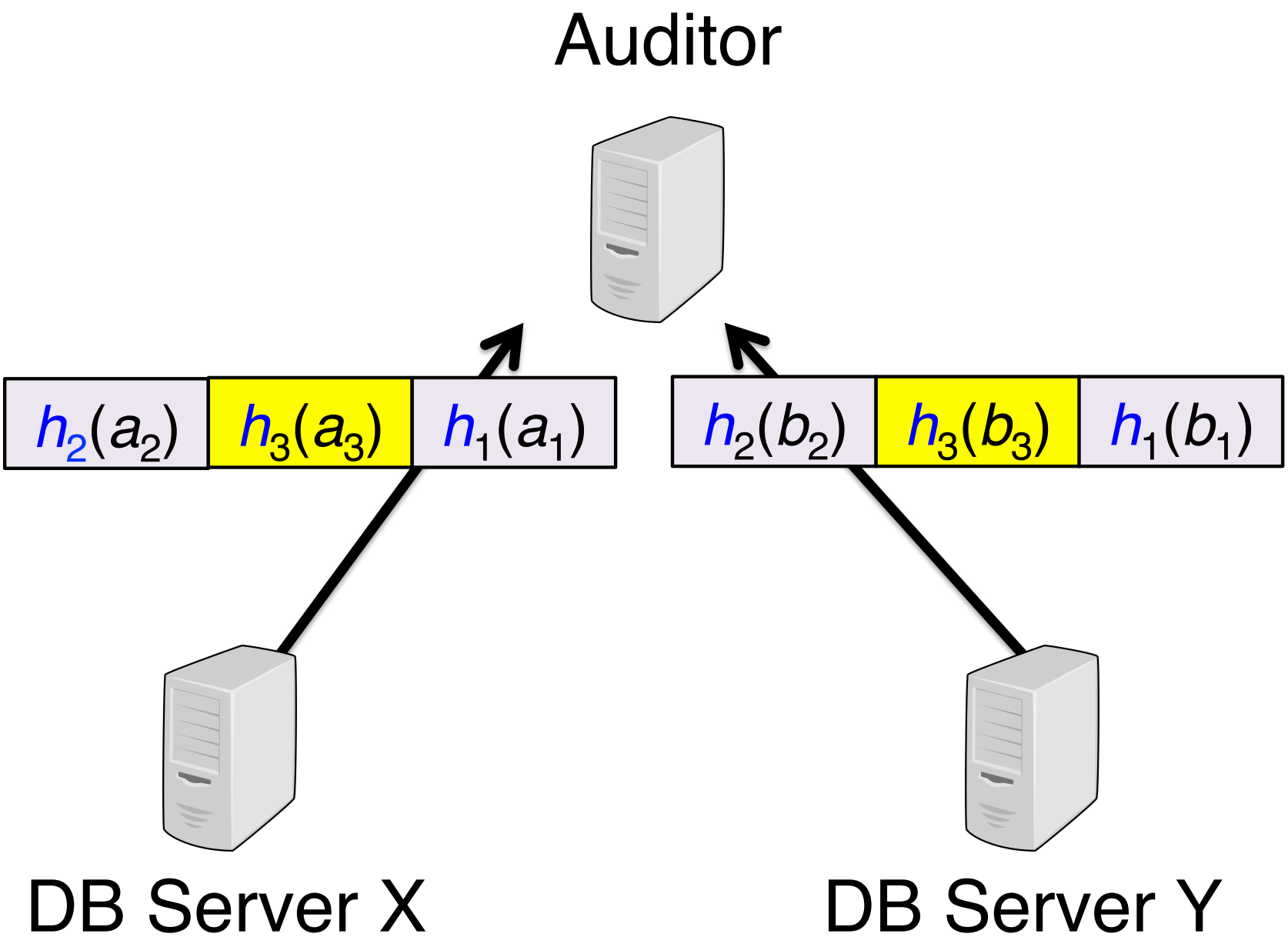


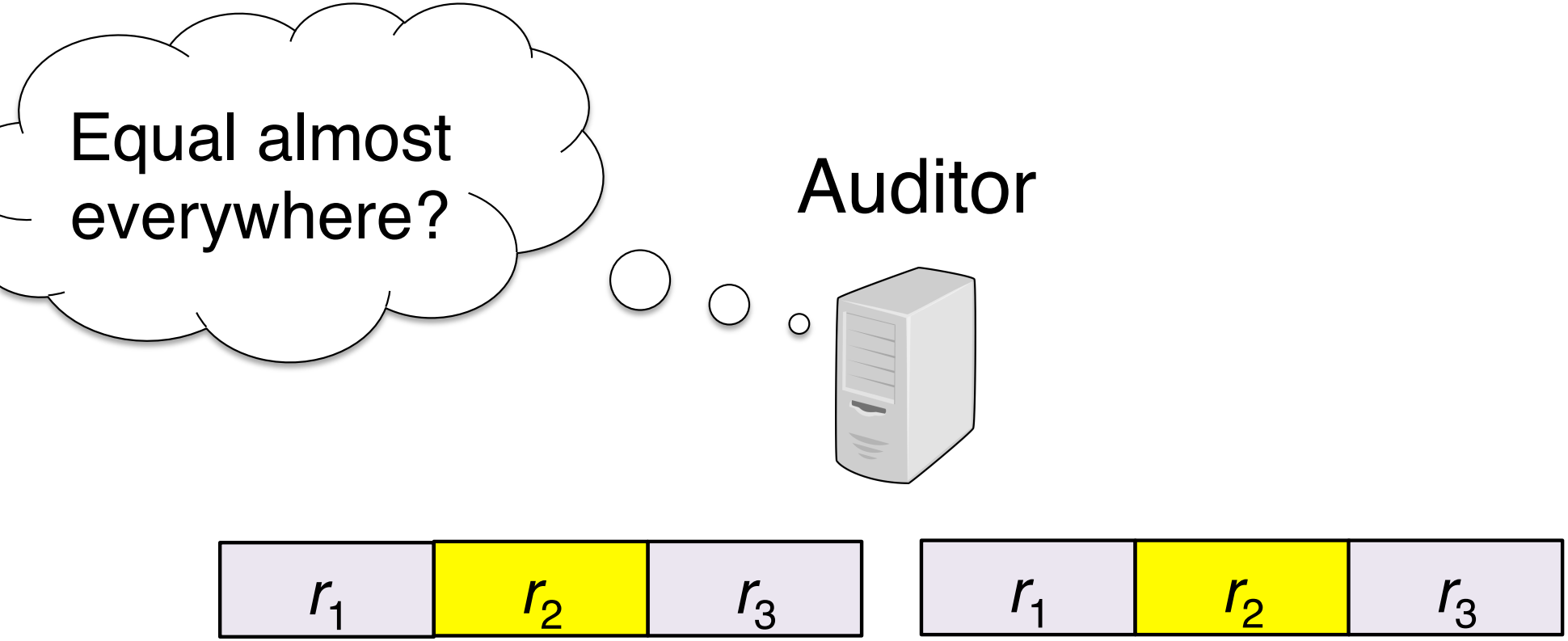
$h_2(a_2)$	$h_3(a_3)$	$h_1(a_1)$

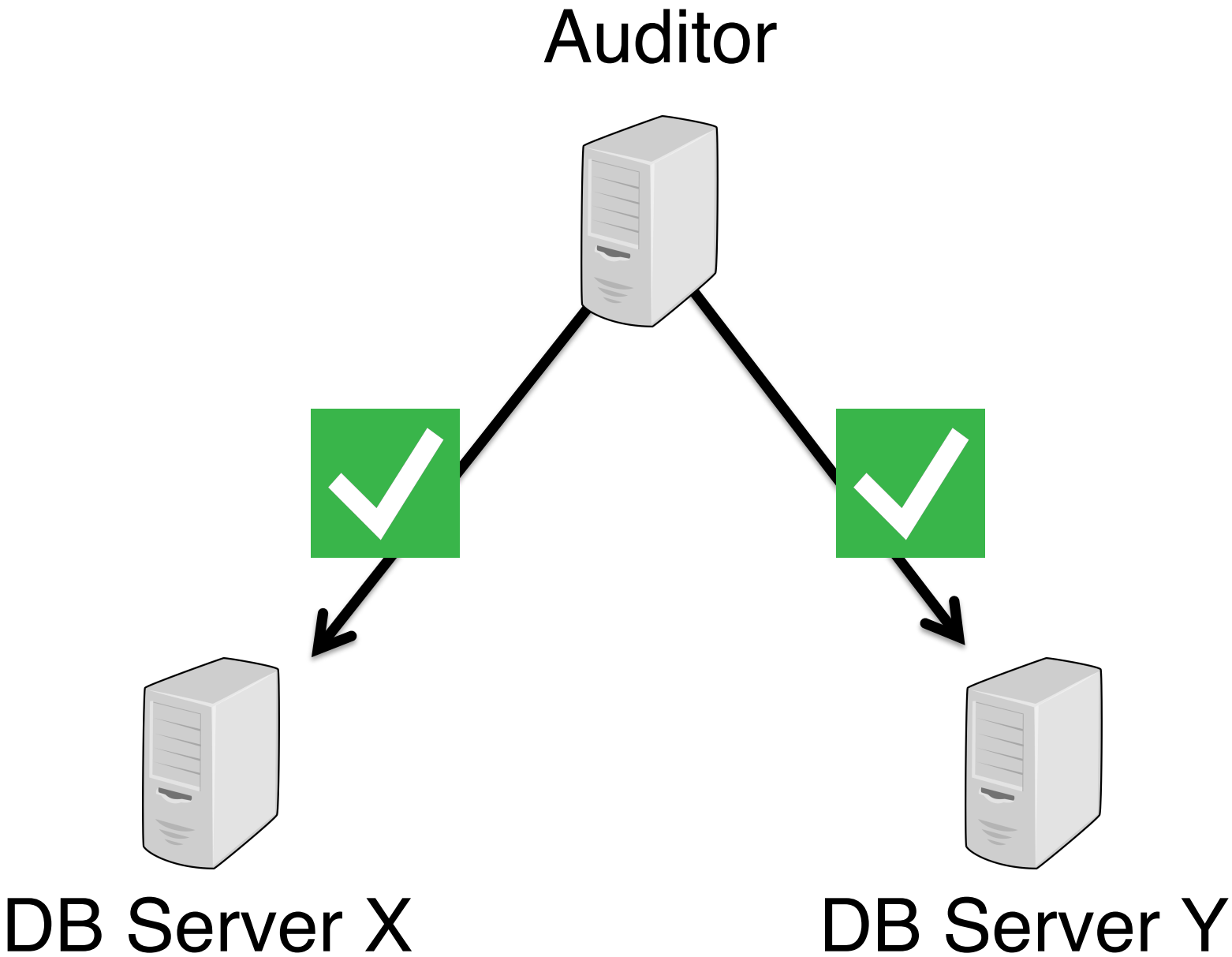
DB Server X

$h_2(b_2)$	$h_3(b_3)$	$h_1(b_1)$

DB Server Y


Auditor
$h_2(a_2)$
$h_3(a_3)$
$h_1(a_1)$
$h_2(b_2)$
$h_3(b_3)$
$h_1(b_1)$
DB Server X
DB Server Y


Equal almost everywhere?
Auditor
r_1
r_2
r_3
r_1
r_2
r_3


Auditor
DB Server X
DB Server Y

Outline

- Motivation
- Definitions and a “Straw man” scheme
- Technical challenges
- **Evaluation**
- Conclusions

Implementation

- Implemented the full protocol in Go
  - 2 DB servers + 1 audit server

- Ran perf evaluation on a network testbed simulating real-world net conditions

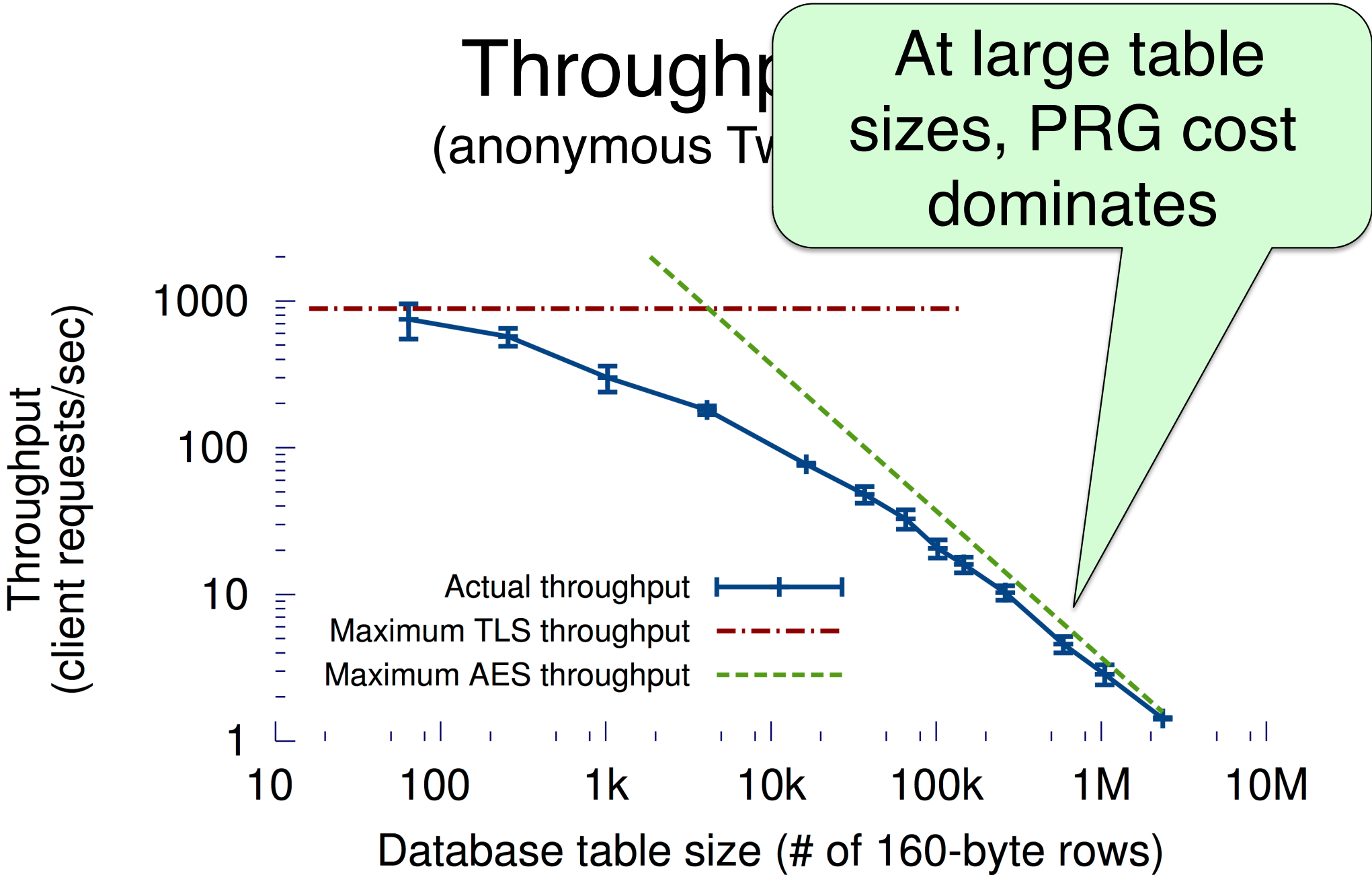
Bottom-Line Result

- For a DB with 65,000 Tweet-length rows, can process **30 writes/second**
- Can process **1,000,000 writes** in 8 hours on a single server

➔ Main bottleneck is PRG expansion

Throughp

(anonymous Tw



Outline

- Motivation
- Definitions and a “Straw man” scheme
- Technical challenges
- Evaluation
- **Conclusions**

Open Problems

1. Reduce $\Theta(L)$ computation cost at server
   - Using multiple rounds per write?
2. Key-homomorphic DPFs
   - Another way to reduce cost at server
3. $(k, k\text{-}1)$-private DPFs without PKC
   - Possible without seed-hom PRGs?

Conclusion

In many contexts, “hiding the metadata” is as important as hiding the data

Combination of crypto tools with systems design ➔ 1,000,000-user anonymity sets

“Multi-user writable PIRs” have applications to private messaging

– Still ∃ barriers to practicality (+ open problems)

Questions?