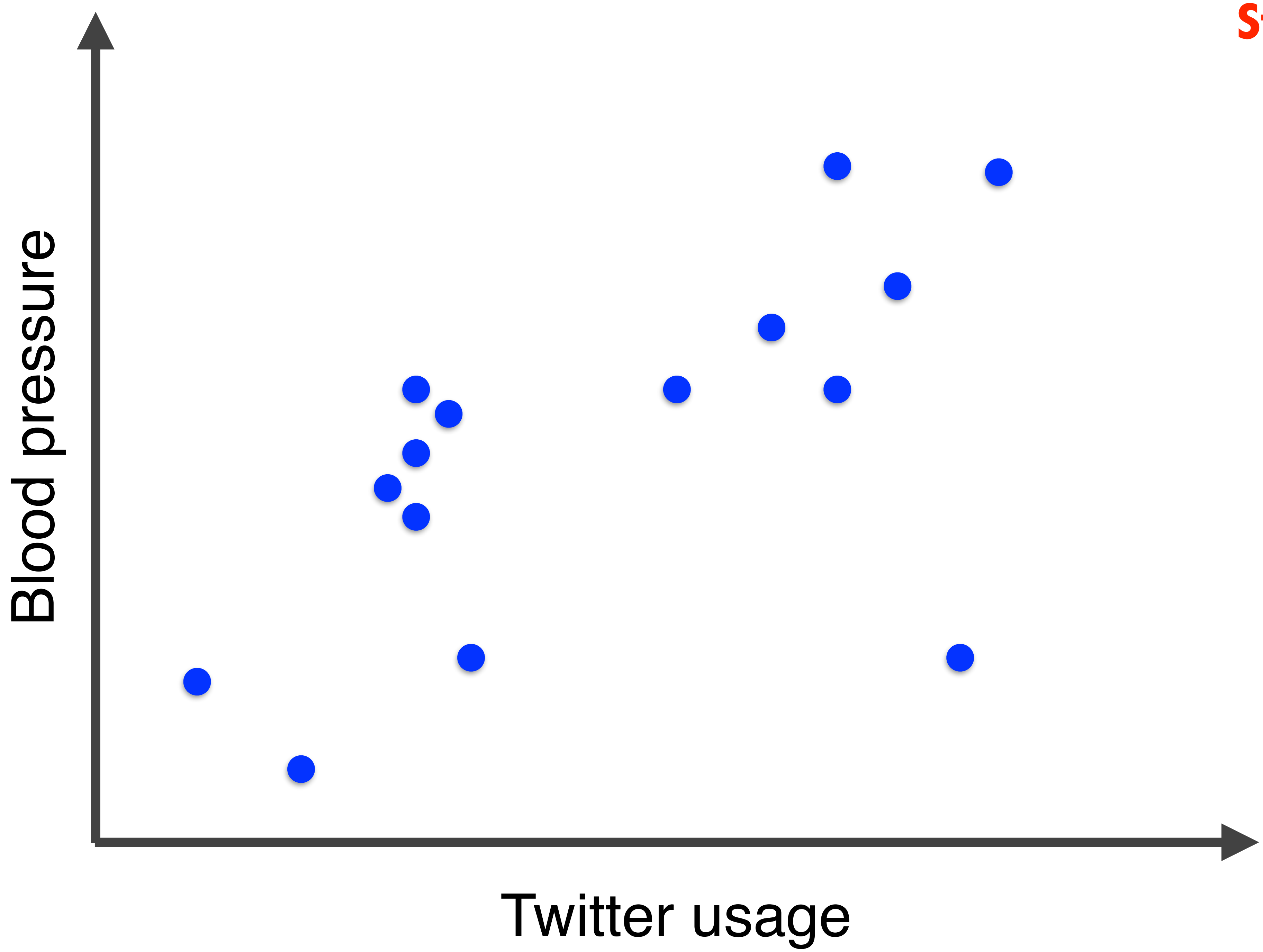
Prio: Private, Robust, and Efficient Computation of Aggregate Statistics

Henry Corrigan-Gibbs and Dan Boneh
Stanford University

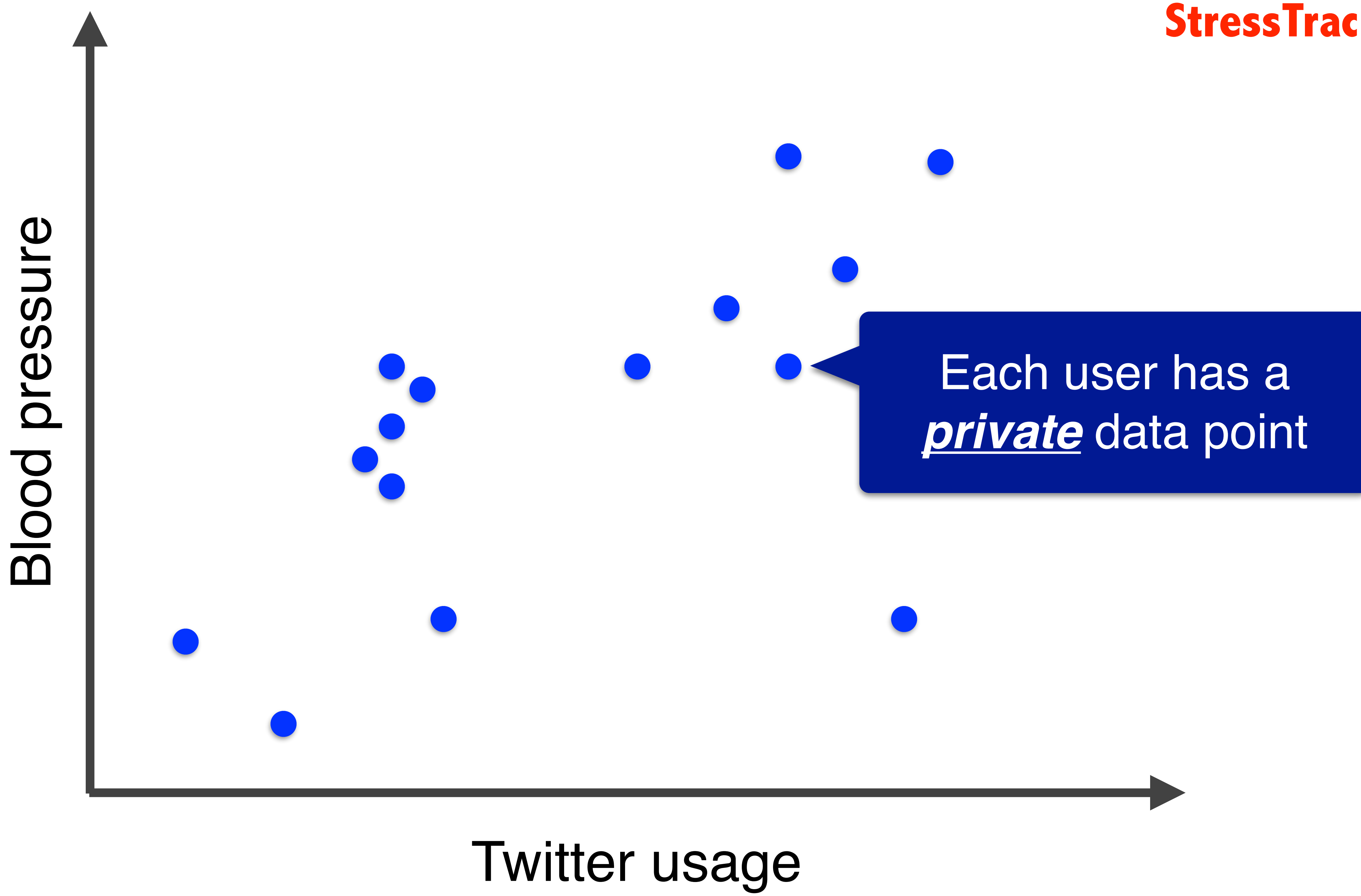
NSDI 2017

Today: Non-private aggregation

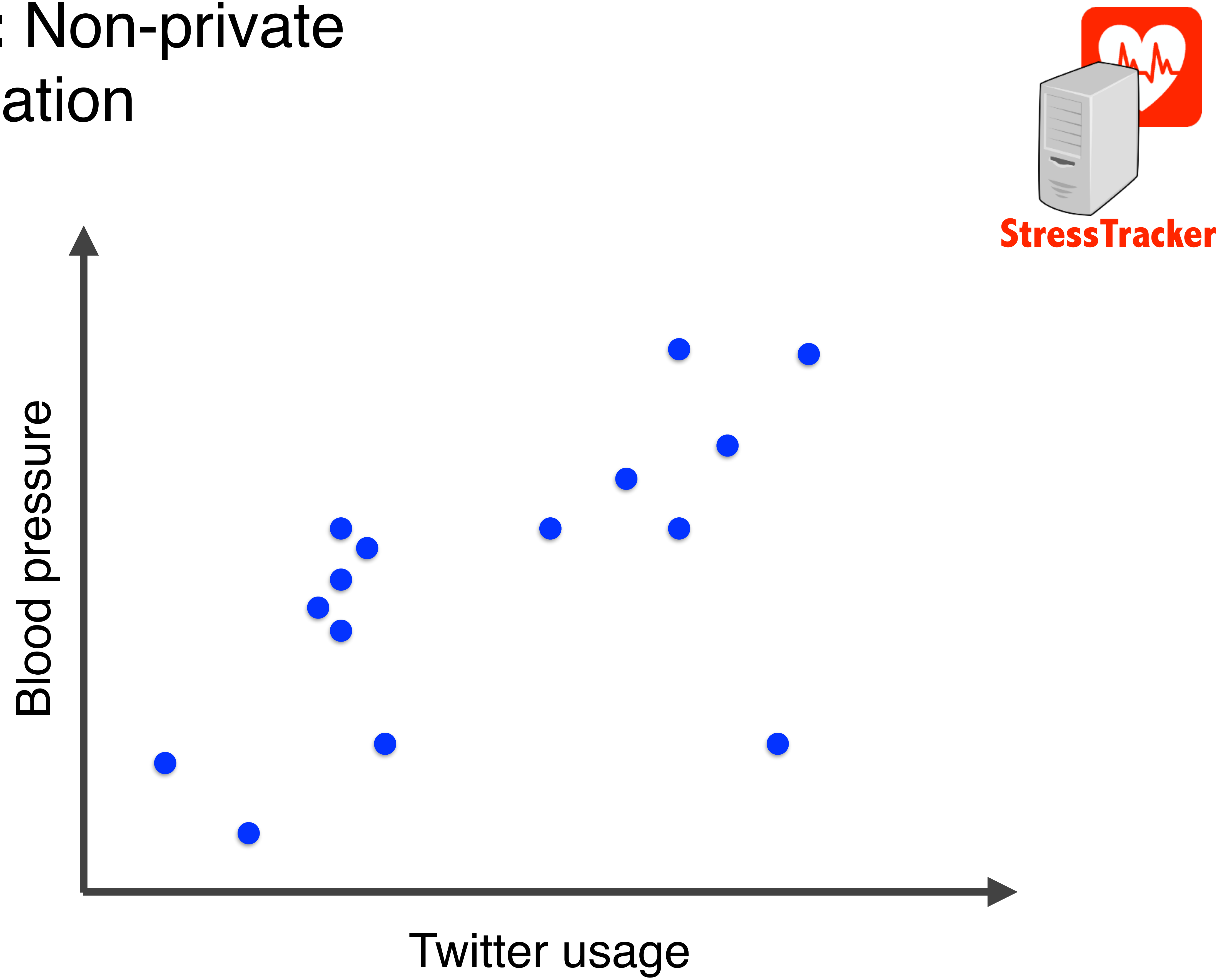


Today: Non-private aggregation

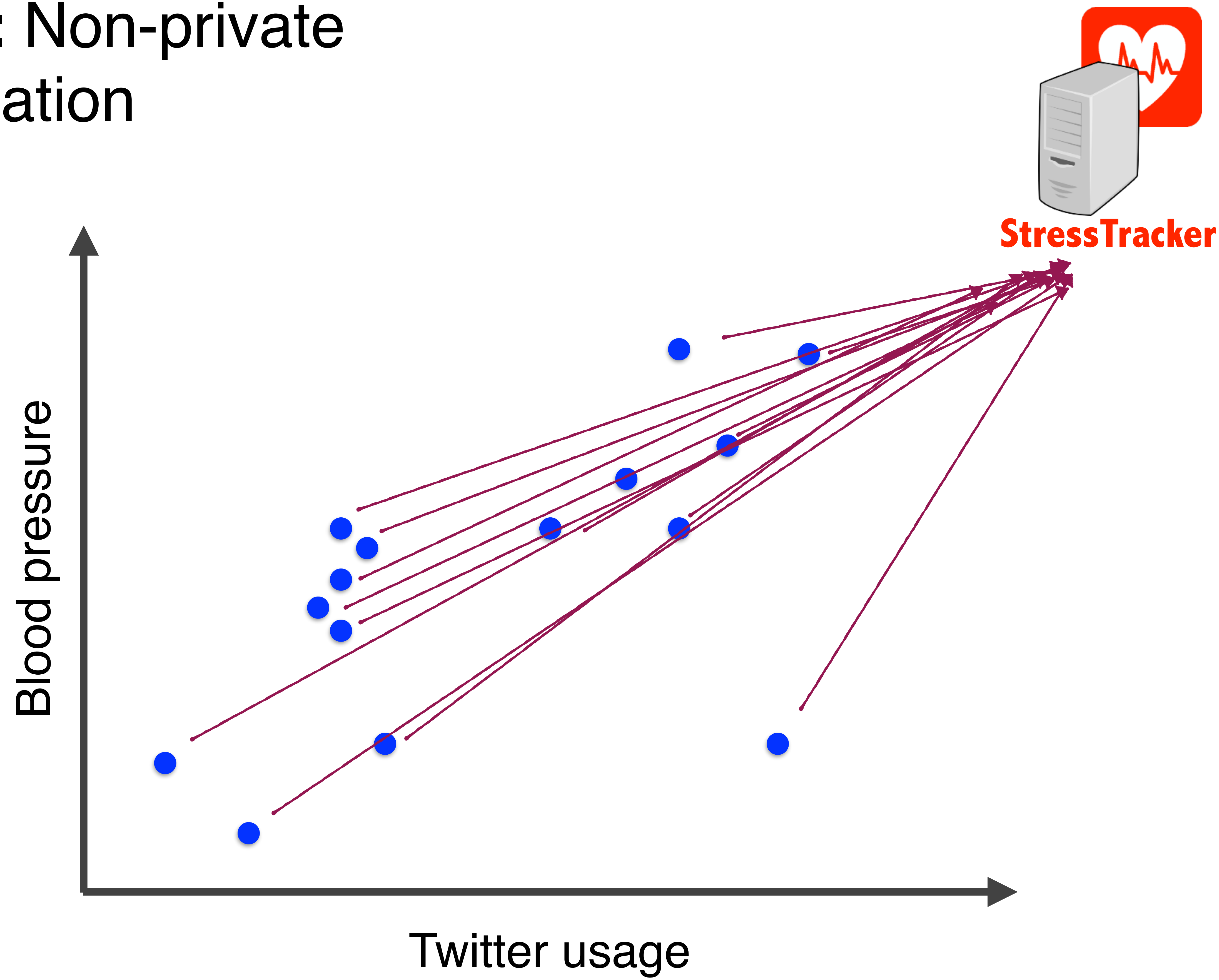
StressTracker



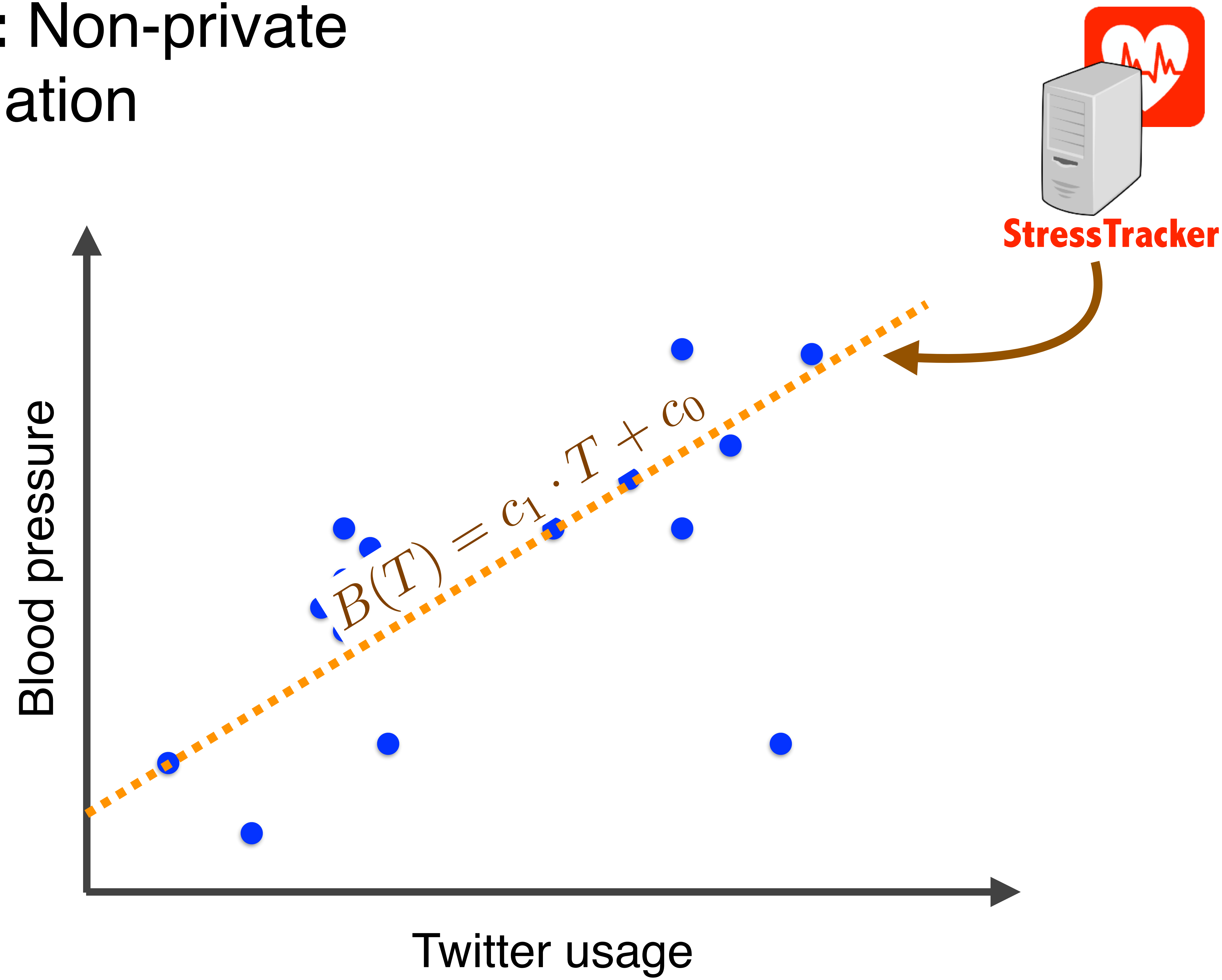
Today: Non-private aggregation



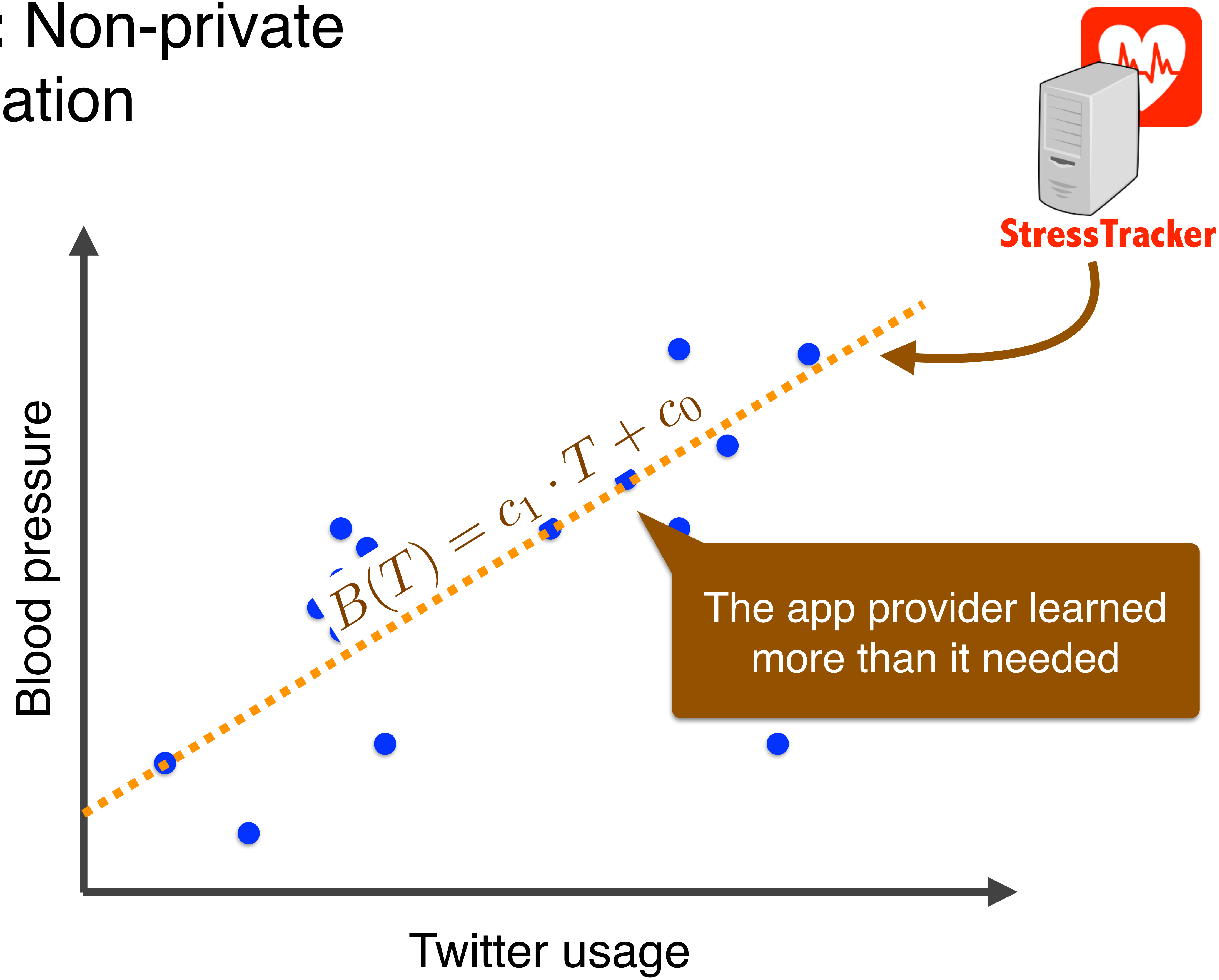
Today: Non-private aggregation



Today: Non-private aggregation

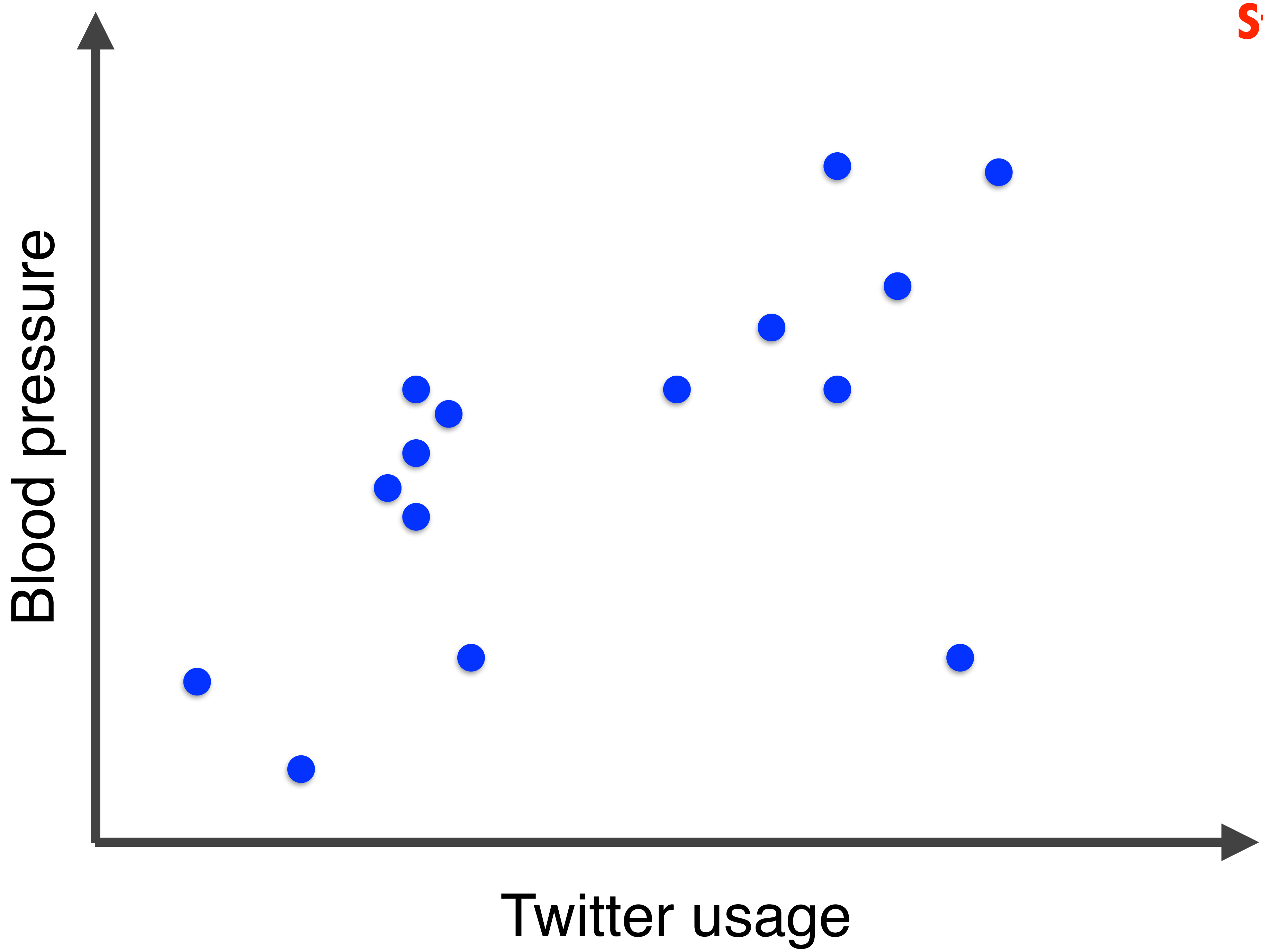


Today: Non-private aggregation

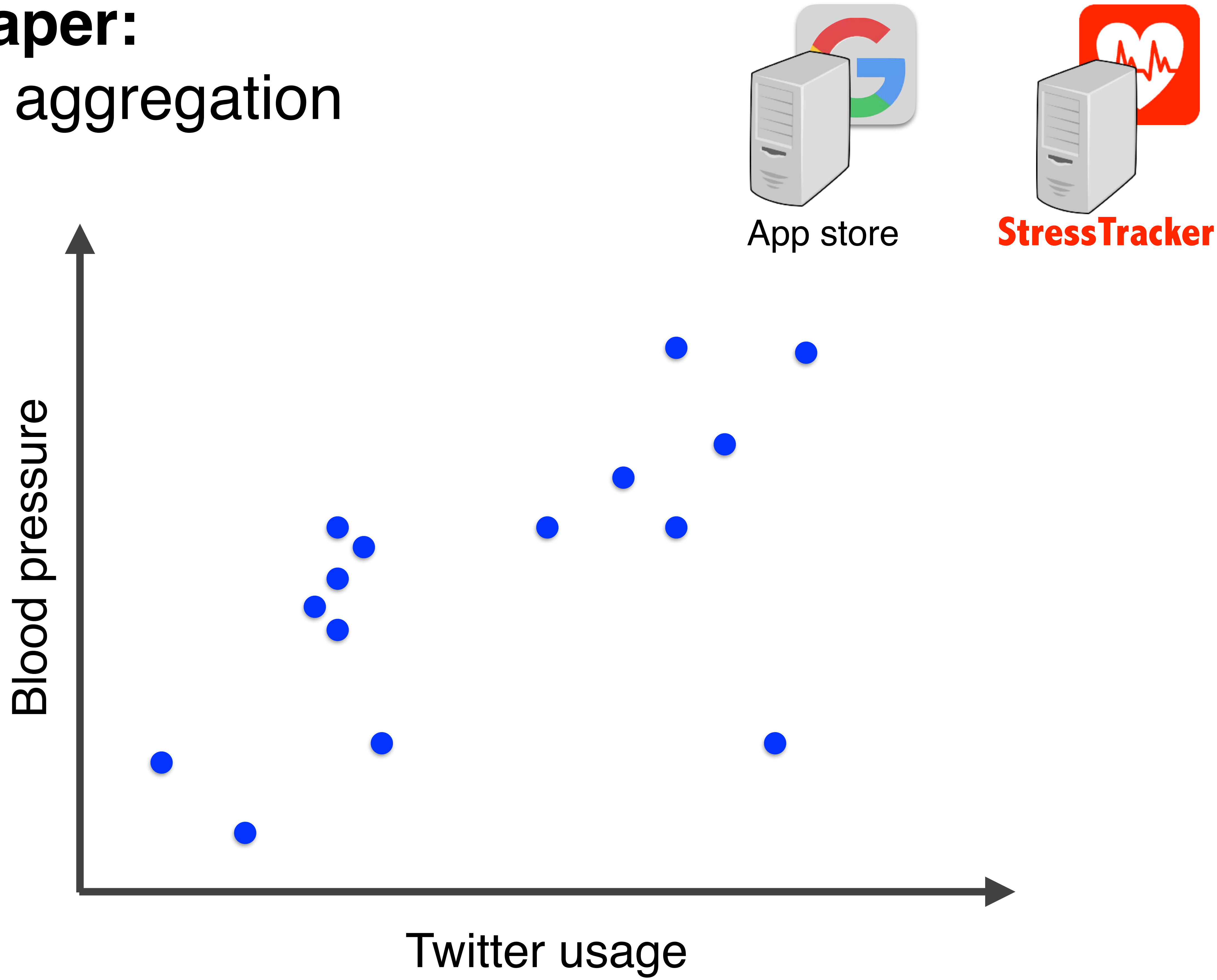


Today: Non-private aggregation

StressTracker

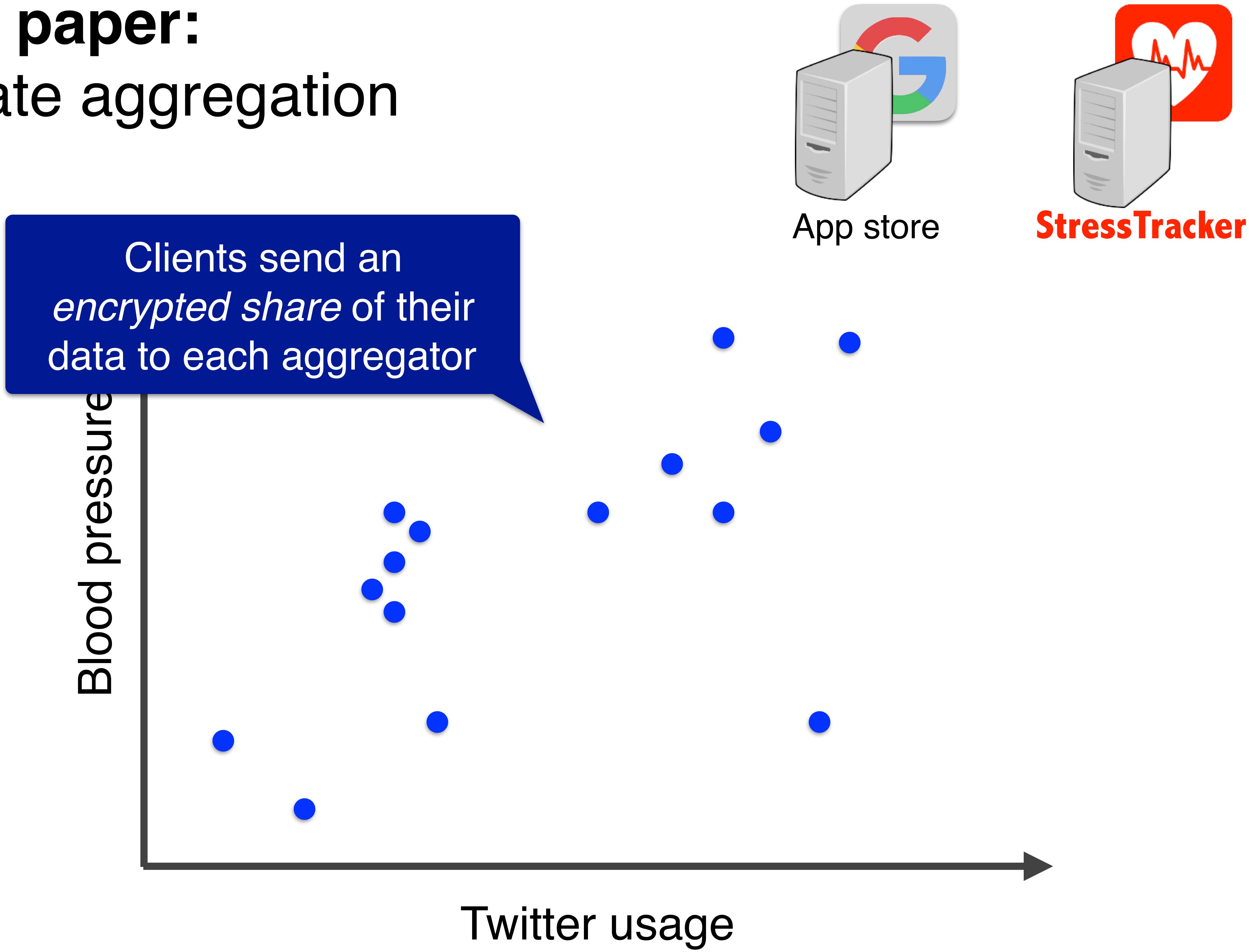


This paper:
Private aggregation

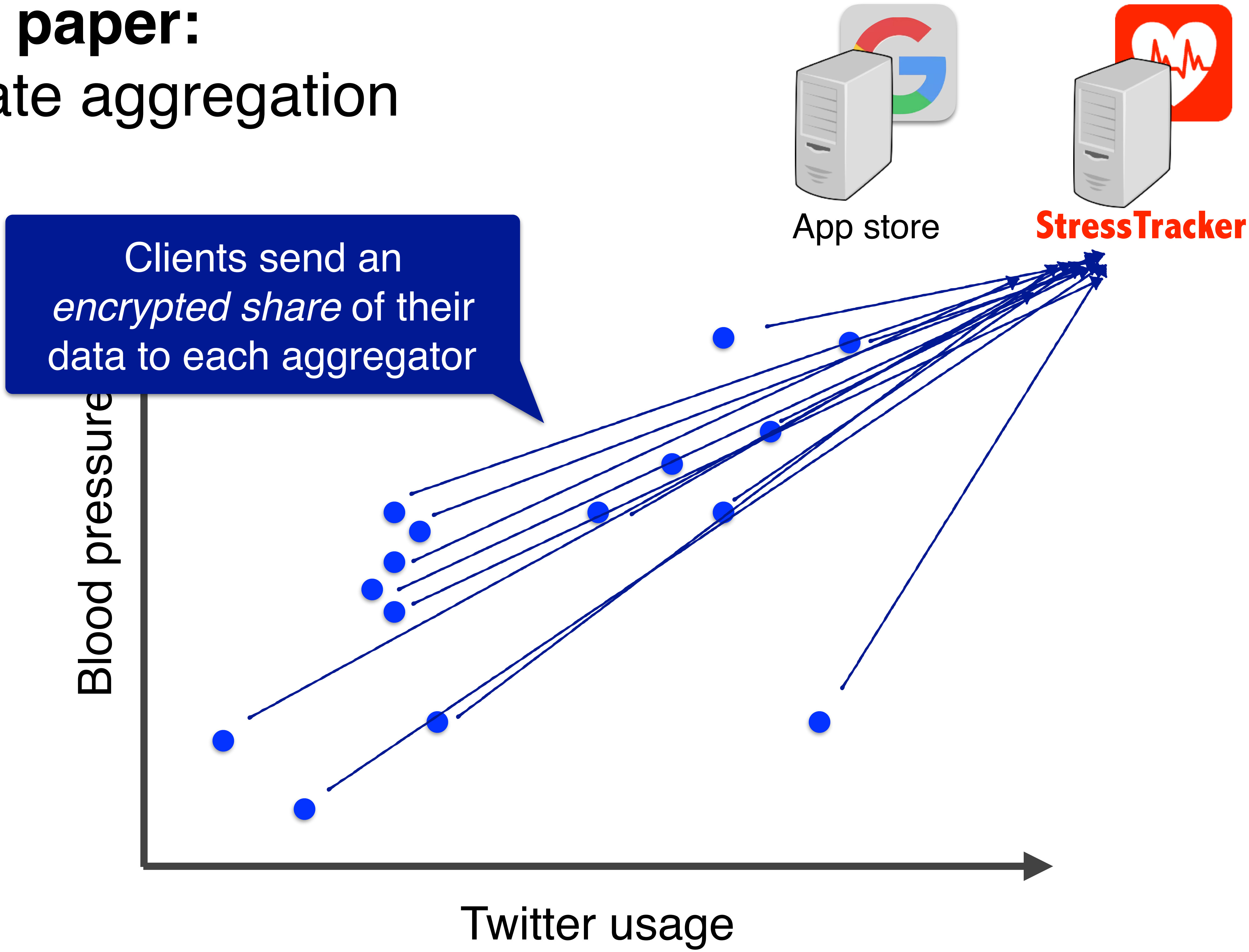


This paper:

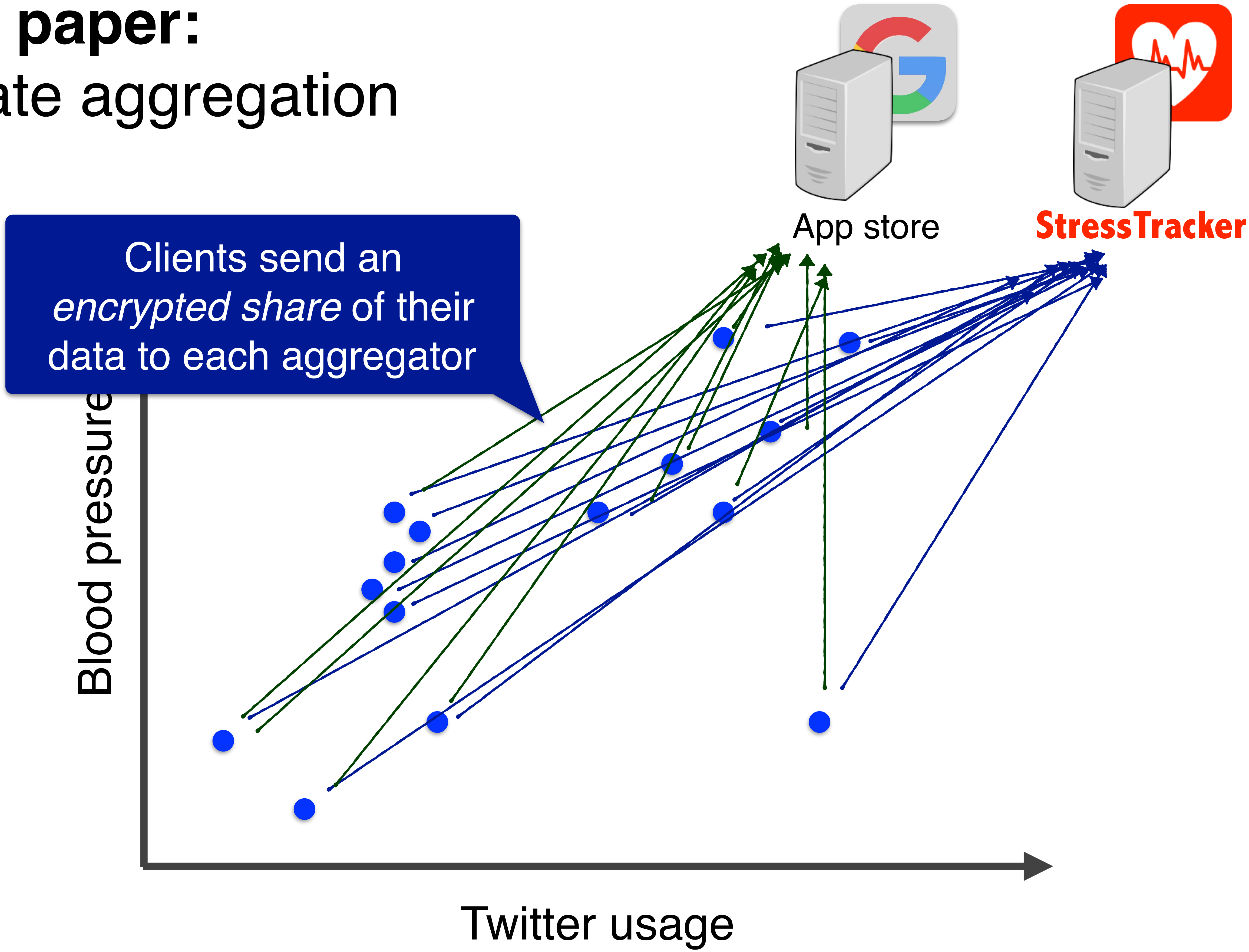
Private aggregation



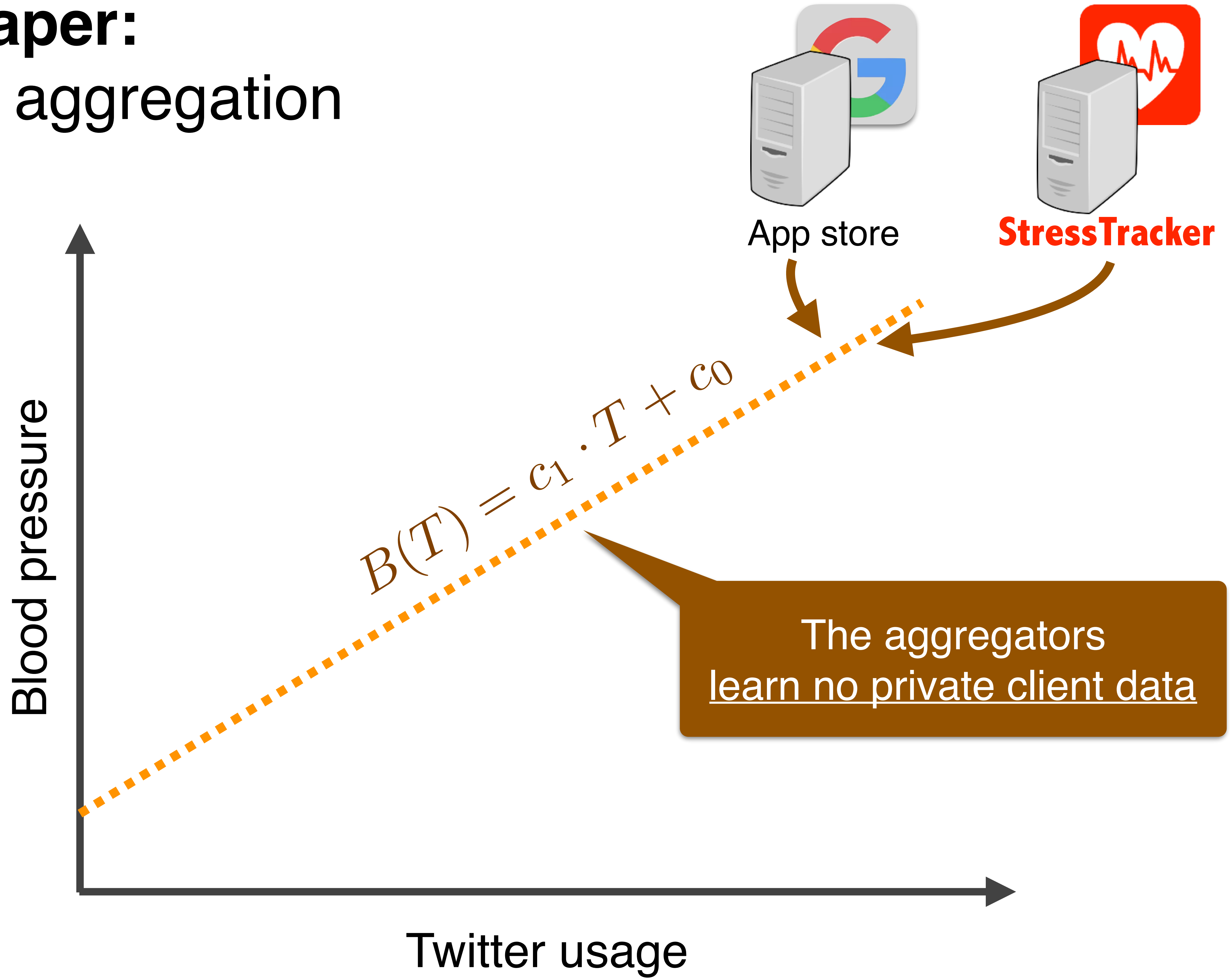
This paper:
Private aggregation



This paper:
Private aggregation

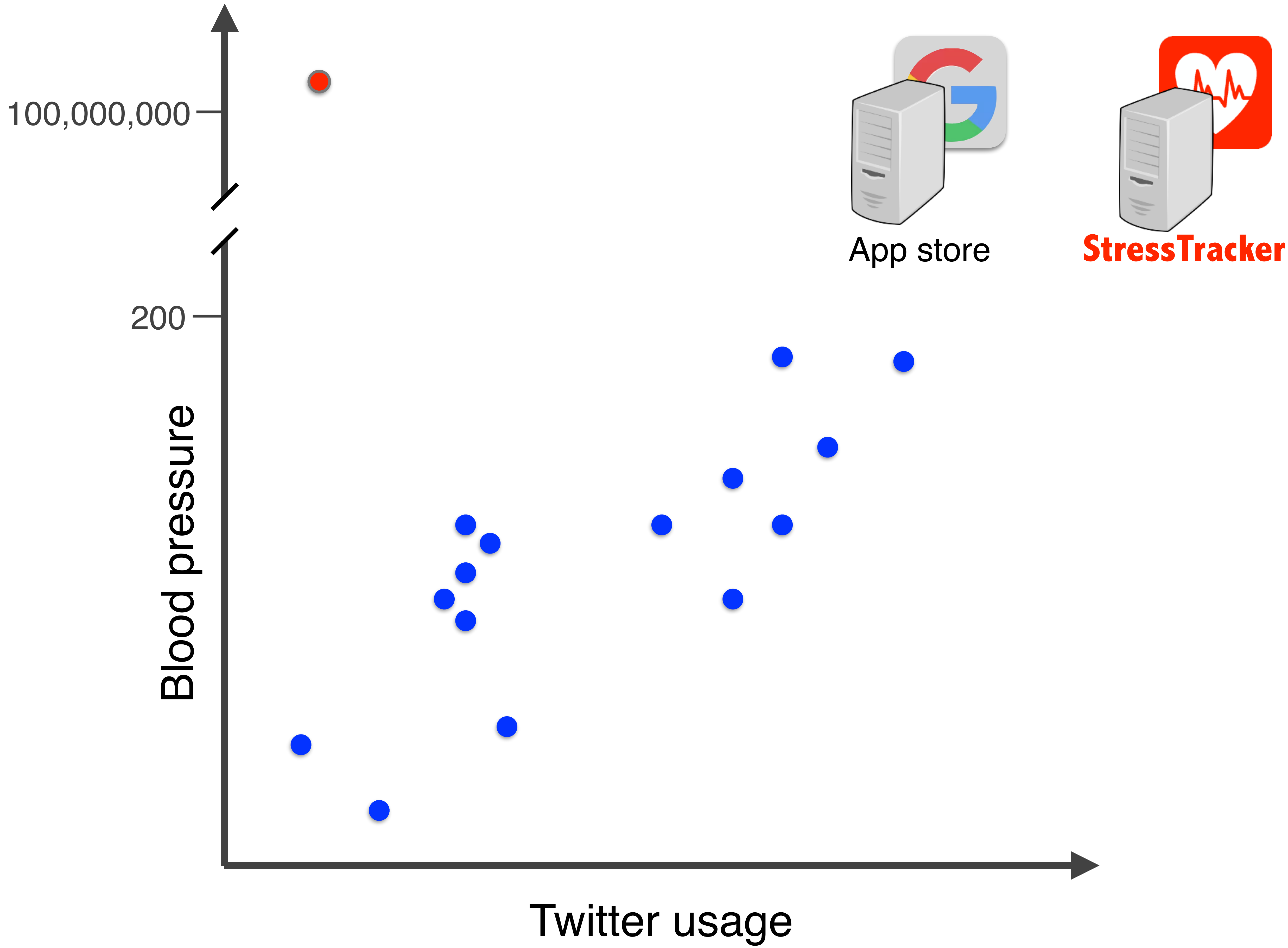


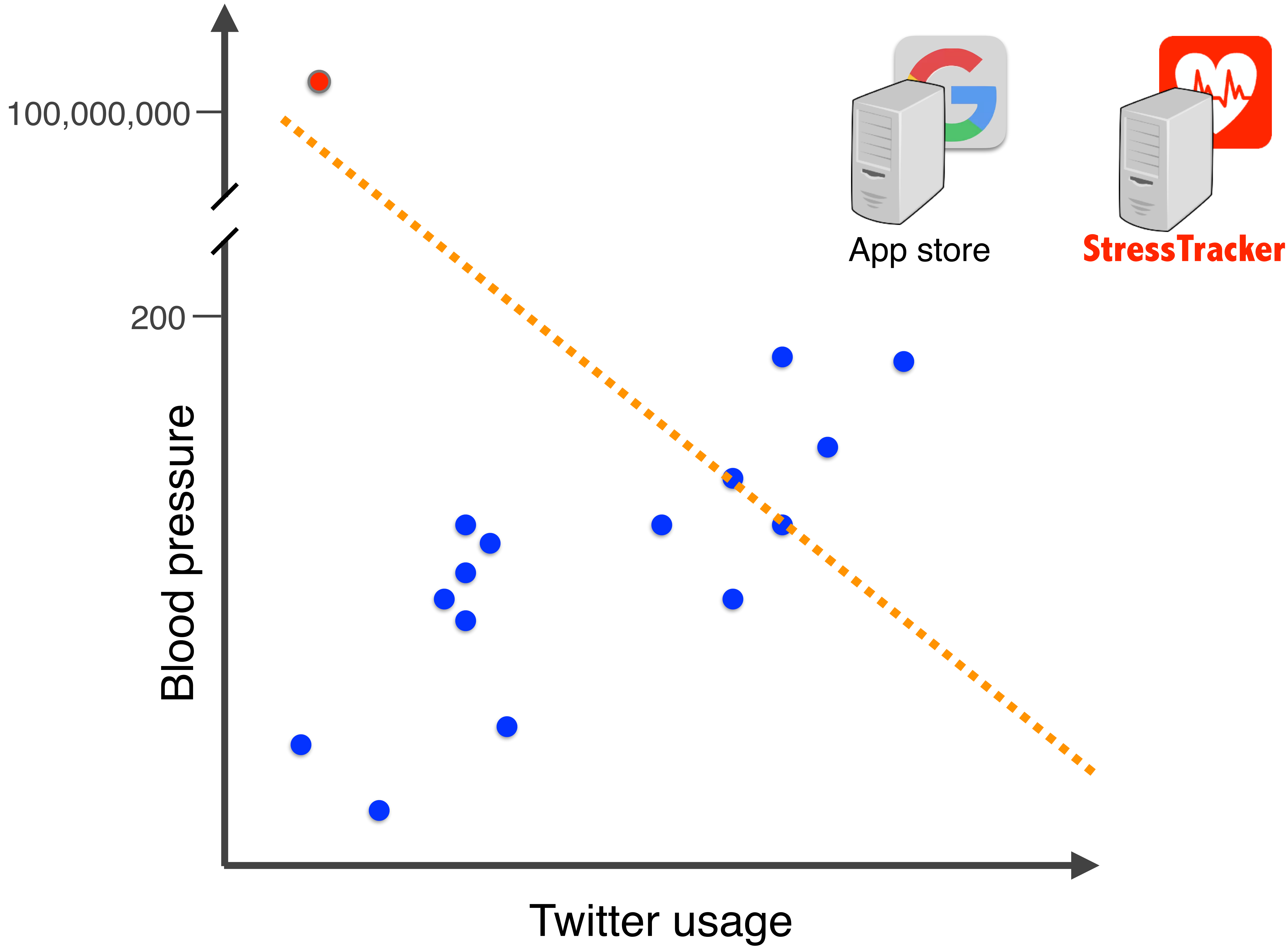
This paper:
Private aggregation



Private aggregation

1. Exact correctness	If all servers are honest, servers learn $f(\cdot)$
2. Privacy	If one server is honest, servers learn only* $f(\cdot)$
3. Robustness	Malicious clients have bounded influence
4. Efficiency	No public-key crypto (apart from TLS) 1000s of submissions per second


100,000,000
200
Blood pressure
Twitter usage
App store
StressTracker


100,000,000
200
Blood pressure
Twitter usage
App store
StressTracker

Private aggregation

1. **Exact correctness** — If <u>all servers</u> are honest, servers learn $f(\cdot)$
2. **Privacy** — If <u>one server</u> is honest, servers learn only* $f(\cdot)$
3. **Robustness** — Malicious clients have bounded influence
4. **Efficiency** — No public-key crypto (apart from TLS)
1000s of submissions per second

Prio is the first system to achieve all four.

Private aggregation

1. **Exact correctness** If <u>all servers</u> are honest, servers learn f(·)
2. **Privacy** If <u>one server</u> is honest, servers learn only* f(·)
3. **Robustness** Malicious clients hav
4. **Efficiency** No public-key crypto
1000s of submission

…and Prio supports a wide range of aggregation functions f(·)

Prio is the first system to achieve all four.

Private aggregation

1. **Exact correctness** — If <u>all servers</u> are honest, servers learn f(·)
2. **Privacy** — If <u>one server</u> is honest, servers learn only* f(·)
3. **Robustness** — Malicious clients have bounded influence
4. **Efficiency** — No public-key crypto (apart from TLS)
1000s of submissions per second

Prio is the first system to achieve all four.

Contributions

1. **Secret-shared non-interactive proofs (SNIPs)**
   – Client proves that its encoded submission is well-formed
   – We do not need the power of traditional "heavy" crypto tools

2. **Aggregatable encodings**
   Can compute sums privately $\Longrightarrow$ Can compute $f(\cdot)$ privately
   …for many f's of interest

Related systems

- Additively homomorphic encryption
  P4P (2010), Private stream aggregation (2011), Grid aggregation (2011),
  PDDP (2012), SplitX (2013), PrivEx (2014), PrivCount (2016),
  Succinct sketches (2016), …
- Multi-party computation [GMW87], [BGW88]
  FairPlay (2004), Brickell-Shmatikov (2006), FairplayMP (2008), SEPIA (2010),
  Private matrix factorization (2013), JustGarble (2013), …
- Anonymous credentials/tokens
  VPriv (2009), PrivStats (2011), ANONIZE (2014), …
- Randomized response [W65], [DMNS06], [D06]
  RAPPOR (2014, 2016)

Prio is the first system to achieve
exact correctness, privacy, robustness, efficiency.

Outline

- **Background: The private aggregation problem**
- A straw-man solution for private sums
- Providing robustness with SNIPs
- Evaluation
- Encodings for <u>complex</u> aggregates

Outline

- Background: The private aggregation problem
- **A straw-man solution for private sums**
- Providing robustness with SNIPs
- Evaluation
- Encodings for <u>complex</u> aggregates

Warm-up: Computing private sums

Warm-up: Computing private sums

- Every device i holds a value x_i
- We want to compute

$$f(x_1, \ldots, x_N) = x_1 + \ldots + x_N$$

  without learning any users' private value x_i.

Warm-up: Computing private sums

- Every device i holds a value x_i
- We want to compute

$$f(x_1, \ldots, x_N) = x_1 + \ldots + x_N$$

without learning any users' private value x_i.

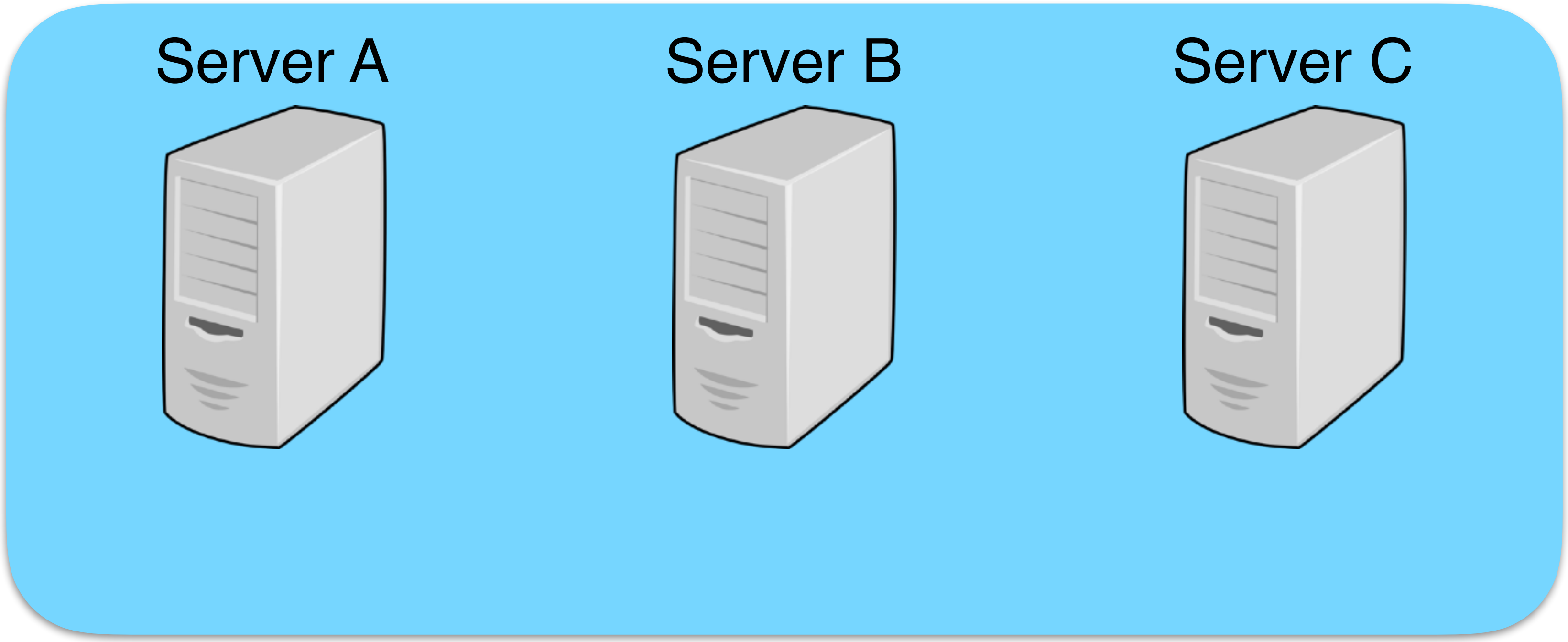
Example: Privately measuring traffic congestion.

$$x_i \quad = 1 \qquad \text{if user } i \text{ is on the Bay Bridge}$$
$$\quad = 0 \qquad \text{otherwise}$$

The sum $x_1 + \ldots + x_N$ yields the number of app users on the Bay Bridge.

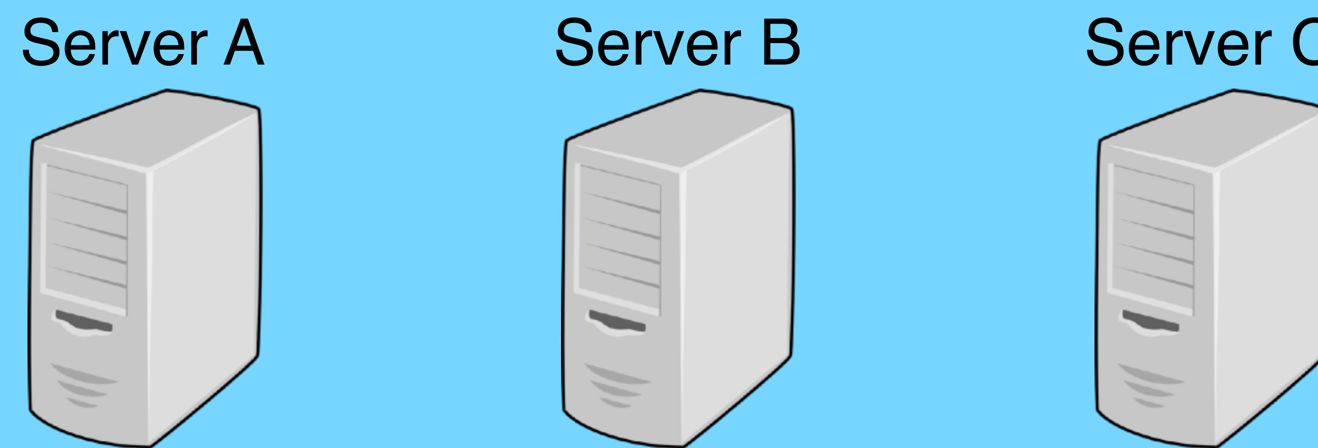
Private sums: A “straw-man” scheme

[Chaum88], [BGW88], …
[KDK11] [DFKZ13] [PrivEx14] …



Private sums: A “straw-man” scheme

[Chaum88], [BGW88], …
[KDK11] [DFKZ13] [PrivEx14] …

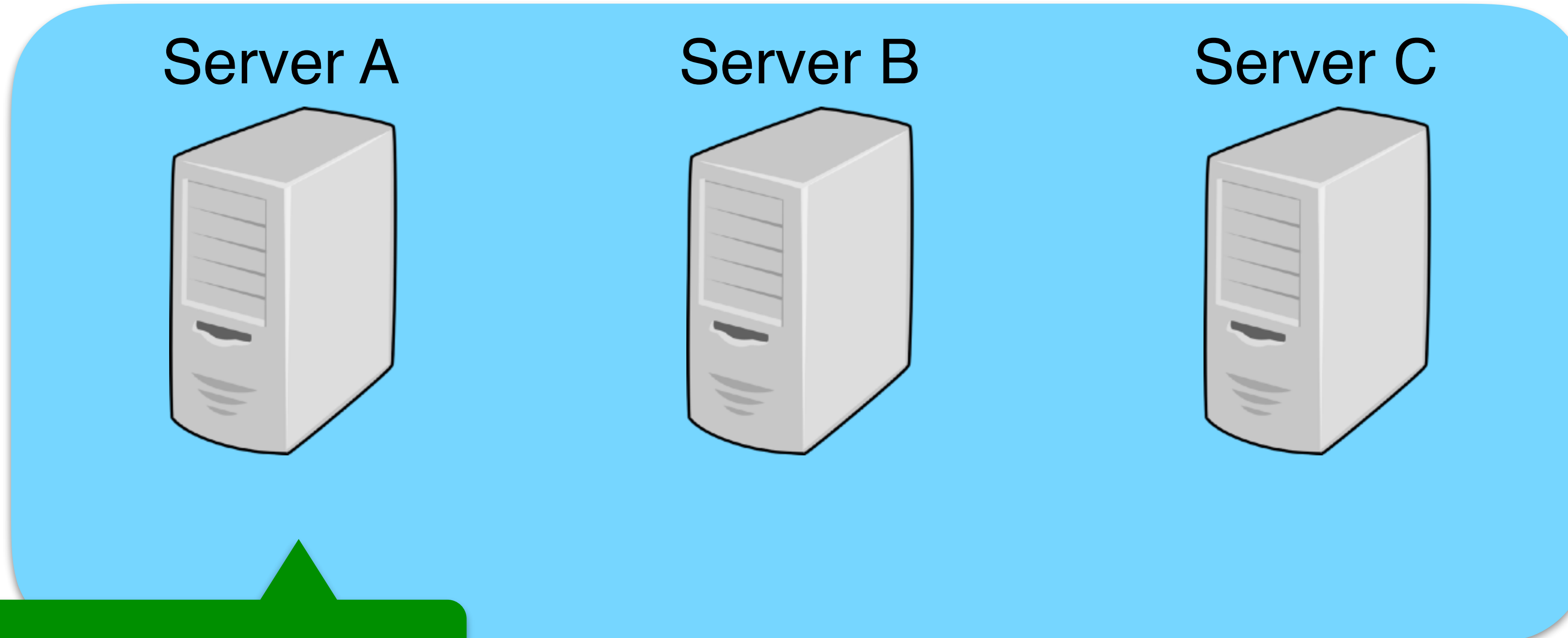


Assume that the servers are non-colluding.

Equivalently: that at least one server is honest.

Private sums: A “straw-man” scheme

[Chaum88], [BGW88], …
[KDK11] [DFKZ13] [PrivEx14] …



Assume that the servers are non-colluding.

Equivalently: that at least one server is honest.

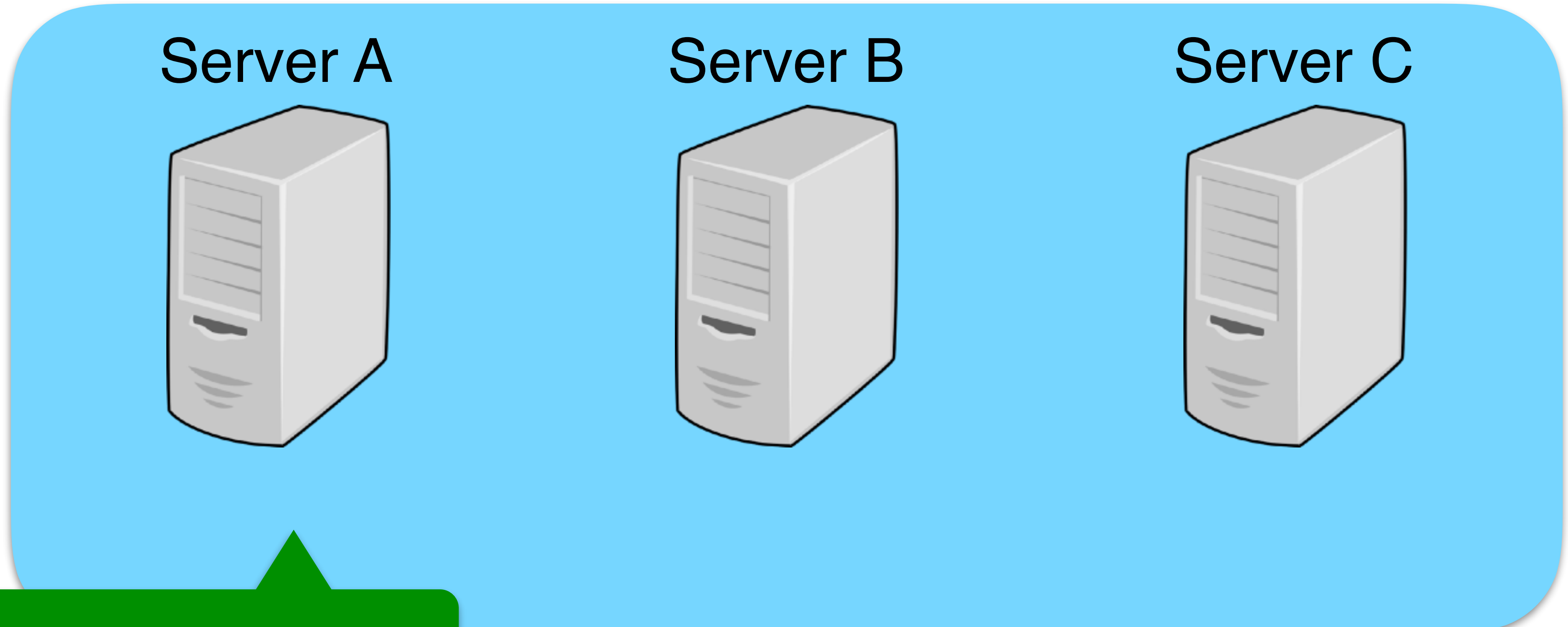
App store

App

StressTracker

Private sums: A “straw-man” scheme

[Chaum88], [BGW88], …
[KDK11] [DFKZ13] [PrivEx14] …

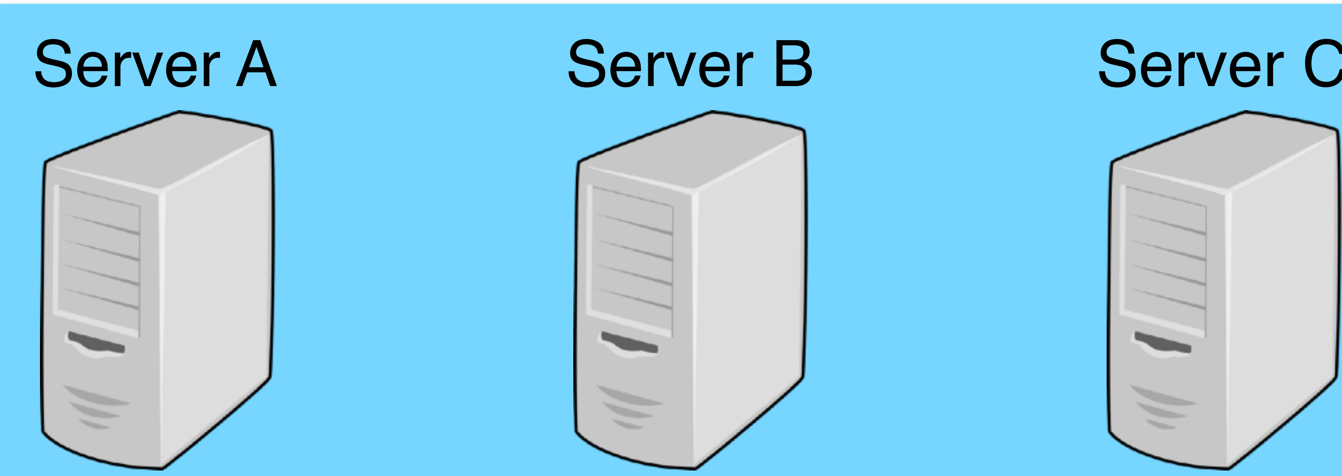


Assume that the servers are non-colluding.

Equivalently: that at least one server is honest.

Private sums: A "straw-man" scheme

[Chaum88], [BGW88], …
[KDK11] [DFKZ13] [PrivEx14] …

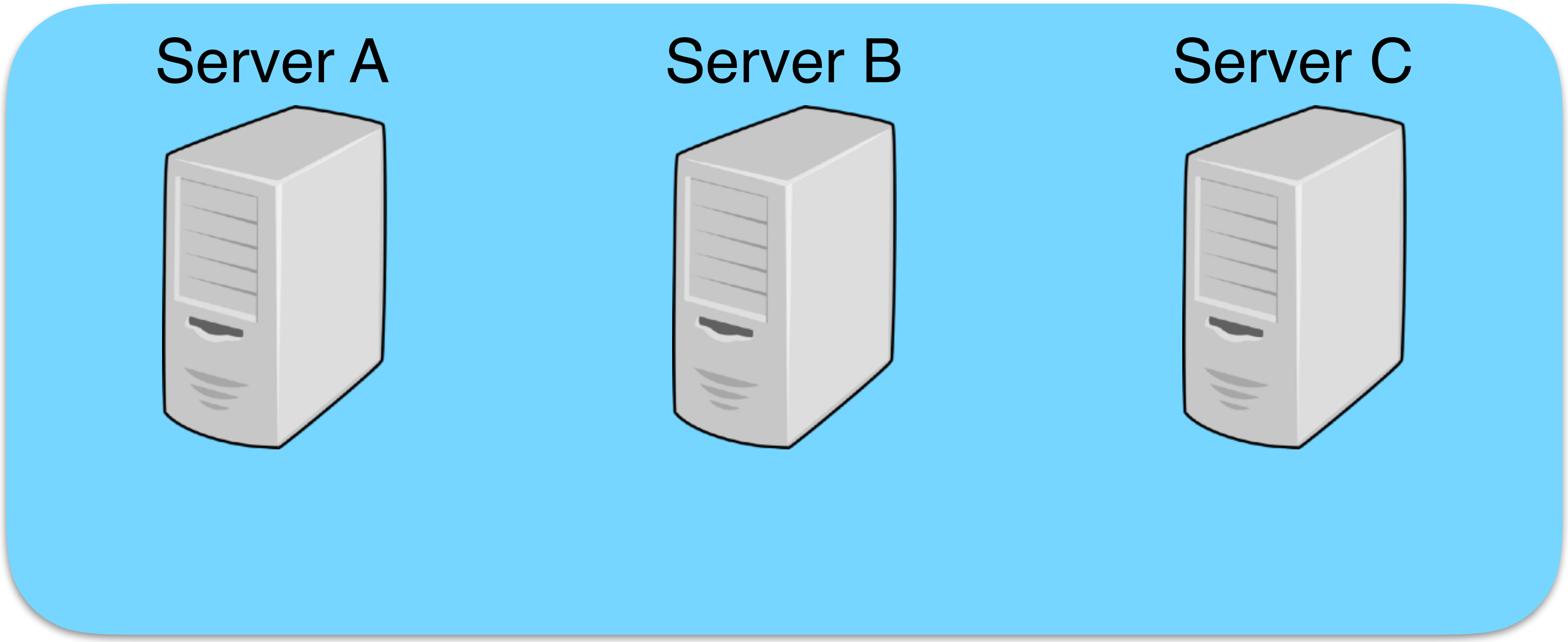


Assume that the servers are non-colluding.

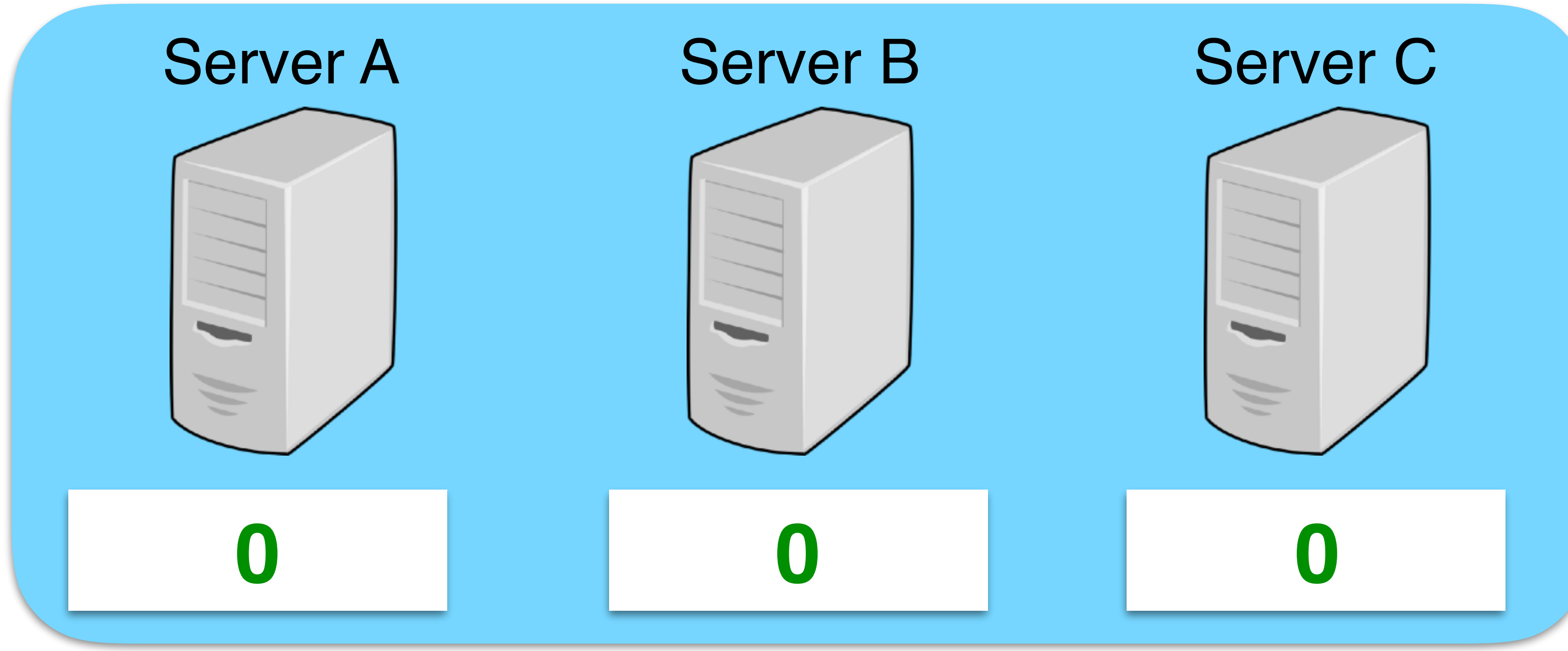
Equivalently: that at least one server is honest.

Private sums: A “straw-man” scheme

[Chaum88], [BGW88], …
[KDK11] [DFKZ13] [PrivEx14] …



Private sums: A "straw-man" scheme



1

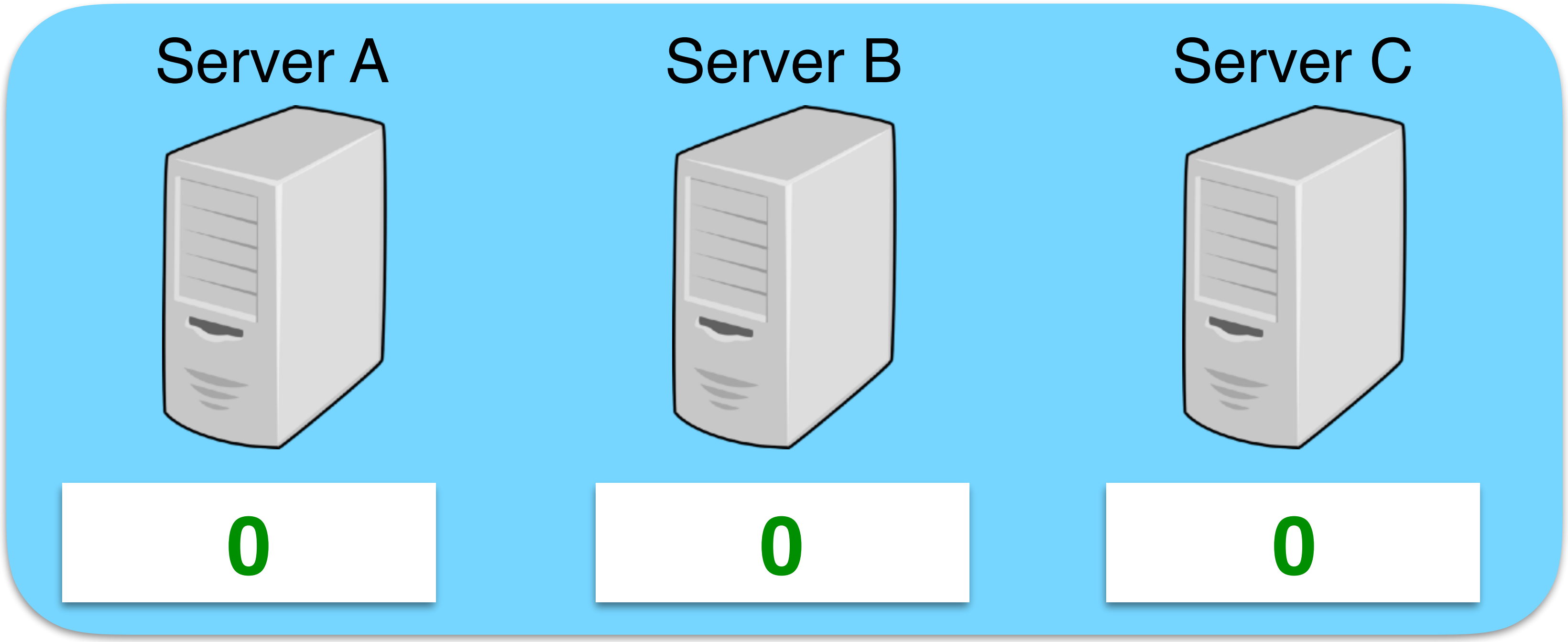
<u>Secret sharing</u>

Pick three random "shares" that sum to 1.

$1 = 15 + (-12) + (-2) \pmod{31}$

Need all three shares to recover the shared value.

Private sums: A “straw-man” scheme



1

Secret sharing

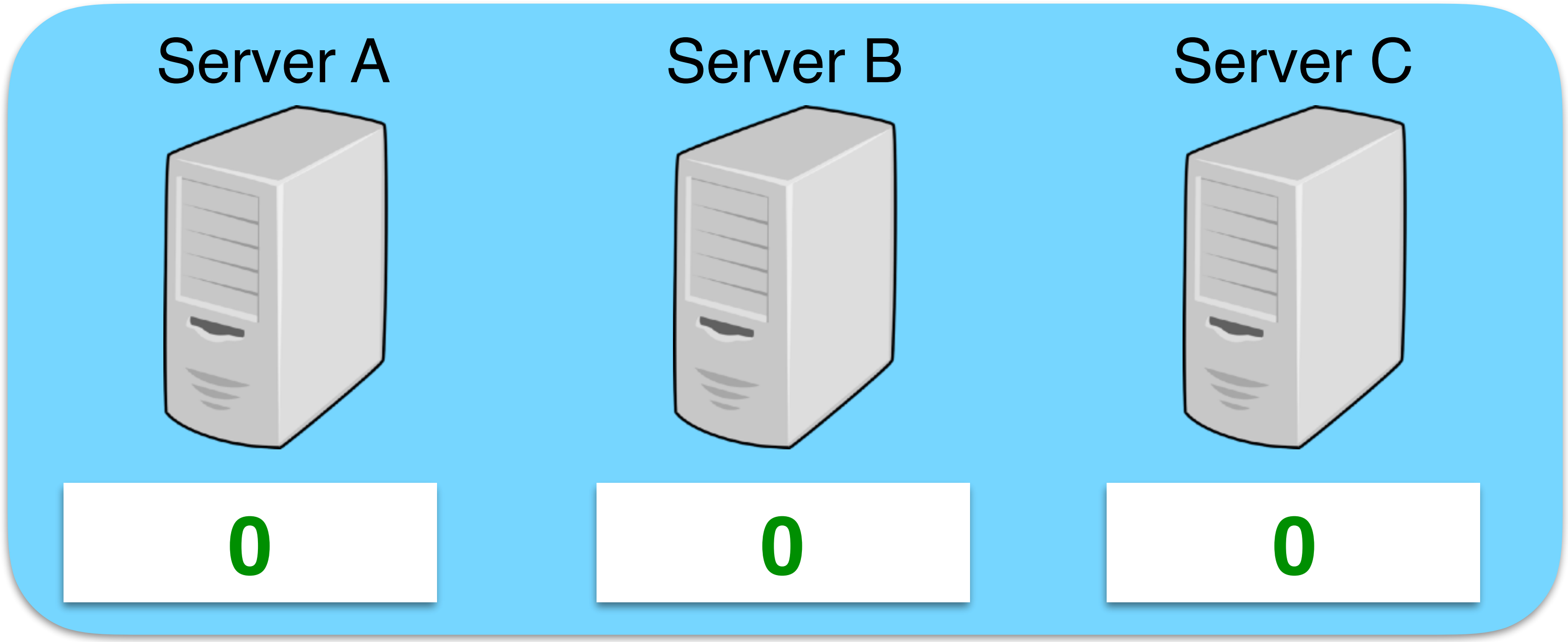
Pick three random “shares” that sum to 1.

In real system, we use a “big” prime

$1 = 15 + (-12) + (-2) \pmod{31}$

Need all three shares to recover the shared value.

Private sums: A “straw-man” scheme



1

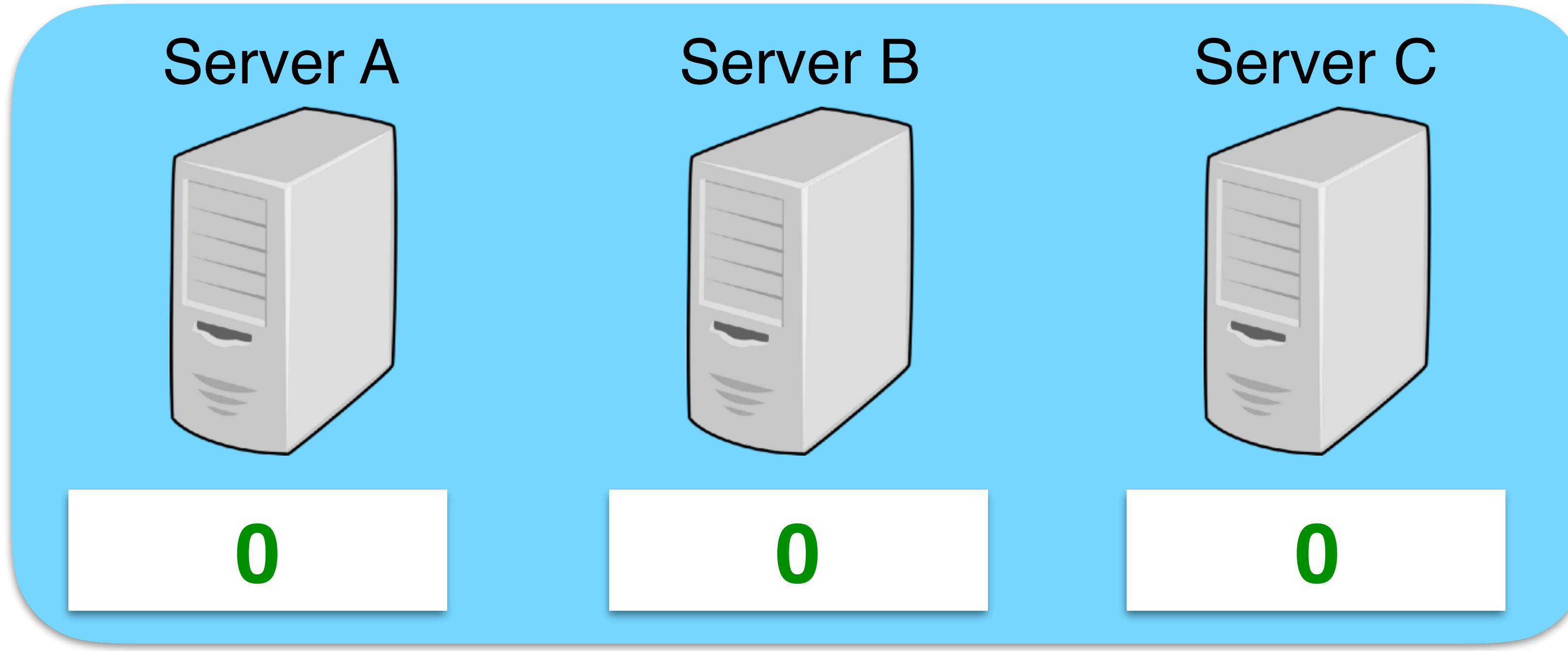
<u>Secret sharing</u>

Pick three random “shares” that sum to 1.

$1 = 15 + (-12) + (-2) \pmod{31}$

Need all three shares to recover the shared value.

Private sums:
A “straw-man” scheme



1

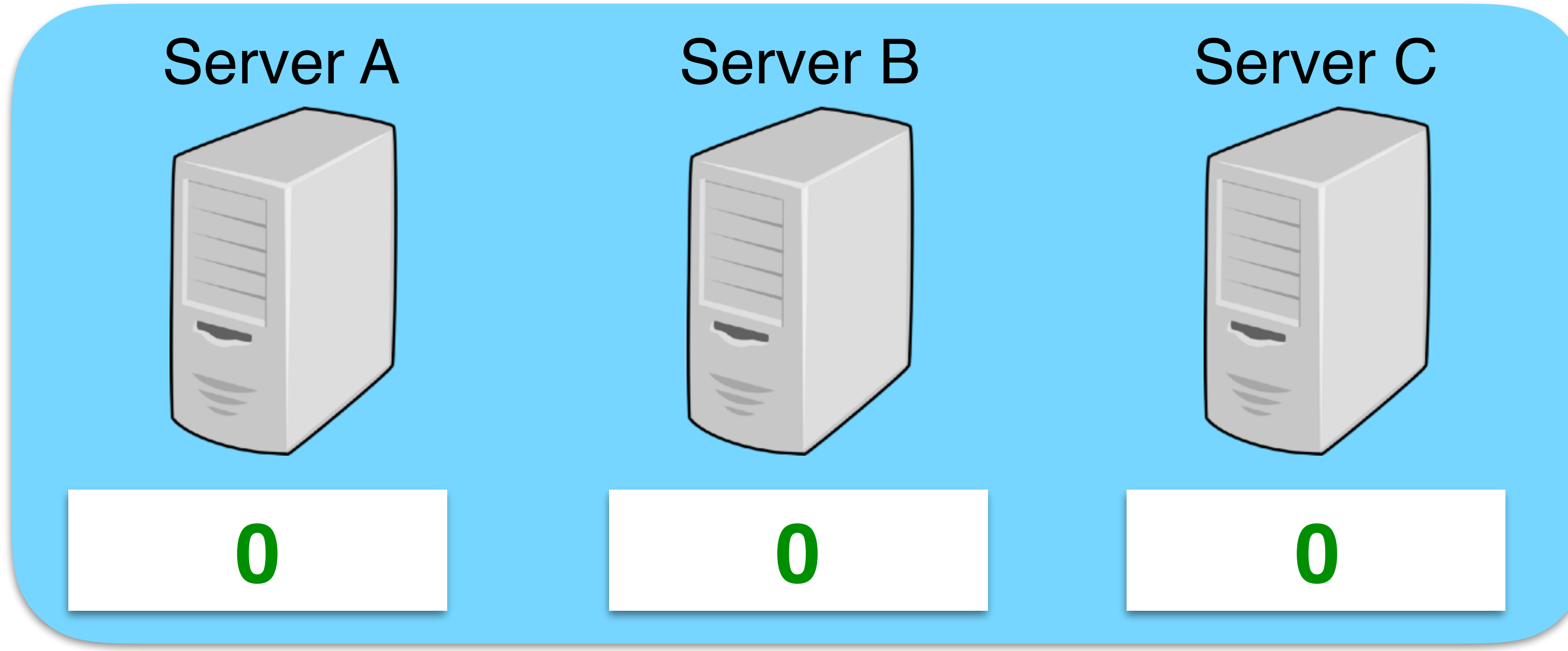
Secret sharing

Pick three random “shares” that sum to 1.

$1 = 15 + (-12) + (-2) \pmod{31}$

Need all three shares to recover the shared value.

Private sums: A "straw-man" scheme



15 -12 -2

1

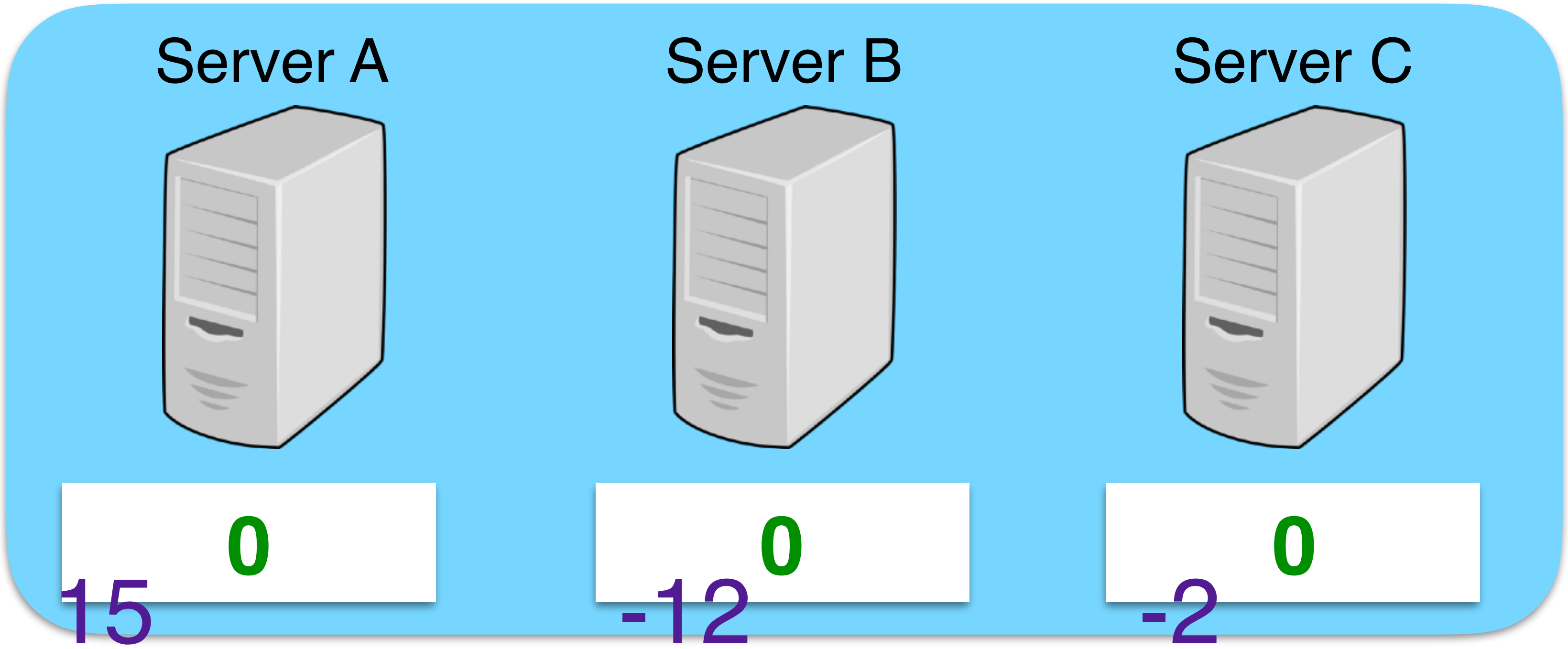
<u>Secret sharing</u>

Pick three random "shares" that sum to 1.

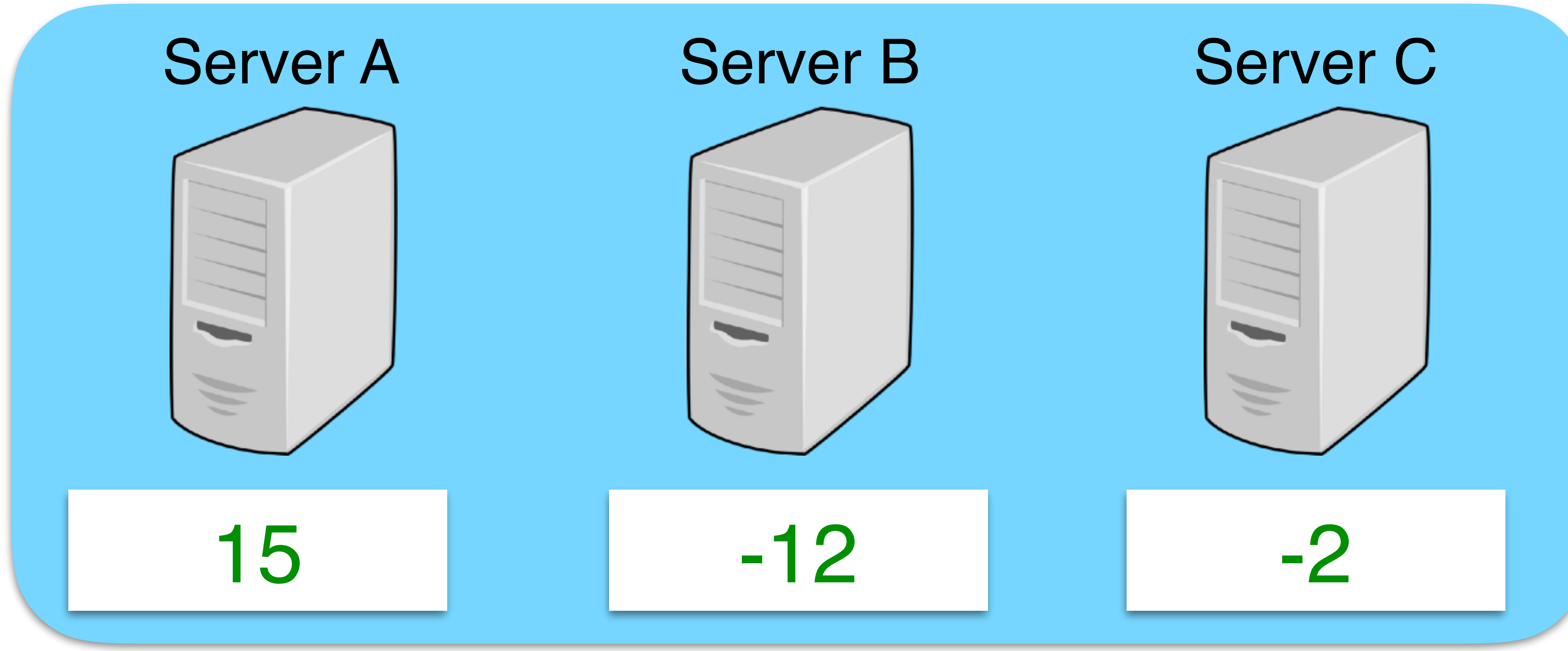
$1 = 15 + (-12) + (-2) \pmod{31}$

Need all three shares to recover the shared value.

Private sums: A “straw-man” scheme

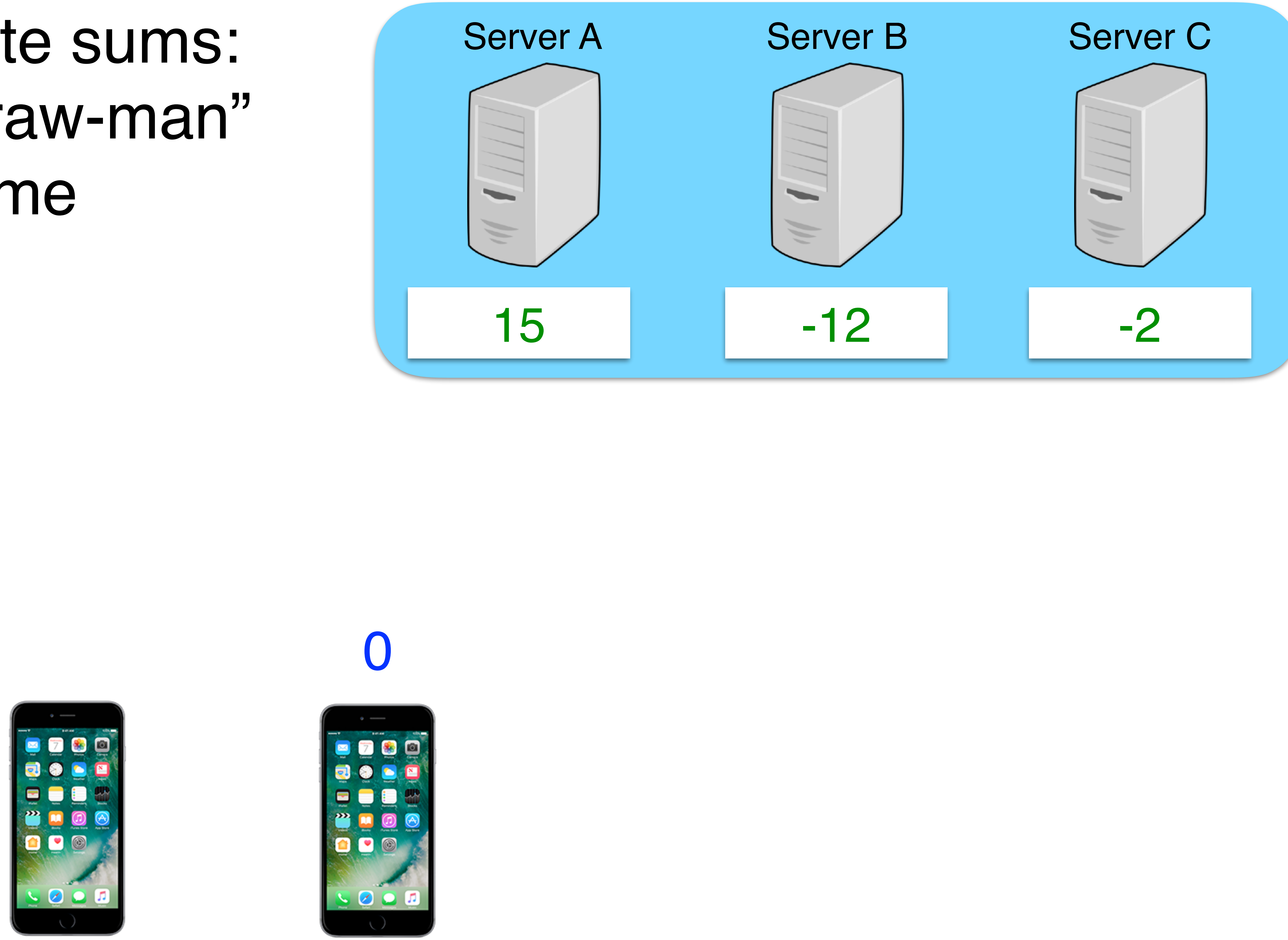


Private sums: A “straw-man” scheme

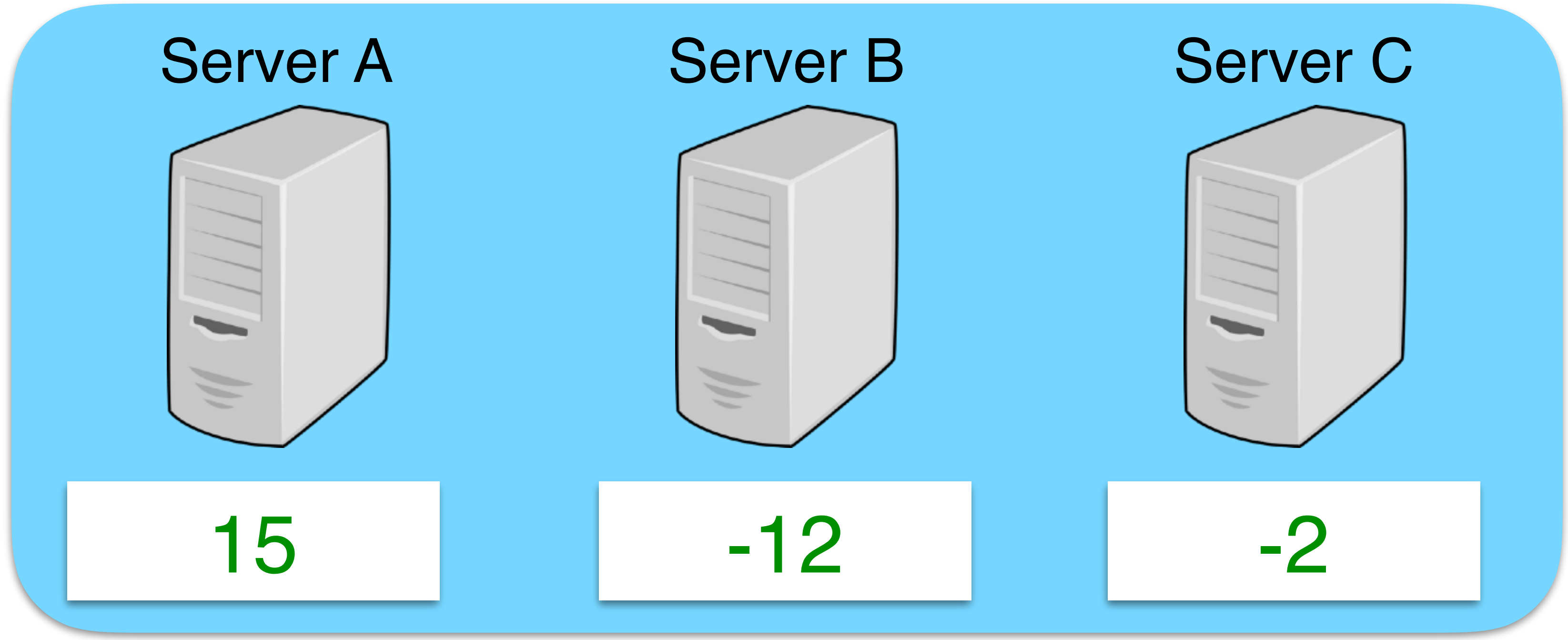


1

Private sums: A “straw-man” scheme

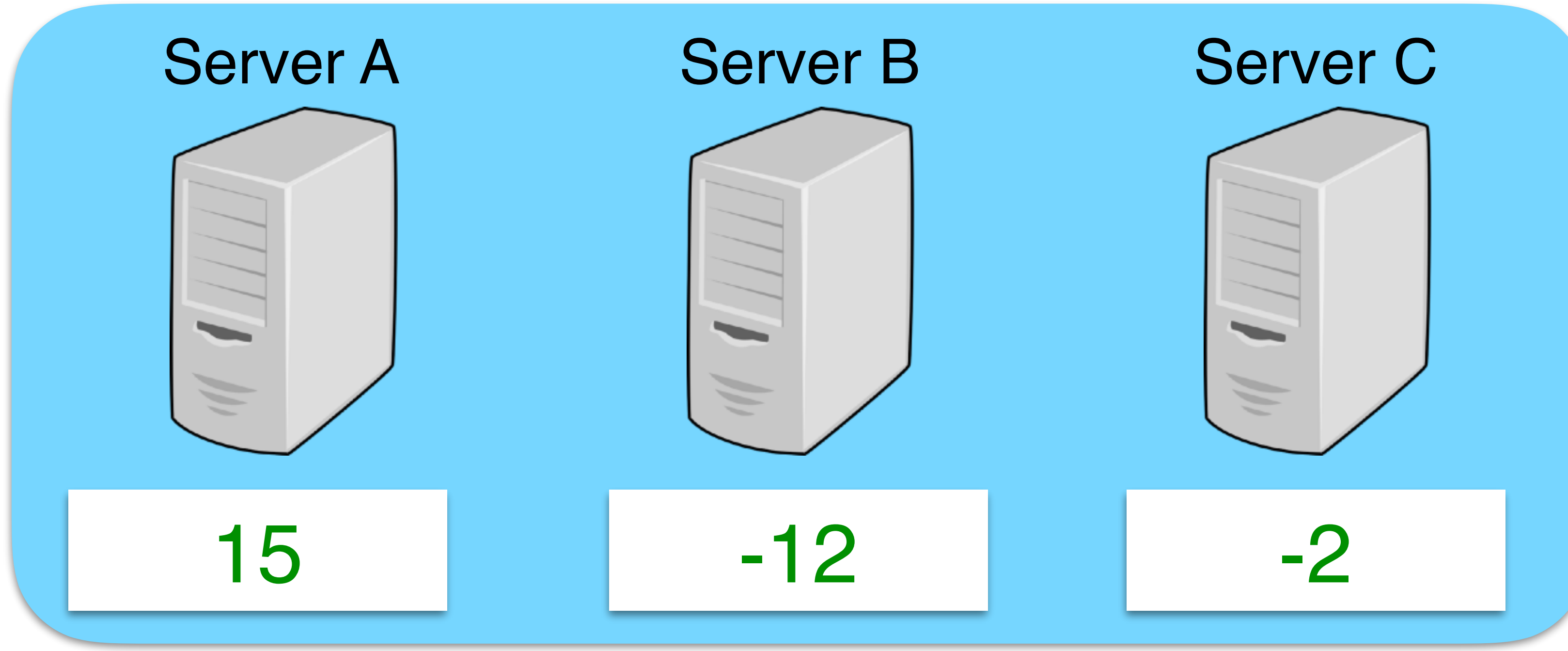


Private sums: A “straw-man” scheme



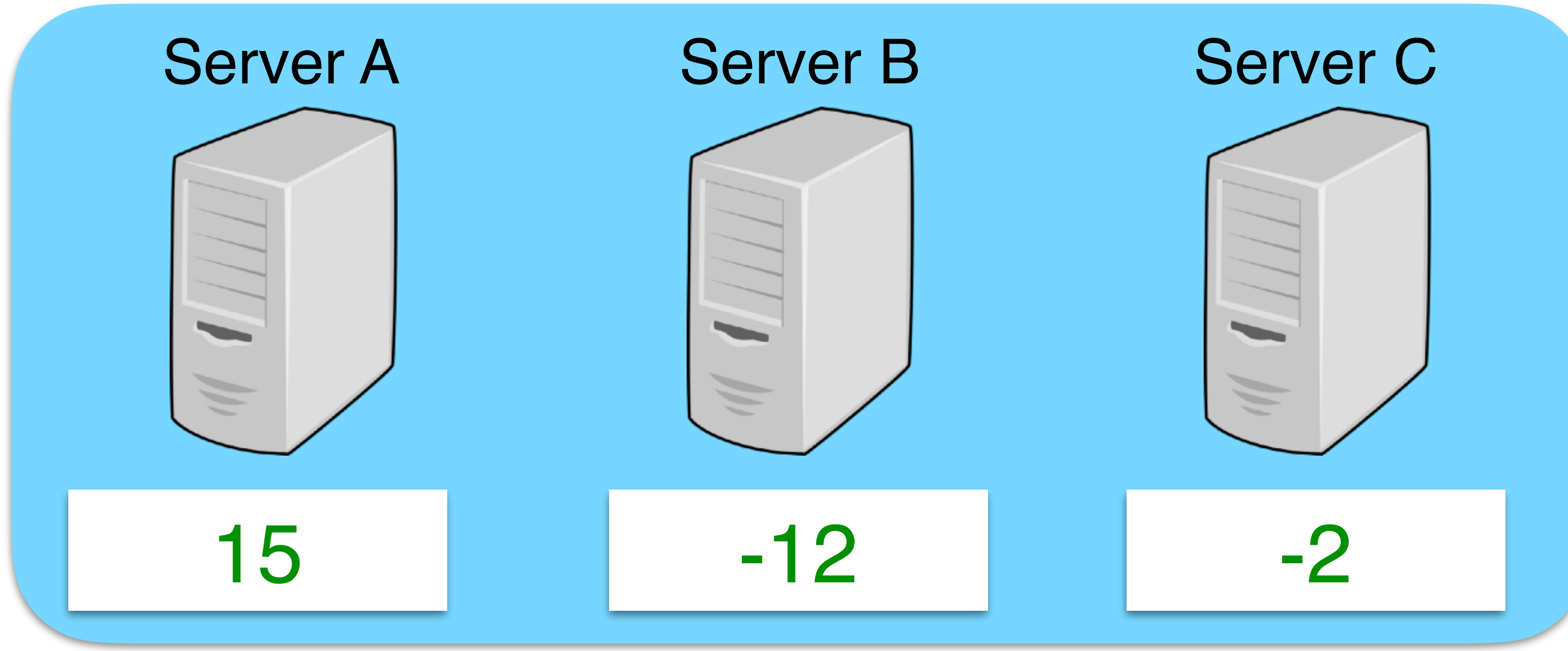
$0 = (-10) + 7 + 3$

Private sums: A “straw-man” scheme



$0 = (-10) + 7 + 3$

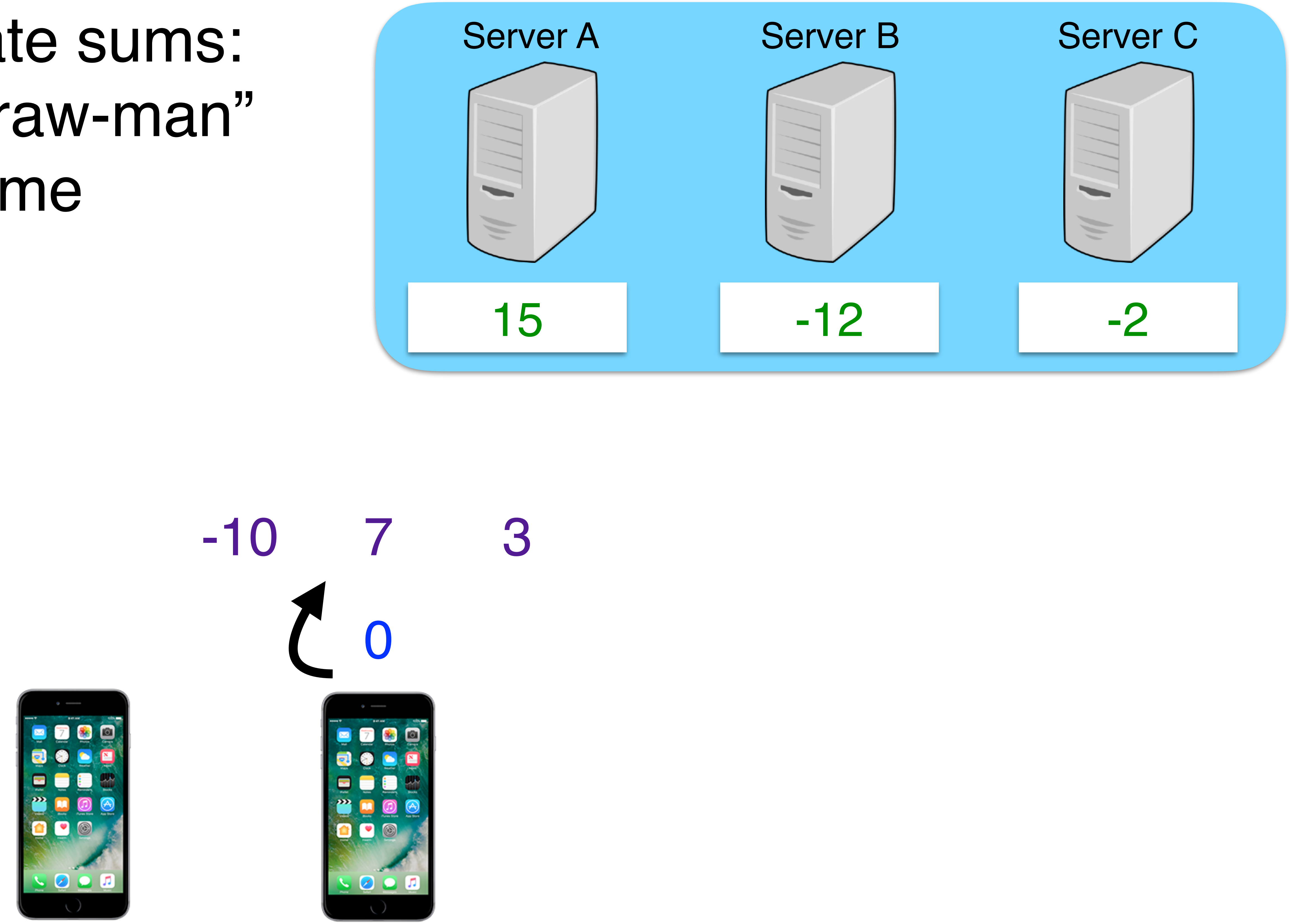
Private sums: A “straw-man” scheme



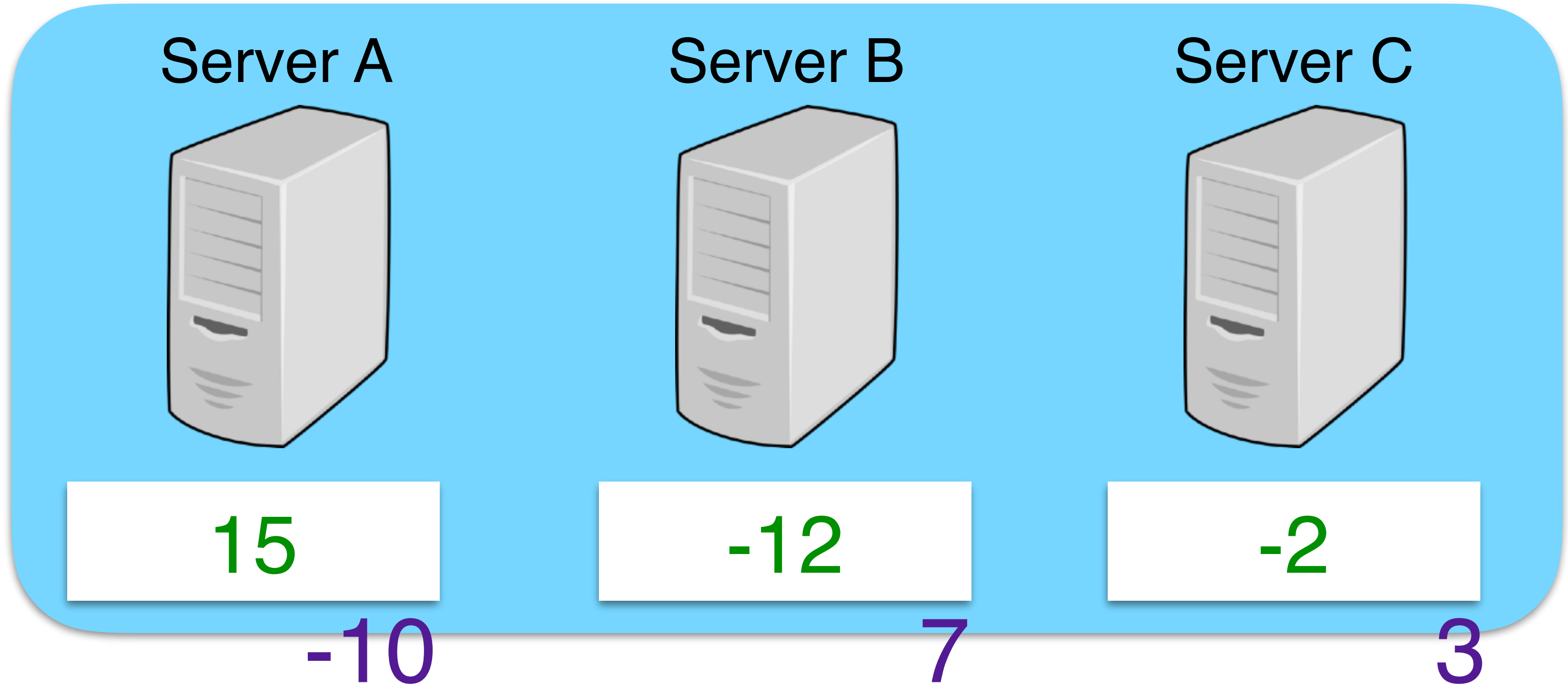
-10 7 3

$0 = (-10) + 7 + 3$

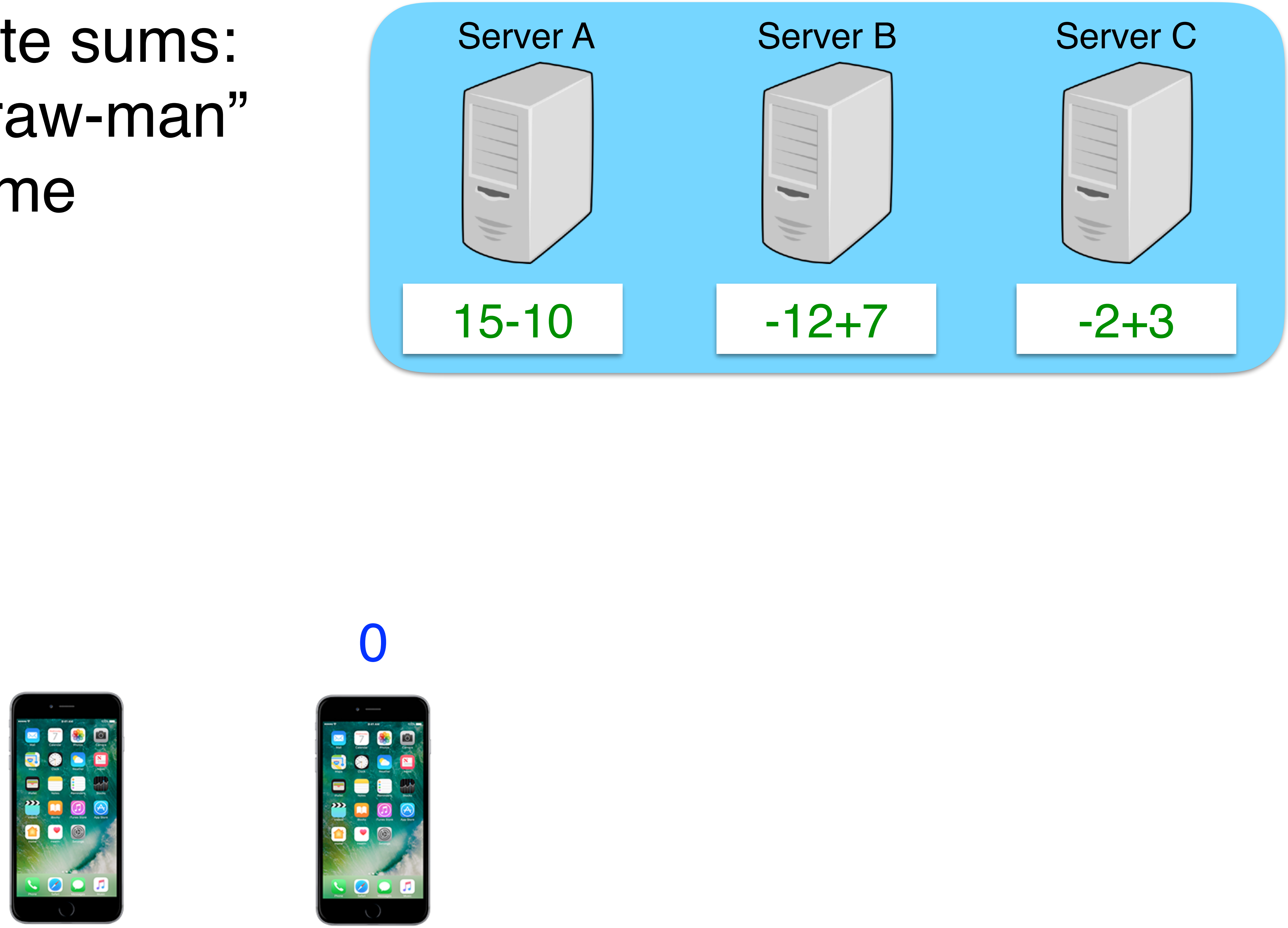
Private sums: A “straw-man” scheme



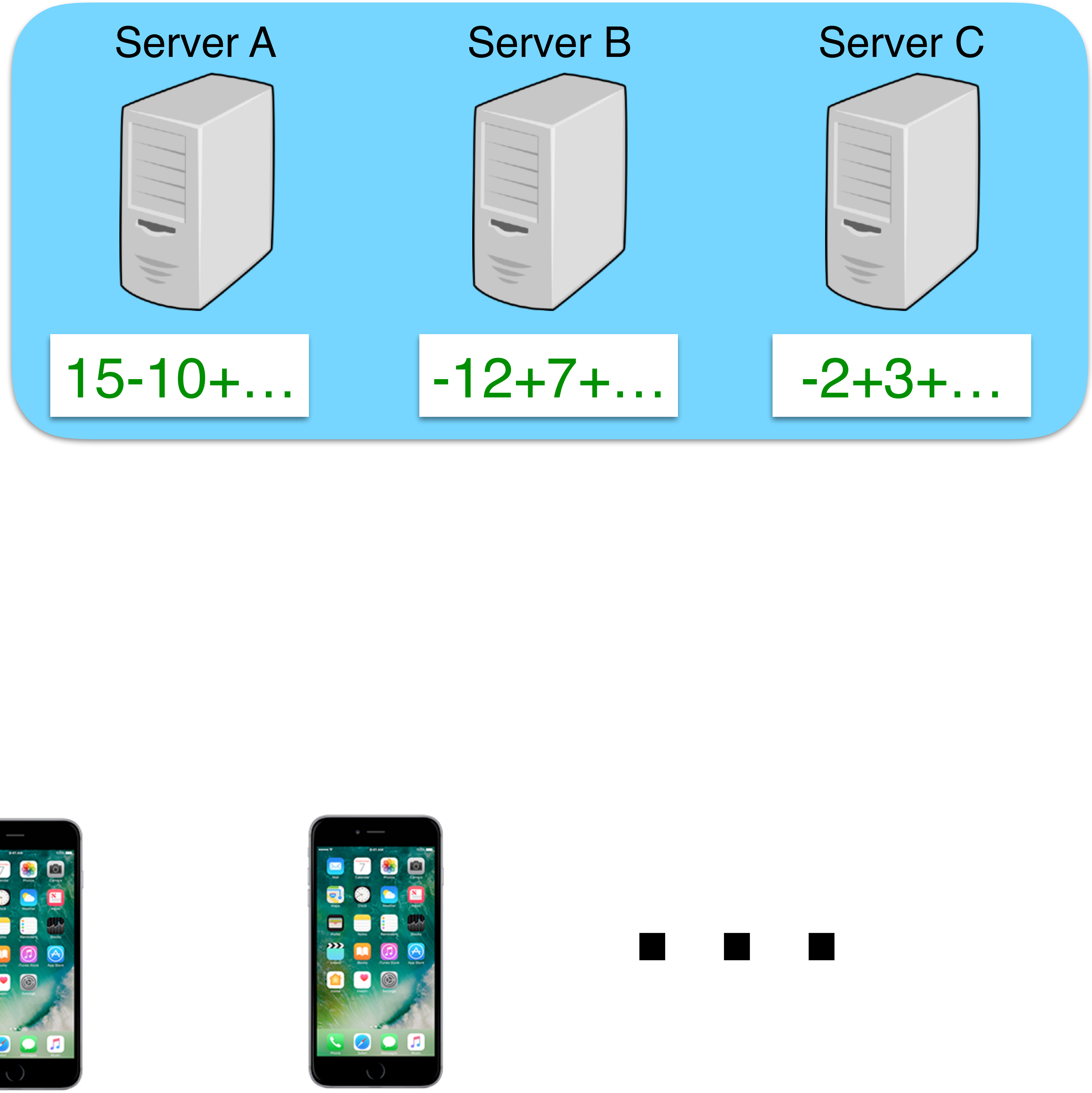
Private sums: A “straw-man” scheme



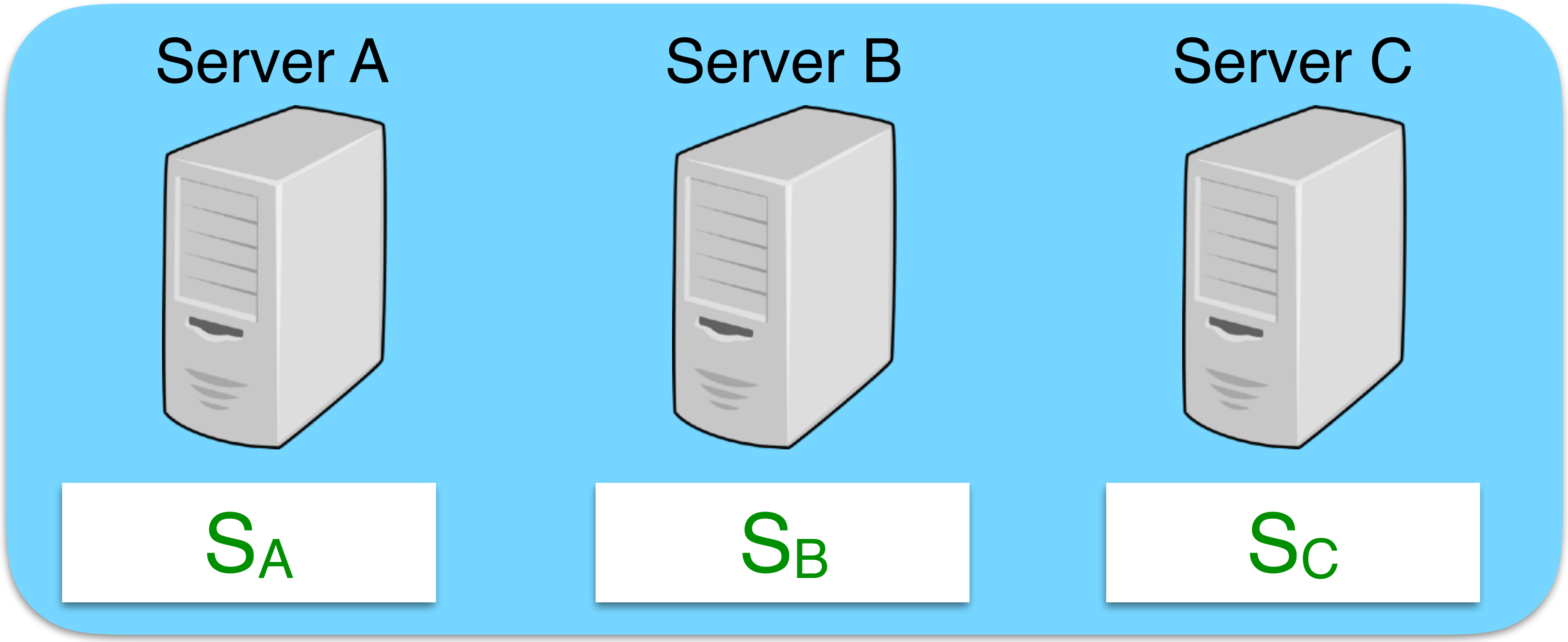
Private sums: A "straw-man" scheme



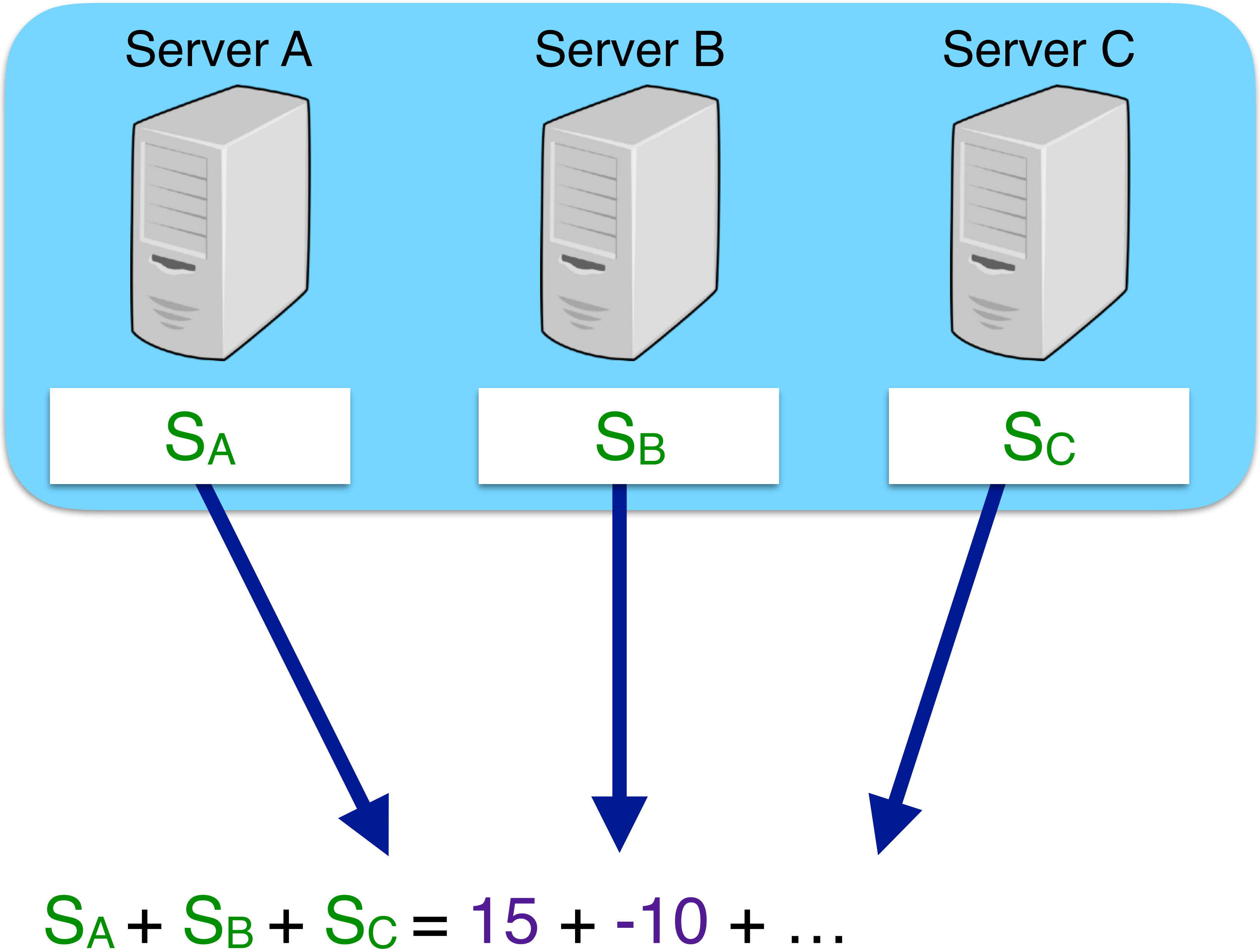
Private sums: A “straw-man” scheme



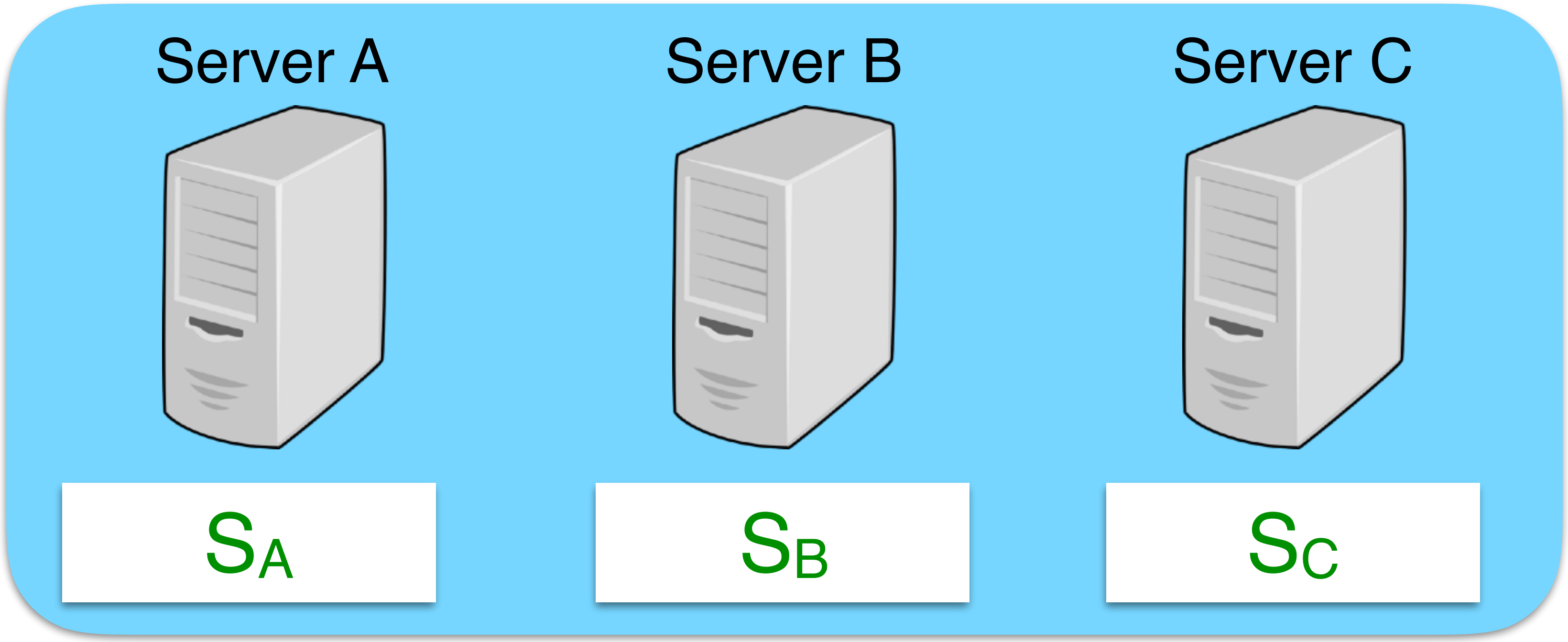
Private sums: A “straw-man” scheme



Private sums:
A “straw-man” scheme

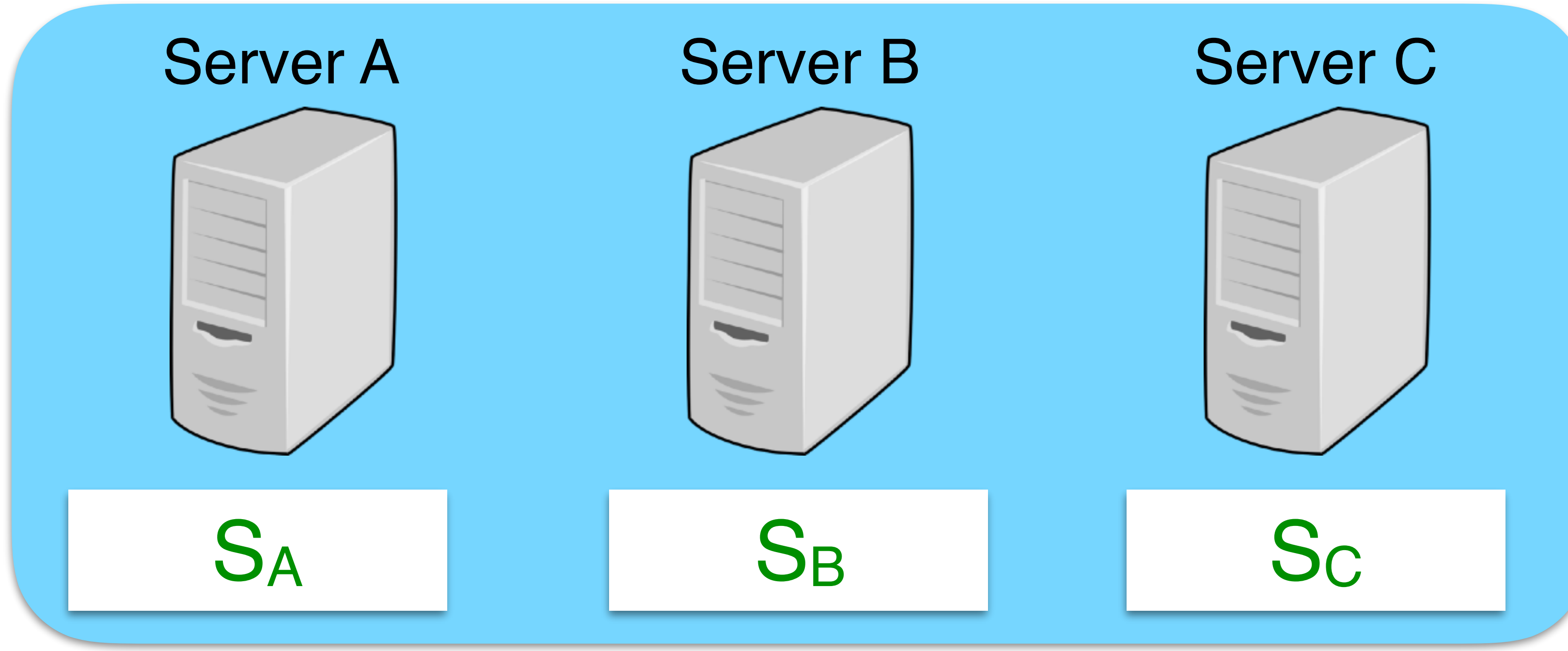


Private sums: A "straw-man" scheme



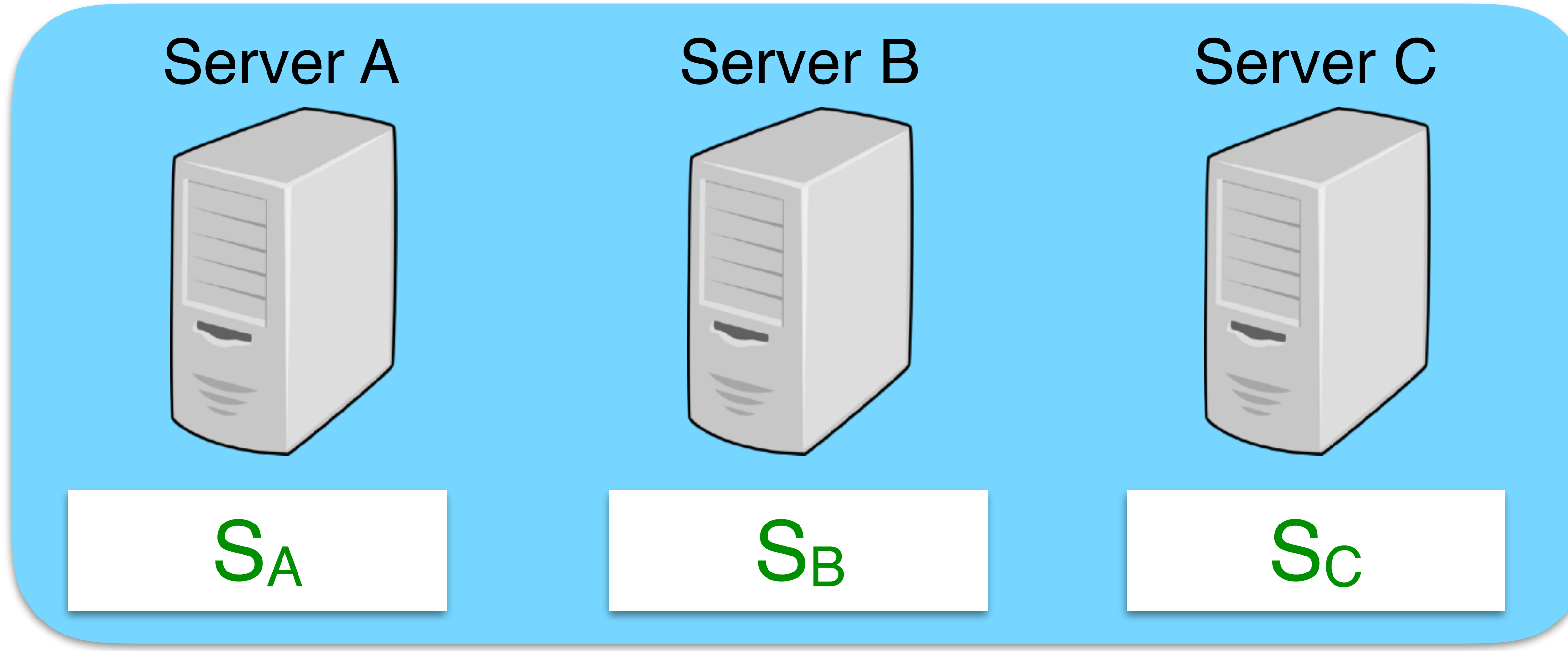
$S_A + S_B + S_C = 15 + -10 + \ldots$

Private sums: A “straw-man” scheme



$$
\begin{aligned}
S_A + S_B + S_C &= 15 + -10 + \ldots \\
&= 1 + 0 + \ldots + 1
\end{aligned}
$$

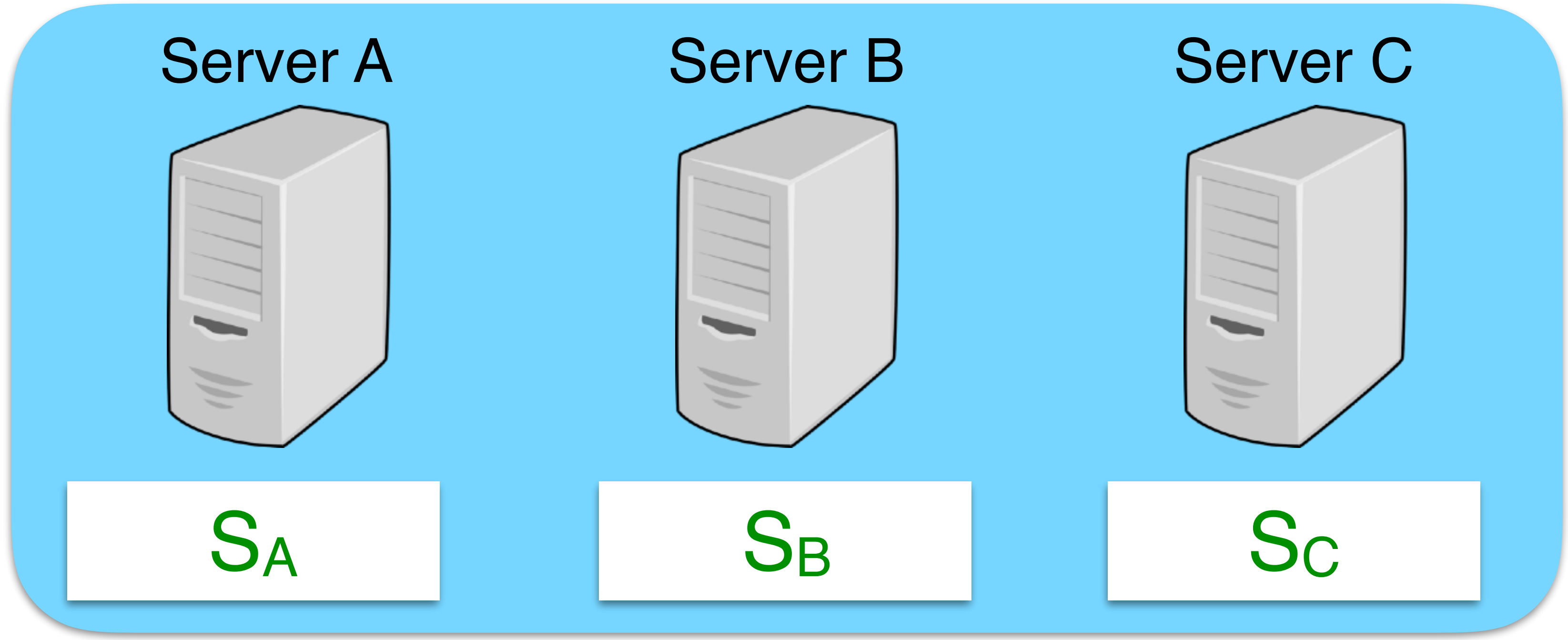
Private sums: A “straw-man” scheme



$$
\begin{aligned}
S_A + S_B + S_C &= 15 + -10 + \ldots \\
&= 1 + 0 + \ldots + 1
\end{aligned}
$$

Servers learn the sum of client values and learn *nothing else*.

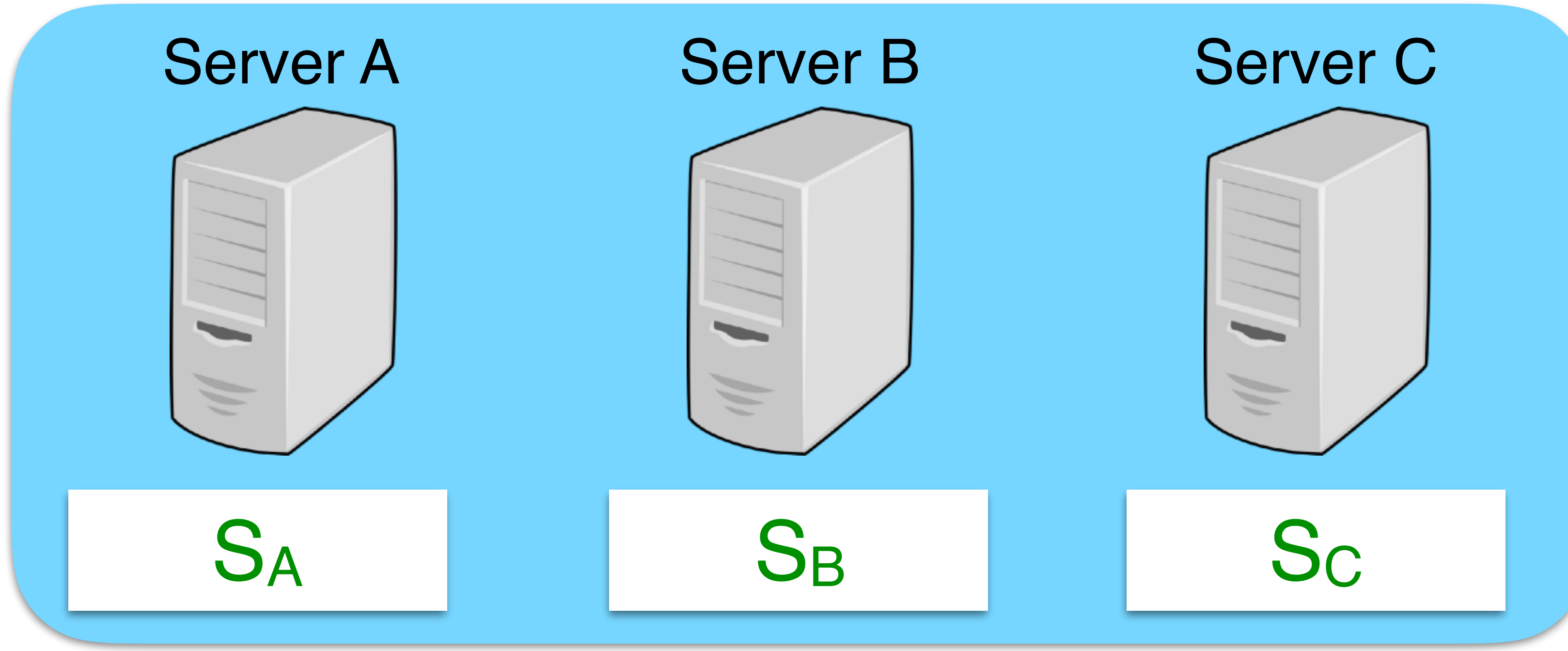
Private sums: A “straw-man” scheme



$$S_A + S_B + S_C = 15 + -10 + \ldots$$
$$= 1 + 0 + \ldots + 1$$

Servers learn the sum of client values and learn *nothing else*.

Private sums: A "straw-man" scheme



$$S_A + S_B + S_C = 15 + -10 + \ldots$$
$$= 1 + 0 + \ldots + 1$$

Learn that three phones are on the Bay Bridge—don't know which three

Computing private sums

Computing private sums

Exact correctness: If everyone follows the protocol, servers compute the sum of all x_is.

Privacy: Any proper subset of the servers learns nothing but the sum of the x_is.

Efficiency: Follows by inspection.

Computing private sums

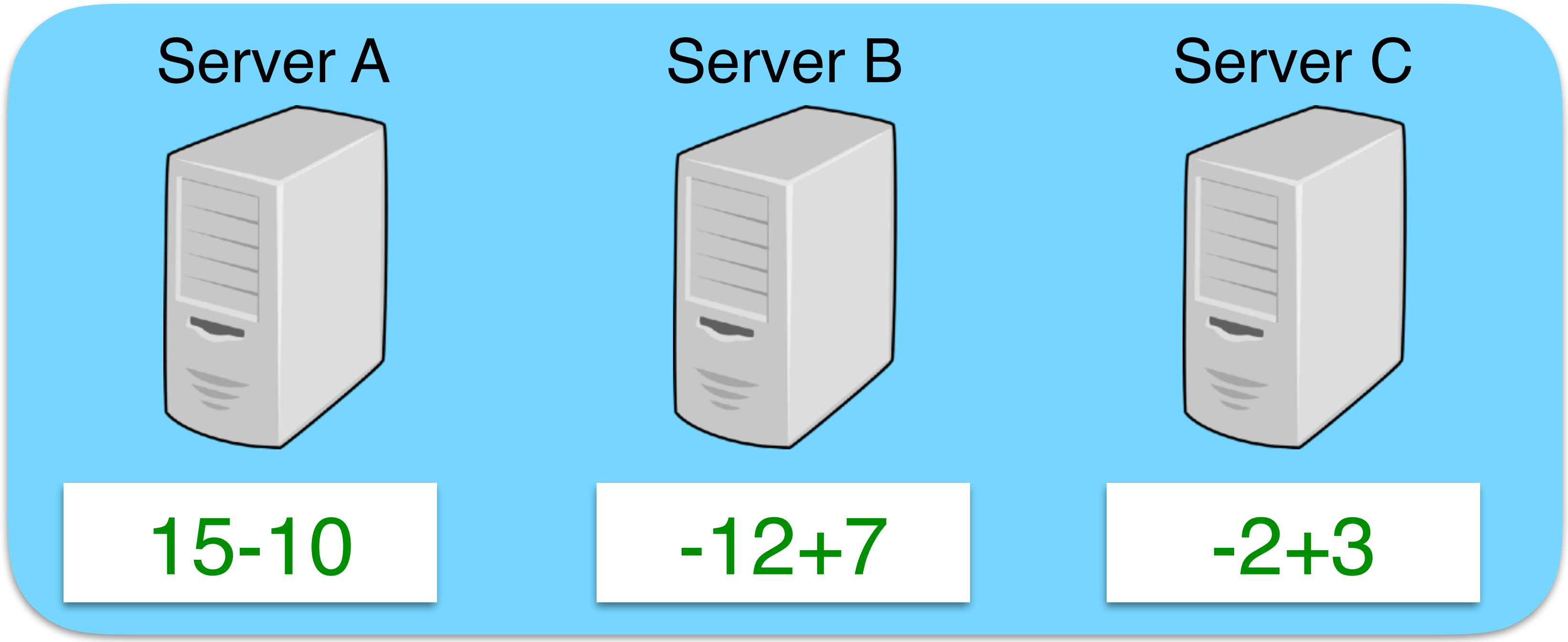
Exact correctness: If everyone follows the protocol, servers compute the sum of all x_is.

Privacy: Any proper subset of the servers learns nothing but the sum of the x_is.

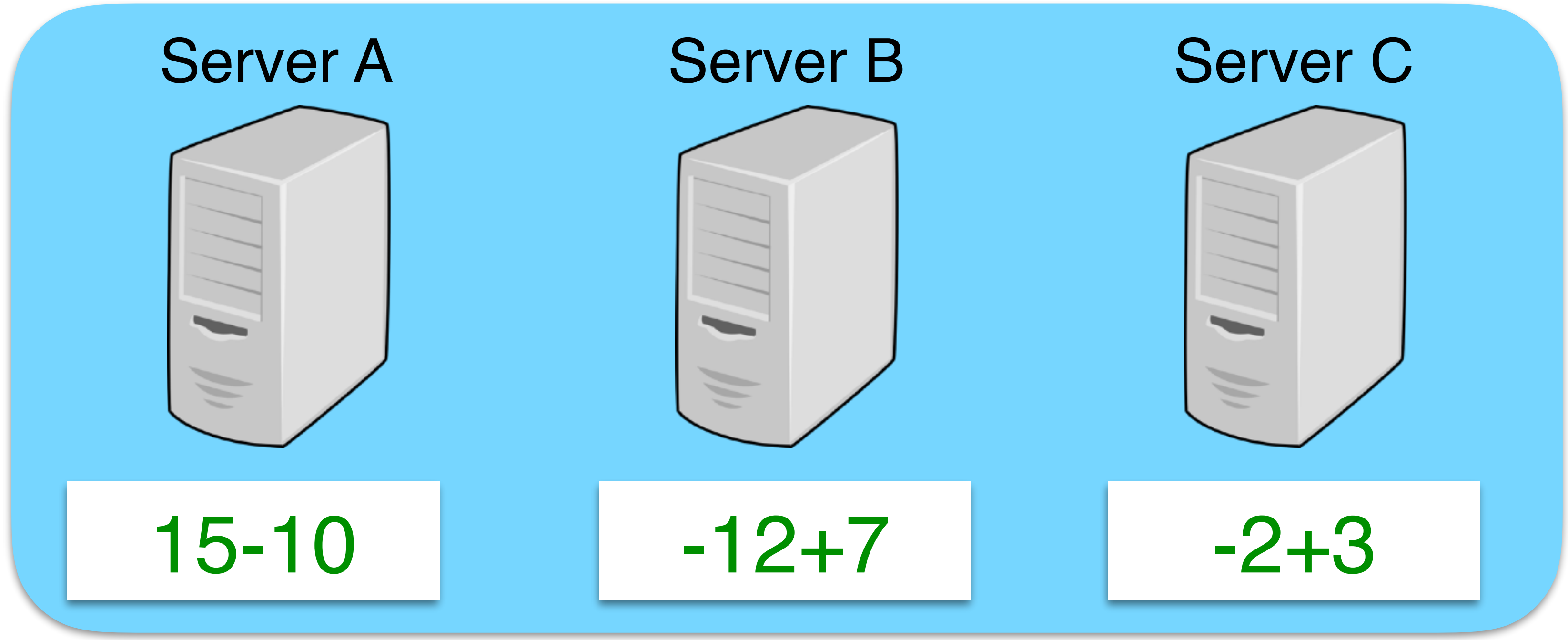
Efficiency: Follows by inspection.

Robustness: ???

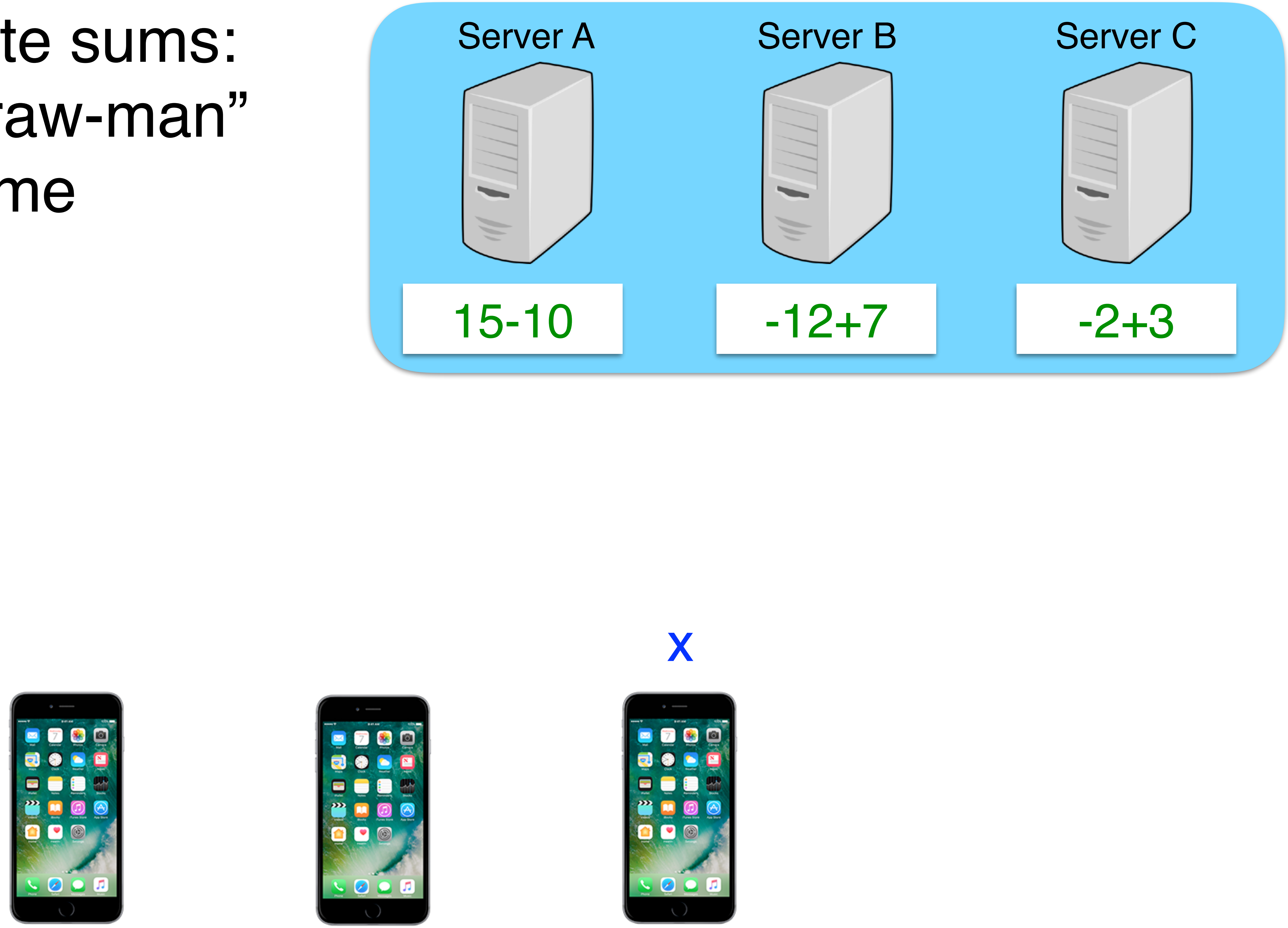
Private sums:
A “straw-man” scheme



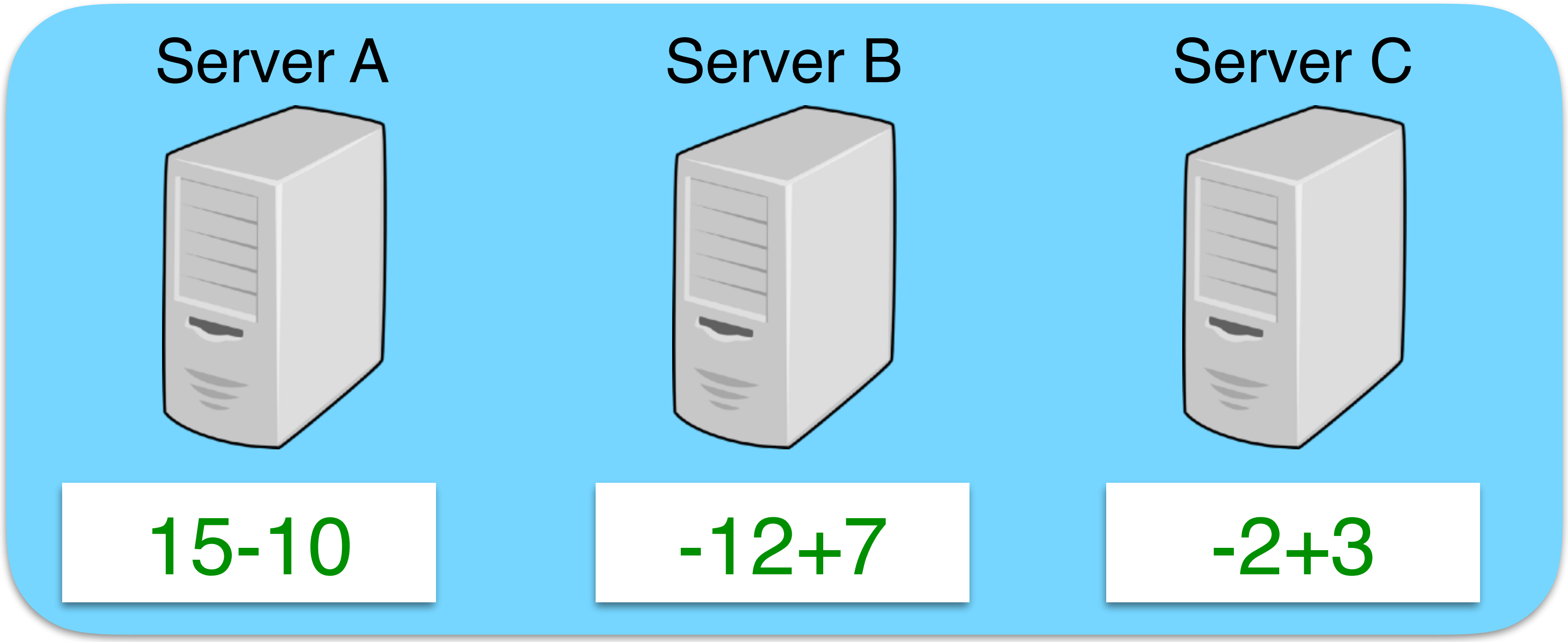
Private sums: A “straw-man” scheme



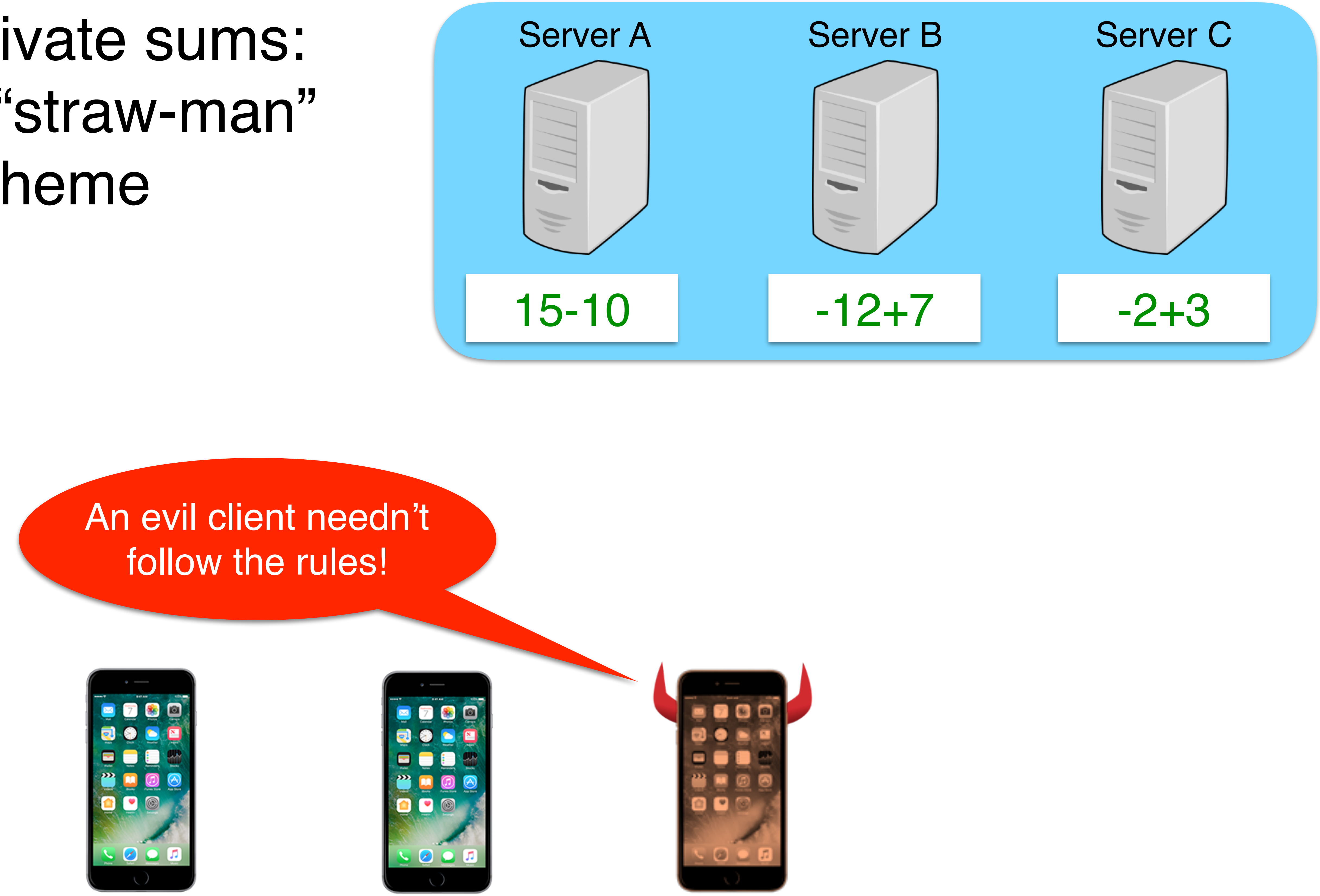
Private sums: A "straw-man" scheme



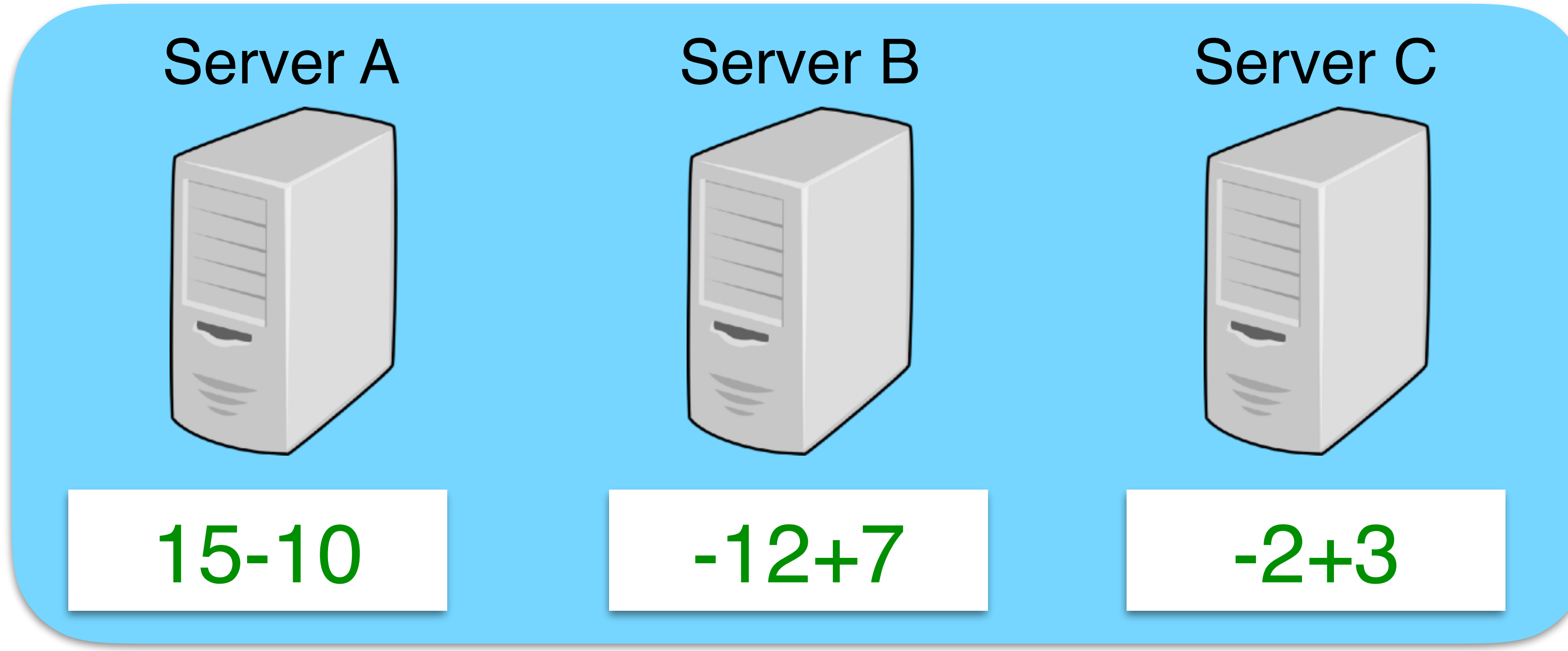
Private sums: A "straw-man" scheme



Private sums: A "straw-man" scheme



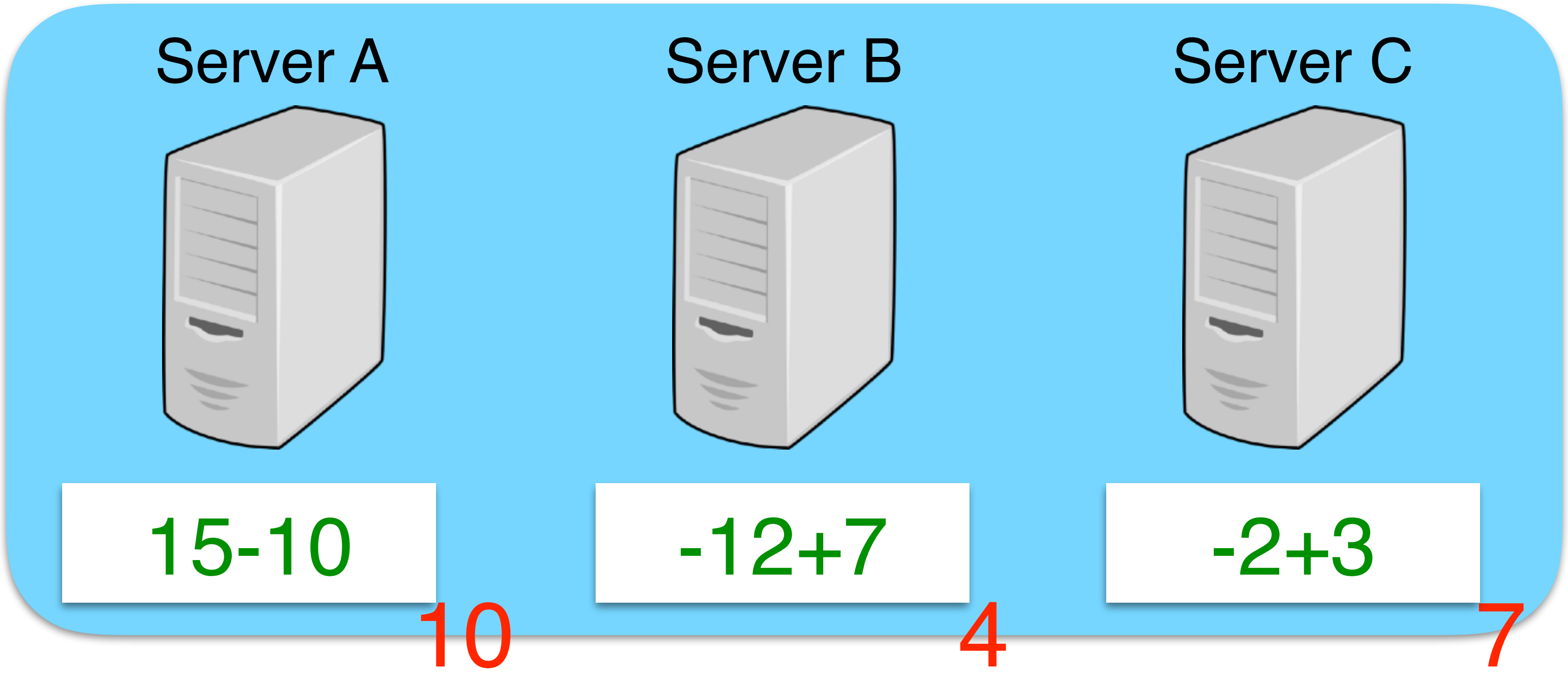
Private sums: A “straw-man” scheme



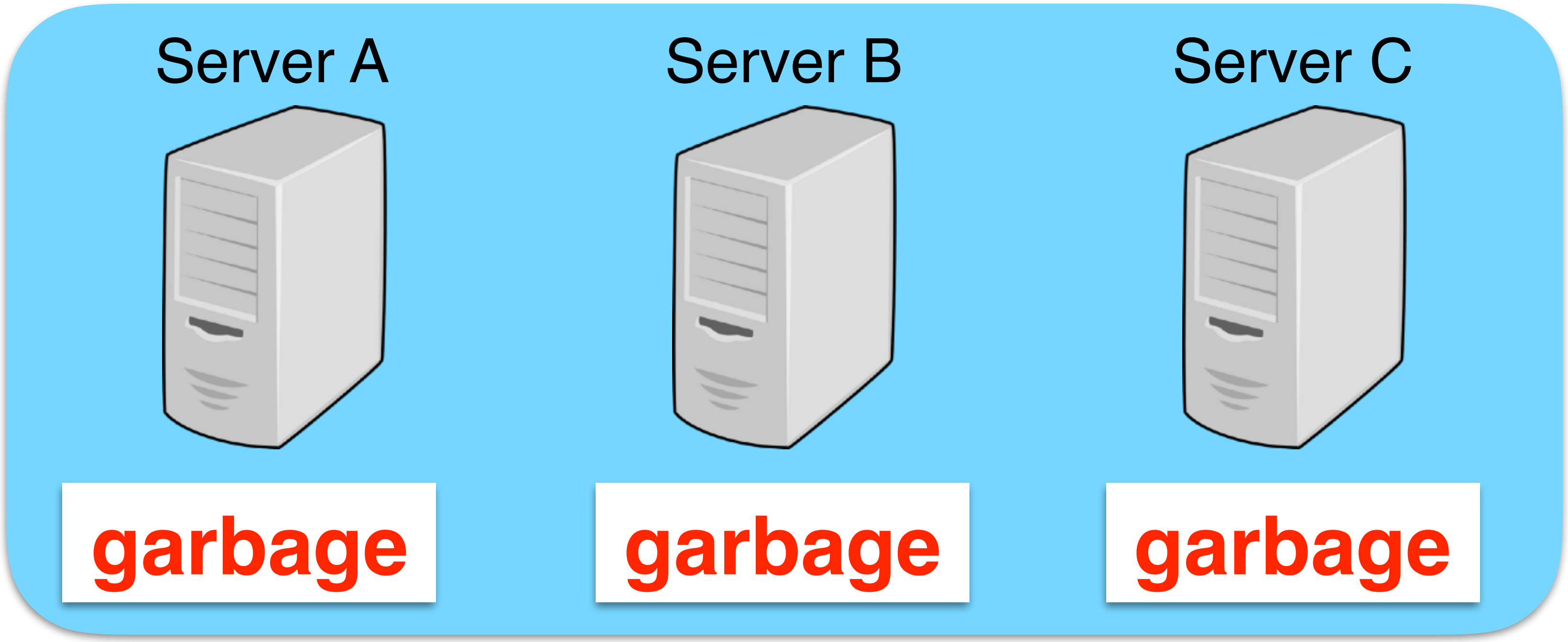
An evil client needn’t follow the rules!

$10 + 4 + 7 = \mathbf{21}$

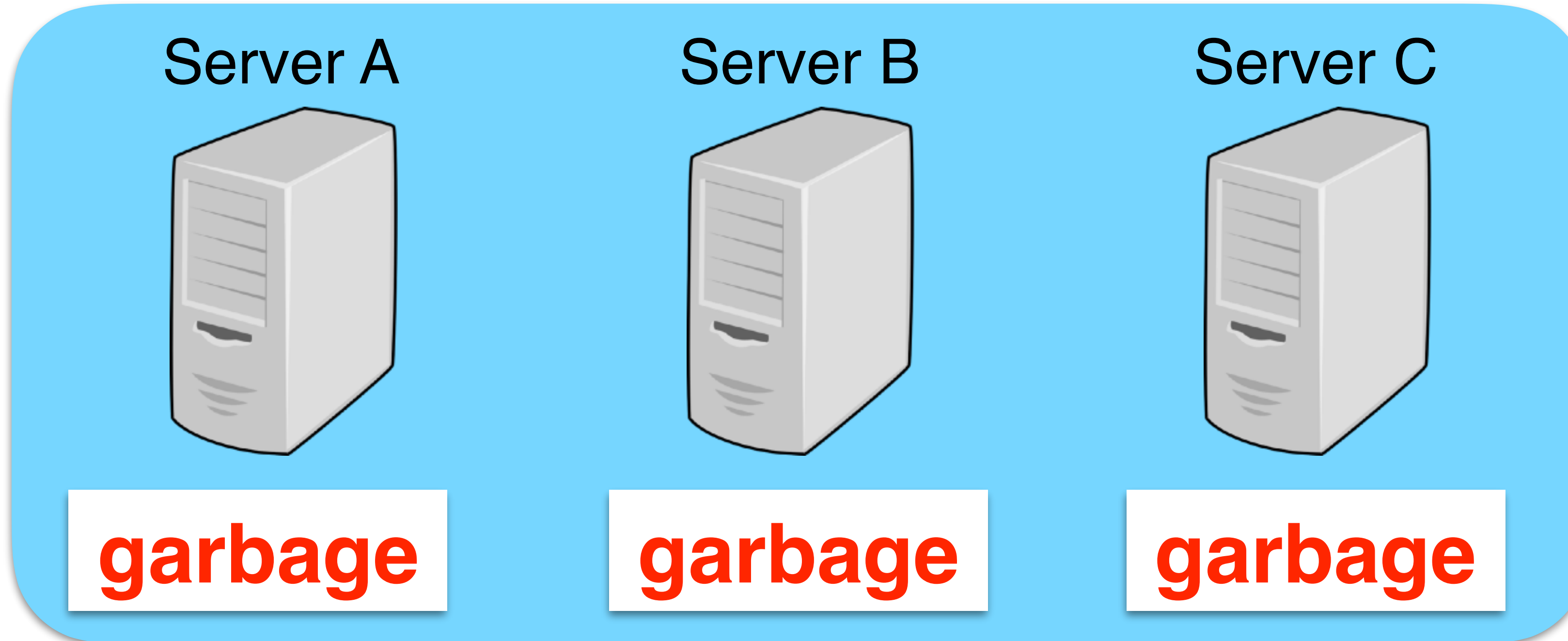
Private sums: A “straw-man” scheme



Private sums: A “straw-man” scheme



Private sums: A “straw-man” scheme



A single bad client **can undetectably corrupt** the sum

Users have **incentives** to cheat

Typical defenses (NIZKs) are costly

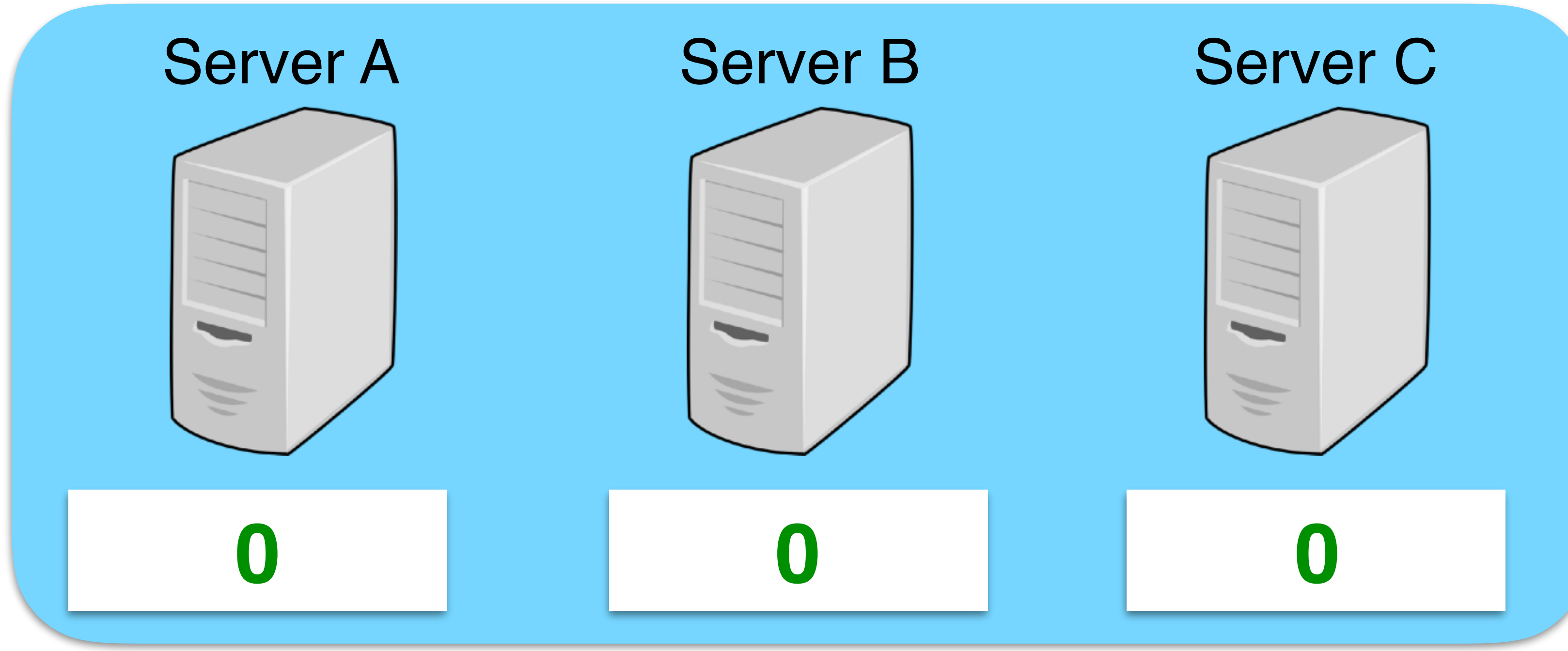
Outline

- Background: The private aggregation problem
- **A straw-man solution for private sums**
- Providing robustness with SNIPs
- Evaluation
- Encodings for <u>complex</u> aggregates

Outline

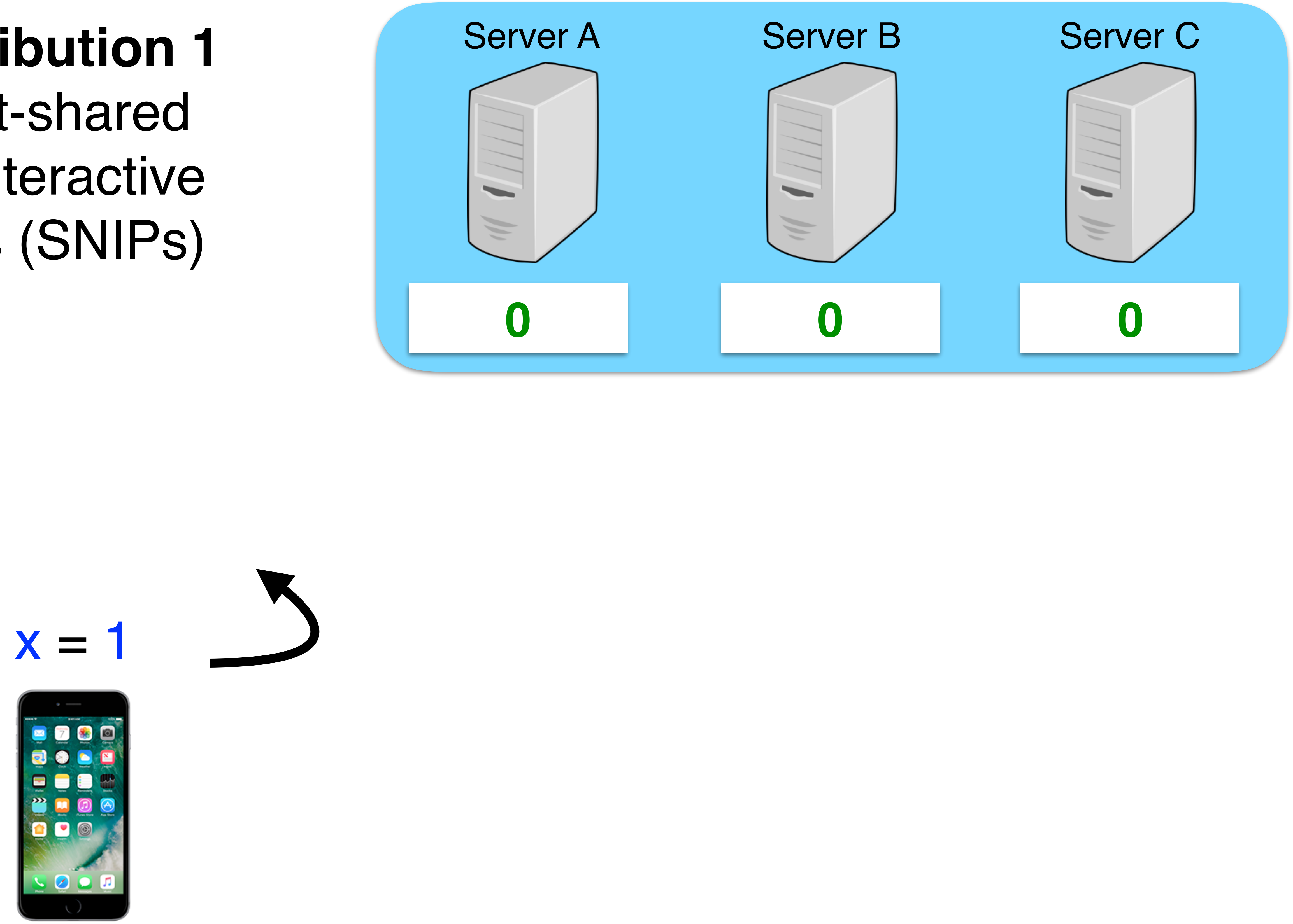
- Background: The private aggregation problem
- A straw-man solution for private sums
- **Providing robustness with SNIPs**
- Evaluation
- Encodings for <u>complex</u> aggregates

Contribution 1
Secret-shared non-interactive proofs (SNIPs)

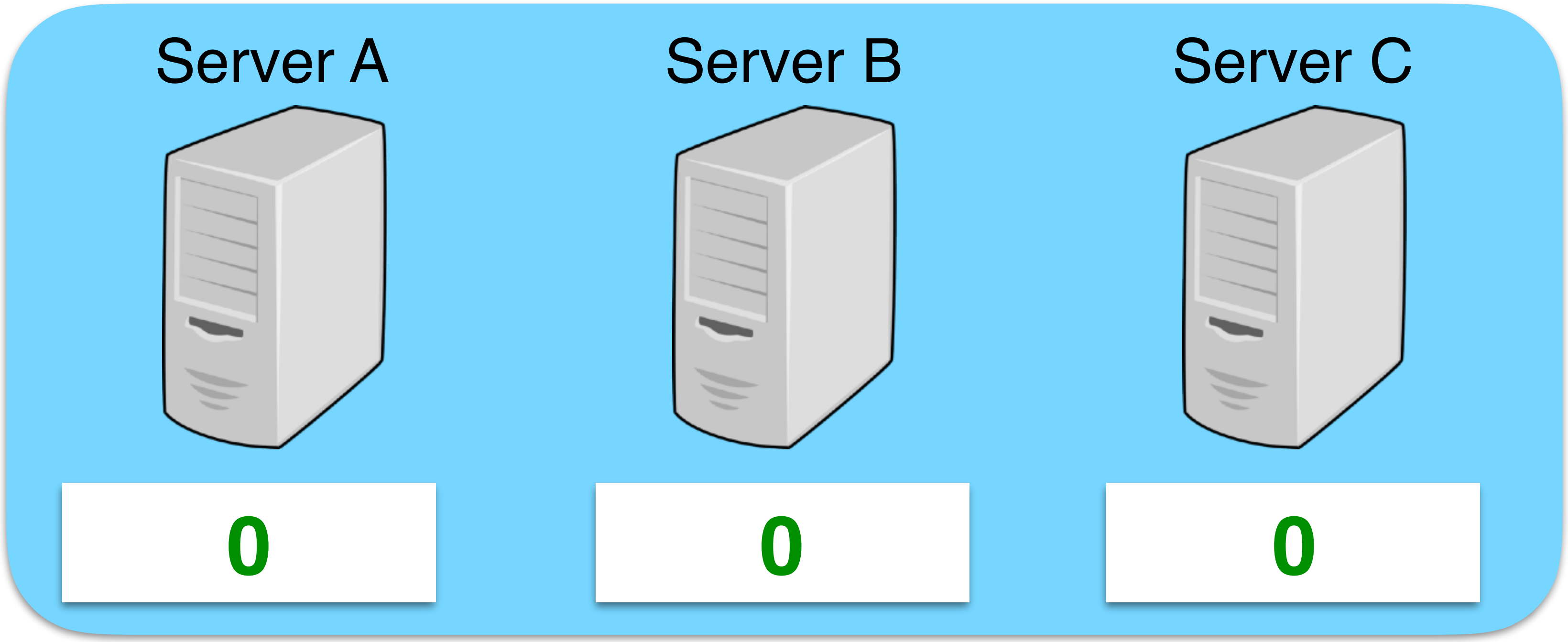


$x = 1$

Contribution 1
Secret-shared non-interactive proofs (SNIPs)



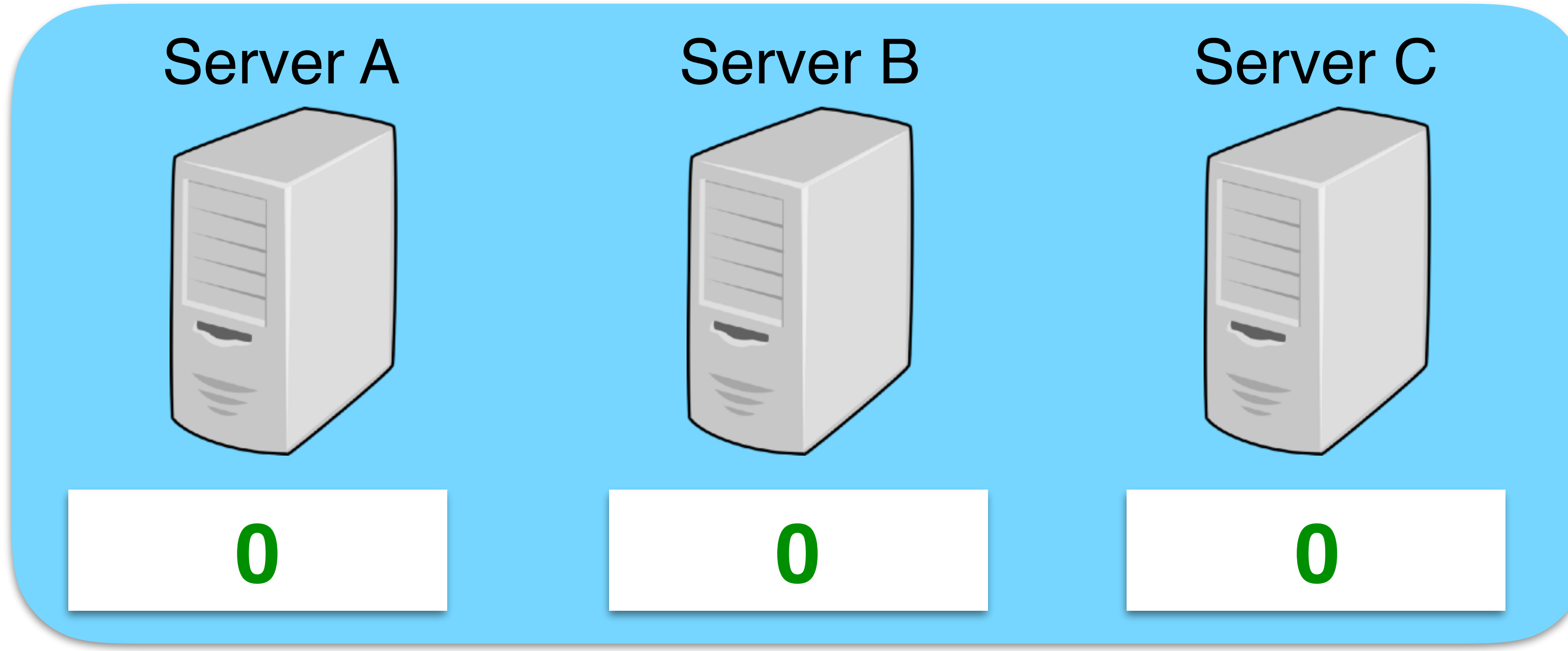
Contribution 1
Secret-shared
non-interactive
proofs (SNIPs)



$15 + (-12) + (-2) = 1$

$x = 1$

Contribution 1
Secret-shared non-interactive proofs (SNIPs)

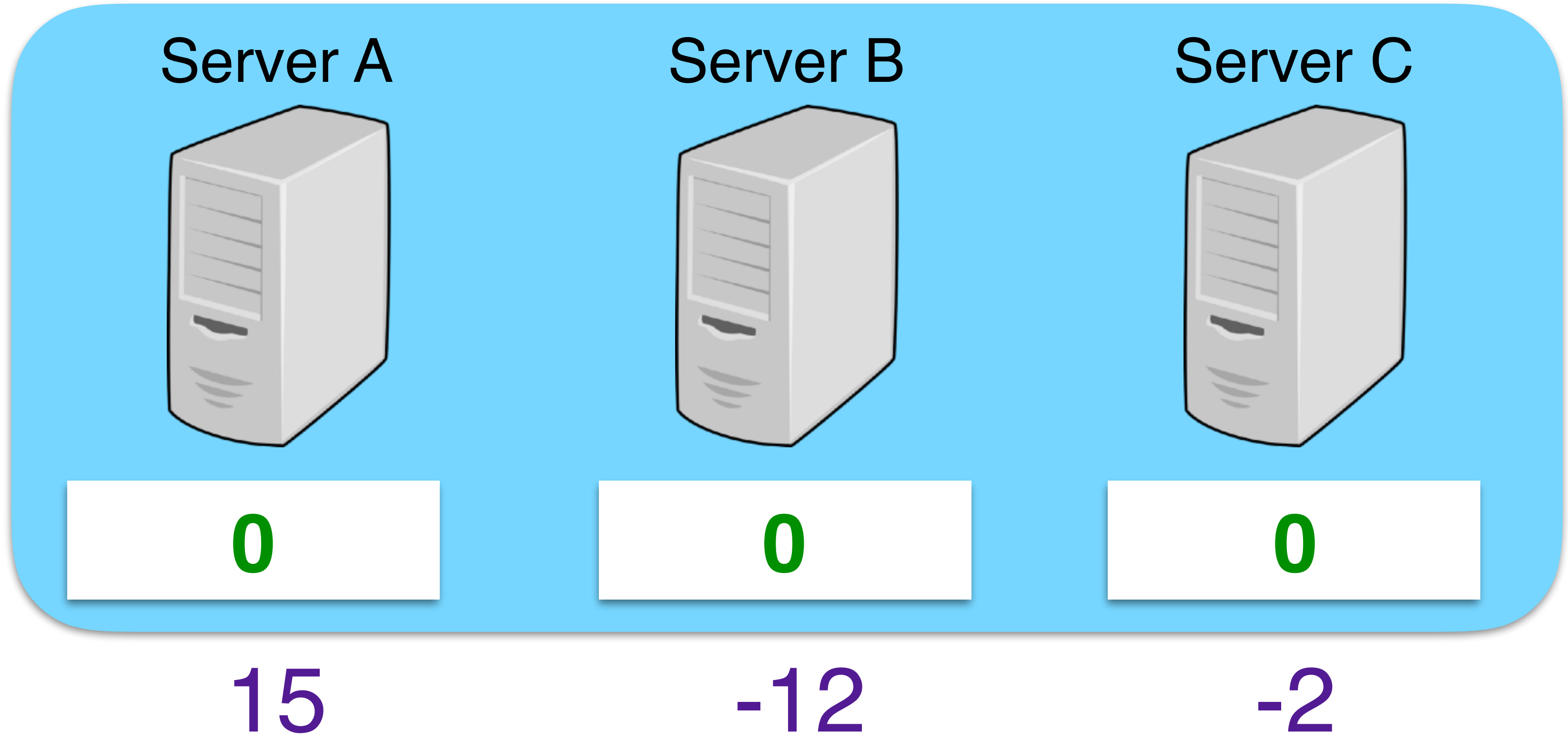


15 -12 -2

x = 1

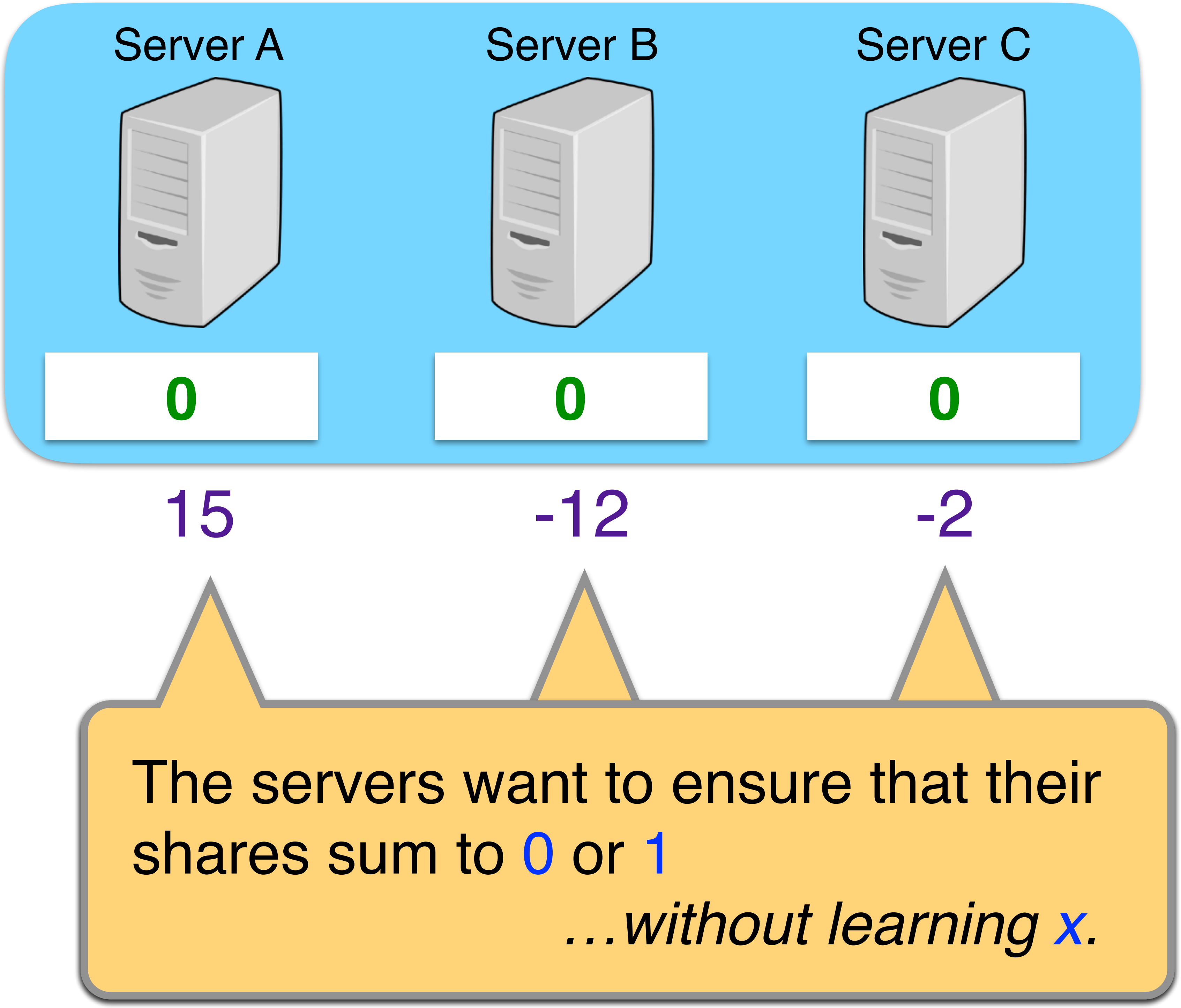
Contribution 1

Secret-shared non-interactive proofs (SNIPs)



$x = 1$

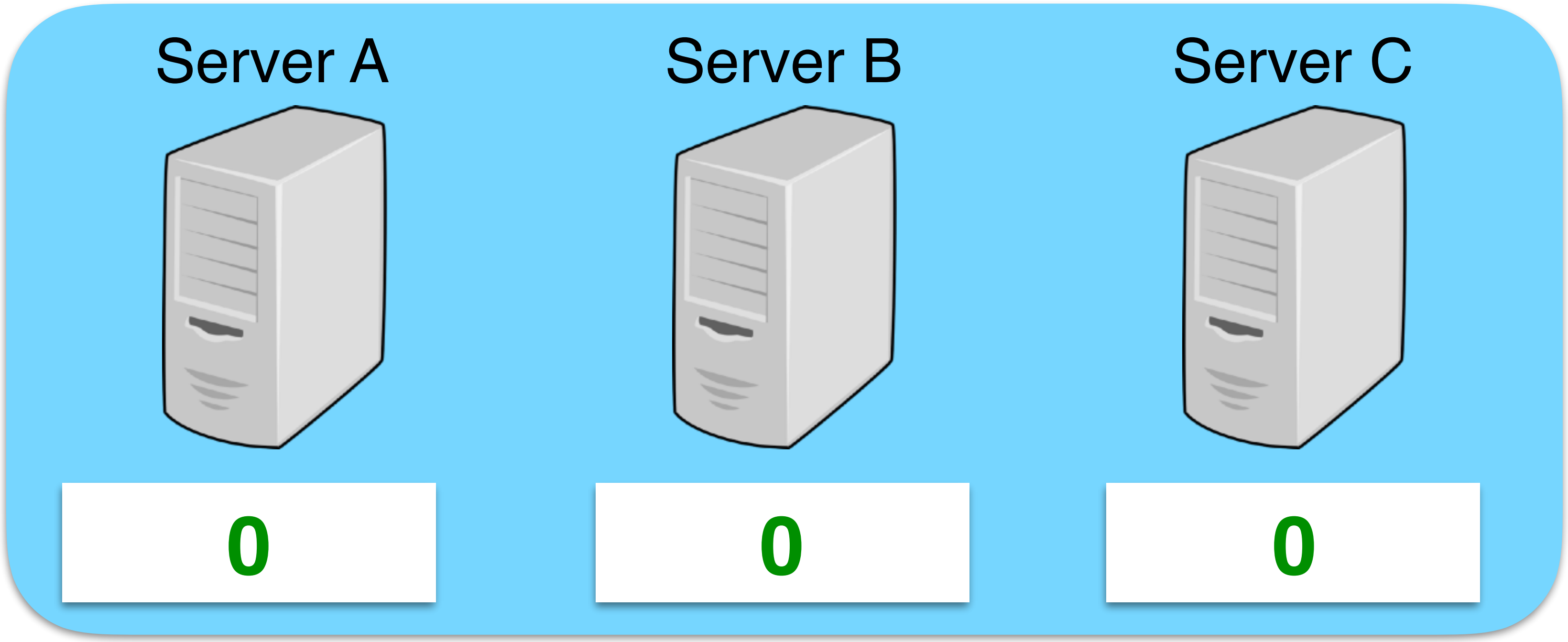
Contribution 1
Secret-shared non-interactive proofs (SNIPs)



x = 1

Contribution 1

Secret-shared non-interactive proofs (SNIPs)



x_a x_b x_c

$x = 1$

More generally, servers

- hold shares of the client’s private value x
- hold an <u>arbitrary</u> public predicate Valid(·)
  – expressed as an arithmetic circuit
- want to test if “Valid(x)” holds, without leaking x

Contribution 1
Secret-shared non-interactive proofs (SNIPs)

Server A Server B Server C

0 0 0

x_a x_b x_c

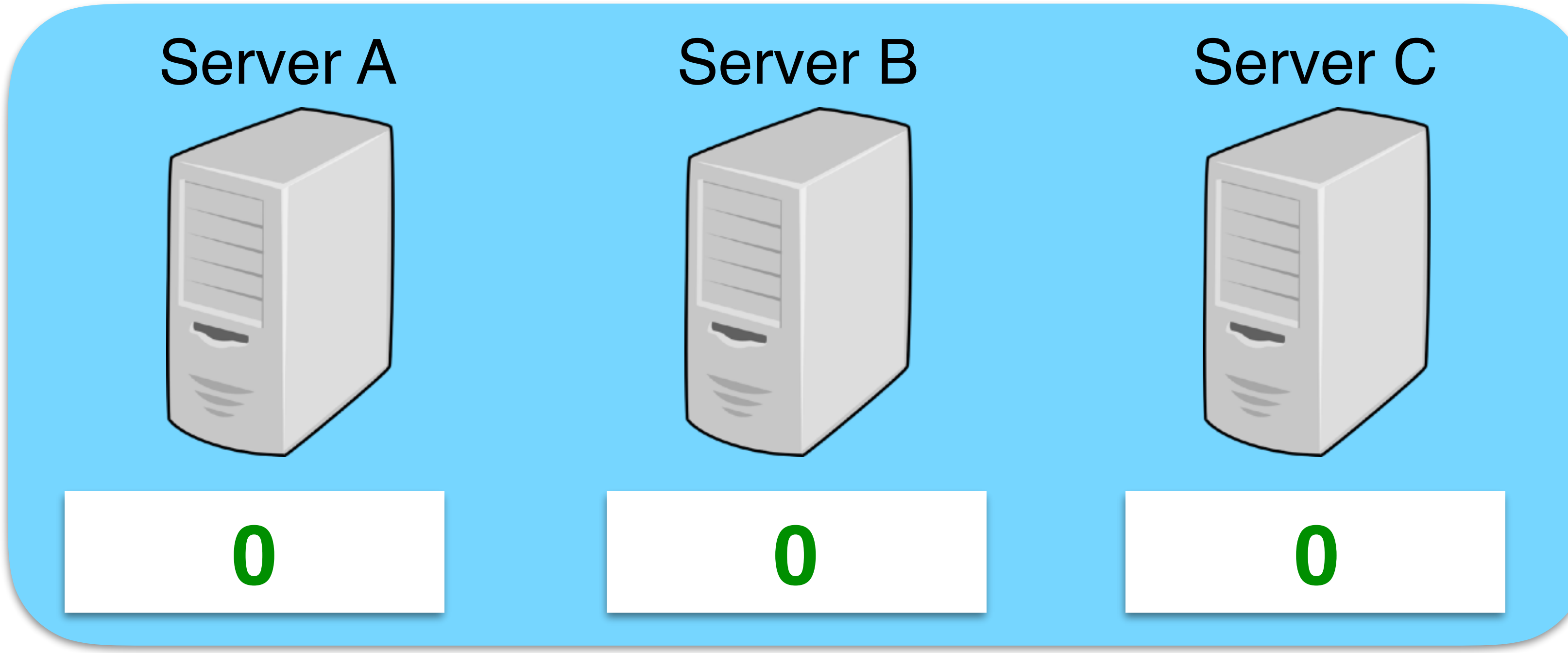
$x = 1$

More generally, servers

- hold shares of the client's private value x
- hold an <u>arbitrary</u> public predicate Valid(·)
  – expressed as an arithmetic circuit
- want to test if "Valid(x

For our running example:
Valid(x) = "$x \in \{0,1\}$"

Contribution 1
Secret-shared non-interactive proofs (SNIPs)



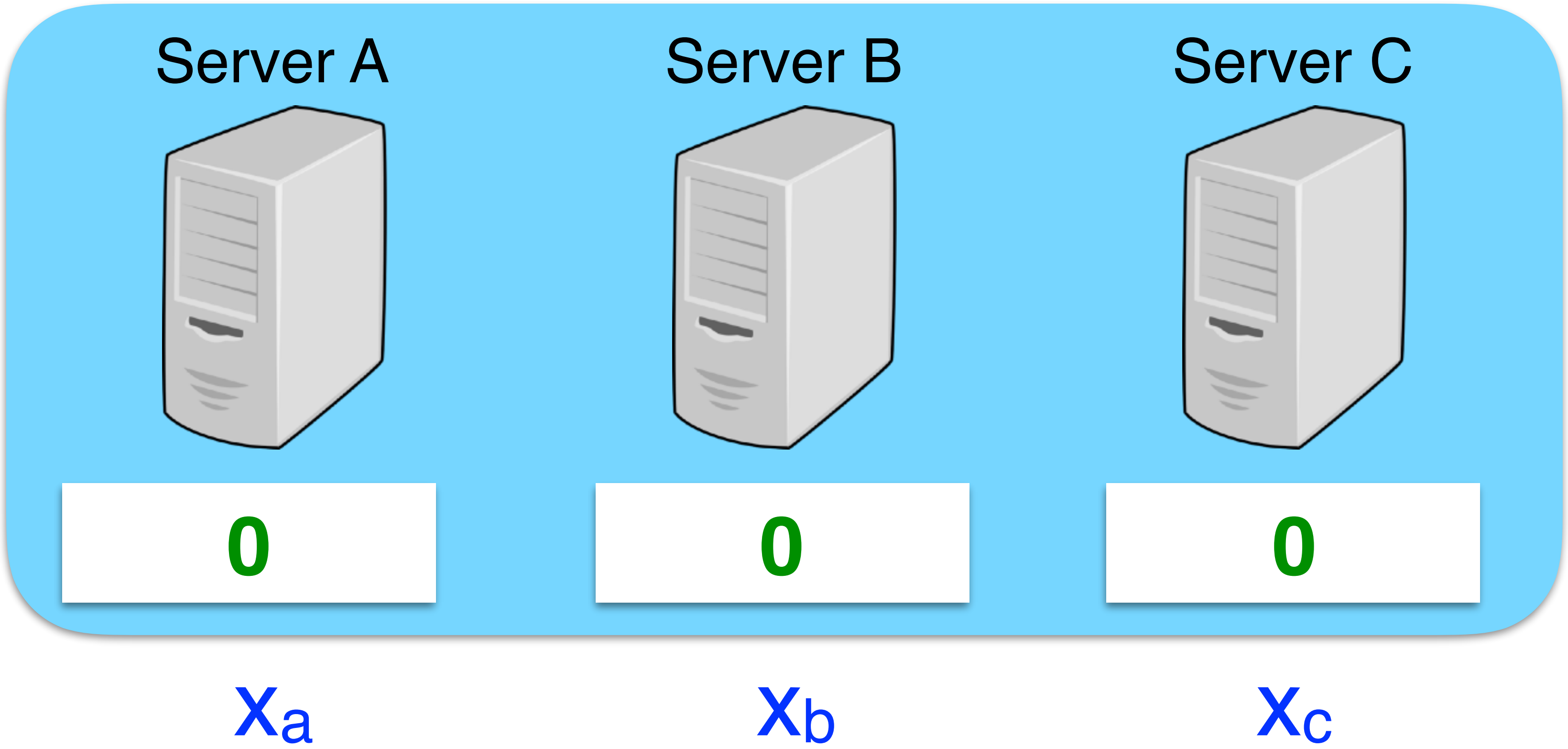
x_a x_b x_c

$x = 1$

More generally, servers

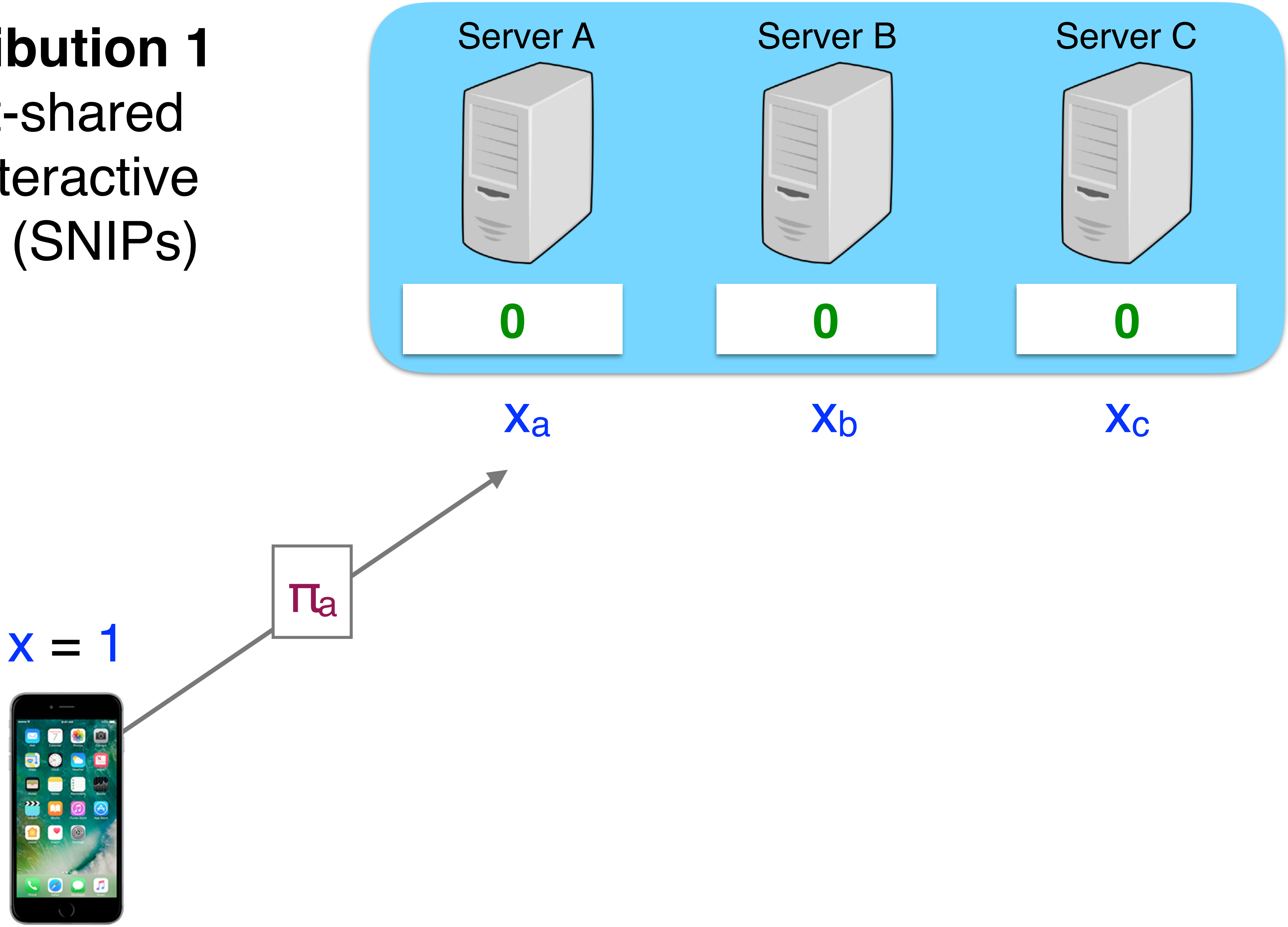
- hold shares of the client's private value x
- hold an <u>arbitrary</u> public predicate $\text{Valid}(\cdot)$
  – expressed as an arithmetic circuit
- want to test if "$\text{Valid}(x)$" holds, without leaking x

Contribution 1
Secret-shared non-interactive proofs (SNIPs)

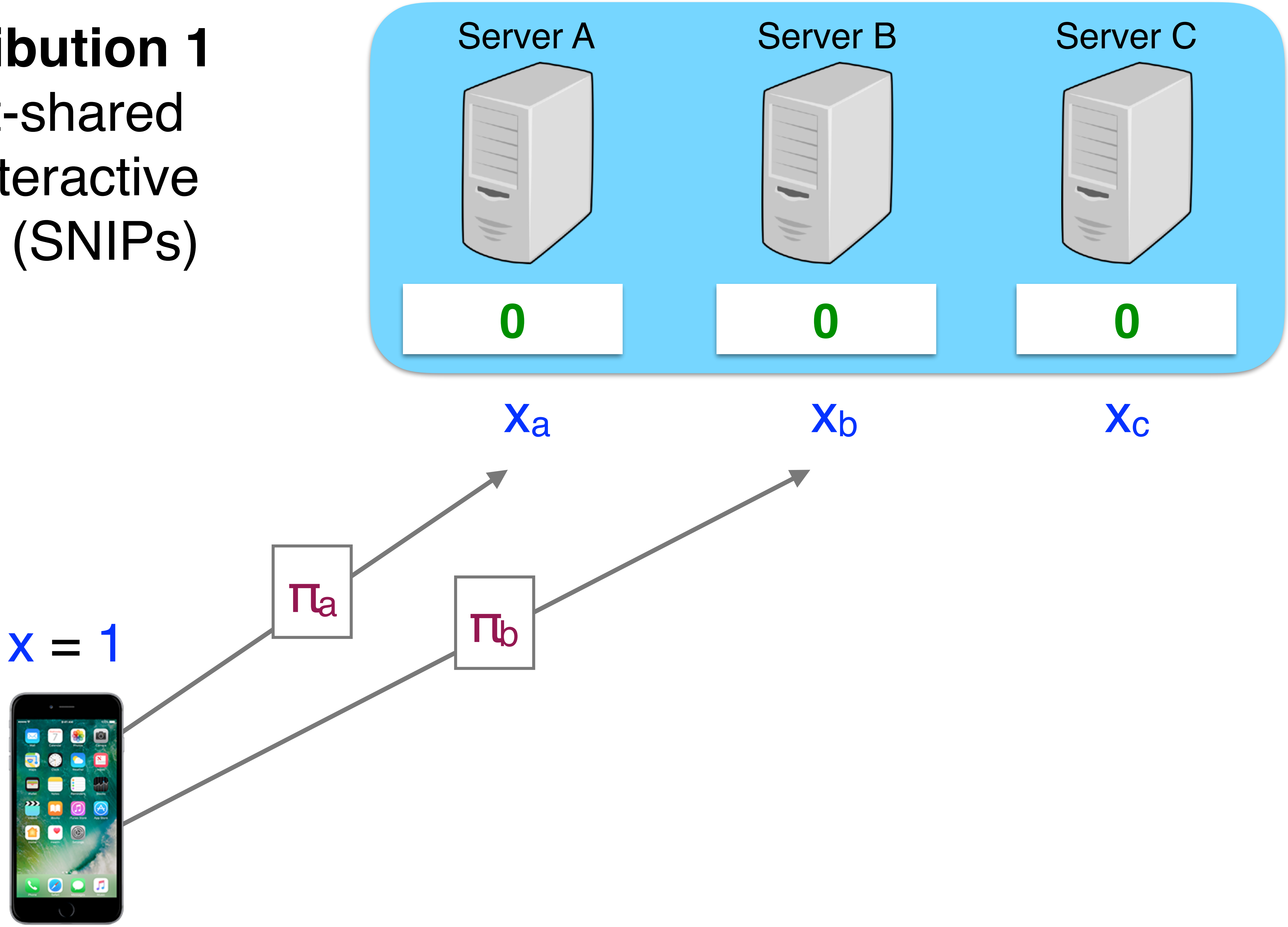


Contribution 1

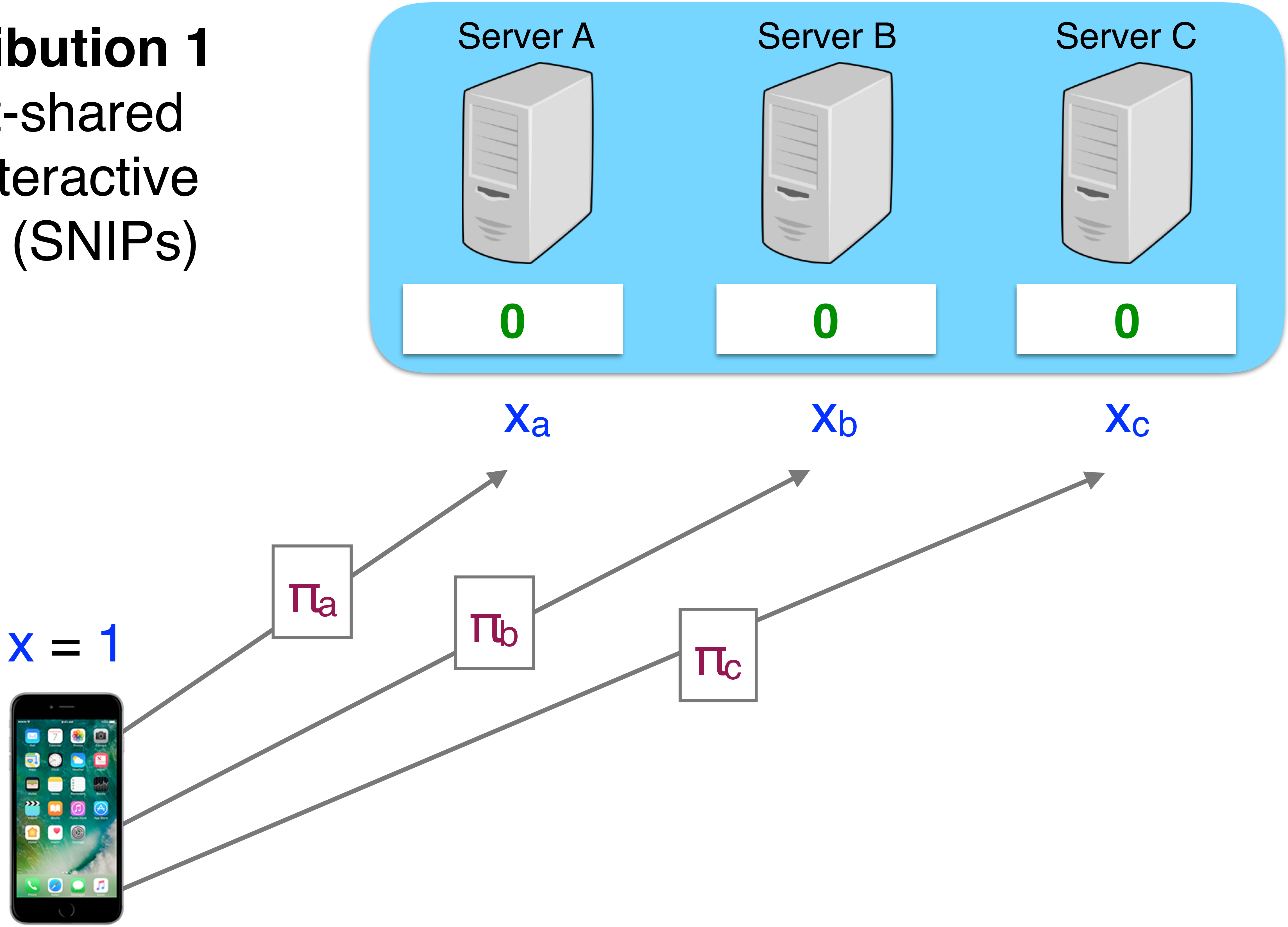
Secret-shared non-interactive proofs (SNIPs)



Contribution 1
Secret-shared non-interactive proofs (SNIPs)

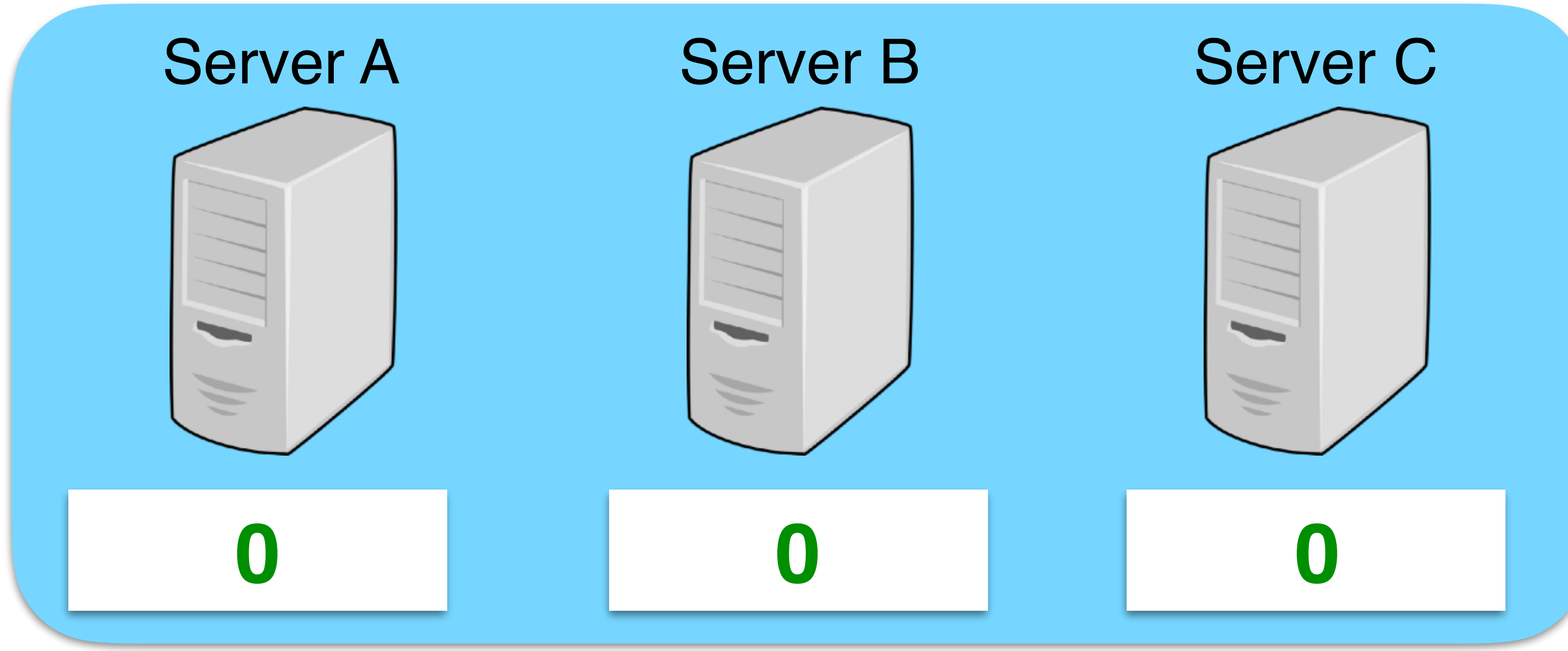


Contribution 1
Secret-shared non-interactive proofs (SNIPs)



Contribution 1

Secret-shared non-interactive proofs (SNIPs)

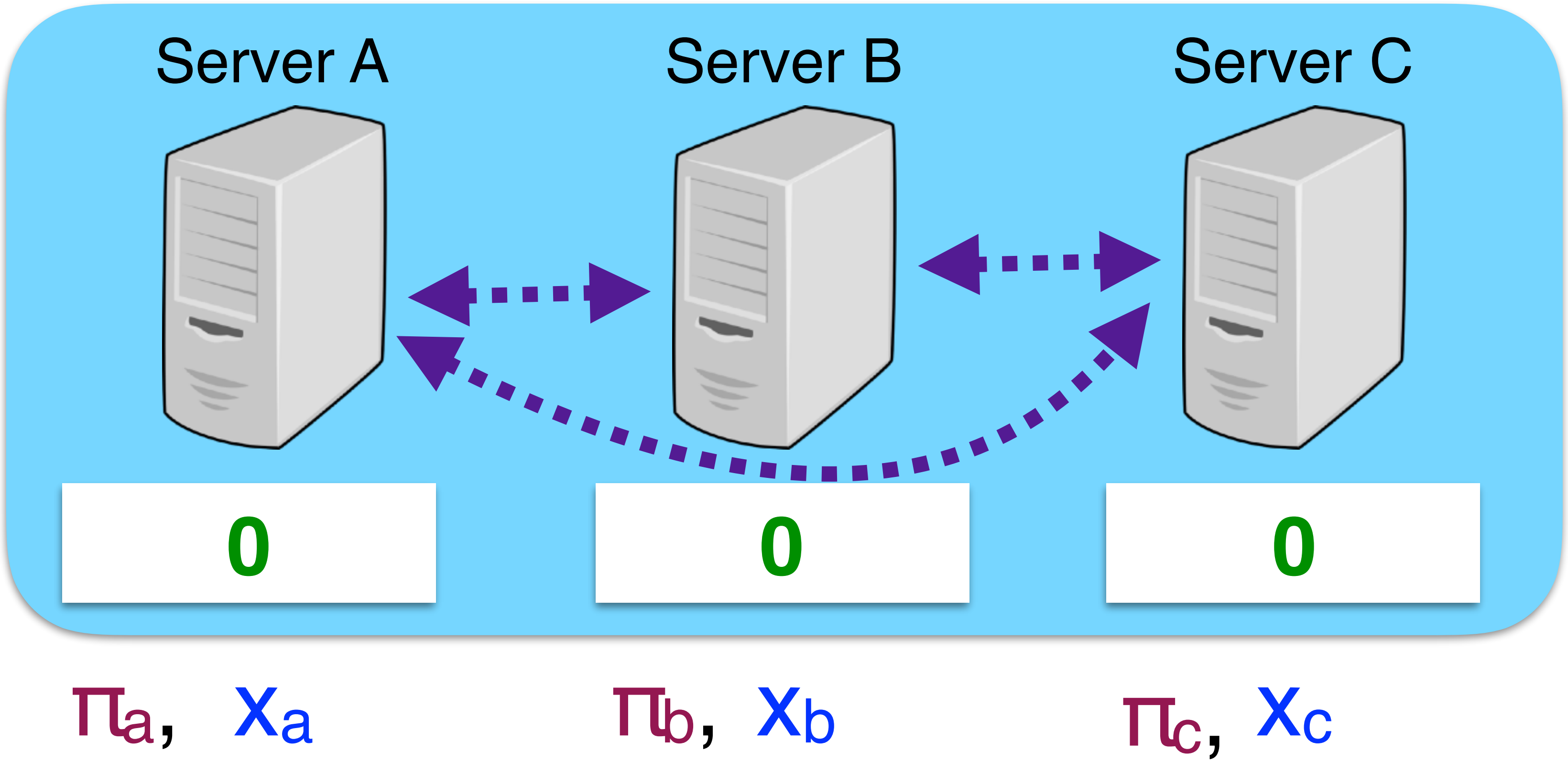


π_a, x_a π_b, x_b π_c, x_c

$x = 1$

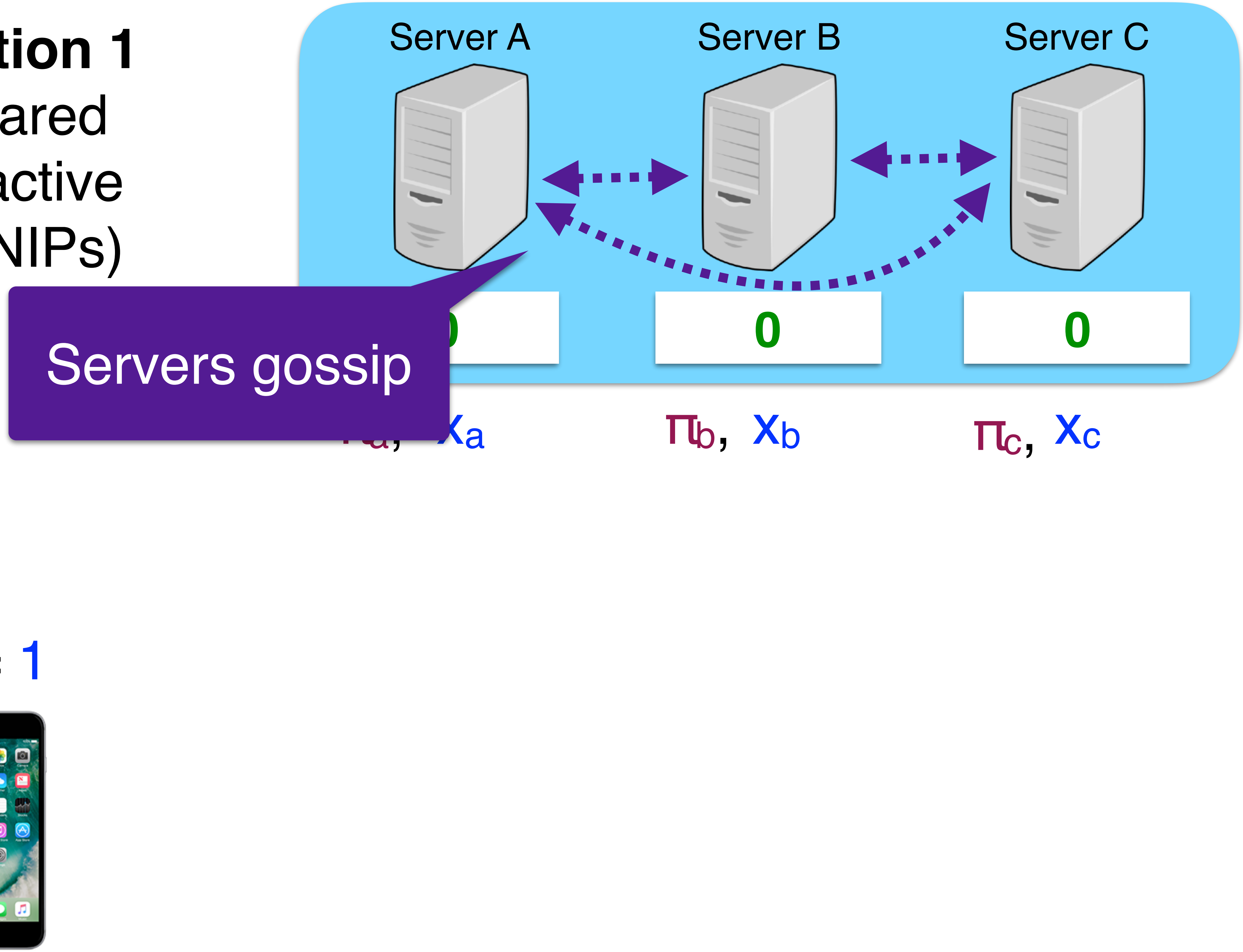
Contribution 1

Secret-shared non-interactive proofs (SNIPs)



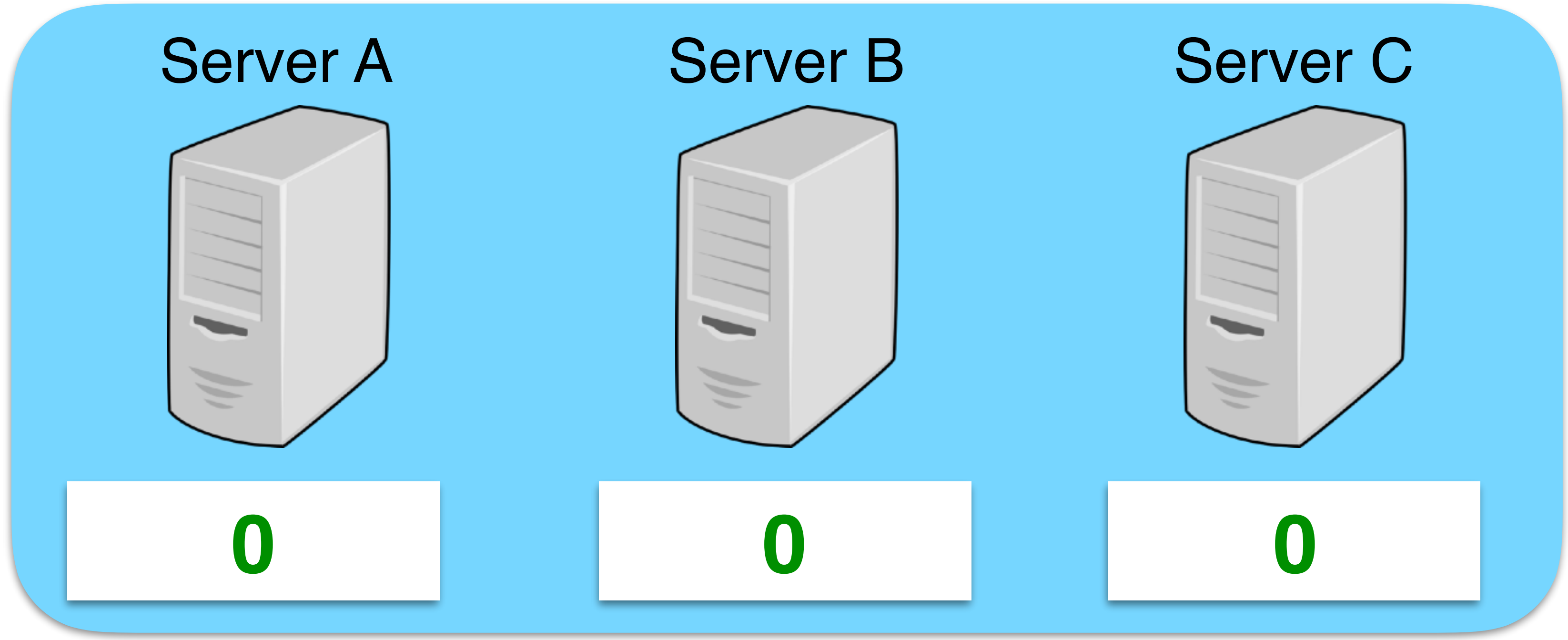
$x = 1$

Contribution 1
Secret-shared non-interactive proofs (SNIPs)



Contribution 1

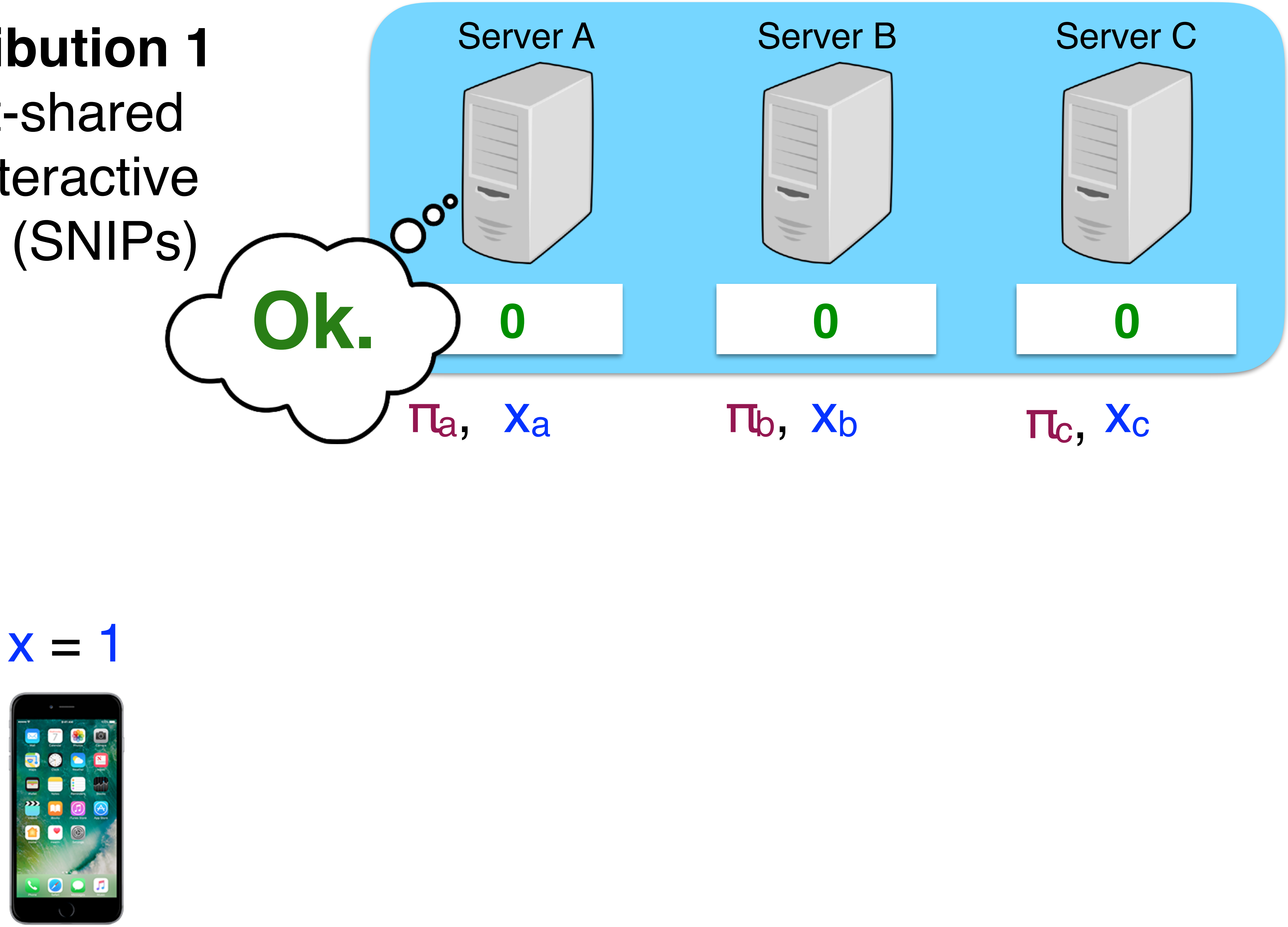
Secret-shared non-interactive proofs (SNIPs)



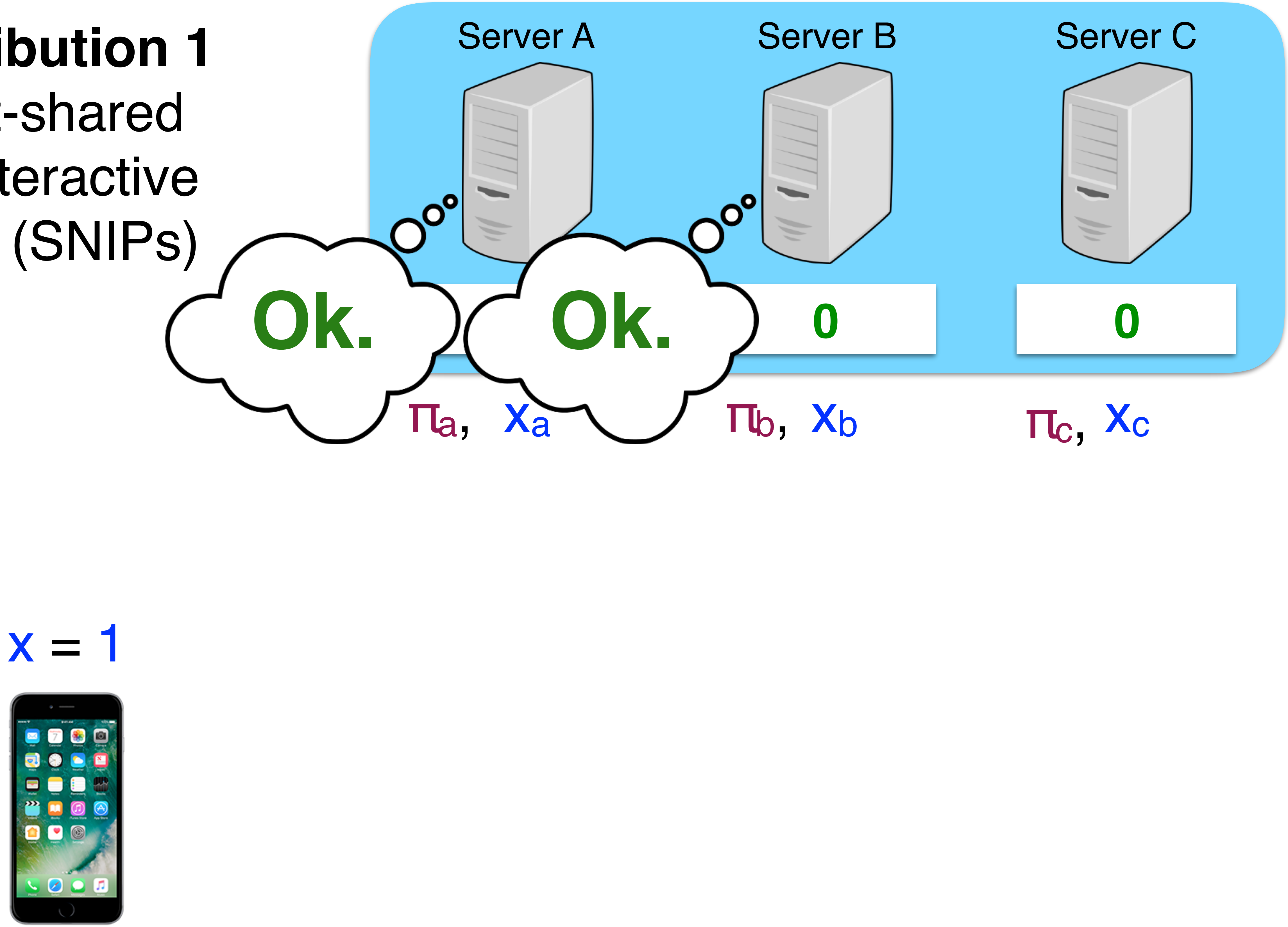
π_a, x_a π_b, x_b π_c, x_c

$x = 1$

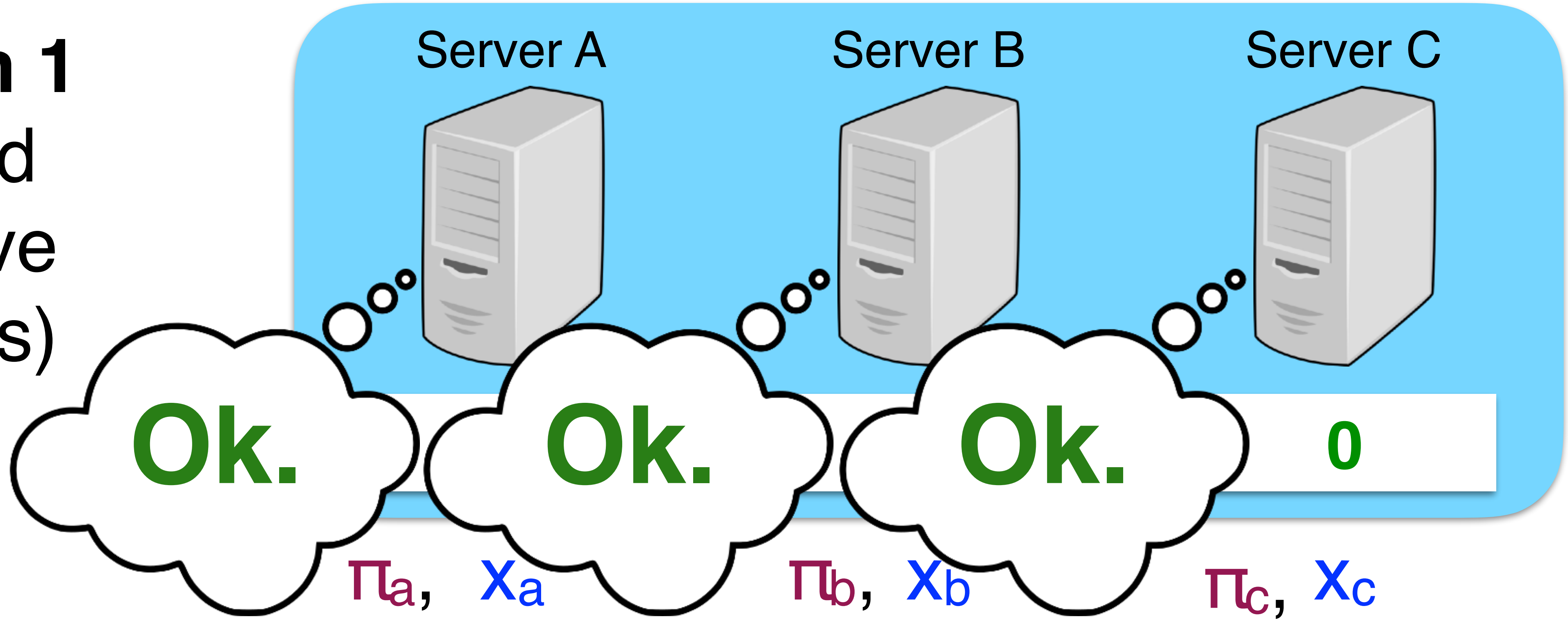
Contribution 1
Secret-shared non-interactive proofs (SNIPs)



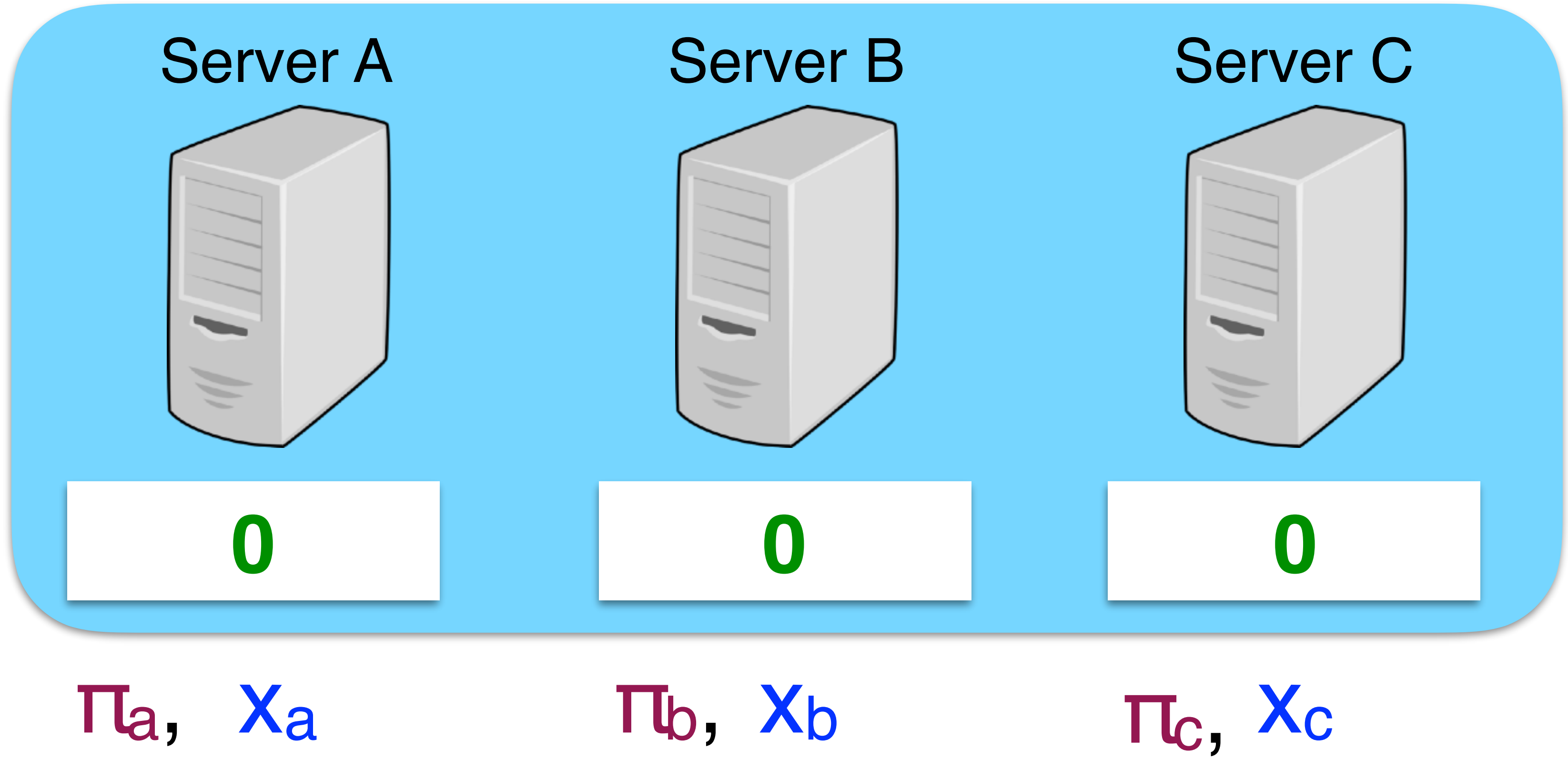
Contribution 1
Secret-shared non-interactive proofs (SNIPs)



Contribution 1
Secret-shared non-interactive proofs (SNIPs)

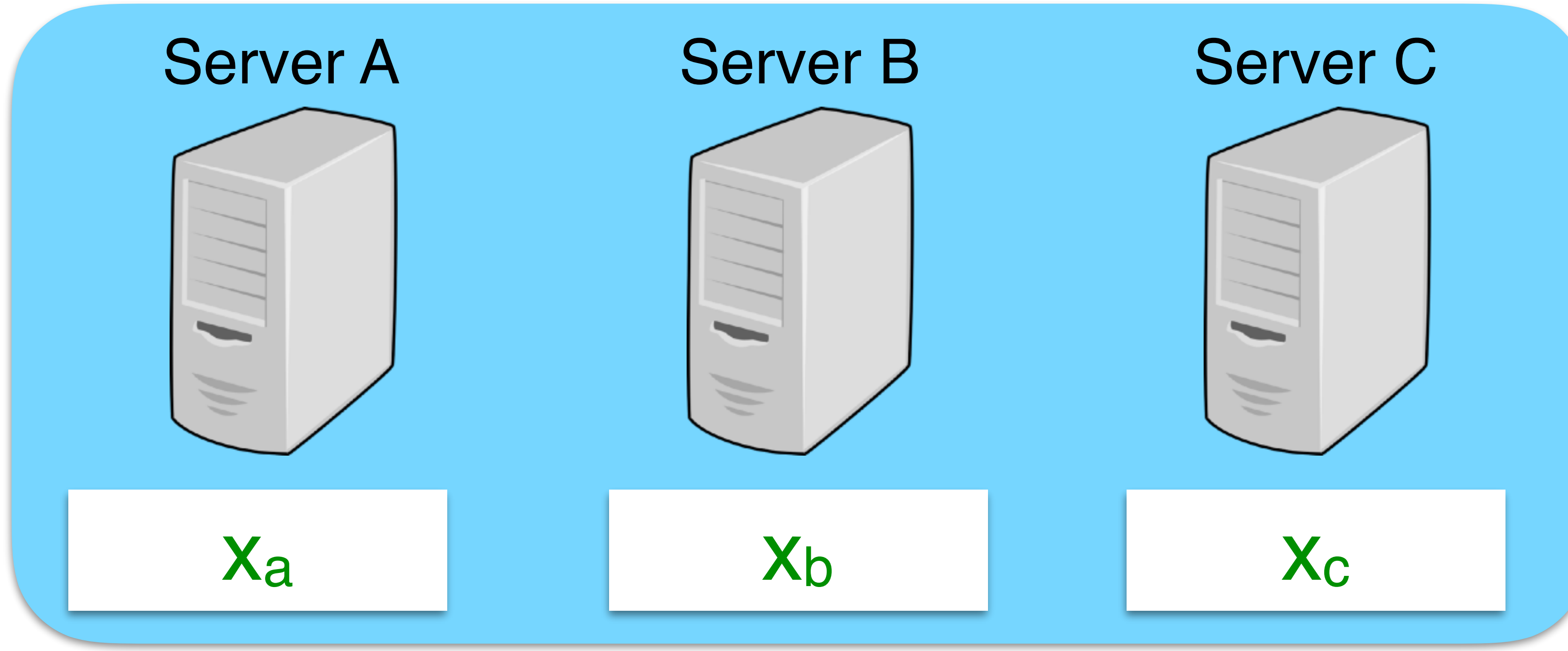


Contribution 1
Secret-shared non-interactive proofs (SNIPs)



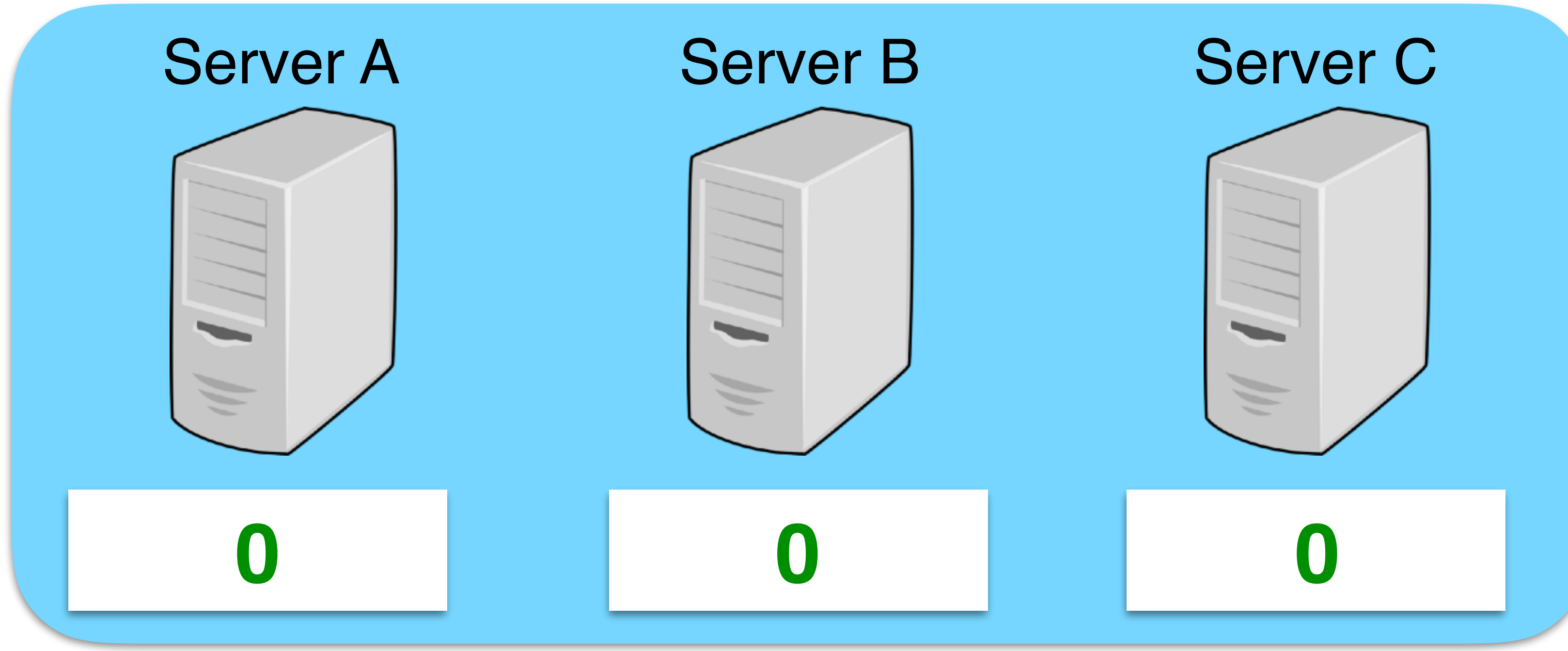
$x = 1$

Contribution 1
Secret-shared non-interactive proofs (SNIPs)



$x = 1$

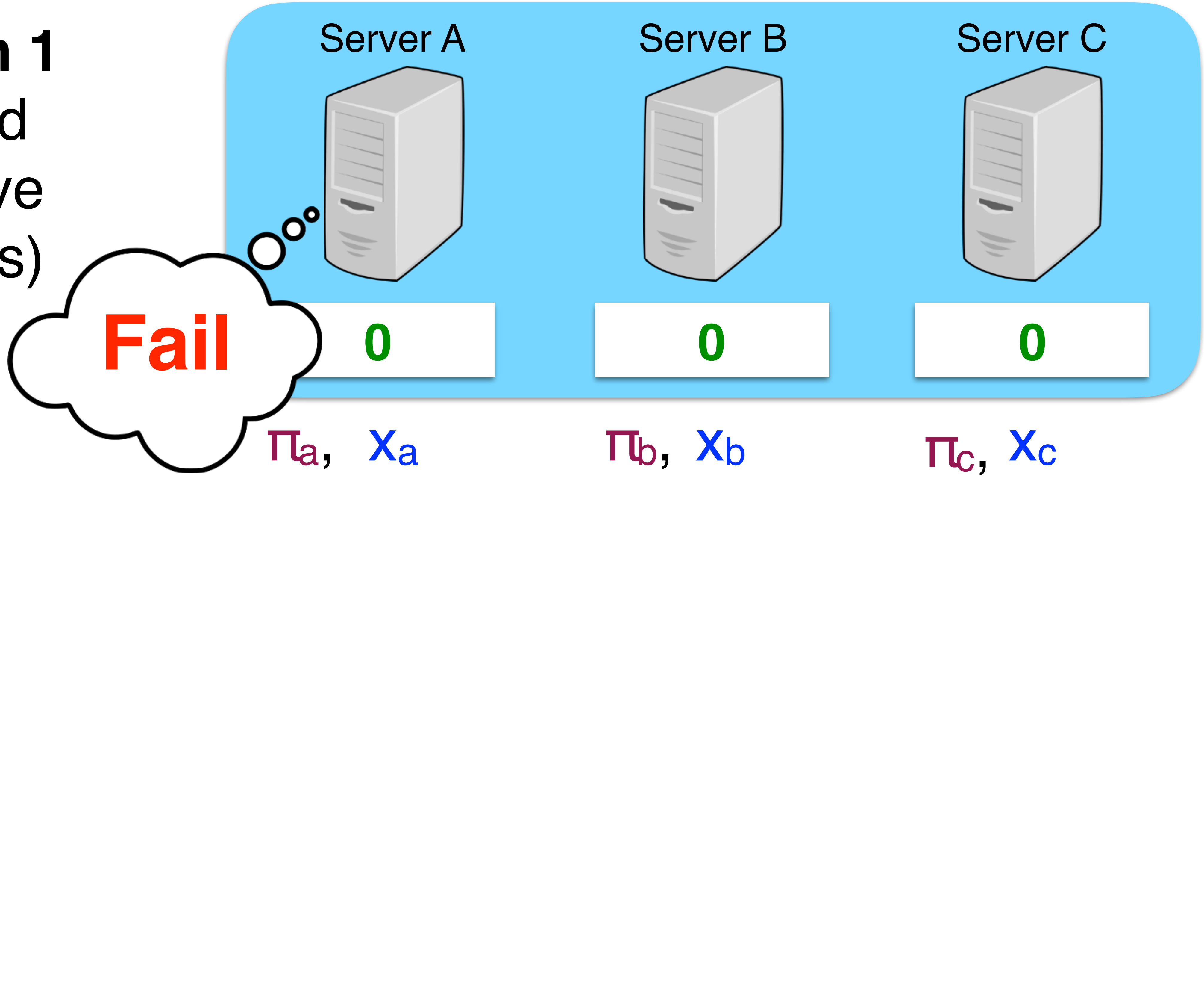
Contribution 1
Secret-shared non-interactive proofs (SNIPs)



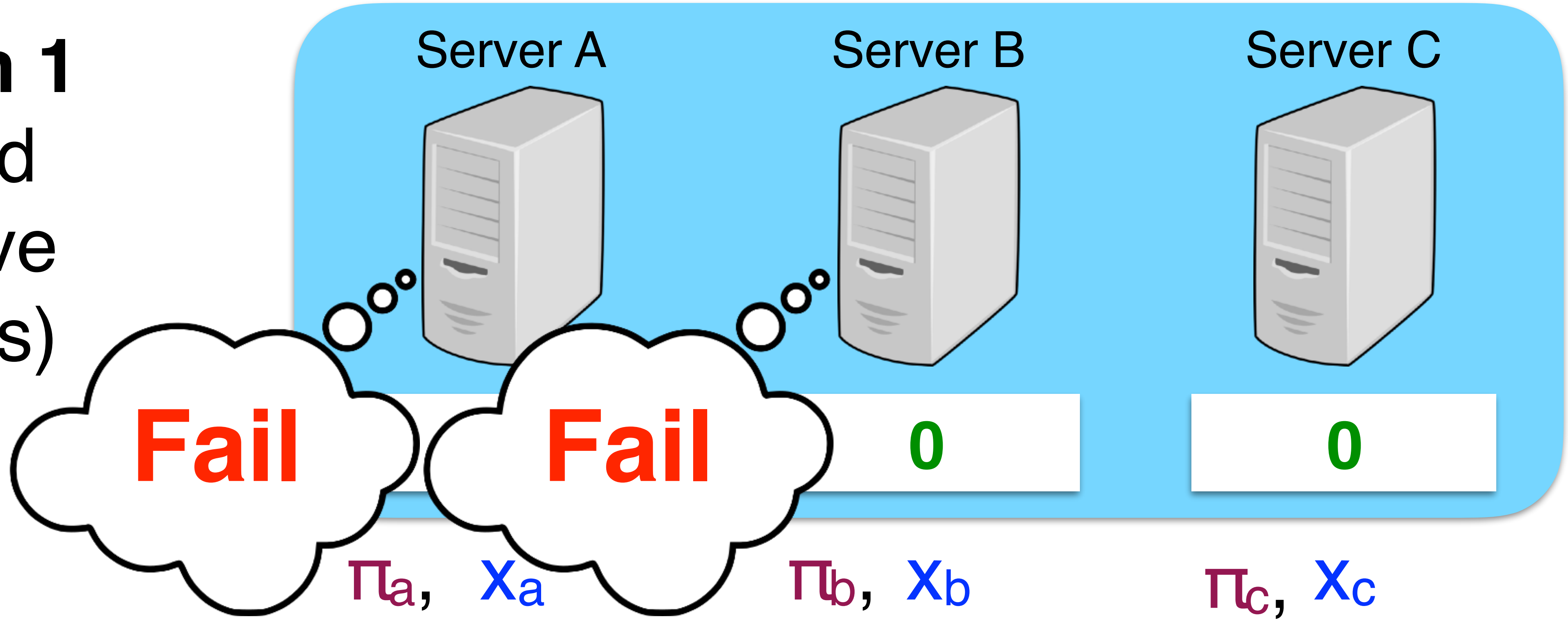
π_a, x_a π_b, x_b π_c, x_c

$x = 1$

Contribution 1
Secret-shared non-interactive proofs (SNIPs)

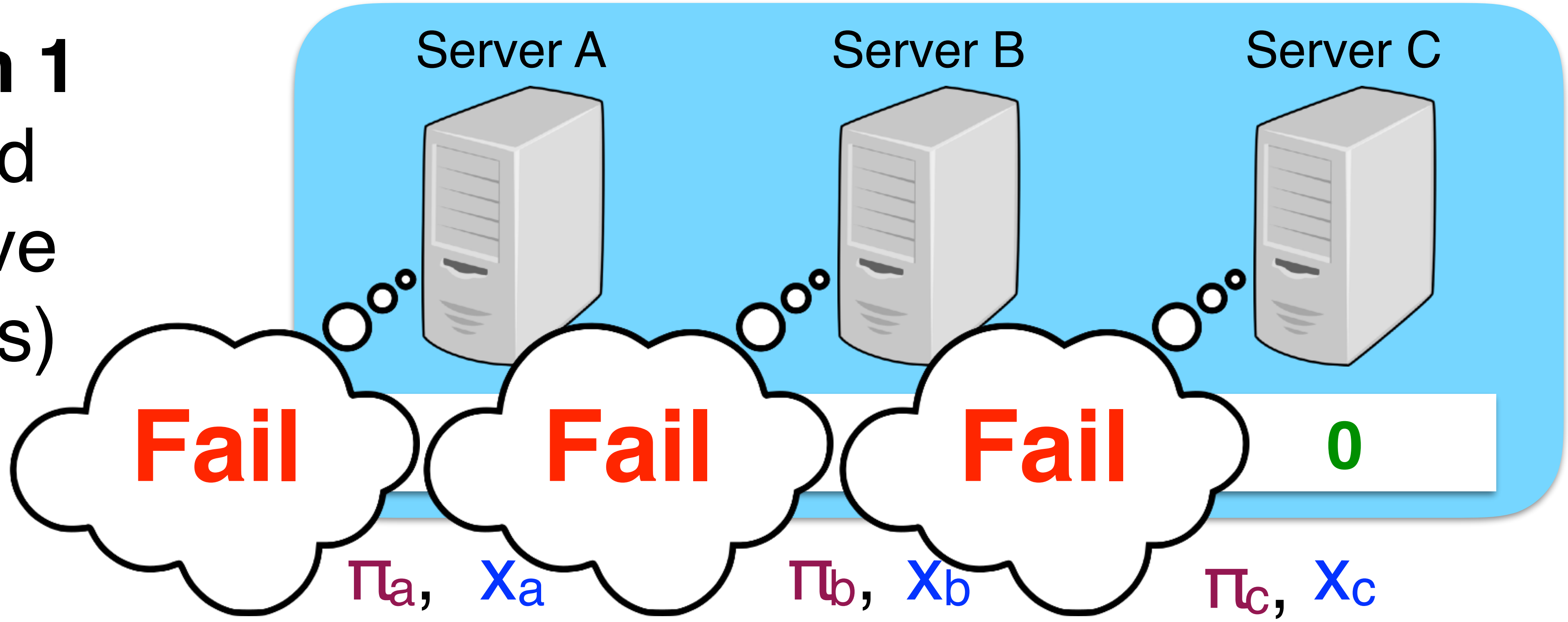


Contribution 1
Secret-shared non-interactive proofs (SNIPs)



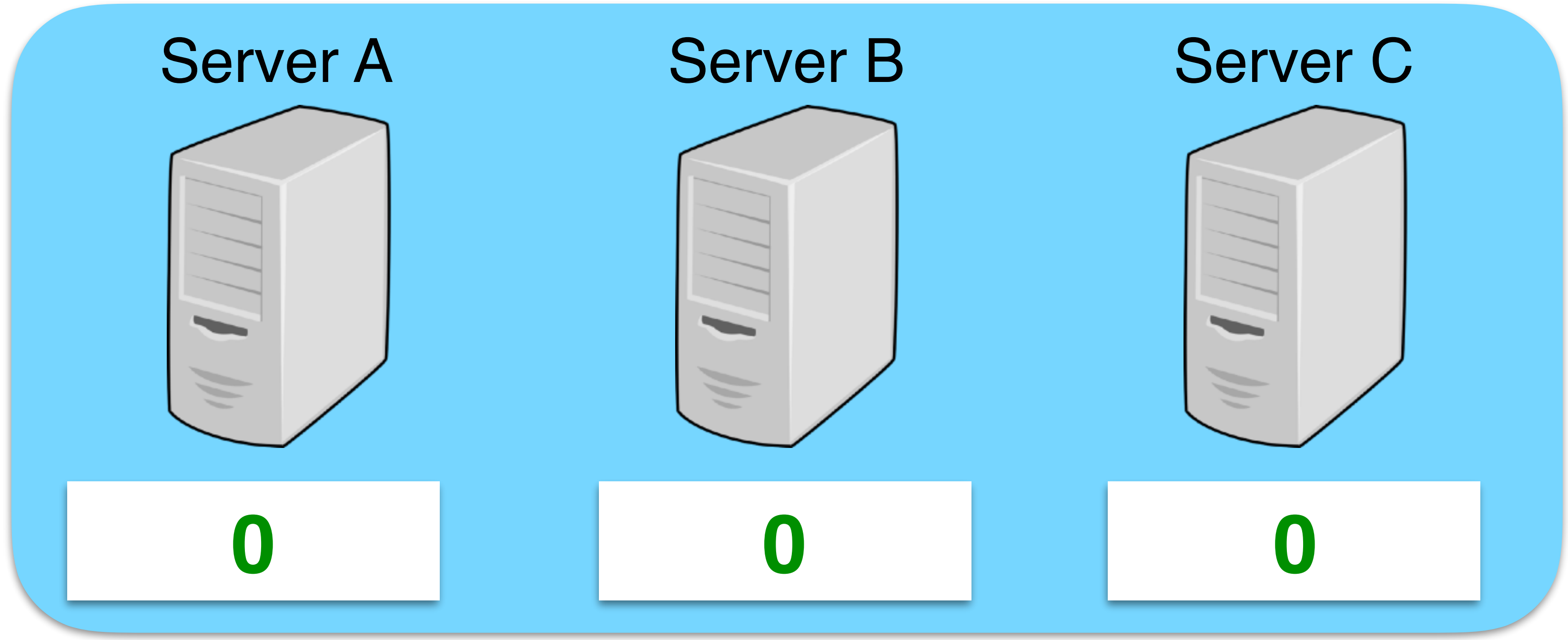
$x = 1$

Contribution 1
Secret-shared non-interactive proofs (SNIPs)



$x = 1$

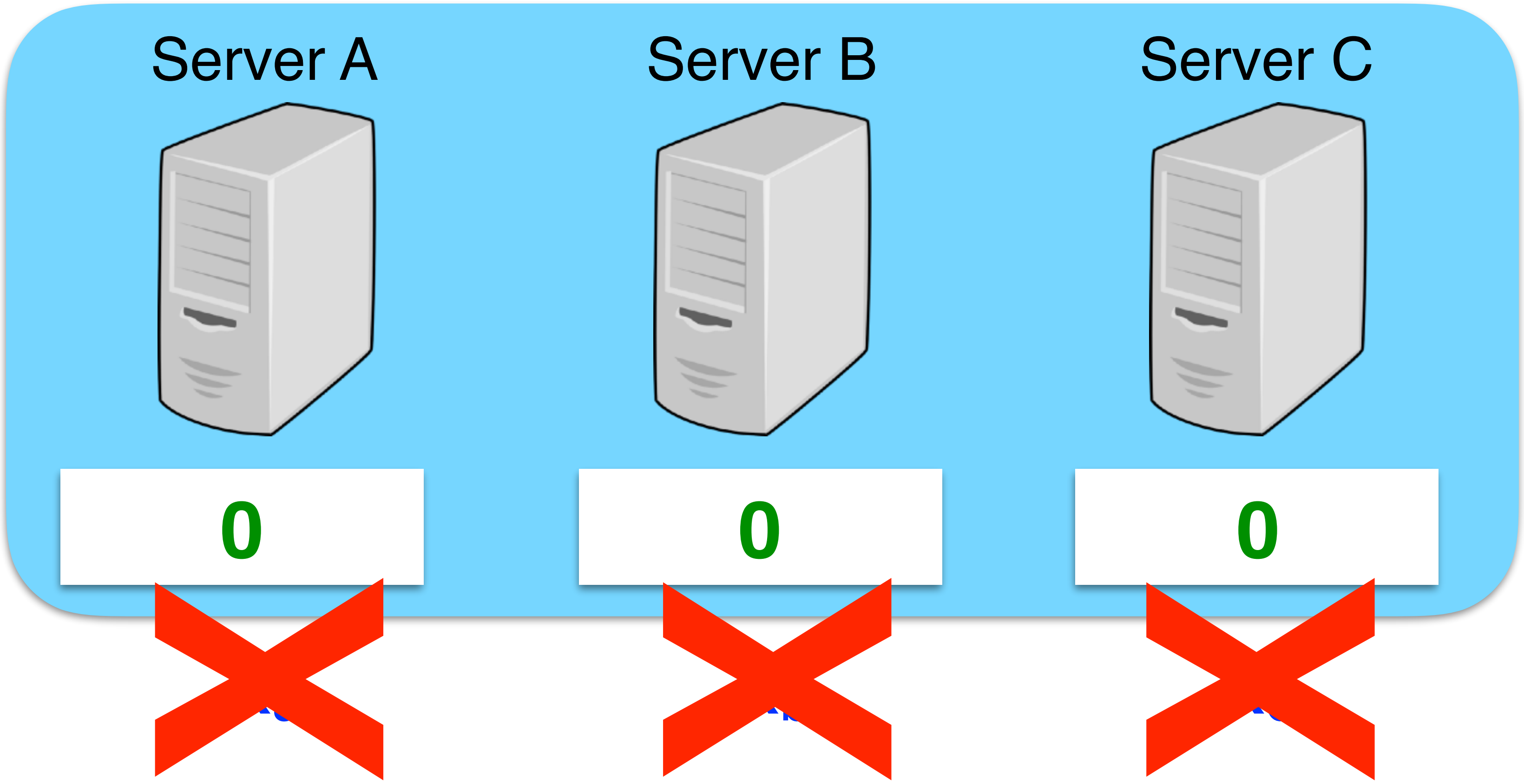
Contribution 1
Secret-shared non-interactive proofs (SNIPs)



π_a, x_a π_b, x_b π_c, x_c

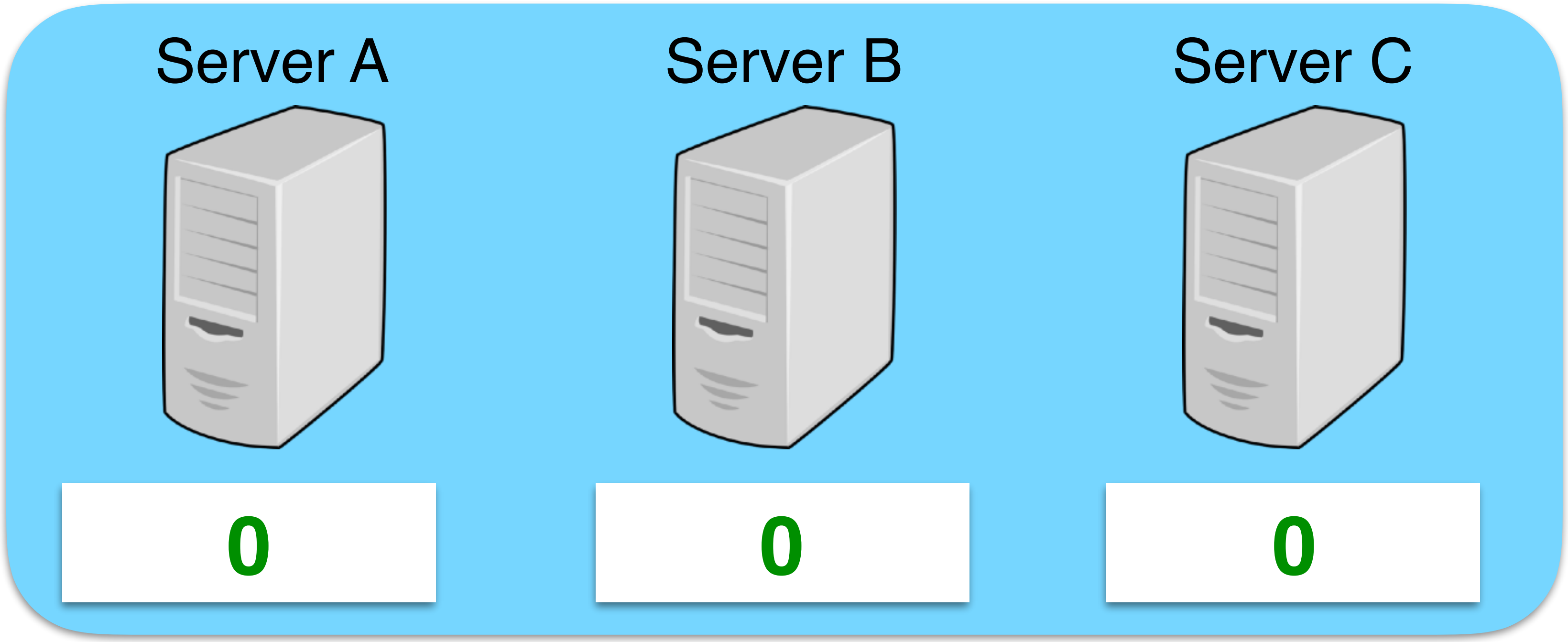
$x = 1$

Contribution 1
Secret-shared non-interactive proofs (SNIPs)



- Prio servers *detect and reject malformed client submissions*
- In this example, each client can influence the aggregate statistic by +/- 1, at most

How SNIPs work

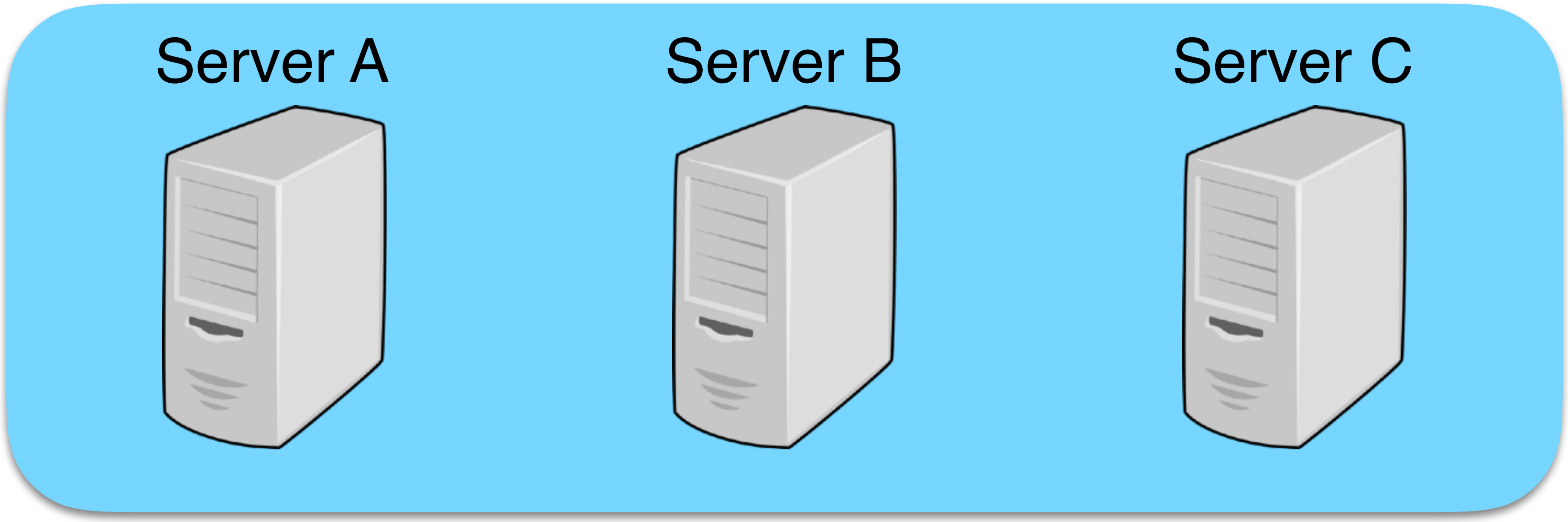


x_a x_b x_c

The servers want to ensure that
$\text{Valid}(x) = \text{Valid}(x_a + x_b + x_c) = 1$
...without learning x.

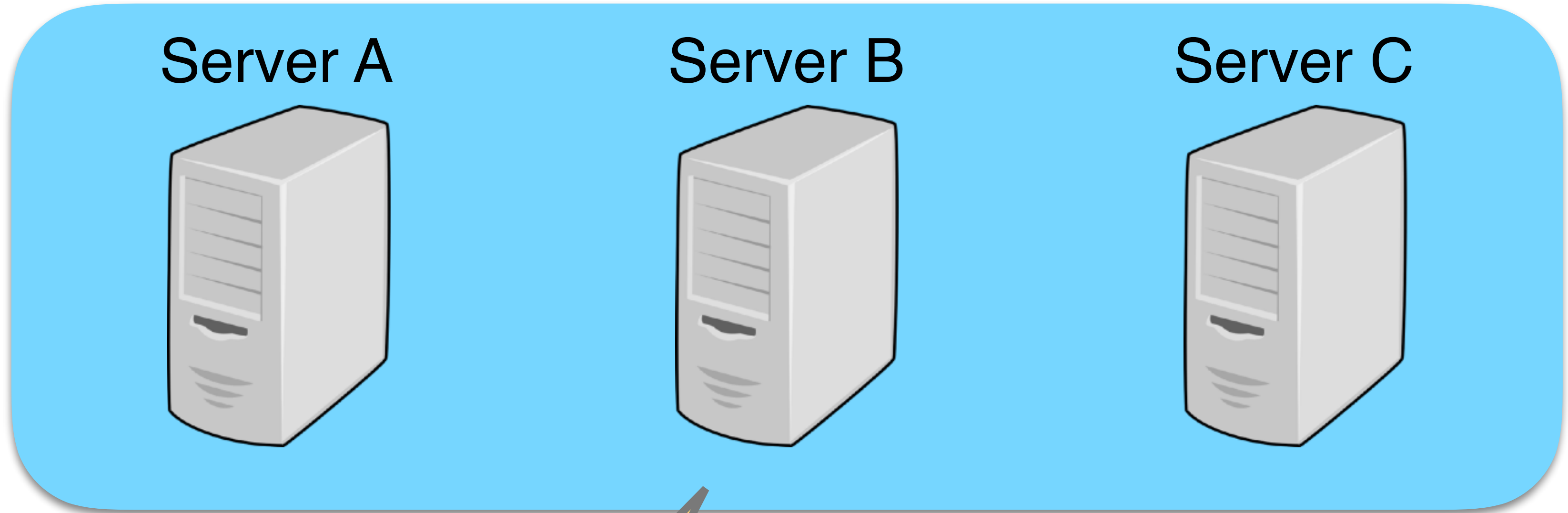
$x = 1$

How SNIPs work



x_a x_b x_c

How SNIPs work

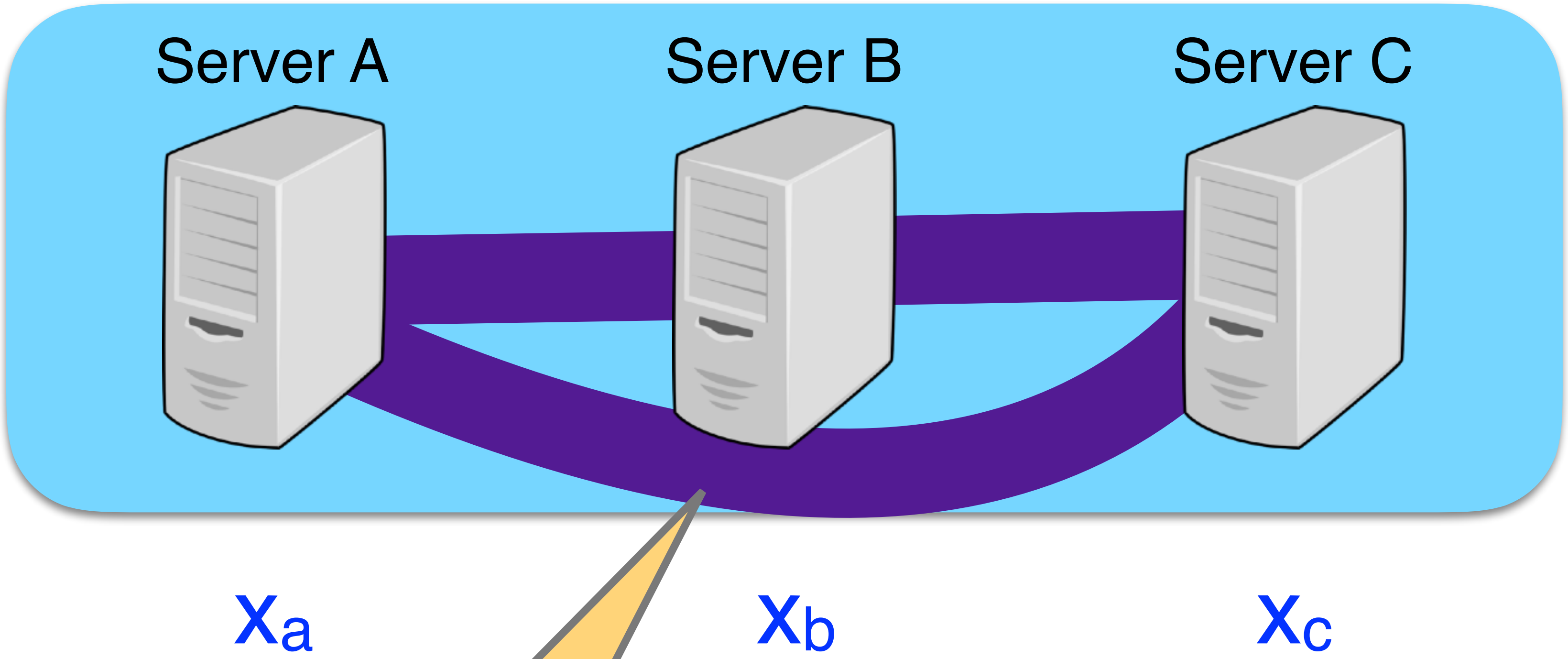


x_a x_b x_c

Could run **secure multiparty computation** to check that Valid(x) = 1.

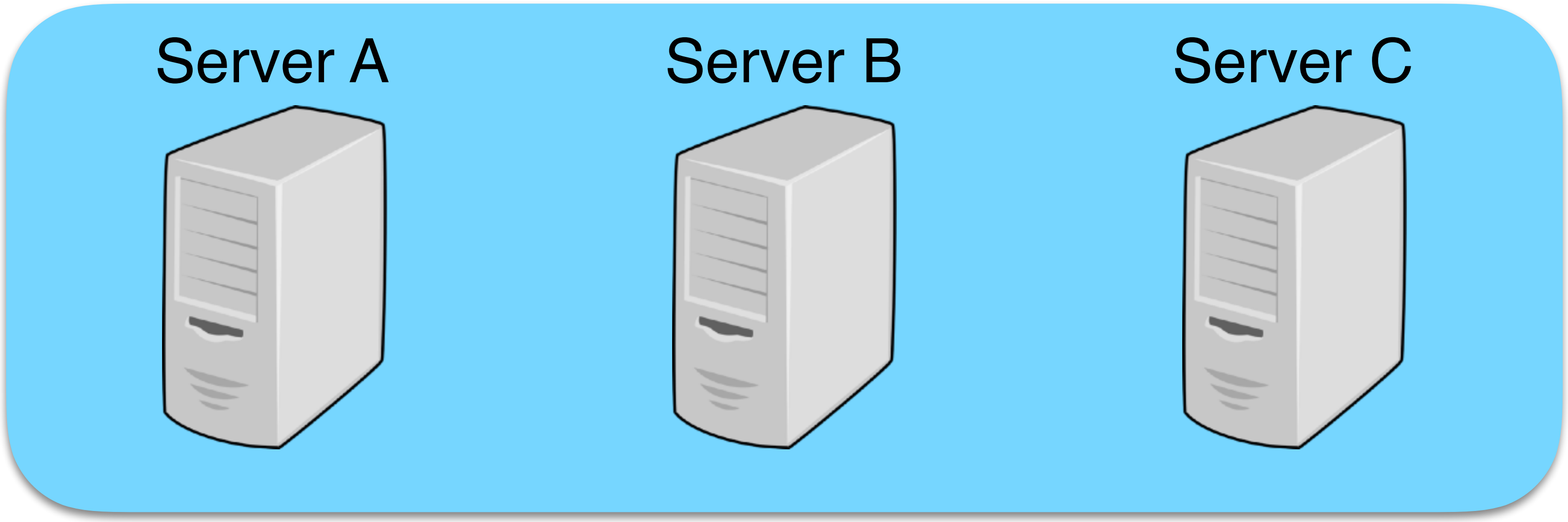
[GMW87], [BGW88]

How SNIPs work



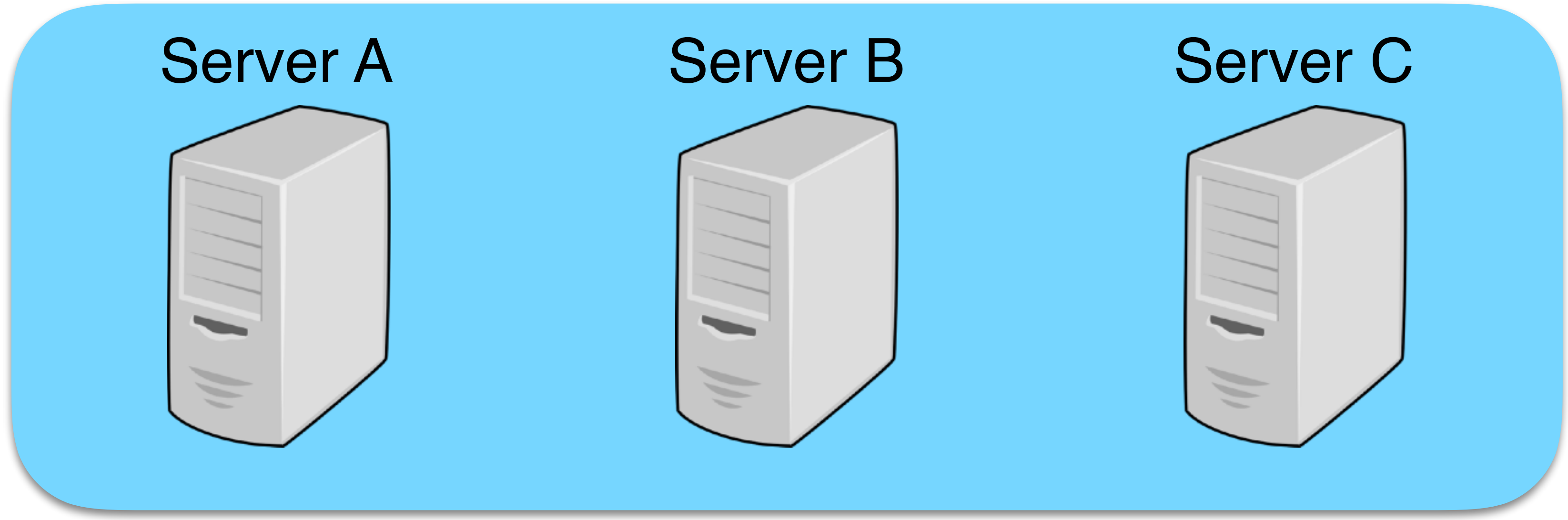
Could run **secure multiparty computation** to check that Valid(x) = 1.
[GMW87], [BGW88]

How SNIPs work



x_a x_b x_c

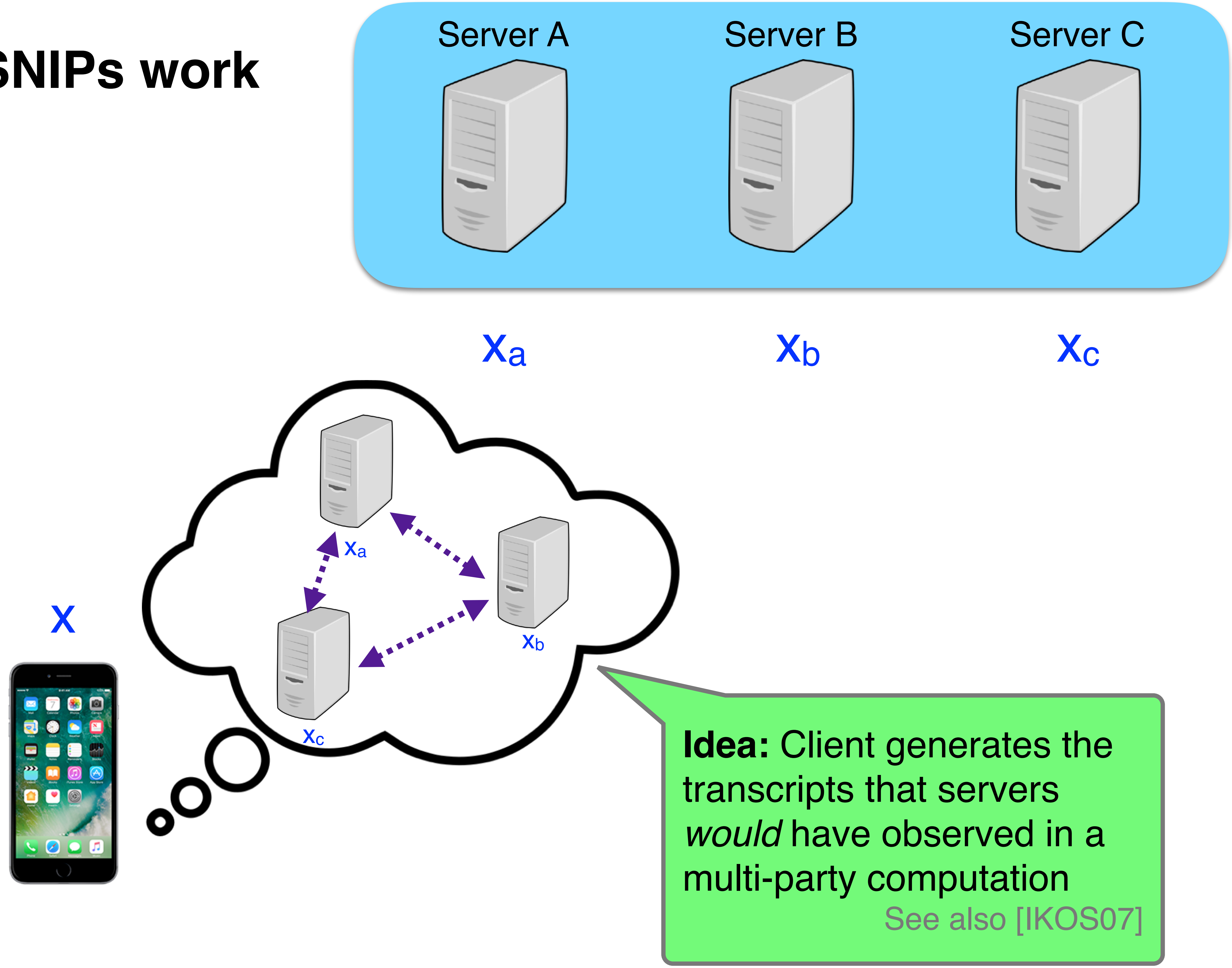
How SNIPs work



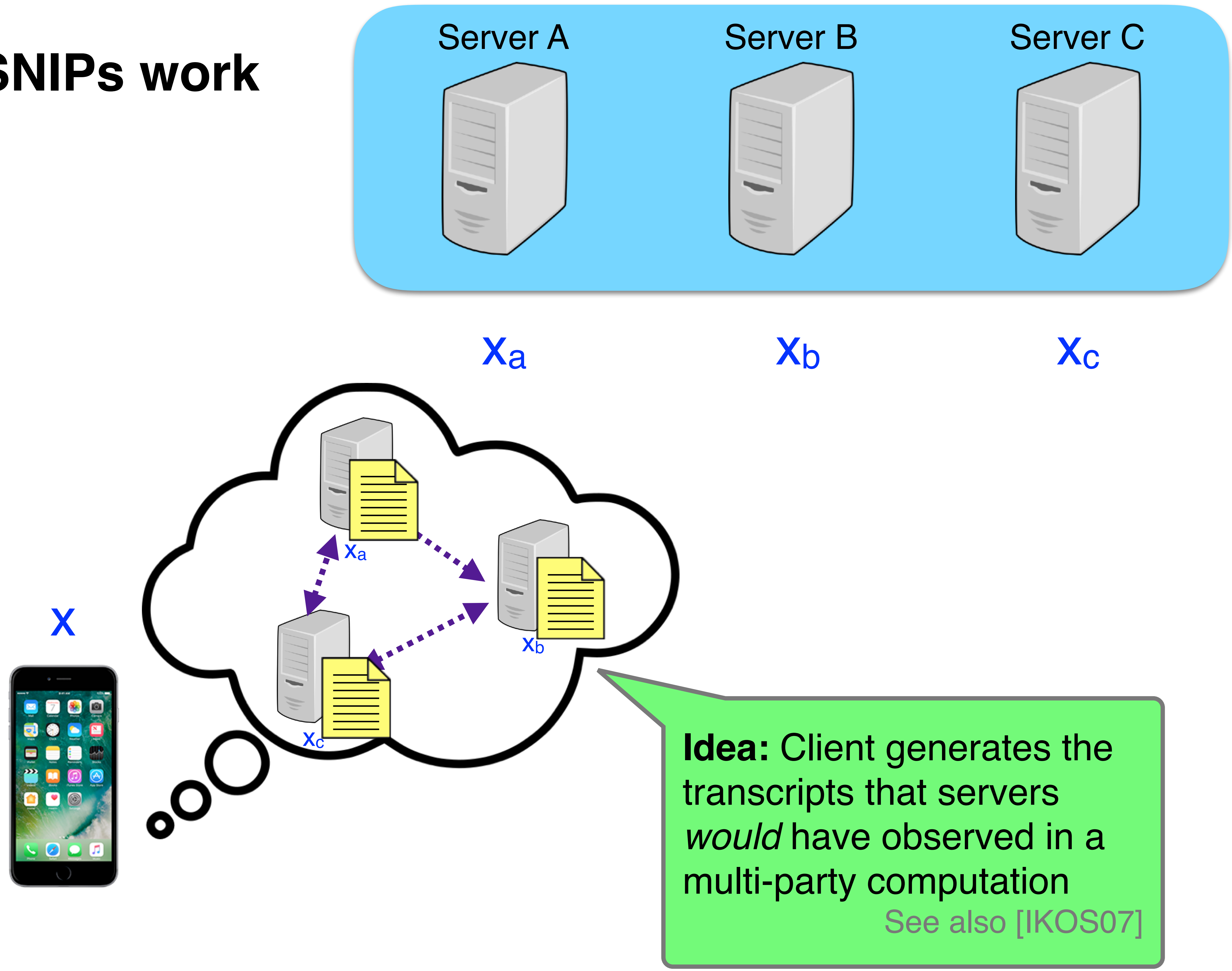
x_a x_b x_c

x

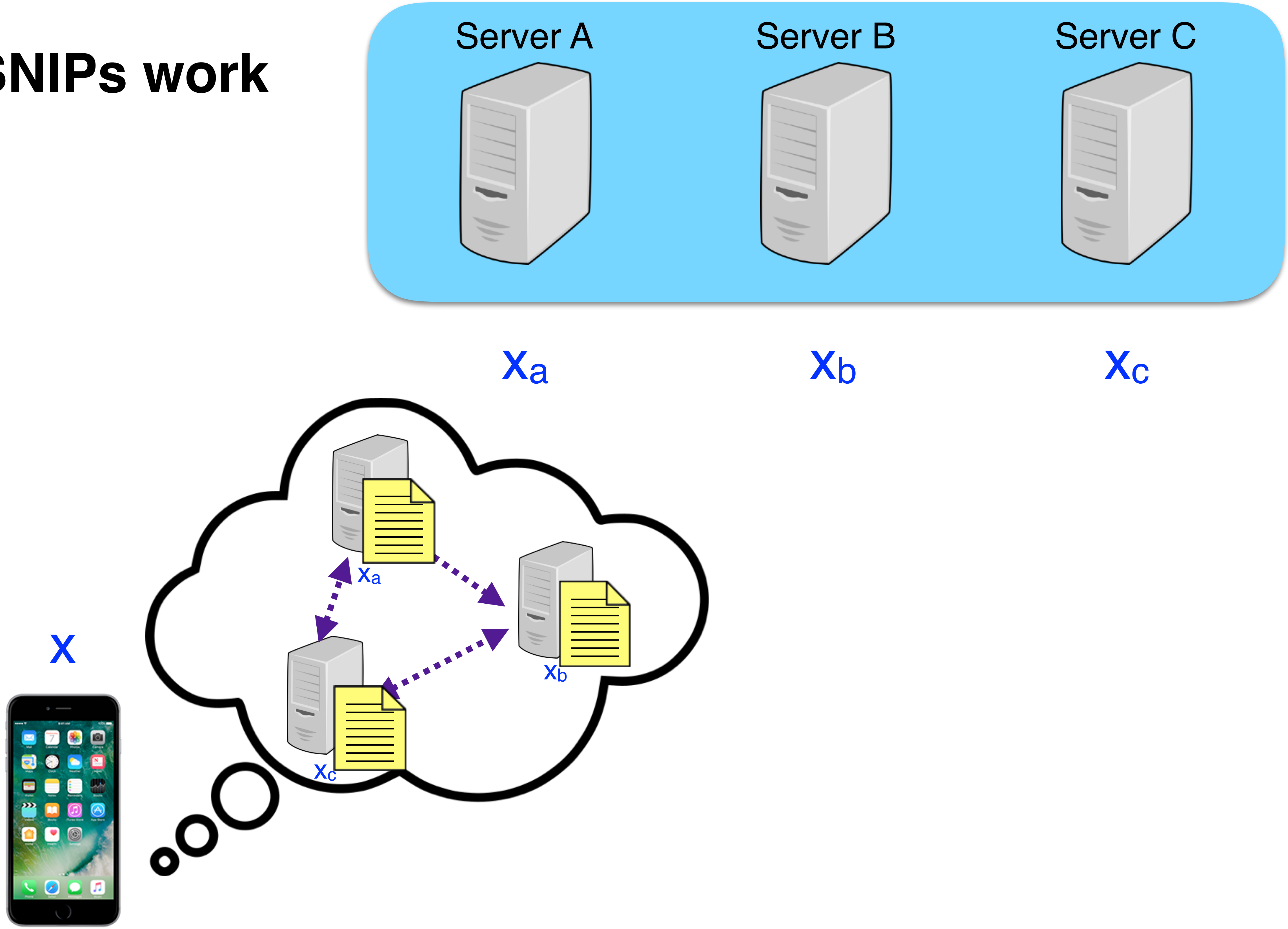
How SNIPs work



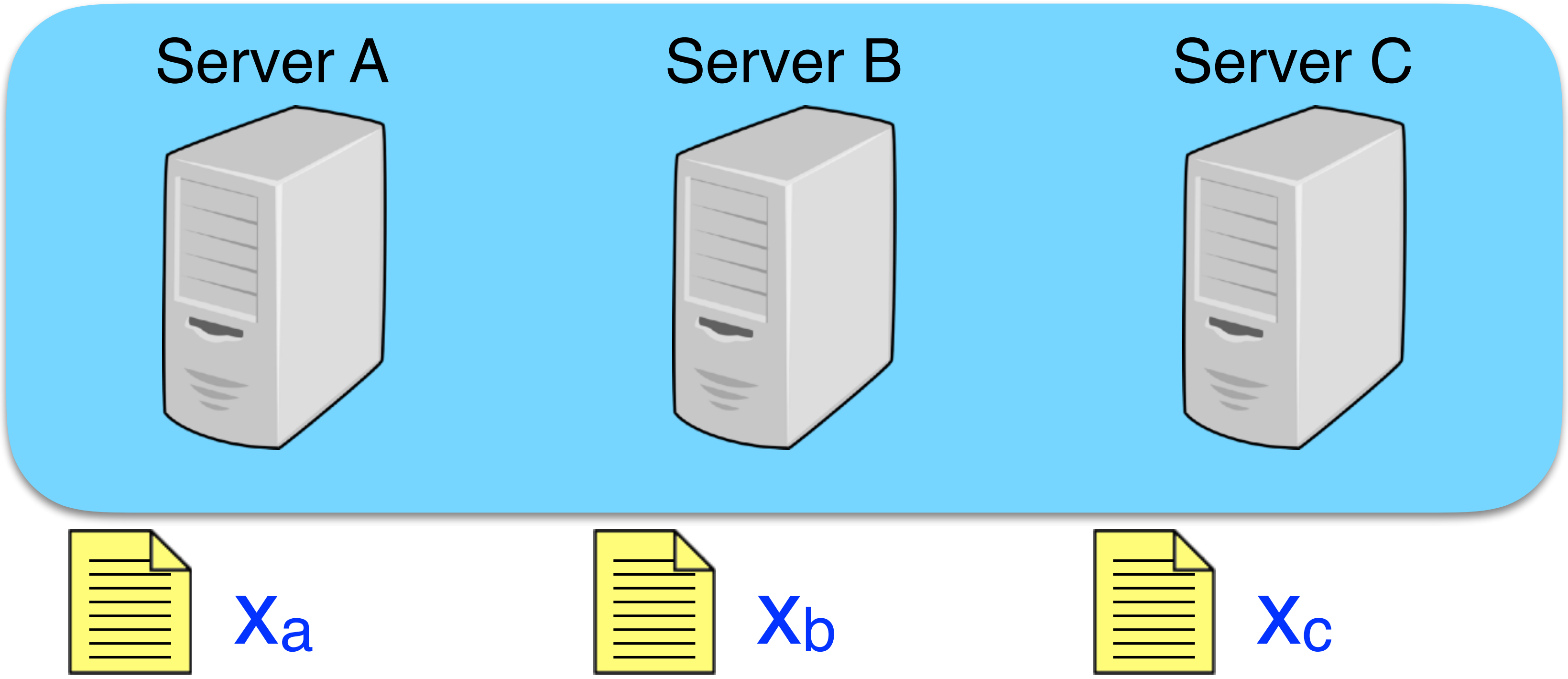
How SNIPs work



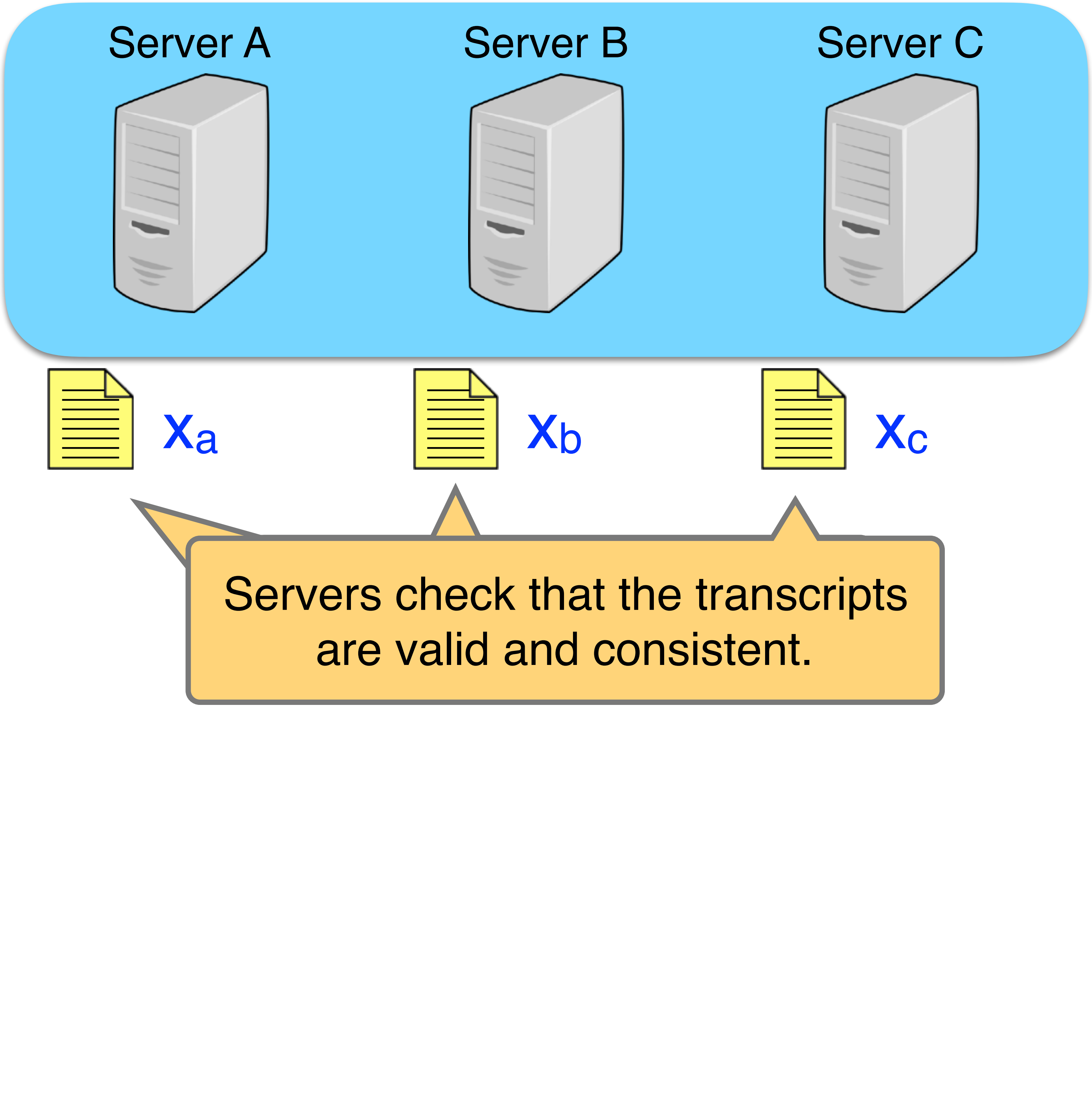
How SNIPs work



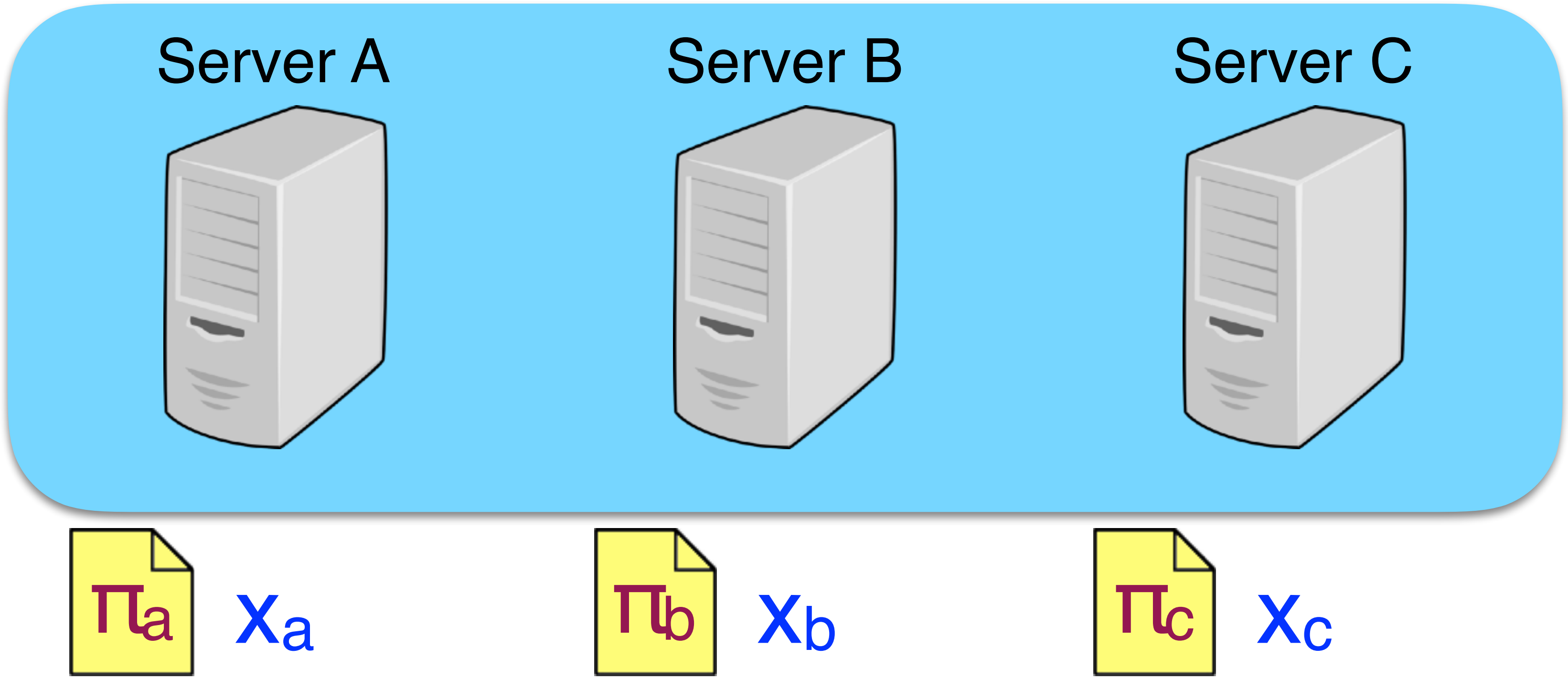
How SNIPs work



How SNIPs work

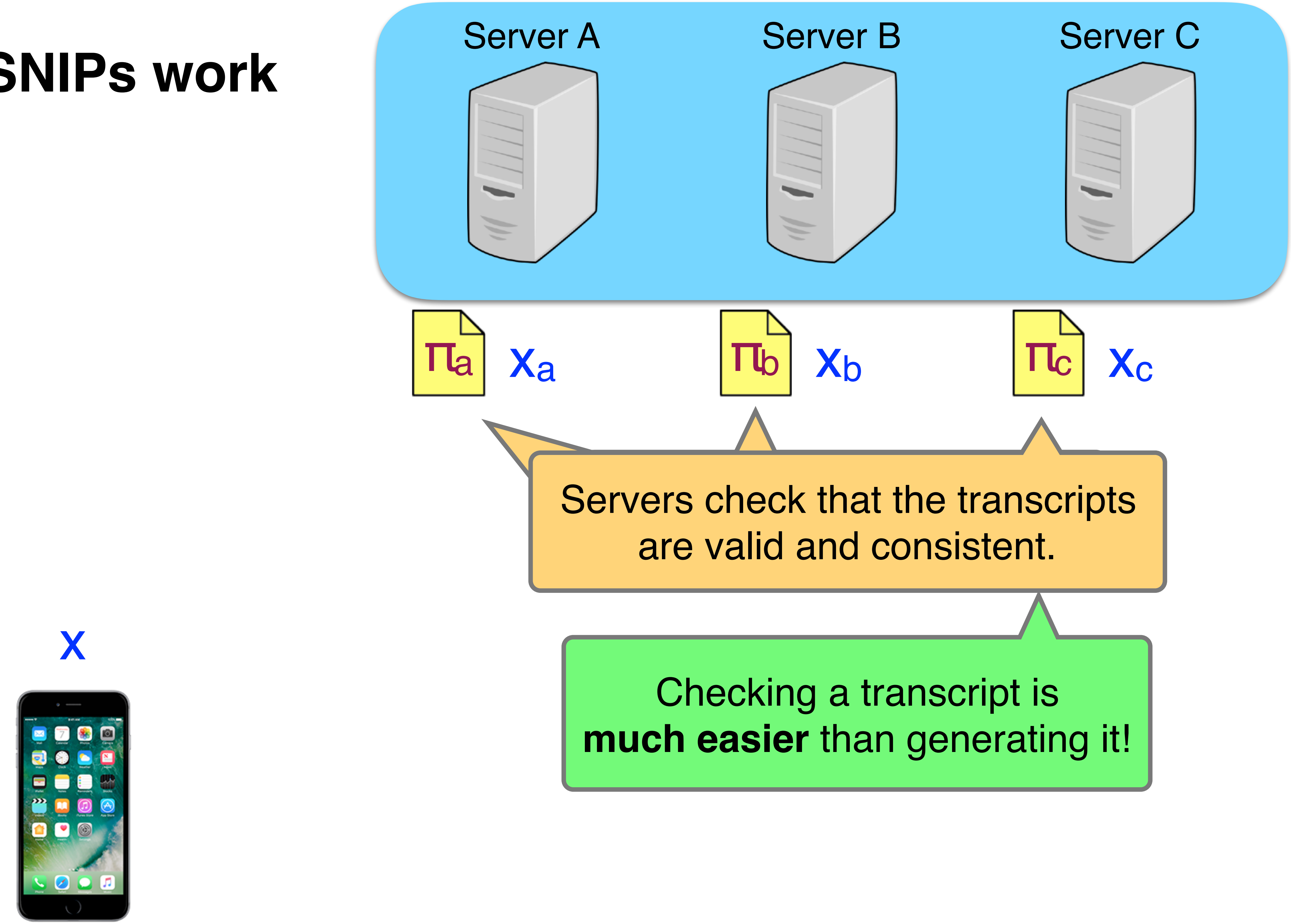


How SNIPs work

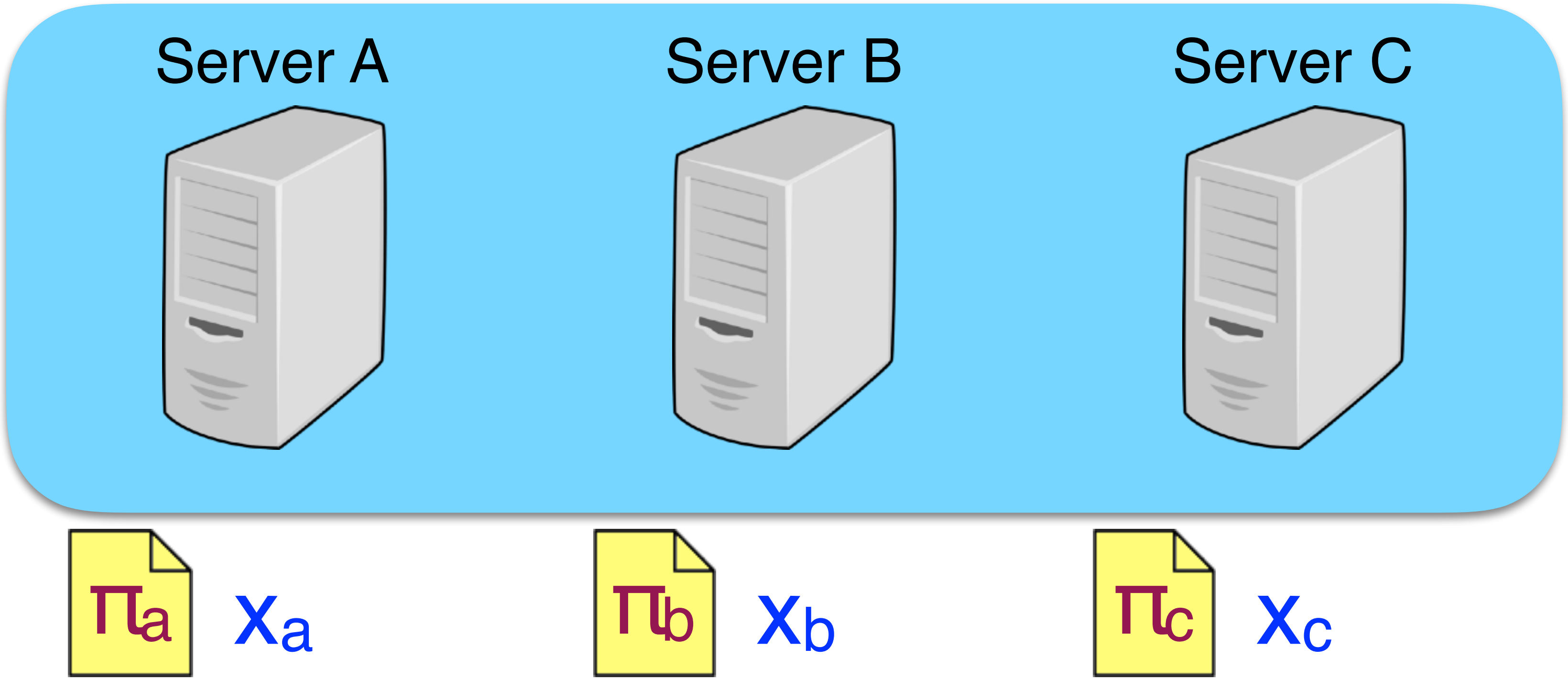


Servers check that the transcripts are valid and consistent.

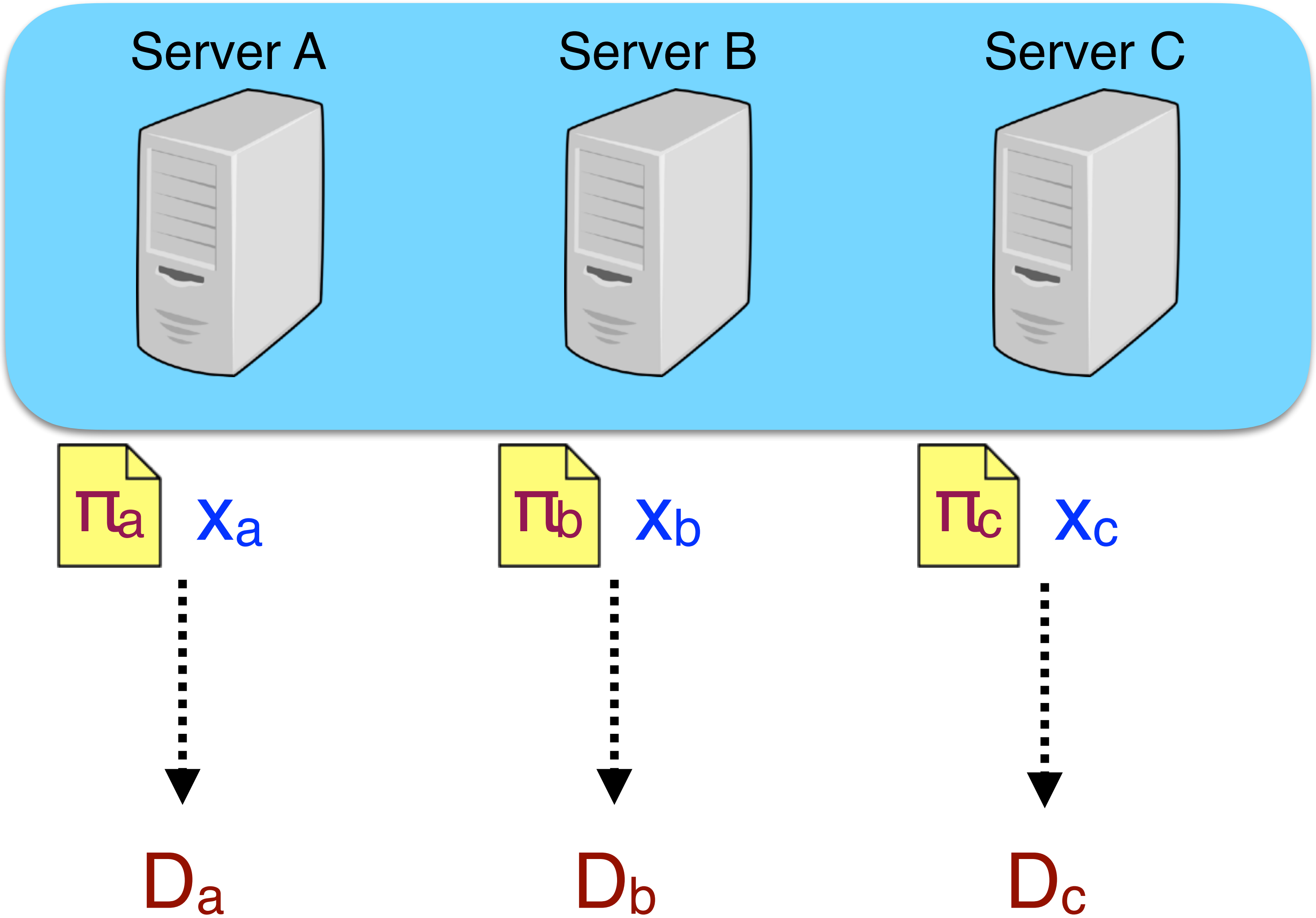
How SNIPs work



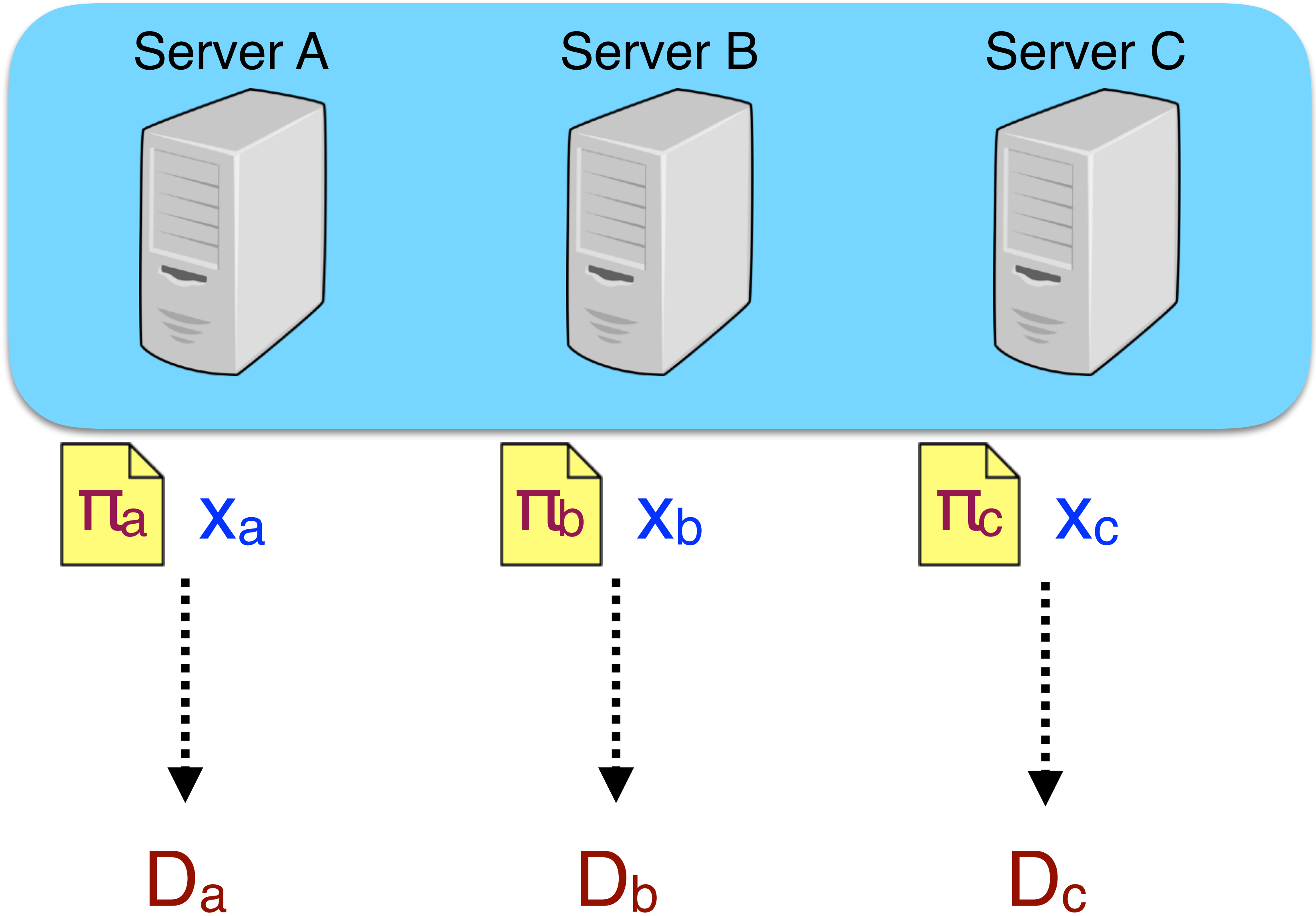
How SNIPs work



How SNIPs work

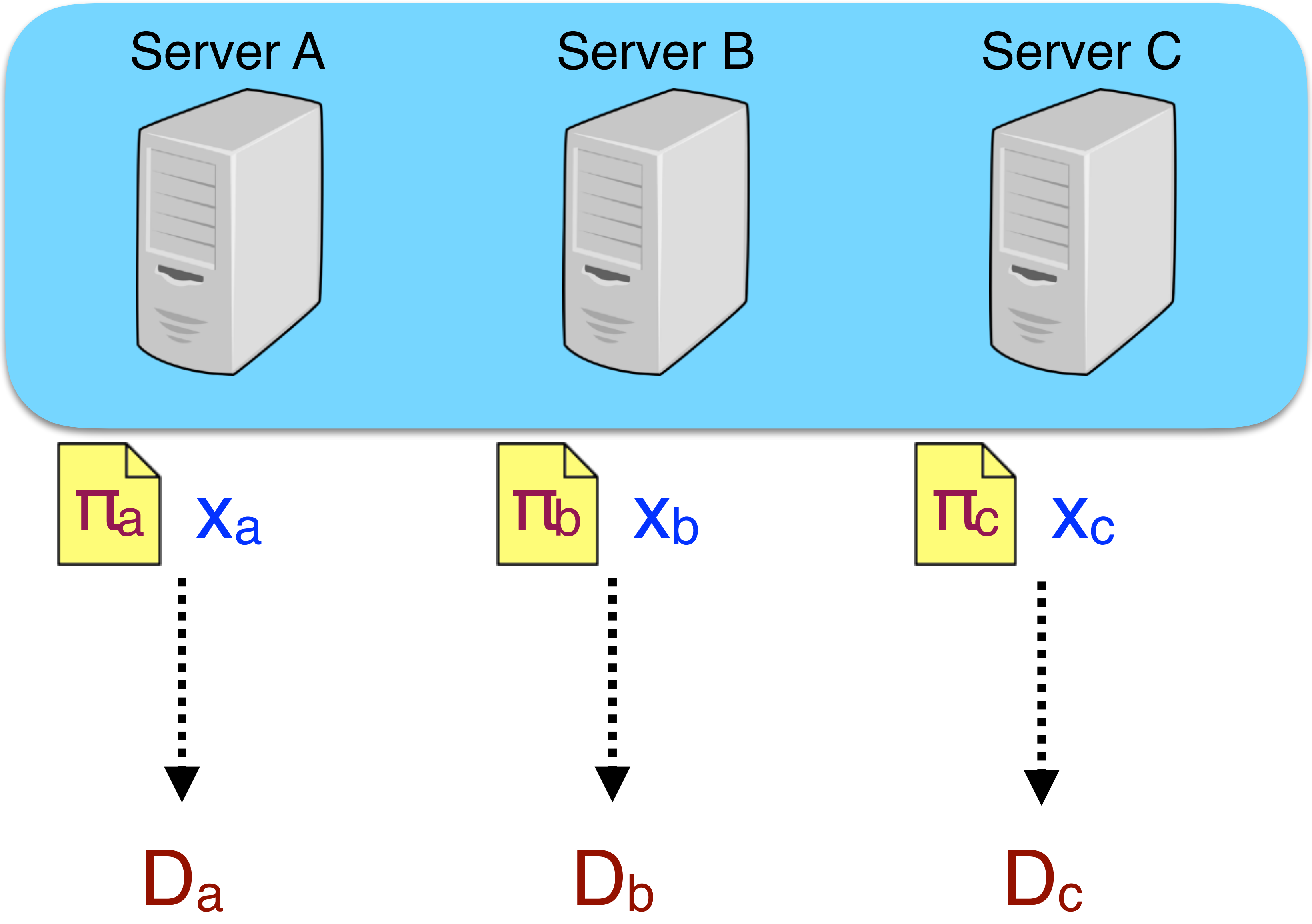


How SNIPs work

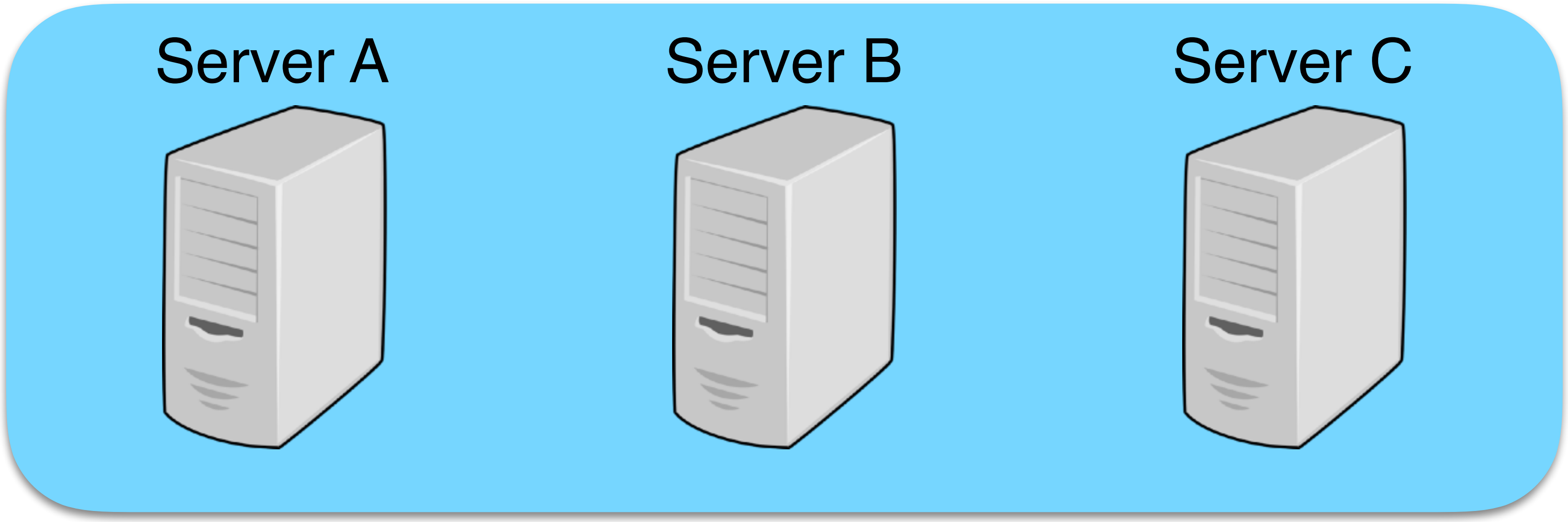


“Randomized digest” of the transcript

How SNIPs work



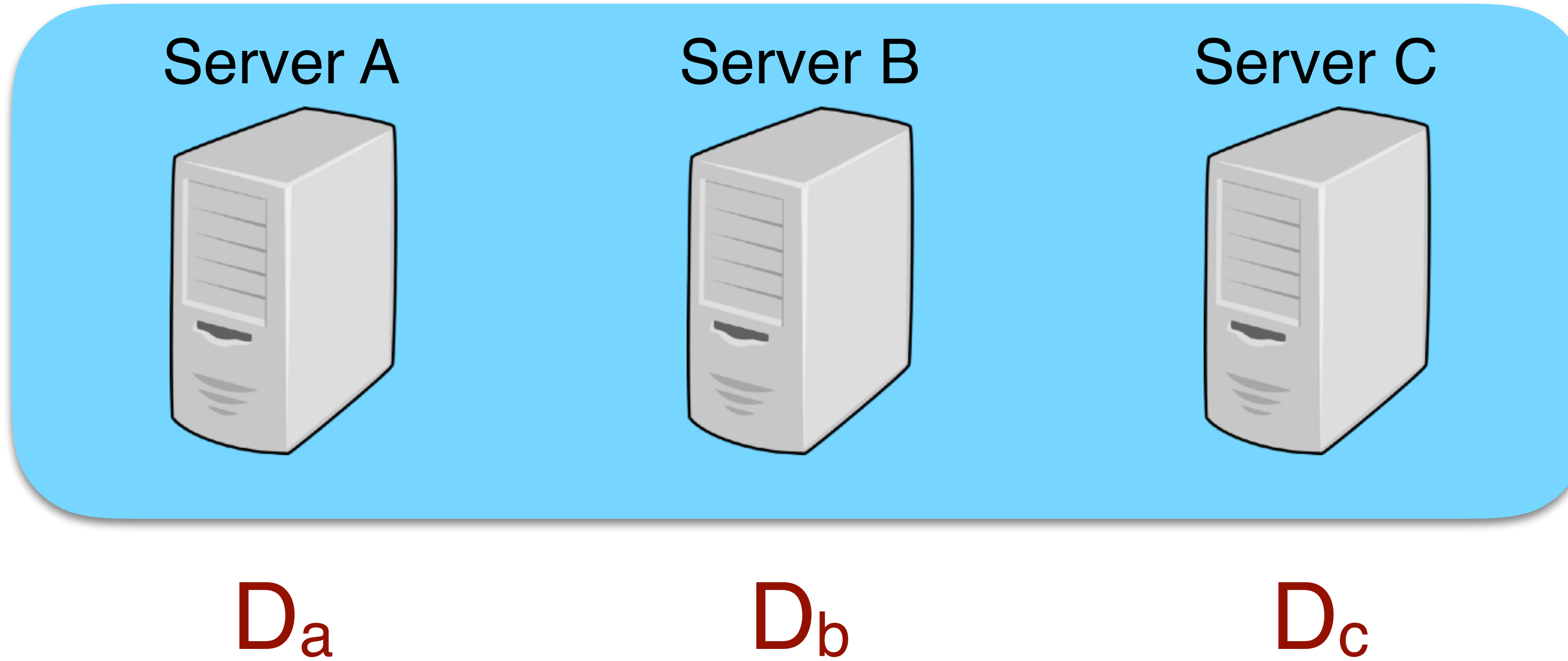
How SNIPs work



D_a D_b D_c

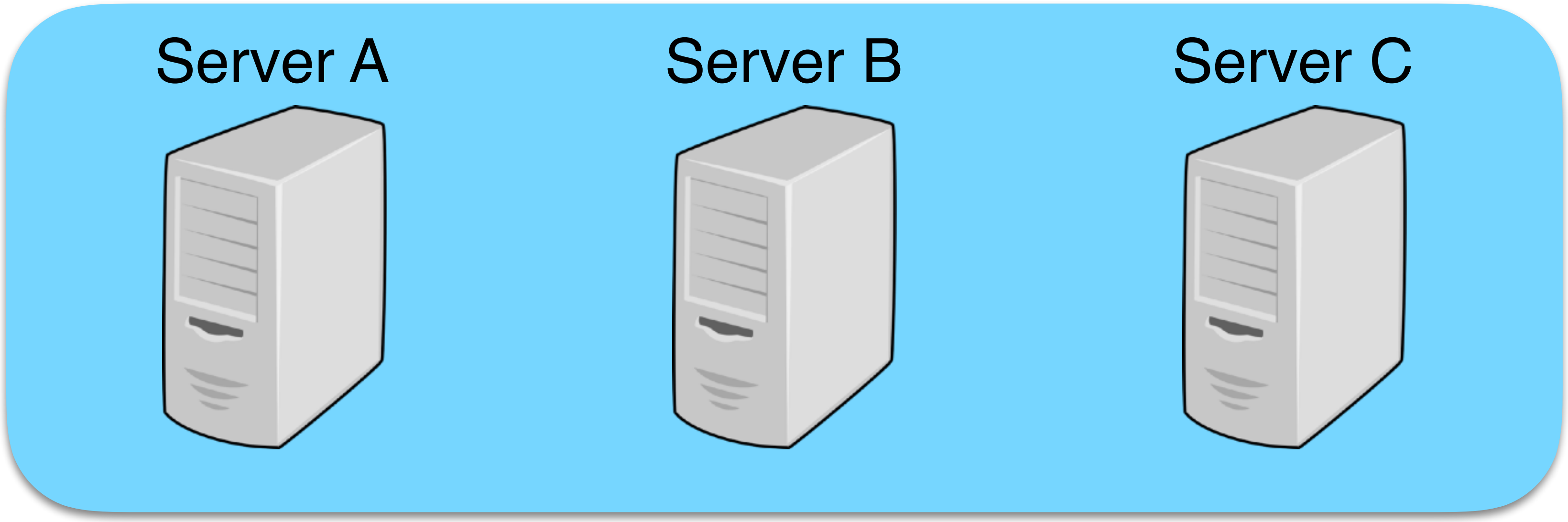
[BFO12]

How SNIPs work



[BFO12]

How SNIPs work



D_a D_b D_c

- If x is valid, $D_a + D_b + D_c = 0$
- If x is invalid, $D_a + D_b + D_c \neq 0$ with high probability

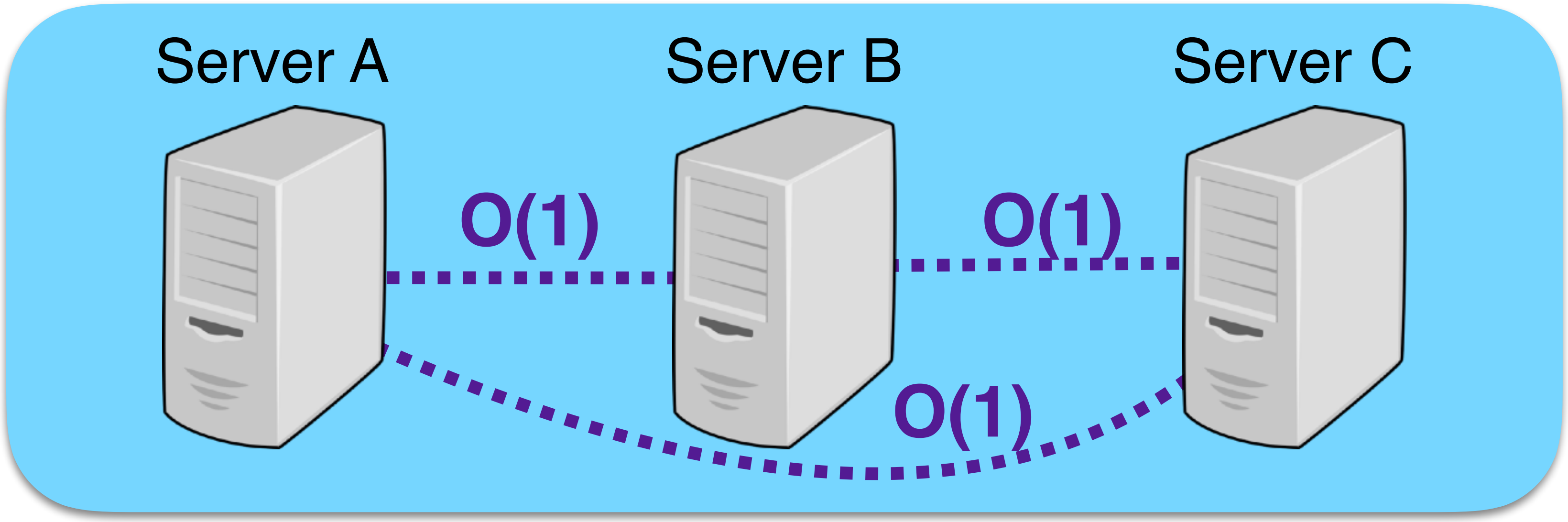
Servers run lightweight multi-party computation to check that

$$D_a + D_b + D_c = 0$$

If so, servers accept x is valid.

[BFO12]

How SNIPs work



D_a D_b D_c

- If x is valid, $D_a + D_b + D_c = 0$
- If x is invalid, $D_a + D_b + D_c \neq 0$ with high probability

Servers run lightweight multi-party computation to check that

$$D_a + D_b + D_c = 0$$

If so, servers accept x is valid.

[BFO12]

M = # of multiplication gates in Valid(·) circuit	**Public-key ops.**		**Communication**		**Slow-down**
	Client	Server	C-to-S	S-to-S	

M = # of multiplication gates in Valid(·) circuit	**Public-key ops.** Client	Server	**Communication** C-to-S	S-to-S	**Slow-down**
Dishonest-maj. MPC [CLOS02], [DPSZ12], …	0	Θ(M)	0	Θ(M)	5,000x at server

M = # of multiplication gates in Valid(·) circuit	**Public-key ops.** Client	Server	**Communication** C-to-S	S-to-S	**Slow-down**
Dishonest-maj. MPC [CLOS02], [DPSZ12], …	0	Θ(M)	0	Θ(M)	5,000x at server
Commits + NIZKs [FS86], [CP92], [CS97], …	Θ(M)	Θ(M)	Θ(M)	Θ(M)	50x at server

M = # of multiplication gates in Valid(·) circuit	**Public-key ops.** Client	Server	**Communication** C-to-S	S-to-S	**Slow-down**
Dishonest-maj. MPC [CLOS02], [DPSZ12], …	0	$\Theta(M)$	0	$\Theta(M)$	5,000x at server
Commits + NIZKs [FS86], [CP92], [CS97], …	$\Theta(M)$	$\Theta(M)$	$\Theta(M)$	$\Theta(M)$	50x at server
Commits + SNARKs [GGPR13], [BCGTV13], …	$\Theta(M)$	$O(1)$	$O(1)$	$O(1)$	500x at client

M = # of multiplication gates in Valid(·) circuit	**Public-key ops.** Client	Server	**Communication** C-to-S	S-to-S	**Slow-down**
Dishonest-maj. MPC [CLOS02], [DPSZ12], …	0	Θ(M)	0	Θ(M)	5,000x at server
Commits + NIZKs [FS86], [CP92], [CS97], …	Θ(M)	Θ(M)	Θ(M)	Θ(M)	50x at server
Commits + SNARKs [GGPR13], [BCGTV13], …	Θ(M)	O(1)	O(1)	O(1)	500x at client
This work: SNIPs	0	0	Θ(M)	O(1)	1x

M = # of multiplication gates in Valid(·) circuit	Public-key ops. Client	Public-key ops. Server	Communication C-to-S	Communication S-to-S	Slow-down
Dishonest-maj. MPC [CLOS02], [DPSZ12], …	0	Θ(M)	0	Θ(M)	5,000x at server
Commits + NIZKs [FS86], [CP92], [CS97], …	Θ(M)	Θ(M)	Θ(M)	Θ(M)	50x at server
Commits + SNARKs [GGPR13], [BCGTV13], …	Θ(M)	O(1)	O(1)	O(1)	500x at client
This work: SNIPs	0	0	Θ(M)	O(1)	1x

For specific Valid() circuits, it is possible to eliminate this cost [BGI16]

M = # of multiplication gates in Valid(·) circuit	**Public-key ops.** Client	Server	**Communication** C-to-S	S-to-S	**Slow-down**
Dishonest-maj. MPC [CLOS02], [DPSZ12], …	0	Θ(M)	0	Θ(M)	5,000x at server
Commits + NIZKs [FS86], [CP92], [CS97], …	Θ(M)	Θ(M)	Θ(M)	Θ(M)	50x at server
Commits + SNARKs [GGPR13], [BCGTV13], …	Θ(M)	O(1)	O(1)	O(1)	500x at client
This work: SNIPs	0	0	Θ(M)	O(1)	1x

Outline

- Background: The private aggregation problem
- A straw-man solution for private sums
- **Providing robustness with SNIPs**
- Evaluation
- Encodings for <u>complex</u> aggregates

Outline

- Background: The private aggregation problem
- A straw-man solution for private sums
- Providing robustness with SNIPs
- **Evaluation**
- Encodings for <u>complex</u> aggregates

Evaluation

- Implemented Prio in Go
  (see optimizations described in paper)
- Five-server cluster in EC2
- System collects the sum of "N" 0/1 values

Four variants

1. No privacy
2. No robustness ("straw man")
3. Prio (privacy + robustness)
4. NIZK (privacy + robustness)

Evaluation

- Implemented Prio in Go
  (see optimizations described in paper)
- Five-server cluster in EC2
- System collects the sum of “N” 0/1 values

E.g., for privately measuring telemetry data.

Four variants

1. No privacy
2. No robustness (“straw man”)
3. Prio (privacy + robustness)
4. NIZK (privacy + robustness)

Evaluation

- Implemented Prio in Go
  (see optimizations described in paper)
- Five-server cluster in EC2
- System collects the sum of “N” 0/1 values

Four variants

1. No privacy
2. No robustness (“straw man”)
3. Prio (privacy + robustness)
4. NIZK (privacy + robustness)

Evaluation

- Implemented Prio in Go
  (see optimizations described in paper)
- Five-server cluster in EC2
- System collects the sum of "N" 0/1 values

Four variants

1. No privacy ← **one** server
2. No robustness ("straw man")
3. Prio (privacy + robustness)
4. NIZK (privacy + robustness)

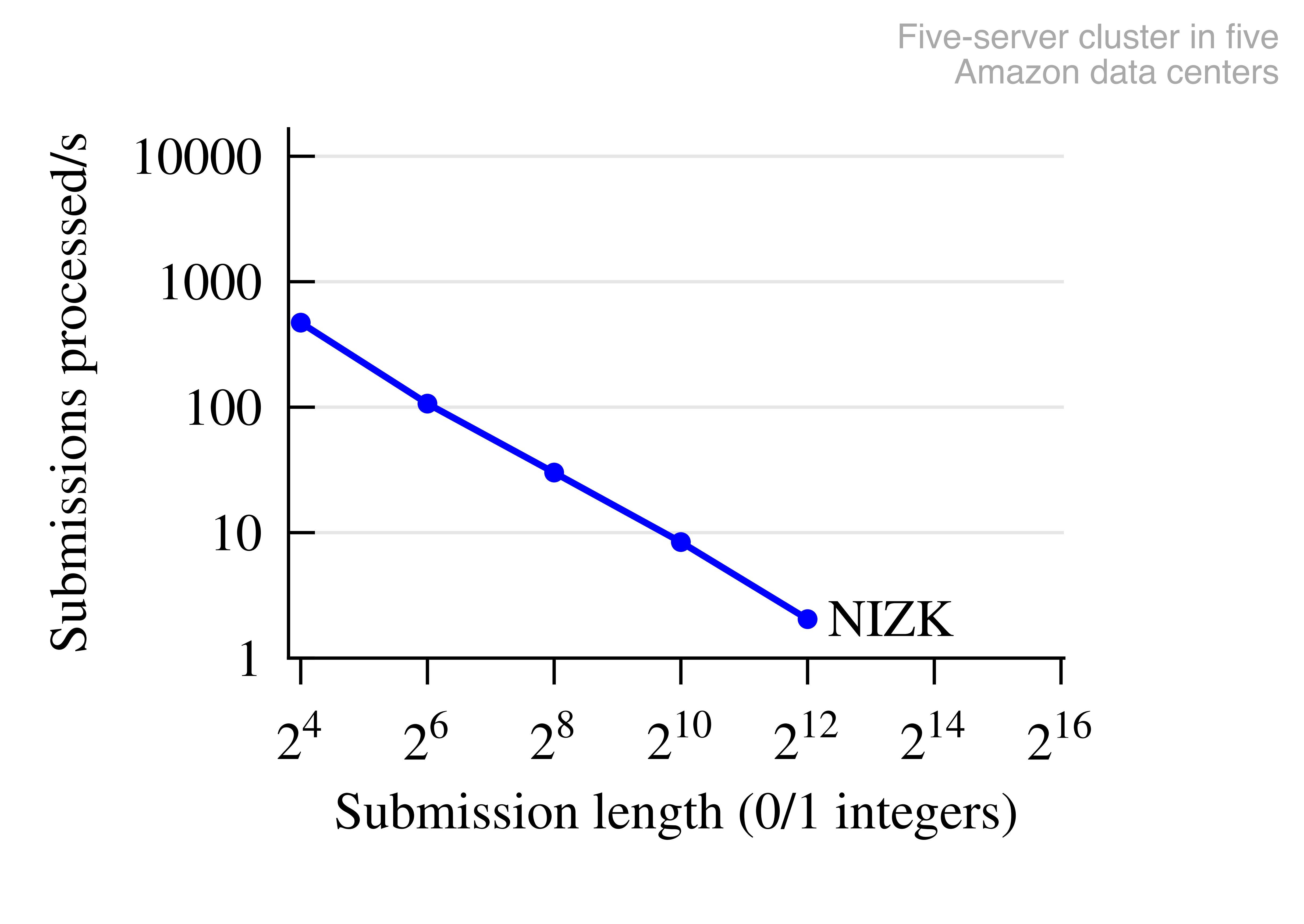
Evaluation

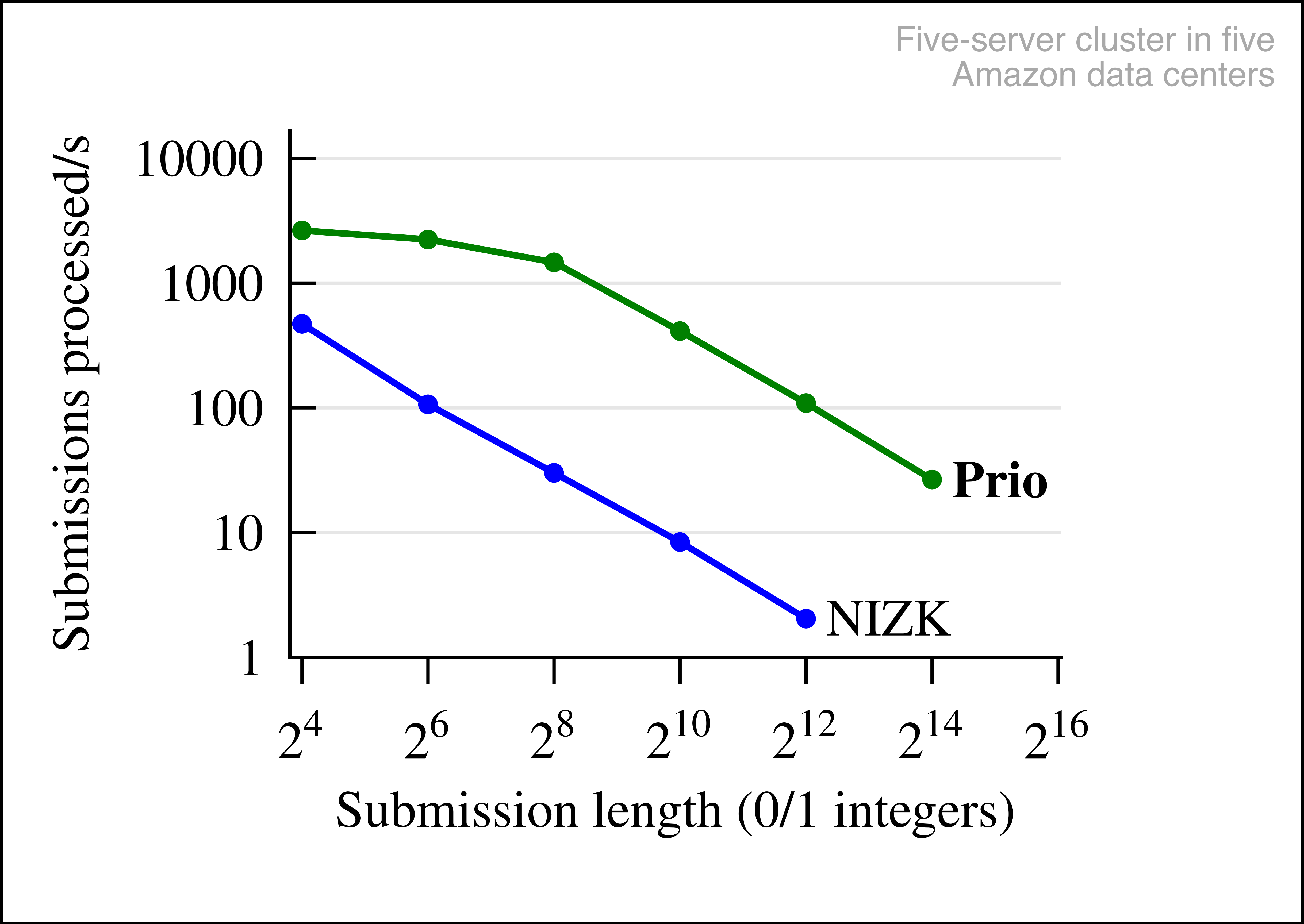
- Implemented Prio in Go
  (see optimizations described in paper)
- Five-server cluster in EC2
- System collects the sum of “N” 0/1 values

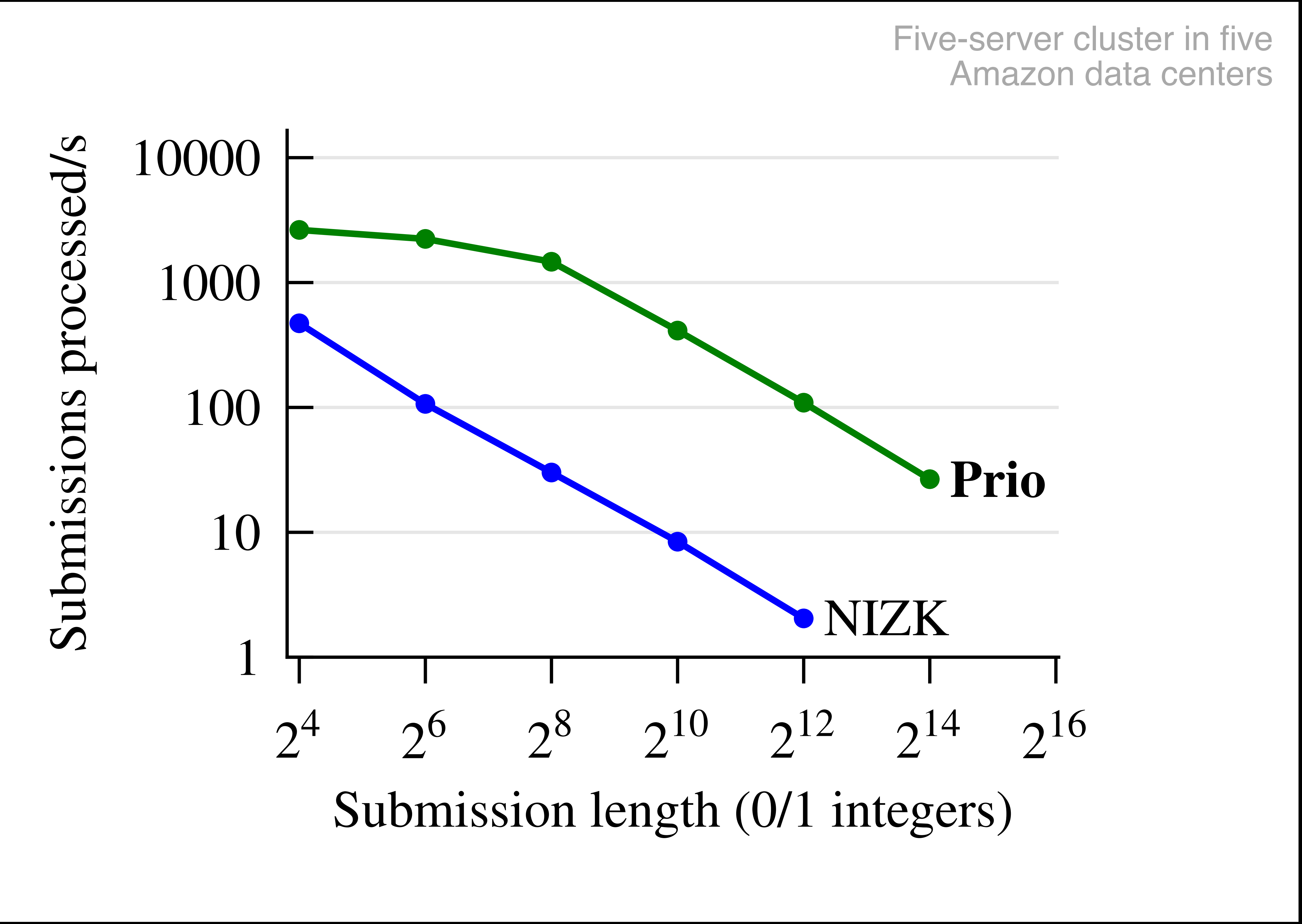
Four variants

1. No privacy ← **one** server
2. No robustness (“straw man”)
3. Prio (privacy + robustness)
4. NIZK (privacy + robustness)

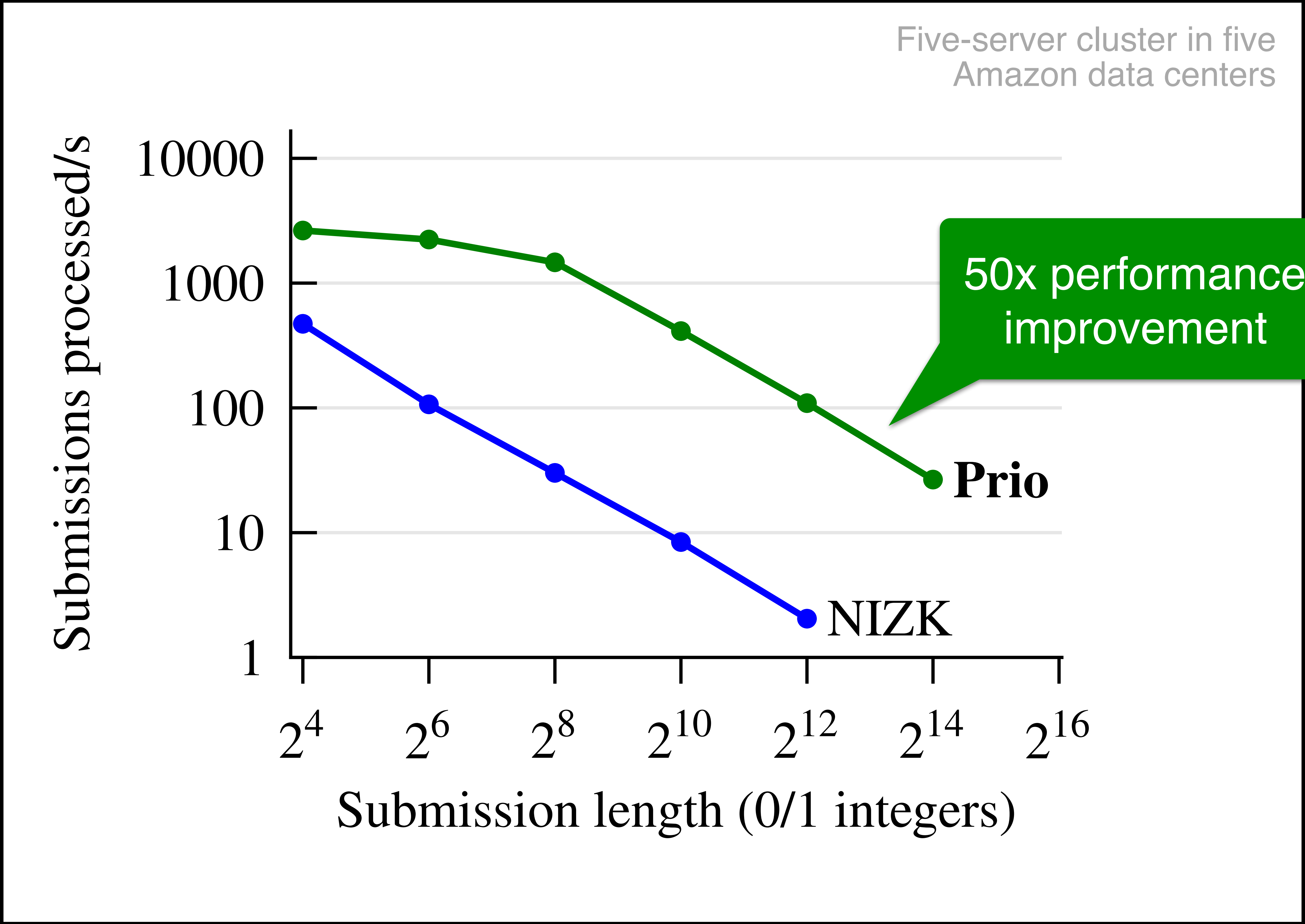
five servers


Five-server cluster in five
Amazon data centers
Submissions processed/s
10000
1000
100
10
1
NIZK
2^4
2^6
2^8
2^{10}
2^{12}
2^{14}
2^{16}
Submission length (0/1 integers)


Five-server cluster in five
Amazon data centers
Submissions processed/s
10000
1000
100
10
1
Prio
NIZK
2^4
2^6
2^8
2^{10}
2^{12}
2^{14}
2^{16}
Submission length (0/1 integers)

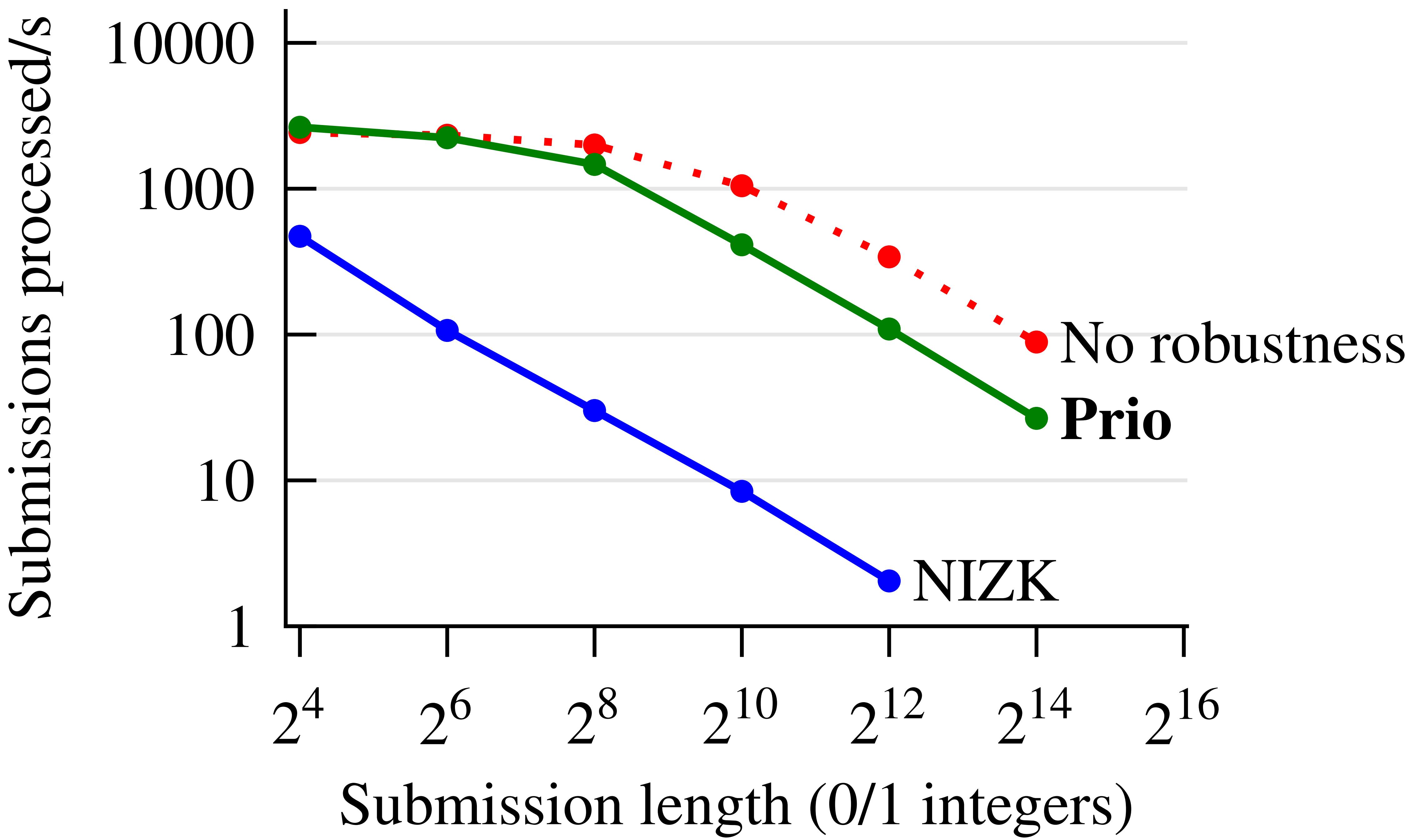


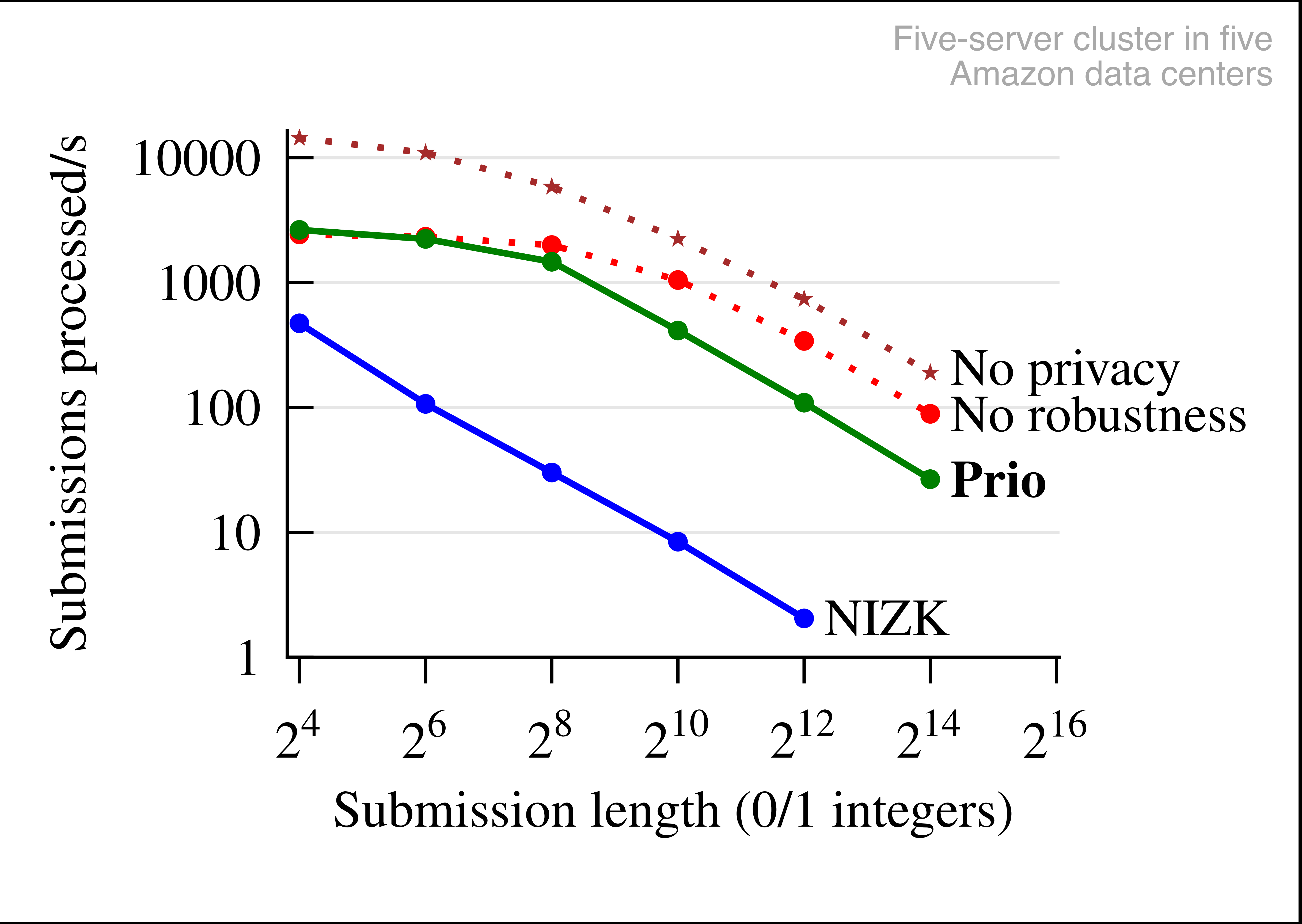
Five-server cluster in five
Amazon data centers
Submissions processed/s
10000
1000
100
10
1
Prio
NIZK
2^4
2^6
2^8
2^{10}
2^{12}
2^{14}
2^{16}
Submission length (0/1 integers)



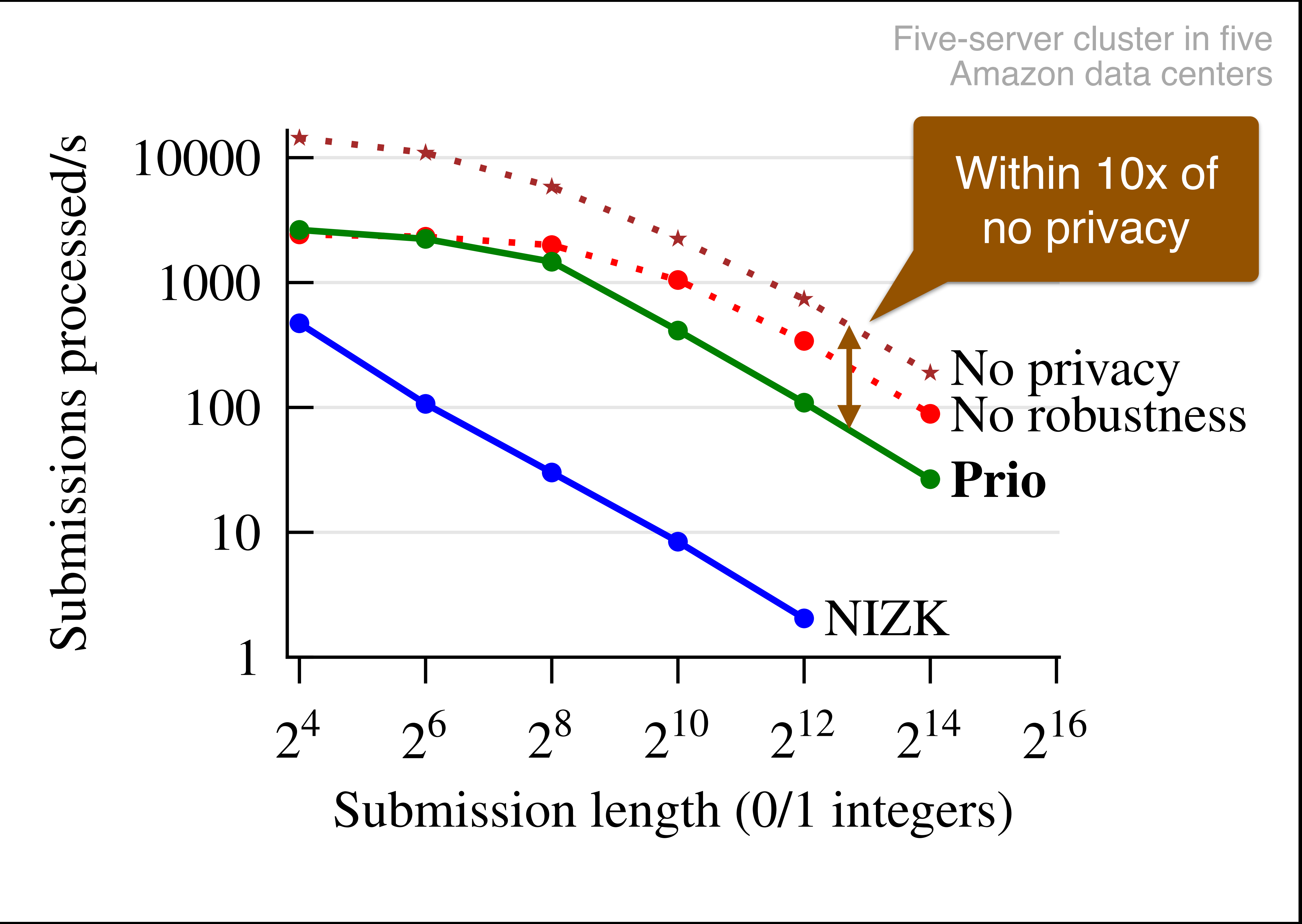
Five-server cluster in five
Amazon data centers
Submissions processed/s
10000
1000
100
10
1
50x performance
improvement
Prio
NIZK
2^4
2^6
2^8
2^{10}
2^{12}
2^{14}
2^{16}
Submission length (0/1 integers)

Five-server cluster in five
Amazon data centers

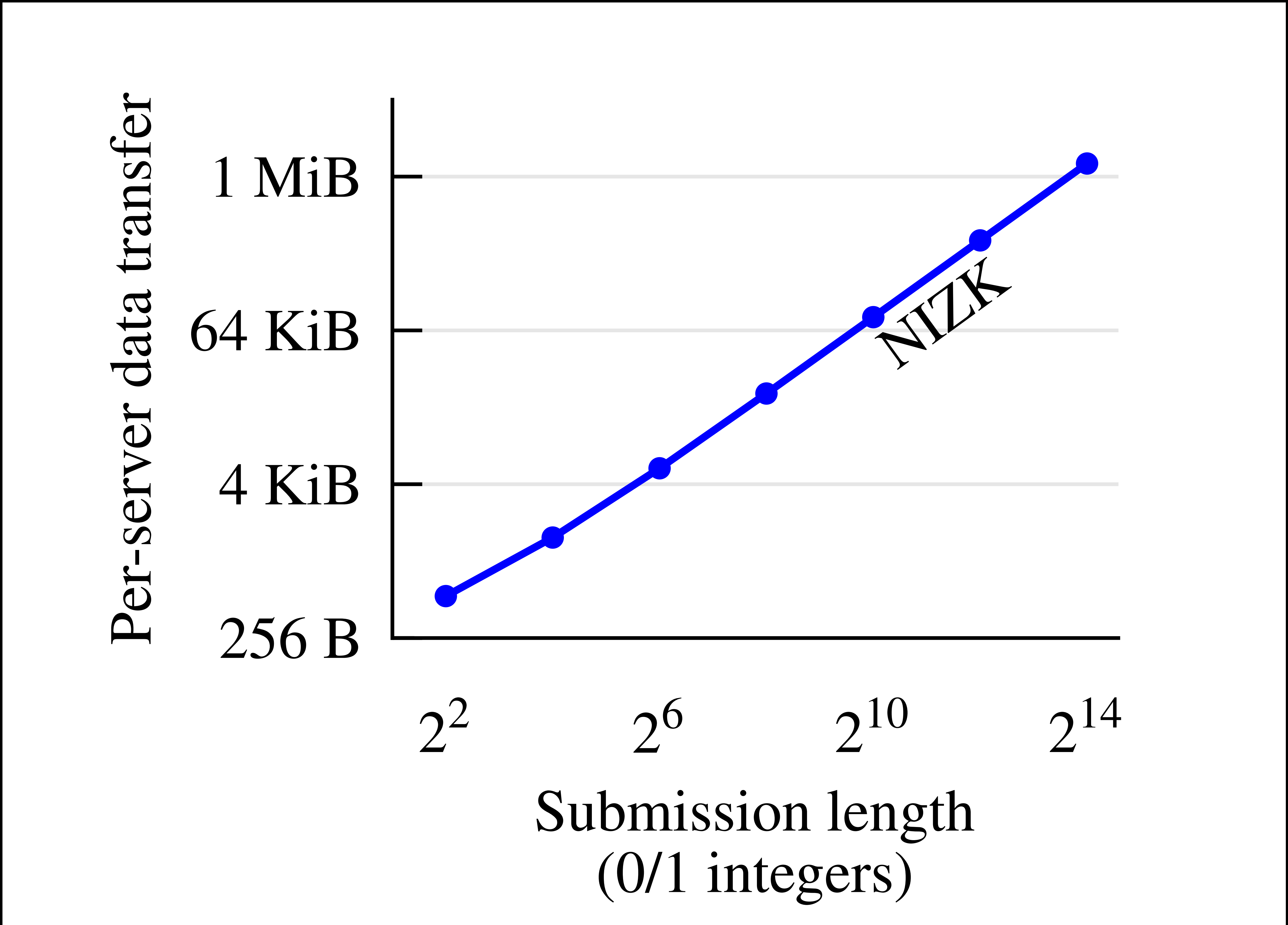

Submissions processed/s
10000
1000
100
10
1
No robustness
Prio
NIZK
2^4
2^6
2^8
2^10
2^12
2^14
2^16
Submission length (0/1 integers)



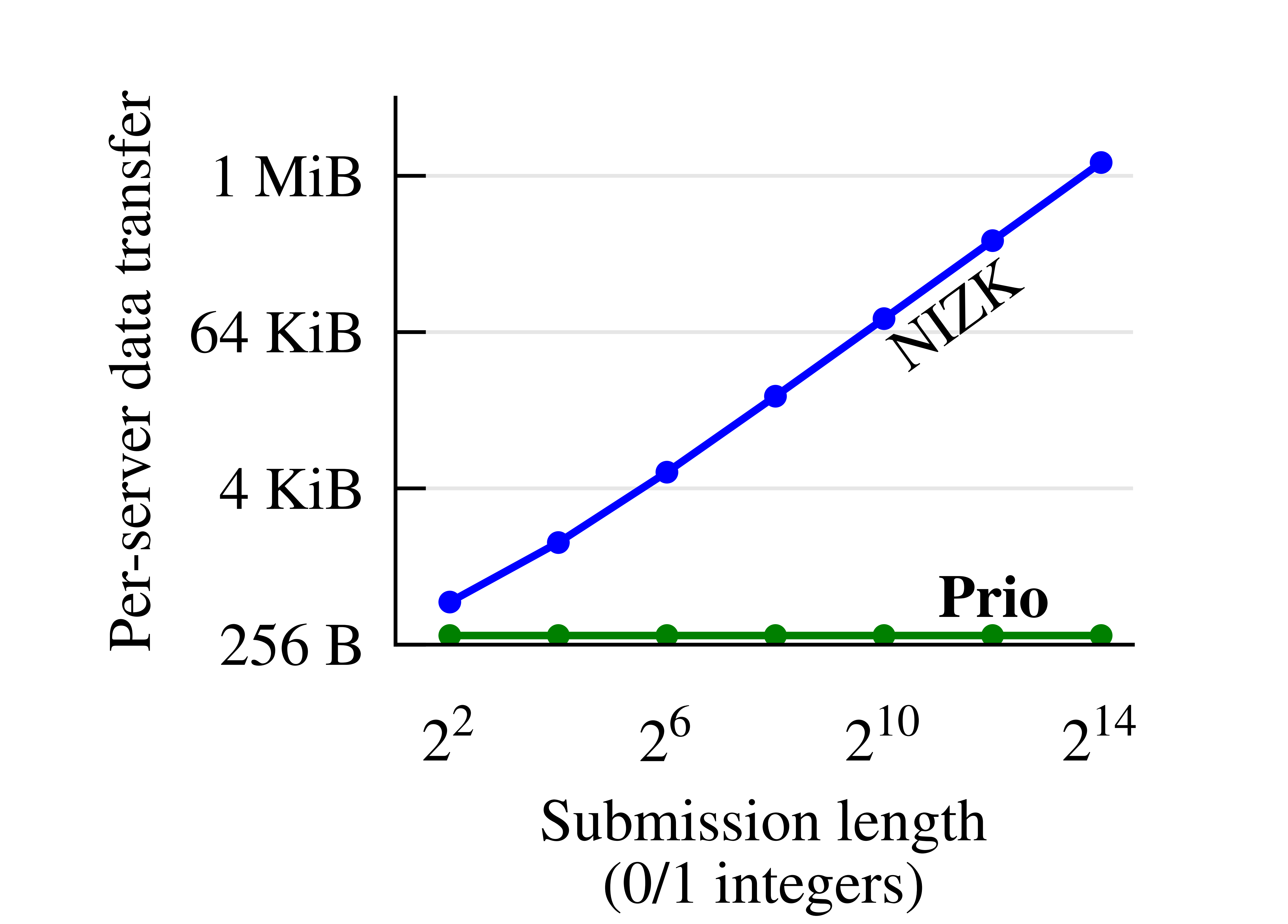
Five-server cluster in five
Amazon data centers
Submissions processed/s
10000
1000
100
10
1
No privacy
No robustness
Prio
NIZK
2^4
2^6
2^8
2^{10}
2^{12}
2^{14}
2^{16}
Submission length (0/1 integers)



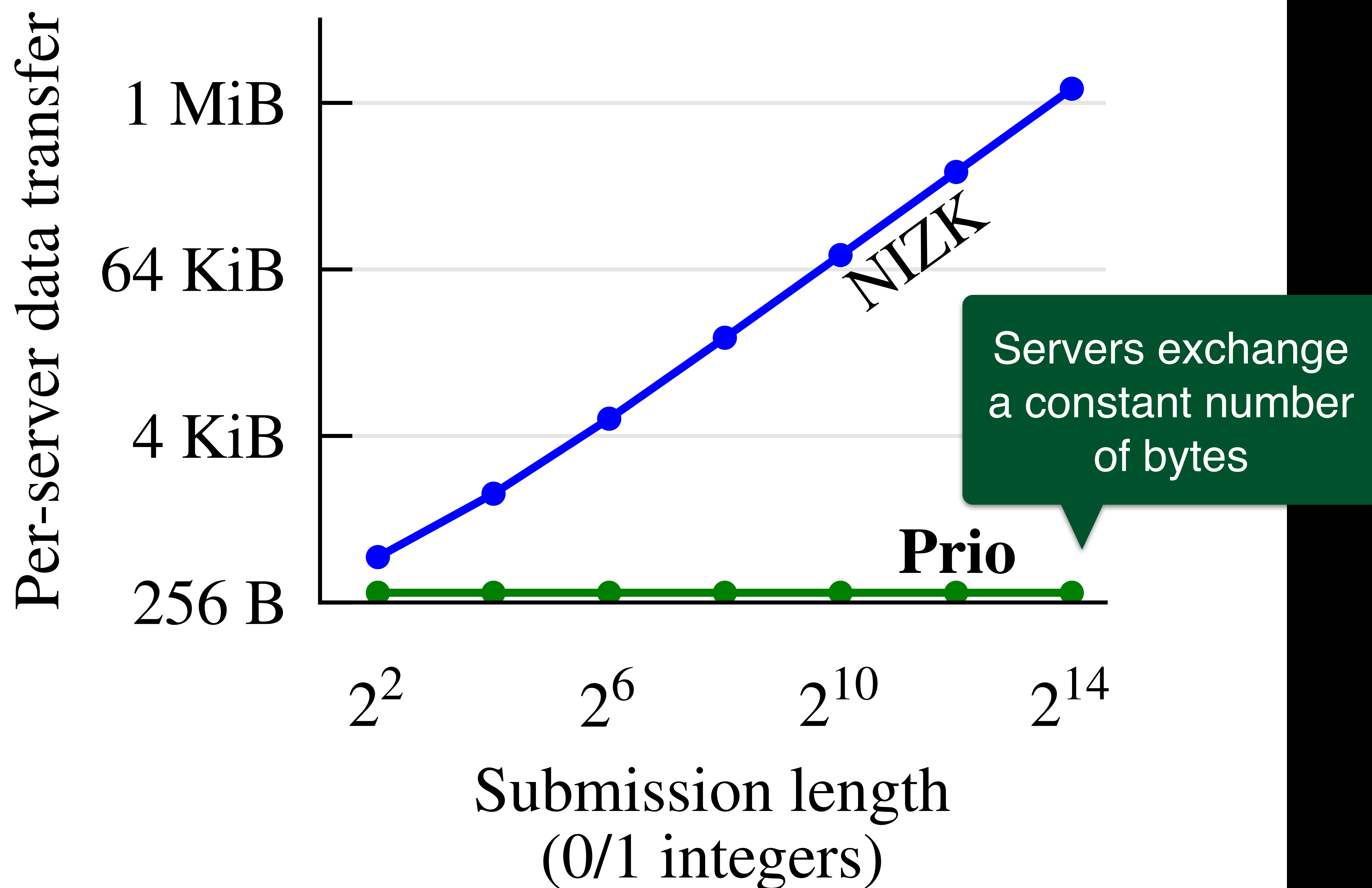
Five-server cluster in five
Amazon data centers
Within 10x of
no privacy
Submissions processed/s
10000
1000
100
10
1
No privacy
No robustness
Prio
NIZK
2^4
2^6
2^8
2^{10}
2^{12}
2^{14}
2^{16}
Submission length (0/1 integers)



Per-server data transfer
1 MiB
64 KiB
4 KiB
256 B
NIZK
2^2
2^6
2^10
2^14
Submission length
(0/1 integers)



Per-server data transfer
1 MiB
64 KiB
4 KiB
256 B
NIZK
Prio
2^2
2^6
2^{10}
2^{14}
Submission length
(0/1 integers)



Per-server data transfer
1 MiB
64 KiB
4 KiB
256 B
NIZK
Prio
Servers exchange a constant number of bytes
2^2
2^6
2^{10}
2^{14}
Submission length (0/1 integers)

Outline

- Background: The private aggregation problem
- A straw-man solution for private sums
- Providing robustness with SNIPs
- **Evaluation**
- Encodings for complex aggregates

Outline

- Background: The private aggregation problem
- A straw-man solution for private sums
- Providing robustness with SNIPs
- Evaluation
- **Encodings for complex aggregates**

Known techniques: Complex statistics

If you can compute private sums, you can compute many other interesting aggregates using known techniques

[PrivStats11], [KDK11], [DFKZ13], [PrivEx14], [MDD16], …

- Average
- Variance
- Standard deviation
- Most popular (approx)
- “Heavy hitters” (approx)
- Min and max (approx)
- Quality of arbitrary regression model (R^2)
- Least-squares regression
- Stochastic gradient descent [Bonawitz et al. 2016]

Known techniques: Complex statistics

If you can compute private sums, you can compute many other interesting aggregates using known techniques

[PrivStats11], [KDK11], [DFKZ13], [PrivEx14], [MDD16], …

- Average
- Variance
- Standard deviation
- Most popular (approx)
- “Heavy hitters” (approx)
- Min and max (approx)
- Quality of arbitrary regression model (R^2)
- Least-squares regression
- Stochastic gradient descent [Bonawitz et al. 2016]

Contribution 2:
SNIP-friendly encodings for these statistics

Known techniques: Complex statistics

If you can compute private sums, you can compute many other interesting aggregates using known techniques

[PrivStats11], [KDK11], [DFKZ13], [PrivEx14], [MDD16], …

- Average
- Variance
- Standard deviation
- Most popular (approx)
- “Heavy hitters” (approx)
- Min and max (approx)
- Quality of arbitrary regression model (R^2)
- Least-squares regression
- Stochastic gradient descent [Bonawitz et al. 2016]

Contribution 2: SNIP-friendly encodings for these statistics

Prio can’t compute <u>all</u> statistics efficiently

Known techniques: Complex statistics

If you can compute private sums, you can compute many other interesting aggregates using known techniques

[PrivStats11], [KDK11], [DFKZ13], [PrivEx14], [MDD16], …

- Average
- Variance
- Standard deviation
- Most popular (approx)
- “Heavy hitters” (approx)
- Min and max (approx)
- Quality of arbitrary regression model (R^2)
- Least-squares regression
- Stochastic gradient descent [Bonawitz et al. 2016]

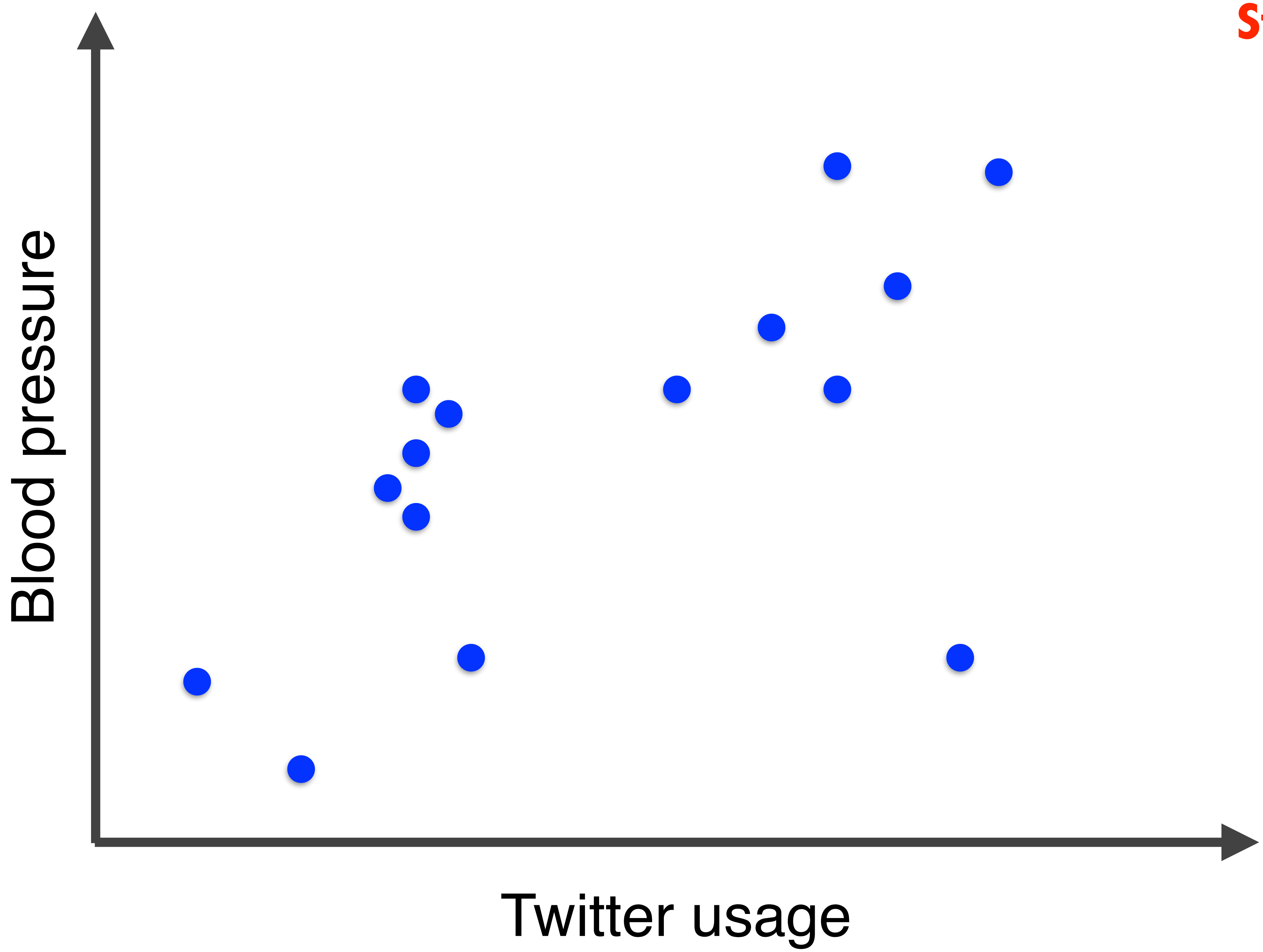
Contribution 2:
SNIP-friendly encodings for these statistics

Prio can’t compute <u>all</u> statistics efficiently

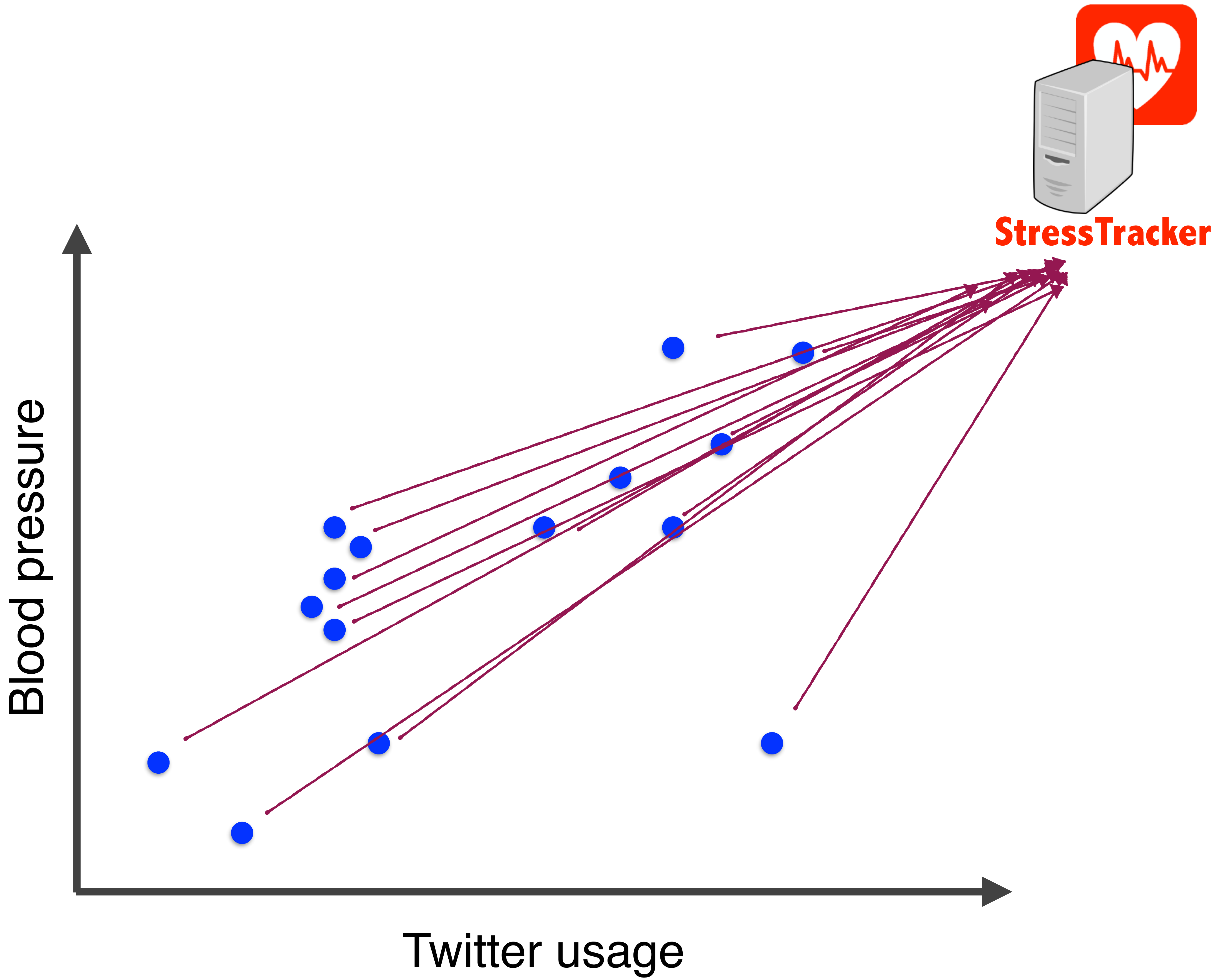
See the paper for the details

Today

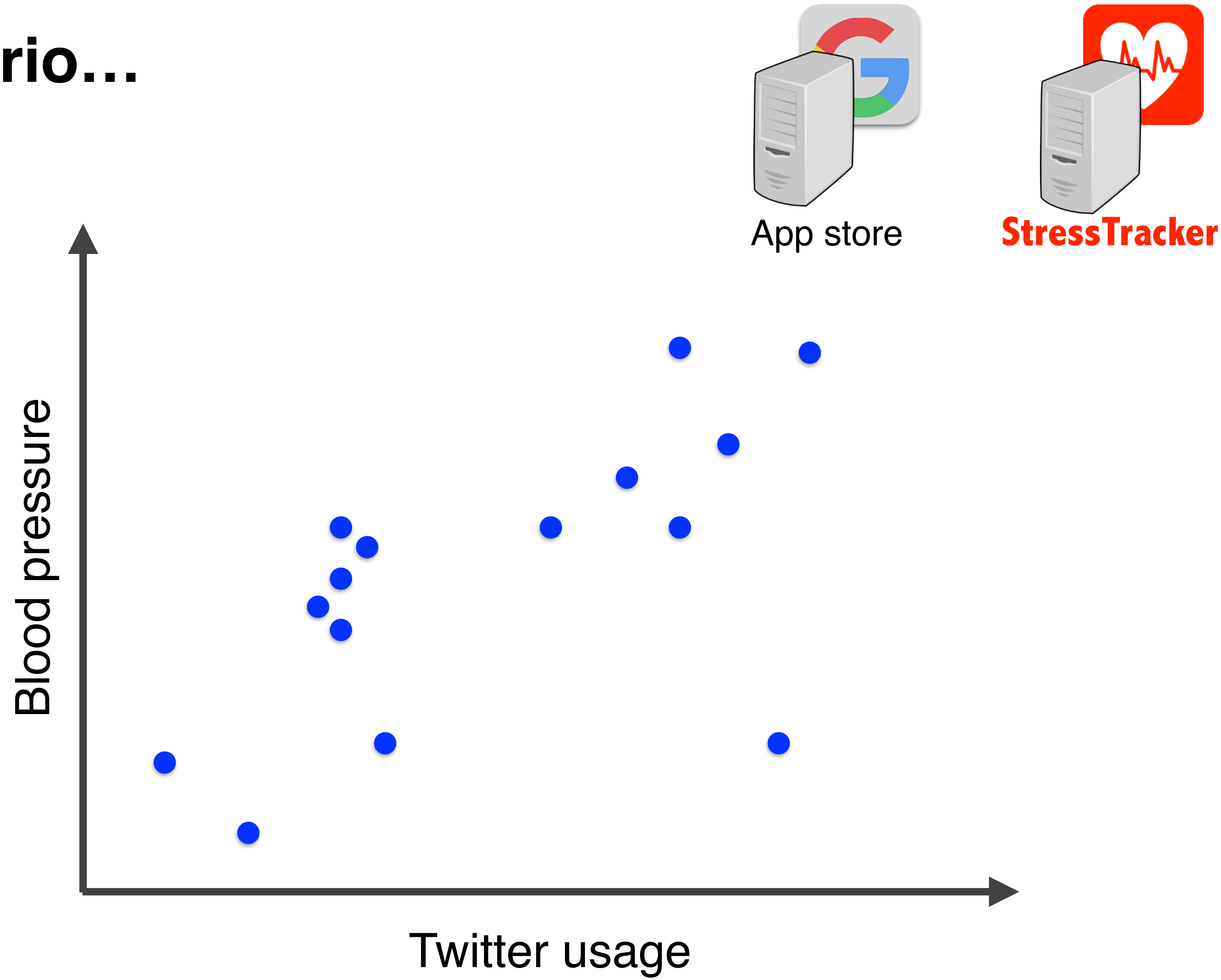
StressTracker



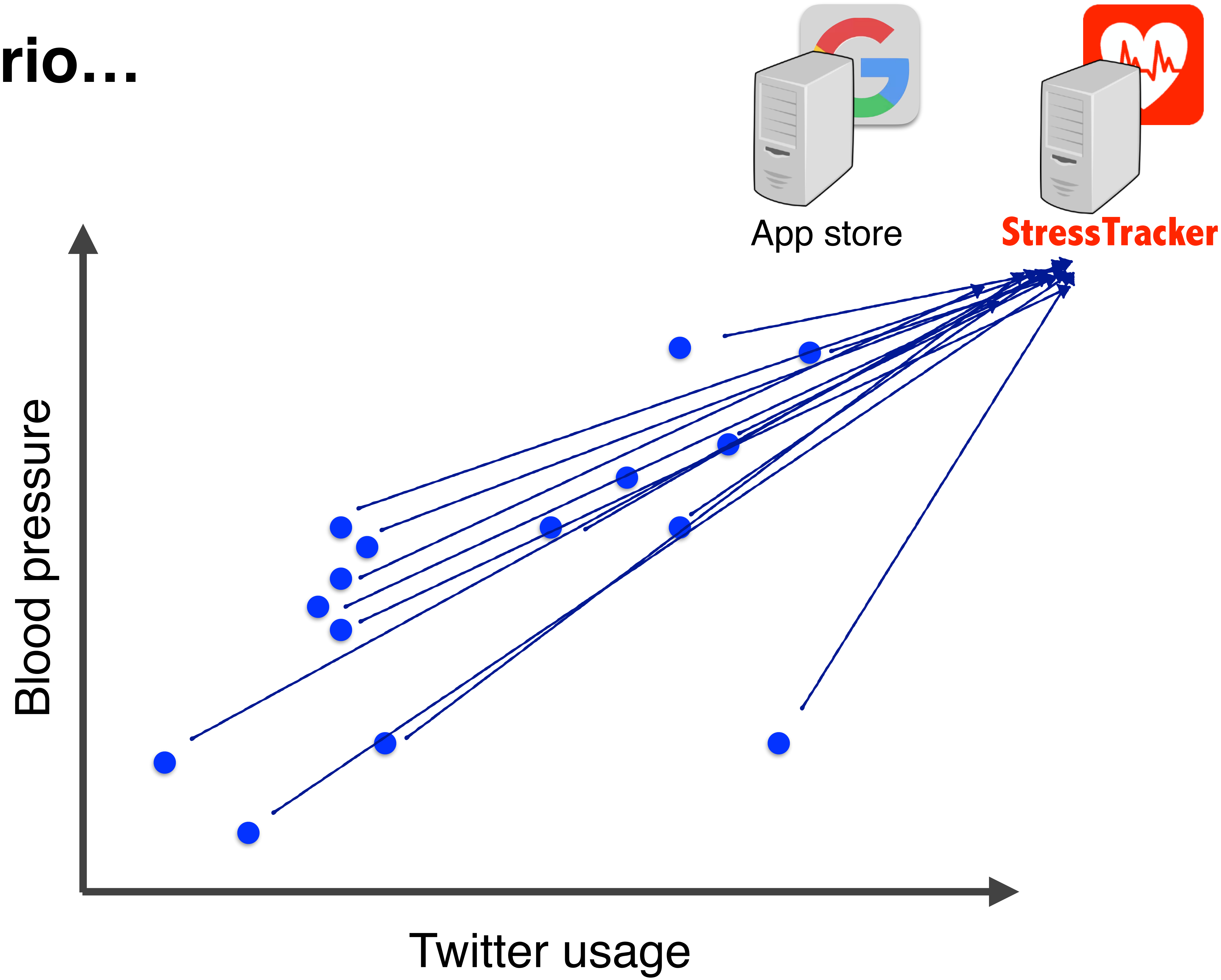
Today



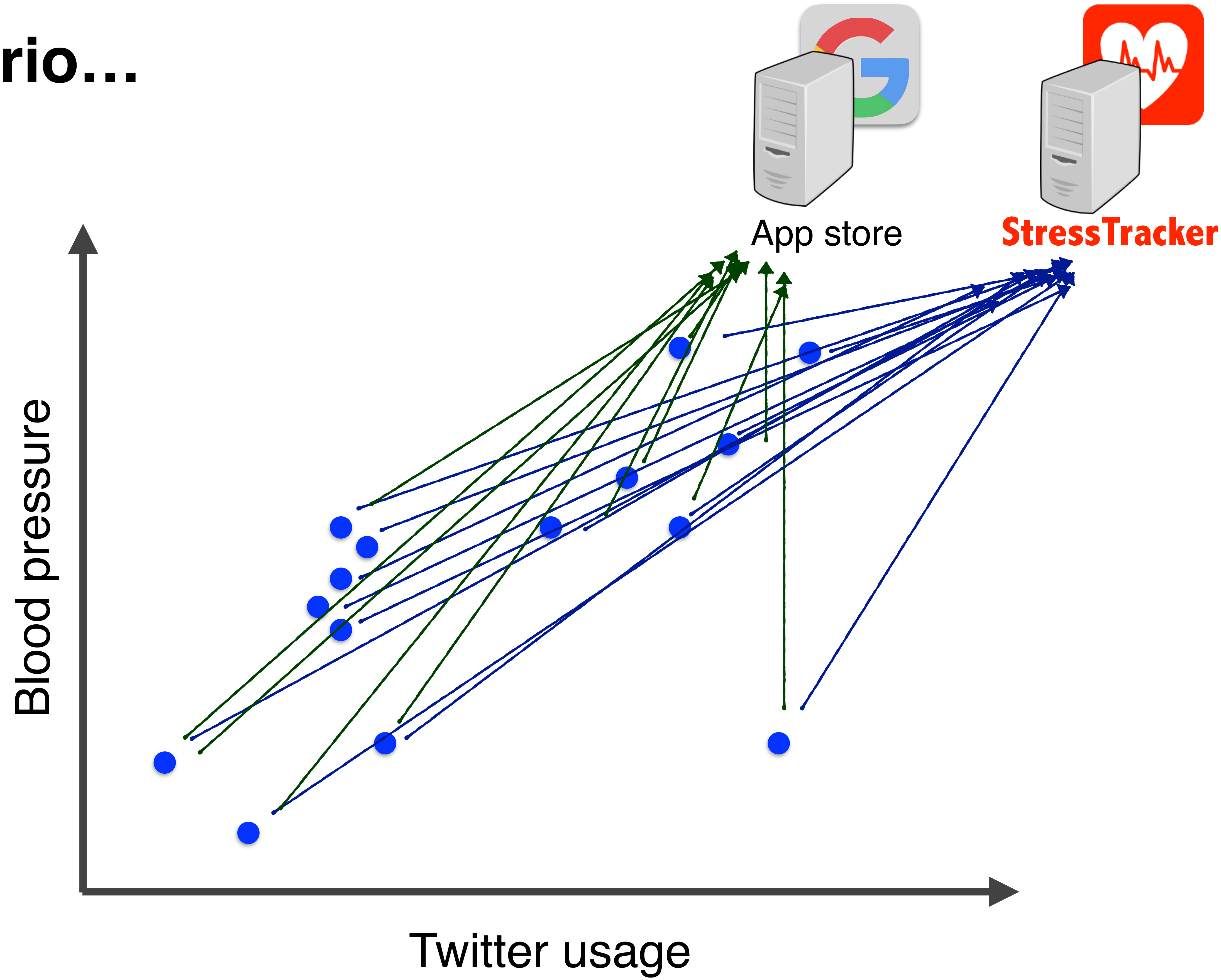
With Prio…



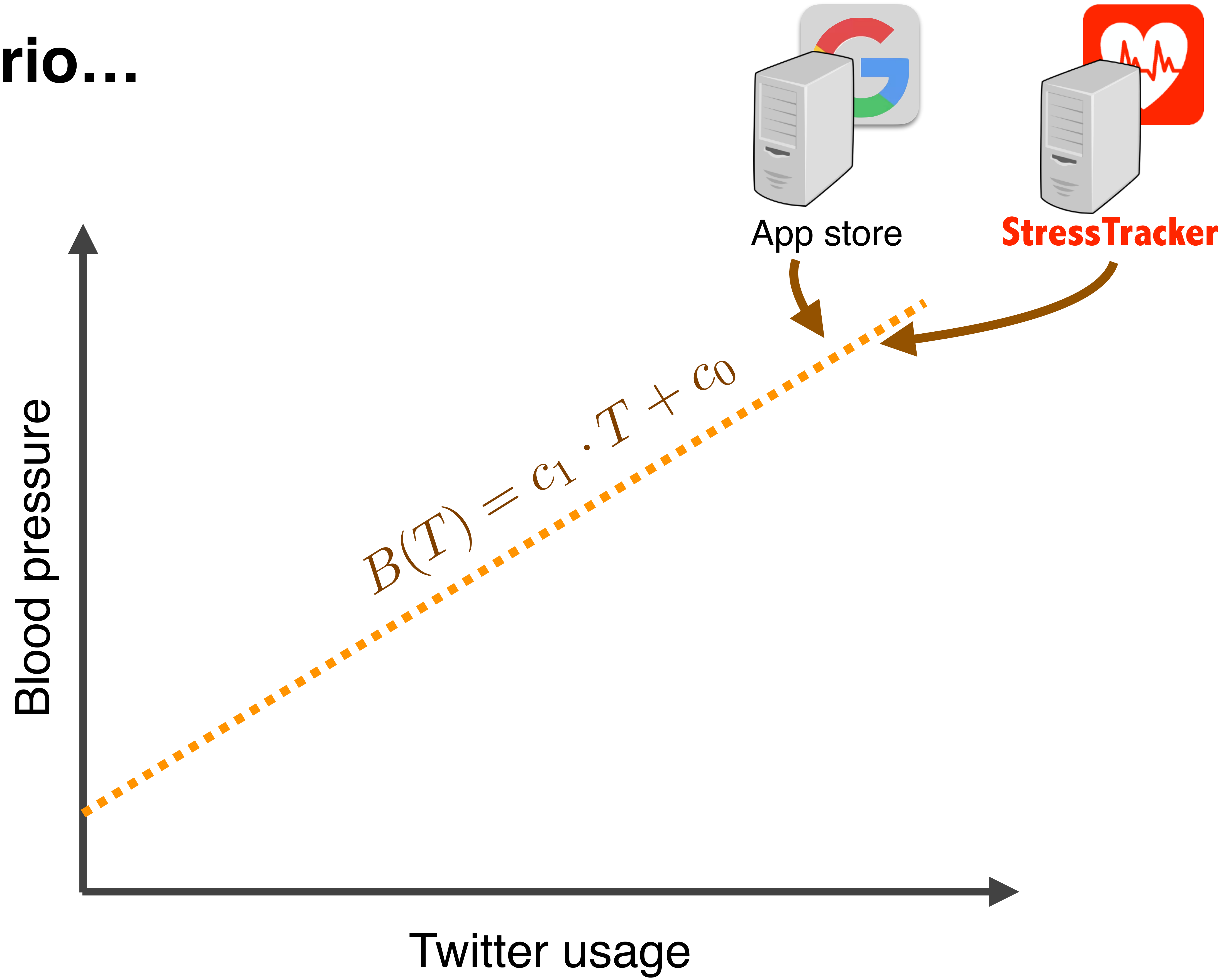
With Prio…



With Prio…



With Prio…



App store
StressTracker
$B(T) = c_1 \cdot T + c_0$
Blood pressure
Twitter usage

Conclusions

- Wholesale collection of sensitive user data puts our security at risk.
- Prio is the first system for aggregation that provides:
  - **exact correctness**,
  - **privacy**,
  - **robustness**, and
  - **efficiency**.
- To do so, Prio uses SNIPs and aggregatable encodings.
- These techniques together bring private aggregation closer to practical.

Thank you!

https://crypto.stanford.edu/prio/

Henry Corrigan-Gibbs
henrycg@cs.stanford.edu

Example Encoding: Average and Variance

[PrivStats11]

Example Encoding: Average and Variance

[PrivStats11]

- Each of N clients holds a value x_i
- Servers want the AVG and VAR of the x_is.

Each client i encodes her value x as the pair

$$(x, y) = (x, x^2)$$

Simple to check that the encoding is valid:

$$\text{Valid}(x, y) = (x^2 - y)$$ [outputs zero if valid]

Example Encoding: Average and Variance

[PrivStats11]

- Each of N clients holds a value x_i
- Servers want the AVG and VAR of the x_is.

Each client i encodes her value x as the pair

$$(x, y) = (x, x^2)$$

Simple to check that the encoding is valid:

$$\text{Valid}(x, y) = (x^2 - y) \qquad \text{[outputs zero if valid]}$$

Use Prio to compute the sum of encodings $\sum_i (x_i, y_i)$

Example Encoding: Average and Variance

[PrivStats11]

- Each of N clients holds a value x_i
- Servers want the AVG and VAR of the x_is.

Each client i encodes her value x as the pair

$$(x, y) = (x, x^2)$$

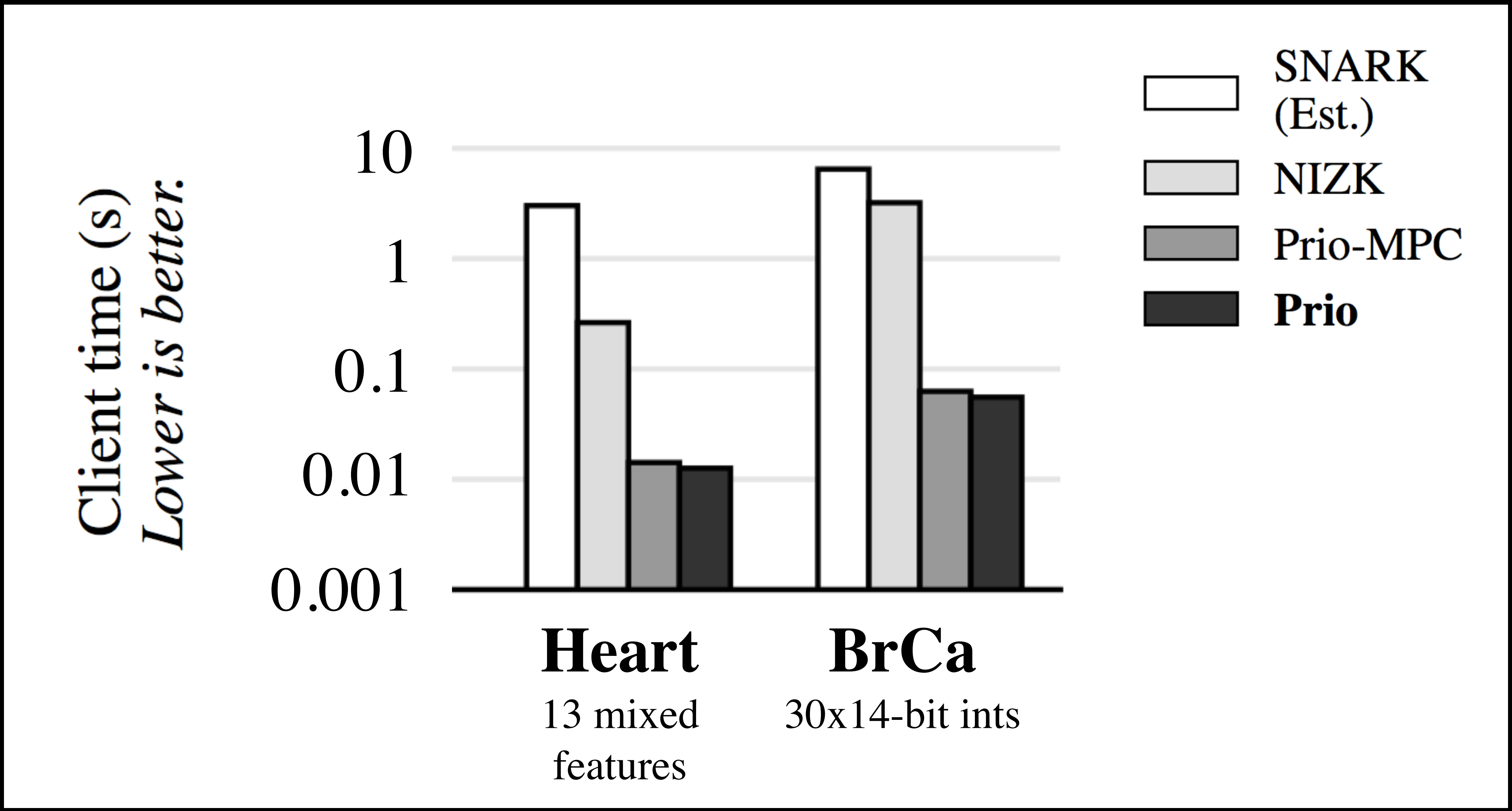
Simple to check that the encoding is valid:

$$\text{Valid}(x, y) = (x^2 - y) \qquad \text{[outputs zero if valid]}$$

Use Prio to compute the sum of encodings $\sum_i (x_i, y_i)$

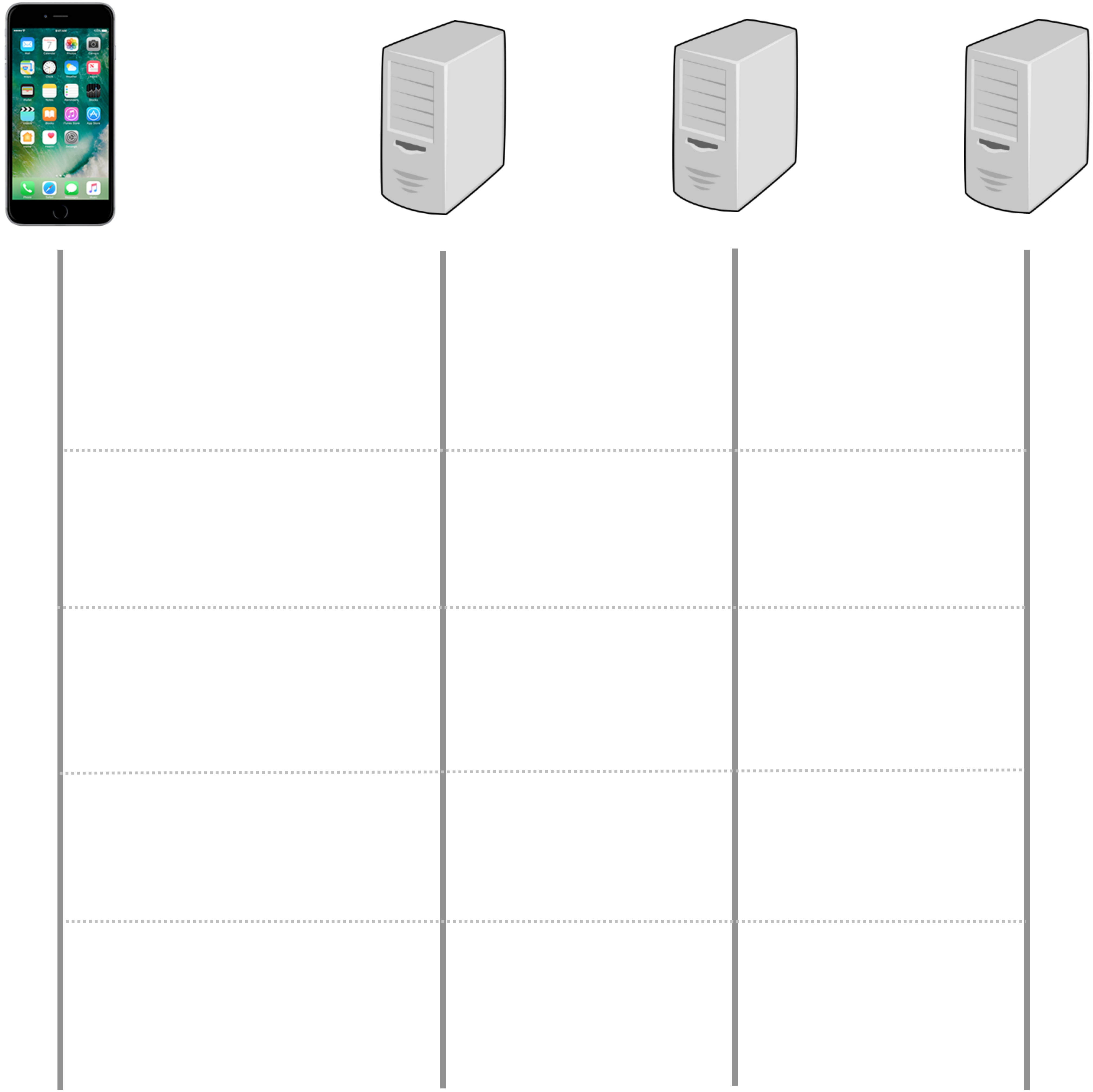
Then recover the statistics:

$$\begin{aligned} \text{AVG}(X) &= (\textstyle\sum_i x_i) \;/\; N \\ \text{AVG}(X^2) &= (\textstyle\sum_i y_i) \;/\; N \quad = (\textstyle\sum_i x_i^2) \;/\; N \\ \text{VAR}(X) &= \text{AVG}(X^2) - \text{AVG}(X)^2 \end{aligned}$$



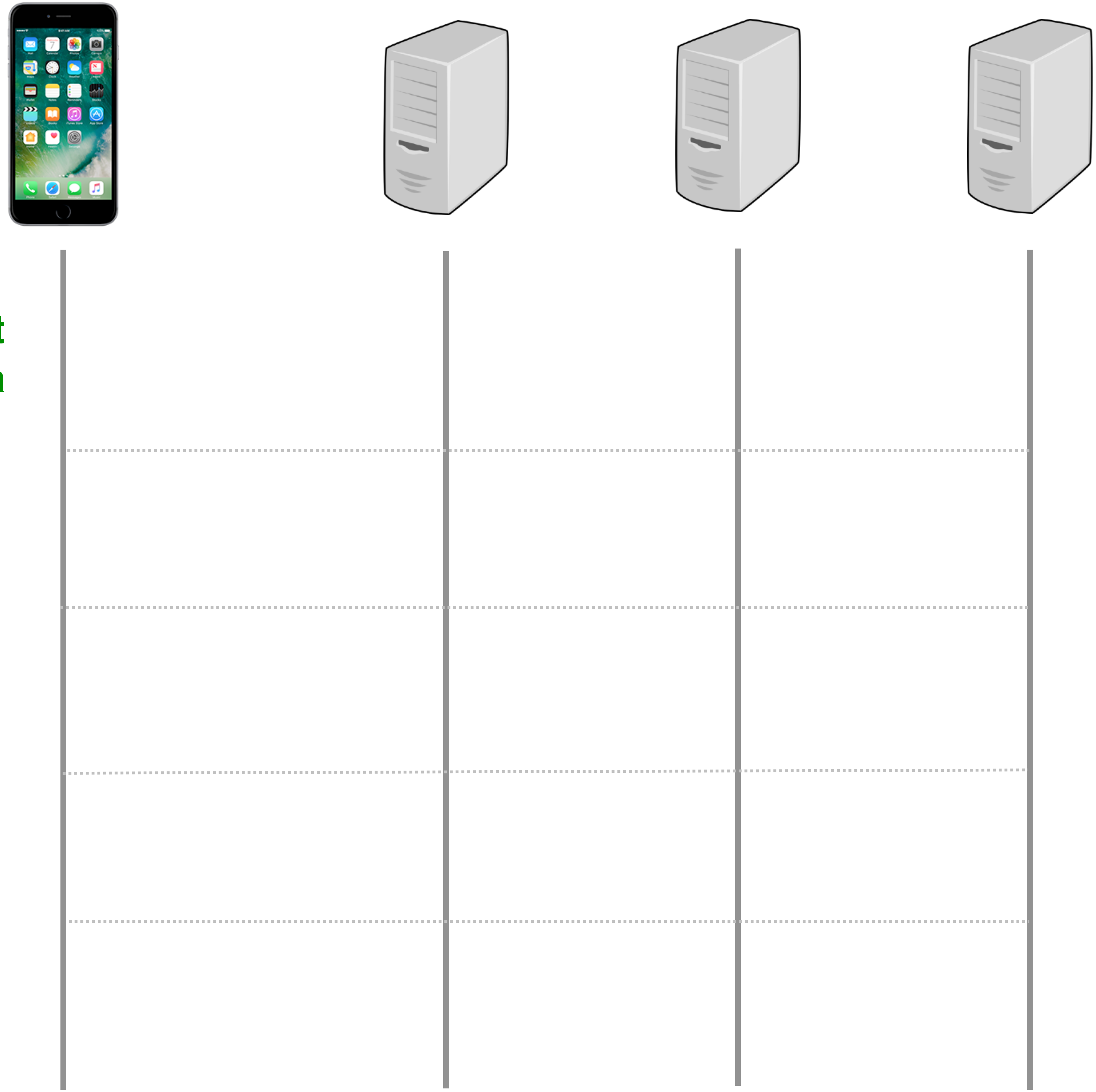
Client time (s)
Lower is better.
10
1
0.1
0.01
0.001
Heart
13 mixed features
BrCa
30x14-bit ints
SNARK (Est.)
NIZK
Prio-MPC
Prio

Using 128-bit integers

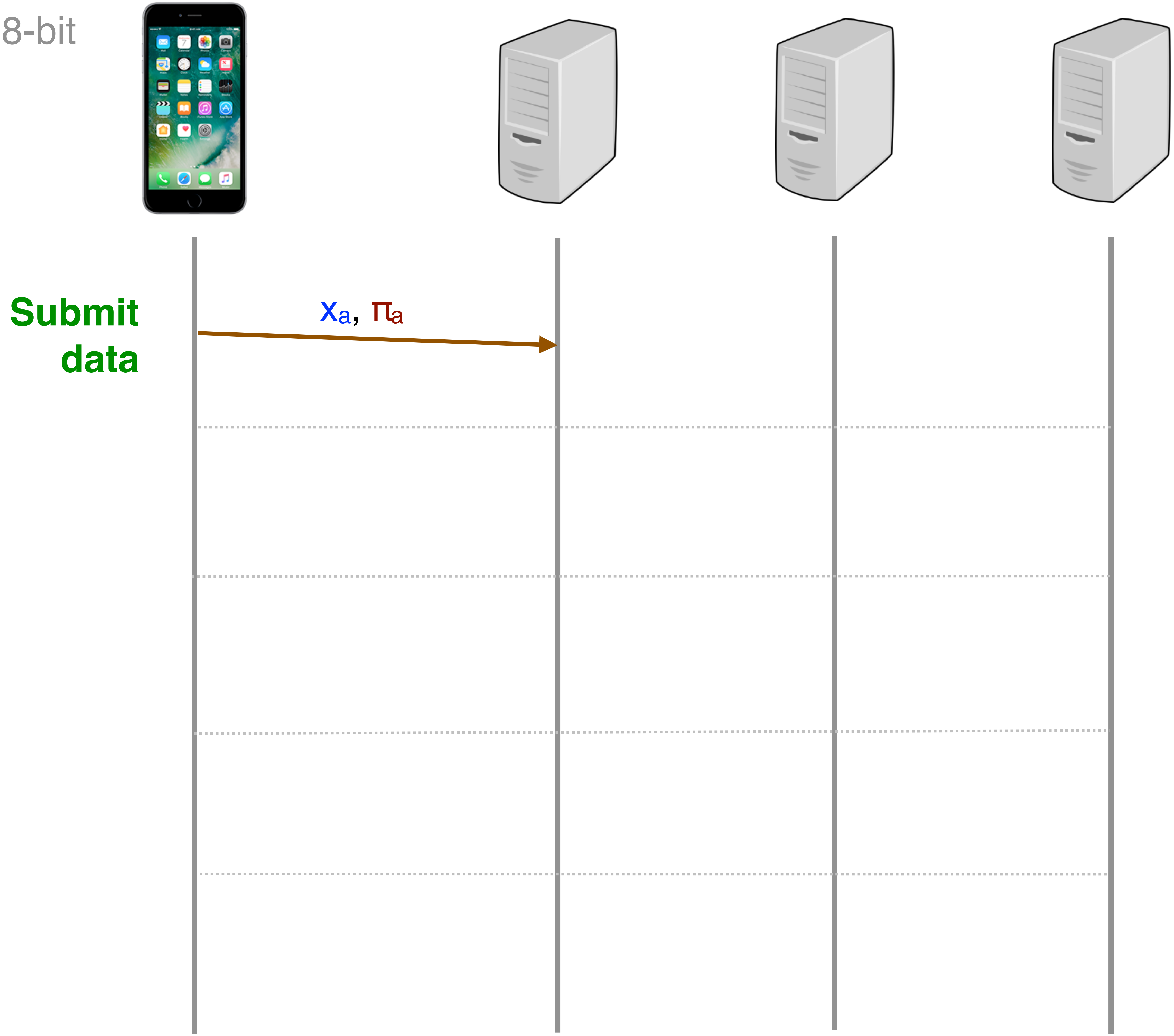


Using 128-bit integers

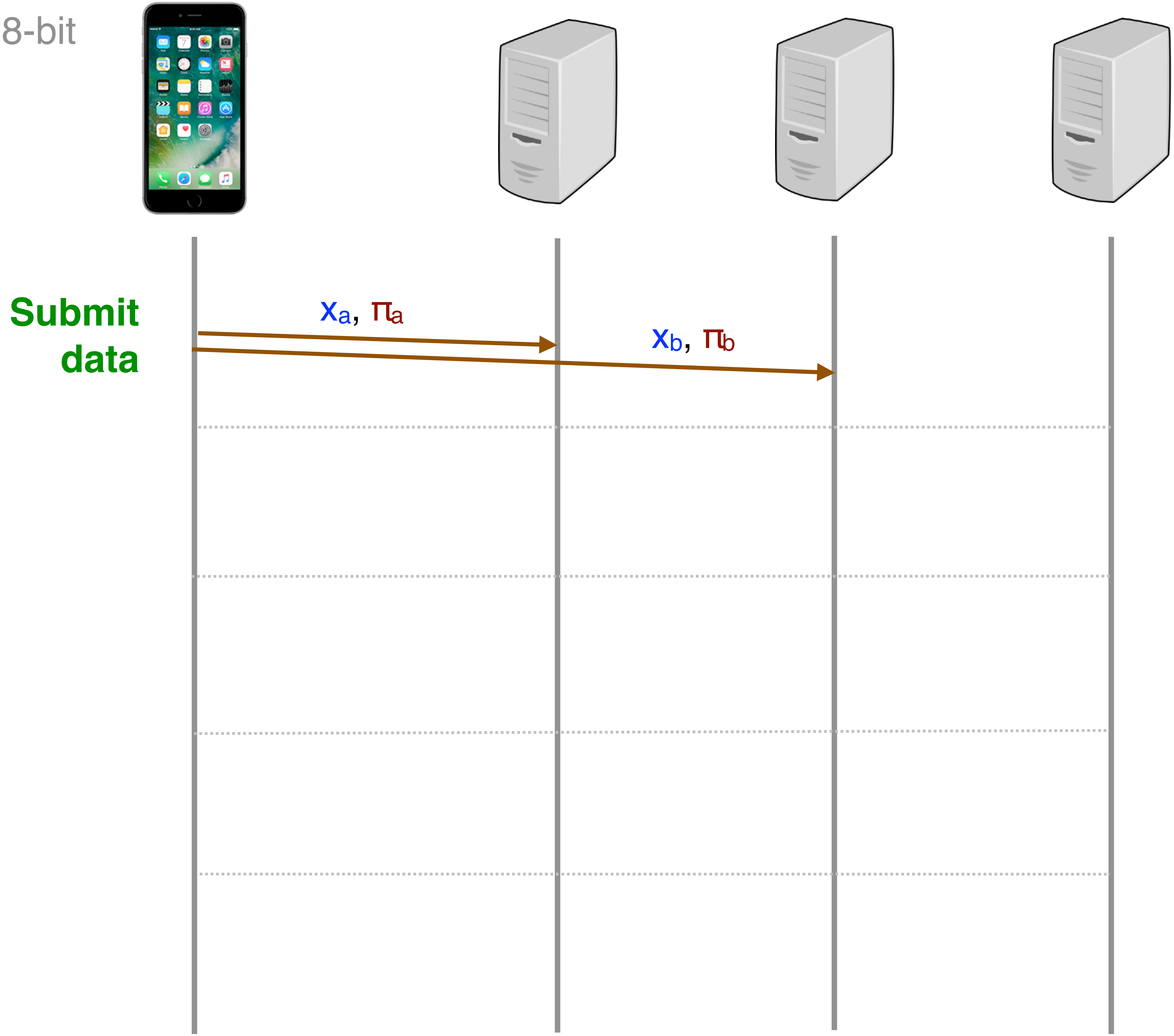
Submit data



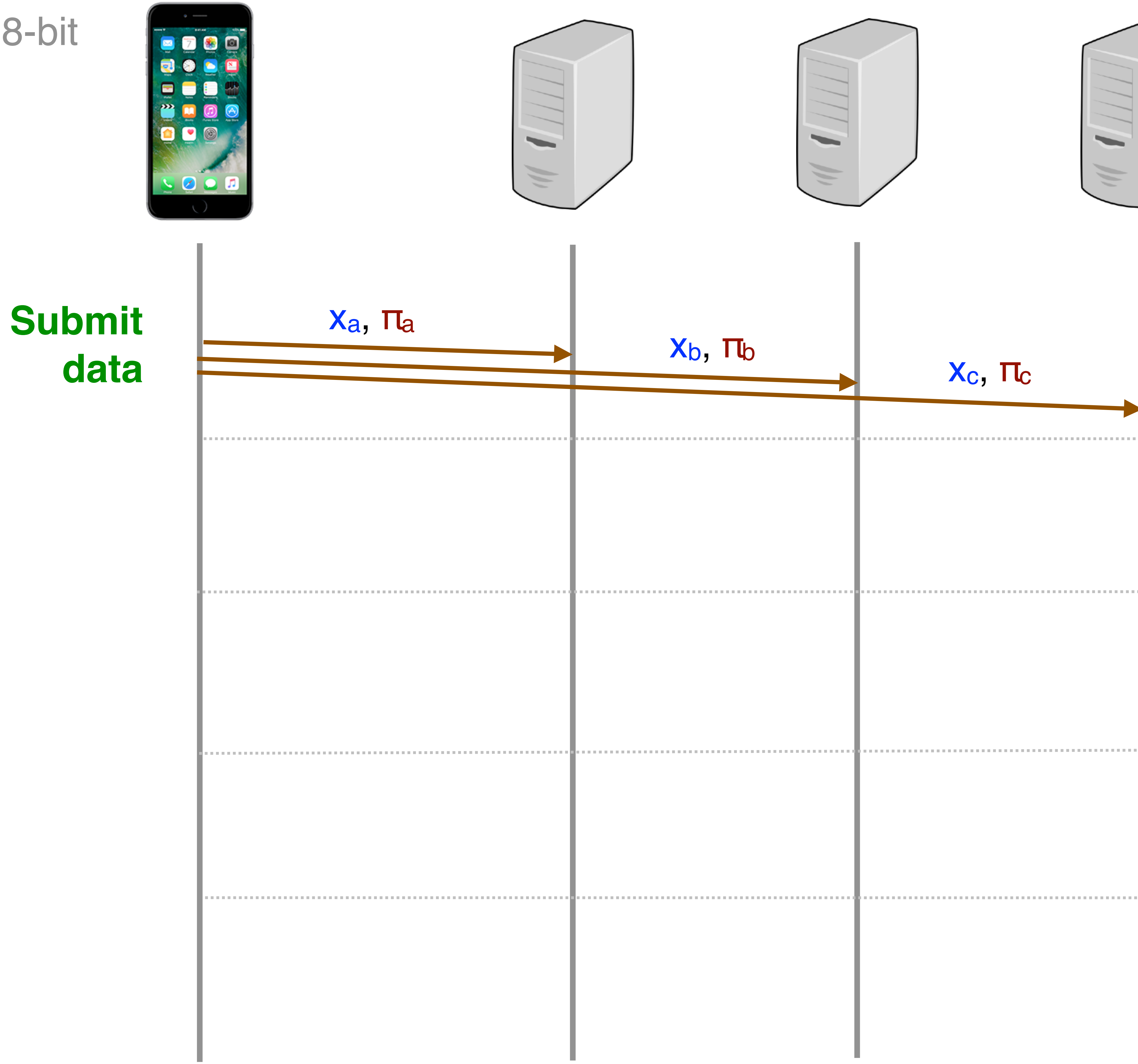
Using 128-bit integers



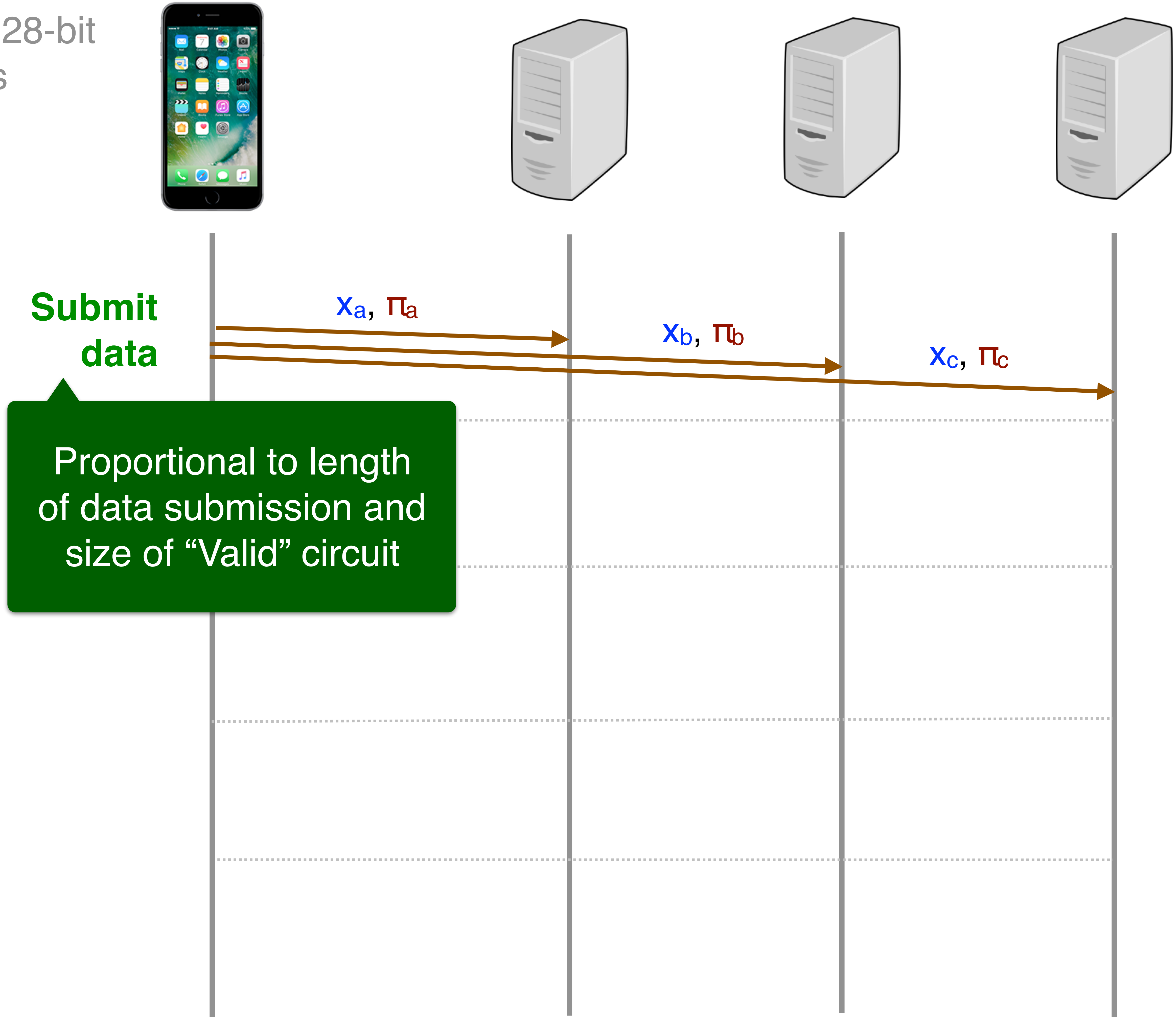
Using 128-bit integers



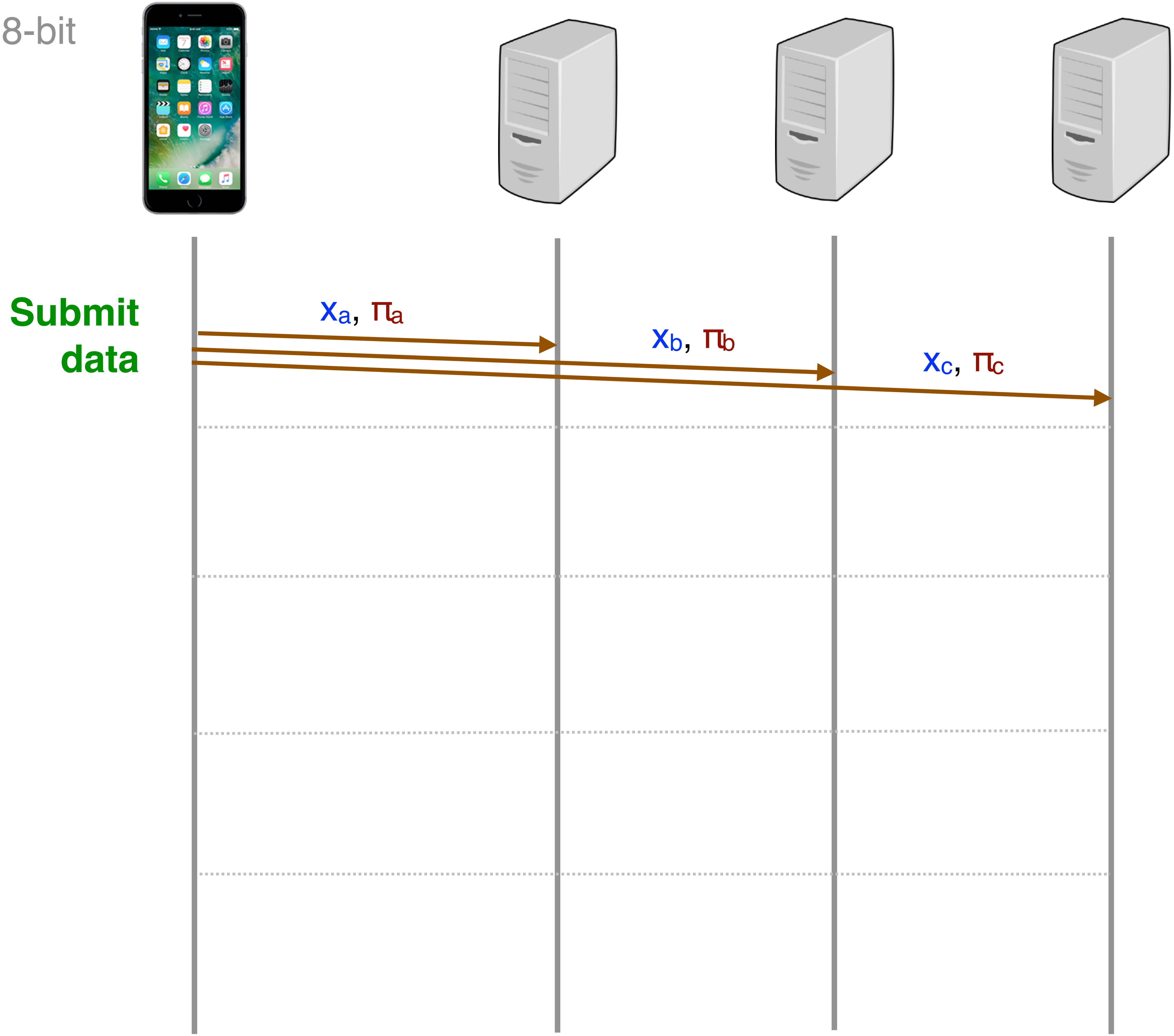
Using 128-bit integers



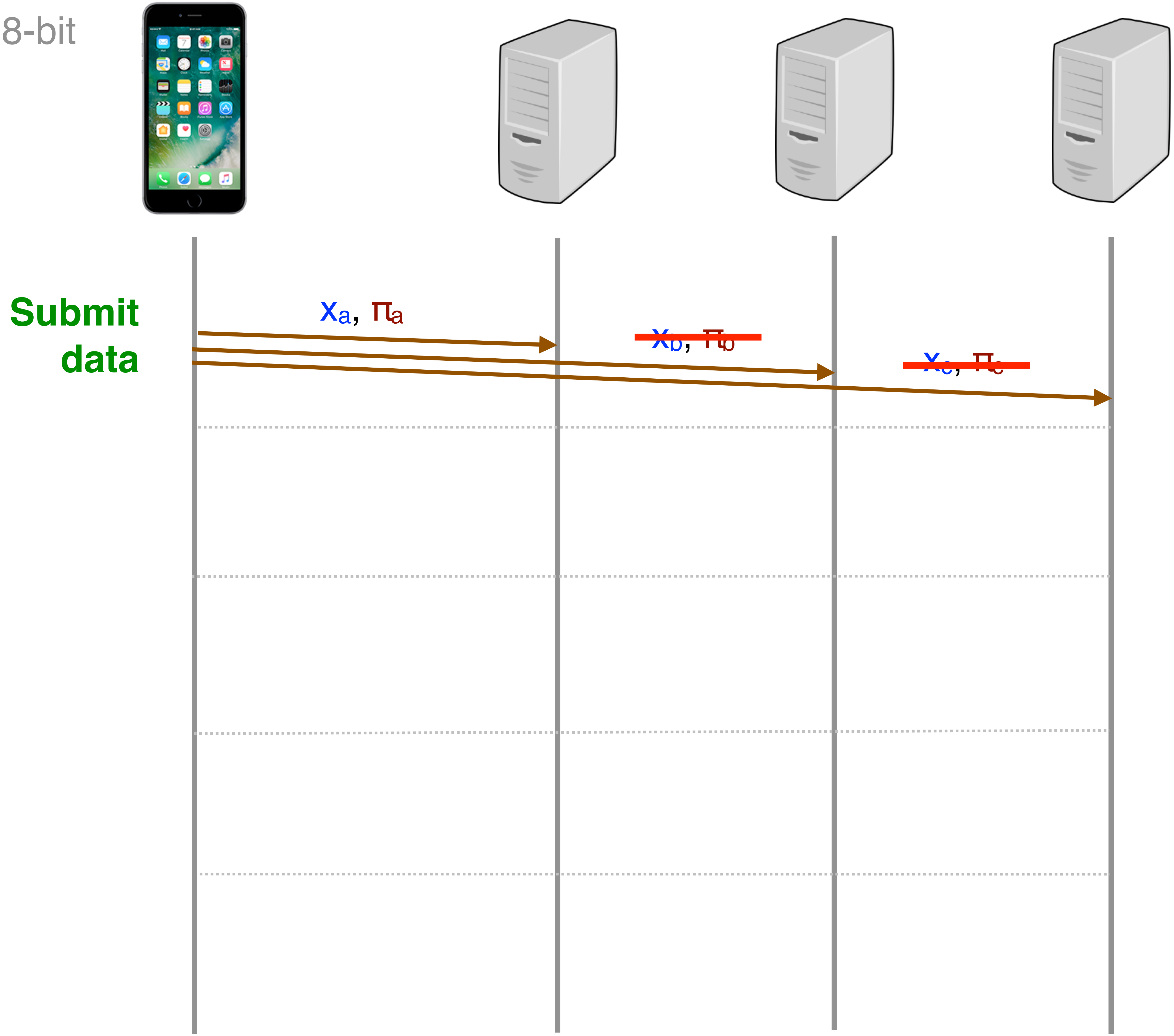
Using 128-bit integers



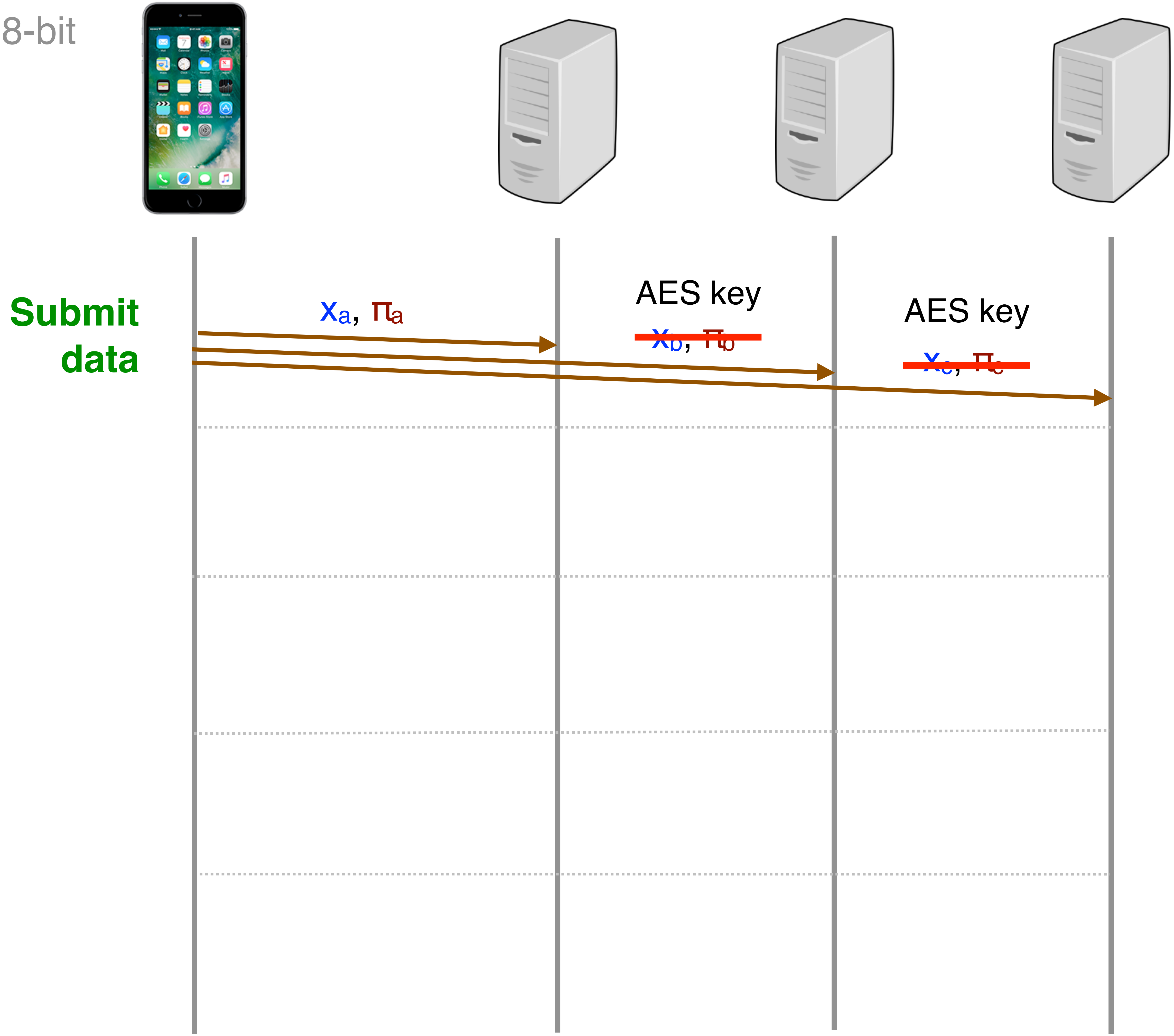
Using 128-bit integers



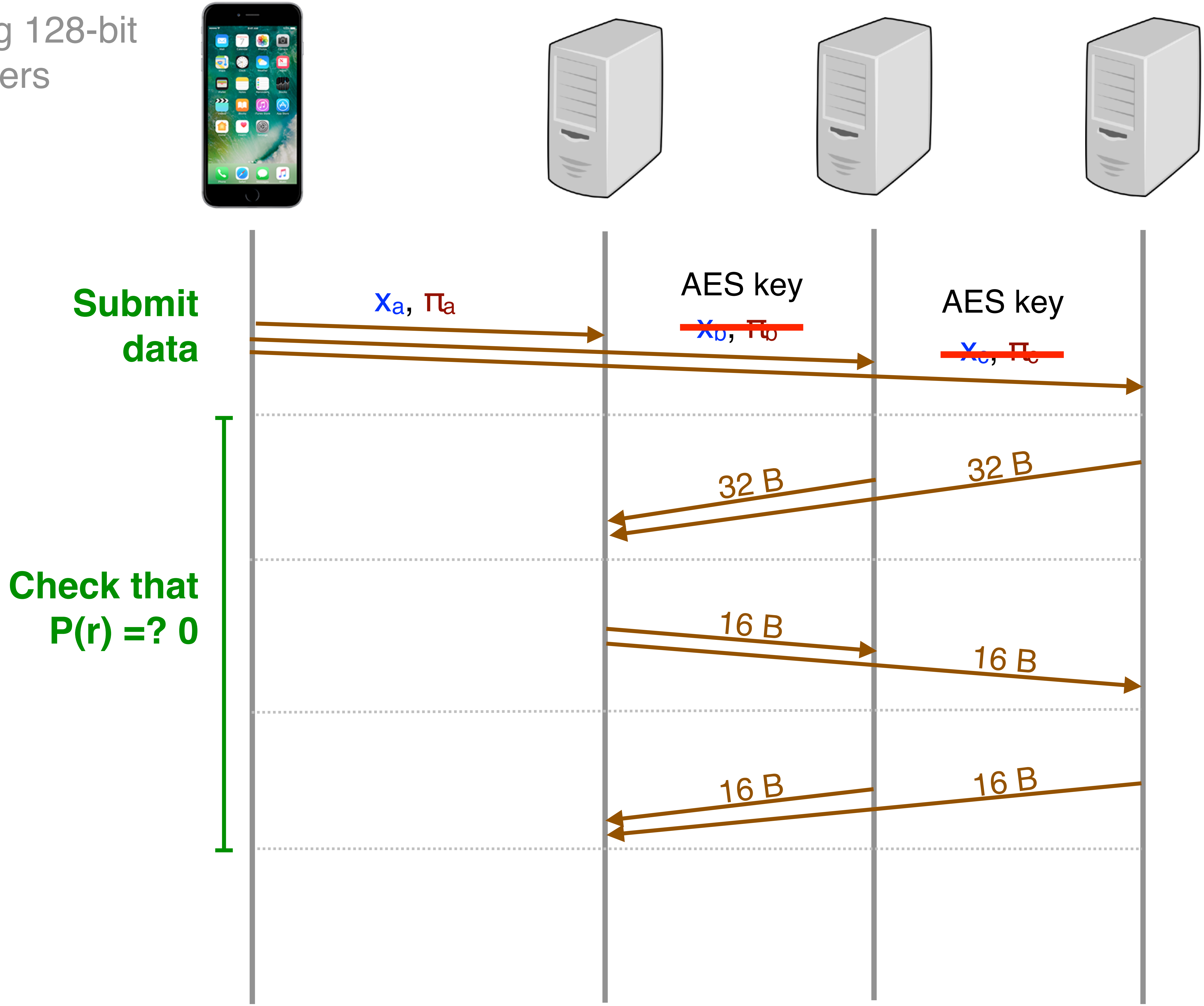
Using 128-bit integers



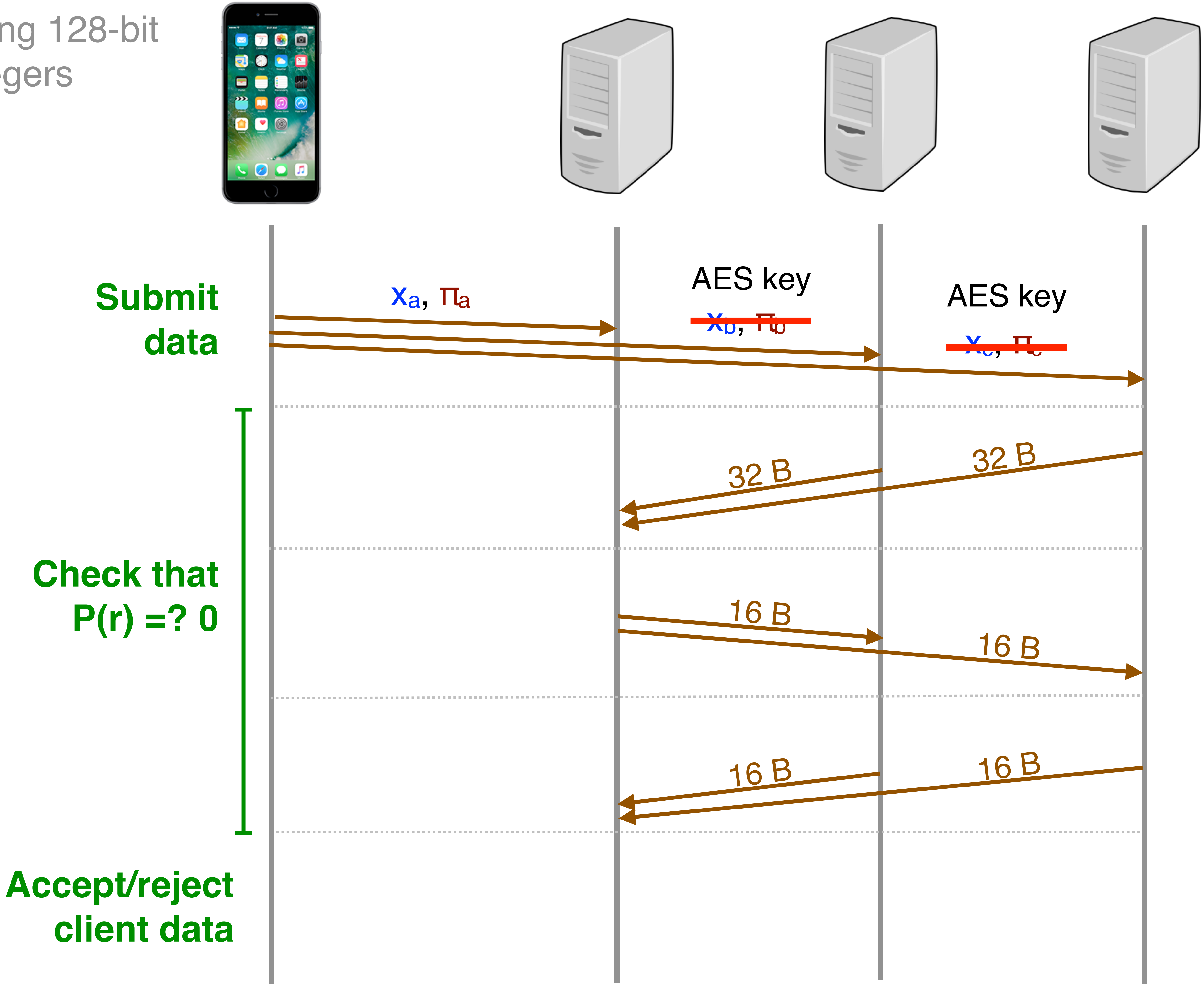
Using 128-bit integers



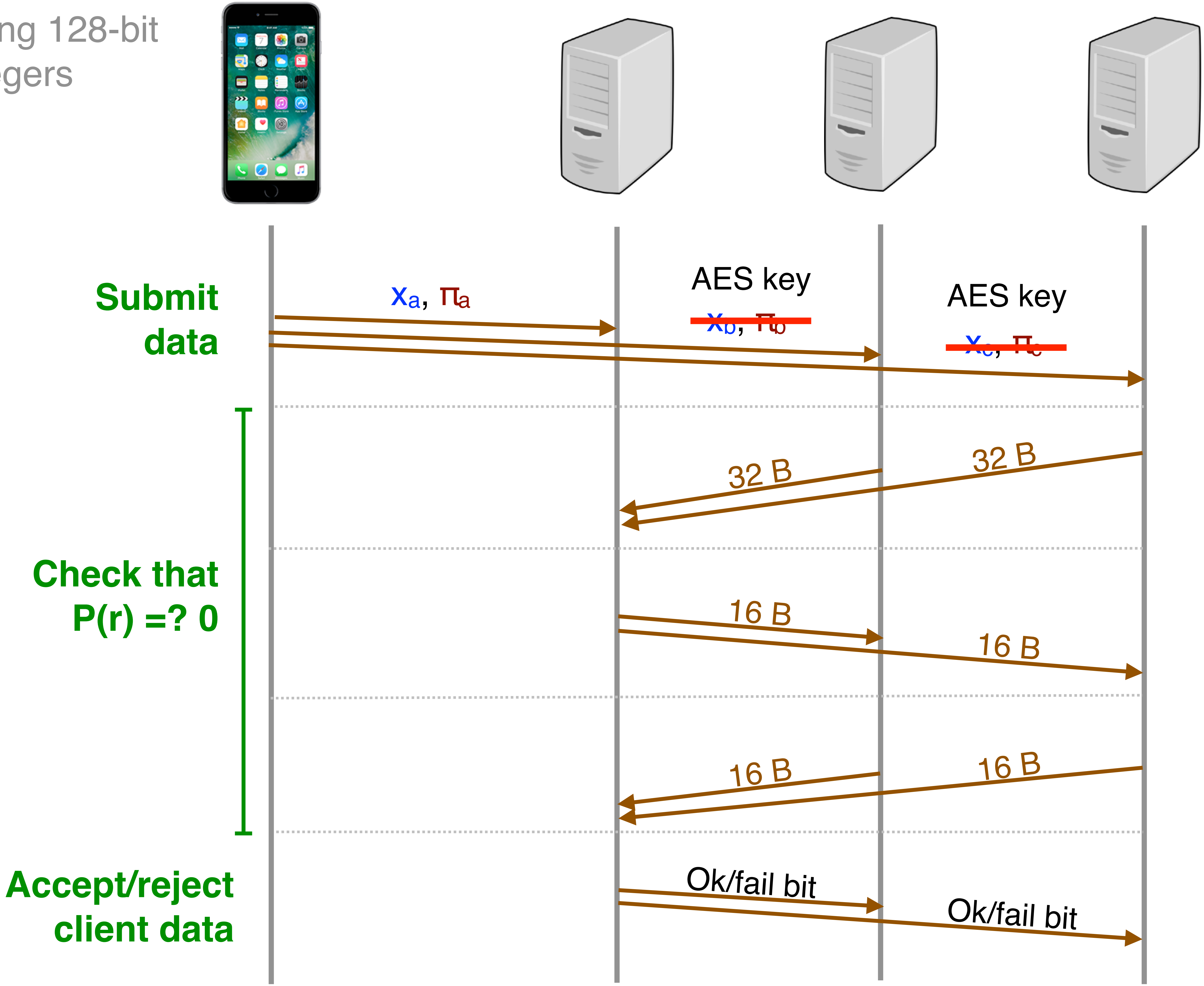
Using 128-bit integers



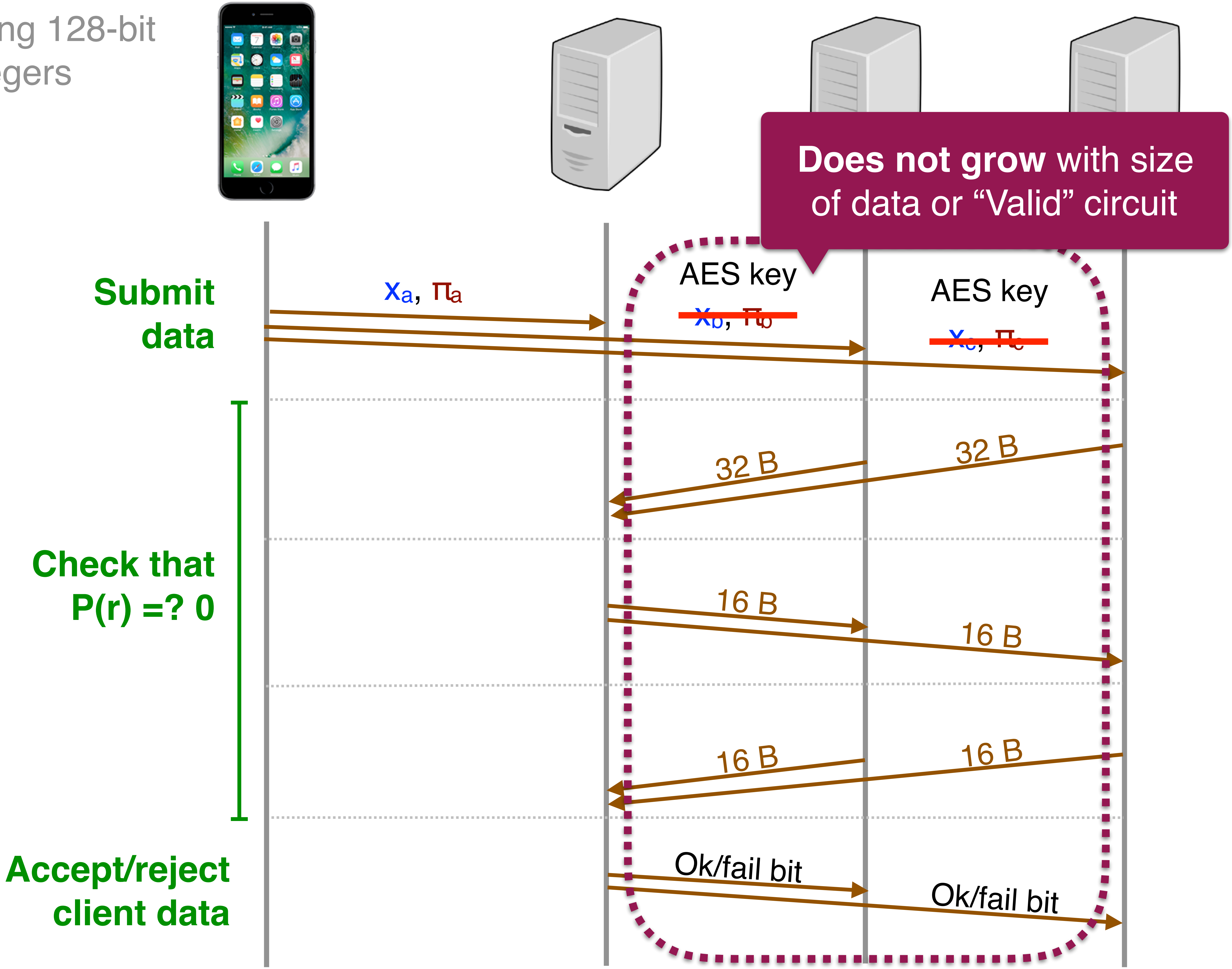
Using 128-bit integers



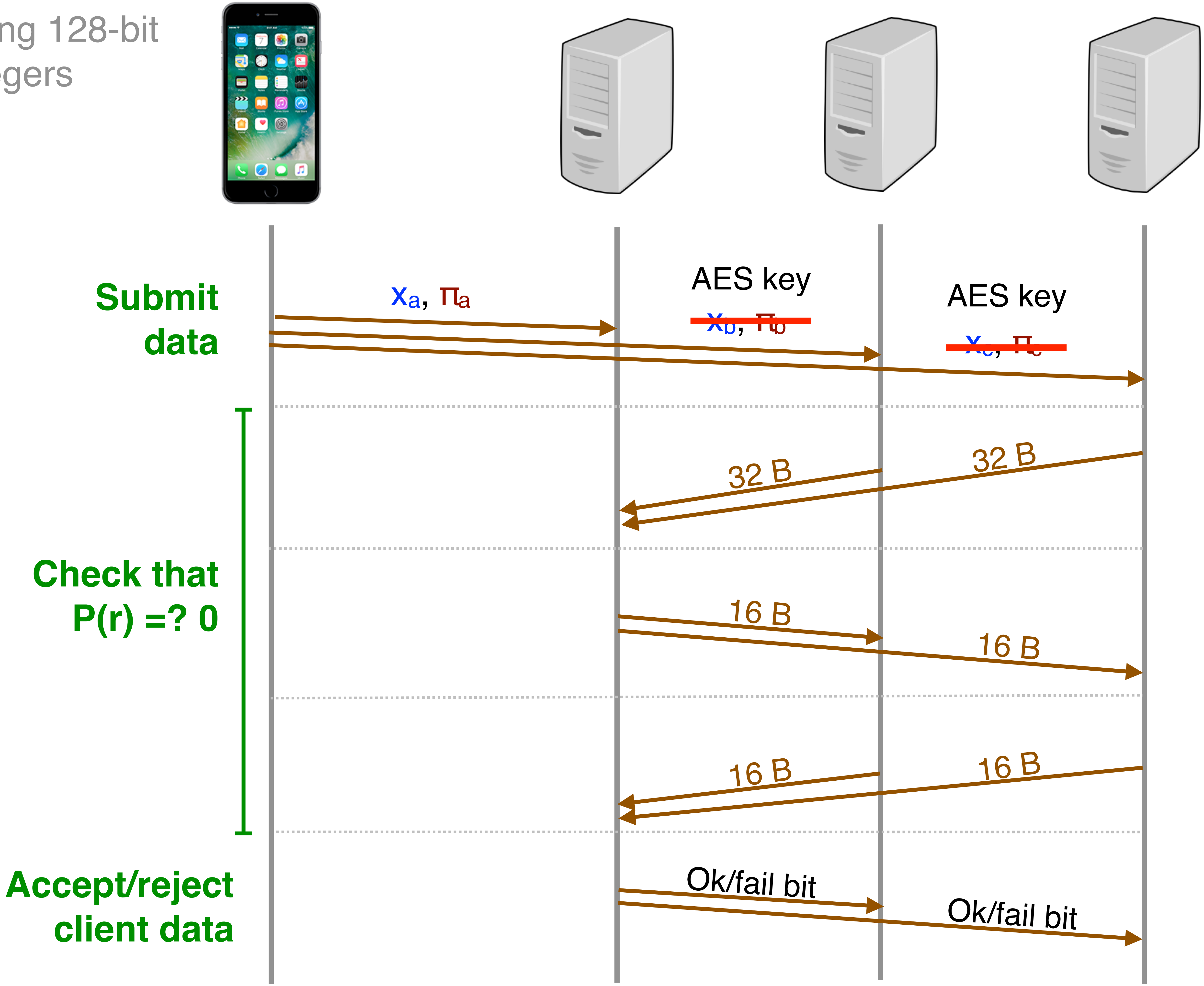
Using 128-bit integers



Using 128-bit integers



Using 128-bit integers



Example Encoding: Average and Variance

[PrivStats11]

Example Encoding: Average and Variance

[PrivStats11]

- Each of N clients holds a 4-bit value x_i
- Servers want the AVG and VAR of the x_is.

Each client encodes her value $x = b_3b_2b_1b_0$ as the tuple

$$(x, y) = (x, x^2, b_3, b_2, b_1, b_0)$$

Example Encoding: Average and Variance

[PrivStats11]

- Each of N clients holds a 4-bit value x_i
- Servers want the AVG and VAR of the x_is.

Each client encodes her value $x = b_3 b_2 b_1 b_0$ as the tuple

$$(x, y) = (x, x^2, b_3, b_2, b_1, b_0)$$

To test validity of the encoding, check that:

$$\mathrm{Valid}(x, y) = \begin{cases} (x^2 - y) = 0 & \text{— y is } x^2 \\ x - \sum_j 2^j b_j = 0 & \text{— b's are the bits of x} \\ b_j \cdot (b_j - 1) = 0 & \text{— b's are 0/1 values} \end{cases}$$