Single-Server Private Information Retrieval and Secure Aggregation in the Shuffle Model

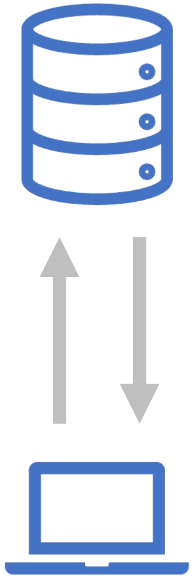
Adrià Gascón Yuval Ishai Mahimna Kelkar

Baiyu Li Yiping Ma Mariana Raykova

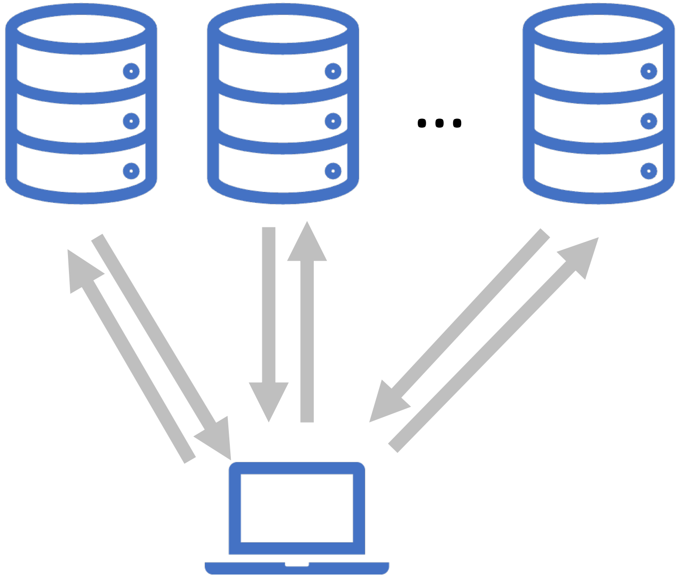
Google Technion Israel Institute of Technology Penn University of Pennsylvania Cornell University Founded A.D. 1865 CORNELL TECH

The two worlds of PIR

Single-server PIR



(Non-colluding) Multi-server PIR



The two worlds of PIR

Single-server PIR

DCR, DDH, …

Plain model
Early works: [KO97], [CMS99], [GR05], ...
Recent: XPIR, SEAL-PIR, Spiral, WhisPIR, …

(R)LWE

Making the server fast

[BIM04]

Preprocessing
[PPY18], SimplePIR, FrodoPIR, Piano, HintlessPIR, …

(Non-colluding) Multi-server PIR

Deployment cost

Plain model
Early works: from coding theory, e.g., [CGKS95]
Recent: DPF-based, efficient construction from [BGI16]

Concretely fast

Preprocessing
[BIM04], [CK20], [KC21], [ISW24], …

The two worlds of PIR

Single-server PIR

DCR, DDH, …

<u>Plain model</u>
Early works: [KO97], [CMS99], [GR05], ...
Recent: XPIR, SEAL-PIR, Spiral, WhisPIR, …

(R)LWE

Making the server fast

[BIM04]

<u>Preprocessing</u>
[PPY18], SimplePIR, FrodoPIR, Piano, HintlessPIR, …

(Non-colluding) Multi-server PIR

Deployment cost

<u>Plain model</u>
Early works: from coding theory, e.g., [CGKS95]
Recent: DPF-based, efficient construction from [BGI16]

Concretely fast

☹ Unfriendly to database updates, big client storage, still fall behind the fast multi-server PIR, …

Can we have a single-server PIR that is as fast as multi-server PIR?

Best of both worlds?

- Revisiting the shuffle model gives us ideas
  - Common in differential privacy literature
  - Proposed in [IKOS06] in the context of PIR (but later didn't receive much attention)
- The shuffle model assumption
  - Many clients make queries simultaneously
  - The queries are shuffled before reaching the server

Best of both worlds?

- Revisiting the shuffle model gives us ideas
  - Common in differential privacy literature
  - Proposed in [IKOS06] in the context of PIR (but later didn't receive much attention)
- The shuffle model assumption
  - Many clients make queries simultaneously
  - The queries are shuffled before reaching the server

Many applications naturally has the feature

Studied in many works on anonymity

Best of both worlds? Yes, in the shuffle model

- Prior results:
  - [IKOS06]: a construction based on non-standard computational assumption
  - [IKLM24]: a class of constructions for IT-secure PIR with sublinear communication

Both works are not concretely efficient

- This work: single-server PIR in the shuffle model that matches the (server) cost of a very fast multi-server PIR

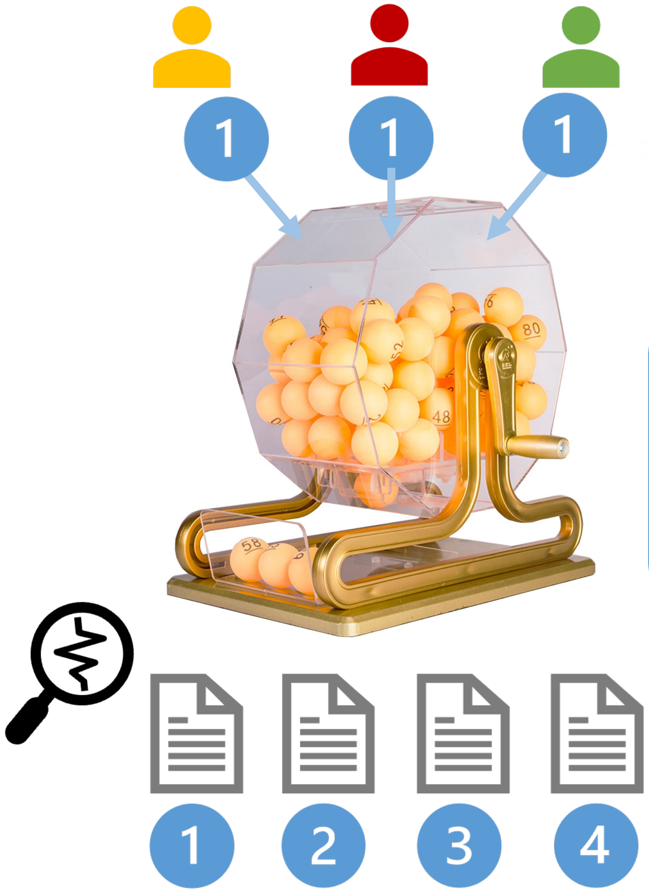
This work

- A construction for single-server PIR in the shuffle model
  - Based on LPN and conjectured MDSD hardness
  - 9—25X improvement on throughput over SimplePIR [HHCG^{+}], depending on parameters
- Computationally secure aggregation in the shuffle model
  - Same assumptions as the PIR construction
  - 25X savings for communication compared to existing best statistical scheme [BBGN20]

Outline

- Shuffle PIR: the security goal
- The "split-and-mix" paradigm
- Our constructions
  - Shuffle PIR based on LPN, MDSD
  - By-product: computationally secure aggregation in the shuffle model
- Discussion and open questions

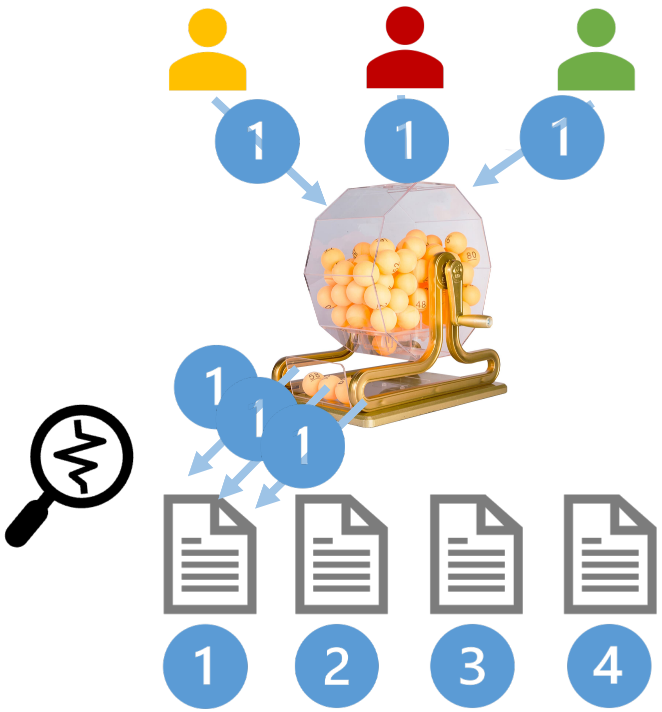
PIR in the shuffle model: the security goal

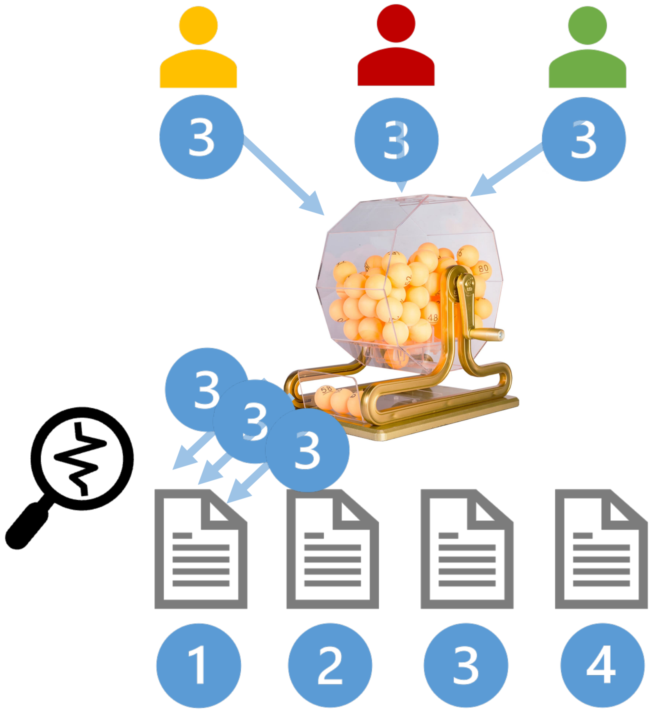


Anonymity does not imply message privacy:
It hides who sends what,
but does not hide the messages themselves

PIR in the shuffle model: the security goal

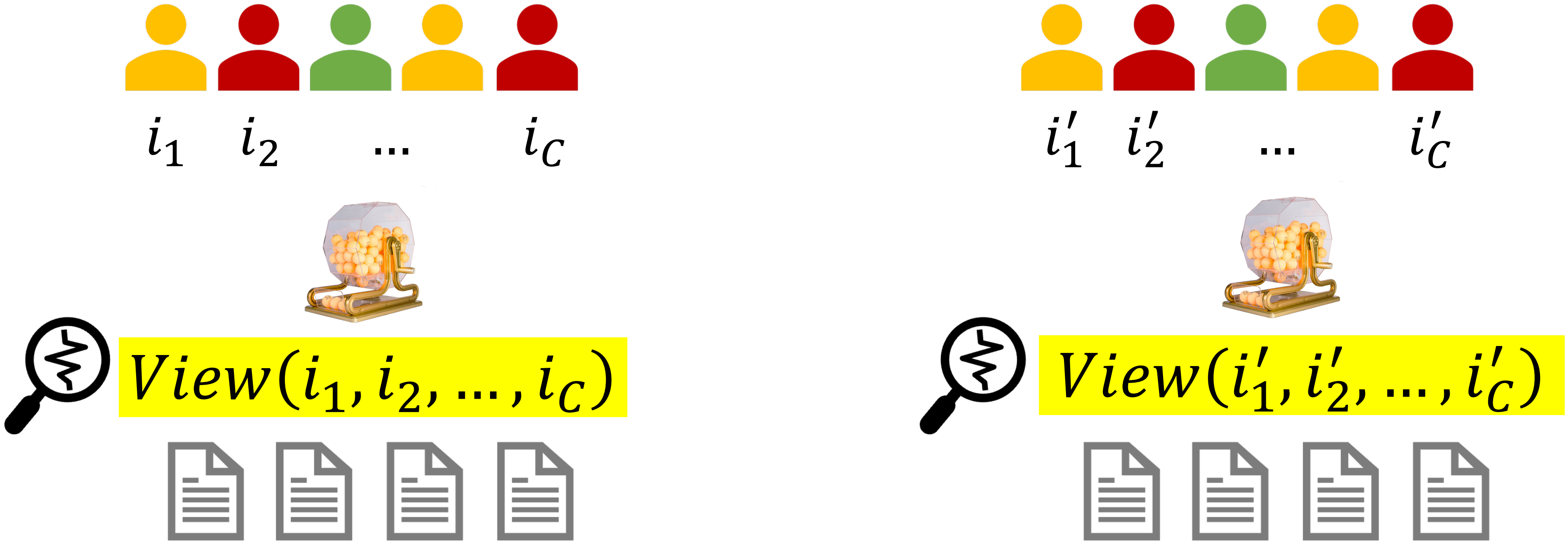
- Anonymization does not trivialize the PIR problem!





PIR in the shuffle model: the security goal

- Anonymization does not trivialize the PIR problem!



Outline

- Shuffle PIR: the security goal
- The “split-and-mix” paradigm
- Our constructions
  - Shuffle PIR based on LPN, MDSD
  - By-product: computationally secure aggregation in the shuffle model
- Discussion and open questions

The split-and-mix paradigm

- Privacy from anonymity [IKOS06]: computing sum

5 1 0 3 8

Take a large enough p, each client splits its inputs into k shares in $\mathbb{Z}_p$

The split-and-mix paradigm

- Privacy from anonymity [IKOS06]: computing sum

5 1 0 3 8

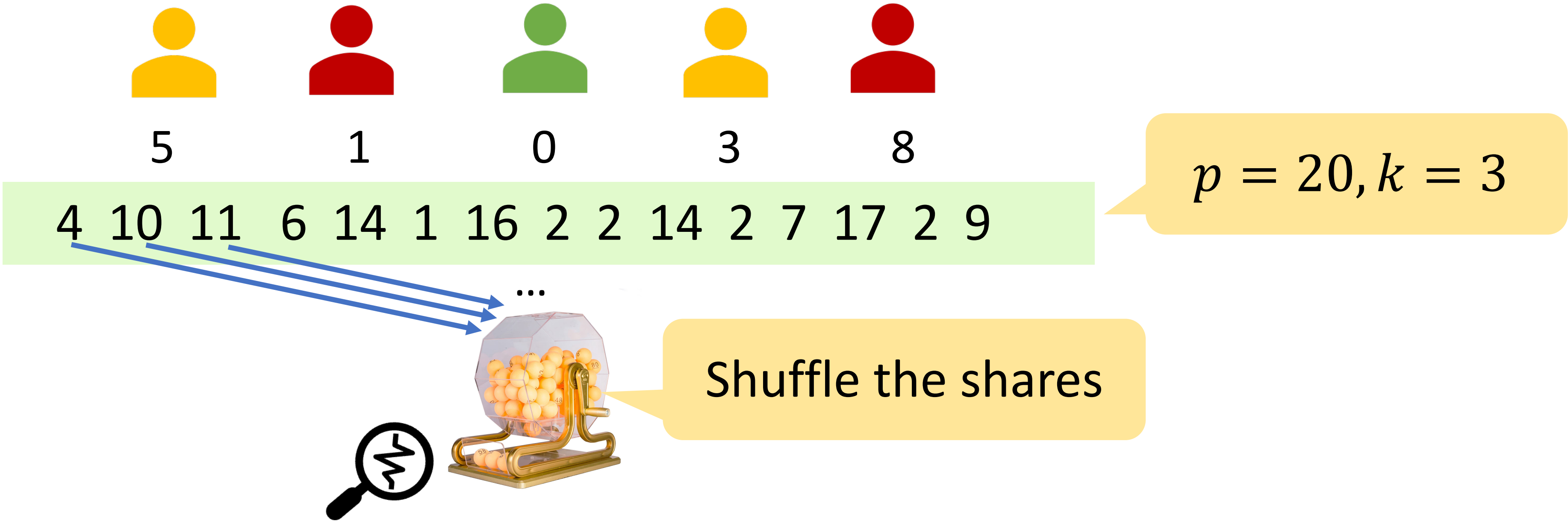
4+10+11 6+14+1 16+2+2 14+2+7 17+2+9

$p = 20, k = 3$

…

The split-and-mix paradigm

- Privacy from anonymity [IKOS06]: computing sum



The shuffled shares leak nothing about the inputs except the sum

The split-and-mix paradigm

- Privacy from anonymity [IKOS06]: computing sum

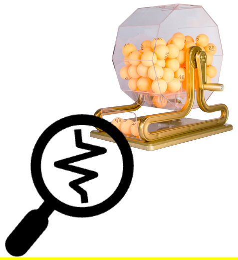
10 2 2 1 1

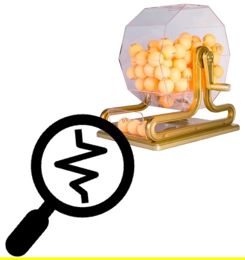
Split into shares

4 4 4 4 0

Split into shares

Any two different configurations with equal sum

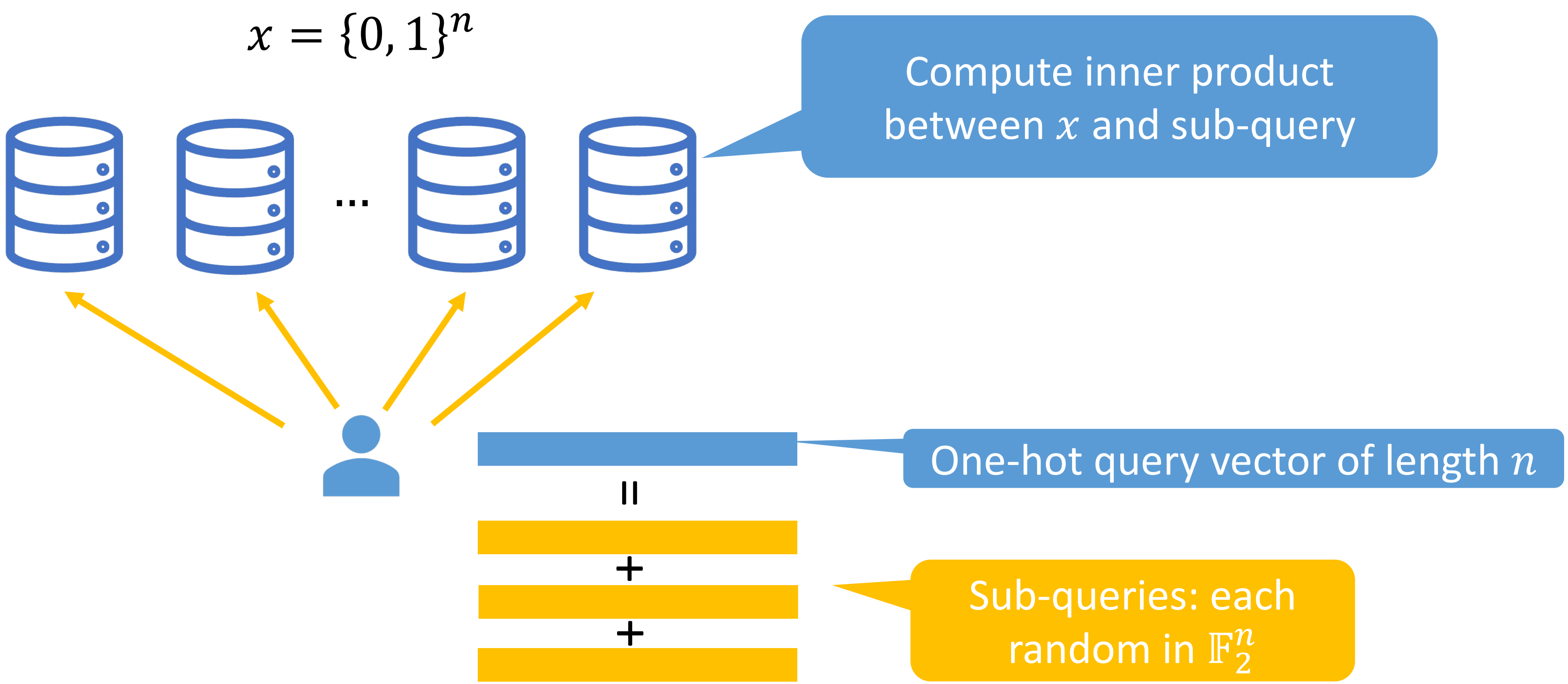




$$View(10, 2, 2, 1, 1) \approx View(4, 4, 4, 4, 0)$$

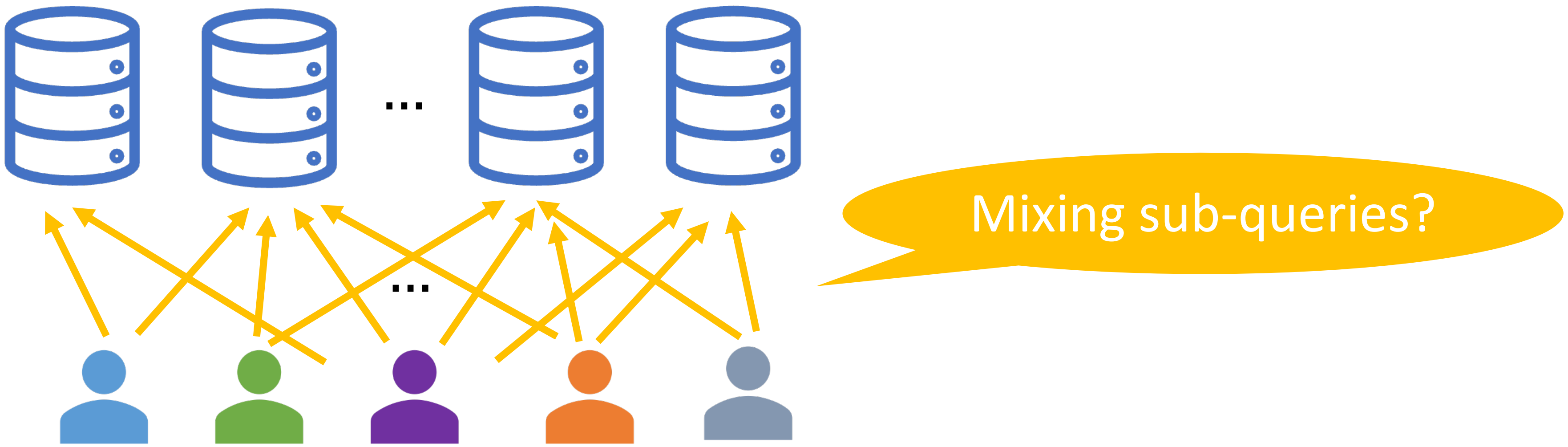
Statistical/computational

Split-and-mix for PIR



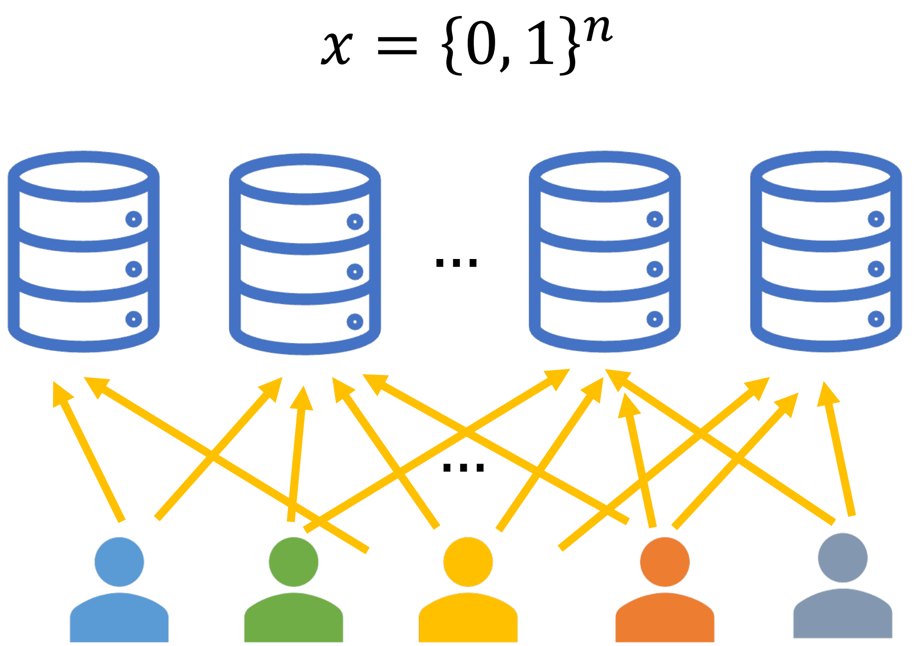
Split-and-mix for PIR

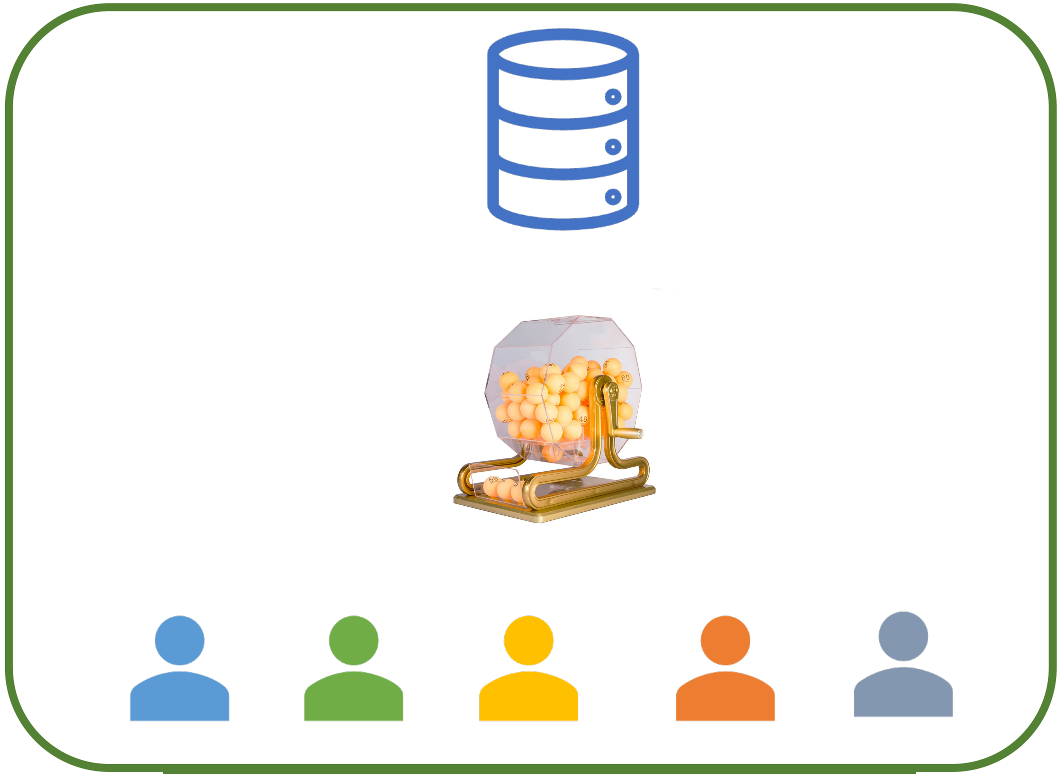
$$x = \{0,1\}^n$$



Split-and-mix for PIR

"PIR in the secret sharing paradigm": view the sub-queries as shares, then mix the sub-queries





Our construction

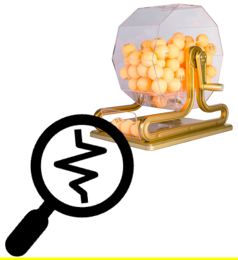
Split-and-mix for PIR

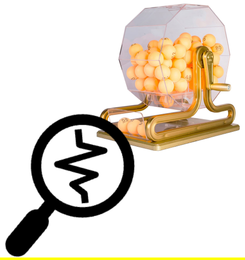
"PIR in the secret sharing paradigm": view the sub-queries as shares, then mix the sub-queries

10 2 2 1 1 4 4 4 4 0

Any two different set of query indices with equal "sum"

Each input is split to k sub-queries





$View(10, 2, 2, 1, 1) \approx View(4, 4, 4, 4, 0)$

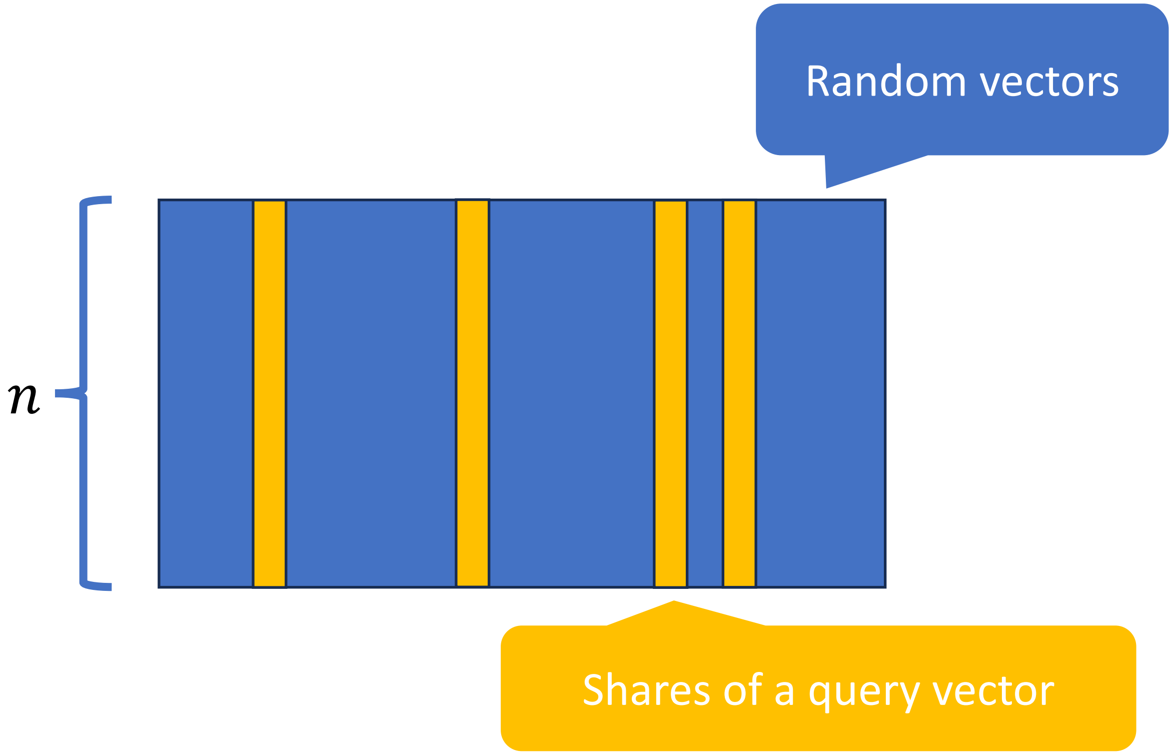
This work: computational

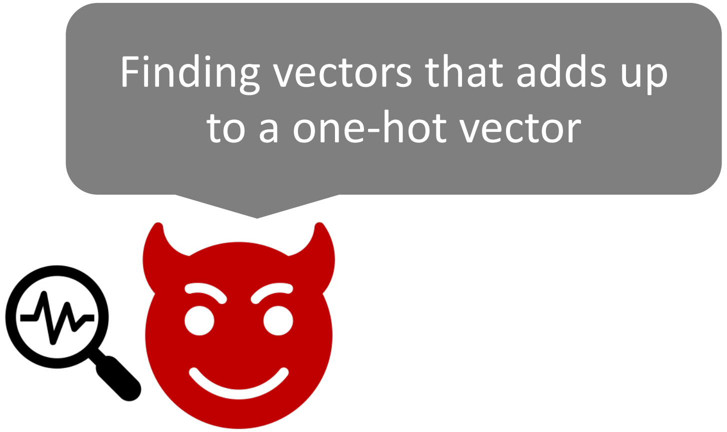
Outline

- Shuffle PIR: the security goal
- The "split-and-mix" paradigm
- Our constructions
  - Shuffle PIR based on LPN, MDSD
  - By-product: computationally secure aggregation in the shuffle model
- Discussion and open questions

Our construction: security analysis

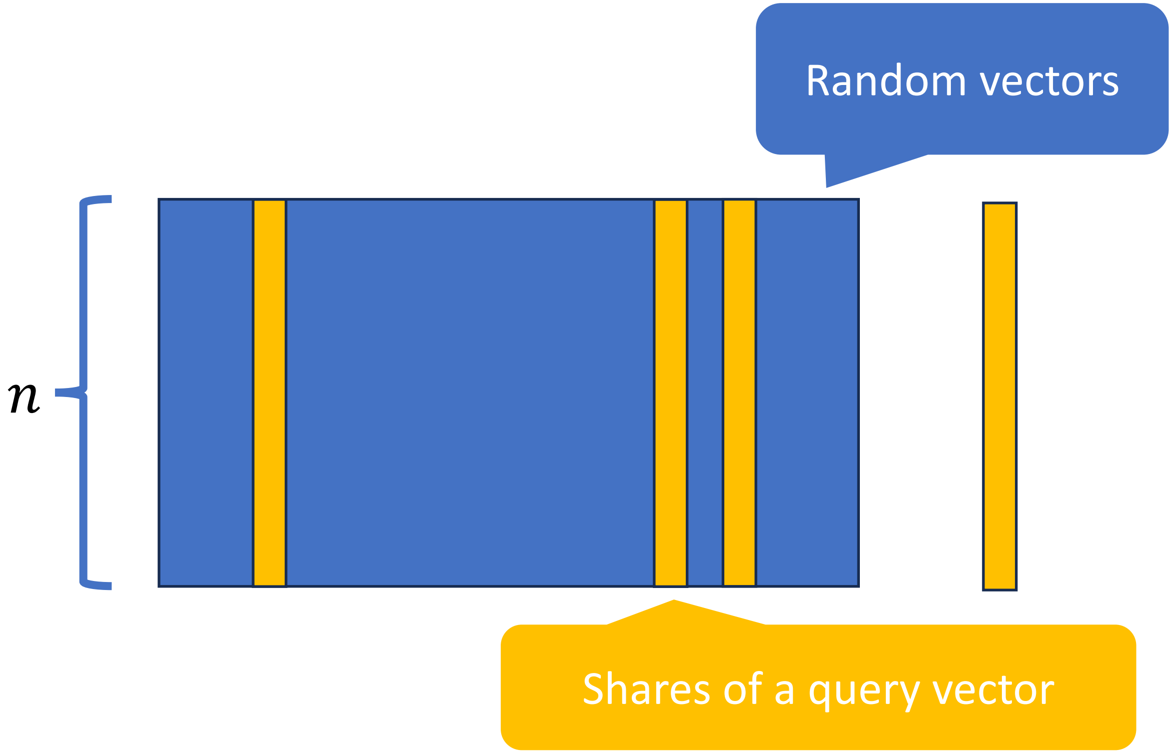
- Relevant computational assumptions: k-sum, syndrome decoding (dual-LPN)





Our construction: security analysis

- Let’s do a slight modification…



Our construction: security analysis

- Decisional syndrome decoding (dual LPN) [BFKL94, AIK07]

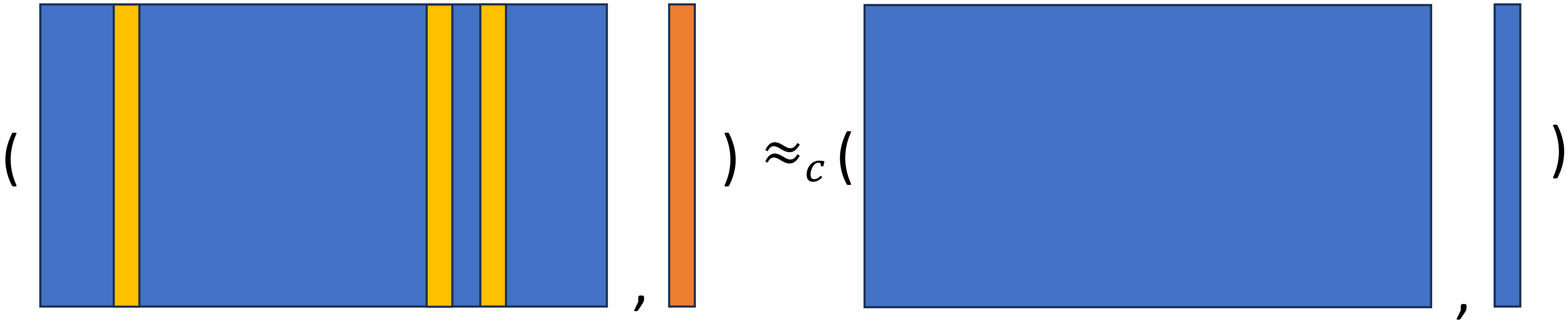
A random matrix

A low-weight random vector

A random vector

$$(H, H \cdot e) \approx_c (H, r)$$

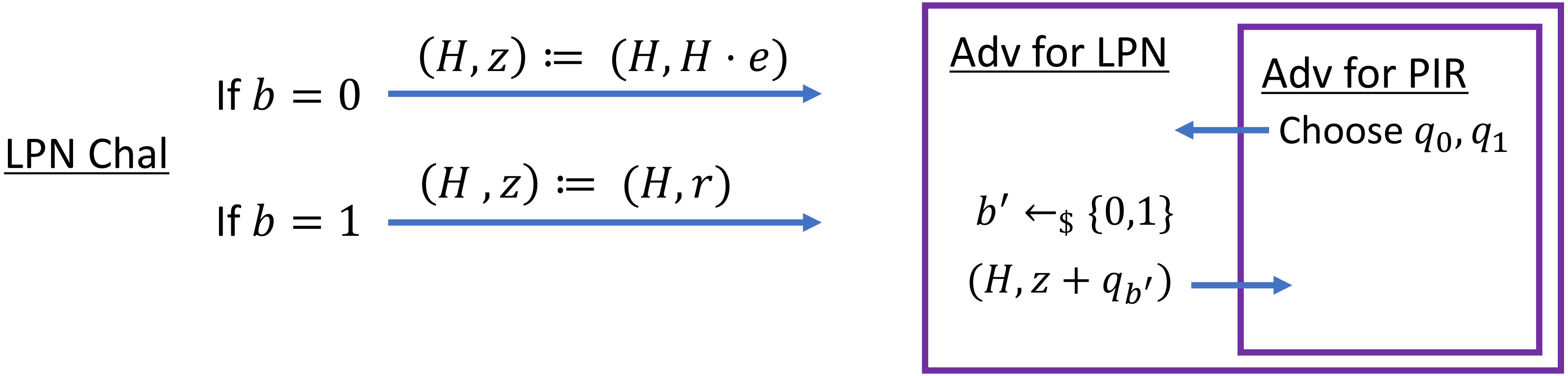
Everything in $\mathbb{F}_2$



Reducing to LPN

- Considering a single client
- $H \cdot e + y = q,$

  where $q \in \mathbb{F}_2^n, H \in \mathbb{F}_2^{n \times m}$ and $e \in \mathbb{F}_2^m$ with Hamming weight k



Conjectured MDSD security

- When there are c clients
- $H \cdot E + Y = Q$, where E is a “structured” matrix, and $Q \in \mathbb{F}_2^{n \times c}$

Multi-Disjoint Syndrome Decoding (MDSD)

A random matrix

A structured matrix with low-weight columns

A random matrix

$$(H, H \cdot E) \approx_c (H, R)$$

H has shares and dummies

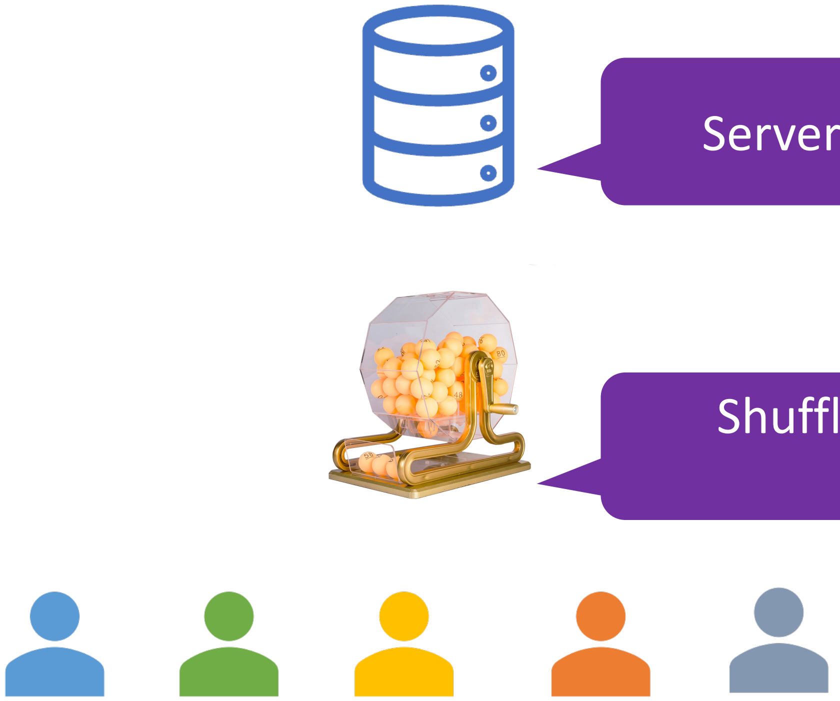
Shuffle PIR: the computational construction

Server answers to each share (inner product)

Shuffler permutes all the shares

Split query vector into k (real) shares, add d dummy shares

Shuffle PIR: the computational construction



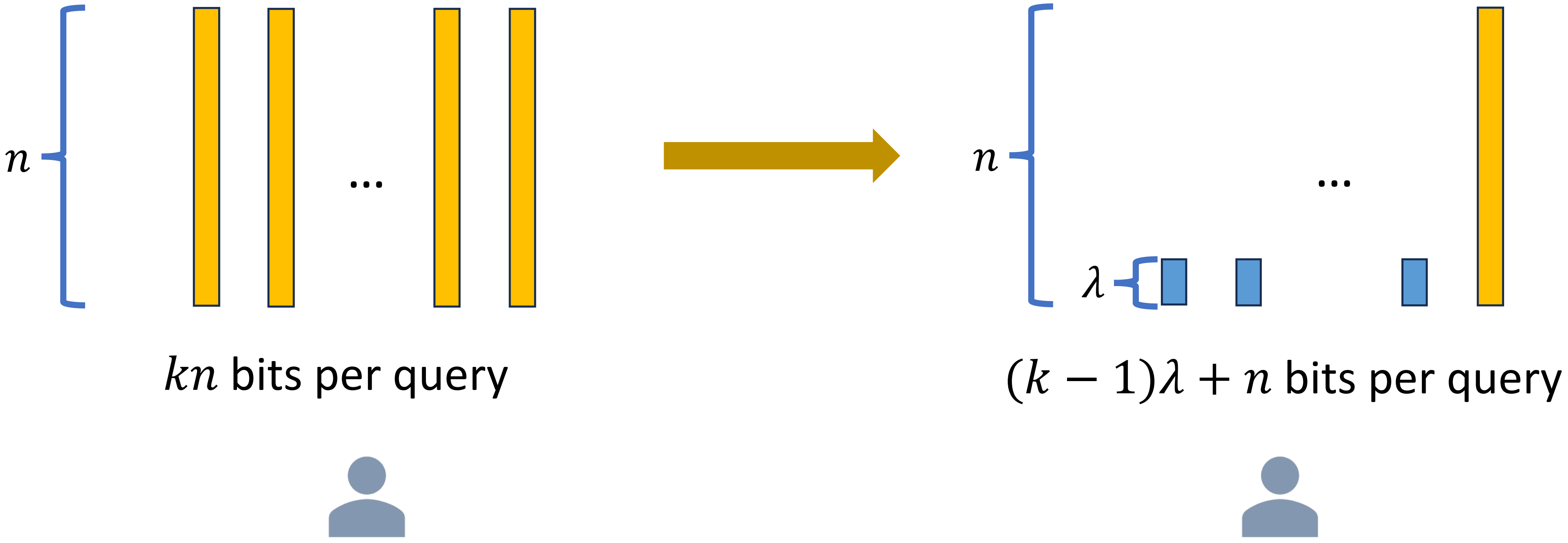
Server sends back answers

Shuffler performs inverse permutation

Optimization for concrete efficiency

- PRG compression
- Batch processing
- Parameter slicing
- Offline/online model

Optimization: PRG compression



Optimization: batch processing

Inner product = sum of the "selected" entries

Database

Share vector (1 0 … 0 1 0 1 … 0 1 1 … 0 1 0 … 1 0)

- ❖ Random access to a large chunk is expensive
- ❖ Sequential access is much faster than random access

Optimization: batch processing

Inner product = sum of the “selected” entries

Database

Share vector 1	(1	0 …	0	1	0	1	…	0	1	1	…	0	1	0	…	1	0)
Share vector 2	(0	0 …	0	0	1	1	…	1	1	1	…	1	0	0	…	0	0)
Share vector 3	(1	1 …	0	1	0	1	…	0	1	1	…	0	1	0	…	1	1)
Share vector 4	(0	0 …	0	1	1	1	…	0	0	1	…	0	1	1	…	0	1)

Optimization: parameter slicing

- The issue of too many dummies:
  - For a database of size $\sim$1K, if each query vector is split into 10 shares, then we need in total $\sim$1M dummies
  - Many dummies $\Rightarrow$ high (and wasteful) anonymity cost and server computation

Optimization: parameter slicing

- Let the SD problem be parameterized by (m, n, k)
- Fix m, k, the smaller n is, the harder the SD instance is

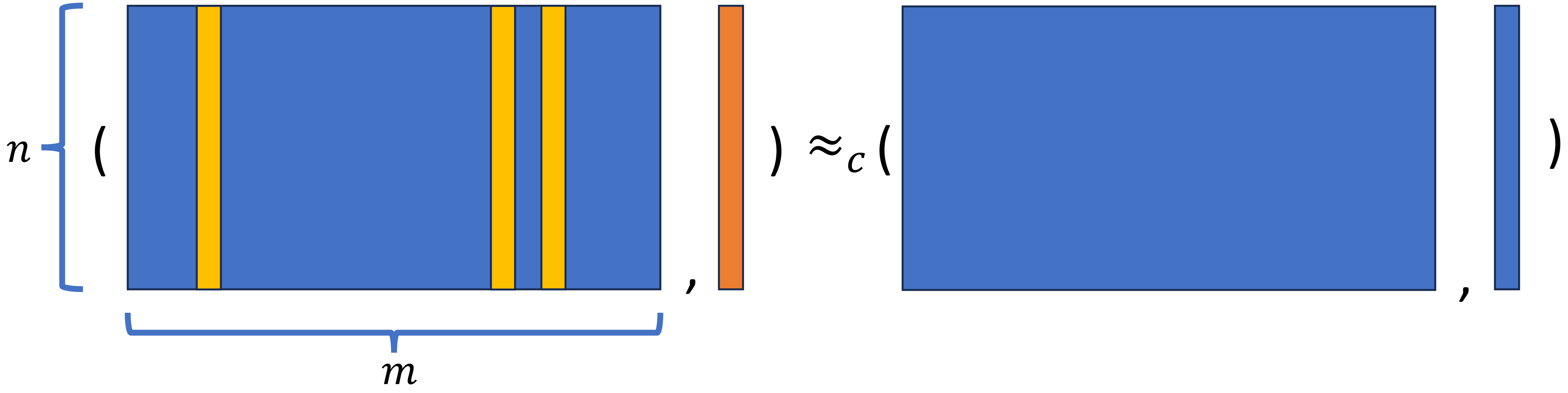
A random matrix

Weight-k

A random vector

$$(H, H \cdot e) \approx_c \ (H, r)$$

Everything in $\mathbb{F}_2$



Optimization: parameter slicing

- Let the SD problem be parameterized by (m, n, k)
- Fix m, k, the smaller n is, the harder the SD instance is

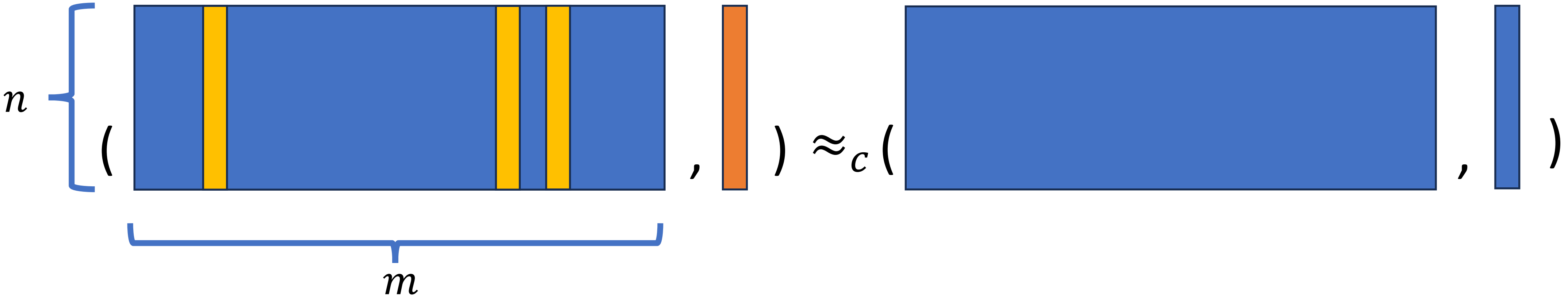
A random matrix

Weight-k

A random vector

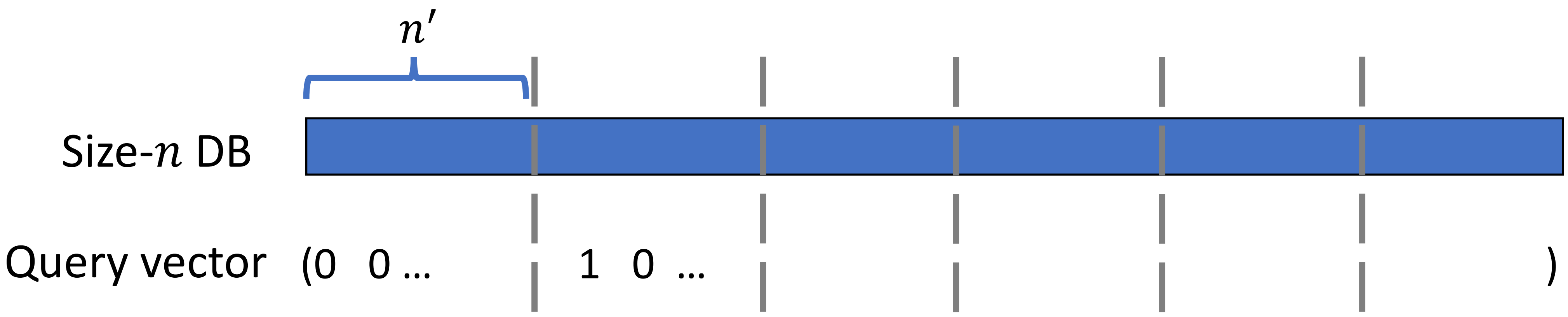
$$(H, H \cdot e) \approx_c (H, r)$$

Everything in $\mathbb{F}_2$



Optimization: parameter slicing

- Let the SD problem be parameterized by (m, n, k)
- Fix m, k, the smaller n is, the harder the SD problem is



Dimension $n' \times m$

Weight k

$$(H, H \cdot e) \approx_c (H, r)$$

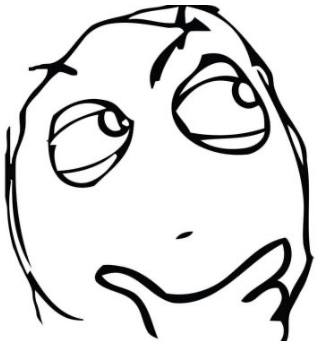
Everything in $\mathbb{F}_2$

Optimization: database slicing

- Let the SD problem be parameterized by (m, n, k)
- Fix m, k, the smaller n is, the harder the SD problem is
- A breakeven point for k and n': when n' becomes smaller,
  - ☺ The corresponding MDSD instance is harder
  - ☹ More MDSD instances, and we need to guarantee the adversary cannot break any of these MDSD instances (can be analyzed by union bound)

Optimization: offline/online model

- Offline: client sends $k-1$ random vectors to the server for preprocessing
- Online: set the last share = query vector – the random vectors



Now we don't shuffle all of them together—will it affect security?

Performance

- PRG compression, batch processing, database slicing, offline/online

Database	# Clients	Our PIR Protocol in Construction 3				SimplePIR		HintlessPIR (ms)	Spiral (ms)
		Offline latency (ms)	Online latency (ms)	Online throughput (GiB/s)	Online throughput (batched, GiB/s)	Latency (ms)	Throughput (GiB/s)		
$2^{20} \times 256$ bytes	1K	132	7	36.64	125.05	45	5.56	575	794
	10K	63							
	100K	47							

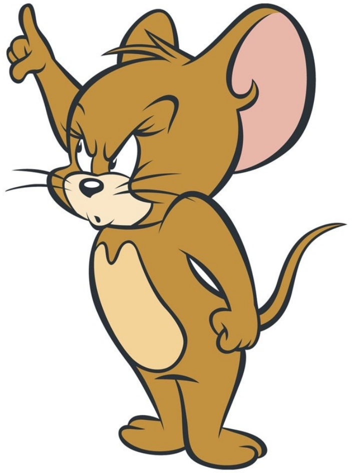
Outline

- Shuffle PIR: the security goal
- The "split-and-mix" paradigm
- Our constructions
  - Shuffle PIR based on LPN, MDSD
  - By-product: computationally secure aggregation in the shuffle model
- Discussion and open questions

Secure aggregation based on MDSD

Isn't the same with the PIR construction?

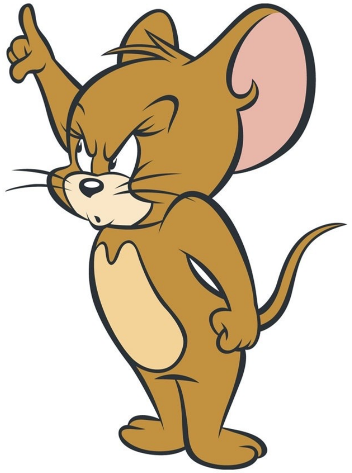
Dummies will change the sum!



Secure aggregation based on MDSD

What should we do?

Using the shuffler to insert shares of zeros!



Long-vector aggregation: optimization

- Same ideas used in many places: key-and-message homomorphic encryption

Short Long

$$Enc(s_1, x_1) + Enc(s_2, x_2) = Enc(s_1 + s_2, x_1 + x_2)$$

We use RLWE to compactly pack data

- c clients send $Enc(s_1, x_1), \ldots Enc(s_c, x_c)$
- Use the shuffle scheme for aggregating $s_1, \ldots, s_c$
  (Similar as before, we can apply PRG compression)

Performance

Input × Field	Input size (KiB)	# Clients	Aggregation Protocol in Construction 1			Info-Theoretic Protocol [BBGN20]		
			# Shares	Upload size (KiB)	Expansion ratio	# Shares	Upload size (KiB)	Expansion ratio
$2^{15} \times \mathbb{F}_2$	4	100	405	10	2.57	6317	103	25.67
		1000	88	5	1.34	3856	64	16.06
		10000	37	5	1.14	2775	47	11.84
$2^{15} \times \mathbb{F}_{65537}$	64	100	410	70	1.10	100819	1639	25.61
		1000	77	65	1.02	61525	1025	16.02
		10000	33	65	1.01	44271	756	11.81

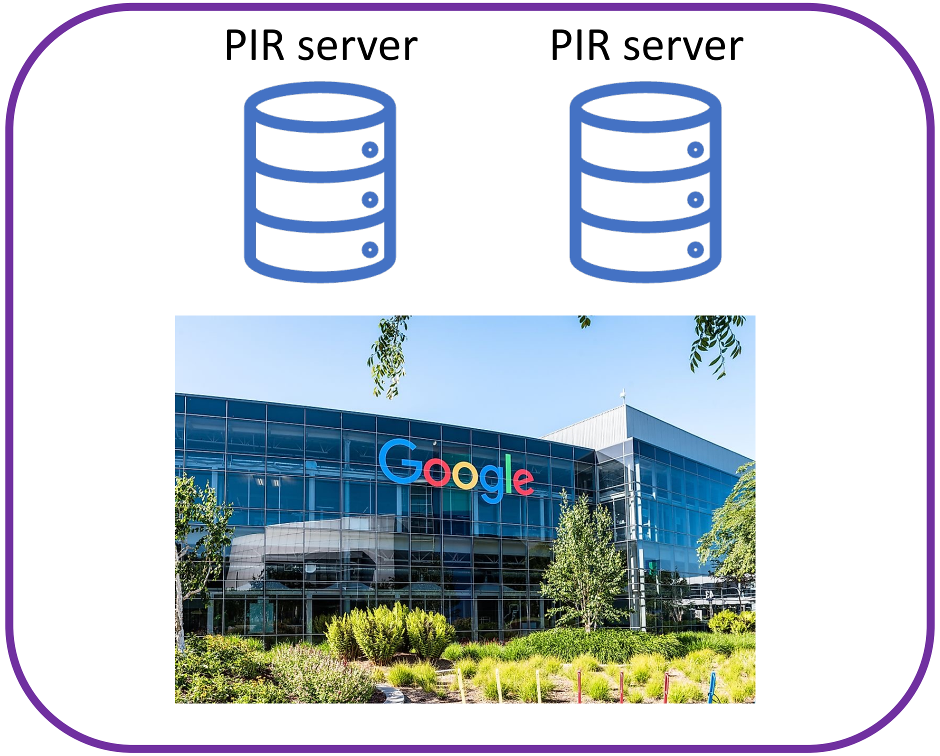
Expansion ratio measures the communication overhead of the protocol over the input vector size

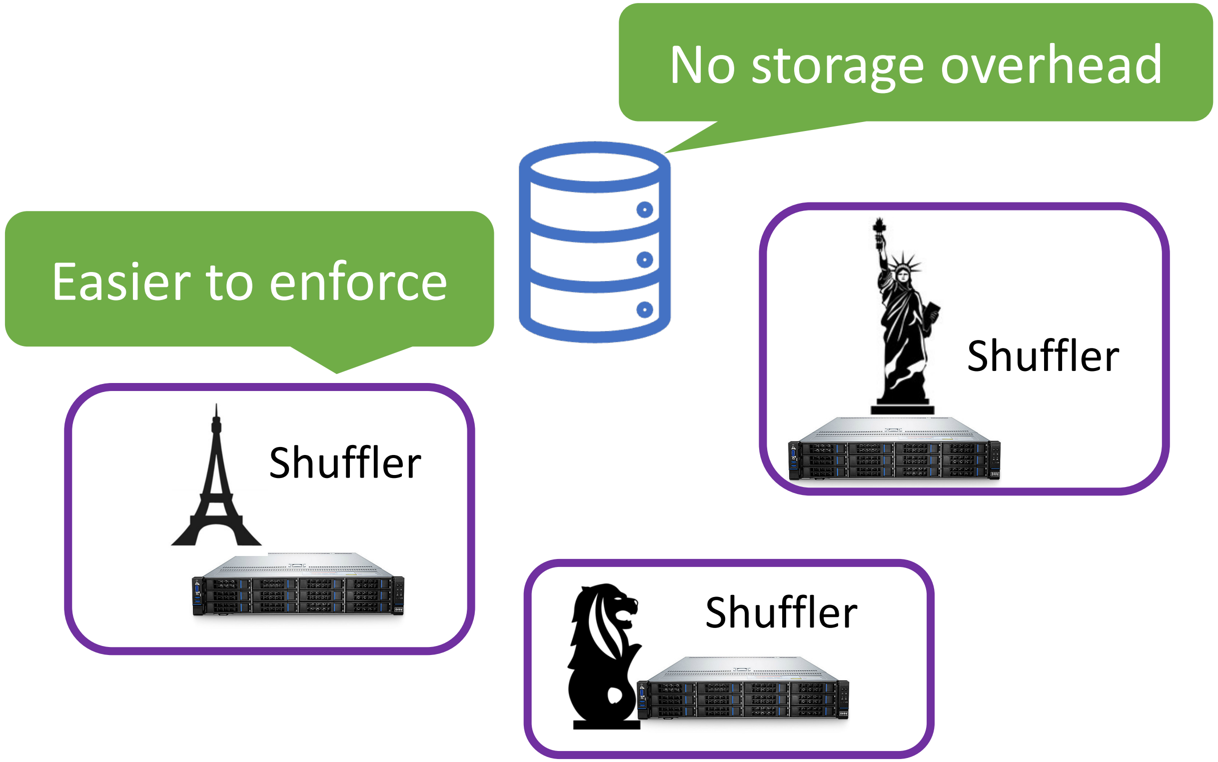
Outline

- Shuffle PIR: the security goal
- The “split-and-mix” paradigm
- Our constructions
  - Shuffle PIR based on LPN, MDSD
  - By-product: computationally secure aggregation in the shuffle model
- Discussion and open questions

Discussion

- Assuming non-colluding servers vs. assuming a two-way anonymous channel





Discussion

- We are in the situation of exploiting tradeoffs: making assumptions, altering models (different types of preprocessing, relaxed security, etc.)
- Guaranteeing different assumptions does not requrie the same amount of efforts: system efforts, law efforts, etc.
- The likelihood of assumptions being compromised in real-world scenarios may vary

We should keep these in mind when introducing new models!

Backup Slides

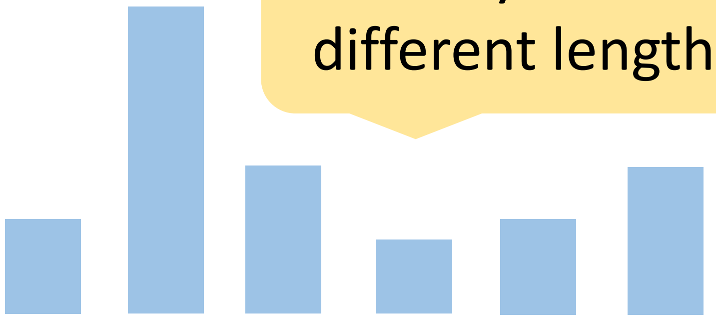
PIR with variable-sized records

- To deploy PIR in real-world applications…

Often assume the same size, mostly $\{0,1\}^n$

Database entries of PIR in theory

They have different lengths



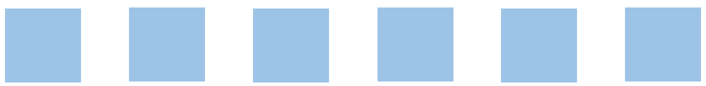
Database records in practice

To retrieve privately, it is necessary to hide record size

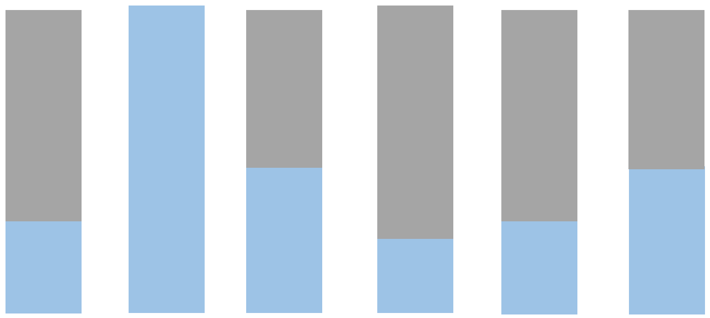
PIR with variable-sized records

- To deploy PIR in real-world applications…

Often assume the same size, mostly $\{0,1\}^n$



Database entries of PIR in theory

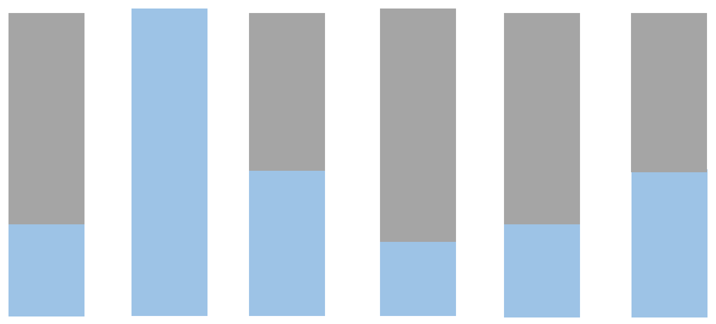


Database records in practice

To retrieve privately, it is necessary to hide record size

PIR with variable-sized records

- Padding solves the problem: how about efficiency?



Database records in practice

Features

The discrepancy between the smallest and the largest record can be huge

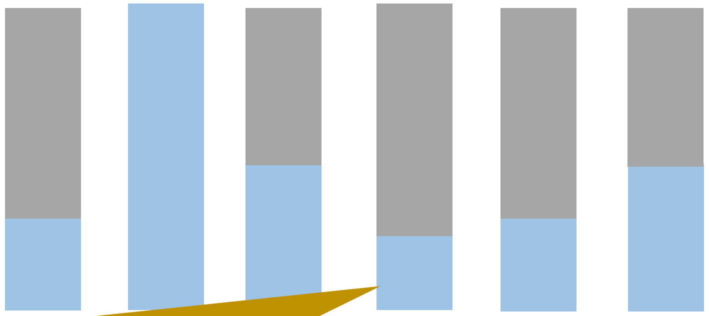
Majority of the records are small

Most users access the small records much more often than the large records

PIR with variable-sized records

- Padding solves the problem: how about e

Waste of server storage
(though can virtually store)



practice

Features

Client who retrieves the small record has to
pay the cost of retrieving the largest record

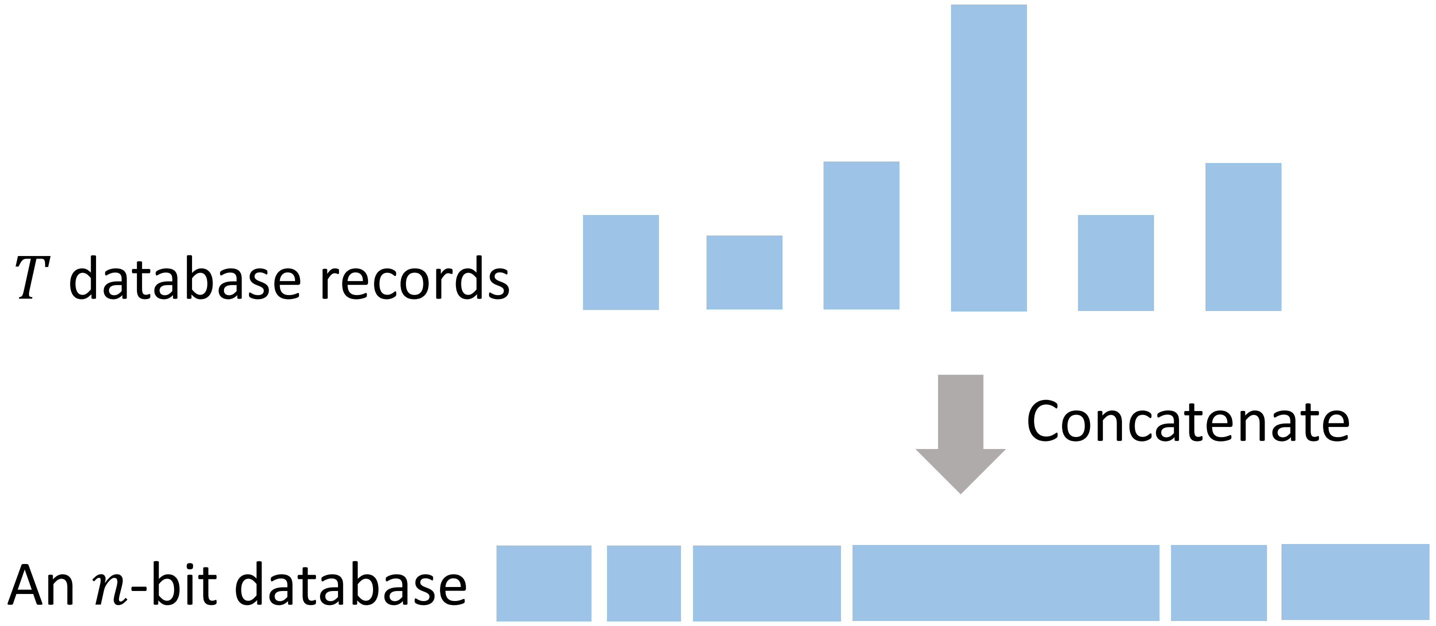
The discrep
an be huge
Majority of the records are small
Most users access the small records much more often than the large records

PIR with variable-sized records

- In the “standard” model, there is no way out
- In the shuffle model: yes, we can
  - No server storage overhead
  - Client communication proportional to the length of the retrieved record
  - Leak only the total size of all queried records

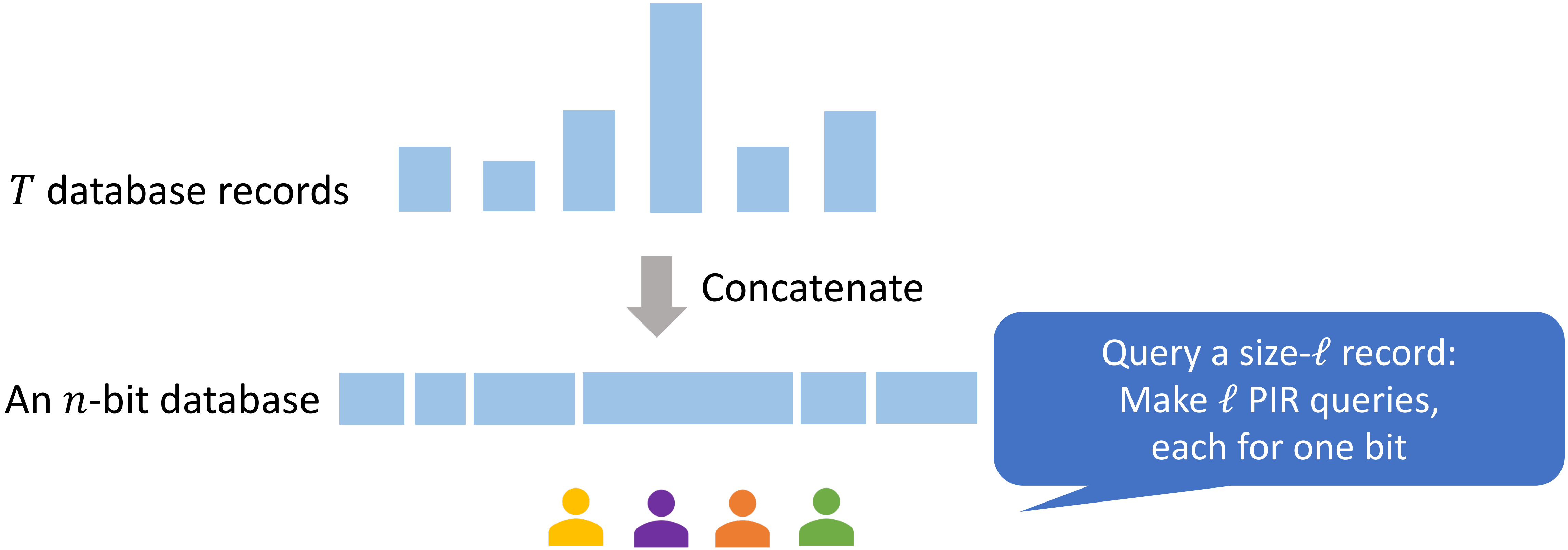
PIR with variable-sized records

- A toy protocol



PIR with variable-sized records

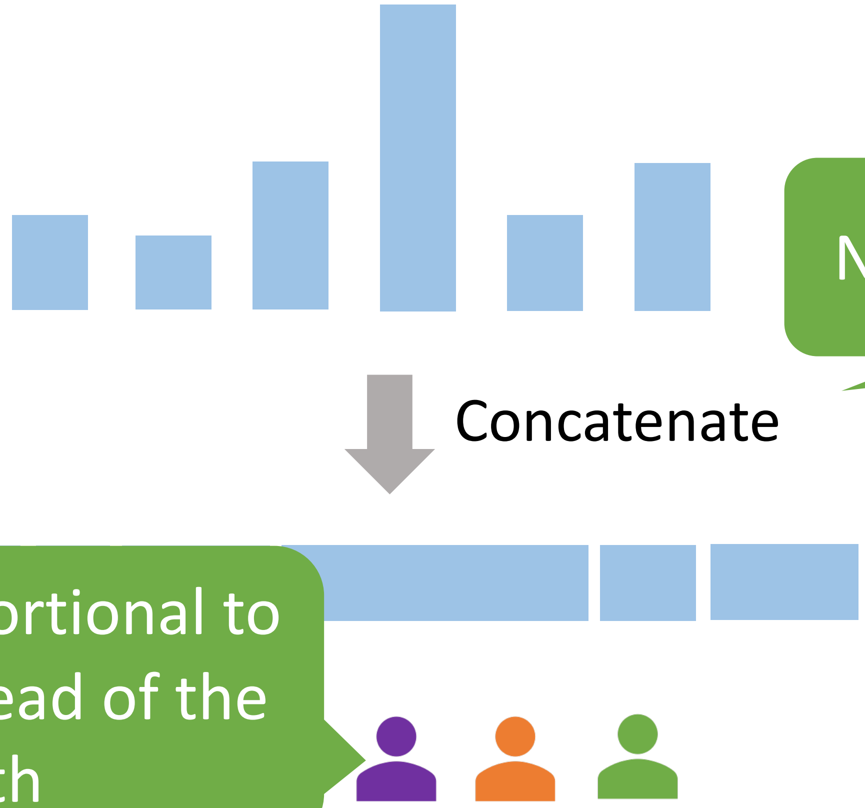
- A toy protocol



PIR with variable-sized records

- A toy protocol

T database records



No server storage overhead

Query a size-ℓ record:
Make ℓ PIR queries,
each for one bit

Communication is proportional to the queried length instead of the maximum length

PIR with variable-sized records

- A toy protocol

T data

...age overhead

Can we do better?

Yes, from ℓ PIR queries to $\text{polylog}\,\ell$ PIR queries

Query a size-ℓ record:
Make ℓ PIR queries,
each for one bit

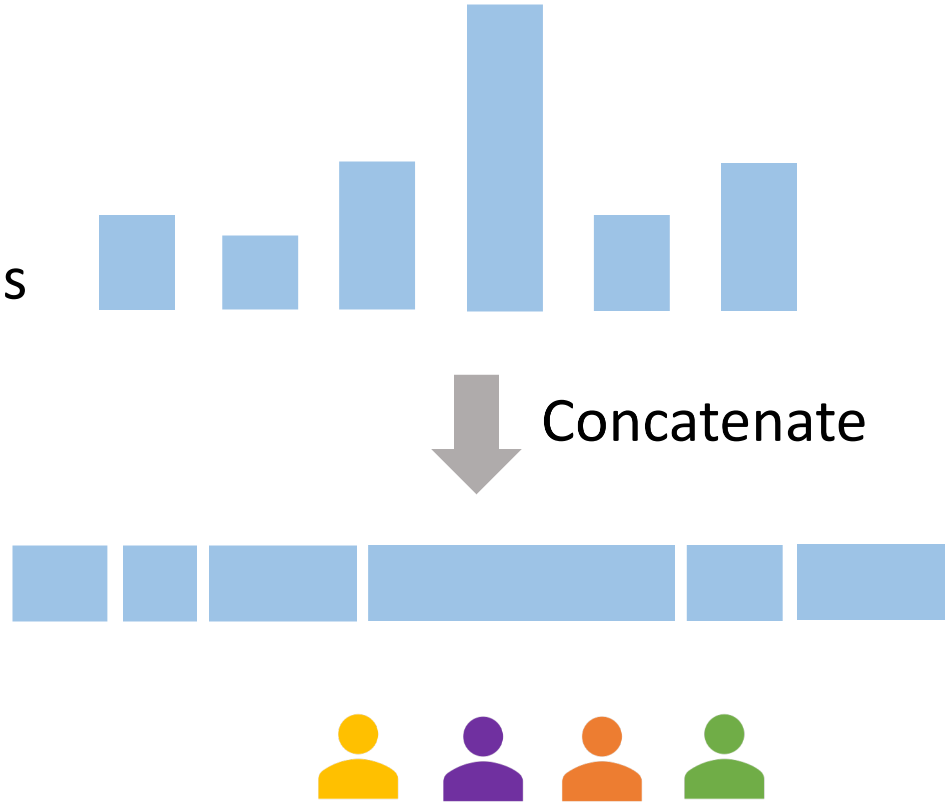
Communication is proportional to the queried length instead of the maximum length

PIR with variable-sized records

- Revisit the toy protocol

T database records

An n-bit database

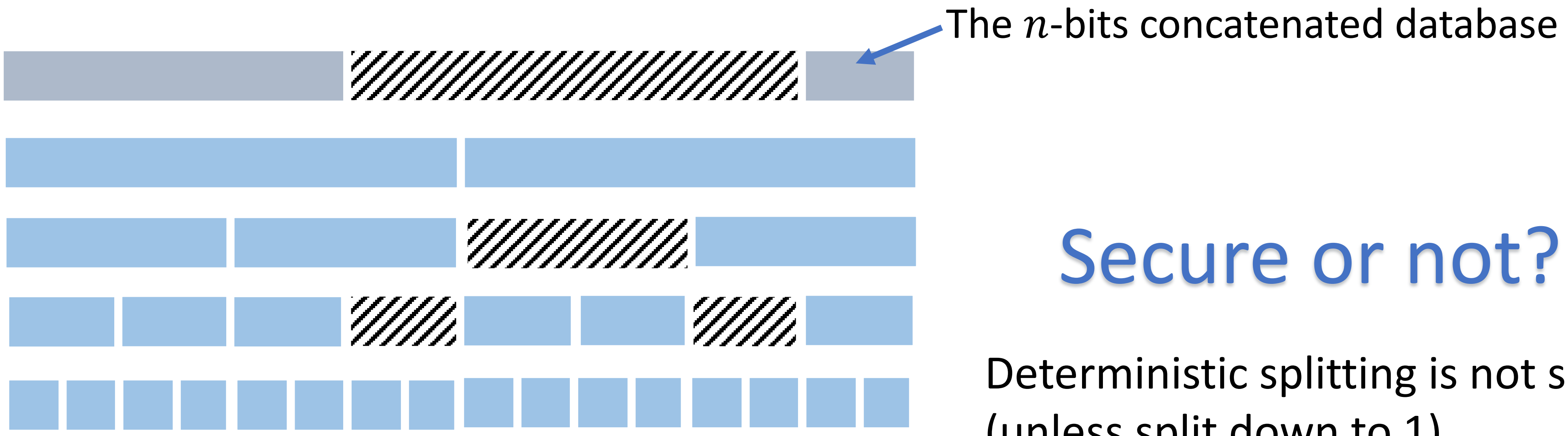


Why not retrieve more bits in each PIR query?

Query a size-ℓ record:
Make ℓ PIR queries,
each for one bit

PIR with variable-sized records

- Splitting records to the powers of two



Server (logically) preprare $\log n$ databases:
the j-th database is partitioned to 2^j bits per entry

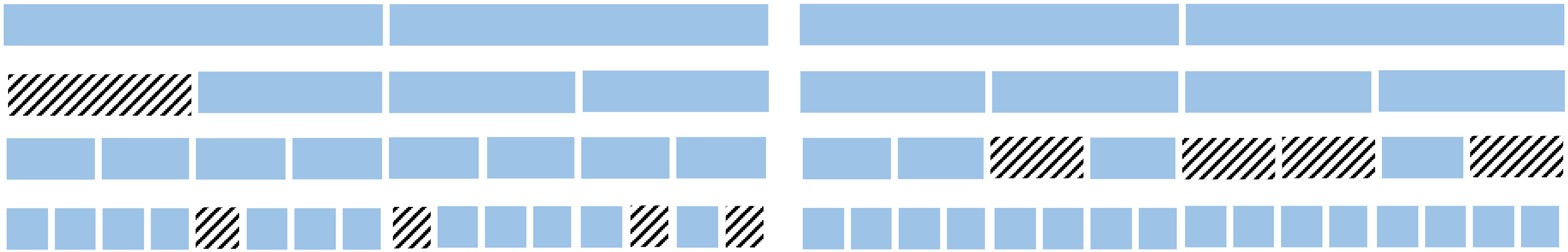
Secure or not?

Deterministic splitting is not secure
(unless split down to 1)

PIR with variable-sized records

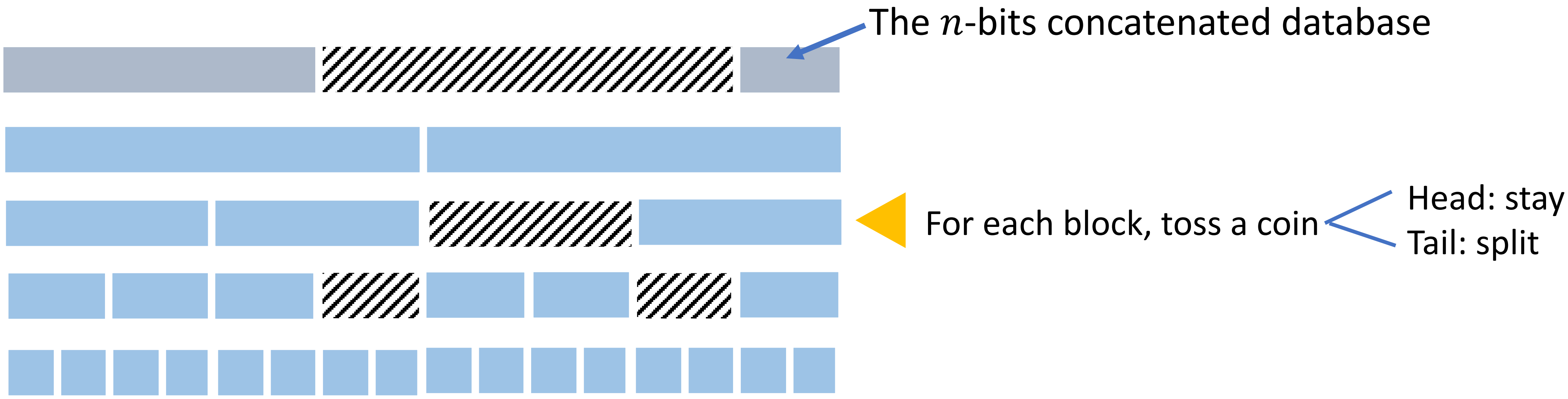
- Splitting records to the powers of two

Consider 5 1 1 1 v.s. 2 2 2 2



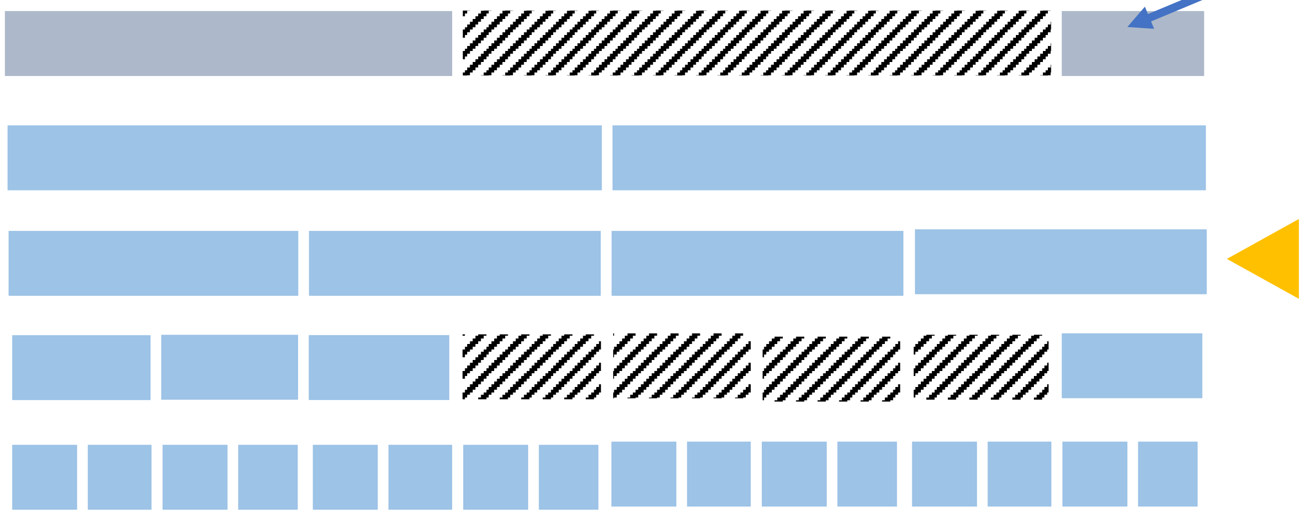
PIR with variable-sized records

- Our approach: recursive splitting



PIR with variable-sized records

- Our approach: recursive splitting



The n-bits concatenated database

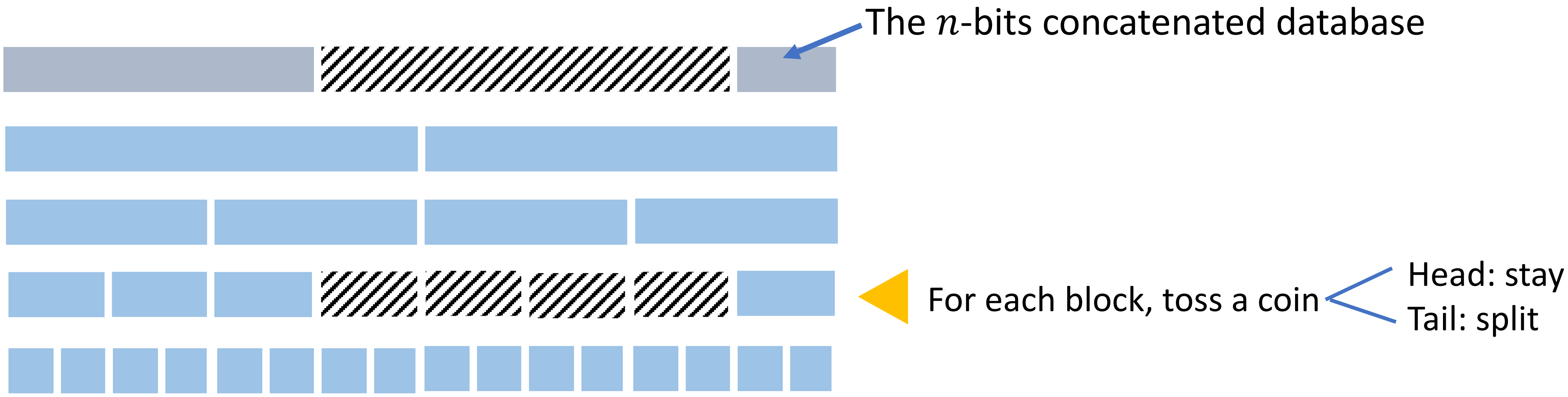
For each block, toss a coin

Head: stay

Tail: split

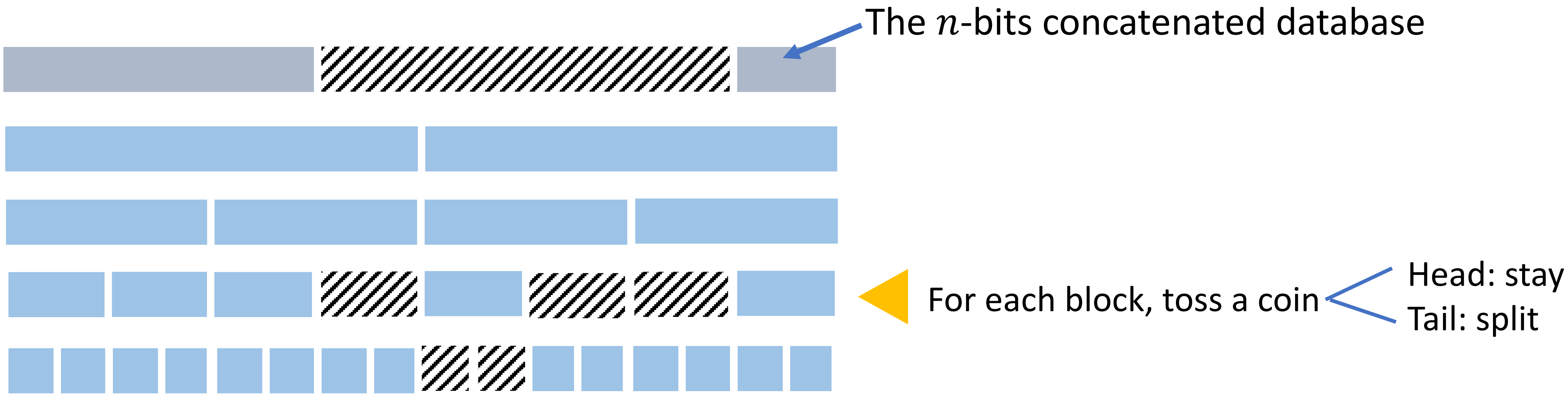
PIR with variable-sized records

- Our approach: recursive splitting



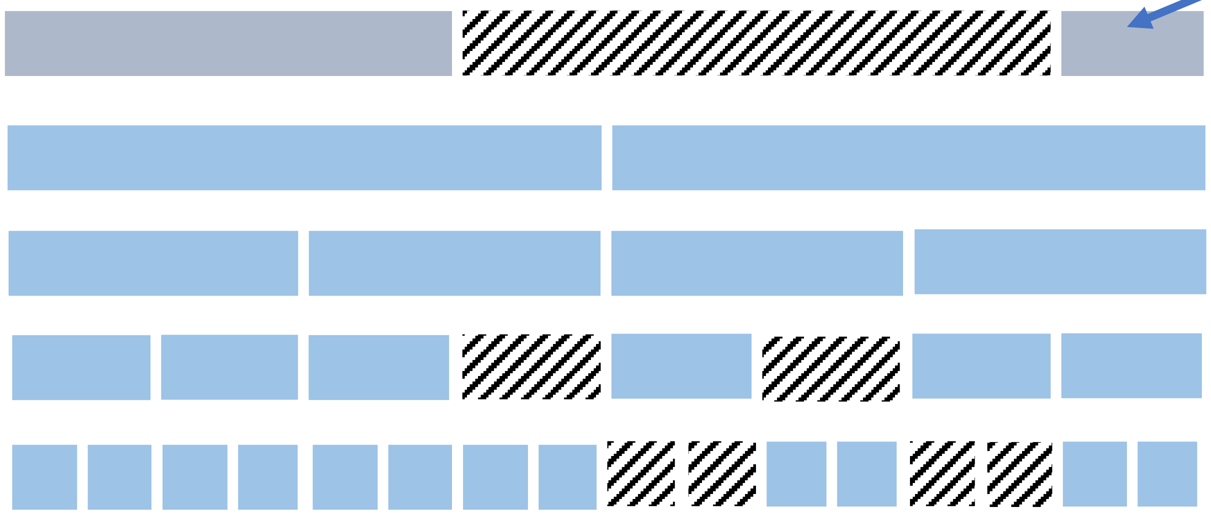
PIR with variable-sized records

- Our approach: recursive splitting



PIR with variable-sized records

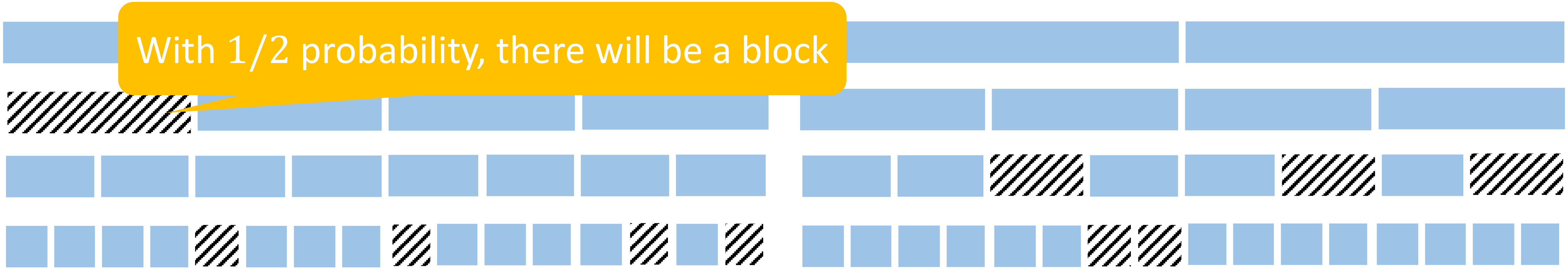
- Our approach: recursive splitting



PIR with variable-sized records

- A complication of recursive splitting: fully split the highest $\log C$ levels

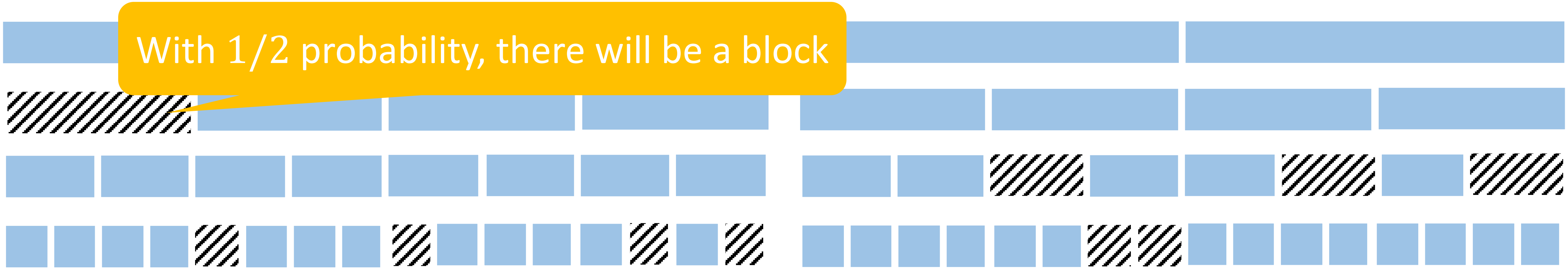
Consider 5 1 1 1 v.s. 2 2 2 2



PIR with variable-sized records

- A complication of recursive splitting: fully split the highest $\log C$ levels

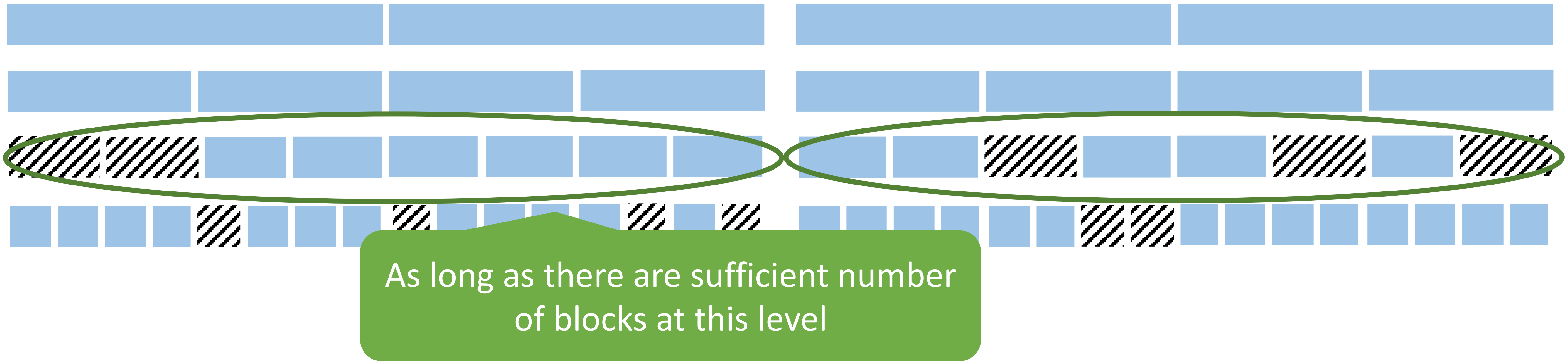
Consider M-3 1 1 1 v.s. M/4 M/4 M/4 M/4



PIR with variable-sized records

- A complication of recursive splitting: fully split the highest $\log C$ levels

Consider M-3 1 1 1 v.s. M/4 M/4 M/4 M/4



PIR with variable-sized records

- Splitting records to the power of two