Single-Server Private Information Retrieval with Sublinear Amortized Time

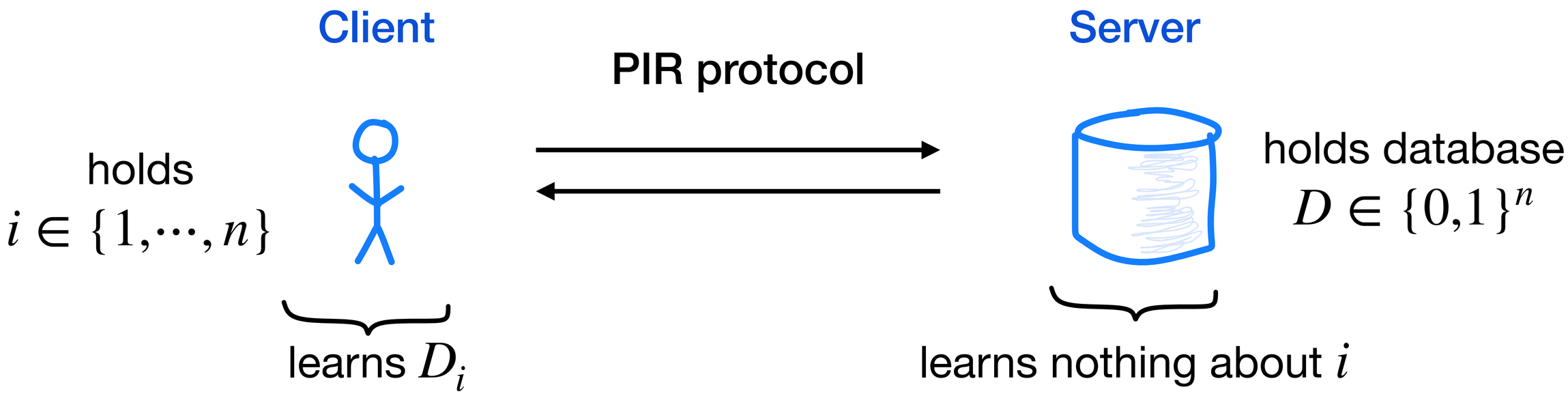
Henry Corrigan-Gibbs
MIT

Alexandra Henzinger
MIT

Dmitry Kogan
Fordefi

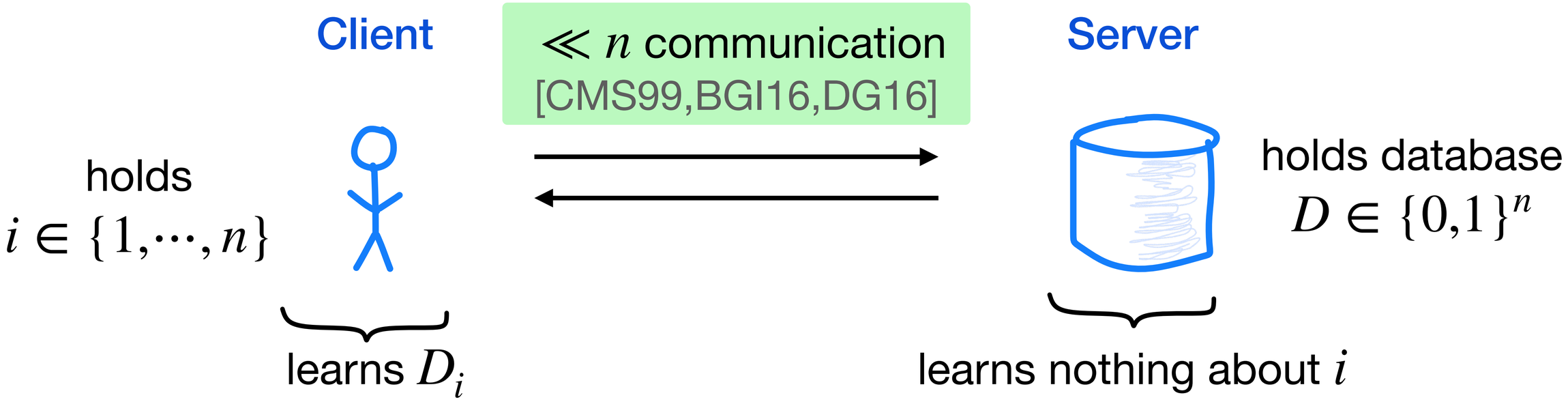
Eurocrypt ’22

Private information retrieval [CGKS95,KO97]

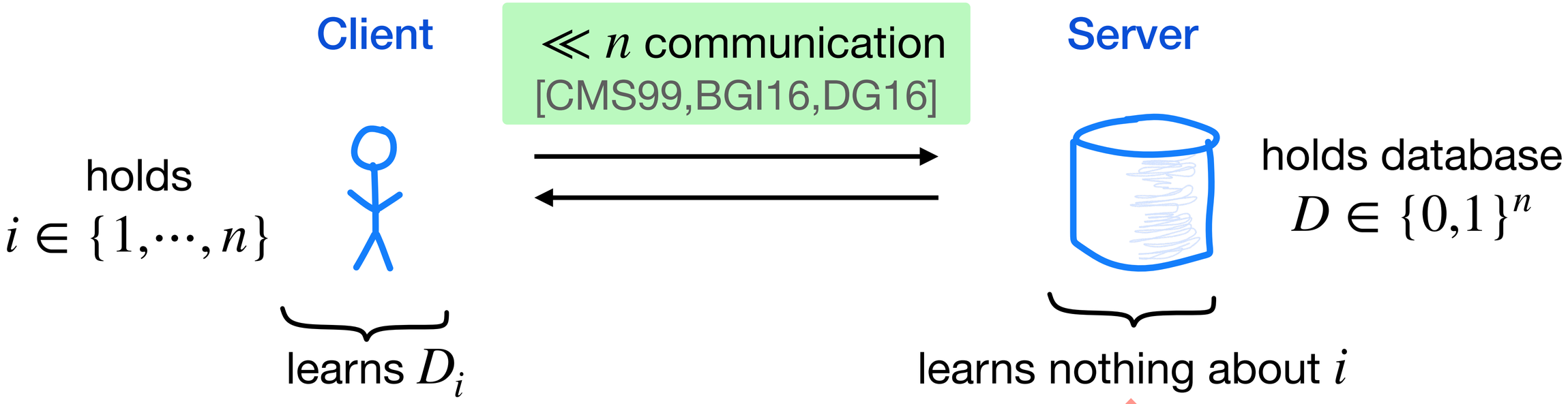


Applications: private media [GCMSAW16], private e-commerce [HOG11], private ads [J01…], private web browsing [KC21], metadata-hiding messaging [AS16…], …

Private information retrieval [CGKS95,KO97]

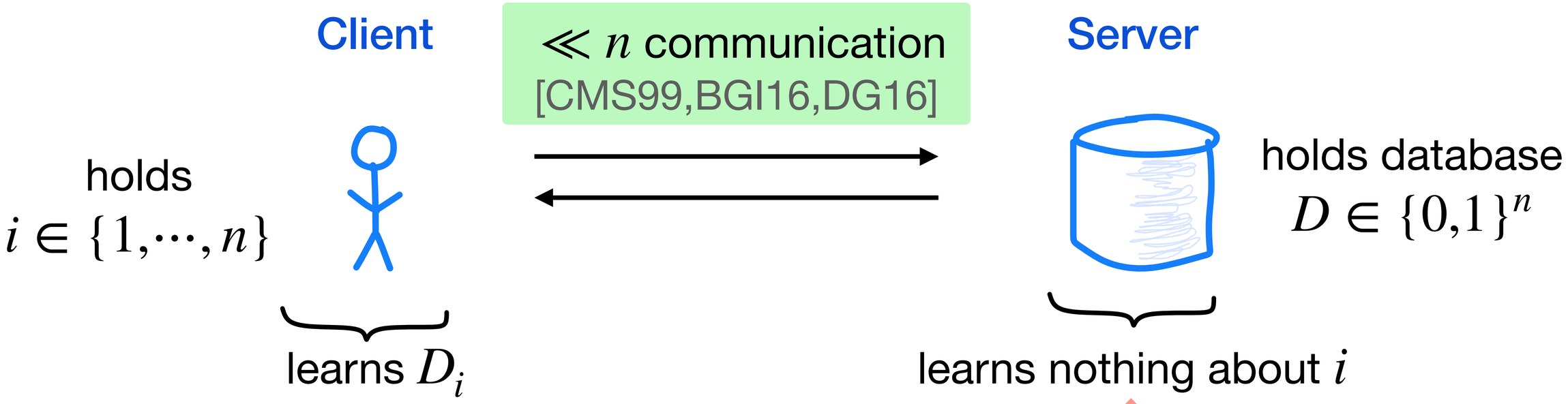


Private information retrieval [CGKS95,KO97]



PIR inherently has **high server-side computation** costs [BIM04,PY22]:
To answer a single query, the server(s) must run in time n.

Private information retrieval [CGKS95,KO97]

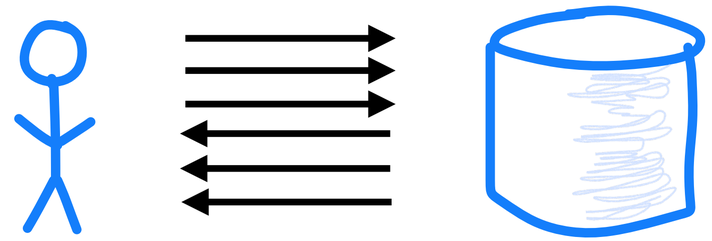


PIR inherently has **high server-side computation** costs [BIM04,PY22]:
To answer a single query, the server(s) must run in time n.

Idea: Amortize the server time over <u>many</u> queries [BIM04,IKOS04]

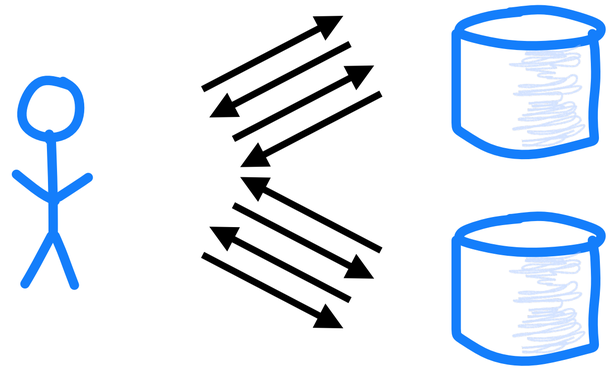
Existing PIR with sublinear time

Batch PIR with non-adaptive queries



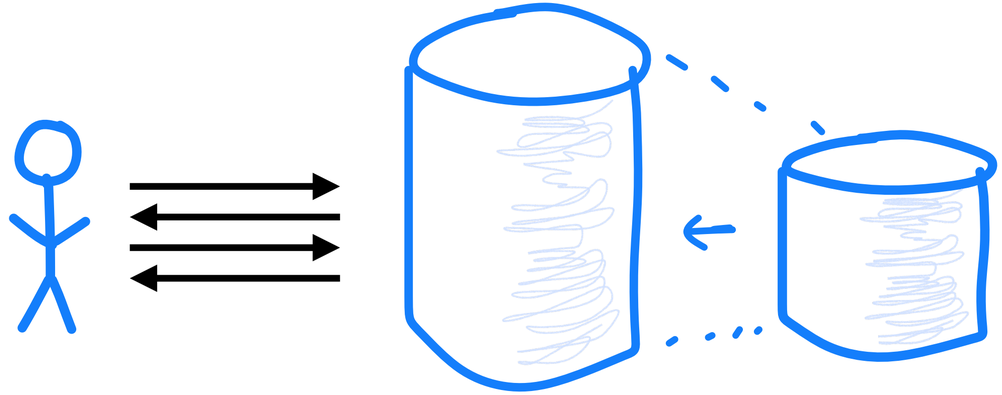
[IKOS04,HHG13,GKL10,AS16,H16,ACLS18,CHLR18]

Offline/online PIR with 2 servers



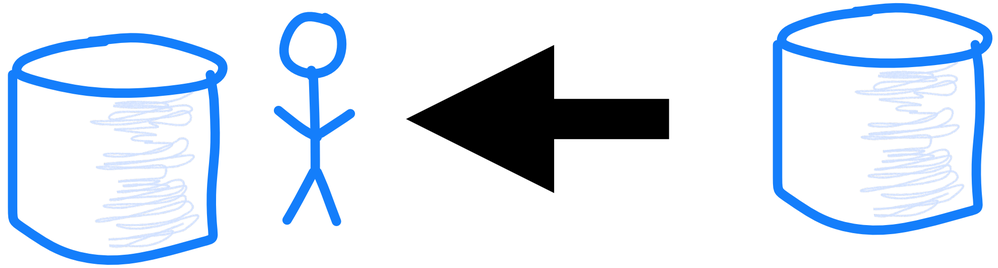
[CK20,SACM21,KC21]

PIR with preprocessing



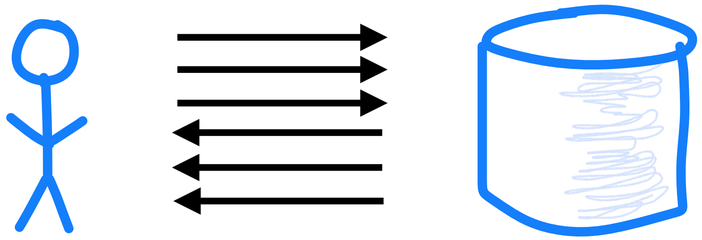
[BIPW17,CHR17,HOWW18]

Download the database



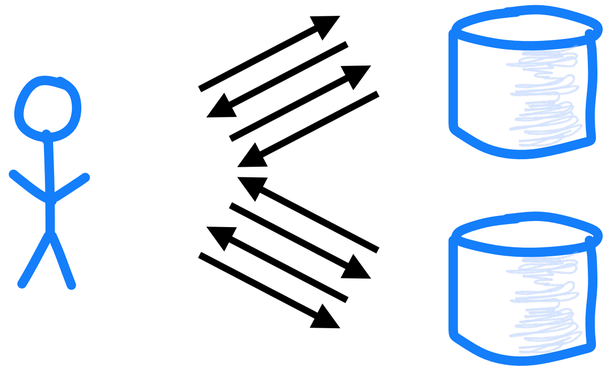
Existing PIR with sublinear time

Batch PIR with non-adaptive queries



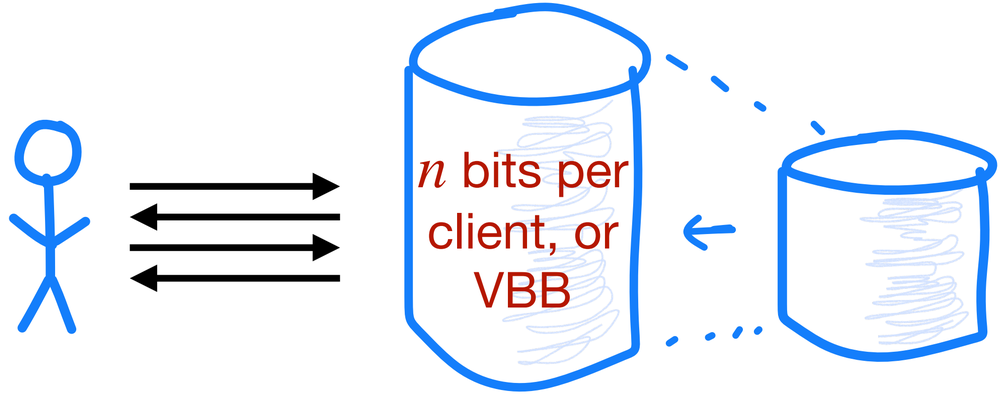
[IKOS04,HHG13,GKL10,AS16,H16,ACLS18,CHLR18]

Offline/online PIR with 2 servers



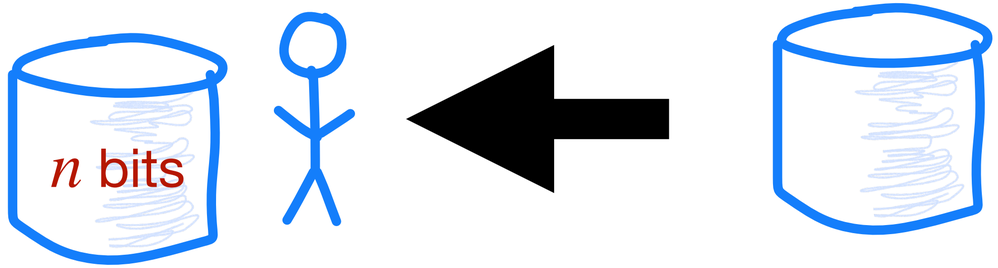
[CK20,SACM21,KC21]

PIR with preprocessing



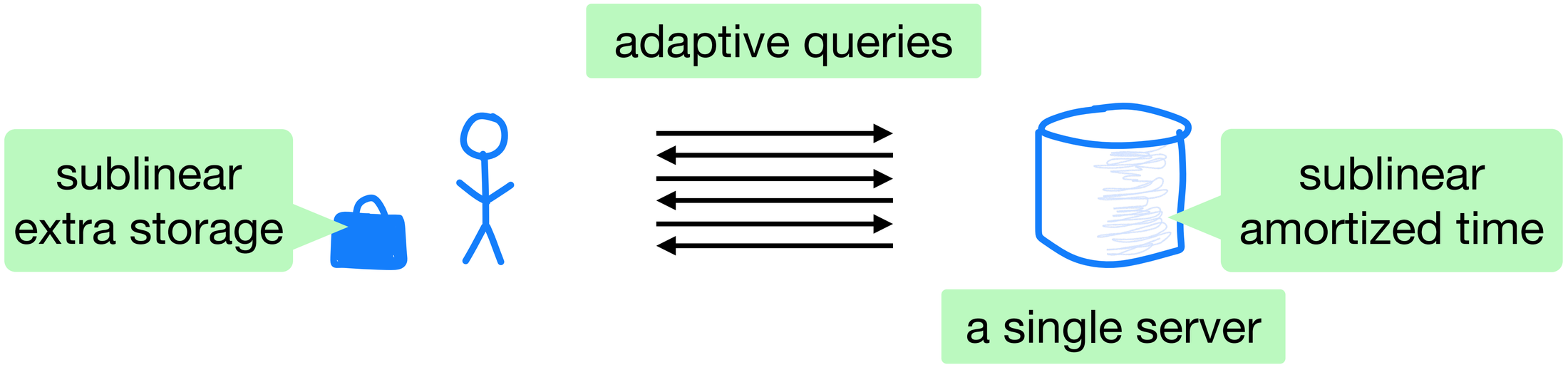
[BIPW17,CHR17,HOWW18]

Download the database



This work

1. **The first PIR schemes** to have:

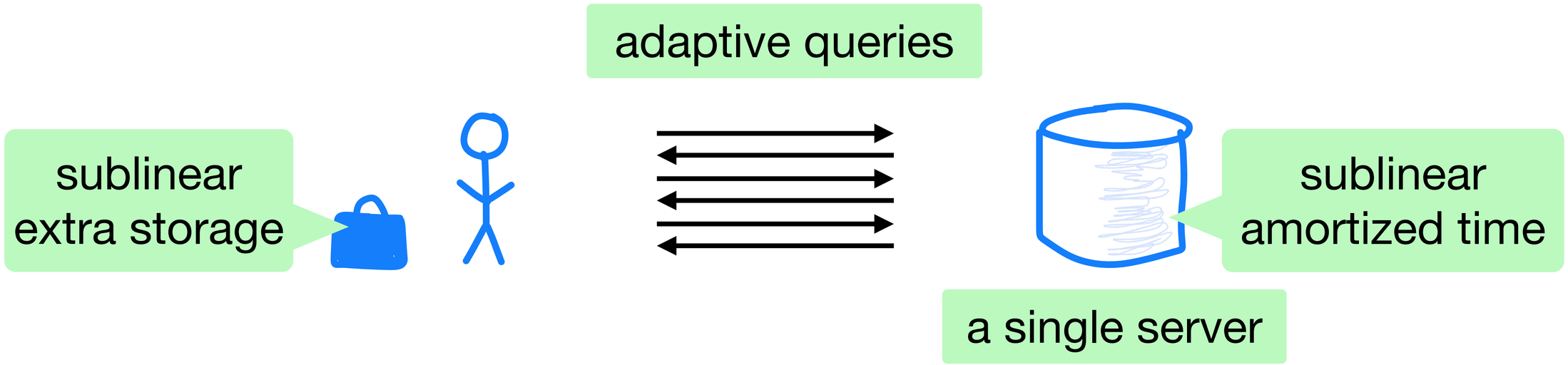


2. **Matching lower bounds** that relate server time and client storage.

This work

Results preview:
$n^{3/4}$ time + storage from DDH
$n^{1/2}$ time + storage from FHE

1. **The first PIR schemes** to have:

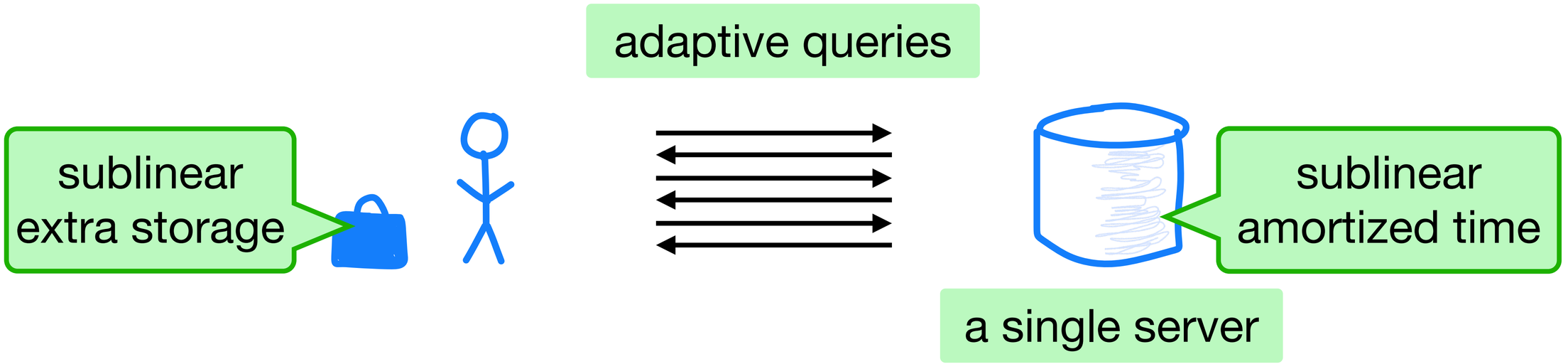


2. **Matching lower bounds** that relate server time and client storage.

This work

Results preview:
$n^{3/4}$ time + storage from DDH
$n^{1/2}$ time + storage from FHE

1. **The first PIR schemes** to have:



2. **Matching lower bounds** that relate server time and client storage.

This talk

1. Background: The offline/online PIR model
2. Our results: New PIR schemes with sublinear time
3. Open questions

This talk

1. Background: The offline/online PIR model
2. Our results: New PIR schemes with sublinear time
3. Open questions

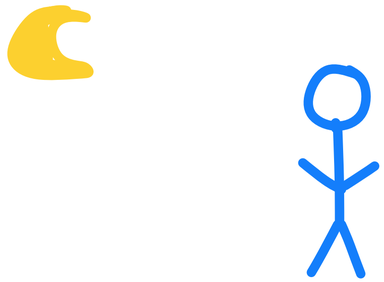
Goal: build PIR for Q adaptive queries, with **sublinear amortized time**

Our approach: build PIR with two phases

1. Once, run a linear-time "offline" phase.

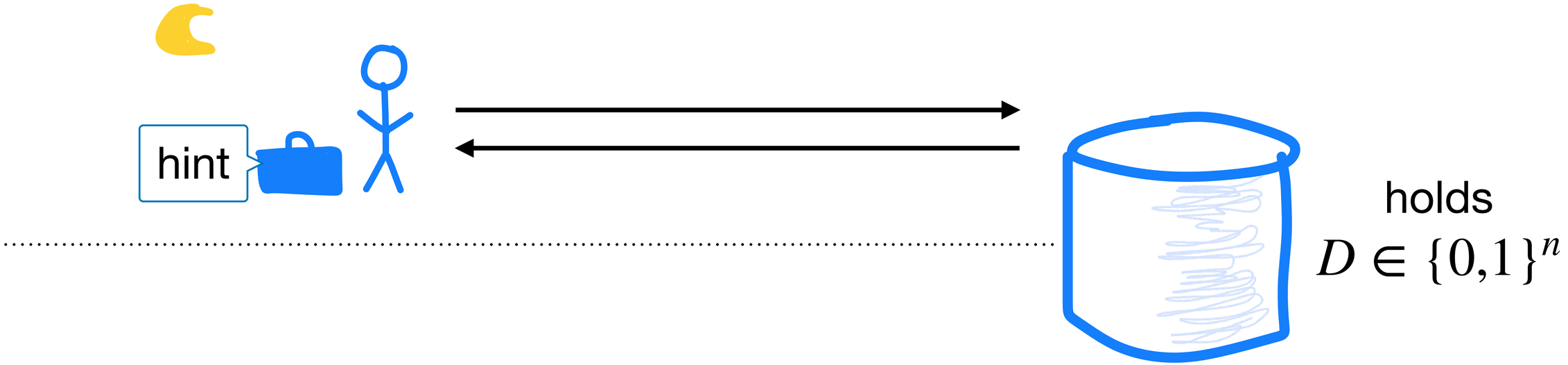
2. For each of the Q queries, run a sublinear-time "online" phase.

Many-query offline/online PIR

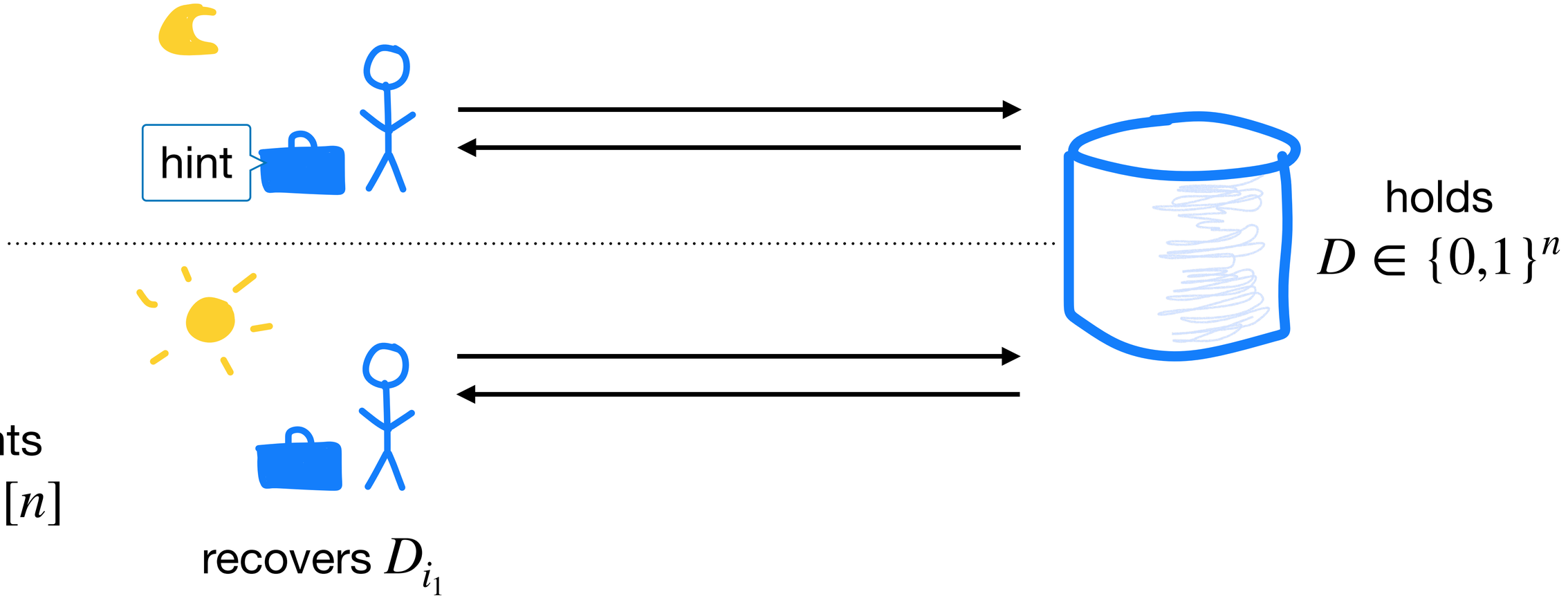


holds
$D \in \{0,1\}^n$

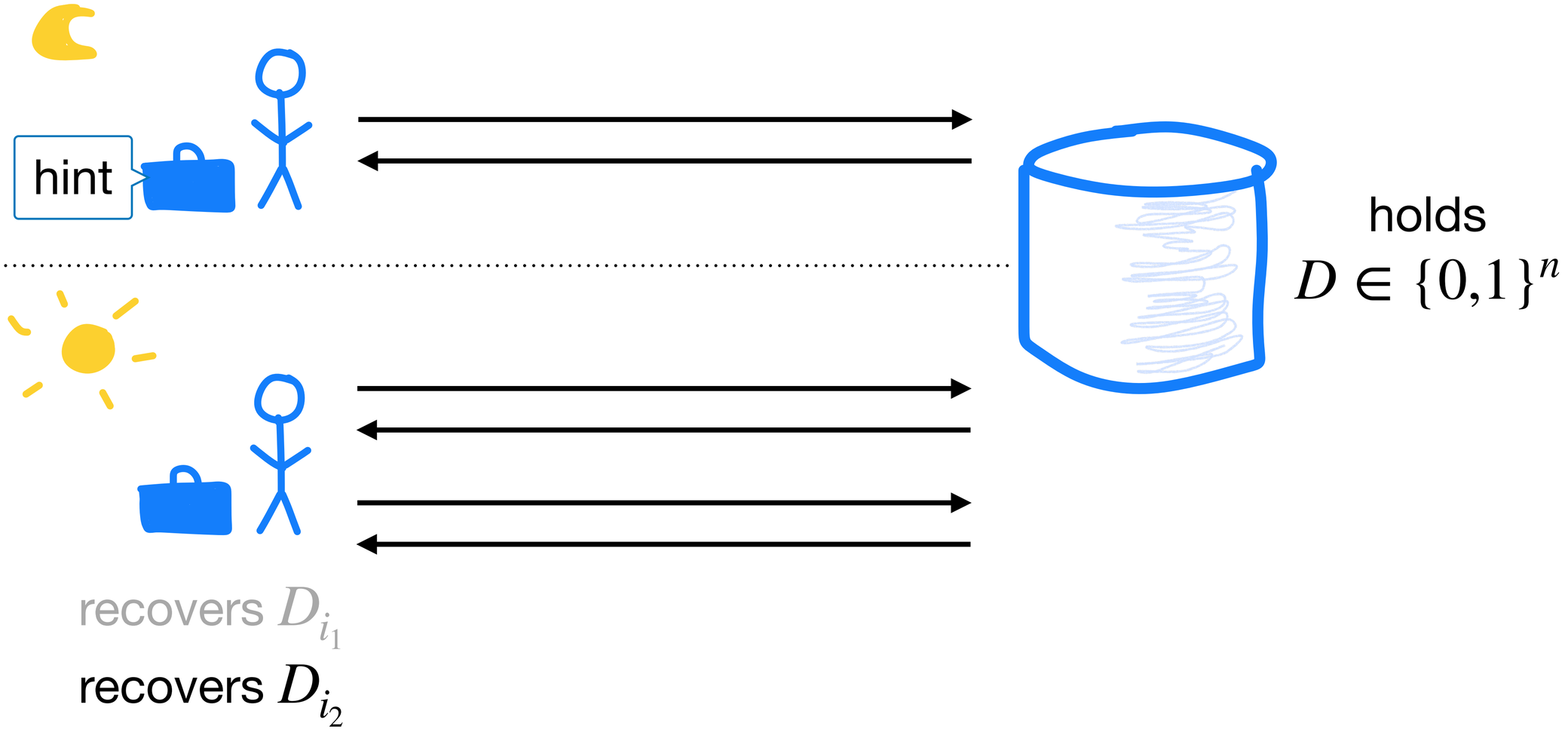
Many-query offline/online PIR



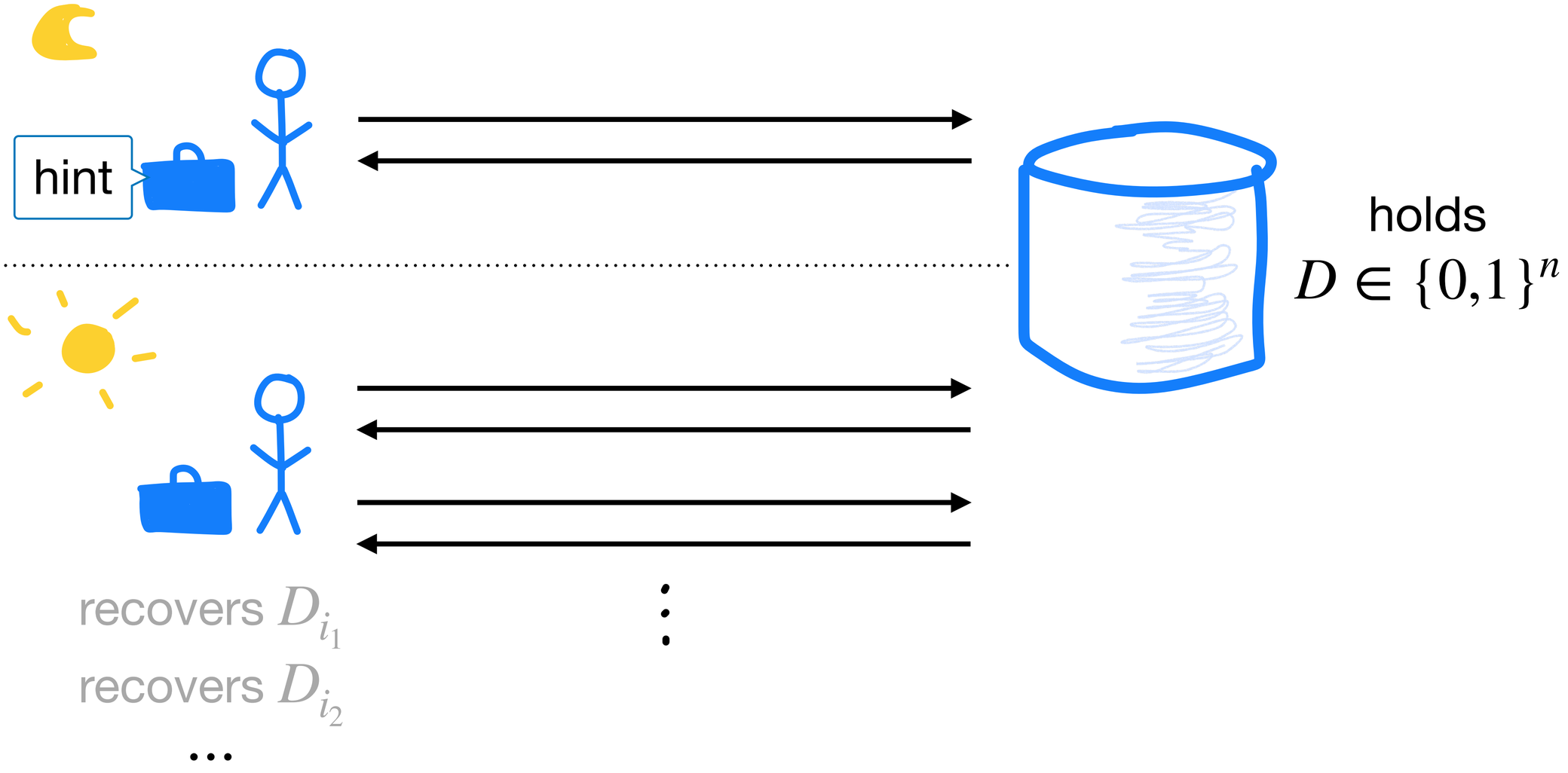
Many-query offline/online PIR



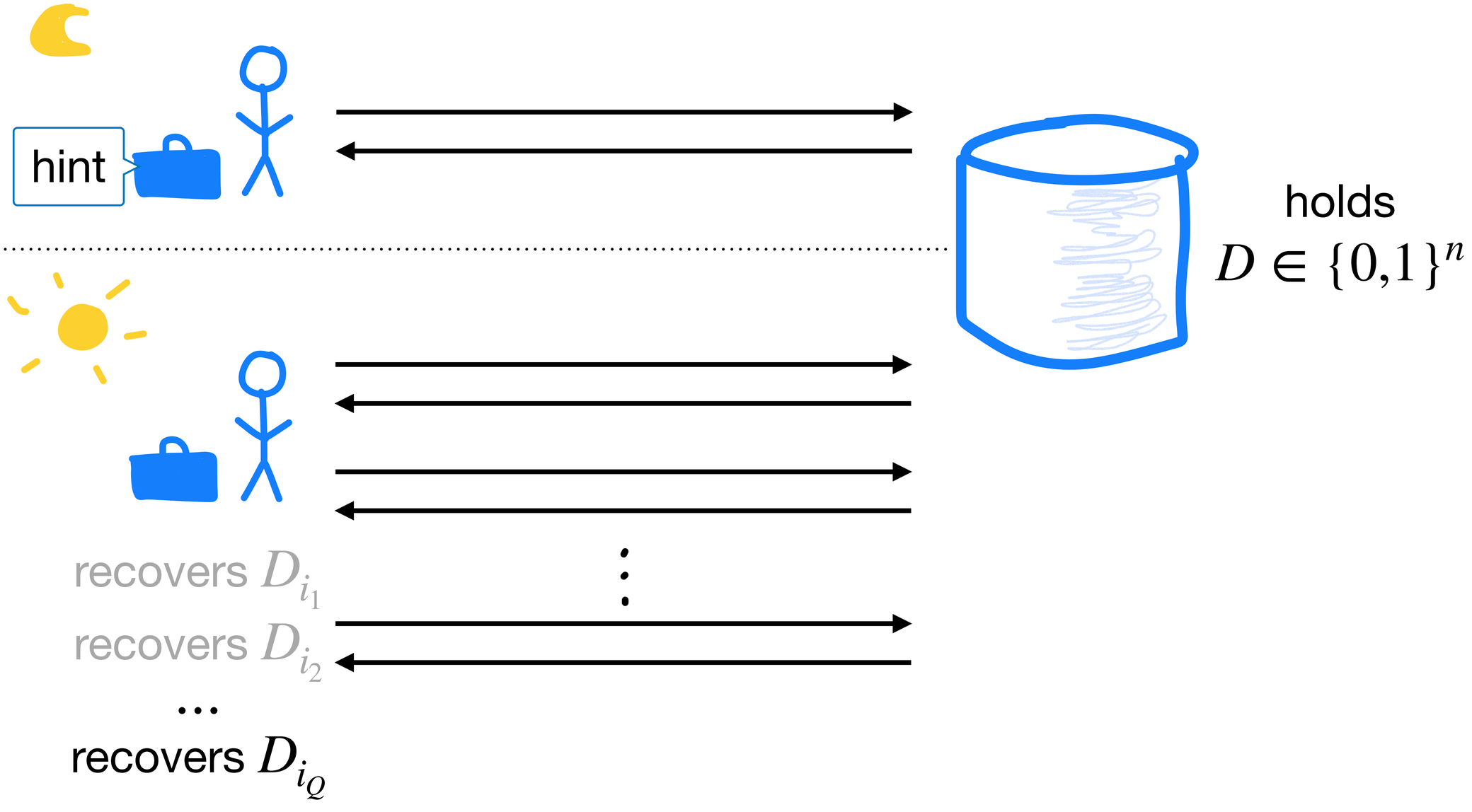
Many-query offline/online PIR



Many-query offline/online PIR



Many-query offline/online PIR



Many-query PIR requirements

- **Correctness**: If the client and server execute the protocol faithfully, for any D, for any $i_1, \cdots, i_Q \in [n]$, the client correctly recovers $D_{i_1}, \cdots, D_{i_Q}$, with overwhelming probability.

- **Malicious security**: Even if the server does not follow the protocol, the server learns nothing about $i_1, \cdots, i_Q$.

  More formally, for any $I, I' \in [n]^Q$,

  $\{\text{Server's view on query sequence } I\} \approx_c \{\text{Server's view on query sequence } I'\}$

Many-query PIR requirements

- **Correctness**: If the client and server execute the protocol faithfully, for any D, for any $i_1, \cdots, i_Q \in [n]$, the client correctly recovers $D_{i_1}, \cdots, D_{i_Q}$, with overwhelming probability.

- **Malicious security**: Even if the server does not follow the protocol, the server learns nothing about $i_1, \cdots, i_Q$.

  More formally, for any $I, I' \in [n]^Q$,

  $\{\text{Server's view on query sequence } I\} \approx_c \{\text{Server's view on query sequence } I'\}$

In our schemes, the queries are independent of the server's past answers.

Many-query PIR requirements

- **Correctness**: If the client and server execute the protocol faithfully, for any D, for any $i_1, \cdots, i_Q \in [n]$, the client correctly recovers $D_{i_1}, \cdots, D_{i_Q}$, with overwhelming probability.

- **Malicious security**: Even if the server does not follow the protocol, the server learns nothing about $i_1, \cdots, i_Q$.
  More formally, for any $I, I' \in [n]^Q$,
  $\{\text{Server's view on query sequence } I\} \approx_c \{\text{Server's view on query sequence } I'\}$

Goal: Minimize communication, computation, and storage costs

This talk

1. Background: The offline/online PIR model
2. Our results: New PIR schemes with sublinear time
3. Open questions

Theorem 1: From linearly homomorphic encryption.

Under DDH, QR, DCR, or LWE, there is a single-server PIR scheme that, on database size n, when the client makes $n^{1/4}$ adaptive queries, has:

- amortized server time $n^{3/4}$,
- client storage $n^{3/4}$ and no extra server storage,
- amortized client time $n^{1/2}$, and
- amortized communication $n^{1/2}$.

Throughout this talk, we omit $\log(n)$ and $\mathrm{poly}(\lambda)$ factors.

Theorem 1: From linearly homomorphic encryption.

Under DDH, QR, DCR, or LWE, there is a single-server PIR scheme that, on database size n, when the client makes $n^{1/4}$ adaptive queries, has:

- amortized server time $n^{3/4}$,
- client storage $n^{3/4}$ and no extra server storage,
- amortized client time $n^{1/2}$, and
- amortized communication $n^{1/2}$.

Theorem 2: From fully homomorphic encryption.

Assuming FHE*, we improve the amortized server time and client storage to $n^{1/2}$, if the client makes $n^{1/2}$ adaptive queries.

Throughout this talk, we omit $\log(n)$ and $\mathrm{poly}(\lambda)$ factors.

Theorem 1: From linearly homomorphic encryption.

Under DDH, QR, DCR, or LWE, there is a single-server PIR scheme that, on database size n, when the client makes $n^{1/4}$ adaptive queries, has:

- amortized server time $n^{3/4}$,
- client storage $n^{3/4}$ and no extra server storage,
- amortized client time $n^{1/2}$, and
- amortized communication $n^{1/2}$.

Theorem 2: From fully homomorphic encryption.

Assuming FHE*, we improve the amortized server time and client storage to $n^{1/2}$, if the client makes $n^{1/2}$ adaptive queries.

We prove that the tradeoff between server time + client storage is optimal.

Theorem 1: From linearly homomorphic encryption.

Under DDH, QR, DCR, or LWE, there is a single-server PIR scheme that, on database size n, when the client makes $n^{1/4}$ adaptive queries, has:

- amortized server time $n^{3/4}$,
- client storage $n^{3/4}$ and no extra server storage,
- amortized client time $n^{1/2}$, and
- amortized communication $n^{1/2}$.

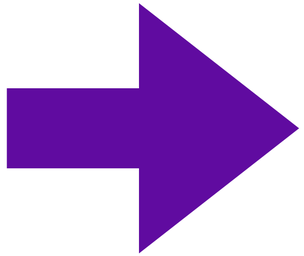
Theorem 2: From fully homomorphic encryption.

Assuming FHE*, we improve the amortized server time and client storage to $n^{1/2}$, if the client makes $n^{1/2}$ adaptive queries.

We prove that the tradeoff between server time + client storage is optimal.

Proof sketch for Theorem 1

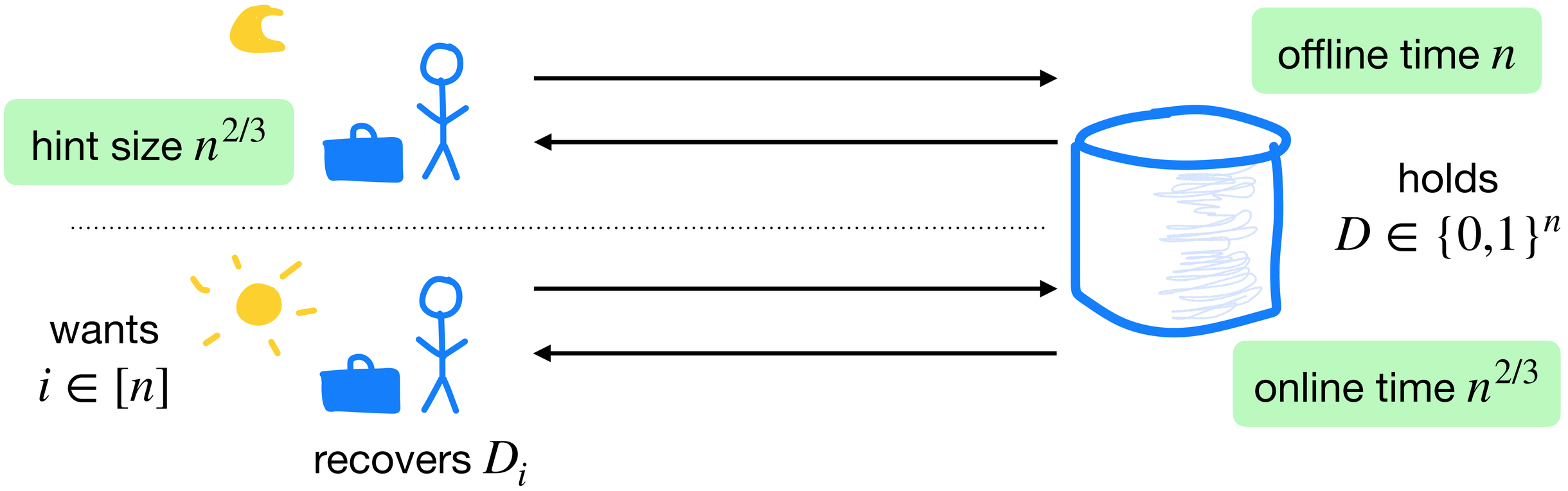
Single-query PIR with sublinear online time
[CK20]



New: Many-query PIR with sublinear amortized time

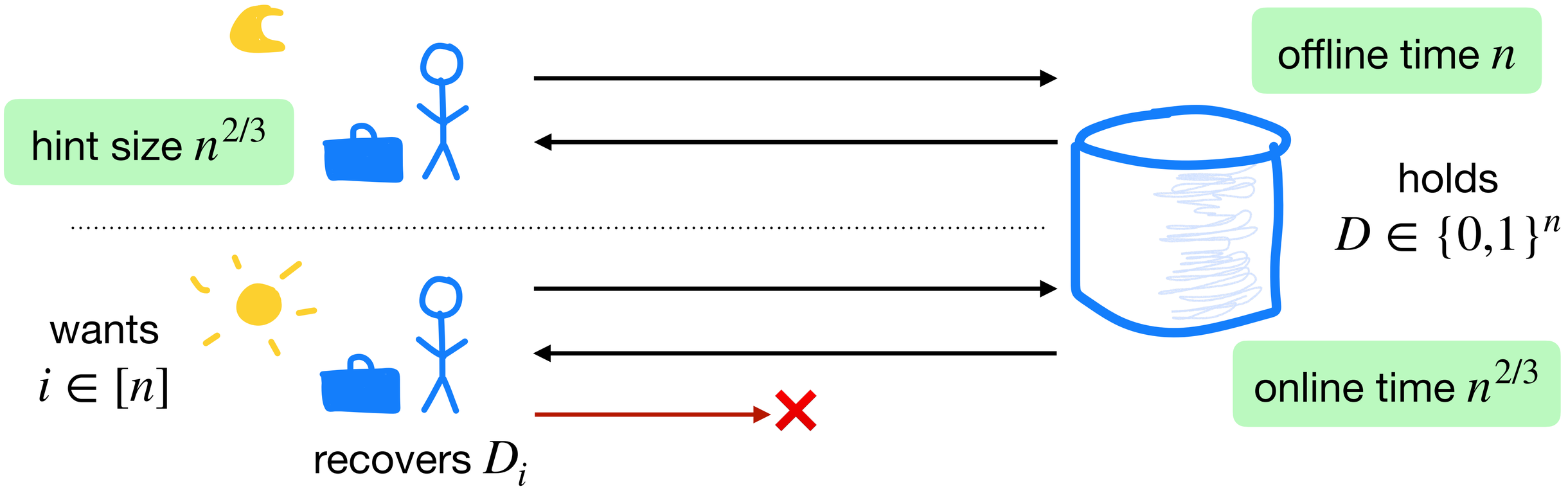
New: generic compiler,
applying ideas from batch codes
[IKOS04]

Single-query PIR with sublinear online time [CK20]



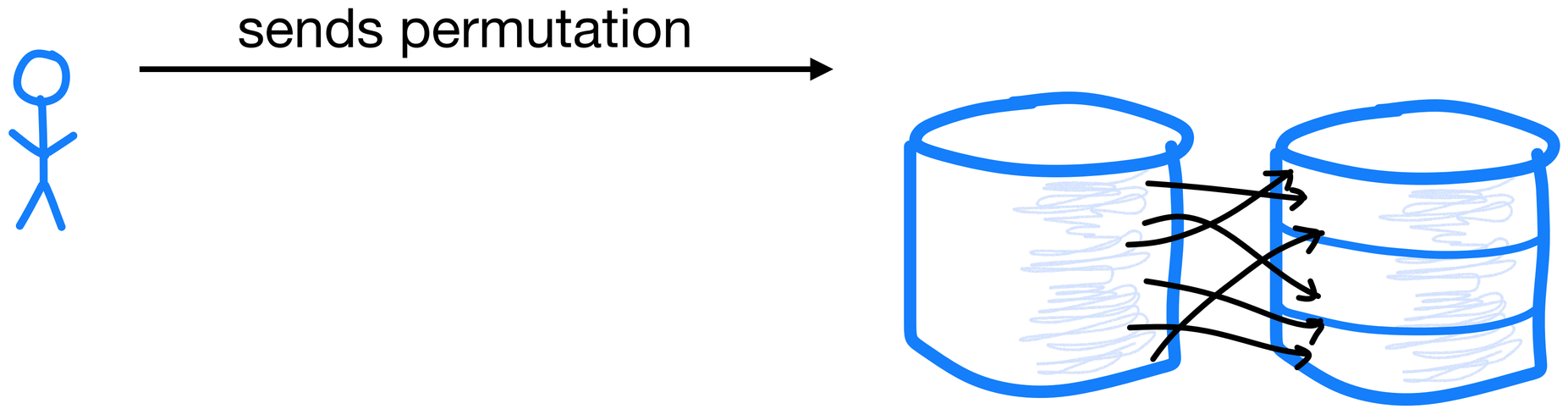
Assuming DDH, QR, DCR, or LWE

Single-query PIR with sublinear online time [CK20]



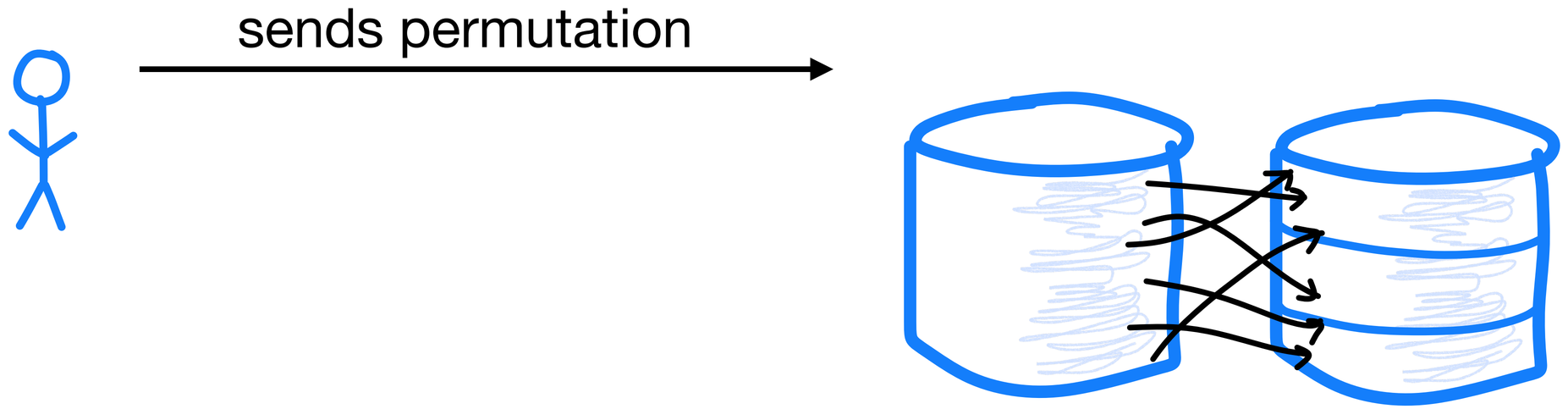
Assuming DDH, QR, DCR, or LWE

Our compiler: To handle Q adaptive queries, split the database into Q random chunks.



holds $D \in \{0,1\}^n$,
in Q chunks of size n/Q

Our compiler: To handle Q adaptive queries, split the database into Q random chunks.



holds $D \in \{0,1\}^n$,
in Q chunks of size n/Q

Observation: When the client makes Q adaptive queries,
at most λ distinct queries fall in any one chunk,
with probability $1 - \mathrm{negl}(\lambda)$.

Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

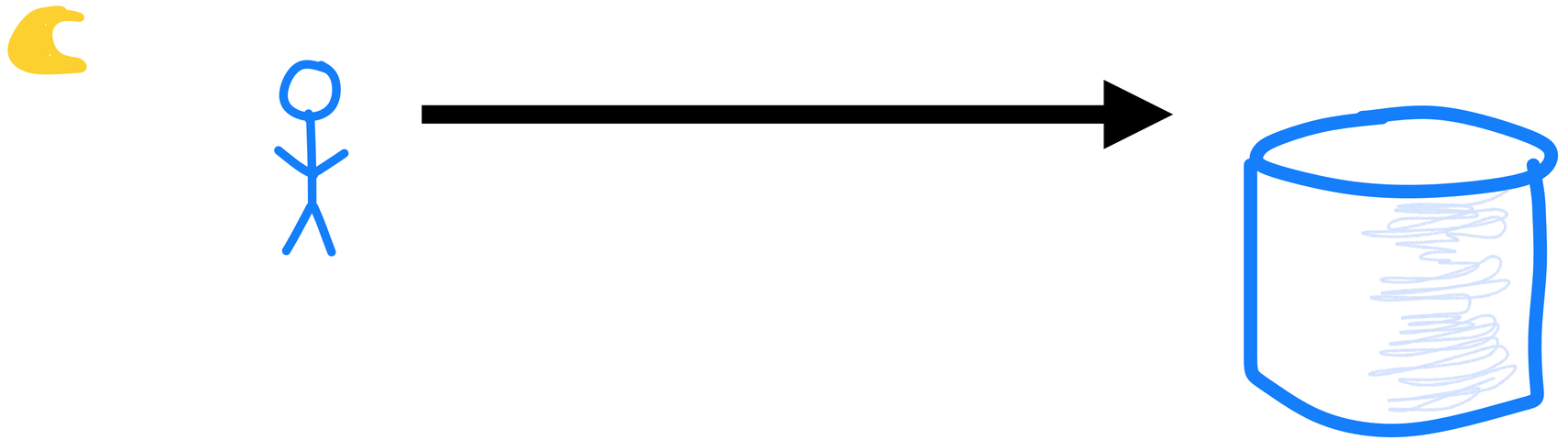
The client caches all recovered bits, to never re-query for the same index.

holds $D \in \{0,1\}^n$

Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

The client caches all recovered bits, to never re-query for the same index.

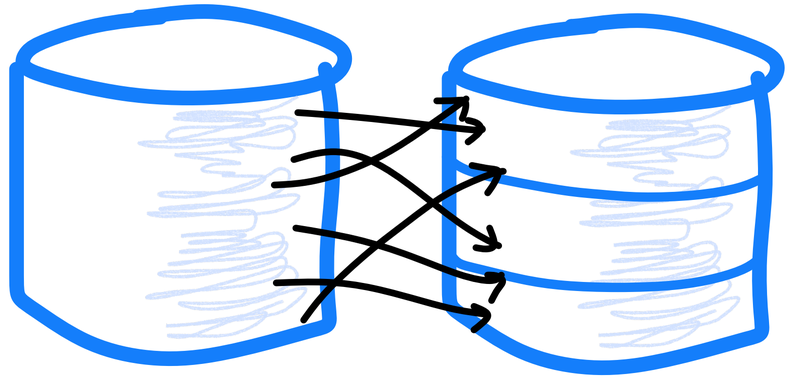


holds $D \in \{0,1\}^n$

Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

The client caches all recovered bits, to never re-query for the same index.

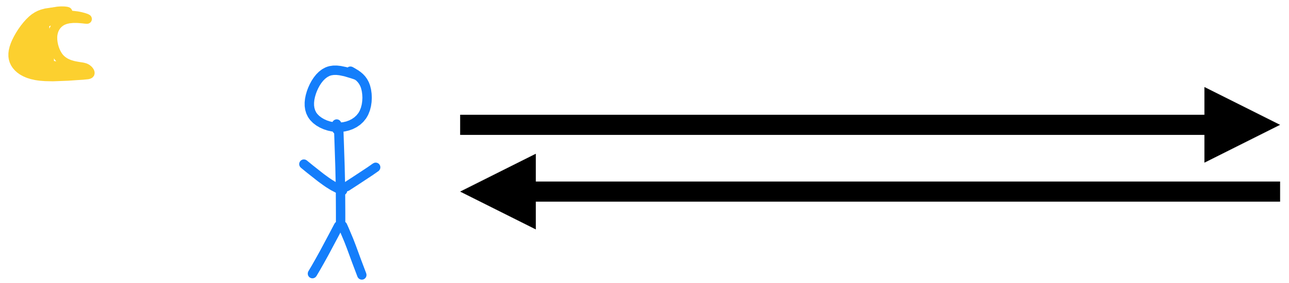


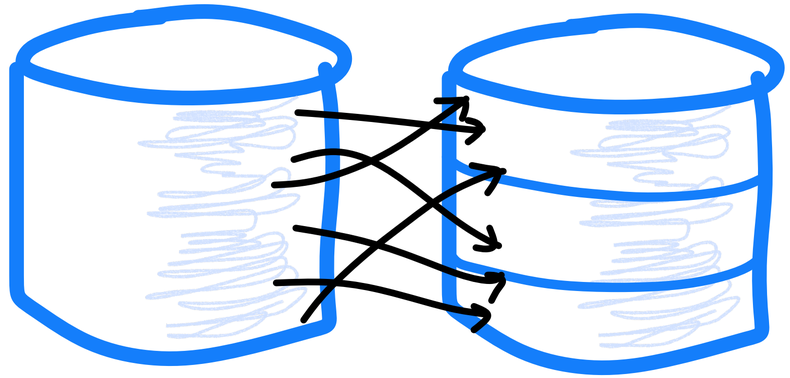
holds $D \in \{0,1\}^n$,
in Q chunks of size n/Q

Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

The client caches all recovered bits, to never re-query for the same index.



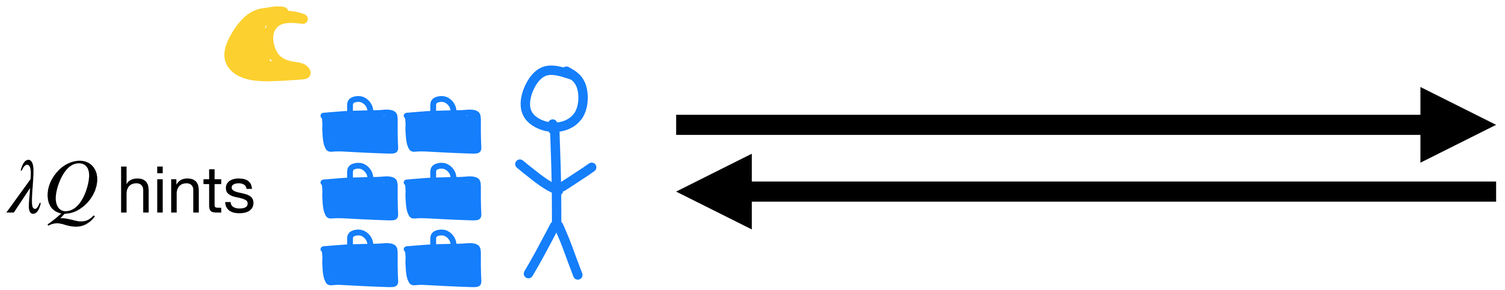


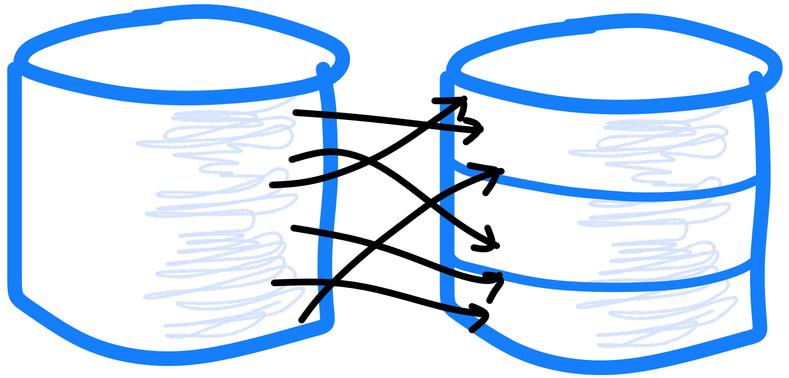
holds $D \in \{0,1\}^n$,
in Q chunks of size n/Q

Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

The client caches all recovered bits, to never re-query for the same index.



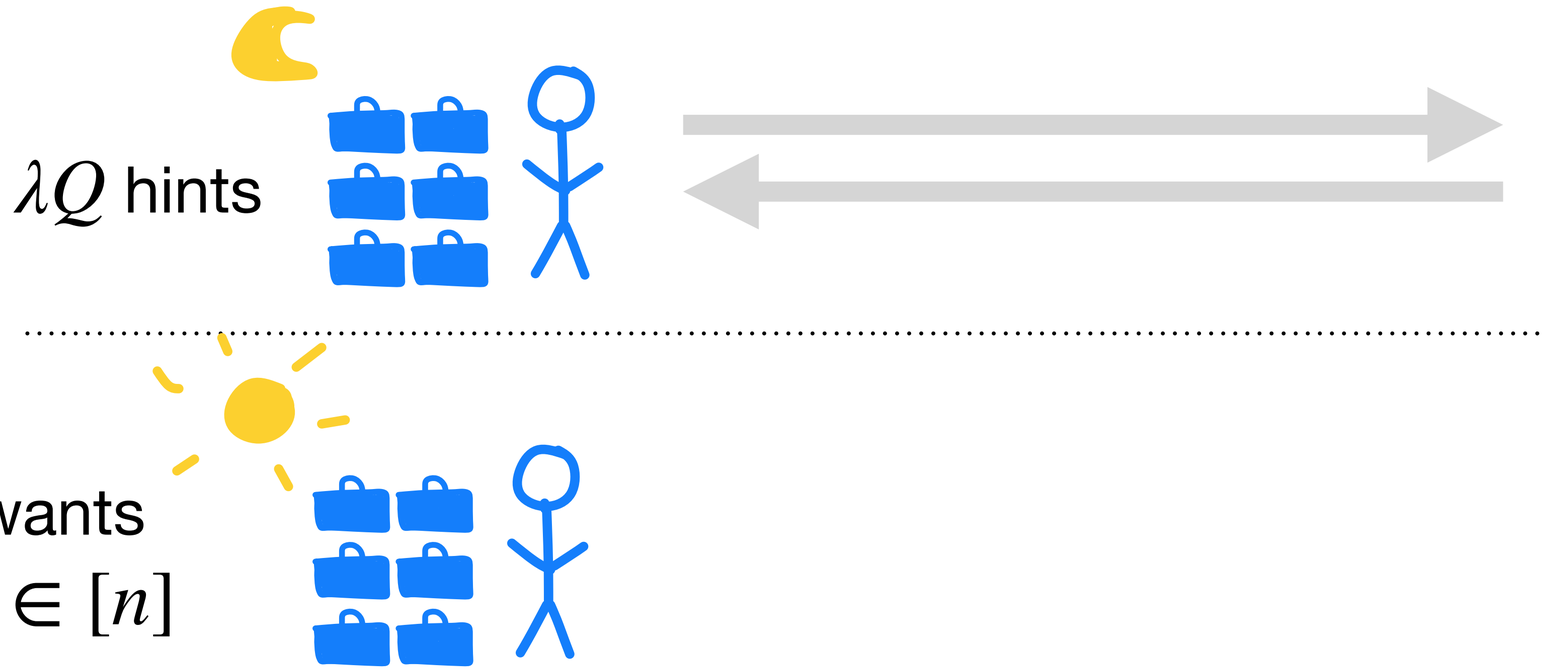


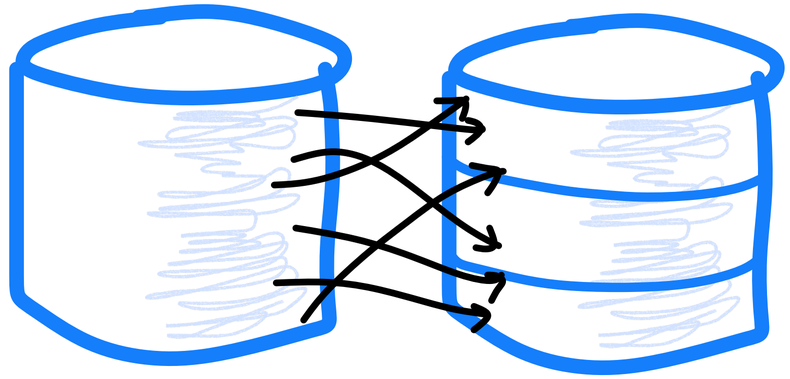
holds $D \in \{0,1\}^n$,
in Q chunks of size n/Q

Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

The client caches all recovered bits, to never re-query for the same index.



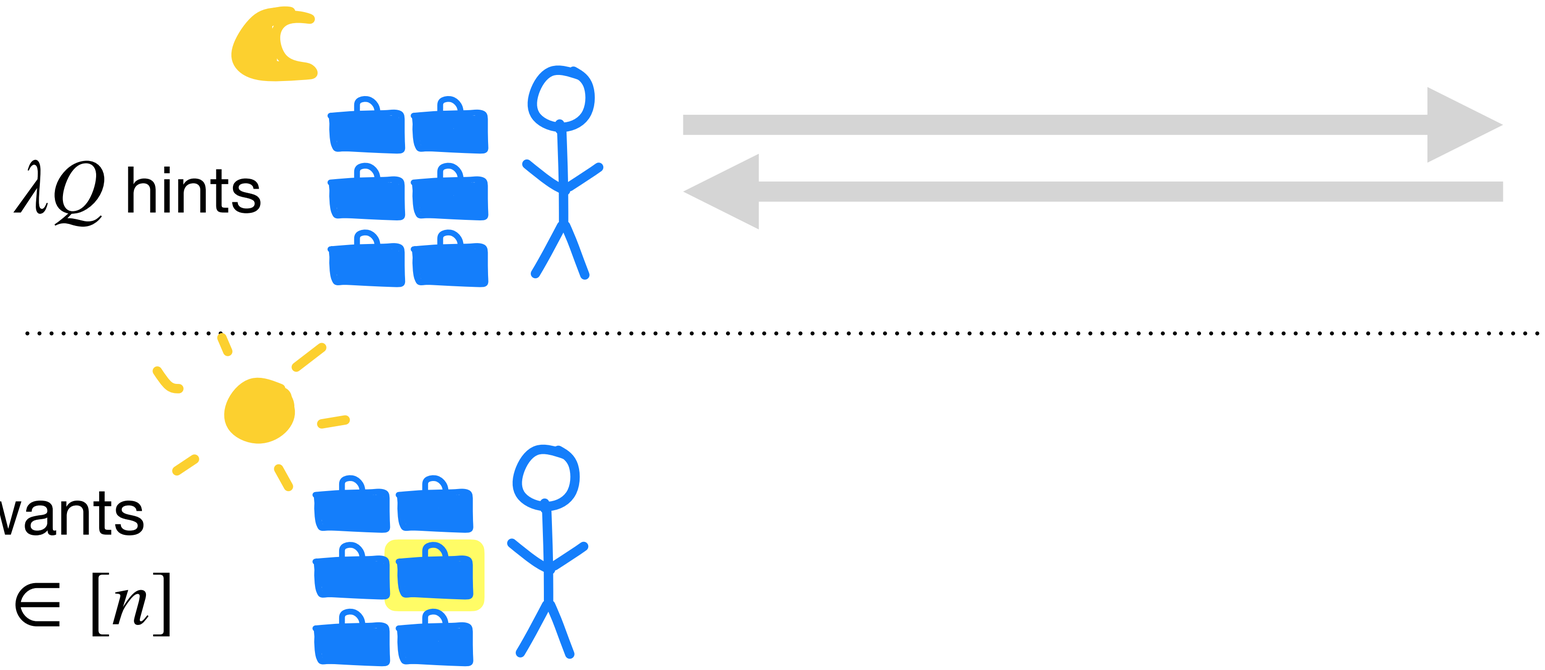


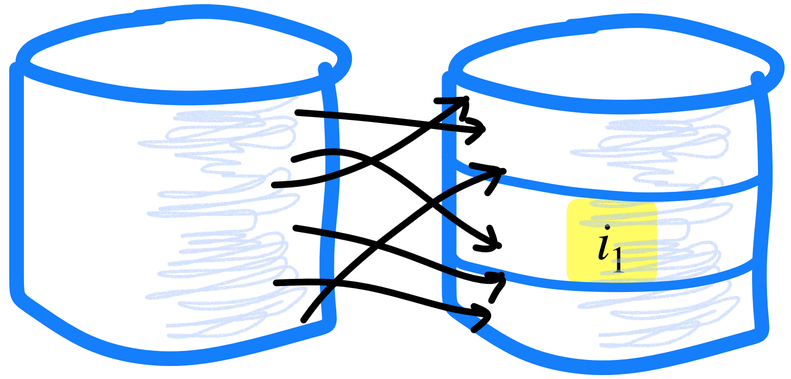
holds $D \in \{0,1\}^n$,
in Q chunks of size n/Q

Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

The client caches all recovered bits, to never re-query for the same index.



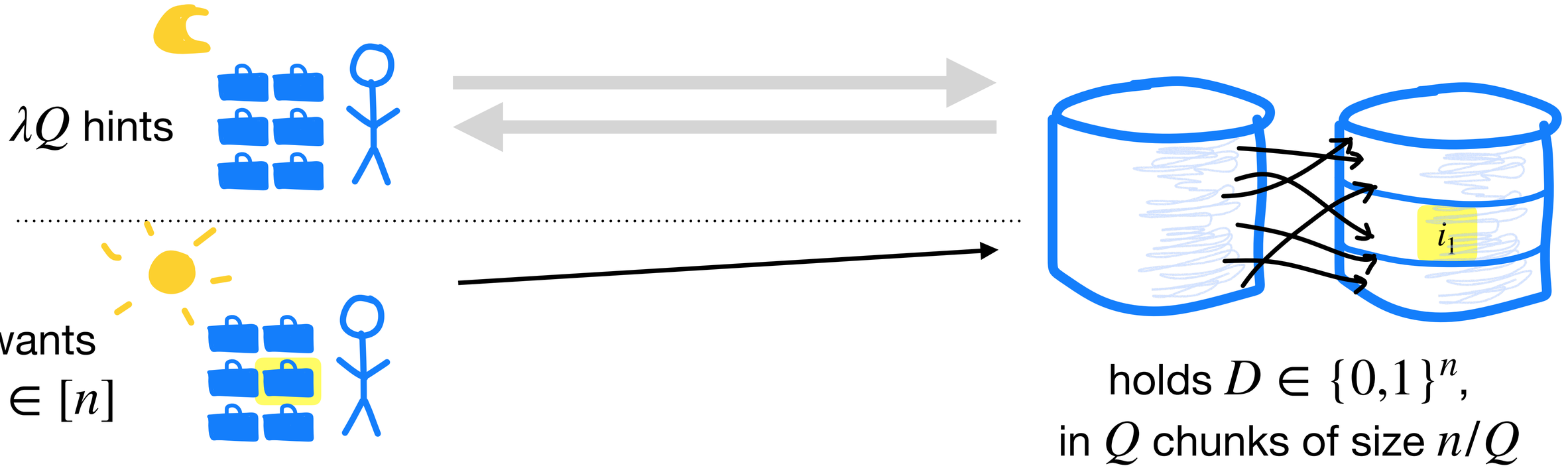


holds $D \in \{0,1\}^n$,
in Q chunks of size n/Q

Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

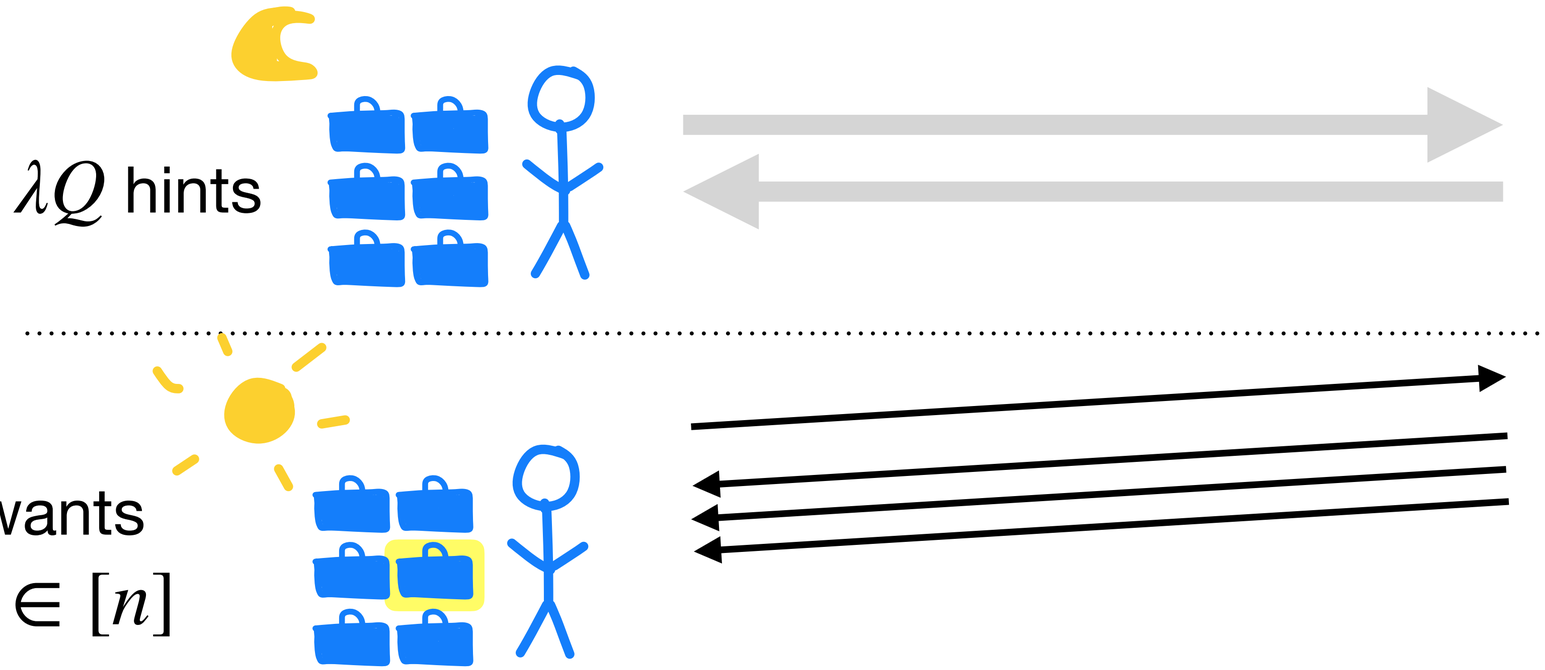
The client caches all recovered bits, to never re-query for the same index.

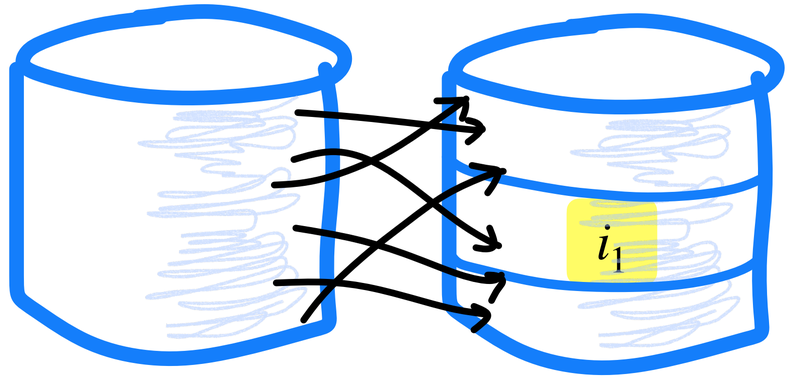


Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

The client caches all recovered bits, to never re-query for the same index.



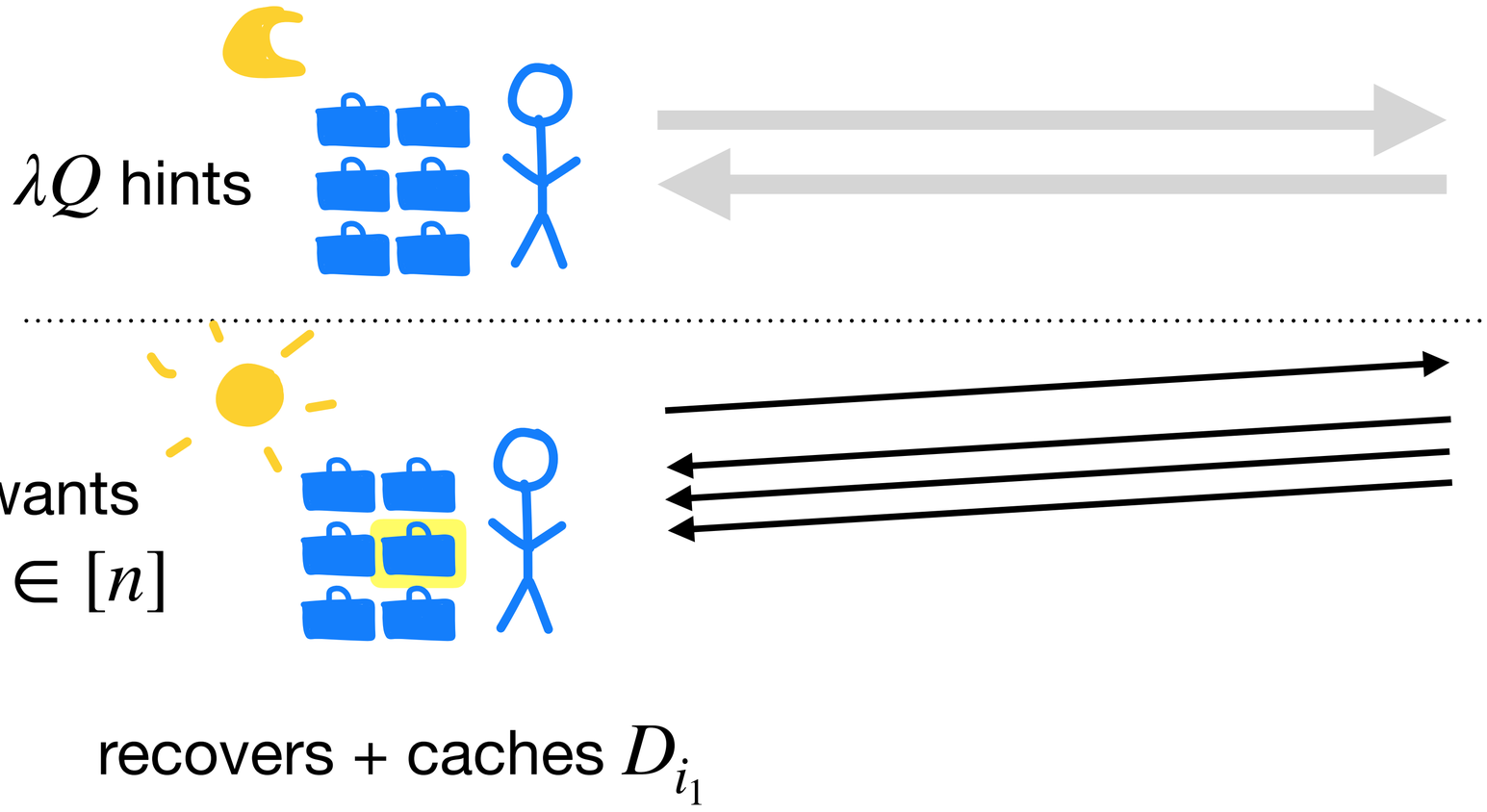


holds $D \in \{0,1\}^n$,
in Q chunks of size n/Q

Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

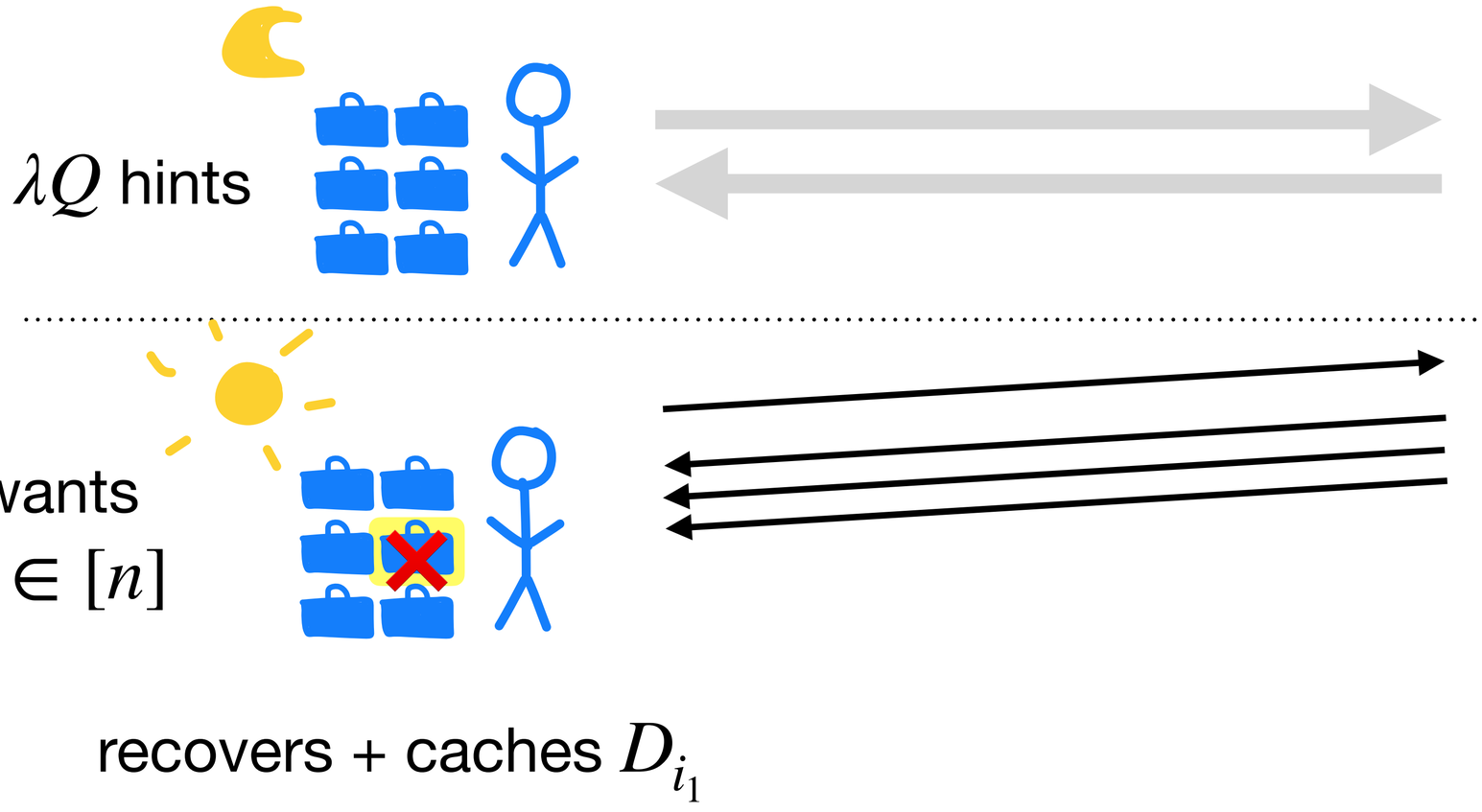
The client caches all recovered bits, to never re-query for the same index.



Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

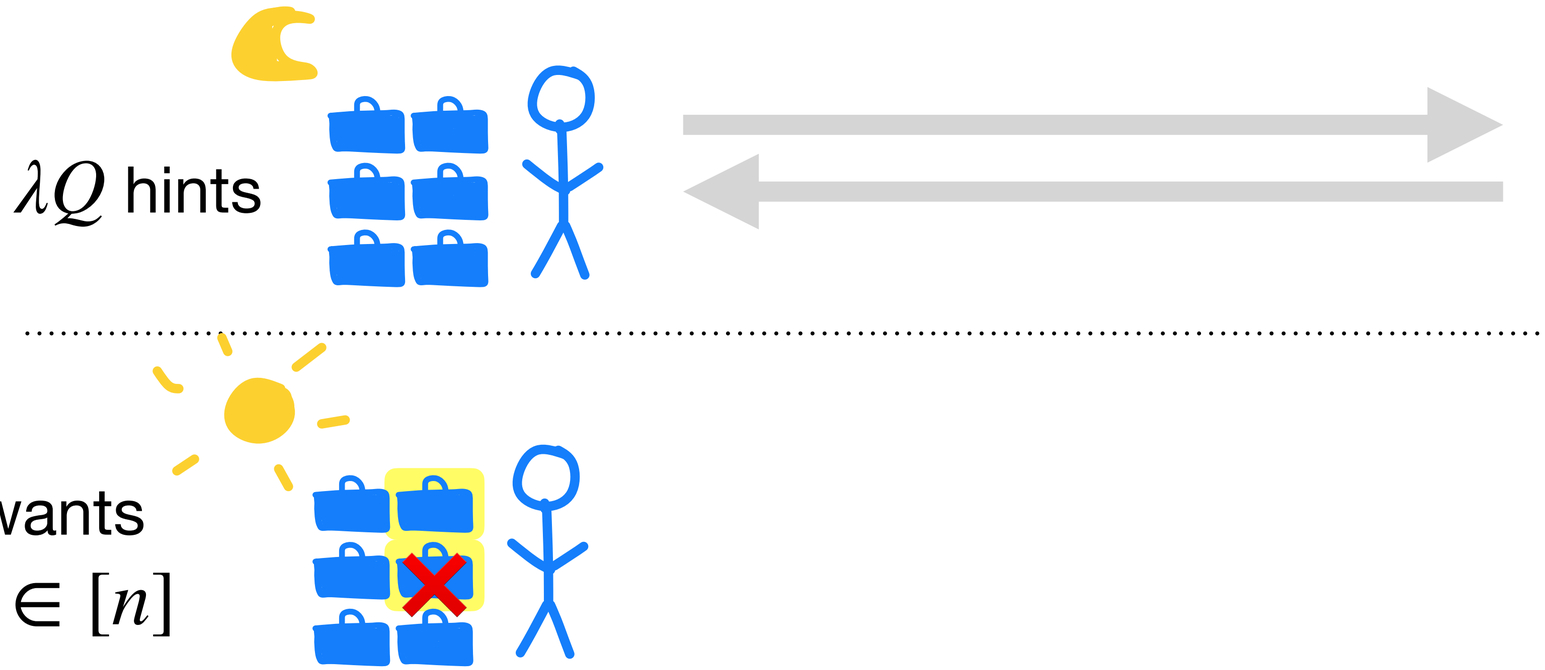
The client caches all recovered bits, to never re-query for the same index.

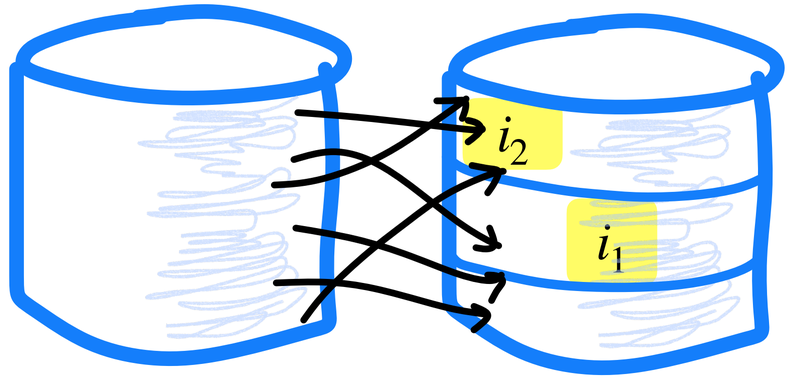


Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

The client caches all recovered bits, to never re-query for the same index.



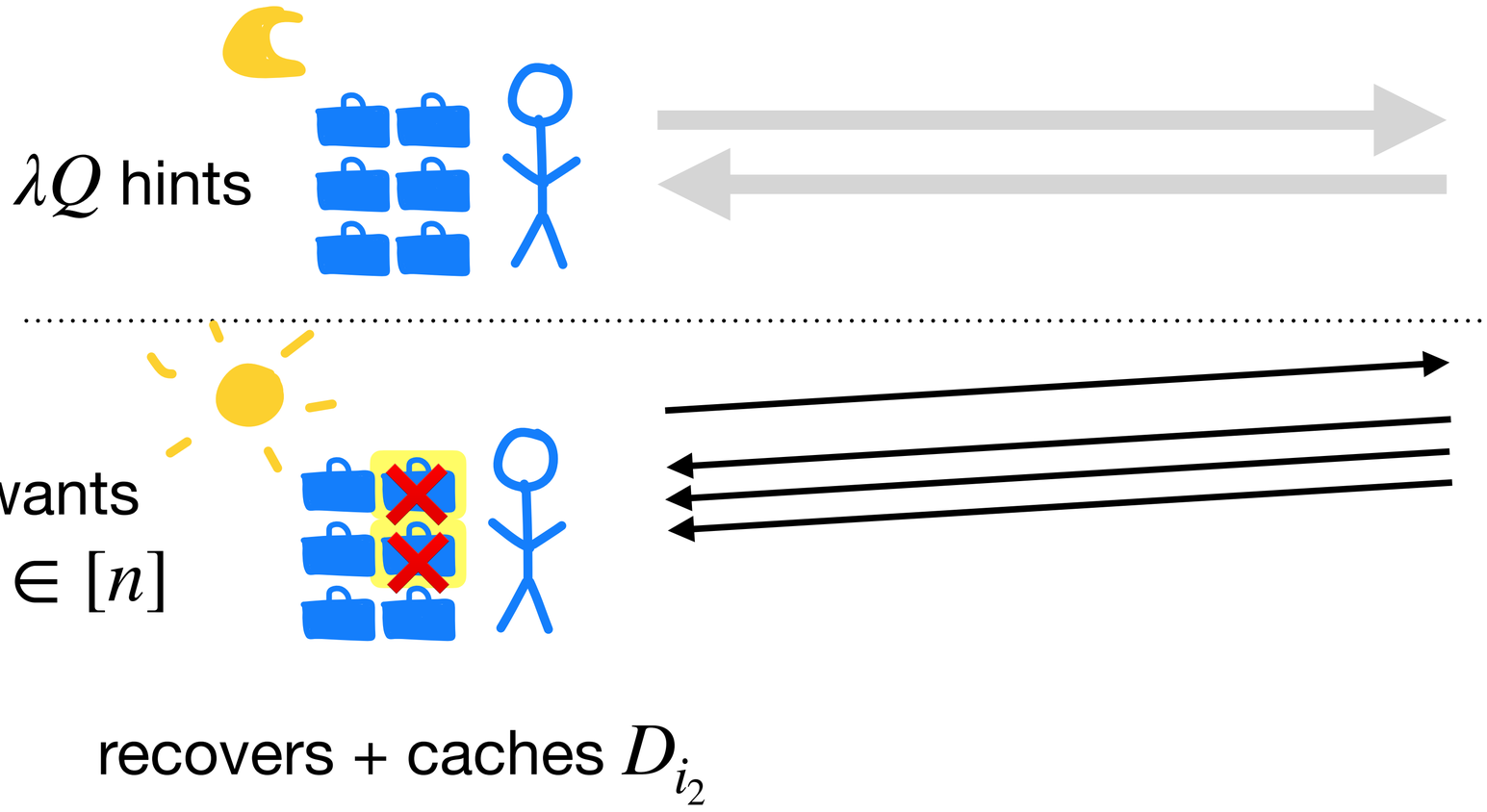


holds $D \in \{0,1\}^n$,
in Q chunks of size n/Q

Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

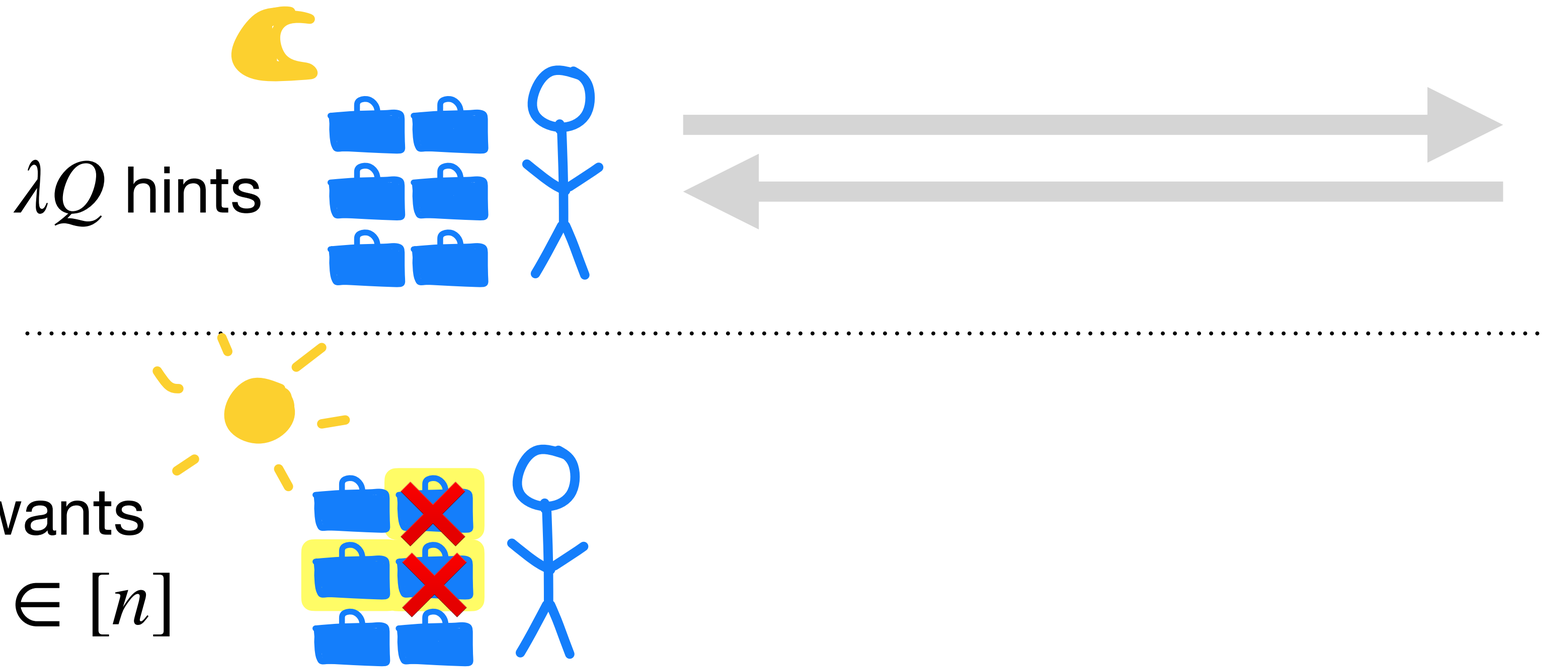
The client caches all recovered bits, to never re-query for the same index.

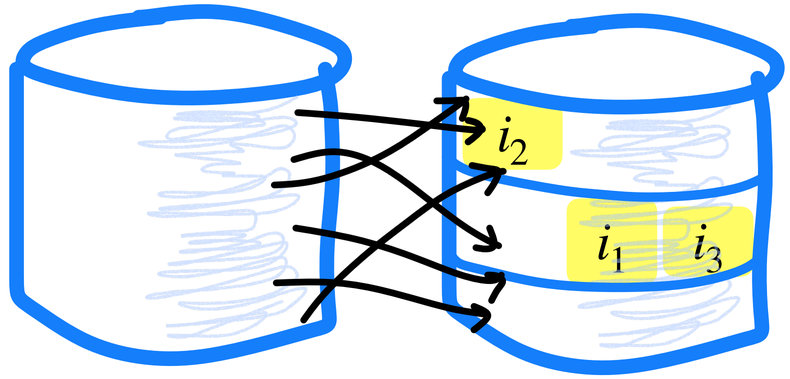


Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

The client caches all recovered bits, to never re-query for the same index.



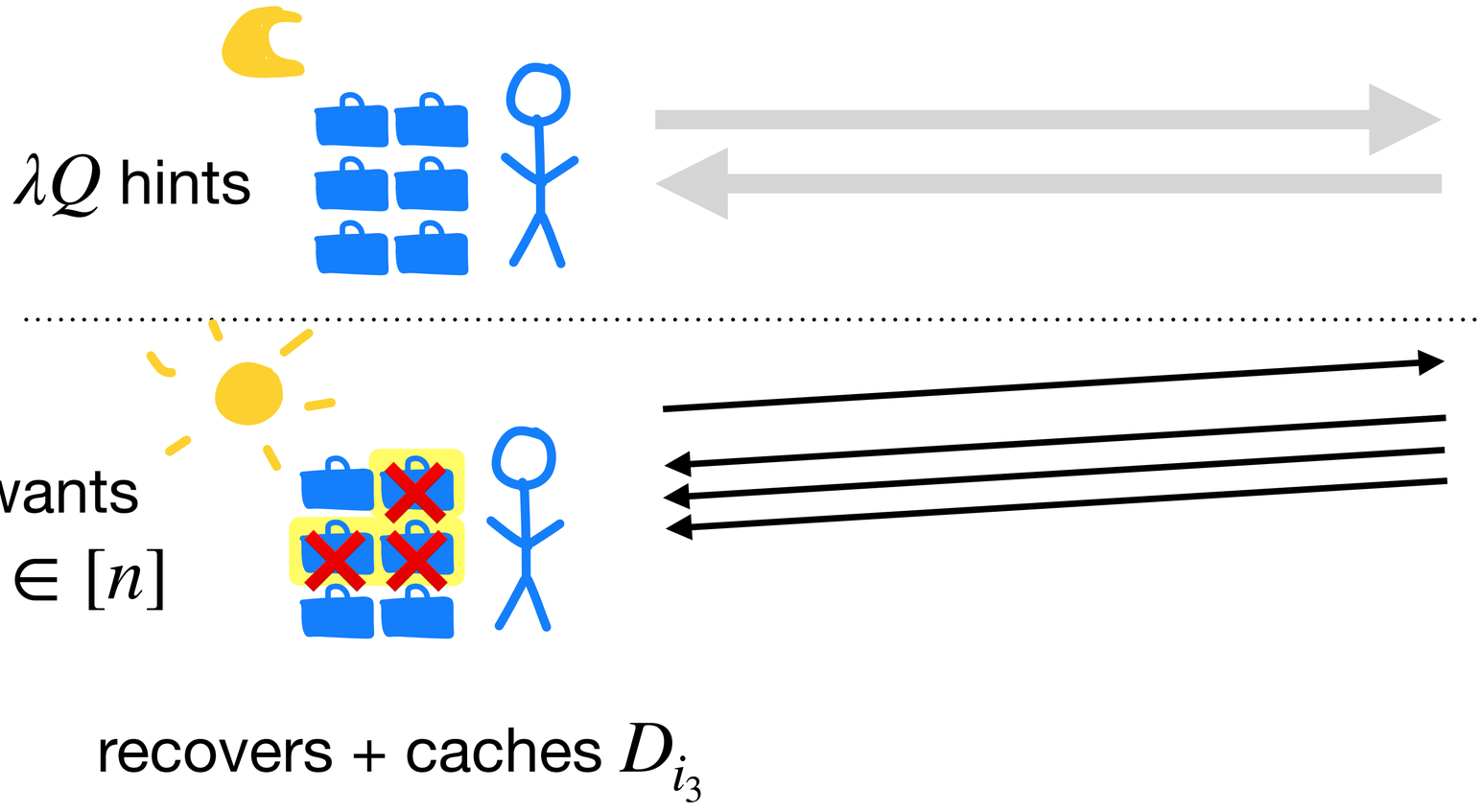


holds $D \in \{0,1\}^n$,
in Q chunks of size n/Q

Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

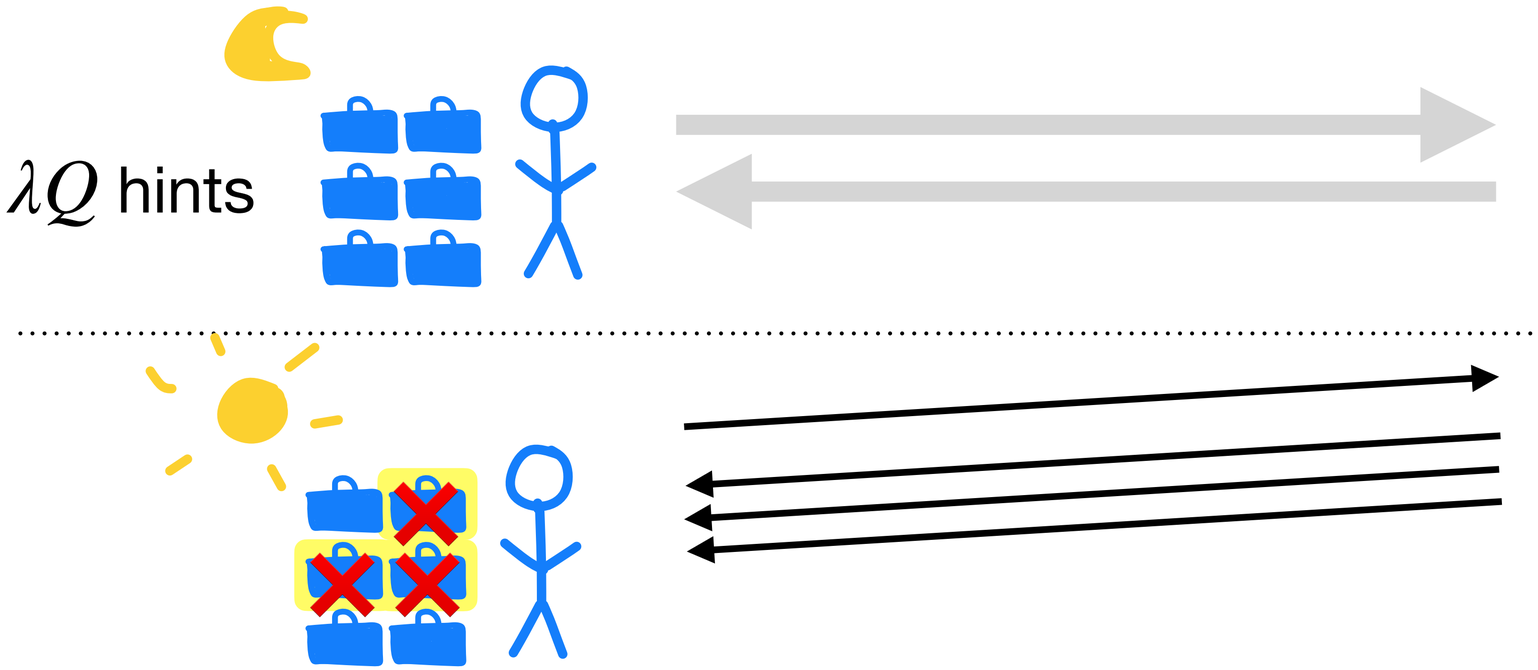
The client caches all recovered bits, to never re-query for the same index.

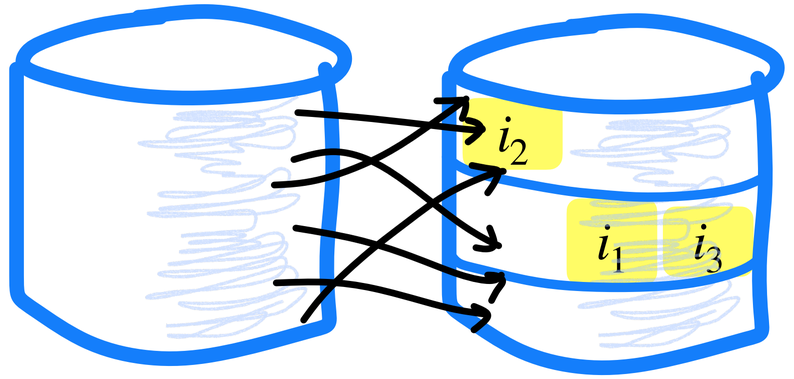


Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

The client caches all recovered bits, to never re-query for the same index.





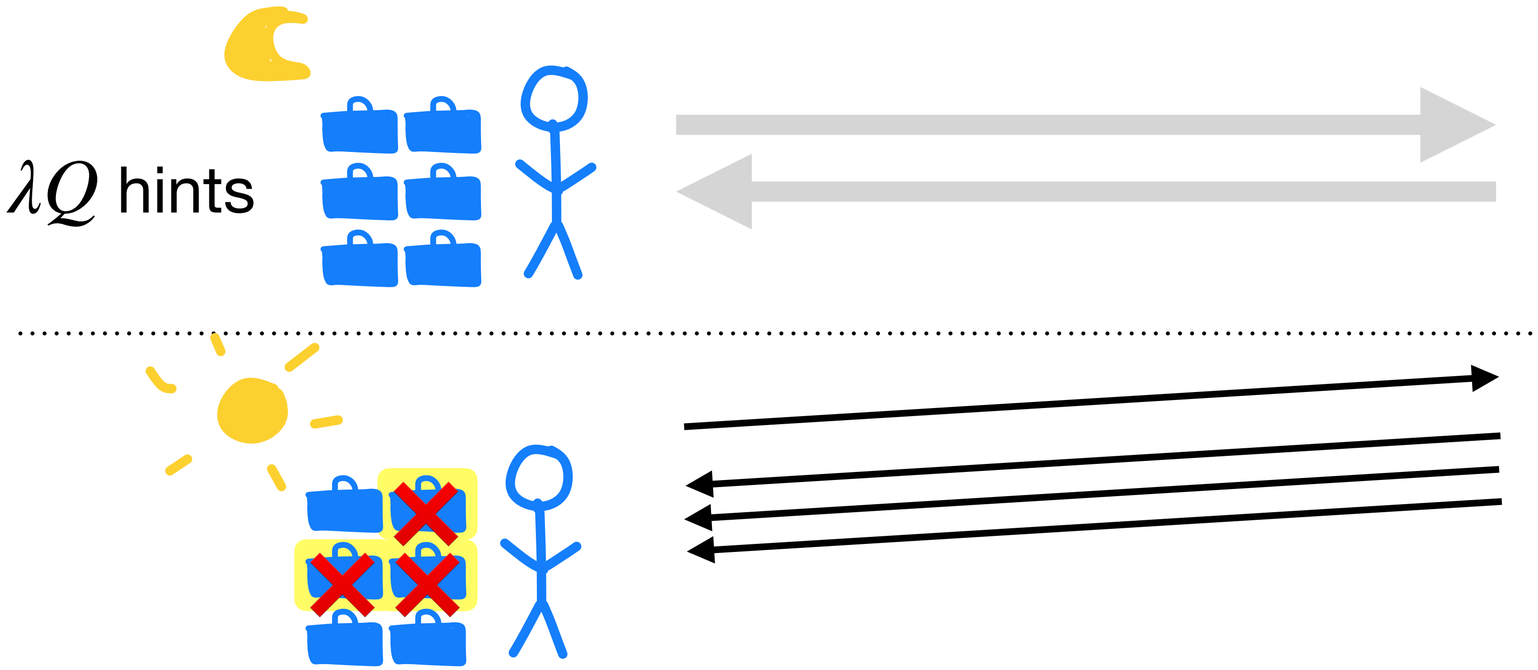
holds $D \in \{0,1\}^n$,
in Q chunks of size n/Q

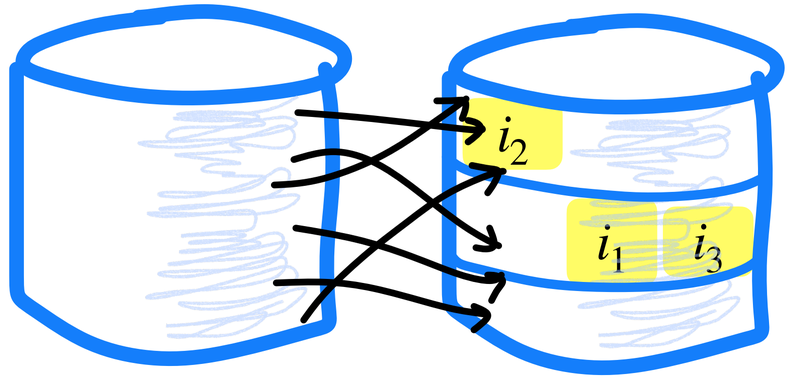
Correctness: for any query sequence, the client does not run out of fresh hints (with overwhelming probability over the choice of permutation).

Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

The client caches all recovered bits, to never re-query for the same index.





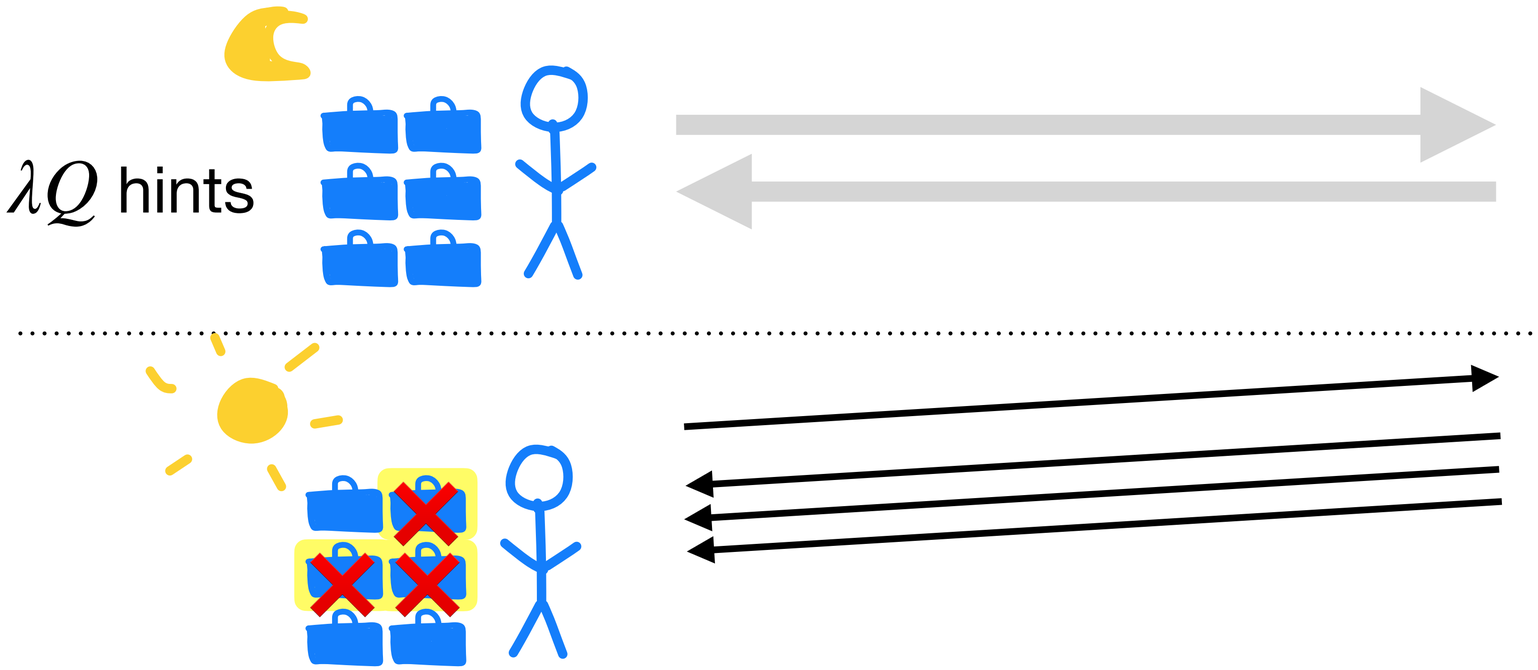
holds $D \in \{0,1\}^n$,
in Q chunks of size n/Q

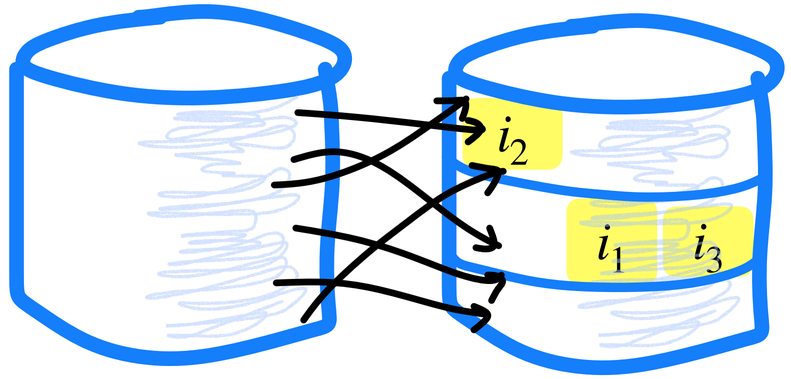
Security: The client's query does not reveal which chunk it is reading.

Offline: Permute + partition the database, then run λ offline phases on each chunk.

Online: Run an online phase on each chunk, using a hint matching the index.

The client caches all recovered bits, to never re-query for the same index.





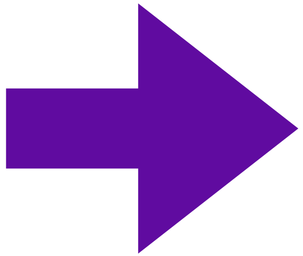
holds $D \in \{0,1\}^n$,
in Q chunks of size n/Q

Cost: We ran the underlying PIR λQ times, on database size n/Q.

Proof sketch for Theorem 1

Input: Single-query PIR with sublinear online time [CK20]

- hint size $n^{2/3}$
- offline time n
- online time $n^{2/3}$



Generic compiler, with $Q = n^{1/4}$ queries

Output: Many-query PIR with sublinear amortized time

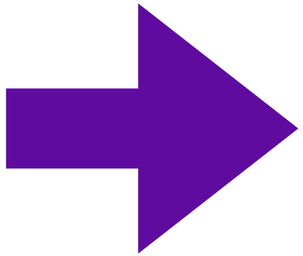
- hint size $n^{3/4}$
- offline time n
- online time $n^{3/4}$

Throughout this talk, we omit $\log(n)$ and $\mathrm{poly}(\lambda)$ factors.

Proof sketch for Theorem 1

Input: Single-query PIR with sublinear online time [CK20]

- hint size $n^{2/3}$
- offline time n
- online time $n^{2/3}$



Generic compiler, with $Q = n^{1/4}$ queries

Output: Many-query PIR with sublinear amortized time

- hint size $n^{3/4}$
- offline time n
- online time $n^{3/4}$

- amortized time $n^{3/4}$

Throughout this talk, we omit $\log(n)$ and $\mathrm{poly}(\lambda)$ factors.

Theorem 1: From linearly homomorphic encryption.

Under DDH, QR, DCR, or LWE, there is a single-server PIR scheme that, on database size n, when the client makes $\geq n^{1/4}$ adaptive queries, has:

- amortized server time $n^{3/4}$,
- client storage $n^{3/4}$ and no extra server storage,
- amortized client time $n^{1/2}$, and
- amortized communication $n^{1/2}$.

Theorem 2: From fully homomorphic encryption.

Assuming FHE, we improve the amortized server time and client storage to $n^{1/2}$, if the client makes $n^{1/2}$ adaptive queries.

Theorem 1: From linearly homomorphic encryption.

Under DDH, QR, DCR, or LWE, there is a single-server PIR scheme that, on database size n, when the client makes $\geq n^{1/4}$ adaptive queries, has:

- amortized server time $n^{3/4}$,
- client storage $n^{3/4}$ and no extra server storage,
- amortized client time $n^{1/2}$, and
- amortized communication $n^{1/2}$.

Theorem 2: From fully homomorphic encryption.

Assuming FHE, we improve the amortized server time and client storage to $n^{1/2}$, if the client makes $n^{1/2}$ adaptive queries.

Proof idea for Theorem 2

Informal claim 1.

Given the parities of $O(Q)$ random, size-n/Q subsets of the database, the client can make Q adaptive queries with online time n/Q.

Prior work [CK20,SACM21,KC21] only supports one adaptive query.

Proof idea for Theorem 2

Informal claim 1.

Given the parities of $O(Q)$ random, size-n/Q subsets of the database, the client can make Q adaptive queries with online time n/Q.

Prior work [CK20,SACM21,KC21] only supports one adaptive query.

Informal claim 2.

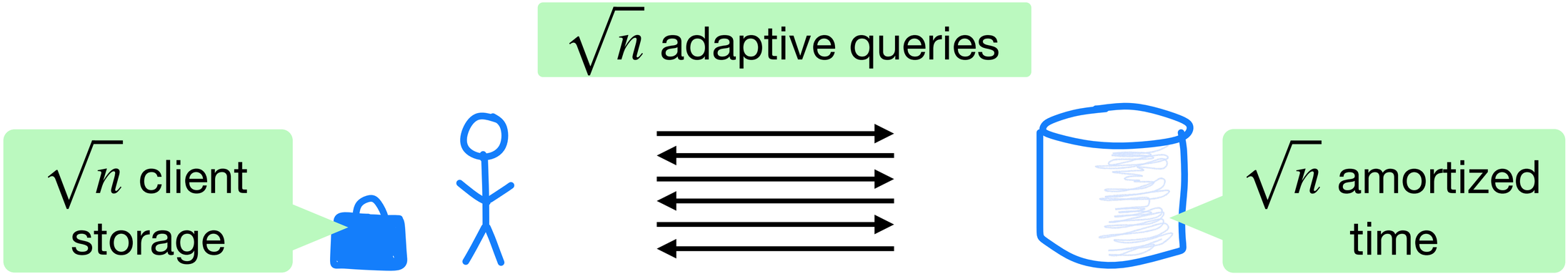
We give a Boolean circuit for retrieving the parities of $O(Q)$ subsets of the database, each of size n/Q, in $O(n)$ gates.

In the offline phase, the server runs the circuit under FHE in linear time.

This talk

1. Background: The offline/online PIR model
2. Our results: New PIR schemes with sublinear time
3. Open questions

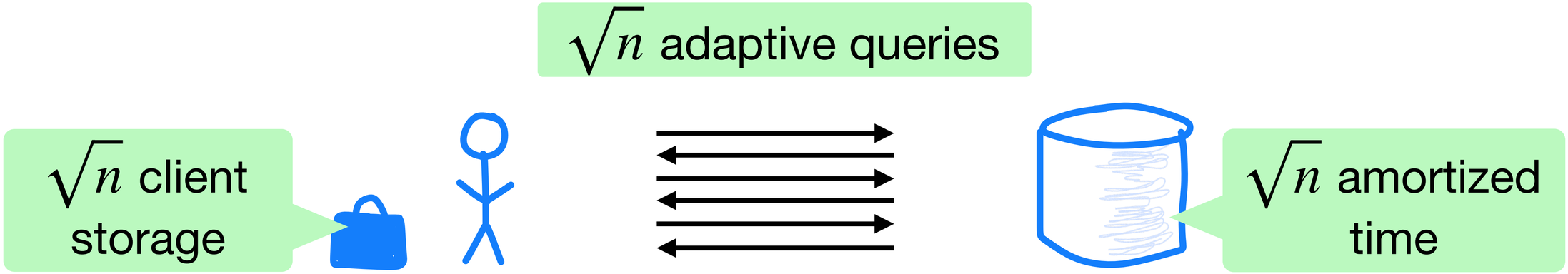
Adaptive single-server PIR with sublinear time + storage is feasible:



But, these schemes are not yet efficient enough for use in practice.

- Follow-up work [ZLTS22] improves the communication to $\tilde{O}_\lambda(1)$.
- Can we construct optimal schemes from assumptions weaker than FHE?
- Can we beat our lower bounds by having the server encode the database?

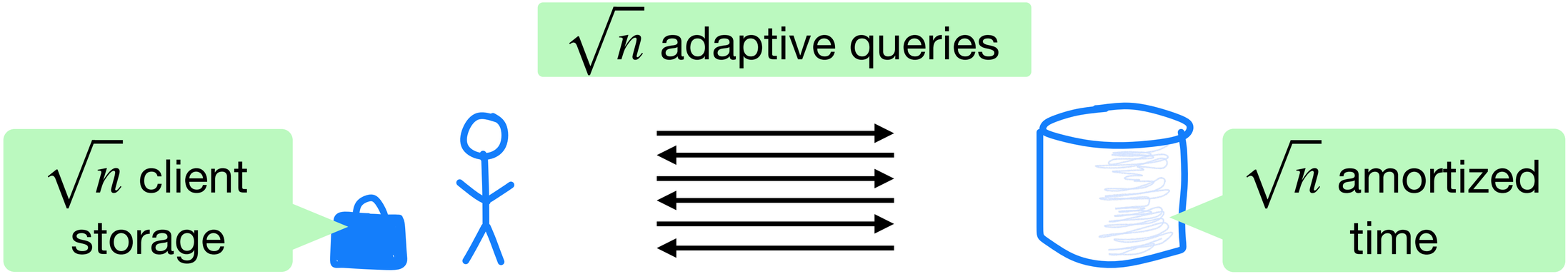
Adaptive single-server PIR with sublinear time + storage is feasible:



But, these schemes are not yet efficient enough for use in practice.

- ➤ Follow-up work [ZLTS22] improves the communication to $\tilde{O}_\lambda(1)$.
- ➤ Can we construct optimal schemes from assumptions weaker than FHE?
- ➤ Can we beat our lower bounds by having the server encode the database?

Adaptive single-server PIR with sublinear time + storage is feasible:



But, these schemes are not yet efficient enough for use in practice.

- ➤ Follow-up work [ZLTS22] improves the communication to $\tilde{O}_\lambda(1)$.
- ➤ Can we construct optimal schemes from assumptions weaker than FHE?
- ➤ Can we beat our lower bounds by having the server encode the database?

ahenz@csail.mit.edu — https://eprint.iacr.org/2022/081