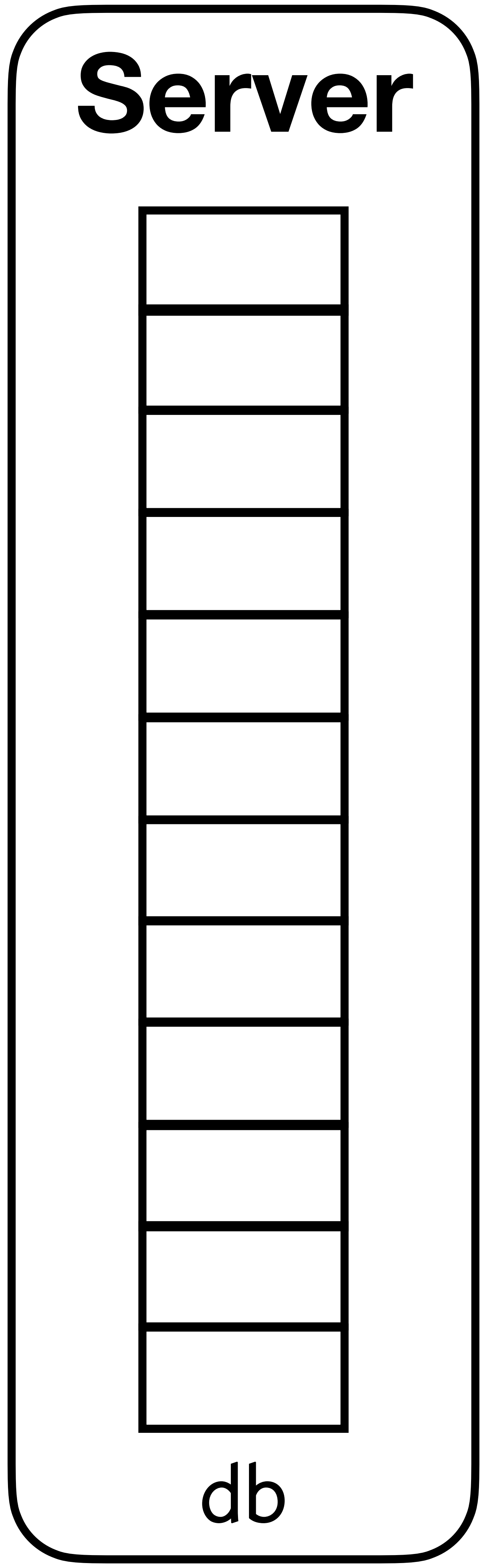
Maliciously-secure PIR (almost) for free

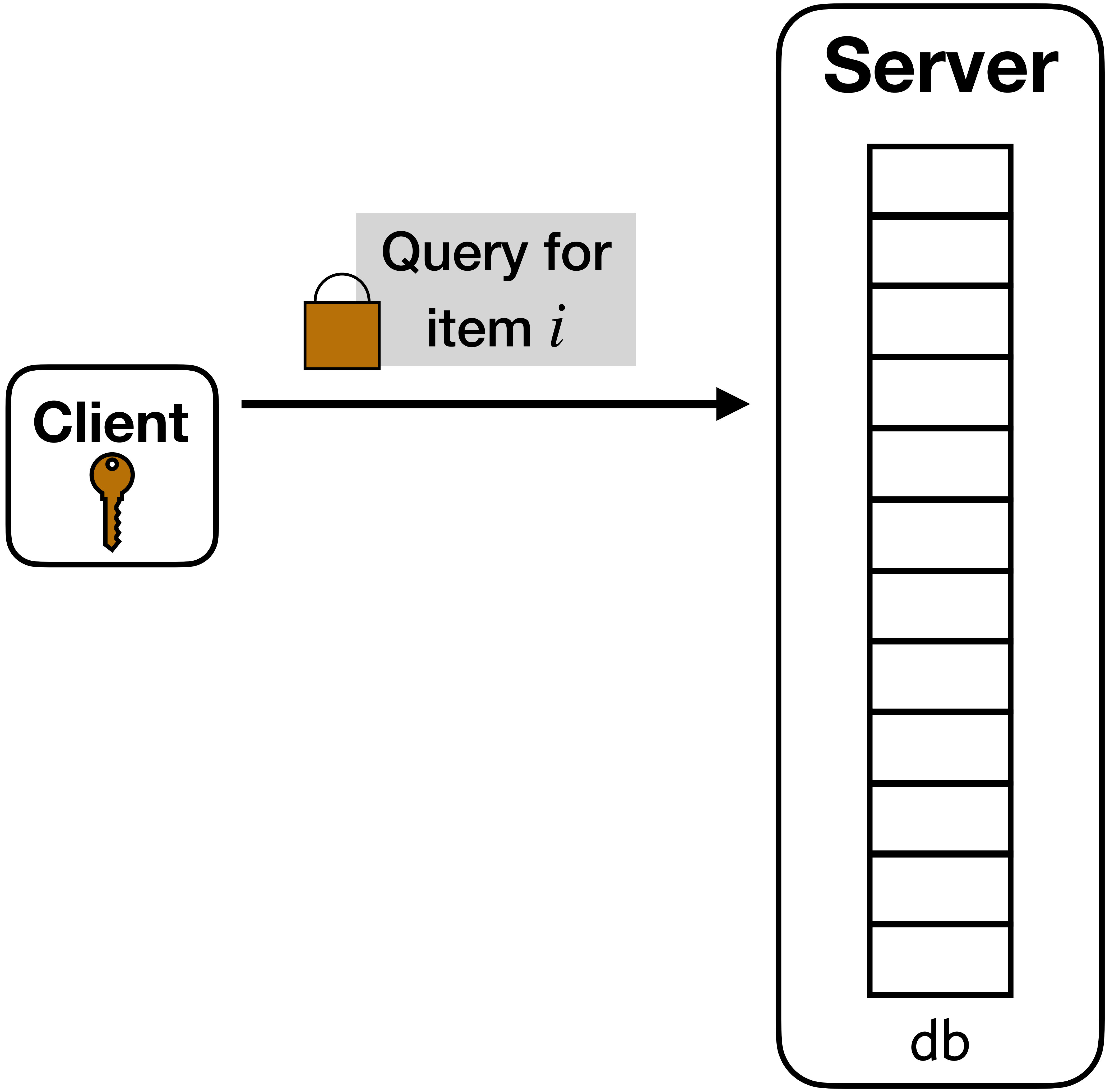
Brett Falk | ***Pratyush Mishra*** | **Matan Shtepel**

UPenn | UPenn | CMU

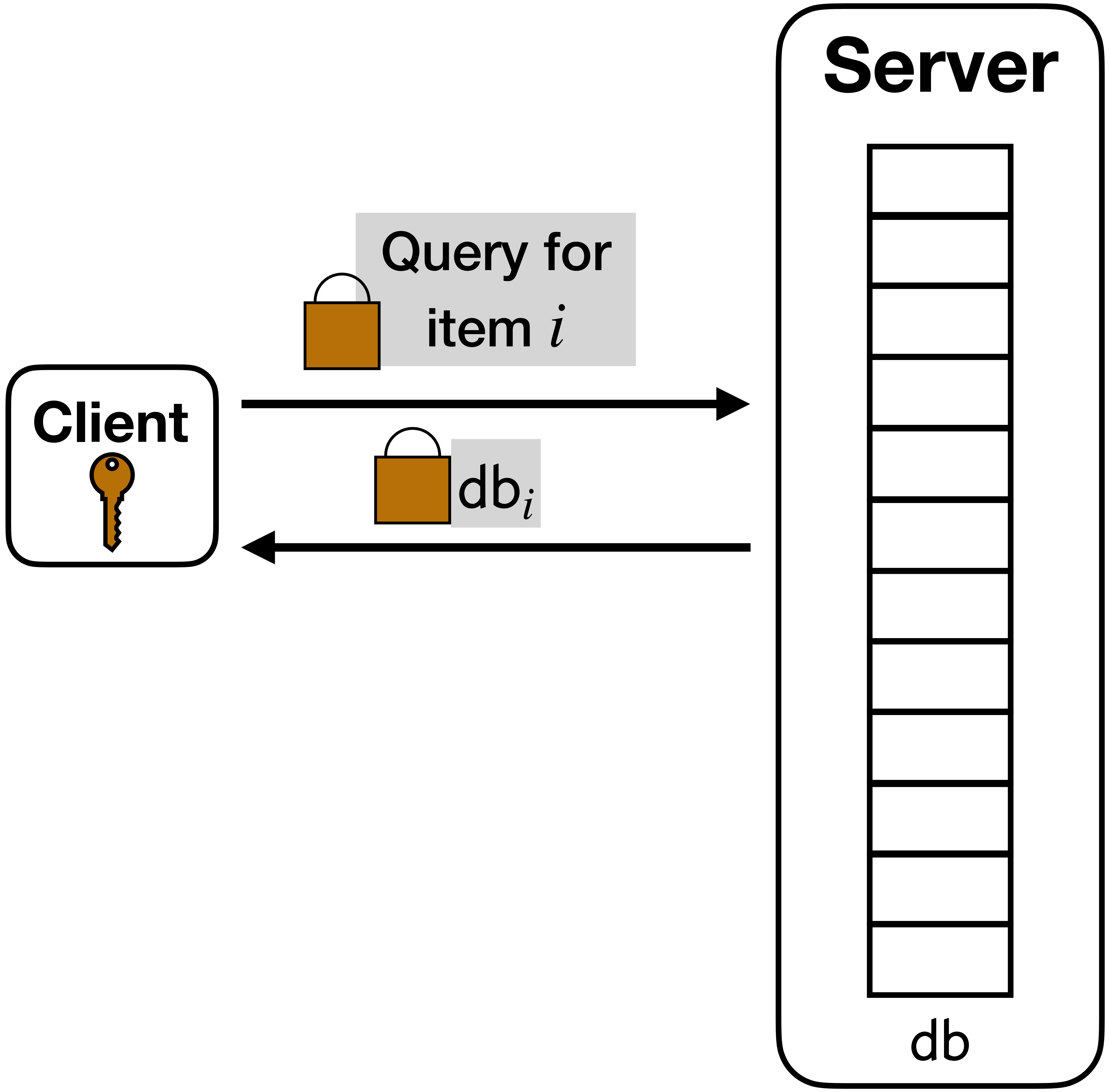
Private Information Retrieval (PIR) [CKSG95, KO97]



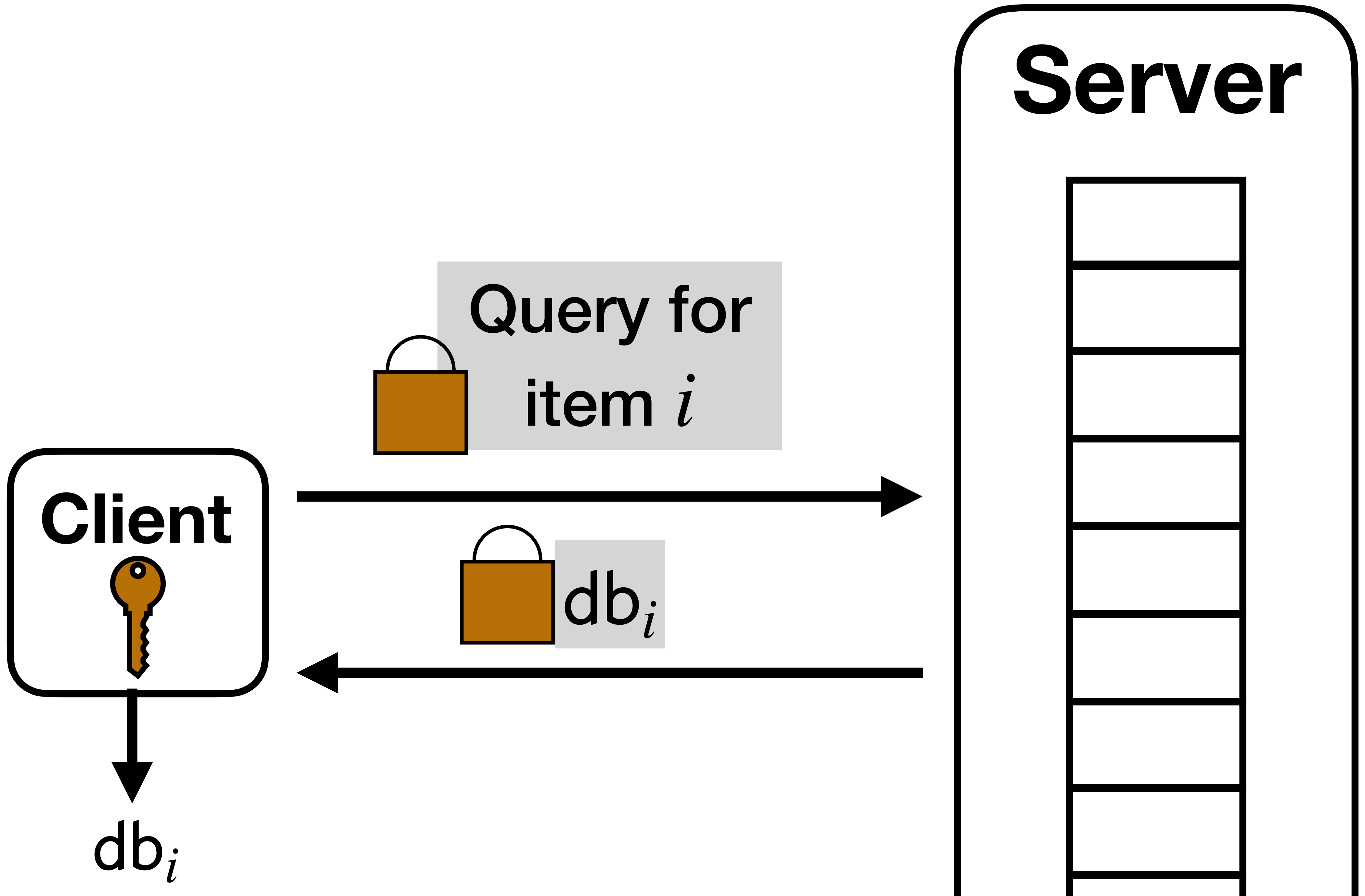
Private Information Retrieval (PIR) [CKSG95, KO97]



Private Information Retrieval (PIR) [CKSG95, KO97]

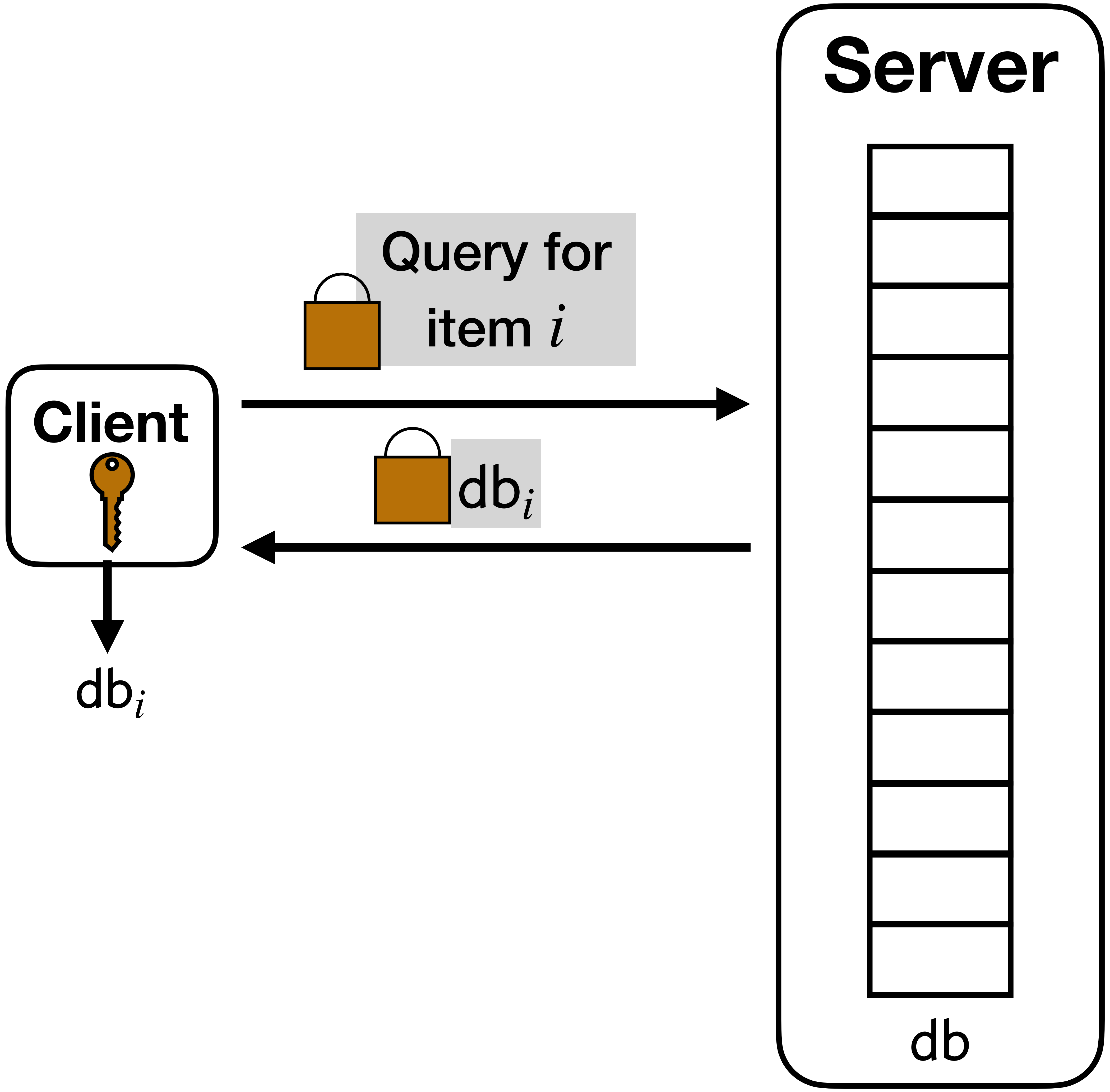


Private Information Retrieval (PIR) [CKSG95, KO97]



Private Information Retrieval (PIR) [CKSG95, KO97]

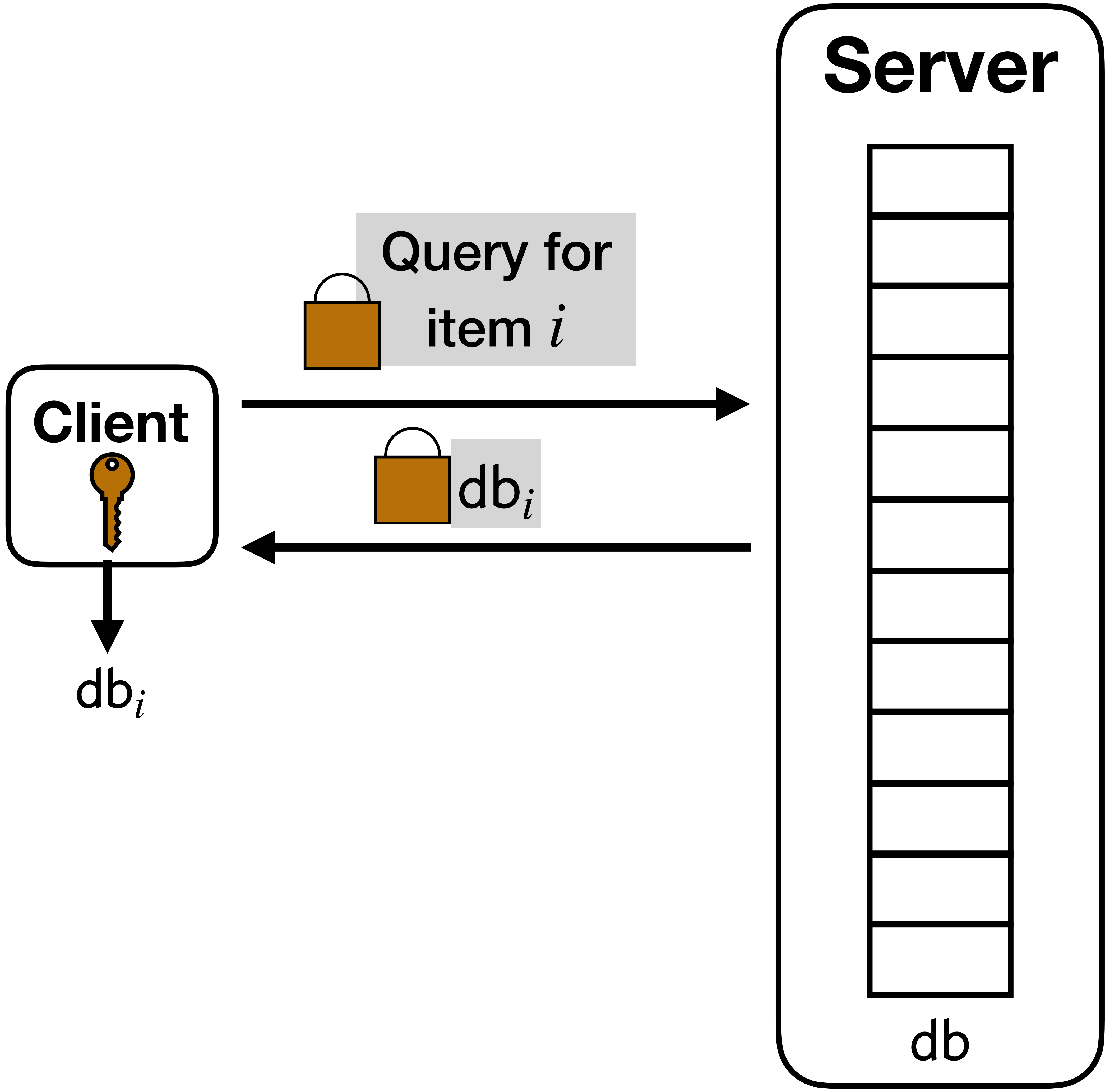
Properties:



Private Information Retrieval (PIR) [CKSG95, KO97]

Properties:

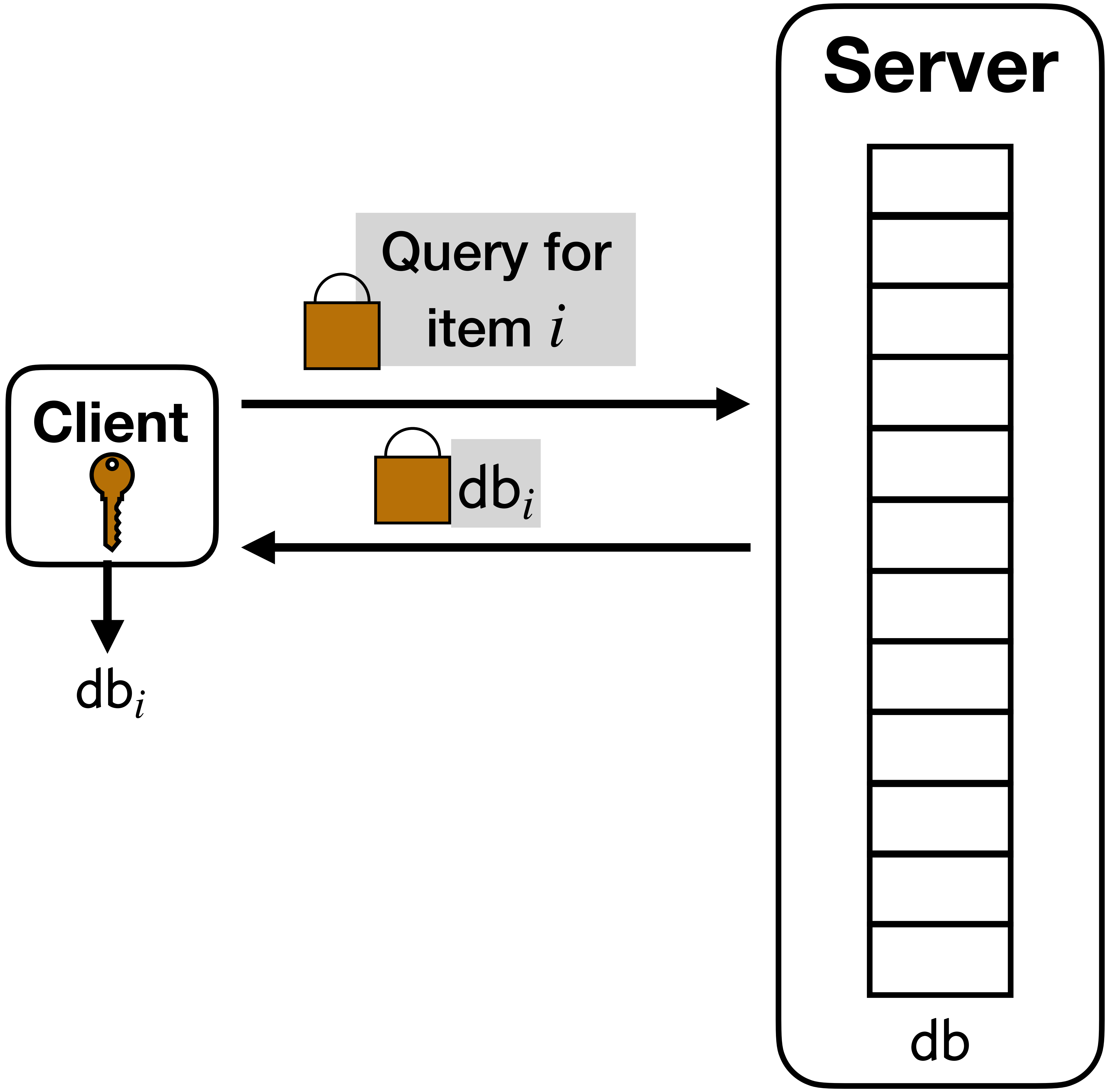
1. Correctness: client outputs db_i



Private Information Retrieval (PIR) [CKSG95, KO97]

Properties:

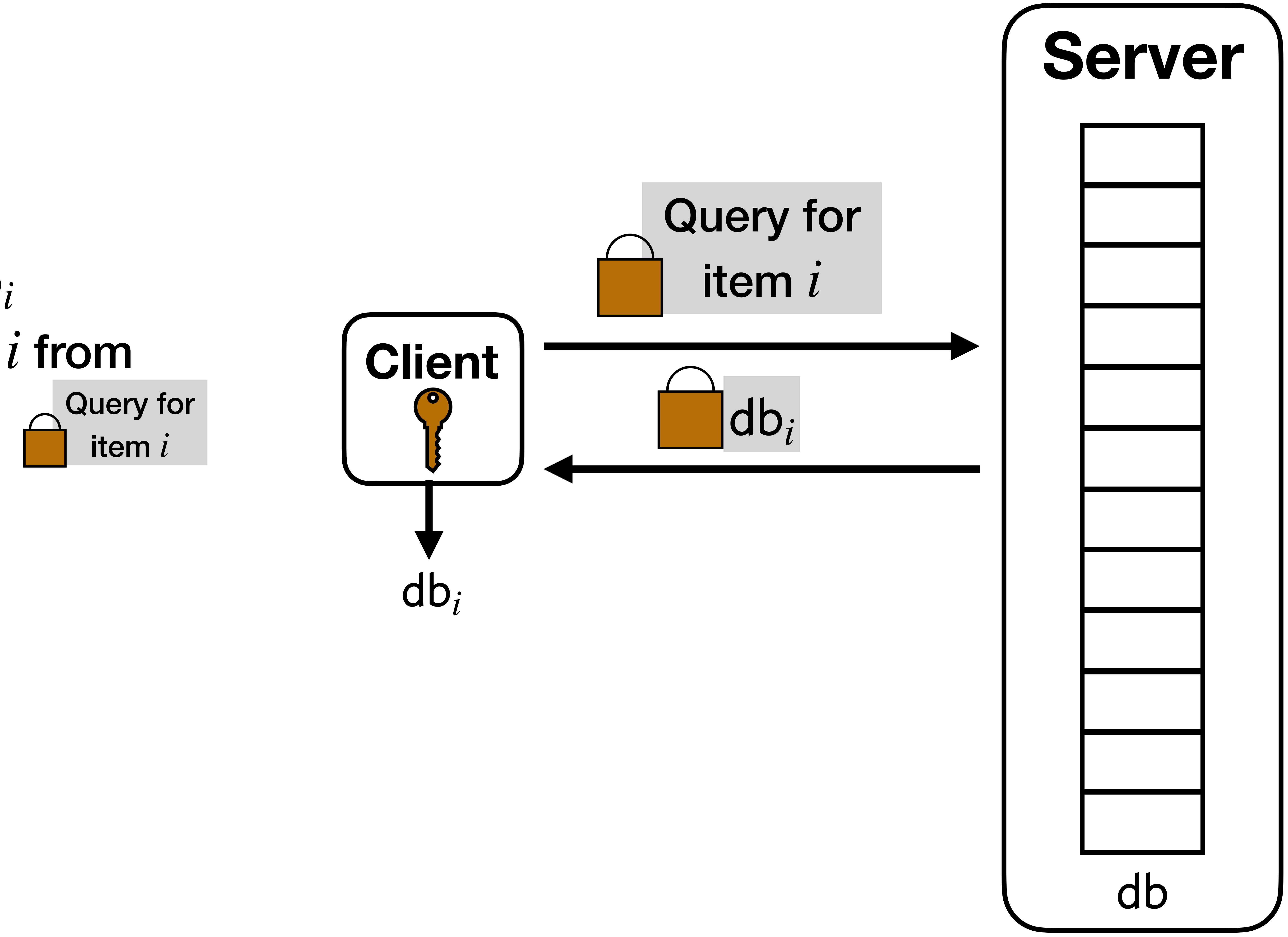
1. Correctness: client outputs db_i
2. Privacy: server does not learn i from



Private Information Retrieval (PIR) [CKSG95, KO97]

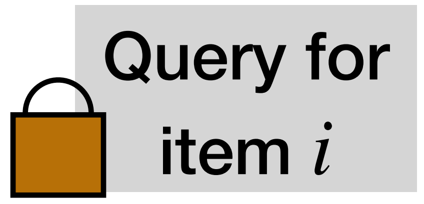
Properties:

1. Correctness: client outputs db_i
2. Privacy: server does not learn i from

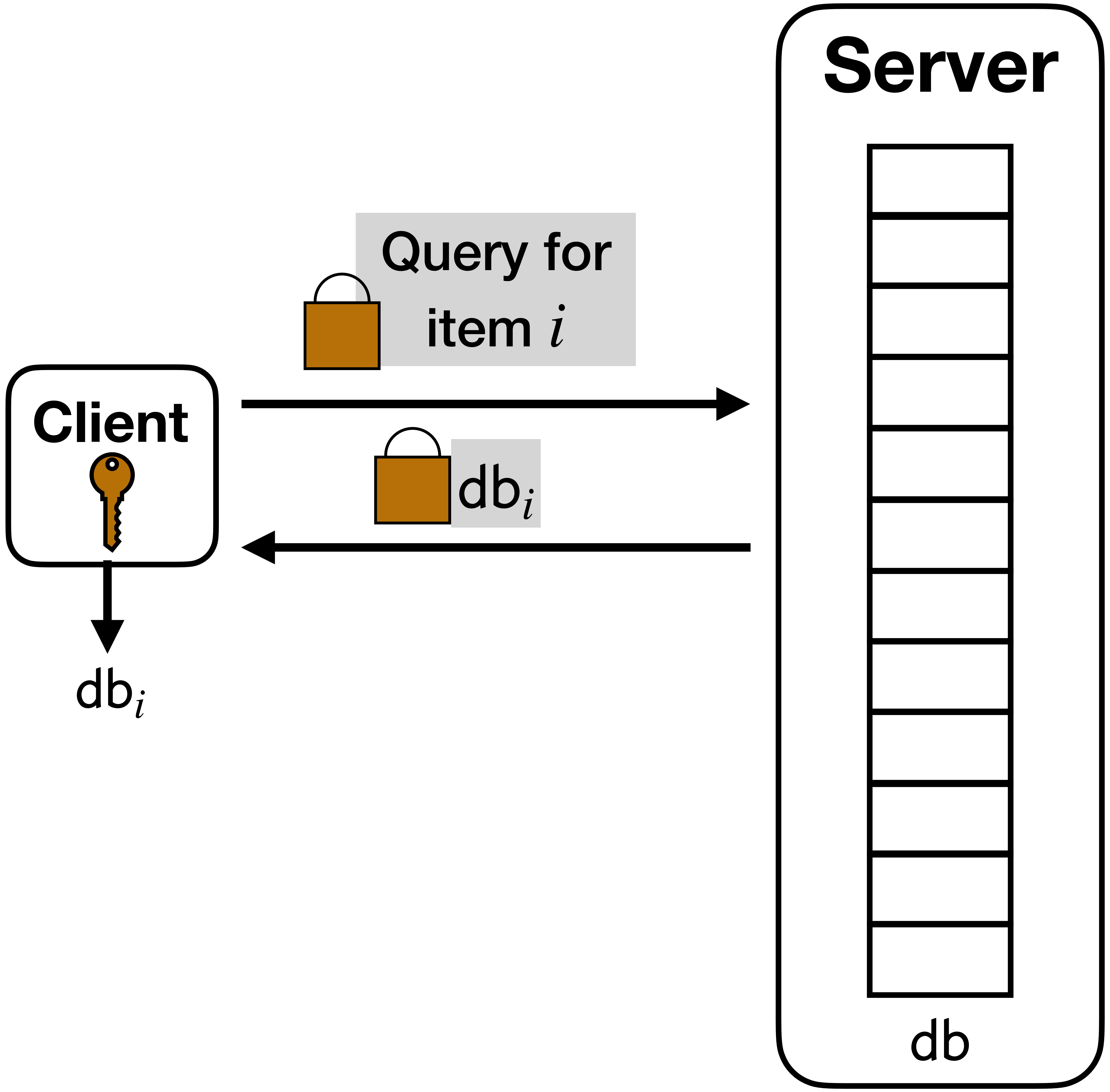


Private Information Retrieval (PIR) [CKSG95, KO97]

Properties:

1. Correctness: client outputs db_i
2. Privacy: server does not learn i from 

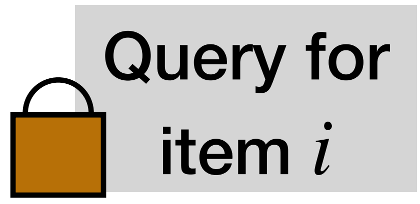
3. Efficiency: communication & computation are “small”



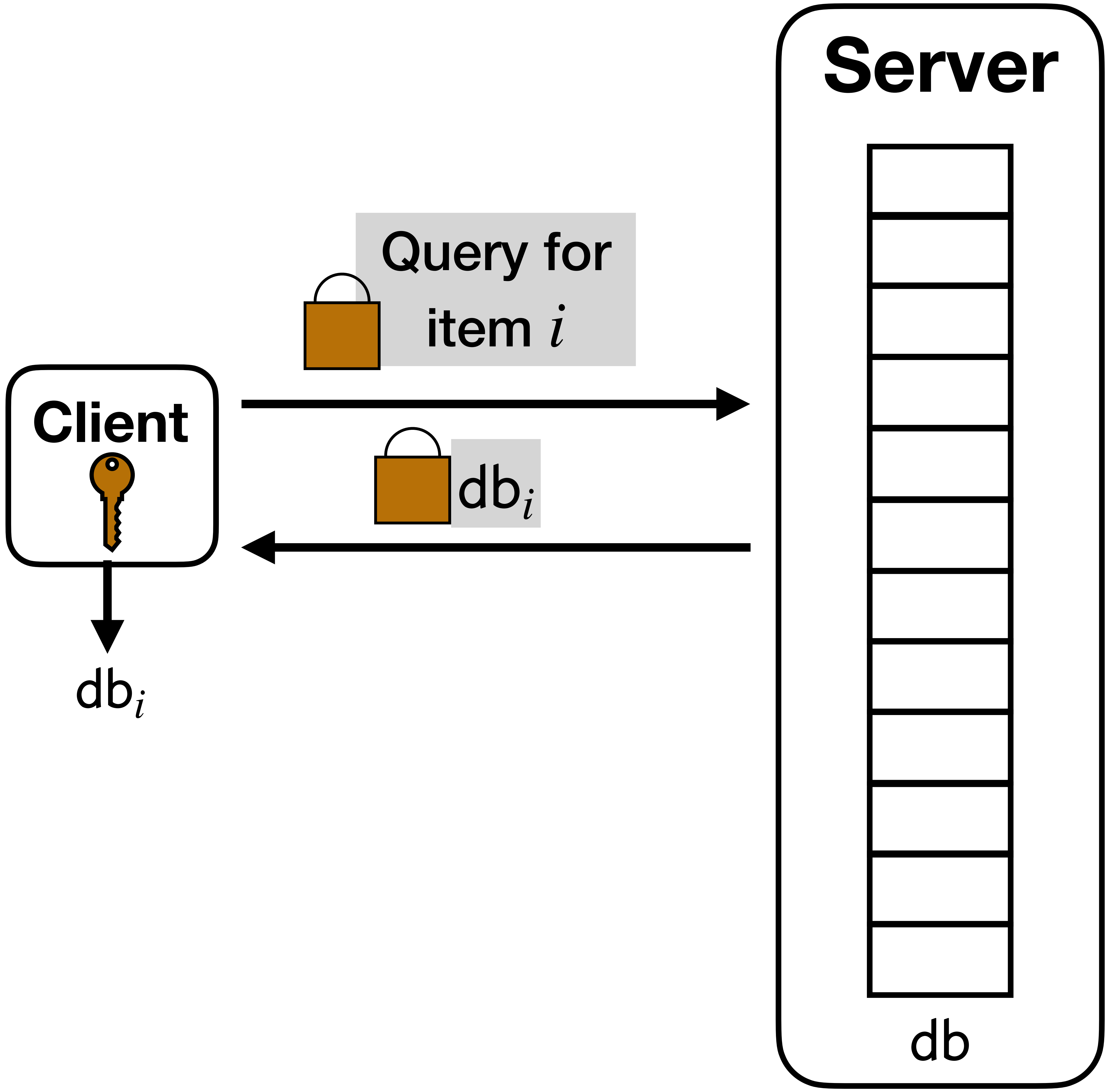
Private Information Retrieval (PIR) [CKSG95, KO97]

Properties:

1. Correctness: client outputs db_i
2. Privacy: server does not learn i from



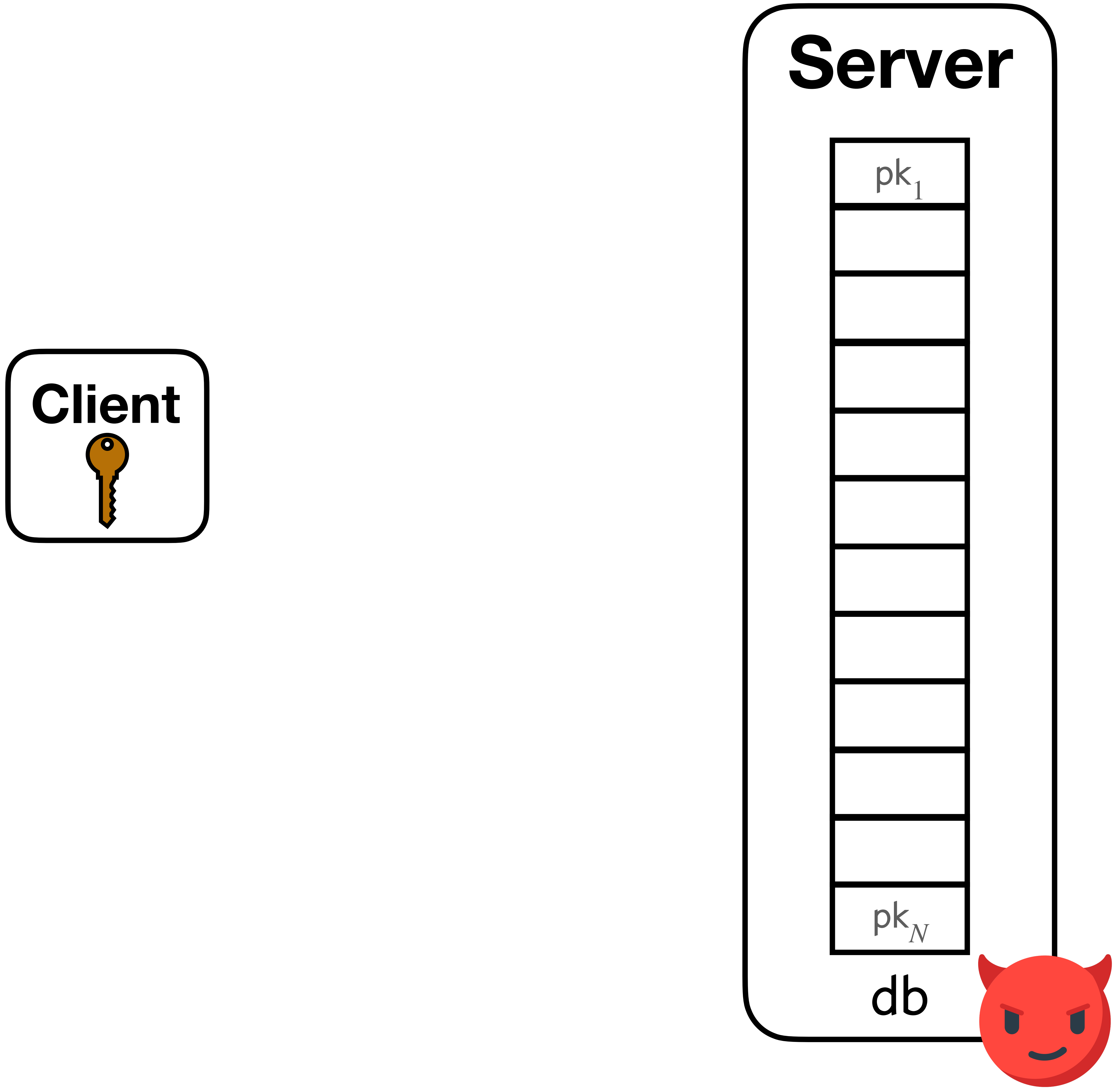
3. Efficiency: communication & computation are "small"



* focus on single-server

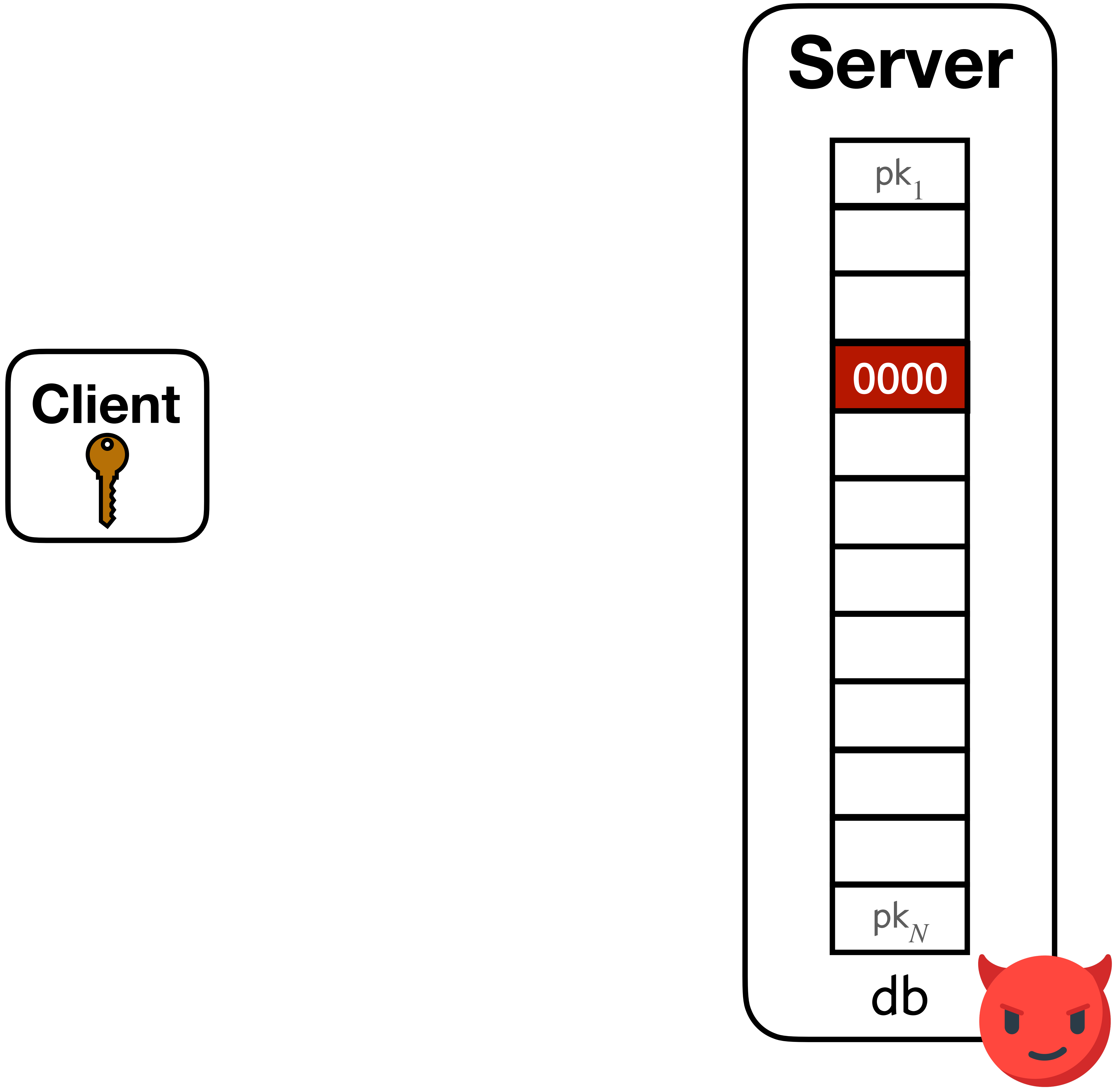
What if the server is malicious?

Selective Failure Attack



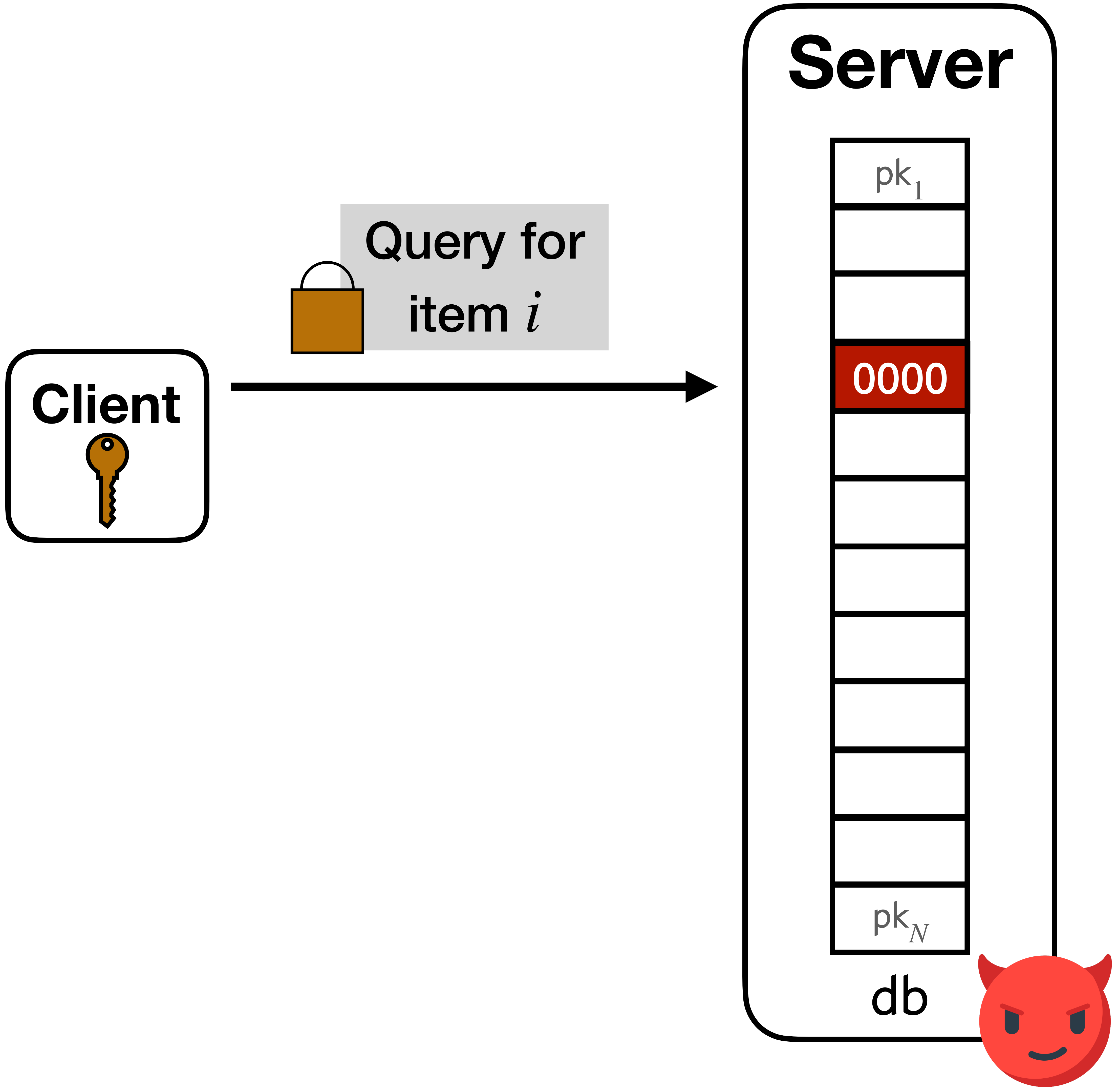
What if the server is malicious?

Selective Failure Attack



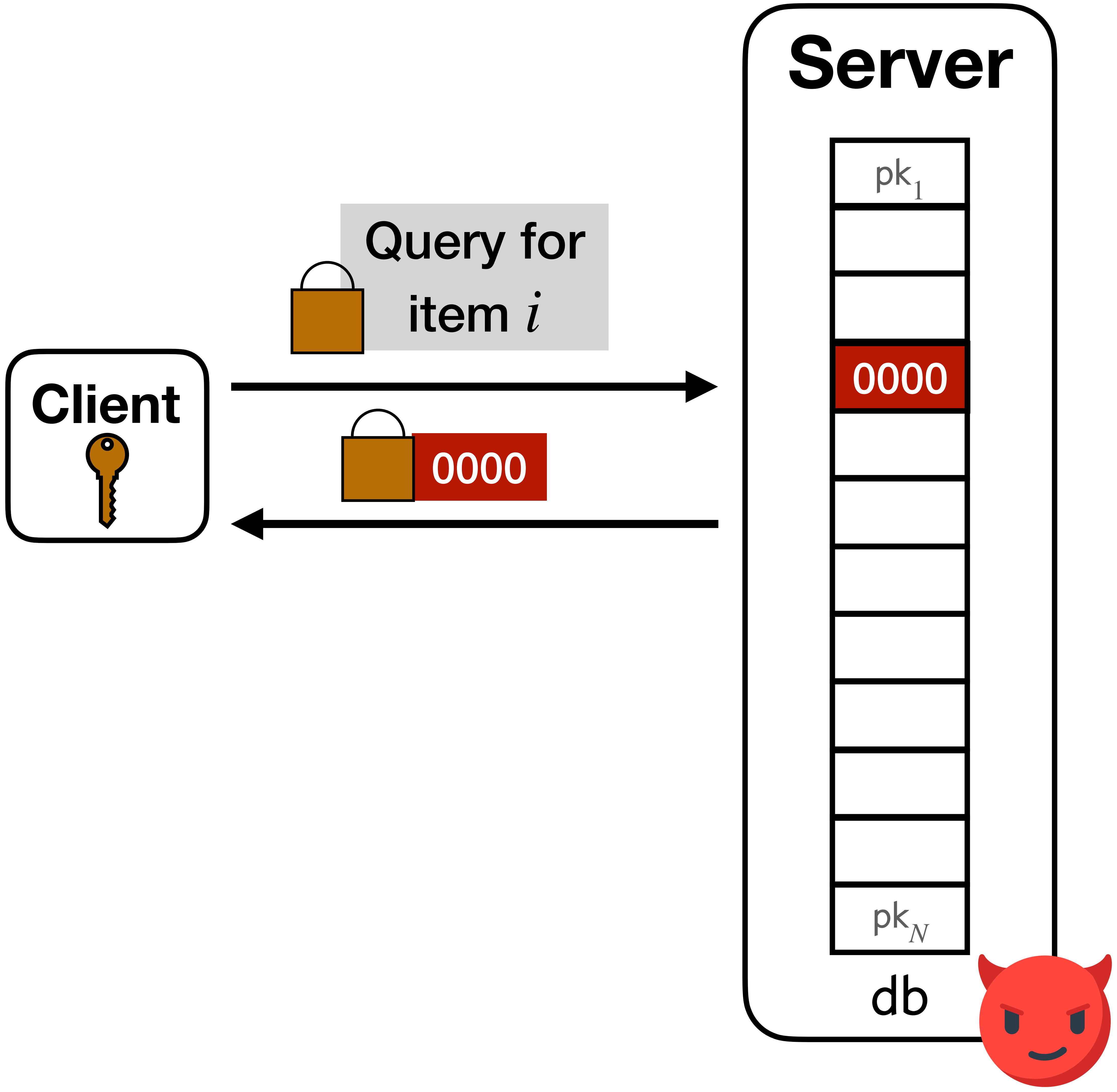
What if the server is malicious?

Selective Failure Attack



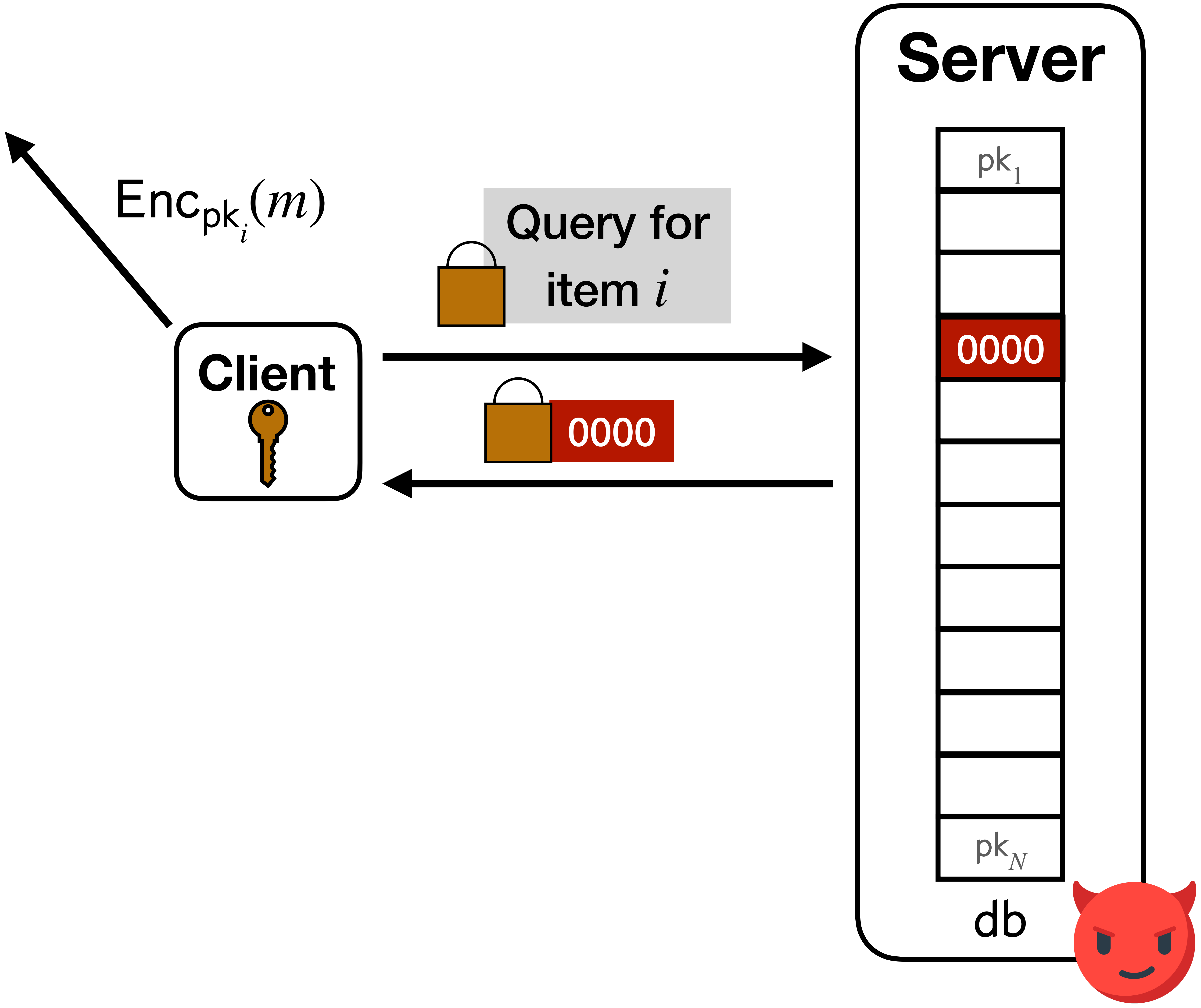
What if the server is malicious?

Selective Failure Attack



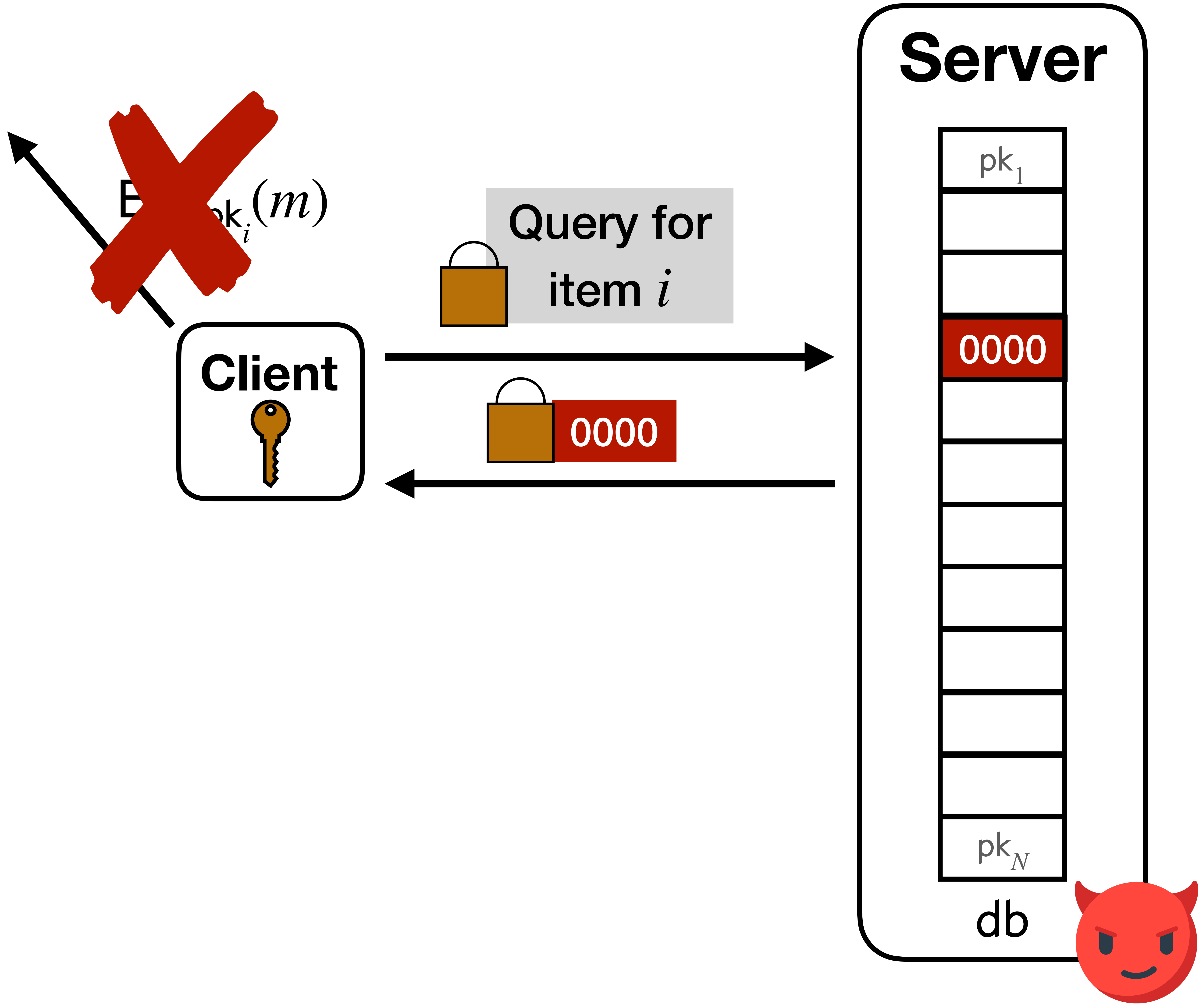
What if the server is malicious?

Selective Failure Attack



What if the server is malicious?

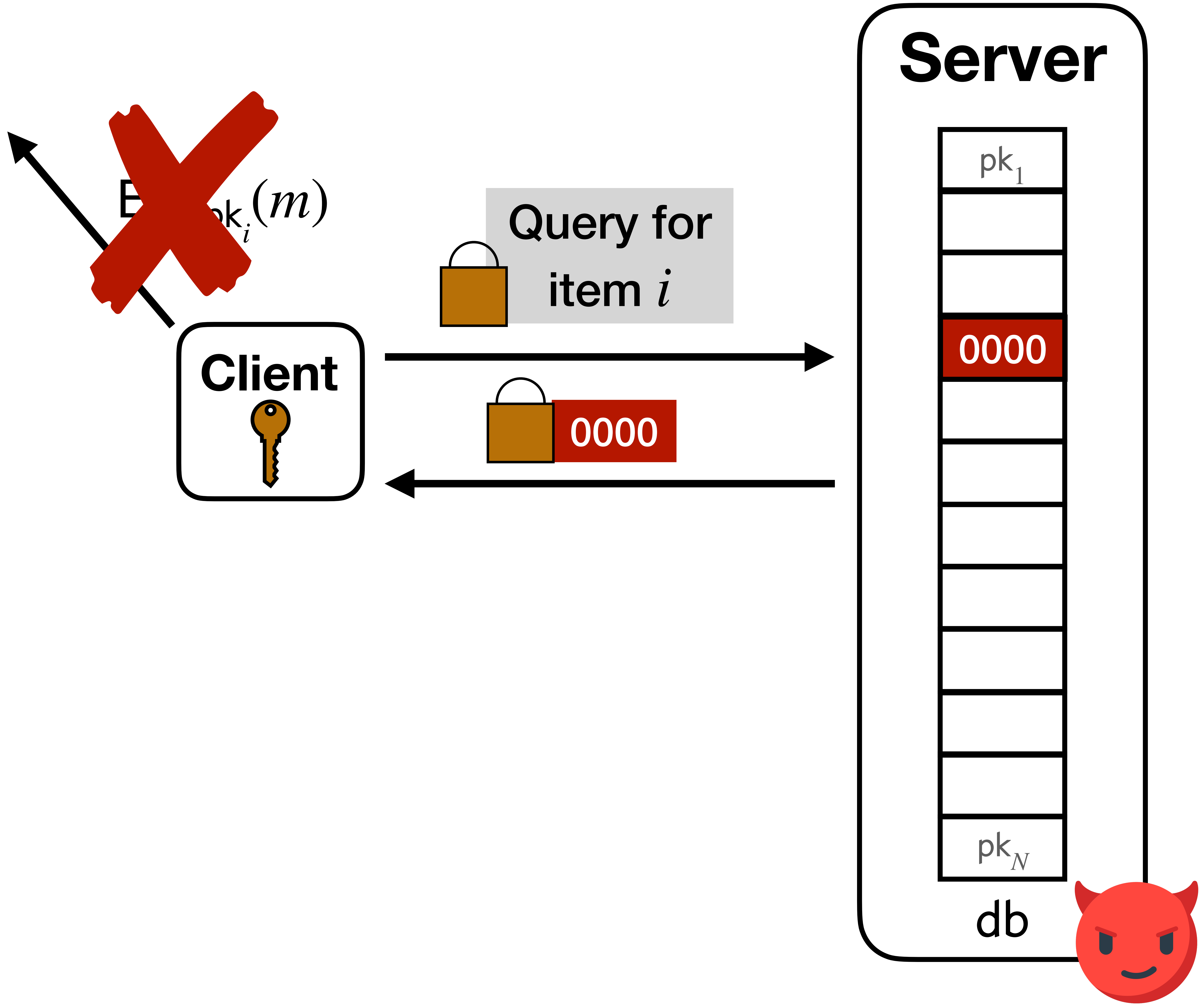
Selective Failure Attack



What if the server is malicious?

Selective Failure Attack

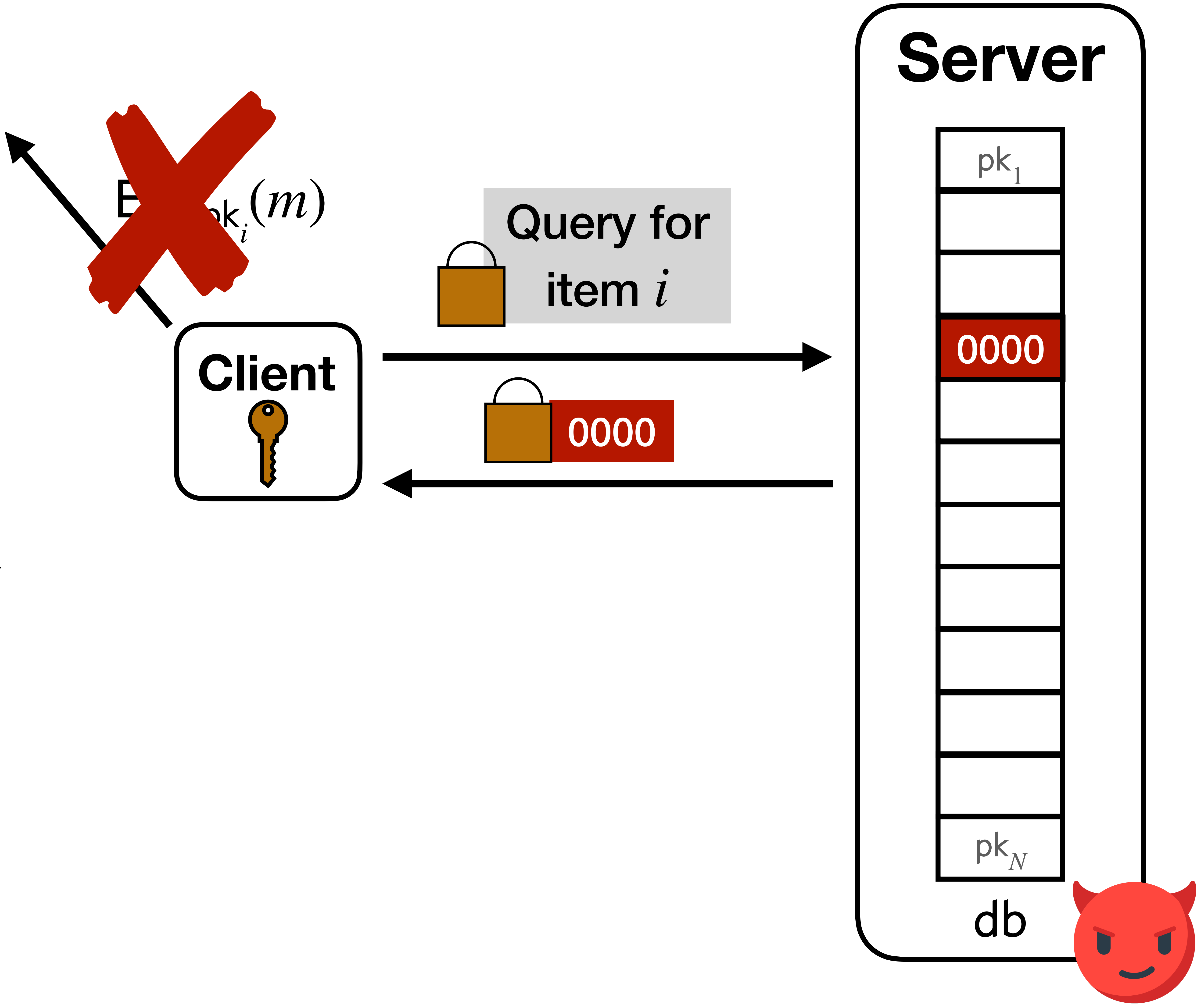
- If the client queries i: it will get garbage and won't be able to preform the "next action."



What if the server is malicious?

Selective Failure Attack

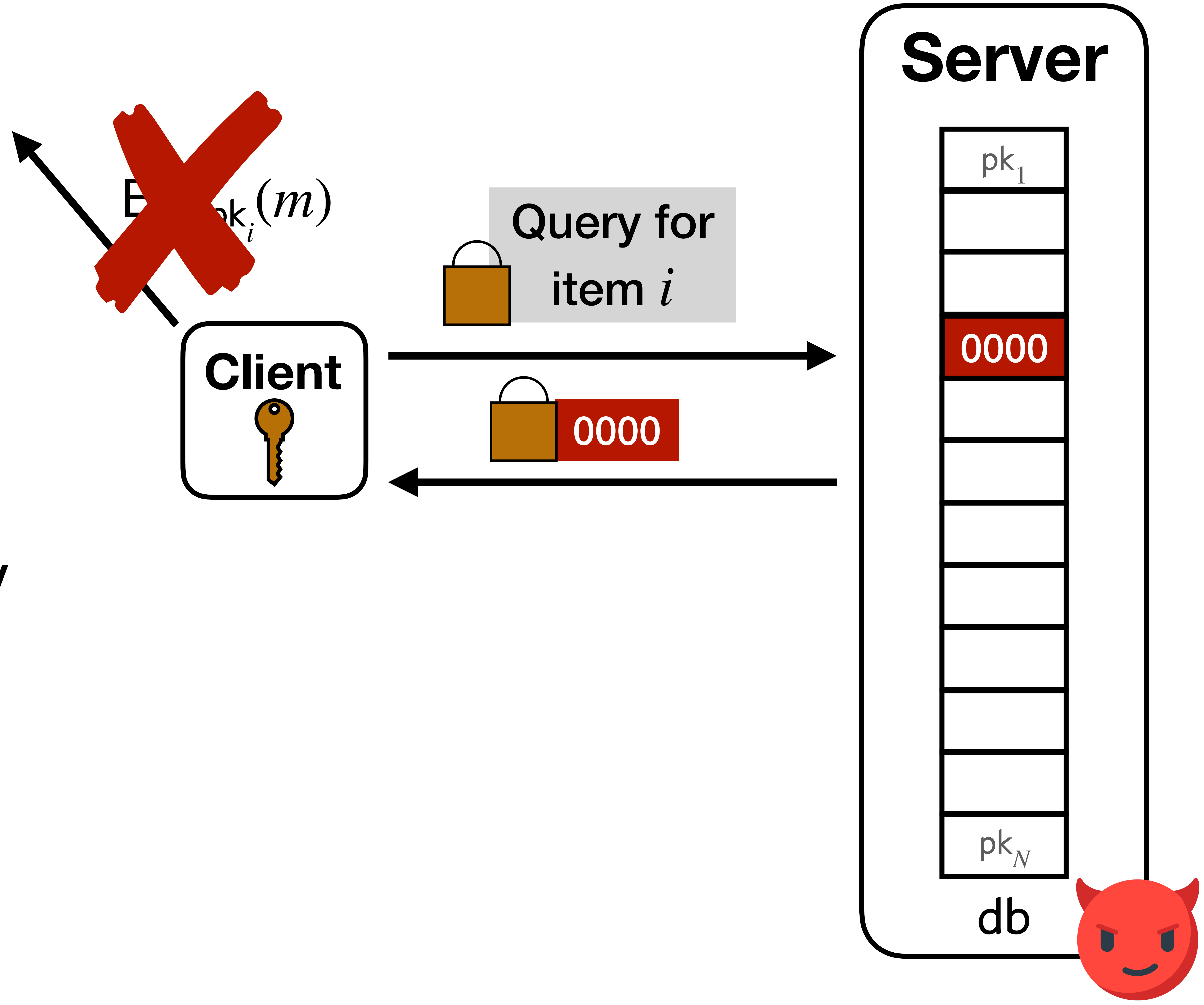
- If the client queries i: it will get garbage and won't be able to preform the "next action."
- If client queries for $j \neq i$: then the client will preform the "next action" correctly, not knowing there are any corruptions
  problem: server can observe this discrepancy to learn i !



What if the server is malicious?

Selective Failure Attack

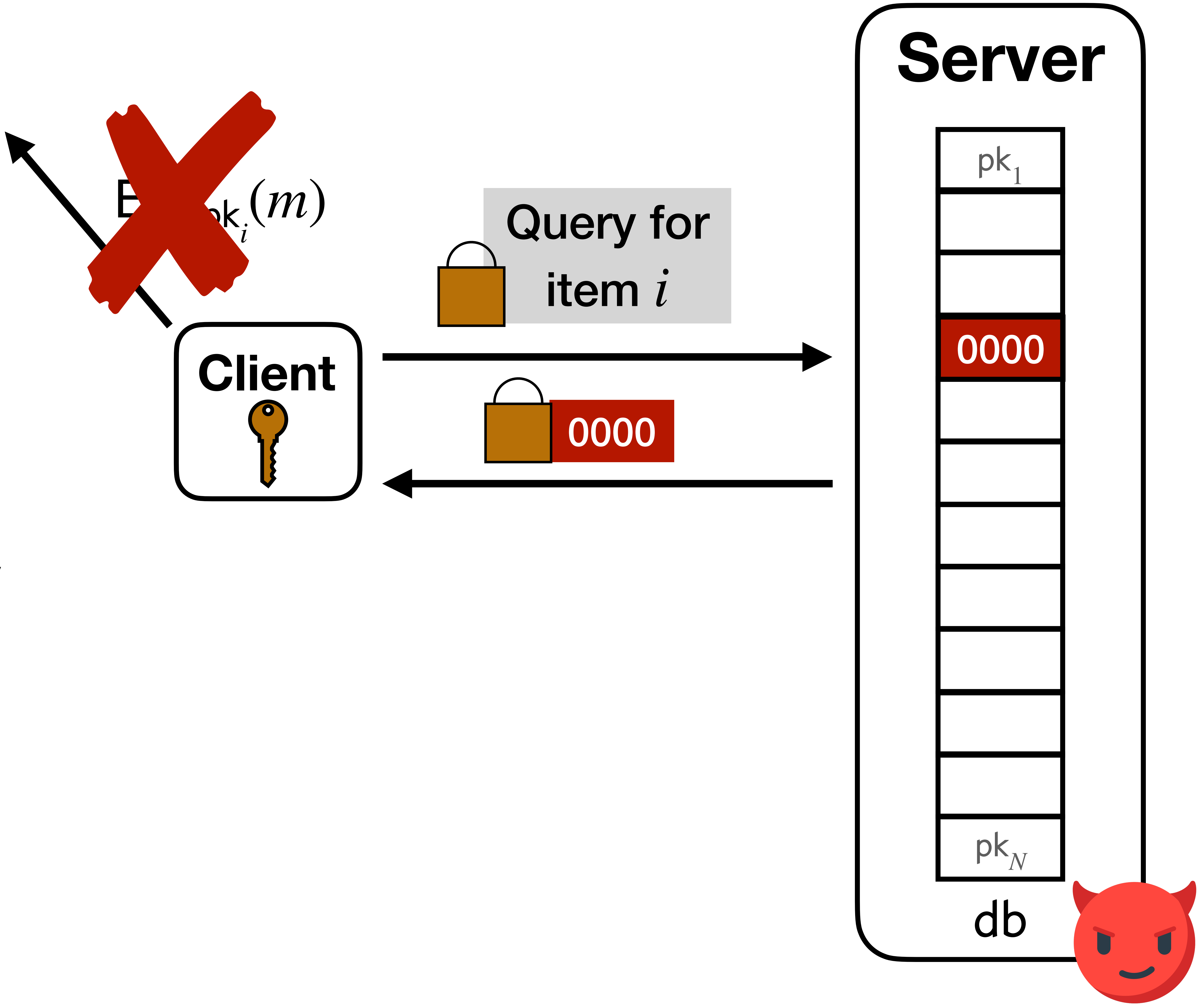
- If the client queries i: it will get garbage and won't be able to preform the "next action."
- If client queries for $j \neq i$: then the client will preform the "next action" correctly, not knowing there are any corruptions
  problem: server can observe this discrepancy to learn i !



What if the server is malicious? [KO97]

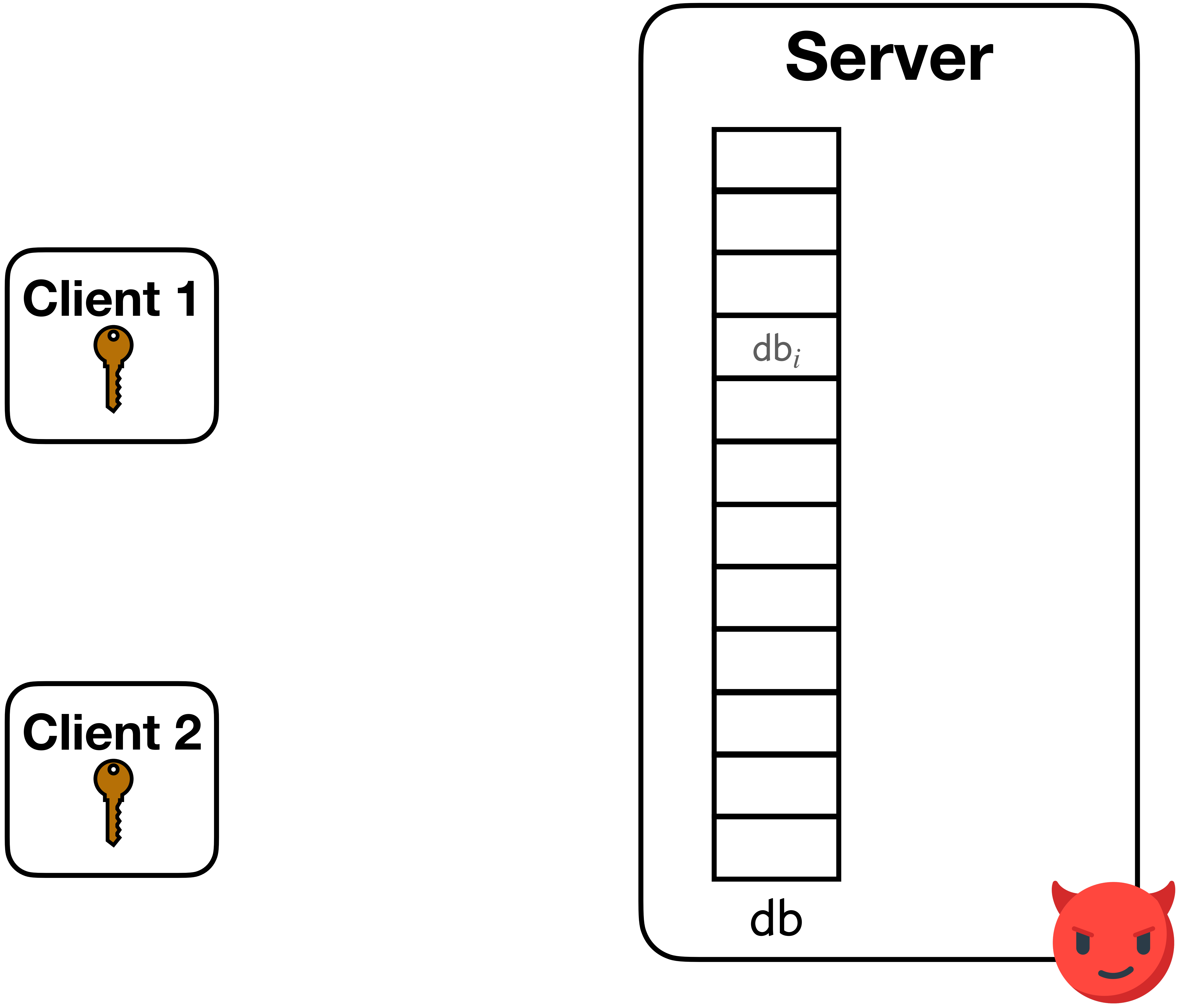
Selective Failure Attack

- If the client queries i: it will get garbage and won't be able to preform the "next action."
- If client queries for $j \neq i$: then the client will preform the "next action" correctly, not knowing there are any corruptions
  <u>*problem:*</u> server can observe this discrepancy to learn i !



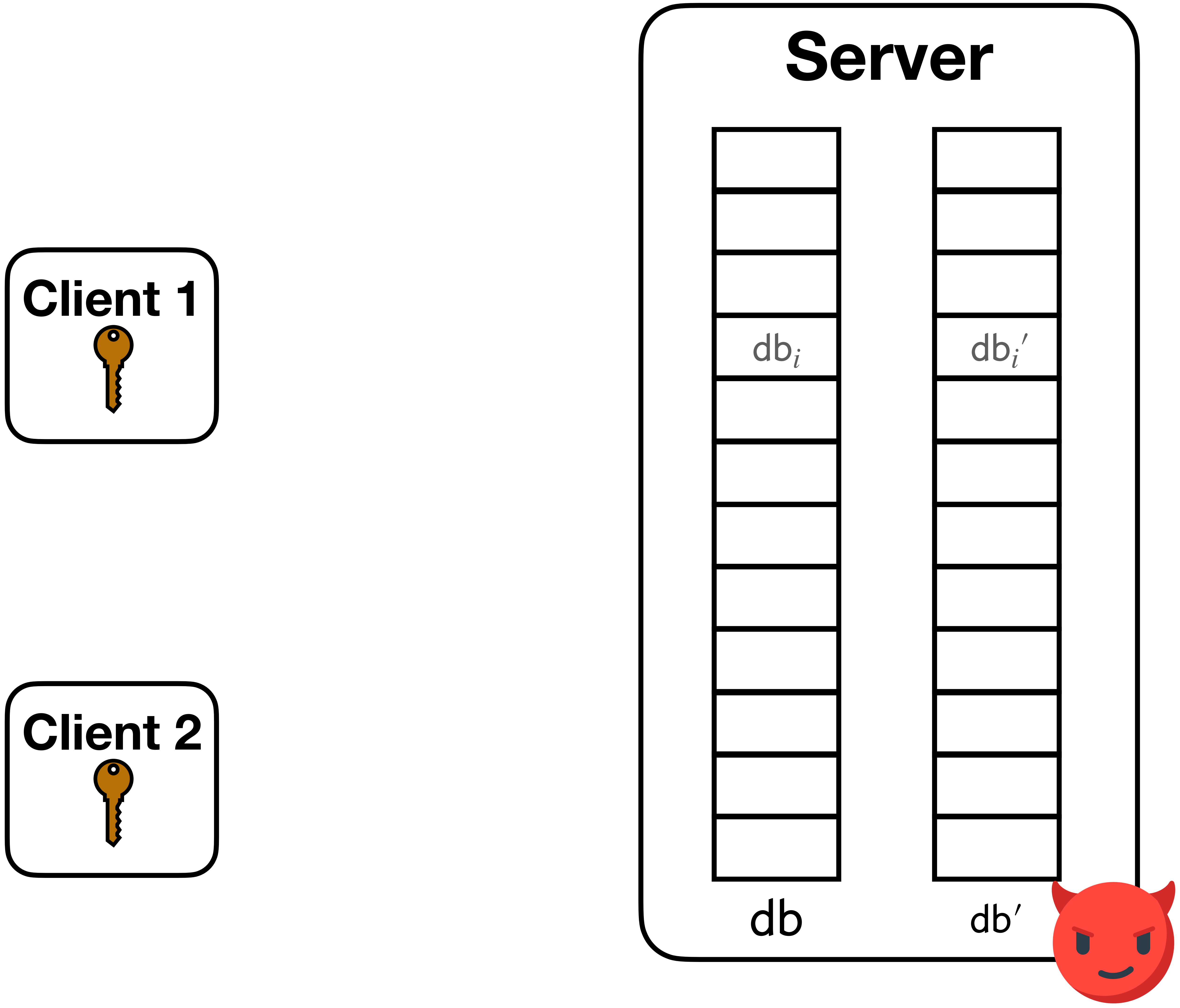
What if the server is malicious?

Incoherent views



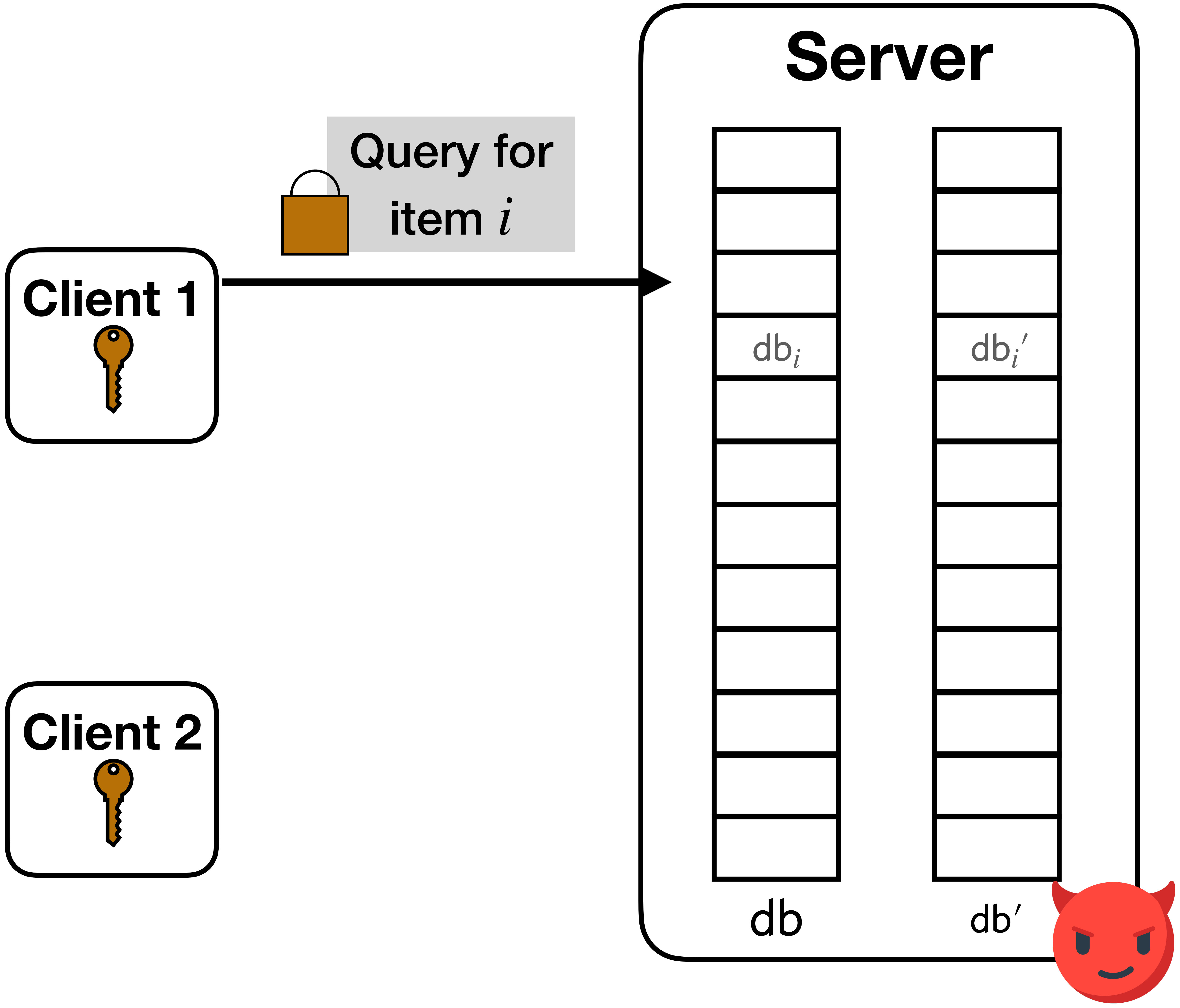
What if the server is malicious?

Incoherent views



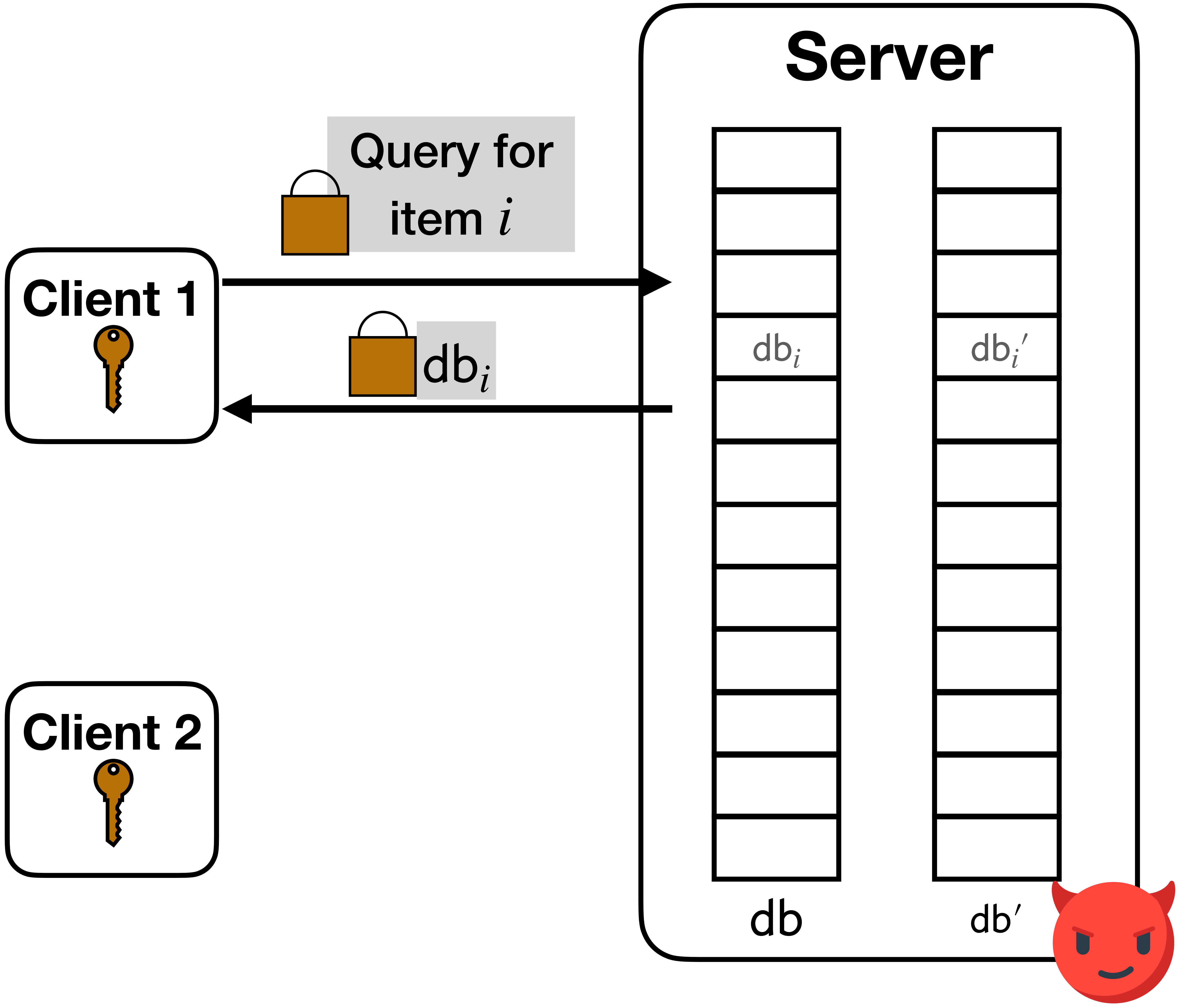
What if the server is malicious?

Incoherent views



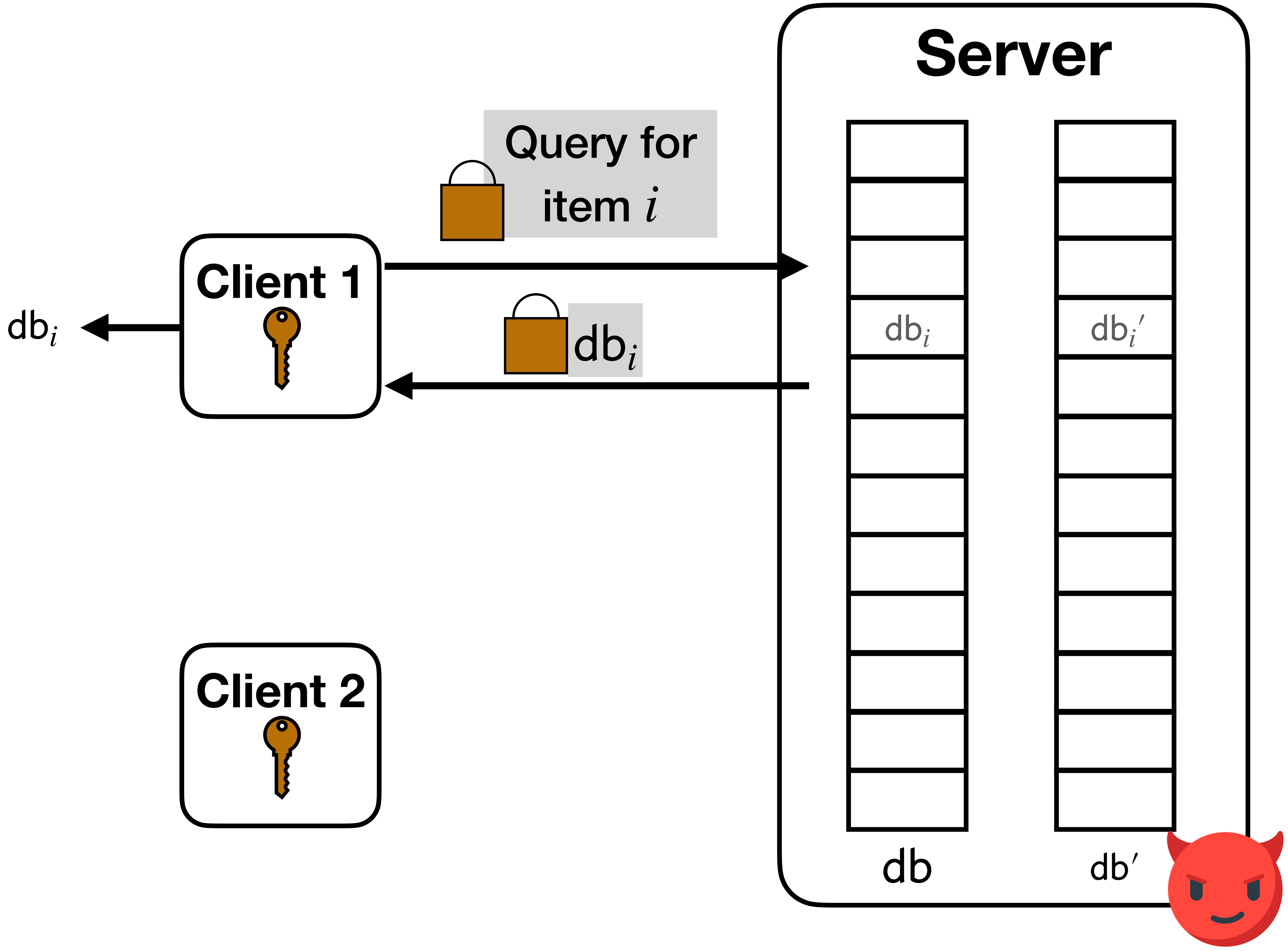
What if the server is malicious?

Incoherent views



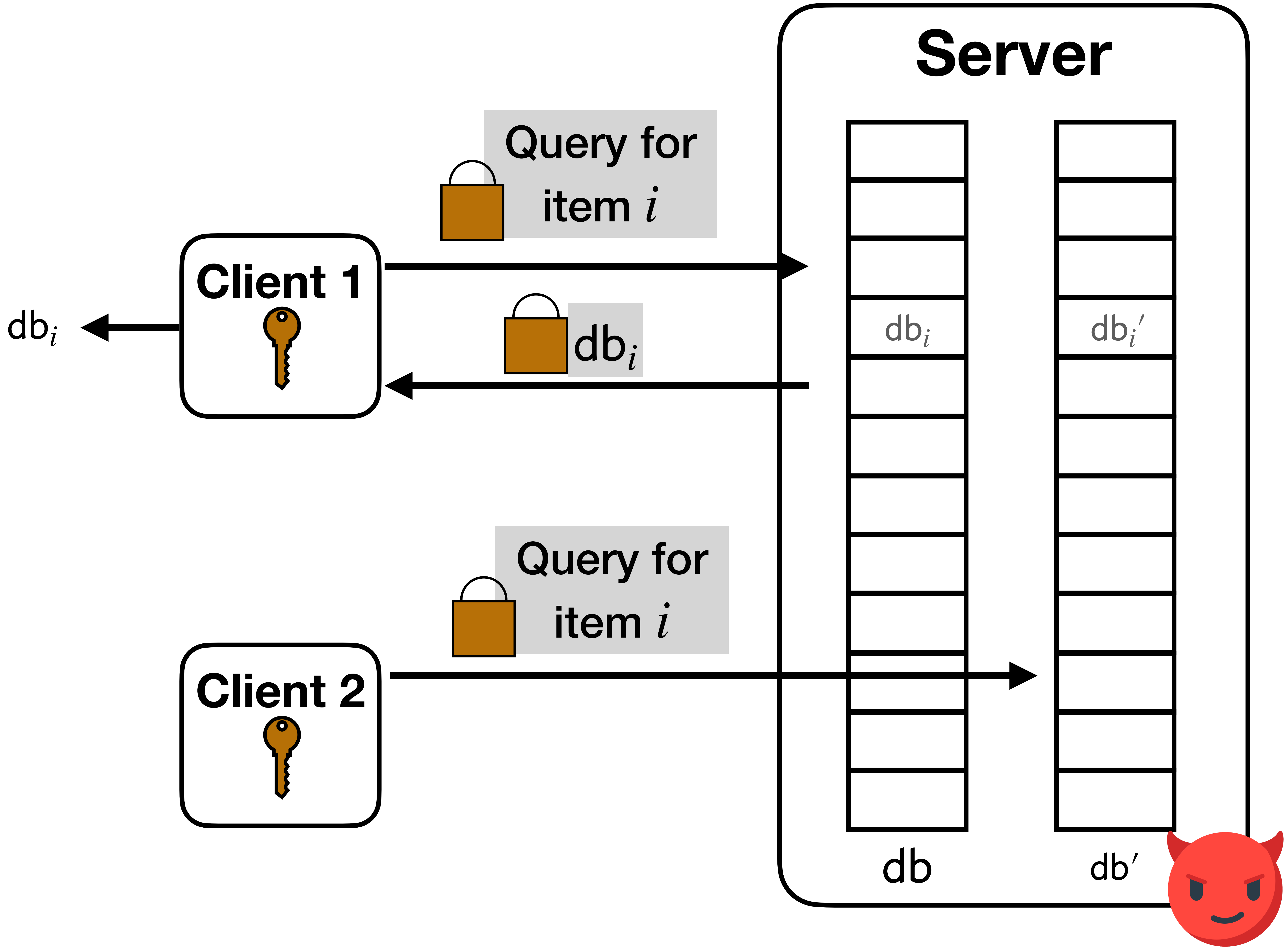
What if the server is malicious?

Incoherent views



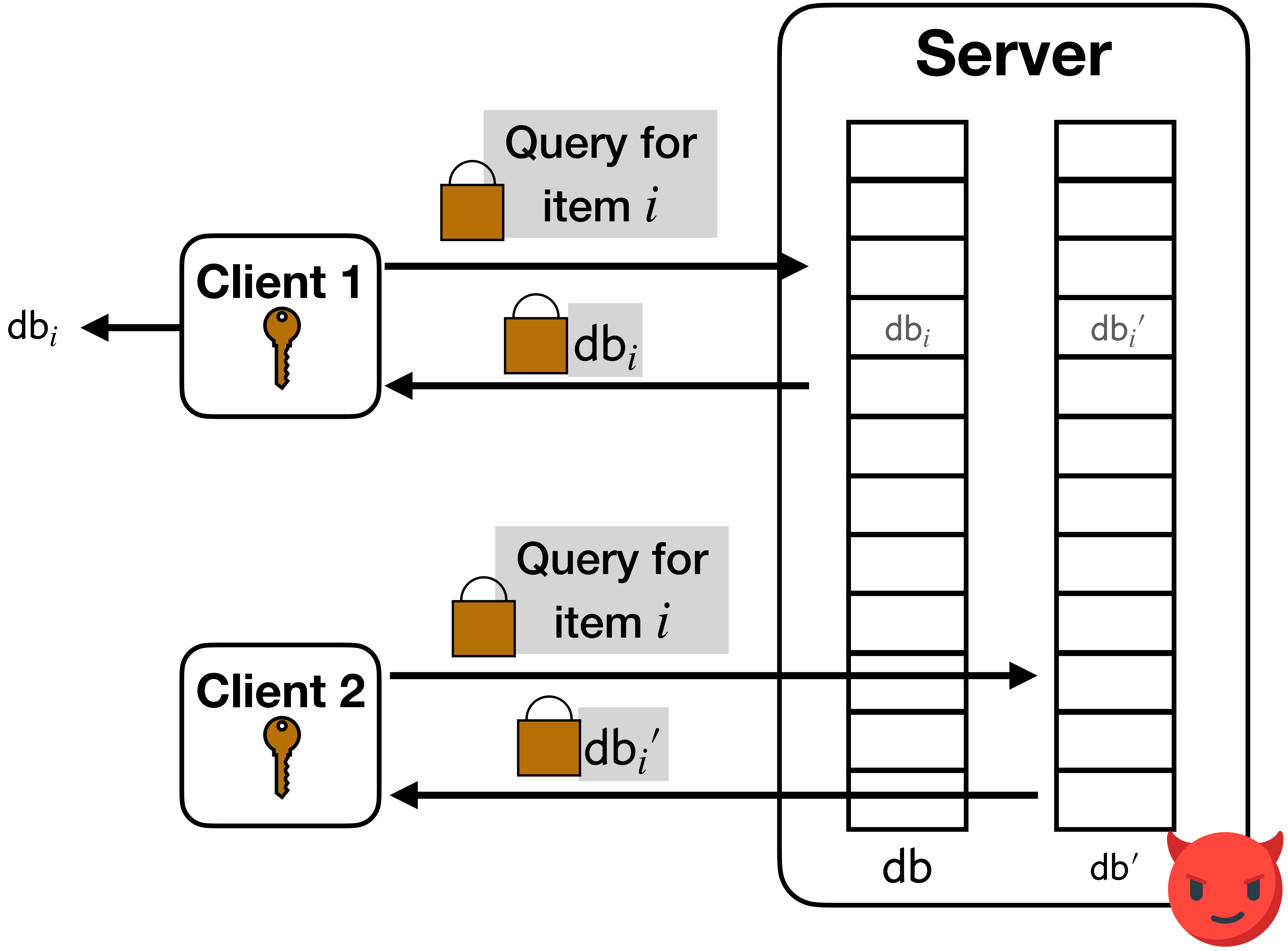
What if the server is malicious?

Incoherent views



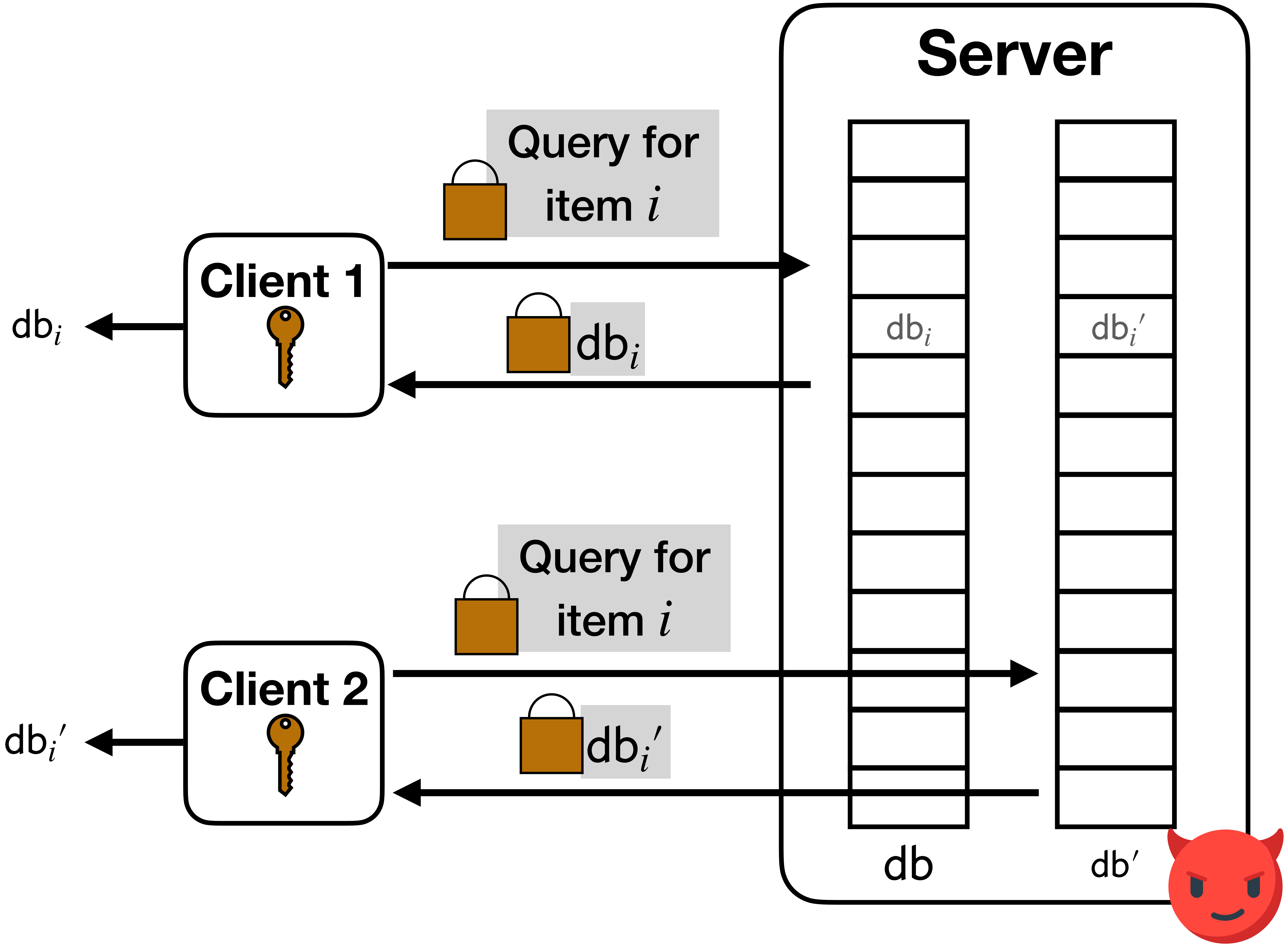
What if the server is malicious?

Incoherent views



What if the server is malicious?

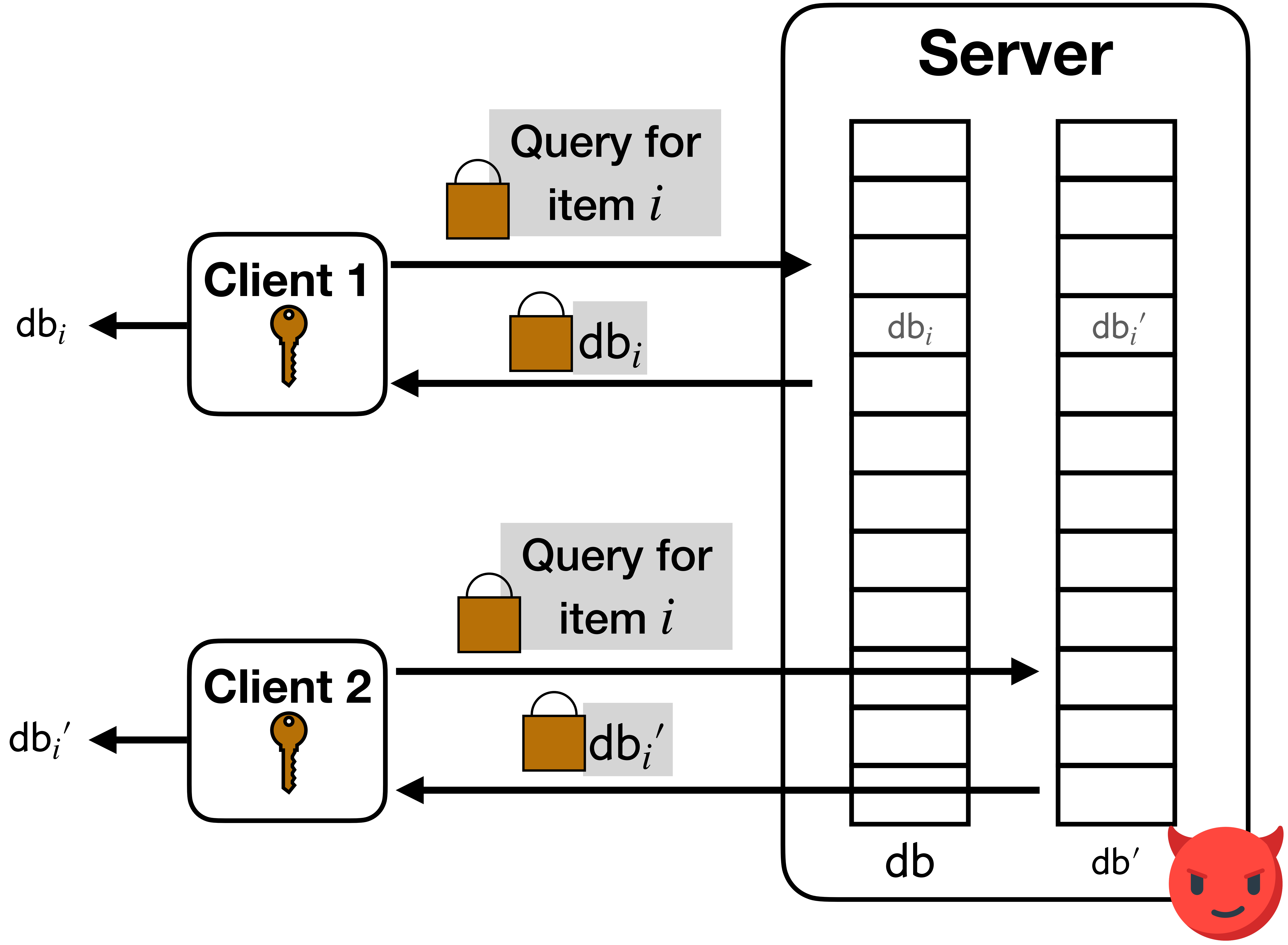
Incoherent views



What if the server is malicious?

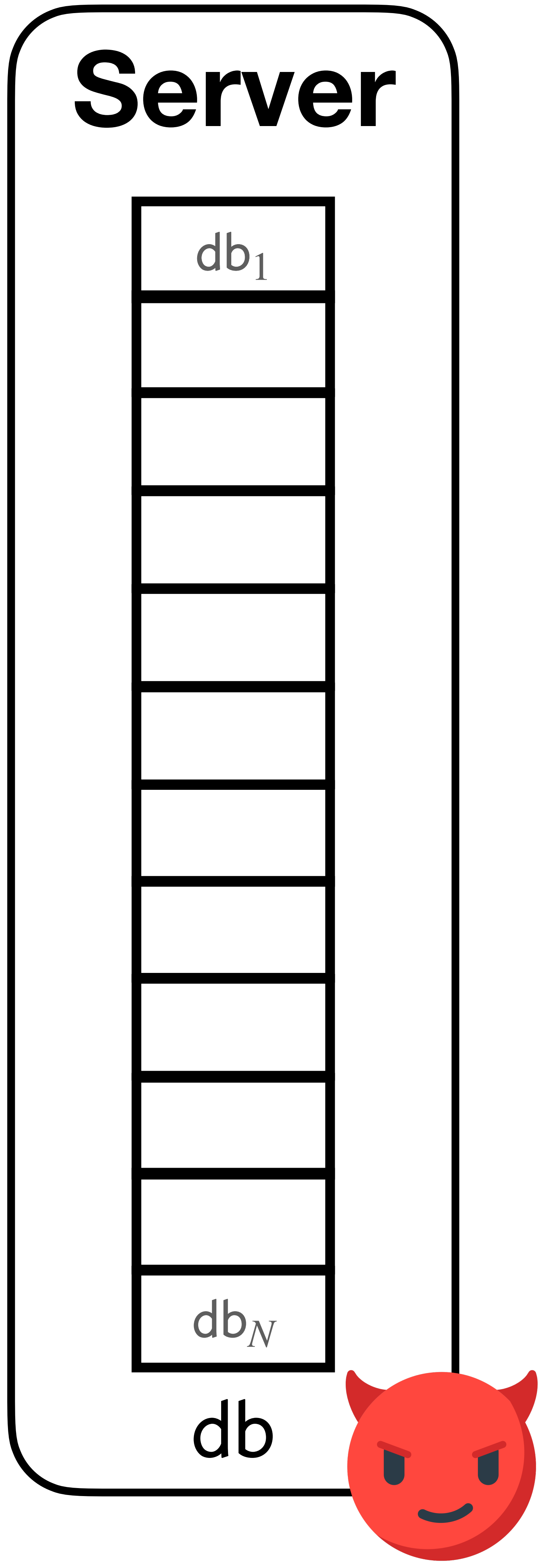
Incoherent views

Problem: clients do not agree on database.



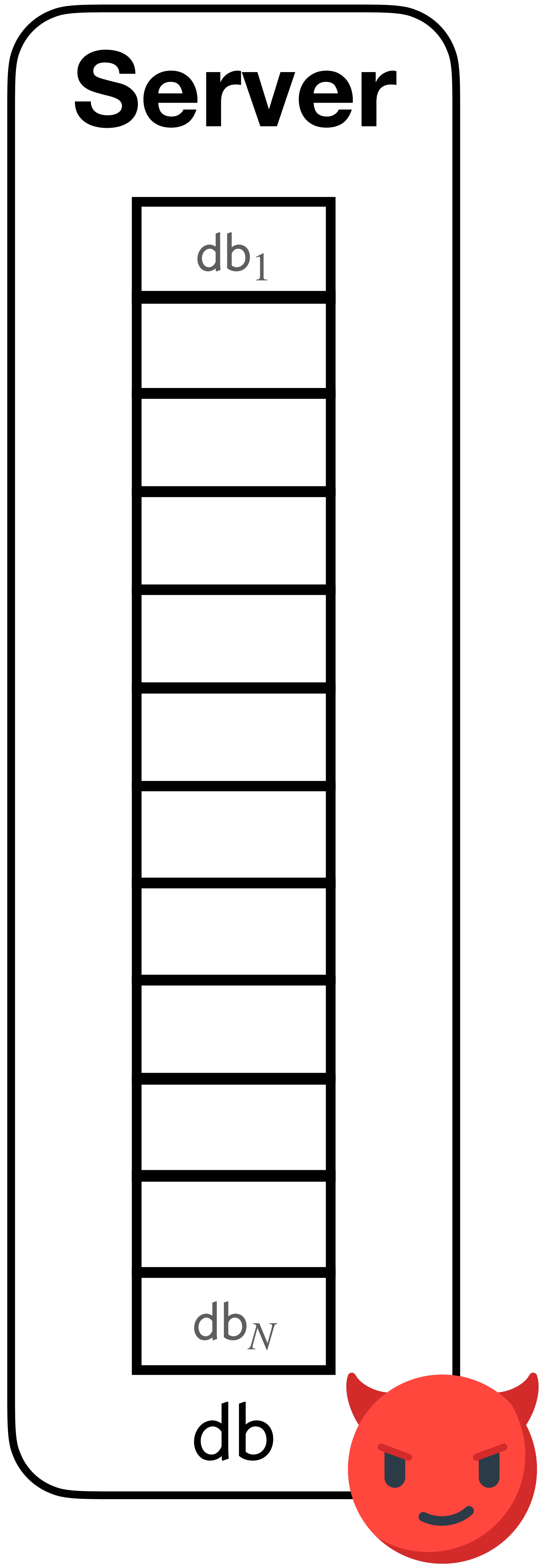
Prior work

Maliciously-secure PIR (mPIR) [CNCWF23; WZLY23; DT23; CL24]

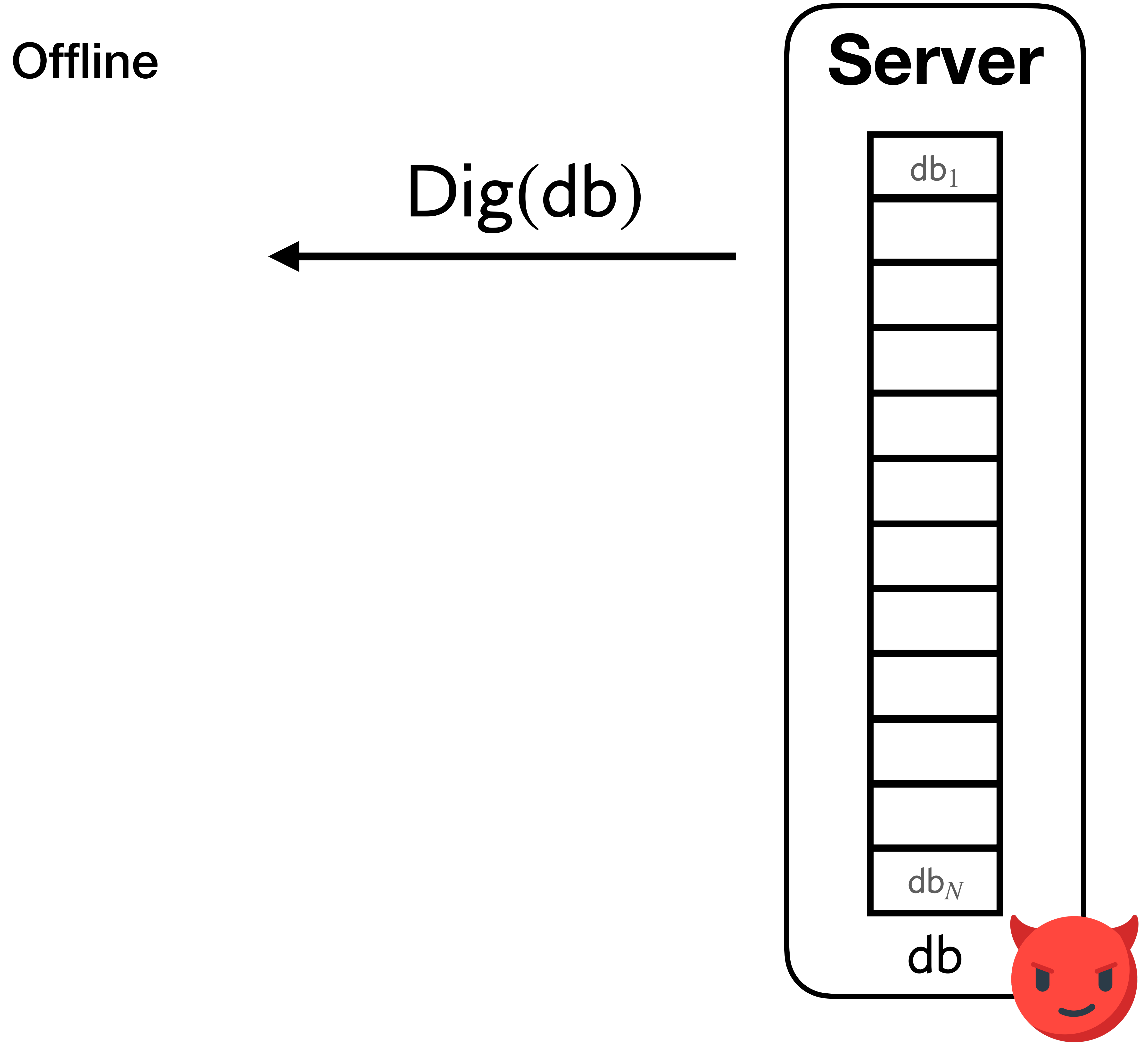


Maliciously-secure PIR (mPIR) [CNCWF23; WZLY23; DT23; CL24]

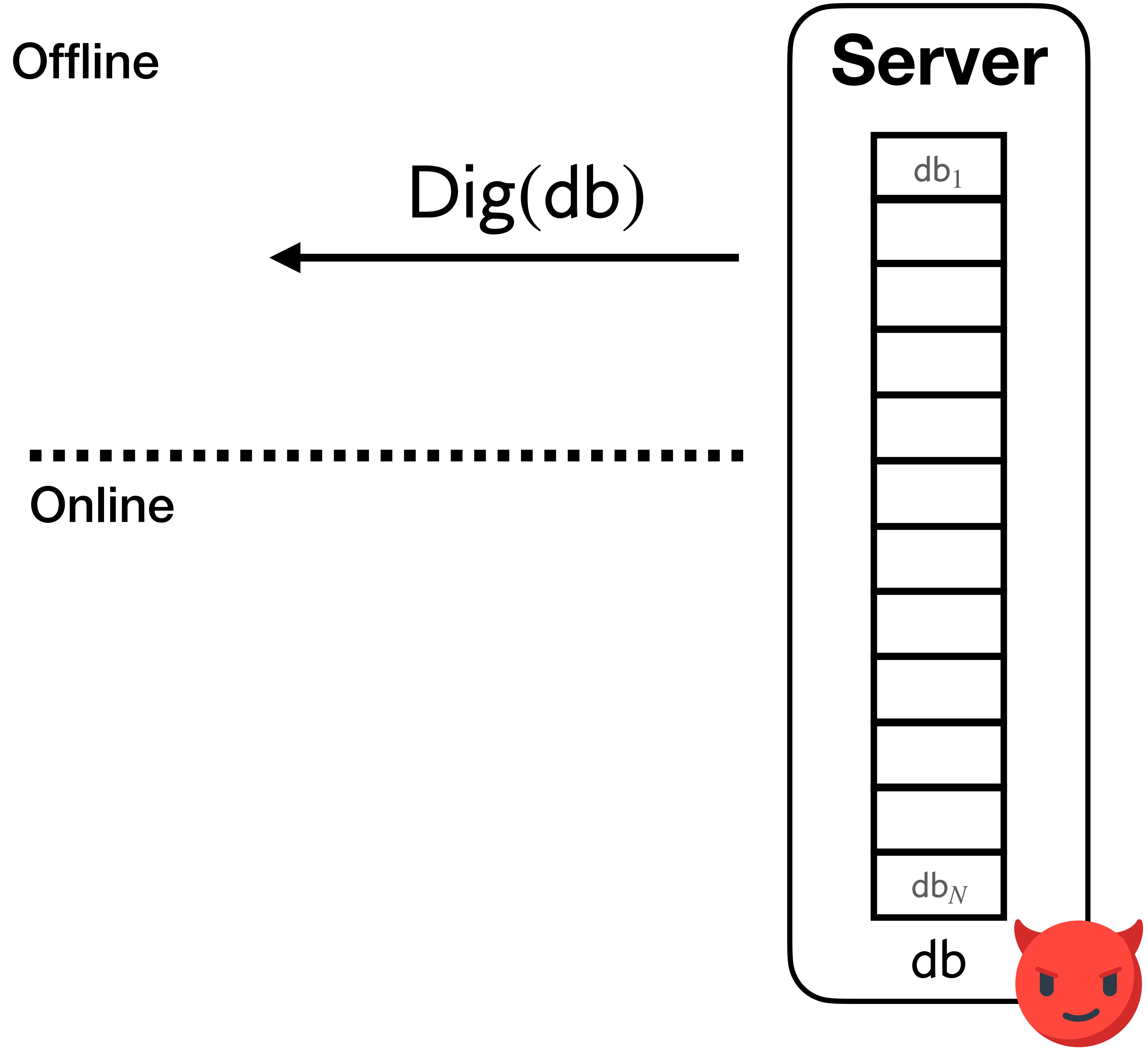
Offline



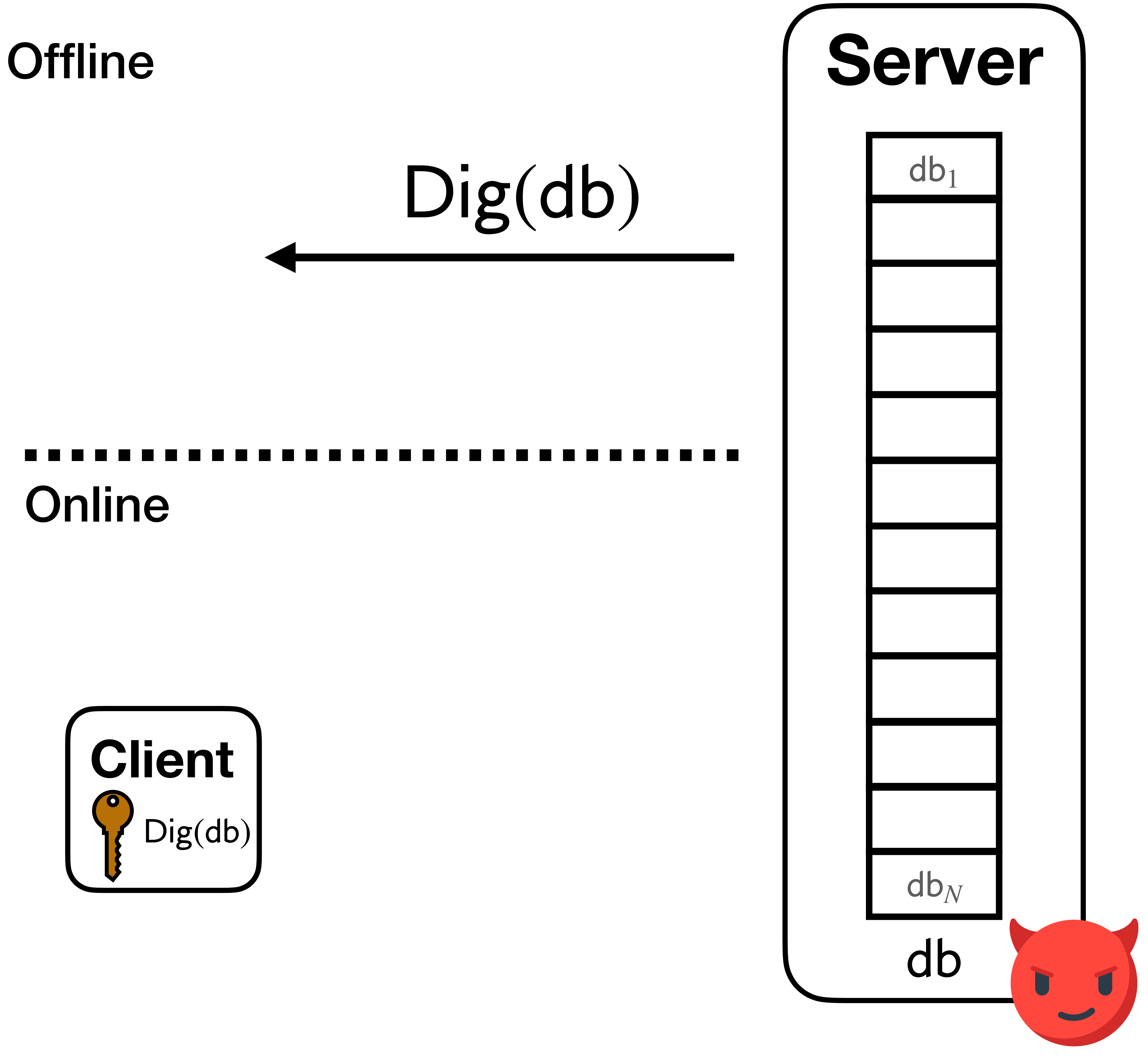
Maliciously-secure PIR (mPIR) [CNCWF23; WZLY23; DT23; CL24]



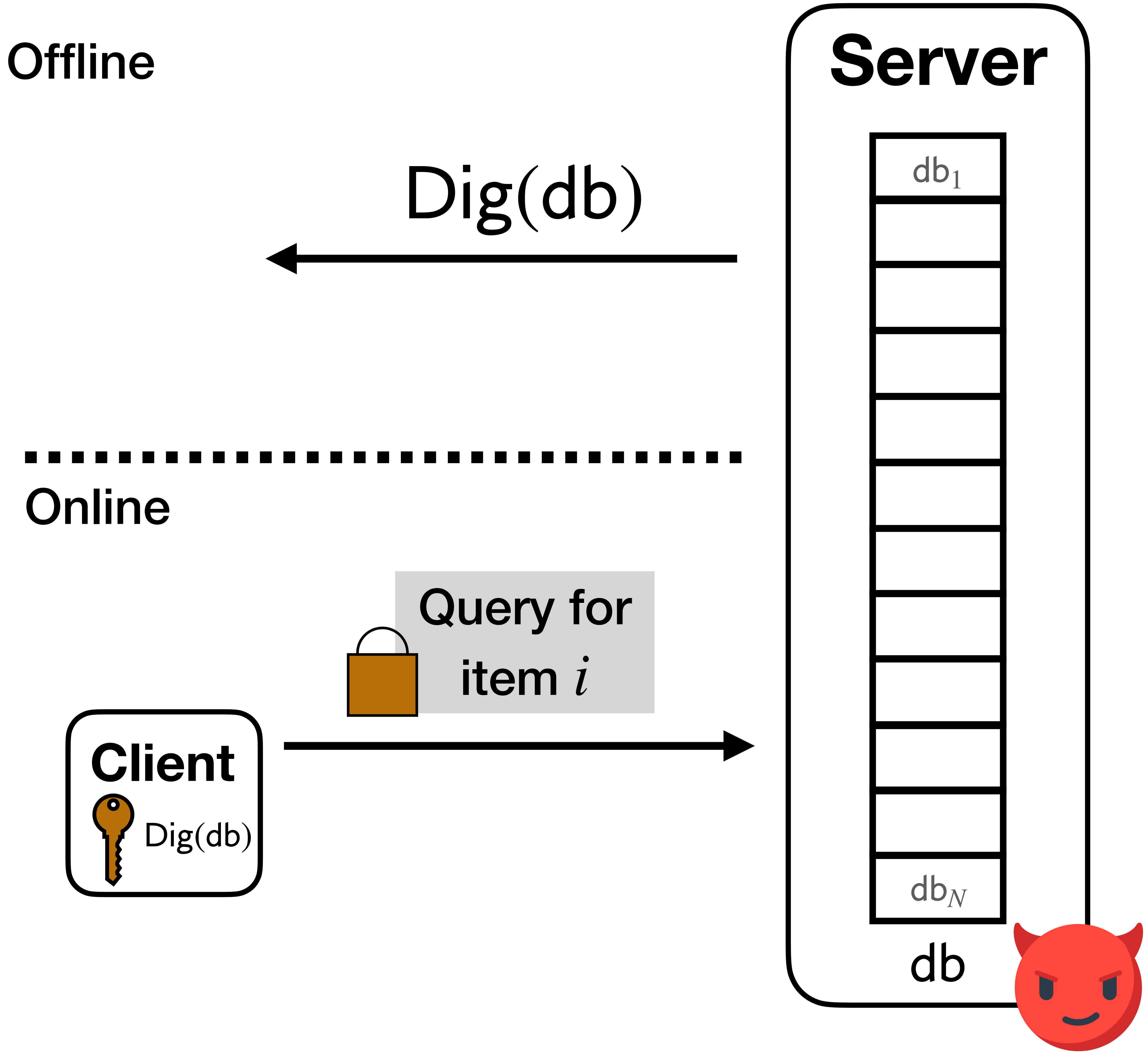
Maliciously-secure PIR (mPIR) [CNCWF23; WZLY23; DT23; CL24]



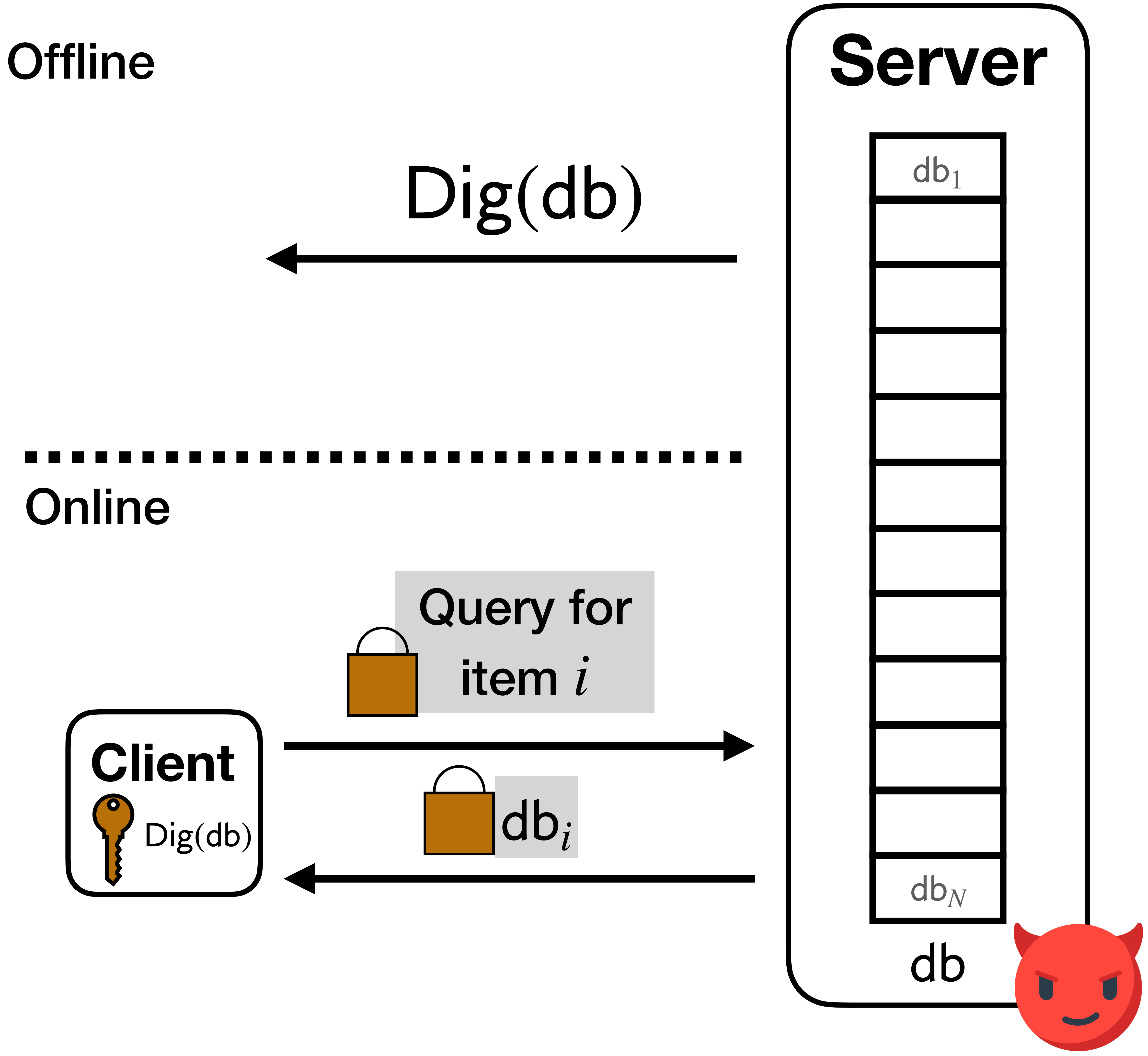
Maliciously-secure PIR (mPIR) [CNCWF23; WZLY23; DT23; CL24]



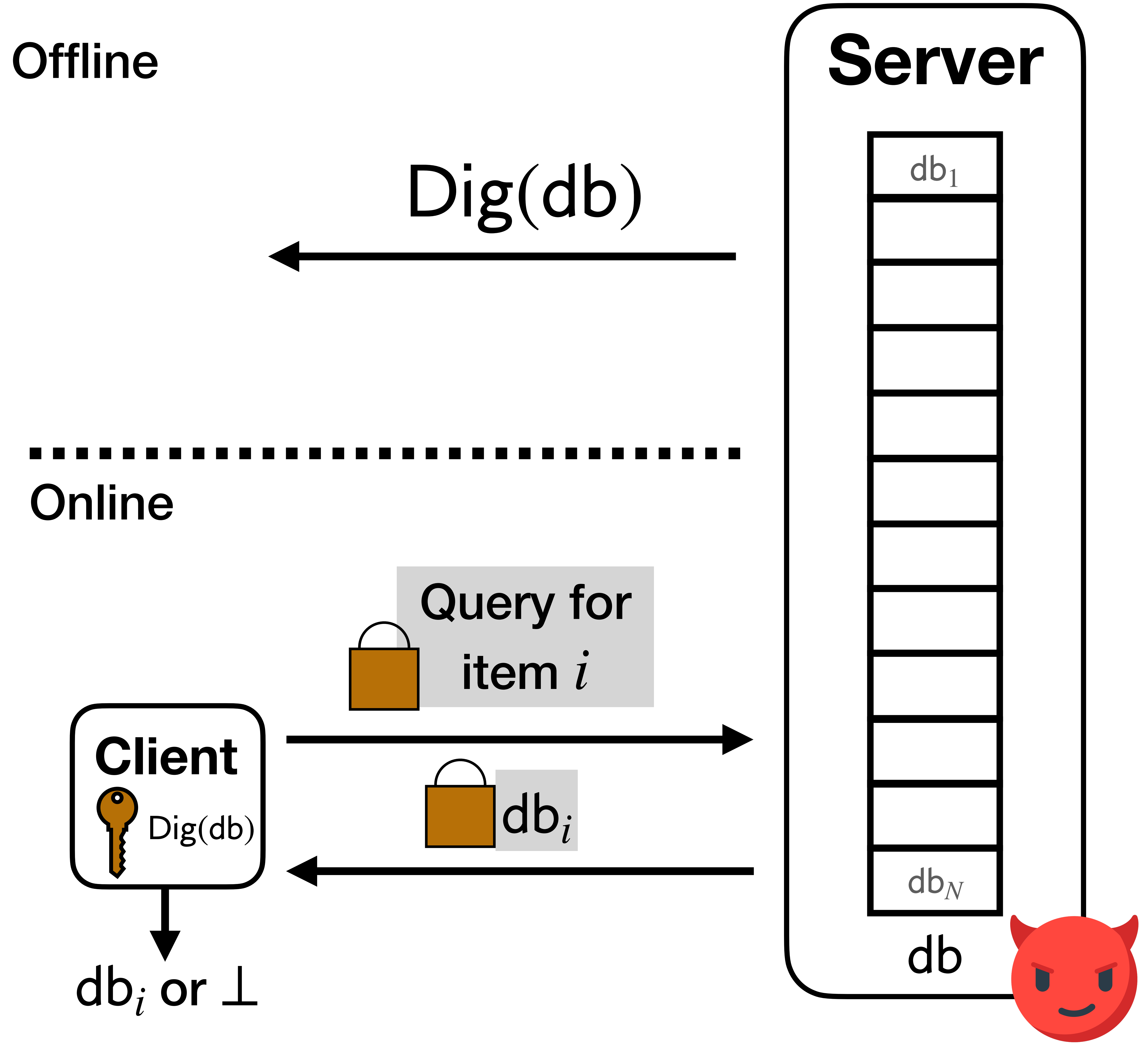
Maliciously-secure **PIR (mPIR)** [CNCWF23; WZLY23; DT23; CL24]



Maliciously-secure PIR (mPIR) [CNCWF23; WZLY23; DT23; CL24]

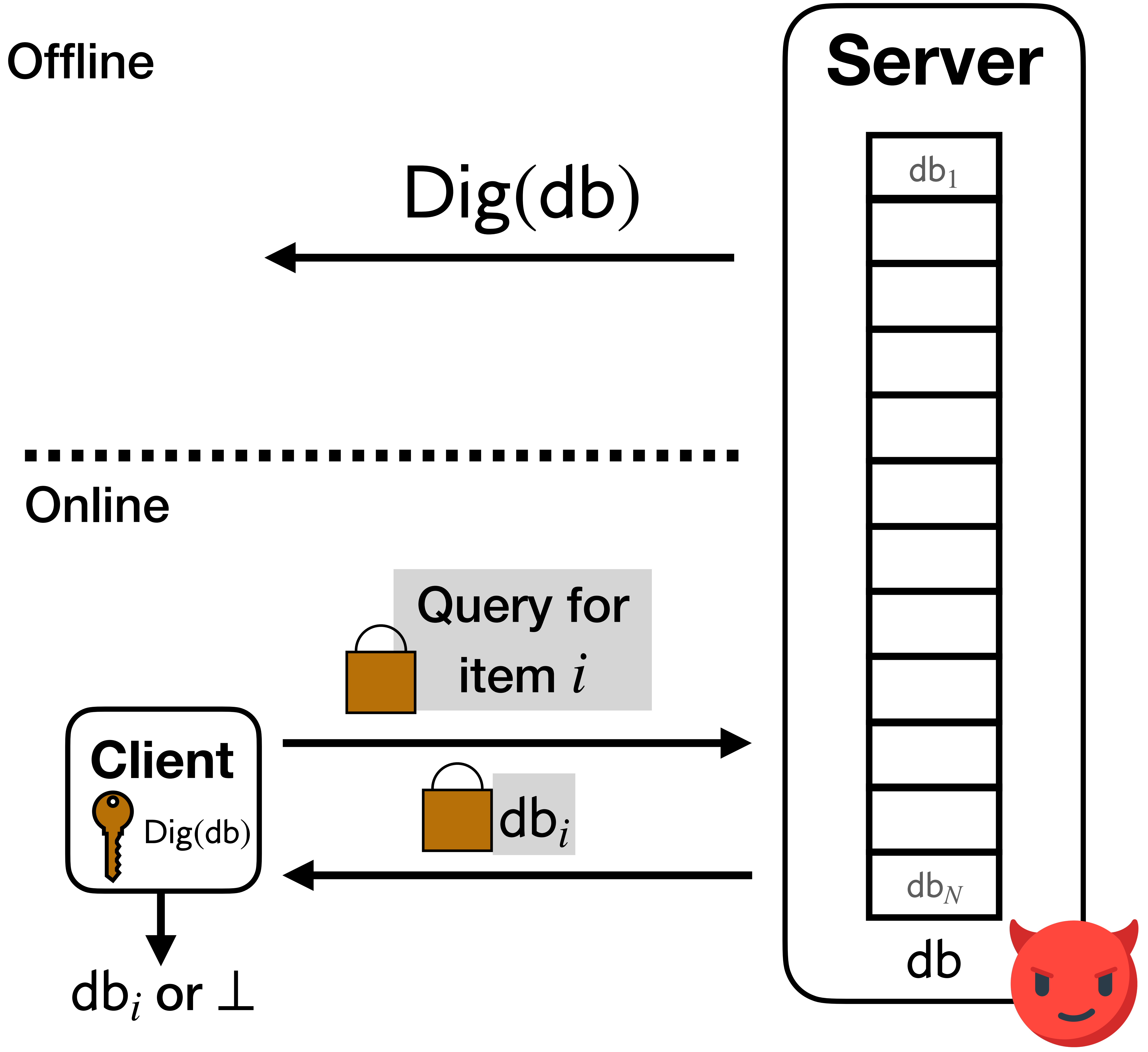


Maliciously-secure PIR (mPIR) [CNCWF23; WZLY23; DT23; CL24]



Maliciously-secure PIR (mPIR) [CNCWF23; WZLY23; DT23; CL24]

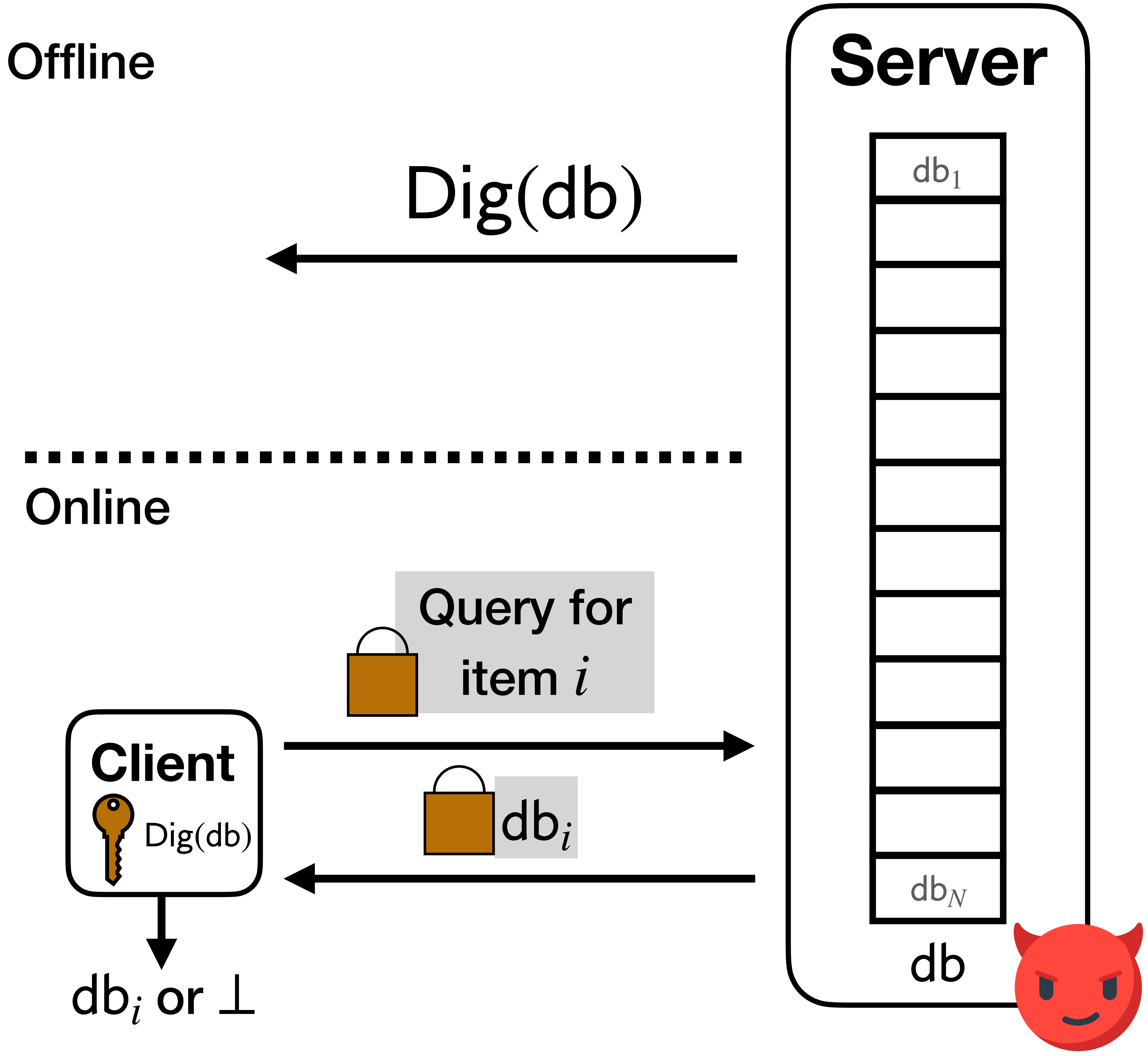
Properties:



Maliciously-secure PIR (mPIR) [CNCWF23; WZLY23; DT23; CL24]

Properties:

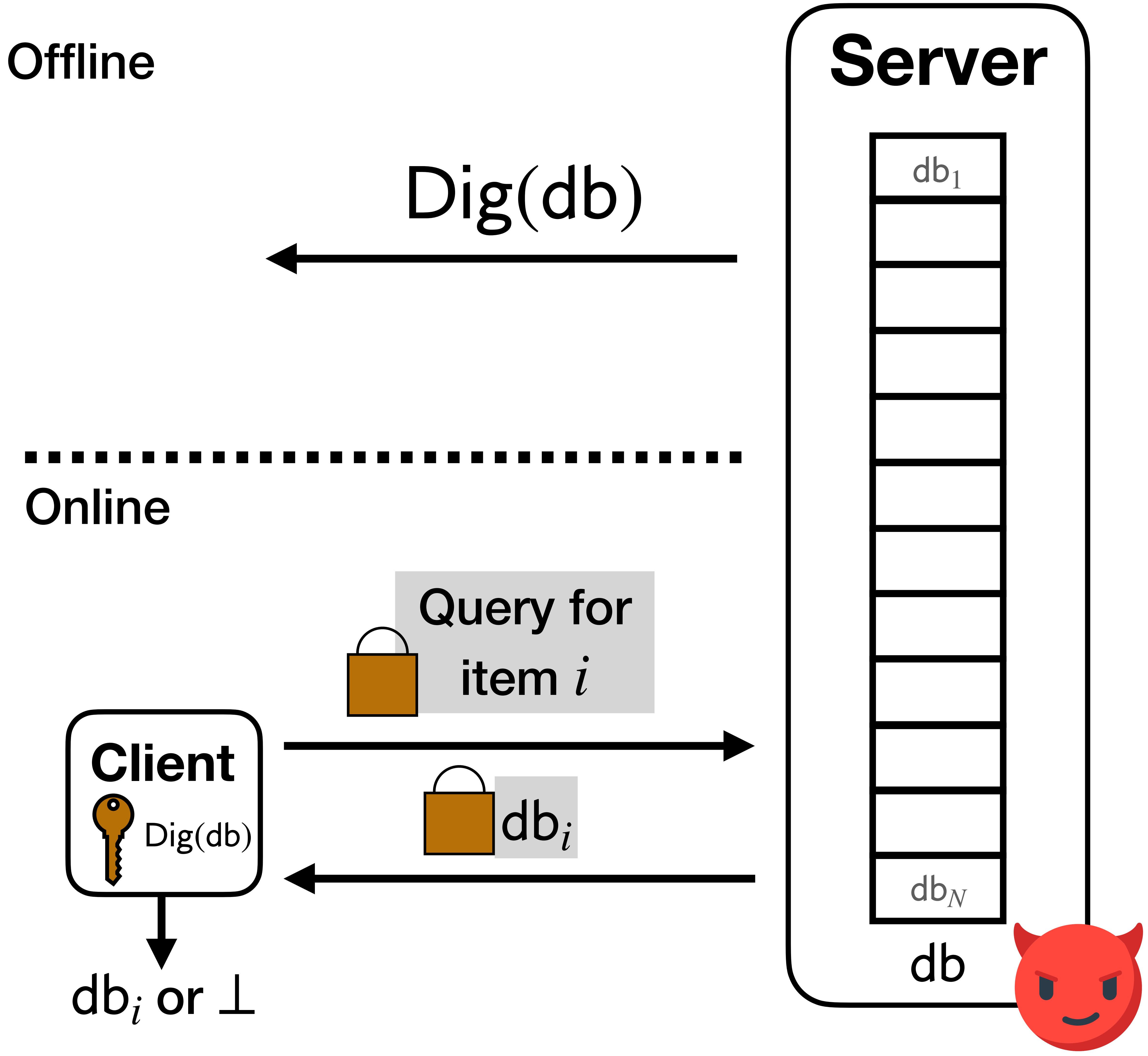
1. Correctness: if client and server are honest, client outputs db_i.



Maliciously-secure PIR (mPIR) [CNCWF23; WZLY23; DT23; CL24]

Properties:

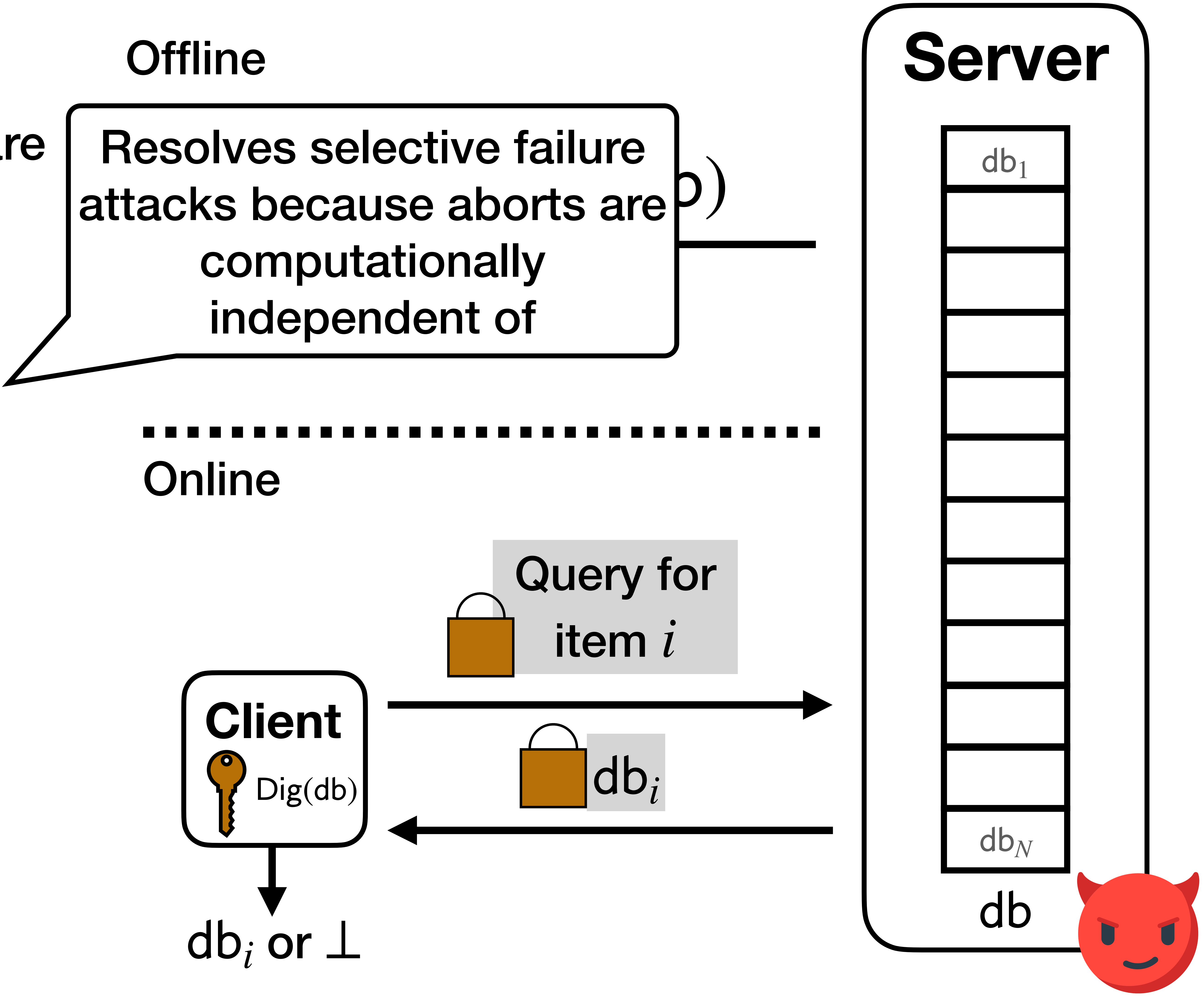
1. Correctness: if client and server are honest, client outputs db_i.
2. Privacy: server does not learn i even if it learns whether client's output is $\perp$.



Maliciously-secure PIR (mPIR) [CNCWF23; WZLY23; DT23; CL24]

Properties:

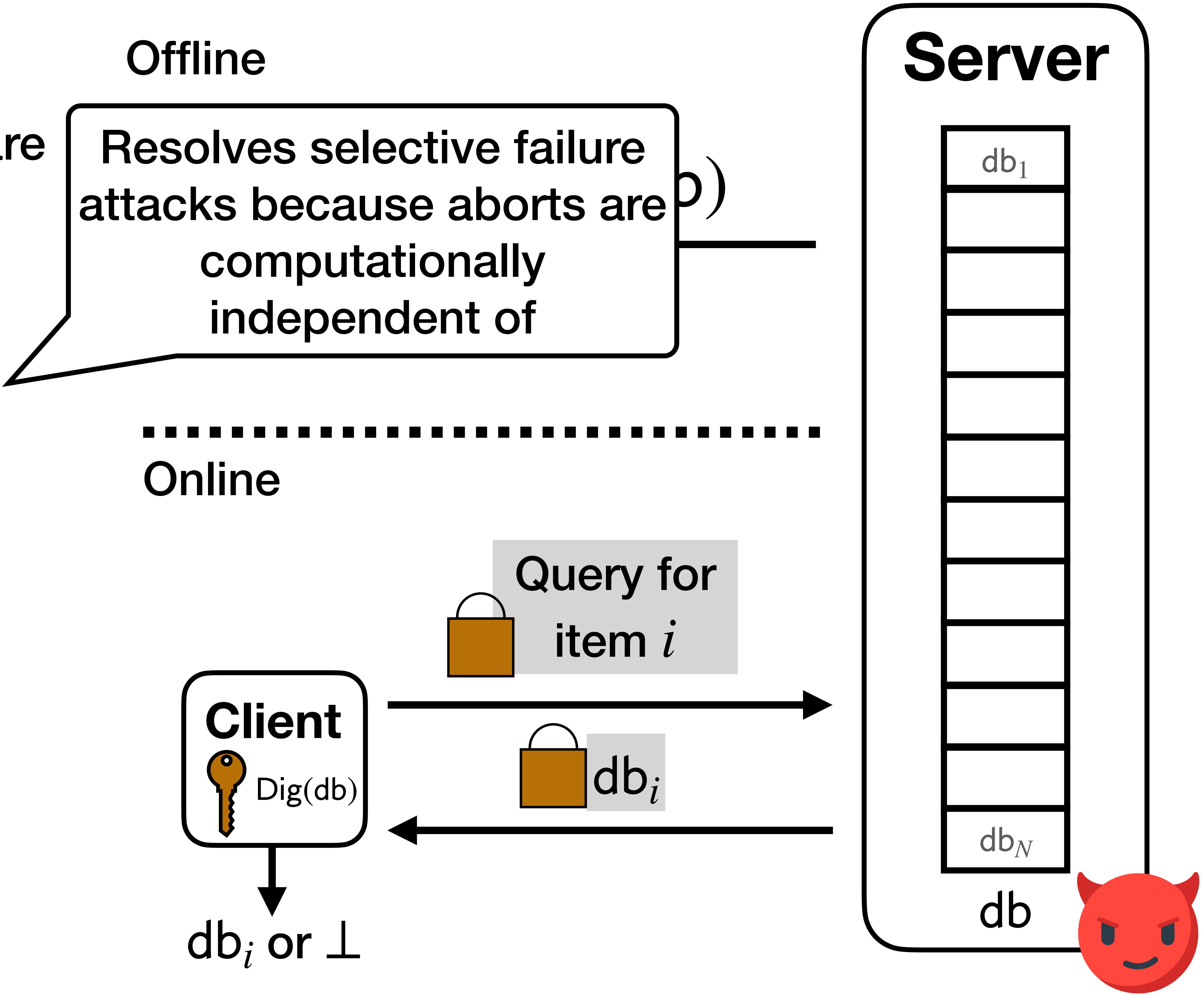
1. Correctness: if client and server are honest, client outputs db_i.
2. Privacy: server does not learn i even if it learns whether client's output is $\perp$.



Maliciously-secure **PIR (mPIR)** [CNCWF23; WZLY23; DT23; CL24]

Properties:

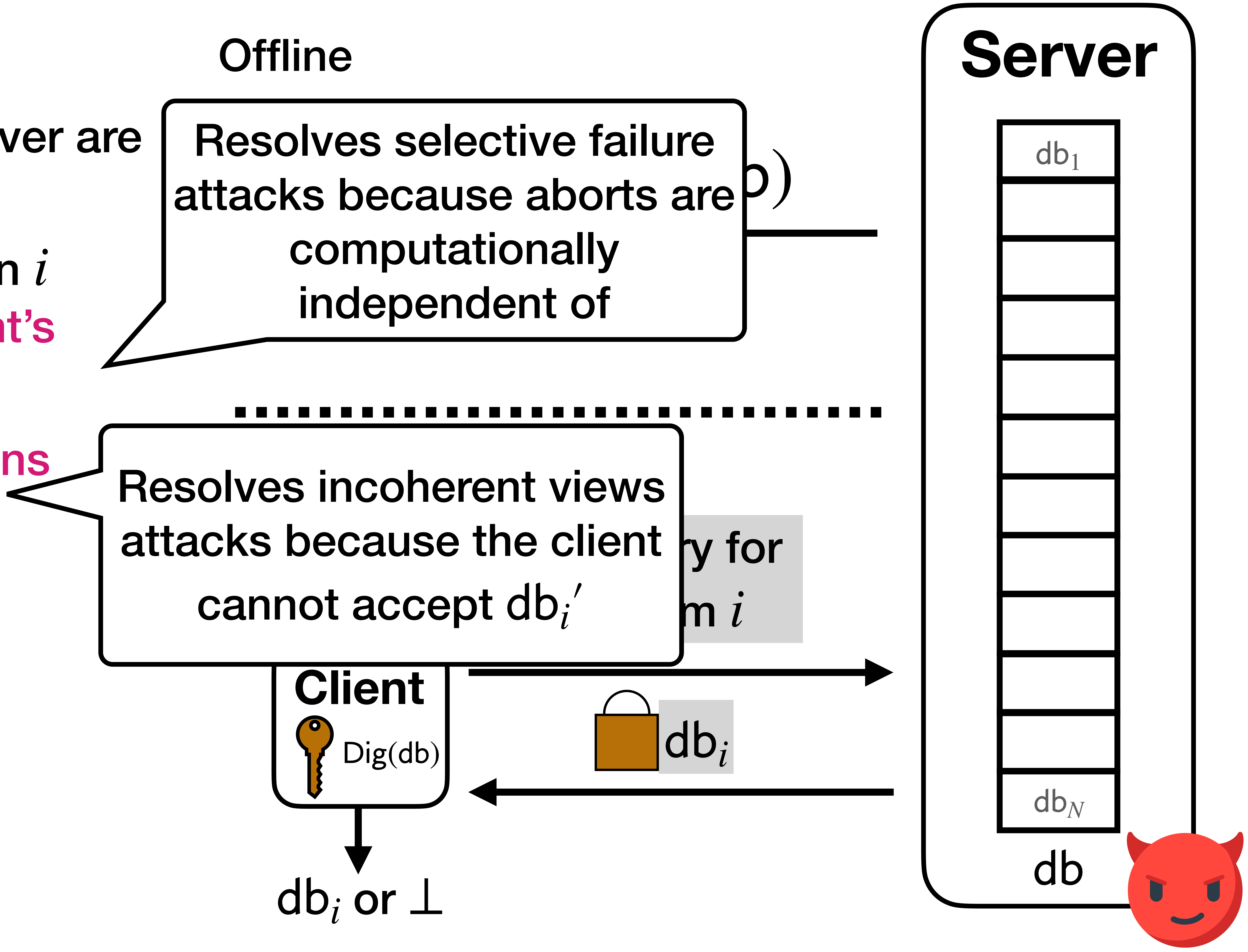
1. Correctness: if client and server are honest, client outputs db_i.
2. Privacy: server does not learn i even if it learns whether client's output is $\perp$.
3. Coherence: a query to i returns either db_i or $\perp$.



Maliciously-secure PIR (mPIR) [CNCWF23; WZLY23; DT23; CL24]

Properties:

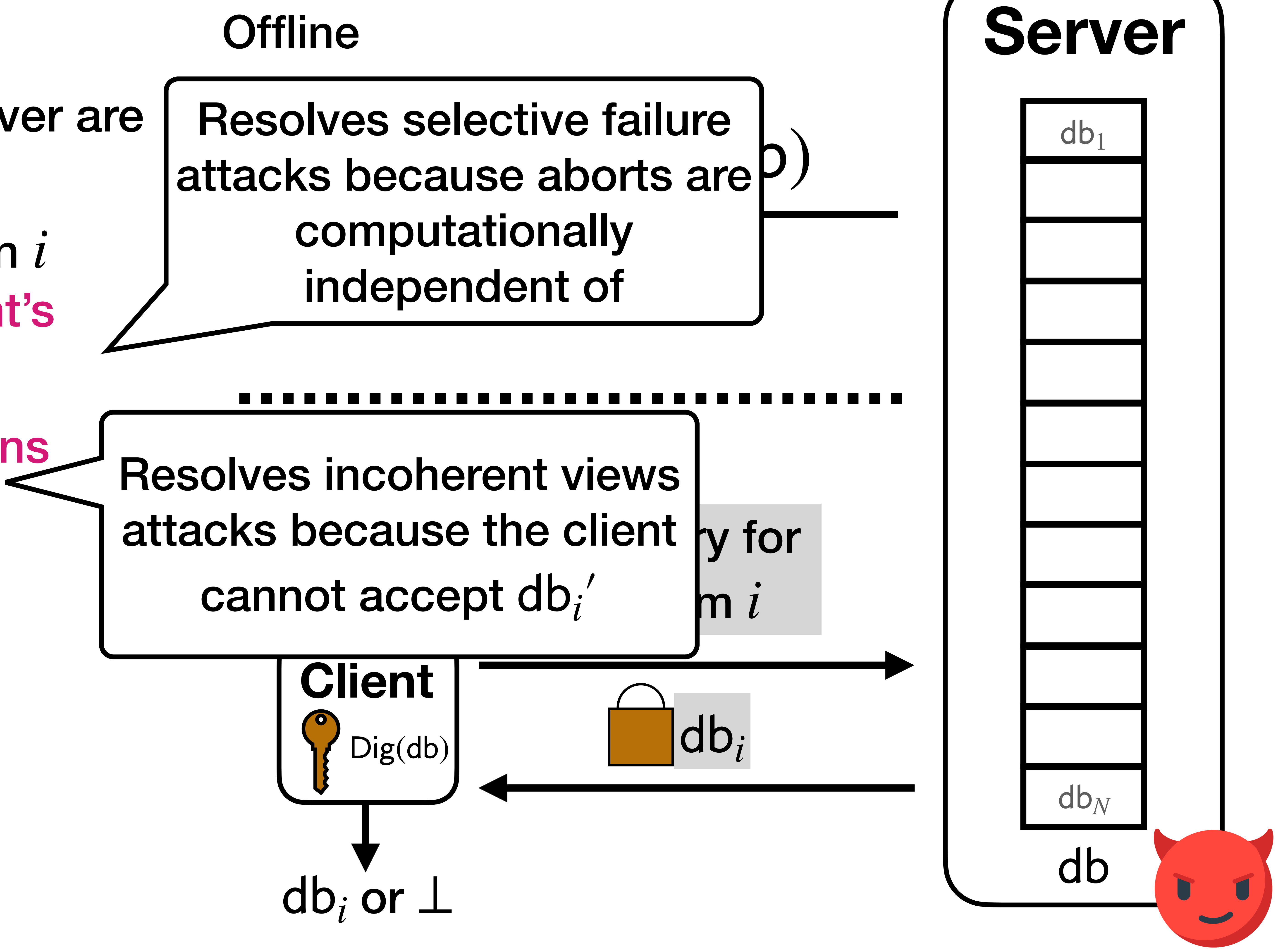
1. Correctness: if client and server are honest, client outputs db_i.
2. Privacy: server does not learn i even if it learns whether client's output is $\bot$.
3. Coherence: a query to i returns either db_i or $\bot$.



Maliciously-secure PIR (mPIR) [CNCWF23; WZLY23; DT23; CL24]

Properties:

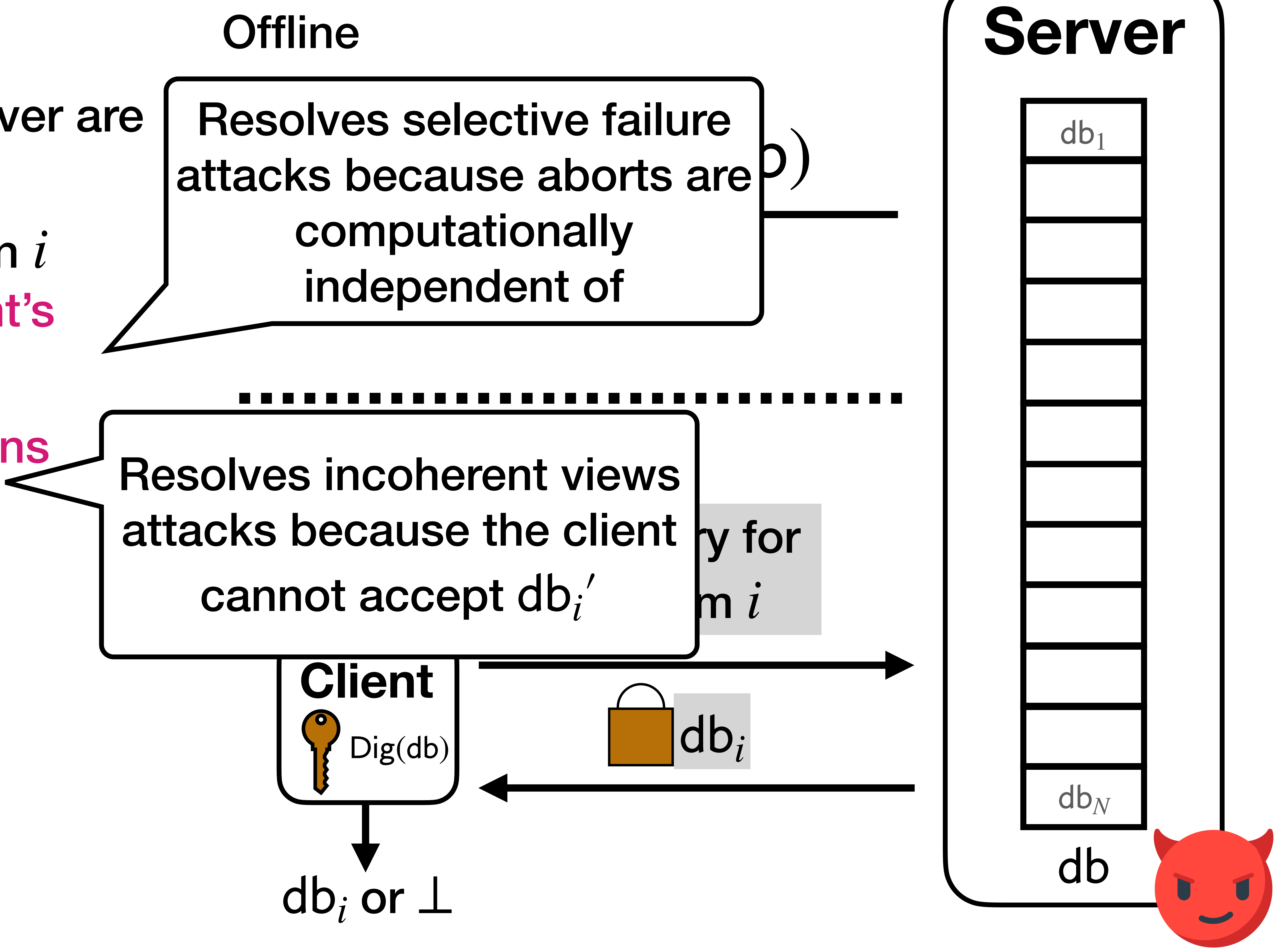
1. Correctness: if client and server are honest, client outputs db_i.
2. Privacy: server does not learn i even if it learns whether client's output is $\bot$.
3. Coherence: a query to i returns either db_i or $\bot$.
4. Efficiency: communication & computation are "low."



Maliciously-secure PIR (mPIR) [CNCWF23; WZLY23; DT23; CL24]

Properties:

1. Correctness: if client and server are honest, client outputs db_i.
2. Privacy: server does not learn i even if it learns whether client's output is $\perp$.
3. Coherence: a query to i returns either db_i or $\perp$.
4. Efficiency: communication & computation are "low."



* Throughout this talk we assume the digest is produced honestly. In the paper we show how to work around that.

Prior work: are we done?

Prior work: are we done?

Scheme	Communication	Computation	Digest size	Assumptions	Methodology
CNCWF23	$O\left(N^{1/2}\right)$	$O(N)$	$O\left(N^{1/2}\right)$	LWE, DDH	Ad-hoc
WZLY23*	$O\left(N^{1/2}\right)$	$O\left(N^{1/2}\right)$	$O\left(N^{1/2}\right)$	OWF*	Ad-hoc
DT23	$O\left(N^{1/2}\right)$	$O(N)$	$O\left(N^{1/2}\right)$	DDH	Ad-hoc
CL24	$O\left(N^{1/2}\right)$	$O(N)$	$O\left(N^{1/2}\right)$	LWE	Ad-hoc

Prior work: are we done?

Scheme	Communication	Computation	Digest size	Assumptions	Methodology
CNCWF23	$O\left(N^{1/2}\right)$	$O(N)$	$O\left(N^{1/2}\right)$	LWE, DDH	Ad-hoc
WZLY23*	$O\left(N^{1/2}\right)$	$O\left(N^{1/2}\right)$	$O\left(N^{1/2}\right)$	OWF*	Ad-hoc
DT23	$O\left(N^{1/2}\right)$	$O(N)$	$O\left(N^{1/2}\right)$	DDH	Ad-hoc
CL24	$O\left(N^{1/2}\right)$	$O(N)$	$O\left(N^{1/2}\right)$	LWE	Ad-hoc

Gaps:

Prior work: are we done?

Scheme	Communication	Computation	Digest size	Assumptions	Methodology
CNCWF23	$O\left(N^{1/2}\right)$	$O\left(N\right)$	$O\left(N^{1/2}\right)$	LWE, DDH	Ad-hoc
WZLY23*	$O\left(N^{1/2}\right)$	$O\left(N^{1/2}\right)$	$O\left(N^{1/2}\right)$	OWF*	Ad-hoc
DT23	$O\left(N^{1/2}\right)$	$O\left(N\right)$	$O\left(N^{1/2}\right)$	DDH	Ad-hoc
CL24	$O\left(N^{1/2}\right)$	$O\left(N\right)$	$O\left(N^{1/2}\right)$	LWE	Ad-hoc

Gaps:

1. Methodology: direct constructions

Prior work: are we done?

Scheme	Communication	Computation	Digest size	Assumptions	Methodology
CNCWF23	$O\left(N^{1/2}\right)$	$O\left(N\right)$	$O\left(N^{1/2}\right)$	LWE, DDH	Ad-hoc
WZLY23*	$O\left(N^{1/2}\right)$	$O\left(N^{1/2}\right)$	$O\left(N^{1/2}\right)$	OWF*	Ad-hoc
DT23	$O\left(N^{1/2}\right)$	$O\left(N\right)$	$O\left(N^{1/2}\right)$	DDH	Ad-hoc
CL24	$O\left(N^{1/2}\right)$	$O\left(N\right)$	$O\left(N^{1/2}\right)$	LWE	Ad-hoc

Gaps:

1. Methodology: direct constructions
2. Assumptions: limited

Prior work: are we done?

Scheme	Communication	Computation	Digest size	Assumptions	Methodology
CNCWF23	$O\left(N^{1/2}\right)$	$O\left(N\right)$	$O\left(N^{1/2}\right)$	LWE, DDH	Ad-hoc
WZLY23*	$O\left(N^{1/2}\right)$	$O\left(N^{1/2}\right)$	$O\left(N^{1/2}\right)$	OWF*	Ad-hoc
DT23	$O\left(N^{1/2}\right)$	$O\left(N\right)$	$O\left(N^{1/2}\right)$	DDH	Ad-hoc
CL24	$O\left(N^{1/2}\right)$	$O\left(N\right)$	$O\left(N^{1/2}\right)$	LWE	Ad-hoc

Gaps:

1. Methodology: direct constructions
2. Assumptions: limited
3. PIR Overhead: $\Omega\left(N^{1/2}\right)$

Contributions

Contributions

Contributions

Theorem 1:
maliciousness compiler
for PIR

Contributions

PIR

Theorem 1:
maliciousness compiler
for PIR

Contributions

PIR

VC

Theorem 1:
maliciousness compiler
for PIR

Contributions

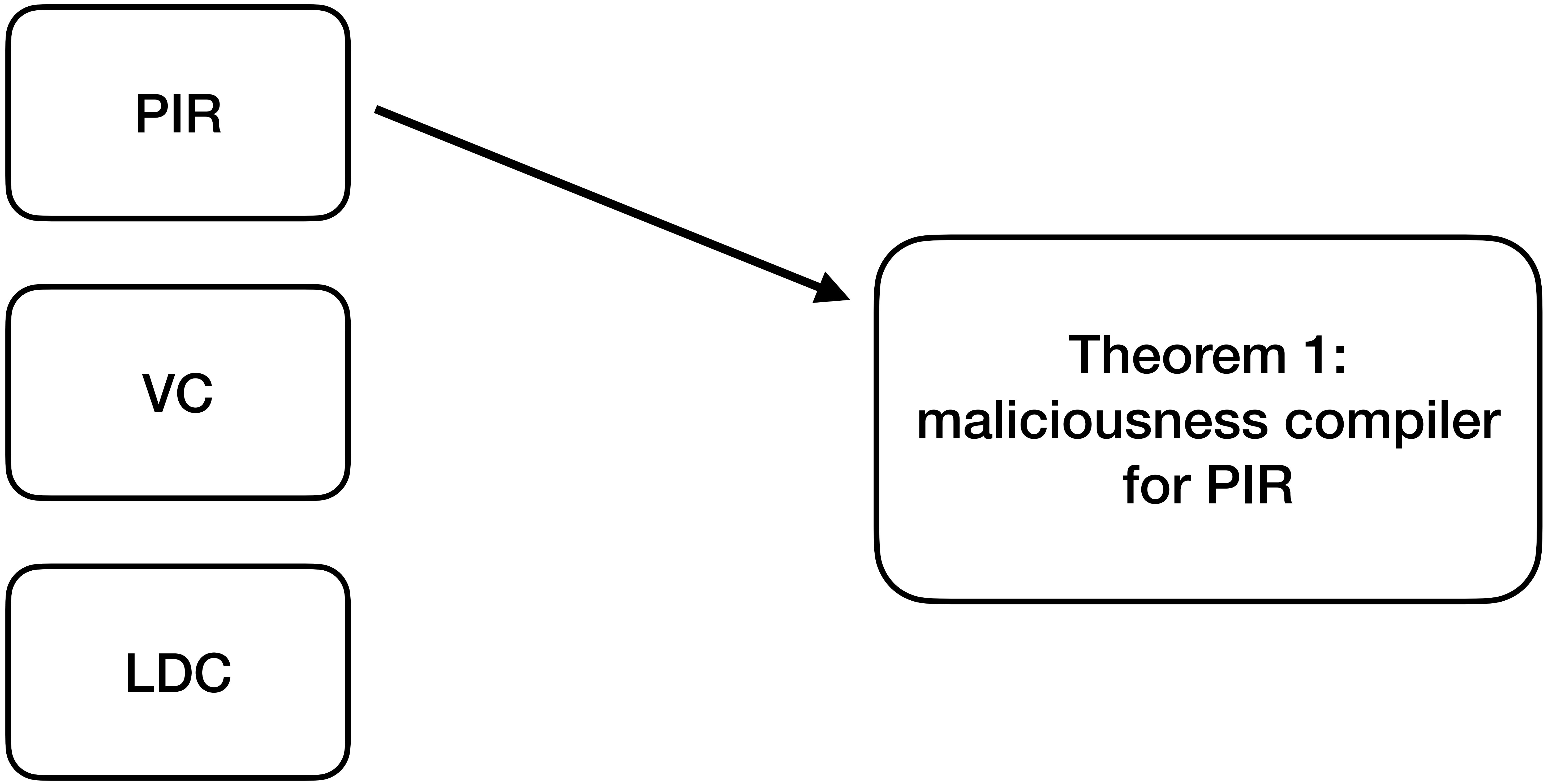
PIR

VC

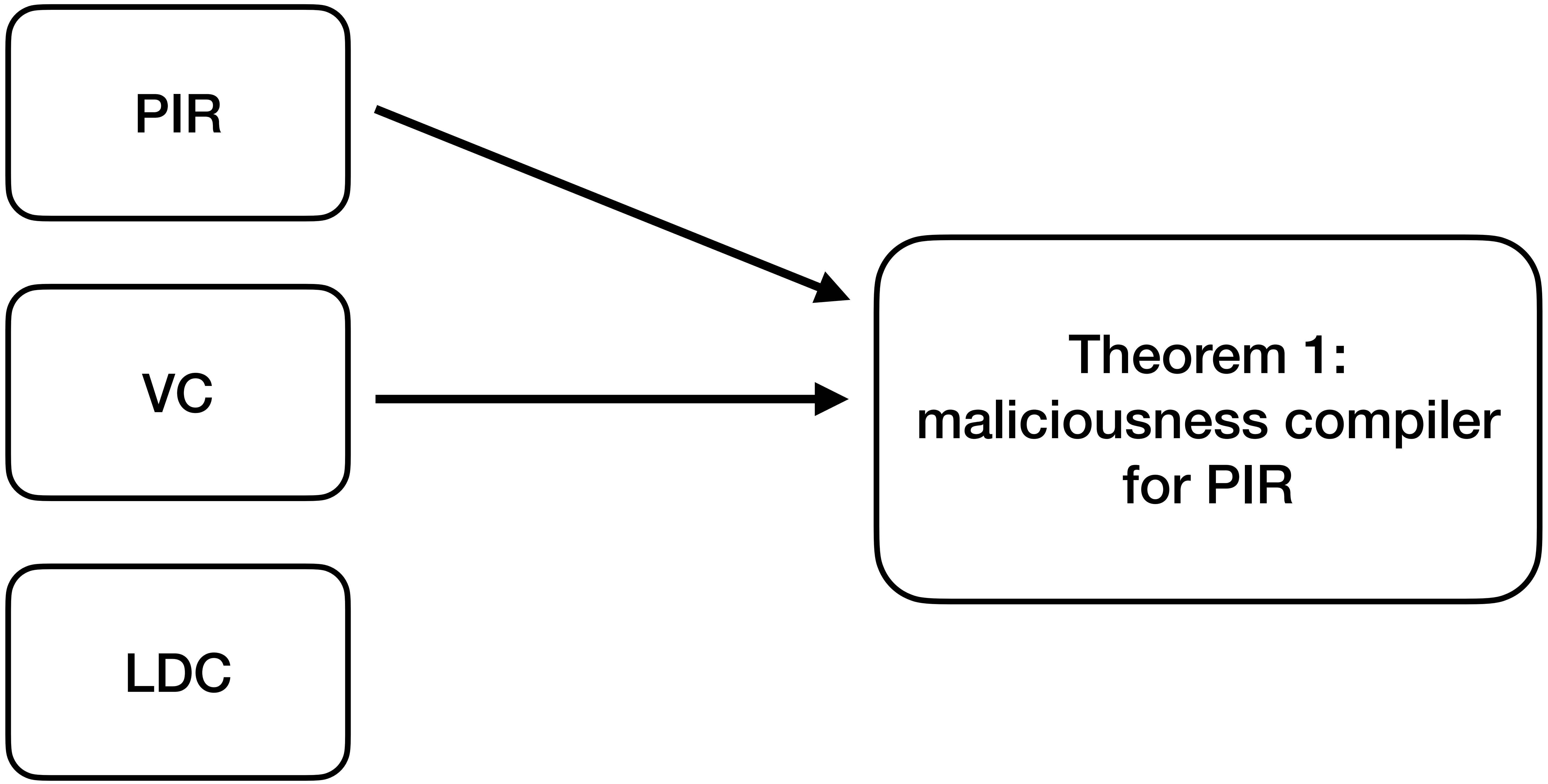
LDC

Theorem 1:
maliciousness compiler
for PIR

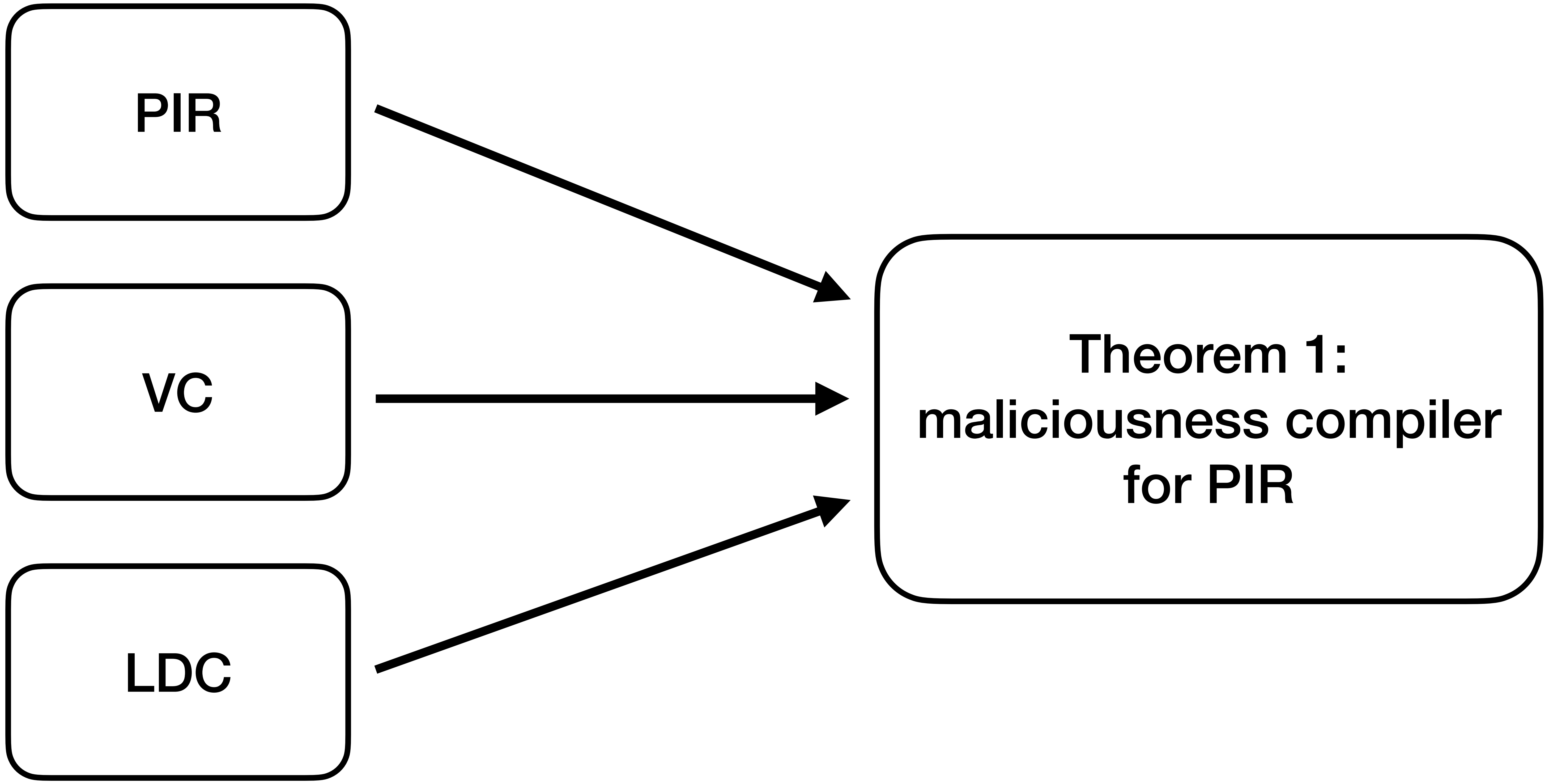
Contributions



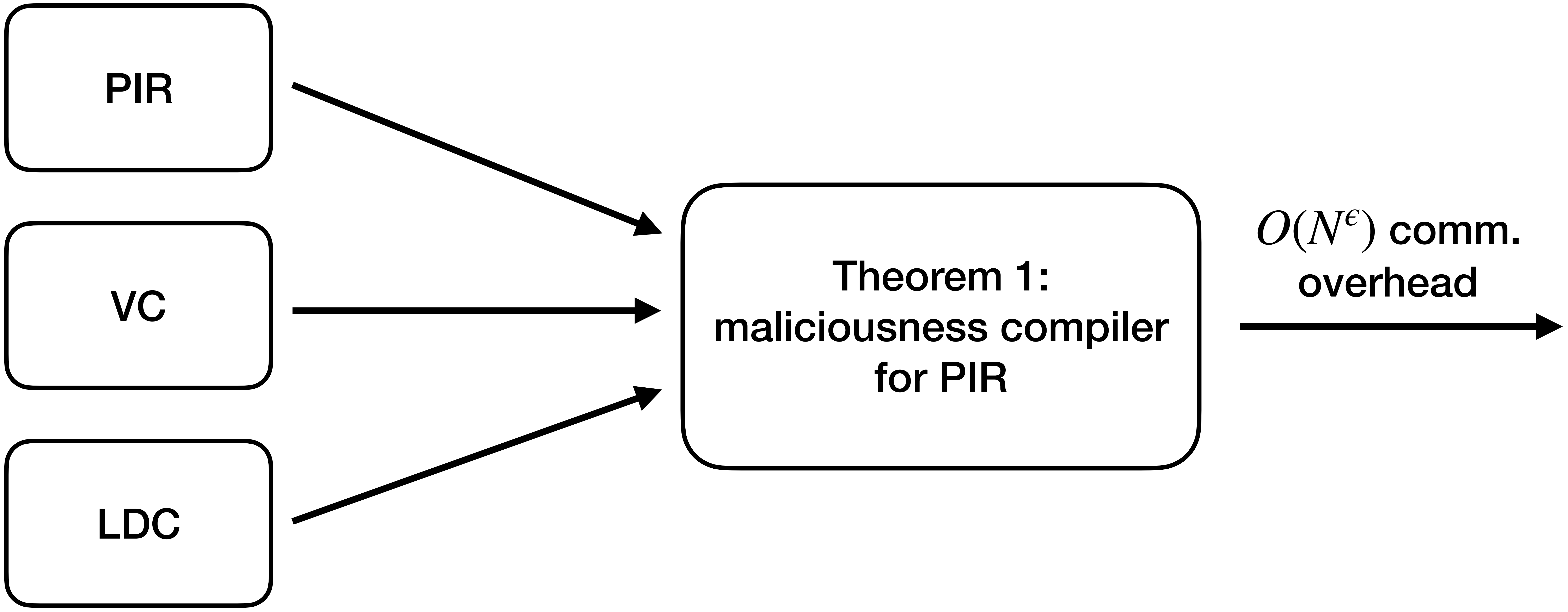
Contributions



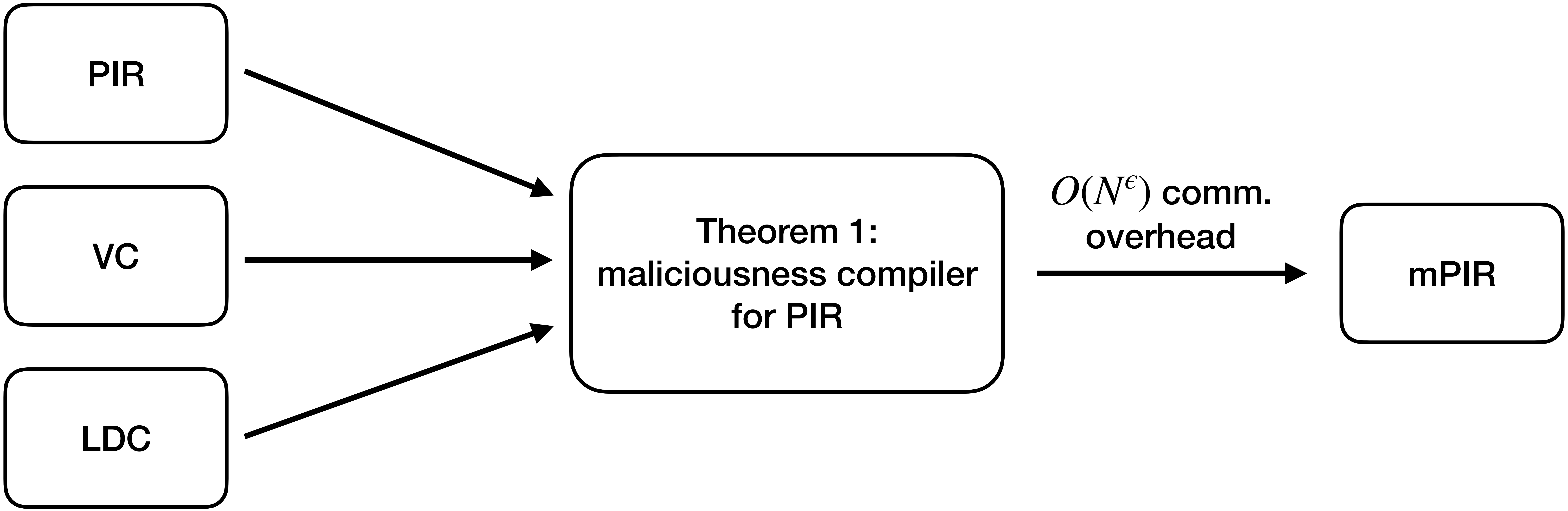
Contributions



Contributions

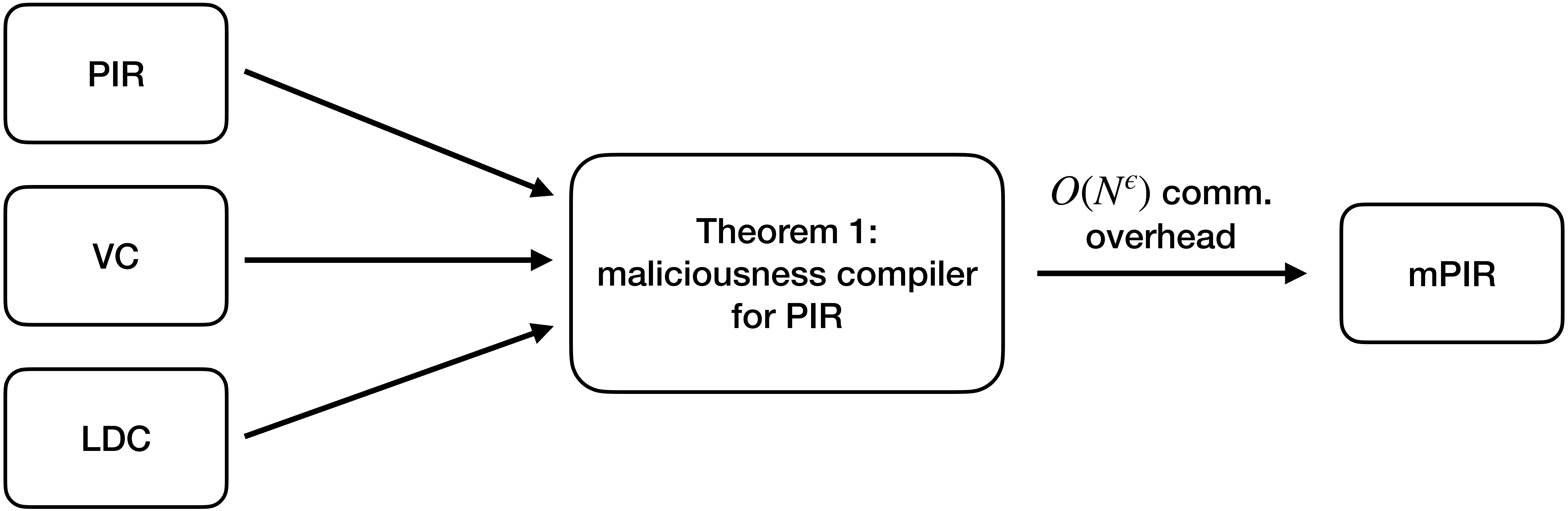


Contributions



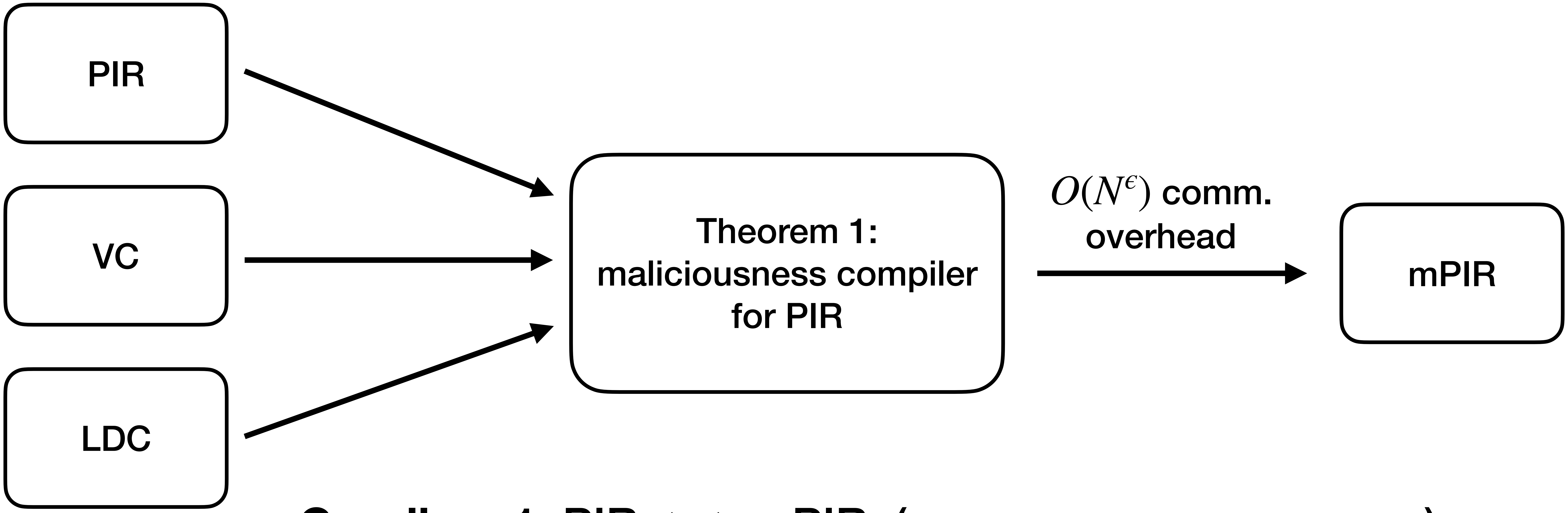
Contributions

1. Methodology: generic compiler



Contributions

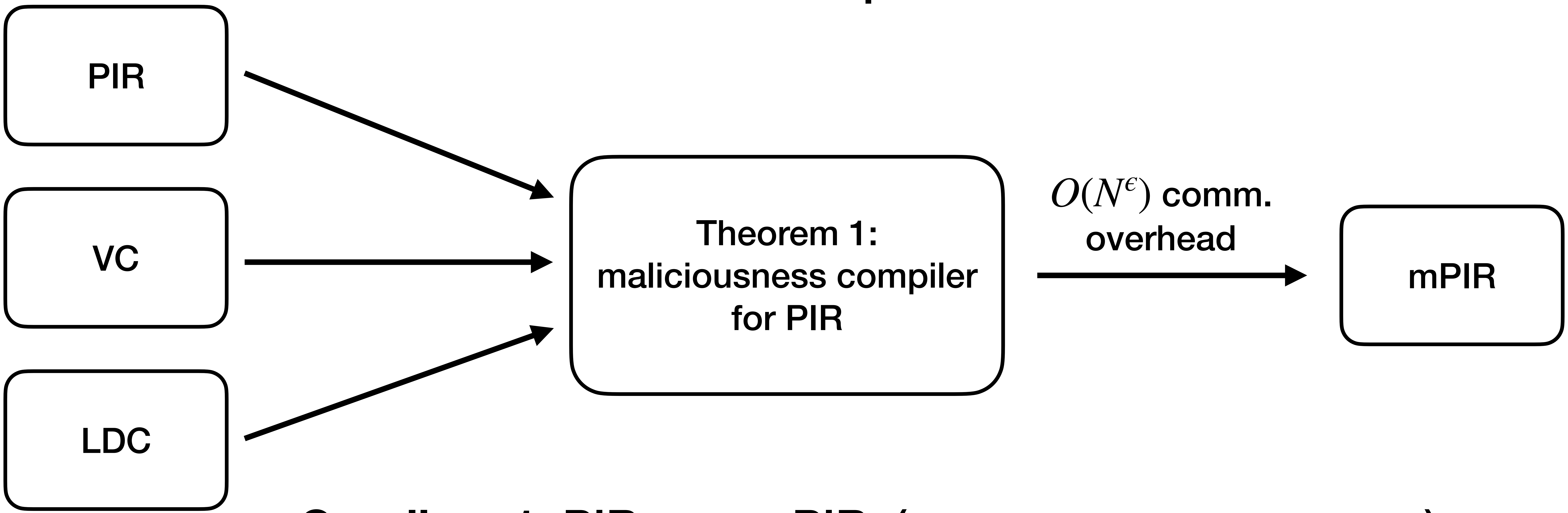
1. Methodology: generic compiler



Corollary 1: PIR $\Longleftrightarrow$ mPIR (PIR $\Longrightarrow$ CRH $\Longrightarrow$ Merkle tree (VC))

Contributions

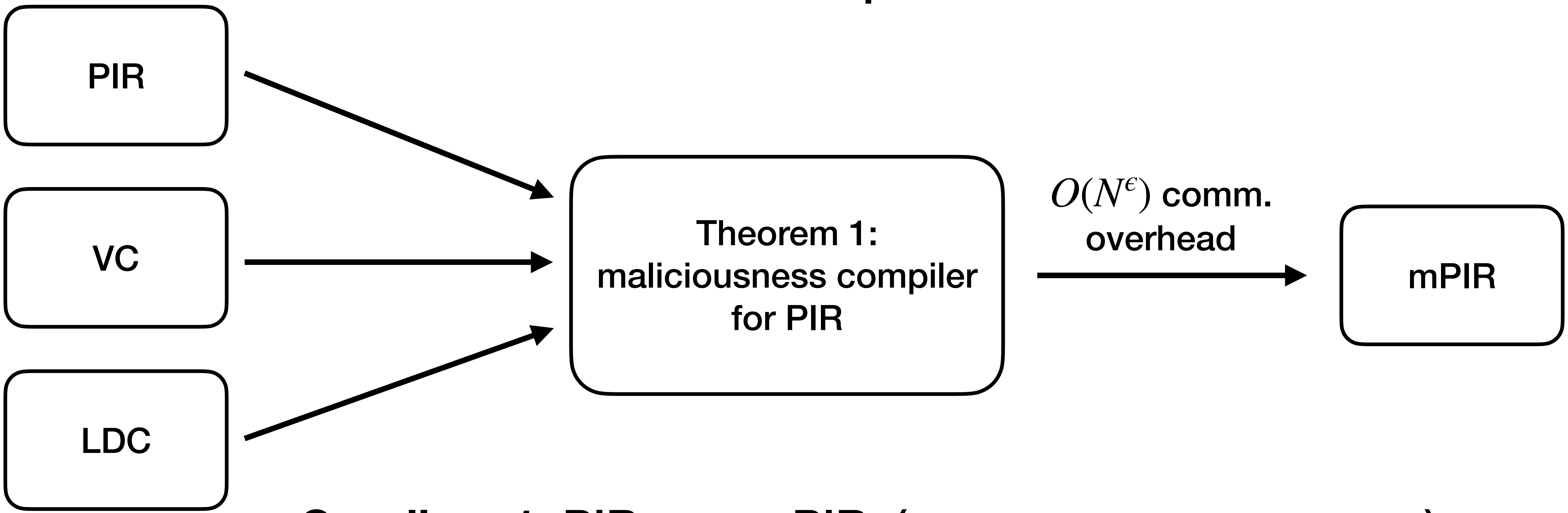
1. Methodology: generic compiler
2. Assumptions: from PIR



Corollary 1: PIR $\Longleftrightarrow$ mPIR (PIR $\Longrightarrow$ CRH $\Longrightarrow$ Merkle tree (VC))

Contributions

1. Methodology: generic compiler
2. Assumptions: from PIR

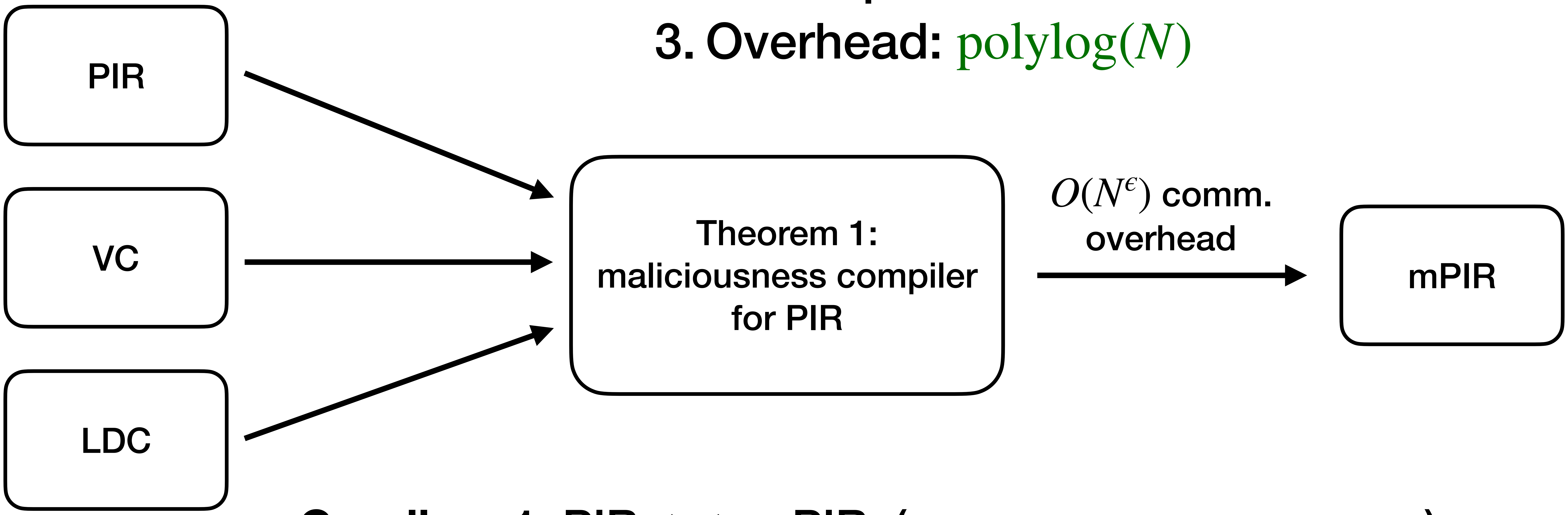


Corollary 1: PIR $\iff$ mPIR (PIR $\implies$ CRH $\implies$ Merkle tree (VC))

Theorem 2: there exists doubly-efficient ($\mathrm{polylog}(N)$) mPIR.

Contributions

1. Methodology: generic compiler
2. Assumptions: from PIR
3. Overhead: $\mathrm{polylog}(N)$



Corollary 1: PIR $\Longleftrightarrow$ mPIR (PIR $\Longrightarrow$ CRH $\Longrightarrow$ Merkle tree (VC))

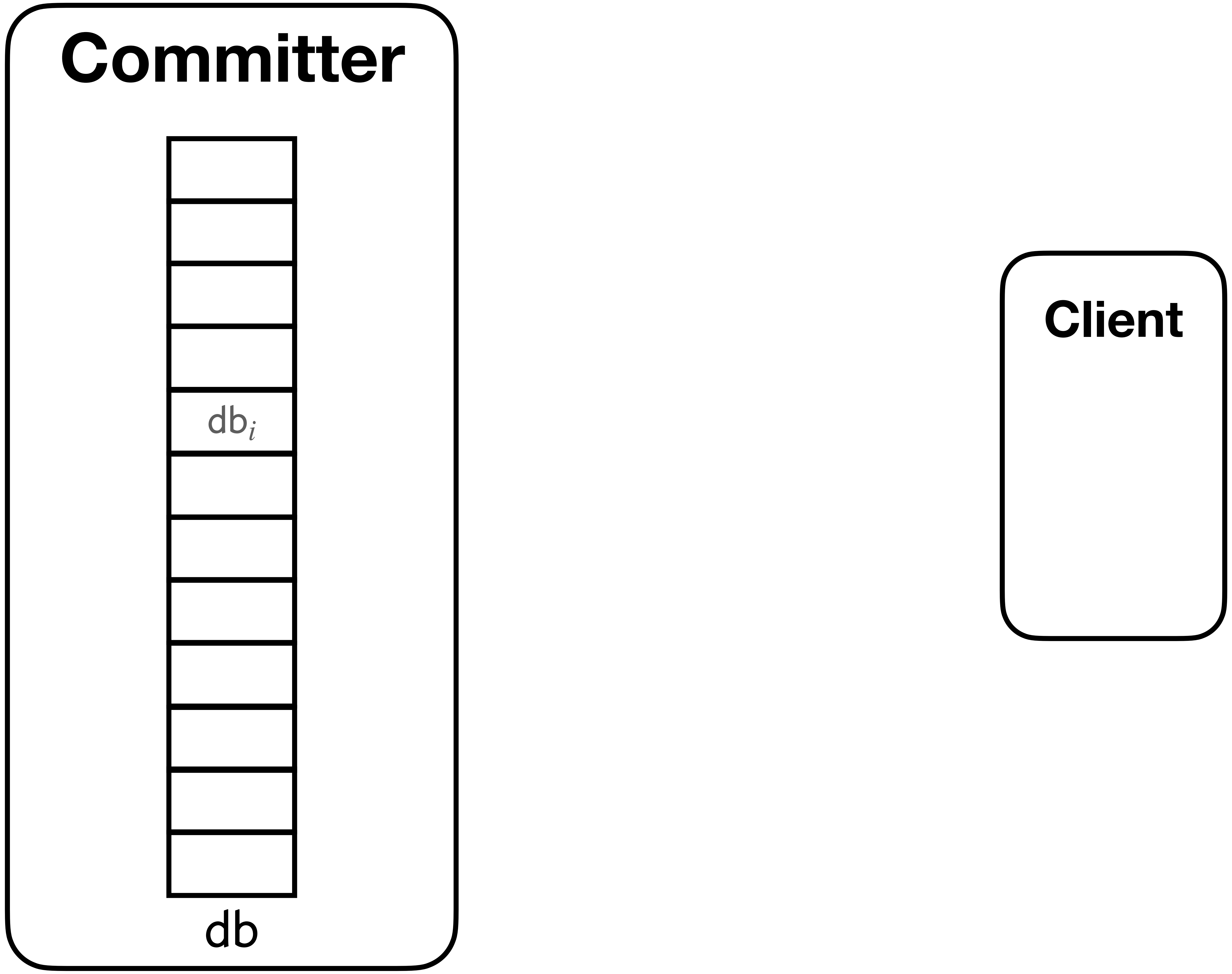
Theorem 2: there exists doubly-efficient ($\mathrm{polylog}(N)$) mPIR.

Contributions

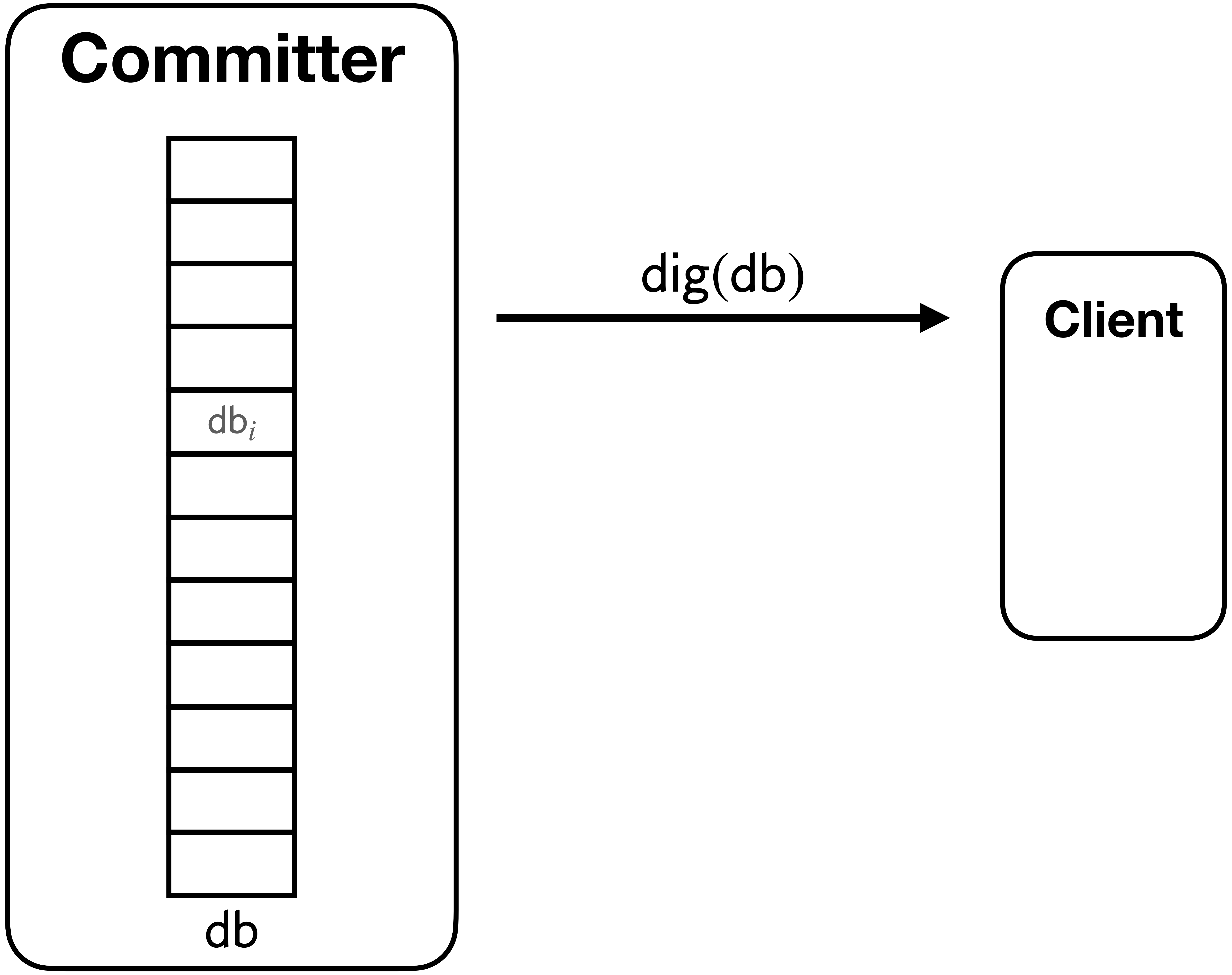
Scheme	Communication	Computation	Digest	Assumptions	Methodology
CNCWF23	$O\left(N^{1/2}\right)$	$O\left(N\right)$	$O\left(N^{1/2}\right)$	LWE, DDH	Ad-hoc
WZLY23	$O\left(N^{1/2}\right)$	$O\left(N^{1/2}\right)$	$O\left(N^{1/2}\right)$	OWF*	Ad-hoc
DT23	$O\left(N^{1/2}\right)$	$O\left(N\right)$	$O\left(N^{1/2}\right)$	DDH	Ad-hoc
CL24	$O\left(N^{1/2}\right)$	$O\left(N\right)$	$O\left(N^{1/2}\right)$	LWE	Ad-hoc
Ours (any PIR)	$\times O\left(N^{\epsilon}\right)$	$\times O(1)$	$\omega(\log N)$	PIR	Compiler
Ours (DePIR)	$O(\text{polylog } N)$	$O(\text{polylog } N)$	$\omega(\log N)$	RingLWE	Compiler

Construction

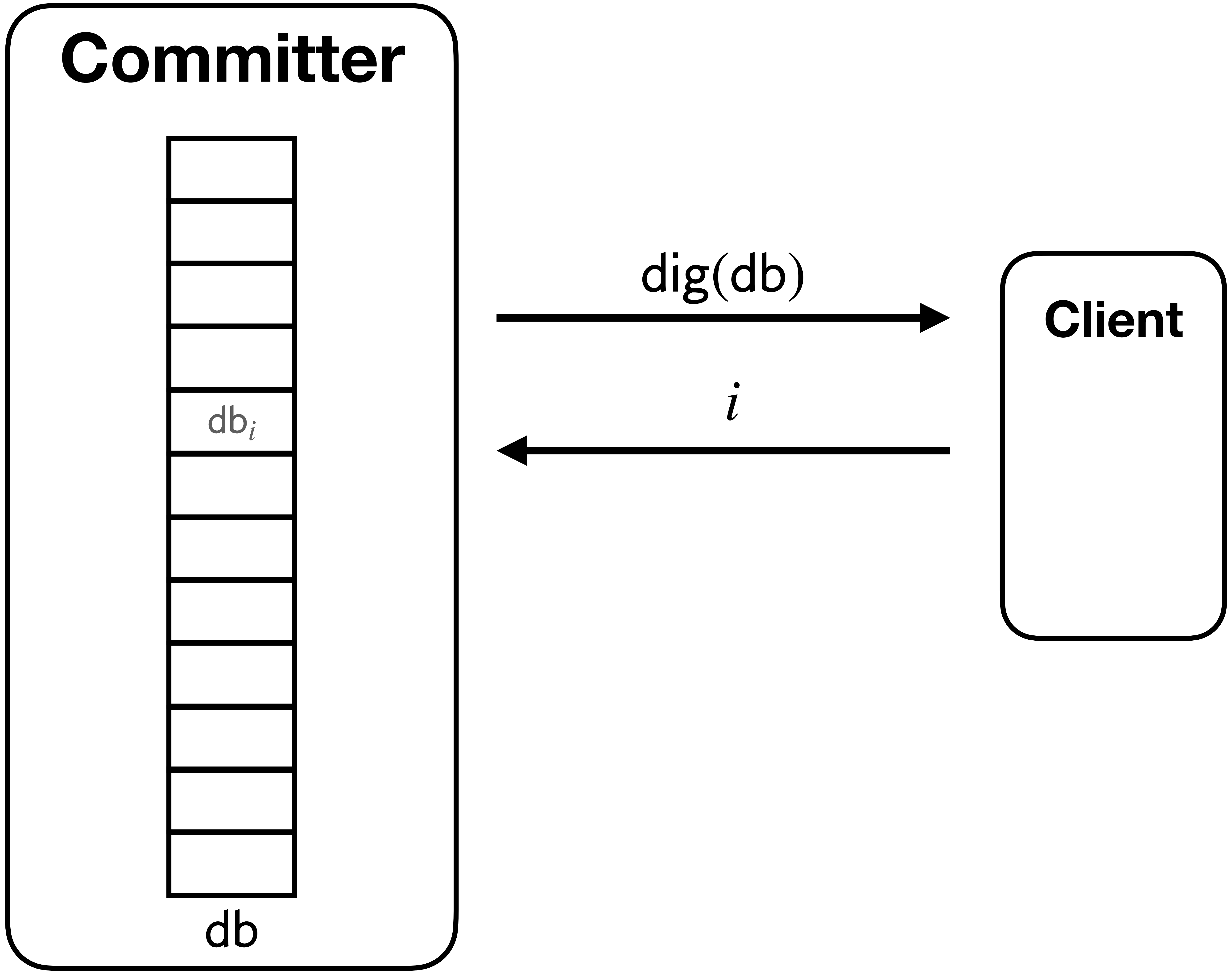
Vector Commitments (VC)



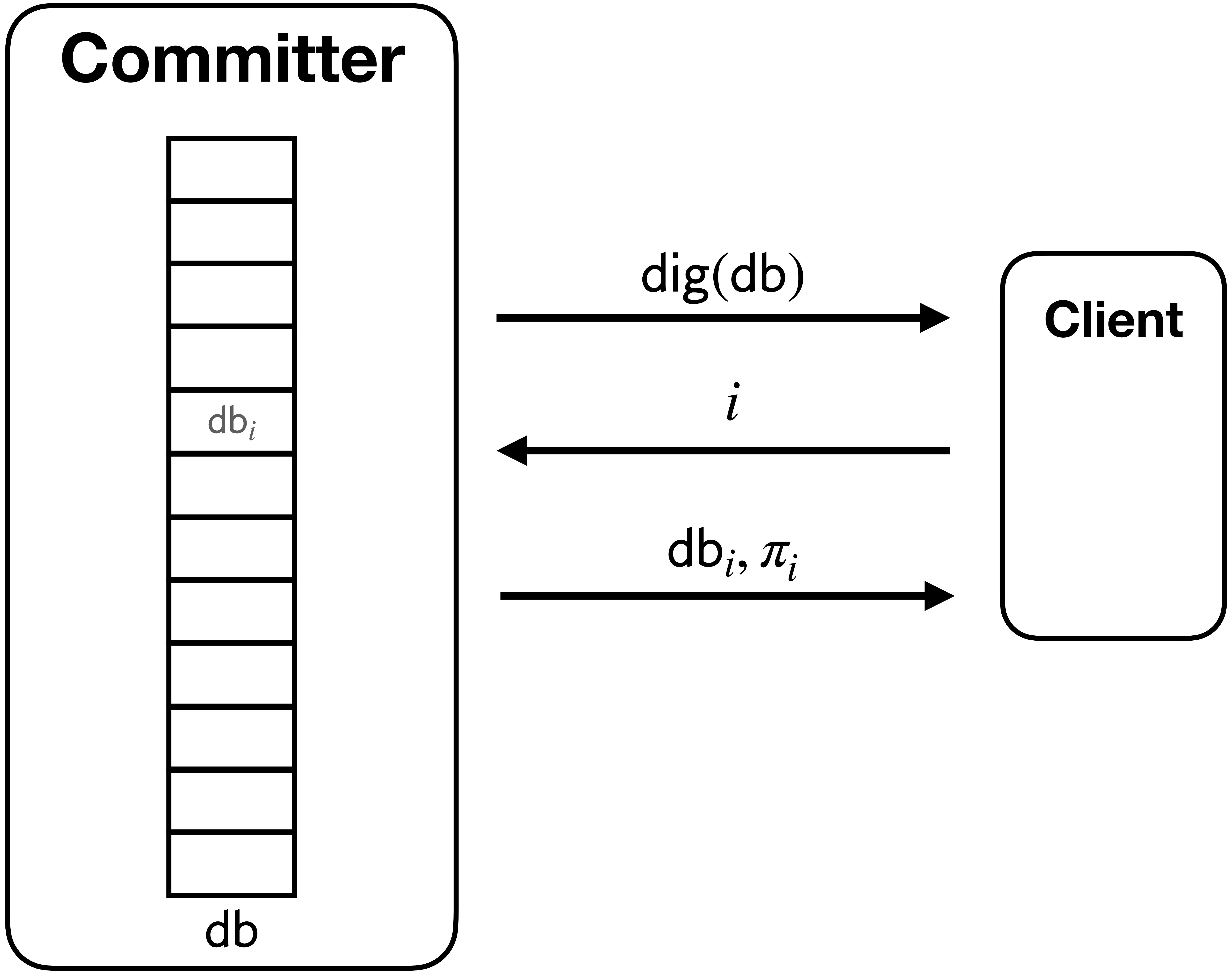
Vector Commitments (VC)



Vector Commitments (VC)

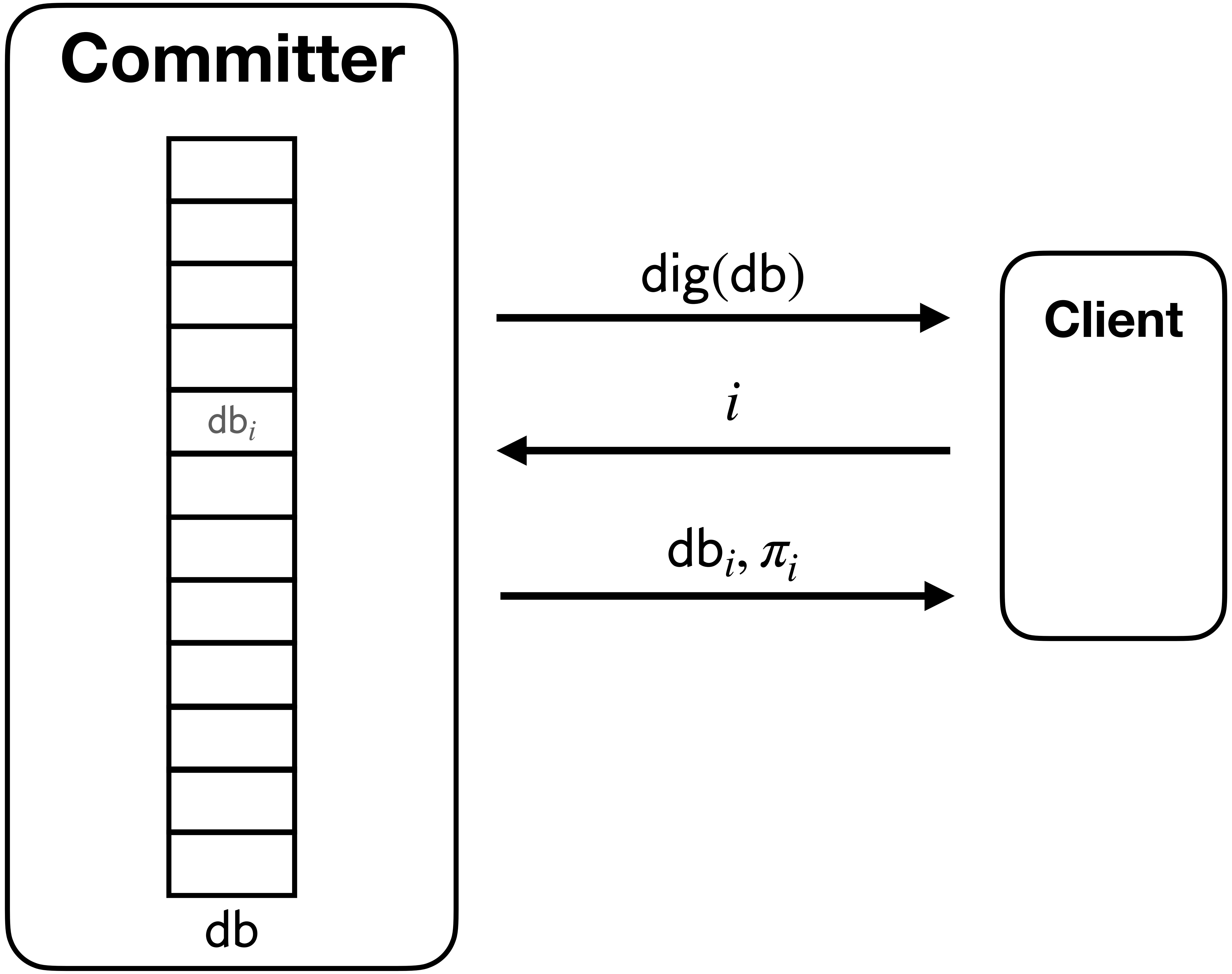


Vector Commitments (VC)



Vector Commitments (VC)

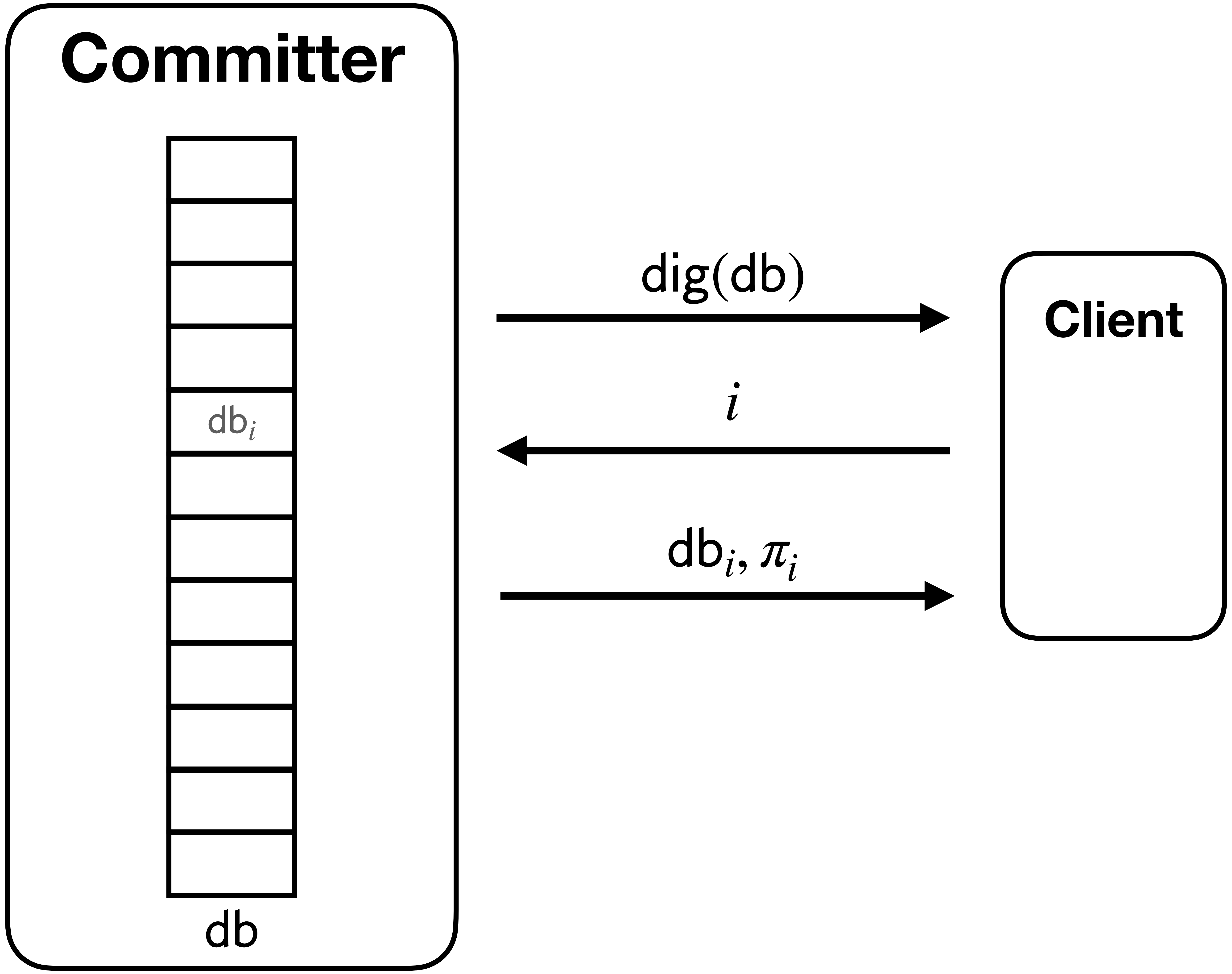
Properties:



Vector Commitments (VC)

Properties:

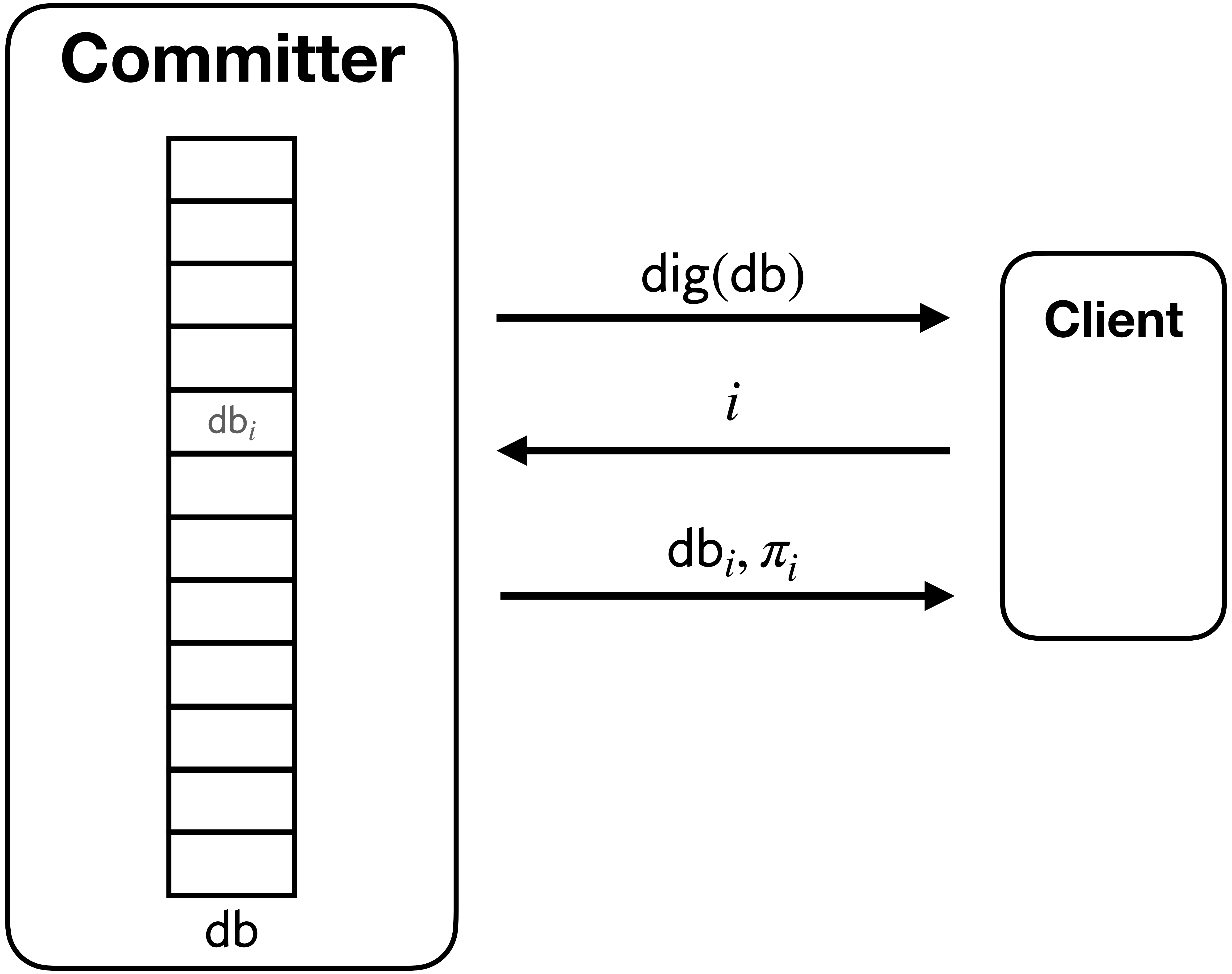
1. Completeness: honest committer convinces



Vector Commitments (VC)

Properties:

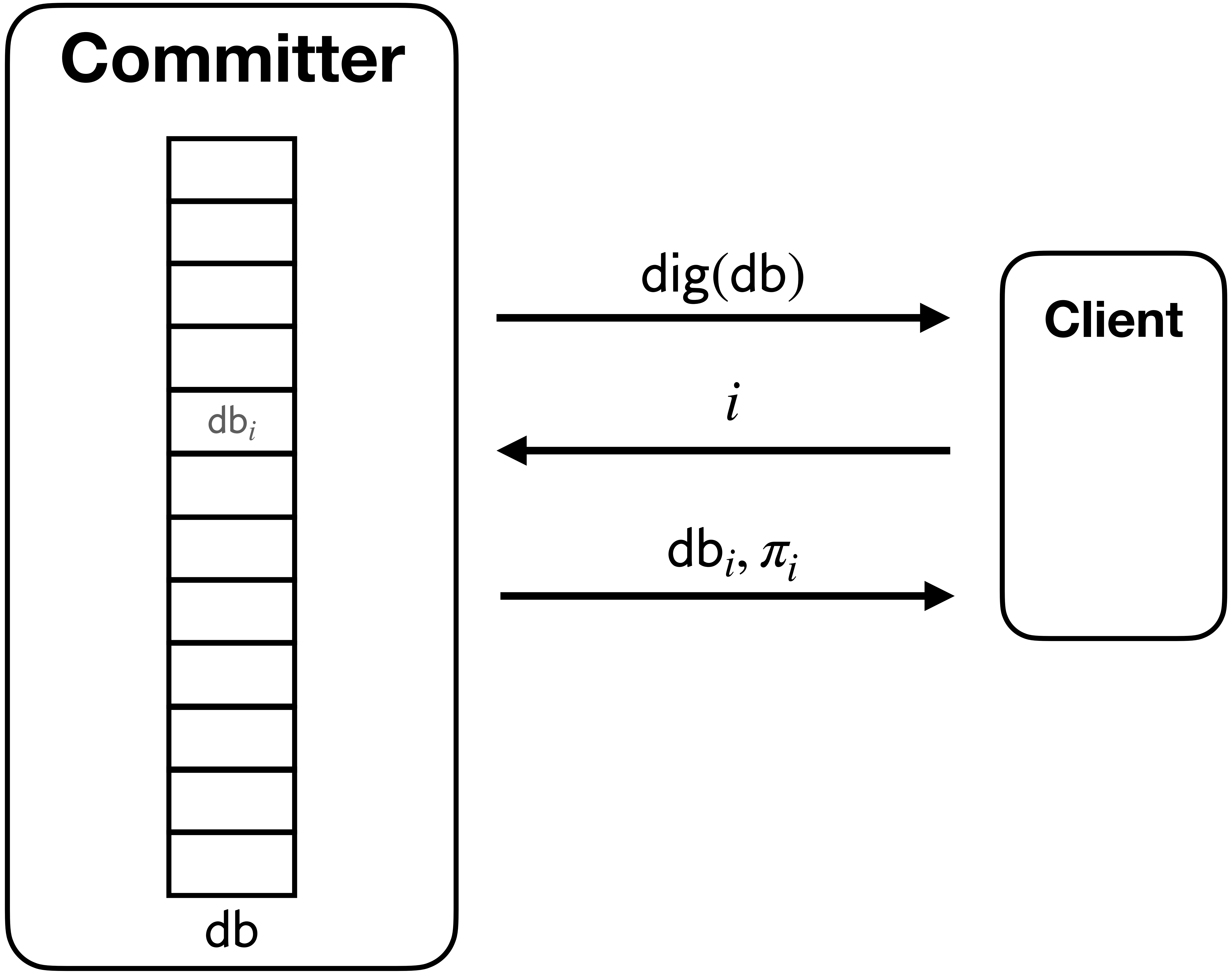
1. Completeness: honest committer convinces
2. Soundness: cannot provide different openings for i



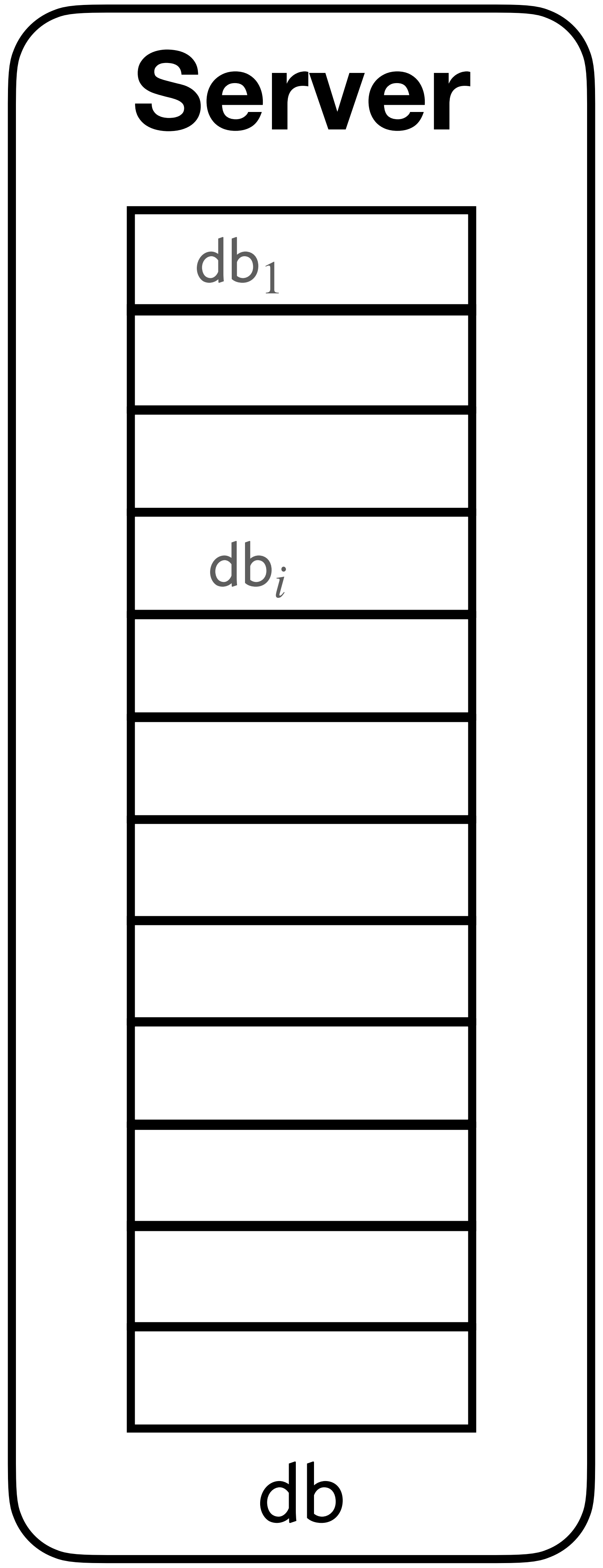
Vector Commitments (VC)

Properties:

1. Completeness: honest committer convinces
2. Soundness: cannot provide different openings for i
3. Efficiency: small $\mathrm{dig}(\mathrm{db})$, π_i

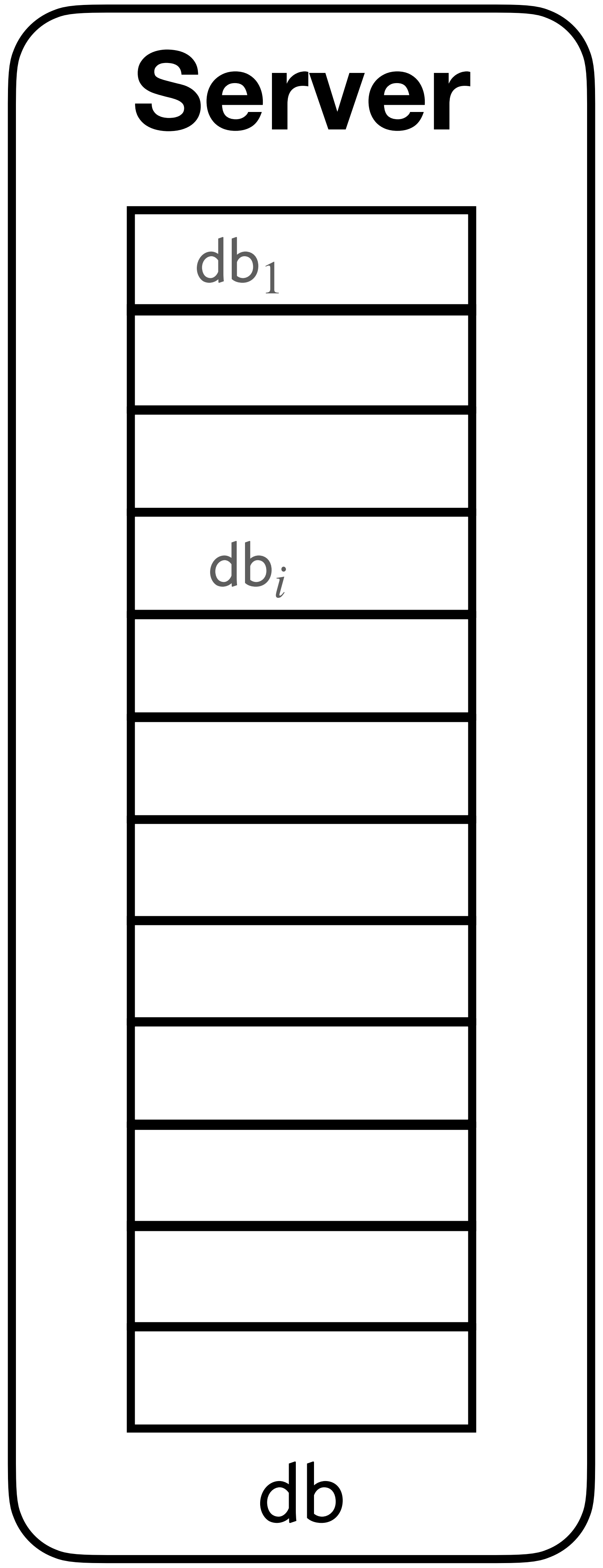


Attempt 1: VC + PIR



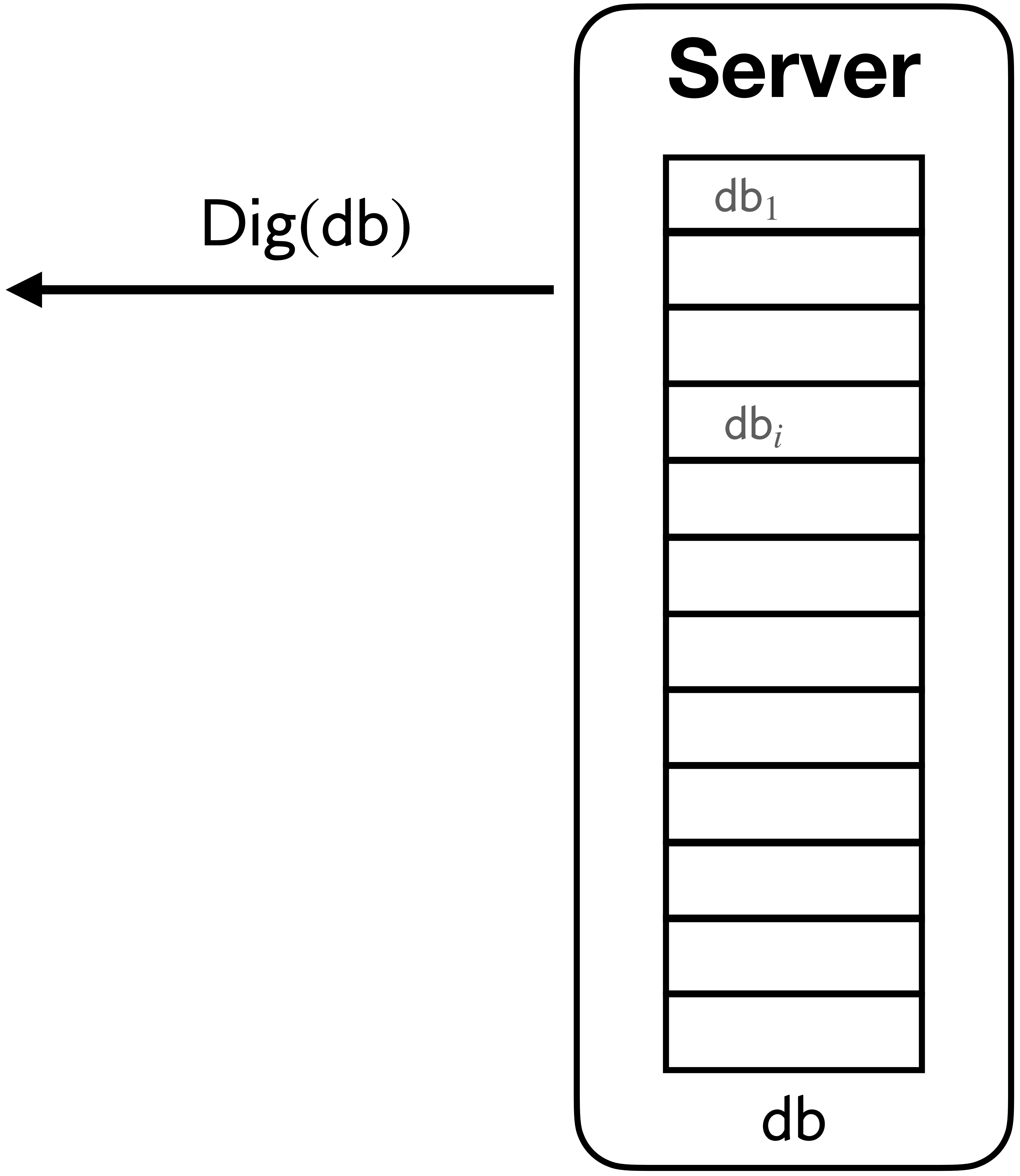
Attempt 1: VC + PIR

Offline

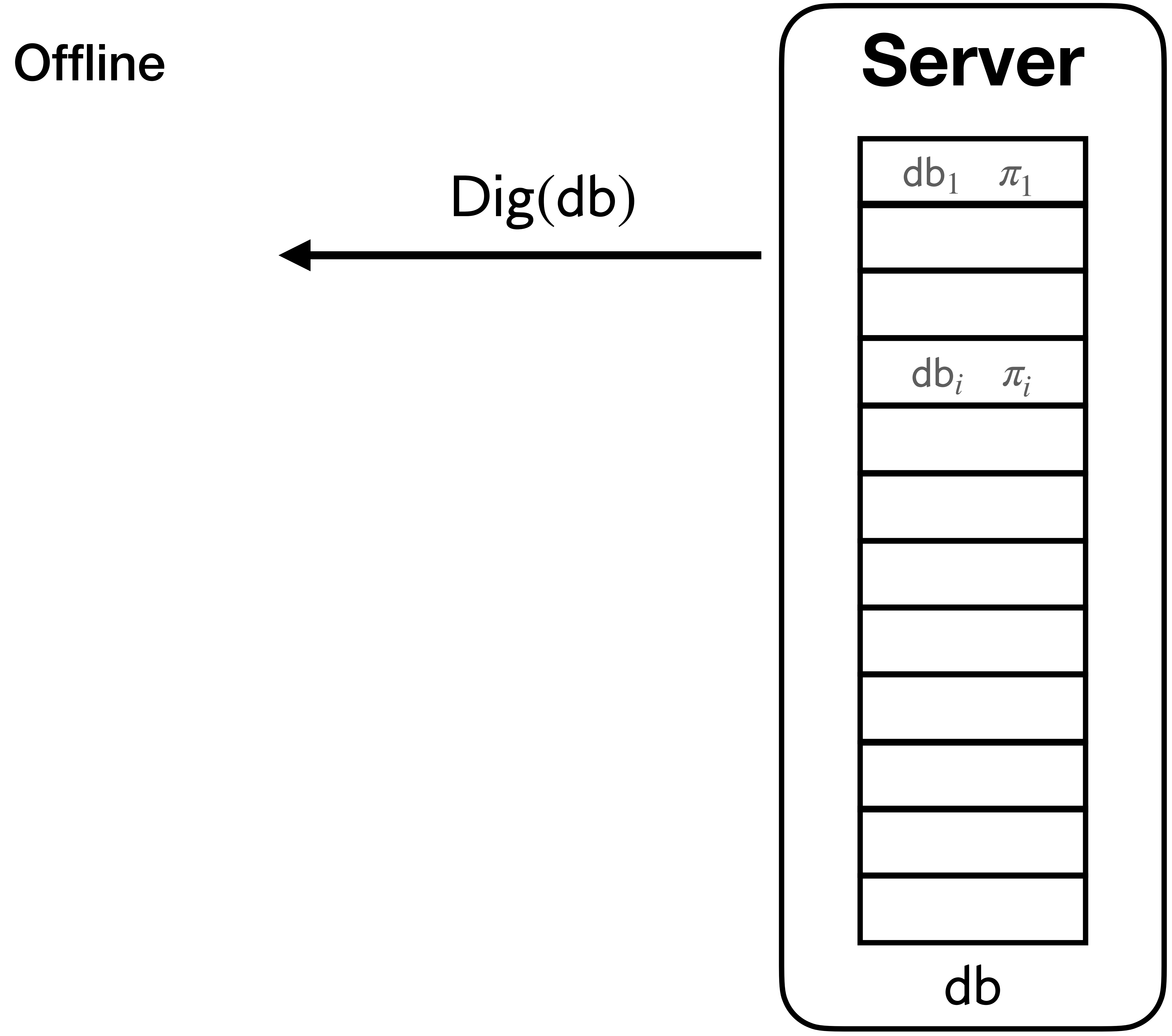


Attempt 1: VC + PIR

Offline

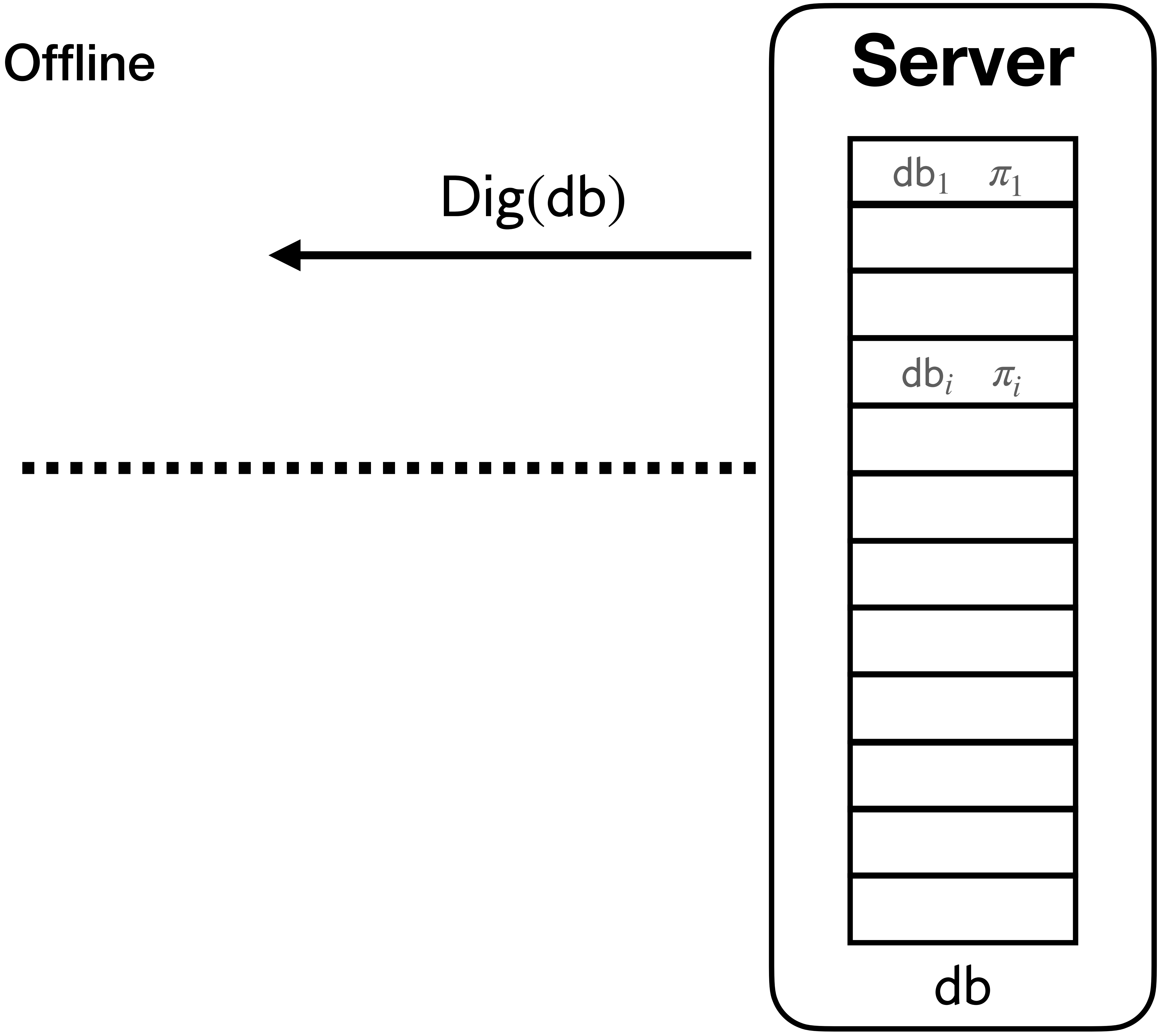


Attempt 1: VC + PIR

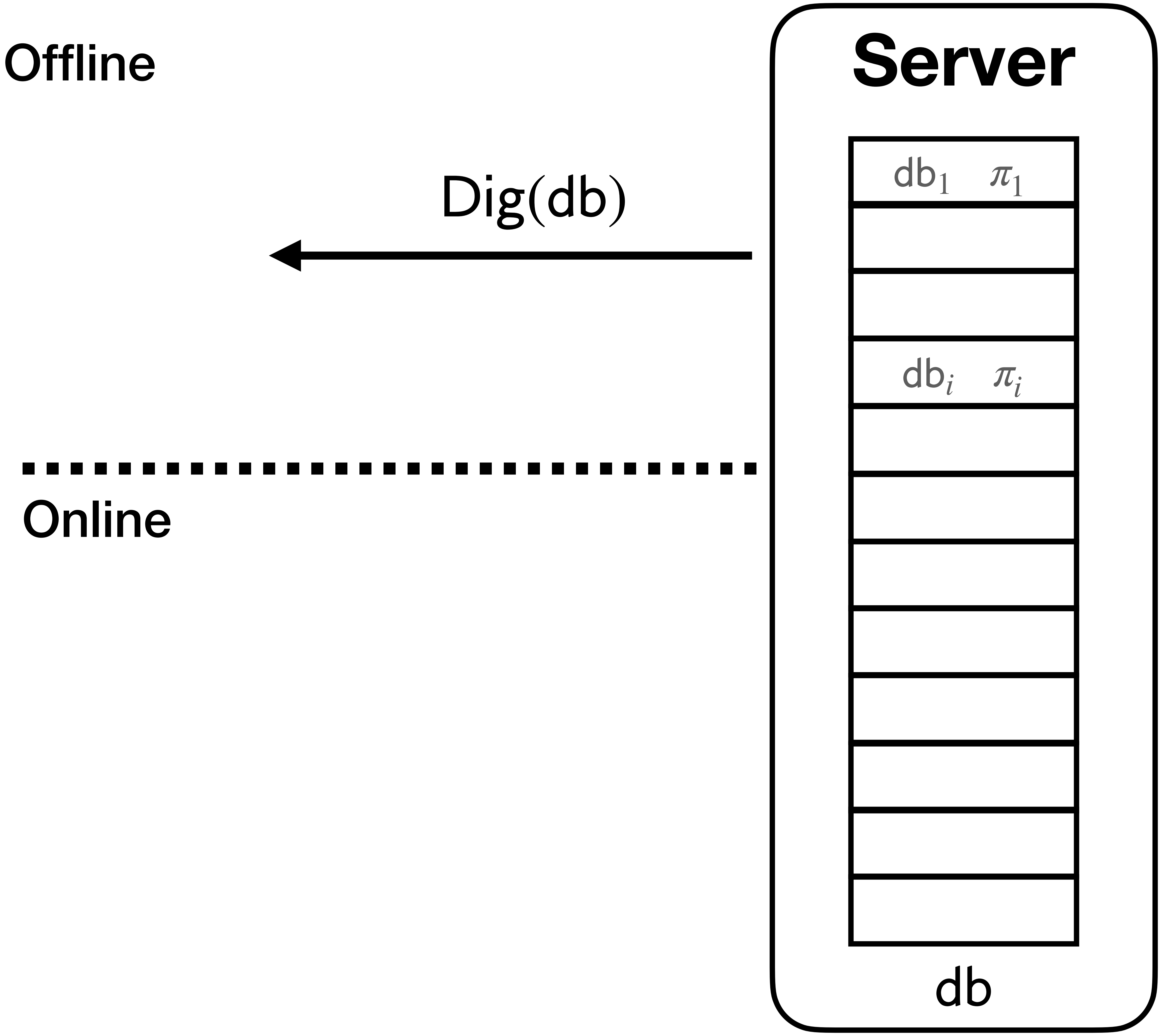


Attempt 1: VC + PIR

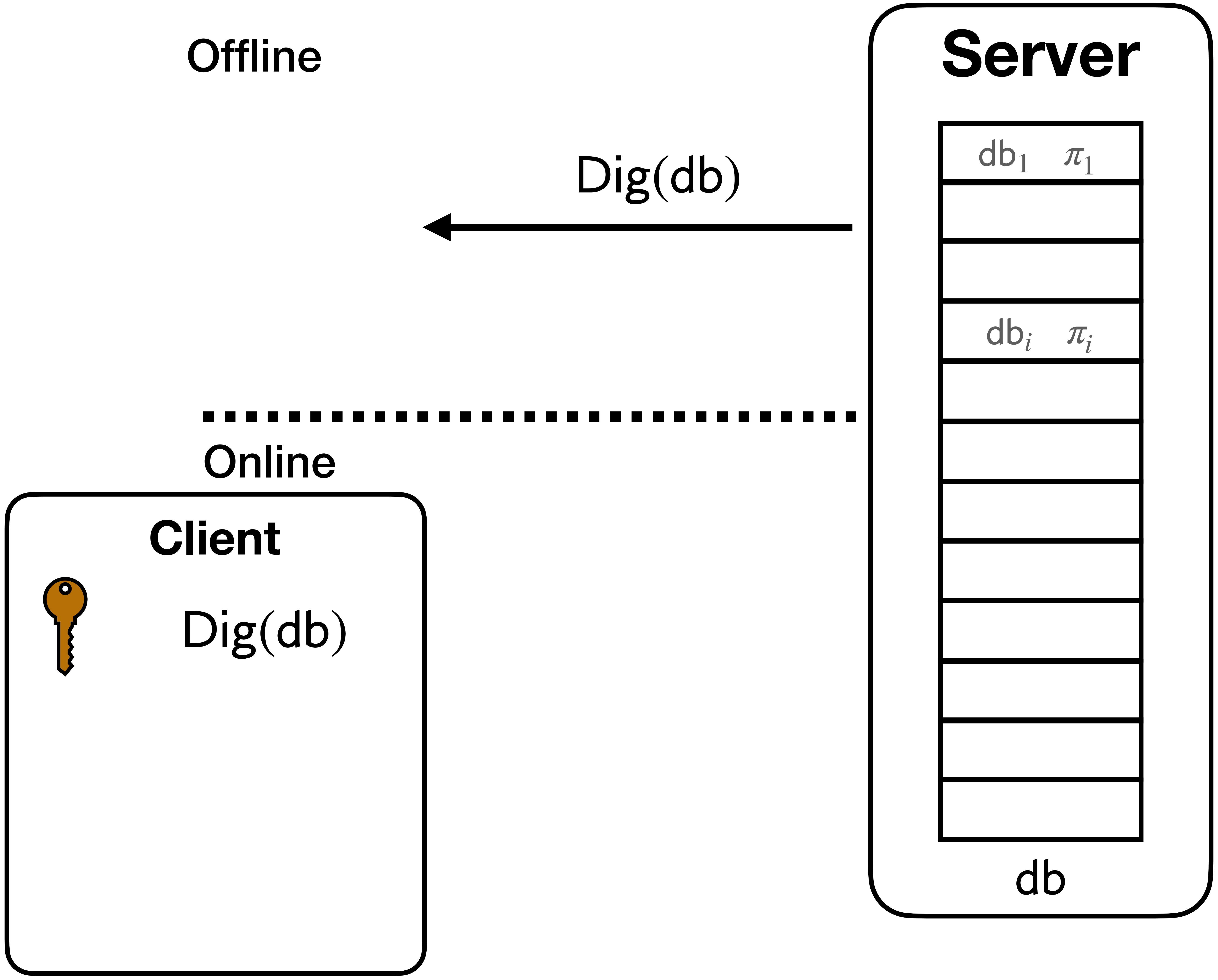
Offline



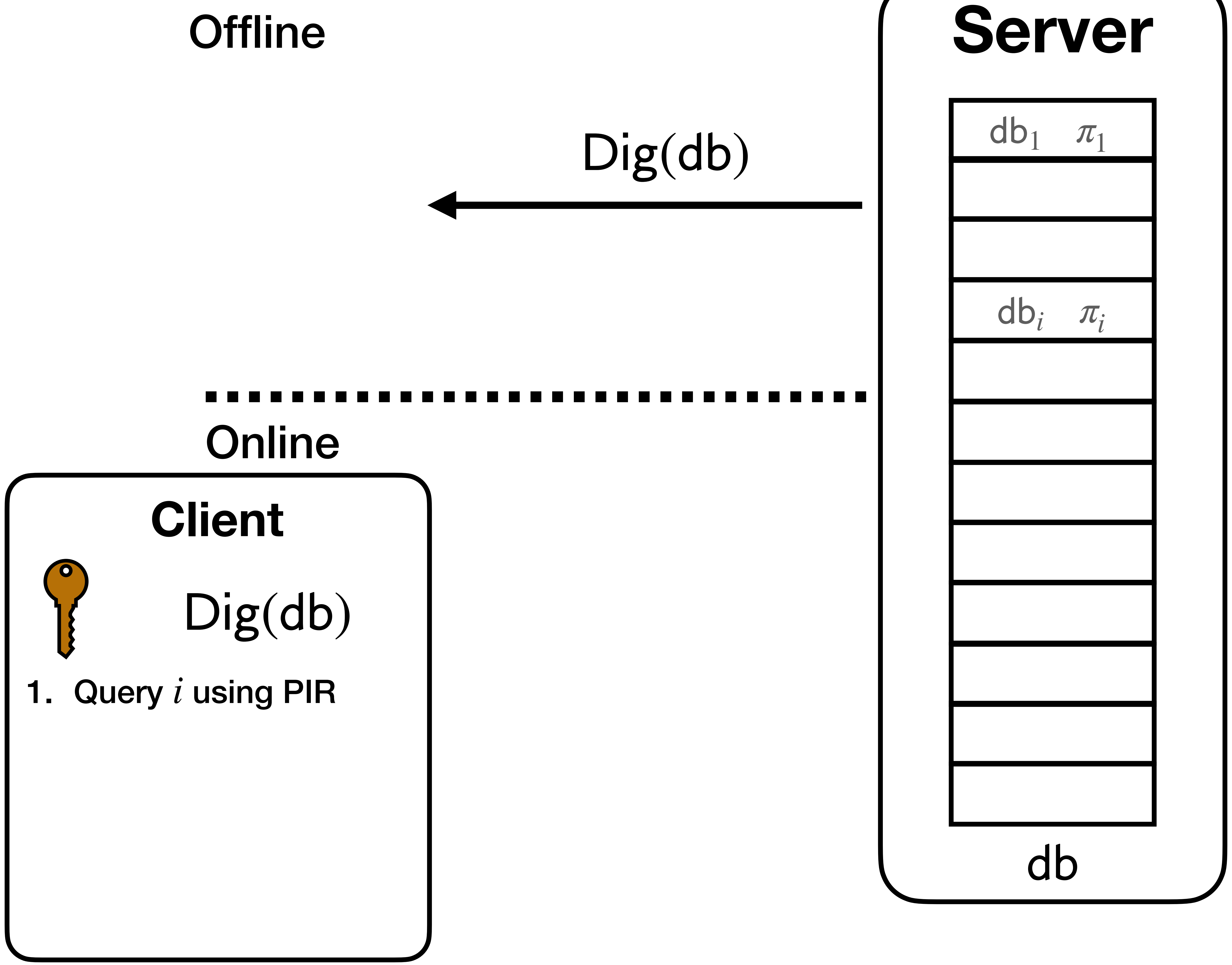
Attempt 1: VC + PIR



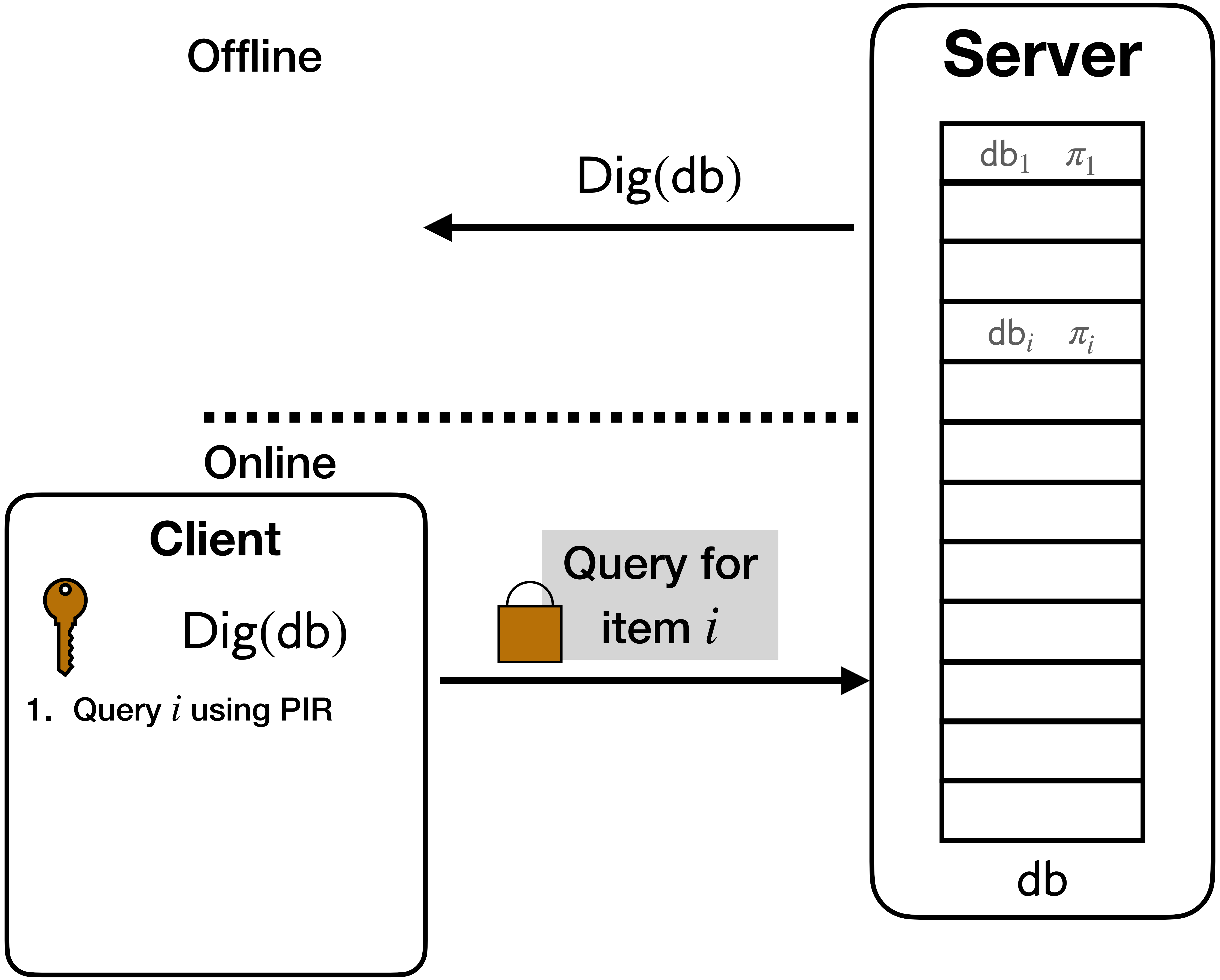
Attempt 1: VC + PIR



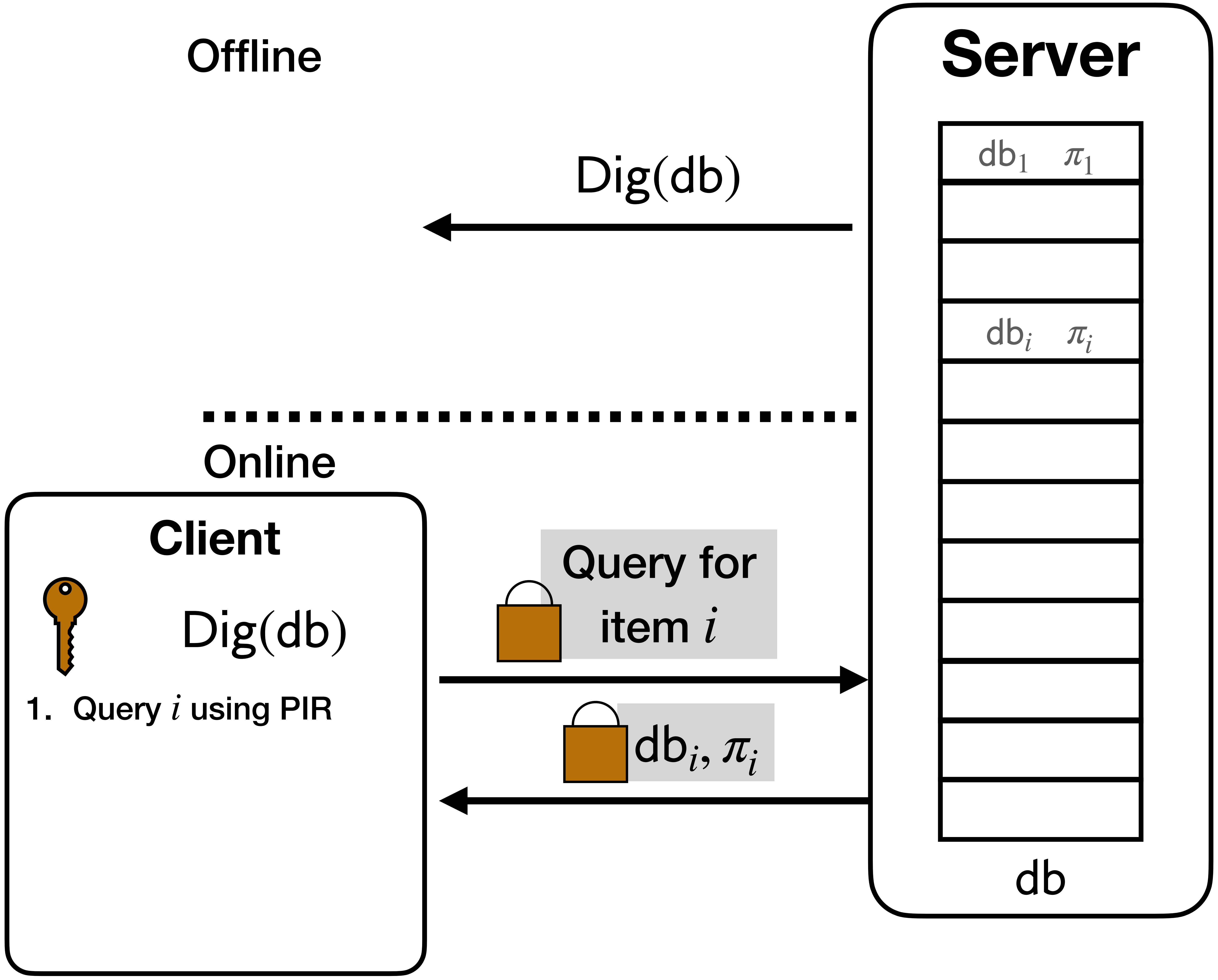
Attempt 1: VC + PIR



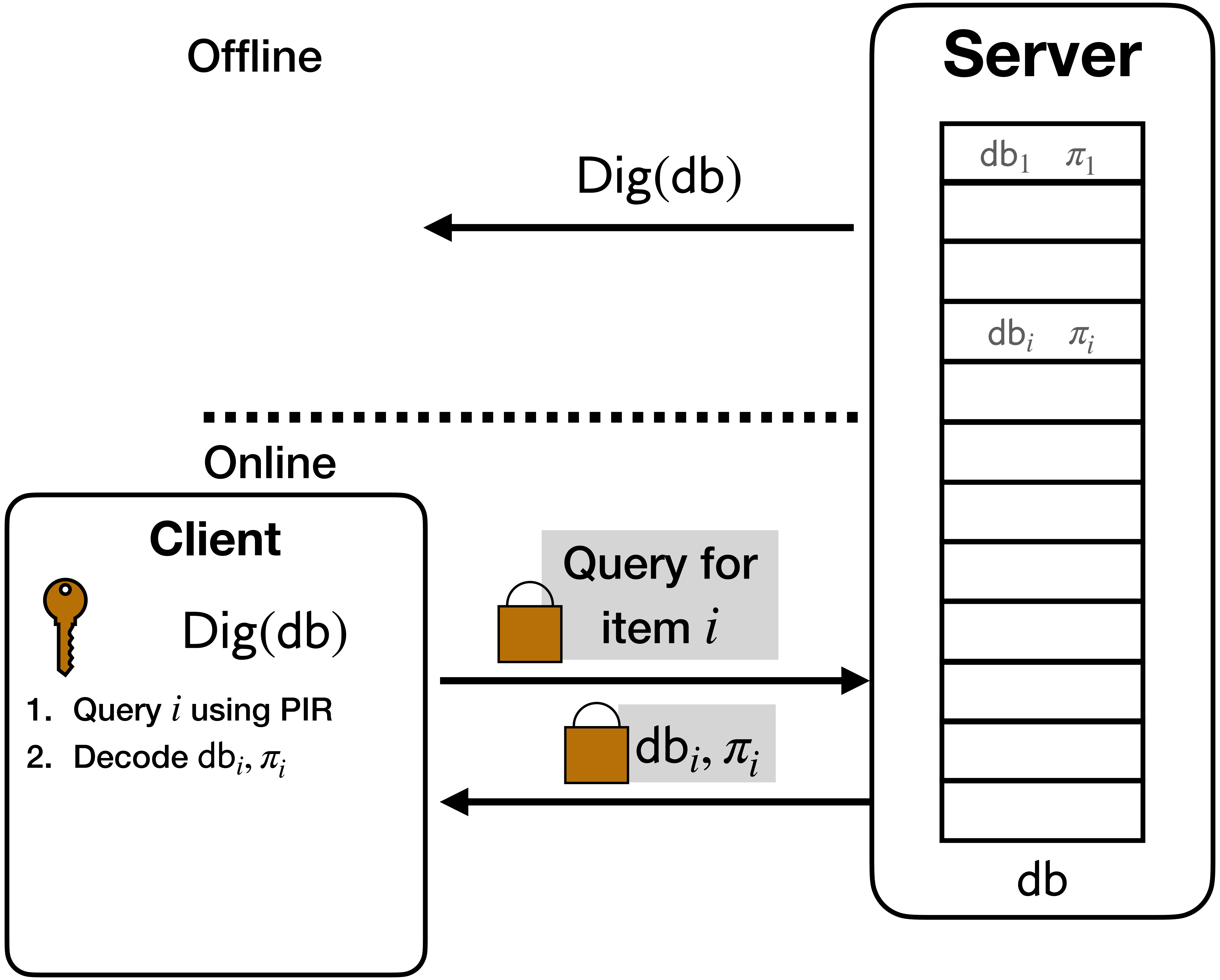
Attempt 1: VC + PIR



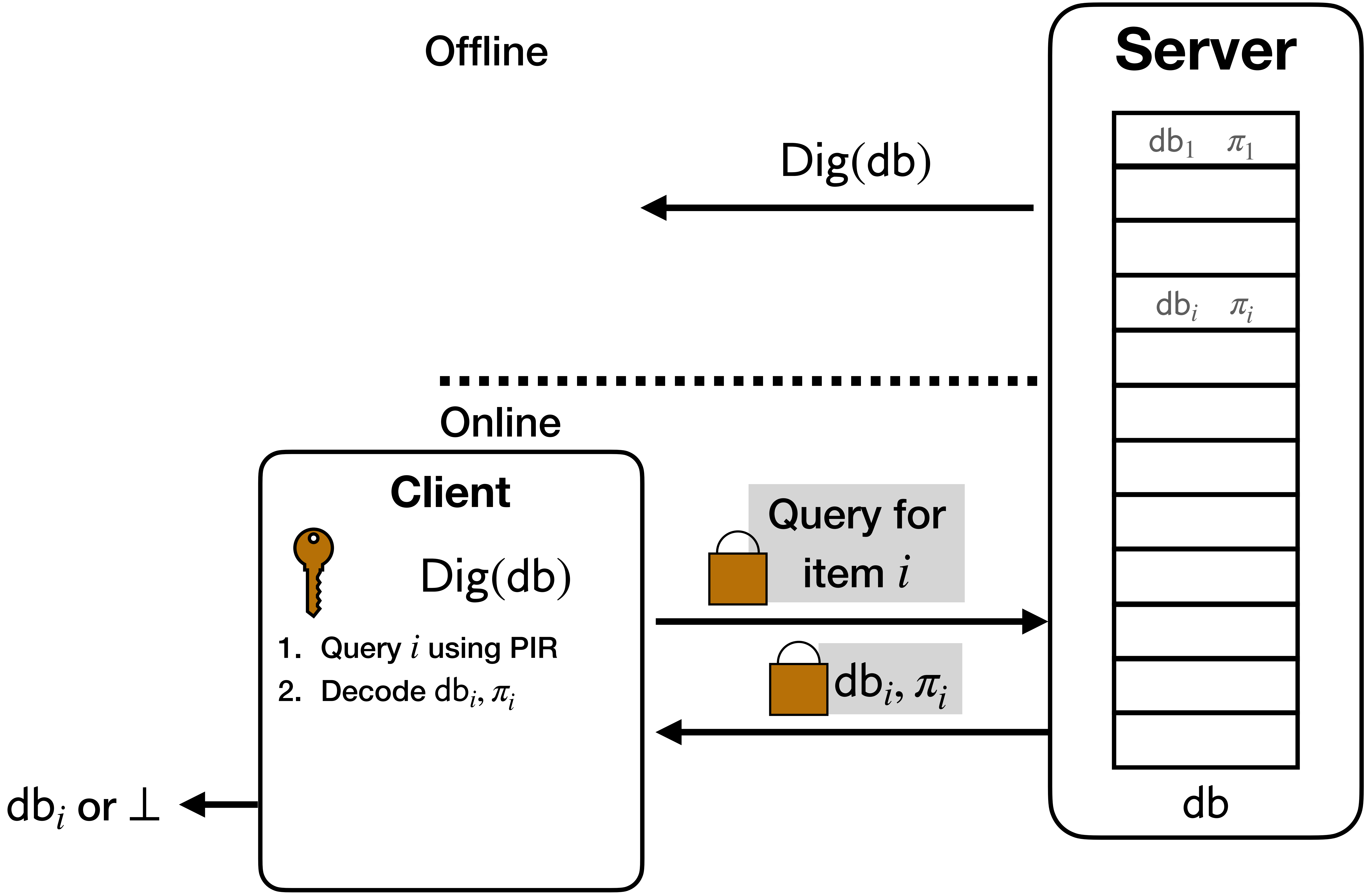
Attempt 1: VC + PIR



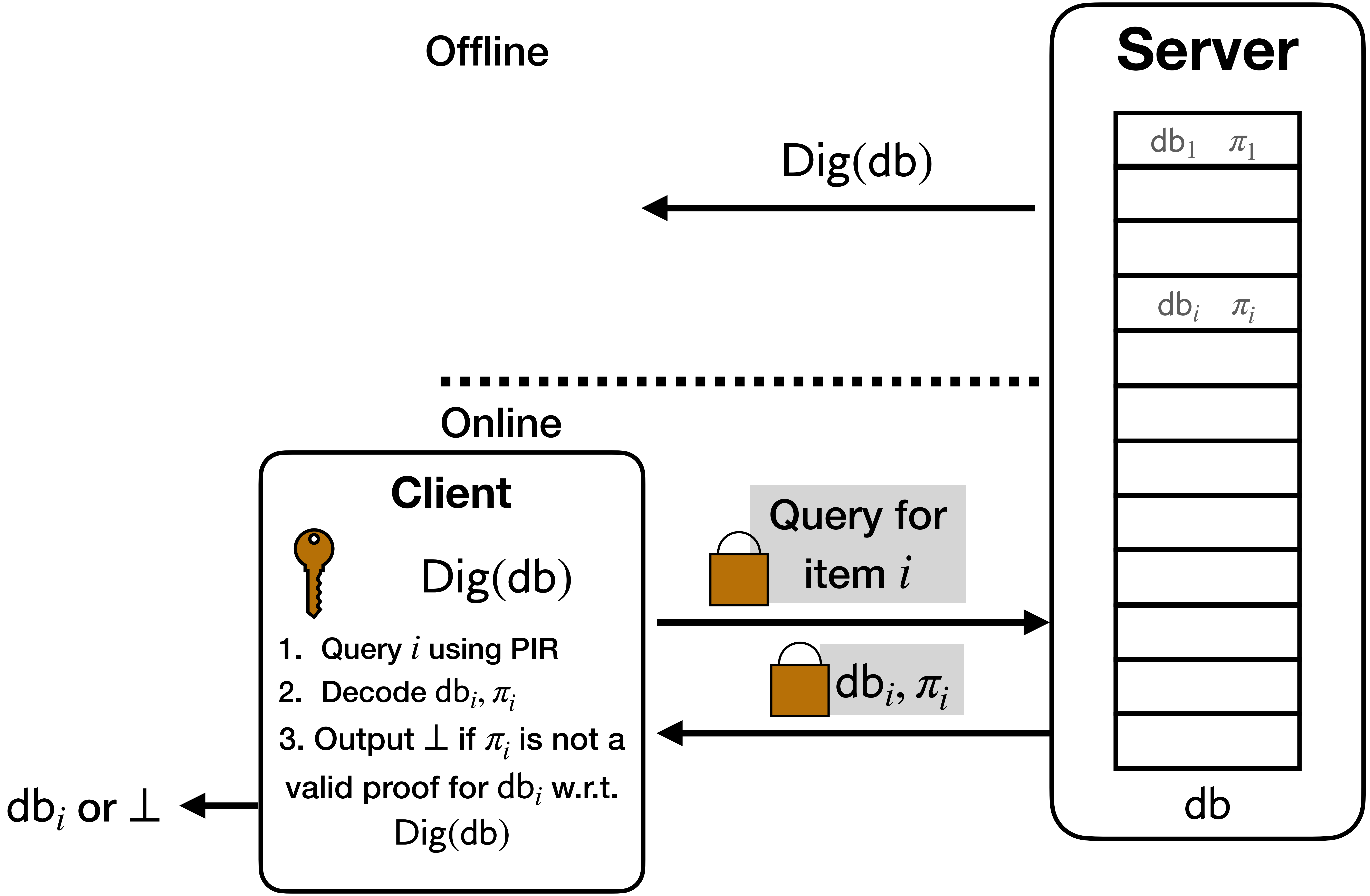
Attempt 1: VC + PIR



Attempt 1: VC + PIR

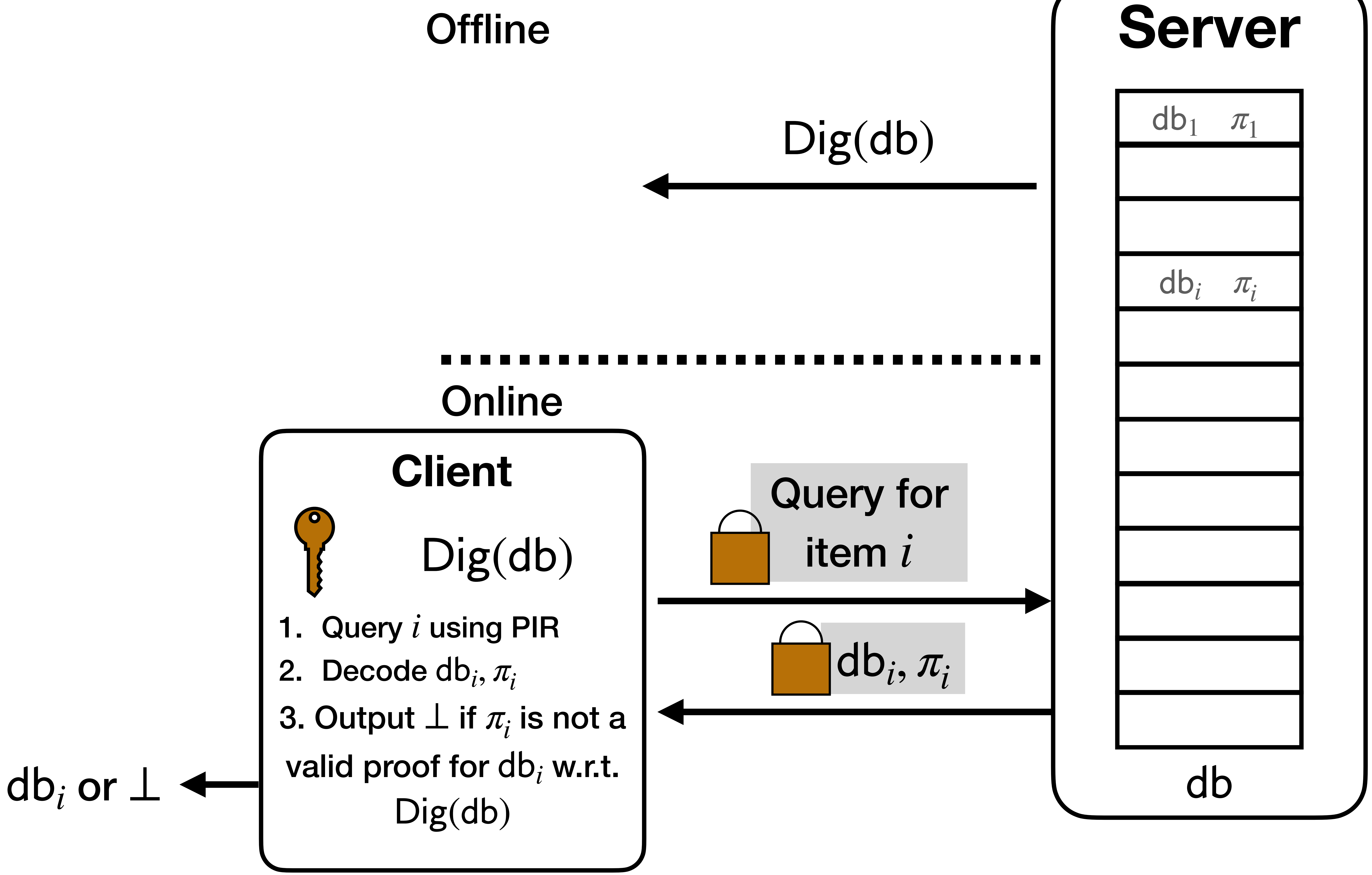


Attempt 1: VC + PIR

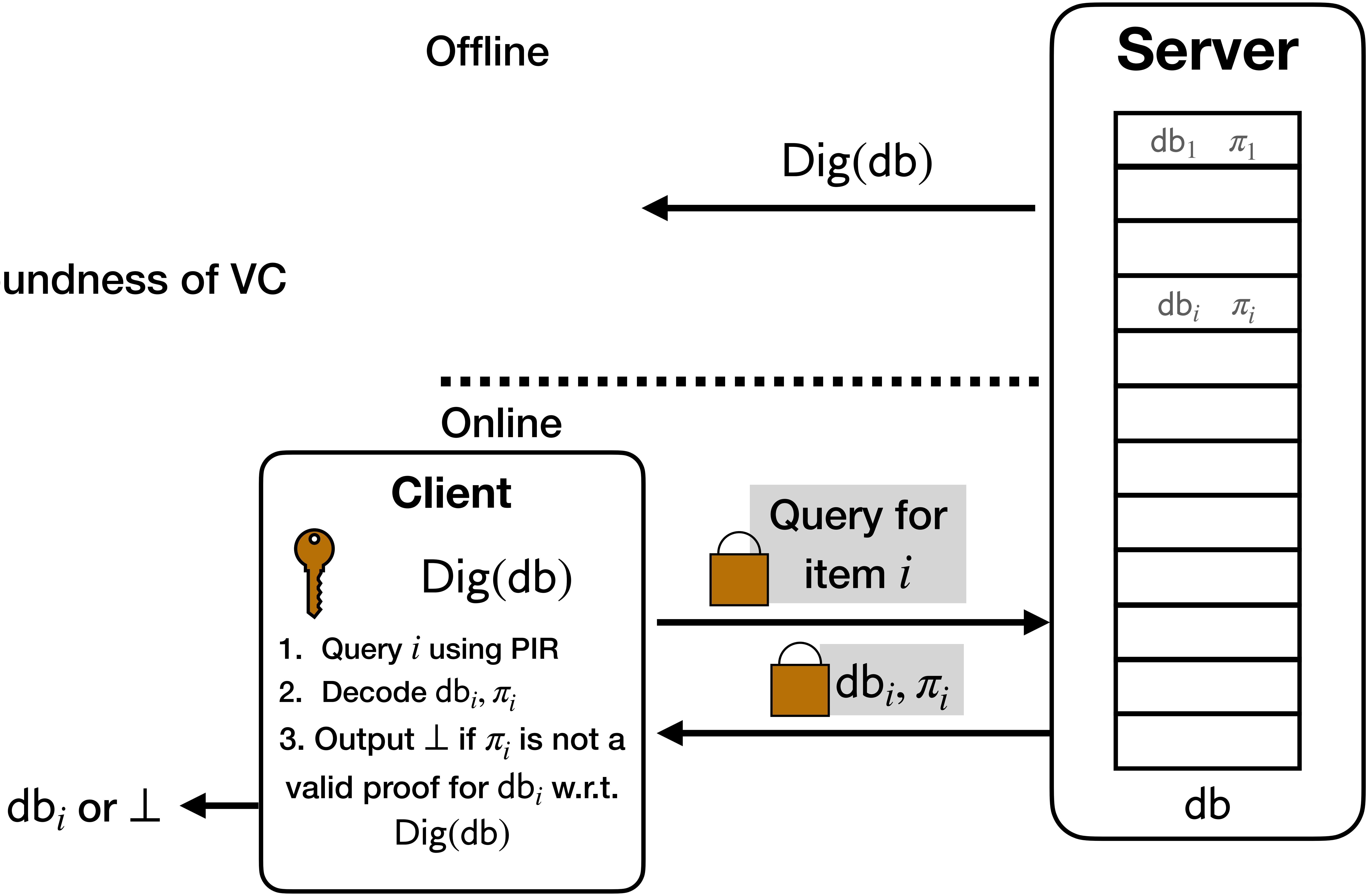


Attempt 1: VC + PIR

Properties:



Attempt 1: VC + PIR



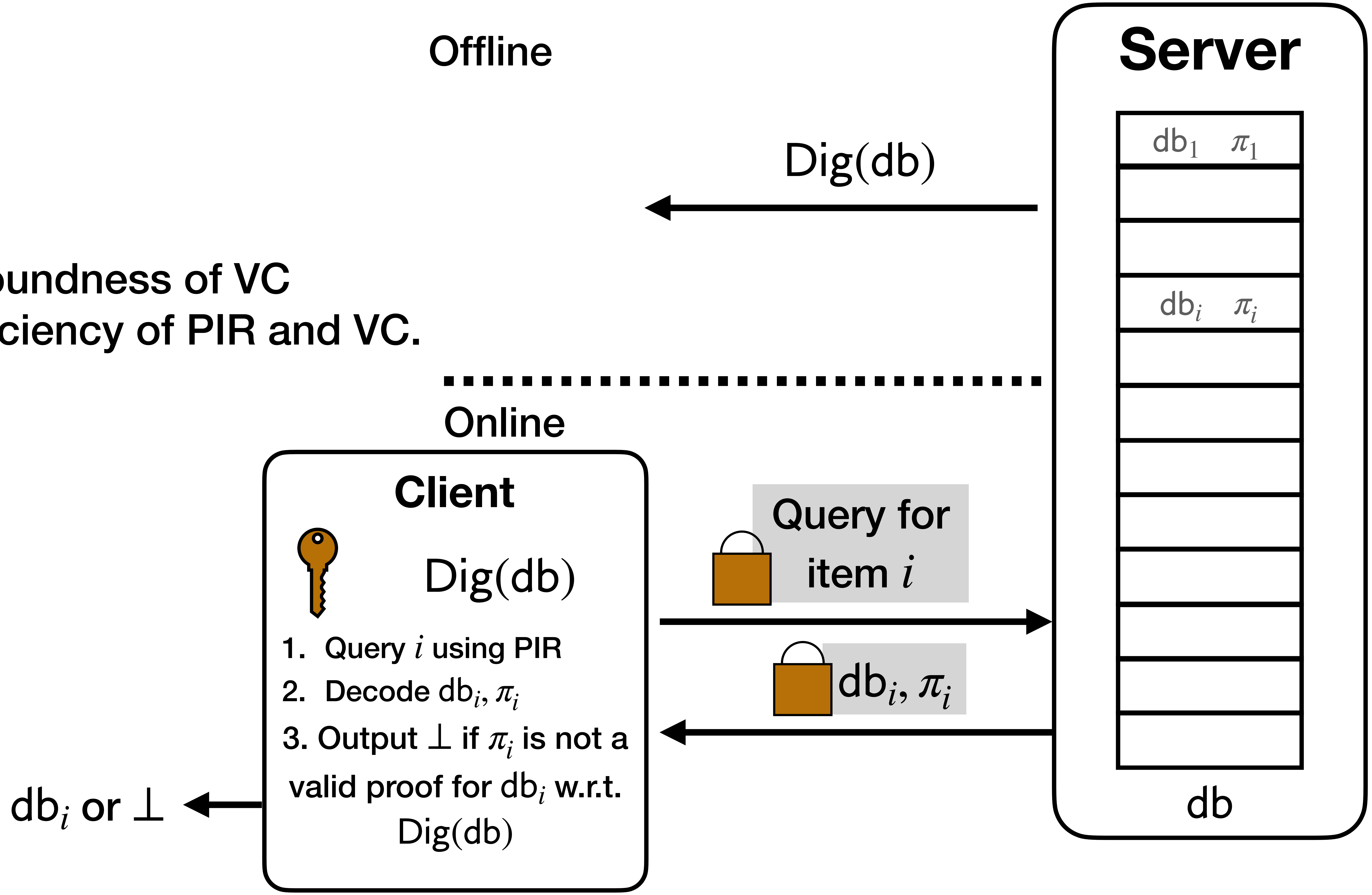
Properties:

1. Coherence: by soundness of VC

Attempt 1: VC + PIR

Properties:

1. Coherence: by soundness of VC
2. Efficiency: by efficiency of PIR and VC.

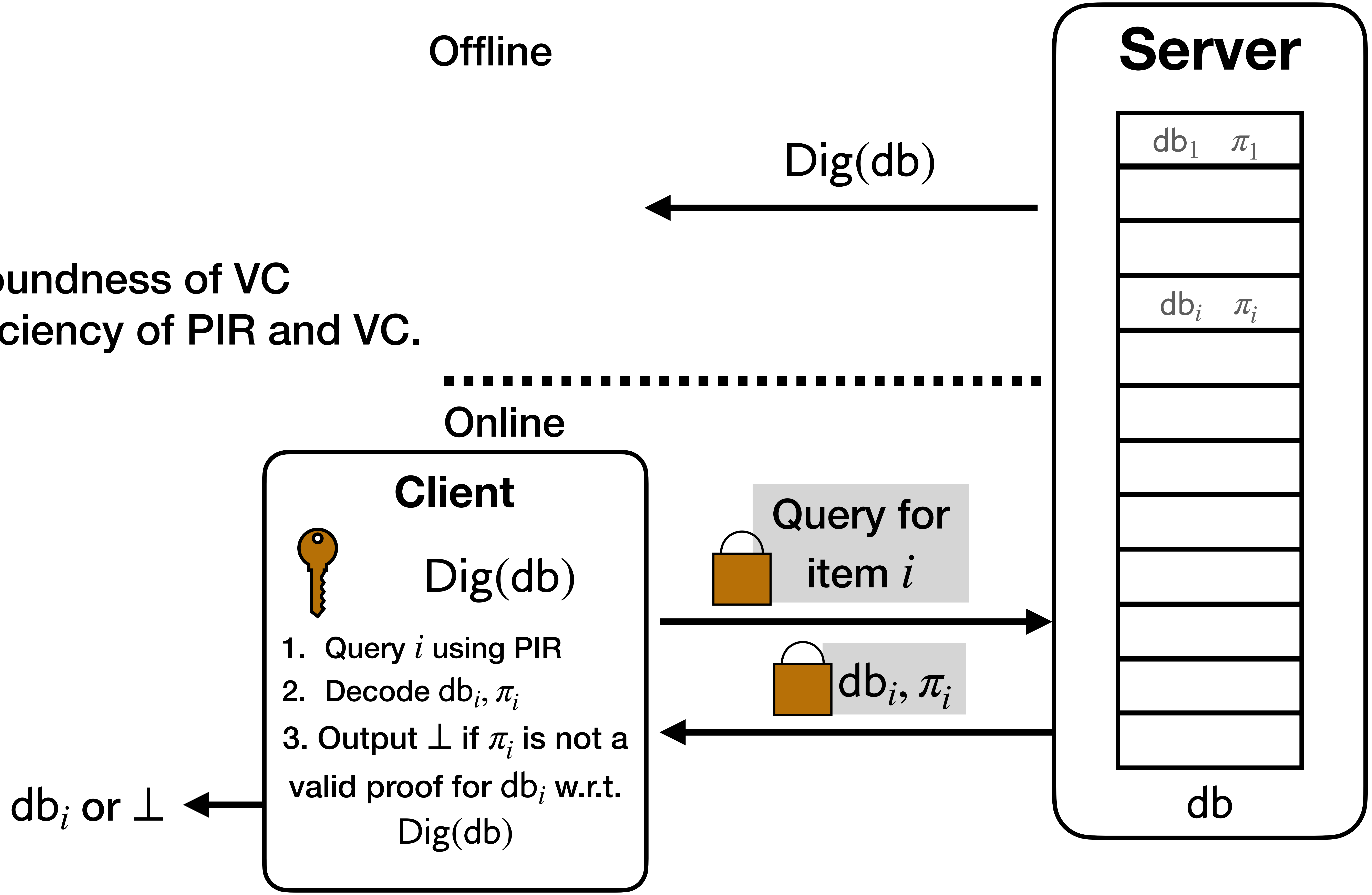


Attempt 1: VC + PIR

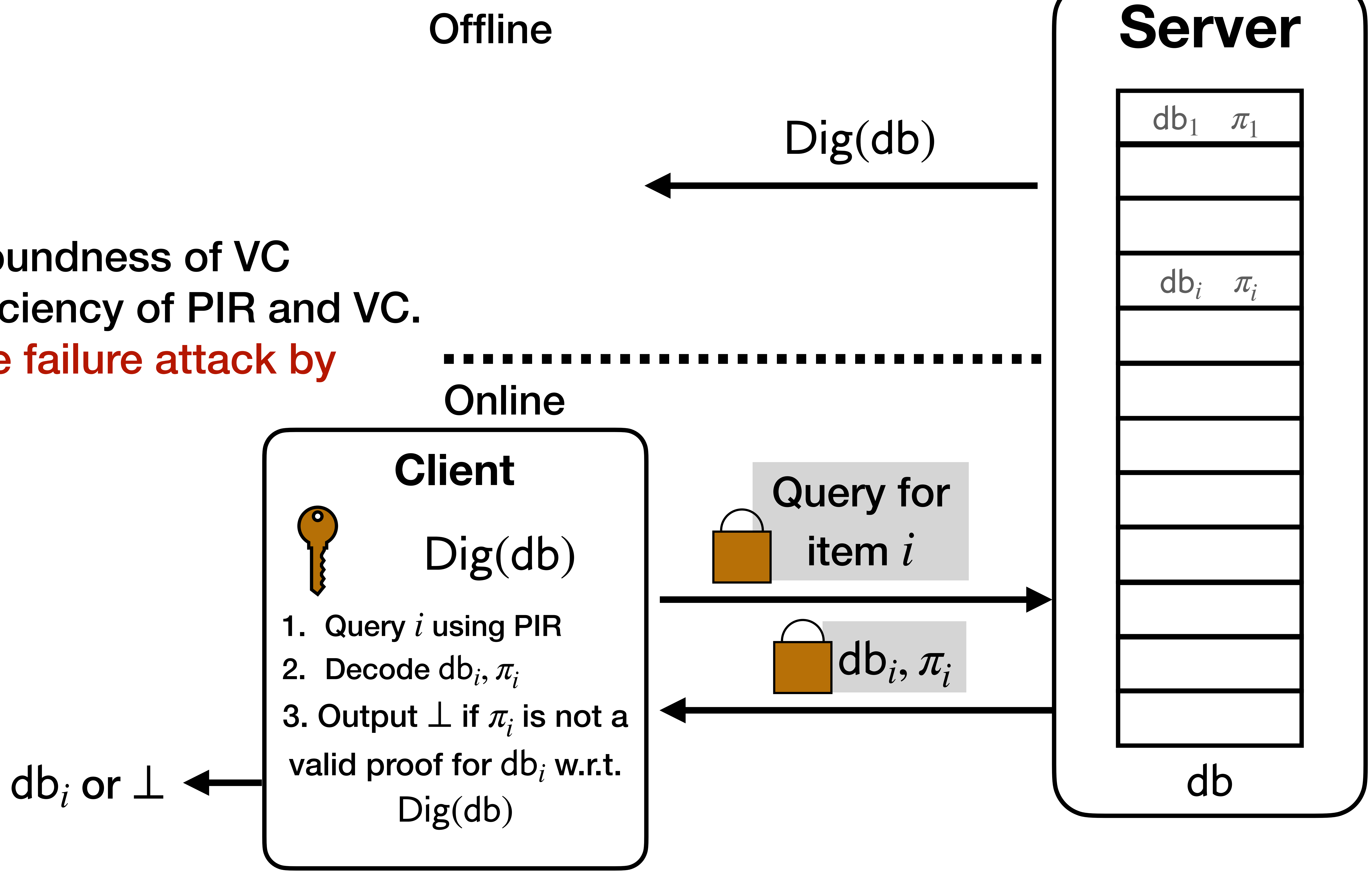
Properties:

1. Coherence: by soundness of VC
2. Efficiency: by efficiency of PIR and VC.

Privacy?



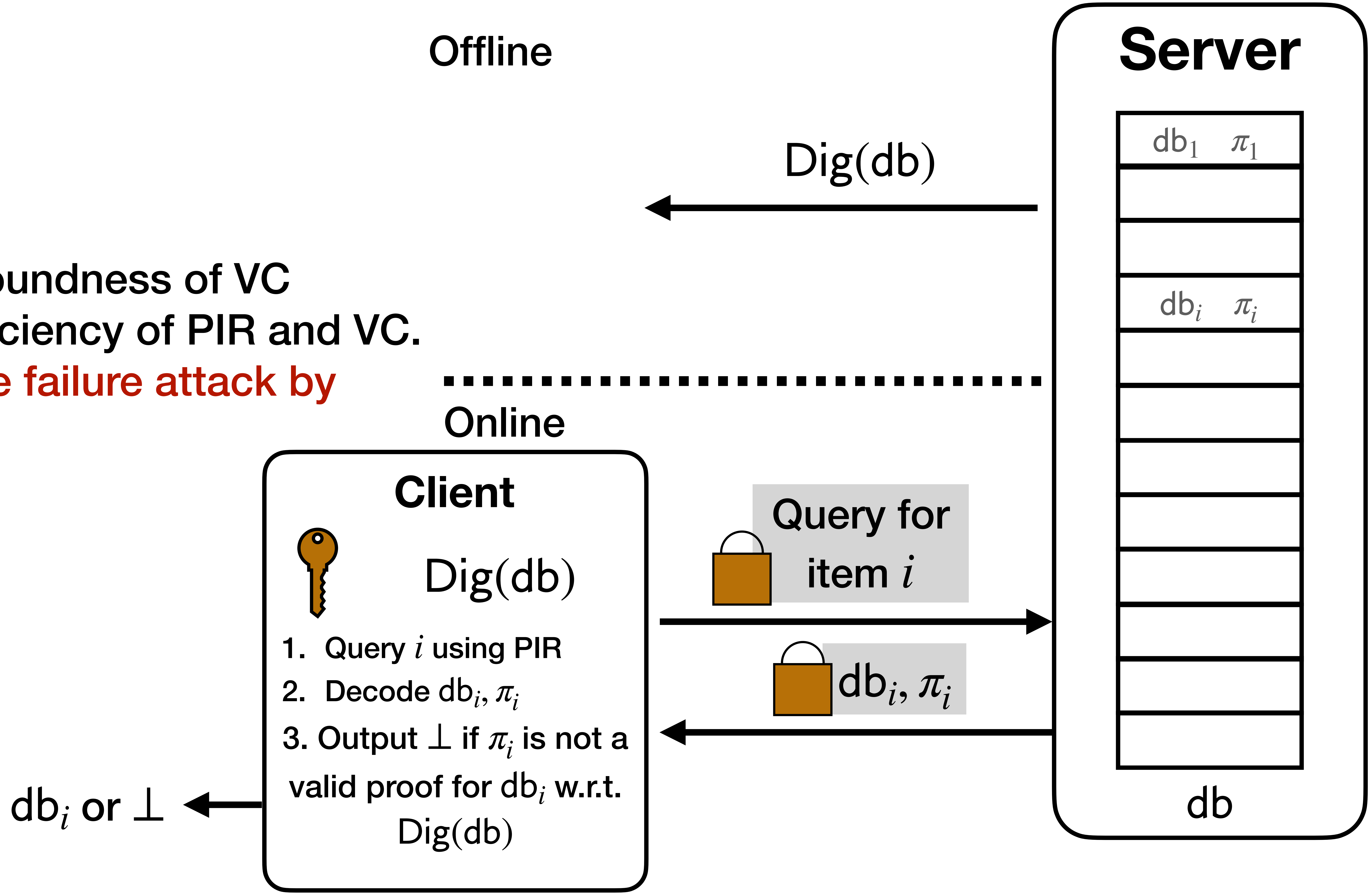
Attempt 1: VC + PIR



Properties:

1. Coherence: by soundness of VC
2. Efficiency: by efficiency of PIR and VC.
3. Privacy? selective failure attack by corrupting π_i.

Attempt 1: VC + PIR



Properties:

1. Coherence: by soundness of VC
2. Efficiency: by efficiency of PIR and VC.
3. Privacy? selective failure attack by corrupting π_i.

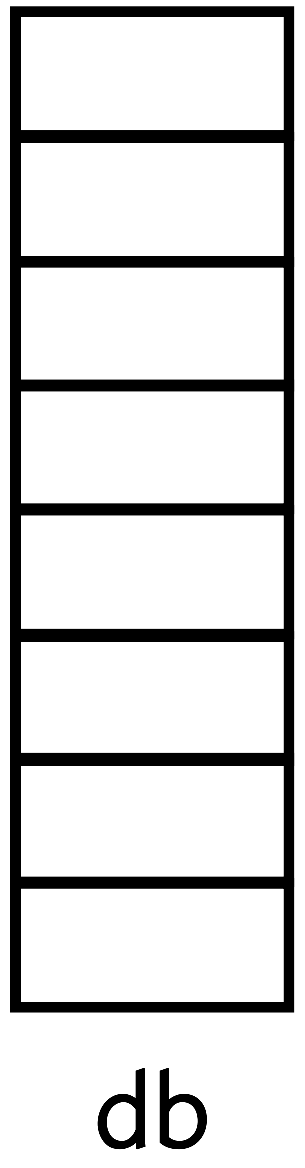
Locally decodable codes (LDC)

Locally decodable codes (LDC)

Encoding

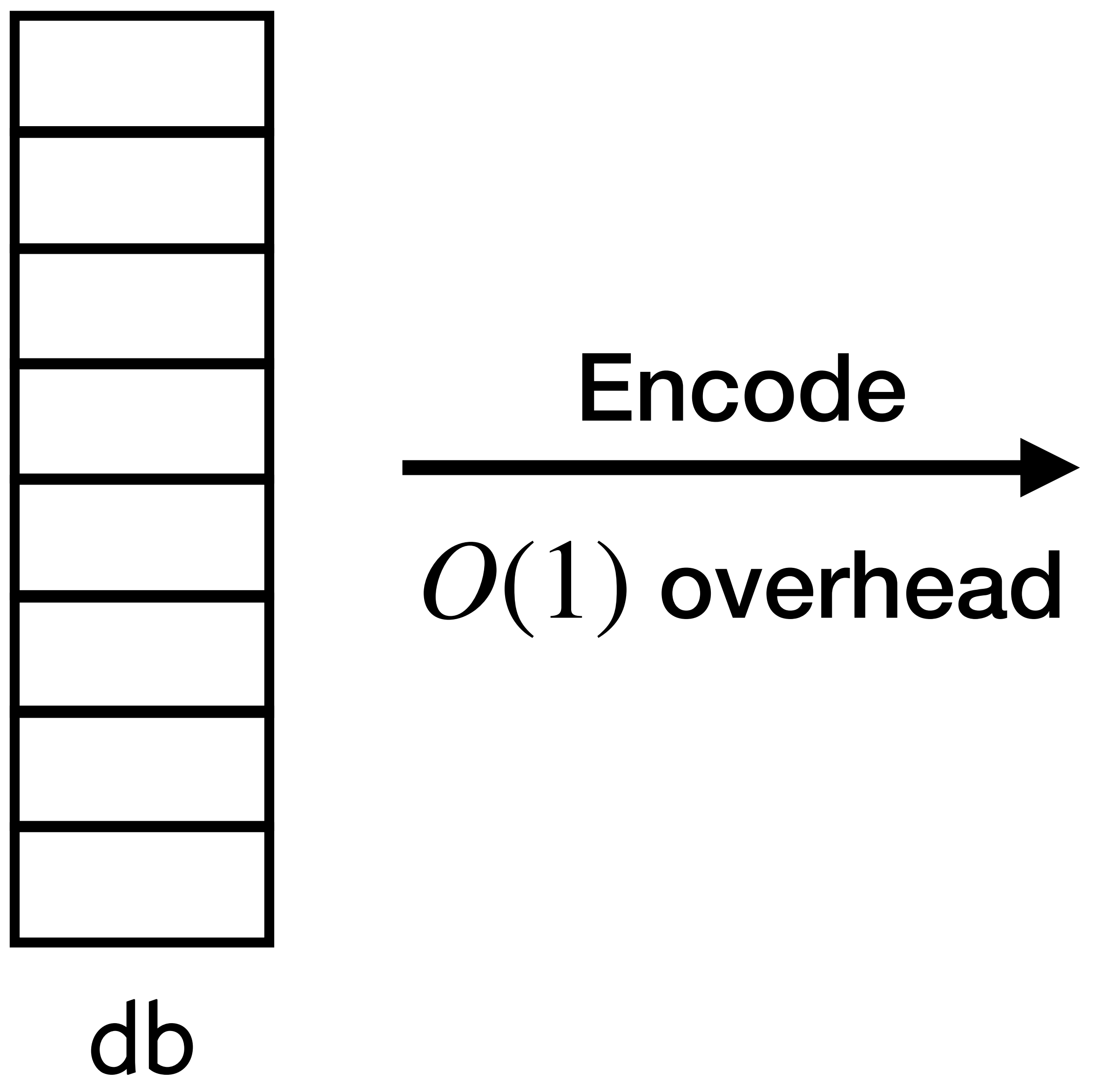
Locally decodable codes (LDC)

Encoding



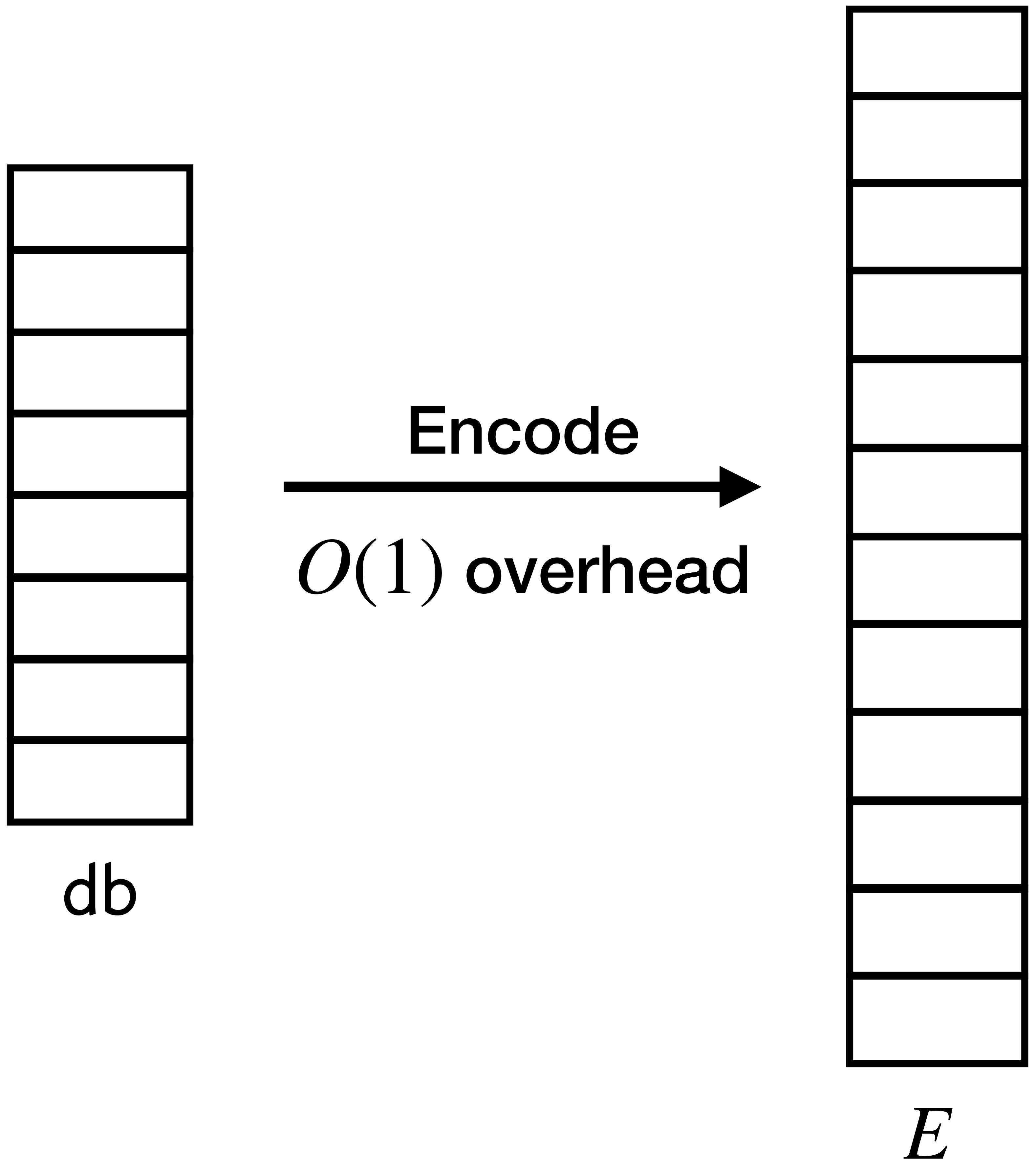
Locally decodable codes (LDC)

Encoding



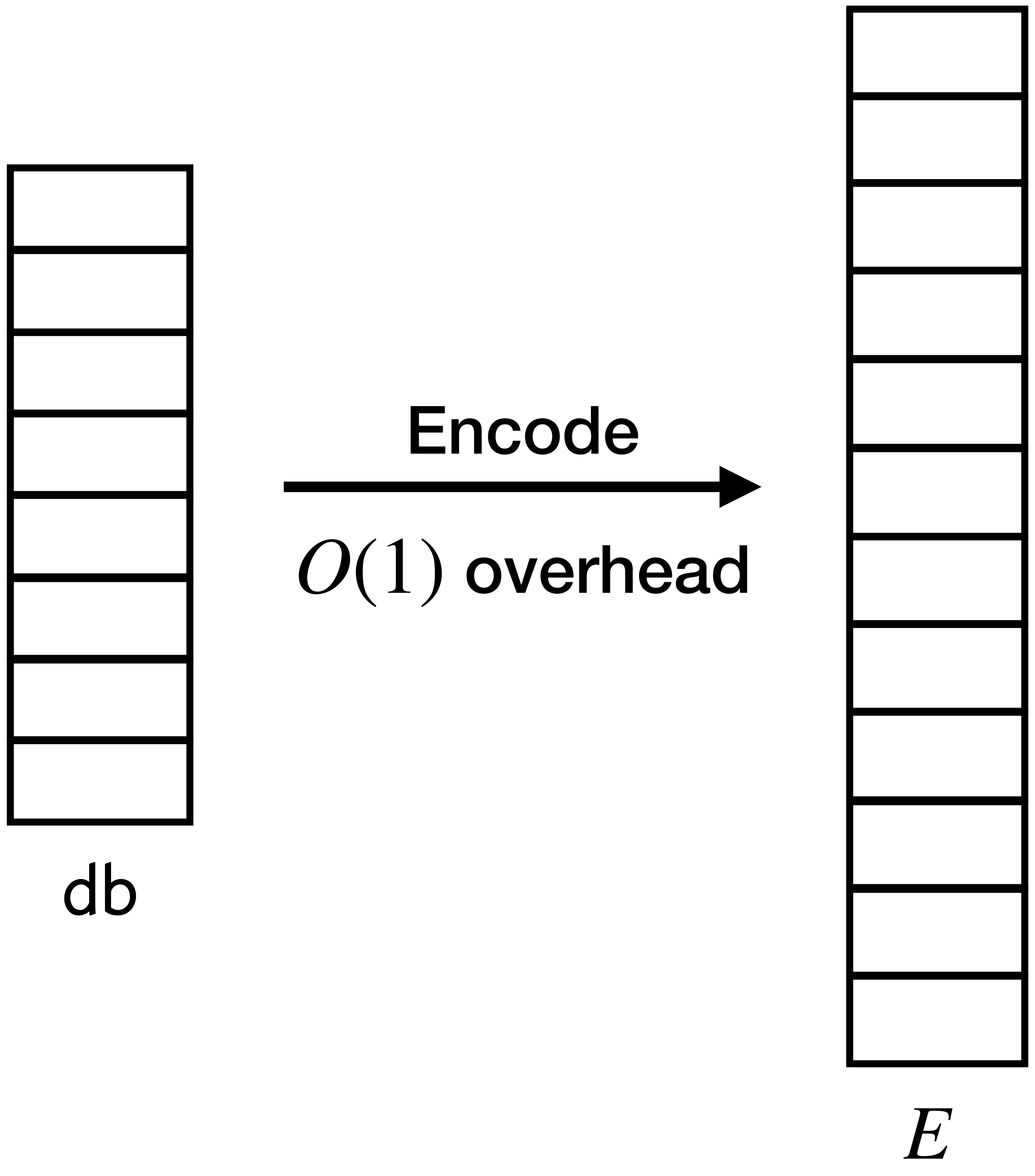
Locally decodable codes (LDC)

Encoding



Locally decodable codes (LDC)

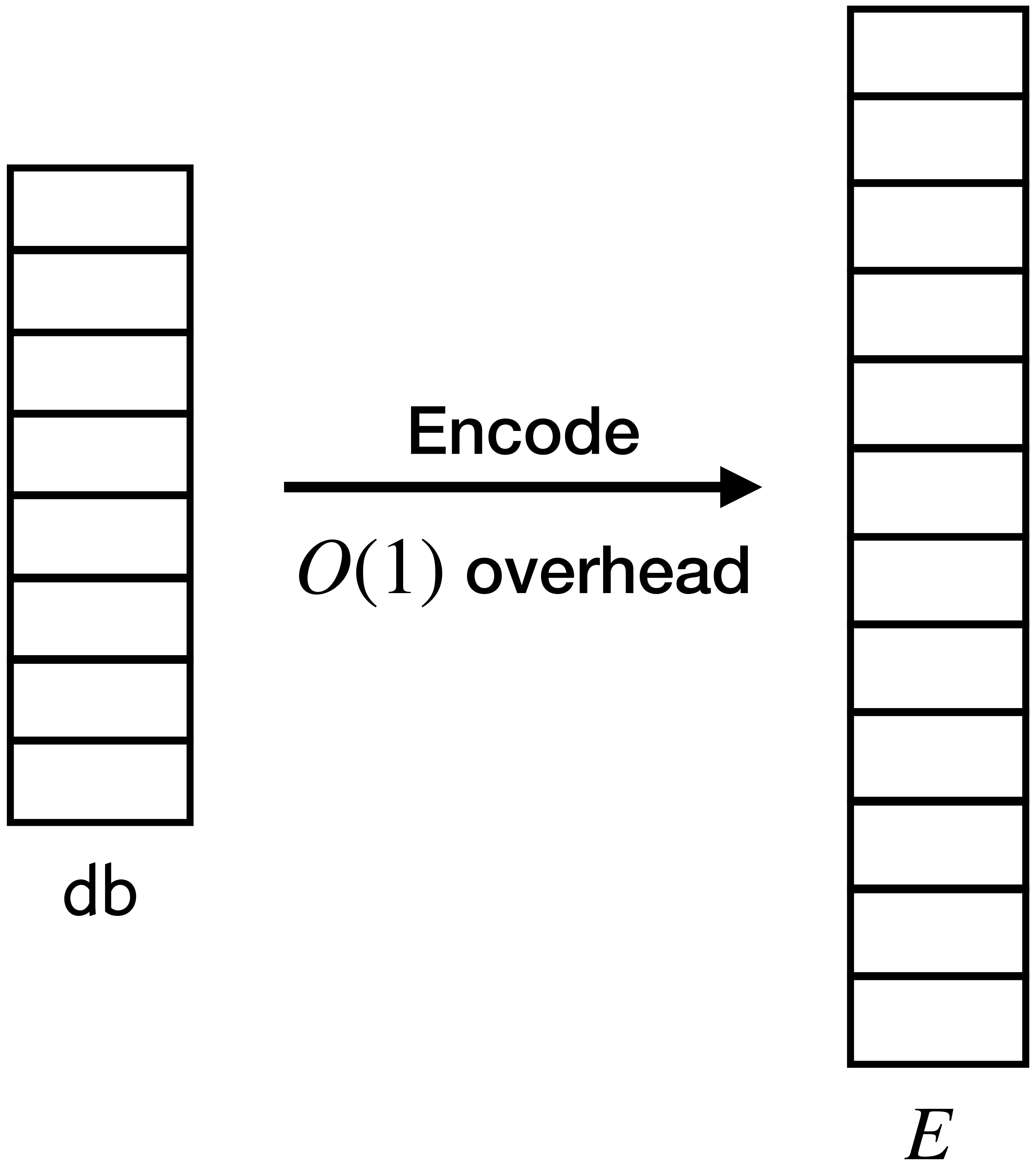
Encoding



***Local* Decoding**

Locally decodable codes (LDC)

Encoding

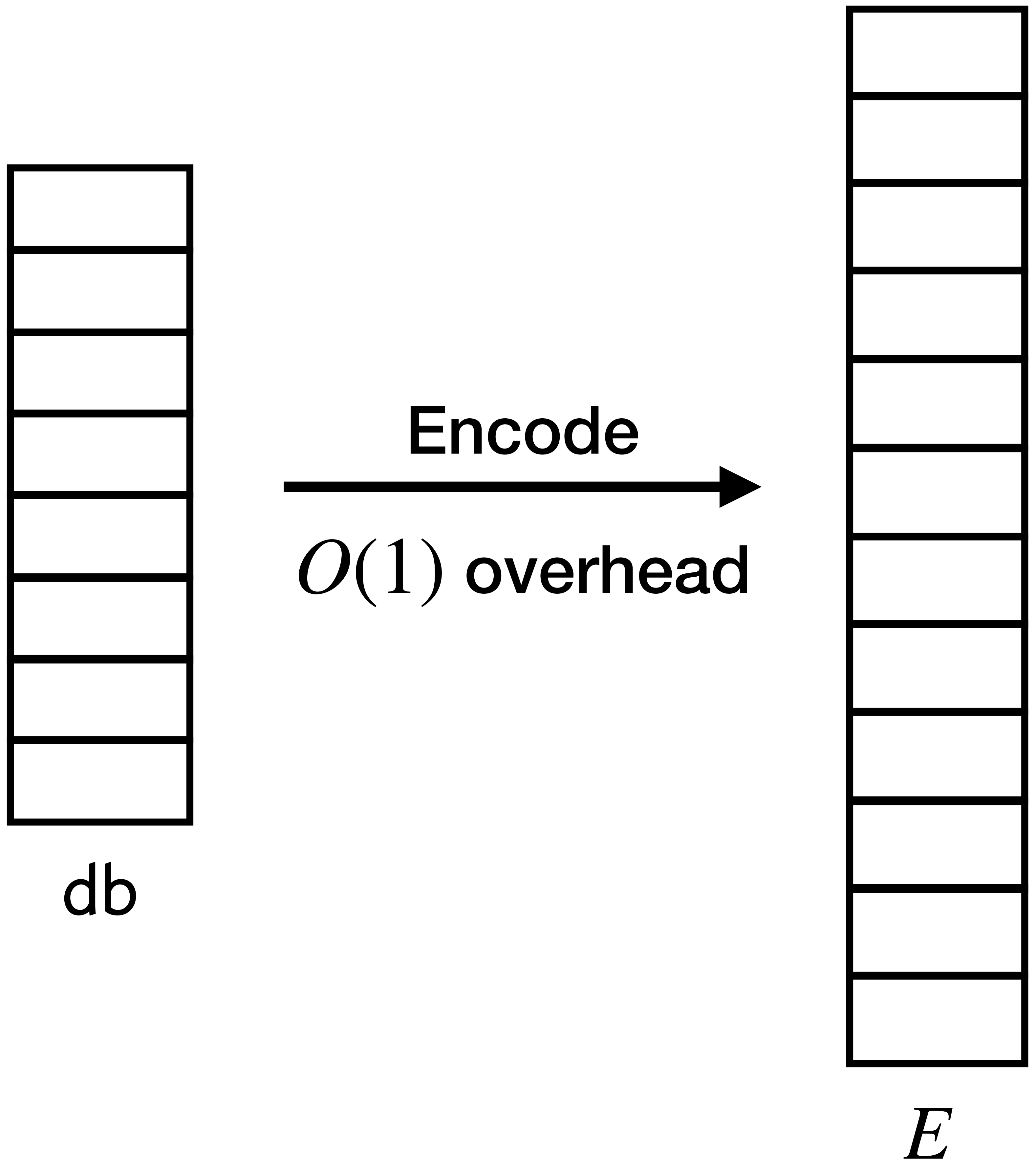


Local **Decoding**

If there are $< 1/3$ corruptions, for all i:

Locally decodable codes (LDC)

Encoding



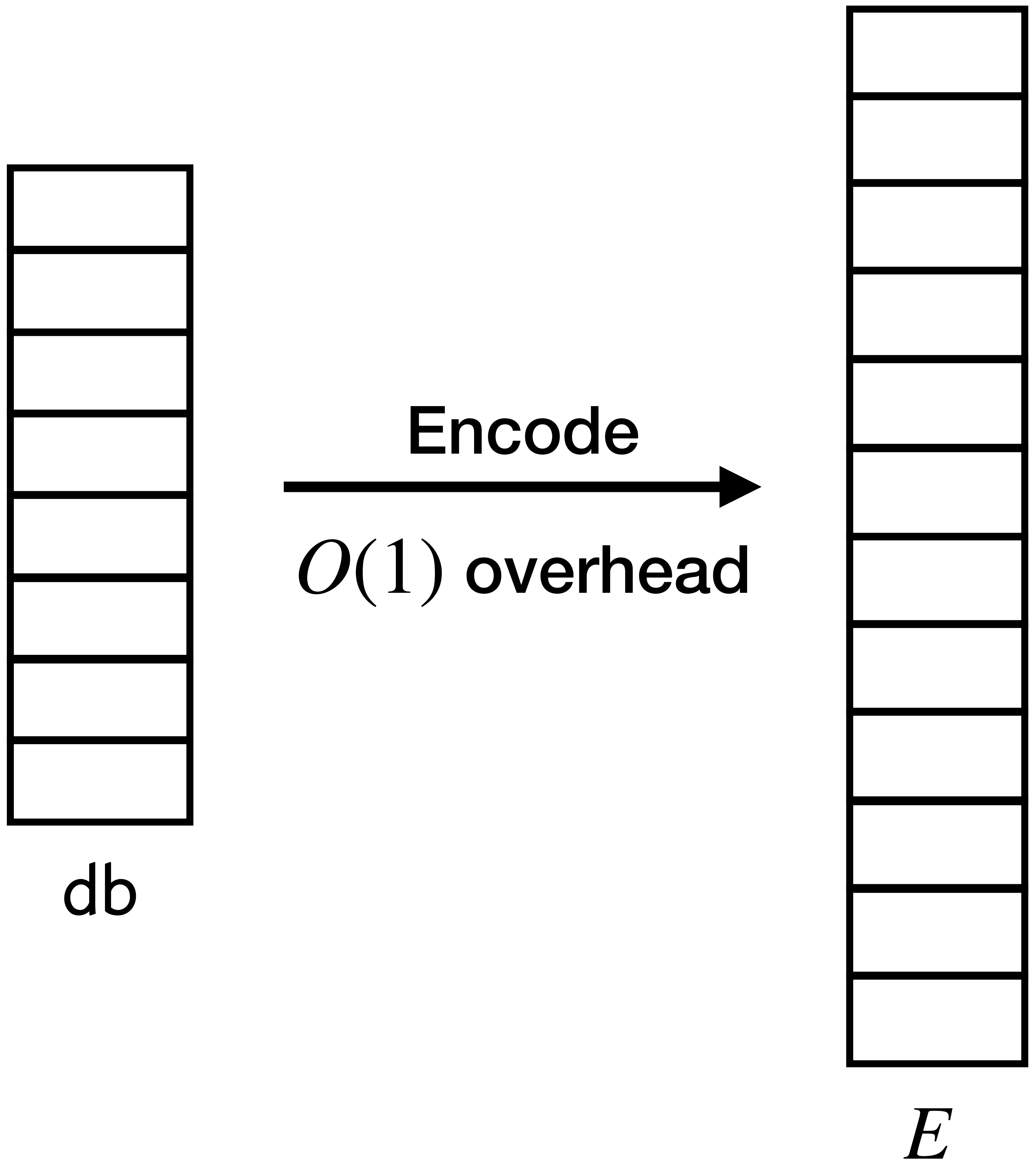
***Local* Decoding**

If there are $< 1/3$ corruptions, for all i:

$$Q \leftarrow \text{LDC.Que}(i)$$

Locally decodable codes (LDC)

Encoding



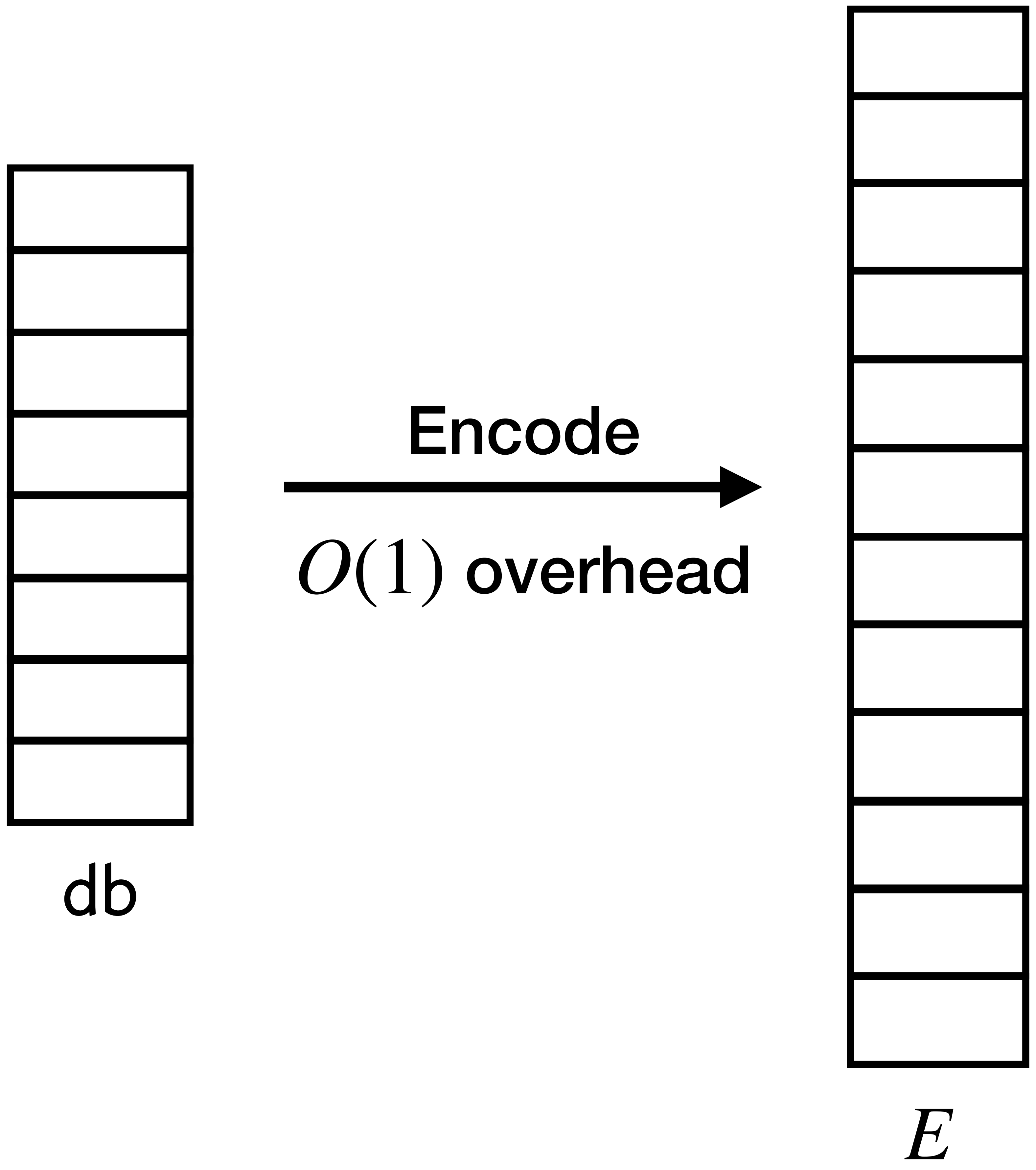
Local **Decoding**

If there are $< 1/3$ corruptions, for all i:

$$Q \leftarrow \text{LDC.Que}(i)$$

Locally decodable codes (LDC)

Encoding



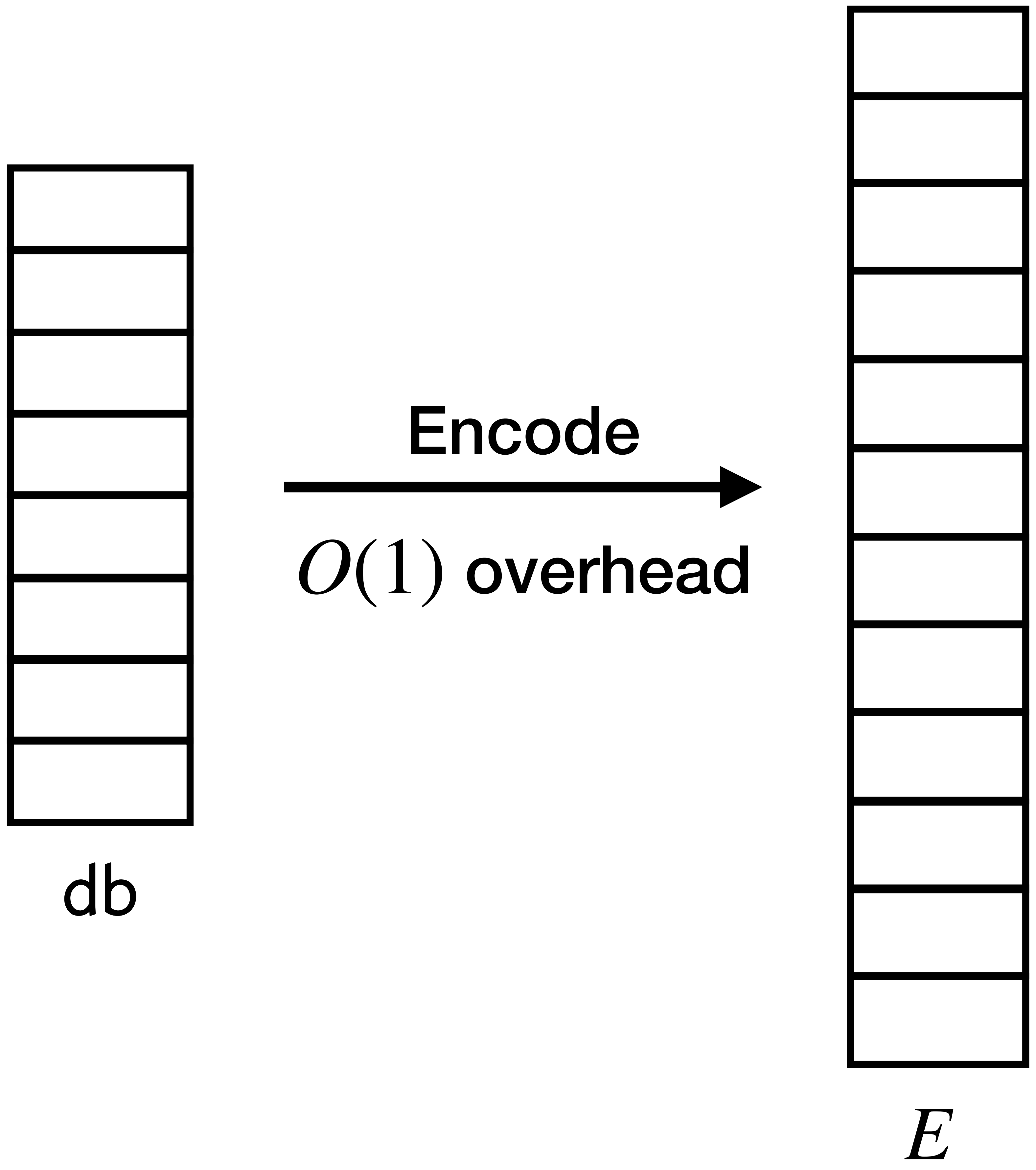
Local **Decoding**

If there are $< 1/3$ corruptions, for all i:

$$\Pr\left[\mathsf{db}_i = \mathsf{LDC.Dec}(E_Q) : \quad Q \leftarrow \mathsf{LDC.Que}(i)\right] > 2/3$$

Locally decodable codes (LDC)

Encoding



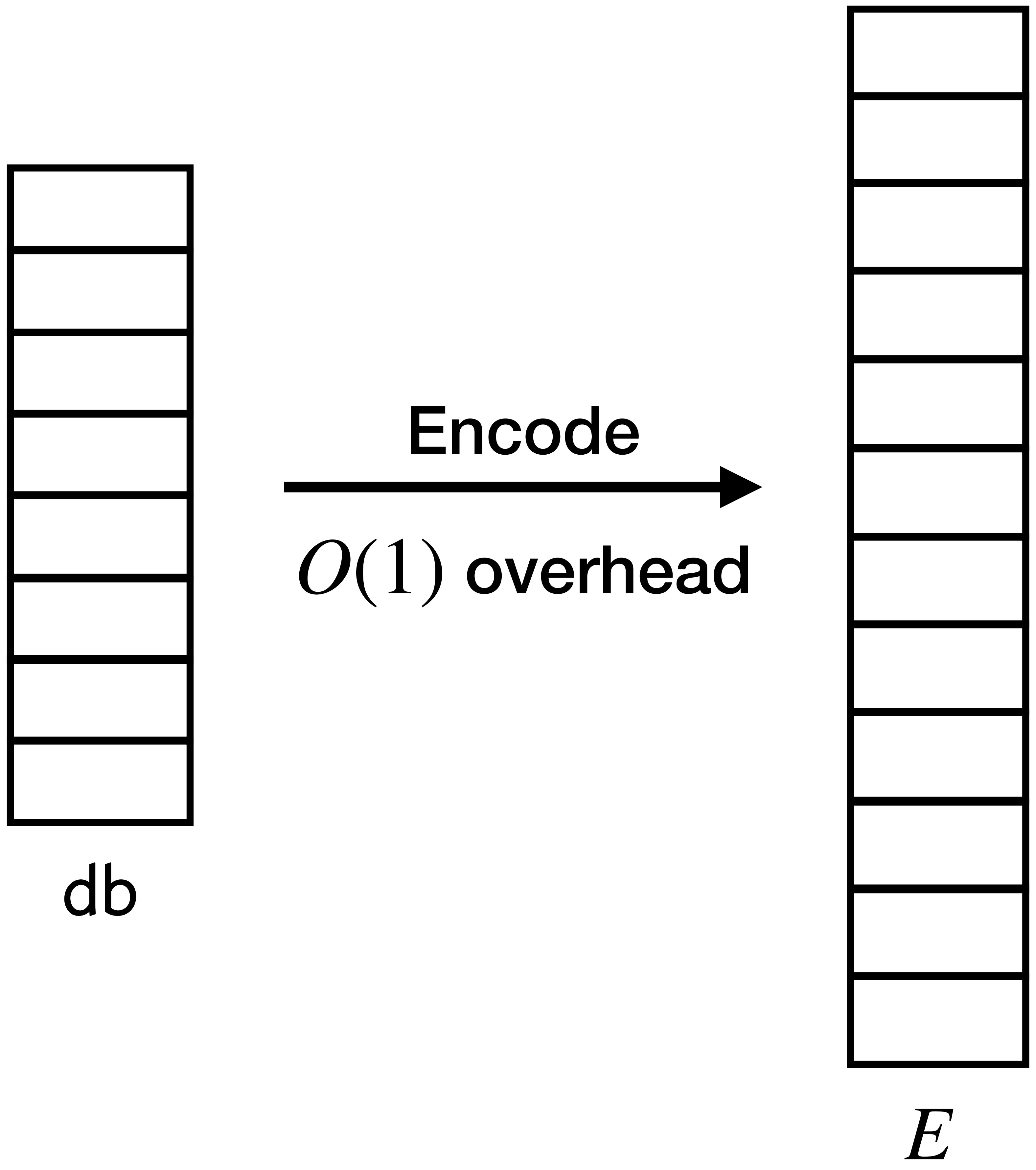
Local **Decoding**

If there are $< 1/3$ corruptions, for all i:

$$\Pr\left[\mathsf{db}_i = \mathsf{LDC.Dec}(E_Q) : \quad Q \leftarrow \mathsf{LDC.Que}(i) \right] > 2/3$$

Locally decodable codes (LDC)

Encoding



***Local* Decoding**

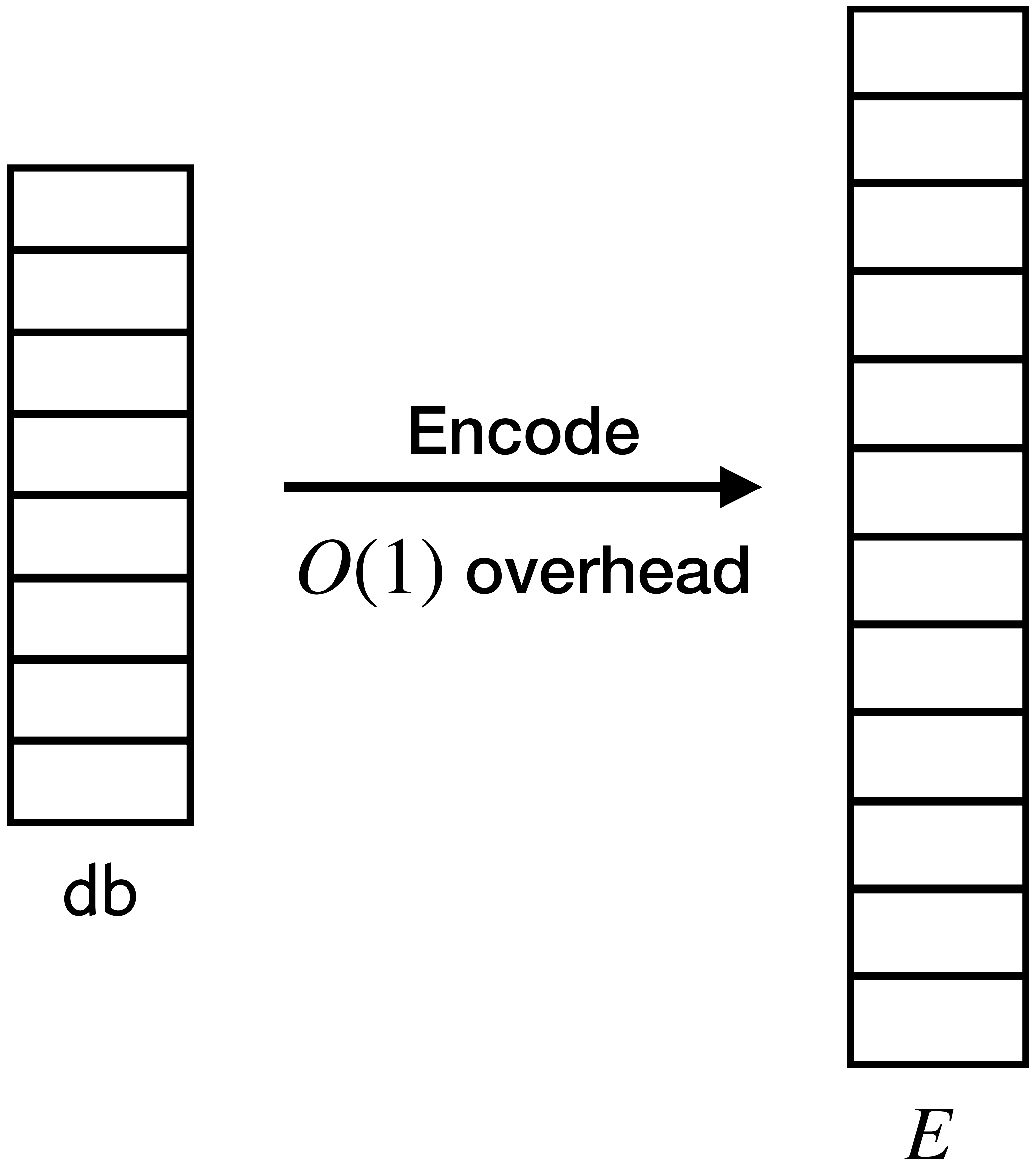
If there are $< 1/3$ corruptions, for all i:

$$\Pr\left[\mathsf{db}_i = \mathsf{LDC.Dec}(E_Q) : \quad Q \leftarrow \mathsf{LDC.Que}(i)\right] > 2/3$$

(Which means Q is "pretty random").

Locally decodable codes (LDC)

Encoding



Local **Decoding**

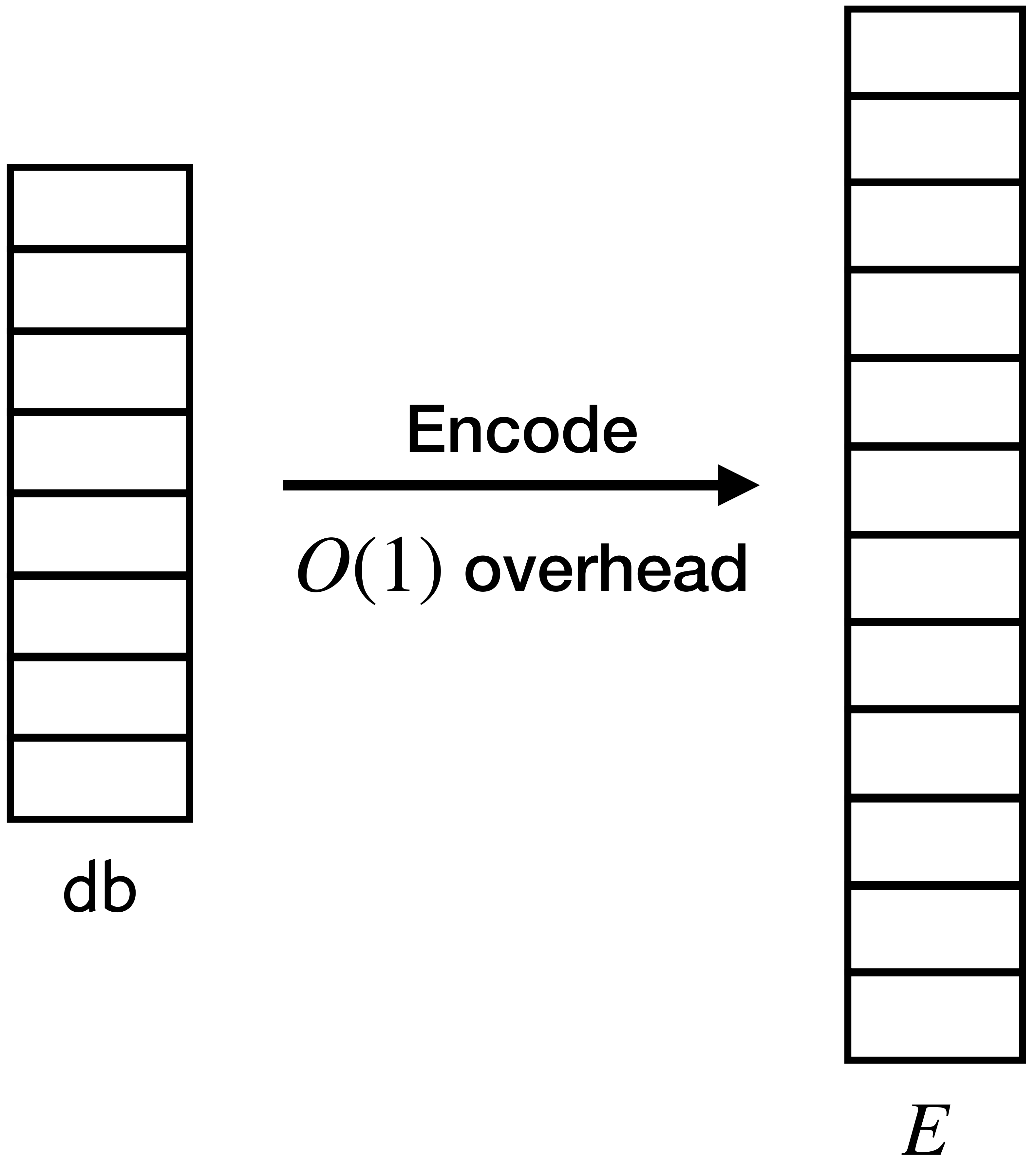
If there are $< 1/3$ corruptions, for all i:

$$\Pr\left[\mathsf{db}_i = \mathsf{LDC.Dec}(E_Q) : \quad Q \leftarrow \mathsf{LDC.Que}(i)\right] > 2/3$$

(Which means Q is "pretty random").

Locally decodable codes (LDC)

Encoding



Local **Decoding**

If there are $< 1/3$ corruptions, for all i:

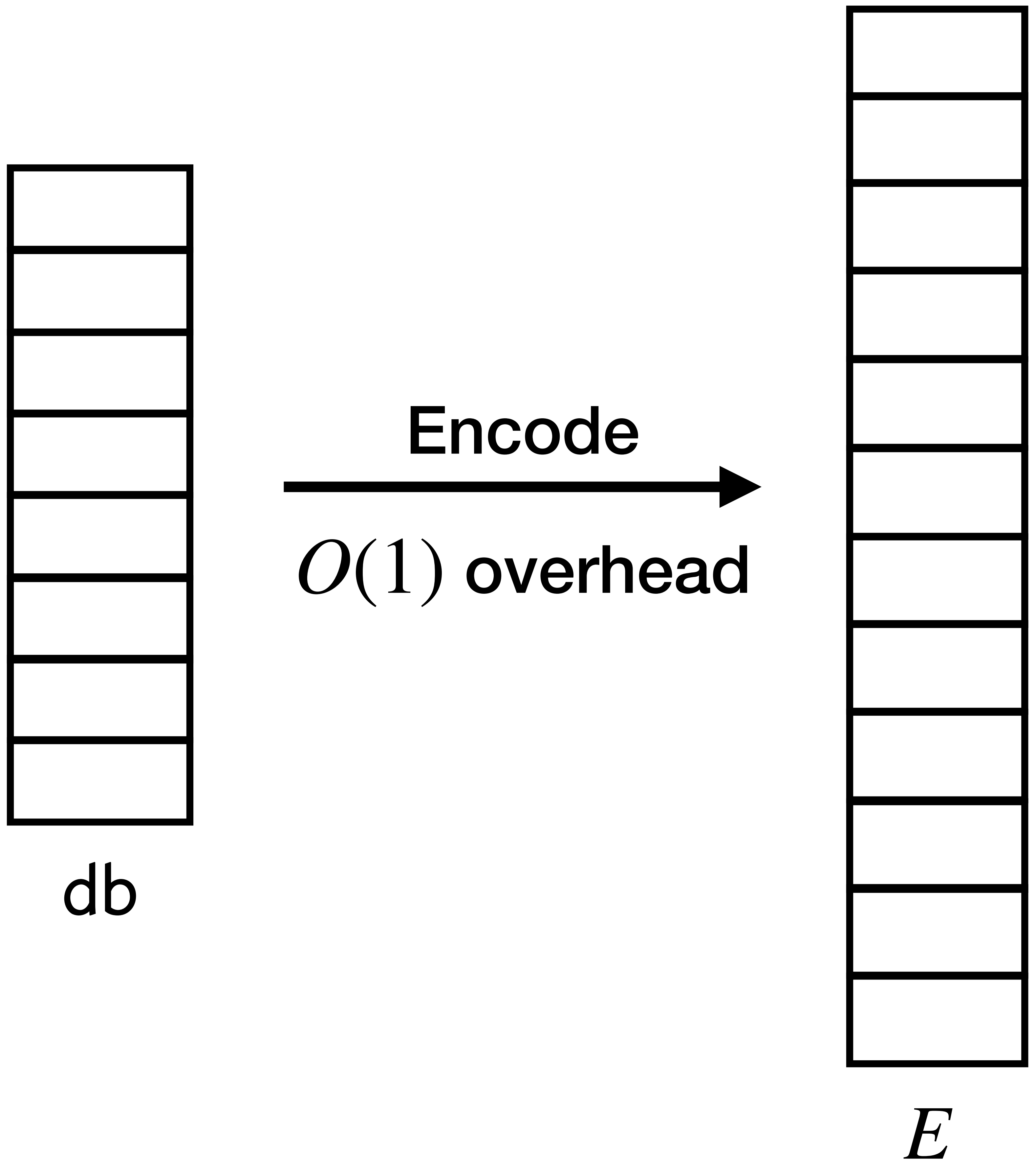
$$\Pr\left[\mathsf{db}_i = \mathsf{LDC.Dec}(E_Q) : \quad Q \leftarrow \mathsf{LDC.Que}(i) \right] > 2/3$$

(Which means Q is "pretty random").

Smoothness:

Locally decodable codes (LDC)

Encoding



Local **Decoding**

If there are $< 1/3$ corruptions, for all i:

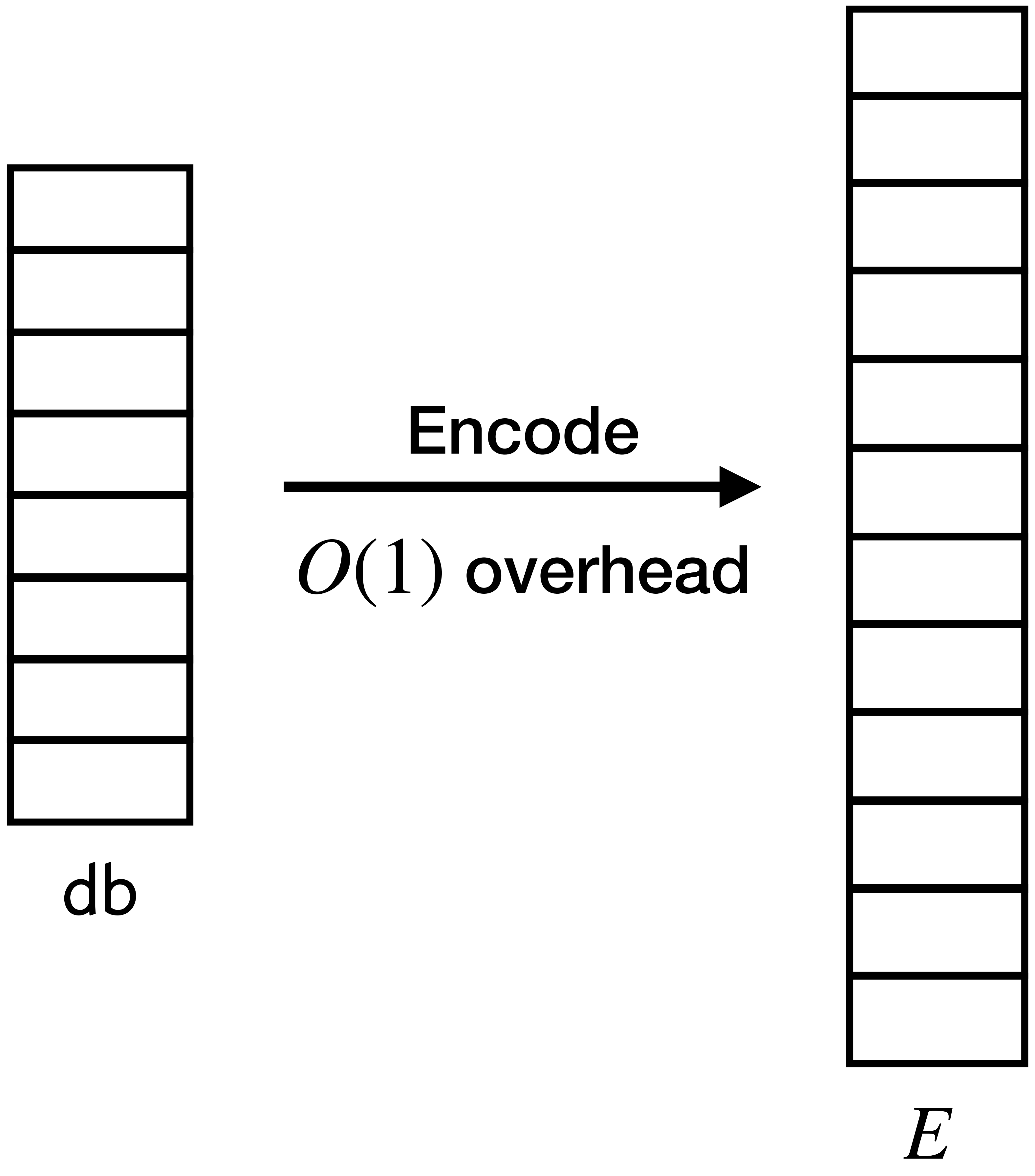
$$\Pr\left[\mathsf{db}_i = \mathsf{LDC.Dec}(E_Q) : \quad Q \leftarrow \mathsf{LDC.Que}(i)\right] > 2/3$$

(Which means Q is "pretty random").

Smoothness:

Locally decodable codes (LDC)

Encoding



***Local* Decoding**

If there are $< 1/3$ corruptions, for all i:

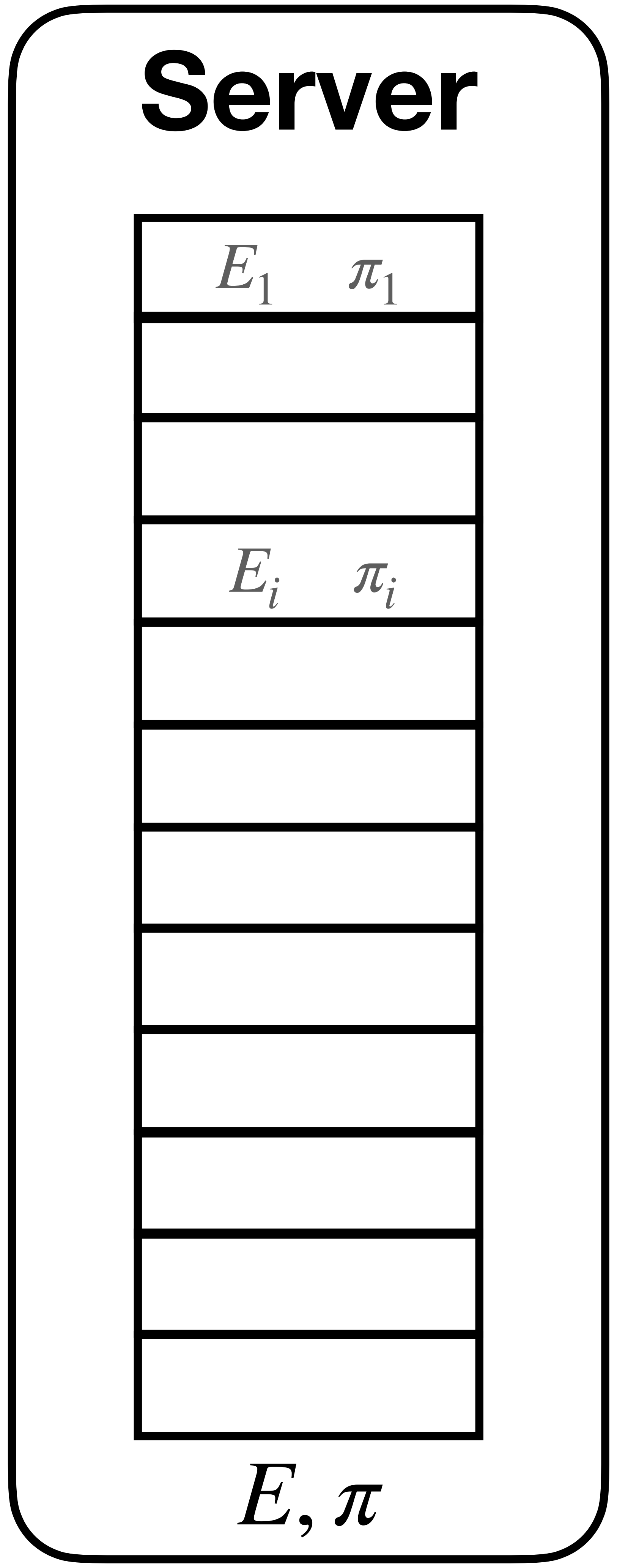
$$\Pr\left[\text{db}_i = \text{LDC.Dec}(E_Q) : \quad Q \leftarrow \text{LDC.Que}(i)\right] > 2/3$$

(Which means Q is “pretty random”).

Smoothness:

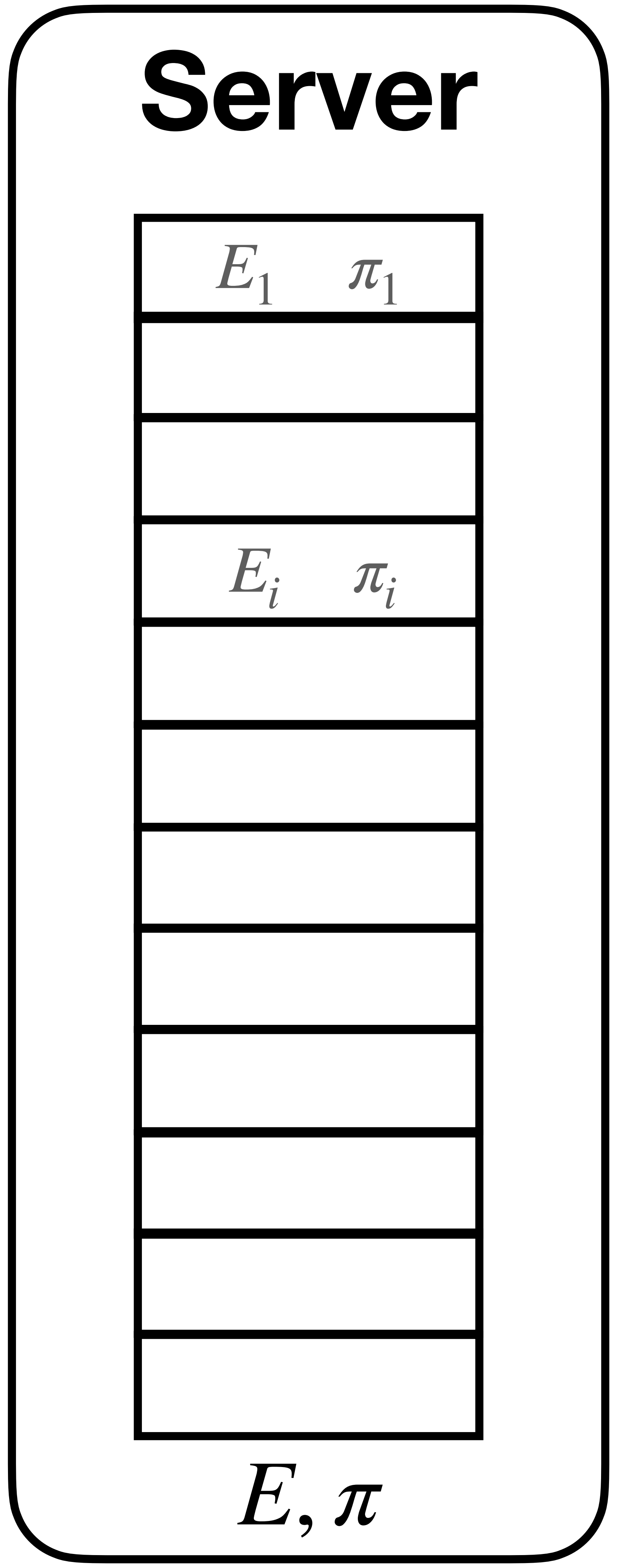
For all i: $x \leftarrow \{Q \leftarrow \text{LDC.Que}(i)\}$
is *uniformly random in* $[|E|]$

Attempt 2: LDC + VC + PIR



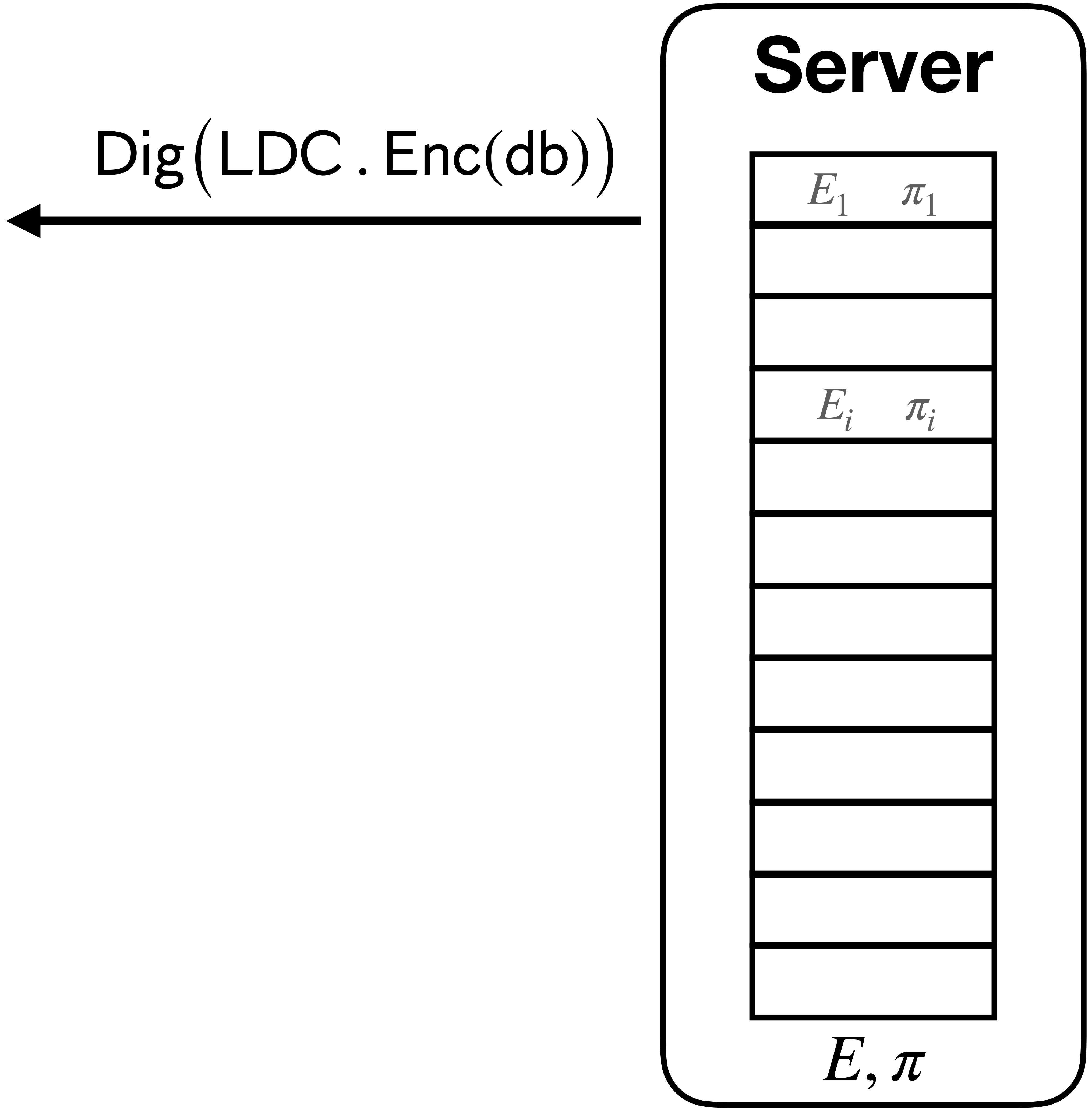
Attempt 2: LDC + VC + PIR

Offline



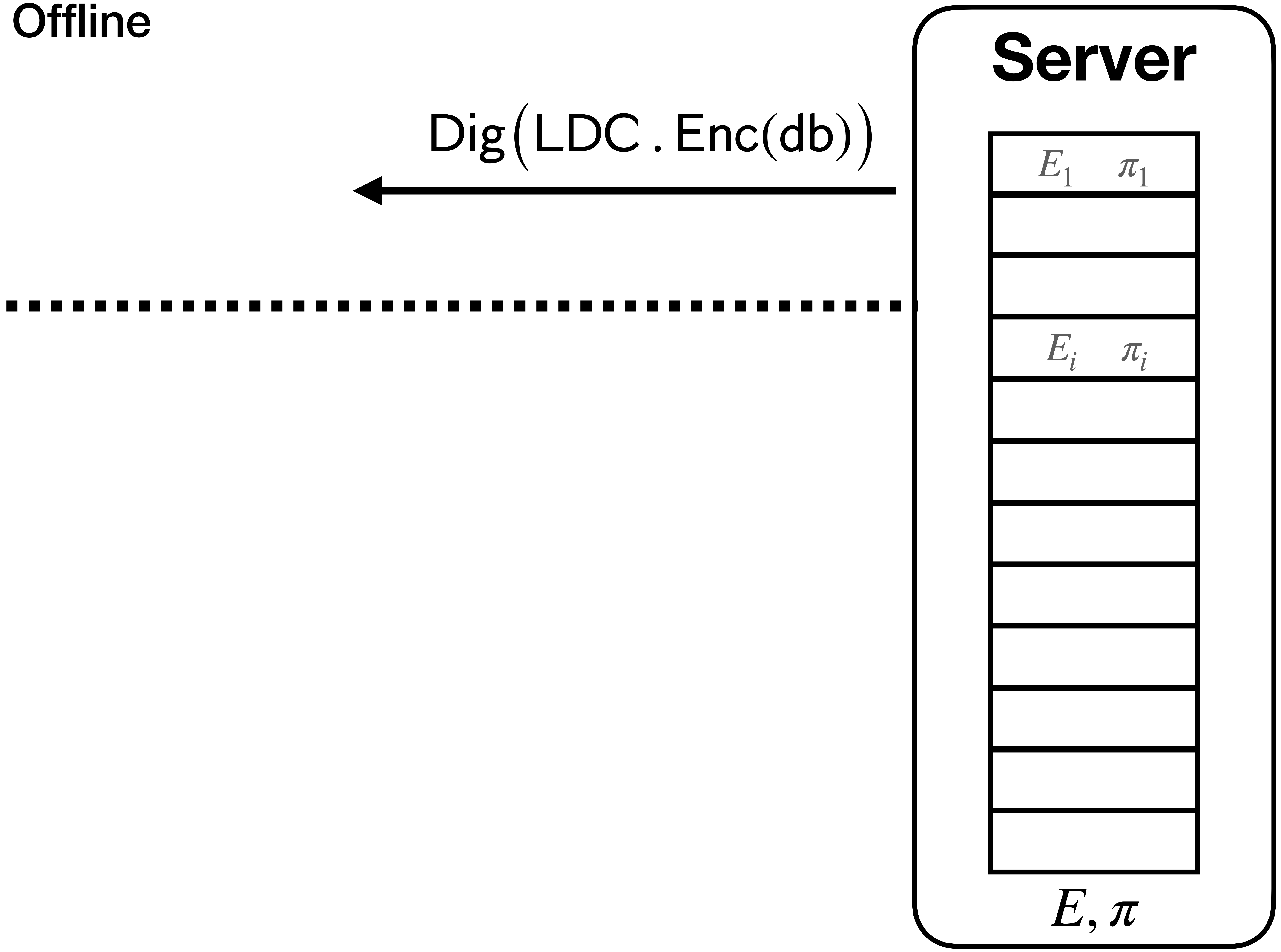
Attempt 2: LDC + VC + PIR

Offline

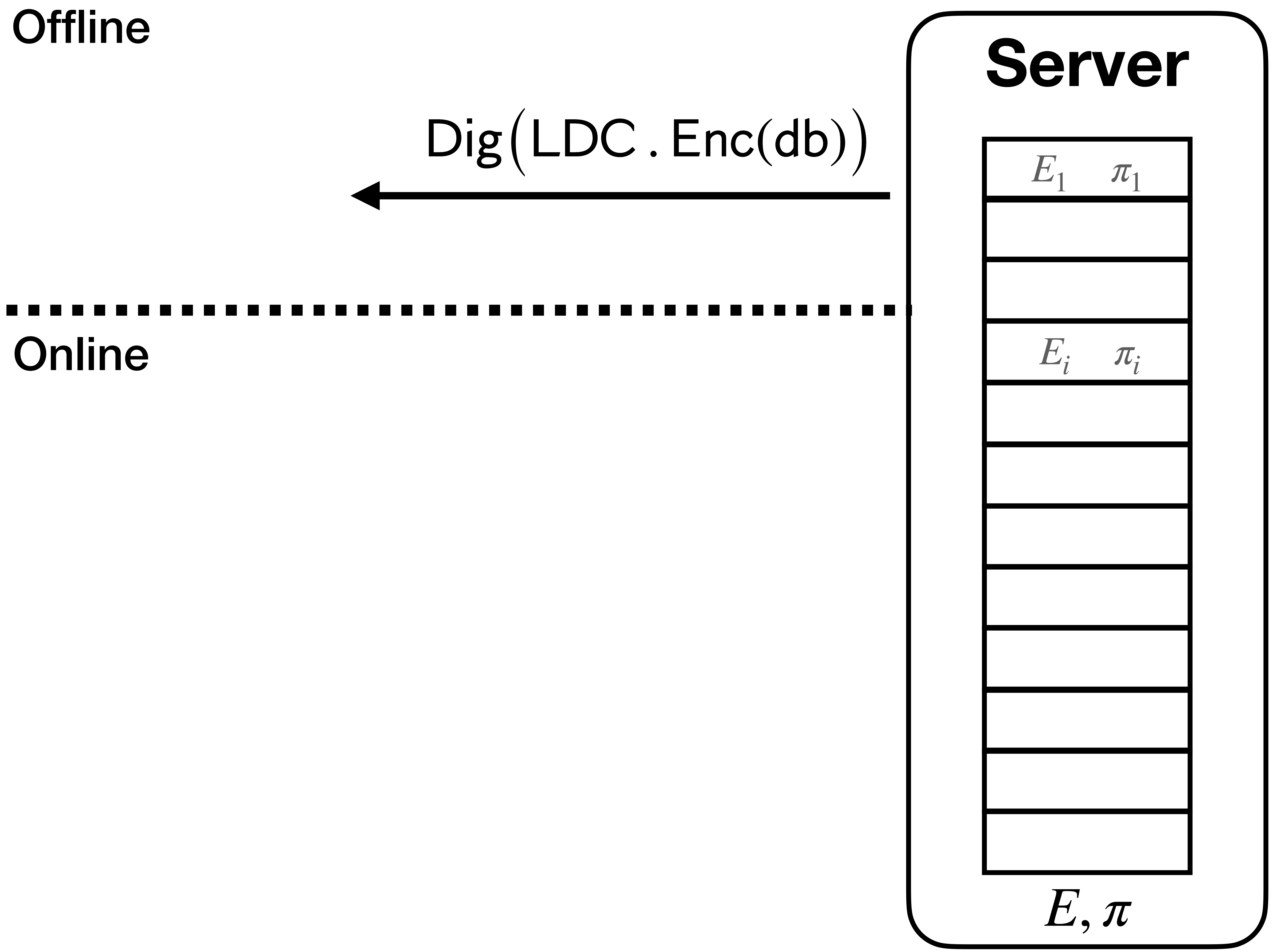


Attempt 2: LDC + VC + PIR

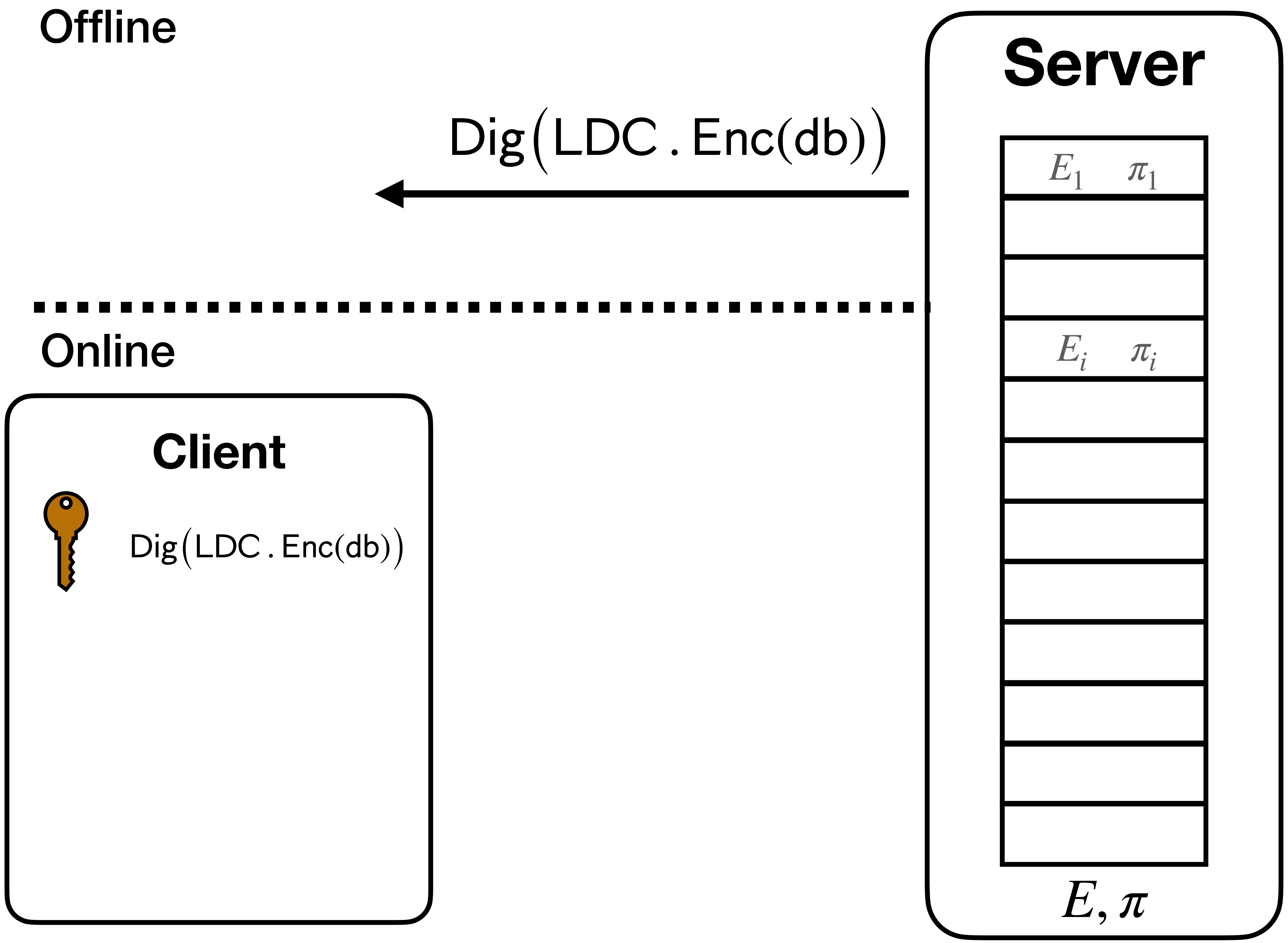
Offline



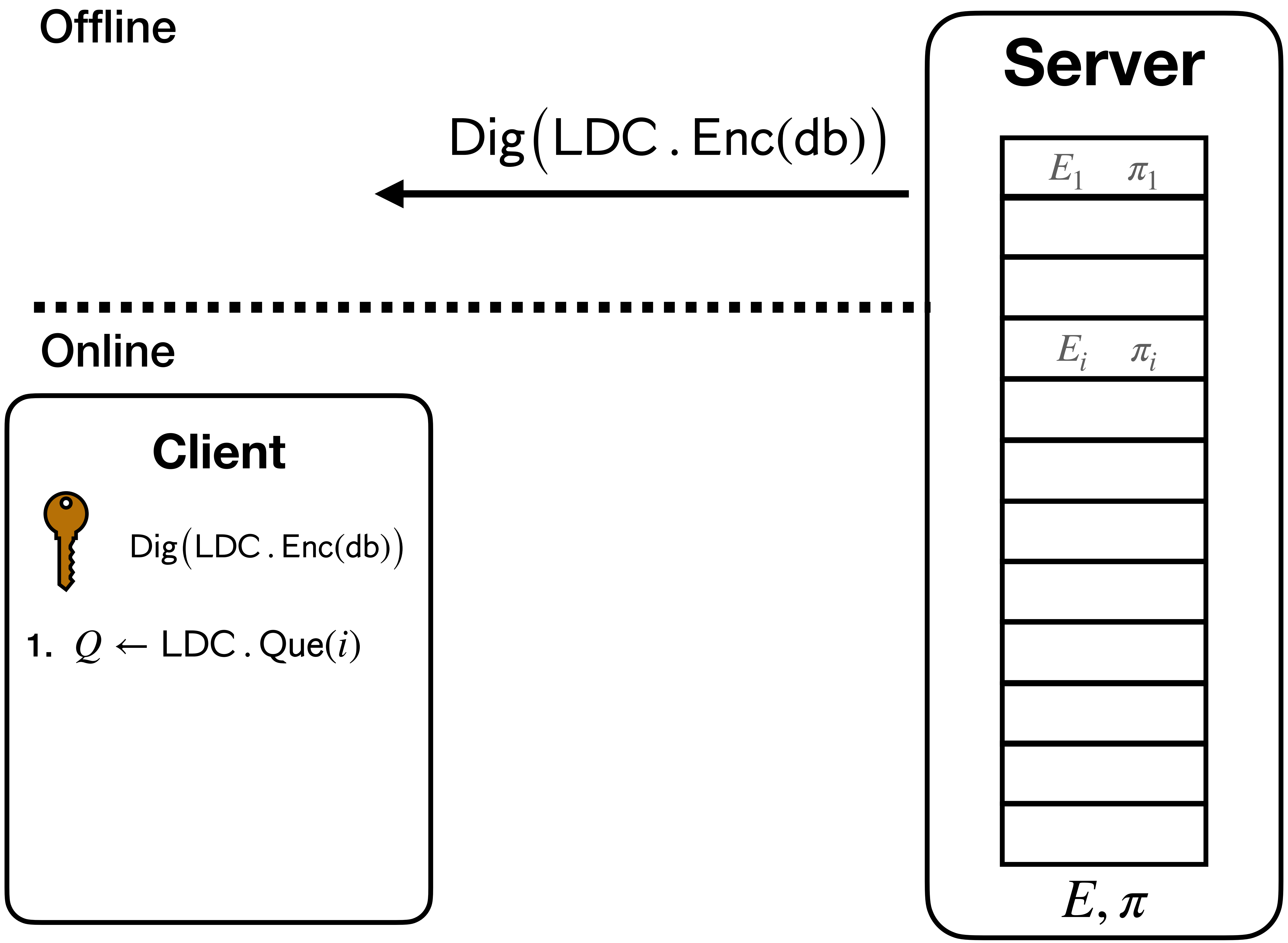
Attempt 2: LDC + VC + PIR



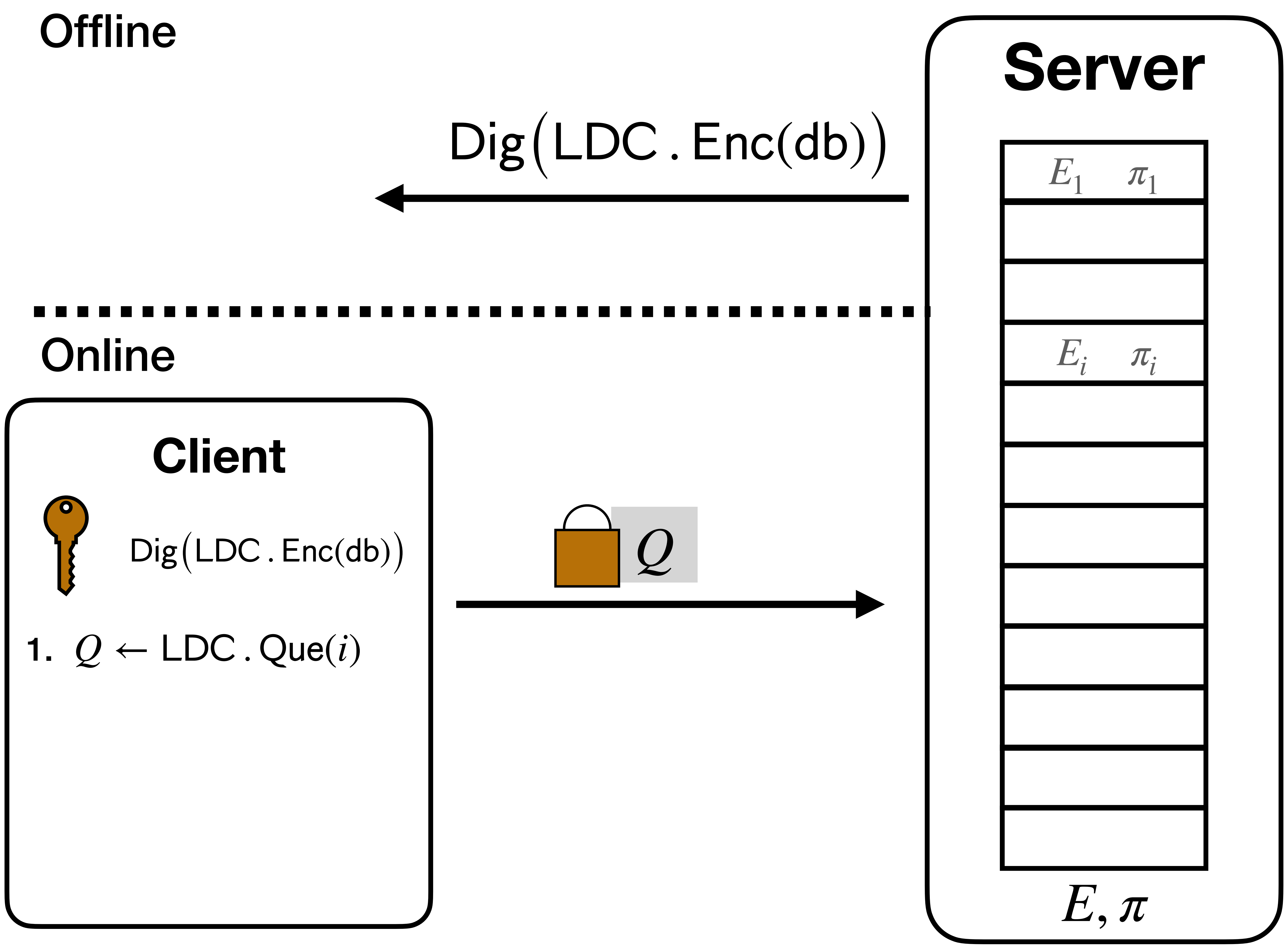
Attempt 2: LDC + VC + PIR



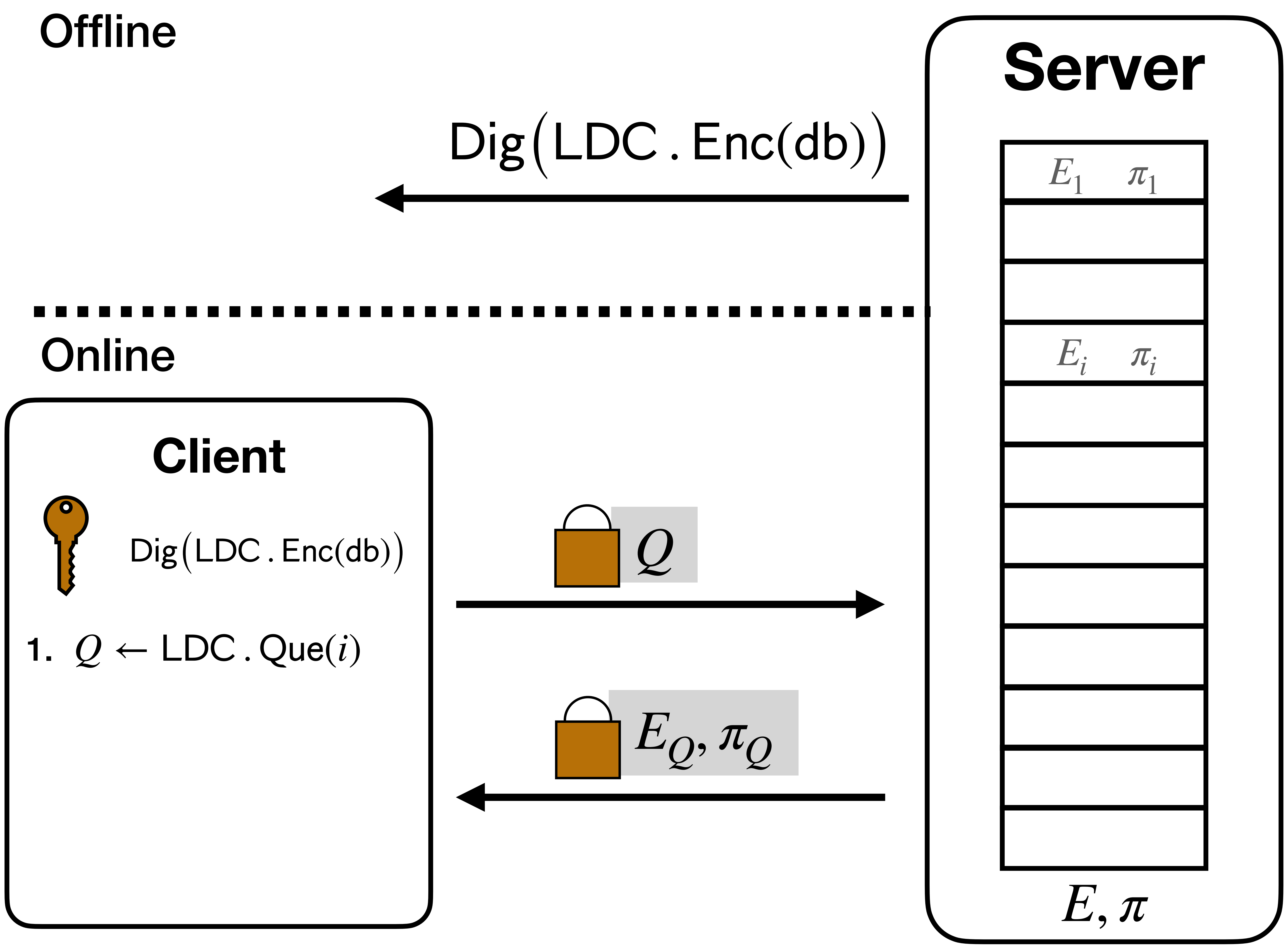
Attempt 2: LDC + VC + PIR



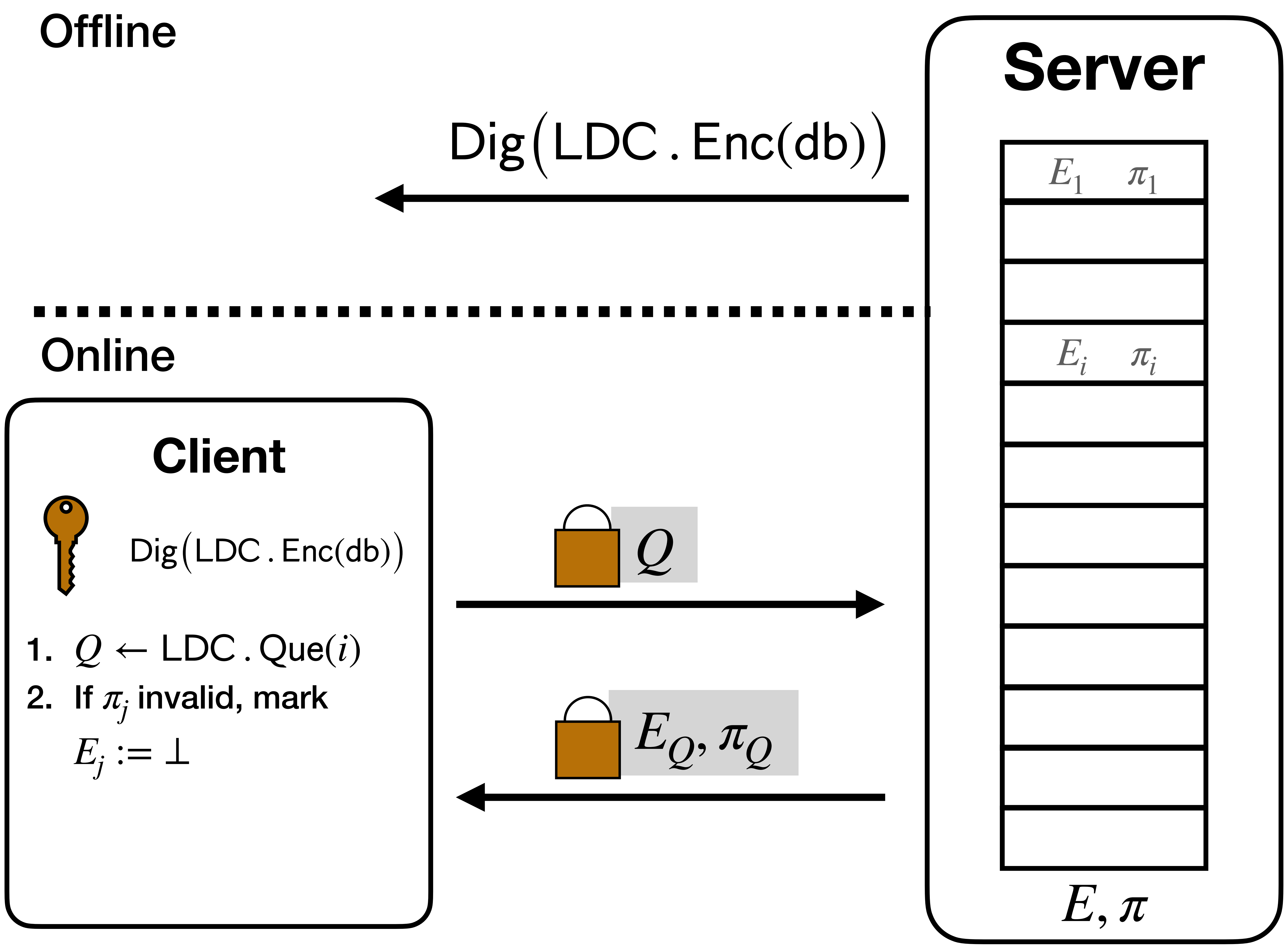
Attempt 2: LDC + VC + PIR



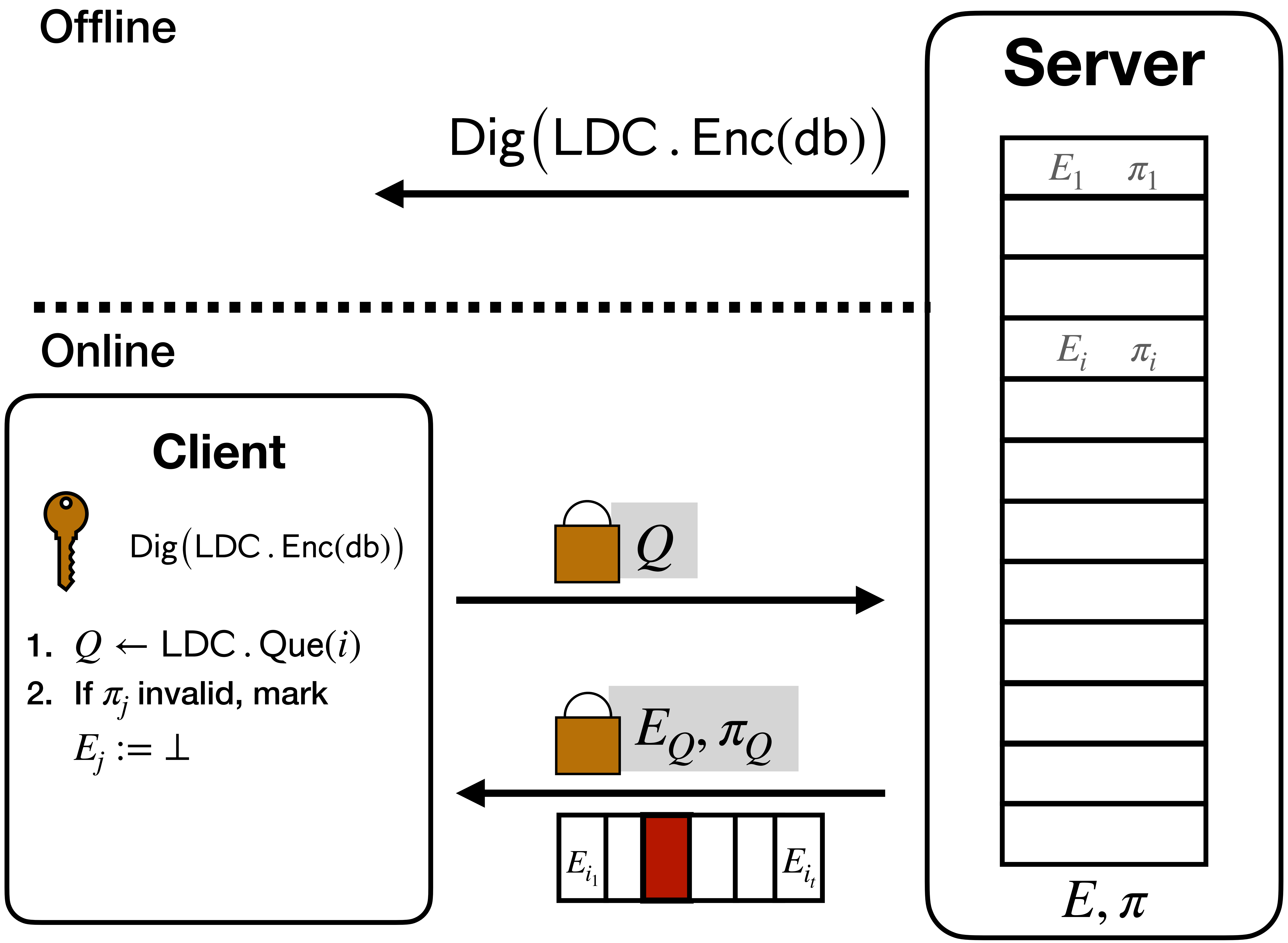
Attempt 2: LDC + VC + PIR



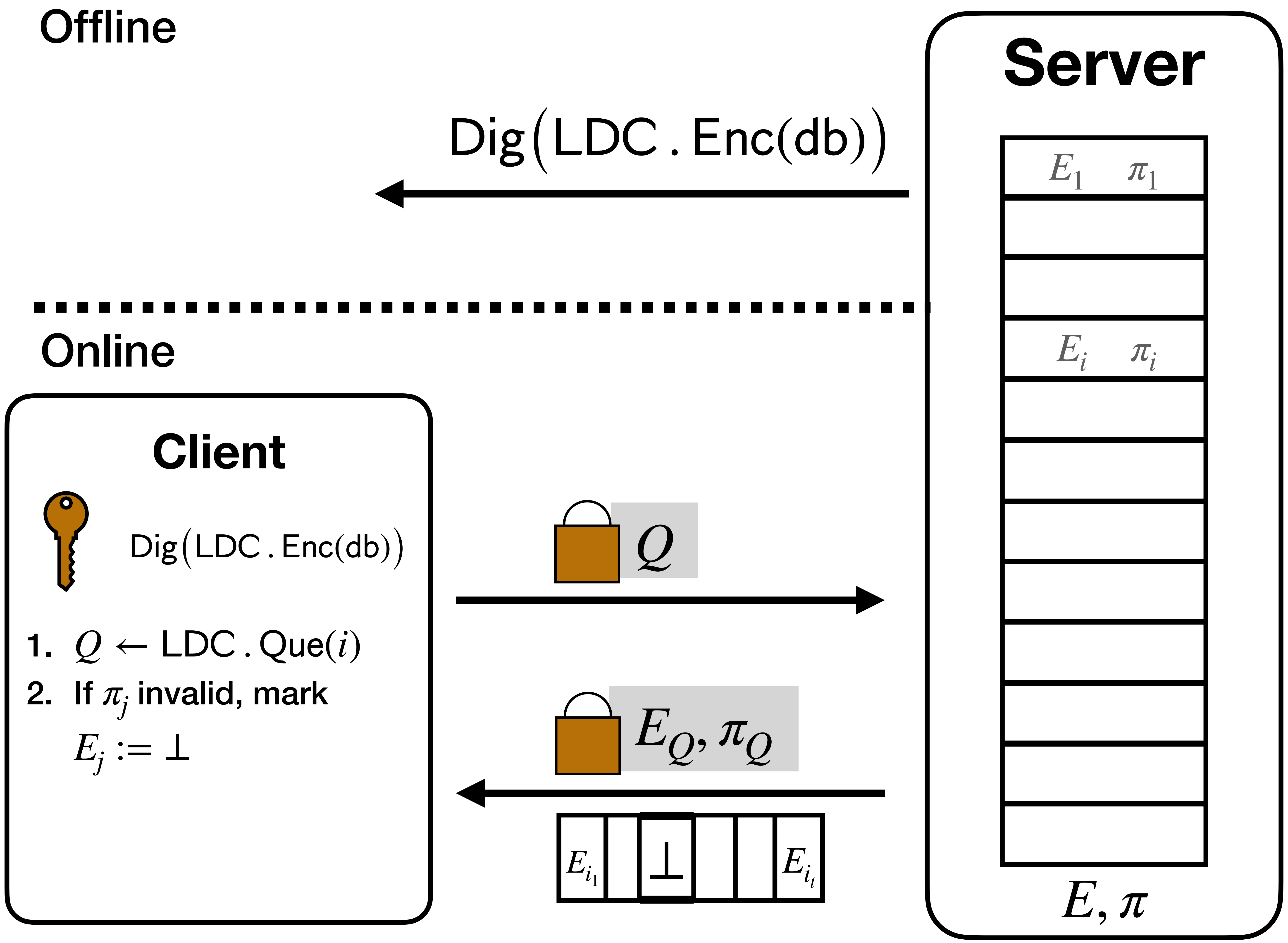
Attempt 2: LDC + VC + PIR



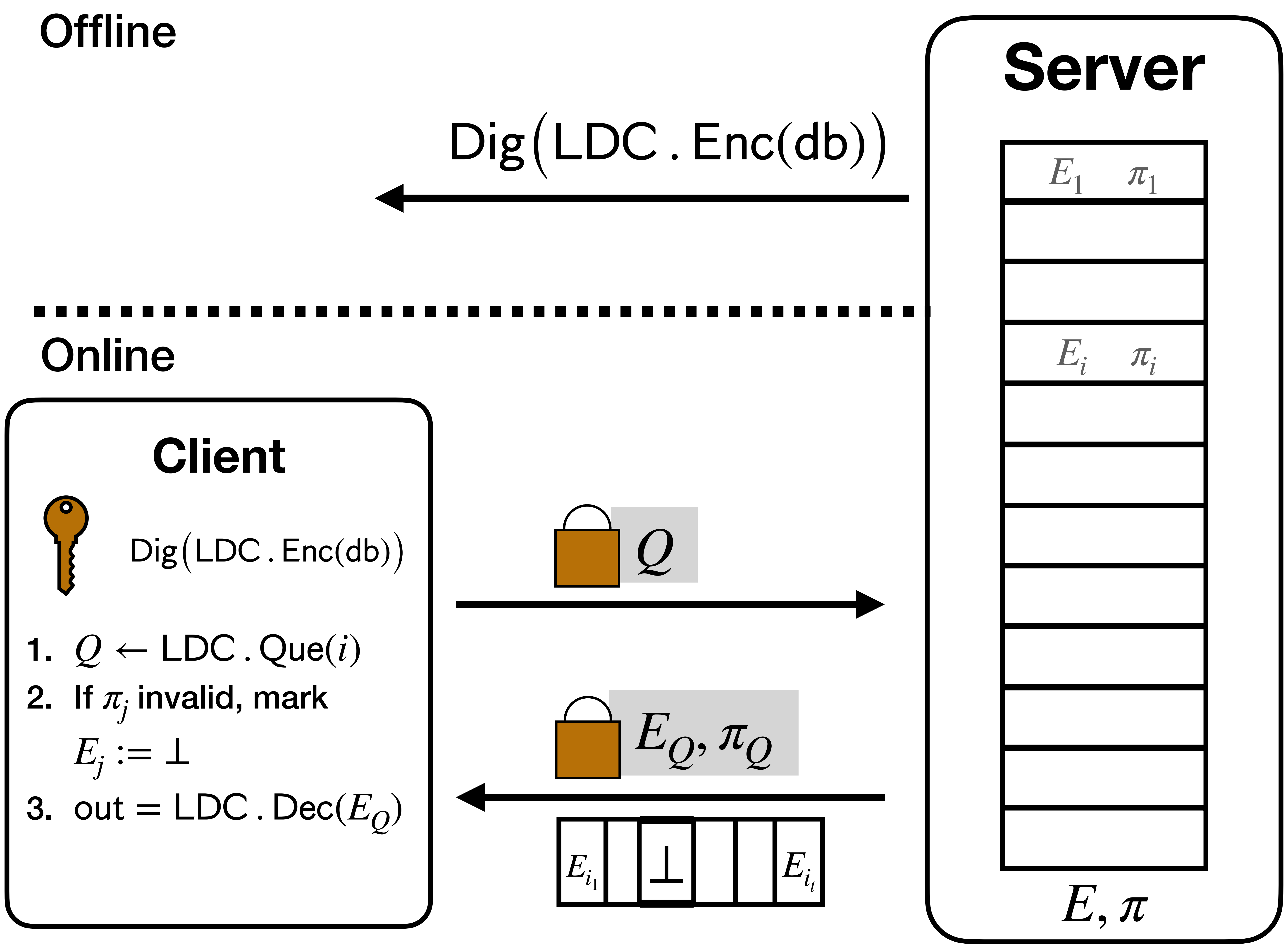
Attempt 2: LDC + VC + PIR



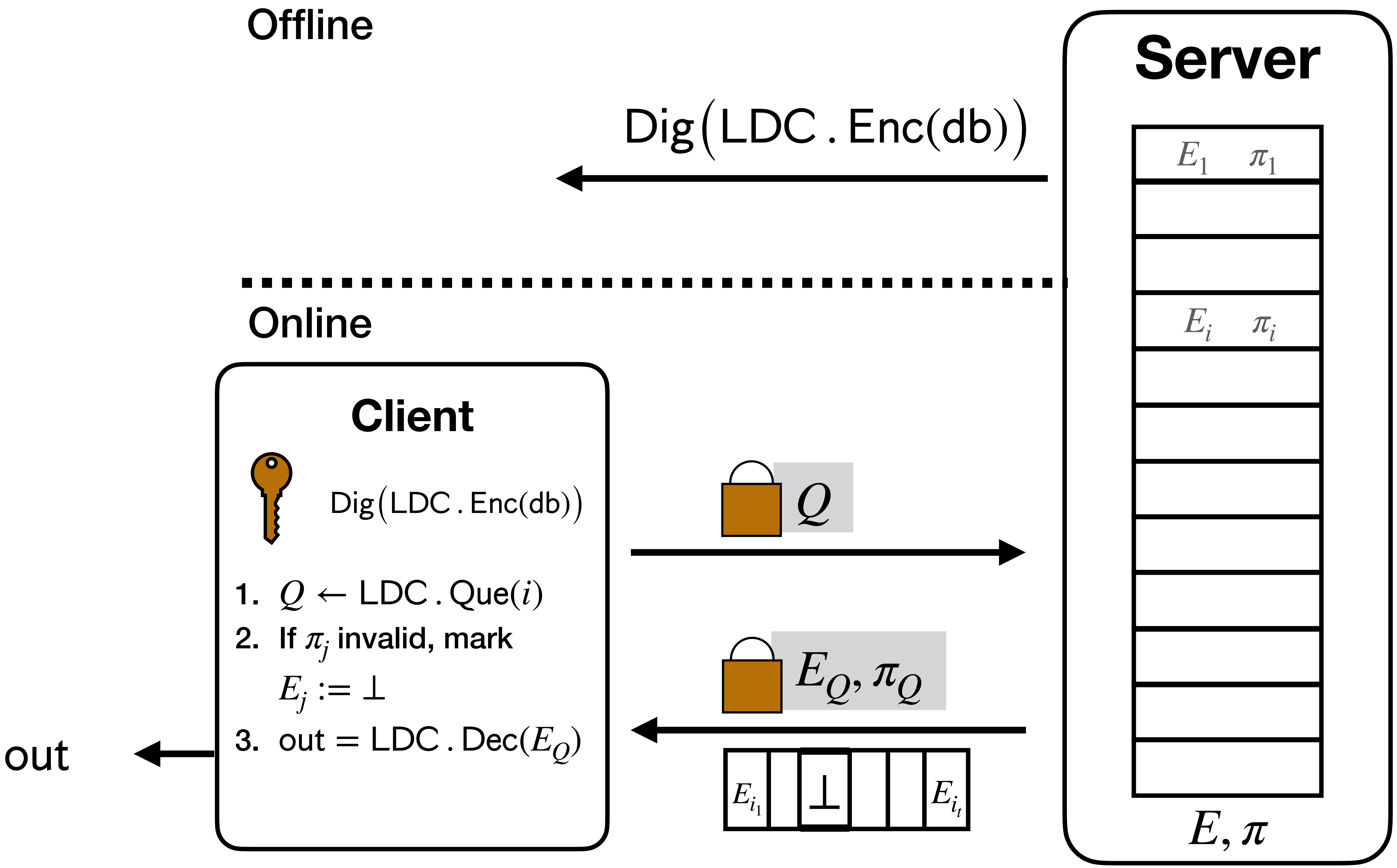
Attempt 2: LDC + VC + PIR



Attempt 2: LDC + VC + PIR

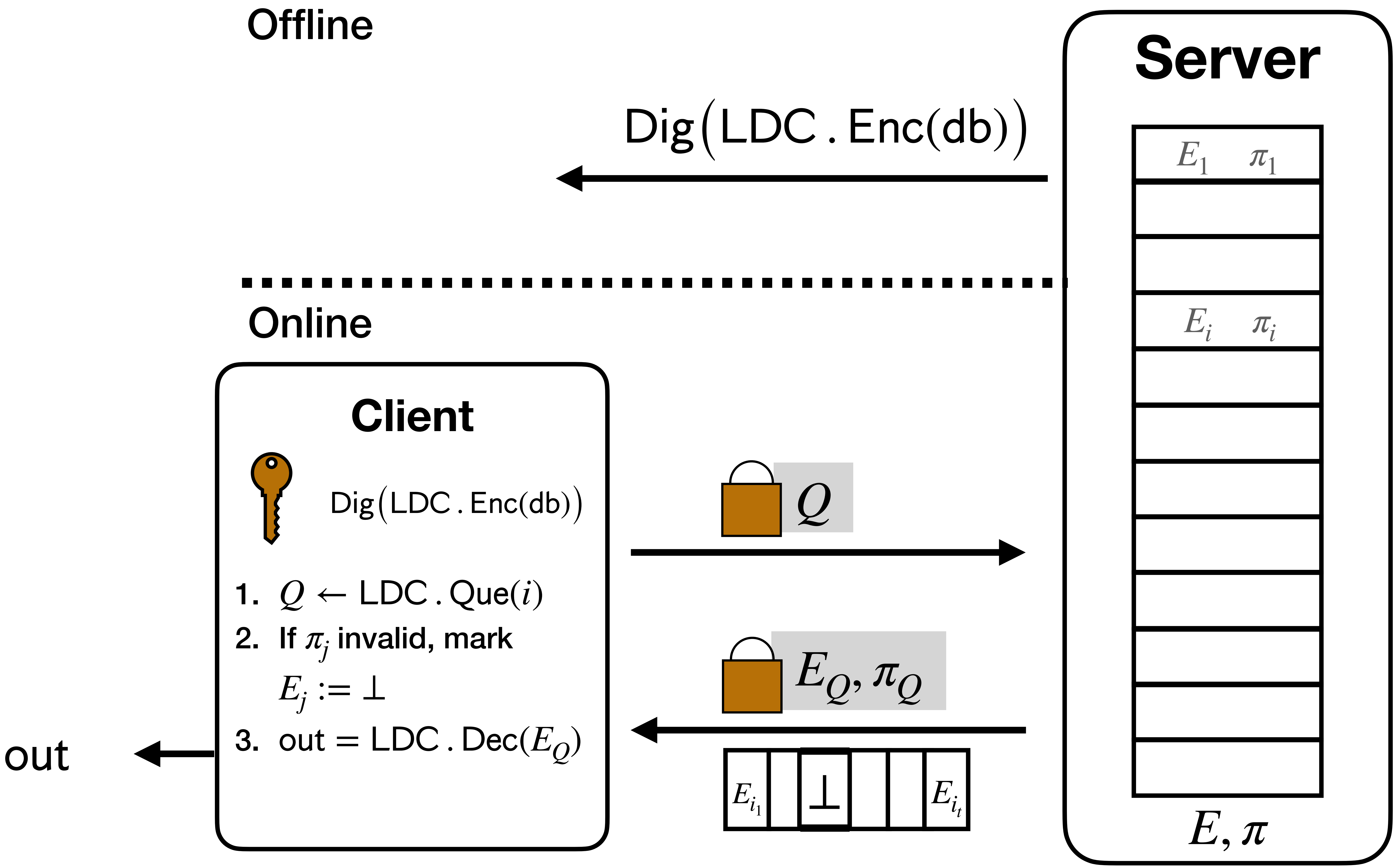


Attempt 2: LDC + VC + PIR



Attempt 2: LDC + VC + PIR

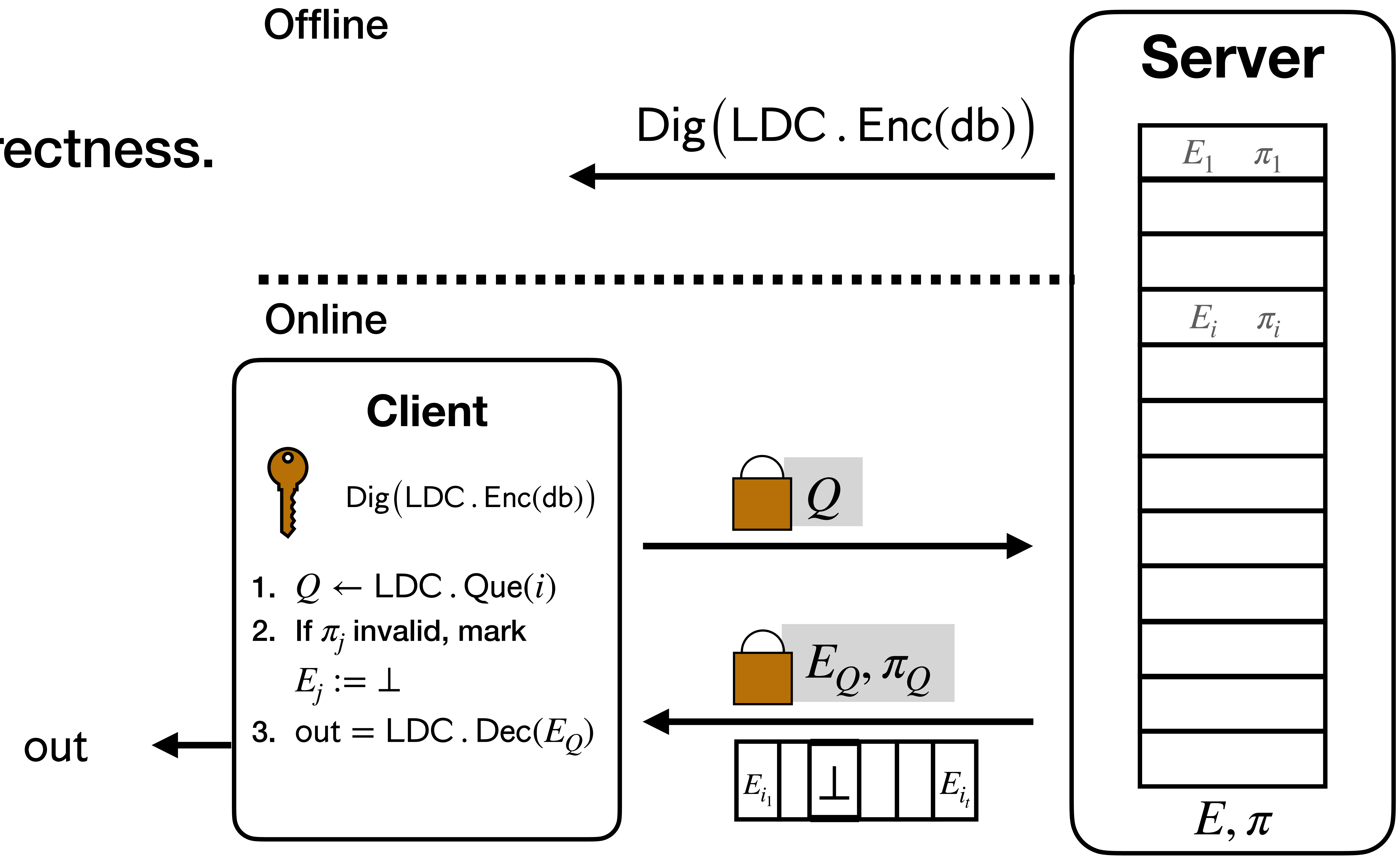
Properties



Attempt 2: LDC + VC + PIR

Properties

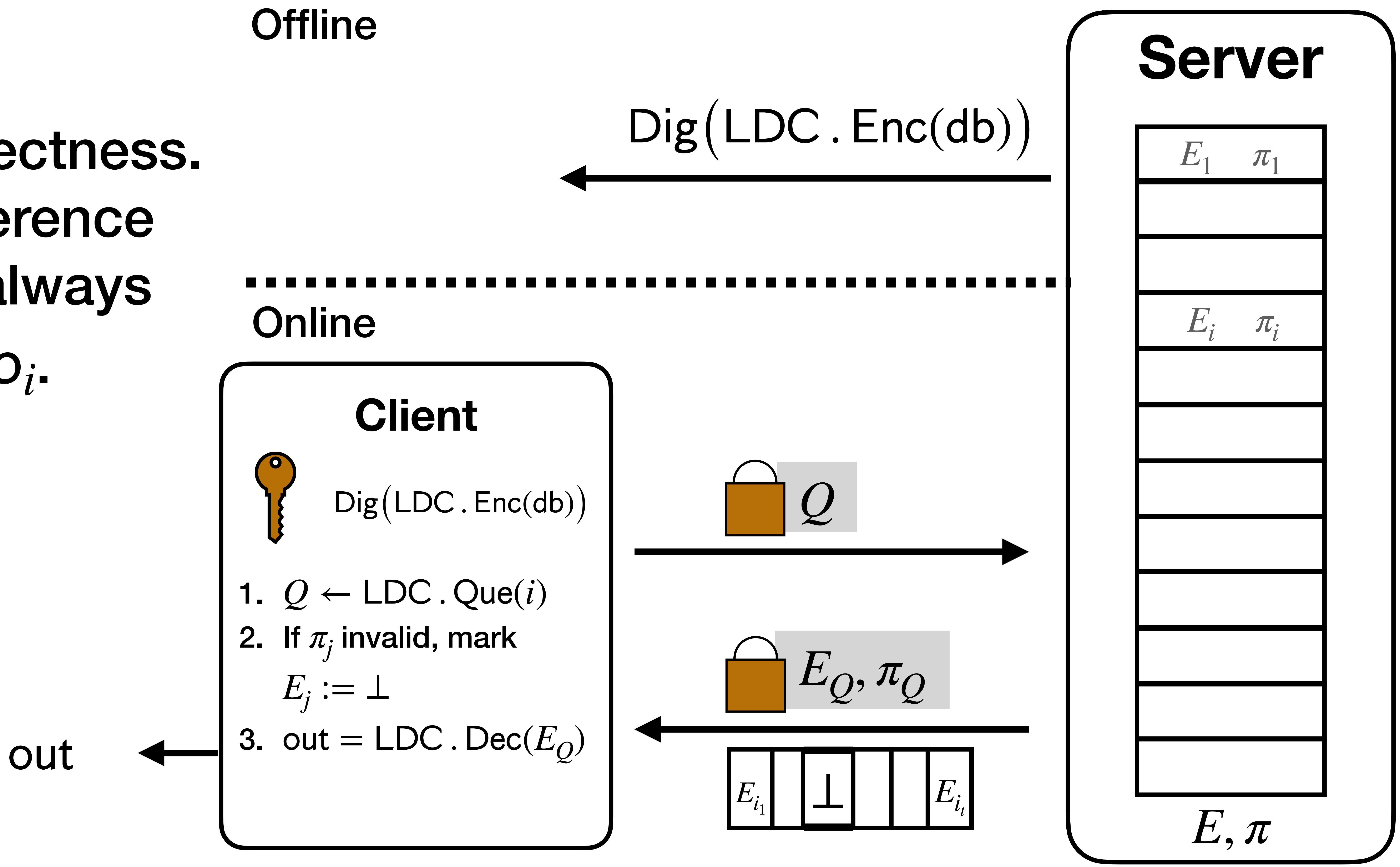
1. Preserves correctness.



Attempt 2: LDC + VC + PIR

Properties

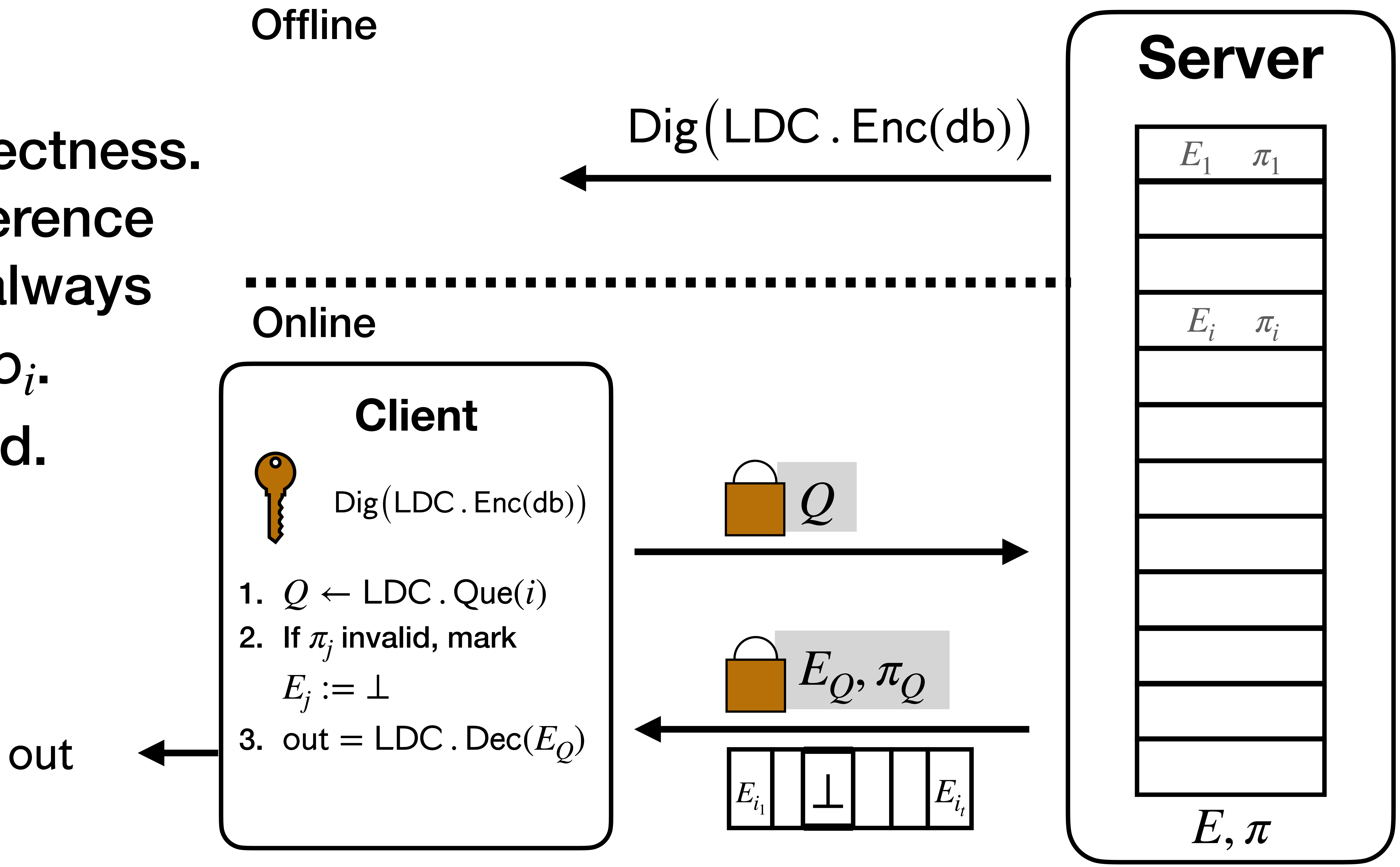
1. Preserves correctness.
2. Preserves coherence because LDC always outputs $\perp$ or db_i.



Attempt 2: LDC + VC + PIR

Properties

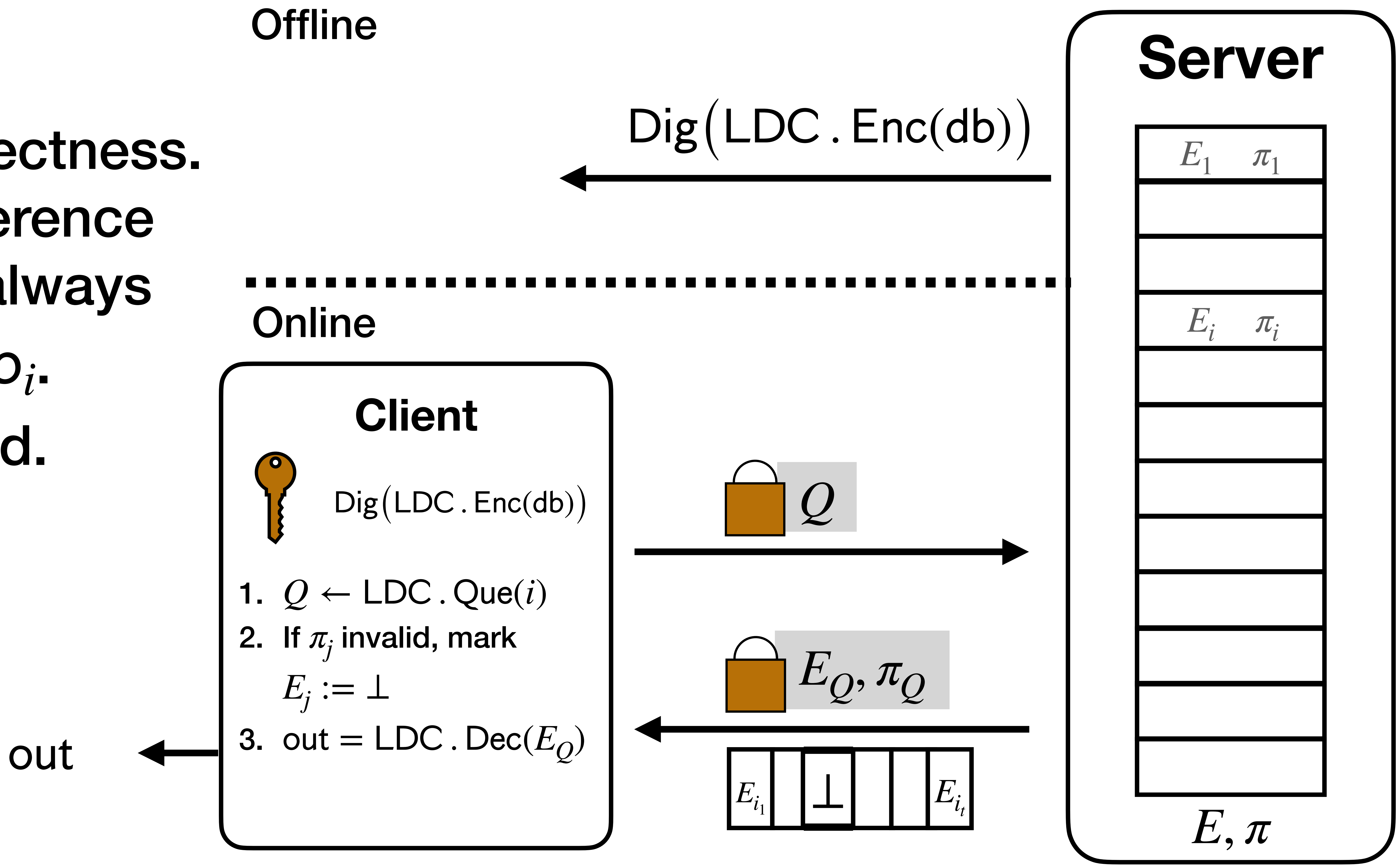
1. Preserves correctness.
2. Preserves coherence because LDC always outputs $\perp$ or db_i.
3. $O(N^{\epsilon})$ overhead.



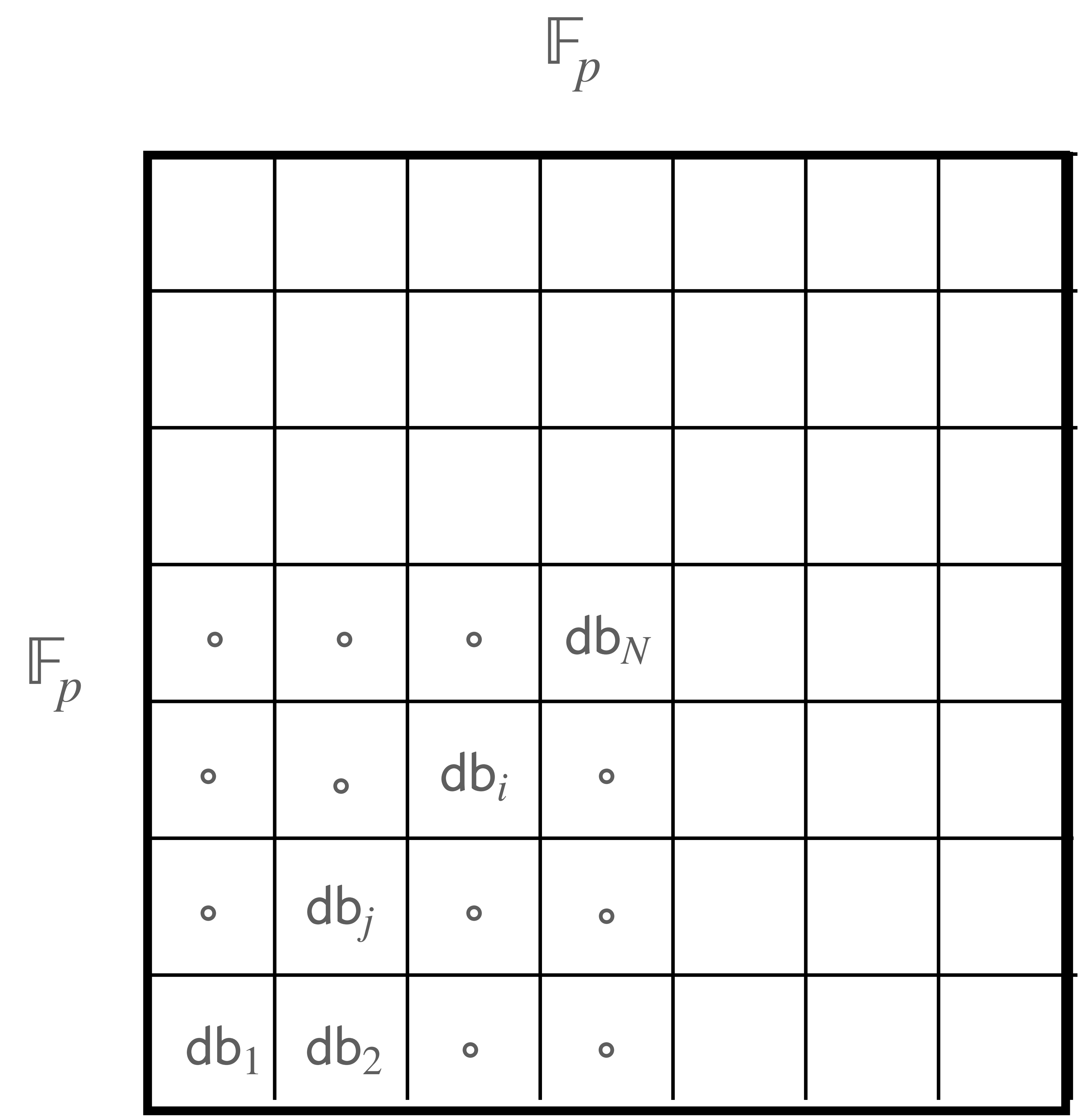
Attempt 2: LDC + VC + PIR

Properties

1. Preserves correctness.
2. Preserves coherence because LDC always outputs $\perp$ or db_i.
3. $O(N^\epsilon)$ overhead.
4. Privacy?

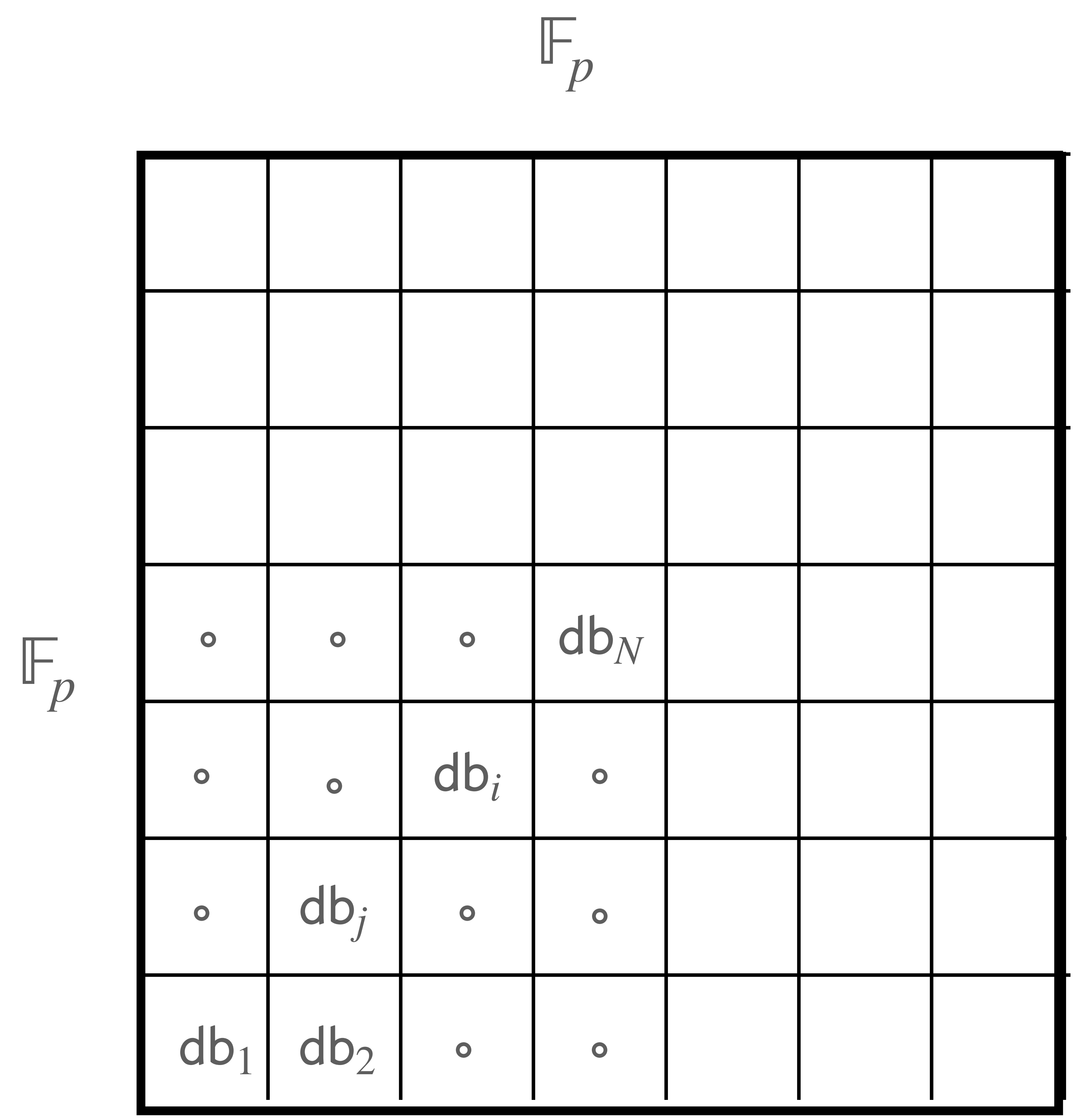


The Reed-Muller (RM) LDC



The Reed-Muller (RM) LDC

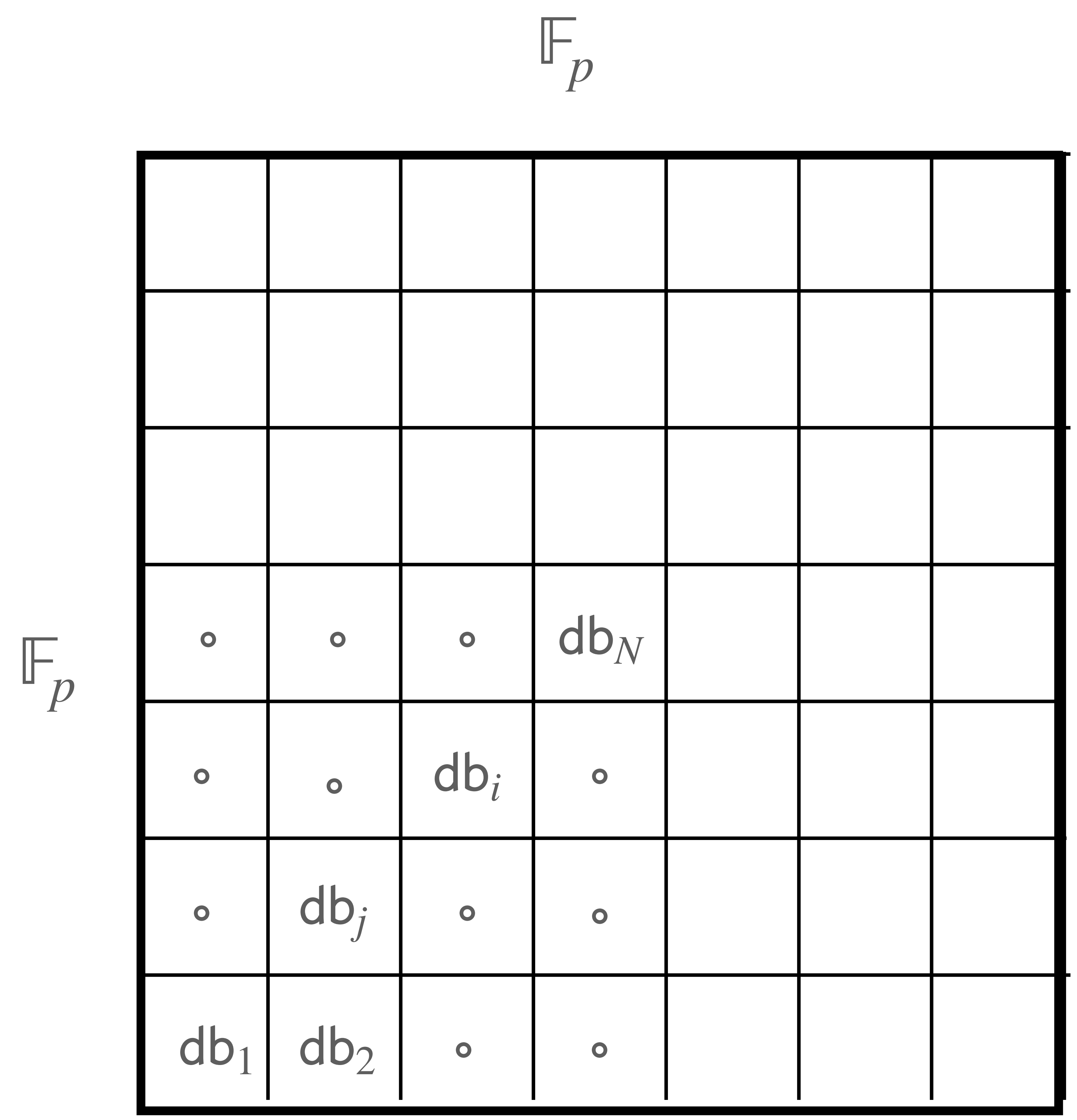
Encoding



The Reed-Muller (RM) LDC

Encoding

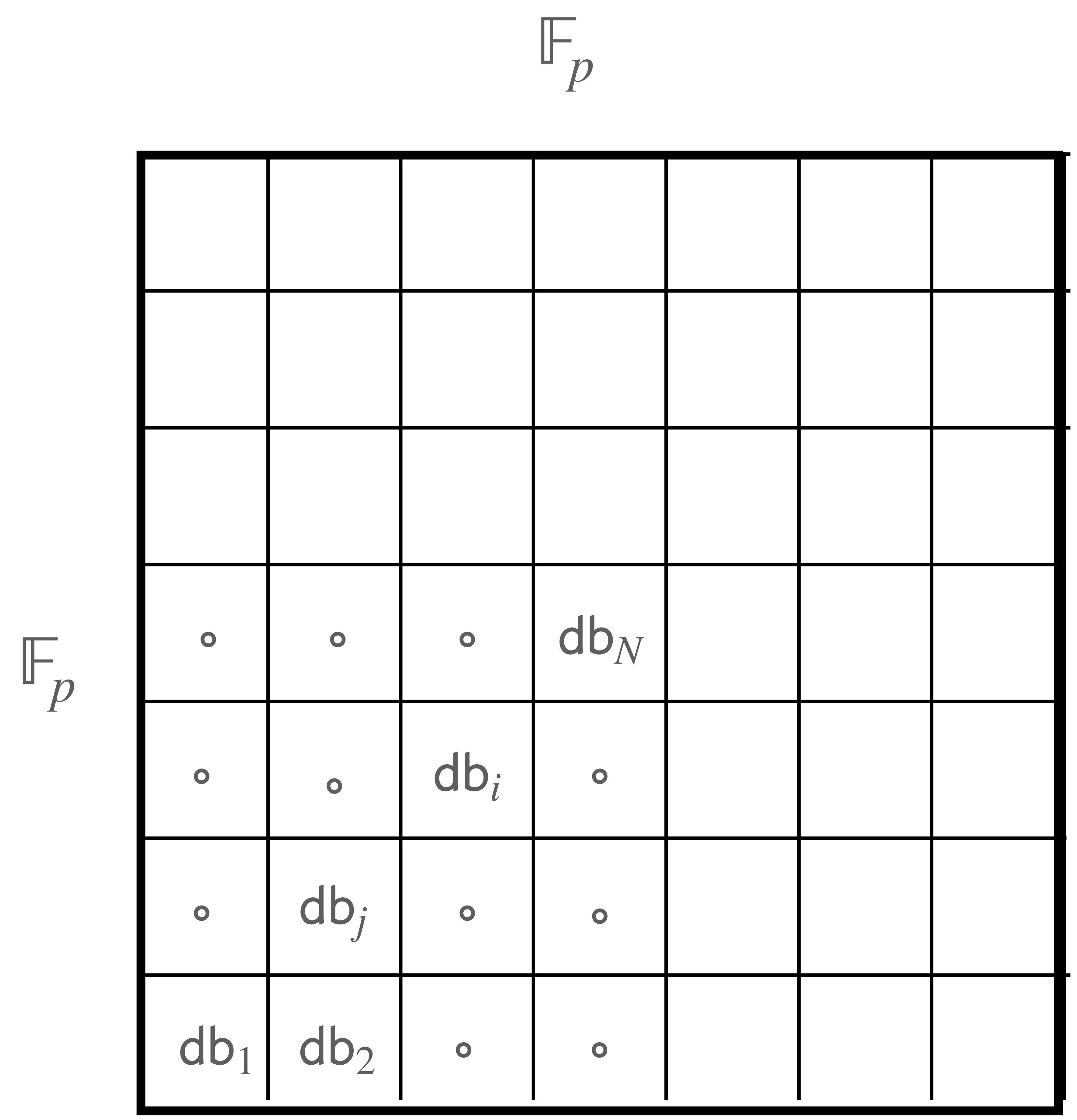
1. Interpolate a bivariate polynomial $f(X, Y)$ of total degree $d < p = O(\sqrt{N})$ that agrees with db.



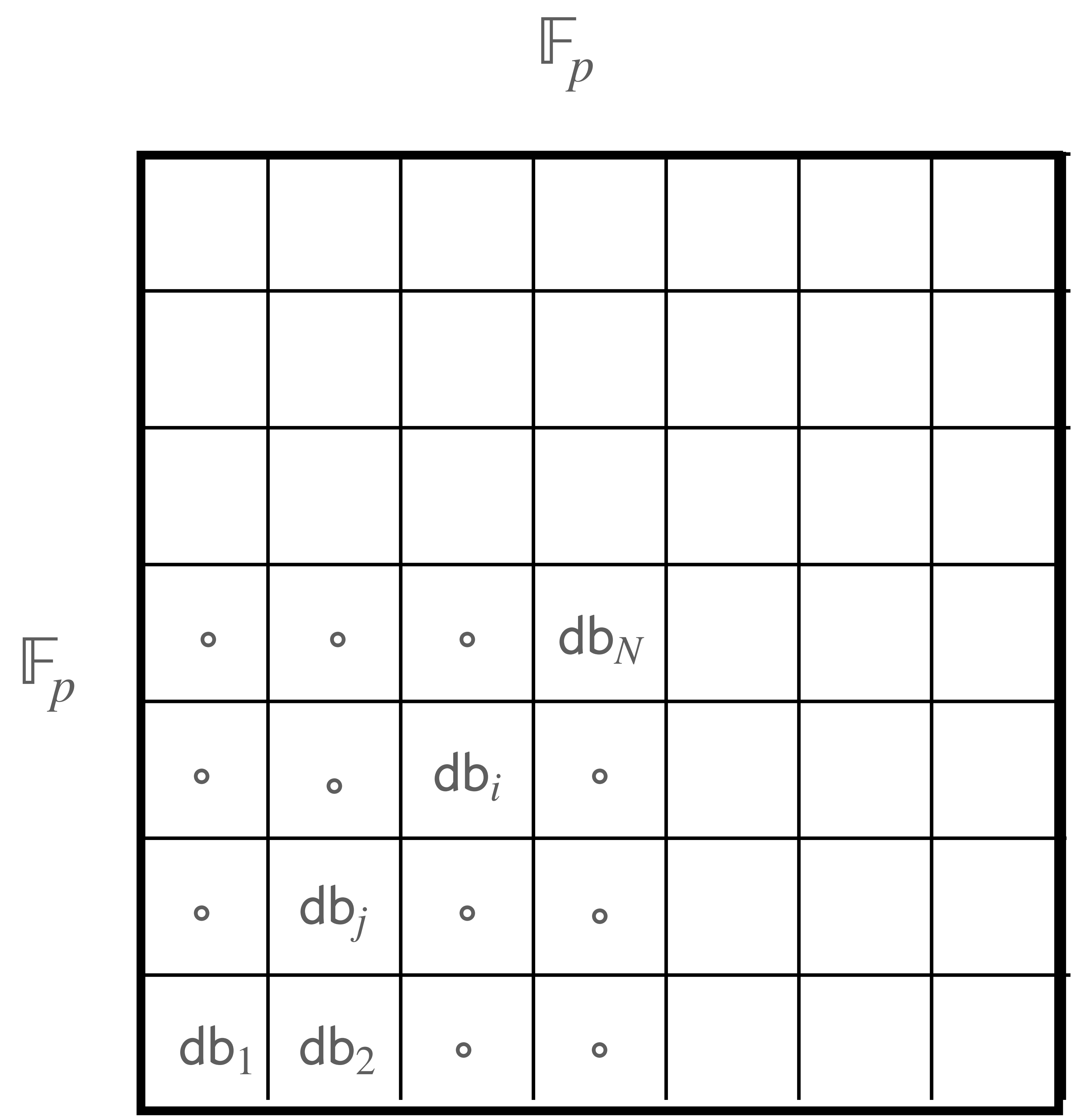
The Reed-Muller (RM) LDC

Encoding

1. Interpolate a bivariate polynomial $f(X, Y)$ of total degree $d < p = O(\sqrt{N})$ that agrees with db.
2. The codeword E is the evaluations of $f(x, y)$ for all $x, y \in \mathbb{F}_p \times \mathbb{F}_p$

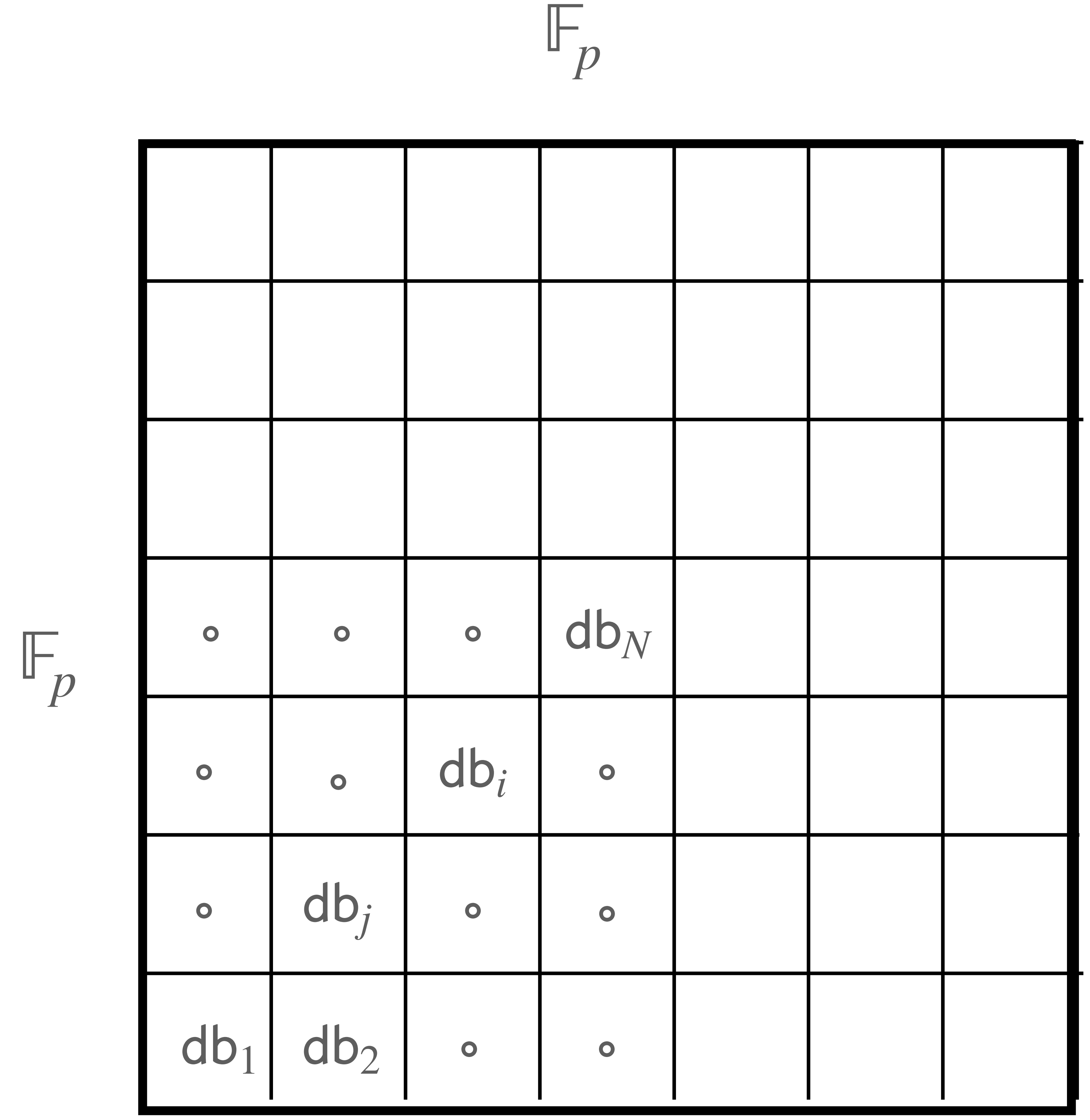


The Reed-Muller (RM) LDC



The Reed-Muller (RM) LDC

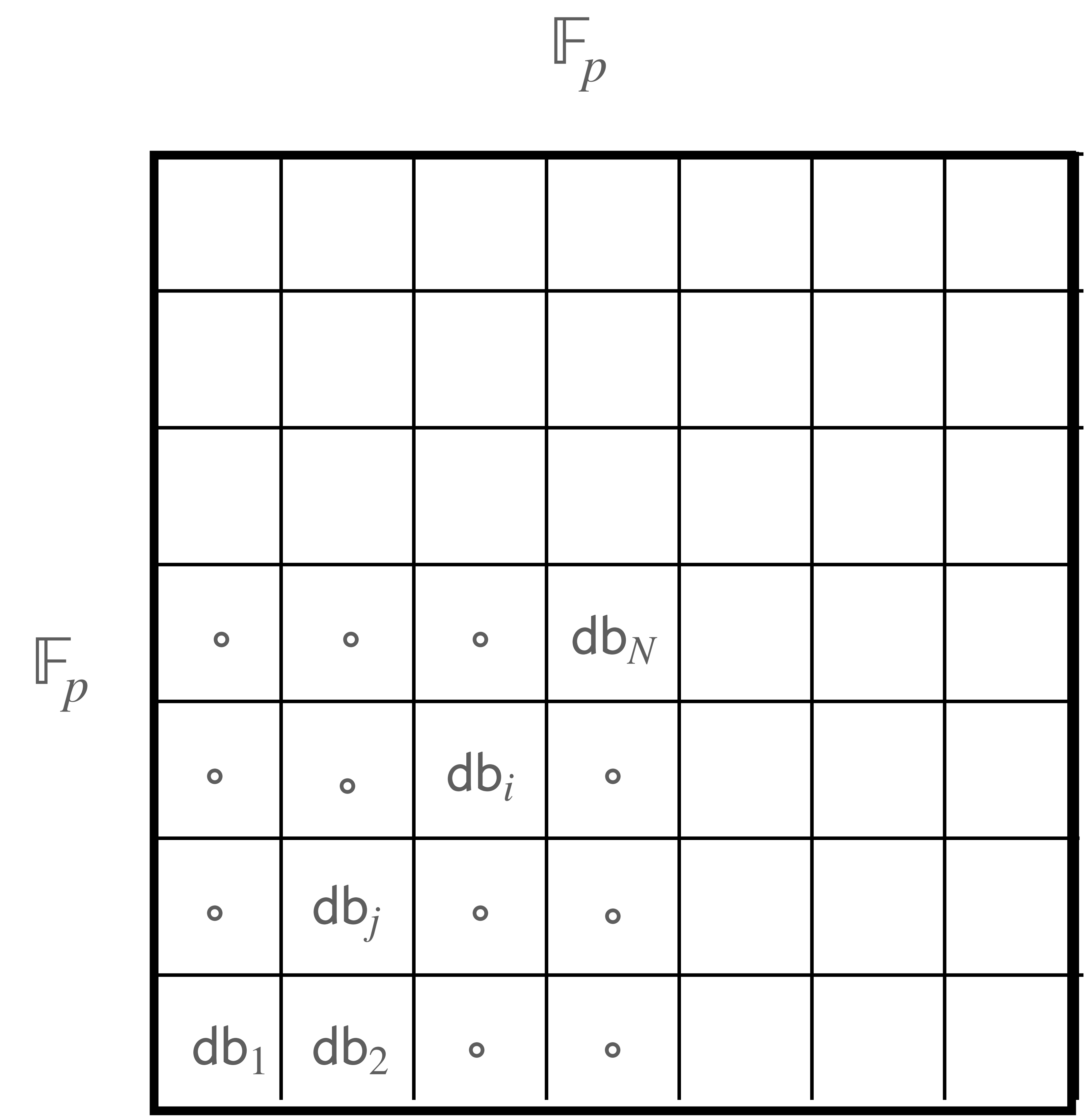
Local decoding



The Reed-Muller (RM) LDC

Local decoding

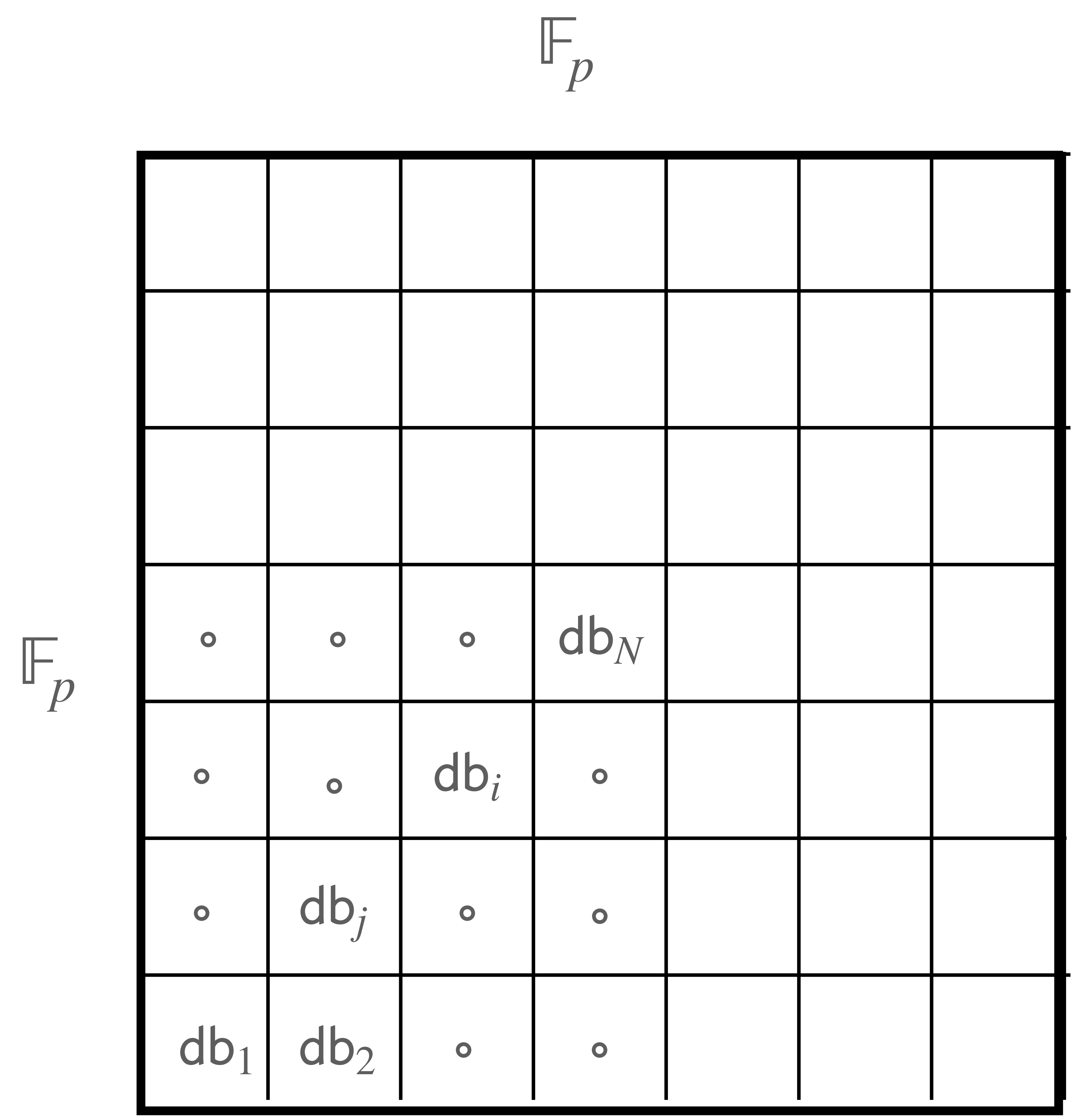
1. Want: db_j



The Reed-Muller (RM) LDC

Local decoding

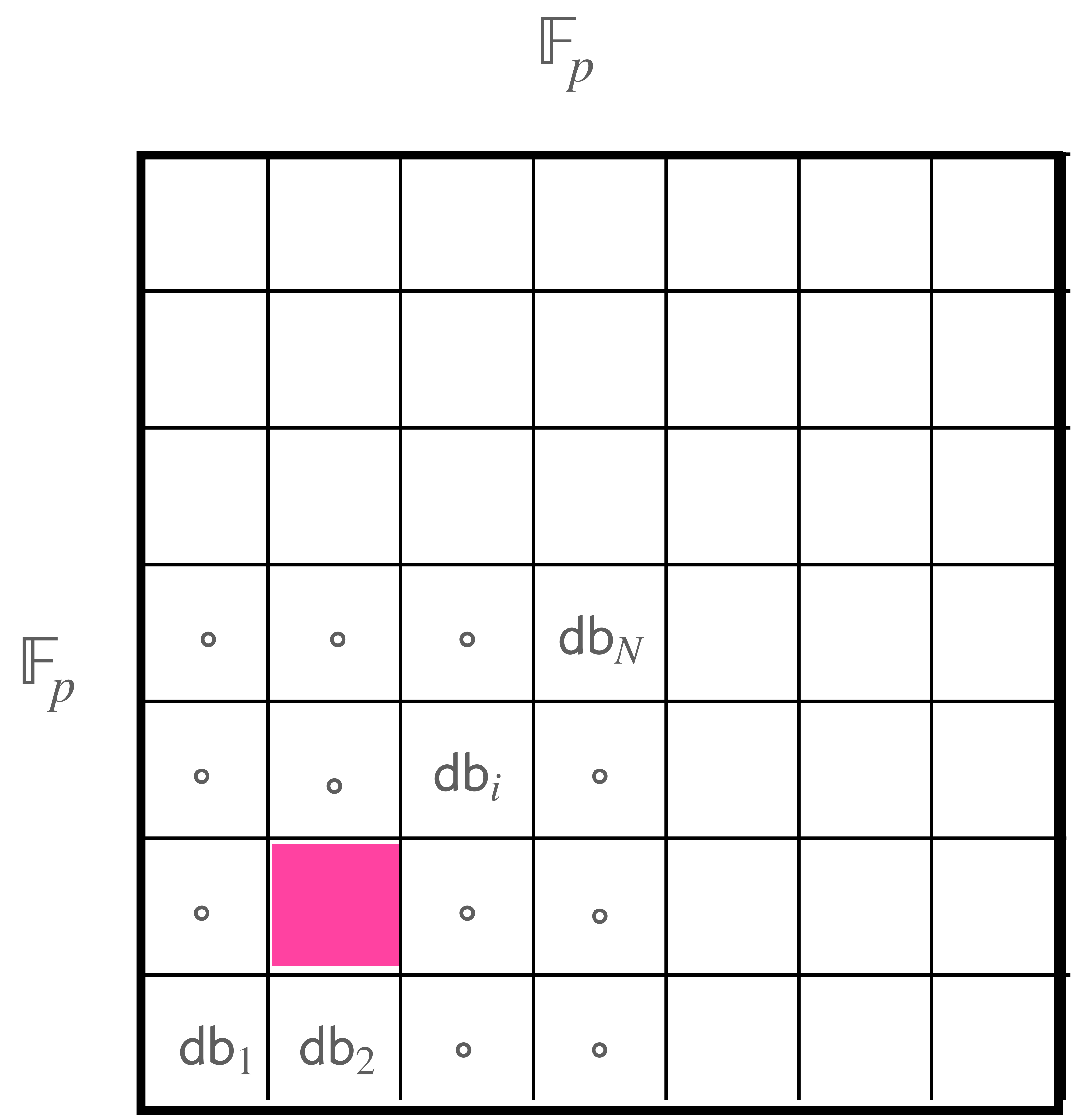
1. Want: db_j
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$: let Q be a random line through db_j.



The Reed-Muller (RM) LDC

Local decoding

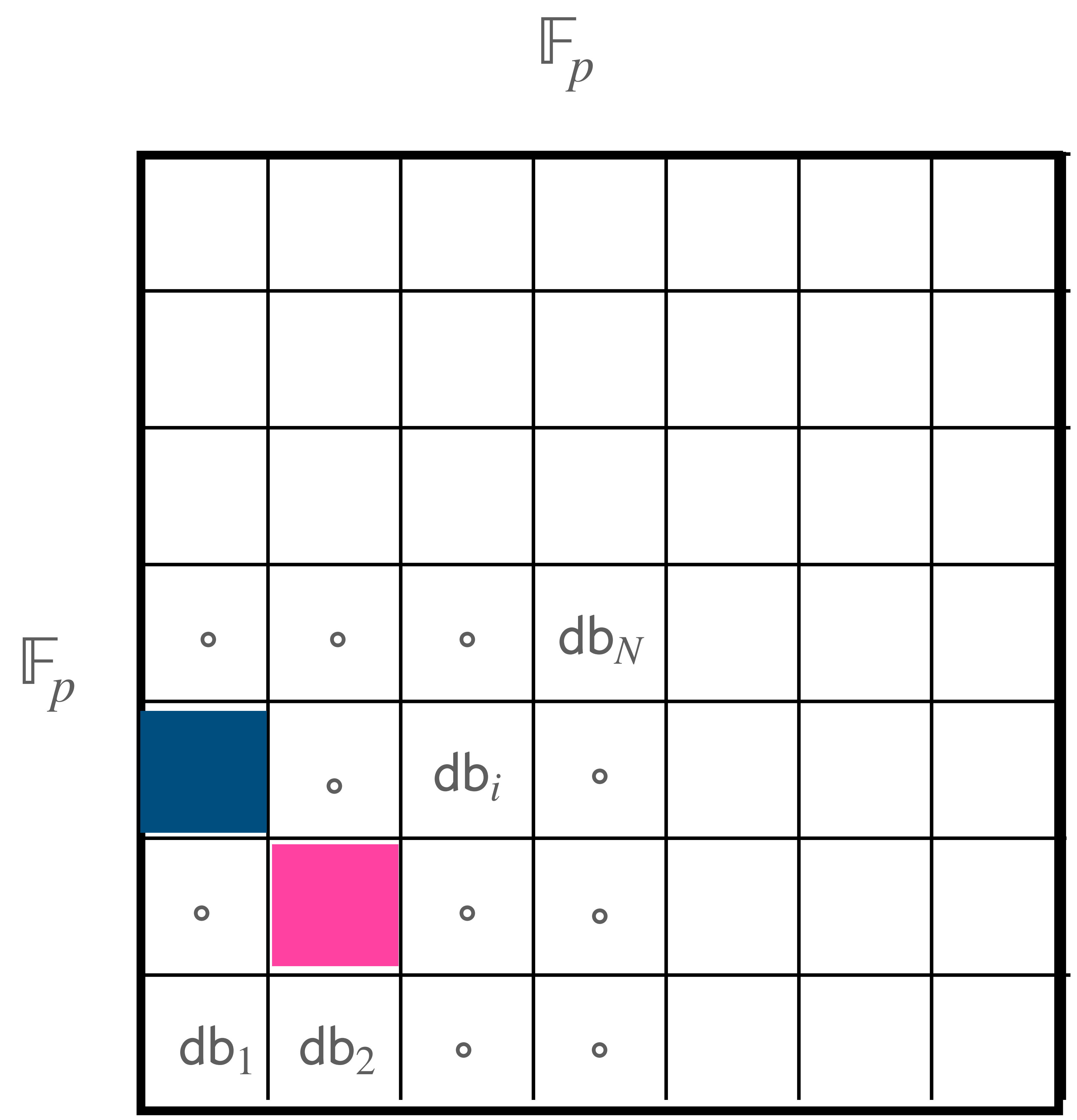
1. Want: db_j
2. $\mathsf{RM}.\mathrm{Que}(j) \rightarrow Q$: let Q be a random line through db_j.



The Reed-Muller (RM) LDC

Local decoding

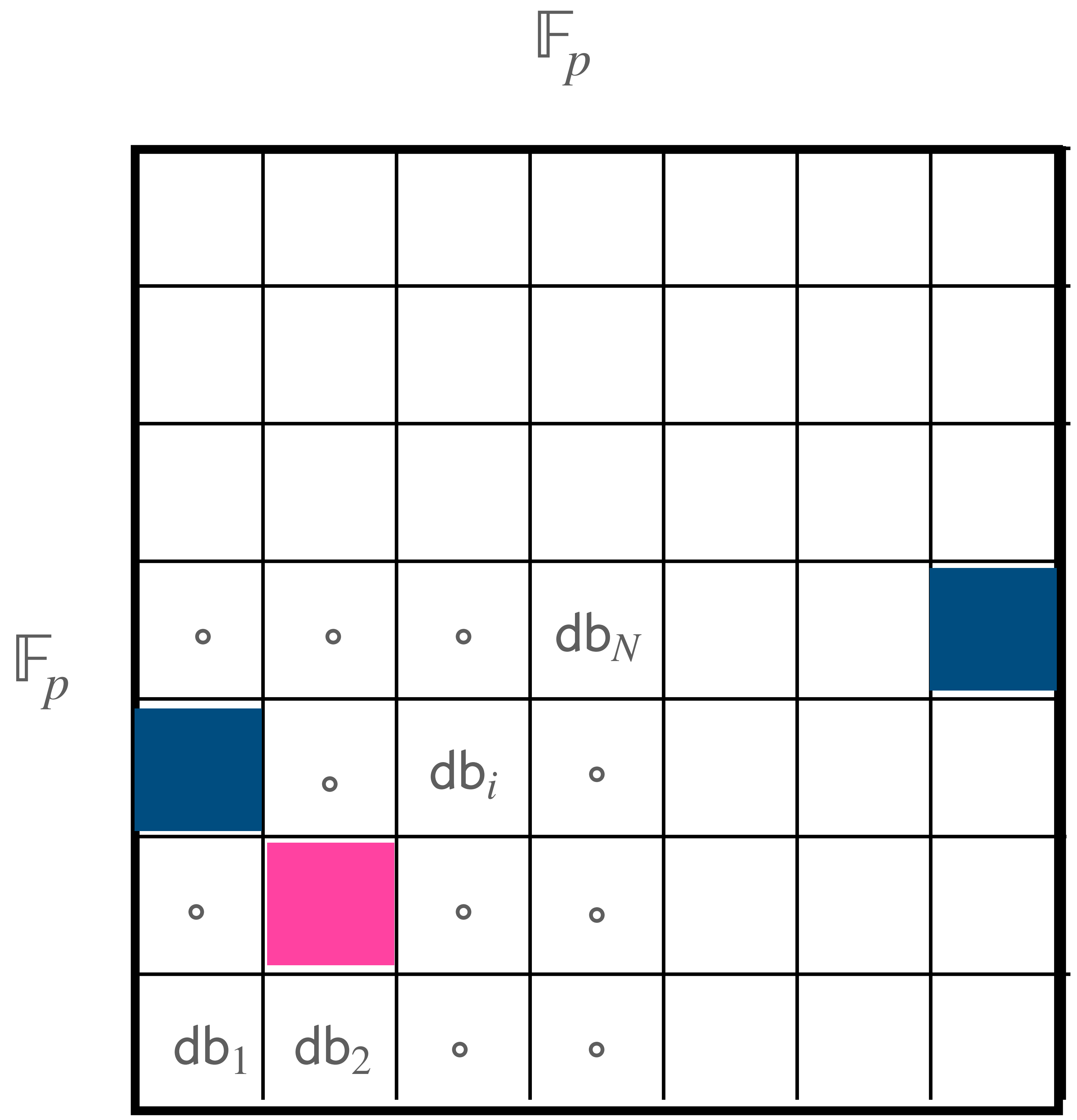
1. Want: db_j
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$: let Q be a random line through db_j.



The Reed-Muller (RM) LDC

Local decoding

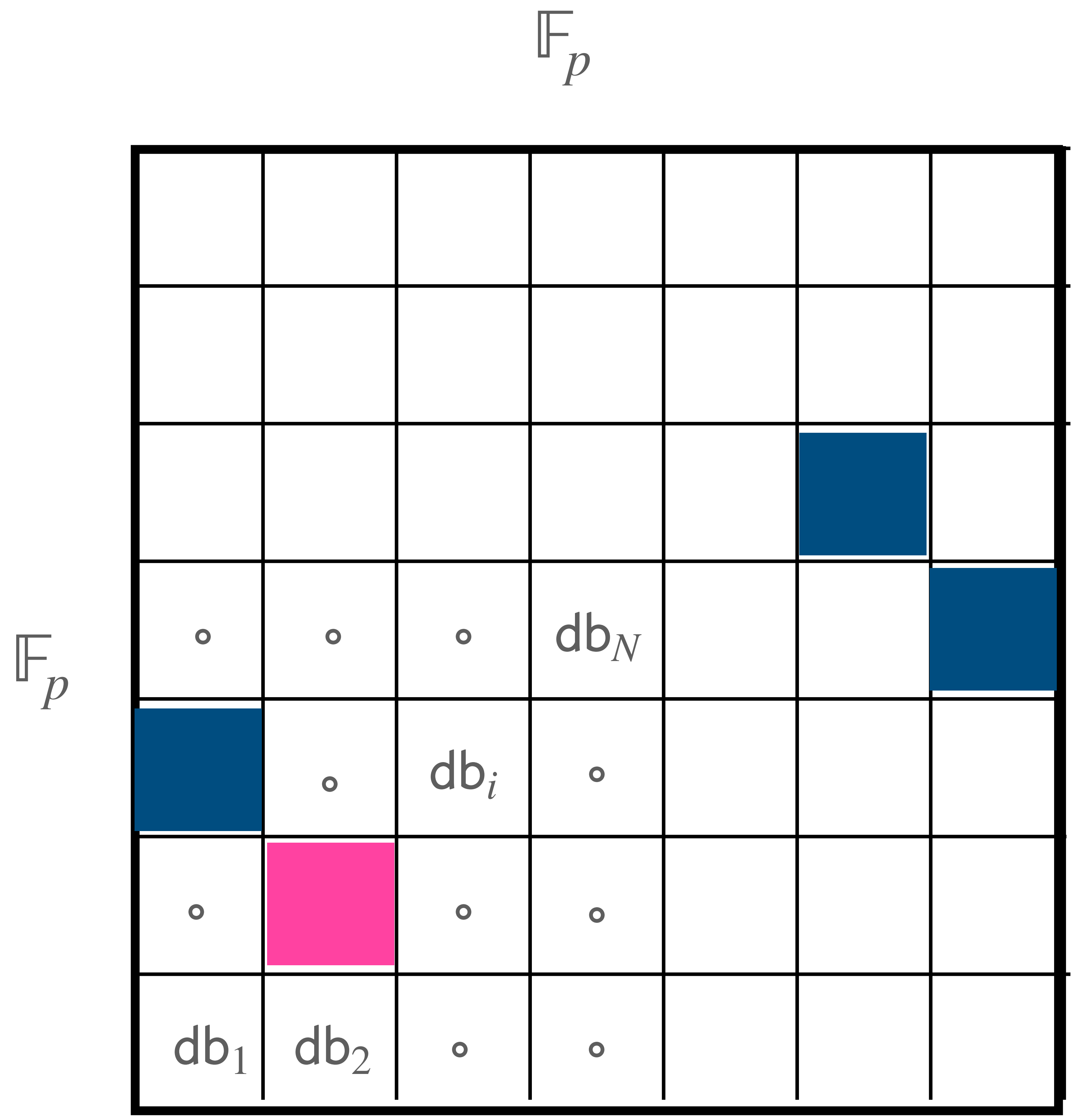
1. Want: db_j
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$: let Q be a random line through db_j.



The Reed-Muller (RM) LDC

Local decoding

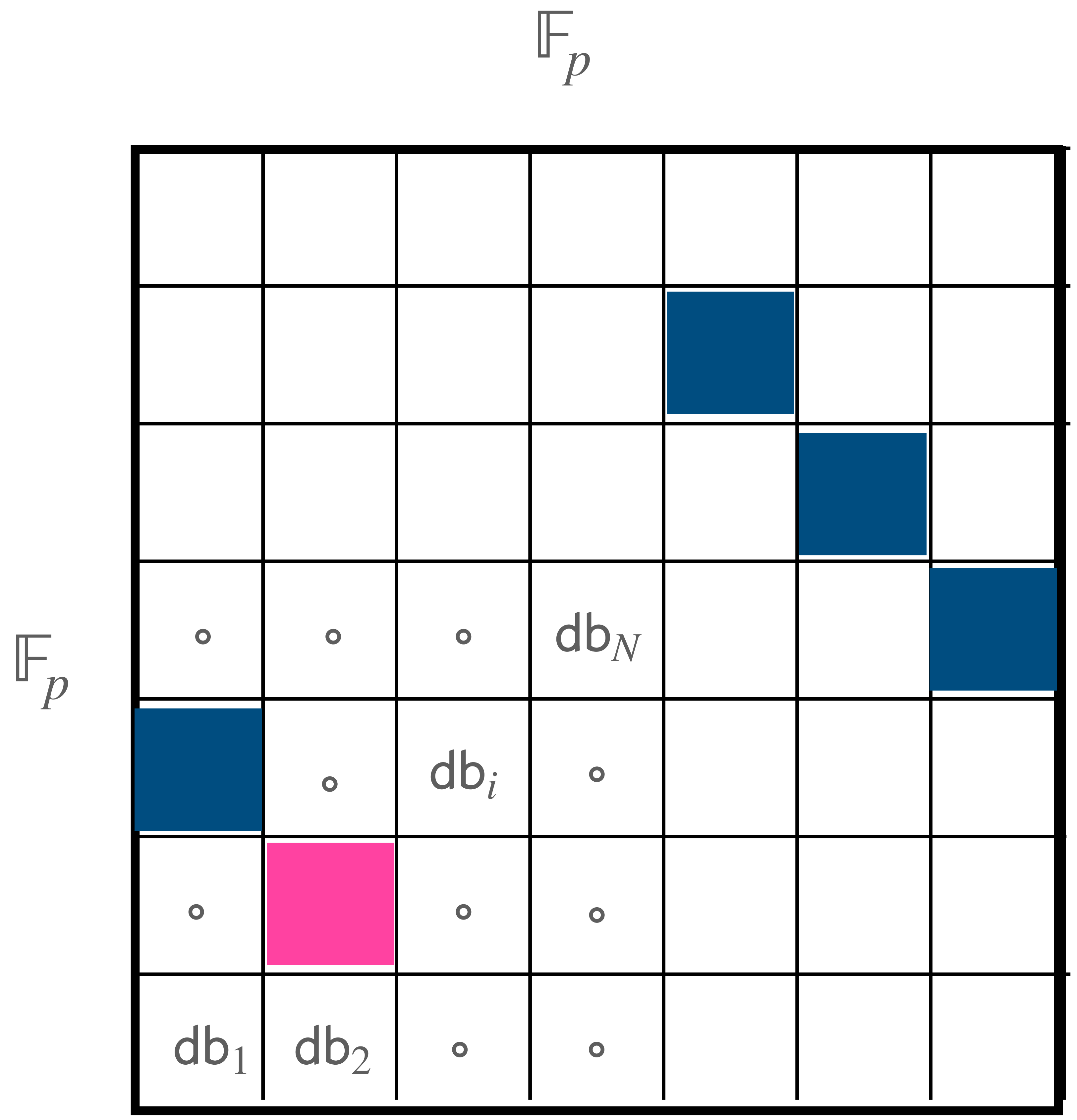
1. Want: db_j
2. $\mathrm{RM.Que}(j) \to Q$: let Q be a random line through db_j.



The Reed-Muller (RM) LDC

Local decoding

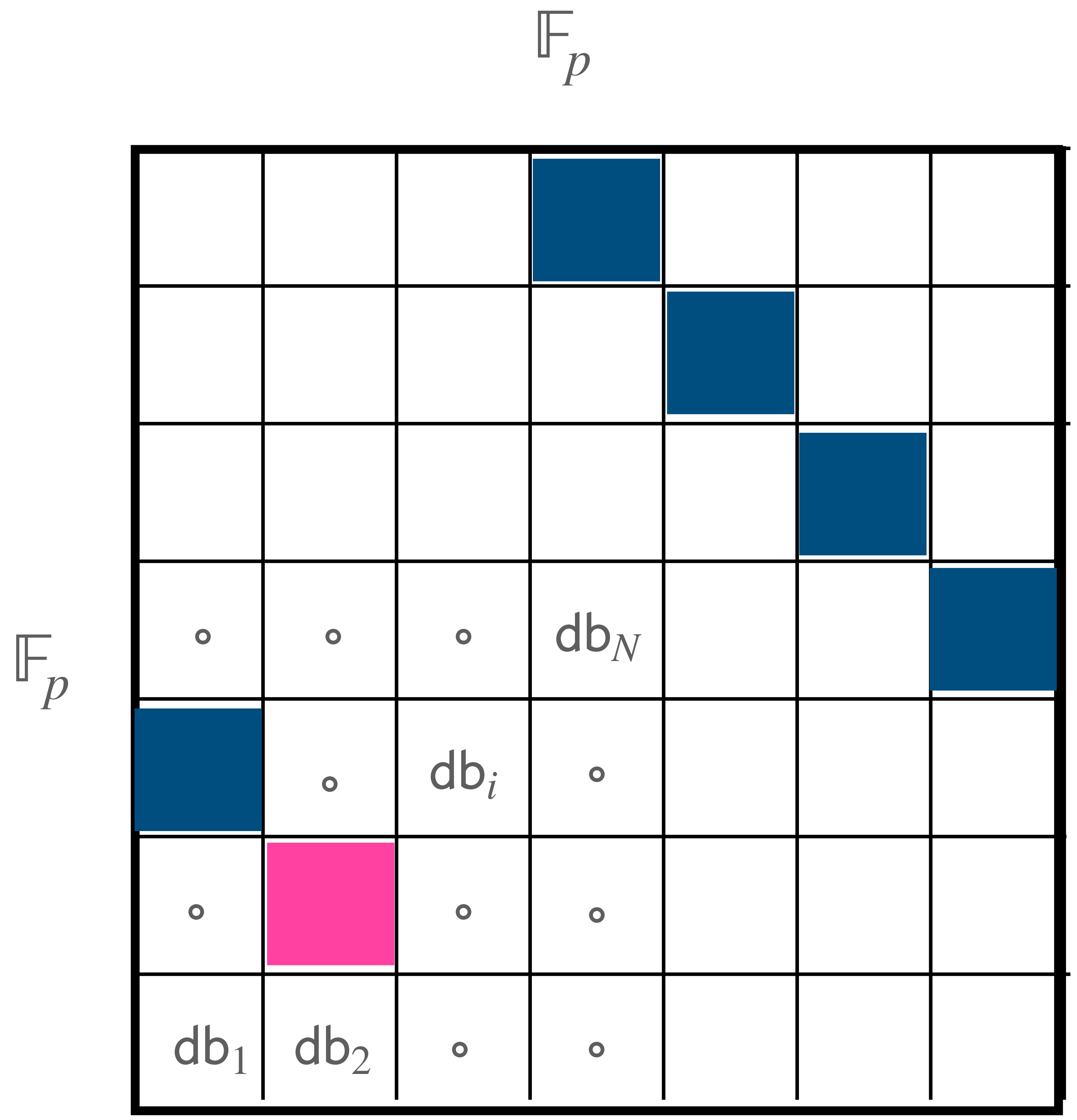
1. Want: db_j
2. $\mathsf{RM.Que}(j) \to Q$: let Q be a random line through db_j.



The Reed-Muller (RM) LDC

Local decoding

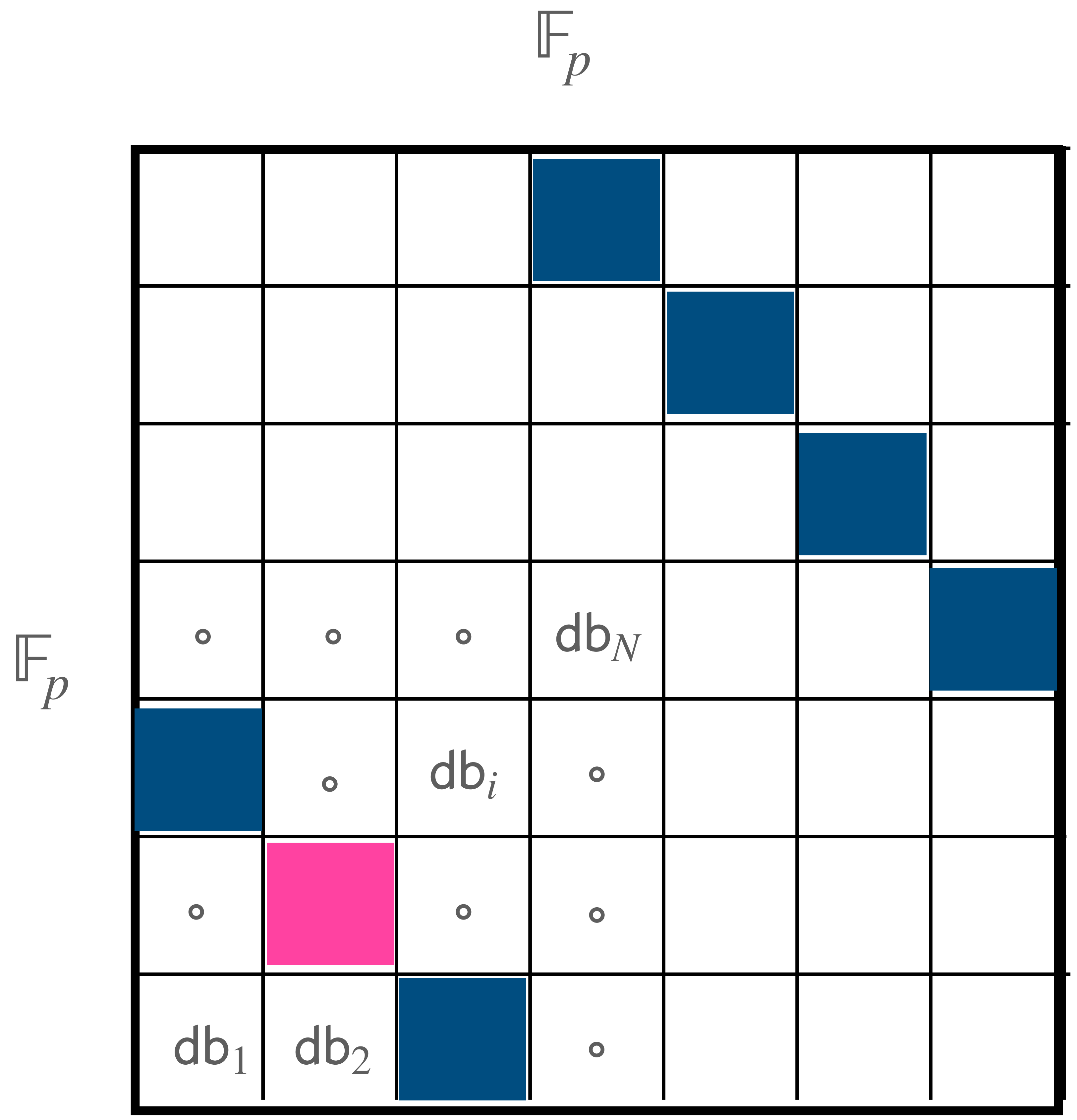
1. Want: db_j
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$: let Q be a random line through db_j.



The Reed-Muller (RM) LDC

Local decoding

1. Want: db_j
2. $\mathsf{RM.Que}(j) \to Q$: let Q be a random line through db_j.

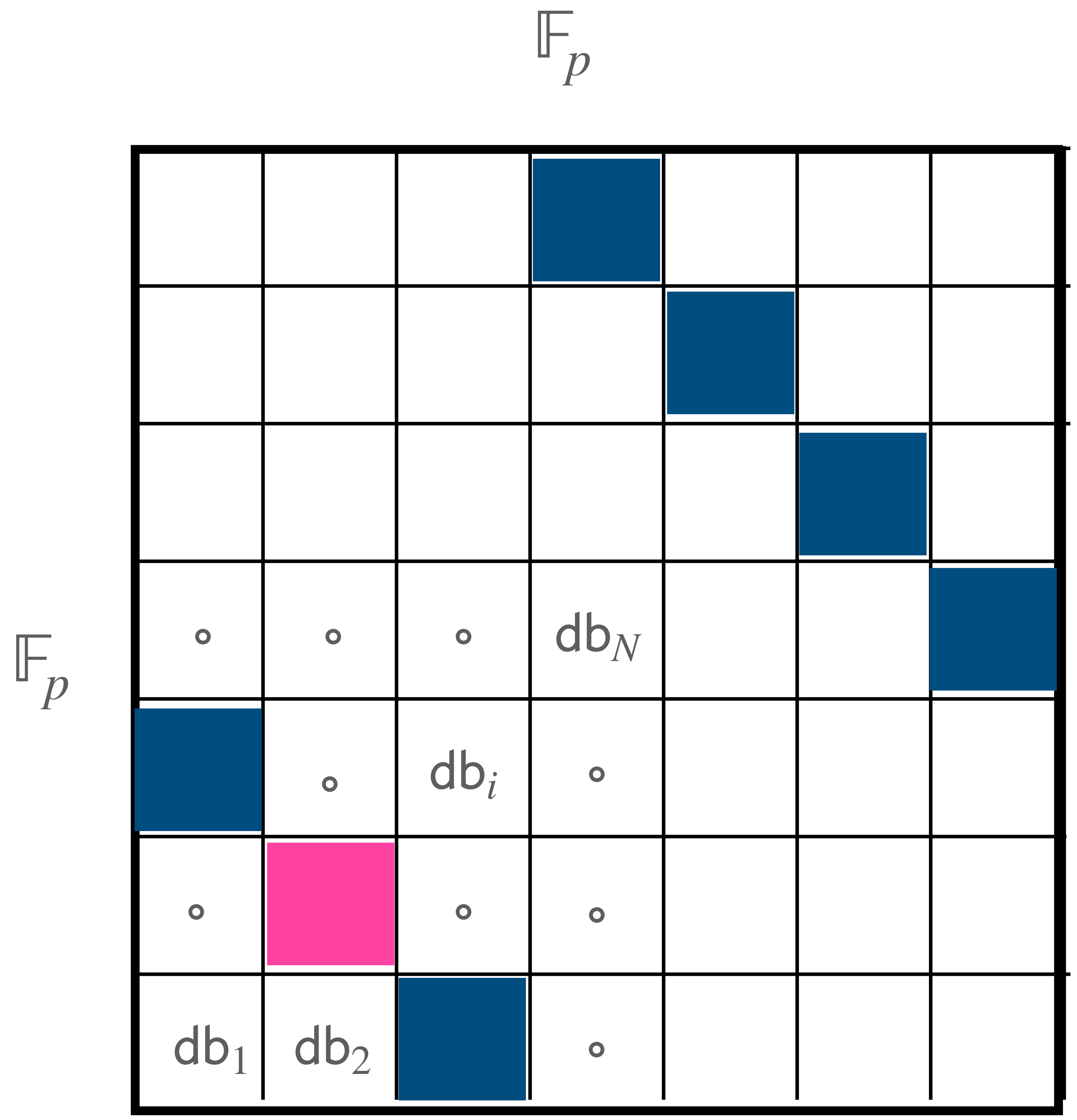


The Reed-Muller (RM) LDC

Local decoding

1. Want: db_j
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$: let Q be a random line through db_j.
3. $\mathsf{RM}.\mathsf{Dec}(E_Q) \to \mathsf{db}_j$: E_Q is a univariate polynomial. Can retrieve db_j from E_Q.

Decoder reads only $p = O(N^{1/2})$ elements!



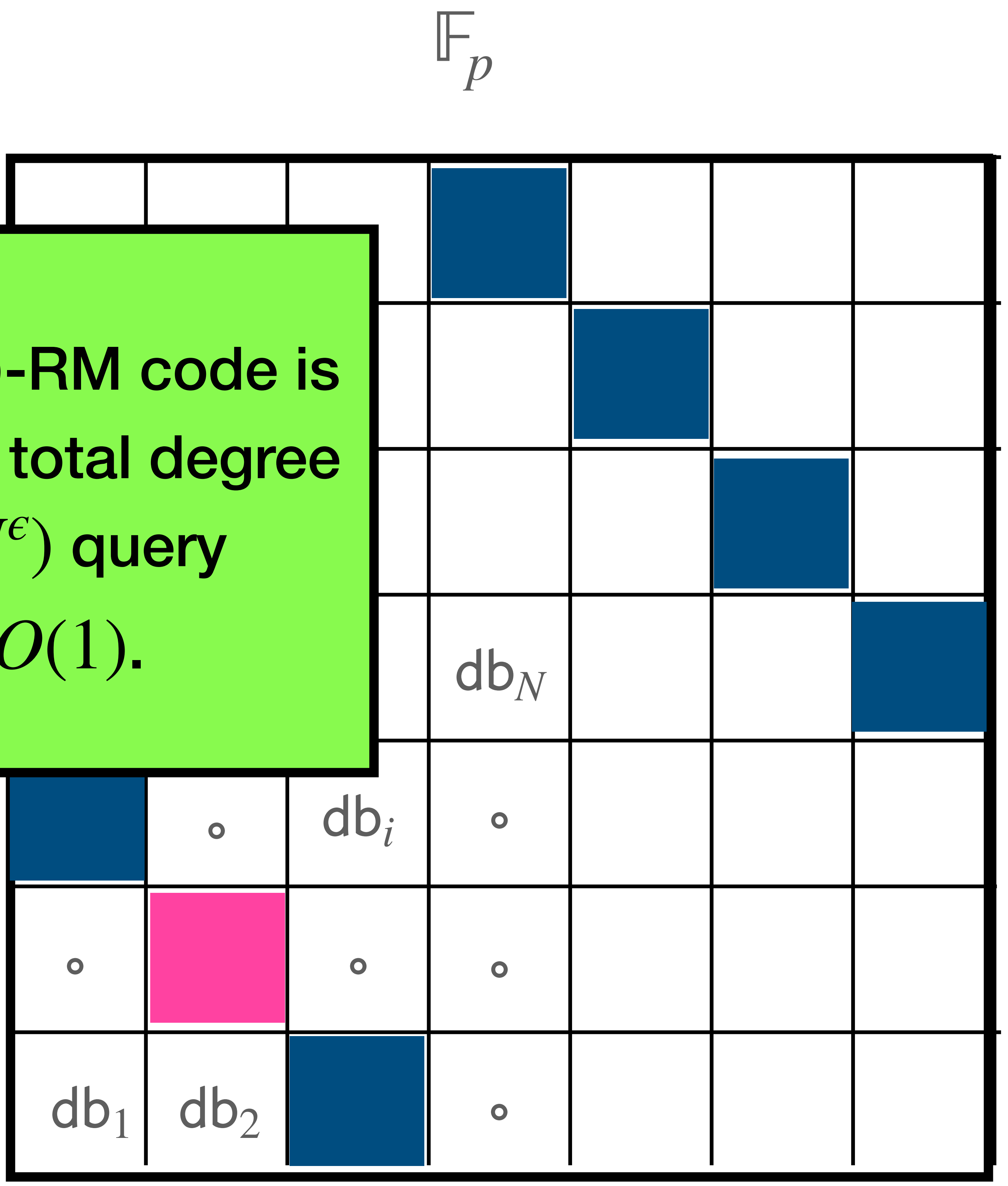
The Reed-Muller (RM) LDC

Local decoding

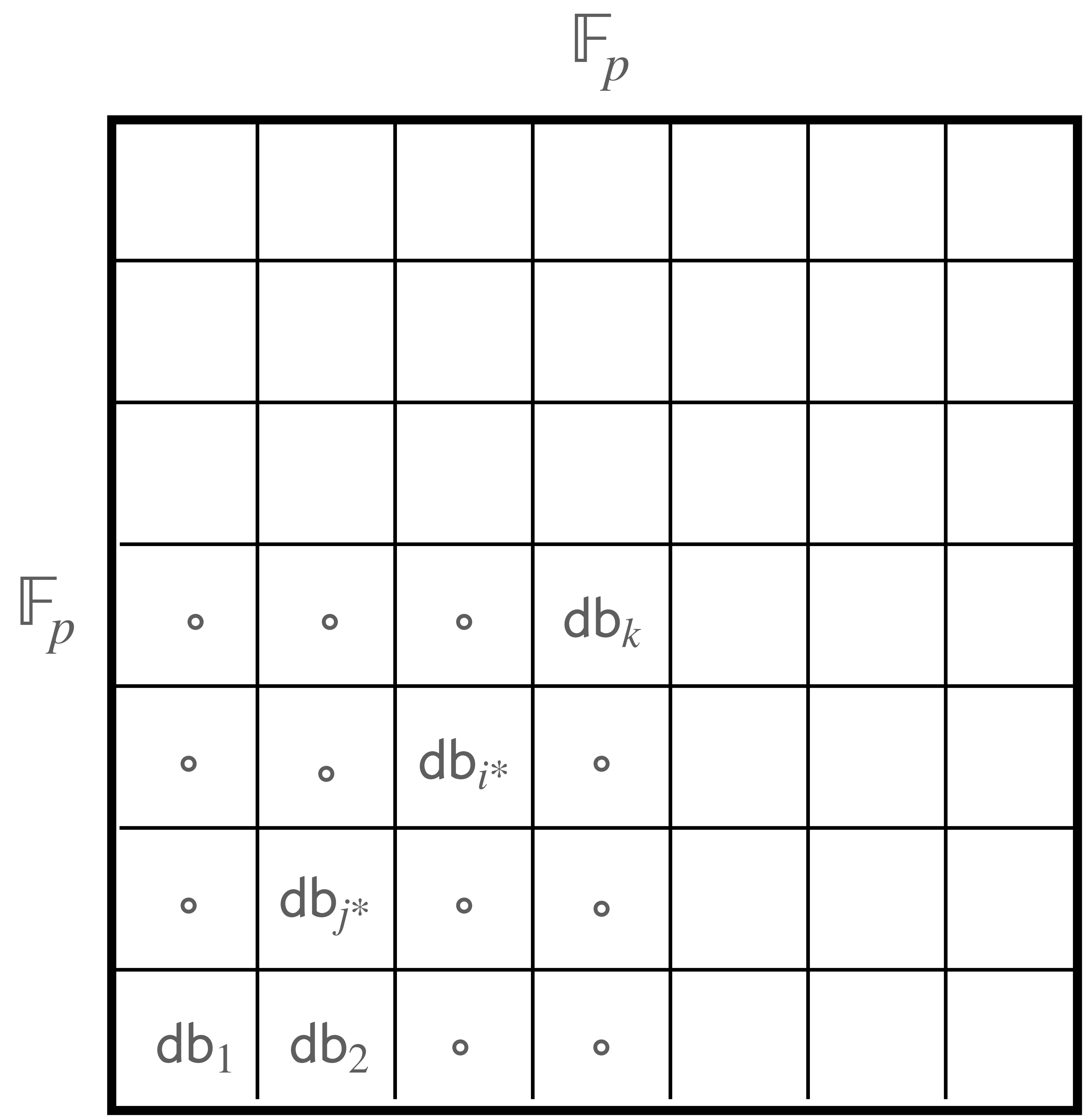
1. Want: db_j
2. $\mathsf{RM.Que}(j) \to Q$: let Q … random line through db_j
3. $\mathsf{RM.Dec}(E_Q) \to \mathsf{db}_j$: E_Q … univariate polynomial. C… retrieve db_j from E_Q.

Decoder reads only $p = O(N^{1/2})$ elements!

More generally (q, m, d)-RM code is m-variate polynomial of total degree d over $\mathbb{F}_q$. Has $O(N^\epsilon)$ query complexity, rate $O(1)$.

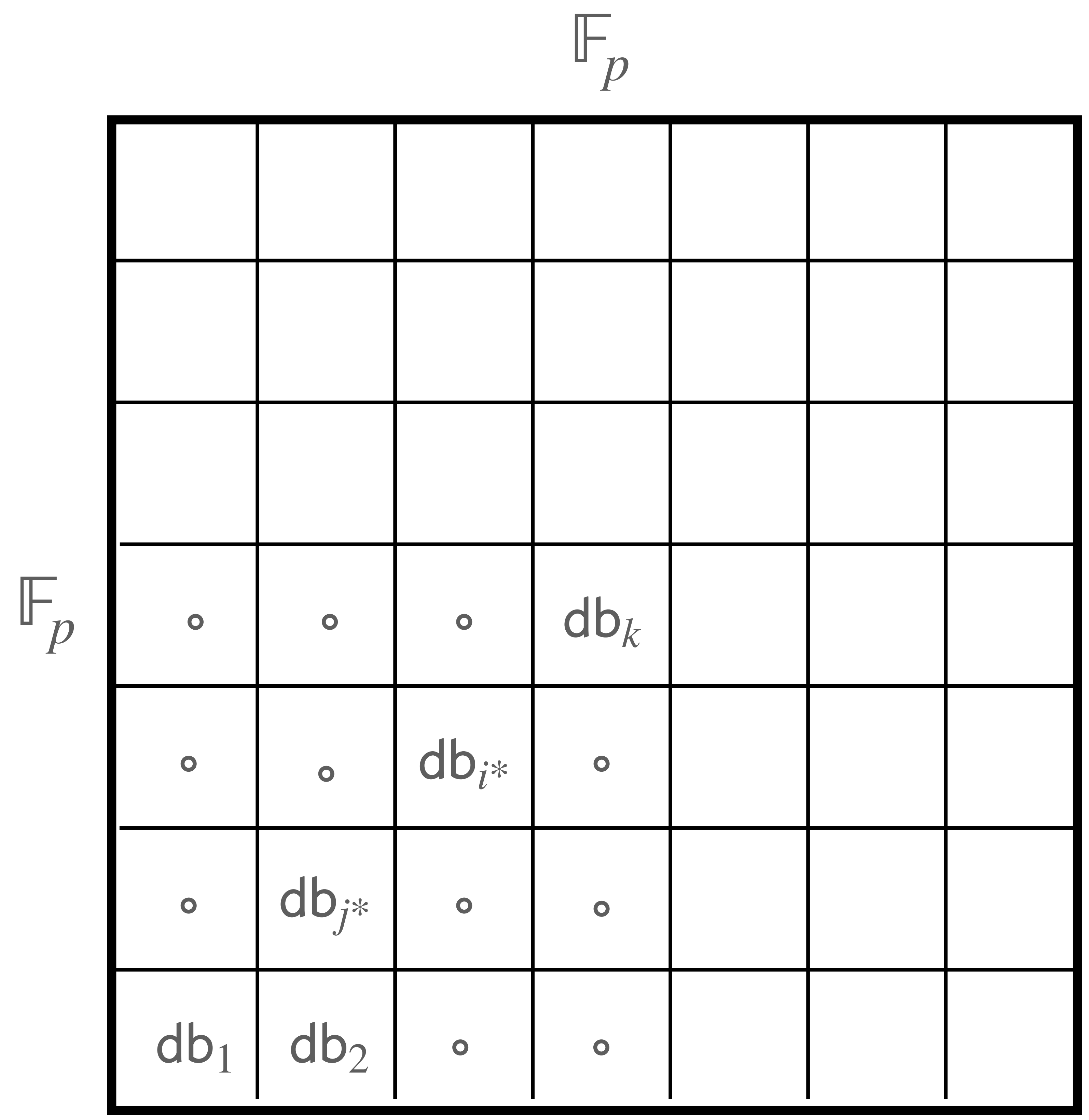


Analyzing privacy: “variance attack”



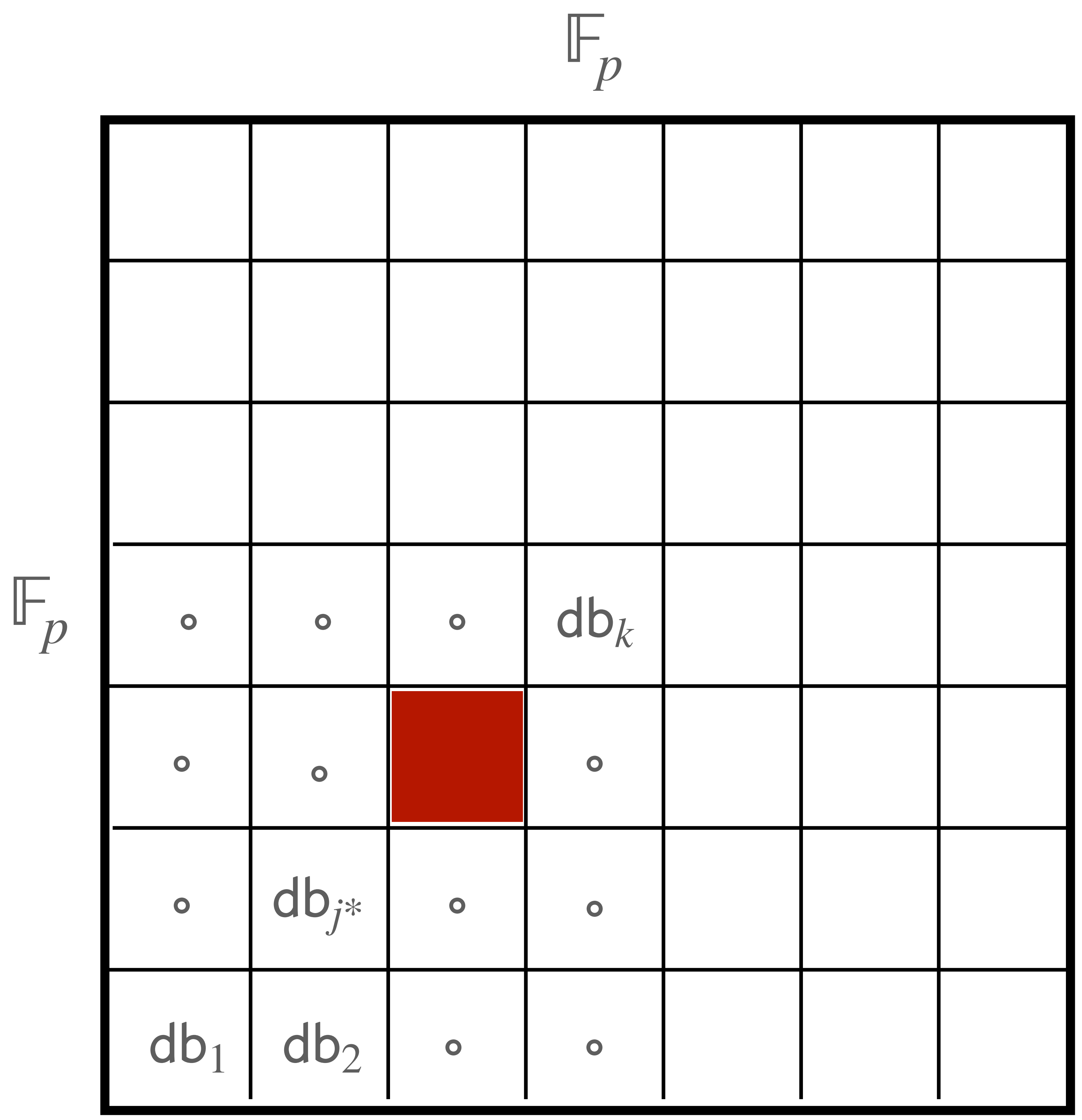
Analyzing privacy: “variance attack”

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:



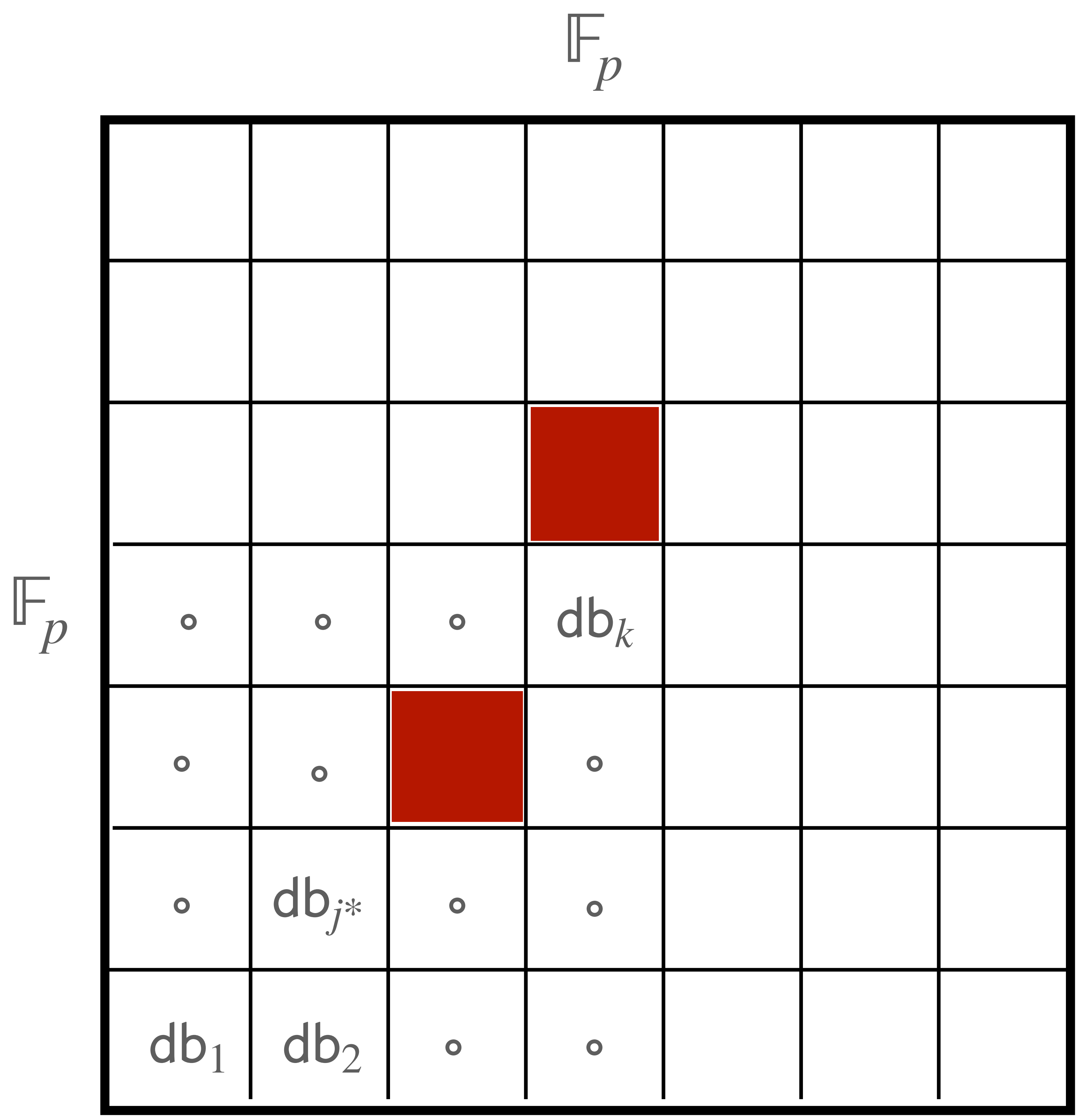
Analyzing privacy: “variance attack”

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:



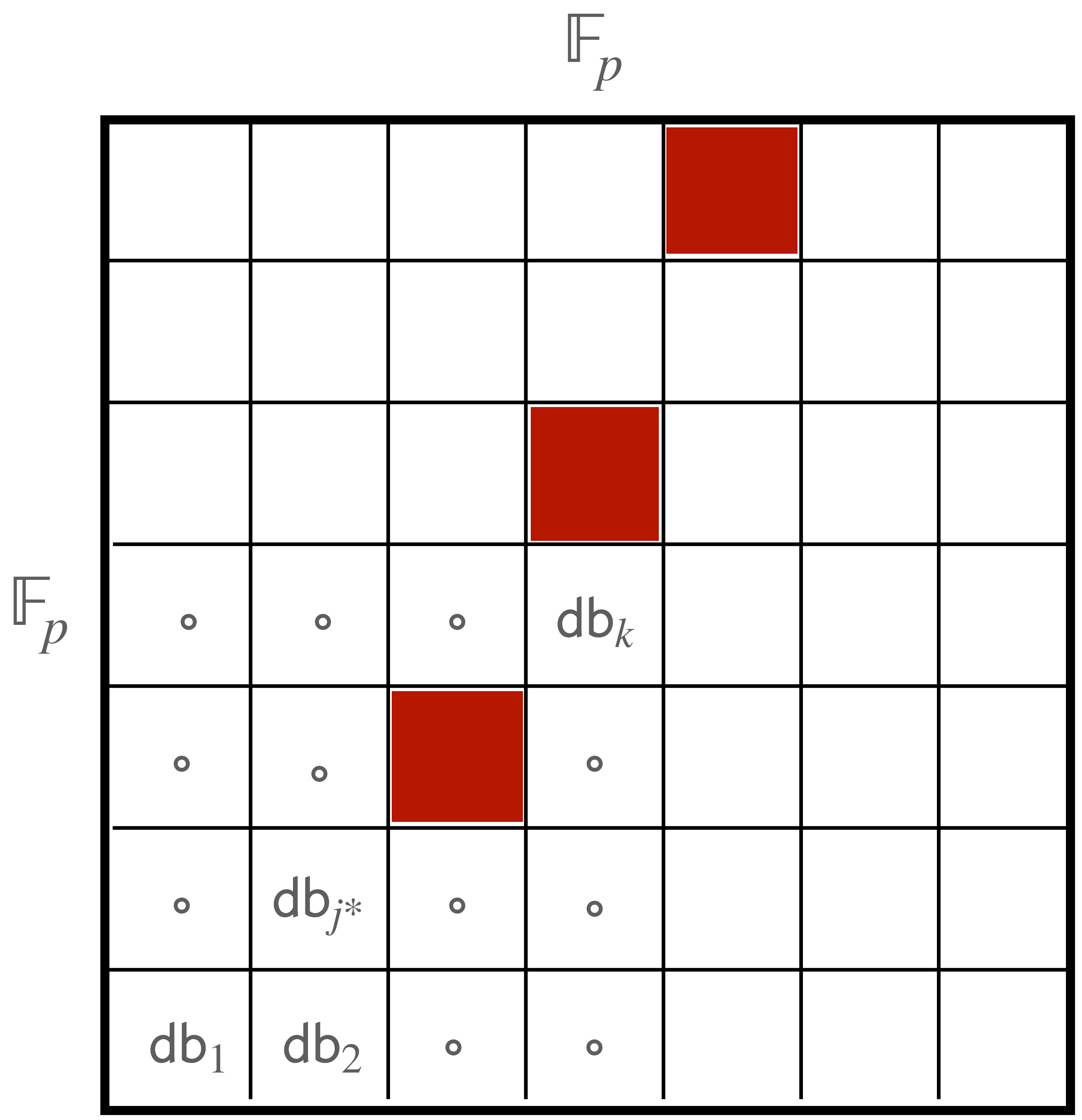
Analyzing privacy: “variance attack”

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:



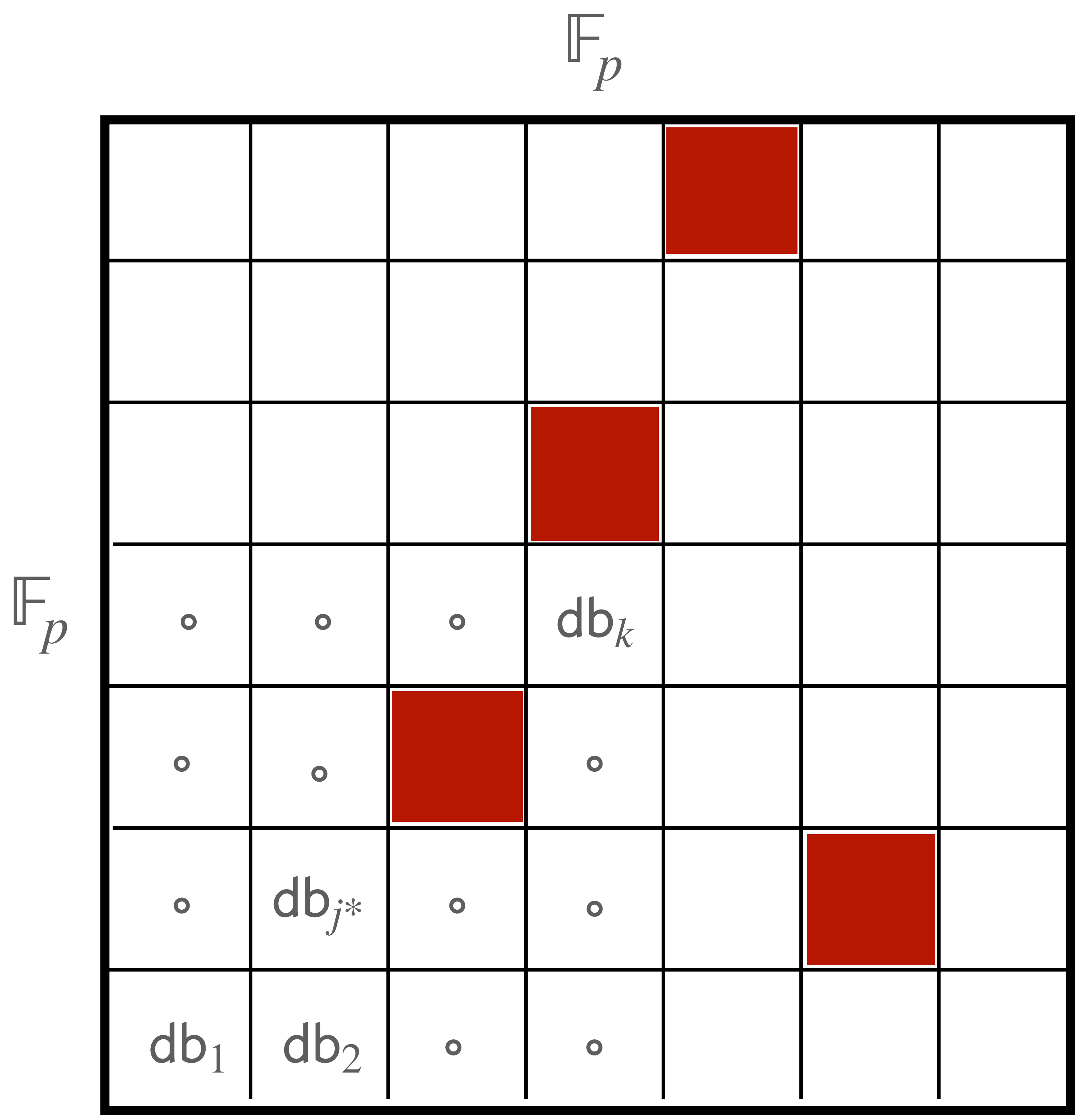
Analyzing privacy: “variance attack”

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:



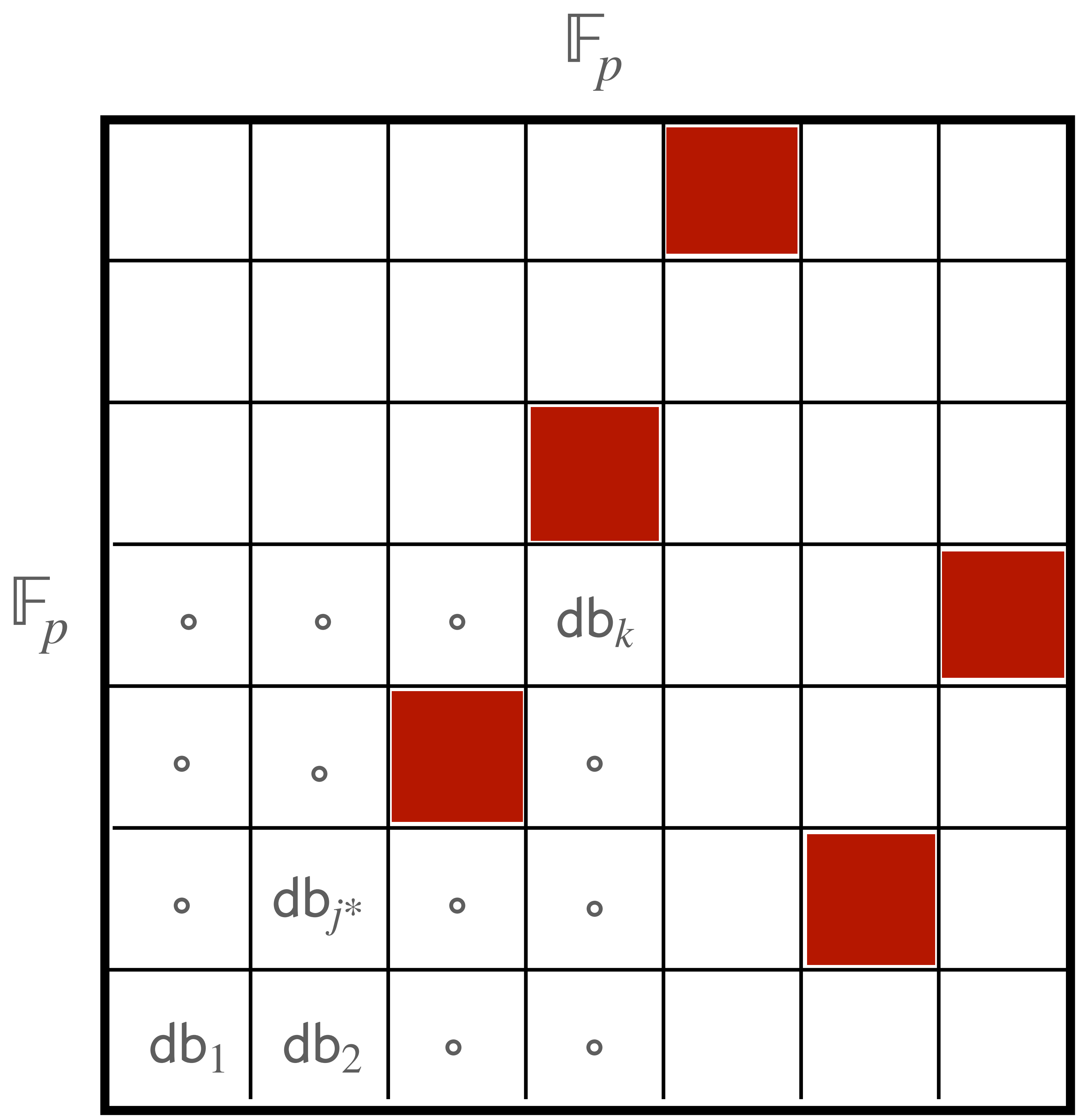
Analyzing privacy: “variance attack”

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:



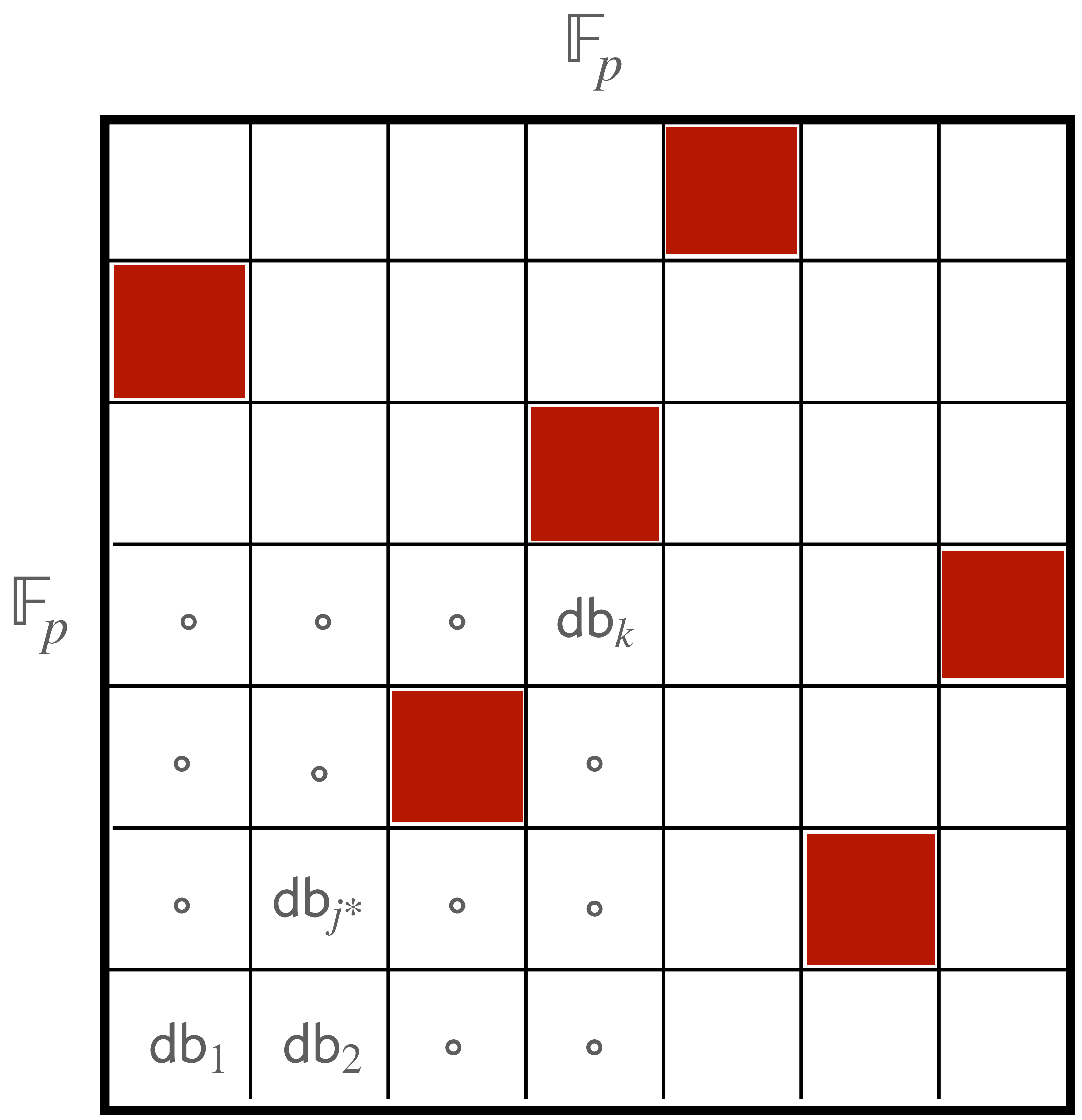
Analyzing privacy: “variance attack”

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:



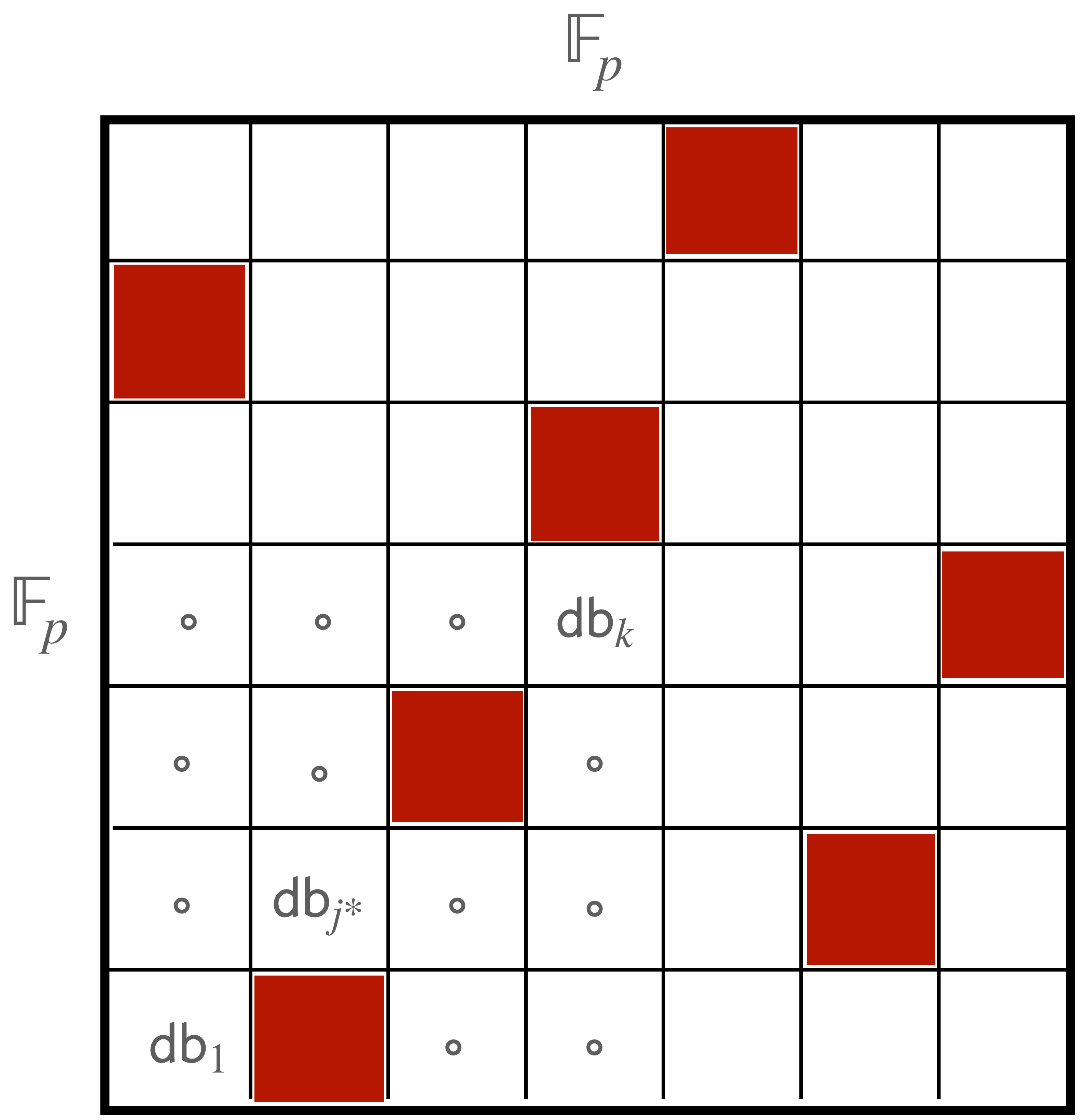
Analyzing privacy: "variance attack"

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:



Analyzing privacy: “variance attack”

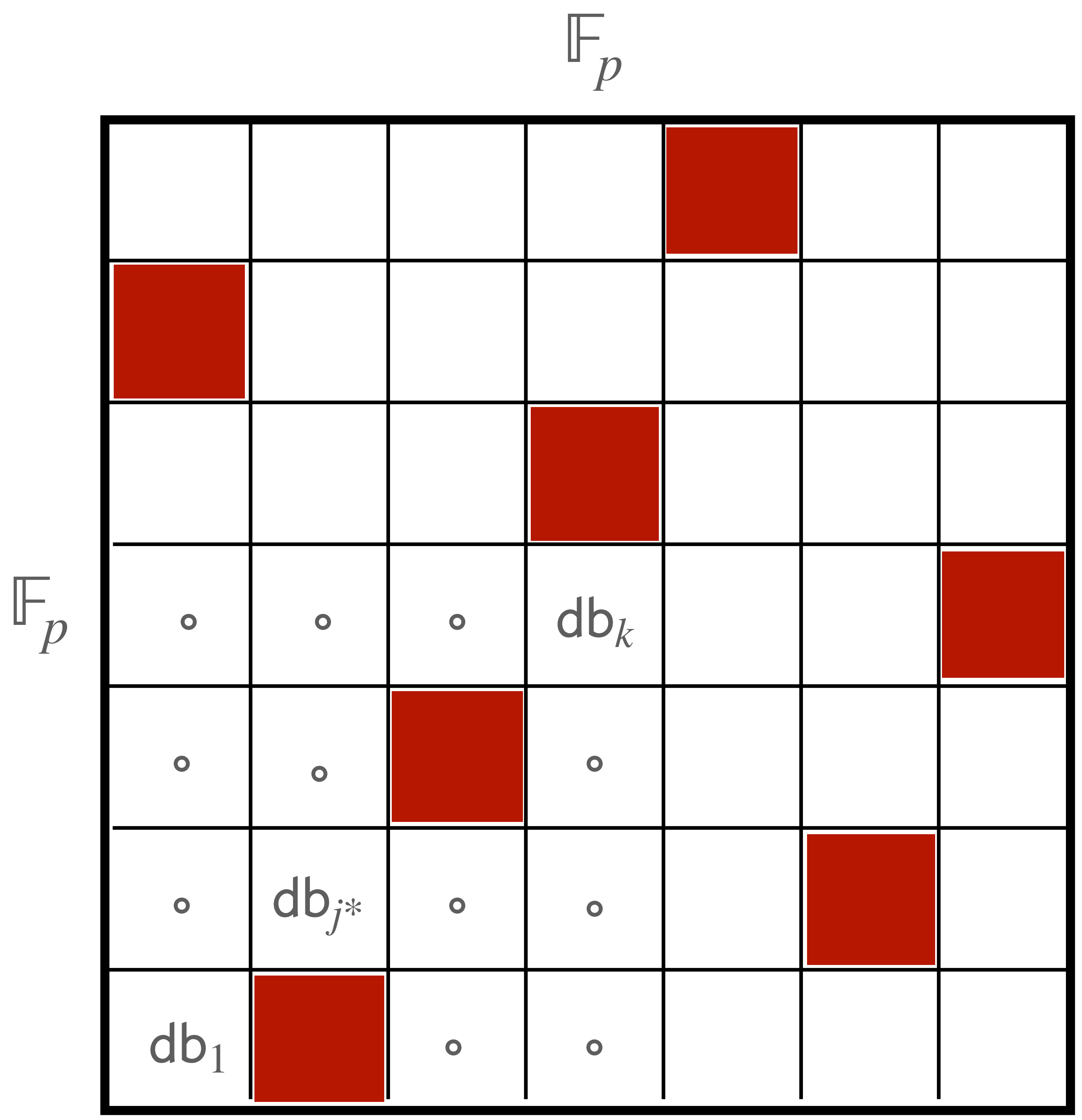
To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:



Analyzing privacy: "variance attack"

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:

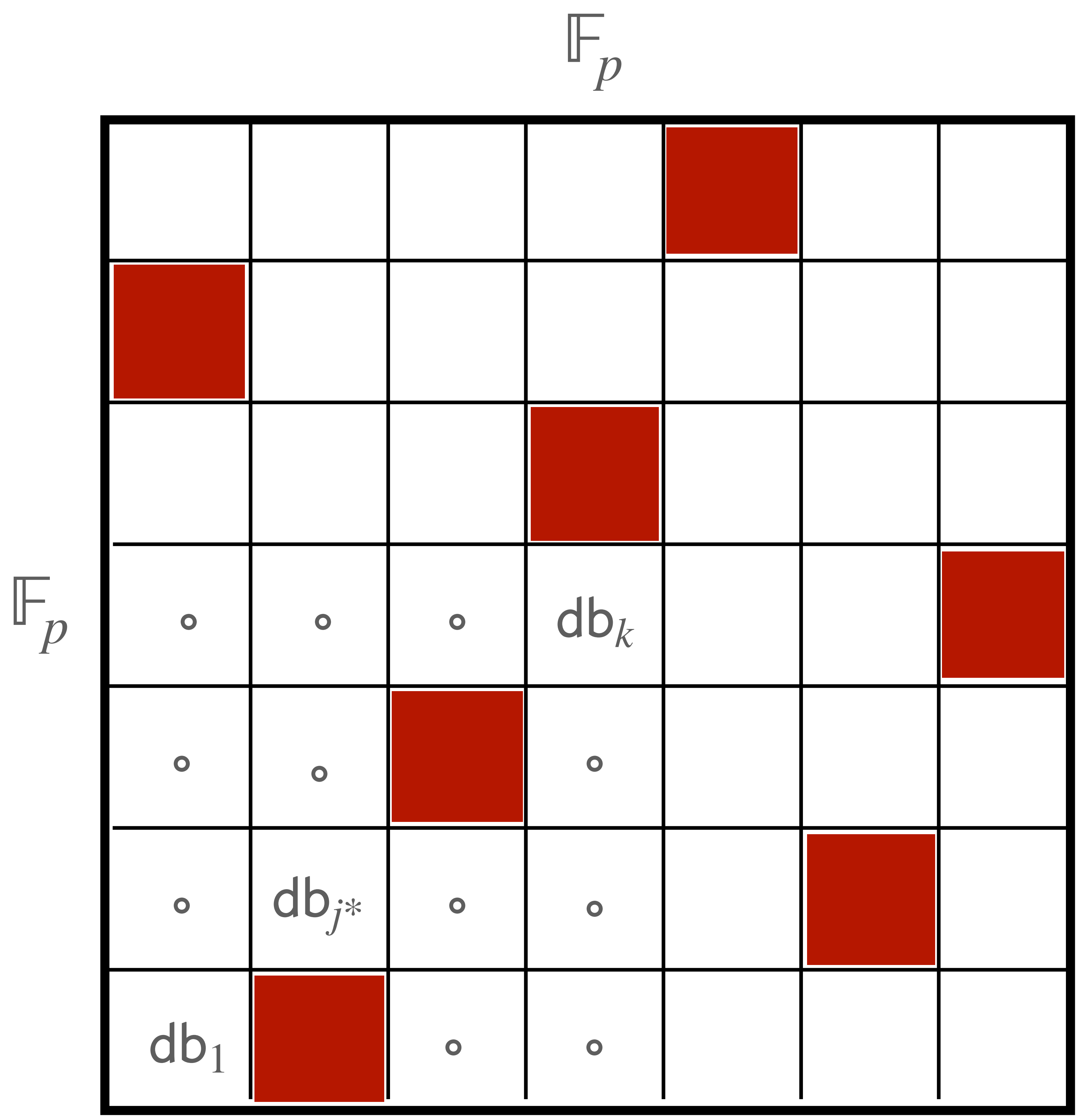
- Client queries for i: query line ℓ w/ prob. $1/\text{poly}(N)$ (there are poly-many lines) and aborts.



Analyzing privacy: “variance attack”

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:

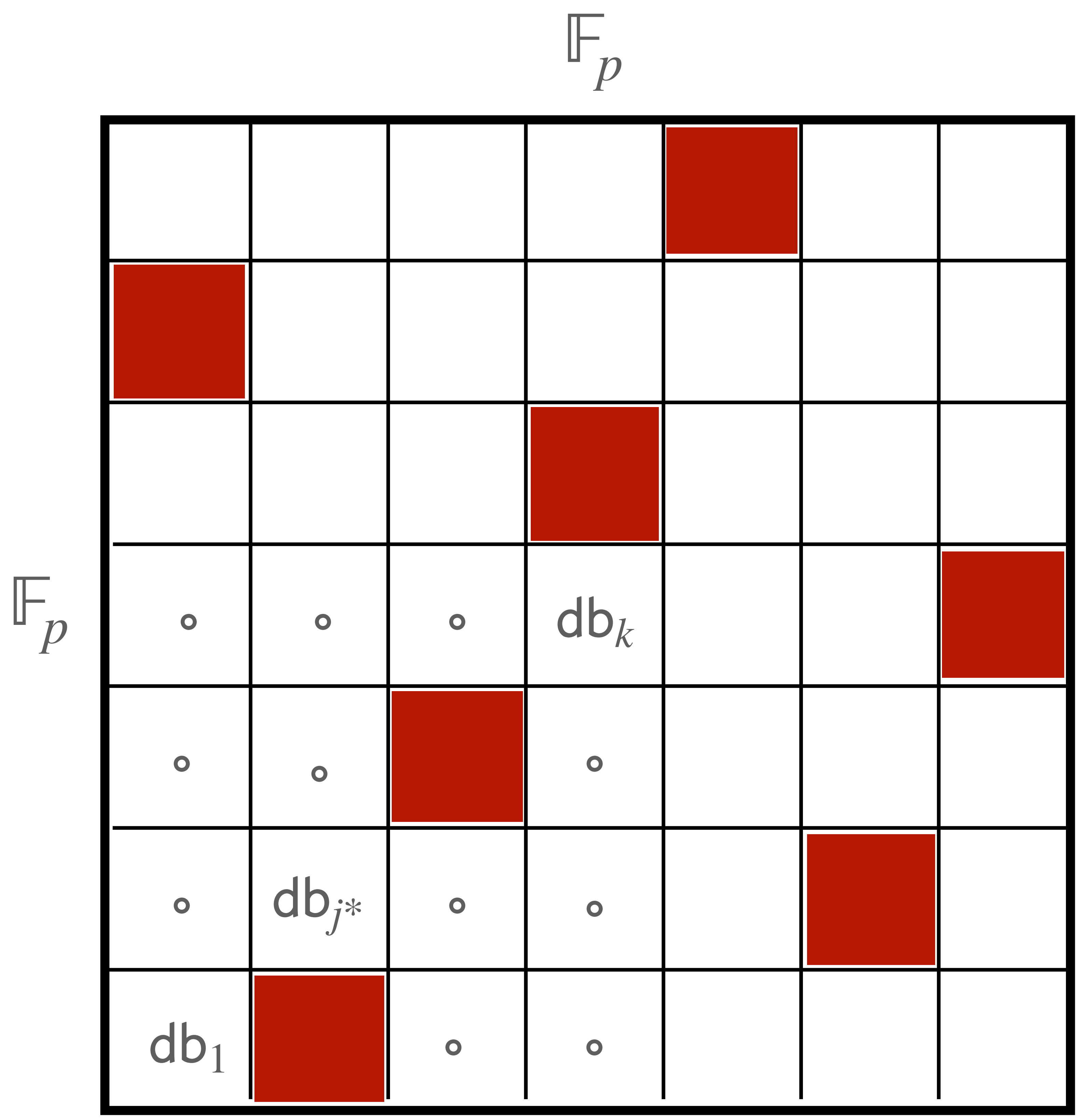
- Client queries for i: query line ℓ w/ prob. $1/\mathrm{poly}(N)$ (there are poly-many lines) and aborts.
- If client queries for $j \neq i$: only one point on the line is corrupt. Client never aborts.



Analyzing privacy: "variance attack"

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:

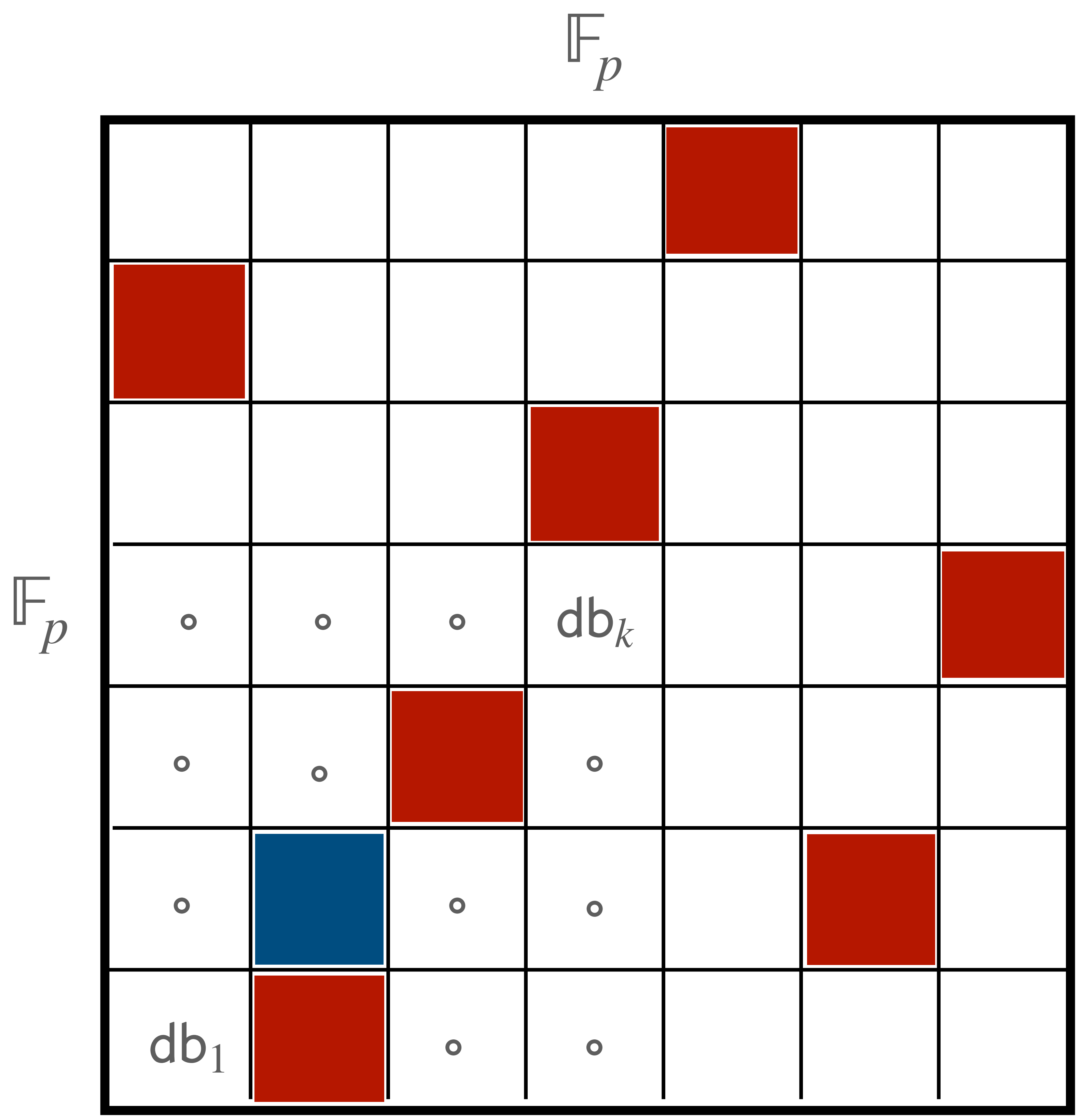
- Client queries for i: query line ℓ w/ prob. $1/\mathrm{poly}(N)$ (there are poly-many lines) and aborts.
- If client queries for $j \neq i$: only one point on the line is corrupt. Client never aborts.



Analyzing privacy: “variance attack”

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:

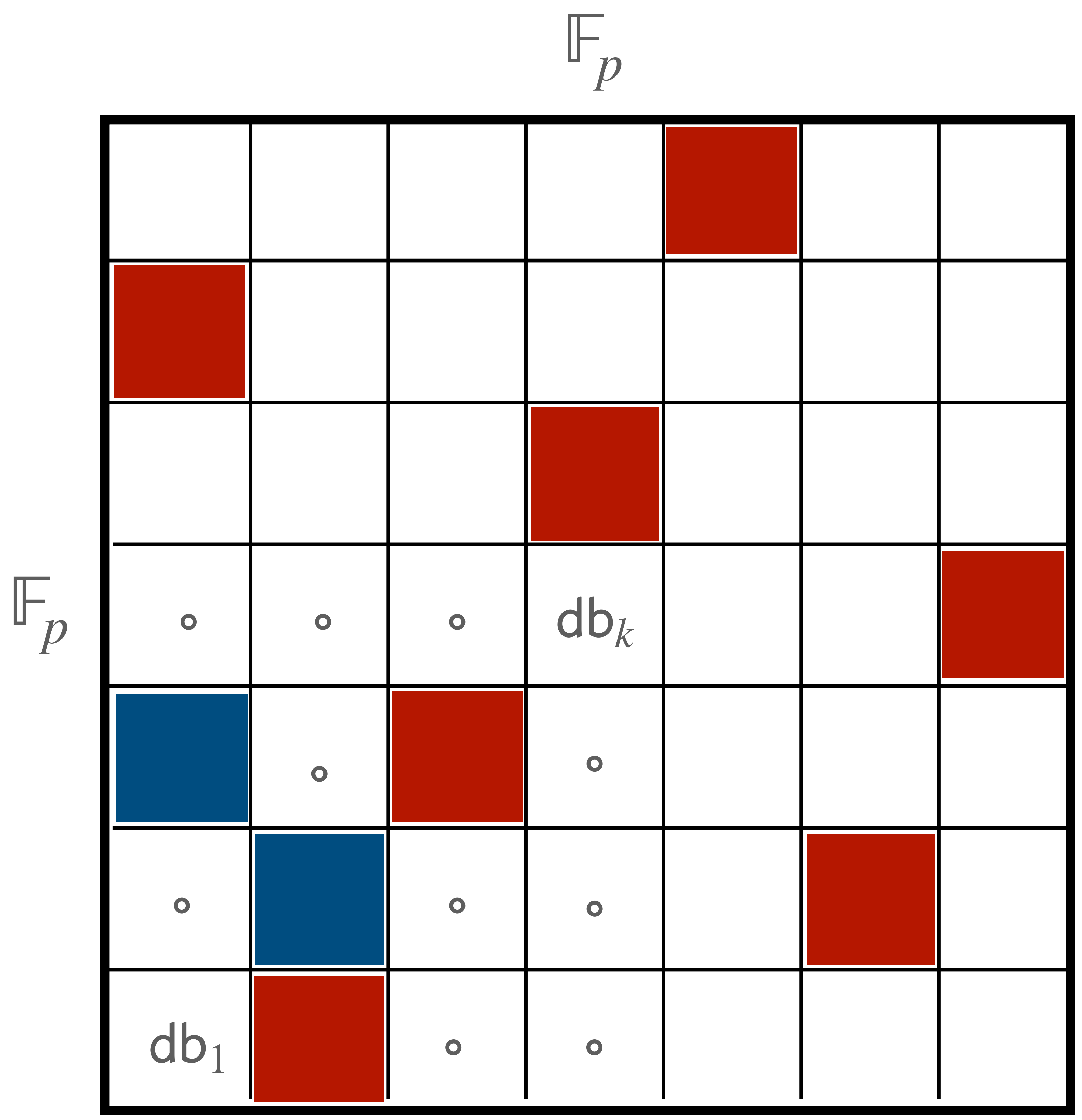
- Client queries for i: query line ℓ w/ prob. $1/\text{poly}(N)$ (there are poly-many lines) and aborts.
- If client queries for $j \neq i$: only one point on the line is corrupt. Client never aborts.



Analyzing privacy: “variance attack”

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:

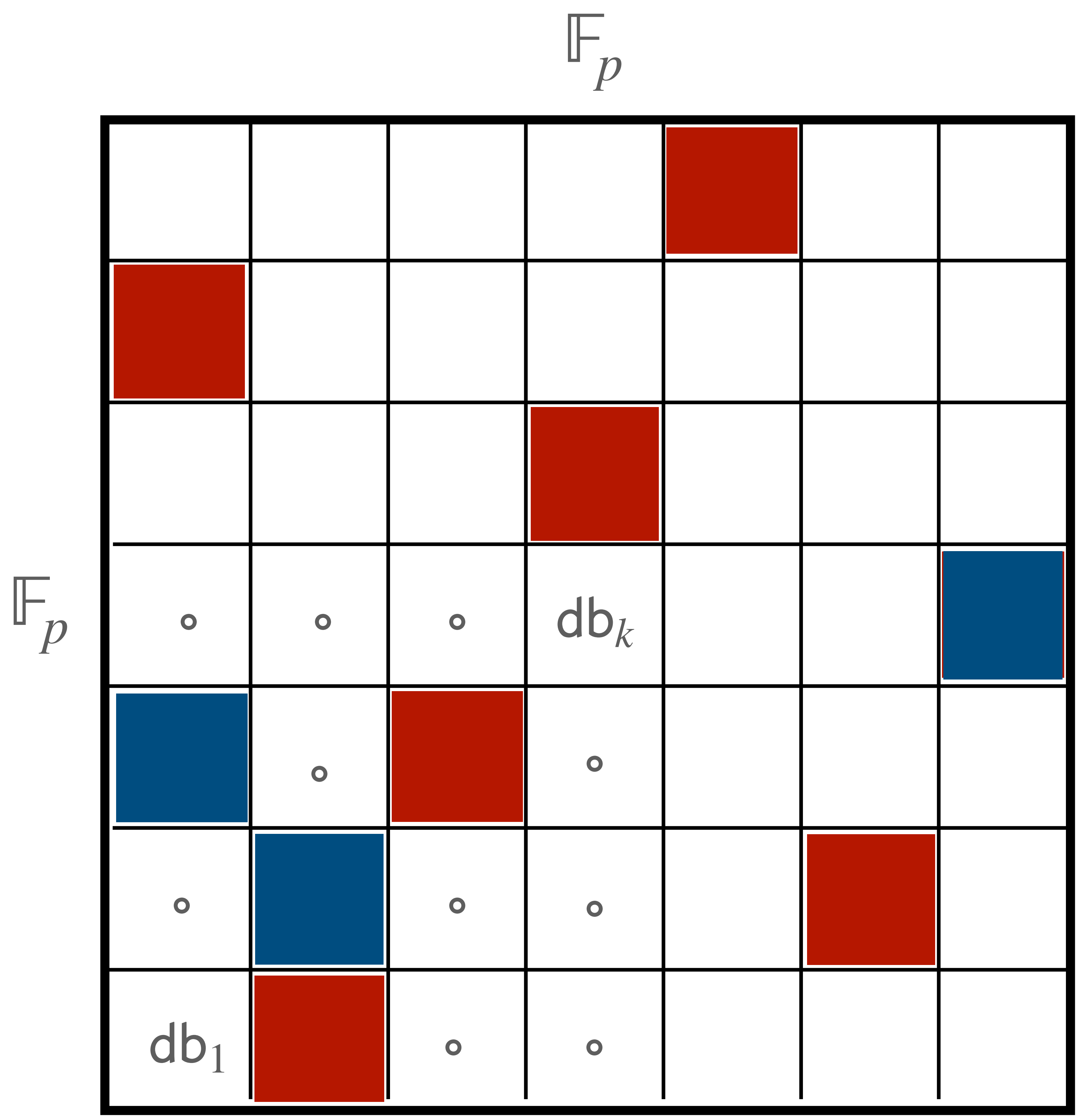
- Client queries for i: query line ℓ w/ prob. $1/\mathrm{poly}(N)$ (there are poly-many lines) and aborts.
- If client queries for $j \neq i$: only one point on the line is corrupt. Client never aborts.



Analyzing privacy: "variance attack"

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:

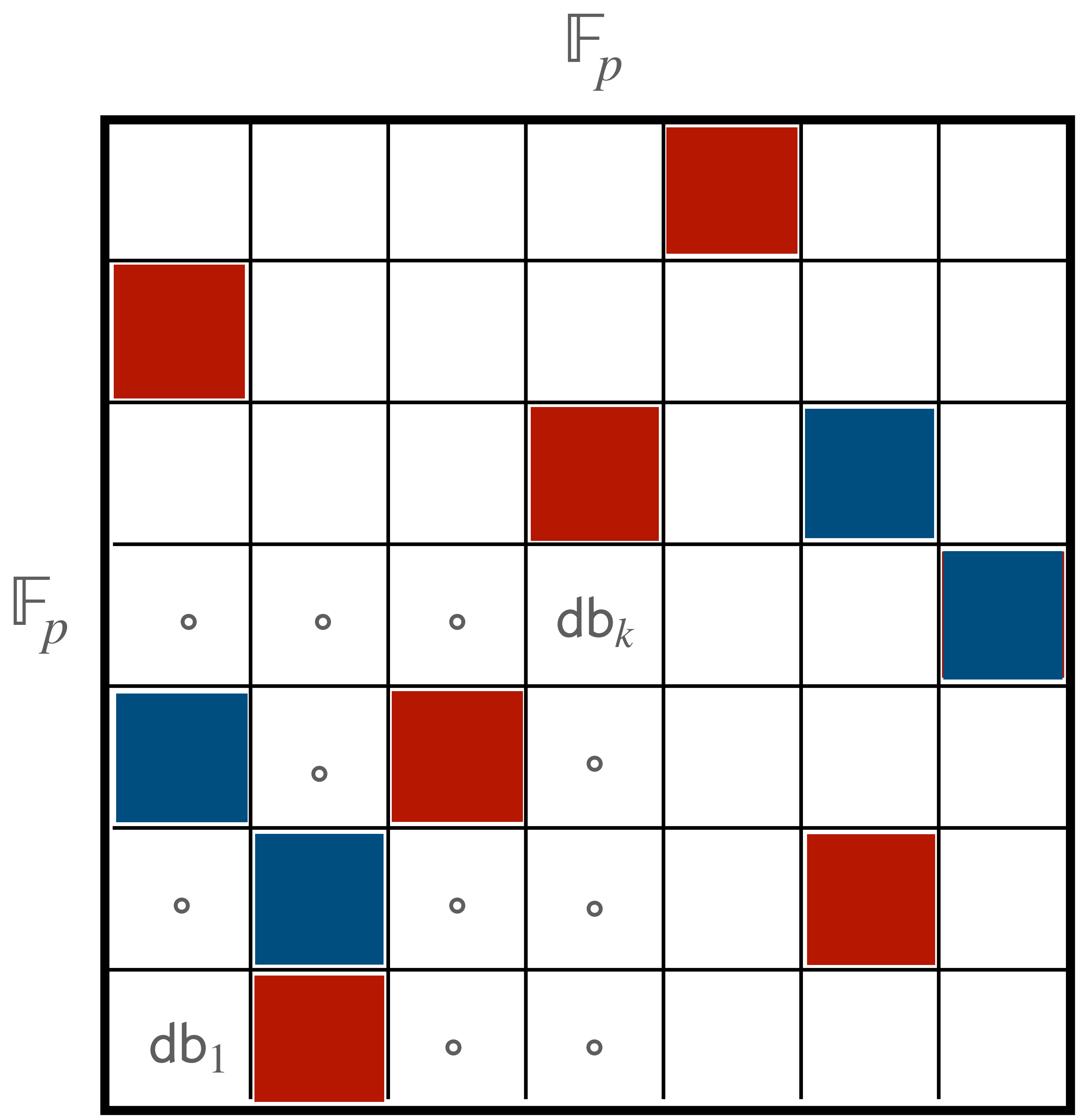
- Client queries for i: query line ℓ w/ prob. $1/\mathrm{poly}(N)$ (there are poly-many lines) and aborts.
- If client queries for $j \neq i$: only one point on the line is corrupt. Client never aborts.



Analyzing privacy: “variance attack”

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:

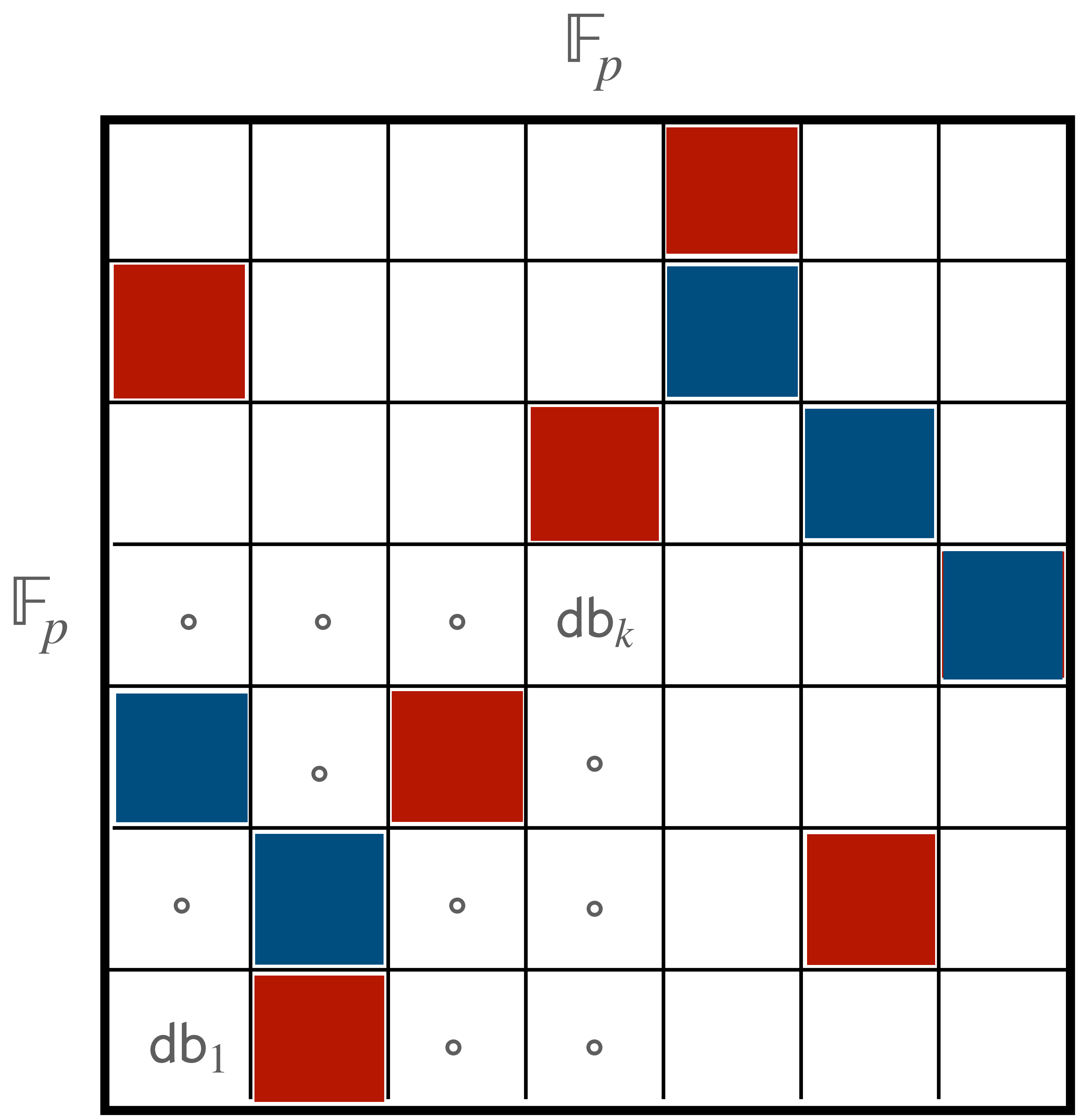
- Client queries for i: query line ℓ w/ prob. $1/\mathrm{poly}(N)$ (there are poly-many lines) and aborts.
- If client queries for $j \neq i$: only one point on the line is corrupt. Client never aborts.



Analyzing privacy: “variance attack”

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:

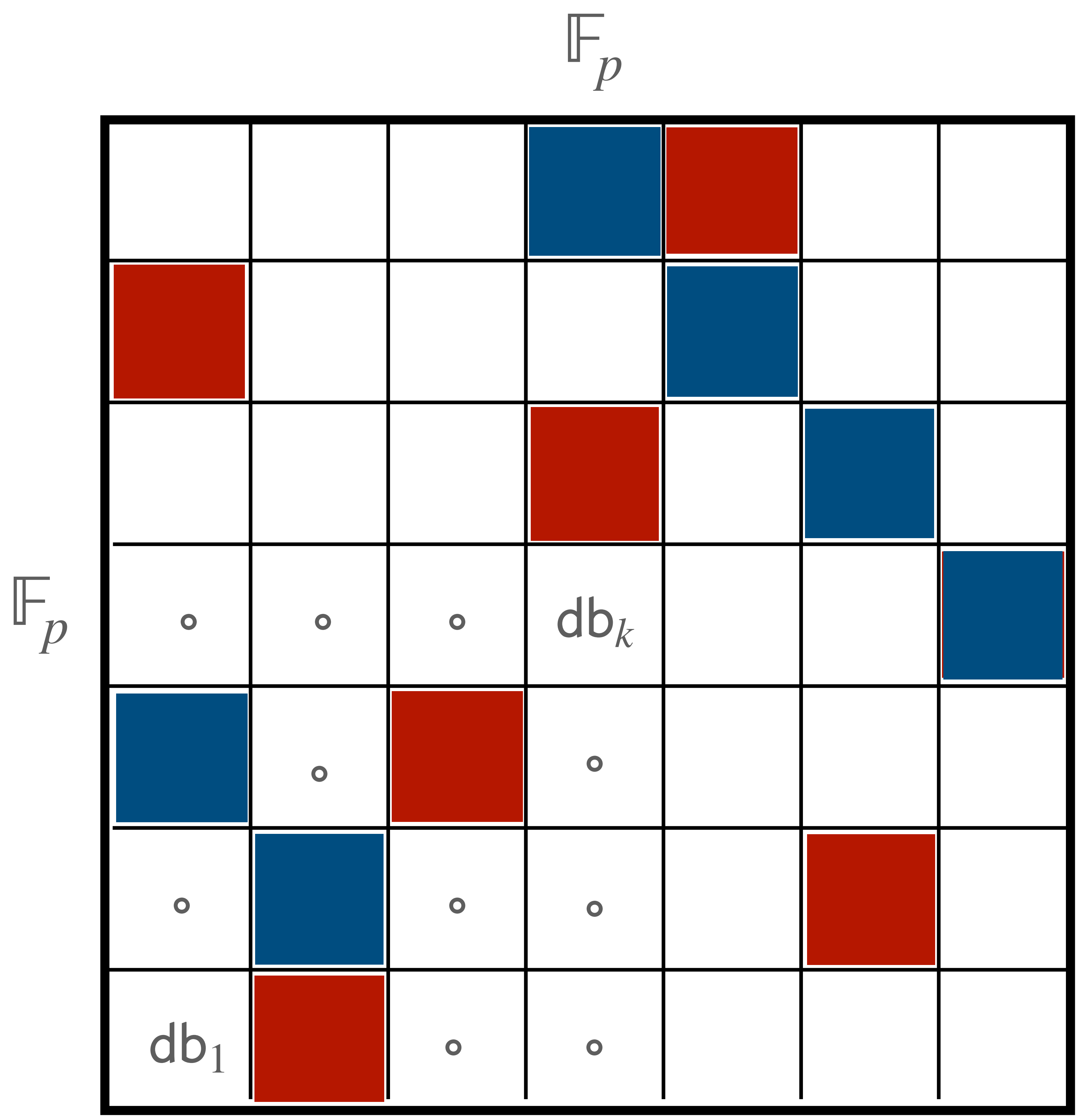
- Client queries for i: query line ℓ w/ prob. $1/\text{poly}(N)$ (there are poly-many lines) and aborts.
- If client queries for $j \neq i$: only one point on the line is corrupt. Client never aborts.



Analyzing privacy: "variance attack"

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:

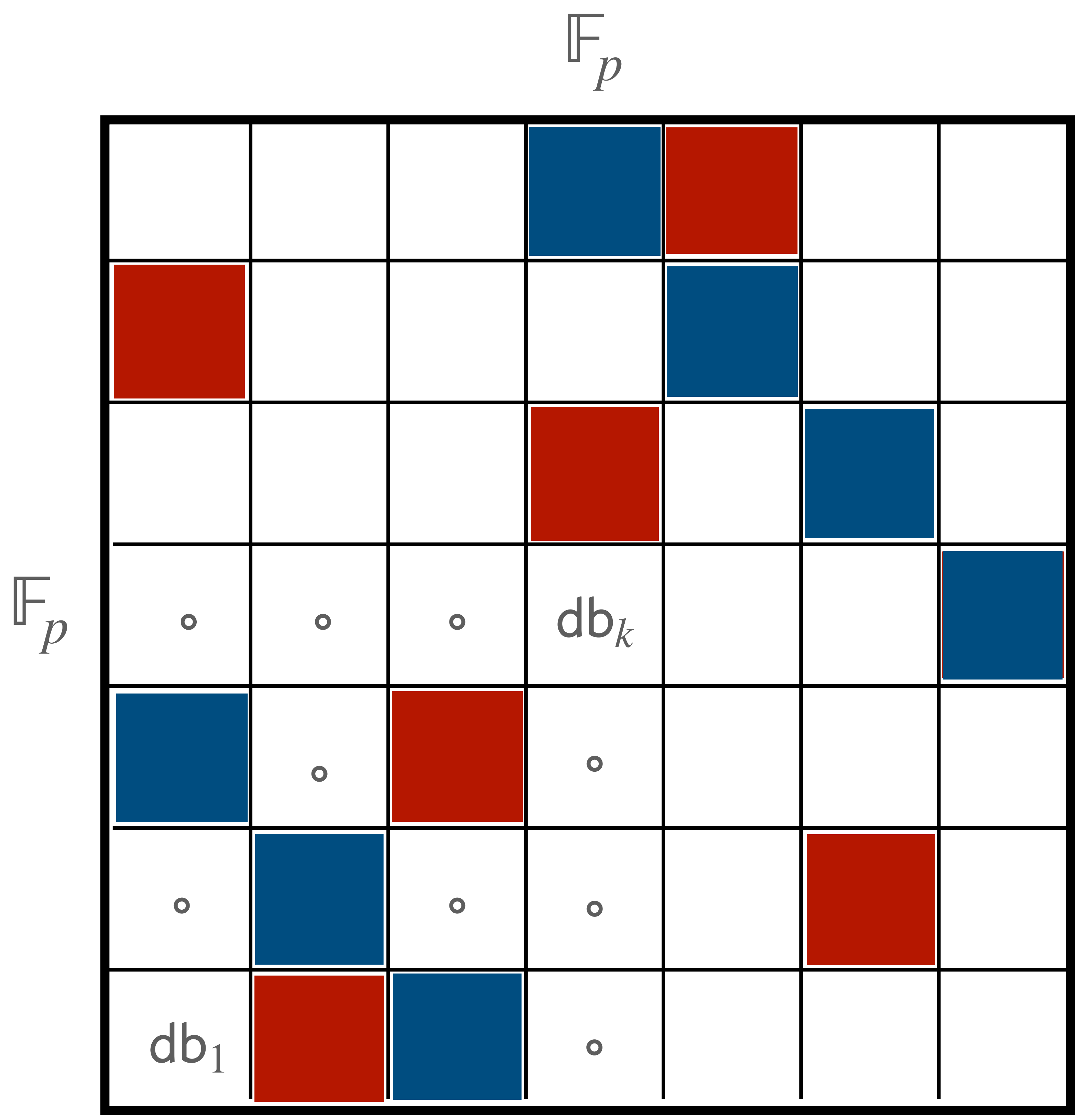
- Client queries for i: query line ℓ w/ prob. $1/\mathrm{poly}(N)$ (there are poly-many lines) and aborts.
- If client queries for $j \neq i$: only one point on the line is corrupt. Client never aborts.



Analyzing privacy: “variance attack”

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:

- Client queries for i: query line ℓ w/ prob. $1/\mathrm{poly}(N)$ (there are poly-many lines) and aborts.
- If client queries for $j \neq i$: only one point on the line is corrupt. Client never aborts.

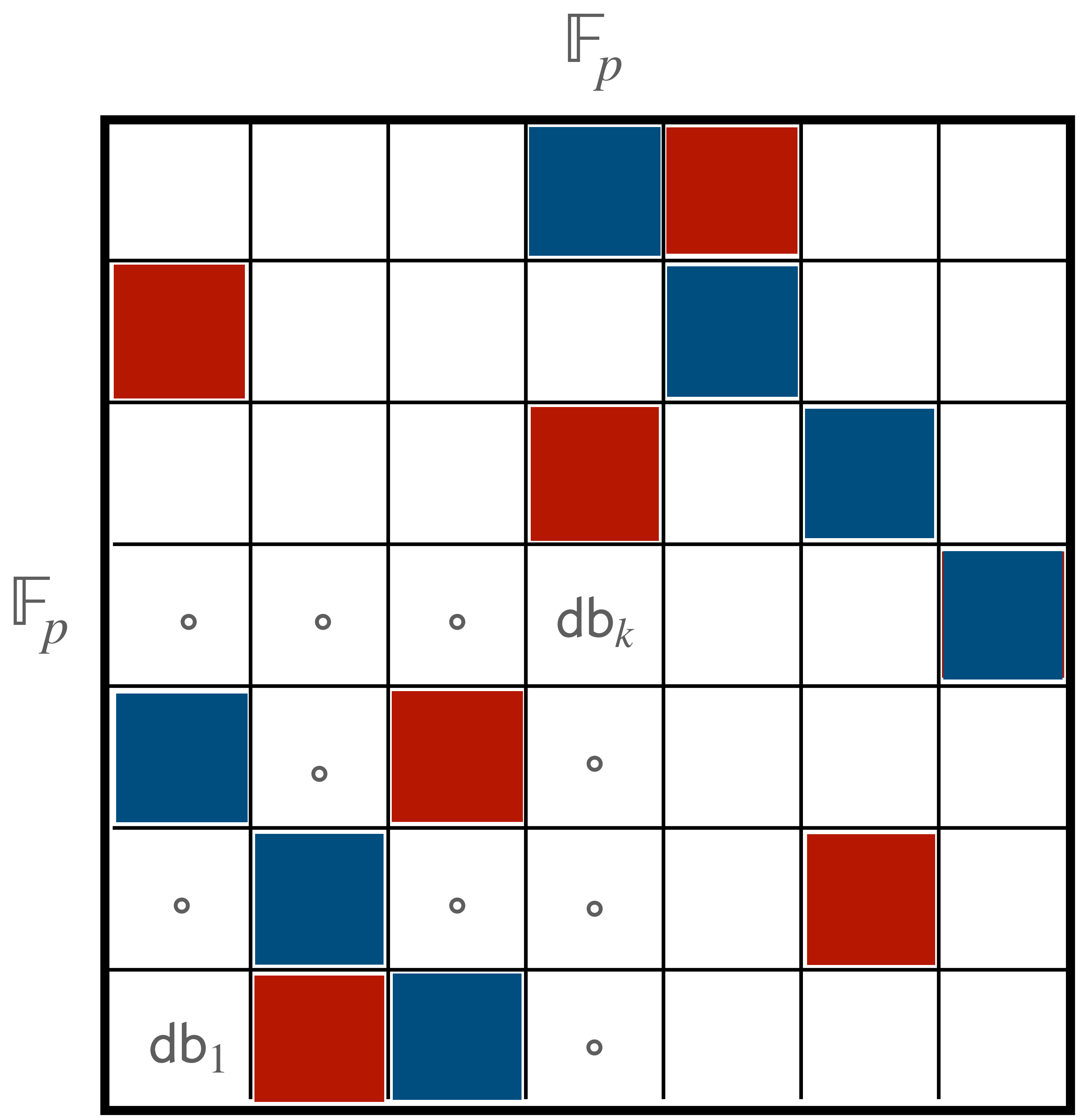


Analyzing privacy: “variance attack”

To introduce selective failure on index i, the adversary corrupts (opening proofs on) line ℓ through db_i:

- Client queries for i: query line ℓ w/ prob. $1/\mathrm{poly}(N)$ (there are poly-many lines) and aborts.
- If client queries for $j \neq i$: only one point on the line is corrupt. Client never aborts.

Selective Failure attack!



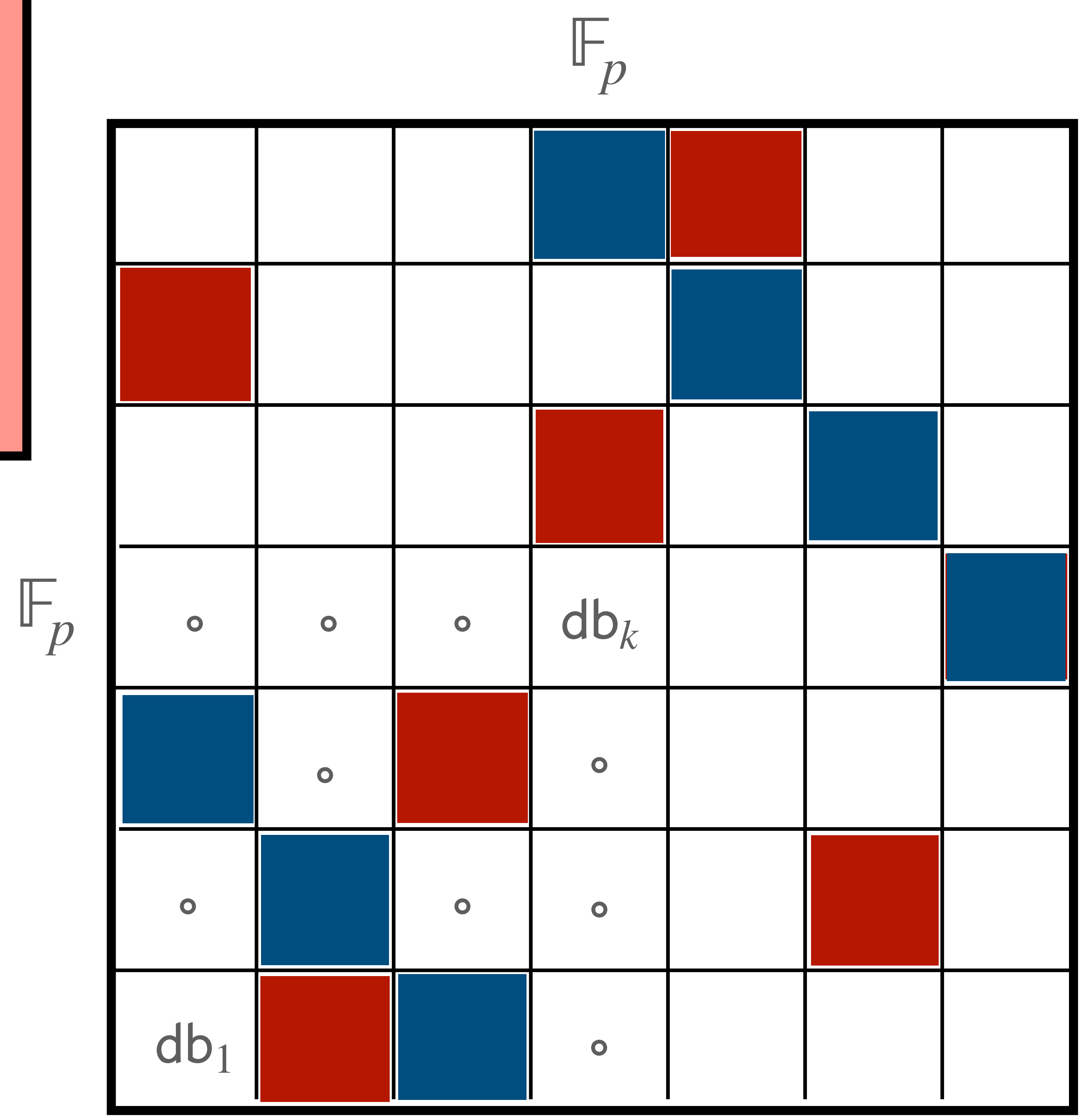
Ana nce attack”

Both indices decoded with good probability

To introd
adversar
ℓ throug

- Client
  $1/\text{poly}($
  aborts.
- If client queries for $j \neq i$: only one point on the line is corrupt. Client never aborts.

Selective Failure attack!



Ana… nce attack”

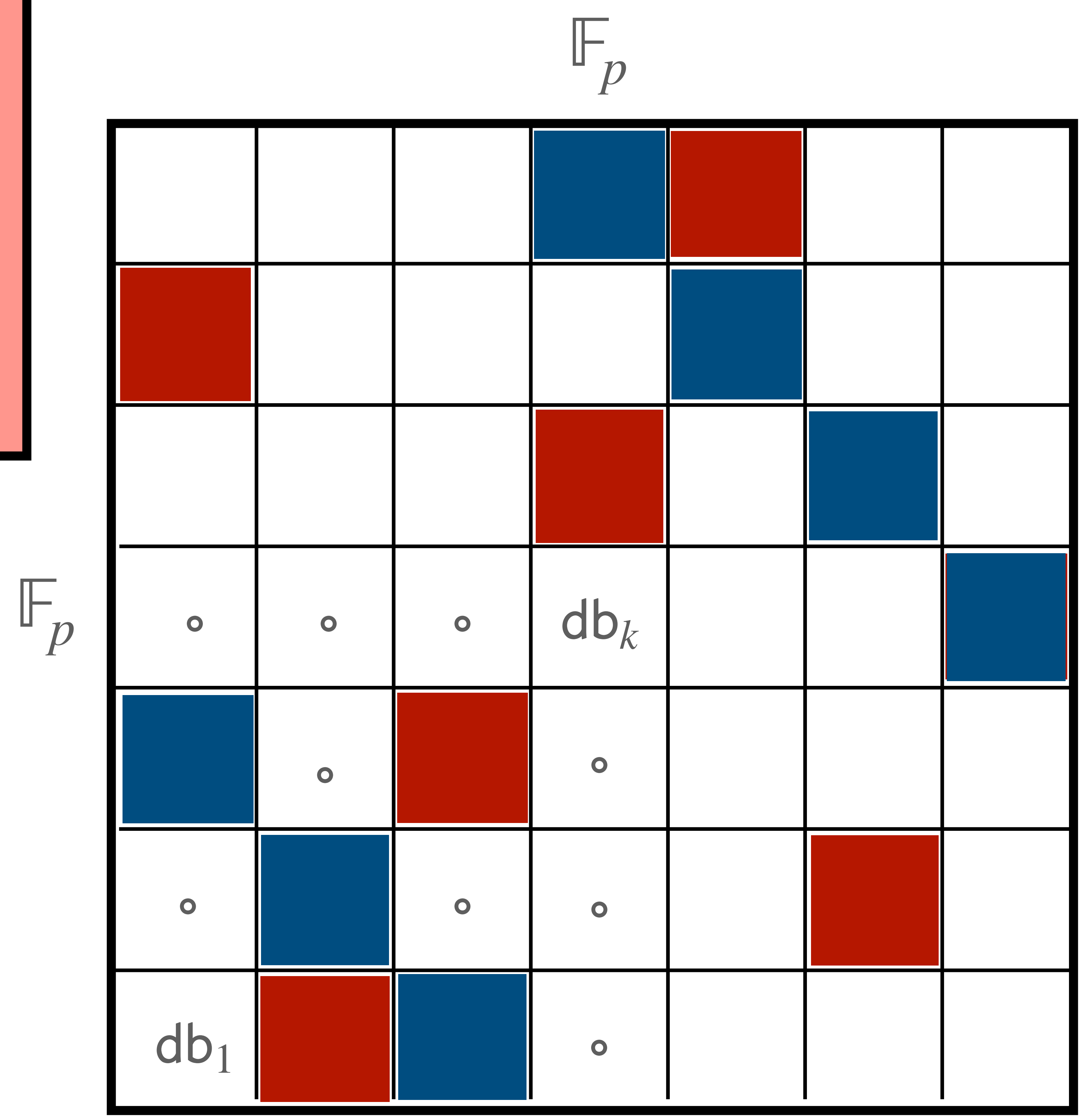
Both indices decoded with good probability

$\Pr[\text{decode } i] > 2/3$

To introd…
adversar…
ℓ throug…

- Client …
  $1/\text{poly}($…
  aborts.
- If client queries for $j \neq i$: only one point on the line is corrupt. Client never aborts.

Selective Failure attack!



Ana… …nce attack”

Both indices decoded with good probability

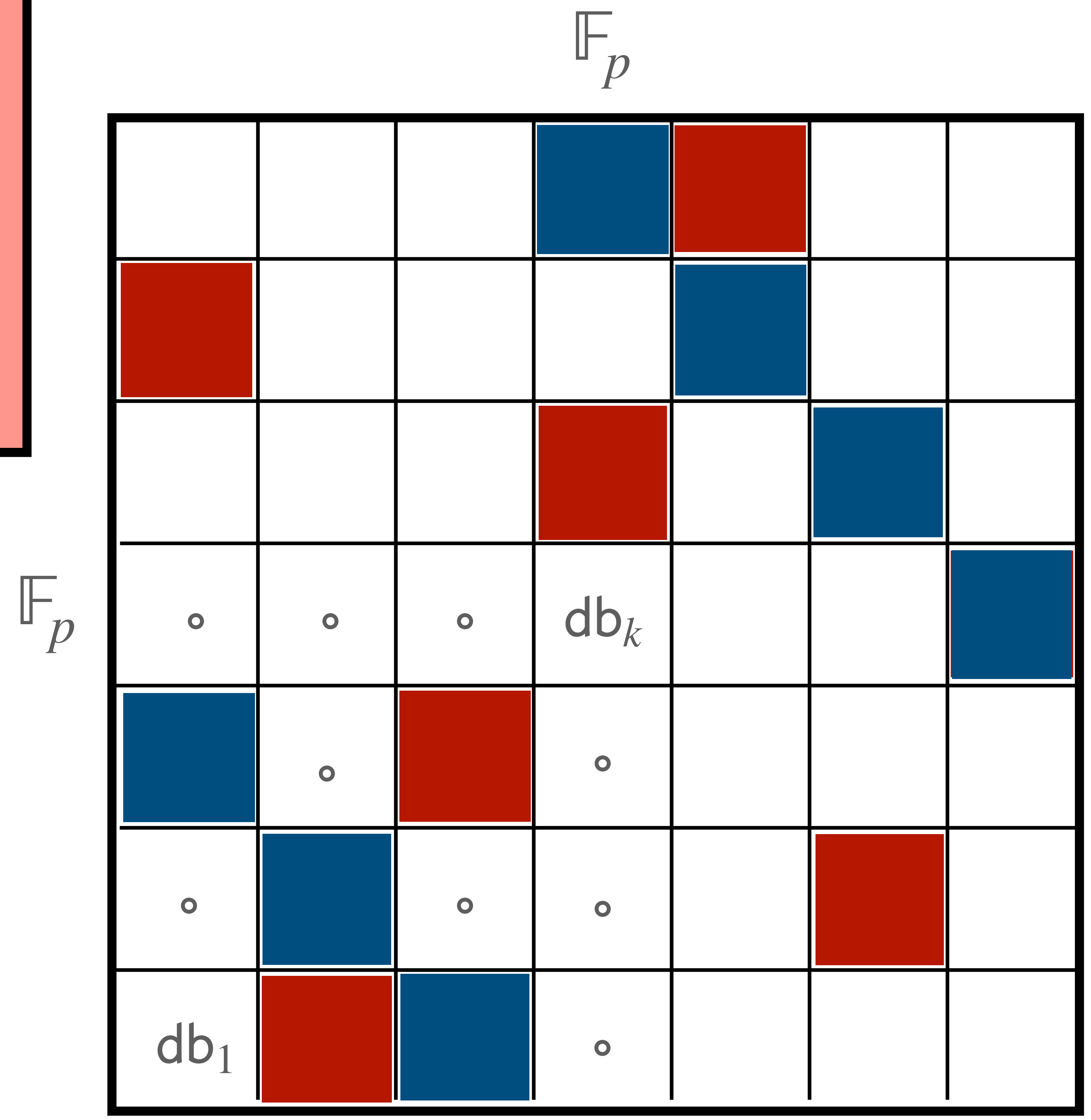
$$\Pr[\text{decode } i] > 2/3$$

$$\Pr[\text{decode } j] > 2/3$$

To introd…
adversar…
ℓ throug…

- Client … $1/\text{poly}($… aborts.
- If client queries for $j \neq i$: only one point on the line is corrupt. Client never aborts.

Selective Failure attack!



Ana nce attack”

Both indices decoded with good probability

$\Pr[\text{decode } i] > 2/3$

$\Pr[\text{decode } j] > 2/3$

However, there could be a big gap between the decoding probabilities:

$|\Pr[\text{decode } i] - \Pr[\text{decode } j]| > \text{notice}(n)$

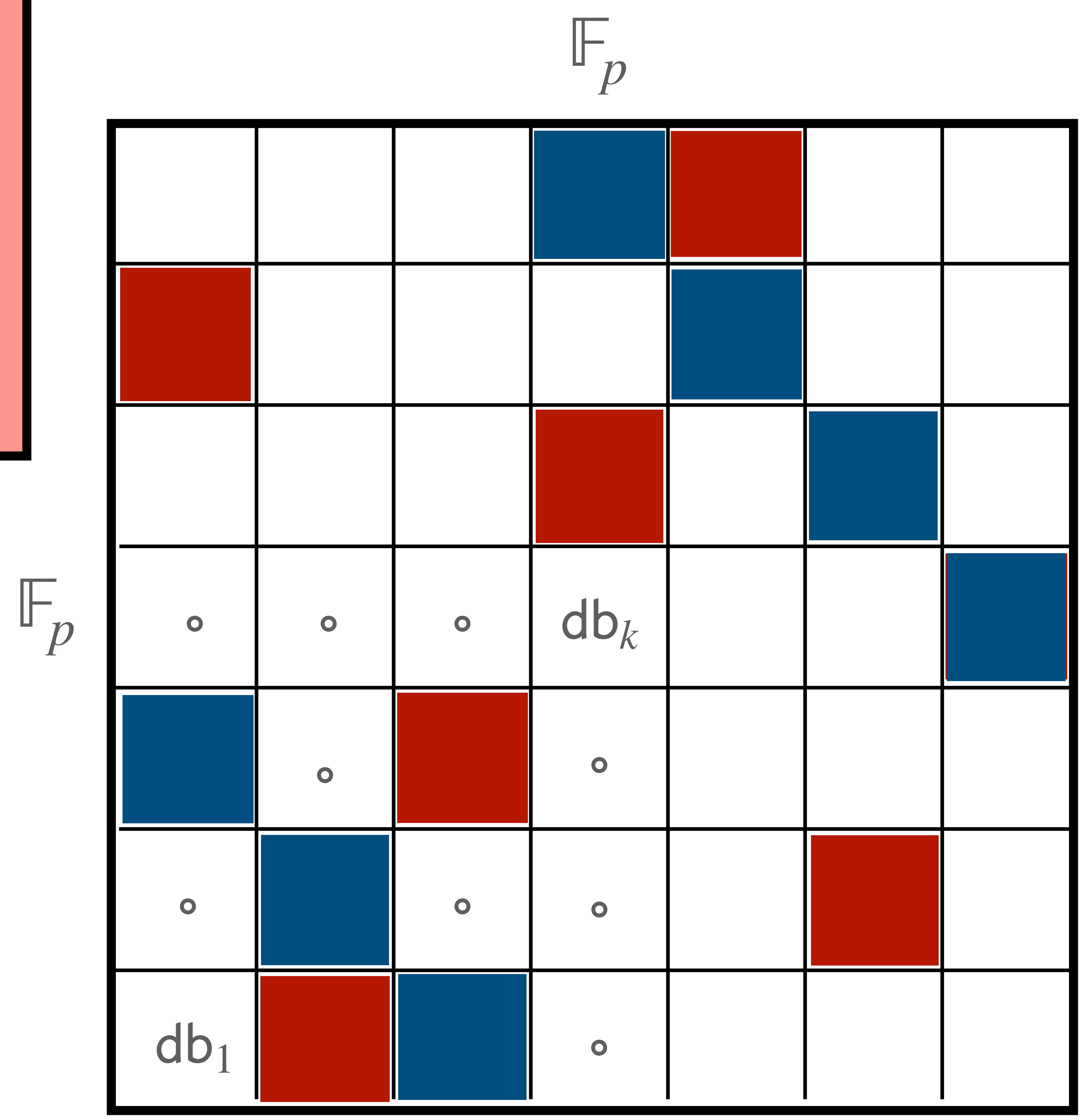
To introd

adversar

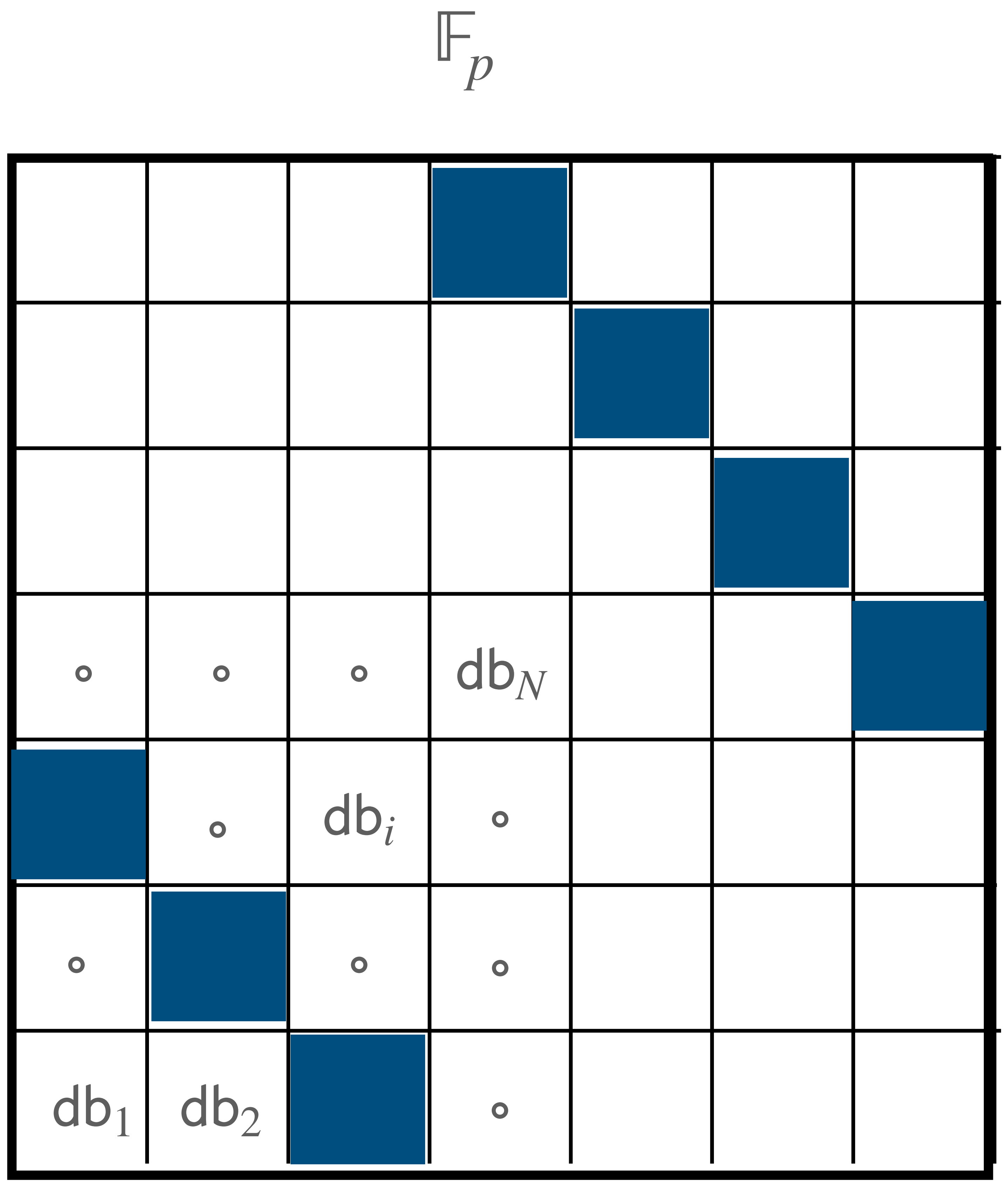
ℓ throug

- Client
  $1/\text{poly}($
  aborts.
- If client queries for $j \neq i$: only one point on the line is corrupt. Client never aborts.

Selective Failure attack!

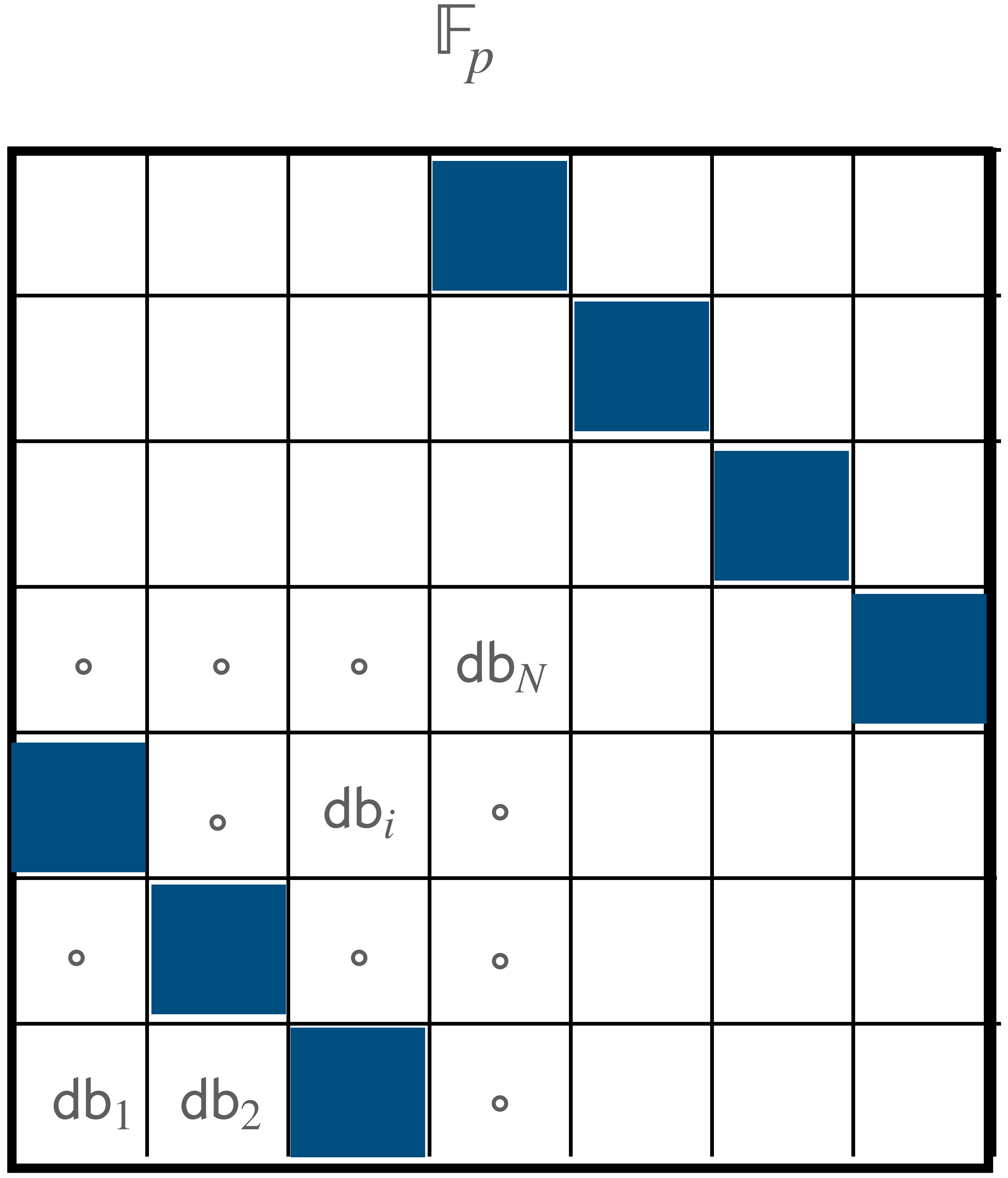


Decrease decoder's success probability: test points



Decrease decoder's success probability: test points

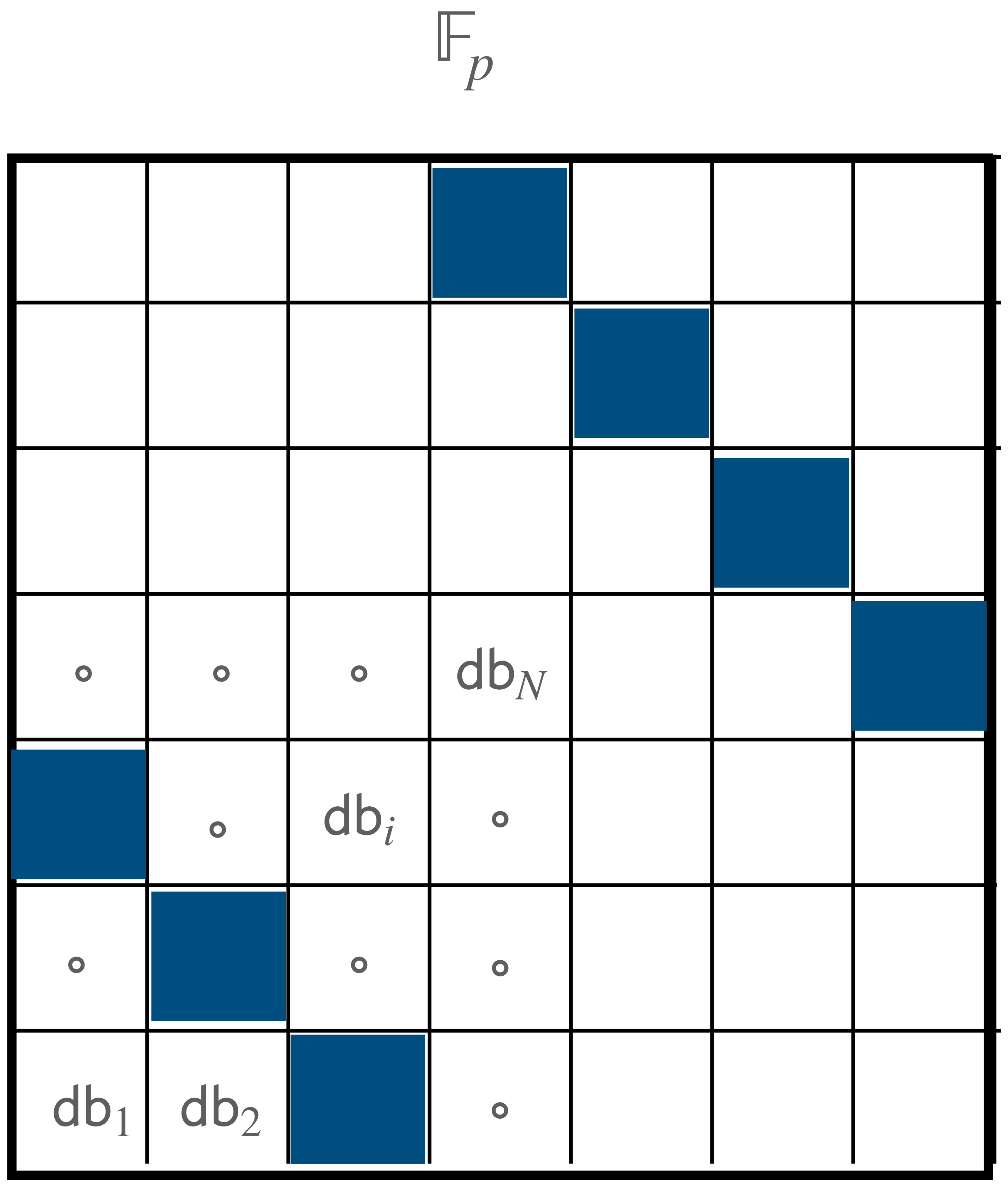
- The approach so far: try to recover from corruptions.



Decrease decoder's success probability: test points

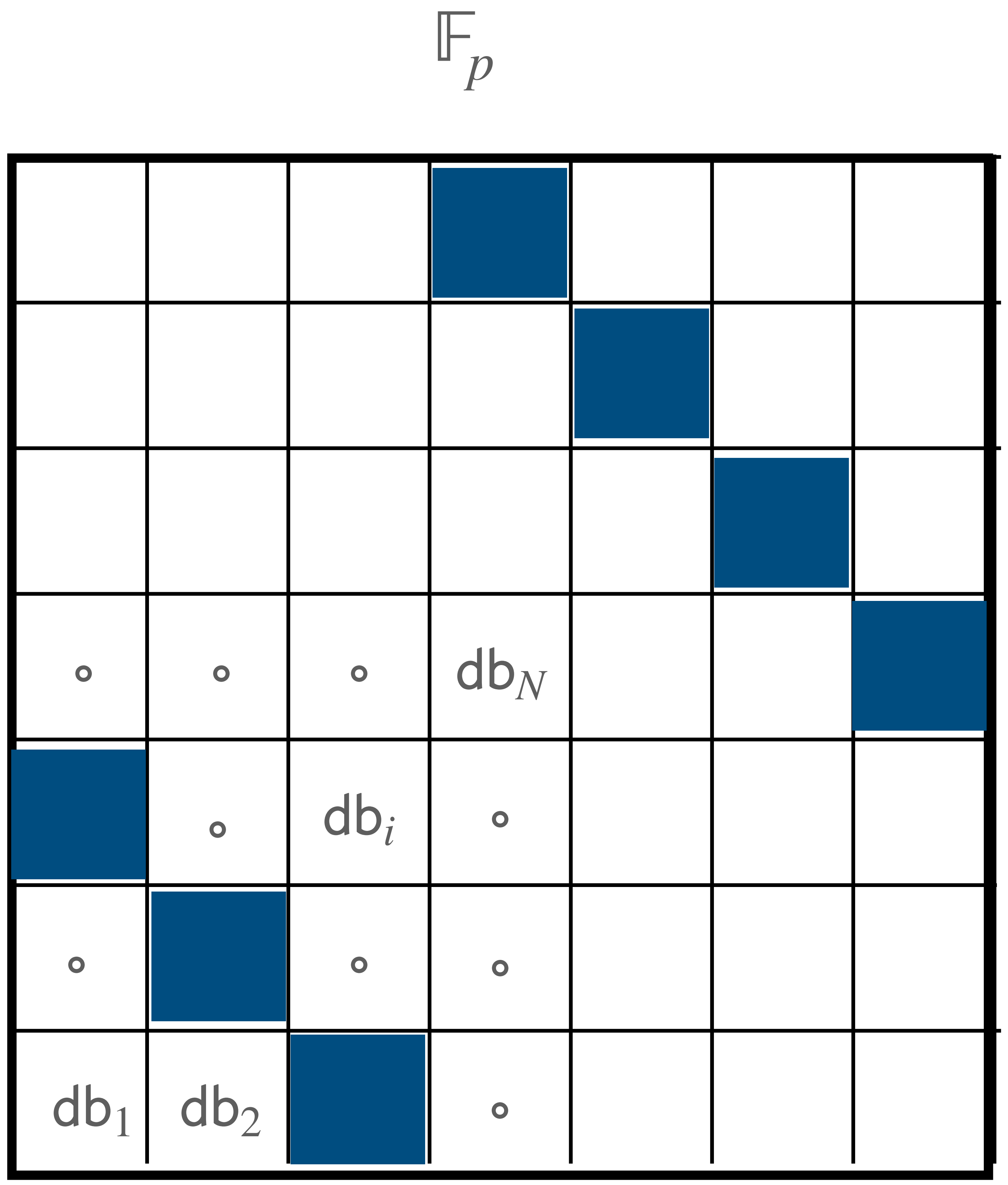
- The approach so far: try to recover from corruptions.

- Naive idea: make more queries to shrink decoding probability gap (recover from even more corruptions).



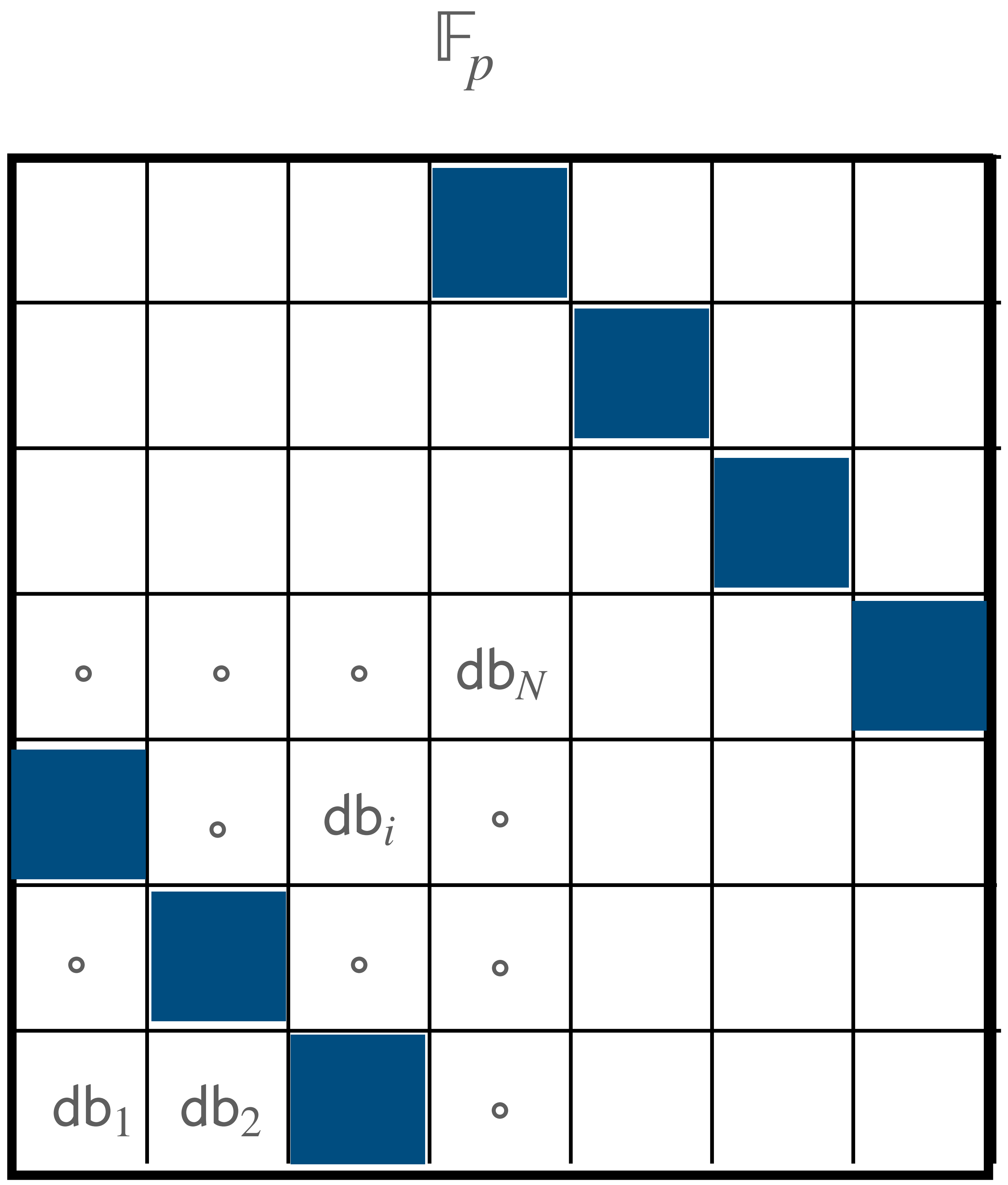
Decrease decoder's success probability: test points

- The approach so far: try to recover from corruptions.
- Naive idea: make more queries to shrink decoding probability gap (recover from even more corruptions).
  - Requires too many queries!



Decrease decoder's success probability: test points

- The approach so far: try to recover from corruptions.
- Naive idea: make more queries to shrink decoding probability gap (recover from even more corruptions).
  - Requires too many queries!
- New approach: try to detect corruptions and reject.

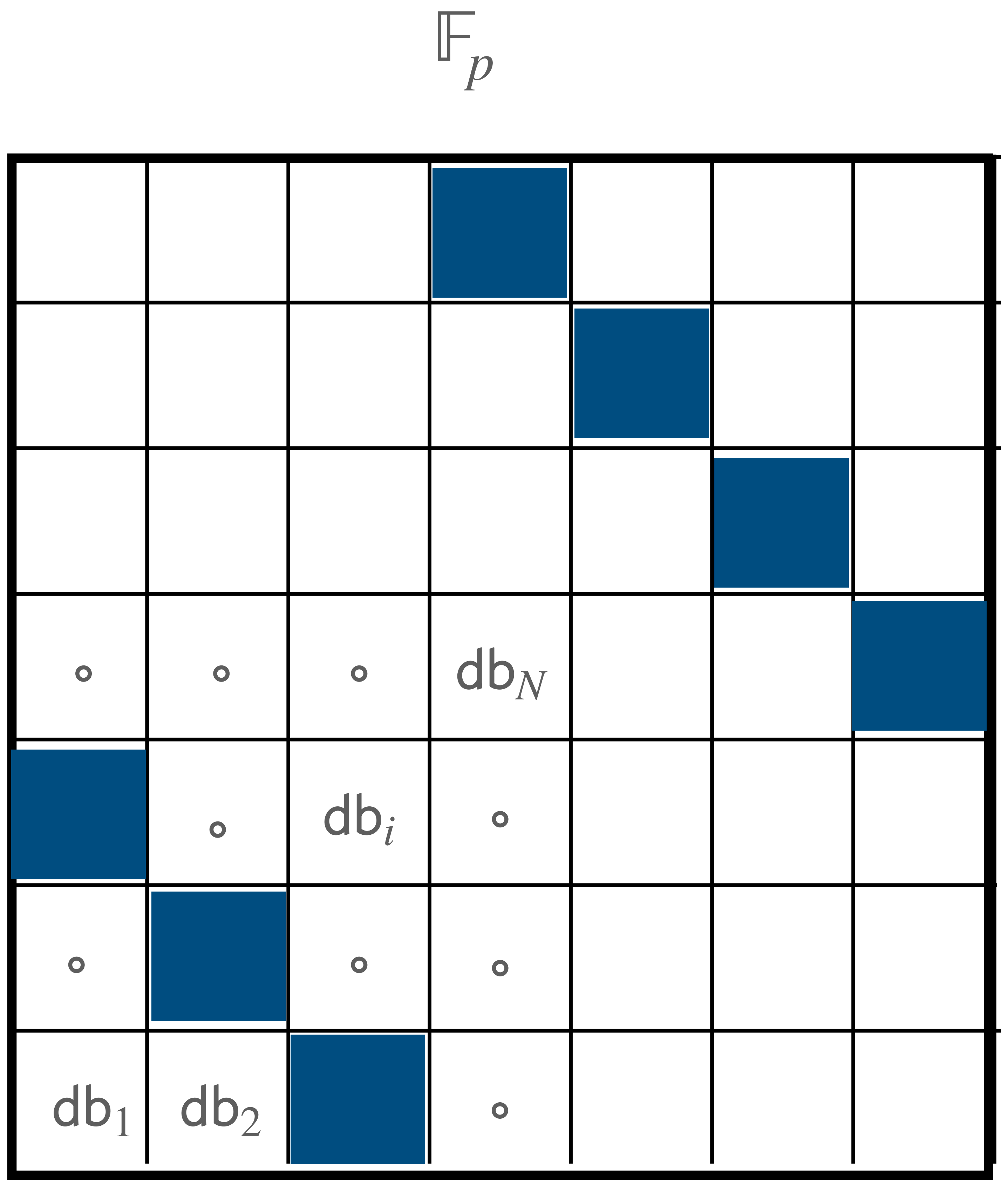


Decrease decoder's success probability: test points

- The approach so far: try to recover from corruptions.

- Naive idea: make more queries to shrink decoding probability gap (recover from even more corruptions).
  - Requires too many queries!

- New approach: try to detect corruptions and reject.
  - Rejecting corruptions in the LDC query introduces selective failure attack because the locations queried are correlated with i.

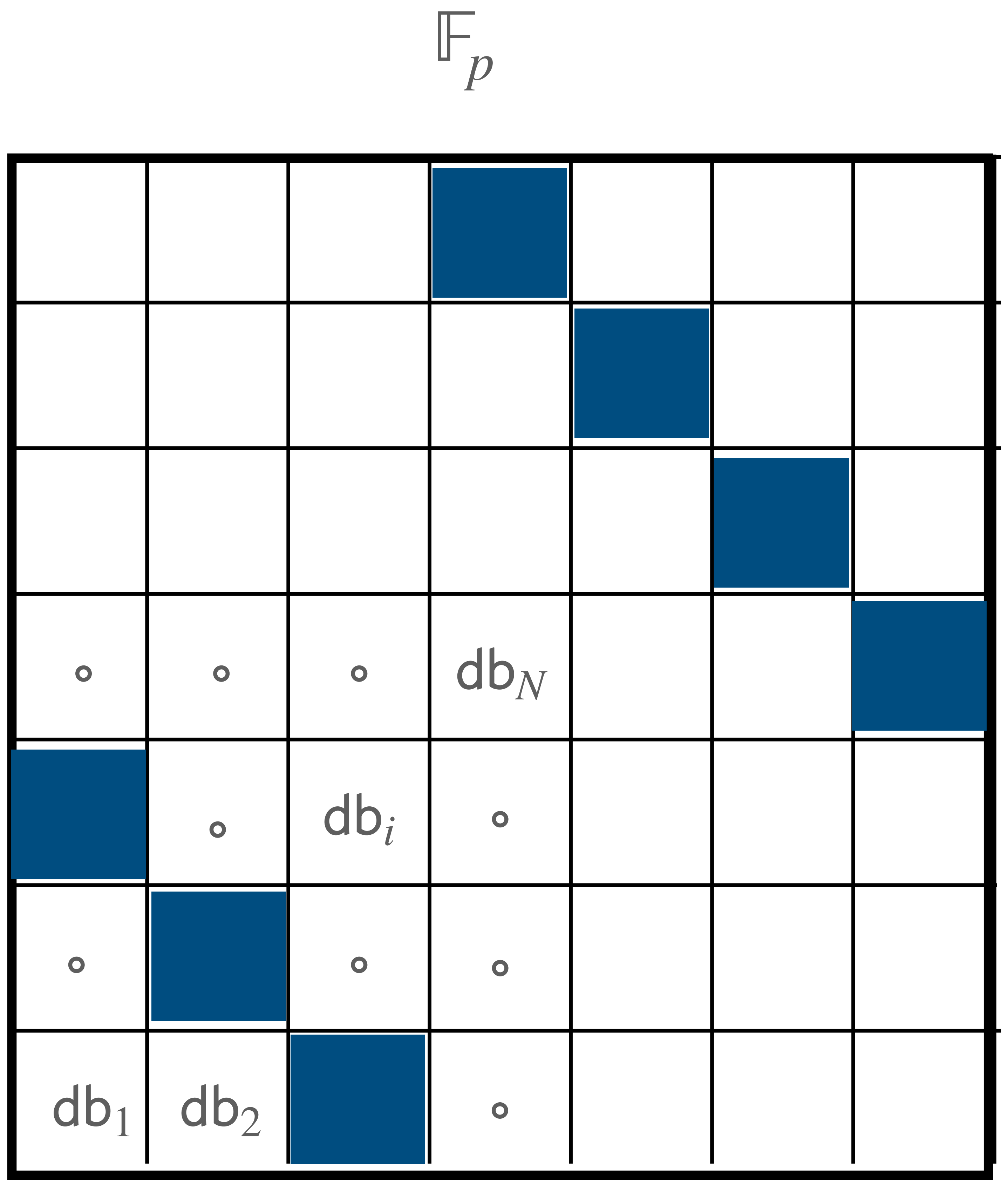


Decrease decoder's success probability: test points

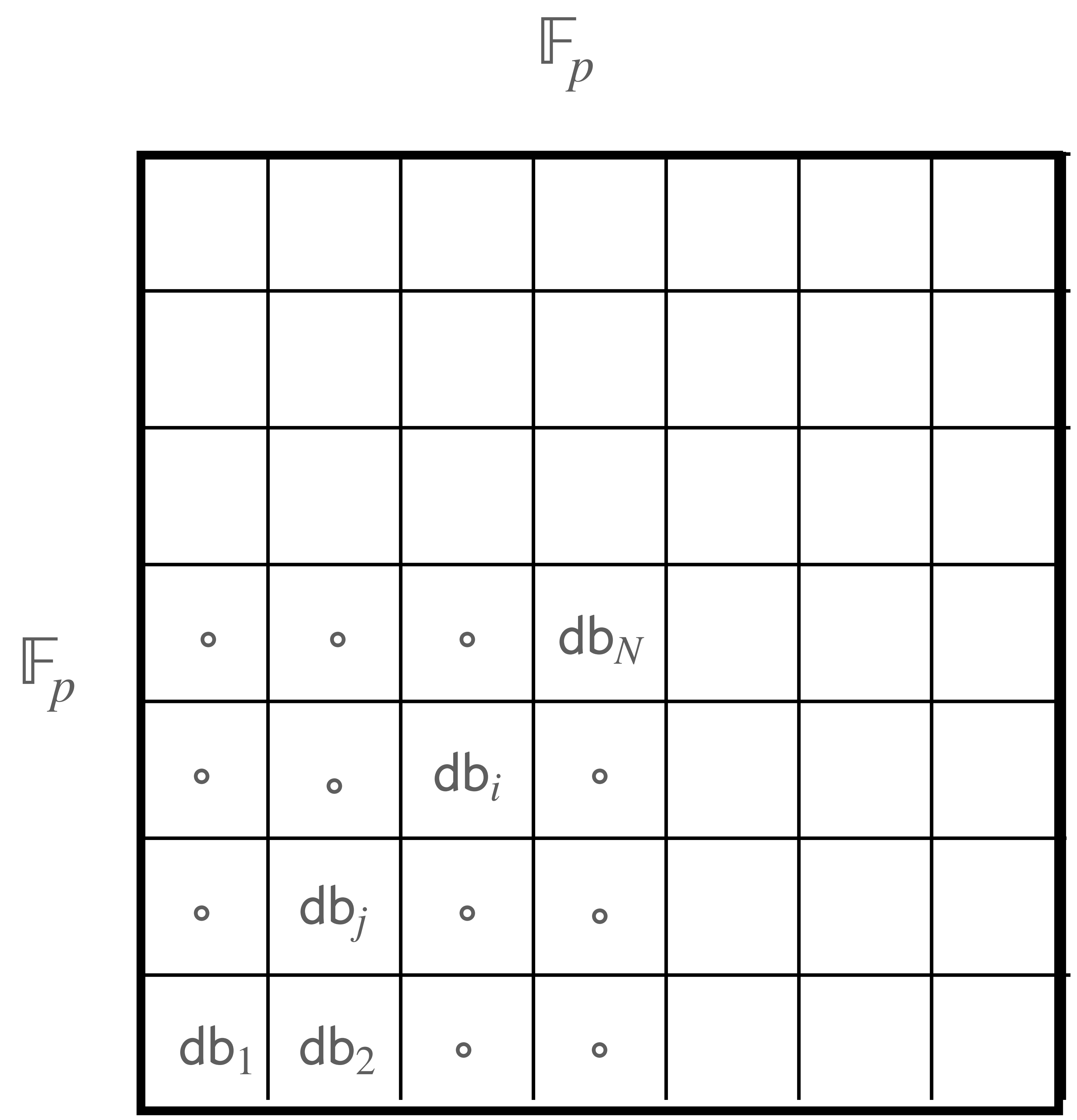
- The approach so far: try to recover from corruptions.

- Naive idea: make more queries to shrink decoding probability gap (recover from even more corruptions).
  - Requires too many queries!

- New approach: try to detect corruptions and reject.
  - Rejecting corruptions in the LDC query introduces selective failure attack because the locations queried are correlated with i.
  - Instead we detect corruptions on a set of random *test* points.

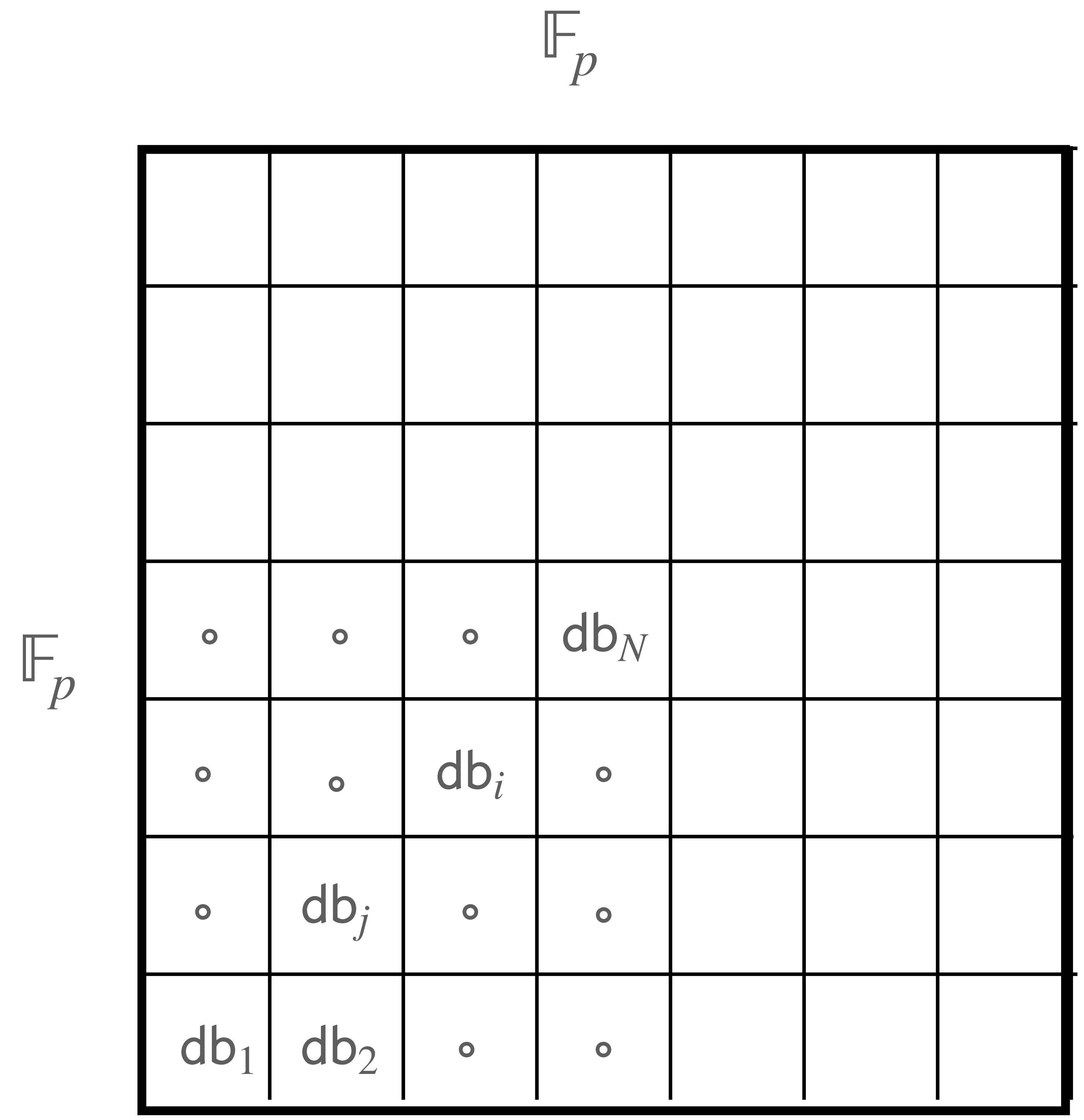


Decrease decoder's success probability: test points



Decrease decoder's success probability: test points

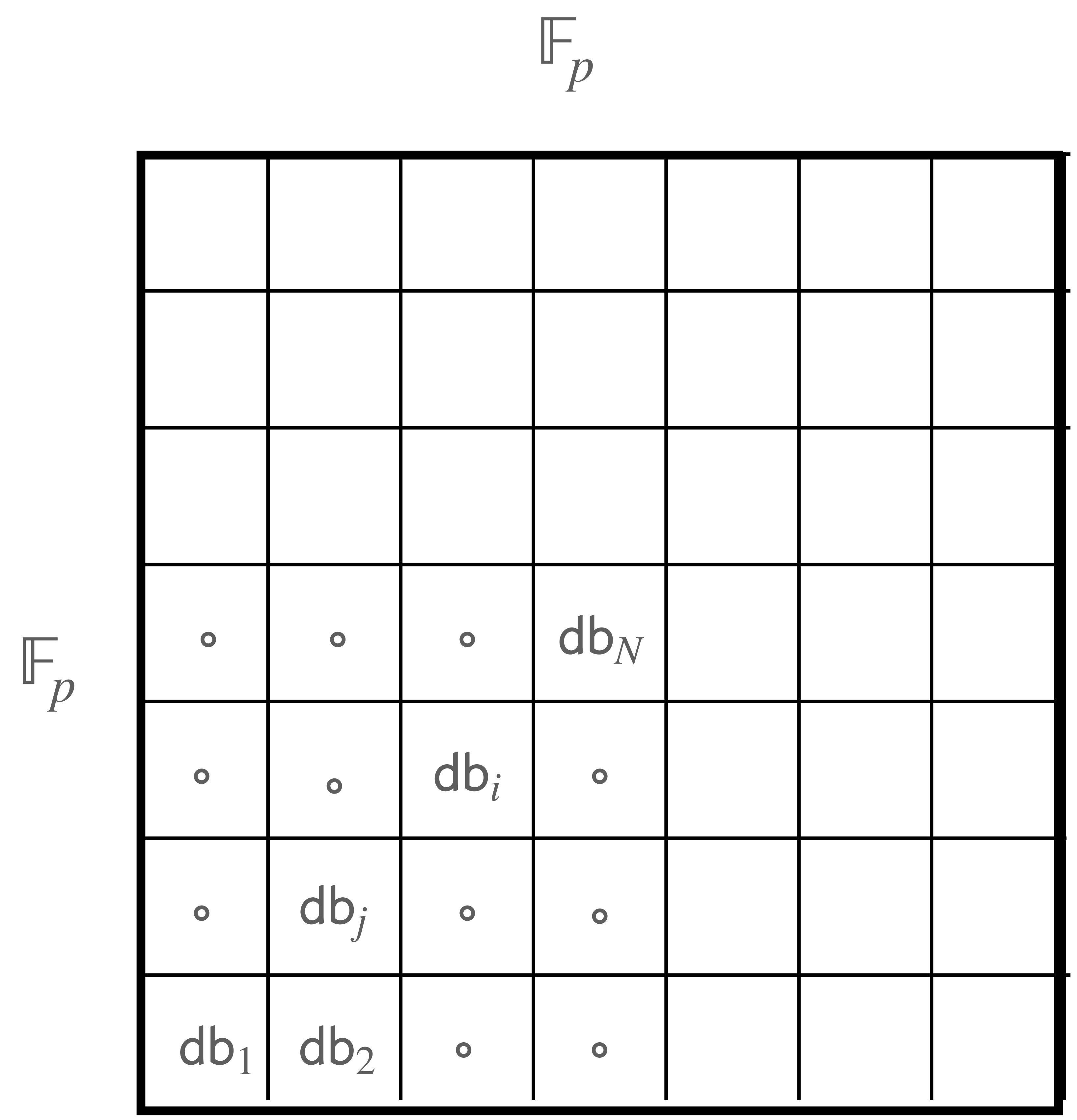
Modified local decoding with test queries



Decrease decoder's success probability: test points

Modified local decoding with test queries

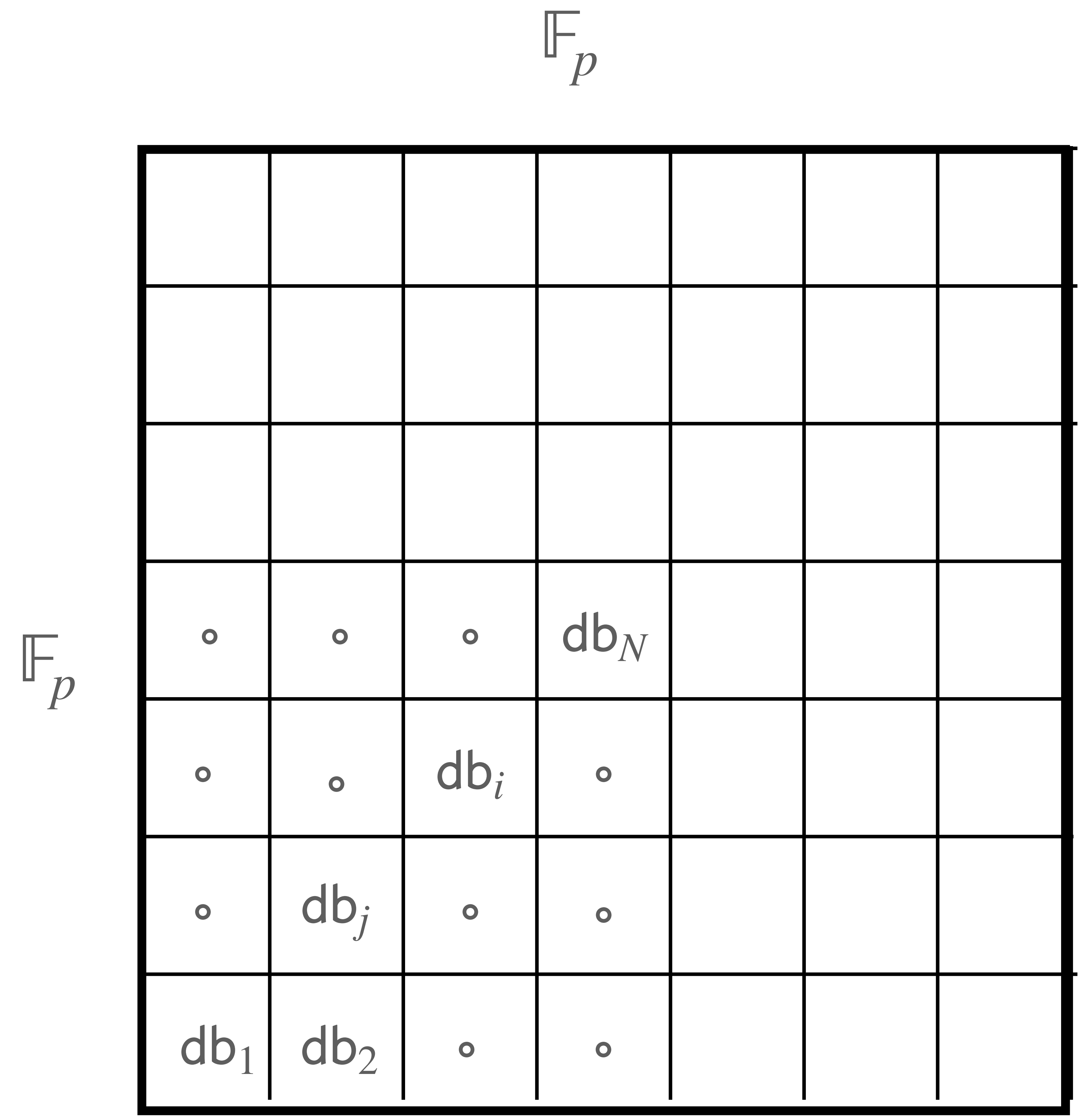
1. Want: db_j



Decrease decoder's success probability: test points

Modified local decoding with test queries

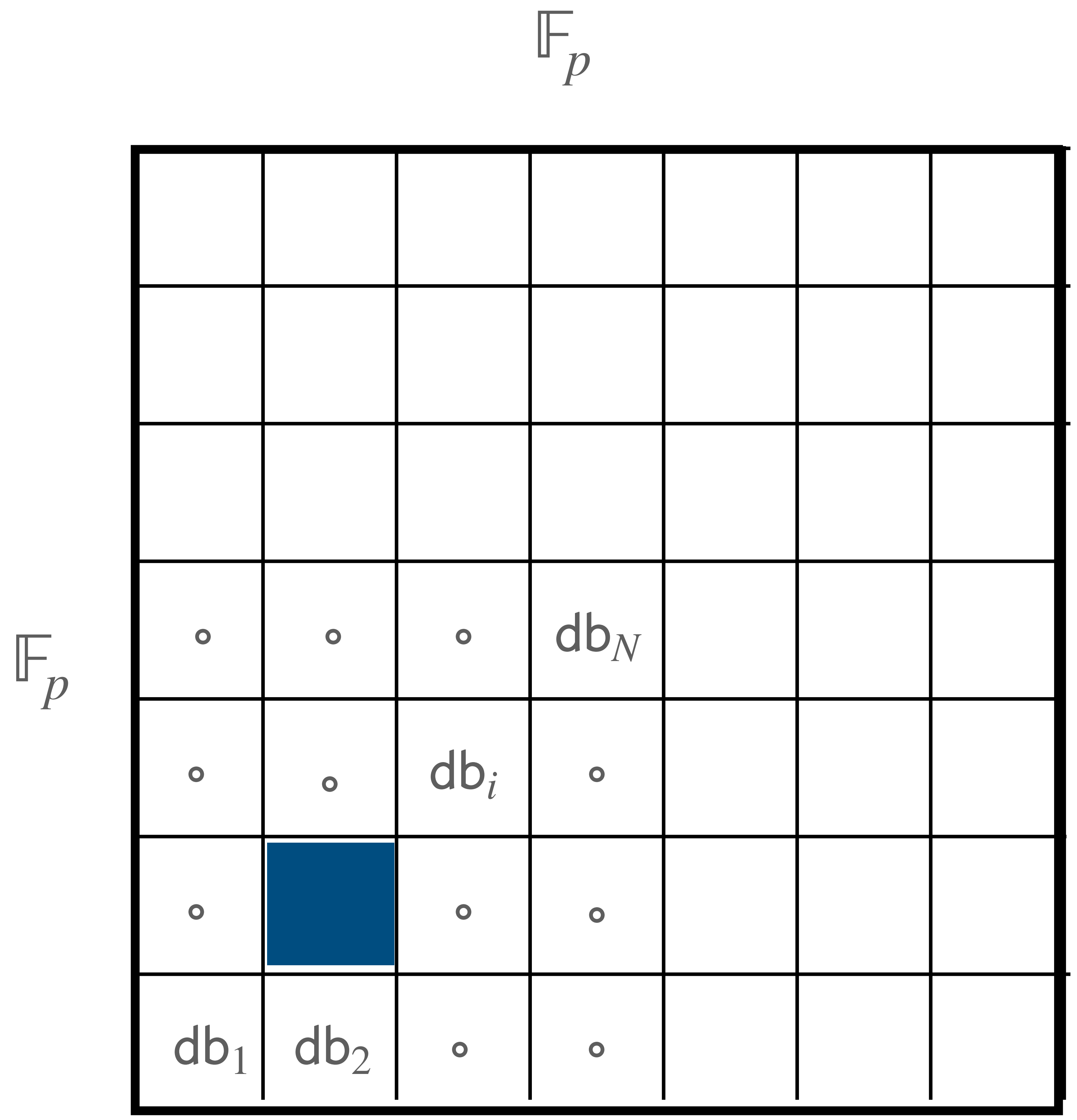
1. Want: db_j
2. $\mathsf{RM}.\mathsf{Que}(j) \rightarrow Q$:



Decrease decoder's success probability: test points

Modified local decoding with test queries

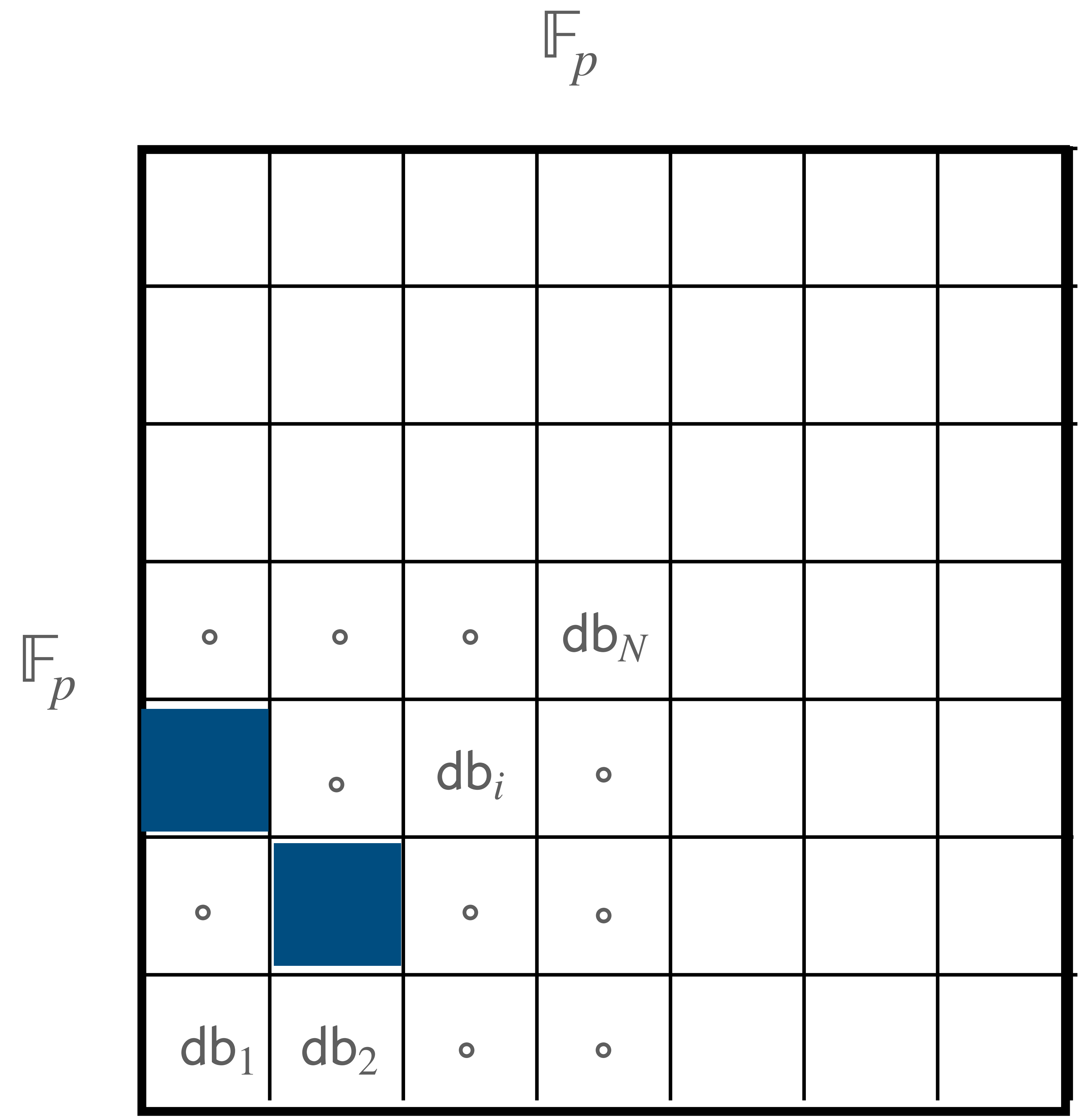
1. Want: db_j
2. $\mathsf{RM.Que}(j) \to Q$:



Decrease decoder's success probability: test points

Modified local decoding with test queries

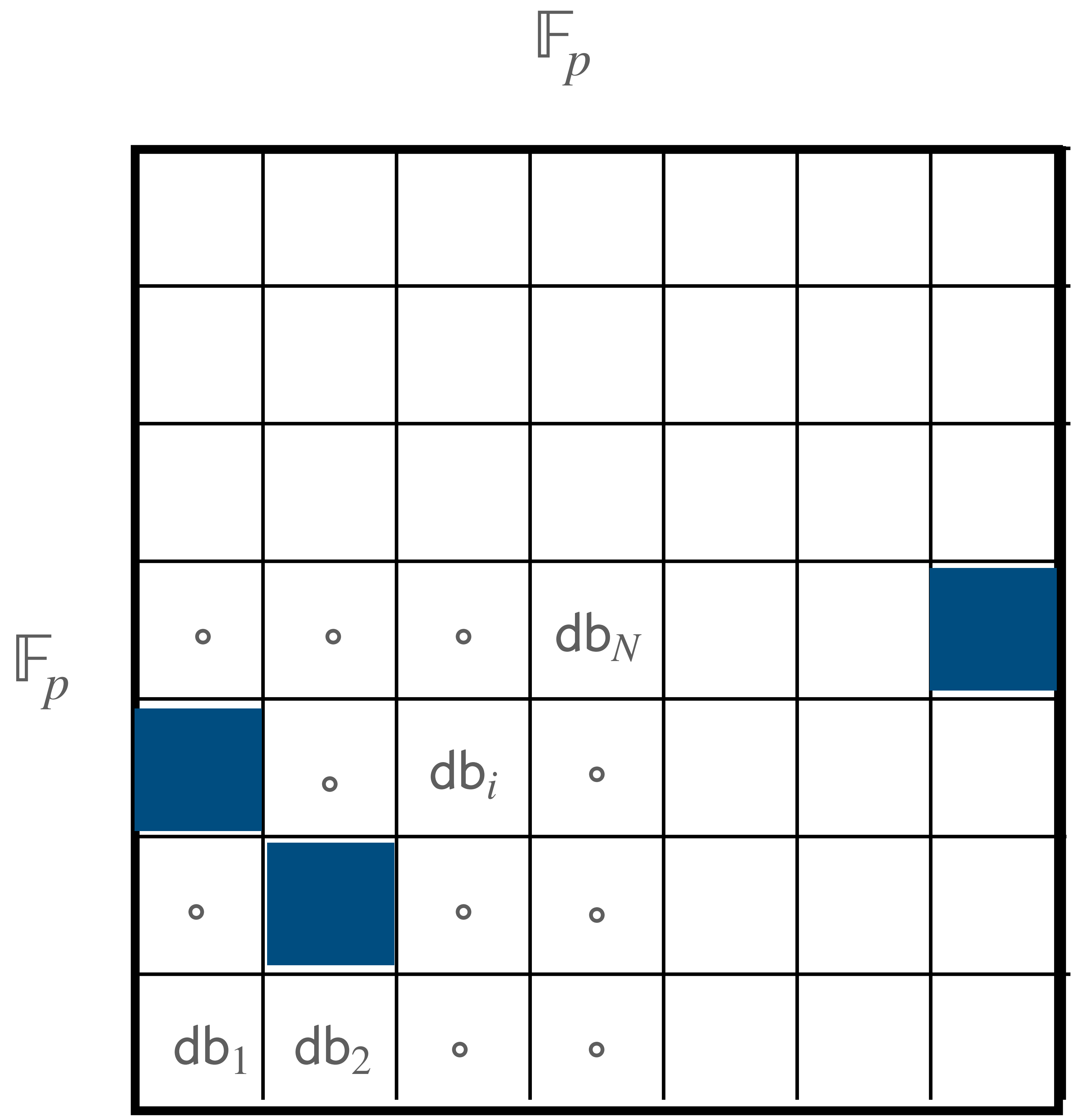
1. Want: db_j
2. $\mathsf{RM.Que}(j) \rightarrow Q$:



Decrease decoder's success probability: test points

Modified local decoding with test queries

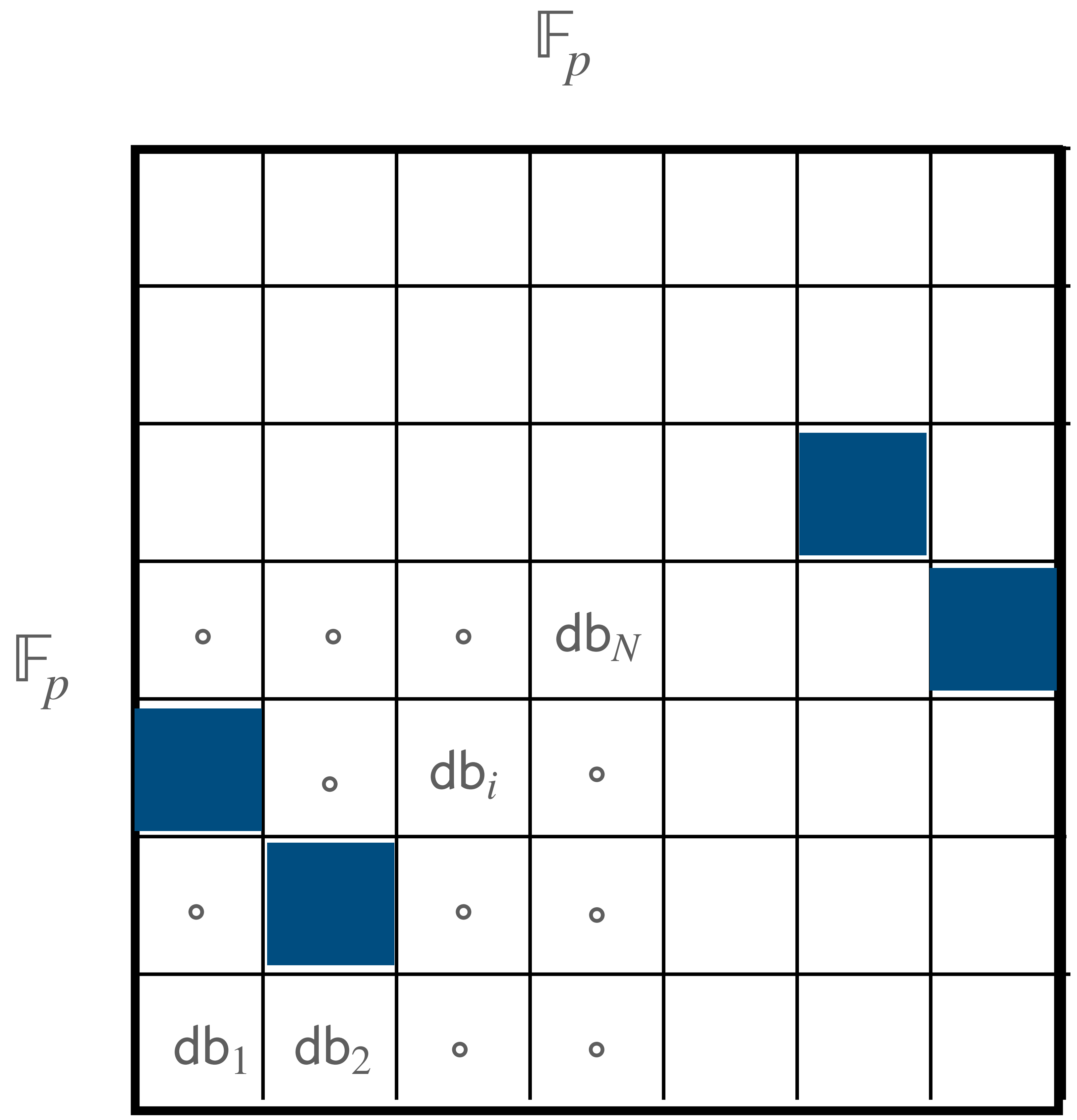
1. Want: db_j
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$:



Decrease decoder's success probability: test points

Modified local decoding with test queries

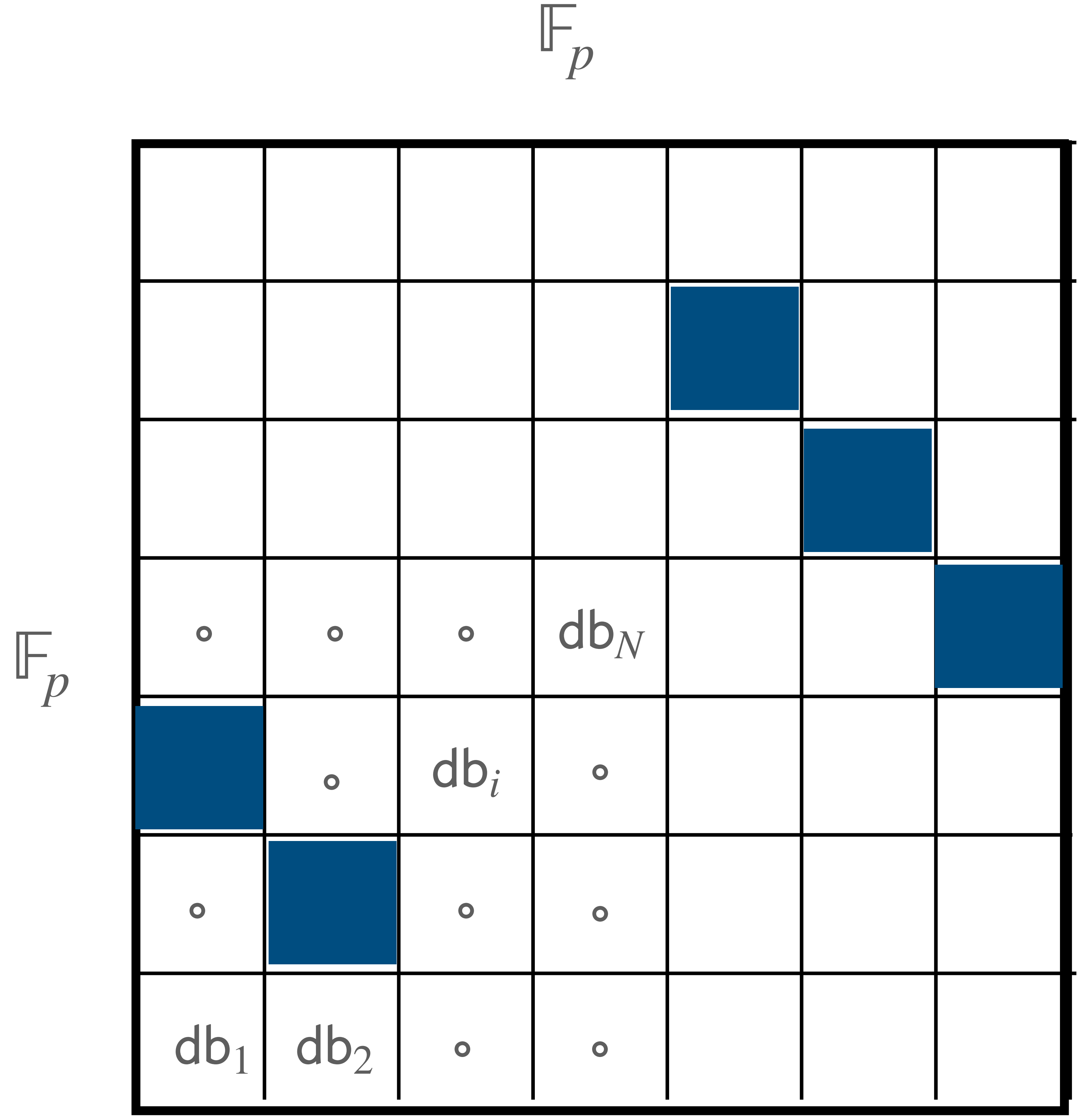
1. Want: db_j
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$:



Decrease decoder's success probability: test points

Modified local decoding with test queries

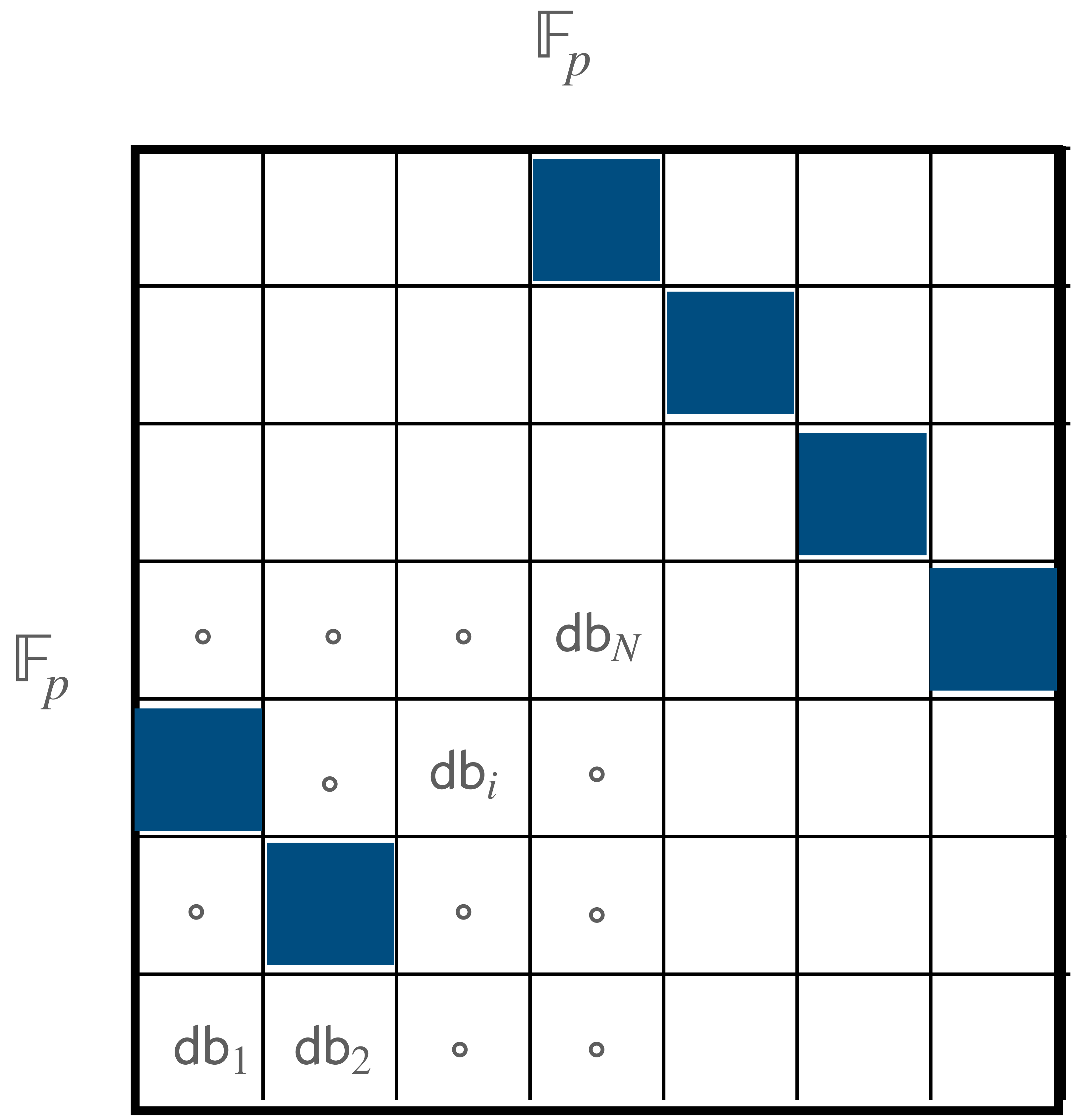
1. Want: db_j
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$:



Decrease decoder's success probability: test points

Modified local decoding with test queries

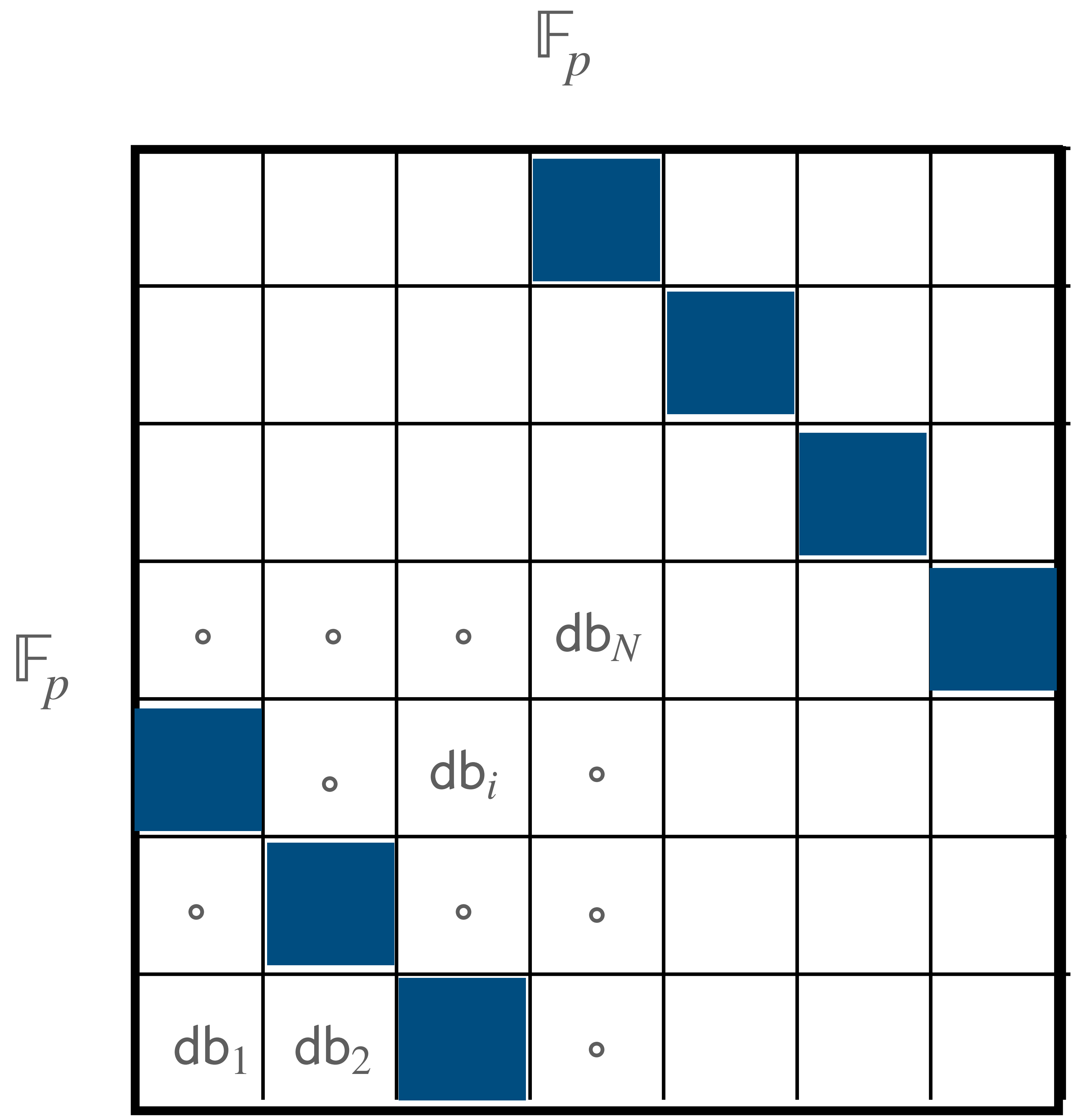
1. Want: db_j
2. $\mathrm{RM}.\mathrm{Que}(j) \to Q$:



Decrease decoder's success probability: test points

Modified local decoding with test queries

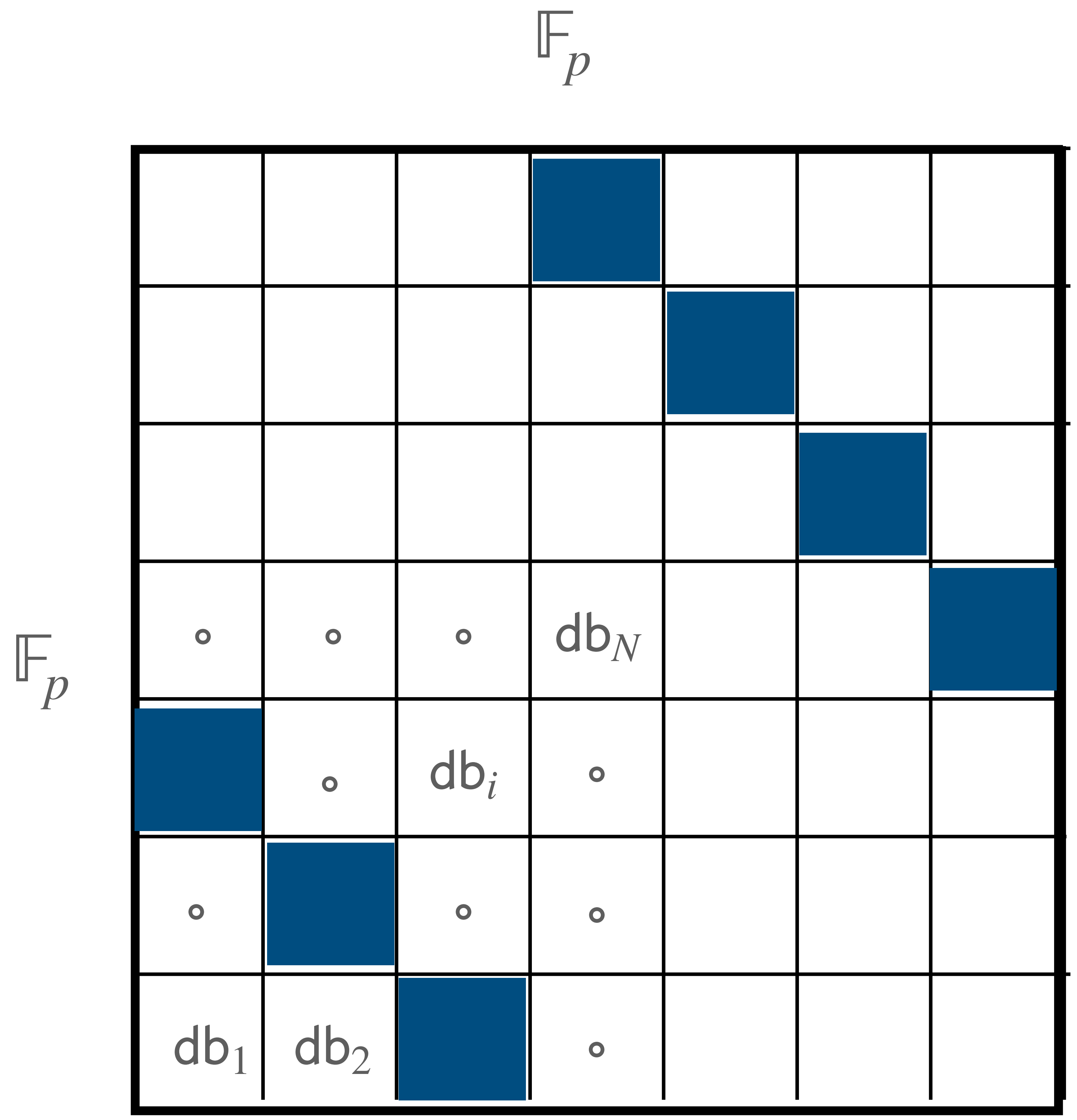
1. Want: db_j
2. $\mathsf{RM.Que}(j) \to Q$:



Decrease decoder's success probability: test points

Modified local decoding with test queries

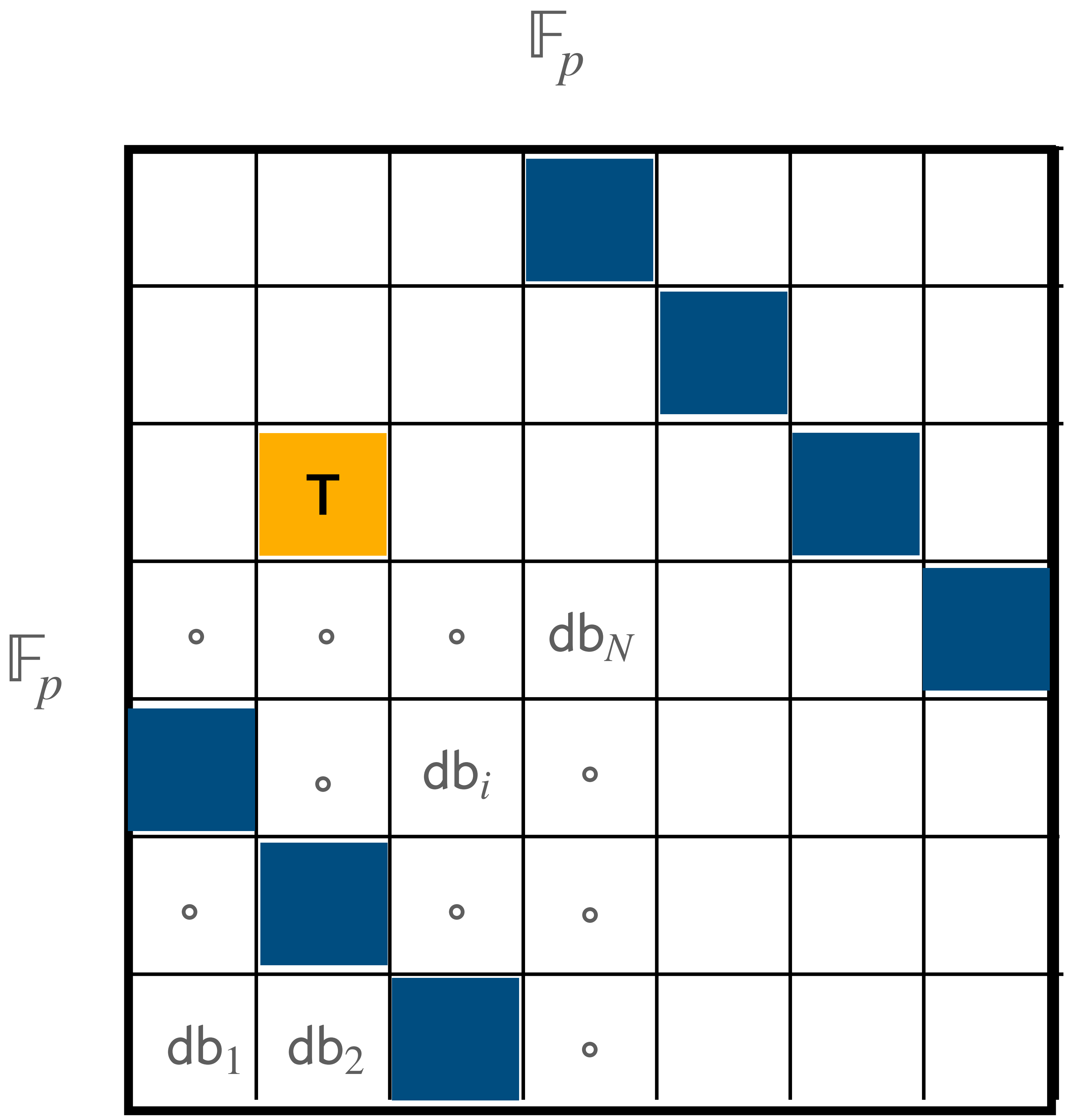
1. Want: db_j
2. $\mathsf{RM.Que}(j) \to Q$:
   1. let $L = L_1, \ldots, L_t$ be random lines through db_j.



Decrease decoder's success probability: test points

Modified local decoding with test queries

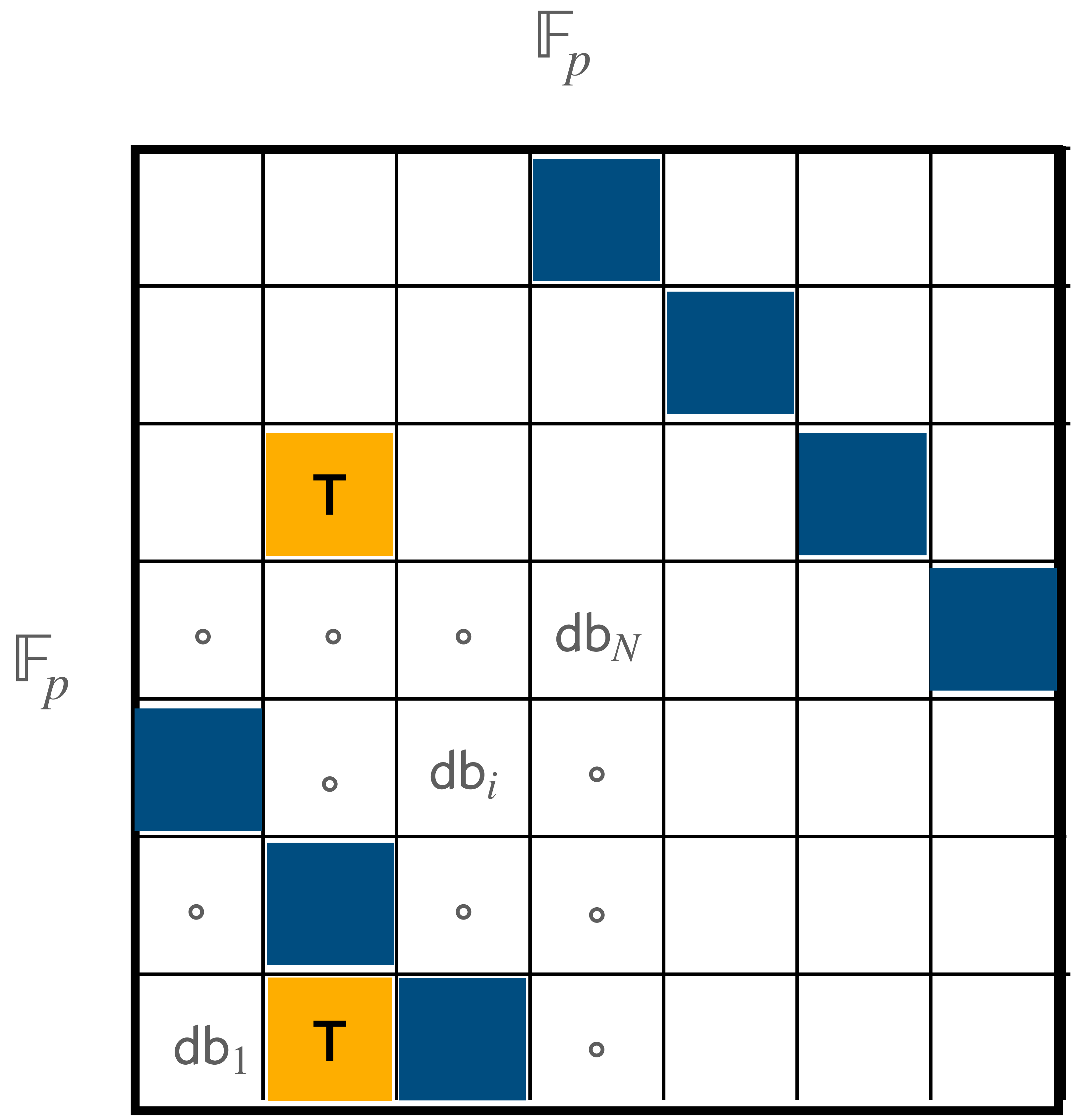
1. Want: db_j
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$:
   1. let $L = L_1, \ldots, L_t$ be random lines through db_j.



Decrease decoder's success probability: test points

Modified local decoding with test queries

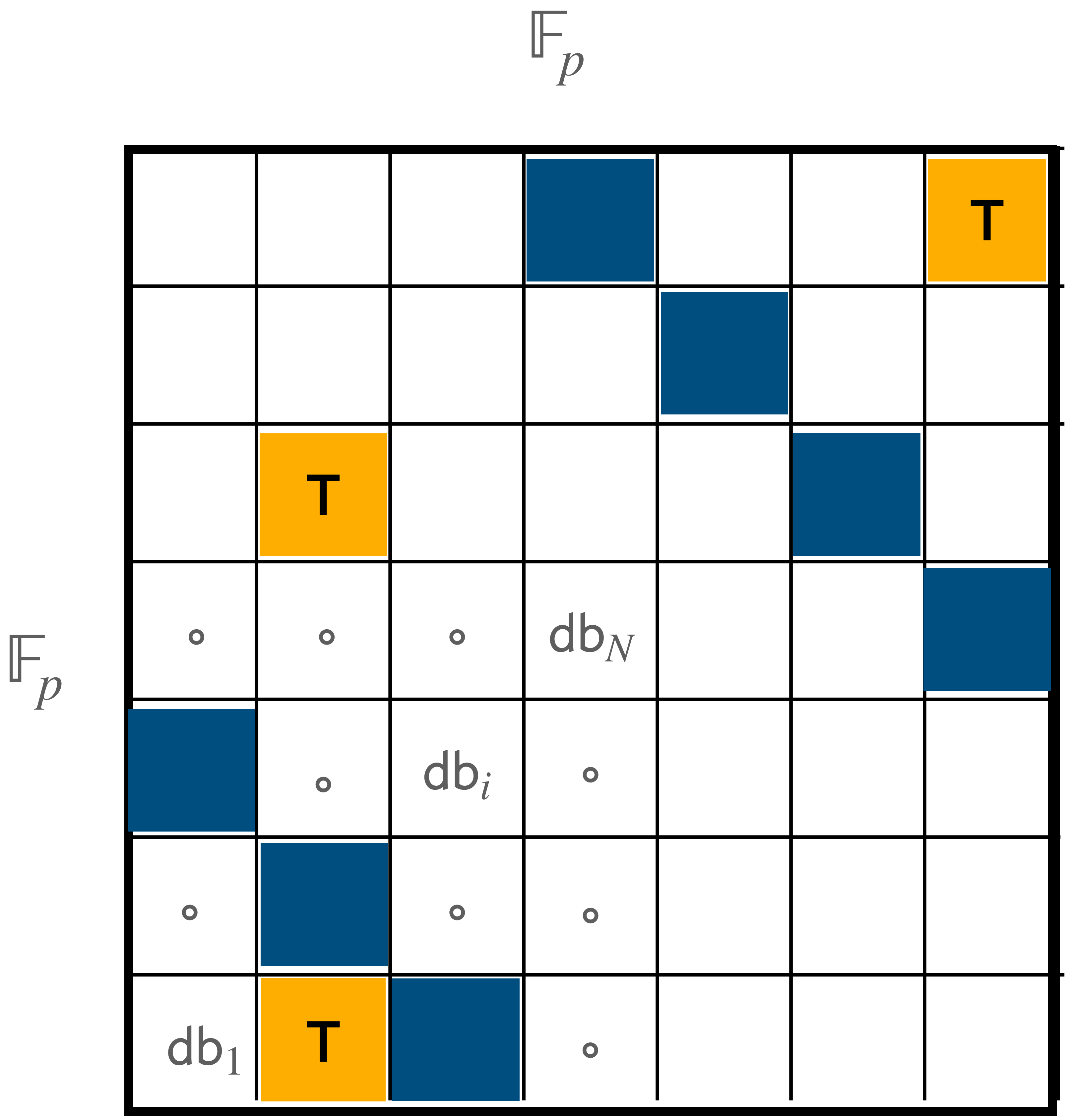
1. Want: db_j
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$:
    1. let $L = L_1, \ldots, L_t$ be random lines through db_j.



Decrease decoder's success probability: test points

Modified local decoding with test queries

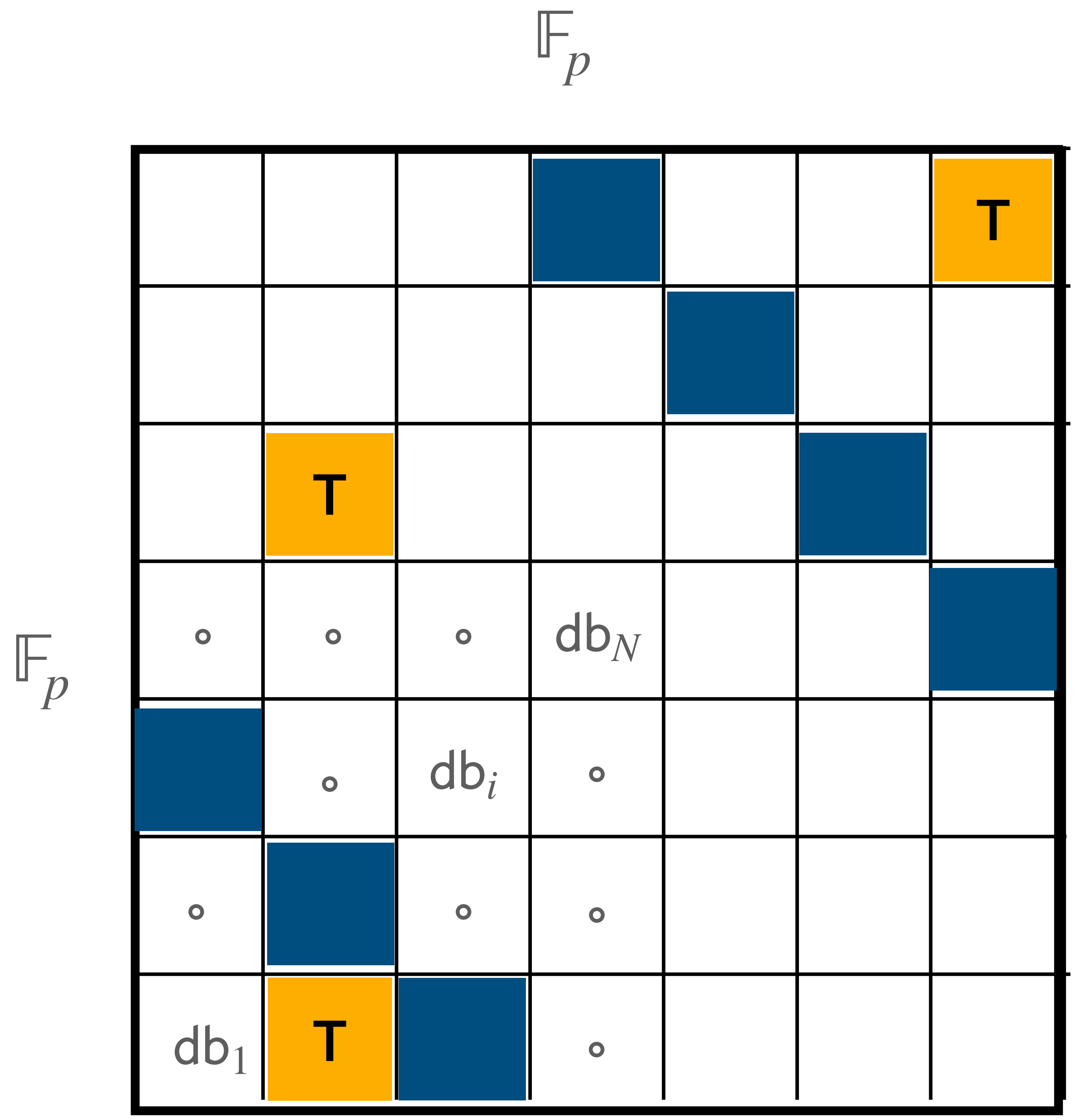
1. Want: db_j
2. $\mathsf{RM}.\mathsf{Que}(j) \rightarrow Q$:
   1. let $L = L_1, \ldots, L_t$ be random lines through db_j.



Decrease decoder's success probability: test points

Modified local decoding with test queries

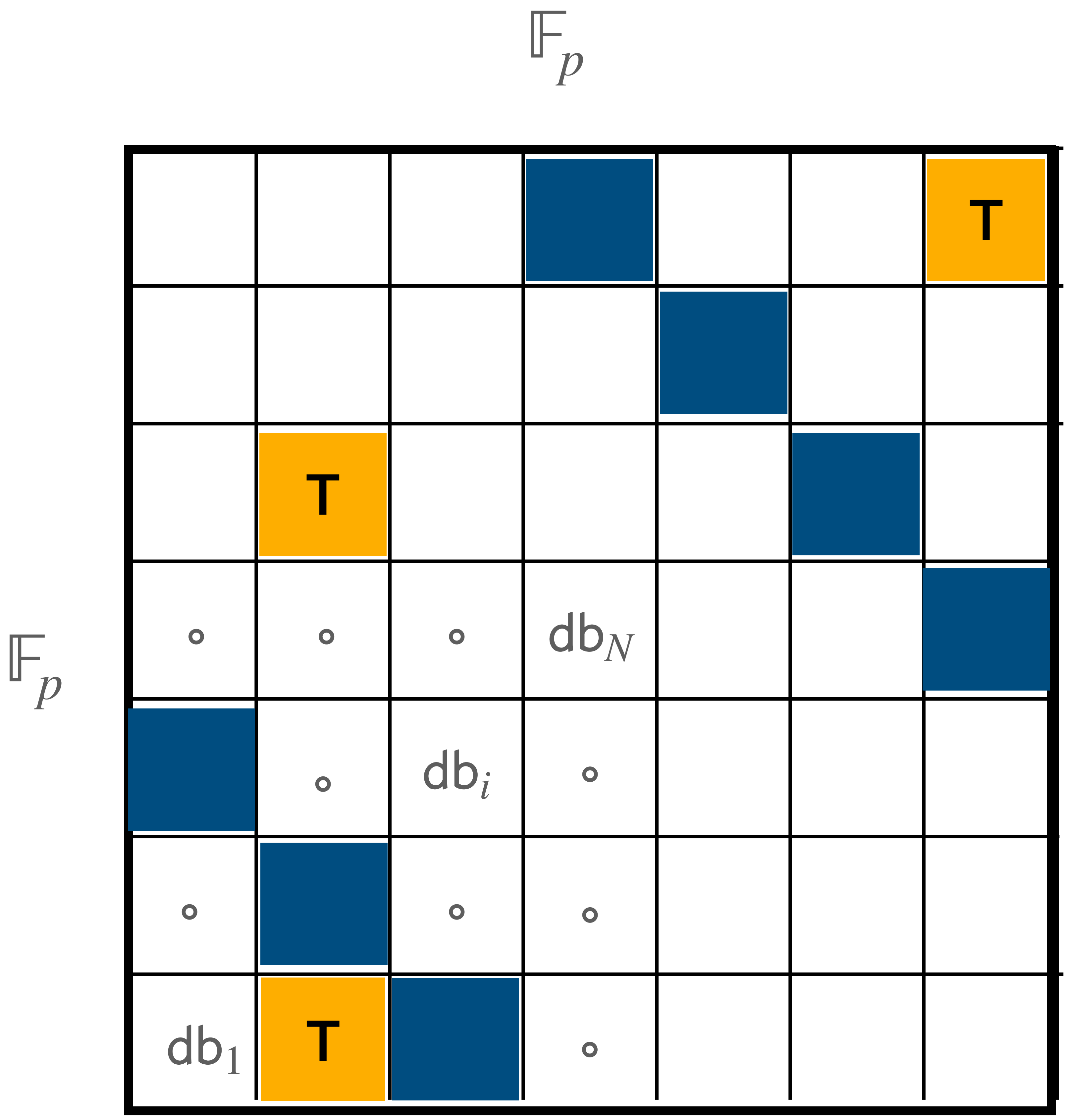
1. Want: db_j
2. $\mathsf{RM.Que}(j) \rightarrow Q$:
   1. let $L = L_1, \ldots, L_t$ be random lines through db_j.
   2. let T be a set of random points.



Decrease decoder's success probability: test points

Modified local decoding with test queries

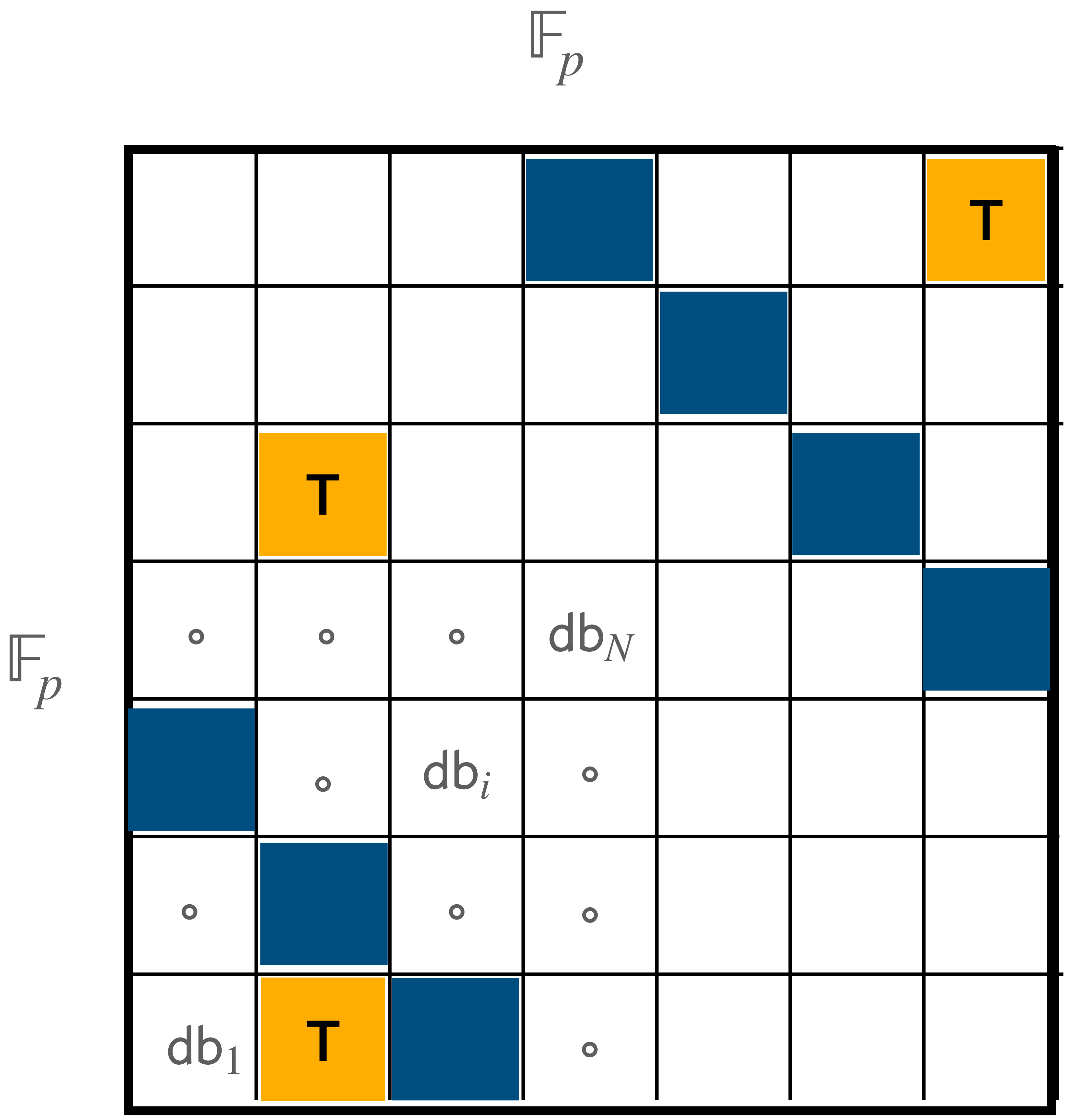
1. Want: db_j
2. $\mathsf{RM.Que}(j) \rightarrow Q$:
   1. let $L = L_1, \ldots, L_t$ be random lines through db_j.
   2. let T be a set of random points.
   3. let $Q = L \cup T$.



Decrease decoder's success probability: test points

Modified local decoding with test queries

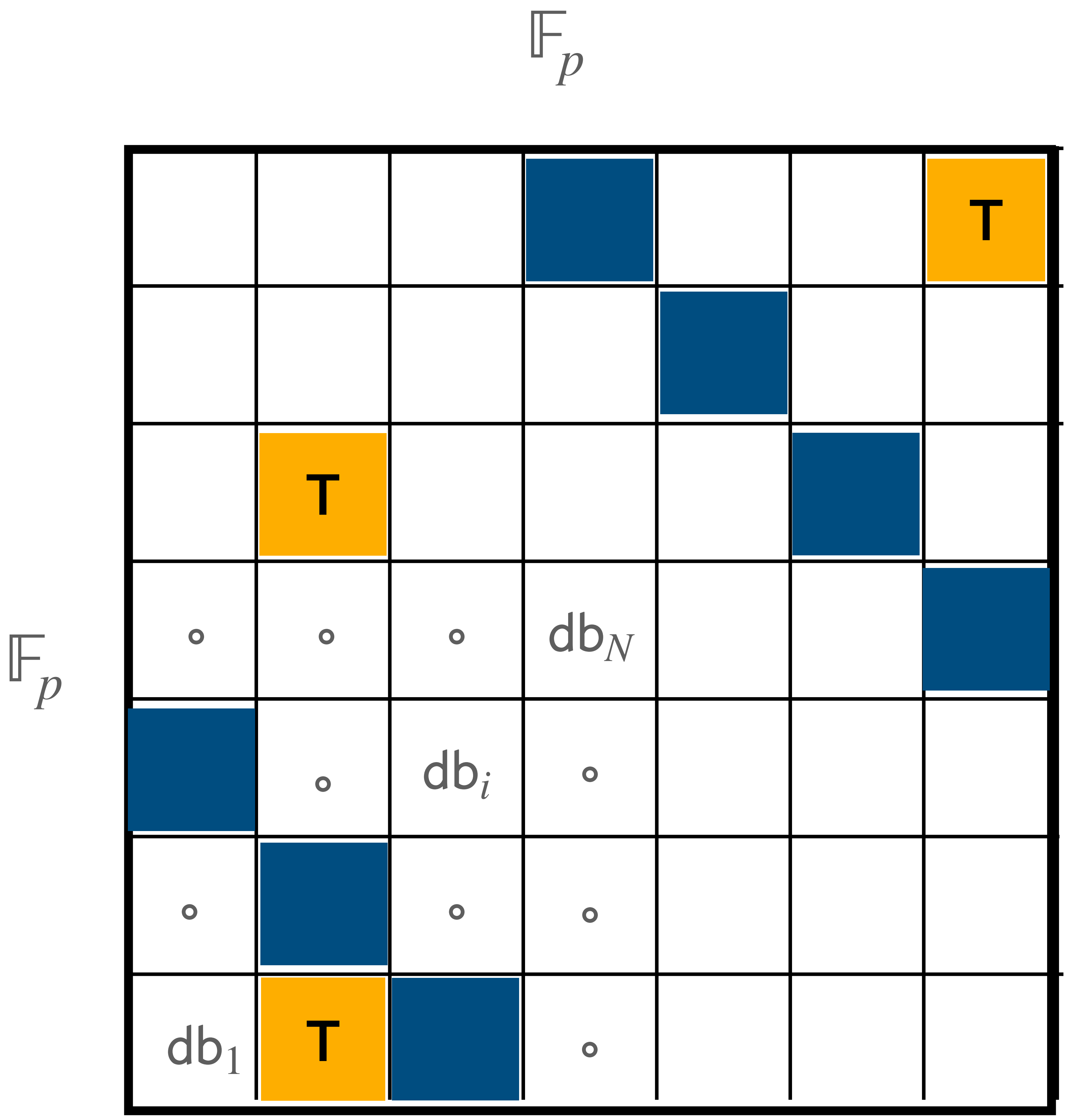
1. Want: db_j
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$:
   1. let $L = L_1, \ldots, L_t$ be random lines through db_j.
   2. let T be a set of random points.
   3. let $Q = L \cup T$.
3. $\mathsf{RM}.\mathsf{Dec}(E_Q) \to \mathsf{db}_j$:



Decrease decoder's success probability: test points

Modified local decoding with test queries

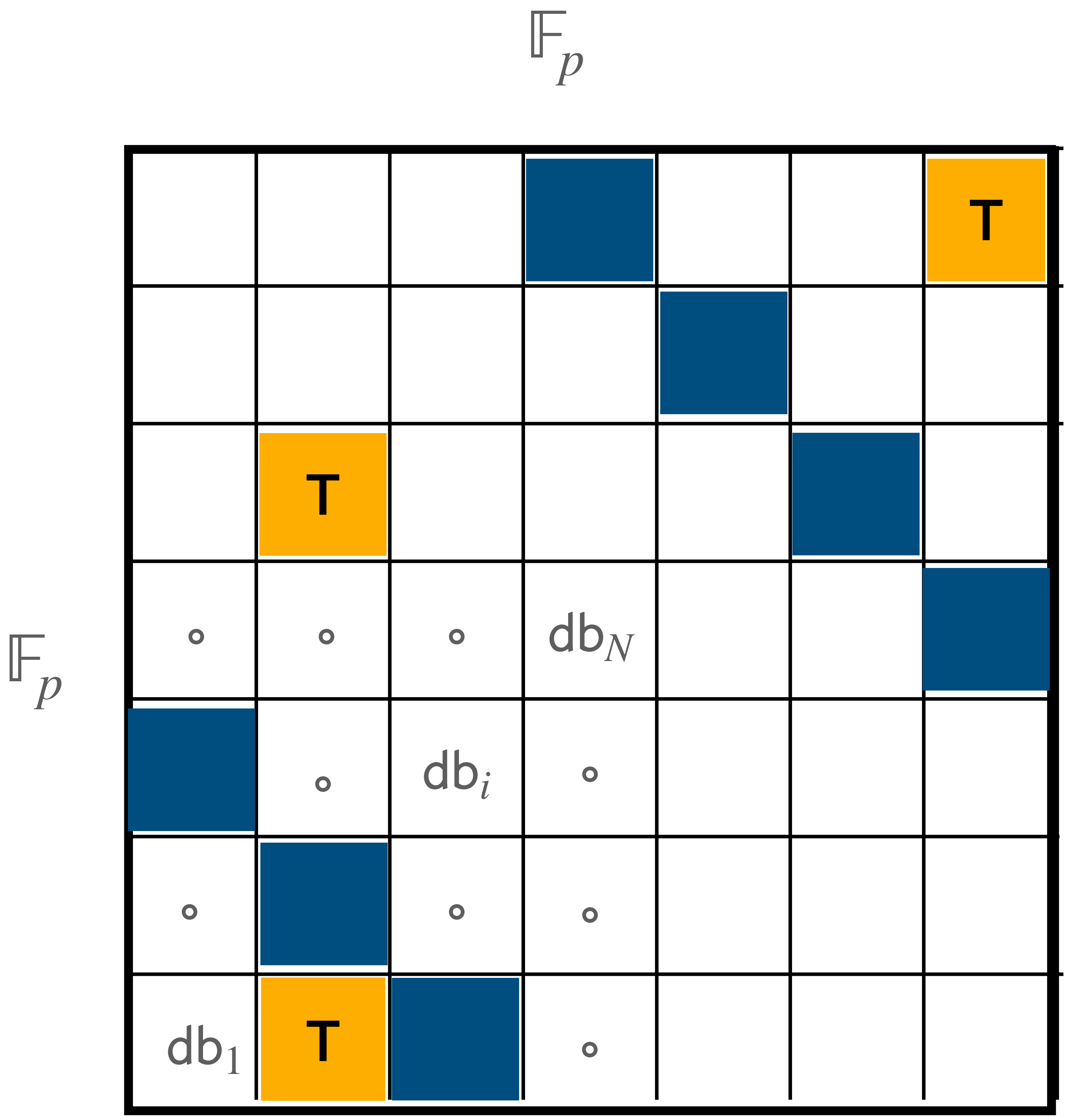
1. Want: db_j
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$:
   1. let $L = L_1, \ldots, L_t$ be random lines through db_j.
   2. let T be a set of random points.
   3. let $Q = L \cup T$.
3. $\mathsf{RM}.\mathsf{Dec}(E_Q) \to \mathsf{db}_j$:
   1. If E_T is corrupt, output $\bot$



Decrease decoder's success probability: test points

Modified local decoding with test queries

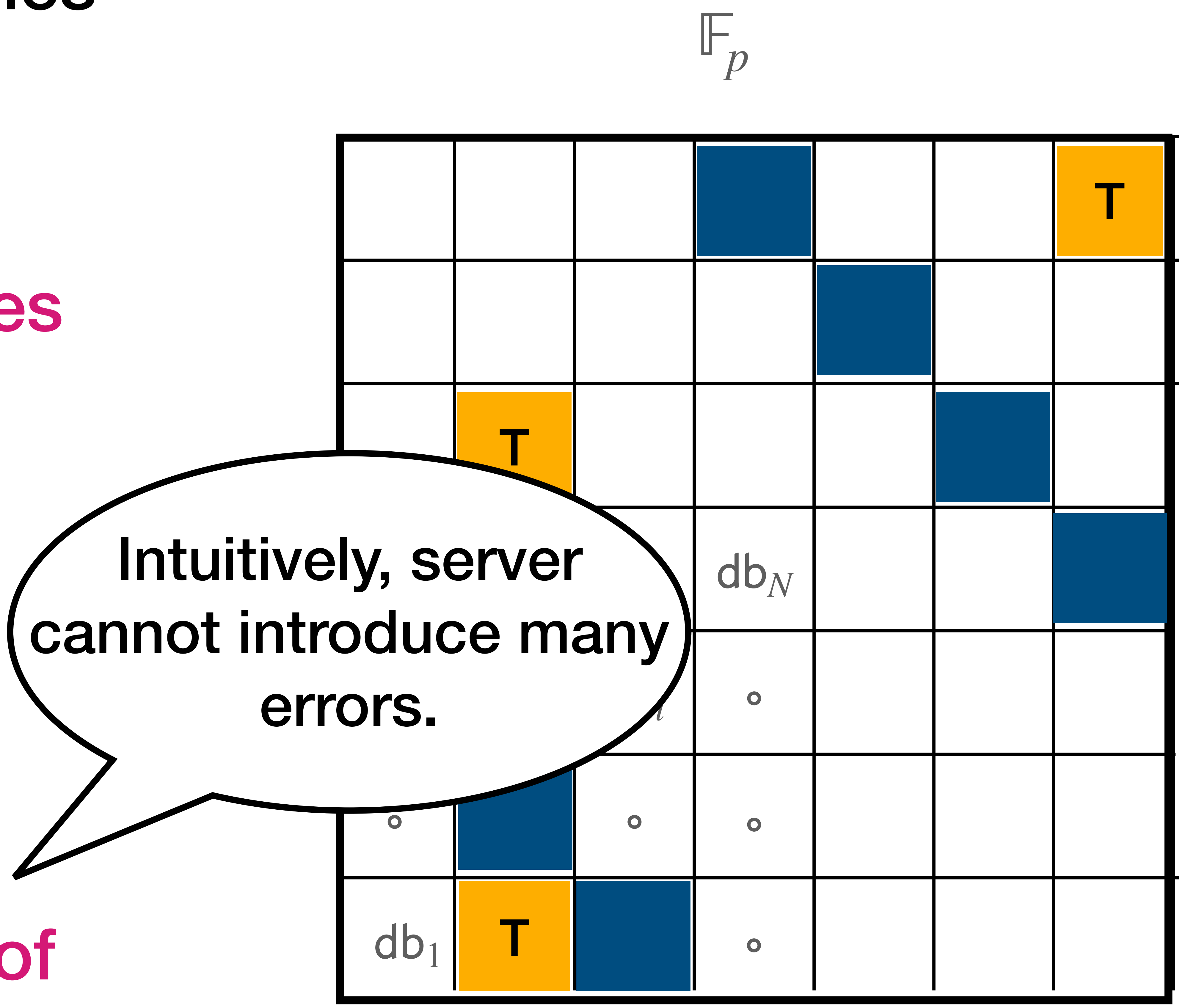
1. Want: db_j
2. $\mathrm{RM.Que}(j) \rightarrow Q$:
   1. let $L = L_1, \ldots, L_t$ be random lines through db_j.
   2. let T be a set of random points.
   3. let $Q = L \cup T$.
3. $\mathrm{RM.Dec}(E_Q) \rightarrow \mathrm{db}_j$:
   1. If E_T is corrupt, output $\perp$
   2. Else, output majority decoding of $E_{L_1}, \ldots, E_{L_t}$.



Decrease decoder's success probability: test points

Modified local decoding with test queries

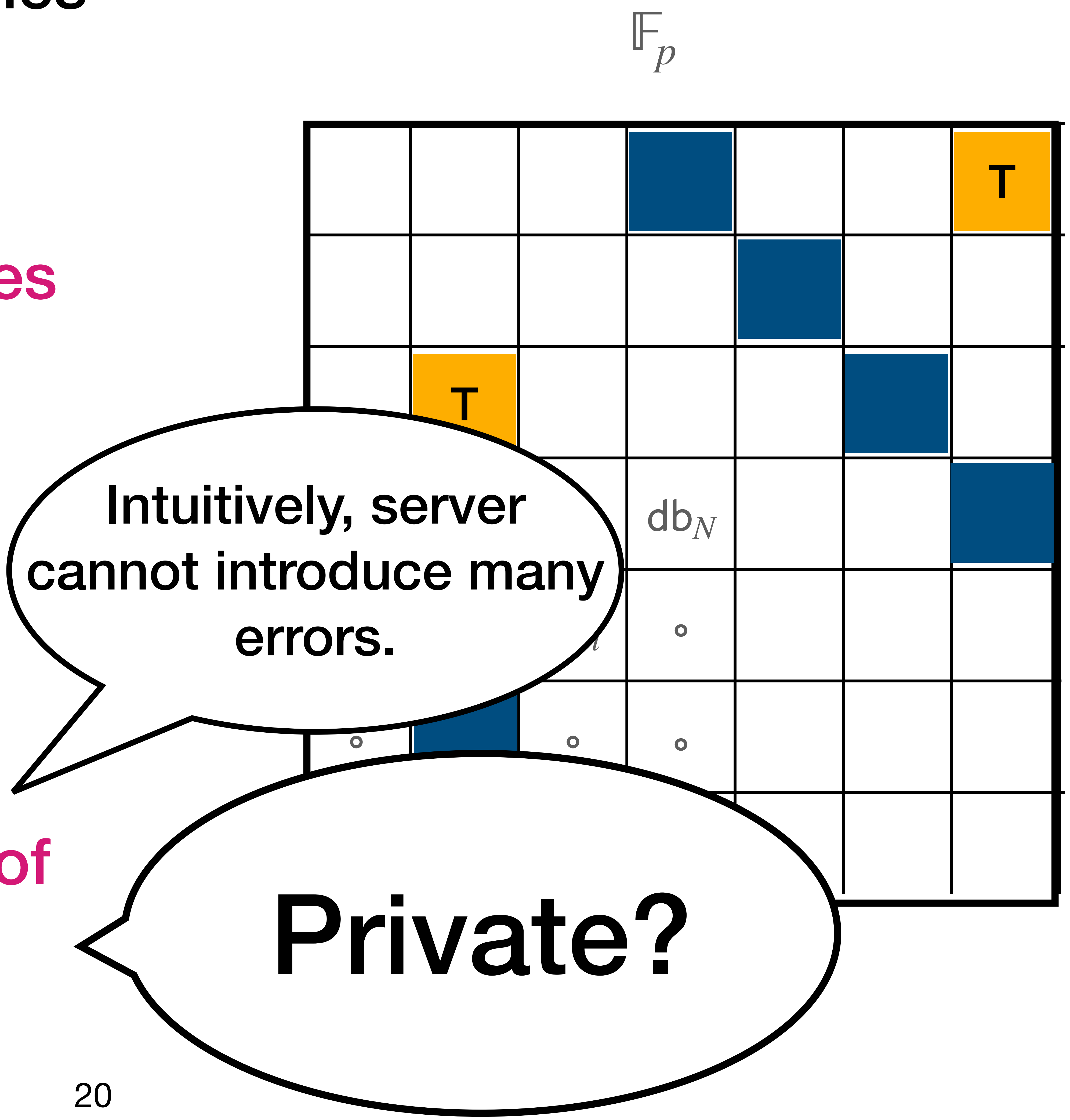
1. Want: db_j
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$:
   1. let $L = L_1, \ldots, L_t$ be random lines through db_j.
   2. let T be a set of random points.
   3. let $Q = L \cup T$.
3. $\mathsf{RM}.\mathsf{Dec}(E_Q) \to \mathsf{db}_j$:
   1. If E_T is corrupt, output $\perp$
   2. Else, output majority decoding of $E_{L_1}, \ldots, E_{L_t}$.



Decrease decoder's success probability: test points

Modified local decoding with test queries

1. Want: db_j
2. $\mathsf{RM.Que}(j) \to Q$:
   1. let $L = L_1, \ldots, L_t$ be random lines through db_j.
   2. let T be a set of random points.
   3. let $Q = L \cup T$.
3. $\mathsf{RM.Dec}(E_Q) \to \mathsf{db}_j$:
   1. If E_T is corrupt, output $\perp$
   2. Else, output majority decoding of $E_{L_1}, \ldots, E_{L_t}$.



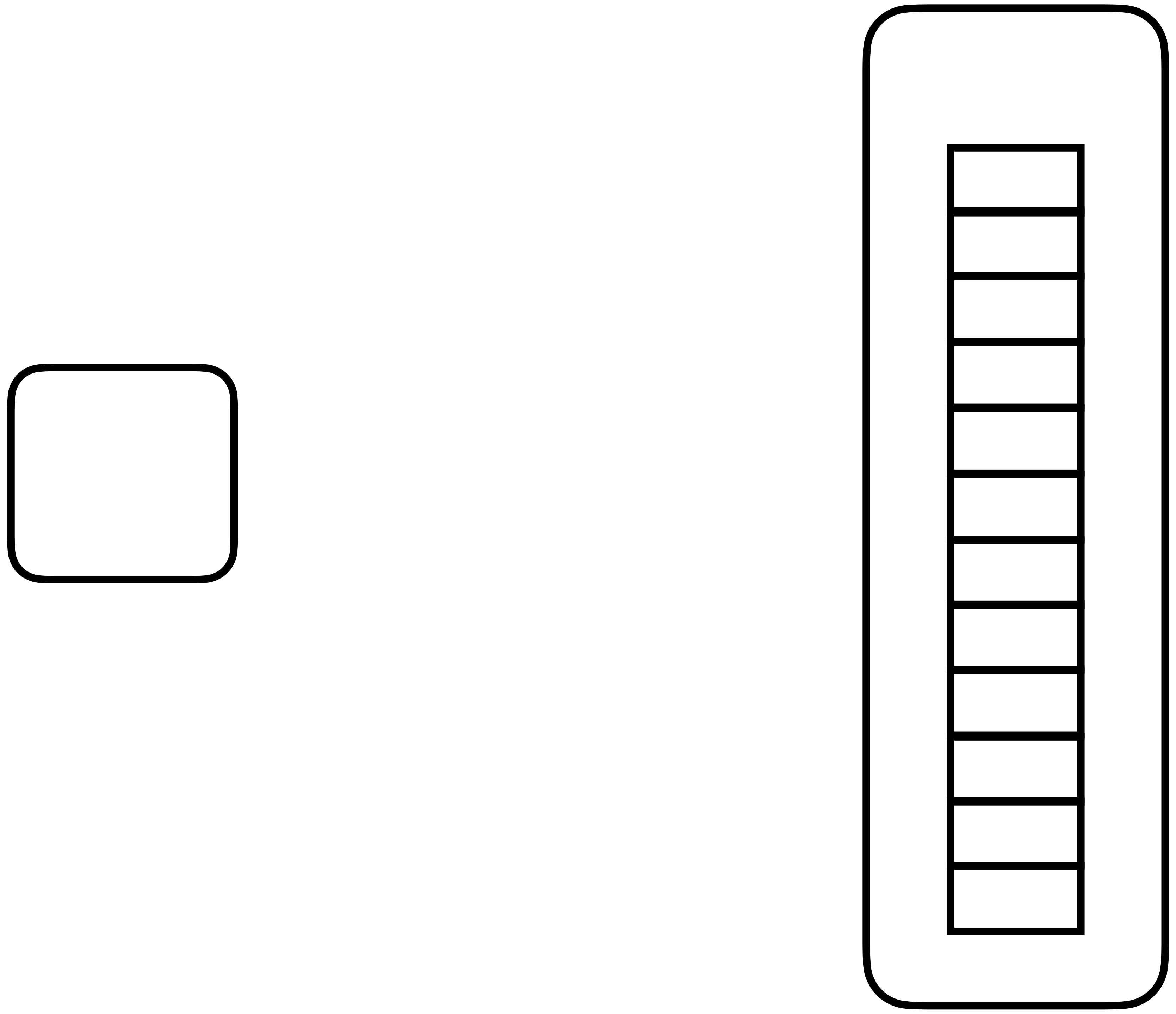
The non-signaling barrier

The non-signaling barrier

- What guarantee does PIR privacy give us on multiple queries?

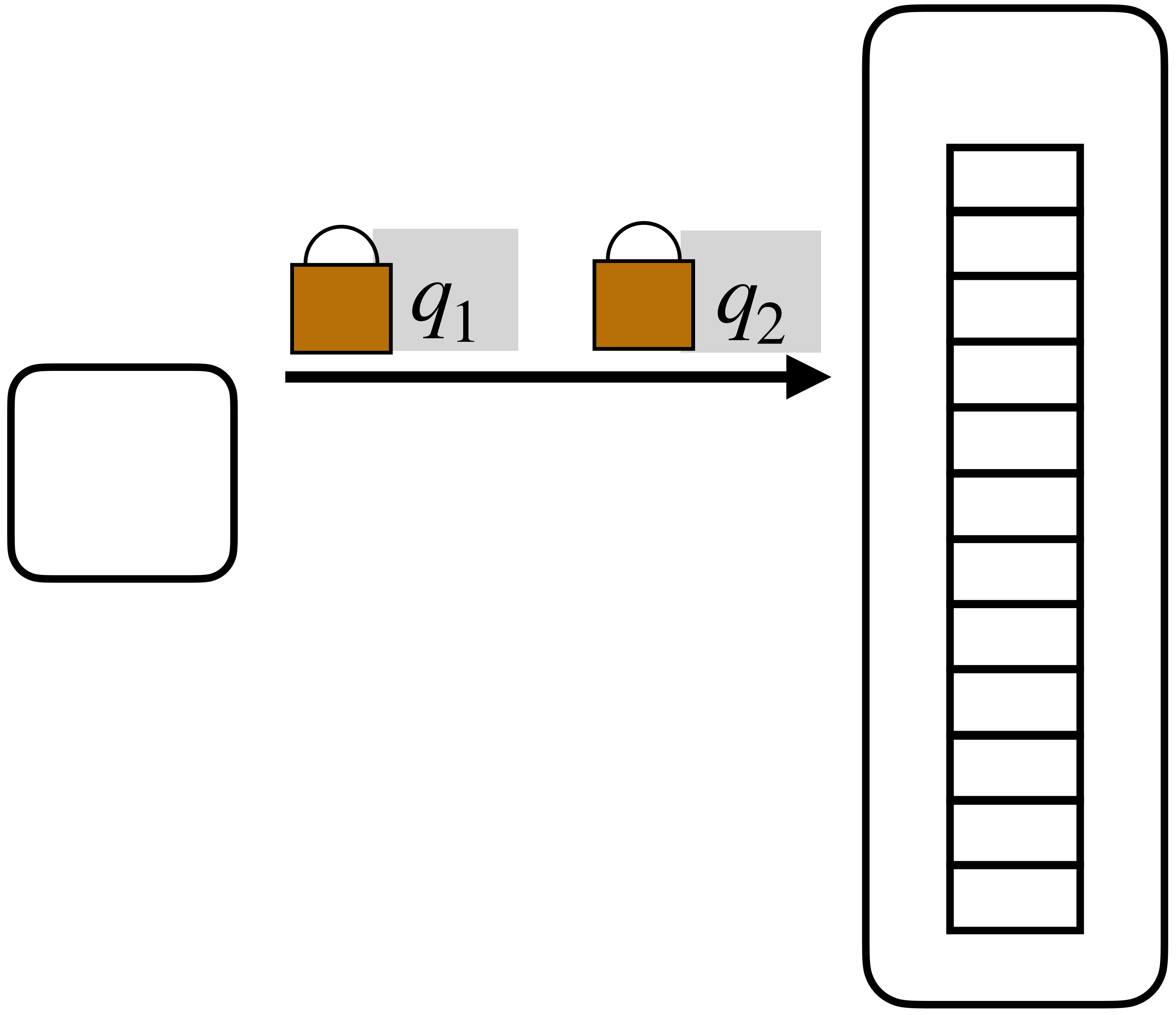
The non-signaling barrier

- What guarantee does PIR privacy give us on multiple queries?



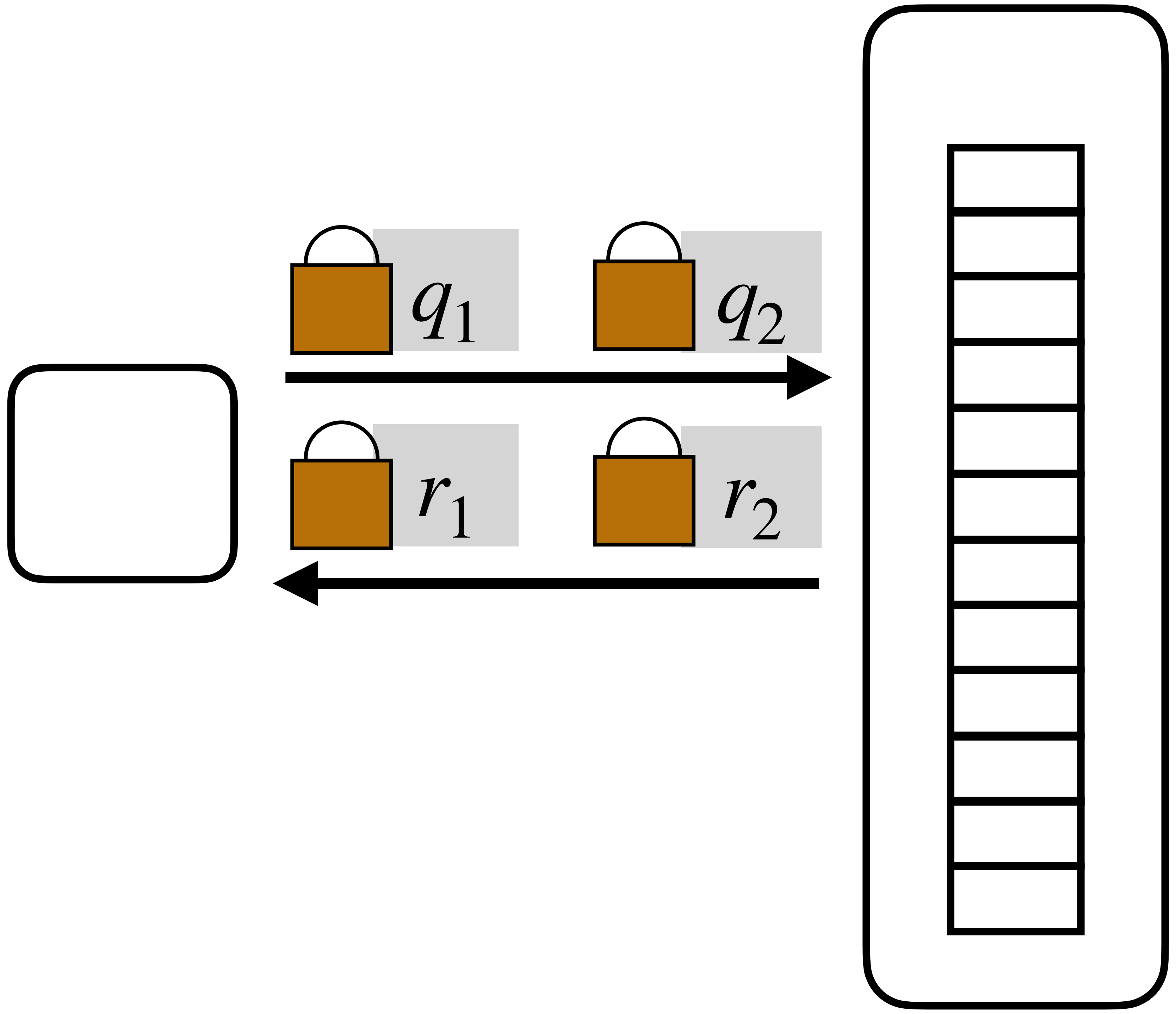
The non-signaling barrier

- What guarantee does PIR privacy give us on multiple queries?



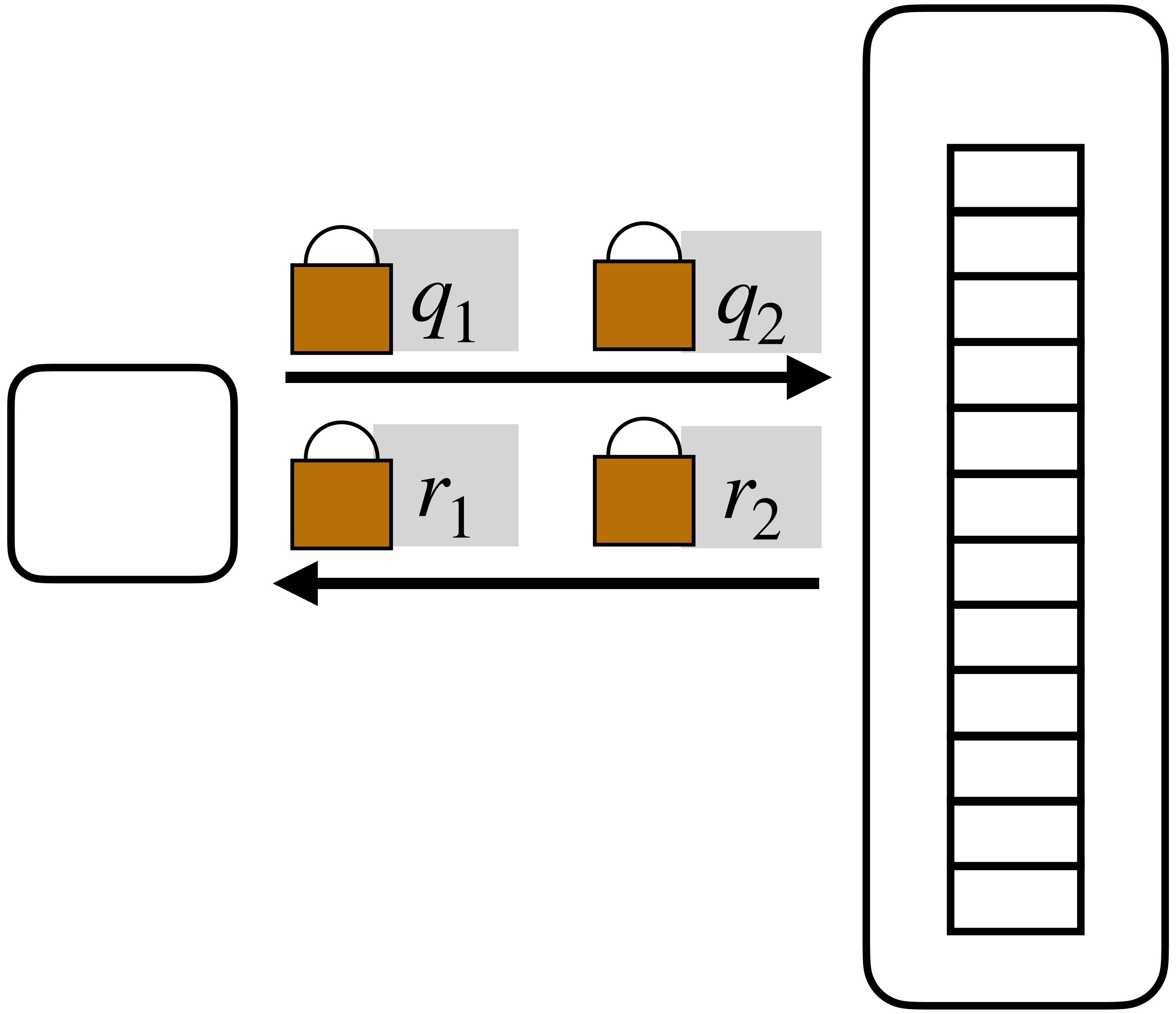
The non-signaling barrier

- What guarantee does PIR privacy give us on multiple queries?



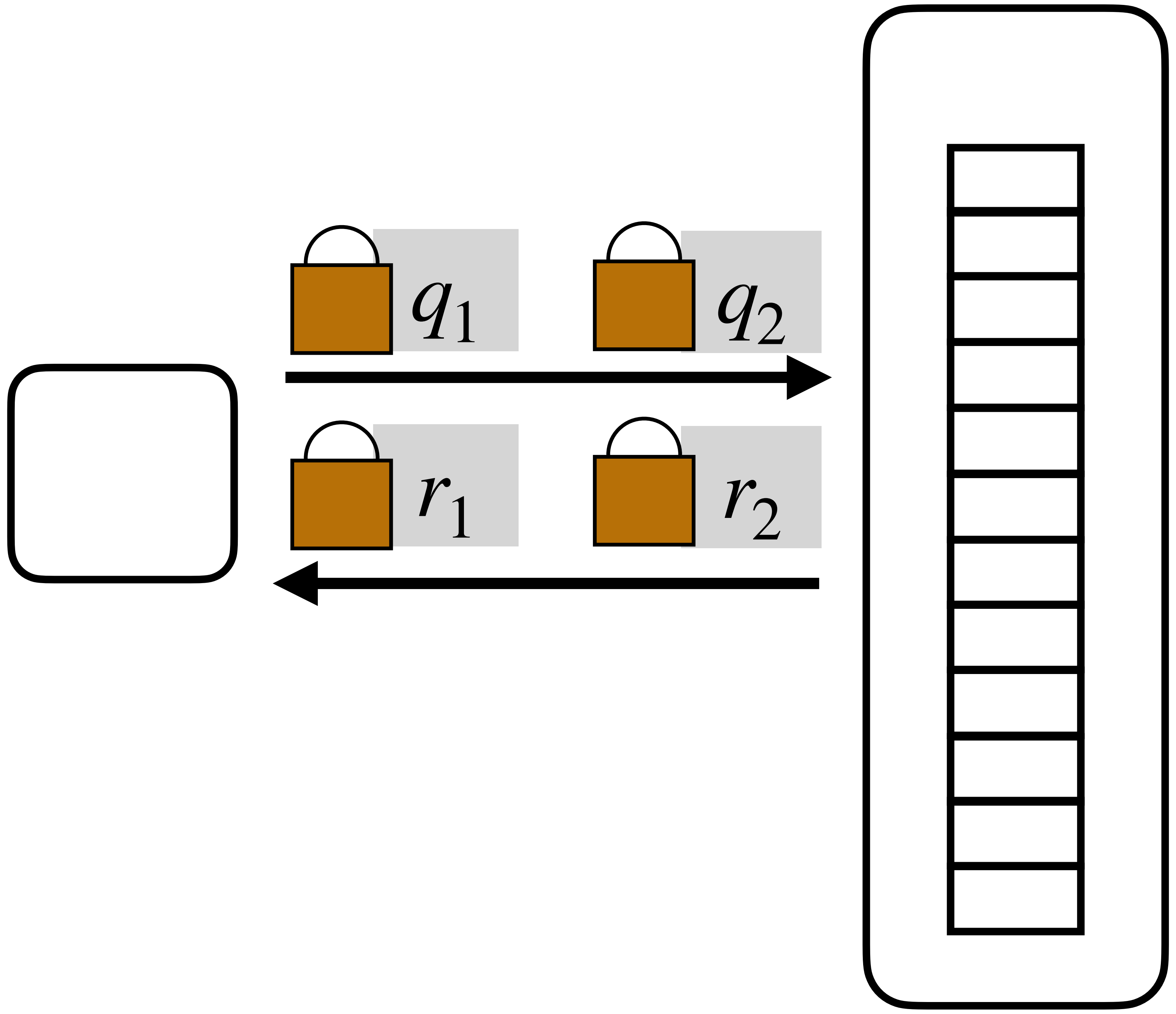
The non-signaling barrier

- What guarantee does PIR privacy give us on multiple queries?
  - Response i is independent of query j?



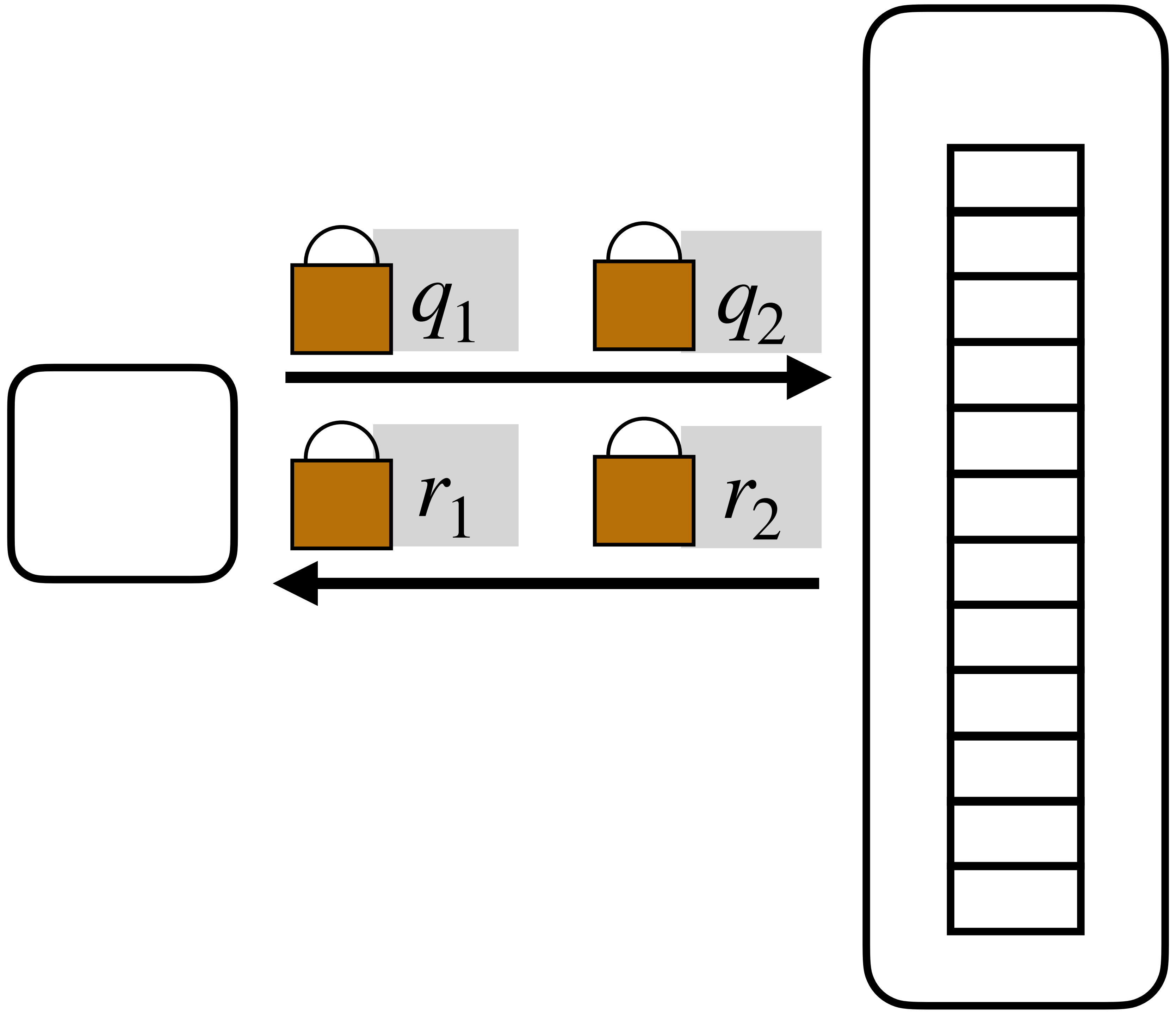
The non-signaling barrier

- What guarantee does PIR privacy give us on multiple queries?
  - Response i is independent of query j?
  - Don't know how to prove this strong guarantee.



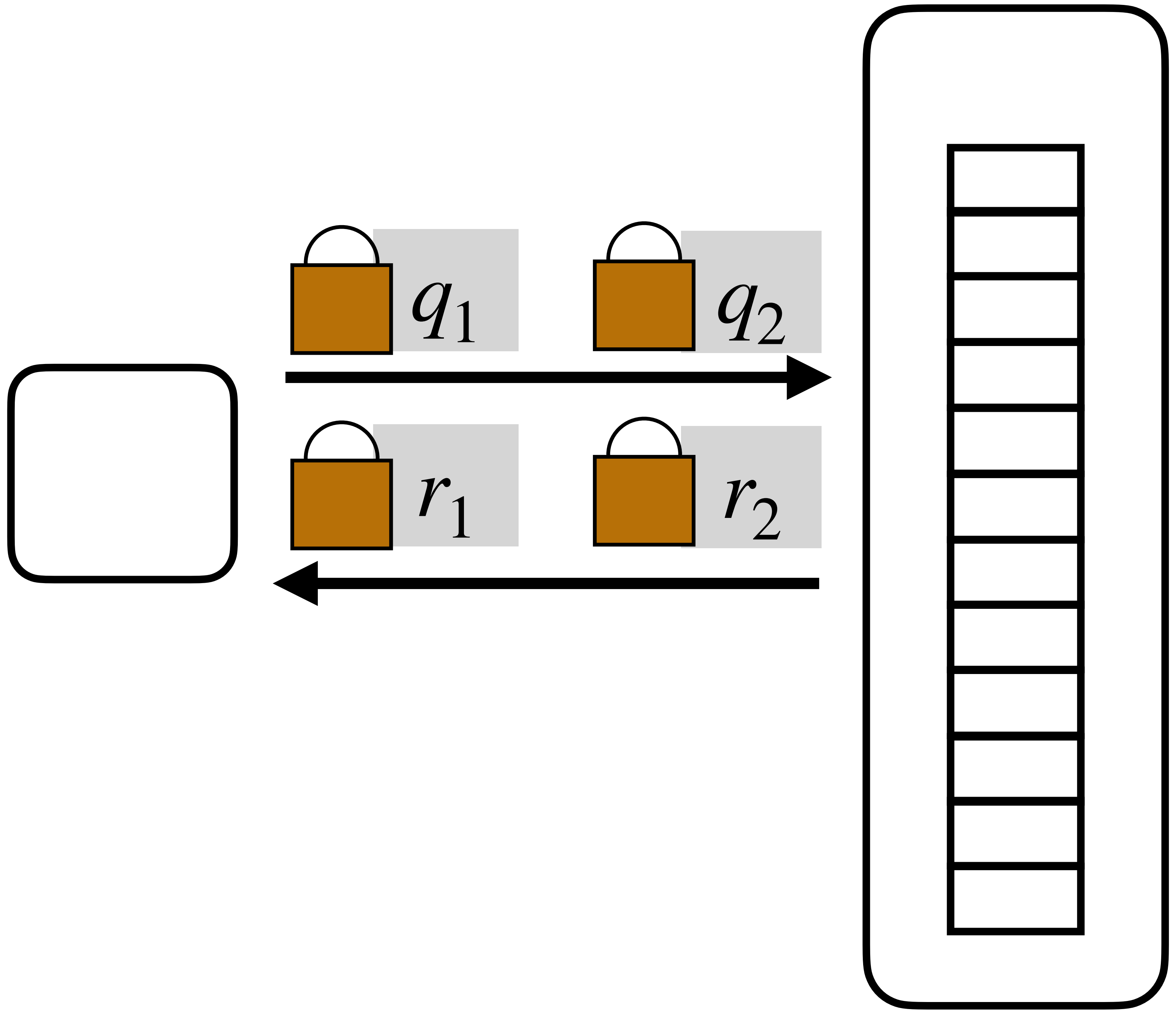
The non-signaling barrier

- What guarantee does PIR privacy give us on multiple queries?
  - Response i is independent of query j?
  - Don't know how to prove this strong guarantee.
- Problem: PIR guarantees that response for i does not "leak information" about query j, but may have "non-signaling" correlations with query j.



The non-signaling barrier

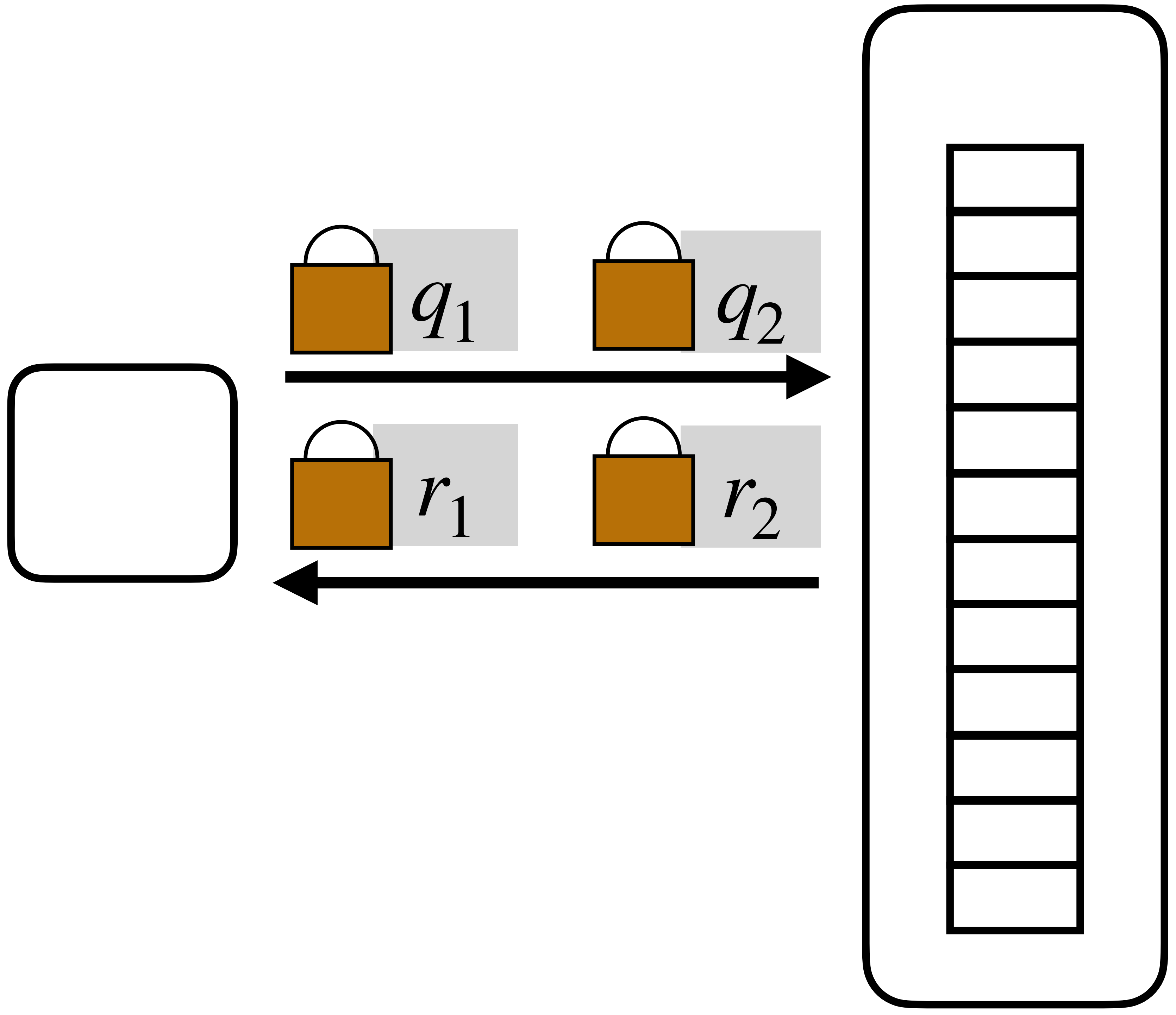
- What guarantee does PIR privacy give us on multiple queries?
  - Response i is independent of query j?
  - Don’t know how to prove this strong guarantee.
- Problem: PIR guarantees that response for i does not “leak information” about query j, but may have “non-signaling” correlations with query j.
  - weaker than “independent responses!”



The non-signaling barrier

- What guarantee does PIR privacy give us on multiple queries?
  - Response i is independent of query j?
  - Don't know how to prove this strong guarantee.
- Problem: PIR guarantees that response for i does not "leak information" about query j, but may have "non-signaling" correlations with query j.
  - weaker than "independent responses!"

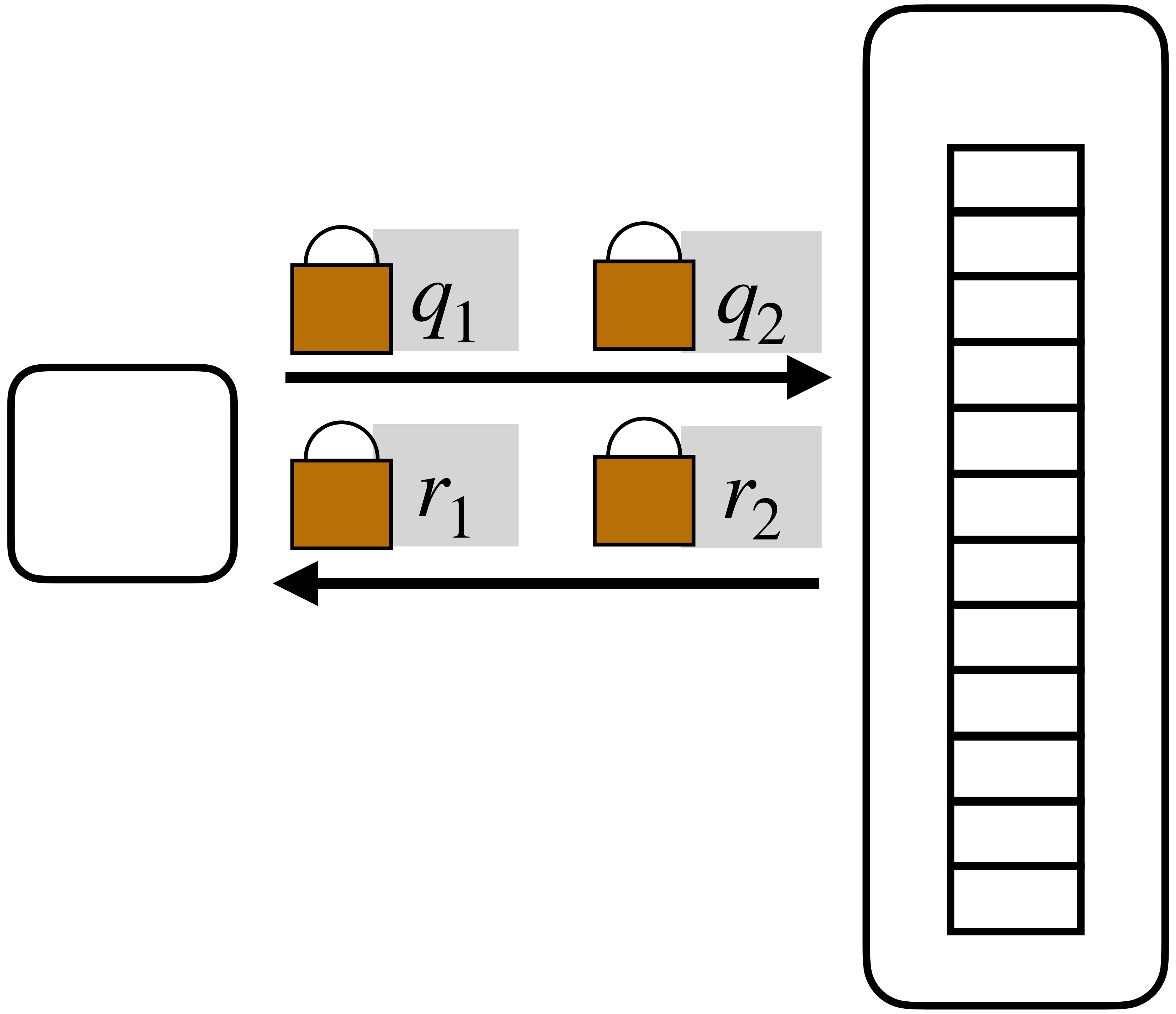
- Are non-signaling correlations actually a problem?



The non-signaling barrier

- What guarantee does PIR privacy give us on multiple queries?
  - Response i is independent of query j?
  - Don't know how to prove this strong guarantee.
- Problem: PIR guarantees that response for i does not "leak information" about query j, but may have "non-signaling" correlations with query j.
  - weaker than "independent responses!"

- Are non-signaling correlations actually a problem?
  - Can potentially allow adversary to differentiate between test and decoding queries — can't prove security.

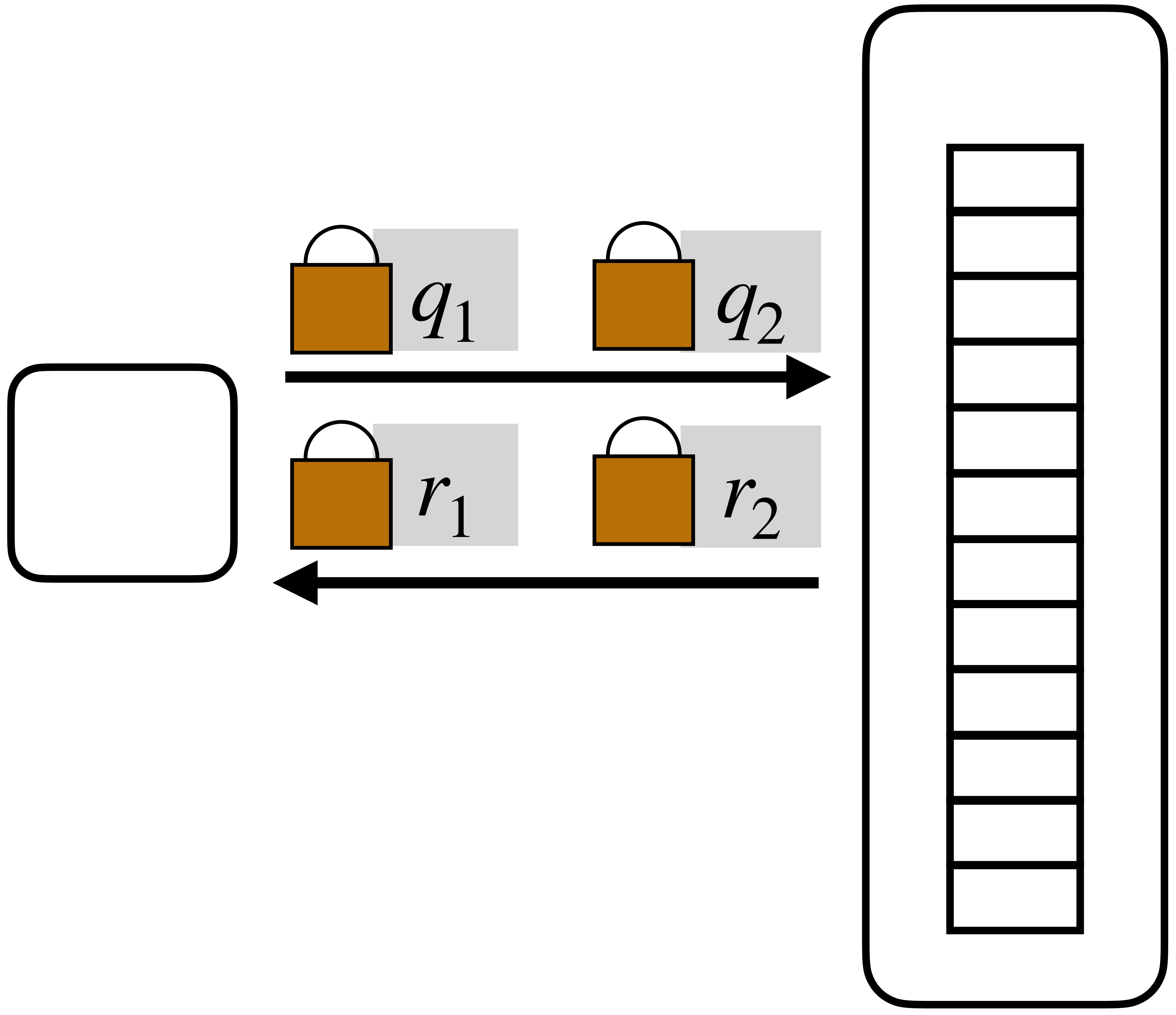


The non-signaling barrier

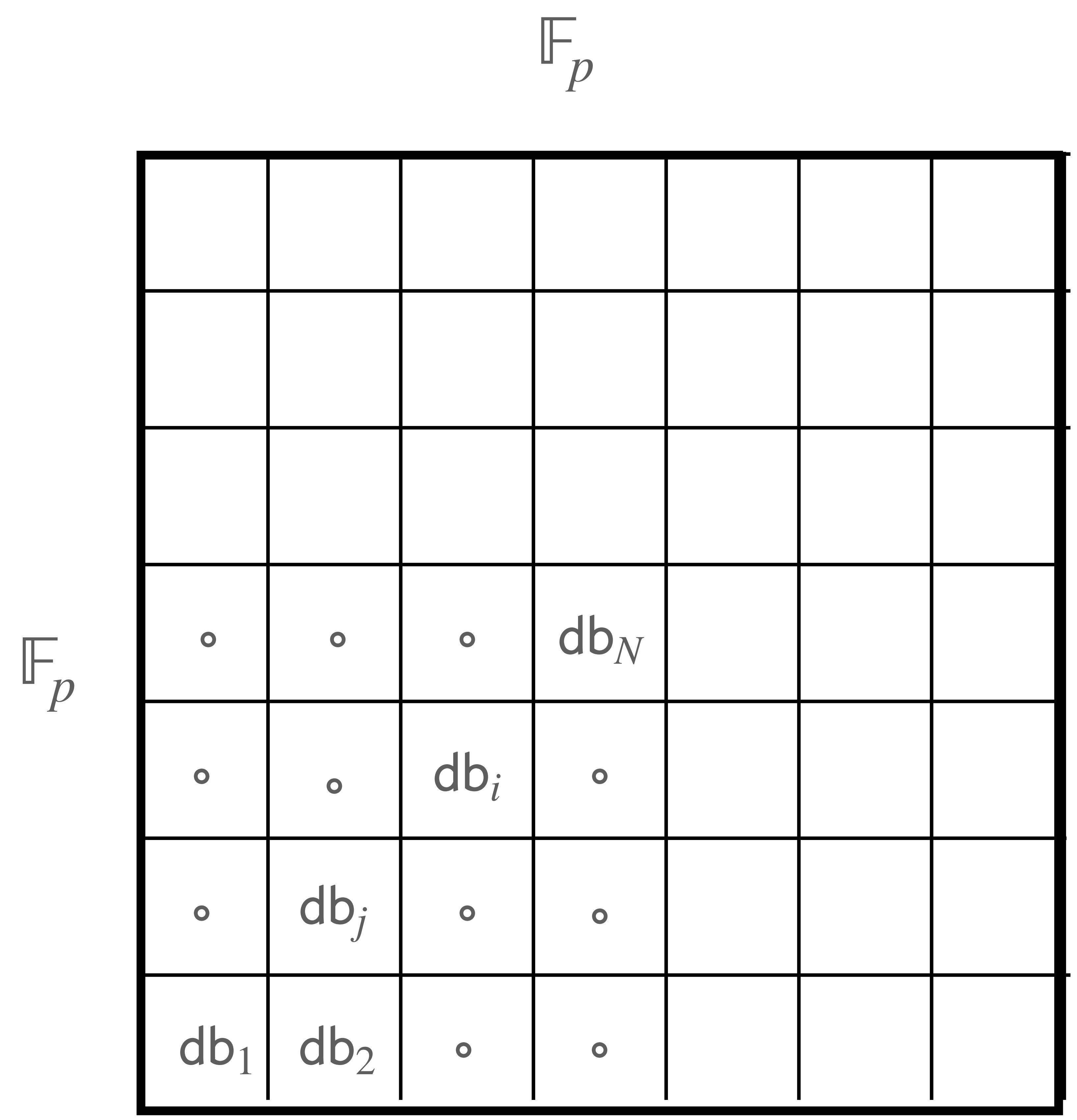
- What guarantee does PIR privacy give us on multiple queries?
  - Response i is independent of query j?
  - Don't know how to prove this strong guarantee.
- Problem: PIR guarantees that response for i does not "leak information" about query j, but may have "non-signaling" correlations with query j.
  - weaker than "independent responses!"

- Are non-signaling correlations actually a problem?
  - Can potentially allow adversary to differentiate between test and decoding queries — can't prove security.

- We show how to overcome this barrier by constructing decoder against NS adversaries with only overhead λ

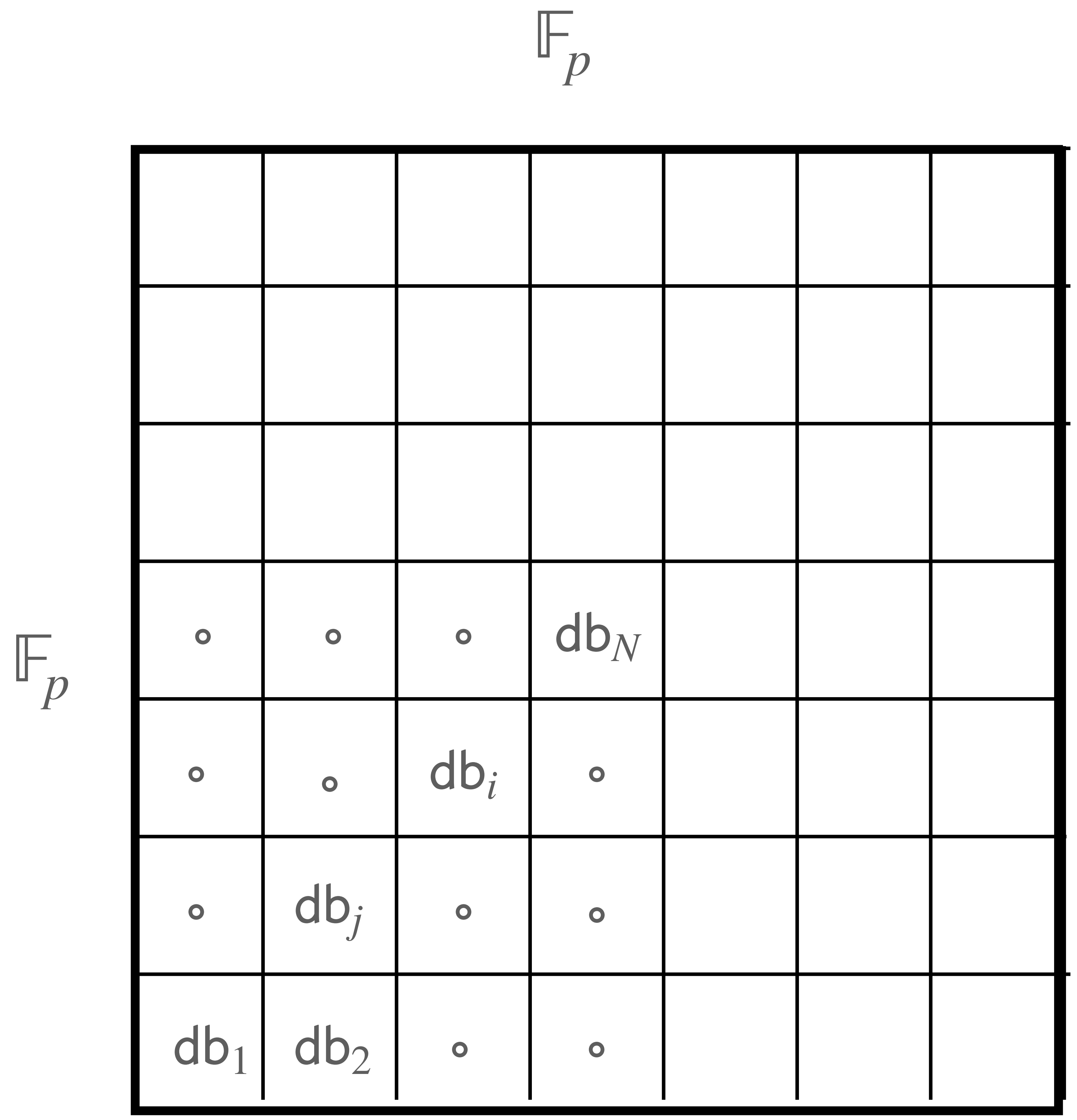


Our construction



Our construction

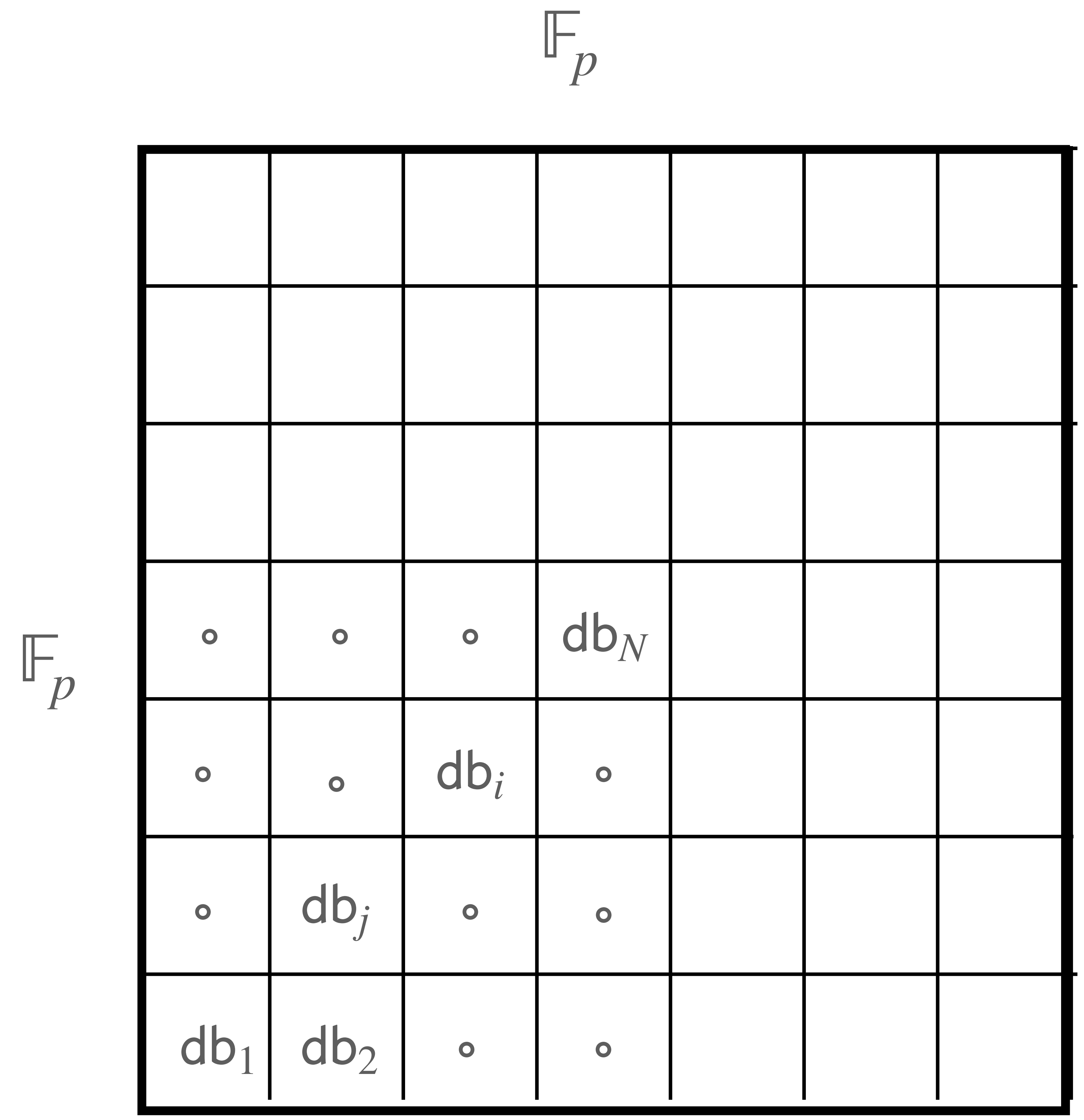
Non-signaling local decoding



Our construction

Non-signaling local decoding

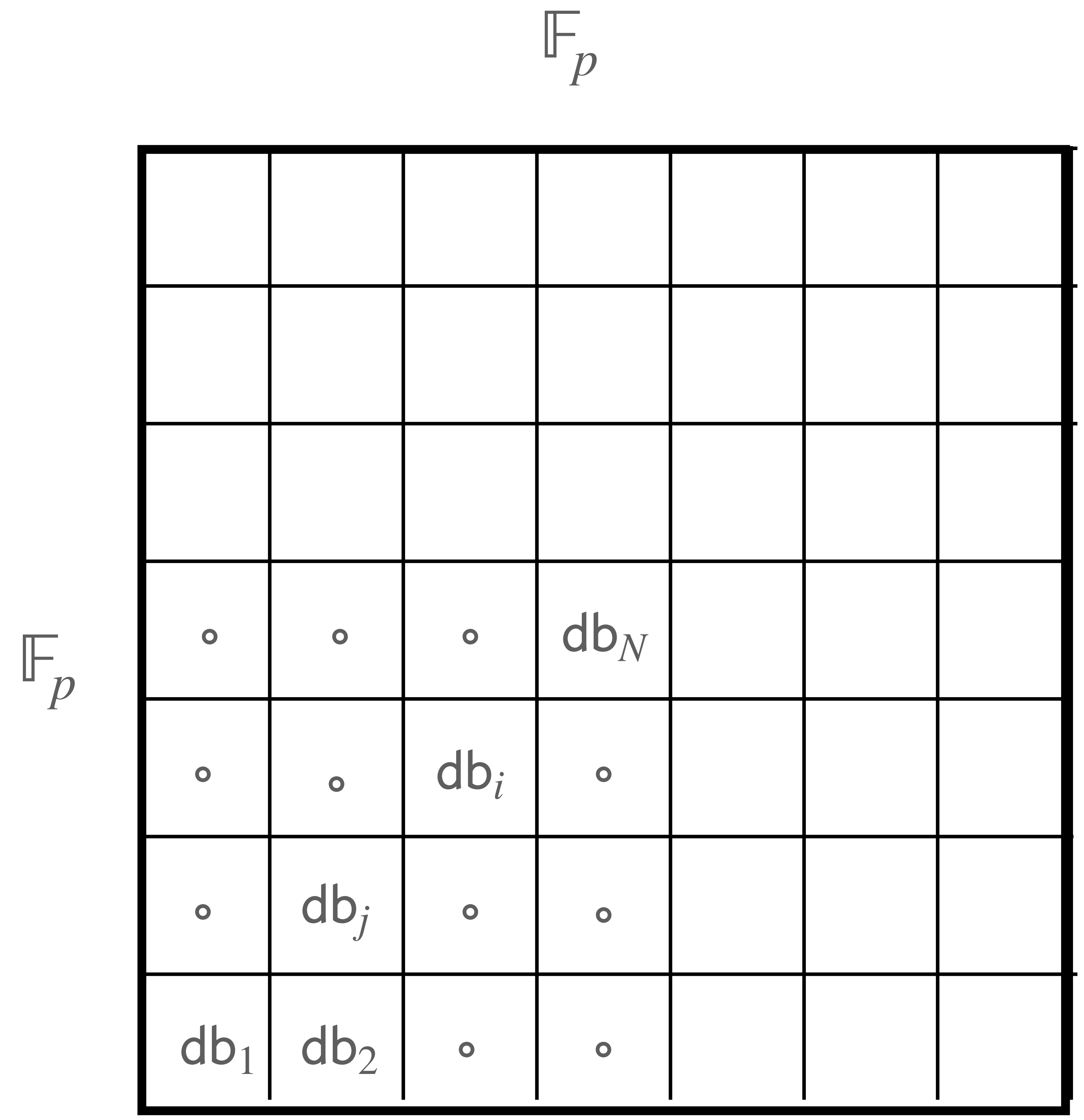
1. Want: db_j .



Our construction

Non-signaling local decoding

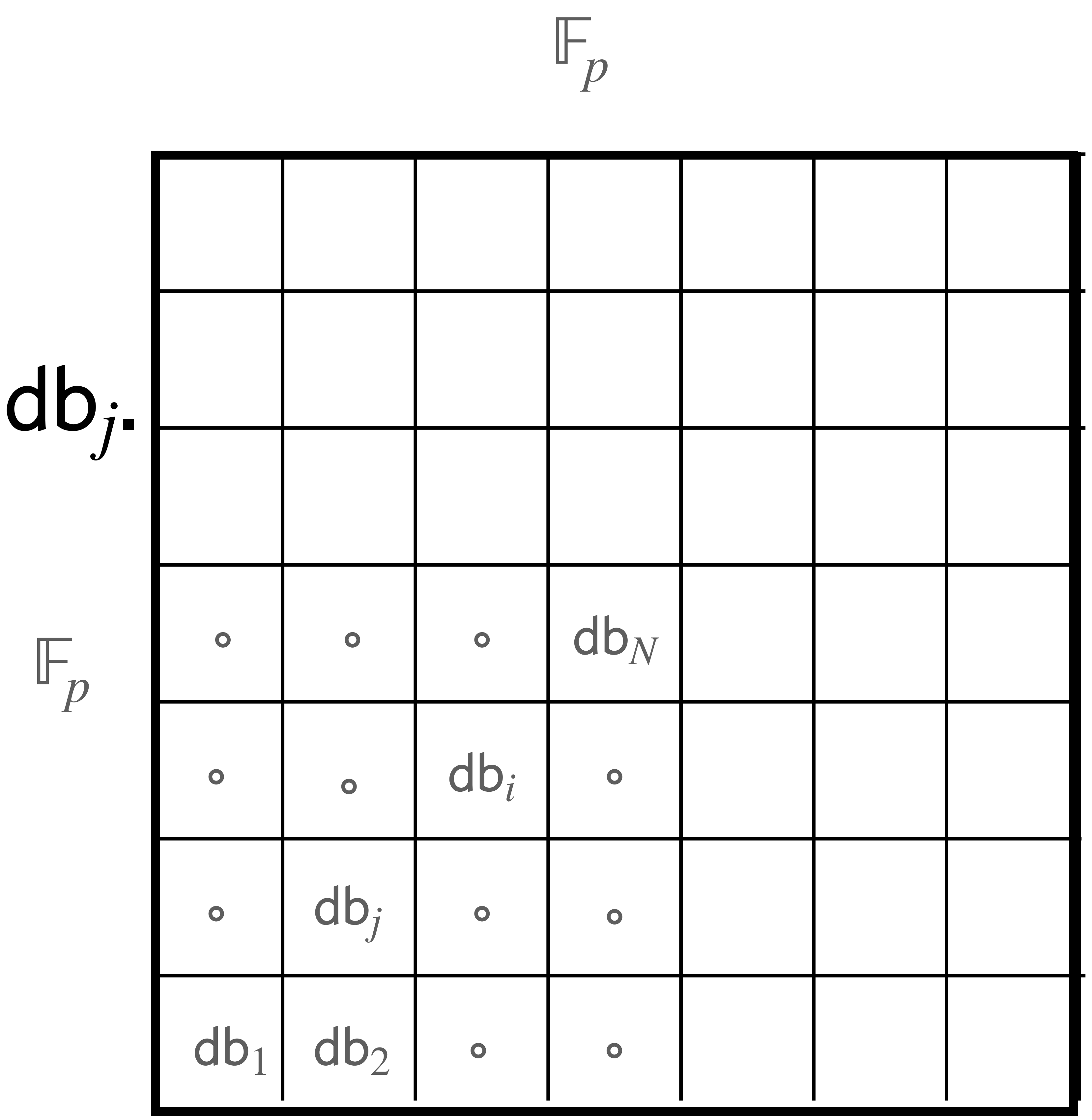
1. Want: db_j .
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$:



Our construction

Non-signaling local decoding

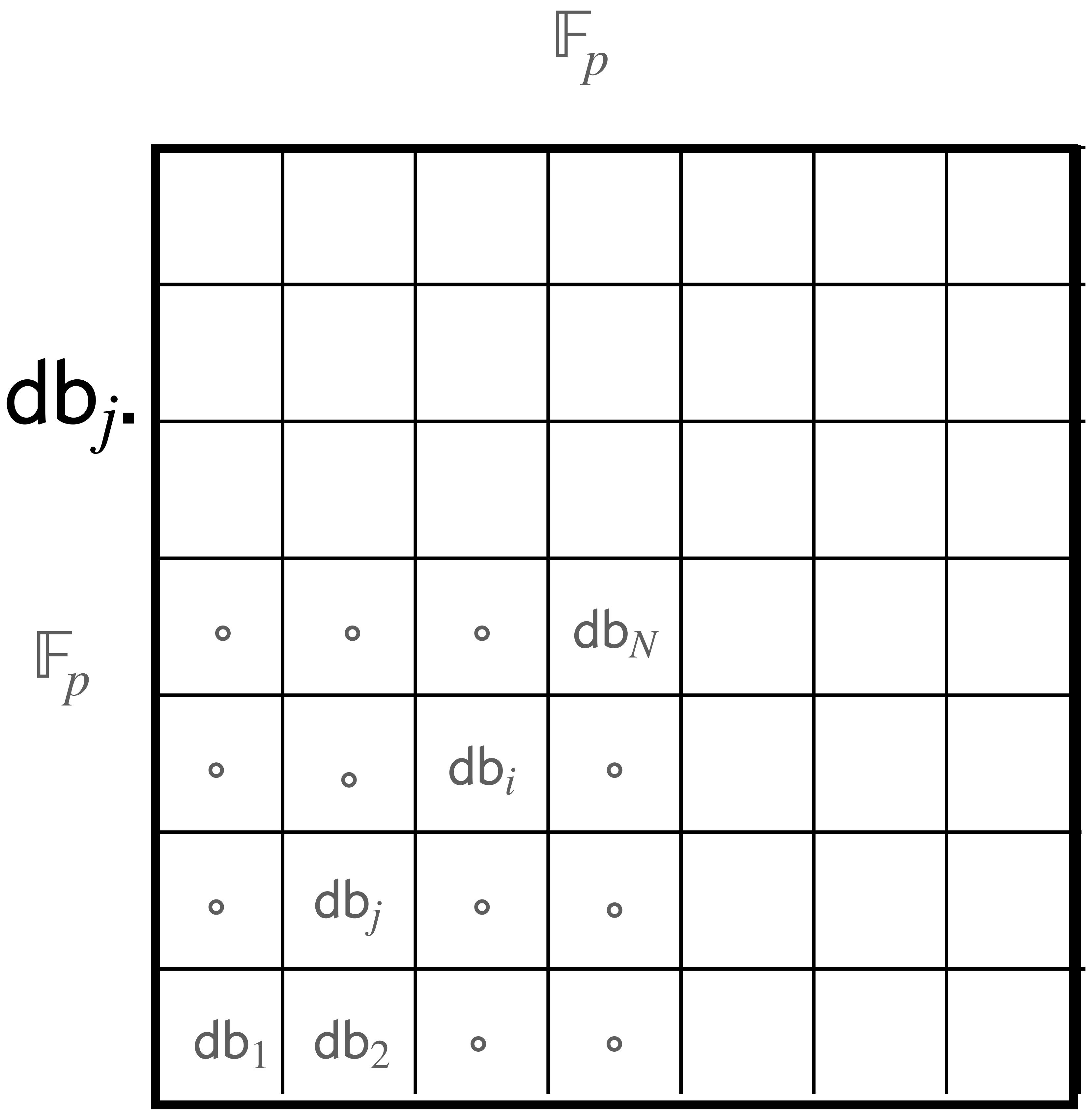
1. Want: db_j .
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$:
    1. let $L_1, \ldots, L_t$ be random lines through db_j.



Our construction

Non-signaling local decoding

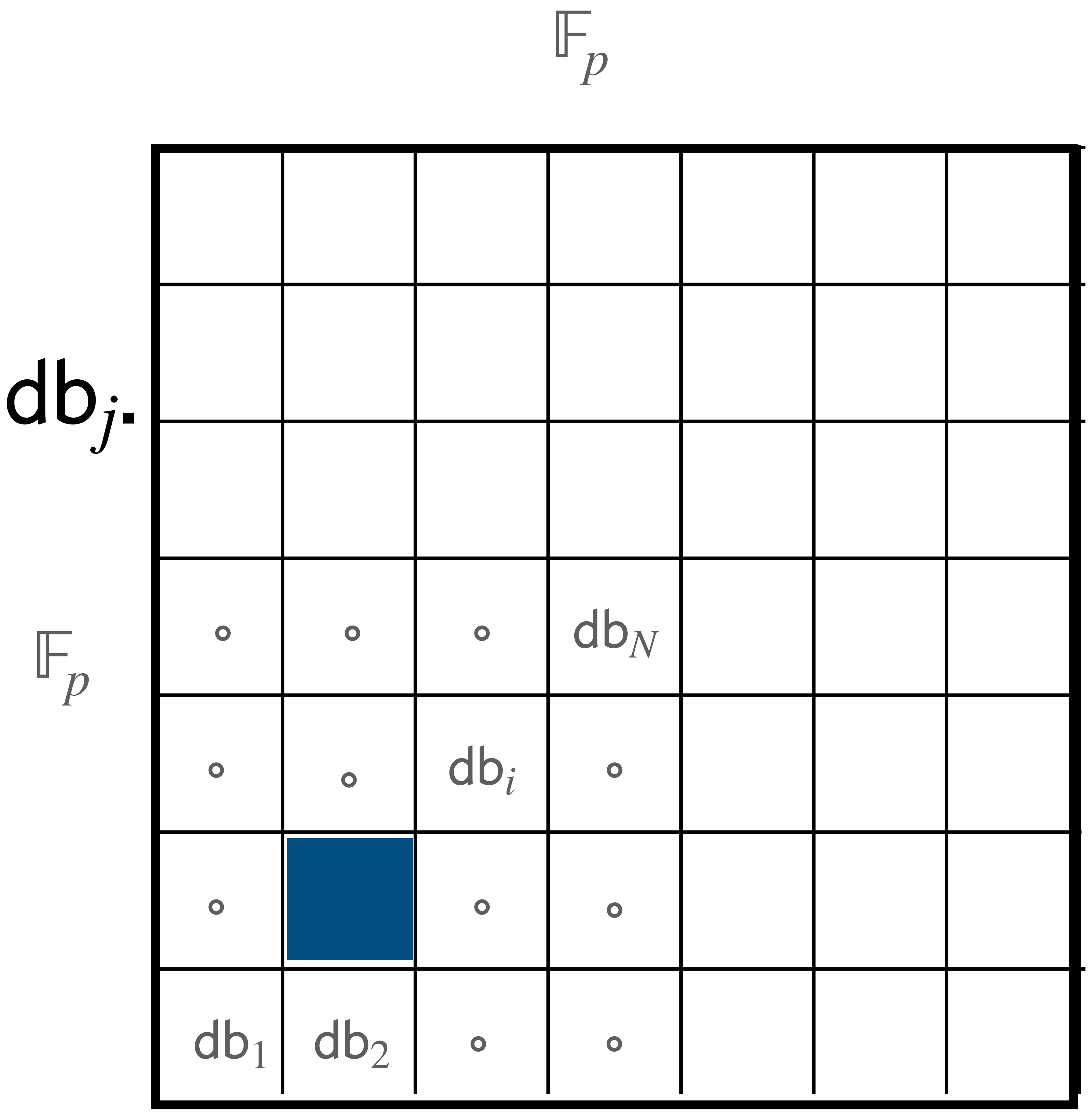
1. Want: db_j .
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$:
   1. let $L_1, \ldots, L_t$ be random lines through db_j.
   2. Pick a random point on each line; call this the test set T.



Our construction

Non-signaling local decoding

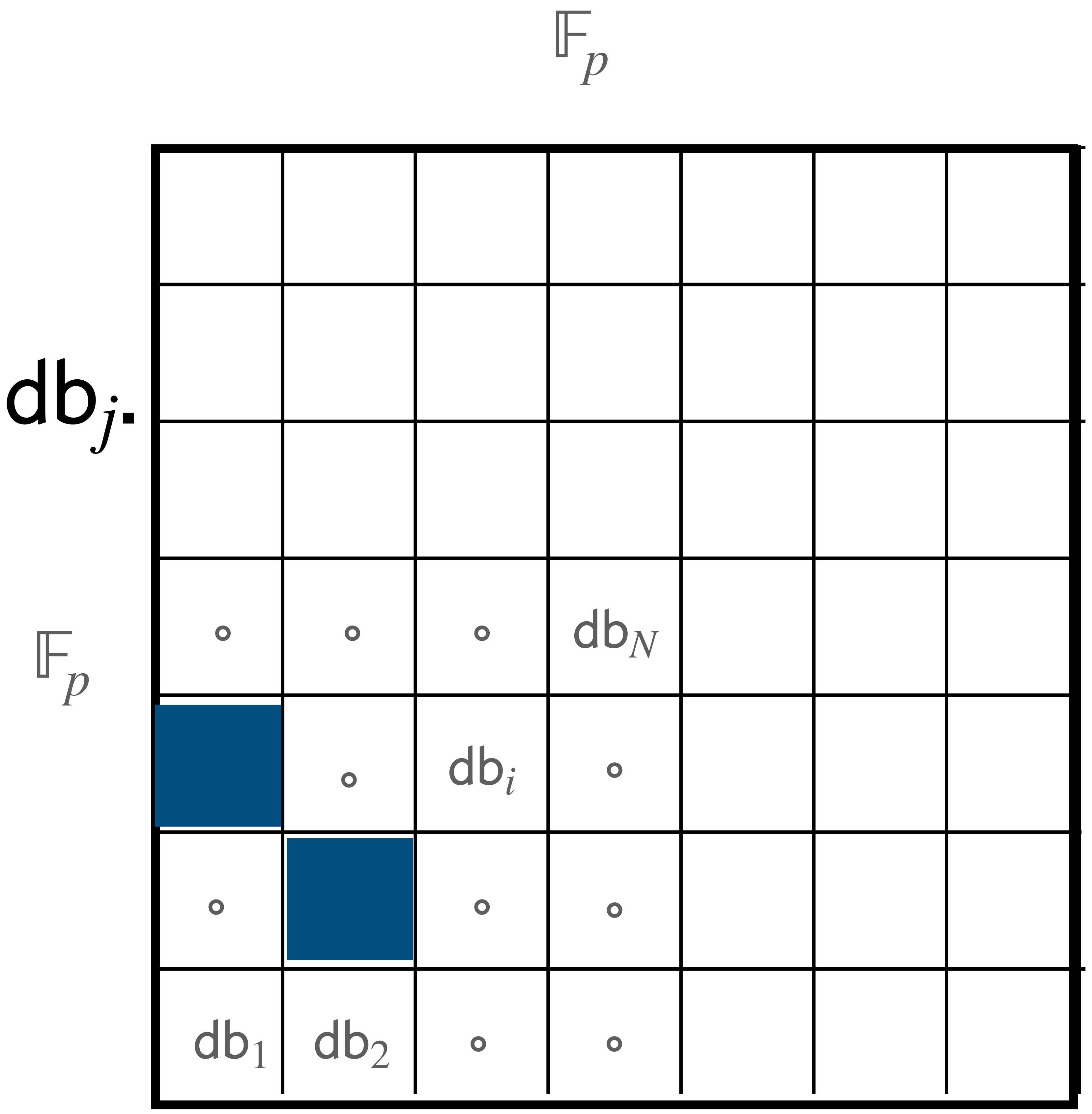
1. Want: db_j .
2. $\mathsf{RM.Que}(j) \to Q$:
   1. let $L_1, \ldots, L_t$ be random lines through db_j.
   2. Pick a random point on each line; call this the test set T.



Our construction

Non-signaling local decoding

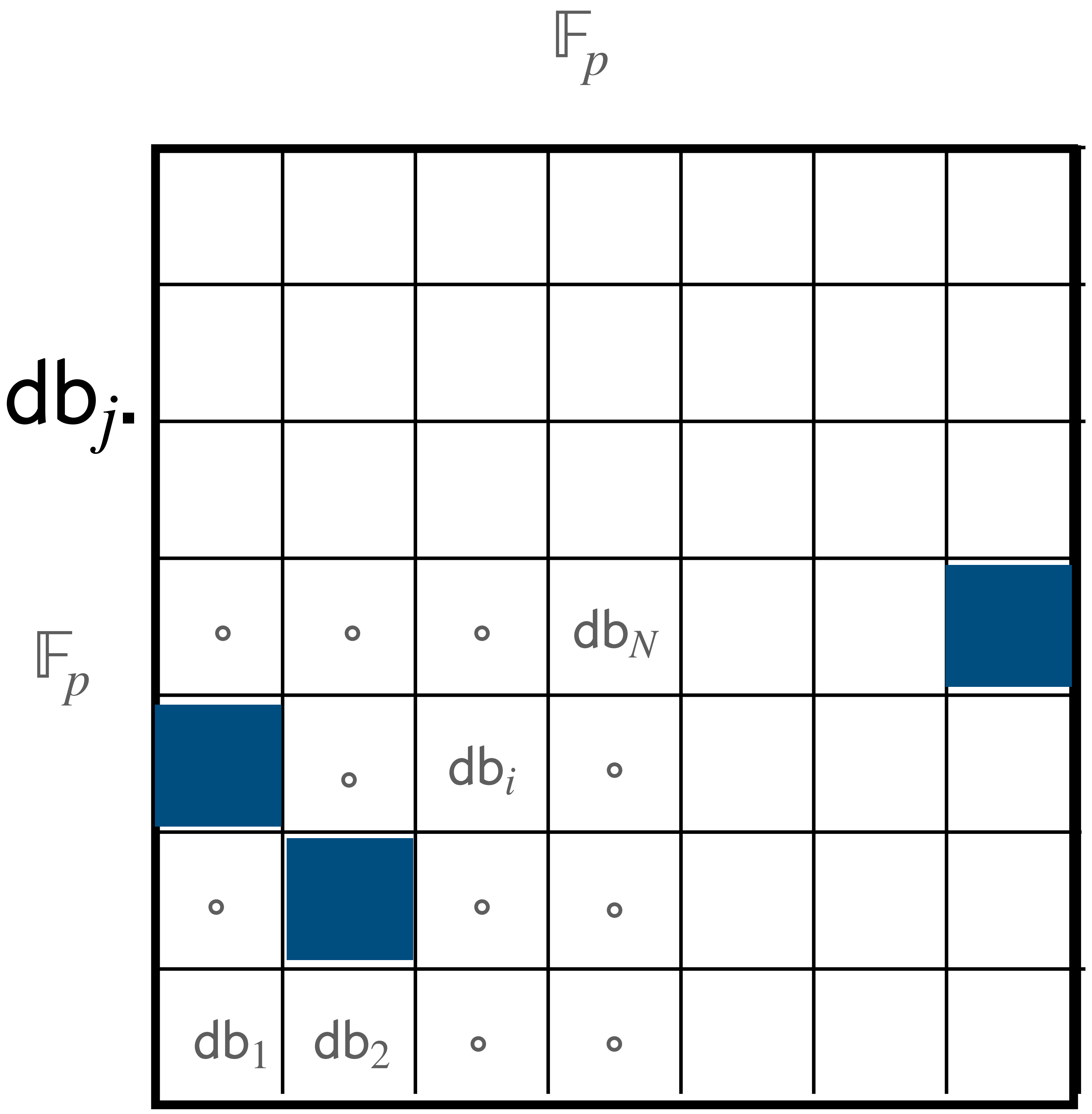
1. Want: db_j .
2. $\mathsf{RM.Que}(j) \to Q$:
   1. let $L_1, \ldots, L_t$ be random lines through db_j.
   2. Pick a random point on each line; call this the test set T.



Our construction

Non-signaling local decoding

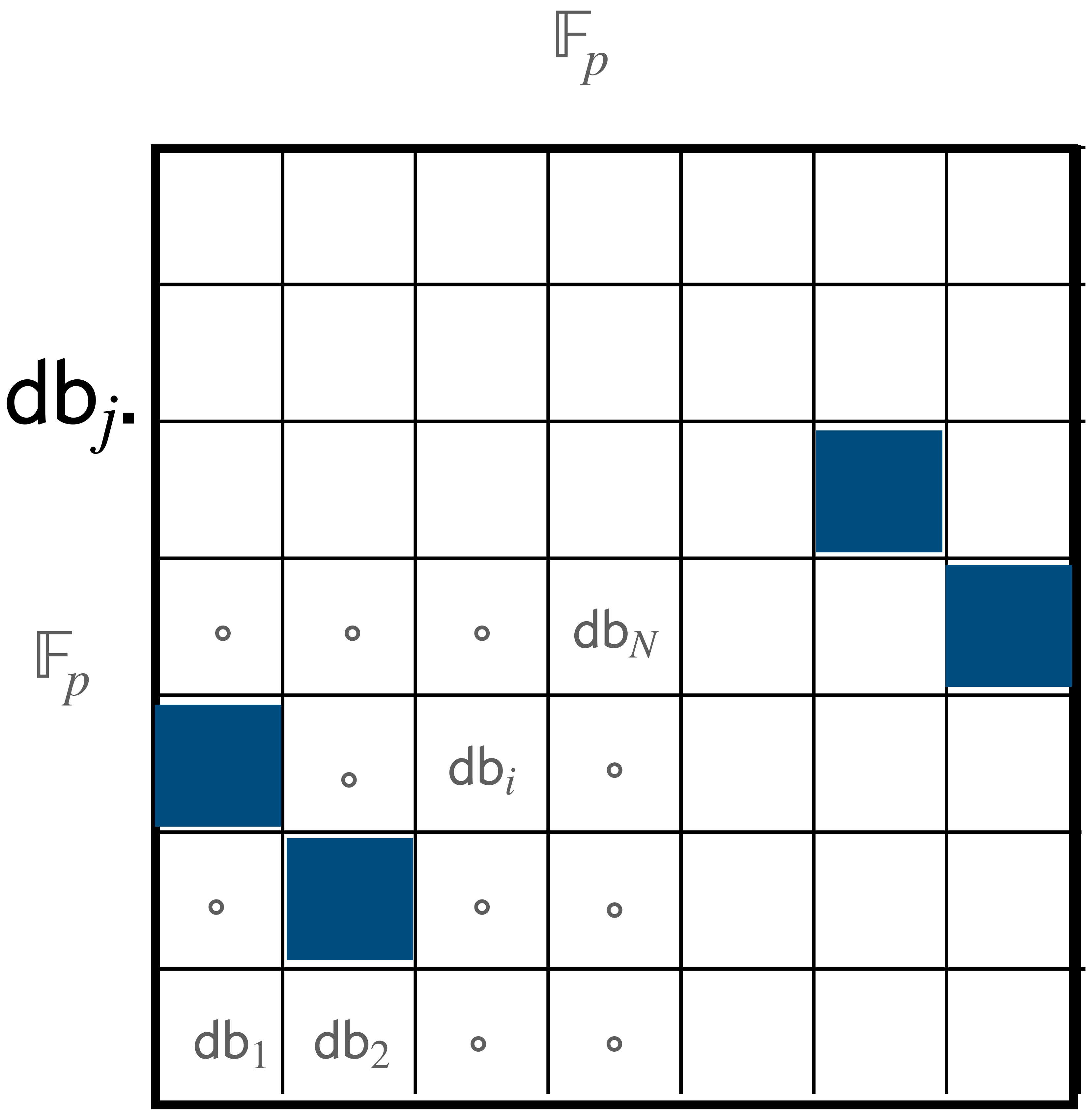
1. Want: db_j .
2. $\mathsf{RM.Que}(j) \to Q$:
   1. let $L_1, \ldots, L_t$ be random lines through db_j.
   2. Pick a random point on each line; call this the test set T.



Our construction

Non-signaling local decoding

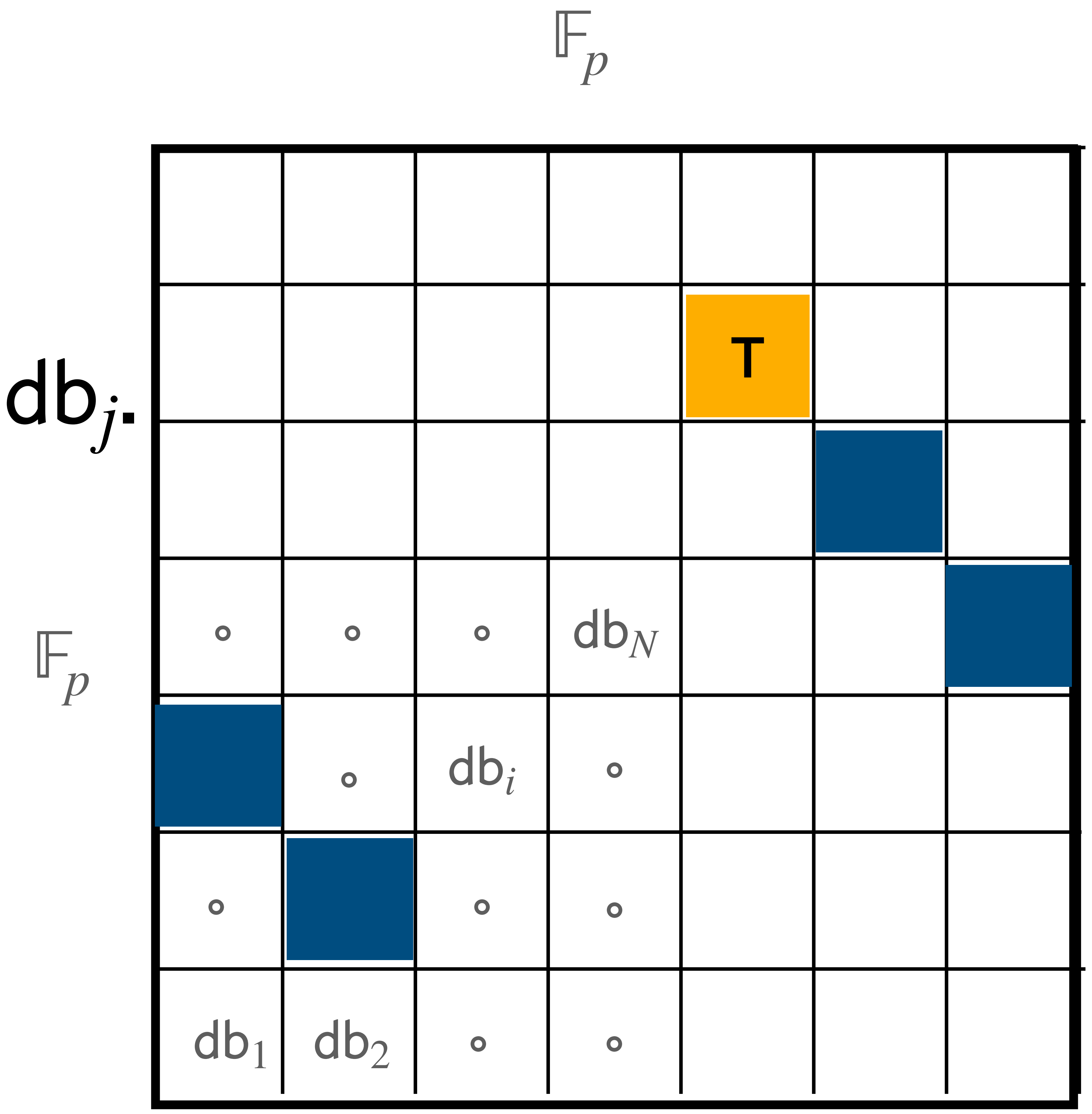
1. Want: db_j .
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$:
   1. let $L_1, \ldots, L_t$ be random lines through db_j.
   2. Pick a random point on each line; call this the test set T.



Our construction

Non-signaling local decoding

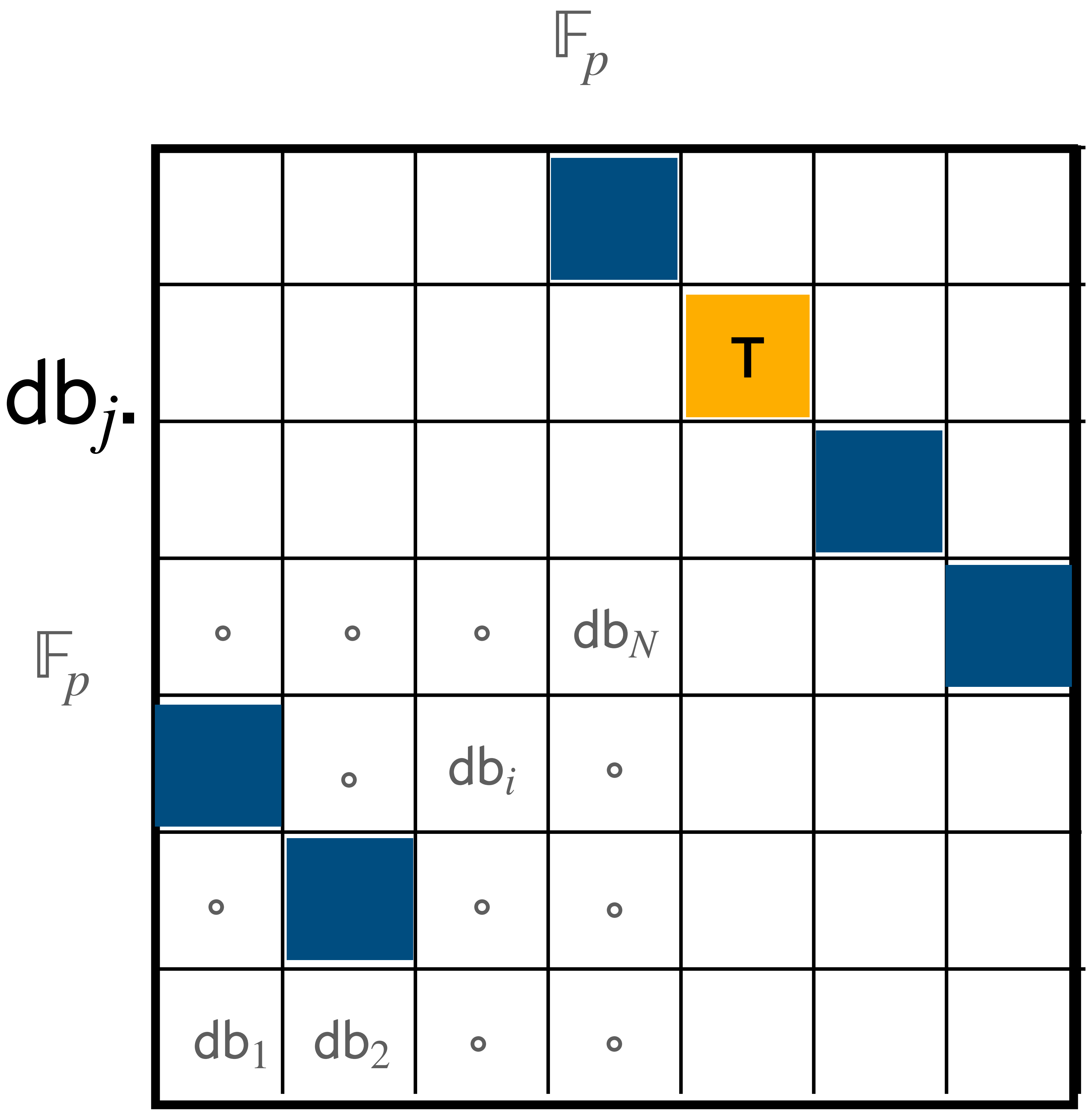
1. Want: db_j .
2. $\mathrm{RM.Que}(j) \to Q$:
   1. let $L_1, \ldots, L_t$ be random lines through db_j.
   2. Pick a random point on each line; call this the test set T.



Our construction

Non-signaling local decoding

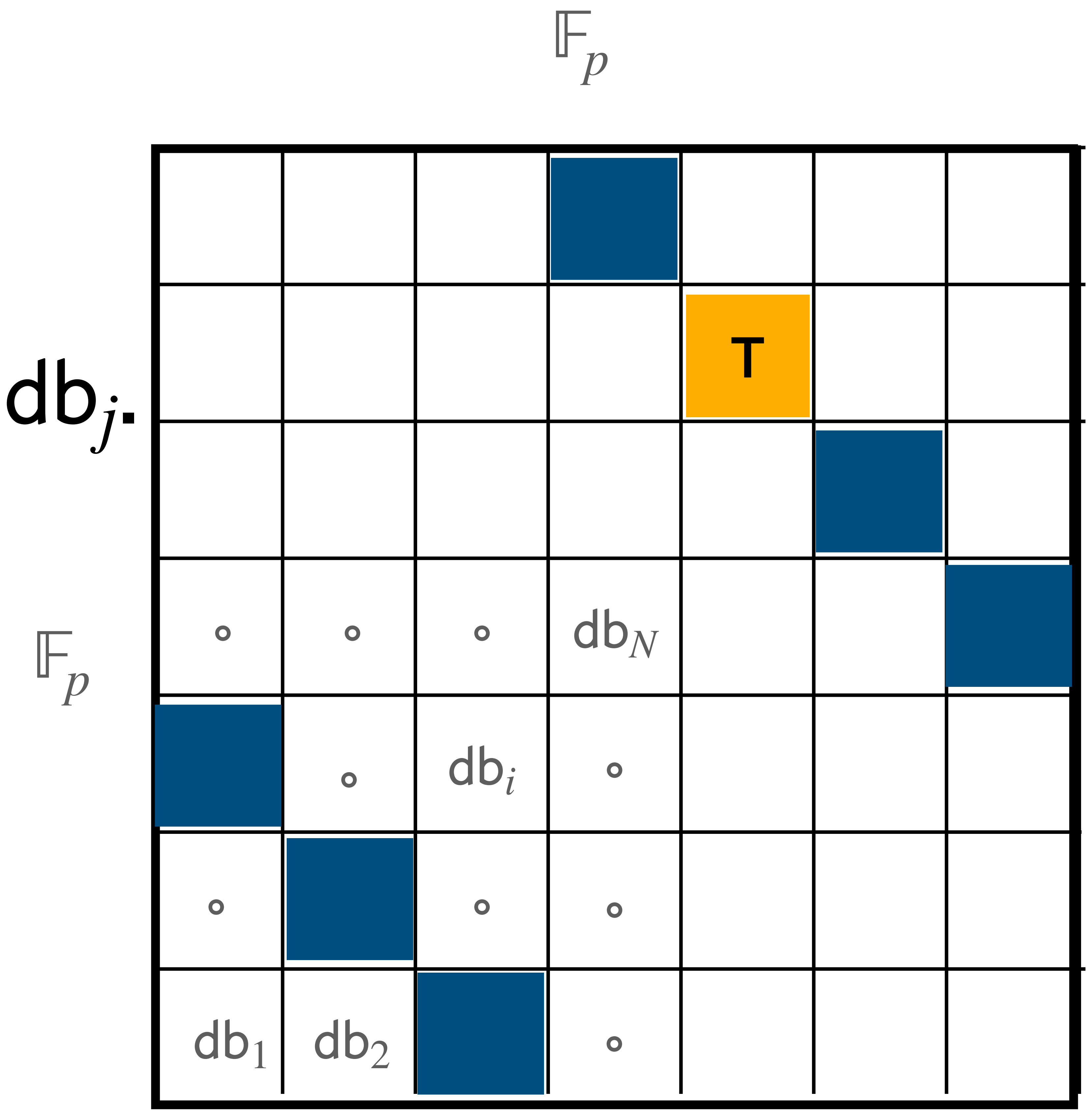
1. Want: db_j .
2. $\mathsf{RM}.\mathsf{Que}(j) \to Q$:
   1. let $L_1, \ldots, L_t$ be random lines through db_j.
   2. Pick a random point on each line; call this the test set T.



Our construction

Non-signaling local decoding

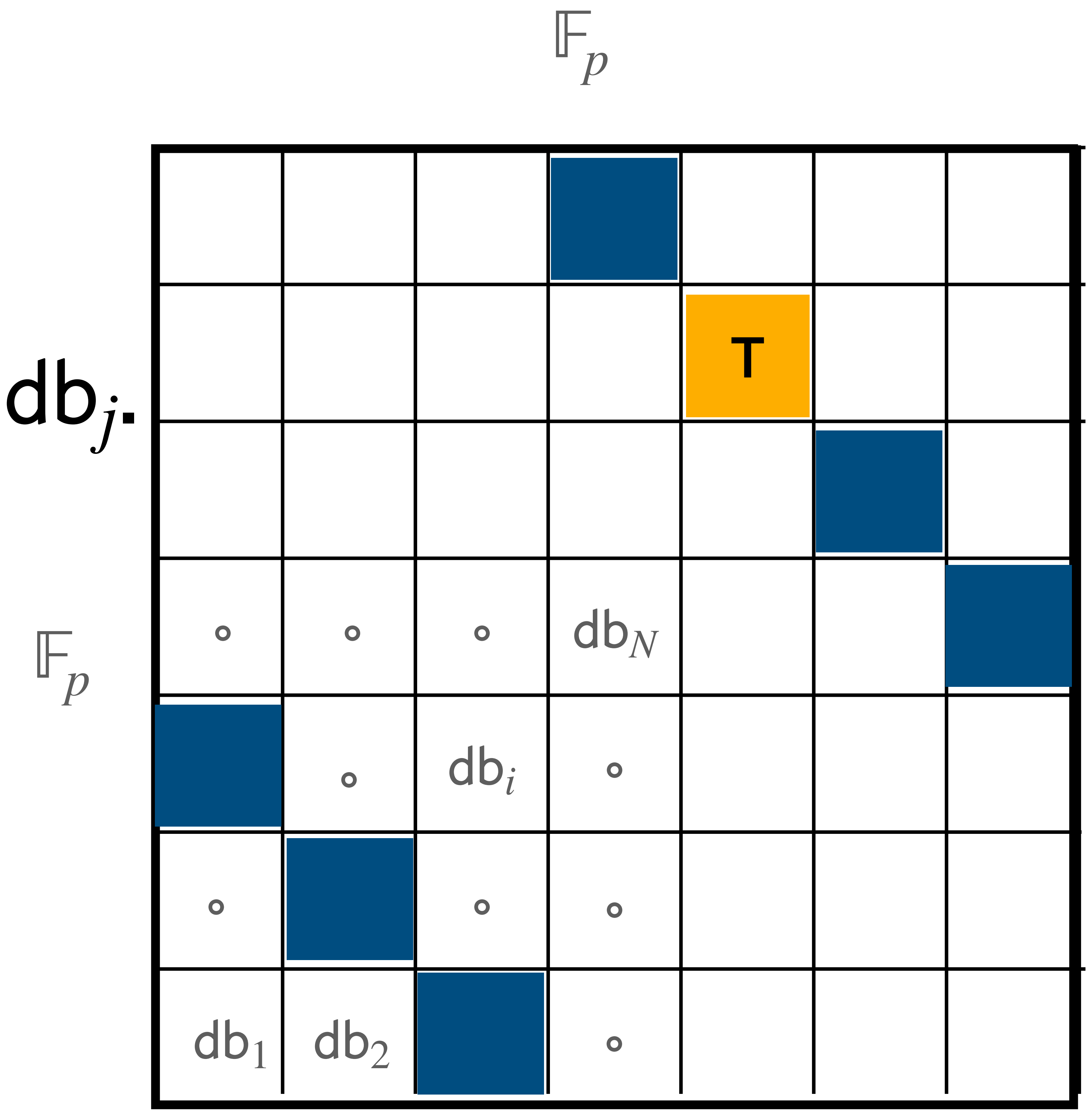
1. Want: db_j .
2. $\mathsf{RM.Que}(j) \rightarrow Q$:
   1. let $L_1, \ldots, L_t$ be random lines through db_j.
   2. Pick a random point on each line; call this the test set T.



Our construction

Non-signaling local decoding

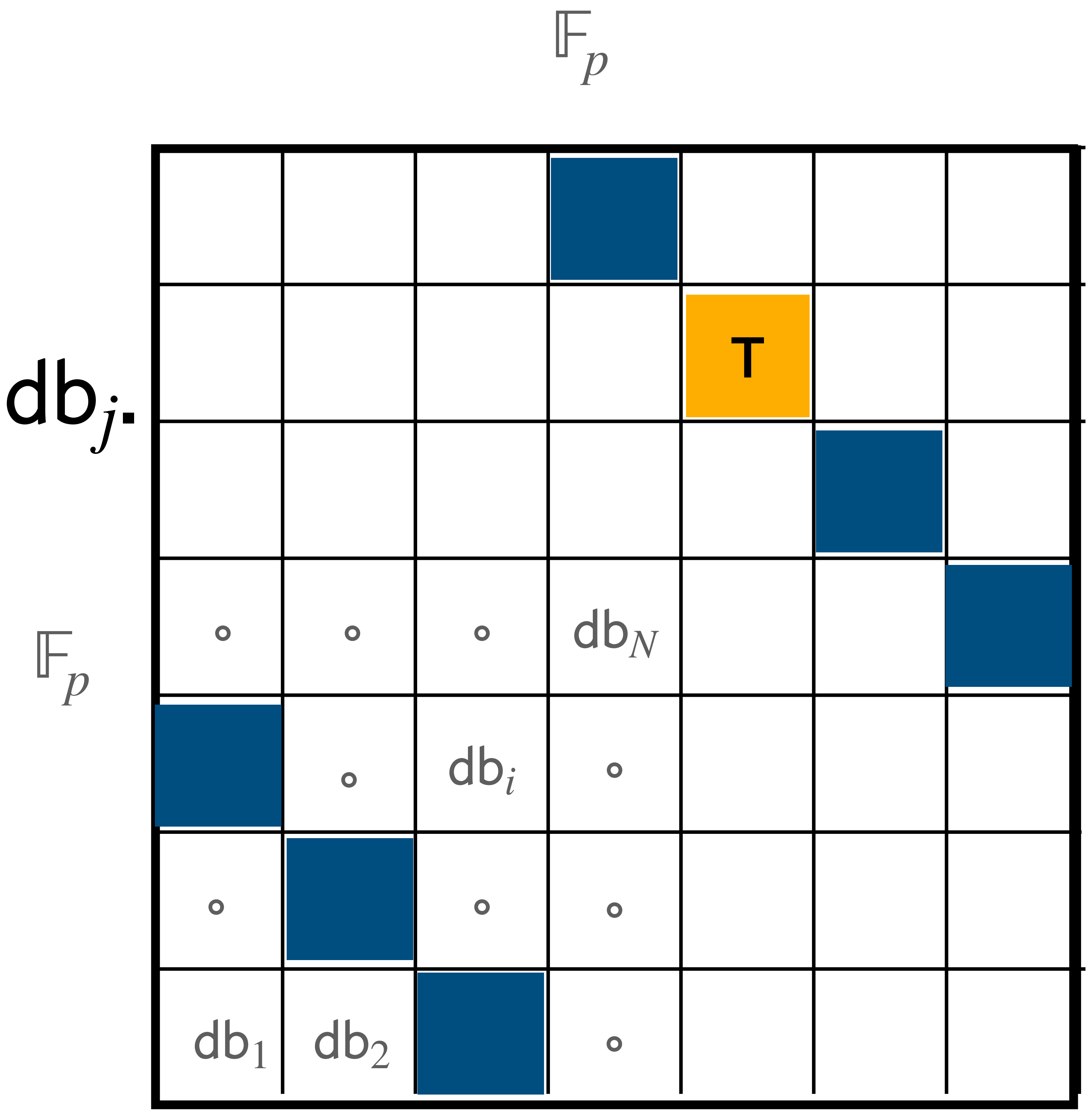
1. Want: db_j .
2. $\mathsf{RM.Que}(j) \to Q$:
    1. let $L_1, \ldots, L_t$ be random lines through db_j.
    2. Pick a random point on each line; call this the test set T.
    3. let $Q = L \cup T$.



Our construction

Non-signaling local decoding

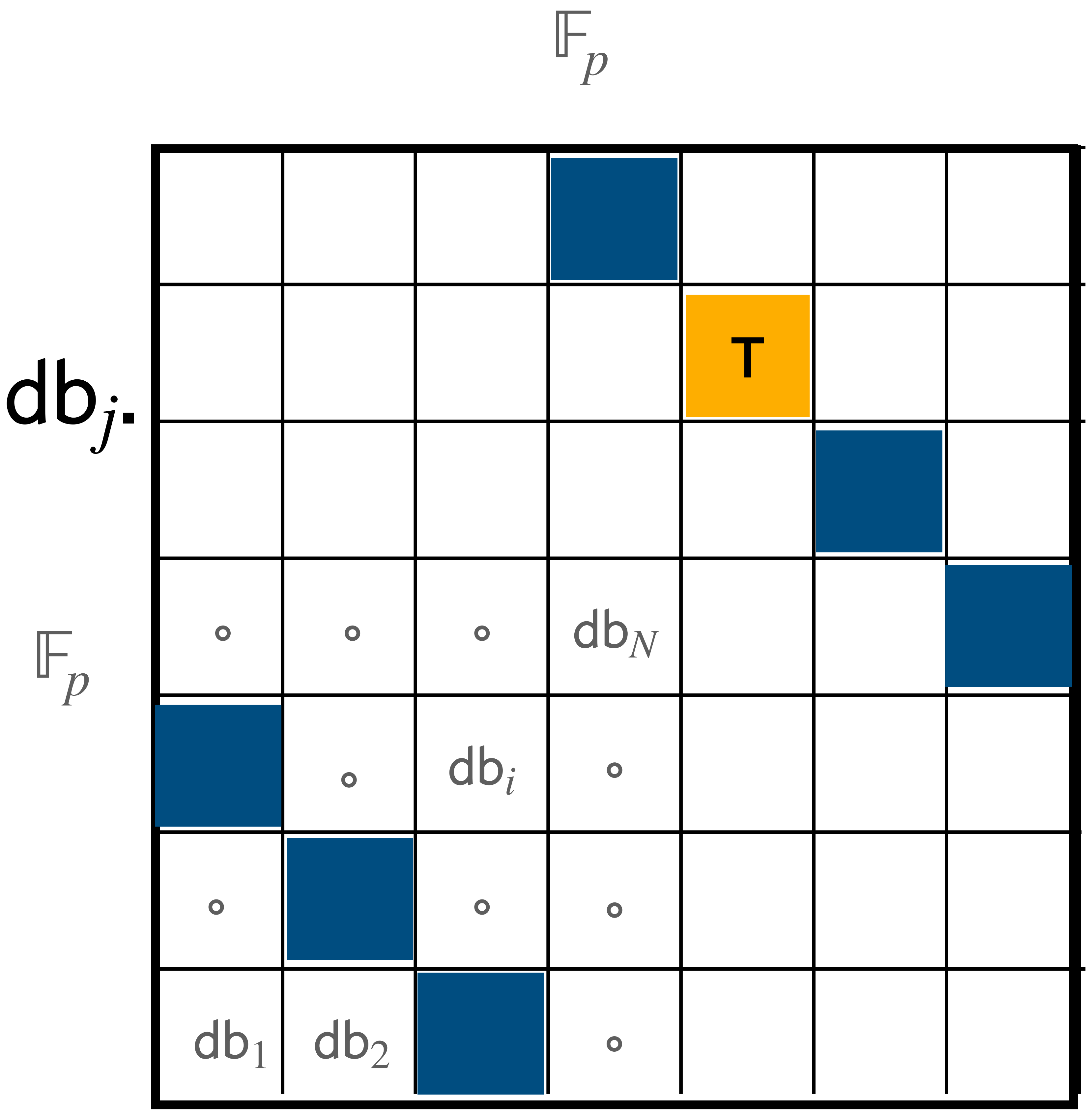
1. Want: db_j .
2. RM . Que$(j) \to Q$:
   1. let $L_1, \ldots, L_t$ be random lines through db_j.
   2. Pick a random point on each line; call this the test set T.
   3. let $Q = L \cup T$.
3. RM . Dec$(E_Q) \to \mathsf{db}_j$:



Our construction

Non-signaling local decoding

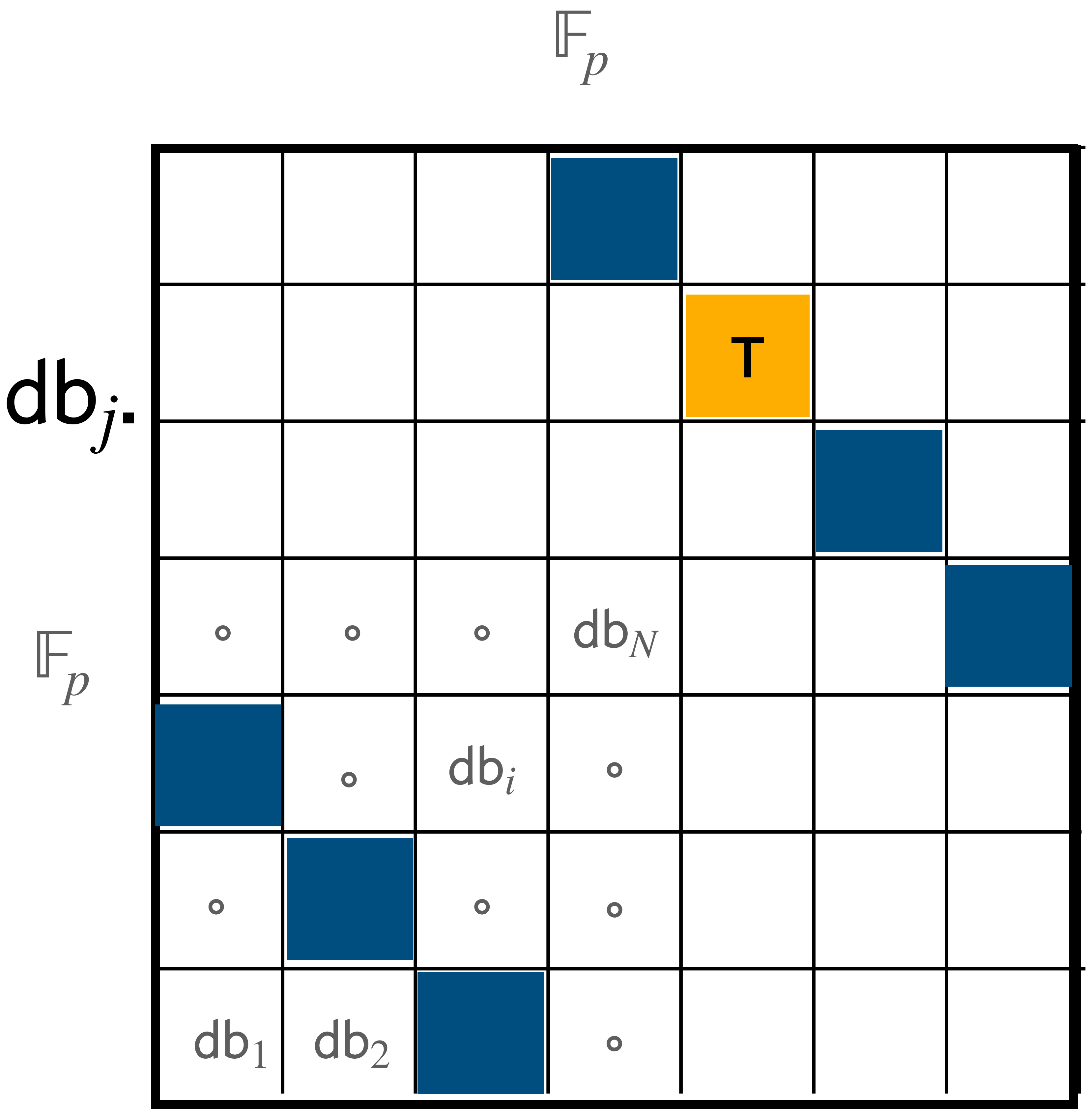
1. Want: db_j .
2. $\mathrm{RM.Que}(j) \rightarrow Q$:
   1. let $L_1, \ldots, L_t$ be random lines through db_j.
   2. Pick a random point on each line; call this the test set T.
   3. let $Q = L \cup T$.
3. $\mathrm{RM.Dec}(E_Q) \rightarrow \mathrm{db}_j$:
   1. If E_T has corruptions, output $\perp$



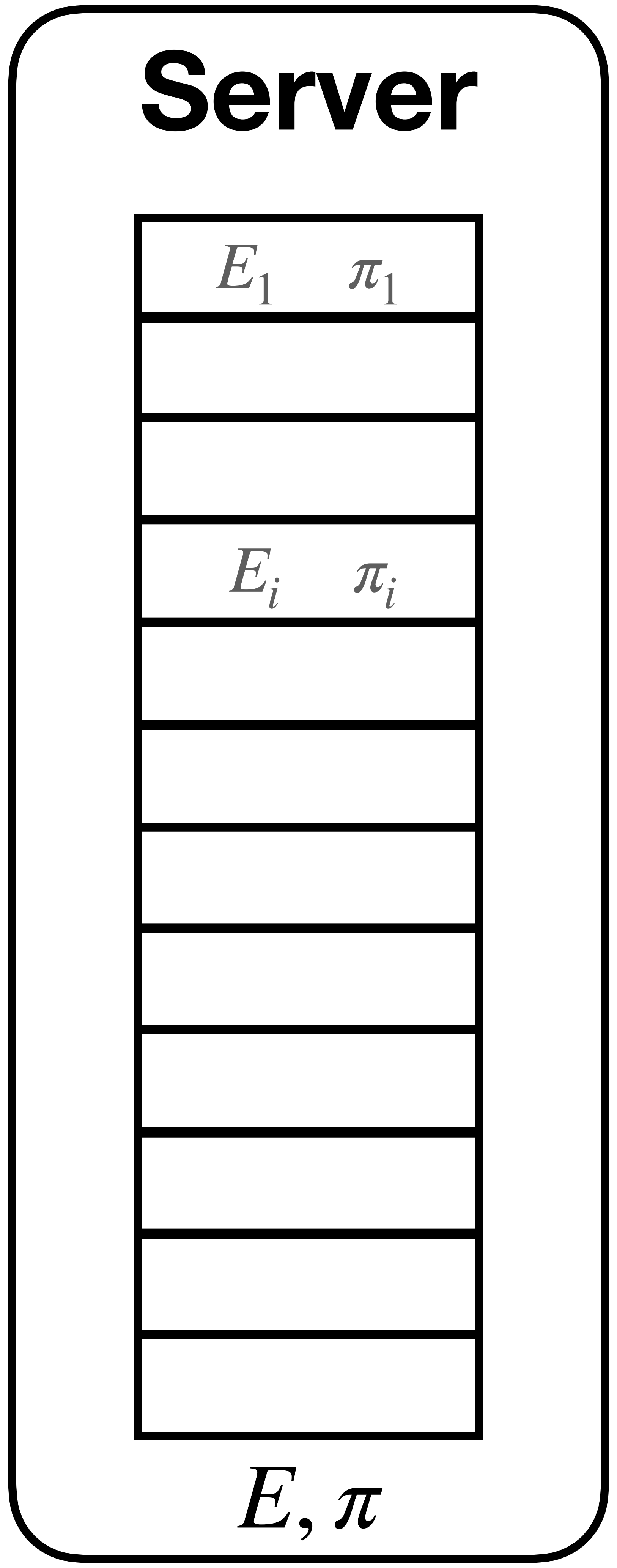
Our construction

Non-signaling local decoding

1. Want: db_j .
2. $\mathsf{RM.Que}(j) \to Q$:
   1. let $L_1, \ldots, L_t$ be random lines through db_j.
   2. Pick a random point on each line; call this the test set T.
   3. let $Q = L \cup T$.
3. $\mathsf{RM.Dec}(E_Q) \to \mathsf{db}_j$:
   1. If E_T has corruptions, output $\perp$
   2. Else, decode $E_{L_1}, \ldots, E_{L_t}$ as before.

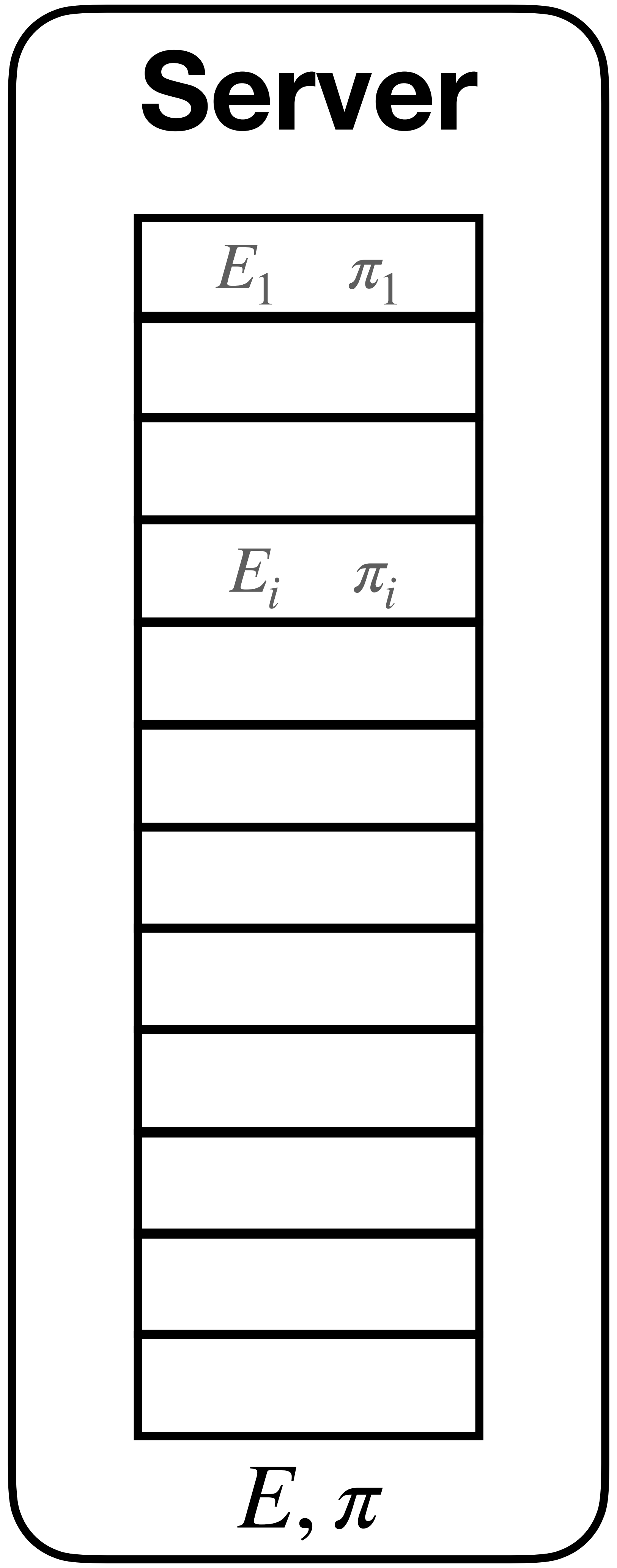


Final Construction



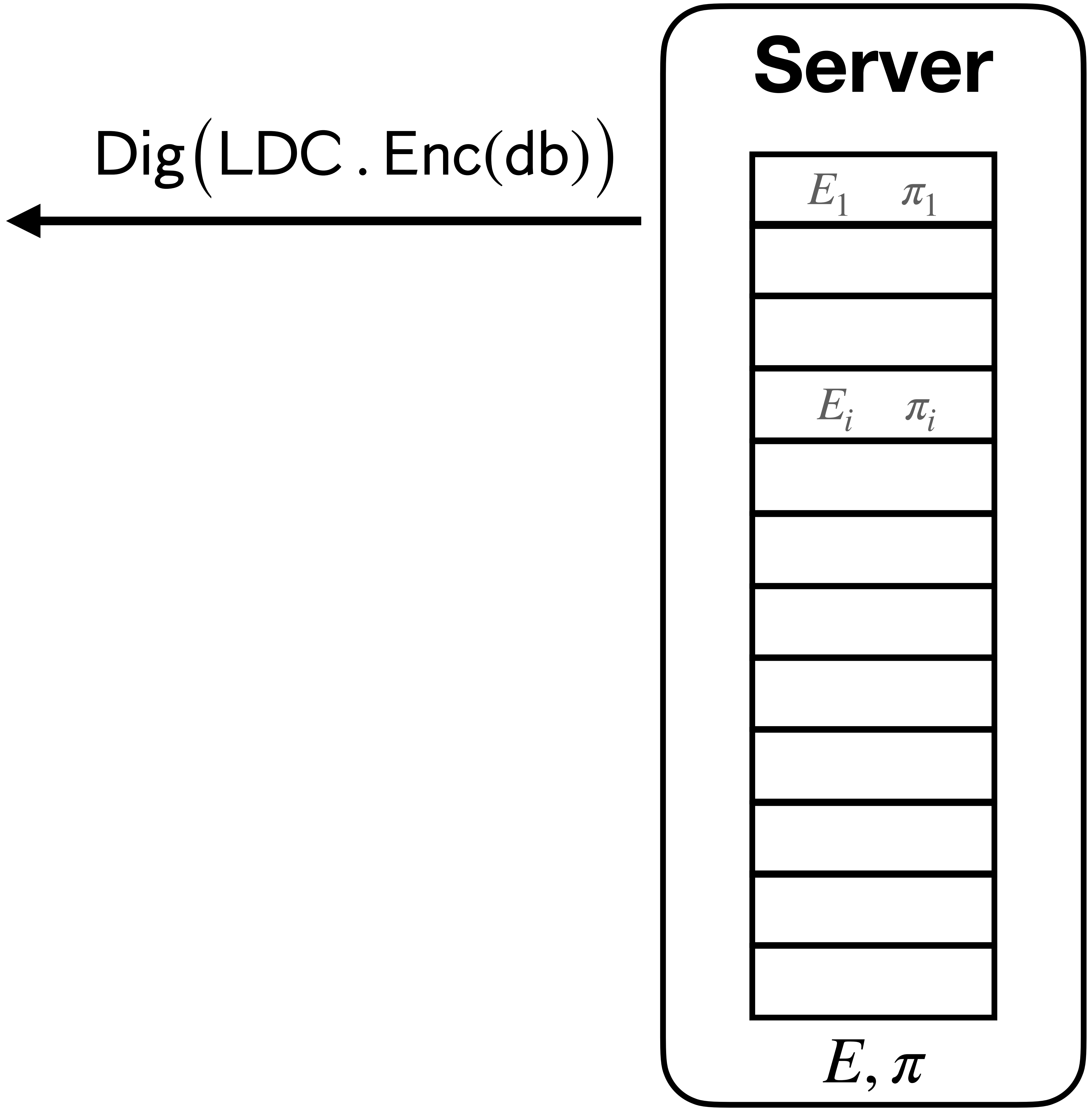
Final Construction

Offline

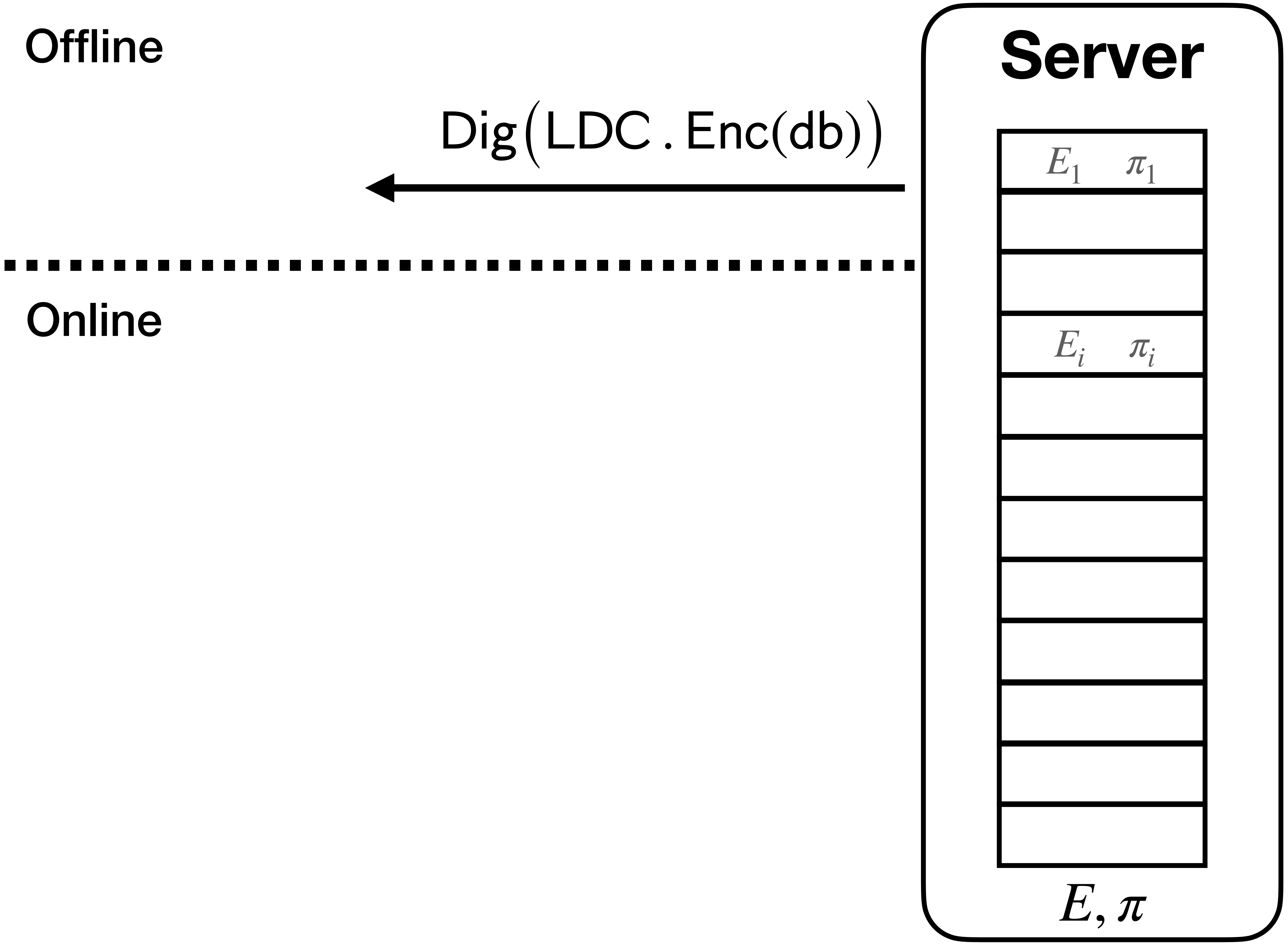


Final Construction

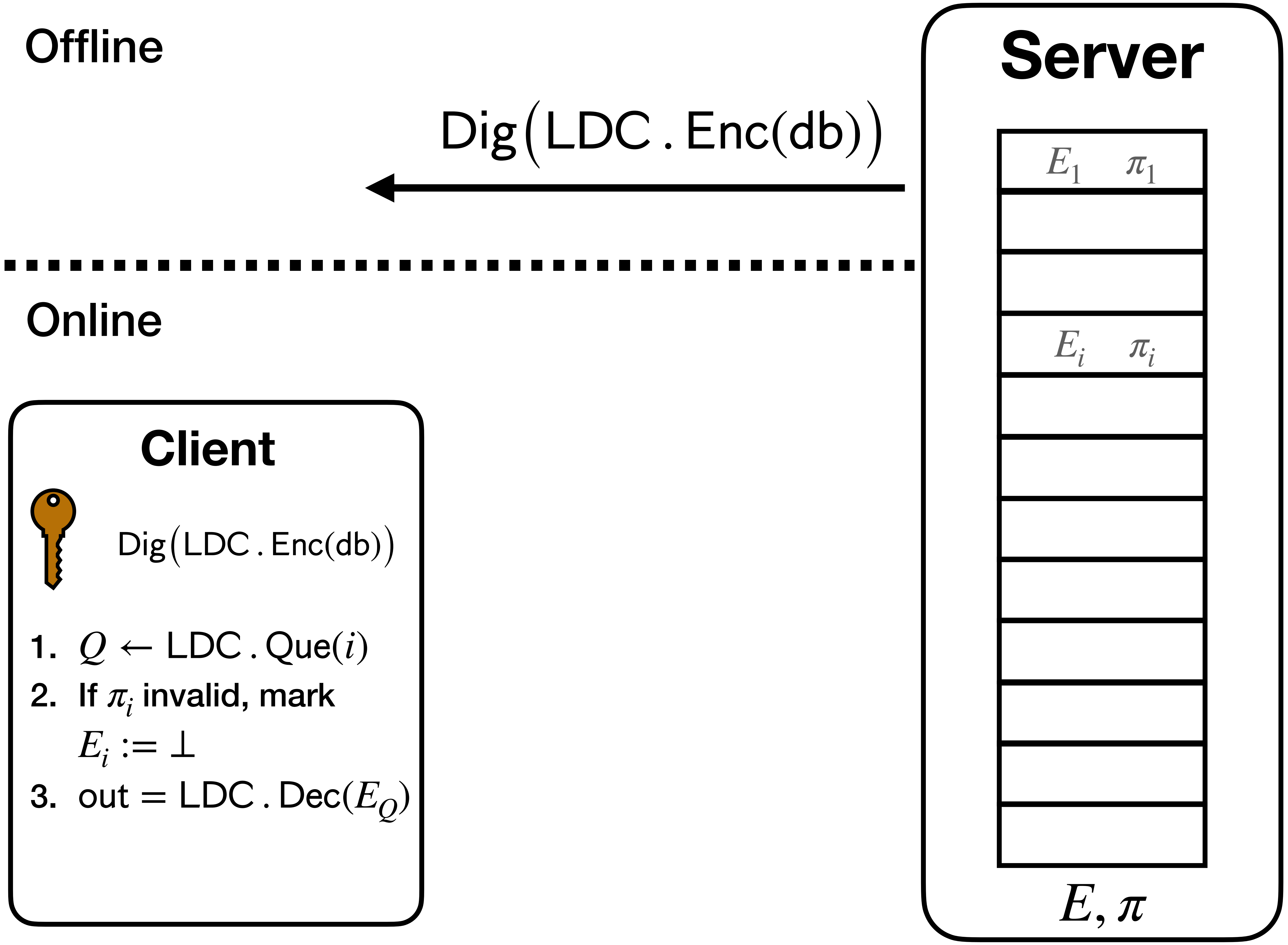
Offline



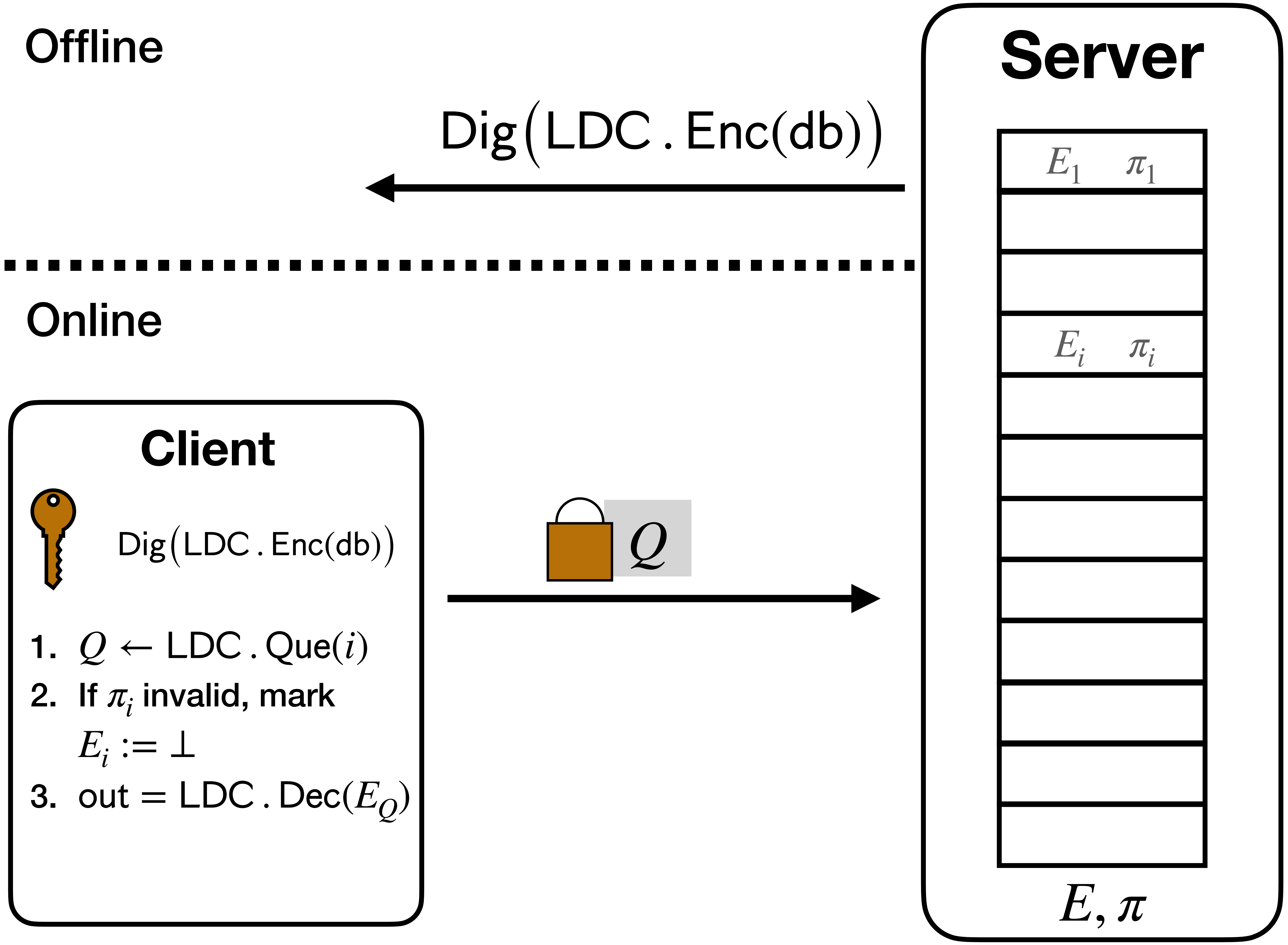
Final Construction



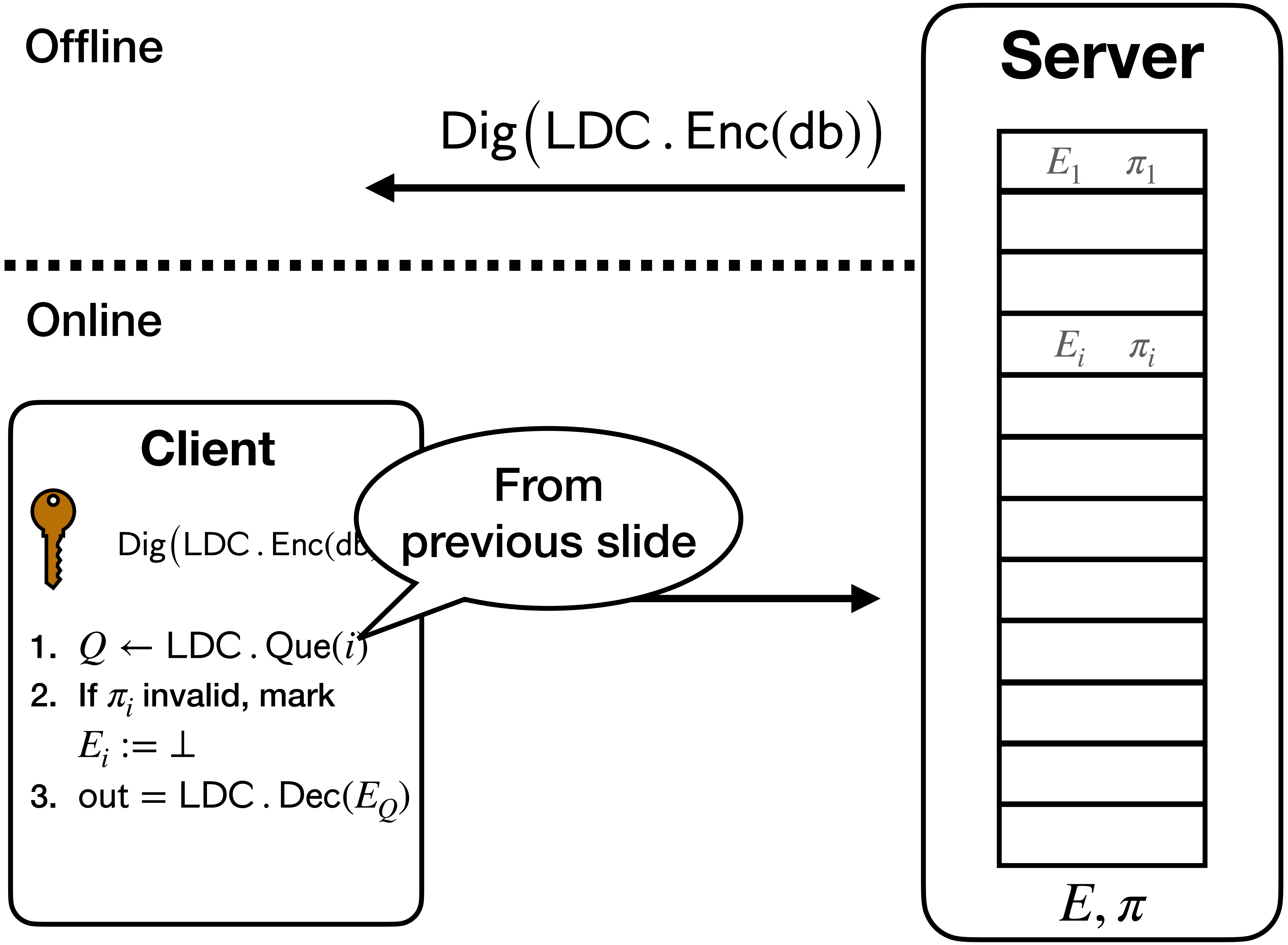
Final Construction



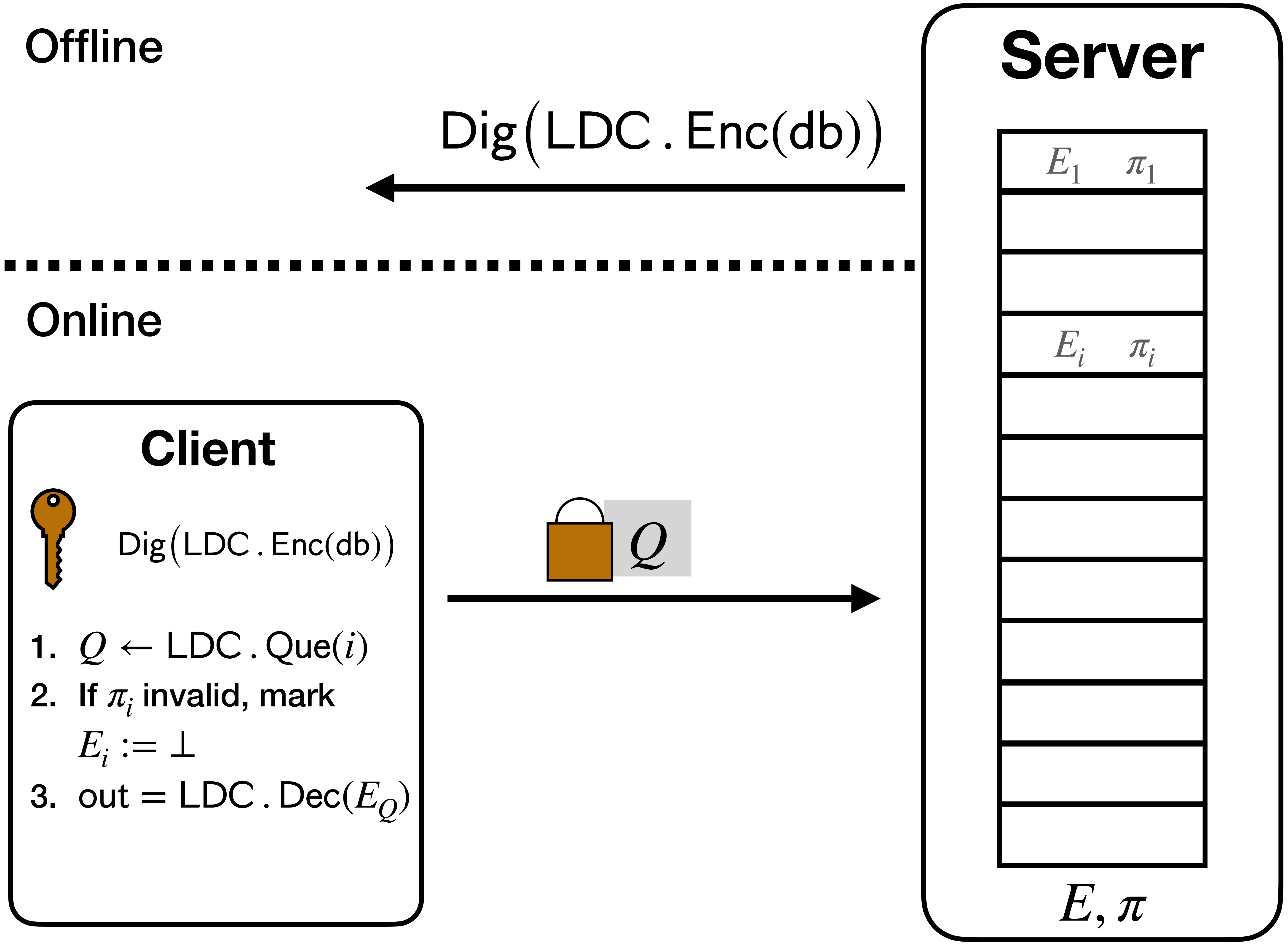
Final Construction



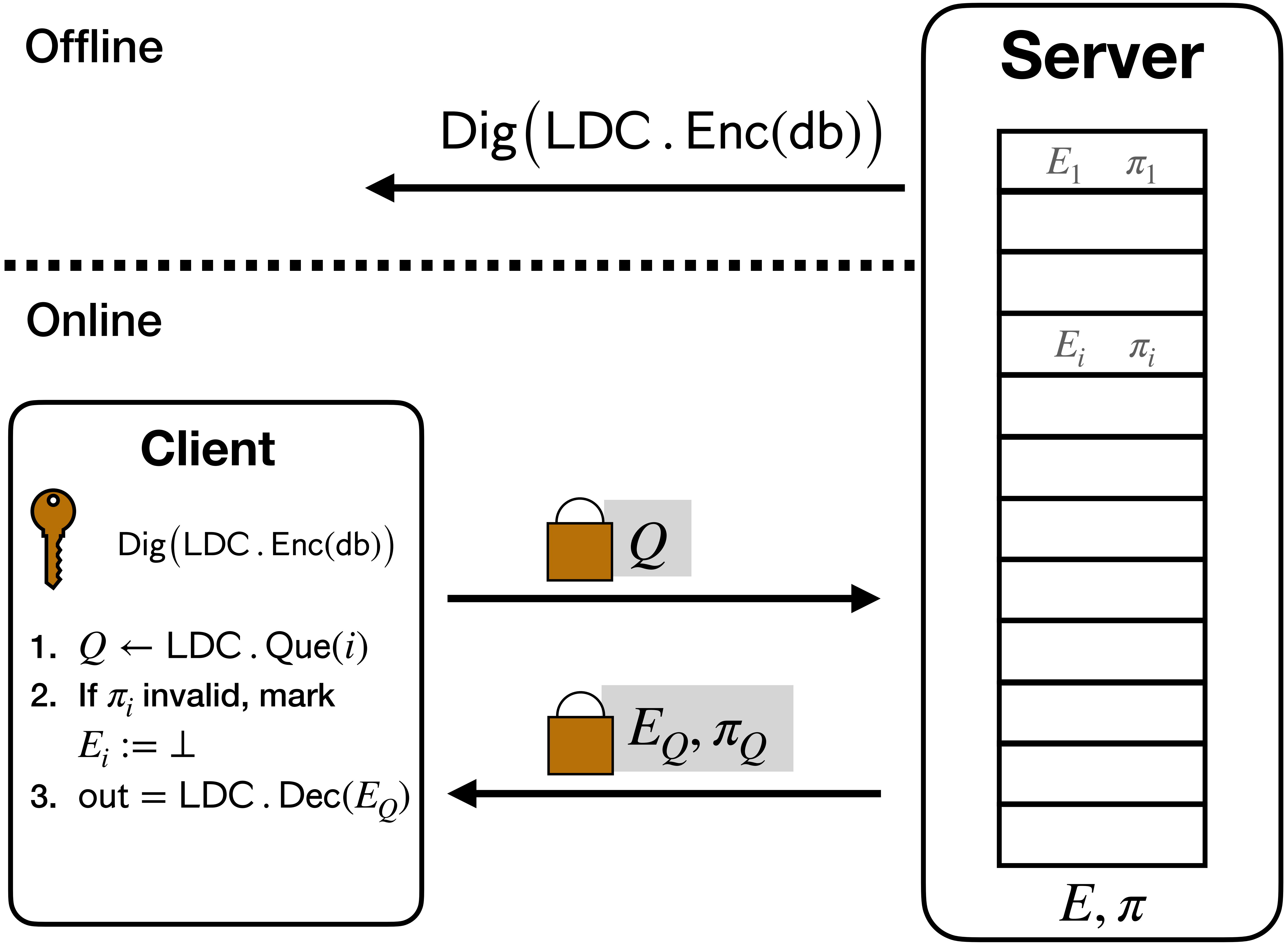
Final Construction



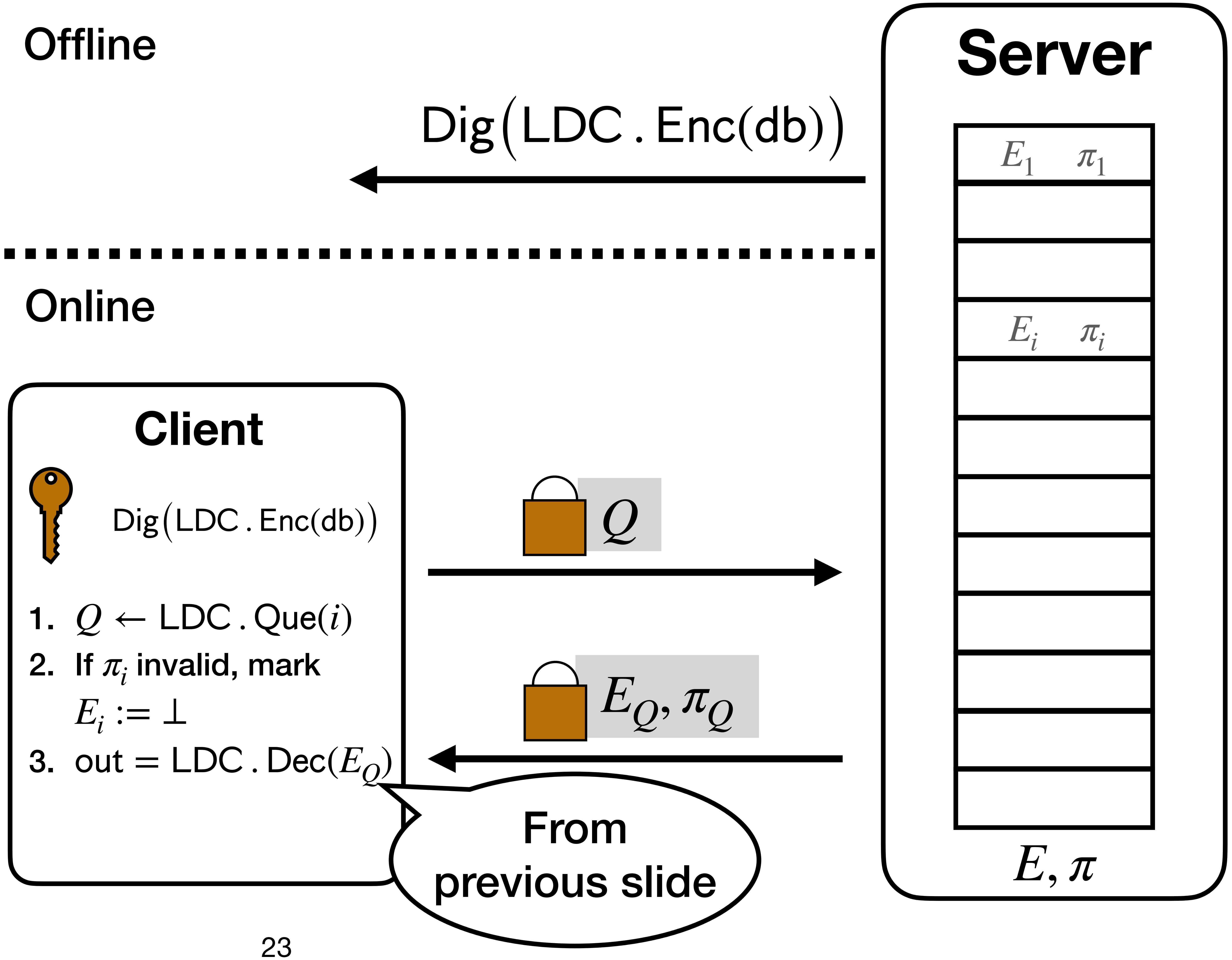
Final Construction



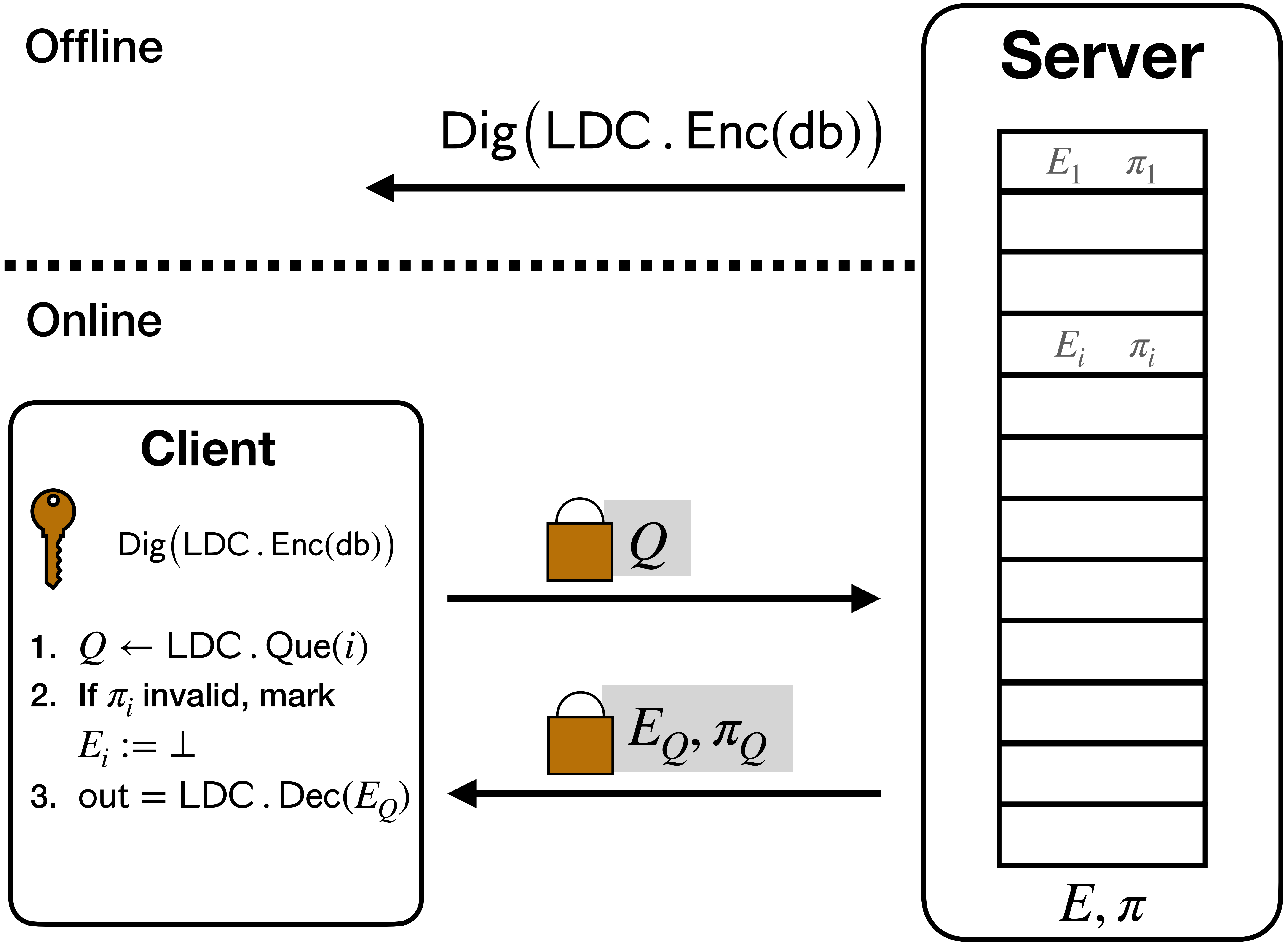
Final Construction



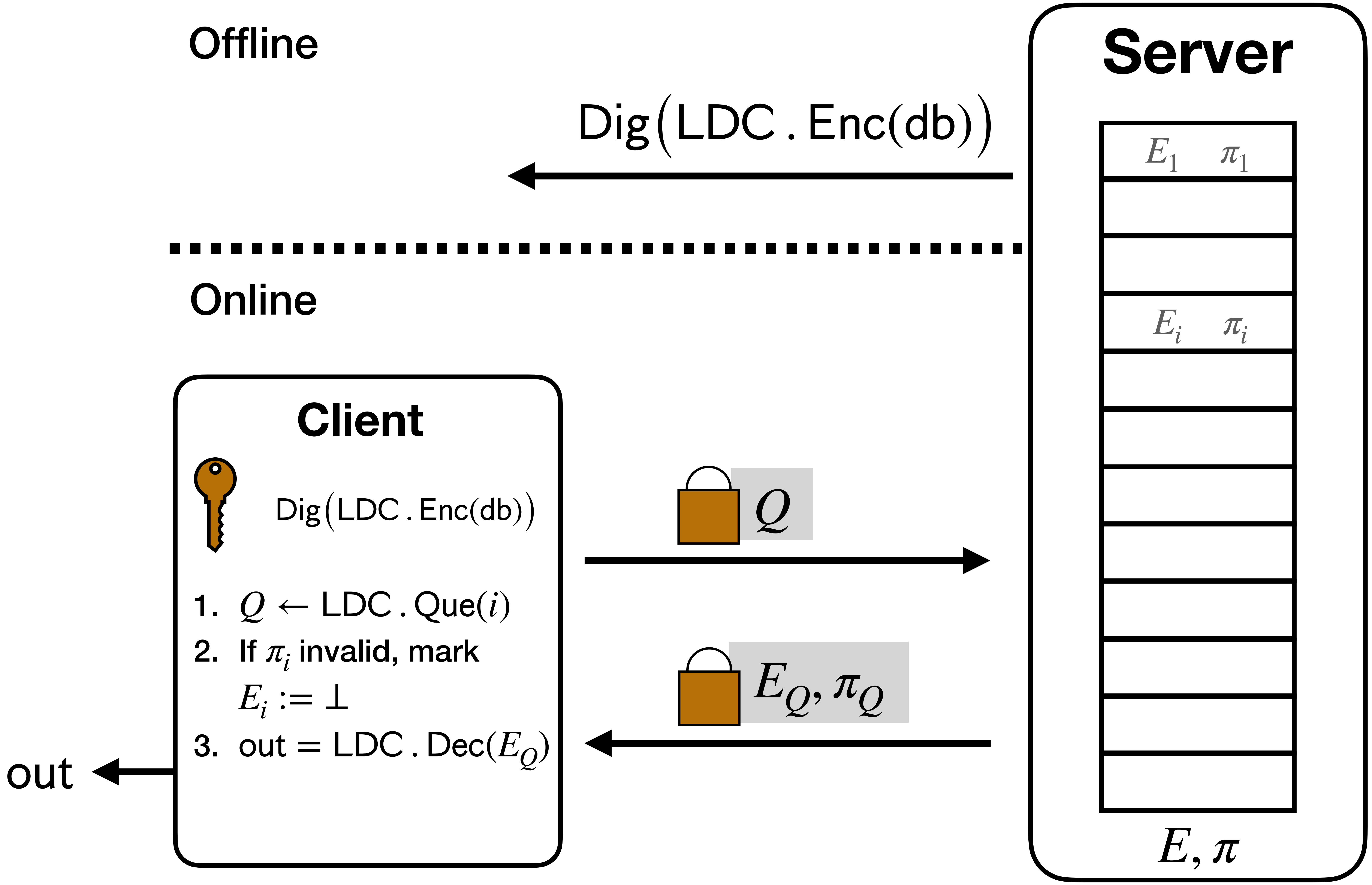
Final Construction



Final Construction

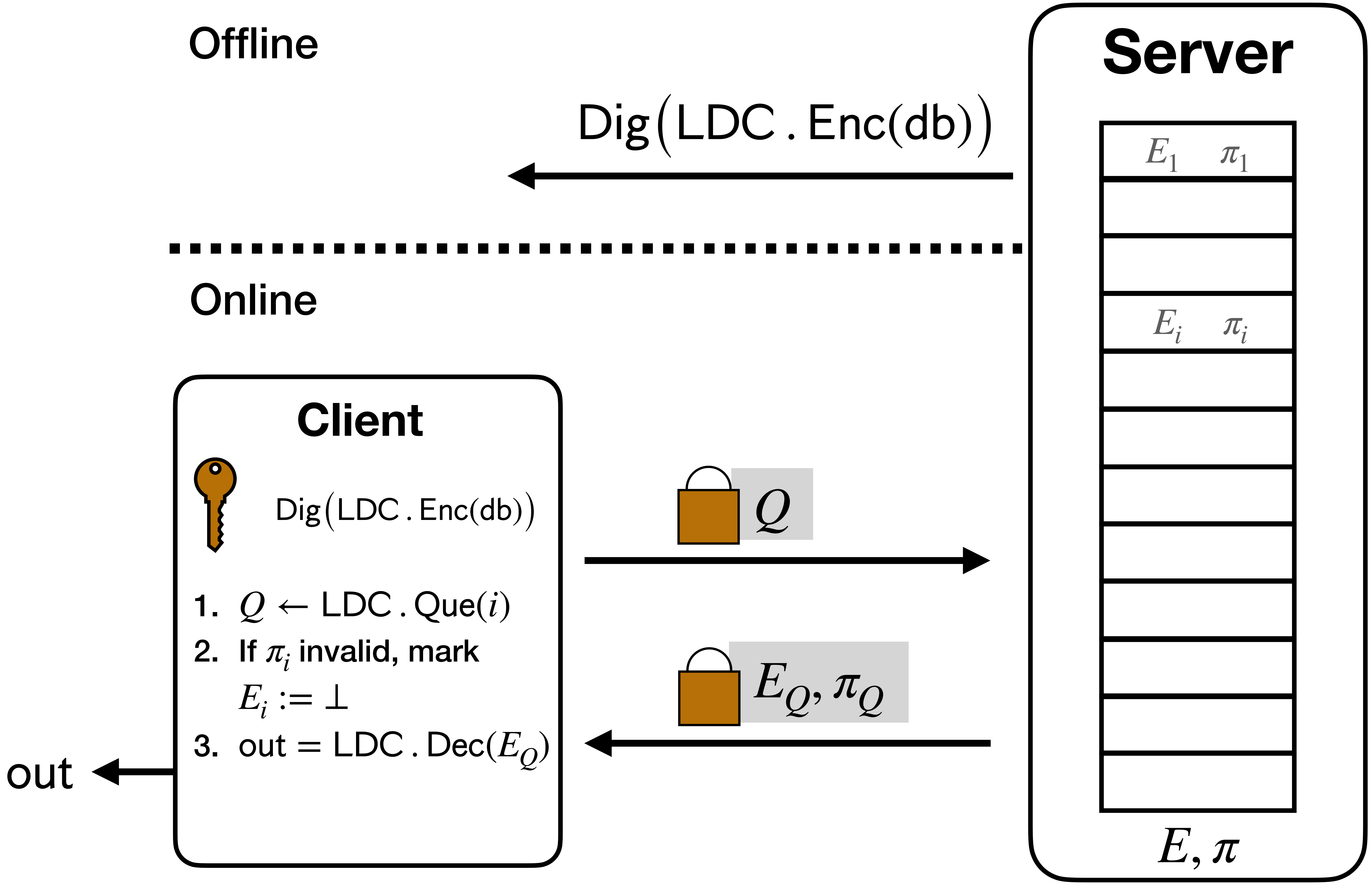


Final Construction



Final Construction

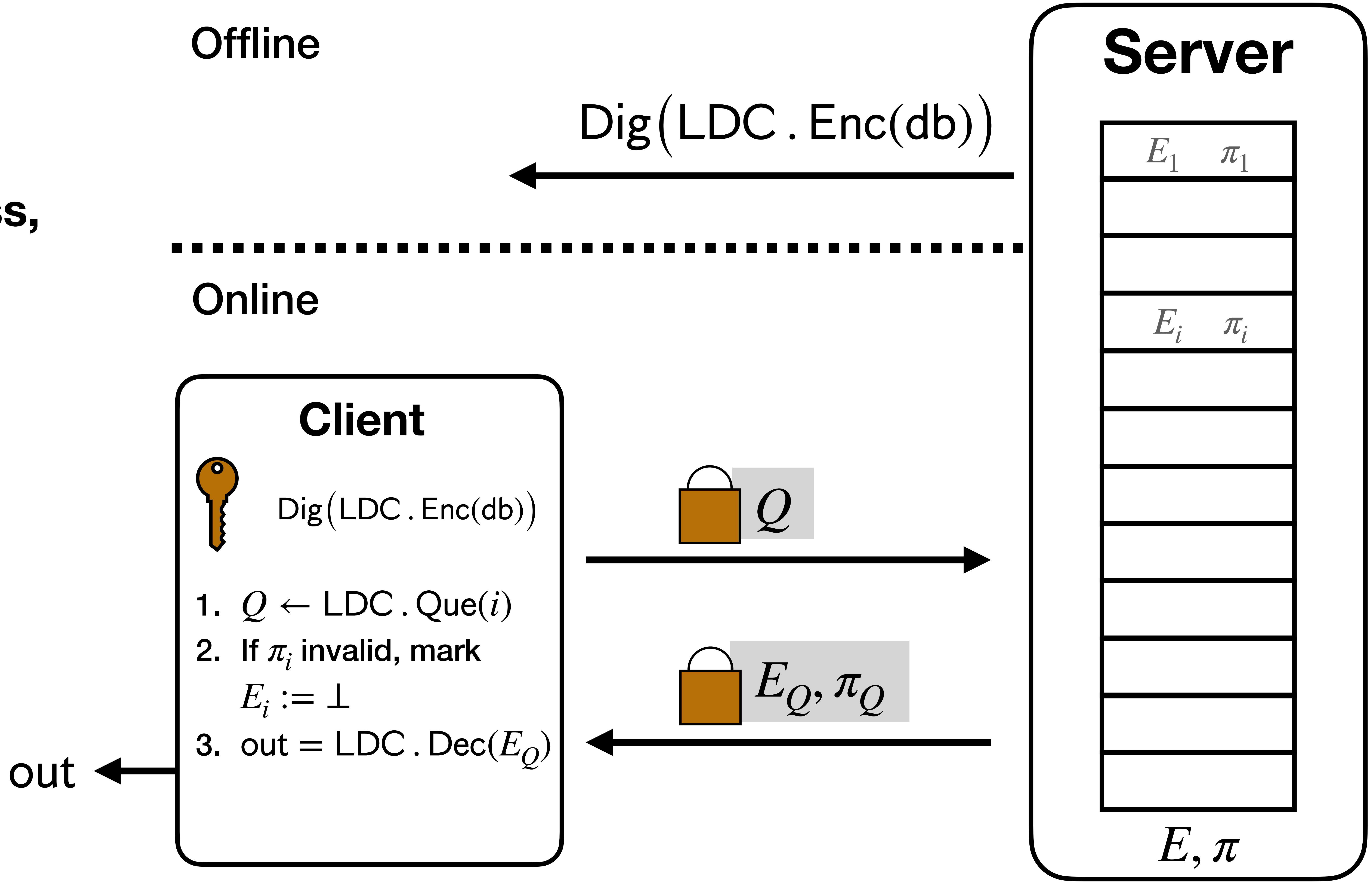
Properties



Final Construction

Properties

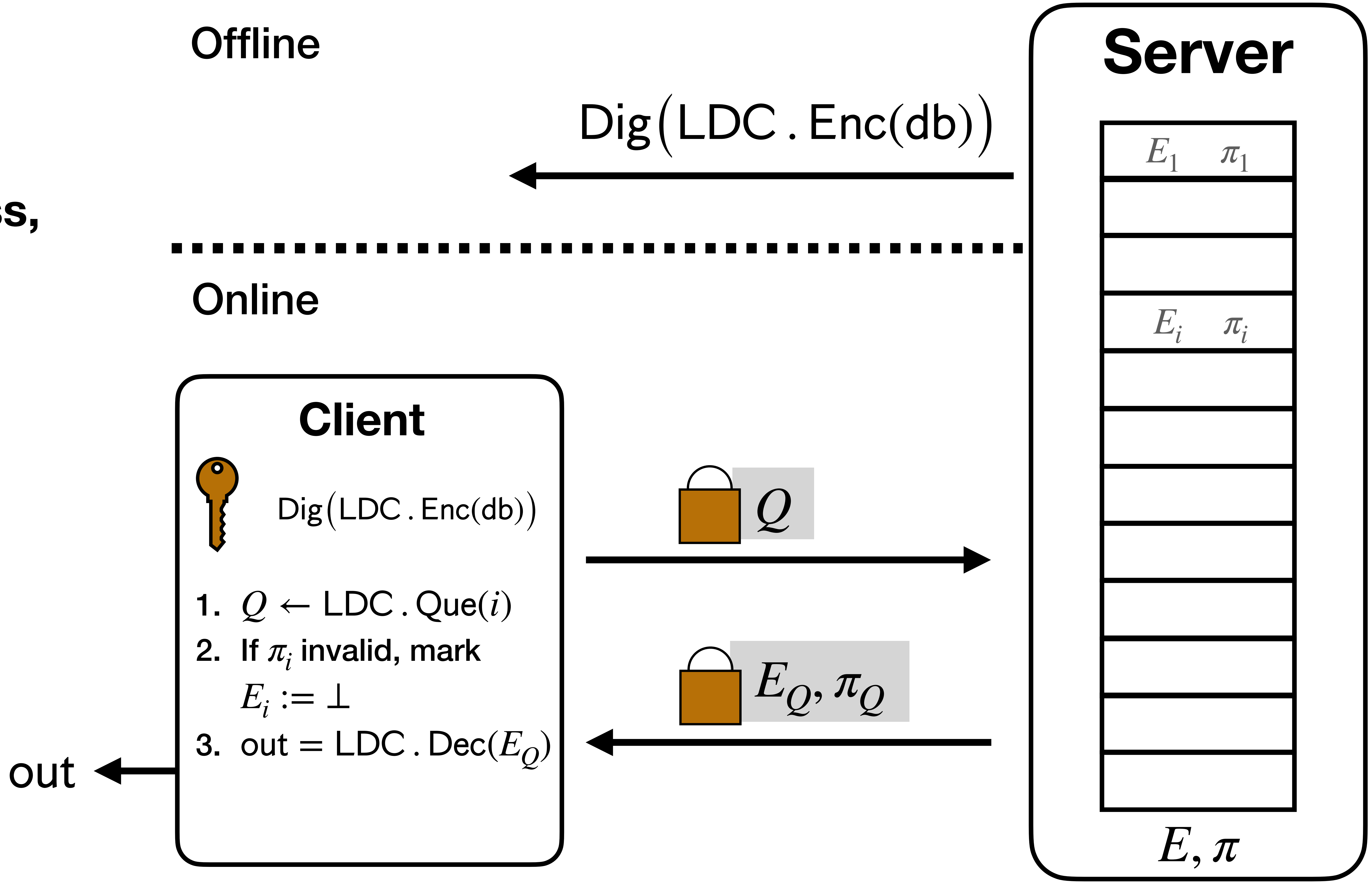
1. **Preserves correctness, coherence.**



Final Construction

Properties

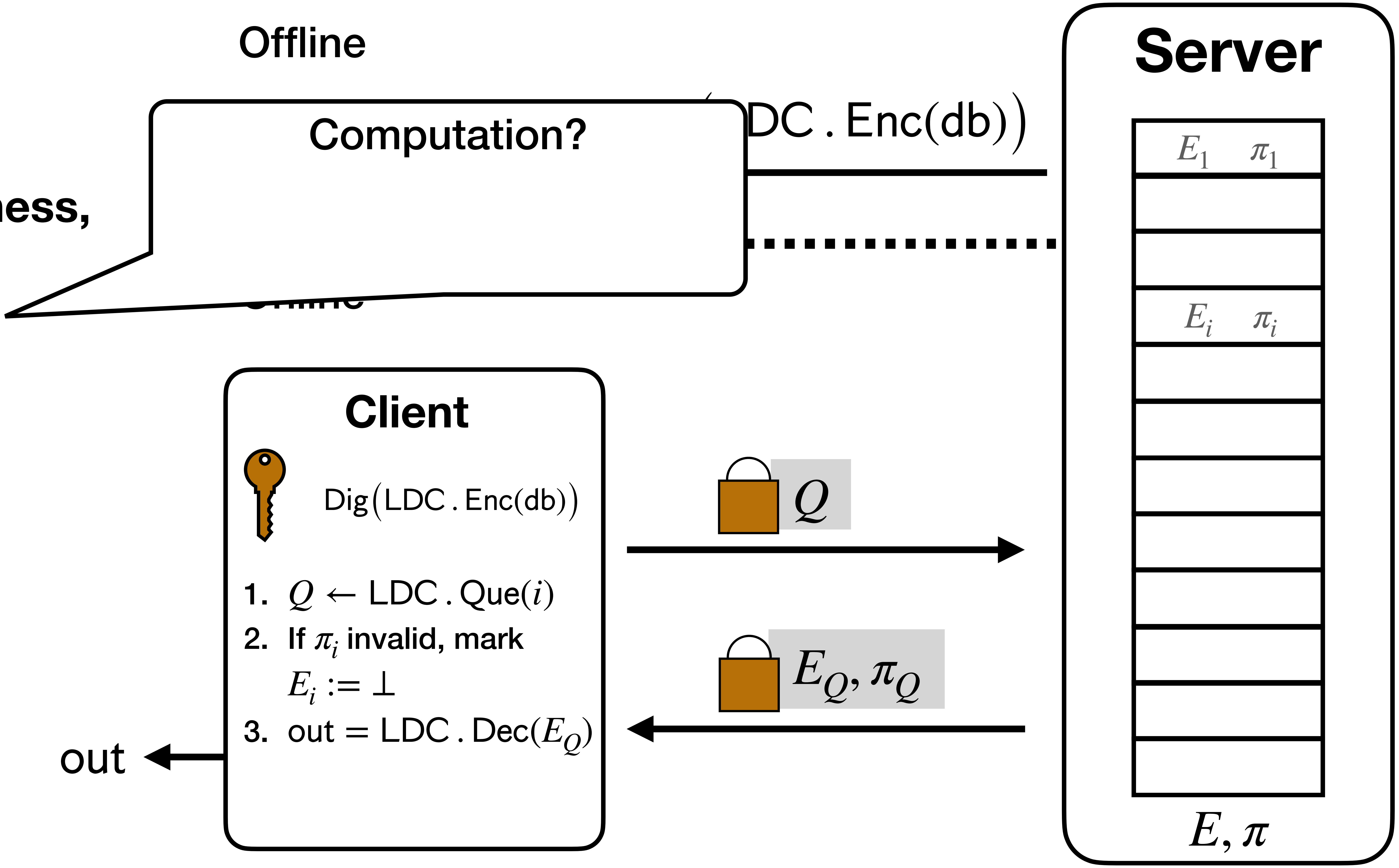
1. **Preserves correctness, coherence.**
2. $O(N^{\epsilon})$ **overhead*.**



Final Construction

Properties

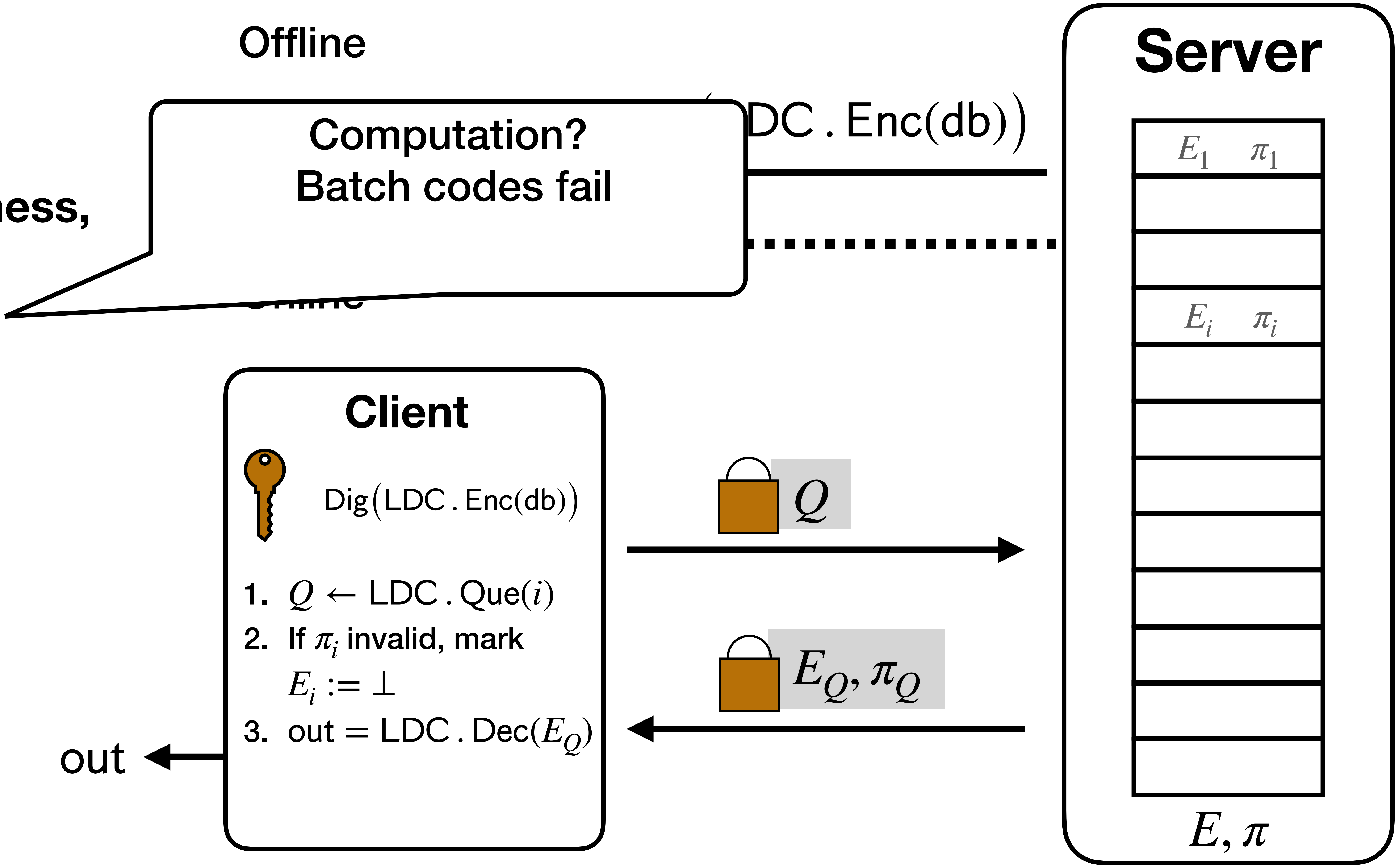
1. **Preserves correctness, coherence.**
2. $O(N^{\epsilon})$ **overhead*.**



Final Construction

Properties

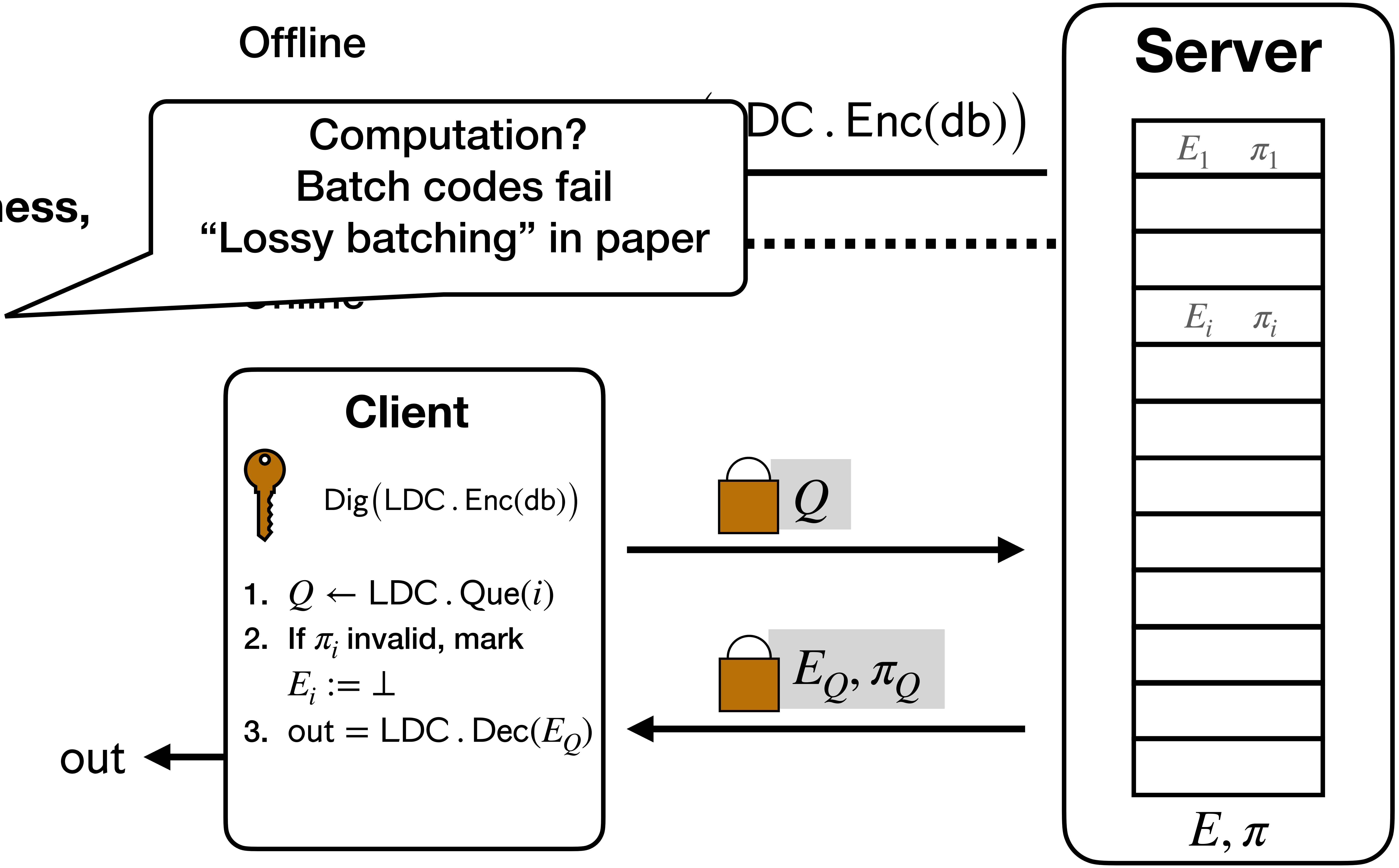
1. **Preserves correctness, coherence.**
2. $O(N^\epsilon)$ **overhead*.**



Final Construction

Properties

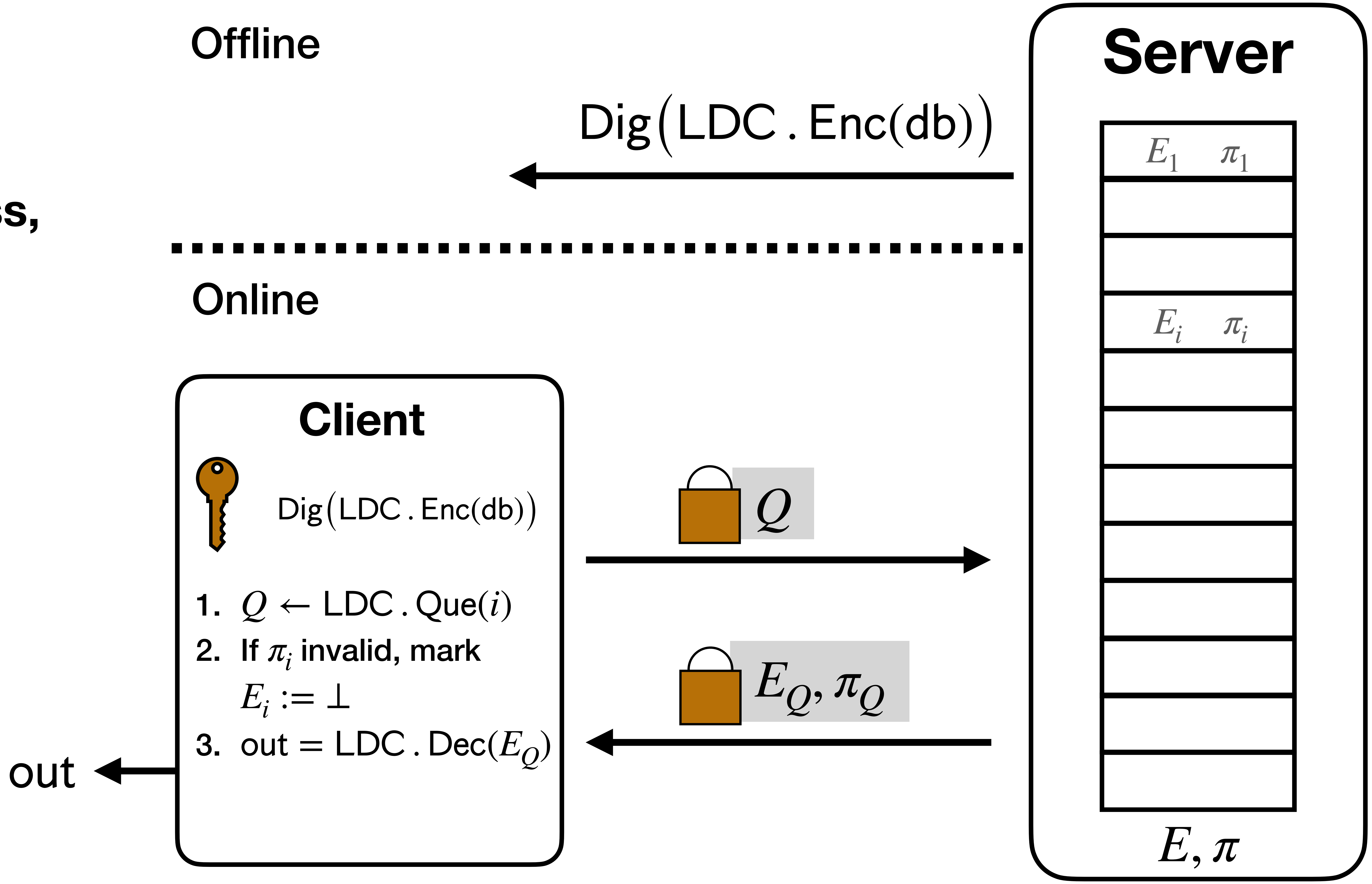
1. **Preserves correctness, coherence.**
2. $O(N^{\epsilon})$ **overhead*.**



Final Construction

Properties

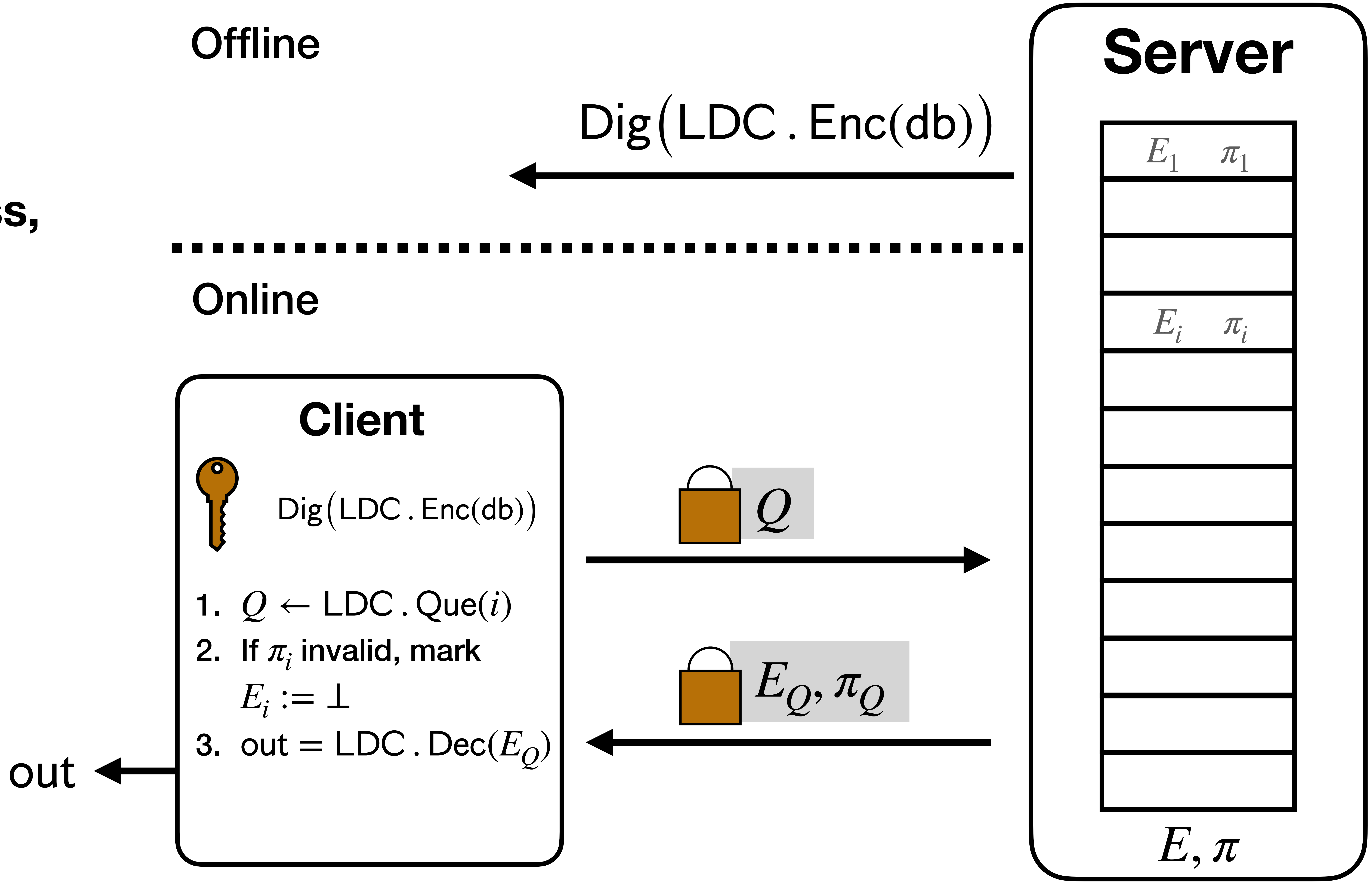
1. **Preserves correctness, coherence.**
2. $O(N^\epsilon)$ **overhead*.**



Final Construction

Properties

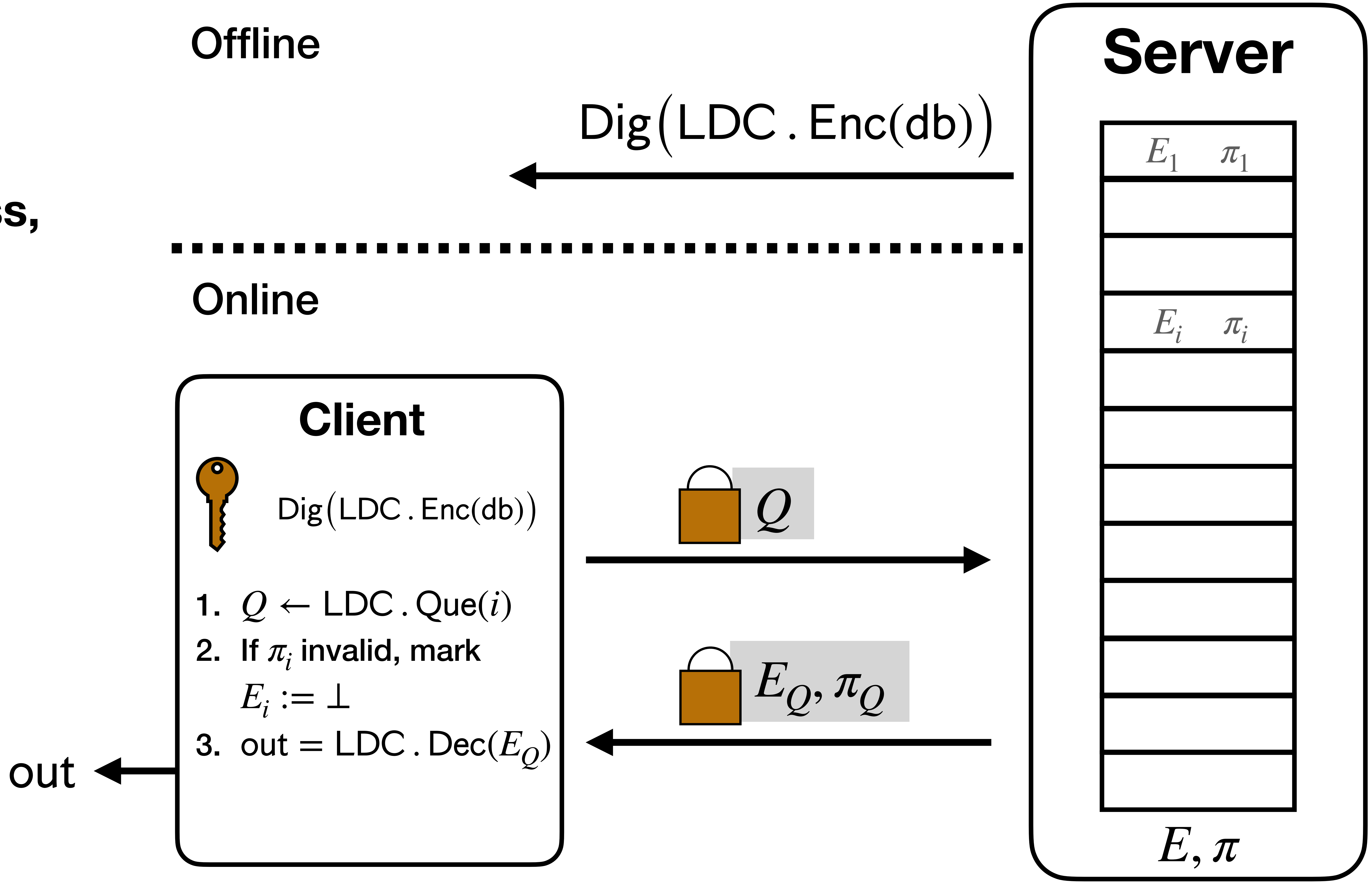
1. **Preserves correctness, coherence.**
2. $O(N^\epsilon)$ **overhead*.**
3. **Privacy:**



Final Construction

Properties

1. **Preserves correctness, coherence.**
2. $O(N^{\epsilon})$ **overhead*.**
3. **Privacy:**



Final Construction

Offline

$\mathrm{Dig}\big(\mathsf{LDC.Enc}(\mathsf{db})\big)$

Properties

1. **Preserves cor**
   coherence.
2. $O(N^{\epsilon})$ **overhe**
3. **Privacy:**

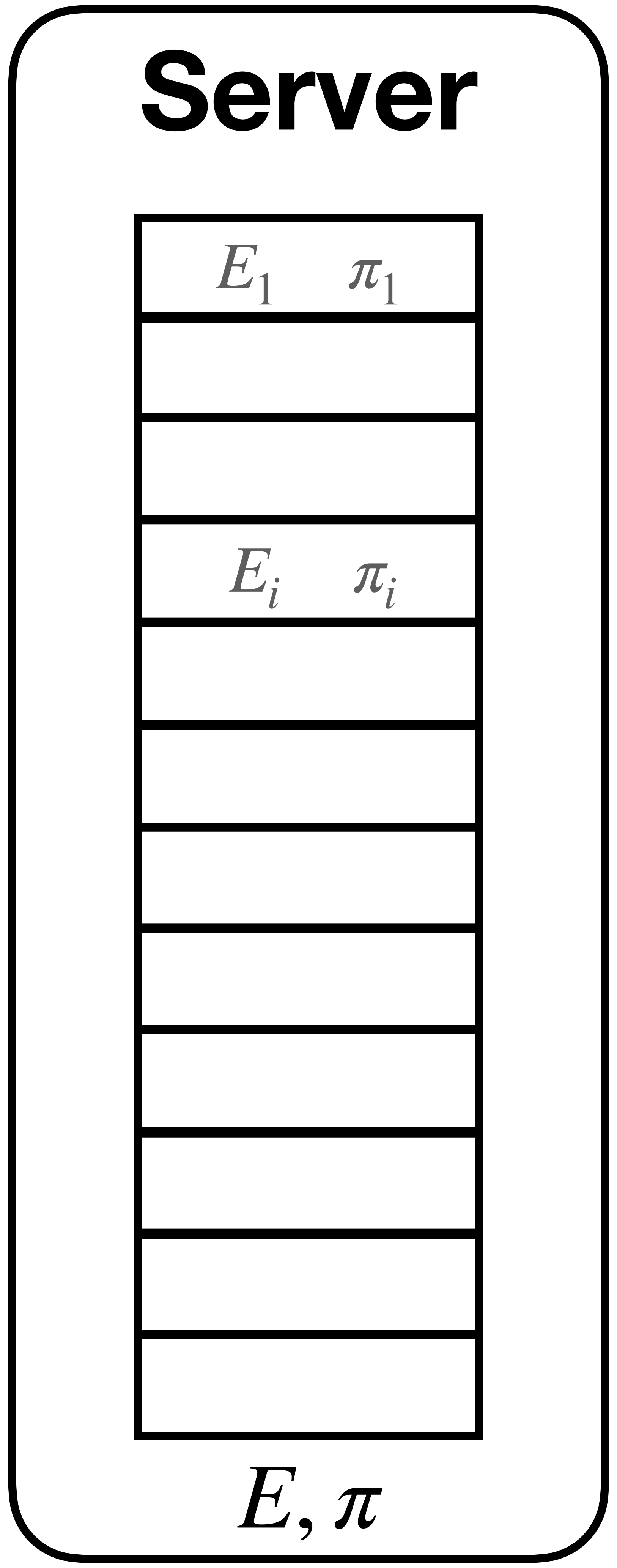
1. $\Pr[\perp$ **on** $i] = \Pr[\perp$ **on** $j]$**: by smoothness of code, test queries are uniformly random and independent of** i **. By non-signaling server must output the same on these distributions.**
2. $\Pr[$**not** $\perp$ **and can't decode**$] = \mathrm{negl}(\lambda)$**: even information theoretic adversary can't guess all test queries!**

$E_i := \perp$

3. $\mathsf{out} = \mathsf{LDC.Dec}(E_Q)$

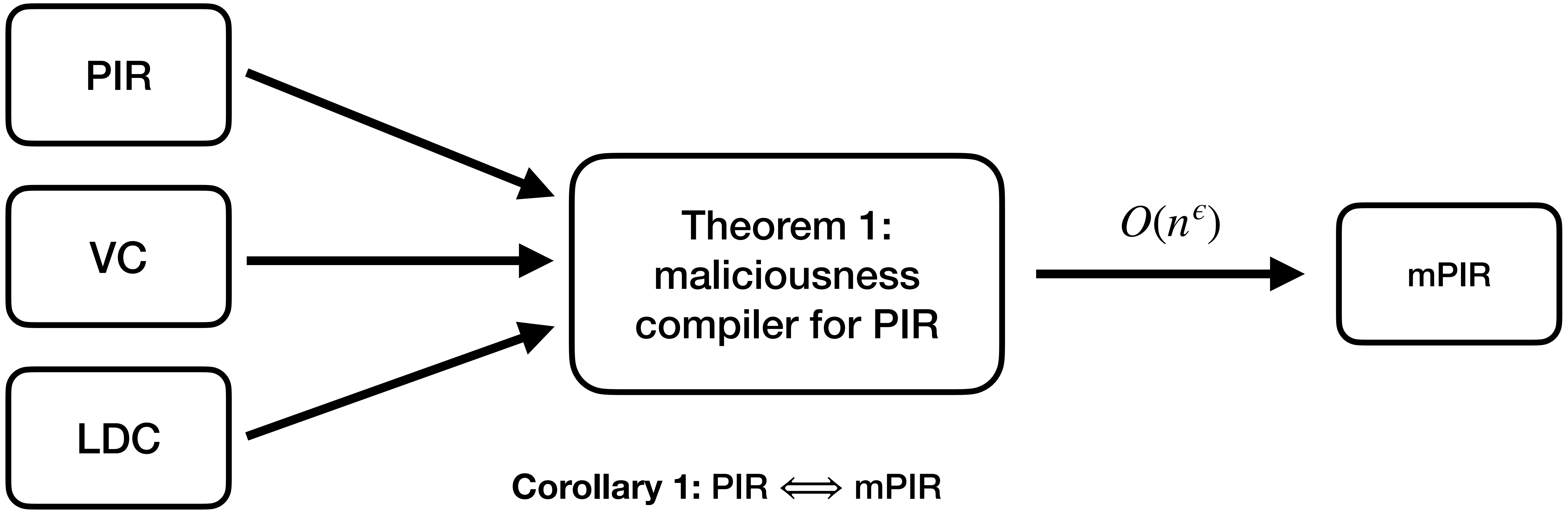
out

E_Q, π_Q



Conclusion

Conclusion



Corollary 1: PIR $\Longleftrightarrow$ mPIR

Theorem 2: there exists doubly-efficient mPIR.

Scheme	Communication	Computation	Digest	Assumptions	Methodology
CNCWF23	$O\left(N^{1/2}\right)$	$O(N)$	$O\left(N^{1/2}\right)$	LWE, DDH	Ad-hoc
WZLY23	$O\left(N^{1/2}\right)$	$O\left(N^{1/2}\right)$	$O\left(N^{1/2}\right)$	OWF*	Ad-hoc
DT23	$O\left(N^{1/2}\right)$	$O(N)$	$O\left(N^{1/2}\right)$	DDH	Ad-hoc
CL24	$O\left(N^{1/2}\right)$	$O(N)$	$O\left(N^{1/2}\right)$	LWE	Ad-hoc
Ours (any PIR)	$\times\, O\left(N^{\epsilon}\right)$	$\times\, O(1)$	$\omega(\log N)$	PIR	Compiler
Ours (DePIR)	$O(\text{polylog } N)$	$O(\text{polylog } N)$	$\omega(\log N)$	RingLWE	Compiler

Open questions

Open questions

1. Theory:

Open questions

1. Theory:
   1. Can we reduce test-query overhead from $O(\lambda N^{\epsilon})$ to $O(N^{\epsilon} + \lambda)$

Open questions

1. Theory:
    1. Can we reduce test-query overhead from $O(\lambda N^{\epsilon})$ to $O(N^{\epsilon} + \lambda)$
    2. What are the properties of LDC with “consistent” decoding?

Open questions

1. Theory:
    1. Can we reduce test-query overhead from $O(\lambda N^{\epsilon})$ to $O(N^{\epsilon} + \lambda)$
    2. What are the properties of LDC with "consistent" decoding?
    3. How well can we decode in the face of non-signaling adversaries?

Open questions

1. Theory:
    1. Can we reduce test-query overhead from $O(\lambda N^\epsilon)$ to $O(N^\epsilon + \lambda)$
    2. What are the properties of LDC with “consistent” decoding?
    3. How well can we decode in the face of non-signaling adversaries?
2. Practice:

Open questions

1. Theory:
    1. Can we reduce test-query overhead from $O(\lambda N^{\epsilon})$ to $O(N^{\epsilon} + \lambda)$
    2. What are the properties of LDC with "consistent" decoding?
    3. How well can we decode in the face of non-signaling adversaries?
2. Practice:
    1. Can we implement these ideas in a practically efficient mPIR?

Thank you!

eprint.iacr.org/2024/964