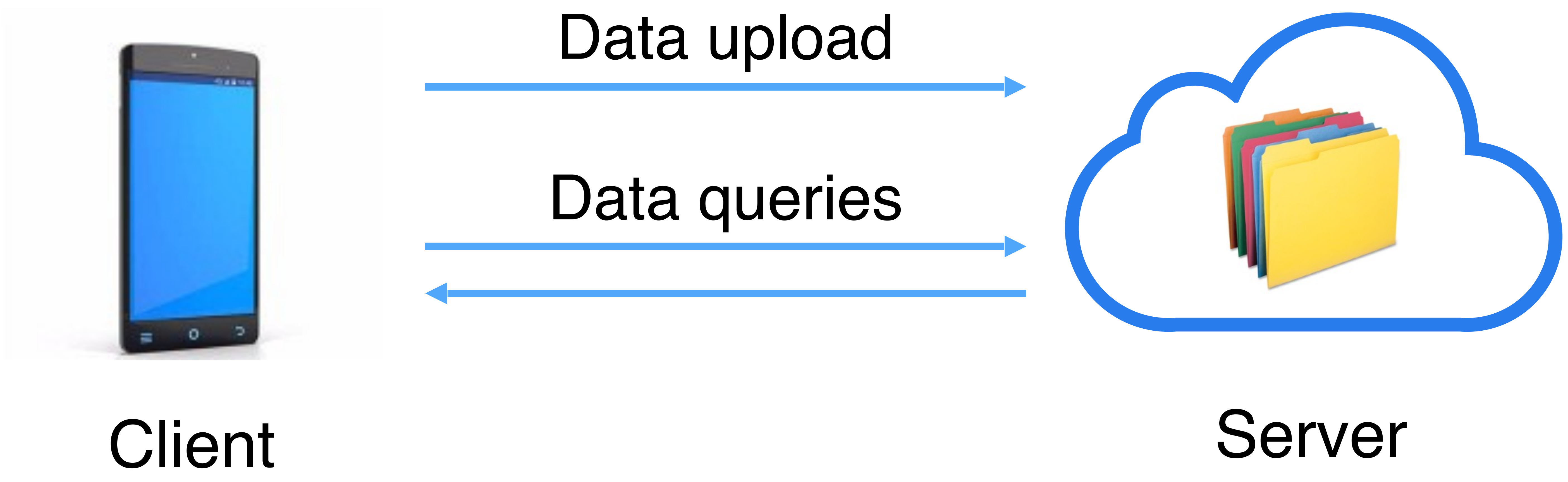
Hermes:

IO-Efficient Forward-Secure Searchable Encryption

Brice Minaud — Inria, ENS, CNRS, PSL University
Michael Reichle — ETH Zürich

Asiacrypt 2023

Outsourcing storage

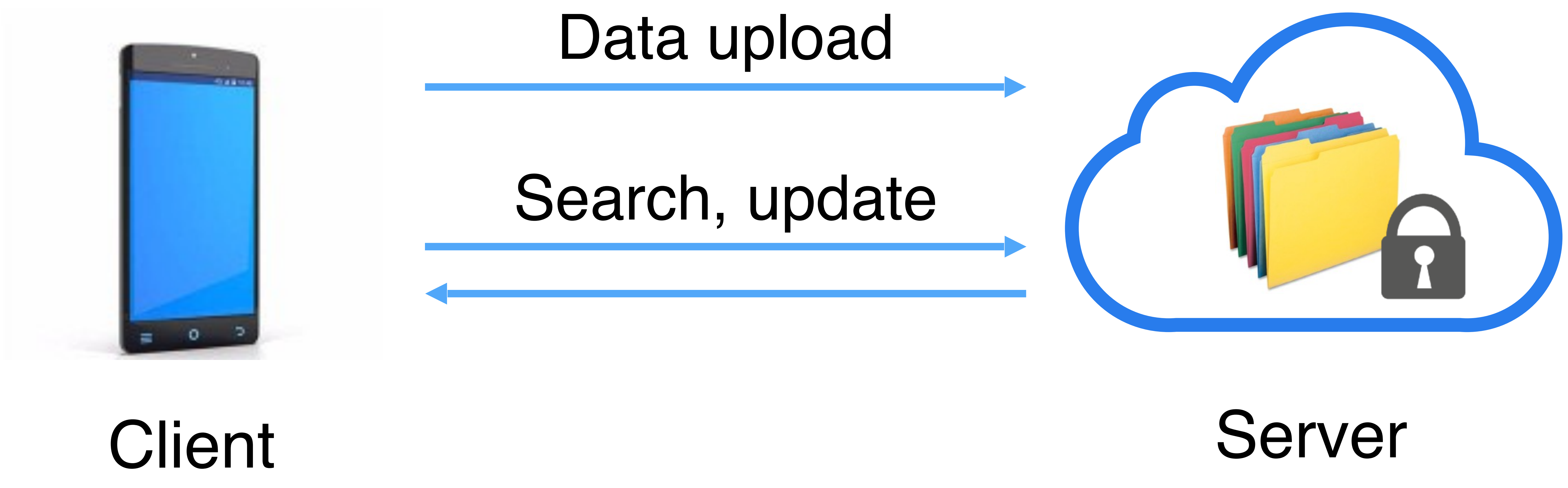


Scenario: Client outsources storage of sensitive data to Server.

Examples:

- Company outsourcing customer/transaction info.
- Private messaging service.

Searchable Encryption



Flavors of computing on encrypted data: FHE, FE, MPC, ORAM...

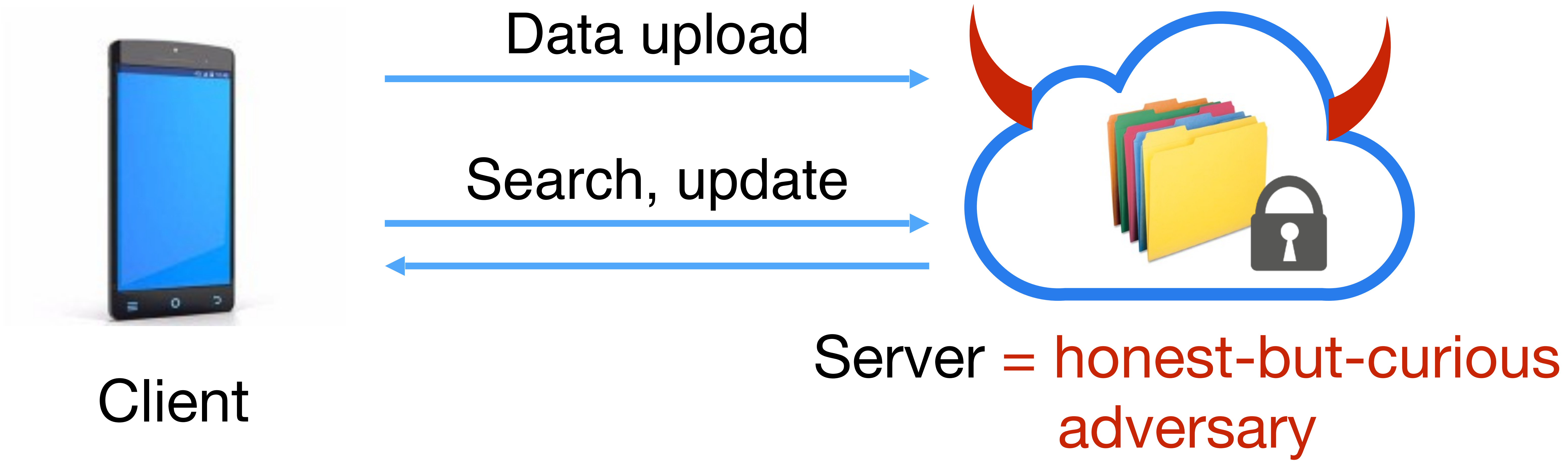
Searchable Symmetric Encryption (SSE):

- High **speed**.

At the cost of:

- Restricted **functionality**. *Search, basic updates.*
- Weaker **security** guarantees.

Security model



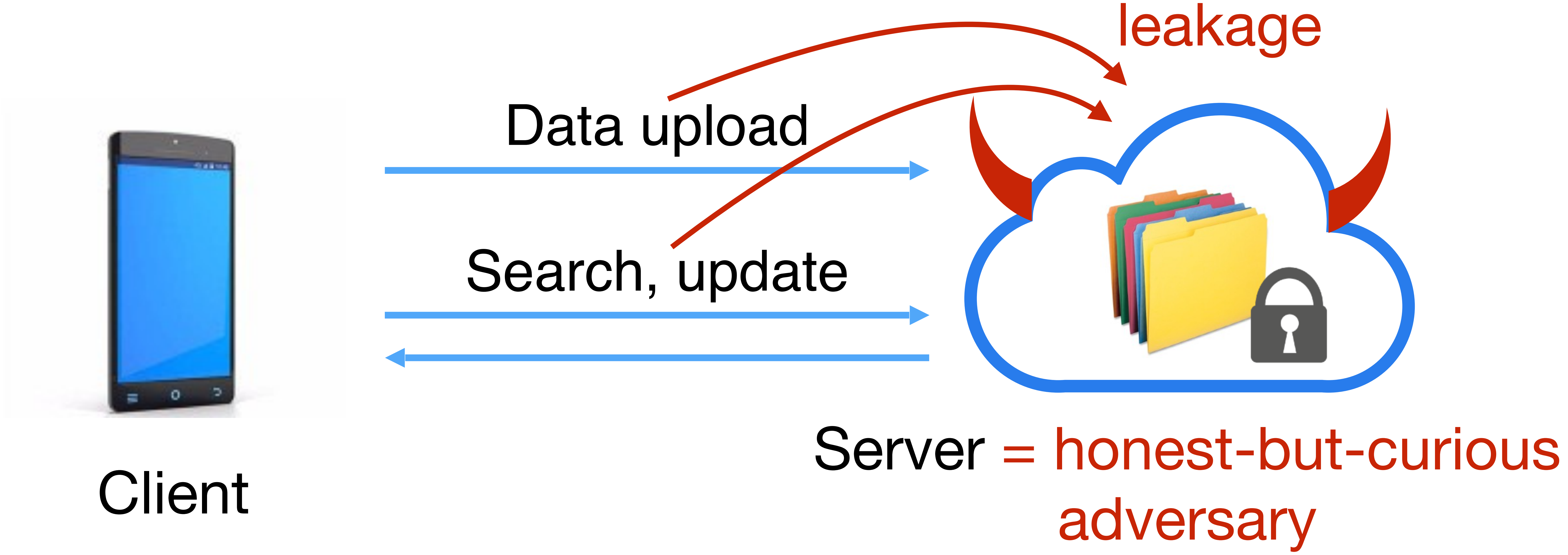
Security model: Server learns **nothing** except specific leakage.

Example:

- **Setup** leaks: total number of elements in database.
- **Search** leaks: IDs of documents matching each query.
- **Forward security:** updates leak no information.

Inference attacks: try to infer sensitive information from leakage.

Security model



Security model: Server learns **nothing** except specific leakage.

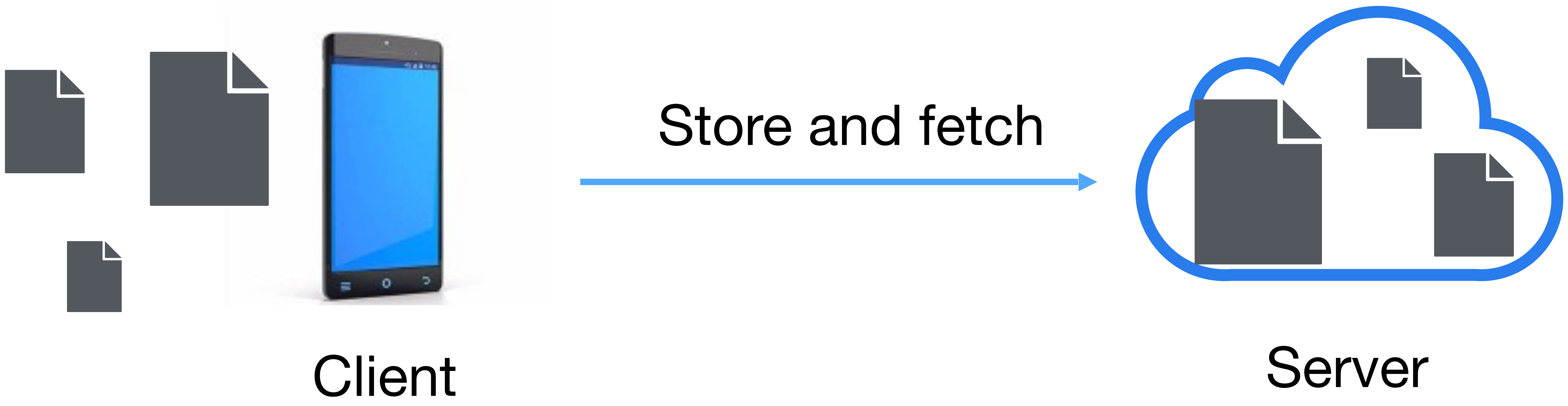
Example:

- **Setup** leaks: total number of elements in database.
- **Search** leaks: IDs of documents matching each query.
- **Forward security:** updates leak no information.

Inference attacks: try to infer sensitive information from leakage.

An inherent bottleneck in SSE

A simple scenario



Minimalist requirement: store and fetch (encrypted) files. *No search. No updates.*

At fetch time:

- The server **can** learn which file is fetched.
- The server **cannot** learn *anything* about other files (→ cannot learn their size).

Simple SSE

id_1 id_2 id_3 ...

Reverse index:

"*car*" $\mapsto$ id_1, id_3

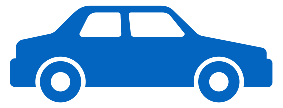
"*duck*" $\mapsto$ id_2, id_3, id_6, ...

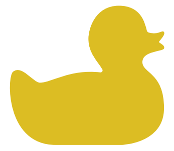
...

Simple SSE

id_1 id_2 id_3 ...

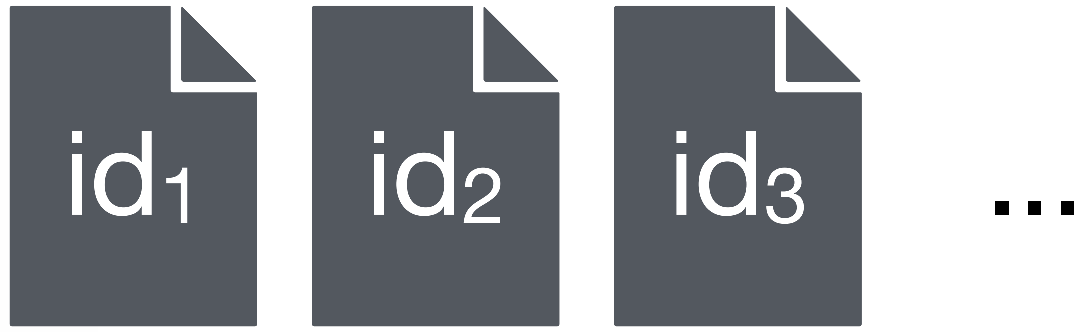
Reverse index:





...

Simple SSE



Reverse index:

...

Simple SSE

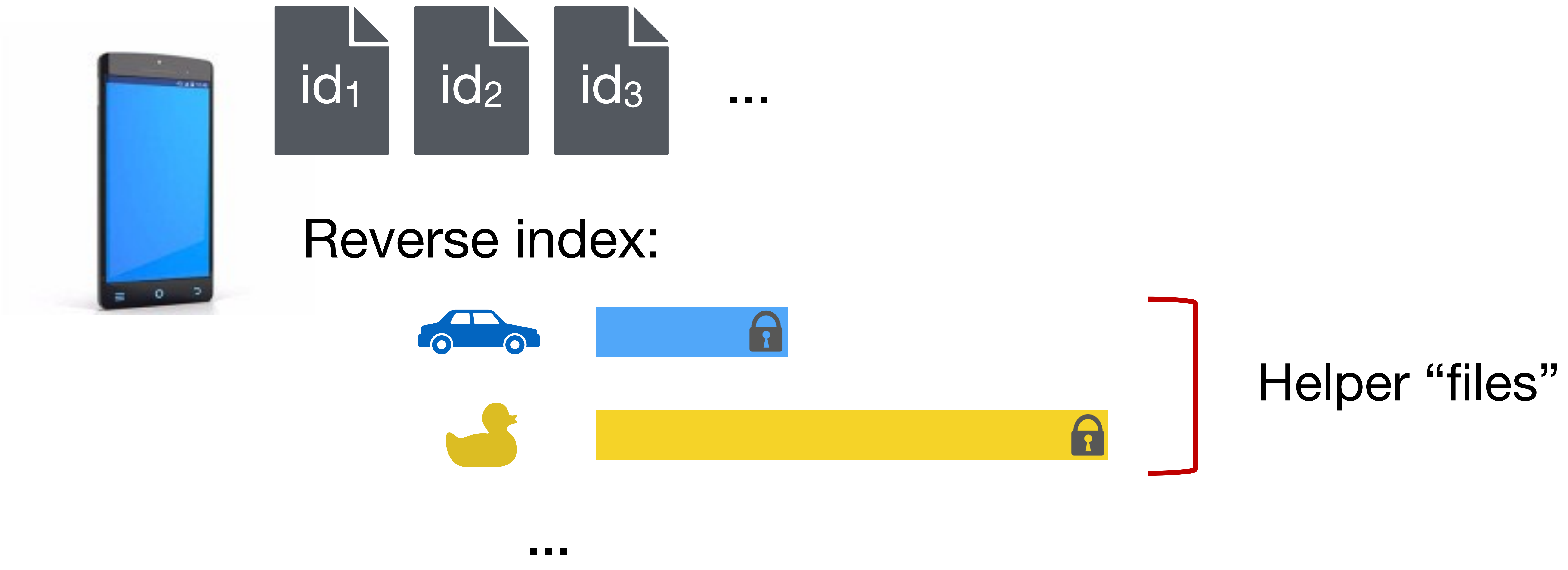
id_1 id_2 id_3 ...

Reverse index:

Helper “files”

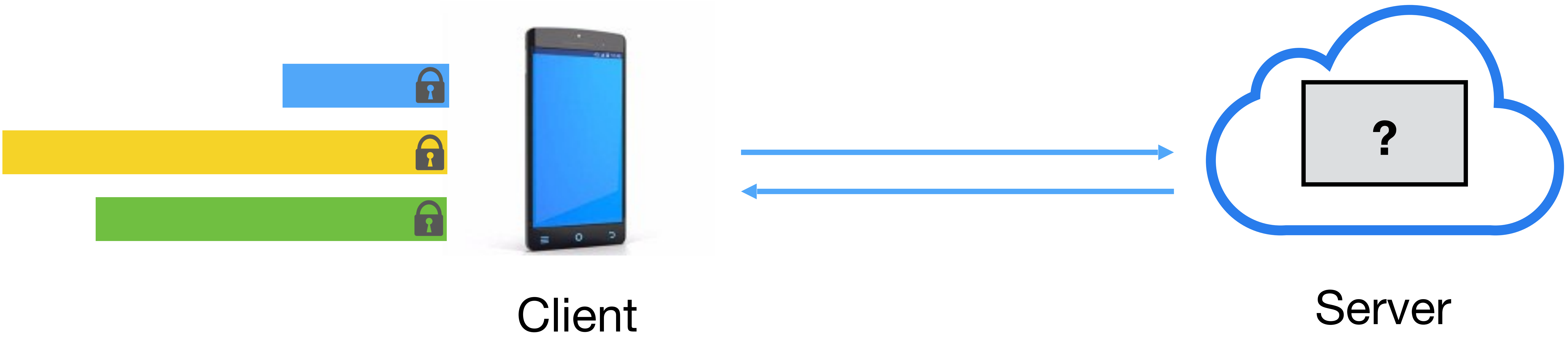
...

Simple SSE

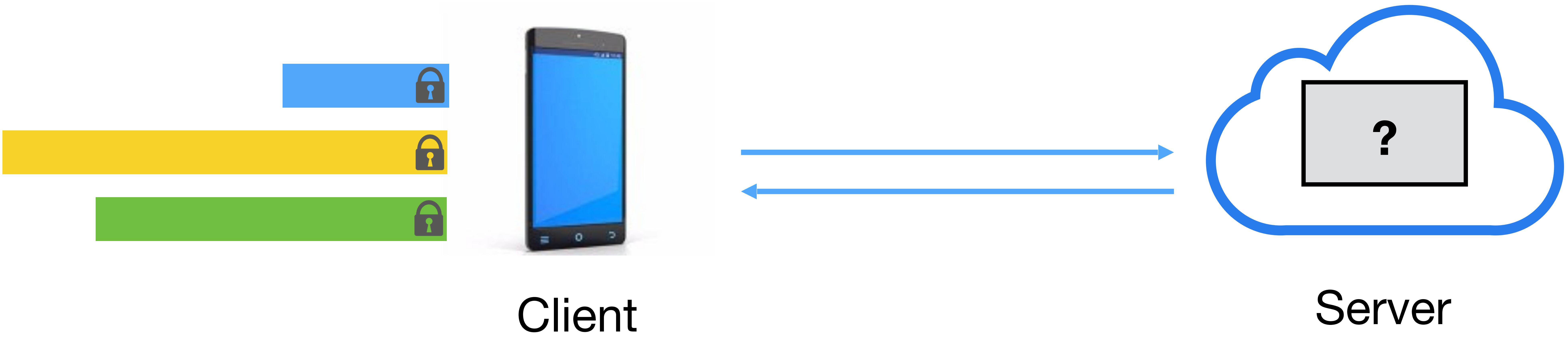


→ size of some files may relate to properties of the original database.

Naive solution



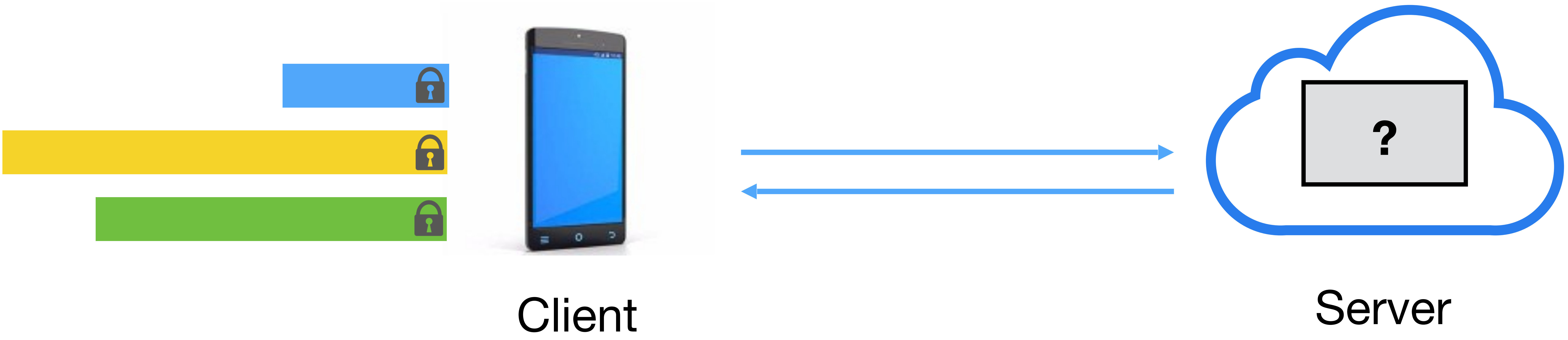
Naive solution



Encrypt files **sequentially**.

Position of one file depends on sizes of other files.

Naive solution

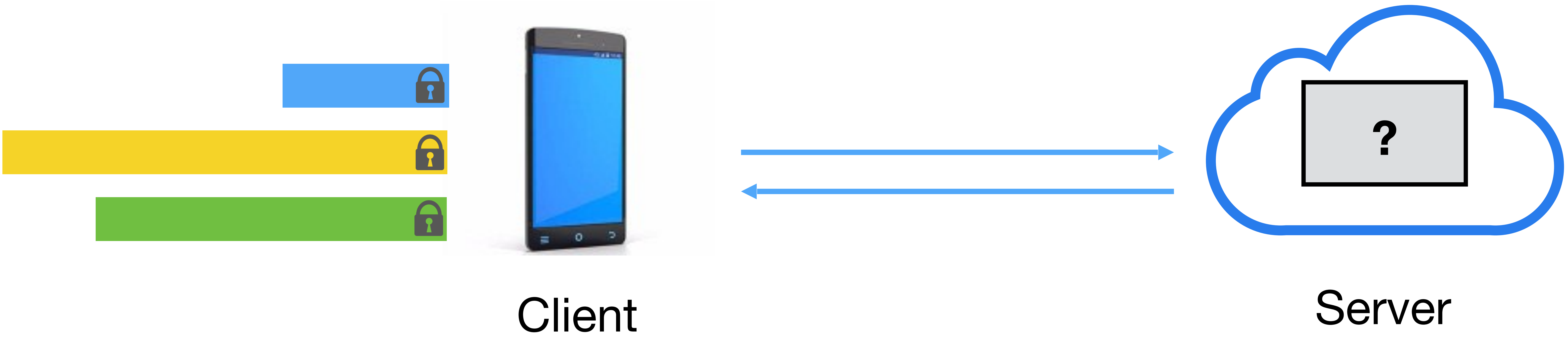


Encrypt files **sequentially**.

✓ Efficient

Position of one file depends on sizes of other files.

Naive solution



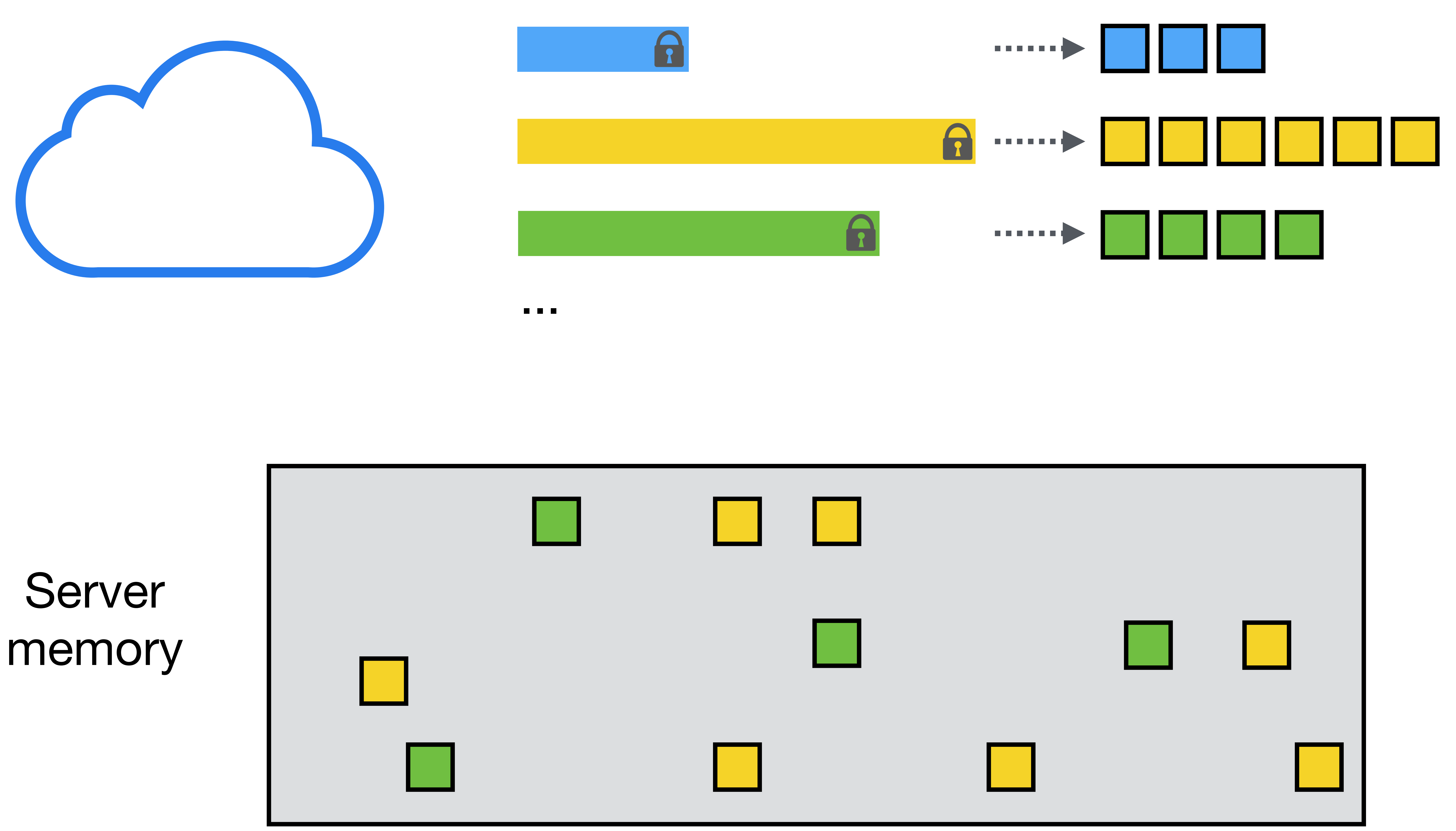
Encrypt files **sequentially**.

✓ Efficient

× Insecure

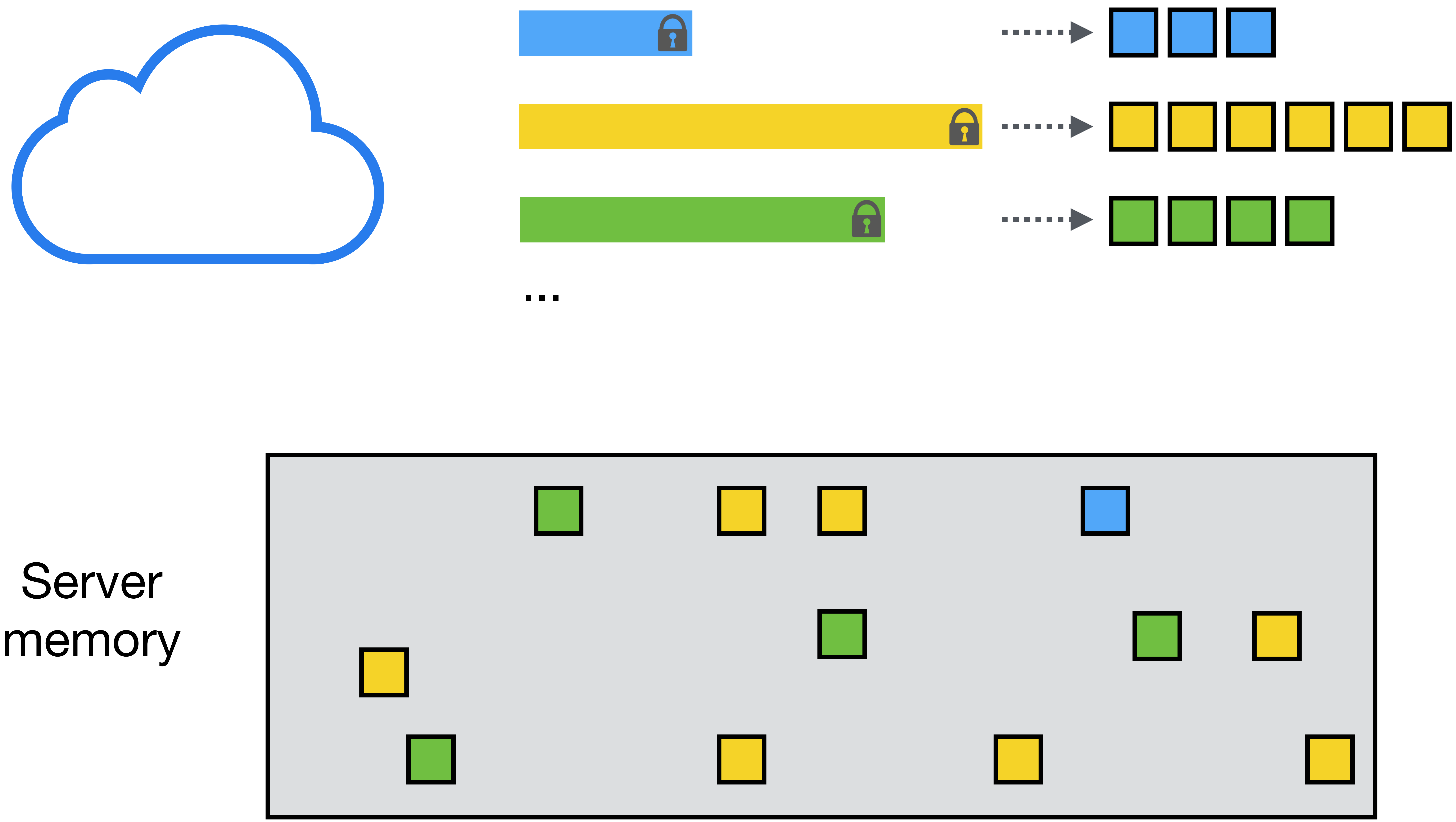
Position of one file depends on sizes of other files.

Standard solution



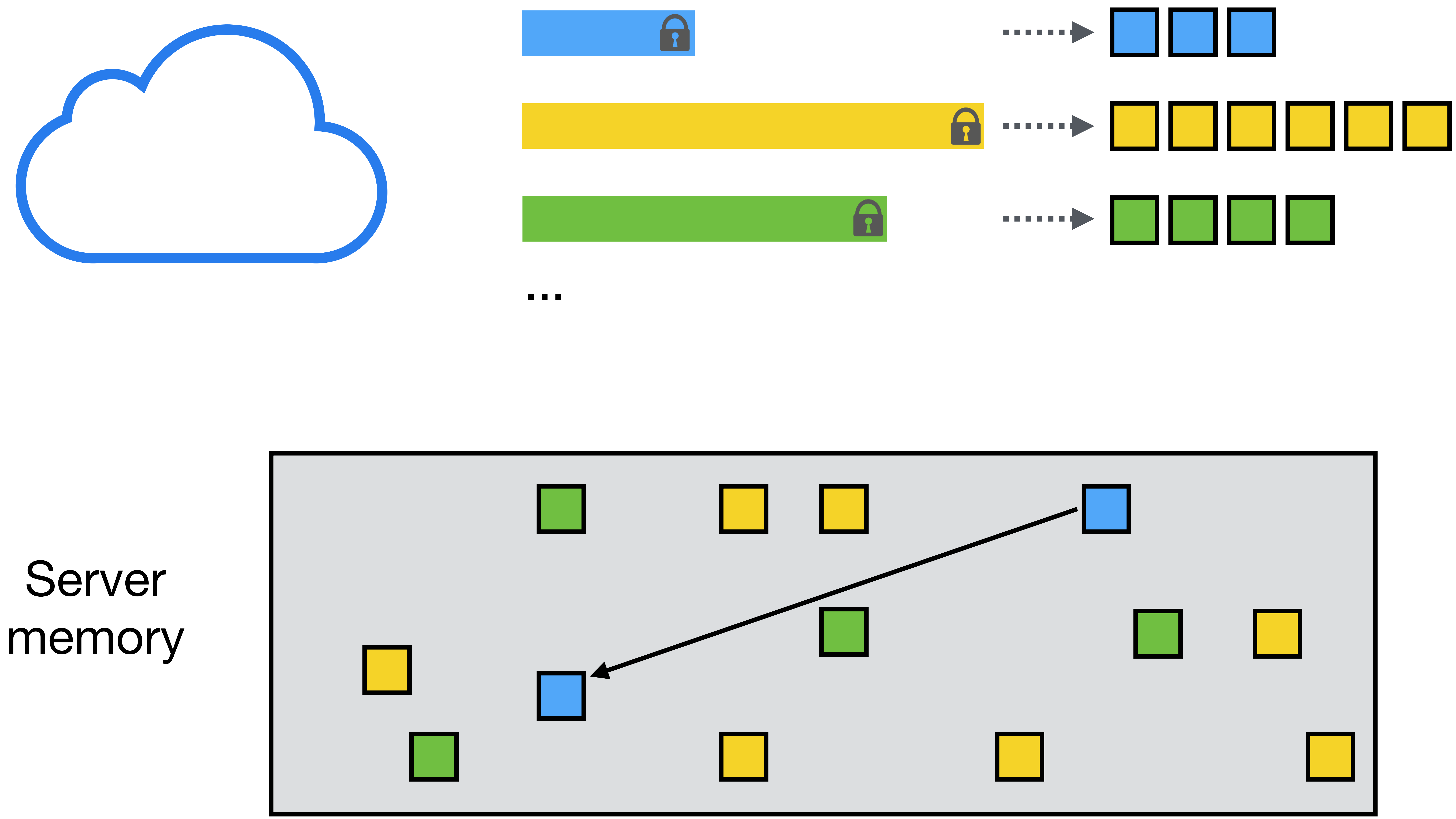
File stored at pseudo-random locations within hash table

Standard solution



File stored at pseudo-random locations within hash table

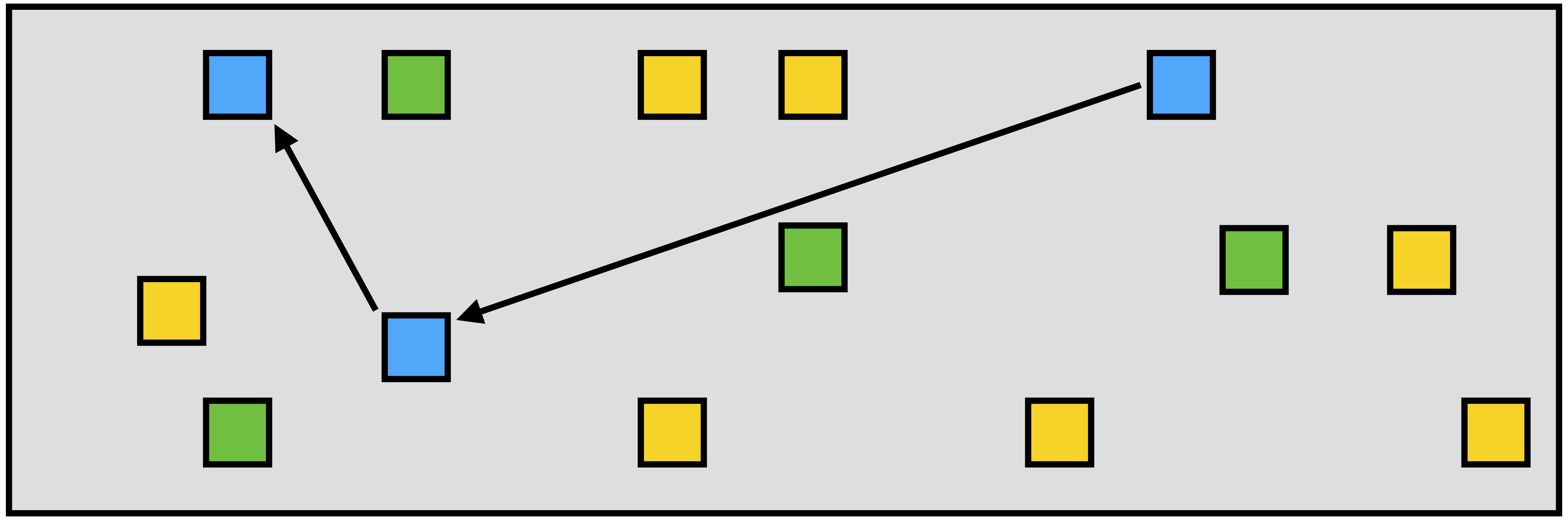
Standard solution



File stored at pseudo-random locations within hash table

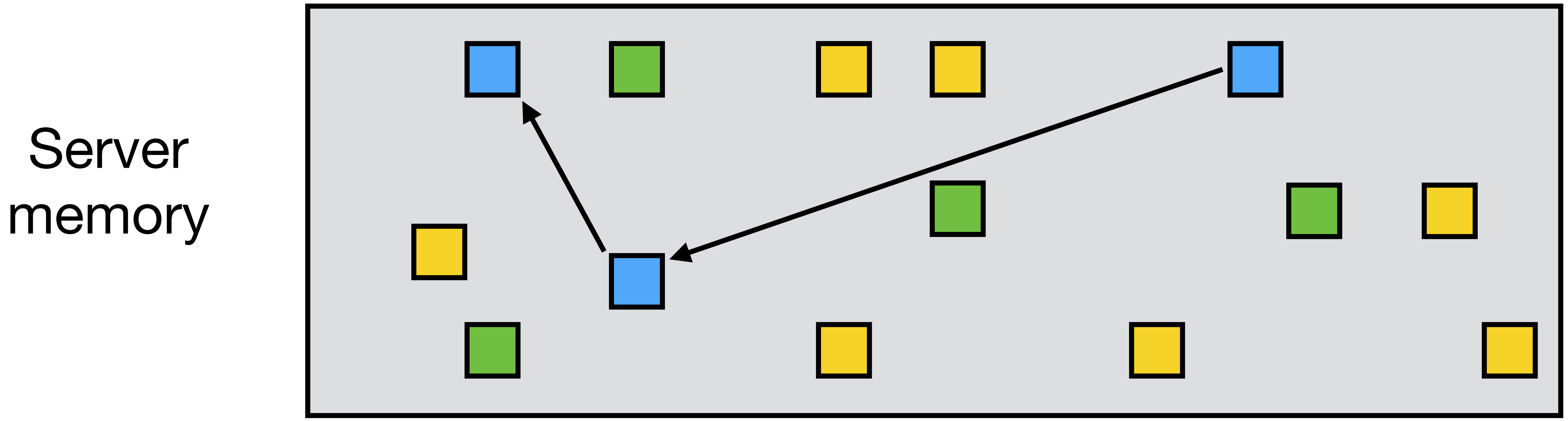
Standard solution

Server memory



File stored at pseudo-random locations within hash table

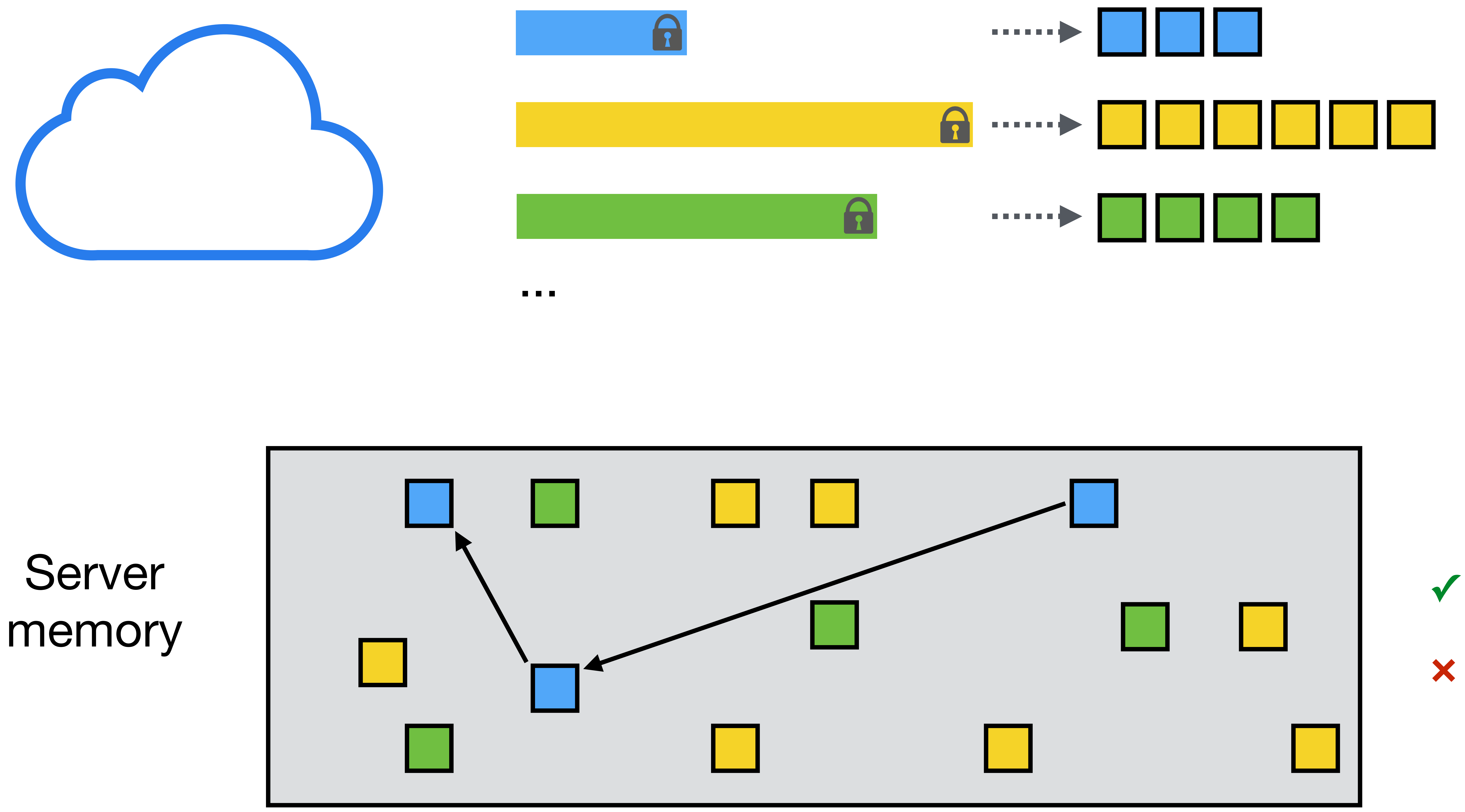
Standard solution



✓ Secure

File stored at pseudo-random locations within hash table

Standard solution



File stored at pseudo-random locations within hash table

✓ Secure

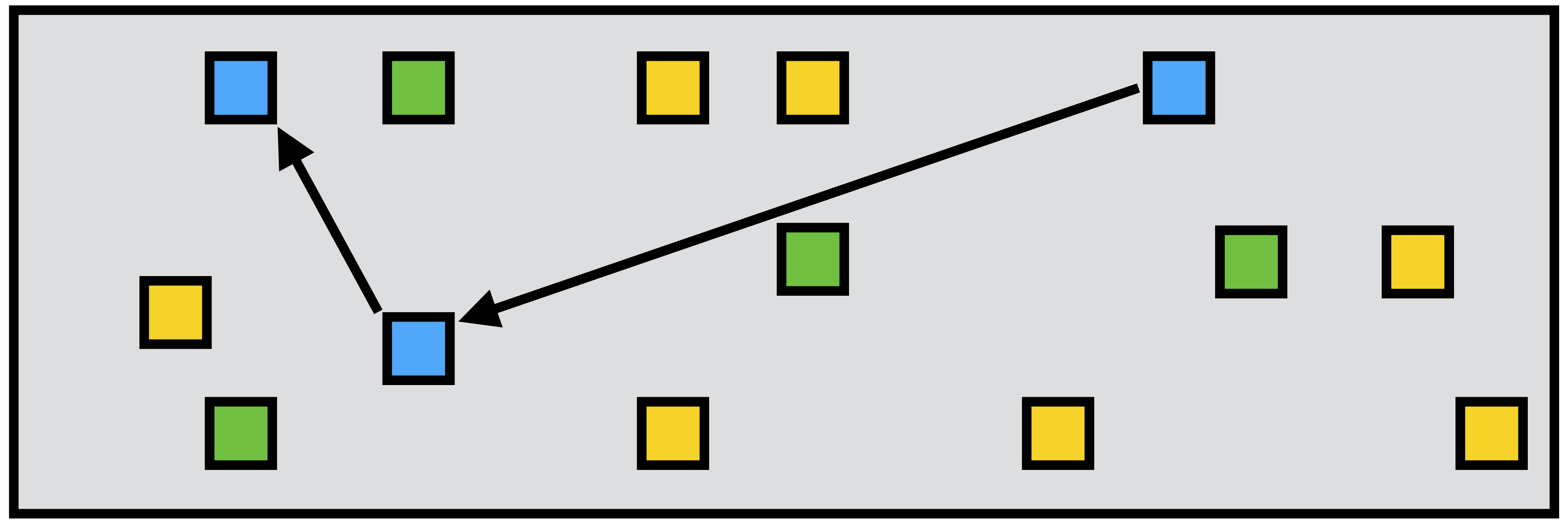
× Inefficient

Problem recap

Sequential storage:

× Insecure

✓ Efficient



Random storage:

✓ Secure

× Insecure

This is inherent.

- **Memory efficiency** asks:

  data position is correlated with content.

- **Security** asks:

  data position is **not** correlated with content.

Formalizing the issue

Cash & Tessaro EC ’15

Locality: #discontinuous memory accesses to fetch enc. file.

Read efficiency: #memory words accessed to fetch enc. file / #memory words of plaintext file.

Storage efficiency: #memory words to store encrypted DB / #memory words of plaintext DB.

Theorem (Cash & Tessaro EC’15):

Insulated file system cannot have O(1) in all 3 measures.

Spawned a long line of work.

Constructions

Asharov, Naor, Segev, Shahaf STOC '16

N = size of DB

Scheme	Locality	Storage eff.	Read eff.
[ANSS16] 1C	O(1)	O(1)	$\tilde{O}(\log N)$
[DP17]	L	O(log N/log L)	O(1)
[DPP18]	O(1)	O(1)	$O(\log^{2/3+\varepsilon} N)$
[MR22]	O(1)	O(1)	$O(\log^{\varepsilon} N)$

Under assumption: longest list size ≤ $N^{1-1/\log\log N}$

[ANSS16] 2C	O(1)	O(1)	$\tilde{O}(\log\log N)$

A second problem

3 efficiency measures: Locality + Read efficiency + Storage efficiency.

Theorem (Cash & Tessaro EC'15):

For **secure** SSE, at least one measure must be $\omega(1)$.

A second problem

3 efficiency measures: Locality + Read efficiency + Storage efficiency.

Theorem (Cash & Tessaro EC'15):

For **secure** SSE, at least one measure must be $\omega(1)$.

Conjecture (Bost CCS'16):

For **forward-secure** SSE, at least one measure must be $\Omega(\log N)$.

Page efficiency

Page efficiency [BBFMR21]

Before:

Storage efficiency + **Locality** + **Read efficiency**

Now:

Storage efficiency: #memory words to store encrypted DB / #memory words of plaintext DB.

Page efficiency: #memory pages accessed to answer a query / #memory pages of plaintext answer.

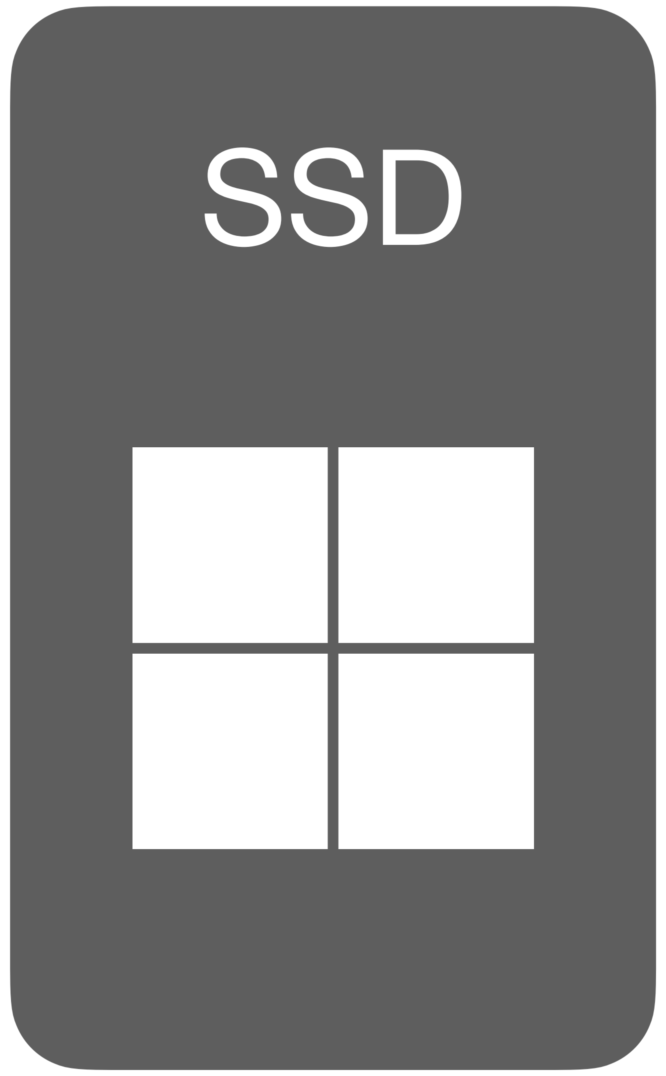
Idea was already implicit in [MM17], to some degree [CJJ+13].

Memory Efficiency

HDDs vs SSDs

Locality:

Number of read (non-adjacent) memory locations



Page Efficiency:

Number of read pages per query

A second problem

Efficiency measures: Locality + Read efficiency + Storage efficiency.

Theorem (Cash & Tessaro EC'15):

For **secure** SSE, at least one measure must be $\omega(1)$.

Conjecture (Bost CCS'16):

For **forward-secure** SSE, at least one measure must be $\Omega(\log N)$.

Efficiency measures: Page efficiency + Storage efficiency.

A second problem

Efficiency measures: Locality + Read efficiency + Storage efficiency.

Theorem (Cash & Tessaro EC'15):

For **secure** SSE, at least one measure must be $\omega(1)$.

Conjecture (Bost CCS'16):

For **forward-secure** SSE, at least one measure must be $\Omega(\log N)$.

Efficiency measures: Page efficiency + Storage efficiency.

Theorem (BBFMR C'21):

For **secure** SSE, can have O(1) in both measures.

A second problem

Efficiency measures: Locality + Read efficiency + Storage efficiency.

Theorem (Cash & Tessaro EC'15):

For **secure** SSE, at least one measure must be $\omega(1)$.

Conjecture (Bost CCS'16):

For **forward-secure** SSE, at least one measure must be $\Omega(\log N)$.

Efficiency measures: Page efficiency + Storage efficiency.

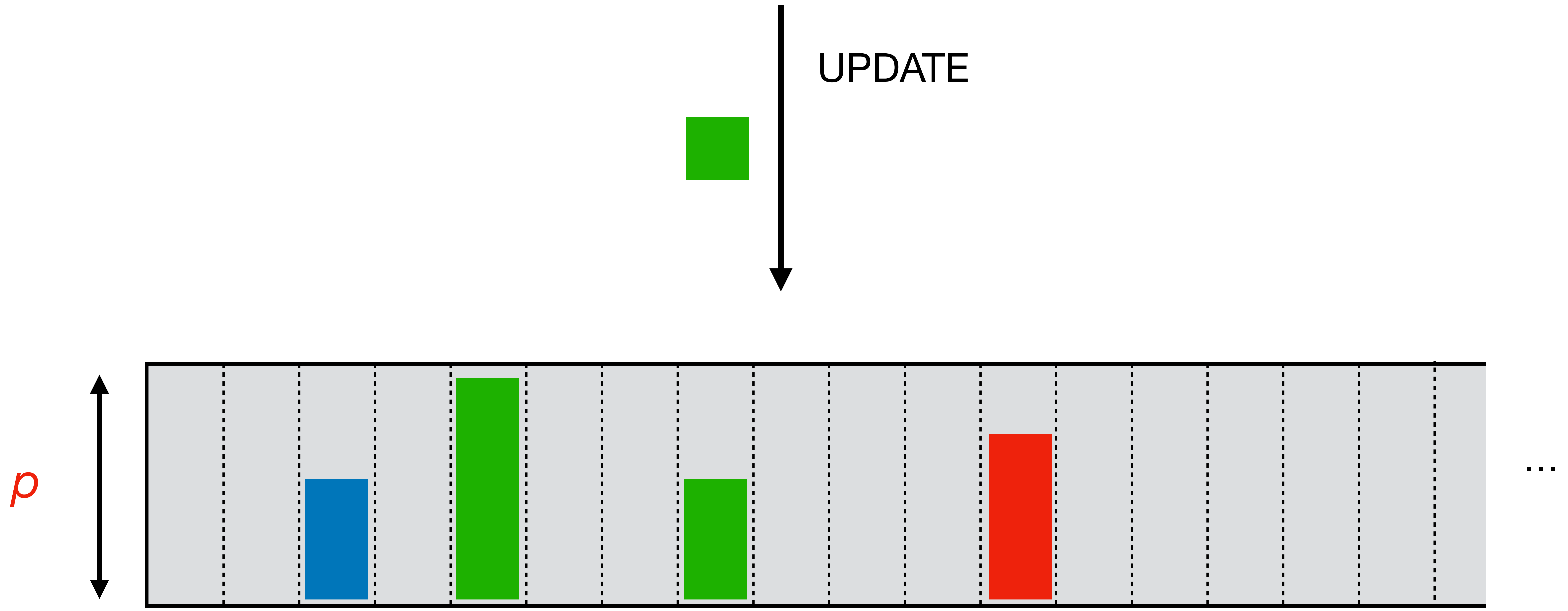
Theorem (BBFMR C'21):

For **secure** SSE, can have O(1) in both measures.

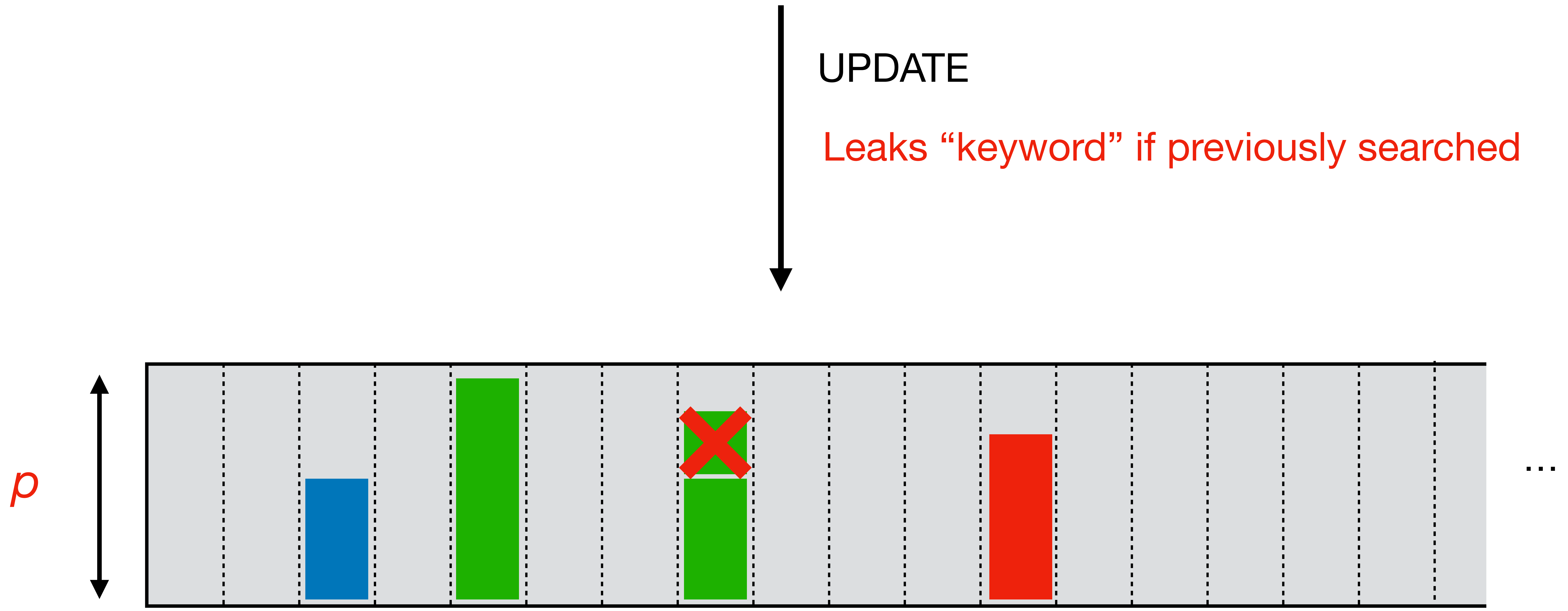
Theorem (*this paper*):

For **forward-secure** SSE, can have $\tilde{O}(\log\log N)$ in both measures.

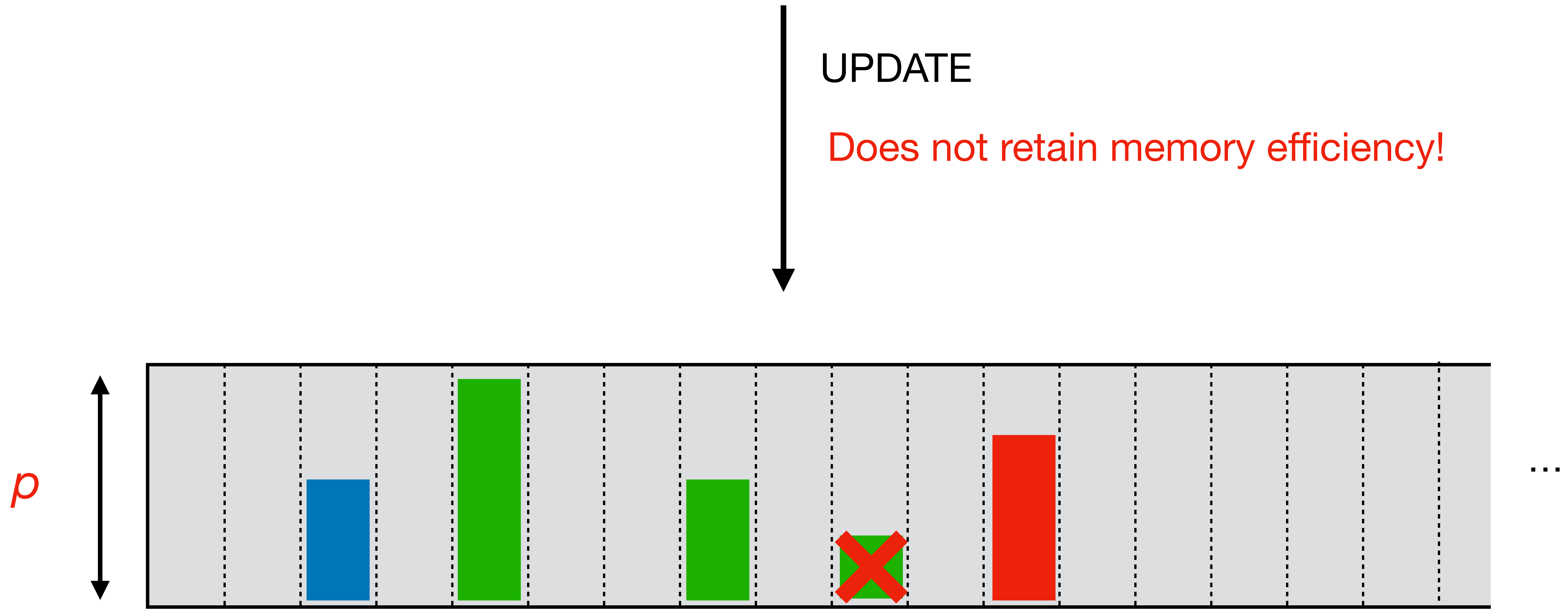
Forward-security vs memory efficiency



Forward-security vs memory efficiency



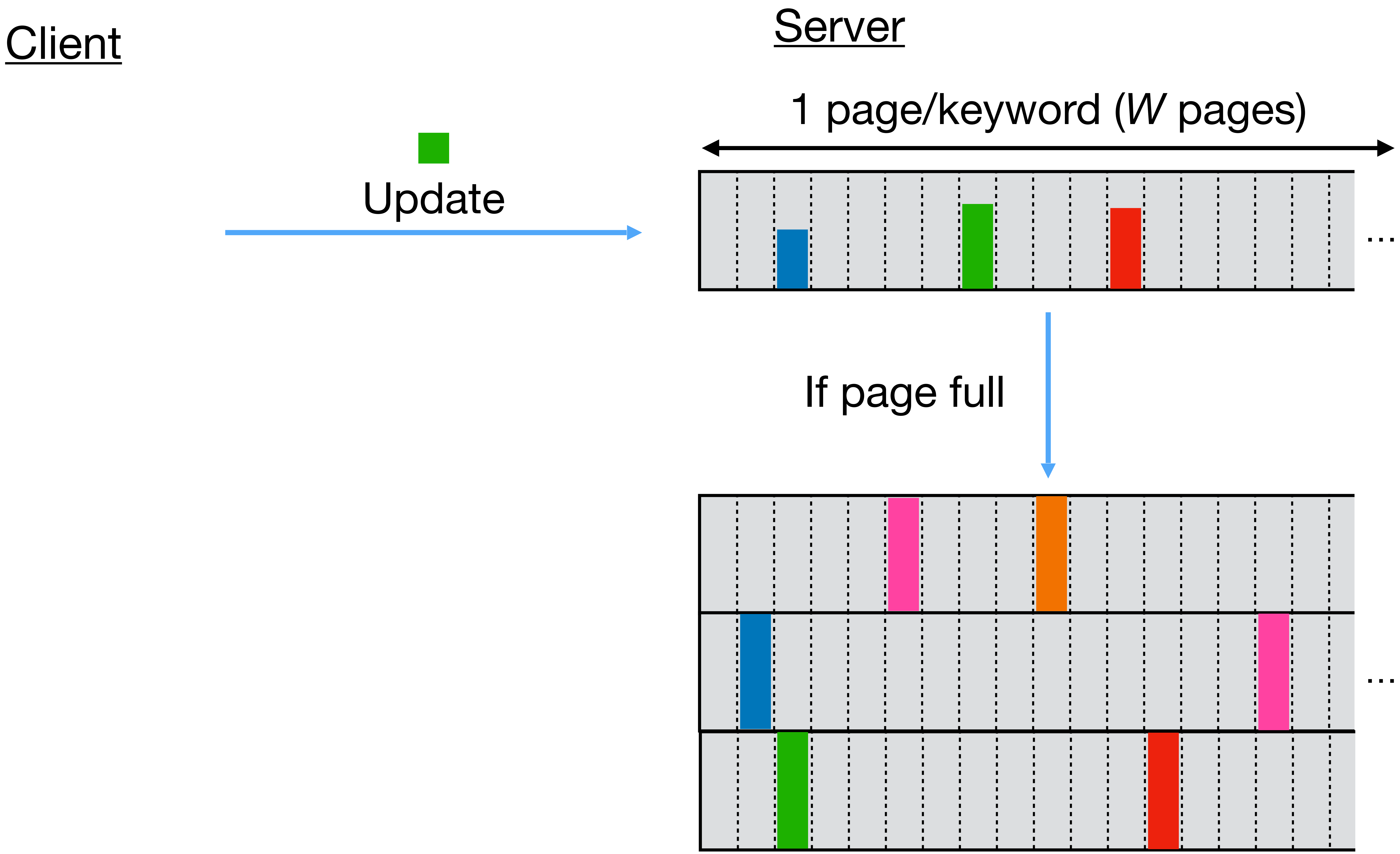
Forward-security vs memory efficiency



Hermes: a solution

...

Basic solution



Full-page SSE scheme

Basic solution

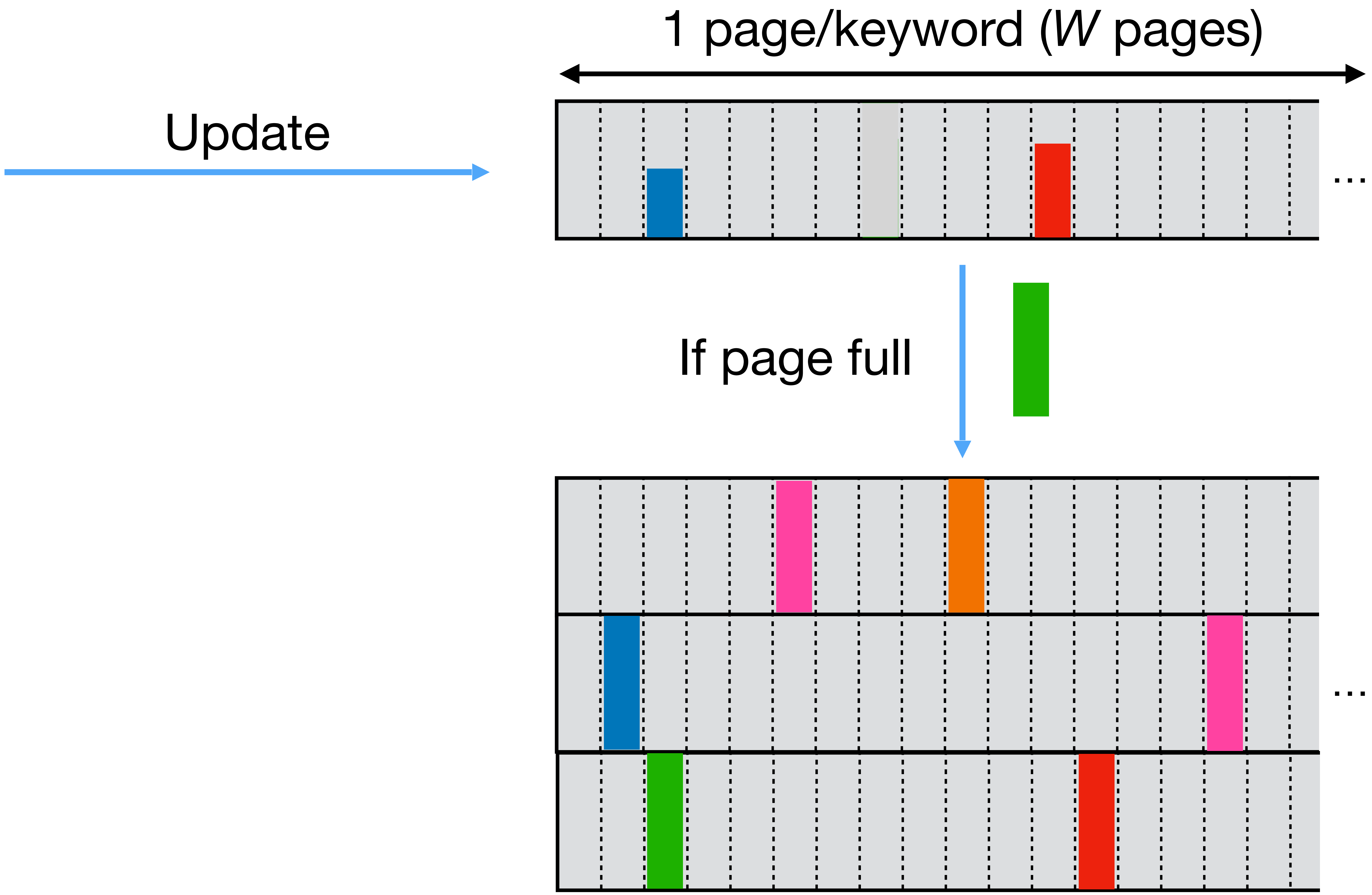
Client

Full-page SSE scheme

Basic solution

Client

Server

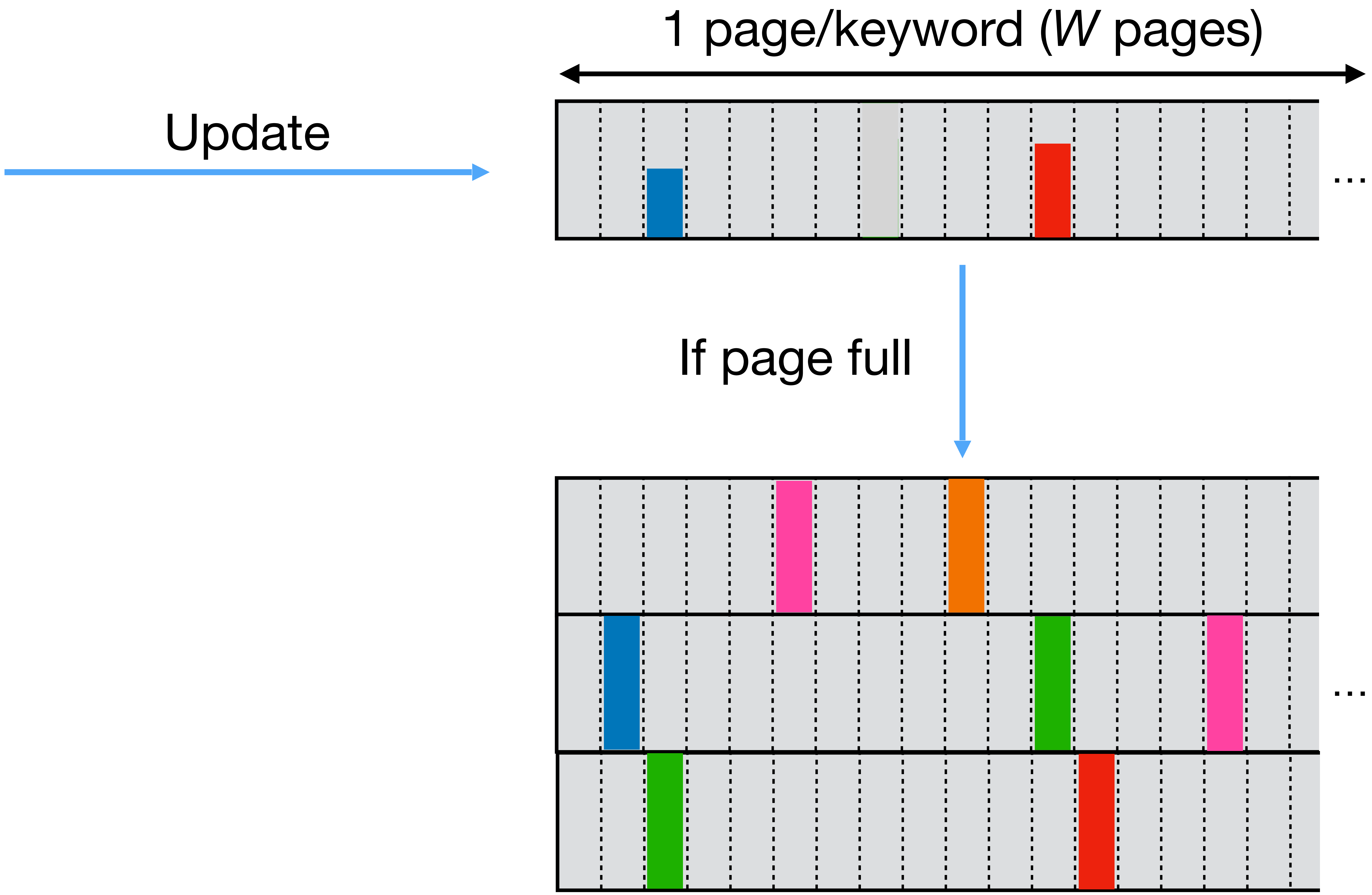


Full-page SSE scheme

Basic solution

Client

Server

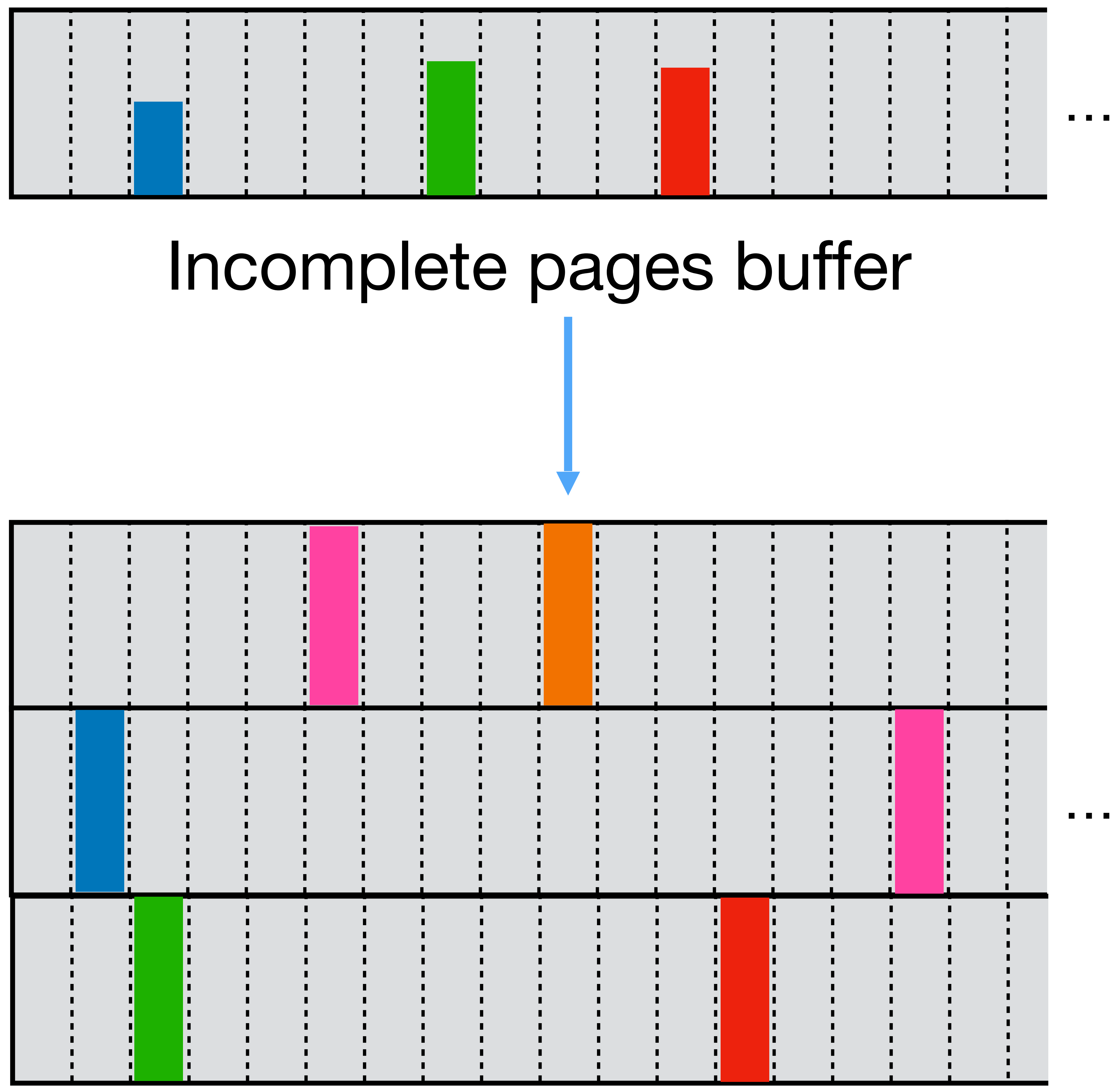


Full-page SSE scheme

IO-DSSE-like approach

Client

Server



Incomplete pages buffer

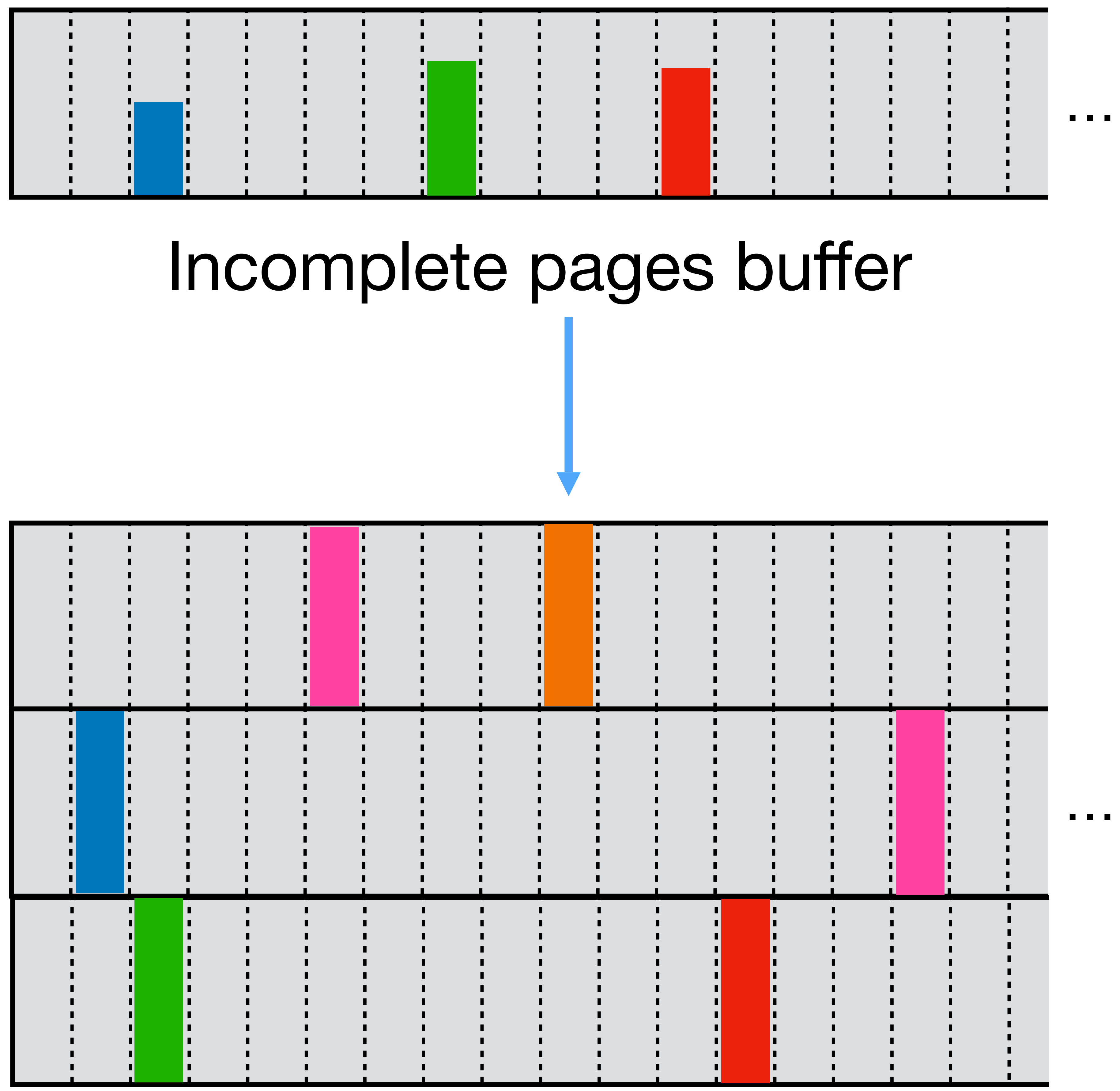
Full-page SSE scheme

Similar to IO-DSSE: Miers & Mohassel, NDSS '17

IO-DSSE-like approach

Client

Server



Full-page SSE scheme

Problems:

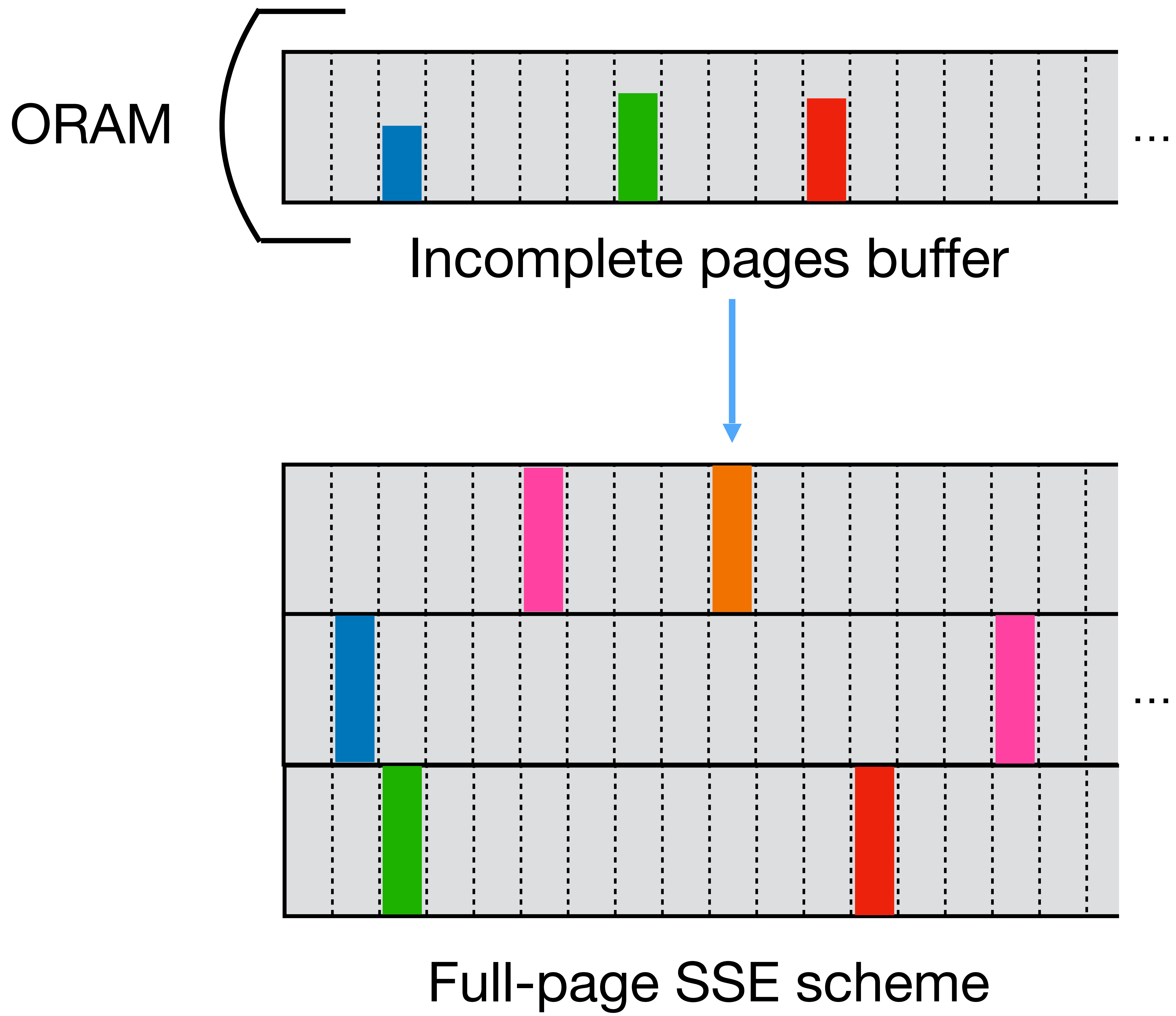
- **Server learns updated keyword,** due to updates in first buffer.
- **Sever learns when a page is full,** due to pushing full pages to SSE.

Similar to IO-DSSE: Miers & Mohassel, NDSS '17

IO-DSSE-like approach

Client

Server



Problems:

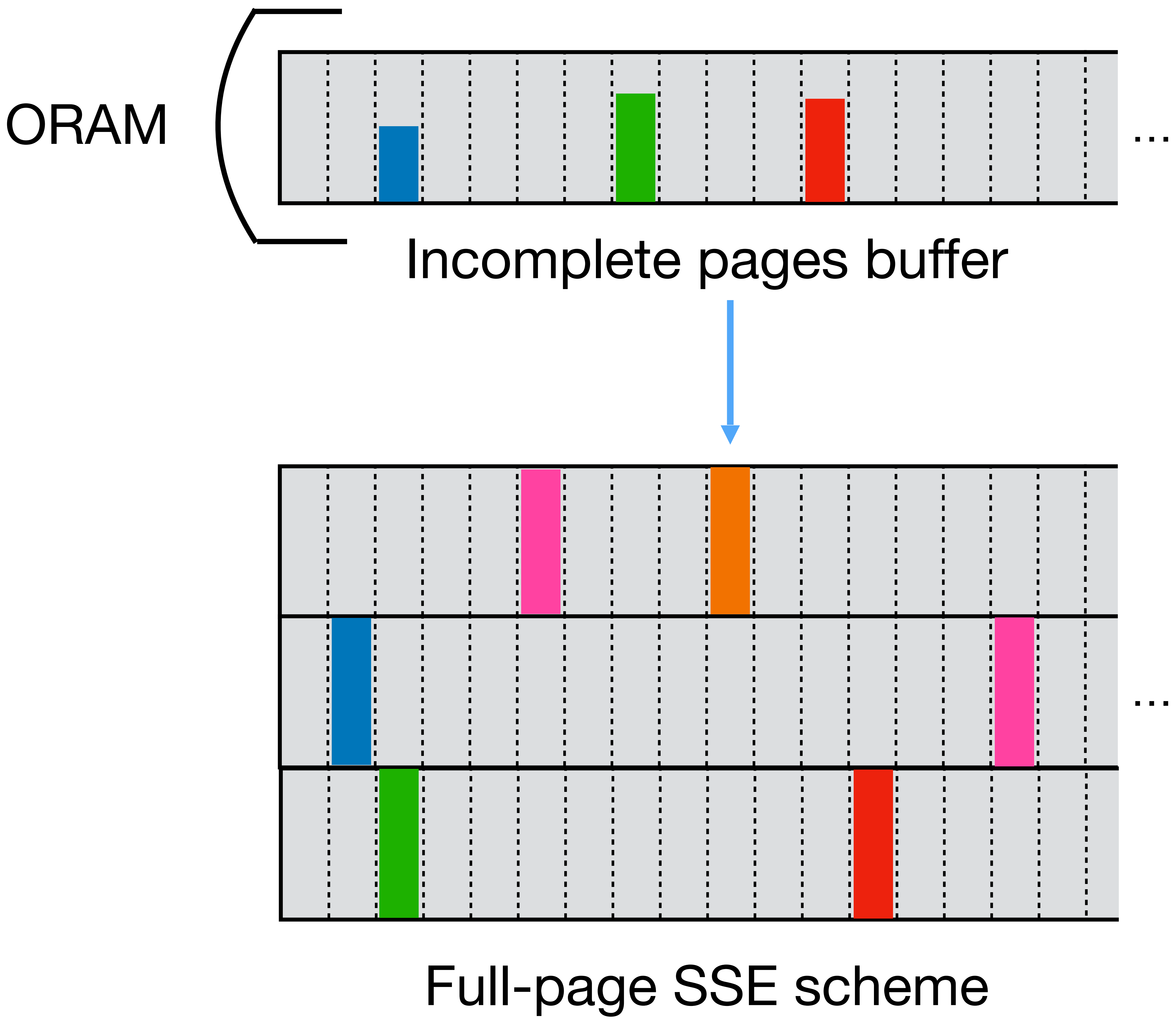
- **Server learns updated keyword,** due to updates in first buffer.
- **Sever learns when a page is full,** due to pushing full pages to SSE.

Similar to IO-DSSE: Miers & Mohassel, NDSS '17

IO-DSSE-like approach

Client

Server



Problems:

- ~~**Server learns updated keyword,** due to updates in first buffer.~~

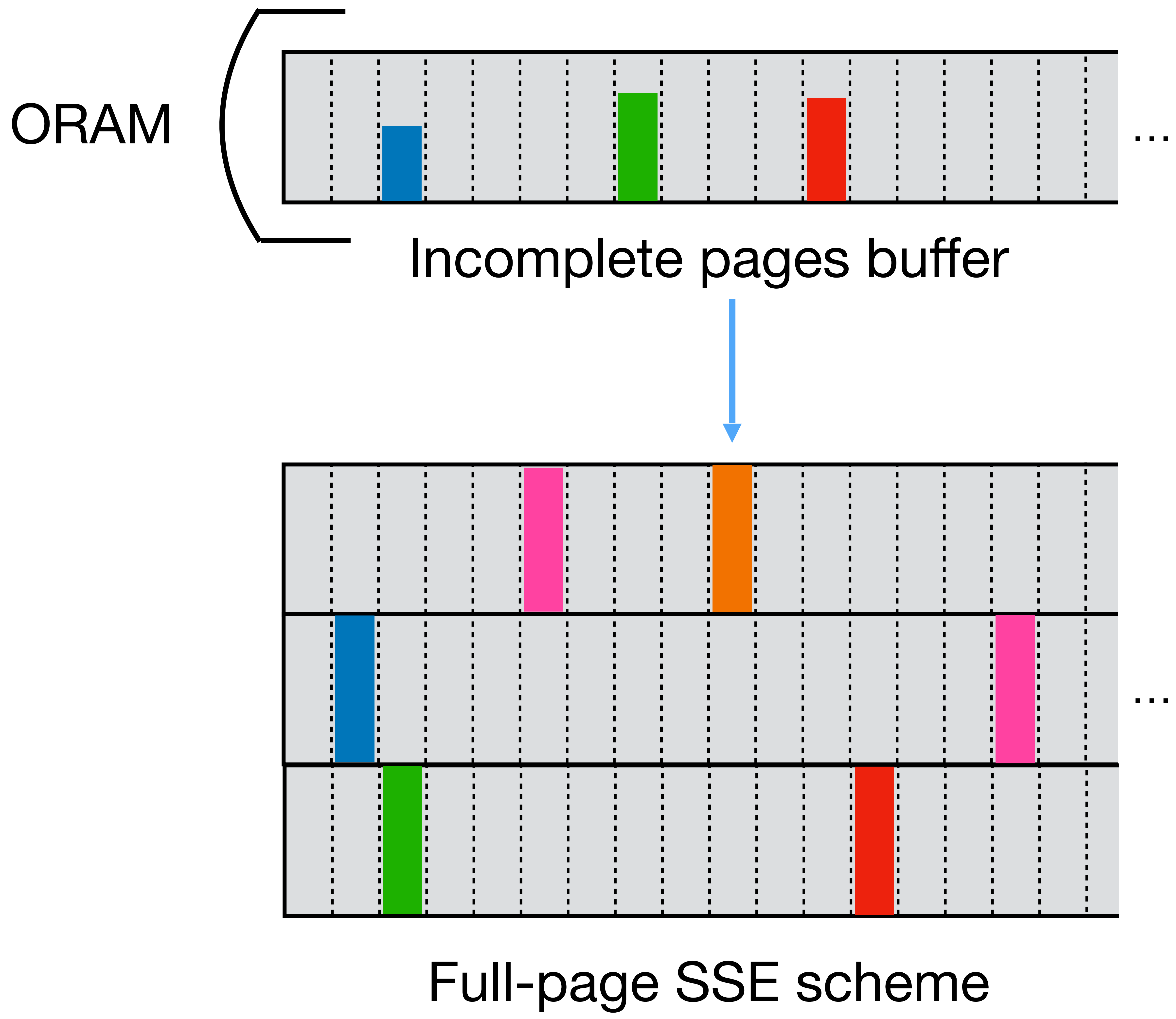
- **Sever learns when a page is full,** due to pushing full pages to SSE.

Similar to IO-DSSE: Miers & Mohassel, NDSS '17

IO-DSSE-like approach

Client

Server



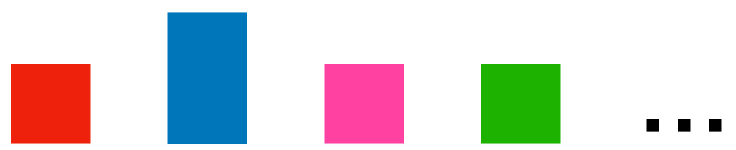
Problems:

- ~~**Server learns updated keyword,** due to updates in first buffer.~~
- **ORAM overhead** Ω(polylog *W*).
- **Sever learns when a page is full,** due to pushing full pages to SSE.

Similar to IO-DSSE: Miers & Mohassel, NDSS '17

Idea #1: client-side buffering

Client

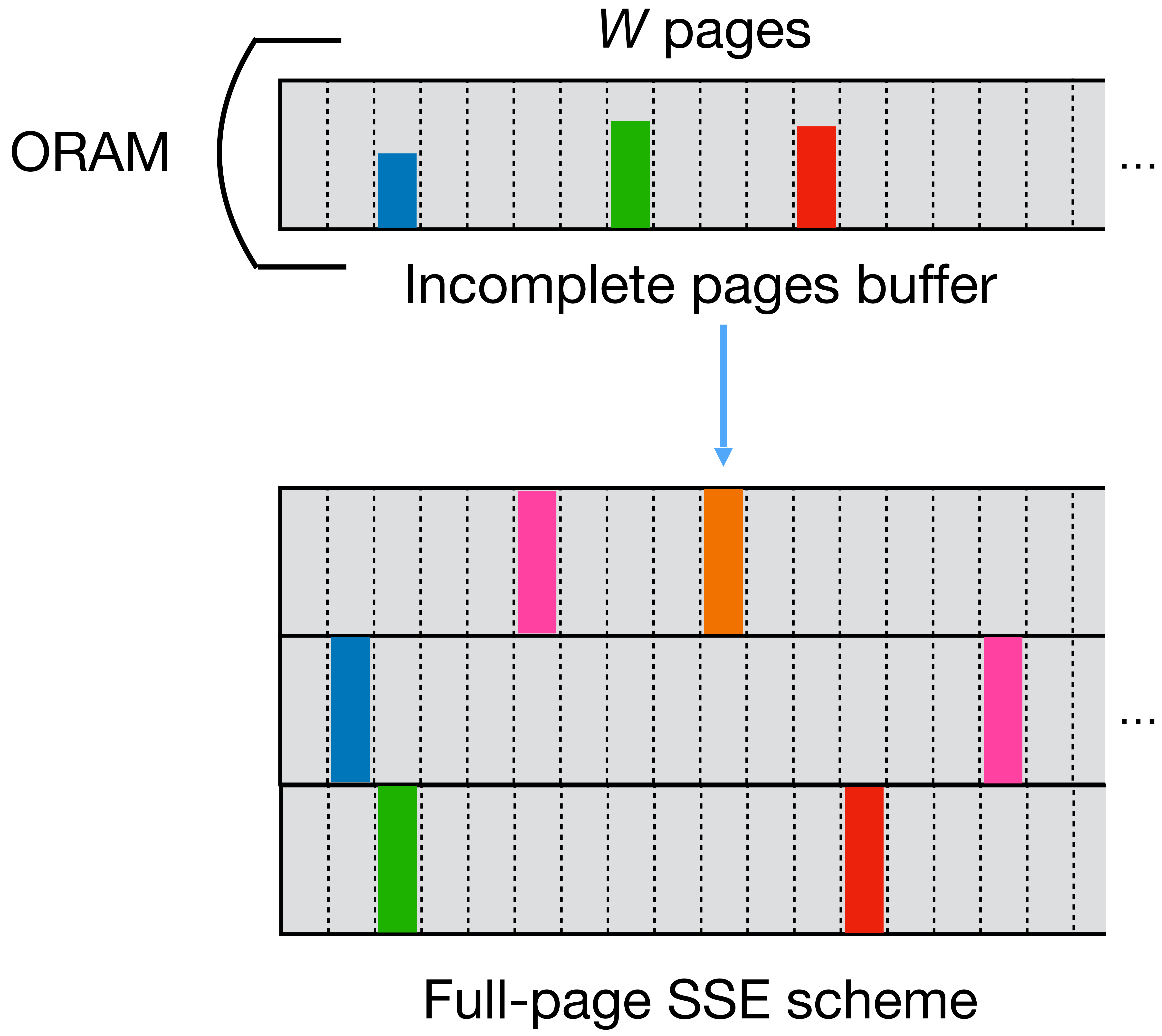


O(*W*) client storage*

Problems:

- ~~**Server learns updated keyword,** due to updates in first buffer.~~
- **ORAM overhead** Ω(polylog *W*).
- **Sever learns when a page is full,** due to pushing full pages to SSE.

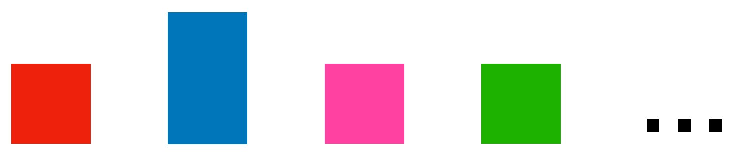
Server



*optimal for forward-secure SSE: [BF19]

Idea #1: client-side buffering

Client



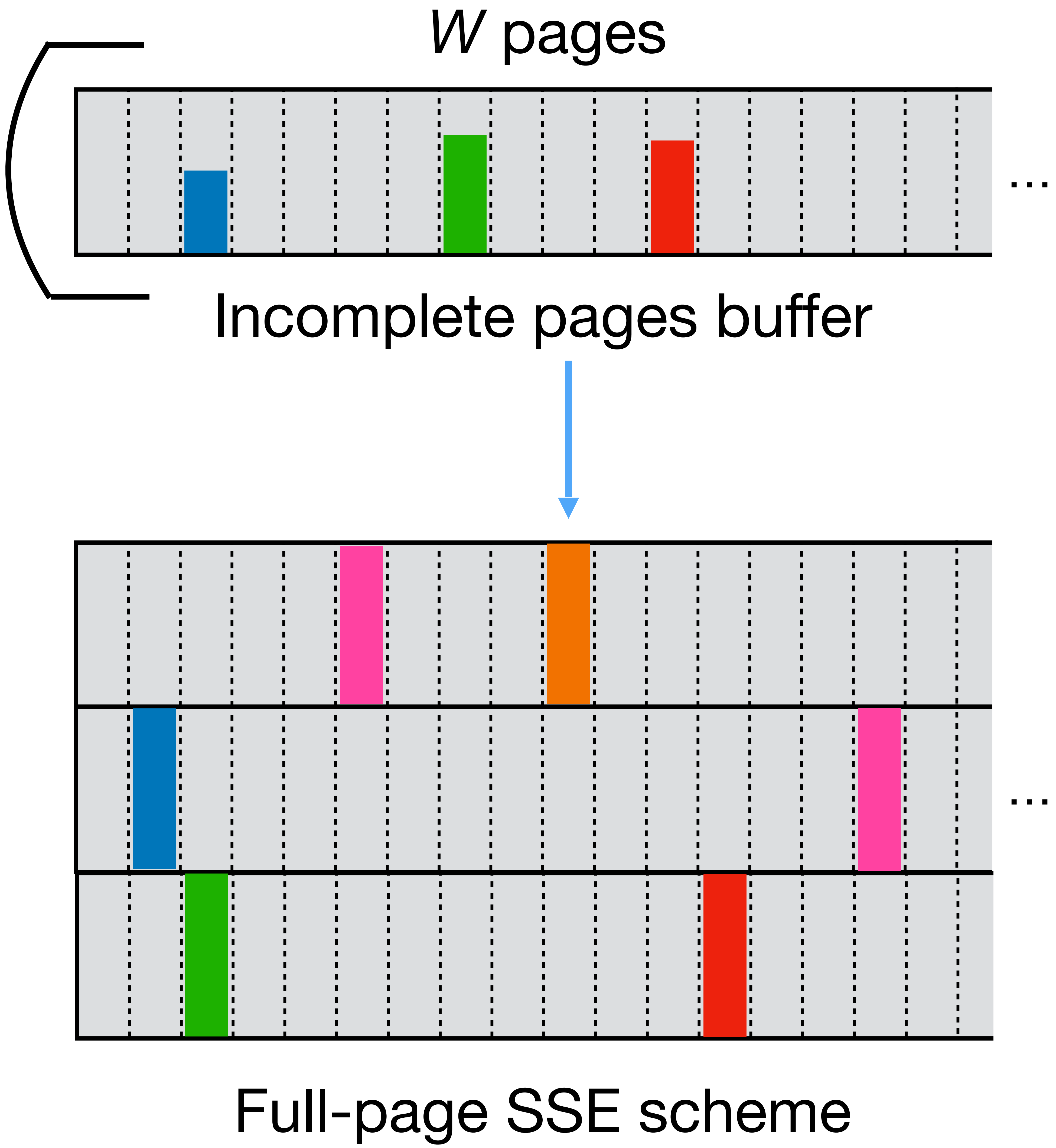
O(*W*) client storage*

Problems:

- ~~**Server learns updated keyword,** due to updates in first buffer.~~
- **ORAM overhead** Ω(polylog *W*).
- **Sever learns when a page is full,** due to pushing full pages to SSE.

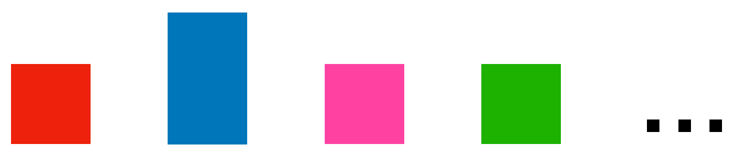
*optimal for forward-secure SSE: [BF19]

Server



Idea #1: client-side buffering

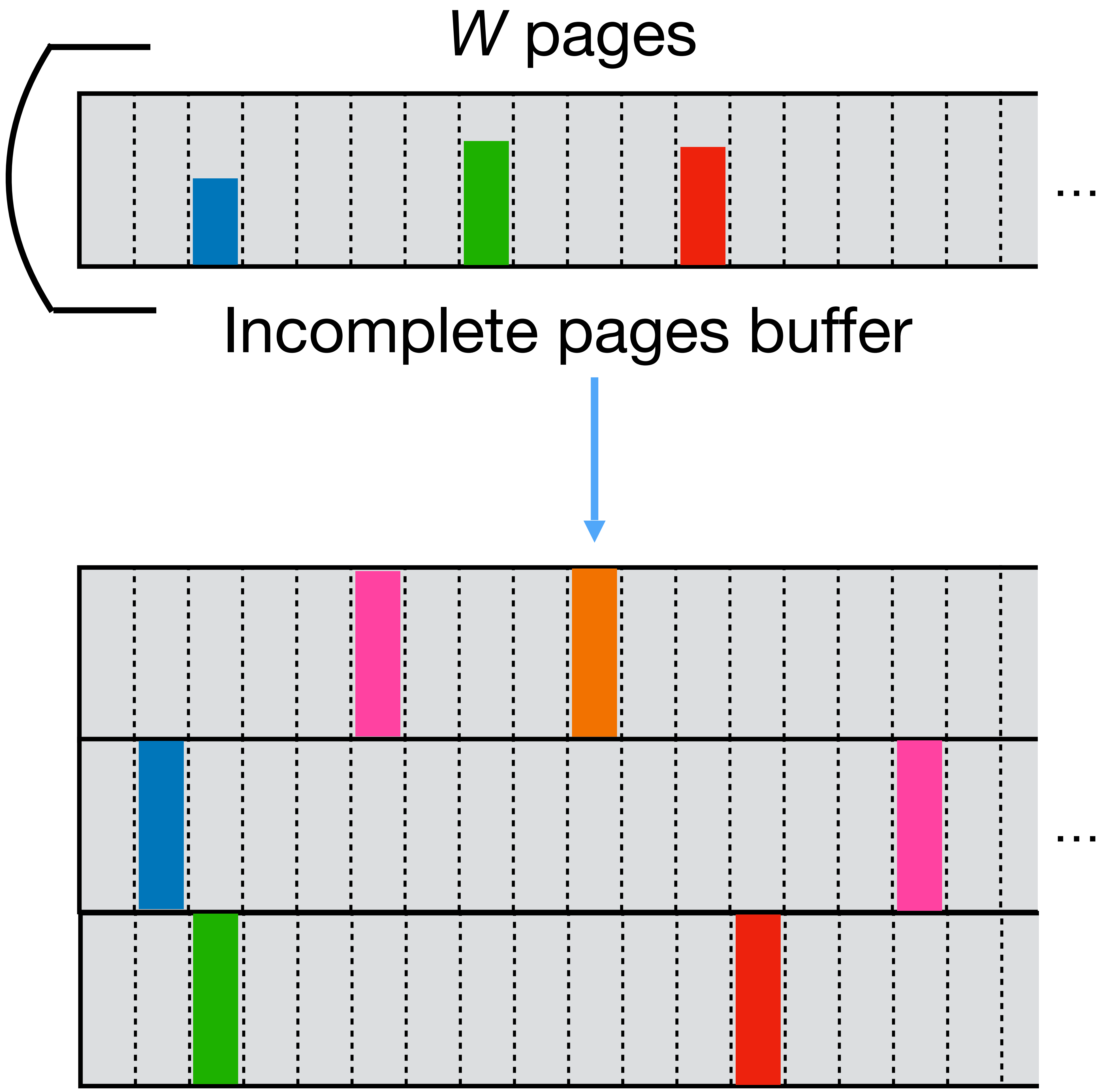
<u>Client</u>



O(W) client storage*

<u>Server</u>

Pushes on fixed schedule



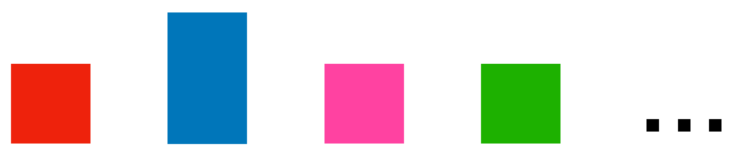
Problems:

- ~~**Server learns updated keyword,** due to updates in first buffer.~~
- **ORAM overhead** Ω(polylog W).
- **Sever learns when a page is full,** due to pushing full pages to SSE.

*optimal for forward-secure SSE: [BF19]

Idea #1: client-side buffering

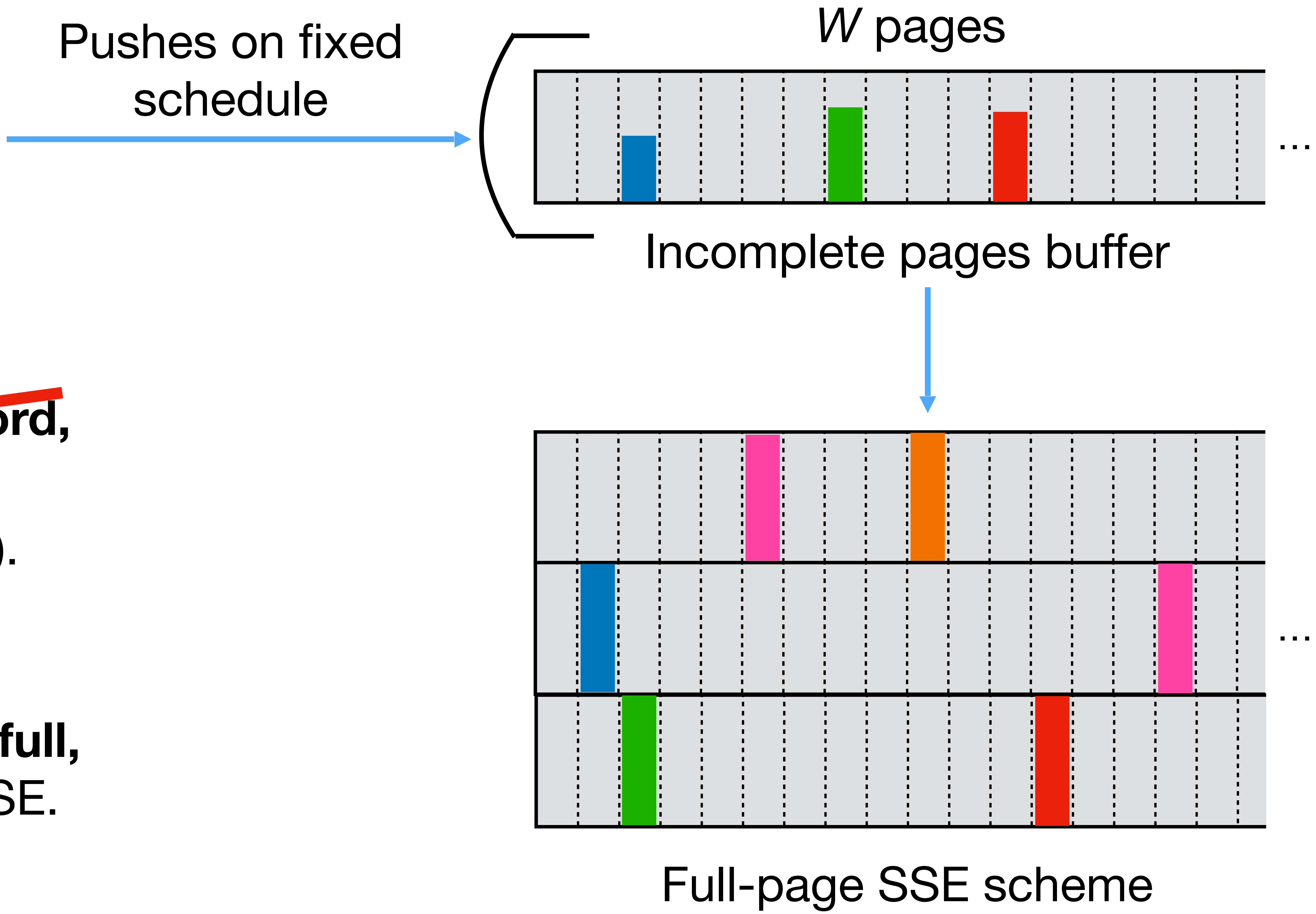
<u>Client</u>



O(W) client storage*

+ Deamortization

<u>Server</u>



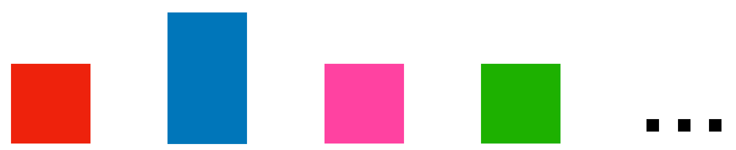
Problems:

- ~~**Server learns updated keyword,** due to updates in first buffer.~~
- **ORAM overhead** Ω(polylog W).
- **Sever learns when a page is full,** due to pushing full pages to SSE.

*optimal for forward-secure SSE: [BF19]

Idea #1: client-side buffering

<u>Client</u>

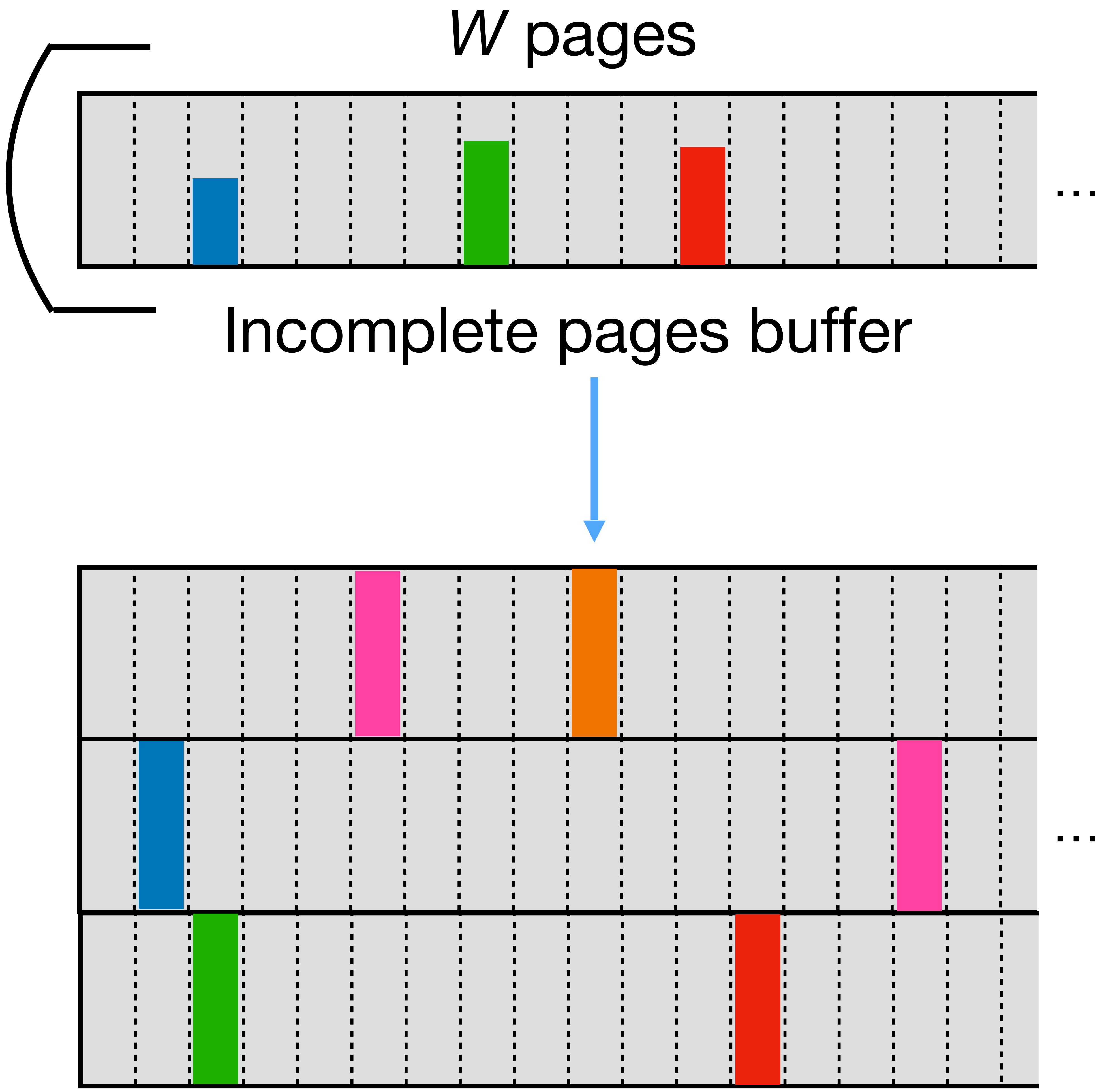


O(W) client storage*

+ Deamortization

Pushes on fixed schedule

<u>Server</u>



Full-page SSE scheme

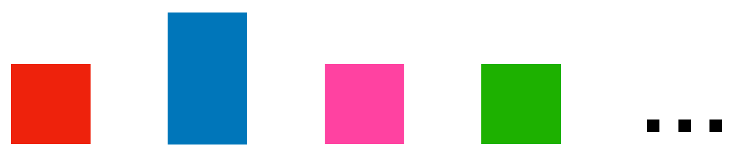
Problems:

- ~~**Server learns updated keyword,** due to updates in first buffer.~~
- ~~**ORAM overhead** Ω(polylog W).~~
- **Sever learns when a page is full,** due to pushing full pages to SSE.

*optimal for forward-secure SSE: [BF19]

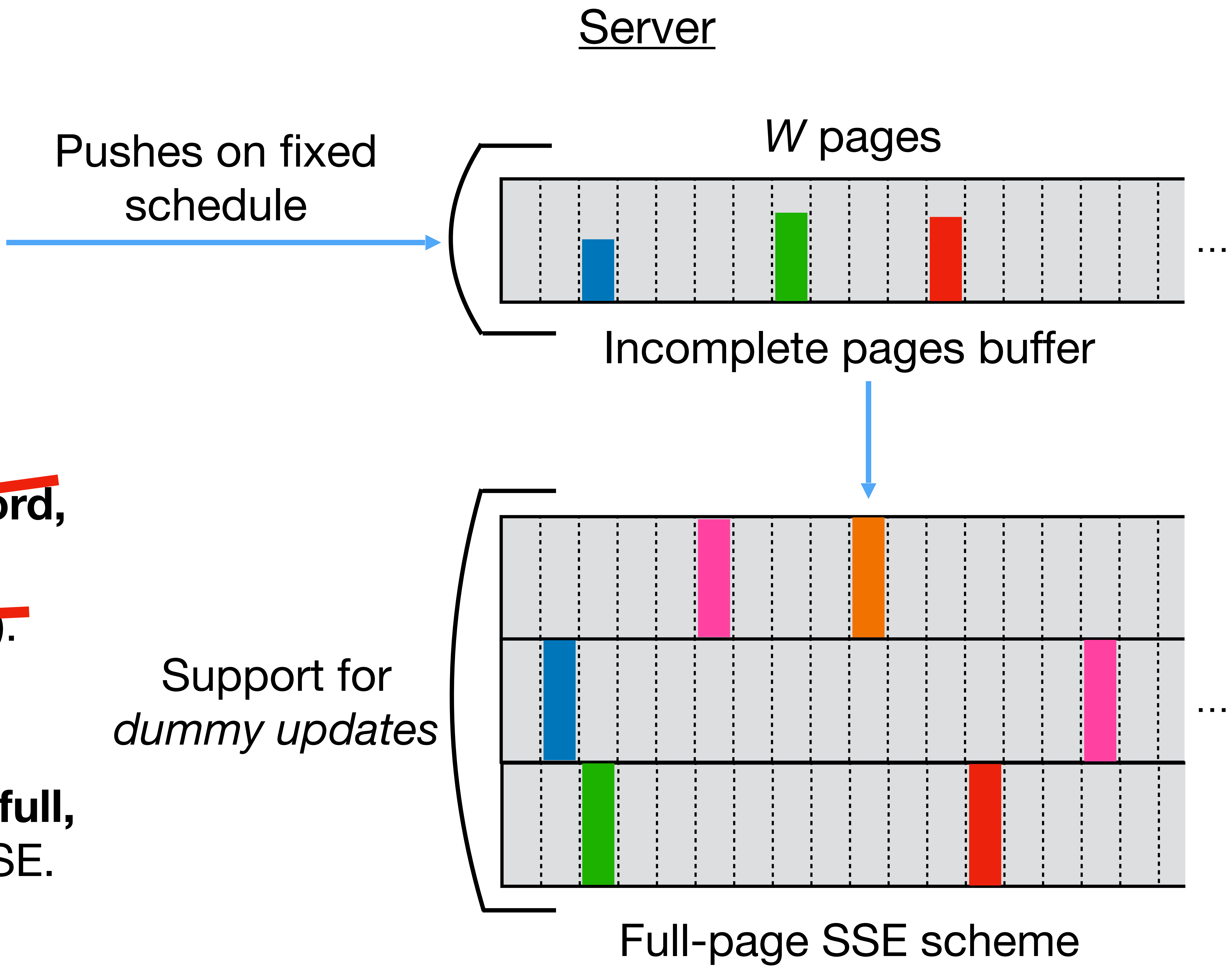
Idea #2: dummy updates

Client



O(W) client storage

+ Deamortization

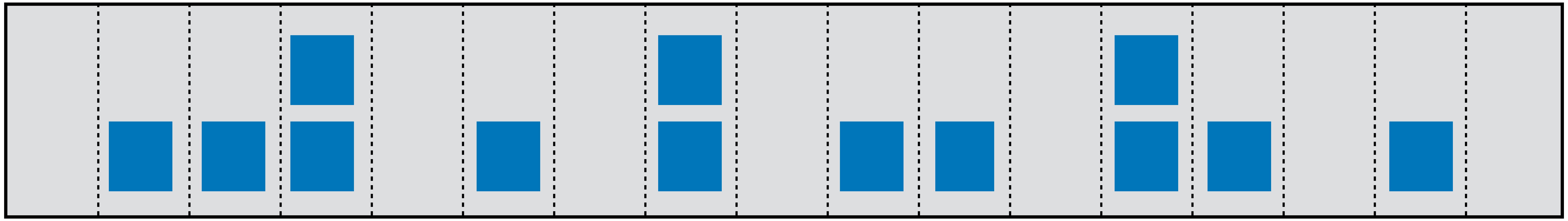


Problems:

- ~~**Server learns updated keyword,** due to updates in first buffer.~~
- ~~**ORAM overhead** Ω(polylog W).~~
- **Sever learns when a page is full,** due to pushing full pages to SSE.

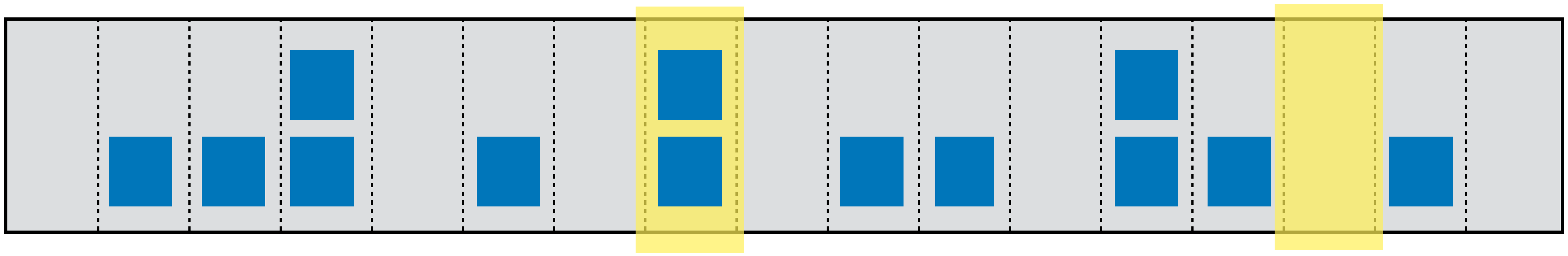
Two-choice allocation

Throw n balls into $m = O(n)$ bins at random



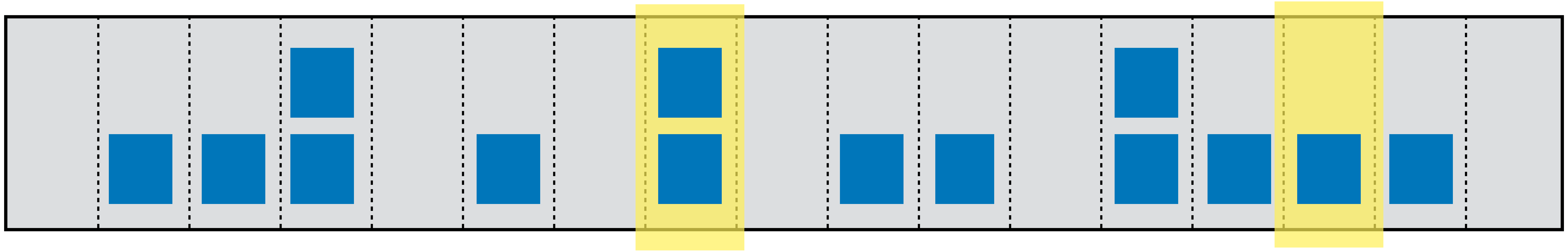
Two-choice allocation

Throw n balls into $m = O(n)$ bins at random



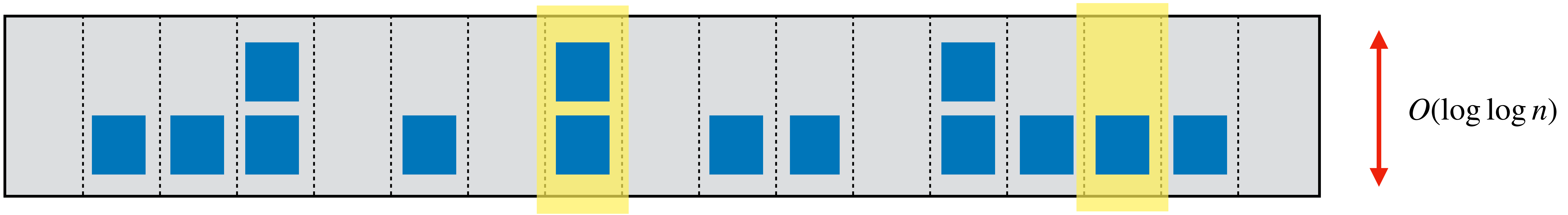
Two-choice allocation

Throw n balls into $m = O(n)$ bins at random



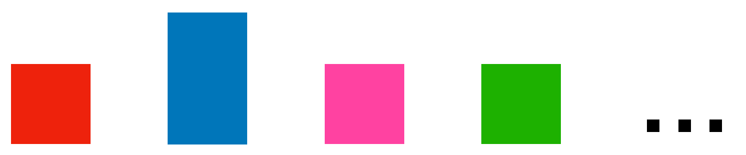
Two-choice allocation

Throw n balls into $m = O(n)$ bins at random



Idea #2: dummy updates

Client

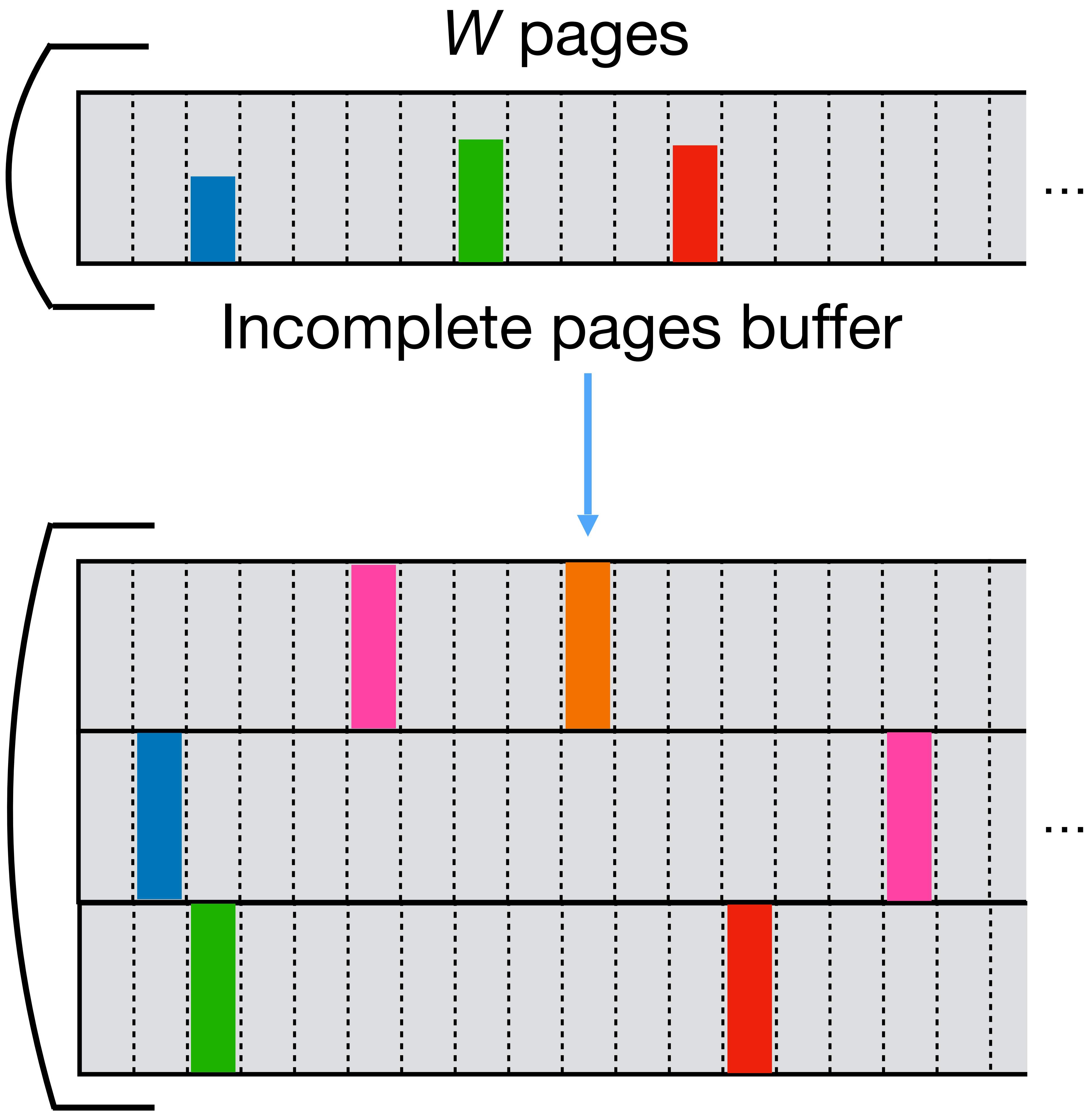


O(*W*) client storage

+ Deamortization

Pushes on fixed schedule

Server

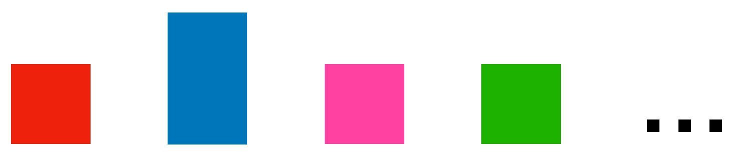


Problems:

- ~~**Server learns updated keyword,** due to updates in first buffer.~~
- ~~**ORAM overhead** Ω(polylog *W*).~~
- ~~**Sever learns when a page is full,** due to pushing full pages to SSE.~~
- **Dummy overhead** Õ(loglog *N*).

Idea #2: dummy updates

Client

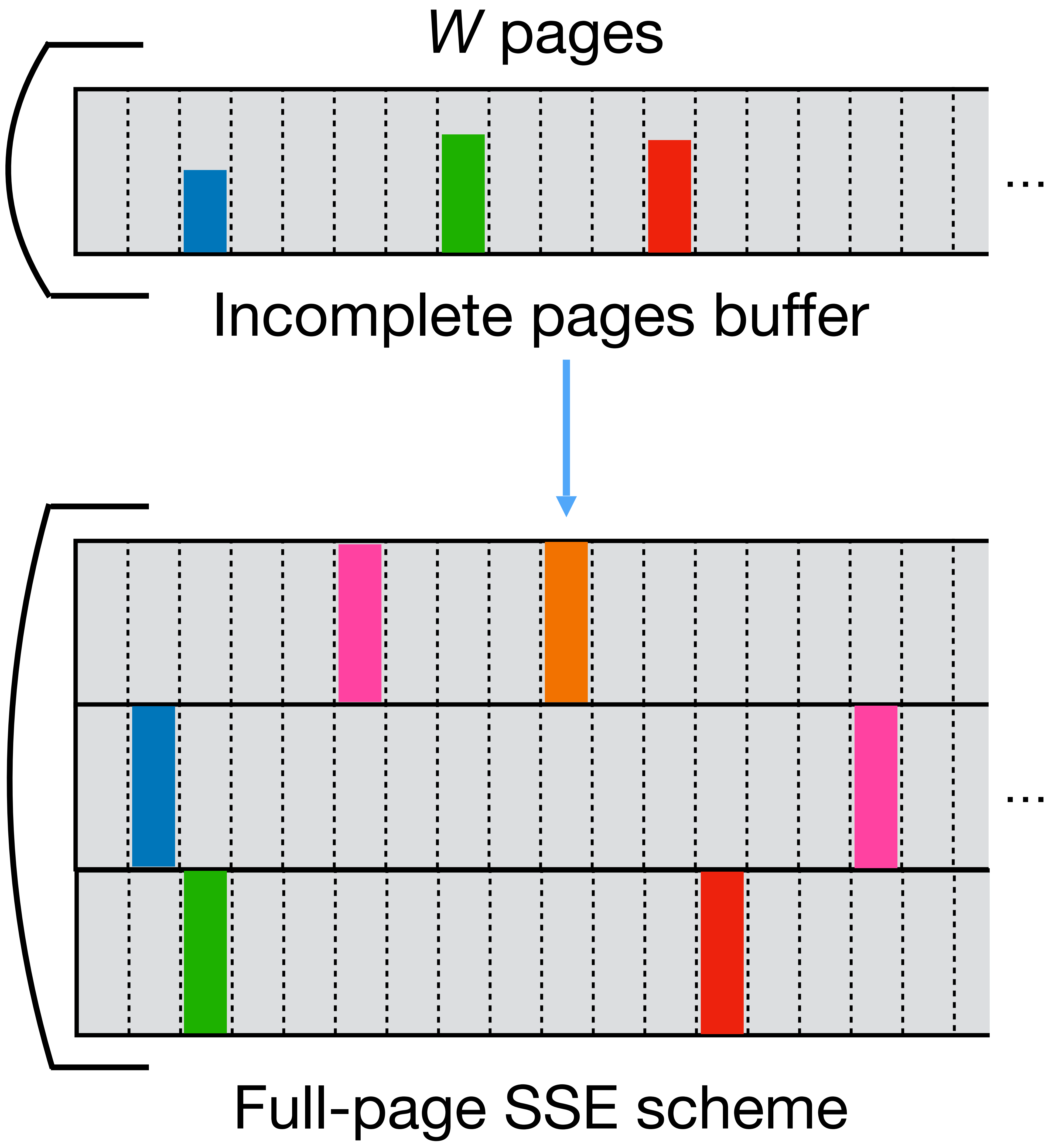


O(*W*) client storage

+ Deamortization

Server

Pushes on fixed schedule



Support for *dummy updates*

Problems:

- **Dummy overhead Õ(loglog *N*).** Reduces to overhead of SSE scheme w/ dummy updates.
- **Modular:** can use any SSE with dummy updates.
- **Two regimes:** this slide assumes *pW* = O(*N*). Other scheme if pW>N.

Conclusion

Takeaways

- Page efficiency circumvents **two** impossibility results for memory-efficient SSE:
  - [CT15] no optimal memory-efficient secure SSE.
  - [B16] no sublogarithmic memory-efficient forward-secure SSE.

  ⇒ *don't have to sacrifice forward security to be memory-efficient.*

- New way to build forward security using buffer+deamortization.

Open questions:

- Can we *prove* memory efficient lower bounds for forward-secure SSE?
- Make this more practical.

Takeaways

- Page efficiency circumvents **two** impossibility results for memory-efficient SSE:
  - [CT15] no optimal memory-efficient secure SSE.
  - [B16] no sublogarithmic memory-efficient forward-secure SSE.

  ⇒ *don't have to sacrifice forward security to be memory-efficient.*

- New way to build forward security using buffer+deamortization.

Open questions:

- Can we *prove* memory efficient lower bounds for forward-secure SSE?
- Make this more practical.

谢谢!