Searchable Encryption and Memory Efficiency

Brice Minaud

Based on joint works with Léonard Assouline, Raphael Bost, Angèle Bossuat, Pierre-Alain Fouque, Charalampos Papamanthou, Michael Reichle

ESSA3 workshop, Bertinoro, 2023

Memory efficiency

"Memory efficiency" = "Locality" ∪ "Page efficiency"

Cash & Tessaro, Eurocrypt '14

Asharov, Naor, Segev, Shahaf, STOC '16

Demertzis, Papamanthou, Sigmod '17

Miers, Mohassel, NDSS '17

Demertzis, Papadopoulos, Papamanthou, Crypto '18

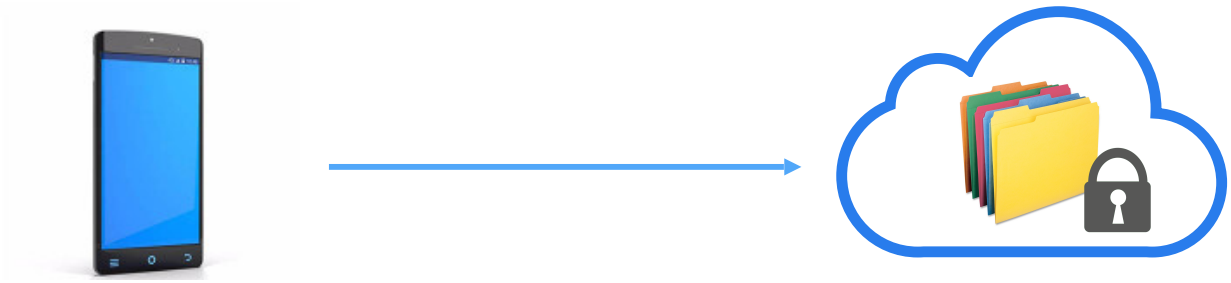
Asharov, Segev, Shahaf, Crypto '18

Bossuat, Bost, Fouque, Minaud, Reichle, Crypto '21

Minaud, Reichle, Crypto '22

Assouline, Minaud, Eurocrypt '23

A simple scenario



Client

Server

Minimalist requirements:

- Just storing encrypted files. *No search.*
- Willing to reveal query and access pattern.
- Static.

Only goal: insulate leakage between files.

> When fetching file *F*, leakage should only depend on *F*.

→ Should leak nothing about sizes of other files (except total DB size).

Wait, how is this about SSE?

Reverse index:

"*car*" ↦ id_1, id_3

"*duck*" ↦ id_2, id_3, id_6, ...

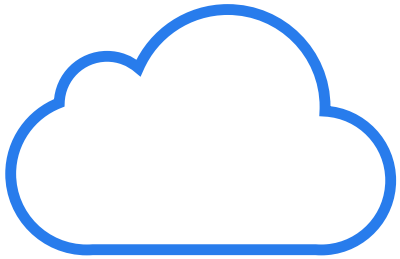
...

Single-keyword SSE

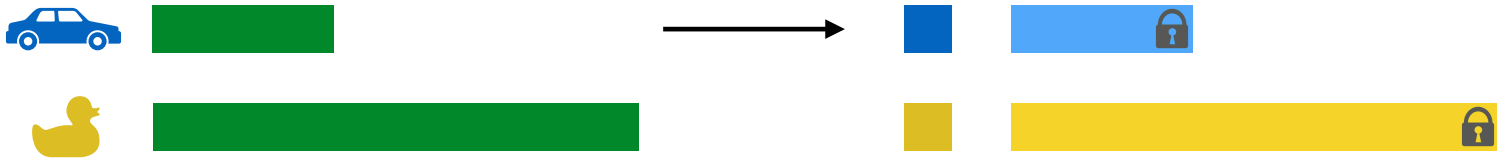
Reverse index:

...

Single-keyword SSE: Setup



Reverse index:



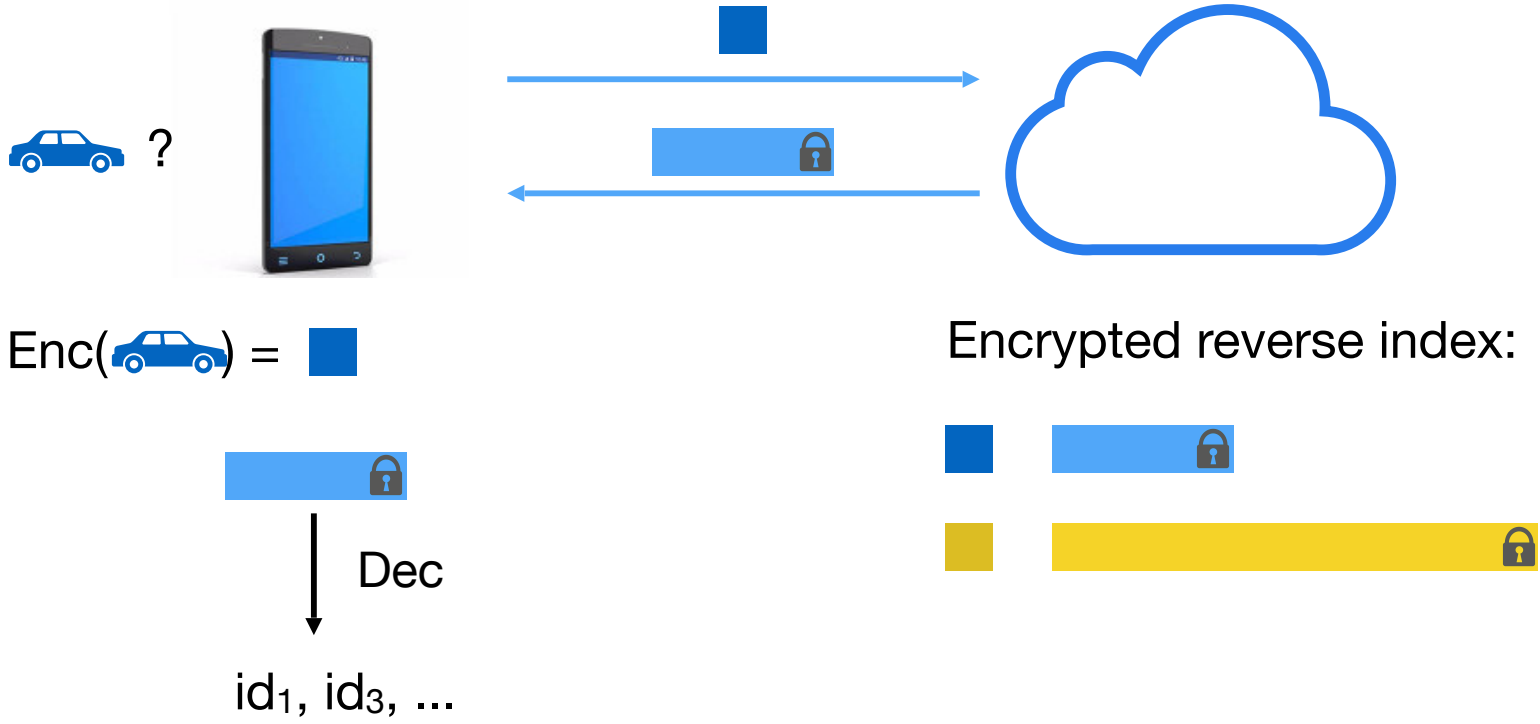
Encrypted reverse index:

Legend:

Enc( ) =

Enc( ) =

Single-keyword SSE: Search

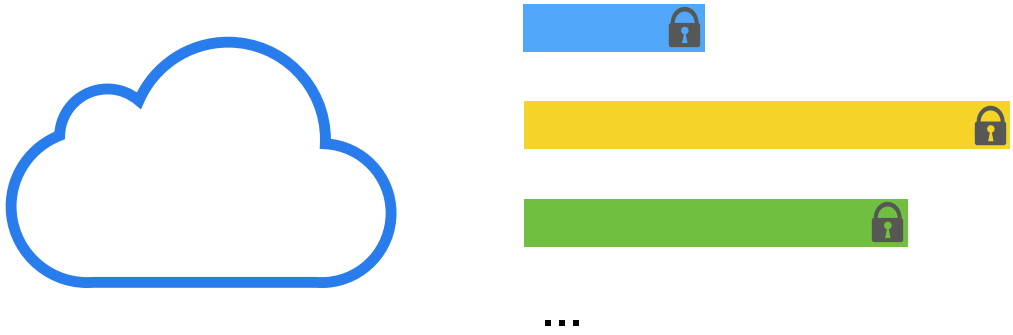


Insulated leakage problem occurs on search index, and on files.

On search index, “file” size = #matching docs.

Naive solutions

Attempt #1: encrypt files **separately**.

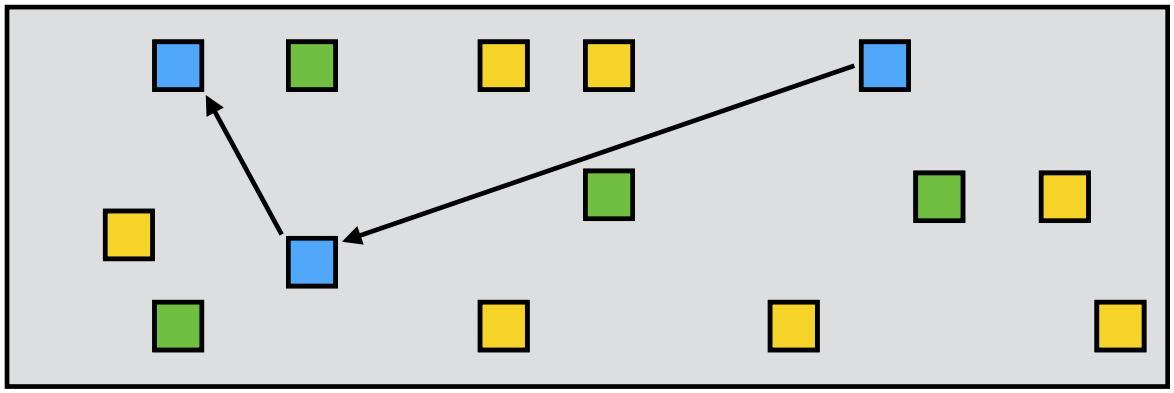


Attempt #2: encrypt files **sequentially**.

Position of one file depends on sizes of other files.

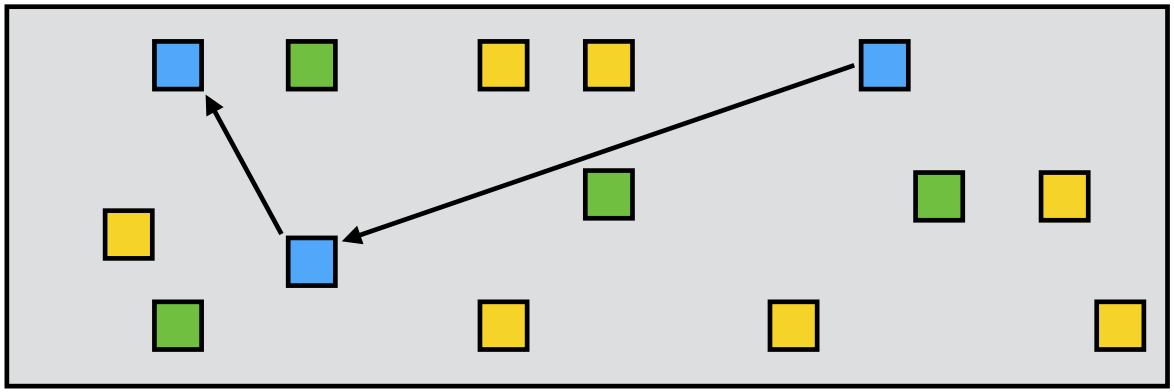
Insulated file storage

Server memory



Insulated file storage, cont'd

Server memory



Security: OK.

File of length $\ell = \ell$ unif. random memory accesses.

Efficiency: Terrible.

Worst-case cost for Hard Disk Drives: reading contiguous memory much cheaper than random locations.

Example : *Σοφος* [B16].

Is the issue inherent?

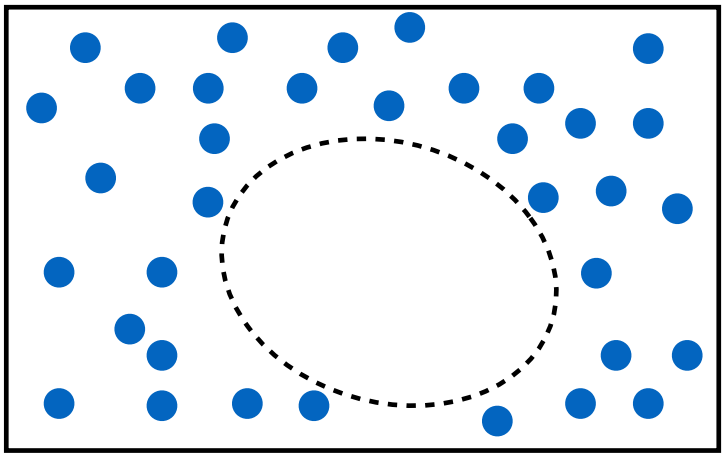
- Memory efficiency asks:

  data position is correlated with content.

- Security asks:

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

In pictures



Formalizing the problem

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.

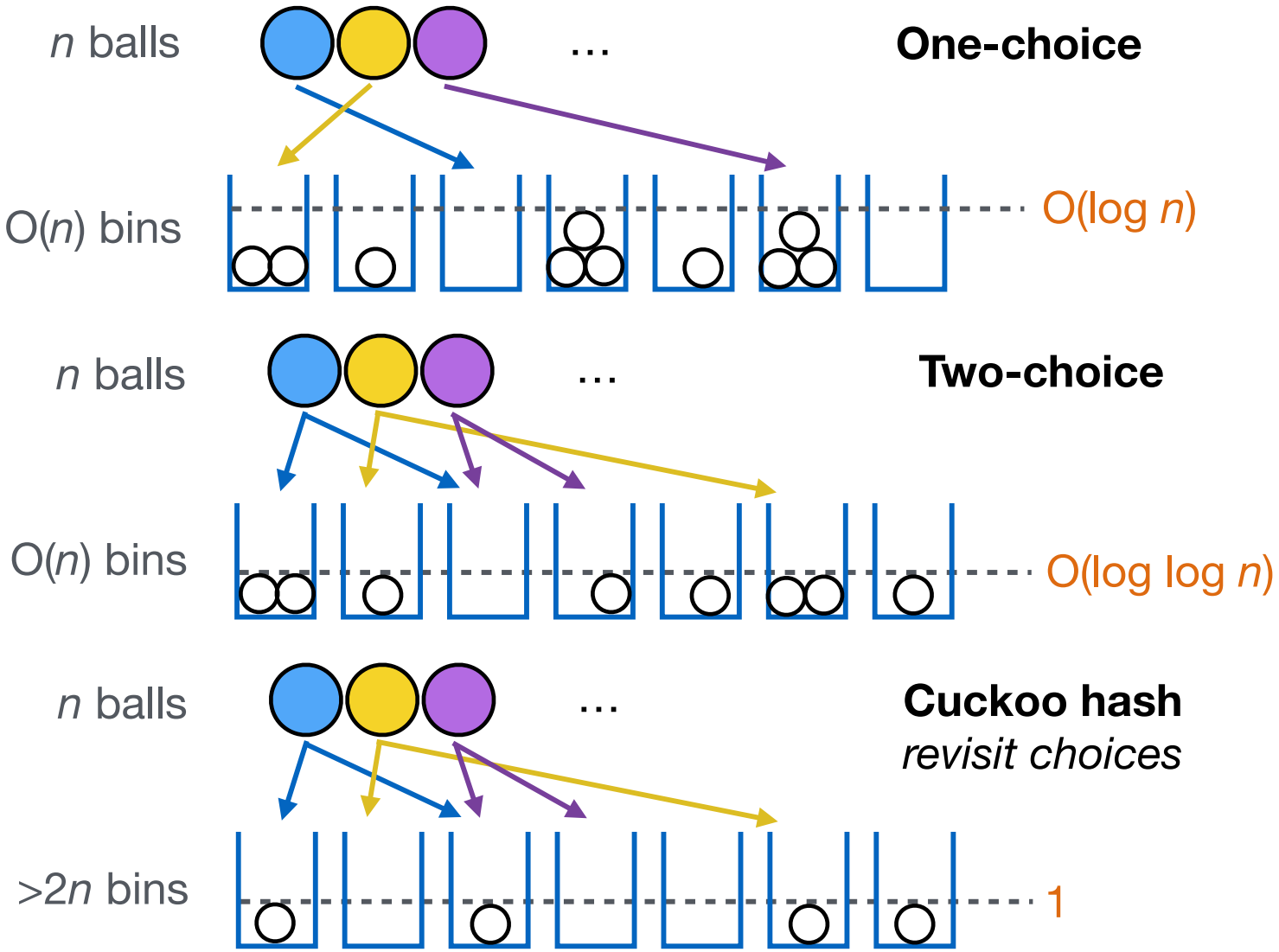
Theorem (Cash & Tessaro EC’15):

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

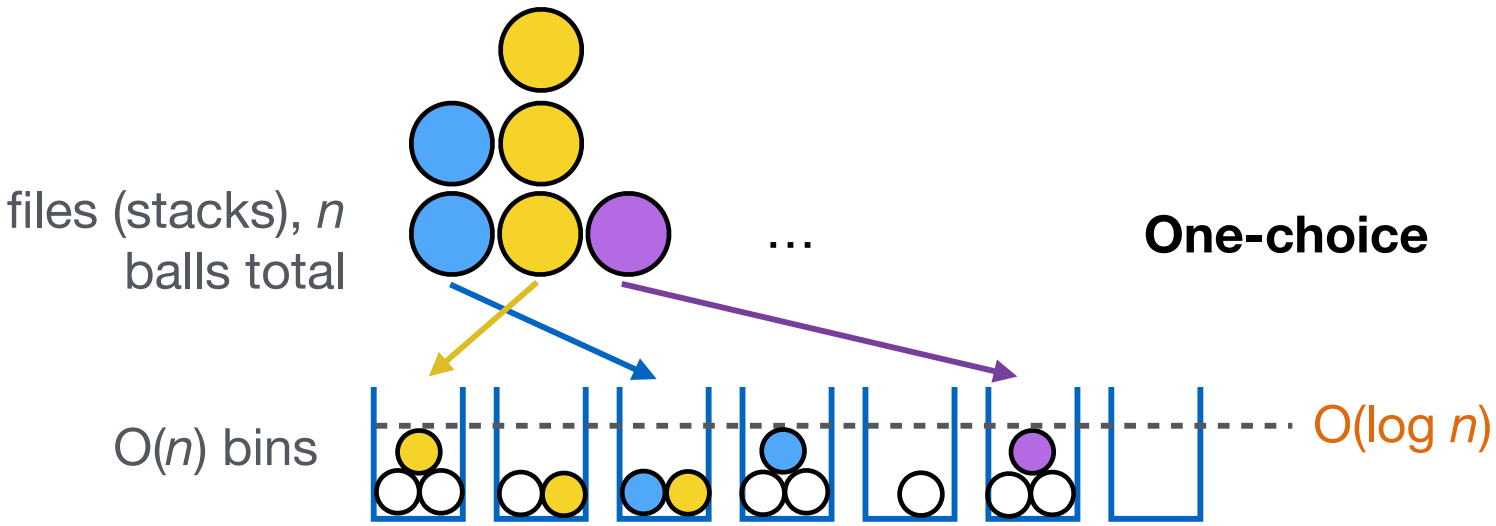
Spawned long line of work.

Locality

Multiple-choice allocation



Asharov et al. construction



Two-choice variant: O(log log *n*)

…assuming largest file size < $n^{1 - 1/\log\log \lambda}$.

Static local SSE

N = size of DB

Scheme	Locality	Storage eff.	Read eff.
[ANSS16] 1C	O(1)	O(1)	$\tilde{O}$(log N)
[ANSS16] 2C	O(1)	O(1)	$\tilde{O}$(log log N)*
[DP17]	L	O(log N/log L)	O(1)

Matching lower bounds in restricted models [ASS18]

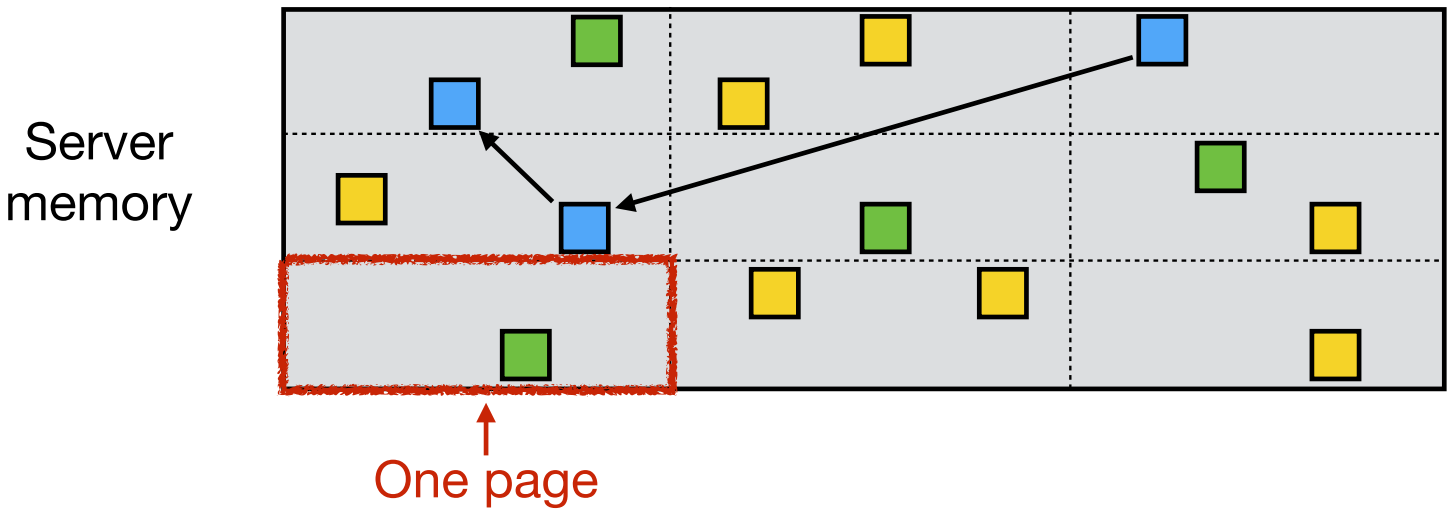
Conjecture: static server memory ⇒ this is impossible:

Locality	Storage eff.	Read eff.
O(1)	O(1)	o(log N)

*under condition: longest list size ≤ $N^{1-1/\log\log N}$

Page efficiency

In pictures



Page-level access enforced by OS and physical memory.

Page efficiency

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].

Page efficiency

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

> Theorem (Cash & Tessaro EC '15):
>
> *Secure* SSE cannot have O(1) in all 3 measures.

Page efficiency + **Storage efficiency**.

Scheme	Storage eff.	Page eff.
[BBFMR21]	O(1)	O(1)

[MR22]: can add dynamism with $\tilde{O}(\log \log n)$ page efficiency.

Side product: **dynamic** scheme matching Asharov et al. **local** construction.

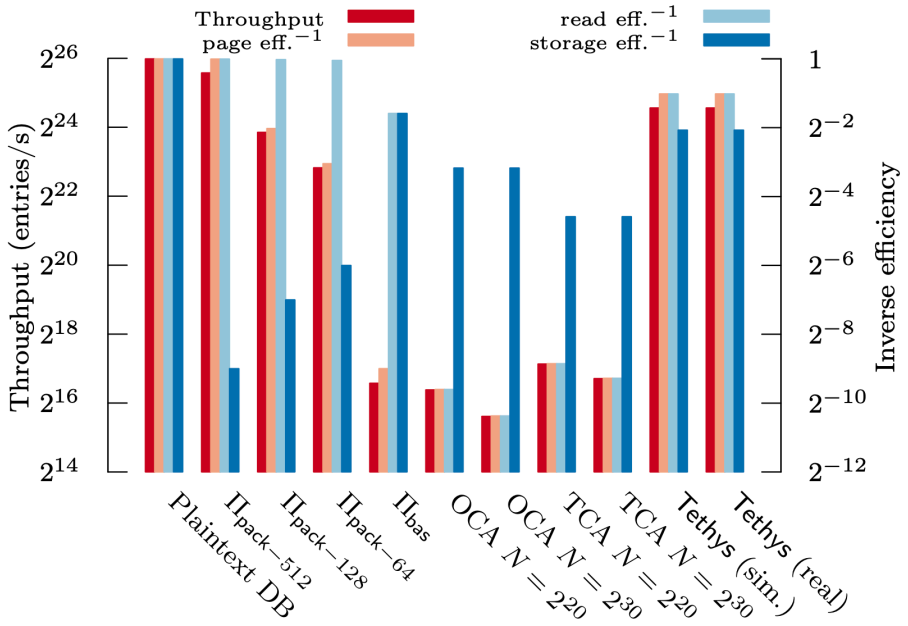
[MR23]: can add forward security "for free" (asymptotically).

Performance evaluation

Theory

Schemes	Client st.	Page eff.	Storage eff.
Π_{bas}	$\mathcal{O}(1)$	$\mathcal{O}(p)$	$\mathcal{O}(1)$
$\Pi_{\text{pack}}, \Pi_{\text{2lev}}$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(p)$
1-Choice	$\mathcal{O}(1)$	$\widetilde{\mathcal{O}}(\log N)$	$\mathcal{O}(1)$
2-Choice	$\mathcal{O}(1)$	$\widetilde{\mathcal{O}}(\log \log N)$	$\mathcal{O}(1)$
Tethys	$\mathcal{O}(p \log \lambda)$	3	$3+\varepsilon$
Pluto	$\mathcal{O}(p \log \lambda)$	3	$3+\varepsilon$
Nilus$_t$	$\mathcal{O}(p \log \lambda)$	$2t+1$	$1+(2/e)^{t-1}$

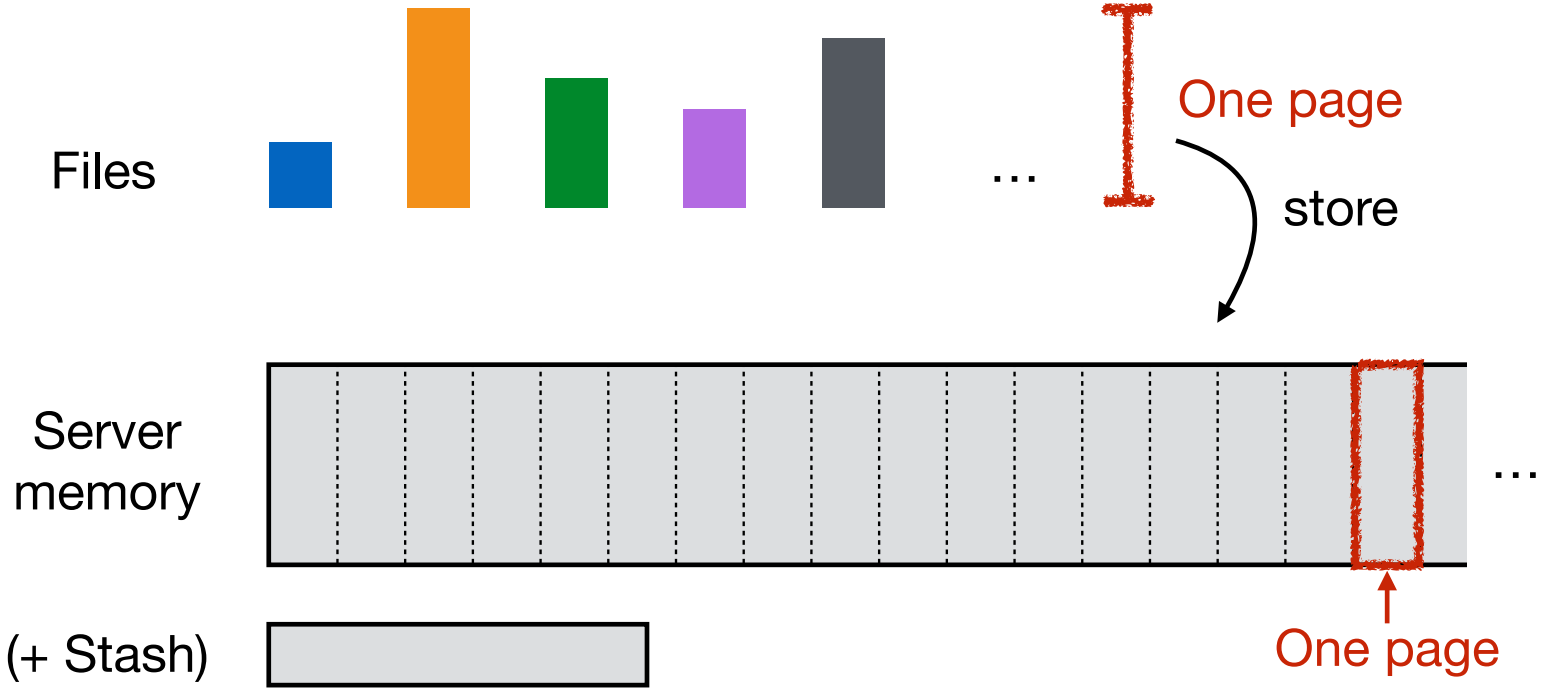
Implementation



Open-source optimized implementation: https://github.com/OpenSSE/

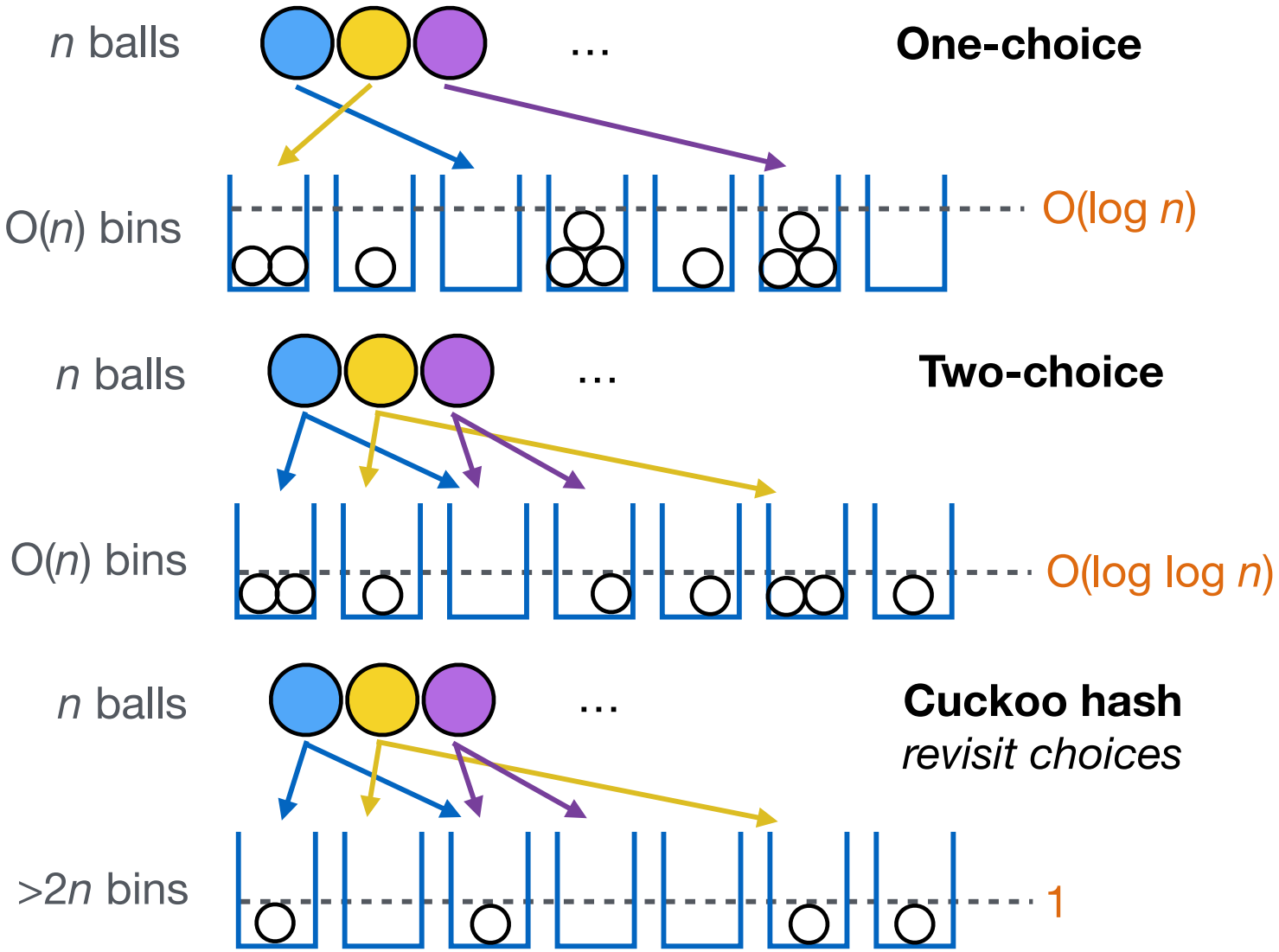
Weighted allocation mechanisms

Problem recap

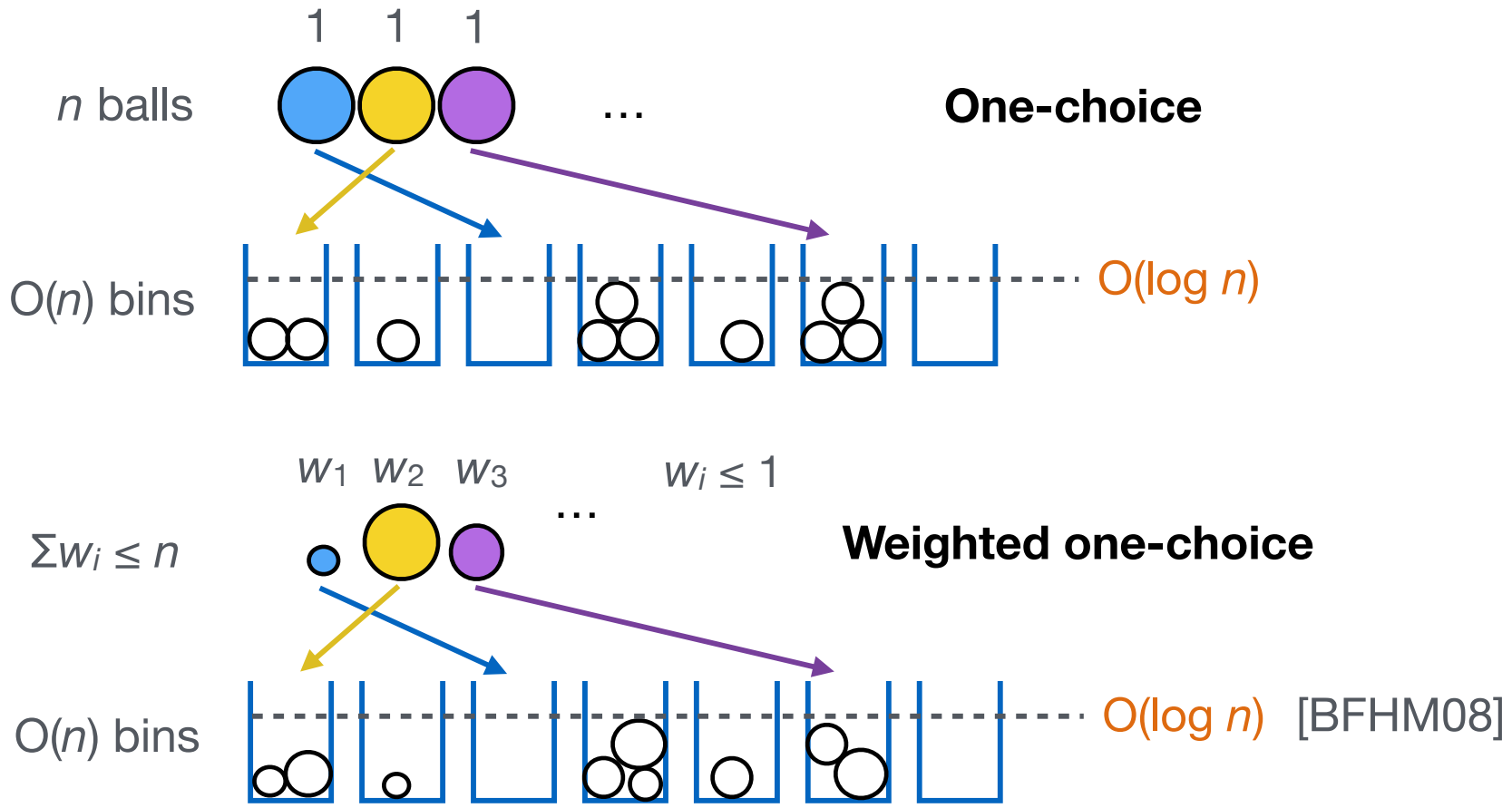


WLOG all files are of size at most one page:

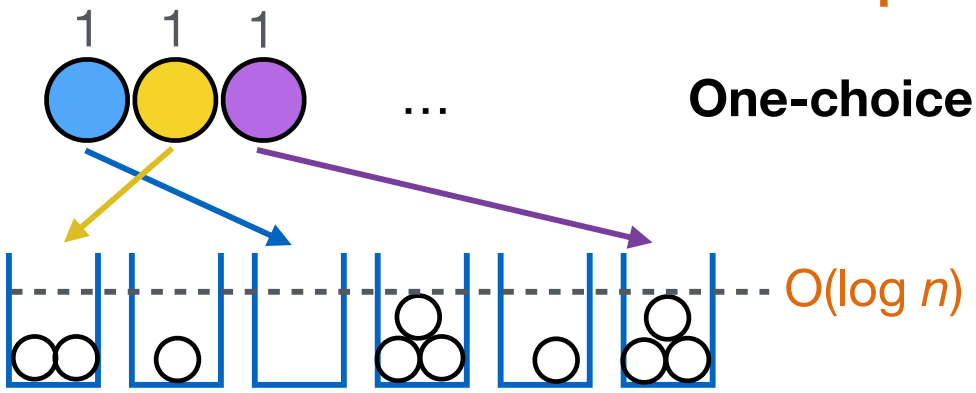
Multiple-choice allocation



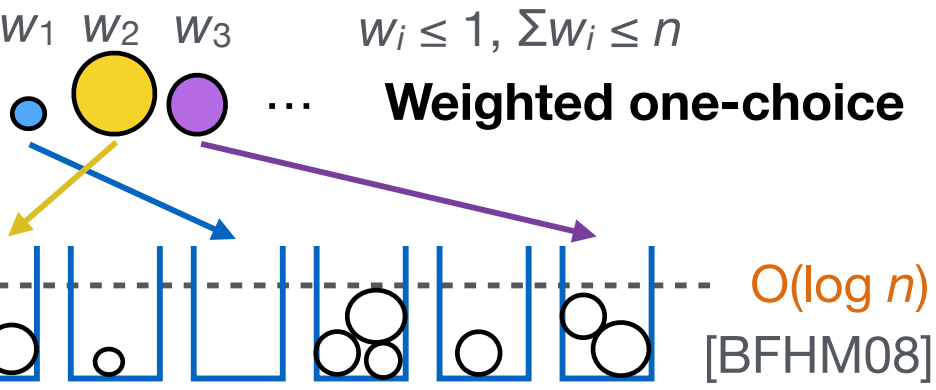
Weighted one-choice



Multiple-choice allocation

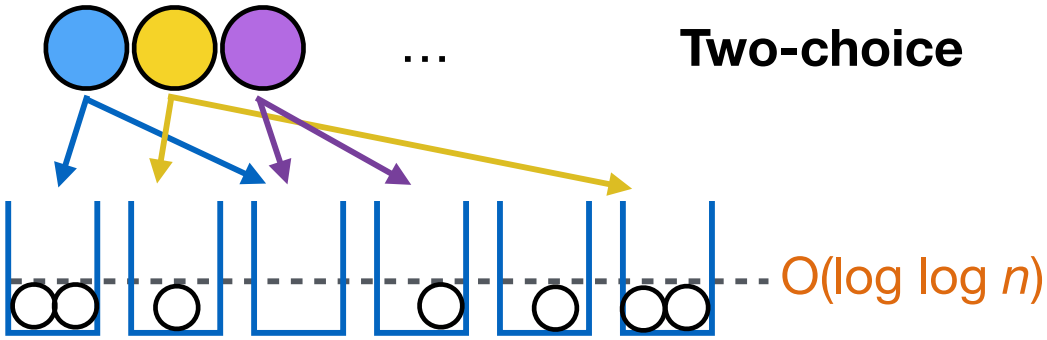


One-choice

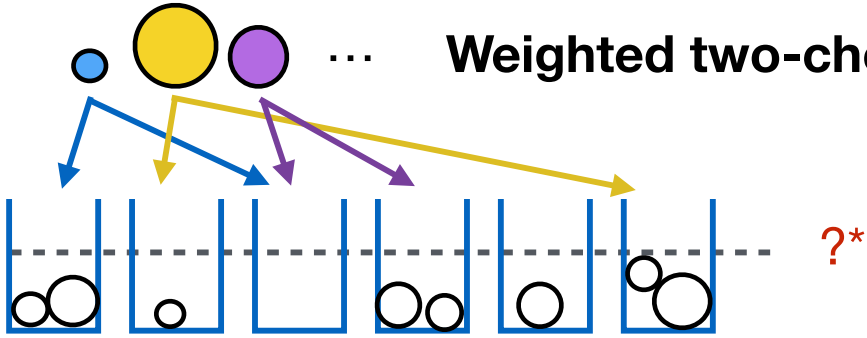


Weighted one-choice

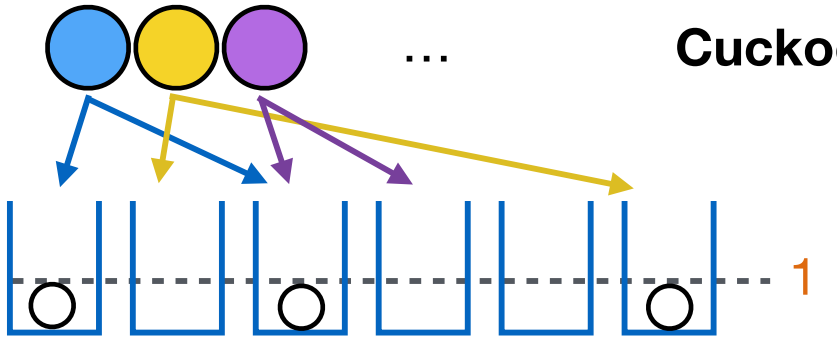
[BFHM08]



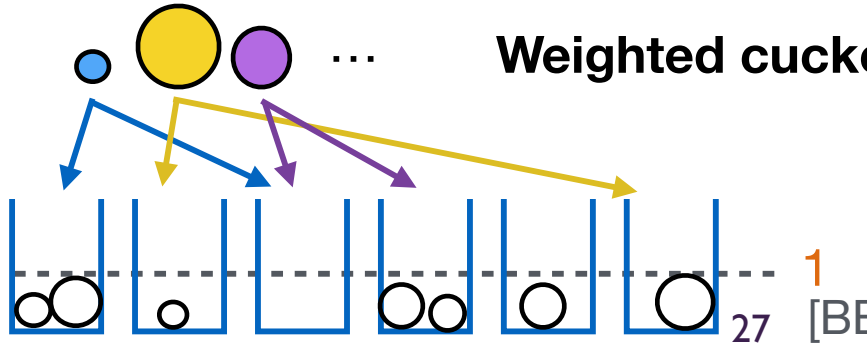
Two-choice



Weighted two-choice



Cuckoo hash



Weighted cuckoo

[BBFMR21]

*Weighted two-choice allocation

Known results:

[TW07,TW14] conjecture is true when ball weights are drawn *independently* from sufficiently smooth distribution.

[MR22] conjecture is true for slight variant of two-choice.

[BFHM08] argues majorization arguments won't suffice.

Conjecture: insert balls of weight (w_i) with weighted two-choice.

If: $\forall i, w_i \leq 1$ and $\Sigma w_i \leq n$

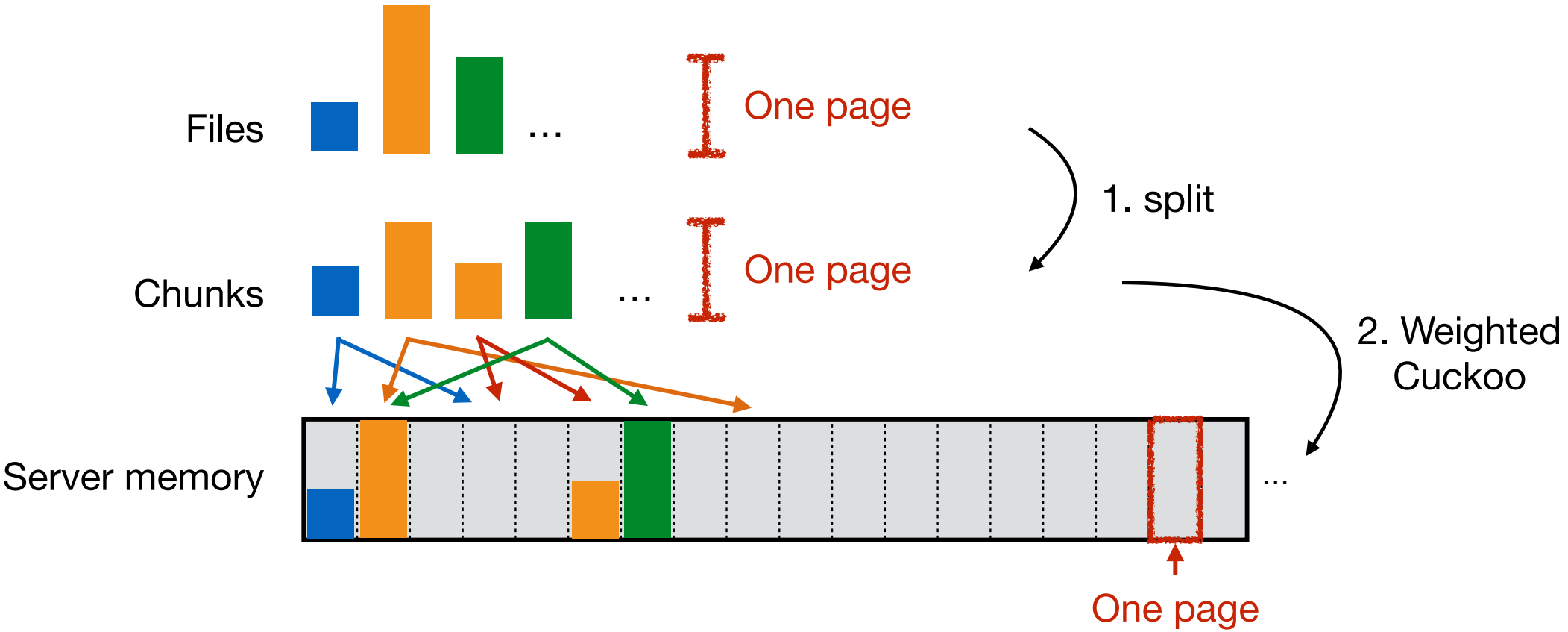
Then: most loaded bin has load $O(\log \log n)$ w.h.p.

A few more conjectures

Conjecture: in weighted cuckoo hashing [BBFMR21], ∃ way to perform insertions in O(1) expected time.

Conjecture: Impossible to have O(1) page efficiency, O(1) storage efficiency, with O(λ) bits of client storage.

Optimal static insulated scheme



Applications

[BBFMR21]:

Weighted cuckoo hashing ⇒ scheme with O(1) page eff. O(1) storage eff.

Same technique likely applies to all reasonable cuckoo variants (e.g. [Y22]).

[PPYY19] uses (unweighted) cuckoo for length-hiding scheme.

Hierarchical ORAM uses cuckoo hashing.

Common point: **static** hash tables.

[MR22]:

Weighted two-choice ⇒ **dynamic** scheme w/ O(log log) page eff. O(1) storage eff.

[APPYY23] uses (unweighted) two-choice for **dynamic** length-hiding scheme.

Weighted ORAM

[AM23]: Weighted Tree ORAM (same proof techniques).

Application to SSE:

Want to hide Access Pattern *and* length. B = upper bound on largest file size.

Recipe:

1. Use Path ORAM with block size B, dimensioned to accomodate **$\Sigma w_i/B$** blocks.
2. Put all your files into ORAM tree *without padding*. Use ORAM normally.
3. **It just works.**

Conclusion

Perspectives

Very simple problem. Just want to read a file without leaking info about lengths of *other* files. *That's it.*

Sub-problem of SSE.

Nice fallout: weighted cuckoo hashing, weighted ORAM.

In practice:

Access pattern:

very damaging.

vs.

This entire talk:
paying heavily to insulate subtle length leakage.

Conjecture: with forward security, impossible to have O(1) page efficiency, O(1) storage efficiency.

Thank you!