Dynamic Local Searchable Symmetric Encryption

- Brice Minaud INRIA — ENS — CNRS — PSL University
- Michael Reichle INRIA — ENS — CNRS — PSL University

Roadmap Crypto’22

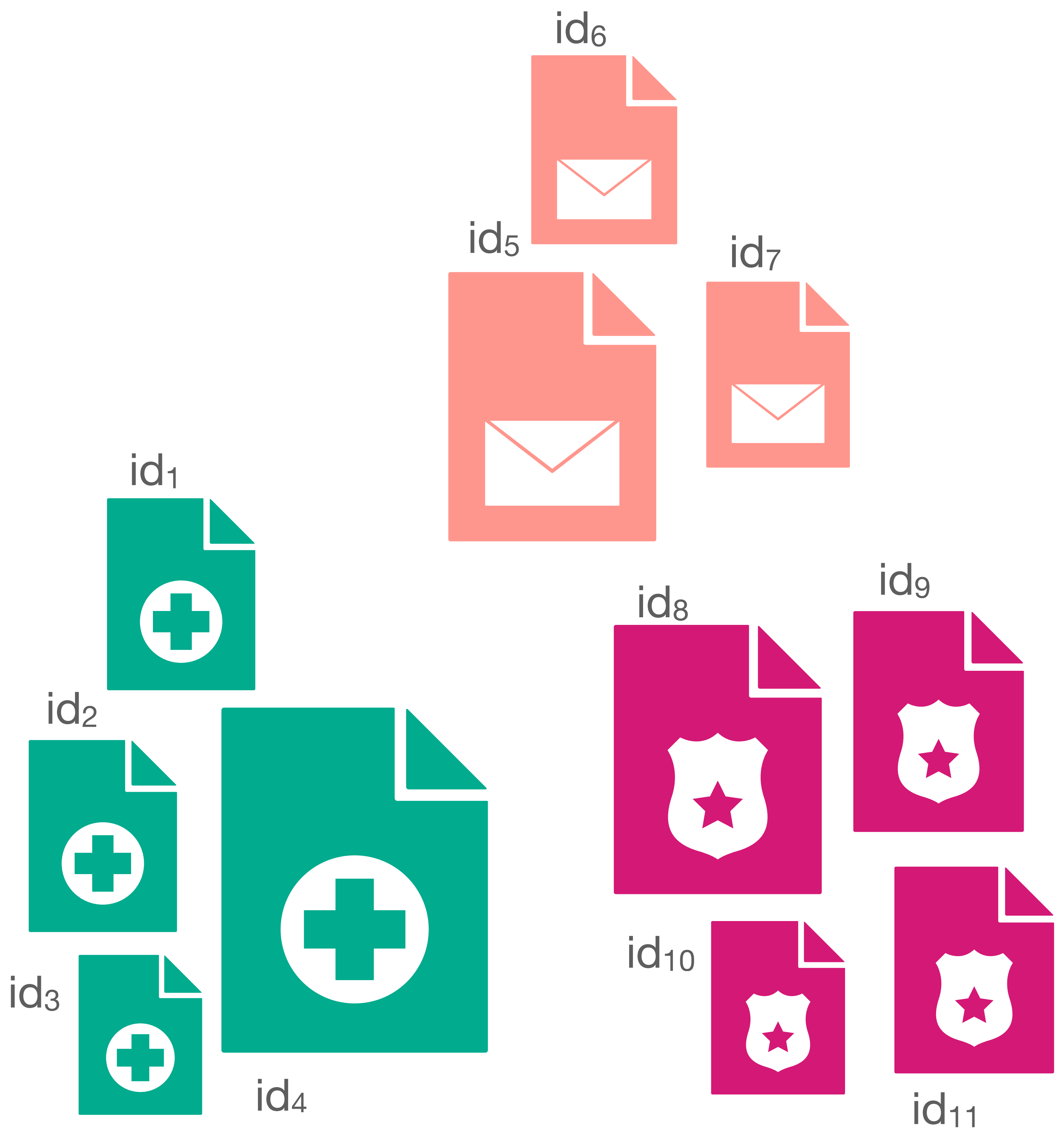
Searchable Symmetric Encryption

Roadmap Crypto’22

Roadmap Crypto’22

Motivation

Outsource confidential information



Motivation

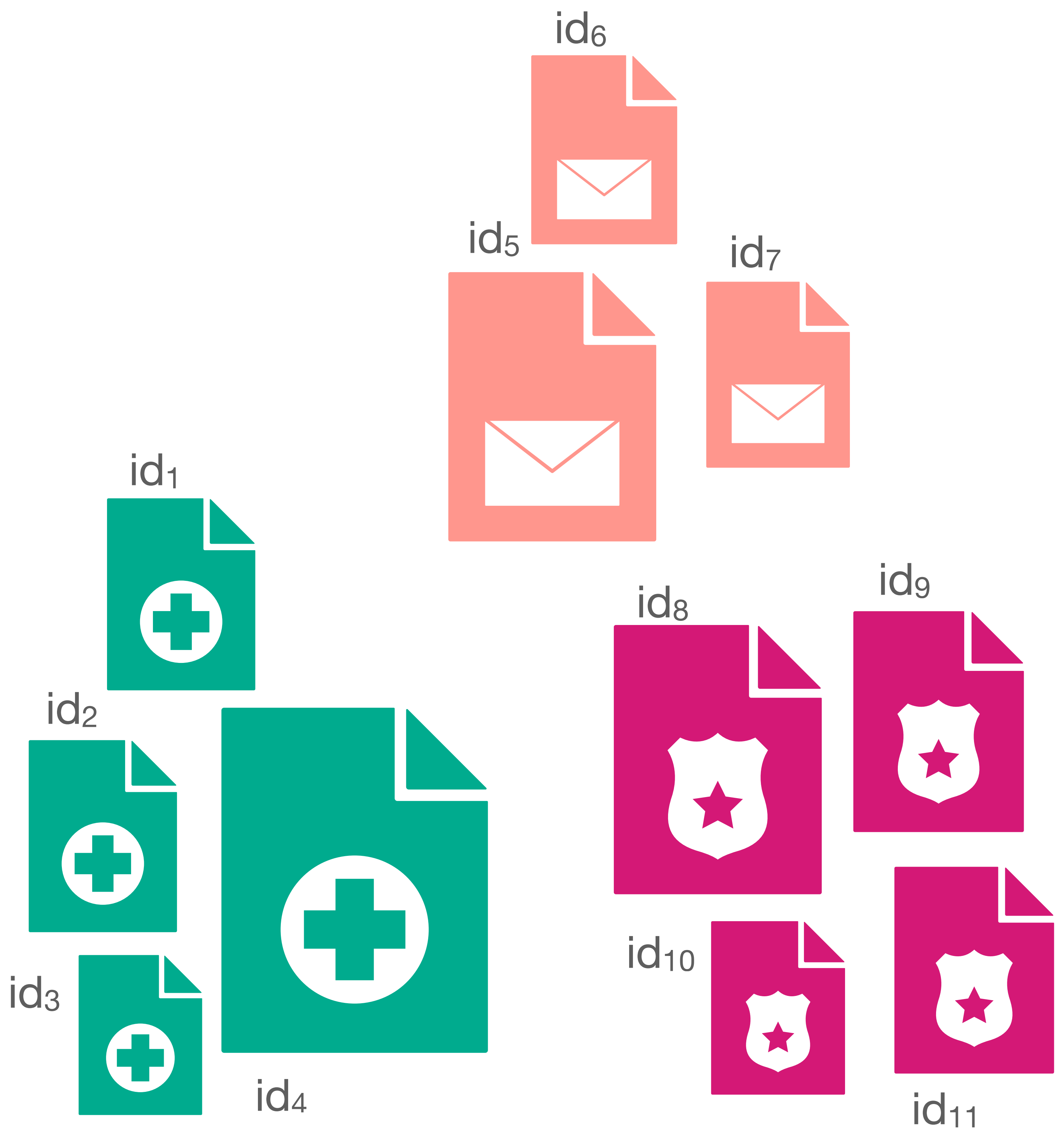
Outsource confidential information

Reverse Index:

“covid” ↦ id_1, id_3

“fire” ↦ id_2, id_3, id_6

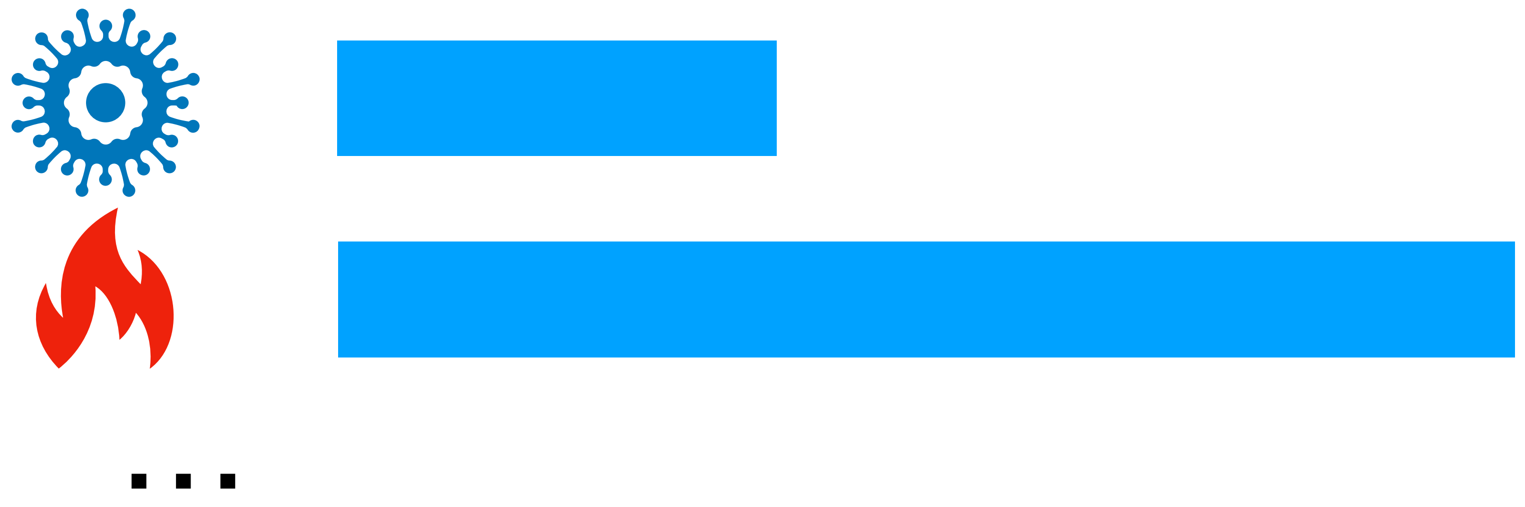
…

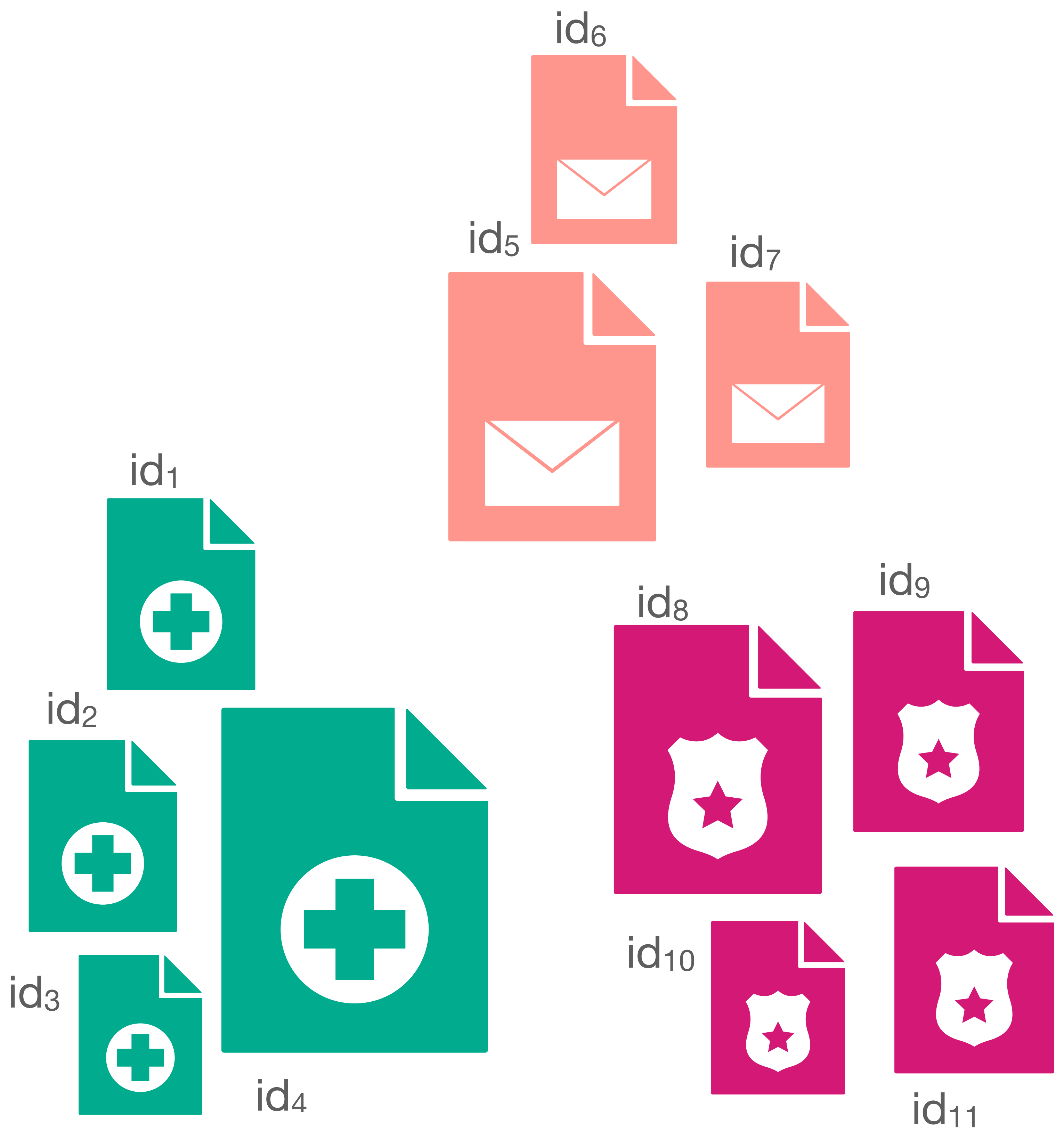


Motivation

Outsource confidential information

Reverse Index:

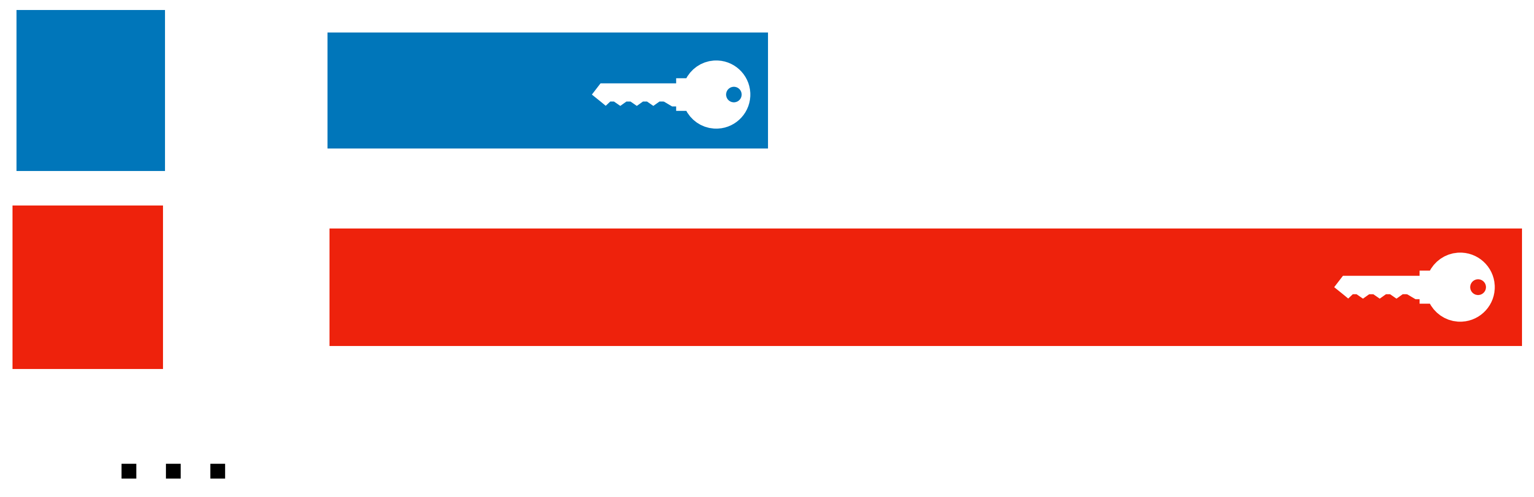


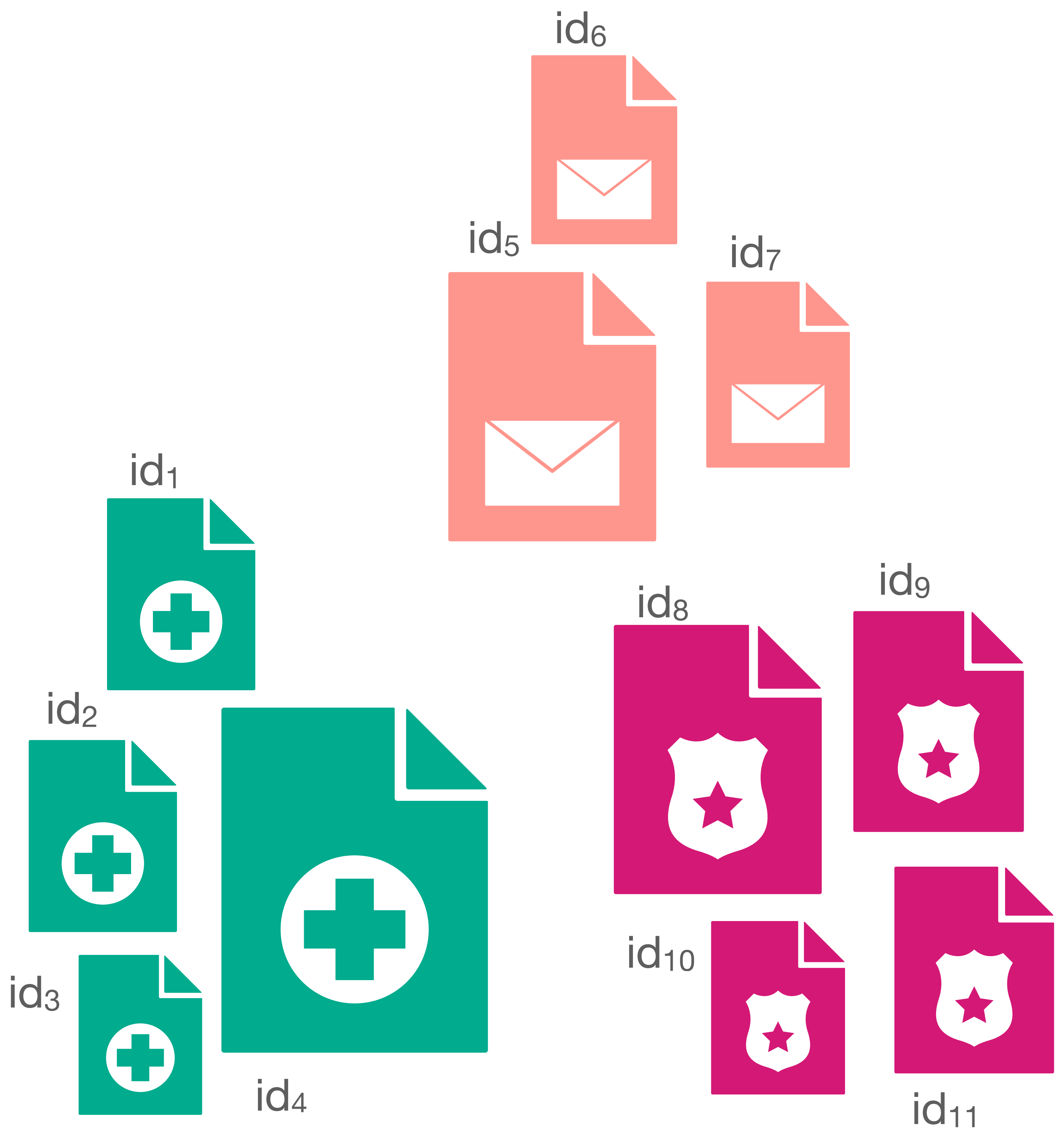


Motivation

Outsource confidential information

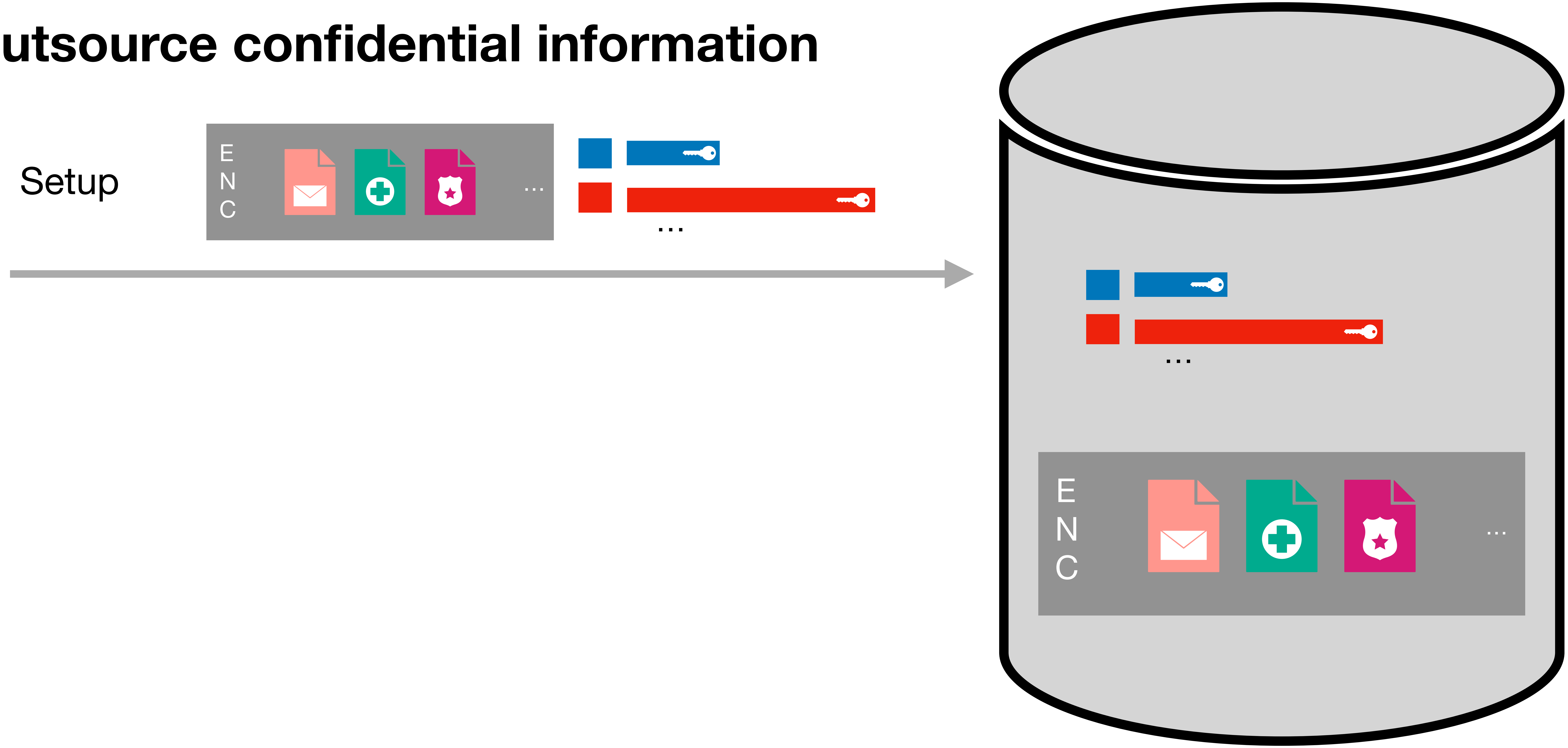
Encrypted Reverse Index:





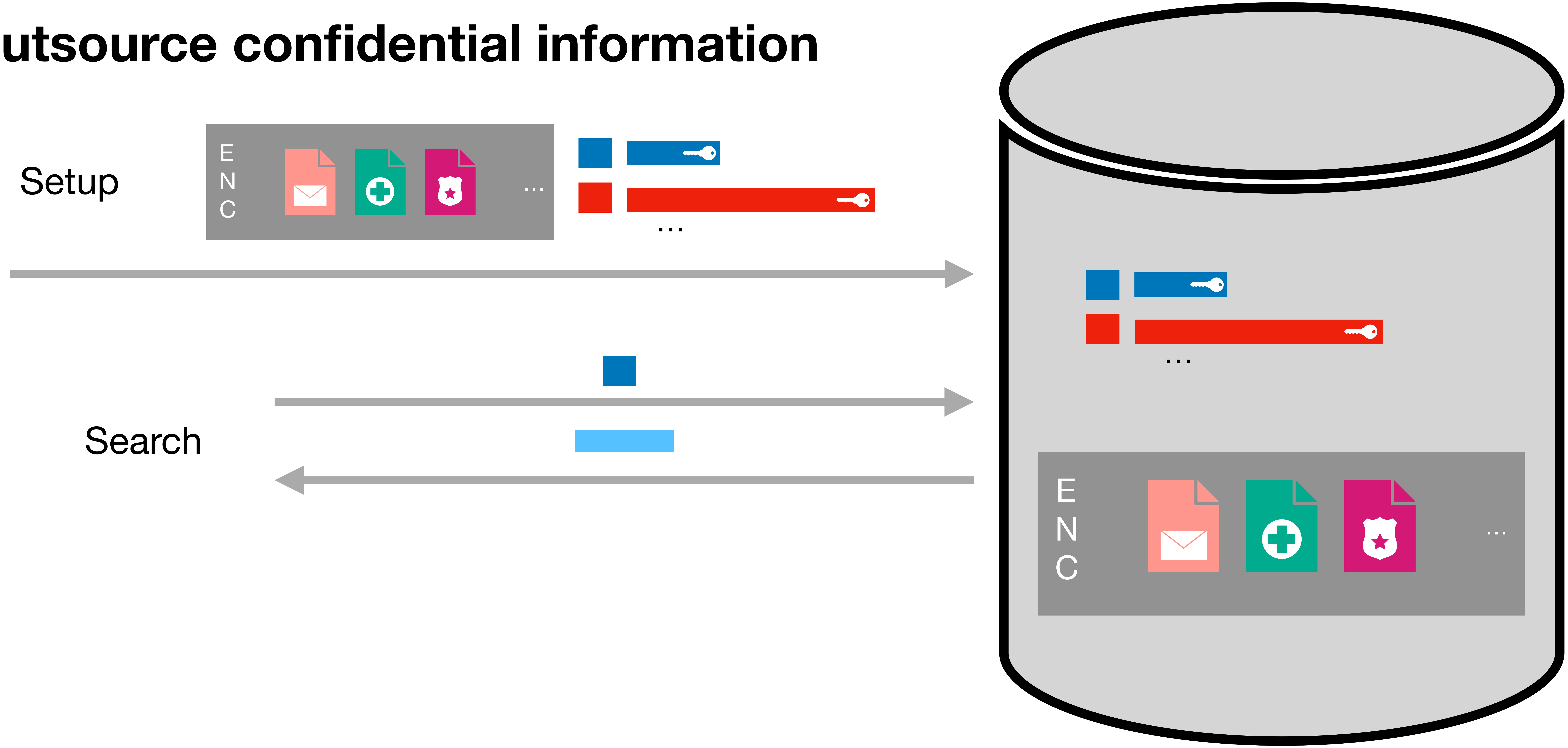
Motivation

Outsource confidential information



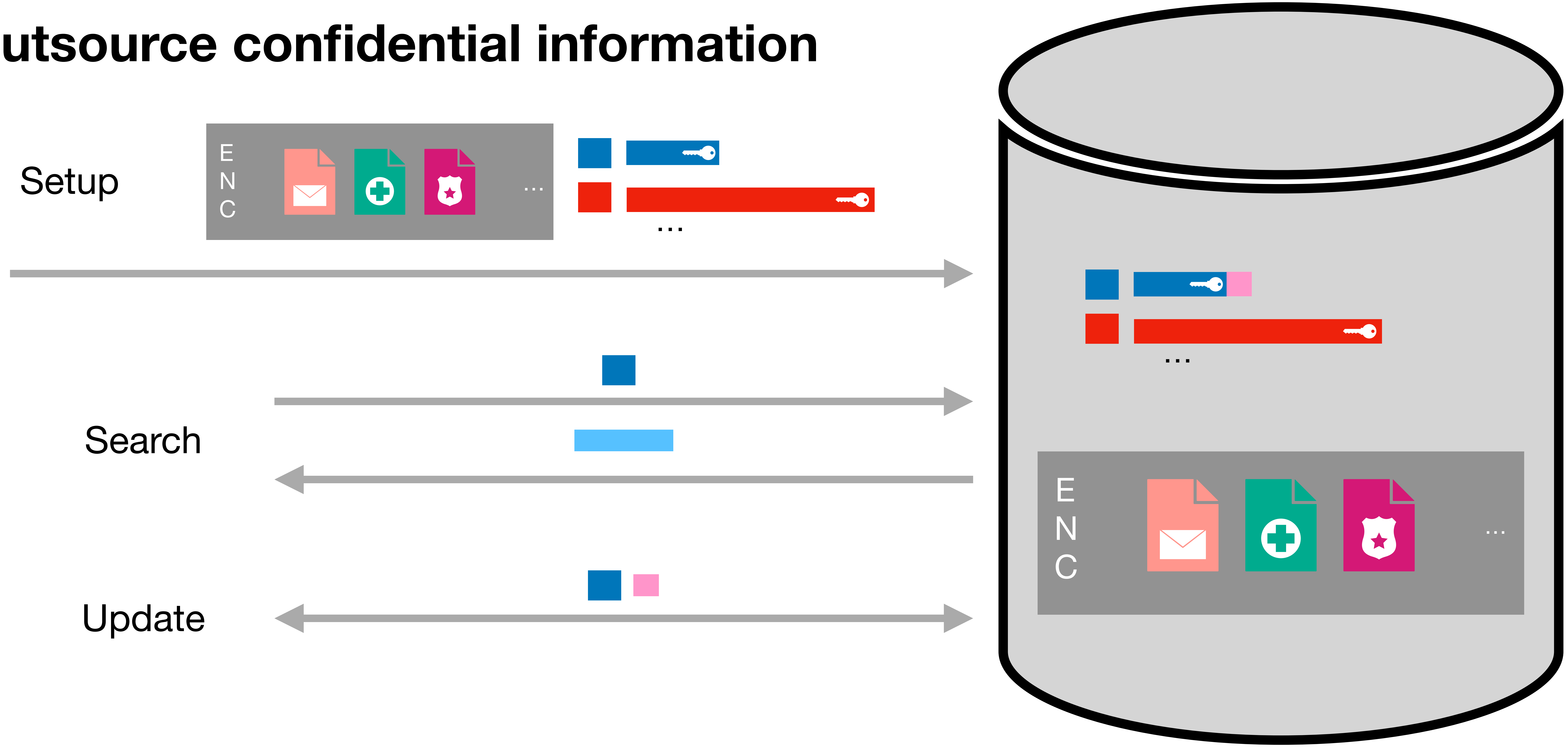
Motivation

Outsource confidential information



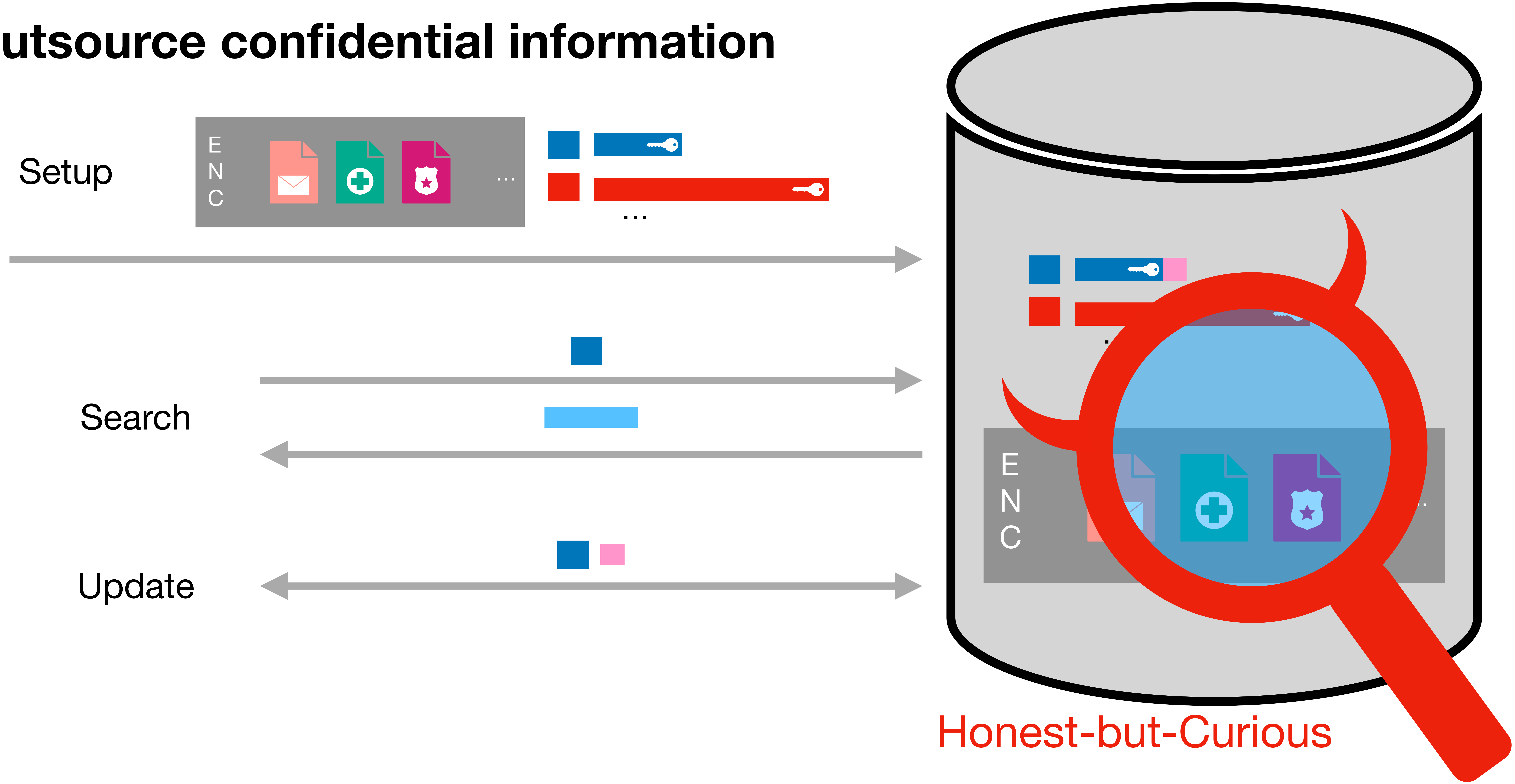
Motivation

Outsource confidential information



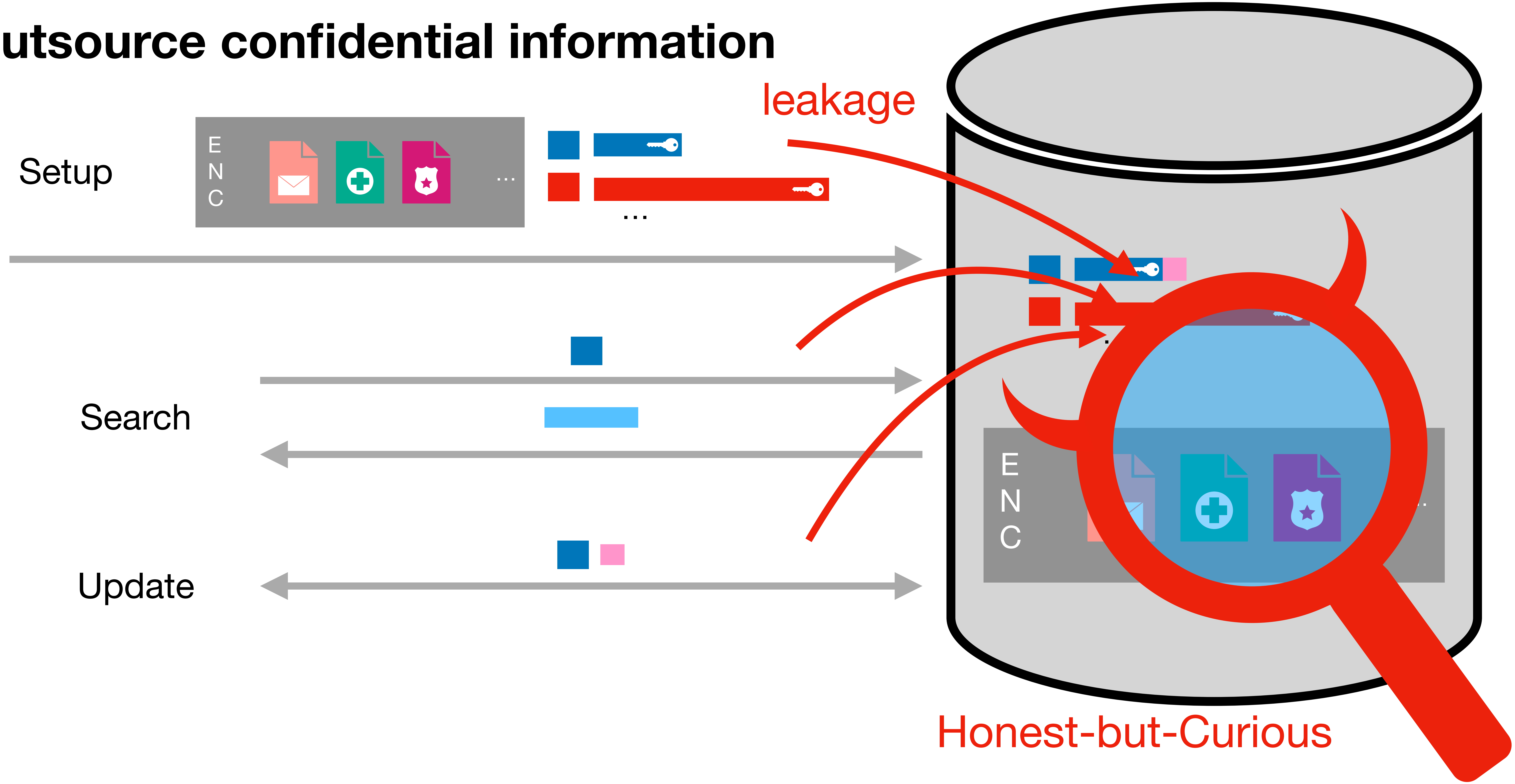
Motivation

Outsource confidential information



Motivation

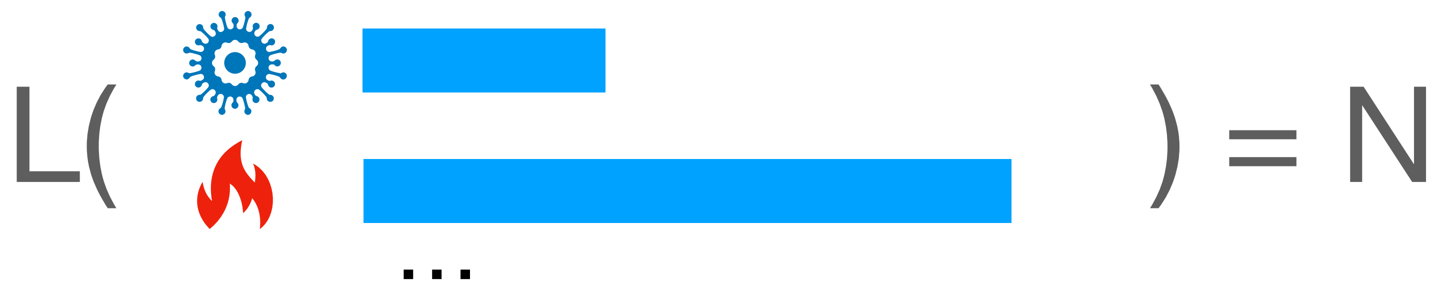
Outsource confidential information



Leakage

Forward Security

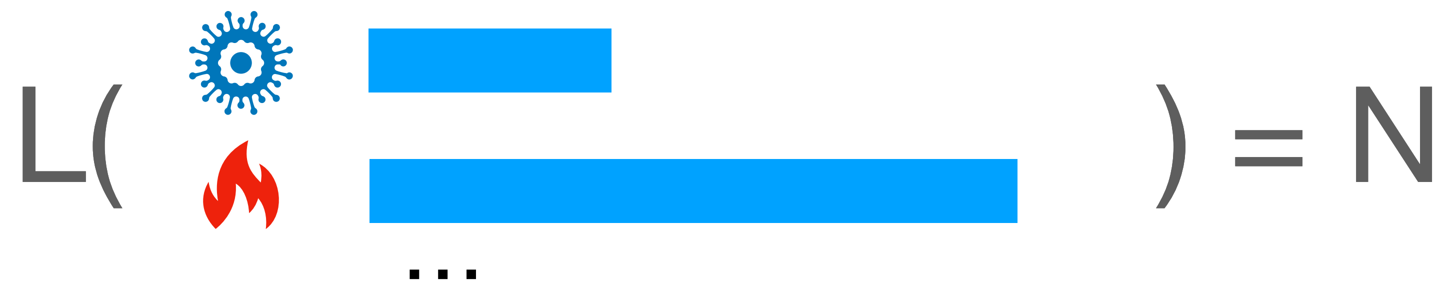
- **Security model:** Server learns nothing except leakage
- Allows for tradeoffs: efficiency / security
- Common leakage:
  - **Setup:** database size
  - **Search:** query pattern, access pattern, number of matching IDs
  - **Update:**



Leakage

Non-Forward Secure SSE

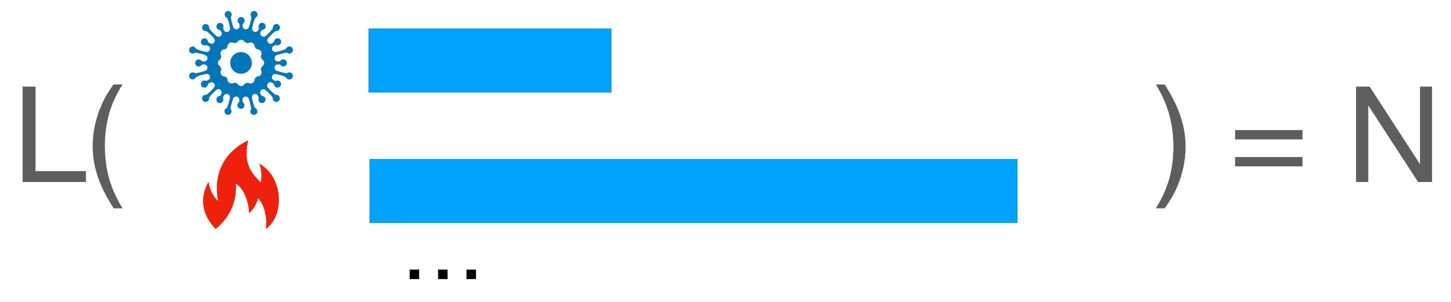
- **Security model:** Server learns nothing except leakage
- Allows for tradeoffs: efficiency / security
- Common leakage:
  - **Setup:** database size
  - **Search:** query pattern, access pattern, number of matching IDs
  - **Update:** query pattern, number of matching IDs



Leakage

Non-Forward Secure SSE

- **Security model:** Server learns nothing except leakage
- Allows for tradeoffs: efficiency / security
- Common leakage:

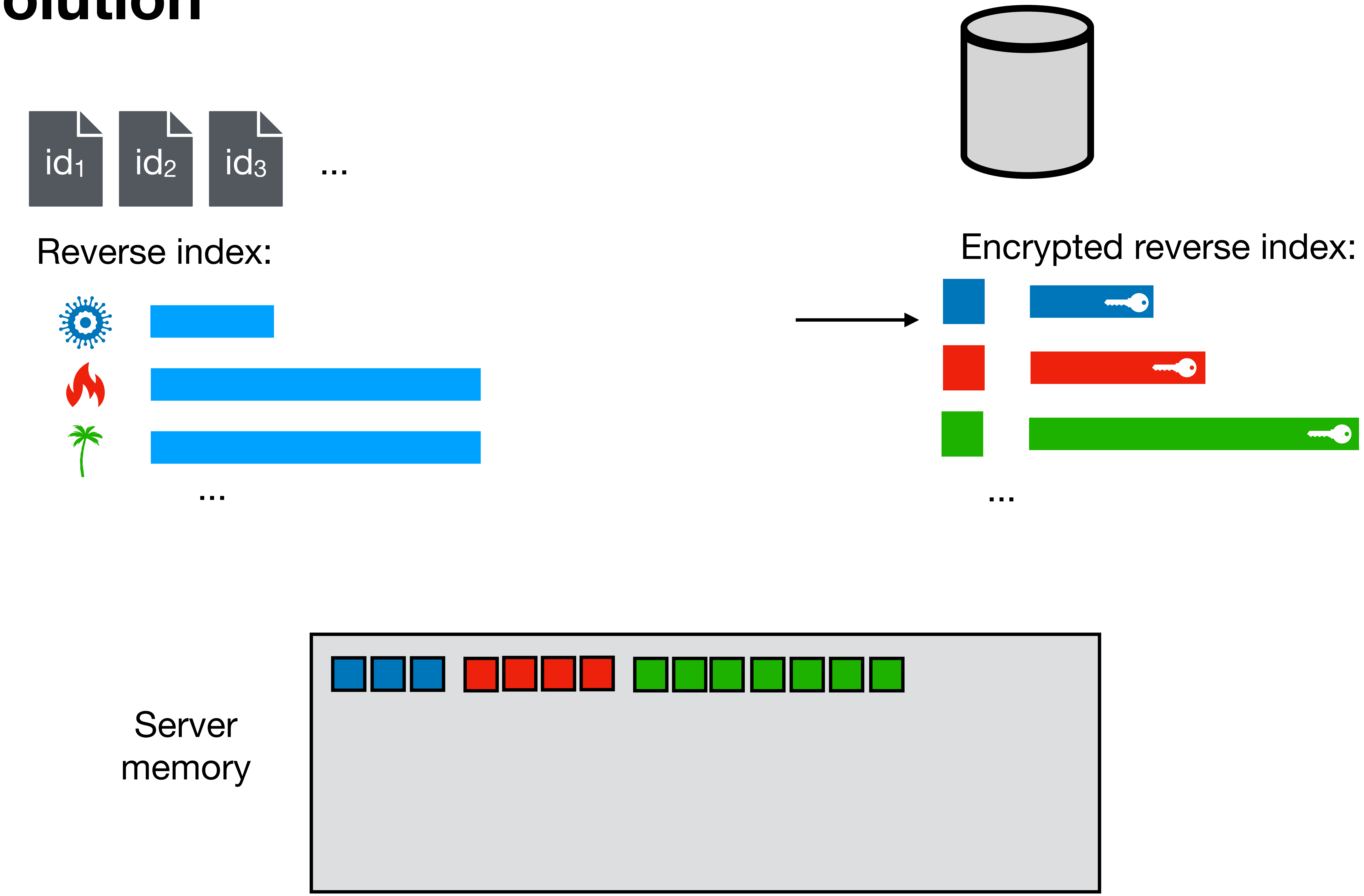


  - **Setup:** database size
  - **Search:** query pattern, access pattern, number of matching IDs
  - **Update:** query pattern, number of matching IDs

No leakage about **unqueried** keywords

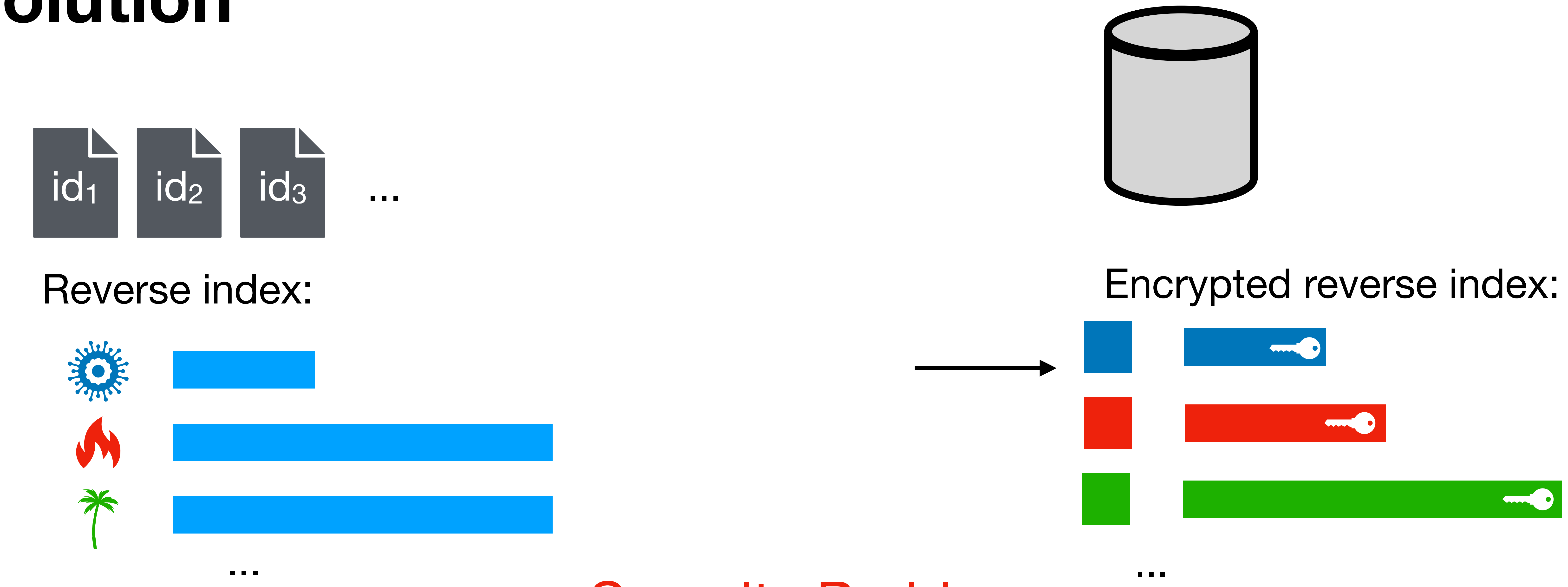
SSE

Non-Solution



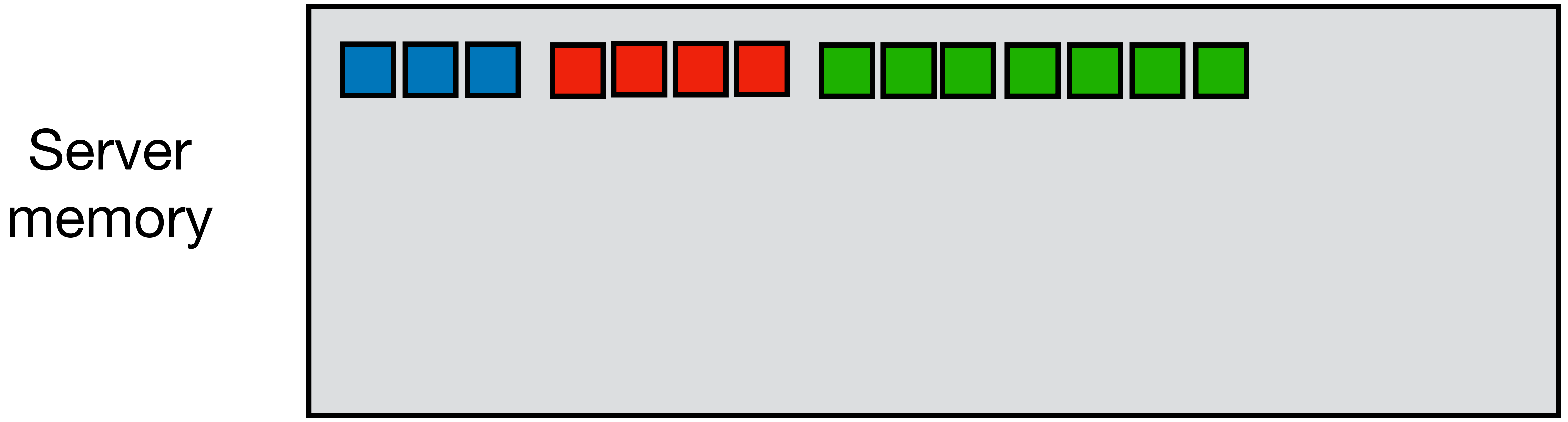
SSE

Non-Solution



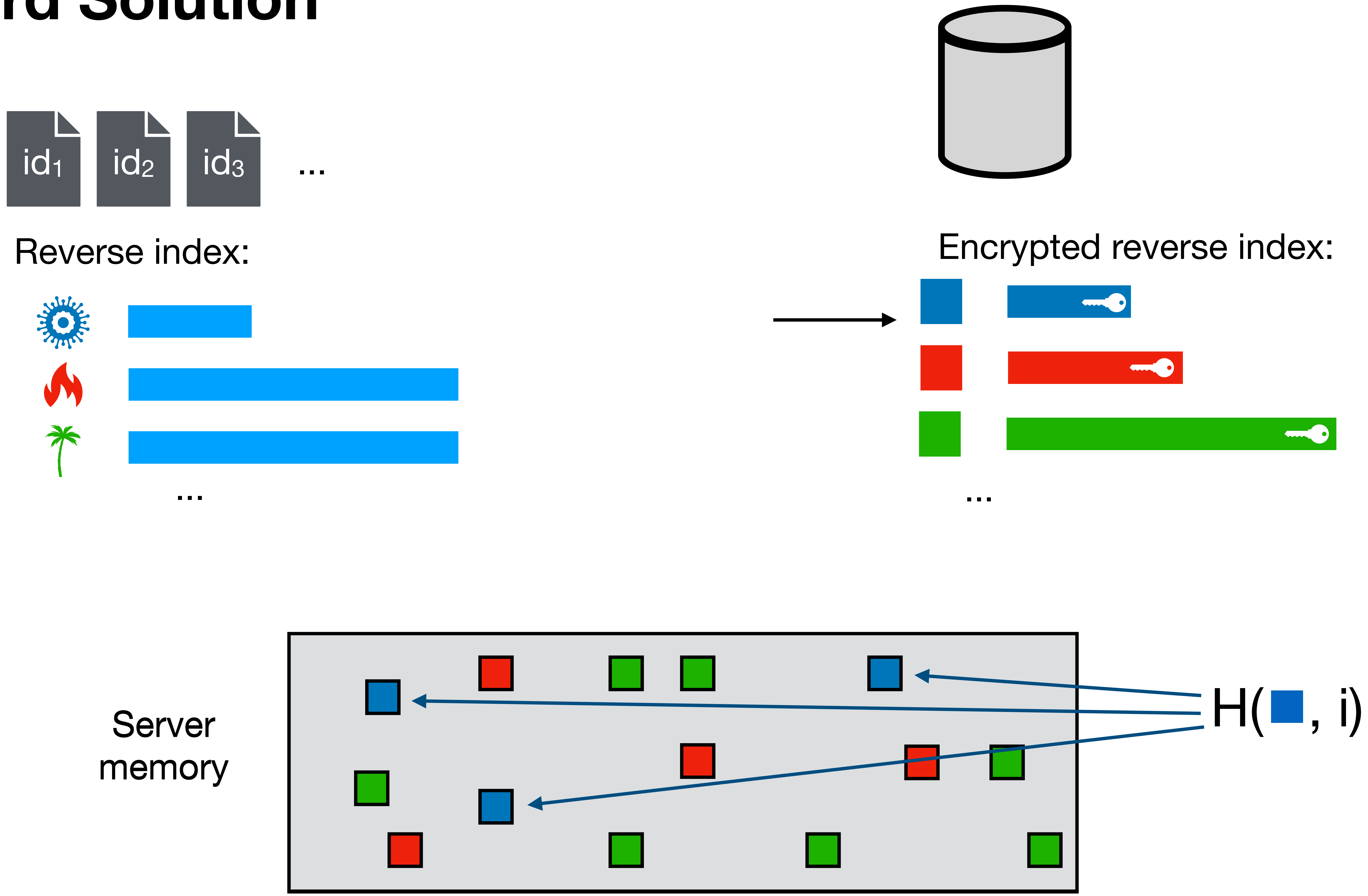
Security Problem

Here, list locations depend on other lists



SSE

Standard Solution



id_1 id_2 id_3 …

Reverse index:

…

Encrypted reverse index:

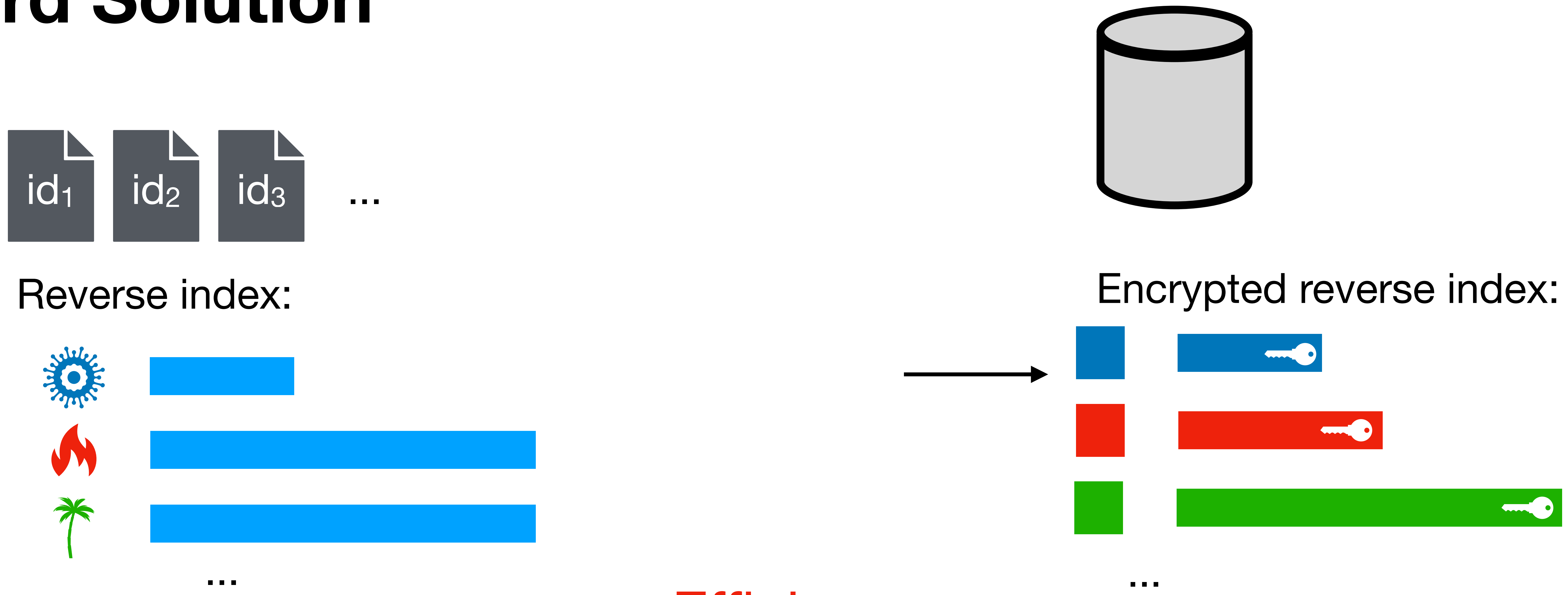
…

Server memory

H(■, i)

SSE

Standard Solution



id_1 id_2 id_3 …

Reverse index:

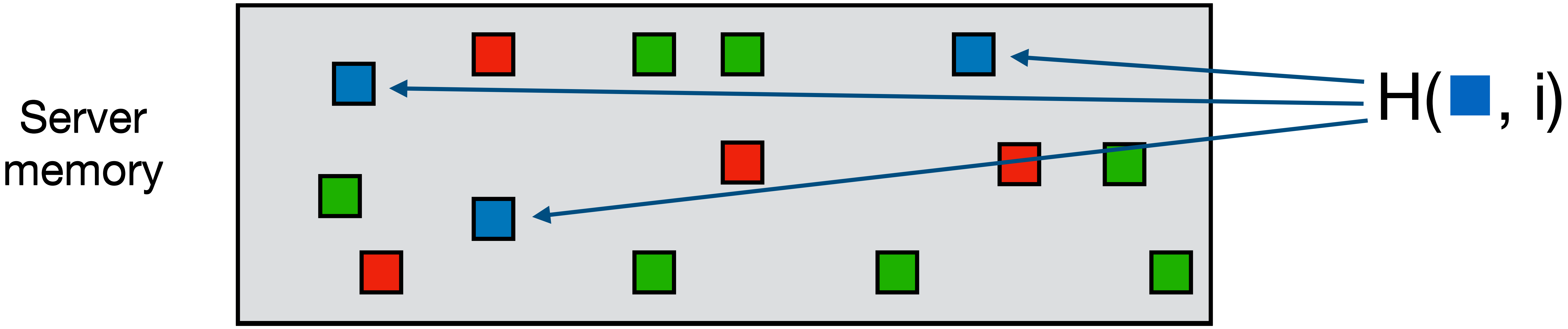
…

Encrypted reverse index:

…

Efficiency

Retrieval induces many random memory accesses



Server memory

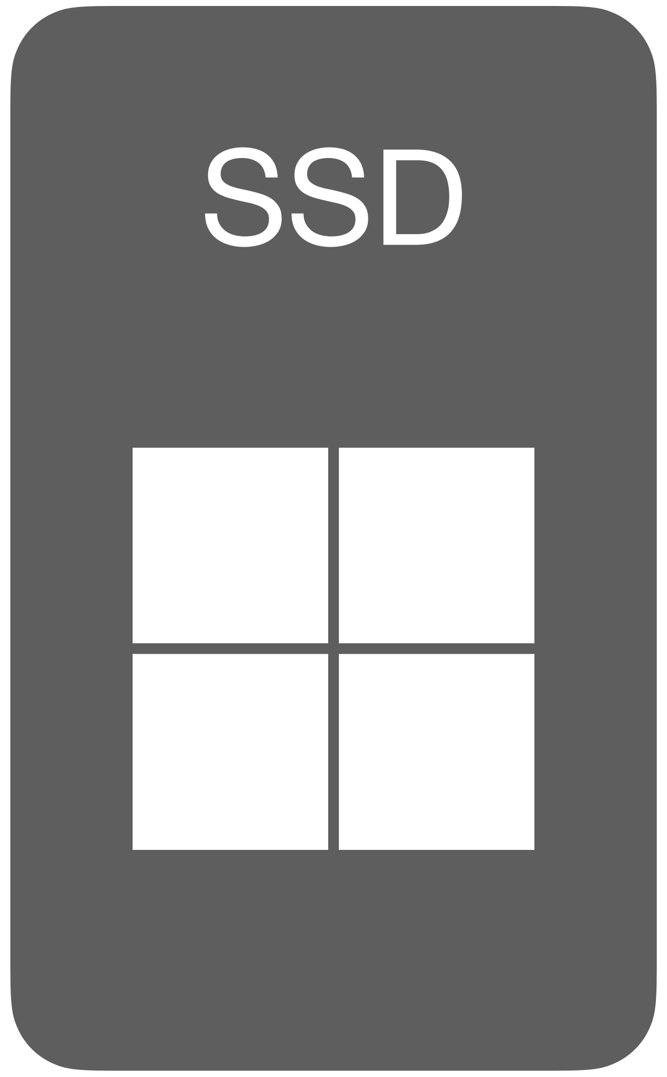
H(🟦, i)

Memory Efficiency

HDDs vs SSDs

Locality:

Number of Read
(non-adjacent)
Memory Locations

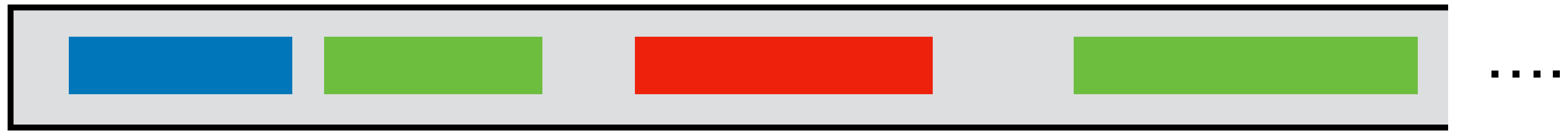


Page Efficiency:

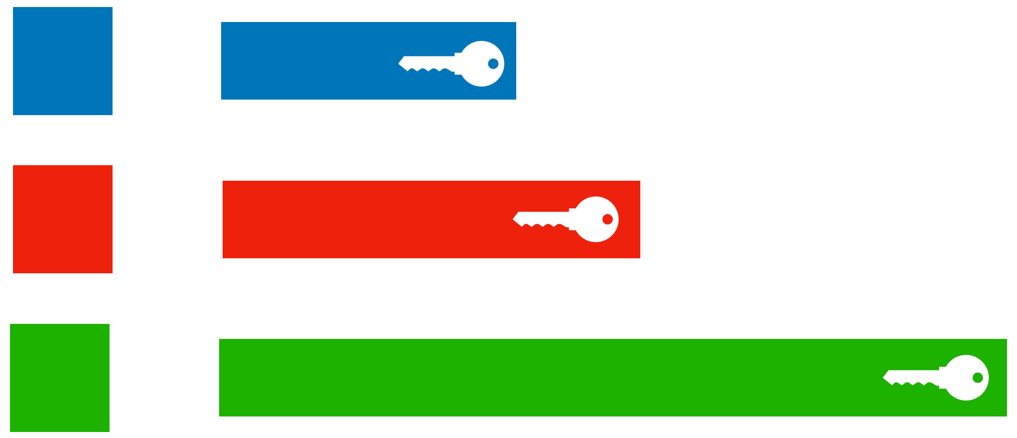
Number of Read
Pages per Query

Locality

Look at server memory as array



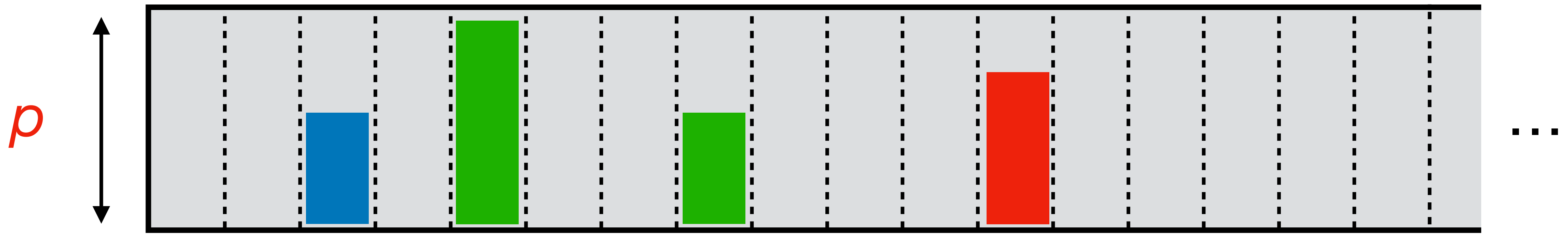
Encrypted reverse index:



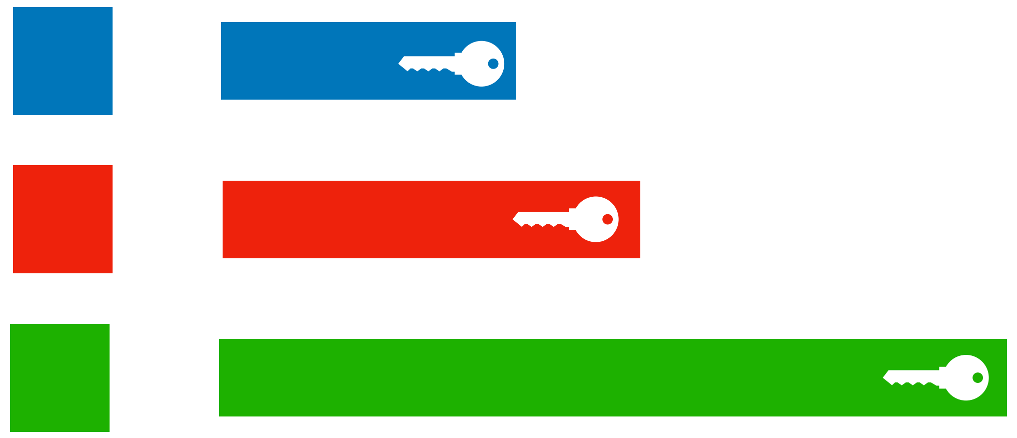
- Goal:
  - Locality: read at most constant disjoint intervals
  - Read Efficiency: read **as little extra data** as possible
  - Storage Efficiency: At most constant blow-up of server memory

Page Efficiency

Look at server memory as pages



Encrypted reverse index:



- Goal:
  - Page Efficiency: Store identifier lists in **as little pages** as possible
  - Storage Efficiency: At most constant blow-up of server memory

Current State of the Art

Constant Storage Efficiency (and Locality)

Read Efficiency:

- **[ANSS16]:** $\tilde{O}(\log \log N)$ *
- **[ASS18]:** $\tilde{O}(\log \log \log N)$ **
- **[DPP18]:** $O(\log^{2/3+\varepsilon} N)$

* restriction on longest list
** even stronger restriction on longest list

Page Efficiency:

- **[BBFMR21]:** $O(1)$ +

+ logarithmic client storage

Current State of the Art

Constant Storage Efficiency (and Locality)

Read Efficiency:

- **[ANSS16]:** $\tilde{O}(\log \log N)$ *
- **[ASS18]:** $\tilde{O}(\log \log \log N)$ **
- **[DPP18]:** $O(\log^{2/3+\varepsilon} N)$

* restriction on longest list
** even stronger restriction on longest list

Page Efficiency:

- **[BBFMR21]:** $O(1)$ +

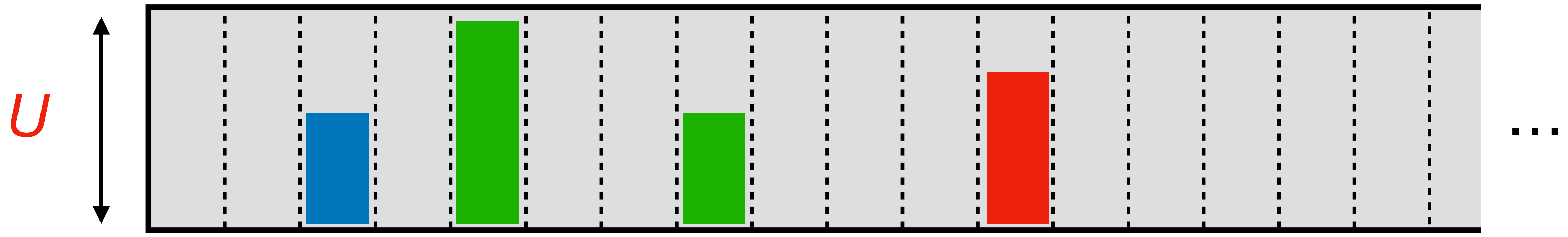
+ logarithmic client storage

Exclusively static constructions

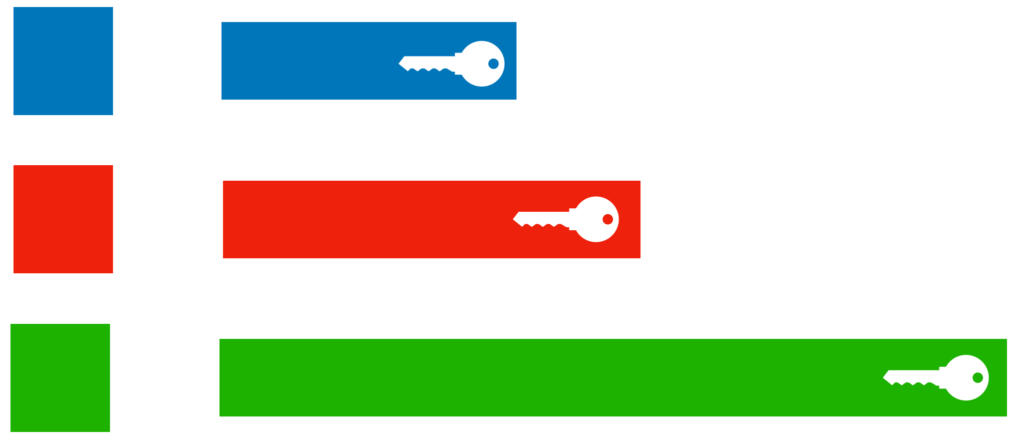
Framework

Balls-into-Bins

Look at server memory as bins



Encrypted reverse index:

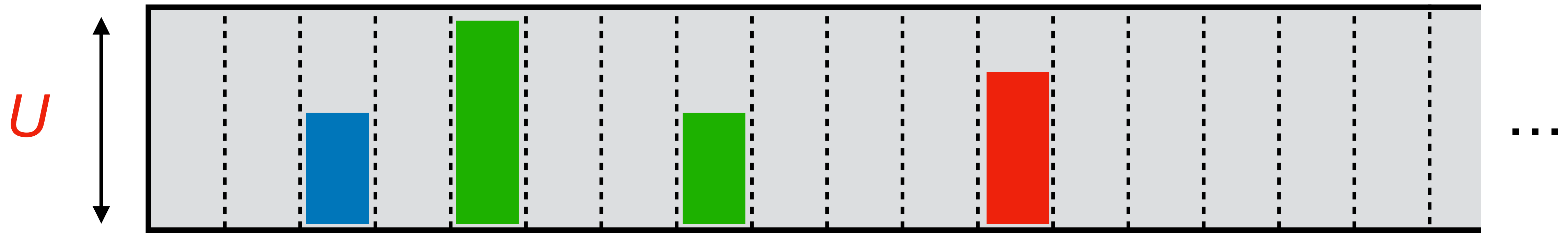


- Approach: throw *weighted* balls into bins via hash function
- Goal: upper bound *U* on maximal bin load

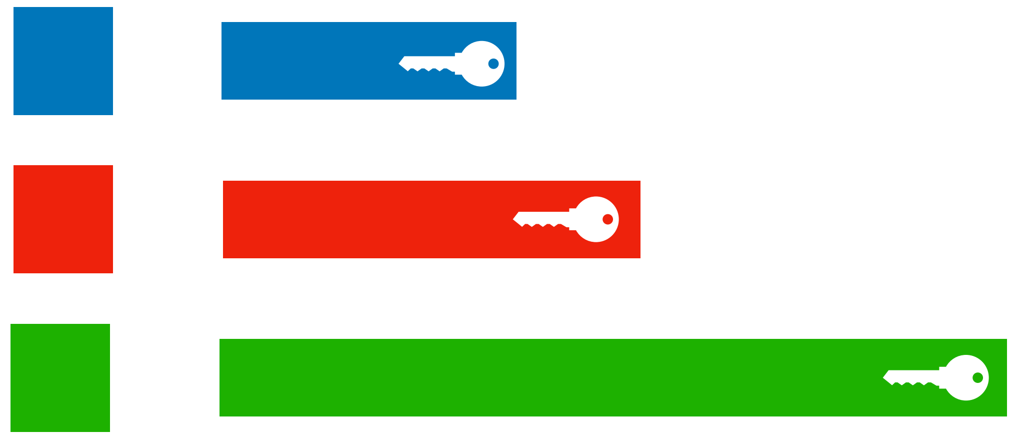
Framework

Balls-into-Bins

Look at server memory as bins



Encrypted reverse index:

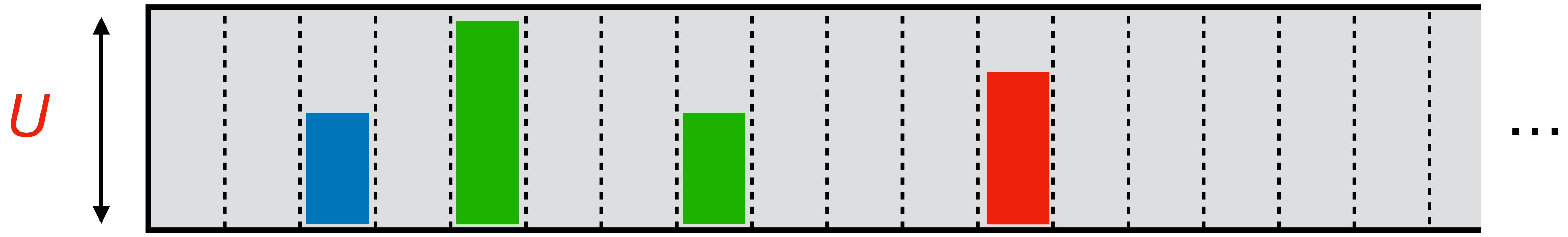


- Approach: throw *weighted* balls into bins via hash function
- Goal: upper bound *U* on maximal bin load
- **Page Efficiency:** interpret bins as pages [BBFMR21]

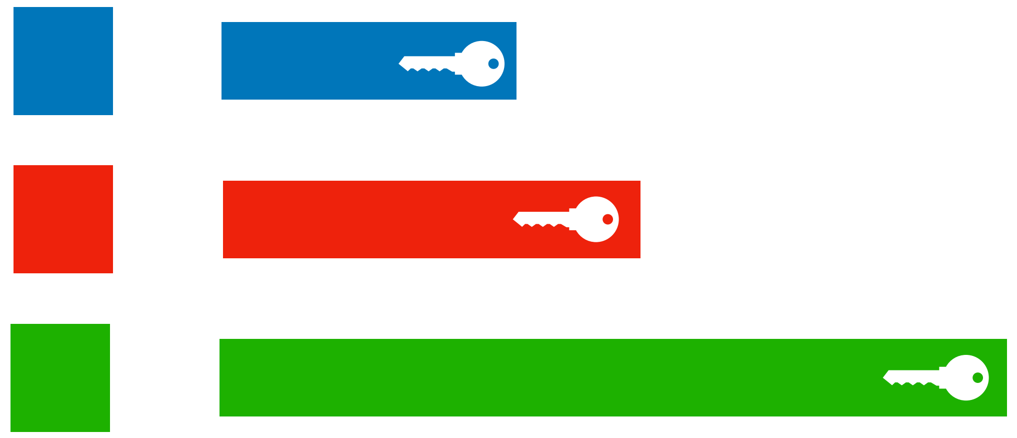
Framework

Balls-into-Bins

Look at server memory as bins



Encrypted reverse index:



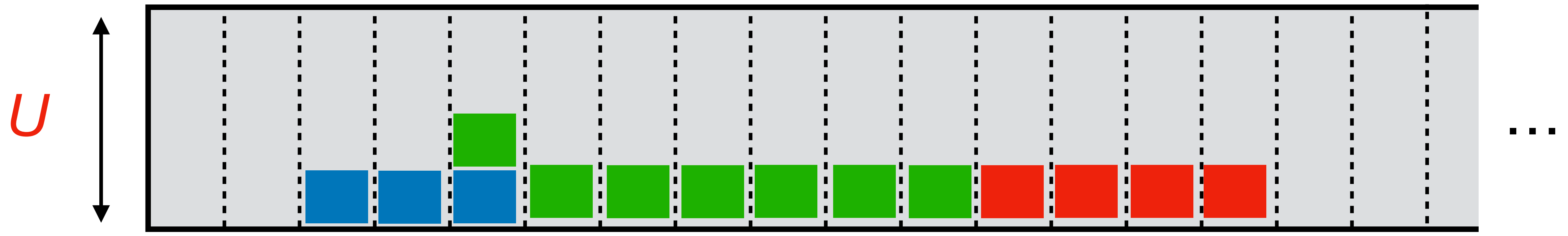
- Approach: throw *weighted* balls 🟦 into bins via hash function
- Goal: upper bound *U* on maximal bin load
- **Page Efficiency:** interpret bins as pages [BBFMR21]

→ SSE with $O(U)$ page efficiency

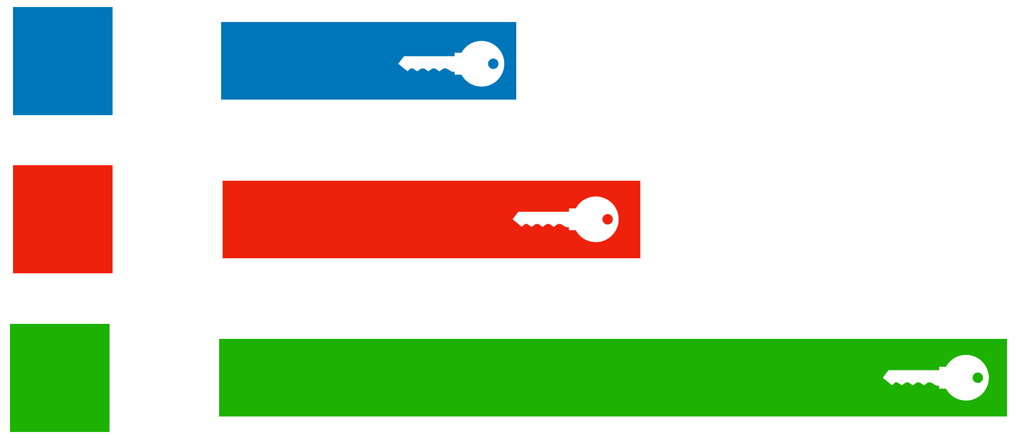
Framework

Balls-into-Bins

Look at server memory as bins



Encrypted reverse index:

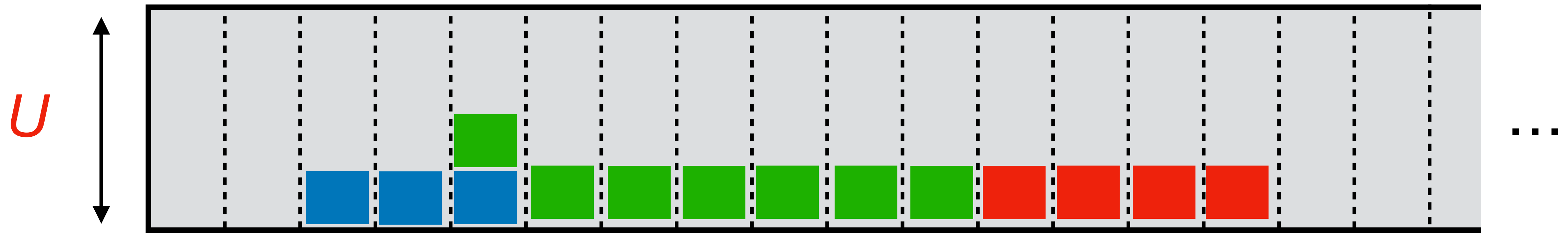


- Approach: throw *weighted* balls into bins via hash function
- Goal: upper bound U on maximal bin load
- **Locality:** store IDs in consecutive bins [ANSS16]

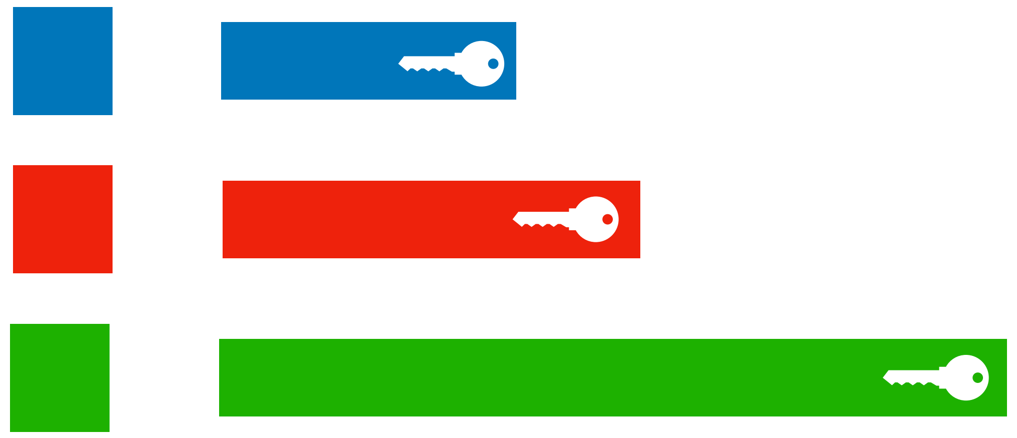
Framework

Balls-into-Bins

Look at server memory as bins



Encrypted reverse index:

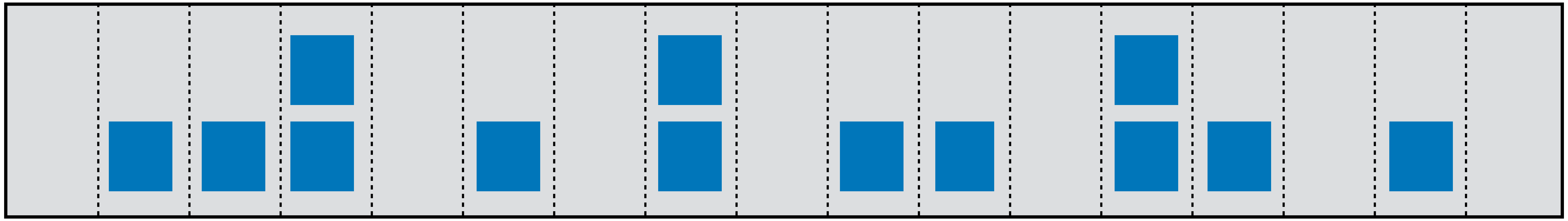


- Approach: throw *weighted* balls into bins via hash function
- Goal: upper bound *U* on maximal bin load
- **Locality:** store IDs in consecutive bins [ANSS16]

→ SSE with *O*(*U*) read efficiency, *O(1)* locality

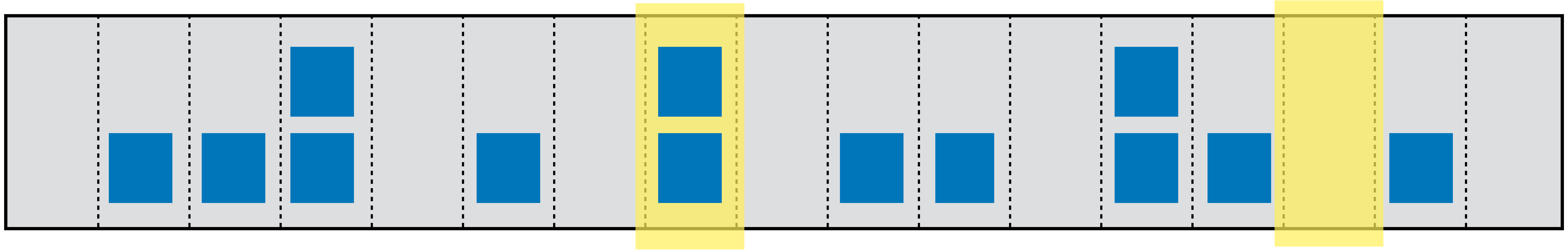
Two-Choice

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



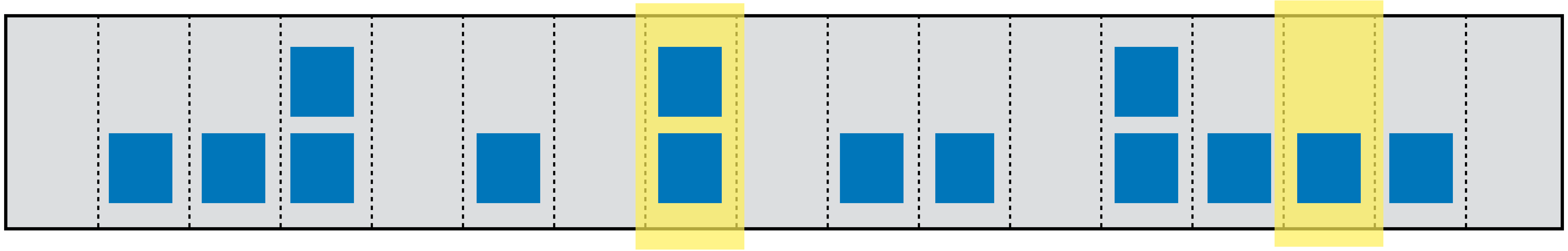
Two-Choice

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



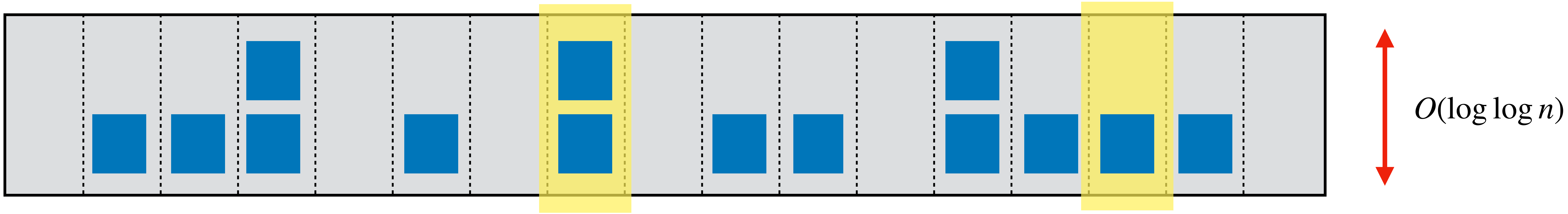
Two-Choice

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



Two-Choice

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



Weighted Two-Choice

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

Weighted Two-Choice

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

- **Require:** weighted 2C
- **Problem:** existing results on 2C are conditional (*distributions, presorting, …)*

Weighted Two-Choice

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

- **Require:** weighted 2C
- **Problem:** existing results on 2C are conditional (*distributions, presorting, …)*

- **Layered2C: modify comparison**
  - behaves "almost" like standard two-choice
  - no distributional assumption or presorting
  - Tight upper bound *U*
- **LayeredSSE:** DSSE with $\tilde{O}(\log \log N/p)$ page efficiency

Generic Local Transform

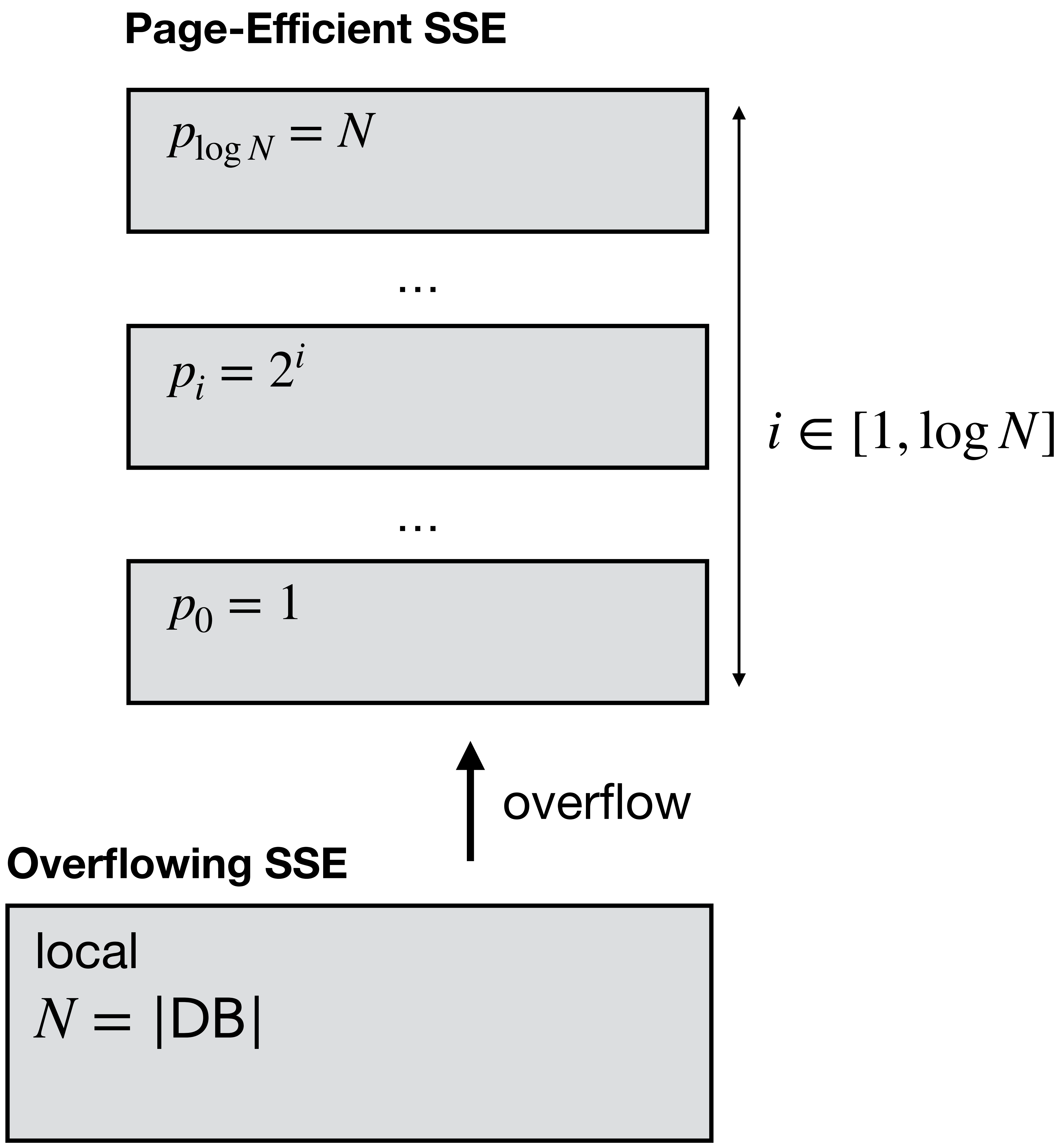
Page-Efficient DSSE → Local DSSE

Overflowing SSE:

- local SSE with **overflow**
- instantiation: variant of 2D-1C [ANSS16]

Page-Efficient SSE:

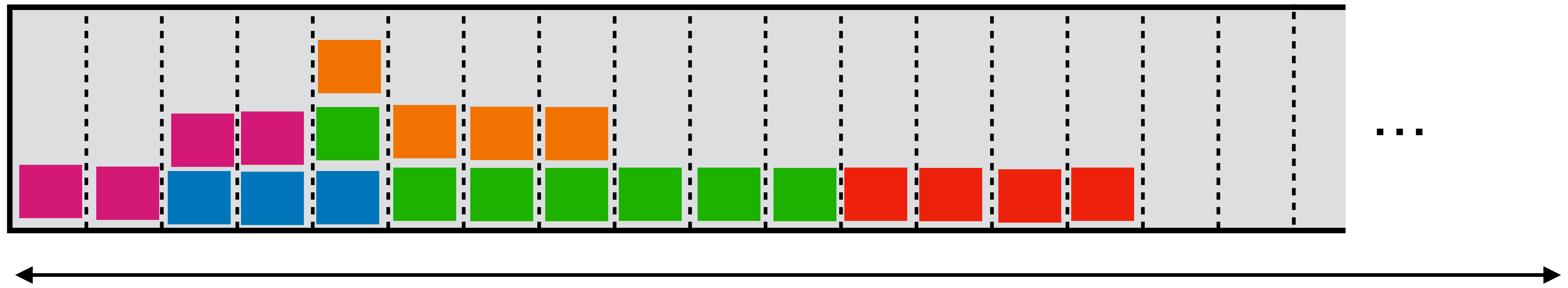
- deals with overflowing items
- instantiation: LayeredSSE



Generic Local Transform

Instantiation

Overflowing SSE: 2D-1C [ANSS16] with cut-off

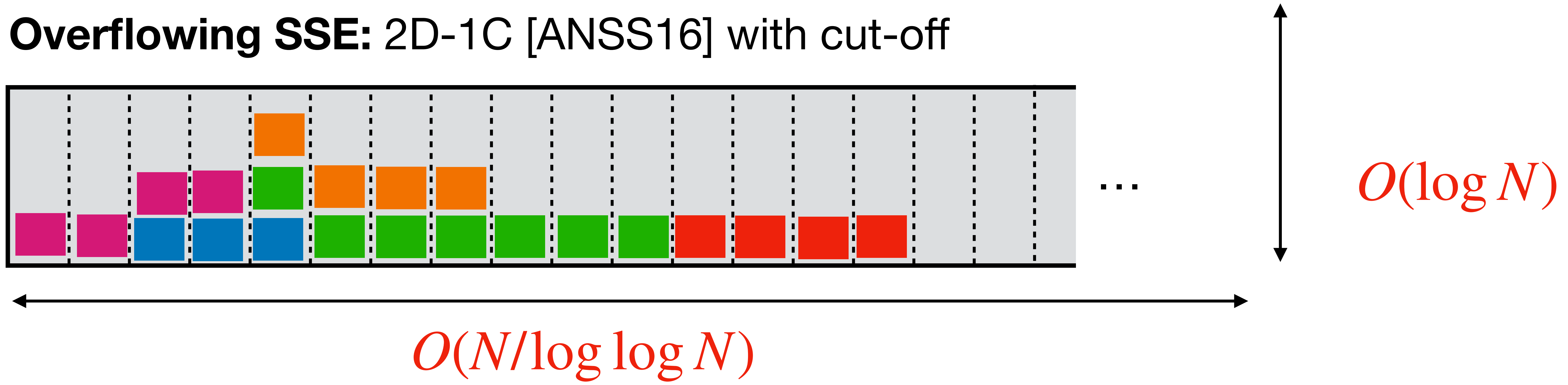


$O(N/\log\log N)$

Generic Local Transform

Instantiation

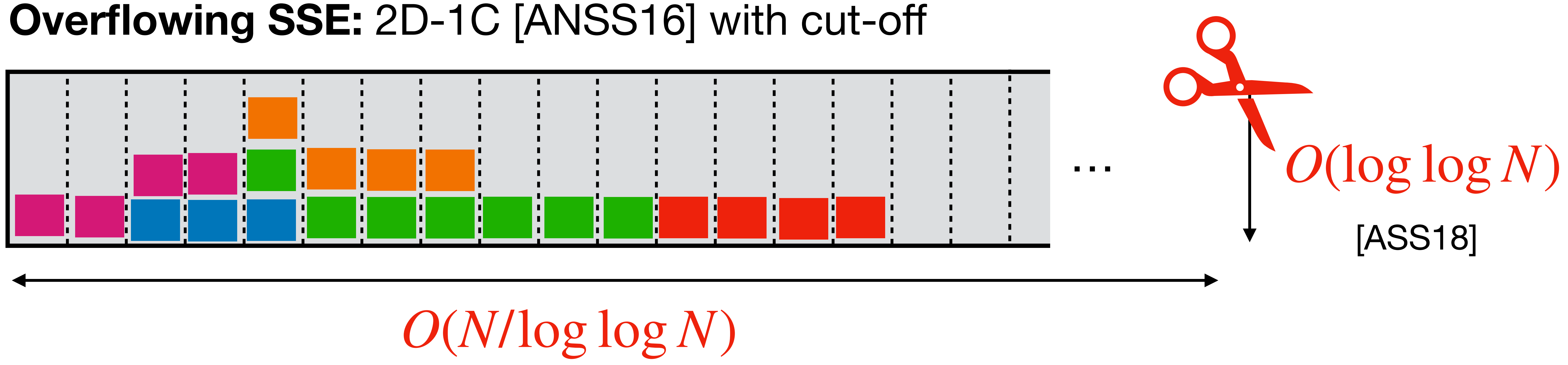
Overflowing SSE: 2D-1C [ANSS16] with cut-off



Generic Local Transform

Instantiation

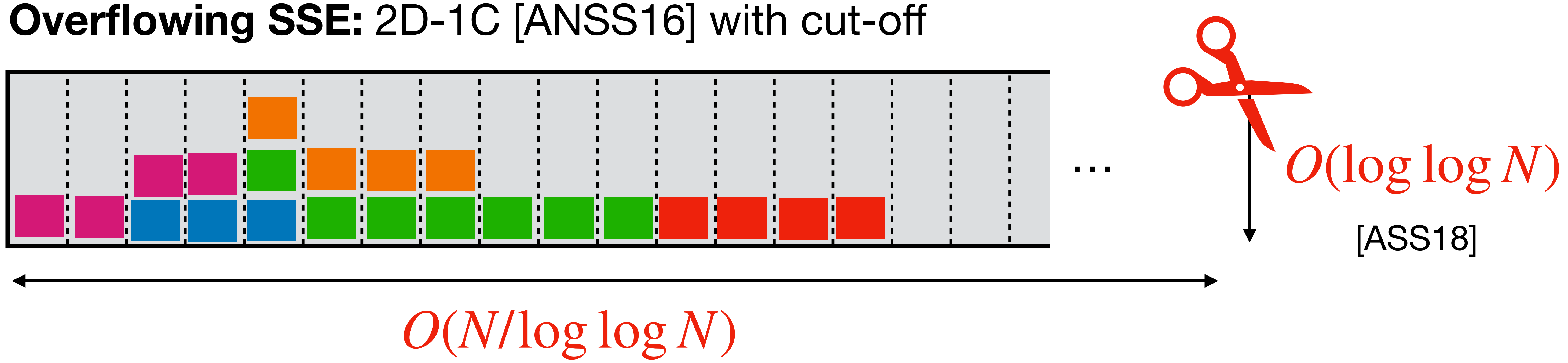
Overflowing SSE: 2D-1C [ANSS16] with cut-off



Generic Local Transform

Instantiation

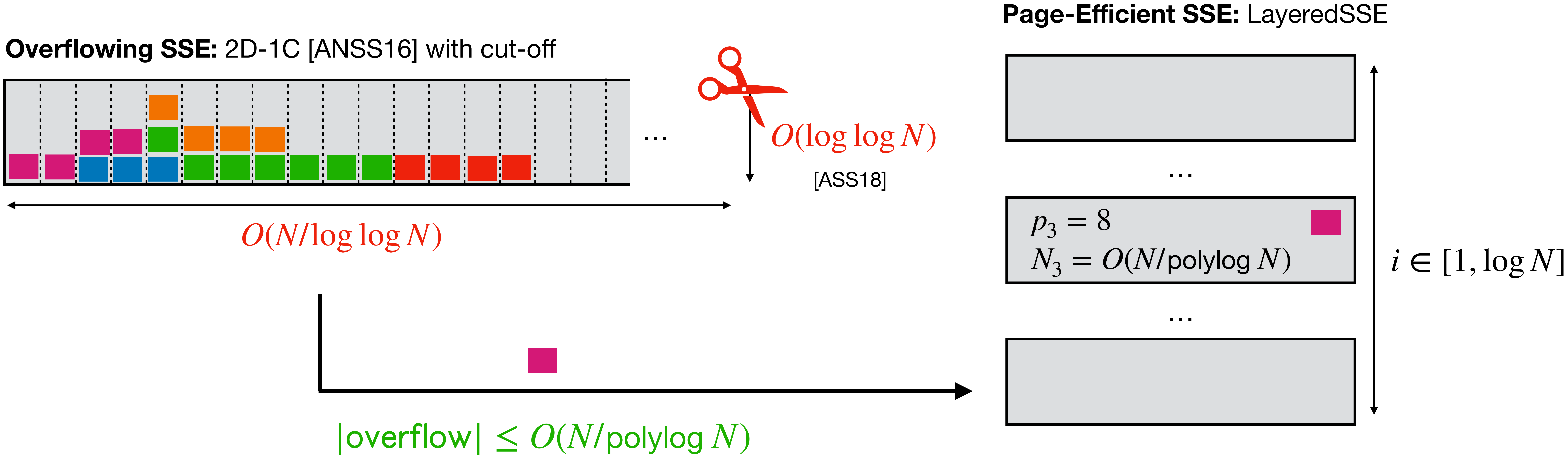
Overflowing SSE: 2D-1C [ANSS16] with cut-off



$|\text{overflow}| \leq O(N/\text{polylog } N)$

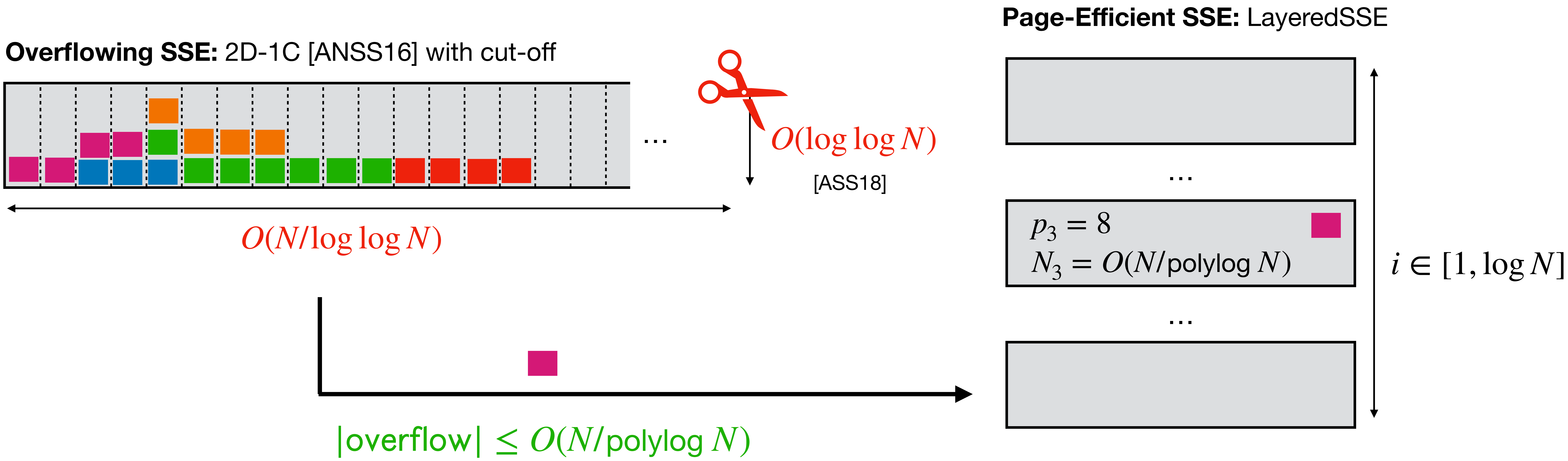
Generic Local Transform

Instantiation


Overflowing SSE: 2D-1C [ANSS16] with cut-off
…
$O(\log \log N)$
[ASS18]
$O(N/\log \log N)$
$|\text{overflow}| \leq O(N/\text{polylog } N)$
Page-Efficient SSE: LayeredSSE
…
$p_3 = 8$
$N_3 = O(N/\text{polylog } N)$
…
$i \in [1, \log N]$

Generic Local Transform

Instantiation



Overflowing SSE: 2D-1C [ANSS16] with cut-off

…

$O(\log \log N)$

[ASS18]

$O(N/\log \log N)$

$|\text{overflow}| \leq O(N/\text{polylog}\, N)$

Page-Efficient SSE: LayeredSSE

…

$p_3 = 8$

$N_3 = O(N/\text{polylog}\, N)$

…

$i \in [1, \log N]$

$\tilde{O}(\log \log N)$ read efficiency *

* restriction on longest list

Unconditional Local SSE

- **Goal:** unconditional SSE
- **State of the art:** $O(\log^{2/3+\varepsilon} N)$ read efficiency [DPP18]

Unconditional Local SSE

- **Goal:** unconditional SSE
- **State of the art:** $O(\log^{2/3+\varepsilon} N)$ read efficiency [DPP18]

Remove bottleneck from [DPP18] via GLT:

1. Generalize the local ORAM of [DPP18]
2. Handle lists with different sizes via different SSE schemes
    - Small, Medium, Large, Huge

Unconditional Local SSE

- **Goal:** unconditional SSE
- **State of the art:** $O(\log^{2/3+\varepsilon} N)$ read efficiency [DPP18]

Remove bottleneck from [DPP18] via GLT:

1. Generalize the local ORAM of [DPP18]
2. Handle lists with different sizes via different SSE schemes
    - Small, Medium, Large, Huge

$O(\log^{\varepsilon} N)$ read efficiency

Recap

- Weighted 2C variant
- First dynamic memory-efficient schemes
- New connection between locality and page efficiency
- Best “unconditional” scheme

Open Problems

- Analysis of “pure” weighted 2C
- Forward secure memory-efficient SSE
- Lower bounds?