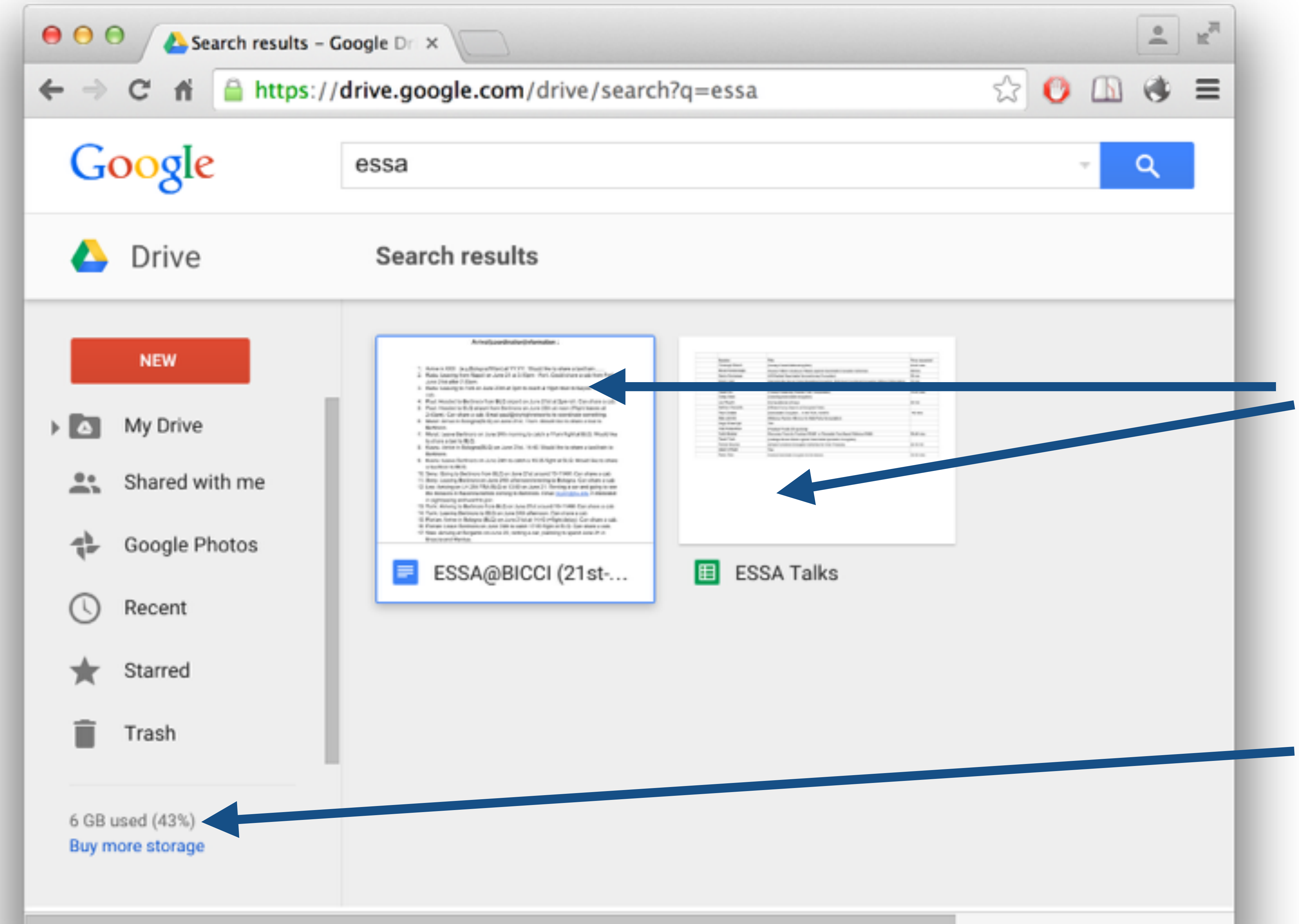
# The Locality of Searchable Symmetric Encryption


David Cash
Rutgers U

Stefano Tessaro
UC Santa Barbara

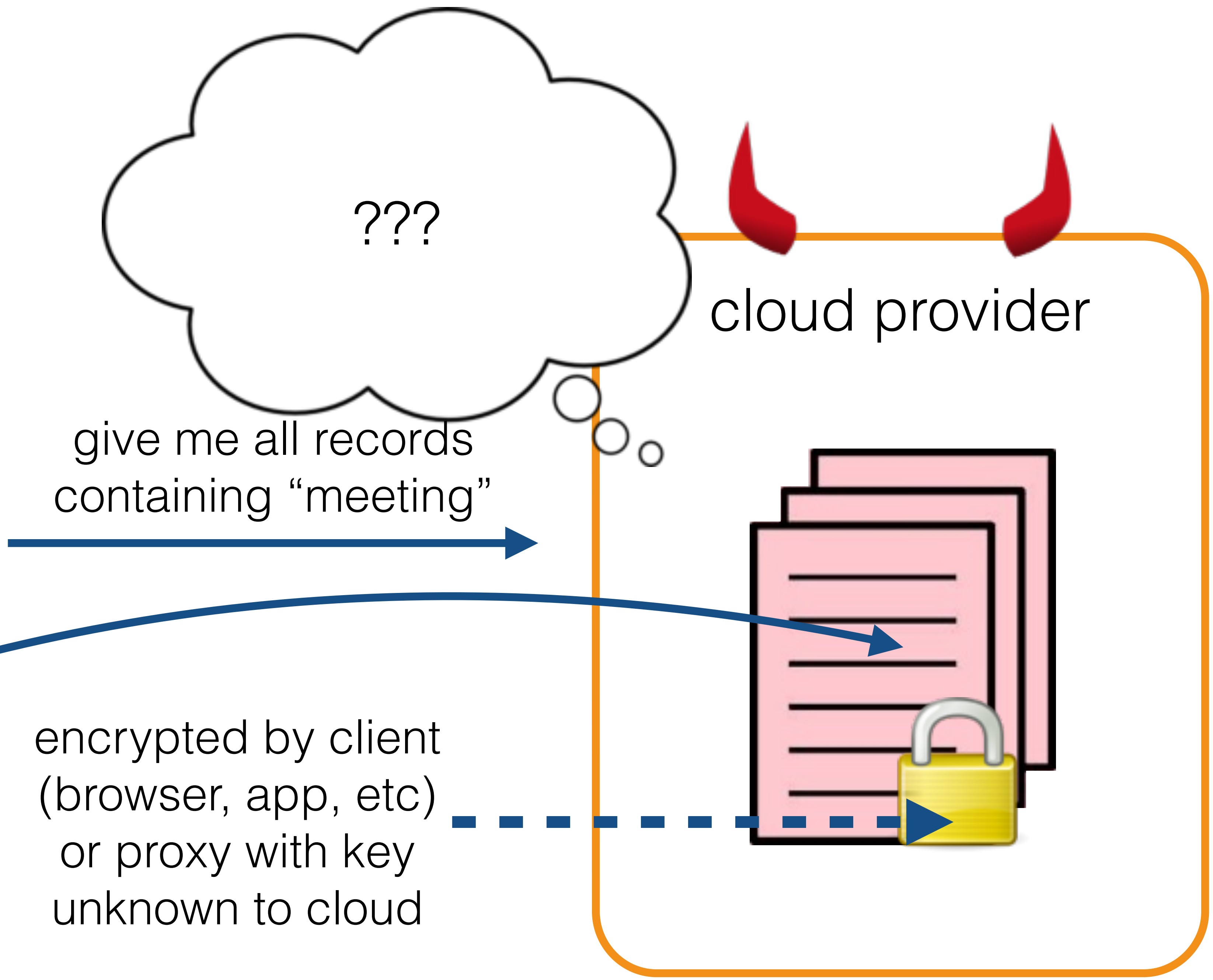
# Outsourced storage and searching




Browser only downloads documents matching query.

Avoids downloading all 6 GB.

# End-to-end encryption and searching




**possible threats:**

- server compromise
- government surveillance
- insider access

- Searching incompatible with privacy goals of traditional encryption

# End-to-end encryption for outsourced storage

box

boxcryptor

SecretSync

tresorit

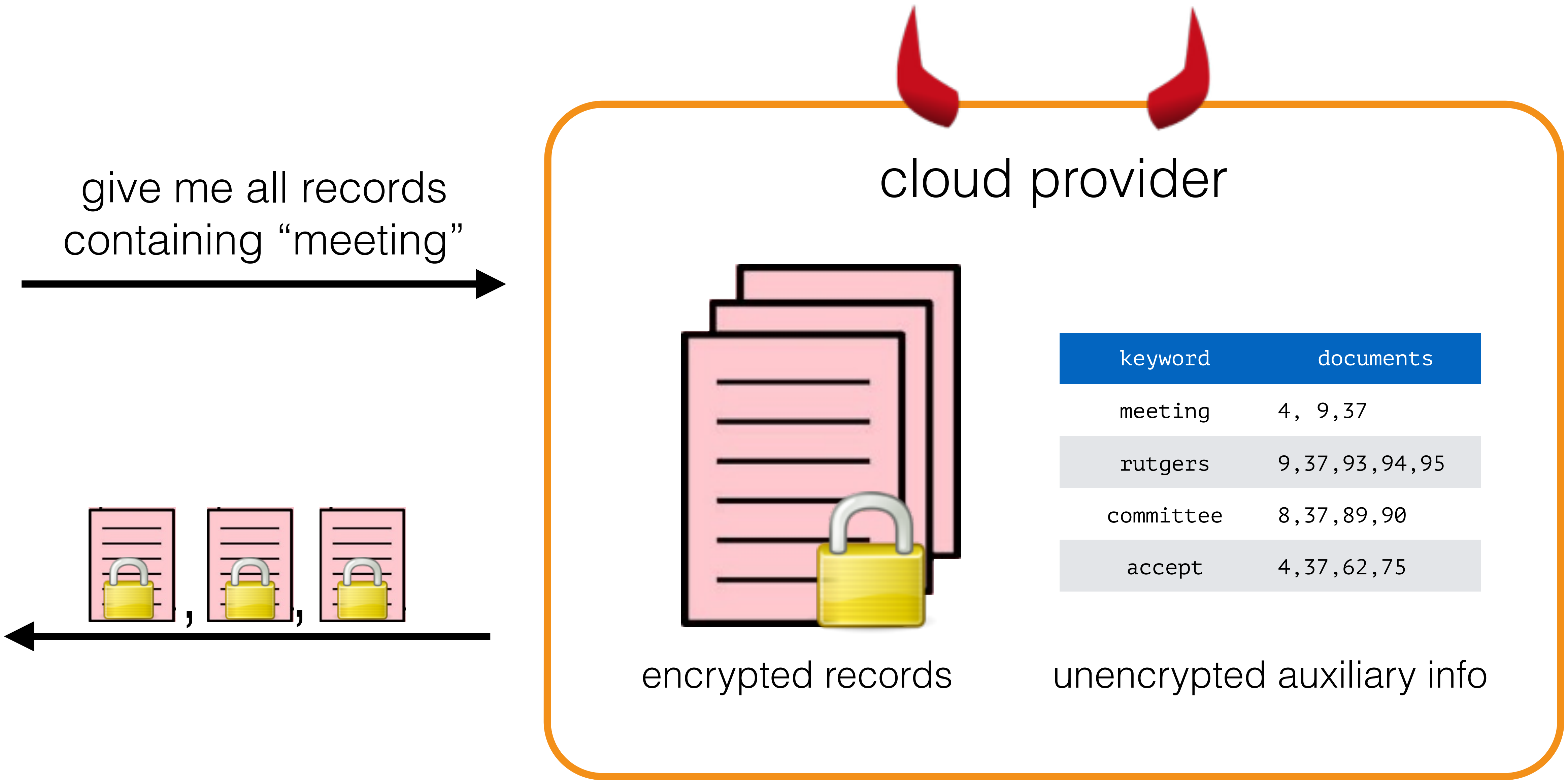
cloudfogger

SPIDEROAK

VIIVO

wuala by LACIE

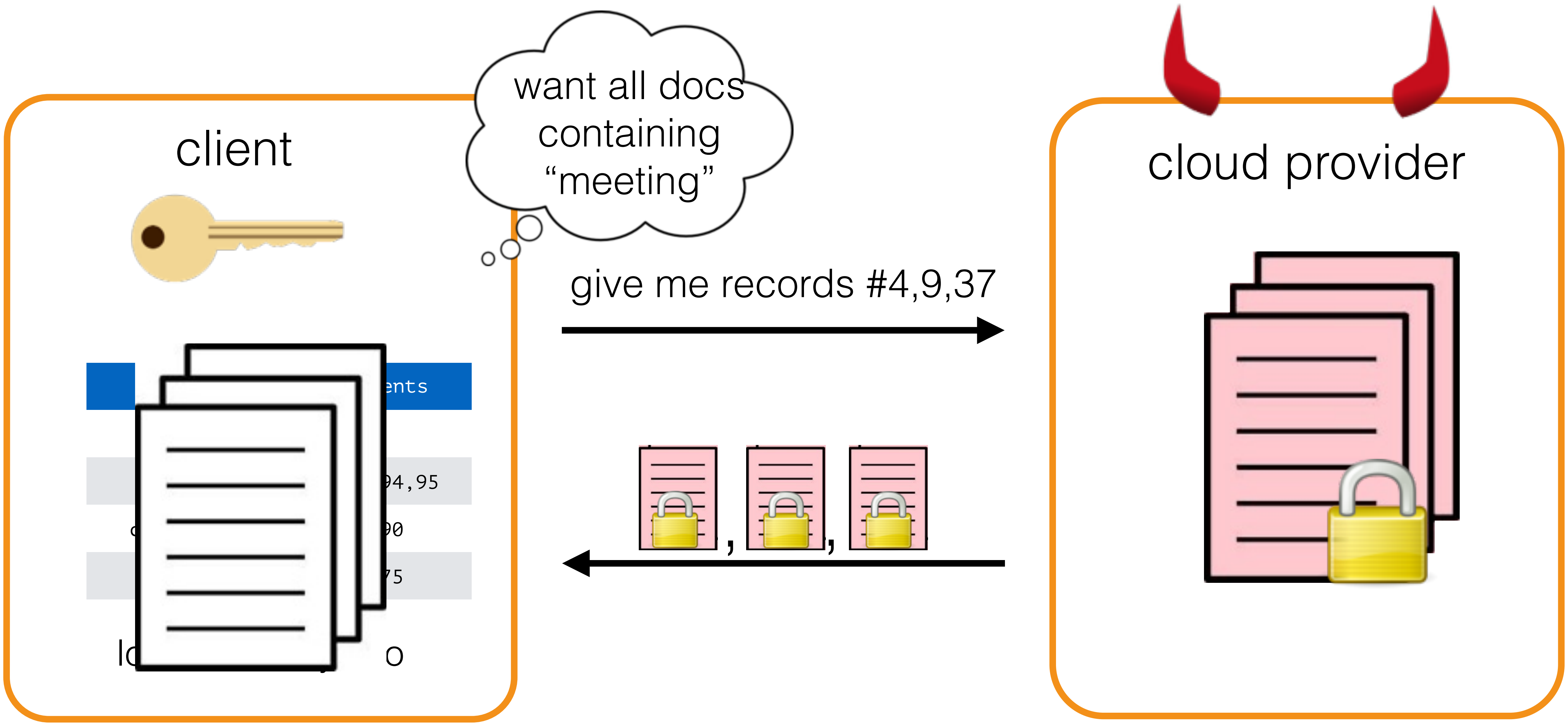
# Search with encryption: possible solution #1




| keyword | documents |
|---|---|
| meeting | 4, 9,37 |
| rutgers | 9,37,93,94,95 |
| committee | 8,37,89,90 |
| accept | 4,37,62,75 |

- unencrypted auxiliary info reveals words in document
- document recovery sometimes possible [Fillmore-Goldberg-Zhu].

# Search with encryption: possible solution #2



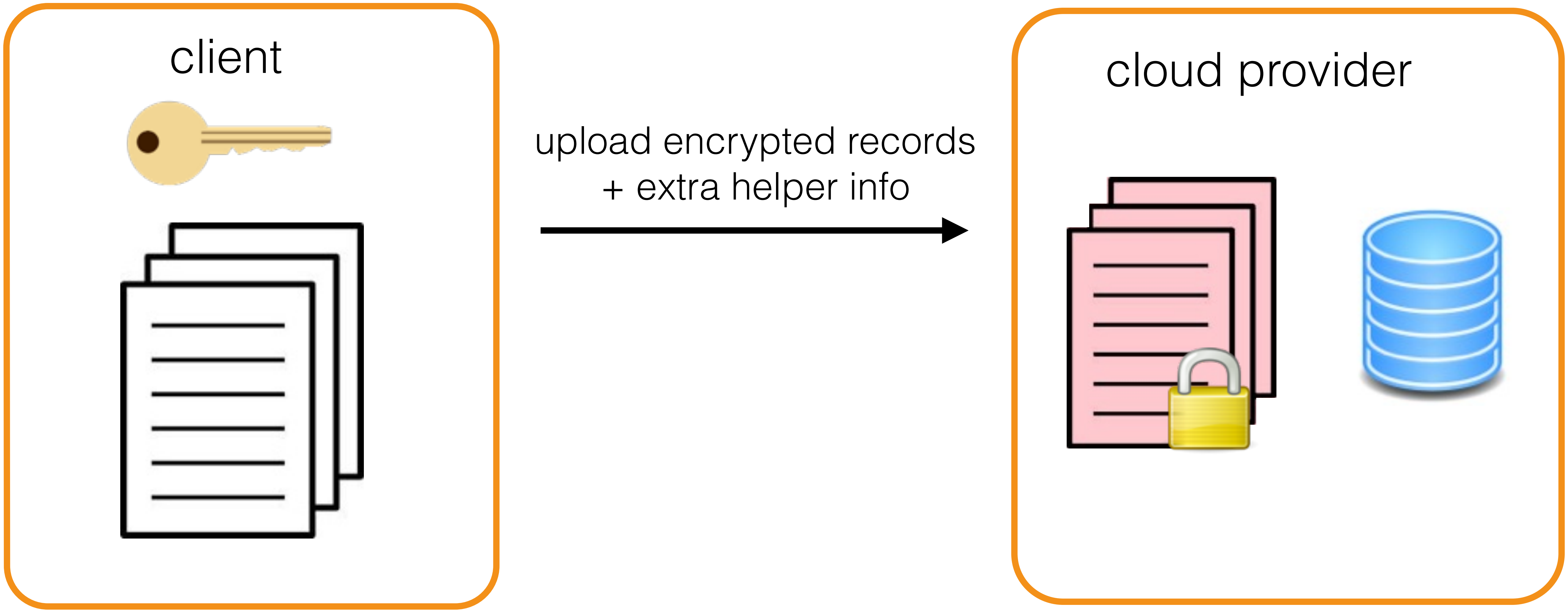

- large state precludes advantages of outsourcing
- even this is not perfect: still leaks "access pattern"

# Searchable encryption: 3 parts

[Song-Wagner-Perrig] , [Curtmola-Garay-Kamara-Ostrovsky], ..

- special protocols to enable provider to “search without decrypting”
- all searching in this talk is for single keywords

1 Encrypted index generation

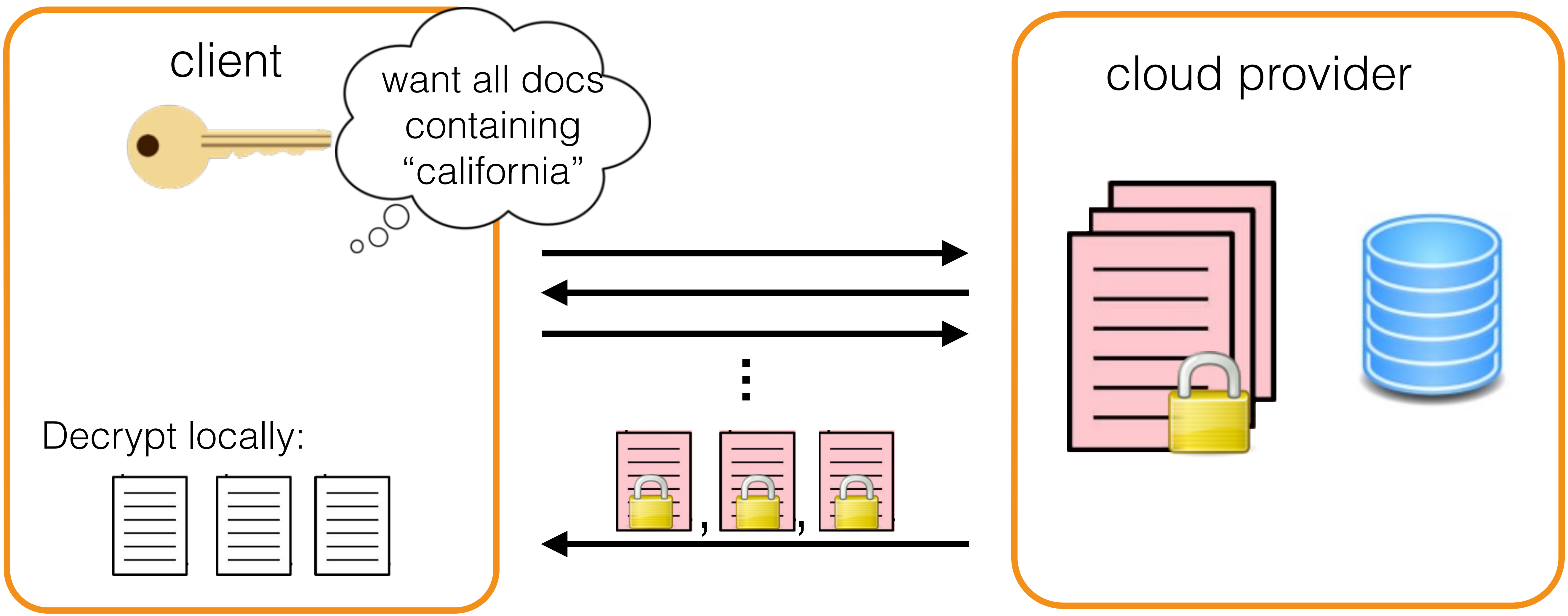


# Searchable encryption: 3 parts [Song-Wagner-Perrig] , [Curtmola-Garay-Kamara-Ostrovsky], . .

- special protocols to enable provider to "search without decrypting"
- all searching in this talk is for single keywords

1 Encrypted index generation

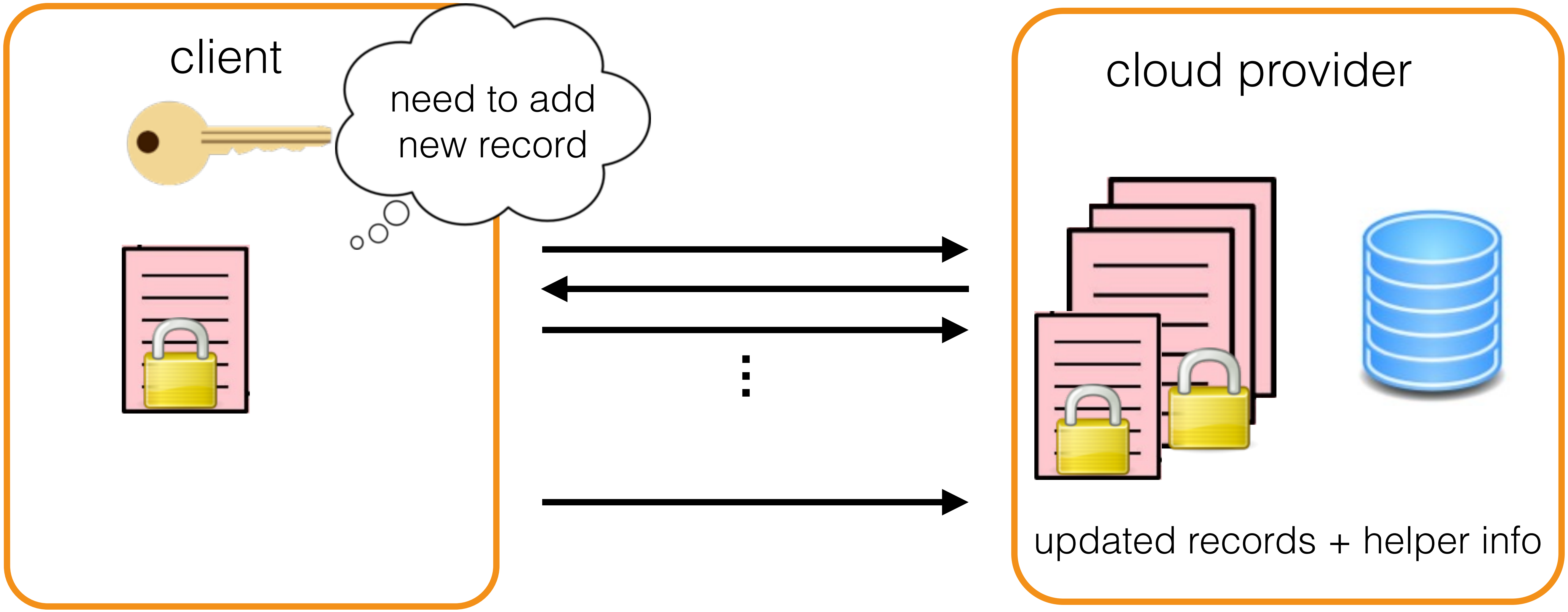
2 Search protocol



# Searchable encryption: 3 parts [Song-Wagner-Perrig], [Curtmola-Garay-Kamara-Ostrovsky], ..

- special protocols to enable provider to "search without decrypting"
- all searching in this talk is for single keywords

1 Encrypted index generation
2 Search protocol
3 Update protocol




- searches should still "work" on added record

# Example searchable encryption

## 1 Encrypted index generation

processing

Inverted index:

| keyword | records |
| --- | --- |
| sunnyvale | 4, 9,37 |
| rutgers | 9,37,93,94,95 |
| committee | 8,37,89,90 |
| accept | 4,37,62,75 |

Encrypted index:

| keyword | records |
| --- | --- |
| 45e8a | 4, 9,37 |
| 092ff | 9,37,93,94,95 |
| f61b5 | 8,37,89,90 |
| cc562 | 4,37,62,75 |

1. Replace each keyword with “keyed hash” (i.e., PRF) of keyword: **H(K,w)**
2. Client saves key **K**

## 2 Search protocol

1. Client sends: **H(K,w)**
2. Server retrieves proper row

## 3 Update protocol

- To add new record, client identifies which rows to add new identifier to

# Example of searchable encryption (strengthened)

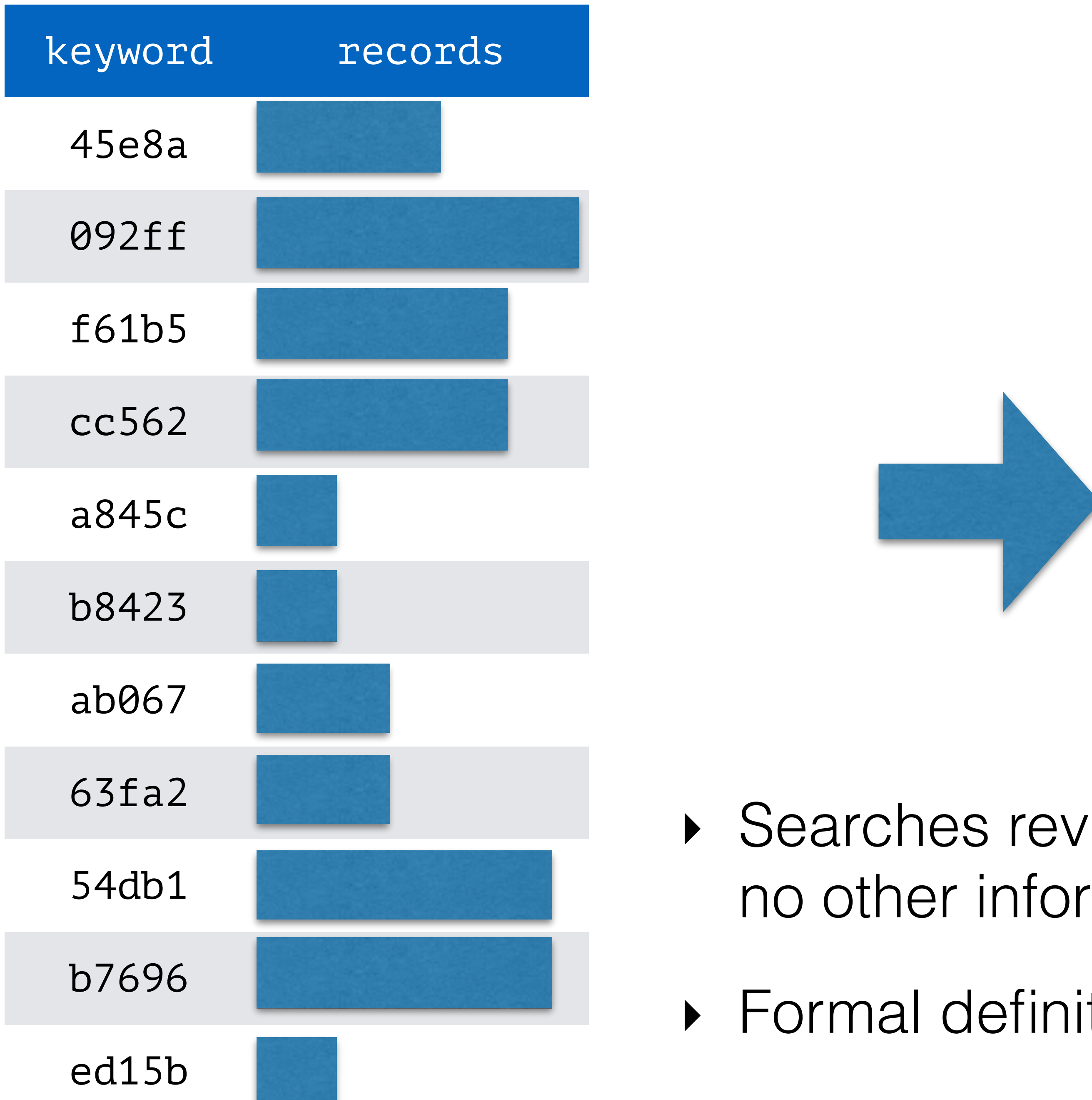
- additionally encrypt rows under different keys
- requires modification of server, but more secure

| keyword | records |
|---|---|
| 45e8a | 4, 9,37 |
| 092ff | 9,37,93,94,95 |
| f61b5 | 8,37,89,90 |
| cc562 | 4,37,62,75 |

| keyword | records |
|---|---|
| 45e8a | |
| 092ff | |
| f61b5 | |
| cc562 | |

# In this talk: Also hide lengths and number of rows

[Curtmola-Garay-Kamara-Ostrovsky], …




```
nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD
```

- Searches reveal intended results but leak no other information
- Formal definition omitted
- Simple construction later

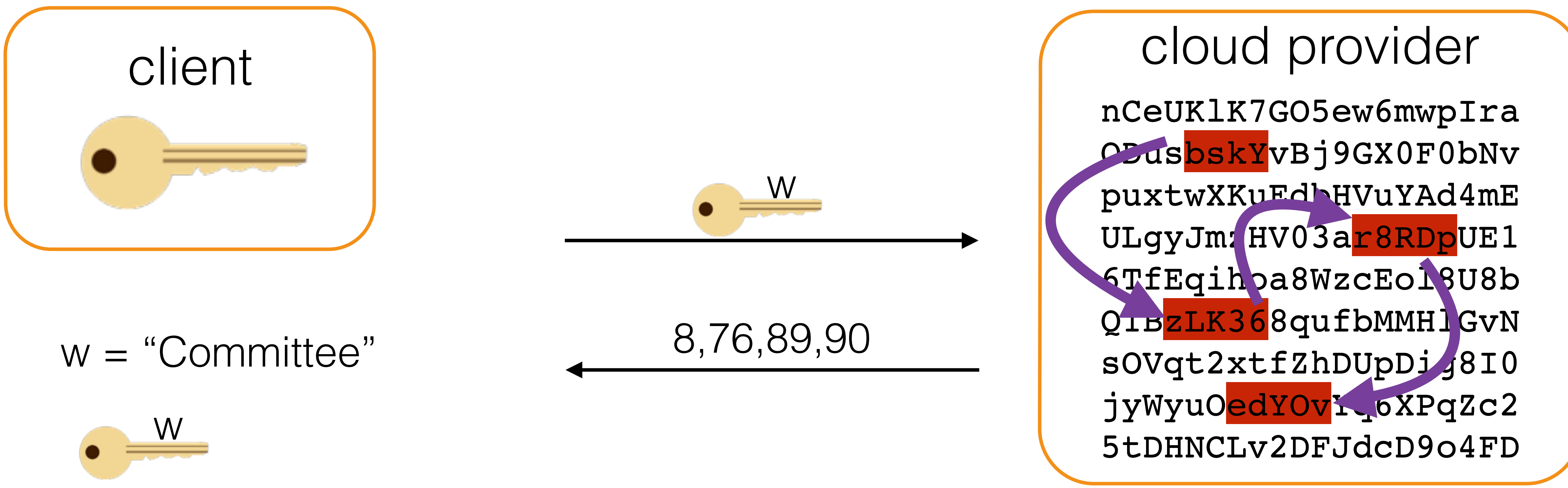
# Performance Bottleneck

systems collaborators and others have complained:

> “ Fine, the asymptotics are optimal, but this stuff is unusably slow for large indexes.

➡ Runtime bottleneck: disk latency, not crypto processing.

# Memory access during encrypted search




➡ constructions access one random part of memory per posting

- one disk seek per posting (≈ only a few bytes, wasteful)

➡ plaintext search can use one contiguous access for entire postings list

# I/O theory (not IO theory)

- count *only* # of blocks moved to/from disk [Aggarwal-Vitter]
  - idea: i/o time overwhelms time for computation
- numerous versions of theory i/o models (see [Vitter] text)
  - optimal results (matching upper/lower bounds) for many problems like sorting, dictionary look-up, …

# Our results: I/O efficiency and searchable encryption

[C., Tessaro'14]

➡ Study I/O efficiency and security

➡ Unconditional I/O lower bounds for searchable encryption

- new proof technique

➡ Construction improving I/O efficiency of prior work

# Our results: I/O efficiency lower bound

"Theorem": Secure searchable encryption must either:

(1) Have a very large encrypted index,

or

(2) Read memory in a highly "non-local" fashion,

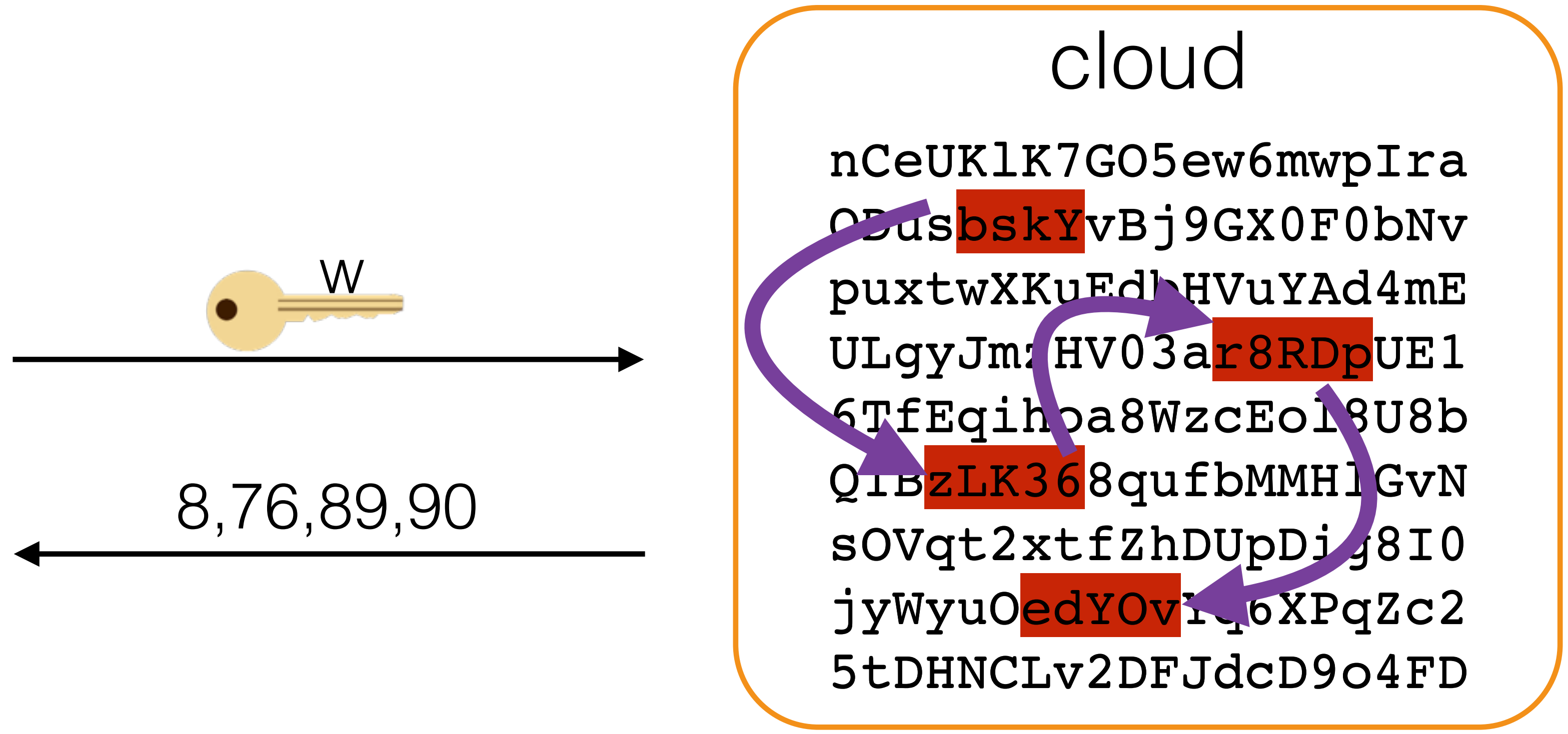
or

(3) Read more memory than a plaintext search.

- ➡ unconditional (no complexity assumptions)
- ➡ applies to any scheme (no assumption about how it works)
- ➡ different type of i/o lower bound: security vs. correctness

# Memory utilization in searching

**Any** construction can be seen as “touching” contiguous regions of memory during search processing:



# Memory utilization in searching

We use three (very coarse) measures:

1. encrypted index size: measured relative to #-postings

N postings total → f(N) bits

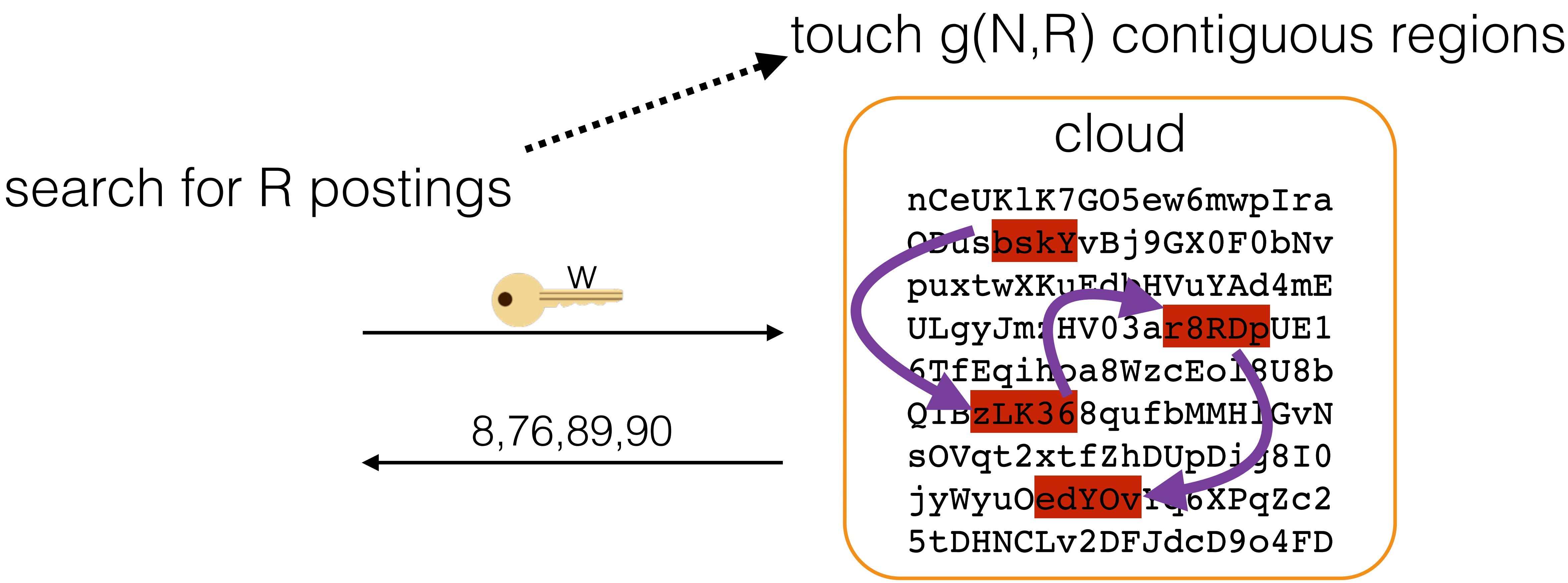
| term | postings |
|---|---|
| "Rutgers" | 4,9,37 |
| "Admissions" | 9,37,93,94,95,96 |
| "Committee" | 8,37,93,94 |
| "Accept" | 2,37,62,75 |


nCeUKlK7GO5ew6mwpIra
ODusbskYvBj9GX0F0bNv
puxtwXKuEdbHVuYAd4mE
ULgyJmzHV03ar8RDpUE1
6TfEqihoa8WzcEol8U8b
Q1BzLK368qufbMMHlGvN
sOVqt2xtfZhDUpDig8I0
jyWyuOedYOvYq6XPqZc2
5tDHNCLv2DFJdcD9o4FD

# Memory utilization in searching

We use three (very coarse) measures:

1. encrypted index size: measured relative to #-postings
2. locality: number of contiguous regions touched



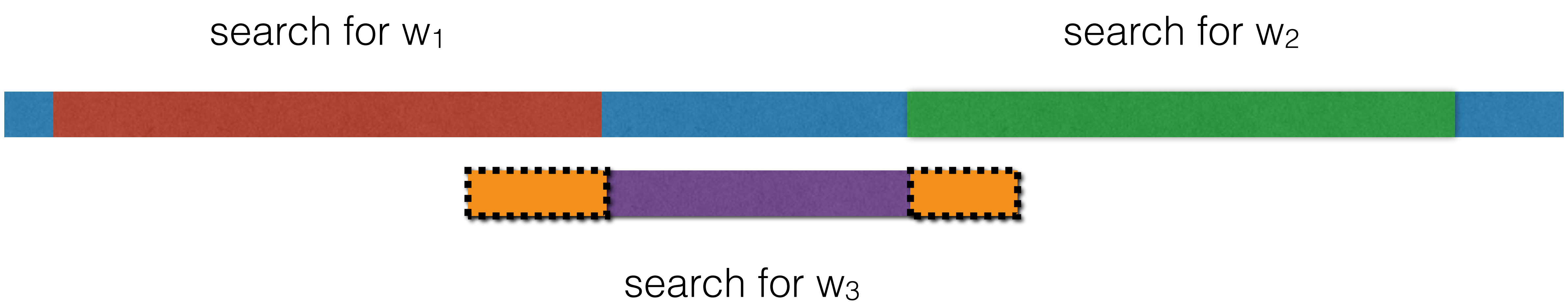
# Memory utilization in searching

We use three (very coarse) measures:

1. encrypted index size: measured relative to #-postings
2. locality: number of contiguous regions touched
3. read overlaps: amount of touched memory common between searches

# Read overlaps

Encrypted index in memory:




Overlap of search for $w_3$ = size of orange regions

- ➡ $\mathbf{h}$-overlap $\Longrightarrow$ any search touches $\leqslant \mathbf{h}$ bits touched by any other possible search
- ➡ **intuition**: large overlaps ≈ reading more bits than necessary
- ➡ small overlap in known constructions (e.g. hash table access)

# Our results: lower bound (formal)

Let N = no. postings in input index

---

Theorem: No length-hiding scheme can have all 3:

1. O(N)-size encrypted index
2. O(1)-locality
3. O(1)-overlap on searches

---

➡ super-linear blow-up in storage/locality or highly overlapping reads

➡ in paper: smooth trade-off

✳ can be circumvented by tweaking security def [CJJJKRS]

# Memory utilization of constructions

$N$ = no. postings in input index, $R$ = no. postings in search

| | Enc Ind Size | Overlap | Locality |
|---|---|---|---|
| lower bound: 1 of | $\omega(N)$ | $\omega(1)$ | $\omega(1)$ |
| [CGKO,KPR,…] | $N$ | $1$ | $R$ |
| [CK] | $N^2$ | $1$ | $1$ |
| trivial "read all" | $N$ | $N$ | $1$ |
| new construction | $N \log N$ | $\log N$ | $\log N$ |

➡ open problem: get closer to lower bound

# Outline

- prior constructions and why they can't be "localized"

- lower bound approach

# Outline

**- prior constructions and why they can't be "localized"**

- lower bound approach

# [CGKO] construction

## Encrypted Index Generation Step 1:

- derive per-term encryption keys: $K_i = \mathrm{PRF}(w_i)$

- encrypt individual postings under respective keys

| term | postings |
|---|---|
| Columbia | 4, 9,37 |
| Big | 9,37,93,94,95 |
| Data | 8,37,89,90 |
| Workshop | 4,37,62,75 |

| term | postings |
|---|---|
| Columbia | |
| Big | |
| Data | |
| Workshop | |

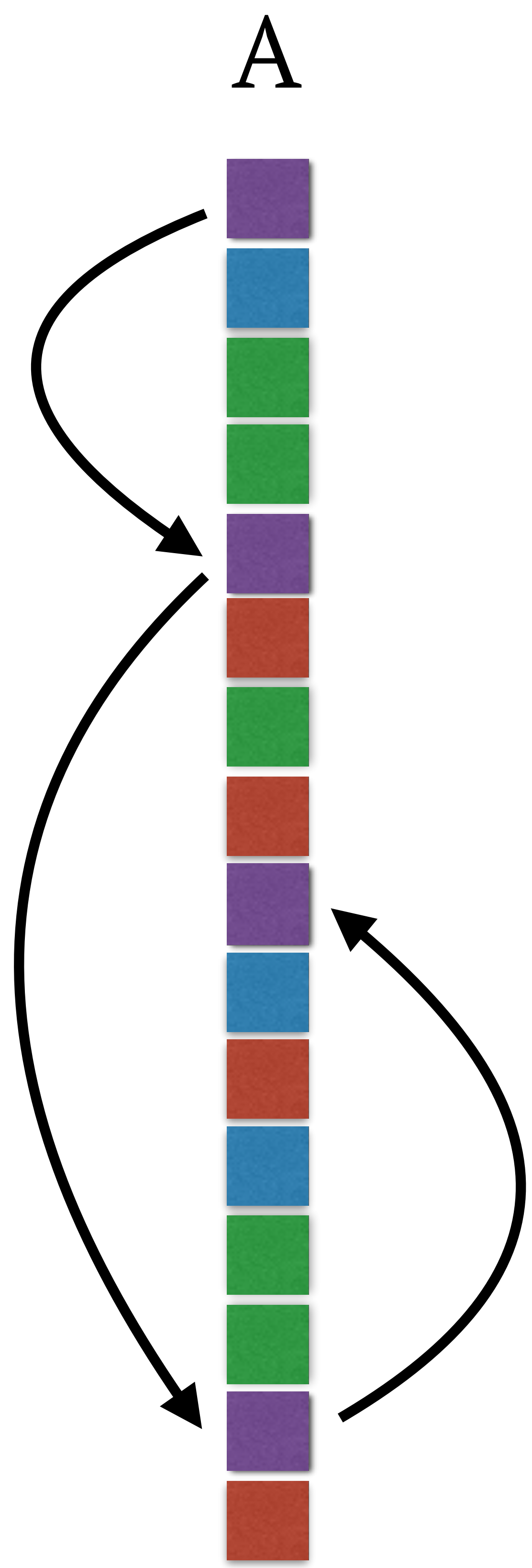
# [CGKO] construction: searching

Encrypted Index Generation Step 2:

1. put ciphertexts in random order in array $\mathrm{A}$

2. link together postings lists with encrypted pointers (encrypted under $\mathrm{K_i}$)

3. encrypted index = $\mathrm{A}$




(example with pointers for word “Workshop”)

# [CGKO] construction: searching

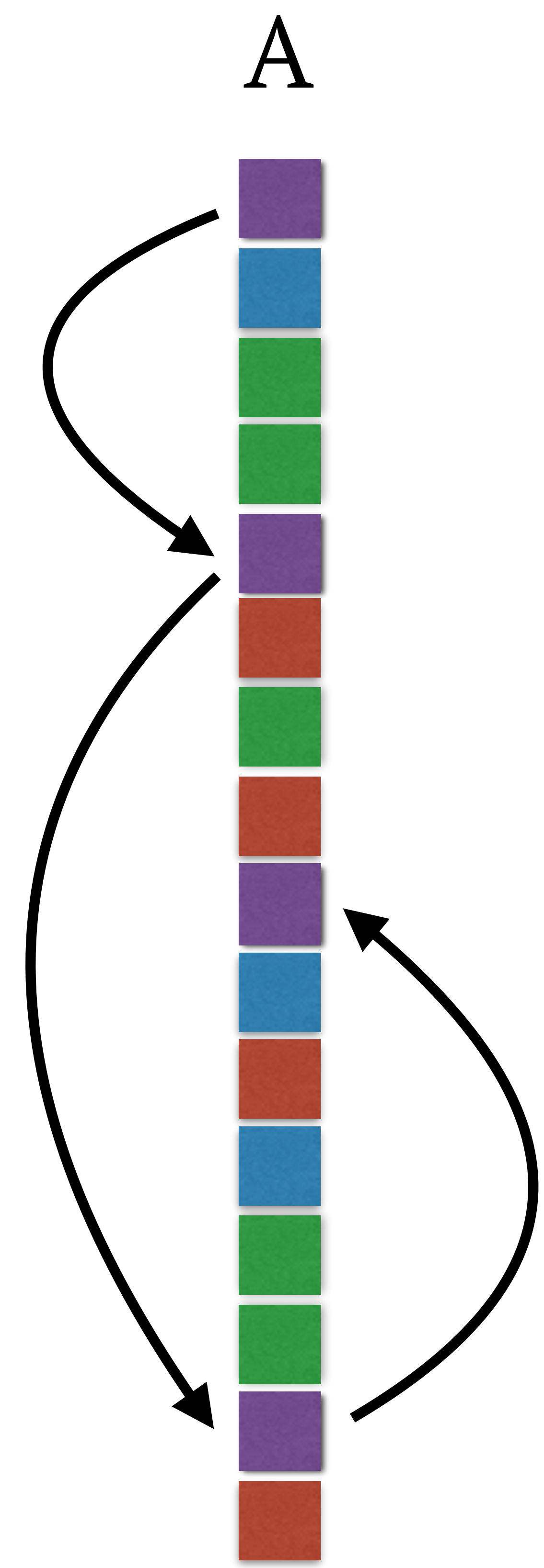
search token generation for $w$:

- re-derive key $K = \mathrm{PRF}(w)$
- token = $K$

---

server search using token:

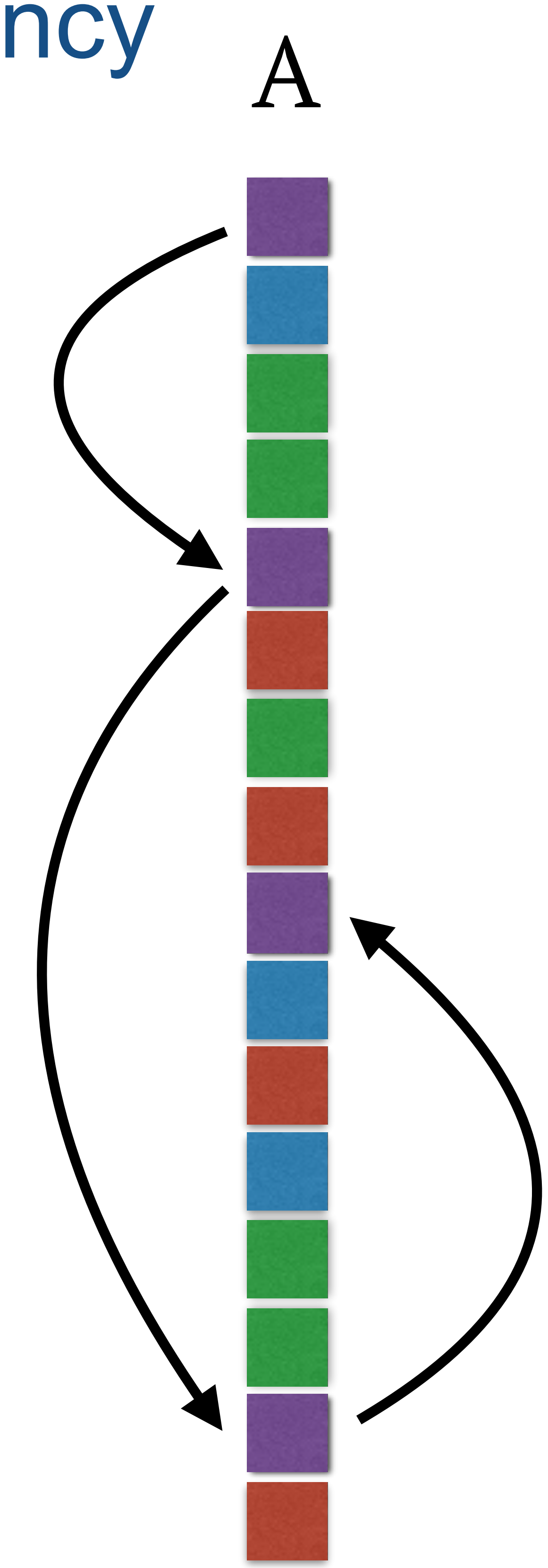
- step through list, decrypt postings/ pointers with $K$



# [CGKO] construction: memory efficiency

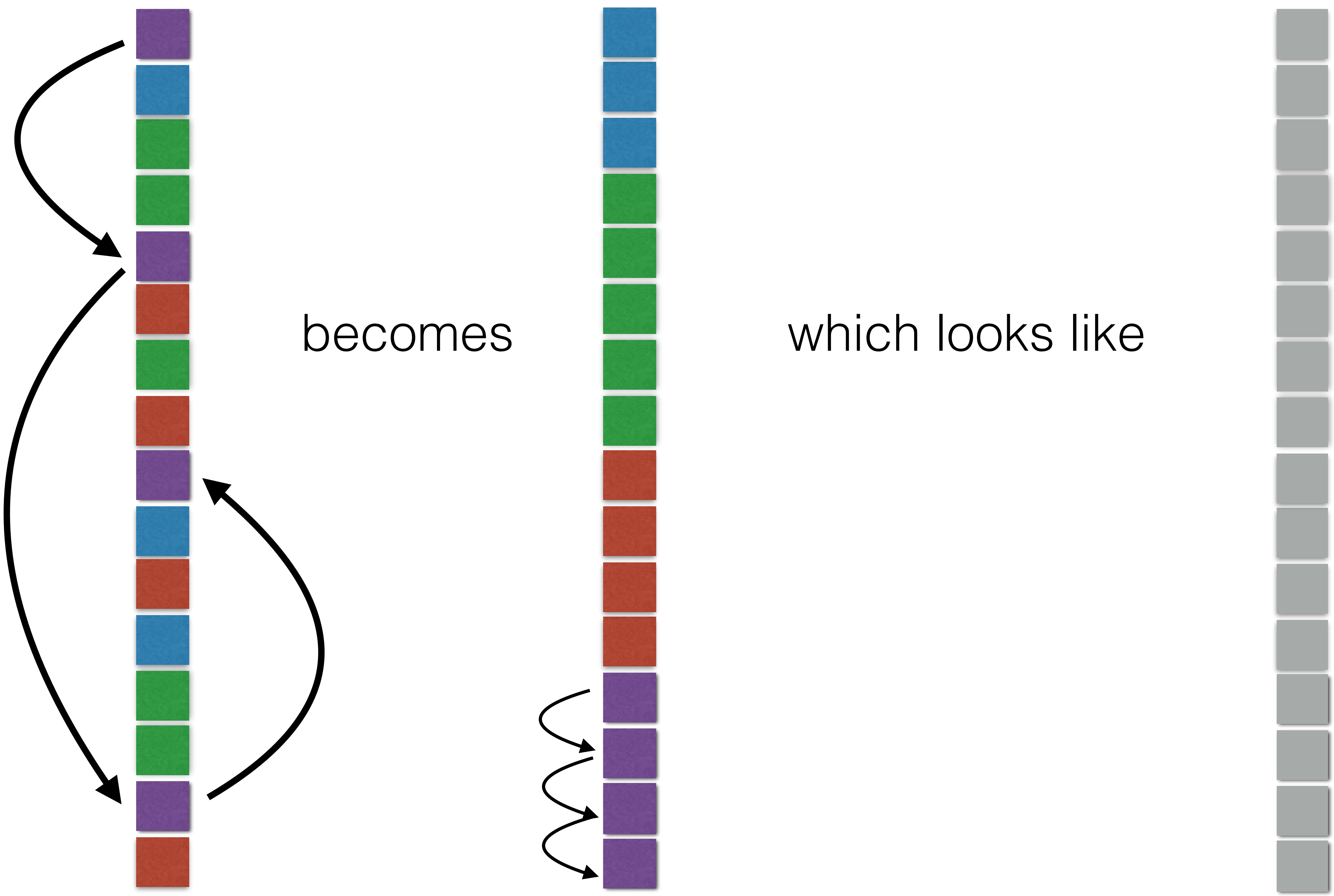
Memory utilization:

- O(N) size index
- O(R) locality for search w/ R postings
- O(1) read overlaps

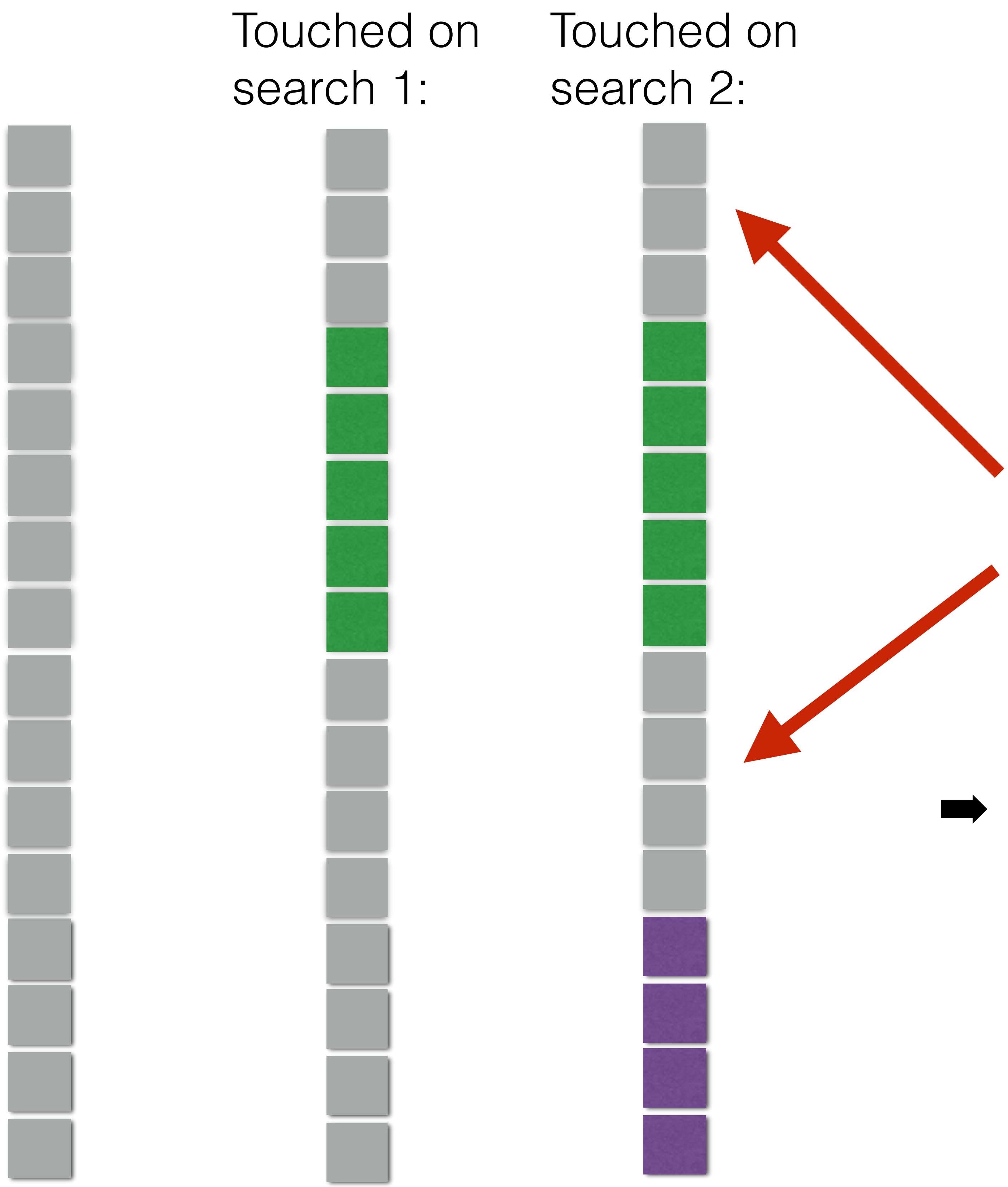


suppose we try to make construction "local"

➡ store encrypted postings lists together.



server can observe memory touched during searches:




composition of untouched regions reveals info about unopened part of index!

➡ e.g. 7 remaining spots do not correspond to a single postings list

# Our Lower Bound (recall)

Let $N$ = no. postings in input index

---

Theorem: No secure searchable encryption can have all 3:

1. $O(N)$-size encrypted index
2. $O(1)$ locality
3. $O(1)$-overlaps between searches

---

➡ proof approach: suppose construction satisfies all 3. then we find an attack

➡ attack looks at where server touches memory, infers info about index

# Warm up: Special Case

we'll show no secure scheme can have all 3:

(1) $<1.5\mathrm{x}$-size encrypted index over plaintext index

(2) exactly $1$-locality (i.e. reads one contiguous region)

(3) $0$-overlaps (i.e. disjoint reads for searches)

➡ "perfectly local construction that reads one region for exactly number of bits needed must double index size"

➡ in paper:

- improve (1) from "double" to "any constant factor" via delicate argument
- improve (2) and (3) via minor tweaks to argument

- We distinguish these two indices:

Index $I_0$

| term | records |
| --- | --- |
| w | p |
| w | p |
| w | p |
| ⋮ | ⋮ |
| w | p |

Index $I_1$

| term | records |
| --- | --- |
| w | p |
| w | p |

❇ terms/identifiers all random strings

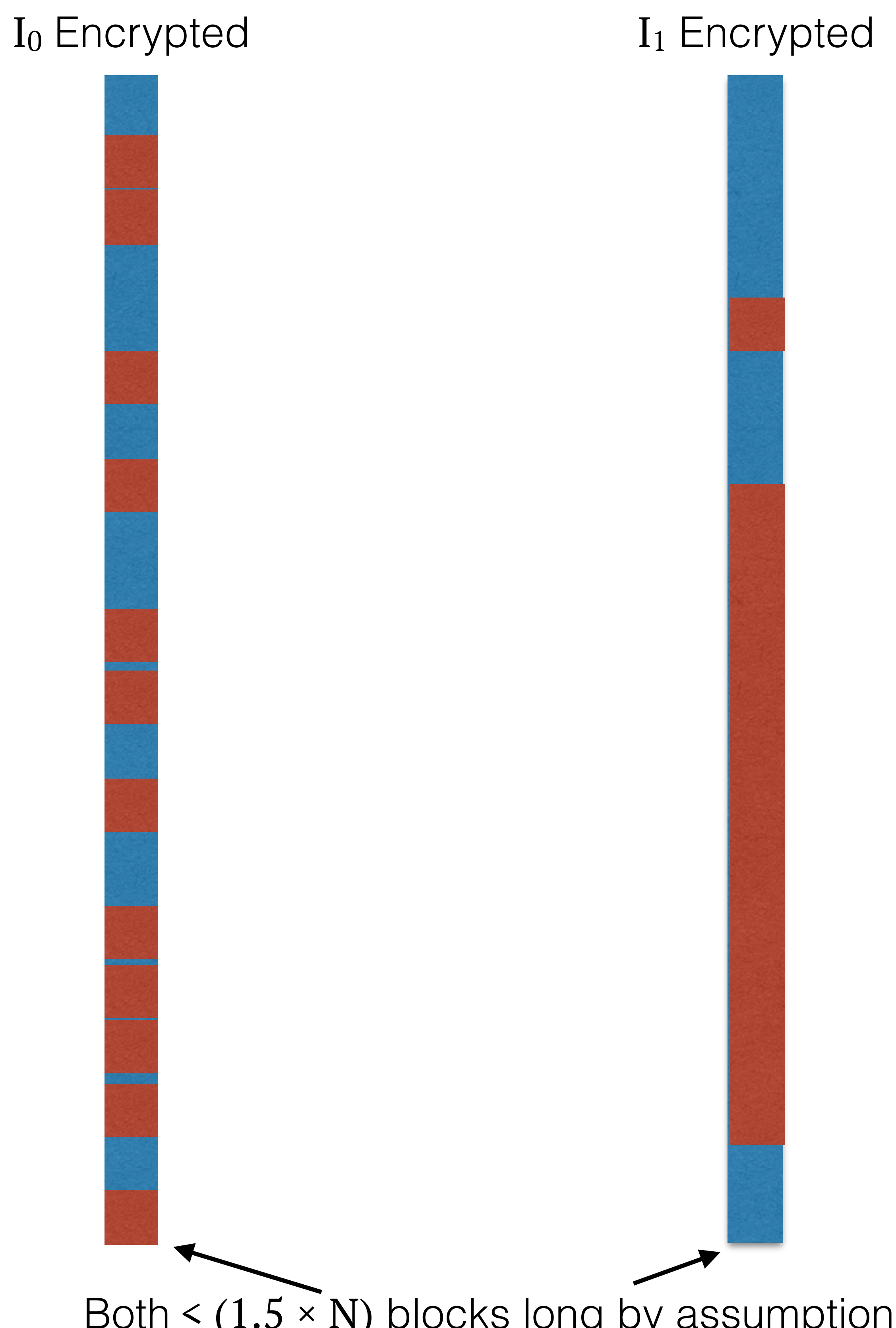
- Examine which region of memory is read when searching for $w_1$

# Attack Intuition

Red regions: Regions that *would be touched* during a search for each keyword

By assumptions:

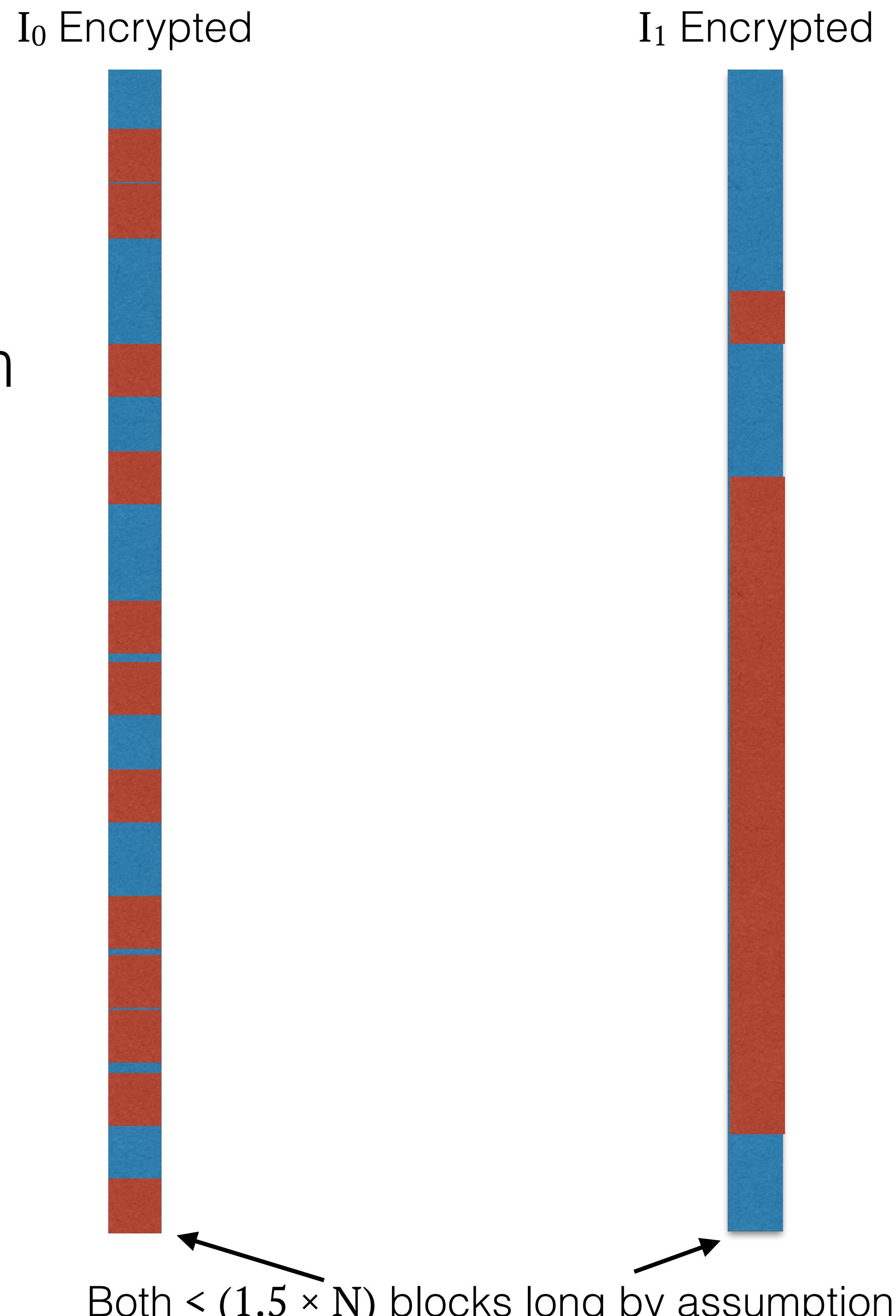
- ➡ **If $I_0$ encrypted**, then N small regions
- ➡ **If $I_1$ encrypted**, then one small region and one huge region



# Attack Intuition

Consider region touched when searching for $w_1$:

- ➡ **If $I_0$ encrypted**, then random small region touched
- ➡ **If $I_1$ encrypted**, then fixed small region touched

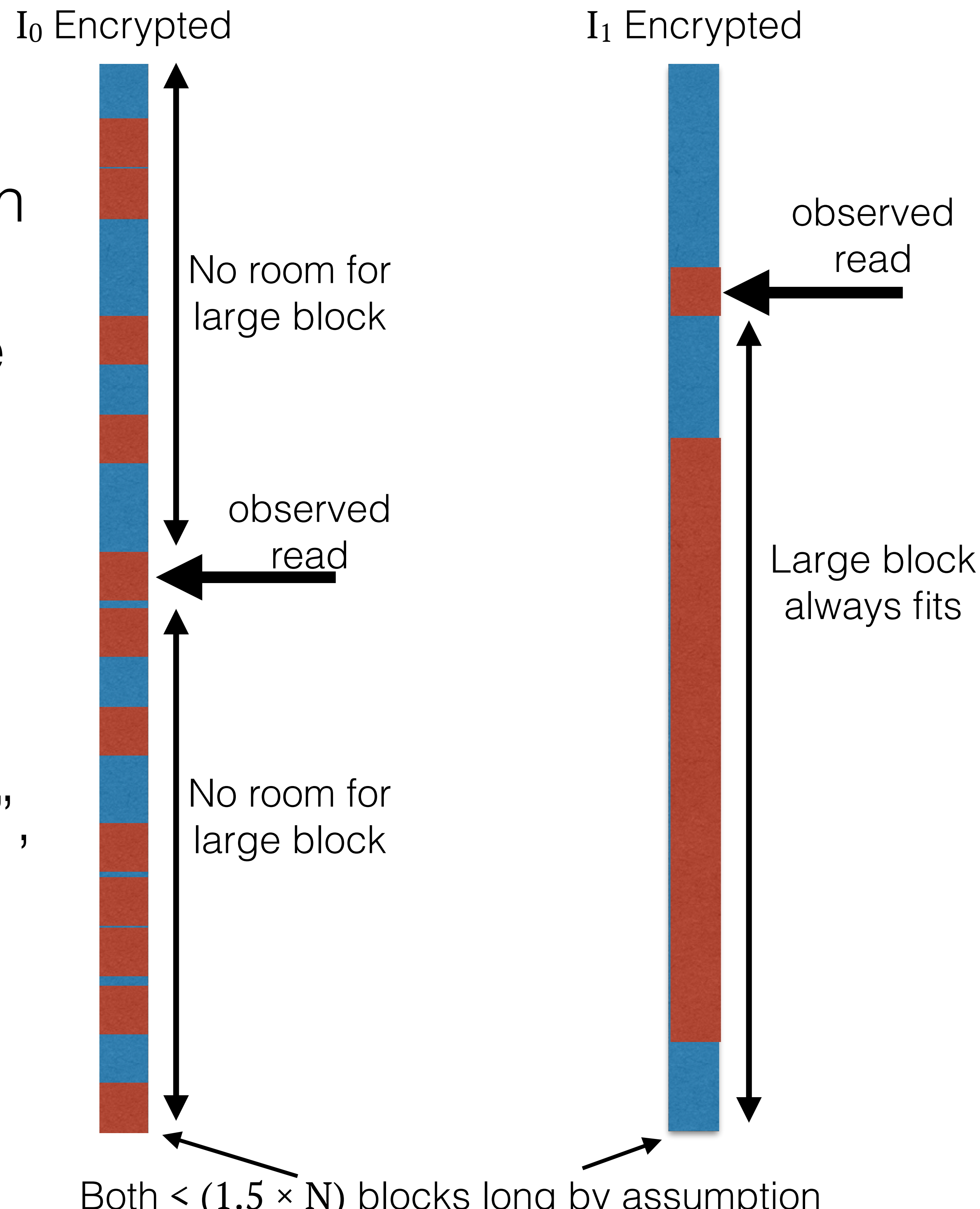


# Attack Intuition

Two observations:

1. If $I_1$ encrypted, touched region must leave large contiguous untouched region on one side
2. If $I_0$ encrypted, $\geqslant 1/N$ chance this does not happen
   - Proof by pigeonhole: $< 1.5N$ places to store $N$ blocks, so one must be "close to center", preventing large block fitting

➡ We check if large block could fit, decides which index was encrypted

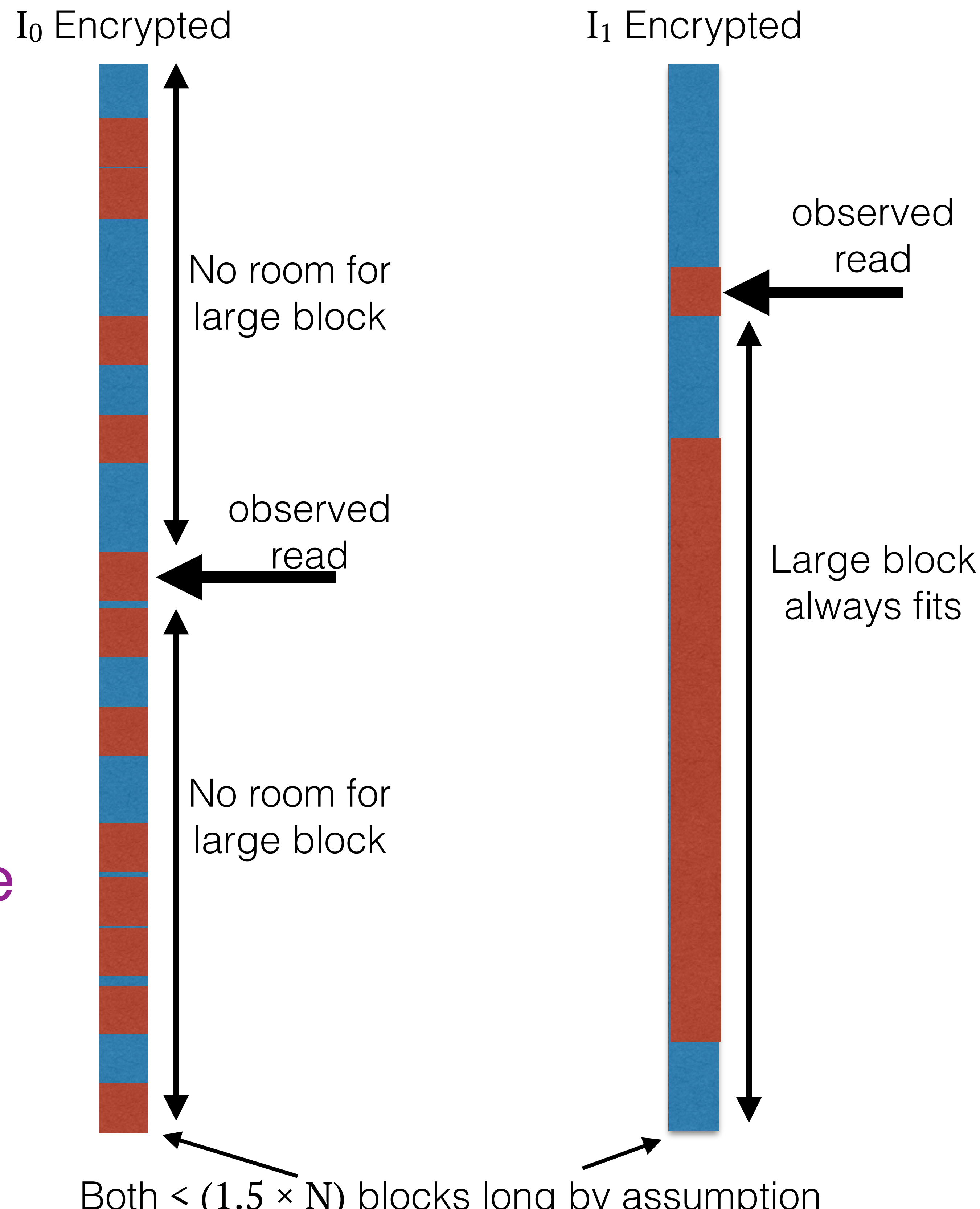


# Attack Intuition

very weak bound so far:

- does not apply if server can read two regions
- does not apply if encrypted index can be slightly larger
- does not apply if tiny amount of overlap allowed

Now:  first deal with larger index (factor k instead of 2), still assume perfect locality



# Stronger Attack Intuition

# Stronger Attack Intuition


➡ We ask to search terms $w_1, \ldots, w_{10}$

# Stronger Attack Intuition


➡ We ask to search terms $w_1, \ldots, w_{10}$

- $I_1$ encrypted $\Longrightarrow$ observe huge contiguous untouched region

# Stronger Attack Intuition

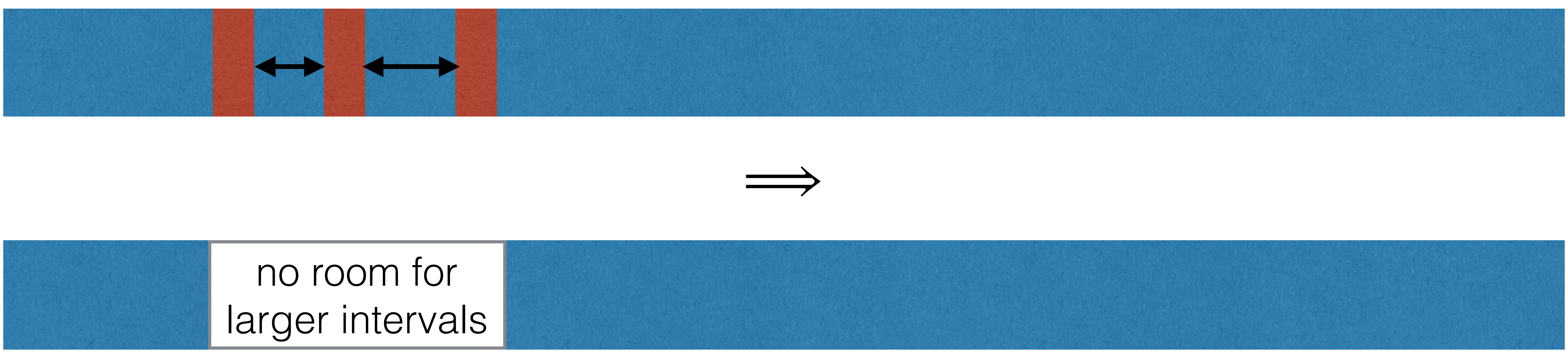

➡ We ask to search terms $w_1, \ldots, w_{10}$

- $I_1$ encrypted $\Longrightarrow$ observe huge contiguous untouched region
- $I_0$ encrypted $\Longrightarrow$ no such region with constant probability

# Tools for the Attack

Exploit simple combinatorics of gaps between random intervals:

- **Lemma 1**: If scheme secure, then memory touched during a O(1)-local search satisfies a mild pseudorandomness condition
- **Lemma 2**: Pseudorandom reads will have “many” small gaps between contiguous regions with constant probability.




➡ Small number of reads prevent lots of area from holding larger postings lists (assuming zero overlap)

# Stronger Attack

Start with all memory unmarked.

1. Observe reads for smallest posting lists.
    - Mark out area where larger intervals will not fit.
2. Observe reads for next larger size of posting lists.
    - Mark out more area where larger intervals will not fit.
3. Iterate for all sizes

➡ Eventually conclude that a huge postings list will not fit at all

➡ Allows distinguishing $I_0$ and $I_1$

# Summary

- ➡ first results showing security requires poor i/o efficiency
- ➡ unconditional lower bounds via new proof technique
  - different from known i/o lower bounds
- ➡ improved theoretical i/o efficiency of prior work

**Q1**: Tighten gap between upper/lower bound?

**Q2**: Fine-grained lower bounds?

**Q3**: Other primitives where i/o efficiency dominates?

Thanks!