Dynamic Searchable Symmetric Encryption

Tom Roeder
eXtreme Computing Group
Microsoft Research

Joint work with Seny Kamara

Encrypted Cloud Backup

- Cloud backup
  - Users want to back up their data
  - The cloud provides storage
- Privacy, integrity, and confidentiality
  - But servers learn much about users this way
  - Honest-but-curious server can read everything
  - Malicious server can make arbitrary changes
- Naïve solution: store all data encrypted
  - User keeps key and decrypts locally
  - Problems: key management, search, cloud computation

Searchable Symmetric Encryption (SSE)

- SSE solves the search problem
  - Encrypt an index
  - User keeps key and generates search tokens
  - Server can use tokens to search encrypted index
- Practical implementations need update
  - Current impls do not have efficient update
  - Either no supported update operations
  - Or each word has size linear in all documents
- We provide two schemes with efficient update
  1. Update (add or delete) per word/doc pair
  2. Update (add or delete) per doc

Overview

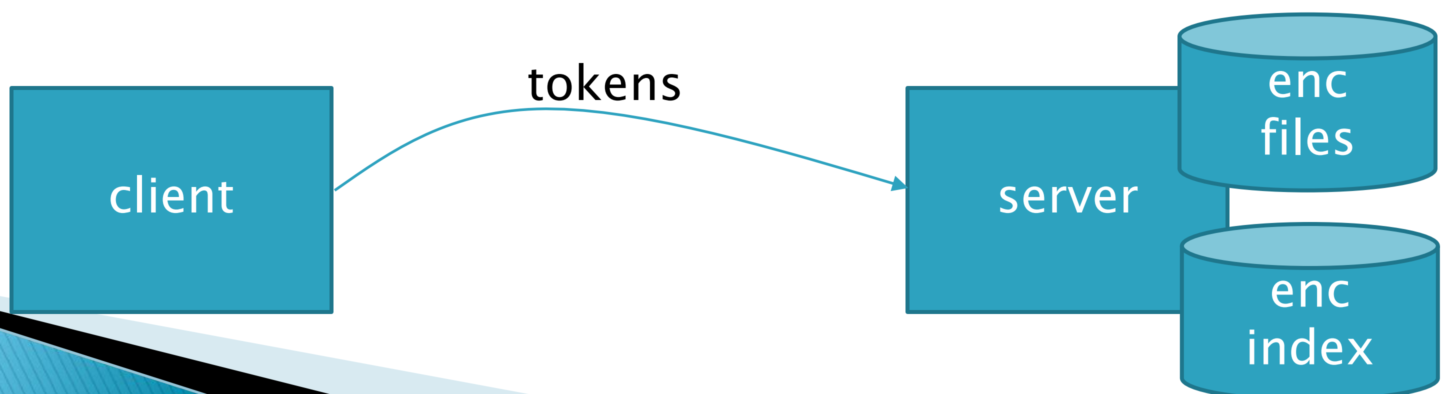
- Introduction
- Dynamic SSE Protocols
- Security Proofs
- Implementation

The Encrypted Search Problem

- User has collection $d_1, d_2, \dots, d_m$ of documents
  - d is a document identifier
  - Each document d has set of unique words W_d
  - Set of all unique words: $w_1, w_2, \dots, w_n$
- Goal: Produce an encrypted index with ops
  - Search(w): returns encrypted doc ids
  - Add(d, W_d): adds the doc id with word set
  - Delete(d): deletes the doc id and all words
  - Expand(): expands the index

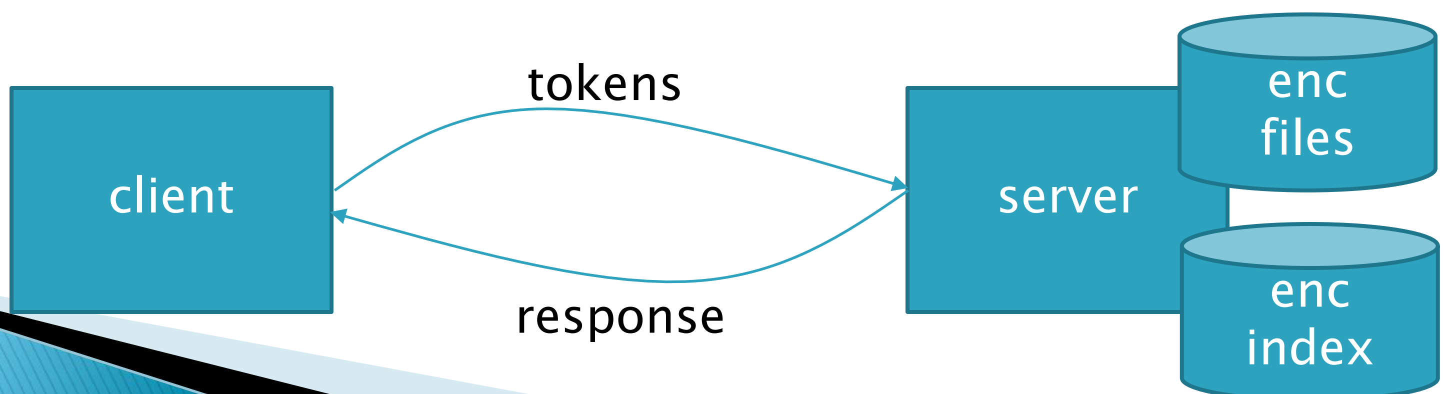
The Encrypted Search Problem

- User has collection $d_1, d_2, \ldots, d_m$ of documents
  - d is a document identifier
  - Each document d has set of unique words W_d
  - Set of all unique words: $w_1, w_2, \ldots, w_n$
- Goal: Produce an encrypted index with ops
  - Search(w): returns encrypted doc ids
  - Add(d, W_d): adds the doc id with word set
  - Delete(d): deletes the doc id and all words
  - Expand(): expands the index



The Encrypted Search Problem

- User has collection $d_1, d_2, \ldots, d_m$ of documents
  - d is a document identifier
  - Each document d has set of unique words W_d
  - Set of all unique words: $w_1, w_2, \ldots, w_n$
- Goal: Produce an encrypted index with ops
  - Search(w): returns encrypted doc ids
  - Add(d, W_d): adds the doc id with word set
  - Delete(d): deletes the doc id and all words
  - Expand(): expands the index

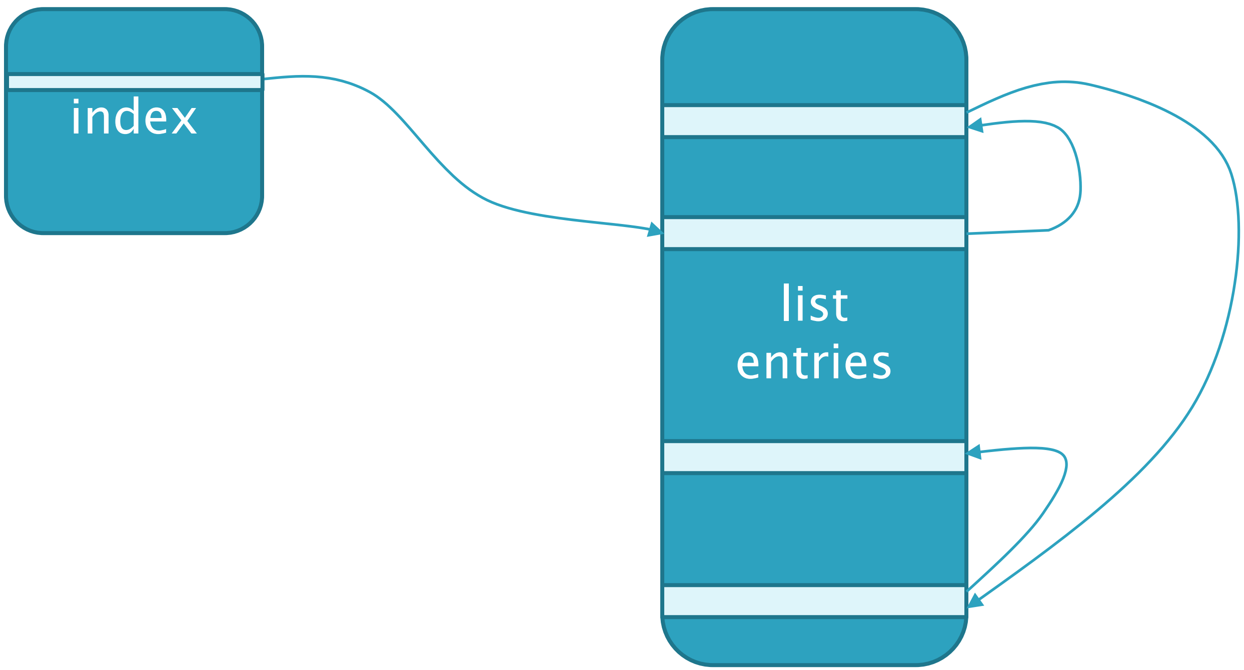


CGKO

- SSE scheme without update operations
- Main idea:
  - Each word is mapped to a token (under PRF)
  - Tokens map to an initial position in encrypted array
  - Each position points to next element in list
- The large encrypted, randomized array hides the document count for each word
- In original form, only secure against non-adaptive adversaries
- Assume honest-but-curious server

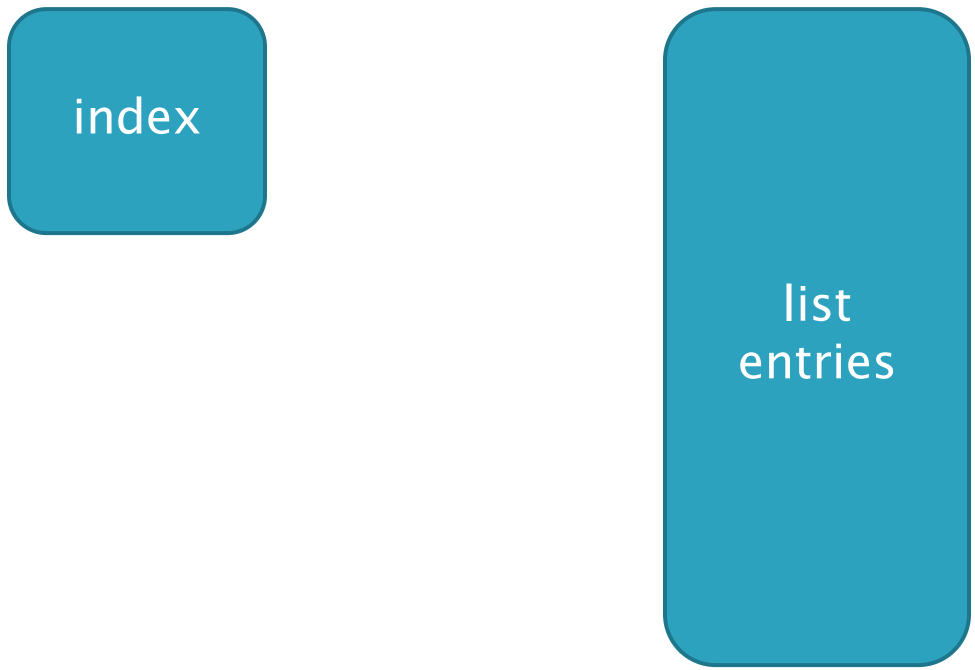
Modified CGKO

- index : $f_{k_c}(w) \rightarrow \langle start \rangle \oplus f_{k_b}(w)$
- list entry : $Enc_{k_w}(next), Enc_{k_e}(d)$



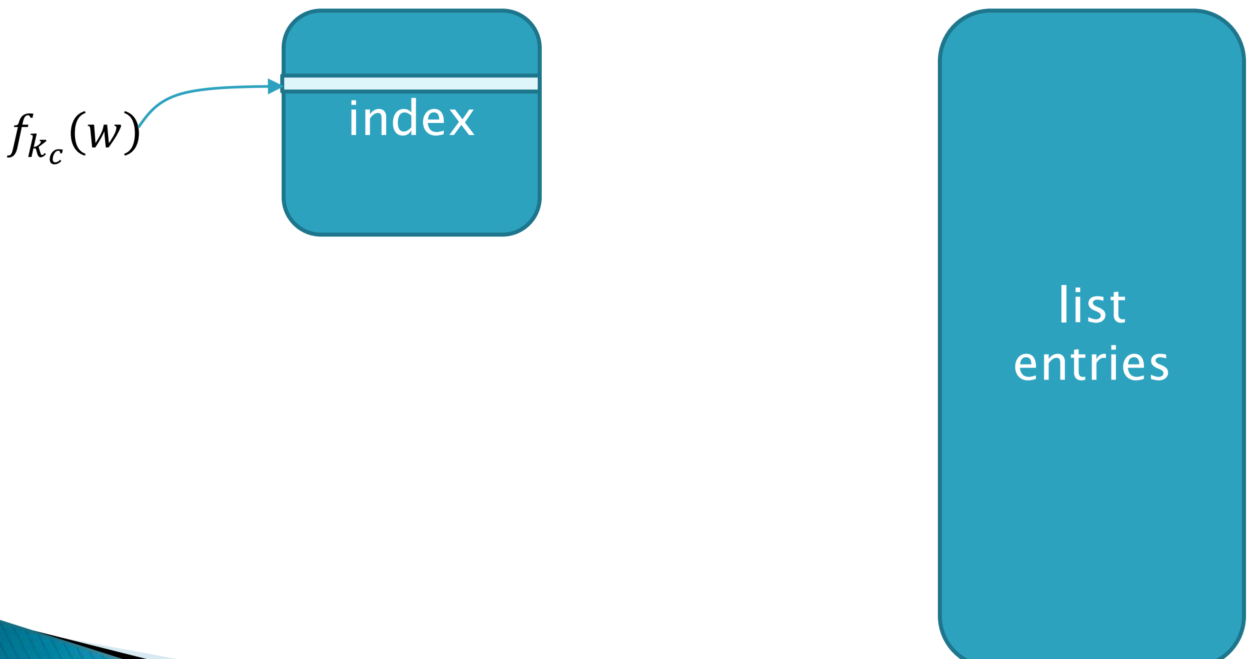
Modified CGKO: Search

- Given
  - w, k_c, k_b, k_g. $k_w = KDF_{k_g}(w)$
  - construct token $f_{k_c}(w), f_{k_b}(w), k_w$



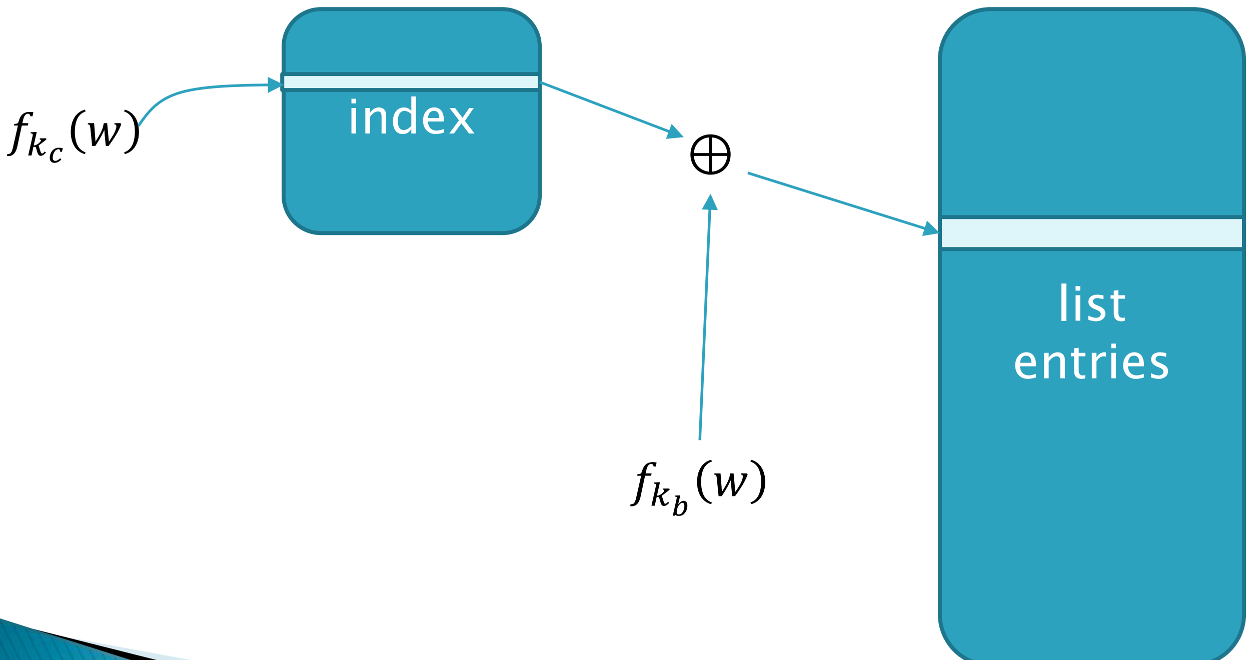
Modified CGKO: Search

- Given
  - w, k_c, k_b, k_g. $k_w = KDF_{k_g}(w)$
  - construct token $f_{k_c}(w), f_{k_b}(w), k_w$



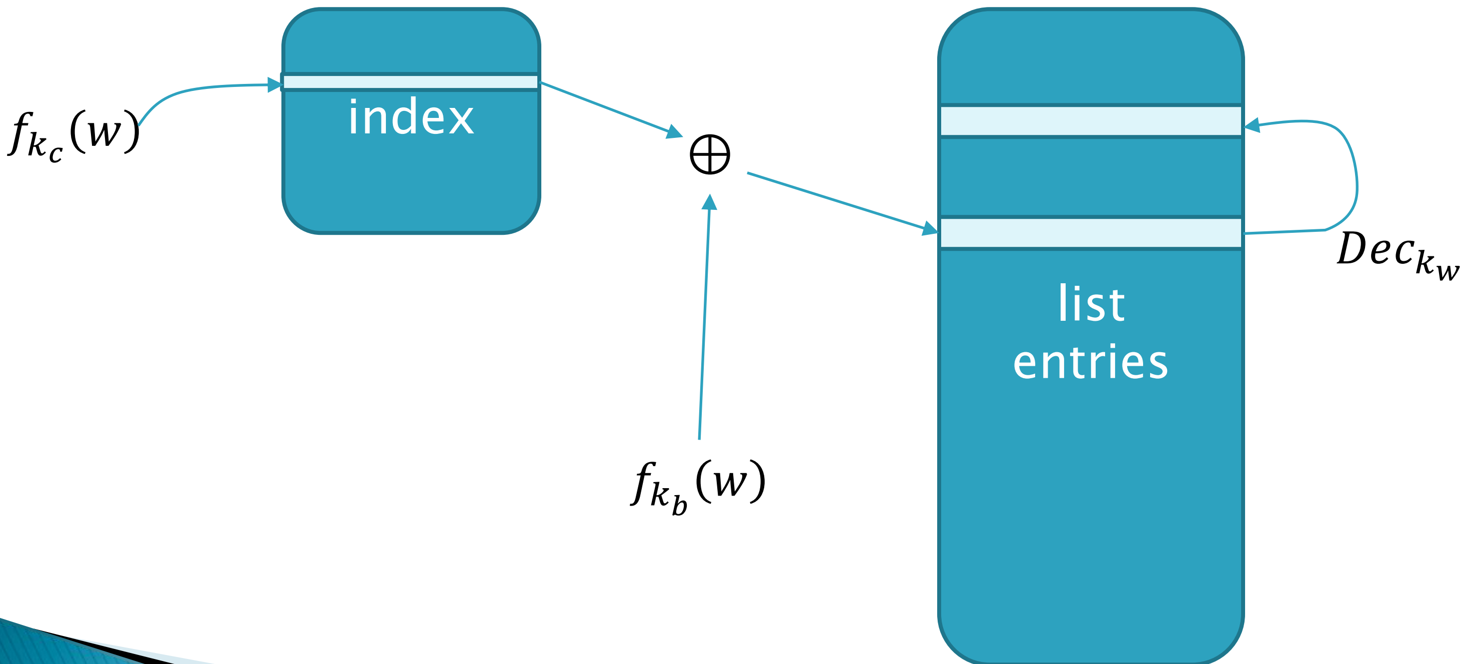
Modified CGKO: Search

- Given
  - w, k_c, k_b, k_g. $k_w = KDF_{k_g}(w)$
  - construct token $f_{k_c}(w), f_{k_b}(w), k_w$



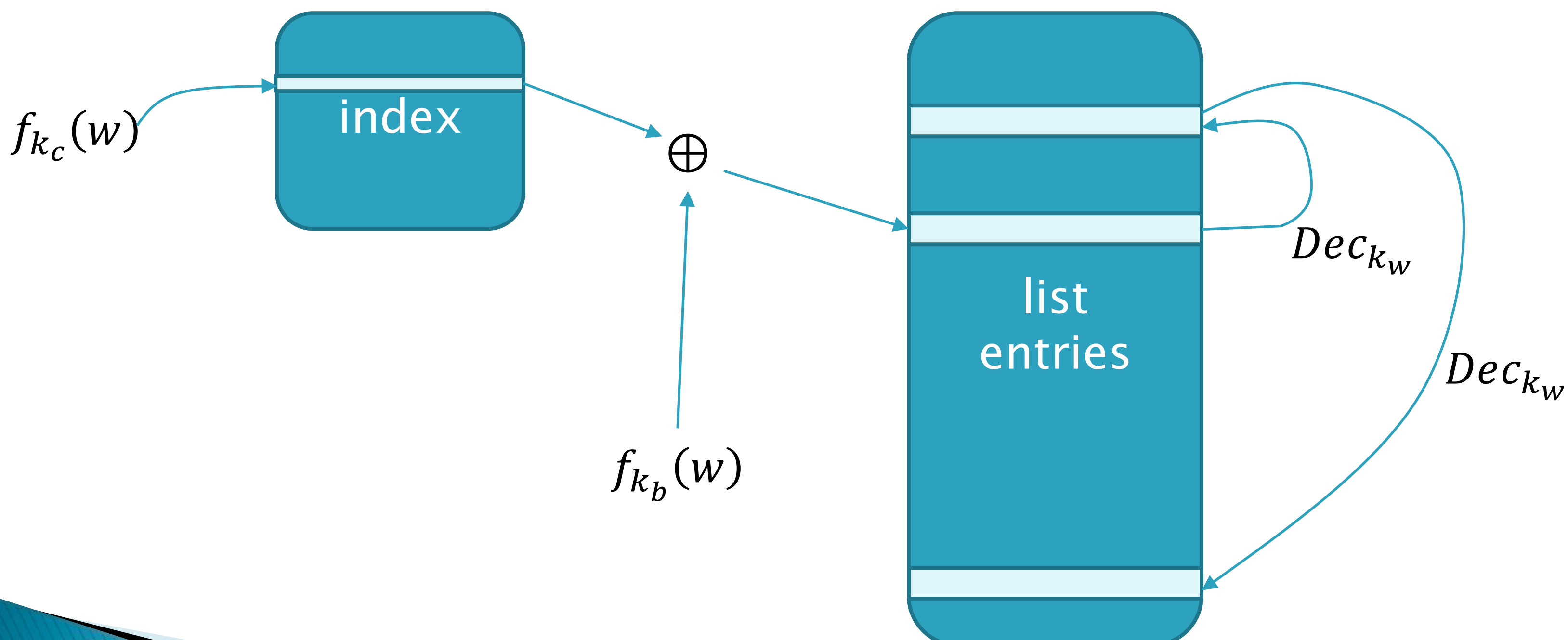
Modified CGKO: Search

- Given
  - w, k_c, k_b, k_g. $k_w = KDF_{k_g}(w)$
  - construct token $f_{k_c}(w), f_{k_b}(w), k_w$



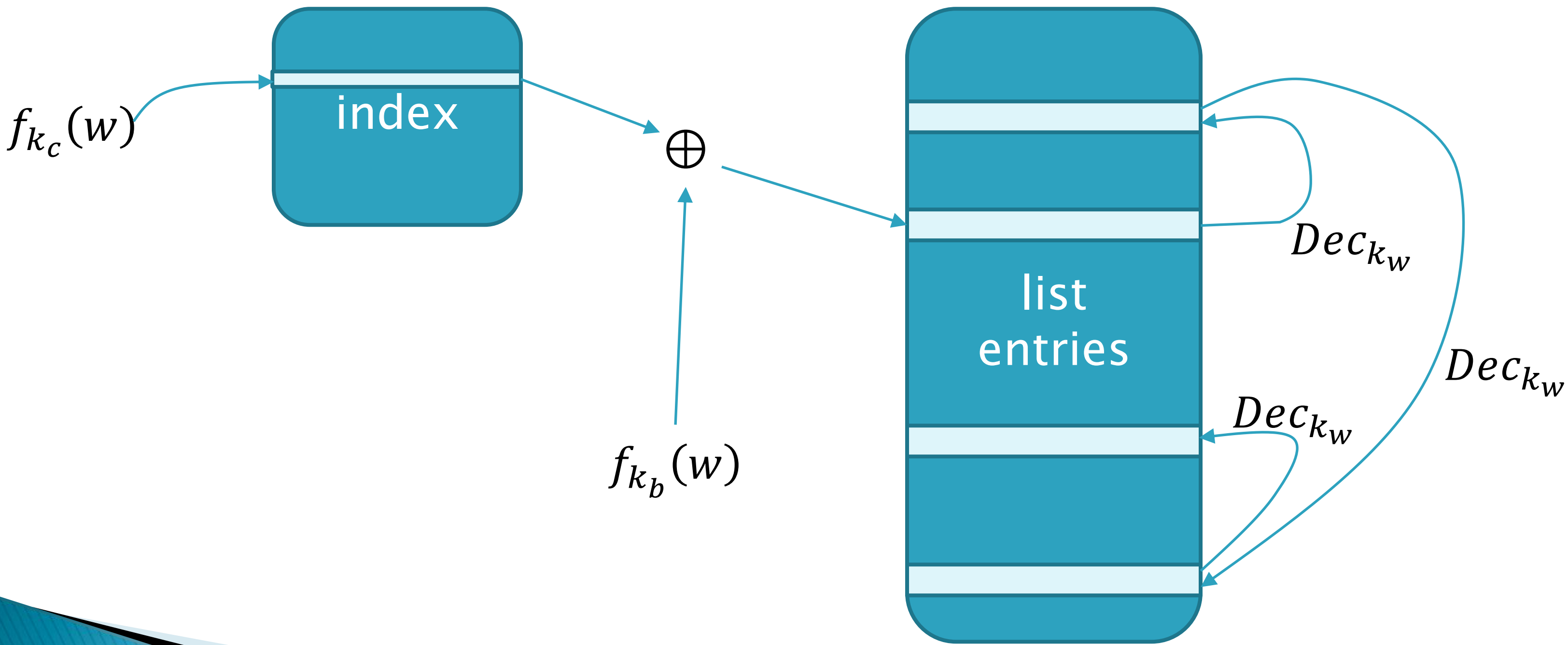
Modified CGKO: Search

- Given
  - w, k_c, k_b, k_g. $k_w = KDF_{k_g}(w)$
  - construct token $f_{k_c}(w), f_{k_b}(w), k_w$



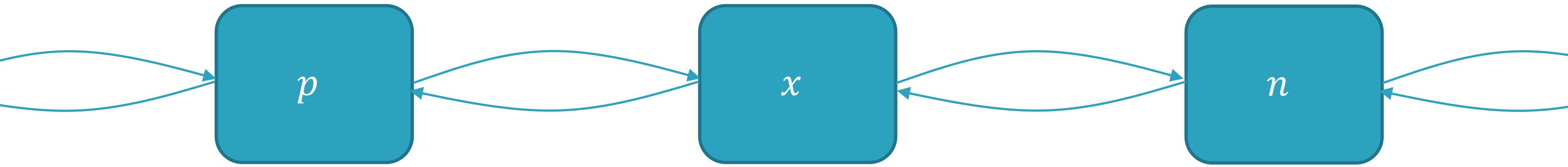
Modified CGKO: Search

- Given
  - w, k_c, k_b, k_g. $k_w = KDF_{k_g}(w)$
  - construct token $f_{k_c}(w), f_{k_b}(w), k_w$



List Patching

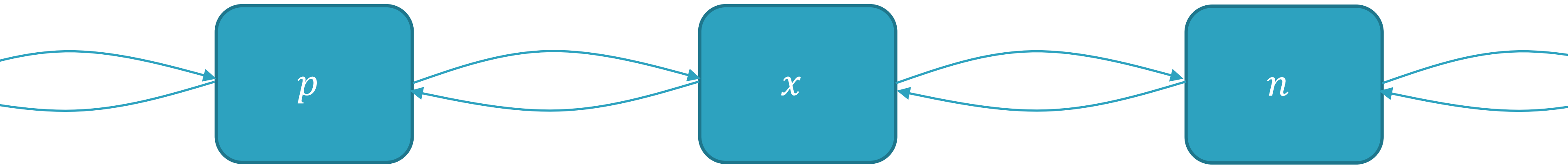
- To delete an entry (x), need
  - Location of entry to delete
  - Location of next (n) and prev (p) entries (if any)
- Use XOR encryption for list pointers



$r, \langle u, x \rangle \oplus f_{k_w}(r)$ $r', \langle p, n \rangle \oplus f_{k_w}(r')$ $r'', \langle x, v \rangle \oplus f_{k_w}(r'')$

List Patching

- To delete an entry (x), need
  - Location of entry to delete
  - Location of next (n) and prev (p) entries (if any)
- Use XOR encryption for list pointers



$r, \langle u, x \rangle \oplus f_{k_w}(r)$

$\oplus \langle 0, x \oplus n \rangle$

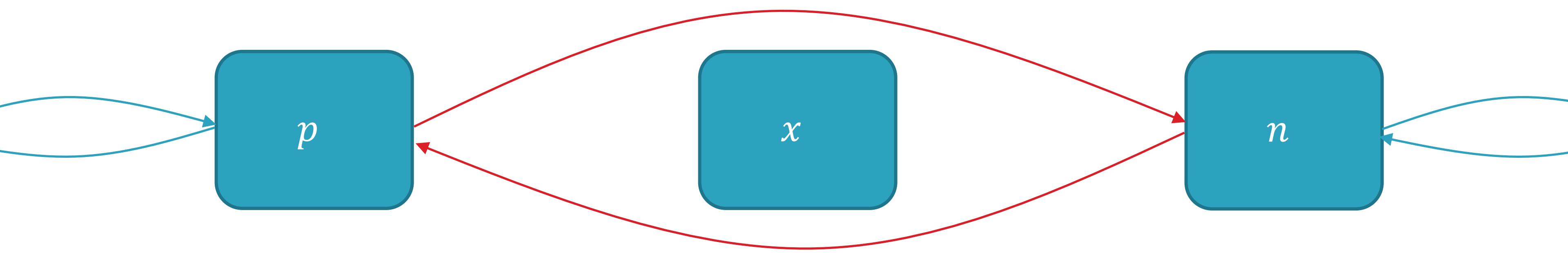
$r', \langle p, n \rangle \oplus f_{k_w}(r')$

$r'', \langle x, v \rangle \oplus f_{k_w}(r'')$

$\oplus \langle x \oplus p, 0 \rangle$

List Patching

- To delete an entry (x), need
  - Location of entry to delete
  - Location of next (n) and prev (p) entries (if any)
- Use XOR encryption for list pointers

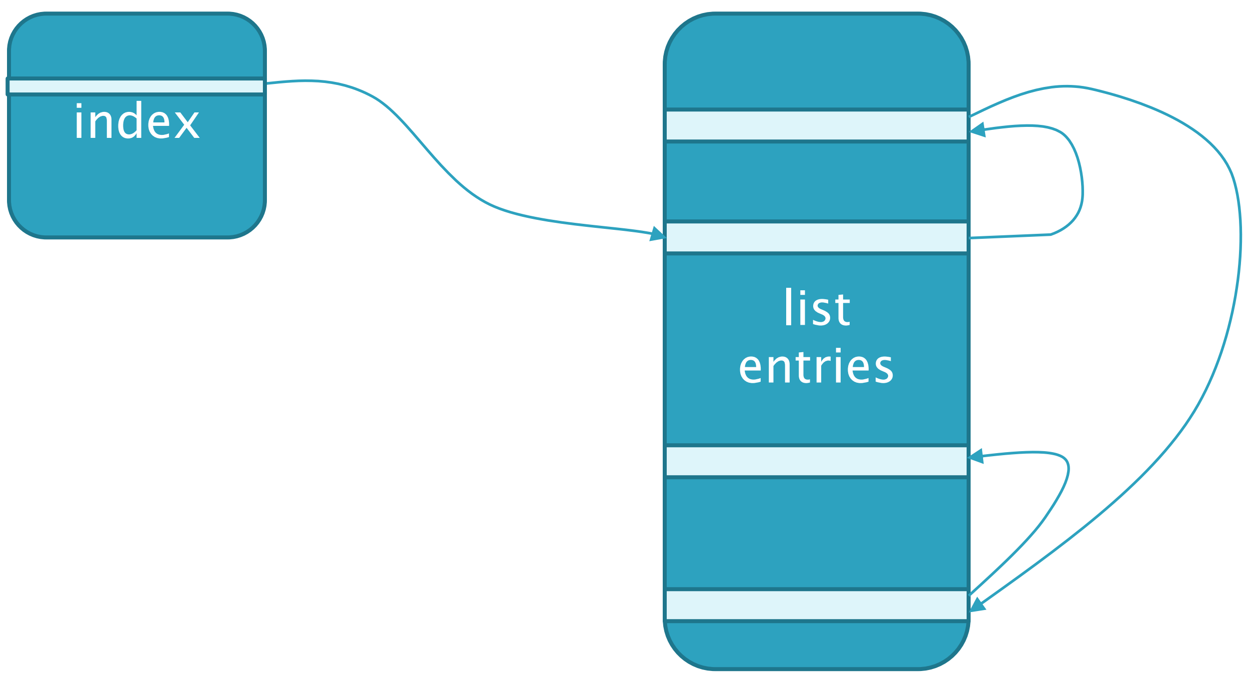


$r, \langle u, n \rangle \oplus f_{k_w}(r)$ $\quad$ $r', \langle p, n \rangle \oplus f_{k_w}(r')$ $\quad$ $r'', \langle p, v \rangle \oplus f_{k_w}(r'')$

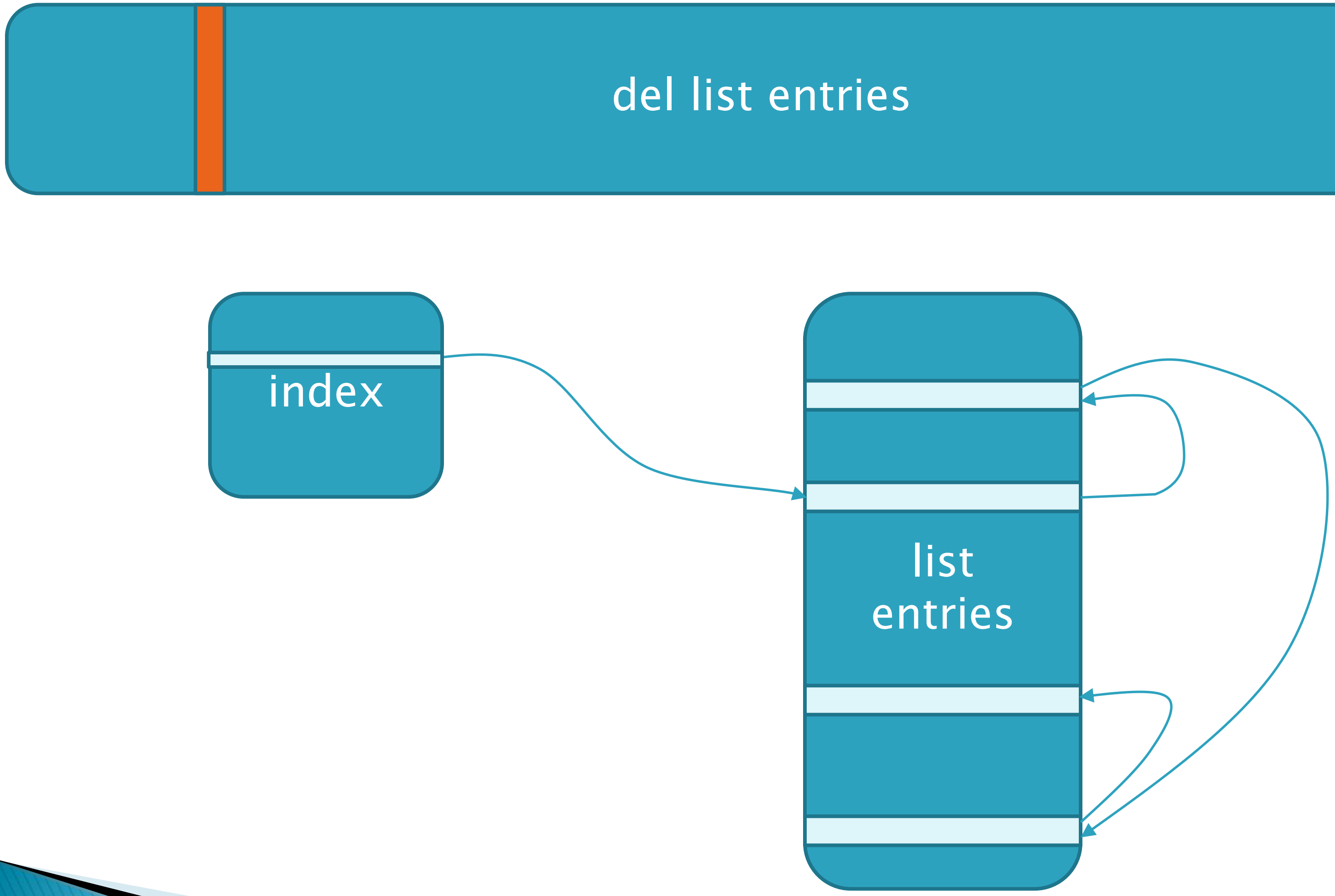
Deletion index

- To patch the data structure
  - E.g., pulling a document out of a list
  - And need a structure to index directly into the lists
- Add deletion index
  - Index: $f_{k_c}(d) \rightarrow \langle start \rangle \oplus f_{k_b}(d)$
  - $r, r', r'', \langle n_d, dn_x, dp_x \rangle \oplus f_{k_d}(r), \langle x, p, n \rangle \oplus f_{k_d}(r'),$
    $f_{k_c}(w) \oplus f_{k_d}(r'')$
  - list structure uses n_d to point to next word for d
  - dn_x and dp_x point to del index entries for n and p
  - 1-1 correspondence between list entries

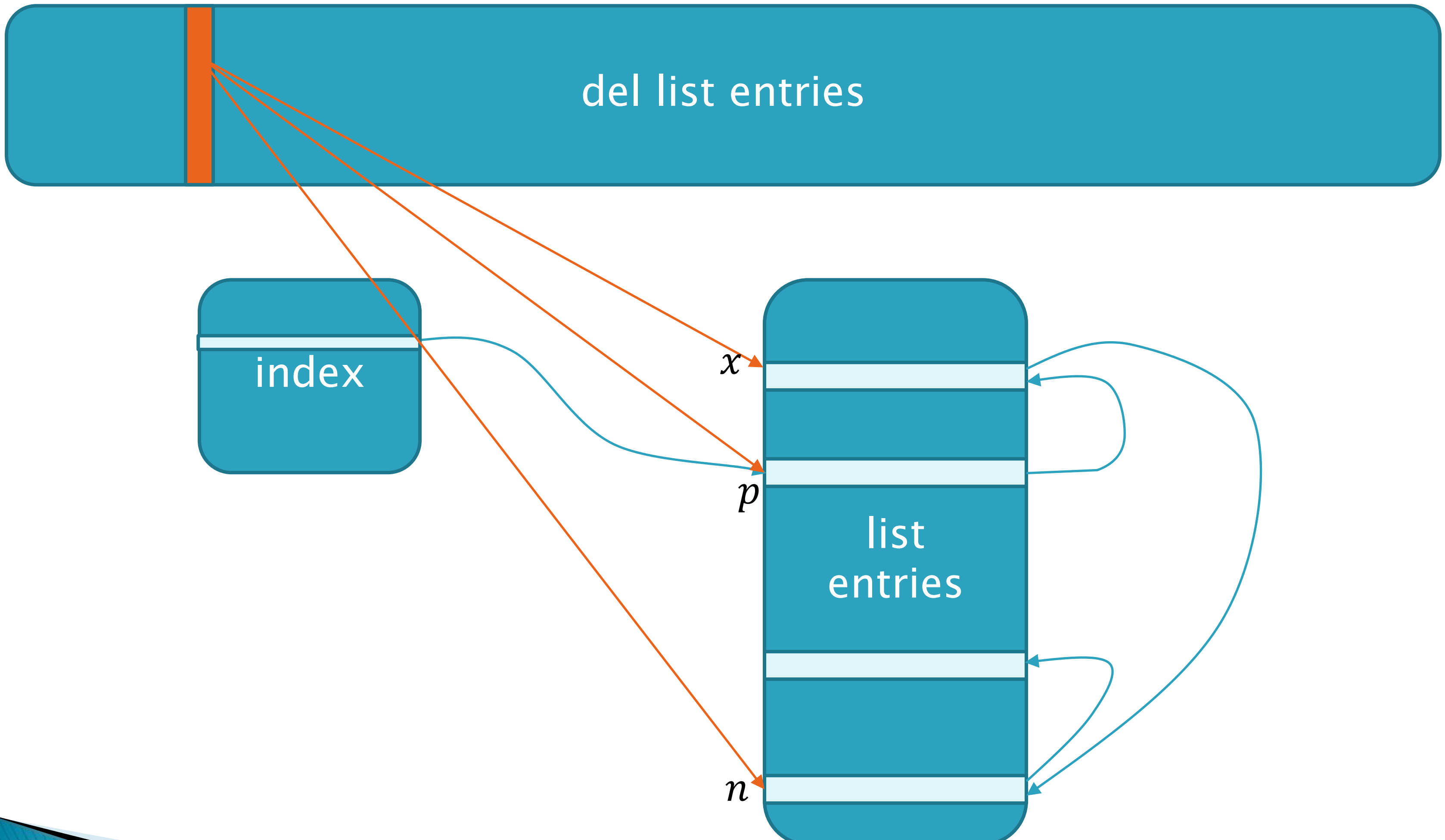
Doc-Based Index



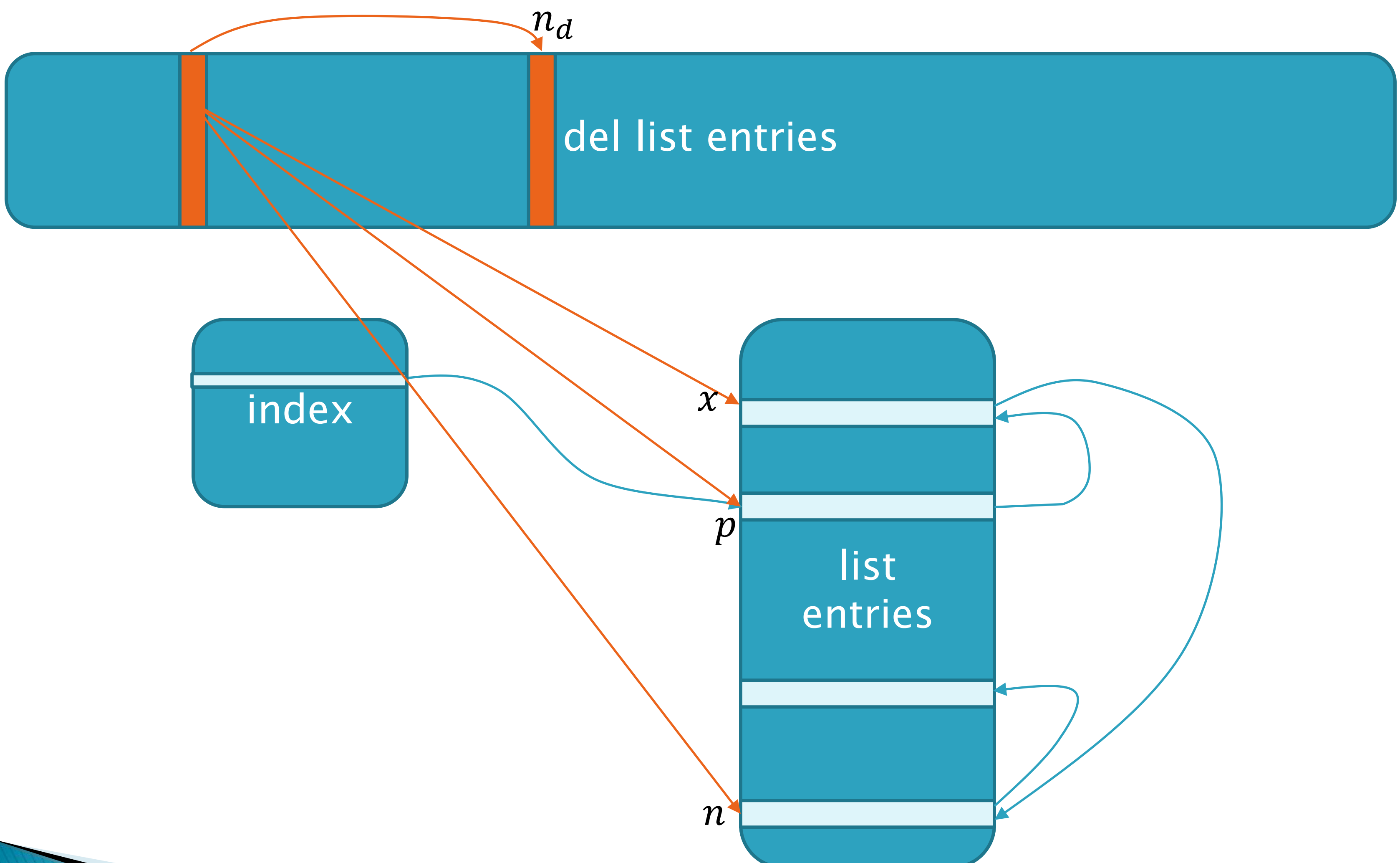
Doc-Based Index



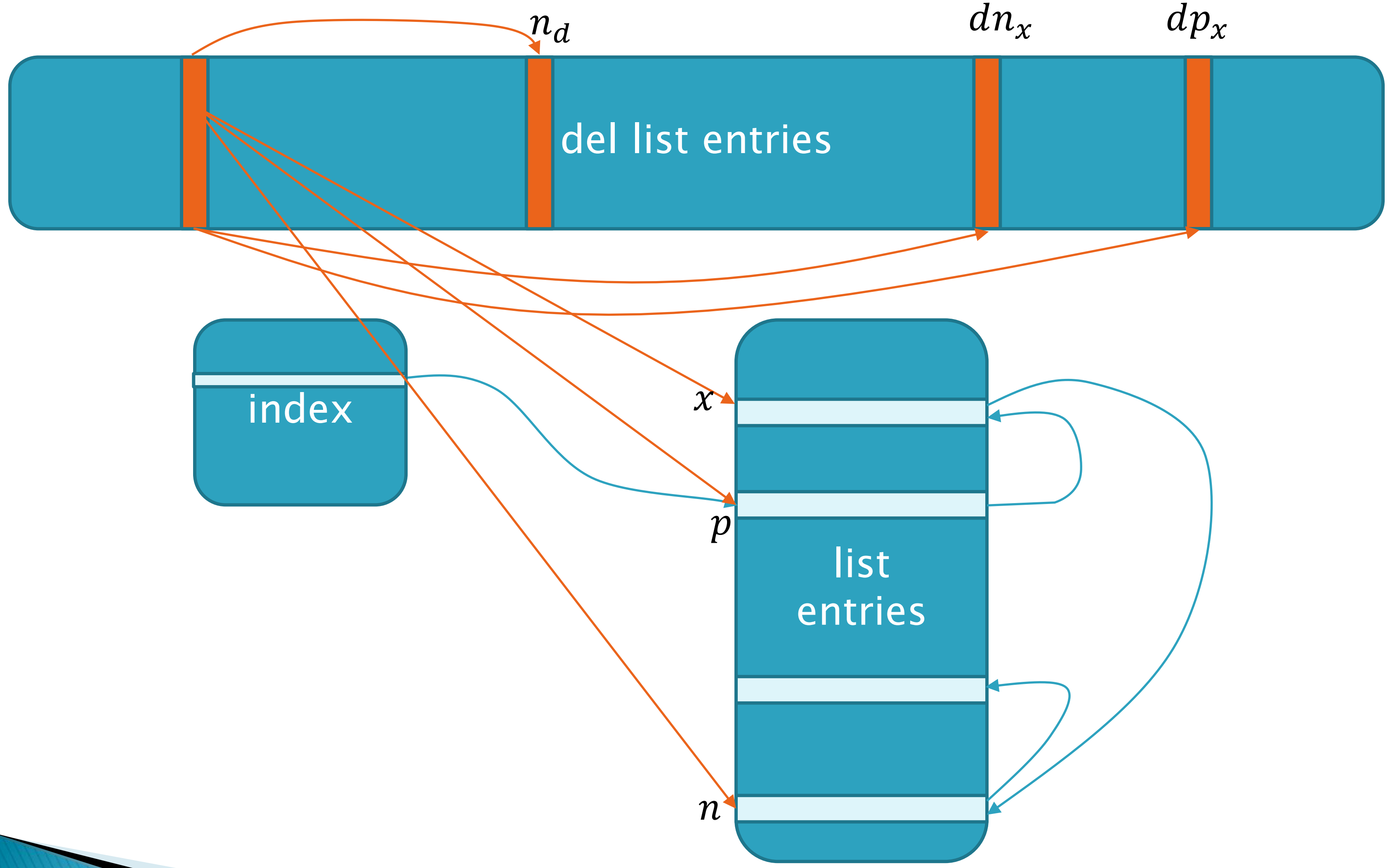
Doc-Based Index



Doc-Based Index

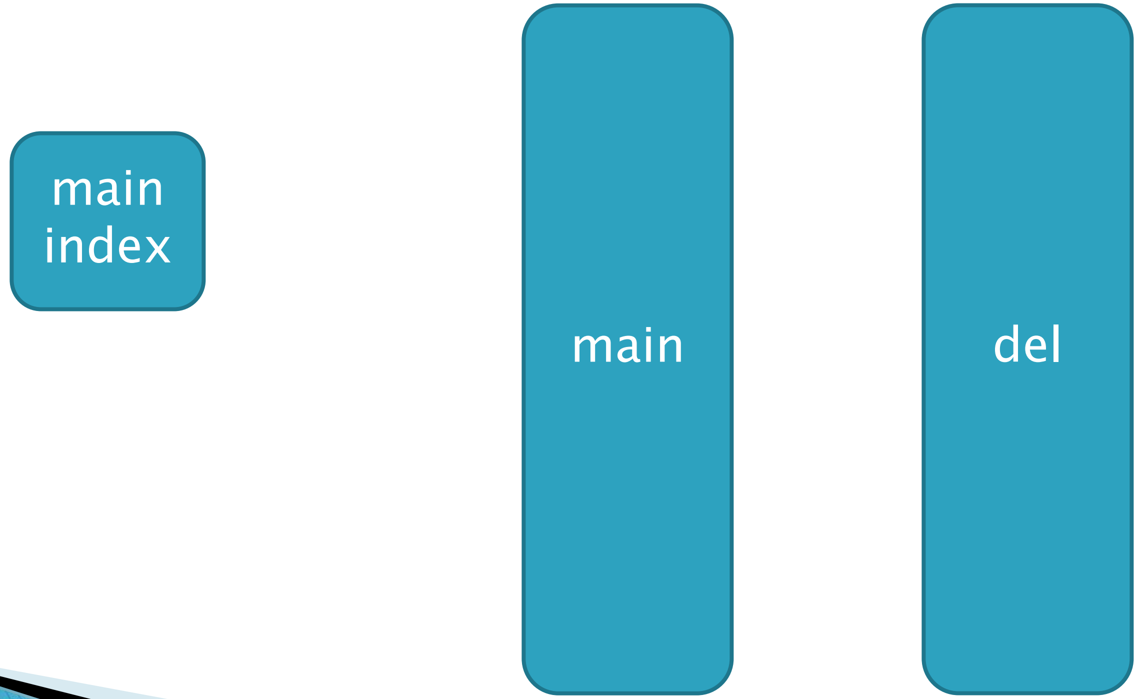


Doc-Based Index



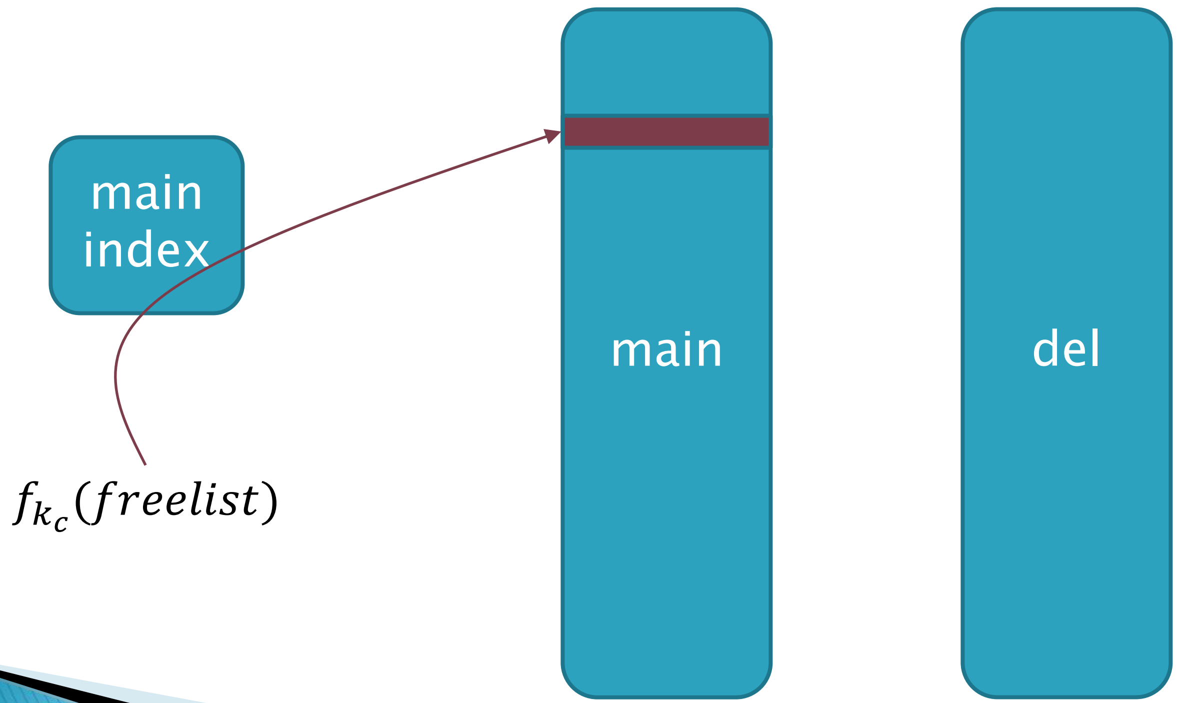
Free List

- Add and delete must track unused space
  - revealing unused would reveal word * doc
  - user must keep track of freelist count



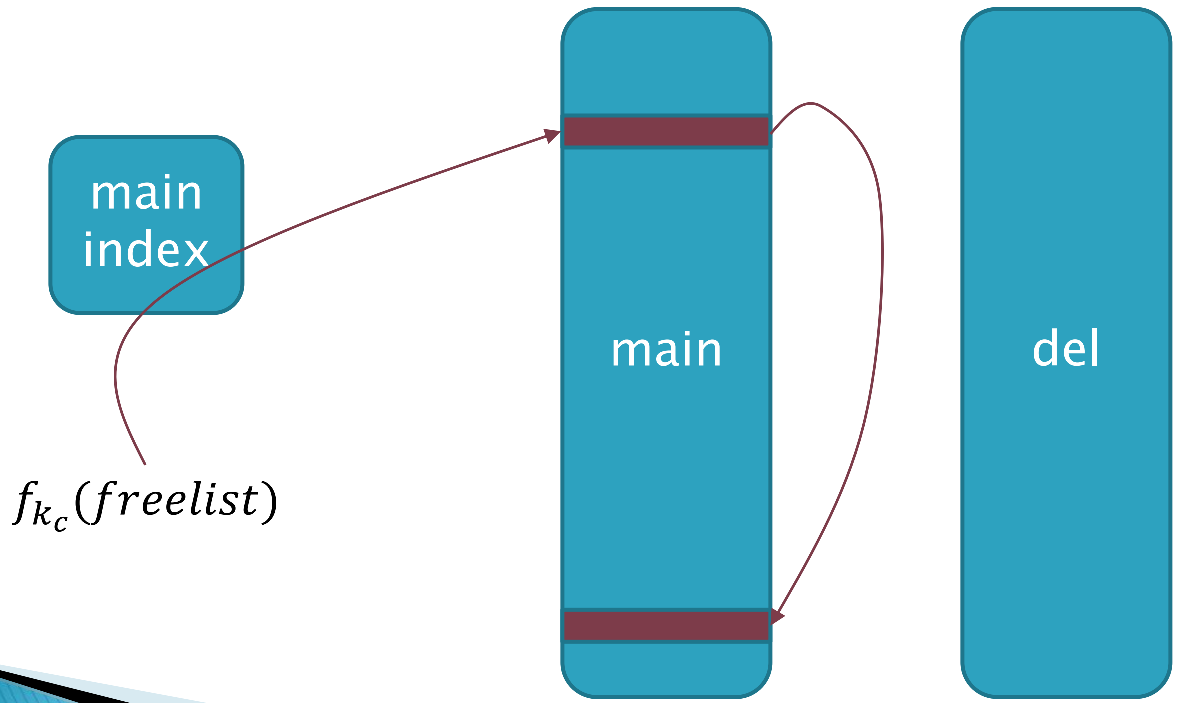
Free List

- Add and delete must track unused space
  - revealing unused would reveal word * doc
  - user must keep track of freelist count



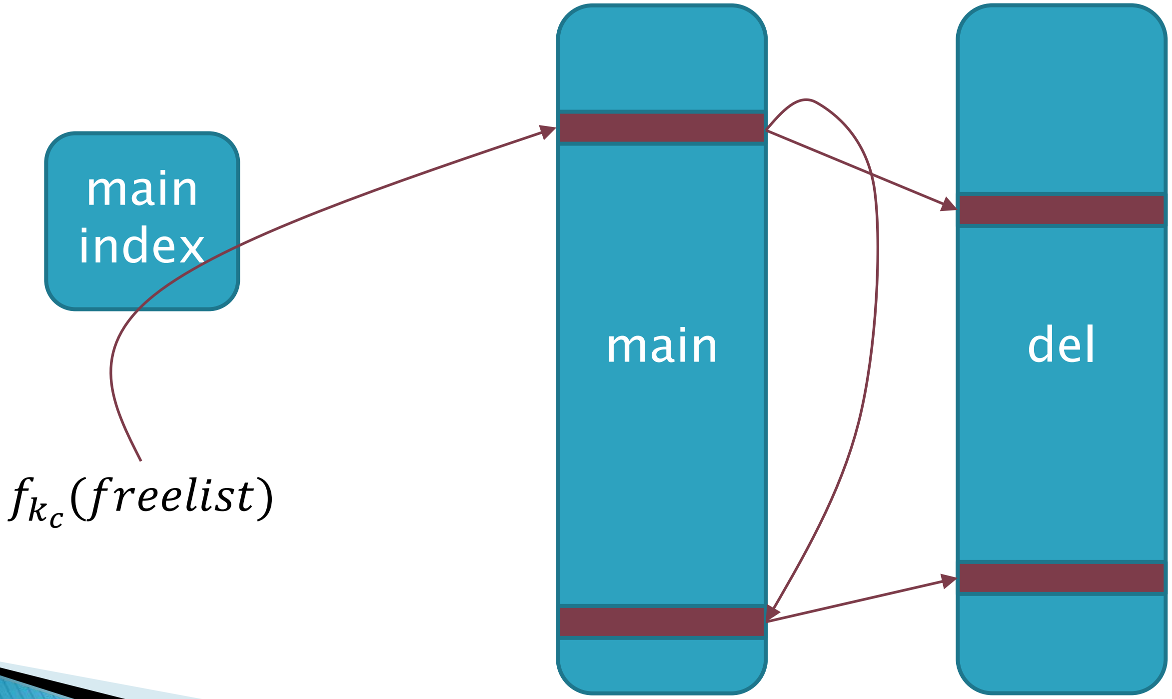
Free List

- Add and delete must track unused space
  - revealing unused would reveal word * doc
  - user must keep track of freelist count



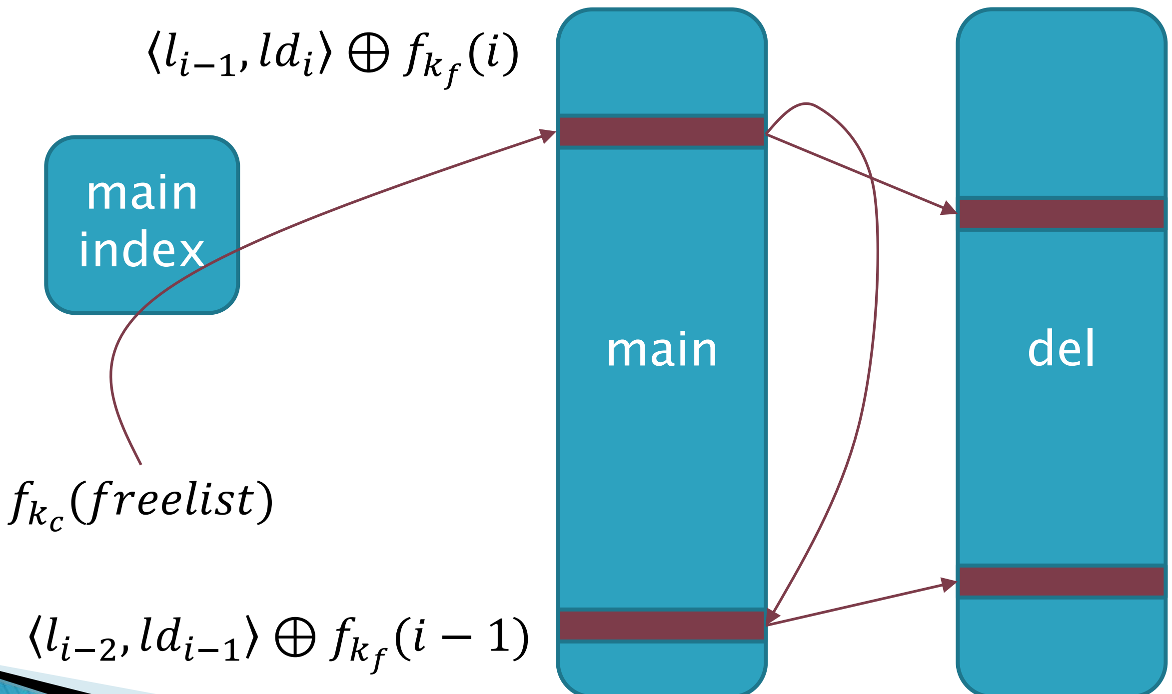
Free List

- Add and delete must track unused space
  - revealing unused would reveal word * doc
  - user must keep track of freelist count



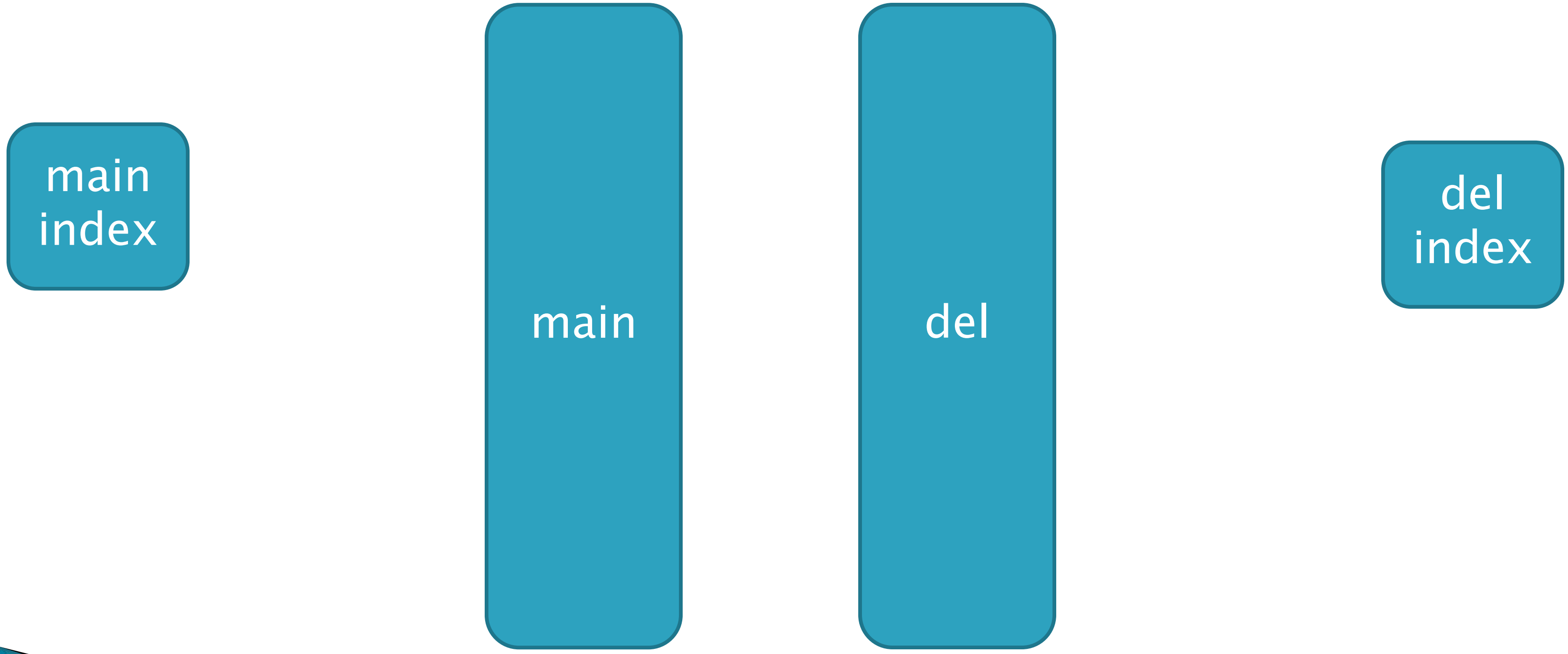
Free List

- Add and delete must track unused space
  - revealing unused would reveal word * doc
  - user must keep track of freelist count



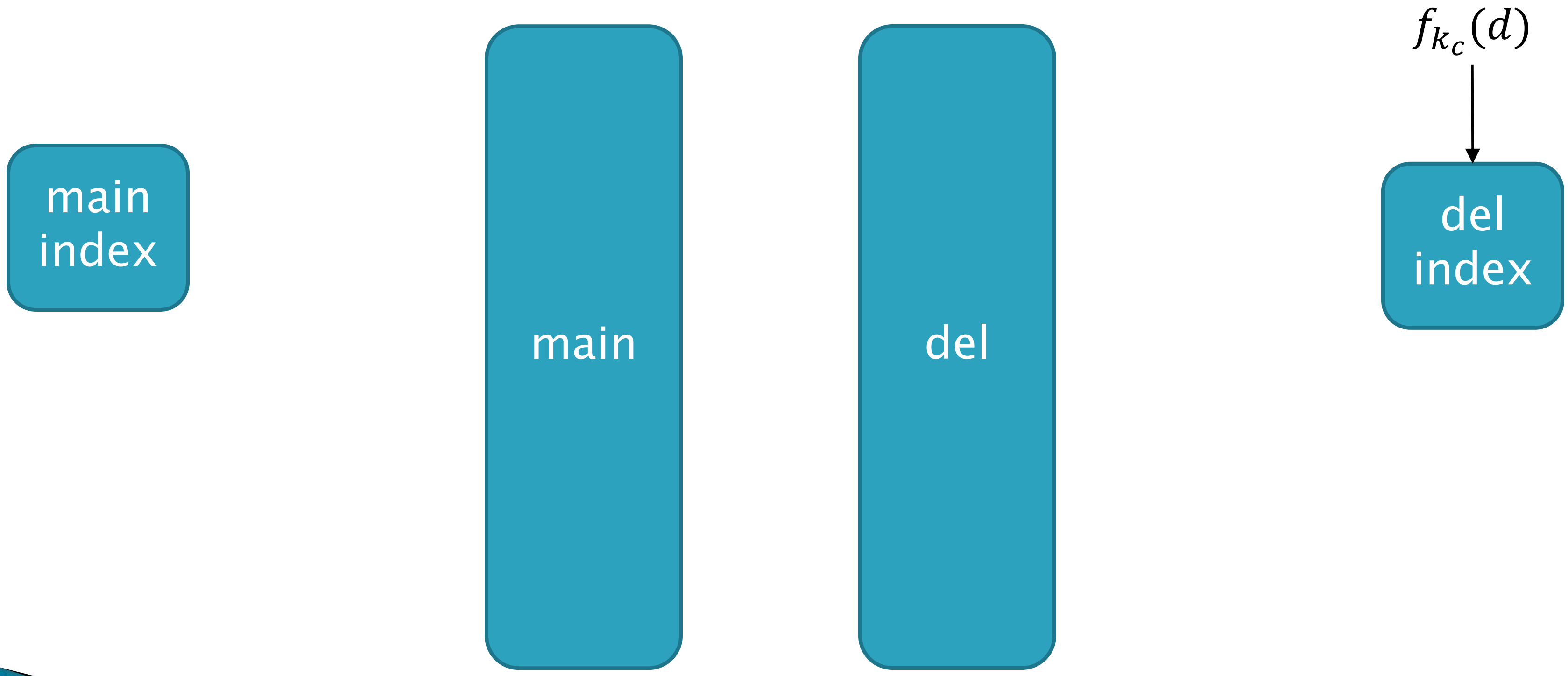
Add a Document

- ⟨doc tokens⟩, ⟨freelist tokens⟩, word count
  - per word: ⟨word tokens⟩, ⟨freelist mask⟩, templates



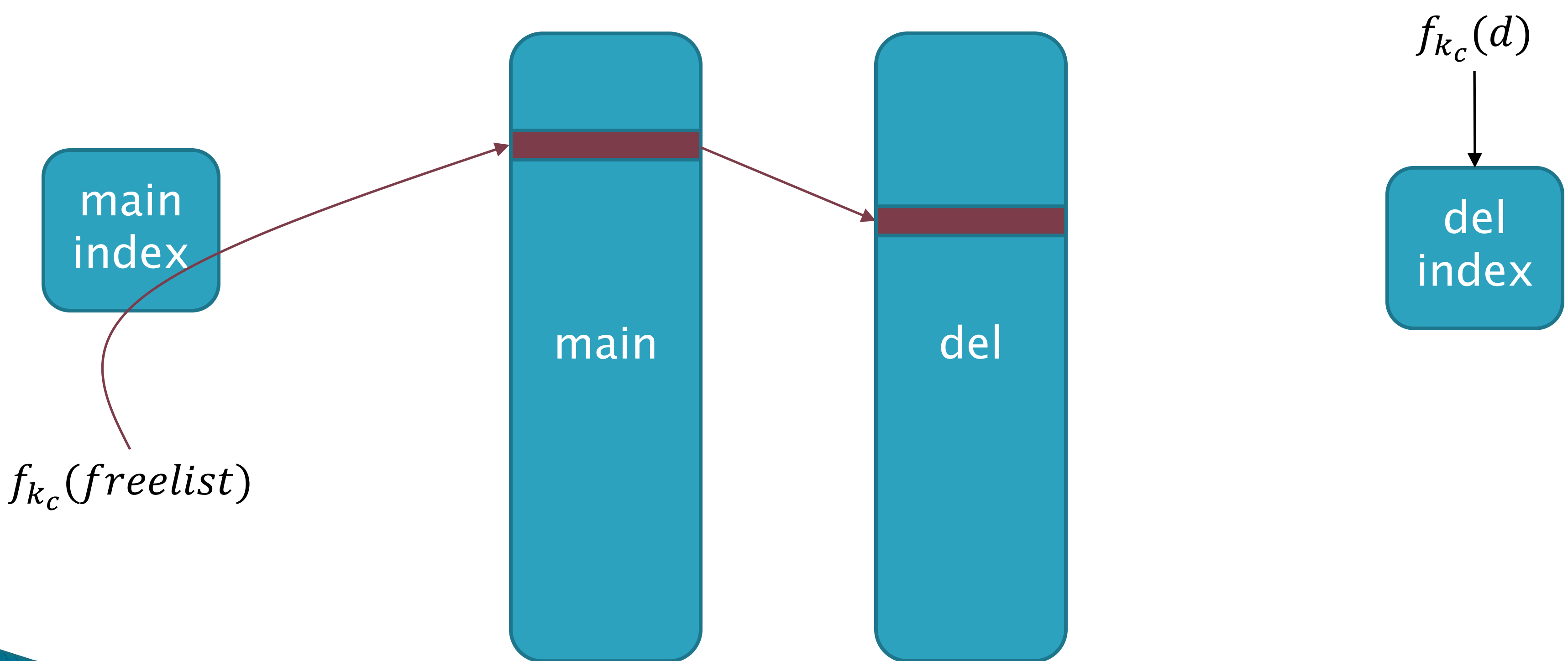
Add a Document

- ⟨doc tokens⟩, ⟨freelist tokens⟩, word count
  - per word: ⟨word tokens⟩, ⟨freelist mask⟩, templates



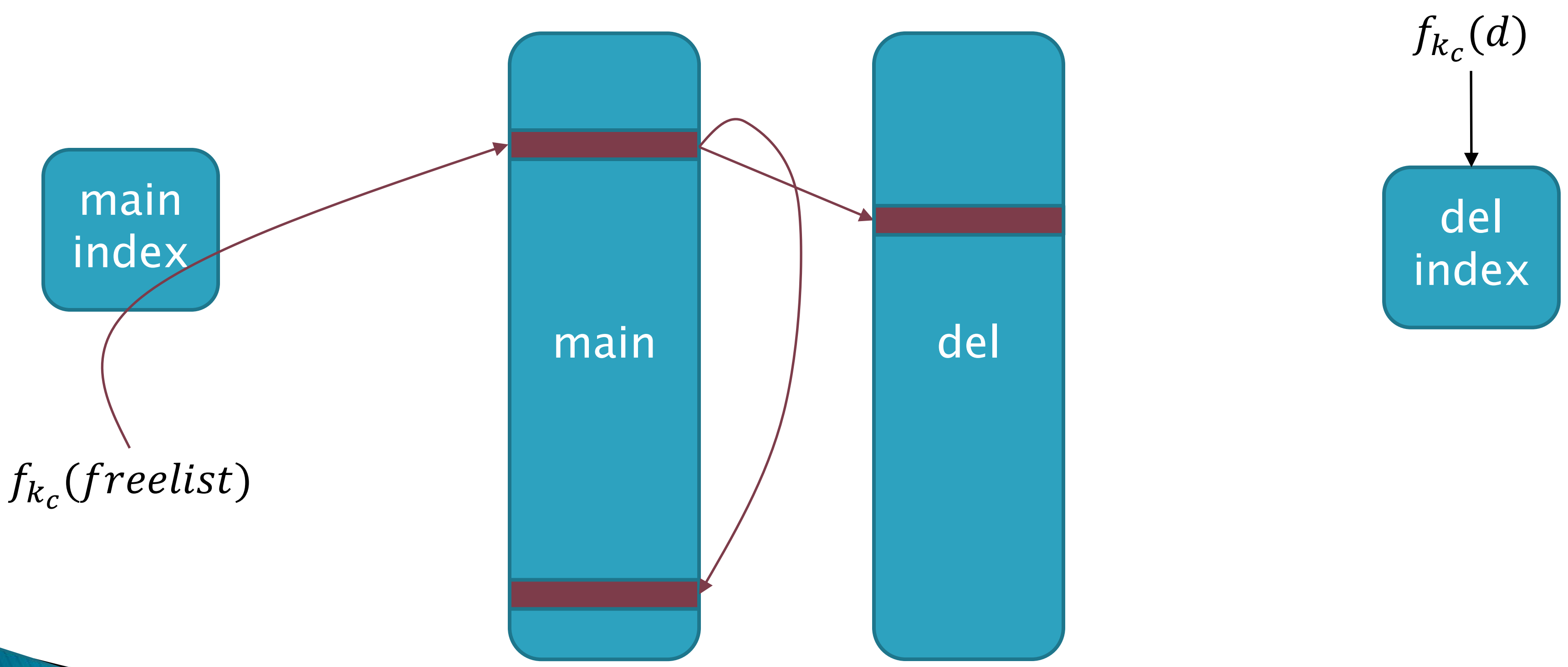
Add a Document

- ⟨doc tokens⟩, ⟨freelist tokens⟩, word count
  - per word: ⟨word tokens⟩, ⟨freelist mask⟩, templates



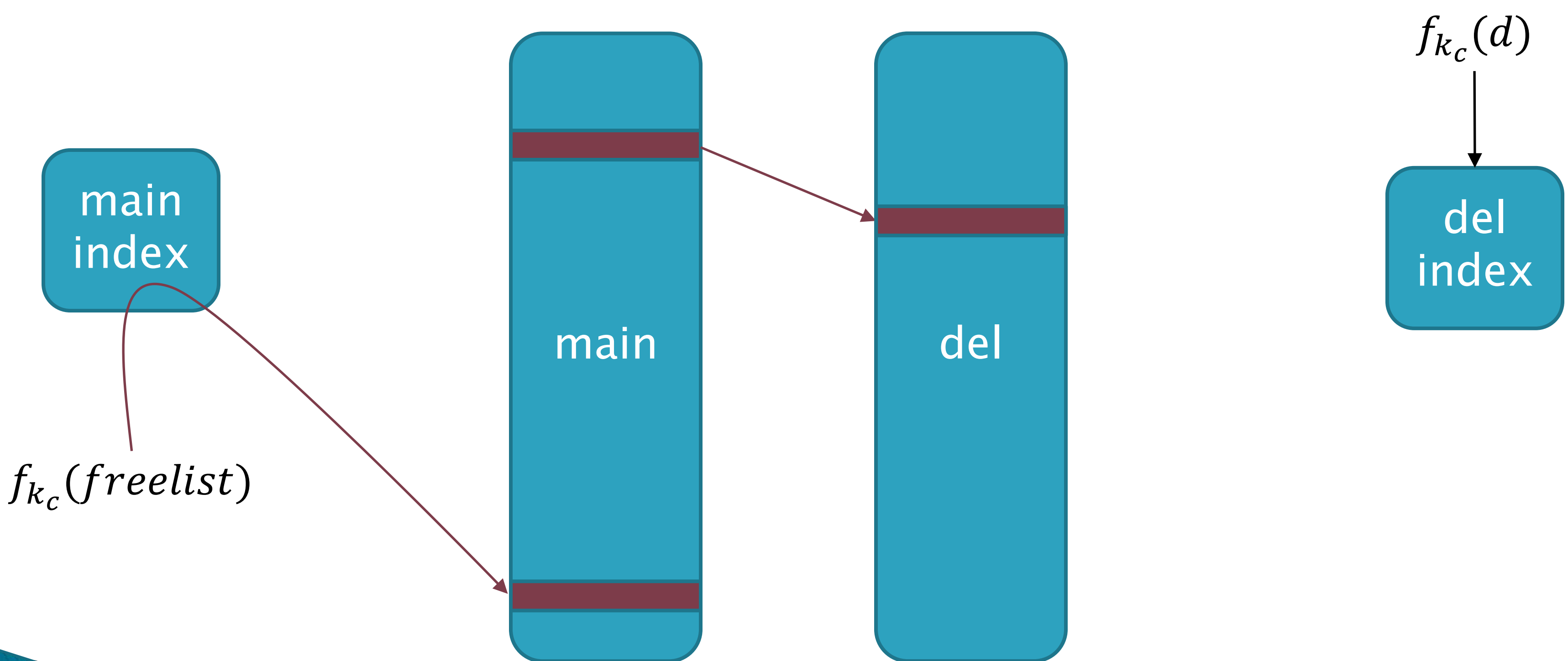
Add a Document

- ⟨doc tokens⟩, ⟨freelist tokens⟩, word count
  - per word: ⟨word tokens⟩, ⟨freelist mask⟩, templates



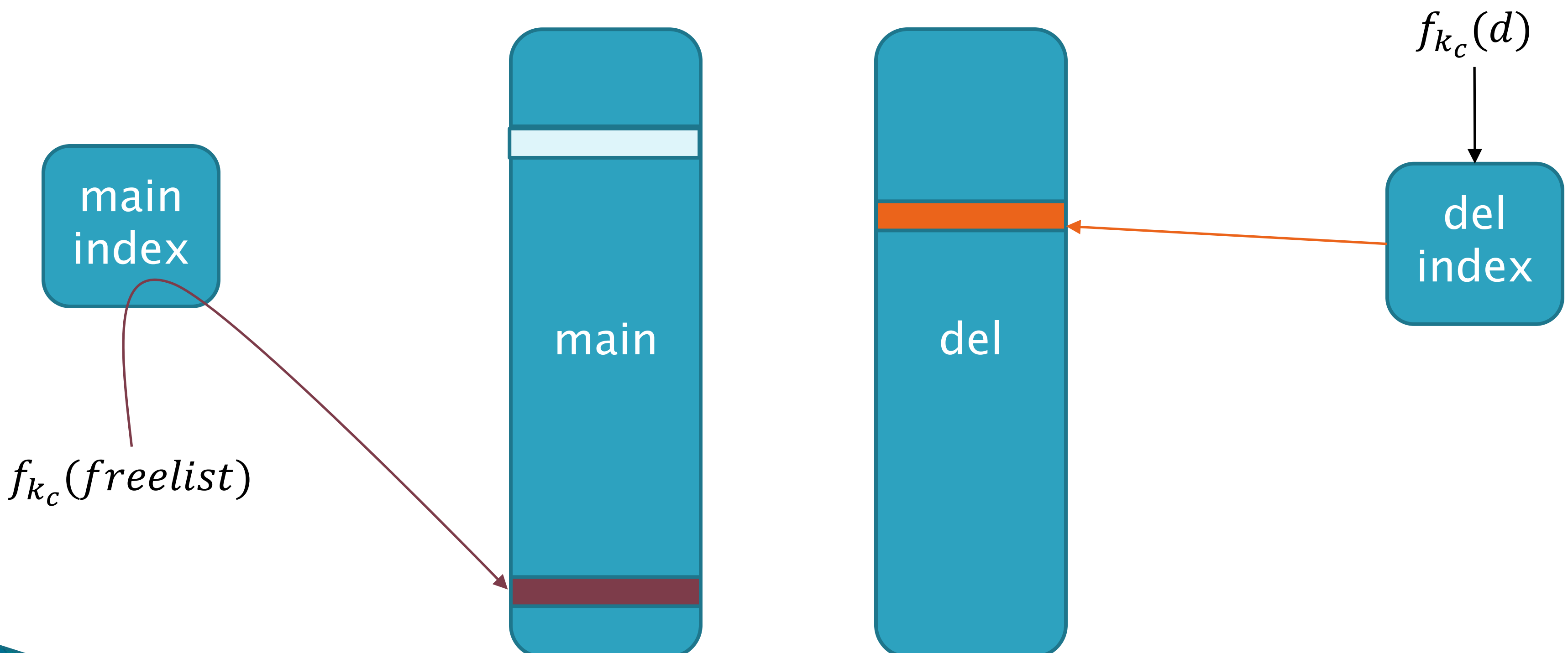
Add a Document

- ⟨doc tokens⟩, ⟨freelist tokens⟩, word count
  - per word: ⟨word tokens⟩, ⟨freelist mask⟩, templates



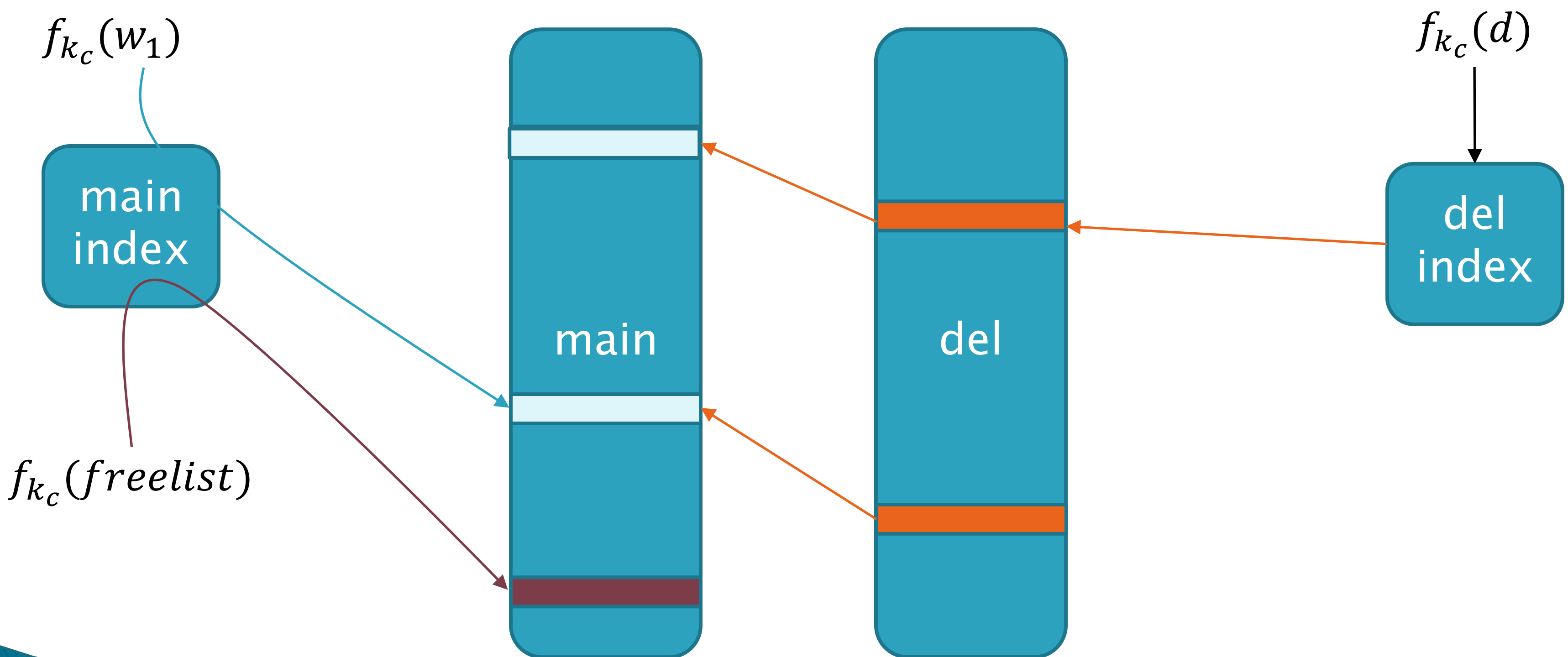
Add a Document

- ⟨doc tokens⟩, ⟨freelist tokens⟩, word count
  - per word: ⟨word tokens⟩, ⟨freelist mask⟩, templates



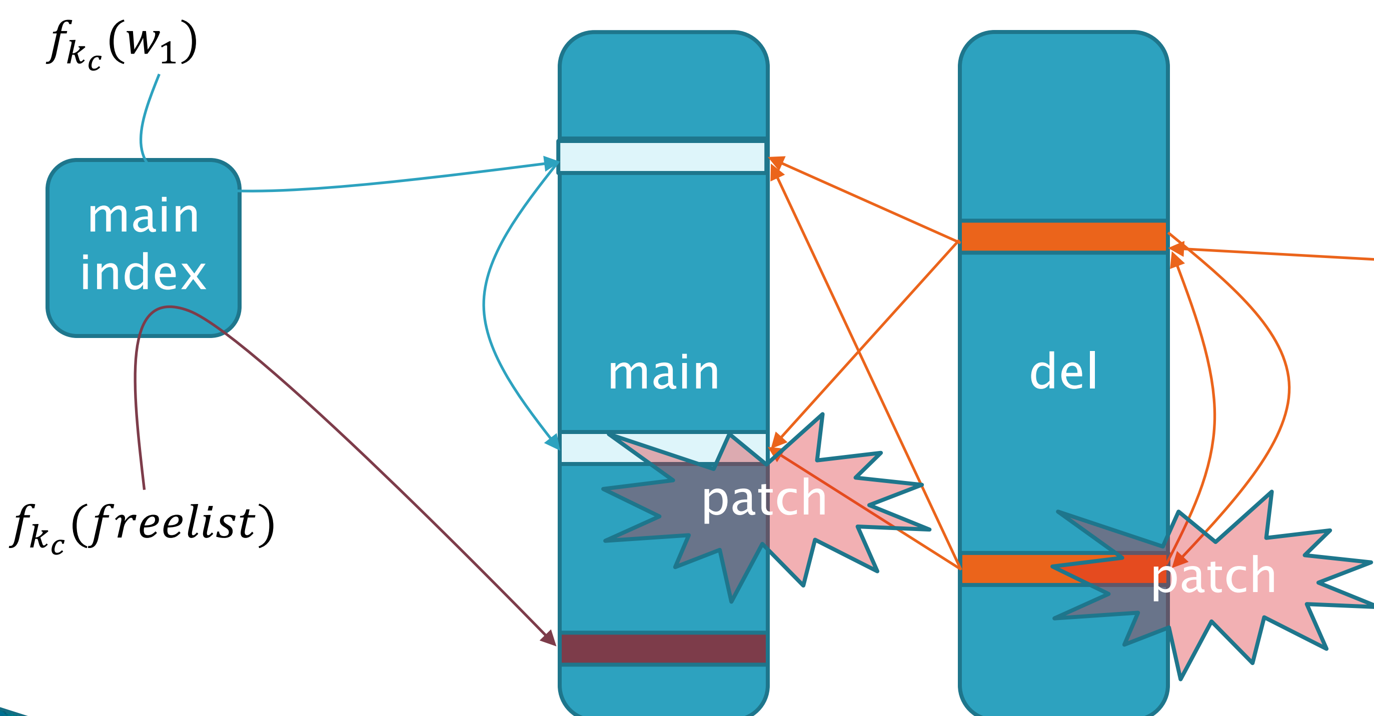
Add a Document

- ⟨doc tokens⟩, ⟨freelist tokens⟩, word count
  - per word: ⟨word tokens⟩, ⟨freelist mask⟩, templates



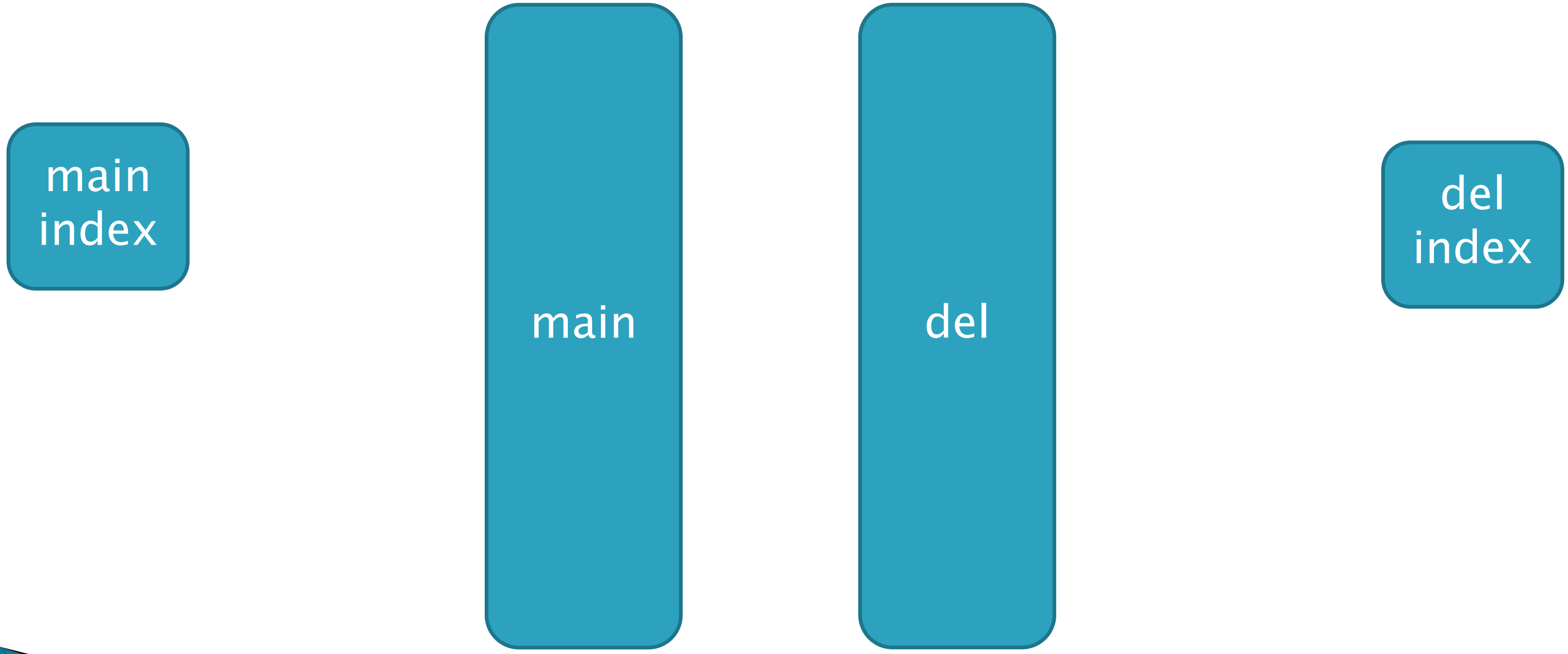
Add a Document

- ⟨doc tokens⟩, ⟨freelist tokens⟩, word count
  - per word: ⟨word tokens⟩, ⟨freelist mask⟩, templates



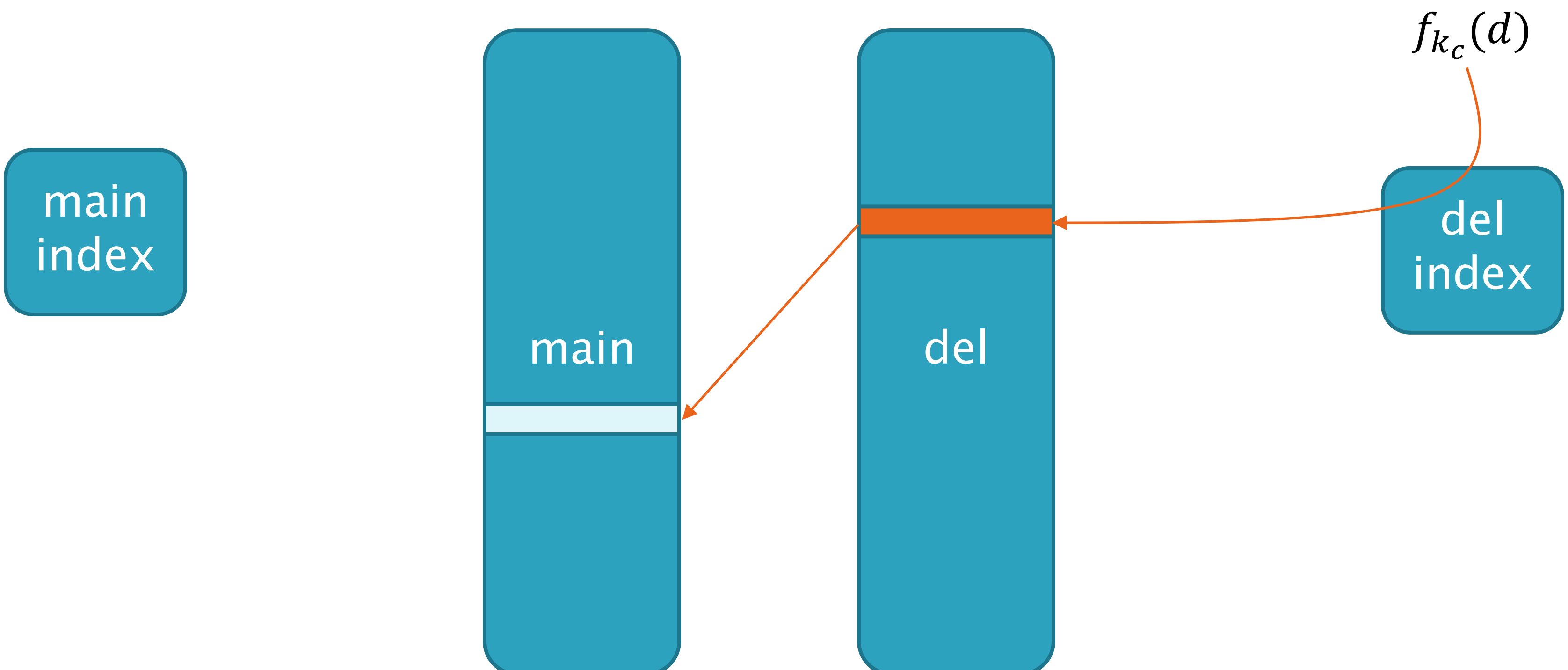
Delete a Document

- ⟨doc tokens⟩, doc key, ⟨freelist tokens⟩, count
  - per word: ⟨freelist mask⟩



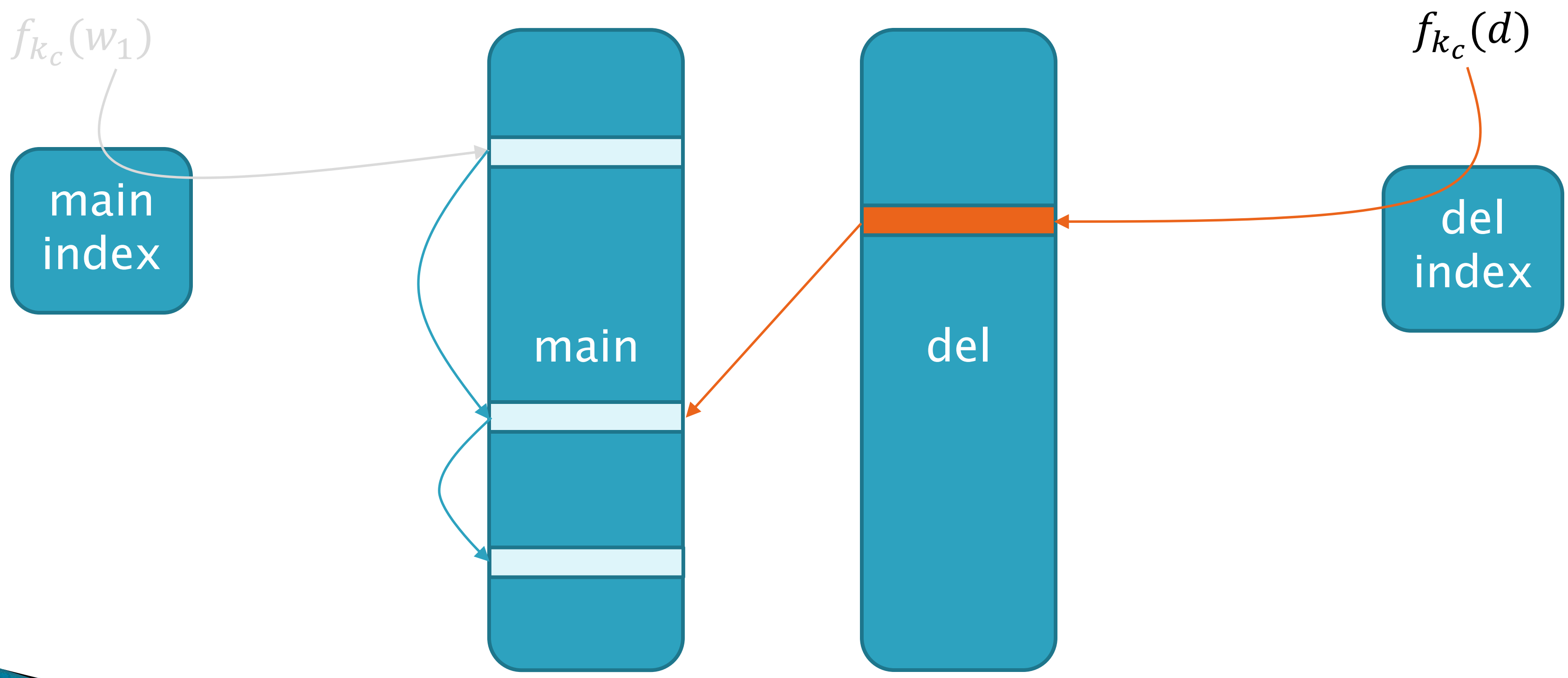
Delete a Document

- ⟨doc tokens⟩, doc key, ⟨freelist tokens⟩, count
  - per word: ⟨freelist mask⟩



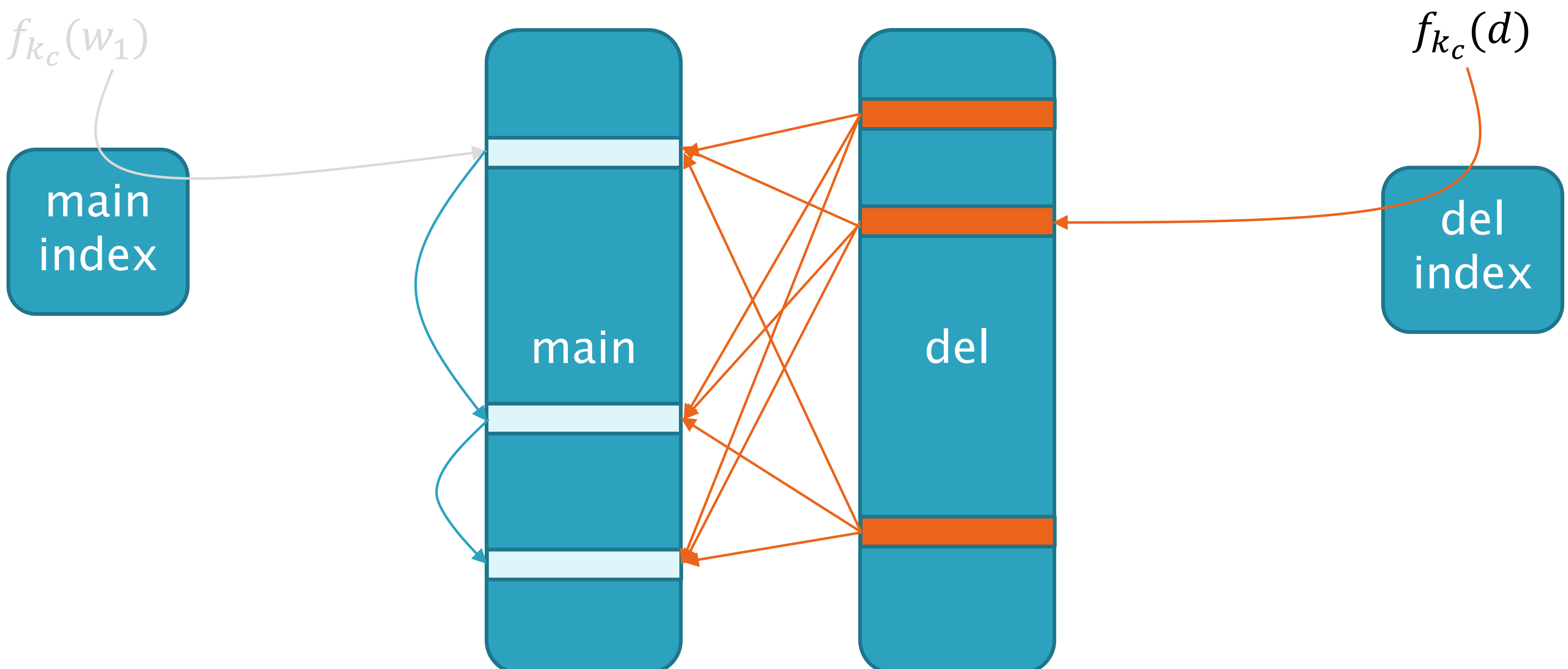
Delete a Document

- ⟨doc tokens⟩, doc key, ⟨freelist tokens⟩, count
  - per word: ⟨freelist mask⟩



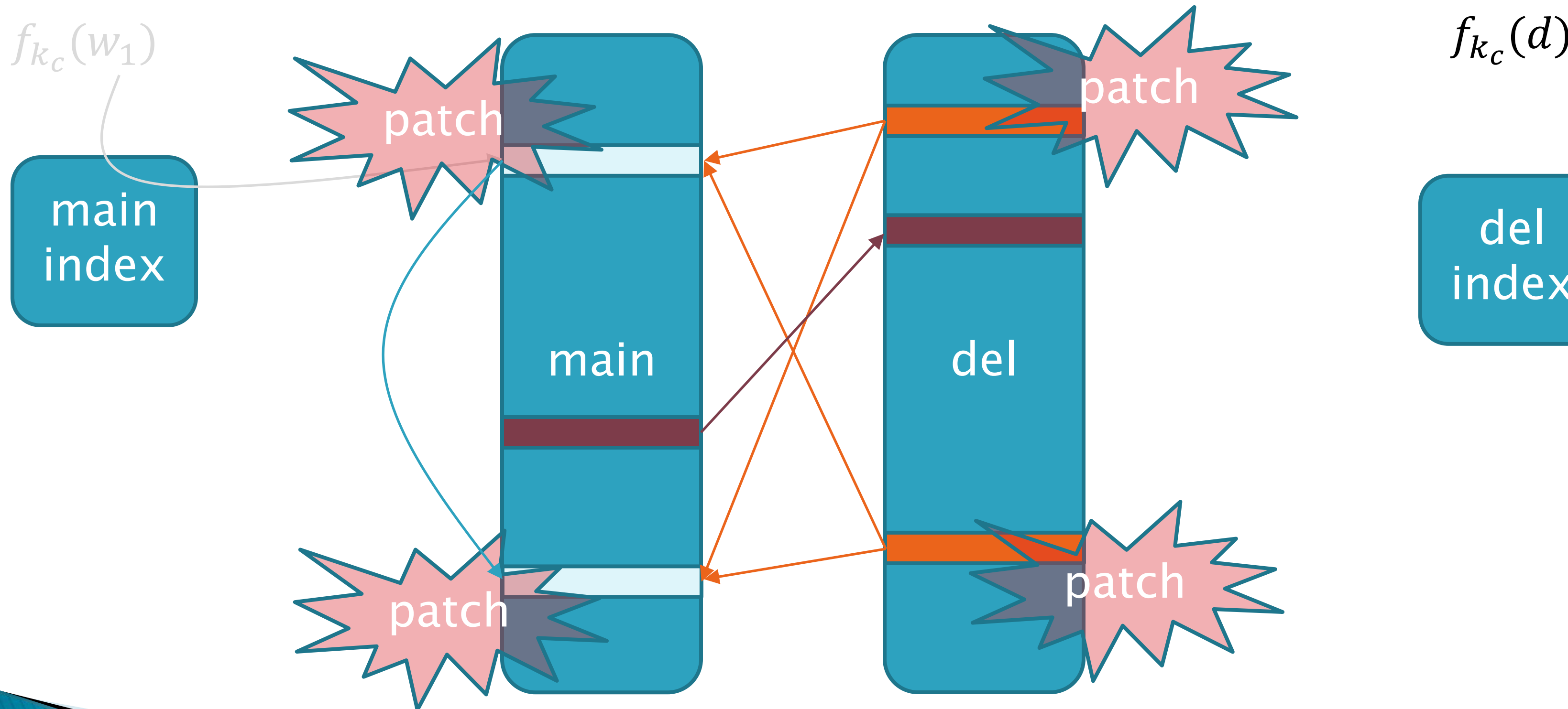
Delete a Document

- ⟨doc tokens⟩, doc key, ⟨freelist tokens⟩, count
  - per word: ⟨freelist mask⟩



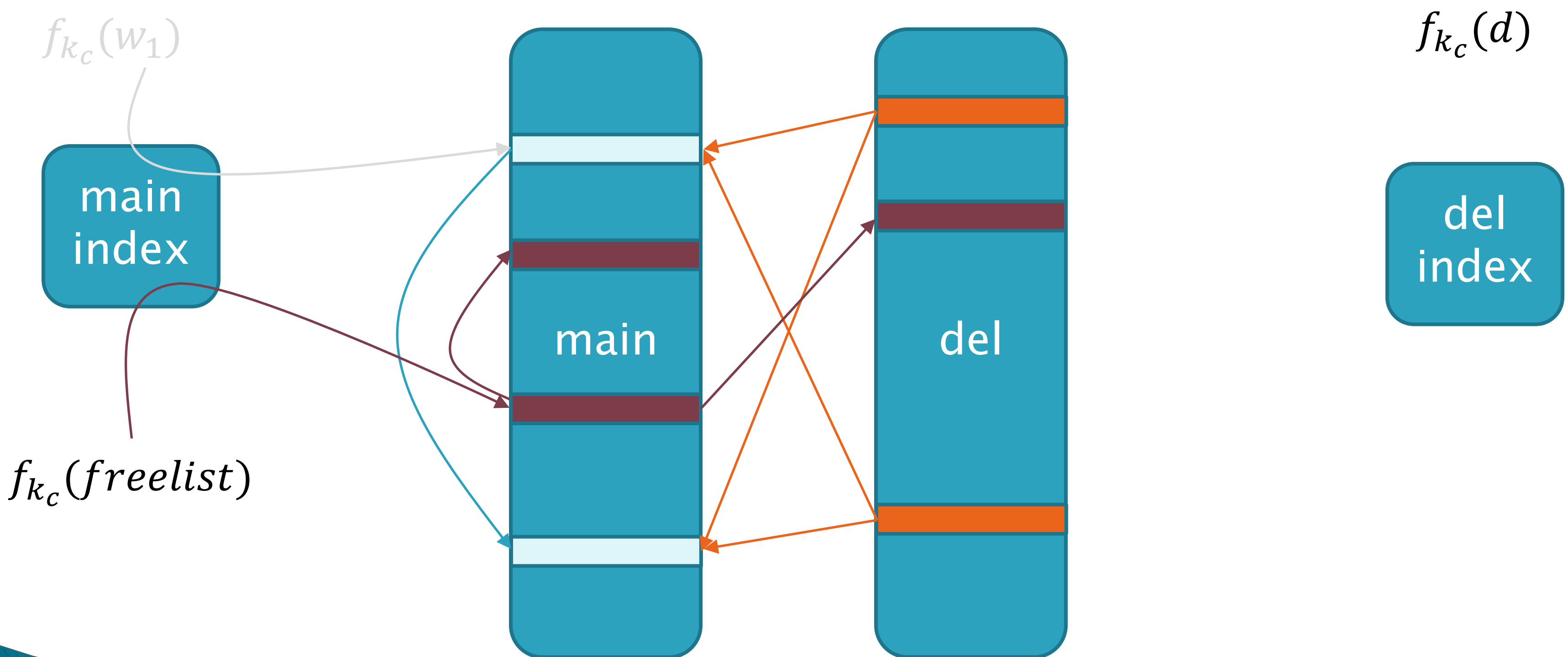
Delete a Document

- ⟨doc tokens⟩, doc key, ⟨freelist tokens⟩, count
  - per word: ⟨freelist mask⟩



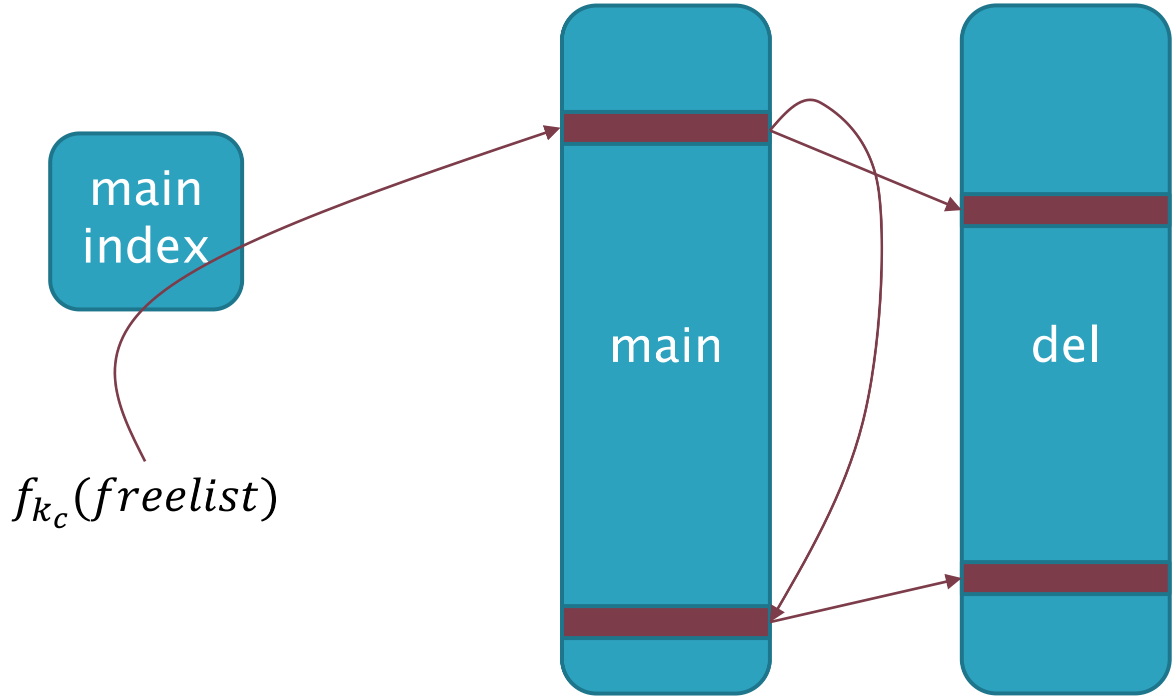
Delete a Document

- ⟨doc tokens⟩, doc key, ⟨freelist tokens⟩, count
  - per word: ⟨freelist mask⟩



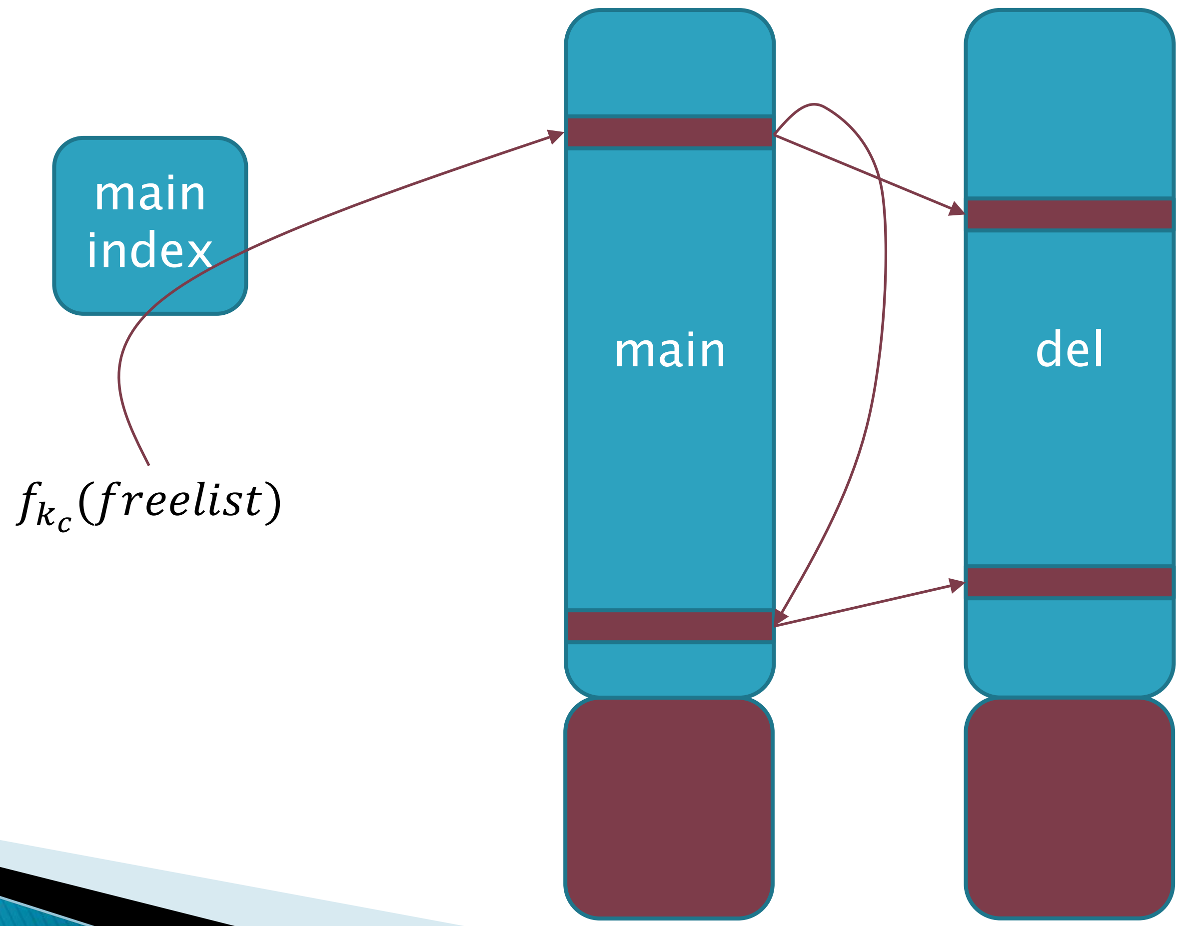
Index Extension

- Index size is fixed at generation time
  - So, add to free list for expansion



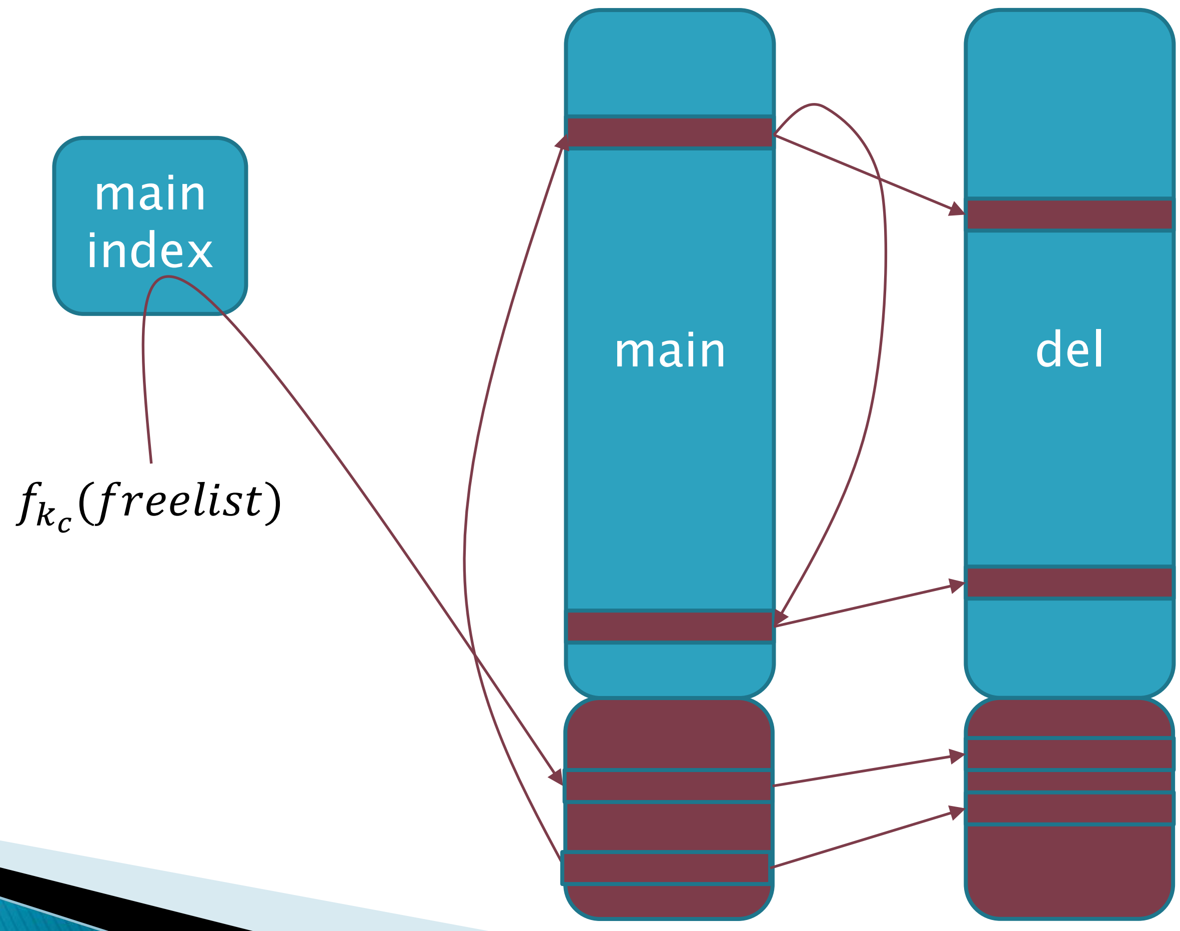
Index Extension

- Index size is fixed at generation time
  - So, add to free list for expansion



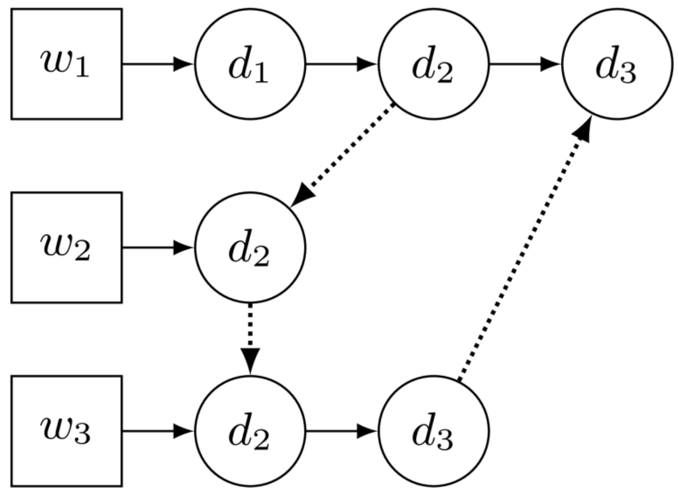
Index Extension

- Index size is fixed at generation time
  - So, add to free list for expansion



A Small Example: Indexes

Index



Main Index M

$f_{k_c}(w_1) \longrightarrow (4 \,||\, 1) \oplus f_{k_b}(w_1)$

$f_{k_c}(w_2) \longrightarrow (0 \,||\, 2) \oplus f_{k_b}(w_2)$

$f_{k_c}(w_3) \longrightarrow (5 \,||\, 0) \oplus f_{k_b}(w_3)$

$f_{k_c}(\mathsf{free}) \longrightarrow 6 \oplus f_{k_b}(\mathsf{free})$

Deletion Index I

$f_{k_c}(d_1) \longrightarrow 1 \oplus f_{k_b}(d_1)$

$f_{k_c}(d_2) \longrightarrow 5 \oplus f_{k_b}(d_2)$

$f_{k_c}(d_3) \longrightarrow 4 \oplus f_{k_b}(d_3)$

A Small Example: Arrays

Main Index M

$f_{k_c}(w_1) \longrightarrow (4 \,||\, 1) \oplus f_{k_b}(w_1)$

$f_{k_c}(w_2) \longrightarrow (0 \,||\, 2) \oplus f_{k_b}(w_2)$

$f_{k_c}(w_3) \longrightarrow (5 \,||\, 0) \oplus f_{k_b}(w_3)$

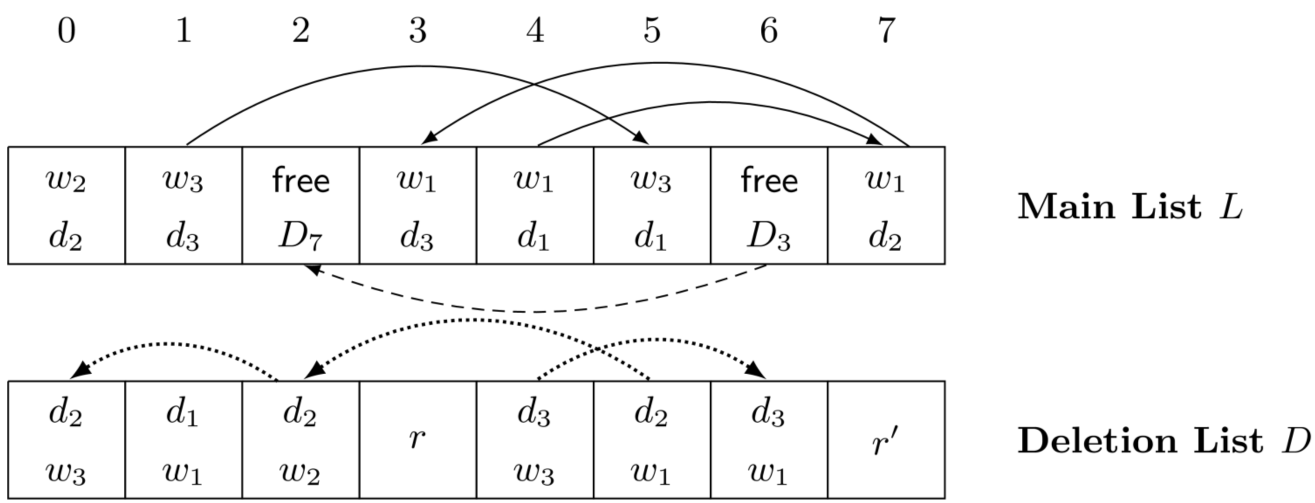
$f_{k_c}(\mathsf{free}) \longrightarrow 6 \oplus f_{k_b}(\mathsf{free})$

Deletion Index I

$f_{k_c}(d_1) \longrightarrow 1 \oplus f_{k_b}(d_1)$

$f_{k_c}(d_2) \longrightarrow 5 \oplus f_{k_b}(d_2)$

$f_{k_c}(d_3) \longrightarrow 4 \oplus f_{k_b}(d_3)$



Word-Based Deletion

- Deletion index uses doc/word pairs:
  - No lists of words per doc
  - $f_{k_c}(d,w) \rightarrow r, r', r'', \langle x,p,n \rangle \oplus f_{k_{d,w}}(r), f_{k_c}\left(d_p, w_p\right) \oplus f_{k_{d,w}}(r'), f_{k_c}(d_n, w_n) \oplus f_{k_{d,w}}(r'')$
- Algorithms similar
  - Search identical
  - Add puts new word on front of list
  - Delete patches to pull word out of list
  - Extension identical

Tradeoffs

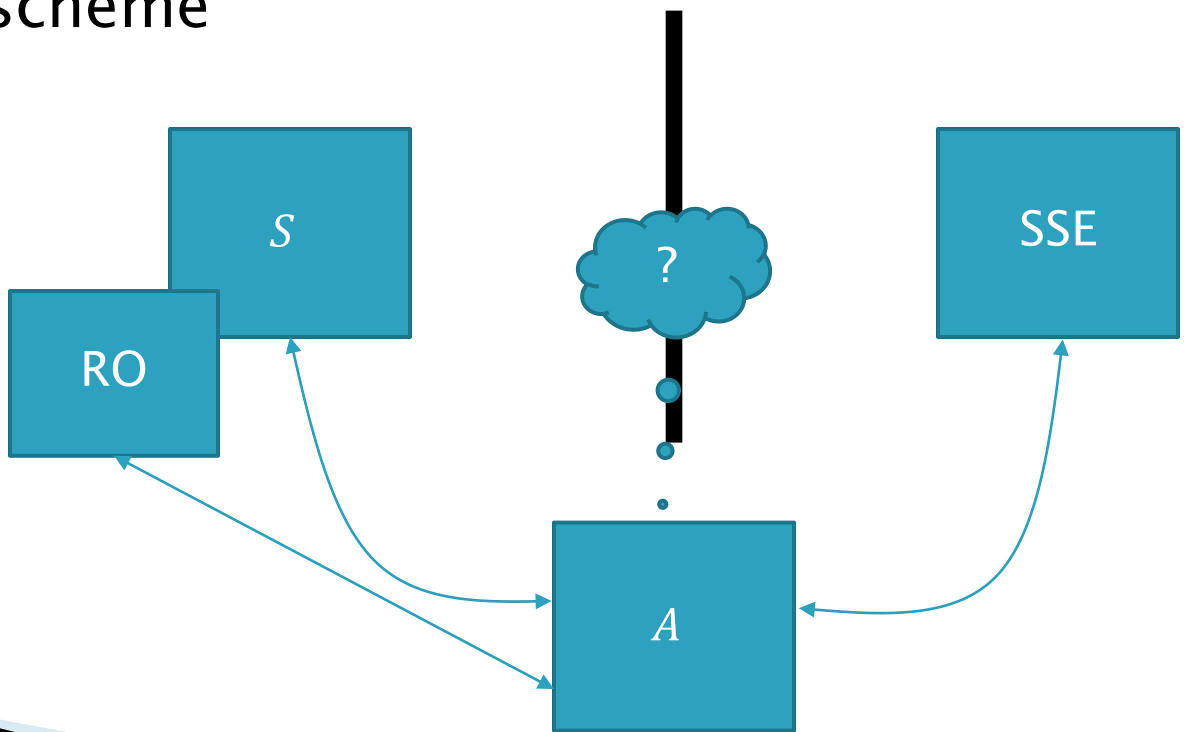
- Word-Based Update
  - Update token linear in number of word changes
  - Hides number of unique words in document
  - Uses less space for index
  - But requires keeping track of diffs on disk
- Doc-Based Update
  - Stateless for client (except freelist count)
  - But reveals the unique words in old and new docs
- We currently use Doc-Based Update
  - Cost of keeping diffs outweighs value of hiding

Overview

- Introduction
- Dynamic SSE Protocols
- Security Proofs
- Implementation

Security Proofs

- Adaptive Simulatability
  - Σ = (Gen, Index, TrapS, Search, Retrieve, TrapA, Add, TrapD, Delete, ExtendIndex) is a dynamic SSE scheme

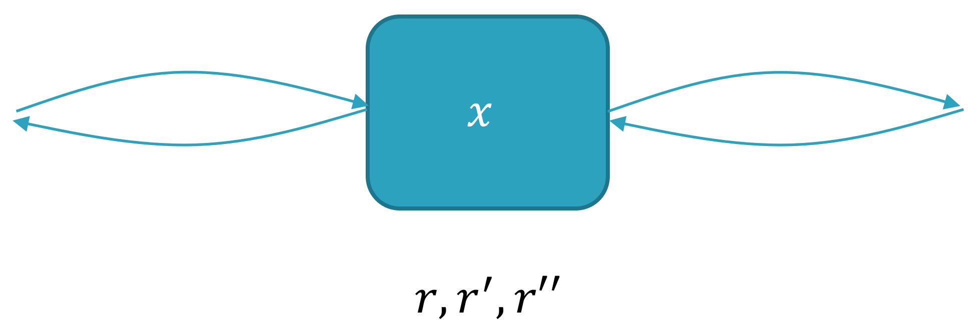


Leakage

- Searchable Symmetric Encryption leaks info
  - Query pattern: unique terms and result counts
  - Access pattern: which documents are retrieved
- Our algorithm leaks a little more
  - unique ID for words in added and deleted docs
    - Update pattern: add to existing, pos of delete
  - tail of the free list
  - amount of index expansion
  - when the index is full

Proof Outline

- Index Generation and Expansion: random
- Search: given number of results
  - If seen search (+ any updates), then repeat
  - Otherwise, choose a random index entry
  - Provide random unused location for first element
  - Choose unused locations for other elements
  - Program random oracle to “decrypt” list (k_w)



Proof Outline

- Index Generation and Expansion: random
- Search: given number of results
  - If seen search (+ any updates), then repeat
  - Otherwise, choose a random index entry
  - Provide random unused location for first element
  - Choose unused locations for other elements
  - Program random oracle to “decrypt” list (k_w)

$$f_{k_w}(r) = \langle p, n \rangle \oplus r'$$

$$r' \oplus f_{k_w}(r) = r' \oplus \langle p, n \rangle \oplus r' = \langle p, n \rangle$$

r, r', r''

Proof Outline: Add and Delete

- Add: given unique IDs of added words
  - Find random locations and setup freelist tokens
  - Choose random index entry and get word tokens
  - Set masks to XOR to chosen pattern

- Delete: given unique IDs of deleted words
  - Choose deletion locations (from prev or random)
  - Choose index entry to delete (from prev or random)
  - Program random oracle to decrypt chosen pattern (k_d)

Overview

- Introduction
- Dynamic SSE Protocols
- Security Proofs
- Implementation

Performance

- Prototype doc-based scheme in C++
- Intel Xeon x64 2.26 GHz with Win 2008 R2
  - Zipf, Docs, Email datasets
  - 500k to 1.5M doc/word pairs
- Results
  - Generation (doc/word pair): 40 μs (c)
  - Search (doc): 8 μs (s)
  - Add (word): 35 μs (c), 2 μs (s)
  - Delete (word): 3 μs (c), 24 μs (s)

Related SSE Schemes

- [CGKO06]
  - Efficient search
  - Provides an adaptive scheme in plain model
  - Doesn't provide any update properties
- [SLDH09]
  - Efficient update via XOR encryption
  - Uses padded lists: linear in number of docs
  - Large storage cost: $O(|w|\ |d|)$

Conclusions

- Dynamic SSE algorithms
- Add and Delete use XOR encryption to modify index
- Practical for real-world applications
- Can trade off leakage for server operations