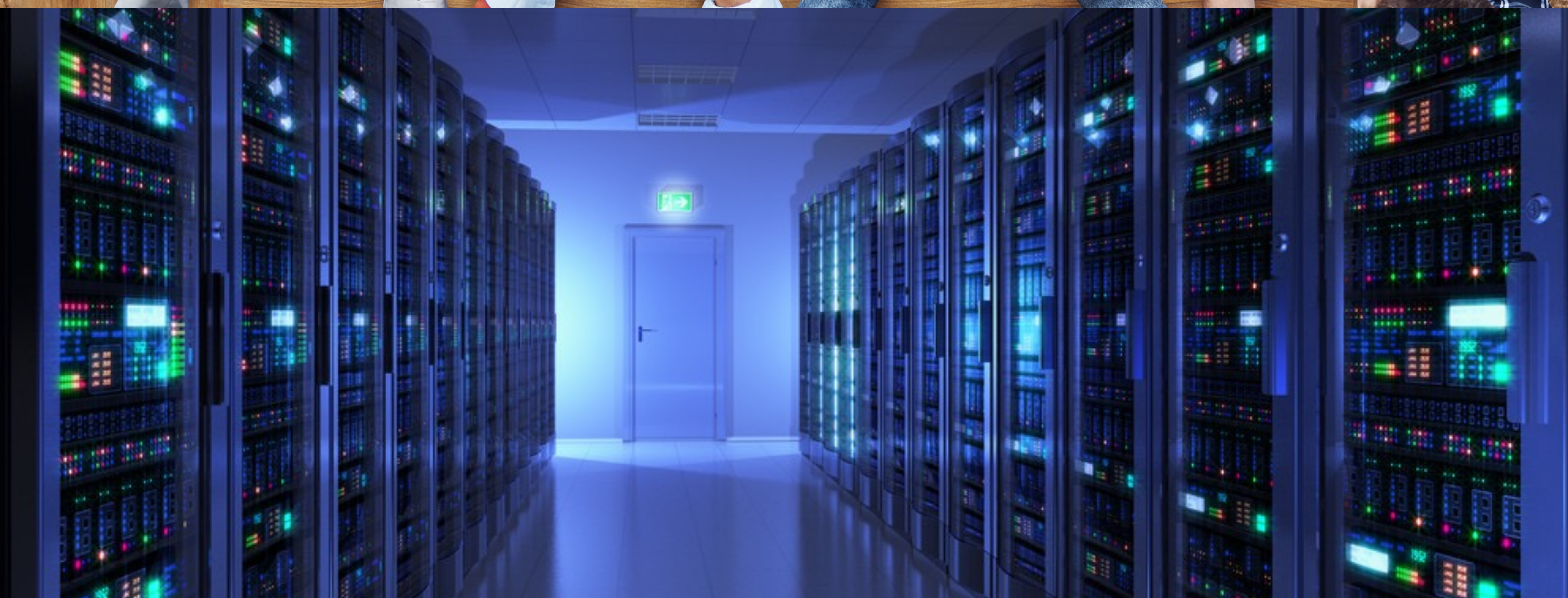
Space- and Time-Efficient Data Structures for Massive Datasets

Giulio Ermanno Pibiri
giulio.pibiri@di.unipi.it

Supervisor
Rossano Venturini

Department of Computer Science
University of Pisa

15/11/2018

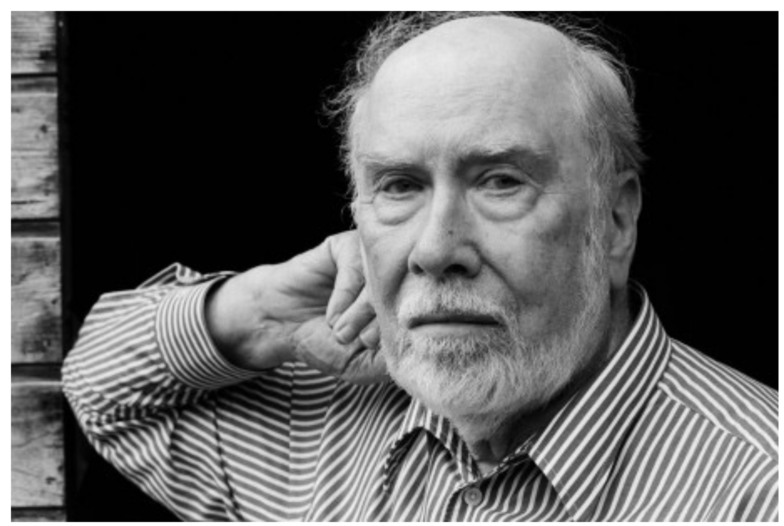

Statistics
Progress
Growth & Decline
Statistical
Analysis


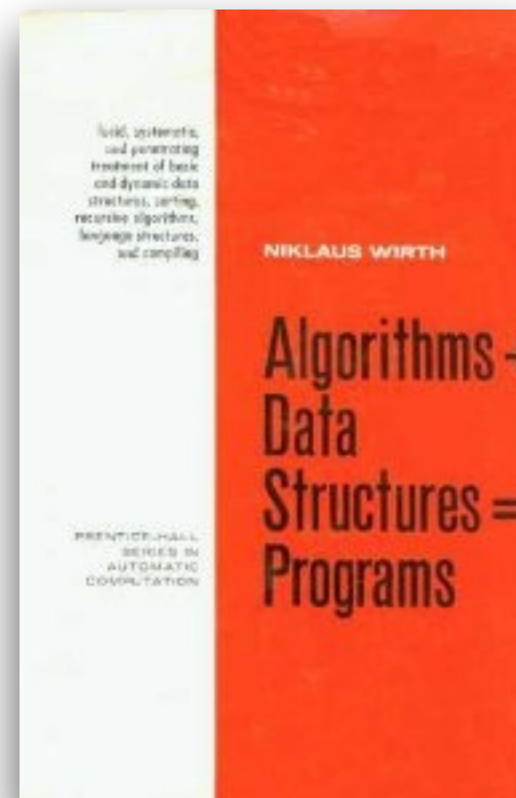
Evidence

The increase of information
does **not** scale with technology.

Evidence

The increase of information
does **not** scale with technology.





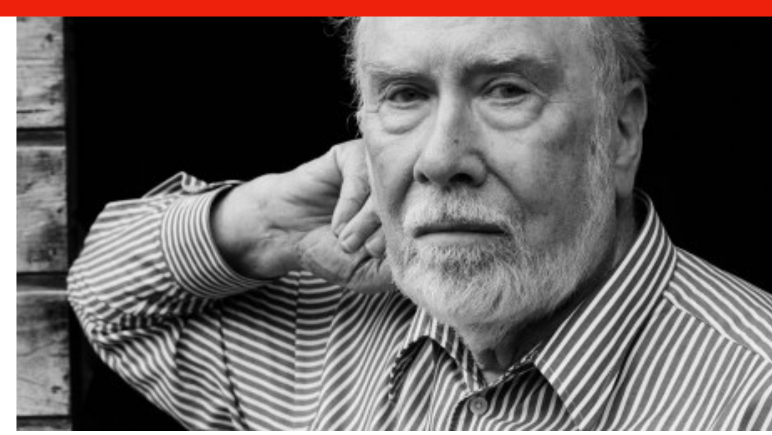
“Software is getting slower more rapidly than hardware becomes faster.”

Niklaus Wirth, A Plea for Lean Software

Evidence

The increase of information
does **not** scale with technology.

Even more relevant today!



“Software is getting slower more rapidly than hardware becomes faster.”

Niklaus Wirth, A Plea for Lean Software

Scenario

Data structures

PERFORMANCE

how quickly a program does its work - **faster** work

time ✓
✗ space

↔

Algorithms

EFFICIENCY

how much work is required by a program - **less** work

Scenario

Data structures

PERFORMANCE

how quickly a program does its work - **faster** work

time ✓
✗ space

Algorithms

EFFICIENCY

how much work is required by a program - **less** work

?

Data compression

✓ space
time ✗

The dichotomy problem

Small vs. fast?

The dichotomy problem

Small vs. fast?

Choose one.

The dichotomy problem

Small vs. fast?

Choose one.

NO

High level thesis

Data Structures + Data Compression ➡ Fast Algorithms

Design **space-efficient** *ad-hoc* data structures,
both from a theoretical *and* practical perspective,
that support **fast data extraction**.

Data Compression & Fast Retrieval *together*.

Achieved results

Clustered Elias-Fano Indexes
Giulio Ermanno Pibiri and Rossano Venturini
ACM Transactions on Information Systems (TOIS)
Full paper, 34 pages, 2017.

Journal paper

Dynamic Elias-Fano Representation
Giulio Ermanno Pibiri and Rossano Venturini
Annual Symposium on Combinatorial Pattern Matching (CPM)
Full paper, 14 pages, 2017.

Conference paper

Variable-Byte Encoding is Now Space-Efficient Too
Giulio Ermanno Pibiri and Rossano Venturini
arXiv (CoRR), April 2018.
Submitted to IEEE Transactions on Knowledge and Data Engineering (TKDE)
Full paper, 12 pages, 2018.

Journal paper

Fast Dictionary-based Compression for Inverted Indexes
Giulio Ermanno Pibiri, Matthias Petri and Alistair Moffat
ACM Conference on Web Search and Data Mining (WSDM)
Full paper, 9 pages, 2019.

Conference paper

Efficient Data Structures for Massive N-Gram Datasets
Giulio Ermanno Pibiri and Rossano Venturini
ACM Conference on Research and Development in Information Retrieval (SIGIR)
Full paper, 10 pages, 2017.

Conference paper

Handling Massive N-Gram Datasets Efficiently
Giulio Ermanno Pibiri and Rossano Venturini
ACM Transactions on Information Systems (TOIS), 2018. To appear.
Full paper, 41 pages, 2018.

Journal paper

Achieved results

Clustered Elias-Fano Indexes Journal paper
Giulio Ermanno Pibiri and Rossano Venturini
ACM Transactions on Information Systems (TOIS)
Full paper, 34 pages, 2017.

Dynamic Elias-Fano Representation Conference paper
Giulio Ermanno Pibiri and Rossano Venturini
Annual Symposium on Combinatorial Pattern Matching (CPM)
Full paper, 14 pages, 2017.

Variable-Byte Encoding is Now Space-Efficient Too
Giulio Ermanno Pibiri and Rossano Venturini Journal paper
arXiv (CoRR), April 2018.
Submitted to IEEE Transactions on Knowledge and Data Engineering (TKDE)
Full paper, 12 pages, 2018.

Fast Dictionary-based Compression for Inverted Indexes
Giulio Ermanno Pibiri, Matthias Petri and Alistair Moffat
ACM Conference on Web Search and Data Mining (WSDM) Conference paper
Full paper, 9 pages, 2019.

integer sequences

Efficient Data Structures for Massive N-Gram Datasets Conference paper
Giulio Ermanno Pibiri and Rossano Venturini
ACM Conference on Research and Development in Information Retrieval (SIGIR)
Full paper, 10 pages, 2017.

Handling Massive N-Gram Datasets Efficiently Journal paper
Giulio Ermanno Pibiri and Rossano Venturini
ACM Transactions on Information Systems (TOIS), 2018. To appear.
Full paper, 41 pages, 2018.

Achieved results

Clustered Elias-Fano Indexes **Journal paper**
Giulio Ermanno Pibiri and Rossano Venturini
ACM Transactions on Information Systems (TOIS)
Full paper, 34 pages, 2017.

Dynamic Elias-Fano Representation **Conference paper**
Giulio Ermanno Pibiri and Rossano Venturini
Annual Symposium on Combinatorial Pattern Matching (CPM)
Full paper, 14 pages, 2017.

Variable-Byte Encoding is Now Space-Efficient Too **Journal paper**
Giulio Ermanno Pibiri and Rossano Venturini
arXiv (CoRR), April 2018.
Submitted to IEEE Transactions on Knowledge and Data Engineering (TKDE)
Full paper, 12 pages, 2018.

Fast Dictionary-based Compression for Inverted Indexes **Conference paper**
Giulio Ermanno Pibiri, Matthias Petri and Alistair Moffat
ACM Conference on Web Search and Data Mining (WSDM)
Full paper, 9 pages, 2019.

integer sequences

Efficient Data Structures for Massive N-Gram Datasets **Conference paper**
Giulio Ermanno Pibiri and Rossano Venturini
ACM Conference on Research and Development in Information Retrieval (SIGIR)
Full paper, 10 pages, 2017.

Handling Massive N-Gram Datasets Efficiently **Journal paper**
Giulio Ermanno Pibiri and Rossano Venturini
ACM Transactions on Information Systems (TOIS), 2018. To appear.
Full paper, 41 pages, 2018.

short strings

Problem 1

Consider a sorted integer sequence.

Problem 1

Consider a sorted integer sequence.

How to represent it as a bit-vector where each original integer is uniquely-decodable, using **as few as possible** bits?

How to maintain **fast decompression speed**?

Problem 1

Consider a sorted integer sequence.

How to represent it as a bit-vector where each original integer is uniquely-decodable, using **as few as possible** bits?

How to maintain **fast decompression speed**?

This is a difficult problem that has been studied since the the '60.

Applications

Inverted indexes

Databases

RDF indexing

E-Commerce

Geo-spatial data

Graph-compression

Applications

Inverted indexes

Google YAHOO!

bing

Databases

IBM Dropbox

ORACLE

RDF indexing

ontotext

E-Commerce

ebay amazon

Geo-spatial data

Pokémon GO

Graph-compression

facebook

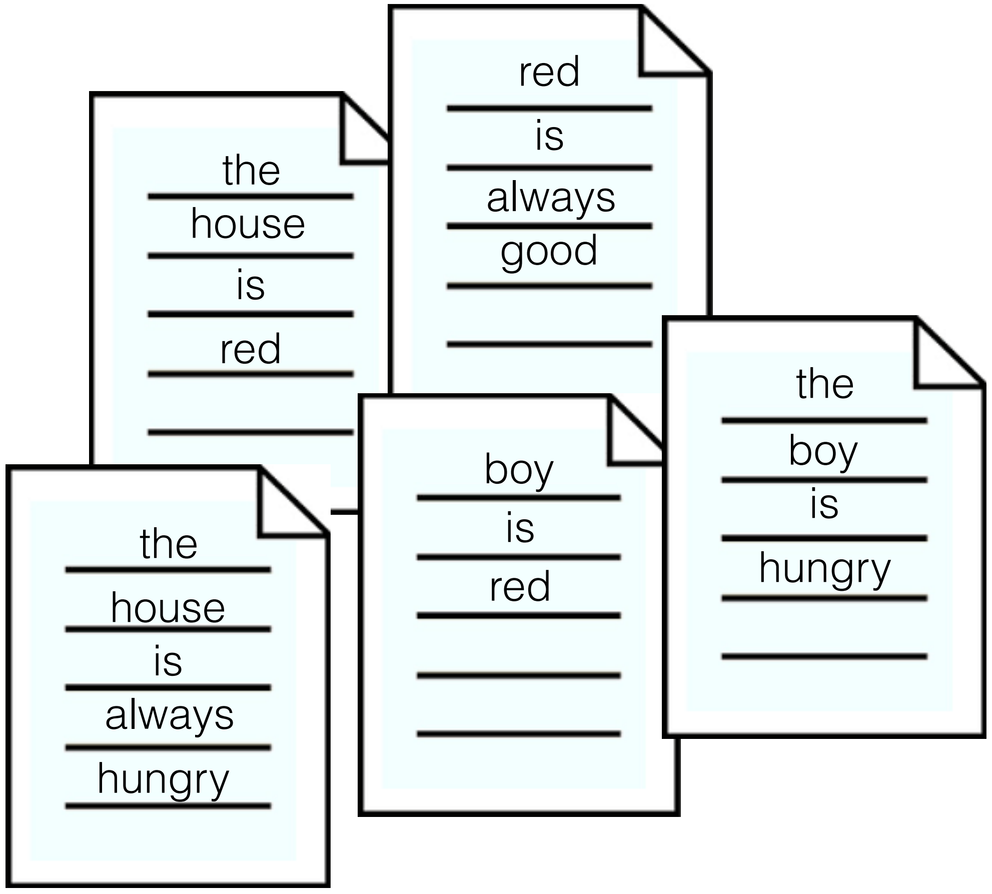
LinkedIn

Inverted indexes

The inverted index is the *de-facto* data structure at the basis of every large-scale retrieval system.

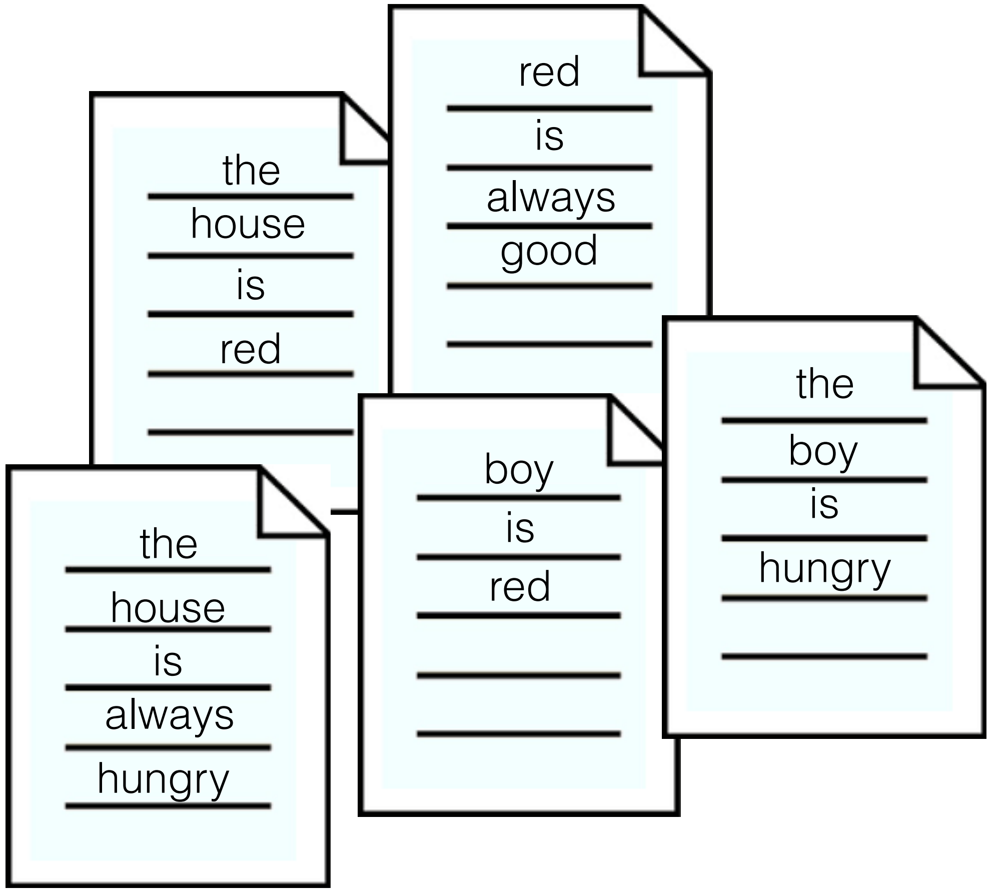
Inverted indexes

The inverted index is the *de-facto* data structure at the basis of every large-scale retrieval system.



Inverted indexes

The inverted index is the *de-facto* data structure at the basis of every large-scale retrieval system.

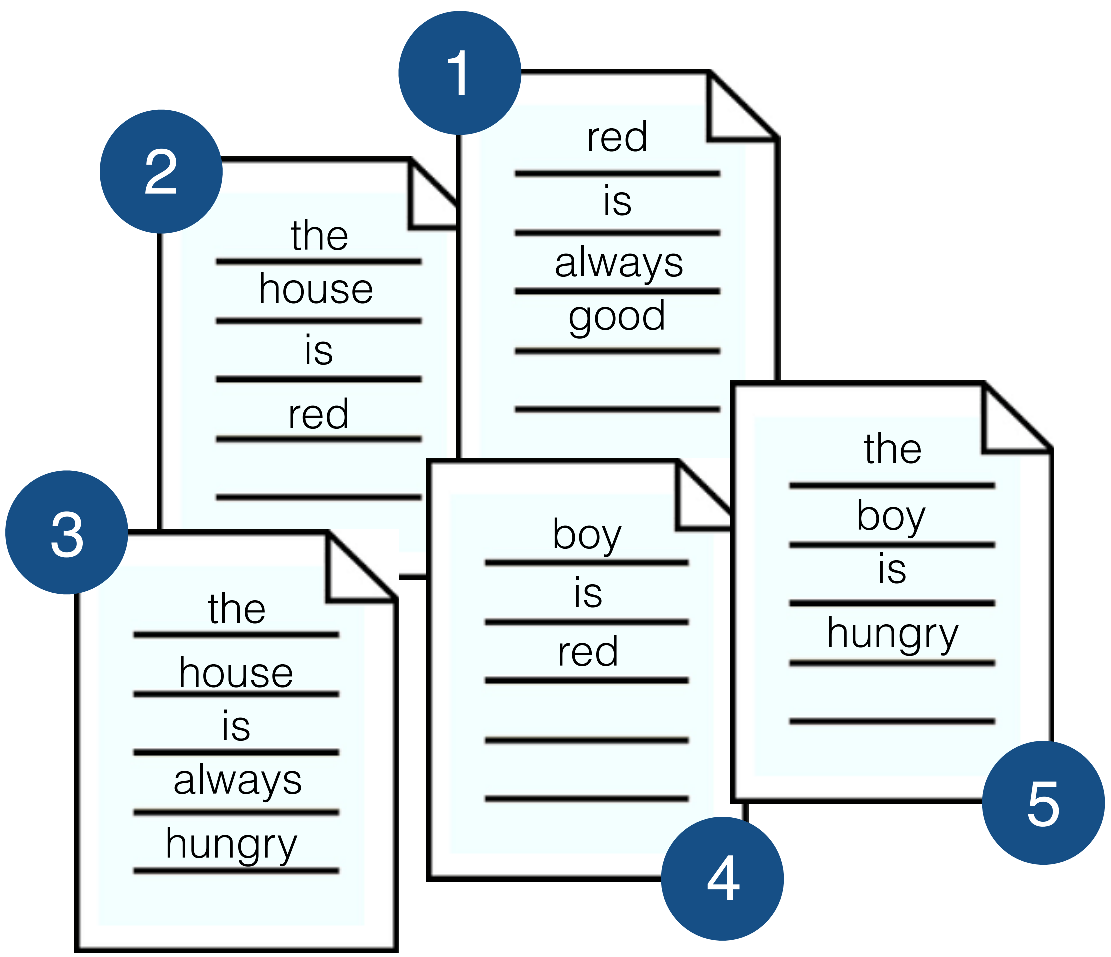


t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8

{always, boy, good, house, hungry, is, red, the}

Inverted indexes

The inverted index is the *de-facto* data structure at the basis of every large-scale retrieval system.

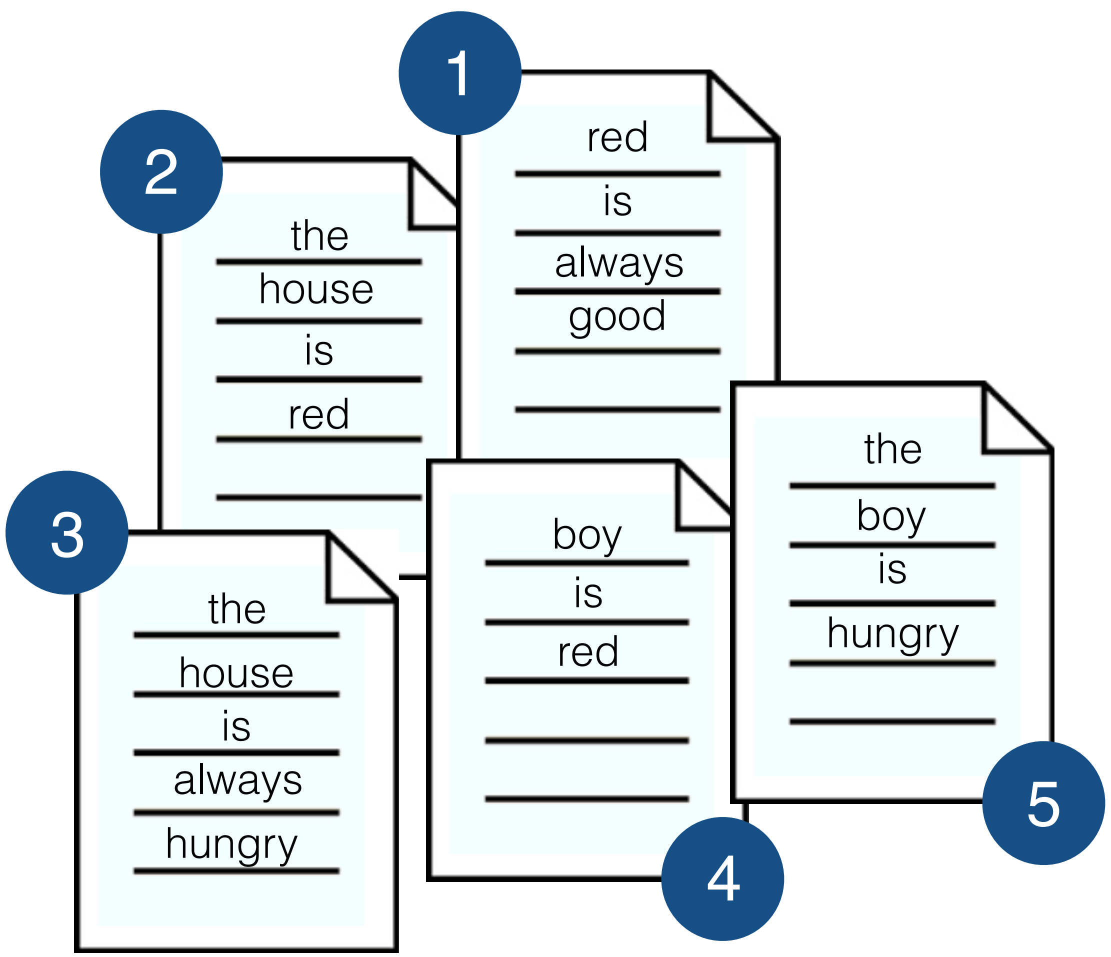


t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8

{always, boy, good, house, hungry, is, red, the}

Inverted indexes

The inverted index is the *de-facto* data structure at the basis of every large-scale retrieval system.



t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8

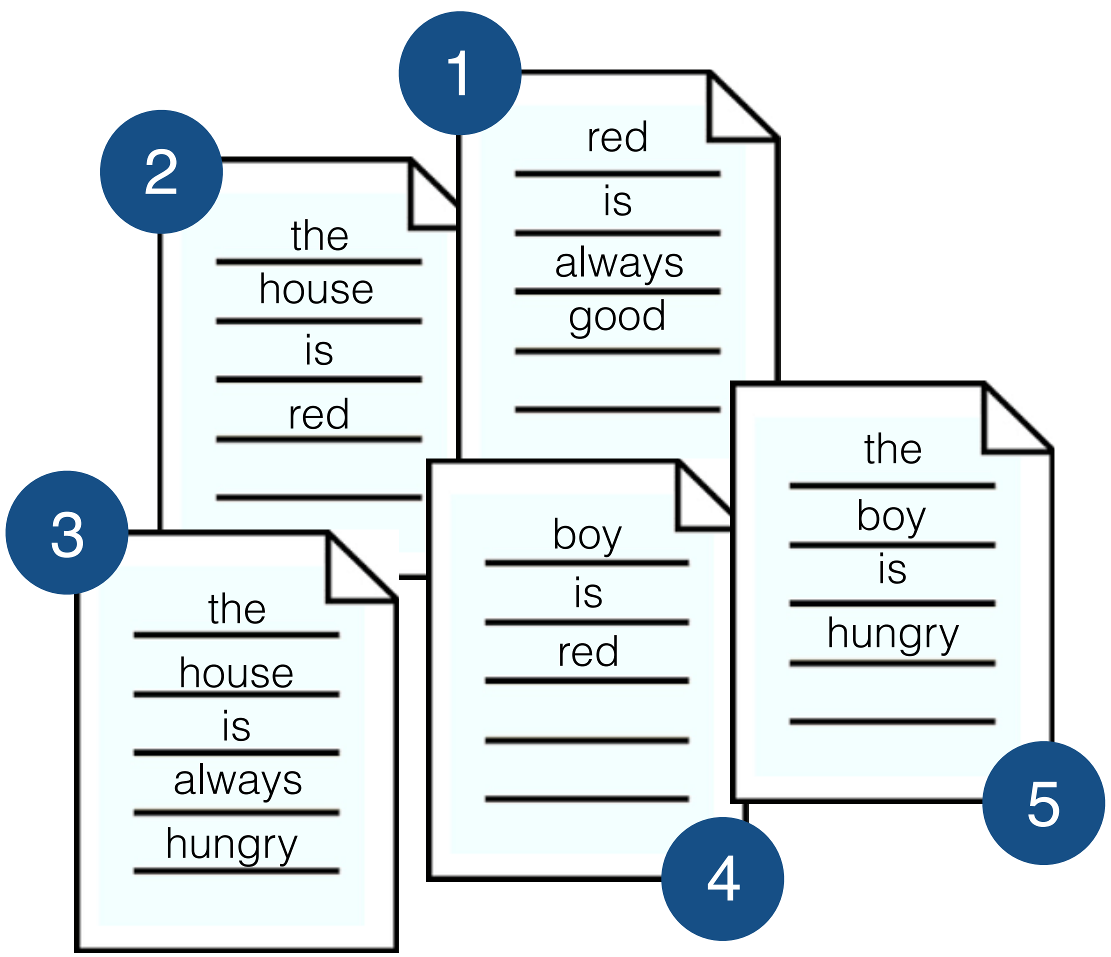
{always, boy, good, house, hungry, is, red, the}

→

L_{t_1}=[1, 3]
L_{t_2}=[4, 5]
L_{t_3}=[1]
L_{t_4}=[2, 3]
L_{t_5}=[3, 5]
L_{t_6}=[1, 2, 3, 4, 5]
L_{t_7}=[1, 2, 4]
L_{t_8}=[2, 3, 5]

Inverted indexes

The inverted index is the *de-facto* data structure at the basis of every large-scale retrieval system.



t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8

{always, boy, good, house, hungry, is, red, the}

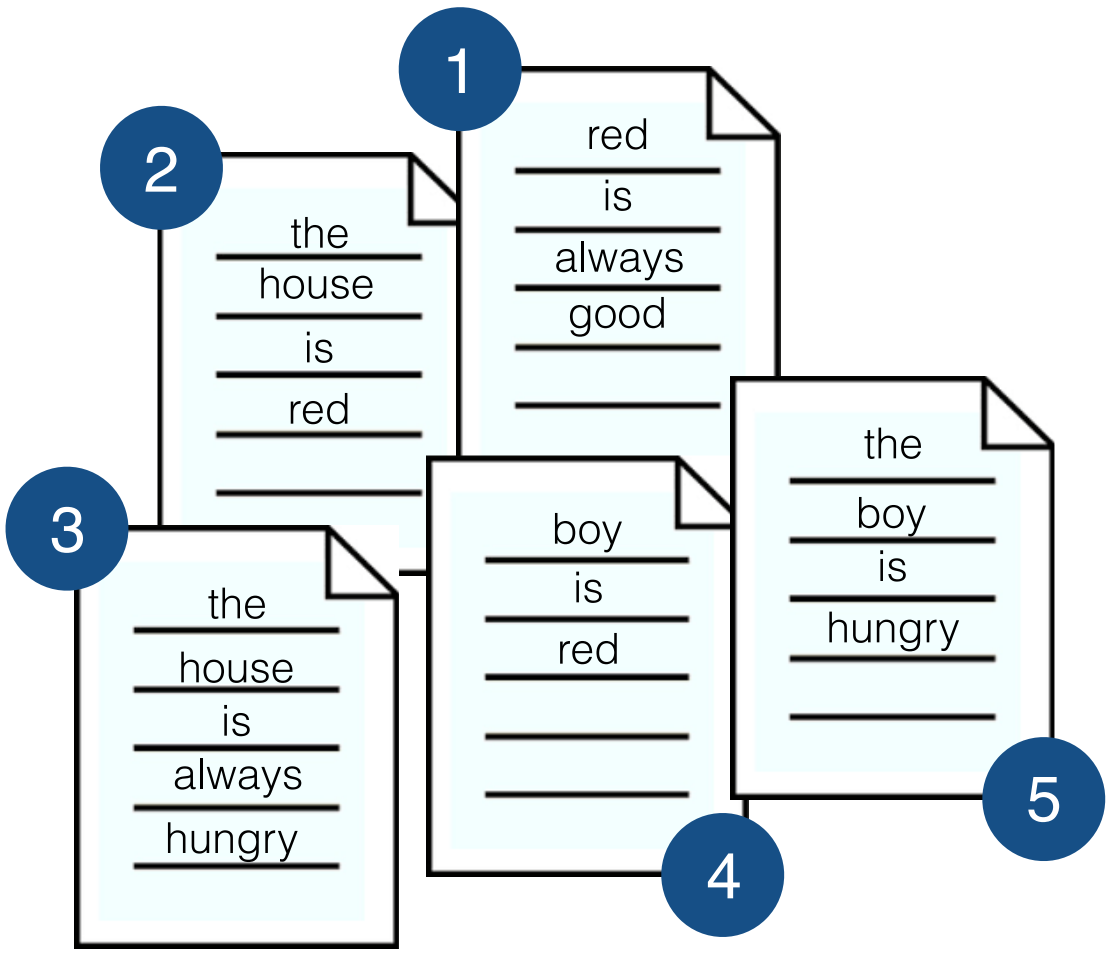
$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

Inverted indexes

Inverted indexes owe their popularity to the
efficient resolution of queries, such as:
"return all documents in which terms $\{t_1,\ldots,t_k\}$ occur".

Inverted indexes

Inverted indexes owe their popularity to the *efficient resolution of queries*, such as: "return all documents in which terms $\{t_1,\ldots,t_k\}$ occur".

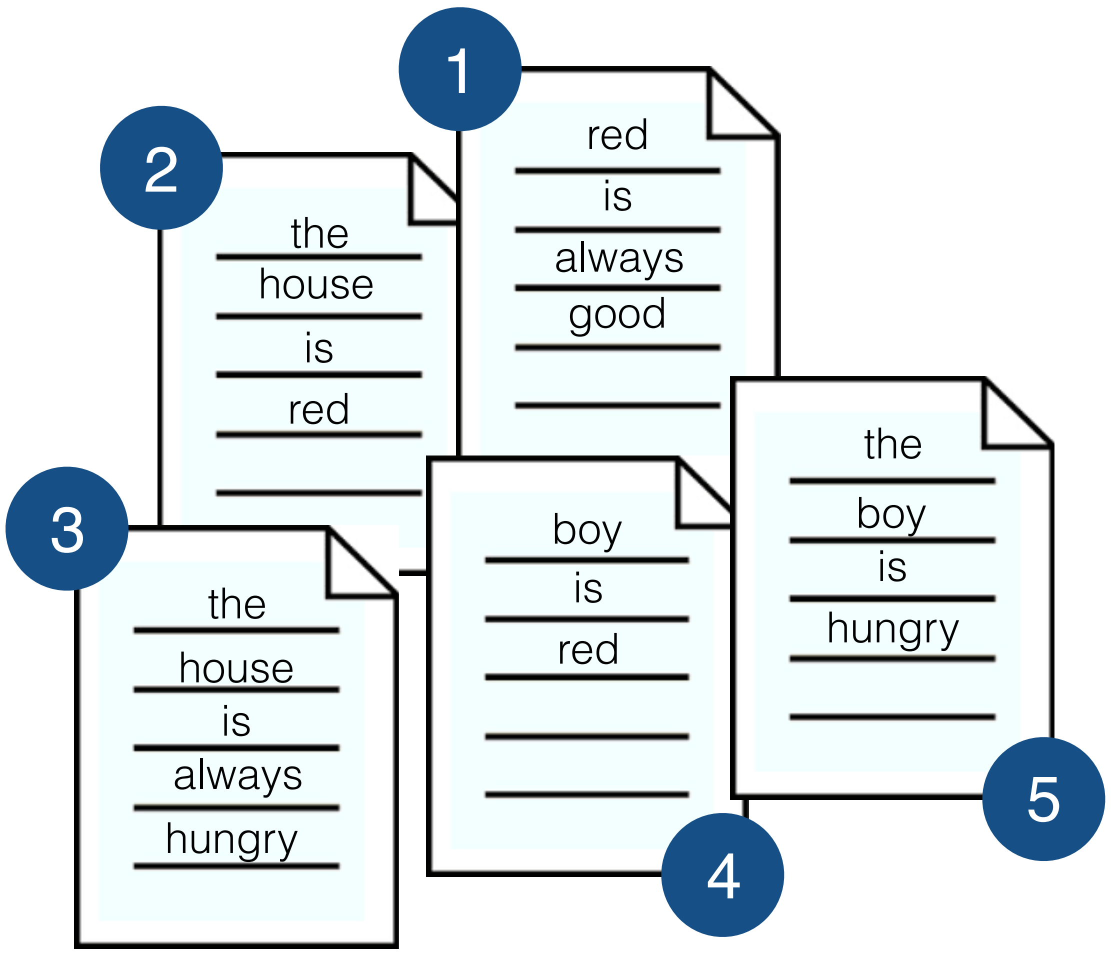


$$\{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

Inverted indexes

Inverted indexes owe their popularity to the *efficient resolution of queries*, such as: "return all documents in which terms $\{t_1,\ldots,t_k\}$ occur".



t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8

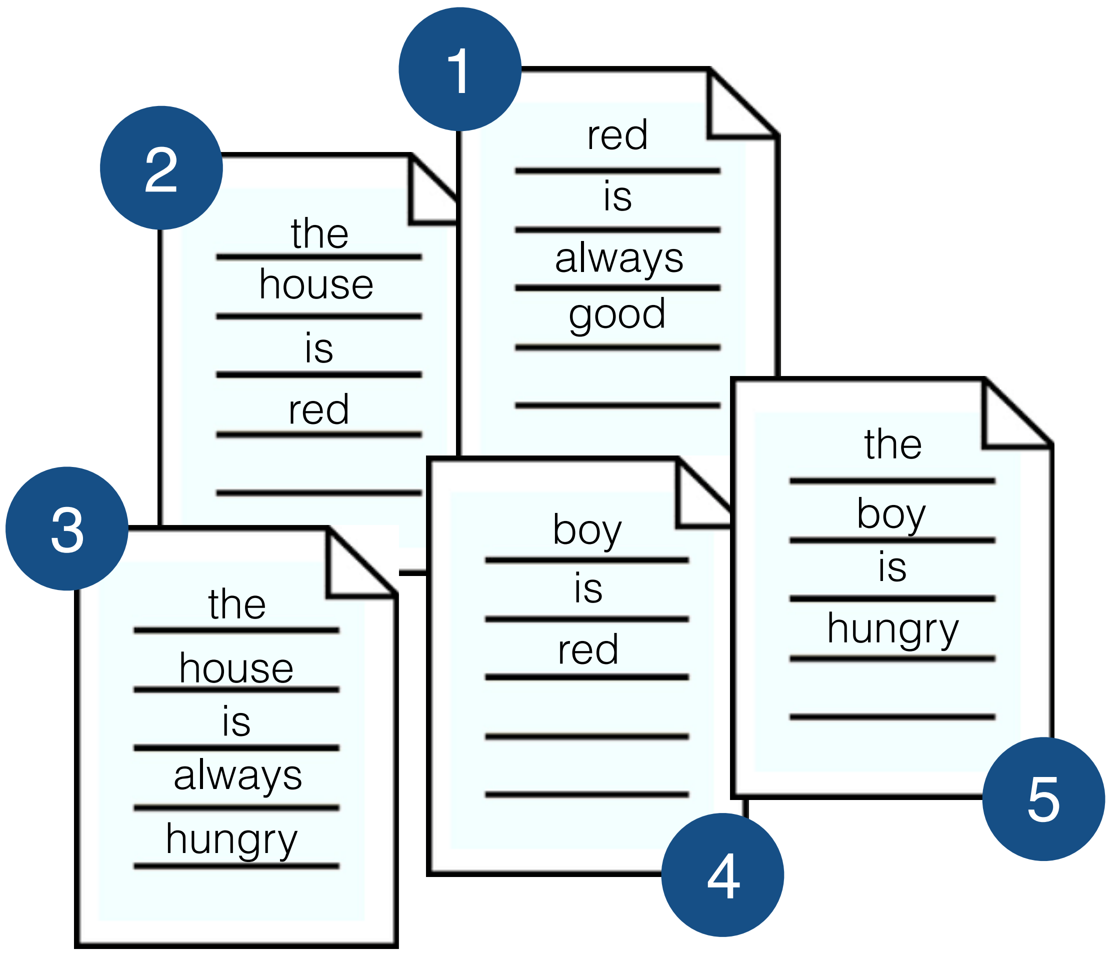
{always, boy, good, house, hungry, is, red, the}

$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

Q = {boy, is, the}

Inverted indexes

Inverted indexes owe their popularity to the
efficient resolution of queries, such as:
“return all documents in which terms $\{t_1,\ldots,t_k\}$ occur”.



t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8

{always, boy, good, house, hungry, is, red, the}

L_{t_1}=[1, 3]
L_{t_2}=[4, 5]
L_{t_3}=[1]
L_{t_4}=[2, 3]
L_{t_5}=[3, 5]
L_{t_6}=[1, 2, 3, 4, 5]
L_{t_7}=[1, 2, 4]
L_{t_8}=[2, 3, 5]

Q = {boy, is, the}

Many solutions

Huge research corpora describing different **space/time** trade-offs.

- Elias Gamma and Delta **'70**
- Variable-Byte Family
- Binary Interpolative Coding
- Simple Family
- PForDelta
- QMX
- Elias-Fano
- Partitioned Elias-Fano **2014**

Many solutions

Huge research corpora describing different **space/time** trade-offs.

- Elias Gamma and Delta
- Variable-Byte Family
- Binary Interpolative Coding
- Simple Family
- PForDelta
- QMX
- Elias-Fano
- Partitioned Elias-Fano

‘70 ↓ **2014**

Space — **Binary Interpolative Coding** — ~**3X** smaller

← Spectrum →

Time — **Variable-Byte Family** — ~**4.5X** faster

Many solutions

Huge research corpora describing different **space/time** trade-offs.

- Elias Gamma and Delta
- Variable-Byte Family
- Binary Interpolative Coding
- Simple Family
- PForDelta
- QMX
- Elias-Fano
- Partitioned Elias-Fano

'70 → **2014**

Space | **Time**

Binary Interpolative Coding ← Spectrum → **Variable-Byte Family**

~**3X** smaller | ~**4.5X** faster

Google, Dropbox, amazon, IBM

Key research questions

Space

Binary Interpolative Coding

← Spectrum →

Time

Variable-Byte Family

~**3X** smaller

~**4.5X** faster

Key research questions

Space | **Time**

Binary Interpolative Coding ← Spectrum → **Variable-Byte Family**

~**3X** smaller | ~**4.5X** faster

1. Is it possible to design an encoding that is **as small as BIC** and **much faster**?

Key research questions

Space

Binary Interpolative Coding

~**3X** smaller

← Spectrum →

Time

Variable-Byte Family

~**4.5X** faster

1. Is it possible to design an encoding that is **as small as BIC** and **much faster**?

2. Is it possible to design an encoding that is **as fast as VByte** and **much smaller**?

Key research questions

Space

Binary Interpolative Coding

~**3X** smaller

← Spectrum →

Time

Variable-Byte Family

~**4.5X** faster

1. Is it possible to design an encoding that is **as small as BIC** and **much faster**?

2. Is it possible to design an encoding that is **as fast as VByte** and **much smaller**?

3. What about **both** objectives at the same time?!

Idea 1 - Clustered inverted indexes (TOIS ’17)

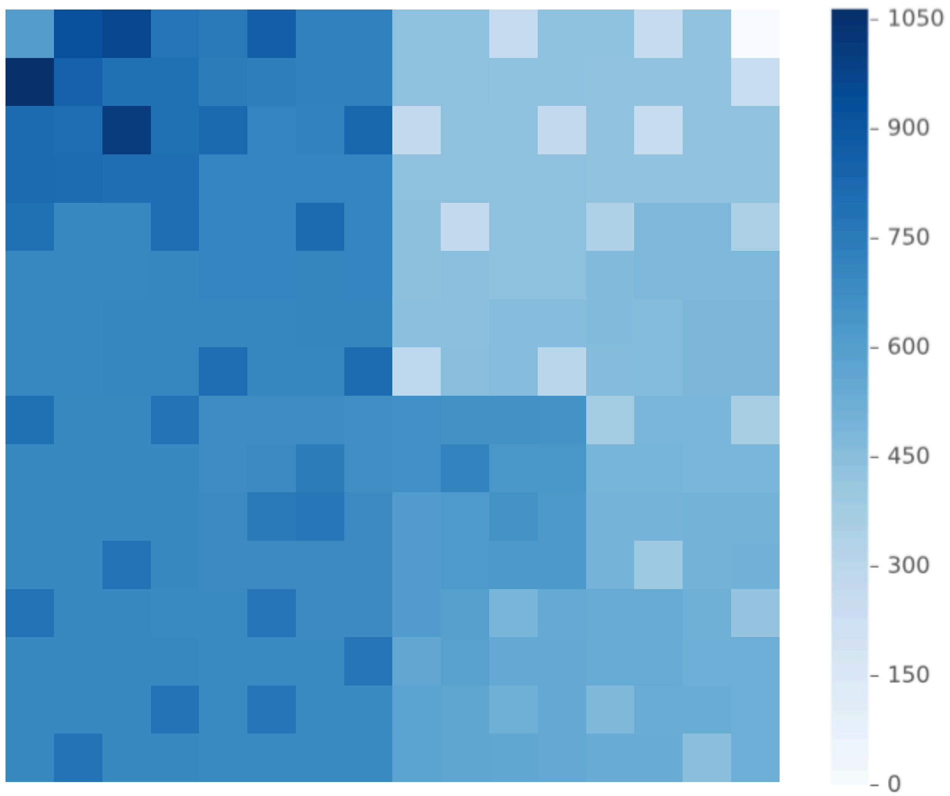
Every encoder represents each sequence **individually**.

No exploitation of redundancy.

Idea 1 - Clustered inverted indexes (TOIS '17)

Every encoder represents each sequence **individually**.

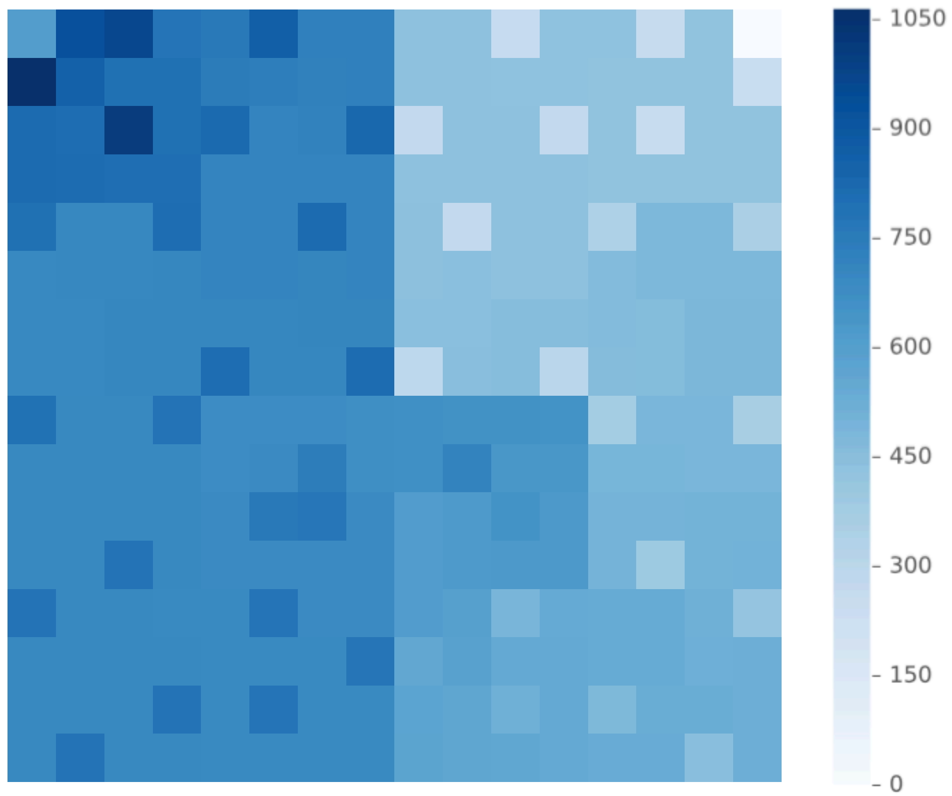
No exploitation of redundancy.



Idea 1 - Clustered inverted indexes (TOIS ’17)

Every encoder represents each sequence **individually**.

No exploitation of redundancy.

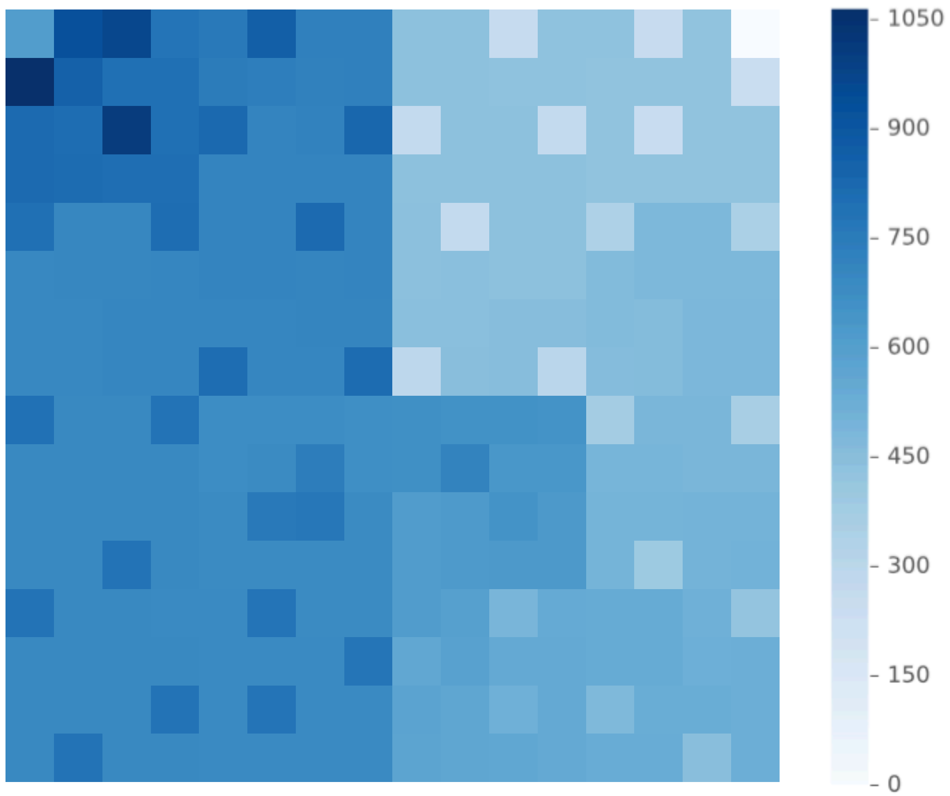


Encode **clusters** of inverted lists.

Idea 1 - Clustered inverted indexes (TOIS '17)

Every encoder represents each sequence **individually**.

No exploitation of redundancy.



Encode **clusters** of inverted lists.

Space

Always better than PEF (by up to **11%**) and better than BIC (by up to **6.25%**)

← Spectrum →

Time

Much faster than BIC (~103%)
Slightly slower than PEF (~20%)

Idea 2 - Optimally-partitioned VByte (TKDE ’18)

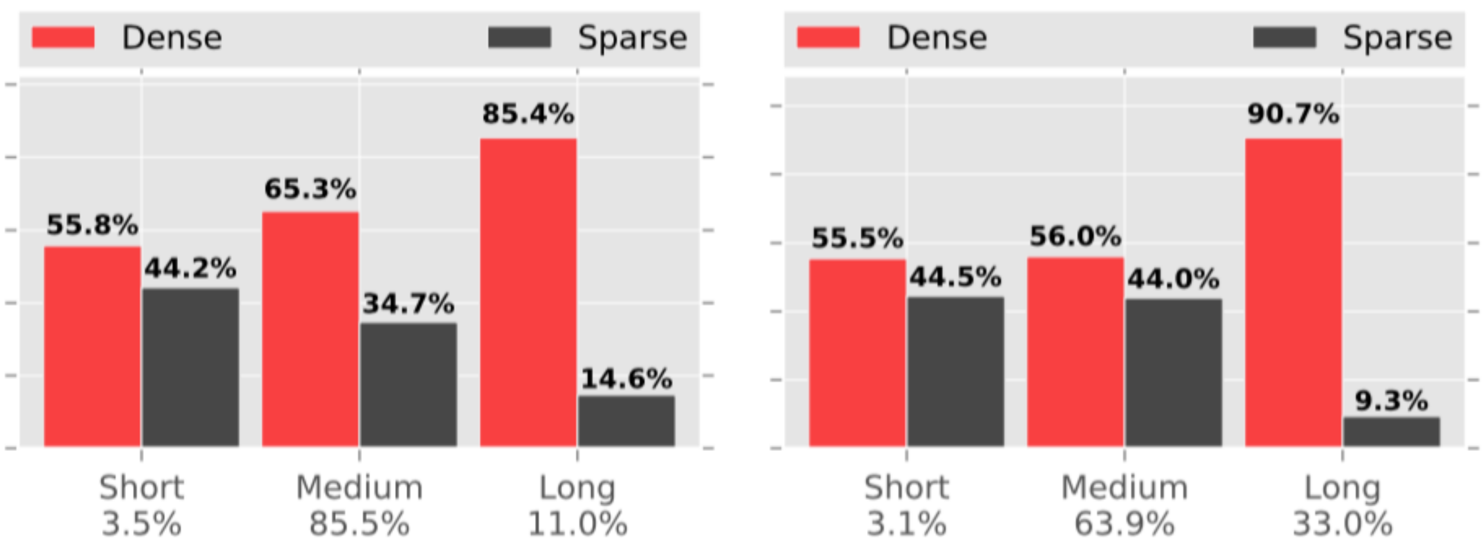
The majority of values are **small** (*very* small indeed).

VByte needs **at least 8 bits** per integer, that is sensibly far away from bit-level effectiveness (BIC: **3.54**, PEF: **4.1** on Gov2).

Idea 2 - Optimally-partitioned VByte (TKDE '18)

The majority of values are **small** (*very* small indeed).

VByte needs **at least 8 bits** per integer, that is sensibly far away from bit-level effectiveness (BIC: **3.54**, PEF: **4.1** on Gov2).

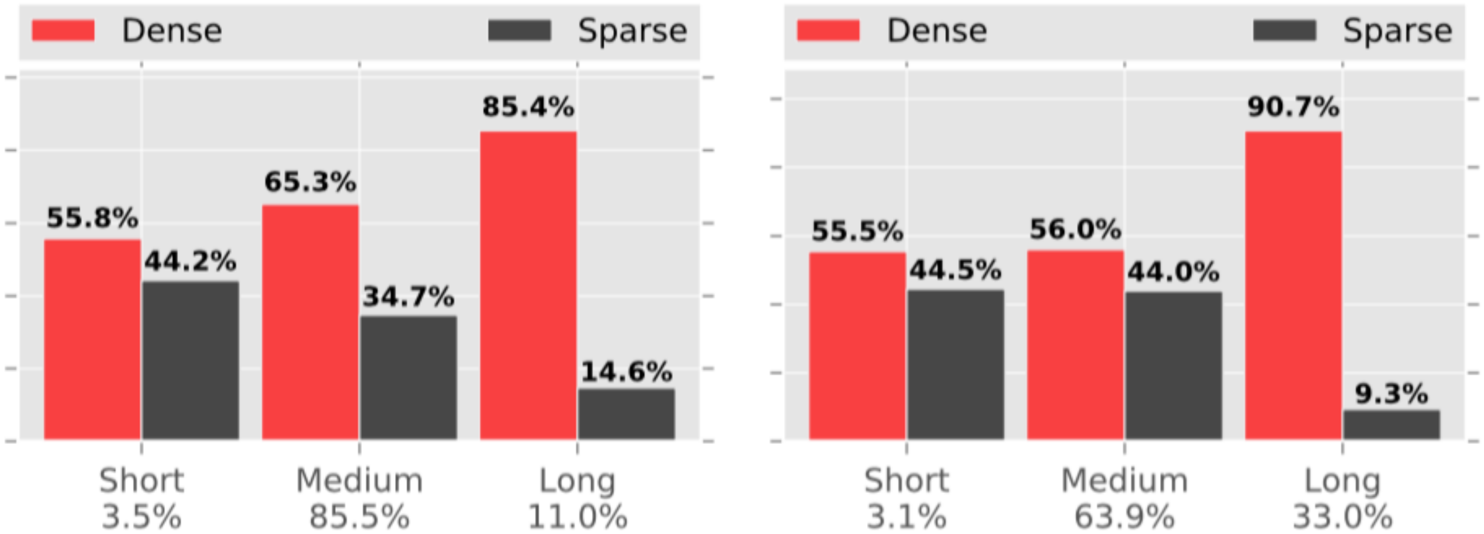


(a) Gov2 (b) ClueWeb09

Idea 2 - Optimally-partitioned VByte (TKDE '18)

The majority of values are **small** (*very* small indeed).

VByte needs **at least 8 bits** per integer, that is sensibly far away from bit-level effectiveness (BIC: **3.54**, PEF: **4.1** on Gov2).



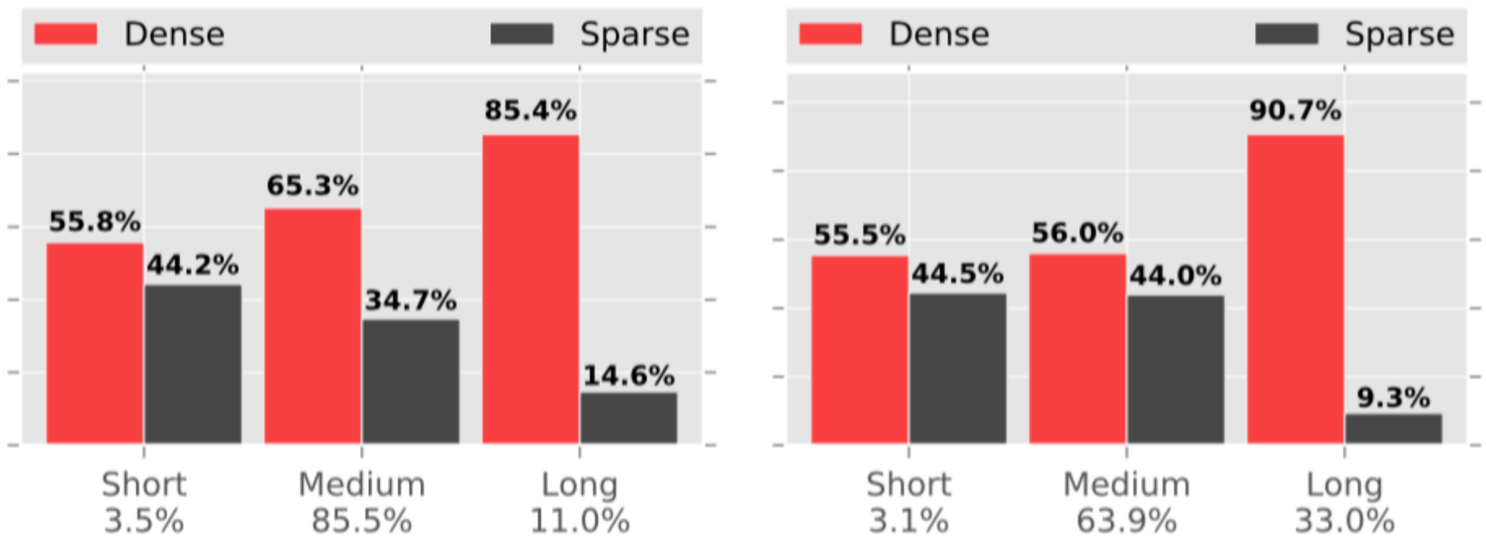
(a) Gov2 (b) ClueWeb09

Encode **dense** regions with unary codes, **sparse** regions with VByte.

Idea 2 - Optimally-partitioned VByte (TKDE '18)

The majority of values are **small** (*very* small indeed).

VByte needs **at least 8 bits** per integer, that is sensibly far away from bit-level effectiveness (BIC: **3.54**, PEF: **4.1** on Gov2).



(a) Gov2 (b) ClueWeb09

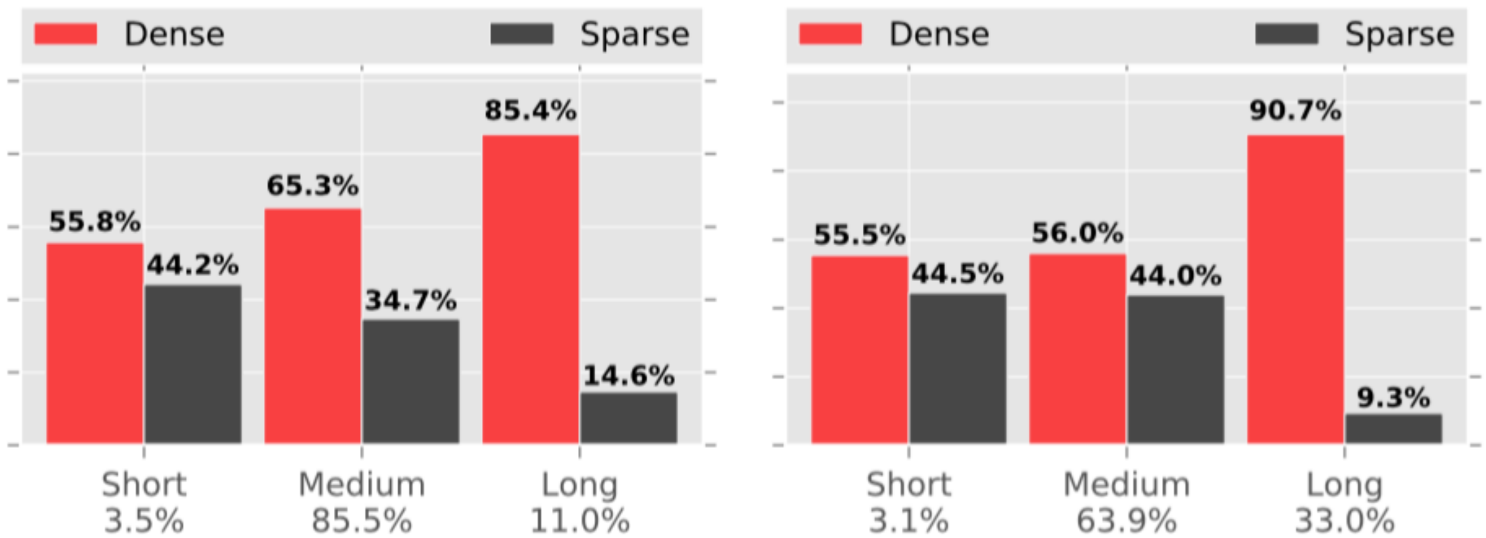
Encode **dense** regions with unary codes, **sparse** regions with VByte.

Optimal partitioning in linear time and constant space.

Idea 2 - Optimally-partitioned VByte (TKDE '18)

The majority of values are **small** (*very* small indeed).

VByte needs **at least 8 bits** per integer, that is sensibly far away from bit-level effectiveness (BIC: **3.54**, PEF: **4.1** on Gov2).



(a) Gov2 (b) ClueWeb09

Encode **dense** regions with unary codes, **sparse** regions with VByte.

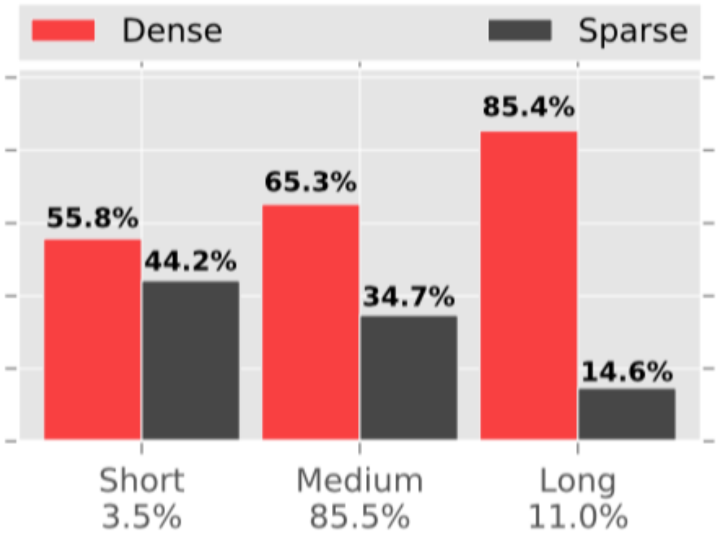
Optimal partitioning in linear time and constant space.

Compression ratio improves by **2X**.

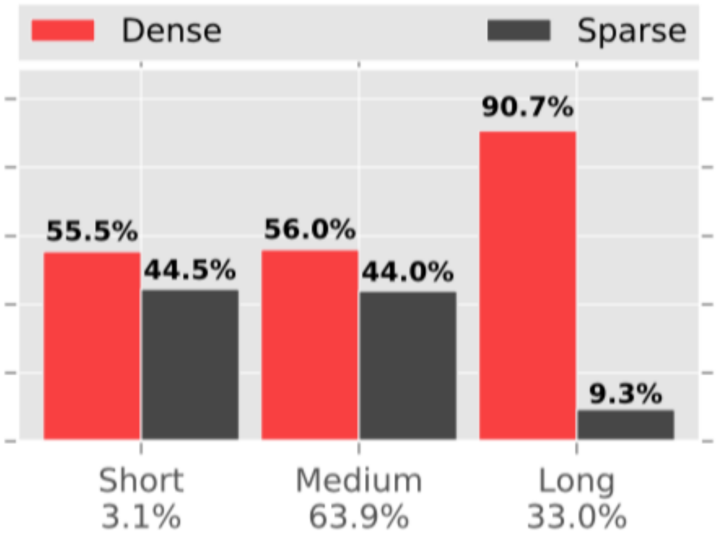
Idea 2 - Optimally-partitioned VByte (TKDE '18)

The majority of values are **small** (*very* small indeed).

VByte needs **at least 8 bits** per integer, that is sensibly far away from bit-level effectiveness (BIC: **3.54**, PEF: **4.1** on Gov2).



(a) Gov2



(b) ClueWeb09

Encode **dense** regions with unary codes, **sparse** regions with VByte.

Optimal partitioning in linear time and constant space.

Compression ratio improves by **2X**.

Query processing speed and sequential decoding **not affected**.

Idea 3 - Dictionary compression (WSDM '19)

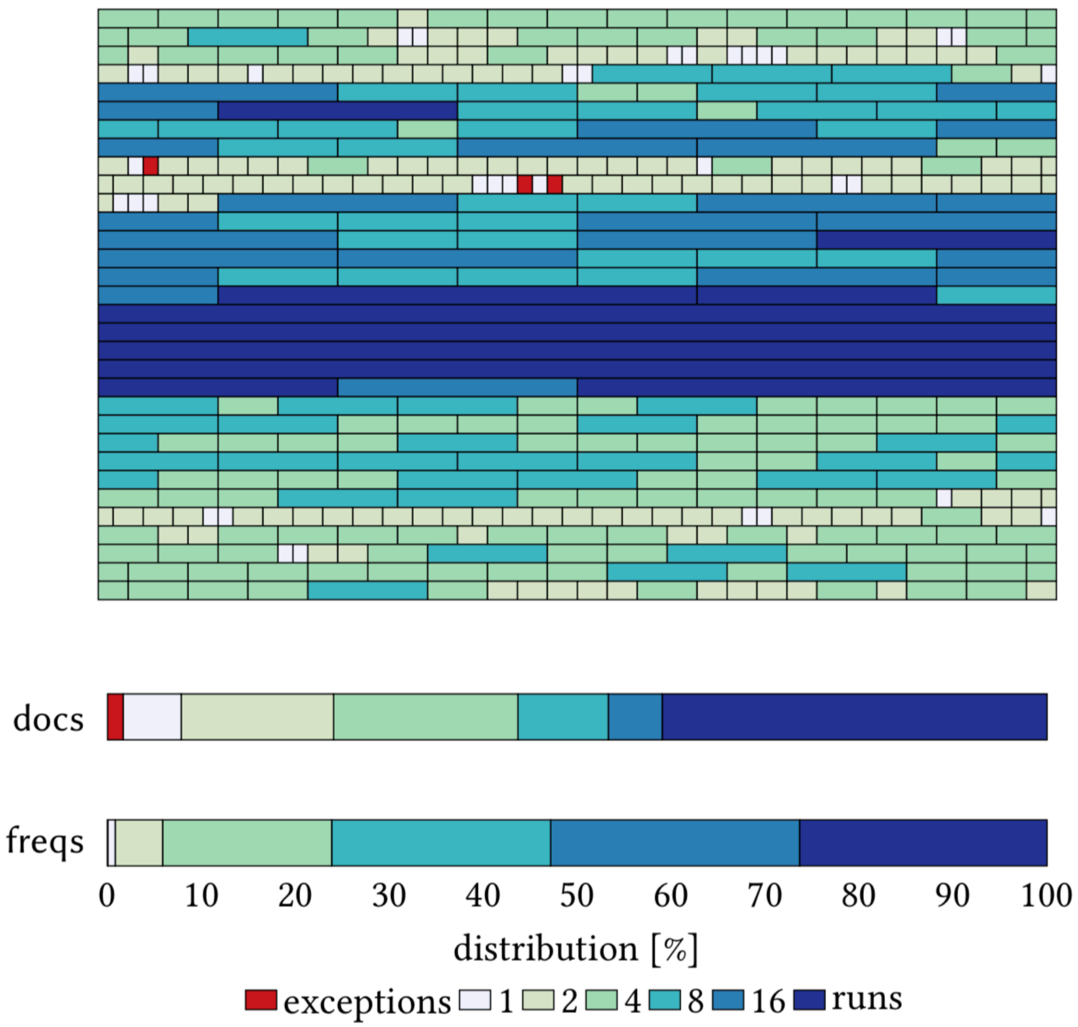
with M. Petri and A. Moffat
(University of Melbourne)

If we consider subsequences of *d*-gaps in inverted lists, these are **repetitive** across the whole inverted index.

Idea 3 - Dictionary compression (WSDM '19)

with M. Petri and A. Moffat
(University of Melbourne)

If we consider subsequences of *d*-gaps in inverted lists, these are **repetitive** across the whole inverted index.

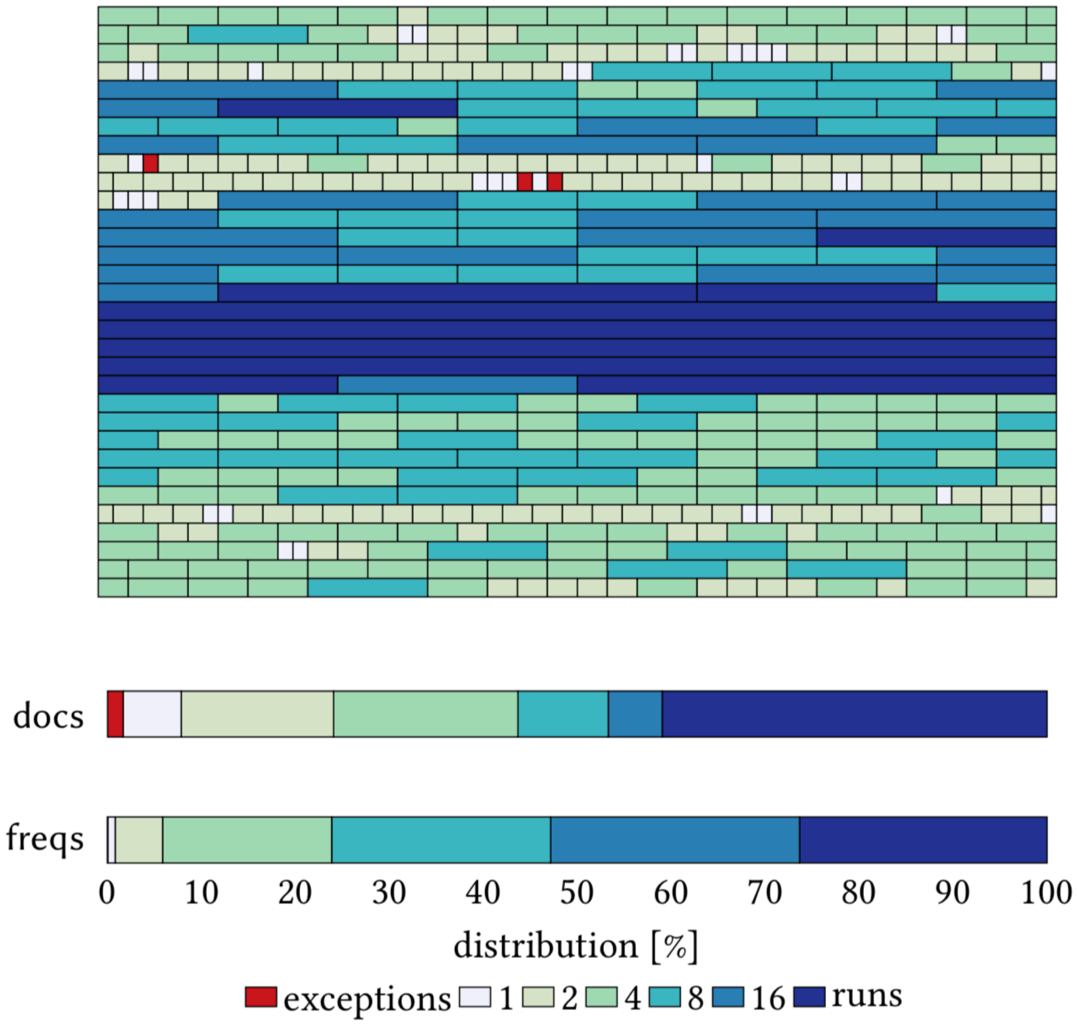


Put the **top-*k* frequent patters** in a dictionary of size *k*.
Then encode inverted lists as sequences of log *k*-bit codewords.

Idea 3 - Dictionary compression (WSDM '19)

with M. Petri and A. Moffat (University of Melbourne)

If we consider subsequences of *d*-gaps in inverted lists, these are **repetitive** across the whole inverted index.



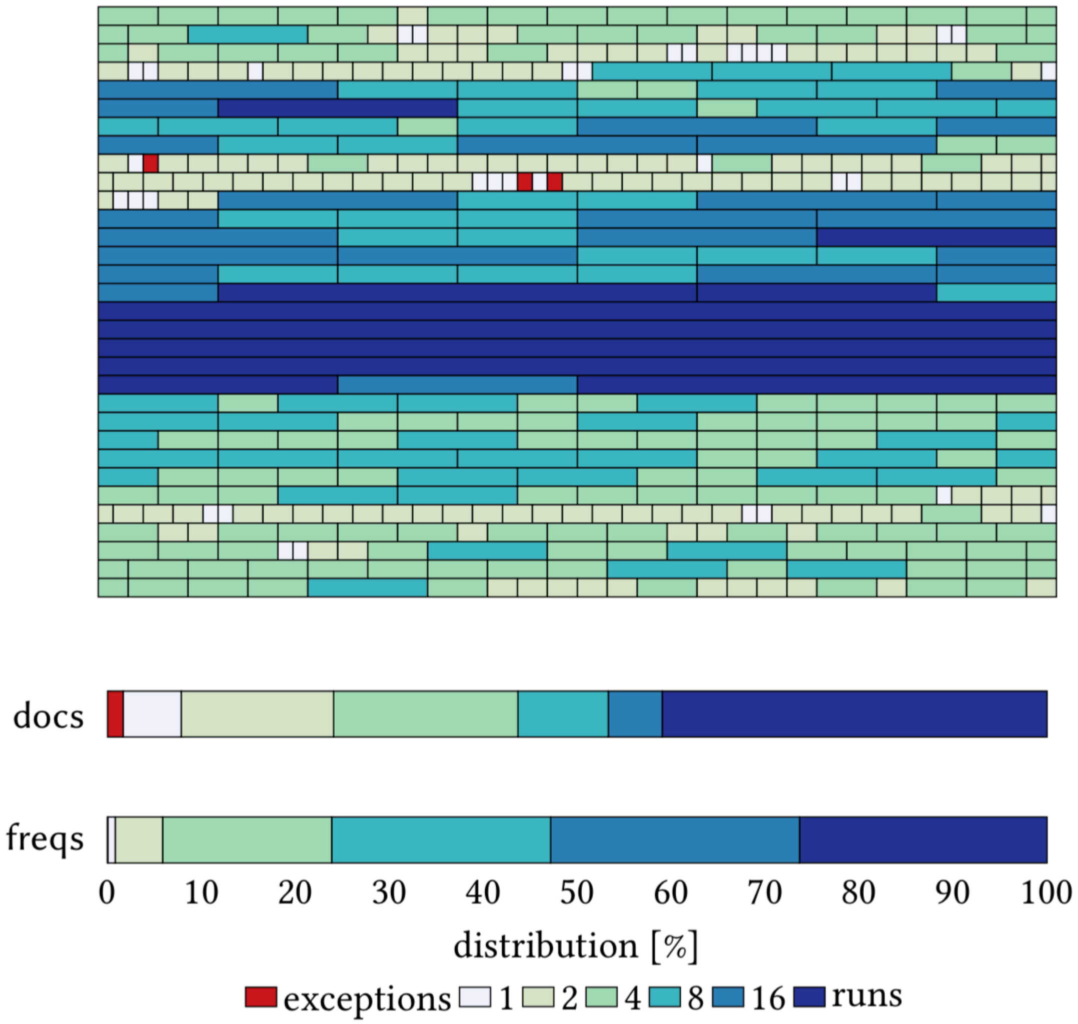
Put the **top-*k* frequent patters** in a dictionary of size *k*.
Then encode inverted lists as sequences of log *k*-bit codewords.

Close to the most space-efficient representation (~**7%** away from BIC).

Idea 3 - Dictionary compression (WSDM '19)

with M. Petri and A. Moffat (University of Melbourne)

If we consider subsequences of *d*-gaps in inverted lists, these are **repetitive** across the whole inverted index.

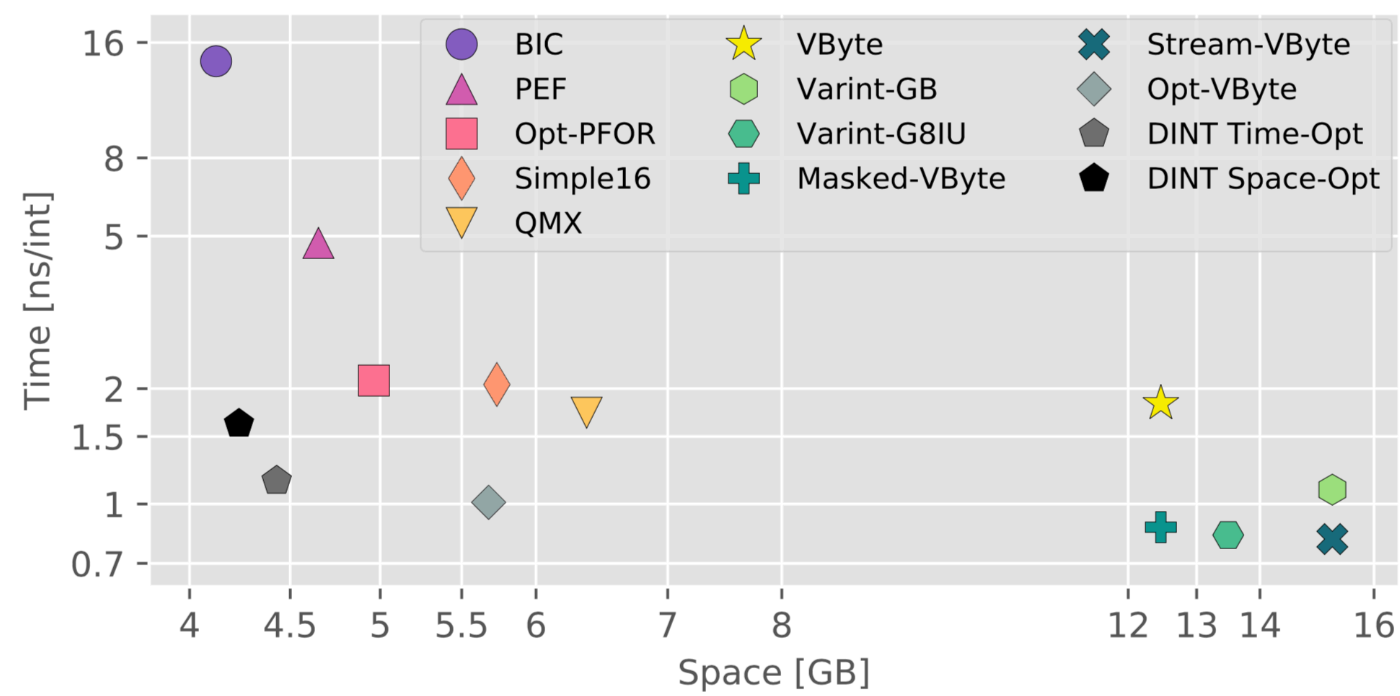


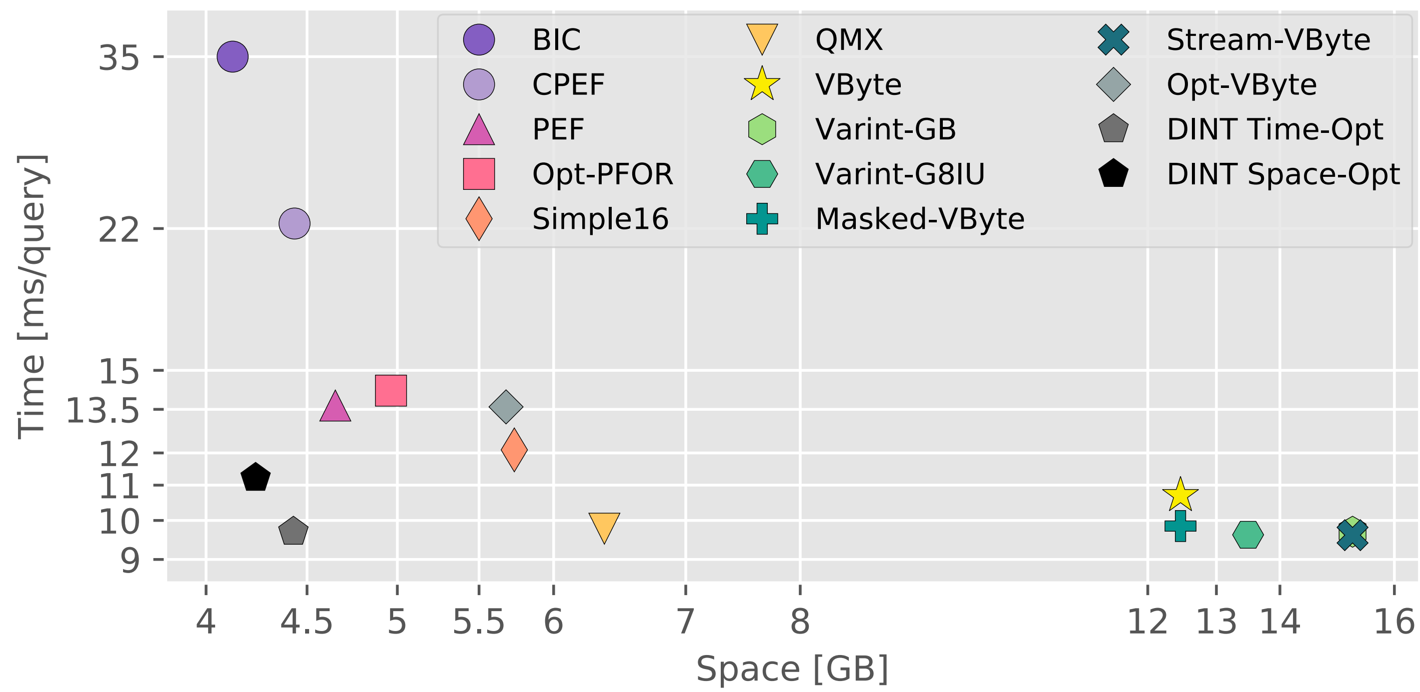
Put the **top-*k* frequent patters** in a dictionary of size *k*.
Then encode inverted lists as sequences of log *k*-bit codewords.

Close to the most space-efficient representation (~**7%** away from BIC).

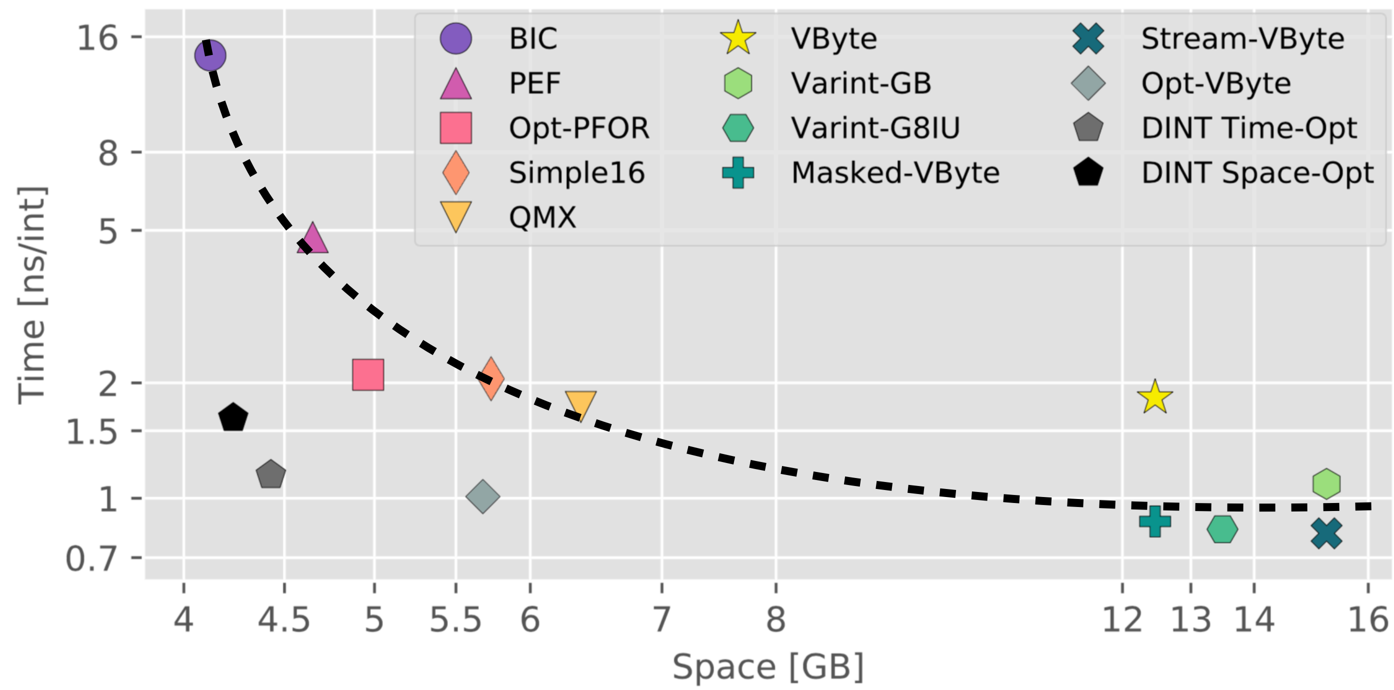
Almost **as fast as** the fastest SIMD-ized decoders.

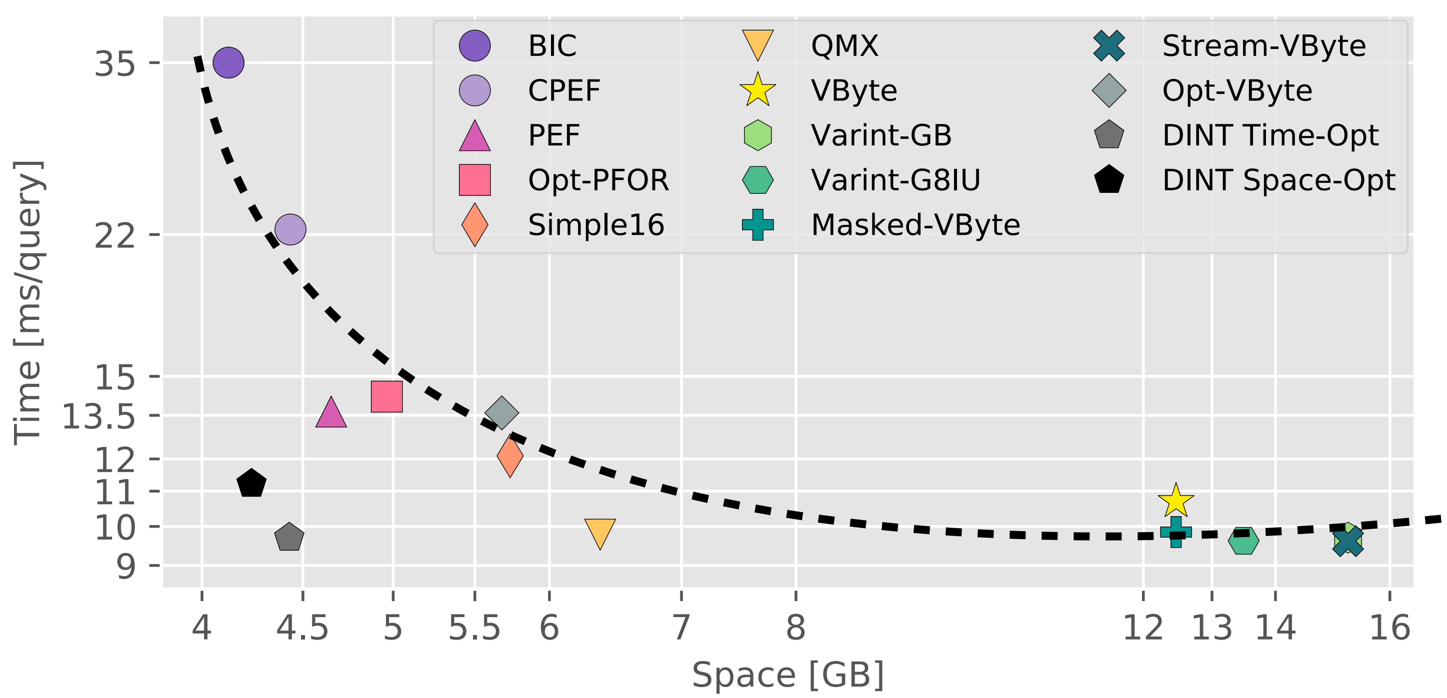
The bigger picture



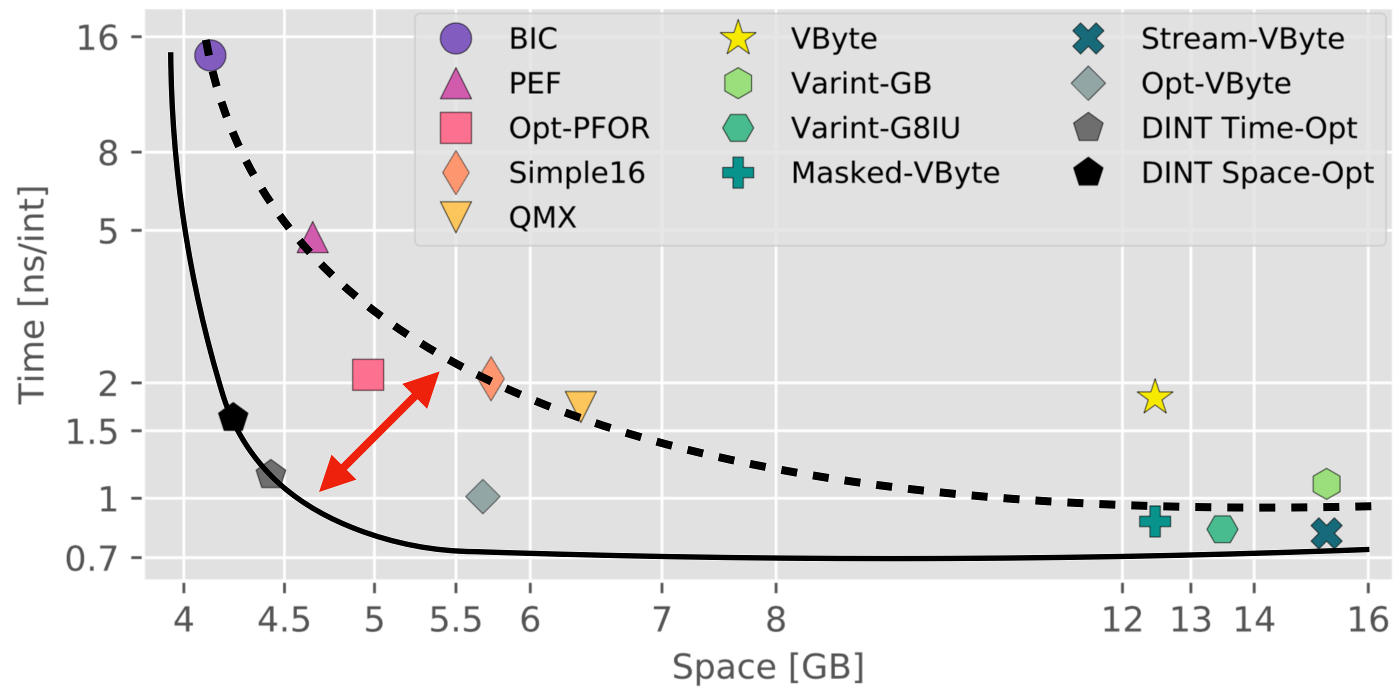


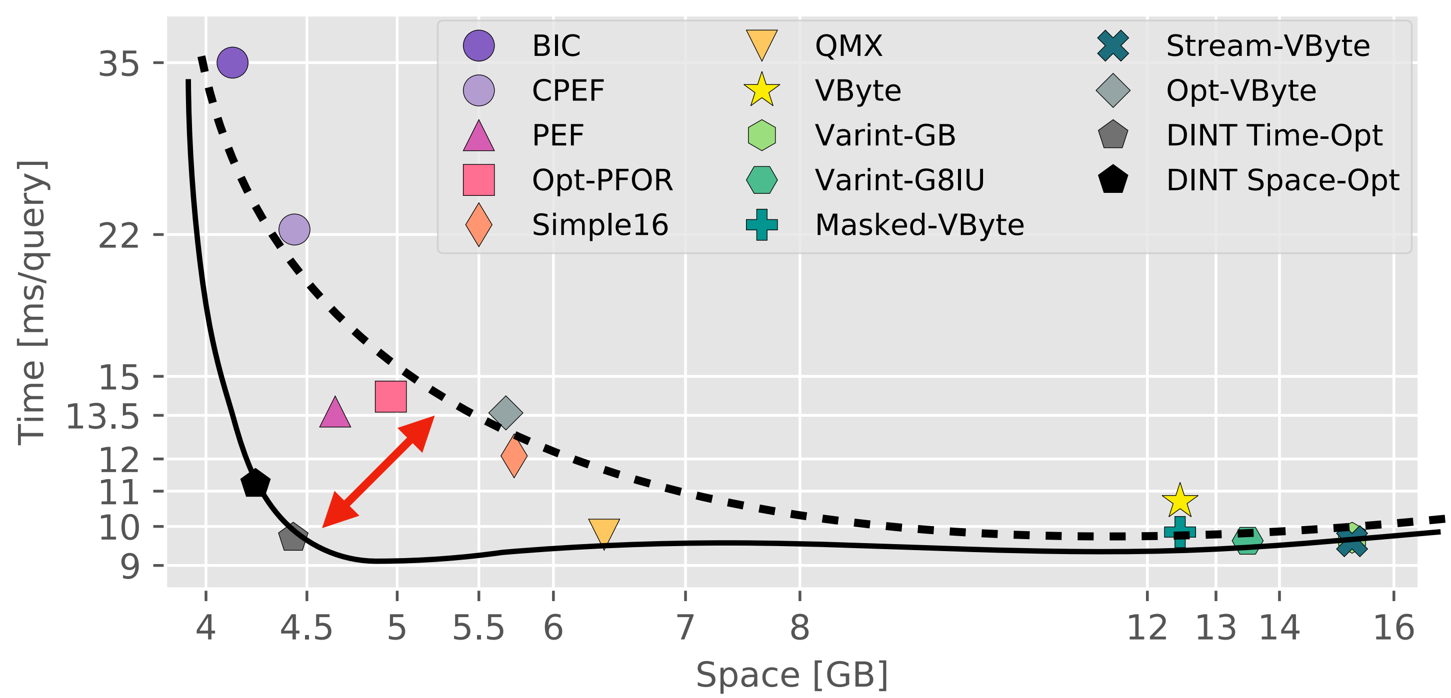
The bigger picture





The bigger picture





Problem 2

Integer data structures

- van Emde Boas Trees
- X/Y-Fast Tries
- Fusion Trees
- Exponential Search Trees
- …

+ time
- space
+ dynamic

Elias-Fano encoding

- EF($S(n,u)$) = $n \log(u/n) + 2n$ bits to encode a sorted integer sequence S
- O(1) **Access**
- O($1 + \log(u/n)$) **Predecessor**

+ time
+ space
- static

Problem 2

Integer data structures

- van Emde Boas Trees
- X/Y-Fast Tries
- Fusion Trees
- Exponential Search Trees
- …

Elias-Fano encoding

- EF($S(n,u)$) = $n \log(u/n) + 2n$ bits to encode a sorted integer sequence S
- $O(1)$ **Access**
- $O(1 + \log(u/n))$ **Predecessor**

+ time
- space
+ dynamic

+ time
+ space
- static

Can we grab the best from both?

Dynamic inverted indexes

Classic solution: use two indexes.
One is big and **cold**; the other is small and **hot**.
Merge them periodically.

Append-only inverted indexes.

Integer dictionaries in succinct space (CPM ’17)

Result 1

For $u = n^\gamma$, $\gamma = \Theta(1)$:

- EF($S(n,u)$) + o(n) bits
- O(1) **Access**
- O(min{1+log(u/n), loglog n}) **Predecessor**

Result 2

- EF($S(n,u)$) + o(n) bits
- O(1) **Access**
- O(1) **Append** (amortized)
- O(min{1+log(u/n), loglog n}) **Predecessor**

Result 3

- EF($S(n,u)$) + o(n) bits
- O(log n / loglog n) **Access**
- O(log n / loglog n) **Insert/Delete** (amortized)
- O(min{1+log(u/n), loglog n}) **Predecessor**

Integer dictionaries in succinct space (CPM '17)

Result 1

For $u = n^\gamma$, $\gamma = \Theta(1)$:

- EF($S(n,u)$) + o(n) bits
- O(1) **Access**
- O(min{1+log(u/n), loglog n}) **Predecessor**

Result 2

- EF($S(n,u)$) + o(n) bits
- O(1) **Access**
- O(1) **Append** (amortized)
- O(min{1+log(u/n), loglog n}) **Predecessor**

Result 3

- EF($S(n,u)$) + o(n) bits
- O(log n / loglog n) **Access**
- O(log n / loglog n) **Insert/Delete** (amortized)
- O(min{1+log(u/n), loglog n}) **Predecessor**

Optimal time bounds for all operations using a sublunar redundancy.

Problem 3

Consider a large text.

Problem 3

Consider a large text.

How to represent all its substrings of size $1 \leq k \leq N$ words for fixed N (e.g., $N = 5$), using **as few as possible** bits?

Fast **Access** to individual N-grams?

How to **estimate** the probability of occurrence of the patterns under a given probability model?

Problem 3

Consider a large text.

How to represent all its substrings of size $1 \leq k \leq N$ words for fixed N (e.g., N = 5), using **as few as possible** bits?

Fast **Access** to individual N-grams?

How to **estimate** the probability of occurrence of the patterns under a given probability model?

This is problem is central to applications in IR, ML, NLP, WSE.

Applications

Next word prediction.

Applications

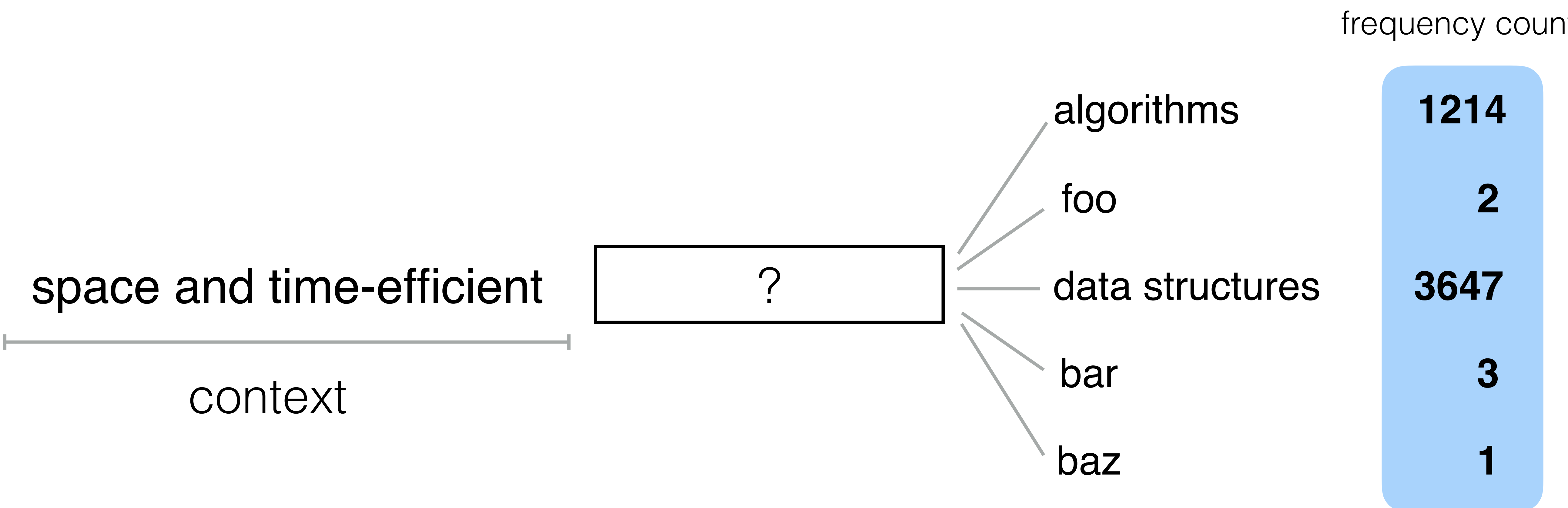
Next word prediction.

space and time-efficient ?

context

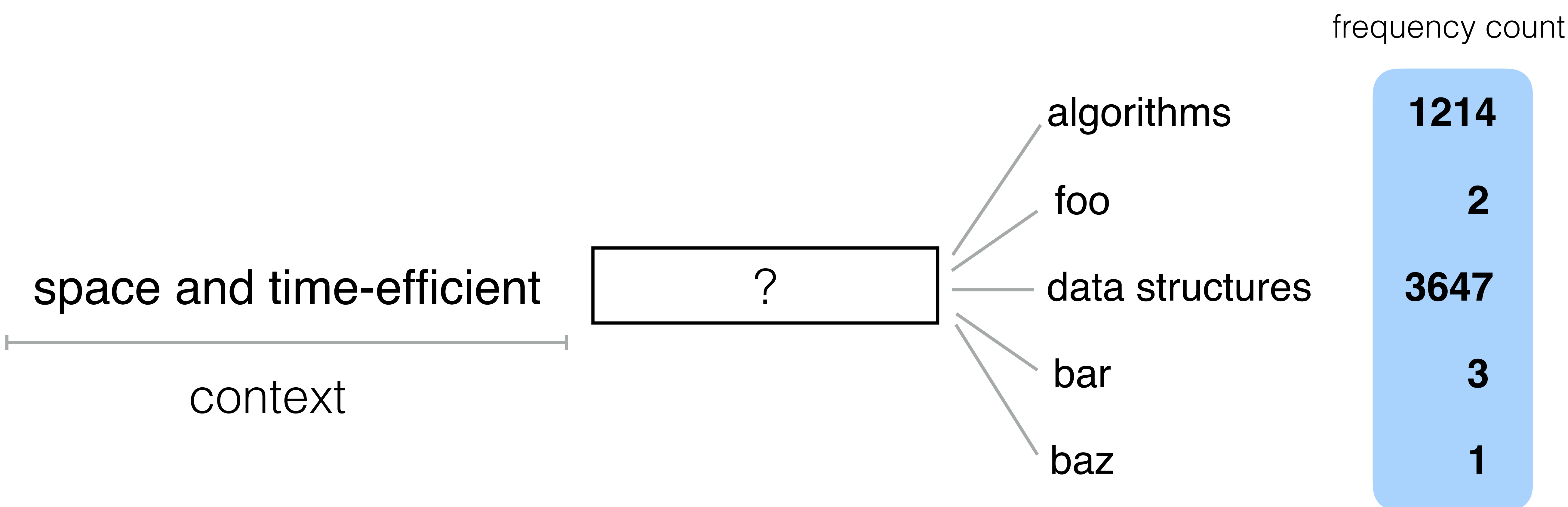
Applications

Next word prediction.



Applications

Next word prediction.



$$\mathcal{P}(\text{“data structures”} \mid \text{“space and time-efficient”}) \approx \frac{f(\text{“space and time-efficient data structures”})}{f(\text{“space and time-efficient”})}$$

What can I help you with?

Siri

Applications

Google Research Blog

The latest news from Research at Google

All Our N-gram are Belong to You

Thursday, August 03, 2006

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. While such models have usually been estimated from training corpora containing at most a few billion words, we have been harnessing the vast power of Google's datacenters and distributed processing infrastructure to process larger and larger training corpora. We found that there's no data like more data, and scaled up the size of our data by one order of magnitude, and then another, and then one more - resulting in a training corpus of *one trillion words* from public Web pages.

Applications

Google Research Blog

The latest news from Research at Google

Google Translate

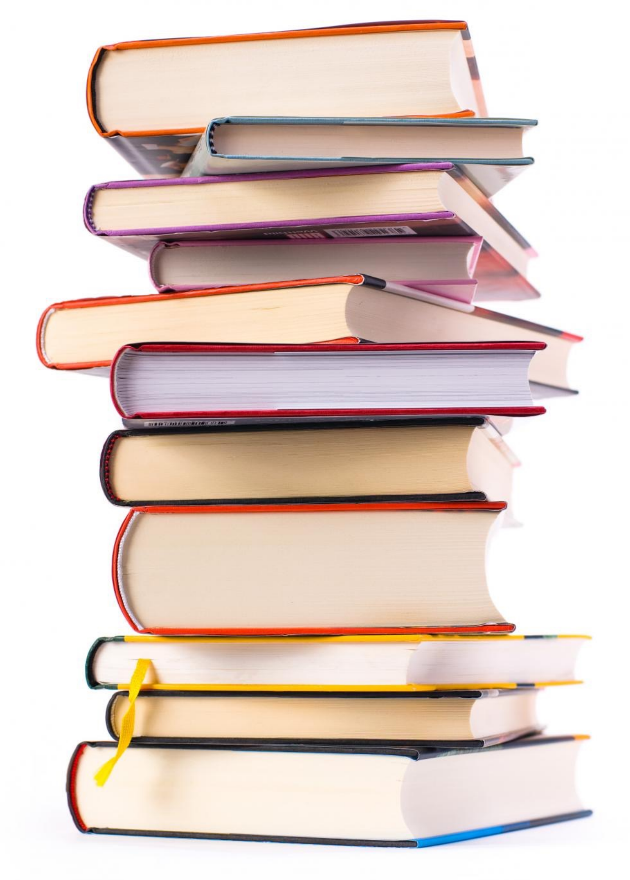
All Our N-gram are Belong to You

Thursday, August 03, 2006

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. While such models have usually been estimated from training corpora containing at most a few billion words, we have been harnessing the vast power of Google's datacenters and distributed processing infrastructure to process larger and larger training corpora. We found that there's no data like more data, and scaled up the size of our data by one order of magnitude, and then another, and then one more - resulting in a training corpus of *one trillion words* from public Web pages.

Indexing



Google Books

~6% of the books ever published

n	number of n-grams
1	24,359,473
2	667,284,771
3	7,397,041,901
4	1,644,807,896
5	1,415,355,596

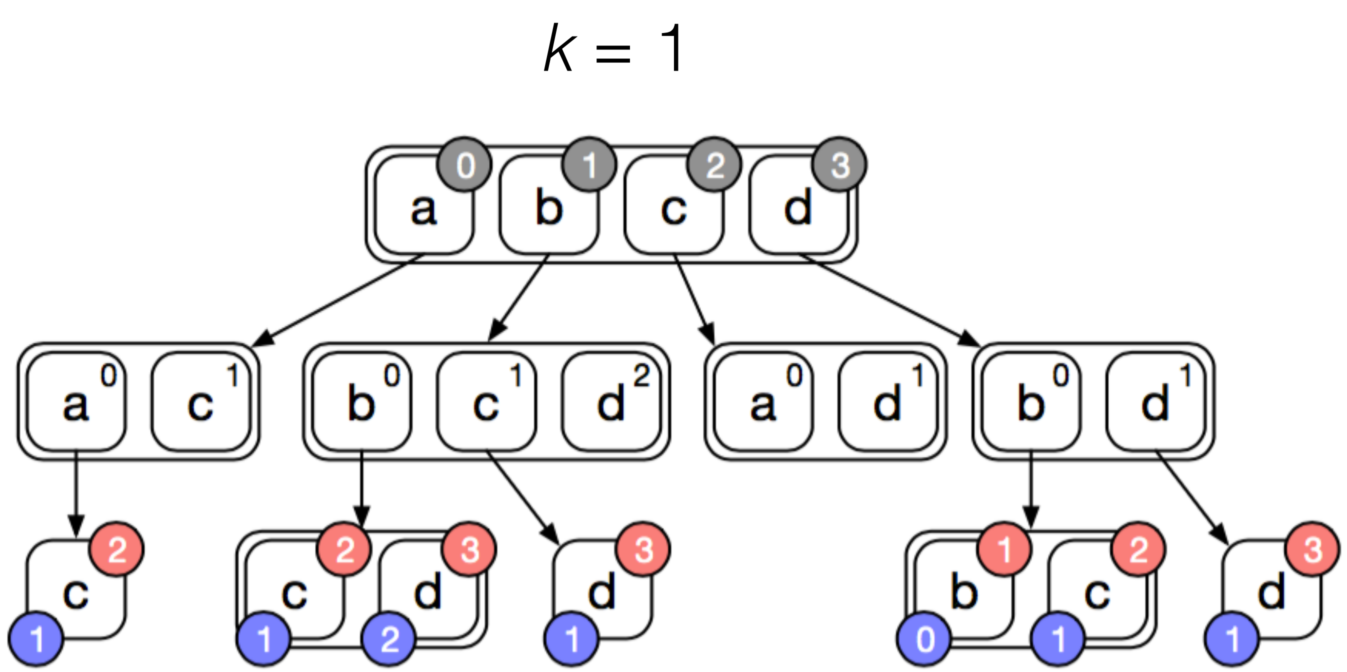
More than 11 billion n-grams.

Idea 1 - Context-based remapped tries (SIGIR '17)

The number of words following a given context is **small**.

Idea 1 - Context-based remapped tries (SIGIR ’17)

The number of words following a given context is **small**.

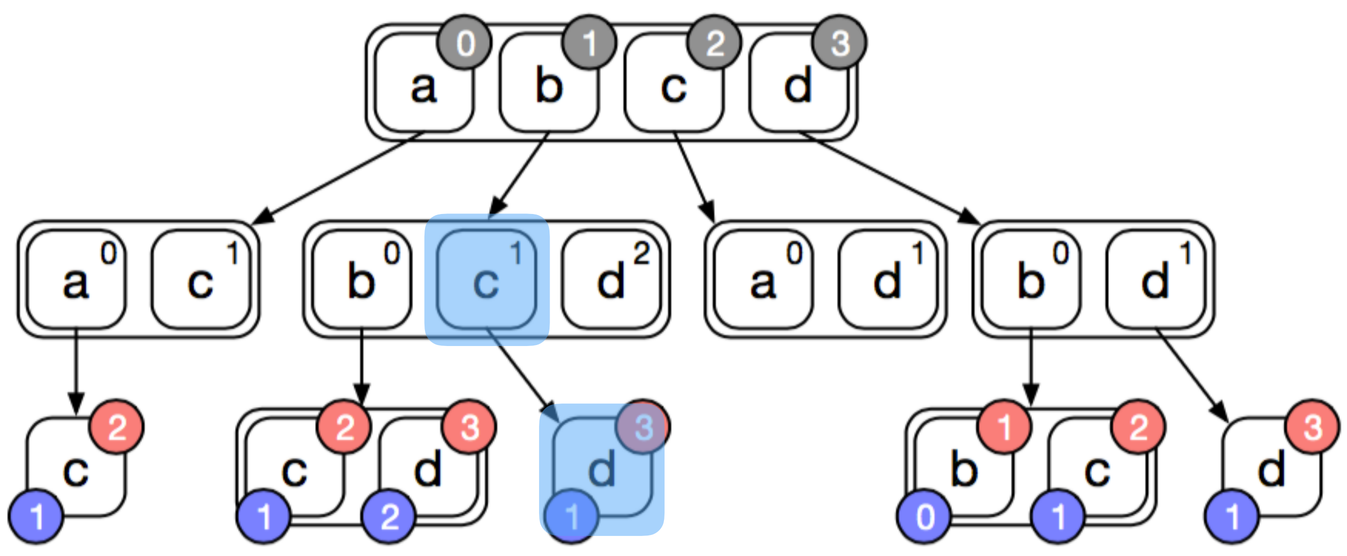


Map a word ID to the **position** it takes within its *sibling* IDs (the IDs following a context of fixed length *k*).

Idea 1 - Context-based remapped tries (SIGIR '17)

The number of words following a given context is **small**.

k = 1

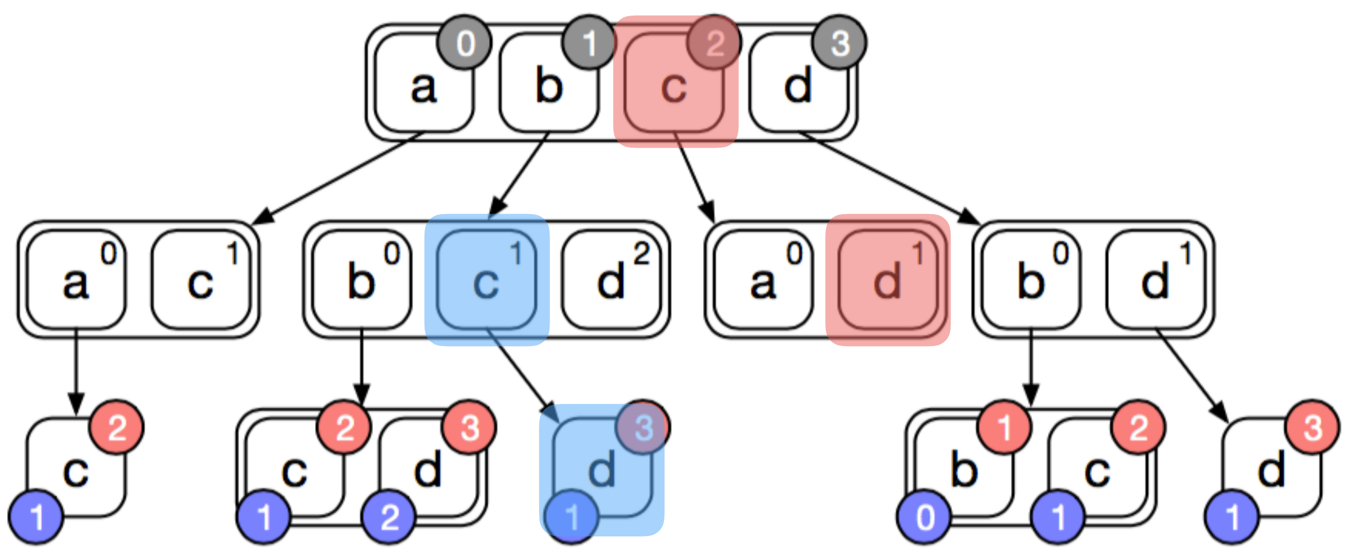


Map a word ID to the **position** it takes within its *sibling* IDs (the IDs following a context of fixed length k).

Idea 1 - Context-based remapped tries (SIGIR ’17)

The number of words following a given context is **small**.

$k = 1$

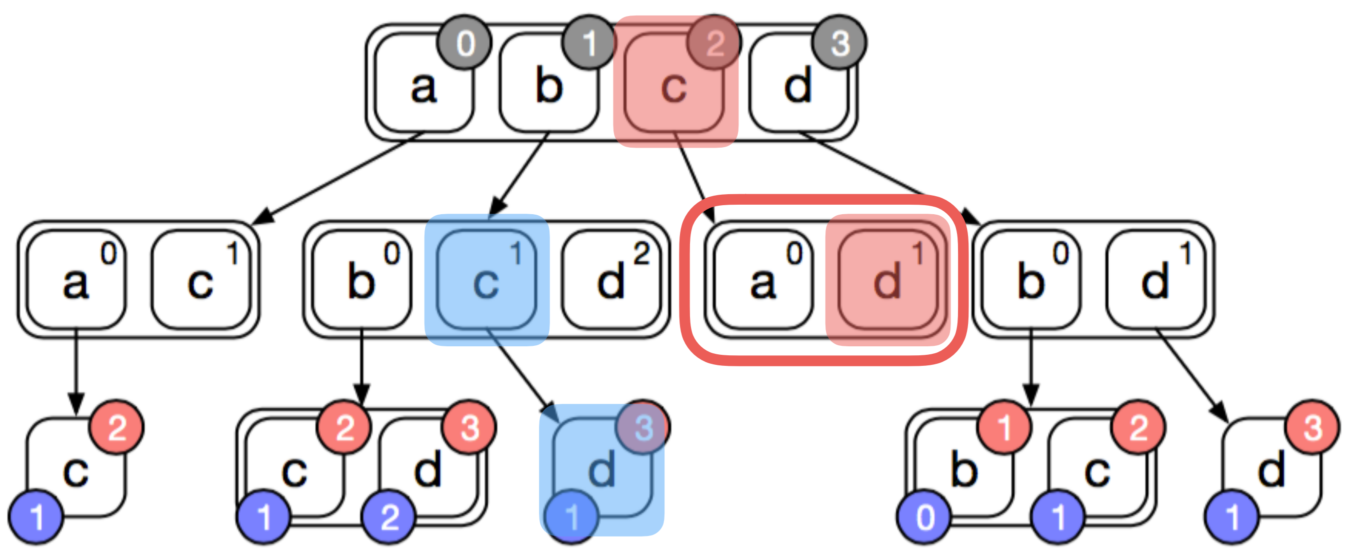


Map a word ID to the **position** it takes within its *sibling* IDs (the IDs following a context of fixed length *k*).

Idea 1 - Context-based remapped tries (SIGIR '17)

The number of words following a given context is **small**.

$k = 1$

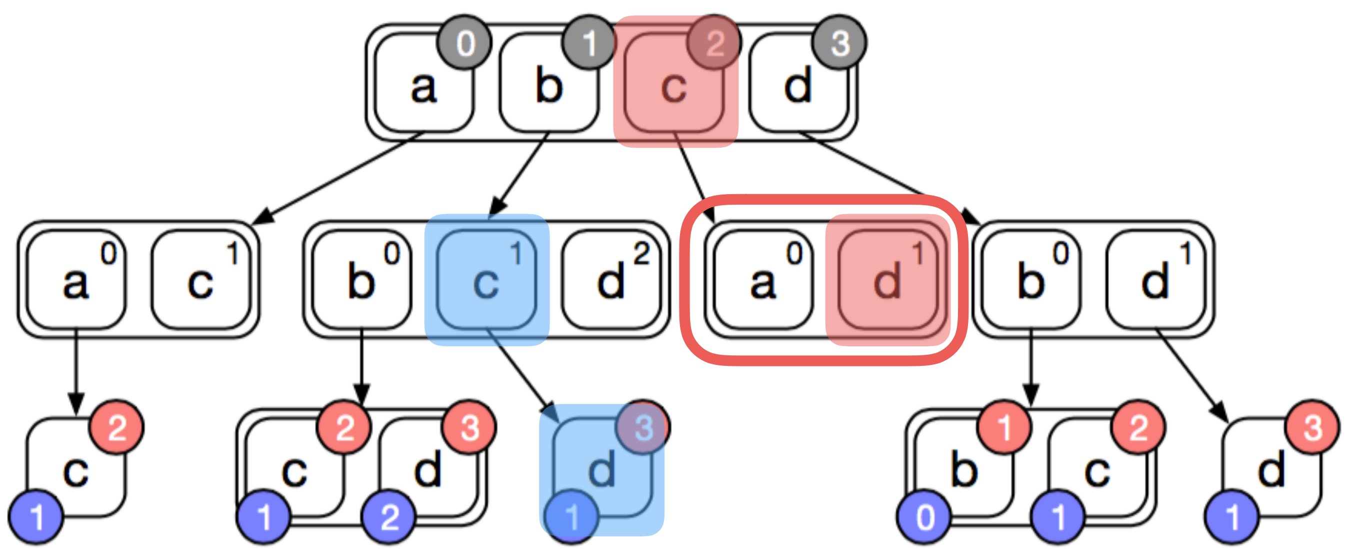


Map a word ID to the **position** it takes within its *sibling* IDs (the IDs following a context of fixed length *k*).

Idea 1 - Context-based remapped tries (SIGIR ’17)

The number of words following a given context is **small**.

$k = 1$

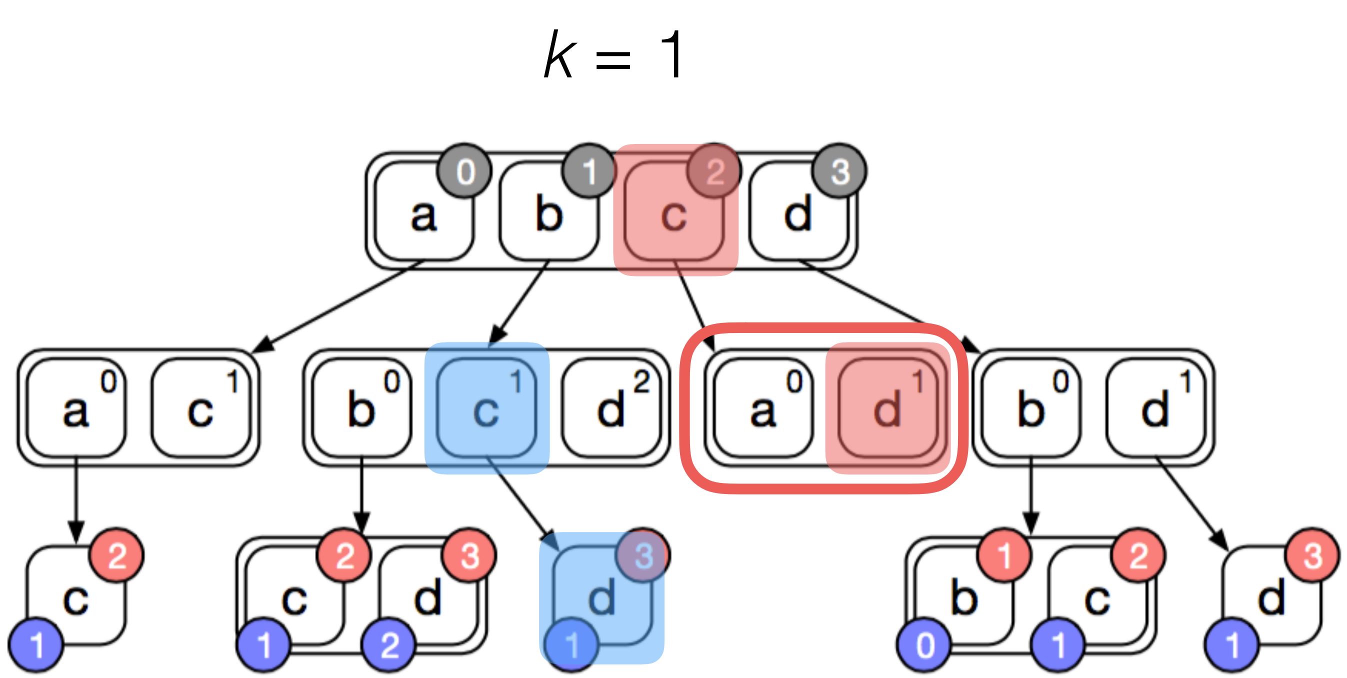


Map a word ID to the **position** it takes within its *sibling* IDs (the IDs following a context of fixed length *k*).

The (Elias-Fano) context-based remapped trie is **as fast as** the fastest competitor, but up to **65% smaller**.

Idea 1 - Context-based remapped tries (SIGIR ’17)

The number of words following a given context is **small**.



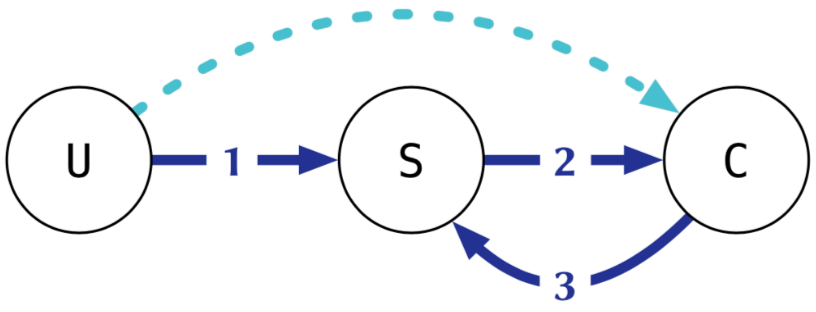
Map a word ID to the **position** it takes within its *sibling* IDs (the IDs following a context of fixed length *k*).

The (Elias-Fano) context-based remapped trie is **as fast as** the fastest competitor, but up to **65% smaller**.

The (Elias-Fano) context-based remapped trie is **even smaller** than the most space-efficient competitors, that are lossy and with false-positives allowed, and up to **5X faster**.

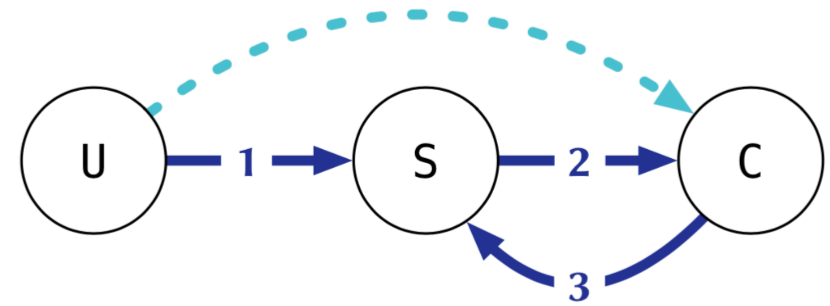
Idea 2 - Fast estimation in external memory (TOIS ’18)

To compute the modified Kneser-Ney probabilities of the n-grams,
the fastest algorithm in the literature uses **3 sorting steps** in external memory.



Idea 2 - Fast estimation in external memory (TOIS '18)

To compute the modified Kneser-Ney probabilities of the n-grams,
the fastest algorithm in the literature uses **3 sorting steps** in external memory.



Computing the distinct left extensions.

1	2	3	4	5	6	7	8	9	10	11	12
A	**A**	**A**	**A**	**B**	**B**	**C**	**C**	**X**	**X**	**X**	**X**
A	B	B	X	A	C	A	A	C	X	X	X
B	A	C	X	X	A	A	B	A	C	X	X
A	X	A	X	X	A	B	C	B	A	C	X
X	X	A	X	X	B	A	A	C	B	A	C

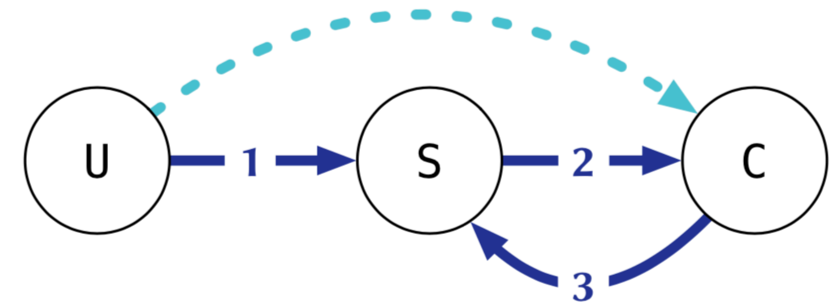
Suffix order

7	1	8	5	2	3	6	9	10	11	12	4
C	A	C	B	A	A	B	X	X	X	X	A
A	**A**	**A**	**A**	**B**	**B**	**C**	**C**	**X**	**X**	**X**	**X**
A	B	B	X	A	C	A	A	C	X	X	X
B	A	C	X	X	A	A	B	A	C	X	X
A	X	A	X	X	A	B	C	B	A	C	X

Context order

Idea 2 - Fast estimation in external memory (TOIS ’18)

To compute the modified Kneser-Ney probabilities of the n-grams,
the fastest algorithm in the literature uses **3 sorting steps** in external memory.



Computing the distinct left extensions.

1	2	3	4	5	6	7	8	9	10	11	12
A	**A**	**A**	**A**	**B**	**B**	**C**	**C**	**X**	**X**	**X**	**X**
A	B	B	X	A	C	A	A	C	X	X	X
B	A	C	X	X	A	A	B	A	C	X	X
A	X	A	X	X	A	B	C	B	A	C	X
X	X	A	X	X	B	A	A	C	B	A	C

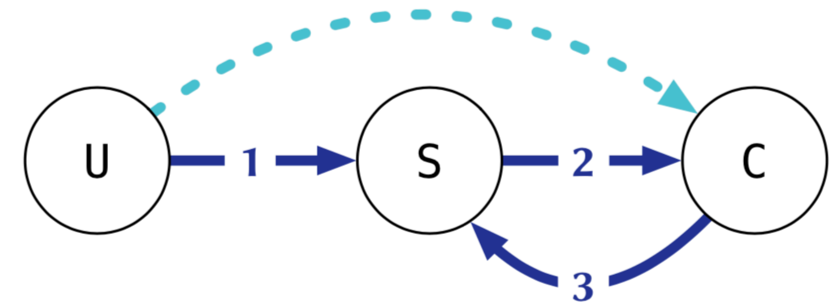
Suffix order

7	1	8	5	2	3	6	9	10	11	12	4
C	A	C	B	A	A	B	X	X	X	X	A
A	**A**	**A**	**A**	**B**	**B**	**C**	**C**	**X**	**X**	**X**	**X**
A	B	B	X	A	C	A	A	C	X	X	X
B	A	C	X	X	A	A	B	A	C	X	X
A	X	A	X	X	A	B	C	B	A	C	X

Context order

Idea 2 - Fast estimation in external memory (TOIS '18)

To compute the modified Kneser-Ney probabilities of the n-grams,
the fastest algorithm in the literature uses **3 sorting steps** in external memory.



Computing the distinct left extensions.

1	2	3	4	5	6	7	8	9	10	11	12
A	**A**	**A**	**A**	**B**	**B**	**C**	**C**	**X**	**X**	**X**	**X**
A	B	B	X	A	C	A	A	C	X	X	X
B	A	C	X	X	A	A	B	A	C	X	X
A	X	A	X	X	A	B	C	B	A	C	X
X	X	A	X	X	B	A	A	C	B	A	C

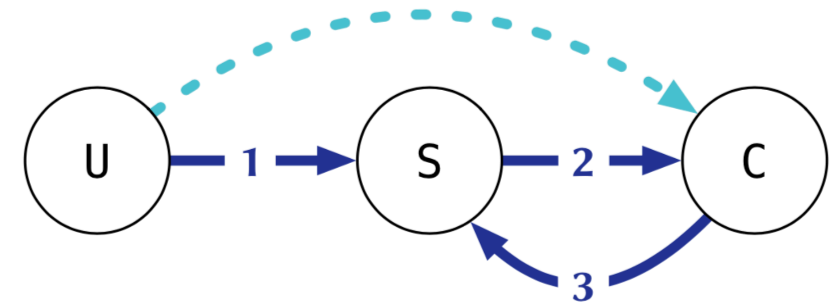
Suffix order

7	1	8	5	2	3	6	9	10	11	12	4
C	A	C	B	A	A	B	X	X	X	X	A
A	**A**	**A**	**A**	**B**	**B**	**C**	**C**	**X**	**X**	**X**	**X**
A	B	B	X	A	C	A	A	C	X	X	X
B	A	C	X	X	A	A	B	A	C	X	X
A	X	A	X	X	A	B	C	B	A	C	X

Context order

Idea 2 - Fast estimation in external memory (TOIS ’18)

To compute the modified Kneser-Ney probabilities of the n-grams,
the fastest algorithm in the literature uses **3 sorting steps** in external memory.



Computing the distinct left extensions.

1	2	3	4	5	6	7	8	9	10	11	12
A	**A**	**A**	**A**	**B**	**B**	**C**	**C**	**X**	**X**	**X**	**X**
A	B	B	X	A	C	A	A	C	X	X	X
B	A	C	X	X	A	A	B	A	C	X	X
A	X	A	X	X	A	B	C	B	A	C	X
X	X	A	X	X	B	A	A	C	B	A	C

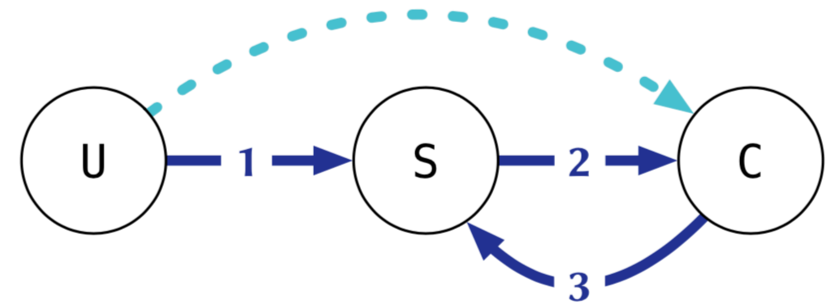
Suffix order

7	1	8	5	2	3	6	9	10	11	12	4
C	A	C	B	A	A	B	X	X	X	X	A
A	**A**	**A**	**A**	**B**	**B**	**C**	**C**	**X**	**X**	**X**	**X**
A	B	B	X	A	C	A	A	C	X	X	X
B	A	C	X	X	A	A	B	A	C	X	X
A	X	A	X	X	A	B	C	B	A	C	X

Context order

Idea 2 - Fast estimation in external memory (TOIS ’18)

To compute the modified Kneser-Ney probabilities of the n-grams,
the fastest algorithm in the literature uses
3 sorting steps in external memory.



Computing the distinct left extensions.

1	2	3	4	5	6	7	8	9	10	11	12
A	**A**	**A**	**A**	**B**	**B**	**C**	**C**	**X**	**X**	**X**	**X**
A	B	B	X	A	C	A	A	C	X	X	X
B	A	C	X	X	A	A	B	A	C	X	X
A	X	A	X	X	A	B	C	B	A	C	X
X	X	A	X	X	B	A	A	C	B	A	C

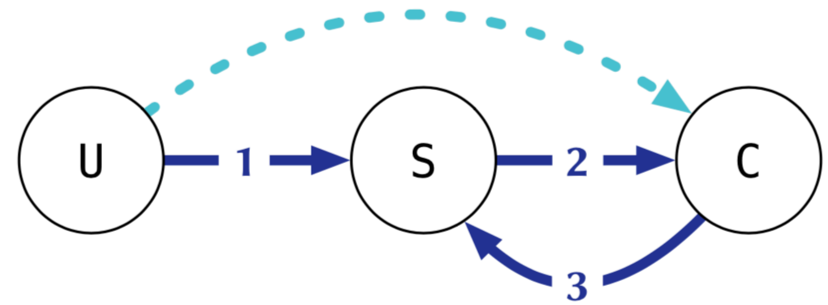
Suffix order

7	1	8	5	2	3	6	9	10	11	12	4
C	A	C	B	A	A	B	X	X	X	X	A
A	**A**	**A**	**A**	**B**	**B**	**C**	**C**	**X**	**X**	**X**	**X**
A	B	B	X	A	C	A	A	C	X	X	X
B	A	C	X	X	A	A	B	A	C	X	X
A	X	A	X	X	A	B	C	B	A	C	X

Context order

Idea 2 - Fast estimation in external memory (TOIS ’18)

To compute the modified Kneser-Ney probabilities of the n-grams,
the fastest algorithm in the literature uses **3 sorting steps** in external memory.



Computing the distinct left extensions.

1	2	3	4	5	6	7	8	9	10	11	12
A	**A**	**A**	**A**	**B**	**B**	**C**	**C**	**X**	**X**	**X**	**X**
A	B	B	X	A	C	A	A	C	X	X	X
B	A	C	X	X	A	A	B	A	C	X	X
A	X	A	X	X	A	B	C	B	A	C	X
X	X	A	X	X	B	A	A	C	B	A	C

Suffix order

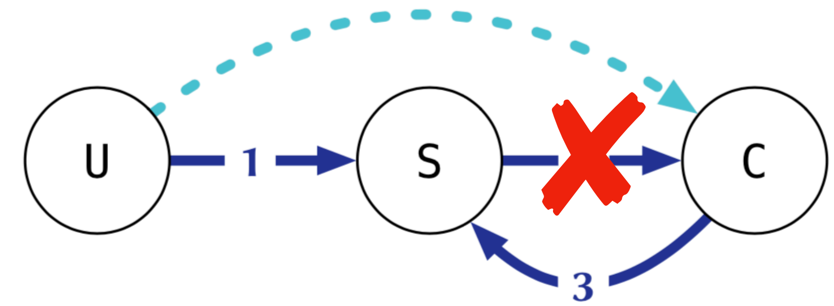
7	1	8	5	2	3	6	9	10	11	12	4
C	A	C	B	A	A	B	X	X	X	X	A
A	**A**	**A**	**A**	**B**	**B**	**C**	**C**	**X**	**X**	**X**	**X**
A	B	B	X	A	C	A	A	C	X	X	X
B	A	C	X	X	A	A	B	A	C	X	X
A	X	A	X	X	A	B	C	B	A	C	X

Context order

Using a scan of the block and O(|V|) space.

Idea 2 - Fast estimation in external memory (TOIS '18)

To compute the modified Kneser-Ney probabilities of the n-grams,
the fastest algorithm in the literature uses **3 sorting steps** in external memory.



Computing the distinct left extensions.

1	2	3	4	5	6	7	8	9	10	11	12
A	**A**	**A**	**A**	**B**	**B**	**C**	**C**	**X**	**X**	**X**	**X**
A	B	B	X	A	C	A	A	C	X	X	X
B	A	C	X	X	A	A	B	A	C	X	X
A	X	A	X	X	A	B	C	B	A	C	X
X	X	A	X	X	B	A	A	C	B	A	C

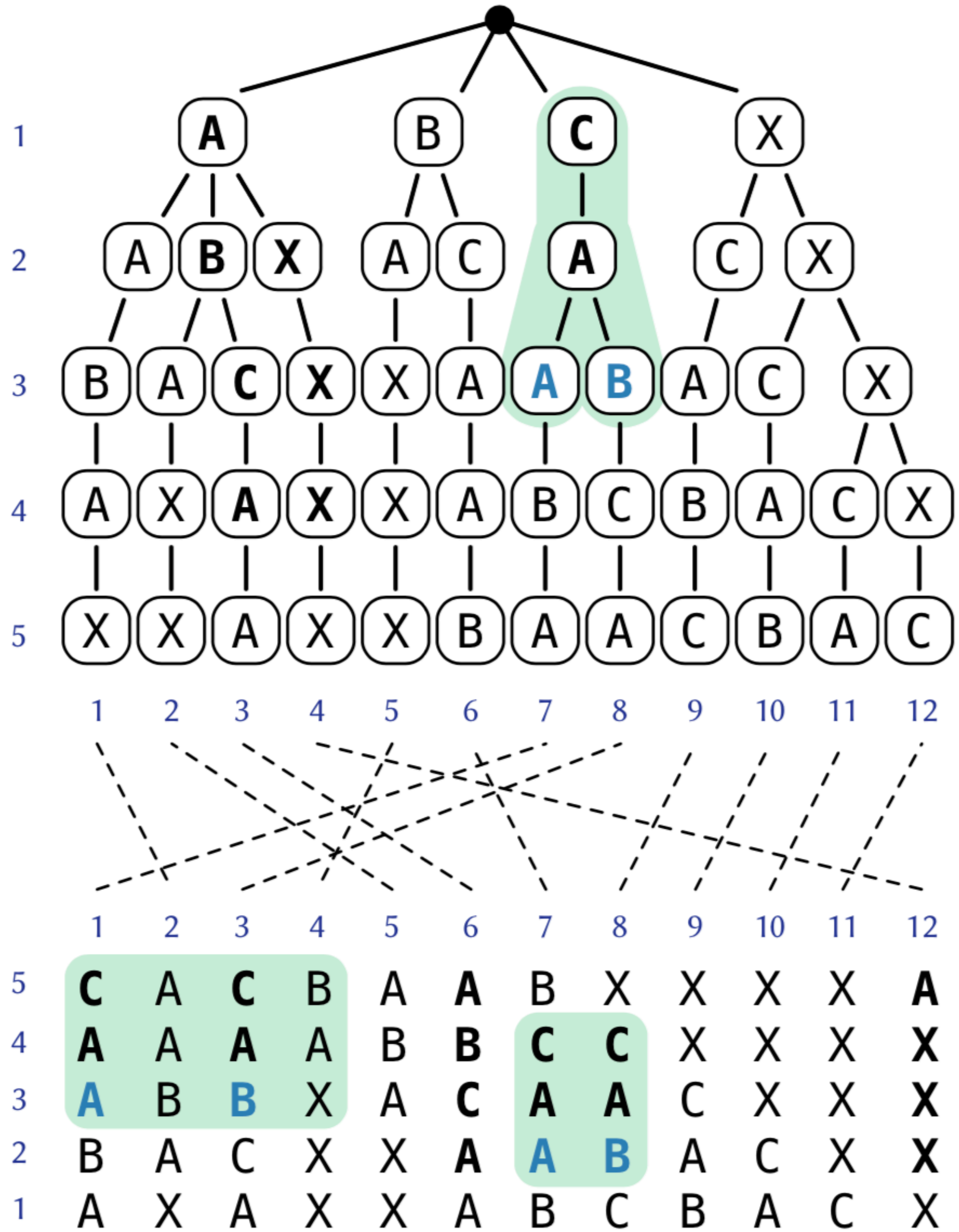
Suffix order

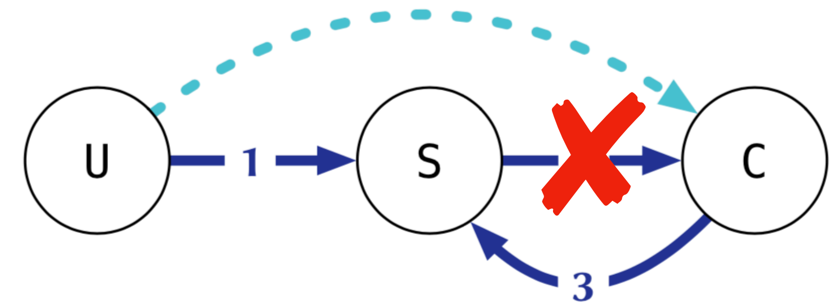
7	1	8	5	2	3	6	9	10	11	12	4
C	A	C	B	A	A	B	X	X	X	X	A
A	**A**	**A**	**A**	**B**	**B**	**C**	**C**	**X**	**X**	**X**	**X**
A	B	B	X	A	C	A	A	C	X	X	X
B	A	C	X	X	A	A	B	A	C	X	X
A	X	A	X	X	A	B	C	B	A	C	X

Context order

Using a scan of the block and O(|V|) space.

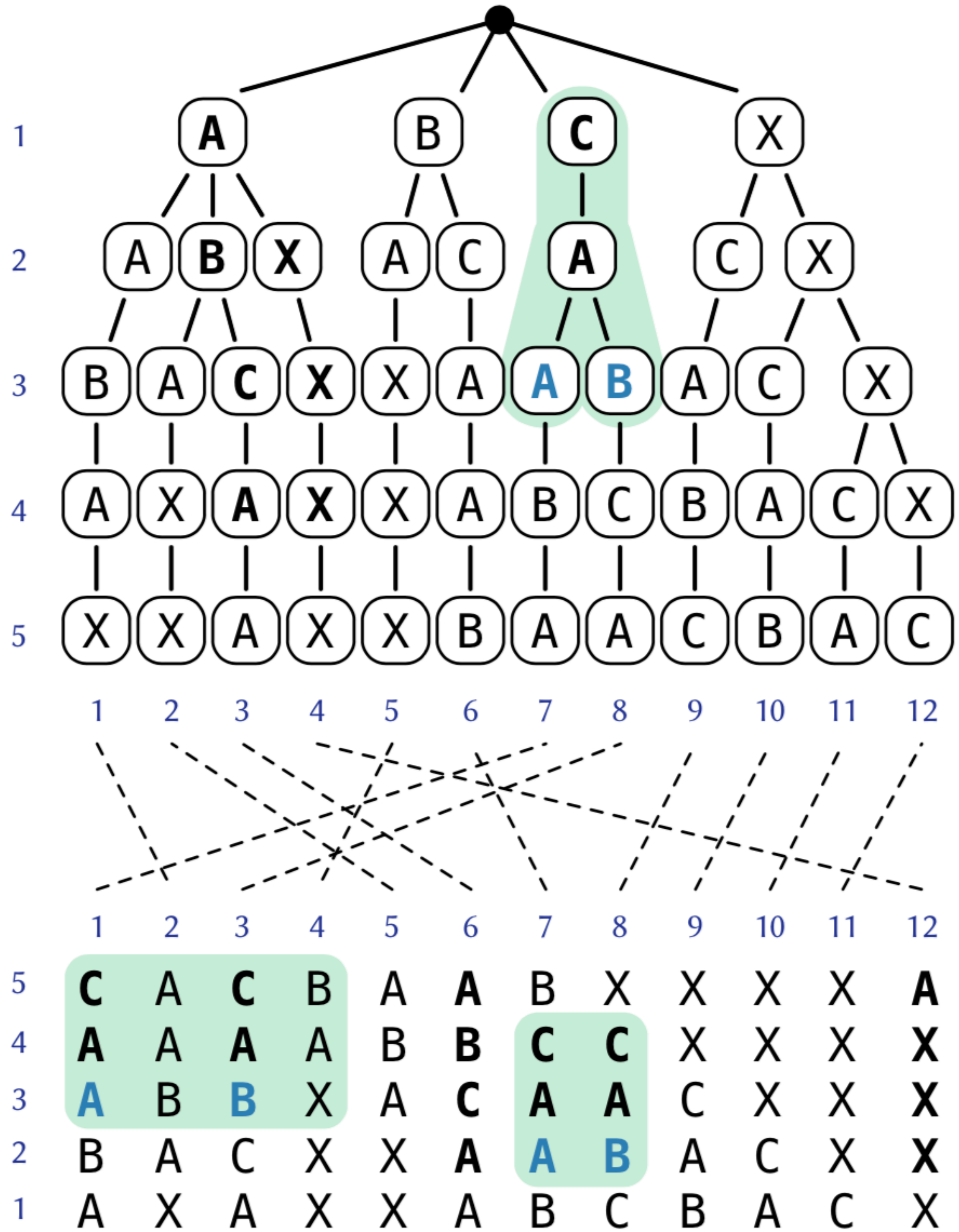
Idea 2 - Fast estimation in external memory (TOIS ’18)

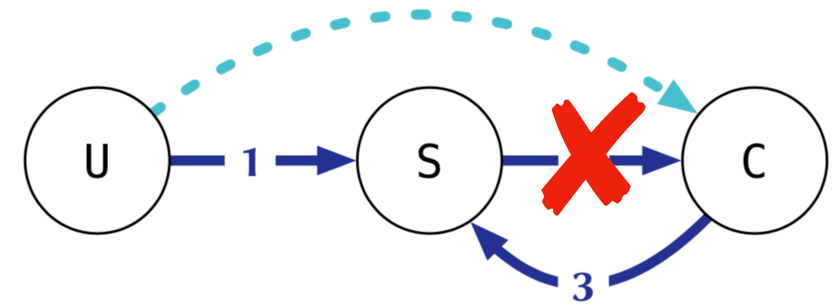




Rebuilding the last level of the trie.

Idea 2 - Fast estimation in external memory (TOIS ’18)

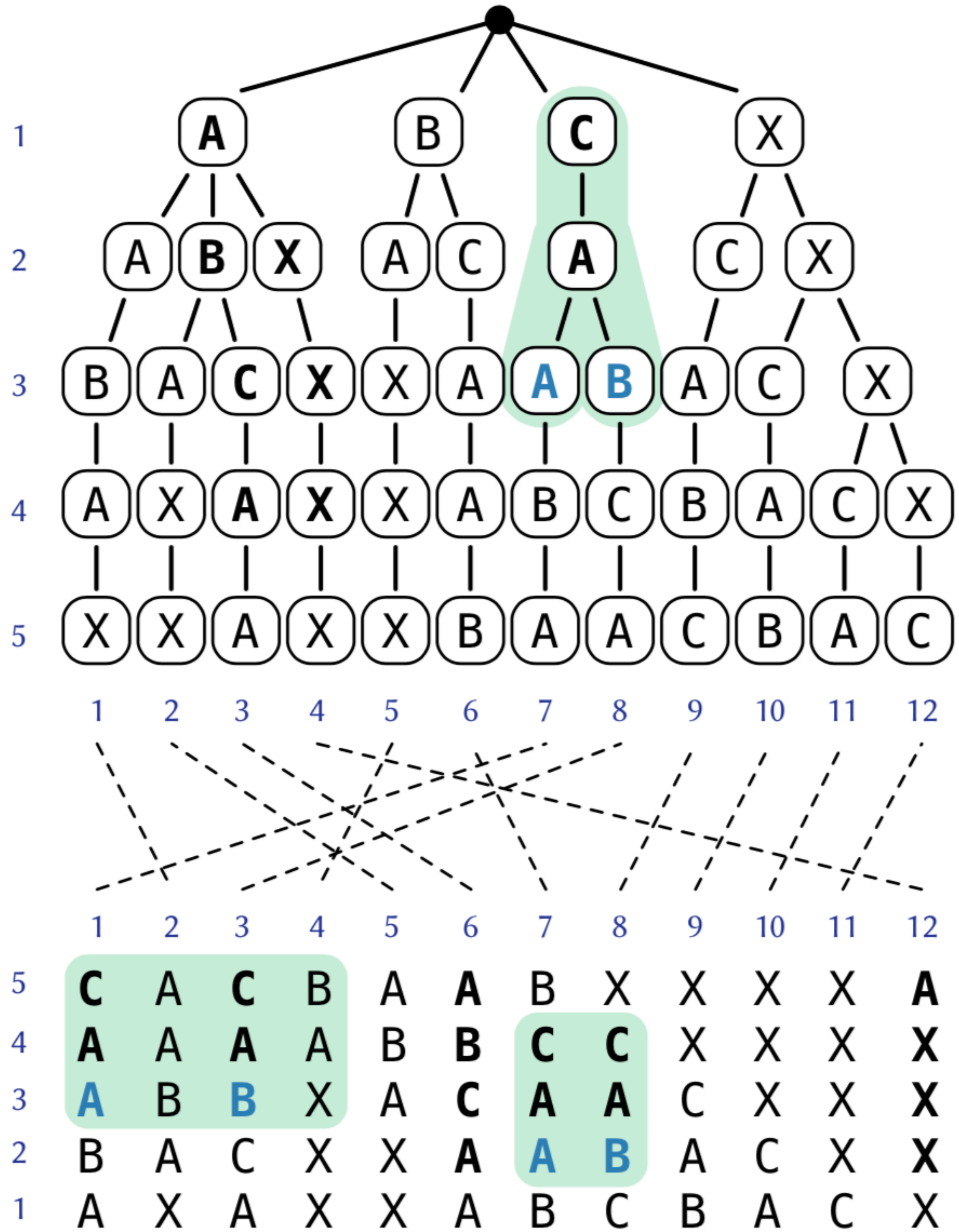


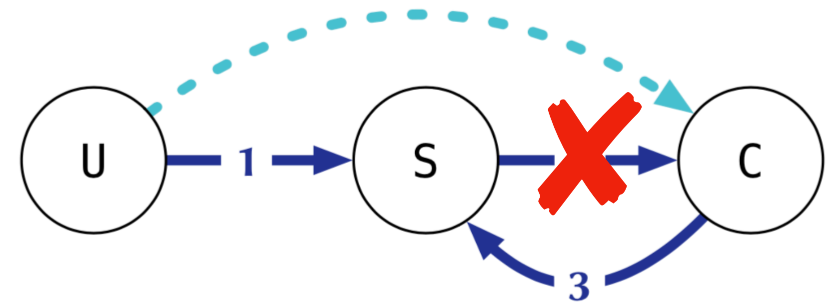


Rebuilding the last level of the trie.

A	4
B	2
C	2
X	4

Idea 2 - Fast estimation in external memory (TOIS ’18)





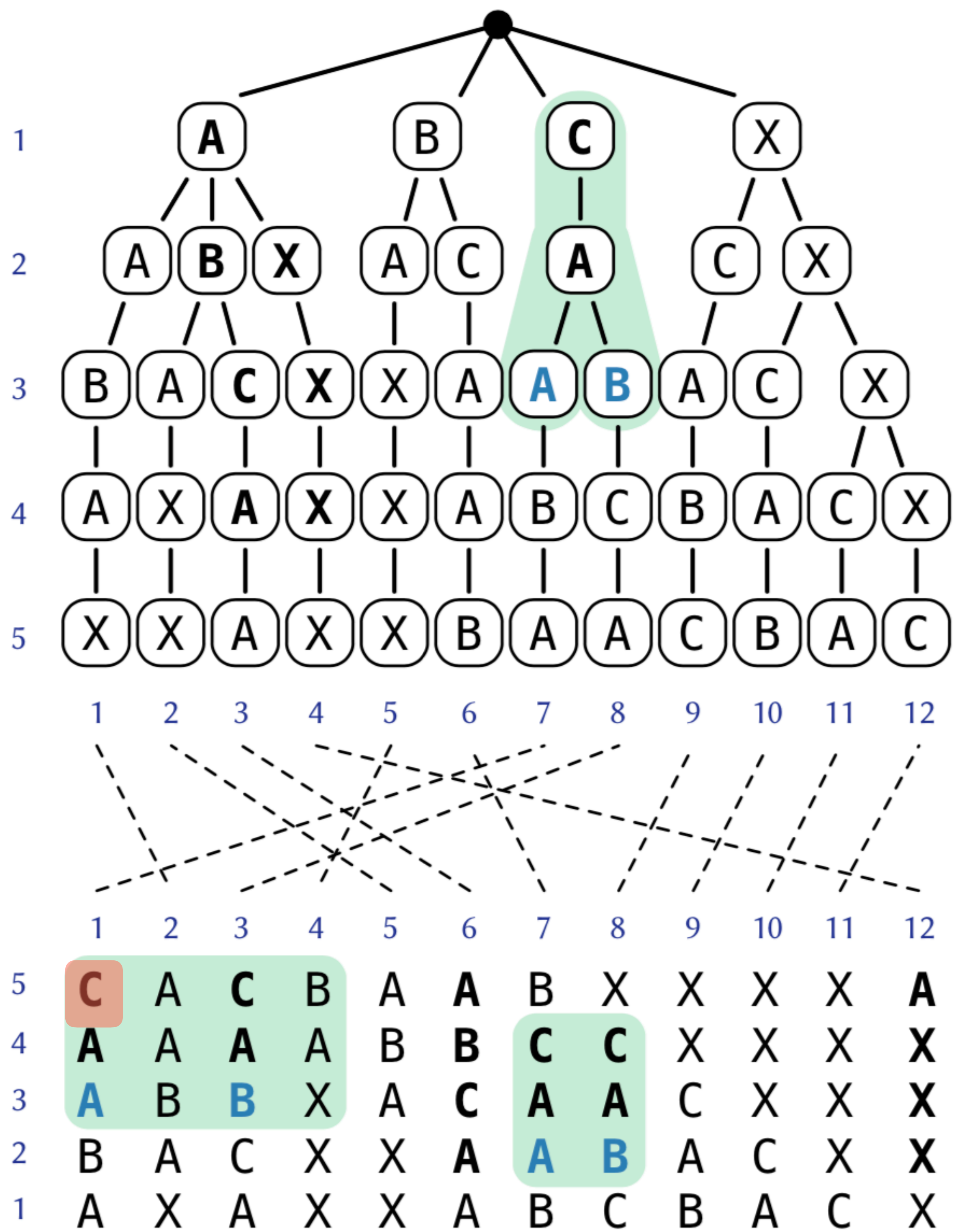
Rebuilding the last level of the trie.

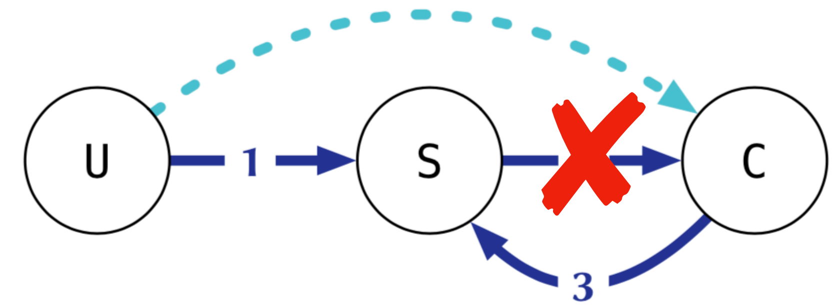
A	4
B	2
C	2
X	4

→

A	1
B	5
C	7
X	9

Idea 2 - Fast estimation in external memory (TOIS ’18)





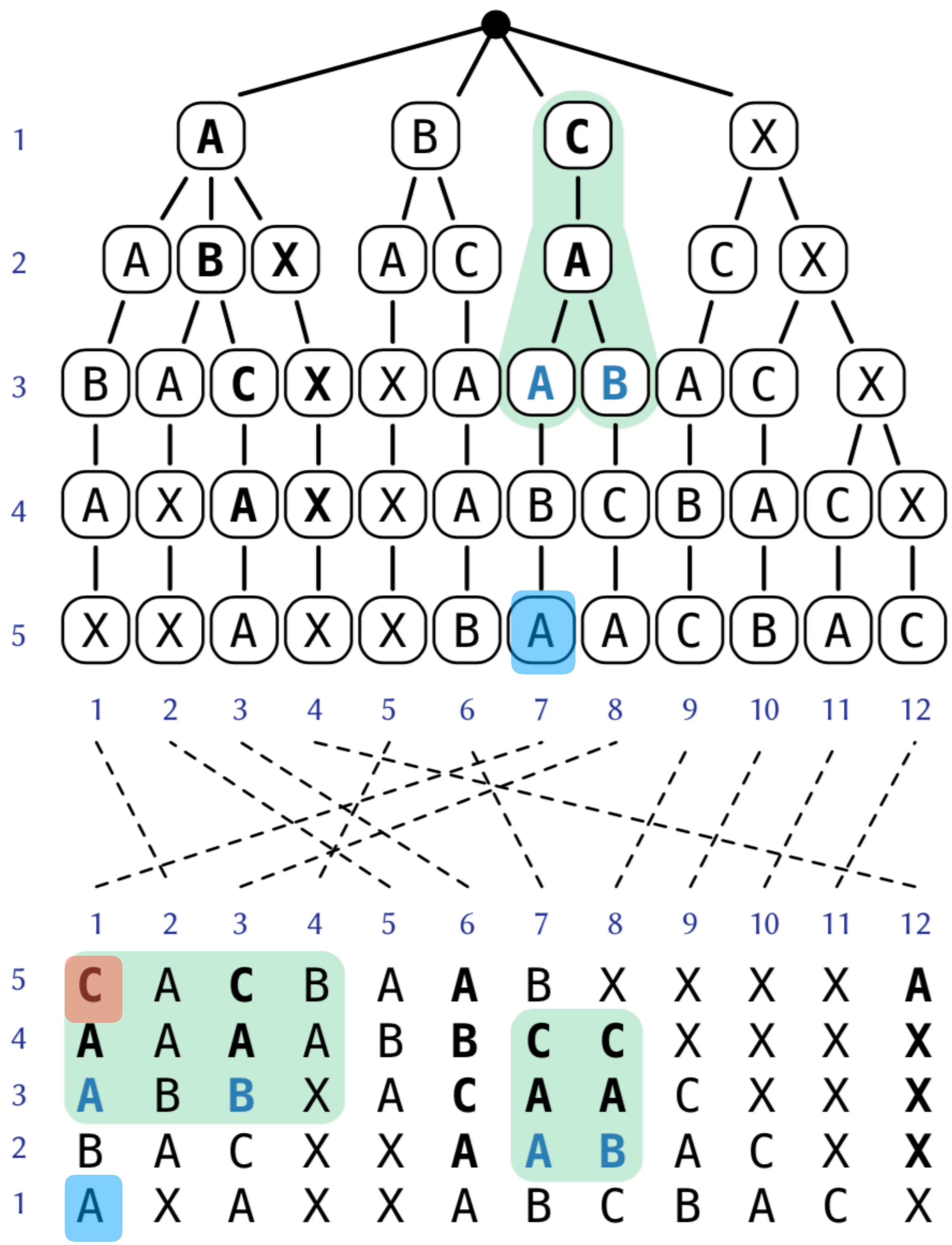
Rebuilding the last level of the trie.

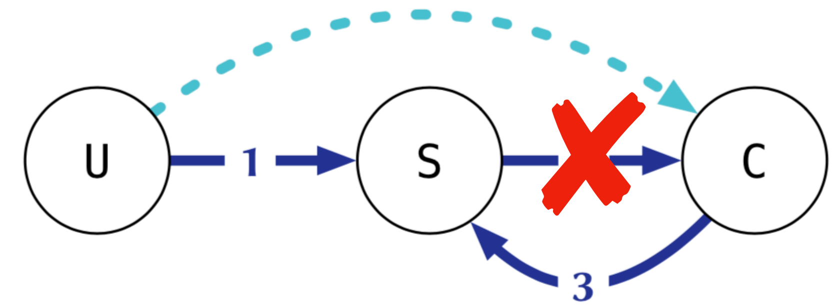
A	4
B	2
C	2
X	4

→

A	1
B	5
C	7
X	9

Idea 2 - Fast estimation in external memory (TOIS '18)





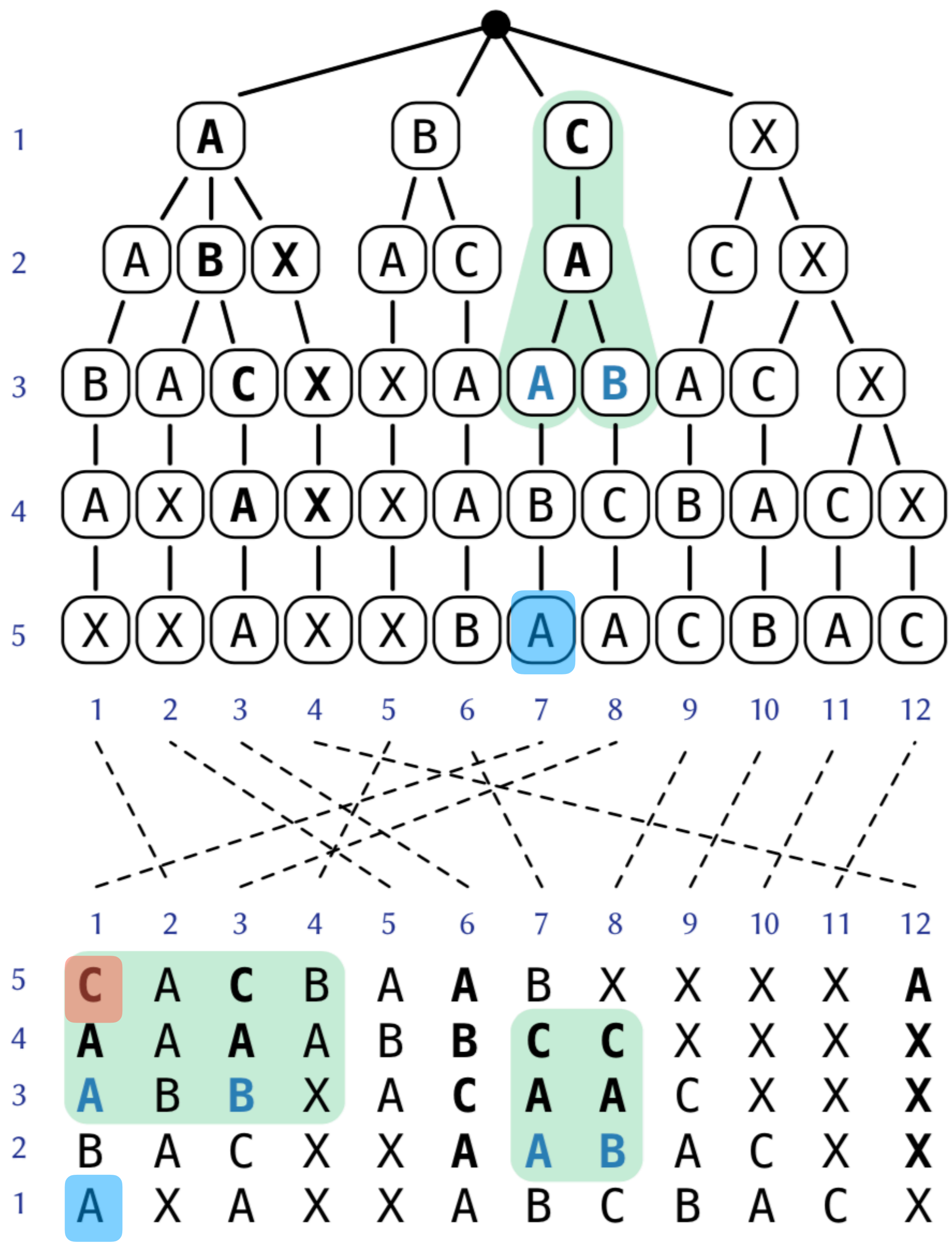
Rebuilding the last level of the trie.

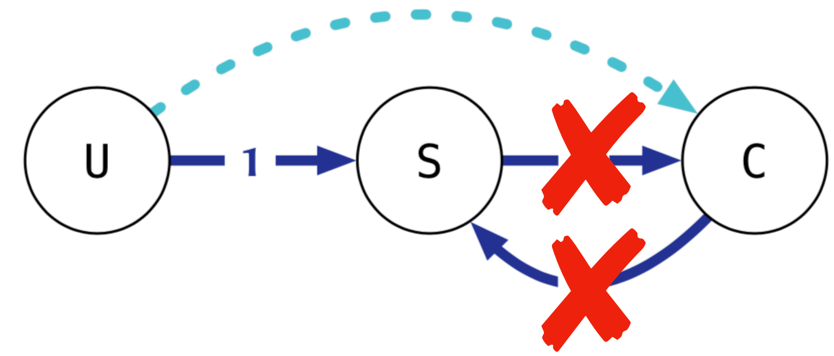
A	4
B	2
C	2
X	4

→

A	1
B	5
C	7
X	9

Idea 2 - Fast estimation in external memory (TOIS '18)





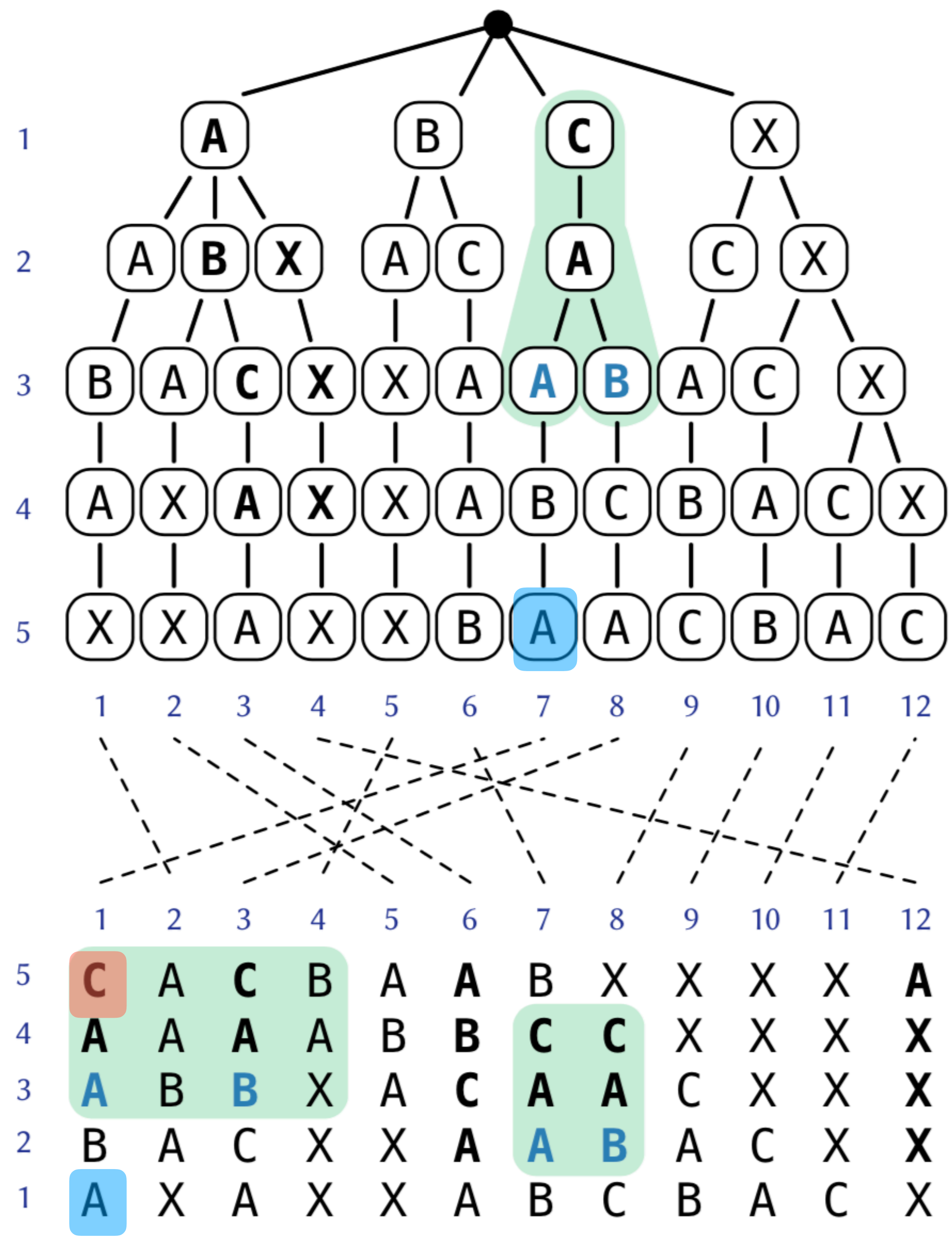
Rebuilding the last level of the trie.

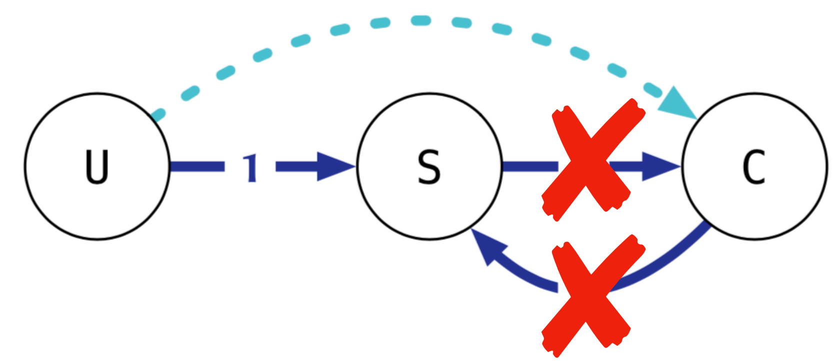
A	4
B	2
C	2
X	4

→

A	1
B	5
C	7
X	9

Idea 2 - Fast estimation in external memory (TOIS '18)





Rebuilding the last level of the trie.

A	4
B	2
C	2
X	4

→

A	1
B	5
C	7
X	9

Estimation runs 4.5X faster with billions of strings.

Thanks for your attention, time, patience!

Any questions?