Efficient Inference for Multinomial Mixed Membership Models

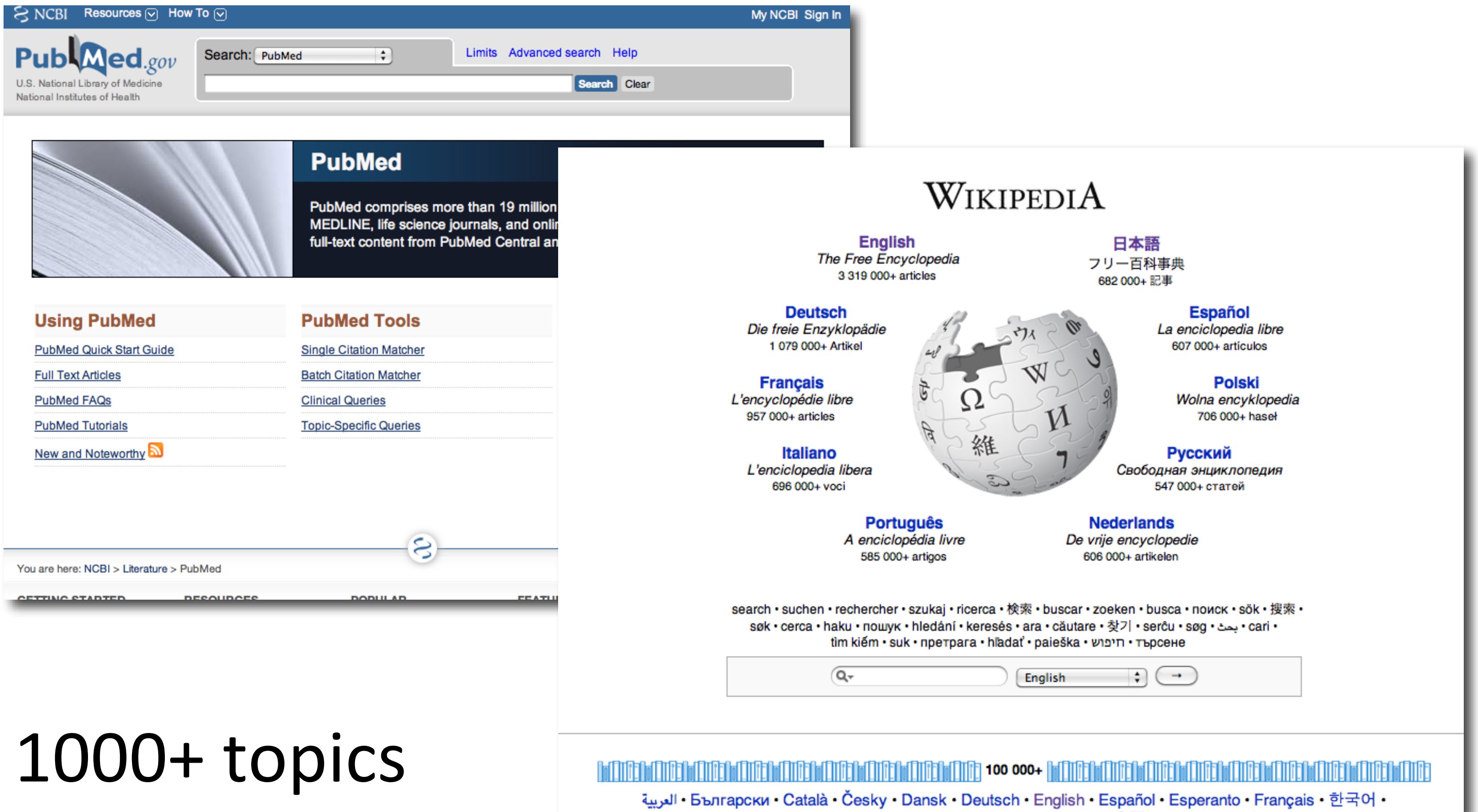
David Mimno
UMass, Amherst

Summary

- How **fast** can we make Gibbs sampling?
- How little **memory** can we use?

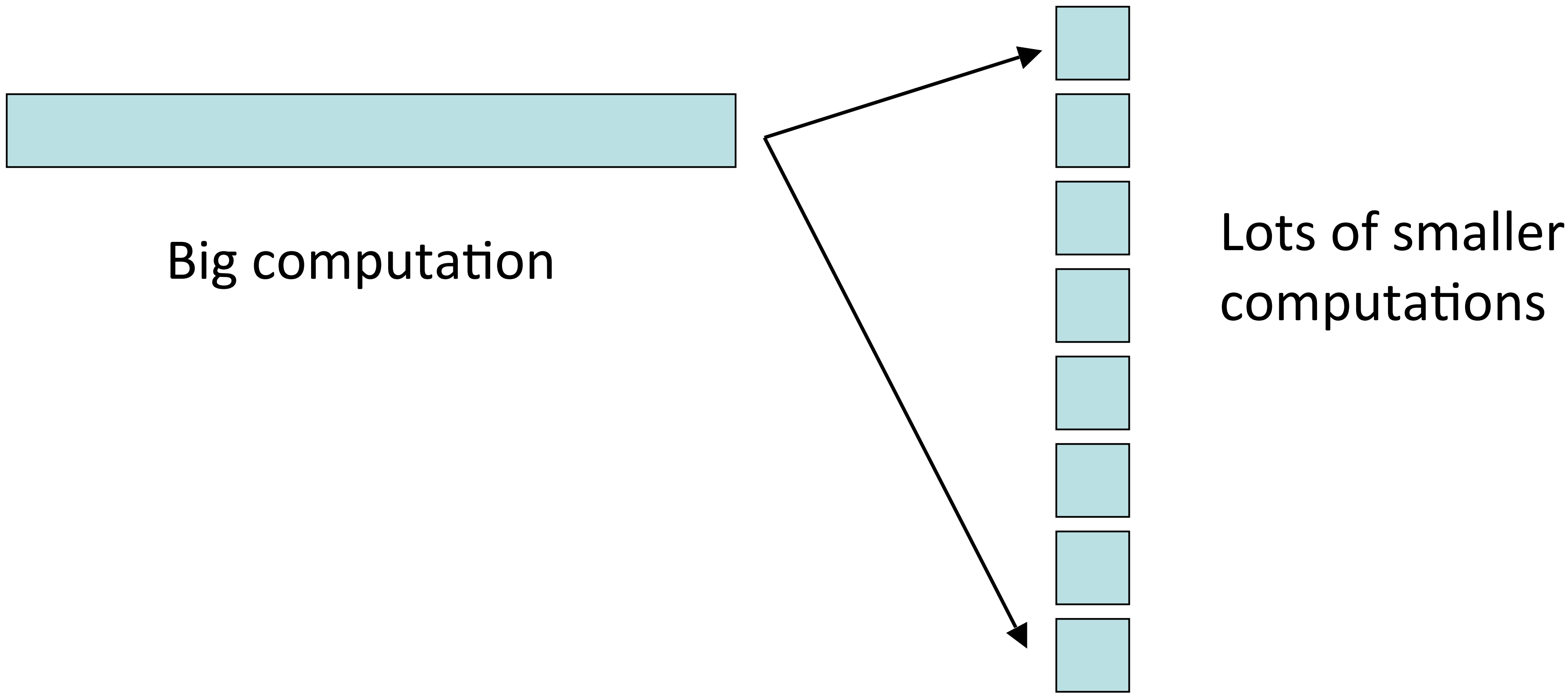
Goal: small collections should be fast,
large collections should be possible

Big collections! Lots of topics!

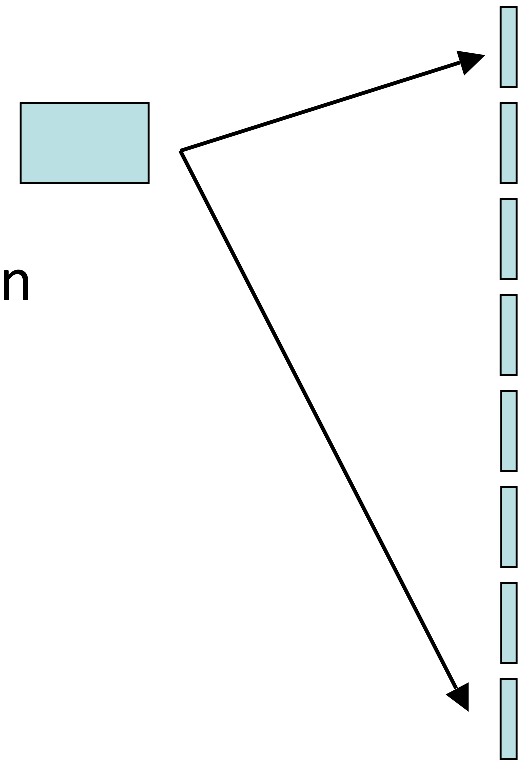


- 1000+ topics

Why not just parallelize?



Why not just parallelize?



Small computation

Lots of *really* small computations

Be smart, then parallelize

Gibbs Sampling for Topic Models

Doc A

dog	0
cat	2
feline	2
hound	0

Doc B

dog	0
dog	0
horse	1
equine	1

Document-topic statistics

Doc A

dog	0
cat	2
feline	2
hound	0

Doc B

dog	0
dog	0
horse	1
equine	1

topics →

docs ↓

	0	1	2
A	2	0	2
B	2	2	0

$N_{t|d}$

Topic-word statistics

Doc A

dog	0
cat	2
feline	2
hound	0

Doc B

dog	0
dog	0
horse	1
equine	1

Word types

Topics

	dog	cat	horse	hound	feline
0	3			1	
1			1		
2		1			1

$N_{w|t}$

score(t | $N_{t|d}$, $N_{w|t}$) =

$$\frac{N_{t|d} + \alpha}{N_d + T\alpha} \times \frac{N_{w|t} + \beta}{N_t + V\beta}$$

For each token,

Doc A

dog	
cat	2
feline	2
hound	0

for each possible topic [0, 1, 2],

Doc B

dog	0
dog	0
horse	1
equine	1

For each token,

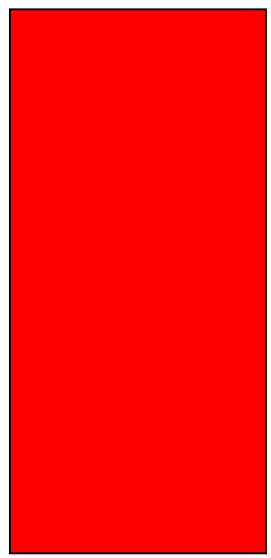
Doc A

dog	
cat	2
feline	2
hound	0

Doc B

dog	0
dog	0
horse	1
equine	1

for each possible topic [0, 1, 2], compute



$\text{score}(0 \mid N_{0|A}, N_{dog|0})$

For each token,

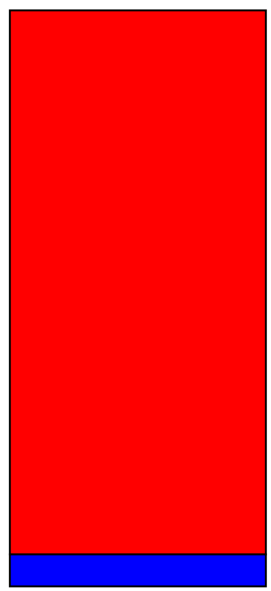
Doc A

dog	
cat	2
feline	2
hound	0

Doc B

dog	0
dog	0
horse	1
equine	1

for each possible topic [0, 1, 2], compute



score(0 | $N_{0|A}$, $N_{dog|0}$)

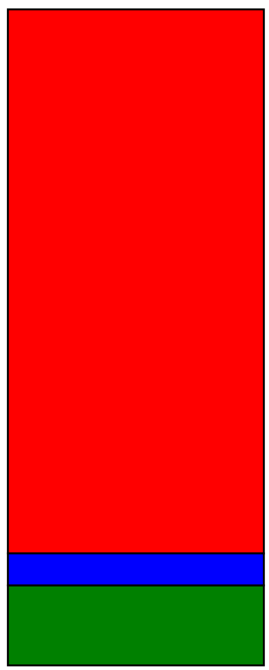
score(1 | $N_{1|A}$, $N_{dog|1}$)

For each token,

Doc A

dog	
cat	2
feline	2
hound	0

for each possible topic [0, 1, 2], compute



$\text{score}(0 \mid N_{0|A}, N_{dog|0})$

$\text{score}(1 \mid N_{1|A}, N_{dog|1})$

$\text{score}(2 \mid N_{2|A}, N_{dog|2})$

Doc B

dog	0
dog	0
horse	1
equine	1

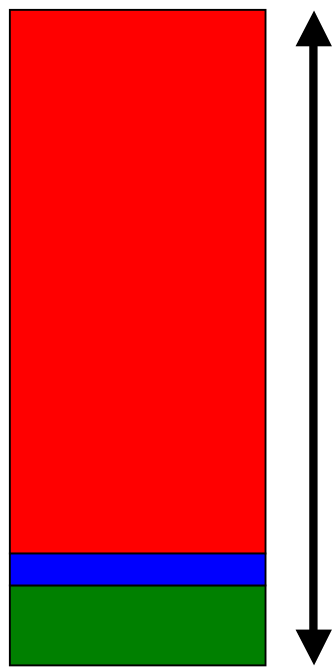
Add up the scores

Doc A

dog	
cat	2
feline	2
hound	0

Doc B

dog	0
dog	0
horse	1
equine	1



$$Z = \text{score}(0 \mid N_{0|A}, N_{dog|0}) + \text{score}(1 \mid N_{1|A}, N_{dog|1}) + \text{score}(2 \mid N_{2|A}, N_{dog|2})$$

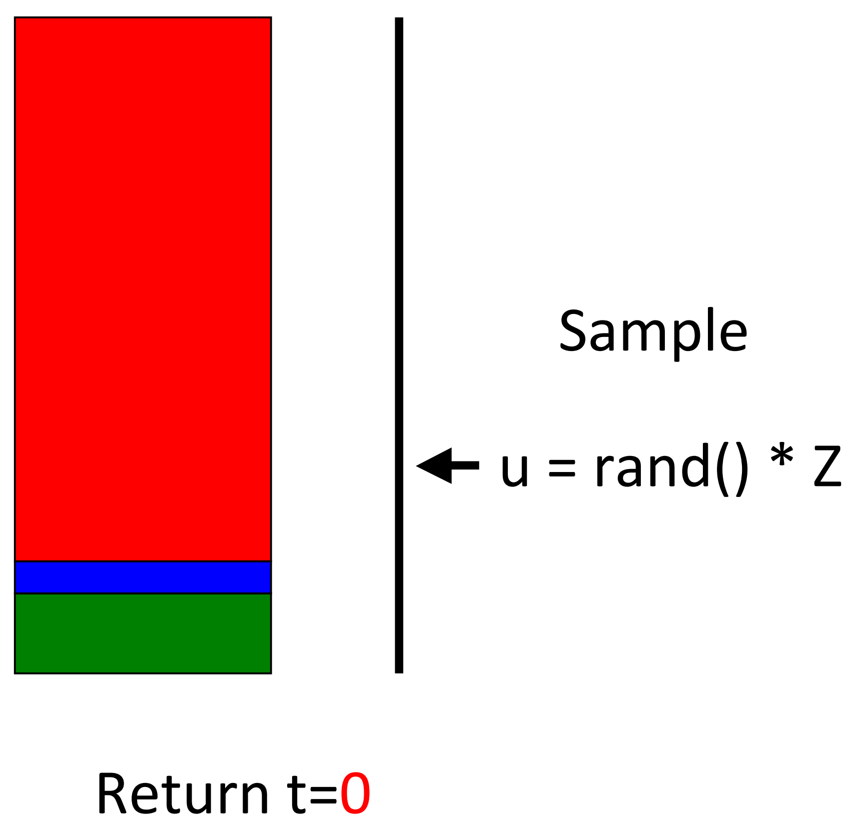
Sample a new topic

Doc A

dog	
cat	2
feline	2
hound	0

Doc B

dog	0
dog	0
horse	1
equine	1



For each token,

Doc A

dog	0
cat	
feline	2
hound	0

for each possible topic [0, 1, 2],

Doc B

dog	0
dog	0
horse	1
equine	1

Summary so far

- For every token,
  - For each possible topic *t* calculate score(t | …)
- Add up scores to **normalizing constant**

$$Z = \sum_t \text{score}(t)$$

- Sample u ~ U(0, Z) and return the corresponding topic *t*.

Performance is dominated by calculation of *Z*

The normalizing constant

$$Z = \sum_t \frac{N_{t|d} + \alpha}{N_d + T\alpha} \times \frac{N_{w|t} + \beta}{N_t + V\beta}$$

The normalizing constant

$$Z = \sum_t \frac{(N_{t|d} + \alpha)(N_{w|t} + \beta)}{N_t + V\beta}$$

The normalizing constant

$$Z = \sum_t \frac{N_{t|d}N_{w|t} + \beta N_{t|d} + \alpha N_{w|t} + \alpha\beta}{N_t + V\beta}$$

The normalizing constant

$$Z = \sum_t \frac{N_{w|t}(N_{t|d} + \alpha)}{N_t + V\beta} + \sum_t \frac{\beta N_{t|d}}{N_t + V\beta} + \sum_t \frac{\alpha\beta}{N_t + V\beta}$$

The normalizing constant

Token-specific

Independent of word and document

$$Z = \sum_t \frac{N_{w|t}(N_{t|d} + \alpha)}{N_t + V\beta} + \sum_t \frac{\beta N_{t|d}}{N_t + V\beta} + \sum_t \frac{\alpha\beta}{N_t + V\beta}$$

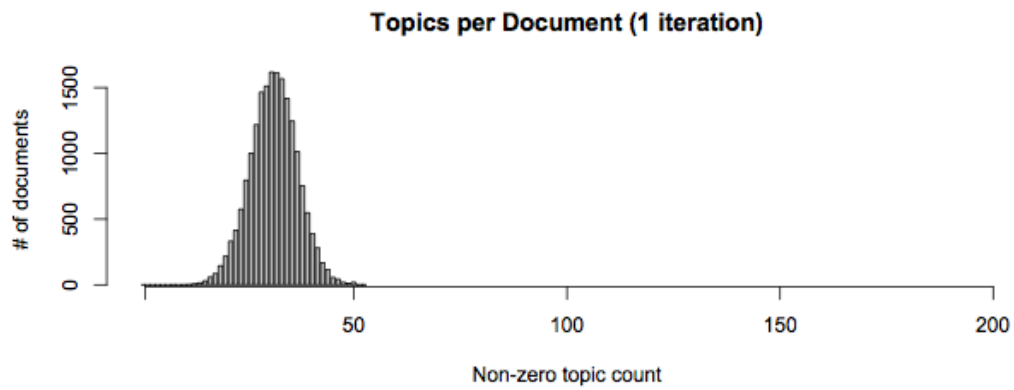
Document-specific

The normalizing constant

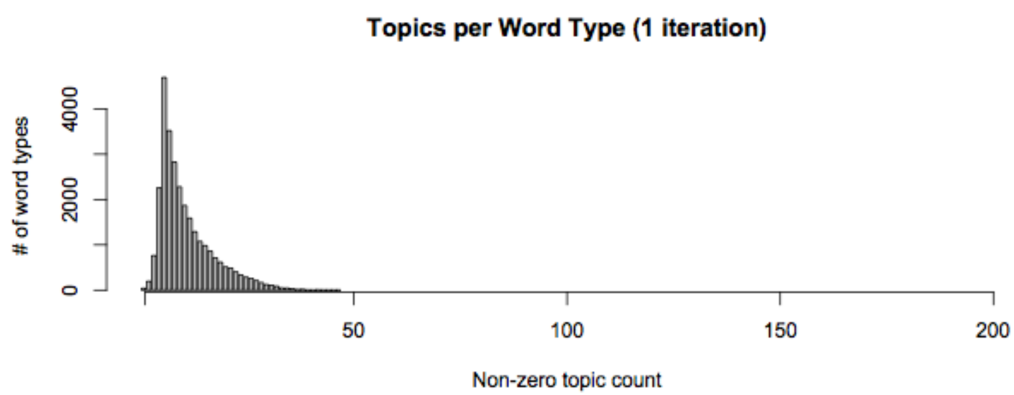
$$Z = \sum_{\boxed{t:N_{w|t}>0}} \frac{N_{w|t}(N_{t|d}+\alpha)}{N_t+V\beta} +$$

$$\sum_{\boxed{t:N_{t|d}>0}} \frac{\beta N_{t|d}}{N_t+V\beta} + \sum_{\boxed{t}} \frac{\alpha\beta}{N_t+V\beta}$$

Statistics are sparse



- $N_{t|d}$: 10-20%
- mostly zeros



- $N_{w|t}$: < 5%
- almost all zeros

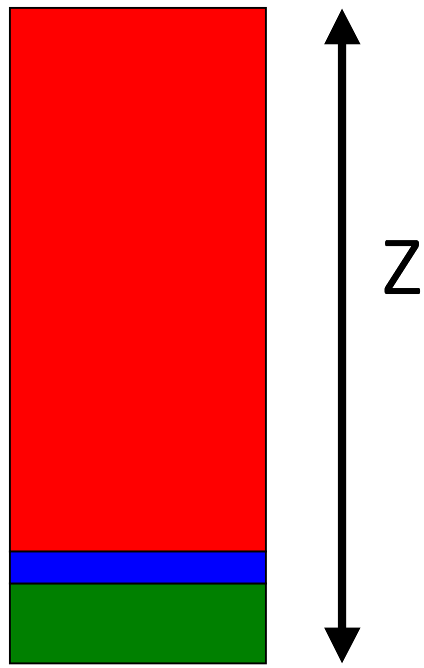
Add up the scores

Doc A

dog	
cat	2
feline	2
hound	0

Doc B

dog	0
dog	0
horse	1
equine	1



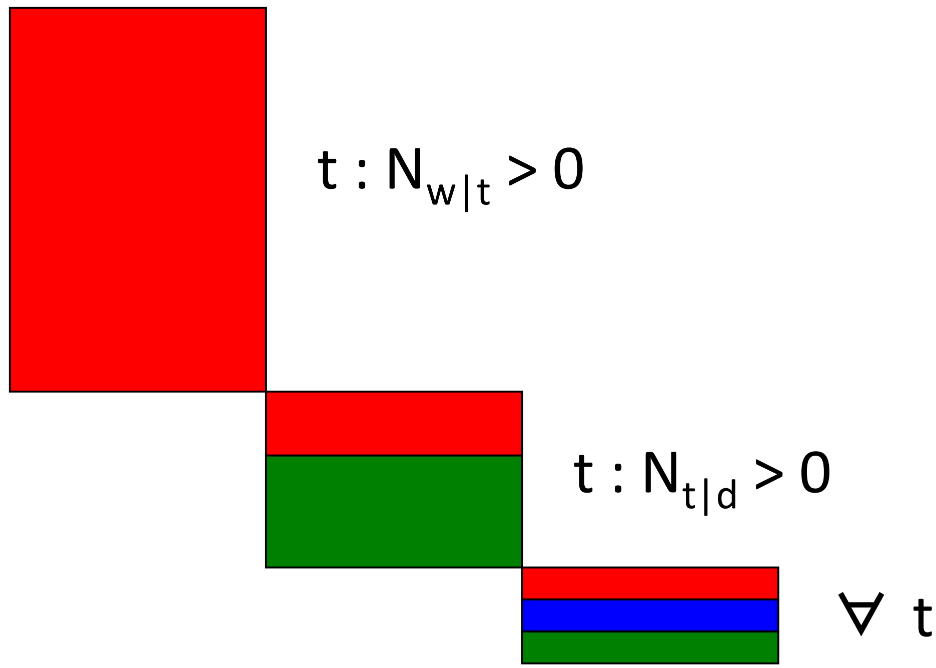
Add up the scores, in blocks

Doc A

dog	
cat	2
feline	2
hound	0

Doc B

dog	0
dog	0
horse	1
equine	1



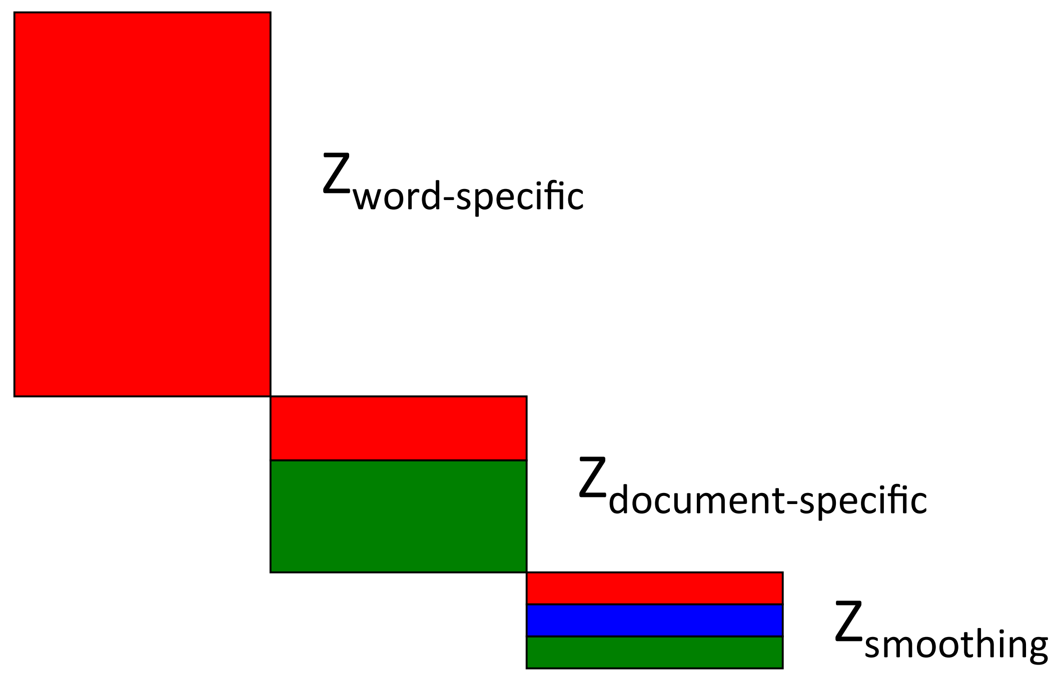
Add up the scores, in blocks

Doc A

dog	
cat	2
feline	2
hound	0

Doc B

dog	0
dog	0
horse	1
equine	1



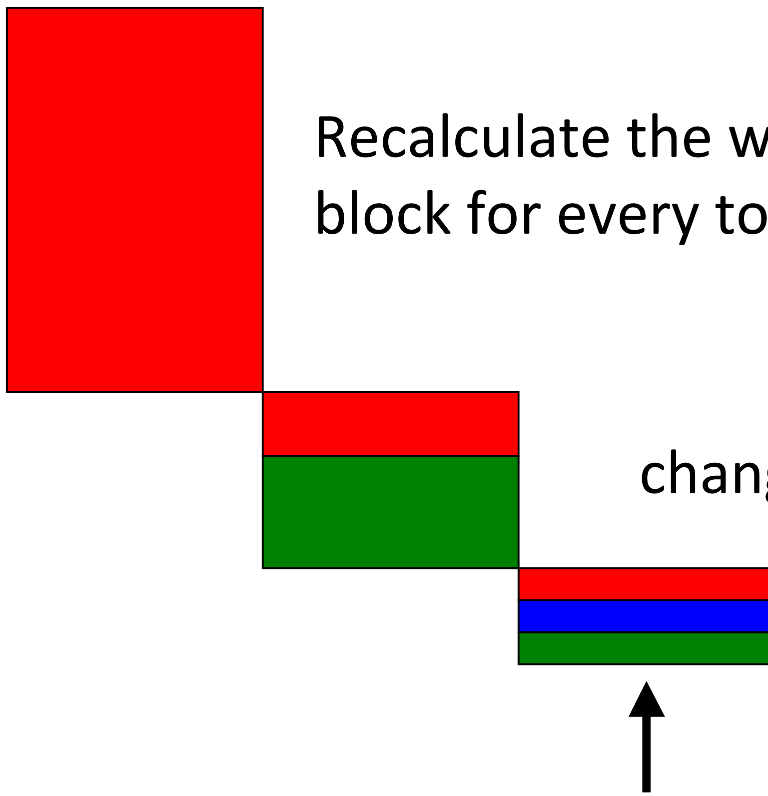
Add up *some* blocks, cache others

Doc A

dog	
cat	2
feline	2
hound	0

Doc B

dog	0
dog	0
horse	1
equine	1



Recalculate the word-specific block for every token

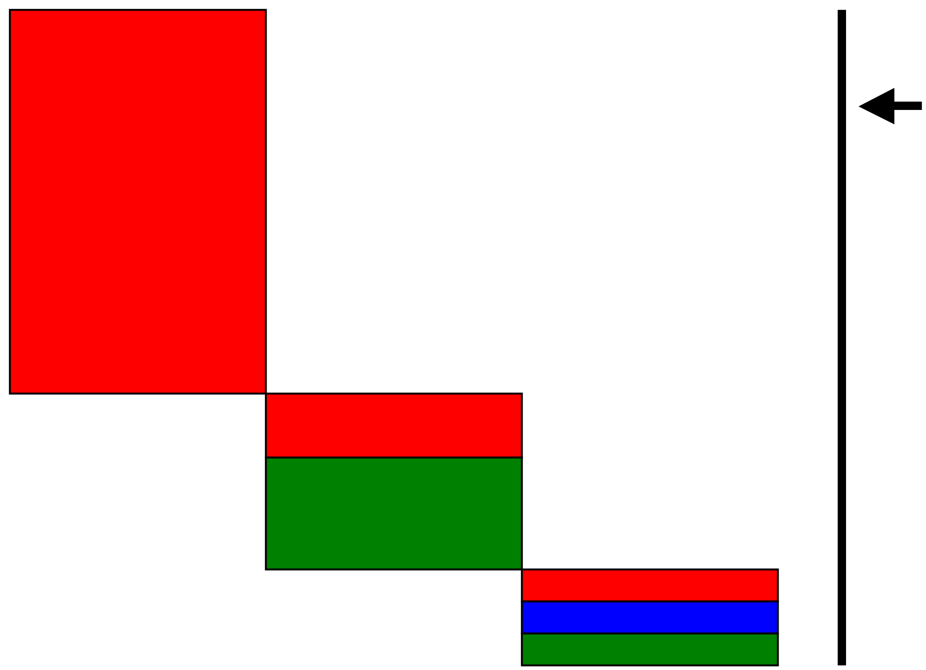
change *at most* two from each of these

The size of this block is *almost* constant

Add up *some* blocks, cache others

Doc A

dog	
cat	2
feline	2
hound	0



Doc B

dog	0
dog	0
horse	1
equine	1

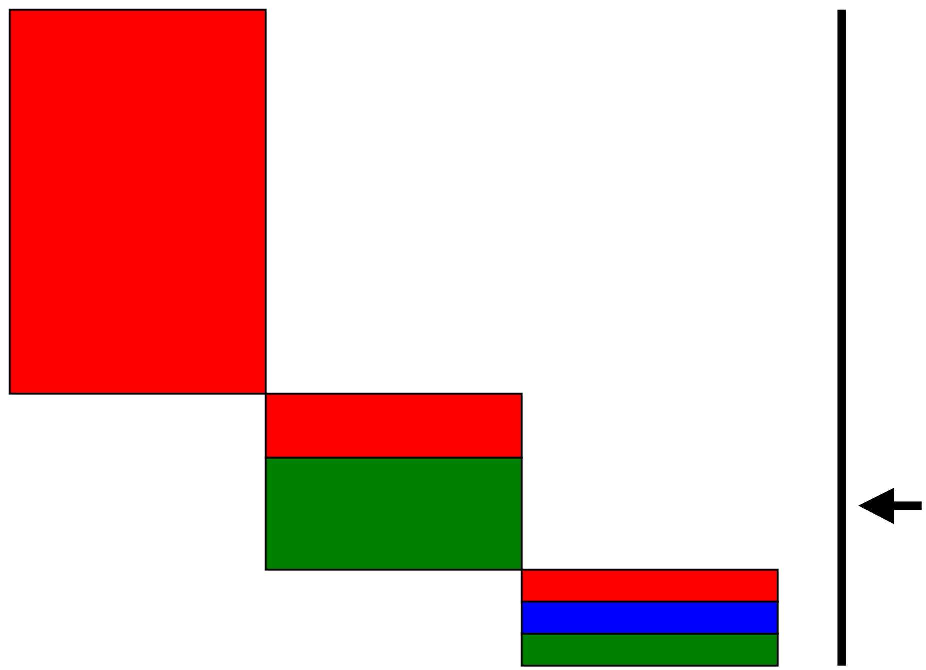
Add up *some* blocks, cache others

Doc A

dog	
cat	2
feline	2
hound	0

Doc B

dog	0
dog	0
horse	1
equine	1



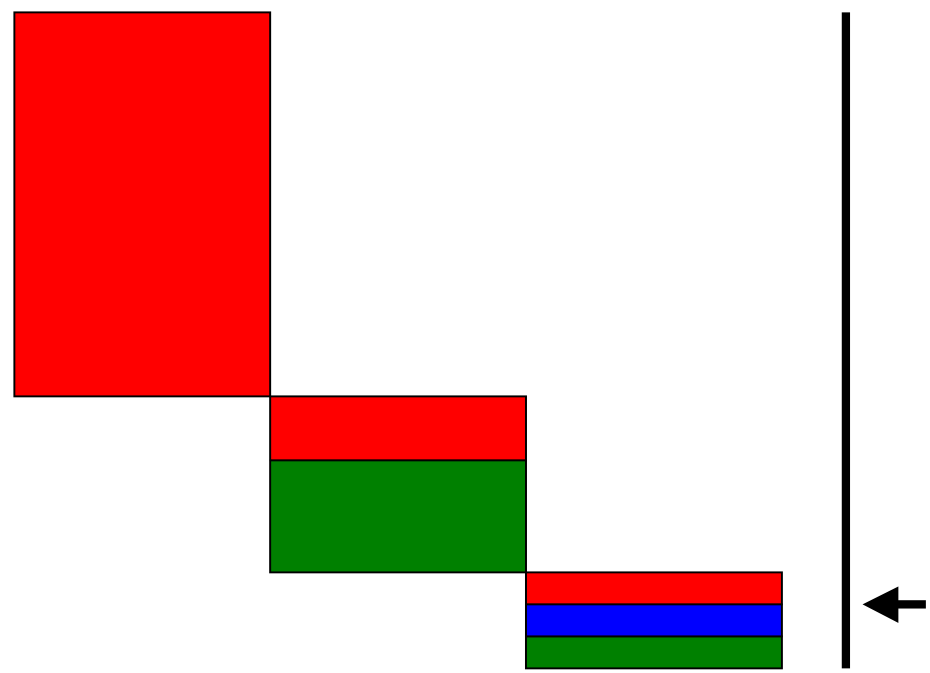
Add up *some* blocks, cache others

Doc A

dog	
cat	2
feline	2
hound	0

Doc B

dog	0
dog	0
horse	1
equine	1



Worst case: we might have to loop over all topics

Summary so far

- We can store most of the computation to calculate *Z* from token to token.
- We can sample exactly from the same distribution, with a new map from (0, *Z*) to topics.

Fast iteration over $\{t : N_{w|t} > 0\}$ is *critical*

Topic-word statistics

Doc A

dog	0
cat	2
feline	2
hound	0

Doc B

dog	0
dog	0
horse	1
equine	1

Word types

Topics

	dog	cat	horse	hound	feline
0	3			1	
1			1		
2		1			1

$N_{w|t}$

Word-topic statistics *transpose*

Doc A

dog	0
cat	2
feline	2
hound	0

Doc B

dog	0
dog	0
horse	1
equine	1

Topics →

Word types ↓

	0	1	2
dog	3		
cat			1
horse		1	
hound	1		
feline			1

$N_{w|t}$

Representations of $N_{w|t}$: arrays

```
int[] typeTopicCounts = new int[T];
```

Topics index array positions.

0	1	2	3
3	0	0	1

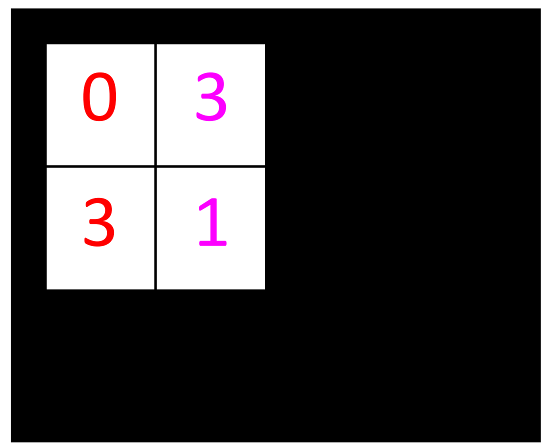
Most entries are zero.

Integer HashMaps

```
TIntIntHashMap typeTopicCounts =
     new TIntIntHashMap();
```

Faster, but…

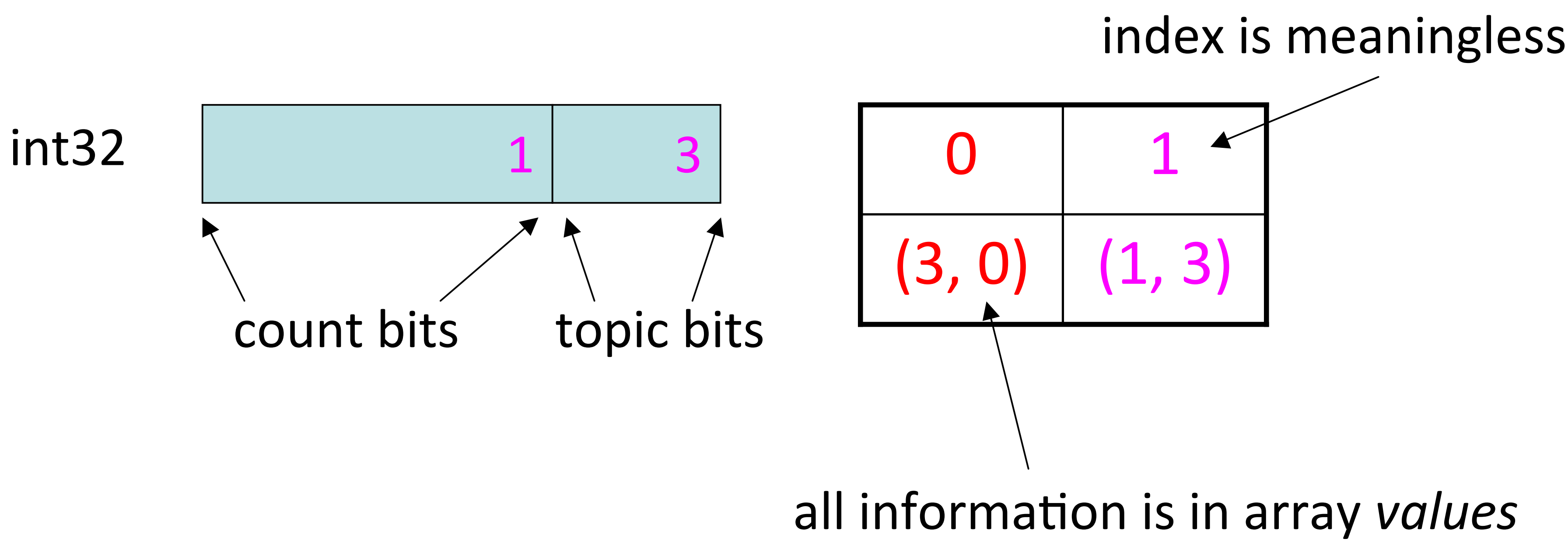
- Complicated
- Not much memory improvement
- Adds dependencies on external libraries (trove, fastutil).



(giant black box)

Encoded integer arrays

```
int[] typeTopicCounts = new int[N_w]
```



Huge speedup, 2x faster than HashMaps

Example

	…	8	…	15	…	83	…	96	…
“dog”	0	3	0	24	0	9	0	1	0

100 topics, so 4x100 bytes = 400

Example

	…	8	…	15	…	83	…	96	…
“dog”	0	3	0	24	0	9	0	1	0

0	1	2	3
(24, 15)	(9, 83)	(3, 8)	(1, 96)

4 non-zero topics, 4 x 4 bytes = 16!

Example

	…	8	…	15	…	83	…	96	…
“dog”	0	3	0	24	0	9	0	1	0

0	1	2	3
24<<7 + 15	9<<7 + 83	3<<7 + 8	1<<7 + 96

100 topics, $2^7 > 100$

Basic Operations: iteration

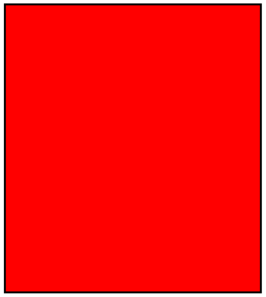
0	1	2	3
(24, 15)	(9, 83)	(3, 8)	(1, 96)

$t = v \ \& \ 0x1111111$

$N_{w|t} = v >> 7$

Basic Operations: iteration

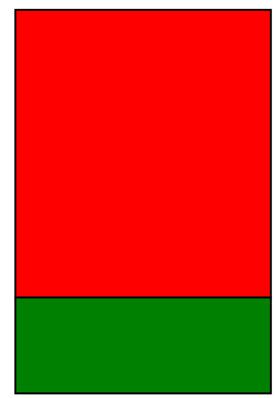
0	1	2	3
(24, 15)	(9, 83)	(3, 8)	(1, 96)

$Z_{\text{word-specific}}$ = 

Basic Operations: iteration

0	1	2	3
(24, 15)	(9, 83)	(3, 8)	(1, 96)

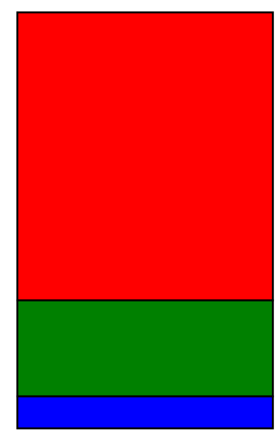
$Z_{\text{word-specific}} =$



Basic Operations: iteration

0	1	2	3
(24, 15)	(9, 83)	(3, 8)	(1, 96)

$Z_{\text{word-specific}} =$



Basic Operations: increment

0	1	2	3
(24, 15)	(9, 83)	(3, 8)	(1, 96)

$N_{w|83}$++

Basic Operations: increment

0	1	2	3
(24, 15)	(9, 83)	(3, 8)	(1, 96)

```
v = (9+1)<<7 + 83
```

$N_{w|83}$++

Basic Operations: increment

0	1	2	3
(24, 15)	(10, 83)	(3, 8)	(1, 96)

```
v = (9+1)<<7 + 83
```

$N_{w|83}$++

Basic Operations: increment

0	1	2	3
(**1**, 15)	(**1**, 83)	(**1**, 96)	0

```
v = (1+1)<<7 + 83
```

$N_{w|83}$++

Basic Operations: increment

0	1	2	3
(1, 15)	(1, 83)	(1, 96)	0

```
v = (1+1)<<7 + 83
```

$N_{w|83}$++

Basic Operations: increment

0	1	2	3
(1, 15)	(2, 83)	(1, 96)	0

$N_{w|83}$++

Basic Operations: increment

0	1	2	3
(2, 83)	(1, 15)	(1, 96)	0

$N_{w|83}$++

Basic Operations: decrement

0	1	2	3
(1, 15)	(1, 83)	(1, 96)	0

`v = (1–1)<<7 + 83`

$N_{w|83}$--

Basic Operations: decrement

0	1	2	3
(1, 15)	0	(1, 96)	0

```
v = (1−1)<<7 + 83
```

$N_{w|83}$--

Basic Operations: decrement

0	1	2	3
(1, 15)	(1, 96)	0	0

$N_{w|83}$--

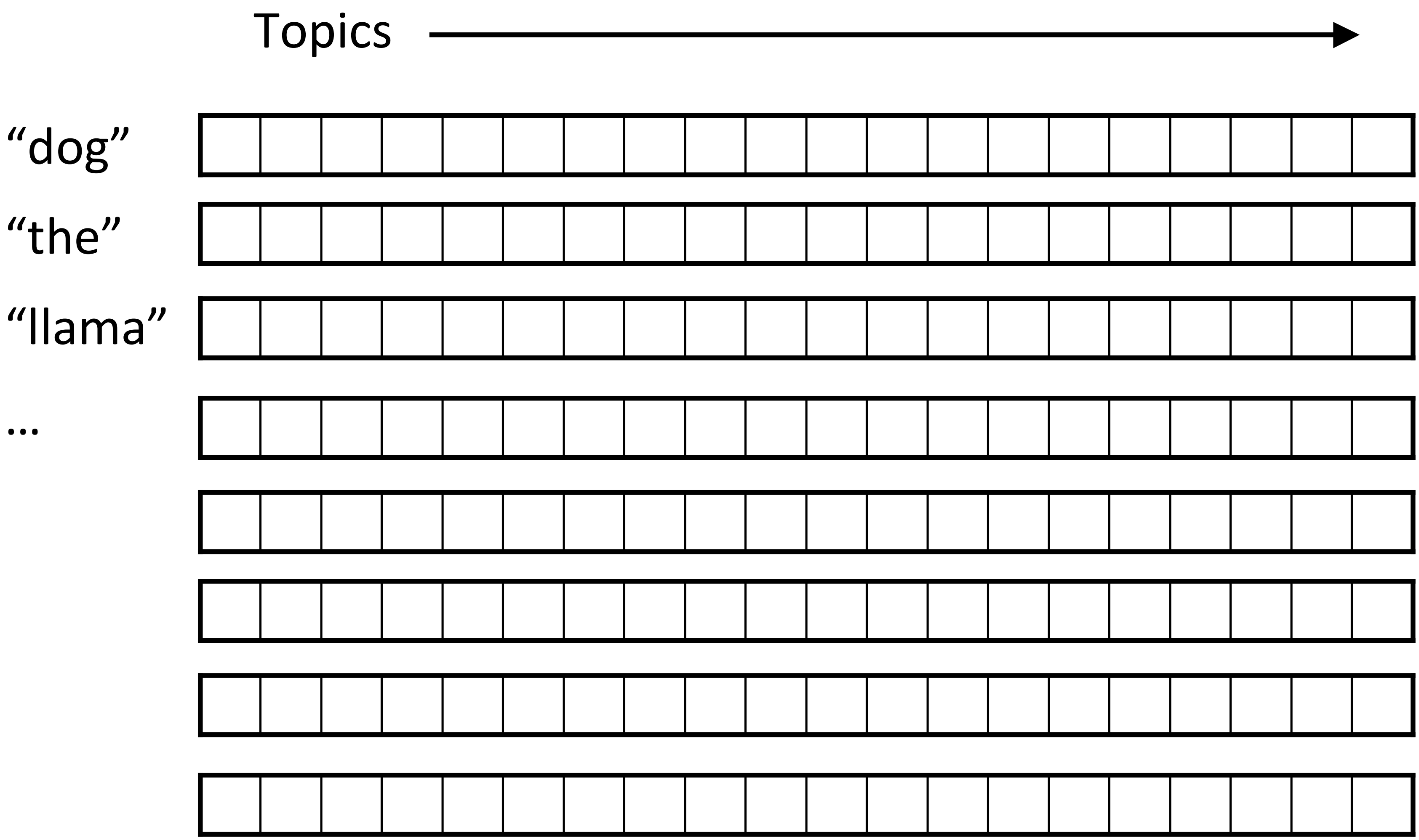
Basic Operations

- Iterate:
  - linear in number of *non-zero* topics
- Increment and decrement:
  - theoretically the same as iterate, but usually constant

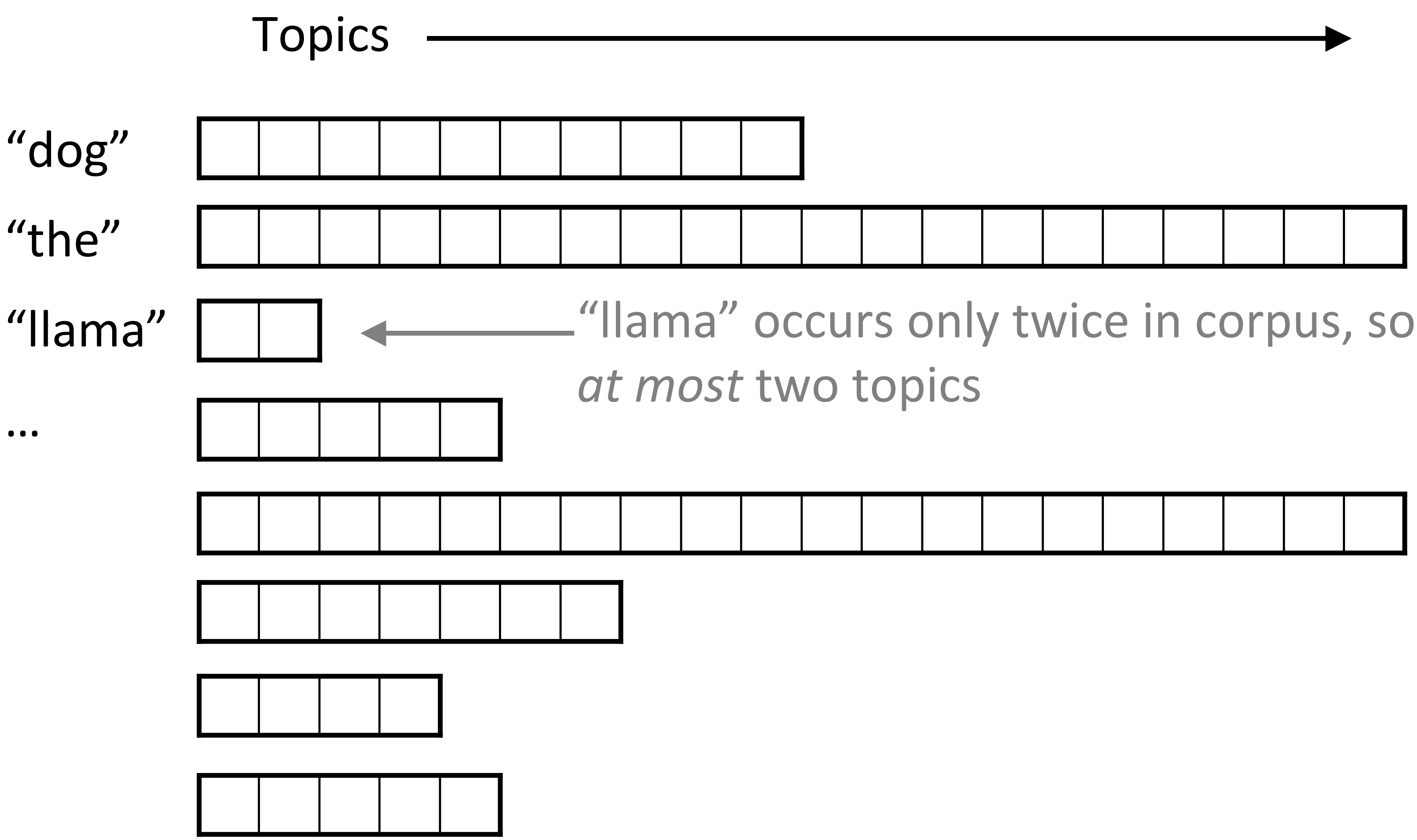
Memory allocation

- Conservative:
  - Count total occurrences of each word type: N_w
  - For each word type, allocate `int[min(T, N_w)]`
- More savings:
  - For each type allocate `int[min(T, N_w)/k]`, increase memory as needed (or not)

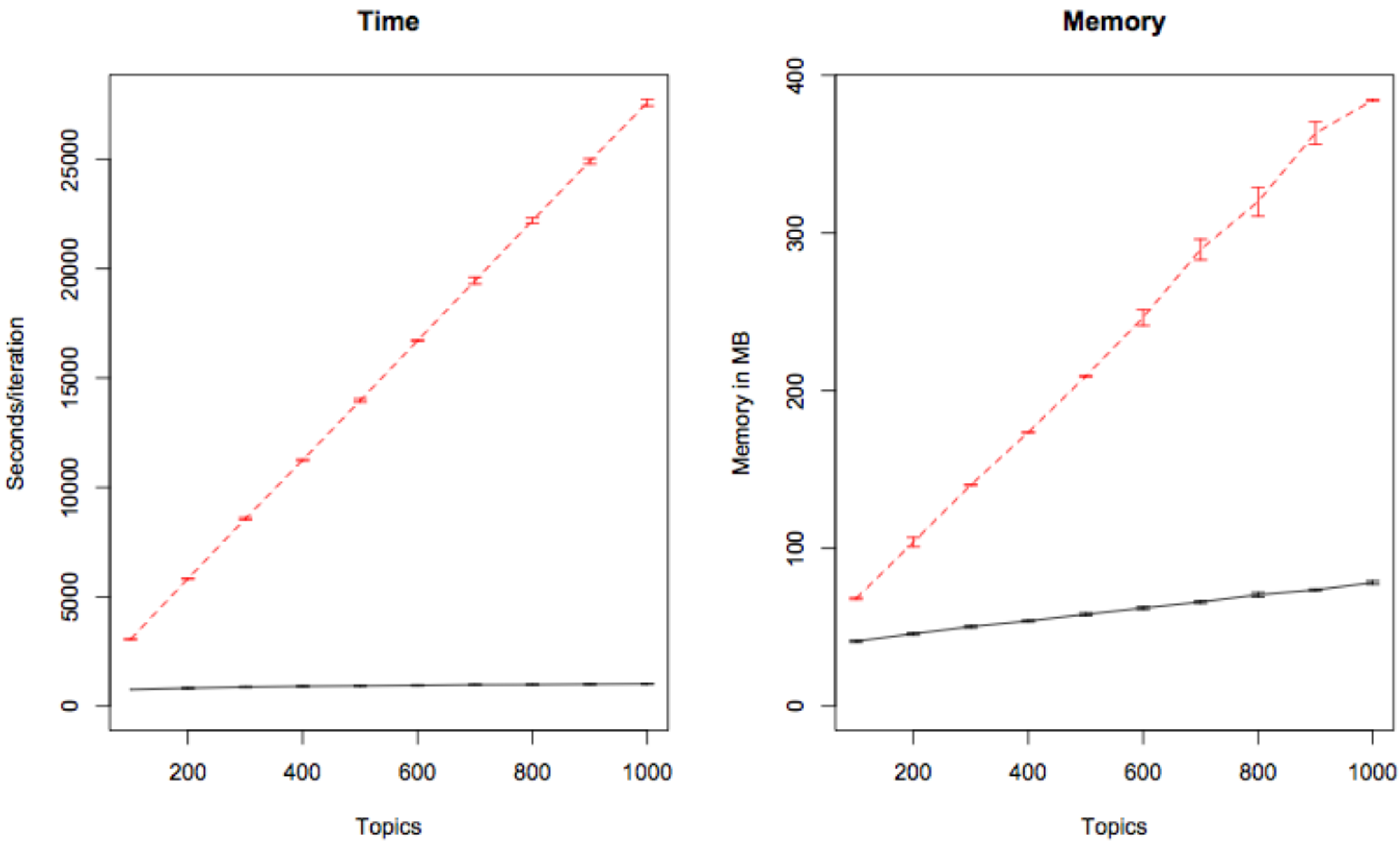
Standard allocation



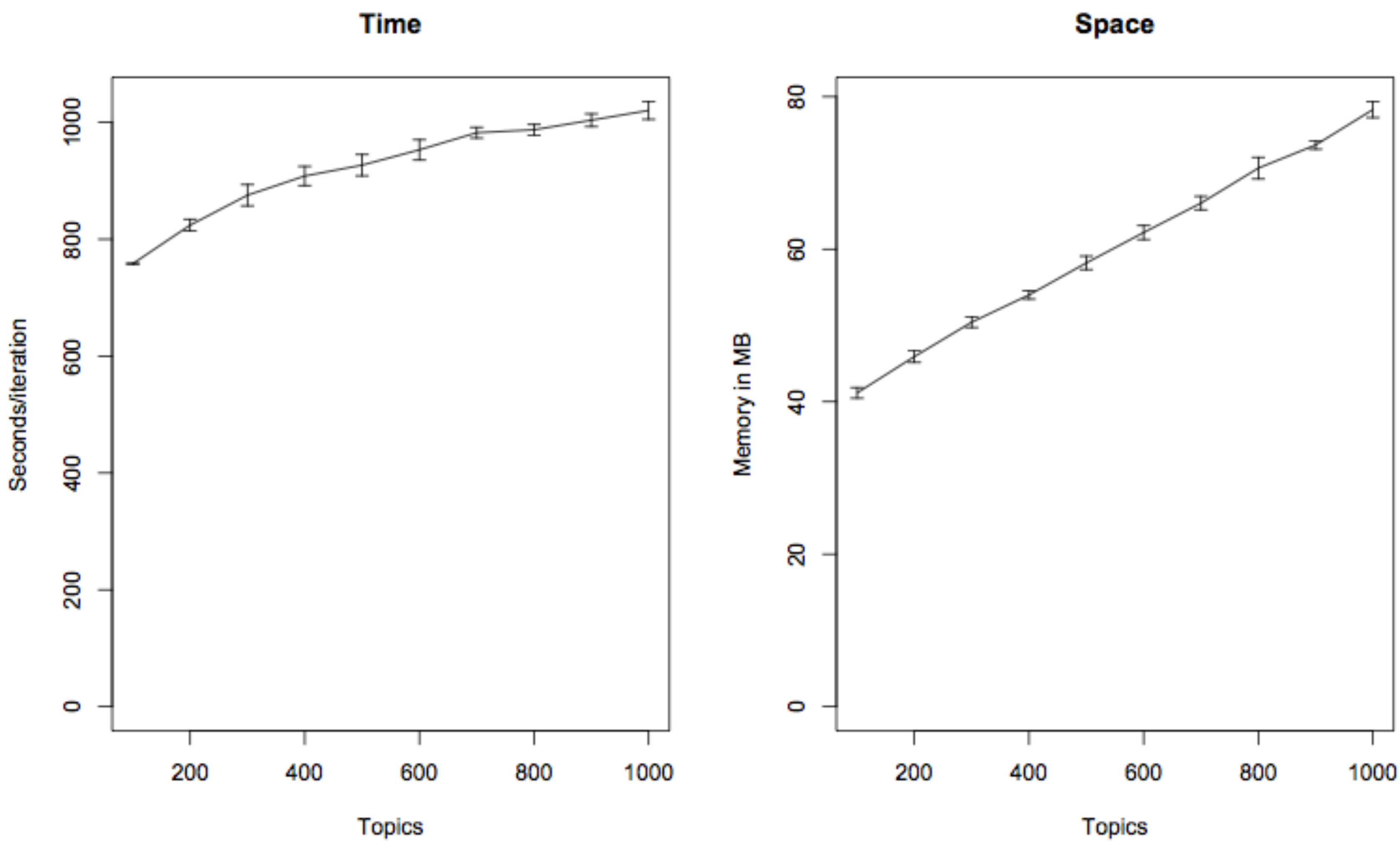
Conservative allocation



Huge savings: time and memory



Huge savings: time and memory



Large-scale Polylingual Topics

- Wikipedias in 40 languages (AR to ZH)

  (Articles linking to same English page share P(t|d))
- 1,200,000 *English* headwords
- 7,000,000 articles
- 300,000,000 tokens
- 4,000,000 distinct word types
  - Drop rare words (< 5 tokens)
  - Drop common words (> 5% of docs)
- 500 topics

One (1) CPU

Thanks!

- Paper:

  Efficient Methods for Topic Model Inference on Streaming Document Collections. Limin Yao, David Mimno, Andrew McCallum. KDD 2009. (See section 3.4)
- Code:

  http://mallet.cs.umass.edu

  (Implemented in cc.mallet.topics.WorkerRunnable)
- 40 language Wikipedia model:

  http://christo.cs.umass.edu/wiki40