Placing Skips Optimally in Expectation

Flavio Chierichetti,
Silvio Lattanzi,
Federico Mari
Alessandro Panconesi

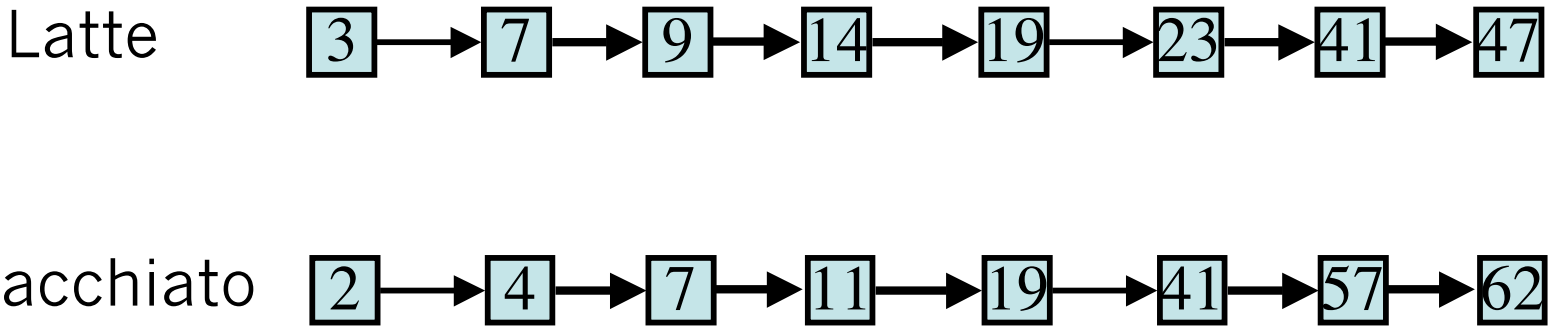
SAPIENZA
UNIVERSITÀ DI ROMA

Supported by YAHOO! RESEARCH

Problem Statement

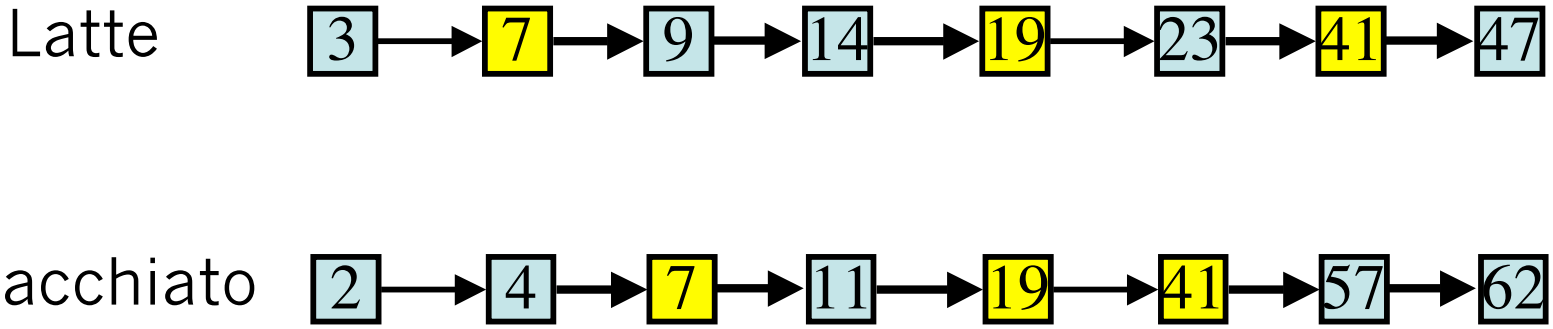
Answering conjunctive queries

query: latte macchiato



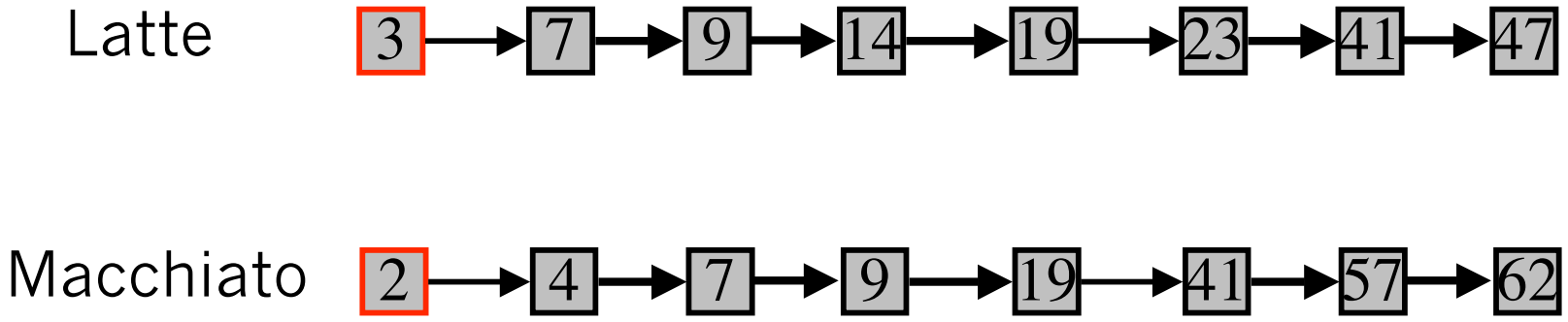
Answering conjunctive queries

query: latte macchiato

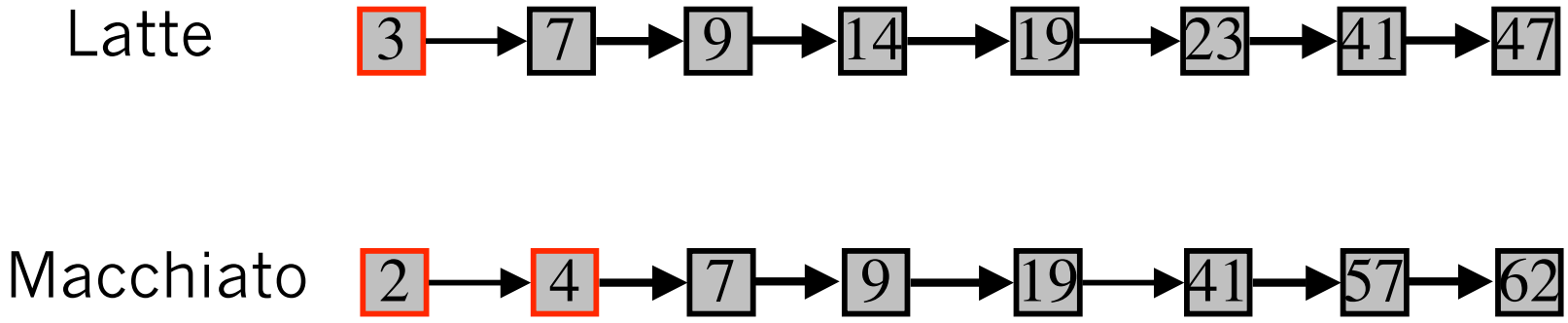


Compute the intersection of 2 sorted lists

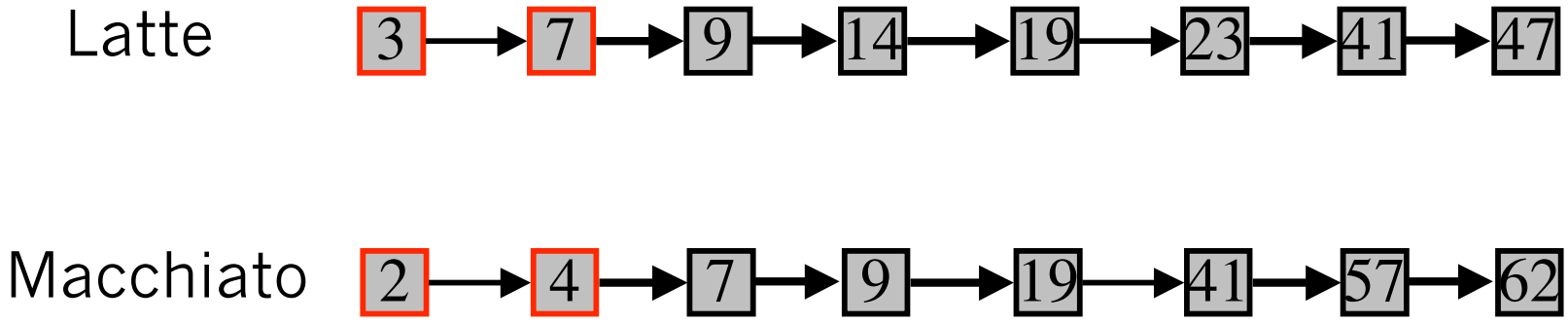
Merging



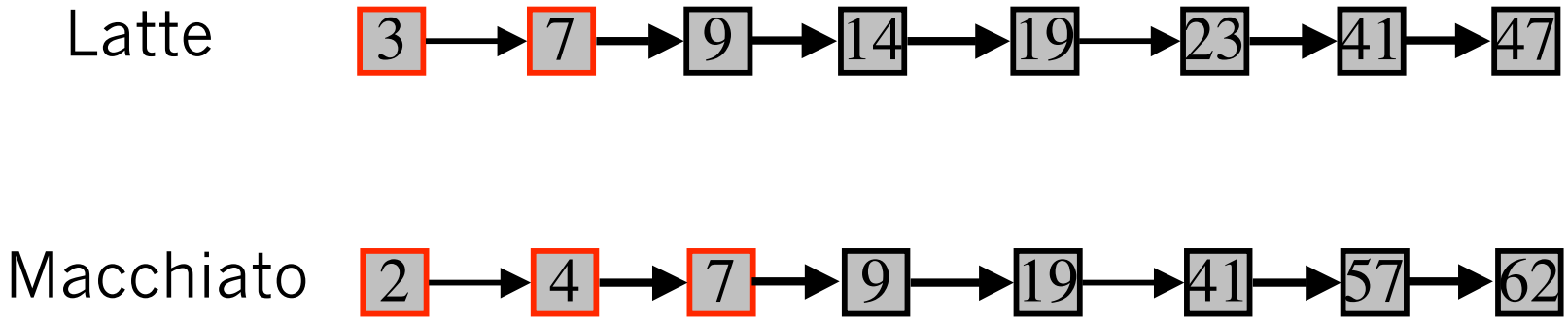
Merging



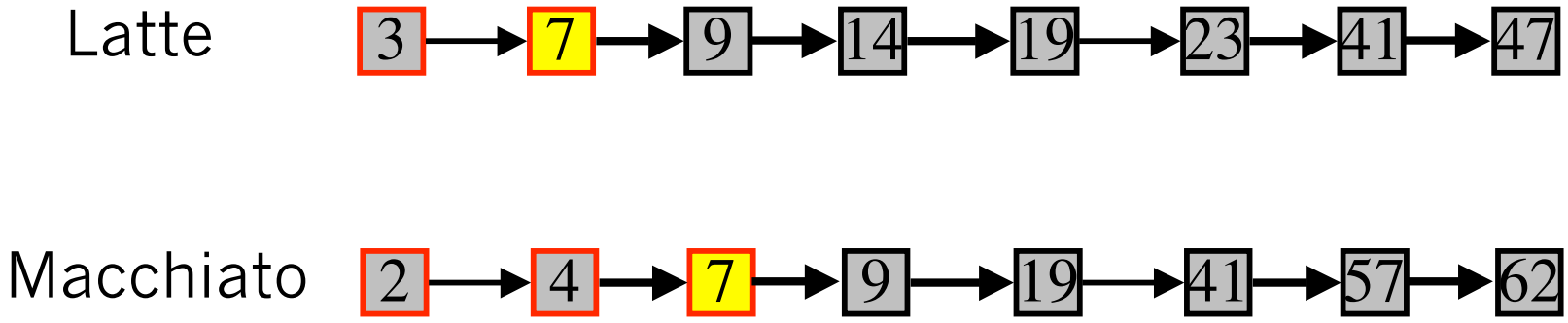
Merging



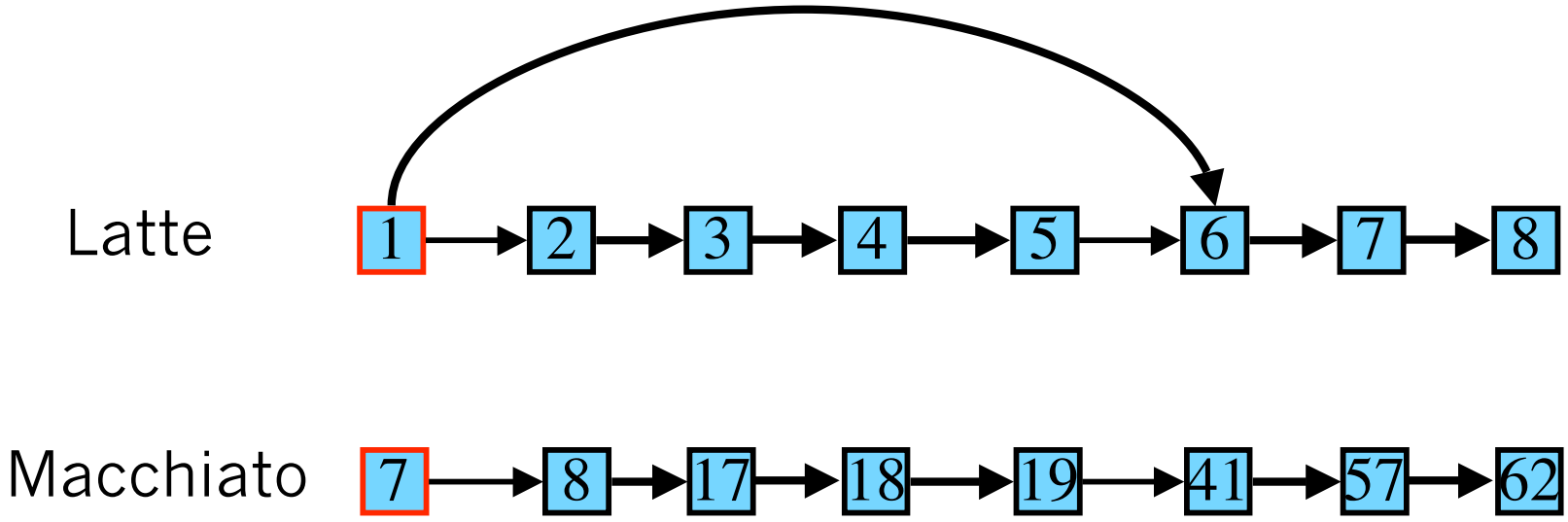
Merging



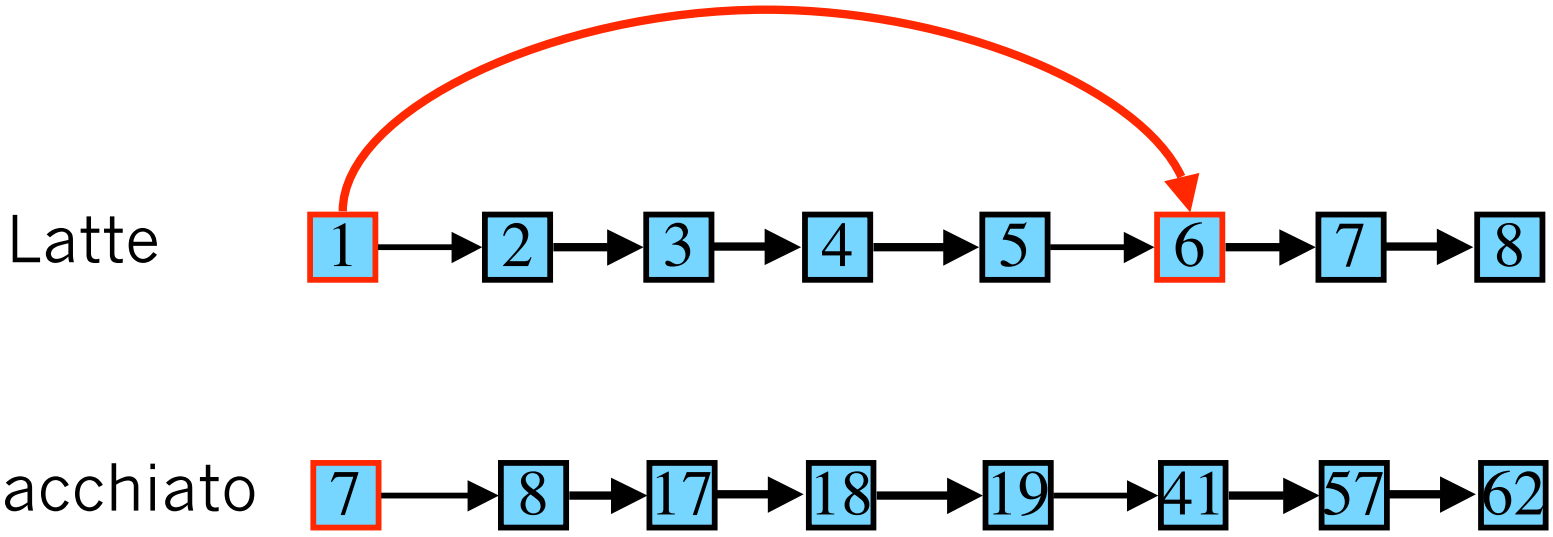
Merging



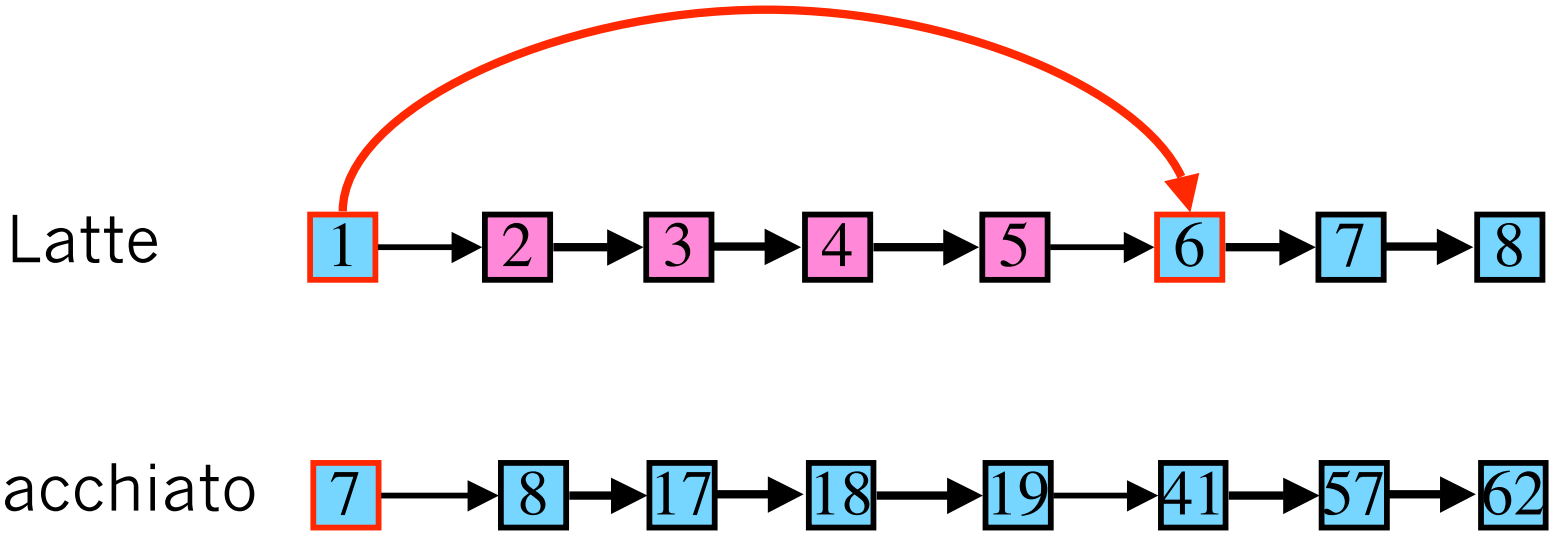
Skips



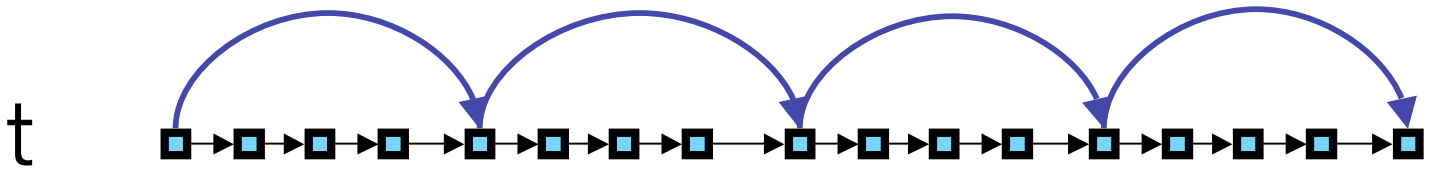
Skips



Skips



Conventional WSDM



Skips are placed every √N many positions

Question

If we know the query distribution, can we place skips better?

Problem statement

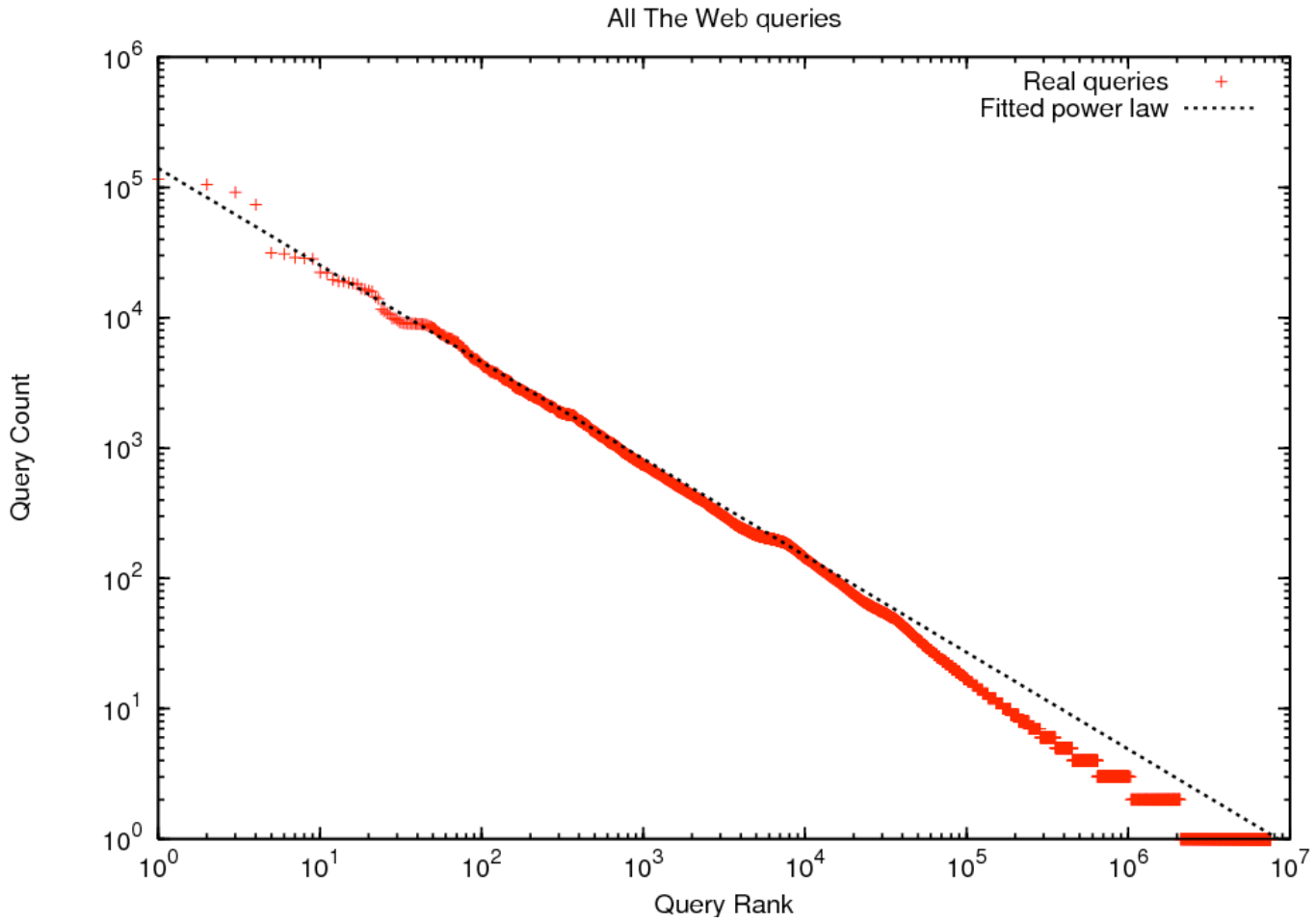
If we know the query distribution, can we place skips in order to minimize the expected time of a merge?

Problem statement

If we know the query distribution, can we place skips in order to minimize the expected time of a merge?

Is the assumption realistic?

The Power of the Law



The query distribution contains a lot of information.
Can we **provably** take advantage of it?

Algorithms to follow work with any query distribution whatsoever

Algorithms to follow work with any query distribution whatsoever

..and can be extended to deal with soft conjunctions

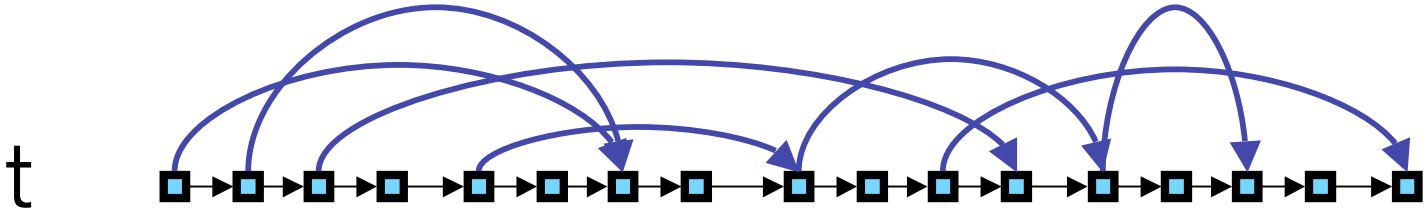
Outline

- Skip placement policies
- A matter of definitions
- Algorithms
- Experiments

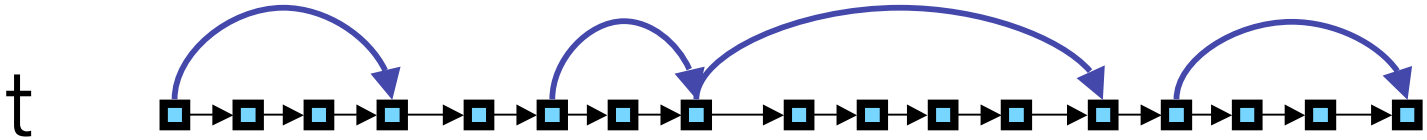
Skip Placement Policies

Spaghetti Skips

Spaghetti Skips

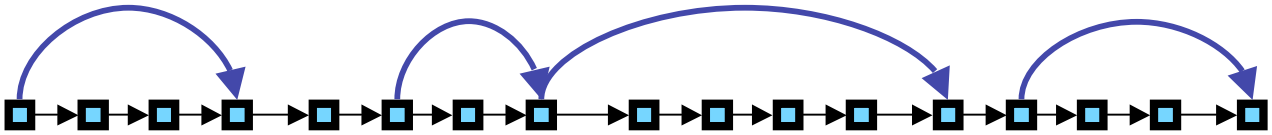


Simple Skips



Simple Skips

t

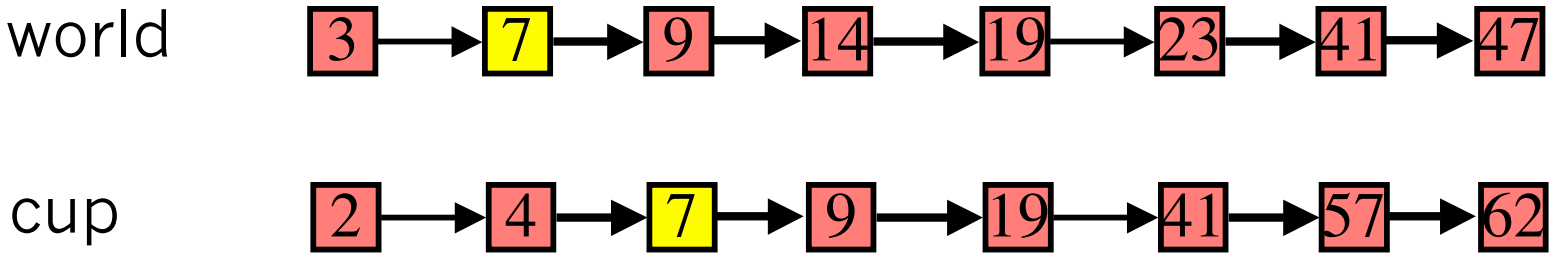


This is the most interesting case

A Matter of Definitions

Useful Documents: 1

q : world cup

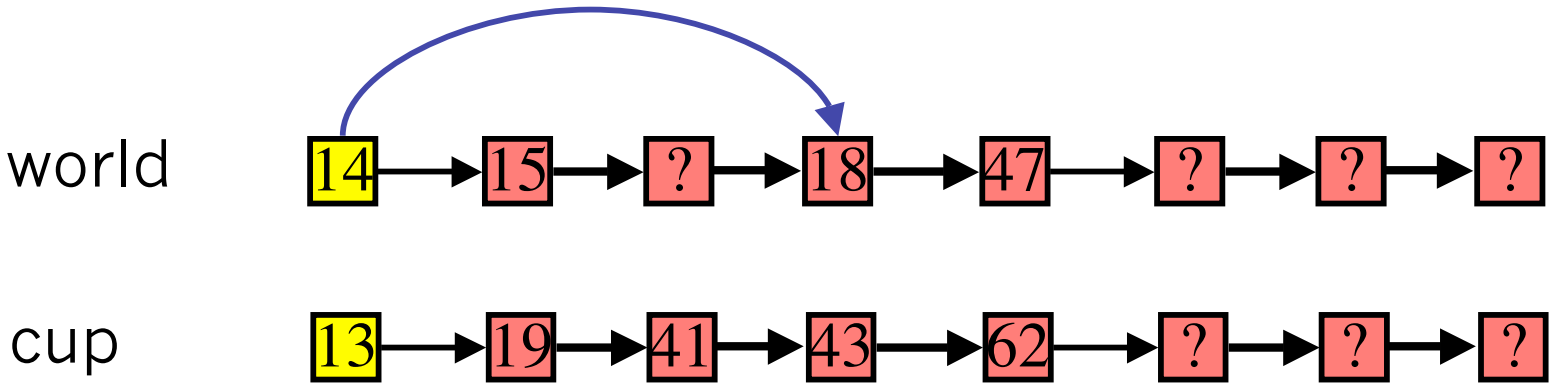


Relevant docs are “useful”

But usefulness does not coincide with relevance

Useful Documents: 2

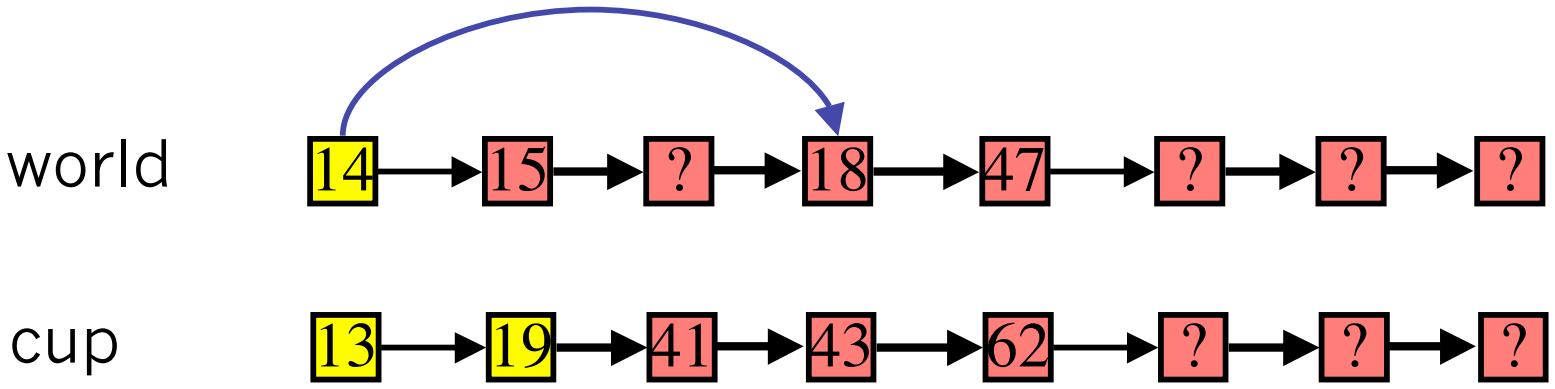
q : world cup



Is the skip useful for **q**?

Useful Documents: 2

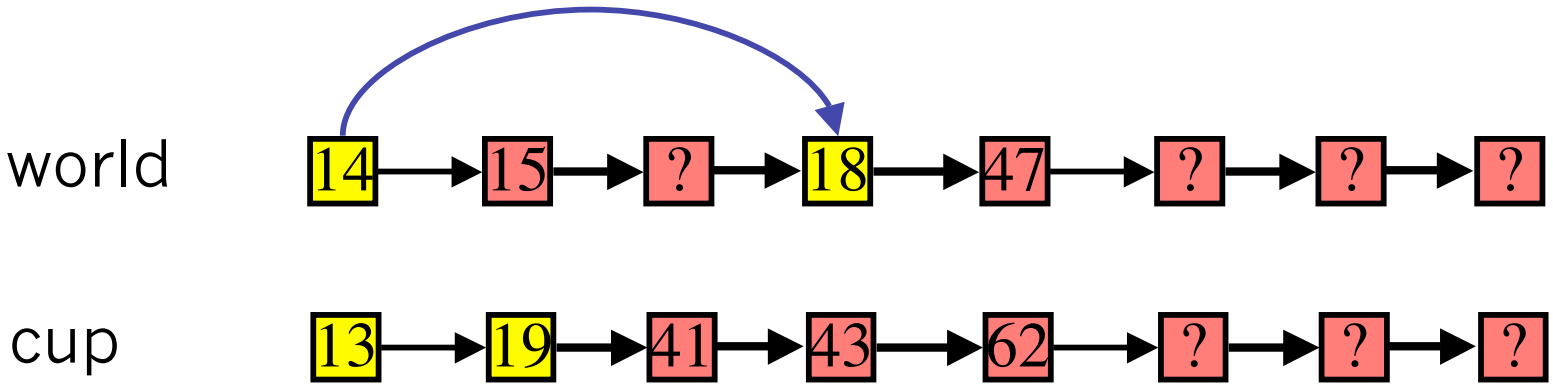
q : world cup



Is the skip useful for **q**?

Useful Documents: 2

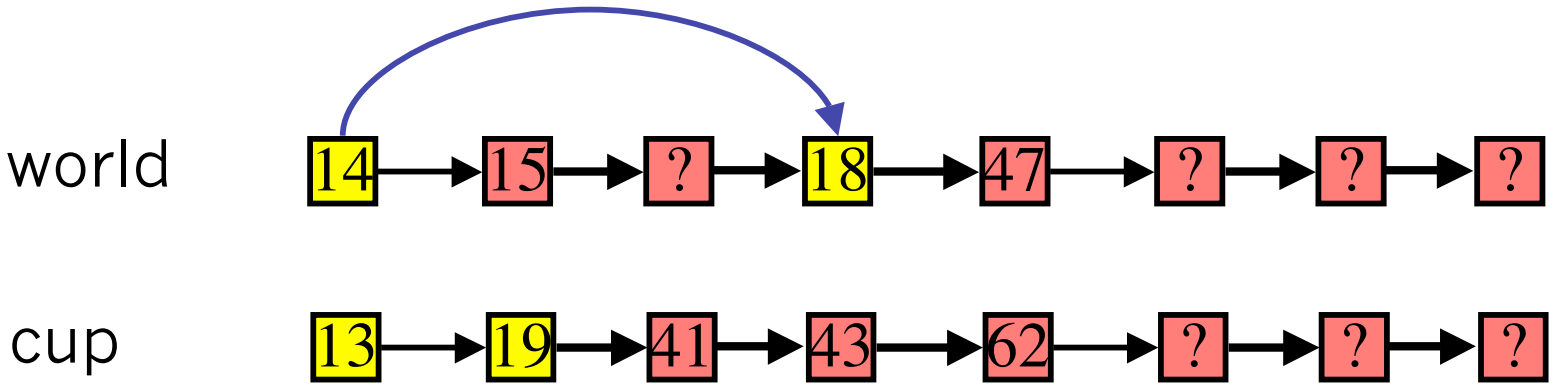
q : world cup



Is the skip useful for **q**?

Useful Documents: 2

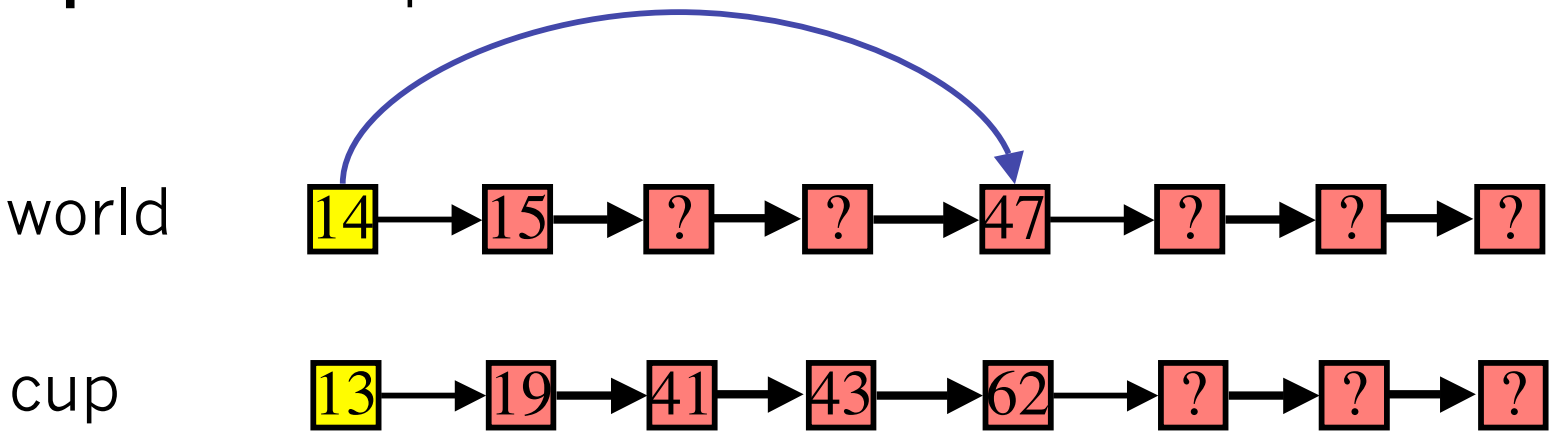
q : world cup



The skip is useful

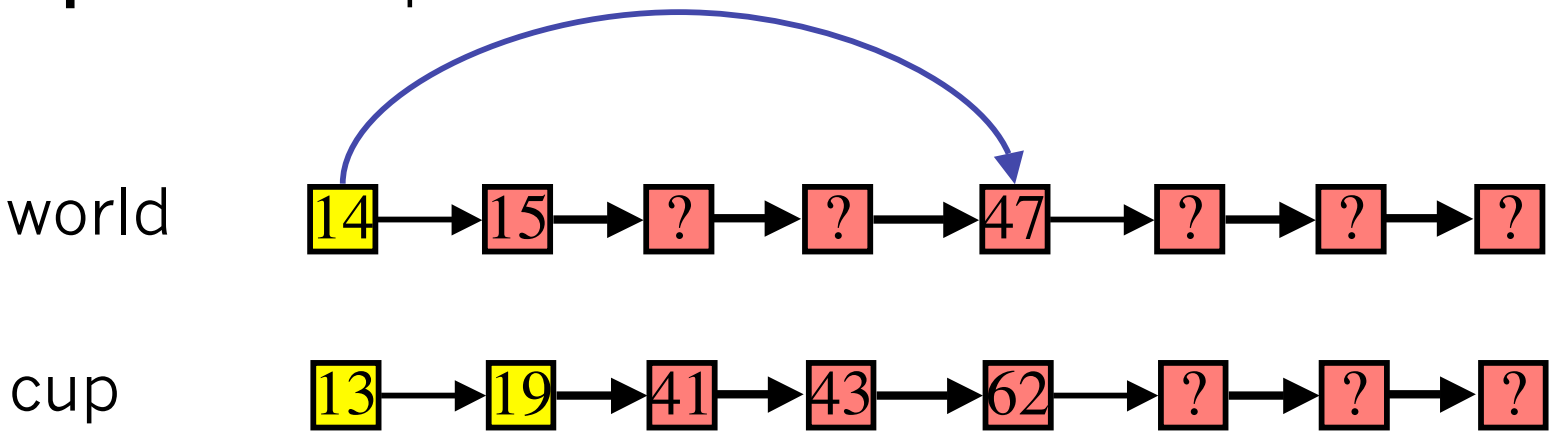
Useful Documents: 2

q : world cup



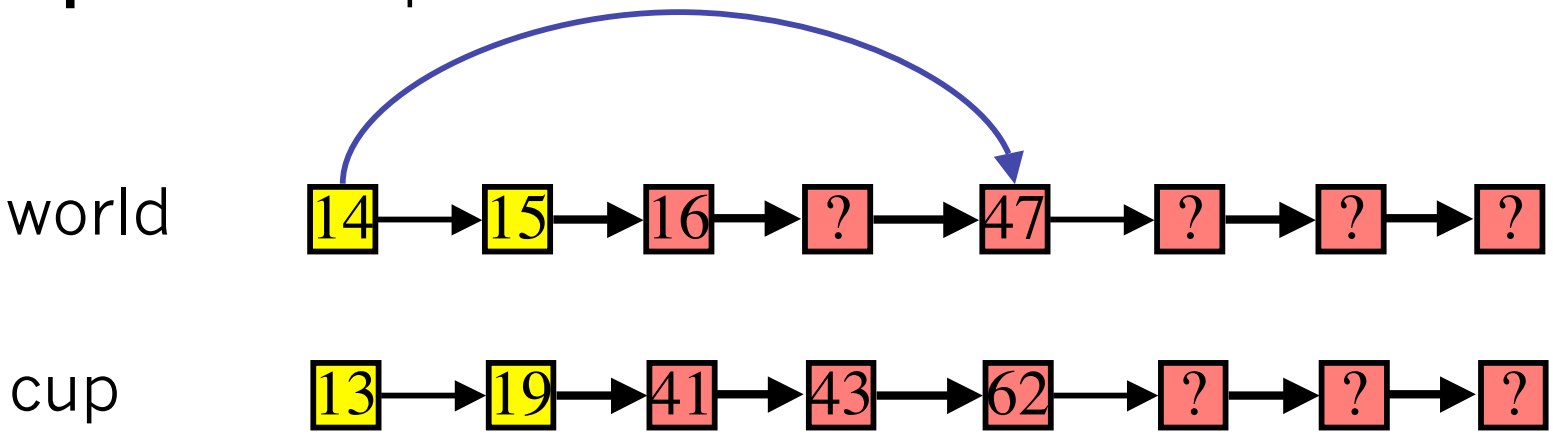
Useful Documents: 2

q : world cup



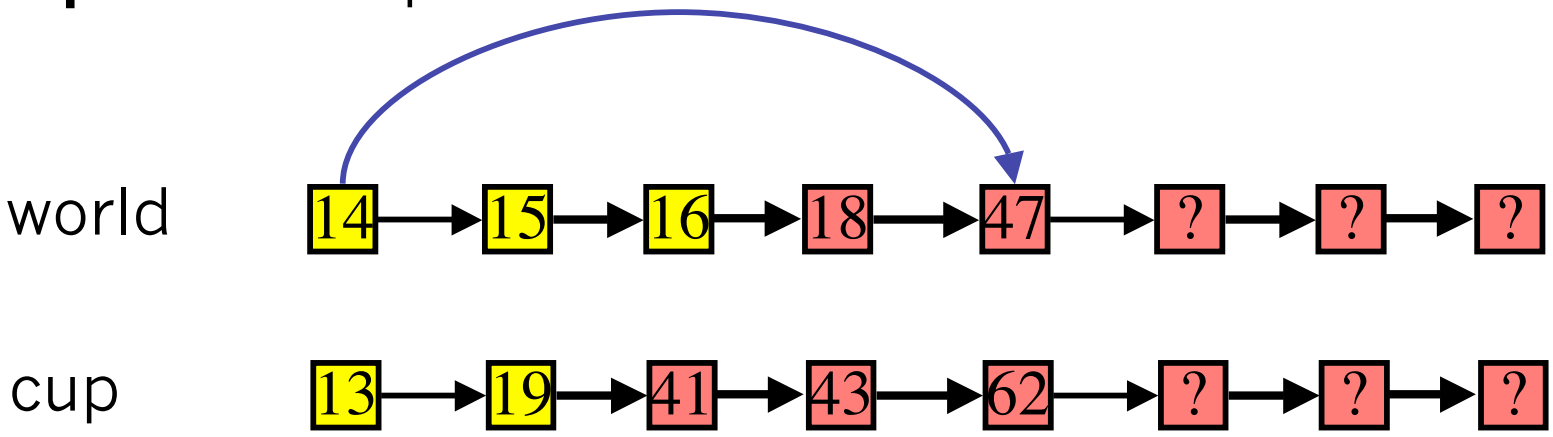
Useful Documents: 2

q : world cup



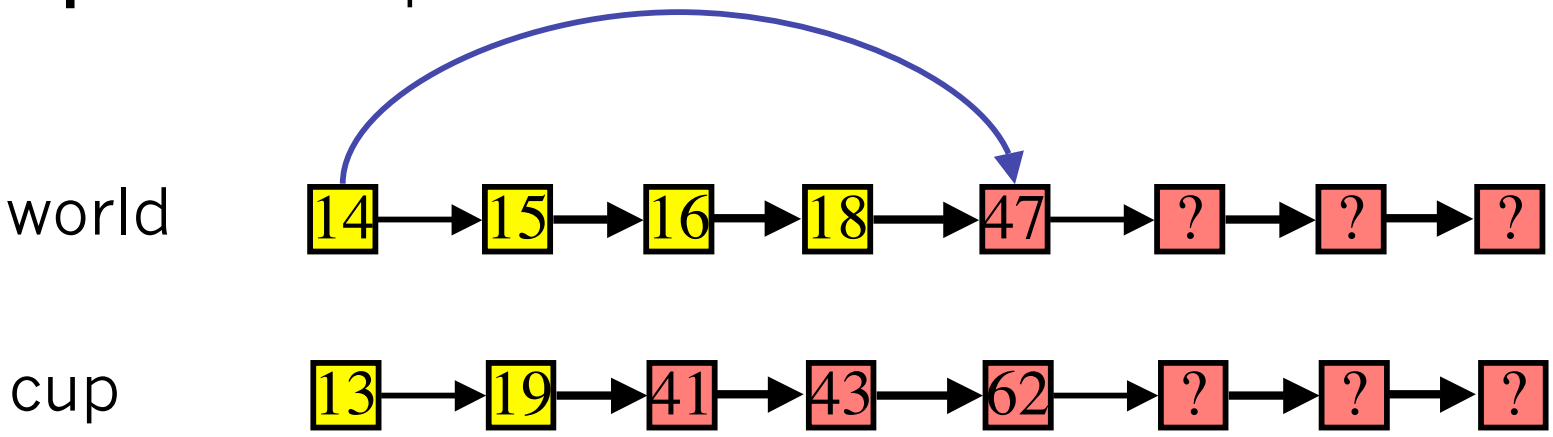
Useful Documents: 2

q : world cup



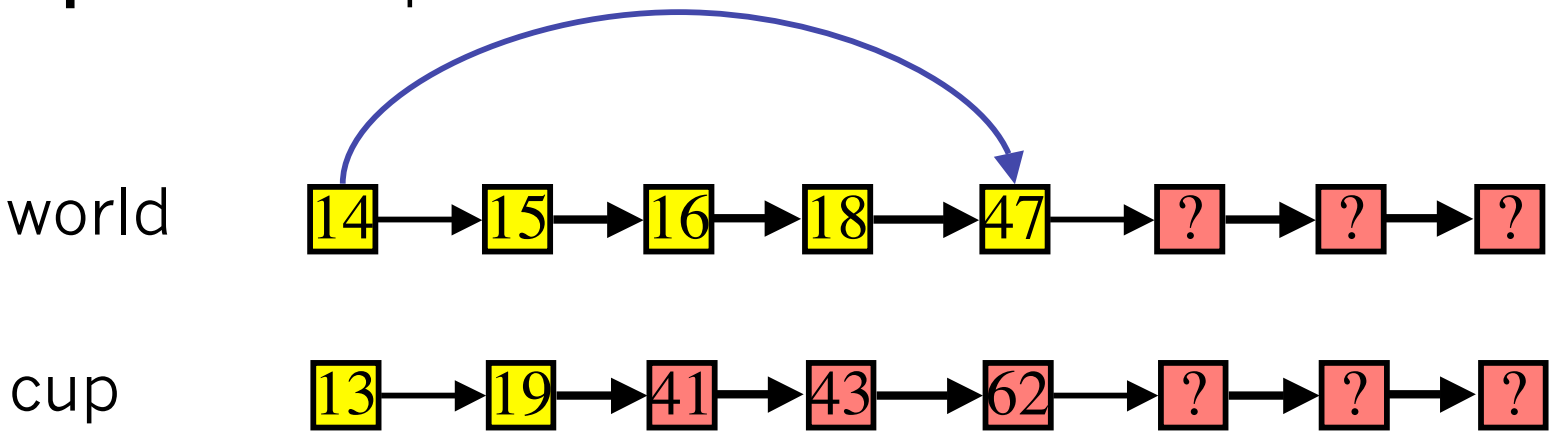
Useful Documents: 2

q : world cup



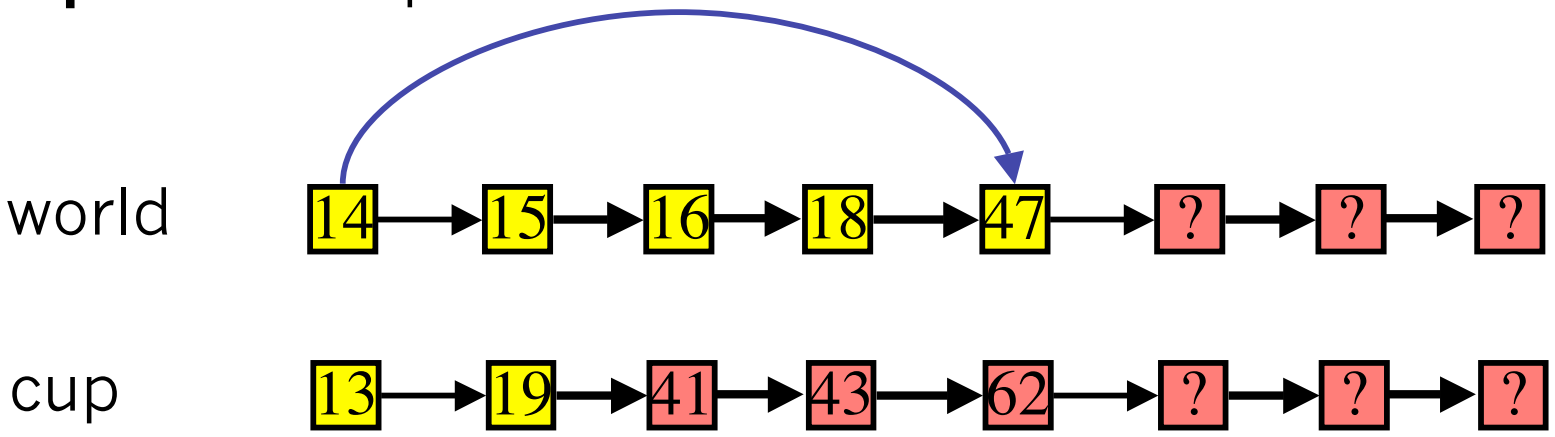
Useful Documents: 2

q : world cup



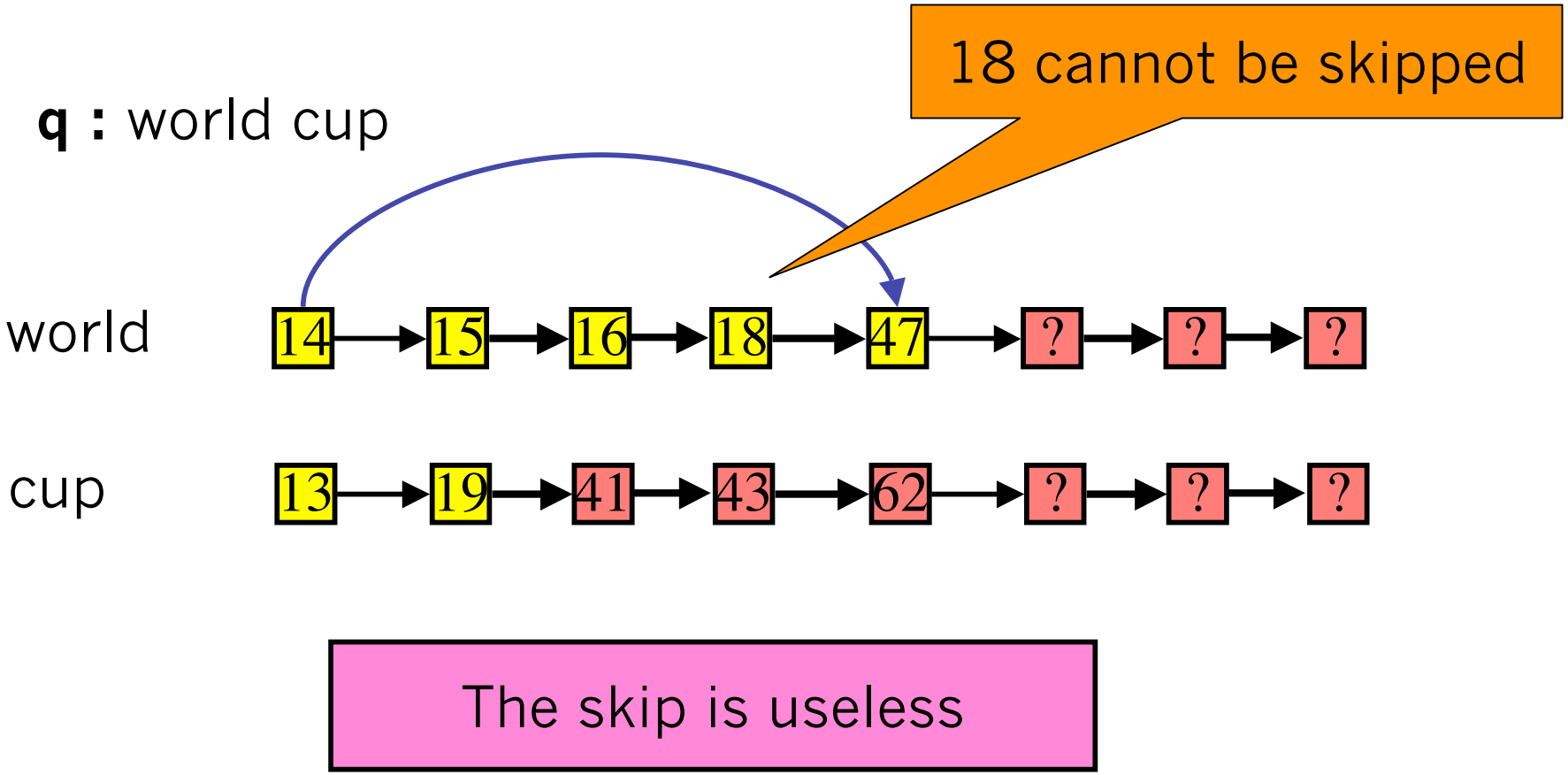
Useful Documents: 2

q : world cup



The skip is useless

Useful Documents: 2

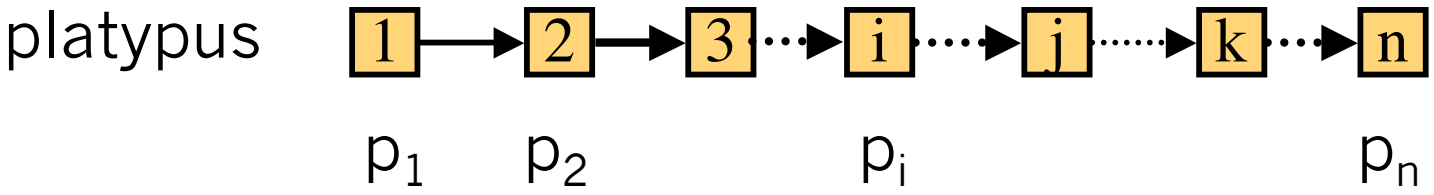


Useful Documents: 2

Useful documents are those that cannot
be avoided during a merge

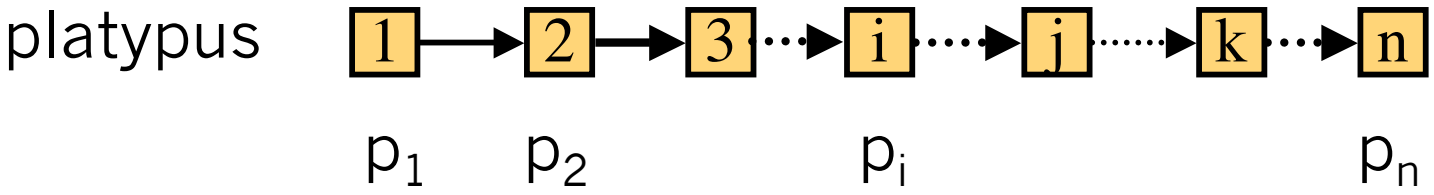
Induced Distributions

The query distribution induces another distribution on the postings



Induced Distributions

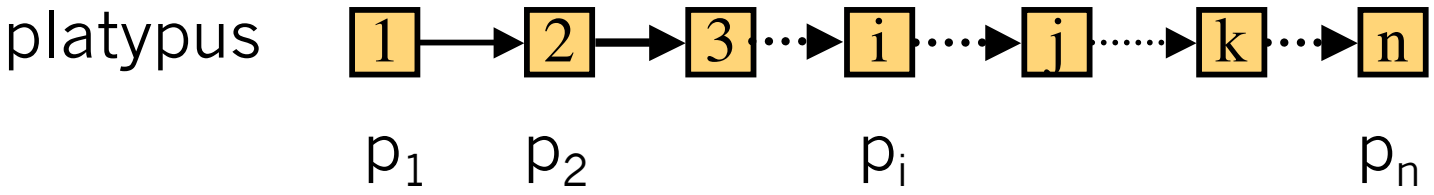
The query distribution induces another distribution on the postings



p_i = Pr(i useful for q | platypus ∈ q)

Induced Distributions

The query distribution induces another distribution on the postings



We will assume this distribution to be known

Induced Distributions

In practice these probabilities can be approximated using a small sample of the query universe

Making Life Simple

Events like “a is useful” and “b is useful” are not independent

..but from now on we will assume that they are

Making Life Simple

Events like “a is useful” and “b is useful” are not independent

..but from now on we will assume that they are

This simplifying assumption will be vindicated by our experiments

Algorithms

Algorithms

Input: a list with, for each doc, the probability that it is useful

Output: skip placement that minimizes the expected time to merge

Algorithms

Input: a list with, for each doc, the probability that it is useful

Output: skip placement that minimizes the expected time to merge

cost of a merge = #elements read in posting list

Algorithms

- O(nt) algorithm for spaghetti skips, where t is the average length of a skip
- O(n log n) for simple skips

Algorithms

- O(nt) algorithm for spaghetti skips, where t is the average length of a skip
- O(n log n) for simple skips

O(n log n) algorithm for simple skips is by far the most interesting

Simple Skips

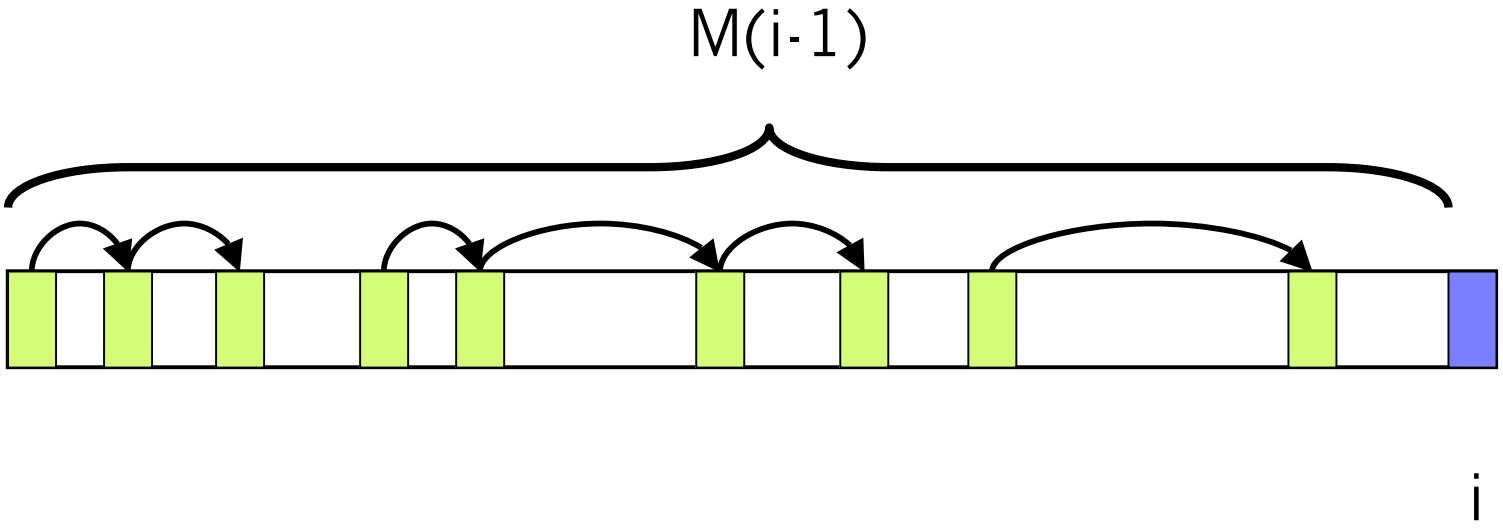
t: $d_1 d_2 .. d_i .. d_n$ & $p_1 p_2 .. p_n$

- Build the solution from left to right
- M(i) is best placement for prefix $d_1 .. d_i$

Computing M(i)

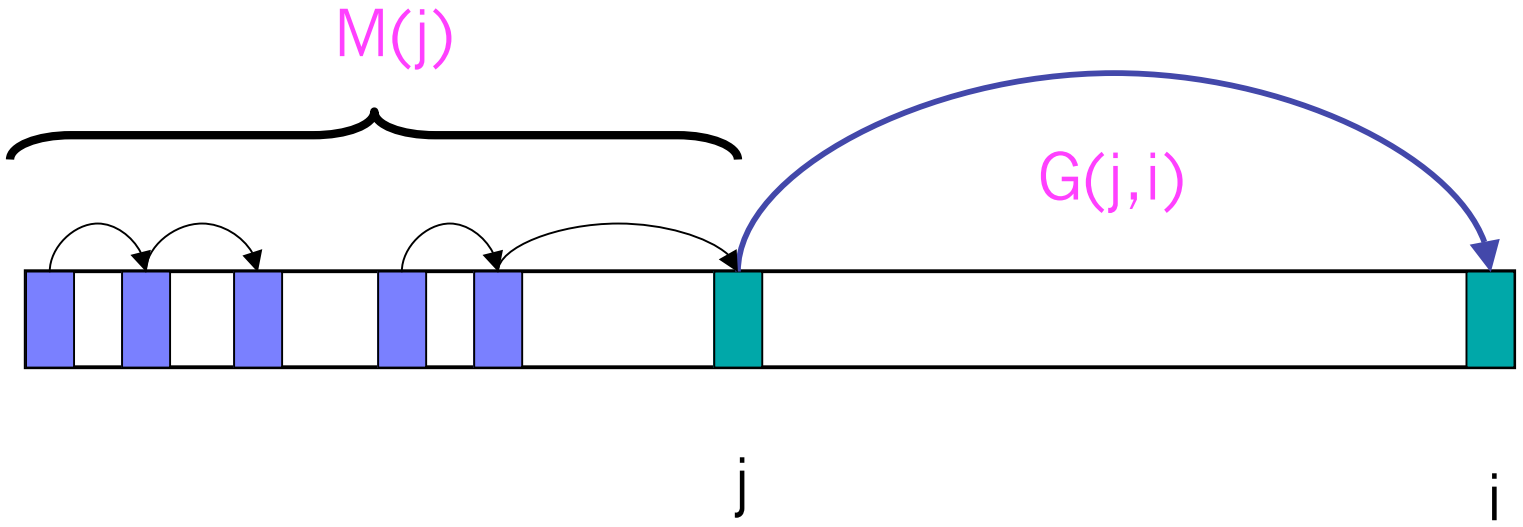
In computing M(i) we have two choices. We either place a skip landing at position i or we do not

Computing M(i)

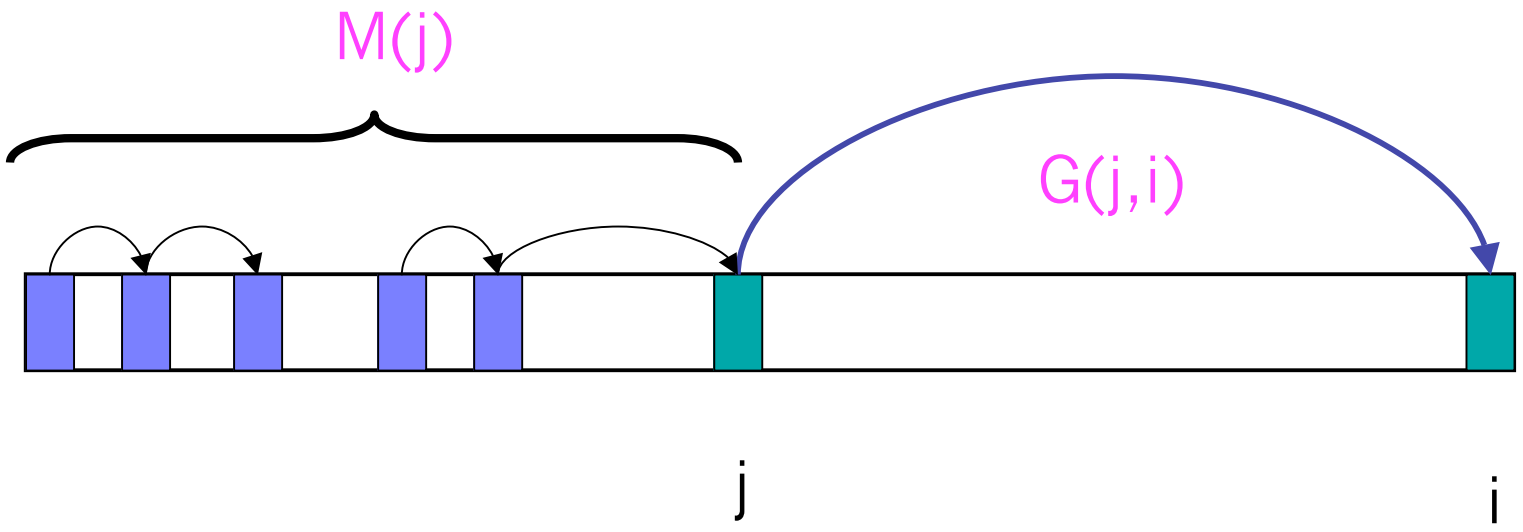


If we place no skip to i then M(i) = M(i-1)

Computing M(i)

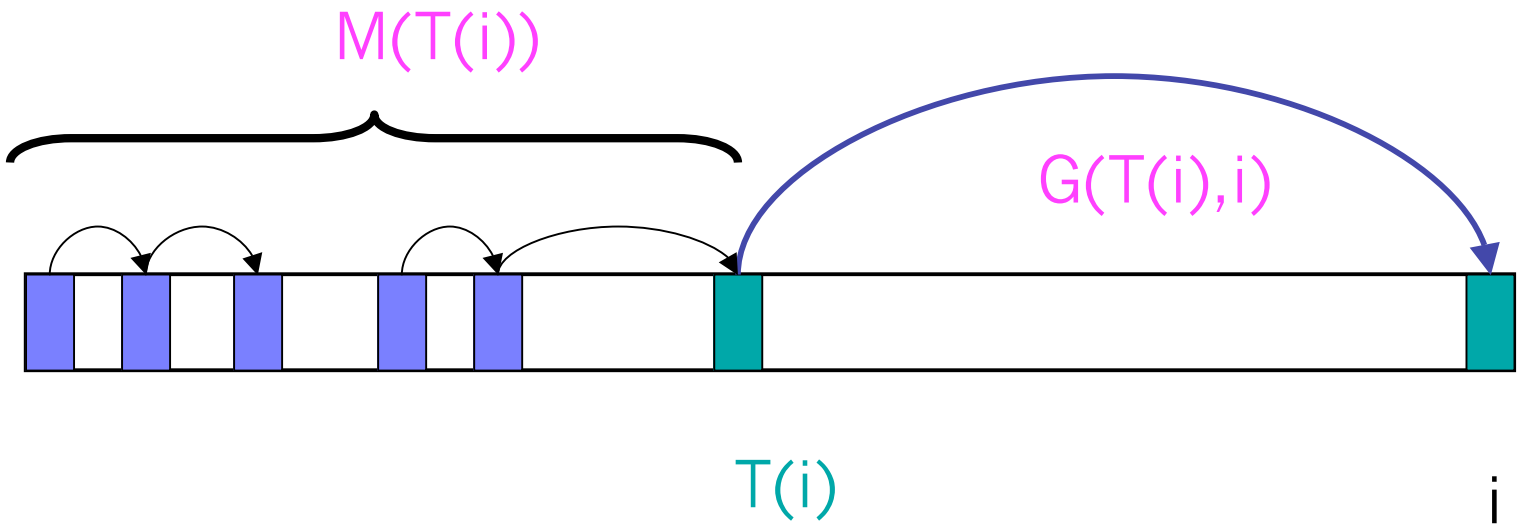


Computing M(i)



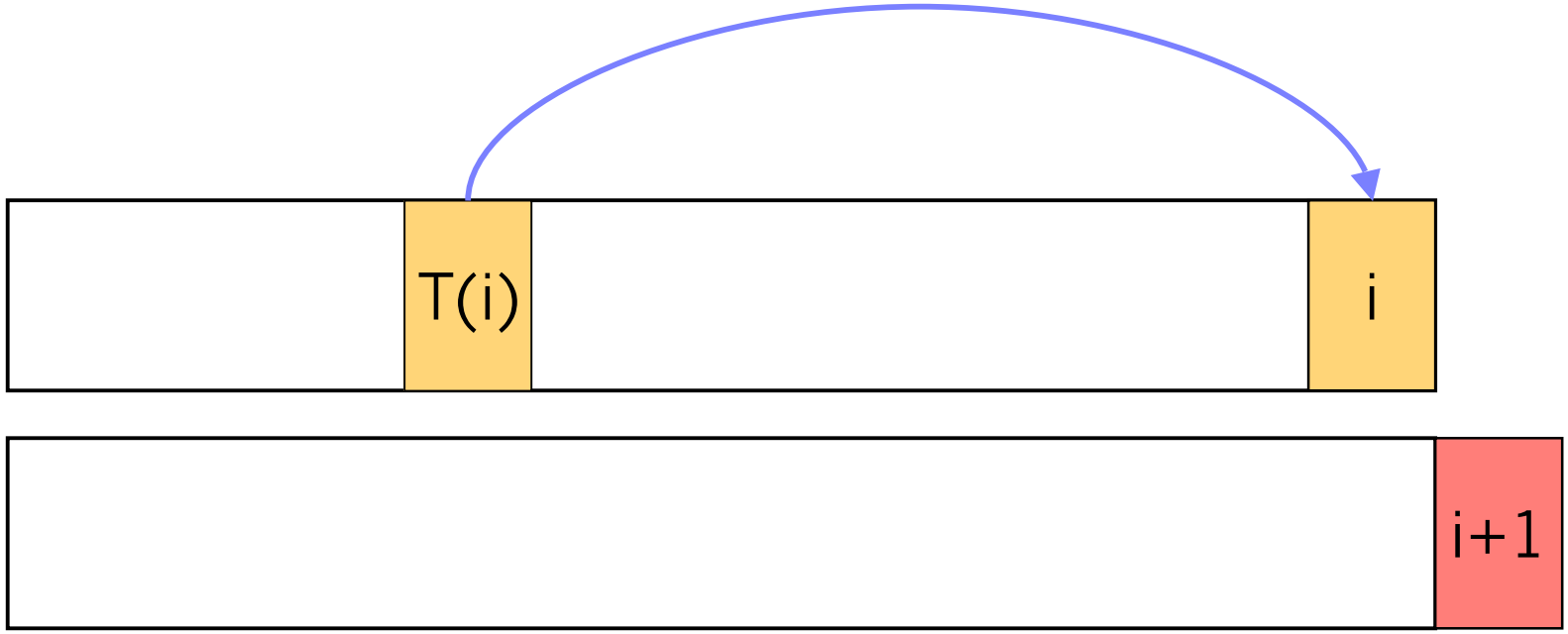
$$M(i) = \max \{ M(i\text{-}1), \max_j M(j) + G(j, i) \}$$

Computing M(i)

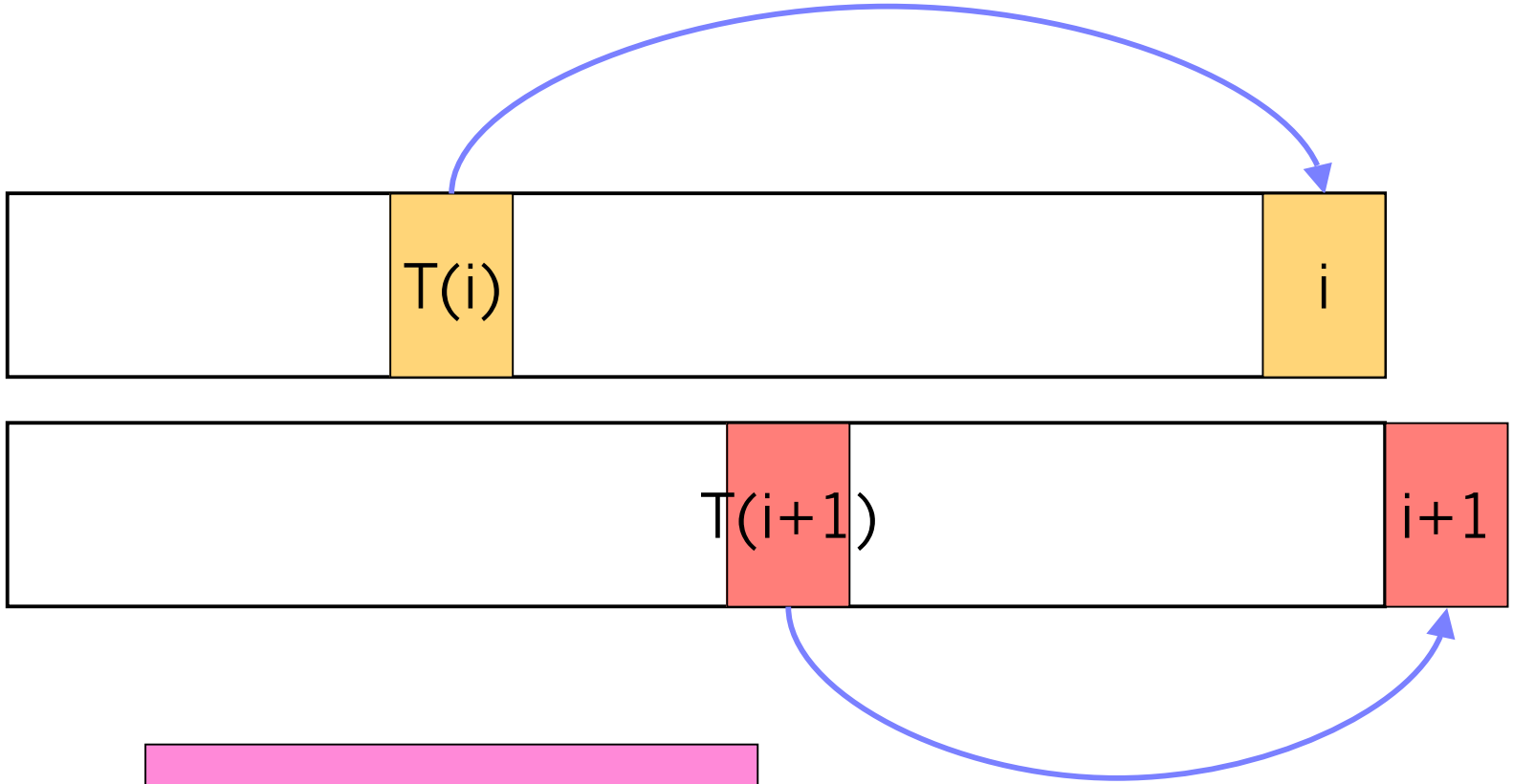


$$\max_j M(j) + G(j, i) = M(T(i)) + G(T(i),j)$$

Monotonicity of T(i)

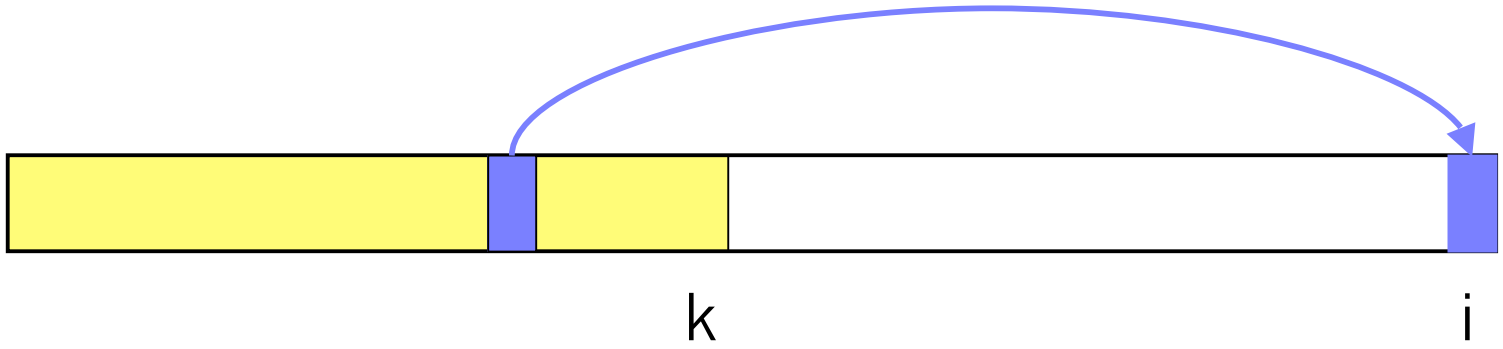


Monotonicity of T(i)



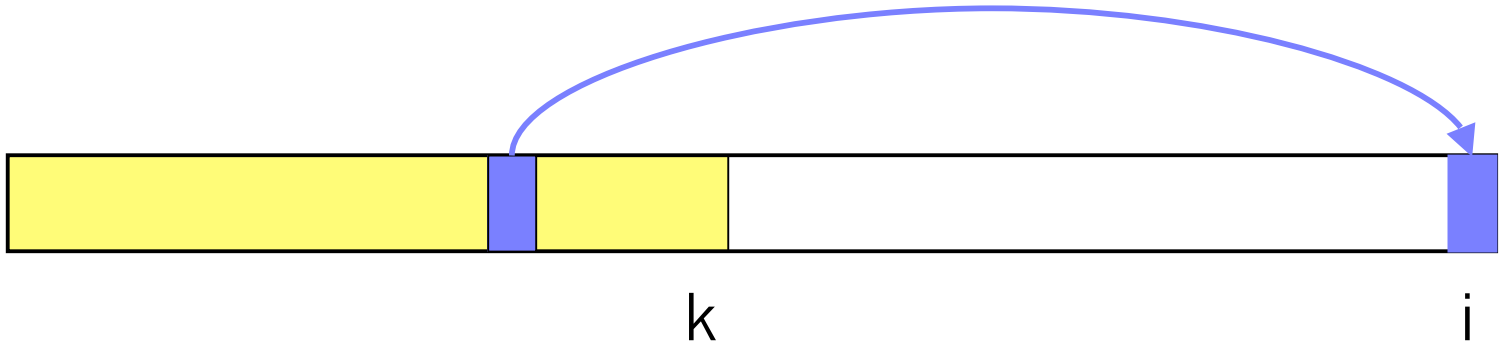
$T(i) \leq T(i+1)$

Monotonicity of T(i,k)



T(i,k) is best jump to i under the additional constraint that it must start no later than k

Monotonicity of T(i,k)



Key lemma: $T(i,k) \leq T(i+1,k)$

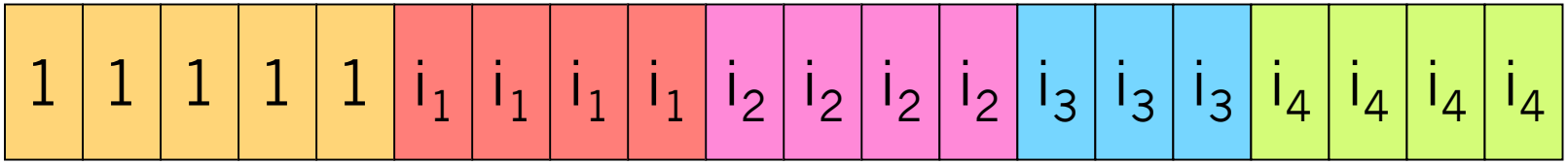
Monotonicity of T(i,k)

Let î be the smallest index i such that T(i,k)=k. Then,

$$T(j,k) = \begin{cases} k & j \geq \hat{\imath} \\ T(j,k-1) & j < \hat{\imath} \end{cases}$$

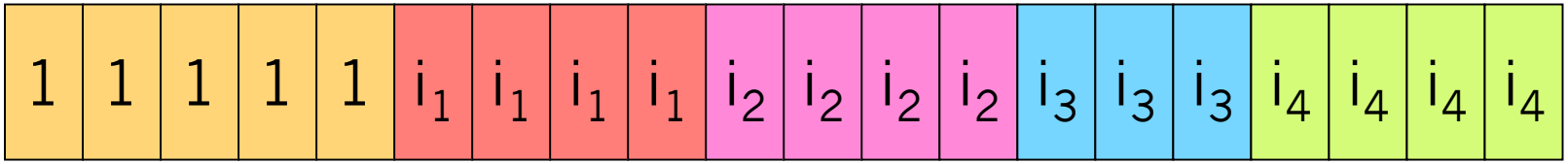
Updating T(i,k)

T(i,k-1)



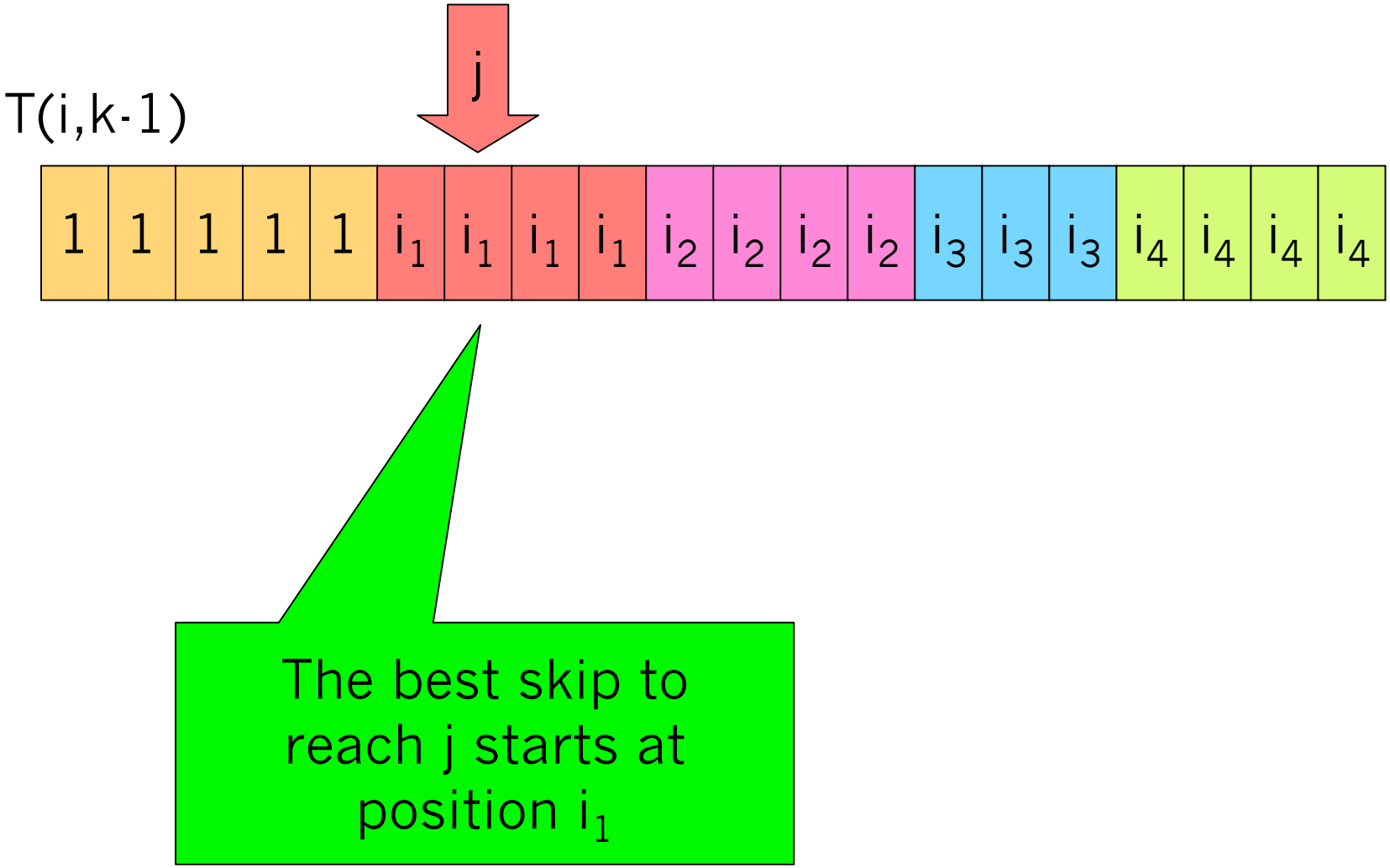
Updating T(i,k)

T(i,k-1)



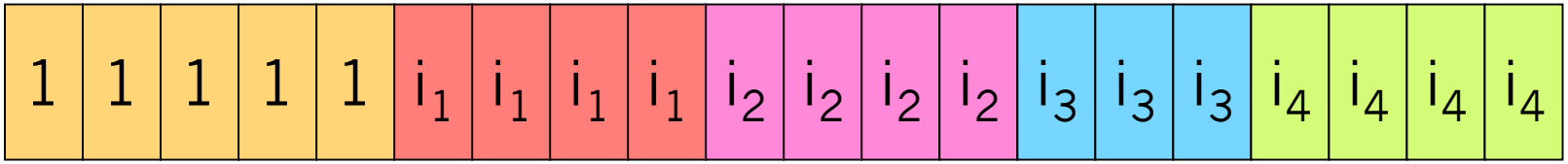
$$1 < i_1 < i_2 < i_3 < i_4 < k\text{-}1$$

Updating T(i,k)



Updating T(i,k)

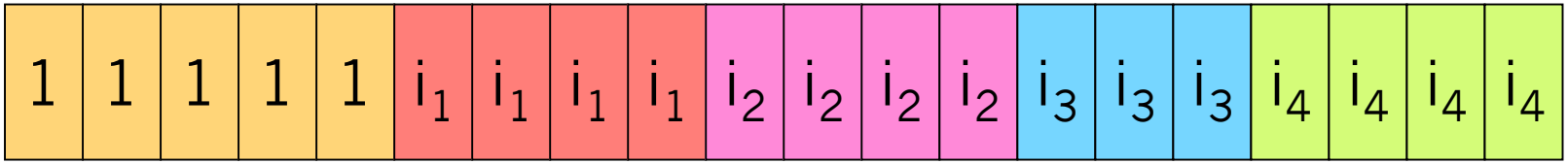
T(i,k-1)



T(i,n) gives the optimal placement

Updating T(i,k)

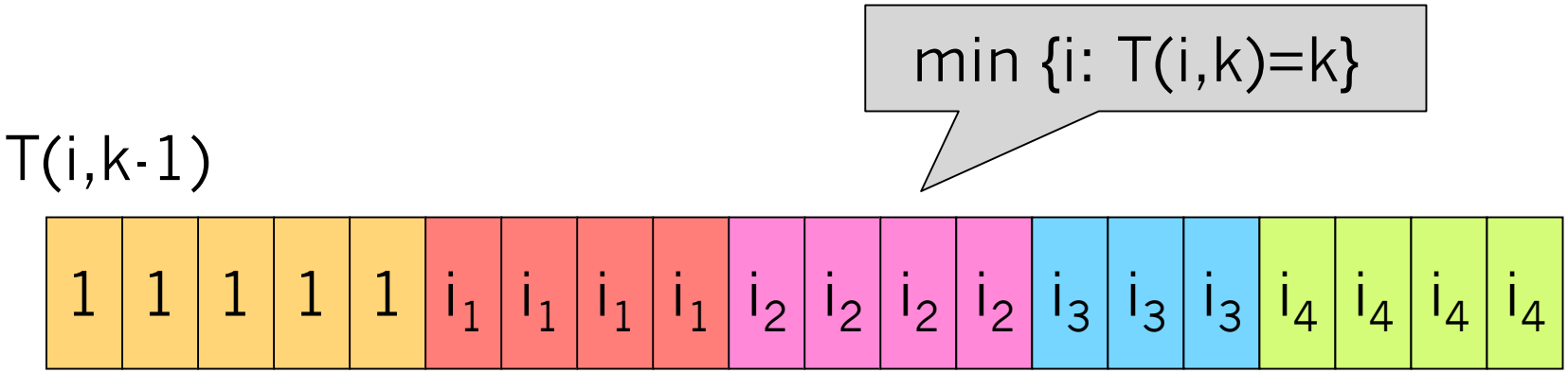
T(i,k-1)



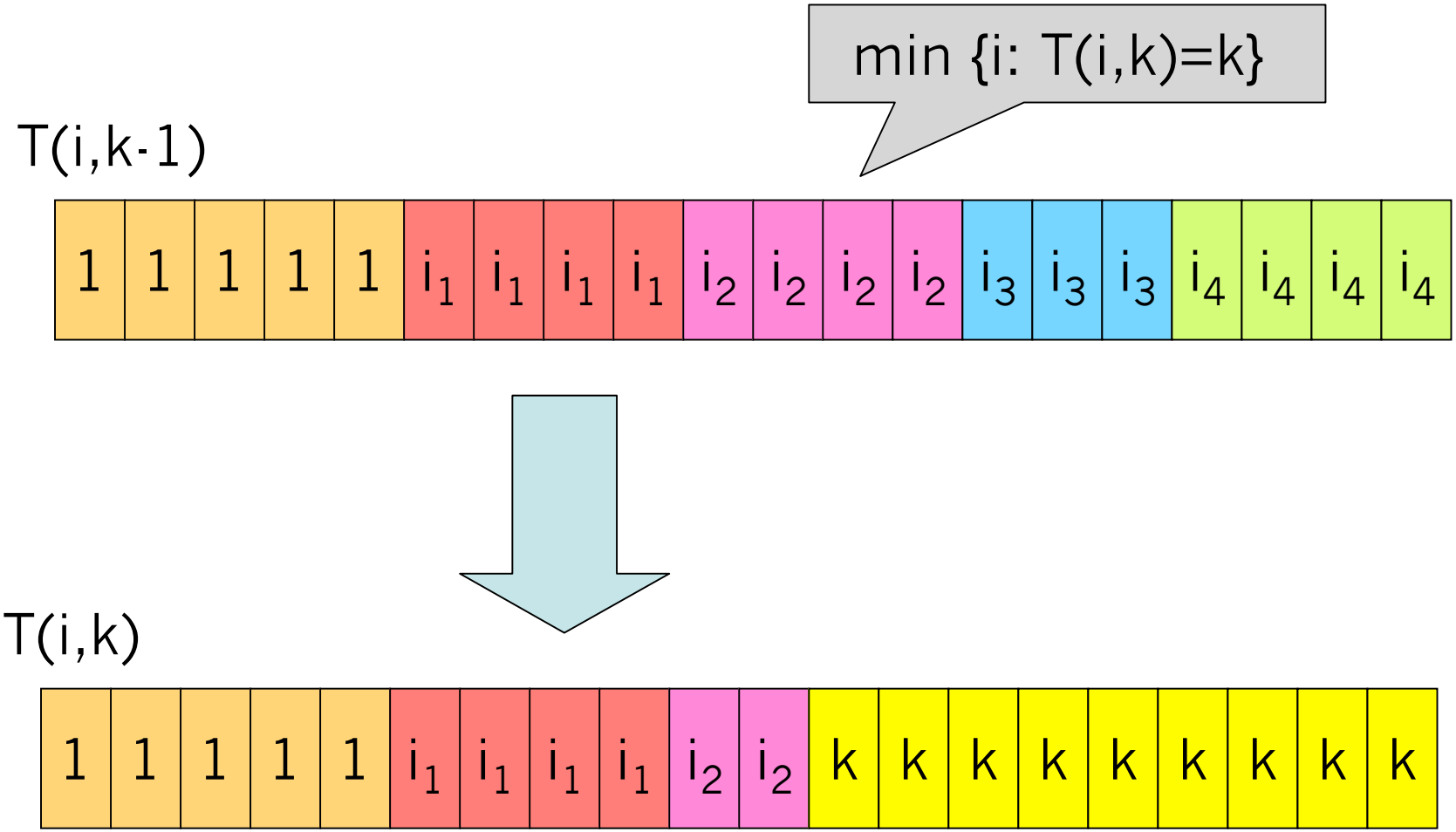
T(i,n) gives the optimal placement

T(•,1) → T(•,2) → ... → T(•,k) → ... → T(•,n)

Updating T(i,k)



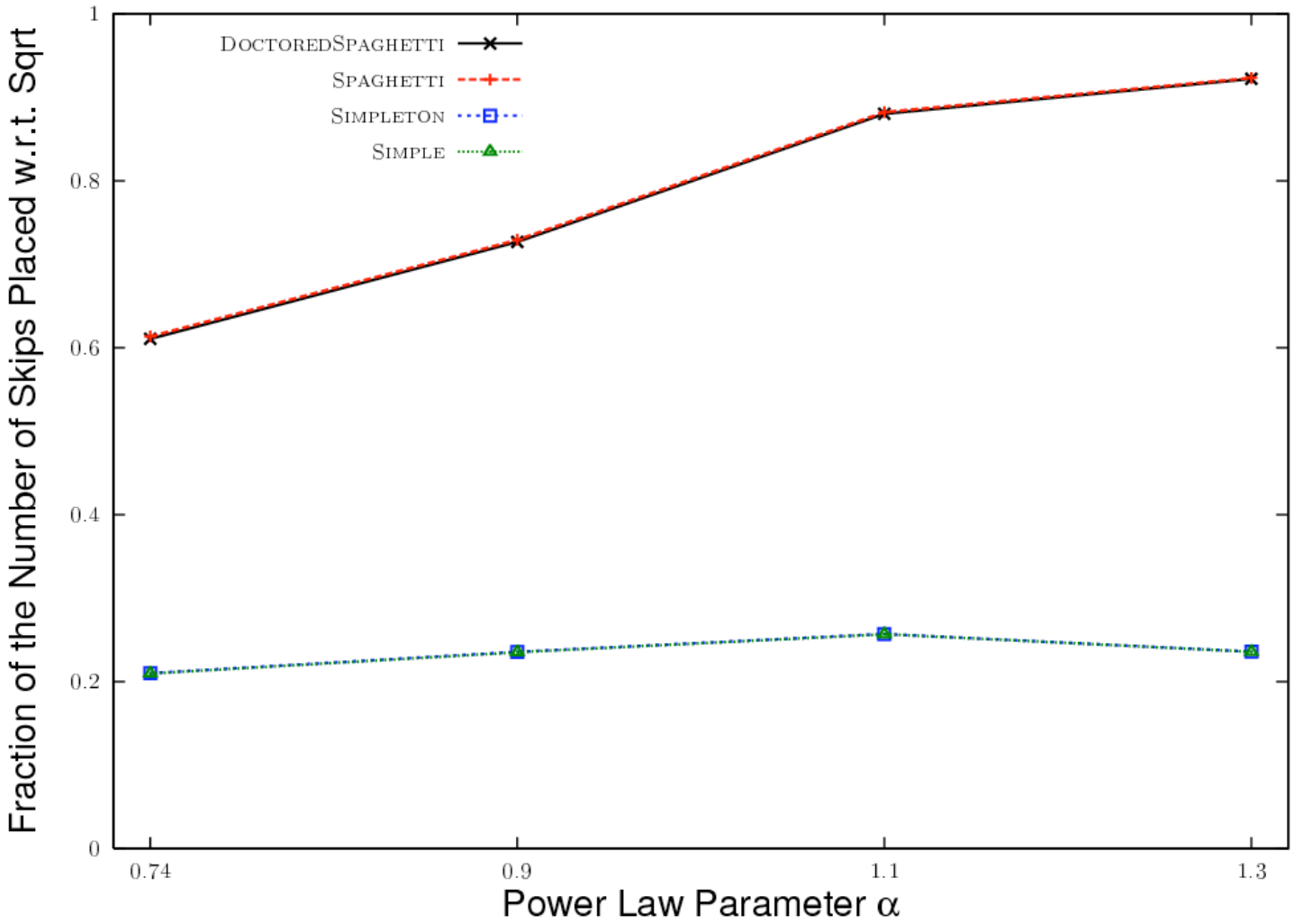
Updating T(i,k)



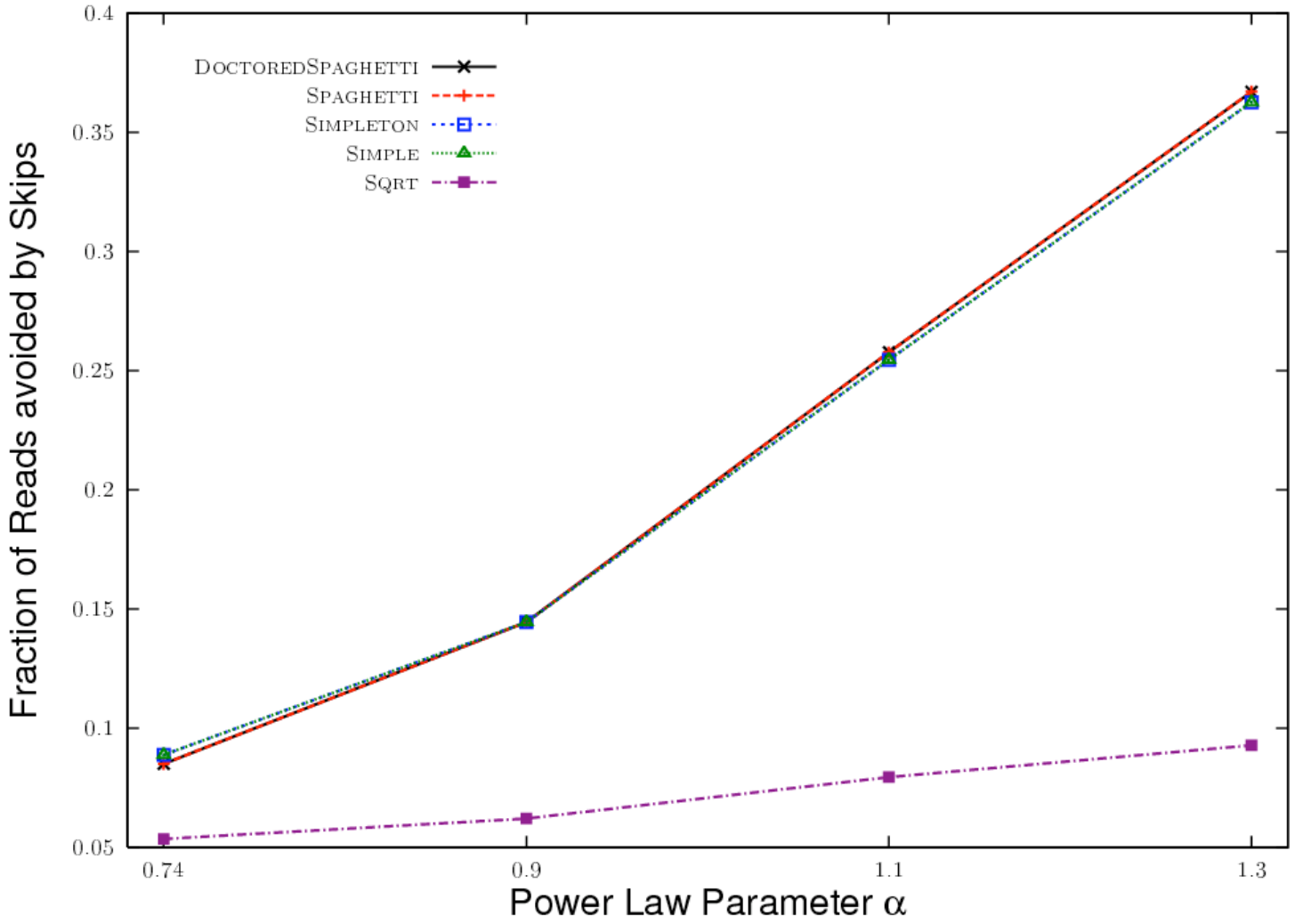
The resulting algorithm takes O(N logN)
where N is the length of the list

Experiments

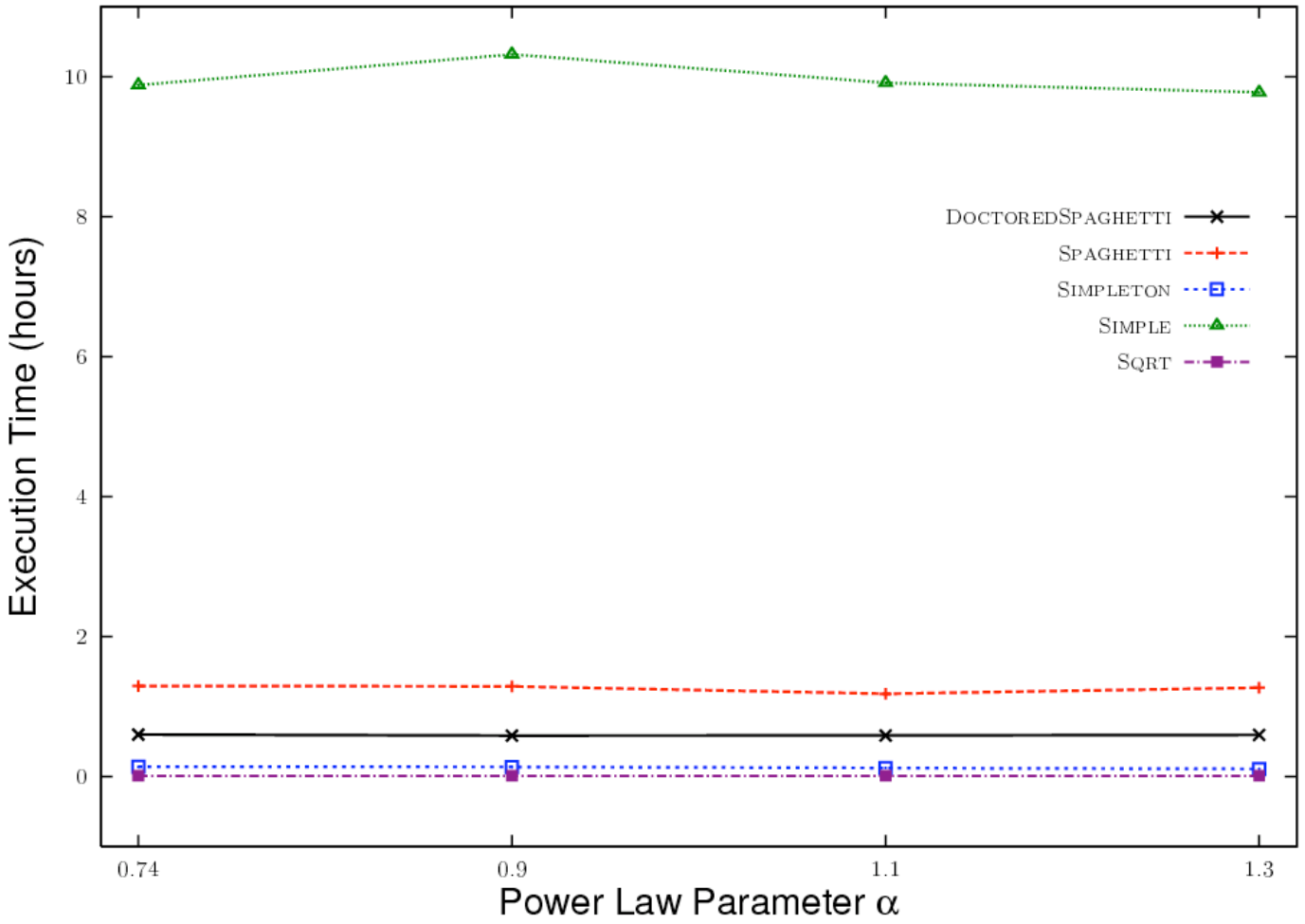
Space


DoctoredSpaghetti
Spaghetti
Simpleton
Simple
Fraction of the Number of Skips Placed w.r.t. Sqrt
Power Law Parameter α
0
0.2
0.4
0.6
0.8
1
0.74
0.9
1.1
1.3

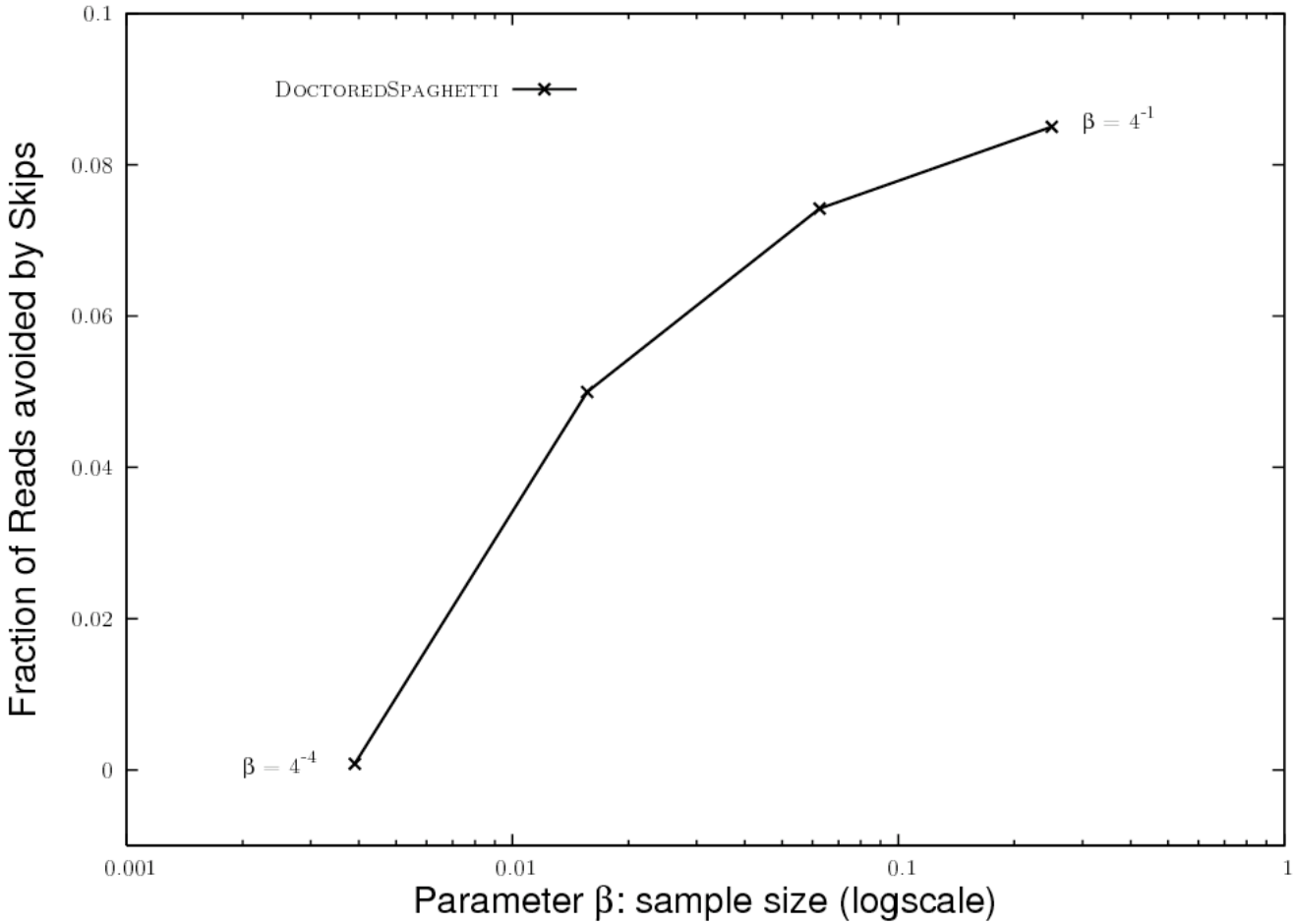
Time to merge



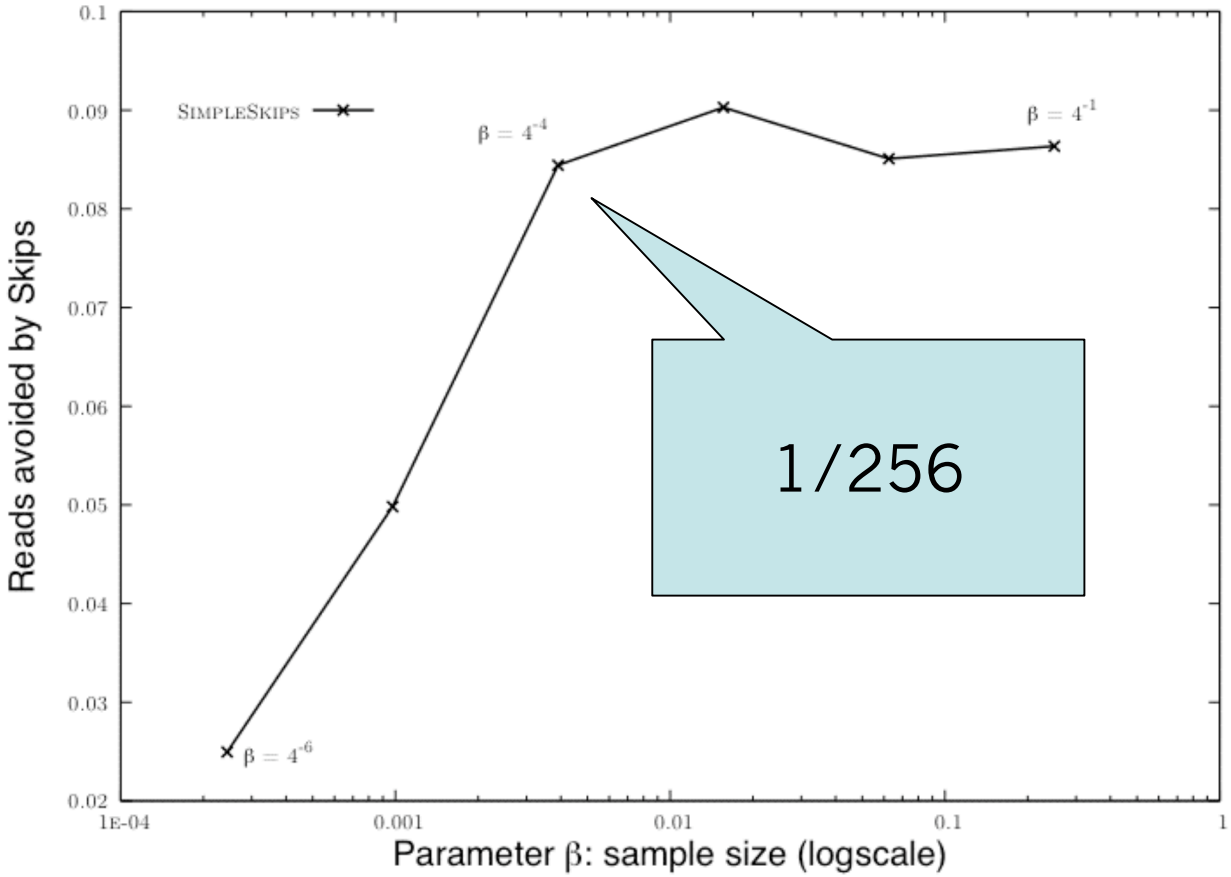
Build up time



Size of query sample for spaghetti skips



Size of query sample for simple skips



The Bottomline

Simple skips are the solution of choice (for power law distributions):

- They merge as fast as spaghetti skips (the general case)
- They occupy less space
- Build time is much faster
- They need a smaller sample to collect statistics on document usefulness

Summing up

- First attempt to exploit in a rigorous way knowledge of the distribution
- Much work remains to be done but results are encouraging

Extensions

- Taking the cache into account
- Taking dependencies into account
- Compare against skip list and other data structures

Thanks for your attention