Superseding Traditional Indexes
with Multicriteria Data Structures

~

GIORGIO VINCIGUERRA

PhD student in Computer Science
giorgio.vinciguerra@phd.unipi.it

Outline

~

1. Multicriteria data structures
2. The dictionary problem
   - External memory model
   - Multiway trees
   - Novel approaches
   - Our results
3. Bonus slides

Motivation

1. Algorithms and data structures often offer a collection of different trade-offs (e.g. time, space occupancy, energy consumption, ...)
2. Software engineers have to choose the one that best fits the needs of their application
3. These needs change with time, data, devices, and users


Welcome Home
Device setup
1 device found
GoogleHome0896 is ready for set up

Multicriteria Data Structures

~

A *multicriteria data structure* selects the best data structure within some performance and computational constraints

FAMILY
of data structures

CONSTRAINTS
space, time, energy…

OPTIMISATION
find the best structure

The dictionary problem

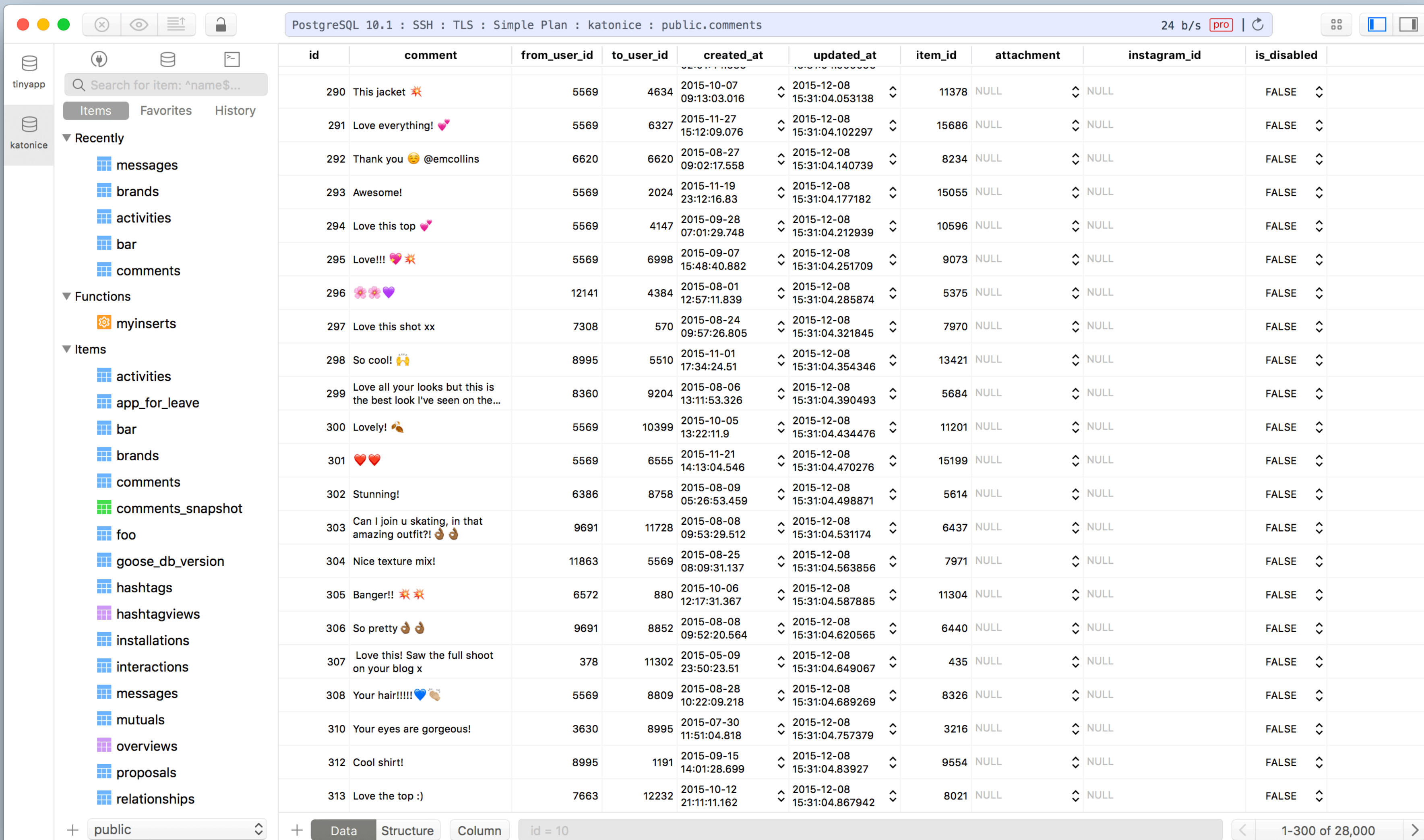
We are given a set of “objects”, and we are asked to store them succinctly and to support efficient retrieval

Databases

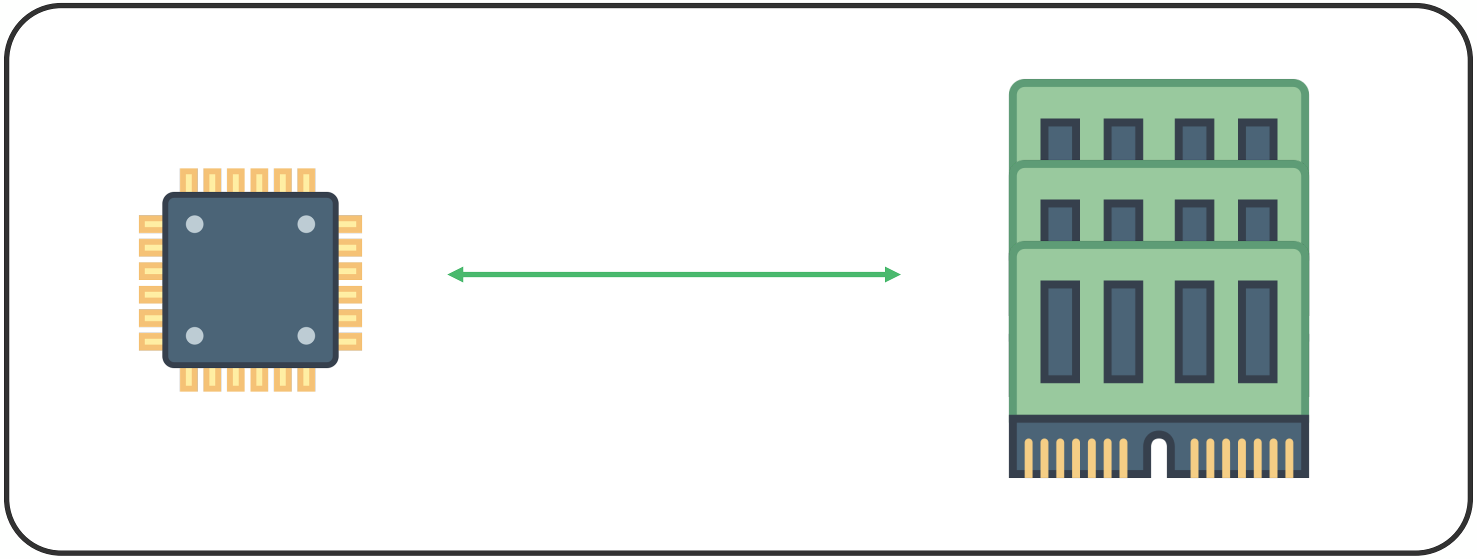
File Systems

Search Engines

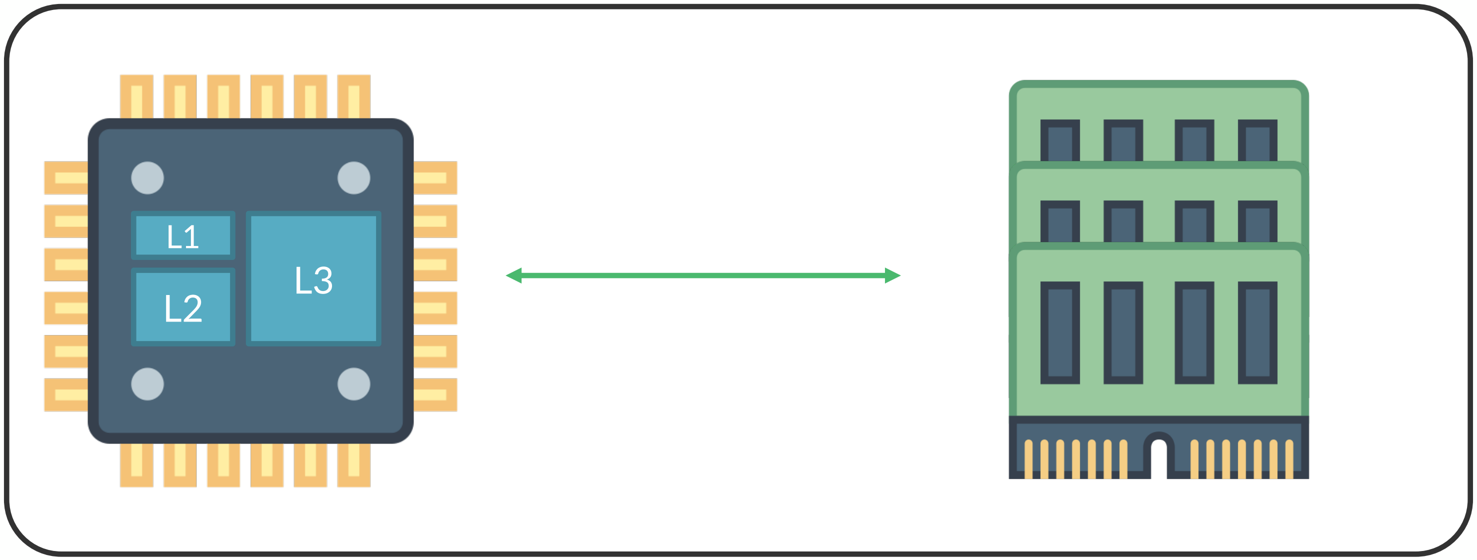
Social Networks


PostgreSQL 10.1 : SSH : TLS : Simple Plan : katonice : public.comments
24 b/s
pro
tinyapp
katonice
Search for item: ^name$...
Items
Favorites
History
Recently
messages
brands
activities
bar
comments
Functions
myinserts
Items
activities
app_for_leave
bar
brands
comments
comments_snapshot
foo
goose_db_version
hashtags
hashtagviews
installations
interactions
messages
mutuals
overviews
proposals
relationships
public
id
comment
from_user_id
to_user_id
created_at
updated_at
item_id
attachment
instagram_id
is_disabled
290 This jacket 💥 5569 4634 2015-10-07 09:13:03.016 2015-12-08 15:31:04.053138 11378 NULL NULL FALSE
291 Love everything! 💕 5569 6327 2015-11-27 15:12:09.076 2015-12-08 15:31:04.102297 15686 NULL NULL FALSE
292 Thank you 😊 @emcollins 6620 6620 2015-08-27 09:02:17.558 2015-12-08 15:31:04.140739 8234 NULL NULL FALSE
293 Awesome! 5569 2024 2015-11-19 23:12:16.83 2015-12-08 15:31:04.177182 15055 NULL NULL FALSE
294 Love this top 💕 5569 4147 2015-09-28 07:01:29.748 2015-12-08 15:31:04.212939 10596 NULL NULL FALSE
295 Love!!! 💖💥 5569 6998 2015-09-07 15:48:40.882 2015-12-08 15:31:04.251709 9073 NULL NULL FALSE
296 🌸🌸💜 12141 4384 2015-08-01 12:57:11.839 2015-12-08 15:31:04.285874 5375 NULL NULL FALSE
297 Love this shot xx 7308 570 2015-08-24 09:57:26.805 2015-12-08 15:31:04.321845 7970 NULL NULL FALSE
298 So cool! 🙌 8995 5510 2015-11-01 17:34:24.51 2015-12-08 15:31:04.354346 13421 NULL NULL FALSE
299 Love all your looks but this is the best look I've seen on the... 8360 9204 2015-08-06 13:11:53.326 2015-12-08 15:31:04.390493 5684 NULL NULL FALSE
300 Lovely! 🍂 5569 10399 2015-10-05 13:22:11.9 2015-12-08 15:31:04.434476 11201 NULL NULL FALSE
301 ❤️❤️ 5569 6555 2015-11-21 14:13:04.546 2015-12-08 15:31:04.470276 15199 NULL NULL FALSE
302 Stunning! 6386 8758 2015-08-09 05:26:53.459 2015-12-08 15:31:04.498871 5614 NULL NULL FALSE
303 Can I join u skating, in that amazing outfit?! 👌👌 9691 11728 2015-08-08 09:53:29.512 2015-12-08 15:31:04.531174 6437 NULL NULL FALSE
304 Nice texture mix! 11863 5569 2015-08-25 08:09:31.137 2015-12-08 15:31:04.563856 7971 NULL NULL FALSE
305 Banger!! 💥💥 6572 880 2015-10-06 12:17:31.367 2015-12-08 15:31:04.587885 11304 NULL NULL FALSE
306 So pretty 👌👌 9691 8852 2015-08-08 09:52:20.564 2015-12-08 15:31:04.620565 6440 NULL NULL FALSE
307 Love this! Saw the full shoot on your blog x 378 11302 2015-05-09 23:50:23.51 2015-12-08 15:31:04.649067 435 NULL NULL FALSE
308 Your hair!!!!!💙👏 5569 8809 2015-08-28 10:22:09.218 2015-12-08 15:31:04.689269 8326 NULL NULL FALSE
310 Your eyes are gorgeous! 3630 8995 2015-07-30 11:51:04.818 2015-12-08 15:31:04.757379 3216 NULL NULL FALSE
312 Cool shirt! 8995 1191 2015-09-15 14:01:28.699 2015-12-08 15:31:04.83927 9554 NULL NULL FALSE
313 Love the top :) 7663 12232 2015-10-12 21:11:11.162 2015-12-08 15:31:04.867942 8021 NULL NULL FALSE
Data
Structure
Column
id = 10
1-300 of 28,000

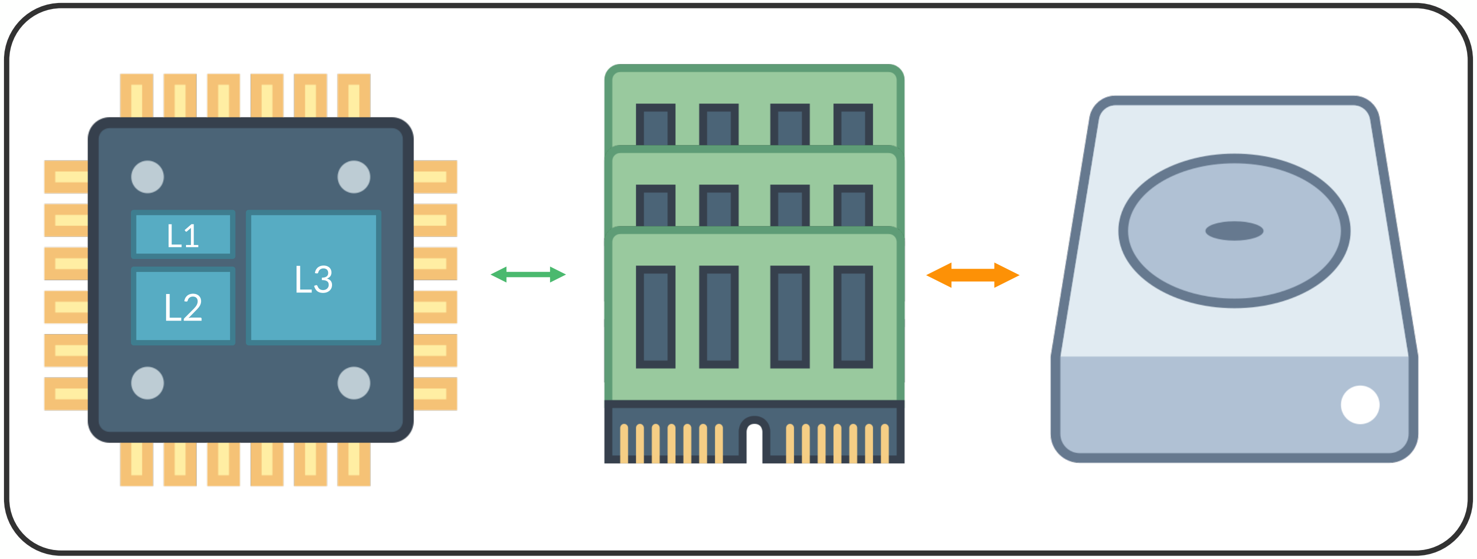
Memory hierarchy



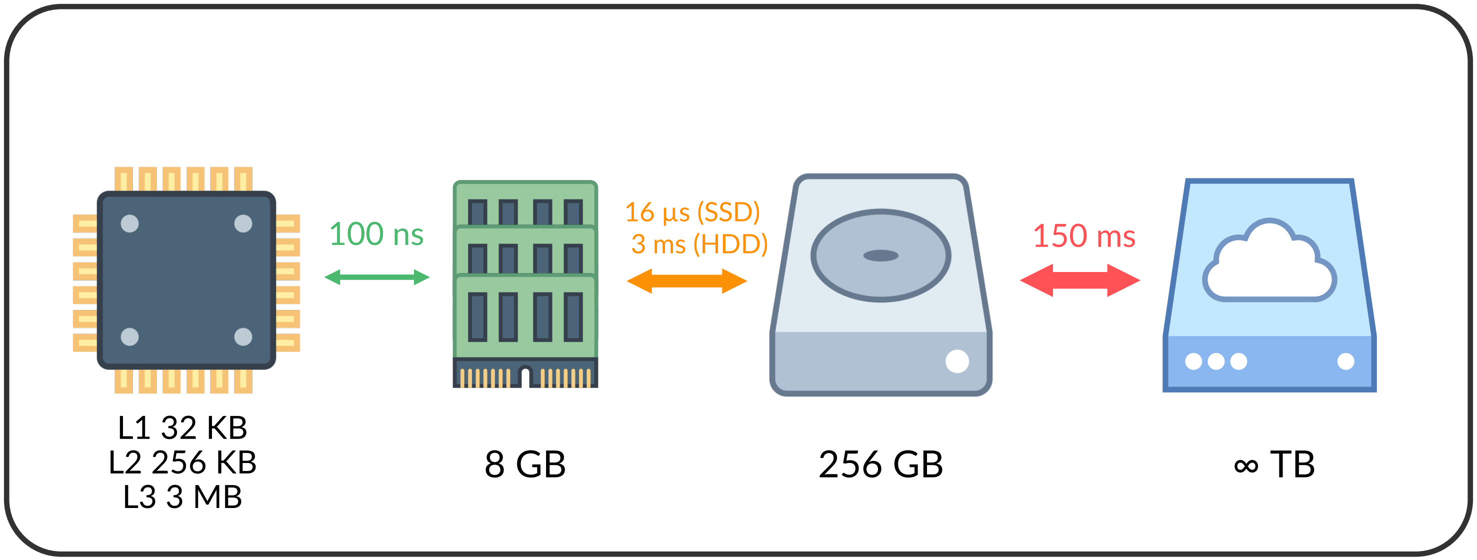
Memory hierarchy



Memory hierarchy



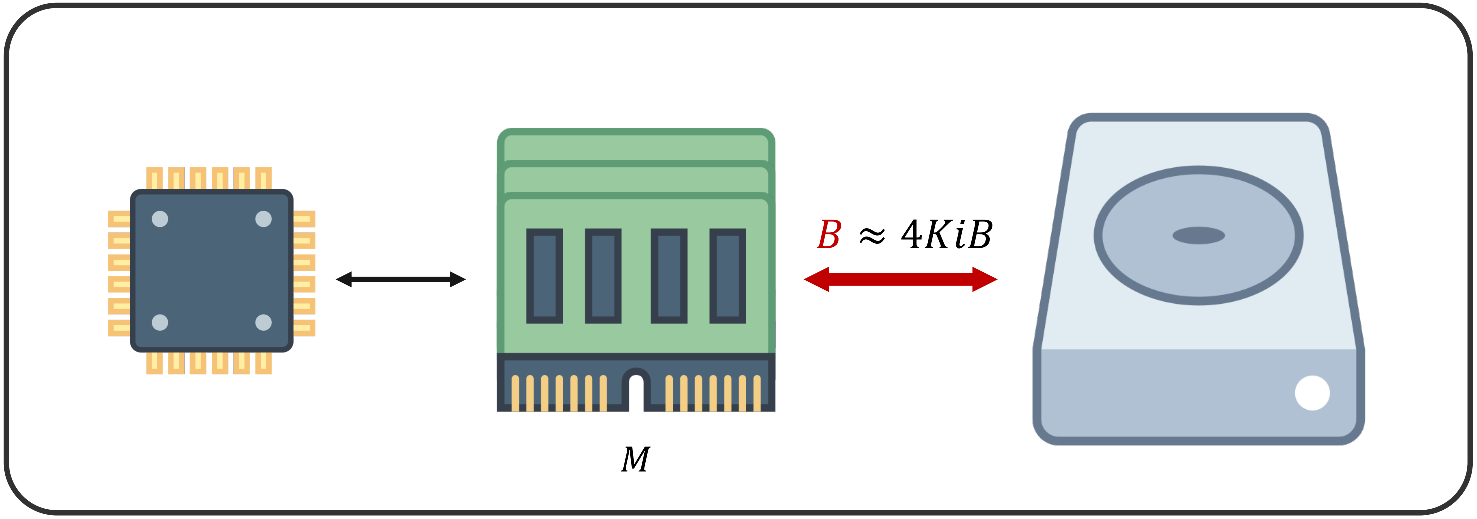
Memory hierarchy




people.eecs.berkeley.edu/~rcs/research/interactive_latency.ht
Fork me on GitHub
Latency Numbers Every Programmer Should Know
1990
1ns
L1 cache reference: 181ns
Branch mispredict: 603ns
Main memory reference: 207ns
1,000ns ≈ 1µs
Compress 1KB wth Zippy: 362,000ns ≈ 362µs
Send 2,000 bytes over commodity network: 1,448,000ns ≈ 1,448µs
SSD random read: 19,000ns ≈ 19µs
Read 1,000,000 bytes sequentially from memory: 3,038,000ns ≈ 3,038µs
Read 1,000,000 bytes sequentially from SSD: 50,000,000ns ≈ 50ms
Disk seek: 20,000,000ns ≈ 20ms
Read 1,000,000 bytes sequentially from disk: 640,000,000ns ≈ 640ms

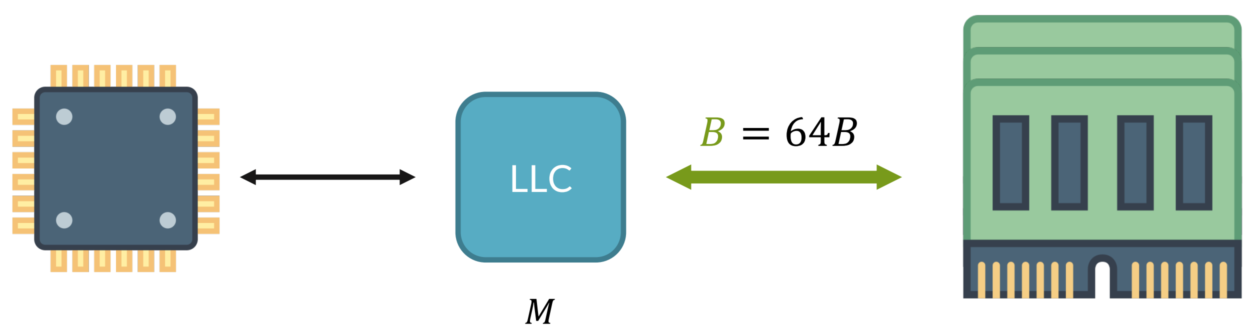
The External Memory (aka I/O) model

1. Internal memory (RAM) of capacity M
2. External memory (disk) of unlimited capacity
3. RAM and disk exchange blocks of size B
4. Count # transfers in Big O instead of # ops



The External Memory (aka I/O) model

1. Internal memory (RAM) of capacity M
2. External memory (disk) of unlimited capacity
3. RAM and disk exchange blocks of size B
4. Count # transfers in Big O instead of # ops



Back to the dictionary problem

~

Integers or reals

We are given a set of "~~objects~~", and we are asked to store them succinctly and to support efficient retrieval

e.g. point and range queries

61	71	12	15	18	1	24	22	88	34	3	10	5	13	55	44	60	2	5	74	90	81

Predecessor search & range queries

~

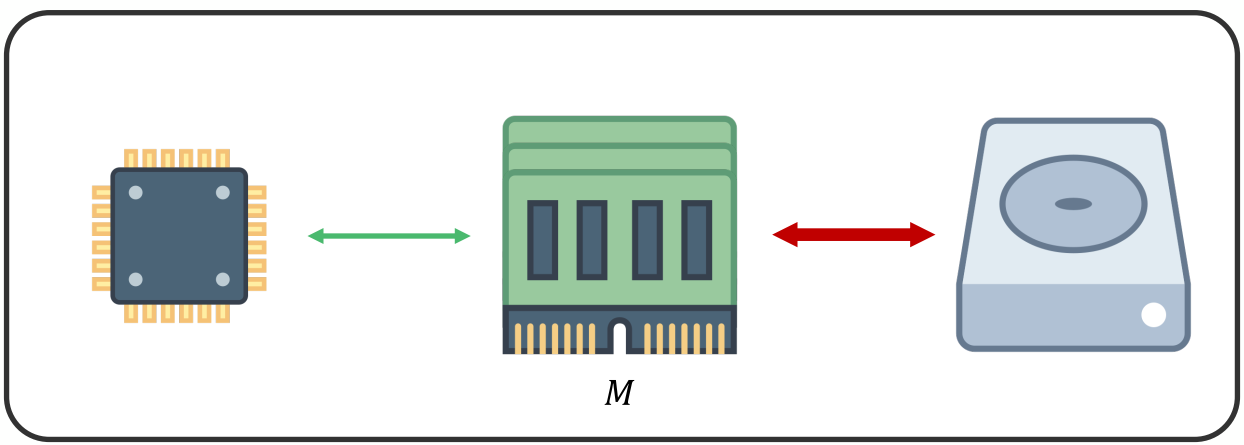
$pred(36) = 36$

$range(67{,}110)$

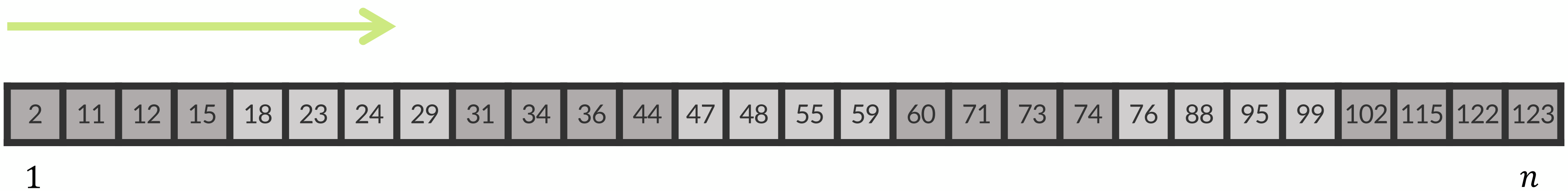
2	11	12	15	18	23	24	29	31	34	36	44	47	48	55	59	60	71	73	74	76	88	95	99	102	115	122	123

1 n

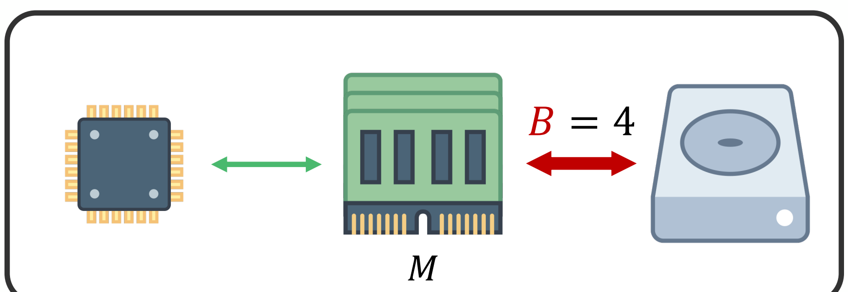
$pred(50) = 48$



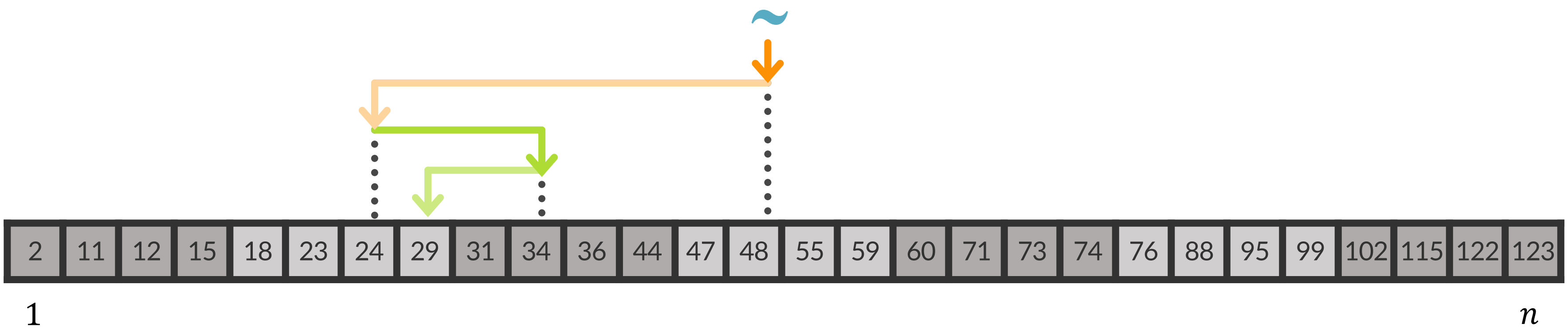
Baseline solutions for predecessor search



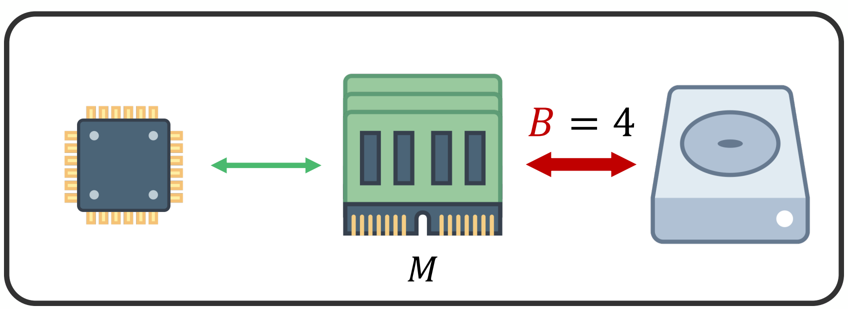
Solution	RAM model **Worst case time**	EM model **Worst case I/Os**	EM model **Best case I/Os**
Scan	$O(n)$	$O(n/B)$	$O(1)$



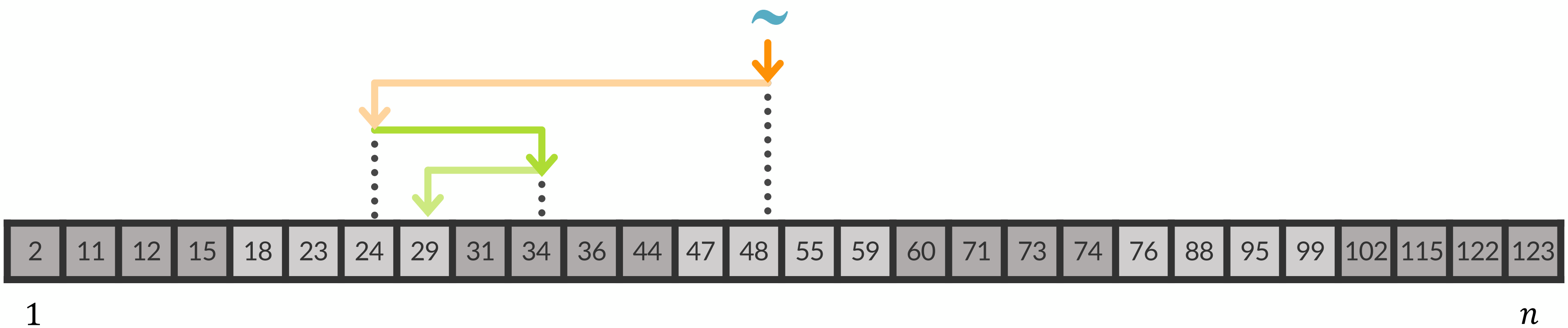
Baseline solutions for predecessor search



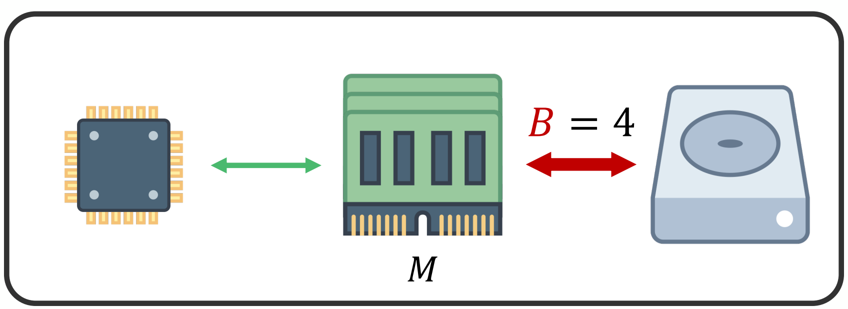
Solution	RAM model **Worst case time**	EM model **Worst case I/Os**	EM model **Best case I/Os**
Scan	$O(n)$	$O(n/B)$	$O(1)$
Binary search	$O(\log n)$		



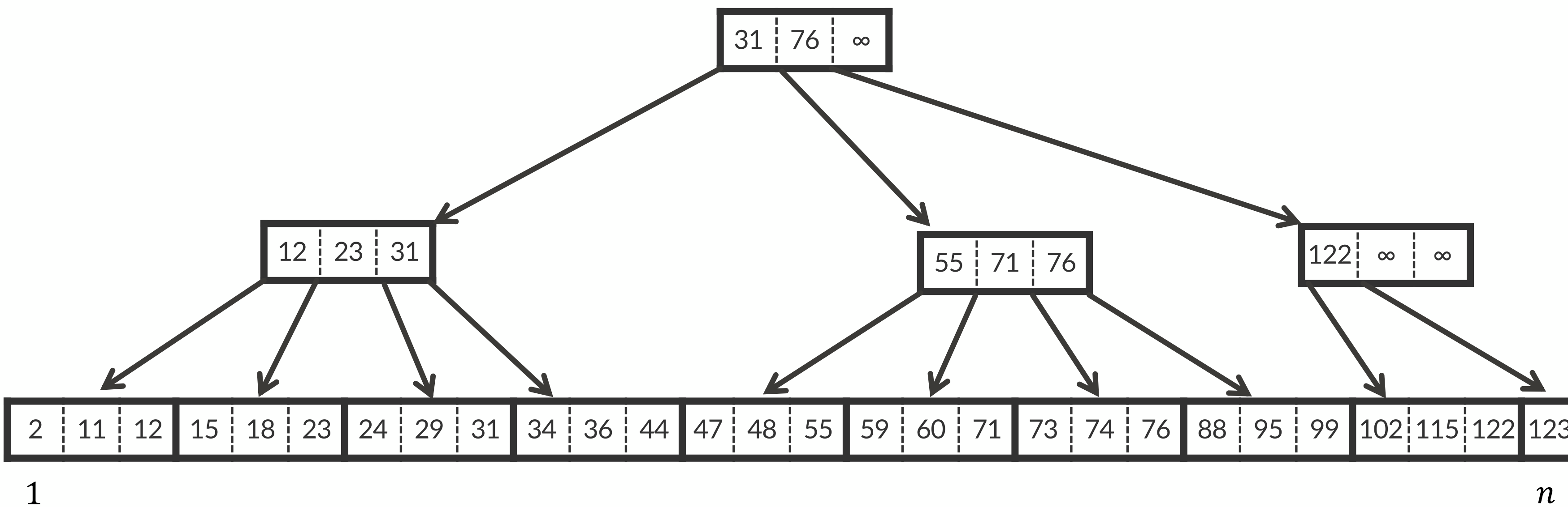
Baseline solutions for predecessor search



Solution	RAM model **Worst case time**	EM model **Worst case I/Os**	EM model **Best case I/Os**
Scan	$O(n)$	$O(n/B)$	$O(1)$
Binary search	$O(\log n)$	$O(\log(n/B))$	$O(\log(n/B))$



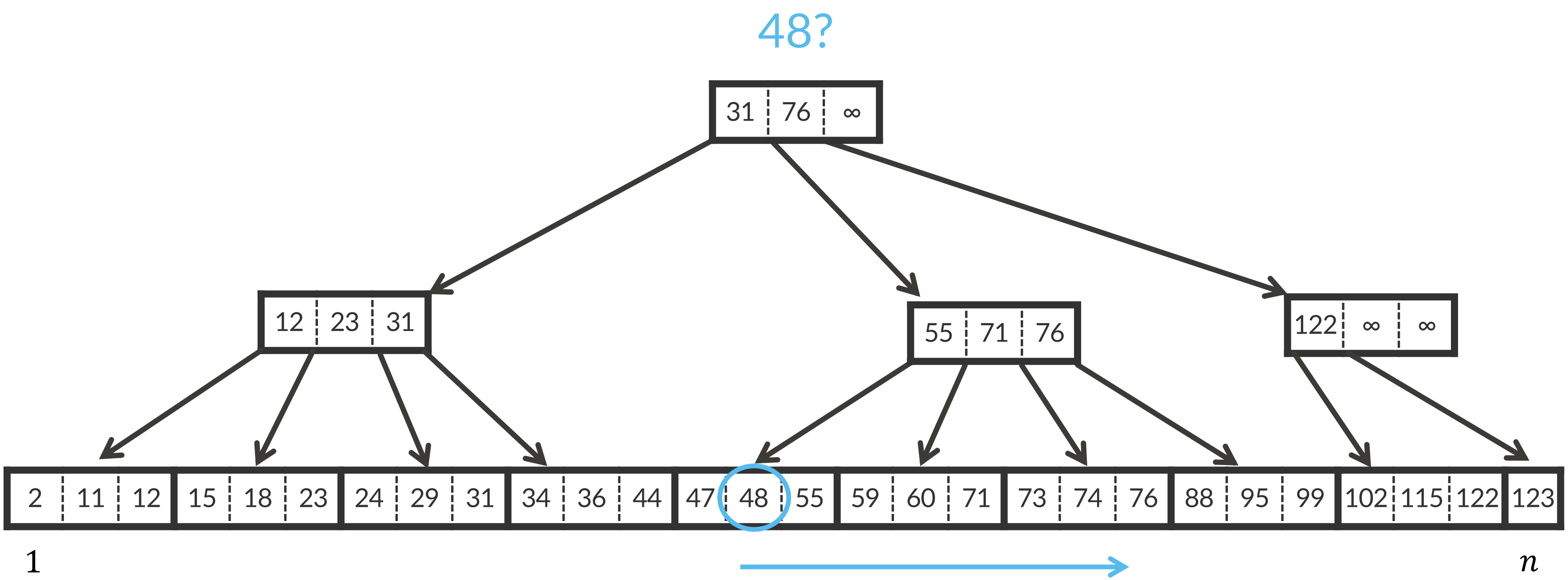
B⁺ trees



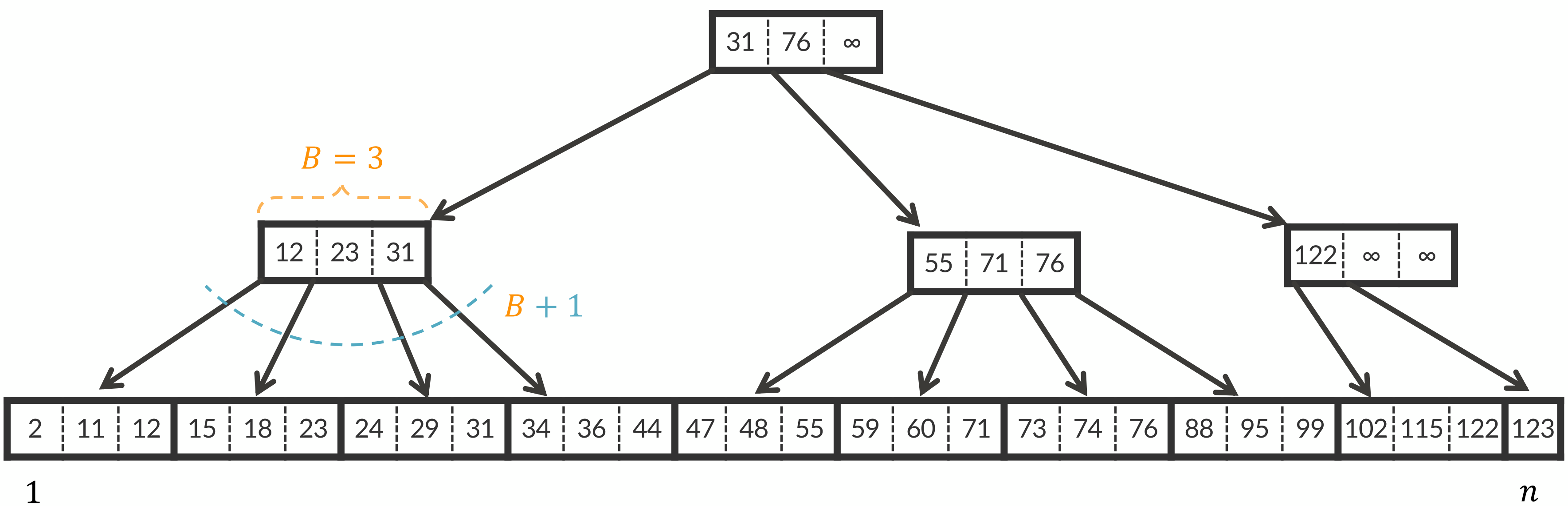
31
76
∞
12
23
31
55
71
76
122
∞
∞
2
11
12
15
18
23
24
29
31
34
36
44
47
48
55
59
60
71
73
74
76
88
95
99
102
115
122
123
1
n

B^+ trees

~


48?
31 76 ∞
12 23 31
55 71 76
122 ∞ ∞
2 11 12
15 18 23
24 29 31
34 36 44
47 48 55
59 60 71
73 74 76
88 95 99
102 115 122
123
1
n

B⁺ trees

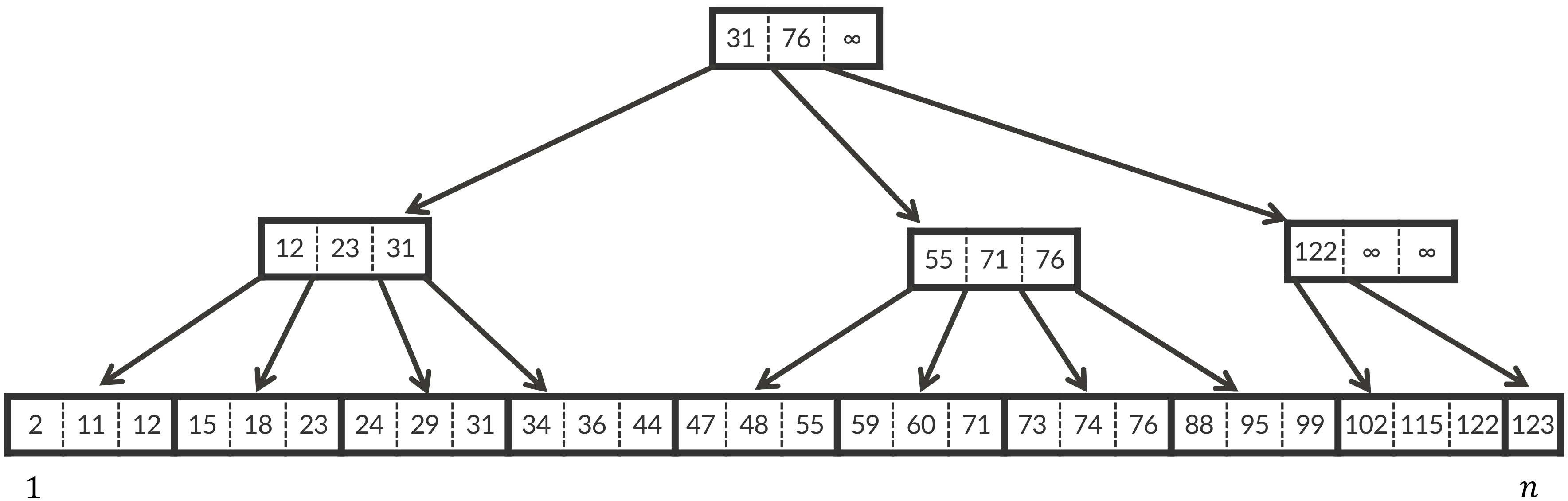


Solution	Space	RAM model **Worst case time**	EM model **Worst case I/Os**	EM model **Best case I/Os**
Scan	$O(1)$	$O(n)$	$O(n/B)$	$O(1)$
Binary search	$O(1)$	$O(\log n)$	$O(\log(n/B))$	$O(\log(n/B))$
B⁺ tree	$O(n)$	$O(\log n)$	$O(\log_B n)$	$O(\log_B n)$

B-trees are everywhere

1. "B-trees have become, de facto, a standard for file organization" Comer. *Ubiquitous B-tree*. ACM Computing Surveys. '79
2. This is still true today

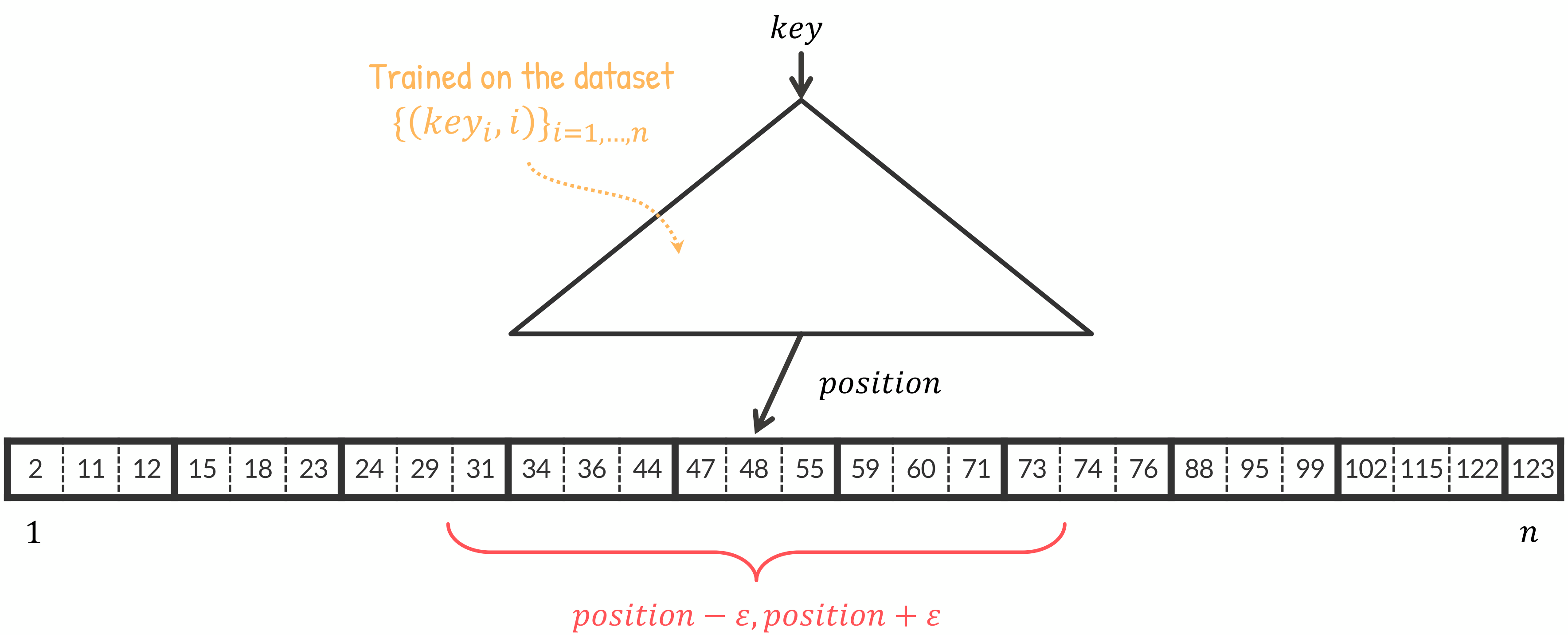
B-trees are everywhere



B-trees are machine learning models

~

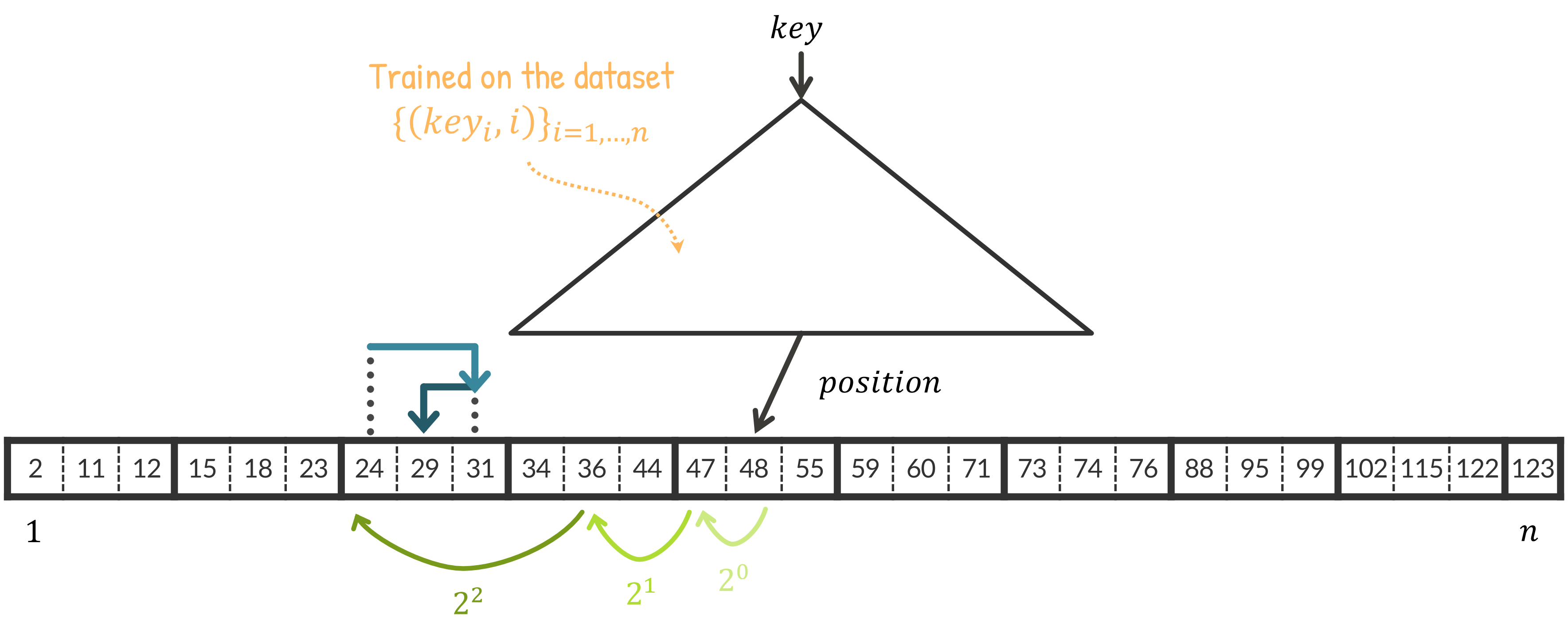
“All existing index structures can be replaced with other types of models, including deep-learning models, which we term learned indexes.”



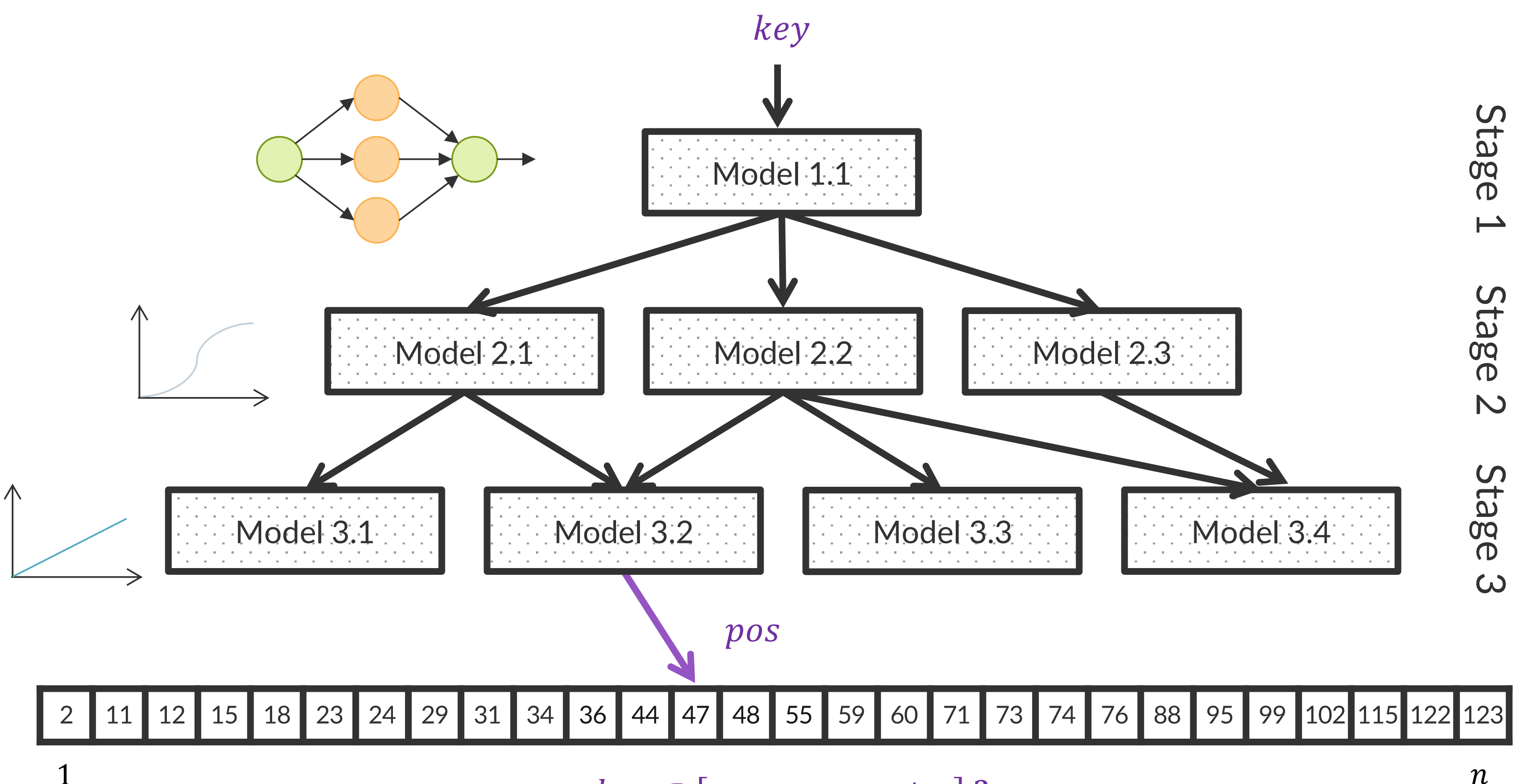
B-trees are machine learning models

~

“All existing index structures can be replaced with other types of models, including deep-learning models, which we term learned indexes.”



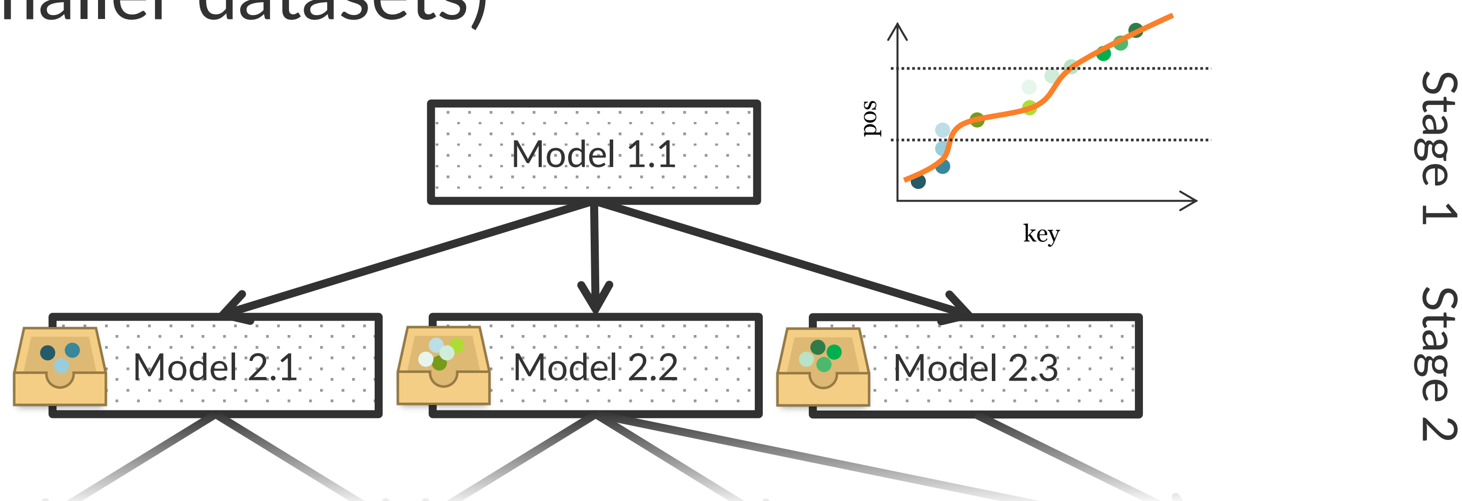
The Recursive Model Index (RMI)



$key \in [pos - \varepsilon, pos + \varepsilon]$?

Construction of RMI

1. Train the root model on the dataset
2. Use it to distribute keys to the next stage
3. Repeat for each model in the next stage (on smaller datasets)



MIT CSAIL + Google

Performance of RMI

~

Type	Config	Map Data			Web Data			Log-Normal Data		
		Size (MB)	Lookup (ns)	Model (ns)	Size (MB)	Lookup (ns)	Model (ns)	Size (MB)	Lookup (ns)	Model (ns)
Btree	page size: 32	52.45 (4.00x)	274 (0.97x)	198 (72.3%)	51.93 (4.00x)	276 (0.94x)	201 (72.7%)	49.83 (4.00x)	274 (0.96x)	198 (72.1%)
	page size: 64	26.23 (2.00x)	277 (0.96x)	172 (62.0%)	25.97 (2.00x)	274 (0.95x)	171 (62.4%)	24.92 (2.00x)	274 (0.96x)	169 (61.7%)
	page size: 128	13.11 (1.00x)	265 (1.00x)	134 (50.8%)	12.98 (1.00x)	260 (1.00x)	132 (50.8%)	12.46 (1.00x)	263 (1.00x)	131 (50.0%)
	page size: 256	6.56 (0.50x)	267 (0.99x)	114 (42.7%)	6.49 (0.50x)	266 (0.98x)	114 (42.9%)	6.23 (0.50x)	271 (0.97x)	117 (43.2%)
	page size: 512	3.28 (0.25x)	286 (0.93x)	101 (35.3%)	3.25 (0.25x)	291 (0.89x)	100 (34.3%)	3.11 (0.25x)	293 (0.90x)	101 (34.5%)
Learned Index	2nd stage models: 10k	0.15 (0.01x)	98 (2.70x)	31 (31.6%)	0.15 (0.01x)	222 (1.17x)	29 (13.1%)	0.15 (0.01x)	178 (1.47x)	26 (14.6%)
	2nd stage models: 50k	0.76 (0.06x)	85 (3.11x)	39 (45.9%)	0.76 (0.06x)	162 (1.60x)	36 (22.2%)	0.76 (0.06x)	162 (1.62x)	35 (21.6%)
	2nd stage models: 100k	1.53 (0.12x)	82 (3.21x)	41 (50.2%)	1.53 (0.12x)	144 (1.81x)	39 (26.9%)	1.53 (0.12x)	152 (1.73x)	36 (23.7%)
	2nd stage models: 200k	3.05 (0.23x)	86 (3.08x)	50 (58.1%)	3.05 (0.24x)	126 (2.07x)	41 (32.5%)	3.05 (0.24x)	146 (1.79x)	40 (27.6%)

Figure 4: Learned Index vs B-Tree

Limitations of RMI

1. Fixed structure with many hyperparameters

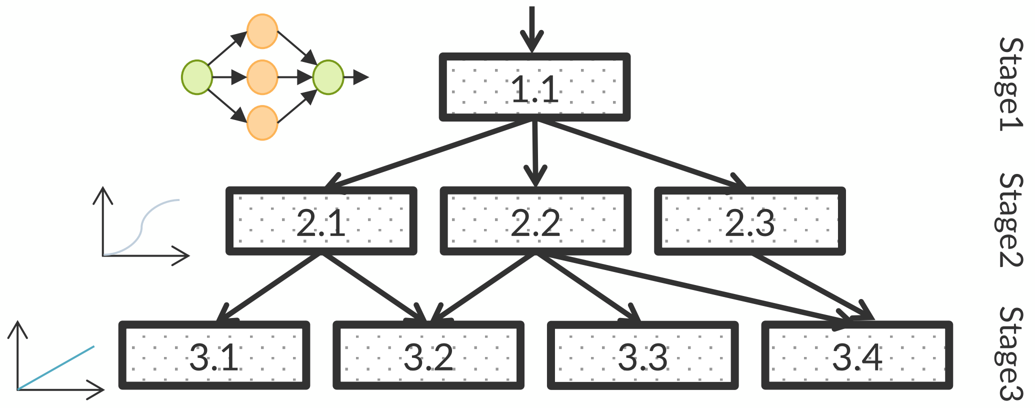
   # stages, # models in each stage, kinds of regression models

2. No a priori error guarantees

   Difficult to predict latencies

3. Models are agnostic to the power of models below

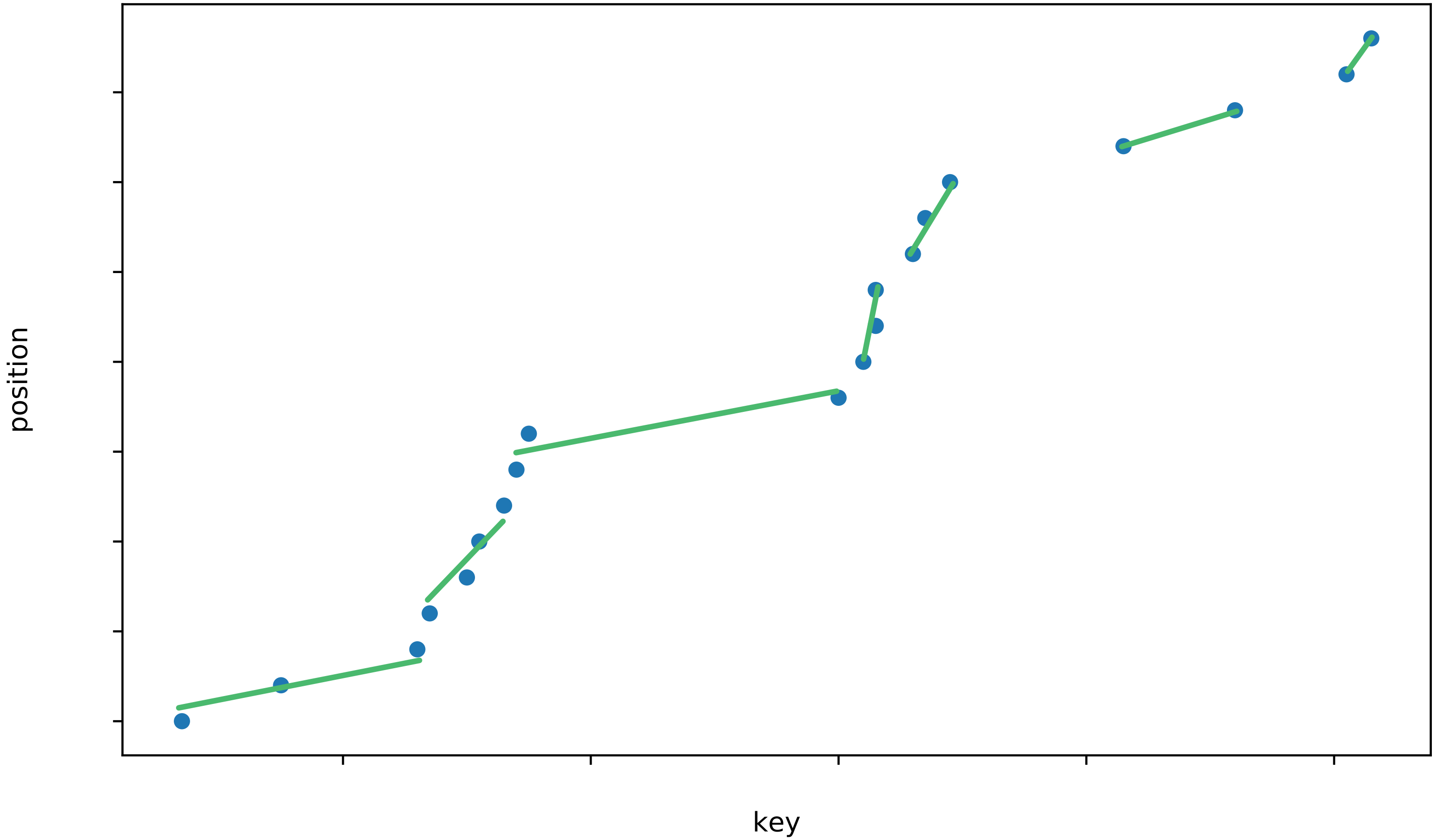
   Can result in underused models (waste of space)



Our idea (submitted)

~

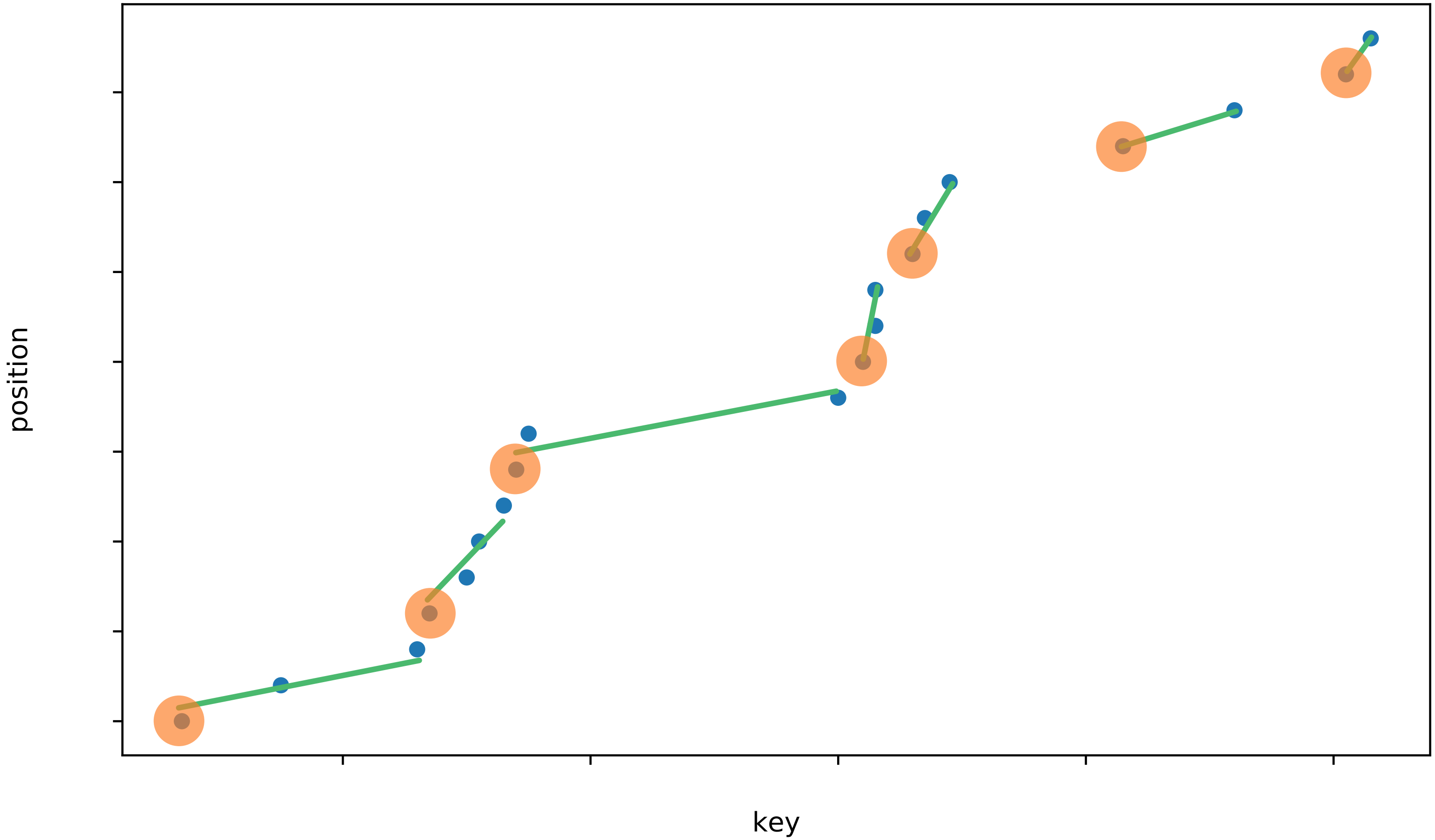
Compute the **optimal** piecewise linear approx with **guaranteed error** ε in $O(n)$



Our idea (submitted)

~

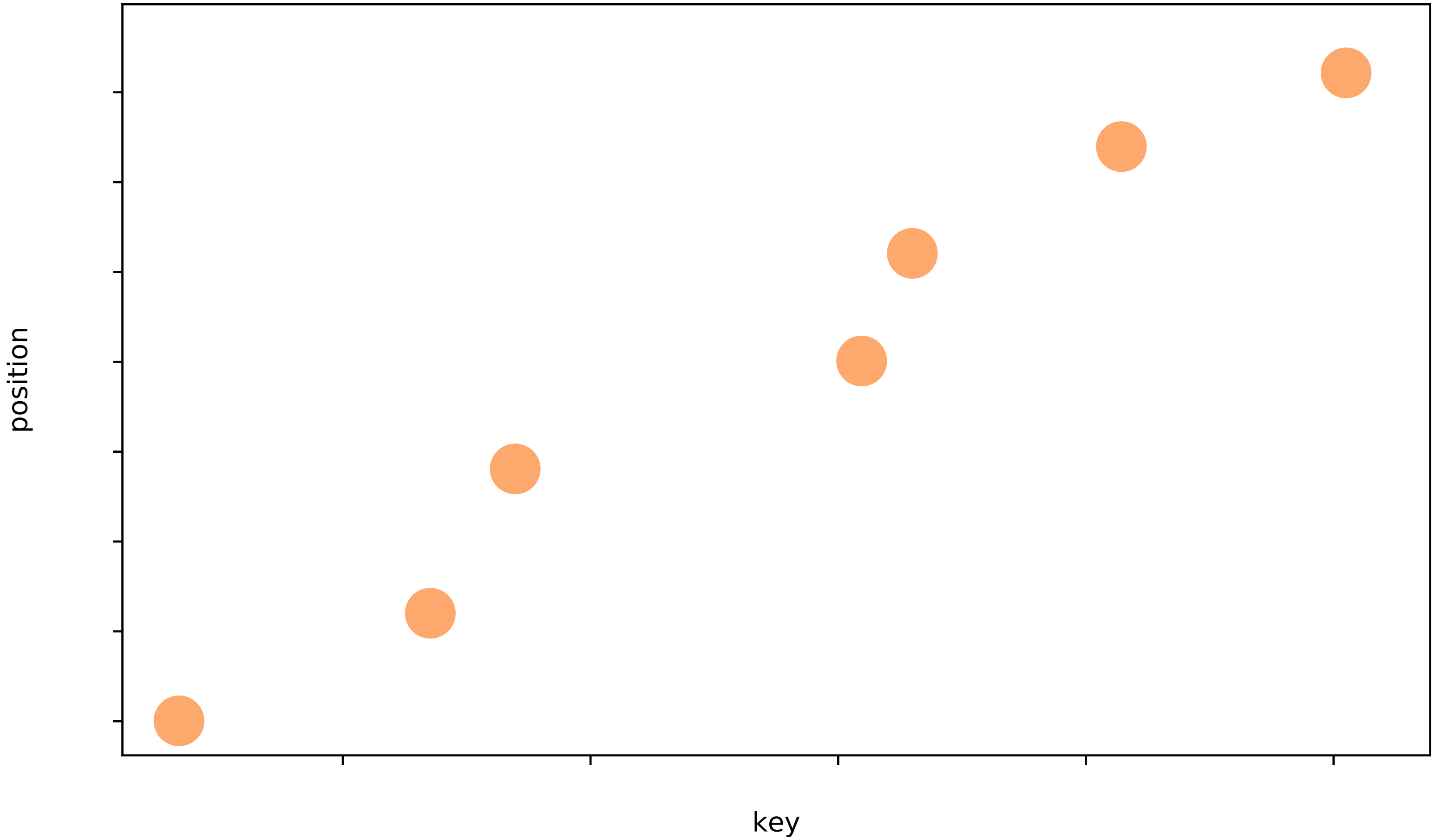
Save the m segments in a vector as triples $s_i = (key, slope, intercept)$



Our idea (submitted)

~

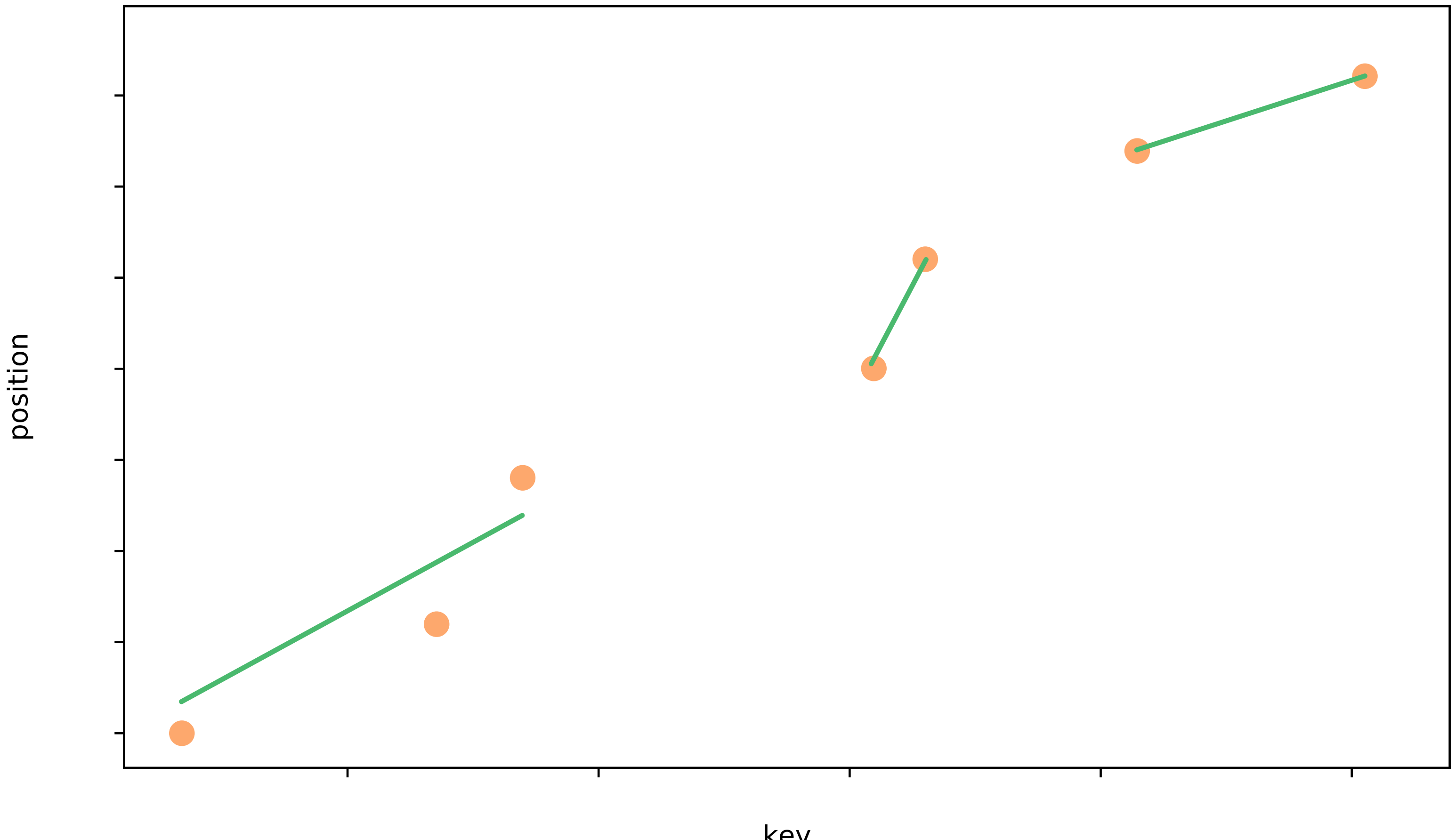
Drop all the points except $s_i.key$



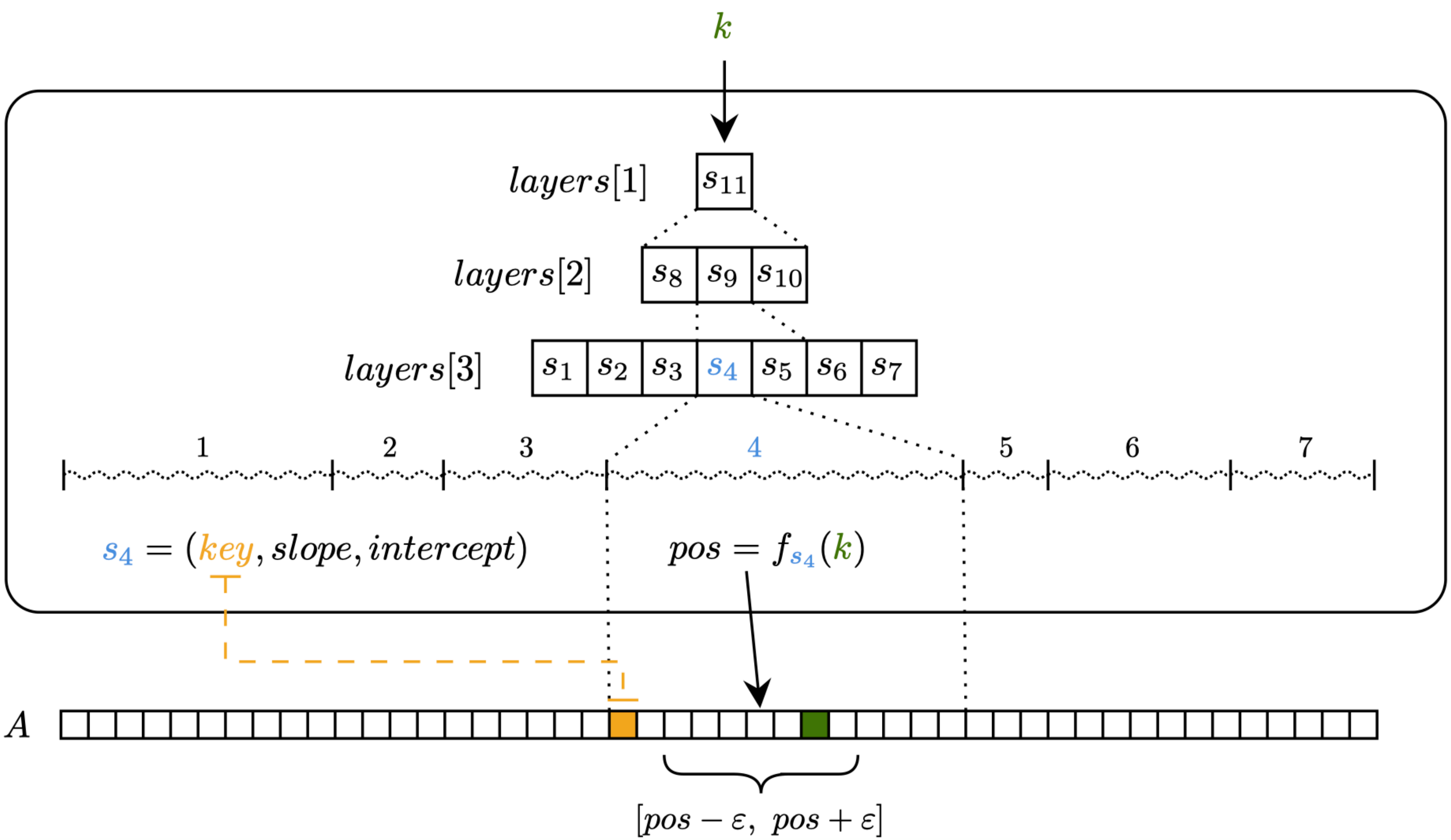
Our idea (submitted)

~

... and repeat!

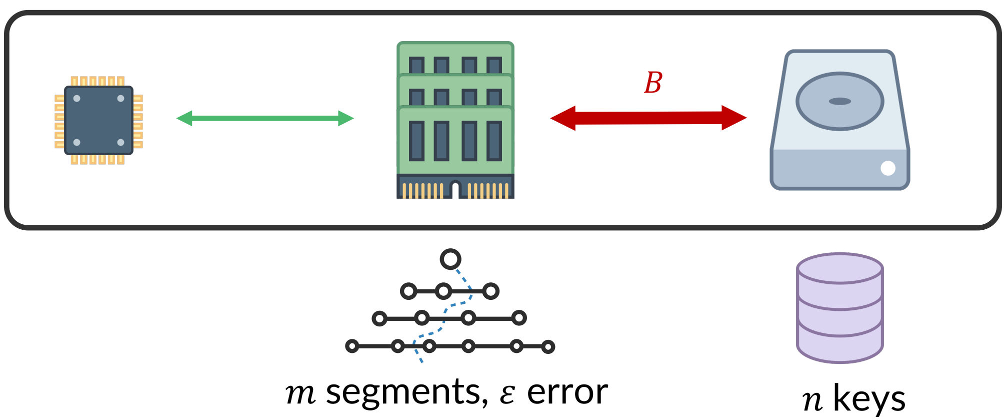


Memory layout of the PGM-index



Some asymptotic bounds

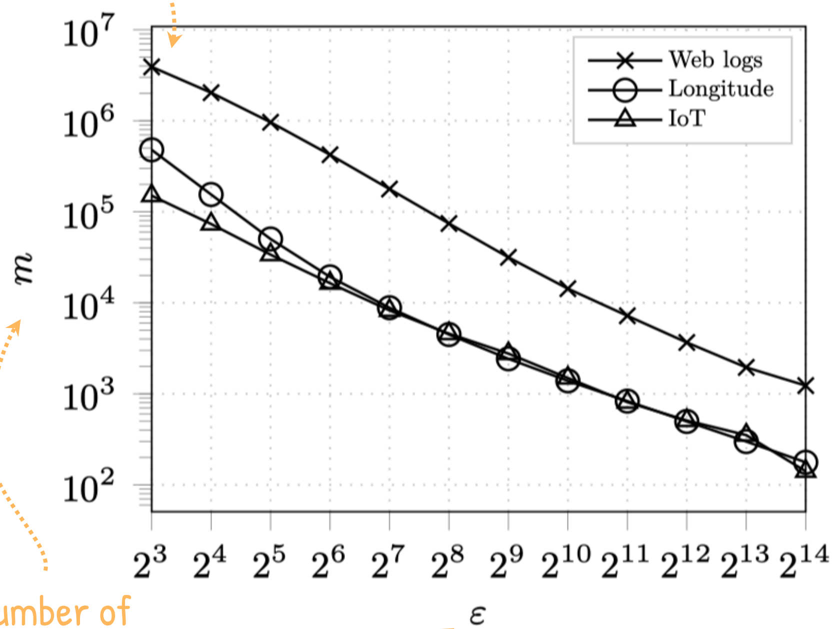
Data Structure	Space of index	RAM model **Worst case time**	EM model **Worst case I/Os**	EM model **Best case I/Os**
Plain sorted array	$O(1)$	$O(\log n)$	$O\left(\log \frac{n}{B}\right)$	$O\left(\log \frac{n}{B}\right)$
Multiway tree	$\Theta(n)$	$O(\log n)$	$O(\log_B n)$	$O(\log_B n)$
RMI	Fixed	$O(?)$	$O(?)$	$O(1)$
PGM-index	$\Theta(m)$	$O(\log m)$	$O(\log_c m)$ $c \geq 2\varepsilon = \Omega(B)$	$O(1)$



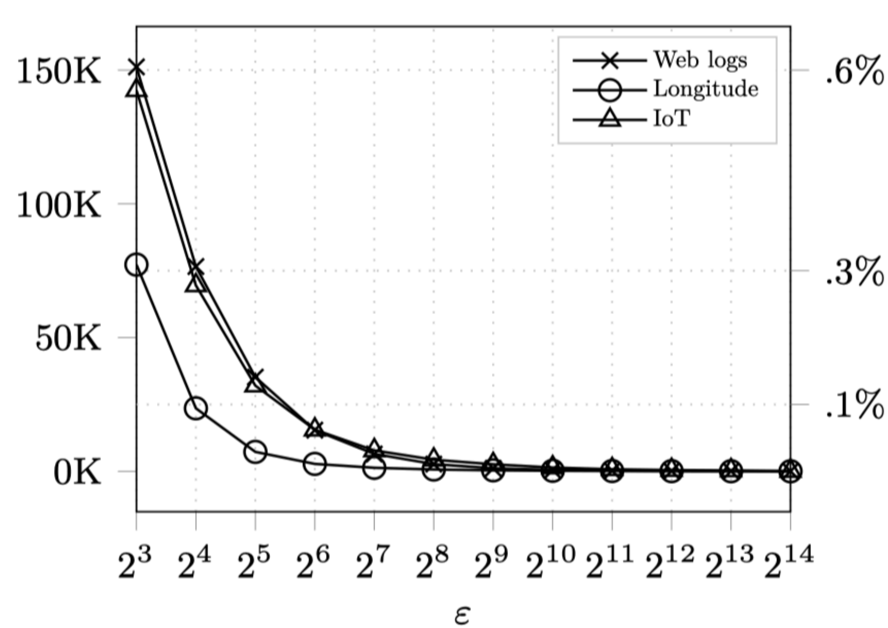
PGM-index in practice

3 seconds to compute

Whole datasets



First 25M entries

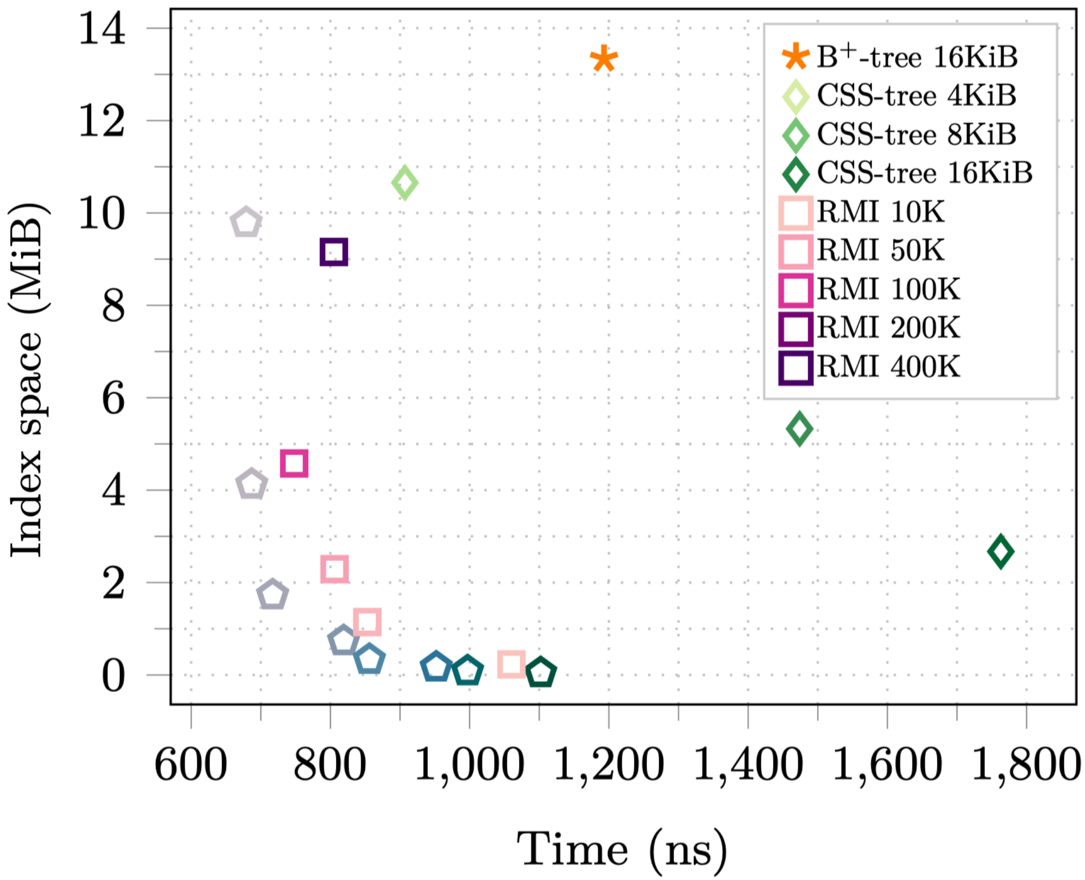


Number of segments

Error of the position estimate

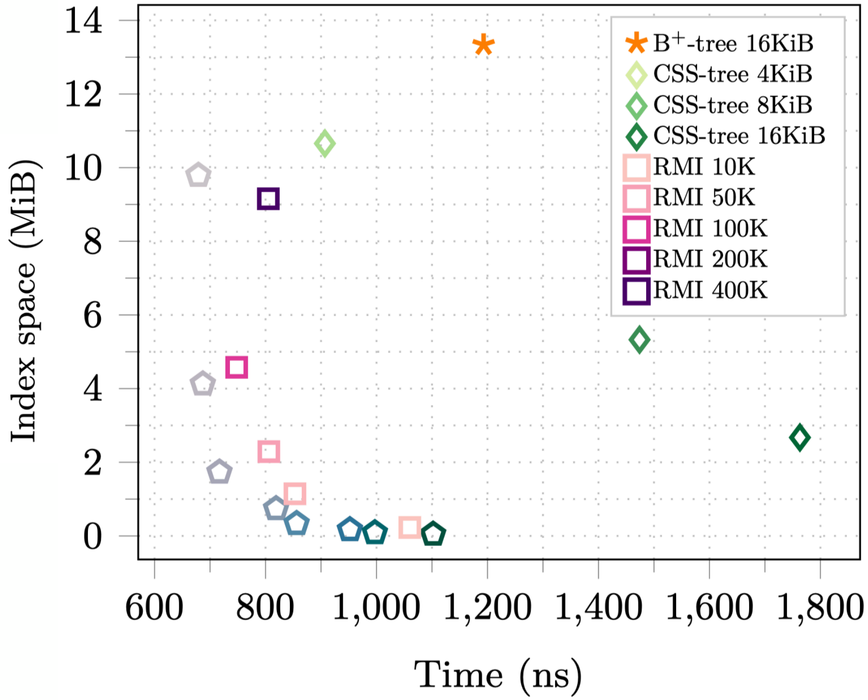
Web logs = 715M points
Longitude = 166M points
IoT = 26M points

Space-time performance



How to explore this space of trade-offs?

Given a space bound S, find efficiently the index that minimizes the query time within space S and vice versa



Back to Multicriteria Data Structures

~

A multicriteria data structure is defined by a family of data structures and an optimisation algorithm that selects the best data structure in the family within some computational constraints

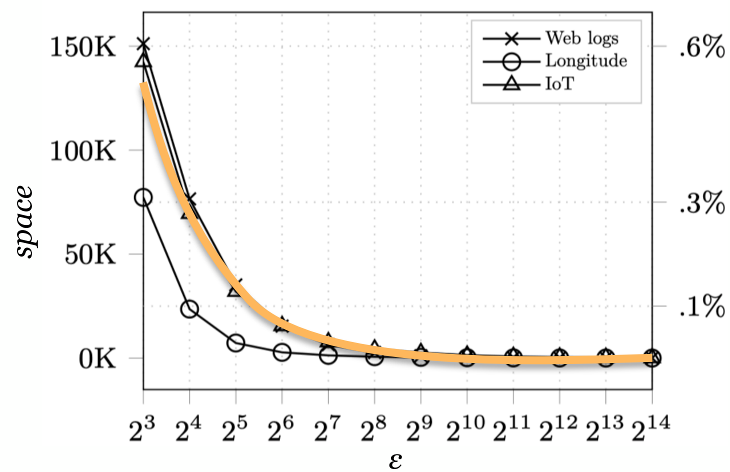
FAMILY
PGM-indexes ∀ε

CONSTRAINTS
Space & Time

OPTIMISATION
???

The Multicriteria PGM-index

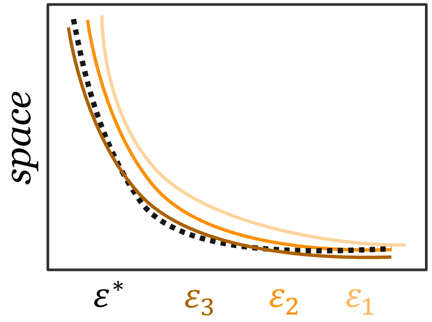
1. We designed a cost model for the space $s(\varepsilon)$ and the time $t(\varepsilon)$
2. … but we don't have a closed formula for $s(\varepsilon)$, it depends on the input array
3. We fit $s(\varepsilon)$ with a power law of the form $a\varepsilon^{-b}$



Under the hood

~

1. A sort of interpolation search over ε values
2. Each iteration improves the fitting of $a\varepsilon^{-b}$ updating a, b
3. Bias the ε-iterate towards the midpoint of a bin. search
4. In practice, given a space (time) bound, it finds the fastest (most compact) index for 715M keys in < 1 min



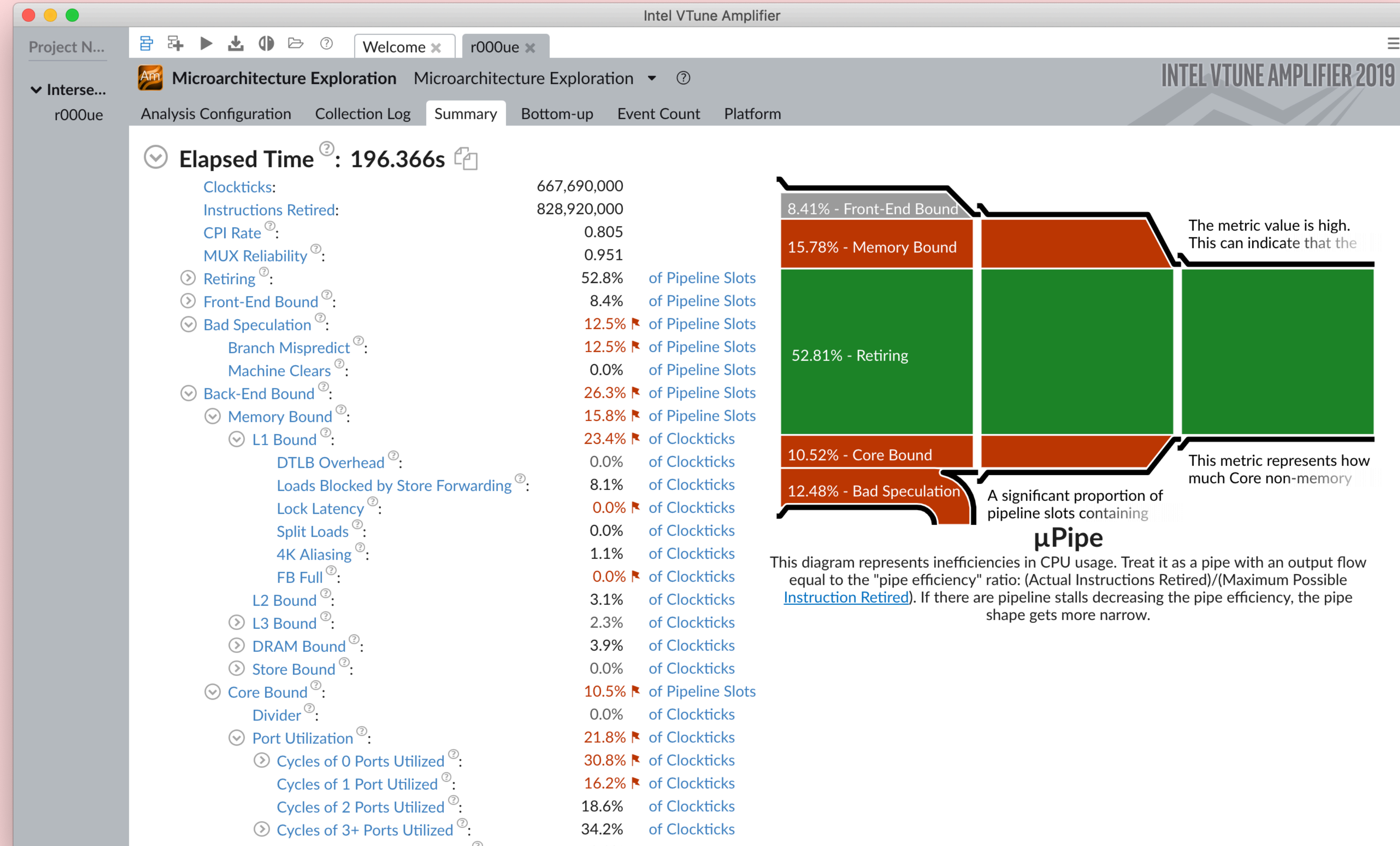
Future work

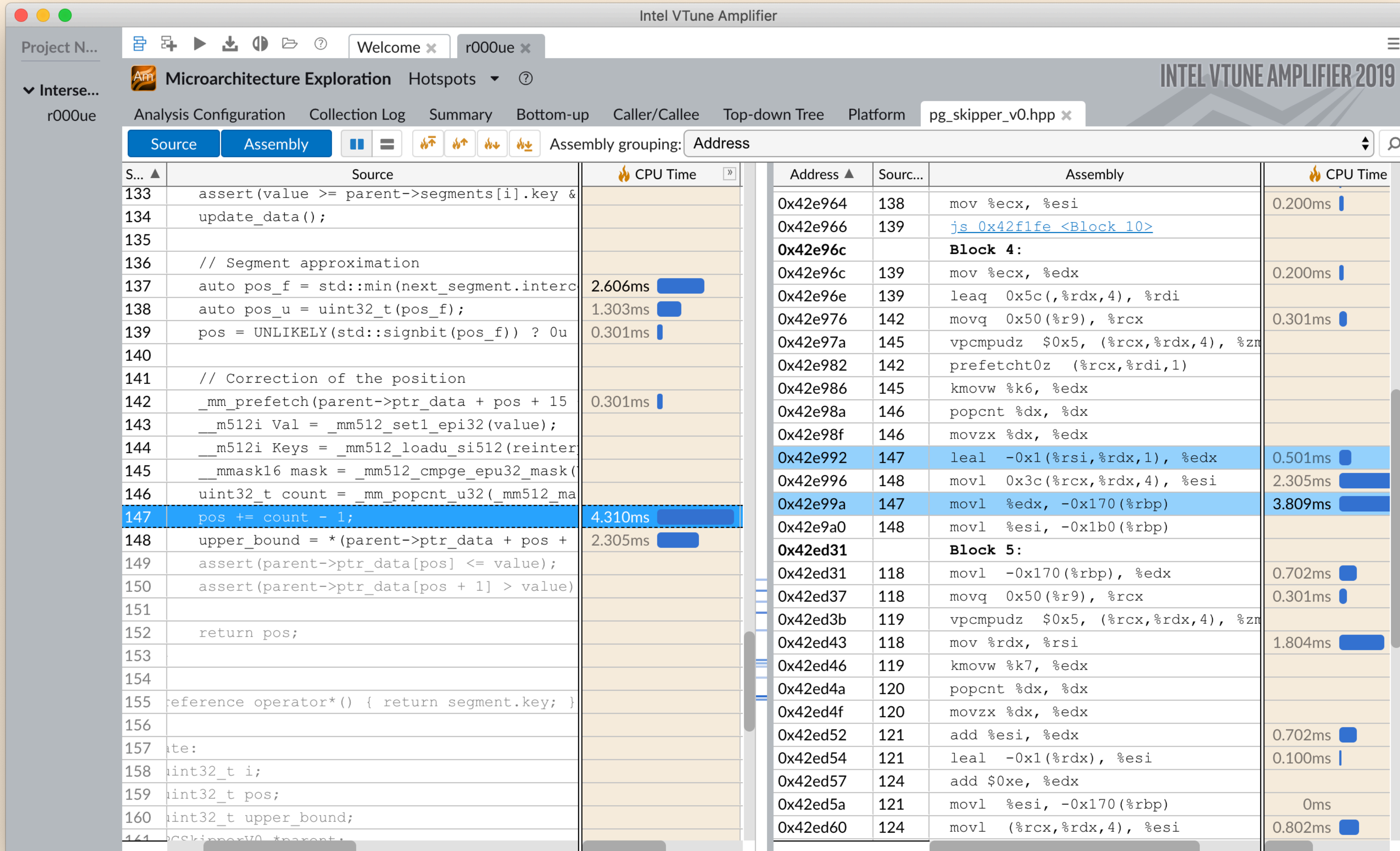
1. Insertion and deletions
2. Non-linear models
3. Compression

Bonus slides

~

Tools that you may find useful


Intel VTune Amplifier
Project N...
Interse...
r000ue
Welcome
r000ue
Microarchitecture Exploration
Microarchitecture Exploration
INTEL VTUNE AMPLIFIER 2019
Analysis Configuration
Collection Log
Summary
Bottom-up
Event Count
Platform
Elapsed Time: 196.366s
Clockticks: 667,690,000
Instructions Retired: 828,920,000
CPI Rate: 0.805
MUX Reliability: 0.951
Retiring: 52.8% of Pipeline Slots
Front-End Bound: 8.4% of Pipeline Slots
Bad Speculation: 12.5% of Pipeline Slots
Branch Mispredict: 12.5% of Pipeline Slots
Machine Clears: 0.0% of Pipeline Slots
Back-End Bound: 26.3% of Pipeline Slots
Memory Bound: 15.8% of Pipeline Slots
L1 Bound: 23.4% of Clockticks
DTLB Overhead: 0.0% of Clockticks
Loads Blocked by Store Forwarding: 8.1% of Clockticks
Lock Latency: 0.0% of Clockticks
Split Loads: 0.0% of Clockticks
4K Aliasing: 1.1% of Clockticks
FB Full: 0.0% of Clockticks
L2 Bound: 3.1% of Clockticks
L3 Bound: 2.3% of Clockticks
DRAM Bound: 3.9% of Clockticks
Store Bound: 0.0% of Clockticks
Core Bound: 10.5% of Pipeline Slots
Divider: 0.0% of Clockticks
Port Utilization: 21.8% of Clockticks
Cycles of 0 Ports Utilized: 30.8% of Clockticks
Cycles of 1 Port Utilized: 16.2% of Clockticks
Cycles of 2 Ports Utilized: 18.6% of Clockticks
Cycles of 3+ Ports Utilized: 34.2% of Clockticks
8.41% - Front-End Bound
15.78% - Memory Bound
52.81% - Retiring
10.52% - Core Bound
12.48% - Bad Speculation
The metric value is high. This can indicate that the
This metric represents how much Core non-memory
A significant proportion of pipeline slots containing
µPipe
This diagram represents inefficiencies in CPU usage. Treat it as a pipe with an output flow equal to the "pipe efficiency" ratio: (Actual Instructions Retired)/(Maximum Possible Instruction Retired). If there are pipeline stalls decreasing the pipe efficiency, the pipe shape gets more narrow.


Intel VTune Amplifier
Project N...
Interse...
r000ue
Welcome
r000ue
Microarchitecture Exploration
Hotspots
INTEL VTUNE AMPLIFIER 2019
Analysis Configuration
Collection Log
Summary
Bottom-up
Caller/Callee
Top-down Tree
Platform
pg_skipper_v0.hpp
Source
Assembly
Assembly grouping:
Address
S...
Source
CPU Time
133 assert(value >= parent->segments[i].key &
134 update_data();
135
136 // Segment approximation
137 auto pos_f = std::min(next_segment.interc 2.606ms
138 auto pos_u = uint32_t(pos_f); 1.303ms
139 pos = UNLIKELY(std::signbit(pos_f)) ? 0u 0.301ms
140
141 // Correction of the position
142 _mm_prefetch(parent->ptr_data + pos + 15 0.301ms
143 __m512i Val = _mm512_set1_epi32(value);
144 __m512i Keys = _mm512_loadu_si512(reinter
145 __mmask16 mask = _mm512_cmpge_epu32_mask(
146 uint32_t count = _mm_popcnt_u32(_mm512_ma
147 pos += count - 1; 4.310ms
148 upper_bound = *(parent->ptr_data + pos + 2.305ms
149 assert(parent->ptr_data[pos] <= value);
150 assert(parent->ptr_data[pos + 1] > value)
151
152 return pos;
153
154
155 reference operator*() { return segment.key; }
156
157 ate:
158 uint32_t i;
159 uint32_t pos;
160 uint32_t upper_bound;
Address
Sourc...
Assembly
CPU Time
0x42e964 138 mov %ecx, %esi 0.200ms
0x42e966 139 js 0x42f1fe <Block 10>
0x42e96c Block 4:
0x42e96c 139 mov %ecx, %edx 0.200ms
0x42e96e 139 leaq 0x5c(,%rdx,4), %rdi
0x42e976 142 movq 0x50(%r9), %rcx 0.301ms
0x42e97a 145 vpcmpudz $0x5, (%rcx,%rdx,4), %zm
0x42e982 142 prefetcht0z (%rcx,%rdi,1)
0x42e986 145 kmovw %k6, %edx
0x42e98a 146 popcnt %dx, %dx
0x42e98f 146 movzx %dx, %edx
0x42e992 147 leal -0x1(%rsi,%rdx,1), %edx 0.501ms
0x42e996 148 movl 0x3c(%rcx,%rdx,4), %esi 2.305ms
0x42e99a 147 movl %edx, -0x170(%rbp) 3.809ms
0x42e9a0 148 movl %esi, -0x1b0(%rbp)
0x42ed31 Block 5:
0x42ed31 118 movl -0x170(%rbp), %edx 0.702ms
0x42ed37 118 movq 0x50(%r9), %rcx 0.301ms
0x42ed3b 119 vpcmpudz $0x5, (%rcx,%rdx,4), %zm
0x42ed43 118 mov %rdx, %rsi 1.804ms
0x42ed46 119 kmovw %k7, %edx
0x42ed4a 120 popcnt %dx, %dx
0x42ed4f 120 movzx %dx, %edx
0x42ed52 121 add %esi, %edx 0.702ms
0x42ed54 121 leal -0x1(%rdx), %esi 0.100ms
0x42ed57 124 add $0xe, %edx
0x42ed5a 121 movl %esi, -0x170(%rbp) 0ms
0x42ed60 124 movl (%rcx,%rdx,4), %esi 0.802ms

The `%timeit` built-in line magic

In [1]:
```
from random import uniform
from itertools import cycle

gen_point = lambda: (uniform(0, 100), uniform(0, 100))
points_pairs = [(gen_point(), gen_point()) for _ in range(100000)]
iter_points_pairs = cycle(points_pairs)
```

In [2]:
```
import math

def py_distance(p1, p2):
    dx = p2[0] - p1[0]
    dy = p2[1] - p1[1]
    return math.sqrt(dx**2 + dy**2)

%timeit pts = next(iter_points_pairs); py_distance(*pts)
```

```
891 ns ± 139 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

In [3]:
```
from scipy.spatial import distance

%timeit pts = next(iter_points_pairs); distance.euclidean(*pts)
```

```
31.9 µs ± 9.77 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Cython

In [ ]:
```
!pip install cython
%load_ext Cython
```

In [5]:
```
%%cython -a

cimport libc.math

def cython_distance((double, double) p1, (double, double) p2):
    cdef double dx = p2[0] - p1[0]
    cdef double dy = p2[1] - p1[1]
    cdef double res = libc.math.sqrt(dx * dx + dy * dy)
    return res
```

Out[5]:

Generated by Cython 0.29.7

Yellow lines hint at Python interaction.
Click on a line that starts with a " + " to see the C code that Cython generated for it.

```
 1:
 2: cimport libc.math
 3:
+4: def cython_distance((double, double) p1, (double, double) p2):
+5:     cdef double dx = p2[0] - p1[0]
+6:     cdef double dy = p2[1] - p1[1]
+7:     cdef double res = libc.math.sqrt(dx * dx + dy * dy)
+8:     return res
```

In [6]:
```
%timeit pts = next(iter_points_pairs); cython_distance(*pts)
```

```
272 ns ± 173 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

3× faster than py_distance
117× faster than scipy.spatial.distance.euclidean

The `%lprun` magic (from the `line_profiler` module)

In [ ]:
```
!pip install line_profiler
%load_ext line_profiler
```

In [9]:
```
%lprun -f distance.euclidean [distance.euclidean(p1, p2) for p1, p2 in points_pairs]
```

```
Total time: 8.01555 s
File: /usr/local/miniconda3/lib/python3.6/site-packages/scipy/spatial/distance.py
Function: euclidean at line 566

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   566                                           def euclidean(u, v, w=None):
   567                                               """
   568                                               Computes the Euclidean distance between two 1-D arrays.
   569
   570                                               The Euclidean distance between 1-D arrays `u` and `v`, is defined as
   571
   572                                               .. math::
   573
   574                                                  {||u-v||}_2
   575
   576                                                  \\left(\\sum{(w_i |(u_i - v_i)|^2)}\\right)^{1/2}
   577
   578                                               Parameters
   579                                               ----------
   580                                               u : (N,) array_like
   581                                                   Input array.
   582                                               v : (N,) array_like
   583                                                   Input array.
   584                                               w : (N,) array_like, optional
   585                                                   The weights for each value in `u` and `v`. Default is None,
   586                                                   which gives each value a weight of 1.0
   587
   588                                               Returns
   589                                               -------
   590                                               euclidean : double
   591                                                   The Euclidean distance between vectors `u` and `v`.
   592
   593                                               Examples
   594                                               --------
   595                                               >>> from scipy.spatial import distance
   596                                               >>> distance.euclidean([1, 0, 0], [0, 1, 0])
   597                                               1.4142135623730951
   598                                               >>> distance.euclidean([1, 1, 0], [0, 1, 0])
   599                                               1.0
   600
   601                                               """
   602    100000    8015546.0     80.2    100.0      return minkowski(u, v, p=2, w=w)
```

The `%lprun` magic (from the `line_profiler` module)

In [ ]:
```
!pip install line_profiler
%load_ext line_profiler
```

In [9]:
```
%lprun -f distance.euclidean [distance.euclidean(p1, p2) for p1, p2 in points_pairs]
```

In [10]:
```
%lprun -f distance.minkowski [distance.euclidean(p1, p2) for p1, p2 in points_pairs]
```

```
                                                         --------
   470                                                   minkowski : double
   471                                                       The Minkowski distance between vectors `u` and `v`.
   472
   473                                                   Examples
   474                                                   --------
   475                                                   >>> from scipy.spatial import distance
   476                                                   >>> distance.minkowski([1, 0, 0], [0, 1, 0], 1)
   477                                                   2.0
   478                                                   >>> distance.minkowski([1, 0, 0], [0, 1, 0], 2)
   479                                                   1.4142135623730951
   480                                                   >>> distance.minkowski([1, 0, 0], [0, 1, 0], 3)
   481                                                   1.2599210498948732
   482                                                   >>> distance.minkowski([1, 1, 0], [0, 1, 0], 1)
   483                                                   1.0
   484                                                   >>> distance.minkowski([1, 1, 0], [0, 1, 0], 2)
   485                                                   1.0
   486                                                   >>> distance.minkowski([1, 1, 0], [0, 1, 0], 3)
   487                                                   1.0
   488
   489                                                   """
   490    100000   1692341.0     16.9     18.8      u = _validate_vector(u)
   491    100000   1292432.0     12.9     14.4      v = _validate_vector(v)
   492    100000     93284.0      0.9      1.0      if p < 1:
   493                                                   raise ValueError("p must be at least 1")
   494    100000    403287.0      4.0      4.5      u_v = u - v
   495    100000     85169.0      0.9      0.9      if w is not None:
   496                                                   w = _validate_weights(w)
   497                                                   if p == 1:
   498                                                       root_w = w
   499                                                   if p == 2:
   500                                                       # better precision and speed
   501                                                       root_w = np.sqrt(w)
   502                                                   else:
   503                                                       root_w = np.power(w, 1/p)
   504                                                   u_v = root_w * u_v
   505    100000   5320230.0     53.2     59.2      dist = norm(u_v, ord=p)
   506    100000     99468.0      1.0      1.1      return dist
```

GIORGIO VINCIGUERRA
PhD student in Computer Science
http://pages.di.unipi.it/vinciguerra/
giorgio.vinciguerra@phd.unipi.it