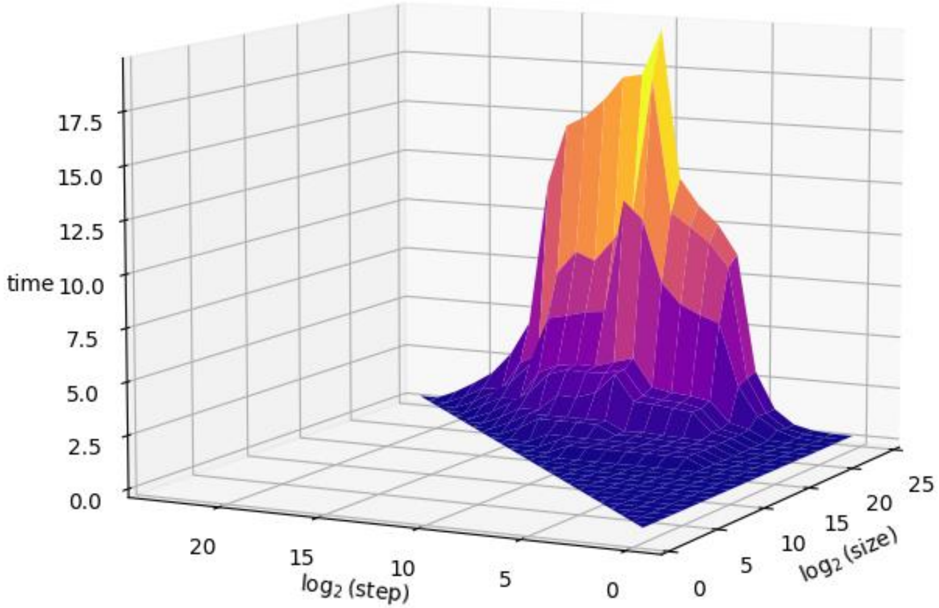
Data Structure Design

Theory and Practice



Gerth Stølting Brodal
Aarhus University
Denmark

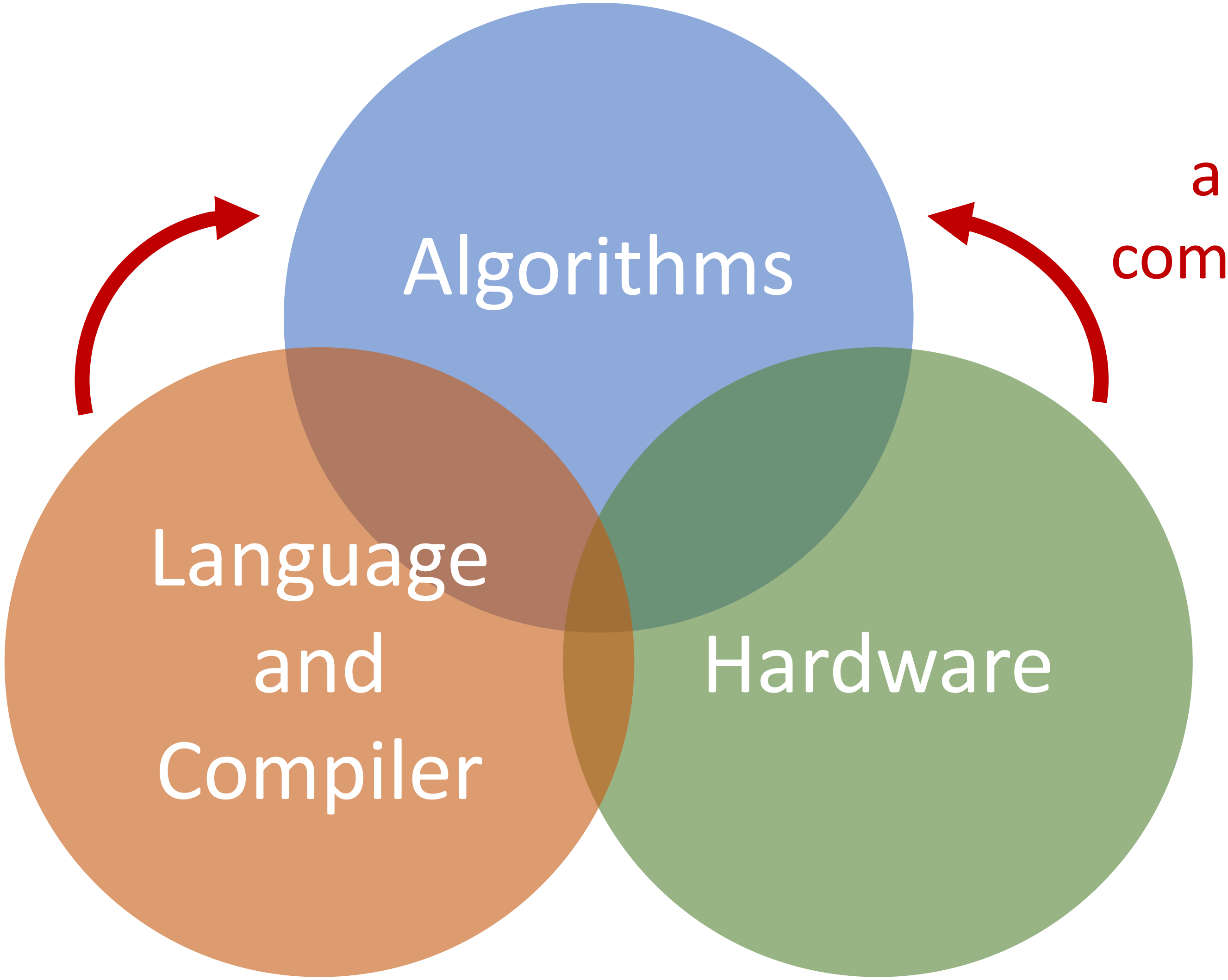
48th International Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 15-19, 2023

Gerth Stølting Brodal

Research
Data structures 1993 –

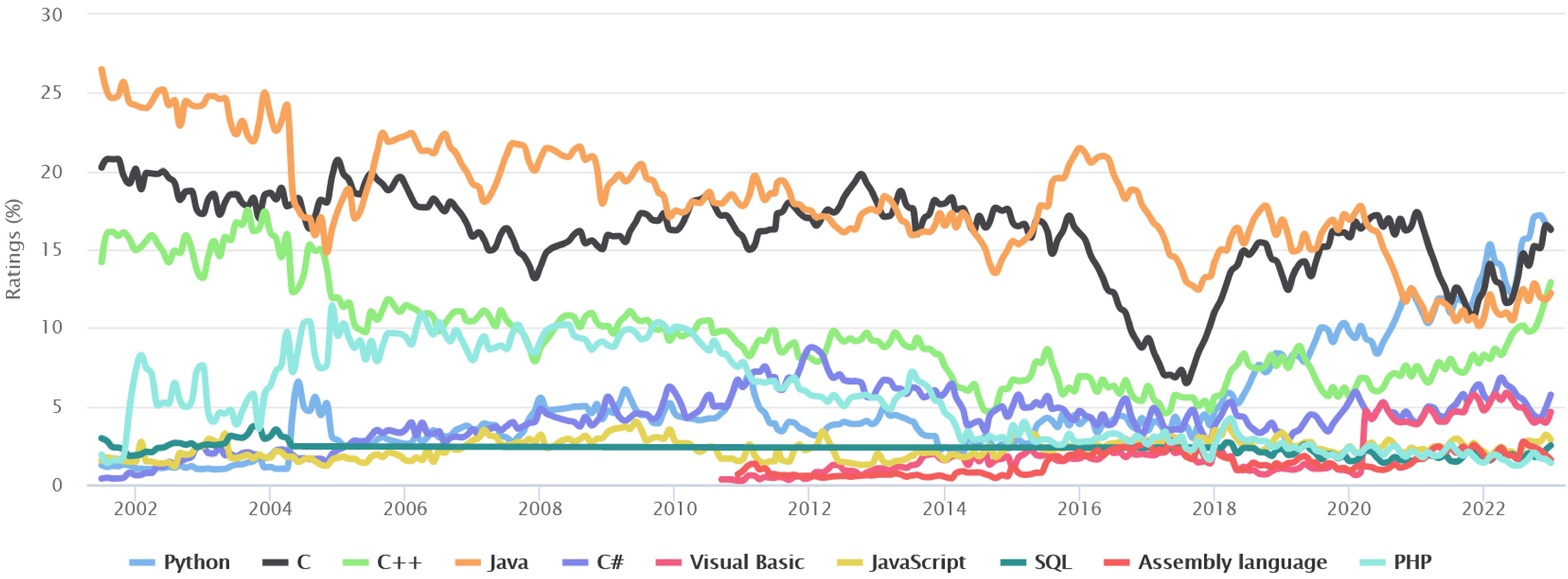
Teaching
Algorithms and Data Structures 2002 –
Introduction to Programming (Python) 2018 –
Bachelor project advising

Efficient Algorithms
= Algorithms + Data structures



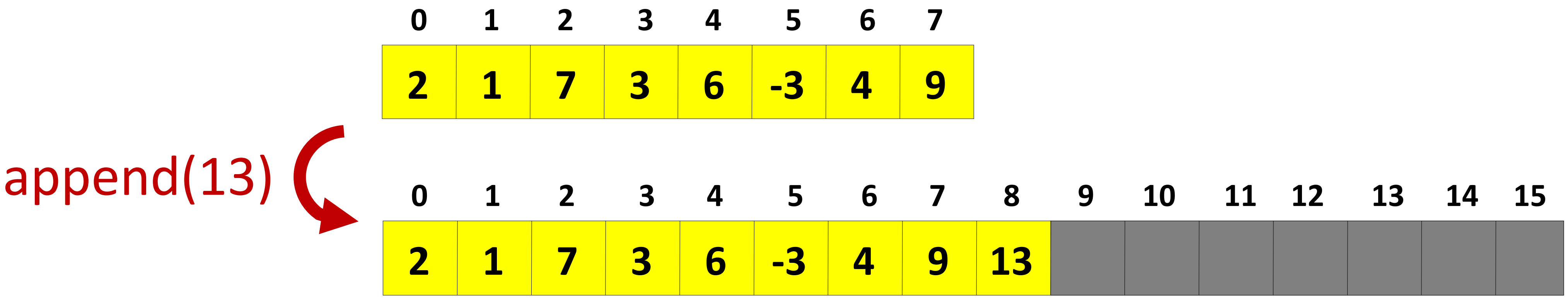
influence on
algorithm design,
computational models,
cost models

TIOBE Programming Community Index



www.tiobe.com/tiobe-index/

Extendable Arrays – Reallocation Strategies



0	1	2	3	4	5	6	7
2	1	7	3	6	-3	4	9

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	1	7	3	6	-3	4	9	13							

C++ vector

+ 100 %

```
size_type
_M_check_len(size_type __n, const char* __s) const
{
    if (max_size() - size() < __n)
        __throw_length_error(__N(__s));

   const size_type __len = size() + (std::max)(size(), __n);
   return (__len < size() || __len > max_size())
       ? max_size()
       : __len;
}
```

Java ArrayList

+ 50 %

```
private int newCapacity(int minCapacity) {
    // overflow-conscious code
    int oldCapacity = elementData.length;
    int newCapacity = oldCapacity + (oldCapacity >> 1);
    if (newCapacity - minCapacity <= 0) {
        if (elementData == DEFAULTCAPACITY_EMPTY_ELEMENTDATA)
            return Math.max(DEFAULT_CAPACITY, minCapacity);
        if (minCapacity < 0) // overflow
            throw new OutOfMemoryError();
        return minCapacity;
    }
    return (newCapacity - MAX_ARRAY_SIZE <= 0)
        ? newCapacity
        : hugeCapacity(minCapacity);
}
```

Python list

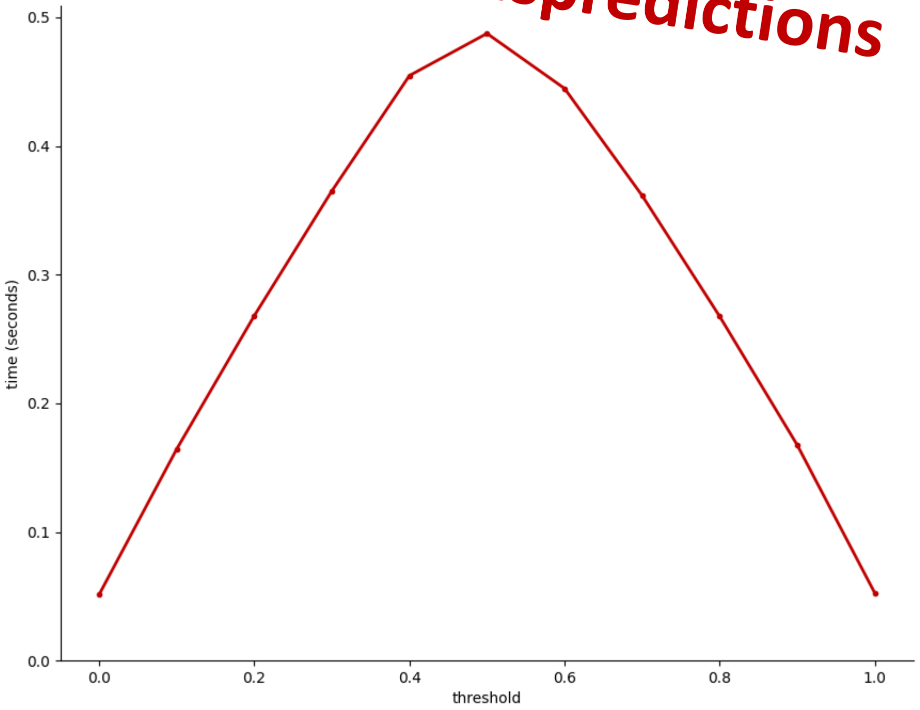
+ 12.5 %

```
static int
list_resize(PyListObject *self, Py_ssize_t newsize)
{
    PyObject **items;
    size_t new_allocated, num_allocated_bytes;
    Py_ssize_t allocated = self->allocated;
    if (allocated >= newsize && newsize >= (allocated >> 1)) {
        assert(self->ob_item != NULL || newsize == 0);
        Py_SIZE(self) = newsize;
        return 0;
    }
    new_allocated =
      (size_t)newsize + (newsize >> 3) + (newsize < 9 ? 3 : 6);
    …
}
```

Branches

```
for (int i=0; i < size; i++)
  if (A[i] <= threshold)
    small ++;
```

threshold	time (seconds)
0.0	0.045
0.5	0.458
1.0	0.046

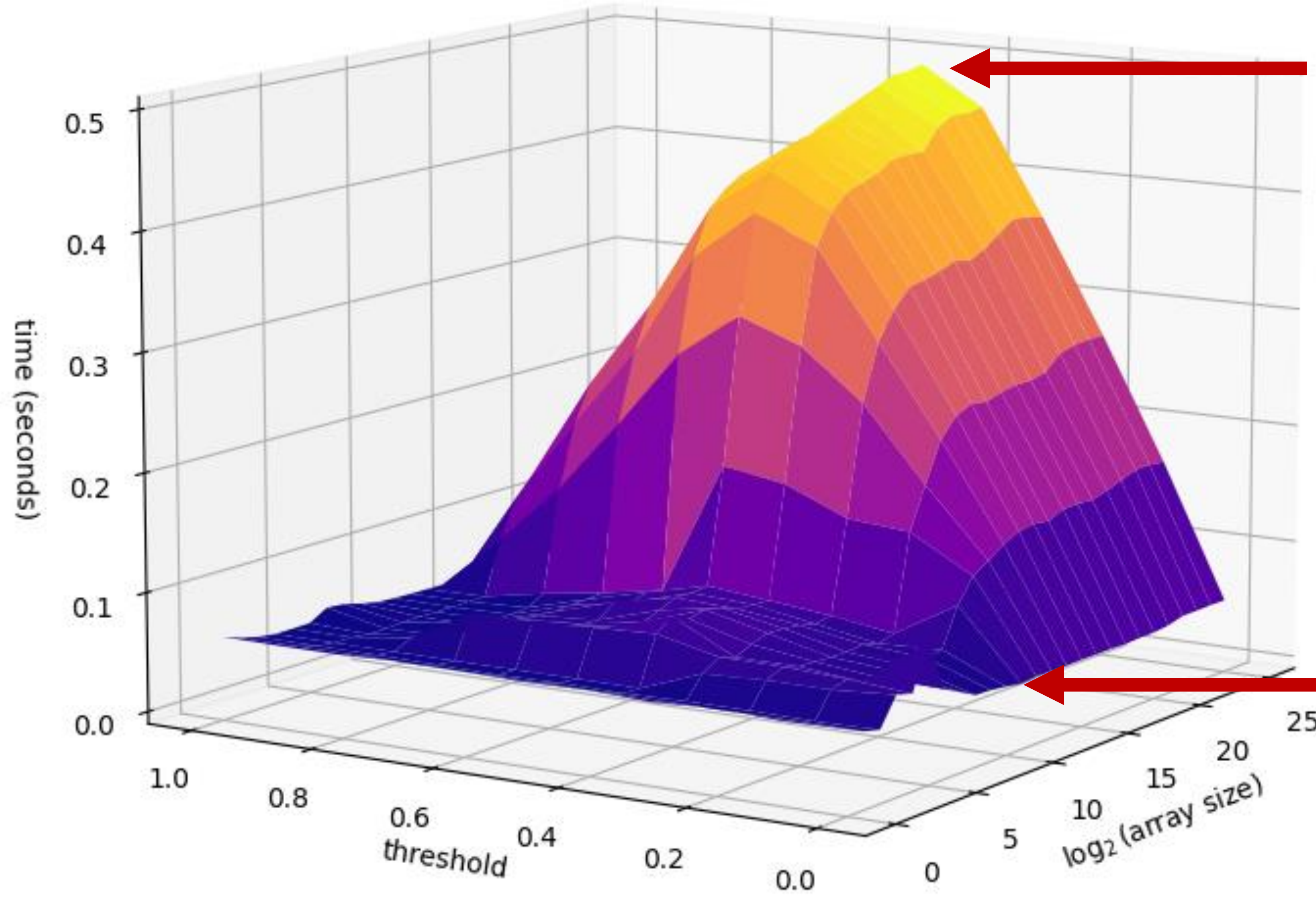


Branch mispredictions


A

random floats in range [0, 1]

Threshold Counting

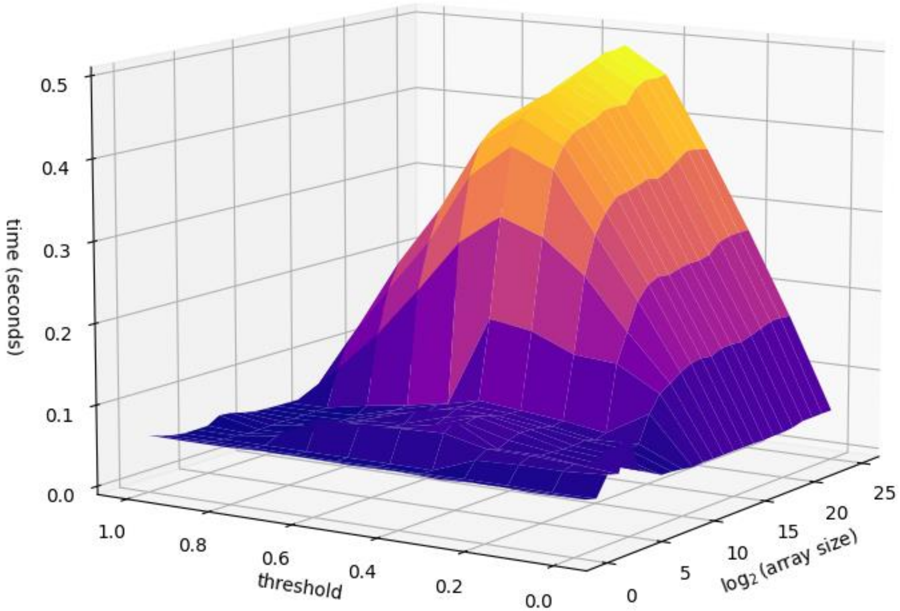


symmetric,
predict as majority

size of branch
prediction table

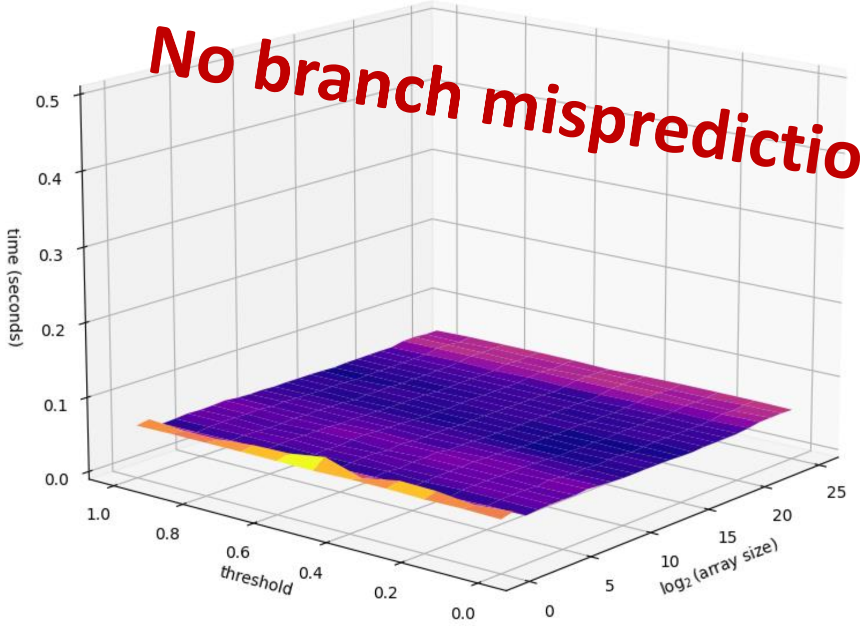
11th Gen Intel Core i7-1165G7 @ 2.80GHz, Windows 10 + cygwin, gcc –O2, performed 2^{27} comparisons (repeatedly ran over array)

```
for (int i=0; i < size; i++)
    if (A[i] <= threshold)
        small ++;
```



```
.L7: comiss (%rax), %xmm6
     jb     .L5
     addq   $1, %r14
.L5: addq   $4, %rax
     cmpq   %rbx, %rax
     jne    .L7
```

gcc -O2 -fno-if-conversion -fno-if-conversion2

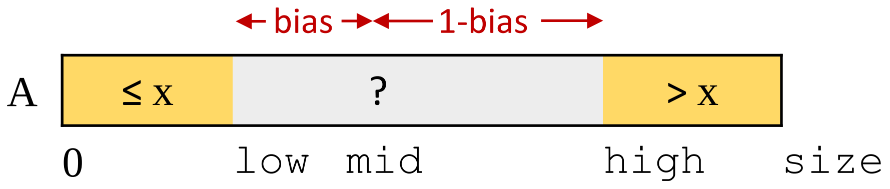


No branch mispredictions

```
.L7: comiss (%rax), %xmm6
     sbbq   $-1, %r14
     addq   $4, %rax
     cmpq   %rbx, %rax
     jne    .L7
```

gcc -O2

Binary Search



← bias →← 1-bias →

A

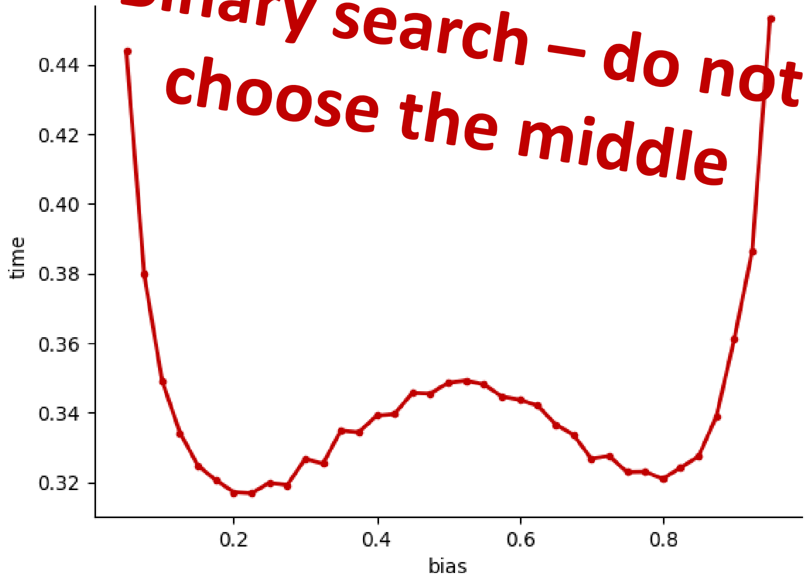
≤ x

?

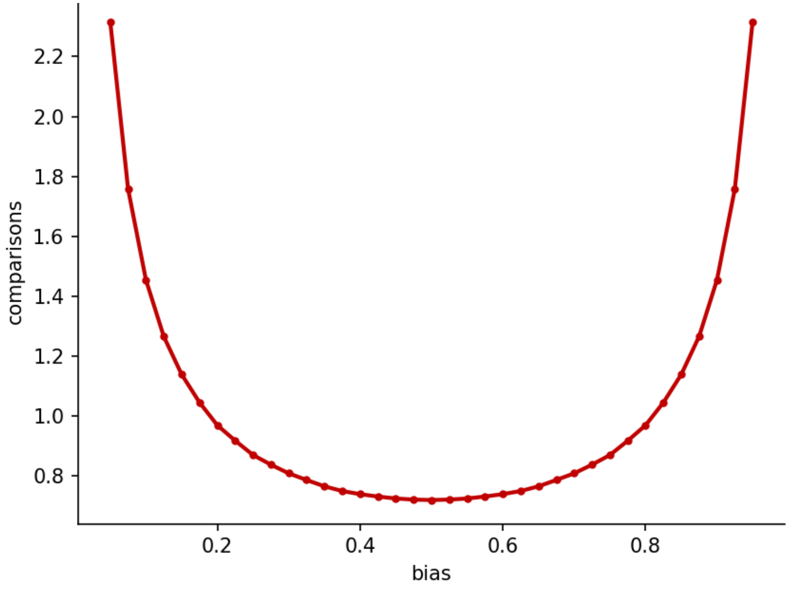
> x

0 low mid high size

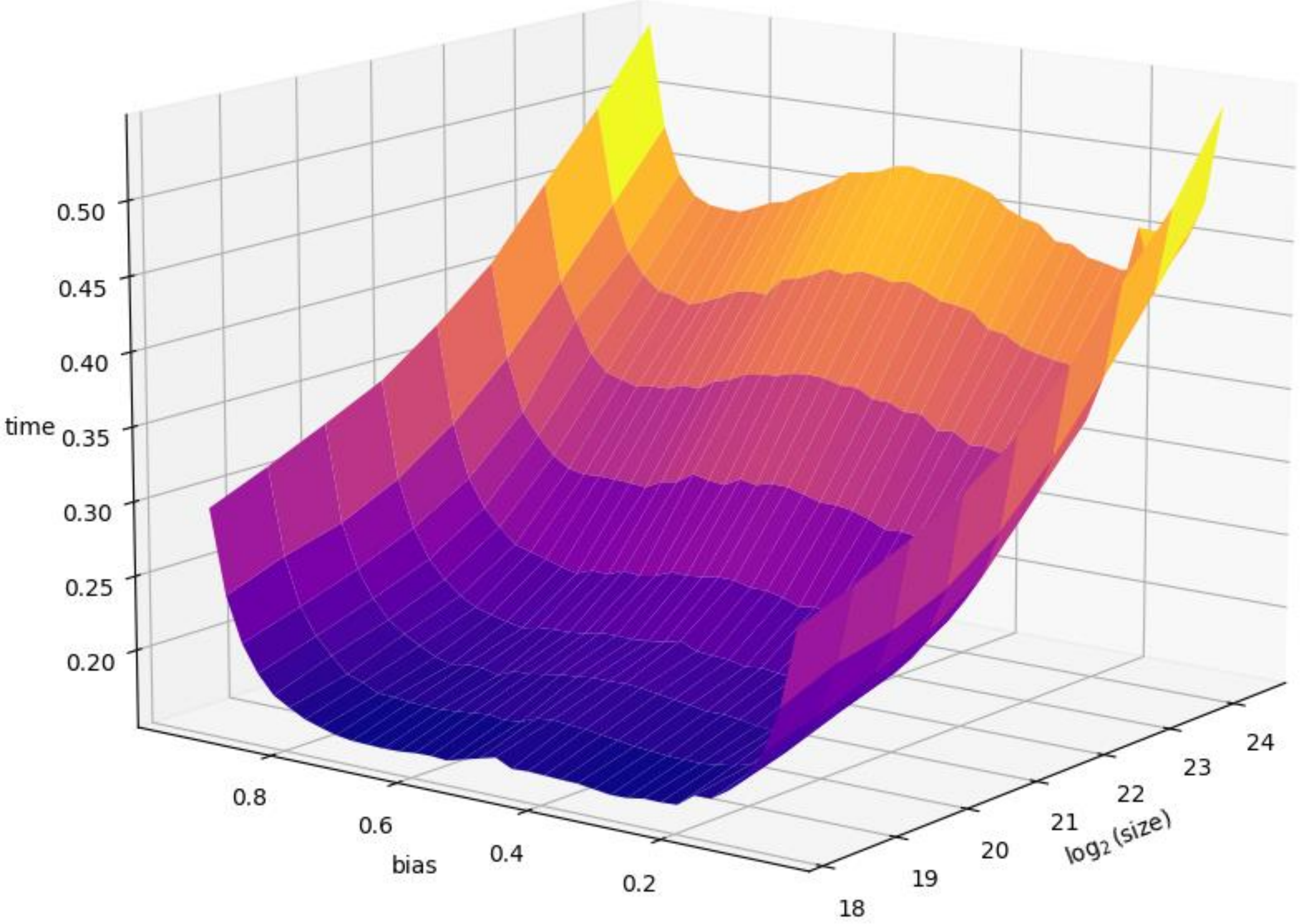
```
int low = 0, high = size;
while (low < high) {
int mid = low+(int)((high-low)*bias);
   if (A[mid] <= x)
      low = mid + 1;
   else
      high = mid;
}
```



Binary search – do not choose the middle



Binary Search

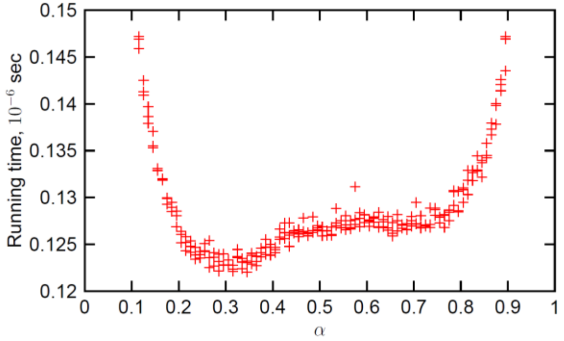


Summary Branch Mispredictions

- Mispredictions can slow done programs by a factor 10
- **Binary search** faster with biased pivot
- **Binary search trees** faster with biased pivots
  [B. and Moruz, ESA 2004]
- **QuickSort** faster with biased pivot
  [Kaligosi and Sanders, ESA 2006]
  – also analyzed different prediction models
- **InsertionSort** O(n^2) comparisons but O(n) mispredictions
- **MergeSort** with InsertionSort for small problems
  (used in standard libraries)
- **Sorting** [B. and Moruz, WADS 2005]
  O($d \cdot n \cdot \log n$) comparisons ⇒ $\Omega(n \cdot \log_d n)$ mispredictions

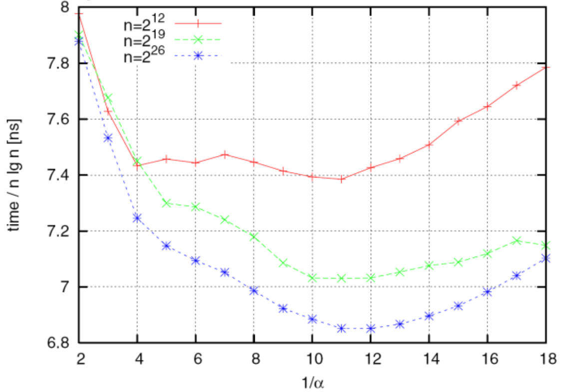
Binary search trees

[B. and Moruz, ESA 2004]



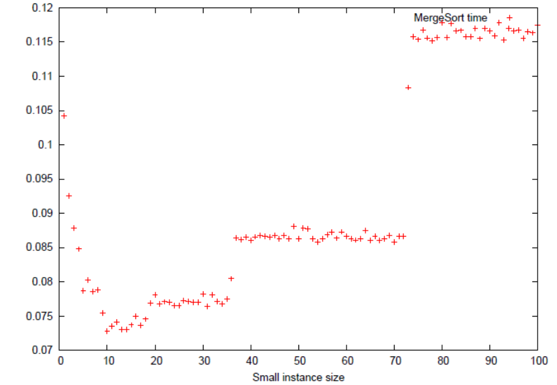
QuickSort

[Kaligosi and Sanders, ESA 2006]

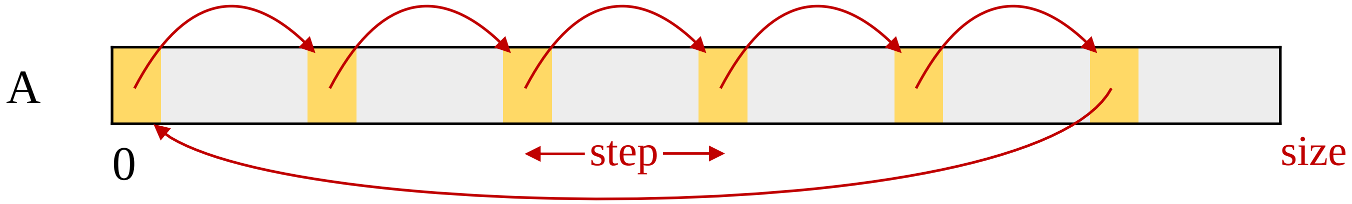


MergeSort

[Moruz]



Pointer Chasing



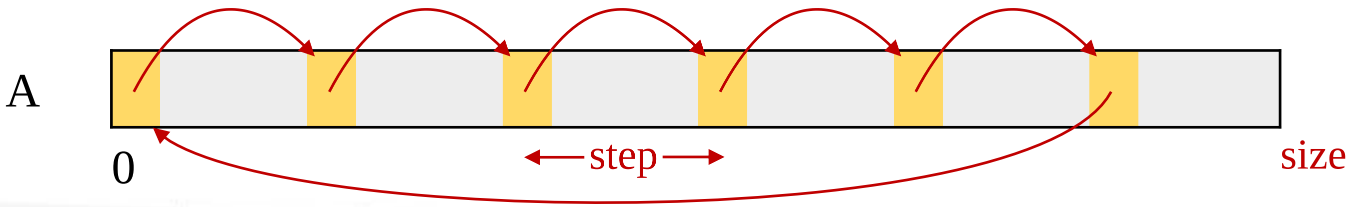
```
position = 0;
for (int i=0; i < iterations; i++)
    position = A[position];
```

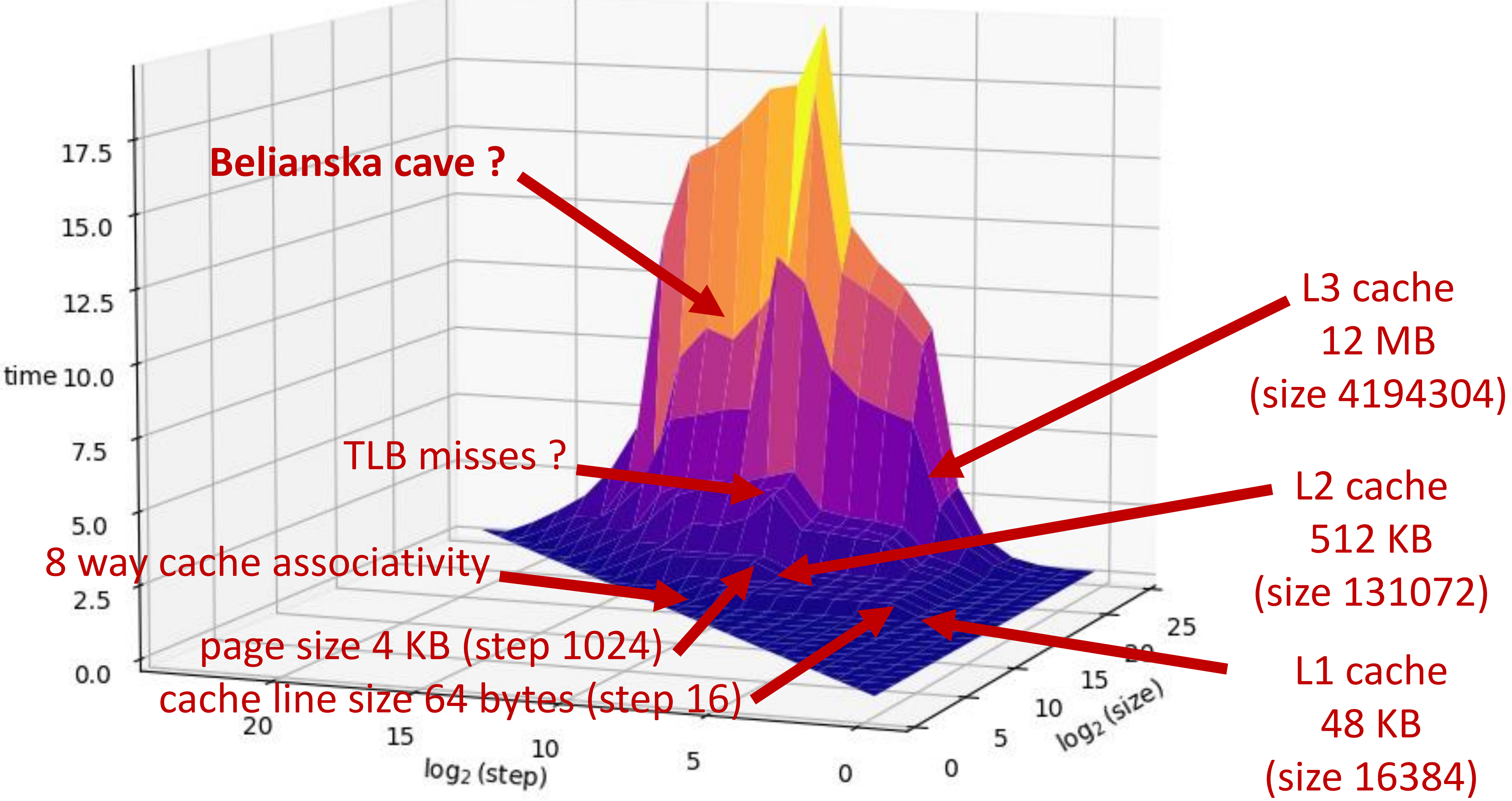
step	time (seconds)
1	0.297
1024	19.5

x 66

(size = 16777216)

Pointer Chasing





Belianska cave ?

TLB misses ?

8 way cache associativity

page size 4 KB (step 1024)

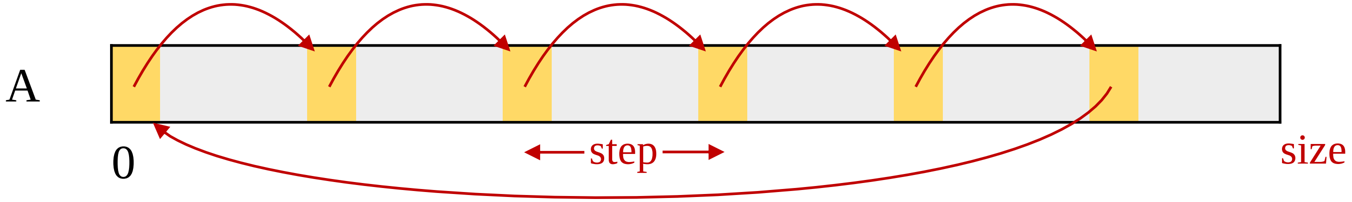
cache line size 64 bytes (step 16)

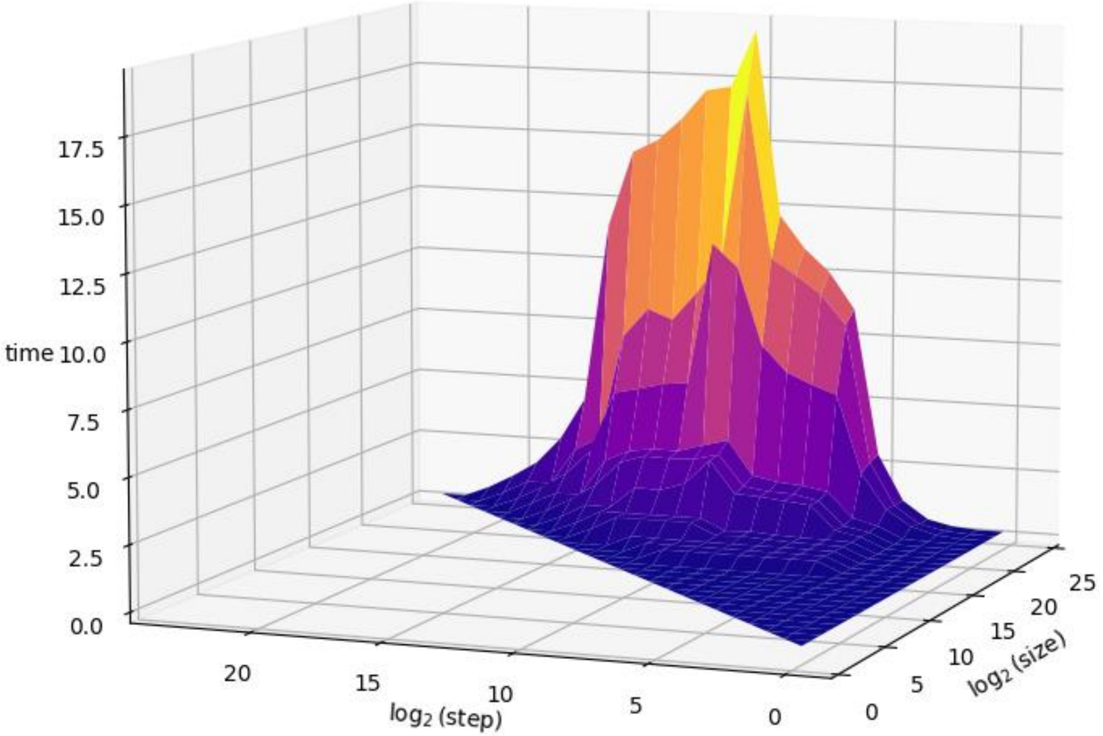
L3 cache
12 MB
(size 4194304)

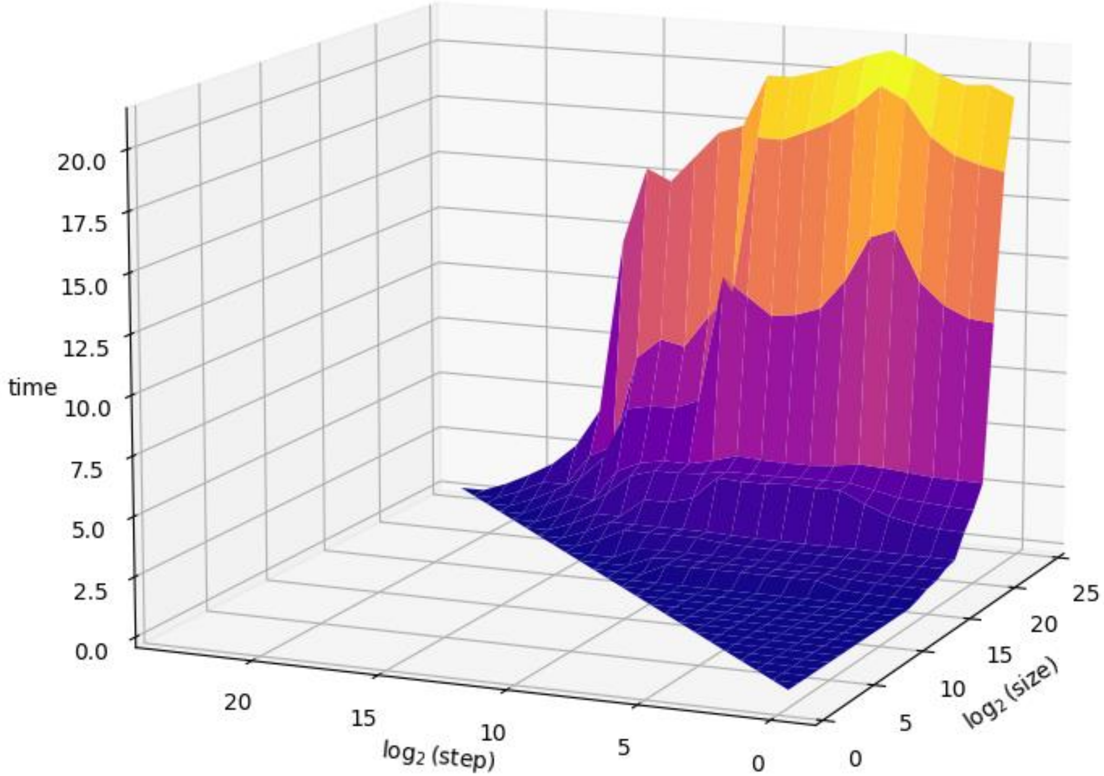
L2 cache
512 KB
(size 131072)

L1 cache
48 KB
(size 16384)

Pointer Chasing

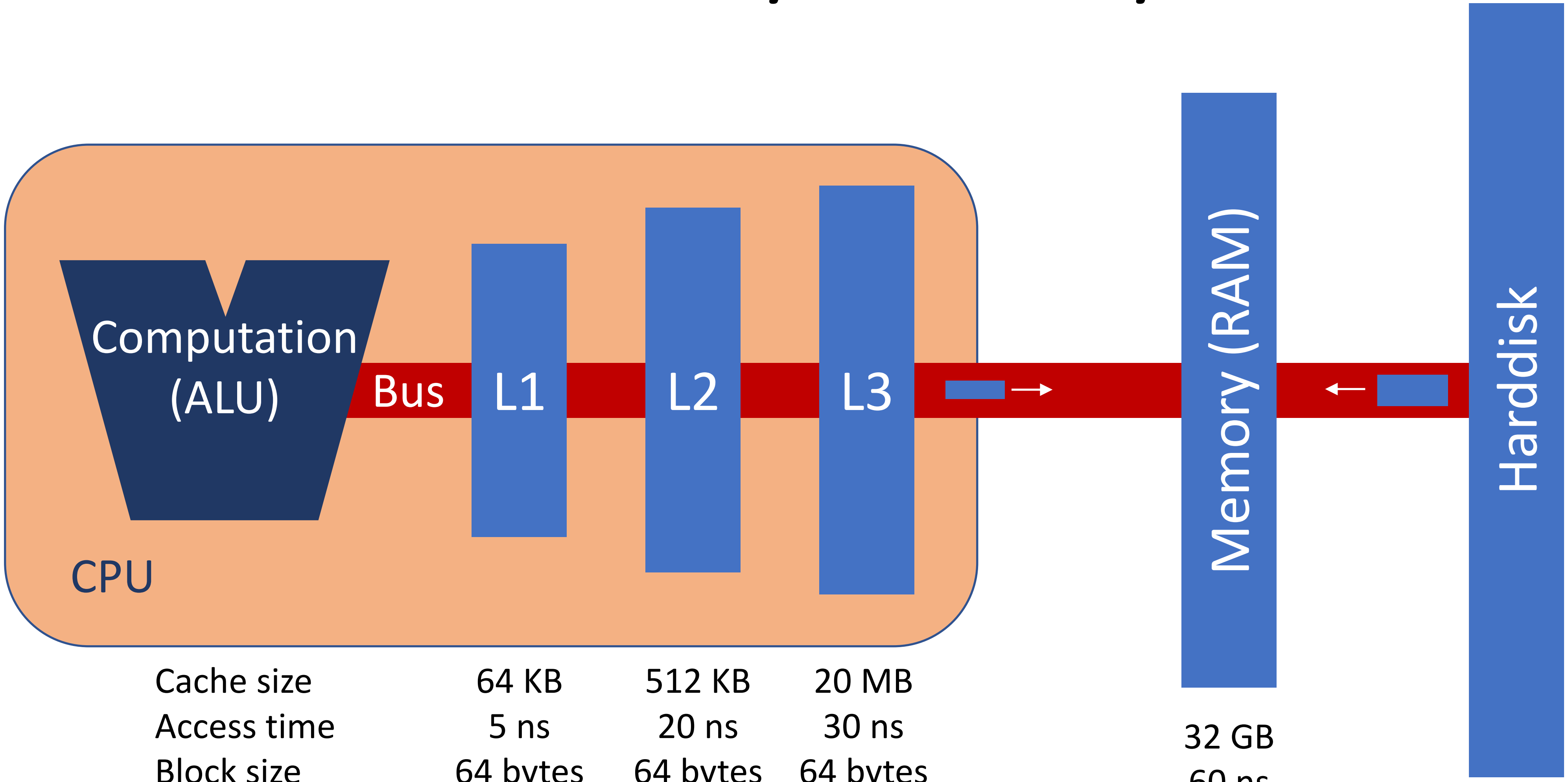






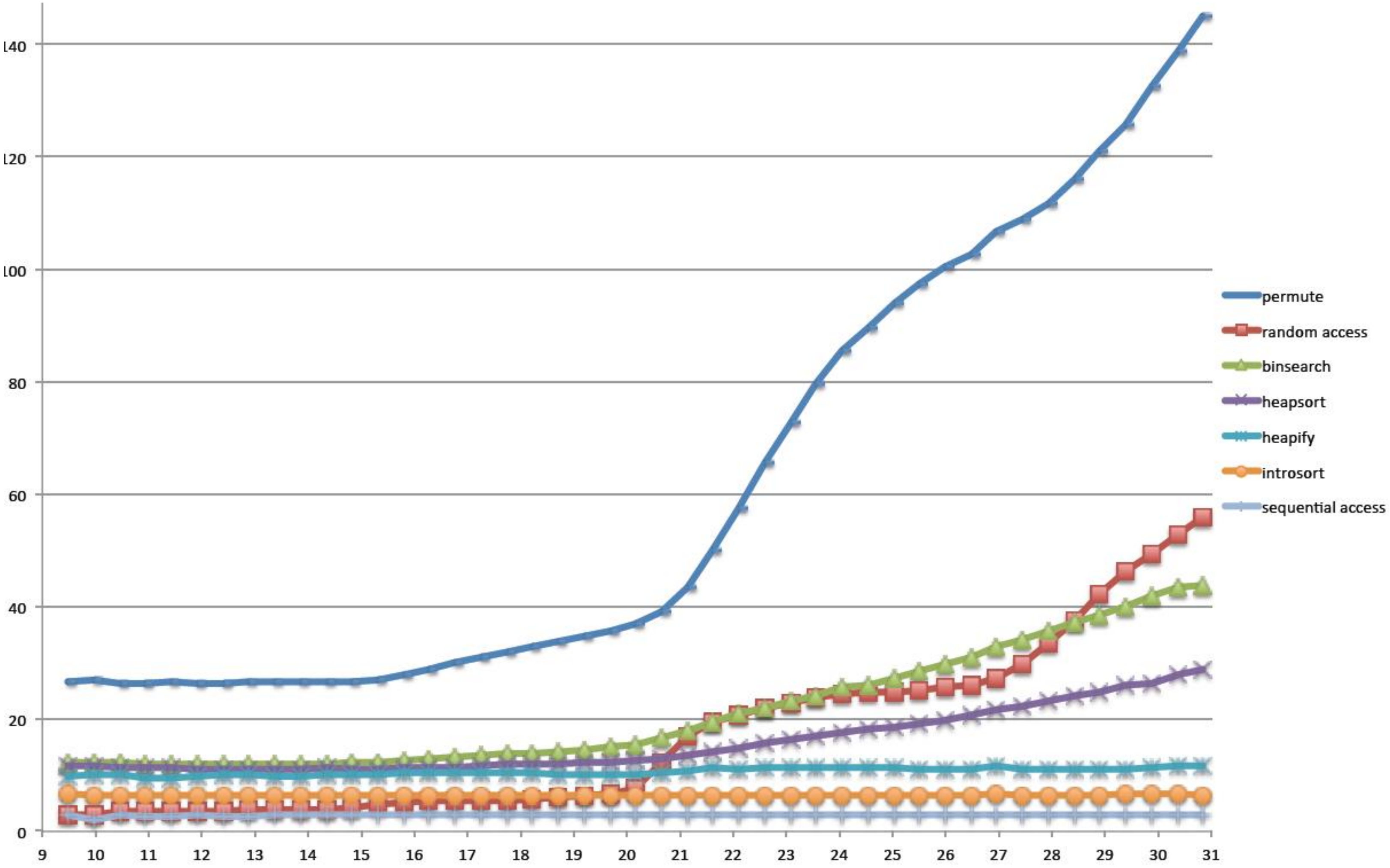
randomly permute pointer cells

Memory Hierarchy



Cache size	64 KB	512 KB	20 MB	32 GB	1 TB
Access time	5 ns	20 ns	30 ns	60 ns	5.000.000 ns
Block size	64 bytes	64 bytes	64 bytes	64 bytes	4.096 bytes

Cost of Address Translation



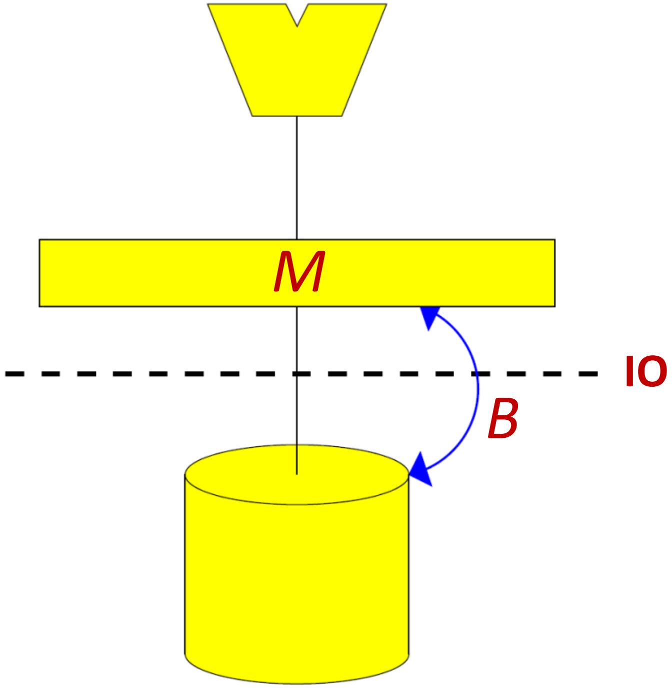
Time / RAM complexity

log *n*

[Jurkiewicz, Mehlhorn, *The cost of address translation*, ALENEX 2013]

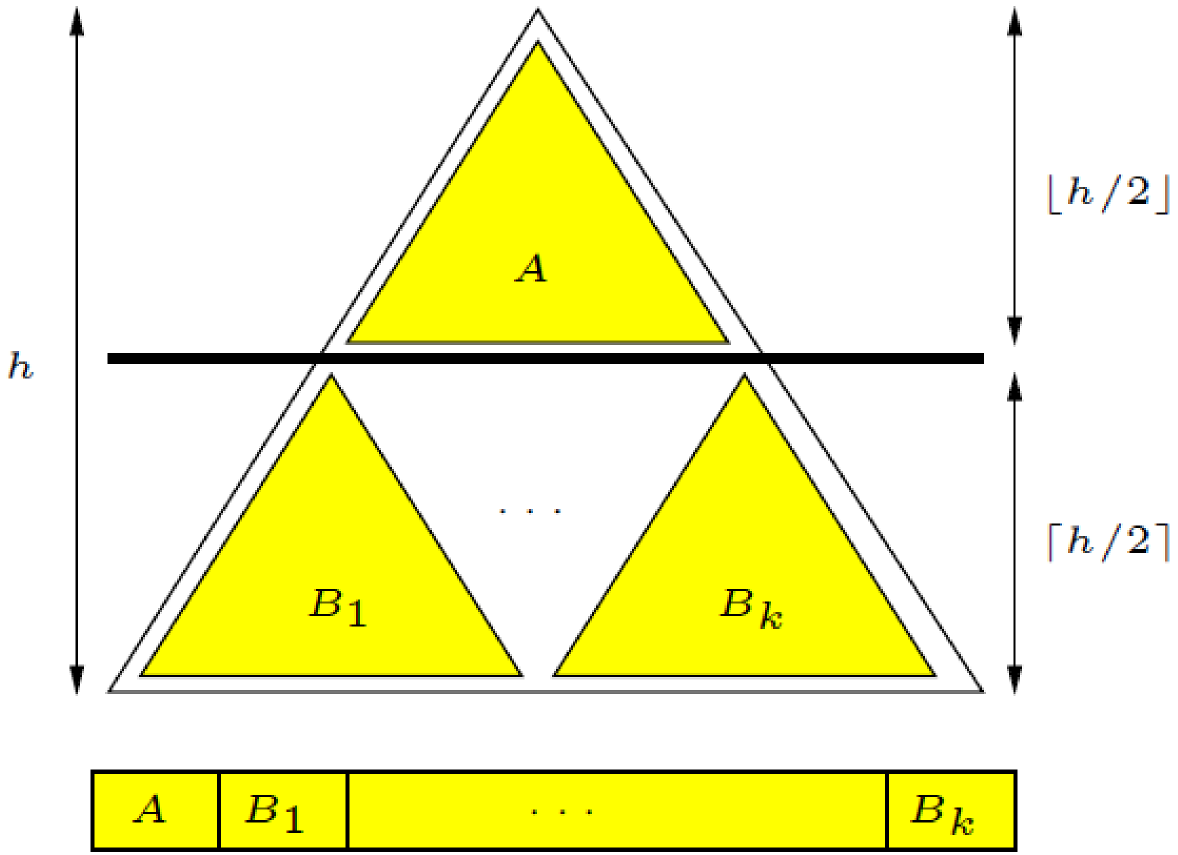
External Memory and Cache-Oblivious Models

- External memory model parameters B and M
- Scanning O(N/B) IOs
- Sorting O($N/B\cdot\log_{M/B}(N/B)$) IOs
- Searching O($\log_B N$) IOs
- Cache oblivious model is like external memory model ... but algorithms do not know B and M (assume optimal cache replacement strategy)
- Optimal on all memory levels (under some assumptions)

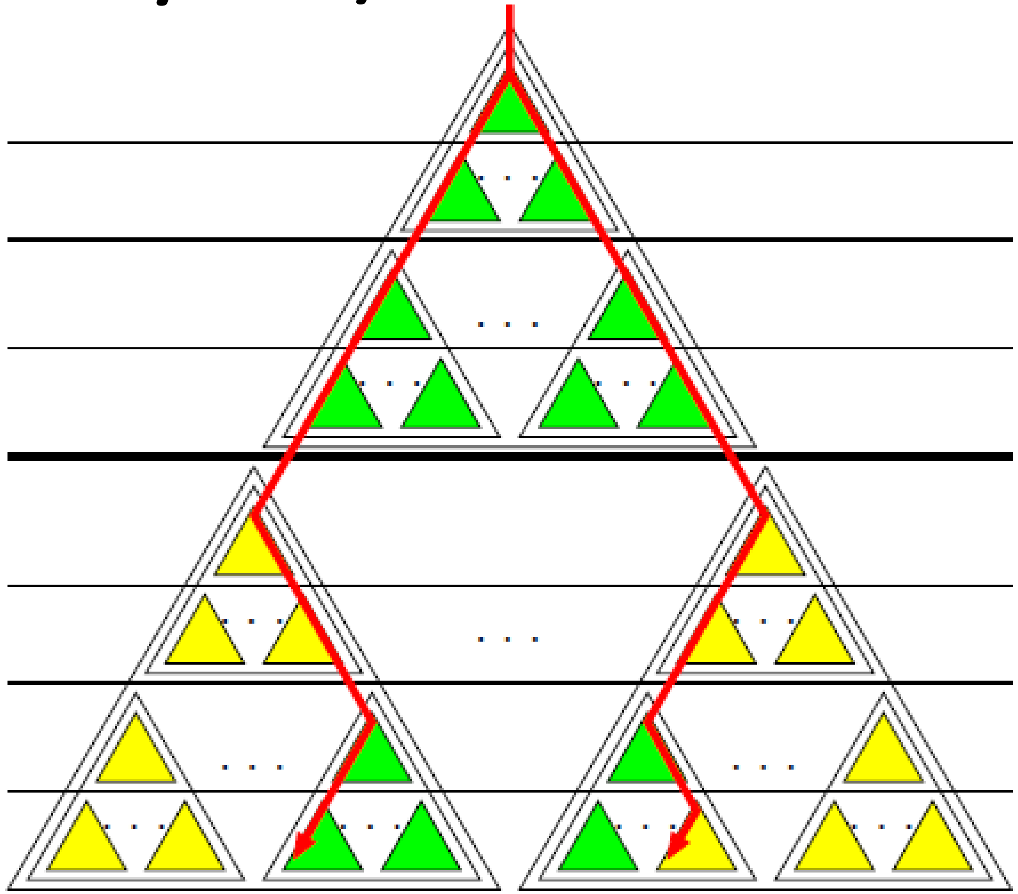


[Aggarwal, Vitter, *The input/output complexity of sorting and related problems*, 1988]
[Frigo, Leiserson, Prokop, Ramachandran, *Cache-Oblivious Algorithms*, 1999]

Recursive Tree Layout
(van Emde Boas layout)



Binary tree

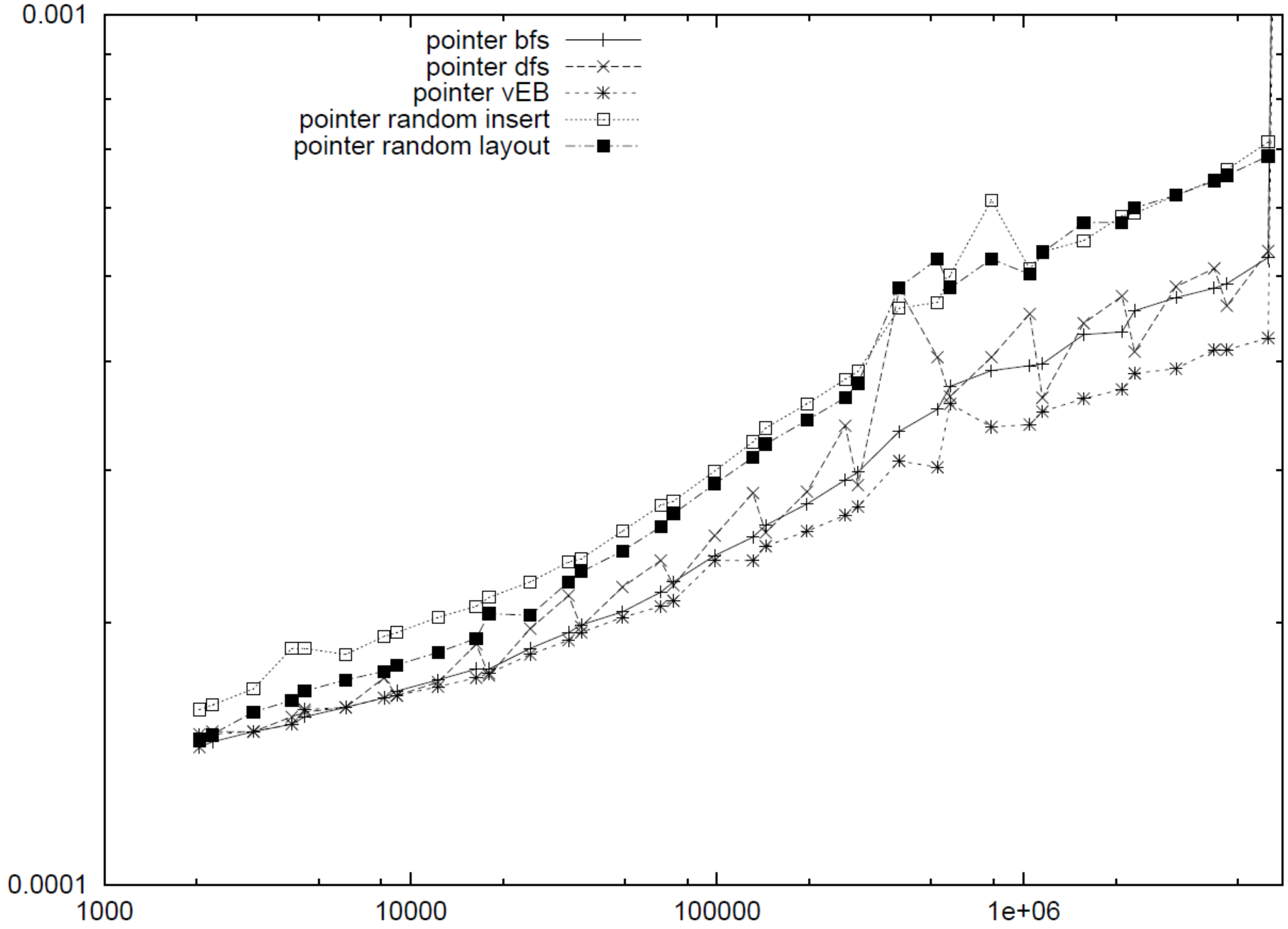


Search $O(\log_B N)$ IOs

Range Searches $O(\log_B N + k/B)$ IOs

[Prokop, MIT MSc thesis *Cache-Oblivious Algorithms*, 1999]

Random Searches in Perfectly Balanced Search Trees



random layouts

van Emde Boas Layout

[B., Fagerberg, Jacob, *Cache-Oblivious Search Trees via Binary Trees of Small Height*, SODA 2002]

No Balanced Search Trees in Python ?

- Python standard library does not contain balanced search trees
- insert_left inserts into a sorted list [binary search $O(\log n)$ + memcopy $O(n)$]

Python shell

```
> L = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
> bisect.insort_left(L, 42)
> print(L)
| [0, 10, 20, 30, 40, 42, 50, 60, 70, 80, 90]
```

- SortedList in sortedcollections essentially combines list-of-lists with bisect

[[1, 5, 15, 28], [35, 38, 38, 41, 44], [46, 61, 63], [70, 87, 89]]

updates $O(\sqrt{n})$ and queries $O(\log n)$

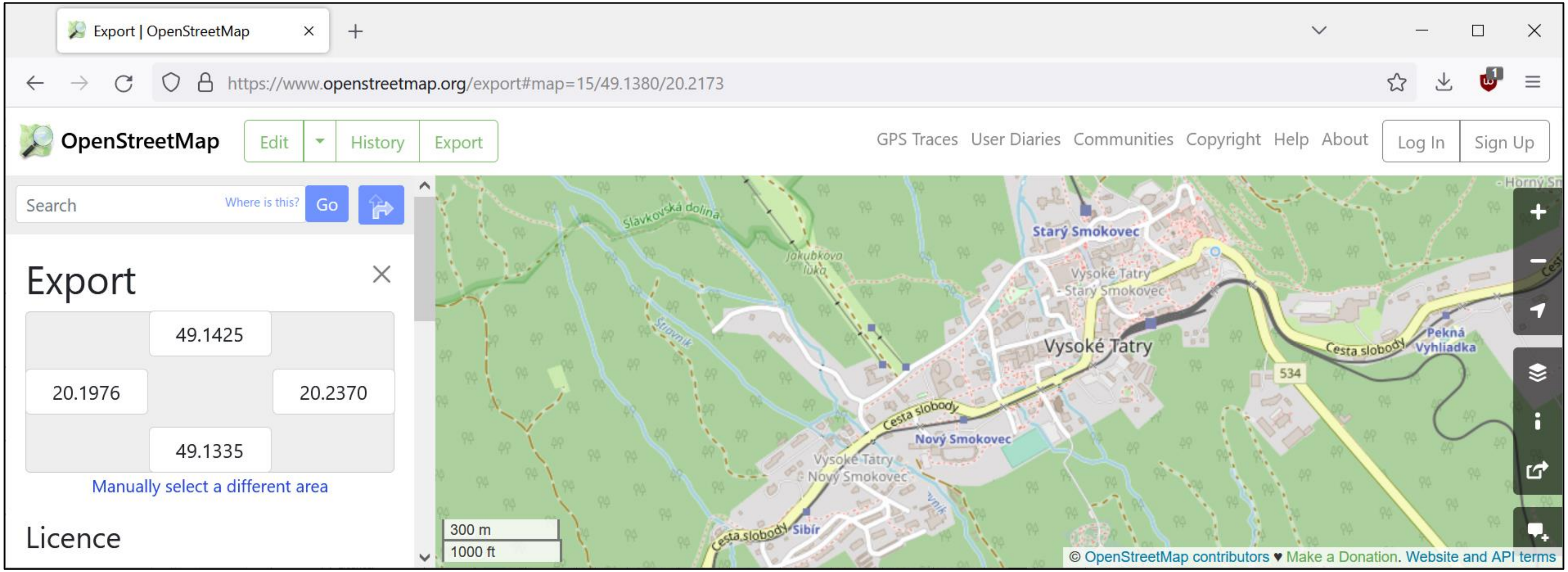
grantjenks.com/docs/sortedcollections/

Summary Hardware Influence

- Random Access Machine (RAM) model great for designing and analyze algorithms
- … but final program performance depends on hardware
- Have an idea of what the bottleneck is in your computation and choose an appropriate abstract model

An Unexpected Journey

- Bachelorproject = shortest paths on Open Street Map graphs
- Students have trouble implementing Dijkstra's algorithm in Java^{TM}



https://www.openstreetmap.org/export#map=15/49.1380/20.2173

Export

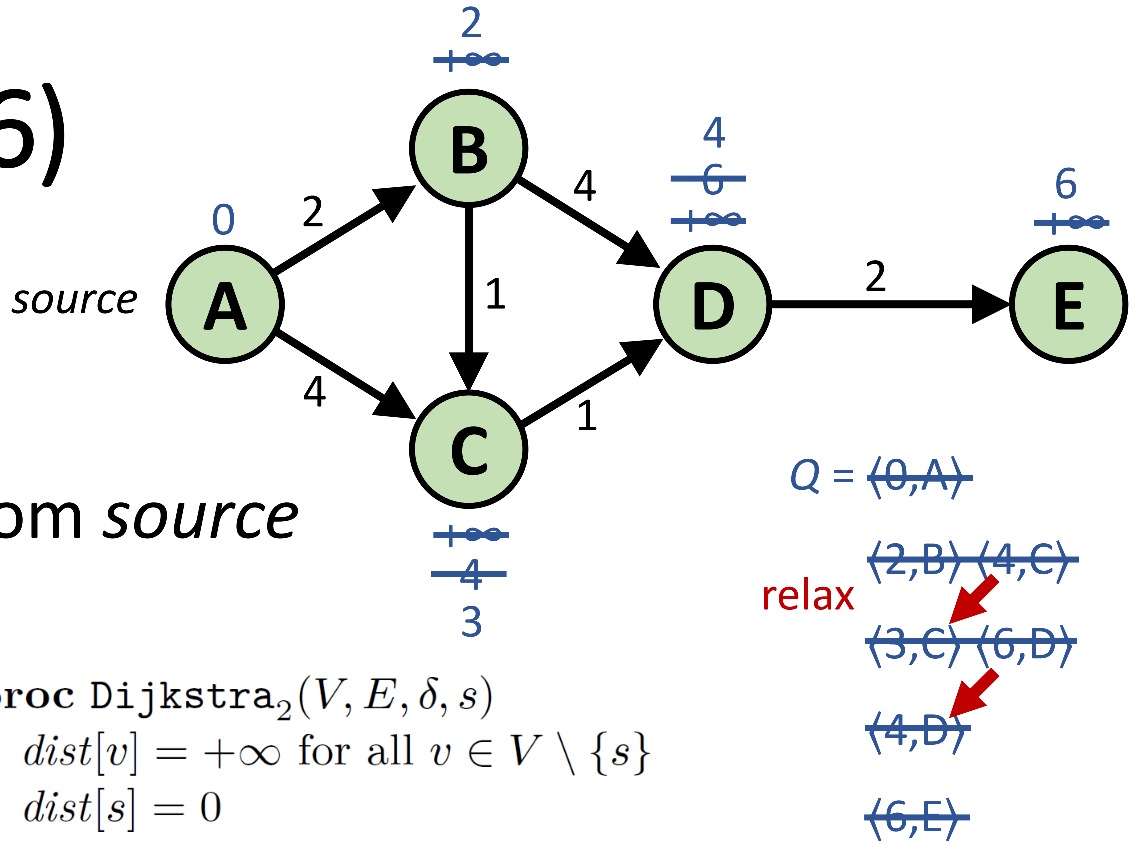
	49.1425	
20.1976		20.2370
	49.1335	

Manually select a different area

Licence

Dijkstra’s Algorithm (1956)

- Non-negative edge weights
- Visits nodes in increasing distance from *source*



```
proc Dijkstra$_1(V, E, \delta, s)$
    $dist[v] = +\infty$ for all $v \in V \setminus \{s\}$
    $dist[s] = 0$
    Insert$(Q, \langle dist[s], s\rangle)$
    while $Q \neq \emptyset$ do
        $\langle d, u\rangle$ = ExtractMin$(Q)$
        for $(u, v) \in E \cap (\{u\} \times V)$ do
            if $dist[u] + \delta(u, v) < dist[v]$ then
                $dist[v] = dist[u] + \delta(u, v)$
                if $v \in Q$ then
                    DecreaseKey$(Q, v, dist[v])$
                else
                    Insert$(Q, \langle v, dist[v]\rangle)$
    return $dist$
```

relax

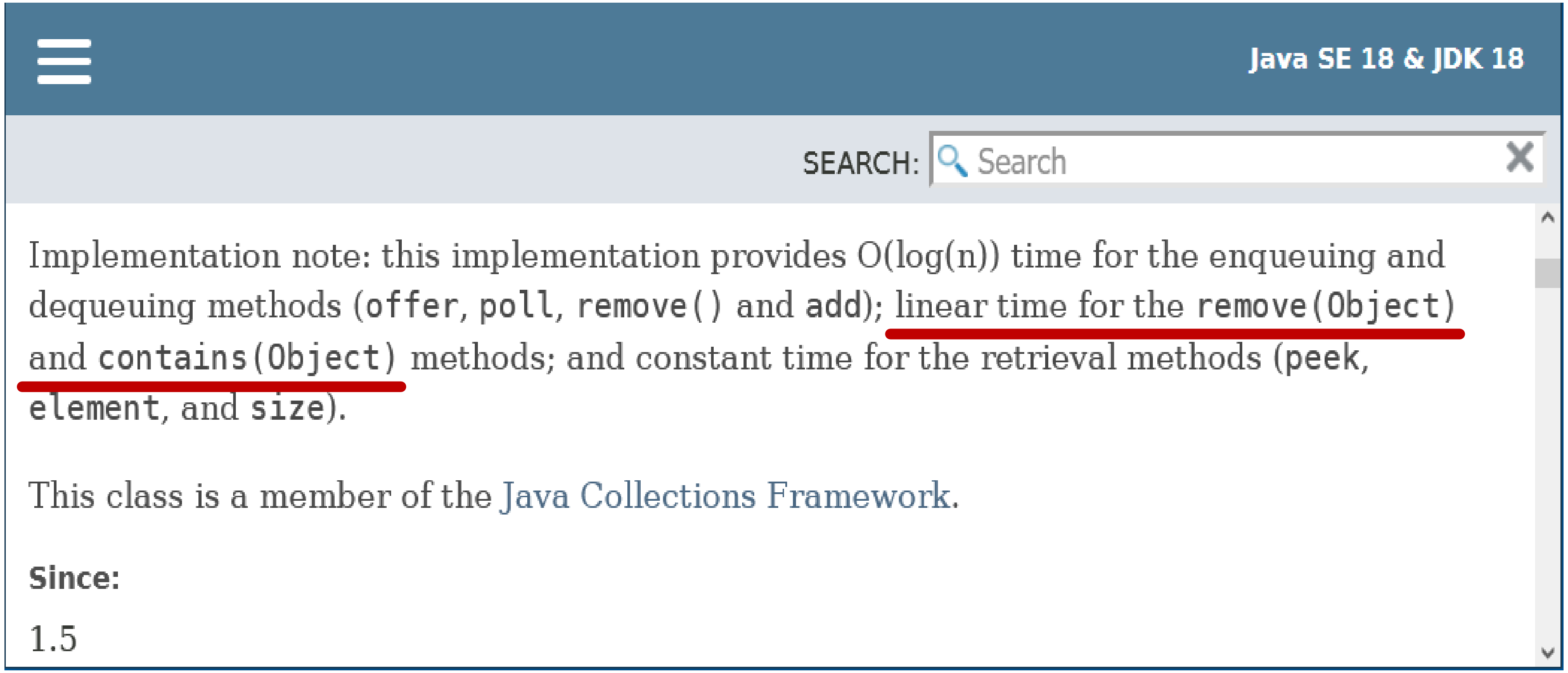
Fibonacci heaps
(Fredman, Tarjan 1984)
⇒ O(m + n · log n)

```
proc Dijkstra$_2(V, E, \delta, s)$
    $dist[v] = +\infty$ for all $v \in V \setminus \{s\}$
    $dist[s] = 0$
    Insert$(Q, \langle dist[s], s\rangle)$
    while $Q \neq \emptyset$ do
        $\langle d, u\rangle$ = ExtractMin$(Q)$
        for $(u, v) \in E \cap (\{u\} \times V)$ do
            if $dist[u] + \delta(u, v) < dist[v]$ then
                $dist[v] = dist[u] + \delta(u, v)$
                if $v \in Q$ then
                    Remove$(Q, v)$
                Insert$(Q, \langle dist[v], v\rangle)$
    return $dist$
```

O(log n) `Remove`
⇒ O(m · log n)

The Challenge - Java's Builtin Binary Heap

- No `decreasekey`
- `remove` O(n) time ⇒ Dijkstra O($m \cdot n$)



Implementation note: this implementation provides O(log(n)) time for the enqueuing and dequeuing methods (`offer`, `poll`, `remove()` and `add`); linear time for the `remove(Object)` and `contains(Object)` methods; and constant time for the retrieval methods (`peek`, `element`, and `size`).

This class is a member of the Java Collections Framework.

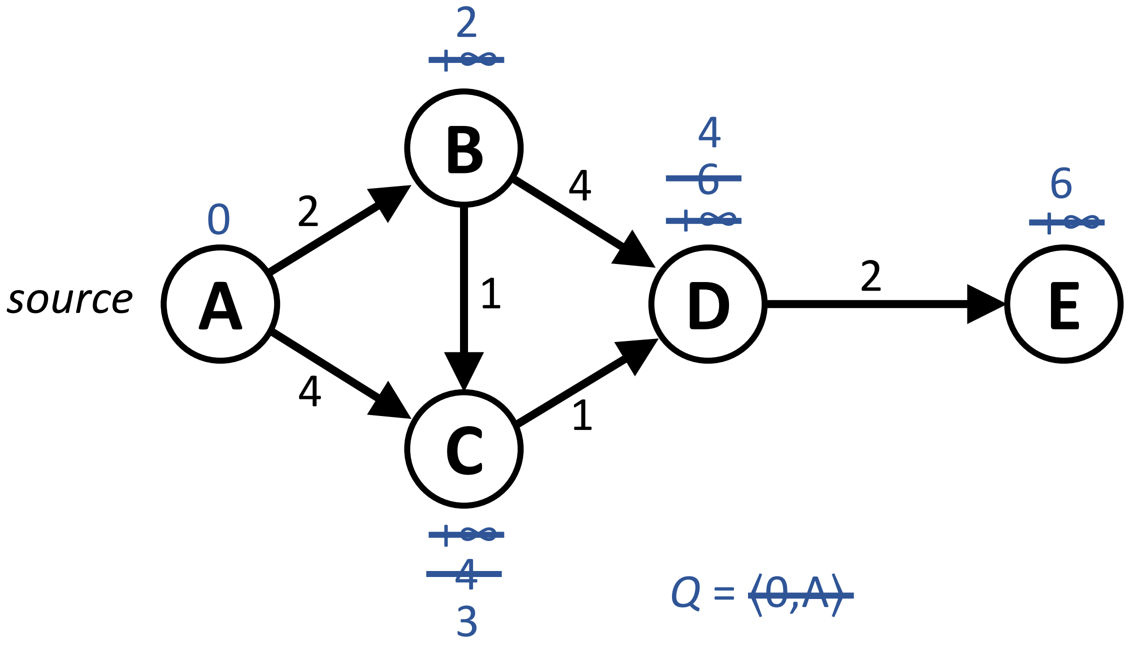
Since:

1.5

Repeated Insertions

- Relax inserts new copies of item
- Skip outdated items

```
proc Dijkstra$_3(V, E, \delta, s)$
    $dist[v] = +\infty$ for all $v \in V \setminus \{s\}$
    $dist[s] = 0$
    Insert$(Q, \langle dist[s], s\rangle)$
    while $Q \neq \emptyset$ do
        $\langle d, u\rangle$ = ExtractMin$(Q)$
outdated ? →    if $d = dist[u]$ then
            for $(u, v) \in E \cap (\{u\} \times V)$ do
                if $dist[u] + \delta(u, v) < dist[v]$ then
                    $dist[v] = dist[u] + \delta(u, v)$
relax
= reinsert →        Insert$(Q, \langle dist[v], v\rangle)$
    return $dist$
```



Q = ~~⟨0,A⟩~~

~~⟨2,B⟩ ⟨4,C⟩~~

~~⟨3,C⟩ ⟨4,C⟩ ⟨6,D⟩~~

~~⟨4,C⟩ ⟨4,D⟩ ⟨6,D⟩~~

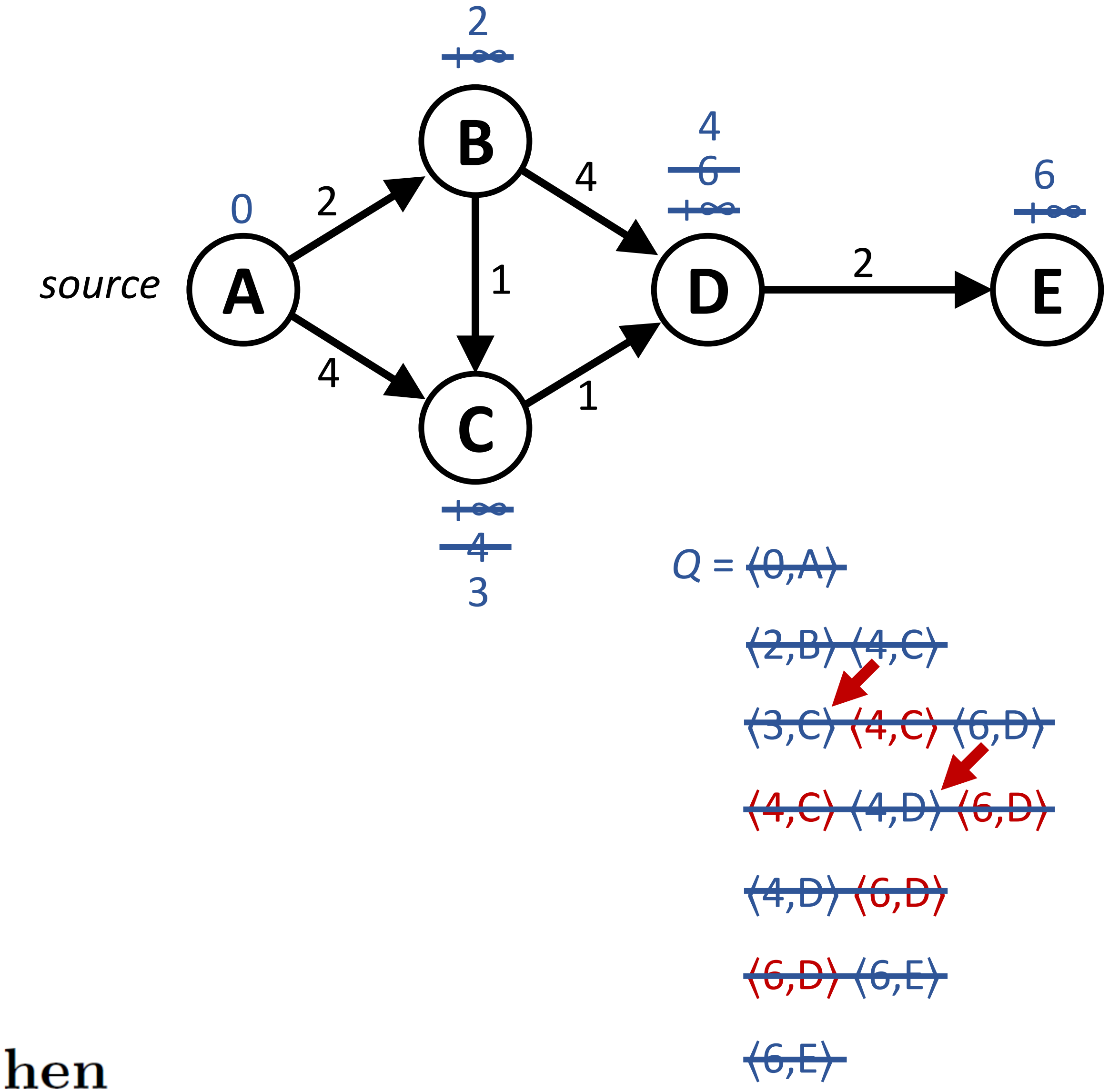
~~⟨4,D⟩ ⟨6,D⟩~~

~~⟨6,D⟩ ⟨6,E⟩~~

~~⟨6,E⟩~~

Using a Visited Set

```
proc Dijkstra_4$(V, E, \delta, s)$
    $dist[v] = +\infty$ for all $v \in V \setminus \{s\}$
    $dist[s] = 0$
    $visited = \emptyset$
    Insert$(Q, \langle dist[s], s\rangle)$
    while $Q \neq \emptyset$ do
        $\langle d, u\rangle$ = ExtractMin$(Q)$
        if $u \notin visited$ then          ← bitvector
            $visited = visited \cup \{u\}$
            for $(u, v) \in E \cap (\{u\} \times V)$ do
                if $dist[u] + \delta(u, v) < dist[v]$ then
                    $dist[v] = dist[u] + \delta(u, v)$
                    Insert$(Q, \langle dist[v], v\rangle)$
    return $dist$
```



Q = ~~⟨0,A⟩~~

~~⟨2,B⟩ ⟨4,C⟩~~

~~⟨3,C⟩ ⟨4,C⟩ ⟨6,D⟩~~

~~⟨4,C⟩ ⟨4,D⟩ ⟨6,D⟩~~

~~⟨4,D⟩ ⟨6,D⟩~~

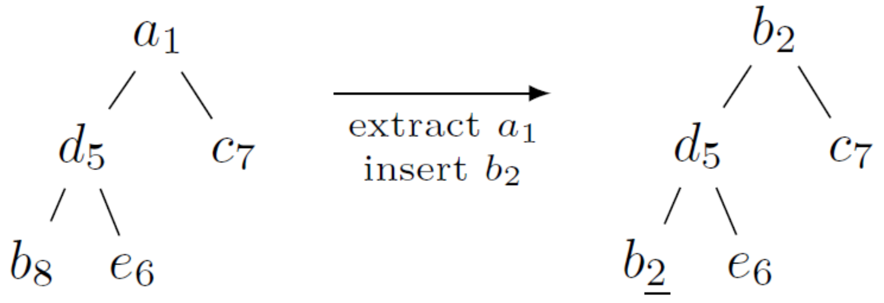
~~⟨6,D⟩ ⟨6,E⟩~~

~~⟨6,E⟩~~

A Shaky Idea…

```
proc Dijkstra$_4(V, E, \delta, s)$
    $dist[v] = +\infty$ for all $v \in V \setminus \{s\}$
    $dist[s] = 0$
    $visited = \emptyset$
    Insert$(Q, \langle dist[s], s\rangle)$
    while $Q \neq \emptyset$ do
d never used ⟶ $\langle \cancel{d}, u\rangle$ = ExtractMin$(Q)$
        if $u \notin visited$ then
            $visited = visited \cup \{u\}$
            for $(u, v) \in E \cap (\{u\} \times V)$ do
                if $dist[u] + \delta(u, v) < dist[v]$ then
                    $dist[v] = dist[u] + \delta(u, v)$
                    Insert$(Q, \langle \cancel{dist[v]}, v\rangle)$
    return $dist$
```

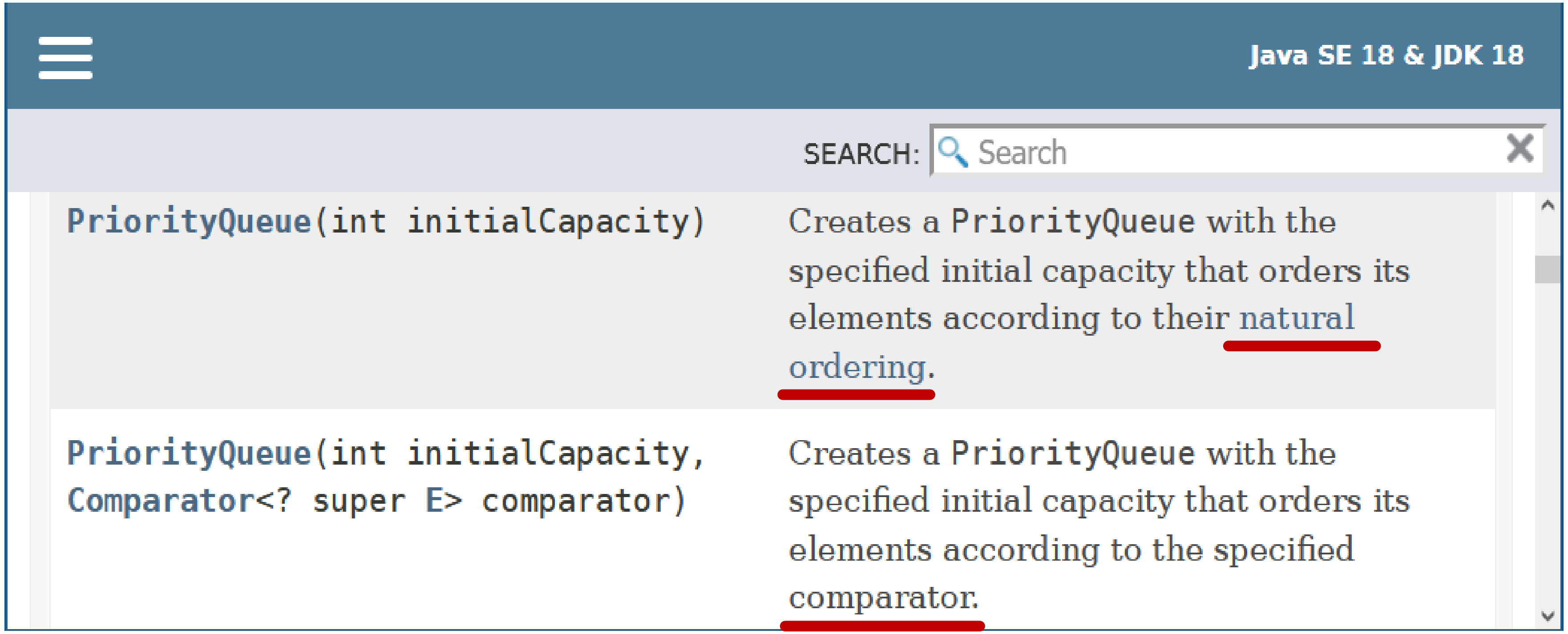
- Q only store nodes (save space)
- Comparator
- Key = current distance *dist*



Heap invariants break

The Challenge - Java’s Builtin Binary Heap

- Comparator function



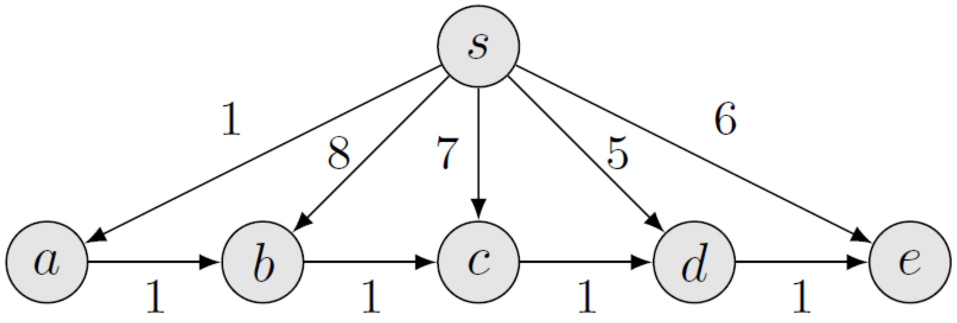
`PriorityQueue(int initialCapacity)`	Creates a `PriorityQueue` with the specified initial capacity that orders its elements according to their natural ordering.
`PriorityQueue(int initialCapacity, Comparator<? super E> comparator)`	Creates a `PriorityQueue` with the specified initial capacity that orders its elements according to the specified comparator.

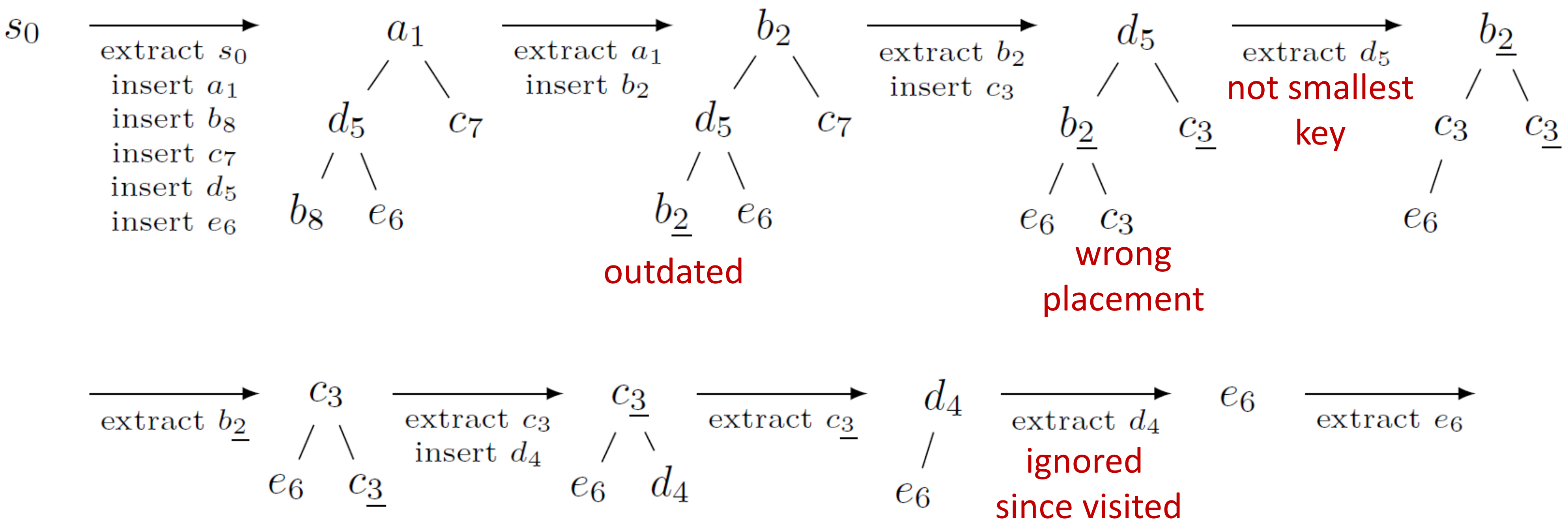
Experimental Study

```
visited = set()
Q = Queue()
Q.insert(Item(0, source))
while not Q.empty():
    u = Q.extract_min().value
    if u not in visited:
        visited.add(u)
        for v in G.out[u]:
            dist_v = dist[u] + G.weights[(u, v)]
            if dist_v < dist[v]:
                dist[v] = dist_v
                parent[v] = u
                Q.insert(Item(dist[v], v))
```

- Implemented Dijkstra_4 in Python
- Stress test on random cliques
- Binary heaps failed (default priority queue in Java and Python)

Binary Heaps Fail using *dist* in a Comparator





outdated

wrong placement

not smallest key

ignored since visited

Experimental Study

```
visited = set()
Q = Queue()
Q.insert(Item(0, source))
while not Q.empty():
    u = Q.extract_min().value
    if u not in visited:
        visited.add(u)
        for v in G.out[u]:
            dist_v = dist[u] + G.weights[(u, v)]
            if dist_v < dist[v]:
                dist[v] = dist_v
                parent[v] = u
                Q.insert(Item(dist[v], v))
```

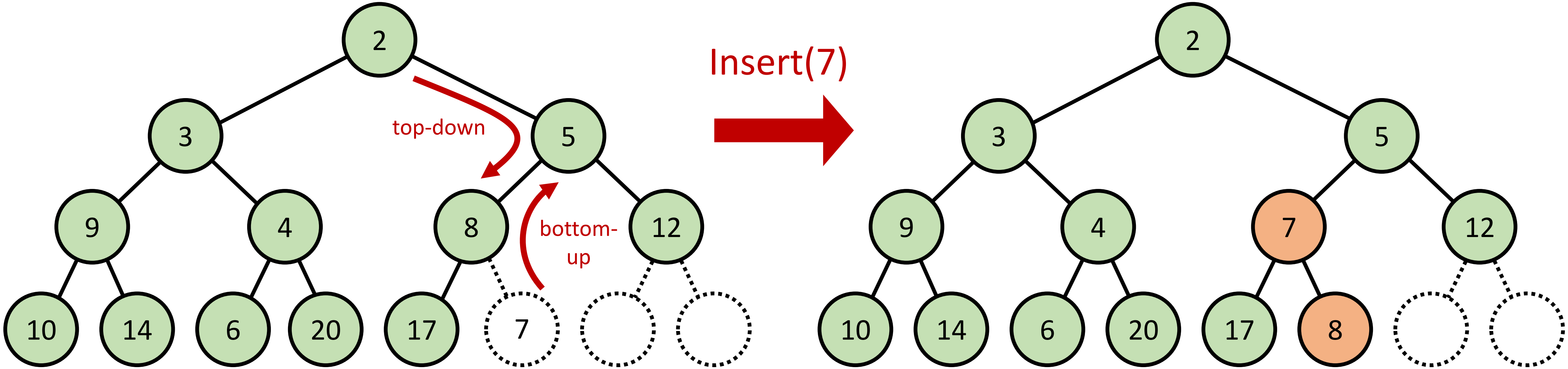
- Implemented Dijkstra_4 in Python
- Stress test on random cliques
- Binary heaps failed (default priority queue in Java and Python)
- Skew heaps worked
- Leftist heaps worked
- Pairing heaps worked
- Binomial queues worked
- Post-order heaps worked
- Binary heaps with top-down insertions worked

unexpected

Pointer based

Implicit (space efficient)

Binary Heap Insertions : Bottom-up vs Top-down

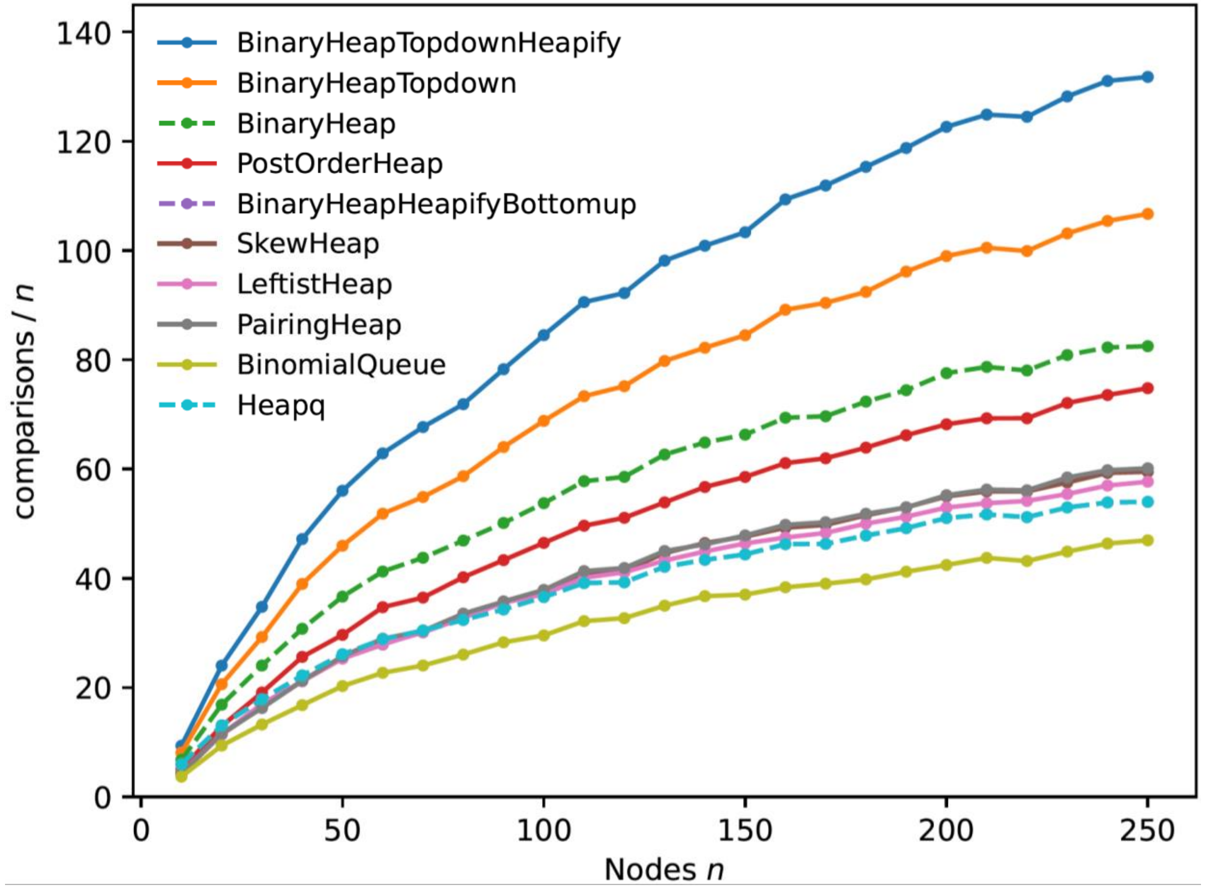


- **Theorem** Skew, left, pairing, binomial, post-order, binary top-down heaps support a generalized notion of heap order with decreasing keys
- **Theorem** Dijkstra$_4$ works correctly

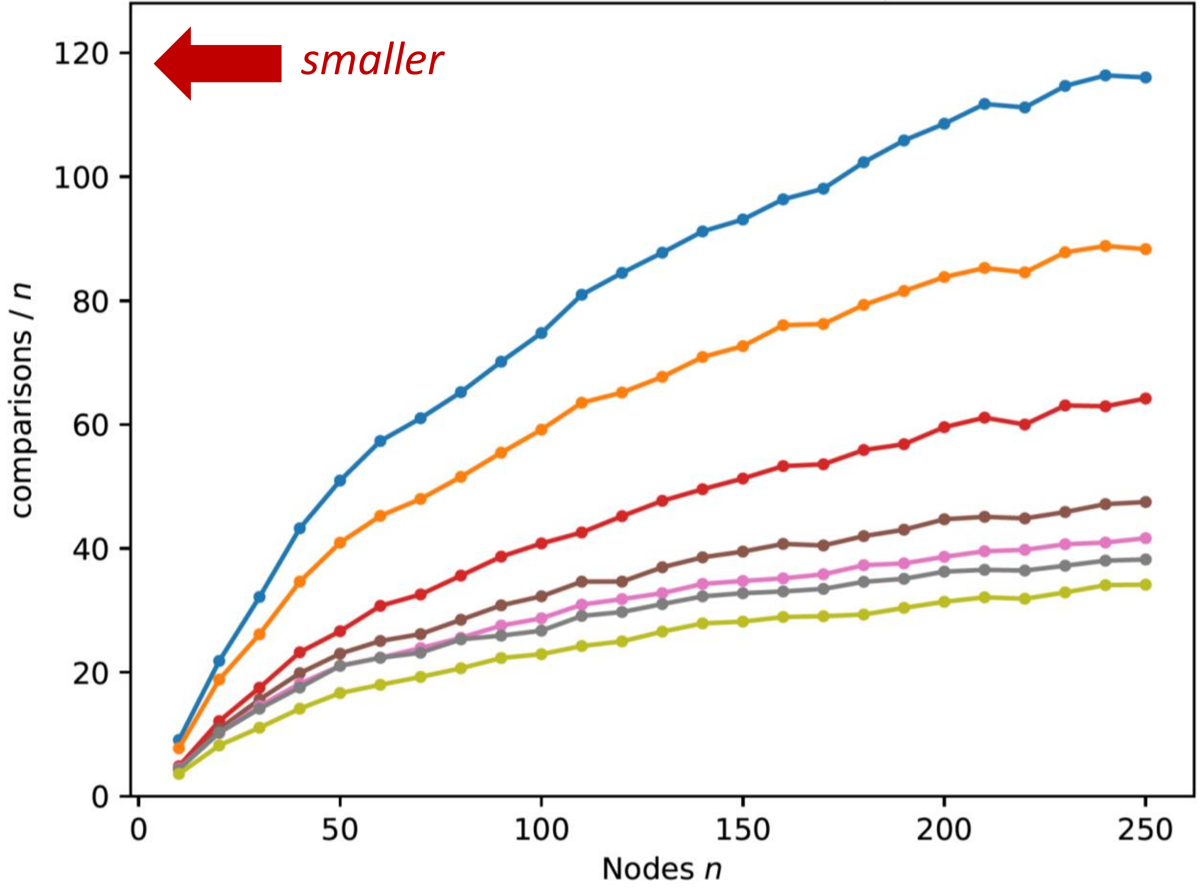
Experimental Evaluation of Various Heaps

- Cliques with uniform random weights
- With decreasing keys less comparisons (outdated items removed earlier)

⟨key, value⟩ pairs



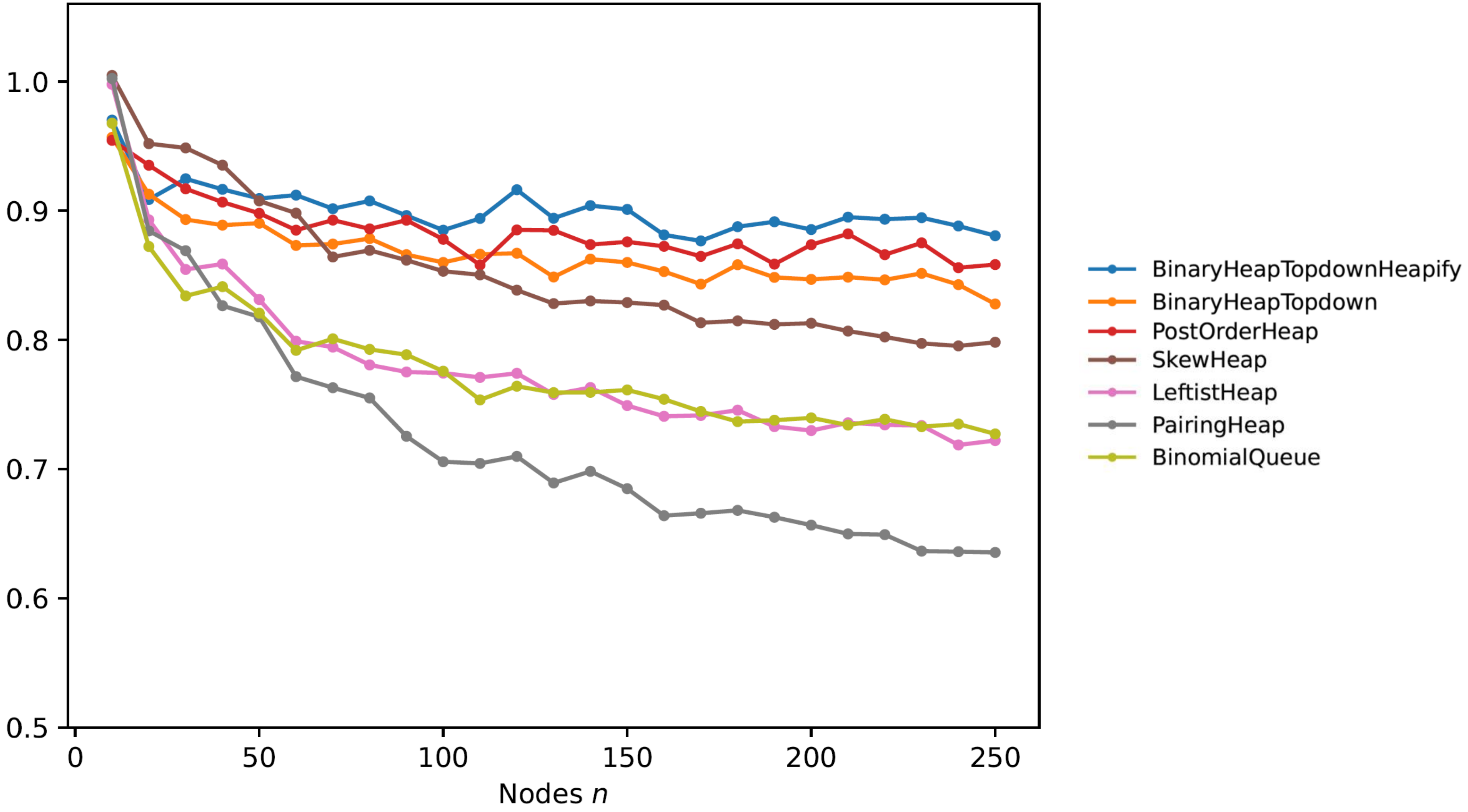
decreasing keys



smaller

Reduction in Comparisons

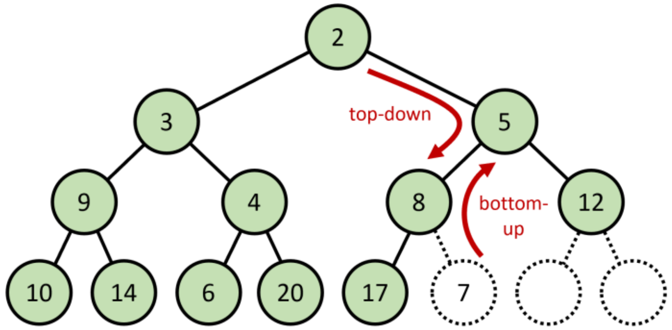
comparisons decreasing keys / comparisons ⟨key, value⟩ pairs



Summary of the Unexpected Journey

- Introduced notion of priority queues with decreasing keys
  … as an approach to deal with outdated items in Dijkstra’s algorithm
- Experiments identified priority queues supporting decreasing keys
  … just had to prove it
- Builtin priority queues in Java and Python are binary heaps
  … do not support decreasing keys
- Binary heaps with top-down insertions do support decreasing keys
  … and also

skew heaps, leftist heaps, pairing heaps,
binomial queues, post-order heaps



The reviewer is always right

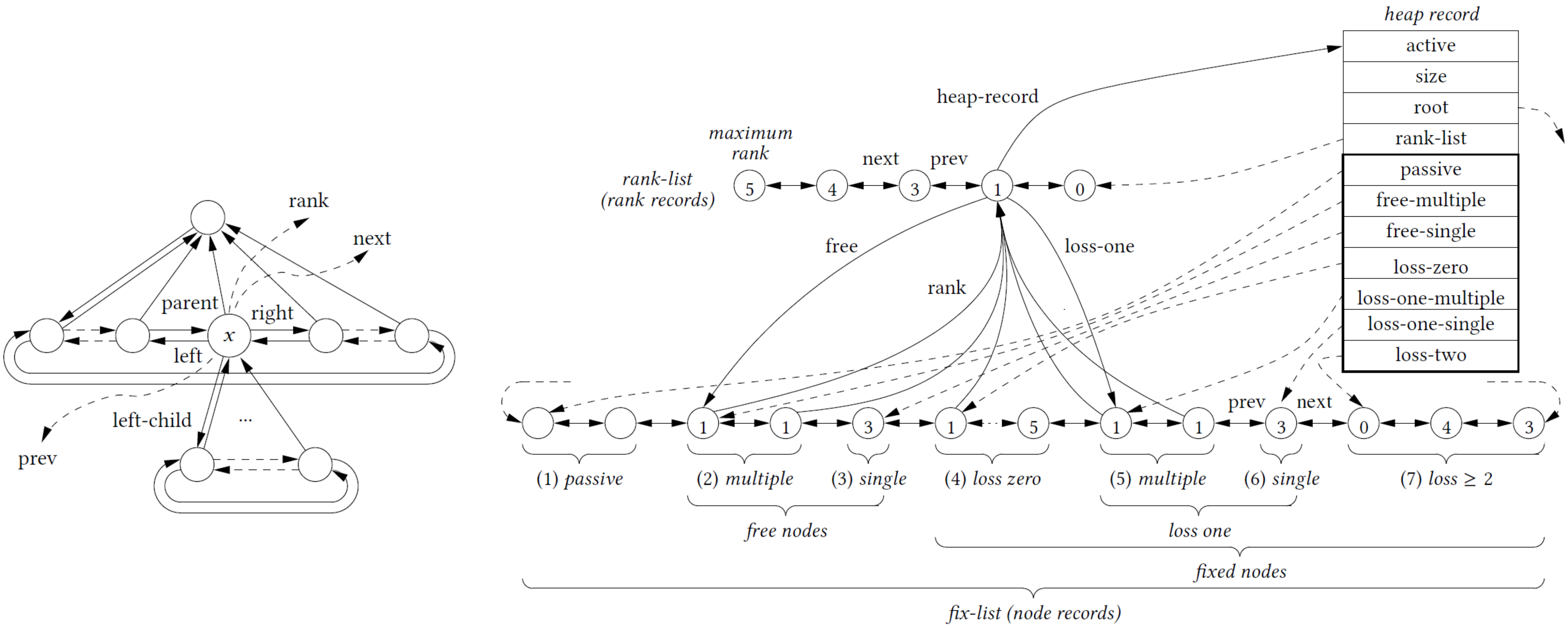
”If there was a implementation where the authors verified that everything did what it was supposed to, I would be more confident that things were correct (I am not talking about a practical implementation, I am talking about one to make sure all invariants hold).”

Anonymous reviewer

Strict Fibonacci Heaps

	Binary heap [Williams 1964] worst-case	**Fibonacci heap [Fredman, Tarjan 1984] amortized**	**Strict Fibonacci heap [B., Lagogiannis, Tarjan 2012] worst-case**
Insert	O(log n)	O(1)	O(1)
ExtractMin	O(log n)	O(log n)	O(log n)
DecreaseKey	O(log n)	O(1)	O(1)
Meld	-	O(1)	O(1)

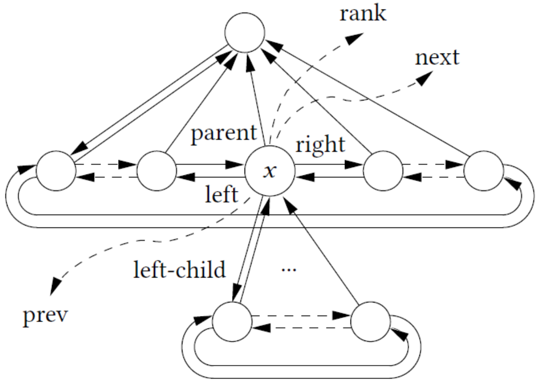
Strict Fibonnacci Heaps

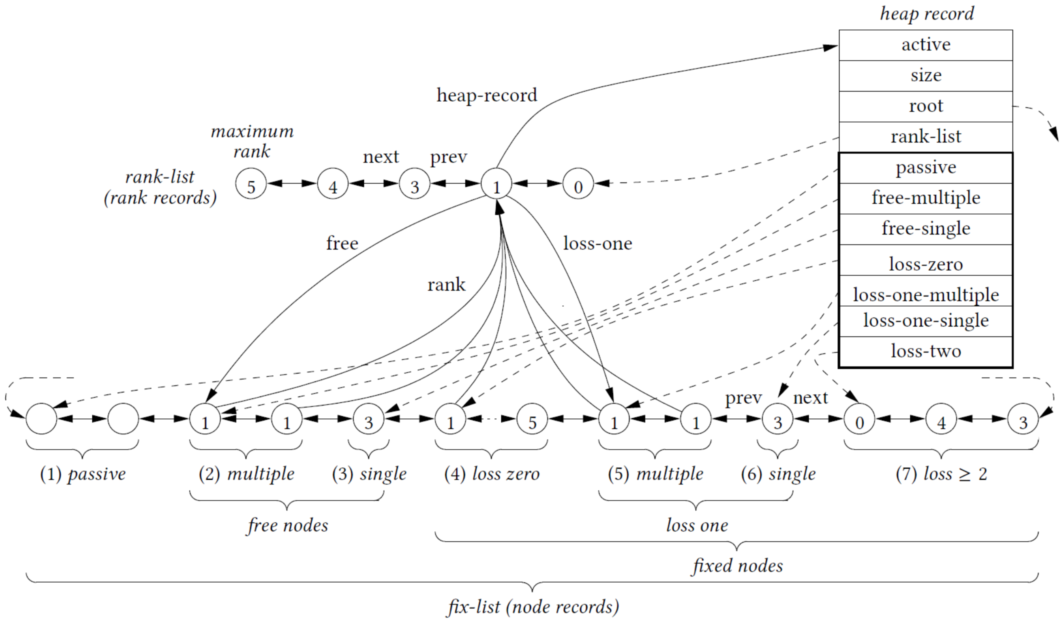


+ many structural invariants

Python Implementation

- 1589 lines
- 215 assert statements
- All claimed invariants turned into assert statements
- Validation methods to traverse full structure to verify all claimed invariants
- Stress test using random inputs
- Supported the theory





www.cs.au.dk/~gerth/strict_fibonacci_heaps.py

Code coverage

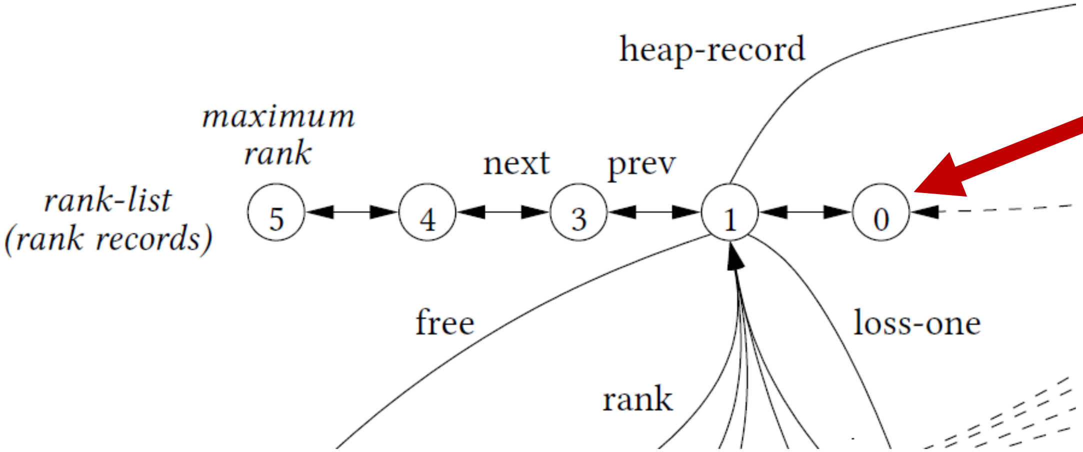
- Used the Python module coverage
- Some code rarely executed
- Repeat random test 1.000.000 times
- Most code executed at least once

- Realized there was code for cases which provably never can occur
- Implementation → new invariants discovered

coverage.readthedocs.io

Branch coverage

- Thought code coverage would find all ”logical errors”
- Found several **if** statements with no **else** part, where condition provably would always be true
- Implementation → new invariants discovered (and assertions added)

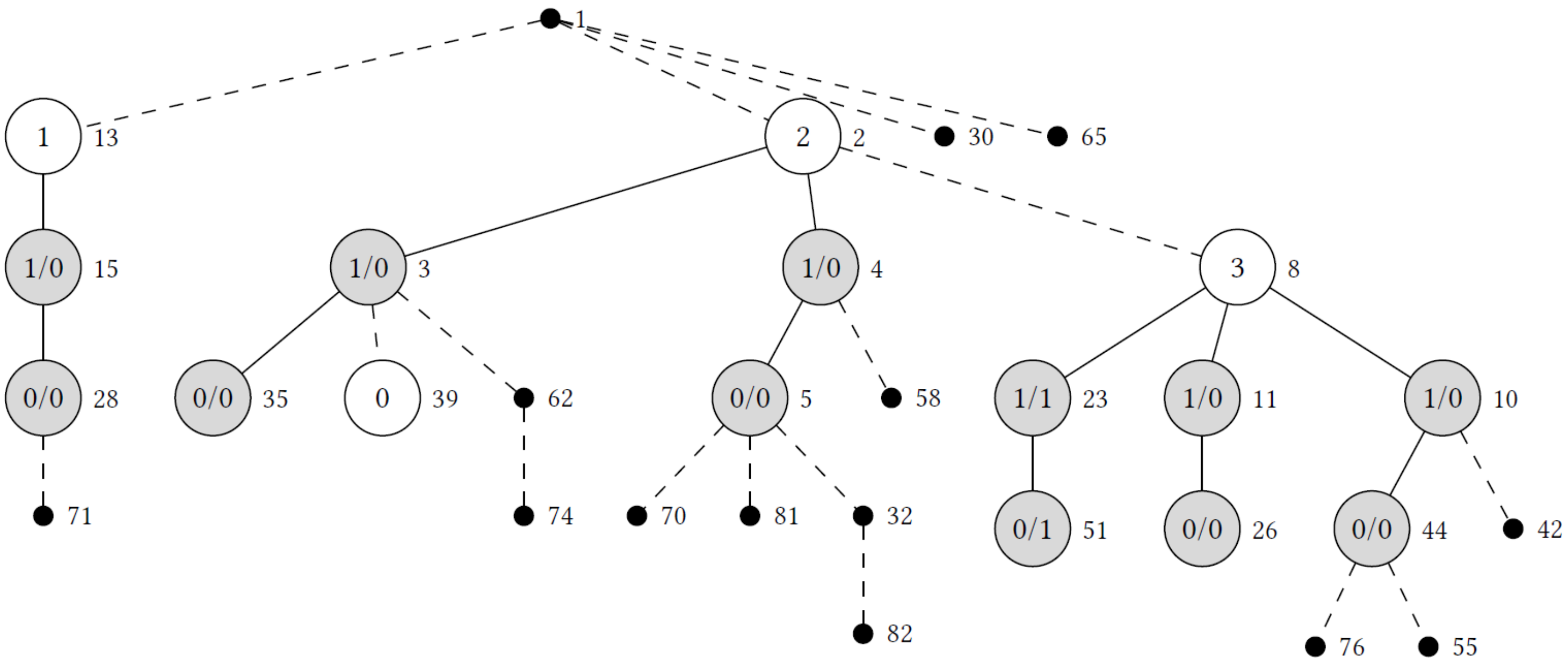


always exists

coverage.readthedocs.io

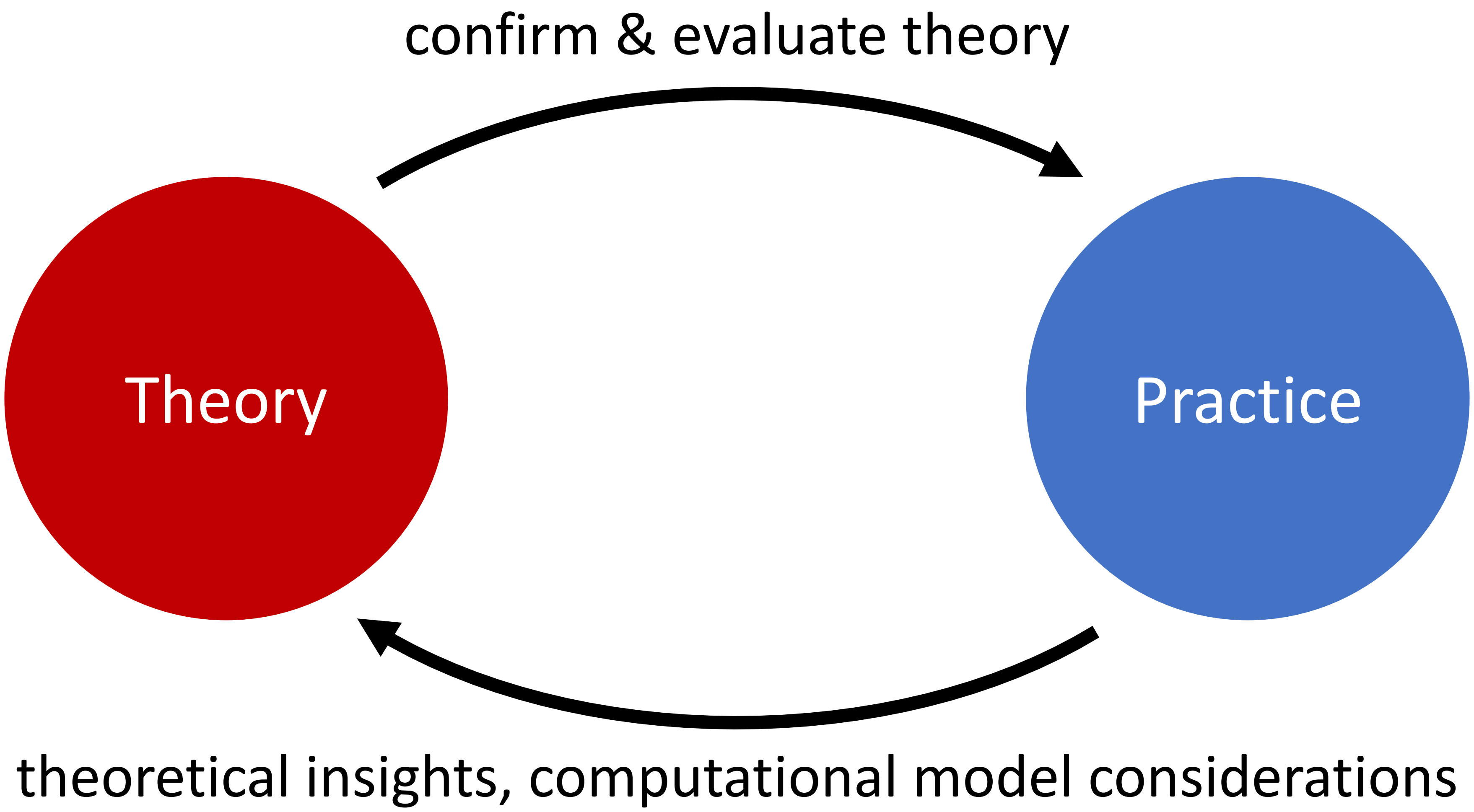
”The first main suggestion is to have at least one figure with a logical diagram of a non-trivial example structure, […]. This would go a long way in giving some idea of what the structure is.”

Anonymous reviewer



- Hard to manually create a figure that was guaranteed to be a real example
- Could use implementation to automatically generate (LaTeX tikz) figures
- Generated random inputs
- Formalized requirements to figure as a loop condition
- Repeat until happy

Data Structure Design



confirm & evaluate theory

Theory

Practice

theoretical insights, computational model considerations