IS WRITING PERFORMANT CODE TOO EXPENSIVE?

TOMEK KOWALCZEWSKI

- Works @ Codewise
- Failed tech lead

How to fail as a new engineering manager

In 8 easy steps

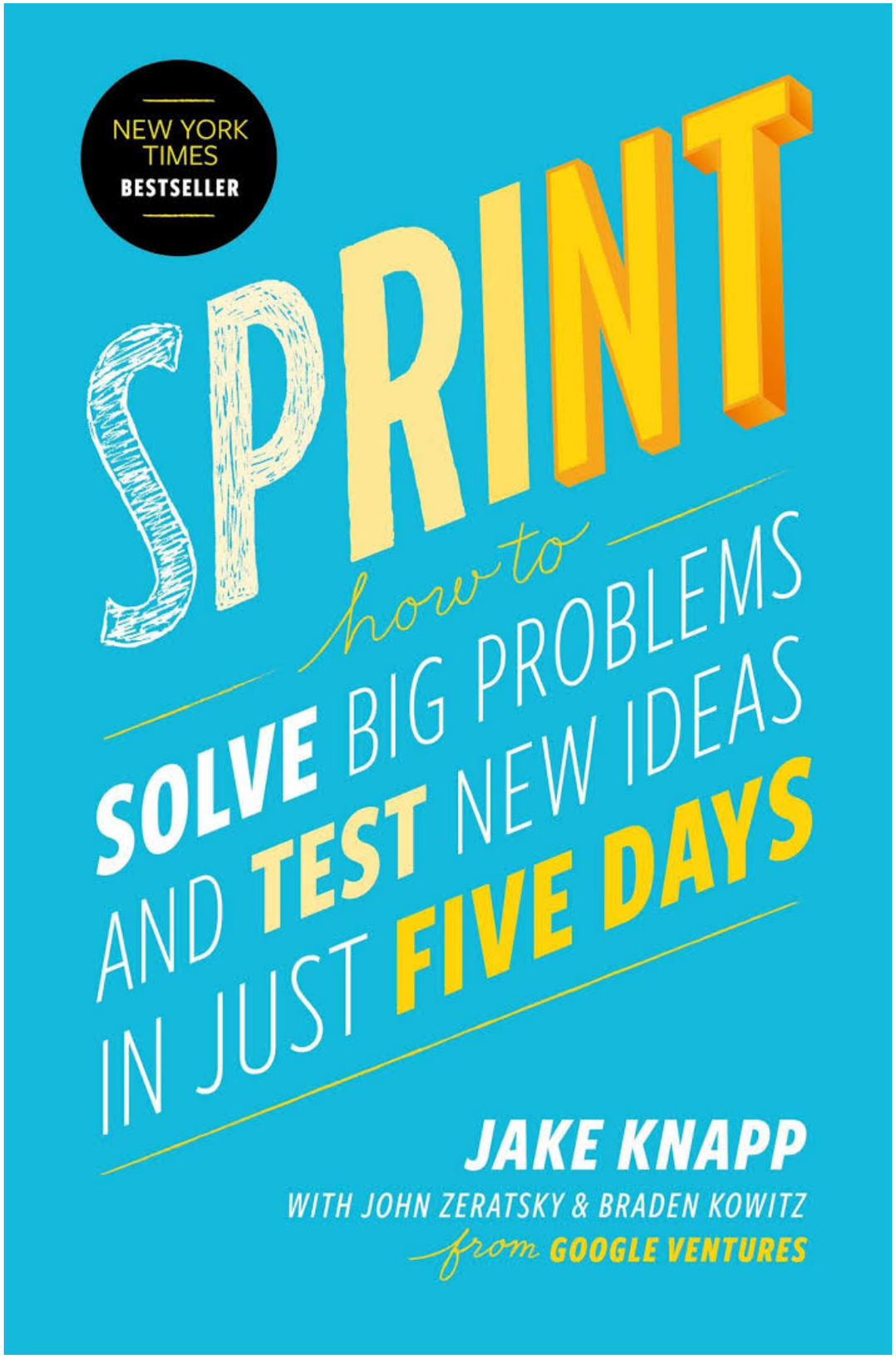
1. Keep coding.

- Spent about half of his carrier optimising allocations :(

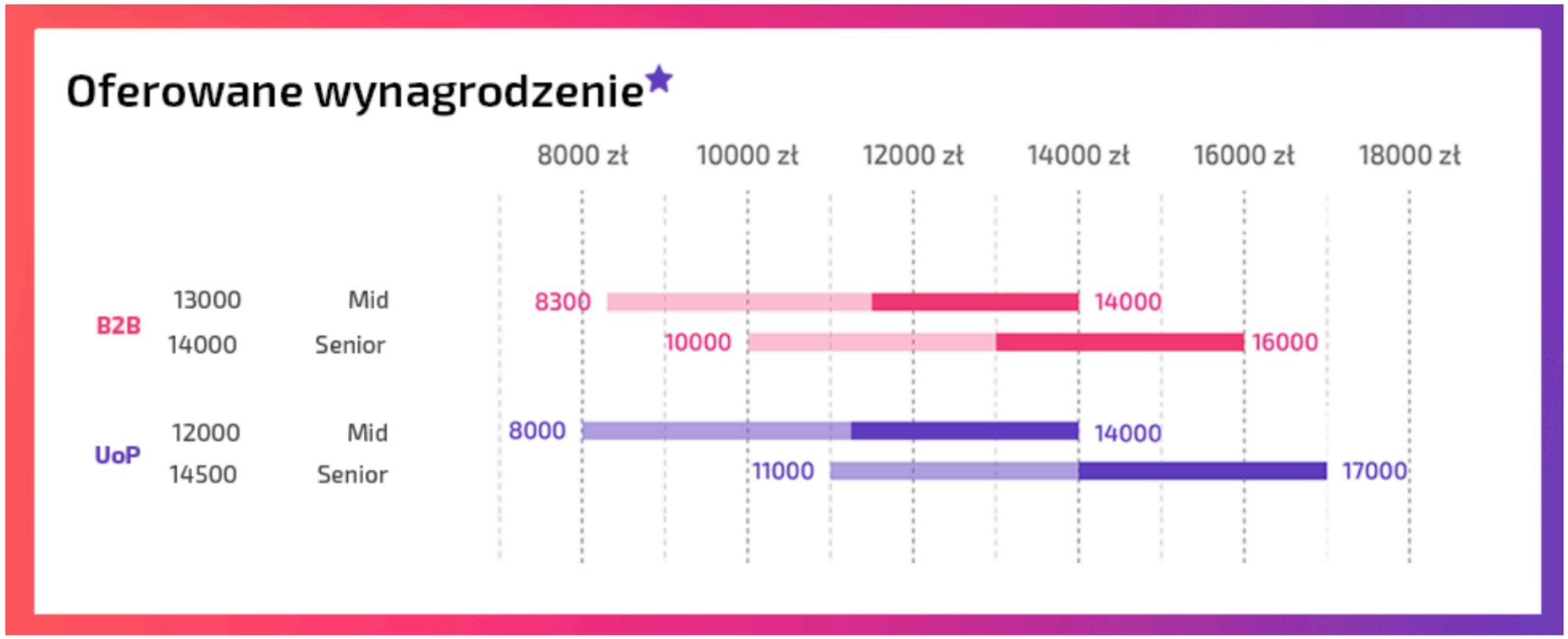
FADS@VLDB WORKSHOP

Failed Aspirations in Database Systems

SPRINT


NEW YORK TIMES BESTSELLER
SPRINT
how to
SOLVE BIG PROBLEMS AND TEST NEW IDEAS IN JUST FIVE DAYS
JAKE KNAPP
WITH JOHN ZERATSKY & BRADEN KOWITZ
from GOOGLE VENTURES

10 000$ / MONTH



no fluff {jobs}

20 000$ / MONTH

x1e.32xlarge	128	vCPU	3,904 GiB
2 x 1,920 SSD			$26.688 per Hour

	Nvidia V100 (2017)	IBM ASCI White (2000)
Number of Processor Cores	3584	8192 (512 nodes x 16 IBM Power3)
Double-Precision Performance	7.5 TeraFLOPS	7.2 TeraFLOPS
NVIDIA NVLink™ v2 Interconnect Bandwidth	2x150 GB/s	N/A
PCIe x16 Interconnect Bandwidth	2x16 GB/s	N/A
Memory Capacity	16 GB	**6 TB DRAM** (Power 3 w/ **16 MB L2 cache**)
Max. overall data transfer speed	900 GB/s	?
Weight	450 gramm	106 tons
Energy consumption	**300W**	**3 MW**

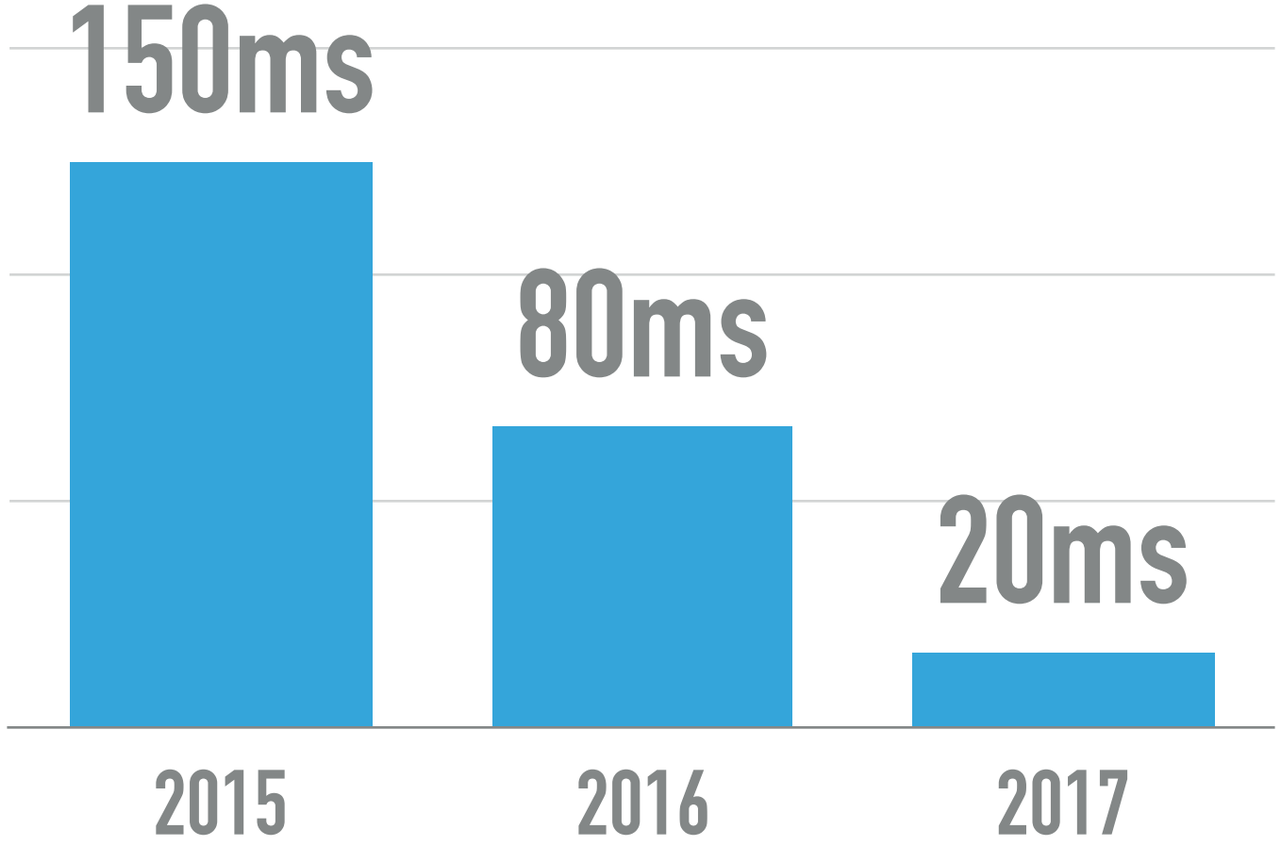
ISSUES SUMMARY

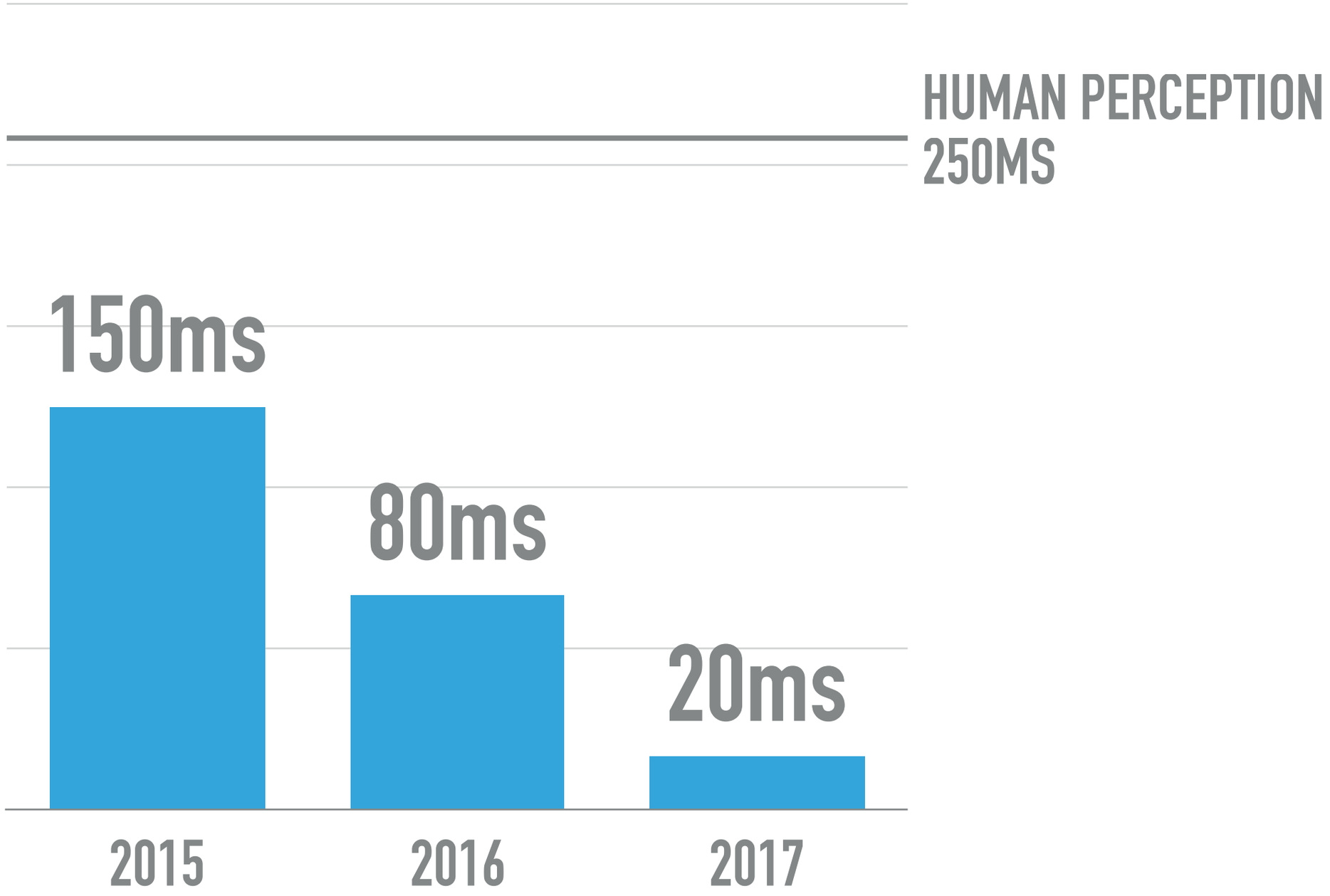
- Compute gets cheaper and faster
- Cost of development goes up
- Technology is only small part of the equation

VERY LARGE DATABASES CONFERENCE 2018

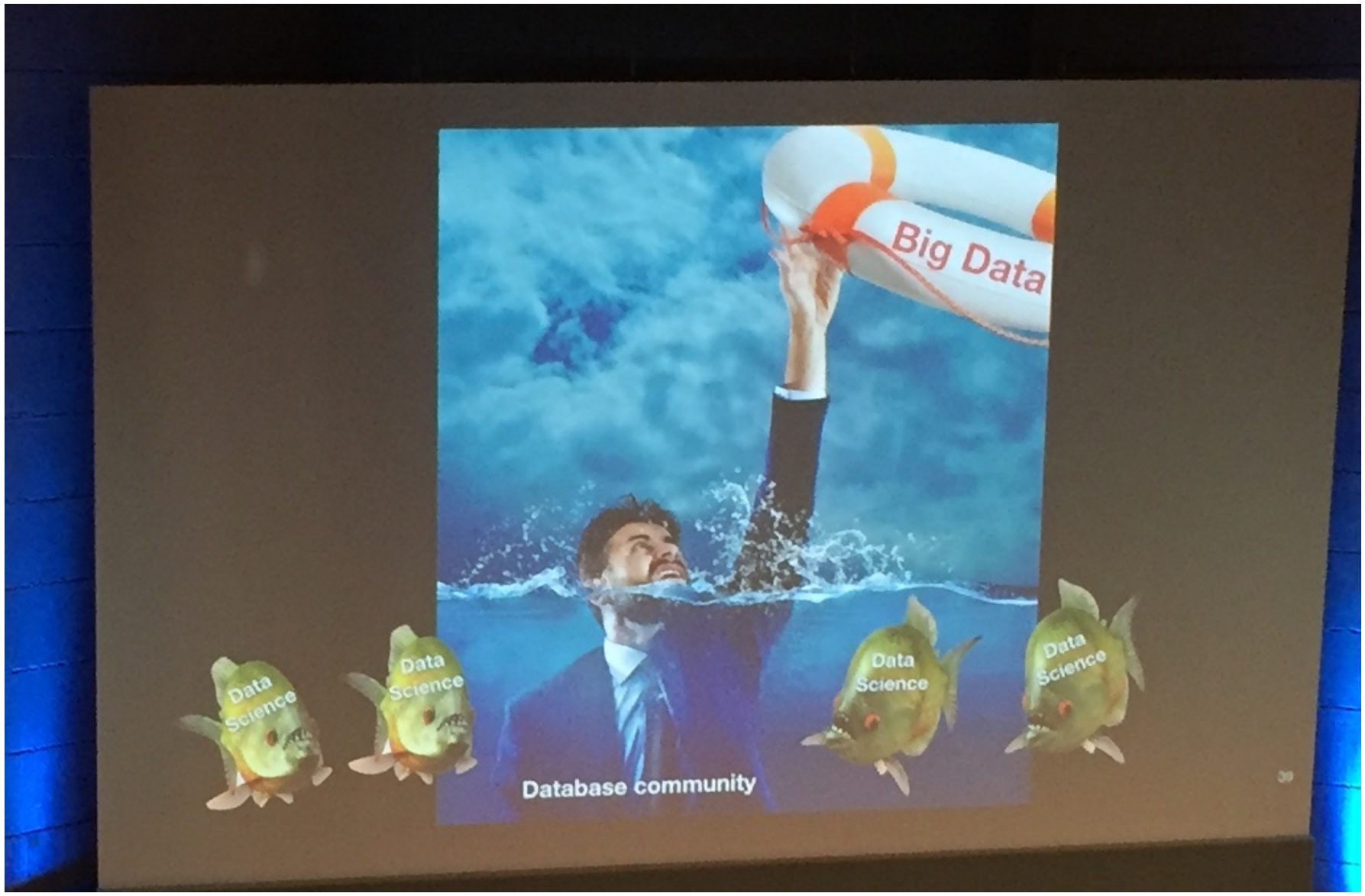
source: vldb

source: wikipedia


150ms
80ms
20ms
2015
2016
2017

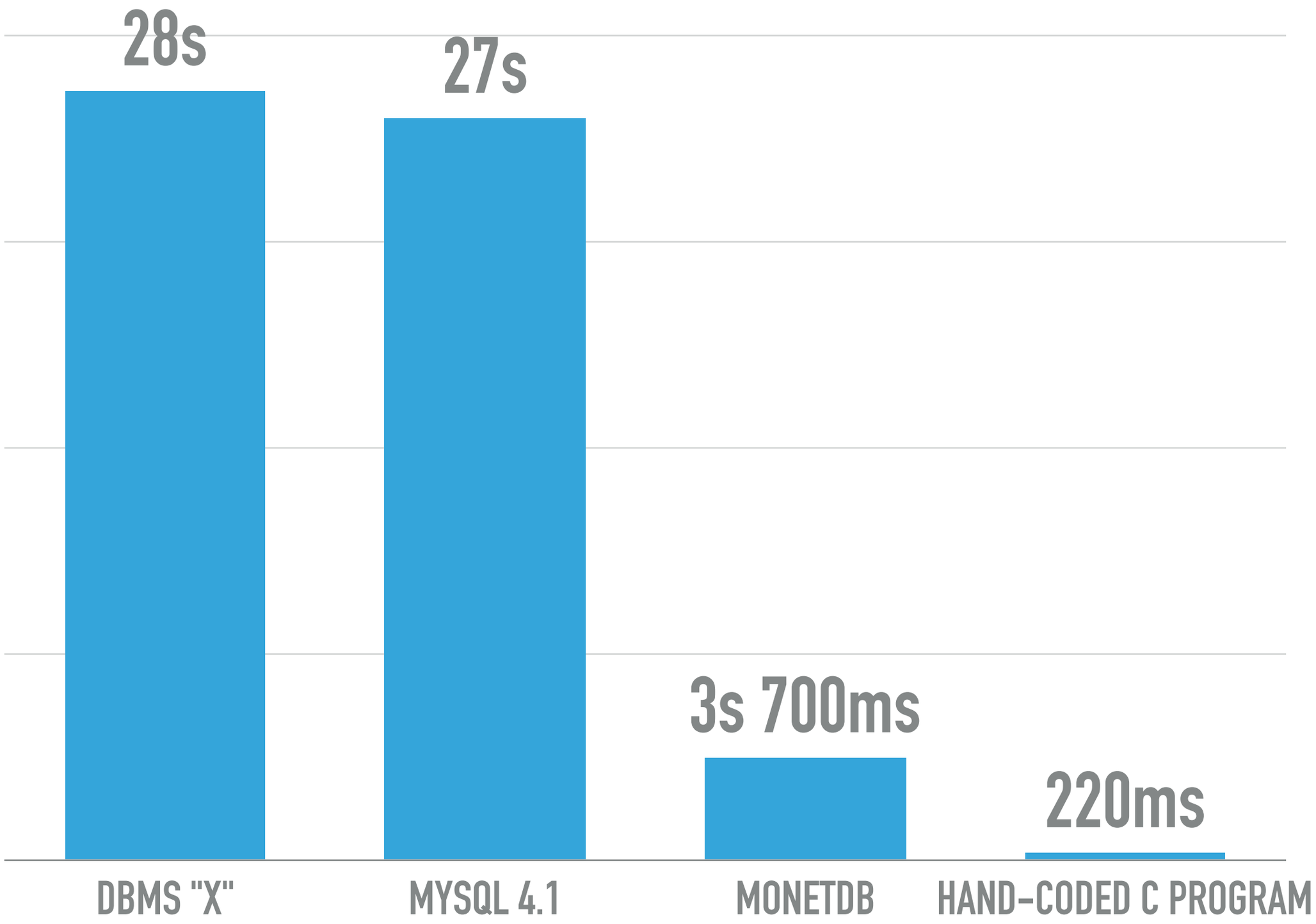

HUMAN PERCEPTION
250MS
150ms
80ms
20ms
2015
2016
2017

VERY LARGE DATABASES CONFERENCE



source: vldb

EFFICIENTLY COMPILING EFFICIENT QUERY PLANS FOR MODERN HARDWARE



Scalability! But at what COST?

Frank McSherry
Unaffiliated

Michael Isard
Microsoft Research

Derek G. Murray
Unaffiliated*

NOW, WE ARE GOING TO INTRODUCE SOME NEW CANDIDATES FOR "STATE-OF-THE-ART" ALGORITHMS, BY WRITING SOME FOR-LOOPS AND RUNNING THEM ON MY LAPTOP.

Frank McSherry

SCALABILITY! BUT AT WHAT COST?

scalable system	cores	twitter
Stratosphere [8]	16	950s
X-Stream [21]	16	1159s
Spark [10]	128	1784s
Giraph [10]	128	200s
GraphLab [10]	128	242s
GraphX [10]	128	251s

SCALABILITY! BUT AT WHAT COST?

scalable system	cores	twitter
Stratosphere [8]	16	950s
X-Stream [21]	16	1159s
Spark [10]	128	1784s
Giraph [10]	128	200s
GraphLab [10]	128	242s
GraphX [10]	128	251s
Single thread (SSD)	1	153s

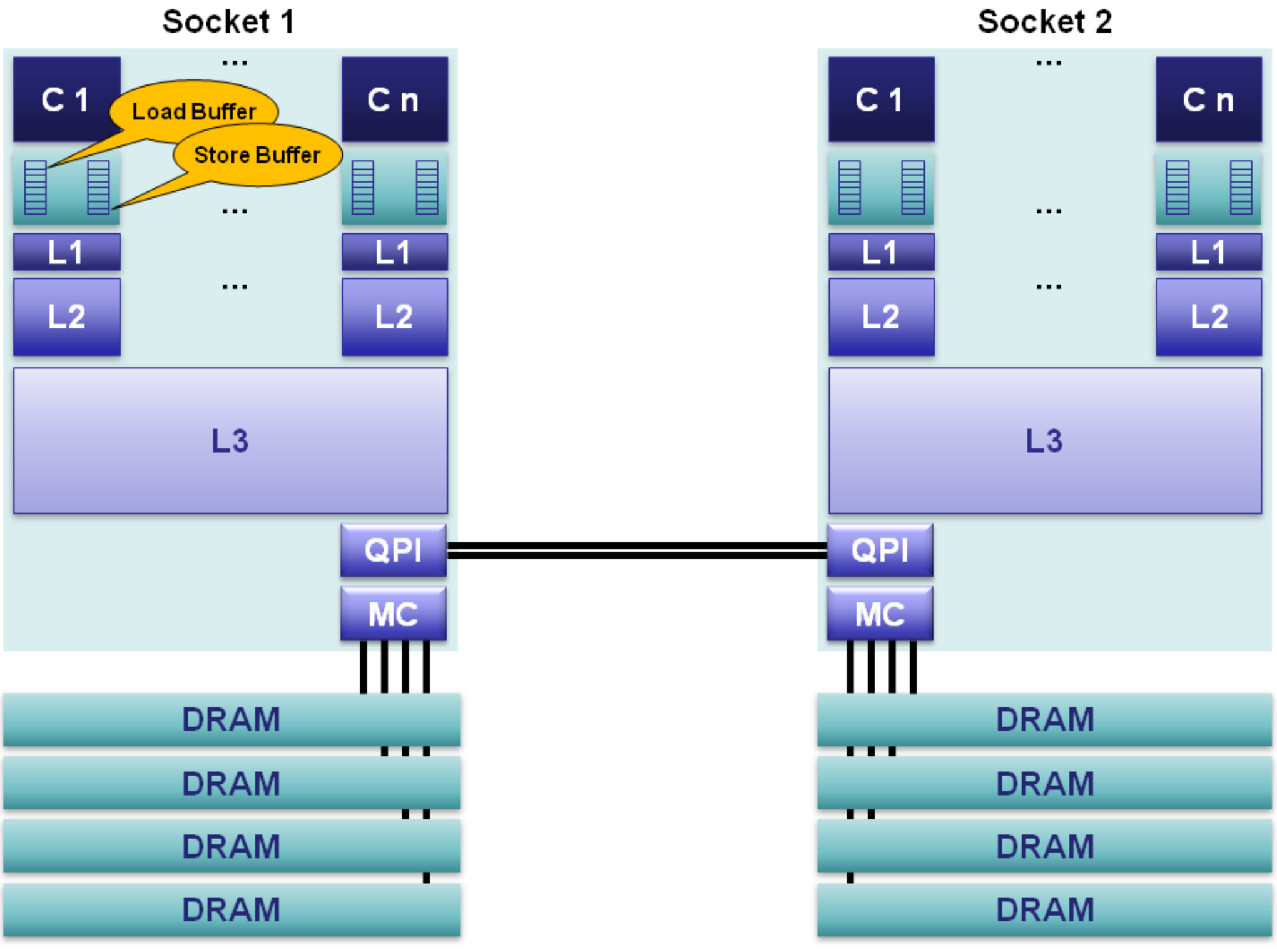
PERFORMANCE!

- Intuition about what is possible
- Impact on real life Java code

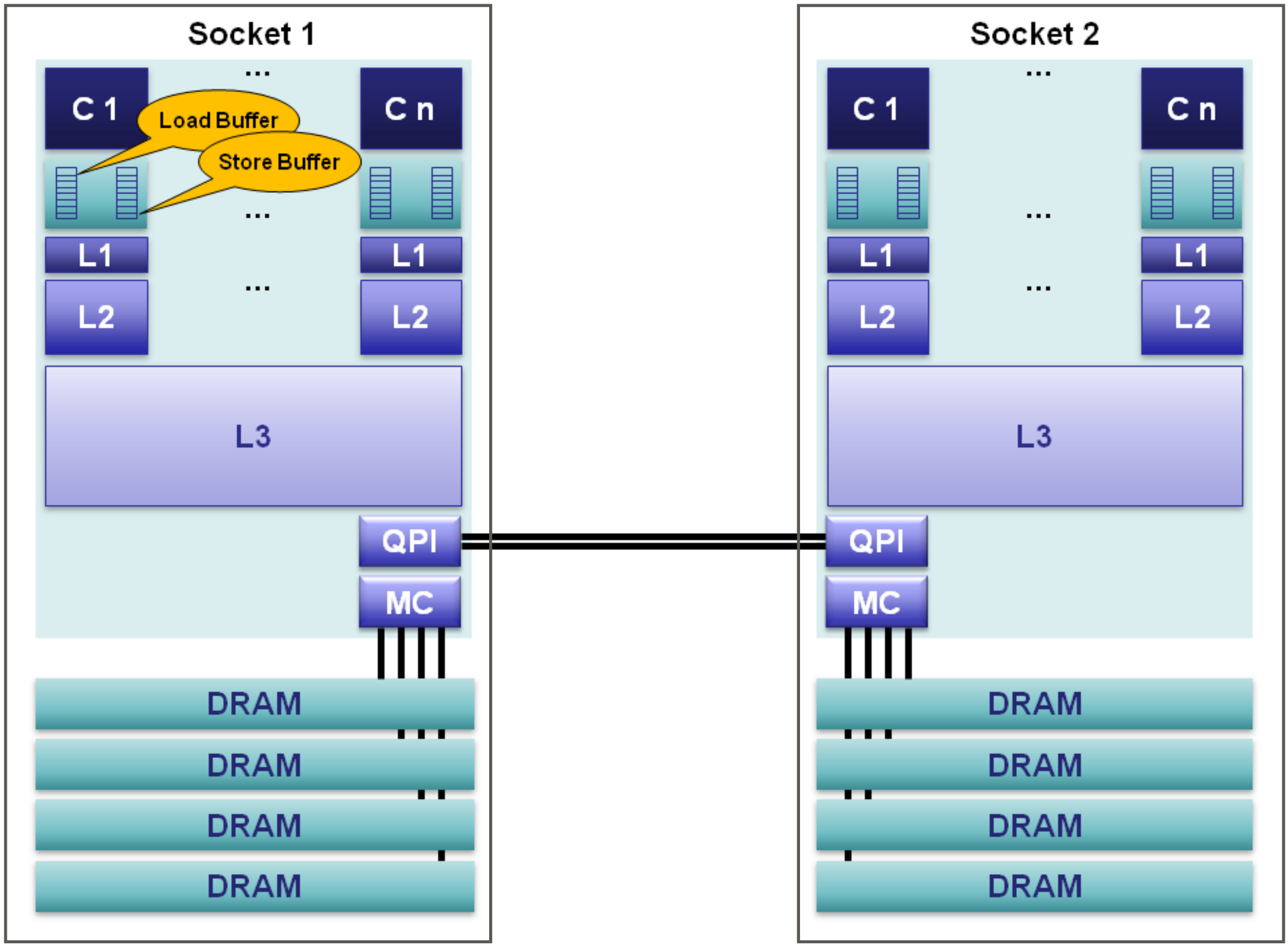
WHAT AFFECTS PERFORMANCE?

- Stuff, lot's of stuff
- Branch prediction
- Prefetching (vs. random access)
- NUMA
- GC, etc…

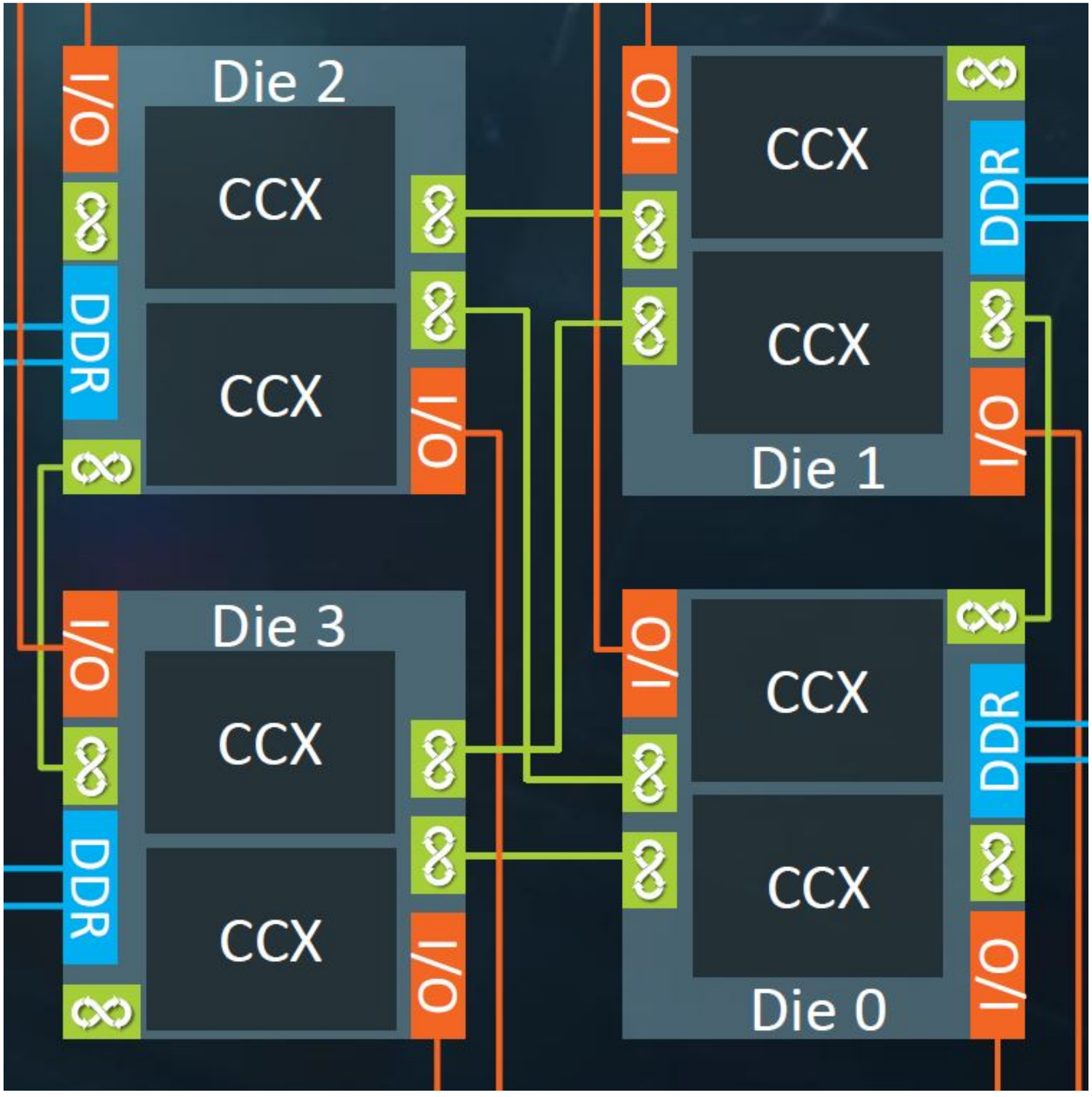
CPU FUNCTIONAL BLOCKS



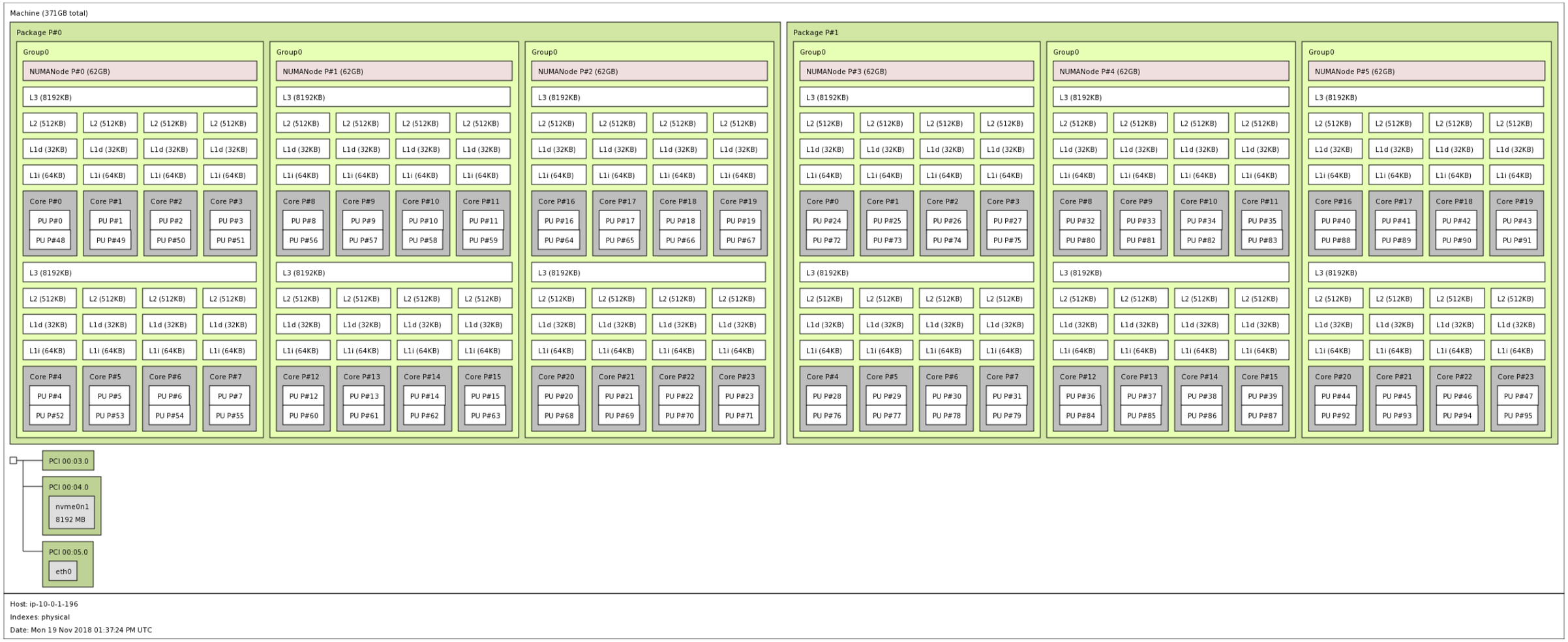
CPU FUNCTIONAL BLOCKS



AMD EPYC DIE

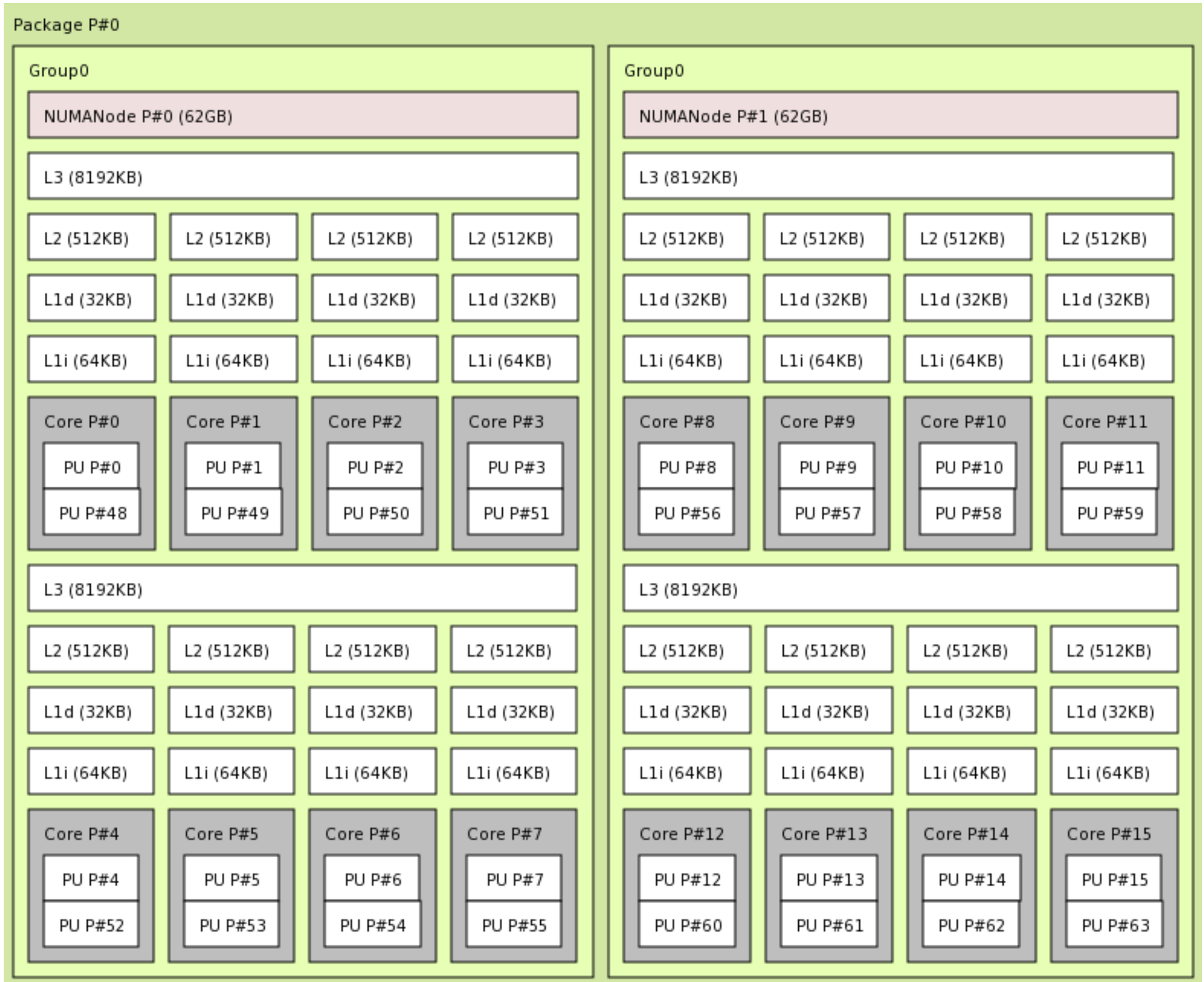

Die 2
CCX
CCX
I/O
DDR
I/O
I/O
CCX
CCX
DDR
I/O
Die 1
Die 3
CCX
CCX
I/O
DDR
I/O
I/O
CCX
CCX
DDR
I/O
Die 0

AMD EPYC DIAGRAM




```
./lstopo --physical --output-format png /tmp/lstopo.png
```

AMD EPYC DIAGRAM



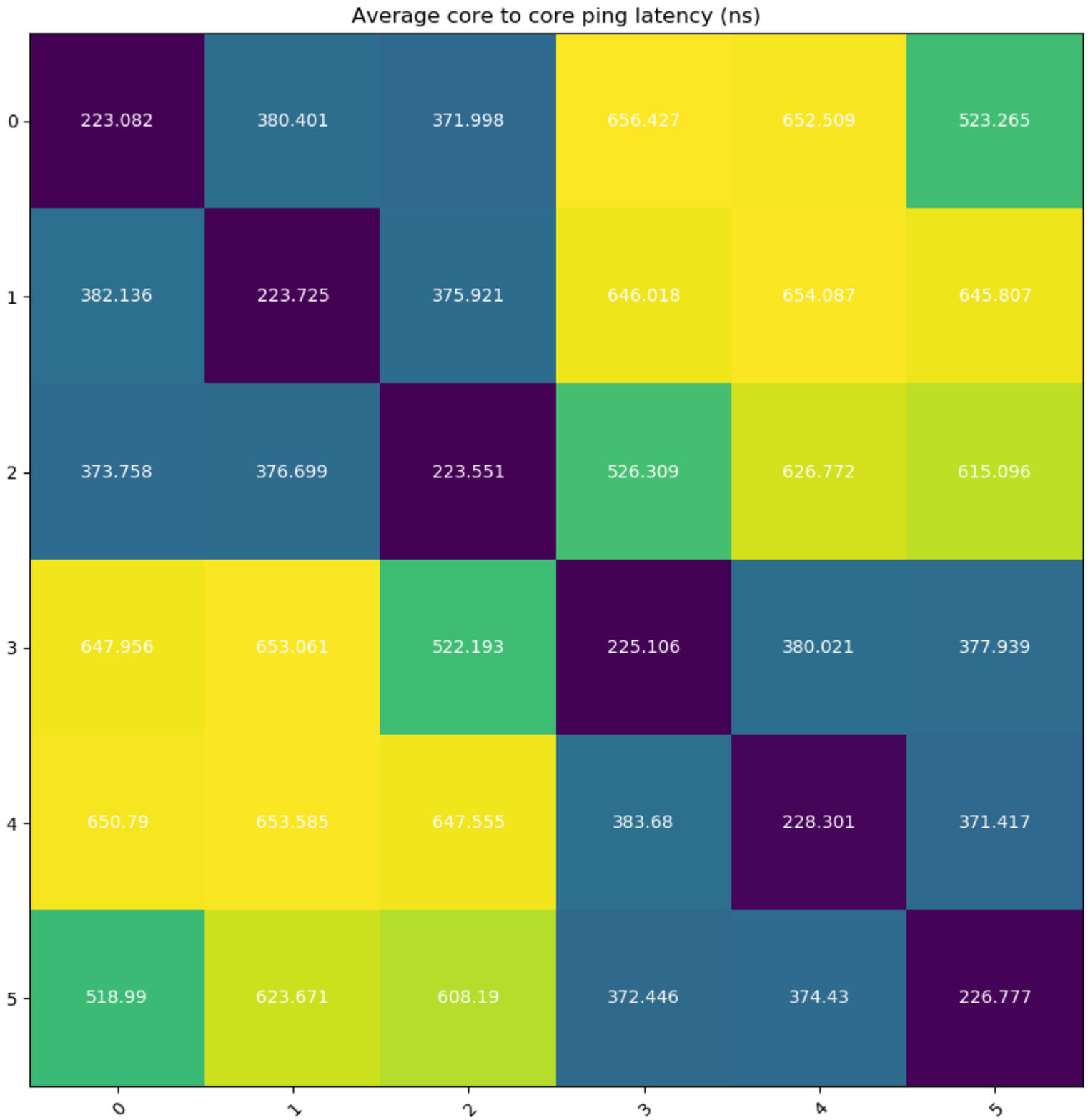
```
<dependency>
    <groupId>org.openjdk.jmh</groupId>
    <artifactId>jmh-core</artifactId>
    <version>1.21</version>
</dependency>

<dependency>
    <groupId>org.openjdk.jmh</groupId>
    <artifactId>jmh-generator-annprocess</artifactId>
    <version>1.21</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.xerial</groupId>
    <artifactId>jnuma</artifactId>
    <version>0.1.3</version>
</dependency>
```

```
@Setup
public void setUp(CoreSequence coreSequence) {
    Numa.runOnNode(node);

    ByteBuffer buffer = Numa.allocOnNode(capacity, node);
    long address = address(buffer);
}
```

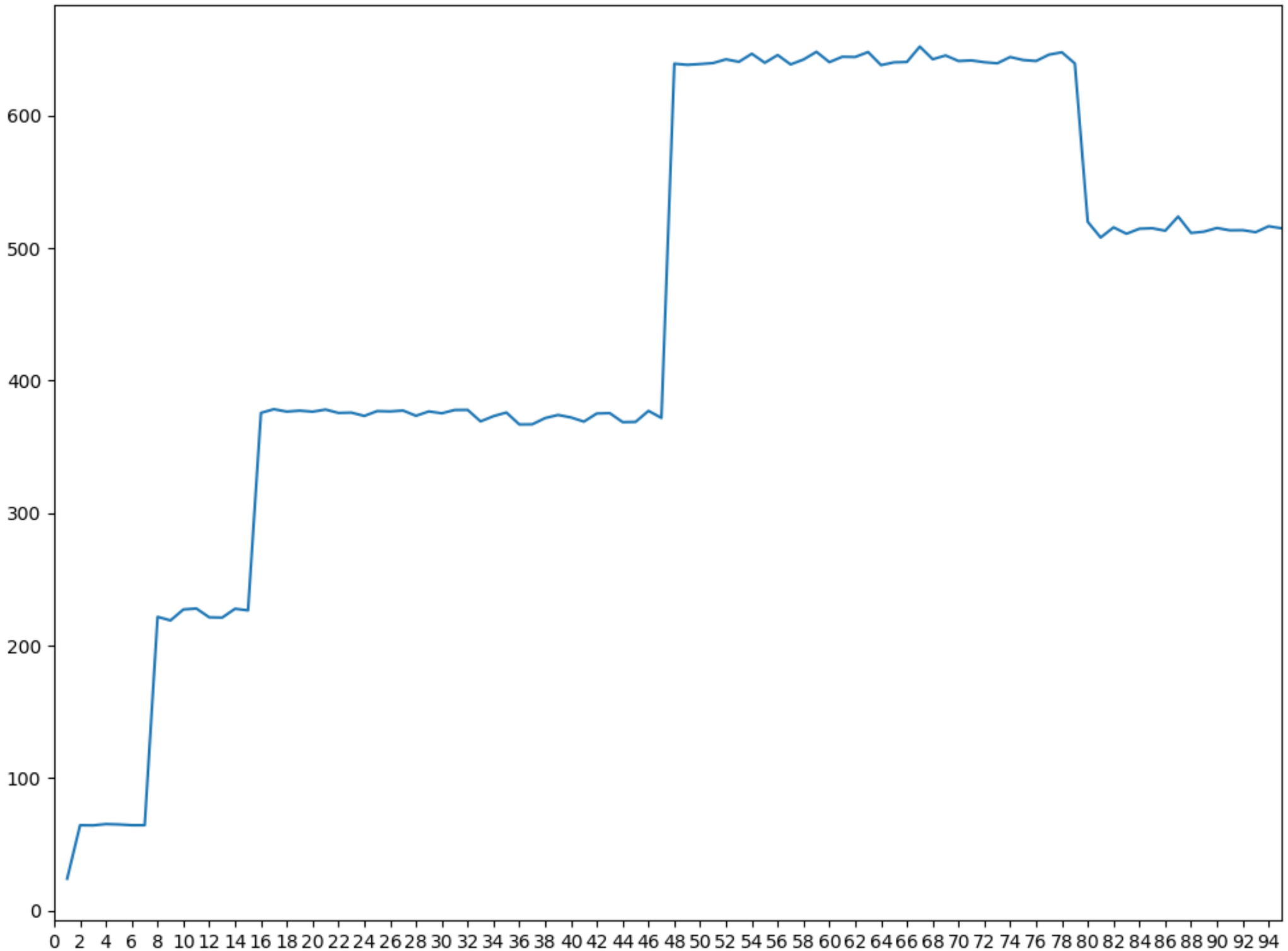
AMD EPYC CORE PING LATENCY (JAVA-JMH)

```
@Benchmark
@Group("pingpong")
public void ping(Control cnt, GroupState groupState, CoreAssigner coreAssigner)
{
    while (!cnt.stopMeasurement && !UNSAFE.compareAndSwapInt(null,
groupState.flagAddress, 0, 1)) {
        // this body is intentionally left blank
    }
}

@Benchmark
@Group("pingpong")
public void pong(Control cnt, GroupState groupState, CoreAssigner coreAssigner)
{
    while (!cnt.stopMeasurement && !UNSAFE.compareAndSwapInt(null,
groupState.flagAddress, 1, 0)) {
        // this body is intentionally left blank
    }
}
```

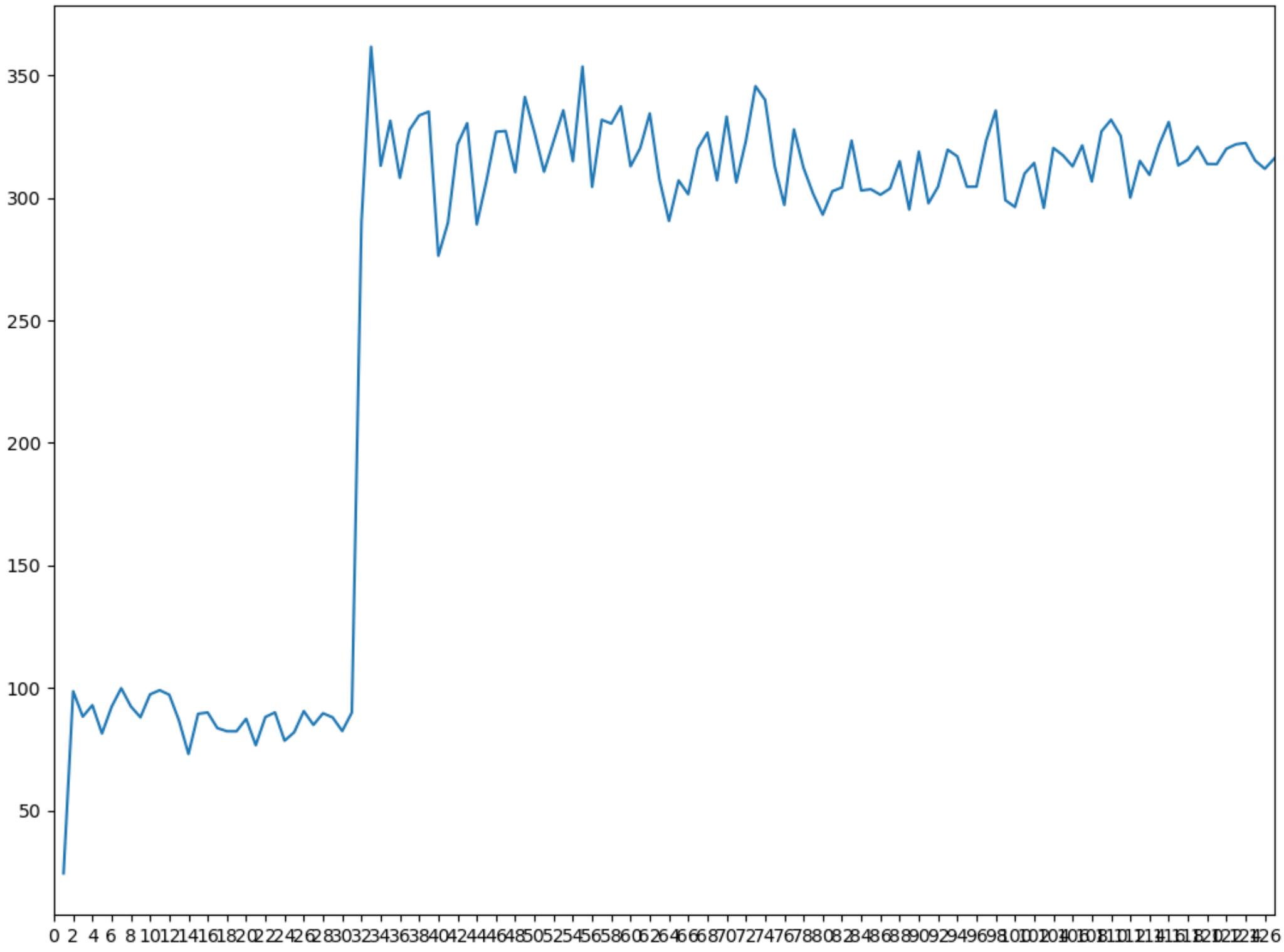
AMD EPYC CORE PING LATENCY (JAVA-JMH)



AMD EPYC CORE PING LATENCY (JAVA-JMH)



INTEL XEON PLATINUM PING LATENCY (JAVA-JMH)



RAM READ THROUGHPUT

A. <500 MB/s

B. <2GB/s

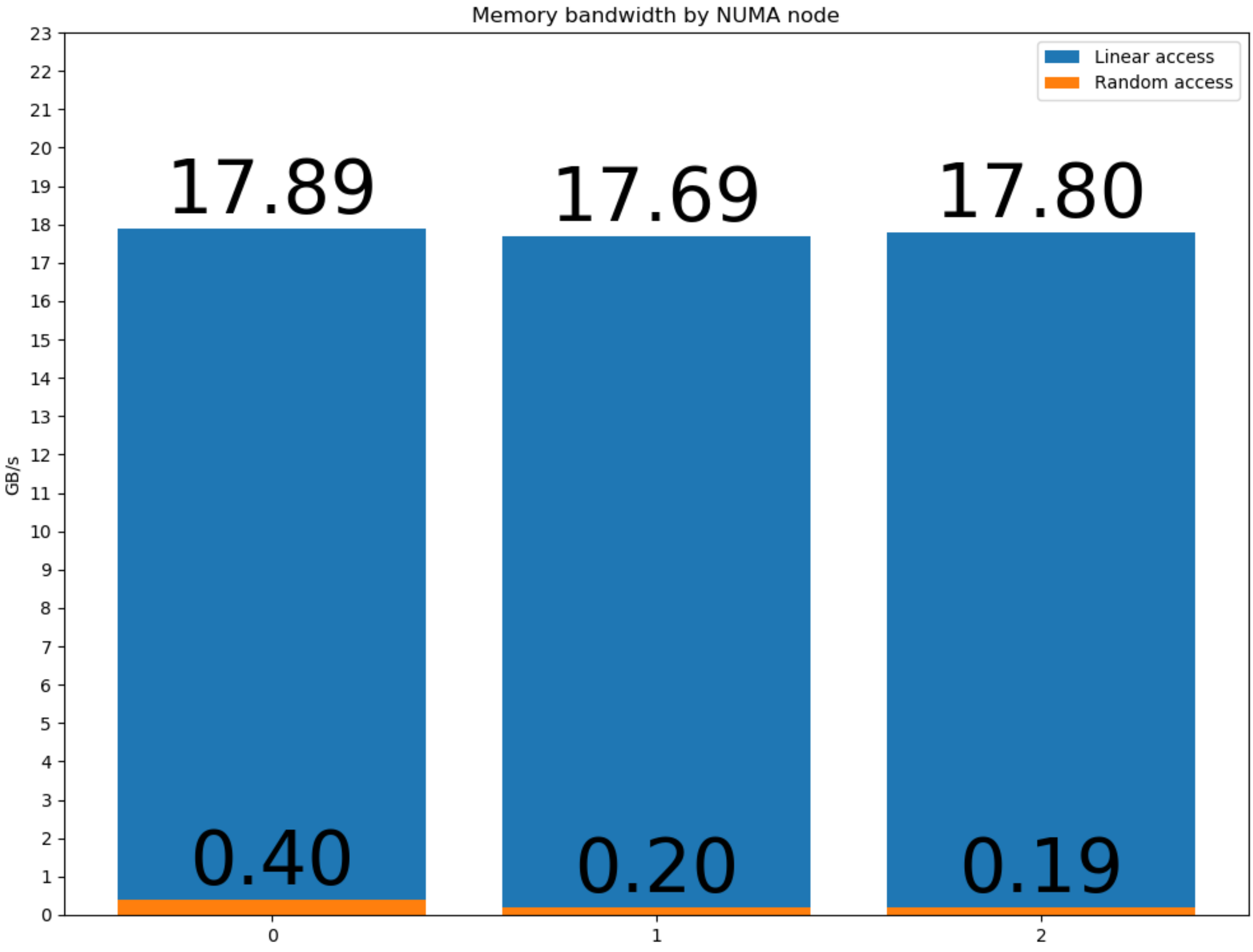
C. <20GB/s

D. ~100GB/s

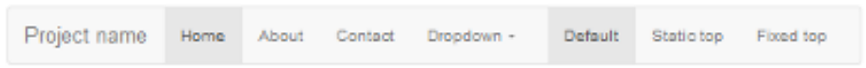
RAM READ THROUGHPUT – PREDICTABLE PATTERN

```
@Benchmark
public int readByteBufferInOrder() {
    Numa.runOnNode(threadNumaNode);

    int result = 0;
    for (int i = 0; i < byteBuffer.limit(); i += 8) {
        result += byteBuffer.getLong(i);
    }

    return result;
}
```

AMD MEMORY BANDWIDTH



A/B TESTING



Welcome to our website

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

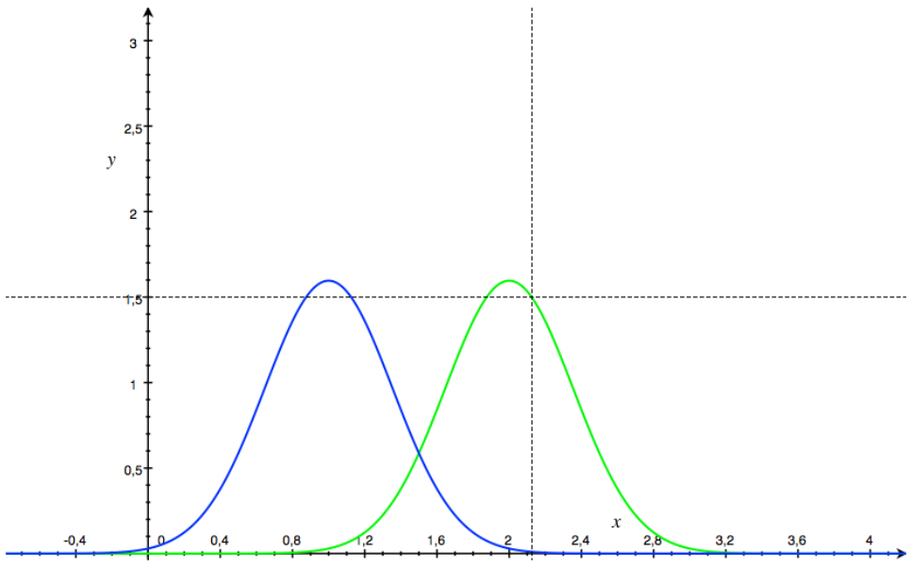
Learn more

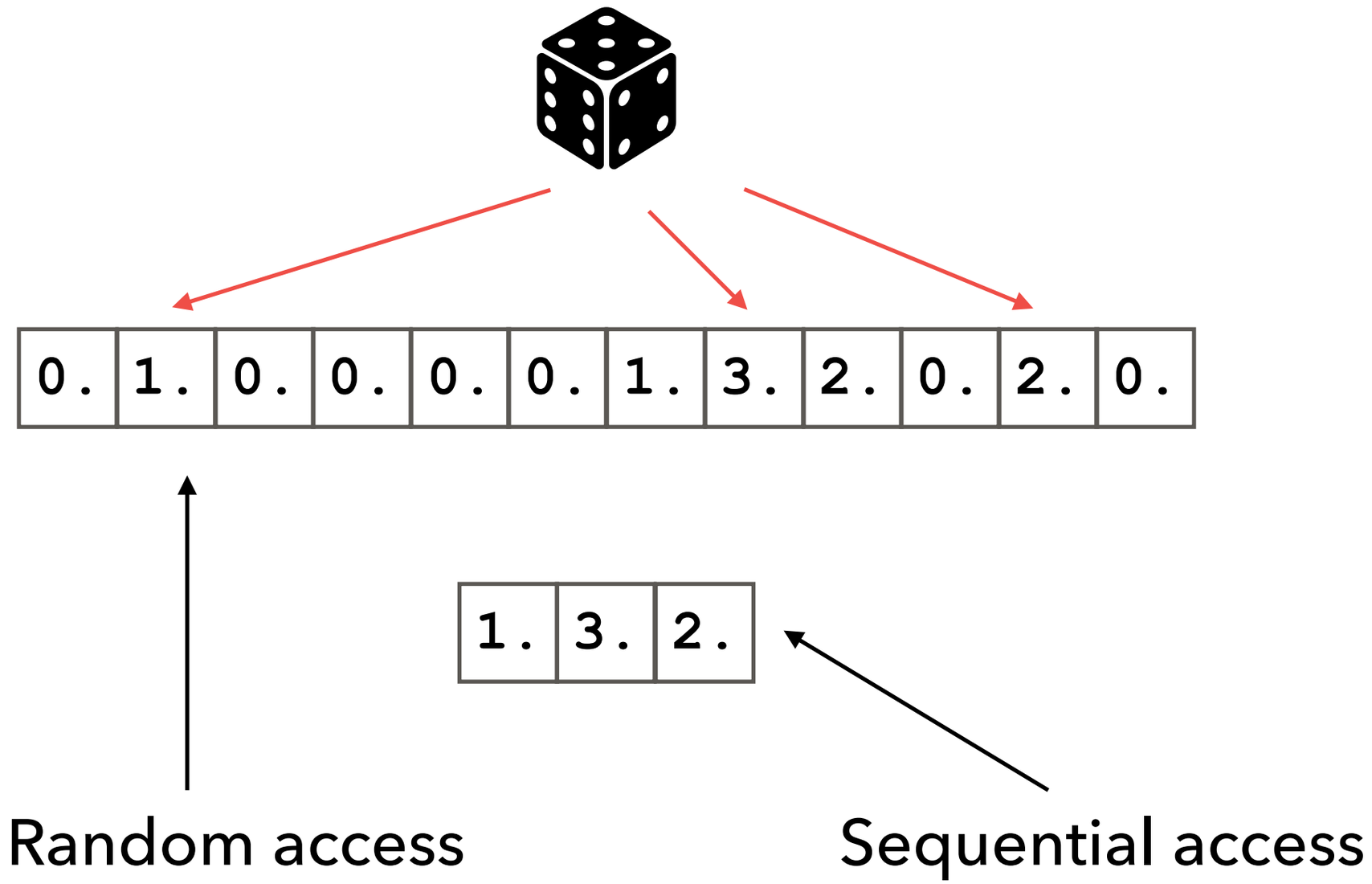
Welcome to our website

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Learn more

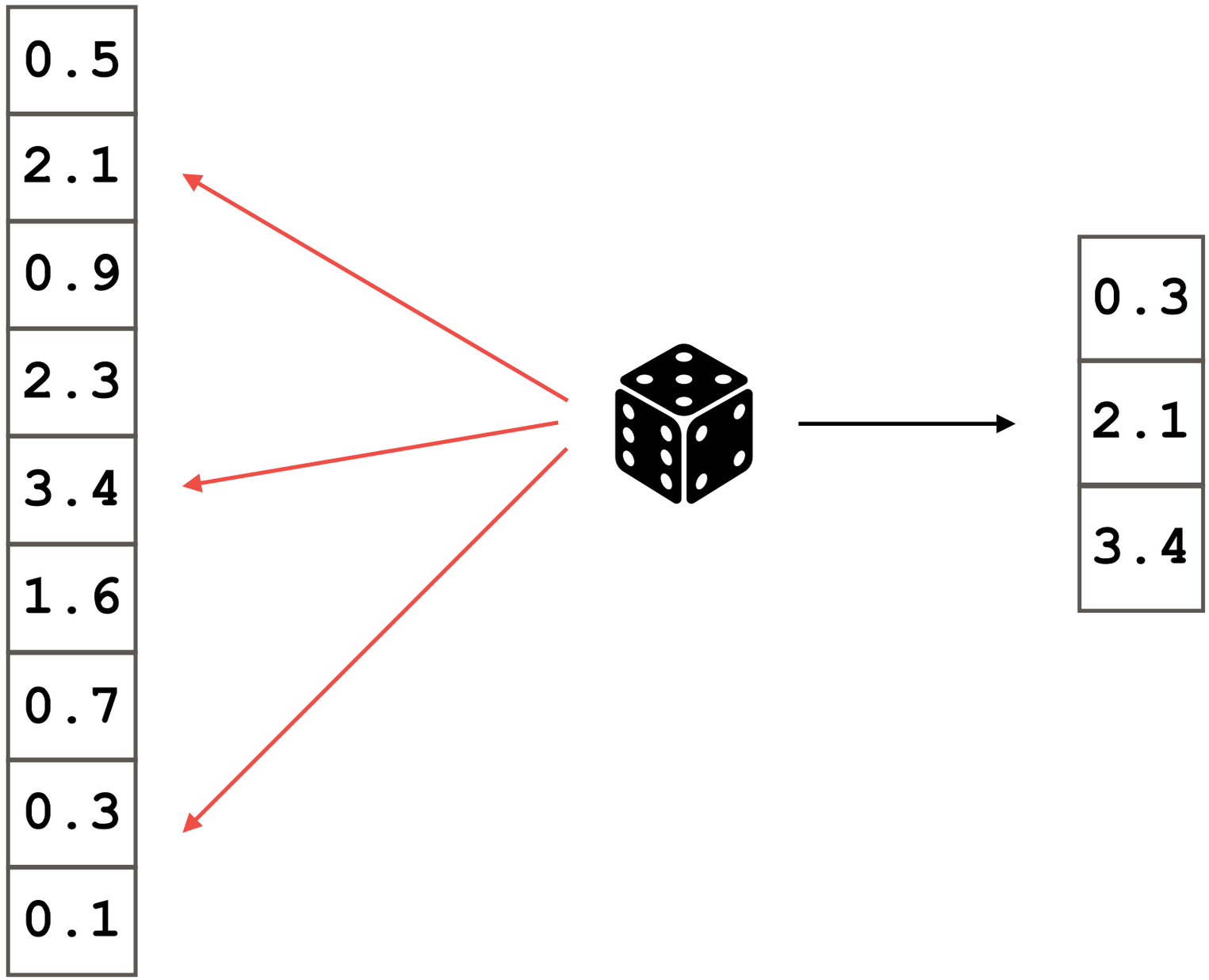
Click rate: 52 % 72 %




0. 1. 0. 0. 0. 0. 1. 3. 2. 0. 2. 0.
1. 3. 2.
Random access
Sequential access

Random access on read

Sequential access on write



```
ThreadLocalRandom random = ThreadLocalRandom.current();
for (int i = 0; i < length; i++) {
    int index = random.nextInt(sampledArraySize);
    double sample = sampledArray[index];

    outputArray[i] = sample;
}
```

2.9ms

```
ThreadLocalRandom random = ThreadLocalRandom.current();
for (int i = 0; i < length; i++) {
    randomArray[i] = random.nextInt(sampledArraySize);
}
```

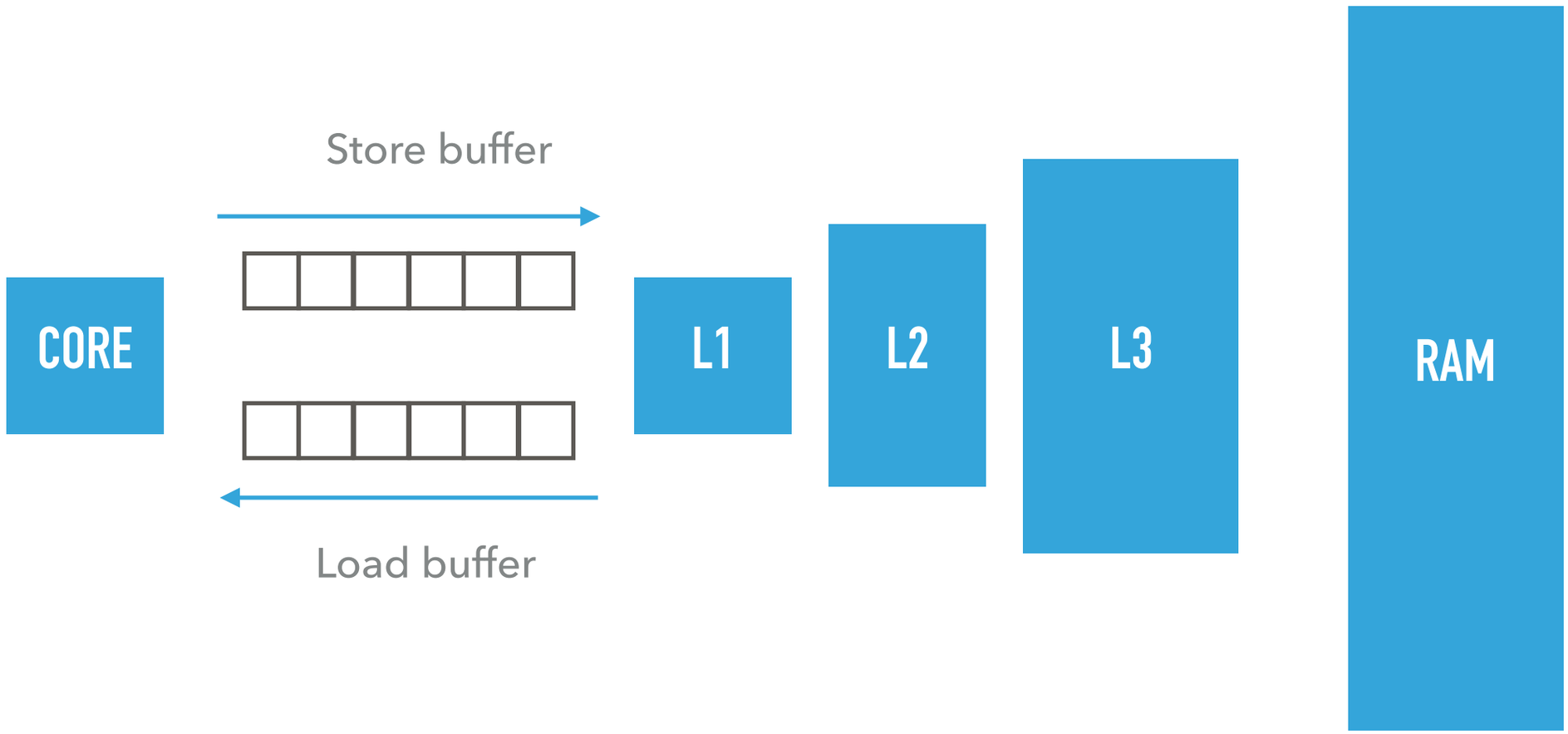
```
for (int i = 0; i < length; i++) {
    int index = randomArray[i];
    double sample = sampledArray[index];
    outputArray[i] = sample;
}
```

2.7ms

```
for (int i = 0; i < length; i++) {
    int index = randomPrecomputedArray[i];
    double sample = sampledArray[index];
    outputArray[i] = sample;
}
```

1.7ms

TEXT



Sequential access on read

0.5
2.1
0.9
2.3
3.4
1.6
0.7
0.3
0.1

Random access on write

7
1
4

Sequential access on read

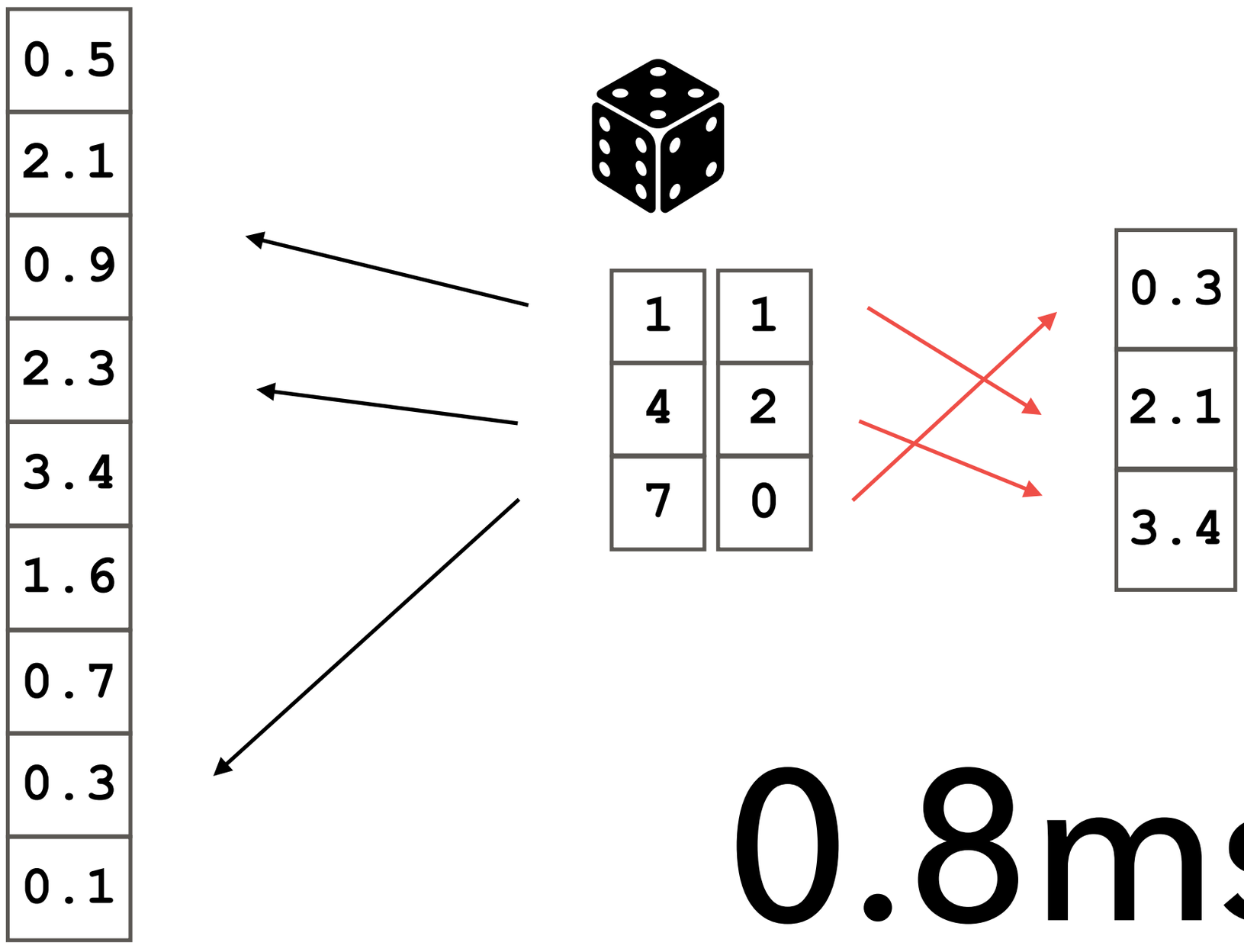
0.5
2.1
0.9
2.3
3.4
1.6
0.7
0.3
0.1

Random access on write

1	1
4	2
7	0

Sequential access on read

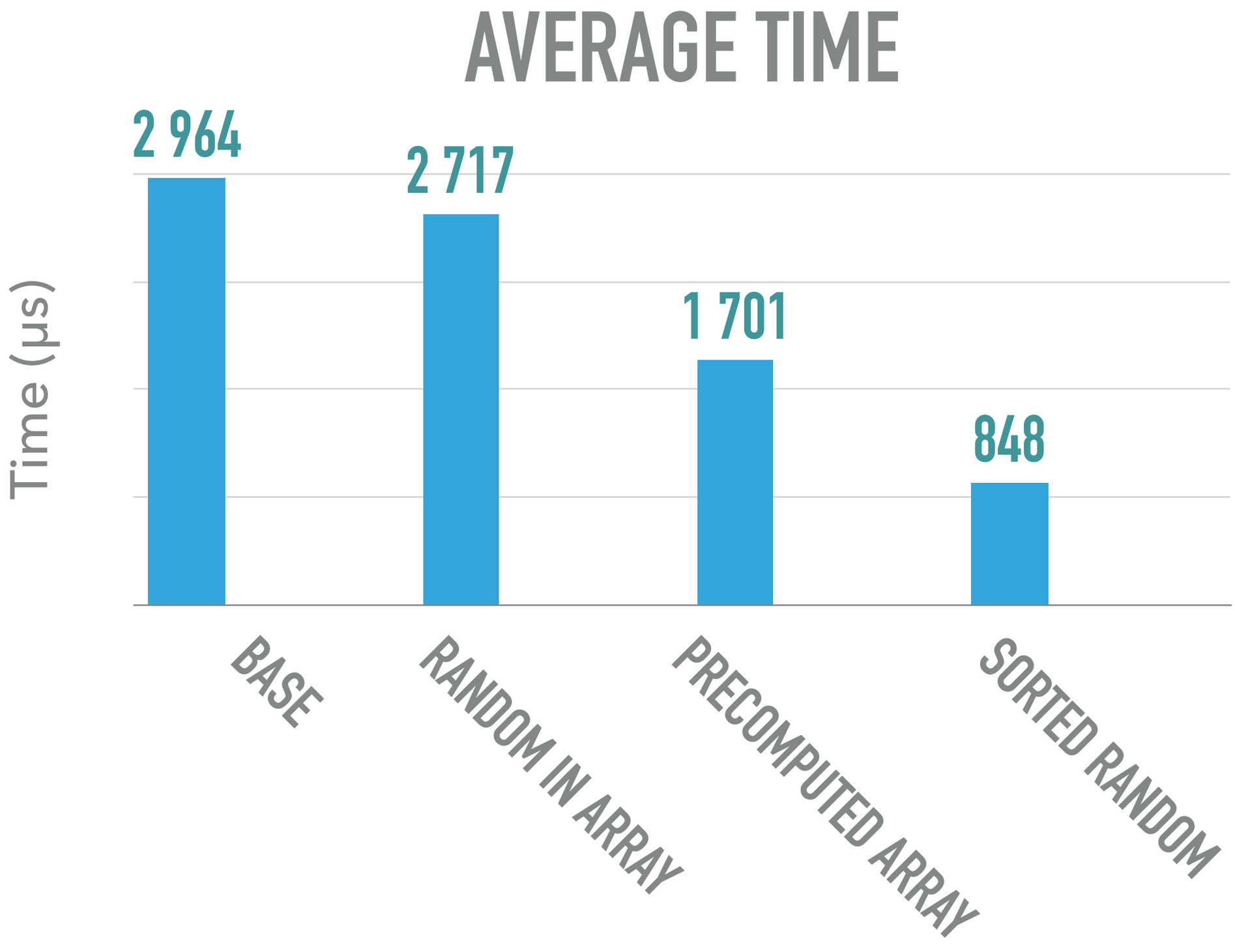
Random access on write

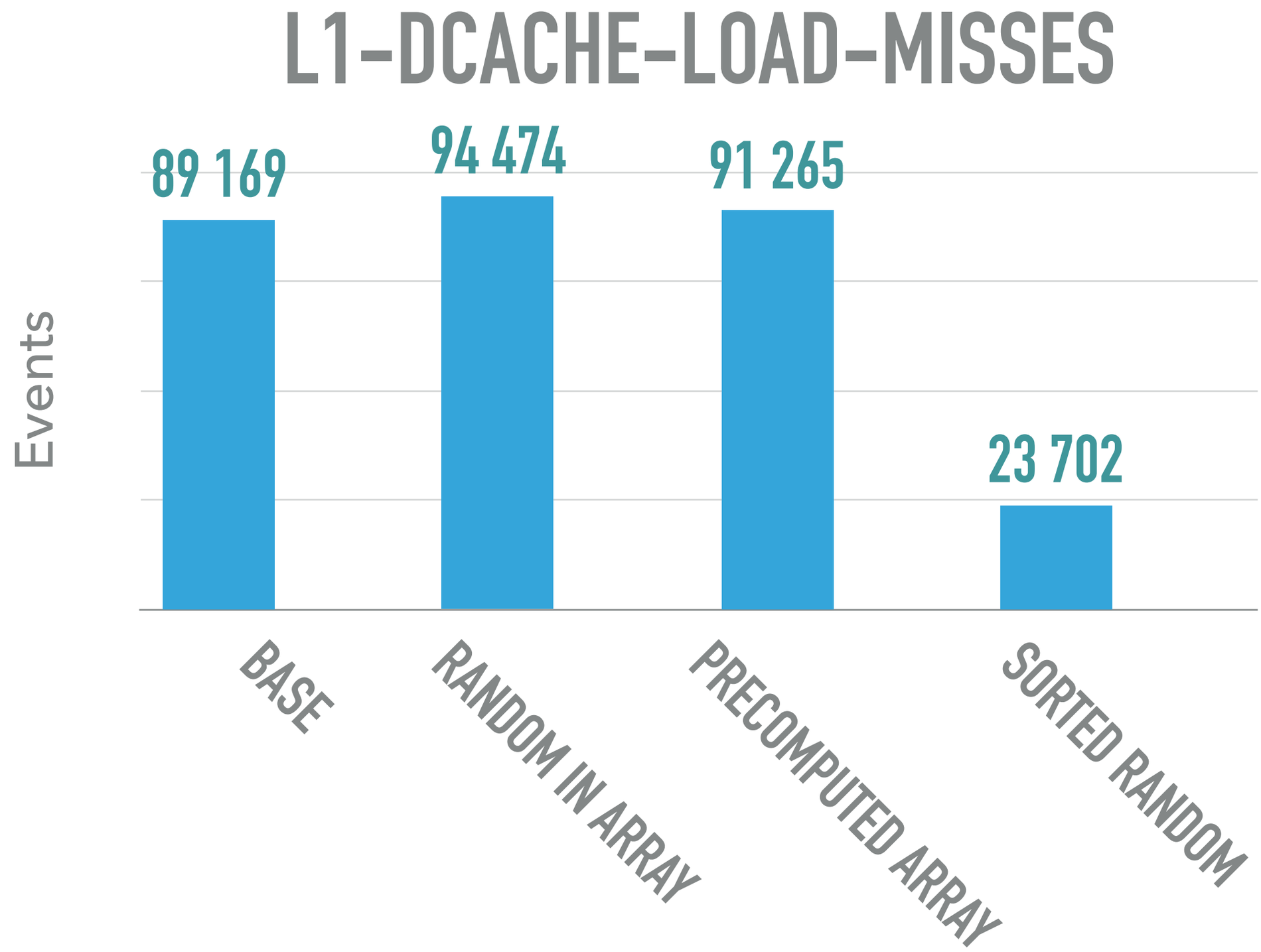


0.8ms

```
for (int i = 0; i < length; i++) {
    long value = randomPrecomputedSotedArray[i];
    int sourceIndex = (int) value;
    int destinationIndex = (int) (value >> 32);

    double sample = sampledArray[sourceIndex];
    outputArray[destinationIndex] = sample;
}
```

0.8ms


AVERAGE TIME
Time (µs)
2 964
2 717
1 701
848
BASE
RANDOM IN ARRAY
PRECOMPUTED ARRAY
SORTED RANDOM


L1-DCACHE-LOAD-MISSES
Events
89 169
94 474
91 265
23 702
BASE
RANDOM IN ARRAY
PRECOMPUTED ARRAY
SORTED RANDOM

RAM THROUGHPUT – READ 400MB OF DATA

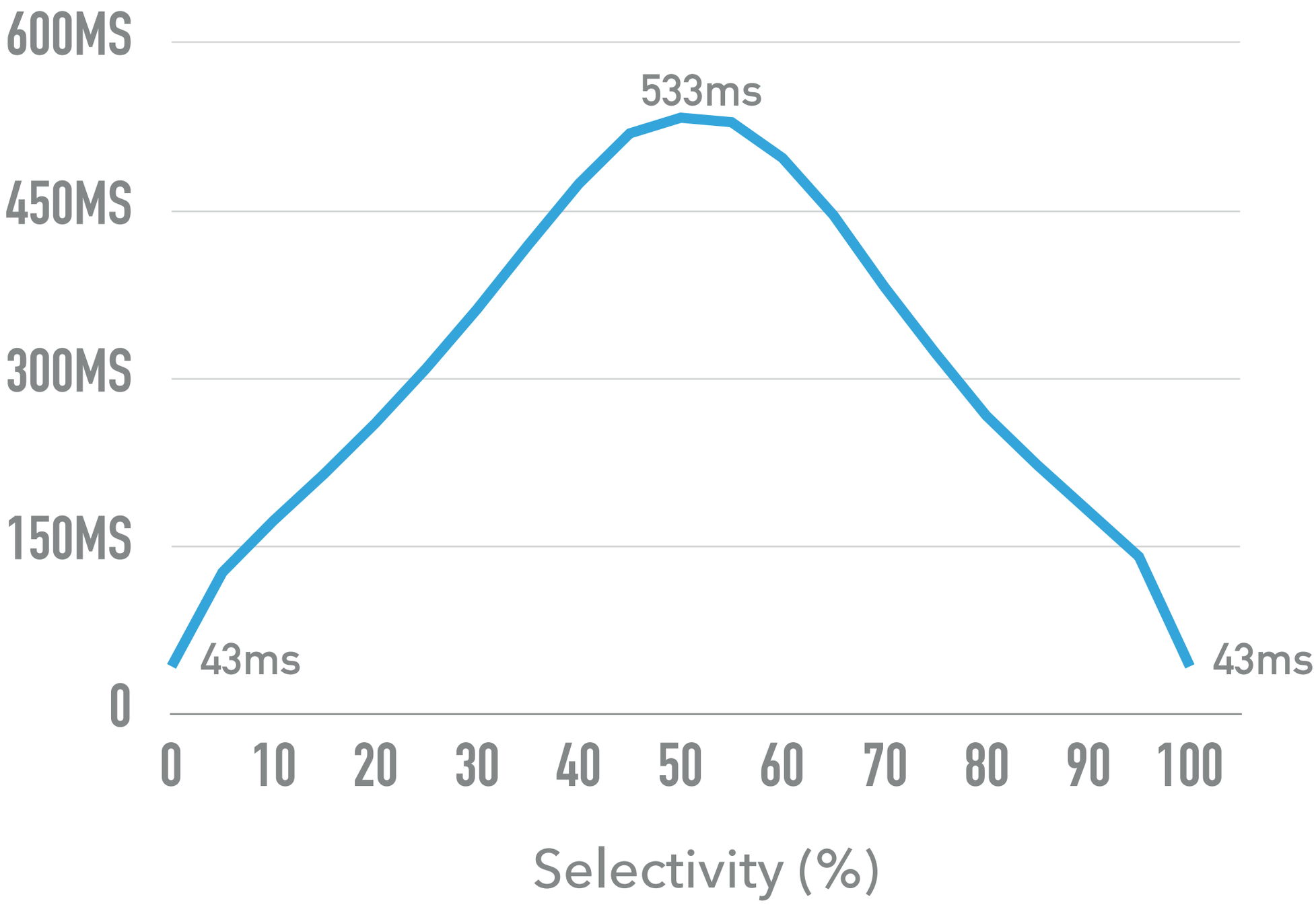
A. ~1s

B. 0.348s

C. 0.150s

D. It depends

MULTI BYTE PROCESSING WITH FULL WORD INSTRUCTIONS



Multiple Byte Processing with Full-Word Instructions

Leslie Lamport
Massachusetts Computer Associates, Inc.

BitWeaving: Fast Scans for Main Memory Data Processing

Yinan Li
University of Wisconsin–Madison
yinan@cs.wisc.edu

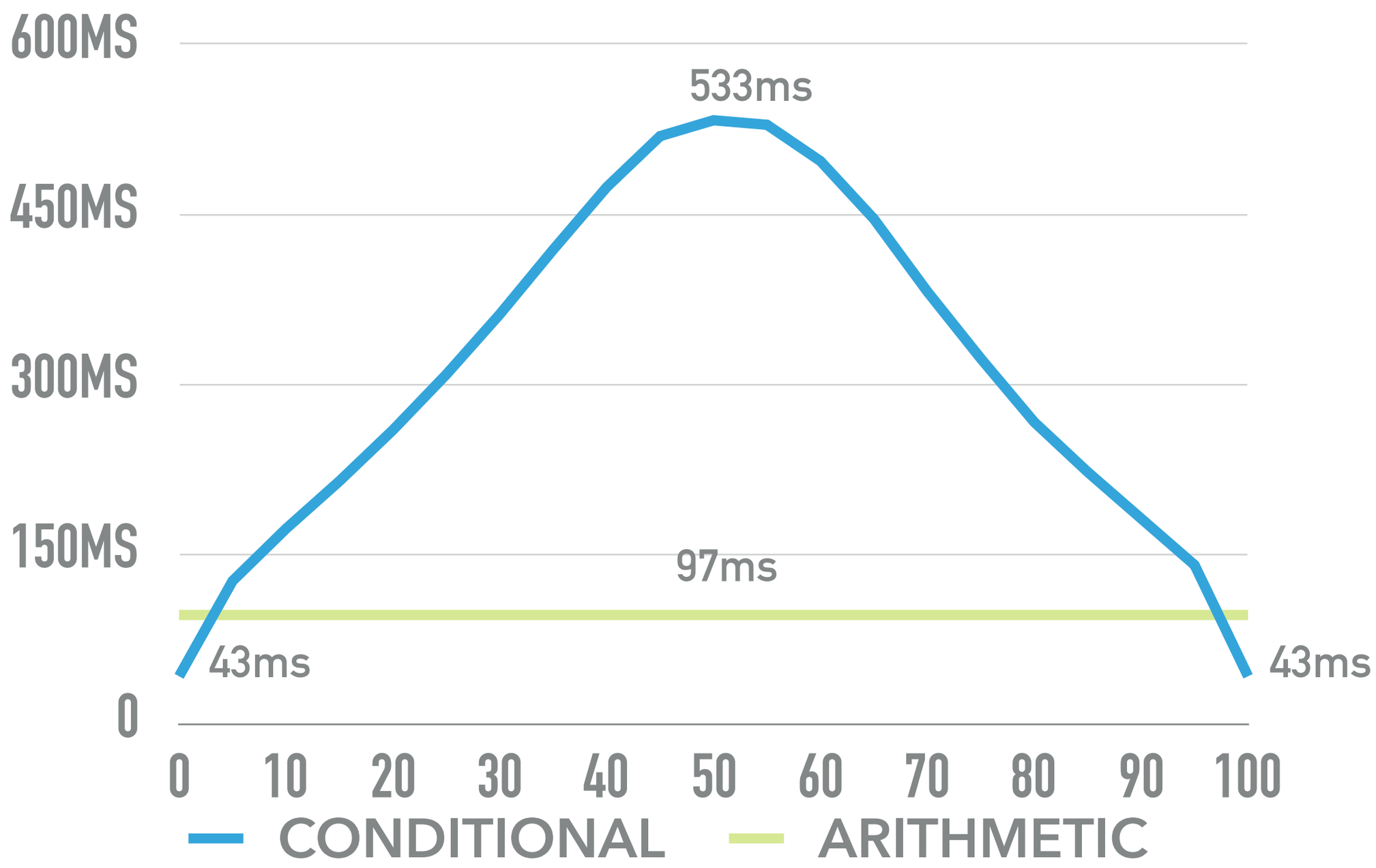
Jignesh M. Patel
University of Wisconsin–Madison
jignesh@cs.wisc.edu

```
MASK          = 0111 1111
~MASK         = 1000 0000
42            = 0010 1010

13            = 0000 1101


13 ^ MASK     = 0111 0010
+ 42          = 1001 1100
&~MASK        = 1000 0000
```

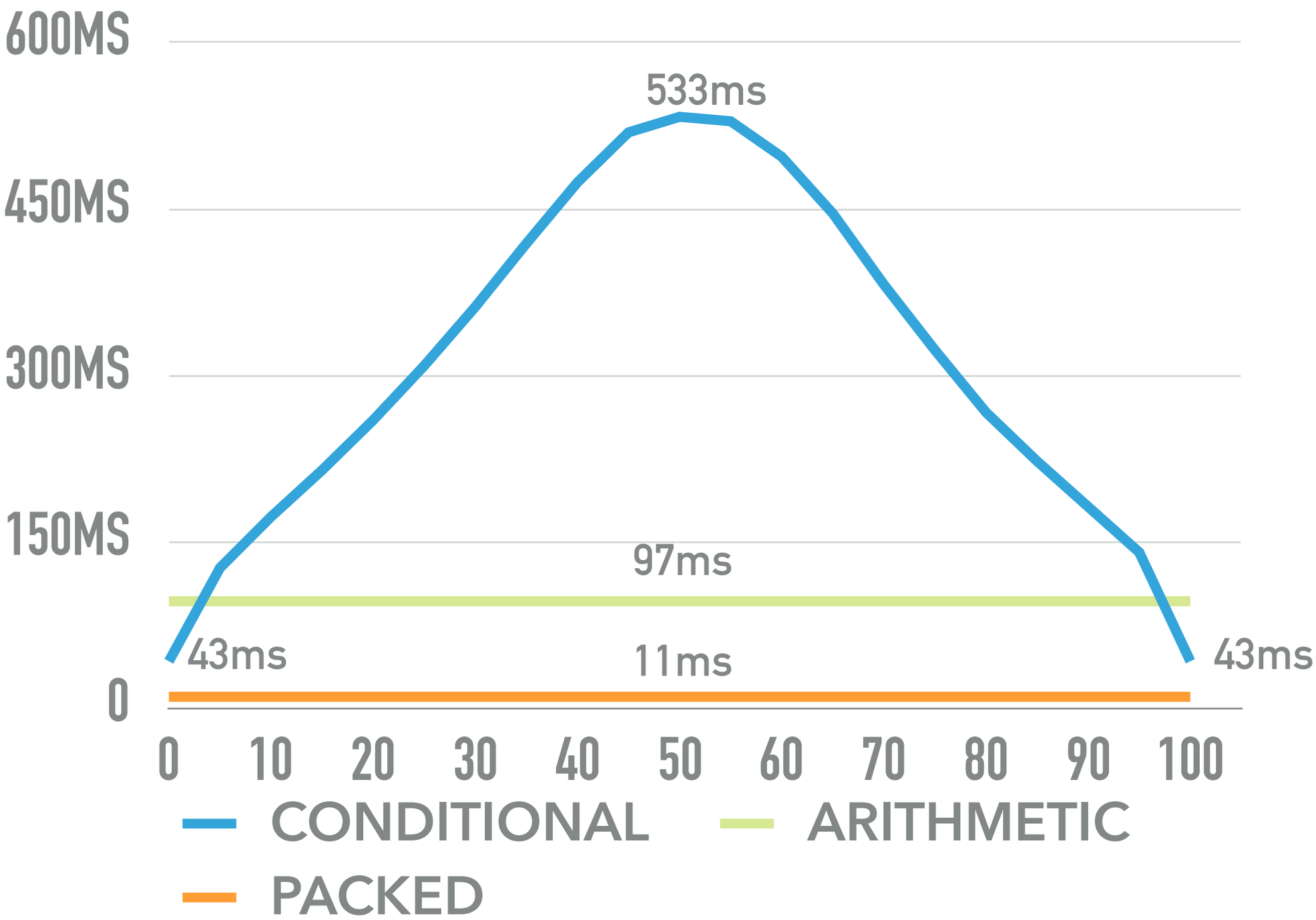
```
@Benchmark
public int plainIf() {
    int result = 0;

    for (int i = 0; i < inputSize; i++) {
        if (onHeapValues.getInt(i * 4) < predicate) {
            result++;
        }
    }

    return result;
}
```

```
@Benchmark
public int plainArithmetic() {
    int result = 0;

    int mask = 0b0111_1111;

    for (int i = 0; i < inputSize; i++) {
        int x = onHeapValues.getInt(i * 4) ^ mask;
        int z = (predicate + x) & ~mask;
        result += Integer.bitCount(z);
    }

    return result;
}
```

MULTI BYTE PROCESSING WITH FULL WORD INSTRUCTIONS



MULTI BYTE PROCESSING WITH FULL WORD INSTRUCTIONS



BRANCHLESS DOOM

This directory provides a branchless, mov-only version of the classic DOOM video game.

DOOM, running with only mov instructions.

BRANCHLESS DOOM

This directory provides a branchless, mov-only version of the classic DOOM video game.

“The mov-only DOOM renders approximately one frame every 7 hours, so playing this version requires somewhat increased patience.”

DOOM, running with only mov instructions.

- COMPUTE IS GETTING FASTER
- DEVELOPERS ARE GETTING MORE EXPENSIVE
  - (GOOD :))
- SOFTWARE & LIBRARIES ARE GETTING SLOWER
- OPPORTUNITIES AHEAD

- O(NLOGN) ALGORITHM IS 50 000 TIMES FASTER THAN O(N^2)
- DO WE WANT TO ADD 50 000 TIMES MORE SERVERS OR OPTIMISE?

REFERENCES

- Repository with examples:
  - https://github.com/tkowalcz/beaker
- Efficiently Compiling Efficient Query Plans for Modern Hardware
- Constant-Time Query Processing
- https://danluu.com/branch-prediction/
- Multiple Byte Processing with FullWord Instructions
- http://www.vldb.org/2017/keynotes.php

COPYRIGHTS

- Houston opportunities by XKCD.com
- A/B Testing example: https://en.wikipedia.org/wiki/A/B_testing#/media/File:A-B_testing_example.png by Maxime Lorant
- AMD for EPYC diagrams