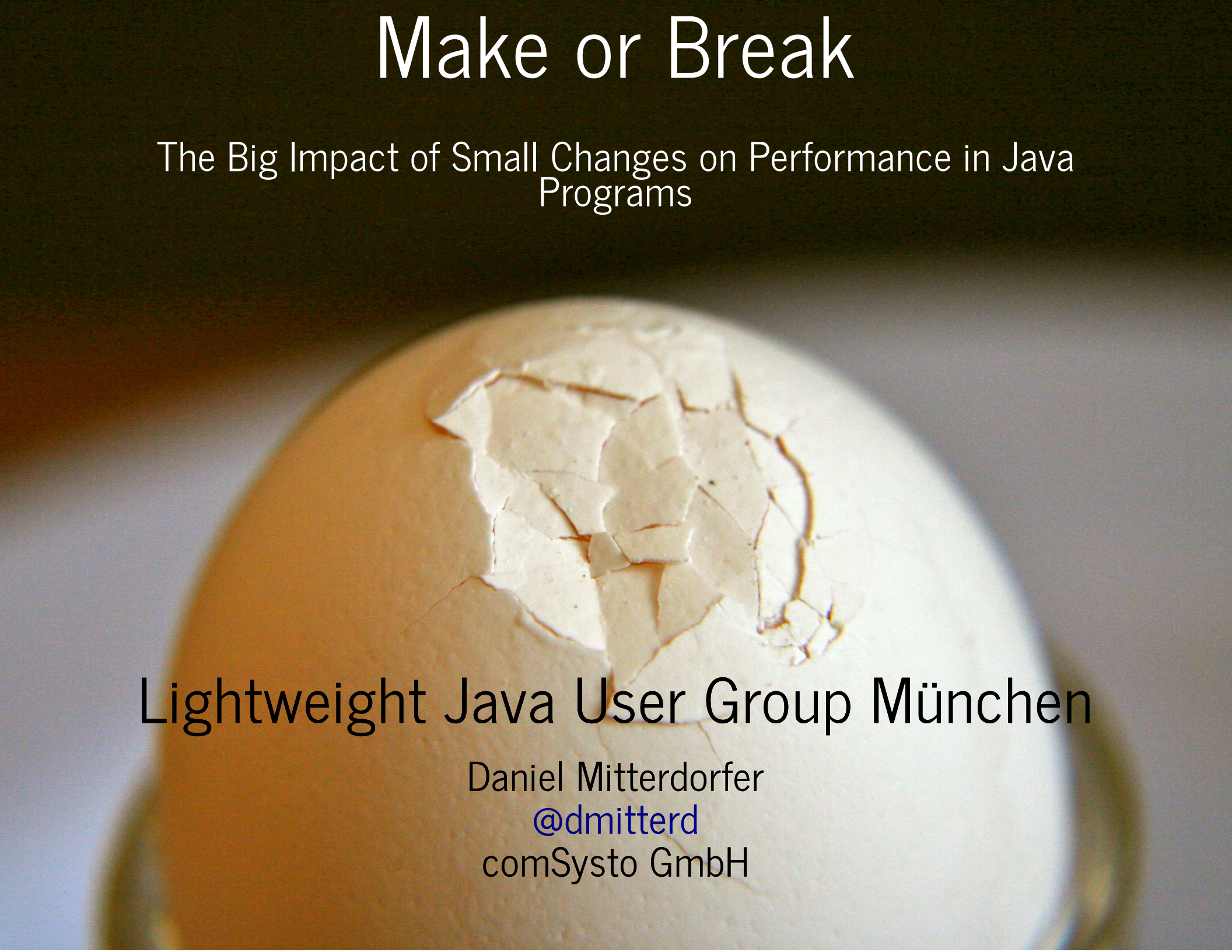
Make or Break

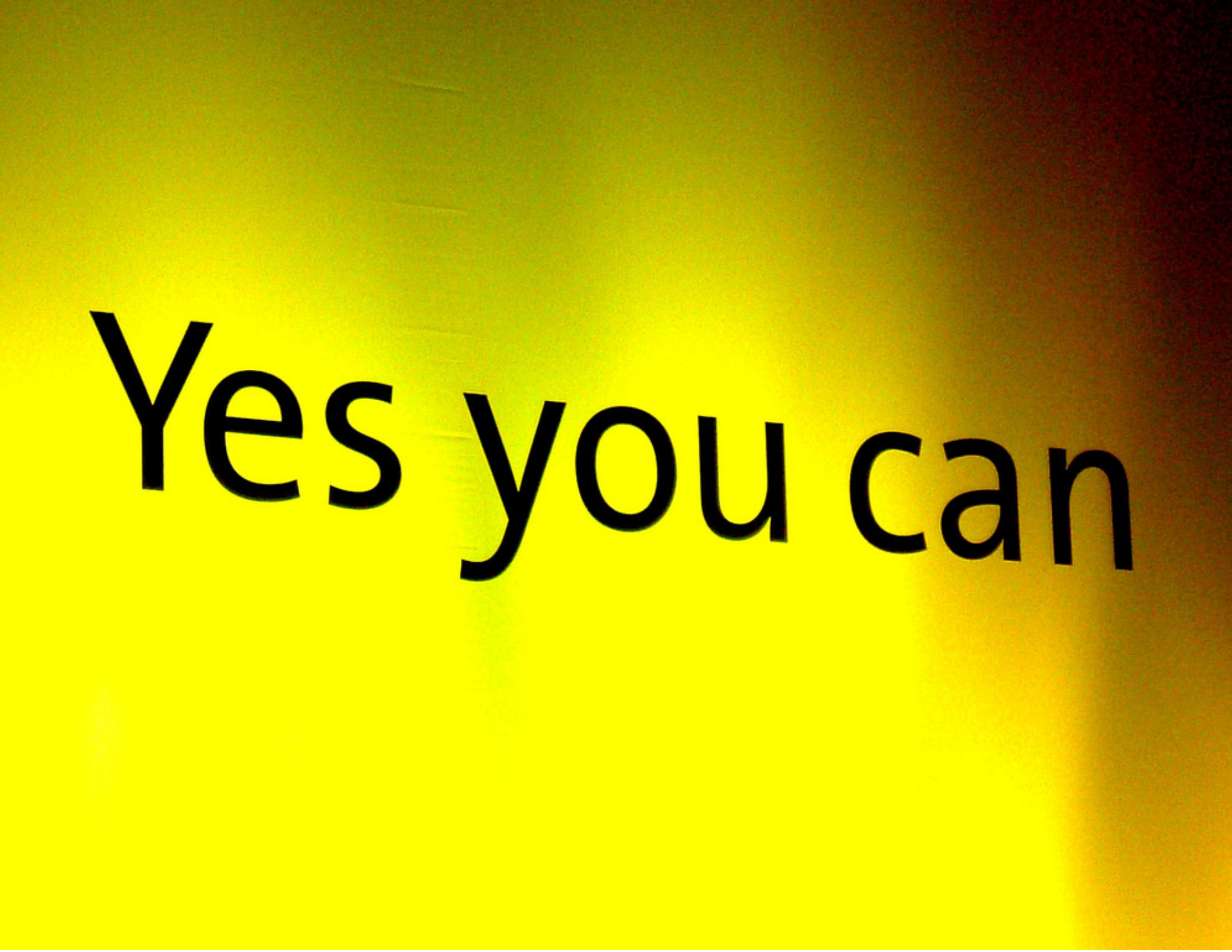
The Big Impact of Small Changes on Performance in Java Programs



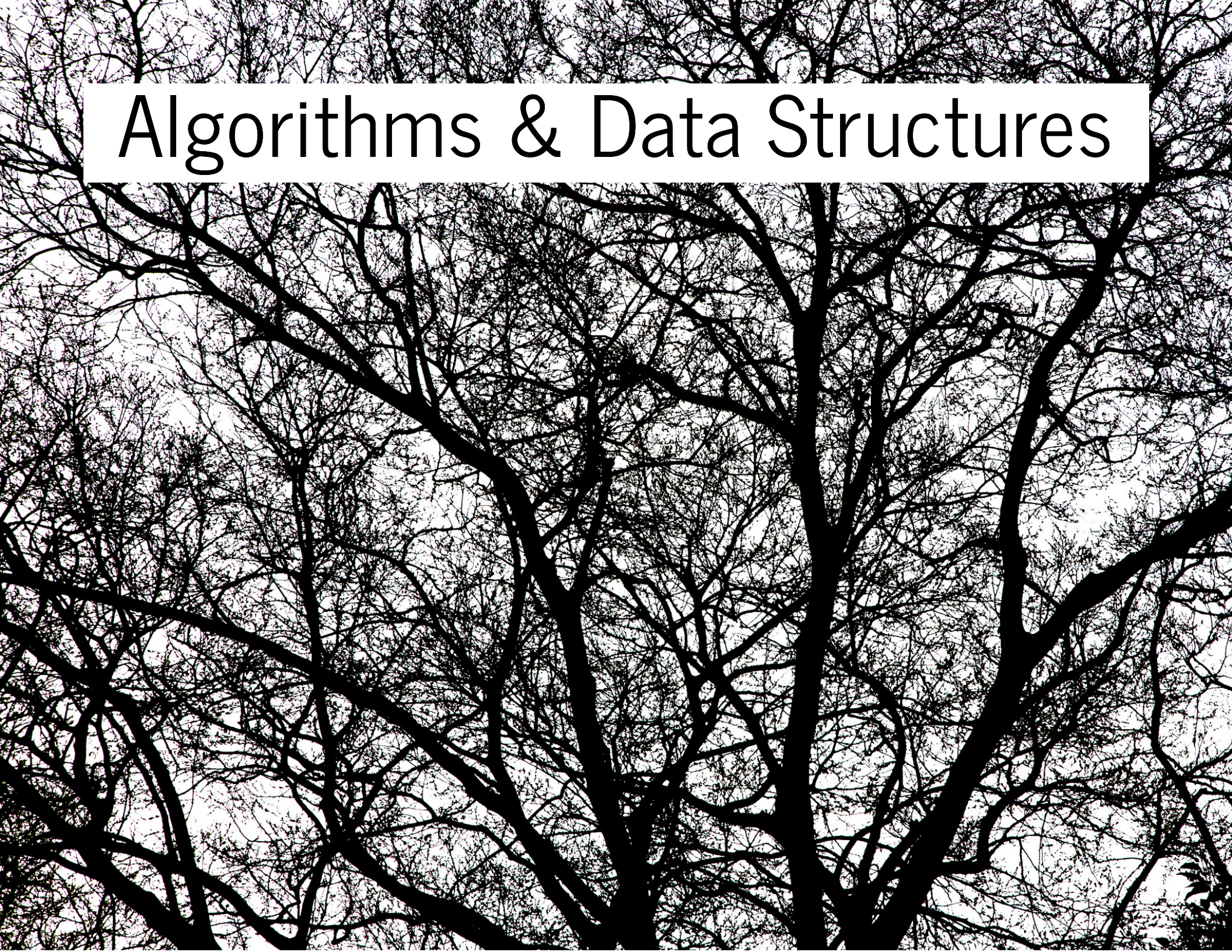
Lightweight Java User Group München

Daniel Mitterdorfer
@dmitterd
comSysto GmbH

Some ingredients for faster software

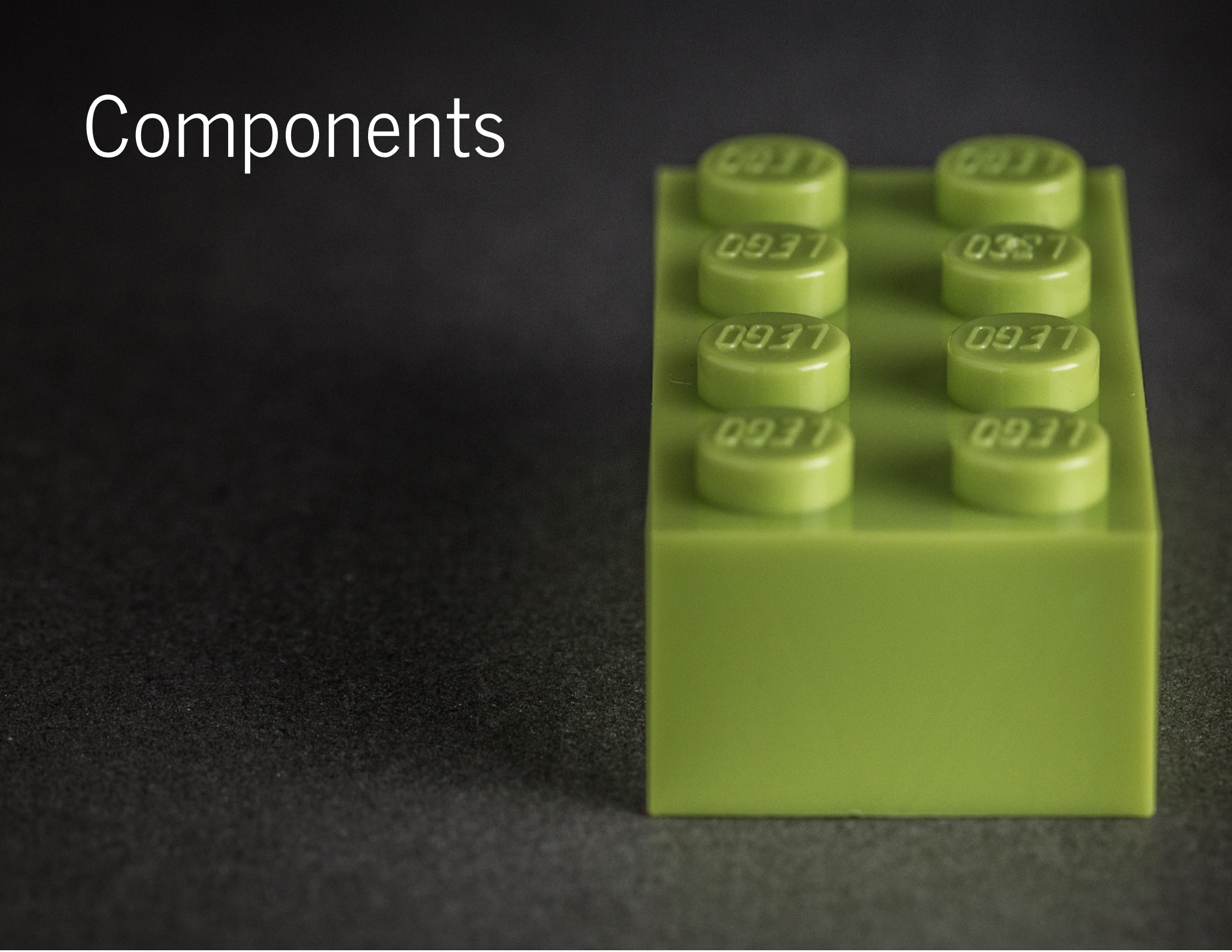

Yes you can

Algorithms & Data Structures




Play Tricks

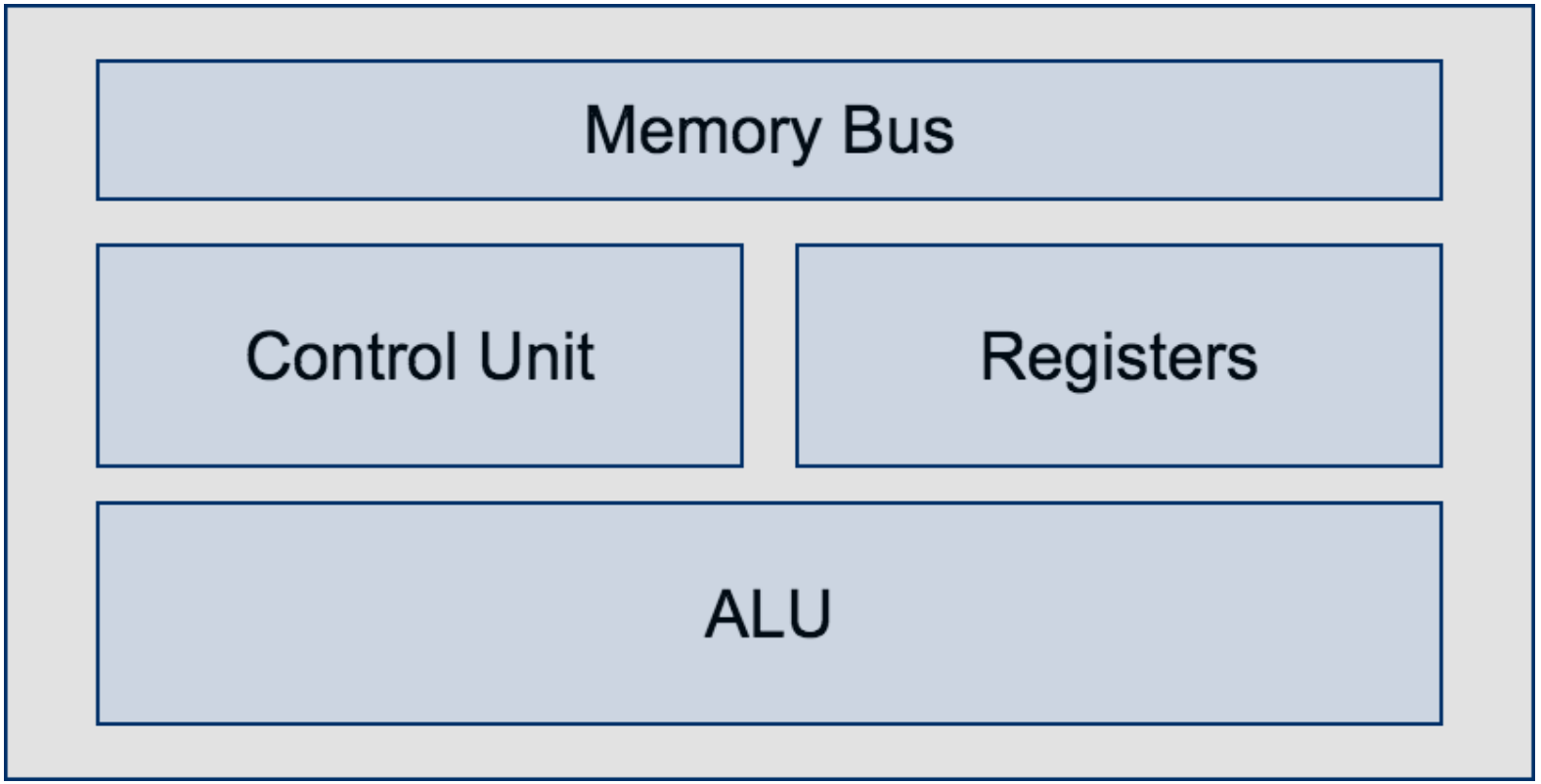
Components



Mechanical Sympathy

Coined by Martin Thompson

What's in a CPU?

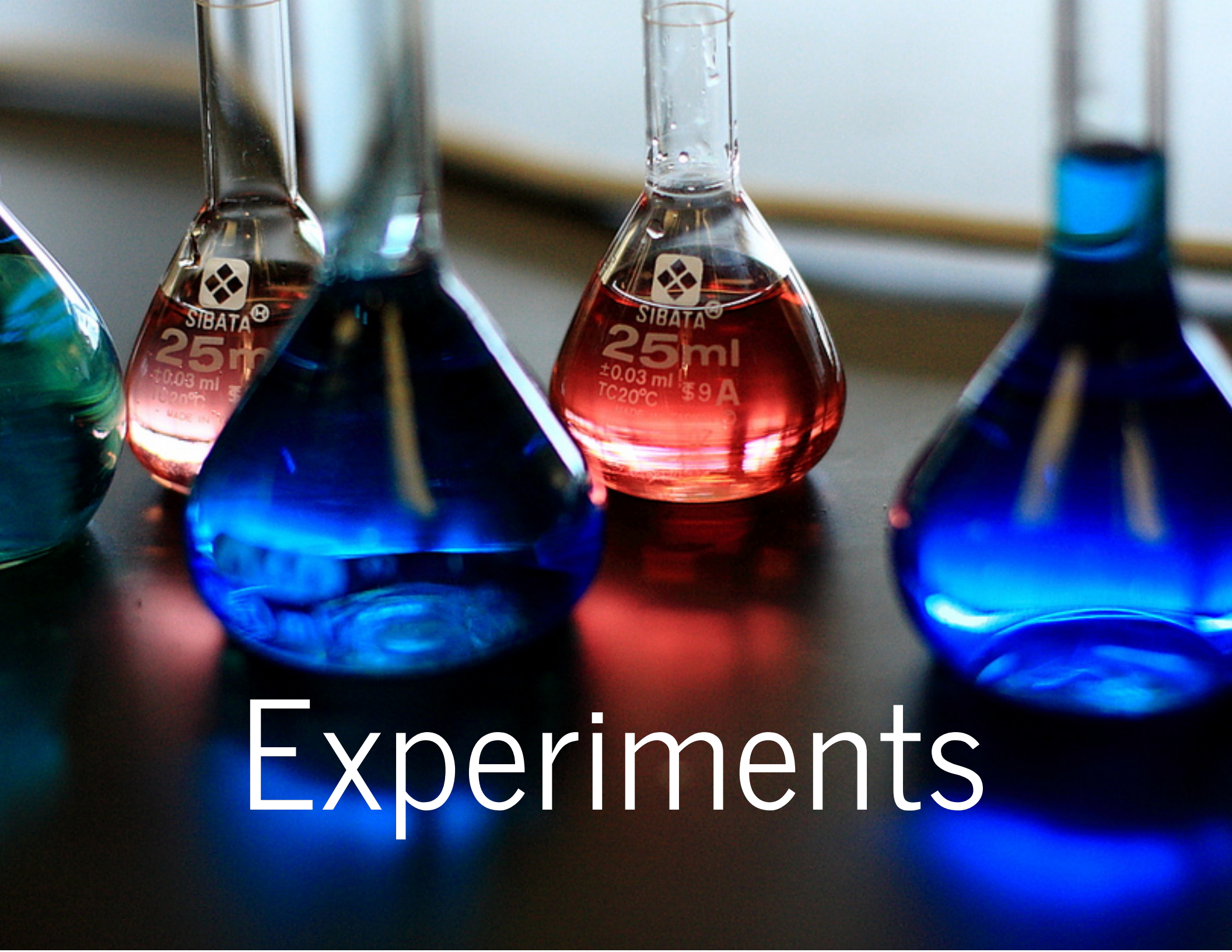


That was in the (early) 80s

Cache Hierarchies


Pipelining

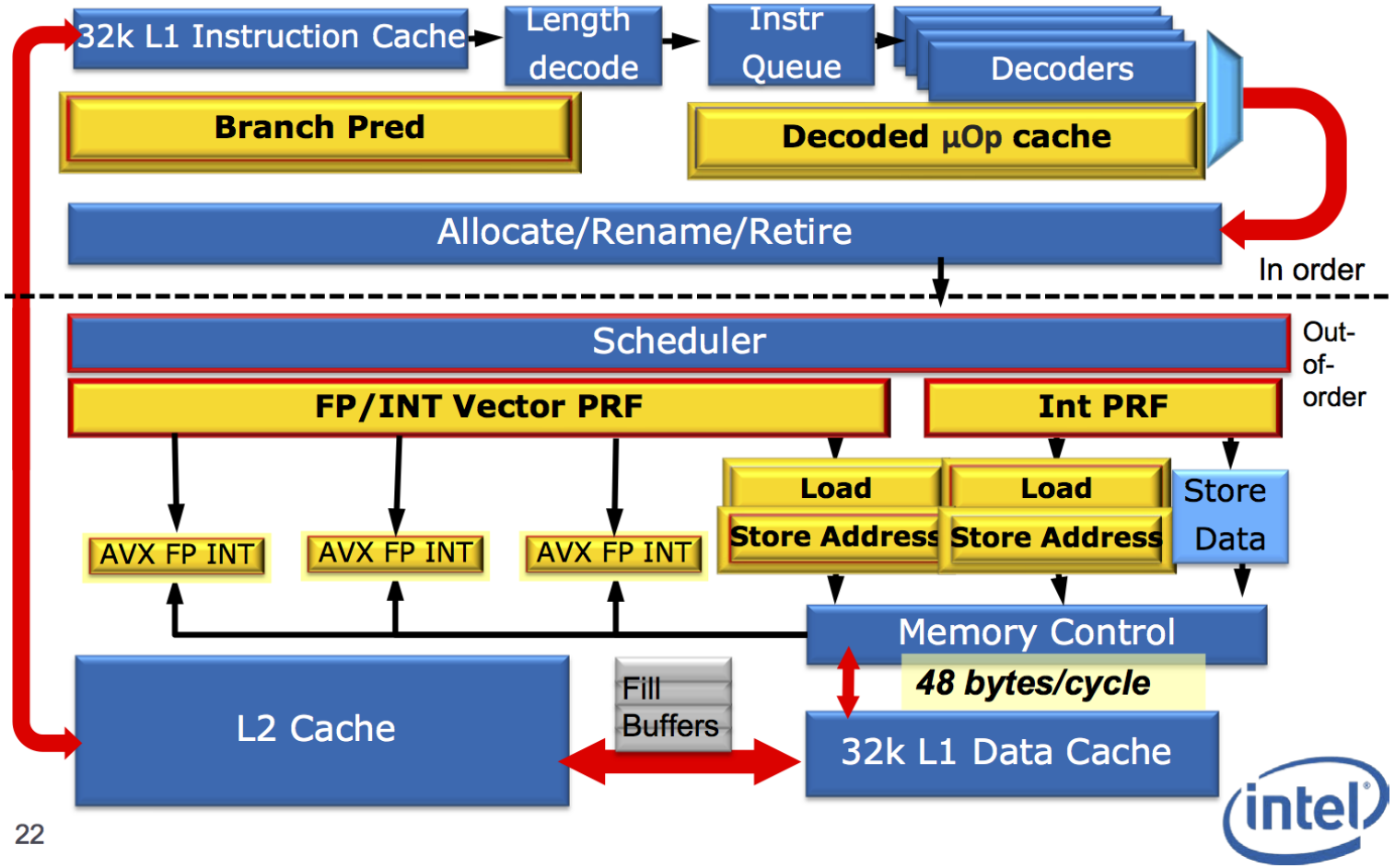
Multicore



Experiments

Hardware: x86

i7-2635QM (Sandy-Bridge quad-core)

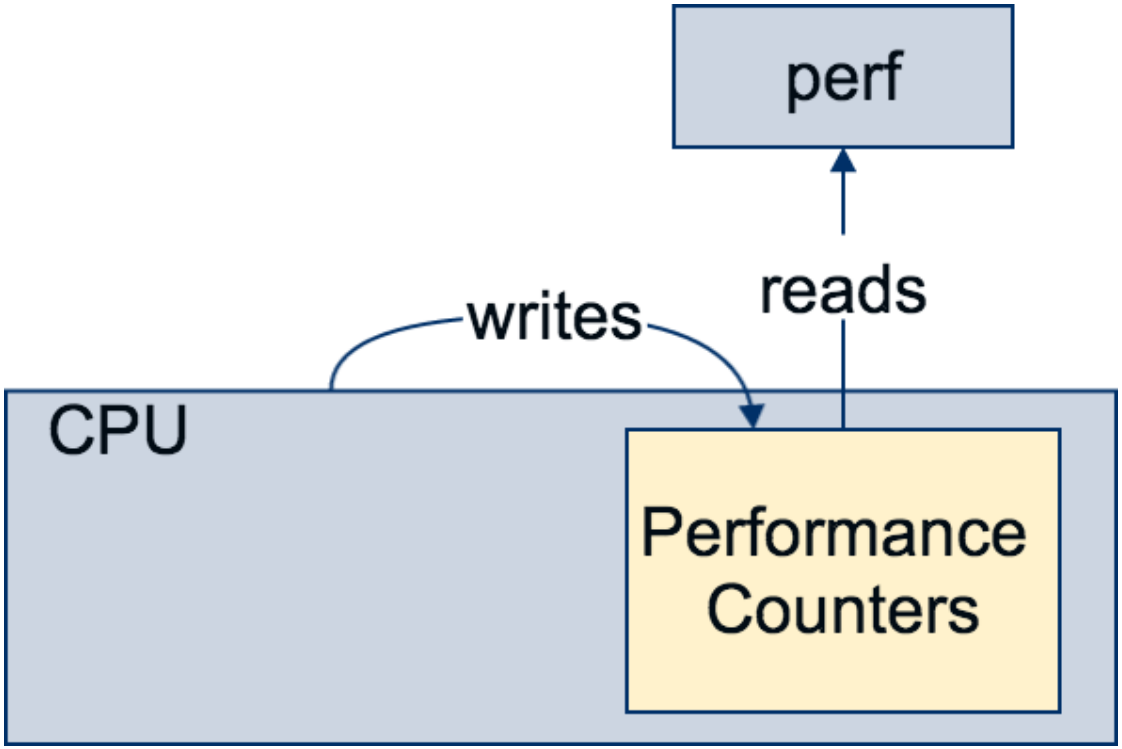


Software: JMH

- **The** microbenchmarking framework on the JVM
- In a nutshell: Add `@Benchmark`; JMH takes care of the rest*
- `perf` support

* ok, it's not that easy but you get the gist

man perf



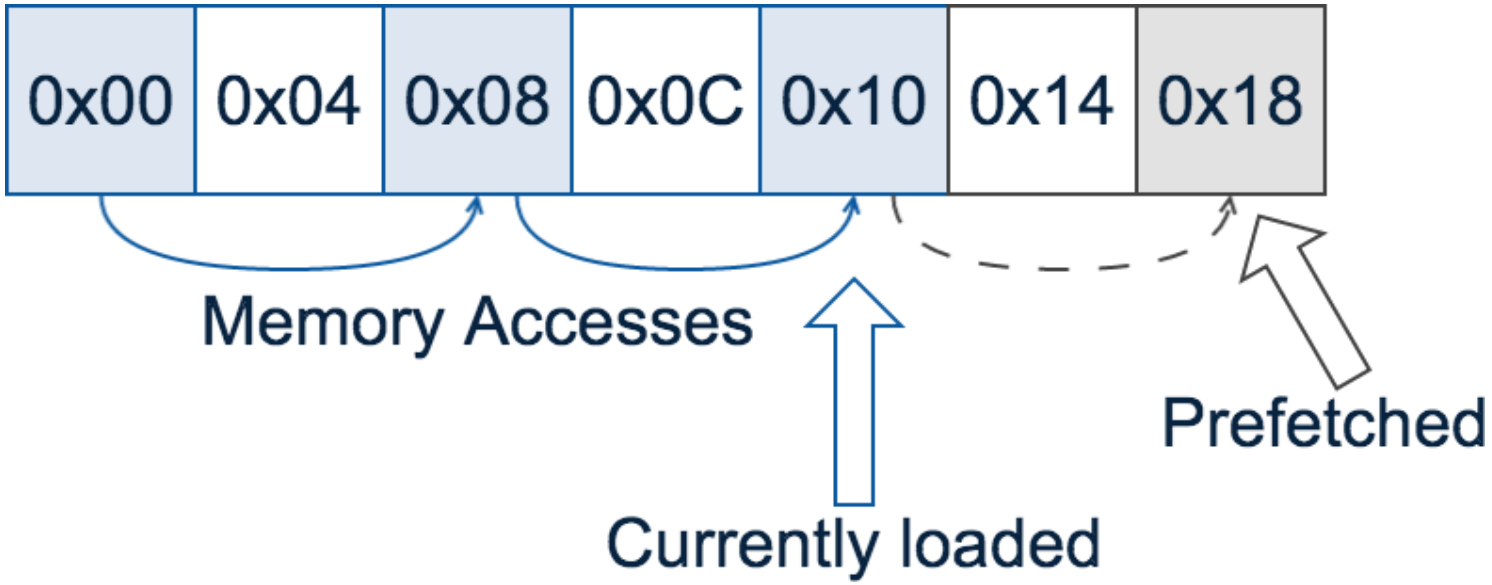
Experiments

- To Prefetch or not to Prefetch?
- False Sharing
- Puzzling Branch Prediction

To Prefetch or not to Prefetch?

Prefetching Unit

CPU speculatively loads data based on memory access patterns

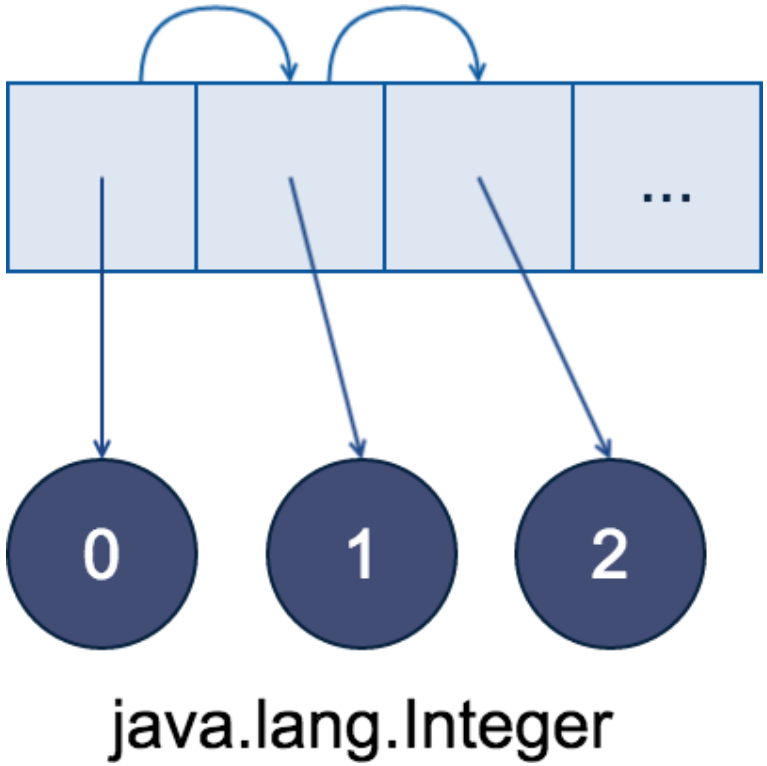


Contenders: `int[]`

Contiguous array: Linear memory access pattern for traversal:

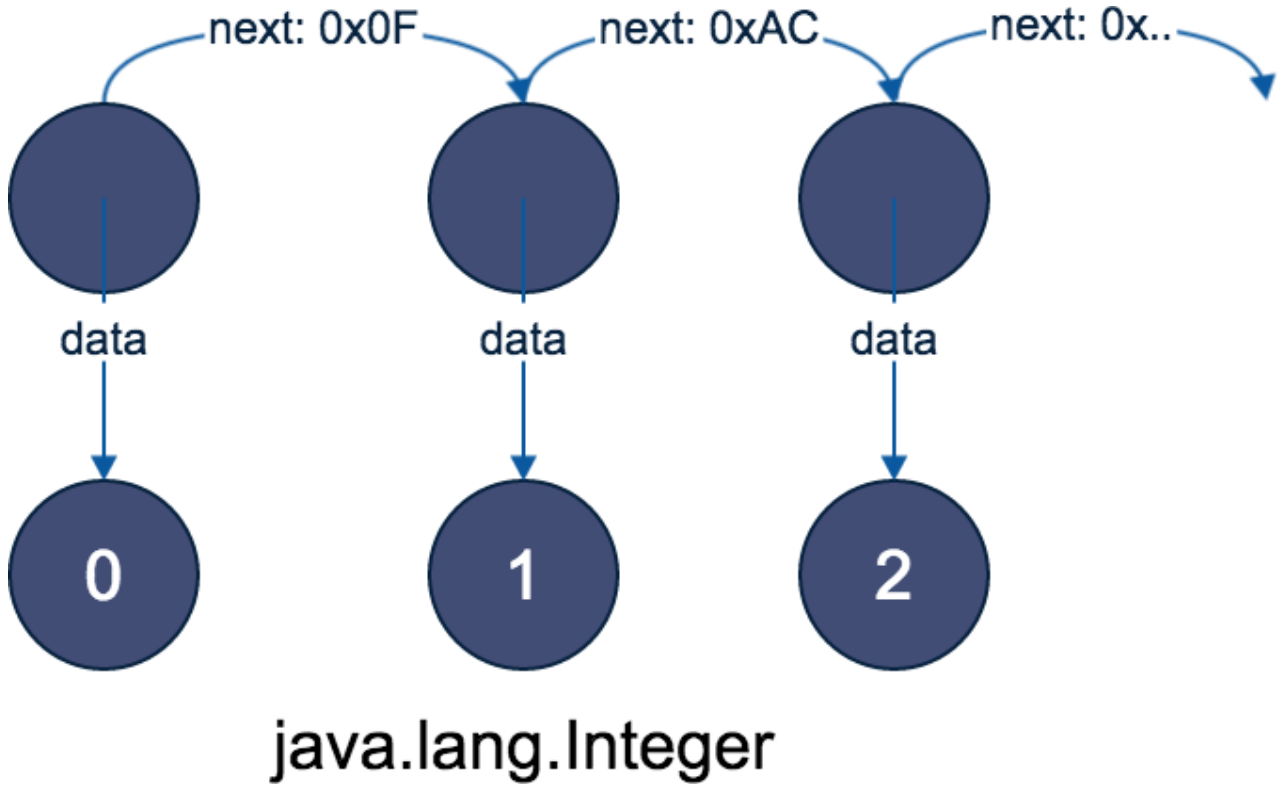
Contenders: `ArrayList`

Linear memory access pattern for array traversal; pointer chasing for elements:



Contenders: `LinkedList`

Nonlinear memory access pattern for traversal and elements:



Experiment Setup

- Task: Calculate the sum of all elements

Benchmark: Setup

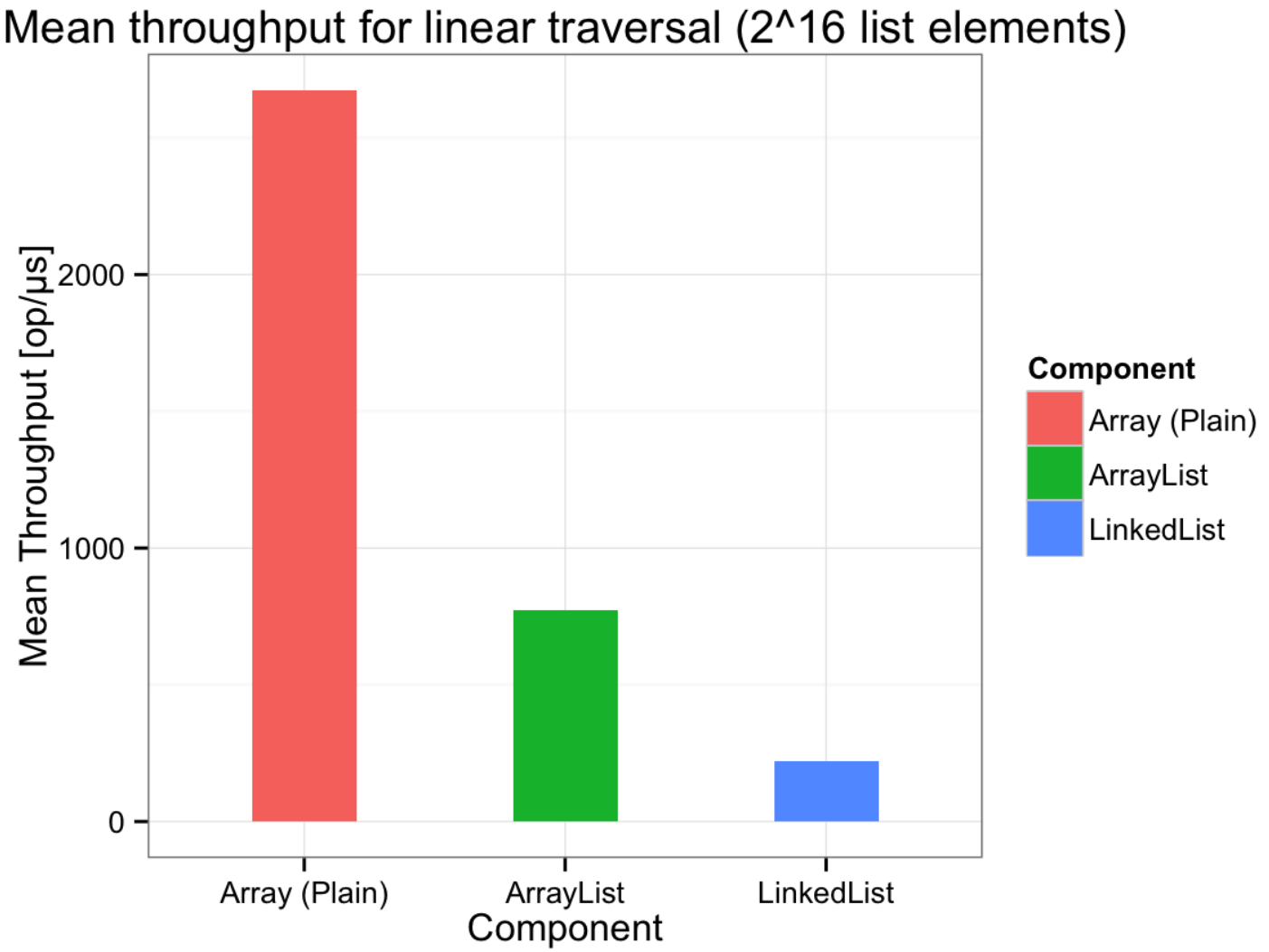
```
@State(Scope.Benchmark)
public class PointerChasingBenchmark {
  private static final int PROBLEM_SIZE = 64 * 1024; /* 64 K elements */
  private final int[] array = new int[PROBLEM_SIZE];
  private final List<Integer> linkedList = new LinkedList<>();
  private final List<Integer> arrayList = new ArrayList<>(PROBLEM_SIZE);

  @Setup
  public void setUp() {
      for (int idx = 0; idx < PROBLEM_SIZE; idx++) {
          linkedList.add(idx);
          arrayList.add(idx);
          array[idx] = idx;
      }
  }
  // ...
}
```

Benchmark: `LinkedList`

```
@State(Scope.Benchmark)
public class PointerChasingBenchmark {
  // .. Setup ..

  @Benchmark
  @OperationsPerInvocation(PROBLEM_SIZE)
  public long sumLinked() {
    long sum = 0;
    for (int val : linkedList) {
      sum += val;
    }
    return sum;
  }
}
```

Note: the other benchmark methods are identical except for their type

Results


Mean throughput for linear traversal (2^16 list elements)
Mean Throughput [op/µs]
2000
1000
0
Array (Plain)
ArrayList
LinkedList
Component
Component
Array (Plain)
ArrayList
LinkedList

Why the difference?

Read CPU performance monitoring data with JMH's `perf` profiler

Metric	`int[]`	`ArrayList`	`LinkedList`
L1-dcache-loads	$67 * 10^9$	$57 * 10^9$	$25 * 10^9$
L1-dcache-load-misses (relative to L1 cache hits)	6 %	11 %	20 %

Conclusion

Pointer indirection renders prefetching ineffective

Take Aways and Suggestions

- Memory access patterns matter: Prefer linear access
- Watch Oracle's work on Value Objects (JEP 169)

False Sharing

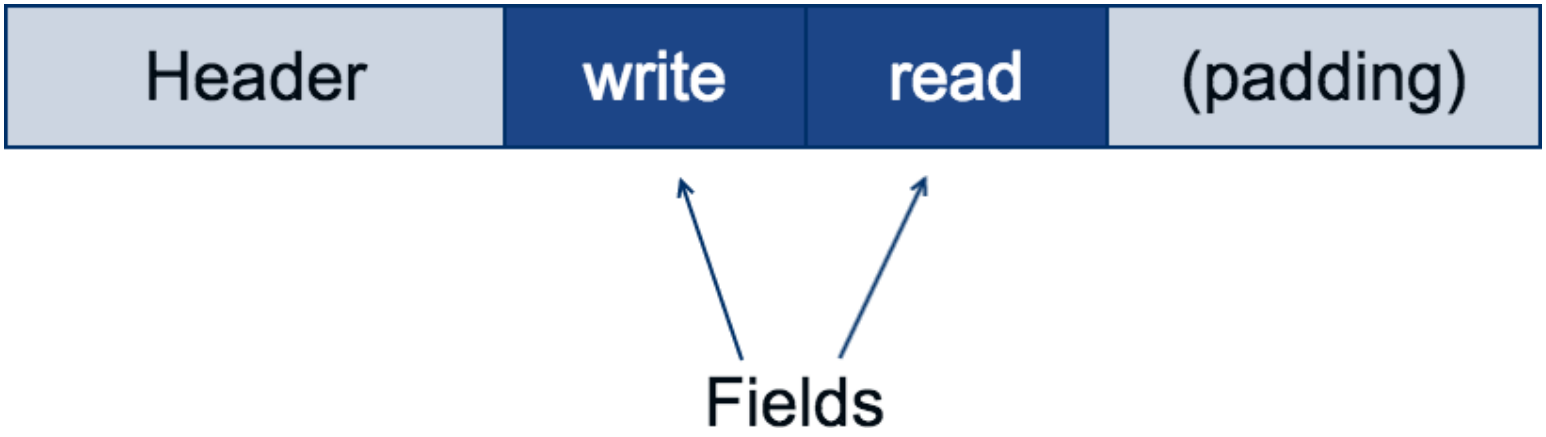
Experiment Setup

- Task: Three readers and a writer access two unrelated fields of a shared object

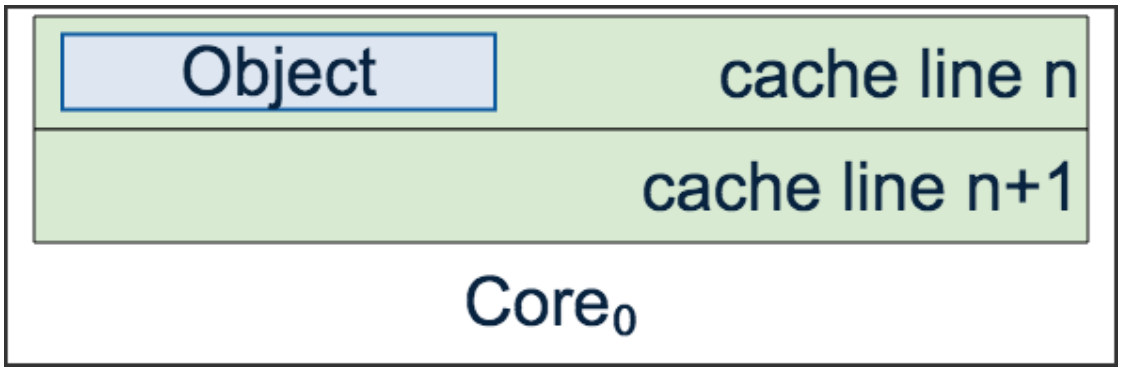
Benchmark

```
@Threads(4)
public class FalseSharingMicroBenchmark {
  @State(Scope.Benchmark)
  public static class FalselySharedState {
    public long write;
    public long read;
  }

  @Group("false_sharing")
  @GroupThreads(1)
  @Benchmark
  public void produce(FalselySharedState s) {
    s.write++;
  }

  @Group("false_sharing")
  @GroupThreads(3)
  @Benchmark
  public long consume(FalselySharedState s) {
    return s.read;
  }
}
```

A Java Object in Memory



The Processor's View

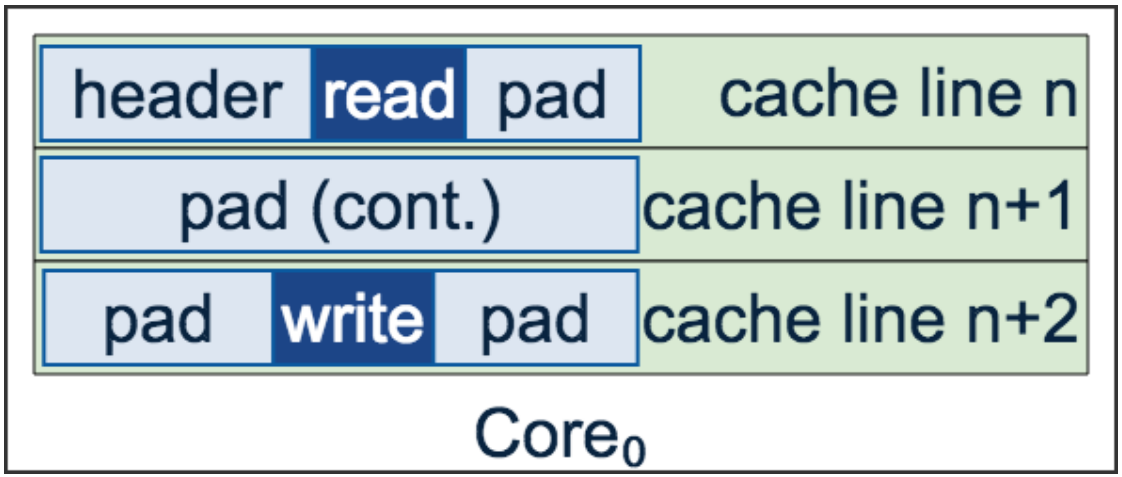


False Sharing

False Sharing

Countermeasures

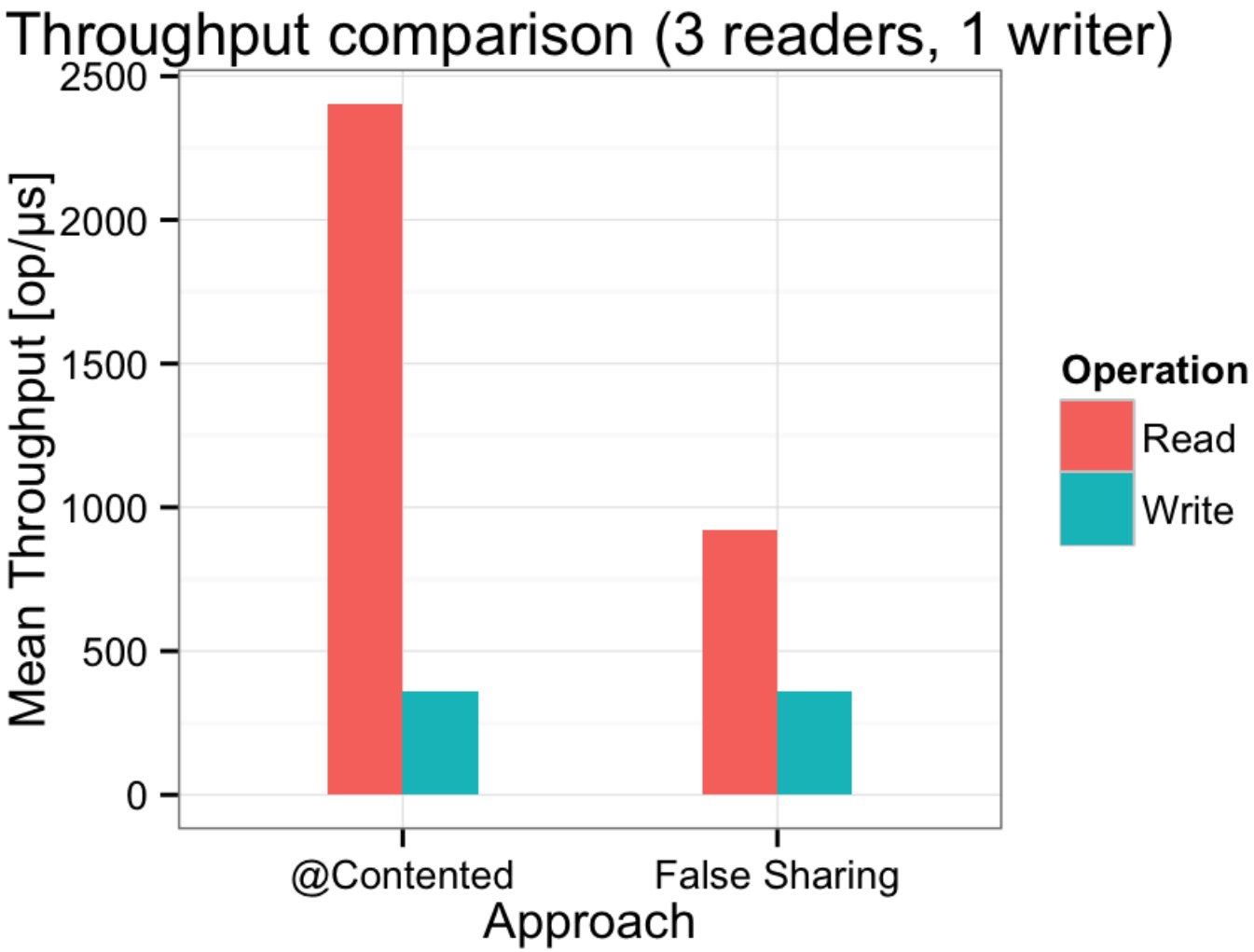
Field padding, e.g. with `@sun.misc.Contended`



Benchmark

```
@Fork(value = 5, jvmArgs = "-XX:-RestrictContended")
@Threads(4)
public class ContendedAccessMicroBenchmark {
  @State(Scope.Benchmark)
  public static class ContendedState {
    @Contended
    public long write;
    public long read;
  }

  @Group("contended")
  @GroupThreads(1)
  @Benchmark
  public void produce(ContendedState s) {
    s.write++;
  }

  @Group("contended")
  @GroupThreads(3)
  @Benchmark
  public long consume(ContendedState s) {
    return s.read;
  }
}
```

Results


Throughput comparison (3 readers, 1 writer)
Mean Throughput [op/µs]
2500
2000
1500
1000
500
0
@Contented
False Sharing
Approach
Operation
Read
Write

Why the difference?

Read CPU performance monitoring data with JMH's `perf` profiler

Metric	`@Contended`	False Sharing
L1-dcache-loads	$231 * 10^9$	$96 * 10^9$
L1-dcache-load-misses (relative to L1 cache hits)	0.01 %	1.7 %

Conclusion

Increased bus traffic due to shared cache lines reduces throughput

Take Aways and Suggestions

- Sometimes, object layout matters in multi-threaded code
- Use libraries like Nitsan Wakart's JCTools which implement countermeasures

Puzzling Branch Prediction

Credits: Example based on a Stackoverflow discussion

Branch Prediction

Keep instruction pipeline full by guessing what will be done next

- Static branch prediction
- Dynamic branch prediction: Based on history
- Loop detector
- Meta-predictor

Contenders: `int[]` sorted / unsorted

- Array size: 2^{16} elements each
- Array values: [0, 255] randomly distributed

Experiment Setup

- Task: Calculate the sum of all elements >= 128

Benchmark: Setup

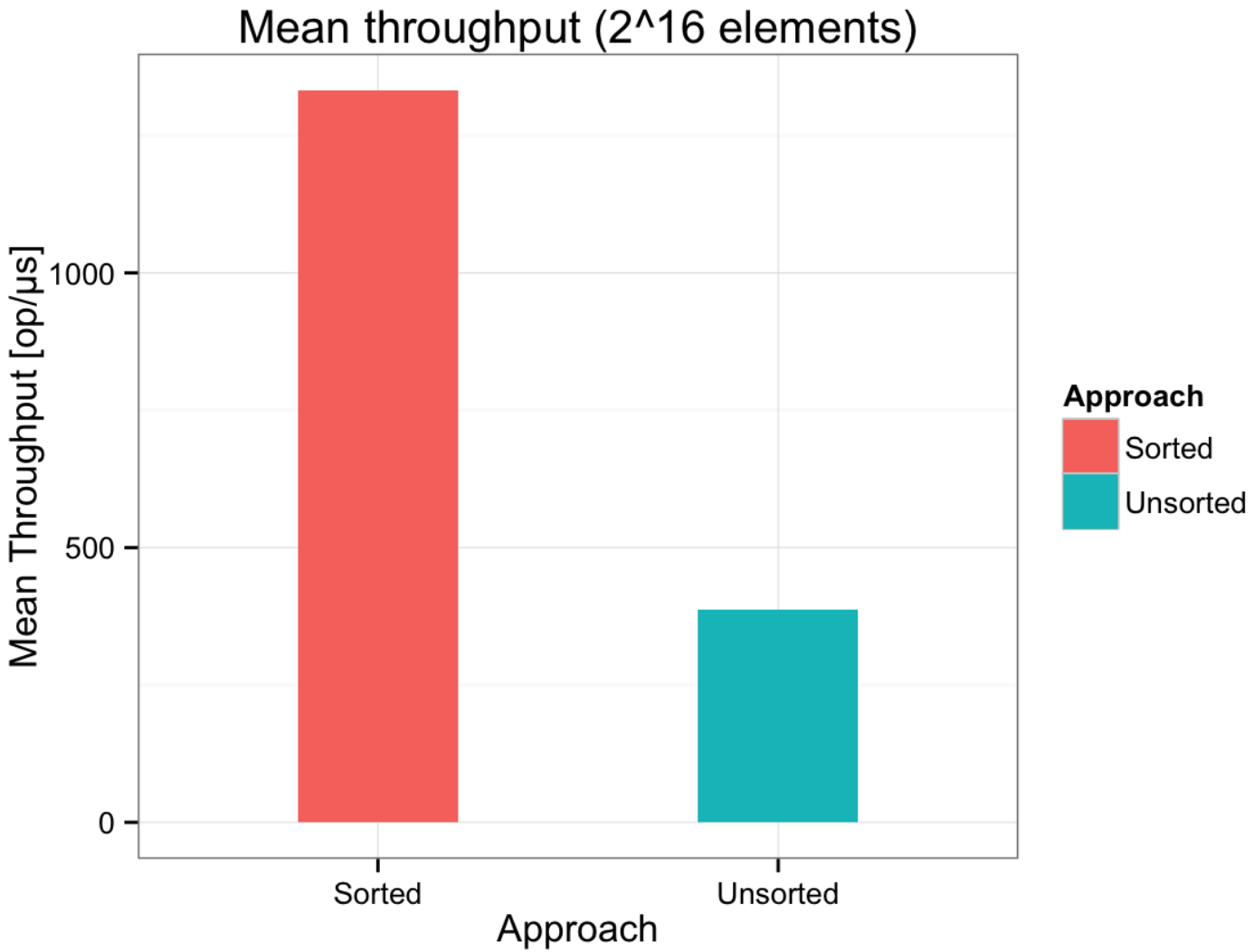
```
@State(Scope.Benchmark)
public class ArraySumBenchmark {
  private static final int PROBLEM_SIZE = 64 * 1024; /* 64 K elements */
  private final int[] unsortedArray = new int[PROBLEM_SIZE];
  private final int[] sortedArray = new int[PROBLEM_SIZE];

  @Setup
  public void setUp() {
    Random random = new Random(System.currentTimeMillis());
    for (int idx = 0; idx < unsortedArray.length; idx++) {
      unsortedArray[idx] = random.nextInt() % 256;
    }
    // create a sorted copy
    System.arraycopy(unsortedArray, 0, sortedArray, 0, unsortedArray.length);
    Arrays.sort(sortedArray);
  }
  // ...
}
```

Benchmark

```
@Benchmark
@OperationsPerInvocation(PROBLEM_SIZE)
public long benchmarkSumUnsorted() {
  long sum = 0;
  for (int idx = 0; idx < unsortedArray.length; idx++) {
    if (unsortedArray[idx] >= 128) {
      sum += unsortedArray[idx];
    }
  }
  return sum;
}
```

Note: the "sorted" benchmark is identical except for the array reference

Results


Mean throughput (2^16 elements)
Mean Throughput [op/µs]
1000
500
0
Sorted
Unsorted
Approach
Approach
Sorted
Unsorted

Why the difference?

Read CPU performance monitoring data with JMH's `perf` profiler

Metric	sorted	unsorted
branch-misses (relative to all branches)	0.01 %	17.3 %

Conclusion

Randomly taken paths render the branch predictor useless and stall the pipeline

Take Aways and Suggestions

- Avoid branches in critical loops or...
- Make them predictable by following a common branching pattern

Summary

Understand Hardware Behavior

Measure

Resources

Slides: http://bit.ly/java-cpu-talk-munich

Code: http://bit.ly/java-cpu-talk-code

Image Credit

- Humpty Dumpty sat on a wall... by Kate Ter Haar (License: cc-by)
- Spices by Adam Baker (License: cc-by)
- Change the World, Obama. | Nobel Prize for Peace version by sara b. (License: cc-by-nc-nd)
- Tree structure by Michael Heiss (License: cc-by-nc-sa)
- Hidden Card Trick Magic Macro 10-19-09 4 by Steven Depolo (License: cc-by)
- Brick purist by clement127 (License: cc-by-nc-nd)
- Gears 2 by fieldsbh (License: cc-by-nc-sa)
- Cassette Tape by Rolf Venema (License: cc-by-nc-nd)

Image Credit

- no title by Shereen M (License: cc-by-nc-nd)
- Newspaper assembly line by JD Lasica (License: cc-by-nc)
- no title by Herman Layos (License: cc-by-nc-sa)
- Week 29/52 - P52'10 by William Frankhouser (License: cc-by-nc-nd)
- Intel Sandy Bridge schematic (PDF)
- Attention to Detail by Ravenshoe Group (License: cc-by)
- Microscope Night by Machine Project (License: cc-by-nc-sa)
- Stopwatch by William Warby (License: cc-by)

Backup

CPU

- Process model: i7-2635QM
- Micro architecture: Sandy Bridge

CPU

```
cat /proc/cpuinfo #processor 0-7 -> HT enabled

processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 42
model name      : Intel(R) Core(TM) i7-2635QM CPU @ 2.00GHz
stepping        : 7
microcode       : 0x1a
cpu MHz         : 800.312
cache size      : 6144 KB
physical id     : 0
siblings        : 8
core id         : 0
cpu cores       : 4
apicid          : 0
initial apicid  : 0
fpu                     : yes
fpu_exception   : yes
cpuid level     : 13
wp                      : yes
[...]
```

CPU

```
cat /proc/cpuinfo #processor 0-7 -> HT enabled

[...]
flags             : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
bogomips          : 4001.72
clflush size      : 64
cache_alignment   : 64
address sizes     : 36 bits physical, 48 bits virtual
```

OS

```
uname:
Linux 3.15.8-1-ARCH #1 SMP PREEMPT x86_64 GNU/Linux
```