PERFORMANCE AND PREDICTABILITY

Richard Warburton

@richardwarburto
insightfullogic.com

maximuscle
vodafone

RENAULT
LOTUS
GIVES YOU WINGS
CLEAR
TOTAL

Why care about low level rubbish?

Branch Prediction

Memory Access

Storage

Conclusions

Technology or Principles

“60 messages per second.”

“8 ElasticSearch servers on AWS, 26 front end proxy serves. Double that in backend app servers.”

60 / (8 + 26 + 2 * 26) = **0.7 messages / server / second.**

http://highscalability.com/blog/2014/1/6/how-hipchat-stores-and-indexes-billions-of-messages-using-el.html

Performance Discussion

Product Solutions

“Just use our library/tool/framework, and everything is web-scale!”

Architecture Advocacy

“Always design your software like this.”

Methodology & Fundamentals

“Here are some principles and knowledge, use your brain“

Latency Numbers

```
L1 cache reference                            0.5 ns
Branch mispredict                             5   ns
L2 cache reference                            7   ns
14x L1 cache
Mutex lock/unlock                            25   ns
Main memory reference                       100   ns
20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy              3,000   ns
Send 1K bytes over 1 Gbps network        10,000   ns    0.01 ms
Read 4K randomly from SSD*              150,000   ns    0.15 ms
Read 1 MB sequentially from memory      250,000   ns    0.25 ms
Round trip within same datacenter       500,000   ns    0.5  ms
Read 1 MB sequentially from SSD*      1,000,000   ns    1    ms
Disk seek                            10,000,000   ns   10    ms
Read 1 MB sequentially from disk     20,000,000   ns   20    ms
Send packet CA->Netherlands->CA     150,000,000   ns  150    ms
```

Stolen (cited) from https://gist.github.com/jboner/2841832

Low Hanging is a cost-benefit analysis

So when does it matter?

Informed Design & Architecture *

* this is not a call for premature optimisation

Case Study: Messaging

- 1 Thread reading network data
- 1 Thread writing network data
- 1 Thread conducting admin tasks

Unifying Theme: Be Predictable

An opportunity for an underlying system:

- Branch Prediction
- Memory Access
- Hard Disks

Do you care?

Many problems **not** Predictability Related

- Networking
- Database or External Service
- Minimising I/O
- Garbage Collection
- Insufficient Parallelism

Use an Optimisation Omen

Why care about low level rubbish?

Branch Prediction

Memory Access

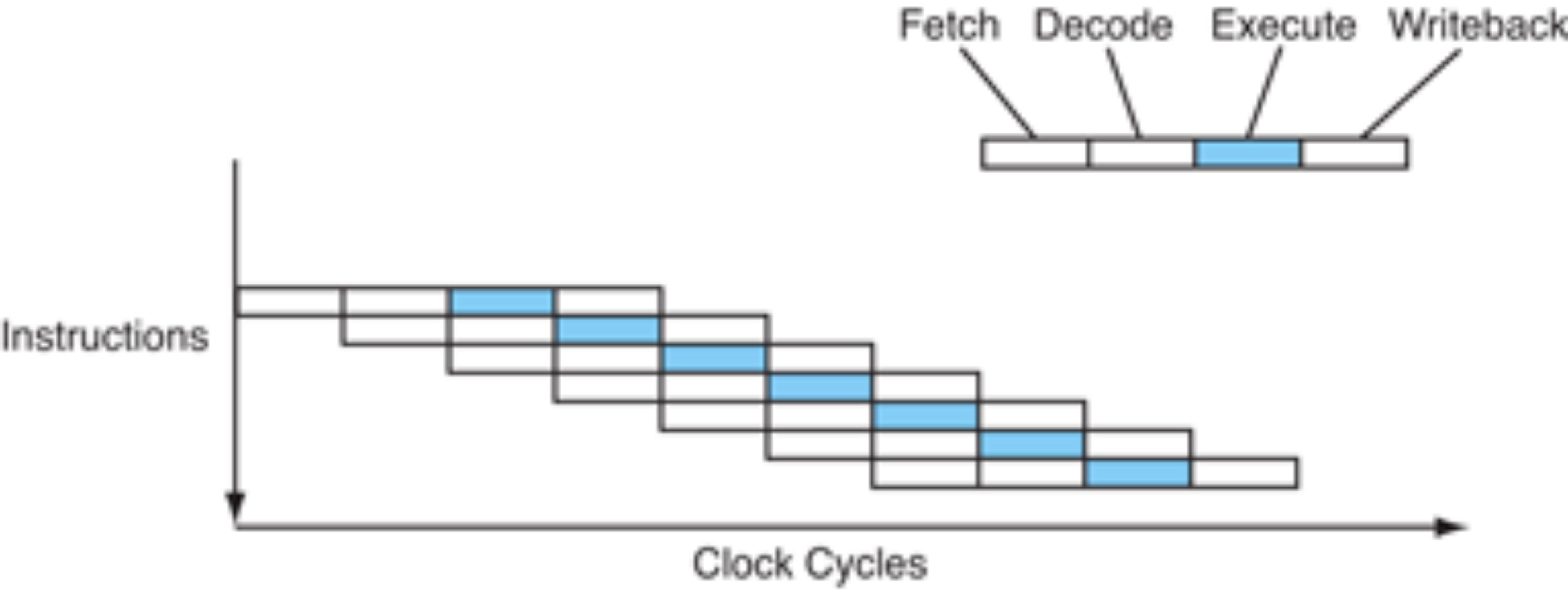
Storage

Conclusions

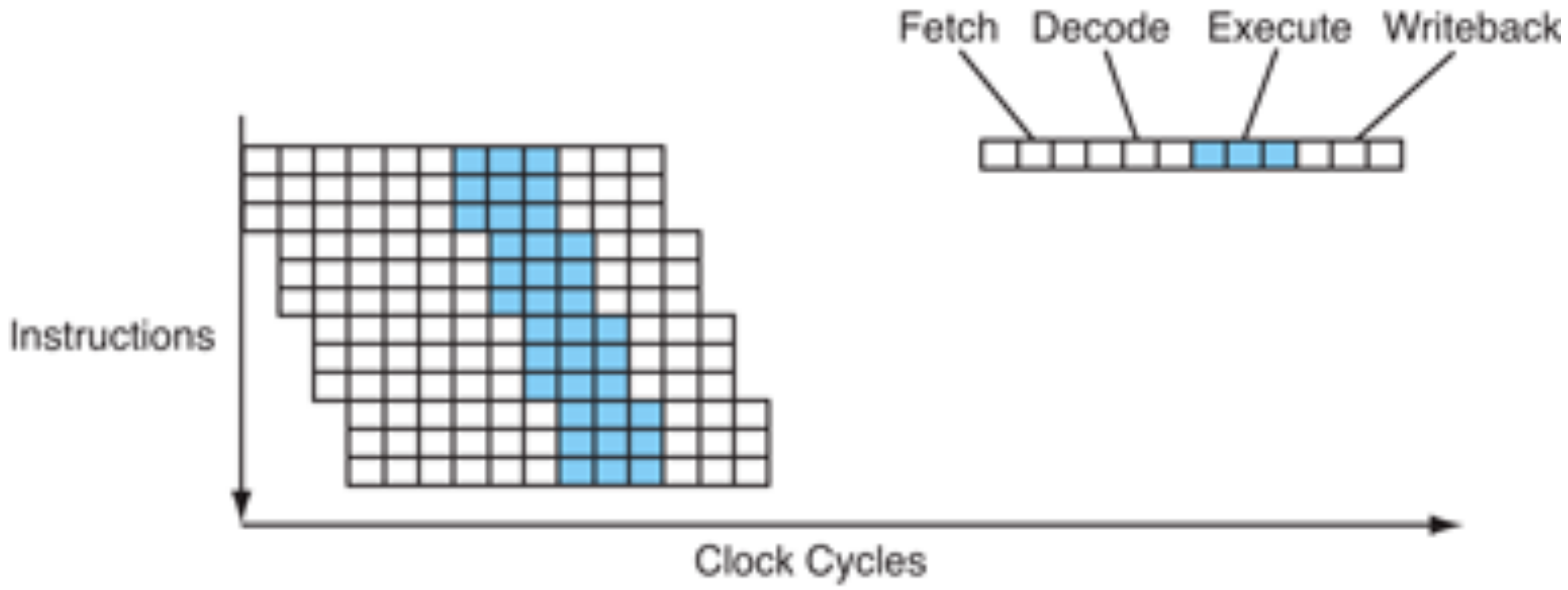
What 4 things do CPUs actually do?

Fetch, Decode, Execute, Writeback

Pipelined


Fetch
Decode
Execute
Writeback
Instructions
Clock Cycles

Super-pipelined & Superscalar



What about branches?

```
public static int simple(int x, int y, int z) {
    int ret;
    if (x > 5) {
        ret = y + z;
    } else {
        ret = y;
    }
    return ret;
}
```

Branches cause stalls, stalls kill performance

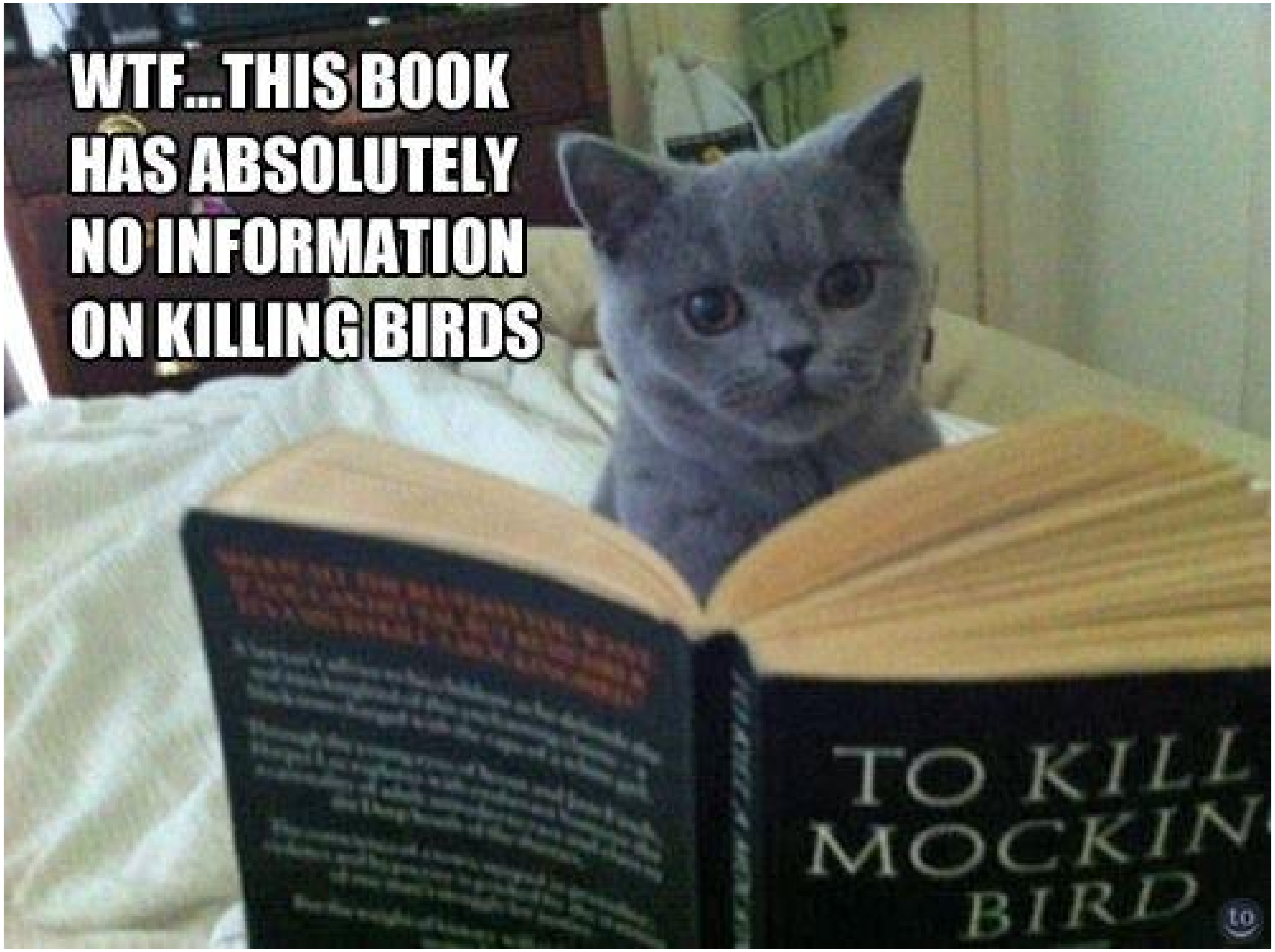
Can we eliminate branches?

Strategy: predict branches and speculatively execute

Static Prediction

A forward branch defaults to not taken

A backward branch defaults to taken


WTF...THIS BOOK HAS ABSOLUTELY NO INFORMATION ON KILLING BIRDS
TO KILL MOCKIN BIRD

Conditional Branches

```
if(x == 0) {
    x = 1;
}
x++;
```

```
mov eax, $x
cmp eax, 0
jne end
mov eax, 1
end:
inc eax
mov $x, eax
```

Static Hints (Pentium 4 or later)

`__emit 0x3E` defaults to taken

`__emit 0x2E` defaults to not taken

don’t use them, flip the branch

Dynamic prediction: record history and predict future

Branch Target Buffer (BTB)

a log of the history of each branch

also stores the program counter address

its finite!

Local

record per conditional branch histories

Global

record shared history of conditional jumps

Loop

specialised predictor when there’s a loop (jumping in a cycle n times)

Function

specialised buffer for predicted nearby function returns

N level Adaptive Predictor

accounts for up patterns of up to N+1 if statements

Optimisation Omen

Use Performance Event Counters (Model Specific Registers)

Can be configured to store branch prediction information

Profilers & Tooling: perf (linux), VTune, AMD Code Analyst, Visual Studio, Oracle Performance Studio

Demo `perf`

Summary

CPUs are Super-pipelined and Superscalar

Branches cause stalls

Simplify your code! Especially branching logic and megamorphic callsites

Why care about low level rubbish?

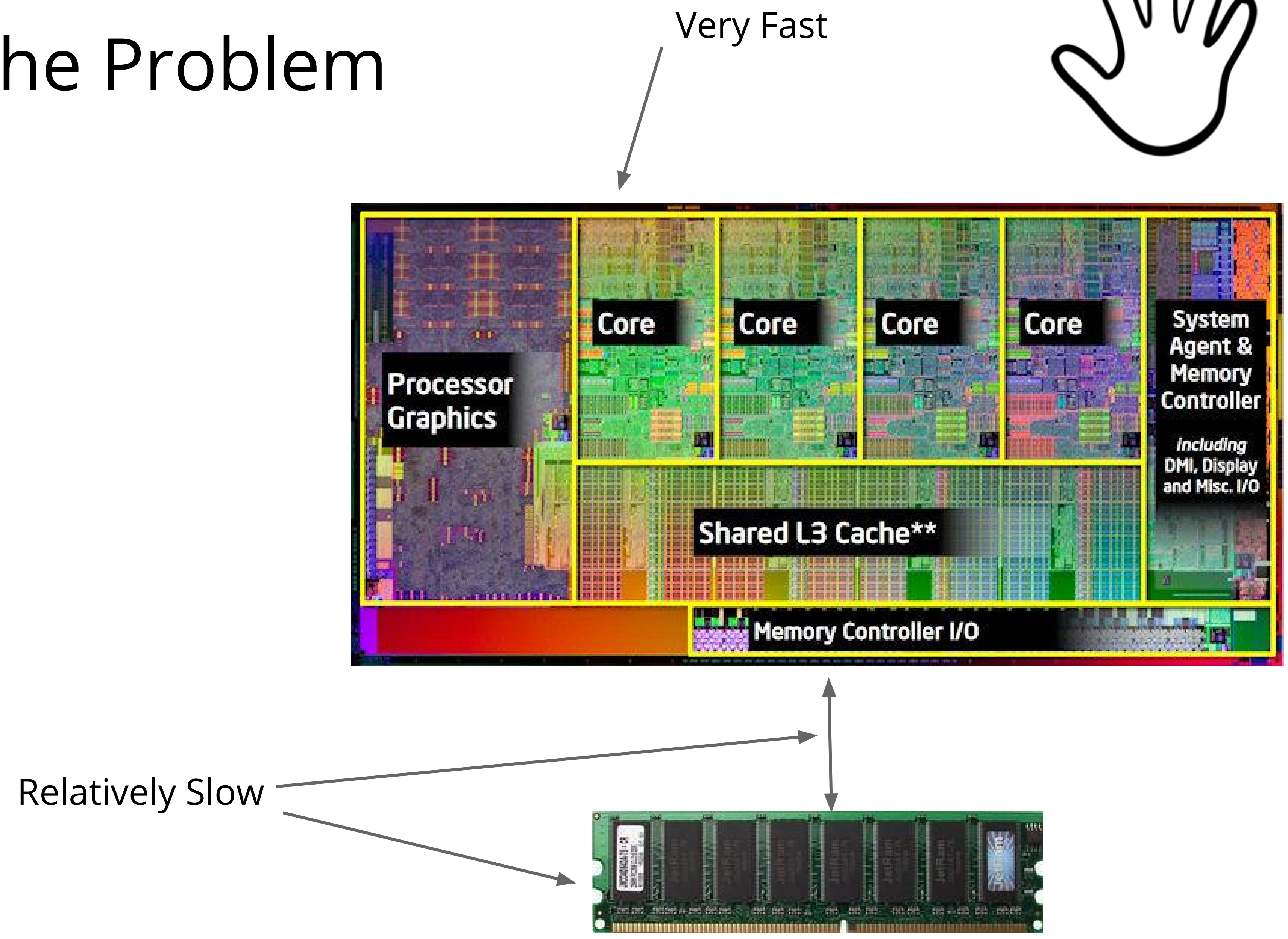
Branch Prediction

Memory Access

Storage

Conclusions

The Problem


Very Fast
Processor Graphics
Core
Core
Core
Core
System Agent & Memory Controller
including DMI, Display and Misc. I/O
Shared L3 Cache**
Memory Controller I/O
Relatively Slow

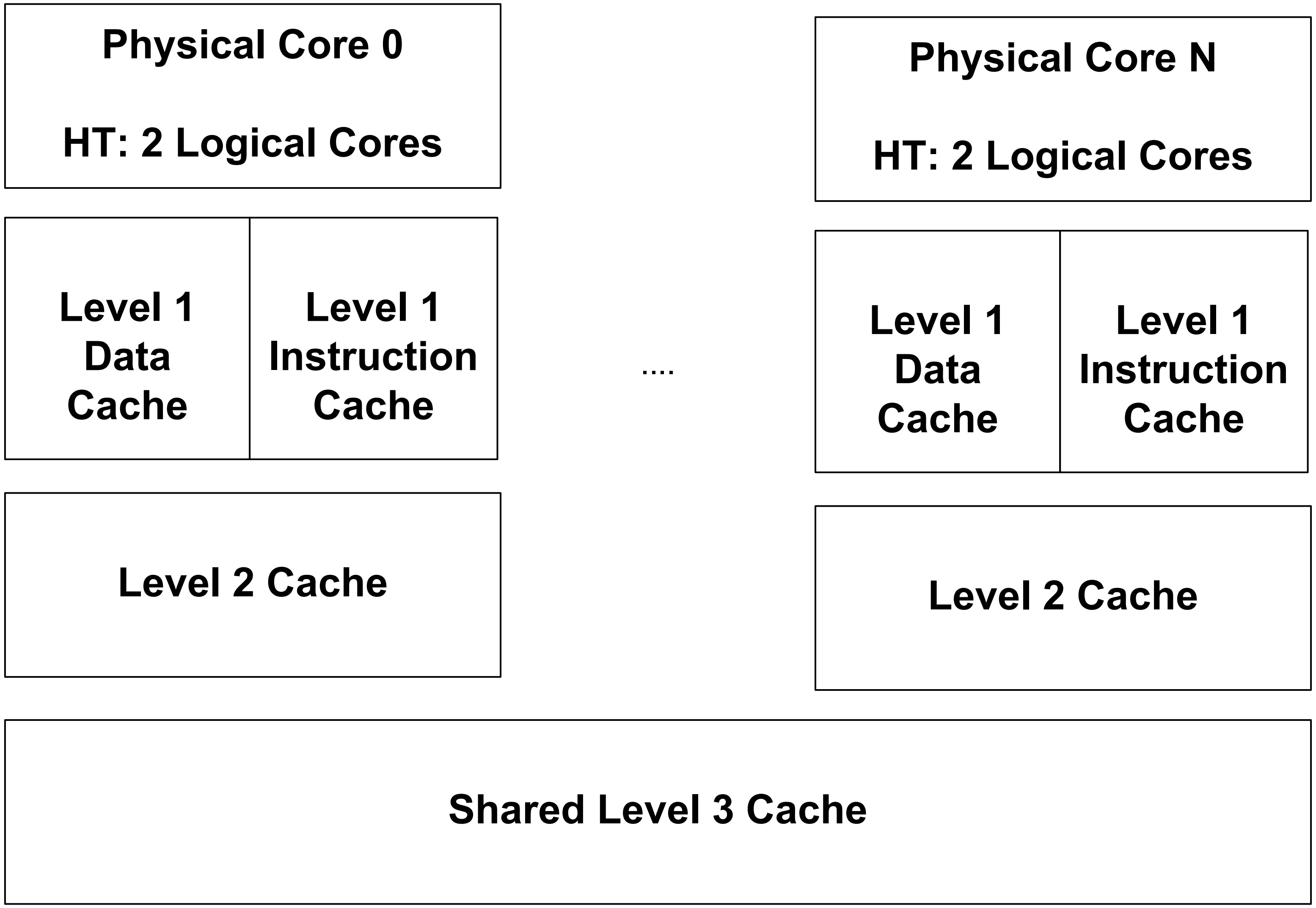
The Solution: CPU Cache

Core Demands Data, looks at its cache

- If present (a "hit") then data returned to register
- If absent (a "miss") then data looked up from memory and stored in the cache

Fast memory is expensive, a small amount is affordable

Multilevel Cache: Intel Sandybridge



How bad is a miss?

Location	**Latency in Clockcycles**
Register	0
L1 Cache	3
L2 Cache	9
L3 Cache	21
Main Memory	**150-400**

Prefetching

Eagerly load data

Adjacent & Streaming Prefetches

Arrange Data so accesses are predictable

Temporal Locality

Repeatedly referring to same data in a short time span

Spatial Locality

Referring to data that is close together in memory

Sequential Locality

Referring to data that is arranged linearly in memory

General Principles

Use smaller data types (*-XX:+UseCompressedOops*)

Avoid 'big holes' in your data

Make accesses as linear as possible

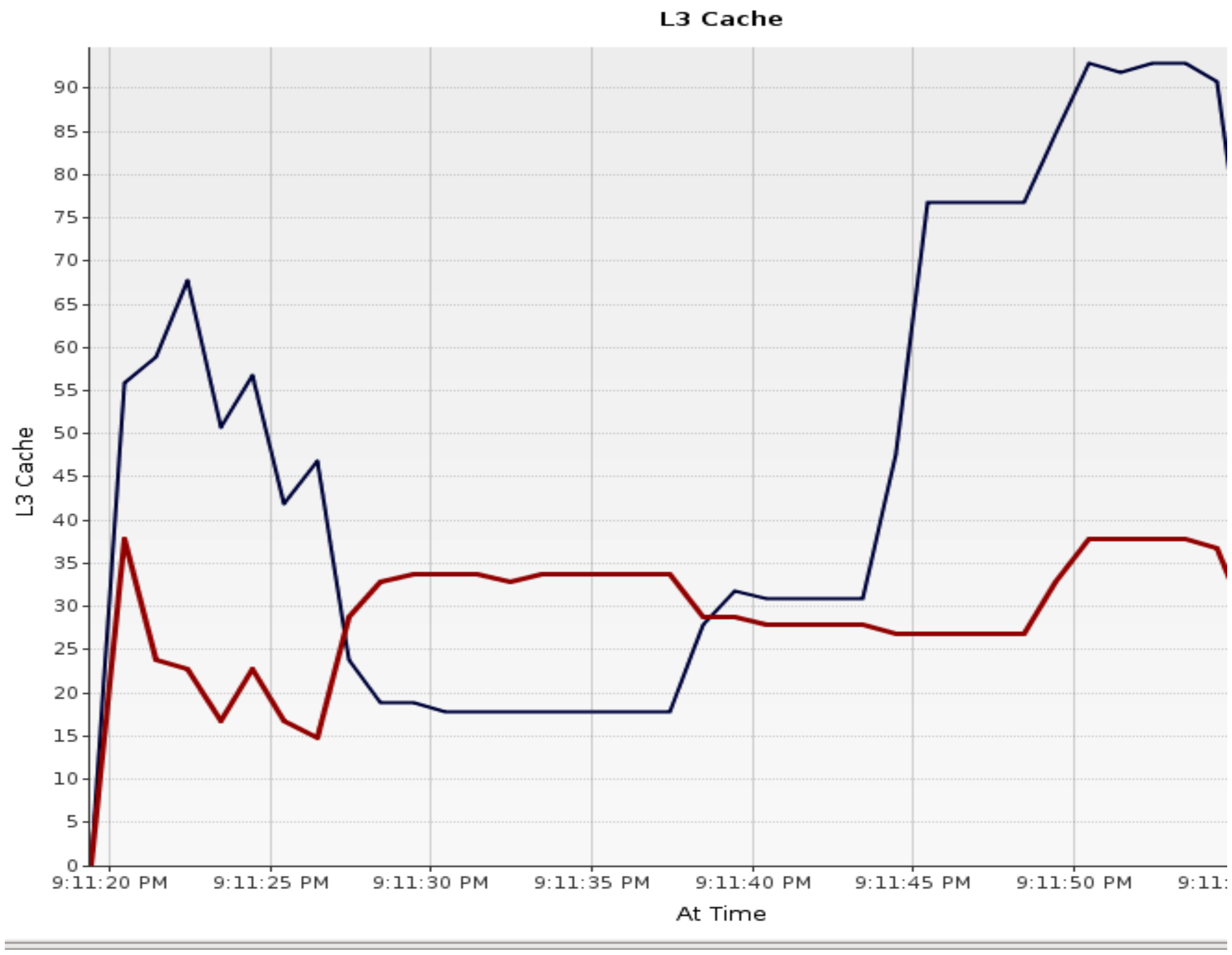
Primitive Arrays

```
// Sequential Access = Predictable

for (int i=0; i<someArray.length; i++)
   someArray[i]++;
```

Primitive Arrays - Skipping Elements

```
// Holes Hurt

for (int i=0; i<someArray.length; i += SKIP)
    someArray[i]++;
```

Primitive Arrays - Skipping Elements



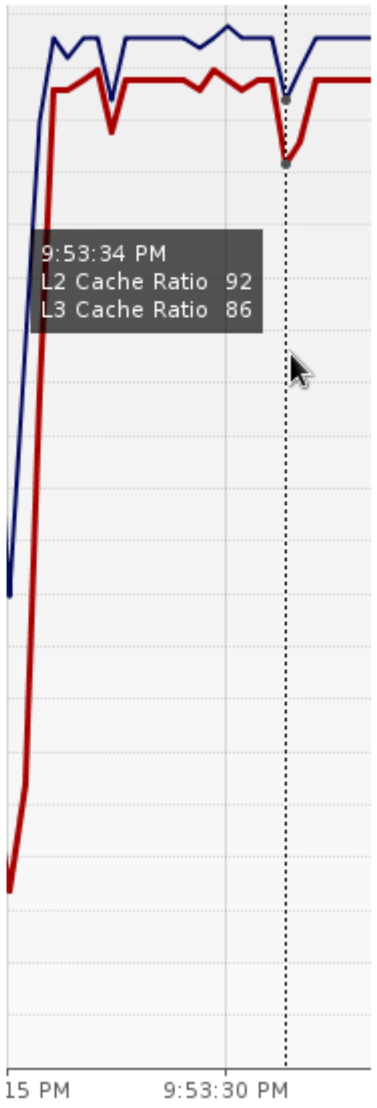
Multidimensional Arrays

Multidimensional Arrays are really Arrays of Arrays in Java. (Unlike C)

Some people realign their accesses:

```
for (int col=0; col<COLS; col++) {
 for (int row=0; row<ROWS; row++) {
  array[ROWS * col + row]++;
 }
}
```

Bad Access Alignment

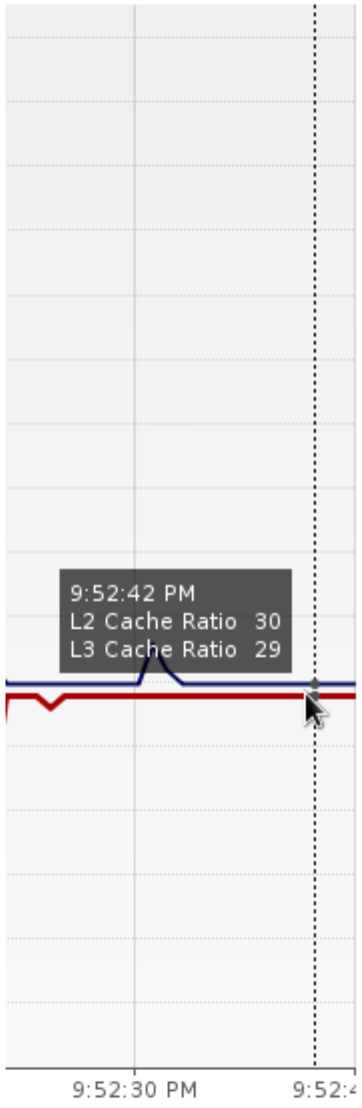


Strides the wrong way, bad locality.

```
array[COLS * row + col]++;
```

Strides the right way, good locality.

```
array[ROWS * col + row]++;
```



Full Random Access

- L1D - 5 clocks
- L2 - 37 clocks
- Memory - 280 clocks

Sequential Access

- L1D - 5 clocks
- L2 - 14 clocks
- Memory - 28 clocks

Data Layout Principles

Primitive Collections (GNU Trove, GS-Coll, FastUtil, HPPC)

Arrays > Linked Lists

Hashtable > Search Tree

Avoid Code bloating (Loop Unrolling)

Custom Data Structures

Judy Arrays

- an associative array/map

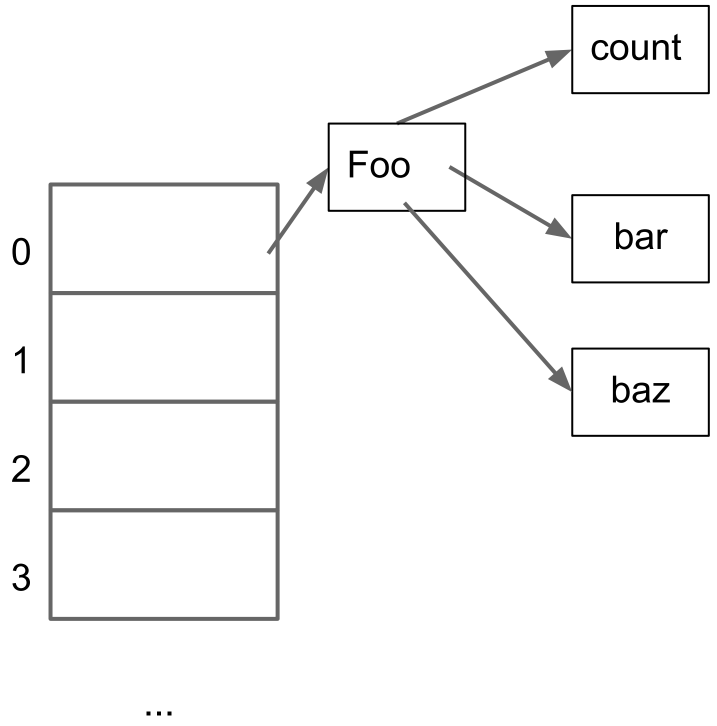
kD-Trees

- generalised Binary Space Partitioning

Z-Order Curve

- multidimensional data in one dimension

Data Locality vs Java Heap Layout



```
class Foo {
   Integer count;
   Bar bar;
   Baz baz;
}

// No alignment guarantees
for (Foo foo : foos) {
   foo.count = 5;
   foo.bar.visit();
}
```

Data Locality vs Java Heap Layout

Serious Java Weakness

Location of objects in memory hard to guarantee.

GC also interferes

- Copying
- Compaction

Optimisation Omen

Again Use Performance Event Counters

Measure for cache hit/miss rates

Correlate with Pipeline Stalls to identify where this is relevant

Object Layout Control

On Heap

http://objectlayout.github.io/ObjectLayout

Off Heap

- Data Structures: Chronicle or JCTools Experimental
- Serialisation: SBE, Cap’n’p, Flatbuffers

Summary

Cache misses cause stalls, which kill performance

Measurable via Performance Event Counters

Common Techniques for optimizing code

Why care about low level rubbish?

Branch Prediction

Memory Access

Storage

Conclusions

Hard Disks

Commonly used persistent storage

Spinning Rust, with a head to read/write

Constant Angular Velocity - rotations per minute stays constant

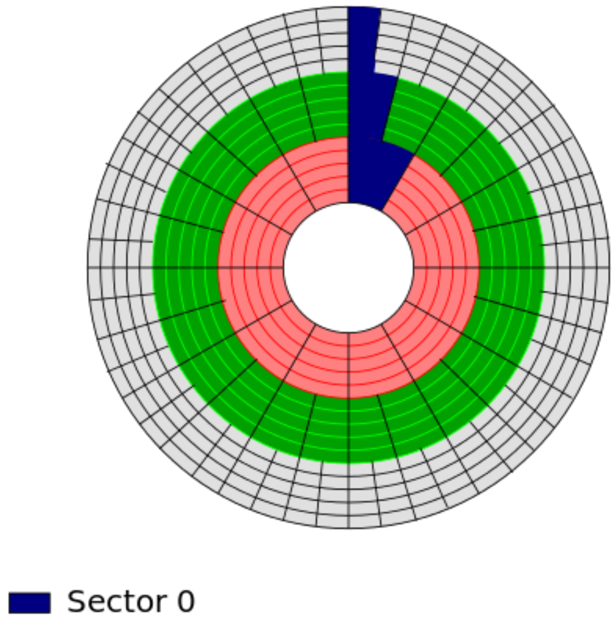
Sectors size differs between device

A simple model

Zone Constant Angular Velocity (ZCAV) /
Zoned Bit Recording (ZBR)

Operation Time =

- Time to process the command
- Time to seek
- Rotational speed latency
- Sequential Transfer TIme



ZBR implies faster transfer at limits than centre (~25%)

Seeking vs Sequential reads

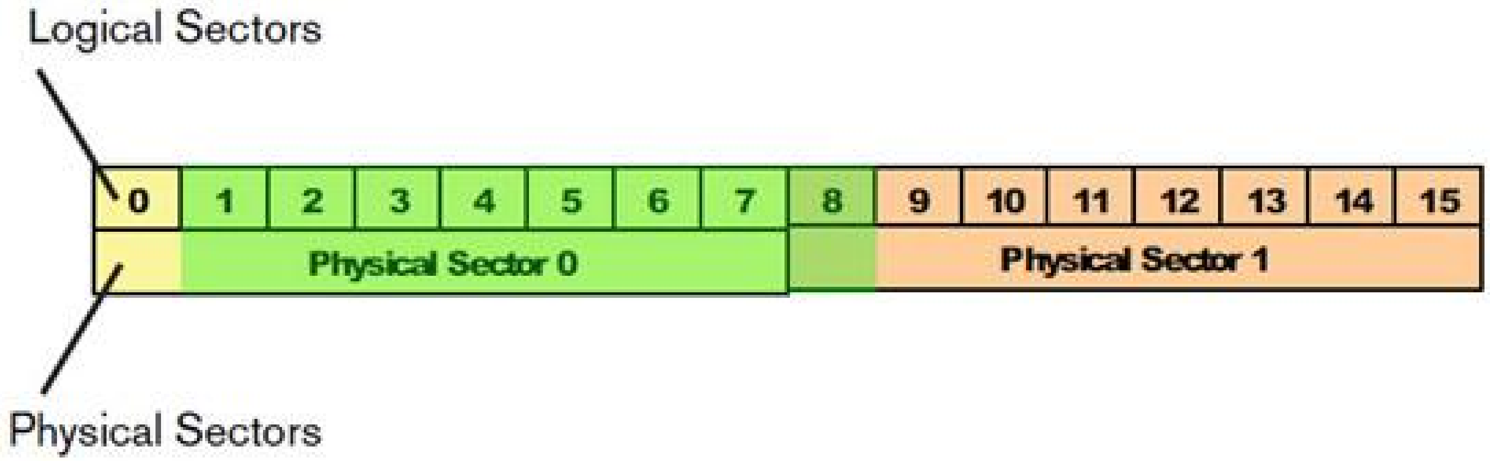
Seek and Rotation times dominate on small values of data

Random writes of 4kb can be 300 times slower than theoretical max data transfer

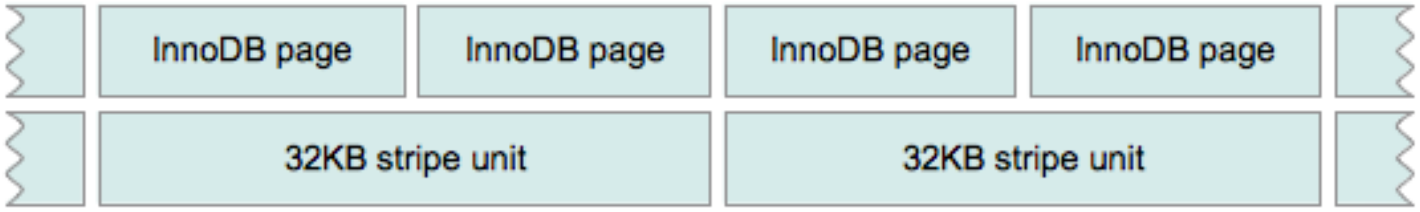
Consider the impact of context switching between applications or threads

Fragmentation causes unnecessary seeks

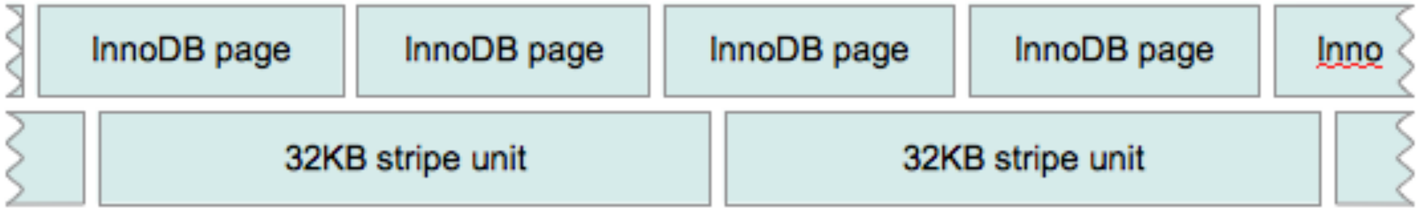
Sector (Mis) Alignment



InnoDB pages aligned to a stripe:



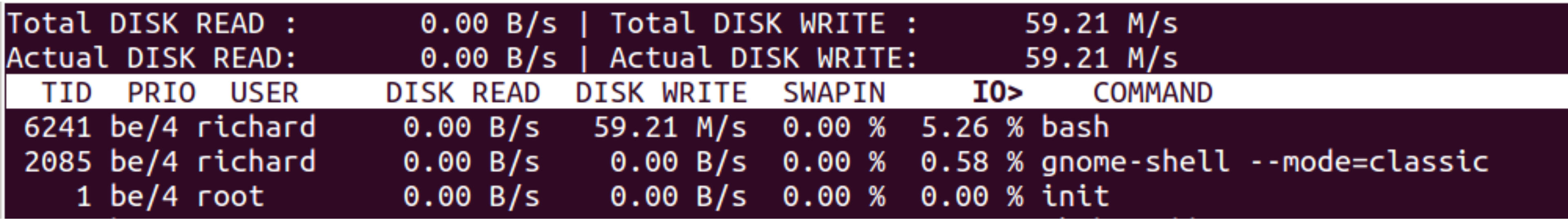
Misaligned InnoDB pages:



Optimisation Omen

1. Application Spending time waiting on I/O

2. I/O Subsystem not transferring much data



```
Total DISK READ :       0.00 B/s | Total DISK WRITE :      59.21 M/s
Actual DISK READ:       0.00 B/s | Actual DISK WRITE:      59.21 M/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 6241 be/4 richard    0.00 B/s   59.21 M/s  0.00 %  5.26 % bash
 2085 be/4 richard    0.00 B/s    0.00 B/s  0.00 %  0.58 % gnome-shell --mode=classic
    1 be/4 root       0.00 B/s    0.00 B/s  0.00 %  0.00 % init
```

Summary

Simple, sequential access patterns win

Fragmentation is your enemy

Alignment can be important

Why care about low level rubbish?

Branch Prediction

Memory Access

Storage

Conclusions

Speedups

- Possible 20 cycle stall for a mispredict (example 5x slowdown)
- 200x for L1 cache hit vs Main Memory
- 300x for sequential vs random on disk
- Theoretical Max

Common Themes

- Principles over Tools
- Data over Unsubstantiated Claims
- Simple over Complex
- Predictable Access over Random Access

More information

Articles

http://www.akkadia.org/drepper/cpumemory.pdf

https://gmplib.org/~tege/x86-timing.pdf

http://psy-lob-saw.blogspot.co.uk/

http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html

http://mechanical-sympathy.blogspot.co.uk

http://www.agner.org/optimize/microarchitecture.pdf

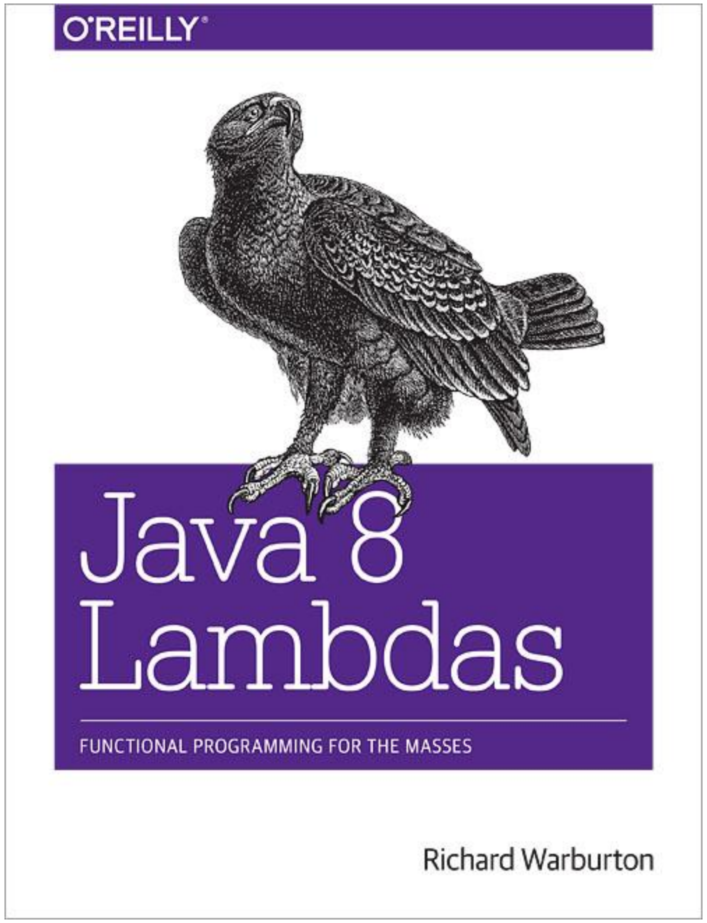
Mailing Lists:

https://groups.google.com/forum/#!forum/mechanical-sympathy

https://groups.google.com/a/jclarity.com/forum/#!forum/friends

http://gee.cs.oswego.edu/dl/concurrency-interest/

http://java8training.com



http://is.gd/javalambdas

Q & A

@richardwarburto
insightfullogic.com
tinyurl.com/java8lambdas