OfficeFloor

Building a 6 million req/sec web server

Slides:
https://officefloor.net/DDDPerth2021.pdf

Daniel Sagenschneider
@sagenschneider

@sagenschneider

Thanks to our Amazing Sponsors!

Platinum Sponsors

valrose

Telstra Purple

Gold Sponsors

Octopus Deploy

VIX

VGW VIRTUAL GAMING WORLDS

IMDEX

Insight.

aws

Microsoft

Silver Sponsors

JOURNEYONE

NDC { Sydney }

FIRST MODE

CyberCX

Progress Telerik

CONFLUENT

MOQdigital.

auth0

family zone

VERSENT

Coffee Cart

Plan it AN NRI COMPANY

elastic

Childcare

bankwest

Quiet Room

auth0

Community Partner

SHE{CODES}

Founder of

OfficeFloor

http://officefloor.net

Inversion of Coupling Control (see DDD Perth 2019)

Focus today is OfficeFloor's web server

Why write a HTTP Server?

OfficeFloor can run on all the above web servers

So why implement our own?

Browsers limit connections

Single Page Application

runs multiple concurrent requests

Browsers only open
~ 2-6 connections

Web Servers

processes 1 request per connection at a time

OfficeFloor not limit request processing

Single Page Application

runs multiple requests

Reduced Latency

Overload handled by Thread Injection

OfficeFloor Web Server

concurrently processes all requests

@sagenschneider

Side Tracked

TechEmpower

Web Framework Benchmarks

https://www.techempower.com/benchmarks/

@sagenschneider

TechEmpower Benchmarks Round 20

Rnk	Framework	JSON	1-query	20-query	Fortunes	Updates	Plaintext
1	lithium	1,599,157	833,811	63,415	659,850	34,887	6,998,356
2	just	1,616,908	668,190	65,399	467,321	35,858	6,992,170
3	drogon	1,070,061	650,753	58,922	666,737	33,377	6,519,621
4	ntex	1,603,310	636,561	34,610	655,964	24,222	7,006,384
5	actix	1,563,586	635,091	34,955	653,529	24,301	7,004,195
6	may-minihttp	1,593,818	635,419	34,617	489,691	21,036	6,991,256
7	wizzardo-http	1,548,467	631,584	31,936	290,654	16,678	7,016,349
8	asp.net core	1,242,834	397,081	22,348	411,986	17,839	7,022,212
9	jooby	1,297,219	548,113	31,840	423,234	16,348	4,031,131
10	beetlex	1,148,005	405,715	23,308	371,228	18,213	4,947,208
11	fiber	1,317,695	395,902	19,808	379,787	11,806	6,413,651
12	atreugo	1,277,526	394,521	19,632	393,762	11,697	6,335,742
13	vert.x	1,128,729	572,605	31,775	340,317	11,812	4,069,297
14	gearbox	1,243,774	368,402	19,390	341,143	11,686	5,725,523
15	quarkus	978,667	536,075	32,182	298,727	14,706	2,557,867
16	greenlightning	873,741	431,296	33,344	326,708	10,805	4,745,071
17	vertx-web	887,266	561,566	31,856	323,065	13,952	2,202,294
18	es4x	874,462	542,881	32,099	248,724	17,326	2,142,787
19	officefloor	1,432,738	356,752	26,445	127,515	11,160	6,417,765
20	kooby: jooby+kotlin	1,208,861	351,054	18,845	307,069	12,085	3,834,752

Top 20

6,417,765 req/sec

…

122 servers entered

Unofficially Top 10

Database Driver change improved test performance

Rnk	Framework	JSON	1-query	20-query	Fortu[illegible]	[illegible]dates	Plaintext
1	lithium	1,603,211	835,680	63,124	[illegible]	35,196	7,000,905
2	just	1,582,676	676,986	66,251	[illegible]	36,479	6,974,053
3	drogon	1,067,286	645,898	59,633	[illegible]	33,744	6,439,076
4	actix	1,603,443	639,313	35,044	[illegible]	24,471	7,002,560
5	ntex	1,614,839	633,475	34,811	[illegible]	24,397	7,019,663
6	may-minihttp	1,603,350	647,900	34,993	494,353	24,513	6,989,196
7	officefloor	1,445,454	577,971	32,043	452,493	17,464	6,552,438
8	wizzardo-http	1,540,760	629,724	31,899	329,520	17,420	7,026,611
9	asp.net core	1,259,769	400,856	21,955	407,082	17,262	7,008,742
10	jooby	1,266,535	545,866	31,969	423,662	16,474	3,946,066

How ?

@sagenschneider

Programming Languages

Languages do NOT dictate performance

OfficeFloor web server written with Java NIO

Just
is written in JavaScript
ranking 2nd

Rnk	Framework	JSON	1-query
1	lithium	1,599,157	833,811
2	just	1,616,908	668,190
3	T drogon	1,070,061	650,753
4	ntex	1,603,310	636,561
5	T actix	1,563,586	635,091
6	may-minihttp	1,593,818	635,419

So what dictates performance?

Speed and Efficiency

I do NOT write *fast* software

Hardware deals in speed

- Better clock cycles
- Increased BUS speeds
- Faster networks

Software deals in efficiency

- Instructions to hardware

@sagenschneider

Must understand hardware

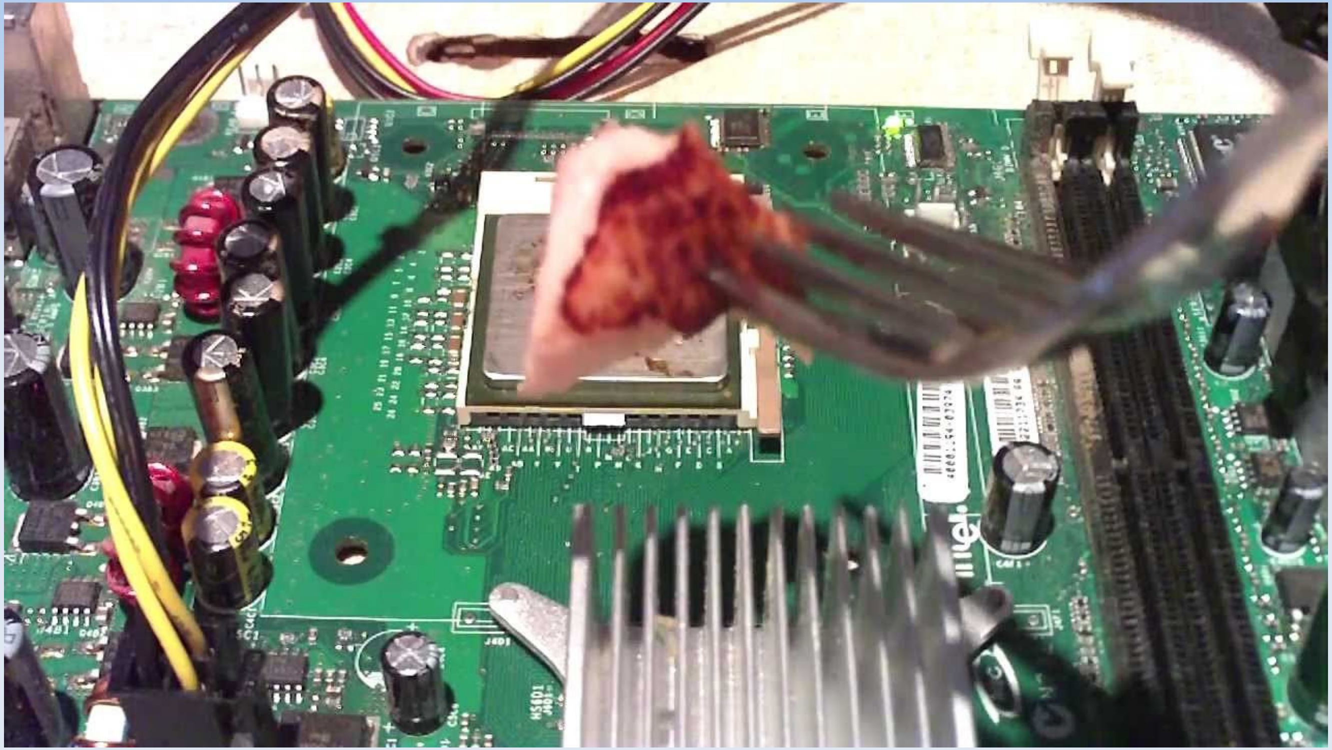
- CPU with registers and caches
- RAM accessed over BUS
- Network cards buffering packets in and out

I’m a Software Developer

Not an Electrical Engineer

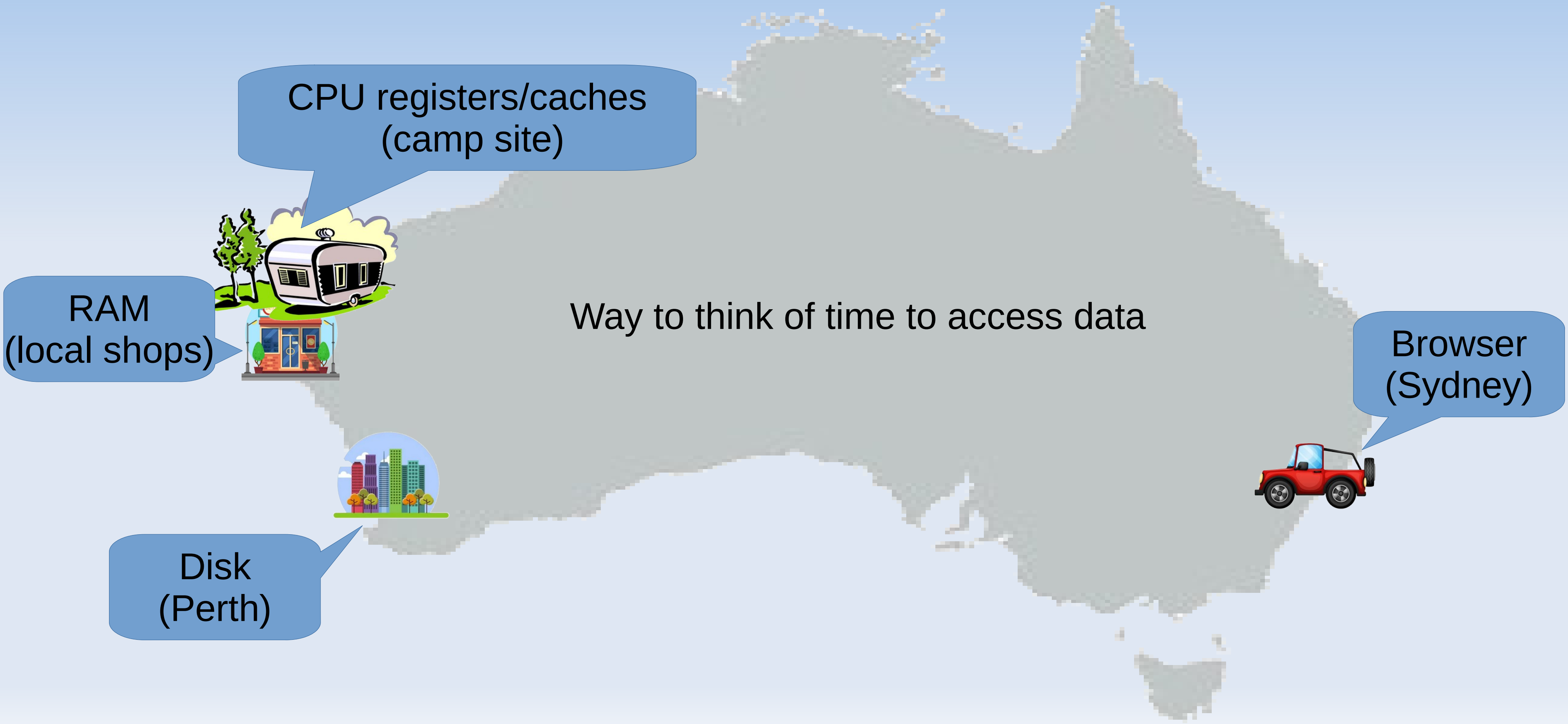
@sagenschneider

Feed data to the CPU



Focus on dynamic web servers (avoid GPUs, DMA, etc)

Process *camps* in CPU



Avoid small chatty networking

Don’t keep driving back and forth over the nullarbor

Don’t take separate cars (lots of small requests)

Packet overheads: 18+20 bytes (Ethernet + TCP)

Wasted bandwidth

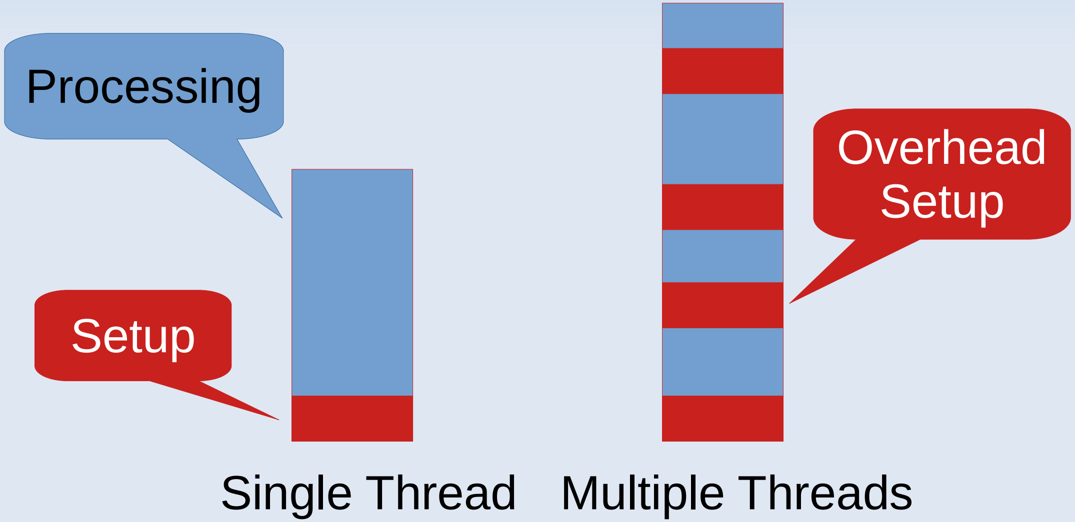
Efficient bandwidth

Built custom Nagle’s algorithm avoiding sleep

More threads is less efficient

Each thread sets up camp

- Load registers
- Warm up caches

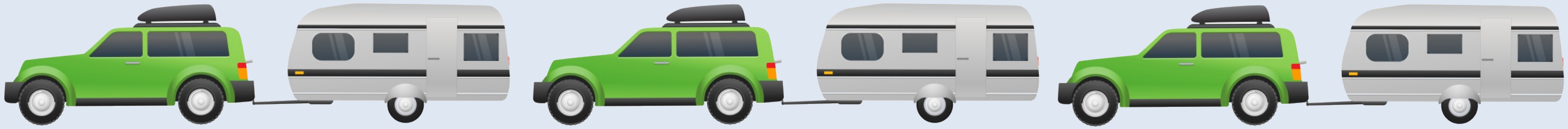


Parallalism under high Concurrency

Child threads consume a camp site (CPU)

Under high concurrency creating extra threads ties up CPUs

Other requests queued for a camp site (CPU)



Thread Coordination

Locking is asking camp host to use camp kitchen

Atomic operations checking continually if free

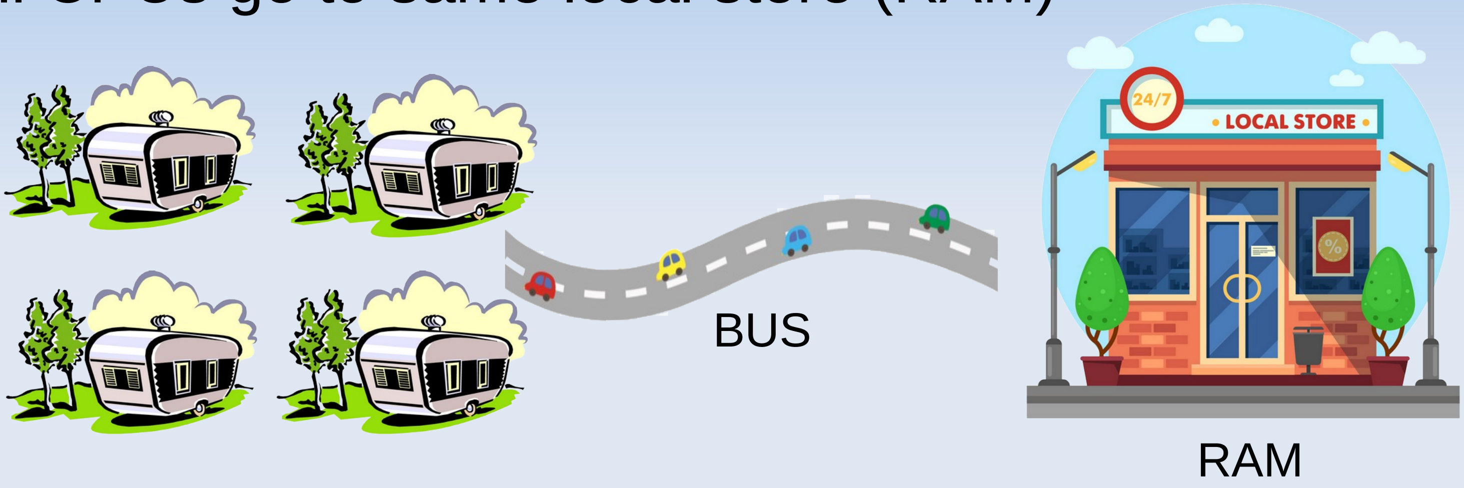
Thread Scheduling

Sleeping threads have to find a camp site (CPU)

Ask to use camp kitchen (lock) and may require moving camp site (CPU)

More CPUs not always better

All CPUs go to same local store (RAM)



You may find cheaper and faster with smaller Cloud instances

OfficeFloor

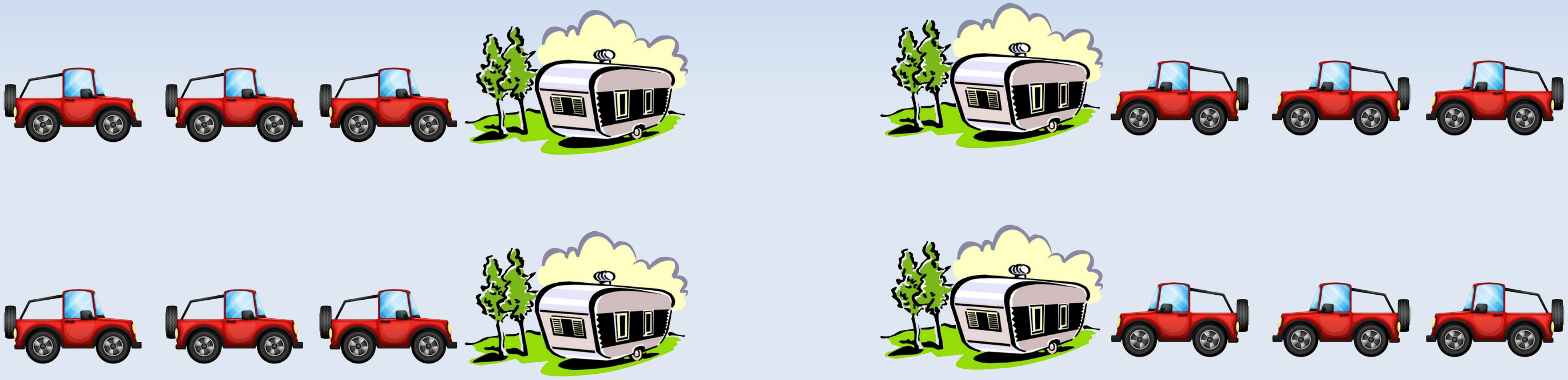
Web Server Threading Model

for 6 million requests per second

OfficeFloor HTTP Server Threading

@sagenschneider

Treat each CPU as its own single threaded server



Ideally want the camp site ready on arrival (avoid setup costs)

GOTCHA: Callback Threads

I/O operations typically have callbacks on separate threads

I have the database data, but
need a camp site (CPU) to process it

Threads start moving camp sites (CPUs)

Thread Affinity

Thread Affinity binds threads to run only on certain CPUs

Only native library used by OfficeFloor web server

https://github.com/OpenHFT/Java-Thread-Affinity

Callback you can camp for a bit,
but I’m NOT moving CPUs

Thanks

Don’t implement Thread Affinity yourself !!!
Needs to be supported by web server

Database Connections

Typically have 1 thread per connection (due to heavy locking)

Minimise threads, so use less connections

SSD's have 0 spindles

Starting rule of thumb:

connections = (core_count * 2) + effective_spindle_count

https://github.com/brettwooldridge/HikariCP/wiki/About-Pool-Sizing

For a 4 CPU database server start with 8 connections only

What about Code

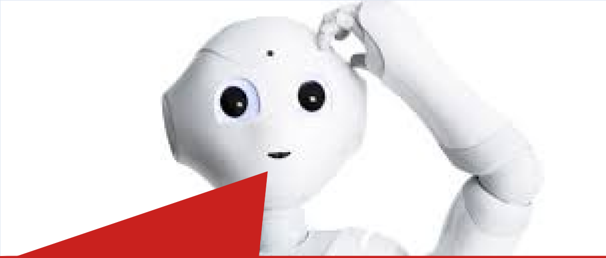
Camping your code in CPUs is about code design

So what about writing more efficient code?

Don’t “out smart” Compilers

- Multiple registers on a CPU (operations processed in parallel)
- Compilers look for code patterns to optimise register use

What is this code doing?



You think you have it bad

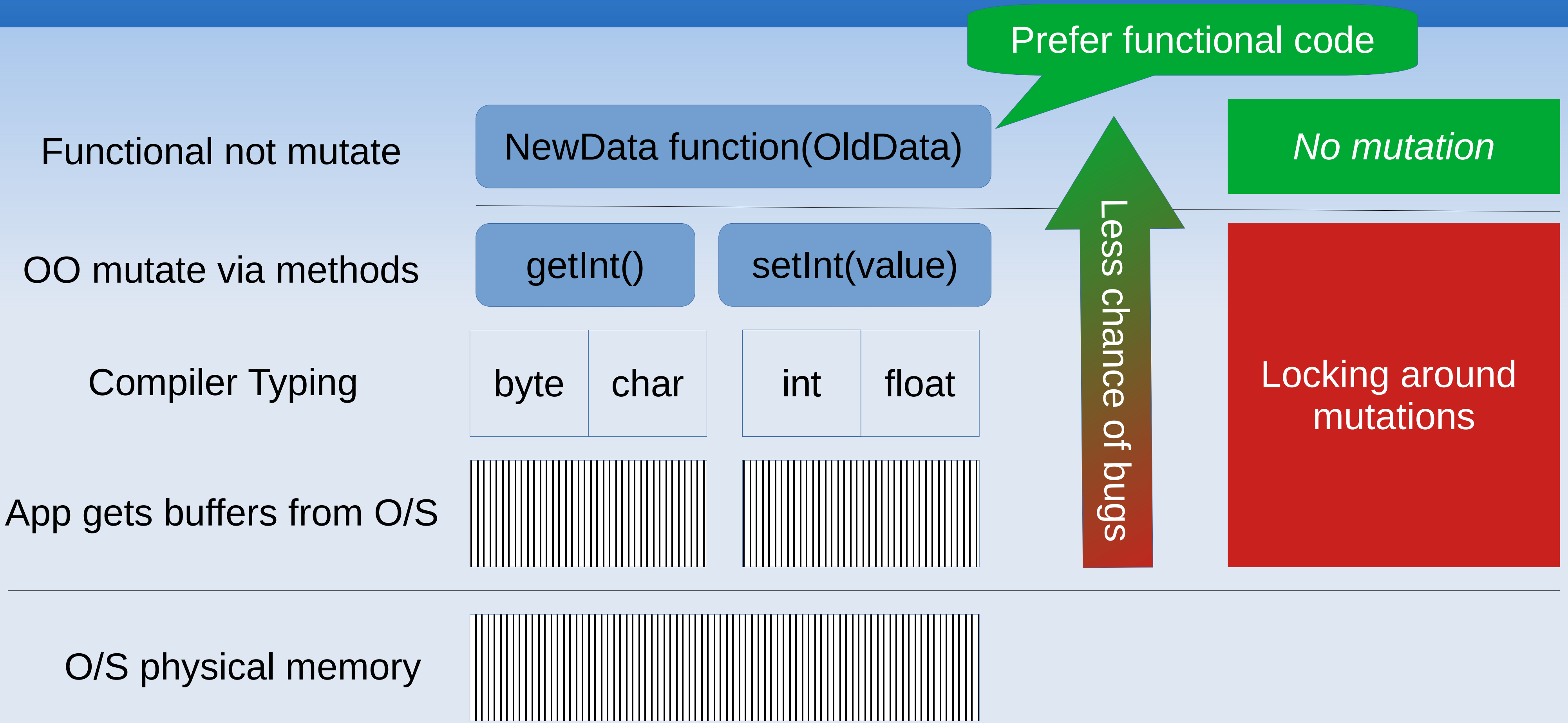
I have to compile and optimise it !!!

Write readable code that “camps” well

What style of readable code?

@sagenschneider

Memory Management



But!

Use memory to our advantage

@sagenschneider

Parsing GET HTTP Method

Functional / Object Oriented

```
httpText = toString(buffer)
httpText.startsWith(“GET”)

(many operations)
```

HTTP in US-ASCII octets

Compiler Typing

```
If (buffer[0] == ‘G’) &&
   (buffer[1] == ‘E’) &&
   (buffer[2] == ‘T’)

(3 operations)
```

64 bit registers (8 bytes)

```
value = buffer.getLong(0)
```

```
value &= ffffff0 0 0 0 0
value == G E T 0 0 0 0 0

(2 operations)
```

PUT:

```
value == P U T 0 0 0 0 0

(1 operation)
```

@sagenschneider

HTTP delimited parsing

US-ASCII HTTP characters consume 7 bits of 8

zero the character (byte)

```
eightBytes = buffer.getLong(0)
eightBytes ^= \r \r \r \r \r \r \r \r
```

check for zero byte

```
eightBytes += 7f 7f 7f 7f 7f 7f 7f 7f
eightBytes &= 80 80 80 80 80 80 80 80
eightBytes == 80 80 80 80 80 80 80 80
```

(4 operations to see if can skip 8 characters)

Bit Twiddling

Only used for short cut parsing repetitive HTTP content

Rest of web server is functional compiler optimised code

Typically, little value for bit twiddling application logic

Write functionaly styled application code

(compiler will generally optimise for you)

Summary

- Hardware is about fast, software is about efficiency
- Understand hardware (at least at logical level)
- Code design is more important than coding language
- Minimise number of threads (ideally 1 per CPU)
  - Also minimise database connections
- Consider atomic operations over locking
- More often scaling instances is better than more CPUs
- Consider functional programming practices

Questions

OfficeFloor

Framework	JSON	1-query	20-query	Fortunes	Updates	Plaintext
officefloor	1,421,681	577,292	32,271	454,446	17,530	6,465,590

Remember for performance less is more efficient

Slides:
http://officefloor.net/DDDPerth2021.pdf

Daniel Sagenschneider
@sagenschneider