

Java at Scale

from the smallest devices to the largest and beyond

QCON 2018

@spoole167



About me

Steve Poole

IBM Lead Engineer / Developer advocate

Making Java Real Since Version 0.9

Open Source Advocate

DevOps Practitioner (whatever that means!)

Driving Change



How do you think about Java?

Like this?

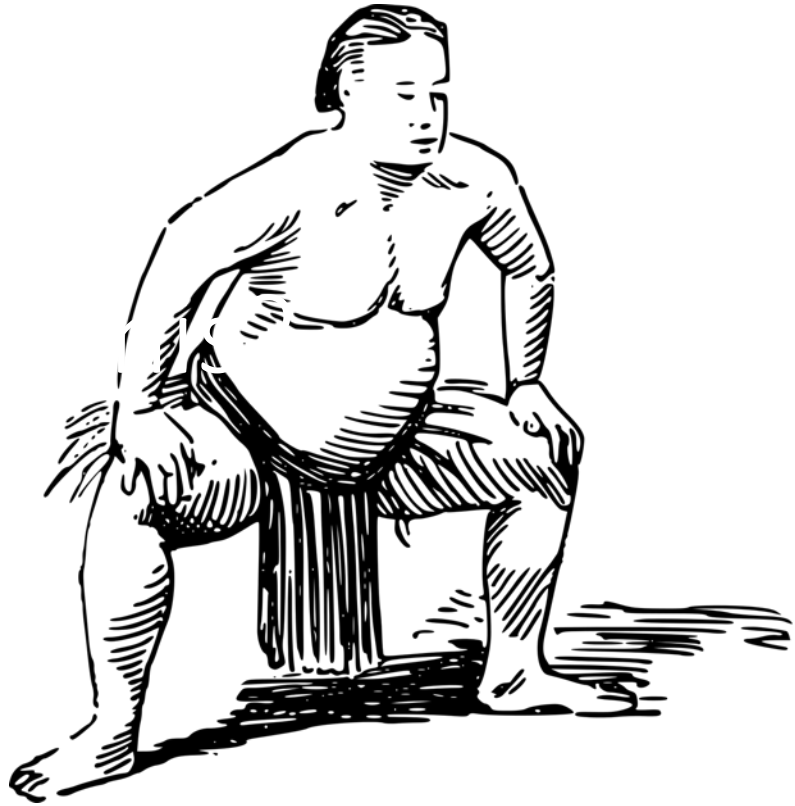


Java ME

Java SE

Java EE

Like this?



Micro Profile



EE vs Spring

Whatever you
think about Java

You're probably only
thinking about Standard
edition
or Enterprise edition

SE or EE..



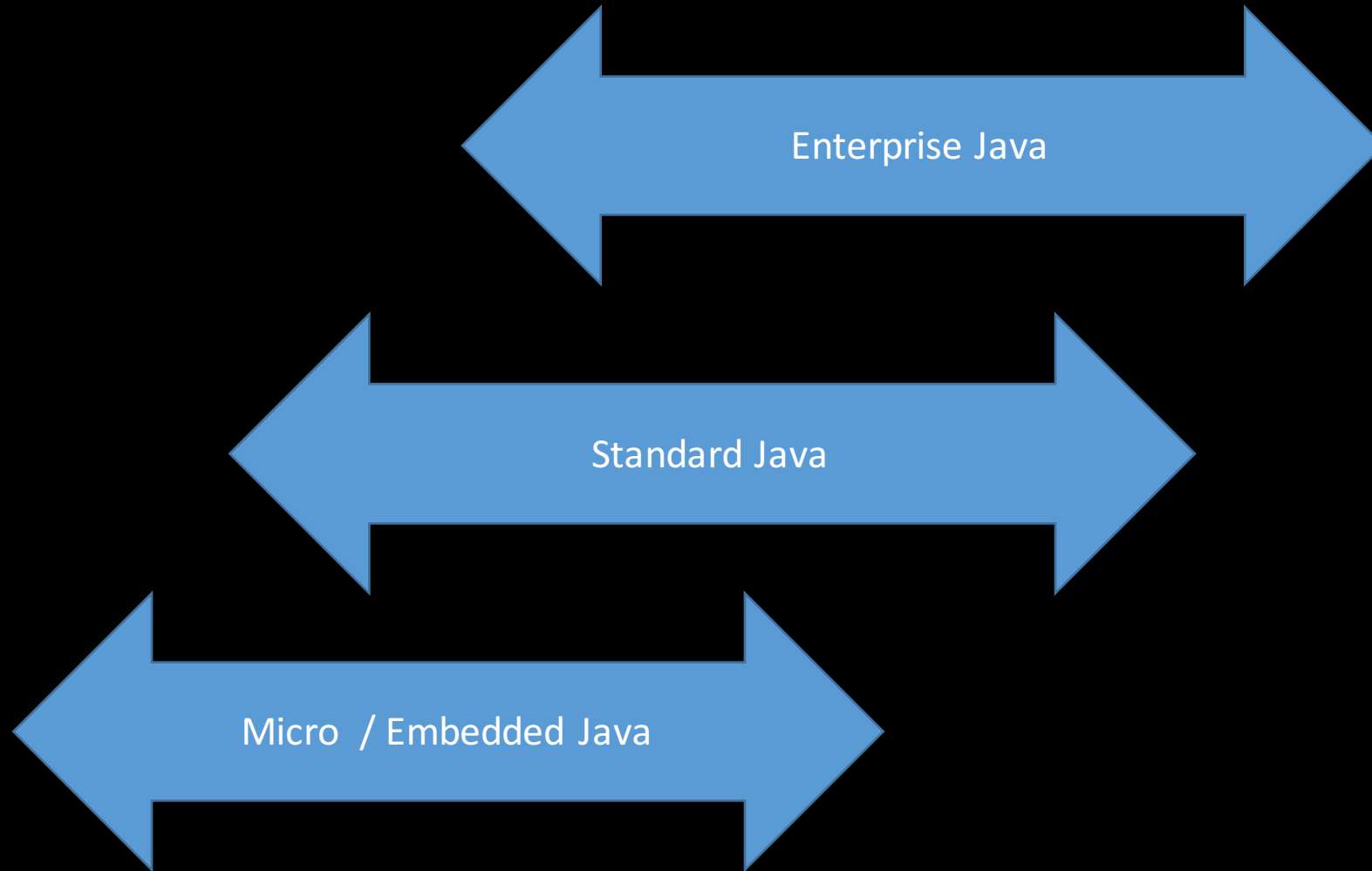
As Java developers we take Java for granted



And ignore the JVM
in the room

In this story – it's the JVM that's the star

Your world might look like this



@spoole167



Class Files

Java Runtime

Threading
Locking
Diagnostics
IO

Memory
Management

JIT

Virtualization Layer (s) and Operating Systems

Hardware

Mine's more
Like this



Mine's more
Like this

Class Files

Java Runtime

Threading
Locking
Diagnostics
IO

Memory
Management

JIT

Virtualization Layer (s) and Operating Systems

Hardware

JVM developers worry about

WORA

So you don't have to



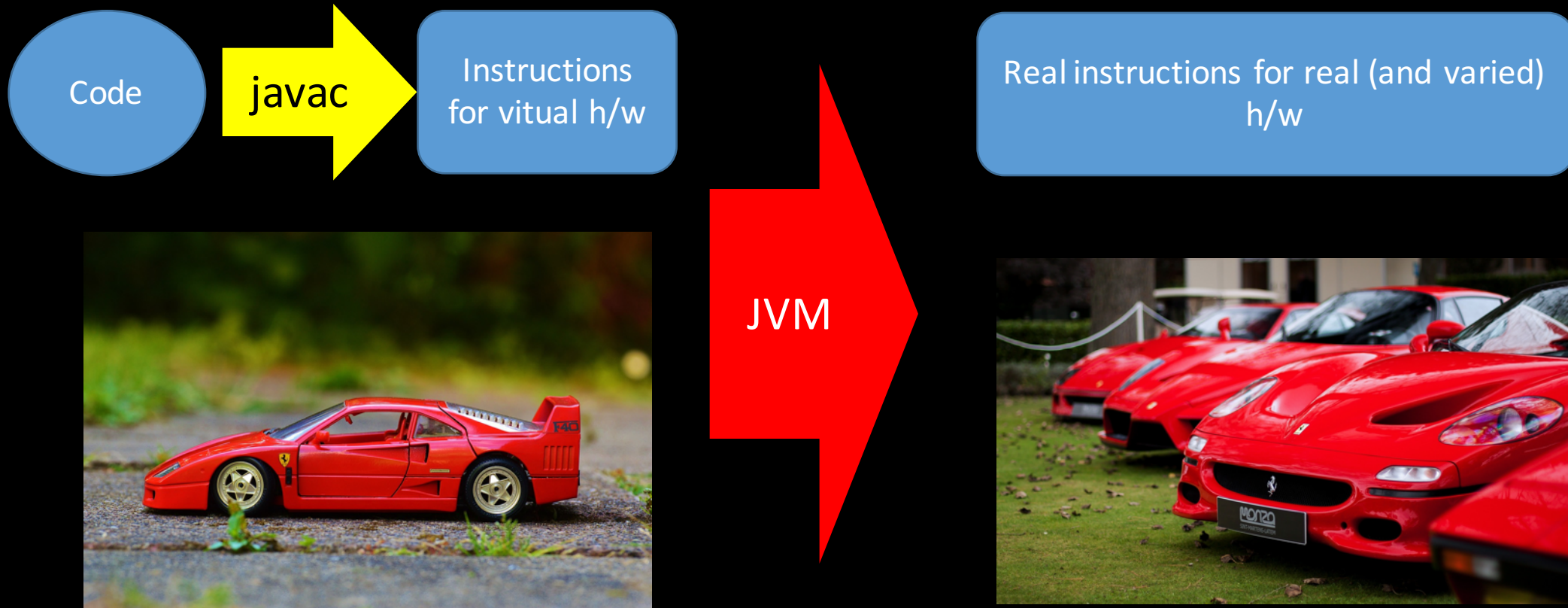
JVMs turn your model of an application
into machine code..

So you don't have to



Wait - what?

JVMs do the heavy lifting in transforming your model of an application into the real thing.



Your model

Multiple real applications

In this talk we're going to explore why this design is so successful and how far it can be bent, squashed and beaten into new shapes.

We'll answer questions like

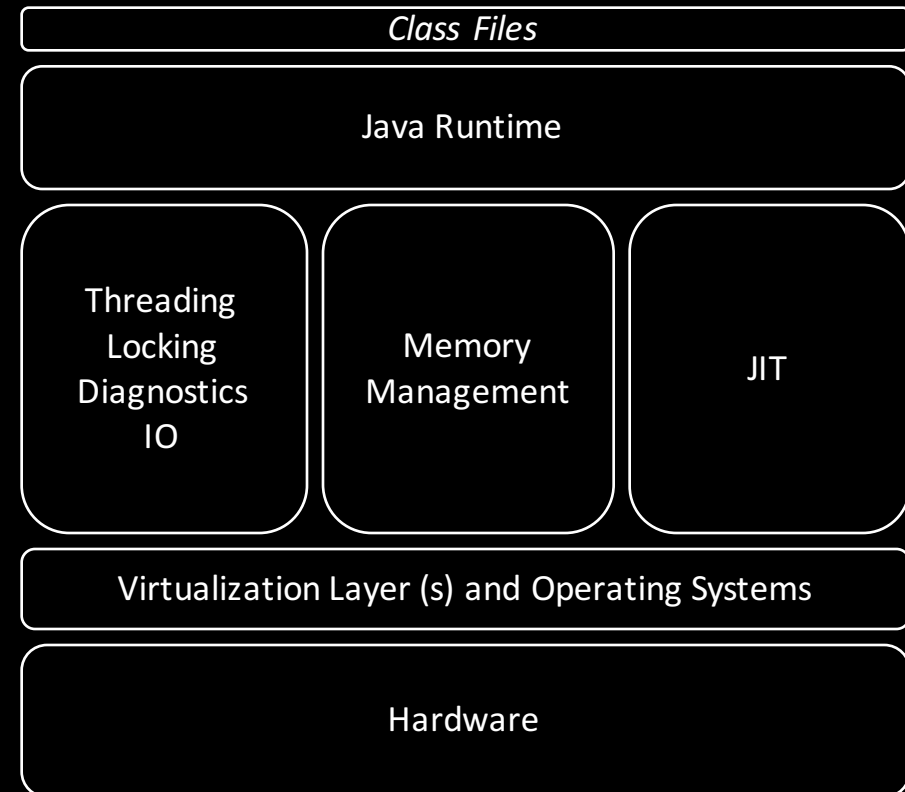
What does 'scale' mean?

How far can you take your Java skills?

What's the future for Java?

What are the important challenges for Java

And are we doing enough to rise to them?

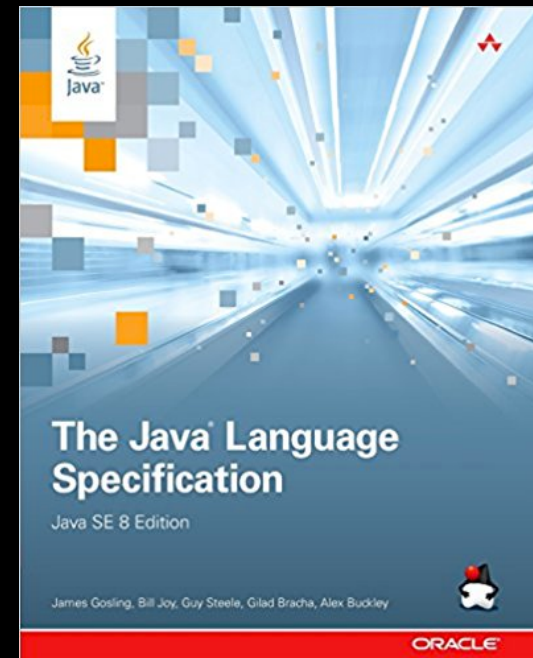
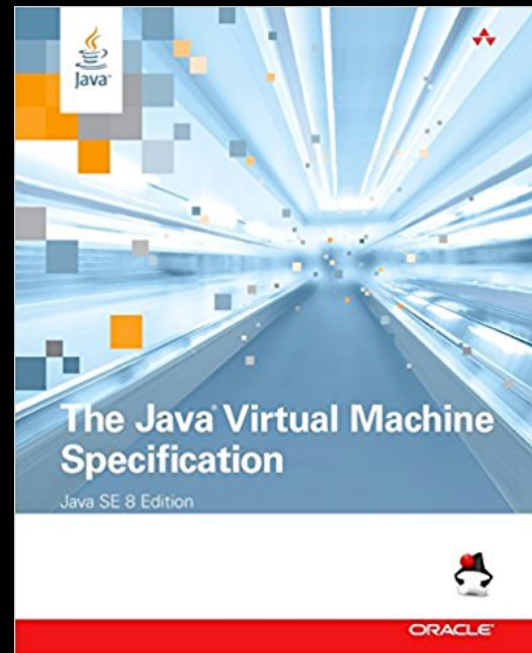


So what is a Java Virtual Machine?

A very quick primer on a Java SE JVM

“The **Java platform** was initially developed to address the problems of building software for networked consumer devices. It was designed **to support multiple host architectures and to allow secure delivery of software components**. To meet these requirements, compiled code had to survive transport across networks, **operate on any client**, and assure the client that it was **safe to run**.”

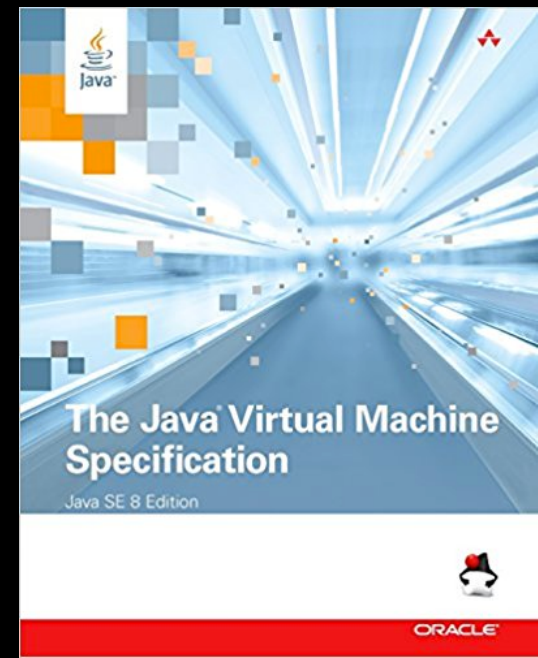
Java® Virtual
Machine
Specification



The Java®
Language
Specification

Important parts

- **The Class File format**
- Data types: ranges, initial conditions, standards to follow
 - (like IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std. 754-1985, New York))
- **How arrays are stored**
- How ClassLoaders work
- **The JVM Instruction Set**
- Bytecode Verification
- How the JVM get started..
 - 5.2 The Java Virtual Machine starts up by creating an initial class or interface using the bootstrap class loader (§5.3.1). The Java Virtual Machine then links the initial class or interface, initializes it, and **invokes the public static method void main(String[]).**

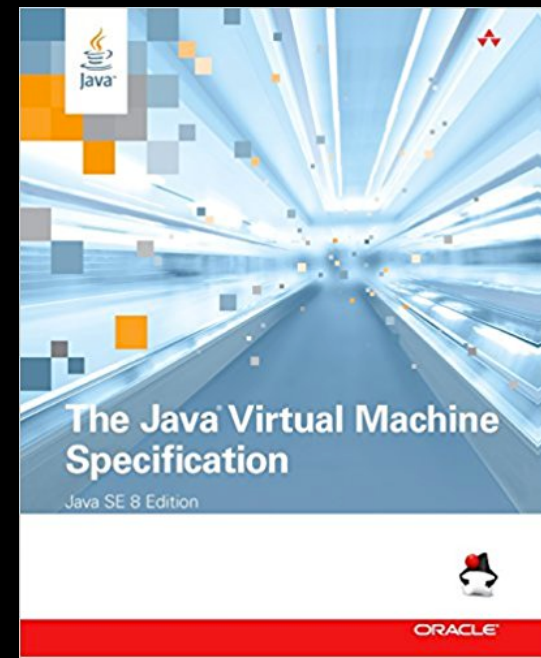


Java® Virtual Machine Specification

Pop quiz:

Which of these are **not** in the JVM spec?

- How the Heap works
 - Size, allocation, deallocation, garbage collection
- How objects are represented in memory
- How the JIT works
- What the semantics of the memory model are?
- What references (aka pointers) to objects look like?
- How finalisation works?
- What the threading model is?
- How synchronization behaves?
- How reflection works?



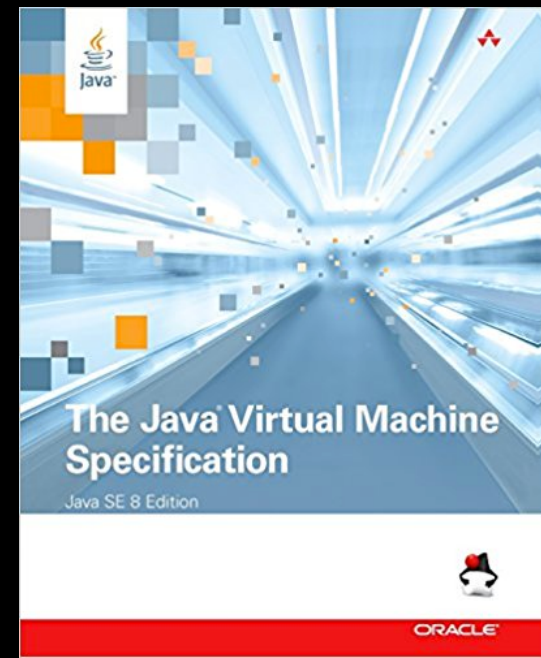
Java[®] Virtual
Machine
Specification

Pop quiz:

Which of these are **not** in the JVM spec?

- How the Heap works
 - Size, allocation, deallocation, garbage collection
- How objects are represented in memory
- How the JIT works
- What the semantics of the memory model are?
- What reference types (pointers) to objects look like?
- How finalizers work?
- What the locking model is?
- How synchronization behaves?
- How reflection works?

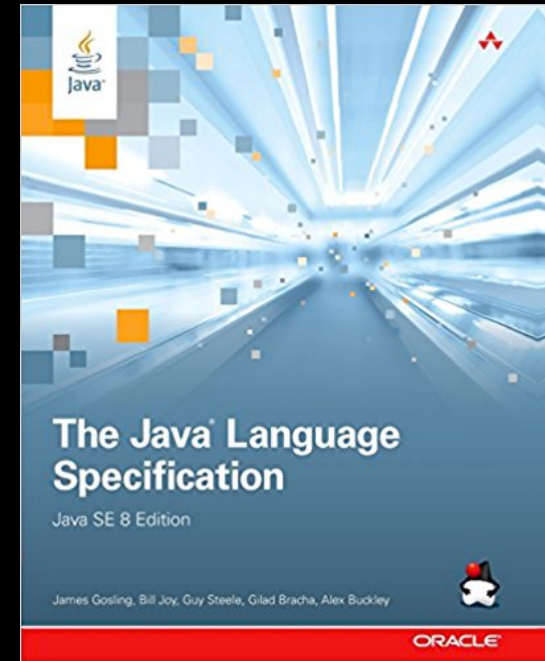
All of them



Java[®] Virtual
Machine
Specification

These items are in the JLS

- What the semantics of the memory model are
- What the threading model is
- How synchronization behaves
- How finalisation should work (ish)



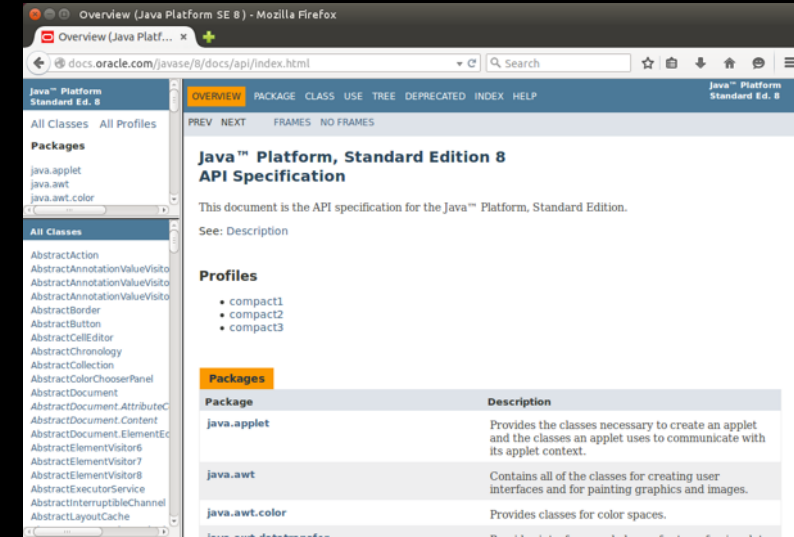
The Java[®]
Language
Specification

Other items are in the API reference

- How reflection works
- How the Security Manager works

“The Java Virtual Machine must provide sufficient support for the implementation of the class libraries of the Java SE Platform. Some of the classes in these libraries cannot be implemented without the cooperation of the Java Virtual Machine.”

“See the specifications of the Java SE Platform class libraries for details.”



These items are not covered by specifications or API docs

- How the Heap works
 - Size, allocation, deallocation, garbage collection
- How objects are represented in memory
- How the JIT works
- What references (aka pointers) to objects look like?

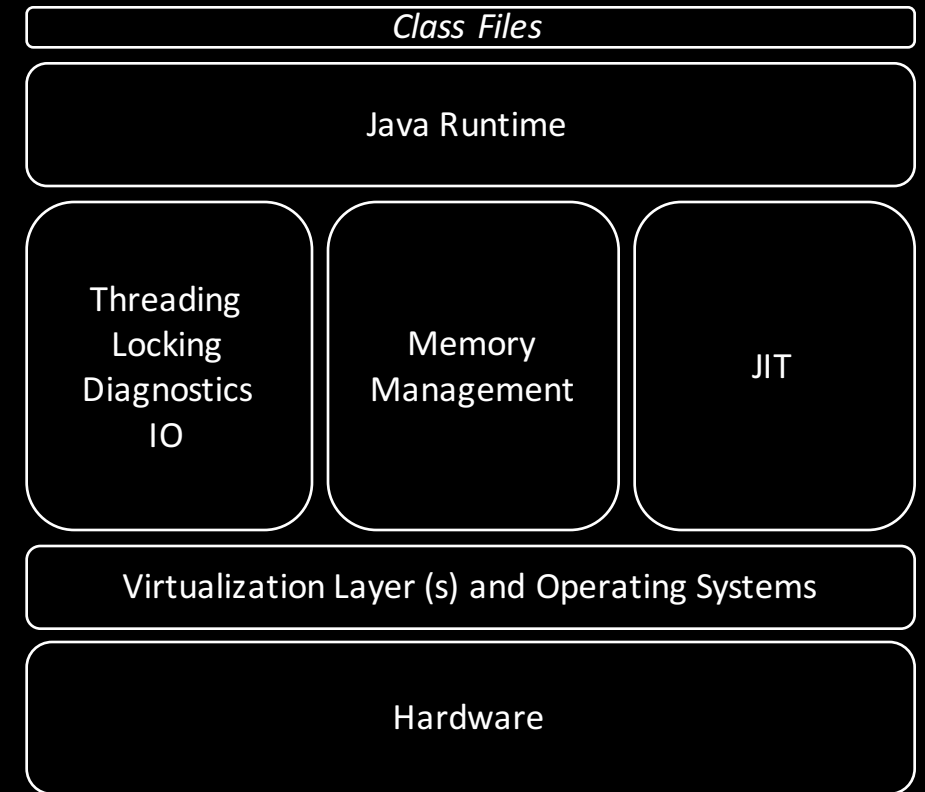
“The existence of a precisely defined virtual machine and object file format need not significantly restrict the creativity of the implementor. The Java Virtual Machine is designed to support many different implementations, providing new and interesting solutions while retaining compatibility between implementations”

@spoole167



Flexible but precise.

- Most of the important bits about JVM execution are flexible in implementation.
- You can drop the API requirements (though you would lose the official coffee cup as your implementation could not pass the JCK)
- Or even drop the Java Language Specification requirements altogether. Classfiles are still Classfiles.
- Or add other JVM related specifications



What does this mean for scaling Java?

JVMs can scale from the very smallest to the very largest devices

Tiny
winy



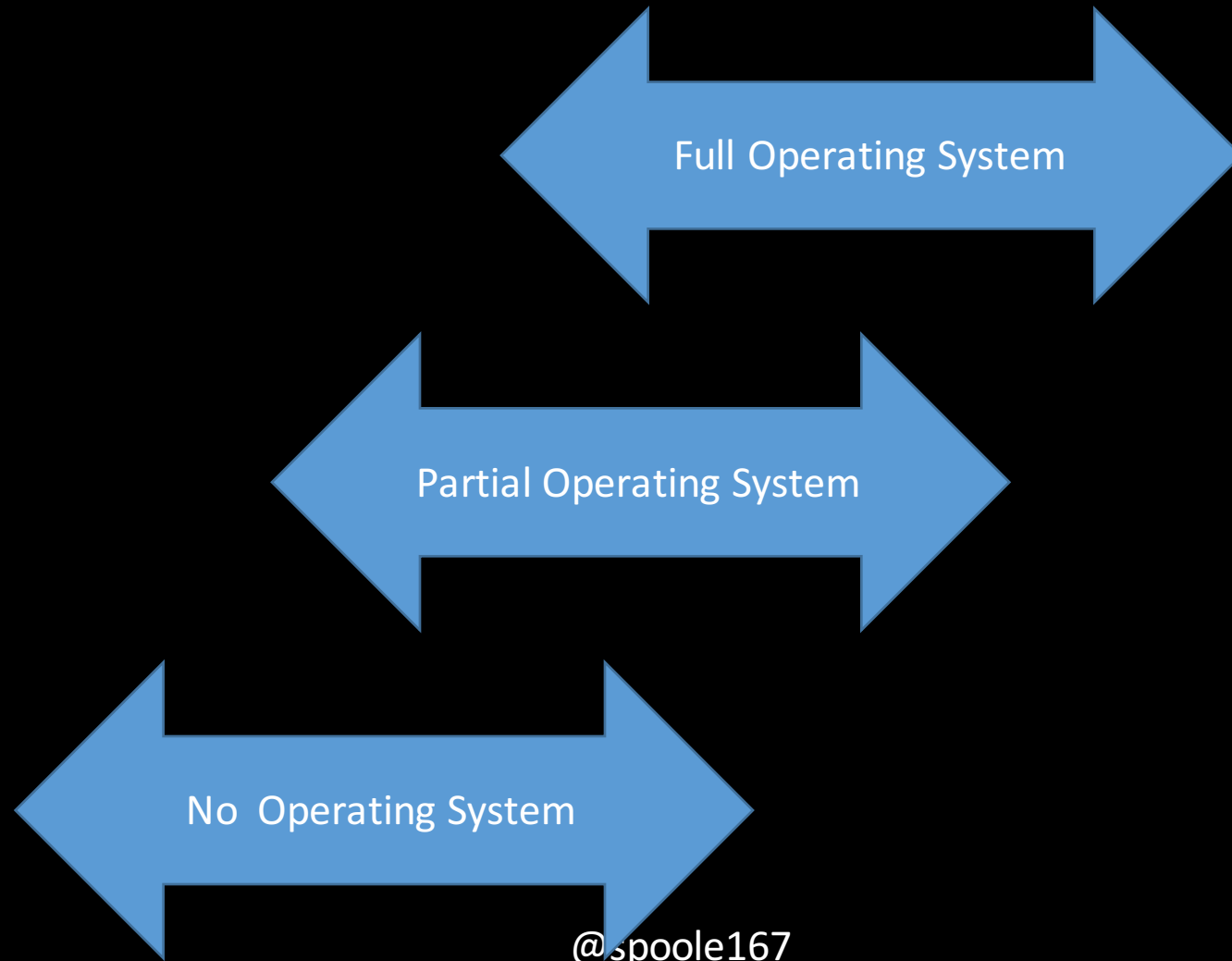
Ginormous
humongous



Technically the major division is whether the device has a complete operating system

Tiny
winy

Ginormous
humongous

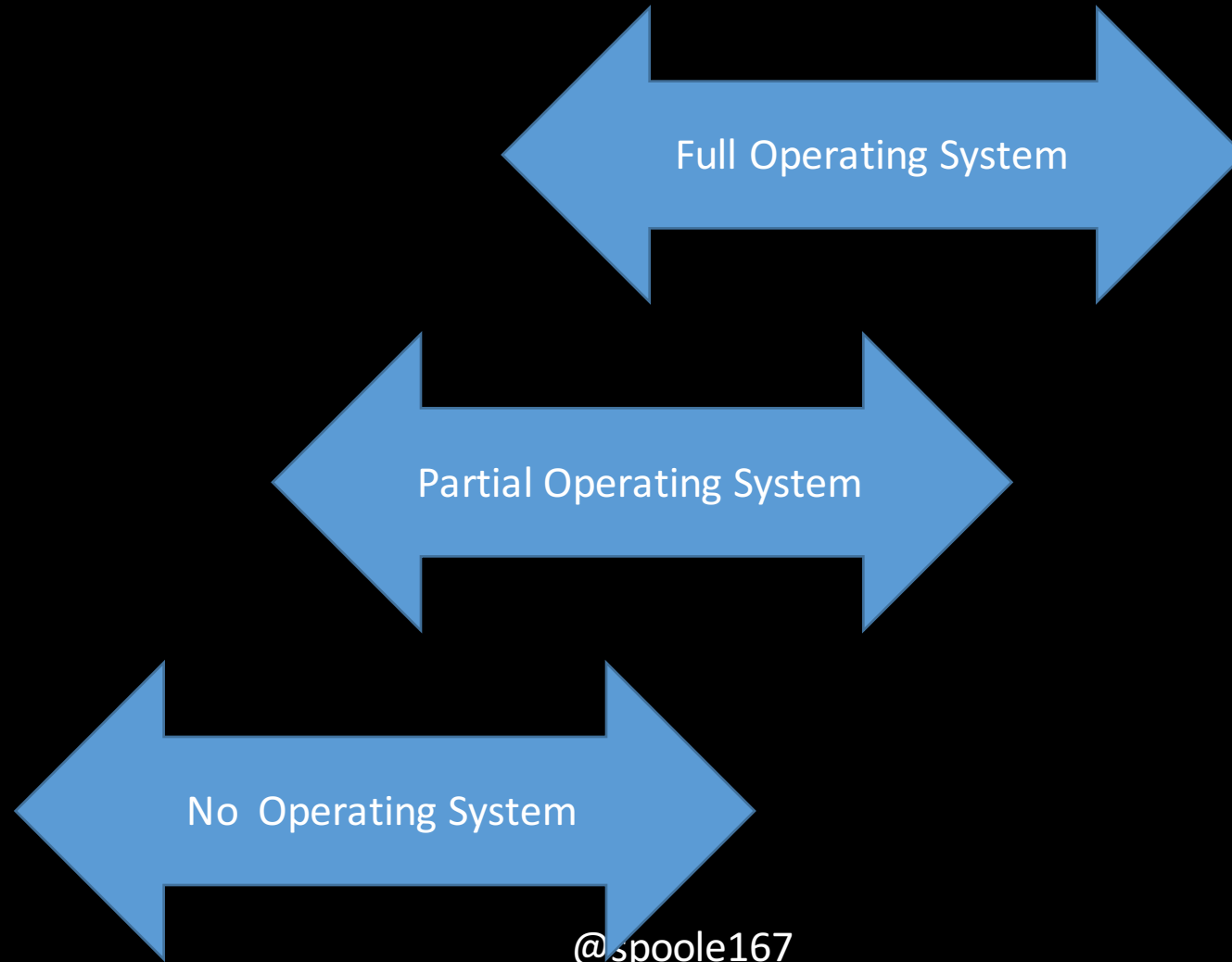


@spoole167

So how far can we take Java?

Tiny
winy

Ginormous
humongous

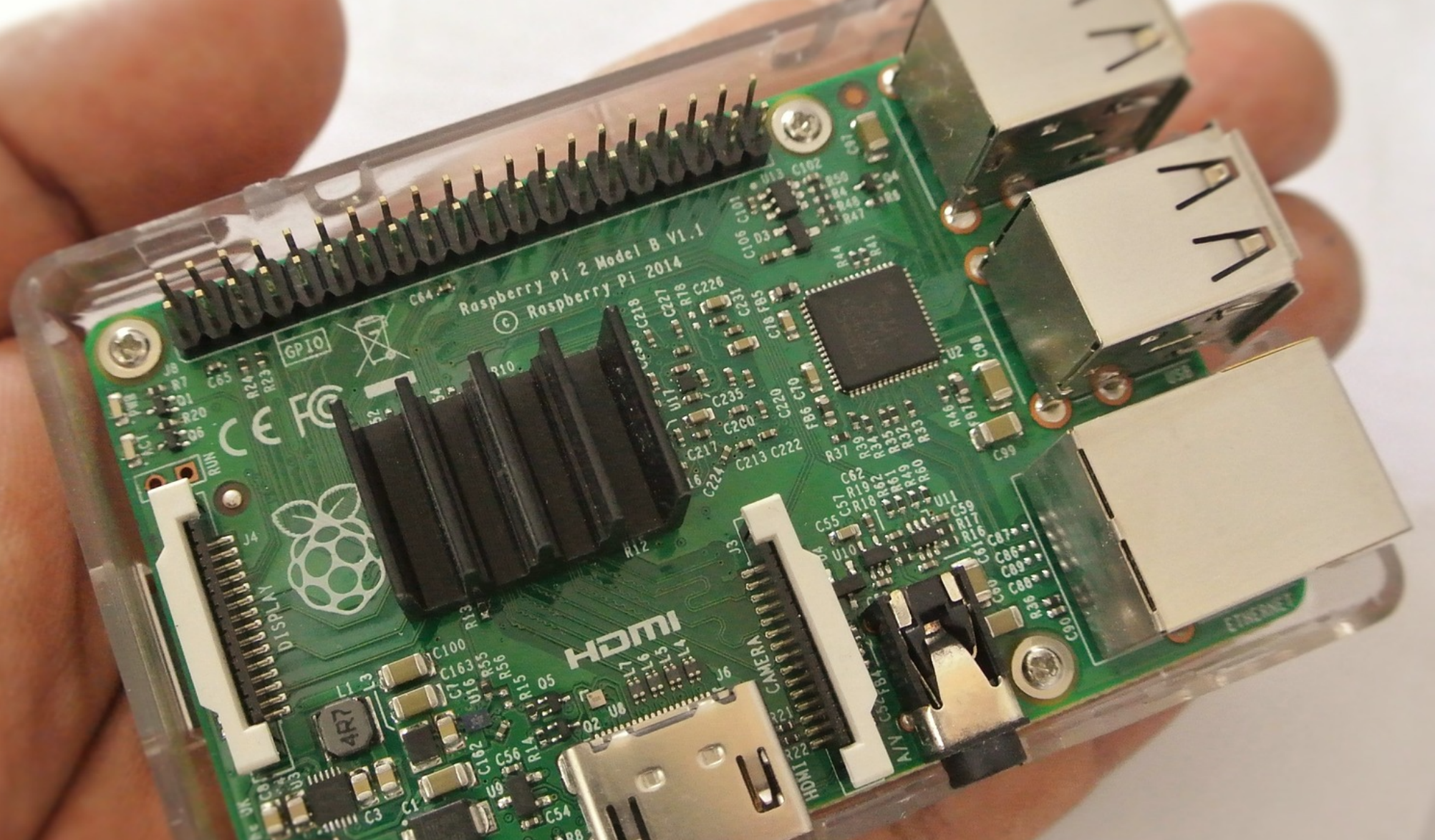


@spoole167

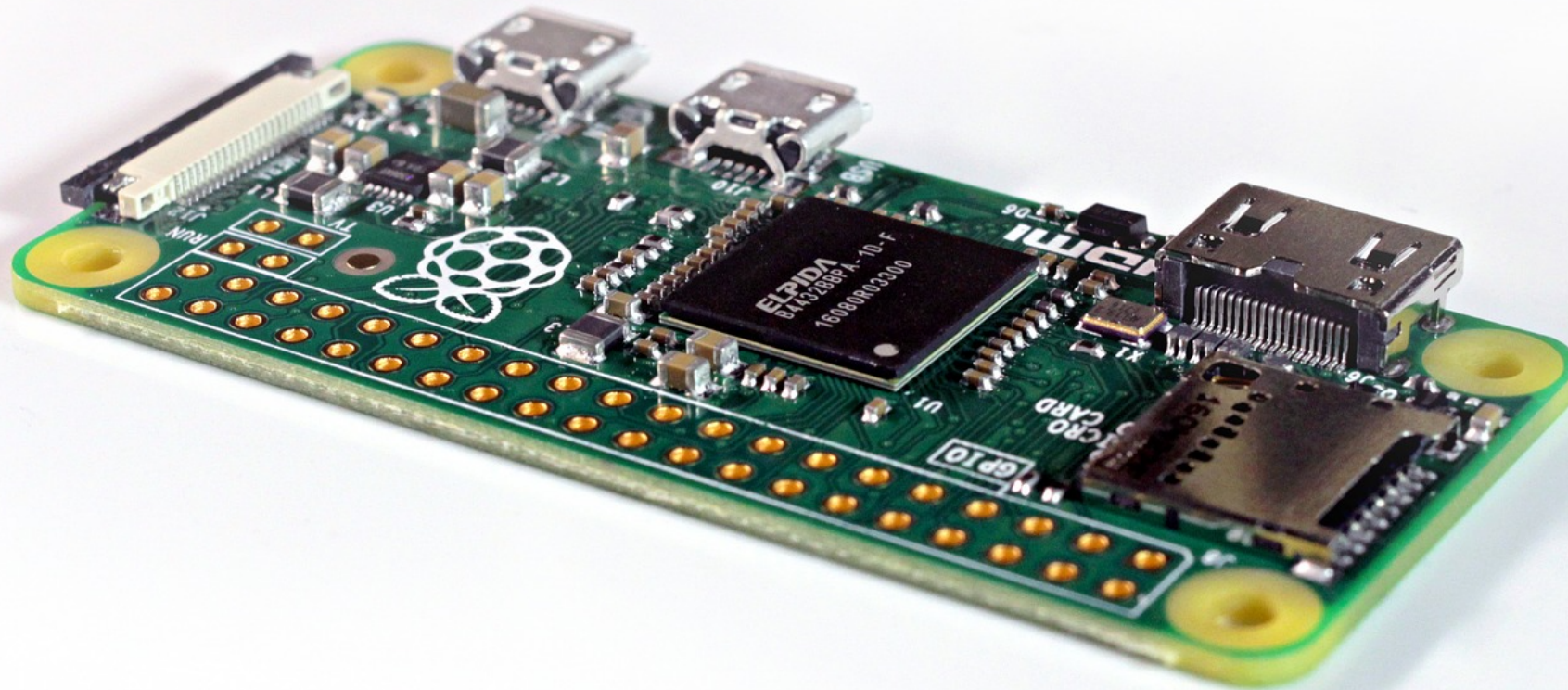


On the small side





Raspberry Pi 3 – smallest Java SE device?



Raspberry Pi Zero – smallest Java SE device?

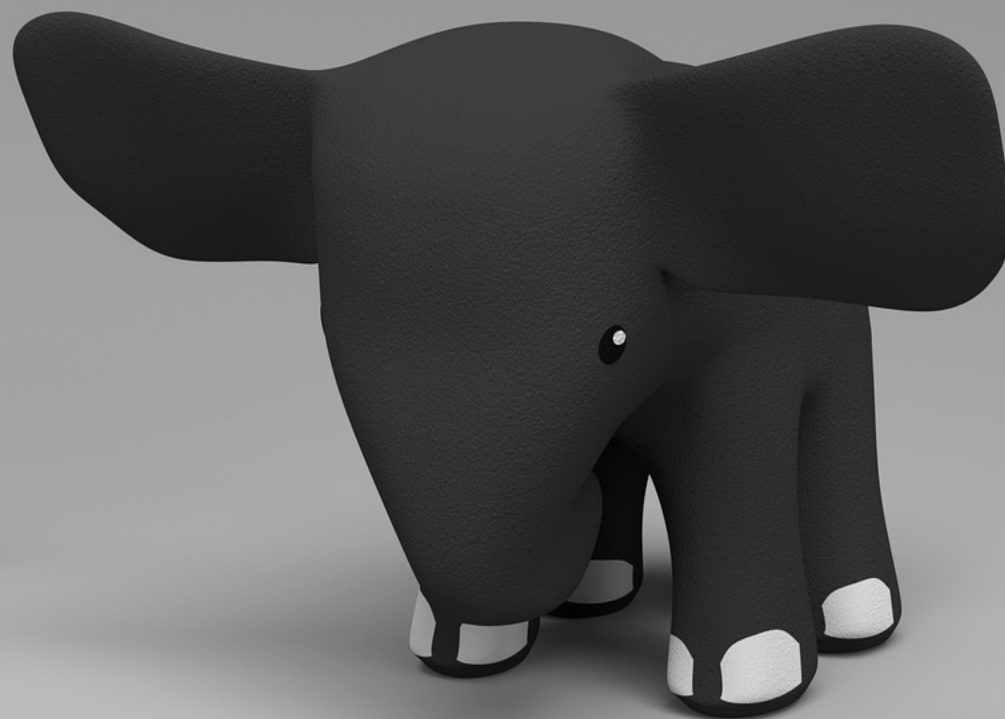
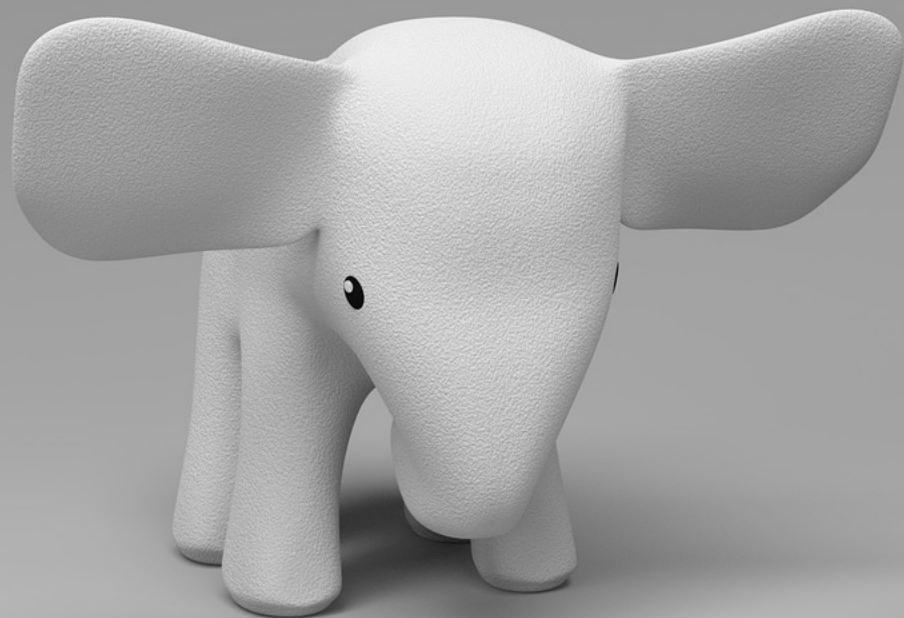
Raspberry Pi

- Pi 3 runs Arm V8 Broadcom BCM2837 64Bit Quad Core running at 1.2GHz
- Zero runs 1GHz single-core CPU with 512MB RAM
- Minimum requirements for Java is 130MB RAM
- Lots of alternatives to the PI
 - Most of them are ARM based
 - Wifi, Bluetooth
 - GPU
- Full Operating System
- Full Java SE support



So not really
that small.

Lets go much smaller



2010 Marketing stats (from Oracle)

- 1.1 billion desktops run Java
- 930 million Java Runtime Environment downloads each year
- **3 billion mobile phones run Java**
- **31 times more Java phones ship every year than Apple and Android combined**
- **100% of all Blu-ray Players Run Java**
- **1.4 billion Java Cards are manufactured each year.**

<http://www.oracle.com/us/corporate/press/173712>

@spoole167

3 Billion Devices Run Java

ATMs, Smartcards, POS Terminals, Blu-ray Players, Set Top Boxes, Multifunction Printers, PCs, Servers, Routers, Switches, Parking Meters, Smart Meters, Lottery Systems, Airplane Systems, IoT Gateways, Programmable Logic Controllers, Optical Sensors, Wireless M2M Modules, Access Control Systems, Medical Devices, Building Controls, Automobiles...



ORACLE®

oracle.com/java
or call 1.800.ORACLE.1

Copyright © 2014, Oracle and/or its affiliates. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

M_115M_CRP00087_Java3BDevices



Oracle 
@Oracle

Follow



#DidYouKnow there are over 13 billion devices running **#Java?** **#JavaOne** **#oow15**
#Java20



5:03 pm - 25 Oct 2015

29 Retweets 22 Likes



 2

 29

 22

Java Ring

20 pounds from Ebay



There's this thing called "Java Card"

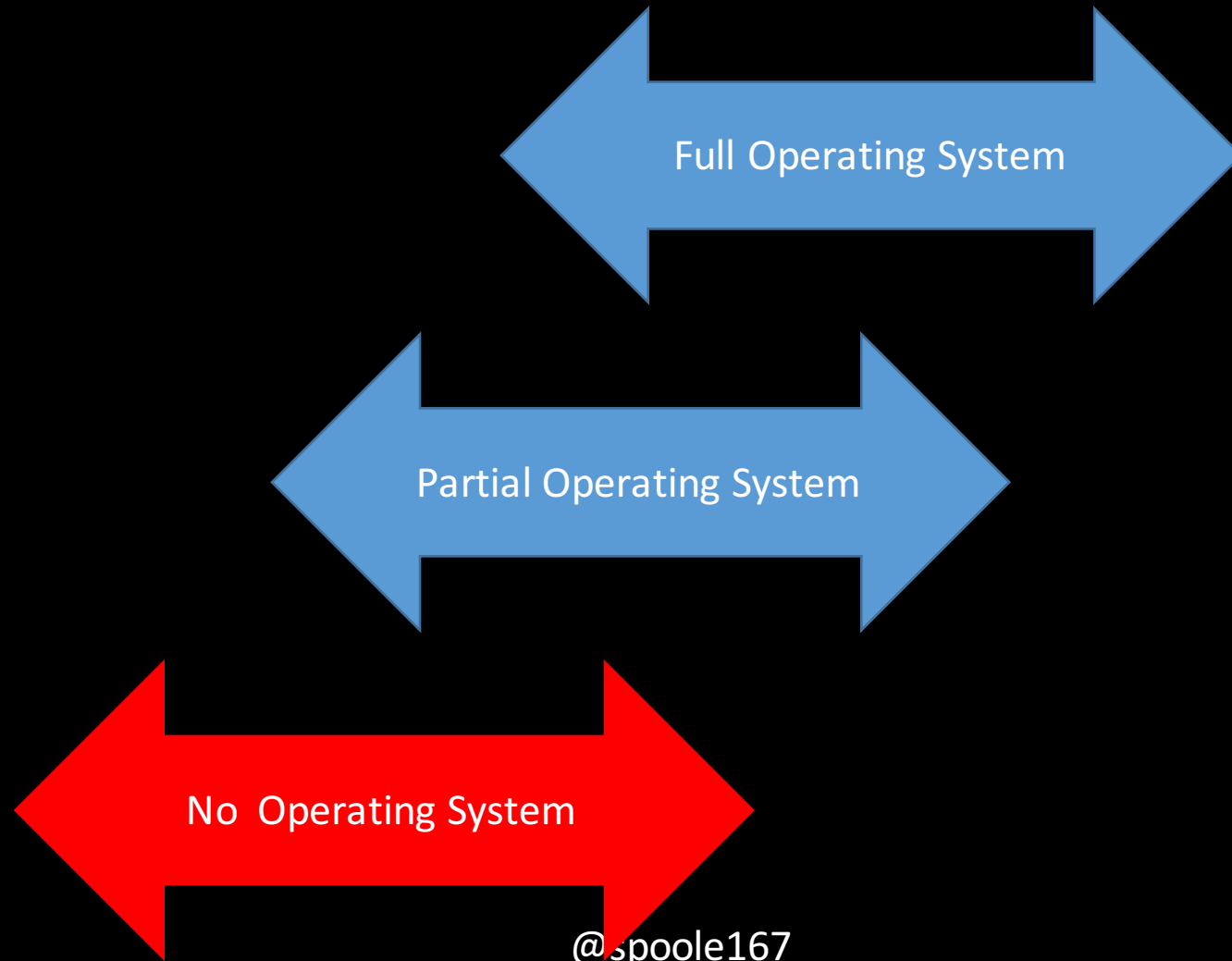
Ships 6 Billion units a year (up from 1.4 Billion in 2010)



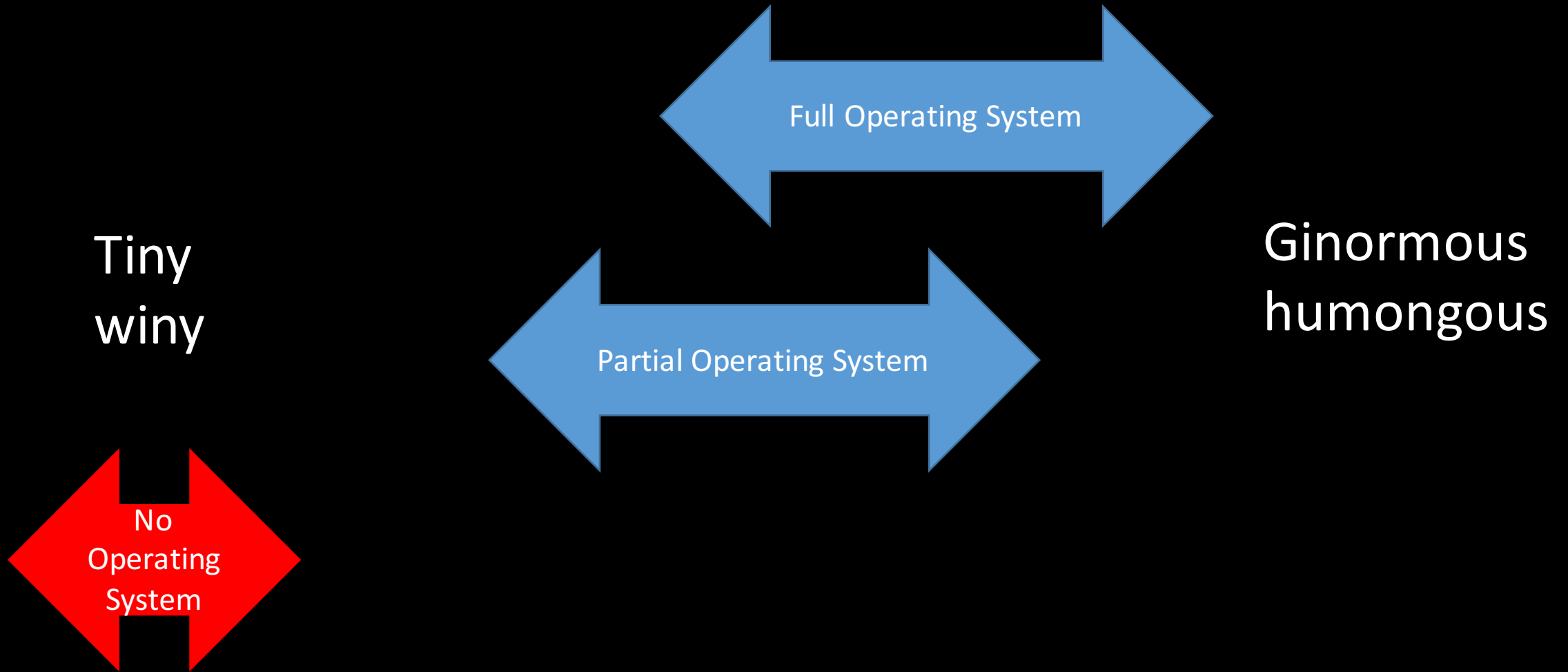
Java Card for when you have no operating system

Tiny
winy

Ginormous
humongous



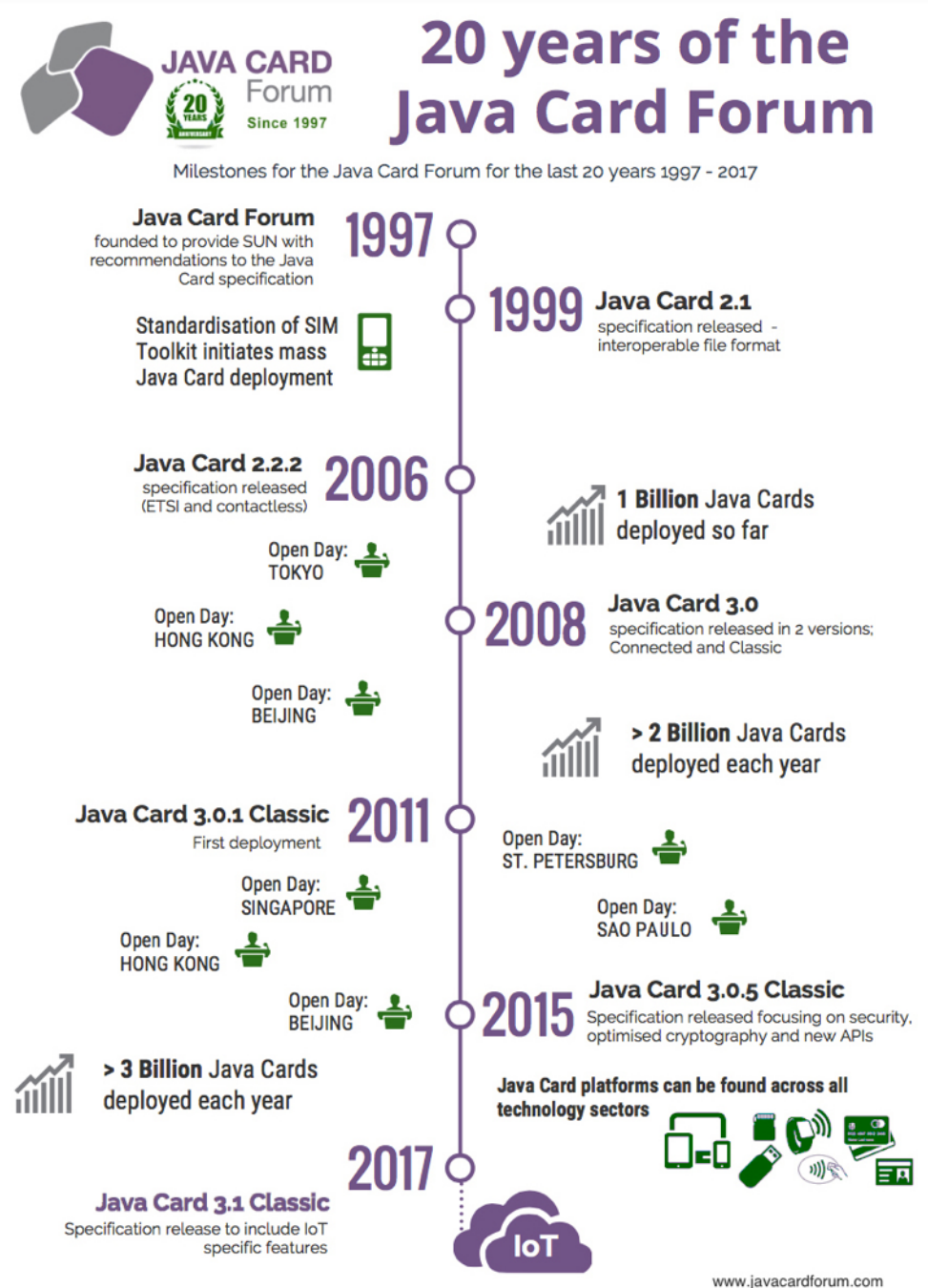
Java Card for when you have no operating system



The 'card' is just one of the delivery mechanisms

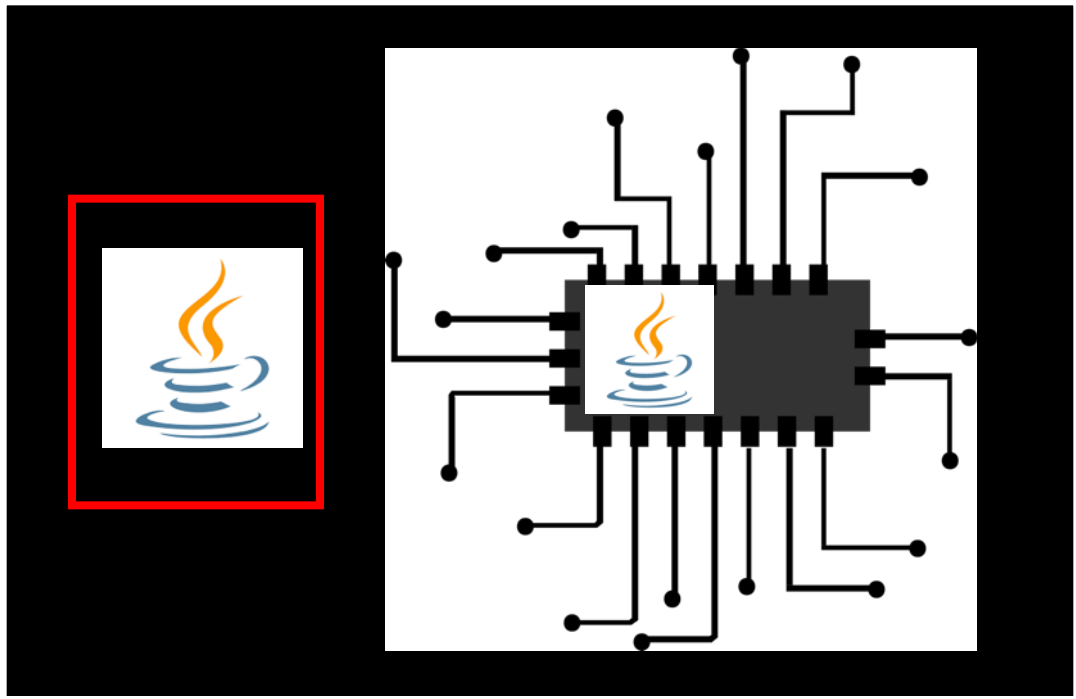
- Think super constrained, very low power environments (or even no power)
 - Maybe 1MB of RAM – though can be as low as 2KB
- A complete Java ecosystem running for the last 20 years
- A really minimal subset of Java with special additions for secure devices
- <https://javacardforum.com/>

@spoole167



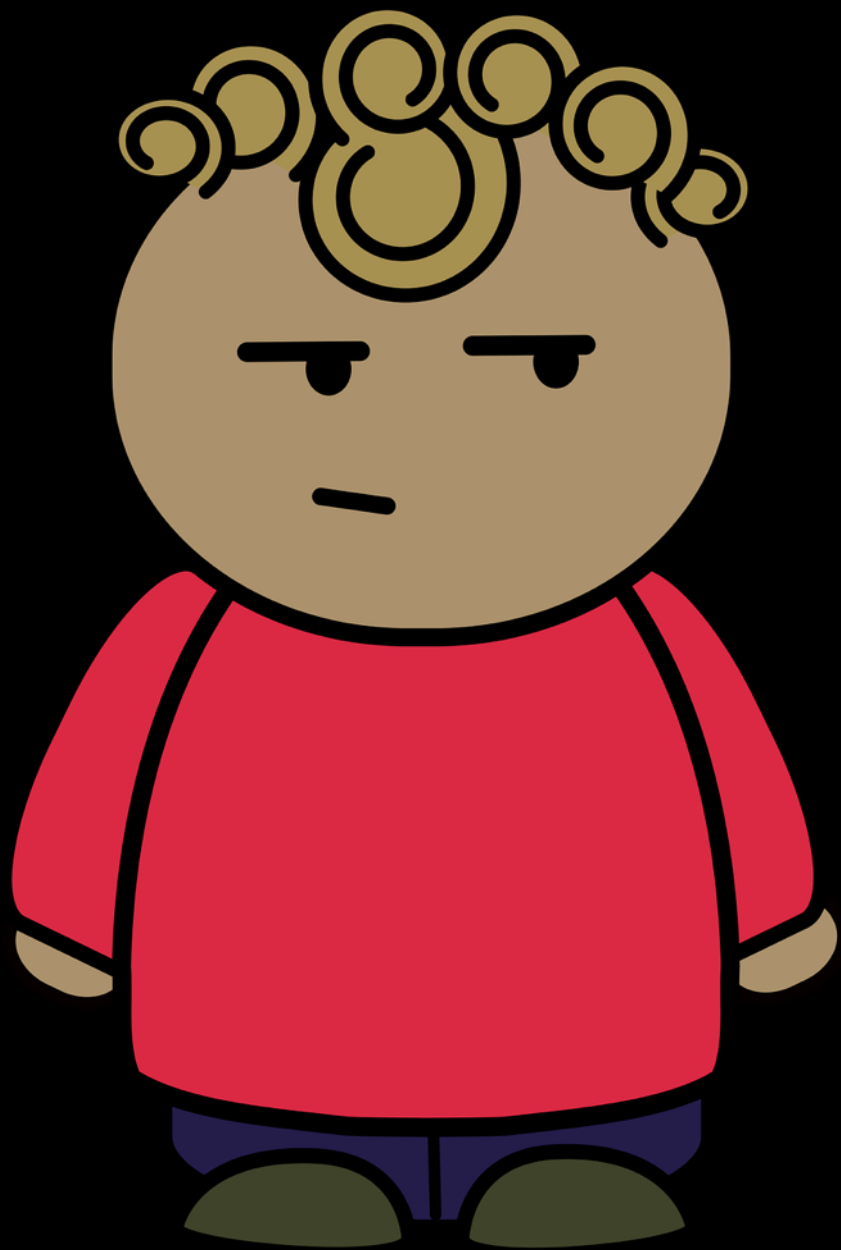
How small can you go?

On the card



In the processor

Next to the processor (separate IC)



Wow – Java inside a chip

What's the catch?

Java Card API specification
Java Card Runtime Environment spec
Java Card Virtual Machine spec

Java Card JVM

- ✓ Available types: Boolean, byte, short, (int is optional)
 - ✓ One dimensional arrays
 - ✓ May have GC
 - ✗ threads, cloning, longs, doubles,
 - ✗ Many of the usual object methods
 - ✗ Characters or Strings!
 - ✗ Keywords like native, sync, transient, volatile etc
- The JVM is always on – it never ends
 - A Java SE programmer is going to find Java Card a little different

Java Card JVM

- Becoming a major IOT edge device type
- Has security physically built in
- Low Power
- Small cost

Want to debug a Java Card in the field?
Want to patch 6 million cards?

Writing code in Java means less chance of code problems.

no hand crafting assembler for small devices

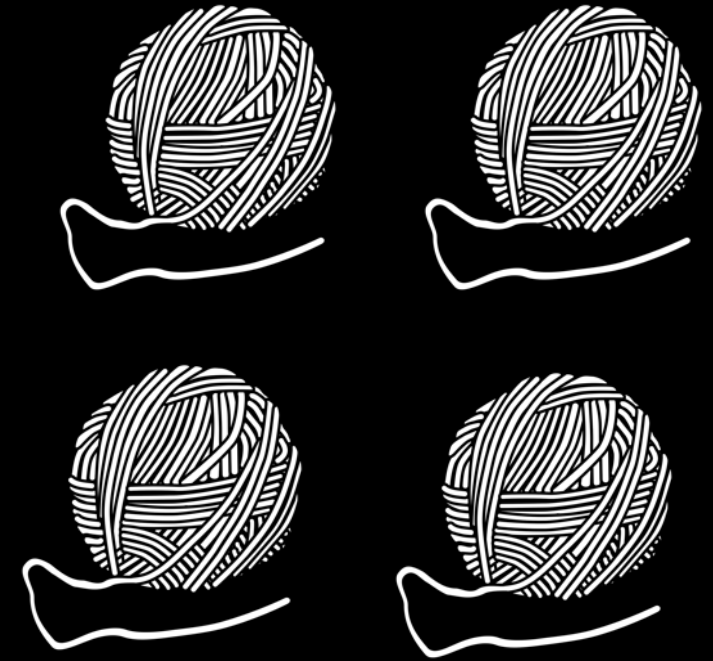
Java offers a superior developer experience

Super sizing the Java SE elephant



Various views of what to scale up

- Larger heaps?
- More threads?
- More JVMs?



Large Heaps



IBM z14 at a glance



Announce: July 17, 2017

System, Processor, Memory

Five hardware models: M01, M02, M03, M04, M05

10 core 5.2GHz 14nm PU SCM

1 - 170 PUs configurable as CPs, zIIPs, IFLs, ICFs

Increased Uniprocessor capacity

Up to 33 sub capacity CPs at capacity settings 4, 5, or 6

CPC Drawers and backplane Oscillator

Enhanced SMT and new instructions for SIMD

Enhanced processor/cache design with 1.5x more on-chip cache sizes

Up to 32 TB DRAM, protected by Redundant Array of Independent Memory (RAIM)

Virtual Flash Memory (VFM)

192 GB HSA

Improved pipeline design and cache management

I/O Subsystem, Parallel Sysplex, STP, Security

PCIe Gen3 I/O fanouts with 16 GBps Buses

6 CSS, 4 Subchannel sets per CSS

0 – 5 PCIe I/O Drawer Gen3 (no I/O Drawer)

Next generation FICON Express16S+

10 GbE RoCE Express2

Integrated Coupling Adapter (ICA SR) and Coupling express LR for coupling links

Support for up to 256 coupling CHPIDs per CPC

CFCC Level 22

Crypto Express6S and CMPSC compression and Huffman Coding compression

STP configuration and usability enhancements (GUI)

IBM zHyperLink Express

OSA-Express6S

Secure Service Container

RAS, simplification and others

L3 Cache Symbol ECC

Acoustic and thin covers (space saving)

N+1 radiator design for Air Cooled System

Drop "Classic" HMC UI

ASHRAE Class A3 design

Enhanced SE and HMC Hardware (security)

Support for ASHRAE Class A3 datacenter

TKE 9.0 LICC

Largesum TCP/IP hardware Checksum (OSA-Express6S)

Pause-less garbage collection

Universal Spare SCM s (CP and SC)

Simplified and enhanced functionality for STP configuration

Enhanced Dynamic Memory Relocation for EDA and CDR

Virtual Flash Memory (replaces IBM zFlash Express)

PR/SM

Up to 170 CPUs per partition

IBM Dynamic Partition Manager updates

Up to 85 LPARs

16 TB Memory per partition

Large Heaps

-Xmx 1TB, -Xmx 10TB -Xmx 32TB

The 1st challenge is how much memory your system can hold.

Then how much the OS will offer to a process

Heaps of 1 .. 4 TBs are not uncommon

IBM Z14 comes with up to 32 TB

Of which up to 10TB can be offered to a JVM

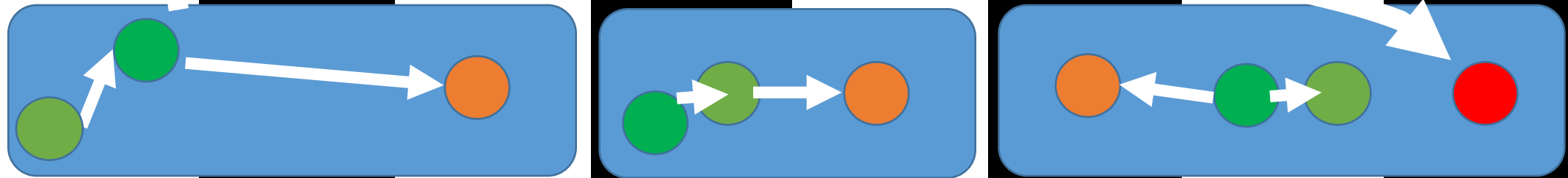
Big heaps start to hit JVM implementation restrictions...

Large Heaps

Heap layouts / GC work to minimize object movement and so tend to group objects to suit

But at some point processor caching get impacted by the constant fetching from uncached pages

The larger the heap – the more likely there are cache misses



Large Heaps

How big can we go?

The JVM design does not restrict the size

The target hardware has limits on pointer sizes (ie 64bit)

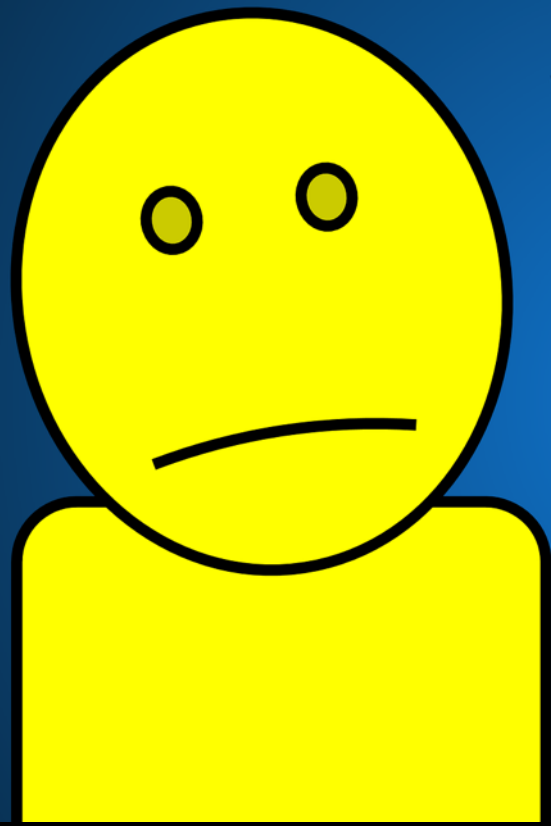
But (in this case) that still gives a range of 16 Exabytes

1 Exabyte is 1 000 000 TB
or 31250 Z14 Mainframes

So the real challenge is improving how we utilise the memory by designing different GC's and improved object locality.

More threads in a single JVM?





**WHAT
WERE YOU
THINKING?**

During development we sometimes see how many threads we can create...

Current high score

> 1000000

Takes hours!

Increasing the number of threads isn't a great way to scale.

Thread switching costs

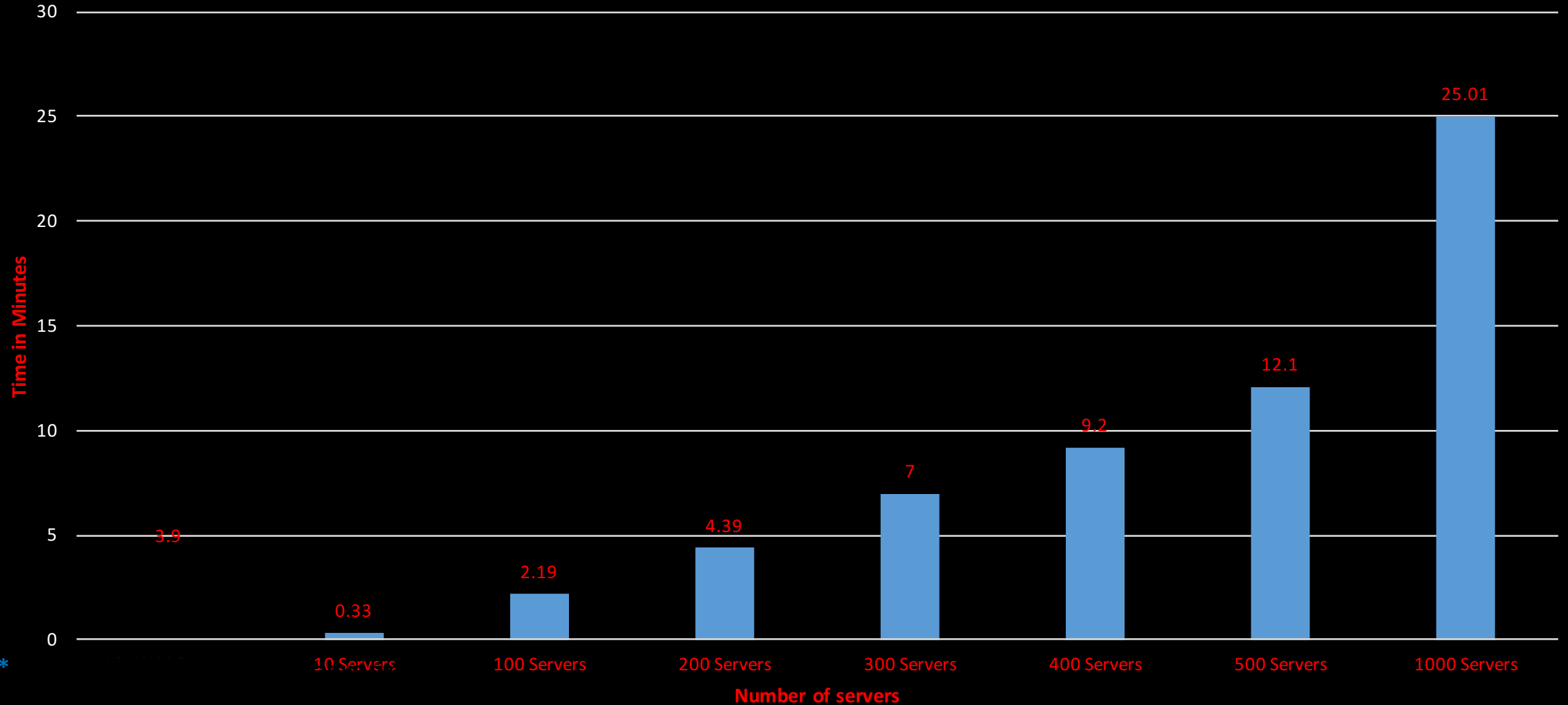
More threads = less memory for the heap

Synchronisation means stopping more threads! – and that takes time.

More concurrent JVMs on a single machine?



Startup Time of 1000 Liberty Servers on Z14



*

@spoole167



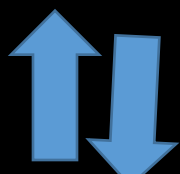
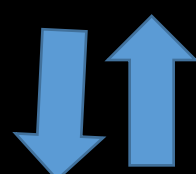
How else can we scale Java?

- Clusters?
- Cloud?

Clusters

Decomposed problem

 Program & data



Executor node

Executor node

Executor node

Executor node

Data Analytics

multiple instances

 Container



Executor node

Executor node

Executor node

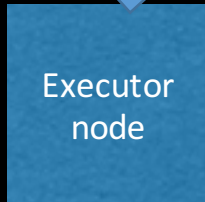
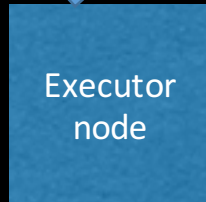
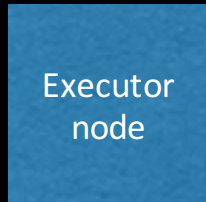
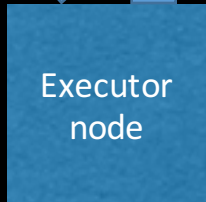
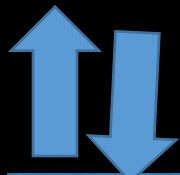
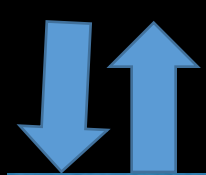
General purpose

Clusters

Where's Java?

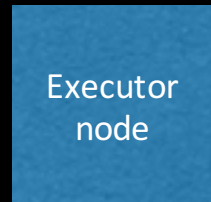
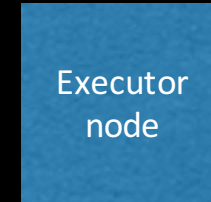
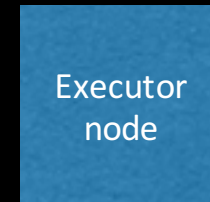
Multiple instances

 Program & data



Data Analytics

 Container



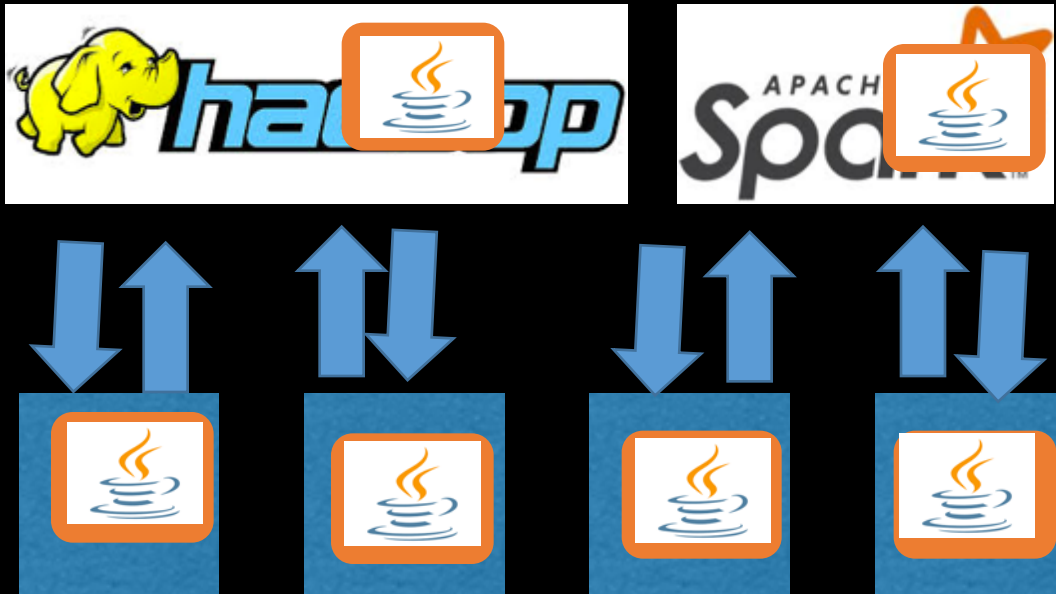
General purpose

Clusters

Where's Java?

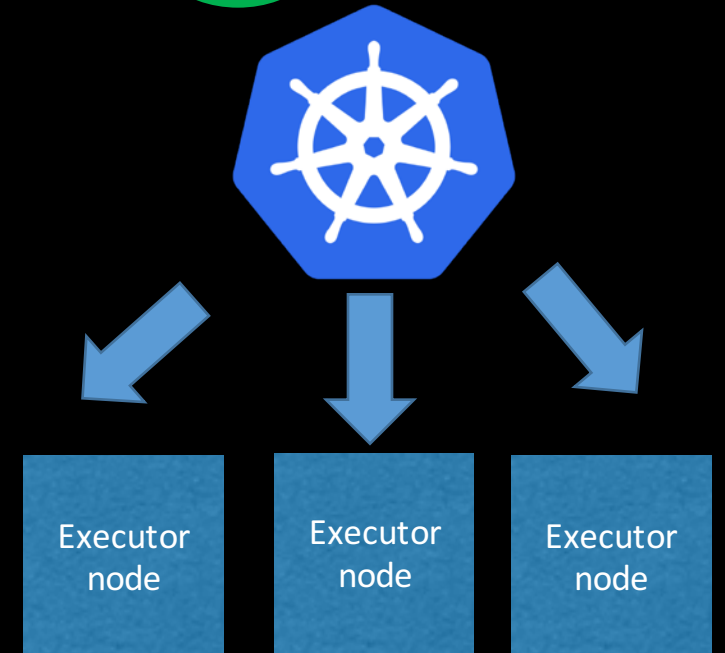
Multiple instances

 Program & data



Data Analytics

 Container



General purpose

Not much we can do to help scale.

- Clusters help scale the application and/or help parallelize it
- Spark clusters of 8000+ nodes are known
- Kubernetes officially supports up to 5000 nodes
- **But there are different economics we can address**

Cloud

Faster, cheaper, easier, better ...



Economics rules



New rules of the game.

Compute == money

New rules of the game.

Compute == money

\$ == GB/hr

New rules of the game.

Compute == money

\$ == GB/hr

-Xmx: \$100

Cloud isn't going away: in fact its coming to you



Cloud is coming to a data center near you

IBM Cloud > Products >

Get the speed of public with the control of private.
IBM Cloud Private. Fast. Flexible. Enterprise-grade.

Build open, cloud-native apps with public services and run them anywhere — on public cloud or on your existing on-premises systems.

→ [Install IBM Cloud Private Community Edition at no charge](#)

→ [Buy IBM Cloud Private](#)

📺 [See how IBM Cloud Private can help \(01:34\)](#)



Announcing IBM Cloud Private

Design, develop, deploy, and manage on premises, containerized cloud applications behind your firewall. [Read the press release.](#)

<https://github.com/IBM/deploy-ibm-cloud-private>

```
git clone  
vagrant up
```

@spoole167



Search items

Helm charts

Deploy your applications and install software packages

ibm-cloudant-dev
Cloudant for Linux.

ibm-charts

ibm-db2warehouse-dev
Db2 Warehouse Developer-C for Non-Prod v2.1.0

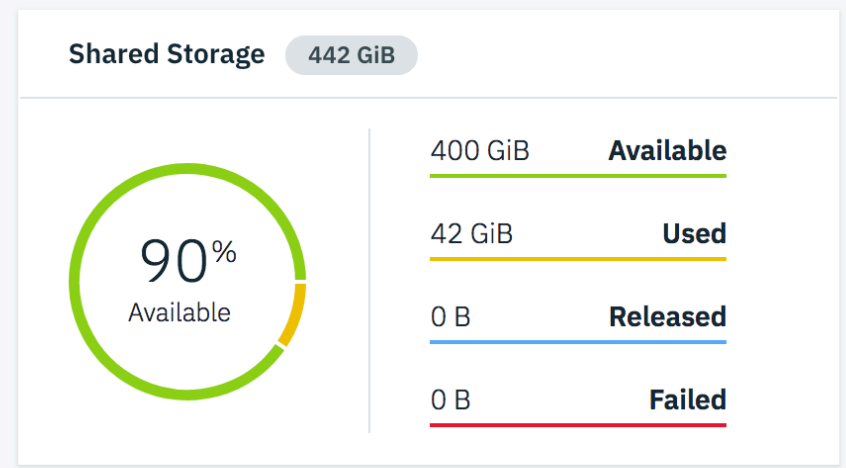
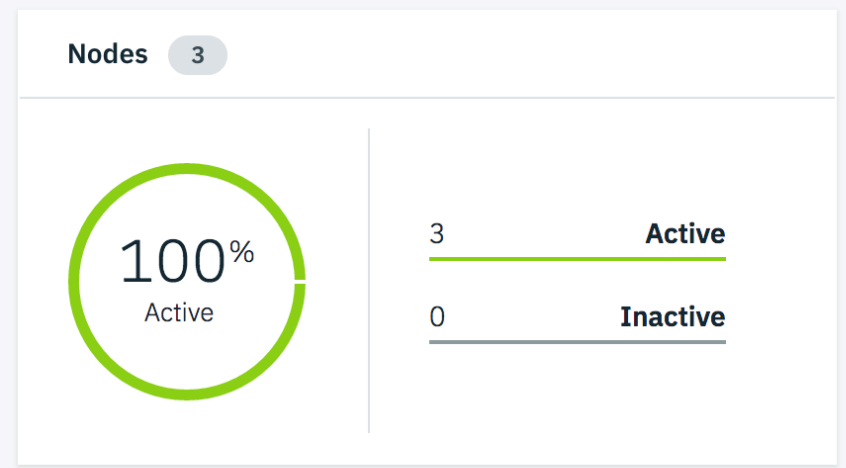
ibm-charts

ibm-icplogging
Log storage and search management solution

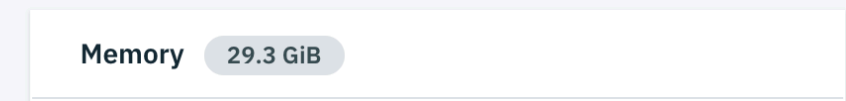
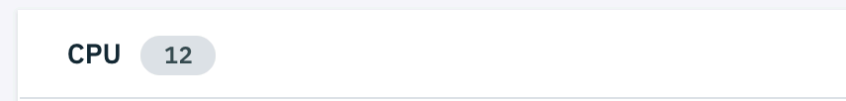
ibm-charts

Dashboard

System Overview



Resource Overview



We need to go from

Very long running, efficient,
monolithic applications

Willing to trade startup time for
throughput

in it for the long haul

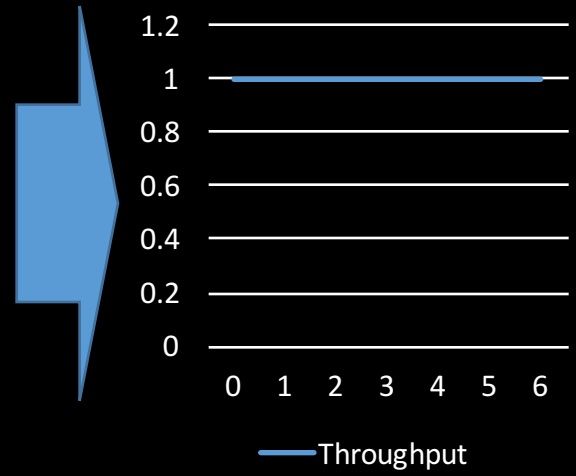
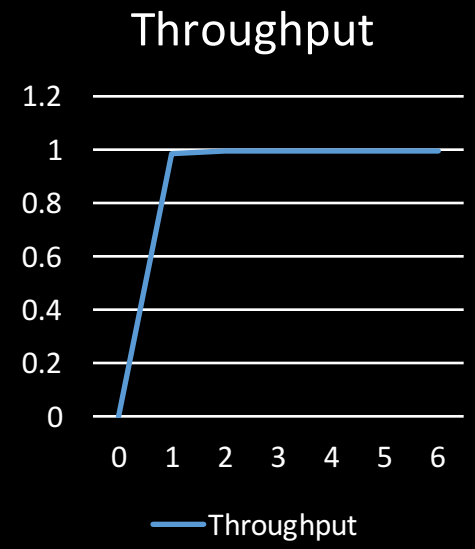
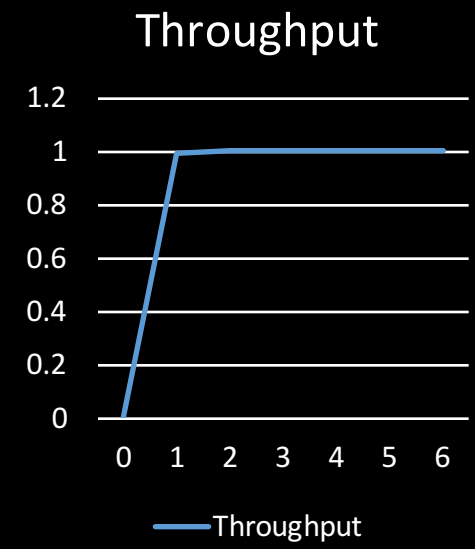
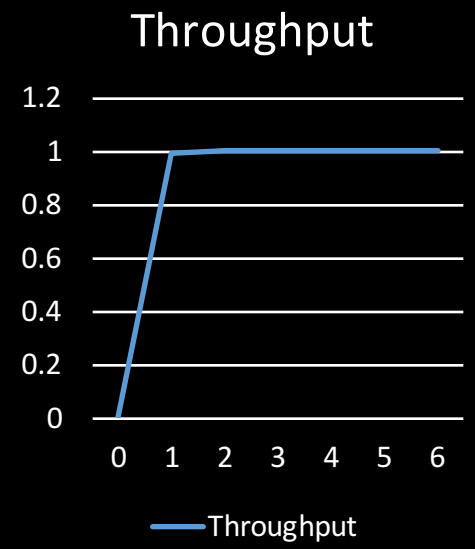
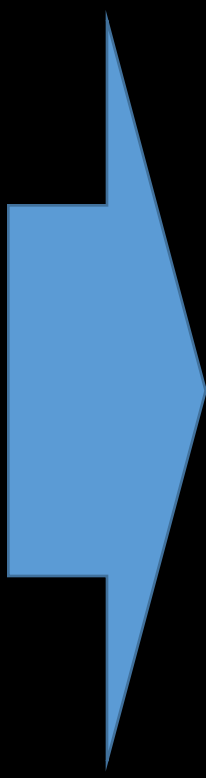
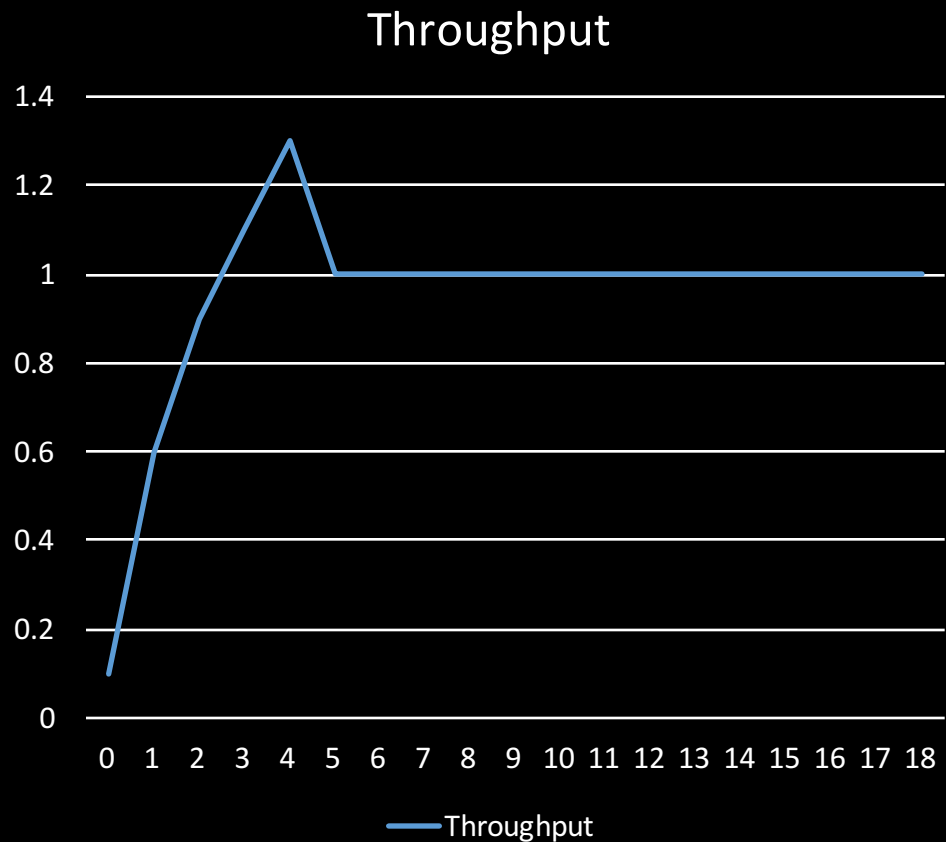


A black and white photograph of several sprinters in a starting crouch on a track. The runners are in the foreground, leaning forward with their hands on the ground, ready to start a race. In the background, a person in a white shirt and cap stands near a metal barrier. The overall scene is dynamic and focused on the beginning of a race.

Now we also need

Short lived, container based, micro service oriented, instant-on, always available, cross server, polyglot services

We need all of these models



Cloud.



Cloud demands smaller footprints, faster startup times

Its about retuning the JVM to do things a little differently.

Some of this is going to need a lot of re-engineering.

It's already begun

```
java -jar ngrinder.jar
```

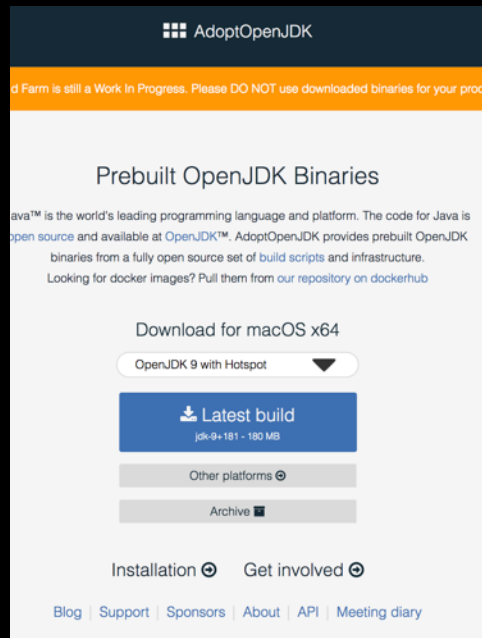
JVM	Process Mem usage (MB)
Hotspot JDK 8	577
OpenJ9 JDK 8	359

Cloud.



And even better...

JVM	Process Mem usage (MB)
Hotspot JDK 8	577
Hotspot JDK 9	340
OpenJ9 JDK 9	240



`java -jar <your jar here>`

<http://www.eclipse.org/openj9/>

<https://adoptopenjdk.net>

OpenJ9



Cloud.



This is all good stuff.

The more important thing is more about what your application needs to do..

Unnecessary baggage (you have loads)

Java applications have to get lighter.

Java 9 modularity will help but you have to consider footprint across the board.

Choose your dependencies wisely

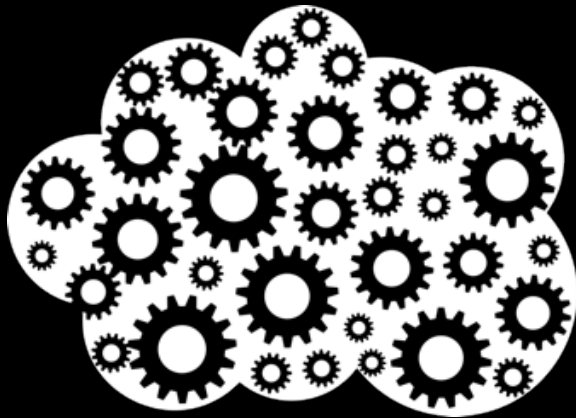
Your choice of OS & distribution is important.

The aim is **'carry on only'**

Your application isn't going on a long trip



The future starts here...



Data Analytics

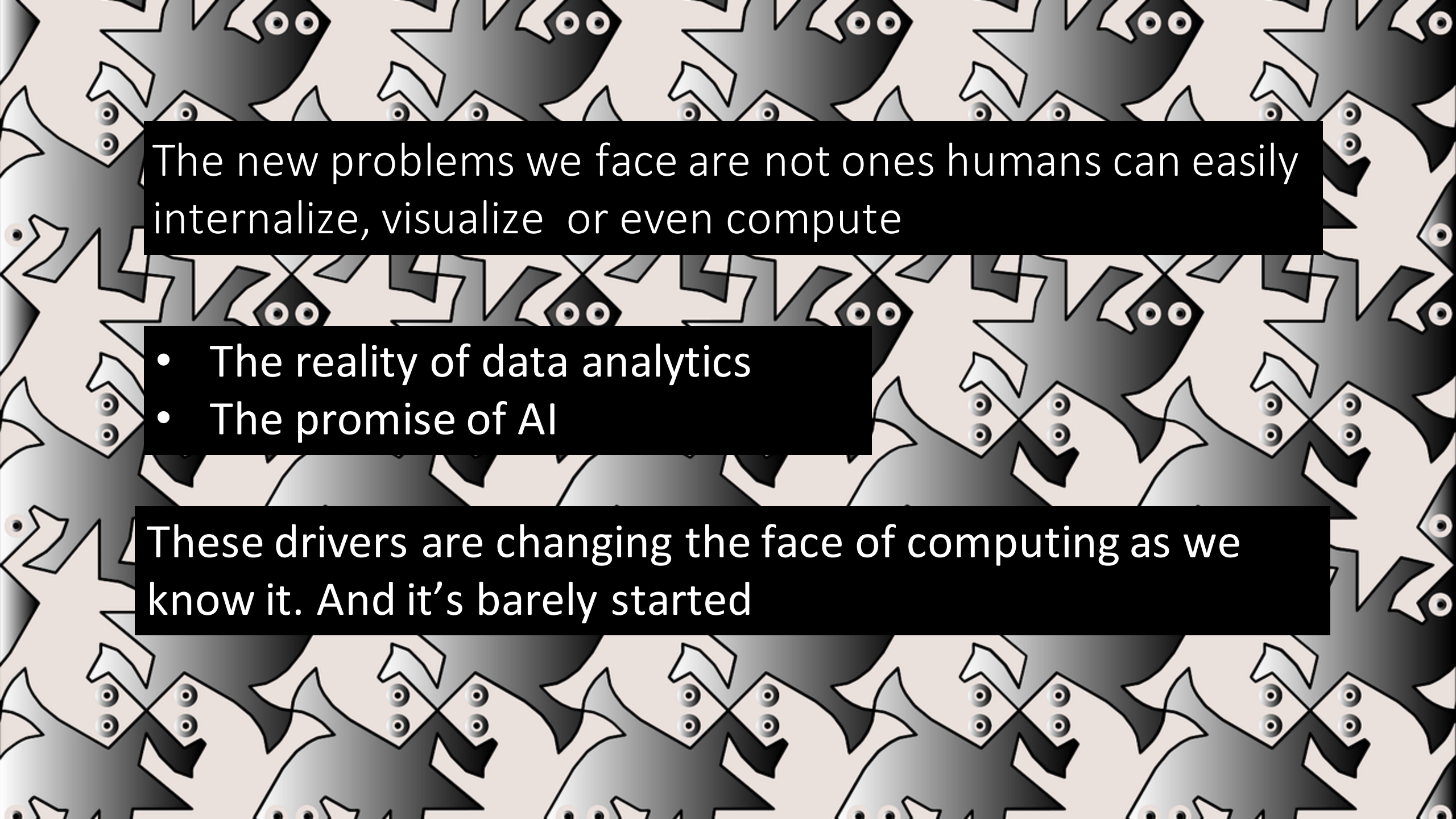


Machine Learning



But it's still driven by economics





The new problems we face are not ones humans can easily internalize, visualize or even compute

- The reality of data analytics
- The promise of AI

These drivers are changing the face of computing as we know it. And it's barely started

The common challenges

Effective optimization

Visualisation of complex patterns

Extracting meaning from enormous amounts of data.

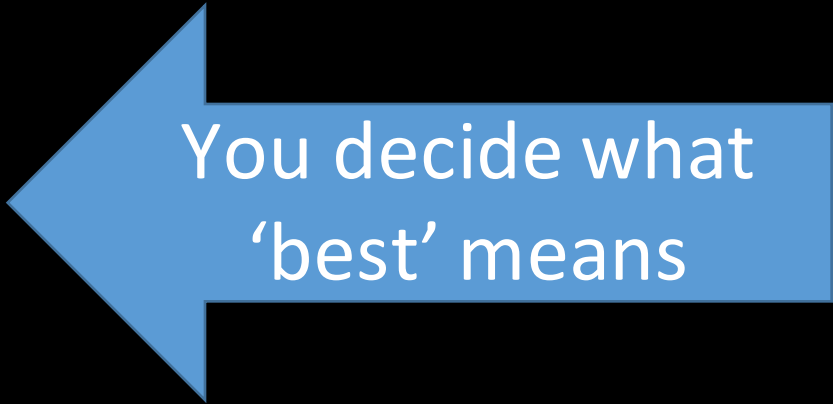
All things human beings are poor at doing

And so are current computer architectures

Machine Learning etc

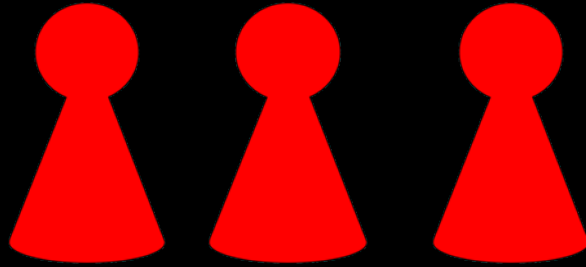
- Helps us find reasonable answers to questions.
- No guarantees on it being the best answer.

Organize a group of people at a table.
So that you get the 'best' conversation

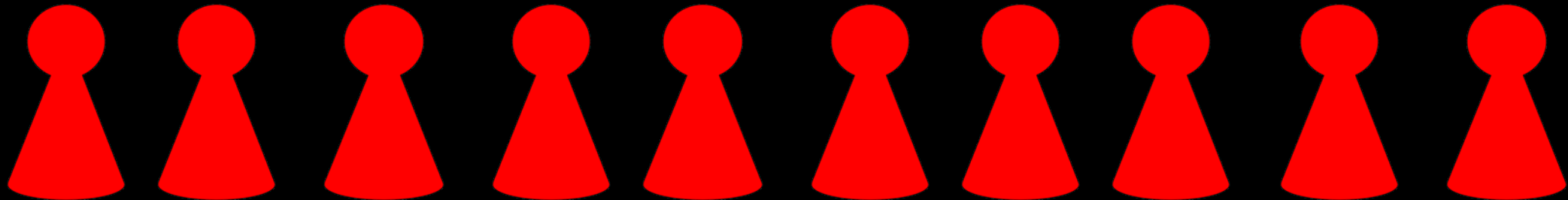


You decide what
'best' means

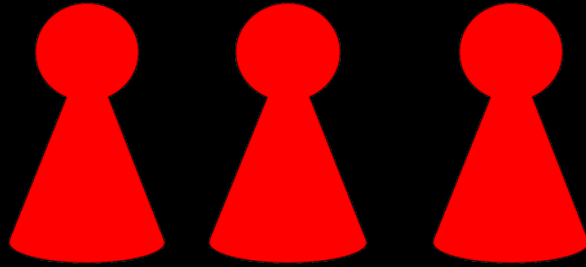
Two or three people - easy



What about 10? - how many combinations to consider?



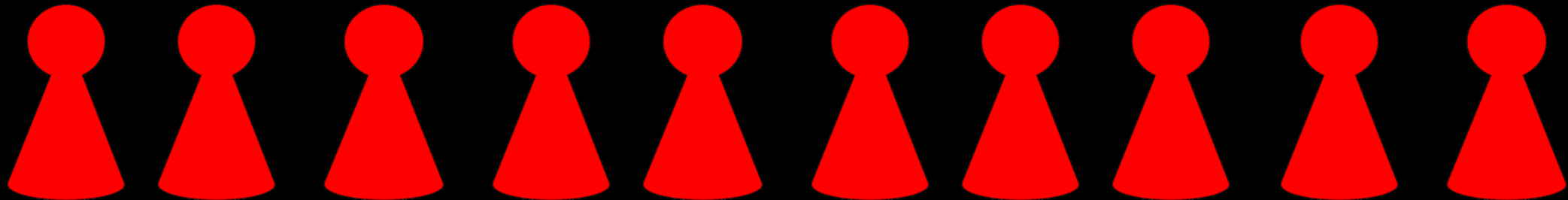
Two or three people - easy



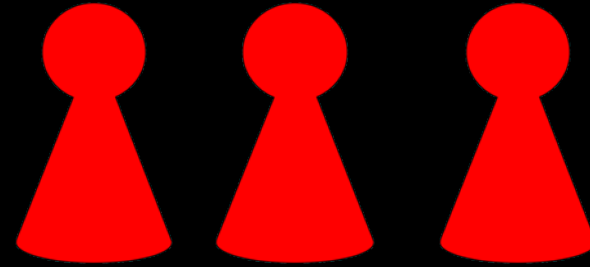
What about 10? - how many combinations to consider?

3628800

Going to need
some help here

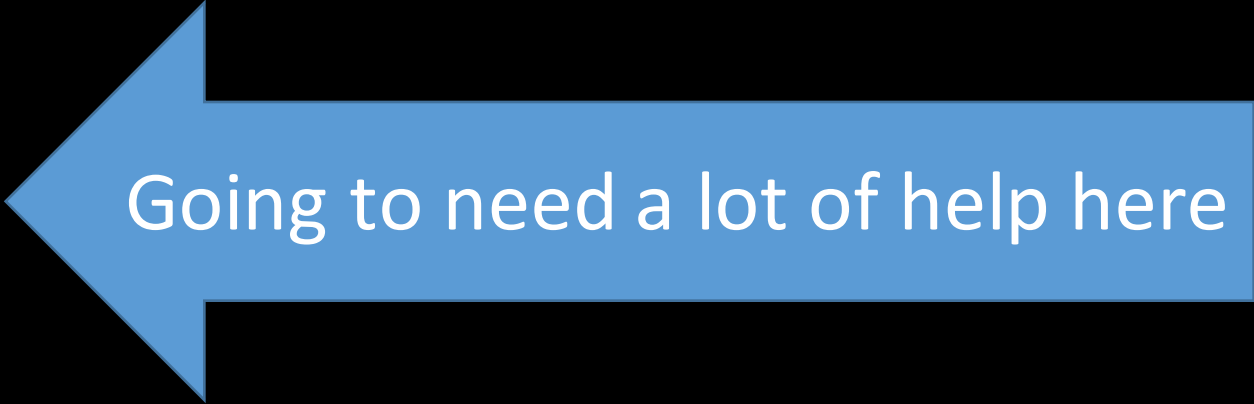


Two or three people - easy



What about 150?

5713383956445854590478932865261054003189553
5786011264182548375833179829124845398393126
5744886753111453771078787468542041626662501
9868450446635594919592206657494259209573577
8929325357290444962472405416790722118445437
1222696755200000000000000000000000000000000
00000

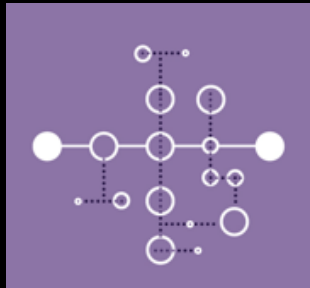


It's this level of scale , complexity and
uncertainly we now have to deal with...

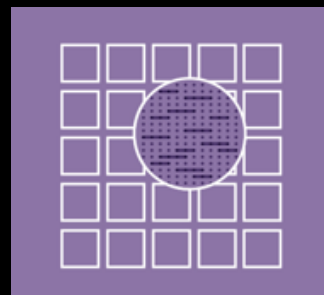
Imagine creating systems that can



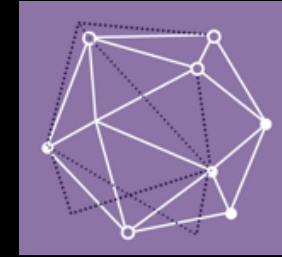
Tailor responses to the personalities of your customers without meeting a single one of them.



identify their own inefficiencies-and address them automatically-in real time.



Uncover patterns, resources, trends and other competitive advantages invisible to competitors and their information systems.



improve themselves over time, learning from and adapting to the world around them.


```

#define REG_PG      vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)      (func)

#define SWAP_ALLOCATE(nr)      (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %%esp, %0, %3" : : "r" (0)); \
    if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
        pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
    PUT_PARAM_RAID(2, sel) = get_state_state();
    set_pid_sum((unsigned long)state, current_state_str(),
        (unsigned long)-1->lr_full; low;
}

```

```
/*  
 * Copyright (c) 2006-2010, Intel Mobile Communications. All rights reserved.  
 *  
 * This program is free software; you can redistribute it and/or modify it  
 * under the terms of the GNU General Public License version 2 as published by  
 * the Free Software Foundation.  
 *  
 * This program is distributed in the hope that it will be useful,  
 * but WITHOUT ANY WARRANTY; without even the implied warranty of  
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
 *  
 * GNU General Public License for more details.  
 *  
 * You should have received a copy of the GNU General Public License  
 * along with this program; if not, write to the Free Software Foundation,  
 * Inc., 675 Mass Ave, Cambridge, MA 02139, USA.  
 */
```

```
#include <linux/kexec.h>  
#include <linux/errno.h>  
#include <linux/io.h>  
#include <linux/platform_device.h>  
#include <linux/multi.h>  
#include <linux/ckevent.h>
```

Machine Learning works because (mostly) humans design and teach it

A ML solution can be a significant investment.
It is also a significant business opportunity

JVMs need to be able to offer good machine learning and data analysis capabilities

Back to clusters



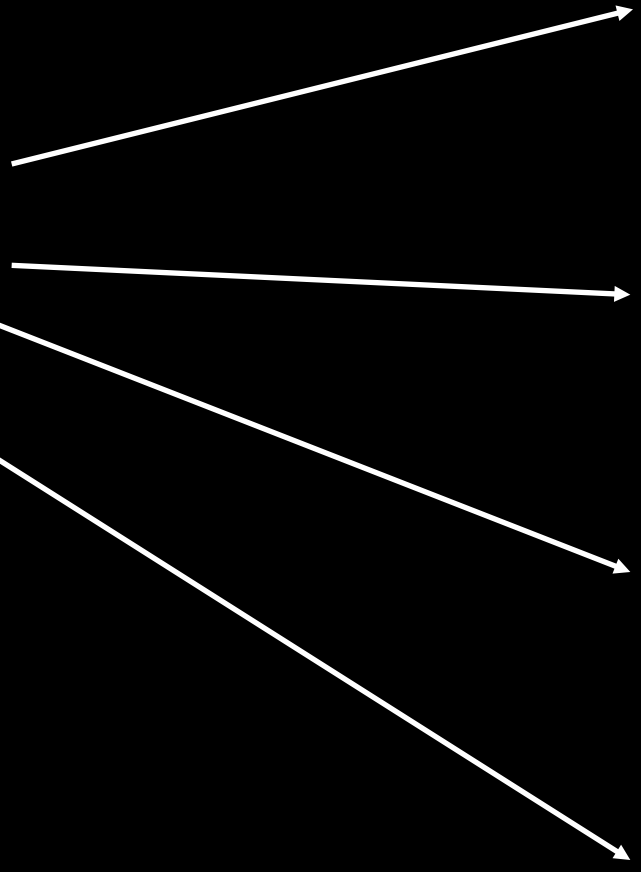
Executor
node

Executor
node

Executor
node

Executor
node

CPU



Executor
node

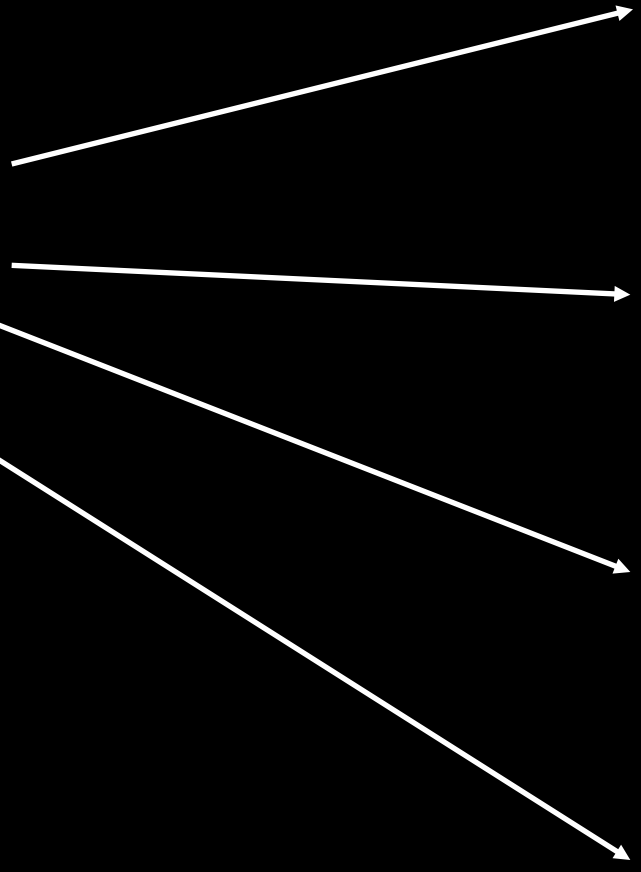
Executor
node

Executor
node

Executor
node

CPU

GPU



Executor node

Executor node

Executor node

Executor node

CPU

GPU

FPGA

ASIC



GPU's don't work like CPU's



They want their data in different forms
They behave differently

You'll have to think differently too

Step 1

```
if(a) {  
    // do this  
} else {  
    // do this instead  
}
```

Step 2

On a standard CPU
Each thread runs in parallel
With its own 'program counter'

Step 1

```
if(a) {  
    // do this  
} else {  
    // do this instead  
}
```

Step 2

Independently.

Step 1

```
if(a) {
```

Step 2

```
    // do this
```

```
} else {
```

Wait

```
    // do this instead
```

```
}
```

Step 1

```
if(a) {
```

Wait

```
    // do this
```

```
} else {
```

Step 3

```
    // do this instead
```

```
}
```

On a GPU

Each thread runs in sync
sharing a program counter

The length of time for the
program to complete is
a factor of all the
paths traveled

GPU's are best when
data driven!

Numbers of GPUs in data centers are growing

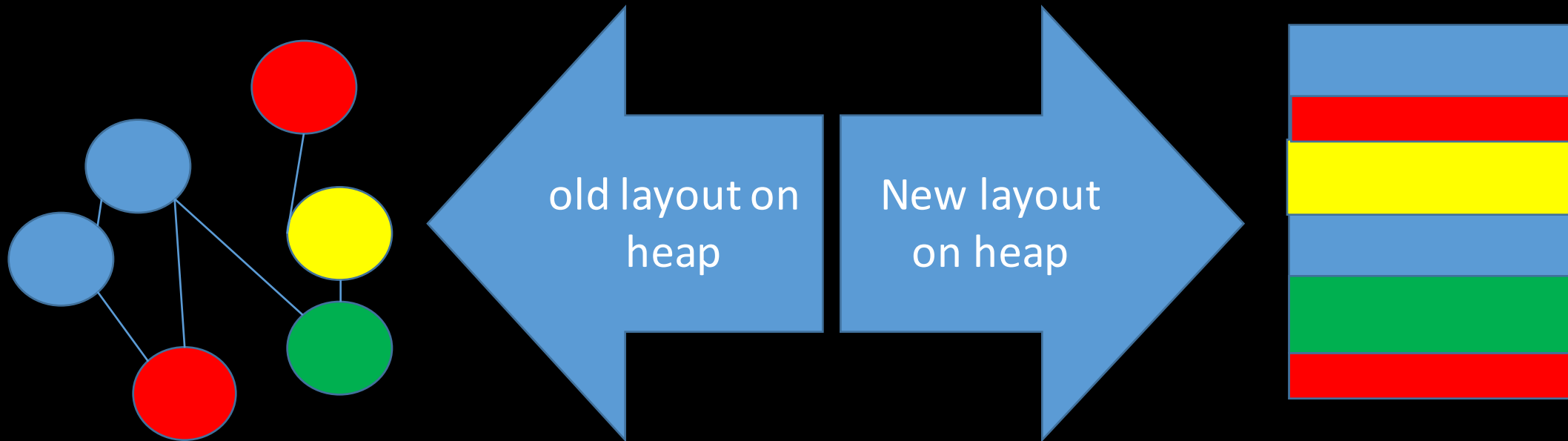
A good reason why JVM developers are working on improved native code interop

<http://openjdk.java.net/projects/panama/>

And things like value types <http://openjdk.java.net/jeps/169>



Effective native interop requires the JVM to be able to pass in data in the form that the native code needs. And do it very fast (bye bye JNI).

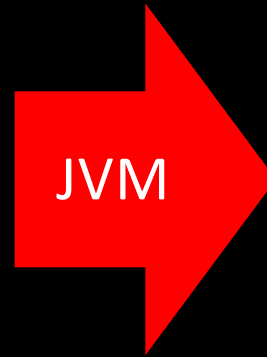


Here's where everything changes.

- Up to now the JVM has done all the heavy lifting in transforming your model of an application into the real thing.



Your model



Multiple real applications

From now on – you and your application will have to change as well

GPUs require you to think differently.

So will AI and Machine Learning..

Neural nets are number crunchers – something GPU's are fantastically good at.

More reasons for Value Types and native interop



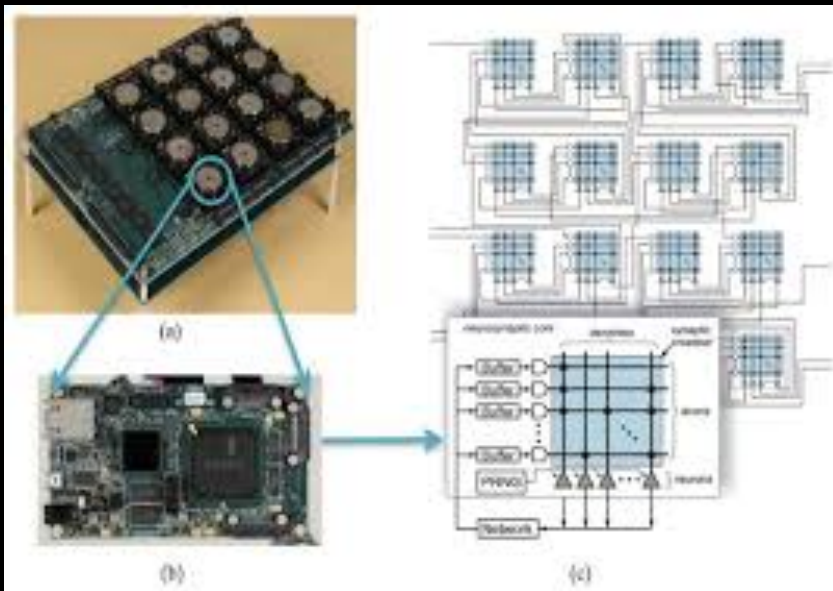
What comes after GPUs?

neuromorphic processors of course

Synapse:

A program to develop a neuromorphic processor that is a new kind of cognitive computer

Designed to simulate the neurones and dendrites of the brain for low power efficient operation



TrueNorth chips have a million computer 'neurons' that work in parallel across 256 million inter-neuron connections

Different from a standard chip

Traditional chips run all of the time.

This new neurosynaptic chip is event-driven and **operates only when it needs to**, resulting in a cooler operating environment and lower energy use.

The neurosynaptic chip veers from the traditional von Neumann architecture, which inherently creates a bottleneck limiting performance of the system.



Traditional computers focus on language and analytical thinking

(Left brain)

Neurosynaptic chips address the senses and pattern recognition

(Right brain)



Over the coming years, IBM scientists hope to meld the two capabilities together to create a **holistic computing intelligence**

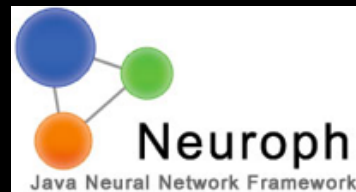
Event driven, Non Von Neumann Neural Network.

Neural Nets want their data in different forms

They behave differently

You'll have to think differently

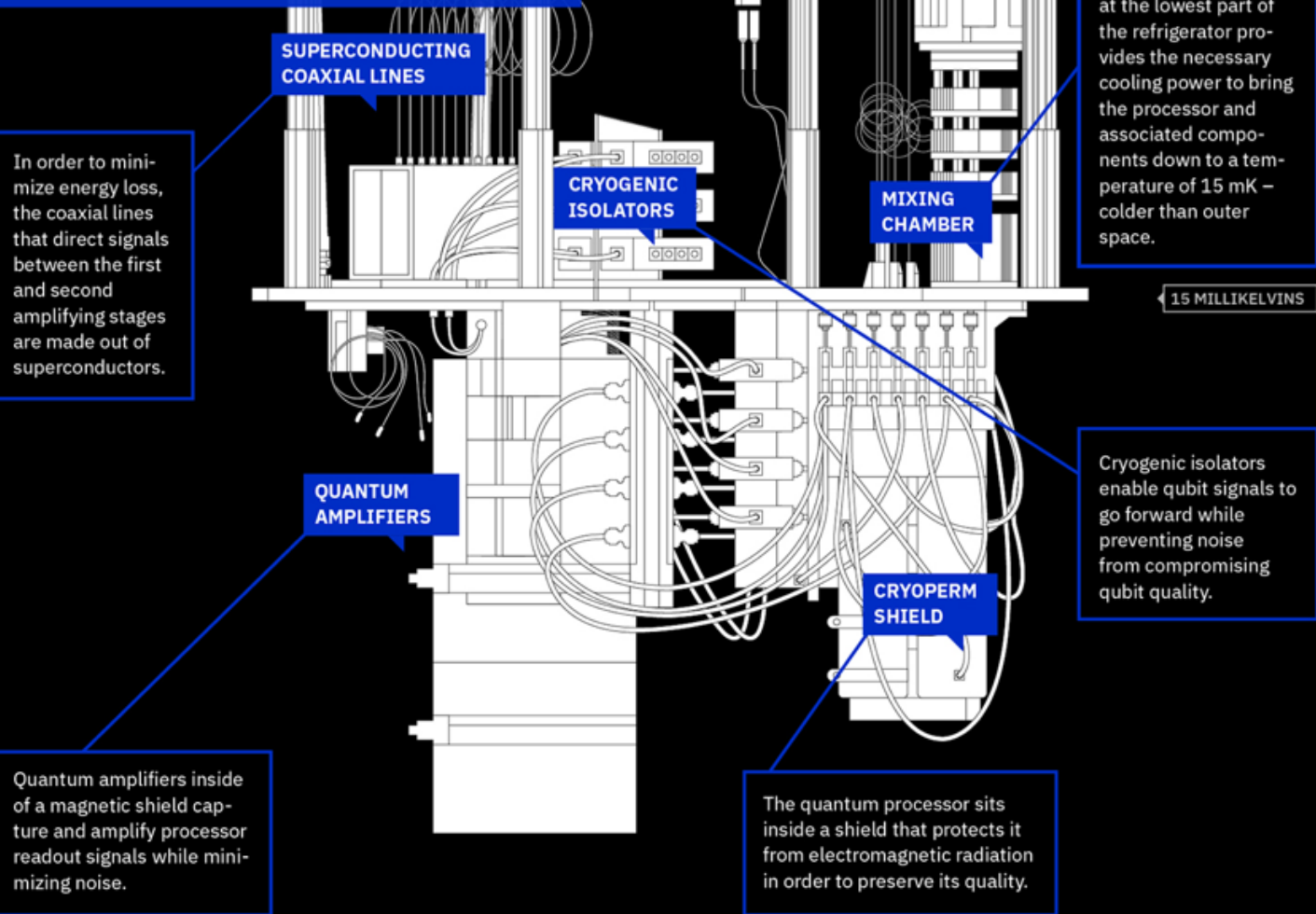
Your applications are going to change



And beyond?

Inside Look: Quantum Computer

Harnessing the power of a quantum processor requires maintaining constant temperatures near absolute zero. Here's a look at how a dilution refrigerator, made from more than 2,000 components, exploits the mixing properties of two helium isotopes to create such an environment.



In order to minimize energy loss, the coaxial lines that direct signals between the first and second amplifying stages are made out of superconductors.

The mixing chamber at the lowest part of the refrigerator provides the necessary cooling power to bring the processor and associated components down to a temperature of 15 mK – colder than outer space.

Cryogenic isolators enable qubit signals to go forward while preventing noise from compromising qubit quality.

Quantum amplifiers inside of a magnetic shield capture and amplify processor readout signals while minimizing noise.

The quantum processor sits inside a shield that protects it from electromagnetic radiation in order to preserve its quality.

Wrap up

Machine Learning works because (mostly) humans design and teach it

A ML solution can be a significant investment.
It is also a significant business opportunity

That has implications...

- 1 Lots of “BlackBox” services and APIs that allow the owner to keep their “secret source” secret
- 2 Investment in the solution is sticky - it’s hard to create so moving to a new runtime is less likely
- 3 Developers will spend more time consuming services into their own applications and less on bespoke solutions
- 4 Developers will be creating new forms of applications around data analytics and machine learning

Which means

- 1 JVMs need to be able to offer good machine learning and data analysis capabilities
- 2 Java Developers will spend more time in transforming data and thinking about problems in data terms
- 3 Java Developers will be learning new technologies and tools

The immediate challenge for Java and the JVM



Is to innovate faster – or loose to the competition

Now you can see why our ecosystem is repositioning for a faster pace

predictable
consistent cadence

easier migration

increased innovation

Allowing everyone to participate (and offering multiple starting points)

Every development team has both common and unique problems to solve.

Open source is key to fast innovation and adoption

OpenJDK
Eclipse OpenJ9
Open Liberty
Eclipse MicroProfile
Java EE
IBM Cloud
Docker
Kubernetes

Giving Java innovation a faster cadence

lambda
streams

modules
reactive streams

panama
valhalla
penrose
amber

And a variety of implementations to choose from

Tomcat
Glassfish
Open Liberty
...

OpenJDK + Hotspot
OpenJDK + OpenJ9

J2EE
Micro-profile

Generating New JVM technologies

Container
awareness
Fast native code
interop

Re-optimised JITs
Re-tuned GC's
AOT
Value Objects

Jit As A Service
<your idea here>

Java Vendor competition *and* collaboration delivered

- The **fastest** runtime environments
- The **most scalable** runtime environments
- The **best** garbage collectors
- The **greatest** dynamically re-optimizing compilers

- And we're still doing it.

Java is going places it's never been before.

The JVM is rising to the challenge – are you?



@spoole167





Trademarks, notes and disclaimers

© IBM Corporation 2017

- IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at ["Copyright and trademark information"](http://www.ibm.com/legal/copytrade.shtml) at www.ibm.com/legal/copytrade.shtml.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
- Other company, product, and service names may be trademarks or service marks of others.
- References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.