IBM

JVM Support for Multitenant Applications

Improving Application Density

Important Disclaimers

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.

WHILST EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

ALL PERFORMANCE DATA INCLUDED IN THIS PRESENTATION HAVE BEEN GATHERED IN A CONTROLLED ENVIRONMENT. YOUR OWN TEST RESULTS MAY VARY BASED ON HARDWARE, SOFTWARE OR INFRASTRUCTURE DIFFERENCES.

ALL DATA INCLUDED IN THIS PRESENTATION ARE MEANT TO BE USED ONLY AS A GUIDE.

IN ADDITION, THE INFORMATION CONTAINED IN THIS PRESENTATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM, WITHOUT NOTICE.

IBM AND ITS AFFILIATED COMPANIES SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.

NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:

- CREATING ANY WARRANT OR REPRESENTATION FROM IBM, ITS AFFILIATED COMPANIES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS

About Me

Steve Poole

Works at IBM's Hursley Laboratory in the UK

Involved in IBM Java VM development since before Java was 1

Currently leading IBM's OpenJDK technical engagement

contact spoole at uk.ibm.com

What should you get from this talk?

- This talk describes a new feature the IBM Java team is working on
  - Google ‘IBM Java 8 beta’ for more information
- By the end of this session, you should be able to:
  - Understand what multitenancy is and what it’s good for
  - Describe the challenges of multitenant Java deployments
  - Understand ideas for new JDK features to convert existing applications into multitenant deployments

Agenda

1. **Don't Repeat Yourself:** Simplify to save time and money
2. **Climbing Mt. Tenant:** Challenges and a route to the top
3. **Neighbourhood Watch:** Dealing with bad behaviour
4. **Risk vs. Reward:** How dense can we go?
5. **Wrap-up:** Summary, and Next steps

Who owns one of these?

Who owns one of these?

Simplifying the software stack by removing all extraneous pieces makes better use of hardware (and the people who run it).

Simple == Cheaper == Predictable == Robust

Don't Repeat Yourself: Simplify to save time & $$$

"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system"

Pragmatic Programmer (Hunt & Thomas)

(or: copy-and-paste encourages problems)

http://www.instructables.com/id/How-To-Create-A-LEGO-Star-Wars-Clone-Army/

Multitenancy == Simplification

- Multitenancy refers to a principle in software architecture where a single instance of the software runs on a server, serving multiple client organizations (tenants).

  Multitenancy is contrasted with a multi-instance architecture where separate software instances (or hardware systems) are set up for different client organizations.

  With a multitenant architecture, a software application is designed to virtually partition its data and configuration, and each client organization works with a customized virtual application instance.

Thanks to WIKIPEDIA

SaaS Opportunity == Efficiency is $$$

- 

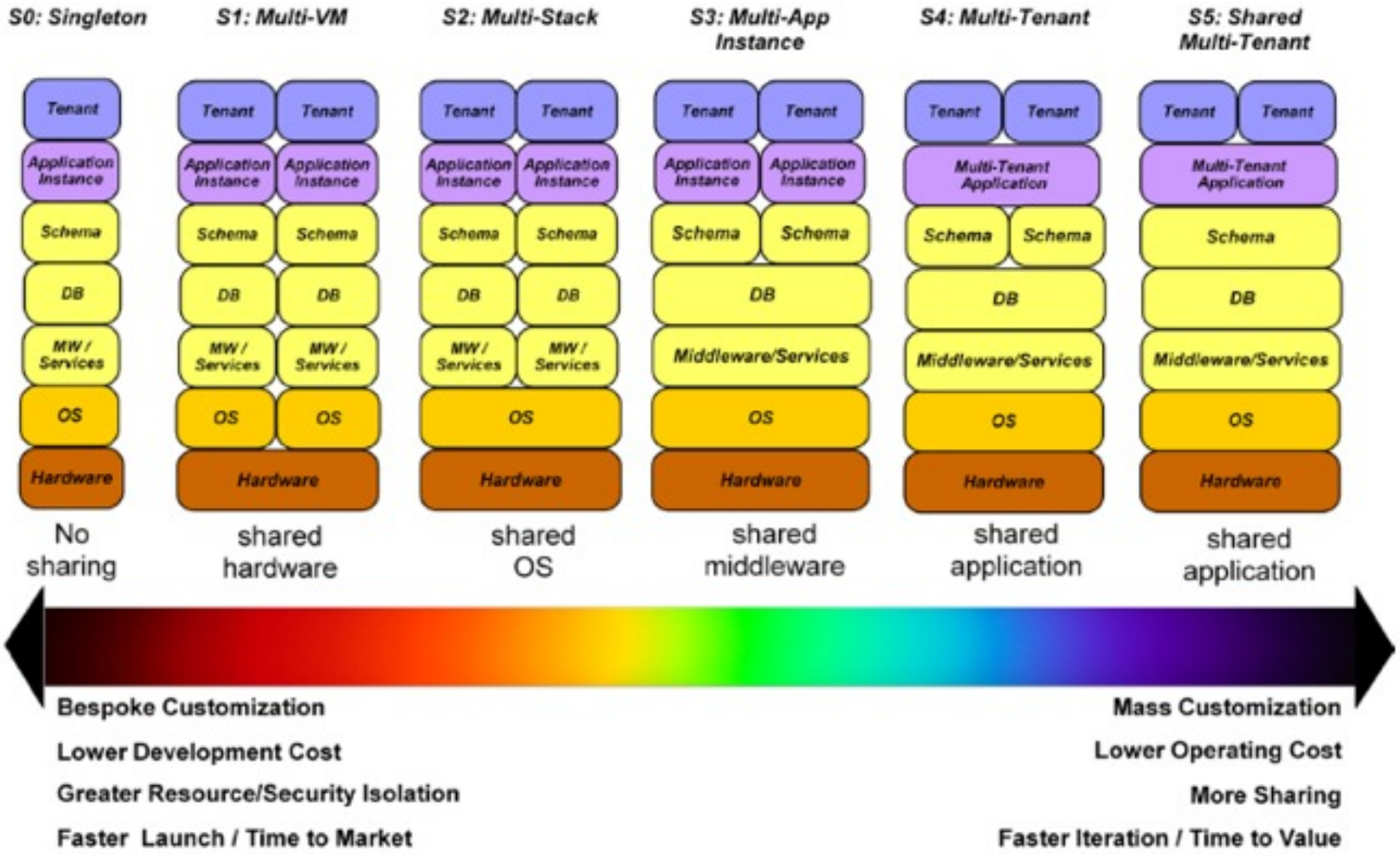
Sales, Service, Social Marketing
- 

Marketing Optimization
- 

Wiki, Bug Trackers, SCM, Build
- 

Application Performance Monitoring

SaaS Tenancy Spectrum

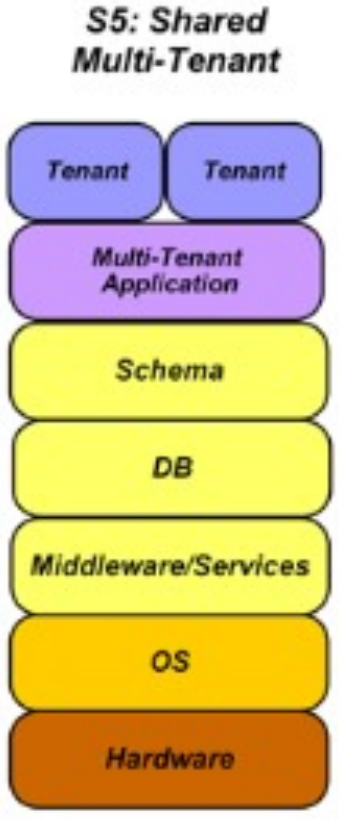


source: Peter Cousins & Jim Colson whitepaper

Efficiencies of Multitenancy

- **Customer viewpoint**
  - Cost: provider runs the service
  - Time to Value: up and running fast, typically upgraded often & quickly
  - Quality of Service: focus on SLA needed not your ability to run infrastructure
  - Bypass IT Backlog: streamlined deployment (handled by provider)

- **Provider viewpoint**
  - Cost: Minimal moving parts / duplication
  - Agility: Upgrades, backups, on-boarding



Climbing Mt. Tenant

Challenges and one* relatively easy route to the top

* of many

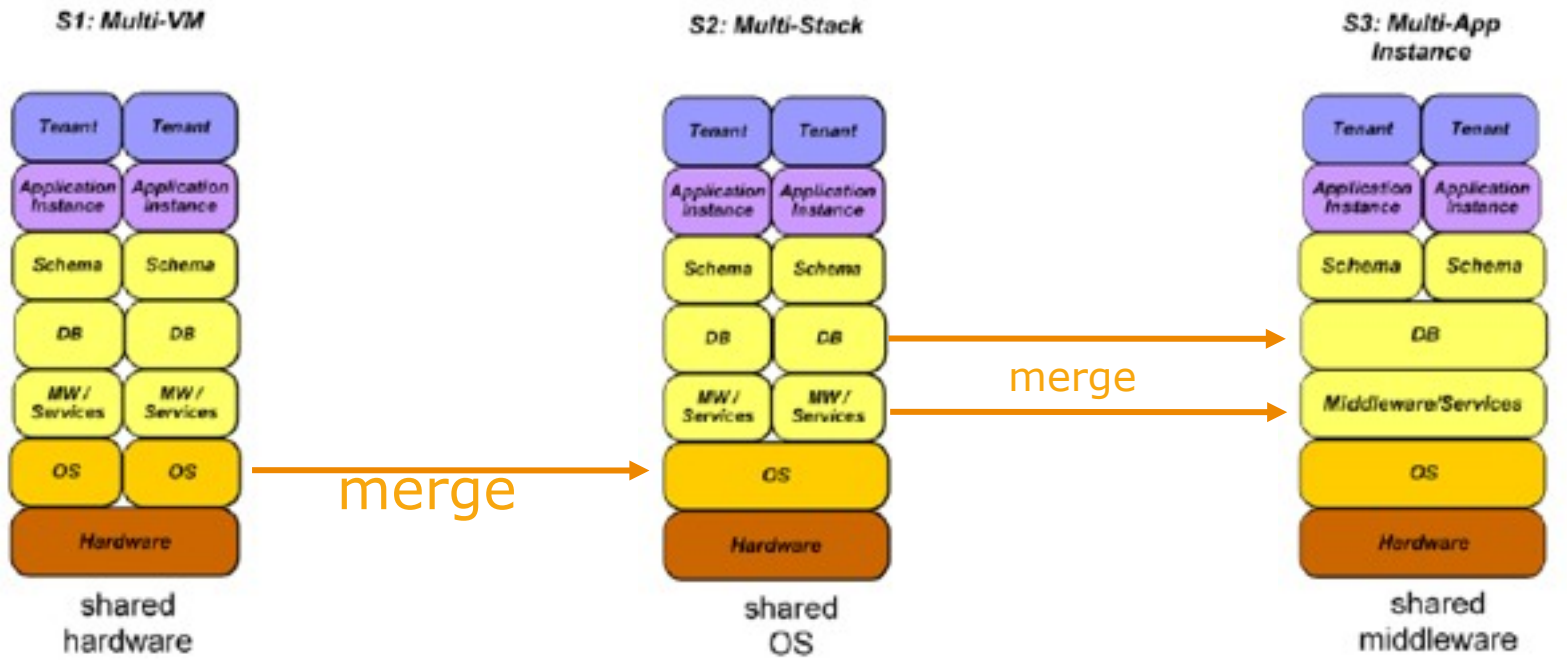
Multitenancy Challenge #1: Isolation

- Same number of eggs (apps), fewer baskets
- You want really good baskets arranged carefully
- Not a new problem

http://circa71.wordpress.com/2010/07/

http://bit.ly/e7G1jb

Multitenancy Challenge #2: Cost of Entry



- ☺ Easy == No app changes
- ☺ Hypervisor sharing only

- ☺ Port Collisions
- ☺ File System Collisions
- ☺ Security Challenges

- ☺ Data Isolation between apps
- ☺ Control over resource hogs
- ☺ JVM can help!!

Cost of Dedicated Middleware (JVM-centric)

(Opportunities for improvement)

Java Heap consumes 100's of MB of memory
- –Heap objects cannot be shared between JVMs
- –GC has a helper thread-per-core by default

Just-in-Time Compiler consumes 10's of MB of memory
- –Generated code is private and big
- –Generated code is expensive to produce
  - Steals time from application
  - Multiple compilation threads by default

No choreography between JVM instances
- –Compilation or GC activity can happen at identical (and bad) times

Challenge: Lower Cost-of-Entry

We need to fix the following

- ☹ Data Isolation between applications
- ☹ Control over resource hogs

Without forcing people to change their applications!

Data Isolation Challenges: Example #1

- Applications embed deployment information like url patterns in code

```
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
...

@WebServlet(name="ServletAddPost", urlPatterns="/add/post")
public class ServletAddPost extends HttpServlet {
  ...
}
```

- Wait! What happens if we try to deploy two copies of this servlet to a single server?

Data Isolation Challenges: Example #2

- Static variables are bad (for sharing)
- Most libraries are full of static variables

LocaleSettings.java

```
1  package javaone;
2
3  import java.util.Locale;
4
5  public final class LocaleSettings {
6
7      public static final Locale CANADA = new Locale("en", "CA");
8      public static final Locale UK = new Locale("en", "GB");
9      public static final Locale USA = new Locale("en", "US");
10
11     private static Locale defaultLocale = CANADA;
12
13     public static void setDefaultLocale(Locale defaultLocale) {
16
17     public static Locale getDefaultLocale() {
```

Wait! What happens if each tenant needs a different default locale?

Inspiration from the Past: VisualAge for Java

- VisualAge for Java was an IDE and Runtime from late 90's early 2000's
- Built in Smalltalk with a VM that understood both Smalltalk and Java bytecodes
- Multiple copies of a program could run in parallel sharing everything
  - –Tiny footprint
  - –Blazing startup
- Also: Sun Project Barcelona

Hardware Requirements

VisualAge for Java, Professional Edition Version 4.0

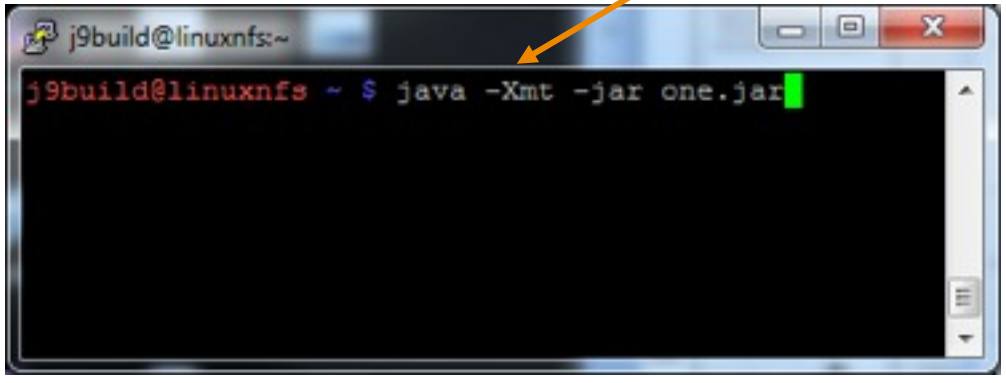
- Processor — Intel Pentium™ II, or faster, compatible processor recommended
- Display — SVGA, 800 x 600 (1024 x 768 recommended)
- CD-ROM drive
- Mouse or pointing device
- Memory requirements
  - 128 MB RAM minimum
  - 256 MB recommended
- For distributed debugger
  - 128 MB RAM minimum
  - 196 MB recommended
- Disk space requirements (based on NTFS, actual disk space on FAT depends on HDD size and partitioning)
  - 350 MB minimum
  - 400 MB or more recommended

Multitenant JDK: Easy isolation and control

- Concept: Add a single argument (`-Xmt` for multi-tenant) to your Java command-line to opt into sharing a runtime with others.
- Result: Your application behaves exactly as it if had a dedicated JVM, but in reality it runs side-by-side with other applications.
- Benefits: Smaller, faster, and eventually smarter
  - Less duplication: (1 GC, 1 JIT), Heap object sharing
  - Fast Startup: JVM is already running and warm when starting apps

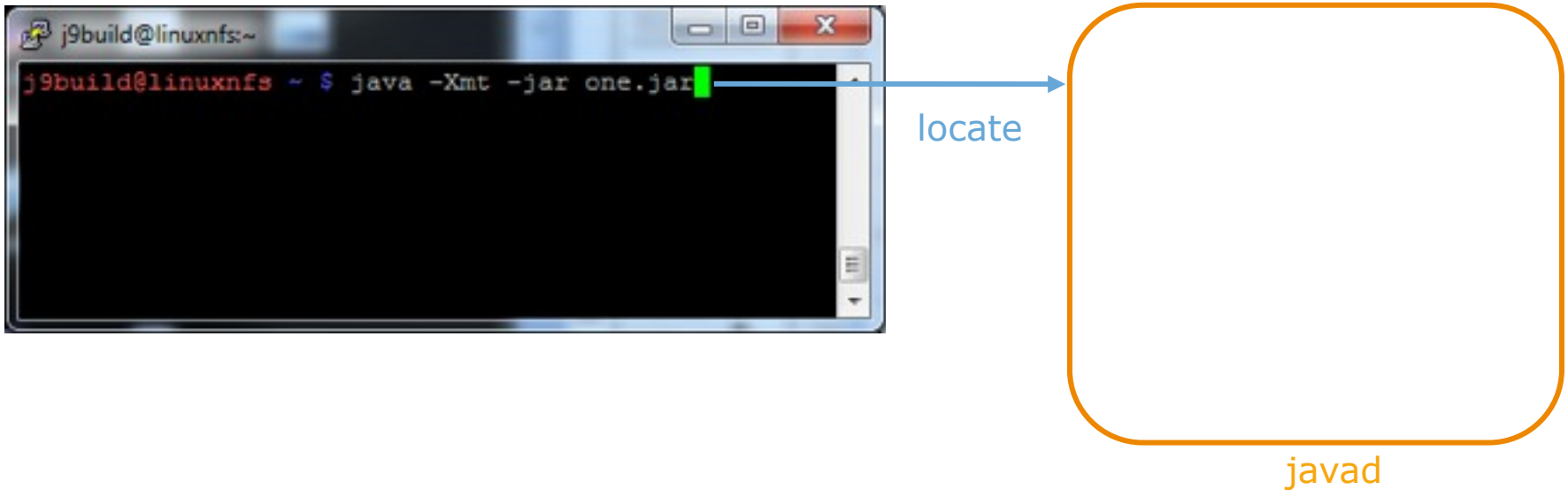
Multitenant JDK: Launch your application

- Opt-in to multitenancy by adding **`-Xmt`**



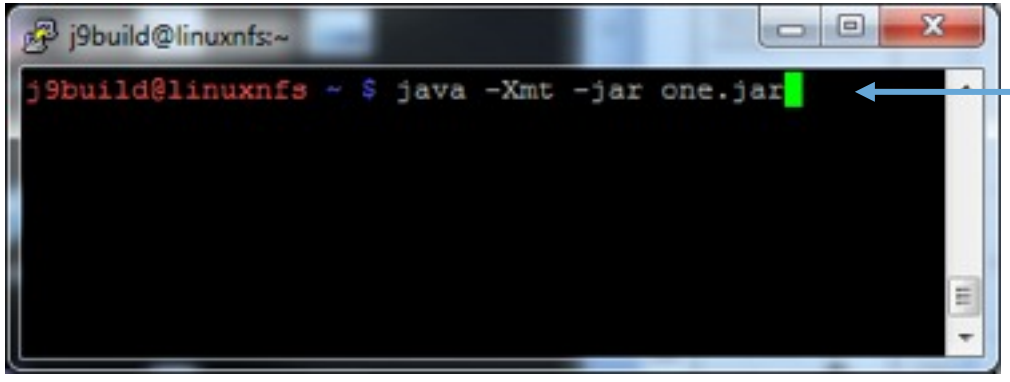
Multitenant JDK: Register with `javad` daemon

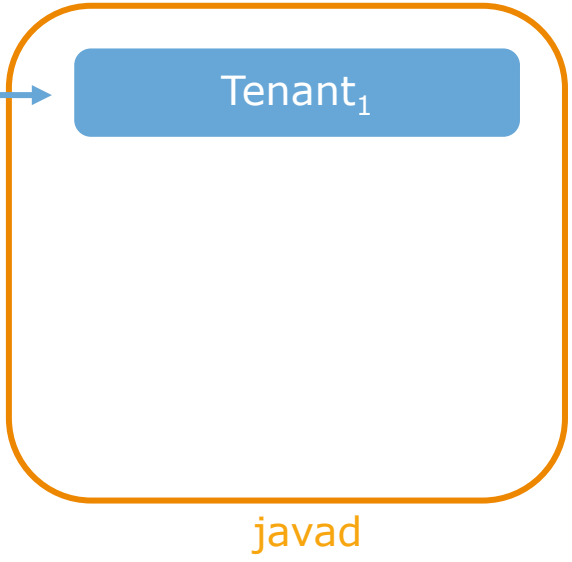
- JVM will locate/start daemon automatically



Multitenant JDK: Create a new tenant

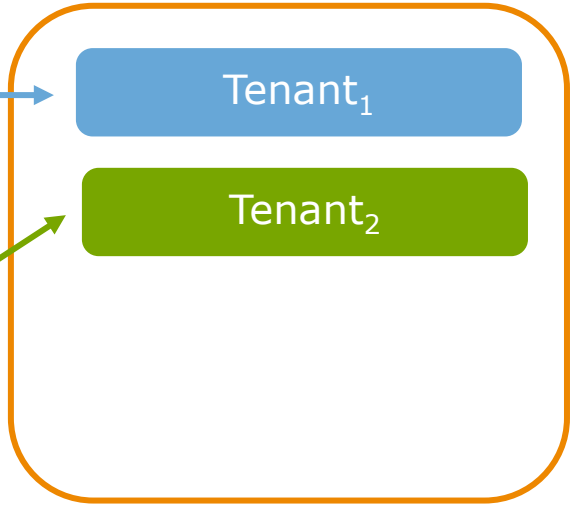
- New tenant created inside the **`javad`** daemon





Multitenant JDK: Create a second tenant

- New tenant created inside the **`javad`** daemon



One copy of common code lives in the javad process.

Most runtime structures are shared.

Solving the Data Isolation Challenge

- What if … the JVM knew about tenants and provided each one with a different view of static variables?
- Meet the `@TenantScope` annotation.

LocaleSettings.java

```
1  package javaone;
2
3  import java.util.Locale;
4  import com.ibm.tenant.TenantScope;
5
6  public final class LocaleSettings {
7
8      public static final Locale CANADA = new Locale("en", "CA");
9      public static final Locale UK = new Locale("en", "GB");
10     public static final Locale USA = new Locale("en", "US");
11
12     private static @TenantScope Locale defaultLocale = CANADA;
13
14     public static void setDefaultLocale(Locale defaultLocale) {
17
18     public static Locale getDefaultLocale() {
```

Tenant1

```
…
  LocaleSettings.setDefaultLocale(
  LocaleSettings.UK );
…
```

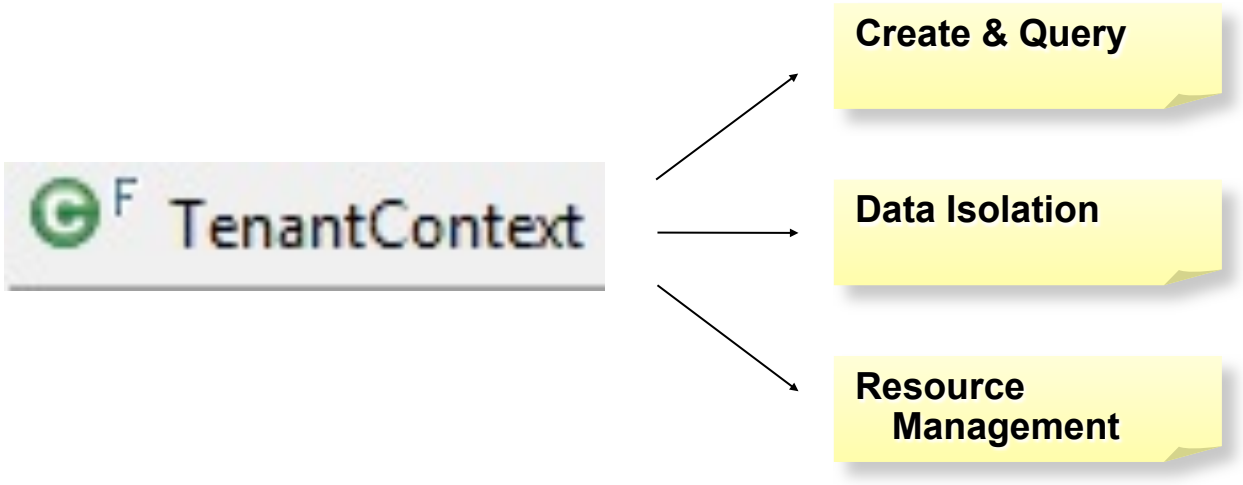
Tenant2

```
…
  LocaleSettings.setDefaultLocale(
  LocaleSettings.USA );
…
```

- `@TenantScope` Semantics: Static variable values are stored per-tenant
  - Trying to limit cost of extra indirection to single-digit throughput with JIT help
- Each tenant has their own `LocaleSettings.defaultLocale`
- Now many tenants can share a single `LocaleSettings class`

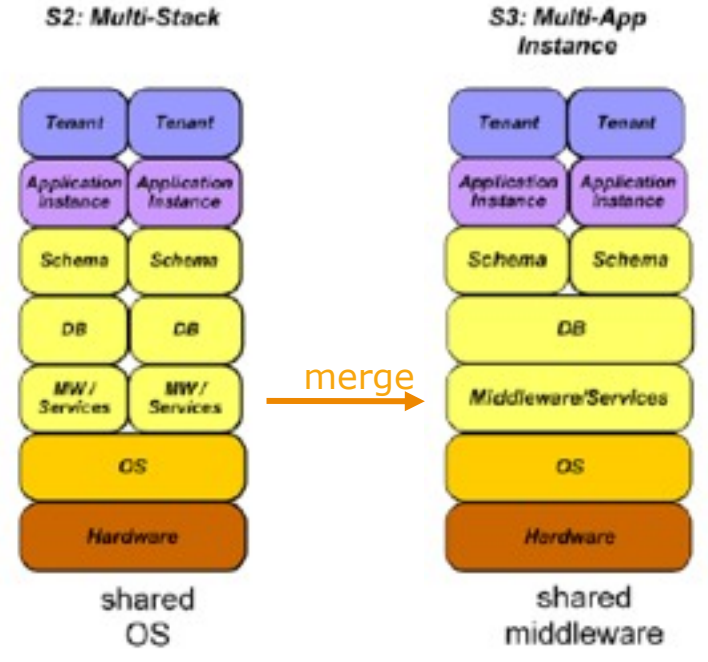
… and let's provide some API to manage Tenants: `TenantContext.class`

- Basic operations on Tenants available to the middleware
  - Data Isolation
  - Resource Management (more in this in a minute)
- Ability for the middleware to differentiate between Tenants
  - Which one is causing the problem?
- Querying the state of Tenants
  - How much free memory do you have?

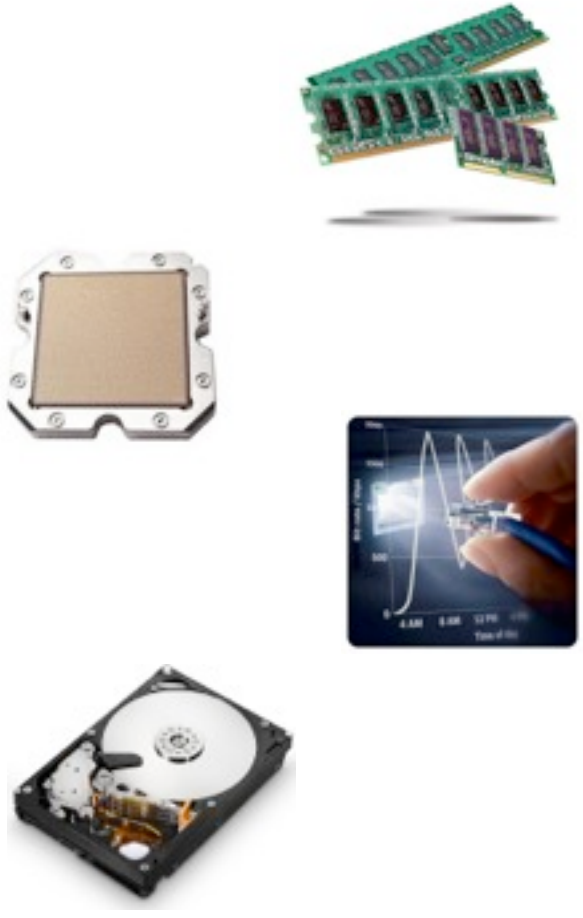


Multitenant JDK: Shared-JVMs that 'feel' dedicated

- `@TenantScope` markup gets added automatically as classes are loaded
- Tenants see dedicated middleware – but behind the curtains classes (and JIT'ed code) are actually shared



Neighbourhood Watch: Dealing with bad behaviour



http://bit.ly/ficwkl

images from http://www.rra.memberlodge.org/Neighbourhood-Watch-Reporting
http://mcsholding.com/DetailsPage.aspx?Page_Id=42

Shared Environments need Resource Control

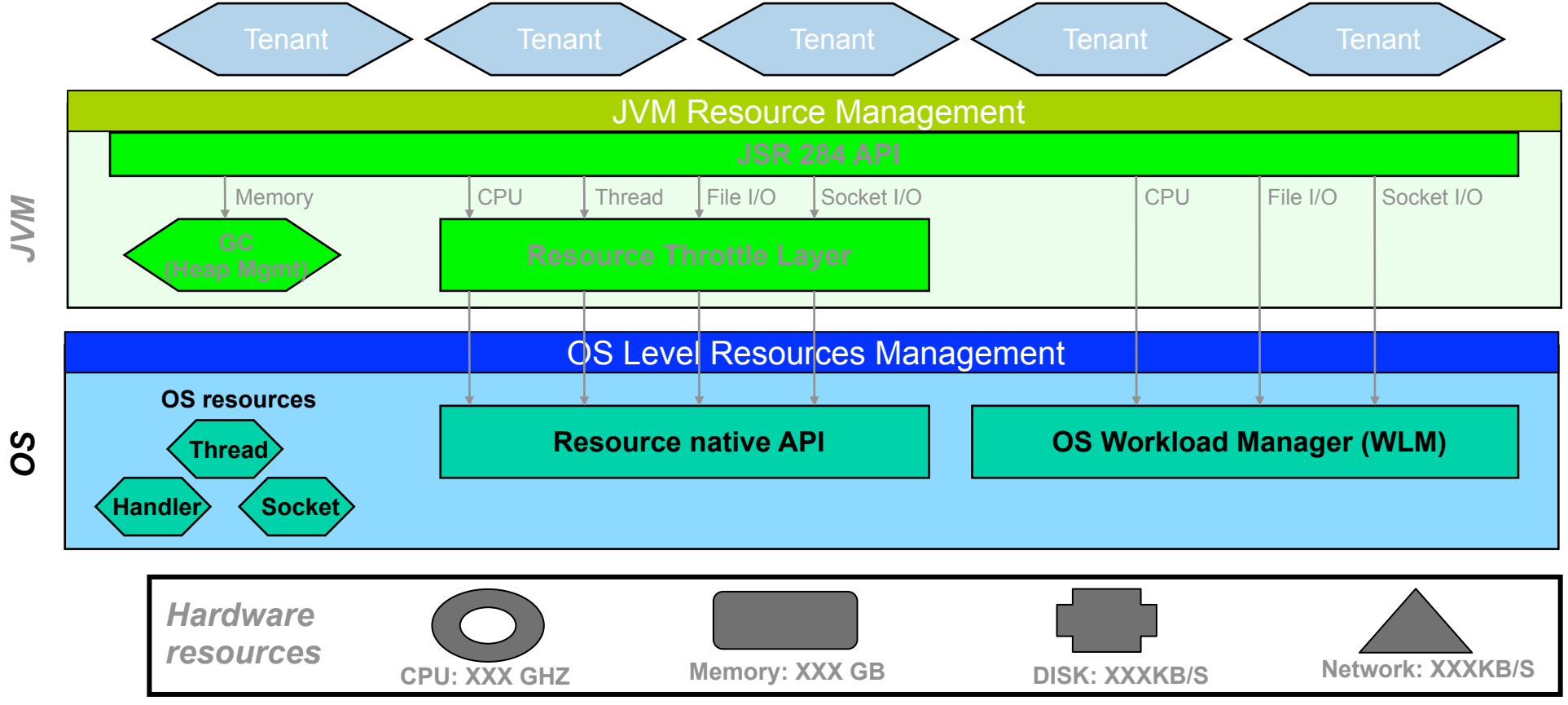
- **The closer your neighbours the better your controls must be**
- **Multitenant JDK provides controls on**
  - CPU time
  - Heap size
  - Thread count
  - File IO: read b/w, write b/w
  - Socket IO: read b/w, write b/w

Resource Control Ergonomics

- **Simple command-line switches for new resources**
  - `-Xlimit:cpu=10-30` `// 10% minimum CPU, 30% max`
  - `-Xlimit:cpu=30` `// 30% max CPU`
  - `-Xlimit:netIO=20M` `// Max bandwidth of 20 Mbps`
- **Existing options get mapped for free**
  - `-Xms8m -Xmx64m` `// Initial 8M heap, 64M max`
- Plus some JMX beans to see how much of each resource you are using
  - i.e. understand how your code uses resources by wrapping in a tenant

JSR-284 Resource Consumption Mgmt API

- **Throttling at Java layer for portability**
- **Or, leveraging OS WLM directly for efficiency (Linux & AIX)**
  - Note: many WLMs tend to like processes, not groups of threads



JVM vs. Operating System CPU Throttling

Benchmark setting

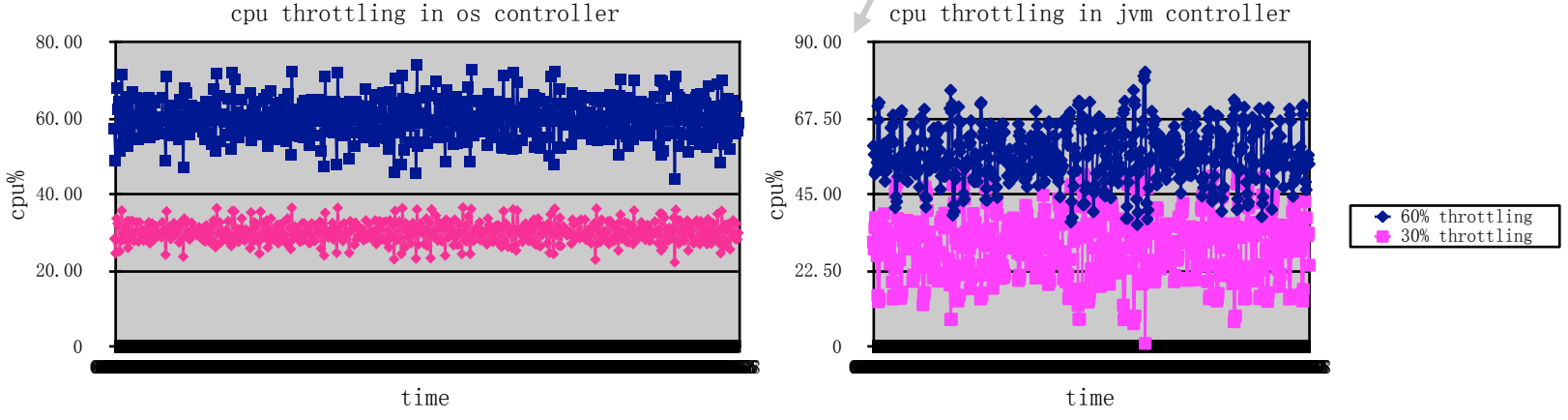
- **Duration comparison**: Linux AMD64, run a CPU-intensive app with 10 threads with 100% CPU quota, each thread doing the same Fibonacci calculation, benchmark the duration
- **Accuracy comparison**: Linux AMD64, run two CPU-intensive apps each doing the same Fibonacci calculation, but with different CPU quota: 60% vs 30%, benchmark the accuracy

Duration

Round	OS as controller	JVM as controller
1	1362s	1267s
2	1167s	1239s
3	1452s	1390s
4	1094s	1122s
5	1139s	1123s
6	1244s	1134s
Average	**1243s**	**1212s**

The shorter duration believed to be inaccurate throttling.

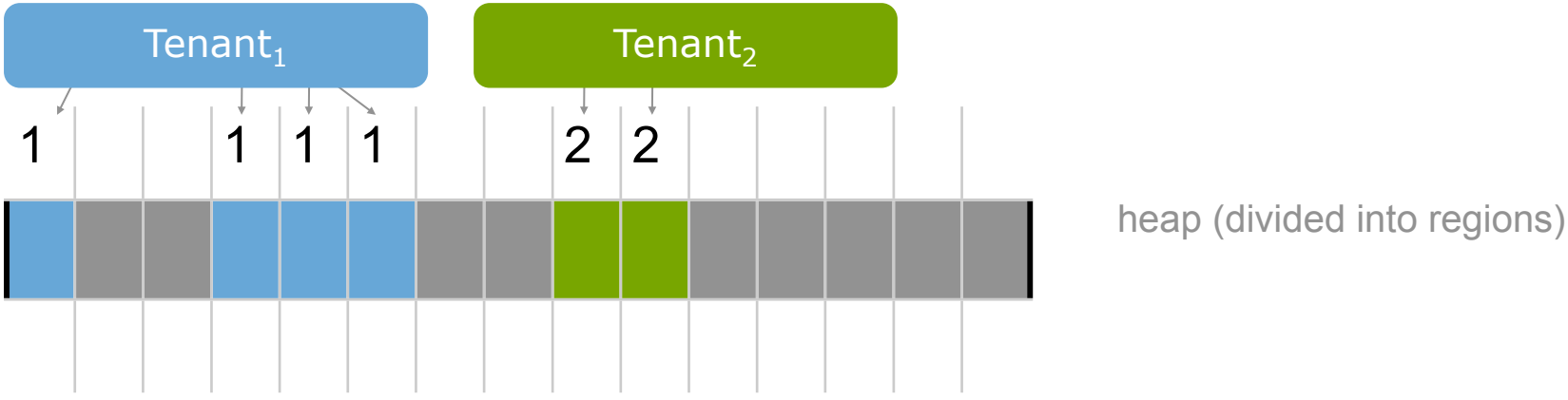
Accuracy



***Result**: JVM control achieves comparable performance, but less accuracy.*

Per-Tenant Heap Consumption

- IBM JDK's have new region-based GC technology which maps nicely to tenants (more @ http://ibm.co/JtWfXr)
- Technique:
  - Each tenant is initially given enough GC regions to satisfy its minimum reservation
  - Code running in tenant scope allocates objects in a region it owns
  - New regions can be requested up to tenant maximum reservation



- Details:
  - Finalization needs to run in the proper tenant context
  - We must be able to map from an object → tenant easily
  - GC read/write barriers provide an opportunity to control inter-tenant references

Risk vs. Reward: How dense can we go?

images from
http://www.colourbox.com/image/street-post-with-risk-st-and-reward-way-signs-image-1449085

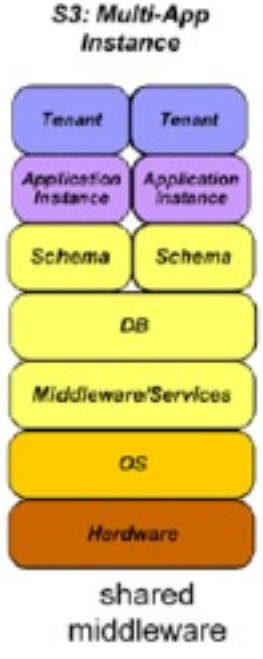
http://www.economist.com/blogs/babbage/2011/11/facebook-and-privacy

Status Today: Exploring Limits of Density

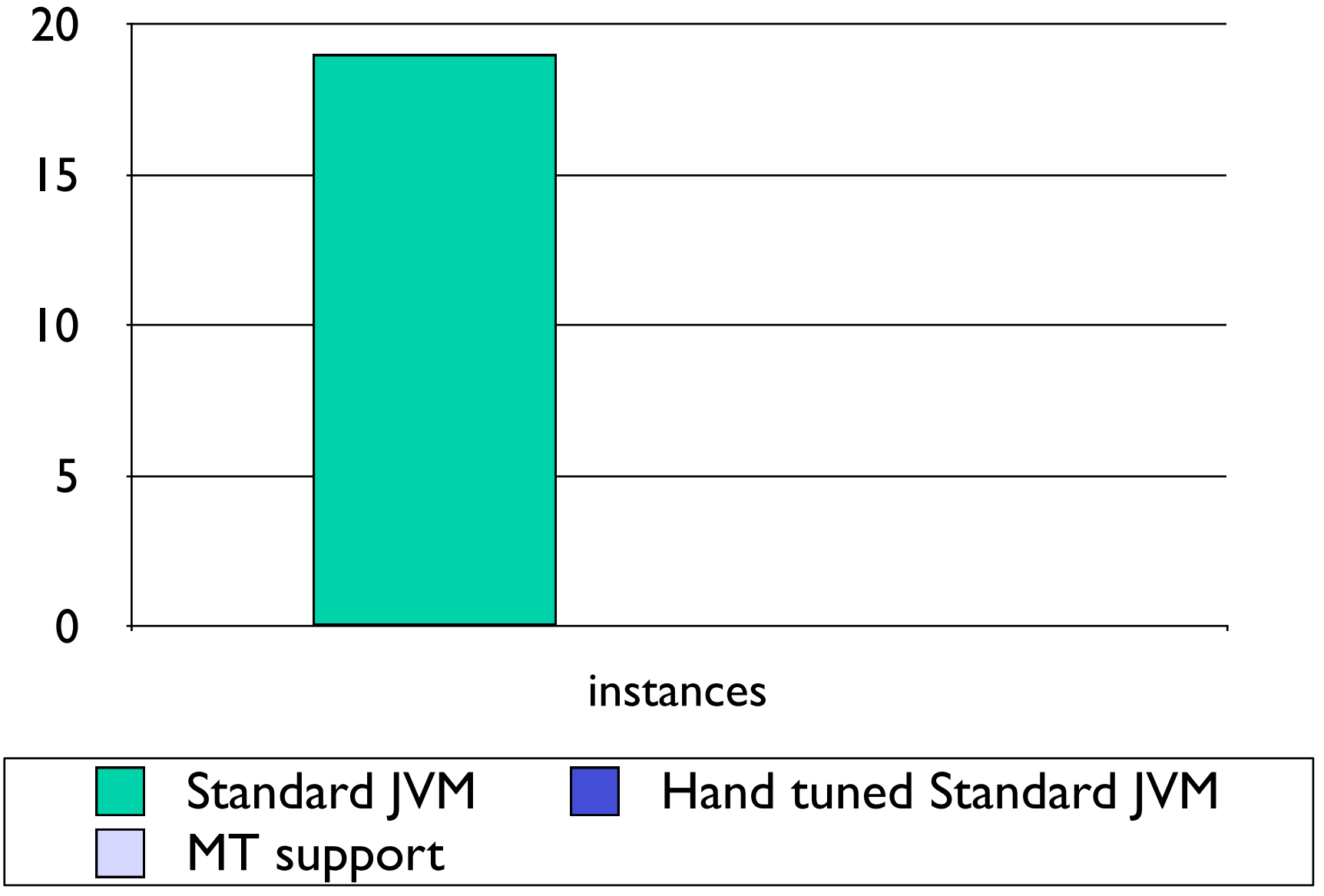
- We are still working hard on:
  - Scaling Up: Liberty-sized workloads are running today, next challenge is to up application size and tenant counts
  - Adding Safety: stronger walls between tenants, robust finalization, and detection/corrective action for ‘zombie’ tenants
  - Quota Enforcement: Evaluating stalling vs. exception throwing options
  - Performance: Measuring density, and improving throughput and some new concerns like: idle behavior, idle->busy responsiveness
  - Simplifying configuration for resource management
- Next Steps
  - We need your feedback: are we on the right track?
  - It is our intention to standardize via the Java Community Process

Current Performance Data

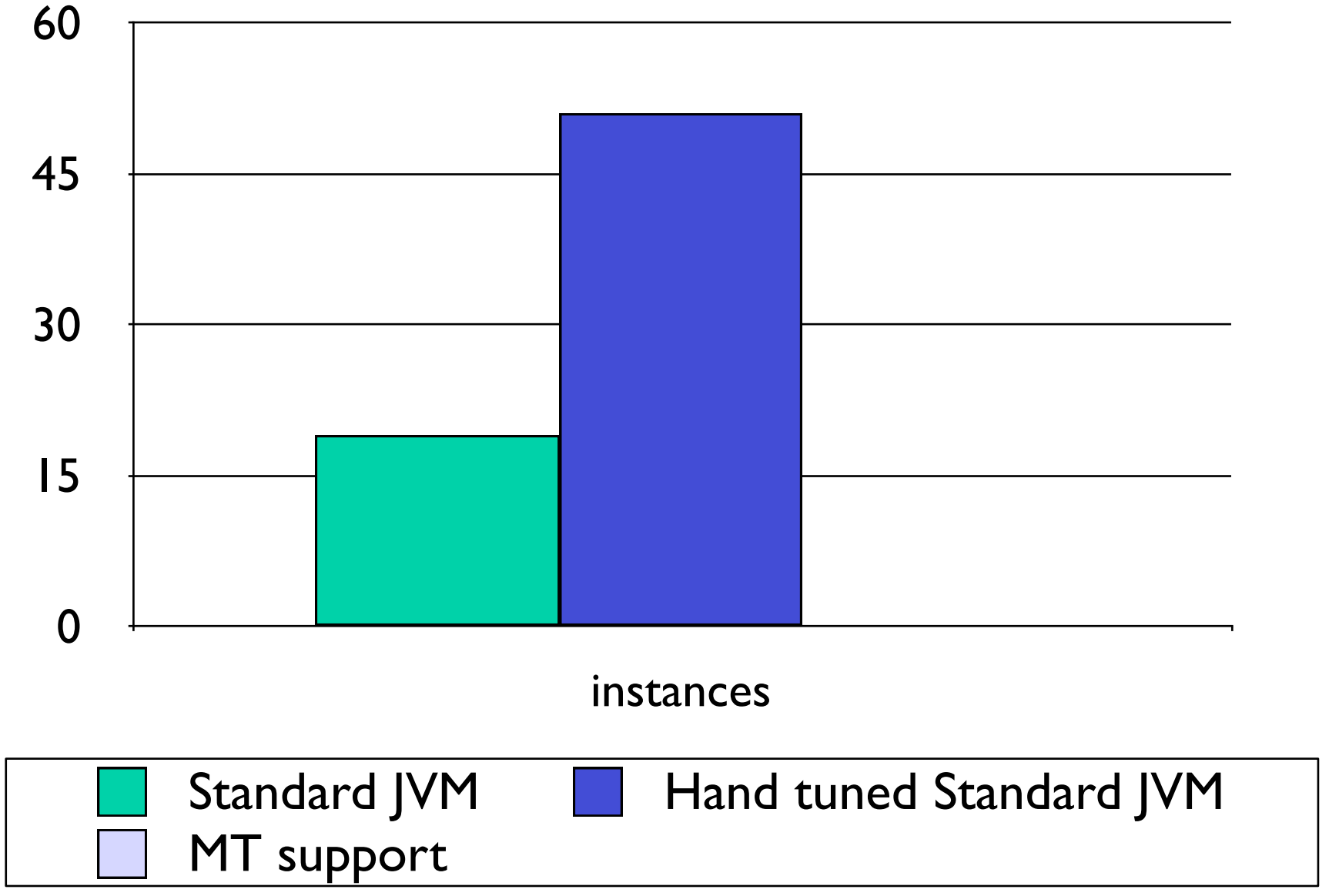
- Environment: Measure standard benchmarks in a 1 GB + 1 core VirtualBox guest
  - Advantage: Easy to control, highly reproducible
- Methodology: Add applications until the system swaps, then it's 'full'
  - More applications is better
  - Per tenant cost is amount of RAM / # tenants



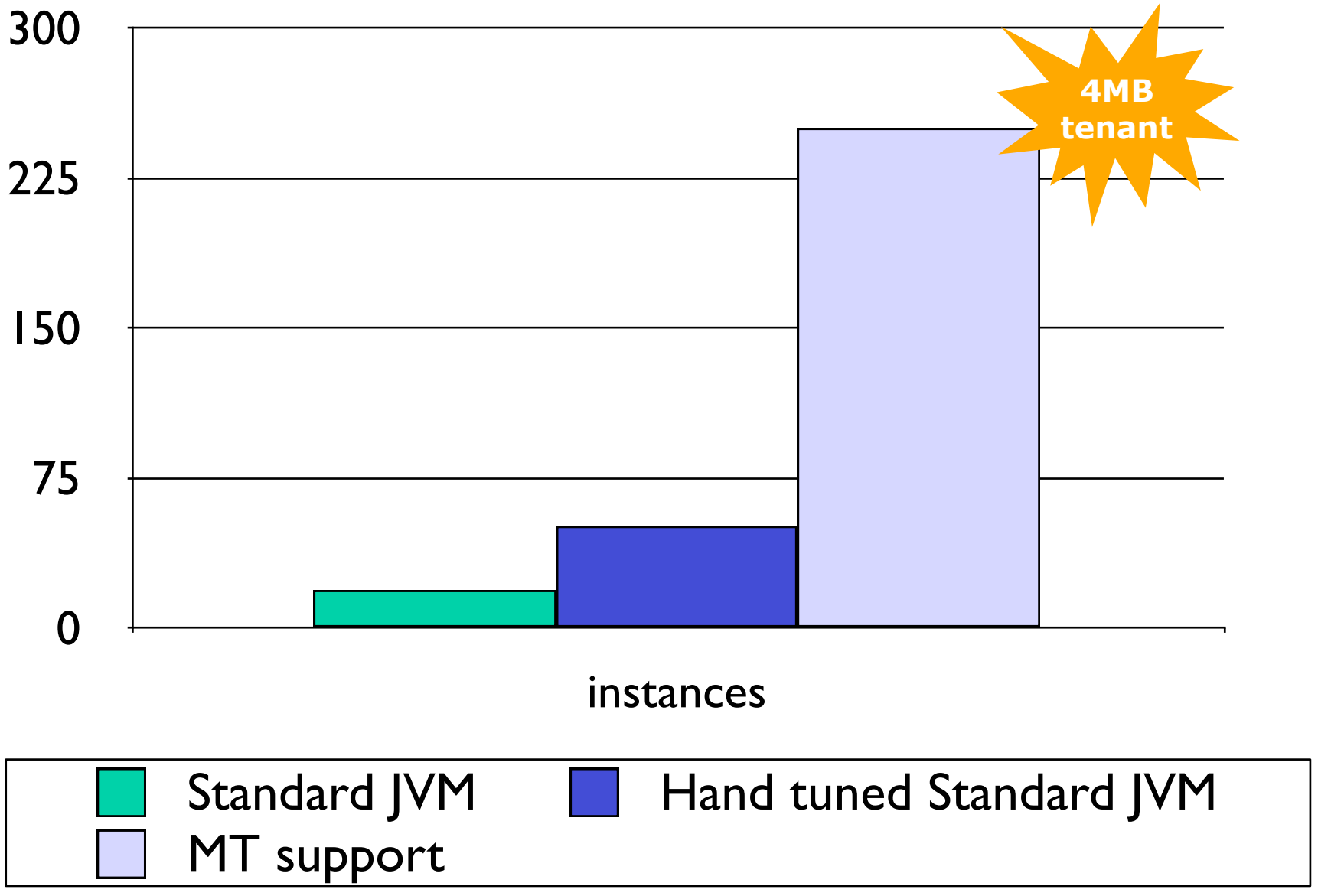
Performance results - standard JVM no tuning



Performance results- hand tuned standard JVM

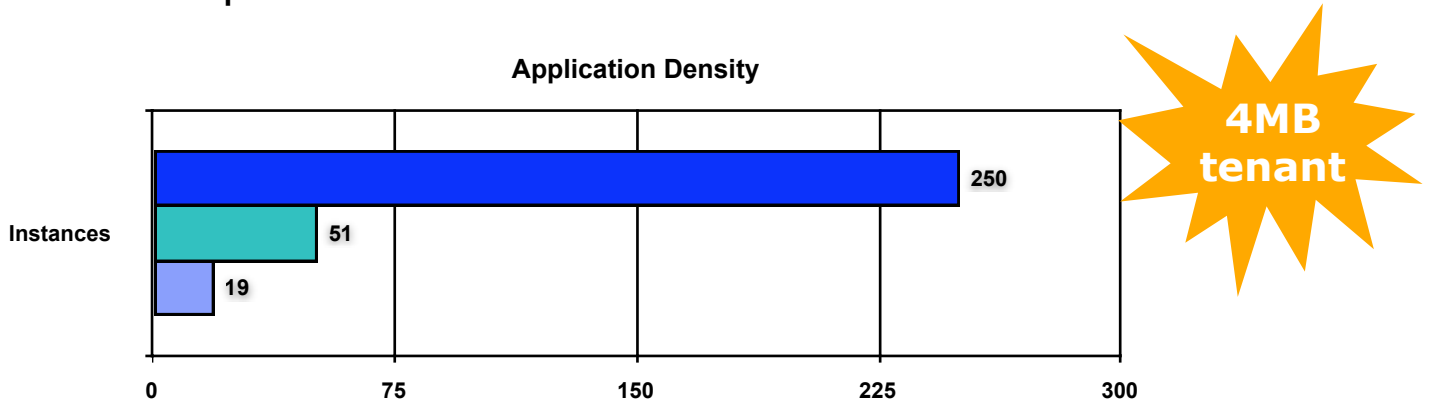


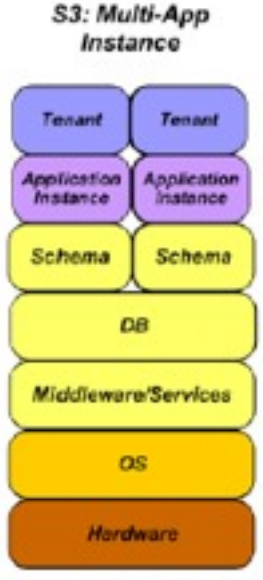
Performance results - Xmt



Current Performance Data

- Environment: Measure standard benchmarks in a 1 GB + 1 core VirtualBox guest
  - Advantage: Easy to control, highly reproducible
- Methodology: Add applications until the system swaps, then it's 'full'
  - More applications is better
  - Per tenant cost is amount of RAM / # tenants
- Results: 3x faster second-run startup, and ~5x the density
  - Still working the JIT support so no throughput #'s yet
- What's shared:
  - Boot & Ext classes and *heap objects* they create (interned Strings)
  - JIT compiled code & metadata





Thorny Technical Problems (many solved)

- Synchronization of 'Shared' Heap Objects: j.l.String and j.l.Class
  - Solution: Give each heap object a per-tenant monitor when contended
- Safe Finalization: Protecting 'shared' services in addition to data
  - Solution: Requires efficient object -> TenantContext mapping, and ability for finalizer to detect and recover from malicious finalizers (i.e. denial-of-finalization attack)
- Support for JNI Natives: allow multiple loadLibrary("foo") calls
  - Solution: run natives back in the 'launcher' process by JNI remoting
  - New challenges: communication latency and NIO direct buffers
- Killing Misbehaved Tenants: Hard!
  - Like j.l.Thread.stop() requires breaking locks on objects + stack unwind
  - Advantage: we can validate only tenant-owned objects are inconsistent
- User Class Loaders: frameworks like OSGi (Jigsaw?) are heavy consumers
  - We want to share identical classes found in different loaders (including JIT code)
  - Can reduce per-tenant footprint by an additional ~25%
- Post-Mortem Debugging: All tenants are listed in javacores, system dumps, etc
  - Must provide per-tenant view by cleansing artifacts or dumping per-tenant views

Roadmap

- **Focus to date has been ‘zero application changes’**
  - We can do even better with tenant-aware middleware
- **API’s used to provide isolation & throttling are available to stack products**
  - JSR-284 (Resource Management)
  - JSR-121 (Isolates)
  - @TenantScope fields
- **Java language (EE7?) and frameworks (EclipseLink) are evolving to have first-class multitenant support**
- Stay tuned for progress: watch the IBM Java 8 beta program

Final Thoughts: What should I be doing to my code today

- Performance Tuning: Measure performance and optimize your code to minimize time spent in GC and cycles consumed when idle.
  - Be a ‘good neighbour’ in a multitenant environment and make better use of hardware today.

- Prepare for Over-commit: Measure and understand busy/idle periods so that you know exactly how much resource is needed, and how to arrange workloads so that ‘spikes’ in activity are staggered.
  - Improve utilization by increasing application density

Conclusion

Simplifying the software stack by removing all extraneous pieces makes better use of hardware (and people who run it).

Multitenancy can make us more efficient:

- Trades isolation for footprint and agility
- JVM support makes multitenancy safer and easier
- Measuring resource usage and load patterns is critical
- Multitenant JDK primitives give us room for future growth

Review of Objectives

Now that you've completed this session, you are able to:

- **Understand what multitenancy is and what it's good for**
  - Per-tenant costs measured in single-digit MB are possible
- **Describe challenges of multitenant Java deployments**
  - Hard for VM guys, should be easy for you
  - Choreography of load / deployment is up to you
- **Understand new JDK features to convert existing applications into multitenant deployments**
  - Are we on the right track? Could you use this in your business?

Thank you

…any questions?