

Waldek Kot

JVM-level virtualization

or: **liberate** your enterprise Java applications **from the Operating System**

Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Agenda

1. Virtualization – brief overview
2. JRockit VE JVM
 - Java Service on the Hypervisor without the OS
3. Demos
4. Q&A

Virtualization – why ?

Typical problems of todays data centers

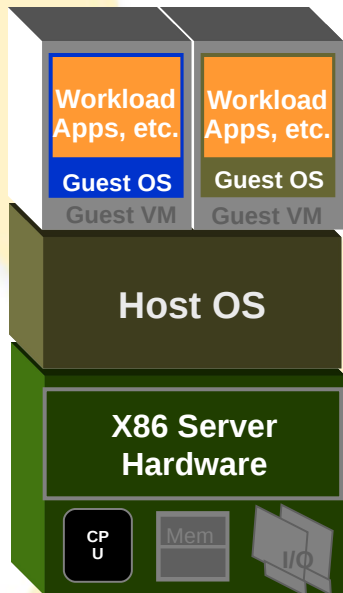
- No space
- Supply of electrical energy – more and more difficult and costly
- Cooling
- Low server utilisation
 - 5-8%
- No flexibility
 - Time to deploy a new server: often more than 3 months
 - Difficult application migration (to other hardware) – not dynamic
- High maintenance costs
 - Labour
 - Often high-end machines required
- Difficult SLA
- Rare variable allocation of IT costs

Virtualization

- Virtualization - not a new idea, but **finally** mainstream
 - x86 world, cheap computing power
- Ability to divide a physical machine into multiple logical (virtual) machines
 - Consolidation – less server machines
 - Lower costs – popular hardware platforms (Intel x86 class)
 - Resource sharing – more optimal usage
- The virtual machine is represented as a... **file**
 - Time to deploy a new server: few minutes
 - Dynamic allocation – pool of resources instead of „1 application – 1 set of servers”
 - Capacity on demand
 - **(live) migrations, HA**
 - Increased reliability and high availability
- SLA
 - Priorities, automation via policies
 - IT as a service (and other cool enterprise ideas...)
- Emerging **software deployment** model
 - Software appliance
 - no more install, configure, tune, patch, ...
 - examples: see Oracle website: edelivery.oracle.com

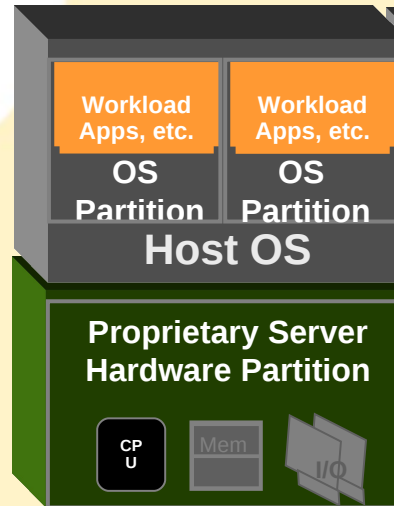
Server Virtualization Technologies

Host OS-based



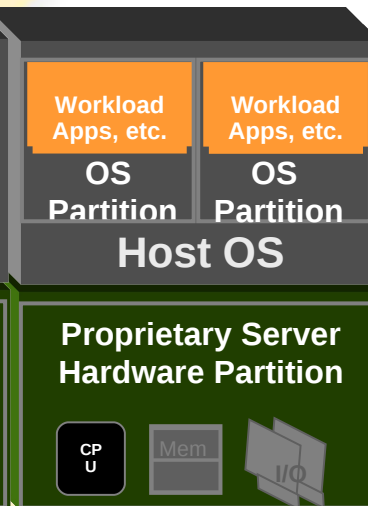
- Primarily desktop
- Easy to use
- Very slow (2 OSES)

Hardware Partitioning



- Excellent isolation
- Expensive, proprietary hw
- Coarse grain resources
- Mix OSES / versions

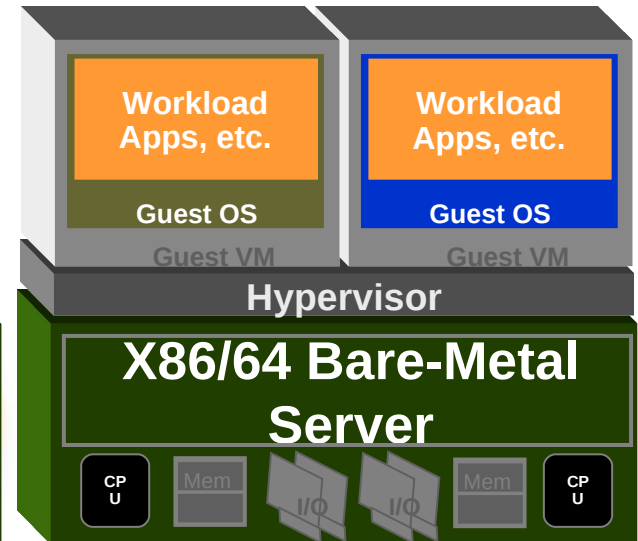
OS Partitioning



- Only moderate isolation
- Potentially good scalability
- Fine-grained resources
- Cannot mix OS/patch levels

Hypervisor-based, e.g.

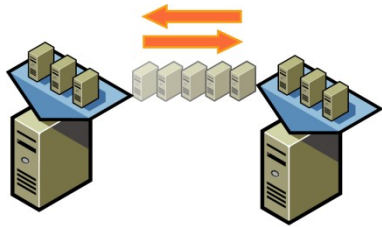
- Oracle VM
- VMware ESX Server
- Citrix XenServer
- Windows Hyper-V



- Excellent isolation
- Affordable, multi-source HW
- Fine-grained resources
- Mix OSES / versions

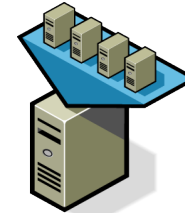
Key Properties of Virtualization

Hardware Independence



Run a virtual machine on any server without modification

Partitioning



Run multiple virtual machines simultaneously on a single physical server

Isolation



Each virtual machine is isolated from other virtual machines on the same server

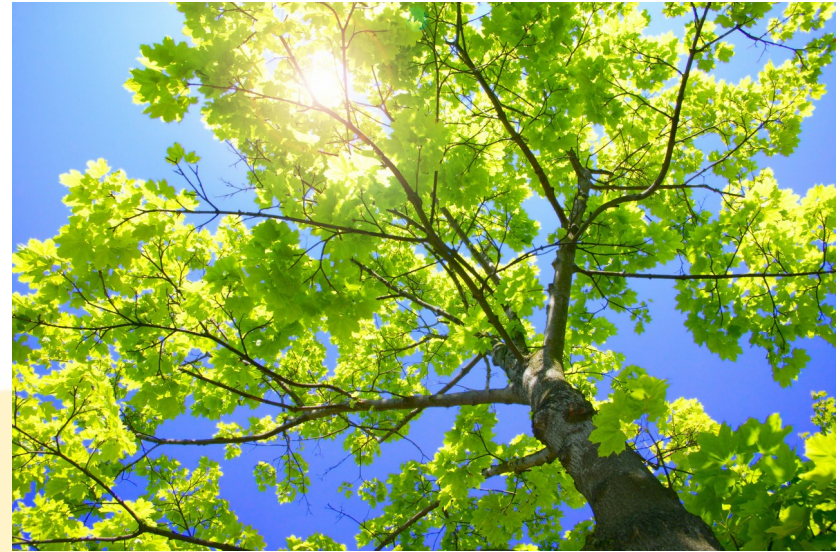
Encapsulation



Virtual machines encapsulate entire systems (hardware configuration, operating system, apps) in files

Virtualization means Green 😊

- IT is becoming more energy efficient all the time...
- ...but more power is needed to meet the needs of today's IT
- Virtualization means being able to do “more with less”



- Virtualization means that data centres are greener and is a key factor in making the IT world more energy efficient and eco-friendly

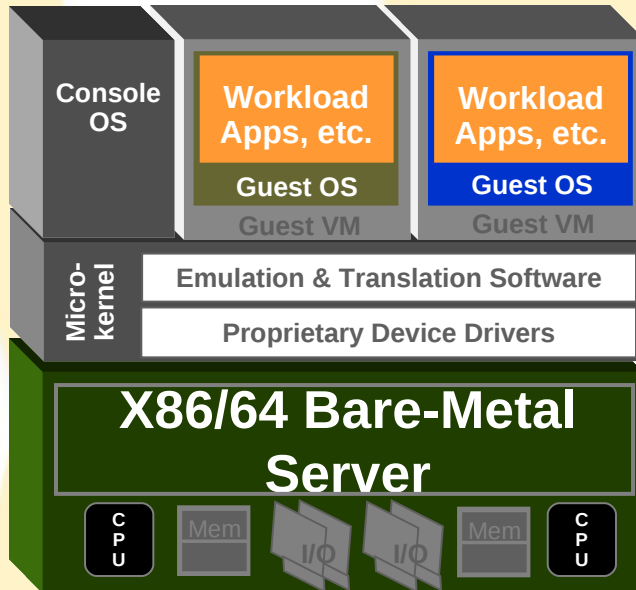
Hypervisors ⇒ Server Virtualization

- Software partitioning
 - Divide a machine into multiple virtual machines
 - One server becomes many
- Like an OS micro-kernel – very few functions
 - Resource isolation / partitioning
 - State-management
 - Scheduling
 - Suspend/Resume/Migrate
- Maturing technology for x86
 - VMware –
 - Xen
 - **OracleVM**
 - support & enterprise features (e.g. management)
 - software certification (Linux, DB, middleware, applications, ...)
 - Microsoft

Anatomy of a Virtualization Server

Emulation-based, e.g.

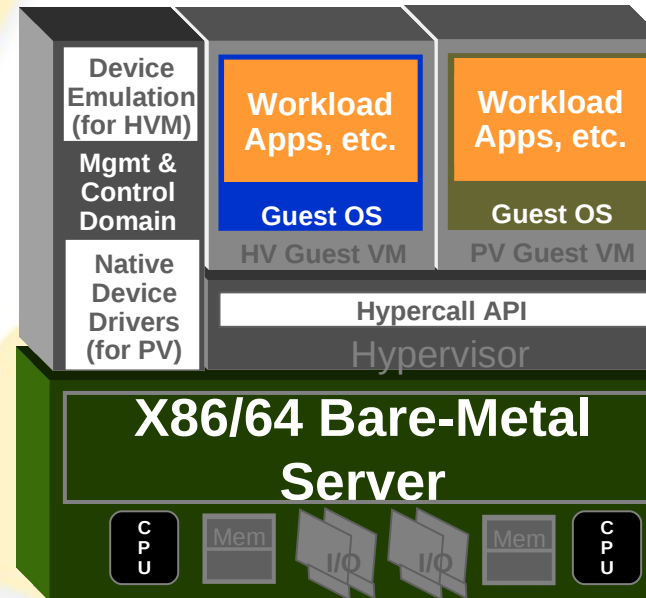
- VMware ESX Server



- Machine images run unmodified: broad compatibility
- Does not leverage or require HV hardware
- Poor I/O scalability due to emulation architecture
- Not open: dependent on virt. vendor for drivers

Paravirtualization (PV)-based, e.g.

- **Oracle VM**
- Citrix XenServer
- Windows Hyper-V



- Excellent scalability, esp. I/O with PV guests
- Requires PV OS kernel for best perf.
- Requires HV hardware for unmodified images
- Open: wide device support: uses native (e.g. Linux) device drivers

Virtualization approaches

- Goal: efficiently insure VM OSES are not “hurting” each other (e.g. Trying to unsafely modify common resources/state, etc.)

<u>Techniques:</u>	<u>Purpose:</u>	<u>Benefits:</u>	<u>Comments:</u>
Paravirtualization (PV) e.g. Oracle VM & Xen	Modify the OS and/or drivers so they know how to behave in a virtual environment	<ul style="list-style-type: none">• Good-to-excellent performance vs. bare-metal	<ul style="list-style-type: none">• PV OS kernel req'd (but rapidly not an issue)
Emulation / translation , e.g. VMware ESX	Design the virtualization server to intercept or “trap” harmful requests and/or translate requests into appropriate forms	<ul style="list-style-type: none">• Use unmodified OS• HVM hardware not req'd (but rapidly not an issue)	<ul style="list-style-type: none">• Poor scalability esp. under I/O load
Hardware virtualization (HVM) ; e.g. Oracle VM & Xen	Design the hardware so it knows how to handle inappropriate requests itself. (Note: PV drivers can be used with an otherwise unmodified OS, e.g. Windows, to improve performance)	<ul style="list-style-type: none">• Use unmodified OS	<ul style="list-style-type: none">• Poor perf. today• HVM hardware req'd (but rapidly not an issue)



JRockit JVM (and JRockit VE)

JRockit Product Family

JROCKIT MISSION CONTROL

- Complete insight into application & JVM behavior
- Zero performance overhead in production environments
- No application modification or configuration required

JROCKIT REAL TIME

- High-performance real-time solution for standard Java
- Industry leading Deterministic Garbage Collector
- Millisecond response times with “five nines” guarantee
- Improve application performance & latency with unique tooling

JROCKIT VIRTUAL EDITION

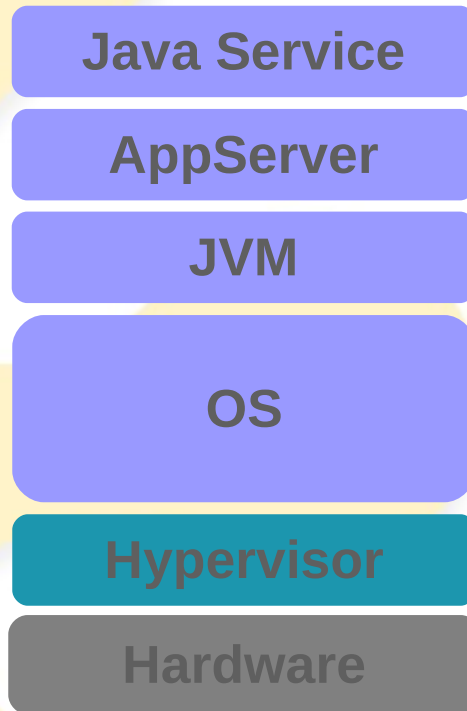
- Fly-weight Java container for virtualized environments
- Improve datacenter efficiency - do more with less
- Simpler and more powerful VM management
- **Scheduled for release in 2009***

JROCKIT JVM

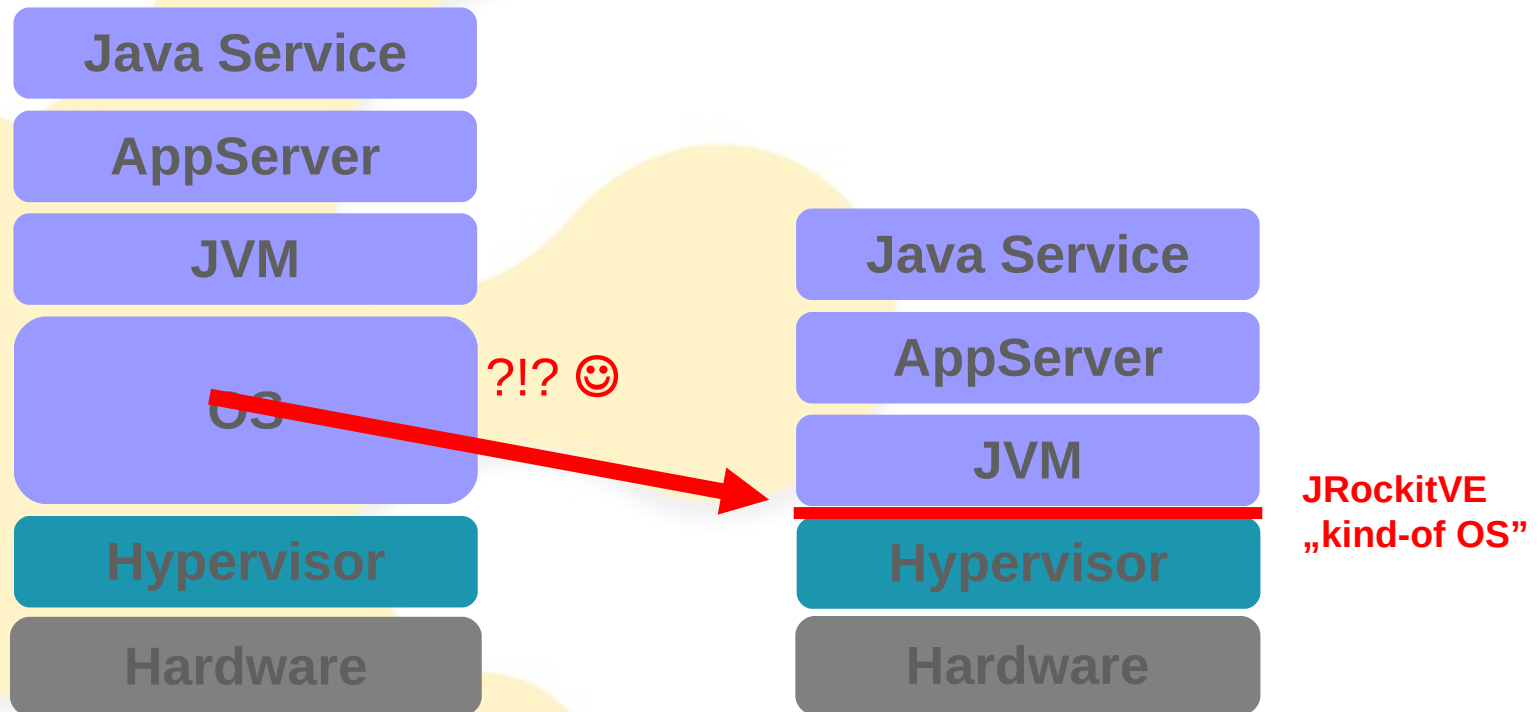
- World-class performance
- Powerful diagnostics
- Full support from Oracle

* Forward-looking statement, see disclaimer on earlier slide

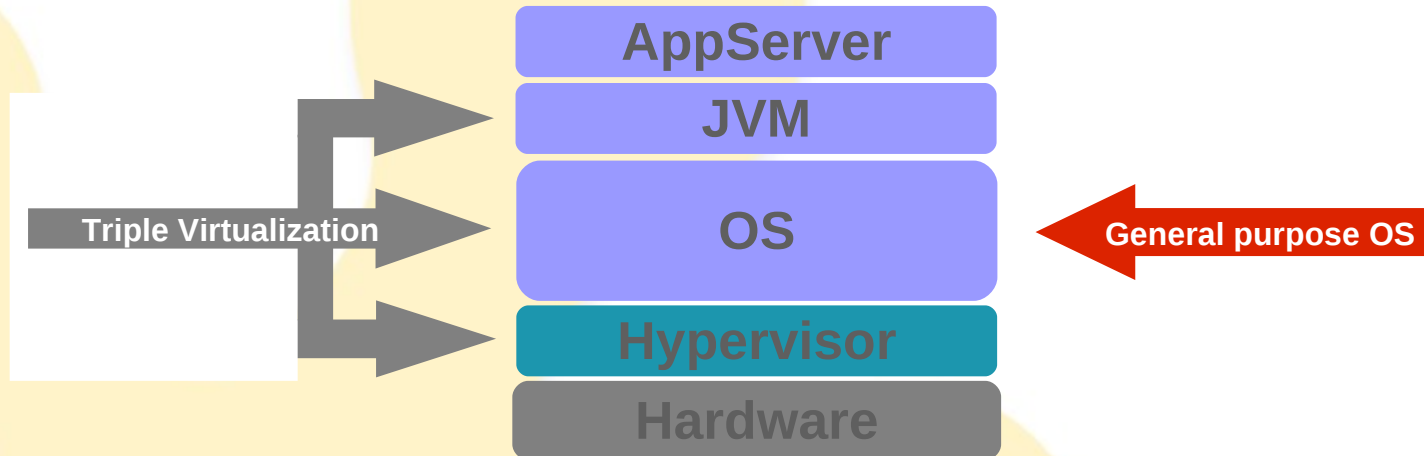
Server-side Java Stack in a Virtualized Environment



Server-side Java Stack in a Virtualized Environment **with JRockit VE**



Zooming in



- Triple virtualization
 - Triple virtualization by Hypervisor, OS, and JVM
 - Virtualization layers uncoordinated
 - Redundant activities in each layer
- Large and slow **general purpose OS**
 - Footprint
 - Maintenance
 - Performance
 - Security threats
- So why not to regain the lost performance

JRockitVE - Hypervisor Optimized Java

From OS process to Java Service on Hypervisor

- Create a Java **Virtual** Machine - JRockitVE
 - Run the JVM **directly on** the hypervisor
 - Use the hypervisor for some services
 - Real Device drivers
 - Storage and network I/O
- **Paravirtualize**: make Java and the hypervisor work together
 - This new teamwork is **unique** in that the new layer is built from the **ground up** on paravirtualization
 - More than JeOS
 - Jusy Enough Operating System

Benefits to End-User

- Reduced disk and memory footprint
- Improve raw speed
- Reduced pause times
- Strengthened security
 - Fewer entry points
 - Fewer virus threats
- Simplified Patching
- Reduced OS-license costs
- High-availability functionality
(suspend/resume/migrate)

Hypervisors Made JRockitVE Feasible

- The barrier to entry for a new OS **was** way too high
 - Bare Metal can coexist with OSES because of hypervisors
 - OS file systems / backups etc continue to work
- Supporting various device drivers was very expensive
 - Similar physical device different from each other
 - On a hypervisor there are only a couple (1 network, 1 disk)

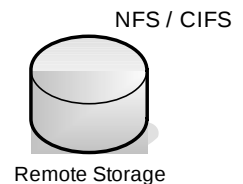
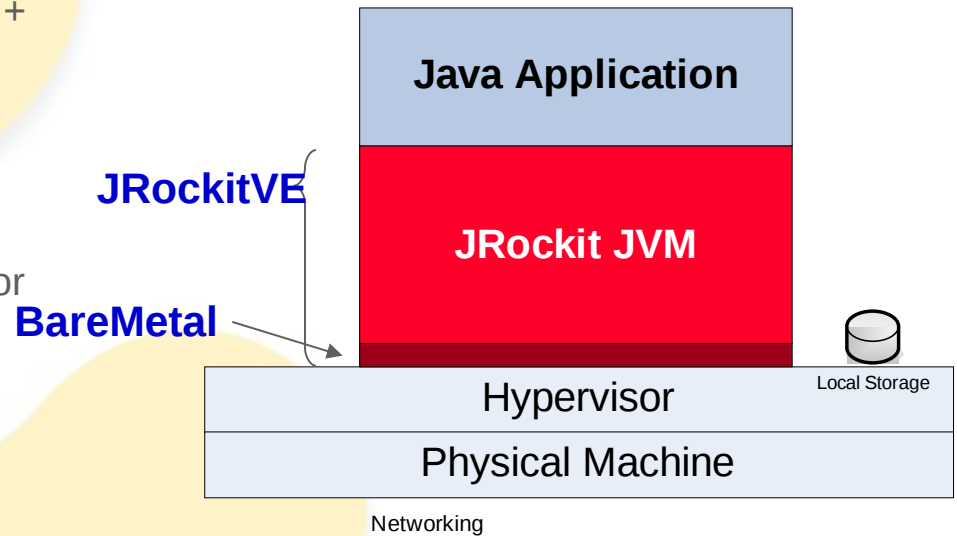
See the DEMO 1...

JRockitVE is **Not a Good Fit** When...

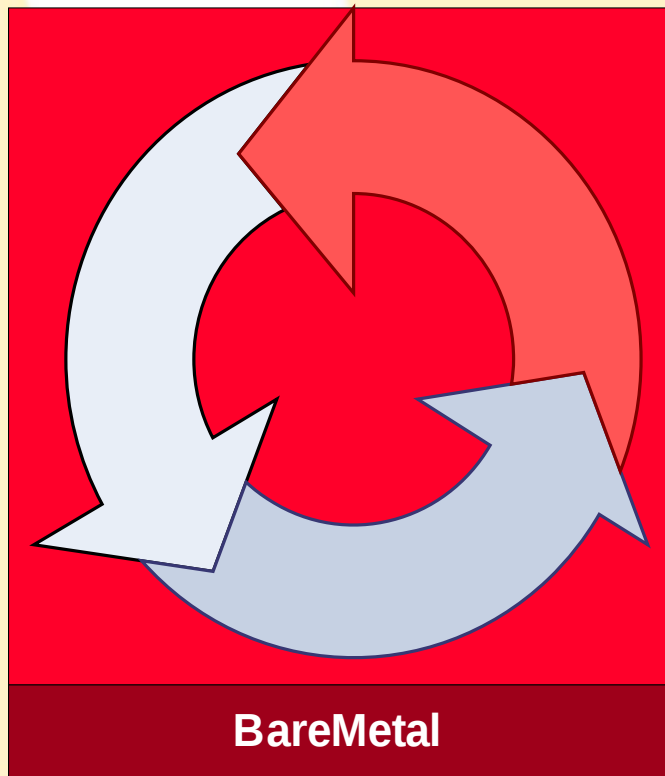
- The application needs a graphical display
 - JRockitVE is a server environment :
 - no screen, no GUI, no sound
 - minimal keyboard input (not for app anyway...)
- The application uses OS dependent native code (JNI) **might be difficult** when loaded in JRockitVE
 - Can be executed using a proxy OS-service, but the overhead is large
 - It will always be slow for JRockitVE
 - But JRockitVE exposes Linux-like ABI (Posix, etc.)
- **Very rare problem** anyway in today's enterprise Java...

JRocketVE Overview – Java Service on Hypervisor

- JRocketVE:
 - JRocket JVM + new "BareMetal" layer (VE) + tools
 - the JRocket JVM is **not modified**
 - Neither are any apps on top
 - like **WebLogic Server**
- BareMetal layer – a *nix-like emulation layer for Java
 - Built from scratch (**not Linux-based, nor *BSD, ...**)
 - Can run a JRocket JVM for Linux
 - Low-level: networking, thread-scheduling, memory management, file storage
- Not an OS in any **normal** sense
 - Single app: JVM
 - Only a single JVM
 - No real device drivers
 - Single-process, single-user



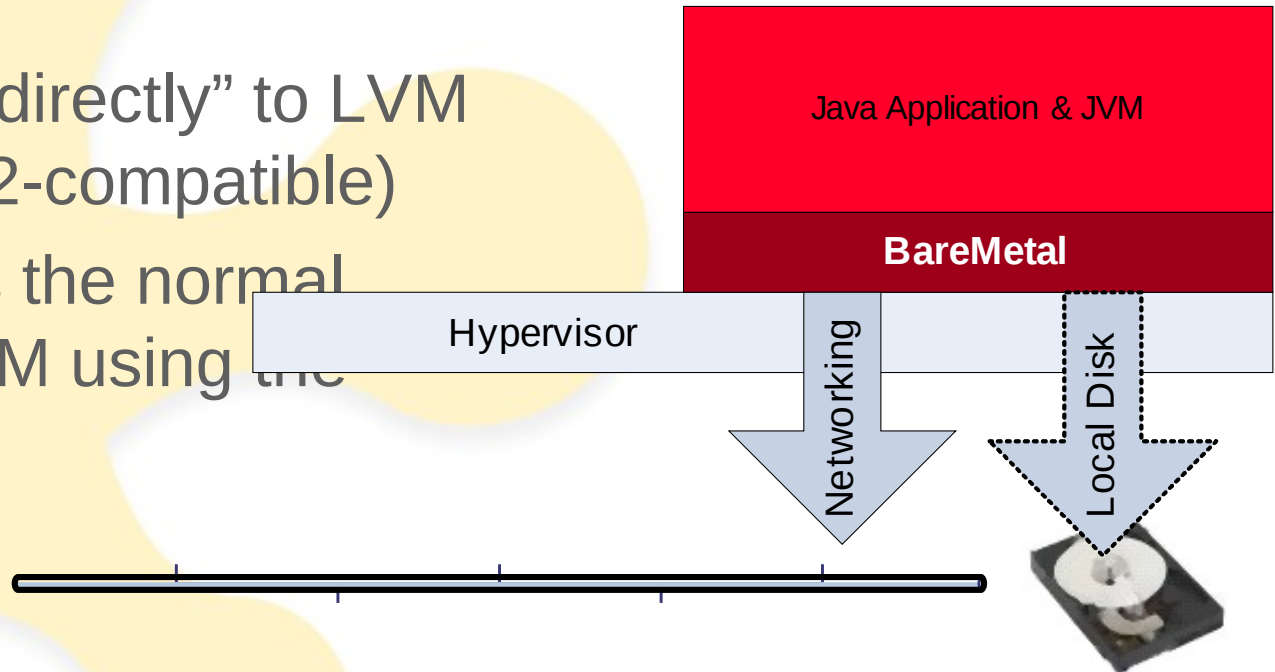
Execution Inside JRockitVE



- Java execution
- JVM native code
 - Thread system
 - Synchronization & locking
 - Garbage collection
- Management Agents (JMX etc.)
- WebLogic Server native code
- Some 3rd party native code
 - Linux ABI (Posix, etc.)

Directly Out

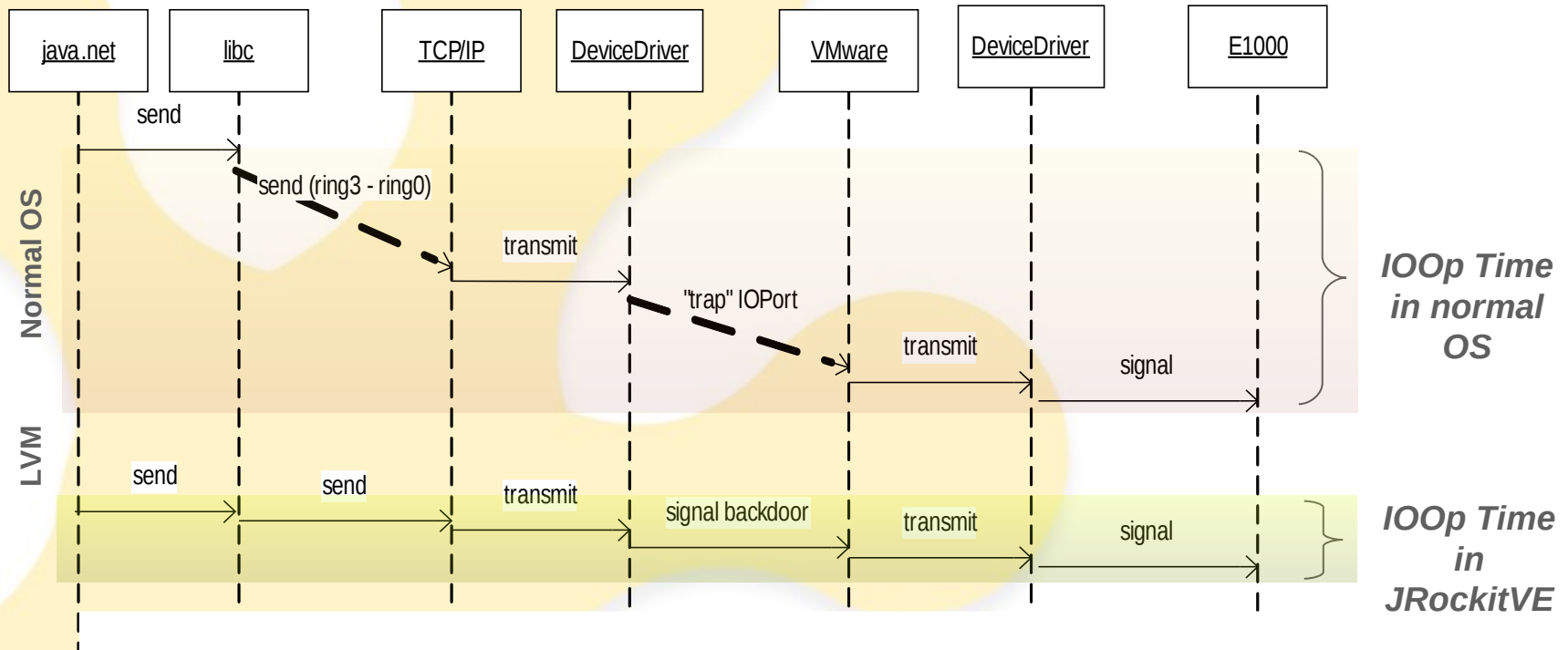
- Networking goes "directly" out through the hypervisor
 - TCP/IP stack implementation in BareMetal layer
- File I/O can go "directly" to LVM Server disk (ext2-compatible)
- Debugging uses the normal JVMTI/-DI to LVM using the network



No Normal Kernel

- Almost everything executes in user mode
 - Exception – page faults (handle stack overflow)
- No need for protection
 - Single process, single-user operating system
 - Java code is per definition protected
 - The JVM is the most privileged level
- Reduced path length for many operations
- Analogy: game consoles (e.g. XBox)

Example: Shorter Pathlength for I/O



- Avoid transition cost from user mode to kernel mode
- Avoid trap from VM to hypervisor
 - Similar effect achieved by installing VMware Tools on Guest OS or **paravirtualizing**

Disk Footprint - Single Image

- An appliance image is small

- Bare Metal itself is very small

- ~2MB

- Needs a JRE/JDK

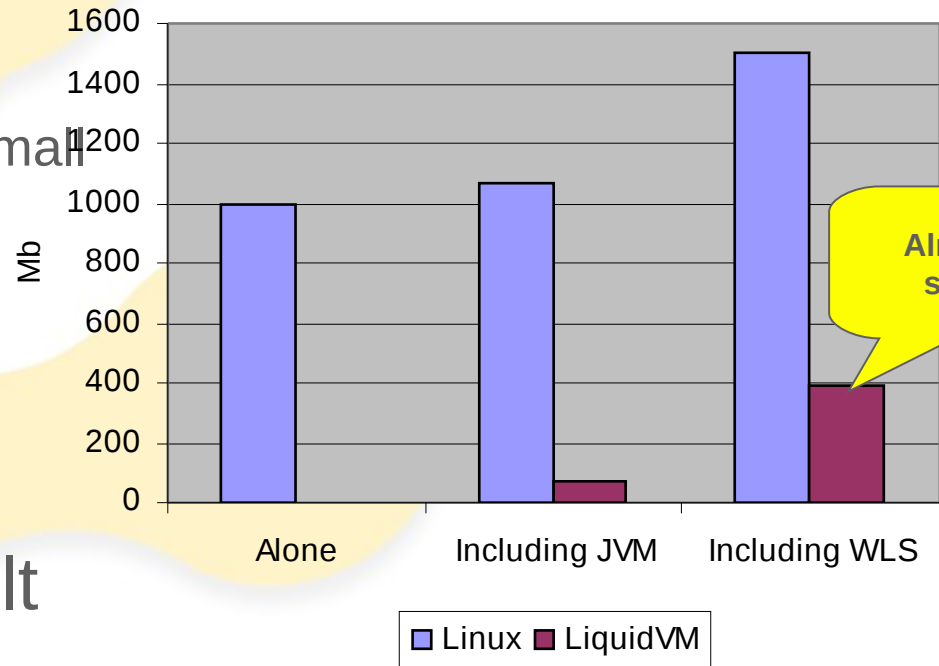
- Additional ~40MB

- WebLogic Server

- Additional ~300MB

- Compare to a default OS installation

- RedHat EL4.0 ~ 1000MB





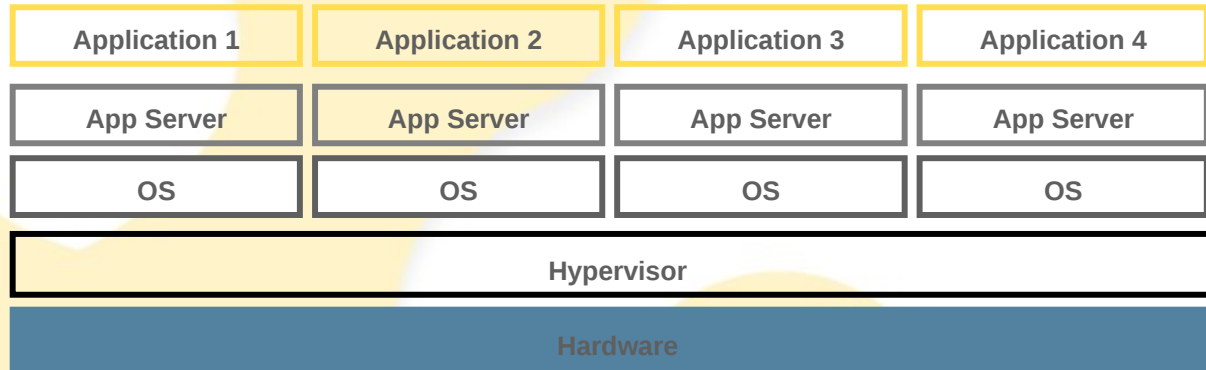
Let's dream a bit for now...

(and diverge for a moment from what is/will be in JRockitVE)

Potentially Hot Combination !

- The „JVM on hypervisor” combination leaves a lot of **potential** for many more interesting optimizations
- Please note: these are just ideas, i.e. no commitment they will ever be realized in practice
 - please refer to the disclaimer slide earlier

Example of potential optimization: Memory Footprint Savings



- Many similar virtual servers running on the same server
- Opportunity to shrink each virtual server's footprint
 - Make each virtual server smaller
 - Share identical data between the different servers
 - Share "free" memory between servers

Example of potential optimization: Less Footprint: Share Free Memory

- A large portion of all memory for a JVM process is free memory
 - The more free memory the less frequent the GC
 - Free memory used to avoid temporary allocation peaks and resulting OutOfMemory-conditions
- Better to GC than to Swap
 - Balloon drivers
 - When hypervisor reports high memory pressure – GC and shrink heap
 - Basically free memory is shared between all the JVMs on the box
- When you have many JVMs on the same box the savings can be substantial

Really - the sky is the limit...

- Other potential ideas:
 - Java application aware scheduling
 - E.g. understanding Java synchronization concepts
 - Runtime profiling
 - Inter-JVM communication
 - Even better deterministic JVM
 - e.g. for real-time workloads
 - ???
- Think about optimizations possible, as a JVM designer – what if you could have scheduler, memory manager, device manager, etc. under your direct control ?



Back to JRockit VE (and reality...)

Security ?

JRockit on

Linux/Windows/Solaris:

- 500+ software packages
- 10+ software sources
- 10 open ports

- JeOS, like those from Oracle (OEL JEOS) or Ubuntu, simplify this, but why not to go further
 - esp. For server-side Java apps

JRockit VE:

- 1 software package
- 1 software sources
- 1 open port (can be disabled)

- „freshness effect” (at least for the time-being)
- also, hypervisor and virtualization software give additional security layers

OK, but how to managed it

anyway: much less needs to touch the OS now

- JRocket tools
 - JRocket Mission Control
 - JRocketVE also has simple diagnostic tools
- JRocket VE has built-in virtual disk and file system
 - Can be „mapped” onto real-storage (by hypervisor)
 - Can be manipulate inside and outside
- Use virtualization management software
 - E.g. OracleVM, OracleVM Manager, Oracle Enterprise Manager
- Store your logs out of virtual disk
- Syslog
- JRocket VE can run SSH-agent
- Also: use the management power of the application server (e.g. WebLogic Server)

The Appliance Tool

- A simple metaphor for turning **local Java applications** into virtualized appliances
- Given a local Java app, a hypervisor agnostic configuration file and the name of a hypervisor
 - Creates a binary blob, an assembled "appliance" that can be dropped into the hypervisor (or even **into the cloud**)
- Binary blobs / appliances can be disassembled to their component parts
- Binary blobs / appliances can be modified without disassembly

See the DEMOS 4-7

Enough talking, time for **DEMOS !**



Demos 1/3

1. Build your own OS kernel

- and see how much the hypervisors simplify this !
- the code is from David Chisnall's book „The Definitive Guide to the Xen Hypervisor” (Prentice Hall)

2. See JRockitVE in action

- Hello World (or: hey, who stole the OS ?)

3. More JRockit in action

- How about running a bigger Java app without the OS
 - maybe WebLogic Server ?

Demos 2/3

1. Build your own Java appliance
 - and run your very own Java applications without the OS
2. More fun 😊
3. More fun 😊

Demos 3/3

1. More fun with the JRockitVE Appliance Tool



Q & A ?

Thank you ☺ !



Waldek Kot

private email:

waldek.kot@gmail.com

business email:

waldemar.kot@oracle.com mobile:

+48 660 78 55 78

see my blog:

<http://jdn.pl/blog/88>