

OS^v

Nadav Har'El
Cloudeus Systems
nyh@cloudeus-systems.com

October 2013, Haifa Linux Club

OSv

- A new operating system for **virtual machines** in the cloud.
- **BSD-licensed** free software.
- Can run Linux software, but not based on Linux.

Why was it written?

How was it written?

What makes it different?

What makes it interesting?



Glauber Costa
KVM, Containers, Xen



Nadav Har'El,
Nested KVM

OS^V



Avi Kivity KVM
originator



Pekka Enberg,
kvm, jvm, slab



Dor Laor, Former kvm
project mngr

Or Cohen



Dmitry Fleytman



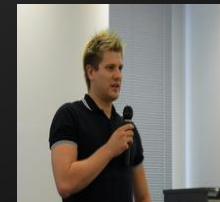
Ronen Narkis



Guy Zana



Christoph Hellwig Tomasz Grabiec



Why OSv?

In the beginning there was hardware

... and then they added
an application

... and then they added
an operating system

... and then they added
managed runtime

... and then they added
a hypervisor

Note how layers added, never removed.

Typical Cloud Stack

Your App

Application Server

JVM

Operating System

Hypervisor

Hardware

A Historical Anomaly

Your App

Application Server

JVM

provides protection and abstraction

Operating System

provides protection and abstraction

Hypervisor

provides protection and abstraction

Hardware

**Our software stack
Congealed into existence.**

Too Many Layers, Too Little Value

Property/Component	VMM	OS	runtime
Hardware abstraction	V	V	V
Isolation	V	V	V
Resource virtualization	V	V	V
Backward compatibility	V	V	V
Security	V	V	V
Memory management	V	V	V
I/O stack	V	V	
Configuration		V	

Too Many Layers, Too Little Value

- **Containers** solve a similar problem by giving up the hypervisor layer.
- But the hypervisor is good!
 - Common
 - Doesn't impose specific OS
 - Security, isolation, etc.
- Instead, **OSv** reduces duplicate roles of the OS.

Virtualization


Virtualization 1.0



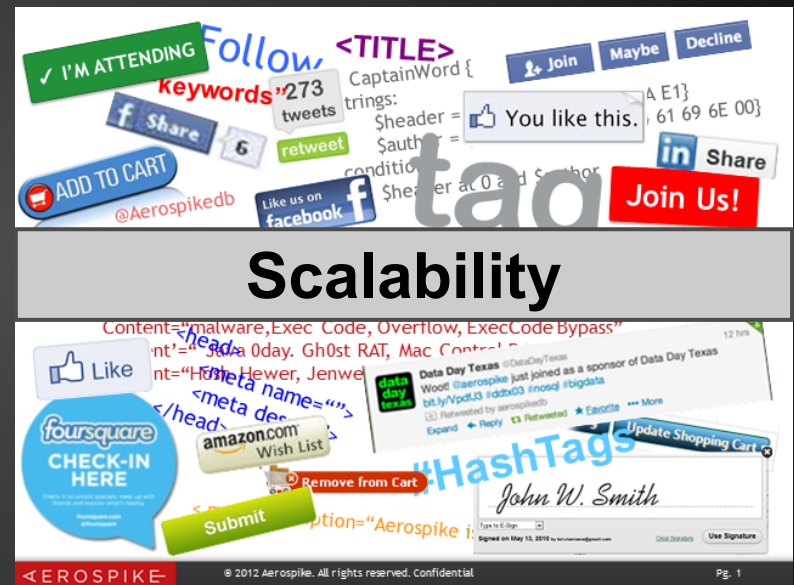
Transformed the
enterprise from
physical2virtual

Virtualization 2.0



Compute node

virtual server

Virtualization 2.0, Massive Scale



Scalability

Content="malware,Exec Code, Overflow, ExecCode Bypass"
nt'=" Java 0day, Gh0st RAT, Mac Control-
nt="Hole Hower, Jenwe
<meta name=">
</head>
foursquare CHECK-IN HERE
amazon.com Wish List
Submit
Remove from Cart
HashTags
John W. Smith
Update Shopping Cart
AEROSPIKE
© 2012 Aerospike. All rights reserved. Confidential
Pg. 1

Virtualization 2.0, Dev/Ops



Virtualization 2.0, agility!

Deployments at Amazon.com

11.6

Seconds mean time
between deployments
(weekday)

1,079

Max number of
deployments in a
single hour

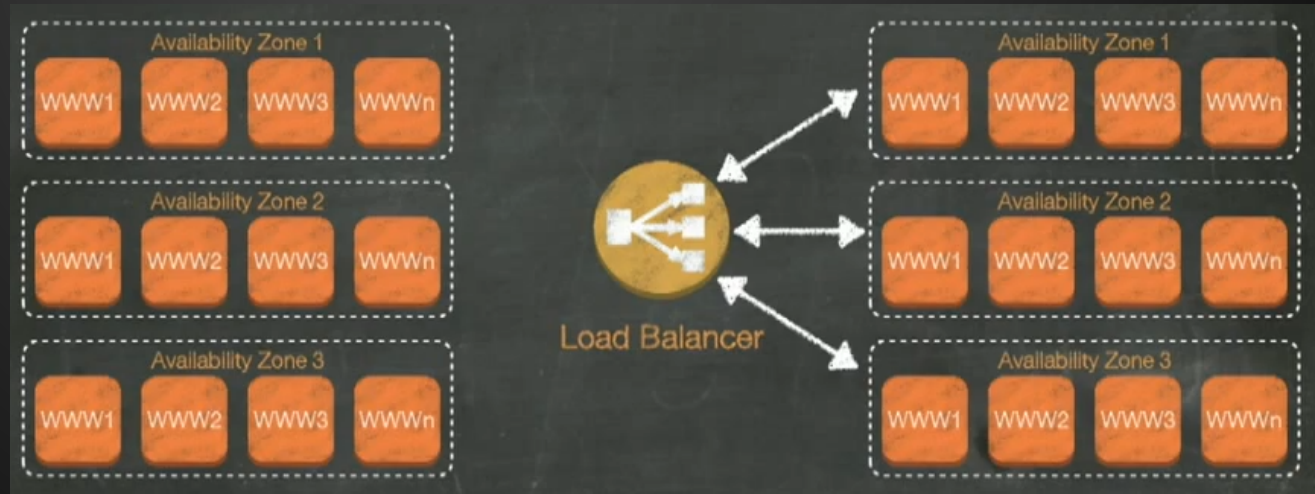
10,000

Mean number of hosts
simultaneously receiving
a deployment

30,000

Max number of hosts
simultaneously
receiving a deployment

**Rolling upgrade
within seconds and
a fall back option**



OS that doesn't get in the way

NO Tuning

NO State

NO Patching

**X4 VMs per sys
admin ratio**

Virtualization 2.0

Linux progress

- Supports **more** hardware.
- **Bigger**, more services.
- **More** libraries, layers, configuration.

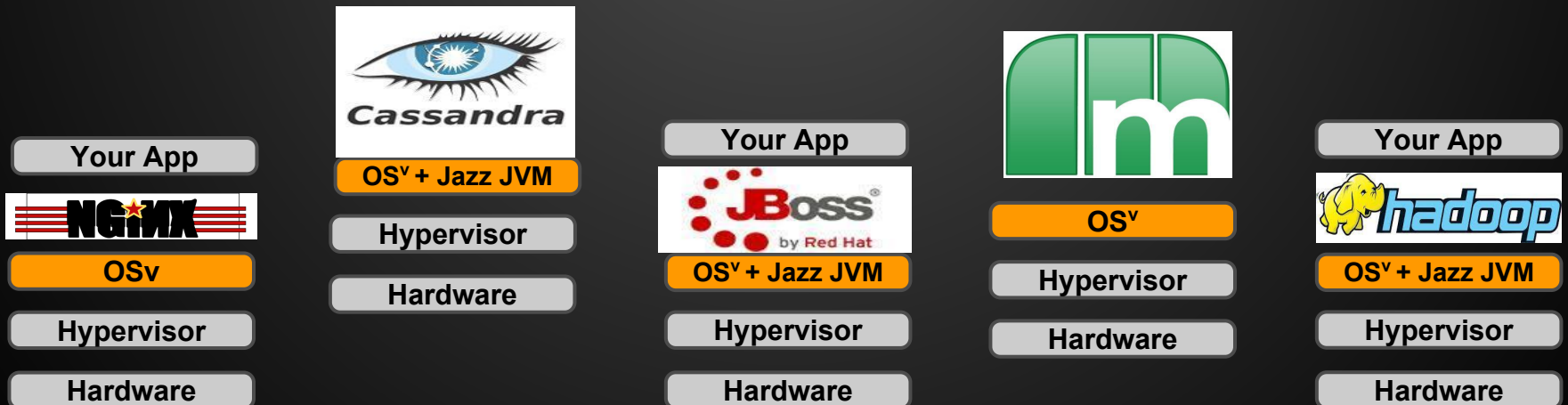
Needs from VM OS

- **Small** amount of virtual hardware.
- **Fewer** and fewer services (cloud and runtime provide them).
- **No** users.
- **No** configuration.

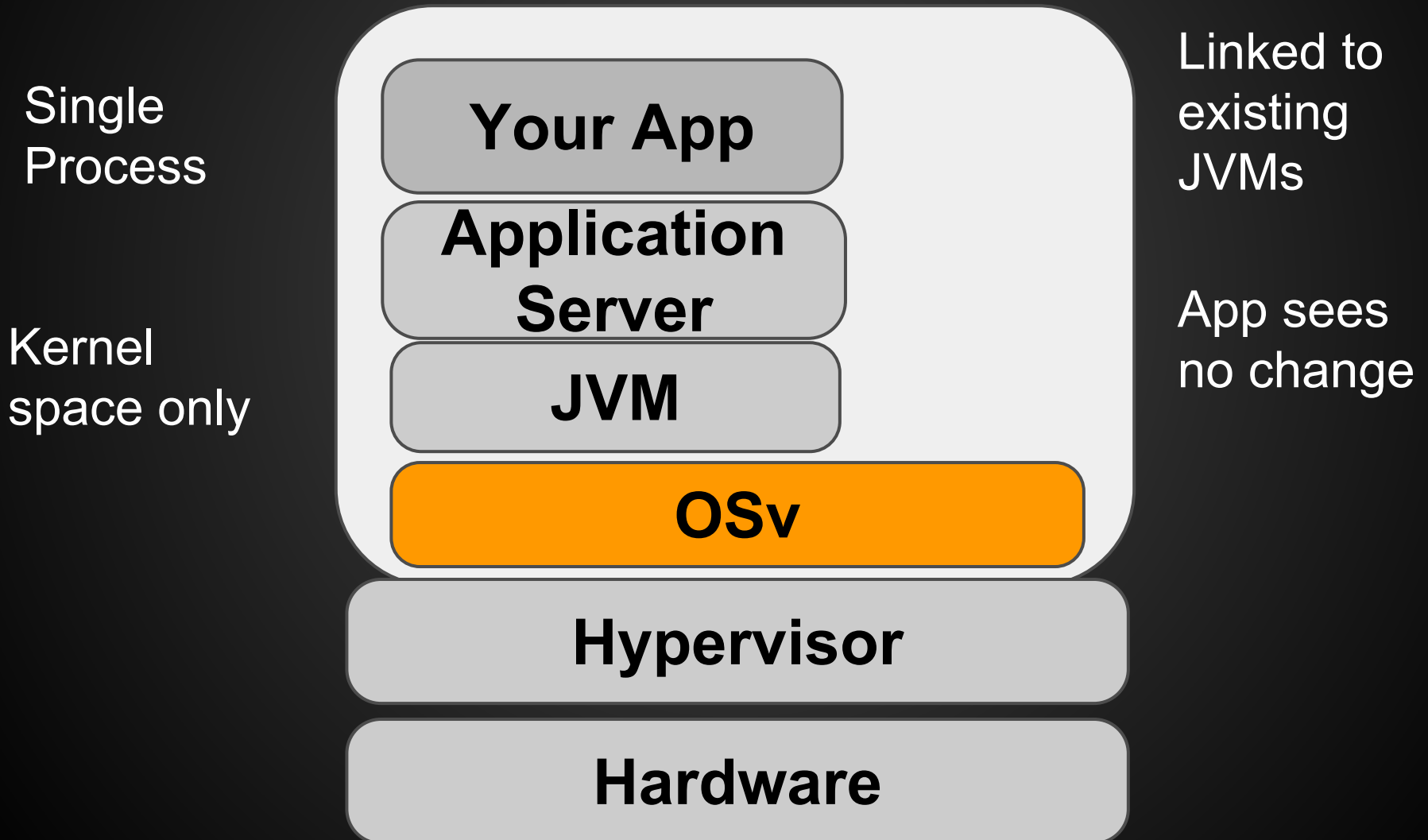
less is more.

Mission statement

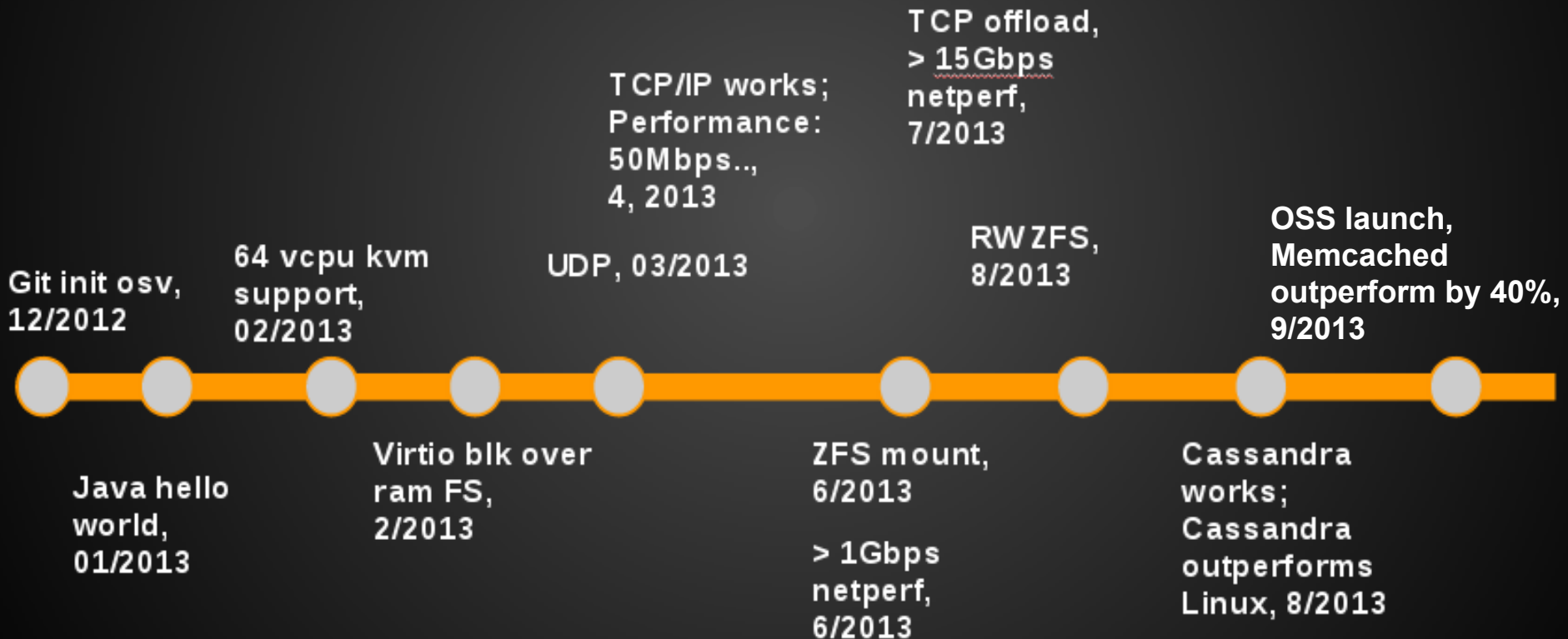
Be the **best OS**
powering virtual machines
in the **cloud**



The **new** Cloud Stack - OS^v



Milestones



OSv's code

- A **new kernel** written from scratch in **C++**.
- **ZFS** filesystem from OpenSolaris (later OpenZFS).
- **TCP/IP** stack from FreeBSD (**temporary**).
- Various libc compatibility functions from **Musl**.
- Some VFS code from Prex (temporary).

Status

- Runs:
 - Java, C, JRuby, Scala, Groovy, Clojure, JavaScript
- Outperforms Linux:
 - SpecJVM, MemCacheD, Cassandra, TCP/IP
- 400% better w/ scheduler micro-benchmark
- < 1sec boot time

Open Source

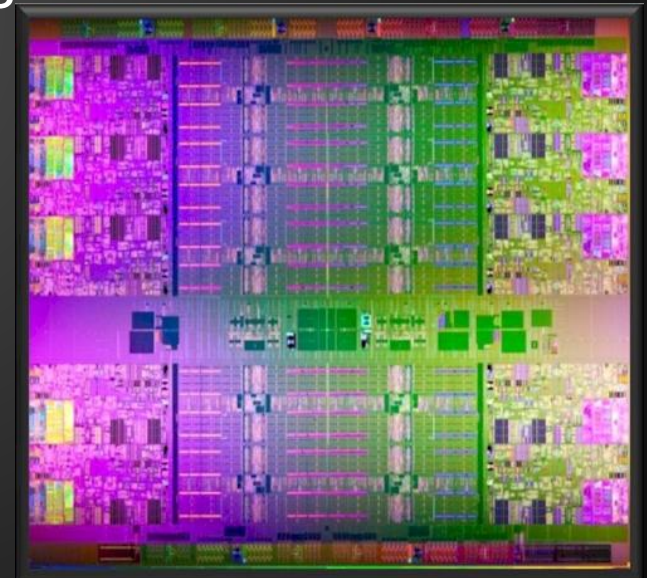
Fork me on GitHub

- These days, credibility == open source
- Looking for cooperation:
 - Kernel-level developers
 - Management stack
 - Dev/ops workflow
- BSD license
- Already have contributions from multiple companies



Architecture ports

- 64-bit x86
 - KVM - running like a bat out of hell
 - Xen HVM - running (still work in progress)
 - VMware - planned in 2 months
- 64-bit ARM - planned
- Others - patches welcome



Why a new kernel?

- Allows us to make different assumptions
 - Single address space
 - Spin-locks are evil
 - No system calls
- Easy to experiment with new ideas without decades of legacy.
- Easier to add new think-outside-the-box APIs.
- Smaller code
 - Less is more
 - Easier to develop

Why a new kernel?

Fun!

Feel like a pioneer!

- You are invited to join the fun, and be a pioneer.
- Work on OSv. It's free software!
- If you want to also make money,
Cloudius Systems is hiring!

OSv's kernel design principles

- **Single address space.**
 - No processes, just threads.
 - No separate kernel address space.
 - No protection between user-space and kernel.
- **No “system calls”, just function calls.**
 - Runs Linux shared-objects by implementing the Linux/Glibc ABI.
 - No copy of the system call arguments.
 - New non-Posix APIs can be really zero-copy.

OSv's kernel design principles

- **Cheap threads and context switches.**
 - No interrupt context - just wake up handler thread.
- **No spinlocks.**
 - Spinlocks are notorious for VM OSs: cause **lock holders preemption** problem.
 - Needed in interrupt context - which we don't have.
 - We have **lock-free mutexes**.

OSv's kernel design principles

- **Linux emulation.**
 - Enough for running common runtimes (JVM) and important applications.
 - But not more than necessary. OSv is not Linux.
- **New OSv APIs for applications/JVM**
 - New APIs for really zero-copy I/O.
 - New APIs giving the single application more access, e.g., JVM can use **page table dirty bits** instead of emulating this feature slowly.

Why C++11

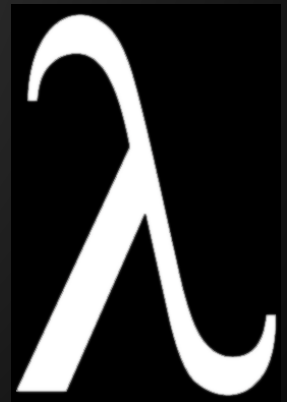
- Forget all they told you about C++!
- We didn't choose C++ for creating complex type hierarchies.
- C++11 finalized in August 2011.
- C++11 is about:
 - Avoiding boilerplate repetition
 - Easy and safe reuse of data structures
 - Rich standard library (STL, Boost)
 - No runtime overhead
 - Support for concurrent memory access (atomic variables, memory ordering)

Less boilerplate code

```
int before(struct something *p)
{
    int r;

    r = -ENOENT;
    if (!p)
        goto out2;
    mutex_lock(&p->lock);
    r = -EINVAL;
    if (!p->y)
        goto out1;
    mutex_lock(&p->y->lock);
    r = ++p->y->n;
    mutex_unlock(&p->y->lock);
out1:
    mutex_unlock(&p->lock);
out2:
    return r;
}
```

```
int after(something* p)
{
    if (!p)
        return -ENOENT;
    WITH_LOCK(p->lock) {
        if (!p->y)
            return -EINVAL;
        WITH_LOCK(p->y->lock)
            return ++p->y->n;
    }
}
```



Tracepoints

```
TRACEPOINT(trace_mutex_lock, "%p", mutex *);  
TRACEPOINT(trace_mutex_lock_wait, "%p", mutex *);
```

```
// ...
```

```
void mutex::lock()  
{  
    trace_mutex_lock(this);
```

```
[/]$ perf stat mutex_lock mutex_lock_wait sched_switch  
mutex_lock  mutex_lock_wait  sched_switch  
      11             0           2  
     885             0          181  
     154             0          152  
     154             0          154  
     404             0          190  
     222             0          157  
     150             0          152
```

New ideas for a new kernel

Lock-free Mutex

Classic mutex lock() implementation:

- On UP:
 - Mutex contains internal state: “locked” flag, and “wait queue”.
 - Disable preemption to protect it.
- On SMP:
 - Use spin lock to protect the state.
- Spin lock only used for a **short while** - during the few instructions setting the internal state. NOT while sleeping.

So no problem, right?

Lock-free Mutex

Spin lock only used for a short while, so no problem, right?

Wrong on a virtual machine!

On VMs, virtual CPUs can “pause” for long durations:

- During an **exit** (host interrupt, I/O, etc.)
- Host CPU overcommit (other guests, host processes).

Lock-free Mutex

On VMs, virtual CPUs can pause for long durations. So:

- One thread lock()s a mutex and its vCPU is **preempted** while holding the spin-lock.
- Other threads on other vCPU try to lock, get stuck in infinite loop instead of going to sleep!

A known problem for VM OSs -

Lock-Holder's Preemption problem.

Lock-free Mutex

Lock-Holder's Preemption problem

- Traditional kernels try to solve this with hacks.
- OSv solves this by using **no spin locks!**
- Mutex implementation without spinlocks:
 - “Lock-free”
 - Uses modern SMP atomic operations: atomic increment, Compare-Exchange, etc.
 - But much harder than first appears!

Lock-free Mutex

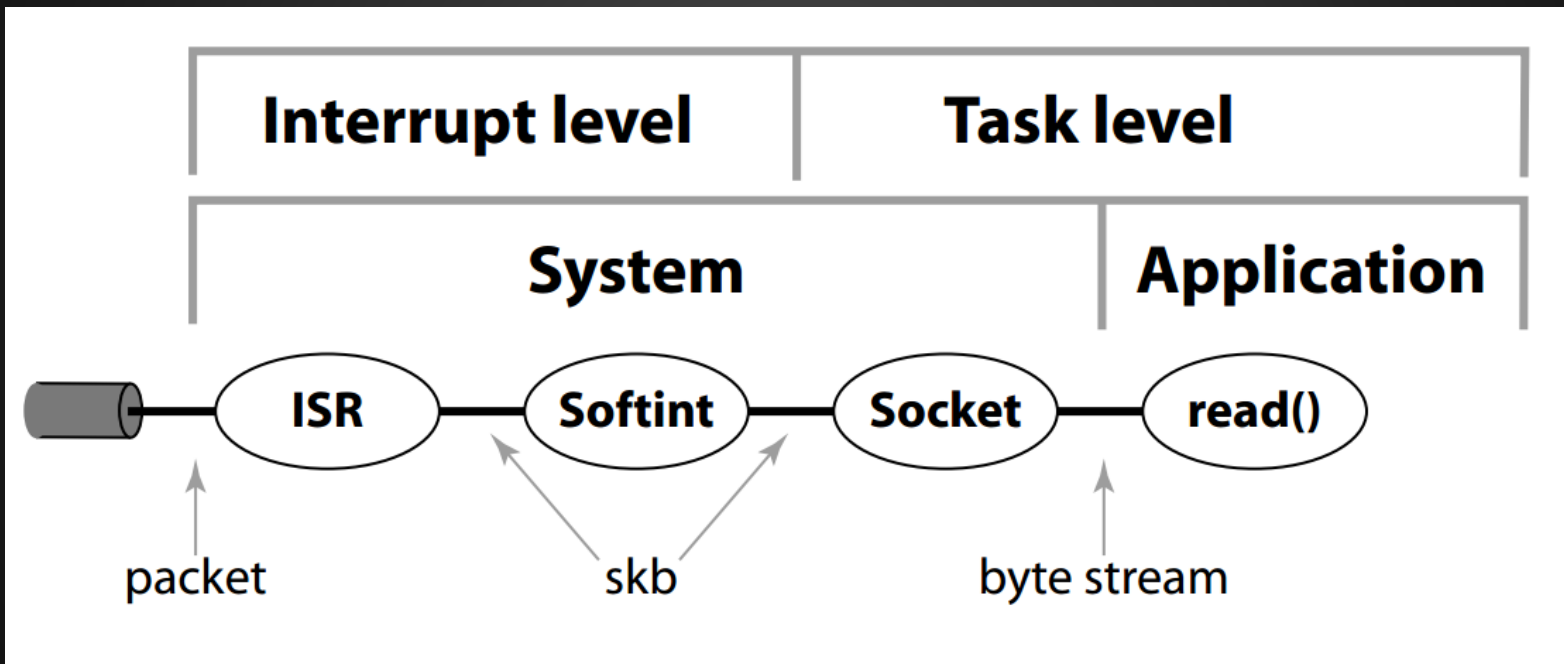
- **Lock():**
 - Atomically check if *locked*, and if not set *locked*=true (CAS).
 - If was already *locked*, atomically add this thread to wait queue (lock-free list algorithm).
- **Unlock():**
 - Wake first thread on the wait queue, if any.
 - If not, unset *locked*.
- **EPIC FAIL! What it:**
 - lock() saw *locked*...
 - unlock() saw empty wait queue, not woke anyone.
 - ...lock() adds to queue, but nobody to wake it!

Lock-free Mutex

- Not easy to fix! (try it)
- Use an algorithm proposed in 2007 by *“Blocking without locking or lfthreads: A lock-free thread library”* by Anders Gidenstam and Marina Papatriantafilou
 - unlock() realizes there’s a concurrent lock but empty queue;
 - It **hands off** the wakeup job to one of the concurrent locks.
 - A common technique in LF algorithms: “helping”.
 - Still easier said than done :-)
 - And really cool.

Rethinking the TCP/IP stack

Common kernel network stack



Rethinking the TCP/IP stack

The common approach sucks on modern SMP:

- Cache bouncing as data moves through layers
 - Interrupt processed on different CPU than read()
 - Linked lists
- Locking costs, even if not contended.
 - Lock on socket, file descriptor, etc.
- Data copying
 - Packets to socket buffer, kernel to user space, etc.

Van Jacobson Net Channels

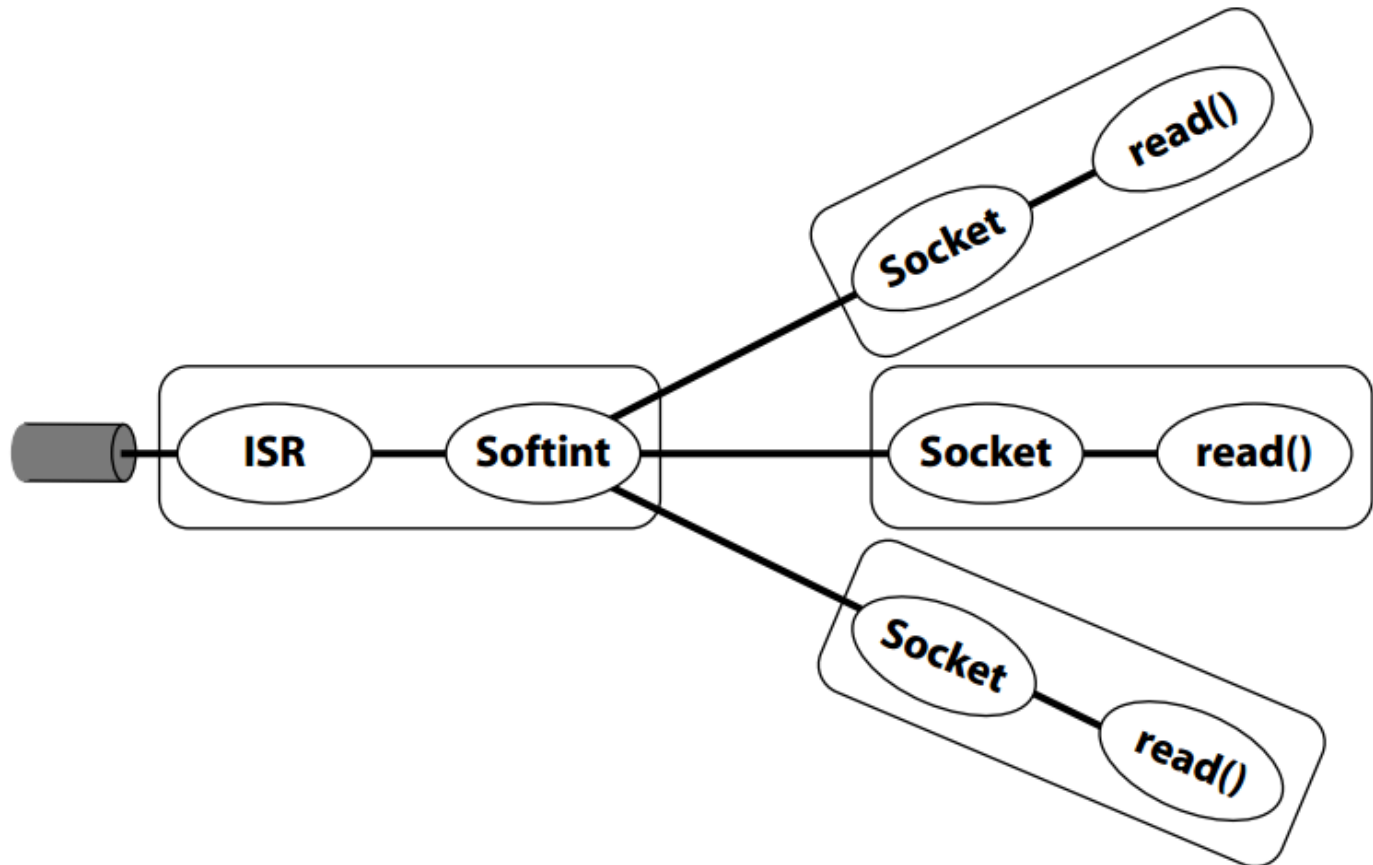


Do as much processing as possible on the CPU where the application is running.

Van Jacobson Net Channels



Net Channel design:



Van Jacobson Net Channels



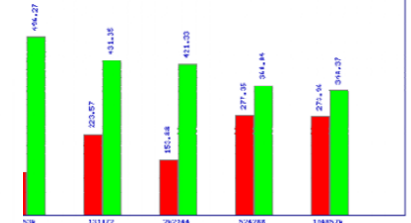
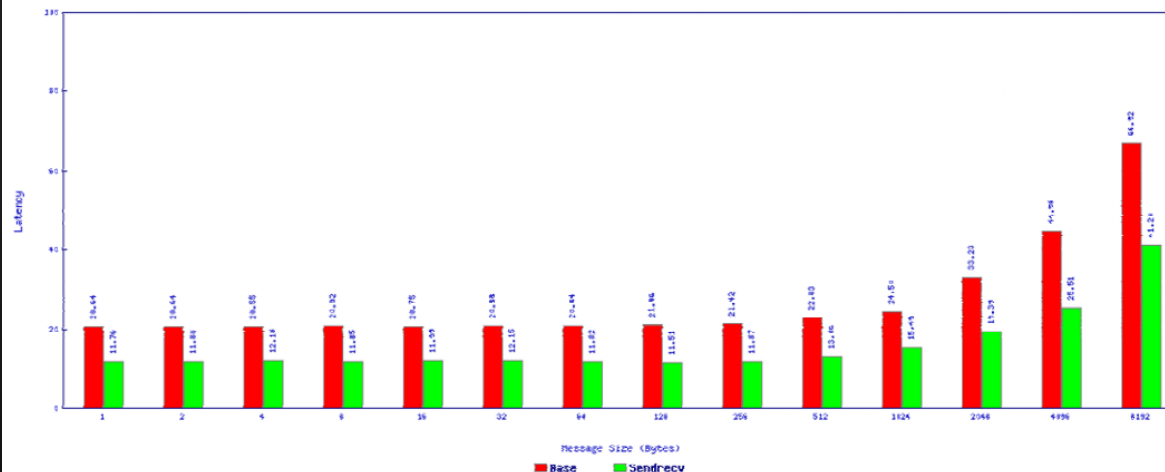
LAM MPI: Intel MPI Benchmark (IMB) using 4 boxes (8 processes) SendRecv bandwidth (bigger is better)

Intel Benchmark Absolute Bandwidth Comparison (Driver = e1000, Lib = POLL, Nodes = 4, Grid = 4(n) x 2(p))



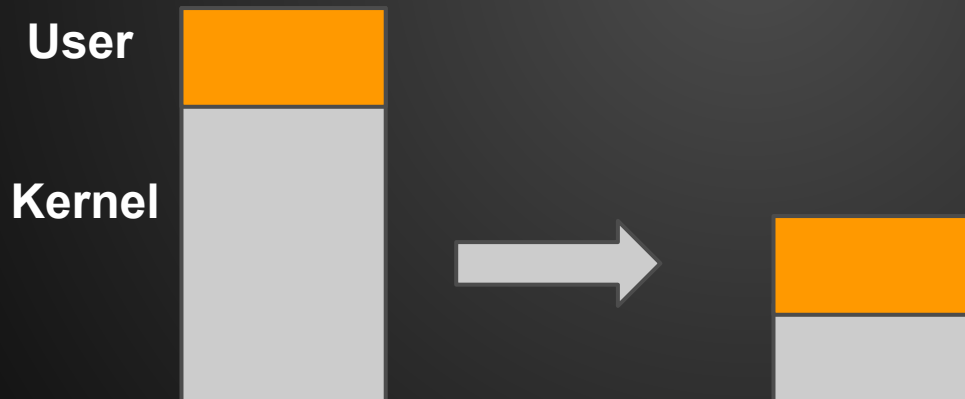
LAM MPI: Intel MPI Benchmark (IMB) using 4 boxes (8 processes) SendRecv Latency (smaller is better)

Intel Benchmark Absolute Latency Comparison (Driver = e1000, Lib = POLL, Nodes = 4, Grid = 4(n) x 2(p))



Virtio-app || Data plane

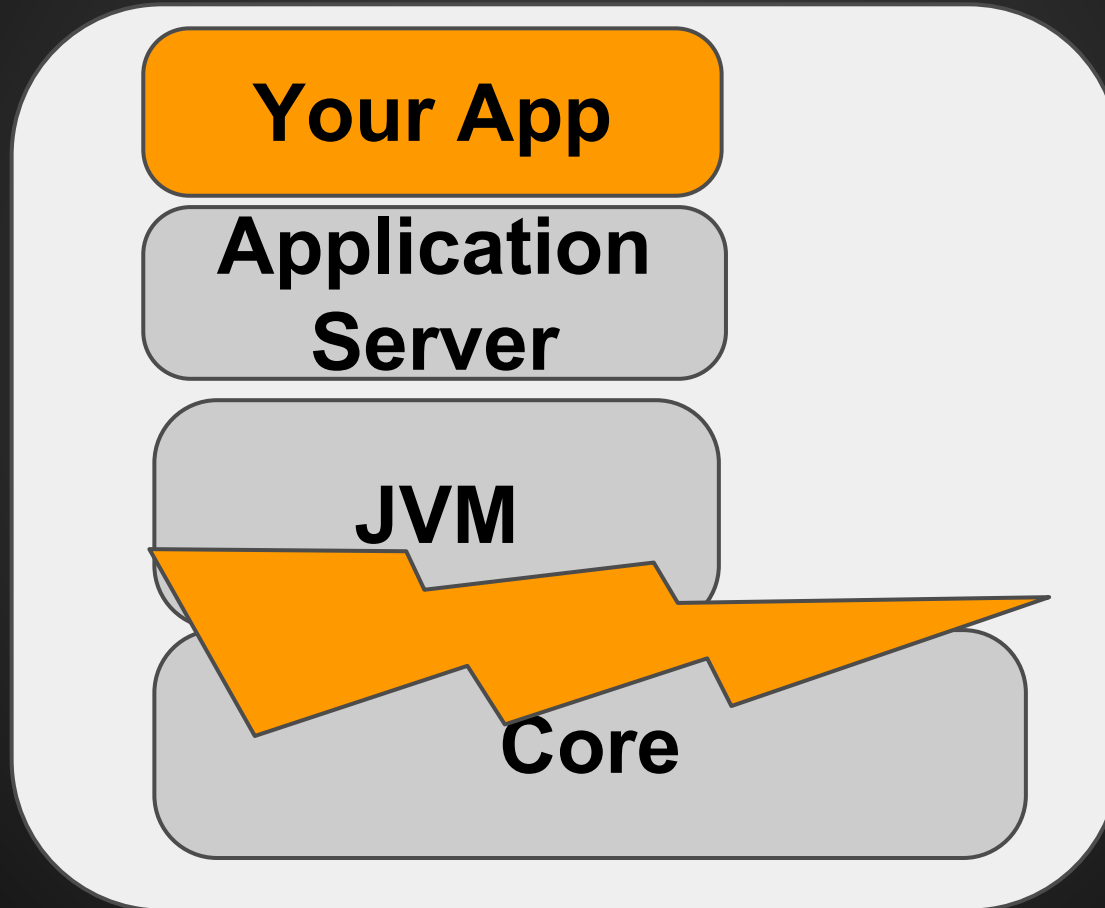
- For specialized applications, bypass the I/O stack completely
- Application consumes data from virtio rings



Integrating the JVM into the kernel

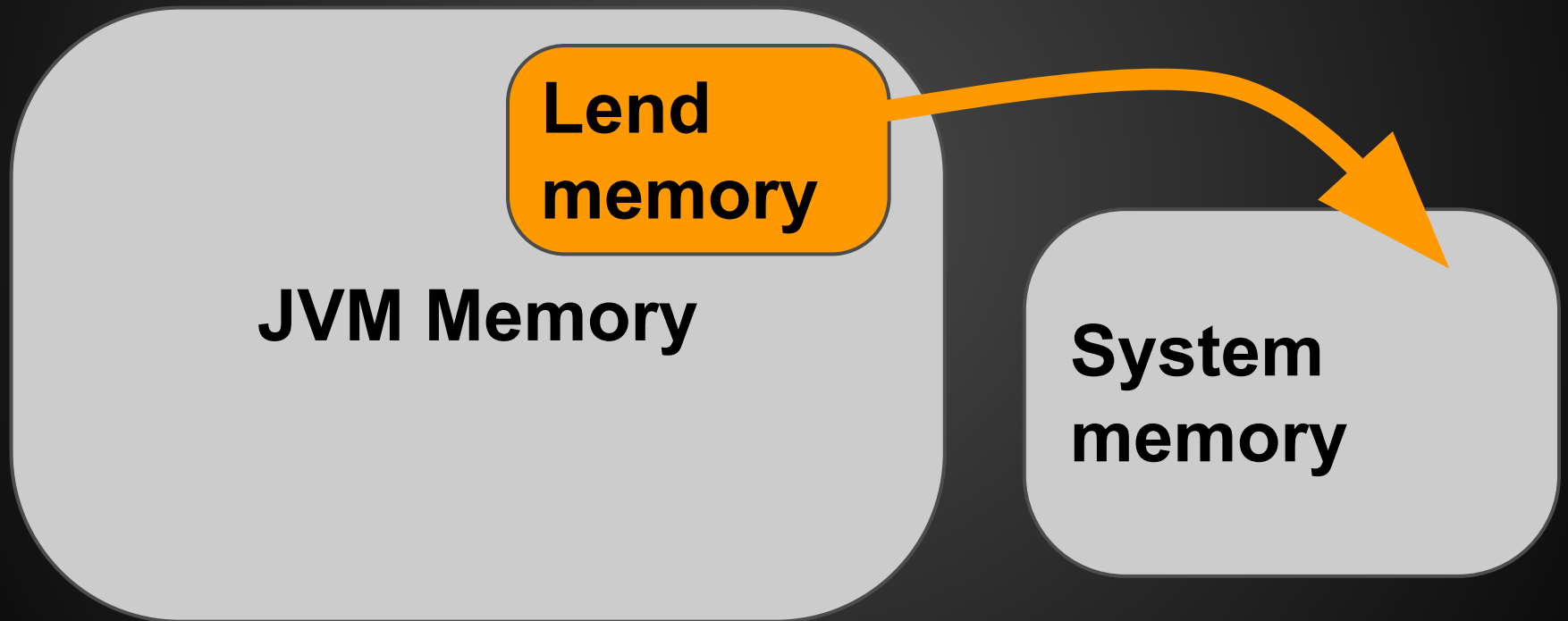
Dynamic
Heap
Memory

TCP in the
JVM + App
context

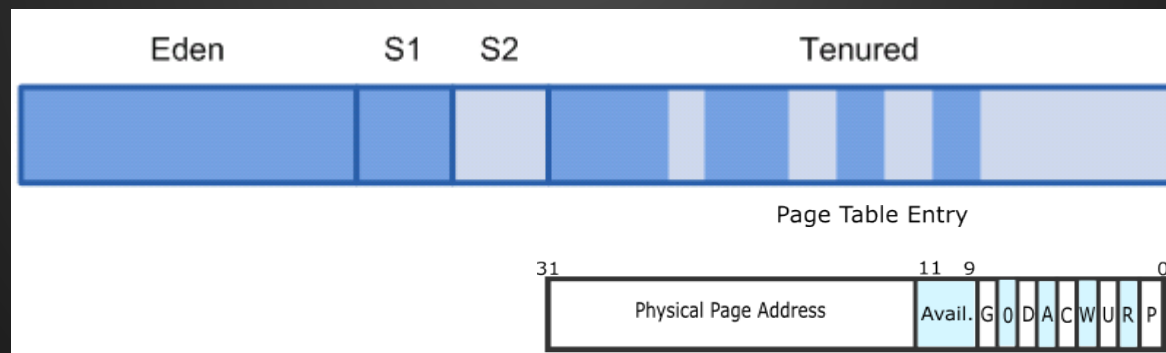
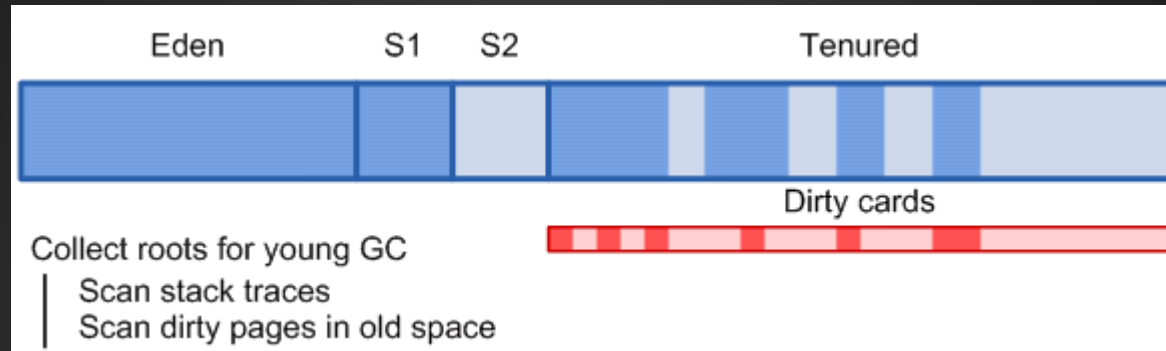


Faster GC

Dynamic heap, sharing is good



Integrating the JVM into the kernel



G - Global
D - Dirty
A - Accessed
C - Cache Disabled
W - Write Through
U - User/Supervisor
R - Read/Write
P - Present

Let's Build A COMMUNITY



nodeJS

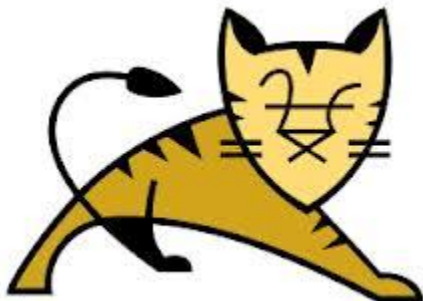


jetty://

Ruby



NGINX



zendServer
Community Edition

Porting a JVM application to OS^v

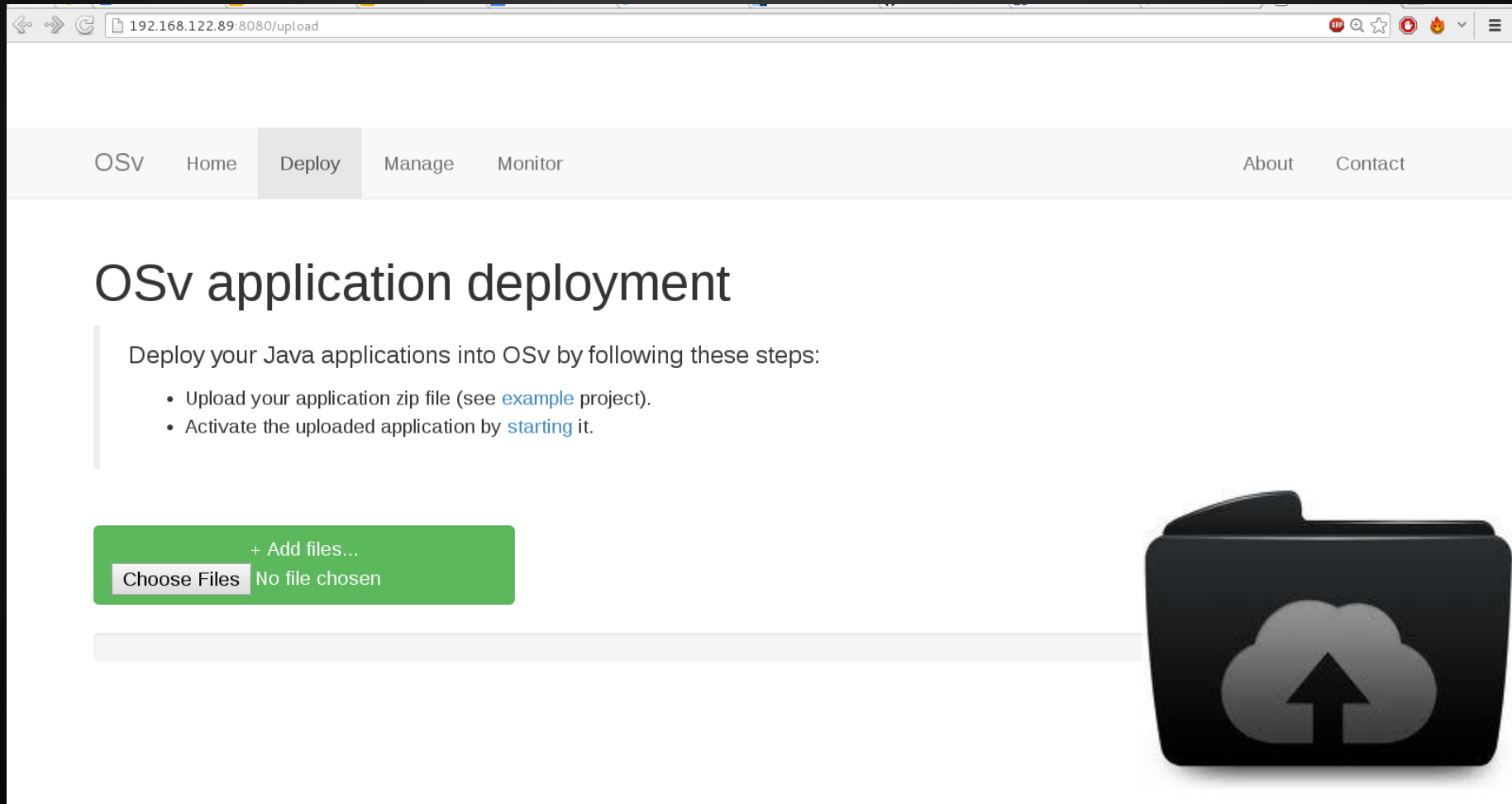
1. Done^{*}

* well, unless the application fork()s

Porting a C/C++ application to OS^V

1. Must be a single-process application.
2. May not fork() or exec().
3. Need to rebuild as a shared object (.so).
4. Other API limitations apply.

Management



Resources



<http://osv.io>



<https://github.com/clouddius-systems/osv>



@ClouddiusSystems



osv-dev@googlegroups.com

OS^v@Clouddius

Cloudivus Systems, OS Comparison

Feature/Property	OS ^v	Traditional OS
Good for:	Machete: Cloud/Virtualization	Swiss knife: anything goes
Typical workload	Single app * VMs	Multiple apps/users, utilities, anything
kernel vs app	Cooperation	distrust
API, compatibility	JVM, POSIX	Any, but versions/releases..
# Config files	0	1000
Tuning	Auto	Manual, requires certifications
Upgrade/state	Stateless, just boots	Complex, needs snapshots, hope..
JVM support	Tailored GC/STW solution	Yet another app
Lines of code	Few	Gazillion
License	BSD	GPL / proprietary