OSv

Dor Laor, Avi Kivity
Cloudius Systems

Glauber Costa

KVM, Containers, Xen

Nadav Har'EL,

Nested KVM

OS^v

Avi Kivity KVM originator

Pekka Enberg,

kvm, jvm, slab

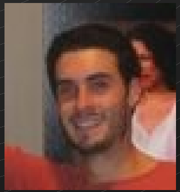
Dor Laor, Former kvm project mngr

Or Cohen

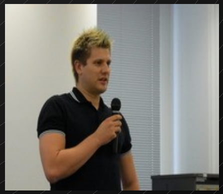
Dmitry Fleytman

Ronen Narkis

Guy Zana



hch



The story so far

In the beginning there was hardware

… and then they added

an application

… and then they added

an operating system

… and then they added

a hypervisor

… and then they added

managed runtime

Notice the pattern?

Typical Cloud Stack

Your App

Application Server

JVM

Operating System

Hypervisor

Hardware

Our software stack
Congealed into existence.

A Historical Anomaly

Your App

Application Server

JVM — provides protection and abstraction

Operating System — provides protection and abstraction

Hypervisor — provides protection and abstraction

Hardware

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	

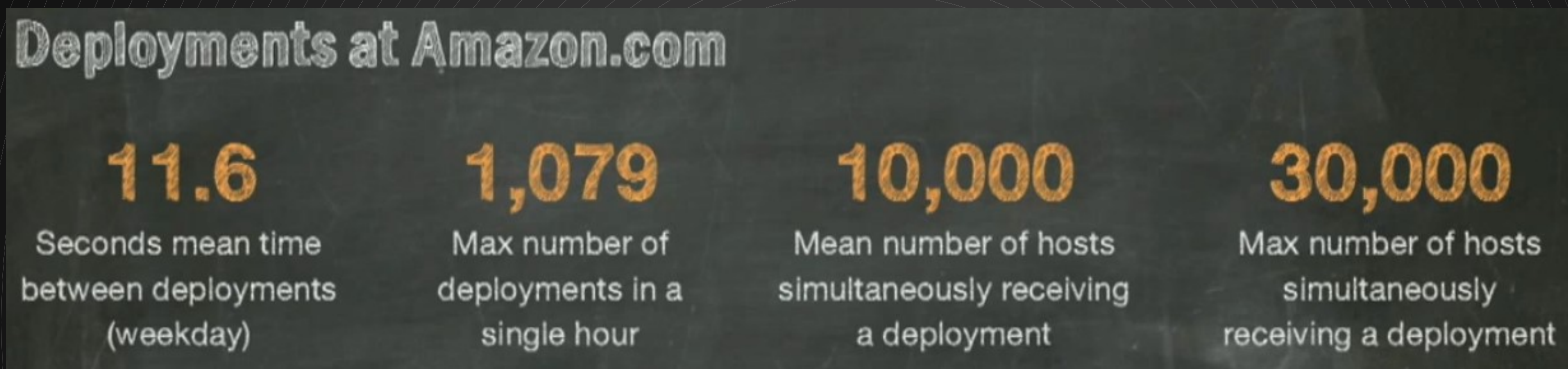
Duplication

Virtualization

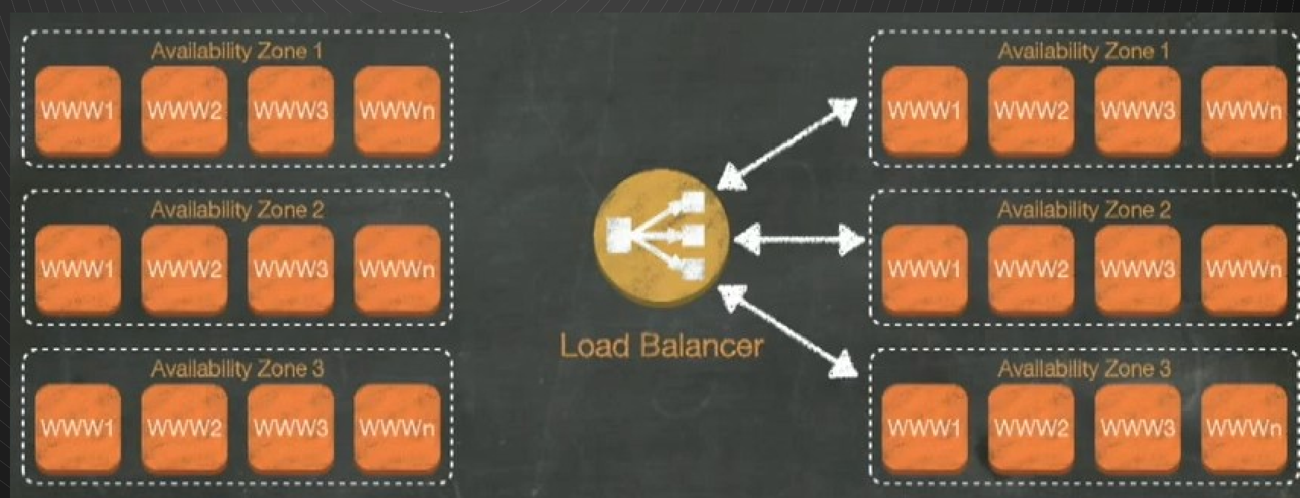
Virtualization 1.0	Virtualization 2.0	Virtualization 2.0, Massive Scale
 Transformed the enterprise from physical2virtual	 Compute node ≠ virtual server	

Virtualization 2.0, Dev/Ops

Virtualization 2.0, agility!



Rolling upgrade within seconds and a fall back option



Virtualization 2.0

Architecture

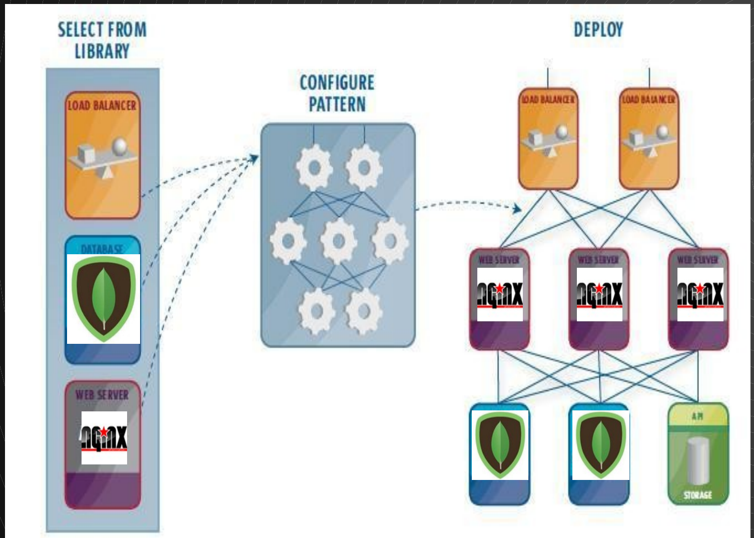


Figure 1. Example cloud-based deployment of an application onto a two-tier Web server architectural pattern.

vServer OS 1.0

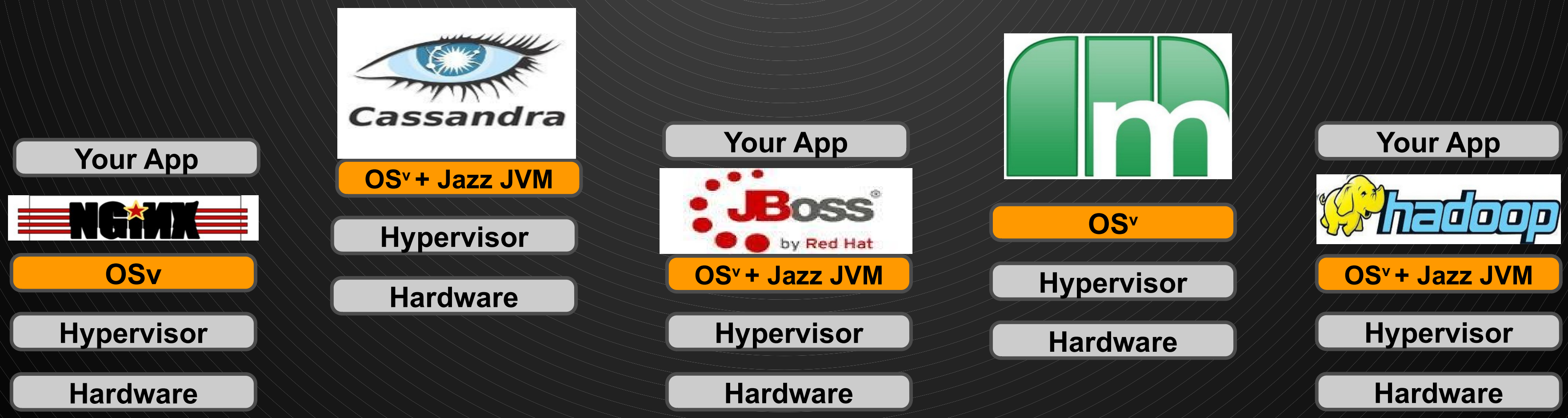
- No Hardware
- No Users
- No app(S)

- Yes Complexity

less is more.

Mission statement

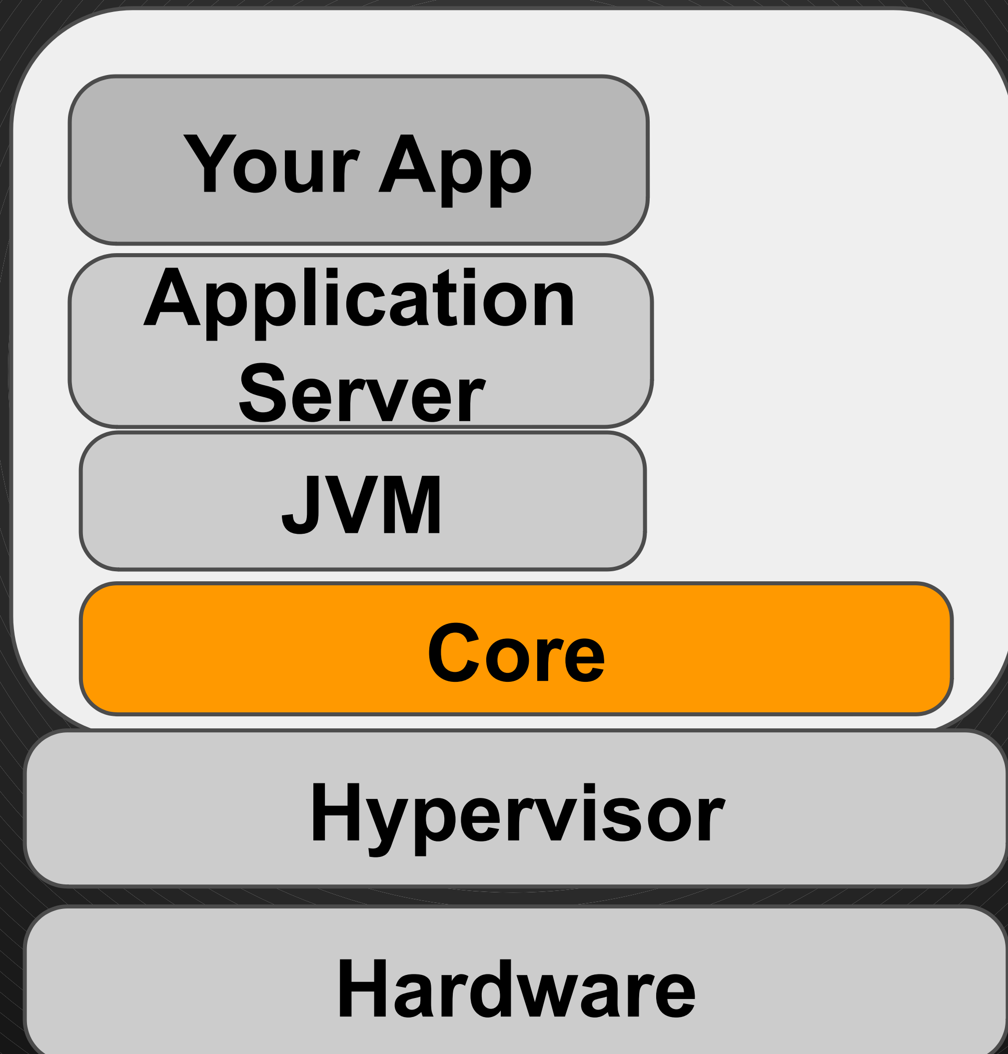
Be the **best OS** powering virtual machines in the **cloud**



The new Cloud Stack - OSv

Single Process

Kernel space only



Linked to existing JVMs

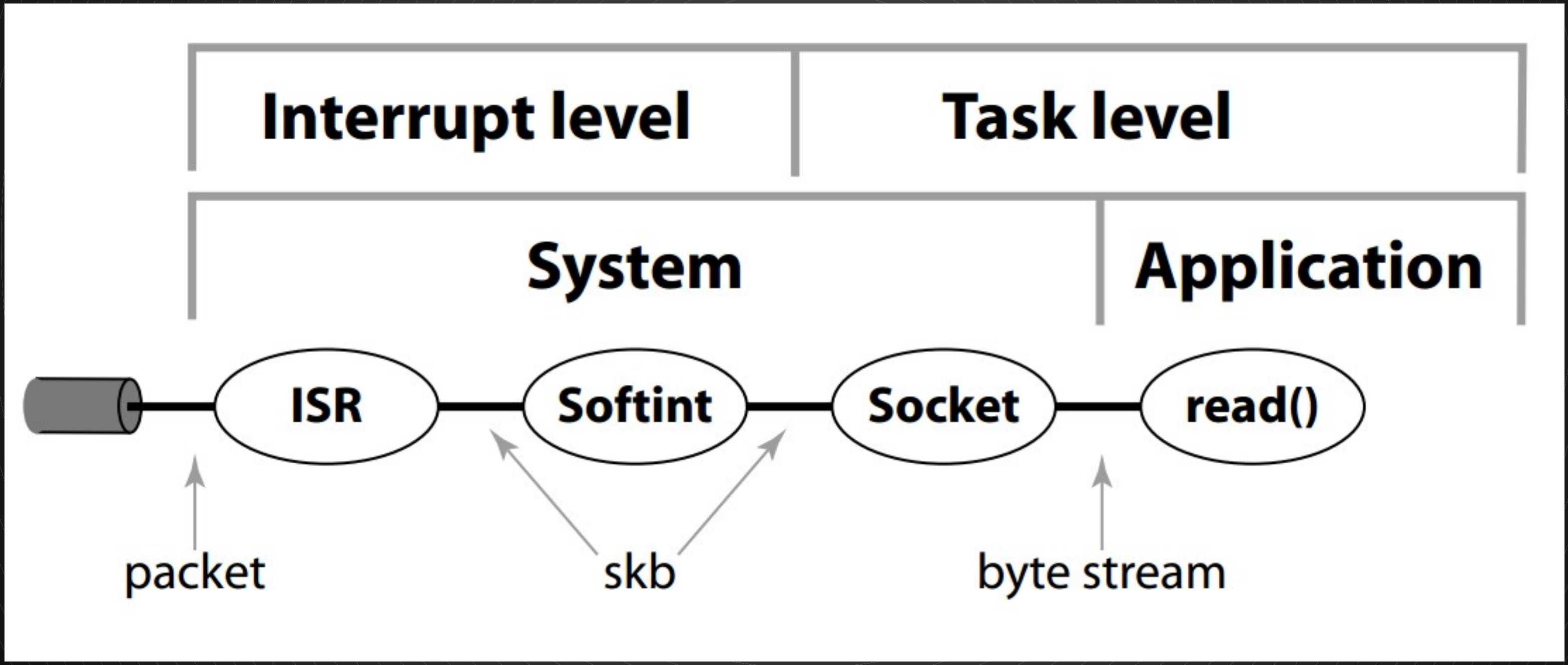
App sees no change

The new Cloud Stack - OSv

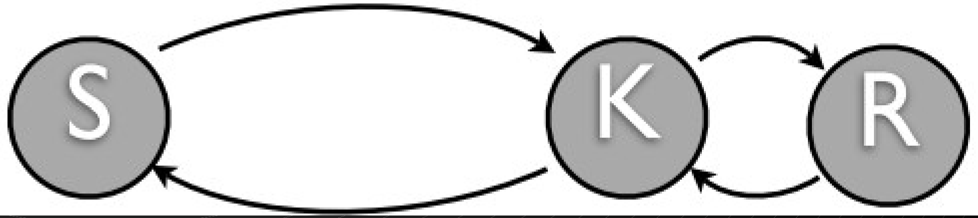
Memory	Huge pages, Heap vs Sys
I/O	Zero copy, full aio, batching
Scheduling	Lock free, low latency
Tuning	Out of the box, auto
CPU	Low cost ctx, Direct signals,..

Van Jacobson == TCP/IP

Common kernel network stack

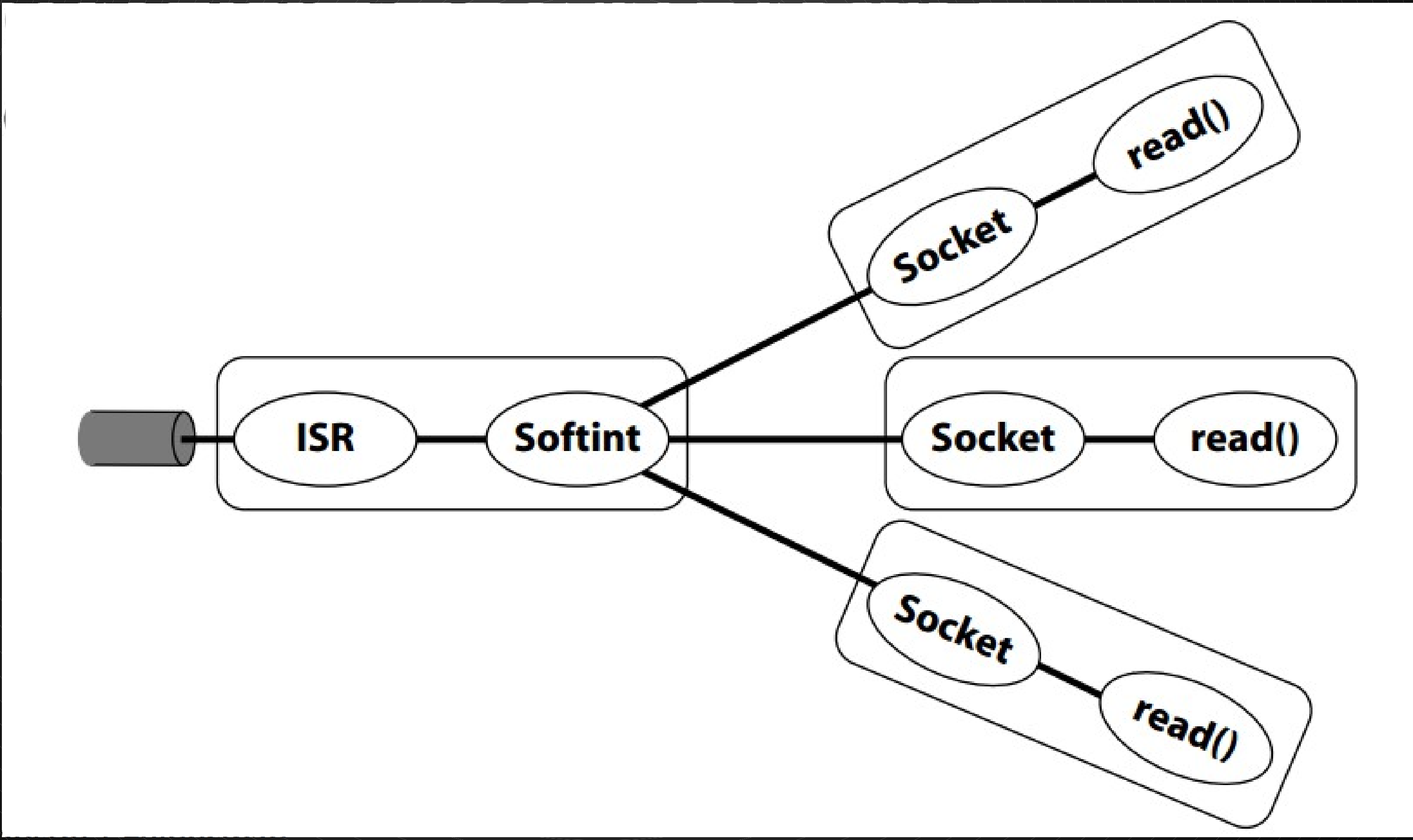


Leads to servo-loop:

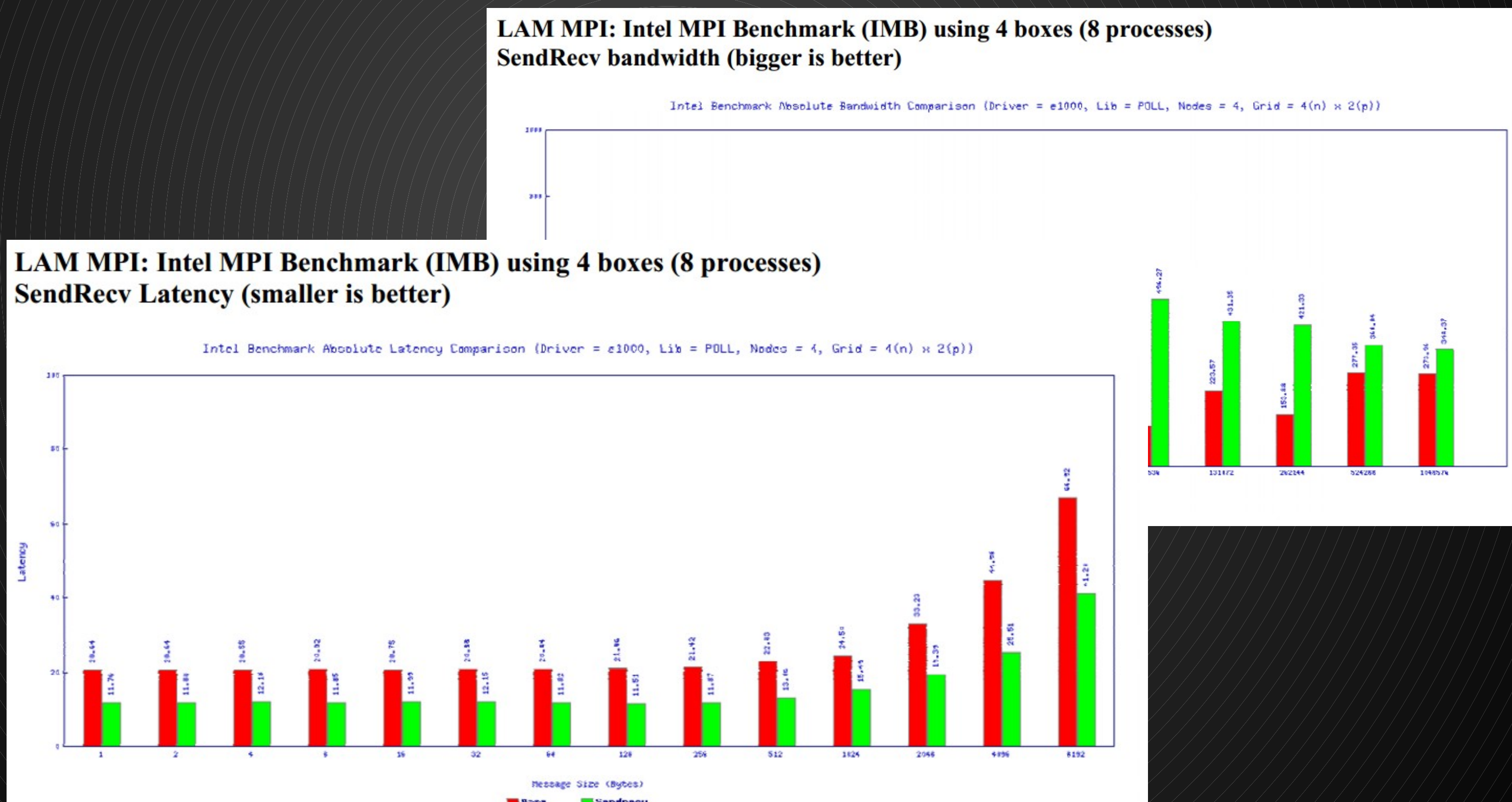


Van Jacobson == TCP/IP

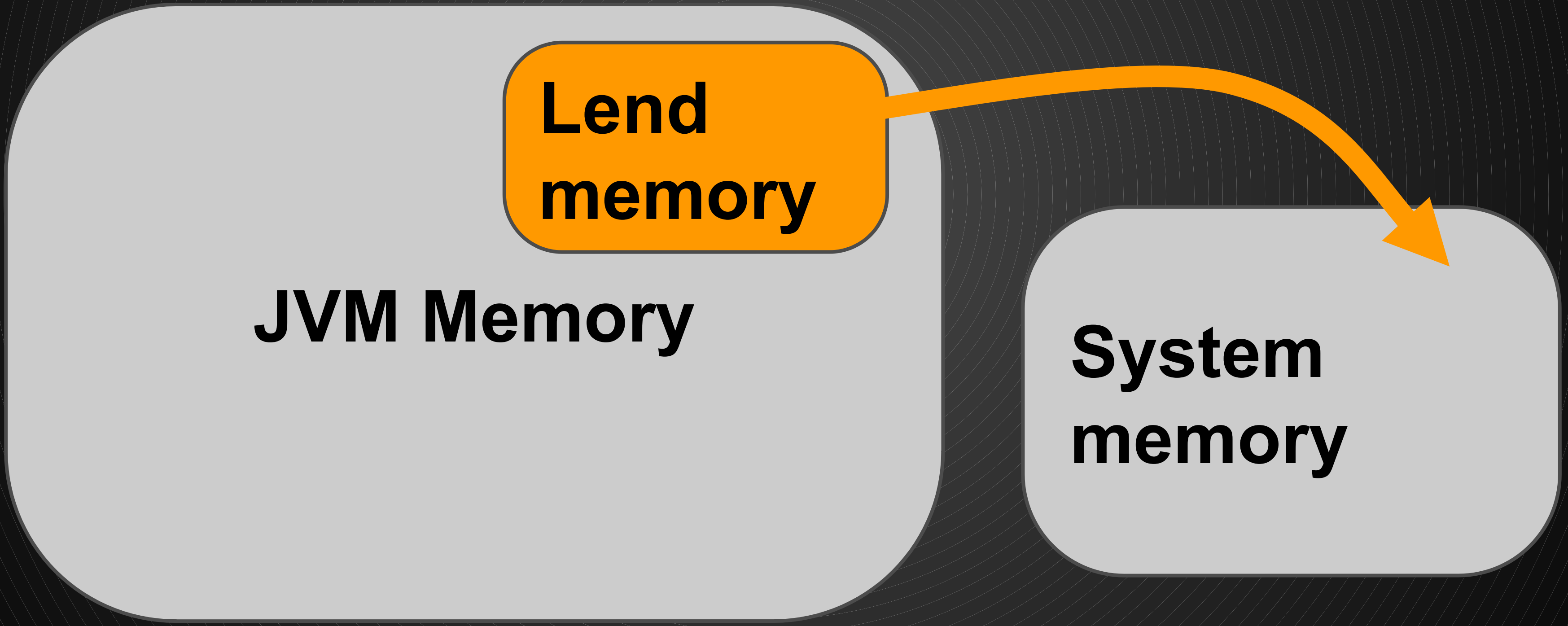
Net Channel design:



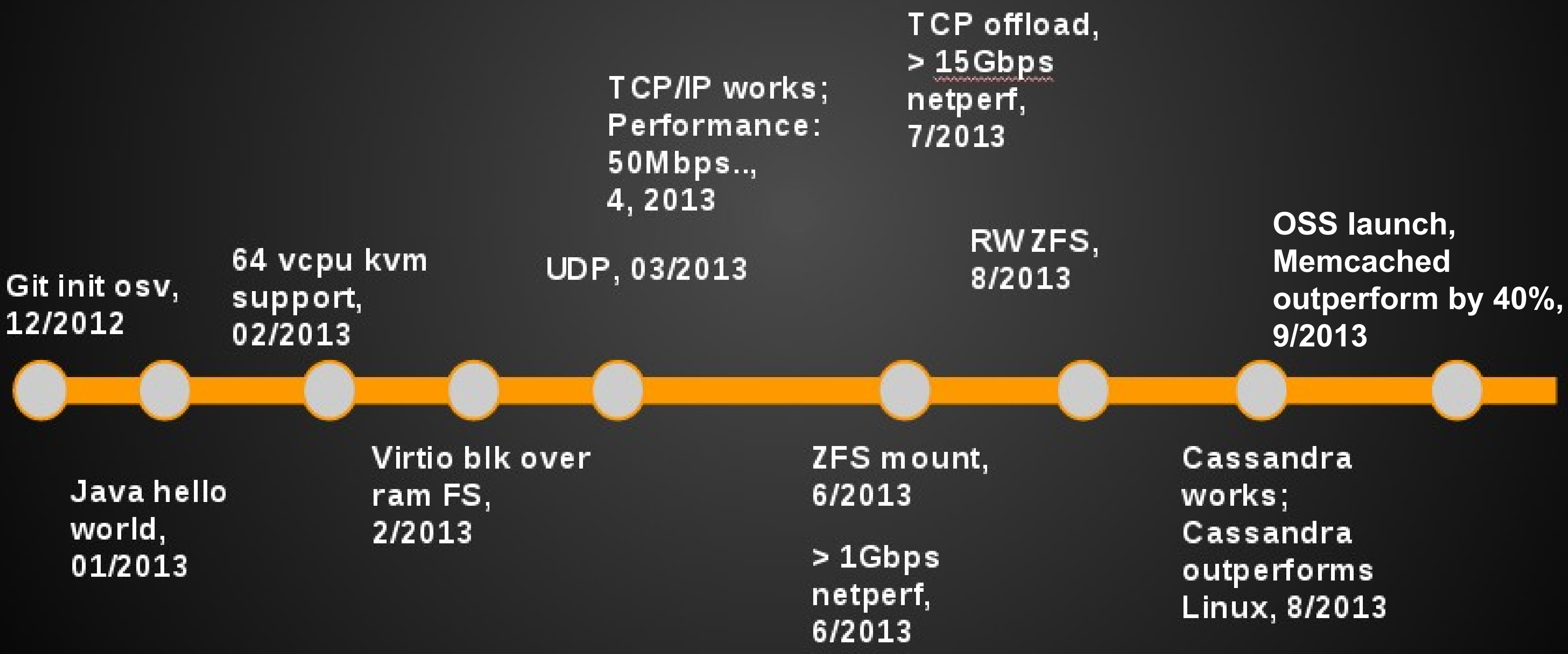
Van Jacobson == TCP/IP



Dynamic heap, sharing is good



Milestones


Git init osv, 12/2012
Java hello world, 01/2013
64 vcpu kvm support, 02/2013
Virtio blk over ram FS, 2/2013
UDP, 03/2013
TCP/IP works; Performance: 50Mbps.., 4, 2013
ZFS mount, 6/2013
> 1Gbps netperf, 6/2013
TCP offload, > 15Gbps netperf, 7/2013
RW ZFS, 8/2013
Cassandra works; Cassandra outperforms Linux, 8/2013
OSS launch, Memcached outperform by 40%, 9/2013

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
- ZFS filesystem
- Huge pages from the very beginning

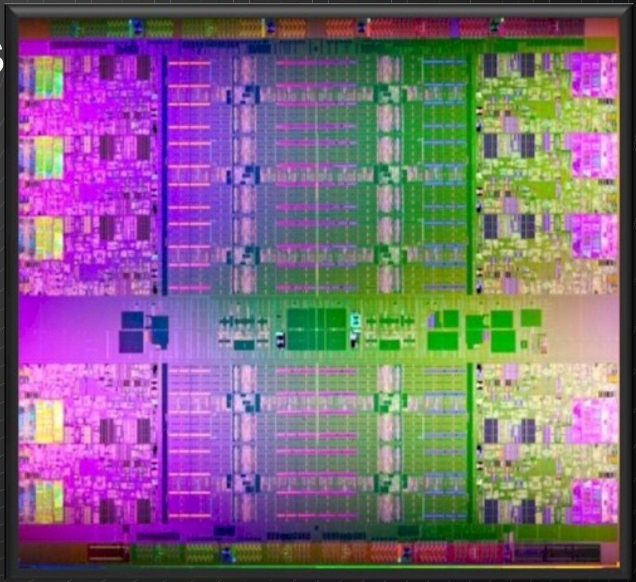
Open Source

Fork me on GitHub

- These days, credibility == open source
- Looking for cooperation:
  - Kernel-level developers
  - Management stack
  - Dev/ops workflow
- BSD-style license

Architecture ports

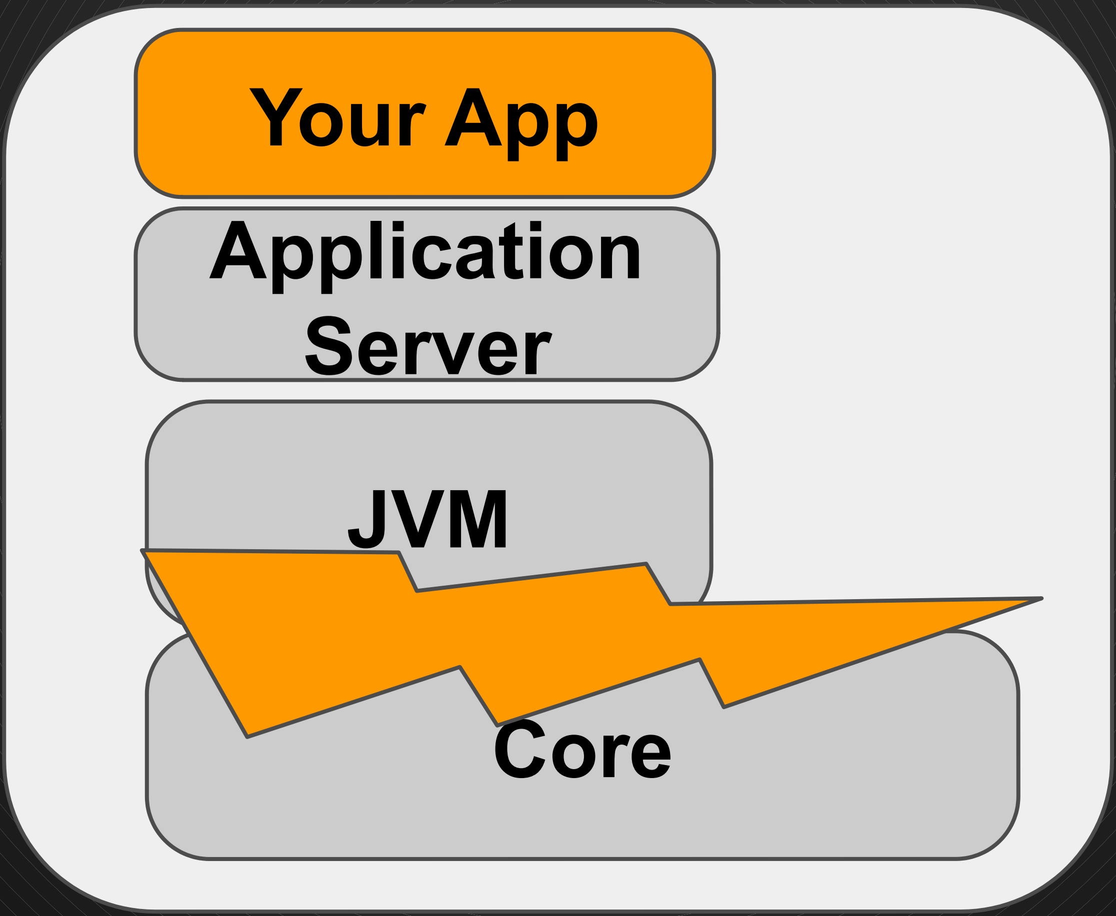
- 64-bit x86
  - KVM - running like a bat out of hell
  - Xen HVM - running (still slow :-( )
  - Xen PV - in progress
  - VMware - planned in 2 months
- 64-bit ARM - planned
- Others - patches welcome



Integrating the JVM into the kernel

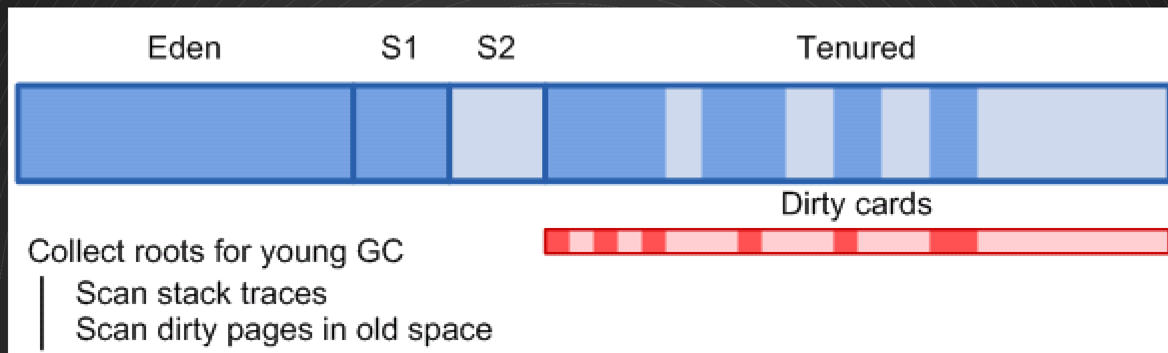
Dynamic Heap Memory

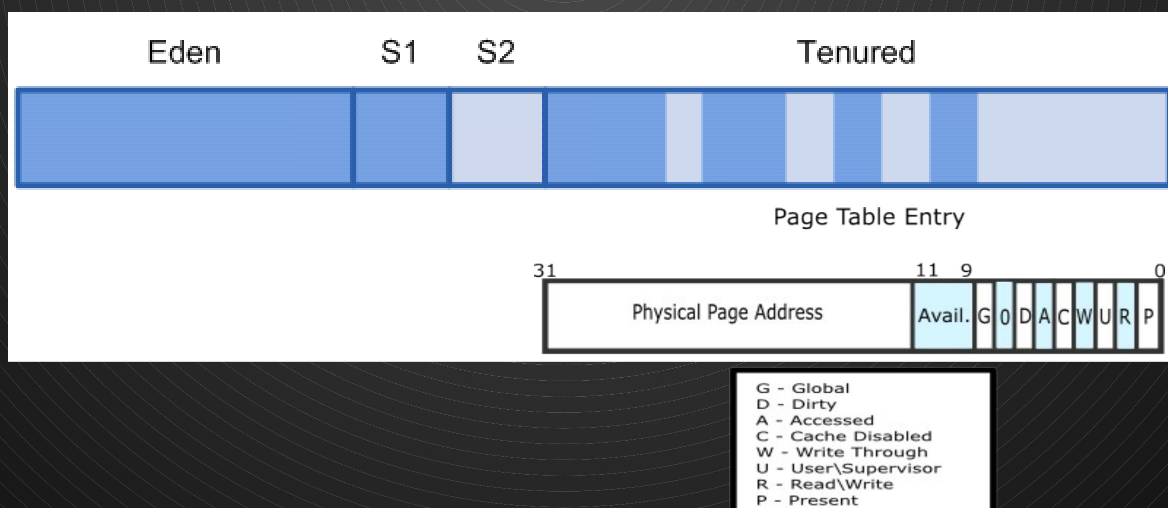
TCP in the JVM + App context



Fast inter thread wakeup

Integrating the JVM into the kernel





Technical deep dive

- C++
- Idle time polling
- Performance and tracing
- Virtio-app

C++

```
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;
    }
}
```

Idle-time polling

- Going idle is **much** more expensive on virtual machines
- So are inter-processor interrupts - IPIs
- Combine the two:
  - Before going idle, **announce** it via shared memory
  - **Delay** going idle
  - In the meanwhile, **poll** for wakeup requests from other processors
- Result: wakeups are faster, both for the processor waking, and for the wakee

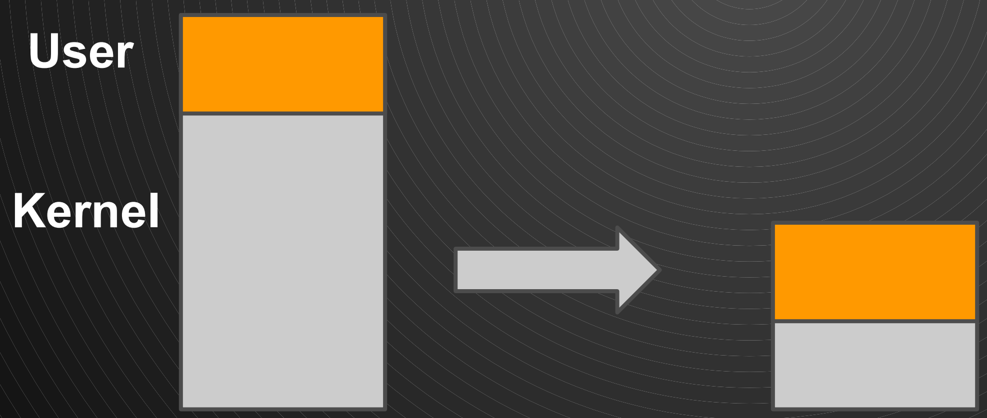
Performance and tracing

```
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
```

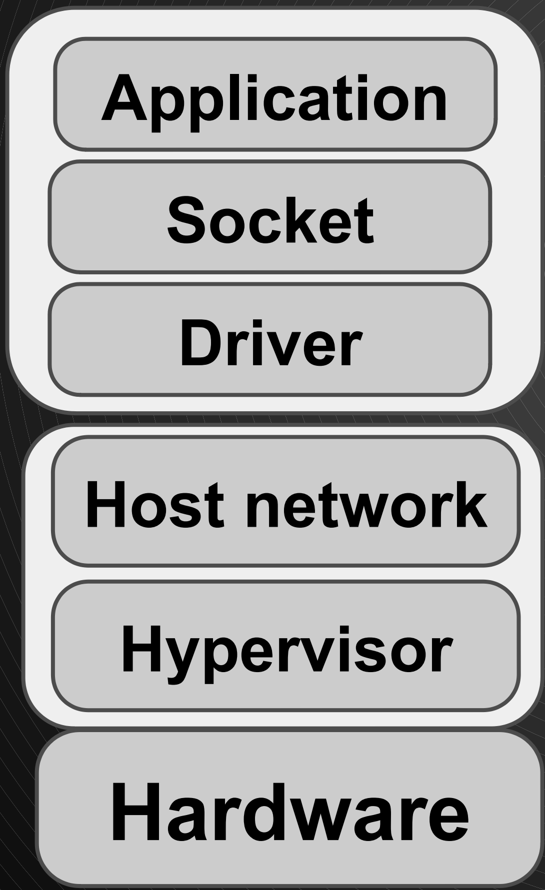
Virtio-app || Data plane

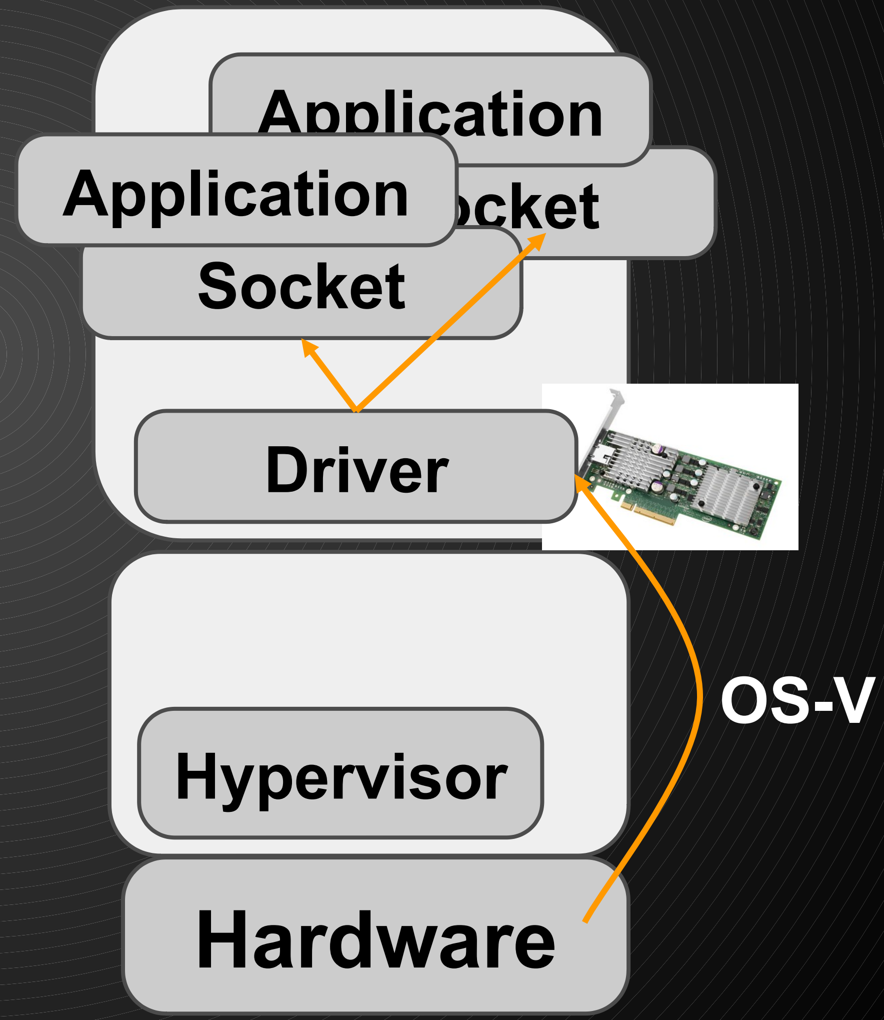
- For specialized applications, bypass the I/O stack completely
- Application consumes data from virtio rings



OSv at the cutting edge front

Traditional





OS^v at the cutting edge front

- Transactional Memory (lock elision)
  Better architecture match with
  higher transaction/sec and less contention

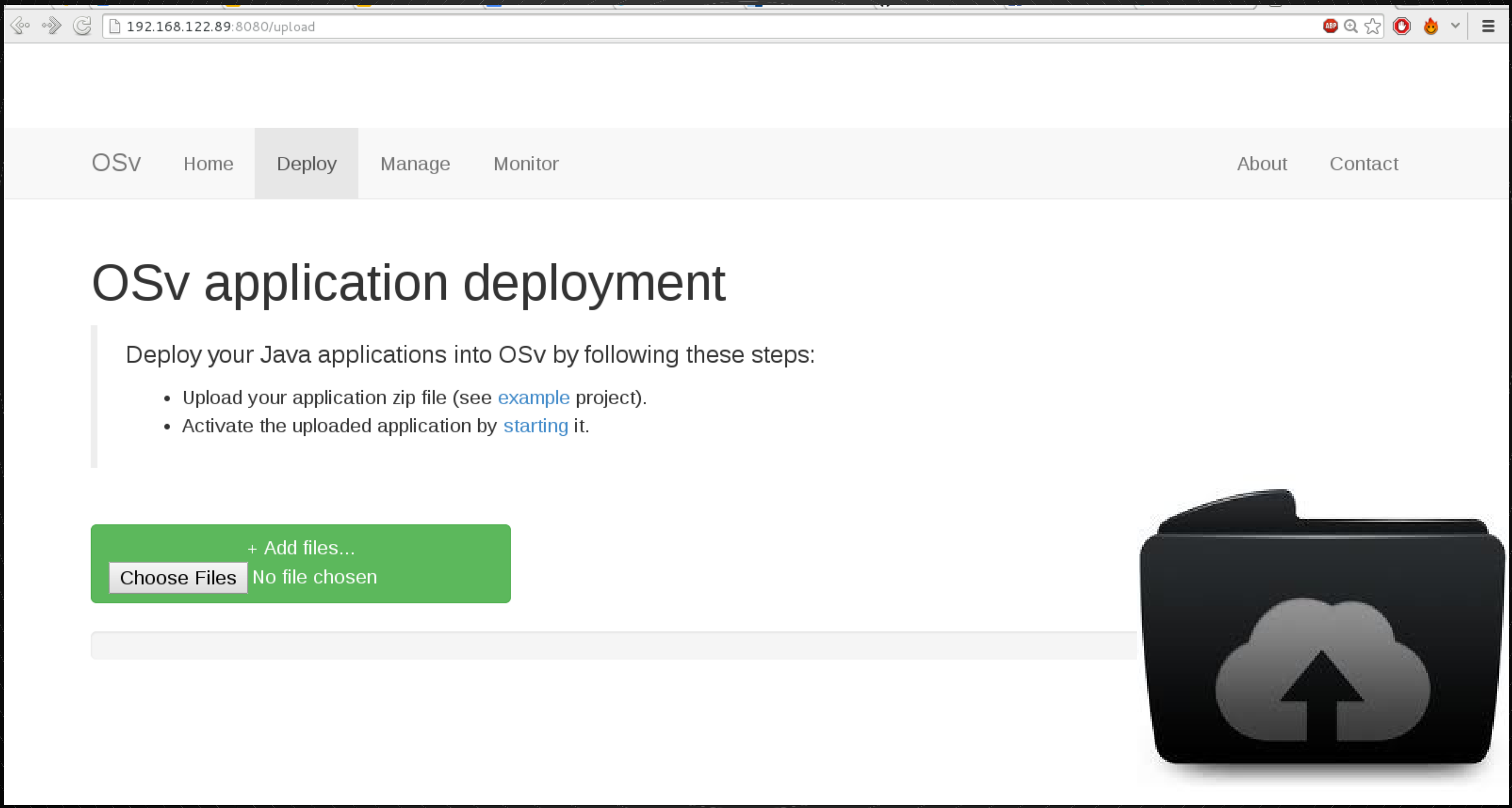
- Perfect match with NVRam abundance
  In the near future we'll see NVRam reaches mainstream adoption. The importance of traditional filesystems will decrease, applications will manage their IO directly using NVRam

OS that doesn't get in the way

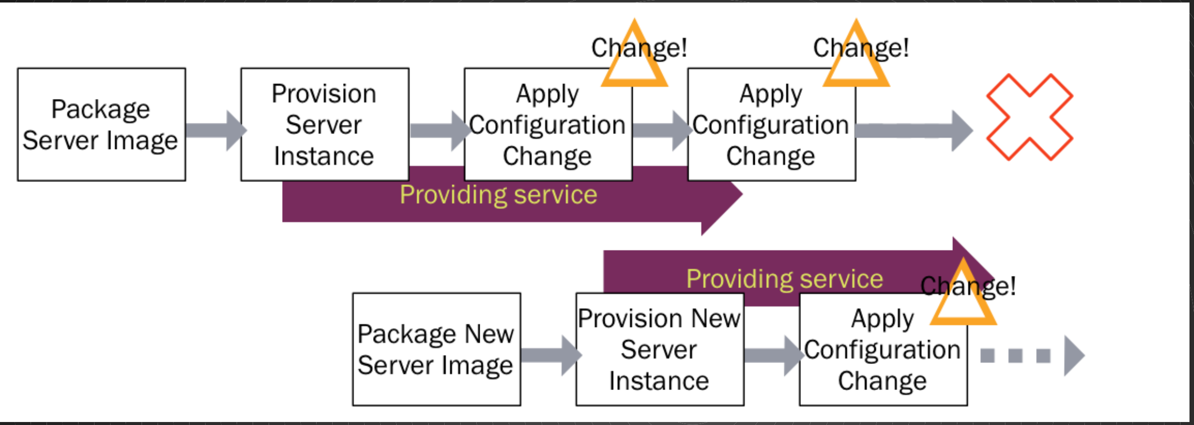
NO Tuning
NO State
NO Patching
✗ 4 VMs per sys admin ratio

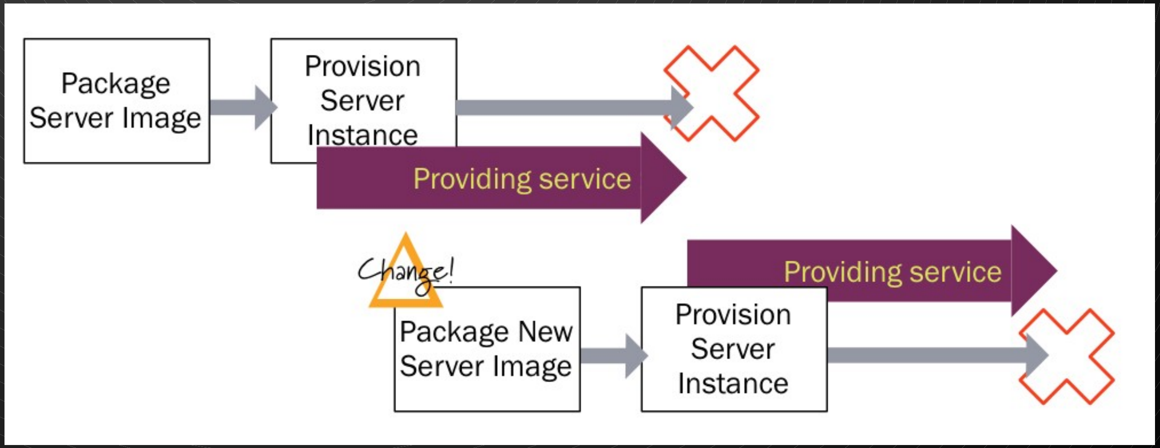
http://www.computerworld.com.au/article/352635/there_best_practice_server_system_administrator_ratio_/

Management


192.168.122.89:8080/upload
OSv
Home
Deploy
Manage
Monitor
About
Contact
OSv application deployment
Deploy your Java applications into OSv by following these steps:
Upload your application zip file (see example project).
Activate the uploaded application by starting it.
+ Add files...
Choose Files
No file chosen

Virtualization 2.0: Stateless servers





Let’s Build A COMMUNITY

Porting a JVM application to OSv

1. Done*

* well, unless the application fork()s

Porting a 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

Resources

http://osv.io

https://github.com/cloudius-systems/osv

@CloudiusSystems

osv-dev@googlegroups.com

OS^v@Cloudius

Cloudius 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