Lessons learned from OS^{V}

Avi Kivity, Glauber Costa
Cloudius Systems

Agenda

Introduction to OS^V

Why C++ for systems programming?

Examples

QEMU and OS^{V} requirements

Glauber Costa

KVM, Containers, Xen

Nadav Har’EL,

Nested KVM

OS^{v}

Avi Kivity KVM originator

Pekka Enberg,

kvm, jvm, slab

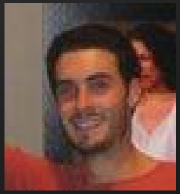
Dor Laor, Former kvm project mngr

Or Cohen

Dmitry Fleytman

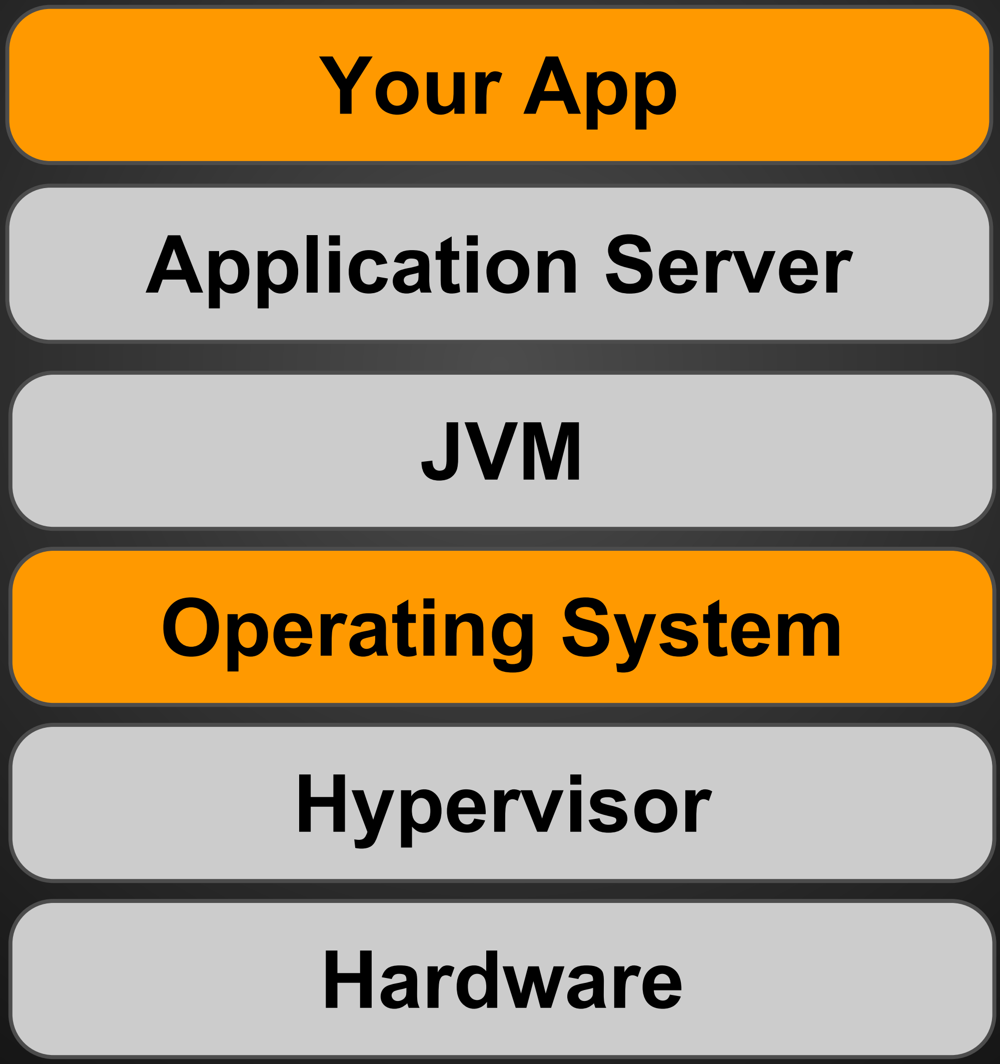
Ronen Narkis

Guy Zana



hch

Typical Cloud Stack



Your App

Application Server

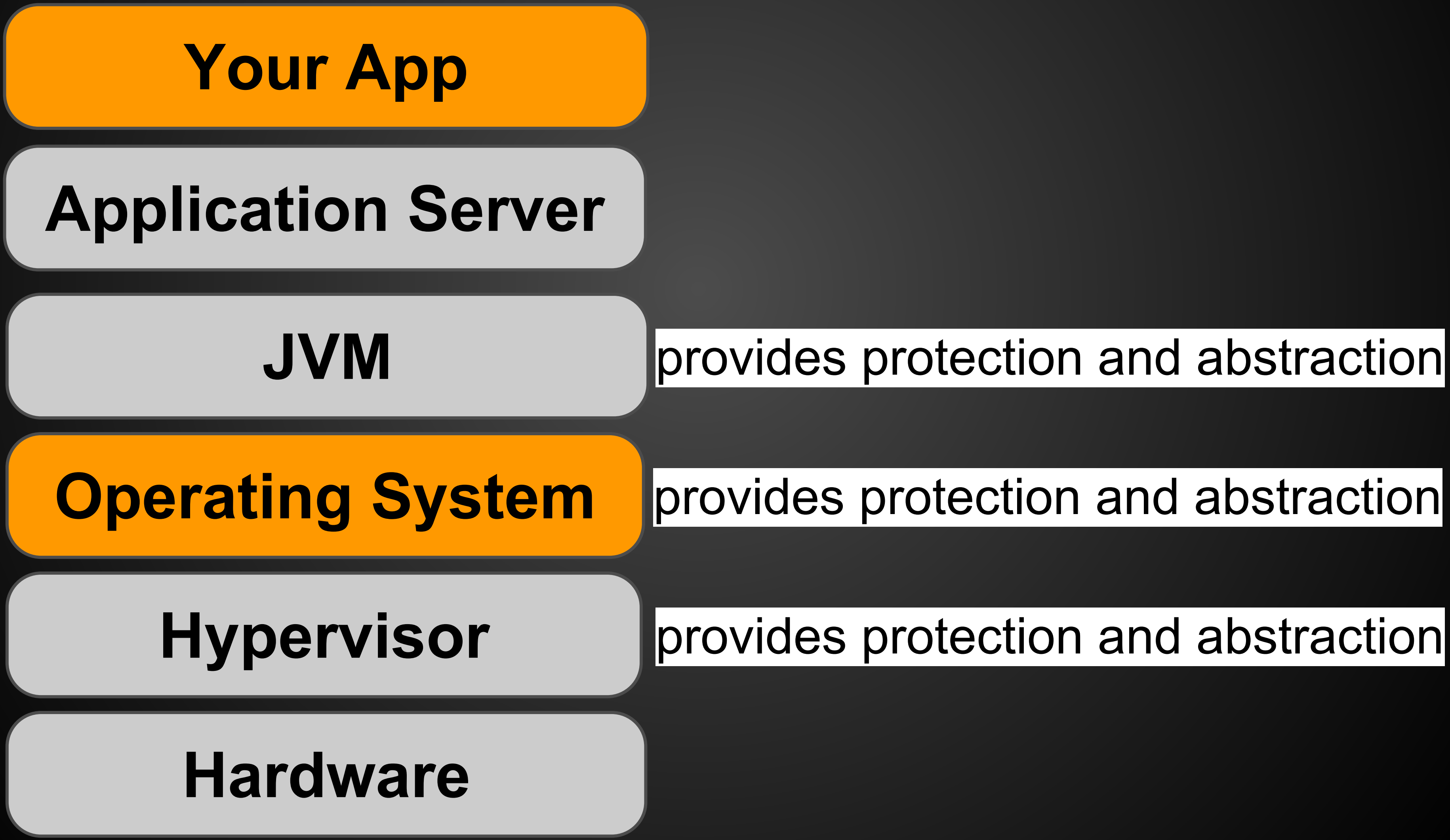
JVM

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

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	

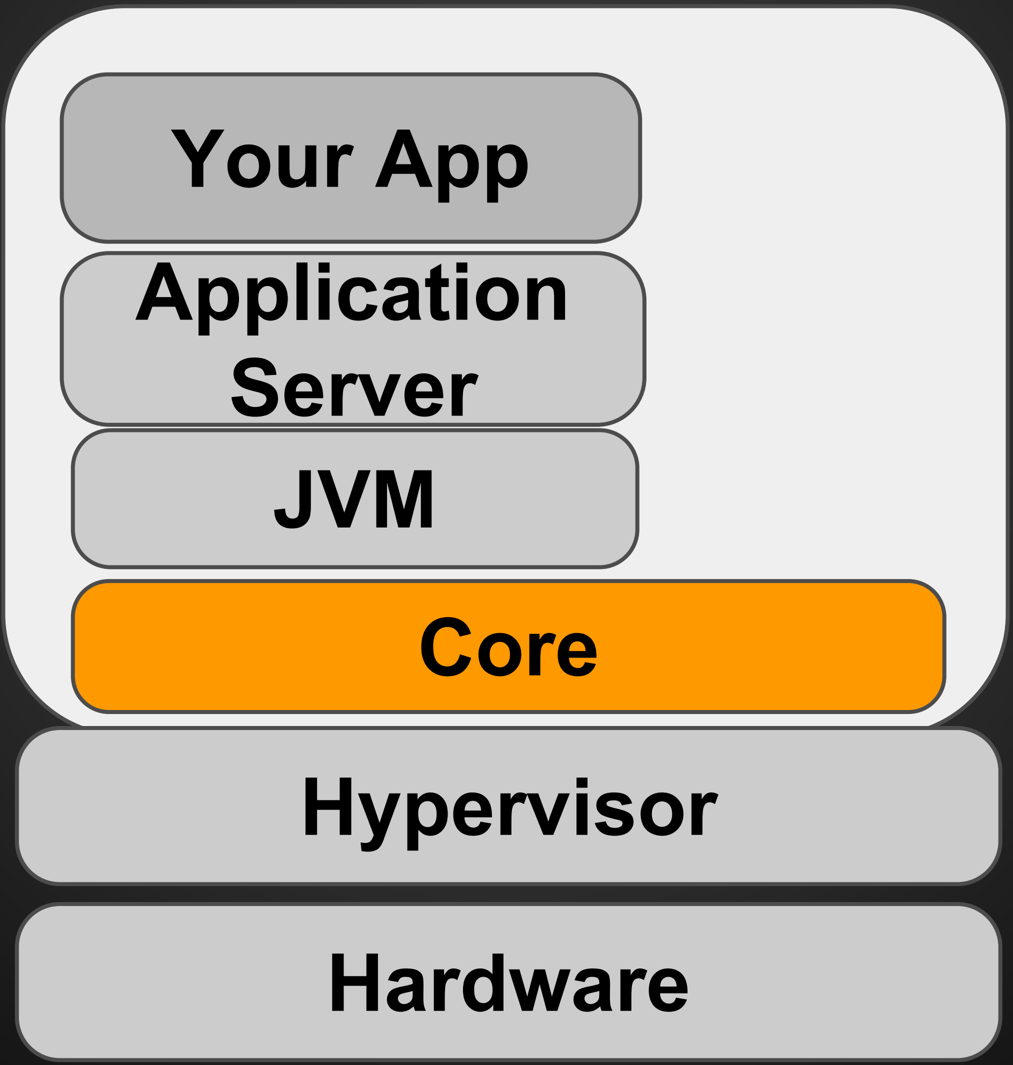
Duplication

less is more.

The new Cloud Stack - OS^v

Single Process

Kernel space only



Linked to existing JVMs

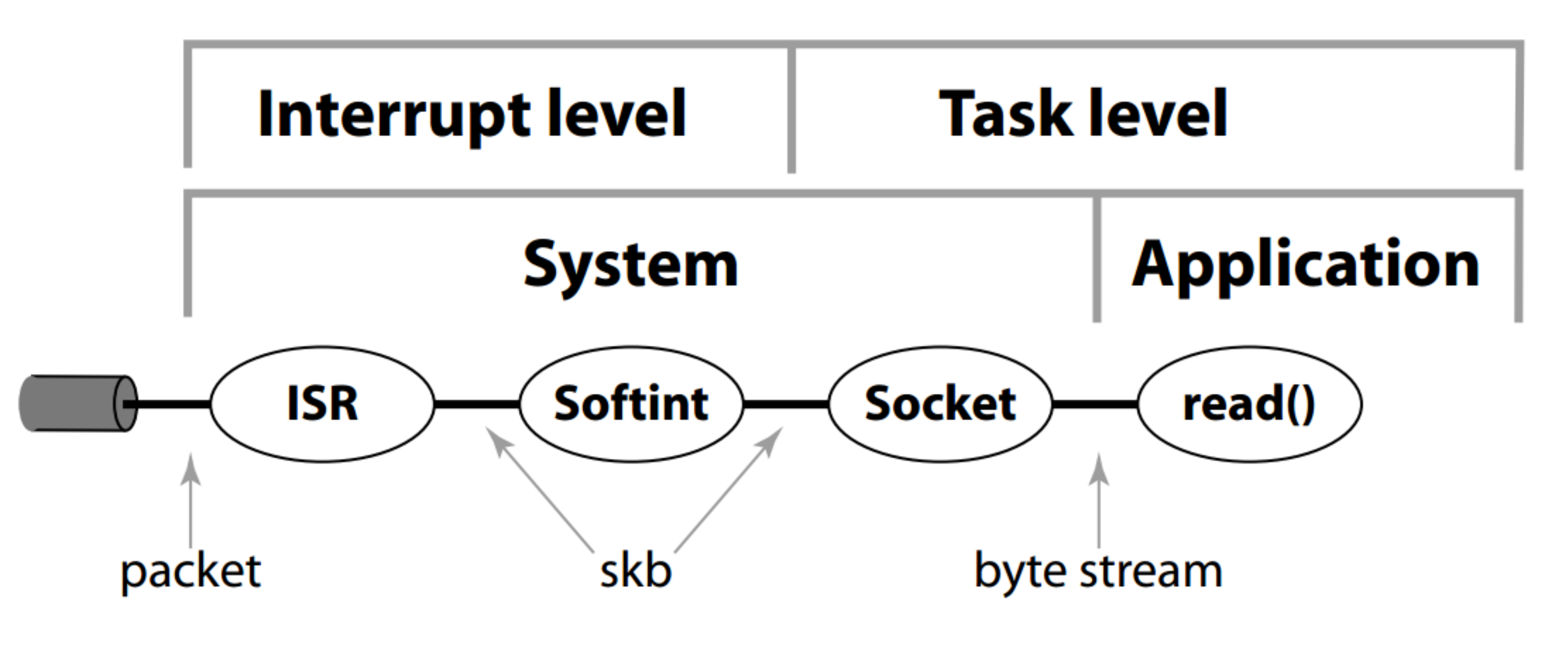
App sees no change

The new Cloud Stack - OS^v

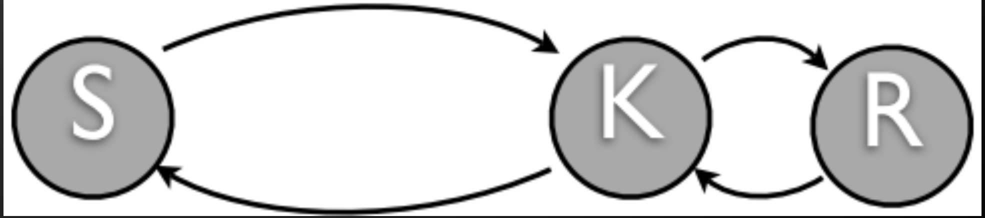
Memory	Huge pages, Heap vs Sys
I/O	Zero copy, full aio, batching
Scheduling	Lock free, low latency
Tuning	Out of the box, auto
CPU	Low cost ctx, Direct signals,..

Van Jacobson == TCP/IP

Common kernel network stack

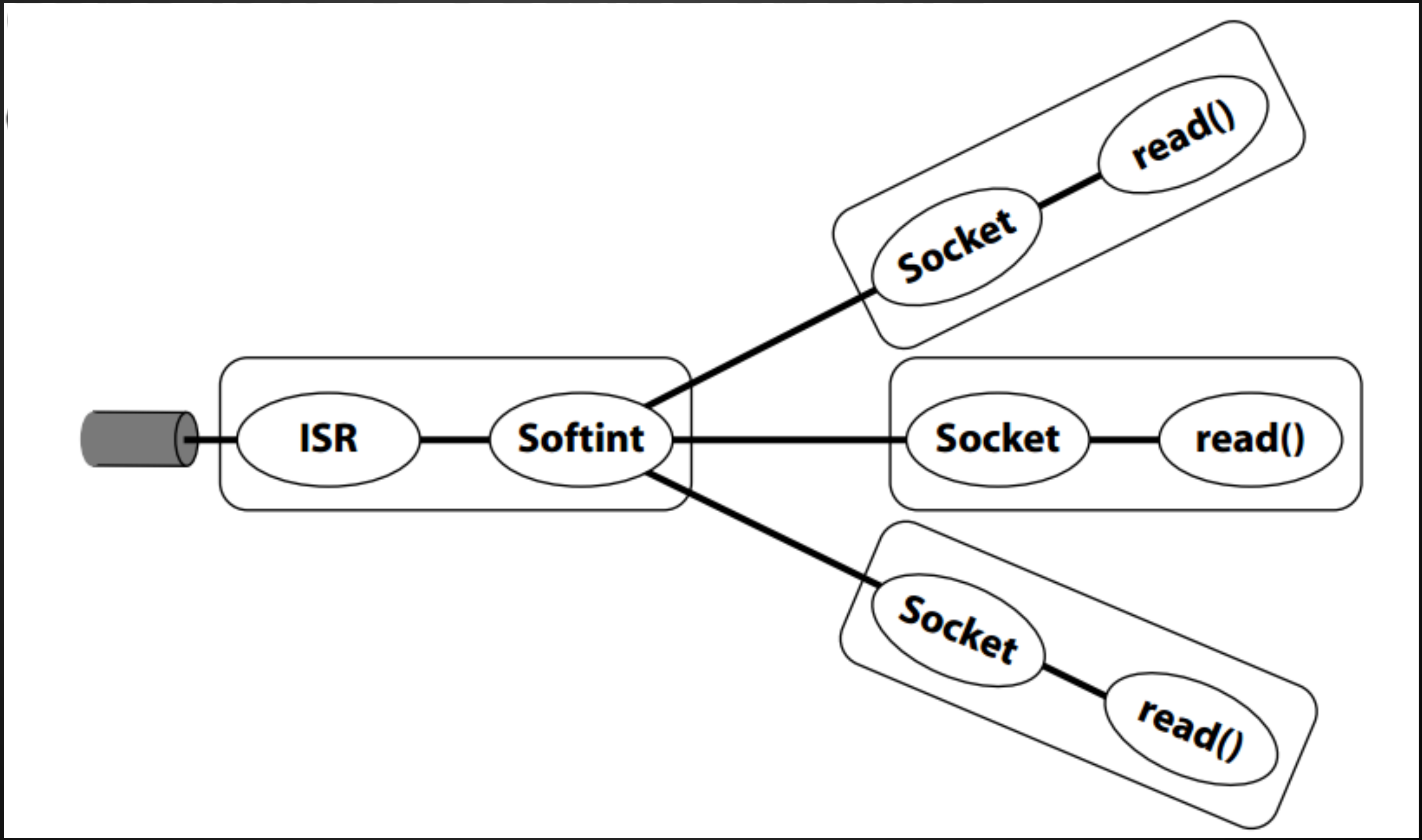


Leads to servo-loop:

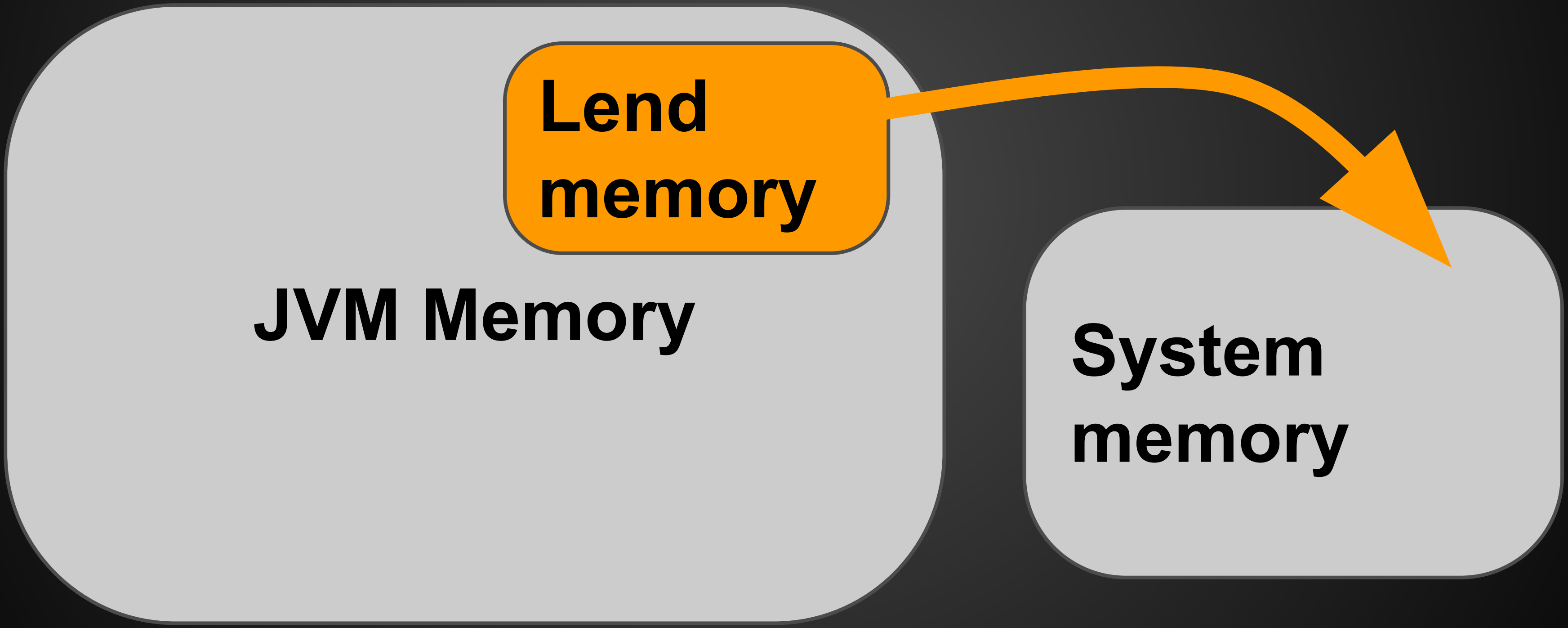


Van Jacobson == TCP/IP

Net Channel design:



Dynamic heap, sharing is good



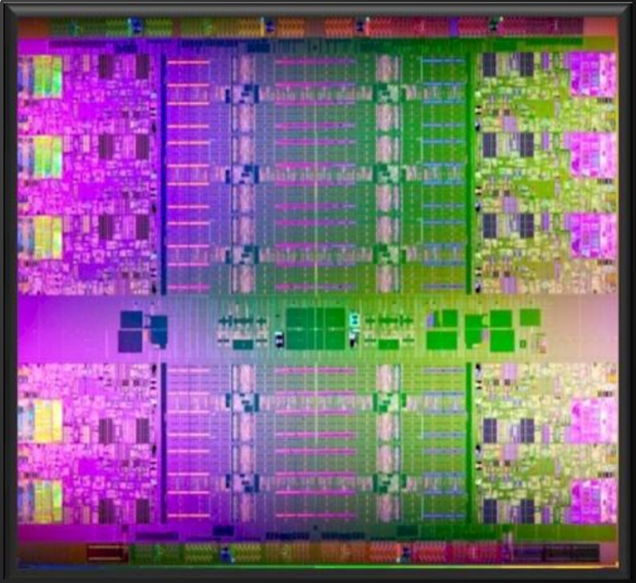
Lend memory

JVM Memory

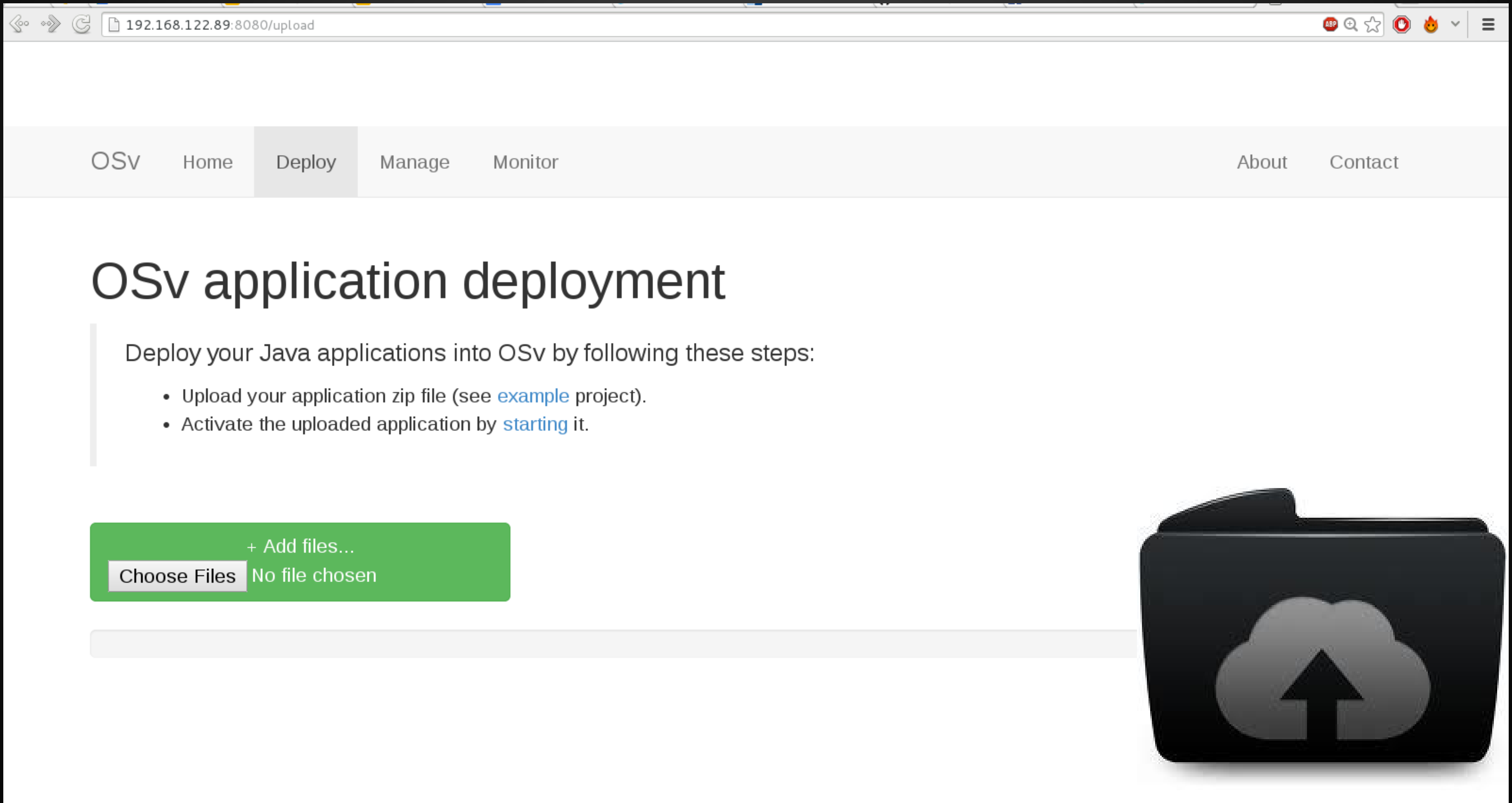
System memory

Architecture ports

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



Management



OSv application deployment

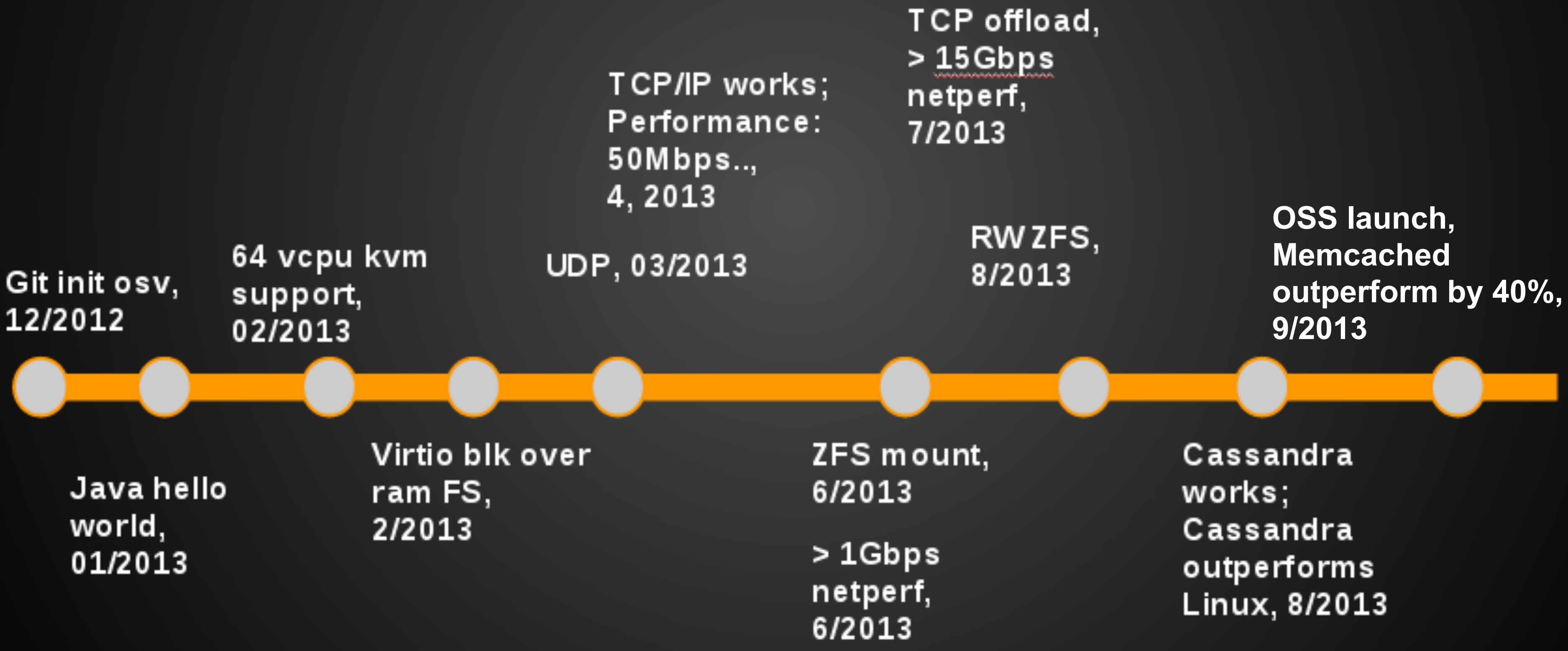
Deploy your Java applications into OSv by following these steps:

- Upload your application zip file (see example project).
- Activate the uploaded application by starting it.

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

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

Two languages called C++

1. Strongly typed object oriented language specialized in leveraging synergies within business process for on demand needs of global companies in a dynamic paradigm shift

Two languages called C++

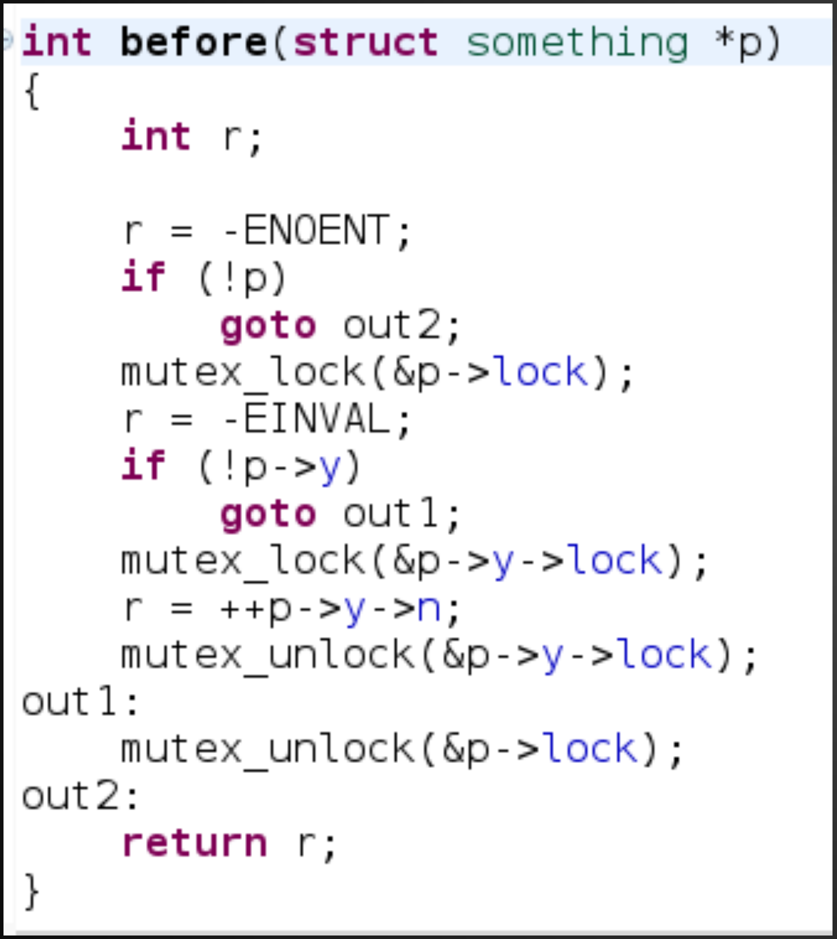
2. A macro language for generating C

Two languages called C++

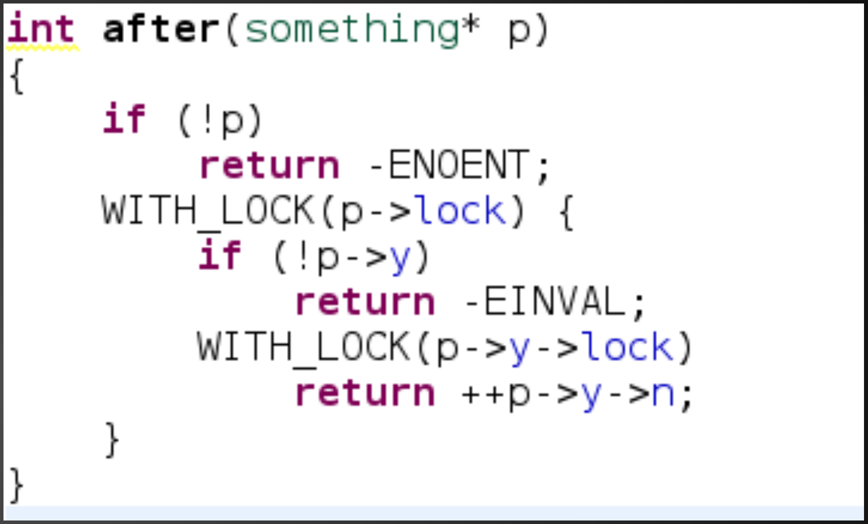
2. A macro language for generating C

- An elaborate macro language
  - Reduce boilerplate
  - Reduce C macros
  - More libraries, reuse
  - Less duplication
- Let the compiler write your C code

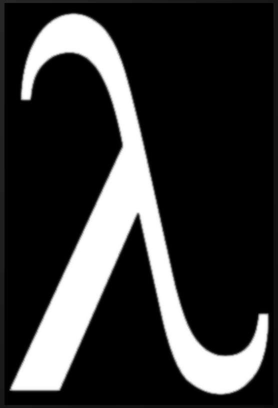
Scoped locking



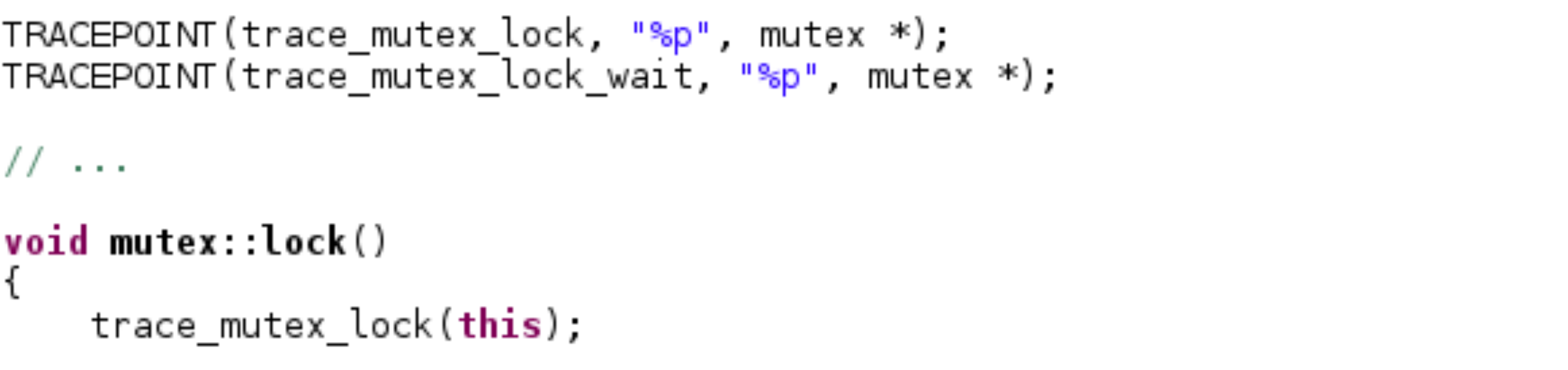
```
int before(struct something *p)
{
    int r;

    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;
}
```



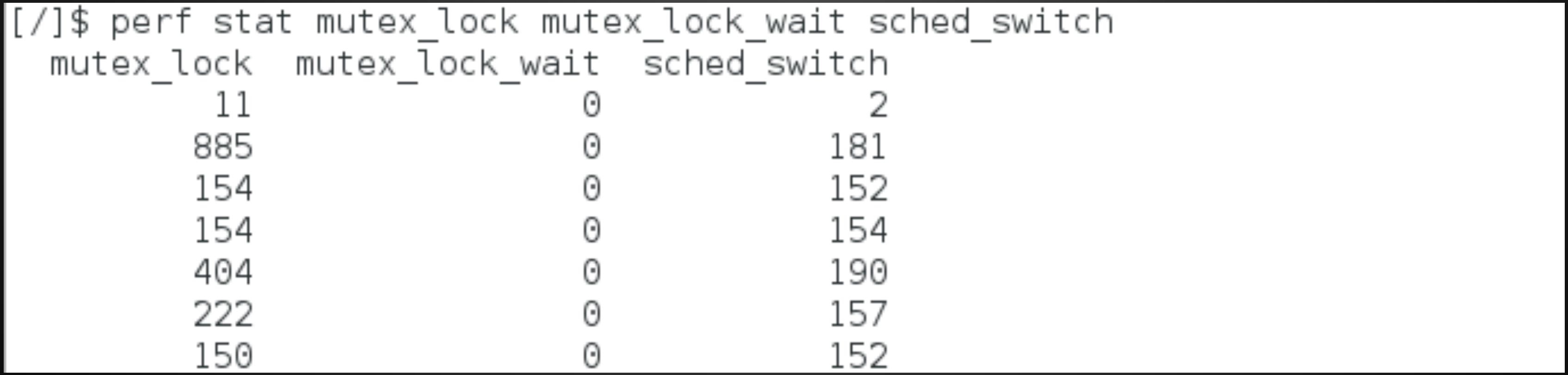
```
int after(something* p)
{
    if (!p)
        return -ENOENT;
    WITH_LOCK(p->lock) {
        if (!p->y)
            return -EINVAL;
        WITH_LOCK(p->y->lock)
            return ++p->y->n;
    }
}
```



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
```

Atomic allocation & initialization

Allocate memory and initialize it in one step

- No need to track the size
- No error checking between steps

Containers

- vector<foo> - growable array
- unordered_map<key, value> - growable hash table
- list<bar> - doubly linked list
- set<whatever> - sorted balanced tree

Reduce the role of laziness in determining key data structures

templates - enforcing concepts at compile time

```
rcu_ptr<vector<device>> device_list;

// update:
device_list.assign(new_device_list);

// read:
auto list = device_list.read();
```

Reference counted objects

shared_ptr<device> - fully automatic reference counting

intrusive_ptr<device> - full manual control

Generic callbacks

```
function<void (int level)> irq_handler;
function<u64 (hw_addr addr, unsigned size)>
    read_callback;

irq_handler = my_irq_handler;
read_callback = bind(this, &my_device::read);
```

Generic callbacks

```
function<void (int level)> irq_handler;
function<u64 (hw_addr addr, unsigned size)>
    read_callback;

irq_handler = my_irq_handler;
read_callback = bind(this, &my_device::read);
```

Signals and slots

```
signal <void ()> system_reset;


...
system_reset.connect([&] { reset_bar0(); });
…



system_reset();
```

Conclusions

- OS^V experience shows modern system programming is made easier in C++
- Boilerplate (and silly mistakes) reduced
- Easy, fast to use and build frameworks
- More fun too!
- Lessons applicable to QEMU

Resources

http://osv.io

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

@CloudiusSystems

osv-dev@googlegroups.com

#irc #osv on FreeNode