DC/OS

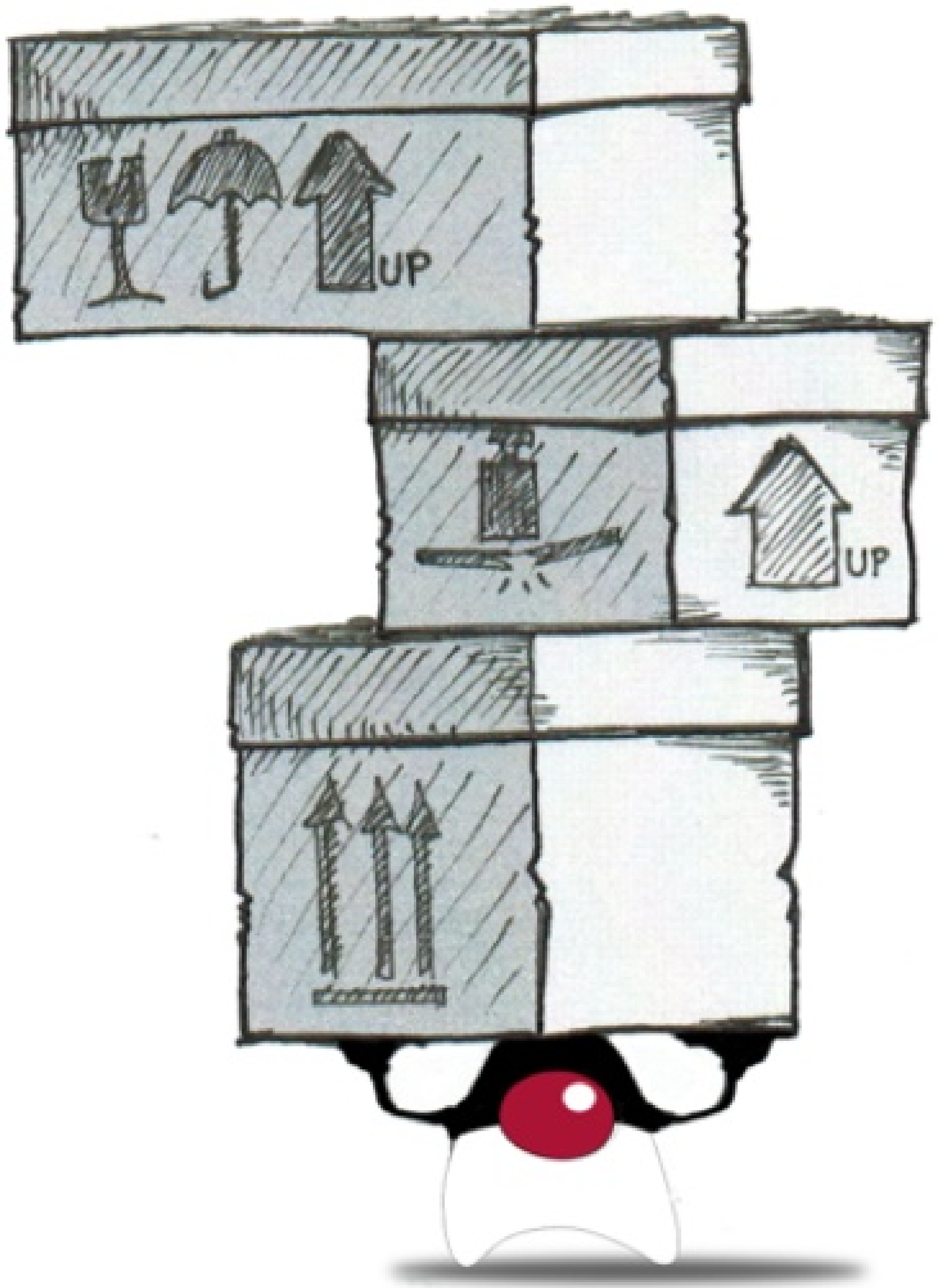
Nobody puts Spark in the Container

Jörg Schad & Ken Sipe

Mesosphere

SPARK SUMMIT EUROPE 2016

NOBODY PUTS
BABY
IN THE CORNER


UP
UP

Ken Sipe

Distributed Applications Engineer,
Mesosphere

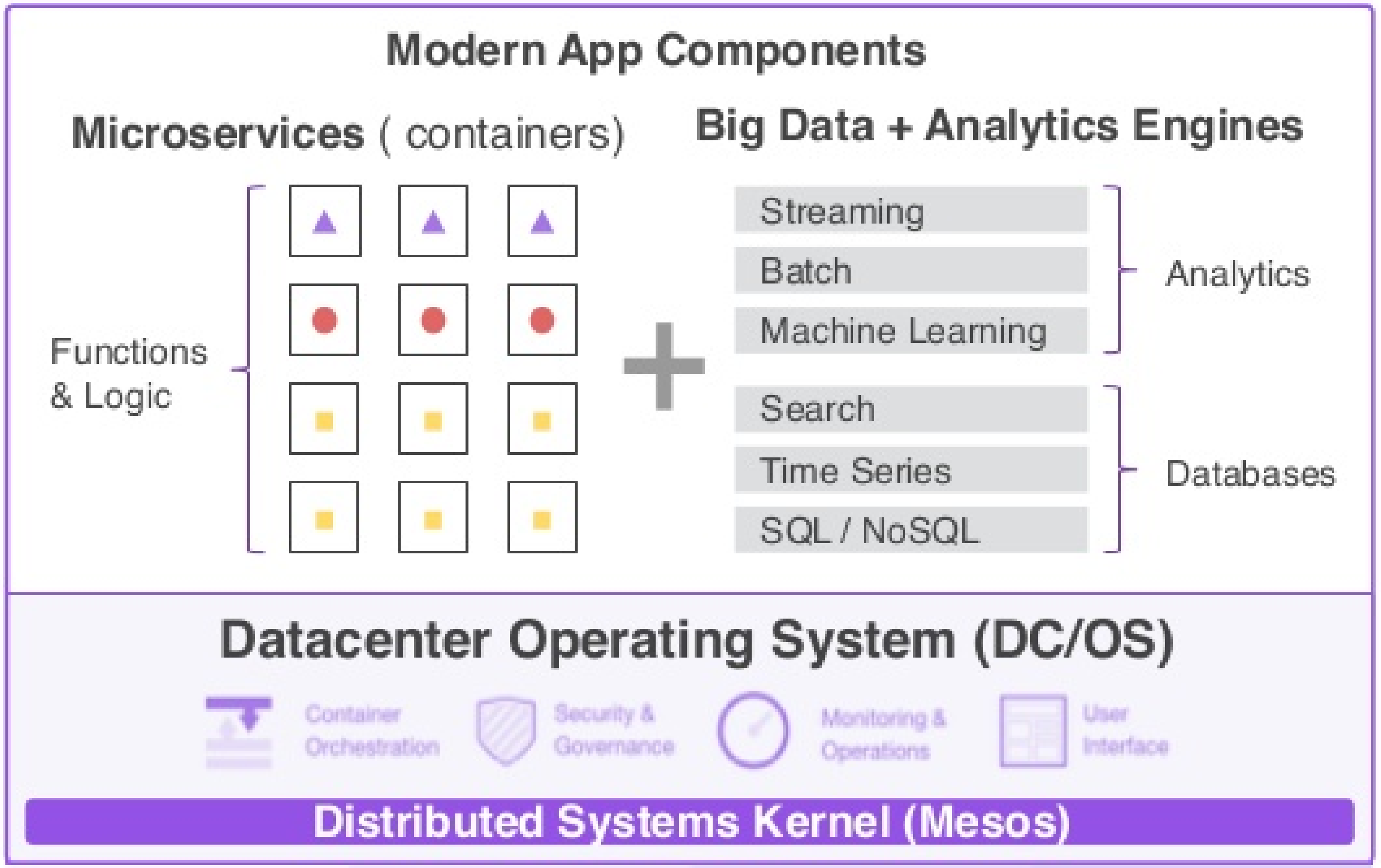
@KenSipe

Jörg Schad

Distributed Systems Engineer,
Mesosphere

@joerg_schad

DATACENTER OPERATING SYSTEM (DC/OS)



Modern App Components

Microservices (containers)

Big Data + Analytics Engines

Functions & Logic

Streaming

Batch

Machine Learning

Analytics

Search

Time Series

SQL / NoSQL

Databases

Datacenter Operating System (DC/OS)

Distributed Systems Kernel (Mesos)

Any Infrastructure (Physical, Virtual, Cloud)

DC/OS

- Container operations & big data operations
- Security, fault tolerance & high availability
- Open Source (ASL2.0)
- Based on Apache Mesos
- Production proven at scale

DC/OS Universe

- Datacenter-wide services to power your apps
- Turnkey installation and lifecycle management

Any Infrastructure

- Requires only a modern linux distro (windows coming soon)
- Hybrid Datacenter

Containers

© Gerard Julien/ AFP

Write Once Run Any Where

High level: it appears* like a lightweight VM

- I can get a shell on it (through SSH or otherwise)
- It "feels" like a VM:
  - own process space
  - own network interface
  - can install packages
  - can run services

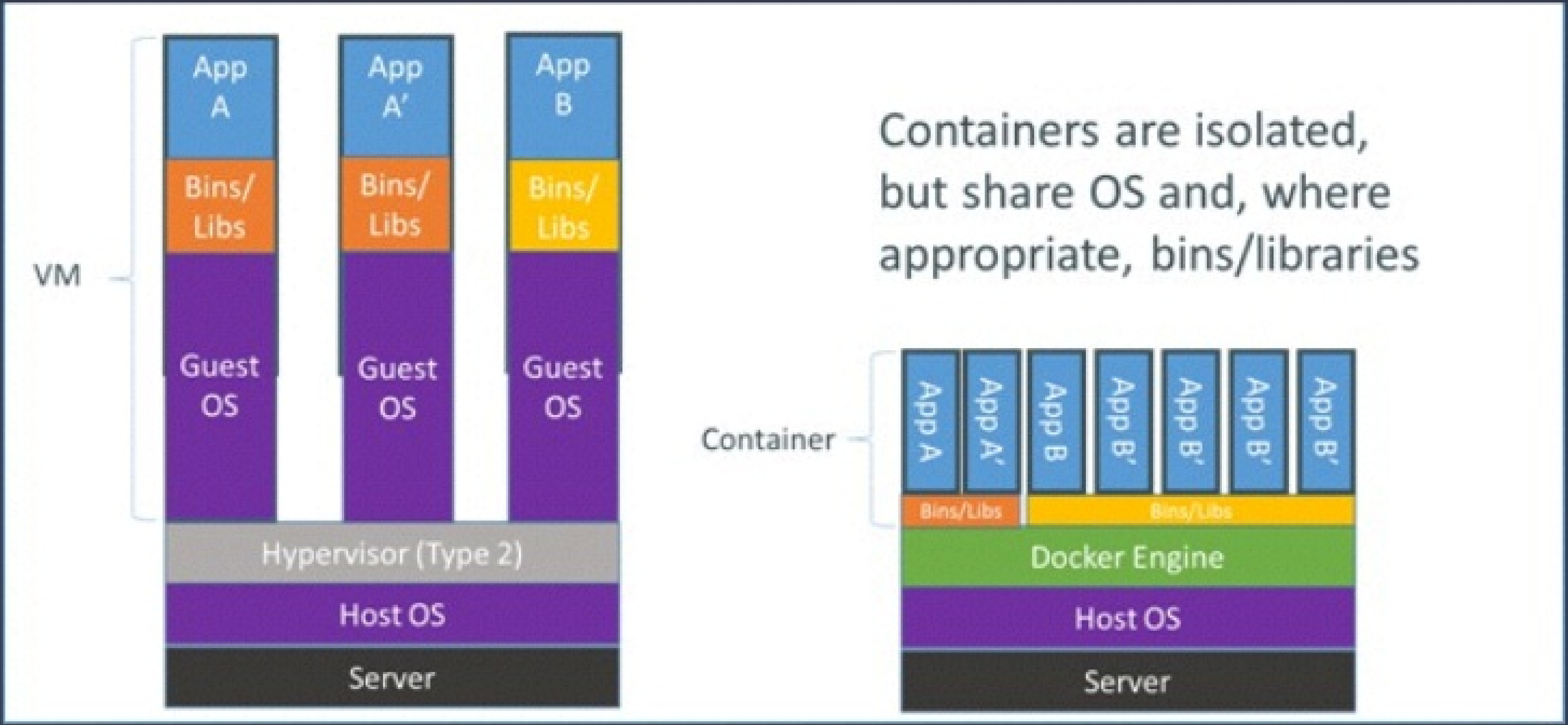
Low level: it's actually chroot on steroids

- It's not like a VM:

  uses the host kernel
  can't boot a different OS

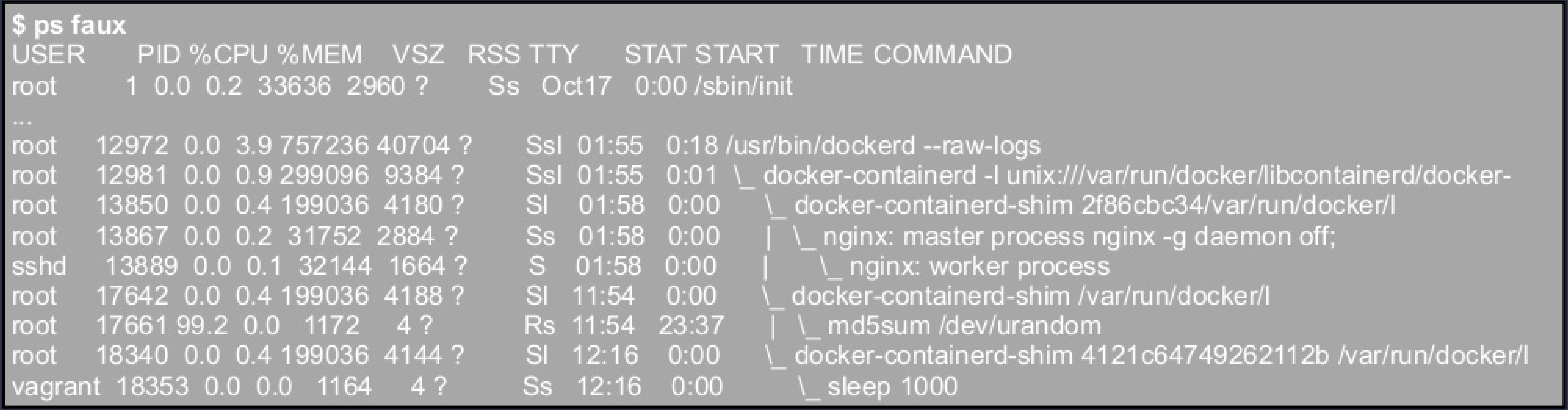
- It's just a bunch of processes visible on the host machine
  - (contrast with VMs which are opaque)

Containers vs Virtual Machines



Containers are isolated, but share OS and, where appropriate, bins/libraries

docker run -d nginx:1.10

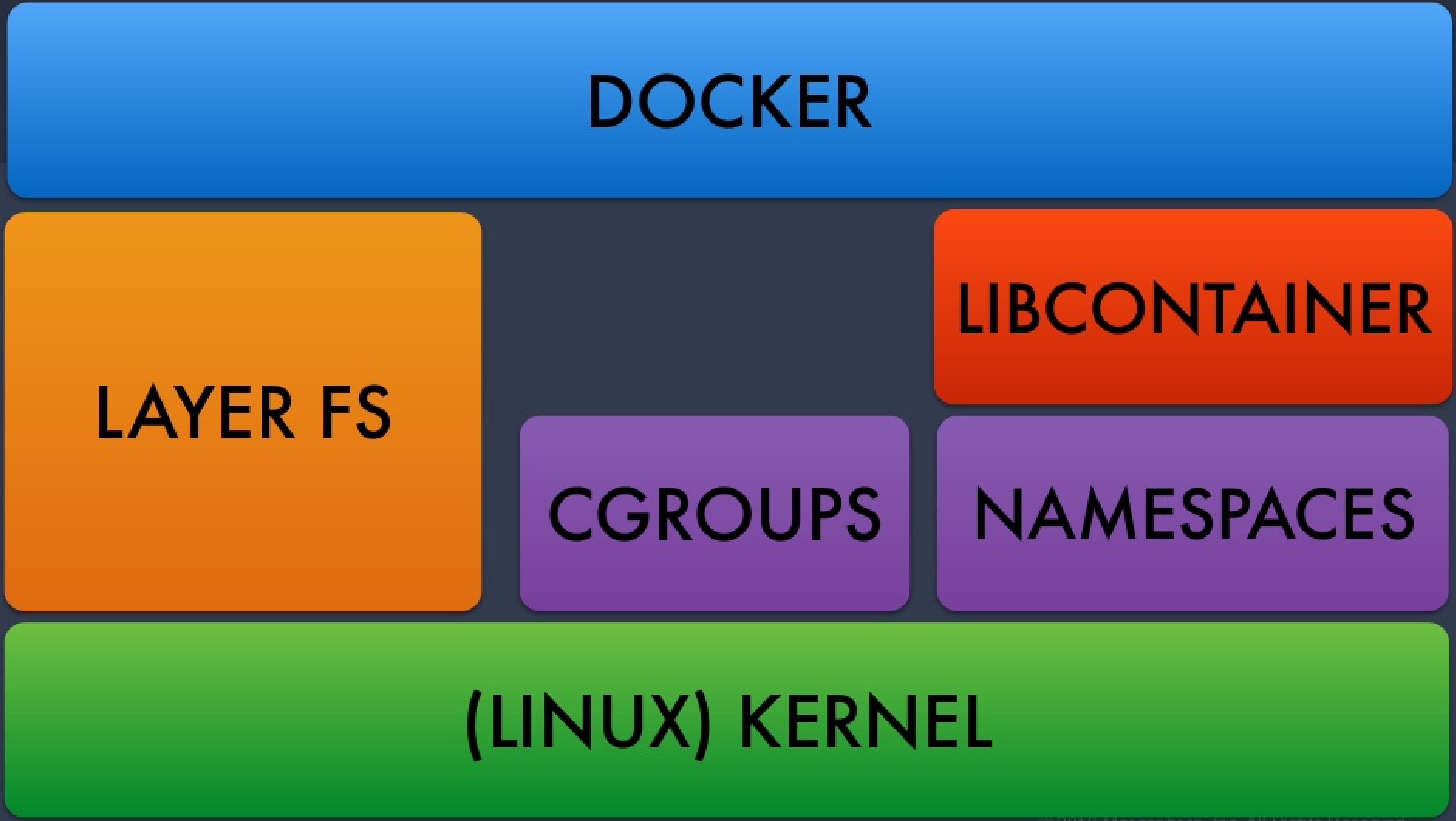


```
$ ps faux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2  33636  2960 ?        Ss   Oct17   0:00 /sbin/init
...
root     12972  0.0  3.9 757236 40704 ?        Ssl  01:55   0:18 /usr/bin/dockerd --raw-logs
root     12981  0.0  0.9 299096  9384 ?        Ssl  01:55   0:01  \_ docker-containerd -l unix:///var/run/docker/libcontainerd/docker-
root     13850  0.0  0.4 199036  4180 ?        Sl   01:58   0:00      \_ docker-containerd-shim 2f86cbc34/var/run/docker/l
root     13867  0.0  0.2  31752  2884 ?        Ss   01:58   0:00      |   \_ nginx: master process nginx -g daemon off;
sshd     13889  0.0  0.1  32144  1664 ?        S    01:58   0:00      |       \_ nginx: worker process
root     17642  0.0  0.4 199036  4188 ?        Sl   11:54   0:00      \_ docker-containerd-shim /var/run/docker/l
root     17661 99.2  0.0   1172     4 ?        Rs   11:54  23:37      |   \_ md5sum /dev/urandom
root     18340  0.0  0.4 199036  4144 ?        Sl   12:16   0:00      \_ docker-containerd-shim 4121c64749262112b /var/run/docker/l
vagrant  18353  0.0  0.0   1164     4 ?        Ss   12:16   0:00          \_ sleep 1000
```

Differences between containers and virtual machines

- Weaker isolation in containers
- Containers run near-native speed CPU/IO
- Containers launch in around 0.1 second (libcontainer)
- Less storage and memory overhead

Isolation



DOCKER

LAYER FS

LIBCONTAINER

CGROUPS

NAMESPACES

(LINUX) KERNEL

Namespaces VS. Cgroups

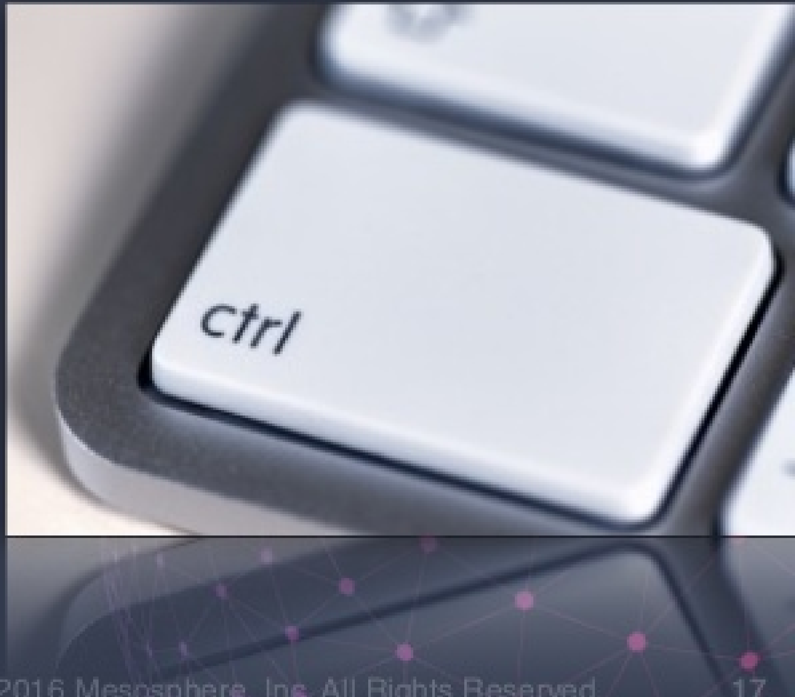
Namespaces provide isolated views:

- pid (processes)
- net (network interfaces, routing...)
- ipc (System V IPC)
- mnt (mount points, filesystems)
- uts (hostname)
- user (UIDs)

Control groups control resources:

- cpu (CPU shares)
- cpuacct
- cpuset (limit processes to a CPU)
- memory (swap, dirty pages)
- blkio (throttle reads/writes)
- devices
- net_cls, net_prio: control packet class and priority
- freezer

Control Groups



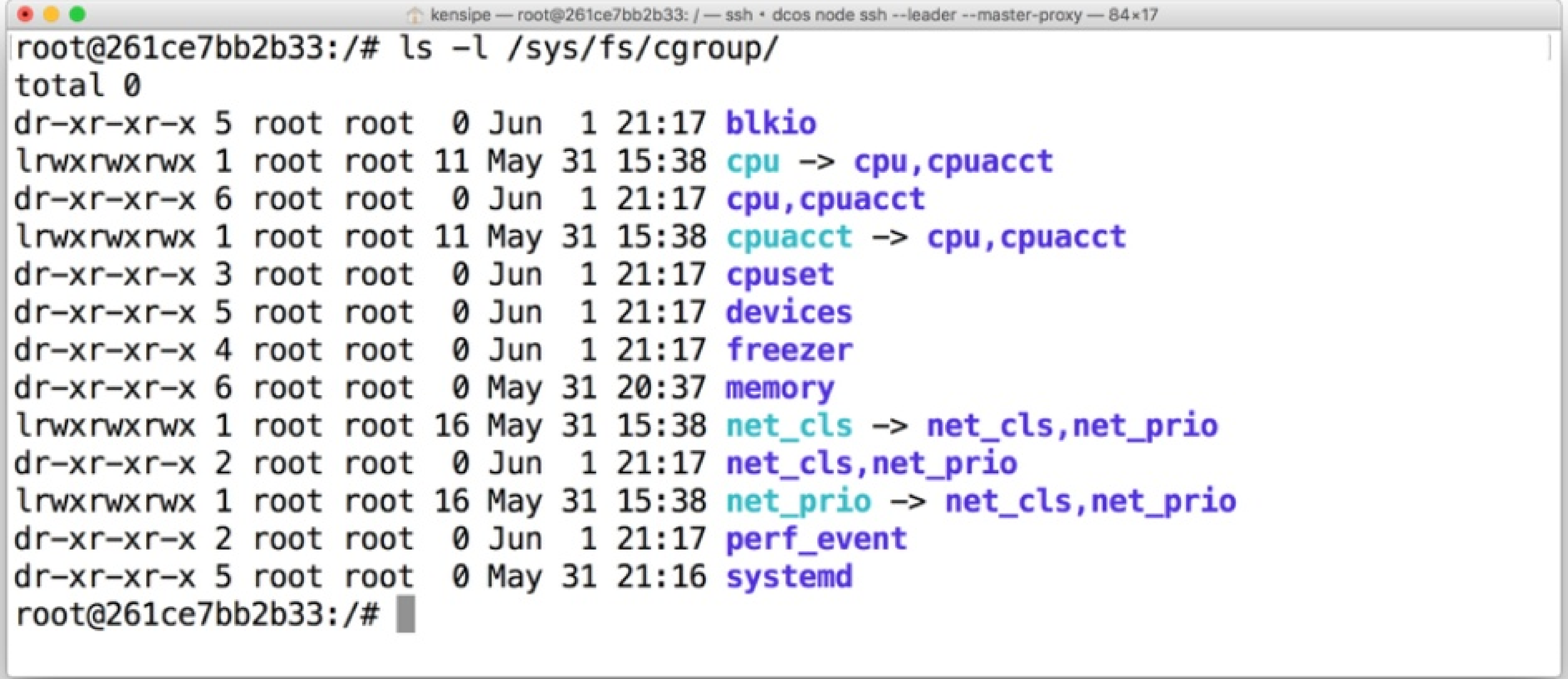
Control groups

- Resource metering and limiting
  - memory
  - CPU
  - block I/O
  - network*
  - device node (/dev/*) access control
- freezer

Control groups - Generalities

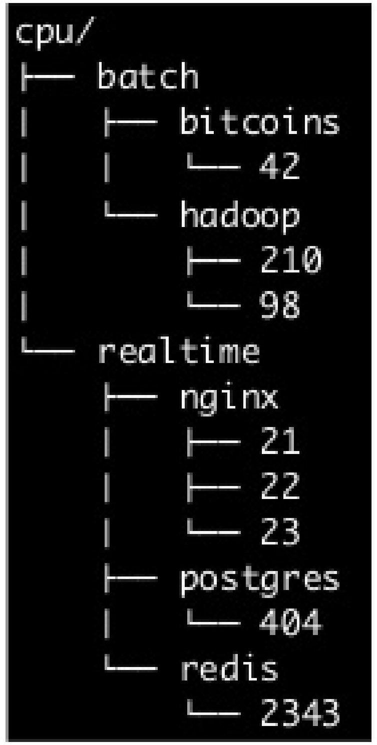
- */sys/fs/cgroup*
- Each subsystem (memory, CPU...) has a hierarchy (tree)
- Each process belongs to exactly 1 node in each hierarchy
- Each hierarchy starts with 1 node (the root)
- Each node = group of processes (sharing the same resources)

DC/OS on CoreOS

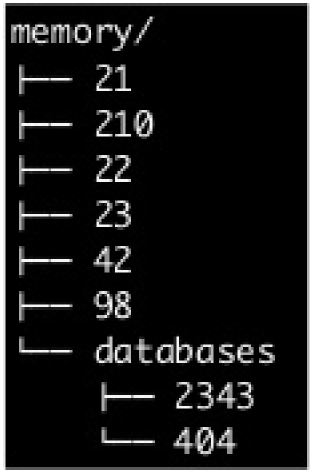


kensipe — root@261ce7bb2b33: / — ssh • dcos node ssh --leader --master-proxy — 84×17

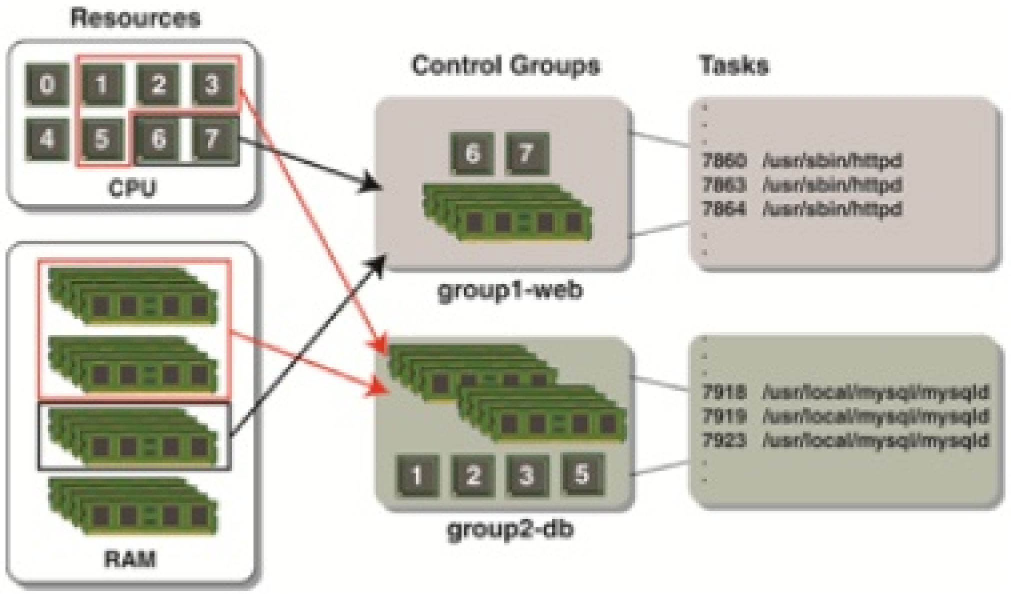
```
root@261ce7bb2b33:/# ls -l /sys/fs/cgroup/
total 0
dr-xr-xr-x 5 root root  0 Jun  1 21:17 blkio
lrwxrwxrwx 1 root root 11 May 31 15:38 cpu -> cpu,cpuacct
dr-xr-xr-x 6 root root  0 Jun  1 21:17 cpu,cpuacct
lrwxrwxrwx 1 root root 11 May 31 15:38 cpuacct -> cpu,cpuacct
dr-xr-xr-x 3 root root  0 Jun  1 21:17 cpuset
dr-xr-xr-x 5 root root  0 Jun  1 21:17 devices
dr-xr-xr-x 4 root root  0 Jun  1 21:17 freezer
dr-xr-xr-x 6 root root  0 May 31 20:37 memory
lrwxrwxrwx 1 root root 16 May 31 15:38 net_cls -> net_cls,net_prio
dr-xr-xr-x 2 root root  0 Jun  1 21:17 net_cls,net_prio
lrwxrwxrwx 1 root root 16 May 31 15:38 net_prio -> net_cls,net_prio
dr-xr-xr-x 2 root root  0 Jun  1 21:17 perf_event
dr-xr-xr-x 5 root root  0 May 31 21:16 systemd
root@261ce7bb2b33:/#
```



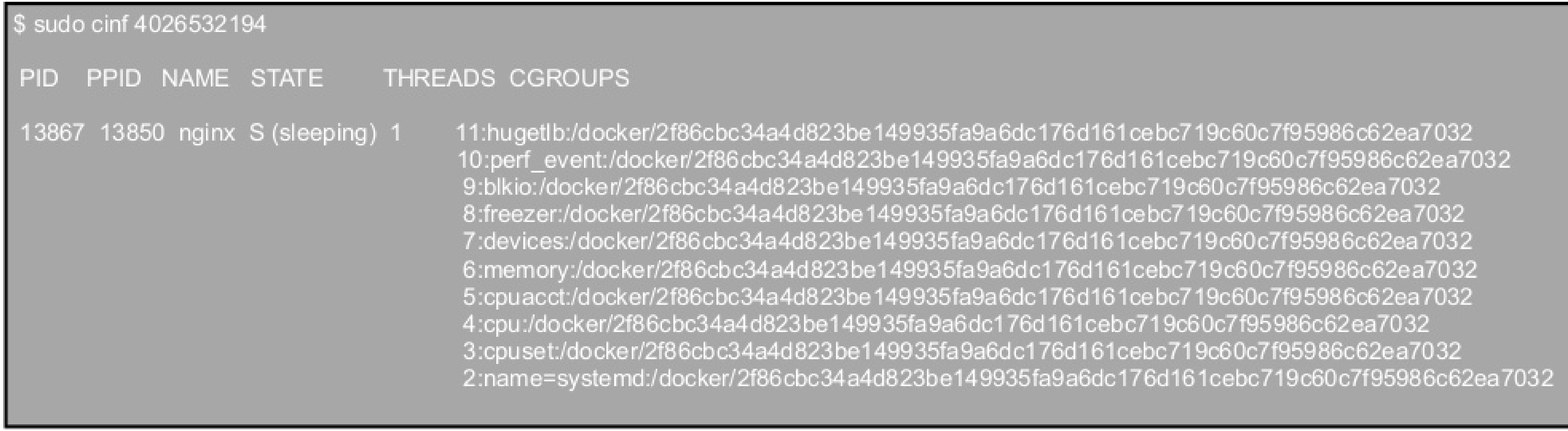
```
cpu/
├── batch
|   ├── bitcoins
|   |   └── 42
|   └── hadoop
|       ├── 210
|       └── 98
└── realtime
    ├── nginx
    |   ├── 21
    |   ├── 22
    |   └── 23
    ├── postgres
    |   └── 404
    └── redis
        └── 2343
```



```
memory/
├── 21
├── 210
├── 22
├── 23
├── 42
├── 98
└── databases
    ├── 2343
    └── 404
```



CINF*



```
$ sudo cinf 4026532194

 PID    PPID   NAME   STATE        THREADS  CGROUPS

 13867  13850  nginx  S (sleeping)  1       11:hugetlb:/docker/2f86cbc34a4d823be149935fa9a6dc176d161cebc719c60c7f95986c62ea7032
                                            10:perf_event:/docker/2f86cbc34a4d823be149935fa9a6dc176d161cebc719c60c7f95986c62ea7032
                                             9:blkio:/docker/2f86cbc34a4d823be149935fa9a6dc176d161cebc719c60c7f95986c62ea7032
                                             8:freezer:/docker/2f86cbc34a4d823be149935fa9a6dc176d161cebc719c60c7f95986c62ea7032
                                             7:devices:/docker/2f86cbc34a4d823be149935fa9a6dc176d161cebc719c60c7f95986c62ea7032
                                             6:memory:/docker/2f86cbc34a4d823be149935fa9a6dc176d161cebc719c60c7f95986c62ea7032
                                             5:cpuacct:/docker/2f86cbc34a4d823be149935fa9a6dc176d161cebc719c60c7f95986c62ea7032
                                             4:cpu:/docker/2f86cbc34a4d823be149935fa9a6dc176d161cebc719c60c7f95986c62ea7032
                                             3:cpuset:/docker/2f86cbc34a4d823be149935fa9a6dc176d161cebc719c60c7f95986c62ea7032
                                             2:name=systemd:/docker/2f86cbc34a4d823be149935fa9a6dc176d161cebc719c60c7f95986c62ea7032
```

*https://github.com/mhausenblas/cinf

Memory cgroup: accounting

- Metrics: swap, total rss, # pages in/out
- Keeps track of pages used by each group:
  - file (read/write/mmap from block devices)
  - anonymous (stack, heap, anonymous mmap)
  - active (recently accessed)
  - inactive (candidate for eviction)

Memory cgroup: limits

- Each group can have **hard** and **soft** limits
- Soft limits are not enforced
- Hard limits will trigger a per-group OOM killer
  - No *OutOfMemoryError*
- Limits can be set for physical, kernel, total memory

```
docker run -it --rm -m 128m fedora bash
```

Cpu cgroup

- Metrics: cpuacct.stats user | system
- Limitations based on type
  - **CPU Shares**
  - **CPU Sets**

CPU Shares

- Priority Weighting across all the cores

- default value is 1024

```
docker run -it --rm -c 512 stress …
```

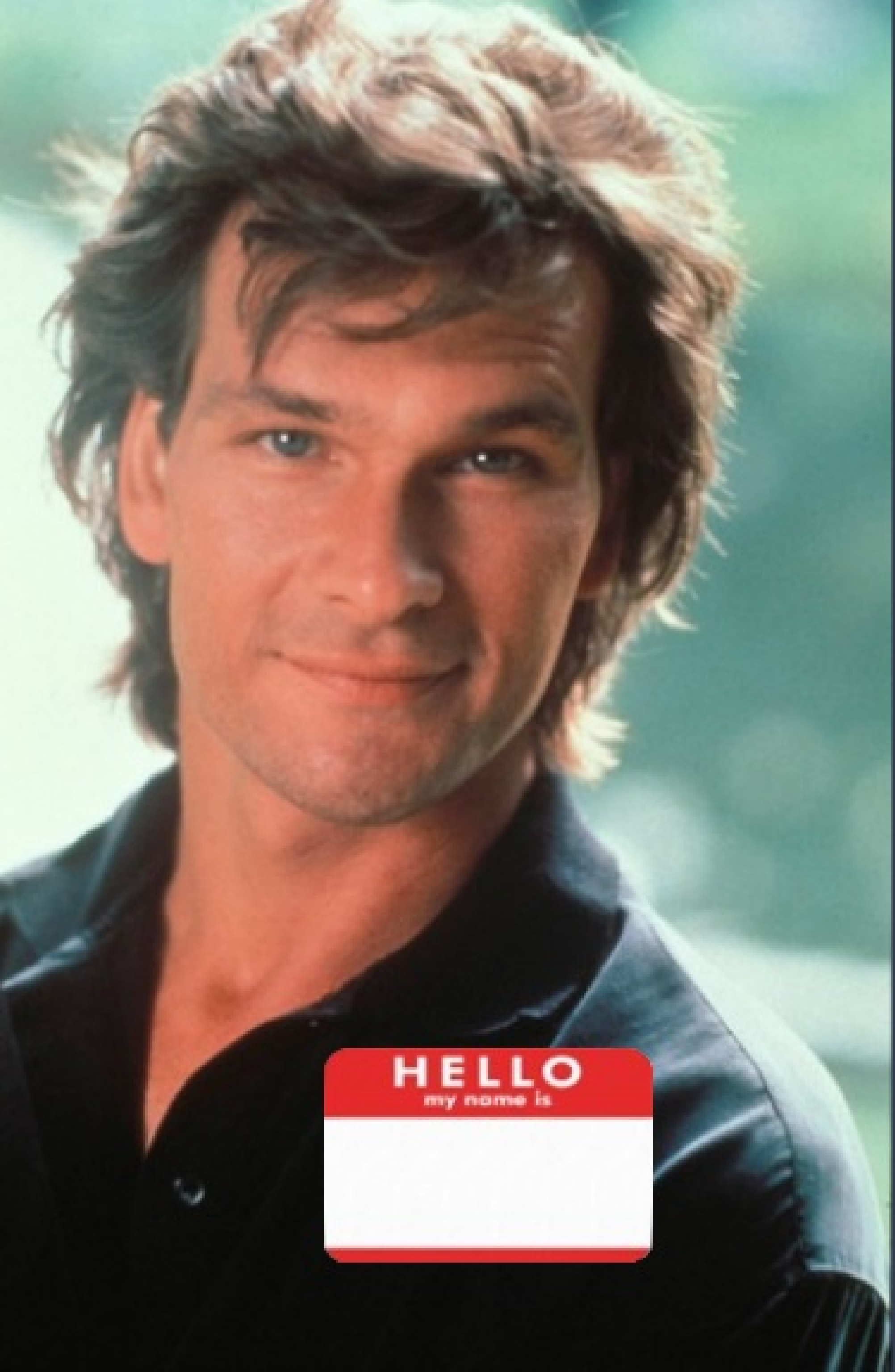
CPU Shares

- sudo cgcreate -g cpu:A
- sudo cgcreate -g cpu:B
- cgroup A: sudo cgset -r cpu.shares=768 A **75%**
- cgroup B: sudo cgset -r cpu.shares=256 B **25%**

CPU Sets

- **Pin** groups to **specific CPU**(s)
- **Reserve CPU**s for specific apps
- Avoid processes bouncing between CPUs
- Also relevant for NUMA systems

```
docker run -it -cpuset=0,4,6 stress
```



Namespaces

Namespaces

- Provide processes with their own view of the system
- Multiple namespaces:
  - pid, net, mnt, uts, ipc, user
- Each process is in one namespace of each type

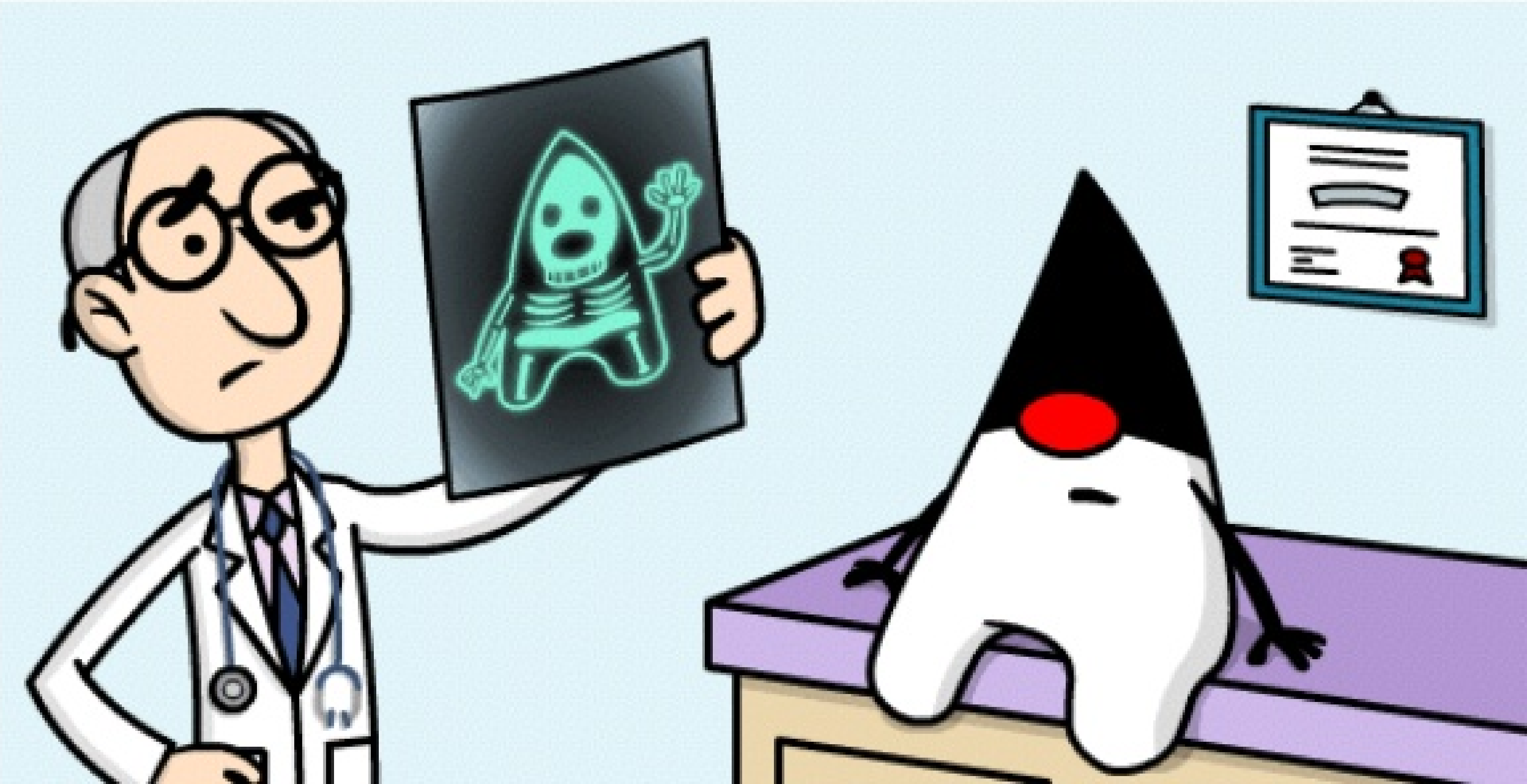
Pid namespace

- Processes within a PID namespace only see processes in the same PID namespace
- Each PID namespace has its own numbering (starting at 1)
- When PID 1 goes away, the whole namespace is killed

Lets Talk Java

Java

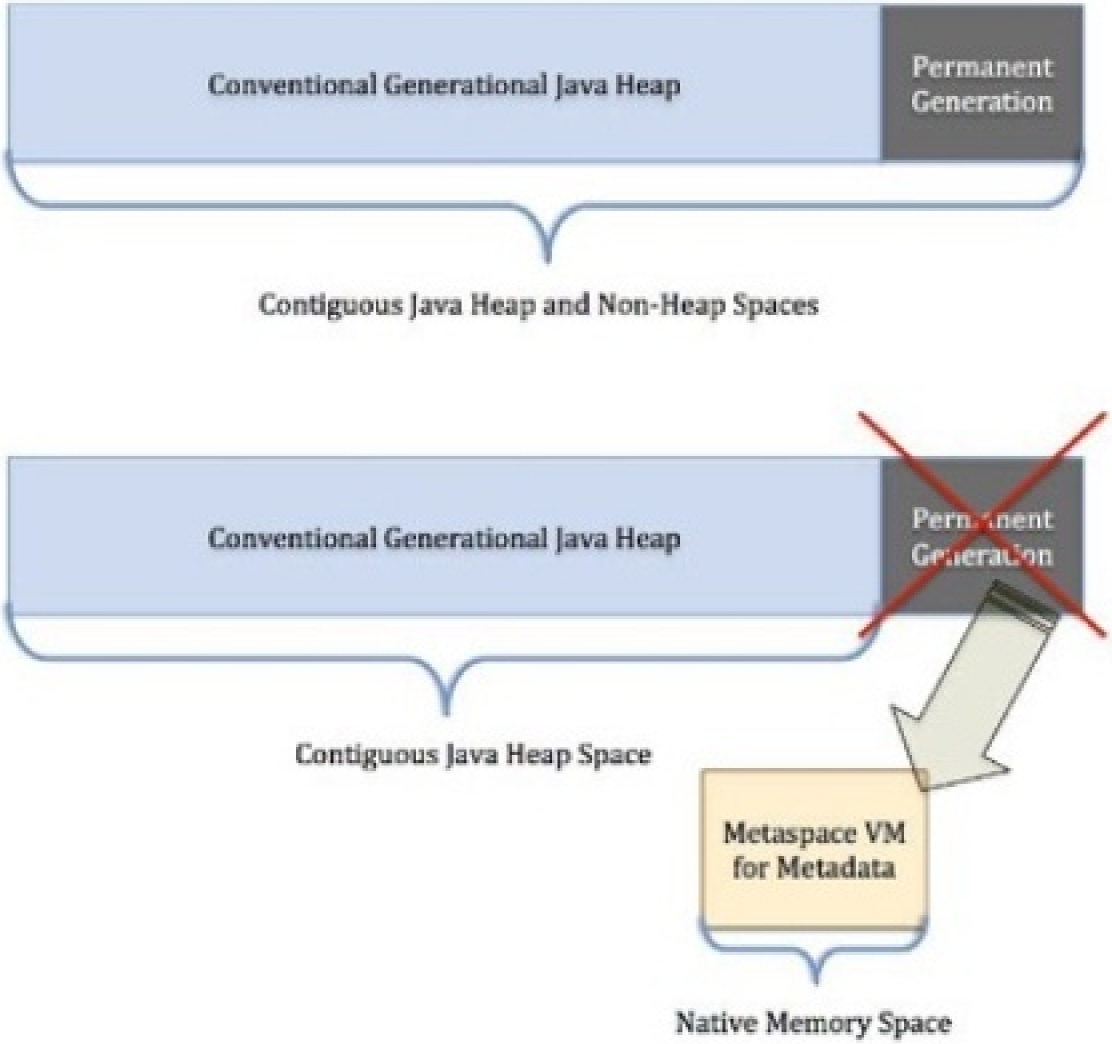
- Java Language + Java Specification + **Java Runtime**



Java Memory Impact

- Native JRE
- Heap
- Perm / meta
- JIT bytecode
- JNI
- NIO
- Threads

From Perm to Metaspace


Conventional Generational Java Heap
Permanent Generation
Contiguous Java Heap and Non-Heap Spaces
Conventional Generational Java Heap
Permanent Generation
Contiguous Java Heap Space
Metaspace VM for Metadata
Native Memory Space

JRE initializations based on core count

- JIT compiler threads
- HotSpot thresholds and optimizations
- Sets the default # threads for GC
- Number of thread in the common fork-join pool
- and more…

Bring it together!

Where Java Gets it’s CPU Information

- JDK 7/8 - resources from sysconf

```
sysconf(_SC_NPROCESSORS_ONLN);
```

- JDK 9 - sched_getaffinity
  - accounts for cpusets

Java with CPU Set

- CPUSET
  - pin to specific CPUs

- Runtime.getRuntime().availableProcessors();   == # cores assigned*

```
docker run -ti --cpuset=0,4,6 …
```

Java with CPU Share

- CPU Share
  - Priority Weighting across all the cores
  - Runtime.getRuntime().availableProcessors(); == **# cores on node**

```
docker run -ti -c 512 …
```

Java and CPU Shares

- Land on a 32 core box
  - 32 cores are seen by the JRE
  - 32 threads set by default for ForkJoinPool

How about memory?

- “But memory constraints are far more problematic and may not even be queryable in general.
- If there are no API's to tell the VM the real resource story what is the VM supposed to do? I don't have any answers to that.”
- “**When the environment lies to the VM about what is available it makes it very hard for the VM to try to adjust.**”

Conclusion

- “The **good** thing about docker containers (and some other like containers) is that they **don’t hide the underlying hardware** from processes like VM technology does.”

- “The **bad** thing about docker containers (and some other like containers) is that they **don’t hide the underlying hardware** from processes like VM technology does.”

Kirk Pepperdine

Thank You!

Learn more by visiting dcos.io and mesosphere.com

journalctl -f _TRANSPORT=kernel

- Mar 10 17:42:39 ip-10-0-1-114.us-west-2.compute.internal mesos-slave[1190]: I0310 17:42:39.848748 1199 status_update_manager.cpp:824] Checkpointing ACK for status update TASK_RUNNING (UUID: 8d13fbb9-b02a-45da-9b52-5393ce8f0746) for task task.datanode.datanode1.1457631756250 of framework d83631ed-34
- Mar 10 17:42:41 ip-10-0-1-114.us-west-2.compute.internal mesos-slave[1190]: I0310 17:42:41.561954 1200 mem.cpp:625] **OOM** notifier is triggered for container **6461cafd-3962-4022-a070-f6e26488dd94**
- Mar 10 17:42:41 ip-10-0-1-114.us-west-2.compute.internal mesos-slave[1190]: I0310 17:42:41.562047 1200 mem.cpp:644] OOM detected for container 6461cafd-3962-4022-a070-f6e26488dd94
- Mar 10 17:42:41 ip-10-0-1-114.us-west-2.compute.internal mesos-slave[1190]: I0310 17:42:41.566249 1200 mem.cpp:685] Memory limit exceeded: Requested: 2080MB Maximum Used: 2080MB