# Putting Condor in a Container

Applying Virtualization Techniques to Batch Systems

Brian Bockelman, UNL

# This is a talk about Virtualization

- This is not a talk about virtual machines.

# To Virtualize

- virtualize |ˈvərCHo͞oəˌlīz|
  - verb [ with obj. ]
  - create a virtual version of (a computing resource or facility).
- We will be virtualizing the worker node, but not by using virtual machines.

# Containers, Broadly Speaking

- Partition system resources using the host kernel.
    - Do not run a complete virtual machine with separate kernel, but run isolated user processes partitioned from the rest of the system.
- It creates a virtualized userland environment, but all containers share the same kernel.

Defacto "implementation": http://lxc.sourceforge.net/

# Partitioning

- Containers typically take advantage of the resource partitioning features available in newer kernels.
  - These are typically implemented via "control groups", or "cgroups" or namespaces.
  - Cgroups are control structures for managing sets of processes in a Linux system.
  - Different cgroup subsystems ("controllers") may act on these structures to control scheduler policy, allocate/limit resources, or account for usage.

http://en.wikipedia.org/wiki/Cgroups

http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

# Cgroups Quick Intro

- The interface to cgroups is not a syscall, but a pseudo-filesystem (like /proc):

```
mkdir -p /cgroup/blkio
mount -t cgroup -o blkio /cgroup/blkio
mkdir /cgroup/blkio/example_session
echo $$ > /cgroup/blkio/example_session/tasks
```

- The above lines mount the cgroup controller, create a sub-cgroup called "example_session", and place the current shell in that cgroup. Any activity started by this session (regardless of daemonized or not!) will be managed by the "blkio" controller. No, I didn't say what blkio does yet...
- Each cgroup is a directory in the filesystem (provides familiar semantics like sub-directories, Unix permissions to manage the cgroup). Processes in the cgroup appear in the "tasks" file.

See last year's Condor Week talk:
http://research.cs.wisc.edu/condor/CondorWeek2011/presentations/bockelman-user-isolation.pdf

# Goal: Containerize Condor

- We want to expose the various partitioning and management techniques in the Linux kernel to Condor, allowing it to better manage jobs.
  - Think of it as a "blend" between a "normal" batch job and a container, to give Condor batch jobs features normally associated with virtualization.

# Containers in Condor

- I break up the work for "containerizing Condor" into three categories:
  - Isolation. Protecting jobs from each other.
  - Accounting. Understanding the resources the batch jobs use.
  - Resource Management. Implementing policies about what resources and how much the jobs can access.

# Isolation Models

- We typically isolate two jobs from each other by using two different usernames. Other possibilities exist:
  - Process isolation ("PID Namespaces").
  - Filesystem isolation. Users see different mounts.

# Process View

## Outside

```
root       949  0.0  0.0  75320   576 ?        Ss    2011   0:15 /usr/sbin/sshd -D
root     29796  0.0  0.1 123840  4432 ?        S    06:27   0:00  \_ sshd: bbockelm [priv]
bbockelm 29803  0.0  0.0 123840  2096 ?        S    06:27   0:00  |   \_ sshd: bbockelm@pts/1
bbockelm 29804  0.0  0.0 116508  2212 pts/1    Ss   06:27   0:00  |       \_ -bash
root     29964  0.0  0.0 155920  2096 pts/1    S    06:33   0:00  |           \_ sudo ./ns_exec -cpm /bin/sh
root     29965  0.0  0.0   4272   340 pts/1    S    06:33   0:00  |               \_ ./ns_exec -cpm /bin/sh
root     29966  0.0  0.0 116492  1964 pts/1    S+   06:33   0:00  |                   \_ /bin/sh
```

## Inside

```
sh-4.2# ps faux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0 116492  1964 pts/1    S    06:33   0:00 /bin/sh
root         3  0.0  0.0 115660  1076 pts/1    R+   06:34   0:00 ps faux
```

CLI-based example, but same holds true for containers.

Wouldn't it be nice if the batch system did this? :)

# Ta-Da

```
[bbockelm@rcf-bockelman condor]$ condor_run ps faux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
bbockelm      1  0.0  0.0 114140  1236 ?        SNs  12:14   0:00 /bin/bash /home/bbockelm/projects/
condor/.condor_run.8661
bbockelm      2  0.0  0.0 115668  1120 ?        RN   12:14   0:00 ps faux
```

# Mount Under Scratch

- In Condor 7.7.5, we introduced the MOUNT_UNDER_SCRATCH config parameter to the sysadmin.
  - Any directory in the list will be mounted from the job's scratch directory (auto-cleaned by Condor after the job).
  - Equivalent to:

```
mount --bind /var/lib/execute/condor/execute/dir_1234/tmp \
             /tmp
```

# No More Leaked Junk in /tmp!

- Sysadmins rejoice!

# Chroots

- In Condor 7.7.5, users are able to request a specific chroot.
  - The sysadmins assign each chroot they have setup a name (such as "SL5")
  - The user adds "+RequestedChroot=SL5" to their submit file.
- Improved isolation coming in 7.8: jobs are completely isolated from each other in the filesystem.

# Filesystem View

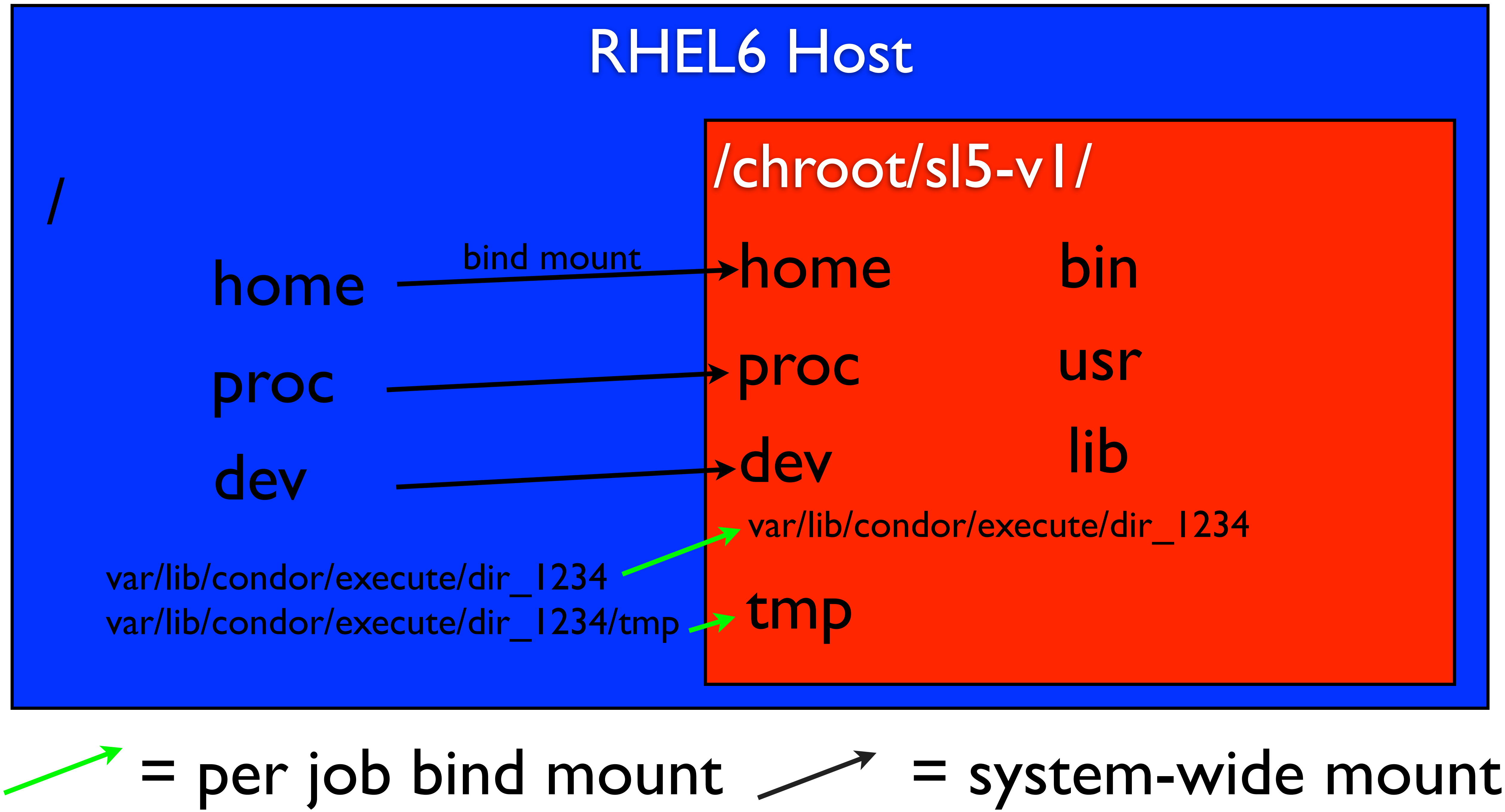
- Containers typically use chroot to provide a completely unique filesystem.

```
[root@red-d15n2 ~]# ls /
bin   cgroup  cvmfs  etc   lib    lost+found  misc  net  proc  sbin     srv  tmp  var
boot  chroot  dev    home  lib64  media       mnt   opt  root  selinux  sys  usr
[root@red-d15n2 ~]# ls /chroot/sl5-v1/root/
bin  boot  builddir  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  sbin  selinux
[root@red-d15n2 ~]# chroot /chroot/sl5-v1/root/
bash-3.2# ls /
bin  boot  builddir  dev  etc  home  lib  lib64  media mnt  opt  proc   root  sbin  selinu
```

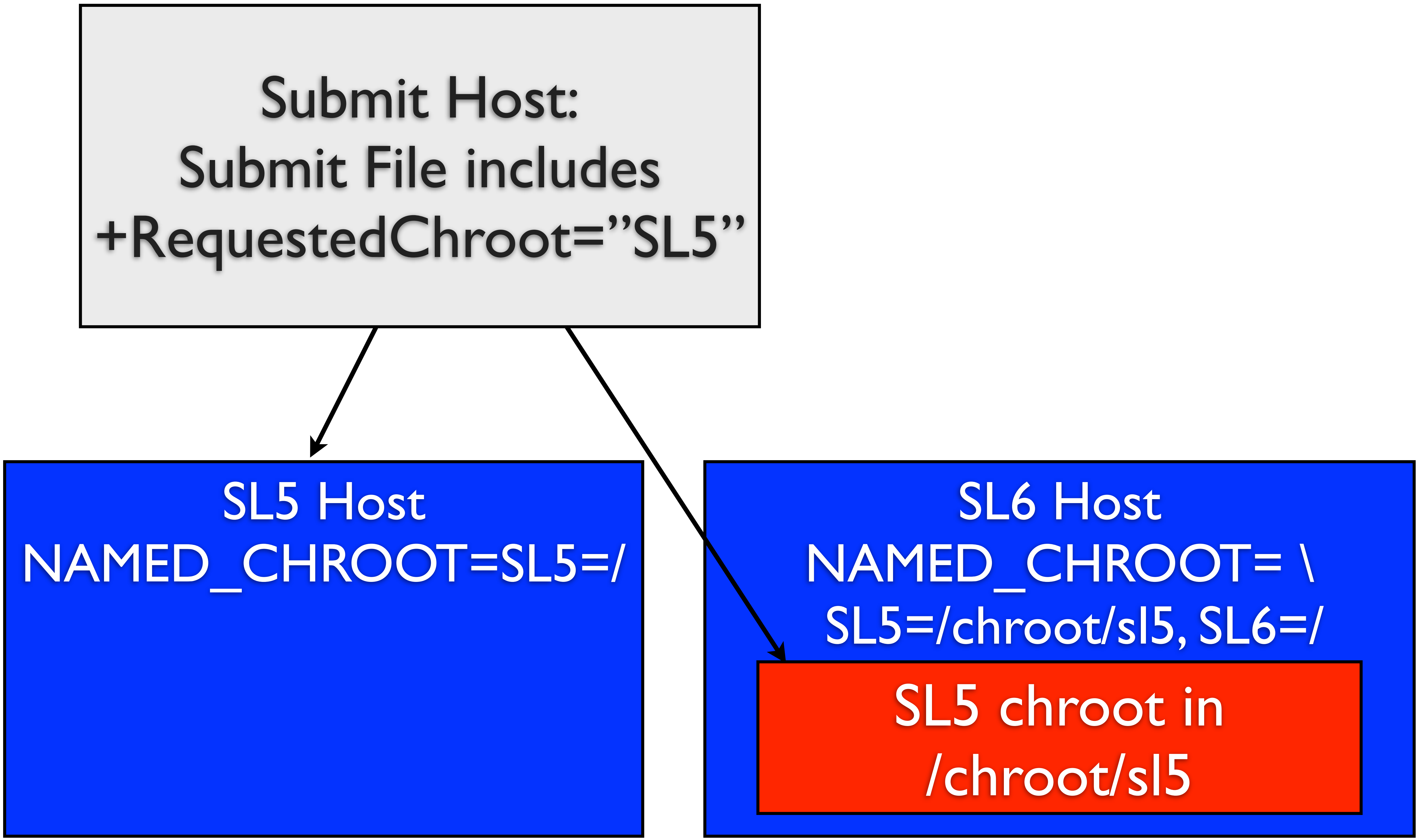
# Chroot at HCC

- Our sysadmins want to run RHEL6 for manageability reasons.
- The experiment that pays our salary requires RHEL5.
- Idea: Create a RHEL5 chroot for each host.

# Chroot setup



# Mixed-mode Pools



# Chroot at HCC

- We have a small wrapper around yum called "chroot-tool" to layout the RHEL5 filesystem.
- Puppet manages bind mounts and the invocation of chroot-tool.
  - Each time we change our userspace configuration, we deploy a new chroot (i.e., /chroot/sl5-v4).
  - Puppet manages where the symlink /chroot/sl5 points.
- Condor starts jobs in /chroot/sl5.
  - Exercise for user: convince yourself we can do atomic upgrades and rollbacks.

https://github.com/bbockelm/hcc-config/blob/master/modules/chroot/manifests/init.pp

https://github.com/bbockelm/RHEL5-chroot

# Accounting

- Linux has some nice statistics *per process*, but Condor wants accounting *per job*.
  - CPU accounting is OK; sum CPU usage of all job processes. Works except when it doesn't.
  - Memory accounting is HORRIBLE!

# Question

- What is the memory footprint of “sleep 5m”?

# Memory Mess

- Summing up processes's memory attributes is a MESS in Linux.
- This does not take into account sharing between processes. In a modern Linux system - and in today's jobs - there is a lot of sharing.
  - Makes today's batch systems wildly inaccurate for accounting.

# Condor in 7.7.0

- Create a cgroup per job relative to a base cgroup (admin-configured). Base cgroup is done so you can manage Condor separately from the system.
- Somewhat equivalent to the following:

```
[root@red-d15n2 ~]# mkdir -p /cgroup/memory/condor/job_1234_5
[root@red-d15n2 ~]# echo $$ > /cgroup/memory/condor/job_1234_5/tasks
[root@red-d15n2 ~]# cat /cgroup/memory/condor/job_1234_5/tasks
13314
16521
[root@red-d15n2 ~]# bash
[root@red-d15n2 ~]# cat /cgroup/memory/condor/job_1234_5/tasks
13314
16522
16531
```

# Memory Accounting

- Tons of statistics can be mined from the memory controller and passed back to Condor.

```
[root@red-d15n2 ~]# cat /cgroup/memory/condor/memory.stat
cache 0
rss 634880
mapped_file 0
pgpgin 602
pgpgout 447
swap 0
inactive_anon 0
active_anon 569344
inactive_file 0
active_file 0
unevictable 0
hierarchical_memory_limit 9223372036854775807
hierarchical_memsw_limit 9223372036854775807
total_cache 0
total_rss 634880
total_mapped_file 0
total_pgpgin 602
total_pgpgout 447
```

# (OOPS)

- New features, new failure modes.
- https://bugzilla.redhat.com/show_bug.cgi?id=816365
  - Set "noswapaccount" kernel boot parameter.

# Network Accounting

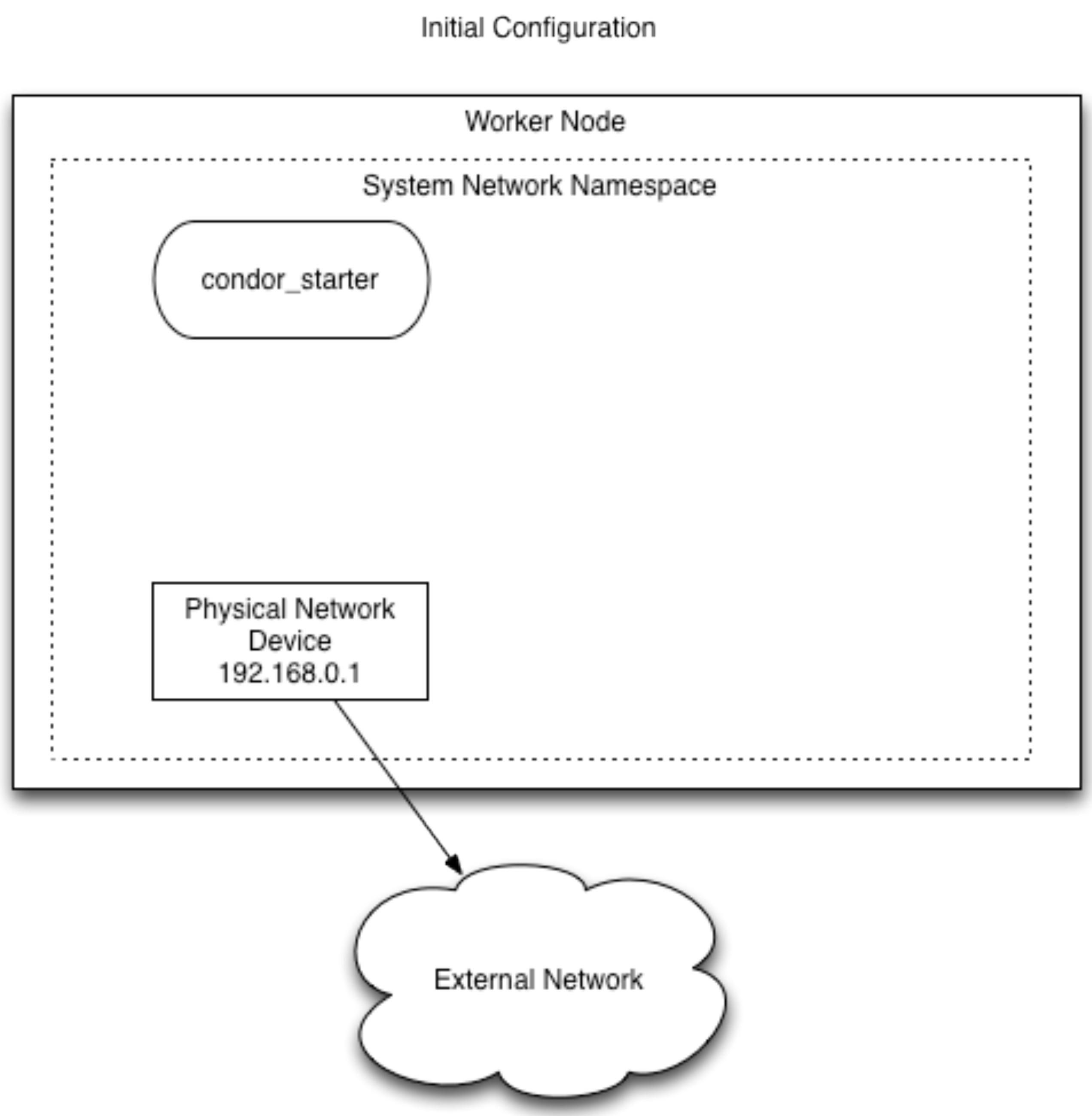
- We are extremely interested in knowing the per-job network I/O figures:
  - Helps us understand if site planning is right.
  - Give appropriate information back to users - and trace a bit about what they did on the network.
  - Compare costs, dollar-for-dollar, against EC2.

http://osgtech.blogspot.com/2011/12/network-accounting-for-condor.html

# Network Namespaces

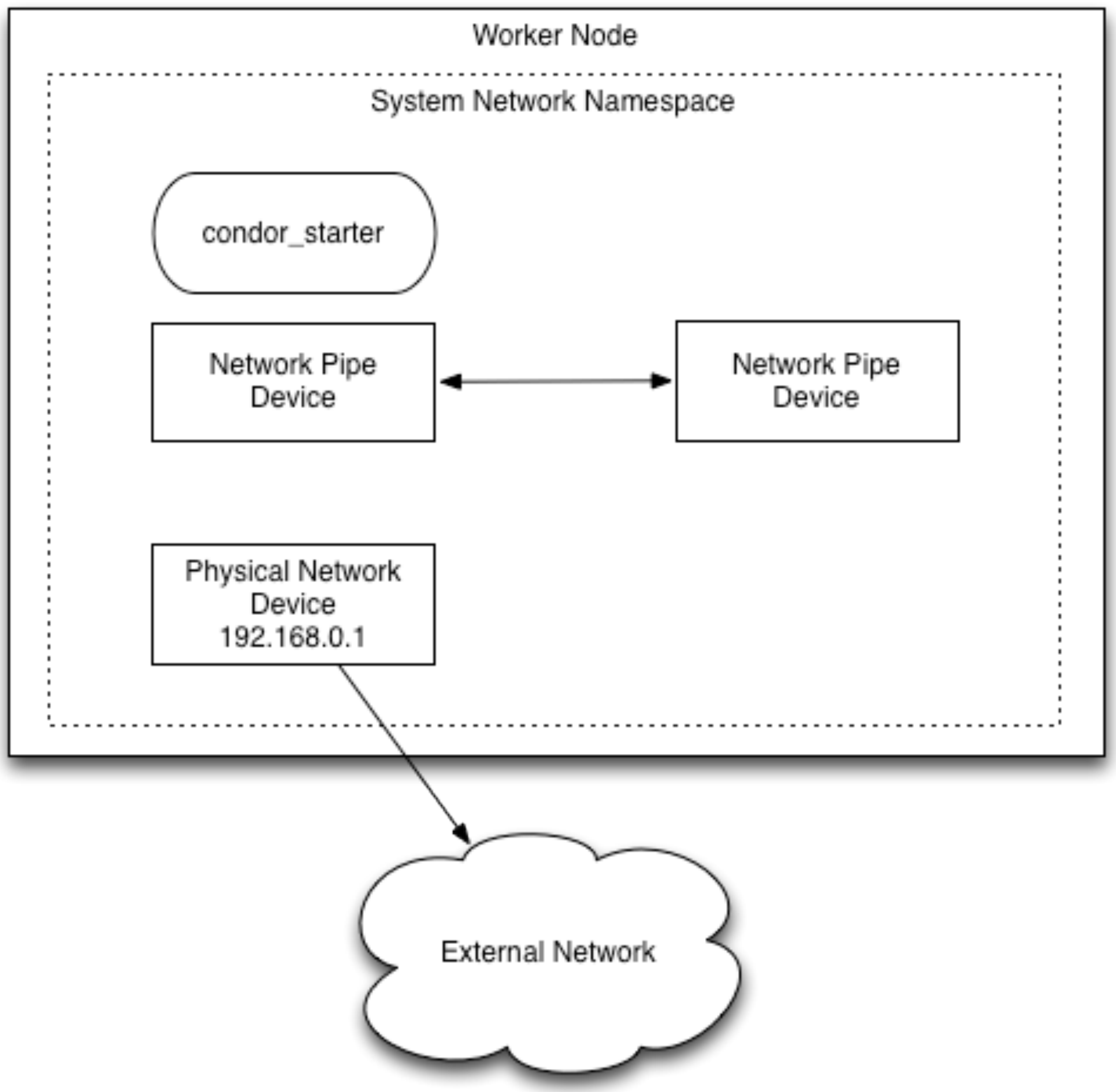
- What's the solution? Namespaces!
  - The "network namespace" is a namespace that can interact with a subset of the network devices on the system.
- The general idea is to create a per-job network device, lock the job to that network device using namespaces, and then do iptables-based accounting for the network device.
  - Approach is illustrated on next slides...

# Network Namespaces: Flipbook

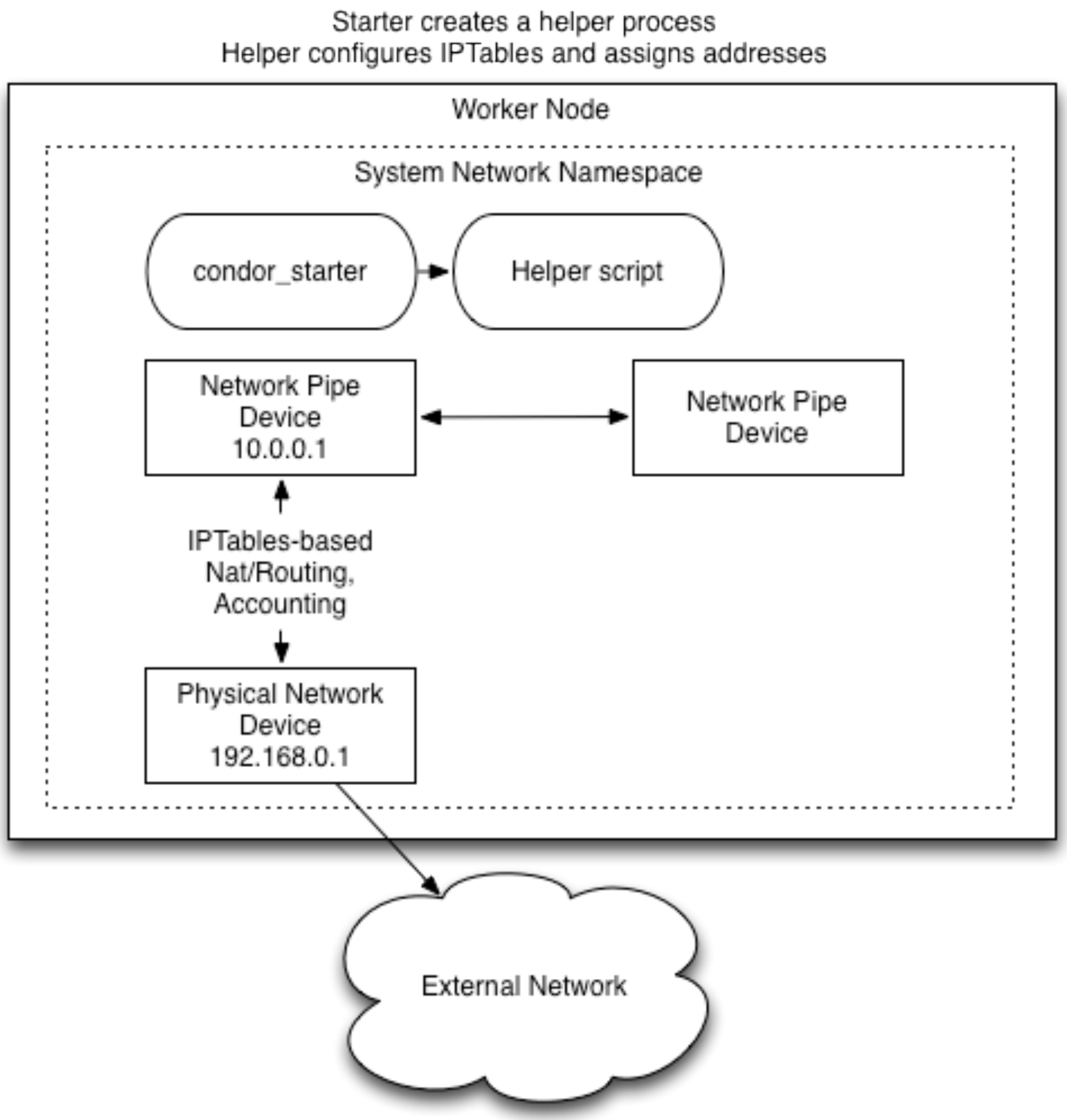


# Network Namespaces: Flipbook

Starter creates network pipes

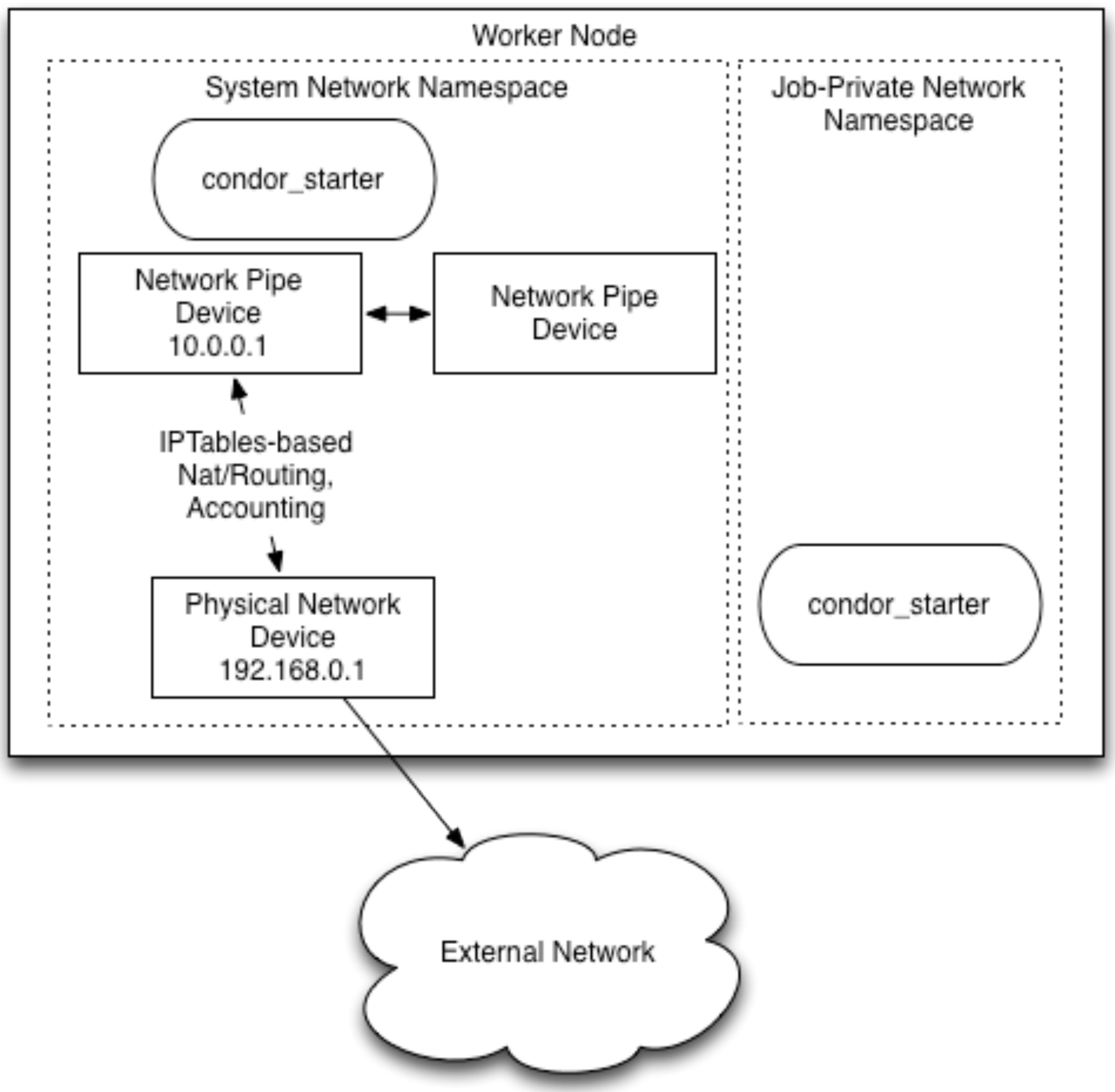


# Network Namespaces: Flipbook

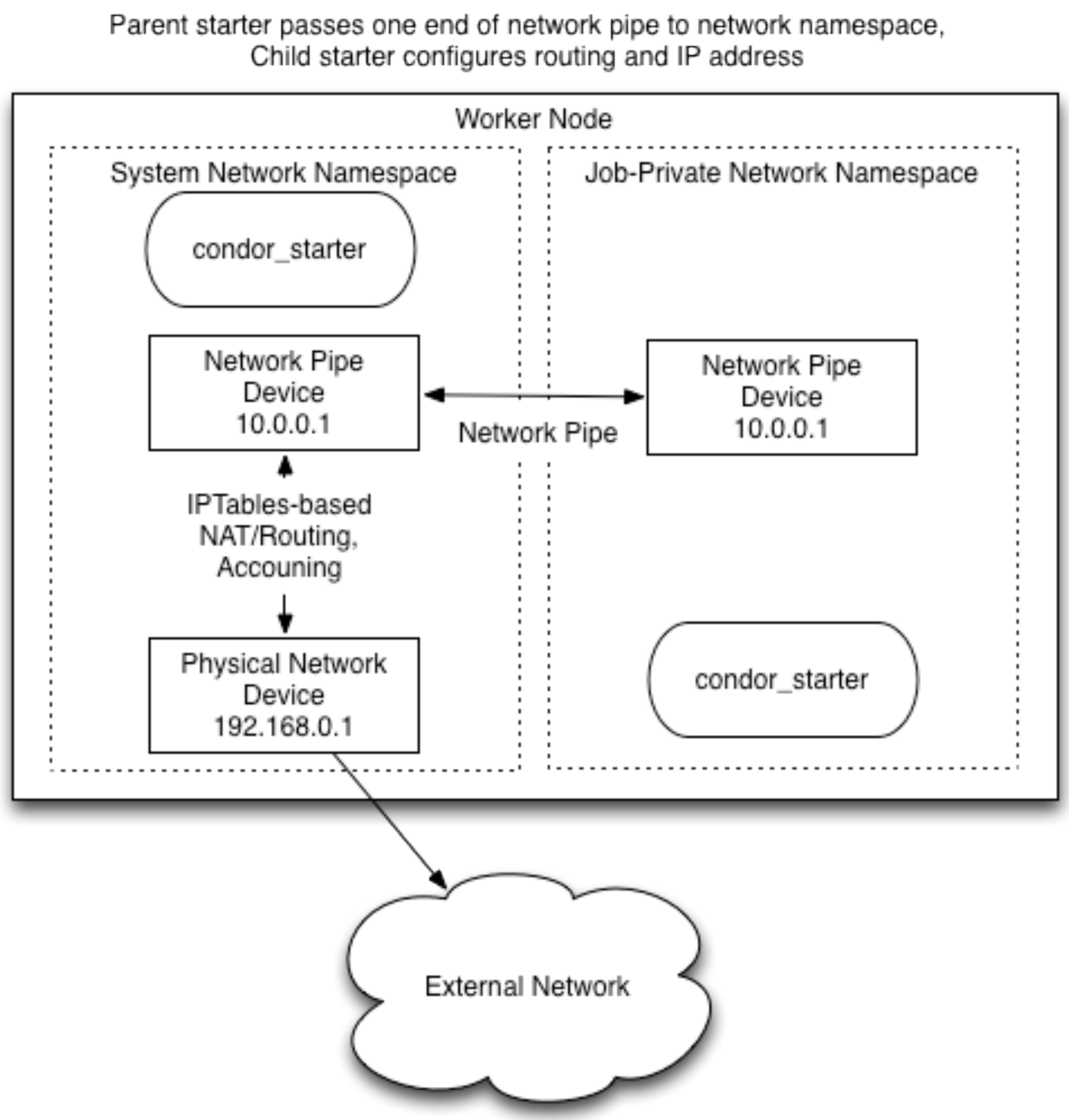


# Network Namespaces: Flipbook

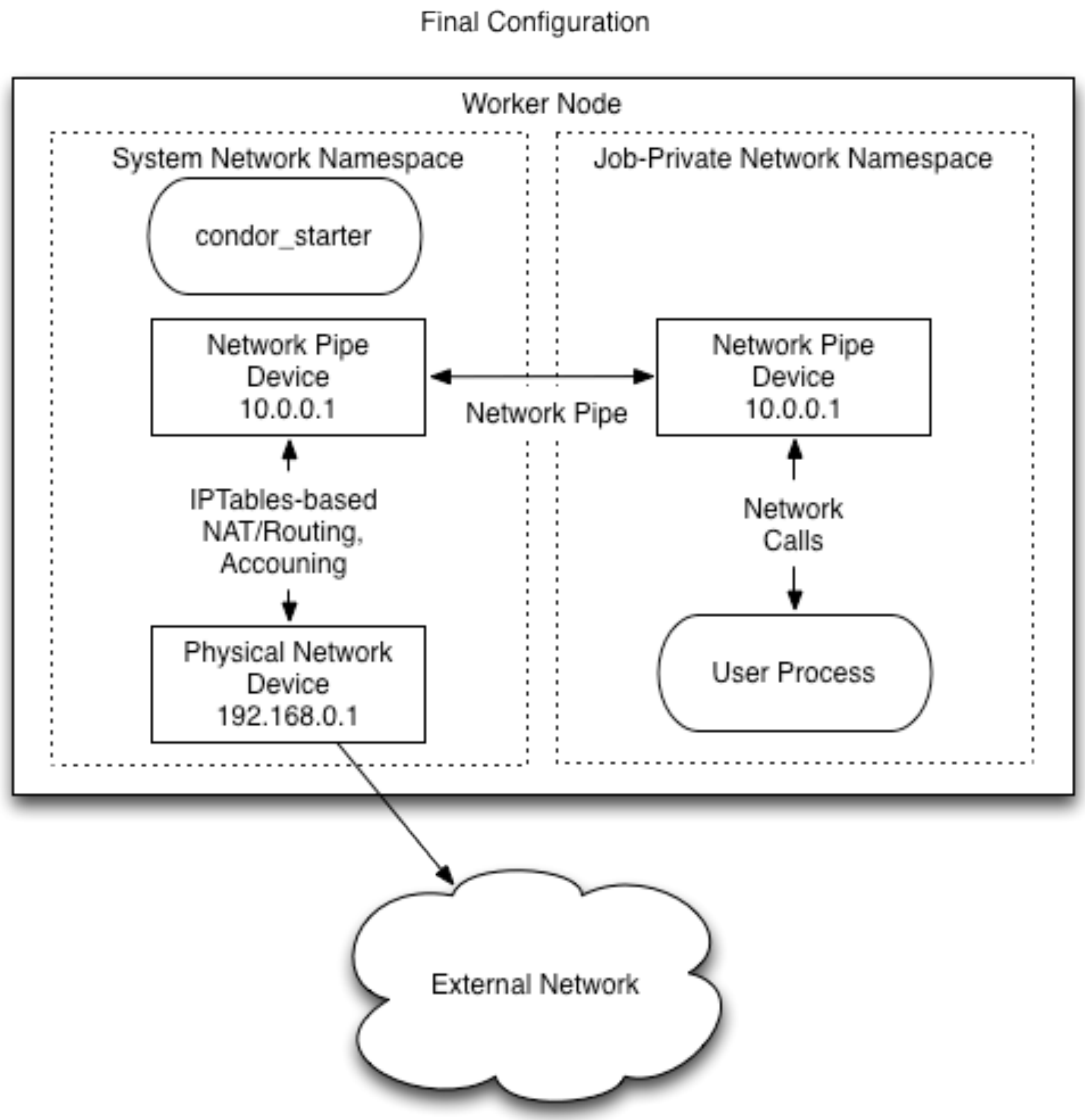
Starter forks new process with new network namespace



# Network Namespaces: Flipbook



# Network Namespaces: Flipbook



# Accounting Portion

- Each time a packet passes through an iptables rule, it is counted.
- While the job runs and finishes, iptables is periodically read, and each rule is published in the ClassAd.
  - The final ClassAd goes to the accounting system, and we can send the "EC2 bill".

# Resulting Chain

```
Chain JOB_12345 (2 references)
 pkts bytes target     prot opt in     out     source               destination
    3   579 ACCEPT     all  --  veth0  em1     anywhere             129.93.0.0/16        /* OutgoingInternal */
    0     0 ACCEPT     all  --  veth0  em1     anywhere            !129.93.0.0/16        /* OutgoingExternal */
    7   674 ACCEPT     all  --  em1    veth0   129.93.0.0/16        anywhere             state RELATED,ESTABLISHED /* IncomingInt
    0     0 ACCEPT     all  --  em1    veth0  !129.93.0.0/16        anywhere             state RELATED,ESTABLISHED /* IncomingExt
    0     0 REJECT     all  --  any    any     anywhere             anywhere             reject-with icmp-port-unreachable
```

## Resulting ClassAd Snippet

```
NetworkOutgoingInternal = 579
NetworkOutgoingExternal = 0
NetworkIncomingInternal = 674
NetworkIncomingExternal = 0
```

# Start Imagining...

- We used this for accounting. There are other possibilities:
  - **Per-job firewall** rules.
  - **Separate VLAN** for certain jobs.
    - Maybe have job traffic for "blessed users" bypass the campus firewall?
    - Have certain jobs connect to a network segment from a different site.
- Basically, this provides Condor with **a "hook" into the network.** Opens the doors to better-coordinated network management in Condor.

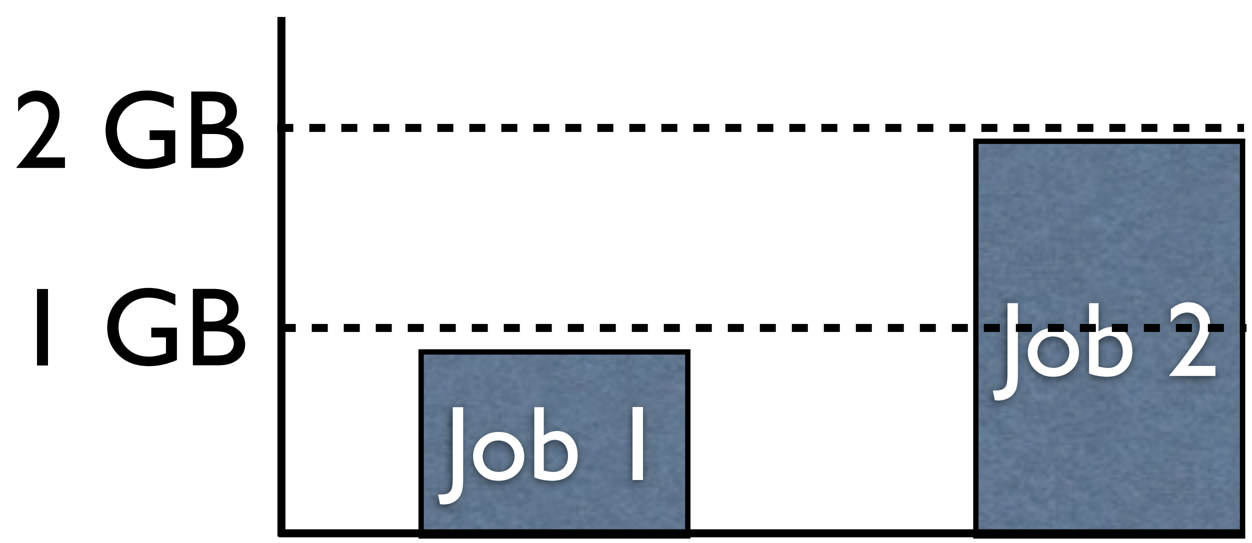
# Resource Management

- POSIX provides few "handles" for resource management.
  - We can measure resources used (accounting). Getting better.
  - However, what happens when the process uses more resources than requested? Outside killing the job, not much!
  - Thus, we encourage users to request the "worst case resource usage", leading to poorer utilization.

Not surprisingly, we'll investigate what the kernel has been up to!

# Memory Management

- Consider this situation in Condor: 2 jobs on a machine with 4GB RAM, asking for 2GB each. Consider the current usage:



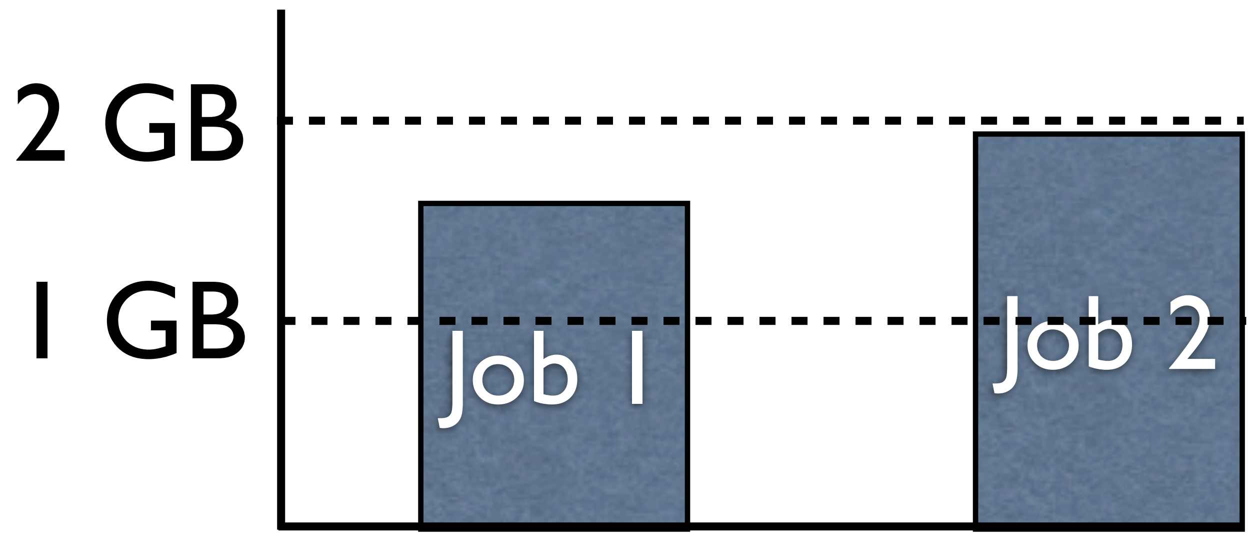

What happens if Job 2 allocates 1GB?

# Memory Management

- You could:
  - (Today) Kill off Job 2.
  - (Today) Do nothing. There is plenty of memory on the system.
  - (With memory cgroup) Swap out 1GB of Job 2, there's a hard limit.

# Memory Management

What about if Job 2 allocates 1GB now?
The job must go into swap!




Today, you can kill the job or
Have random pages from both jobs swapped out.
With cgroups, you can also have a "soft limit" where
Job 2 can take up 250MB more of RAM, but then only
have Job 2 swap.

# Memory in cgroups

- The memory cgroup provides both "soft" and "hard" limits.
  - Soft limits allow you to use idle RAM, but when the system goes into swap, the "nice" job might see some interruption.
  - Hard limits forces the "bad job" to start swapping once it hits 1-byte over the limit.
- In 7.9, this can be controlled by the startd using "MEMORY_LIMIT".

# Conclusions

- We tend to view "the world" as black or white: is it a batch job or a virtual machine?
  - By using containers, we have the ability to mix techniques normally associated with VMs into batch jobs.
  - The power of partitioning and isolation, without the headaches of VM management.
    - Basically, if you can do it in KVM (with respect to partitioning), you can do it in Condor!

# One More Thing

# Memory Cgroup and the OOM

- If the OOM-killer needs to kill a process in a memory cgroup, it can notify a subscribed process and wait for it to act instead.
- HENCE: condor_procd could manage the OOM-killer!
  - Sounds like I have coding to do...

# Etc

- What did I skip during this talk?
  - Block I/O.
  - CPU fairsharing and CPU sets.
  - Process killing with the freeze controller.
  - NFS mount statistics
- See a prior presentation here: http://www.biggrid.nl/news-and-events/singleview/back_to/news-and-events/article/e-infrastructure-colloquium/

# Mount Statistics*

```
device hcc-gridnfs:/osg/data mounted on /opt/osg/data with fstype nfs4 statvers=1.0
    opts:

rw,vers=4,rsize=32768,wsize=32768,namlen=255,acregmin=3,acregmax=60,acdirmin=30,acdirmax=60,
ard,proto=tcp,timeo=600,retrans=2,sec=sys,clientaddr=172.16.15.2,minorversion=0,local_lock=n
ne
    age: 568167
    caps:   caps=0x7ff7,wtmult=512,dtsize=32768,bsize=0,namlen=255
    nfsv4:  bm0=0xfdffafff,bm1=0xf9be3e,acl=0x0
    sec: flavor=1,pseudoflavor=1
    events: 60 1 0 0 0 3 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    bytes:  0 0 0 0 0 0 0 0
    RPC iostats version: 1.0  p/v: 100003/4 (nfs)
    xprt:   tcp 0 0 31 0 0 84 84 0 84 0
    per-op statistics
```

- /proc/self/mountstats provides a wealth of statistics - differs per filesystem, but NFS in particular provides a huge number of statistics (even for each op type!)
    - With FS namespaces, should be possible to start doing this "per job".

* Future work! What NFS statistics do you want to see from the batch system per-job?

# Block I/O

- Similar story for block I/O. We can now access the information *per job* instead of per system or per process.

```
[root@red-d15n2 ~]# cat /cgroup/blkio/blkio.io_serviced
8:48 Read 383
8:48 Write 0
8:48 Sync 383
8:48 Async 0
8:48 Total 383
8:32 Read 383
8:32 Write 0
8:32 Sync 383
8:32 Async 0
8:32 Total 383
8:16 Read 548172
8:16 Write 930060
8:16 Sync 996051
```

# Process Killing

- It’s a side-topic, but if the batch system leaks processes, you don’t manage the resource well!
  - With PID namespaces, if the initial process (PID=1) dies, all other processes in that namespace are wiped out.
  - If not using PID namespaces, we can use the “freeze” controller.

# CPU fairsharing!

- With the cpu cgroup controller, we can fairshare the system's overall CPU time.
  - You can violate the amount of CPU you were given if there's time available, but the amount allocated to each job.
  - The amount of CPU you get are relative to the number of shares you have in your sibling cgroups.

```
[root@red-d15n2 ~]# cat /cgroup/cpu/cpu.shares
1024
[root@red-d15n2 ~]# cat /cgroup/cpu/condor/cpu.shares
512
[root@red-d15n2 ~]# cat /cgroup/cpu/condor/job_1234_0/cpu.shares
128
```