Putting Condor into 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/

Containers

- Often, this is used to invoke a root-level process (init) and a separate mount table/chroot to run a completely virtualized machine.
  - While a separate namespace is seen, you can use *one* file system managed by *one* kernel. The overhead compared to a virtualized file system is negligible.
    - Nor do you have to give the process access to the physical device, like you would to optimize KVM I/O (see recent kernel DoS attacks and security holes for motivation).

Another Example

- The Linux kernel does lots of work to maximize memory utilization.
- When using virtual machines, the host kernel doesn't necessarily have the necessary information about the guest systems beyond the original resource request.
  - Virtualized applications can give hints to the guest OS about what it considers un-important, but the guest can't pass this information to the host.
  - Mild amounts of memory overcommit - *important in Linux* - can lead to priority inversion between guests. The host kernel doesn't know which pages are "nice, but unimportant" to the guests.

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? :)

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
```

Partitioning

- Containers typically take advantage of the resource partitioning features available in newer kernels.
  - These are typically implemented via "control groups", or "cgroups".
  - 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.

Goal: Containerize Condor

- We want to expose the various partitioning and management techniques in the Linux kernel to Condor, allowing it to better manage CMS 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

- Today, we offer isolation via giving a different UID to a batch job.
  - In fact, we've been doing this for a LONG time on Linux. We understand this model VERY well.
- Sometimes we do isolation via "social pressures" when appropriate.
  - Maybe this is incomprehensible to the WLCG crowd, but let's remember it's sometimes appropriate.

Isolation Models!

- Other isolation models exist though! Think again about what containers provide:
  - Process isolation ("PID Namespaces").
  - Filesystem isolation. Users see different mounts.

PID Namespaces

- When creating the process, Condor adds a new flag to clone().
- Processes are isolated to their own little world.
  - Cannot signal/mess with anyone else on the node, even if they are running with the same UID.
- Great, easy, but actually disruptive to mess with clone() in the Condor code because the child no longer knows its "correct" PID.
  - Actually, knowing the PID has been a place where this patch is contentious! Think about the possible debugging headaches!

Filesystem Namespaces

- One thing you could do is a chroot per batch slot.
  - Erm, not so nice!
  - Let's use the fact that mounts unique to the job are tied to the job's lifetime.
- Interesting mounts:
  - World writable dirs: /tmp, /var/tmp.
  - Working directories of other jobs from the same user

Chroots

- In Condor 7.7.5, users will be 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.
- Not feasible/convenient for isolation, but does allow you to provide SL5 environments to users who need it, but still run the newer kernel to do everything mentioned in this presentation.

Mount Under Scratch

- In Condor 7.7.5, we introduce 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!

Accounting

- We've done a poor job of accounting.
  - CPU accounting is actually OK:
    - Polling frequently enough, we can provide fairly accurate accounting except in malicious or "strange" c
  - Memory accounting is HORRIBLE!

Traditionally, Linux has some nice statistics *per process*. However, we want this *per job*. Let's discuss how!

Of course, who decides what is strange!

CPU accounting, pre Condor 7.7.0

- Every 15 seconds, a snapshot of all processes is taken.
  - Ancestry is established by looking at the parent PID for each process (and a few other techniques).
    - Of course, isn't very perfect!
  - Look at how much CPU or memory is used by each process.
  - Total "per-job" is the sum of all processes in the job.

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
```

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
```

Network Accounting

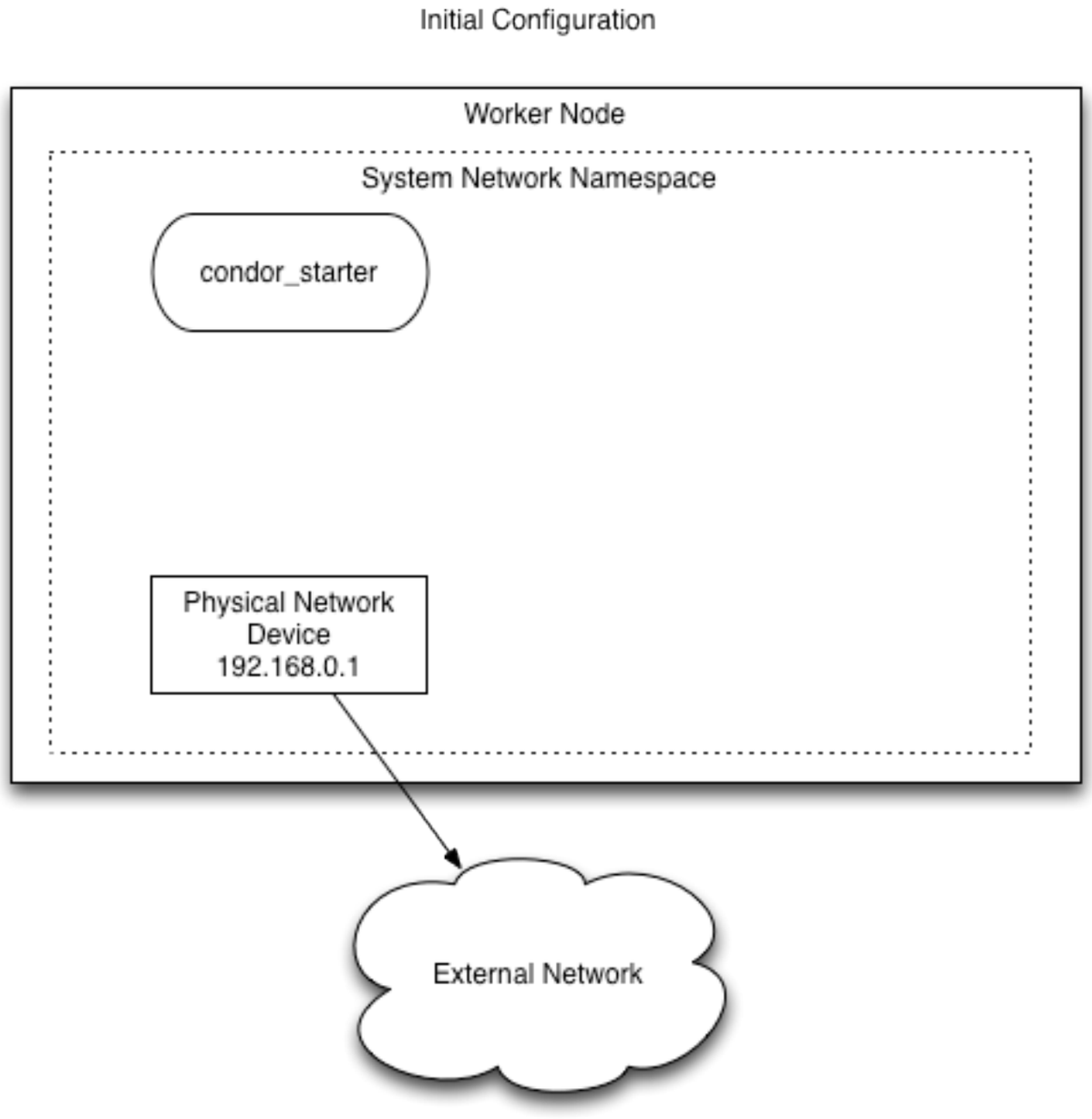
- We are extremely interested in knowing the per-job network I/O figures:
  - Helps us understand if site planning is right.
  - Give appropriate information back to users - and trace a bit about what they did on the network.
  - Compare costs, dollar-for-dollar, against EC2.

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

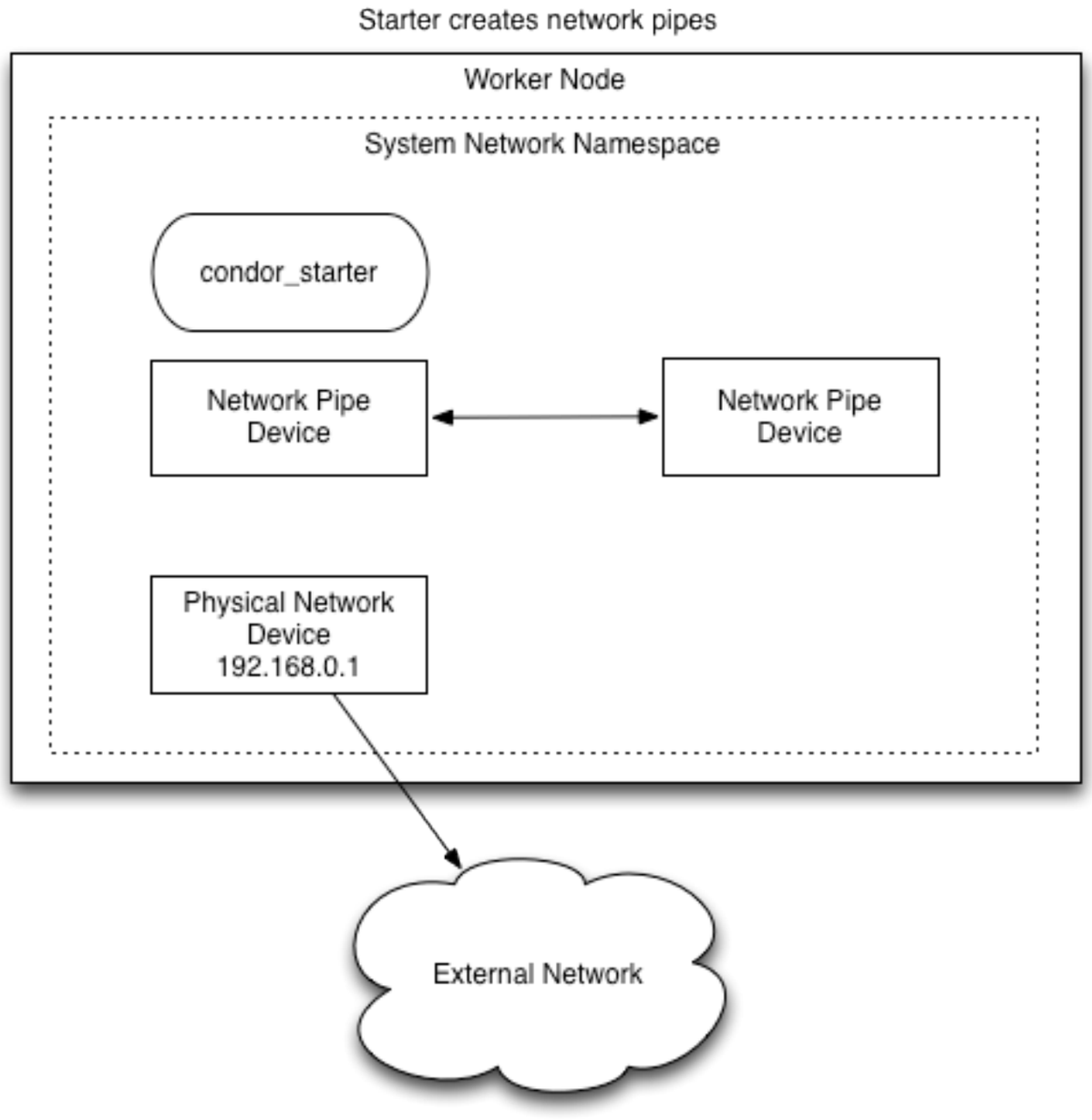
Network Namespaces

- 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...

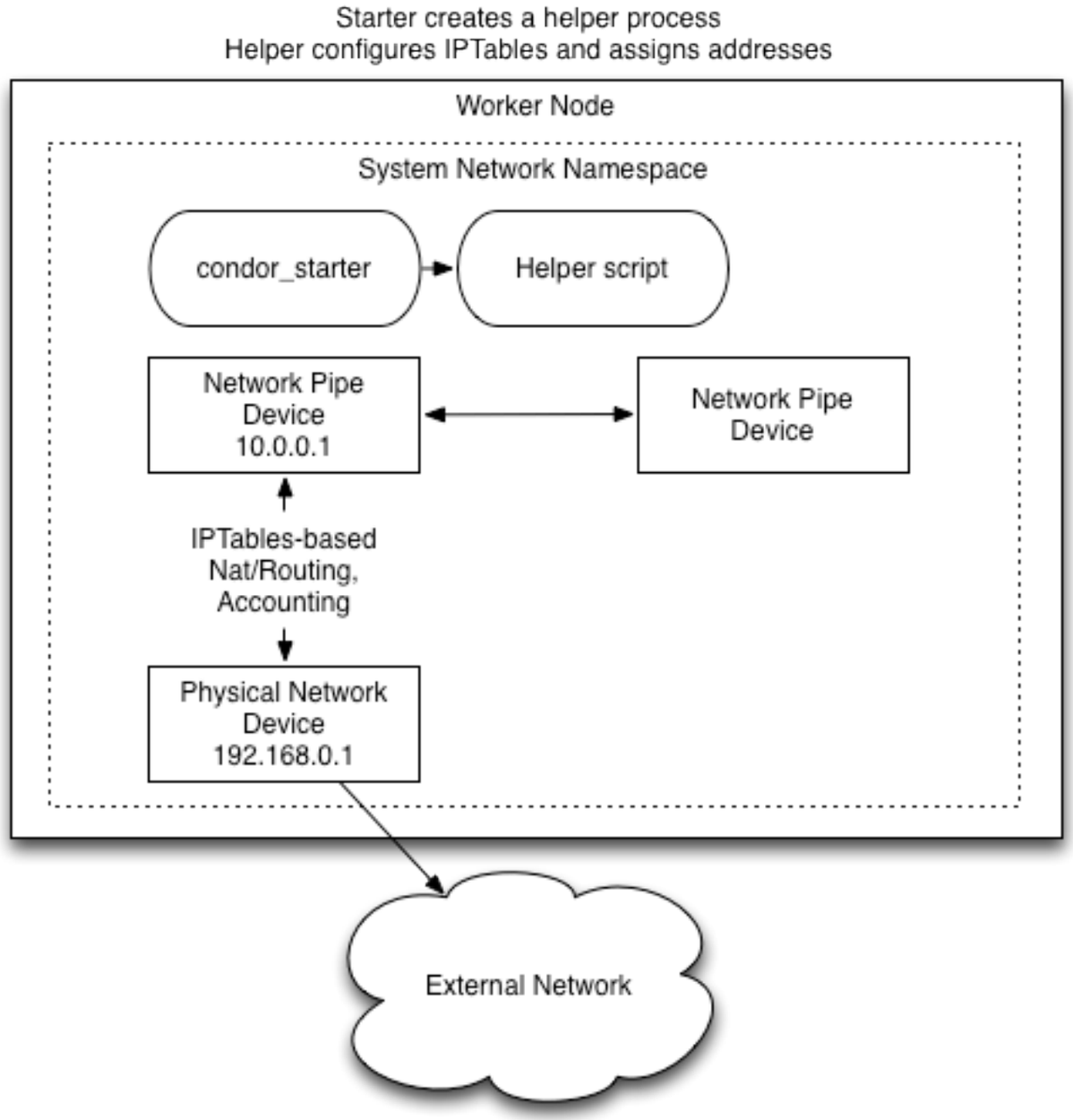
Network Namespaces: Flipbook



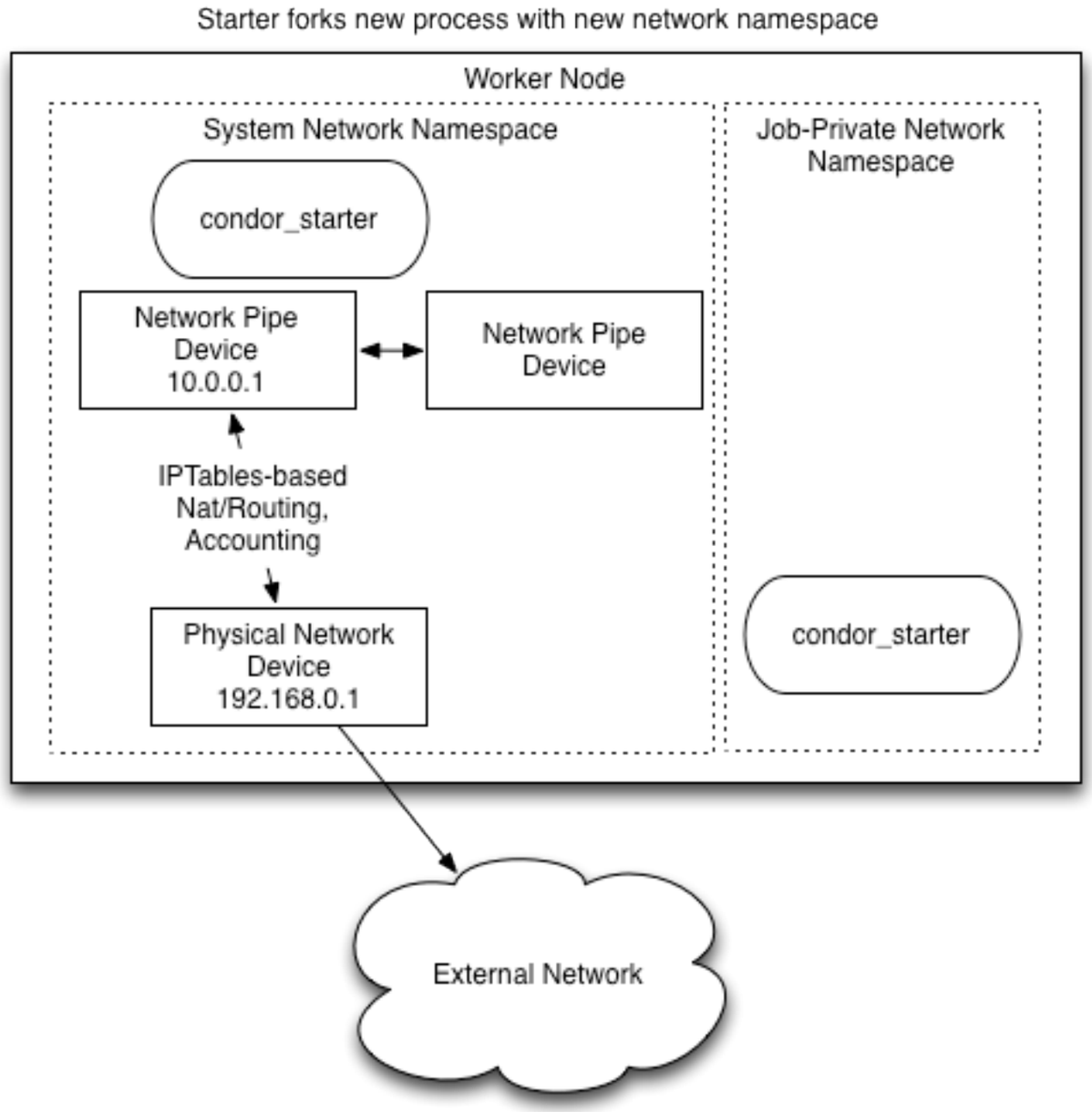
Network Namespaces: Flipbook



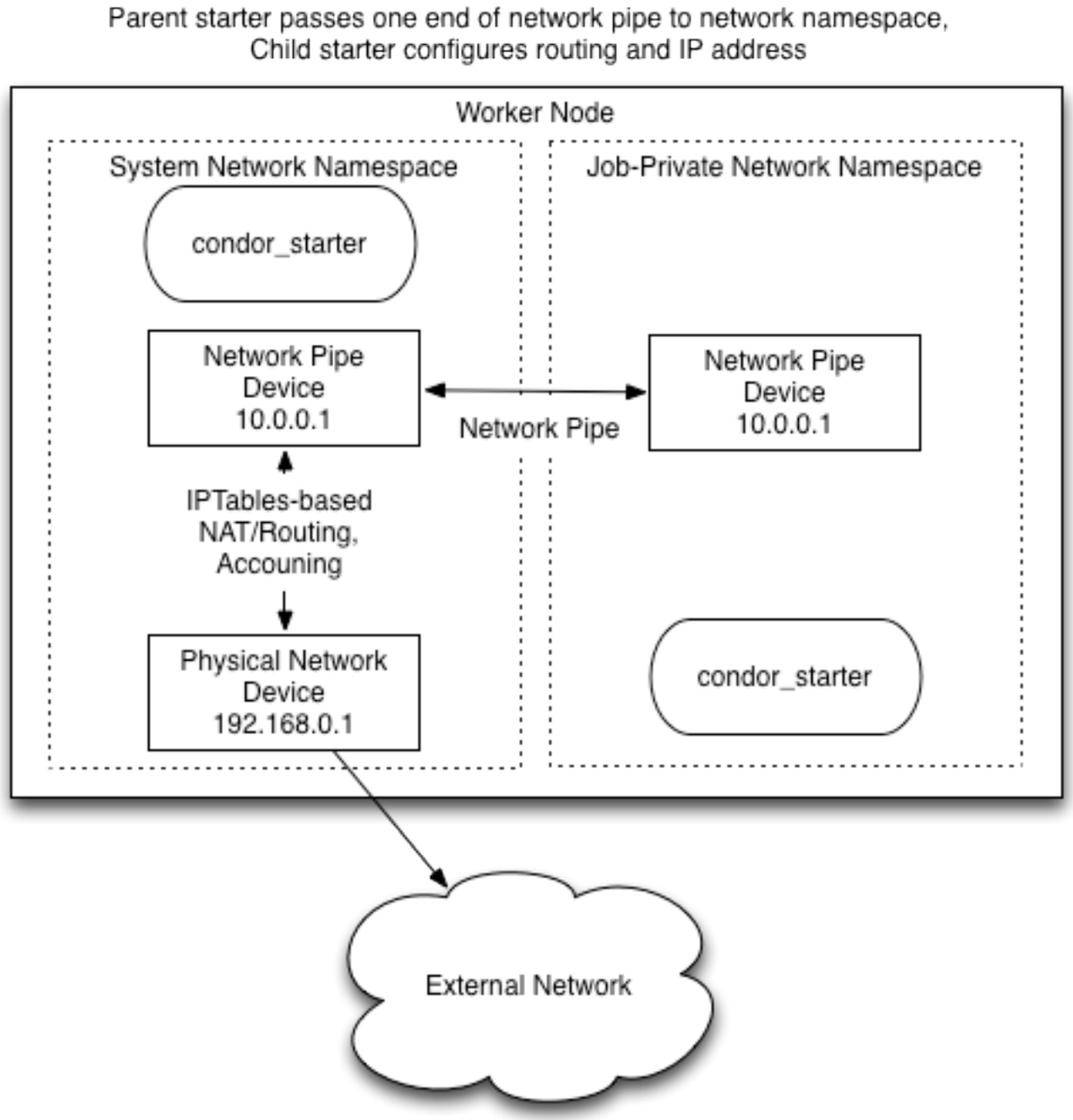
Network Namespaces: Flipbook



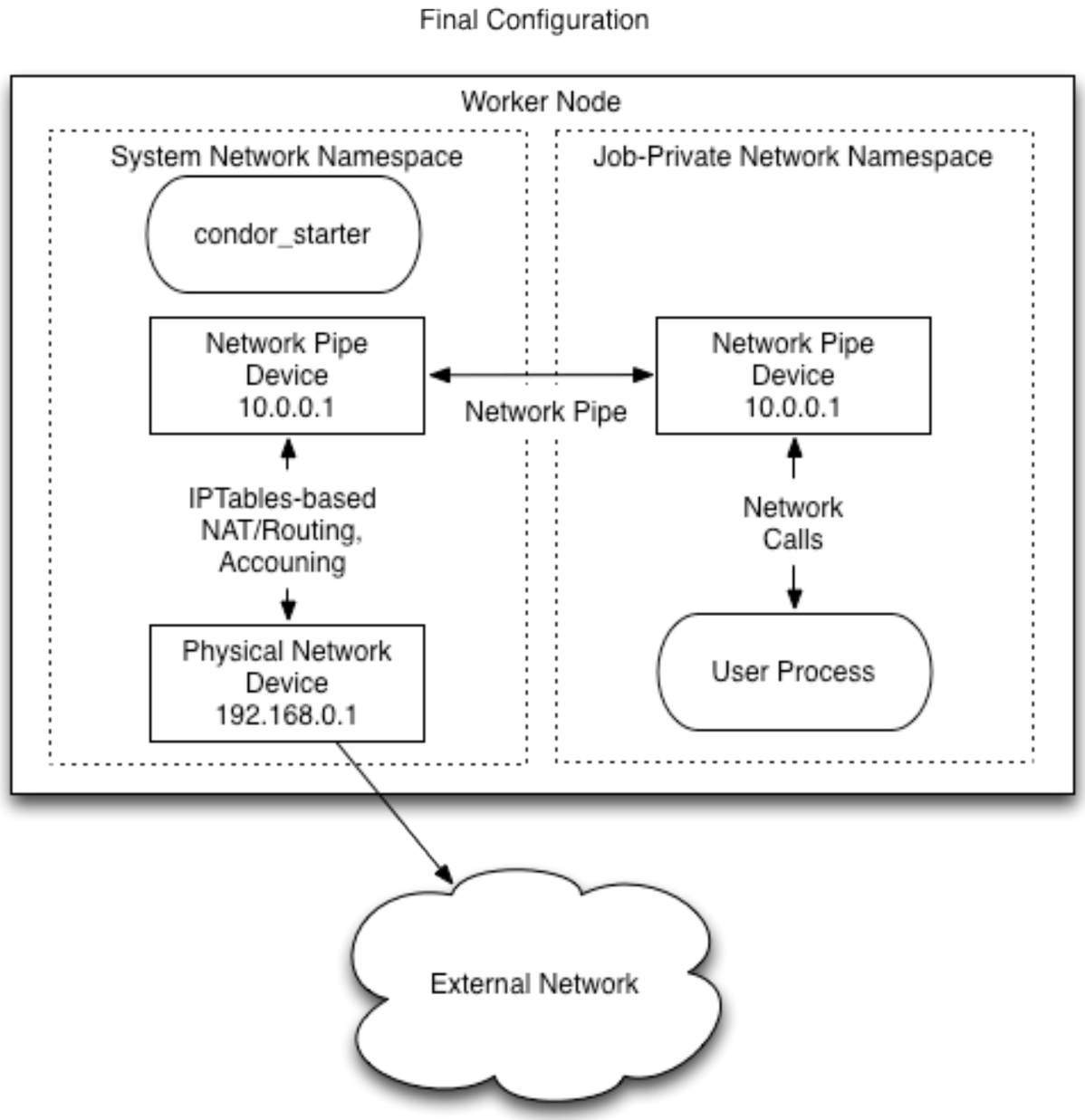
Network Namespaces: Flipbook



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
```

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 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?

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!

CPU sets / affinity

- CPU sets are a fairly well-known technique, used by most batch systems.
  - If I ask for one core, I get access to precisely one core.
  - Very hard partitioning - if I sit idle, no one else gets it. Can adversely affect utilization.

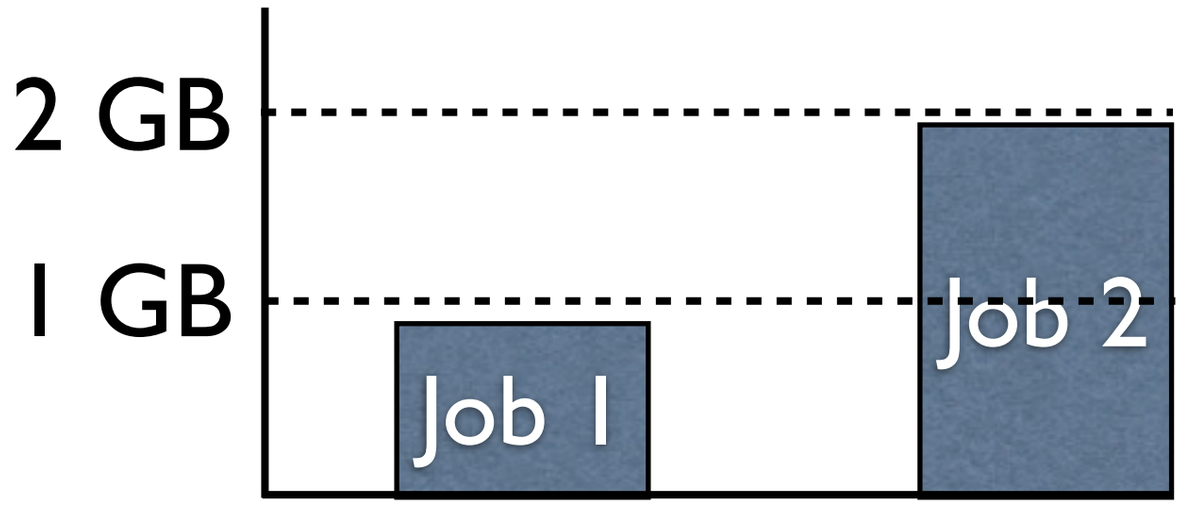
CPU fairsharing!

- 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
```

Memory Management

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



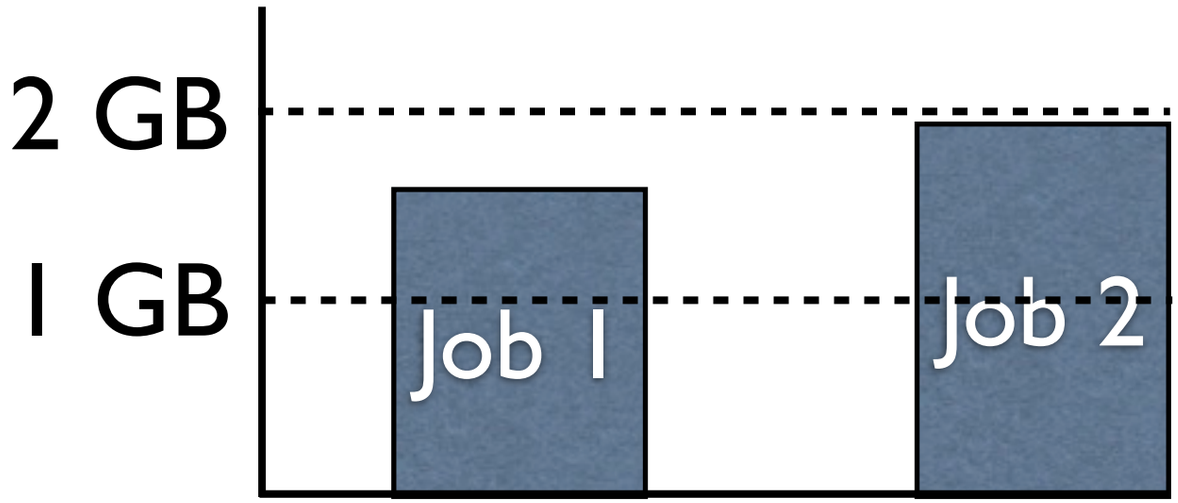
What happens if Job 2 allocates 1GB?

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.

Block I/O

- Perhaps unsurprisingly, the blkio controller can also do fairshare.
- Currently, this is not recursive - you cannot put Condor into /condor, and then fairshare between the jobs. The jobs must go at block level.
- Unknown: how well does swapping get "fairshared".

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.

Freeze Controller

- Put the job in a cgroup with the freeze controller.
  - Write “FREEZE” into the correct file, and the kernel will immediately remove all processes from scheduler queue.
  - Send a kill signal to all processes.
  - Write “THAW” into the correct cgroup file, and the signals are delivered atomically.

Network Management*

- The network accounting previously shown gives us a fantastic handle: a network device per job.
  - There are newer technologies - e.g. OpenFlow - that expose the network as an API we can manipulate.
  - Would be possible to dynamically create a VLAN for a user's set of jobs.
  - Or, trivially, assign jobs to a static VLAN (CMS jobs versus local jobs). No changes needed to the network accounting work besides the script to create the iptables chains.

*(Some) Future Work

What's for Real?

- Things in a Condor release: Cgroups for CPU, block I/O and memory accounting.
- Things committed for next release: MOUNT_UNDER_SCRATCH, chroots
- Things proposed to team under evaluation: Cgroups for resource management.
- Patches rejected by Condor team: PID namespaces
- Things on a github playground: Network namespaces
- Things being explored, not implemented: Dynamic network manipulation with OpenFlow from the batch system, per-job NFS mount statistics.

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!

Conclusions

- This is completely different than clouds and virtual machines.
  - Still a great long life in front of these things! Completely customize the OS, or your science depends on some odd kernel feature. Let's recall the costs of maintaining VMs though!
  - Condor will still launch virtual machines instead of a process if asked.
- However, I believe this work makes significant progress, and maybe make sites "less interested" in clouds!