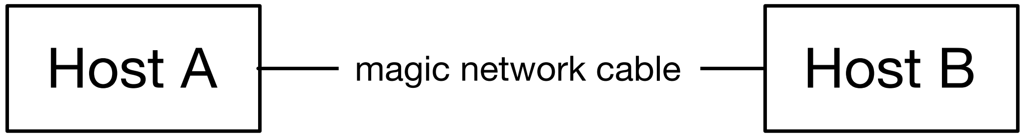
Networking and High Throughput Computing

Garhan Attebury
HTCondor Week 2015

A simpler time

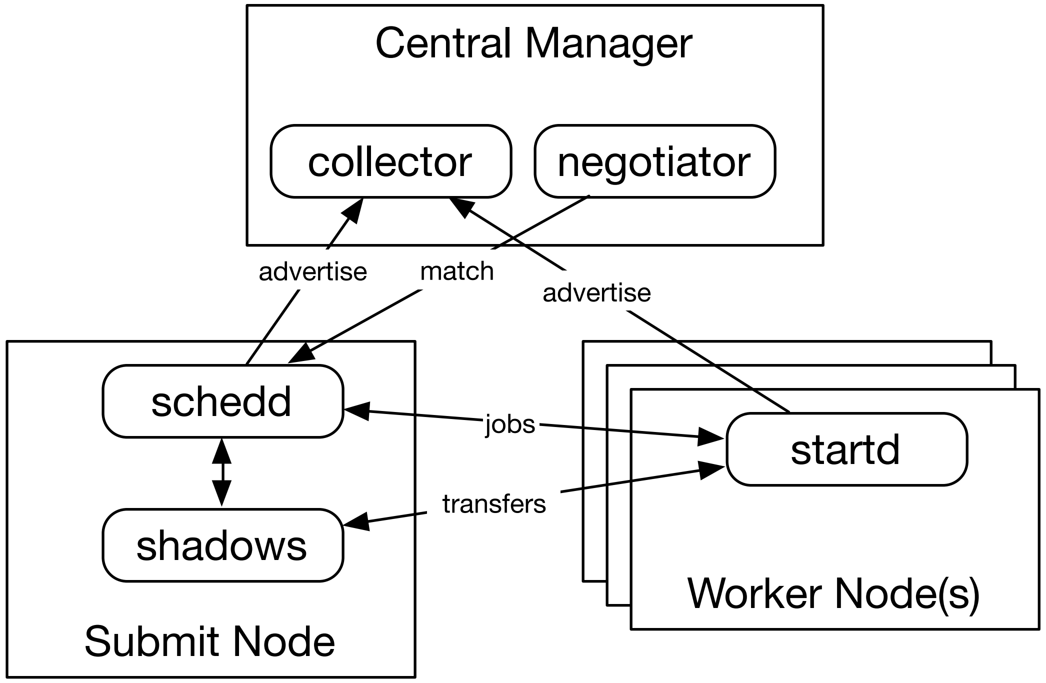
… or at least blissfully ignorant time



What network communication?

- Single host, not much
- Workers and manager
- … multiple schedds
- … dedicated servers
- … multiple pools, global grid, 'clouds'
- … and then do it all at scale

HTCondor Daemons



“Listeners everywhere…”

- Greg Thain
HTCondor and Networking presentation @ CERN

Problems with all this communication

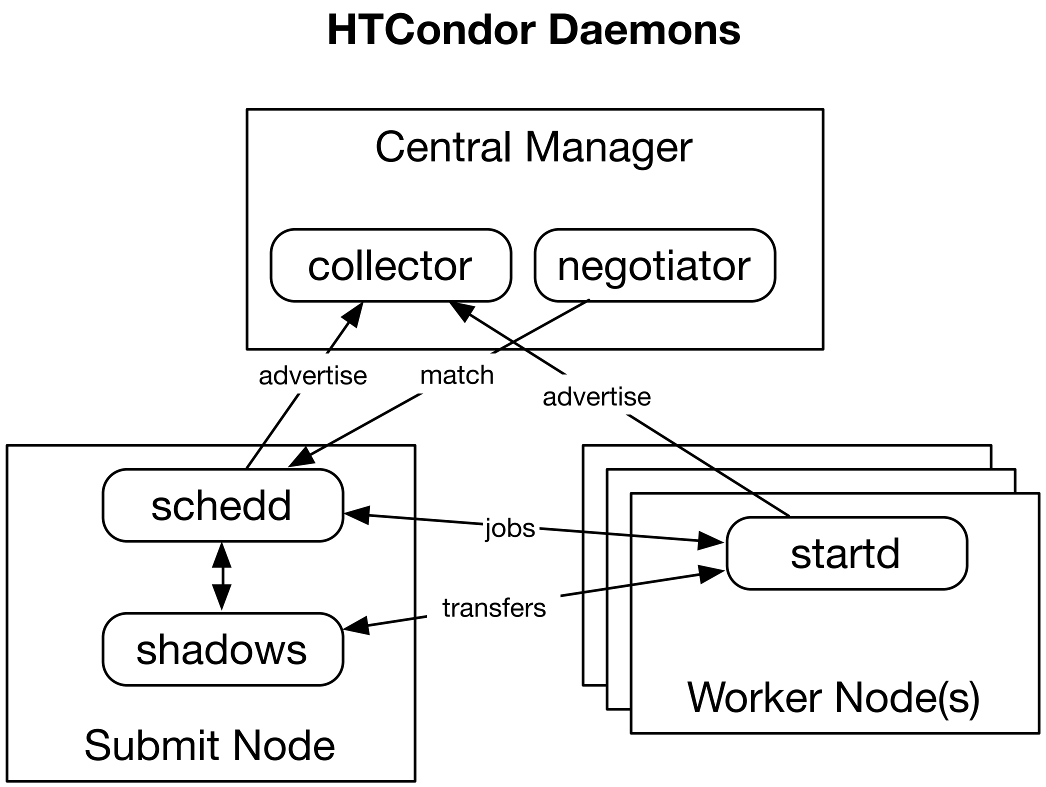
- Daemons can’t communicate due to firewall
- Port exhaustion (stateful firewalls / NAT)

Dealing with firewalls

- As usual, knobs to turn

  HIGHPORT, LOWPORT
  IN_LOWPORT, IN_HIGHPORT,
  OUT_LOWPORT, OUT_HIGHPORT

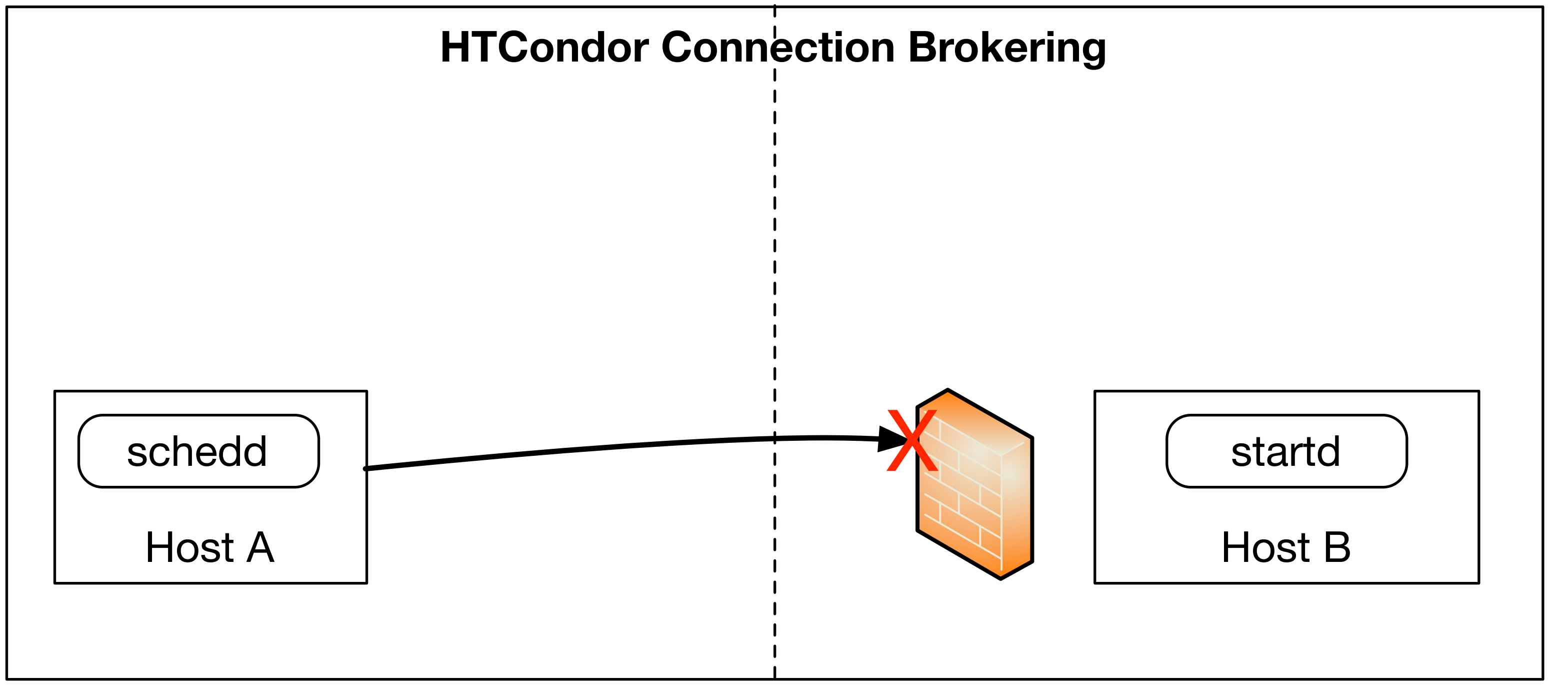
- Firewall configuration
- … config creep
- Enter CCB



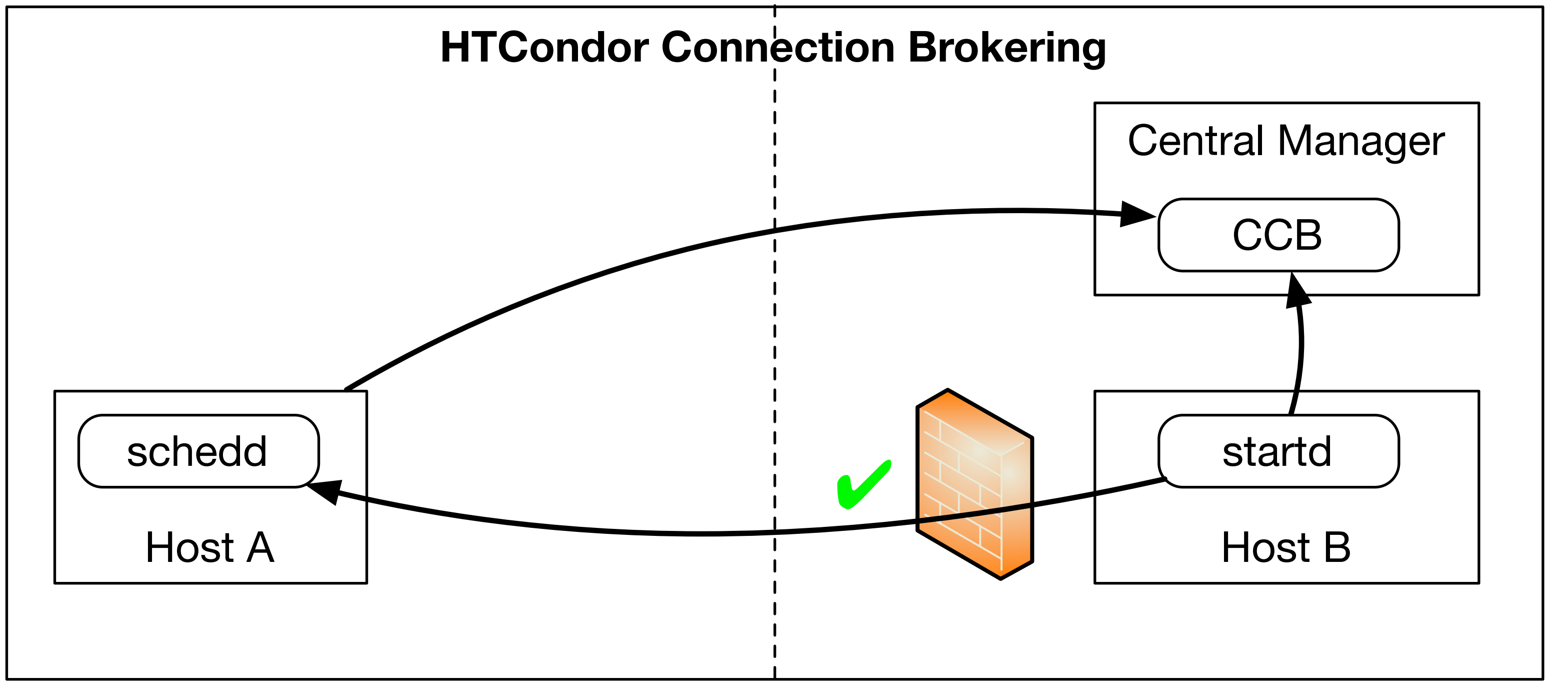
HTCondor ~~Couch~~ Connection Brokering (CCB)

- Bypasses firewall by reversing connection
- Allows communication between private and external daemons
- Runs on one machine (often collector / central manager)
- Caveat: Doesn't work with standard universe
- Caveat: Can't help when all nets are private
- Bonus: Can avoid CCB when private net exists

- schedd on A cannot reach startd on B



- startd on B registers with CCB using CCB_ADDRESS
- schedd on A requests that startd on B ‘calls back’



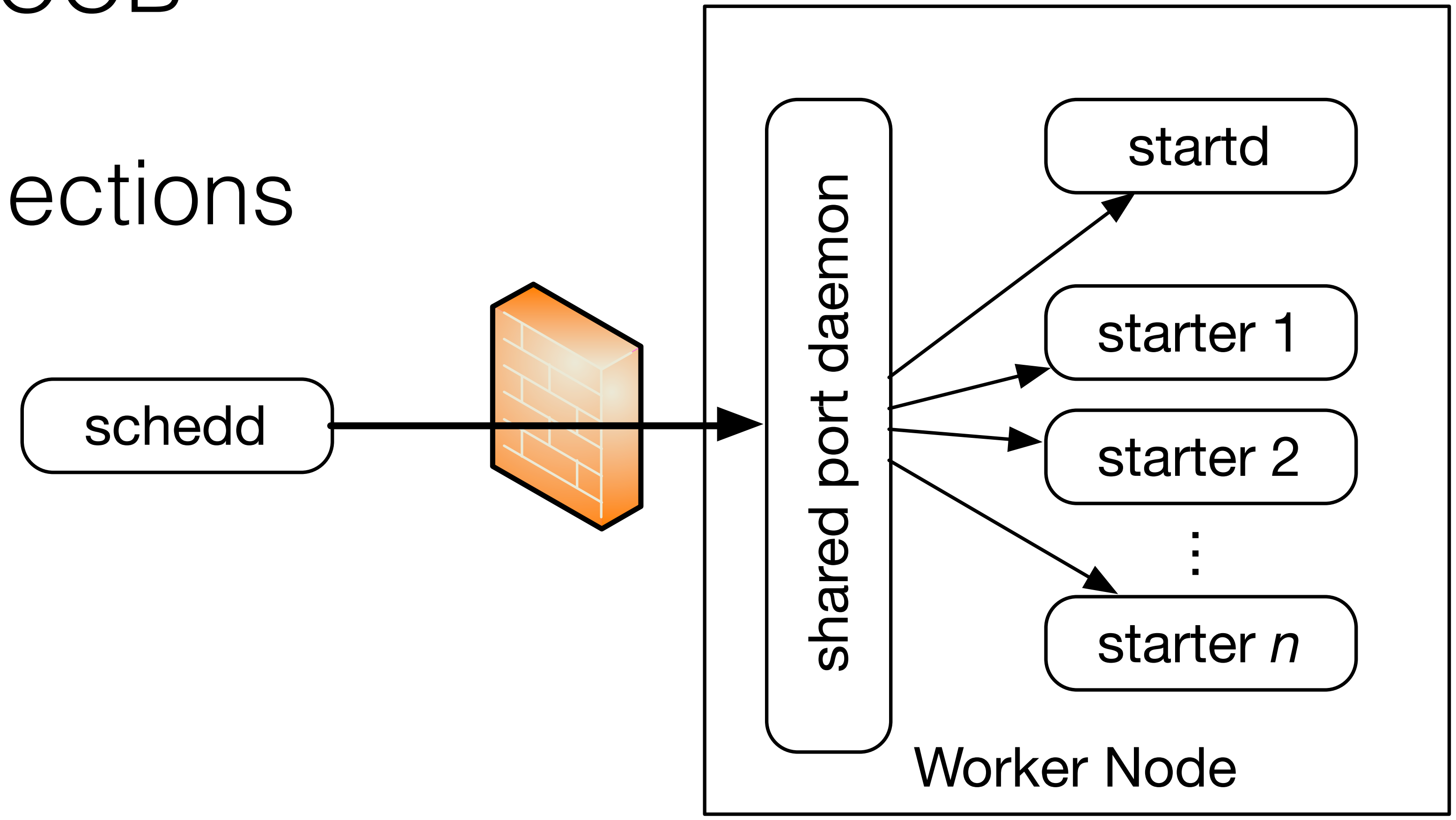
- after callback, two way communication possible

Port exhaustion / NAT annihilation

- Lots of daemons = lots of connections
- Worker: 5 + (5 * NUM_SLOTS)
  Scheduler: 5 + (5 * MAX_JOBS_RUNNING)
- Turnover rate / limited ephemeral ports
- <insert ‘enterprise’ NAT joke here>
  <grumble about conntrack here>
- Enter condor_shared_port

condor_shared_port

- Uses single listener port for all daemon communication: USE_SHARED_PORT = True
- Works with CCB
- Fewer connections



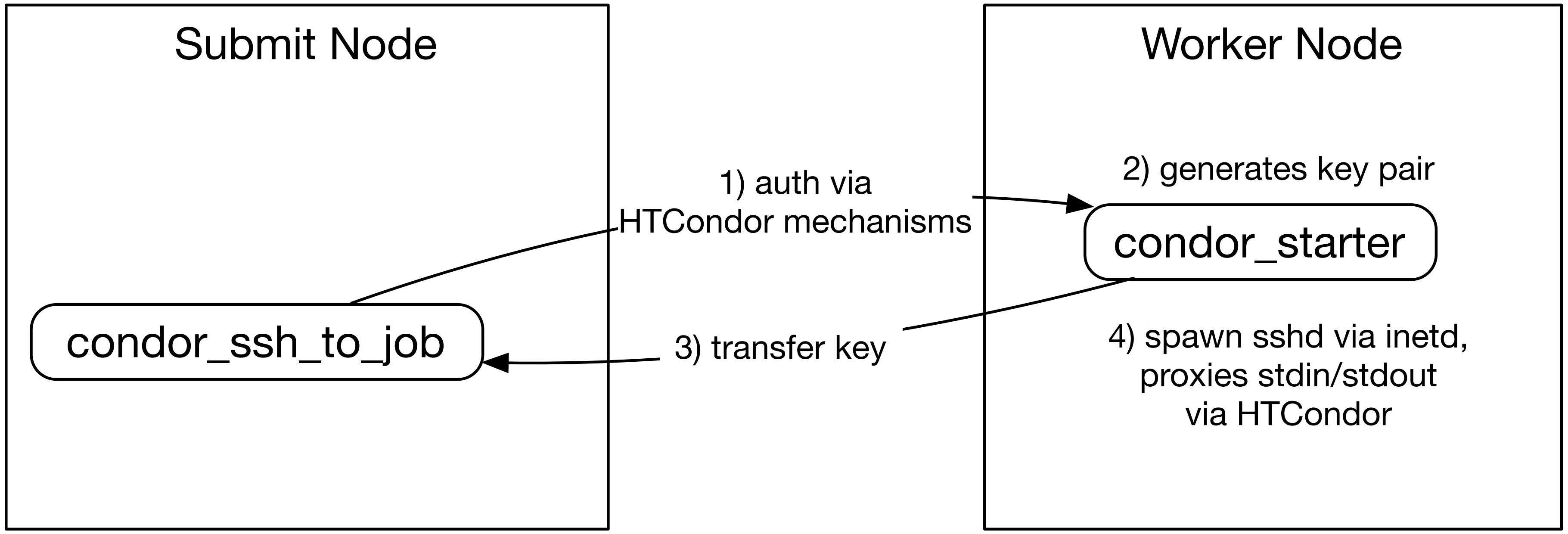
Bonus tidbits

- Knobs to control listening: BIND_ALL_INTERFACES, NETWORK_INTERFACE= (advertises only 2 - at least for now)

- Knobs to survive multihome insanity: PRIVATE_NETWORK_INTERFACE, PRIVATE_NETWORK_NAME

- Proxying: TCP_FORWARDING_HOST

More tidbits: condor_ssh_to_job


Submit Node
Worker Node
1) auth via HTCondor mechanisms
2) generates key pair
condor_starter
condor_ssh_to_job
3) transfer key
4) spawn sshd via inetd, proxies stdin/stdout via HTCondor

- sshd spawned on worker has stdin/stdout attached to the TCP connection from condor_ssh_to_job
- no sshd process listening on the network or running as a user other than the job owner
- works with CCB (causing great paranoia)
- works with EC2 in Grid Universe (more traditional mechanism)

IPv6

- There are knobs for that:
ENABLE_IPV6 = true
ENABLE_IPV4 = false

- Dual stack (mixed-mode) is required on central manager

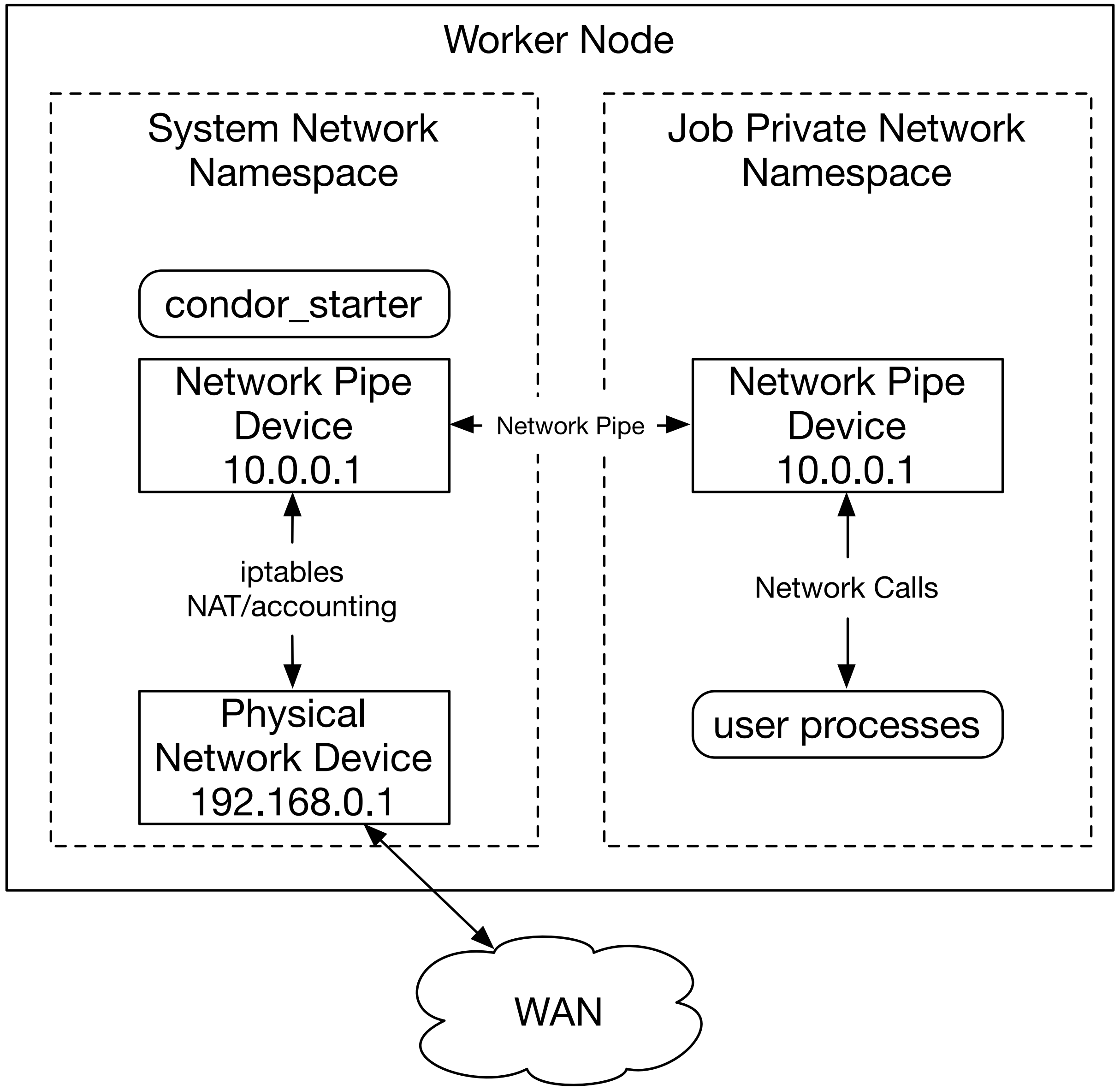
- Still some … strangeness
Eventually of course it should “just work”

- It’s new, it’s under development…
use [bleeding edge] for best results

- “Production” at UNL with dual stack 8.3.1 and 8.3.5 hosts

Future of (not entirely) lies

- Network namespaces
- Accounting
- Network automation (circuits, openflow, etc)
- LARK project


Worker Node
System Network Namespace
Job Private Network Namespace
condor_starter
Network Pipe Device 10.0.0.1
Network Pipe
Network Pipe Device 10.0.0.1
iptables NAT/accounting
Network Calls
Physical Network Device 192.168.0.1
user processes
WAN

Network Accounting

- Per job network accounting
  - Networks *are* a resource, and eventually we might treat them that way
  - Usage metering / triggering
    - Finite resources such as EC2

Network Automation

- Per job policy based automation

  - Job requires no connectivity
  - Job needs traffic priority
  - Job requires special VLAN placement

- Security considerations

- Dynamic circuit allocation
  (OSCARS, OpenFlow, etc)

- Network namespaces work, but are by no means common (yet)

- Accounting is ‘easy’, rest is largely dependent on external environment and needs of the application and underlying science

- HTCondor can provide the means, but what will actually be done in practice is still unknown

[almost entirely empty question slide]