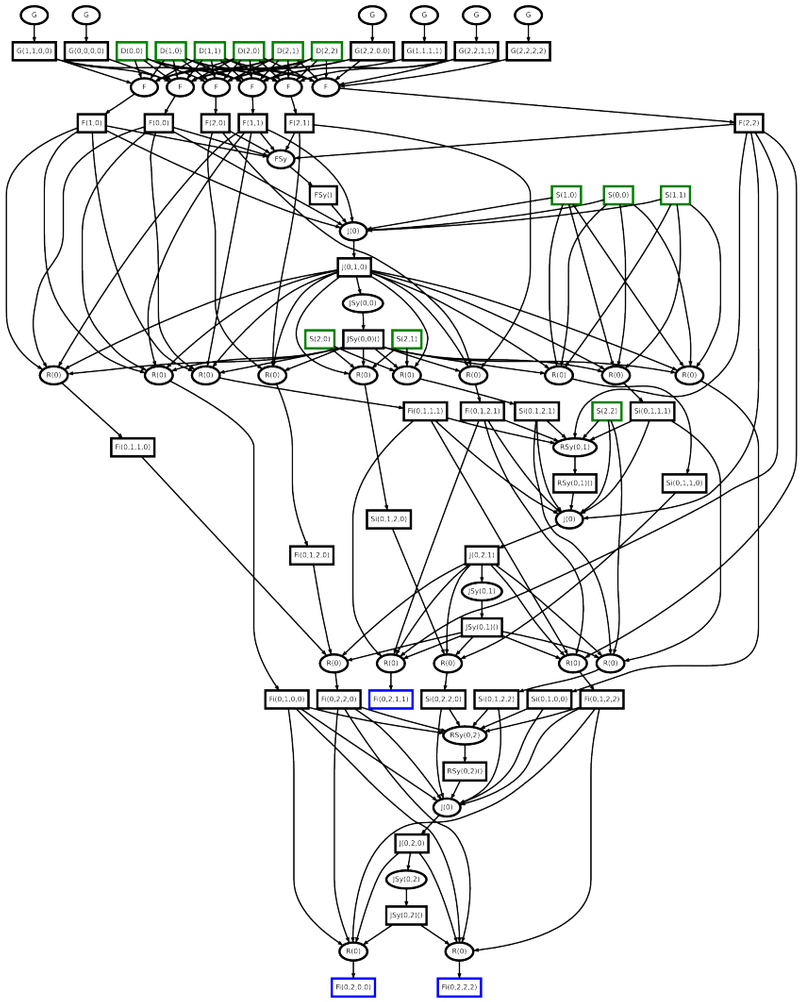
BRASIL
Basic Resource Aggregation Interface Layer

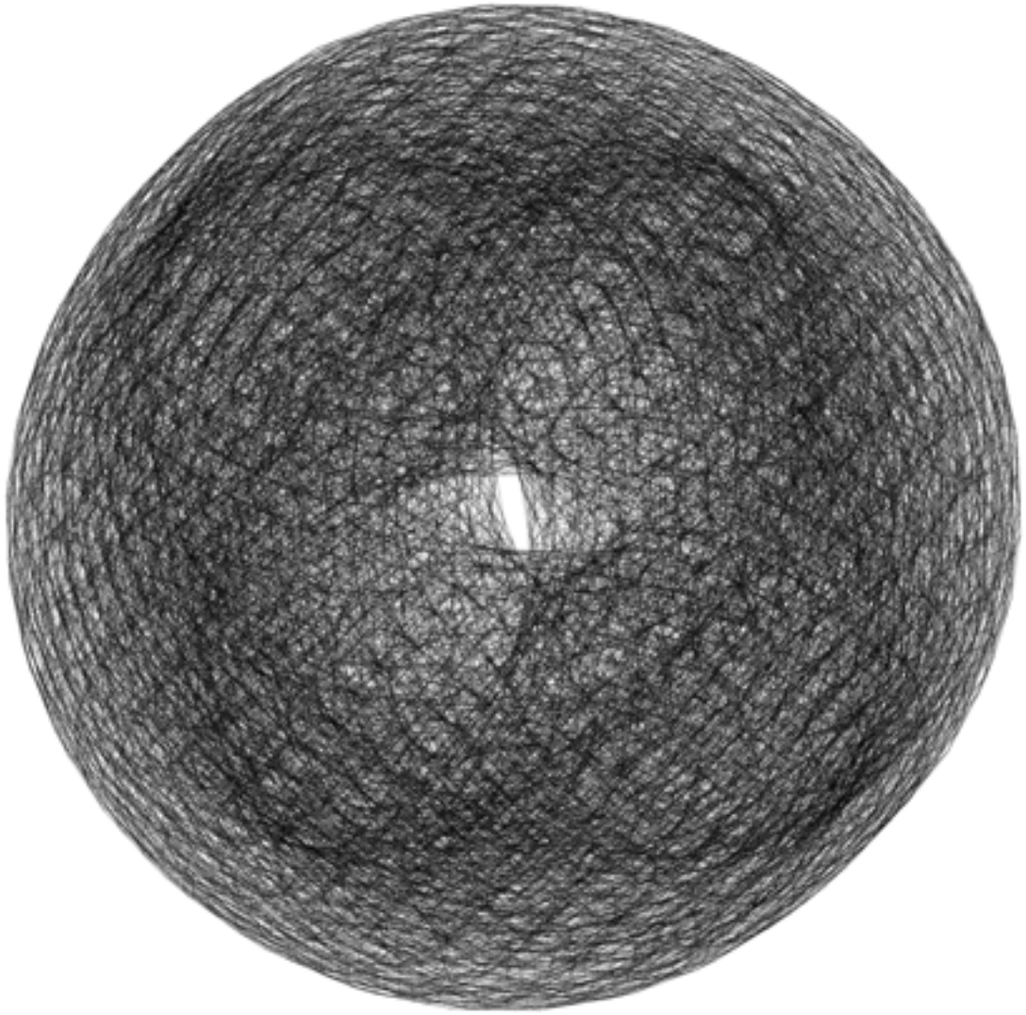
Eric Van Hensbergen
(bergevan@us.ibm.com)
IBM Research Austin

Pravin Shinde (now at ETH Zurich)
Noah Evans (now at Bell-Labs)

Motivation



Data Flow Oriented HPC Problems

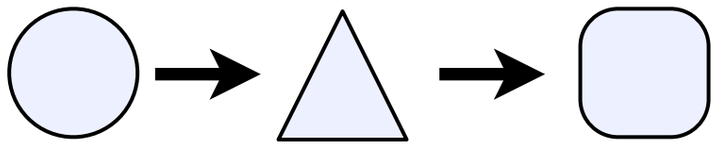


64,000 Node Torus

PUSH Pipelines

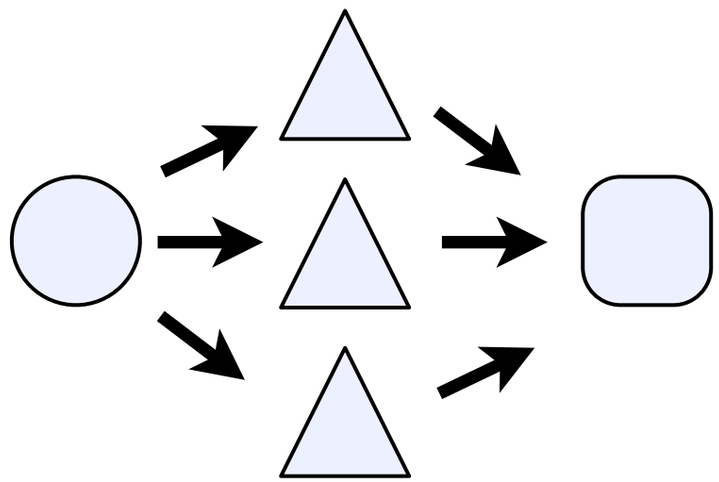
UNIX Model

a | b | c



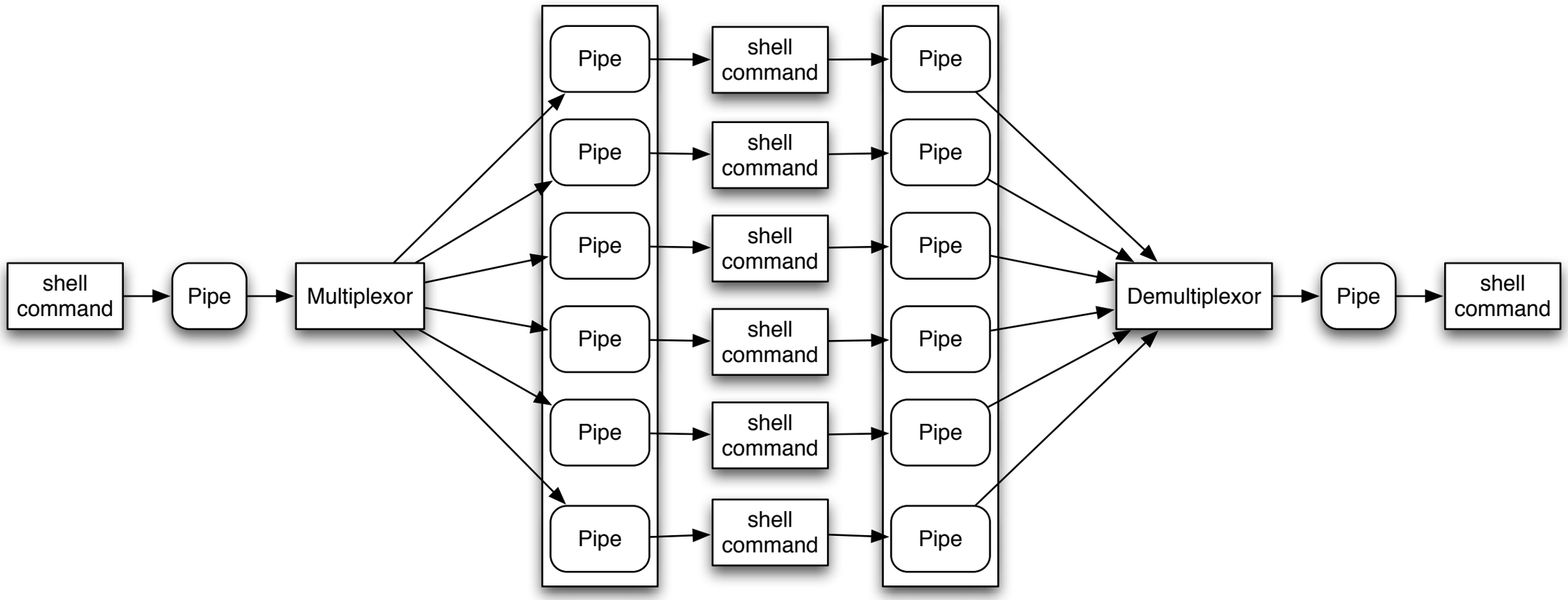
PUSH Model

a |< b >| c

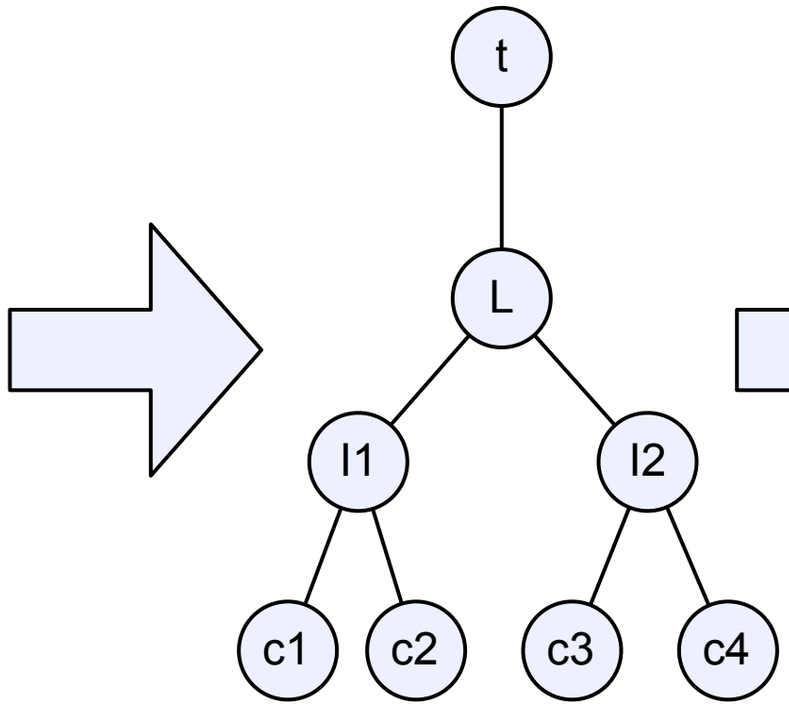


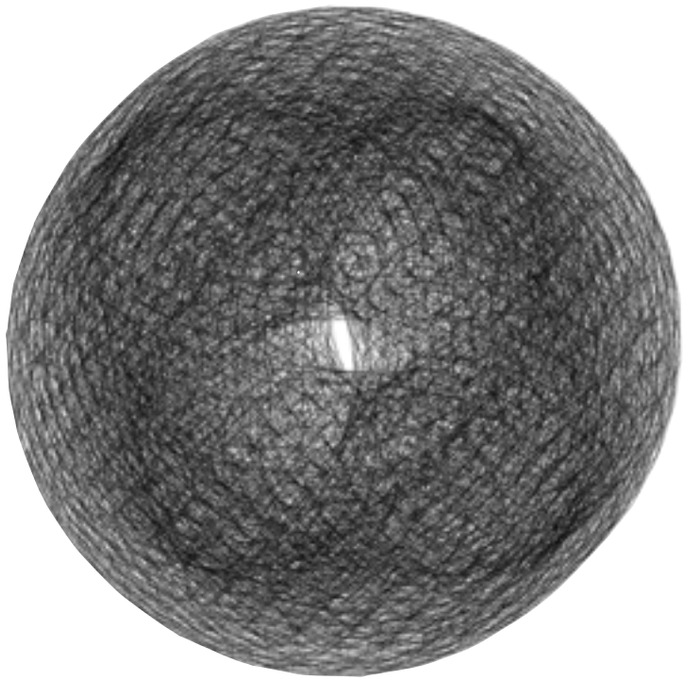
For more detail: refer to PODC09 Short Paper on PUSH Dataflow Shell

PUSH Implementation



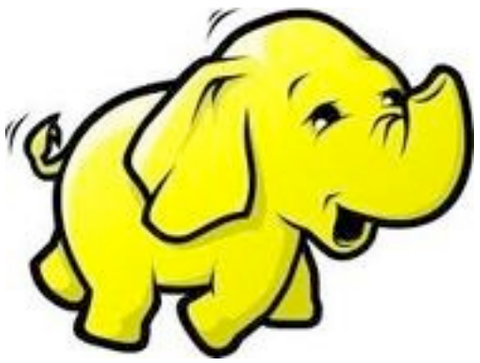
Back to Motivation: 64,000 Nodes





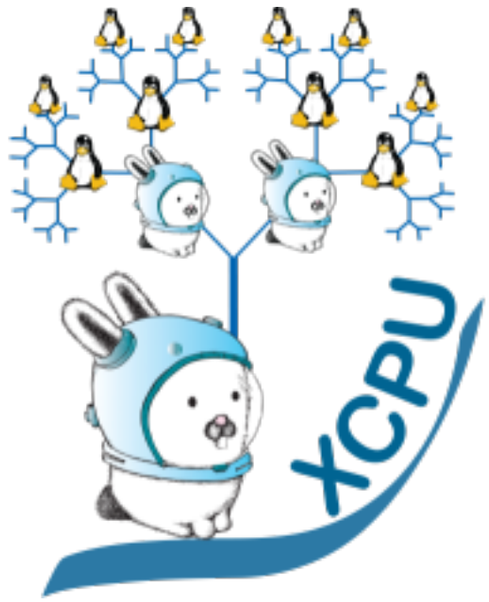
Related Work/Background Work

- MPI
  - "A Compositional Environment for MPI Programs"

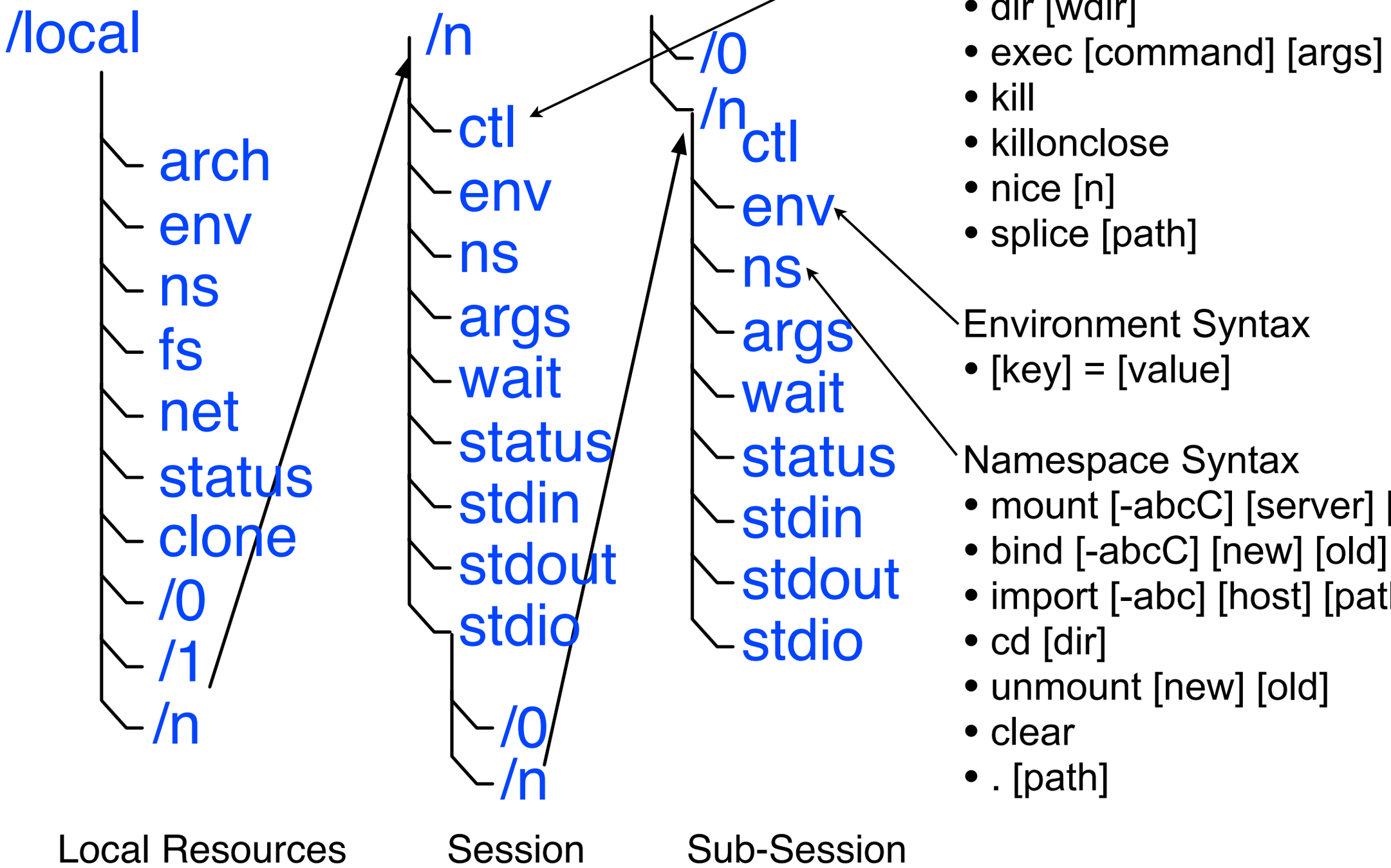


- Hadoop & Other Map/Reduce Solutions

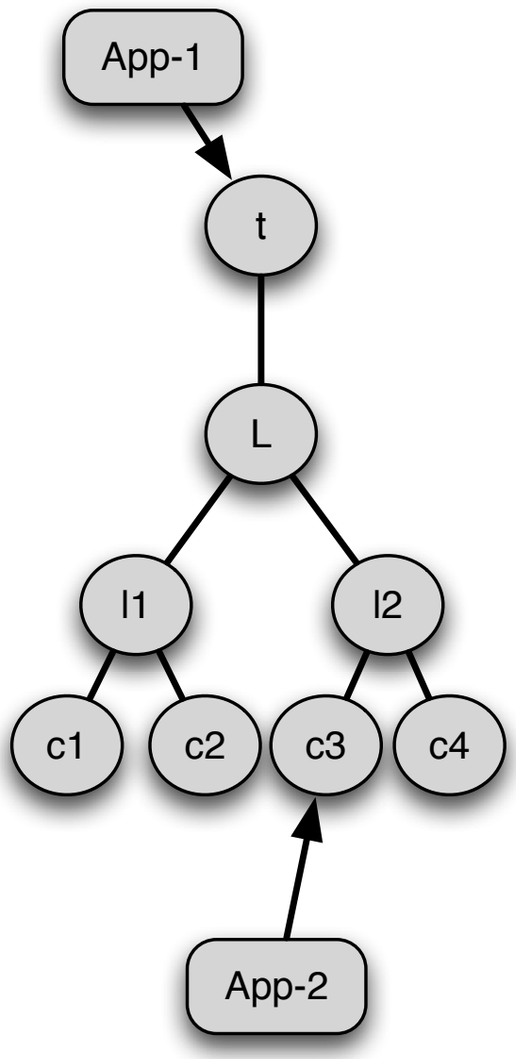
- cpu (from Plan 9) - http://plan9.bell-labs.com/magic/man2html/1/cpu
- xcpu (from LANL) - http://www.xcpu.org/
- xcpu2 (from LANL)



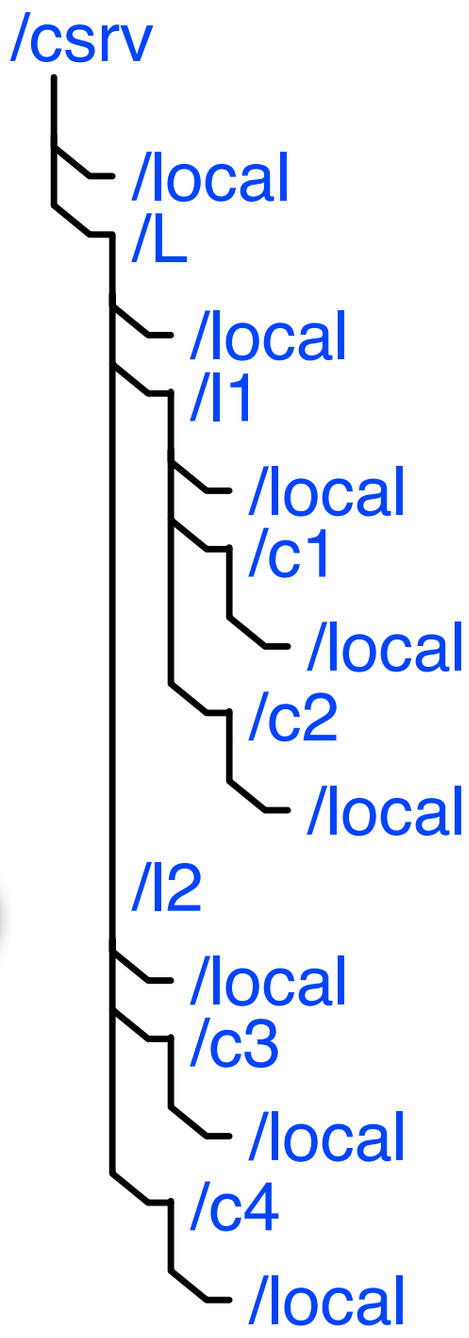
File System Interfaces



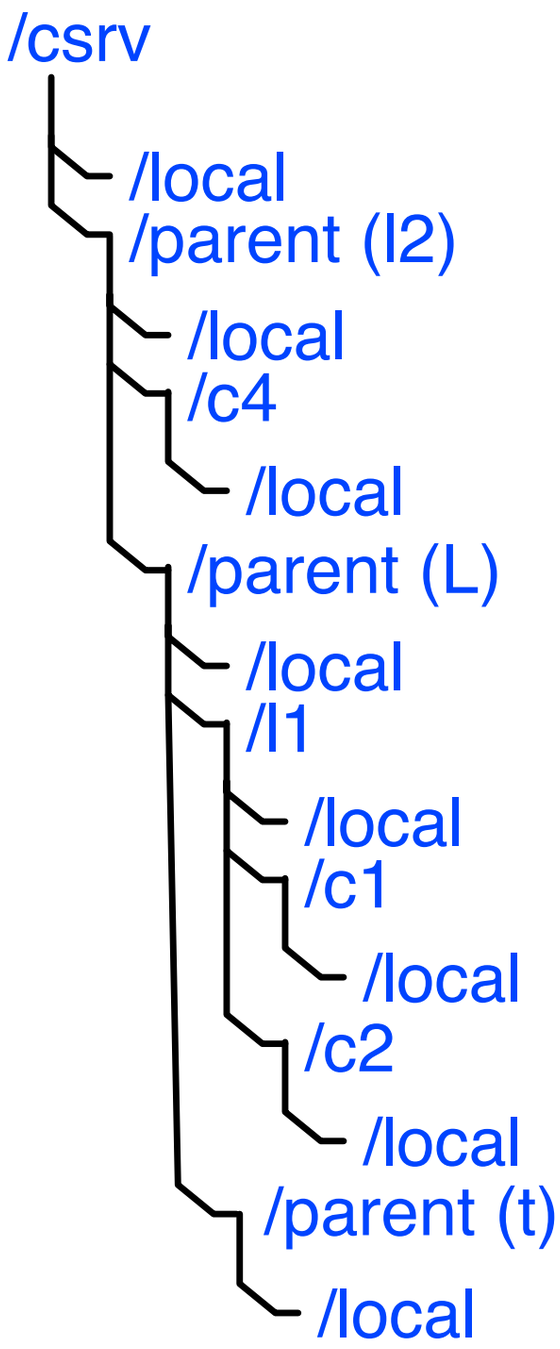
Distribution and Aggregation



App-1 view



App-2 view



Command Line Interface(s)

- Direct File Interaction

```
echo “res 2” > ./mpoint/csrv/local/0/ctl
echo “exec date” > ./mpoint/csrv/local/0/0/ctl
echo “exec wc” > cat ./mpoint/csrv/local/0/1/ctl
echo “xsplice 0 1” > ./mpoint/csrv/0/local/0/ctl
cat ./mpoint/csrv/0/local/0/stdio
```

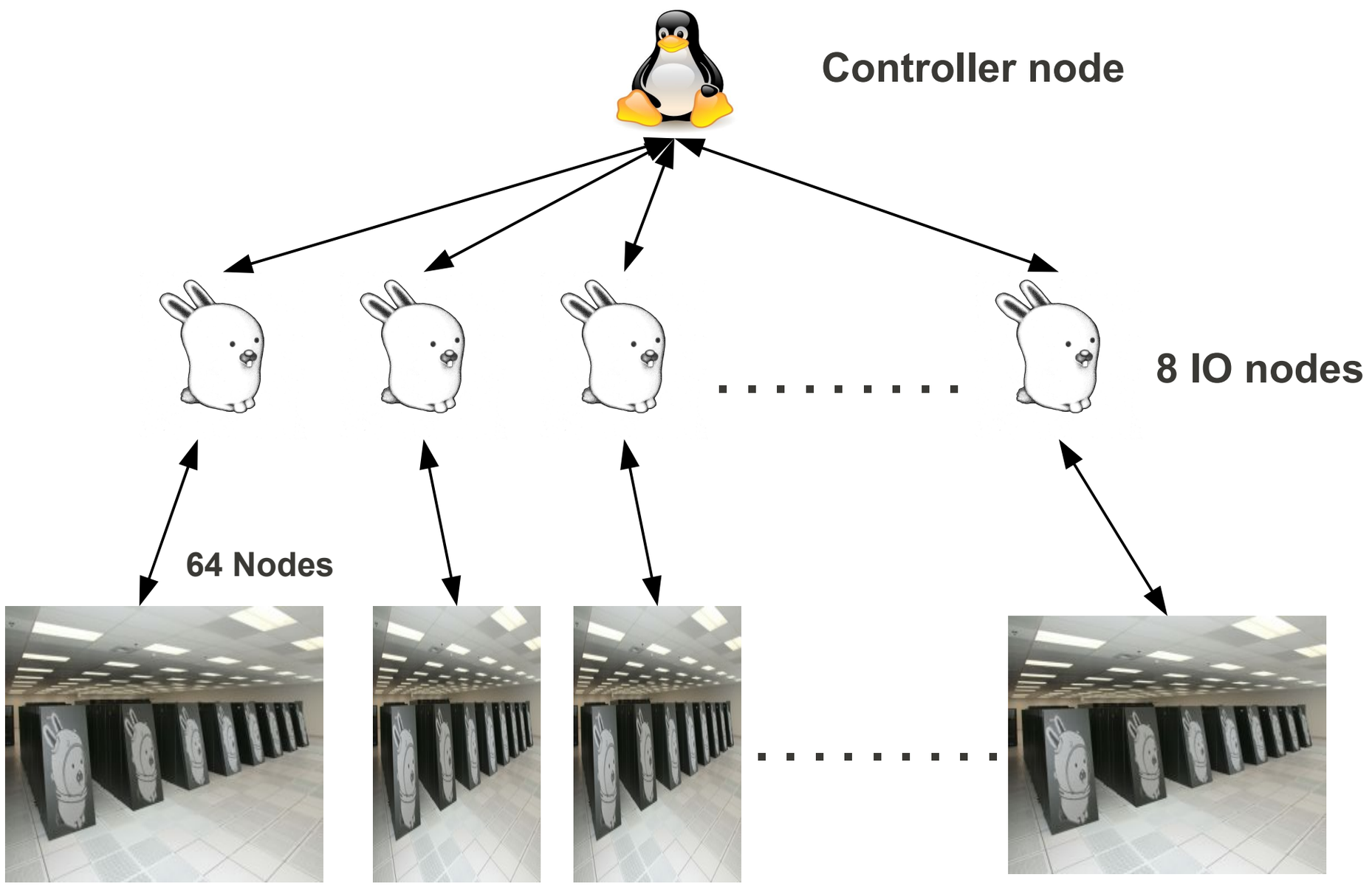
- Python Command Line

```
runJob.py -n 4 /bin/date
```

- PUSH

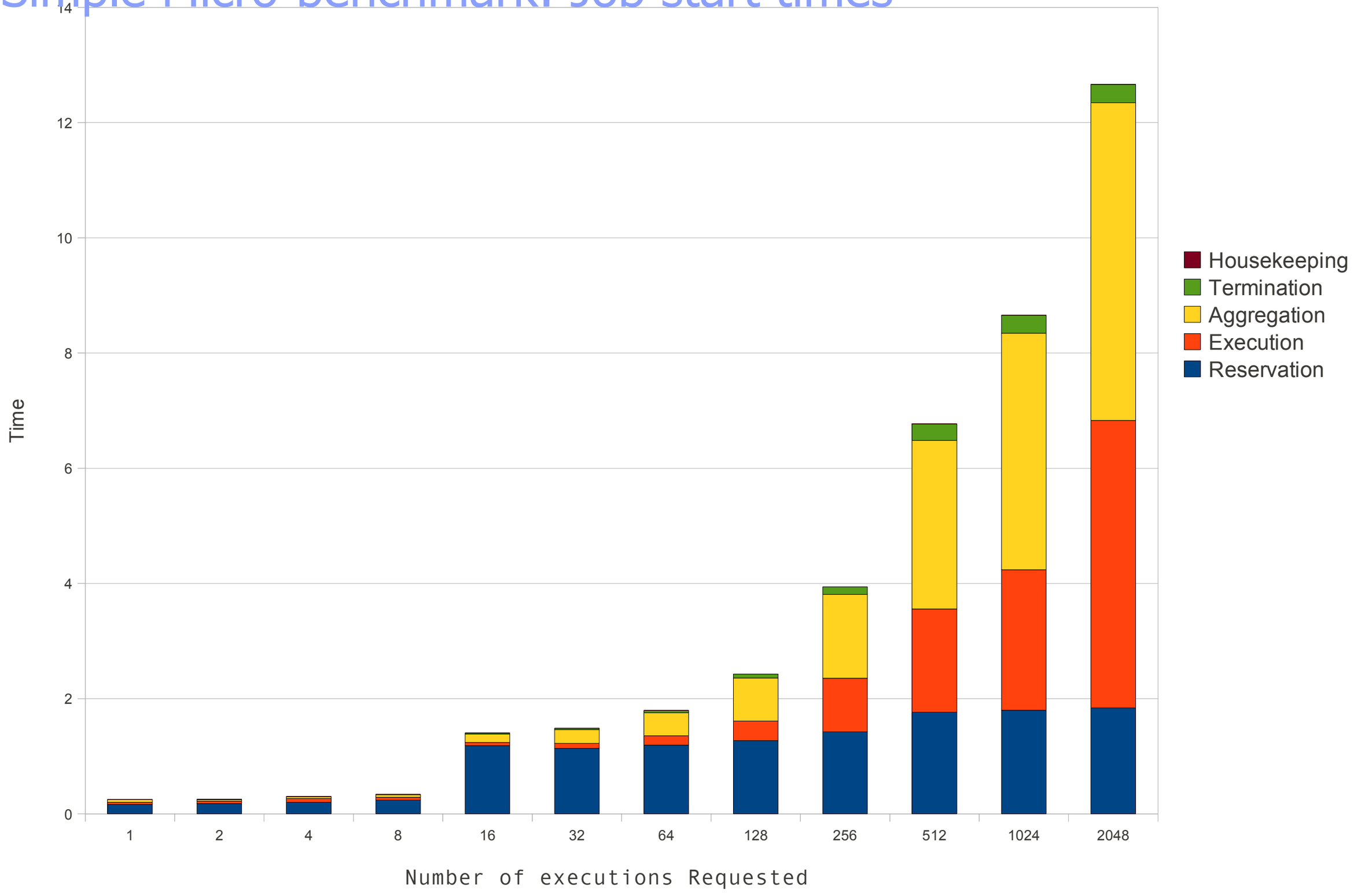
```
ORS=blm find . -type f |< xargs chasen | sort | uniq -c >| sort -rn
```


Evaluation Setup

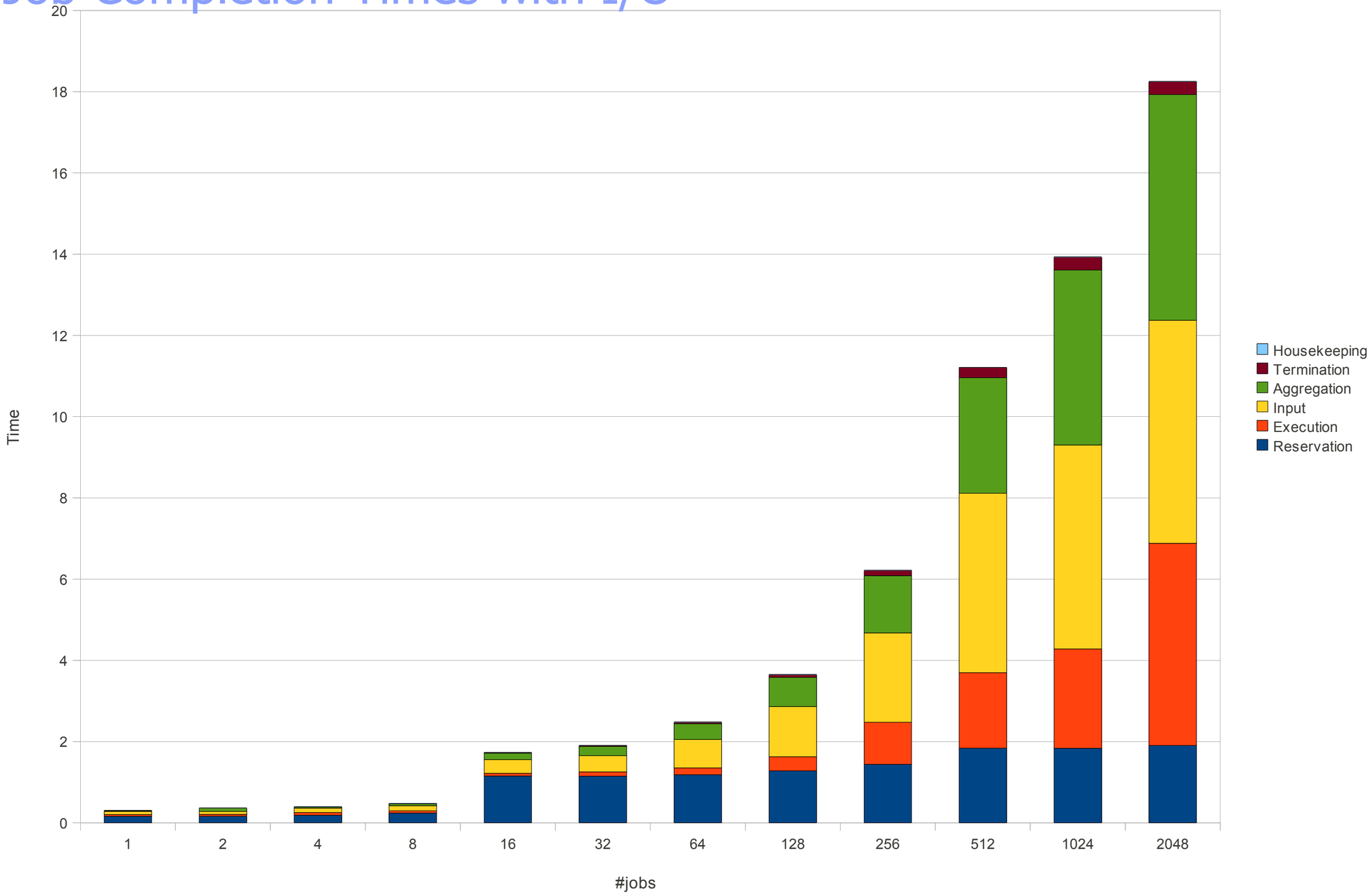


512 total compute nodes

Simple Micro-benchmark: Job start times



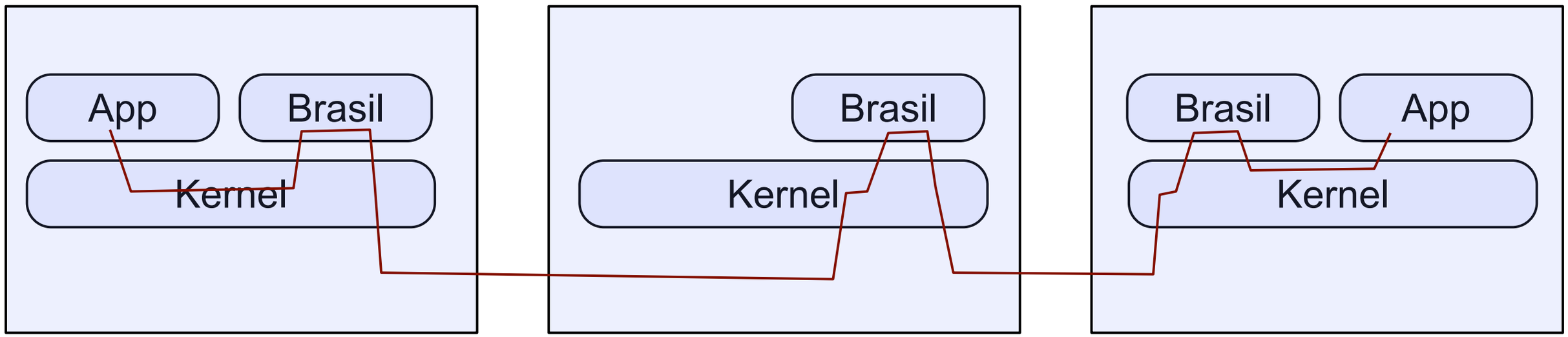
Job Completion Times with I/O



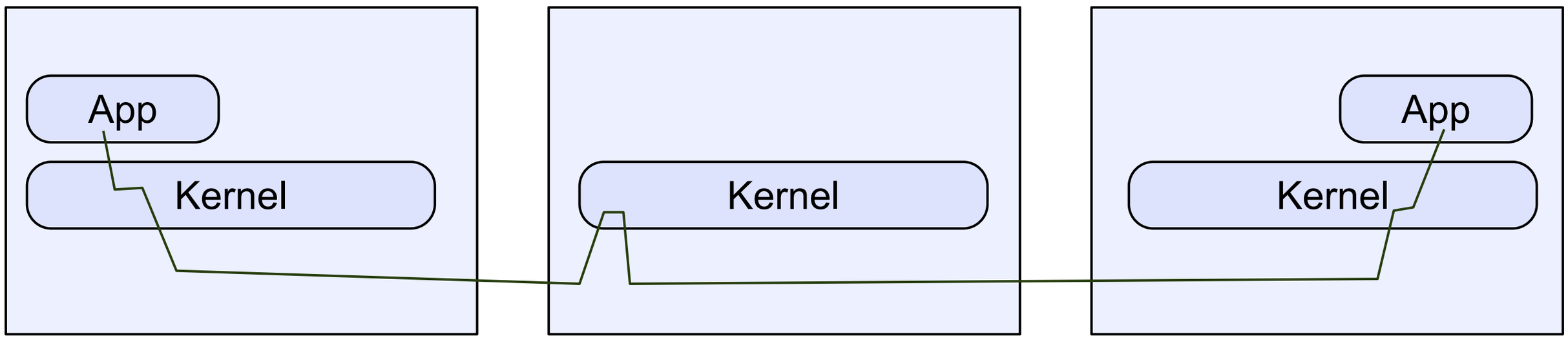
Discussion

- Non-buffered communications channels are great for synchronization, but bad for performance as aggregation points become serialization points.
- Pushing multiplexing to PUSH a mistake
  - adds additional copies of data and additional context switches for I/O to do record separation
  - by pushing them into the infrastructure we can go from 6 copies and context switches down to 2 for each pipeline stage
  - “beltway buffers” and other techniques may reduce this further or eliminate copies altogether
- Transitive mounts of namespace an elegant way to bypass network segmentation -- but it also incurs lots of overhead
- Allowing splice inside a reservation has dubious usefulness. It seems like it might be more useful to allow for individual elements to be spawned outside a reservation.
- Fan-in and Fan-out are too limiting of a use case. What about deterministic delivery? What about many-to-many pipeline components? What about collectives and barriers?
- Fault tolerance & fault debugging properties were poor -- no channel for out-of-band communication of error or logging information. When transitive mounts went down, the system wedged -- but no integrated method of figuring out where the system went down.
- File systems and communication are outside the scope of this work, but still a sore point for performance with the Plan 9 kernel on Blue Gene. Both issues are actively being worked on with a release planned later this summer.

Future Work: Cutting Out The Middle Man



Initiating Terminal or Node | Parent or Gateway Node(s) | Compute Node



Future Work: Splice Optimizations via Direct Communication

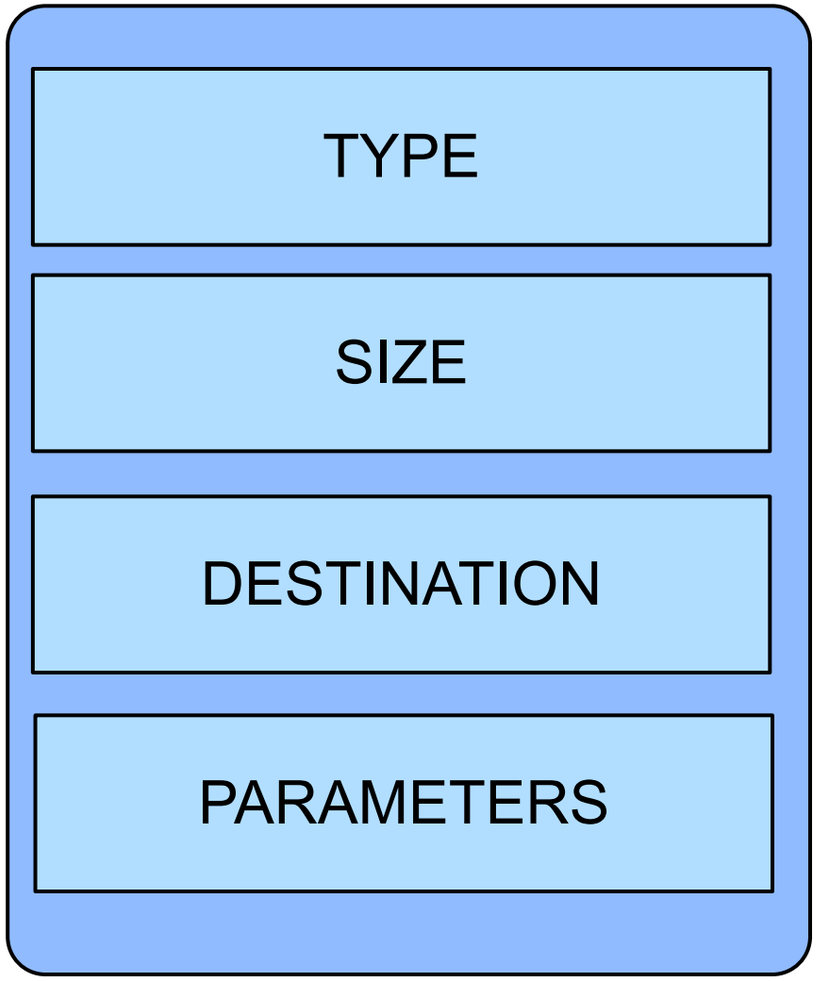
- Splice operations through name space is elegant, but transitive mounting of name space means that data has to flow through hierarchy in order to be spliced from one element to another
- A direct communication path would be much more desirable, particularly on BG/P where hierarchy is constructed on tree, but torus is the preferred node-to-node data transport
- Solution: add a layer of indirection to splice communication
  - add a file to the session directories which contains a list of “locations” for this node which essentially gives a recipe for connecting directly to the channel in order of performance. If the source node cannot use any of these recipes he will default to the name space path.
  - example:

```
% cat ./mpoint/csrv/parent/c4/local/0/location
torus!1.3.4!564
tree!0.23!564
tcp!10.1.3.4!564
```

  - in the simplest embodiment we mount the namespace of the node in order to access its interfaces, but we also want to play with direct connections for data and/or potentially hiding everything inside the csrv interface so the mpipe splice code (and end-users) can be more or less ignorant of how the resources are accessed

Future Work: MULTI-PIPES

- Turn fan-out/fan-in/many-to-many pipeline operations into a system primitives with similar syntax to existing pipe primitives but with semantics of multipipe
  - respect record boundries
  - allow broadcast
  - allow enumerated specification of destination
  - support splice operations
- Build out rest of brasil infrastructure based on this new primitve
  - All stdio channels are multipipes
    - allow record buffers for decoupled performance
  - All ctl channels implemented on top of multipipes
  - etc.
- Same model could potentially be used for implementation of collective operations and barriers within a distributed file system namespace



pwrite(pipefd, buf, sz, ~(0));

Brasil Code Available: http://www.bitbucket.org/ericvh/hare/usr/brasil

New Version In Progress: http://www.bitbucket.org/ericvh/hare/sys/src/cmd/uem

http://goo.gl/5eFB

This project is supported in part by the
U.S. Department of Energy under
Award Number DE-FG02- 08ER25851

AUSTIN RESEARCH LAB

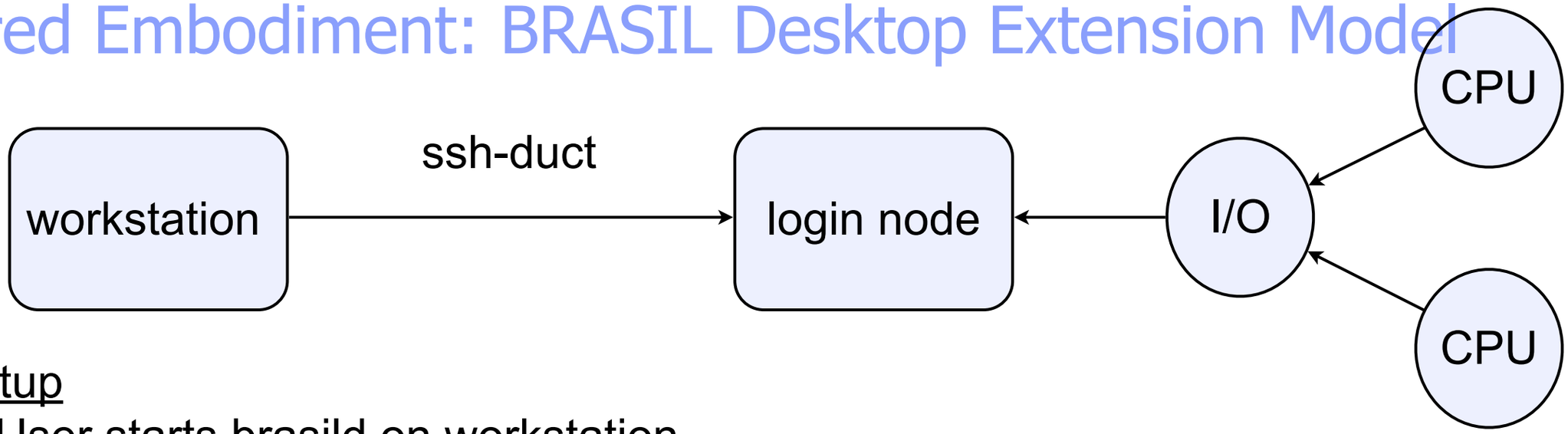
http://www.research.ibm.com/austin

Alcatel·Lucent

Core Concept: BRASIL
Basic Resource Aggregate System Inferno Layer

- Stripped down Inferno - No GUI or anything we can live without, minimal footprint
- Runs as a daemon (no console), all interaction via 9p mounts of its namespace
- Different modes
  - default (exports /srv/brasil or on a tcp!127.0.0.1!5670)
  - gateway (exports over standard I/O - to be used by ssh initialization)
  - terminal (initiates ssh connection and starts a gateway)
- Runs EVERYWHERE
  - User's workstation
  - Surveyor Login Nodes
  - I/O Nodes
  - Compute Nodes

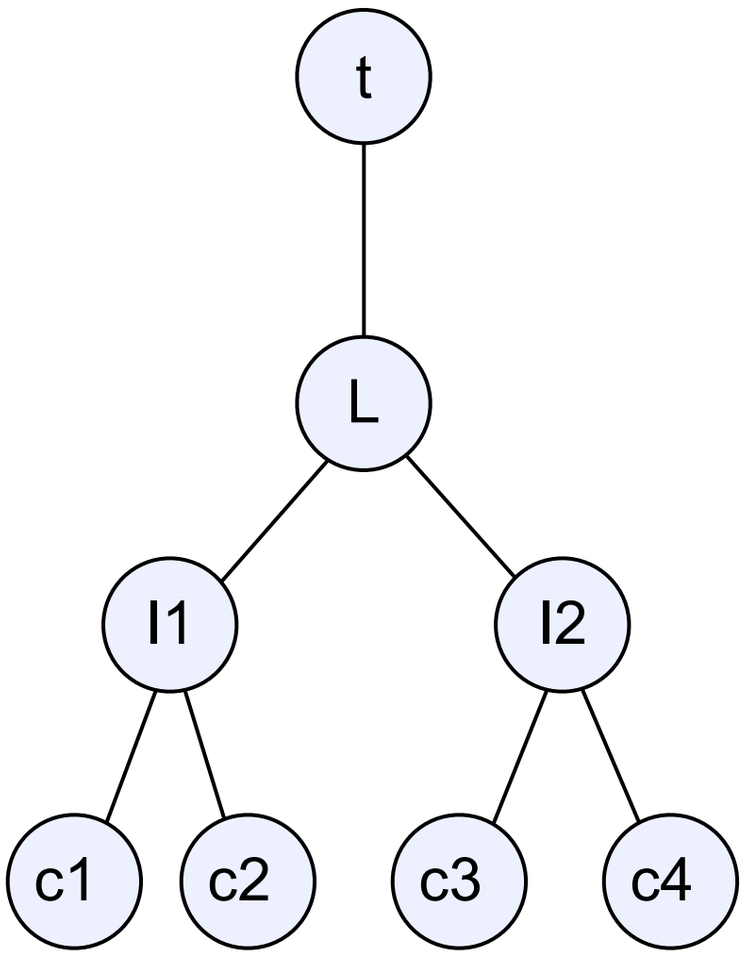
Preferred Embodiment: BRASIL Desktop Extension Model



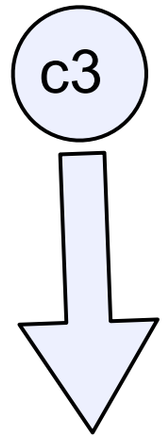
- Setup
  - User starts brasild on workstation
    - brasild ssh's to login node and starts another brasil hooking the two together with 27b-6 and mount resources in /csrv
  - User mounts brasild on workstation into namespace using 9pfuse or v9fs (or can mount from Plan 9 peer node, 9vx, p9p or ACME-sac)
- Boot
  - User runs anl/run script on workstation
    - script interacts with taskfs on login node to start cobalt qsub
    - when I/O nodes boot it will connect its csrv to login csrv
    - when CPU nodes boot they will connect to csrv on I/O node
- Task Execution
  - User runs anl/exec script on workstation to run app
    - script reserves x nodes for app using taskfs
    - taskfs on workstation aggregates execution by using taskfs running on I/O nodes

Core Concept: Central Services

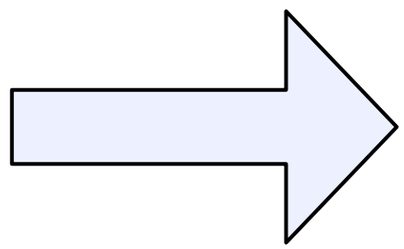
- Establish hierarchical namespace of cluster services
- Automount remote servers based on reference (ie. cd /csrv/criswell)
- Export local services for use elsewhere within the network



/csrv
  /local
  /L
    /local
  /l1
    /local
    /c1
      /local
    /c2
      /local
/l2
    /local
    /c3
      /local
    /c4
      /local

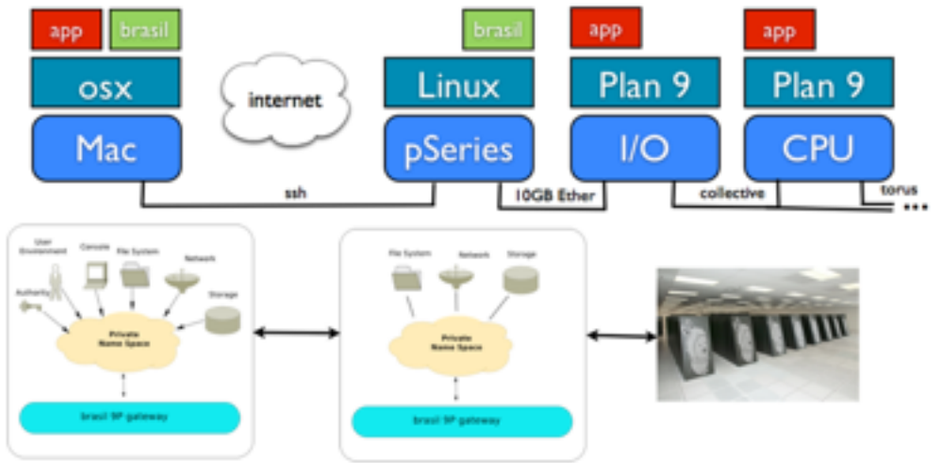


/csrv
  /local
  /l2
    /local
    /c4
      /local
  /L
    /local
    /t
      /local
  /l1
    /local
    /c1
      /local
    /c2
      /local



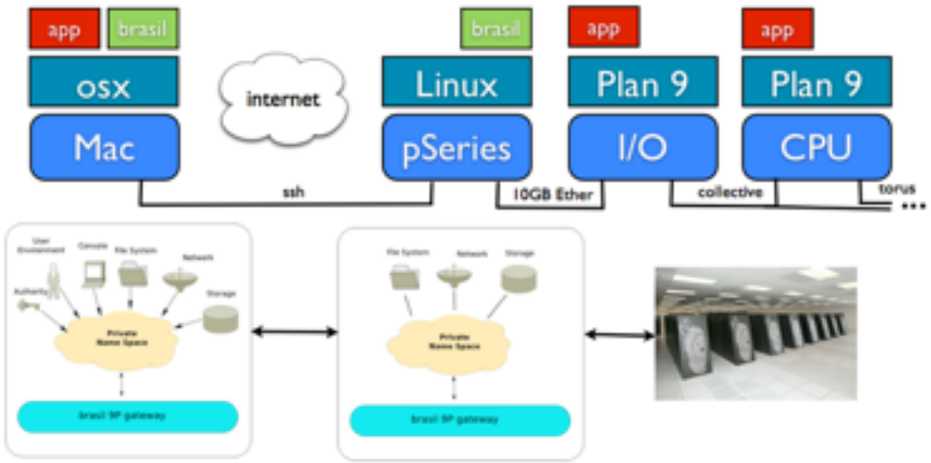
Our Approach: Workload Optimized Distribution

Our Approach: Workload Optimized Distribution

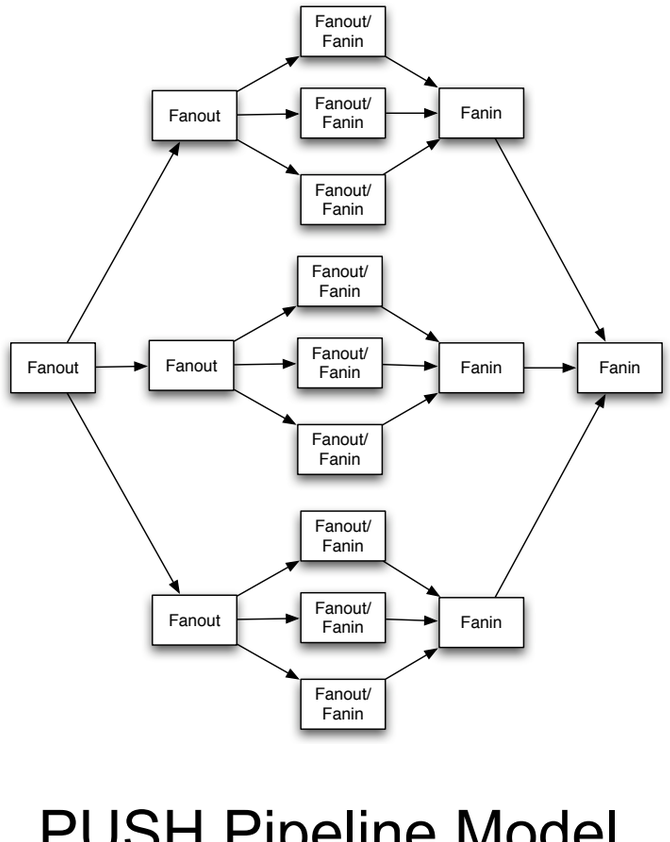


Desktop Extension

Our Approach: Workload Optimized Distribution

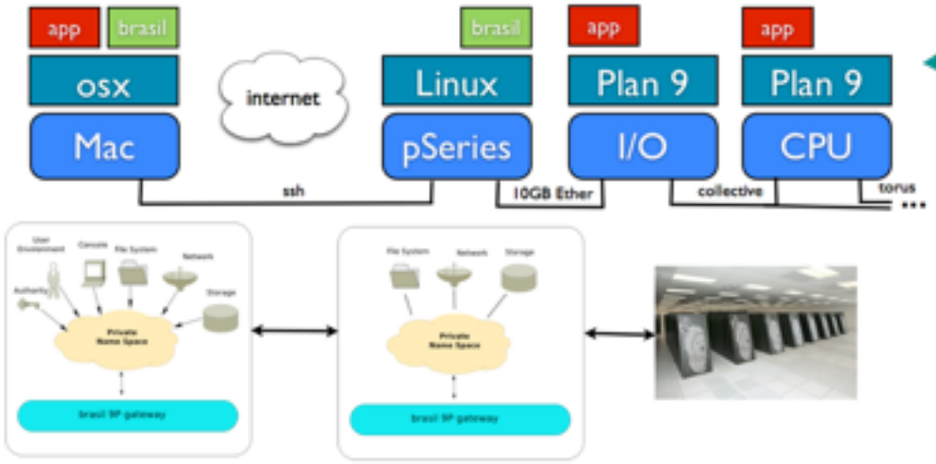


Desktop Extension

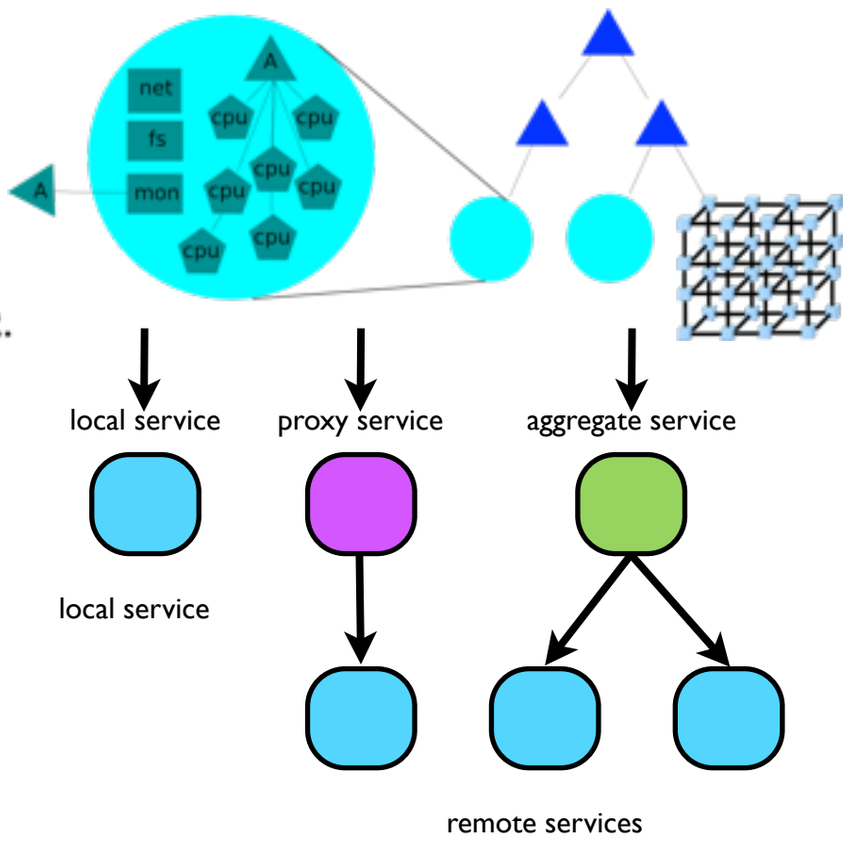


PUSH Pipeline Model

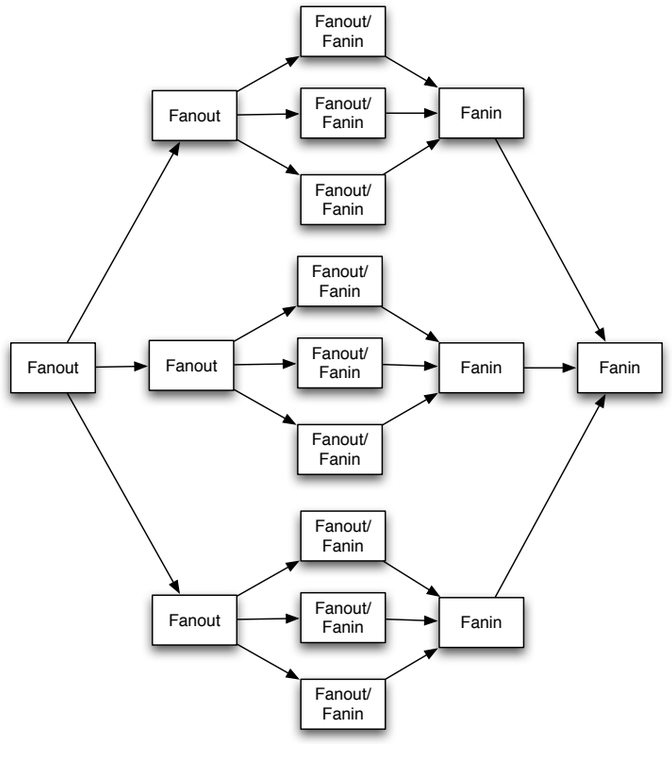
Our Approach: Workload Optimized Distribution



Desktop Extension

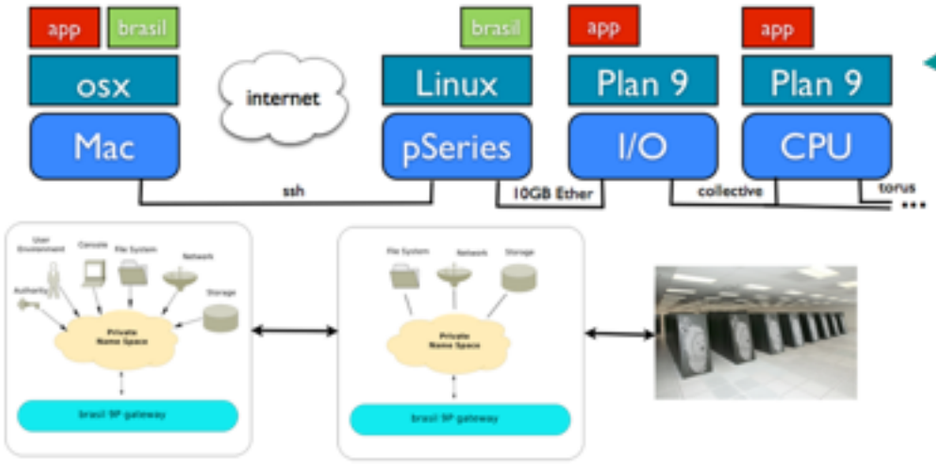


Aggregation Via
Dynamic Namespace
and
Distributed Service
Model

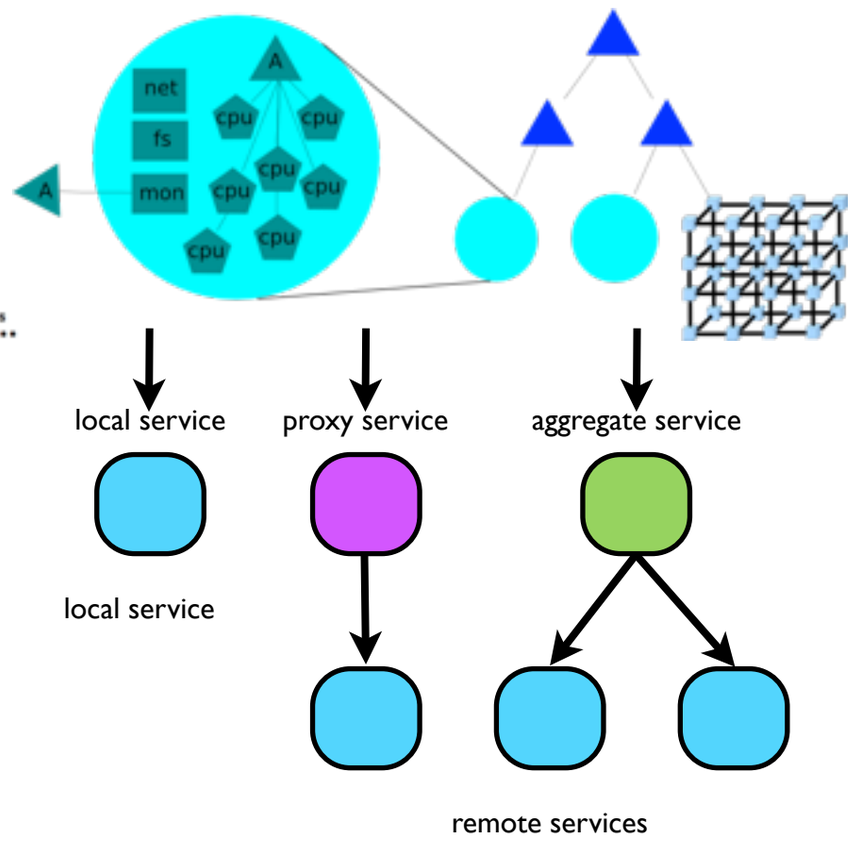


PUSH Pipeline Model

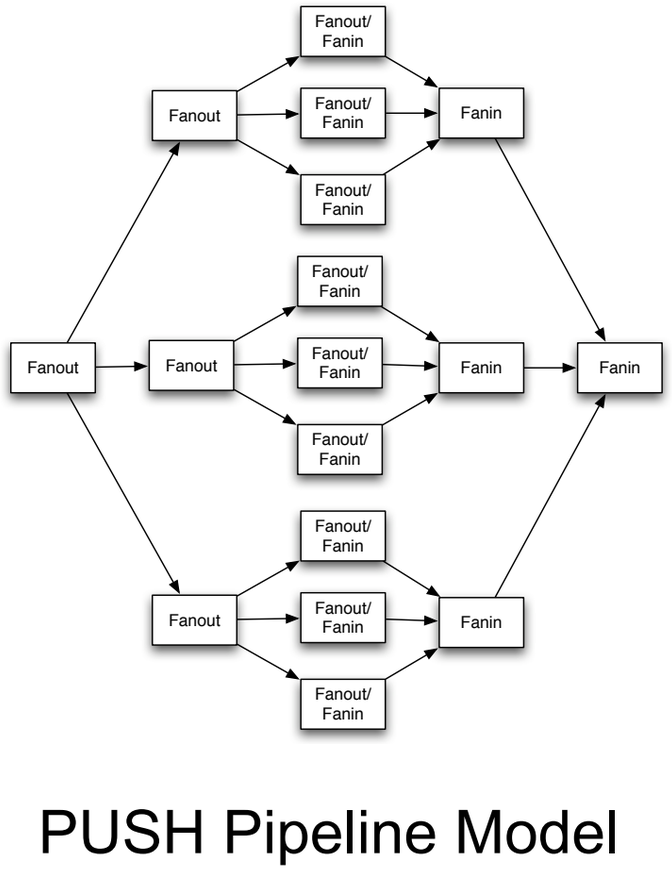
Our Approach: Workload Optimized Distribution



Desktop Extension



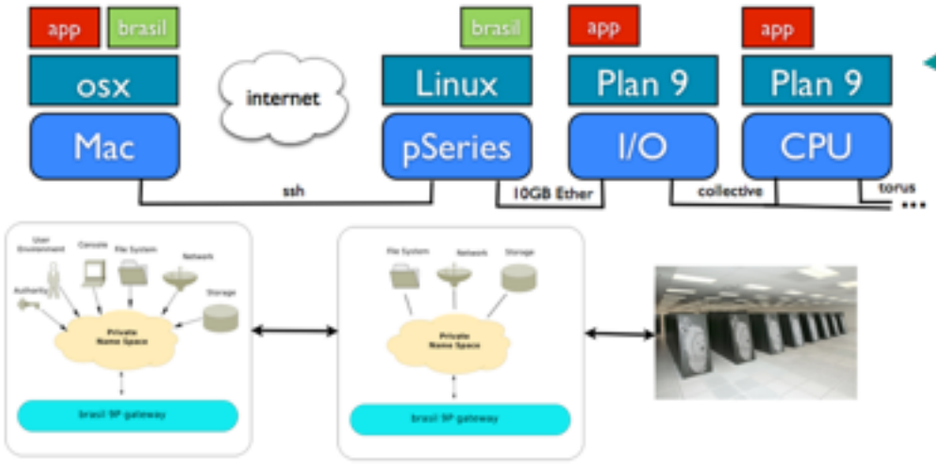
Aggregation Via
Dynamic Namespace
and
Distributed Service
Model



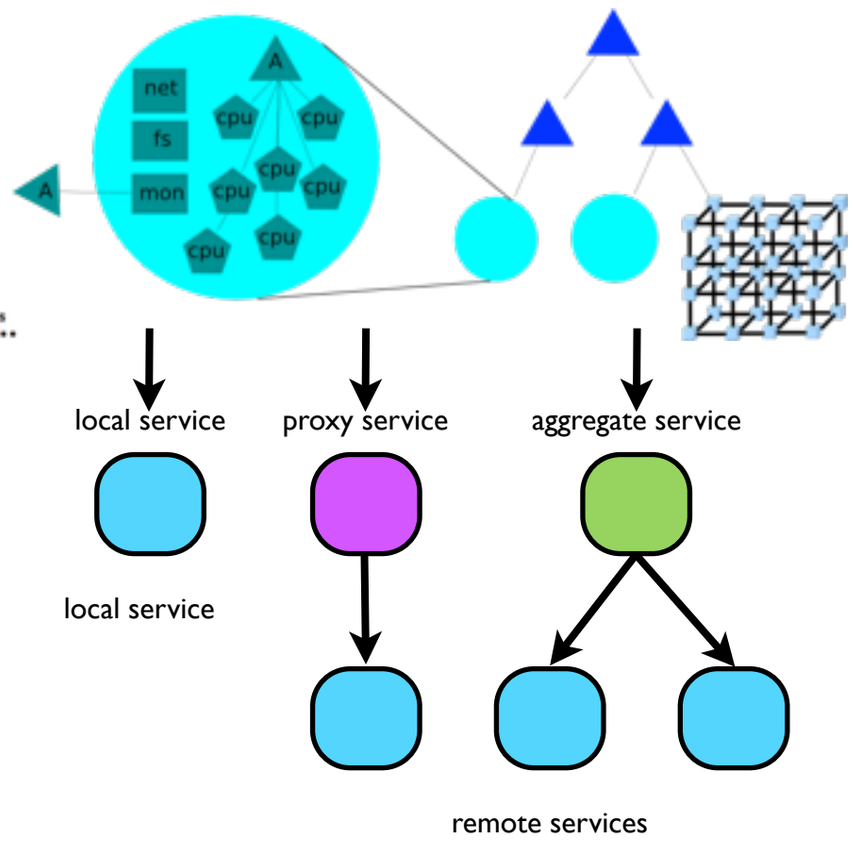
PUSH Pipeline Model

Scaling

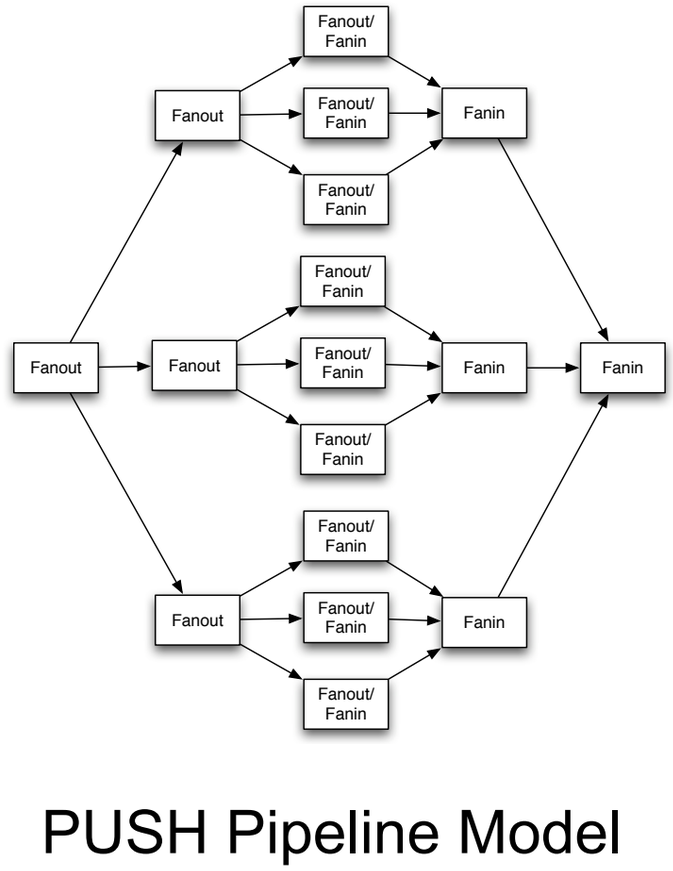
Our Approach: Workload Optimized Distribution



Desktop Extension



Aggregation Via Dynamic Namespace and Distributed Service Model

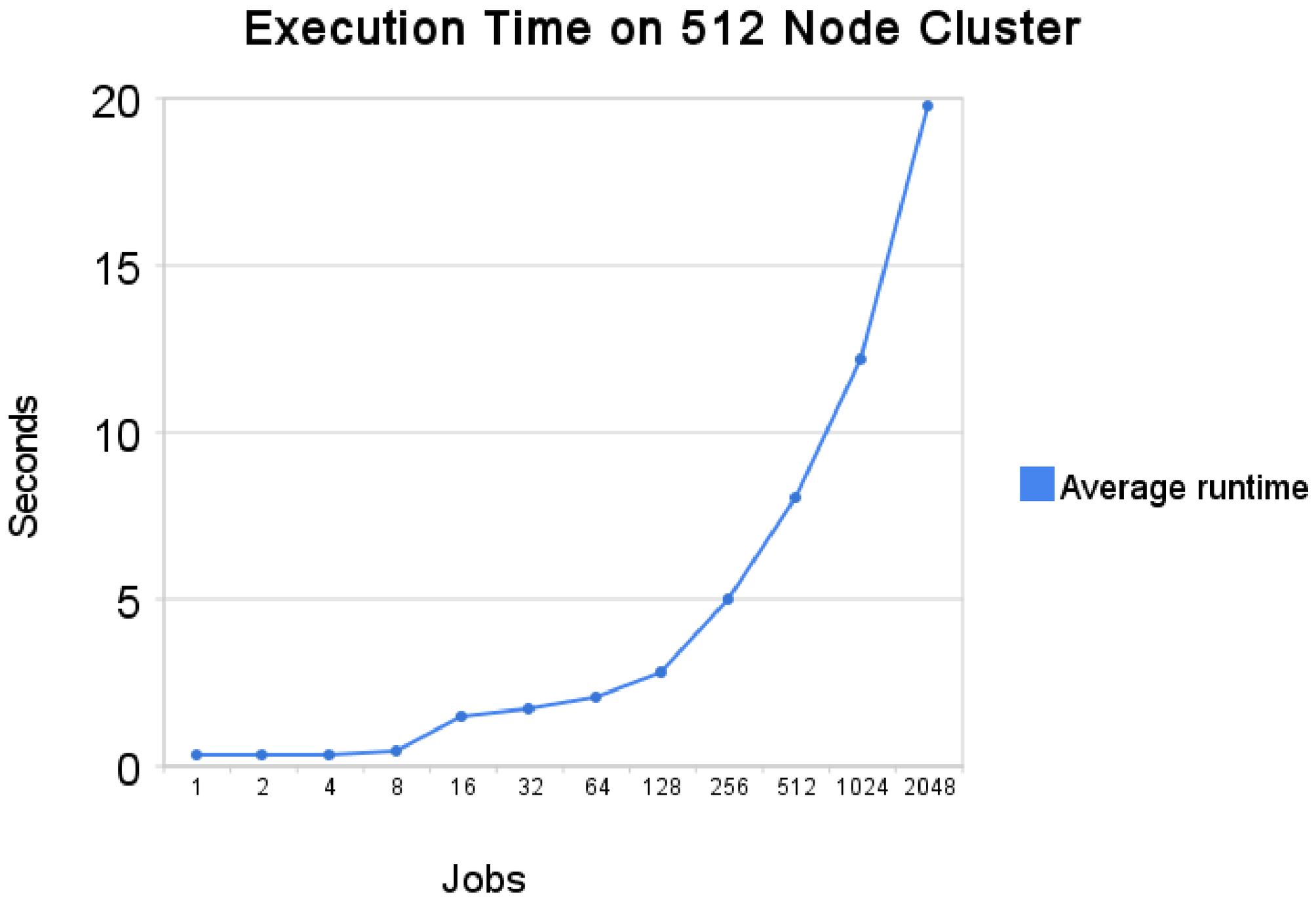


PUSH Pipeline Model

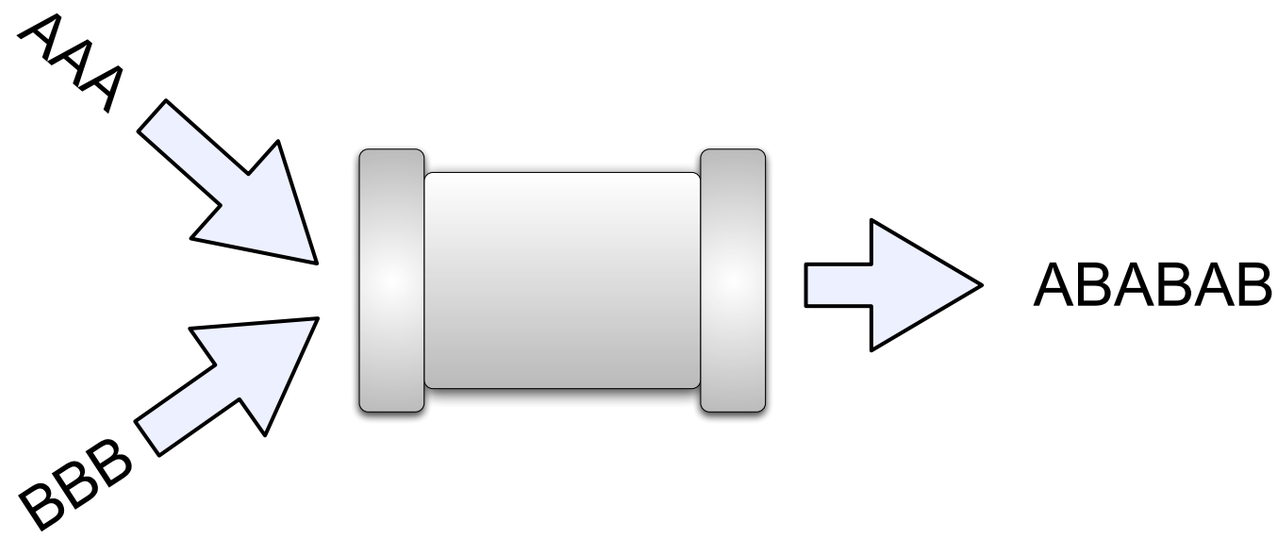
Scaling

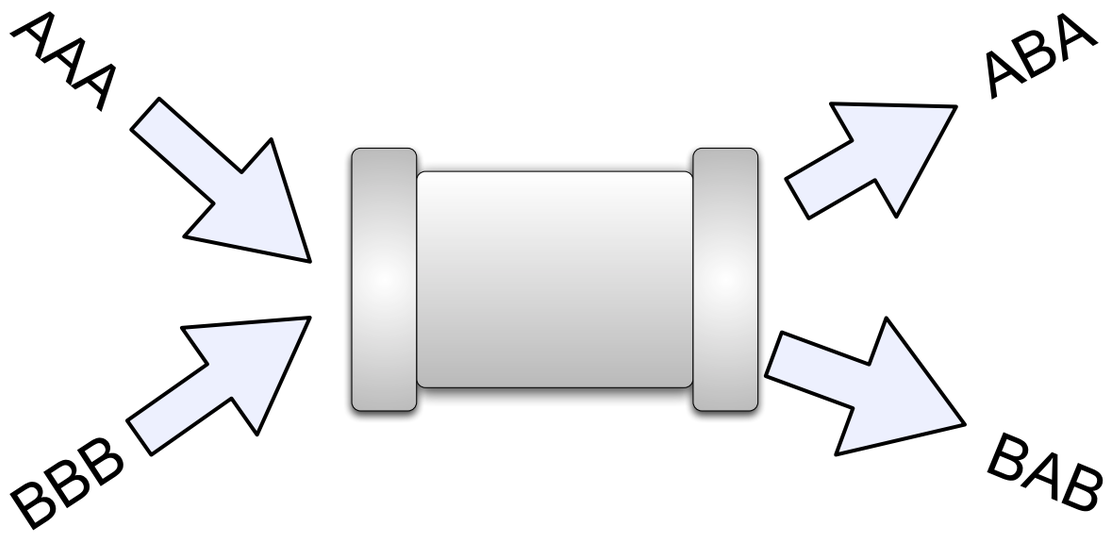
Reliability

"Old" Performance Graph (from USENIX)

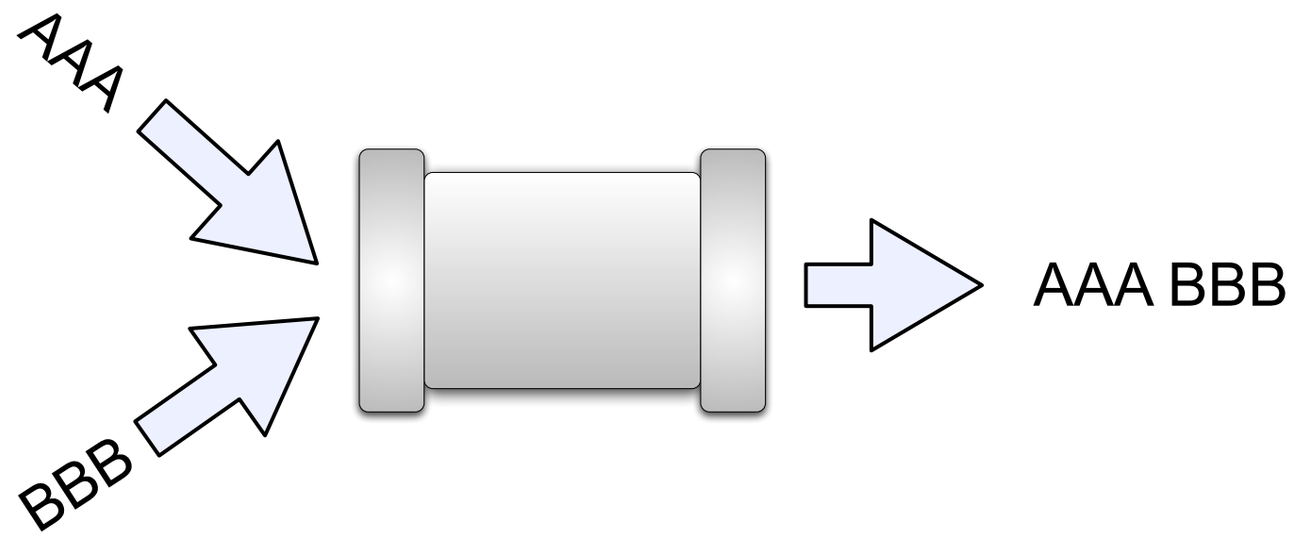


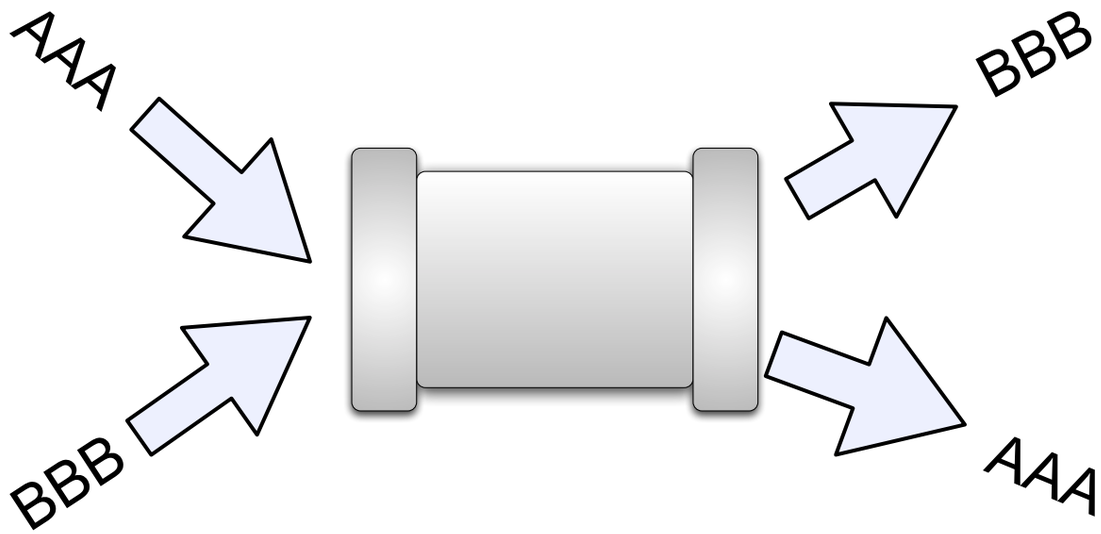
Problem: Limitations of Traditional Pipes



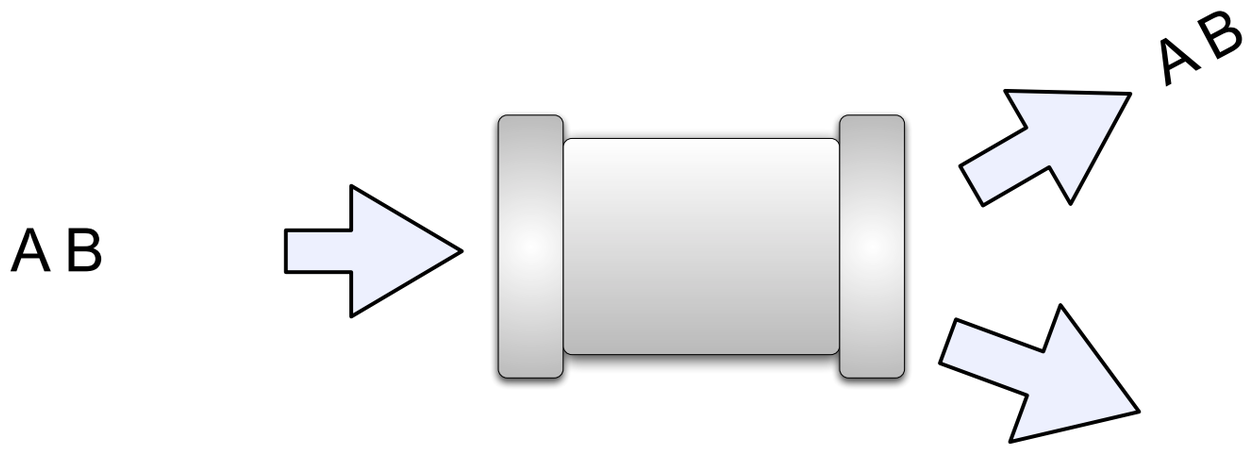


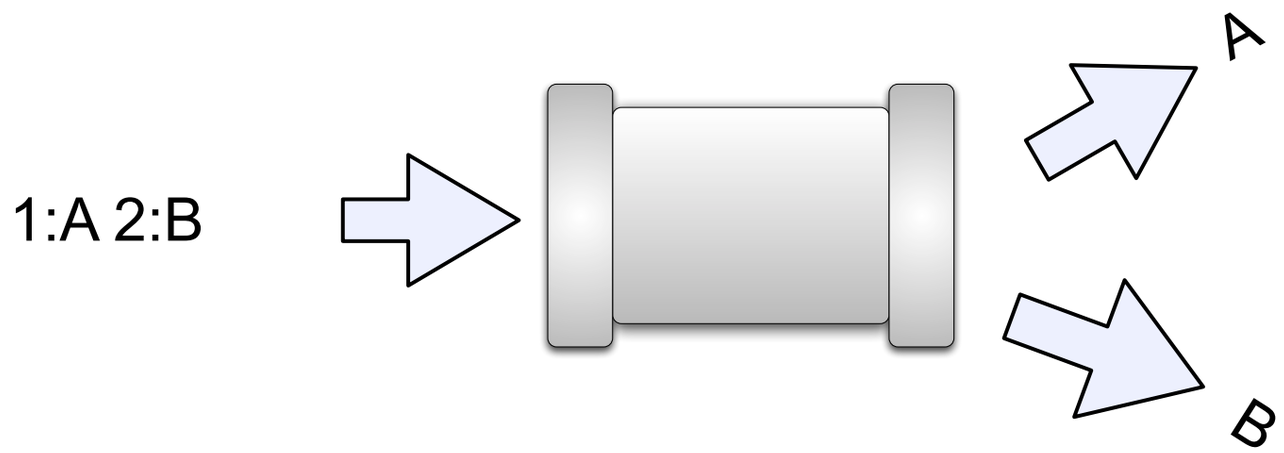
Long Packet Pipes



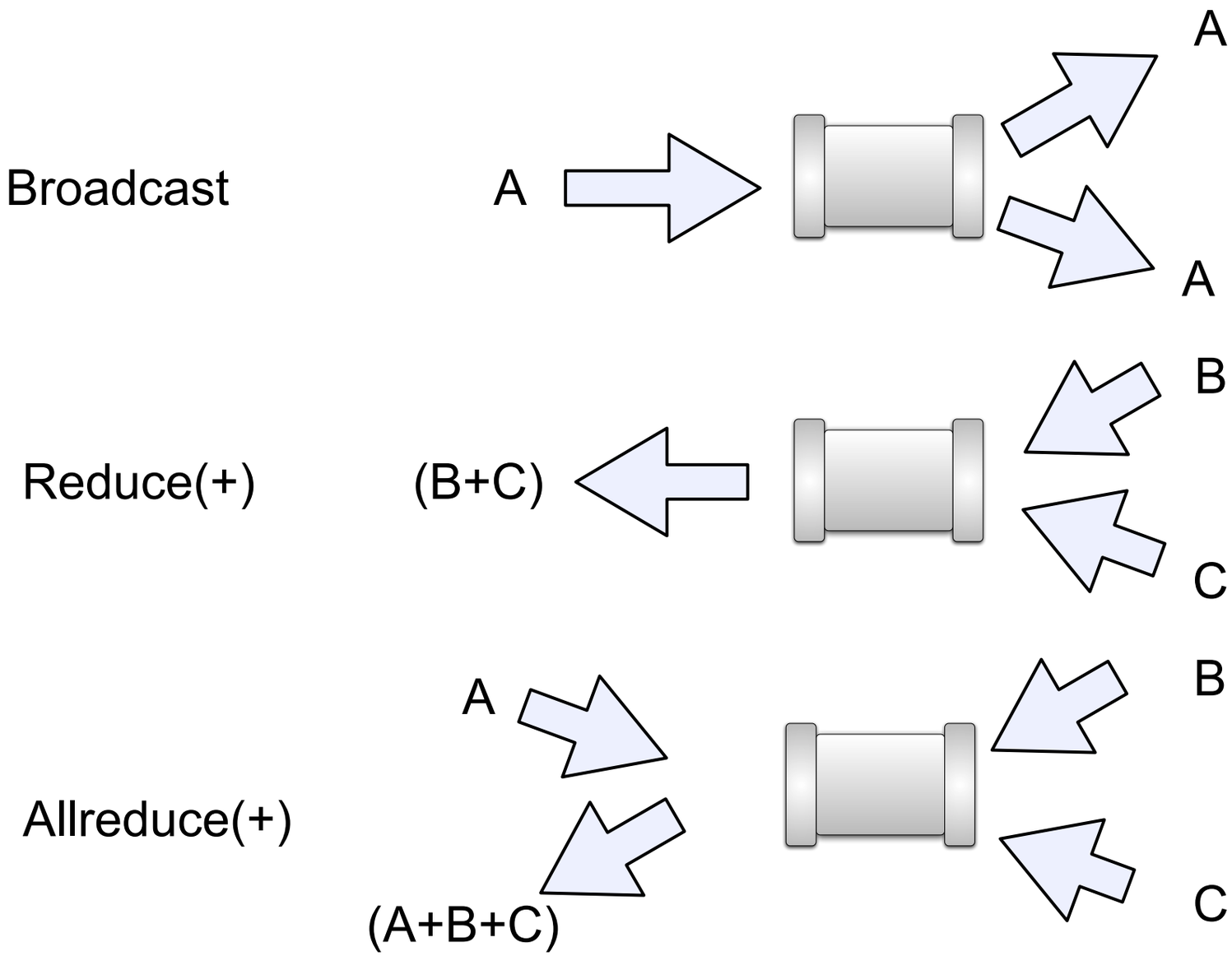


Enumerated Pipes

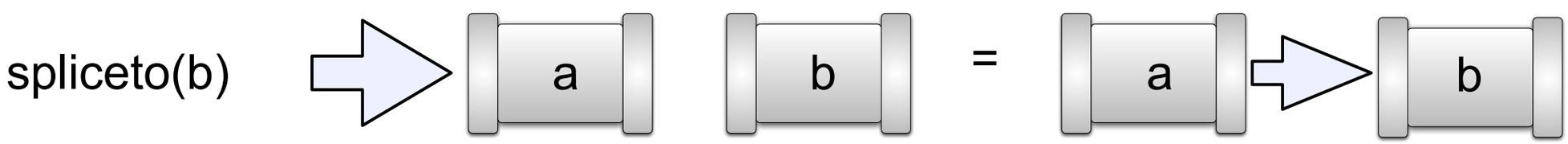


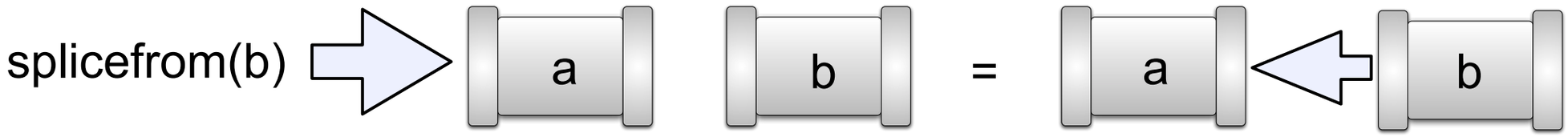


Collective Pipes



Splicing Pipes





Example Simple Invocation

- mpipefs
- mount /srv/mpipe /n/testpipe
- ls -l /n/testpipe

```
--rw-rw-rw- M 24 ericvh ericvh 0 Oct 10 18:10 /n/testpipe/data
```

- echo hello > /n/testpipe/data &
- cat /n/testpipe/data

```
hello
```

Passing Arguments via aname

- mount /srv/mpipe /n/test othername
- ls -l /n/test

  --rw-rw-rw- M 26 ericvh ericvh 0 Oct 10 18:12 /n/test/otherpipe

- mount /srv/mpipe /n/test2 -b bcastpipe
- mount /srv/mpipe /n/test3 -e 5 enumpipe
- ....you get the idea, read the man page for more details

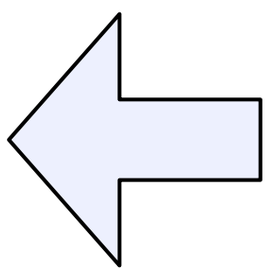
Example for writing control blocks

```
int
pipewrite(int fd, char *data, ulong size, ulong which)
{
        int n;
        char hdr[255];
        ulong tag = ~0;
        char pkttype='p';

        /* header byte is at offset ~0 */
        n = snprint(hdr, 31, "%c\n%lud\n%lud\n\n", pkttype, size, which);
        n = pwrite(fd, hdr, n+1, tag);
        if(n <= 0)
                return n;

        return write(fd, data, size);
}
```

Larger Example (execfs)

/proc

- /clone
- /###
  - /stdin
  - /stdout
  - /stderr
  - /args
  - /ctl
  - /fd
  - /fpregs
  - /kregs
  - /mem
  - /note
  - /noteid
  - /notepg
  - /ns
  - /proc
  - /profile
  - /regs
  - /segment
  - /status
  - /text
  - /wait

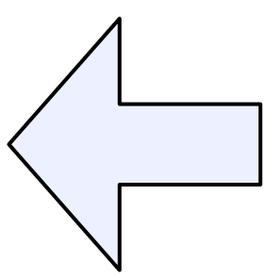


```
mount -a /srv/mpipe /proc/### stdin
mount -a /srv/mpipe /proc/### stdout
mount -a /srv/mpipe /proc/### stderr
```

Really Large Example (gangfs)

/proc

- /gclone
- /status
- /g###
  - /stdin
  - /stdout
  - /stderr
  - /ctl
  - /ns
  - /status
  - /wait



mount -a /srv/mpipe /proc/### -b stdin
mount -a /srv/mpipe /proc/### stdout
mount -a /srv/mpipe /proc/### stderr

...and then, post exec from gangfs clone - execfs stdins are splicedfrom g#/stdin
...and then execfs stdouts and stderrs are splicedto g#/stdout and g#/stderr

...and you can do -e # with stdin to get enumerated instead of brodcast pipes