

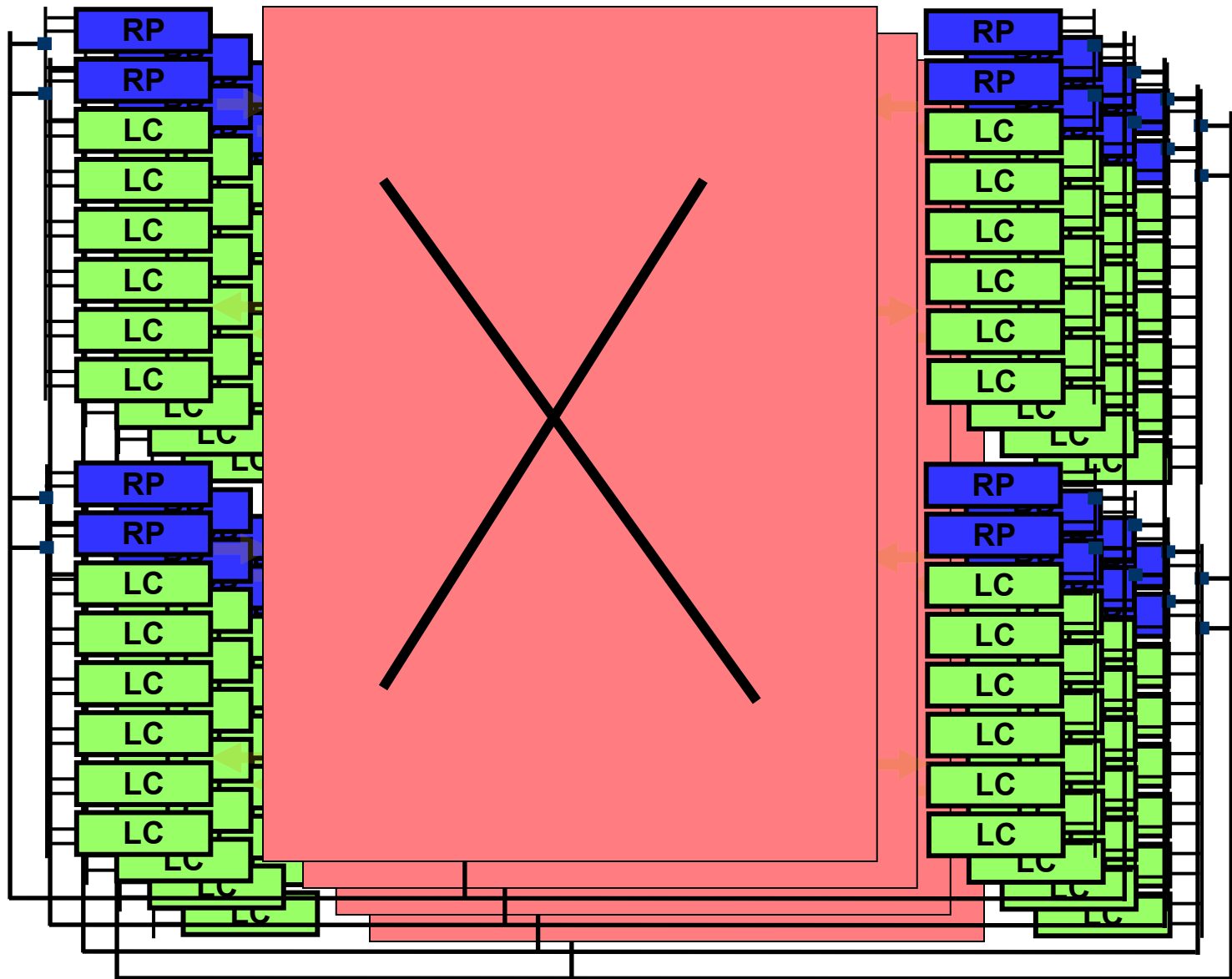


# Scalable I/O Architectures

Herbert Bos

[www.cs.vu.nl/~herbertb](http://www.cs.vu.nl/~herbertb)

Acknowledgments: Willem de Bruijn



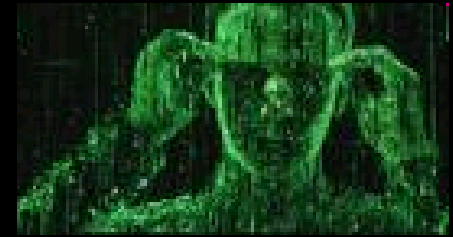
heterogeneous, multiway communication

monitoring

huge performance

# This talk

- can we process at really high rates?



# This talk

- can we process at really high rates?

Many Tbps!

Neo can't see the bits anymore!





# Processing at MATRIX speeds

Herbert Bos

[www.cs.vu.nl/~herbertb](http://www.cs.vu.nl/~herbertb)

Acknowledgments: Willem de Bruijn

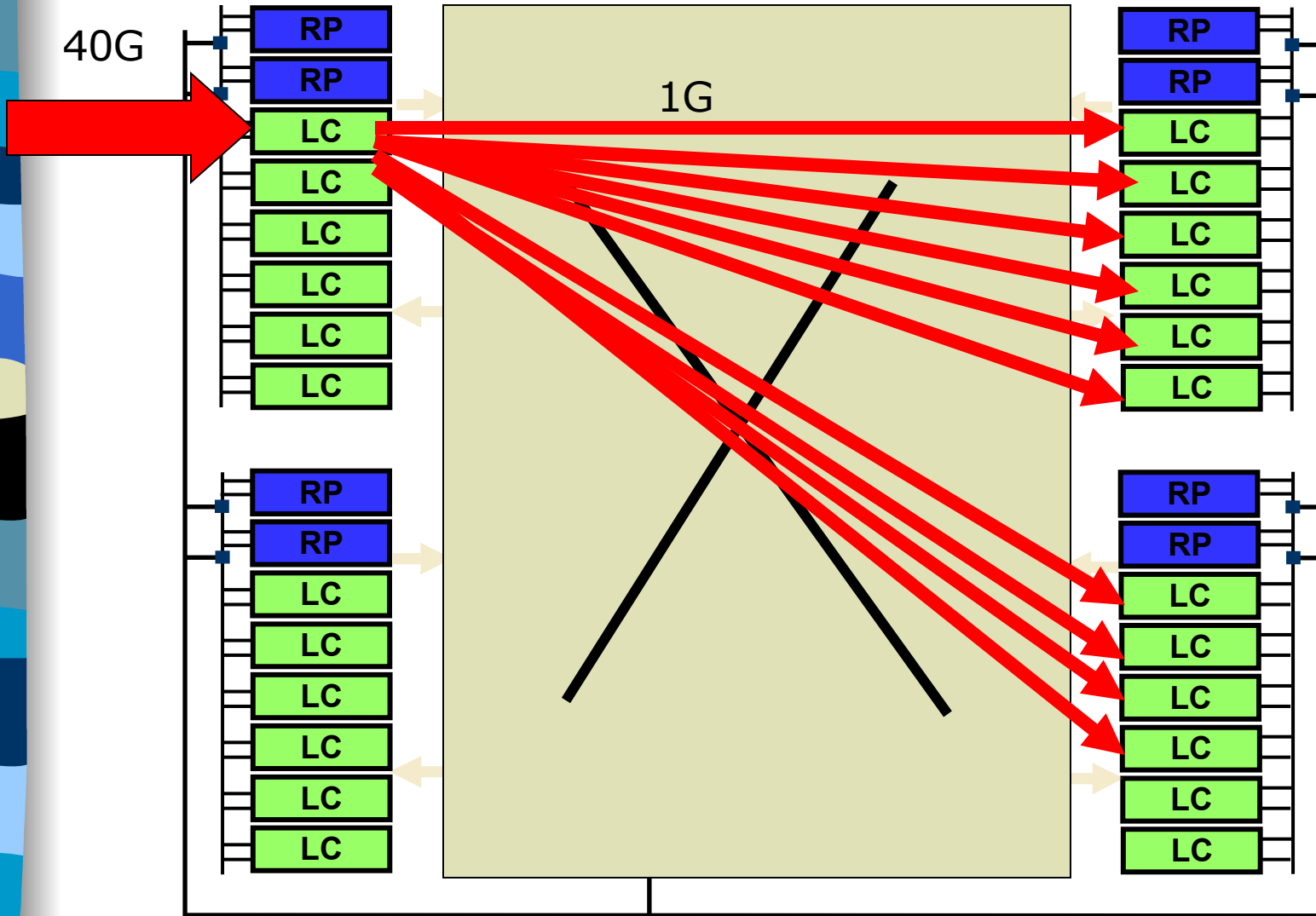


hey guys, maybe  
I could zoom in  
on a single port?



still 40 – 100 Gbps!

# so we distribute the load





# Still

- the number of CPUs is limited
- link rates grow fast
- ➔ we need a scalable I/O architecture

# Goals

- Processing at very high speeds
- I/O architecture
  - supports splitting
  - avoid copies
  - avoid switches
  - nice to TLB + cache
  - avoid allocation
  - avoid VM mapping
  - integrate new hardware smoothly





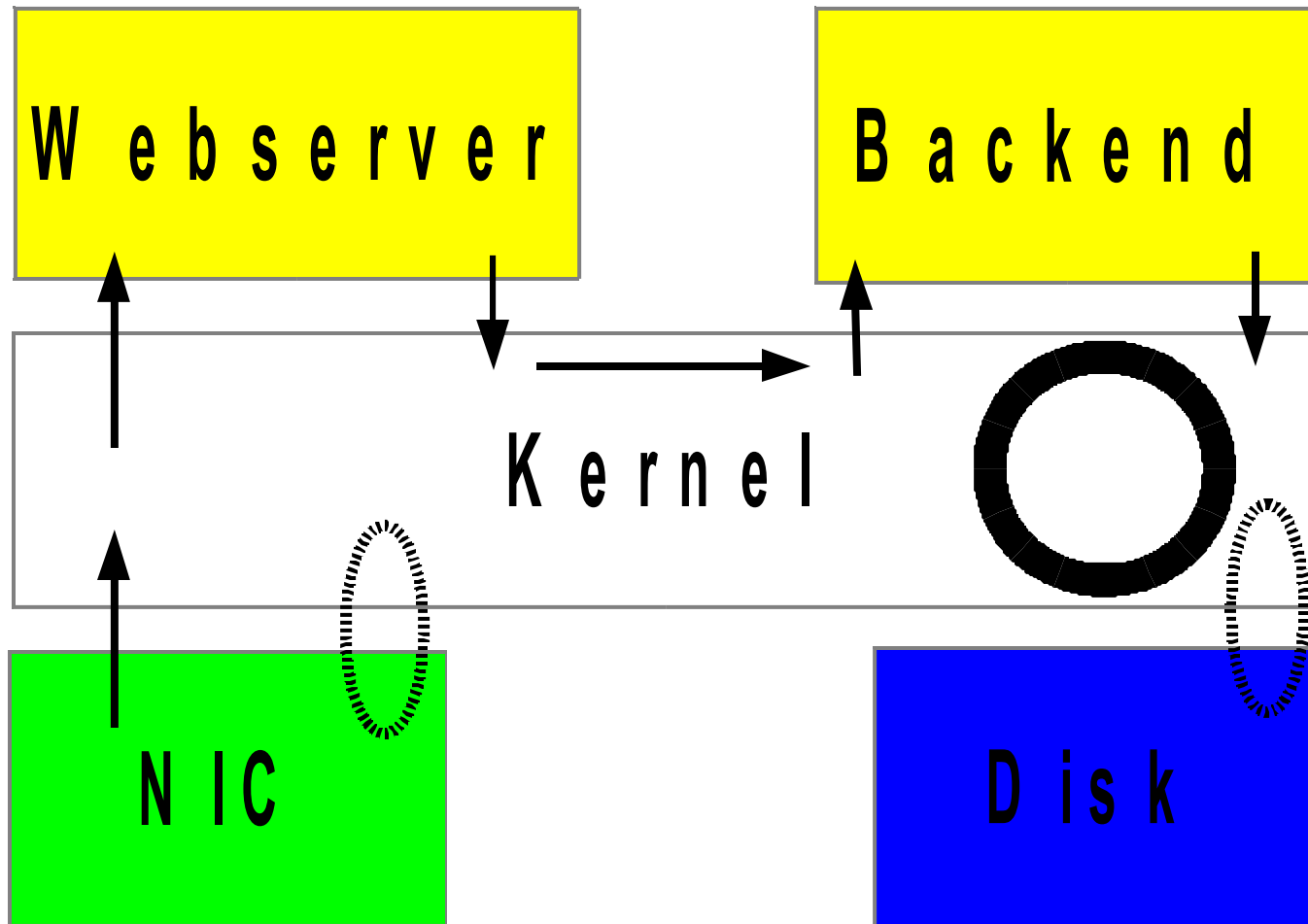
# Memory is major bottleneck

## ■ Current OSs

- copy between kernel subsystems
- copy across U-K boundary
- context switches
- handle packets one at a time
  - allocation
  - mapping
- hard to integrate heterogeneous hardware

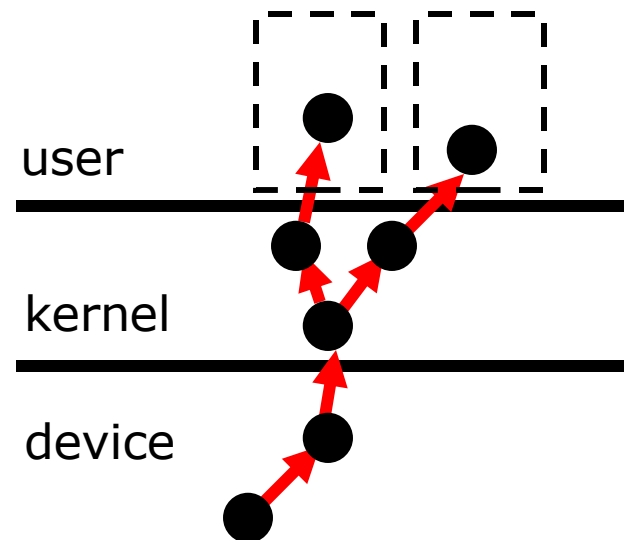
## ■ Problems are structural – no easy fixes

# Consider a web server



# Streamline I/O Architecture

- dynamic paths
- shared ring-buffers
- relocates operations to most suitable hardware and software environment
- streams and filters





# Planes and interfaces

- Streamline explicitly distinguishes between
  - transport plane (beltway buffers)
  - processing plane
  - control plane
- Interfaces:
  - POSIX file API throughout system
  - SRL
  - pipes, sockets, and pcap
  - pipesFS



# Transport Plane



# Beltway Buffers

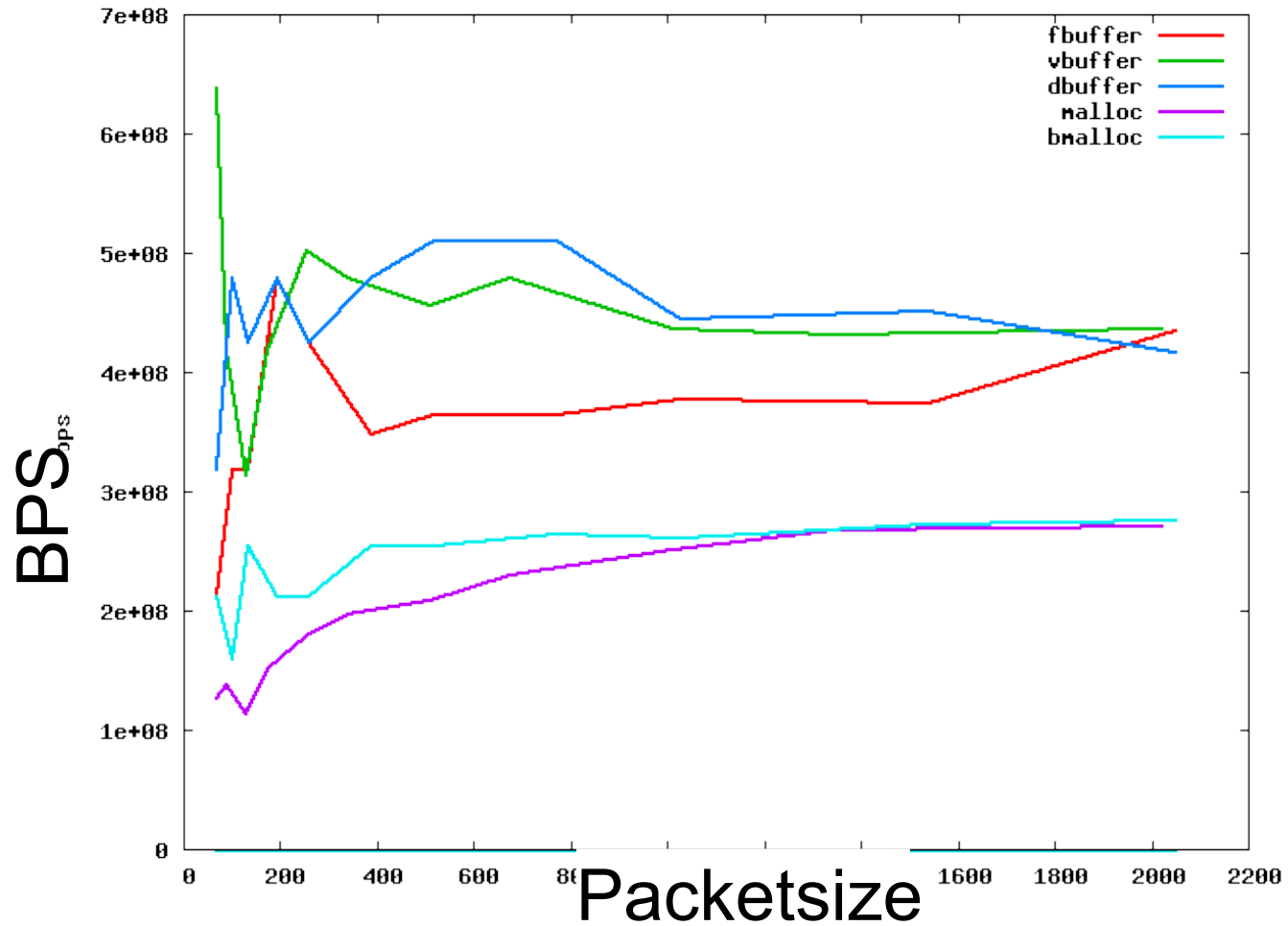
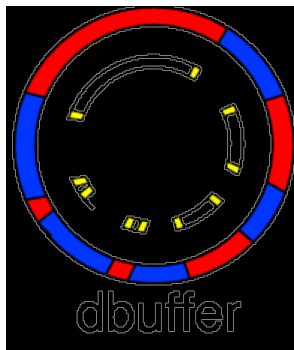
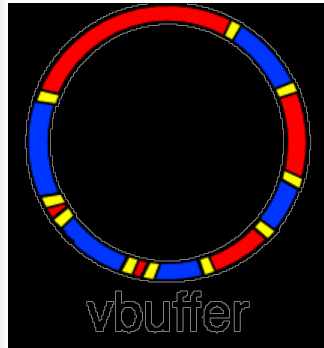
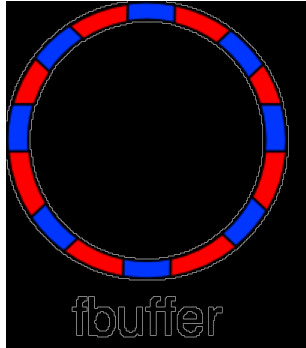
- static ring buffers
- signal coalescing
- uniform interface
- share buffers
  - \*nix unnecessarily restrictive



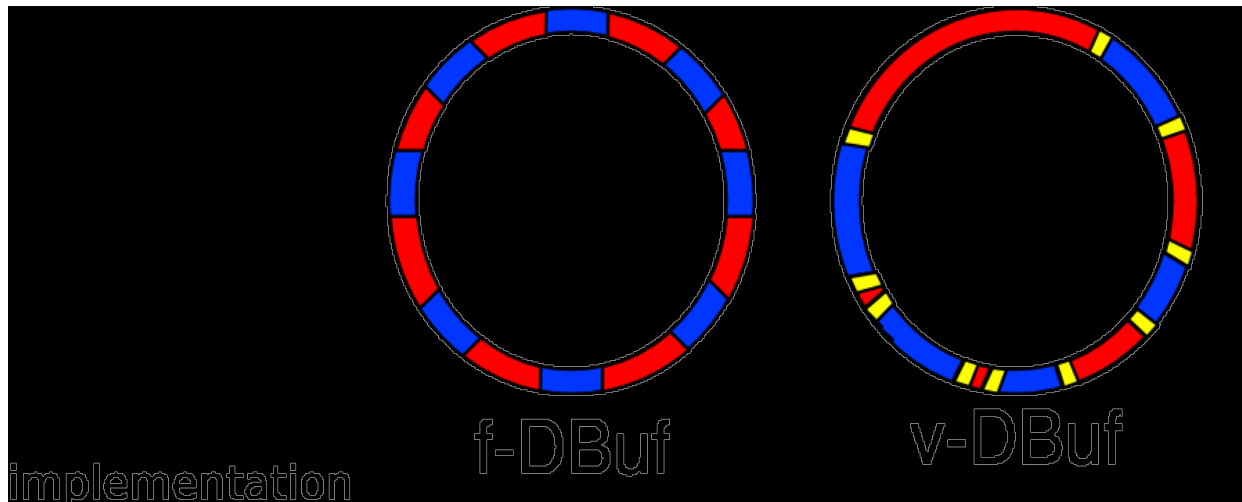
# static shared ring buffers

- advantages:
  - ammortised allocation + mappings
  - copy avoidance
  - cheap sequential access
- disadvantages:
  - memory consumption
  - coarse grain (no per block policies)

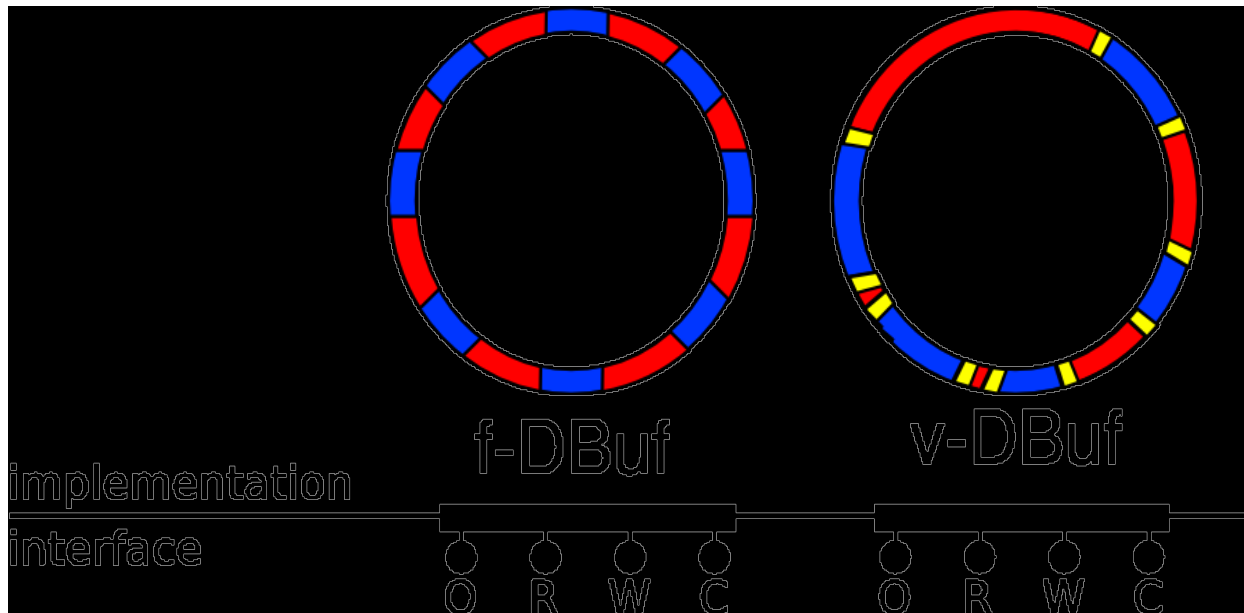
# Ring buffers



# Which ring implementation?

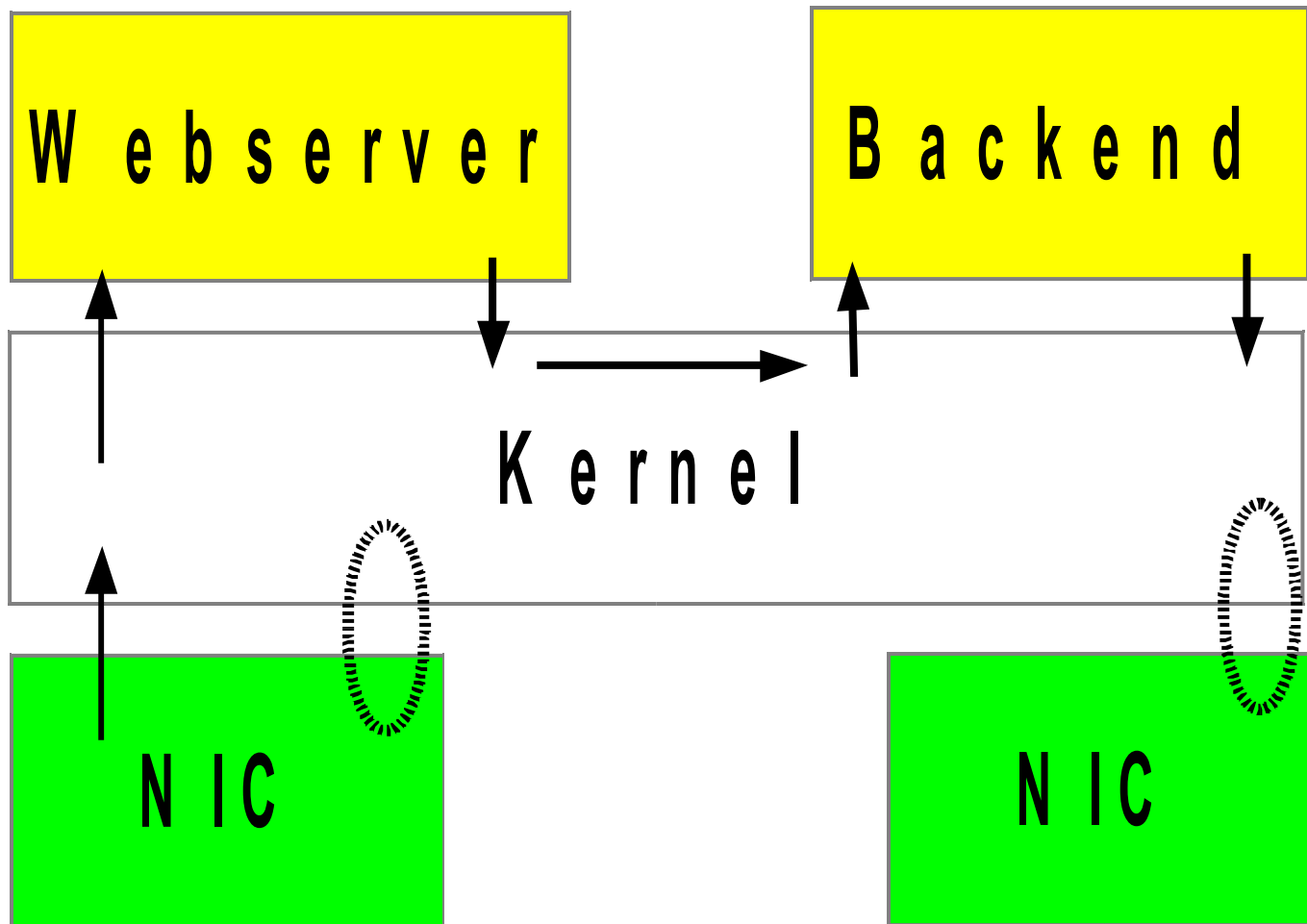


# Which ring implementation?



Posix File API

# One ring to rule them all?



# One ring to rule them all?

W e b s e r v e r

B a c k e n d

K e r n e l

N I C

N I C



# One ring to rule them all?

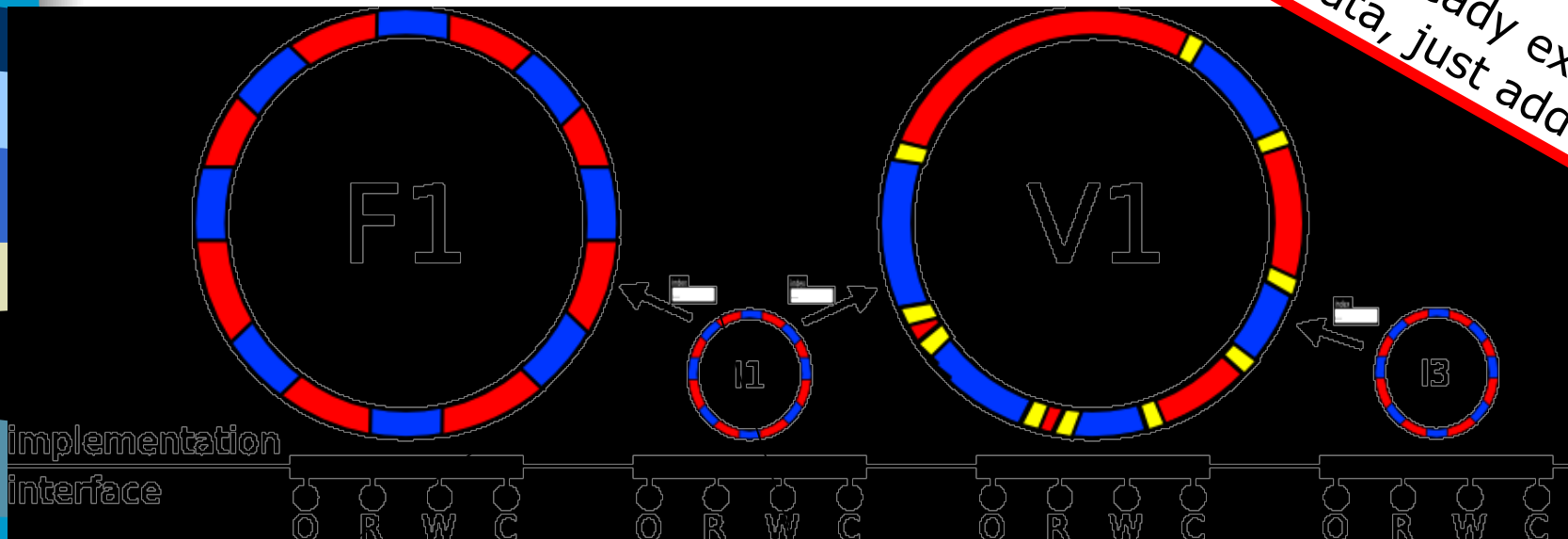
- Won't work because of
- security



- multiprocessing
- distributed memory
- modifications

# copy avoidance

Splice  
write to Ibuf of data that already exists  
in DBUF: do not copy the data, just add a ref



classifier			
buffer id	offset	length	...

read from Ibuf is silently resolved

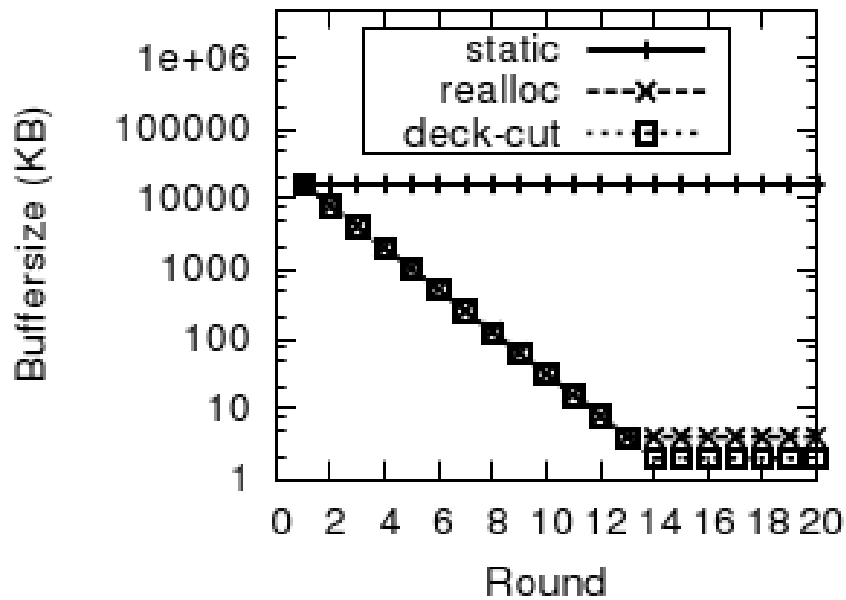


# what about buffer size?

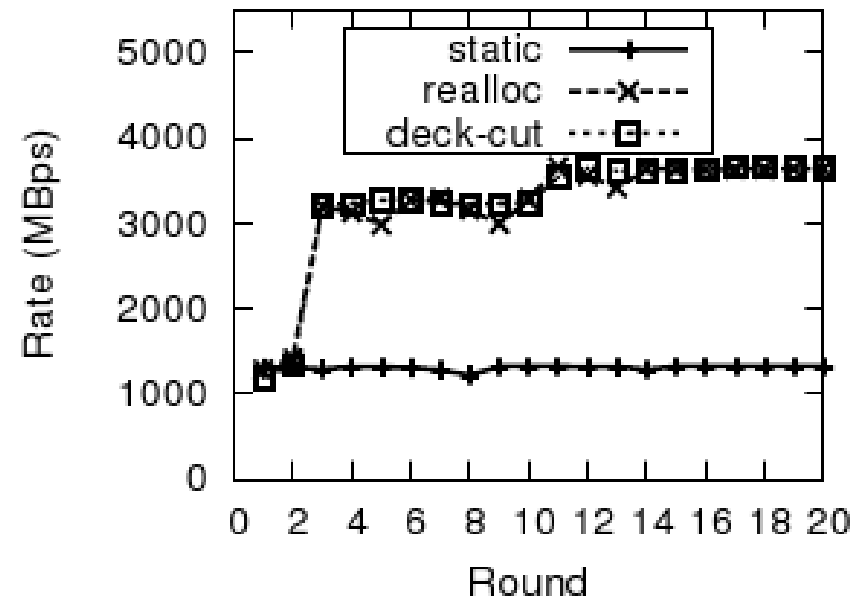
- too small is not good: lots of misses
- too large is not good : lots of misses
- not easy to determine statically  
(cache size varies a lot)
  
- so: scale dynamically



# vary buffer size



(a) Size



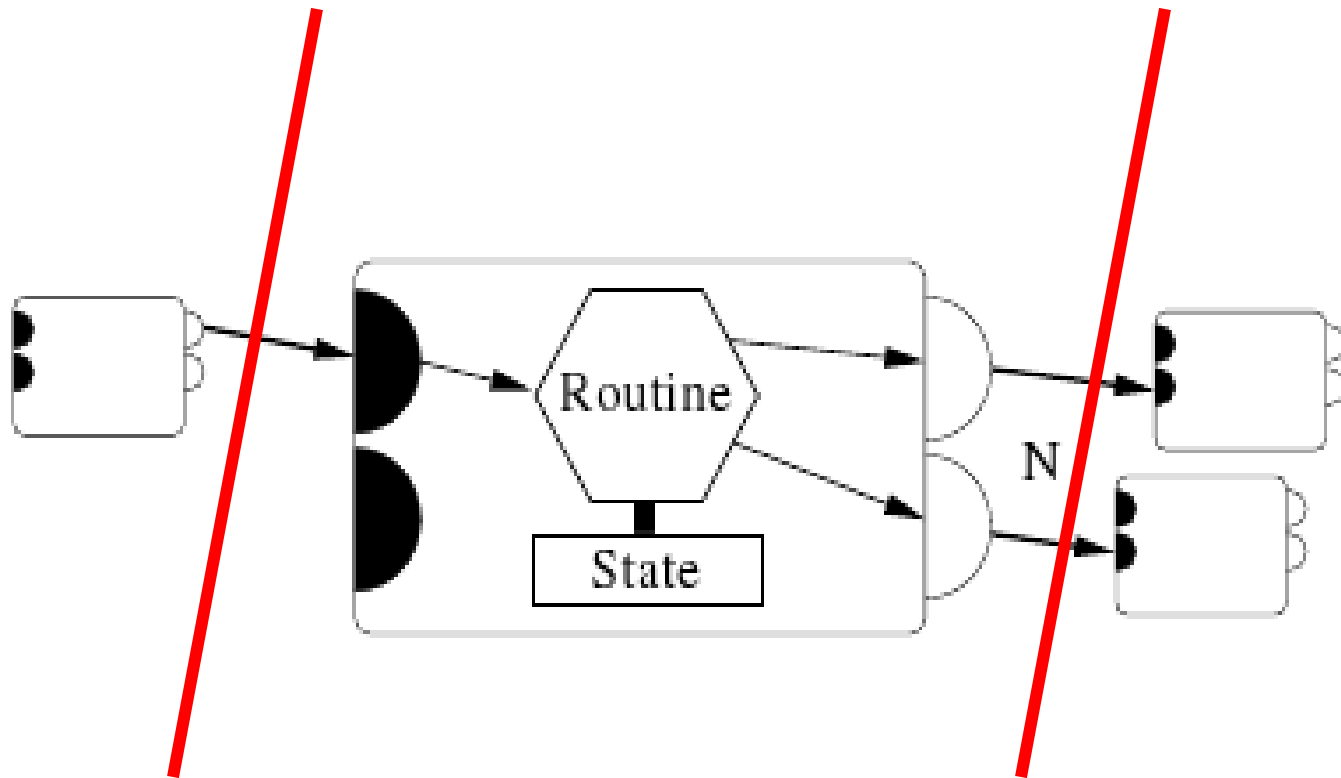
(b) Throughput



# Processing Plane

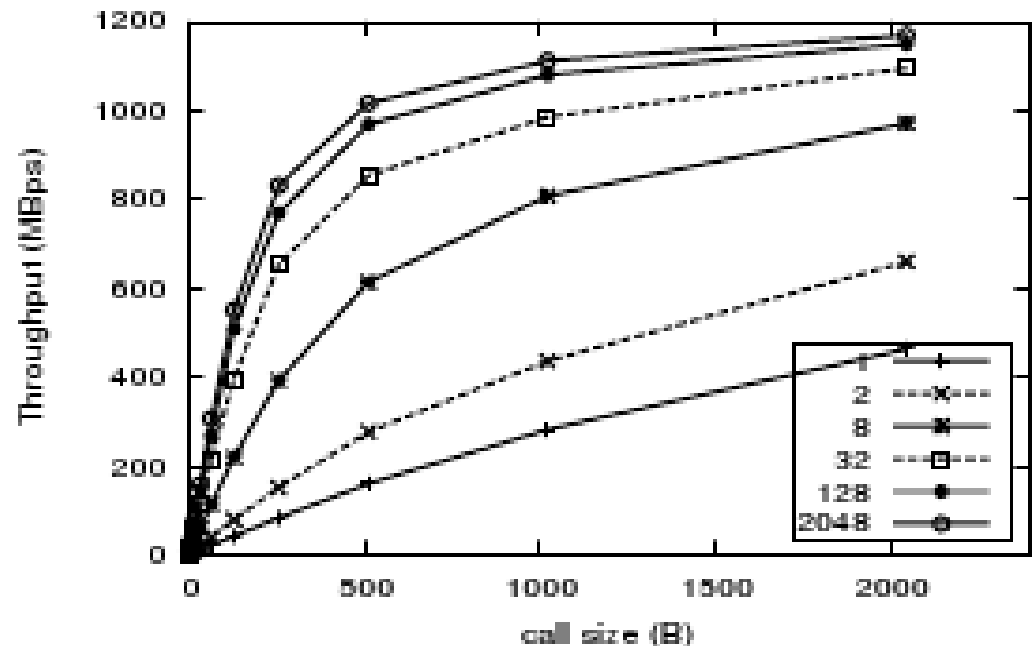
# Processing

- model: extended streams and filters

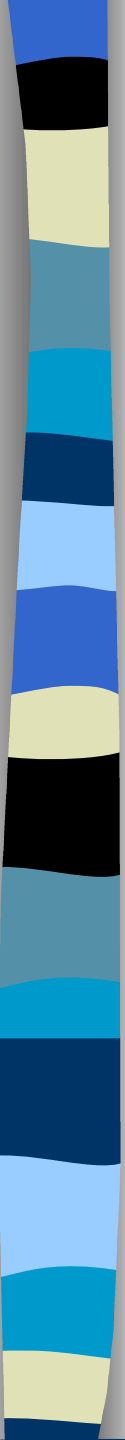


# Runtime system

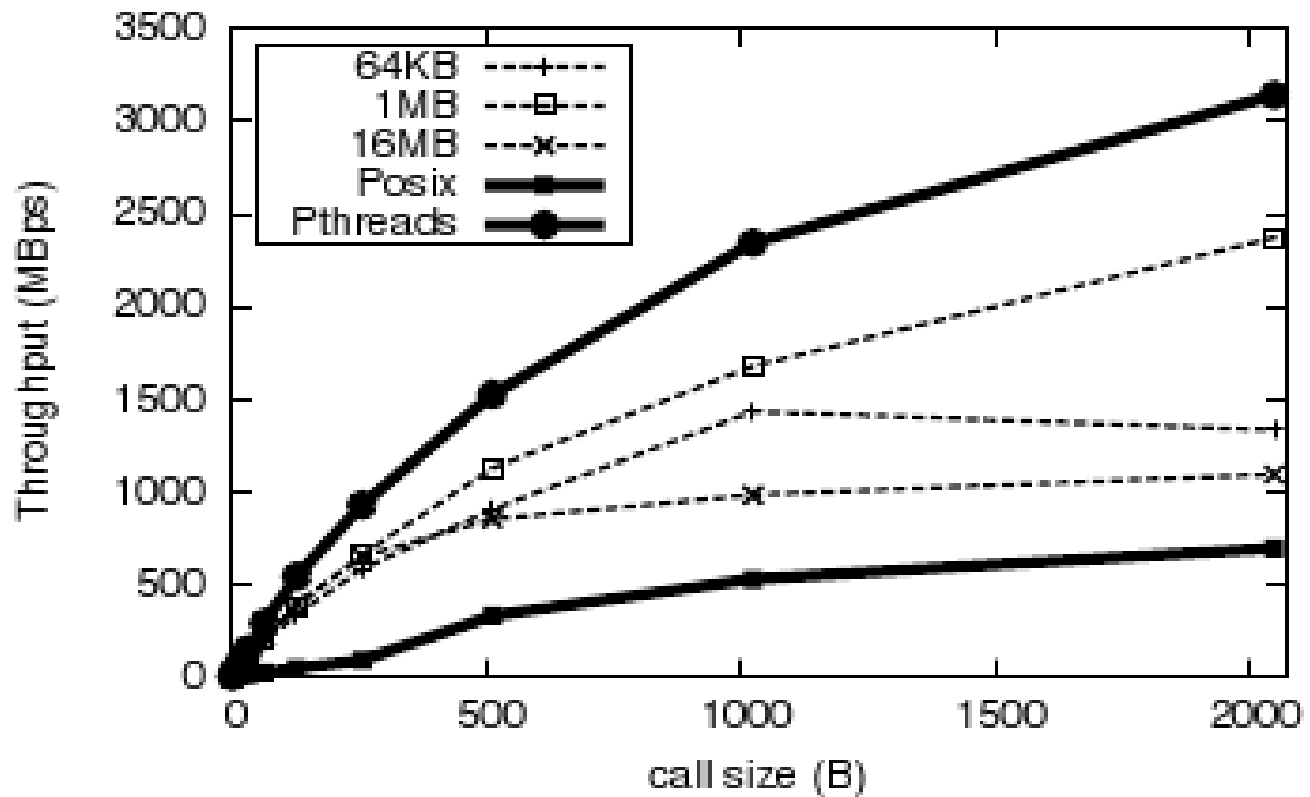
- within a domain: execute as loop
  - good for D cache
- signal coalescing



# Results



# Posix pipes





Scaling up

Can we process at Terabit rates?



# Scaling up

## Can we process at Terabit rates?

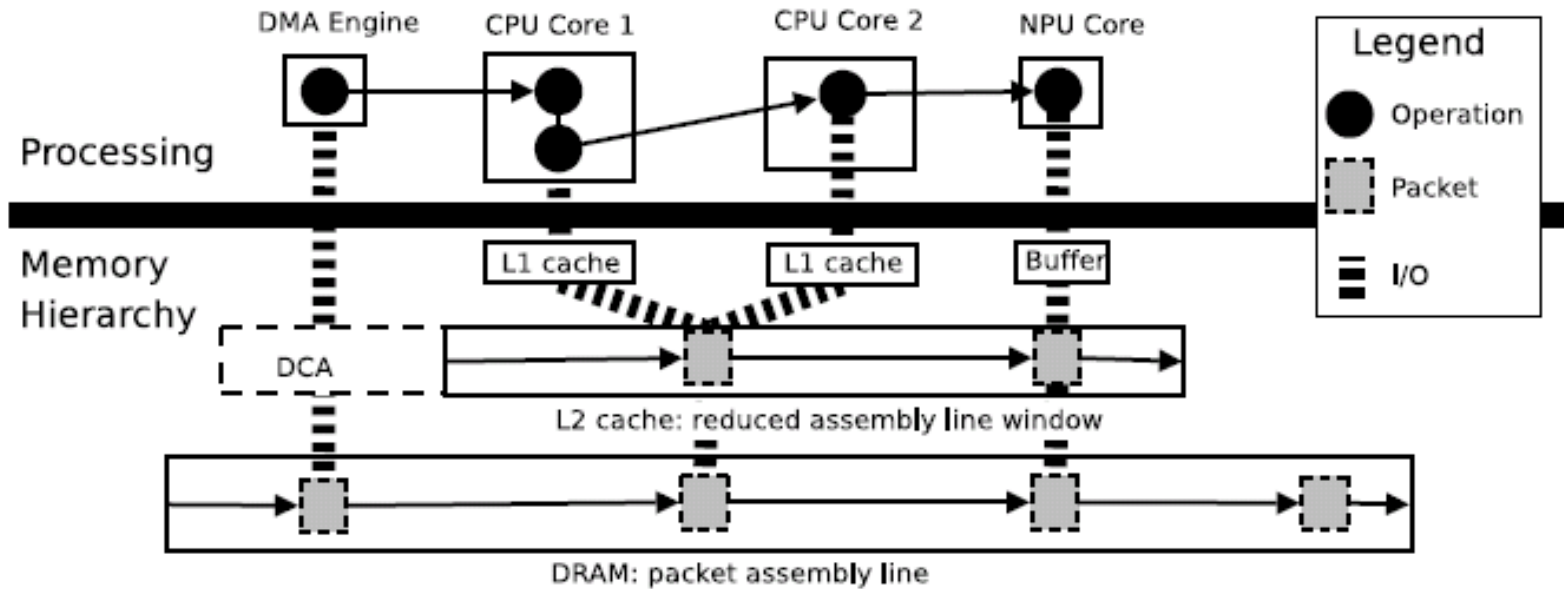
- Closer than you may think
  - NVIDIA GeForce 8800 Ultra  
peak memory bandwidth : 830 Gbps
  - Cell has 200Gbps per node  
(but a slow, shared memory controller)
- Assume manycore
- Forget general purpose



# Model-T

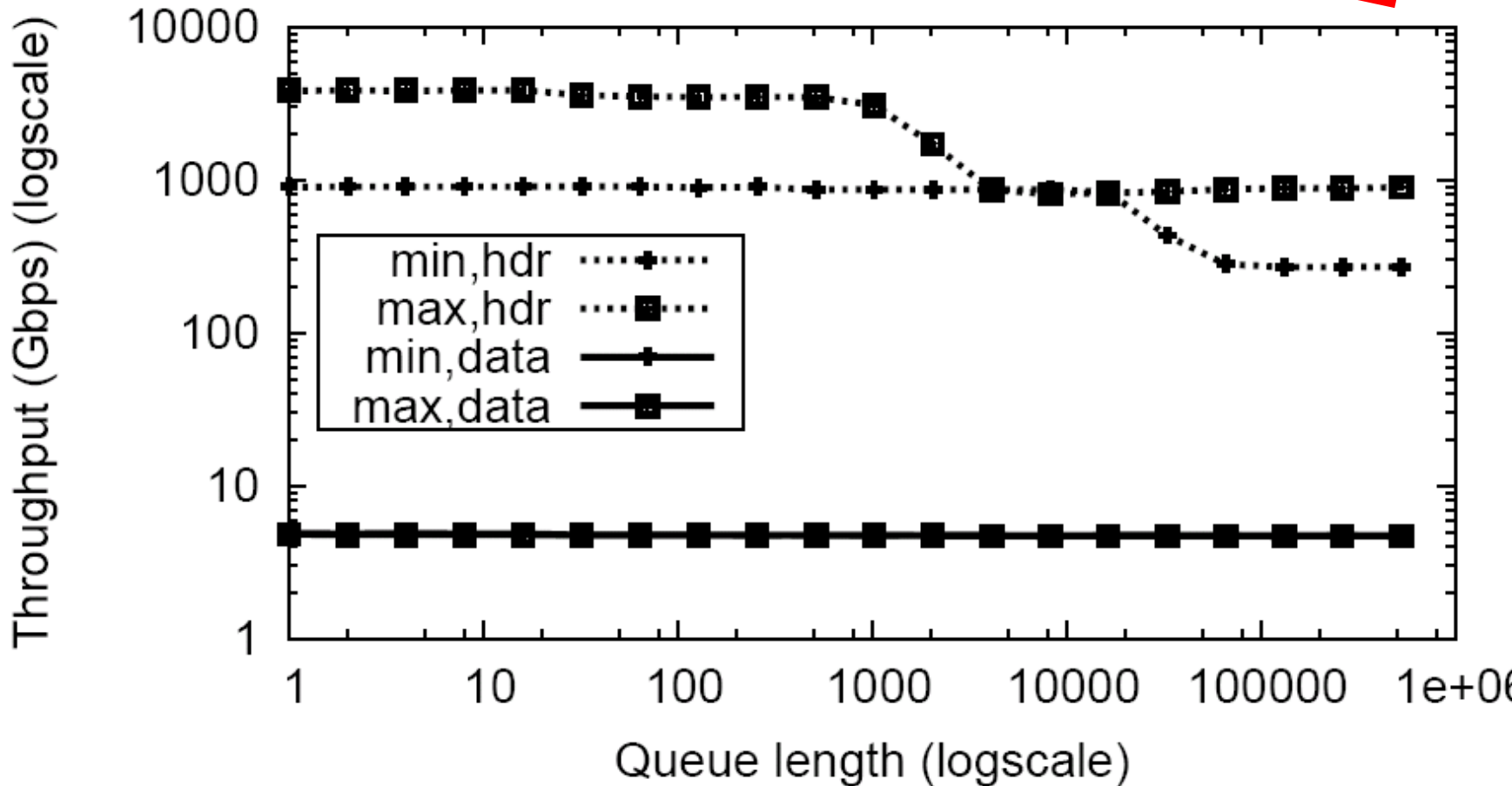
- scale working set to D-cache size
- break up stack in many small operations
- schedule operations
  - in parallel
  - in cache-aware fashion
    - overlapping working sets close together

# Model-T architecture



# Throughput

Core 2 Duo, 2GHz,  
32K split L1, 4MB unified L2





# Conclusions

- current OSs very poor at I/O
- processing at many Gbps possible
- parallelism will help us go further
- useful for
  - routers
  - HPC clusters



# Related work

Container  
Shipping

Pasquale, Anderson & Muller  
IEEE Computer March 1994

Fbufs

Druschel & Peterson  
SOSP 14, 1993

IO-Lite

Pai, Druschel & Zwaenepoel  
OSDI, 1999

Netchannels

Jacobson & Felderman

Splicing

McVoy & Torvalds

# DNS serving with Bind

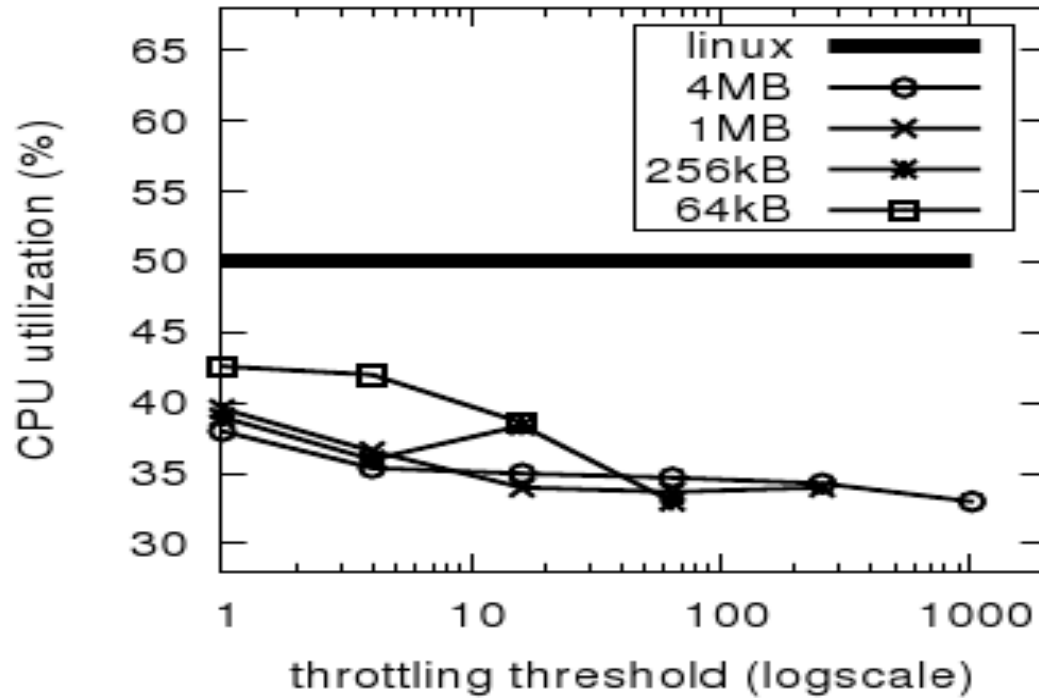
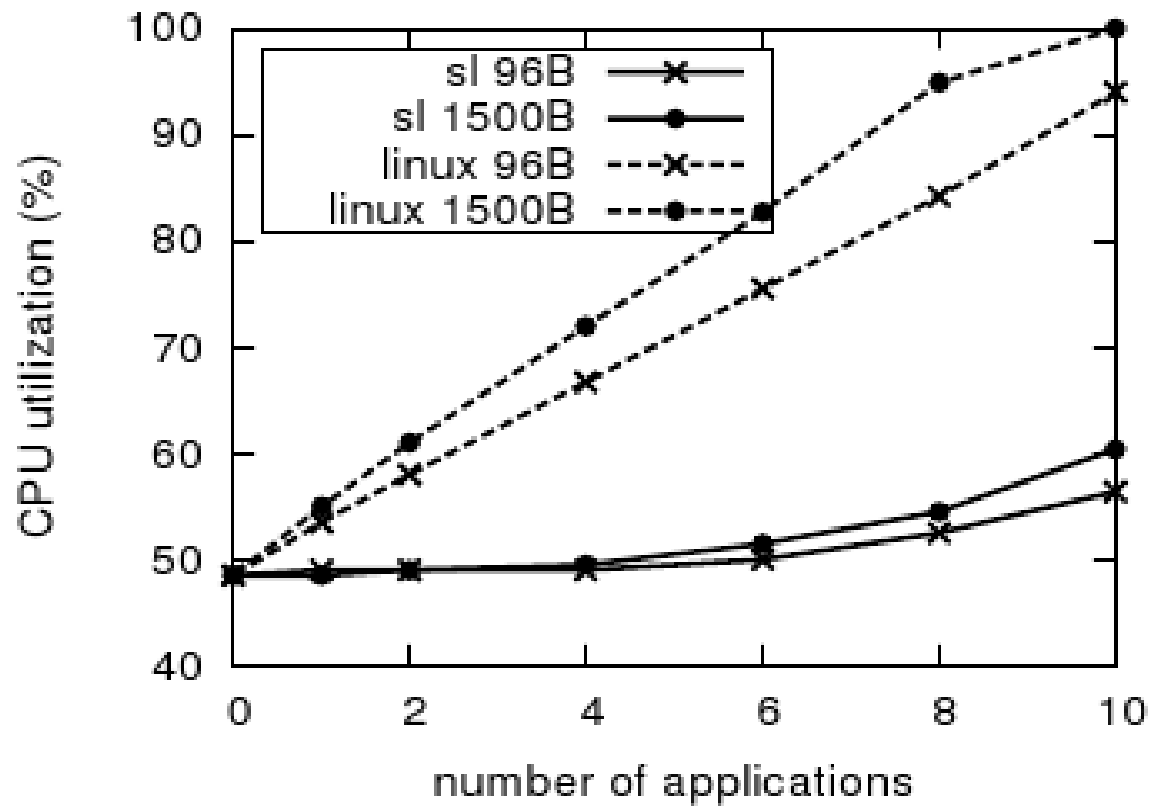
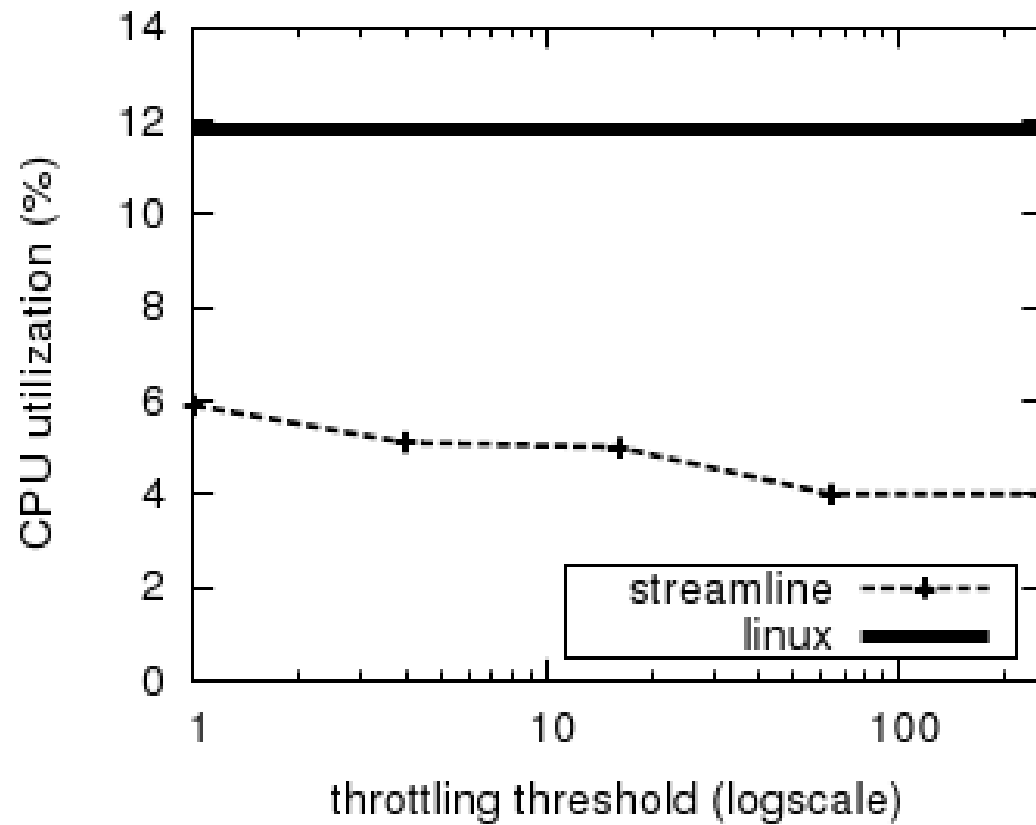


Fig. 12. Bind

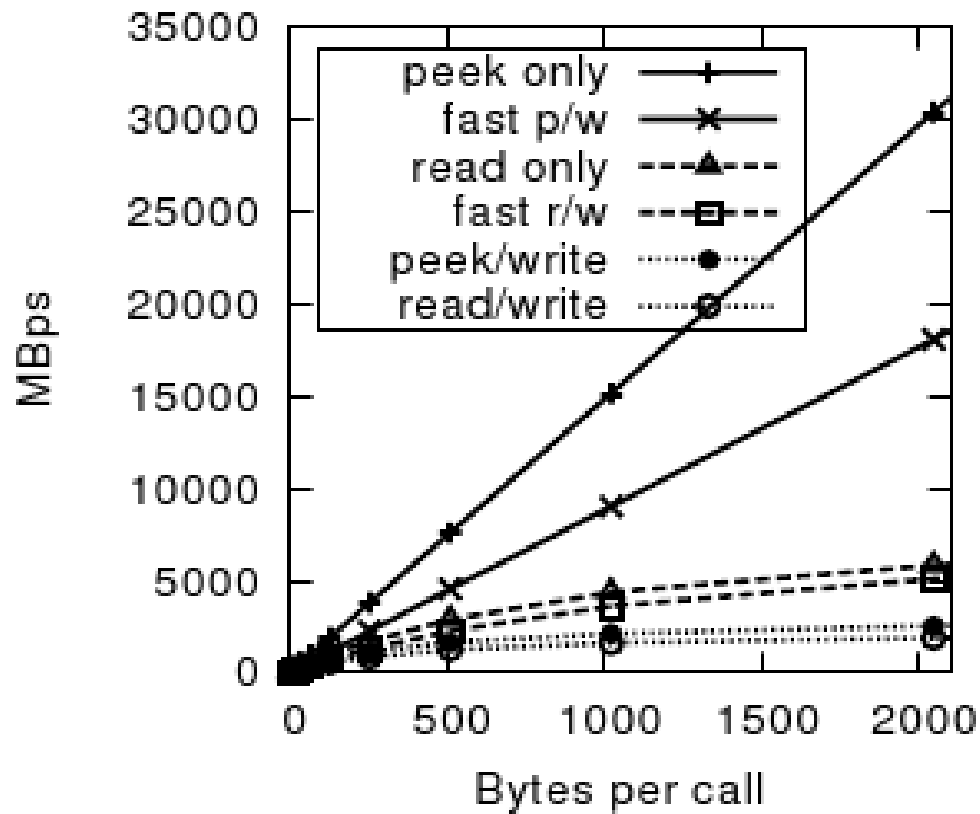
# Tcpdump



# mplayer

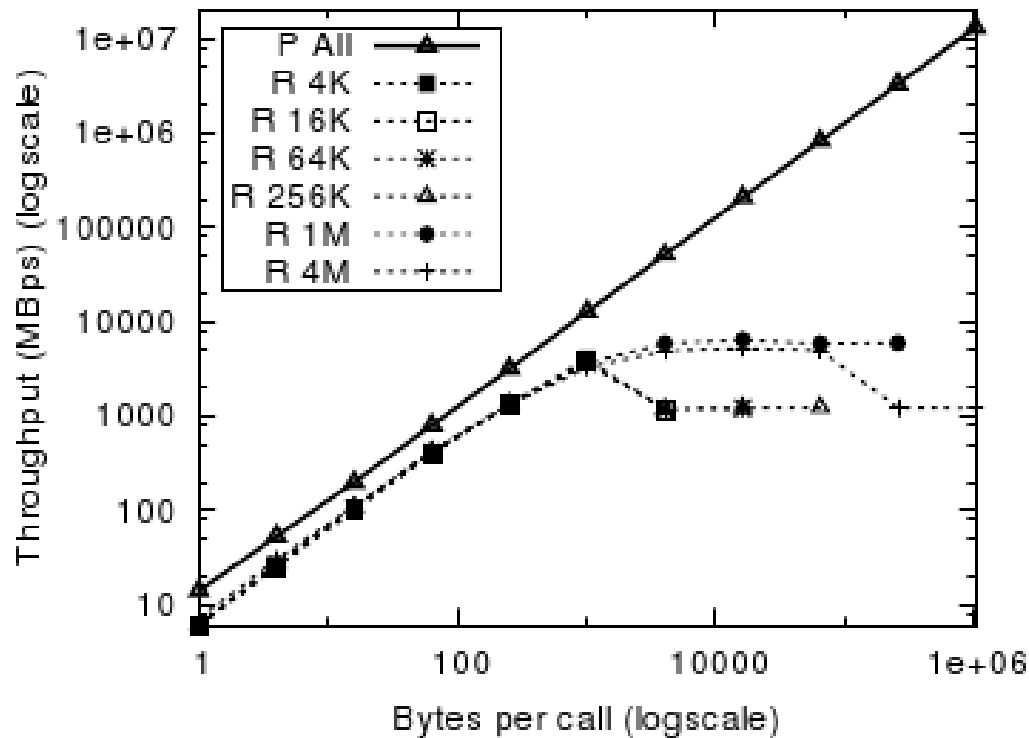


# splicing helps



# One extension

- peek: read without copy



# Control Plane





# Control

- Streamline's 'native' interface for specifying I/O paths
- Looks like UNIX pipes and redirections:
  - (tcp) > (http) > (files)
  - (tcp) > ( (http) | (ssh) | (rtsp) )
  - (dport) -22:23-> (log)



# Placement and transformation

- heuristics
- possible compilation



# Other APIs

- pipes
- sockets
- pcap
- pipesFS



# PipesFS is interesting

- directories represent filters
- pipes represent streams
- Each directory contains:
  - one pipe *all* that holds the filter's buffered output stream
  - numbered subdirectories for all observed classes, each again with its own pipe.
- legacy file ops simply work
- so do tools (mkdir, link, cat)



# pipesFS

- trivial to script together kernel I/O tasks
- `mkdir -p /pipes/rx/untcp/httpcat`  
`/pipes/rx/untcp/http/get/all | compress >`  
`log.Z`

# Filters

- for each block arriving on input port
  - call filter routine
  - may forward datablocks onto output ports (actually the indices are passed)
  - set the classifier field in the Ibuf pointer

