

Memory Management Strategies for Data Serving with RDMA

Dennis Dalessandro

and

Pete Wyckoff (presenting)

Ohio Supercomputer Center

{dennis,pw}@osc.edu

HotI'07 23 August 2007

Motivation

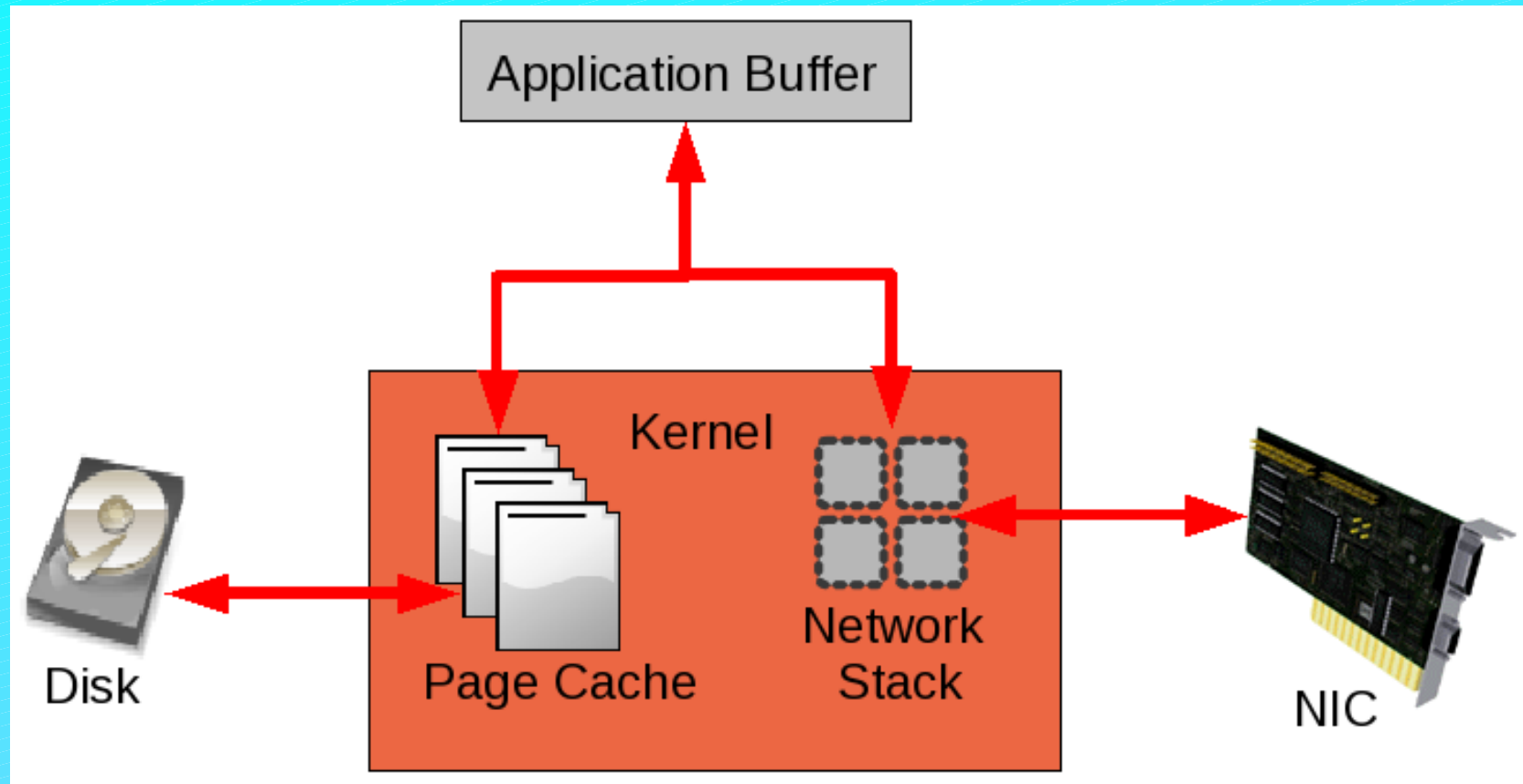
- Increasing demands for
 - bandwidth
 - client scaling
 - dynamic content
 - more interactions
- File transfer is a fundamental component
 - distributed file systems
 - NFS, PVFS, CIFS
 - distributed data-intensive protocols
 - FTP, HTTP, DAV, Bittorrent
 - video-on-demand, multicast conferencing

Can RDMA improve file transfer performance?

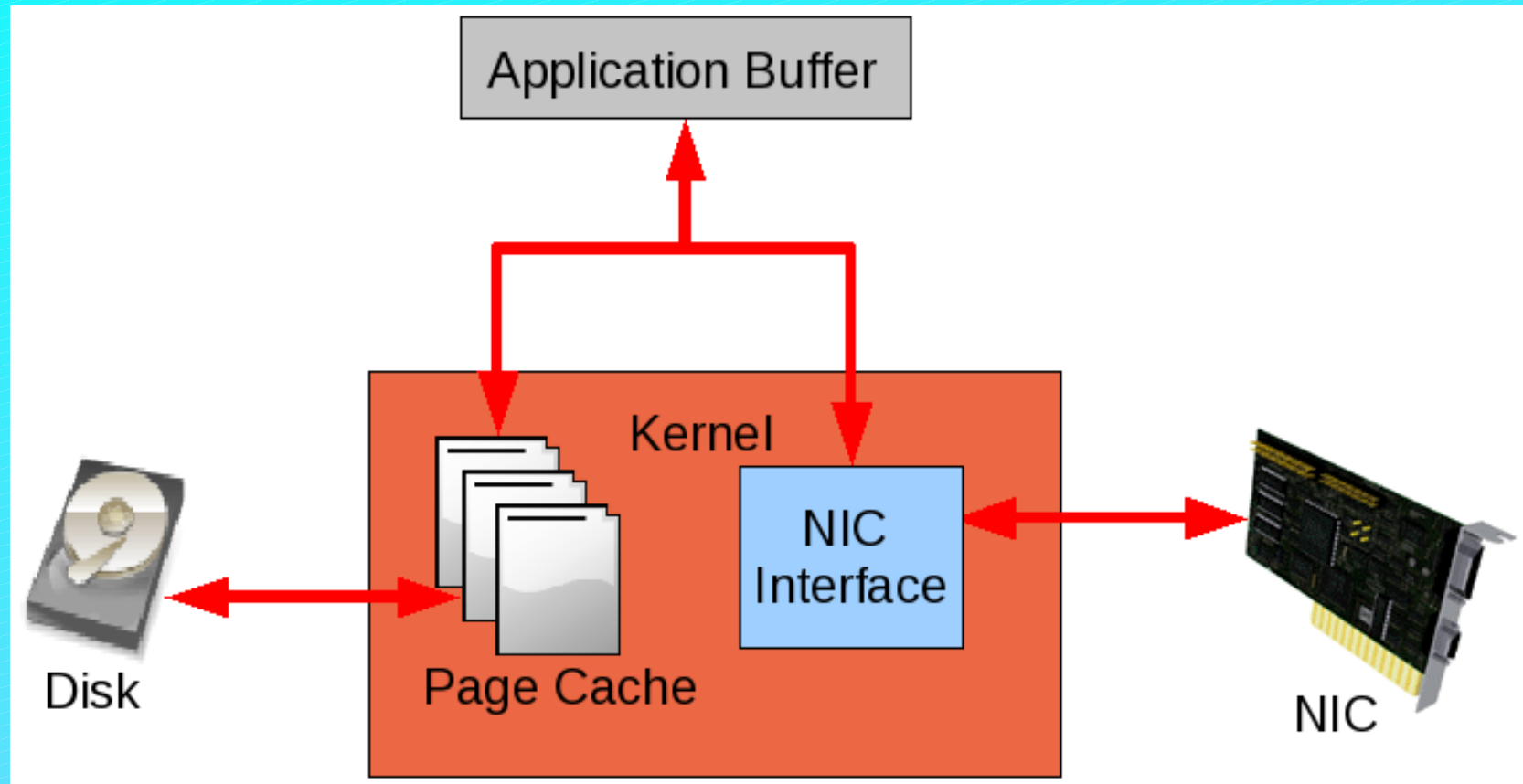
What is RDMA?

- Two major aspects
 - Protocol Offload
 - NIC handles network processing
 - Removes biggest burden from host CPU
 - Zero Copy (with or without OS bypass)
 - Data is moved directly between network and user buffers
- TCP Offload Engine (TOE)
 - TCP/IP stack processing offloaded
 - CPU still moves data from buffers to user memory
- Remote Direct Memory Access (RDMA)
 - TCP/IP offloaded
 - Data goes directly to and from user buffers

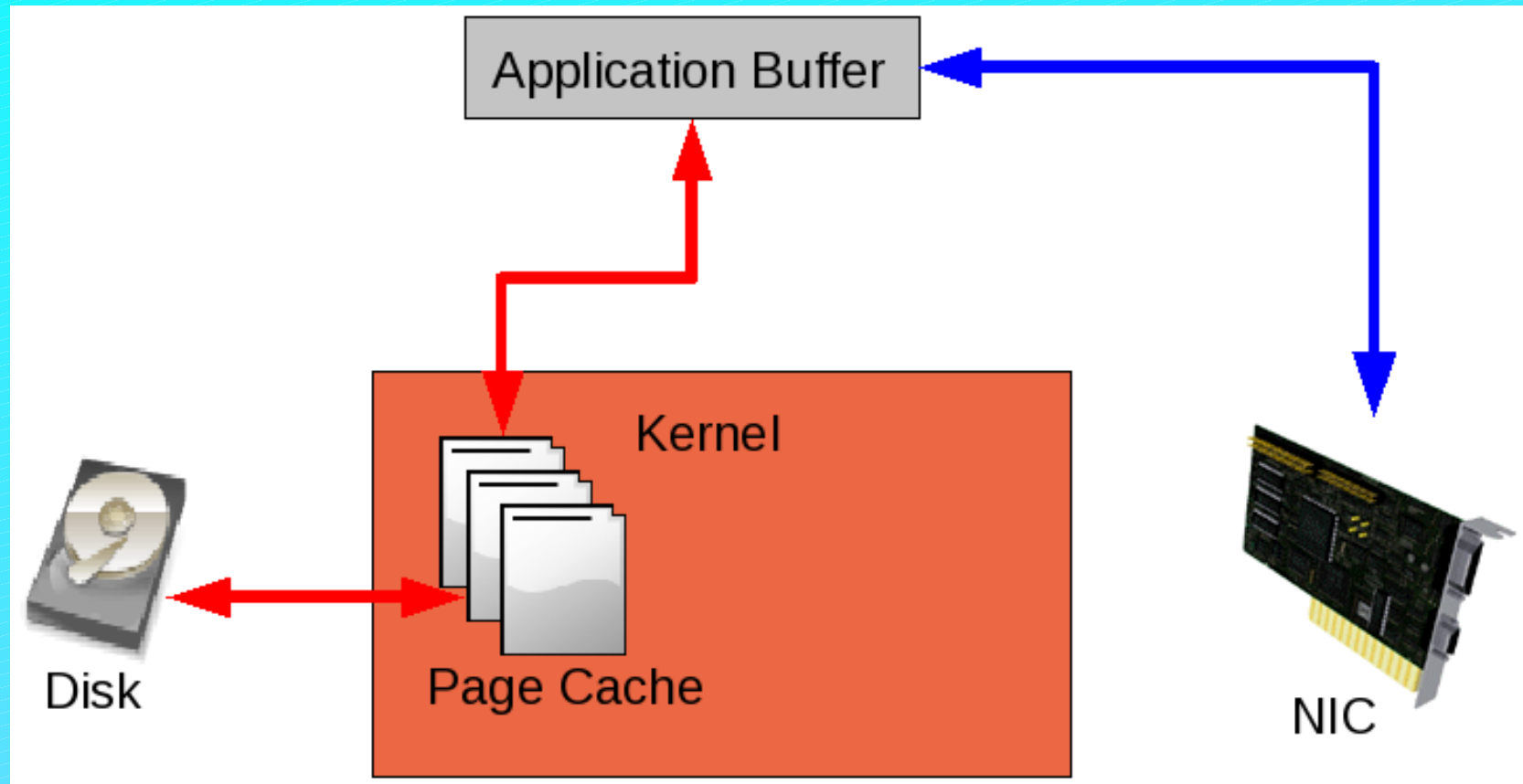
Traditional Network Processing



TCP Offload Engine Processing



RDMA Network Processing



RDMA Usage

- High-performance computing
 - Message passing (MPI)
 - InfiniBand, Myrinet, Quadrics, iWARP
 - File systems
 - PVFS, Lustre, NFSv4
- Various back-end applications
 - Database (unreliable datagrams)
 - Storage interconnect (non-switched)
 - IPoIB or SDP for sockets API on faster hardware
- Little general purpose usage
 - Web serving? back-end only
 - Wide-area communications? rare point-to-point

Why is RDMA limited to HPC?

- Another incompatible fabric
 - Not with iWARP, runs on Ethernet
- High cost of 10 Gb/s Ethernet
 - Dropping rapidly, like Ethernet always does
- Sockets API
 - Completely entrenched
 - Cannot handle RDMA semantics
 - Sockets “extensions” are poor compromises
 - RDMA is not integrated into OS

RDMA in the Linux Community

Recent email from David Miller (Net maintainer)

> *Isn't RDMA part of the “software net stack”*

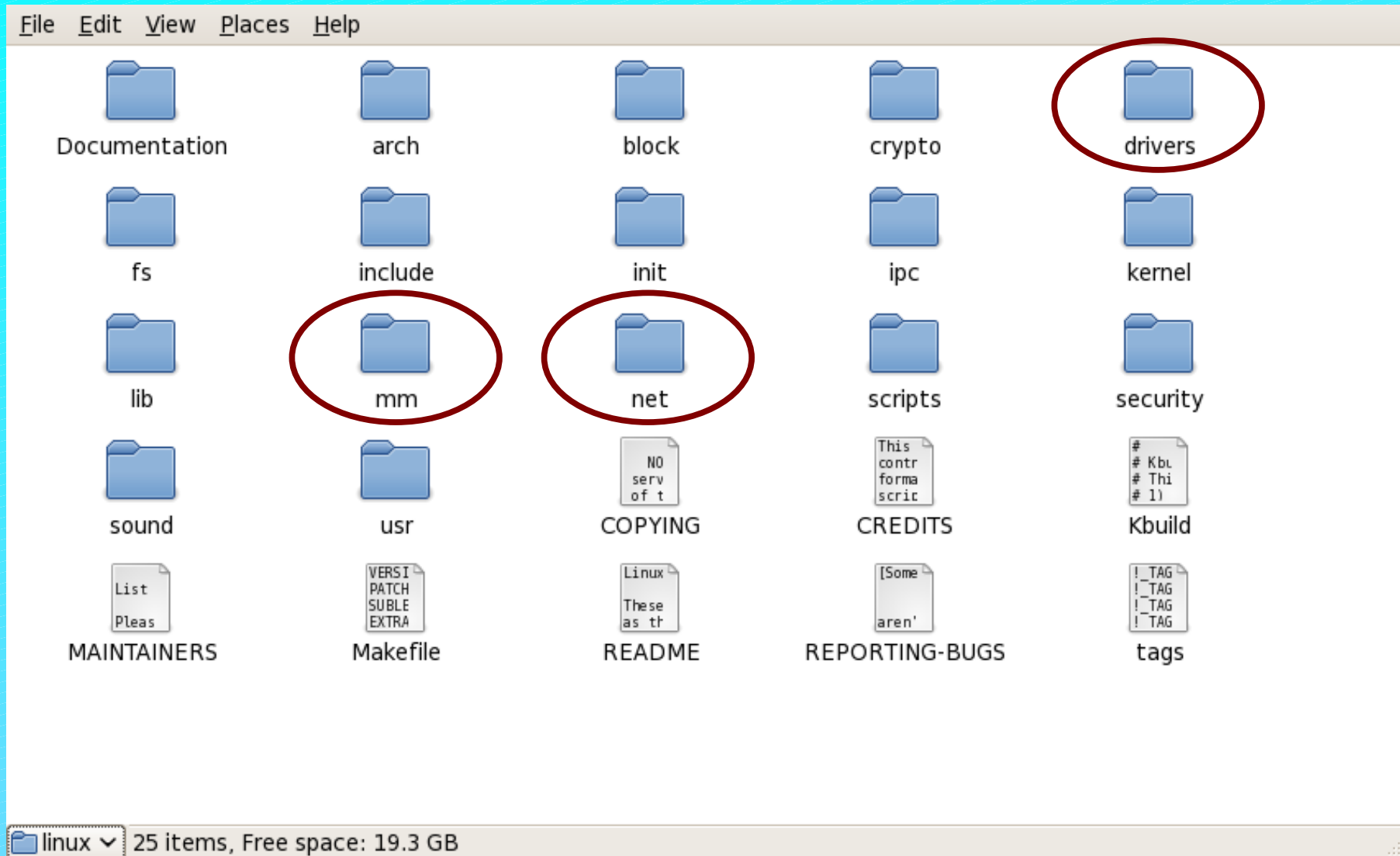
> *within Linux?*

It very much is not so.

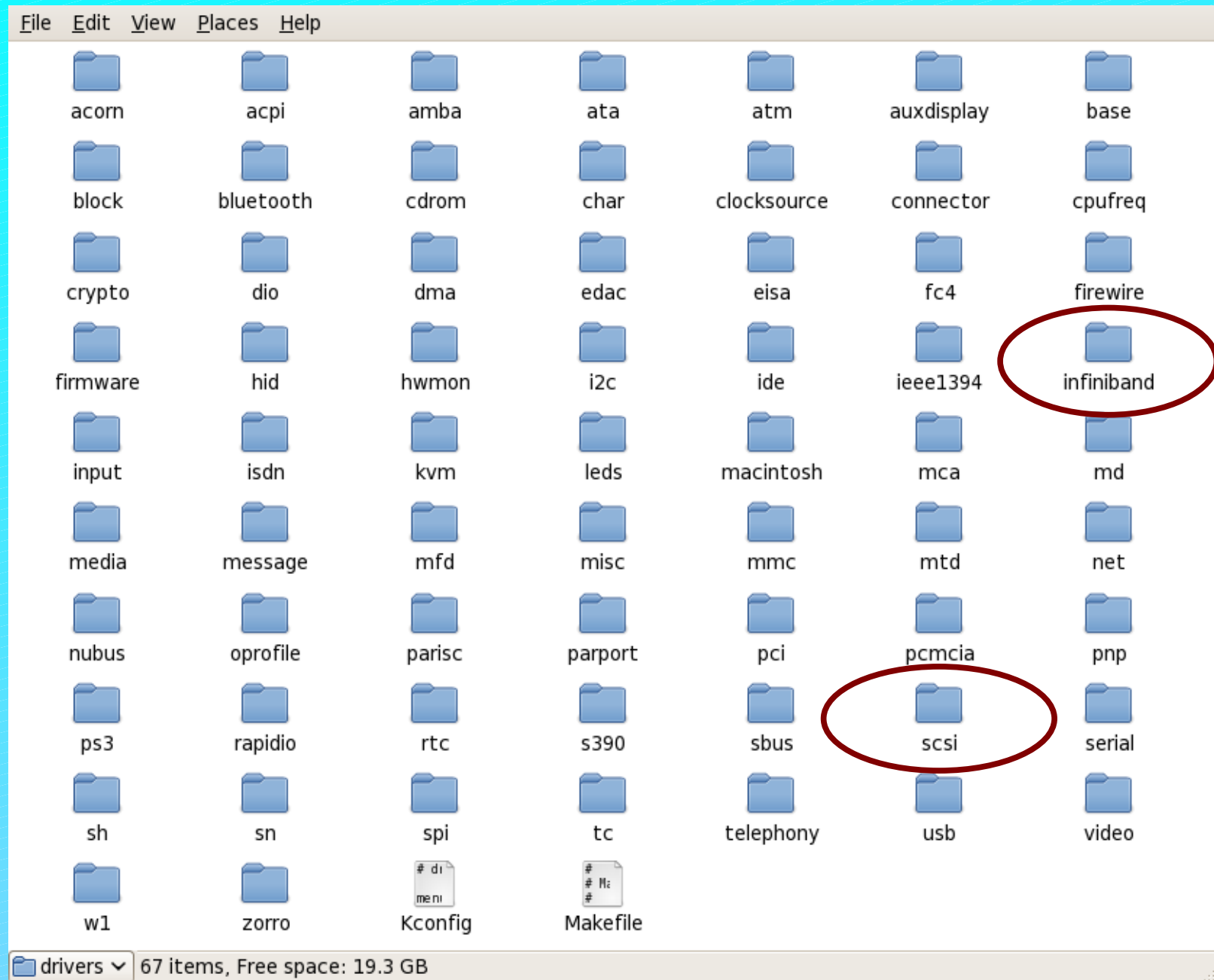
Major complaints

- Firewalling, packet shaping and classification not possible
- Stateful devices bad
- Multi-core will make RDMA obsolete

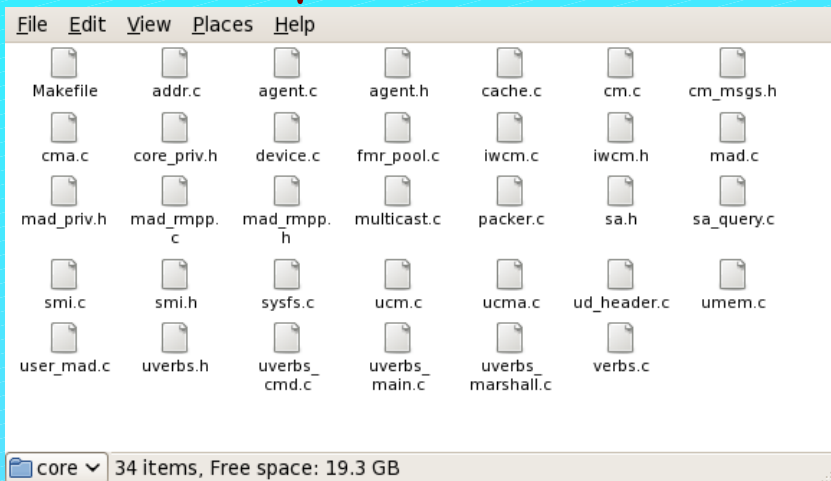
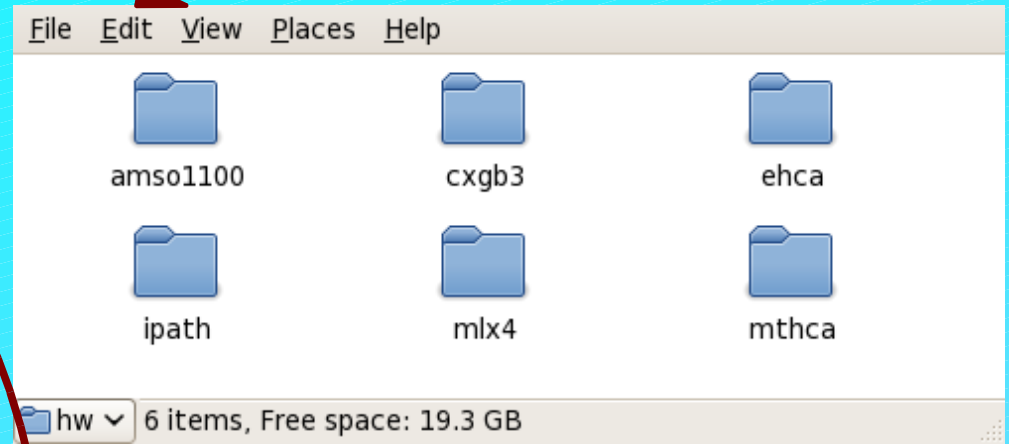
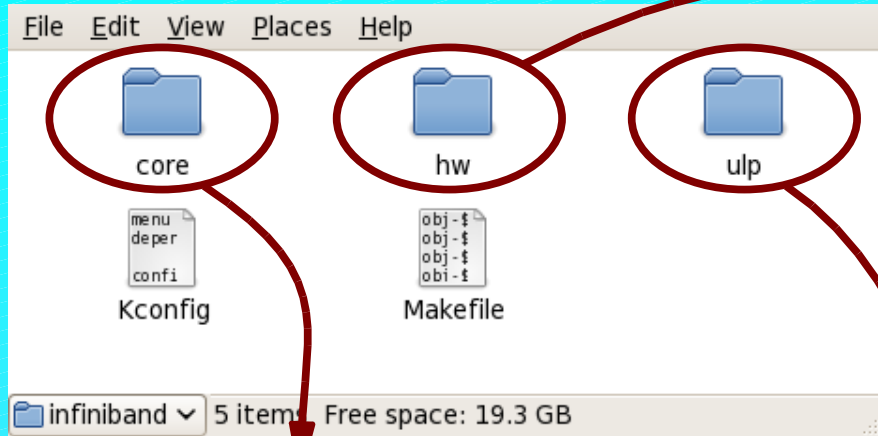
Linux kernel tree



Linux “drivers”



InfiniBand Components

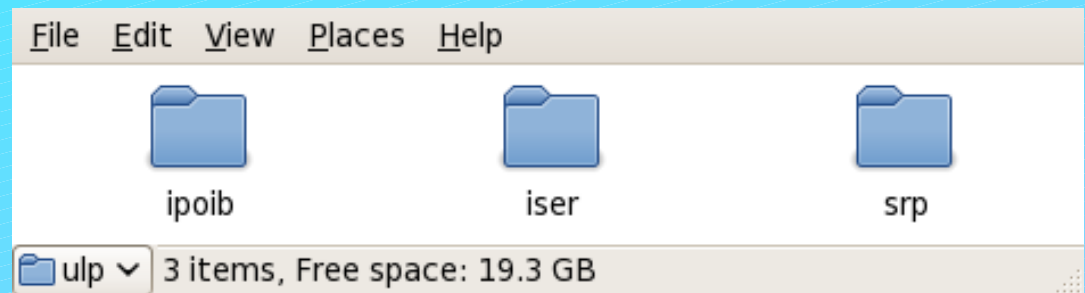


Core

- Verbs interface
- Memory registration
- Connection manager

Hardware

- True device drivers



Upper Layer Protocols

- Ethernet interface
- SCSI transport (iSCSI)
- SCSI transport (SRP)

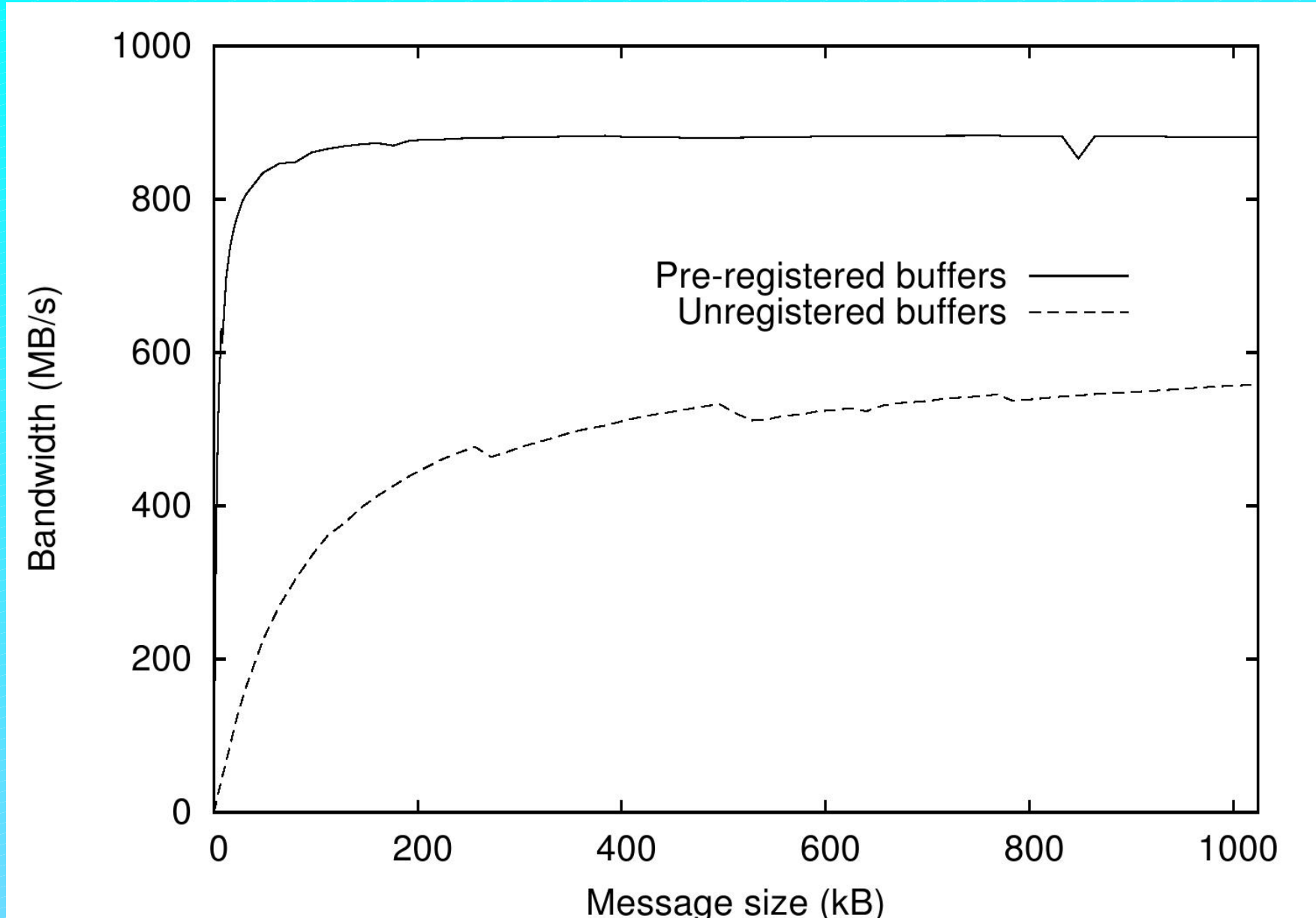
RDMA and OS Mismatches

- Memory registration
 - API exists, but is not part of memory management
 - mmap, munmap
 - mlock, munlock
 - ibv_reg_mr, ibv_dereg_mr
- Connection management
 - API exists and looks like sockets, but separate
 - socket, bind, listen, connect, accept
 - rdma_create_id, rdma_bind_addr, rdma_accept, ...
- File descriptor abstraction
 - RDMA uses Queue Pair and other entities
 - No sharing of QP across user/kernel or processes
 - No unified polling mechanism (readable/writable)

Memory Registration

- Severe application complexity
- Two major approaches
 - static
 - copy all data into and out of fixed buffers
 - allocate per-client resources
 - dynamic
 - register buffers as they are needed
 - possibly cache previous registrations (but not in system)
- Overheads
 - 30 us to register 4 bytes
 - 100 us to register 1 MB
 - 30 to 50 us to deregister
 - 2 to 7 us network latency!

Throughput with Registration

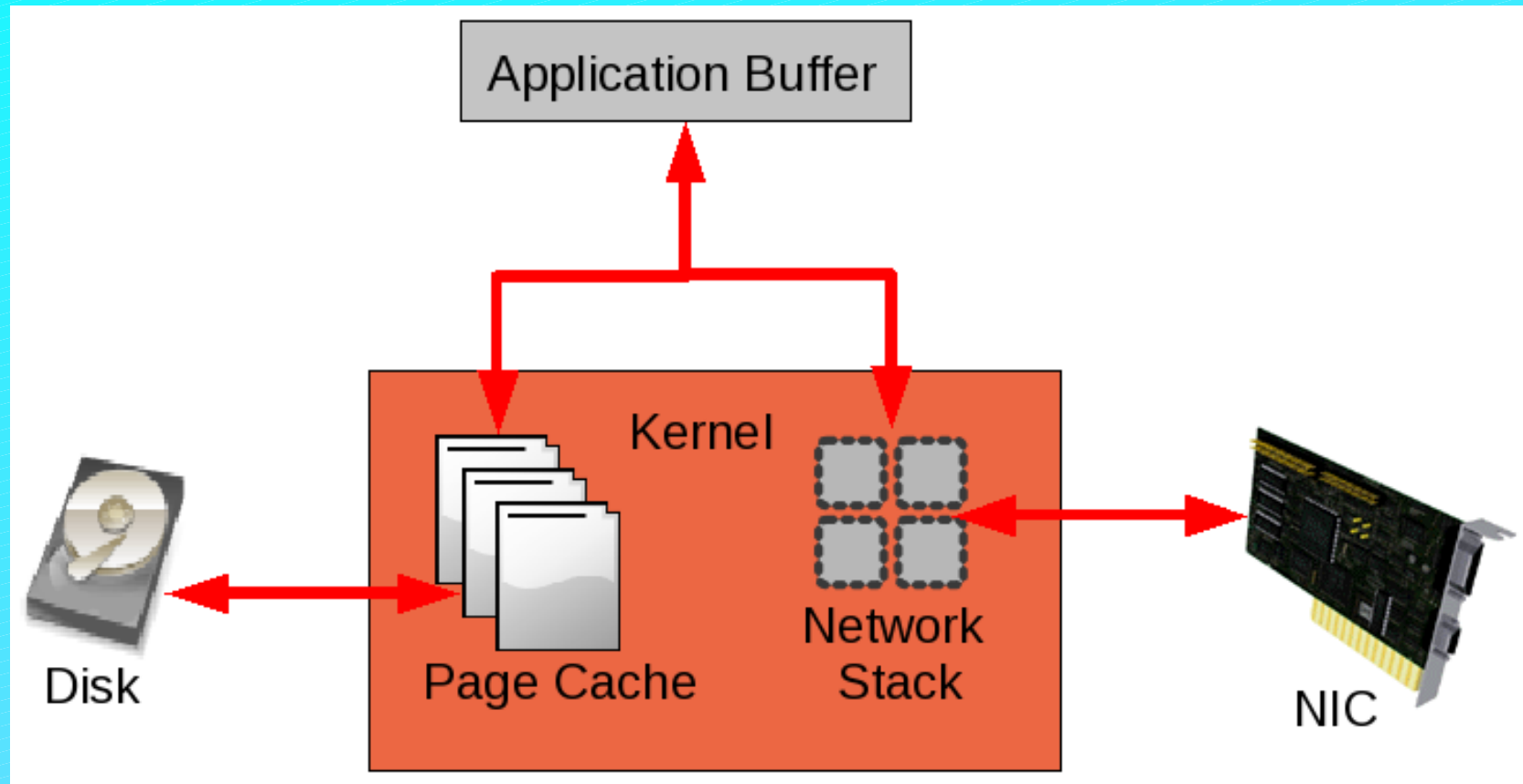


Sendfile

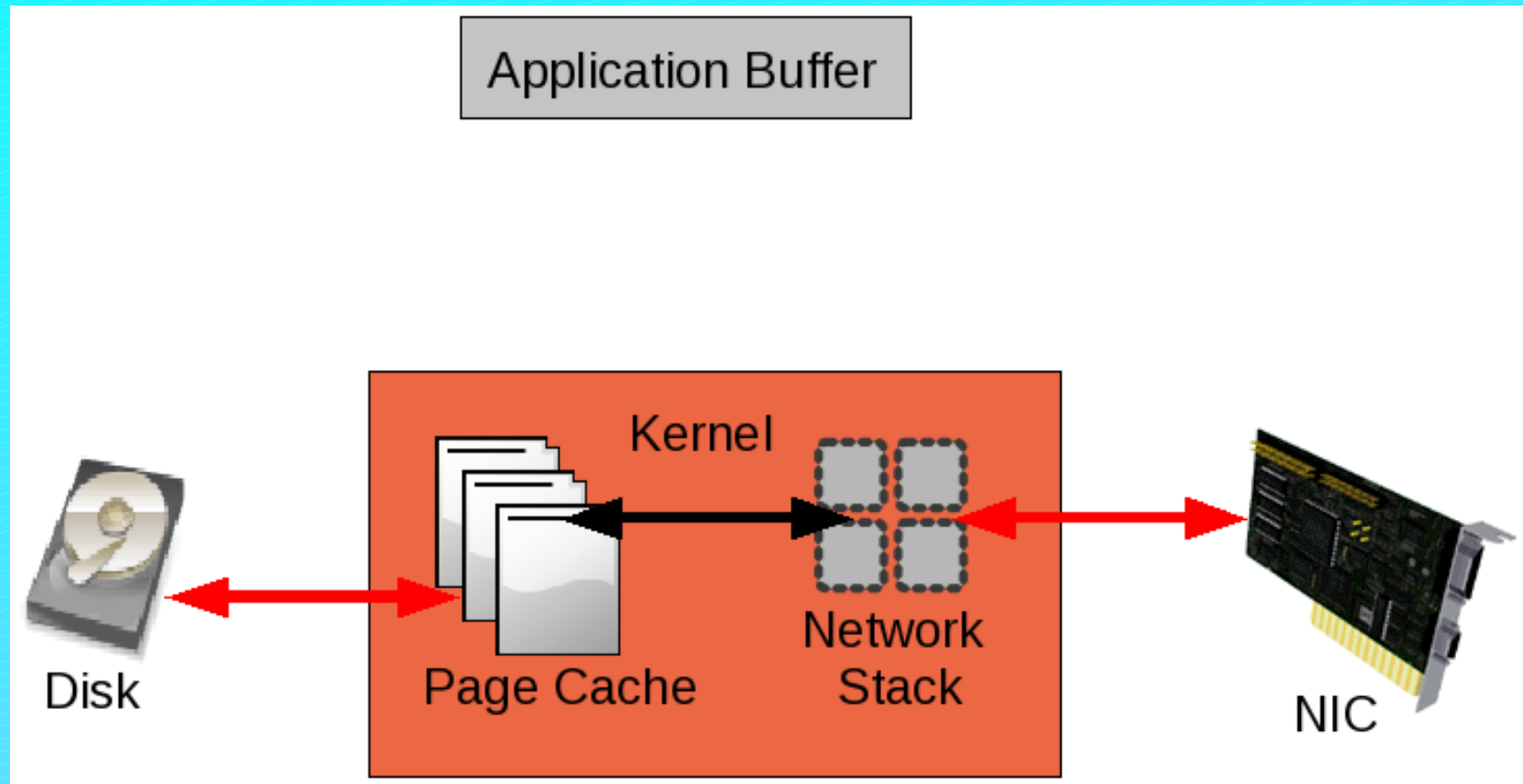
- Ship file data directly to the network
- In Linux, BSDs for many years
- Saves two memory copies for TCP
- Commonly used by Apache and other apps
- Uses file descriptors to represent
 - local file
 - socket connection

Maps poorly to RDMA

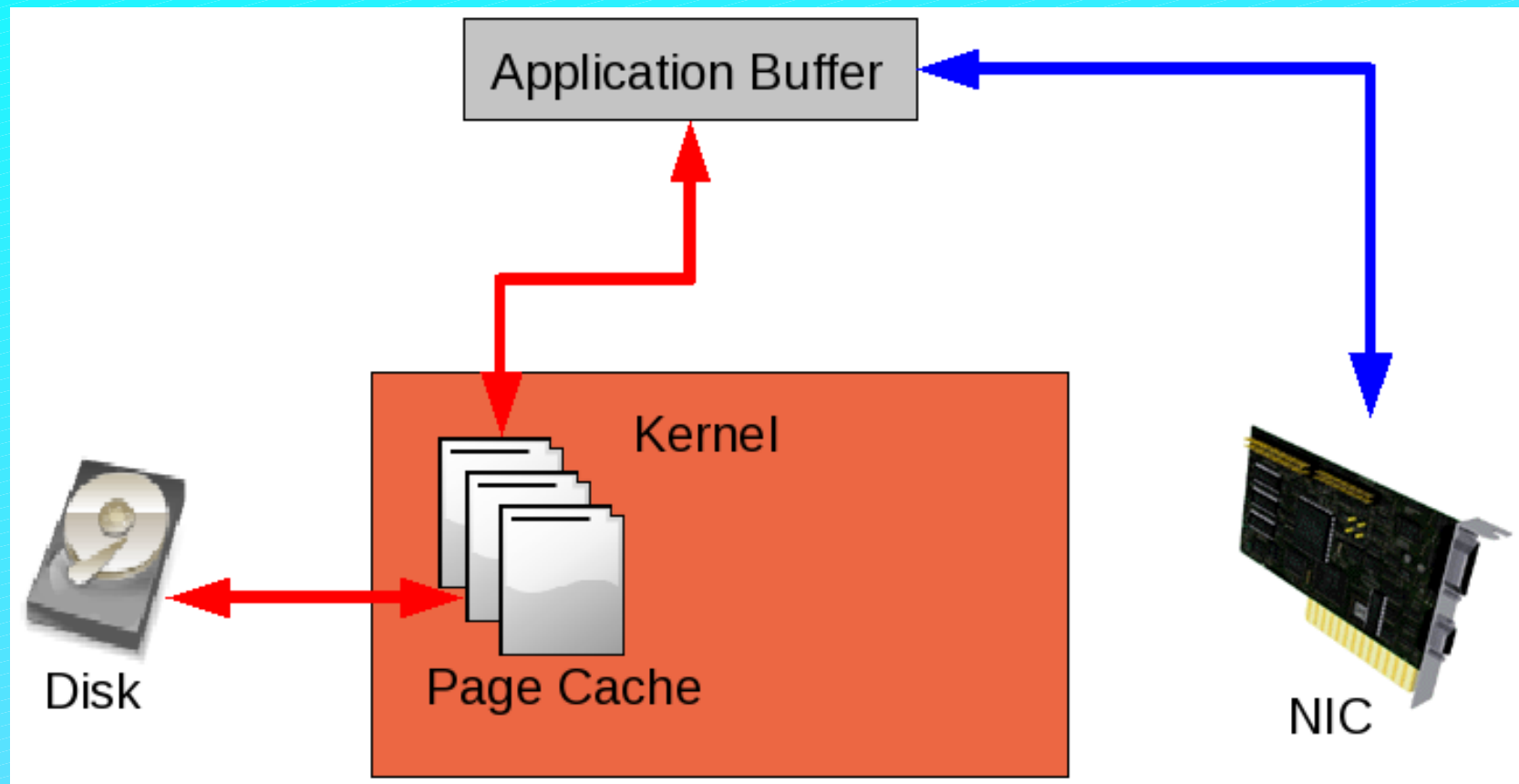
TCP Network Processing



TCP Sendfile Processing

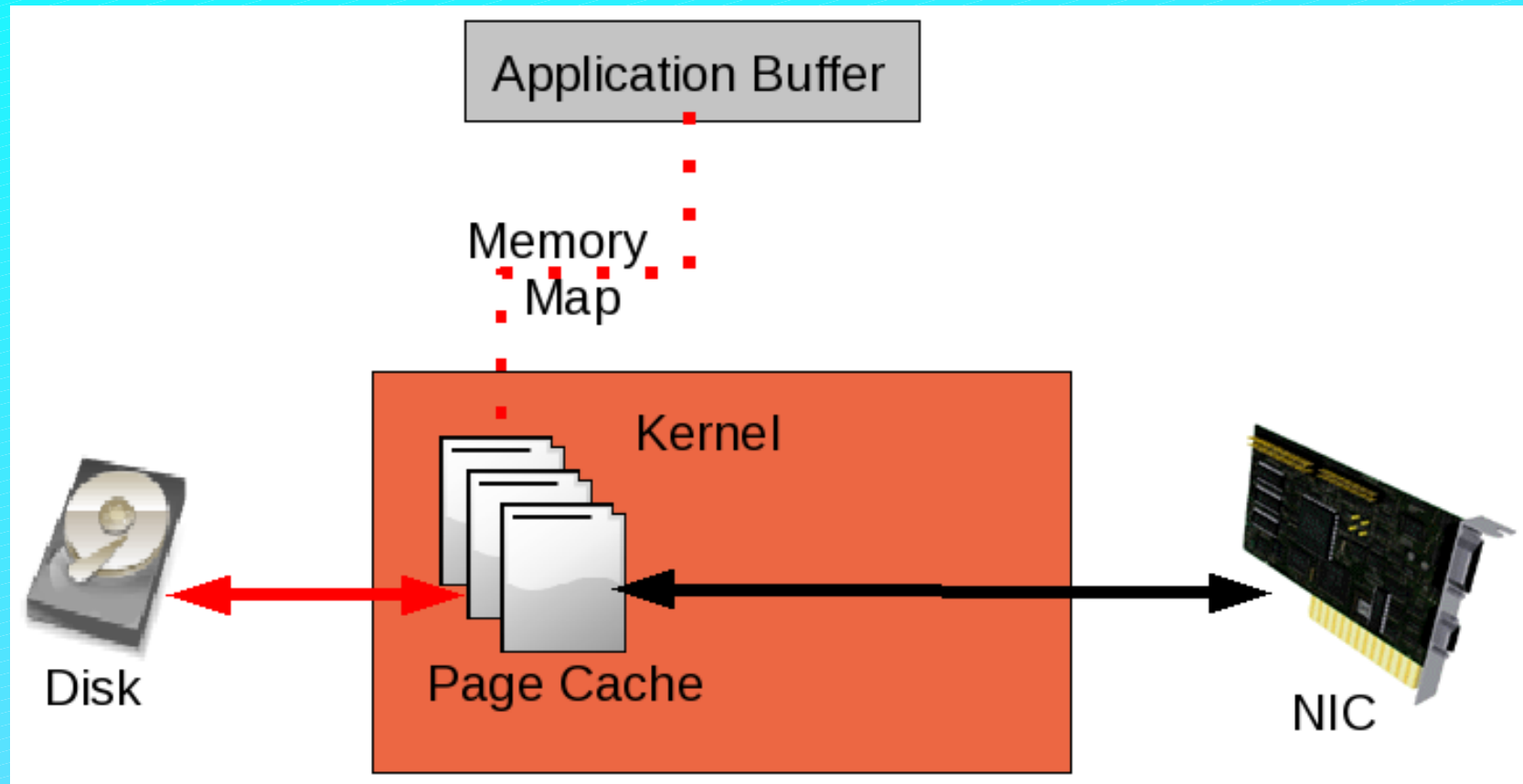


RDMA Network Processing



RDMA requires a memory copy to send a file!

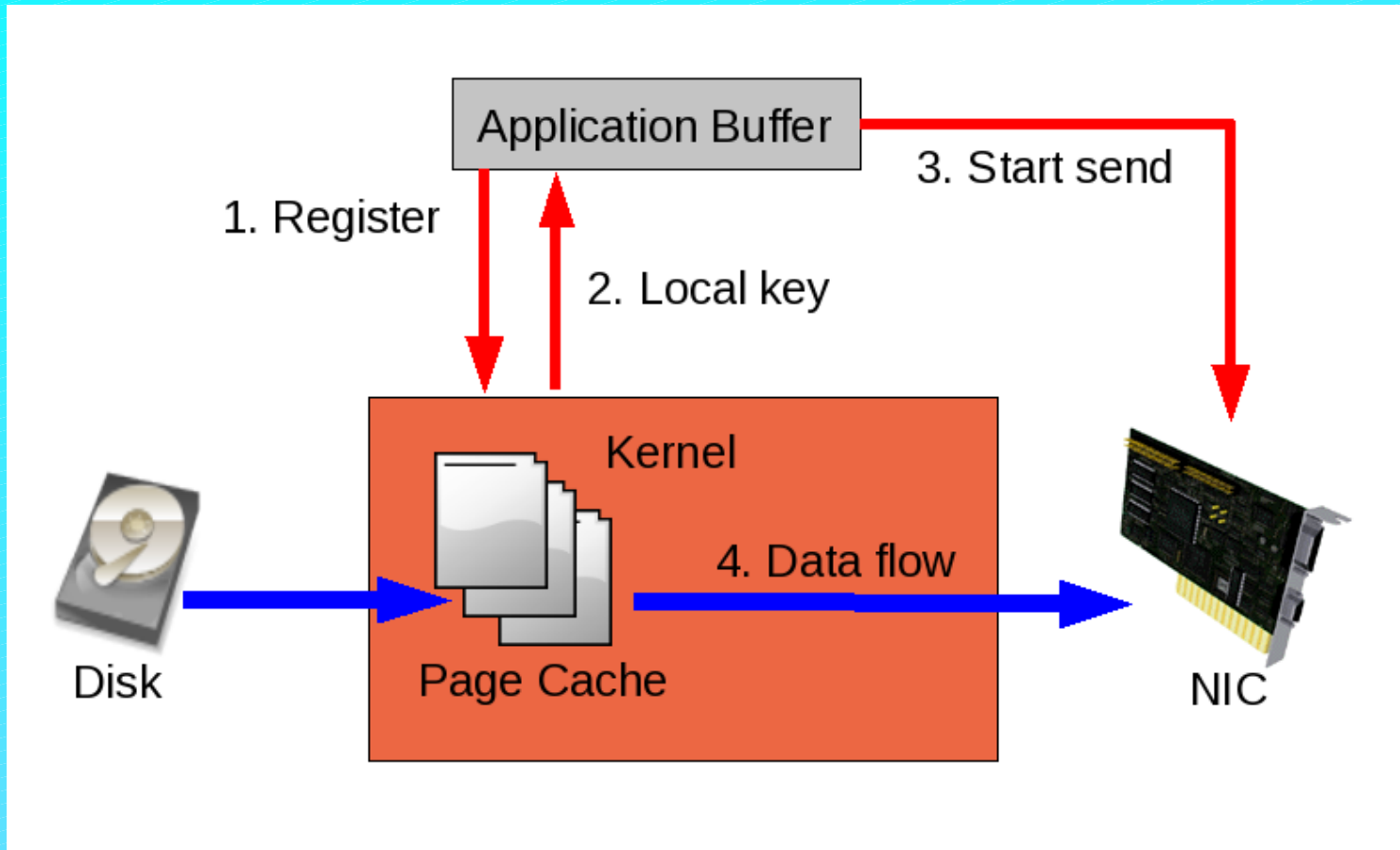
RDMA Sendfile Processing



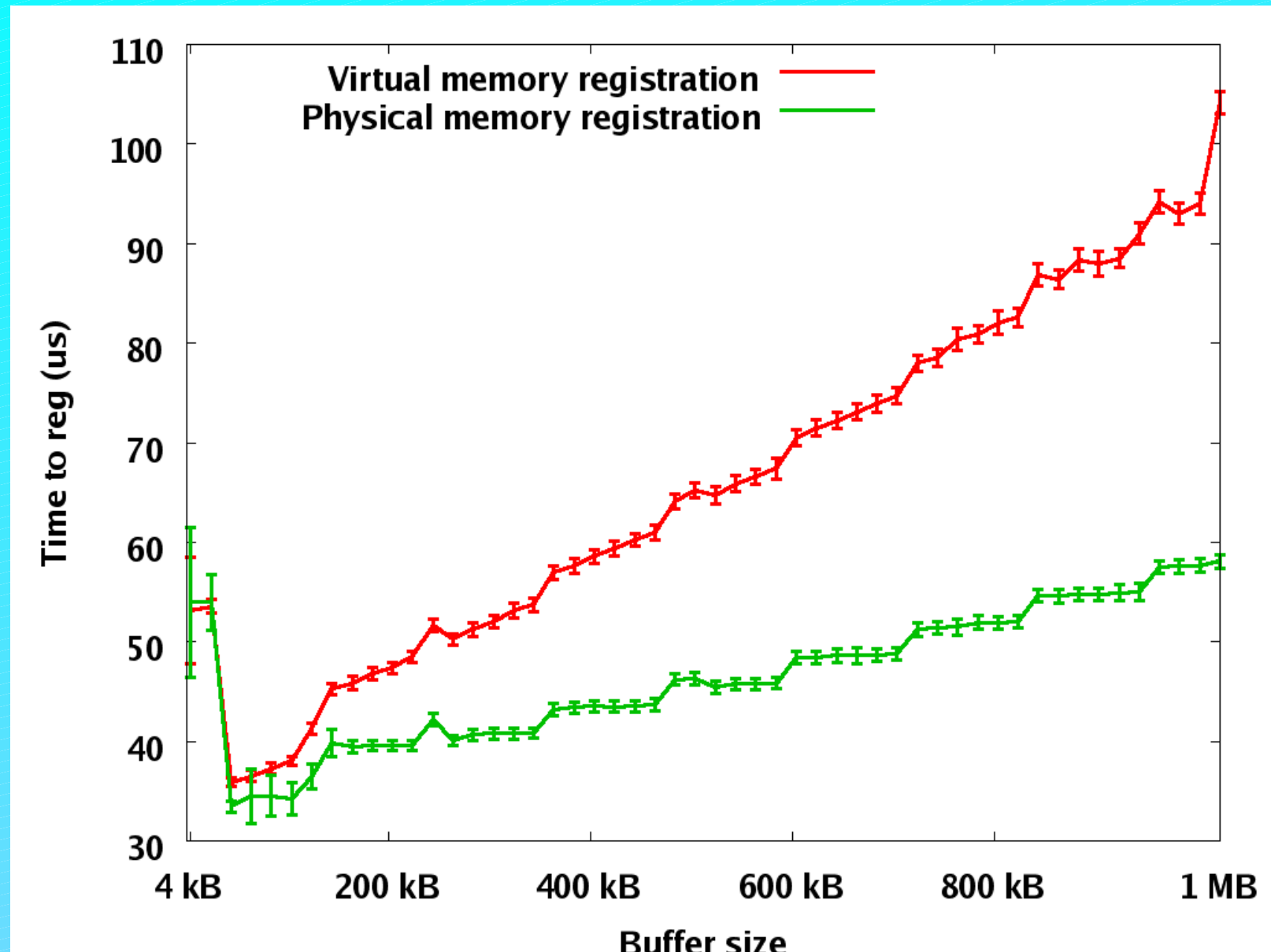
RDMA Sendfile Design

- User space application
 - Handles all connection management and errors
 - Sends and receives all data (including files)
 - Invokes kernel to map file data
 - Issues work requests to NIC using lkey from kernel
 - Never sees or copies file data
- Kernel module
 - Usual open/close/command on character device
 - Map takes PD, fd, offset, len; returns lkey
 - Unmap similar
 - Cleanup code to unmap on application failure
 - Uses physical address directly, no translation

Sendfile Steps



Key Observation



Virtual to physical translation is slow.

RDMA Sendfile Design Alternatives

- Pass QP to kernel and let it send
 - Architecturally not permitted in RDMA
 - Handling completion and errors would be complex
- Open new connection from kernel
 - Then just send directly and close
 - Lots of complex code to stick in the kernel
 - Overhead for each new connection
- Open entire session in kernel
 - Send all other data through kernel, as well as files
 - So much for OS bypass
- Just use pipelining and overlap the registration

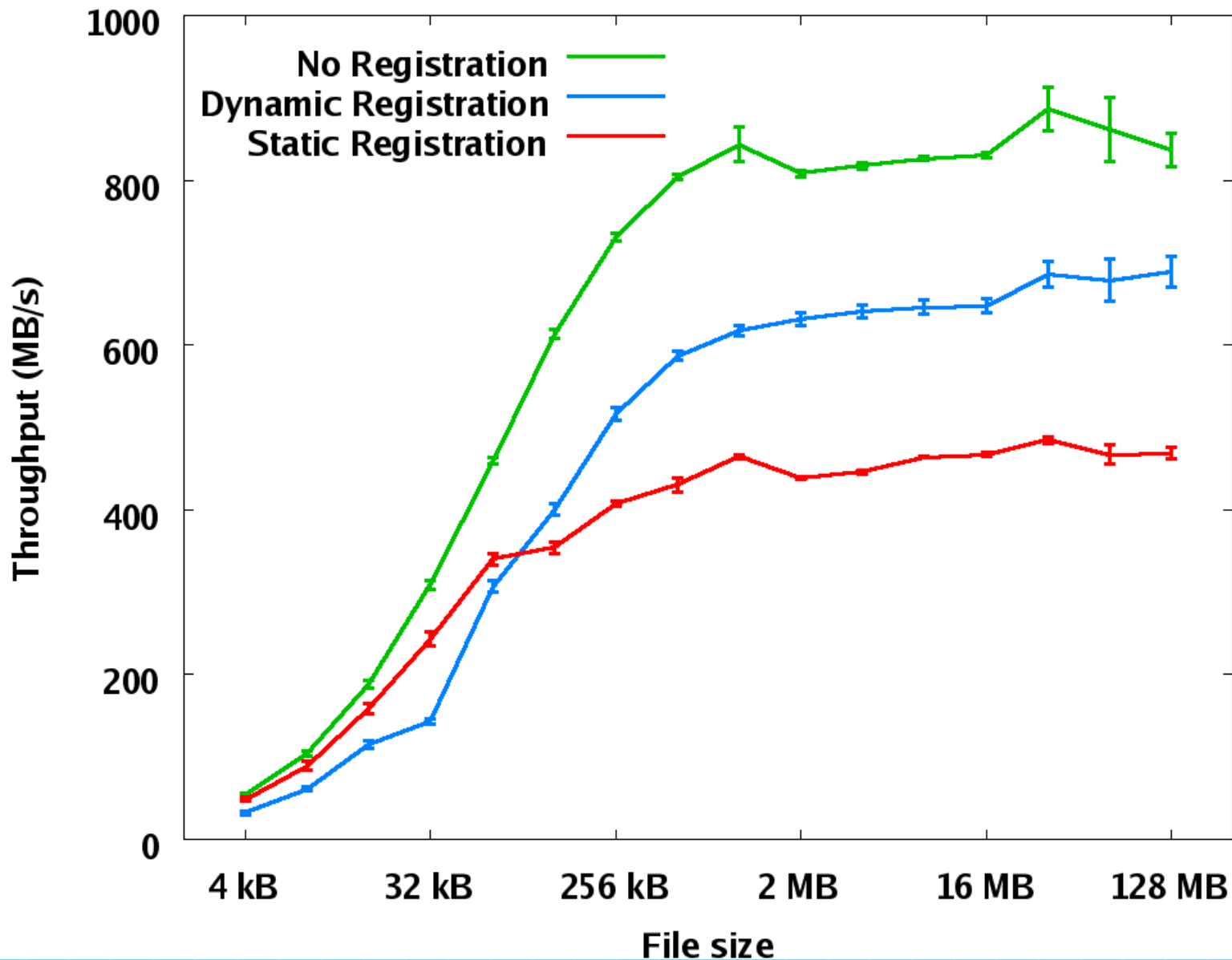
Experimental Results

- Library of alternative implementations
 - Traditional, with or without pipelining
 - RDMA Sendfile, with or without pipelining
- Single test application
- Two Opteron 250 hosts
 - 2.4 GHz
 - 2 GB memory
 - Linux 2.6.20
- Mellanox mem-free 4x SDR InfiniBand
- Mellanox-based 24-port switch

Traditional Approach

- No registration
 - Memory registration overheads out of timing loop
 - As seen in most network benchmarks
- Static registration
 - Fixed buffer registered at startup
 - Files are read directly into that buffer
 - No registration overhead, but memory copy
- Dynamic registration
 - Files are mmap-ed into application address space
 - New address range is registered
 - No memory copy, but virtual-to-physical translation
 - (Current kernels have bug requiring copy, not shown here)

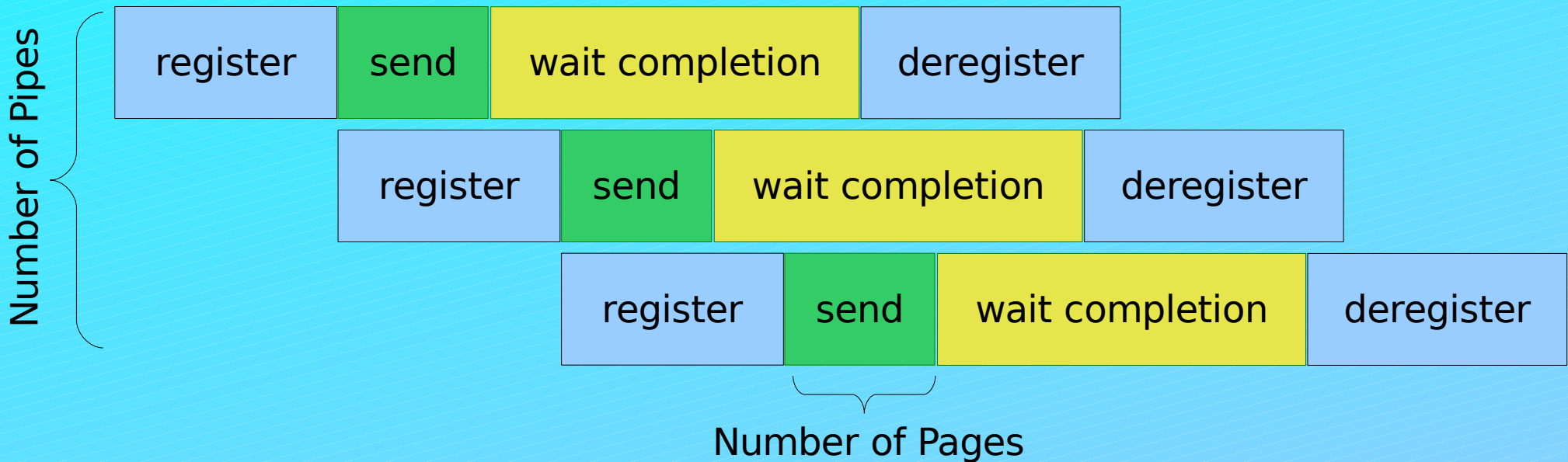
Traditional Approach



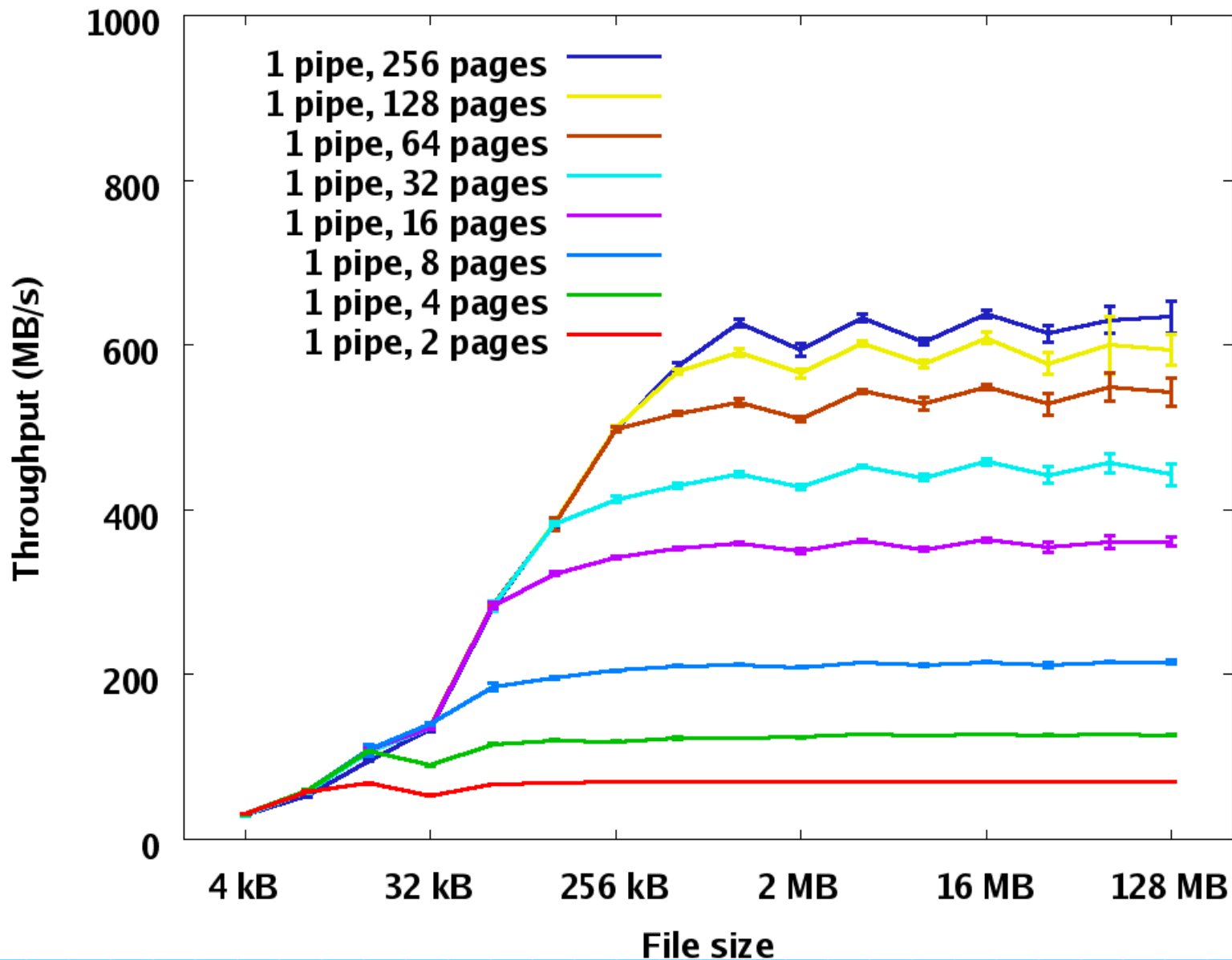
Static initially faster, until out of processor cache.
Dynamic registration overhead clearly visible.

Pipelining Effect

- Split up registration and sending
- Break each file size into sets of *pages*
- Keep some number of *pipes* outstanding

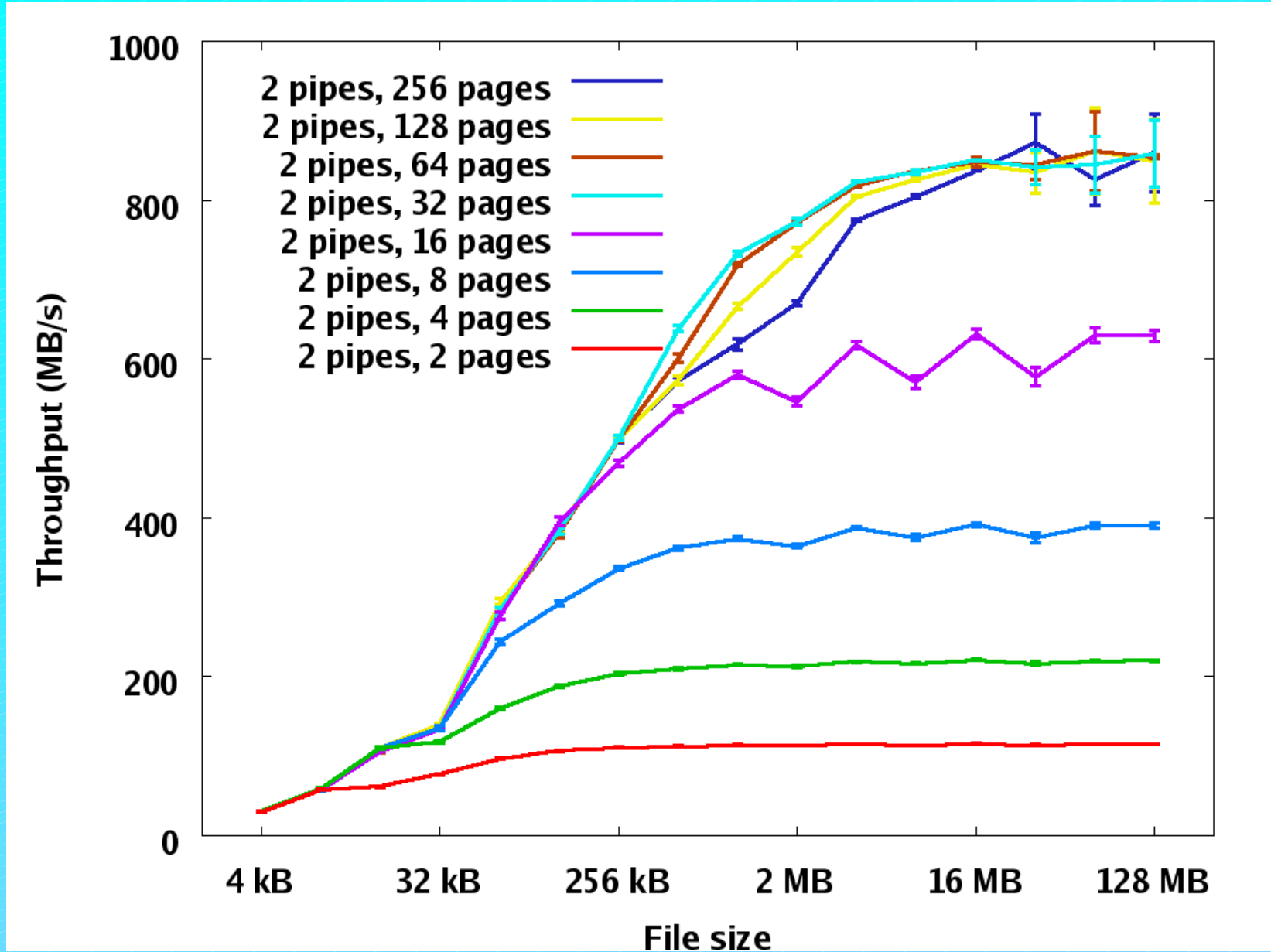


Traditional Pipelining: 1 pipe



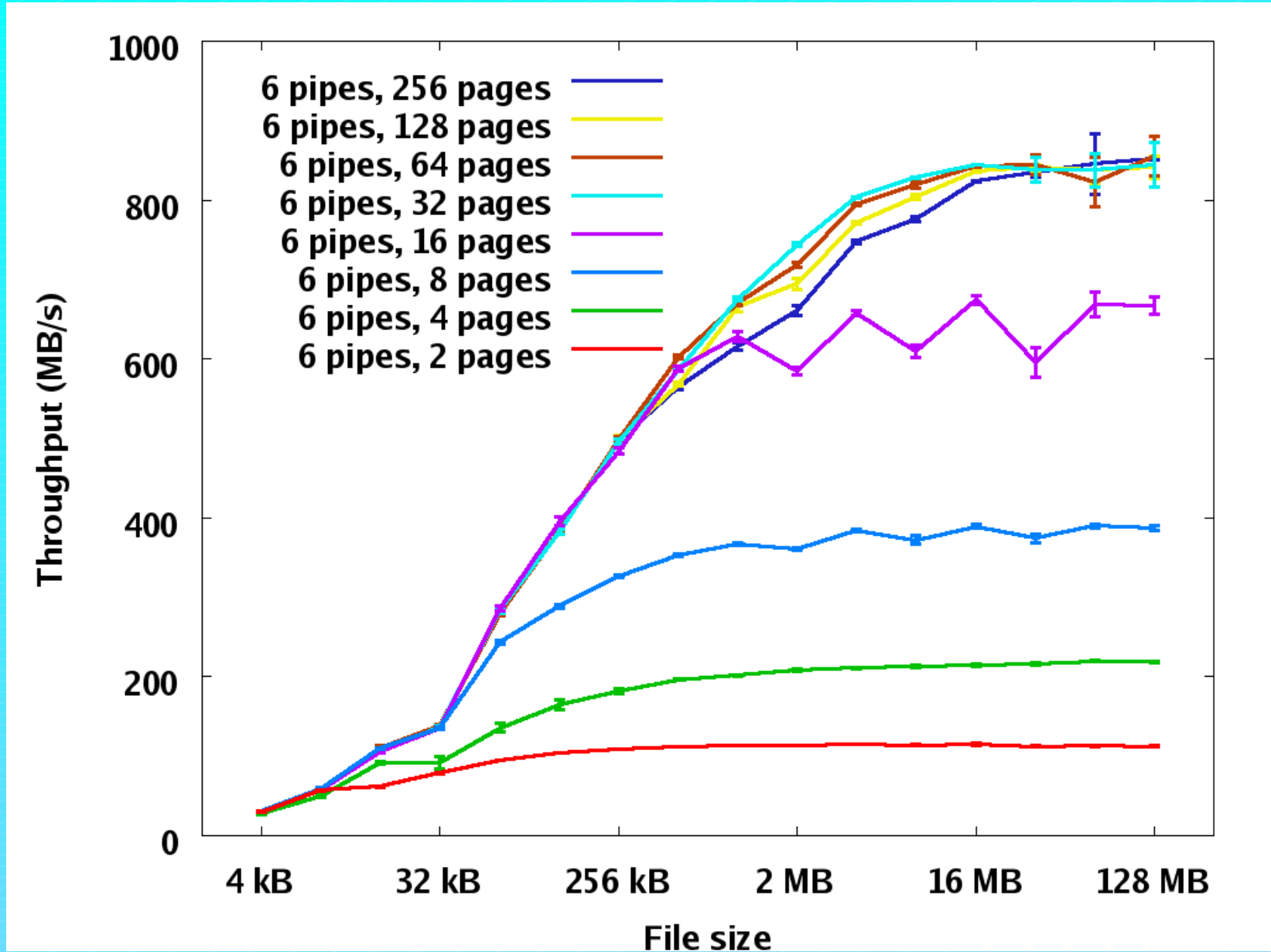
Need large operations to get good throughput.

Traditional Pipelining: 2 pipes



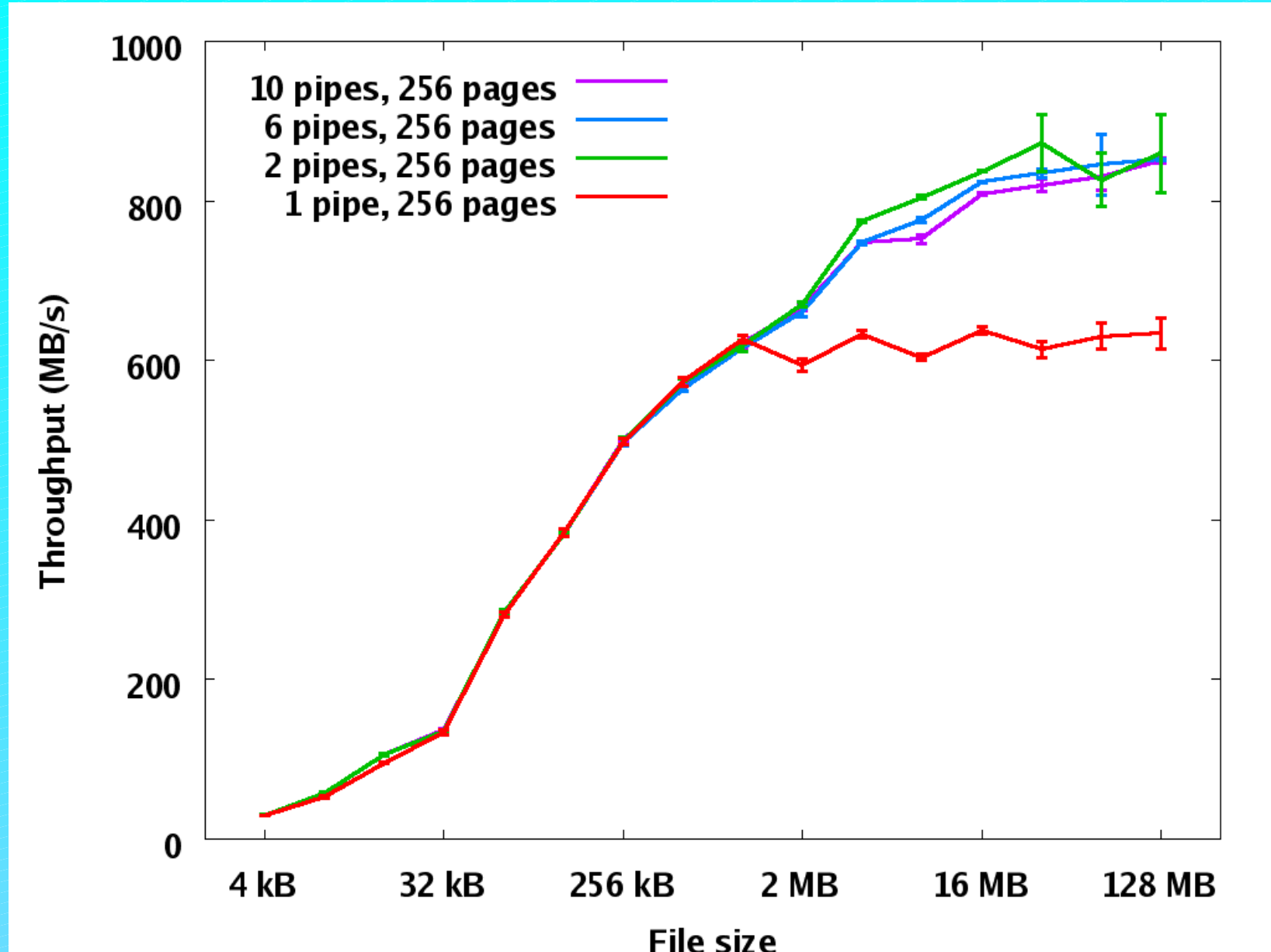
Just two pipes significantly improve throughput.

Traditional Pipelining: 6 pipes



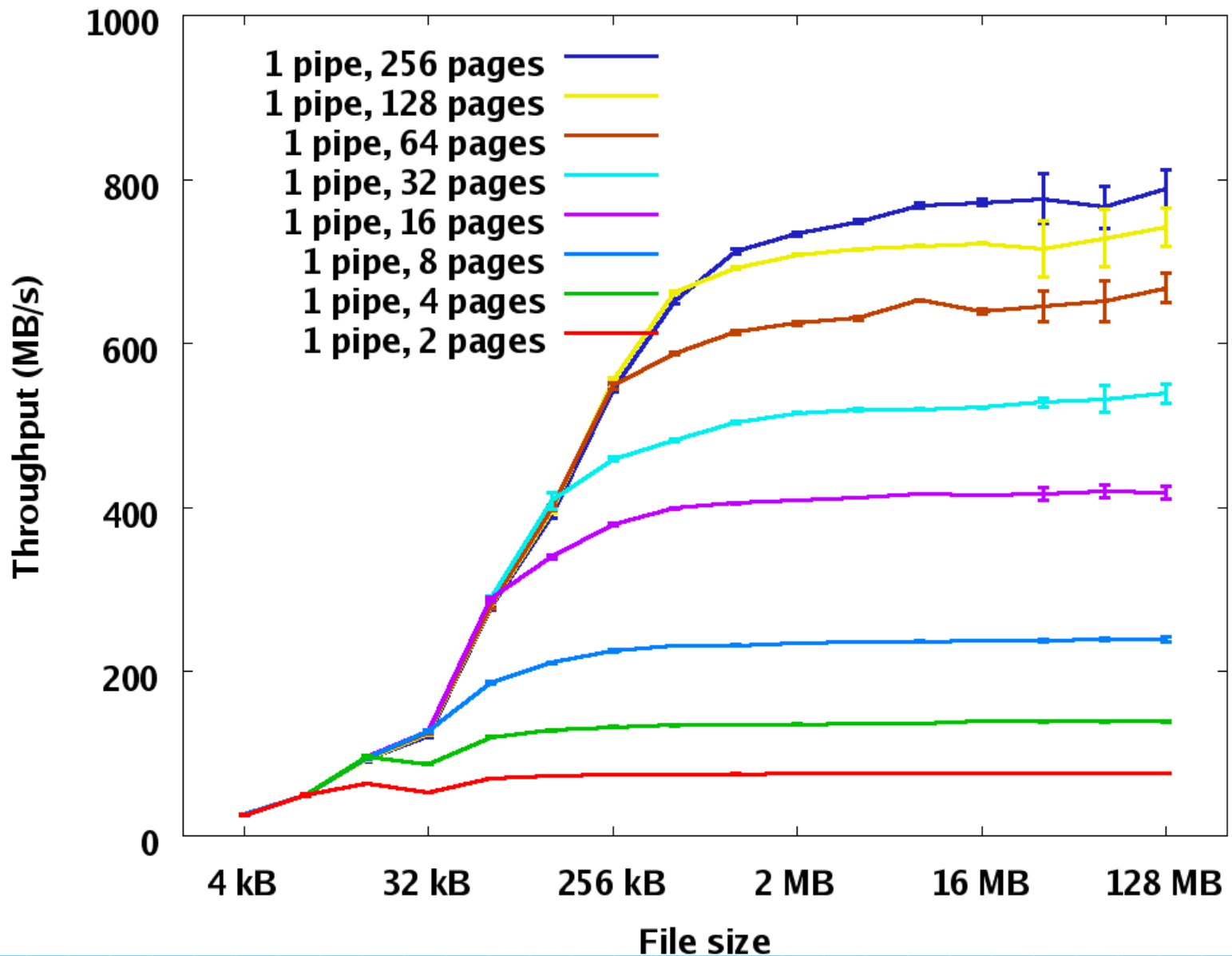
Little difference beyond two pipes.

Traditional Page count: 256 pages



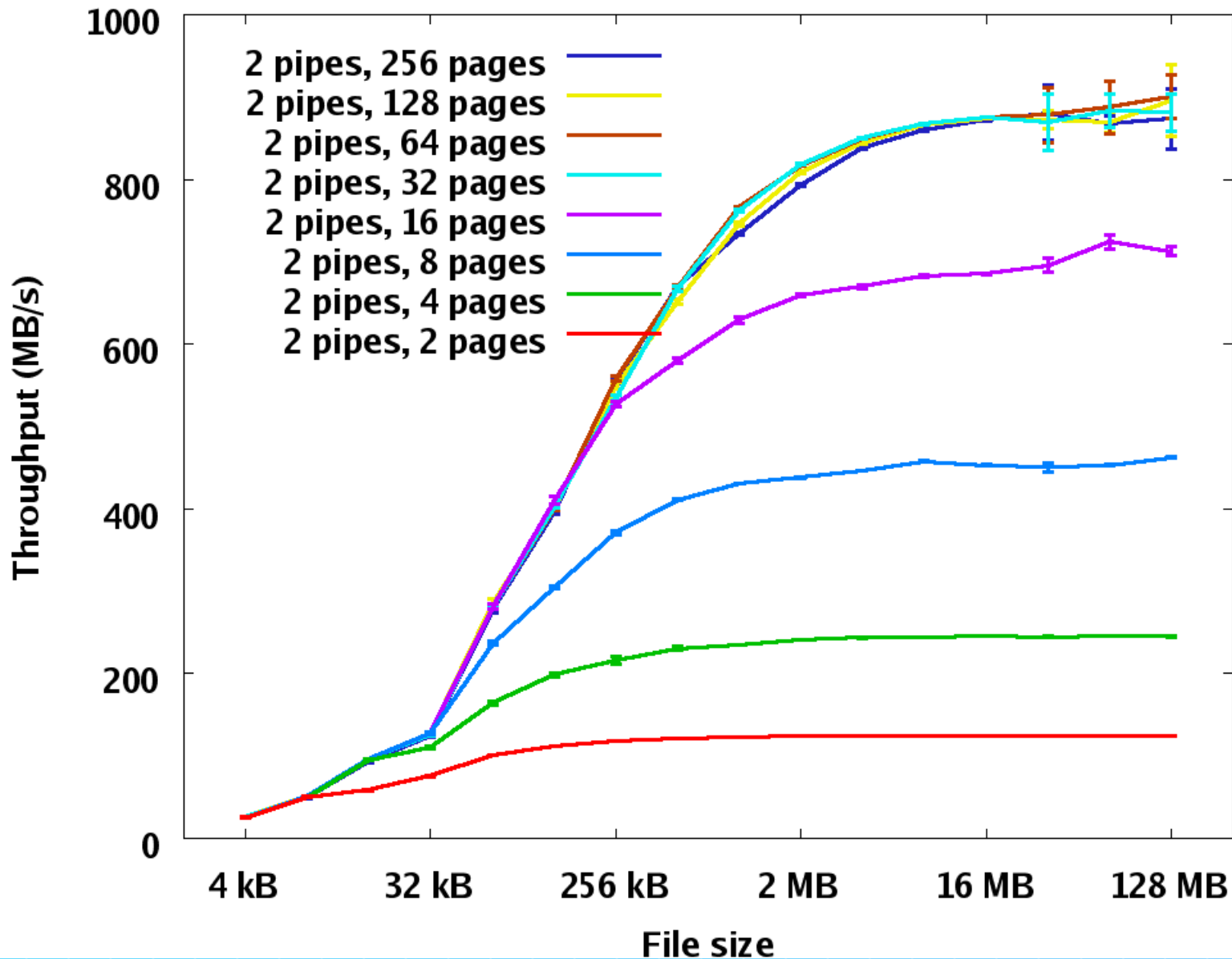
Transpose of previous few graphs.

Sendfile Pipelining: 1 pipe



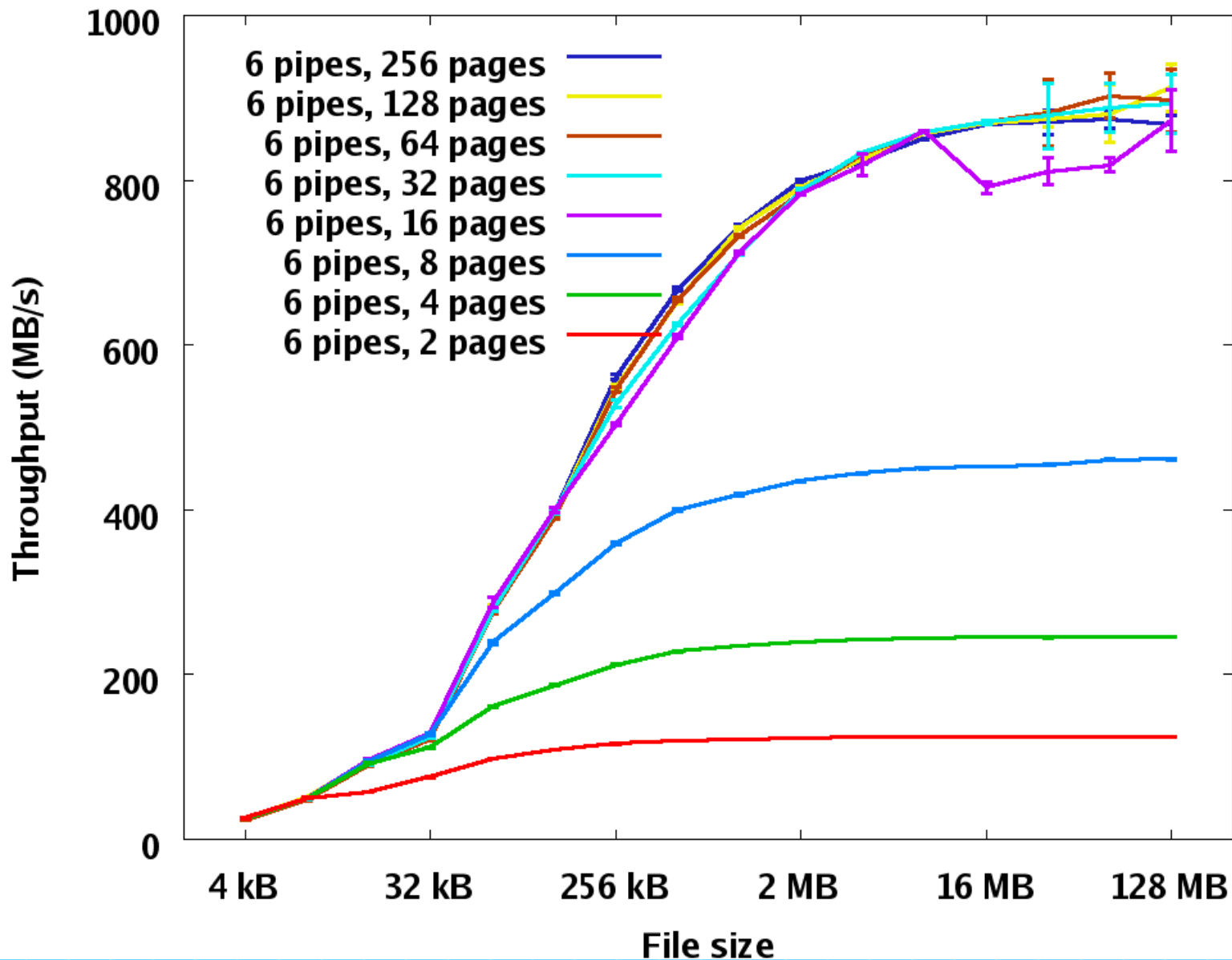
Need large operations to get good throughput.

Sendfile Pipelining: 2 pipes



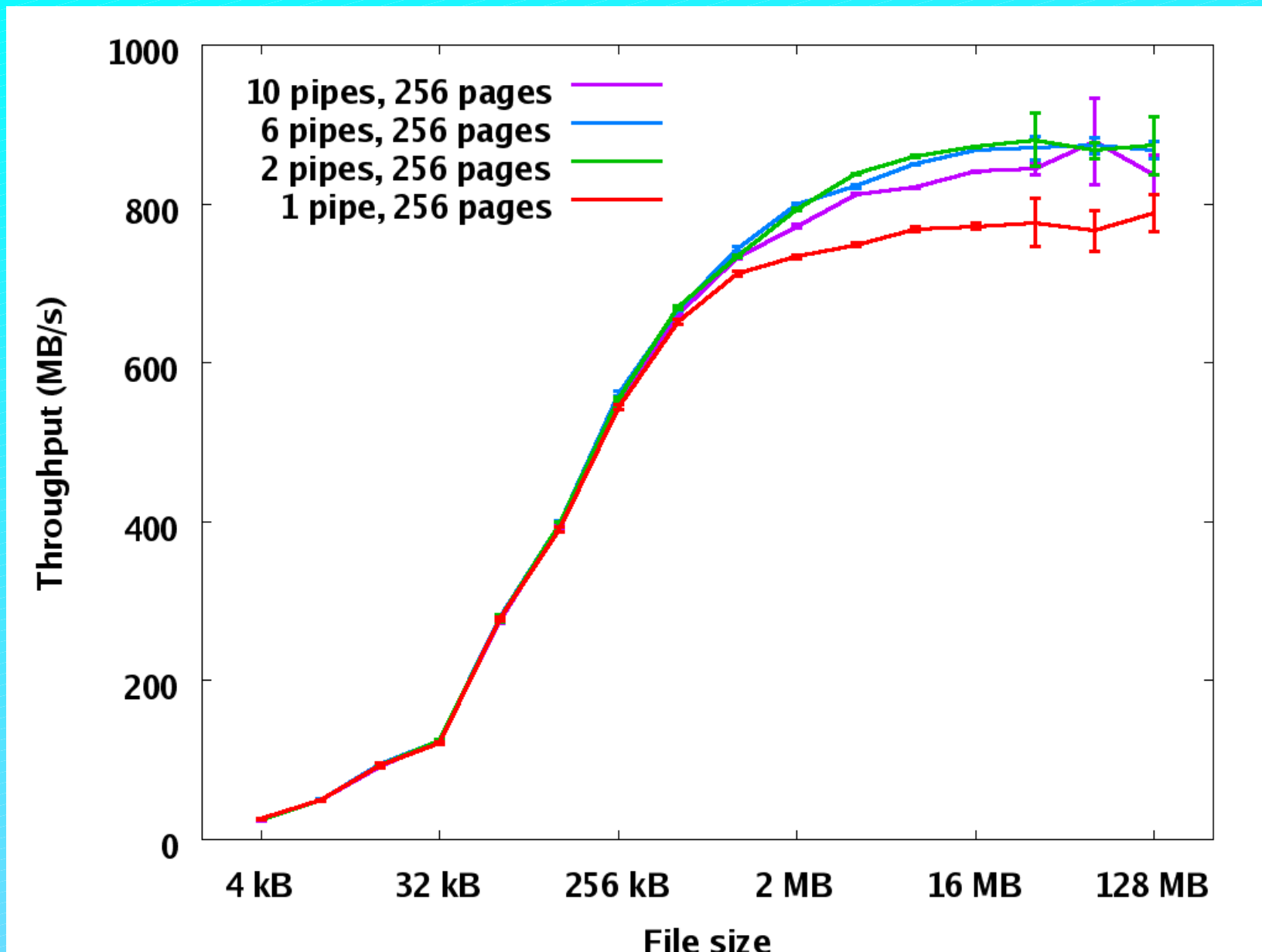
Just two pipes significantly improve throughput.

Sendfile Pipelining: 6 pipes



Little difference beyond two pipes.

Sendfile Page count: 256 pages

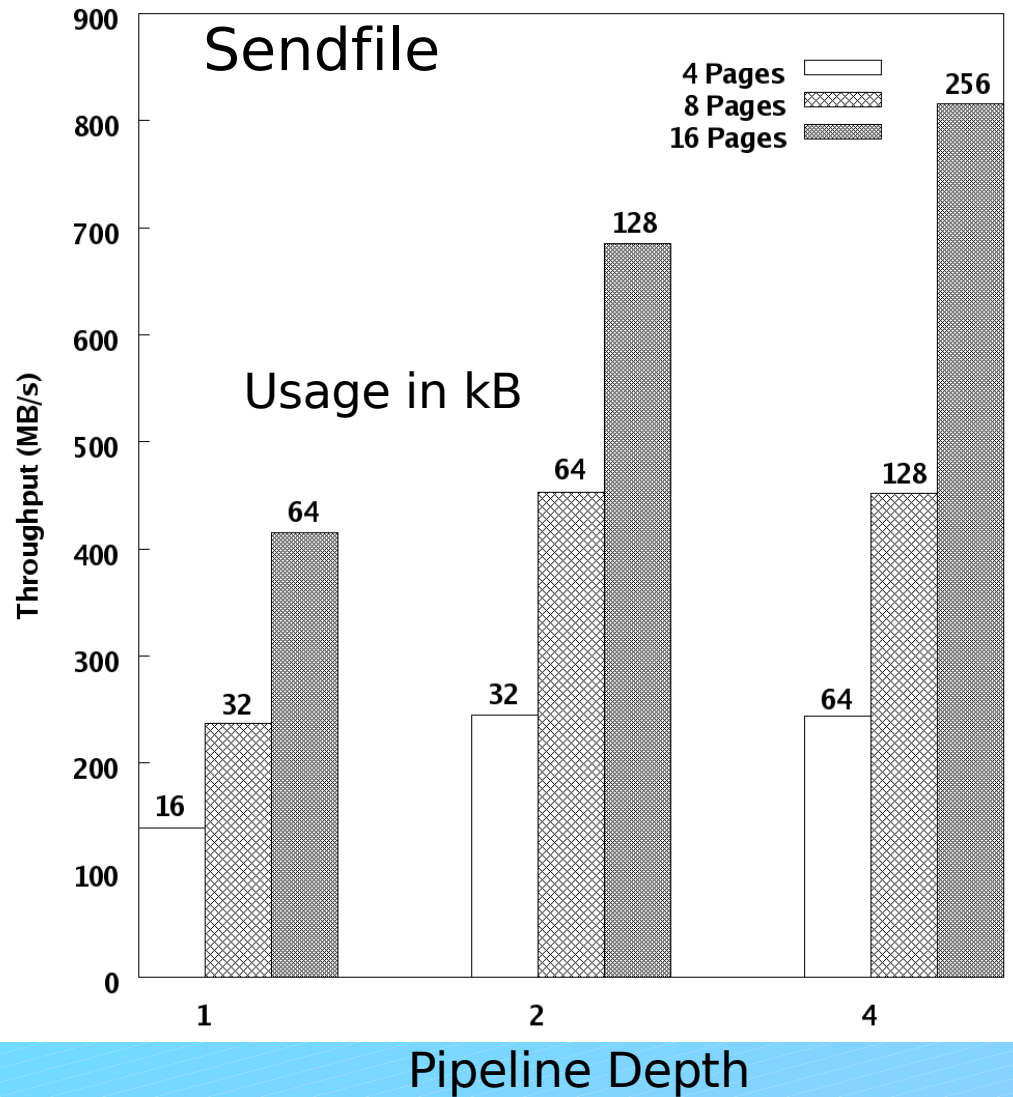
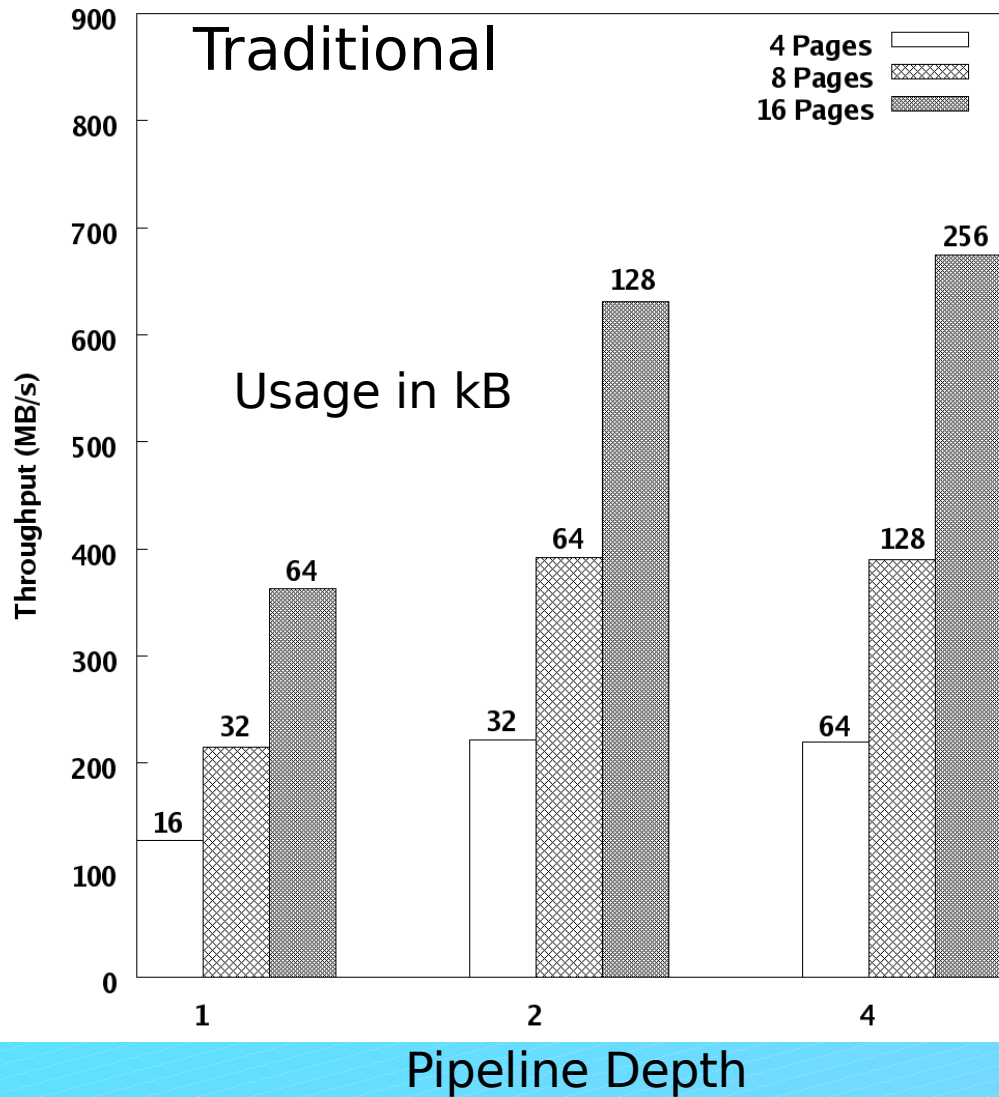


Transpose of previous few graphs.

Per-connection Memory Requirements

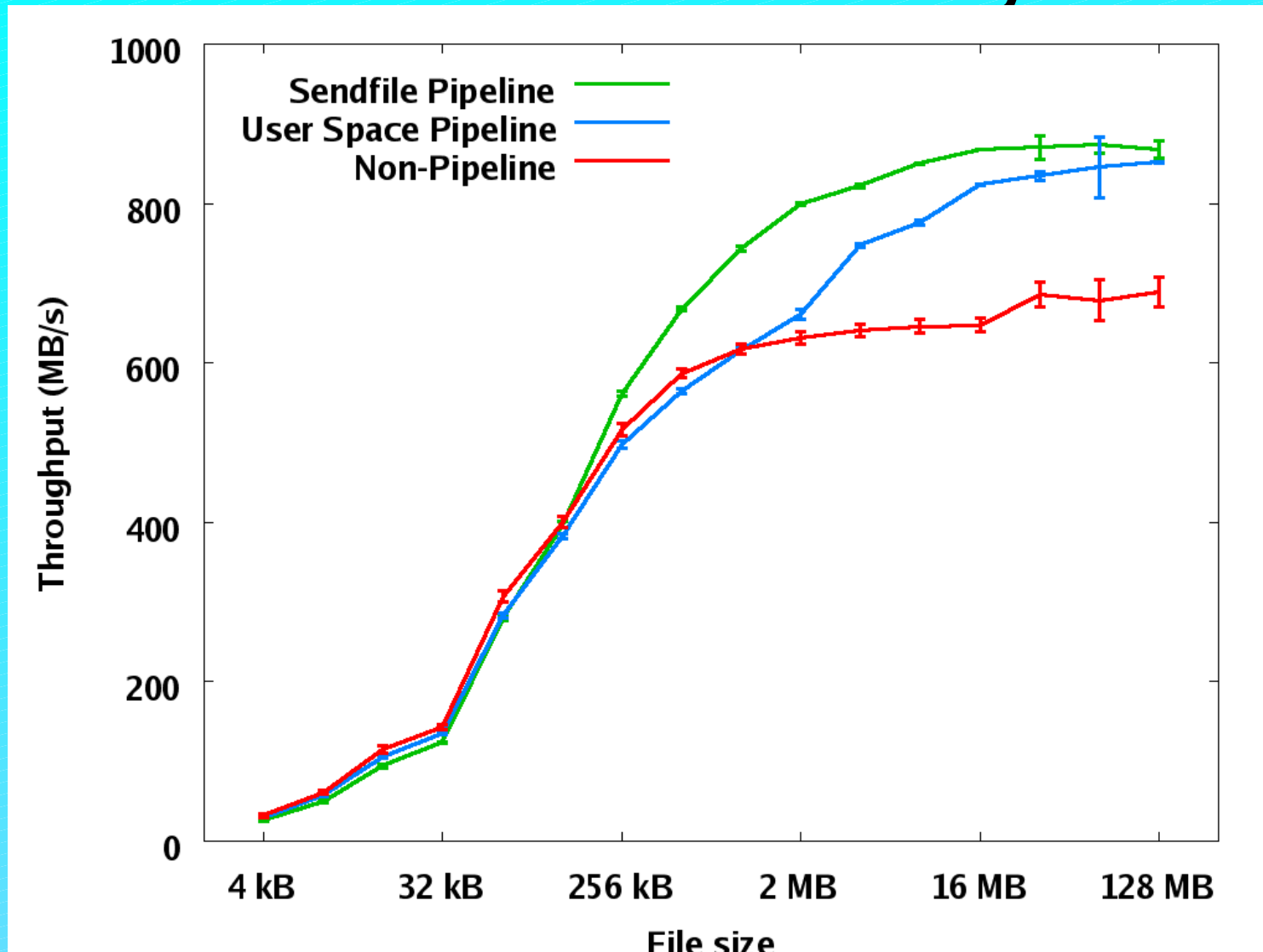
- Real TCP-based servers
 - Many connections
 - Small state per connection
 - Send socket buffer size offers tunable memory
- RDMA approach
 - Both static and dynamic require much more
 - Small message sizes deliver little performance
- Sendfile as a way to limit resource consumption

Memory Usage



RDMA performance needs memory.
Sendfile is better for small (and large) buffers).

Results Summary



All dynamic registration.

Pipelining: 6 pipes, 256 pages each (1 MB).

At higher network speeds, gap should enlarge.

Less CPU utilization and memory overhead with sendfile.

Promising Related Work

- Registered memory allocator (Tipparaju)
 - User-space prototype
 - Could be part of kernel
 - Path to integrate with other memory functions
- Hugepages (Rex)
 - Minimizes virtual-to-physical translation
- Other in-kernel file sending (Park)
 - Even less integrated with existing OS

Conclusions

- Cautionary Tale
 - Is this really a good idea?
 - If you want to do this, our way performs best.
 - App cache, disable page cache, avoid OS bypass
- Sendfile and splice
 - Exist to work around problems in sockets API
 - RDMA has different problems
- Must consider integrated system
 - Networking
 - Memory management
 - File system