A Performance Study of Sequential I/O on Windows NT™ 4.0

How to get the most from your I/O system

Erik Riedel

Carnegie Mellon University

www.cs.cmu.edu/~riedel

Catharine van Ingen, Jim Gray

Microsoft Research - Bay Area Research Center

www.research.microsoft.com/barc

Outline

- Introduction
- Measurements of sequential I/O
  - » single disk
  - » parallelism - multiple requests, disks, busses
  - » some details
  - » some pitfalls
- Summary

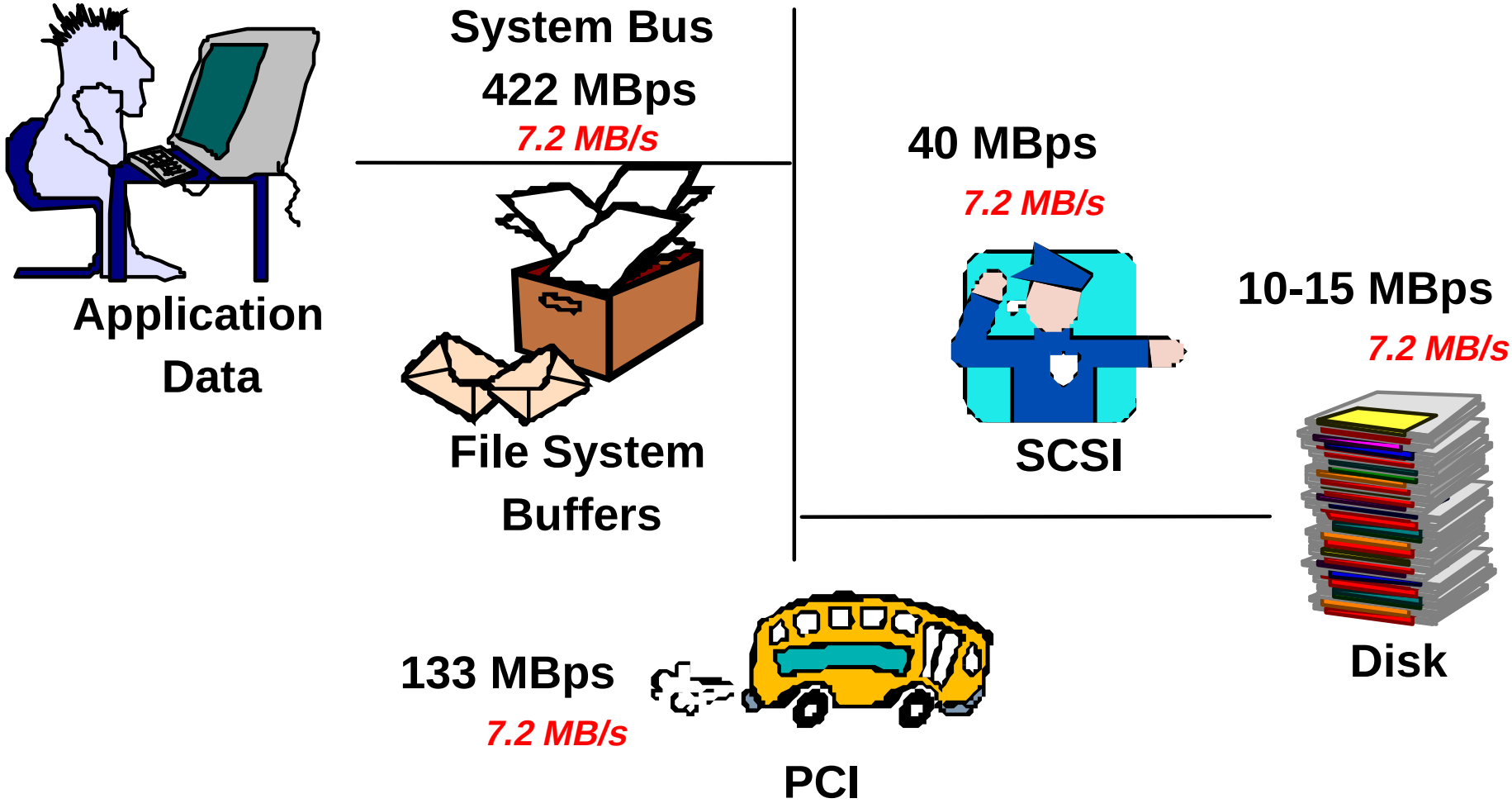
Motivation

- Multimedia
- Data mining
- EOS/DIS metrics
  - » MOX (megabyte objects per second)
  - » SCANS (# of scans of the entire data per day)
- Commodity servers and clusters
- Bandwidth is key
  - » where are the bottlenecks?

PAP (peak advertised performance)

RAP (real application performance)

- Goal: RAP = PAP / 2 (the *half-power* point)



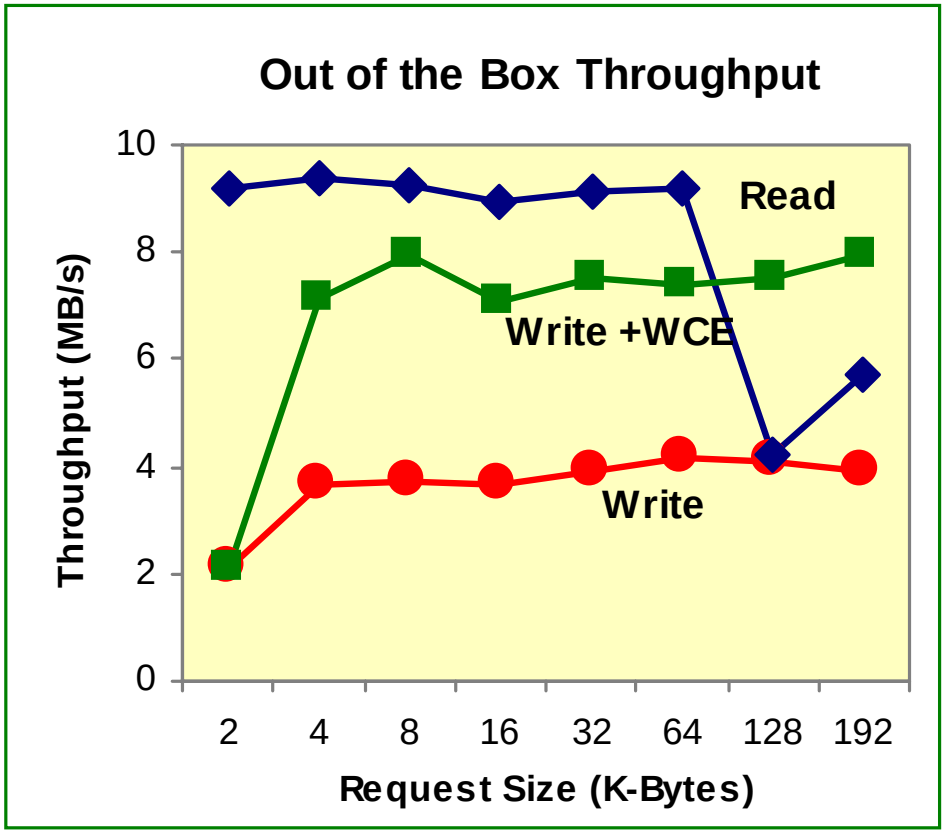
Experimental Setup

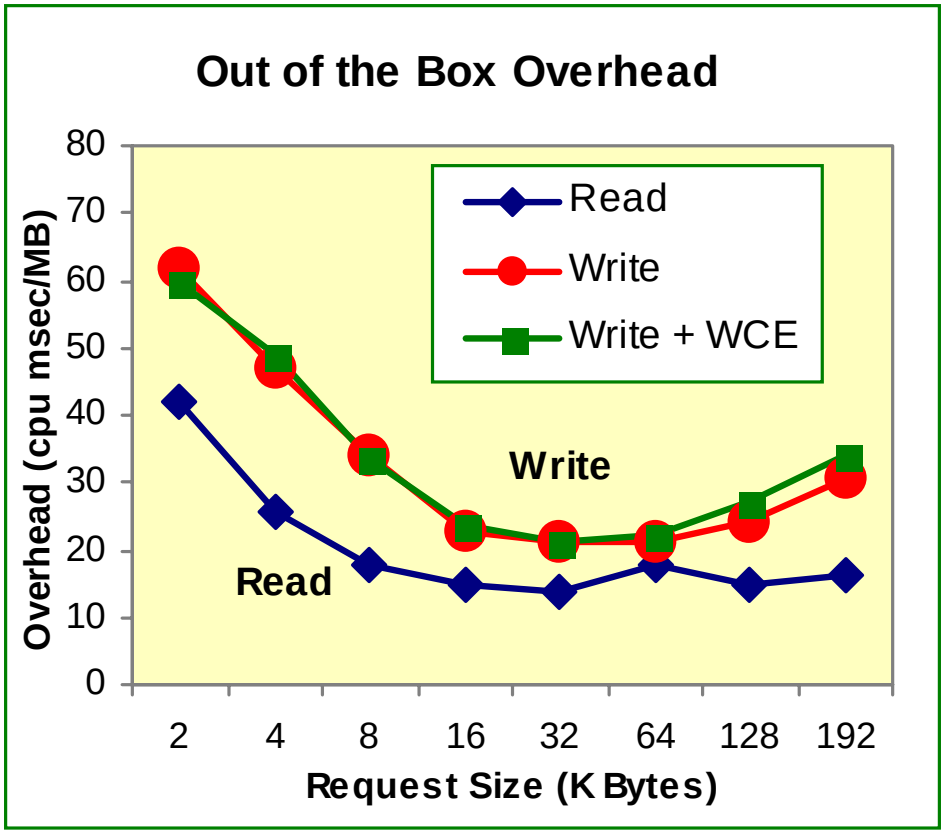
- Gateway G6-200, 200 MHz Pentium Pro
- 64 MB DRAM
- 32-bit PCI
- Adaptec 2940 Fast-Wide (20 MBps)
  and Ultra-Wide (40 MBps) controllers
- Seagate 4GB Barracuda SCSI disks (Fast and Ultra)
  - » (7200 rpm, 7-15 MBps “internal”)
- NT Server 4.0, 1381 SP3, NTFS
- i.e. modest 1997 technology

Out of the Box Performance

- Read throughput is good
- Write is 40% of read
- WCE is fast but dangerous

- 20 ms/MB ~ 2 instr/byte
- CPU will saturate at 50MBps





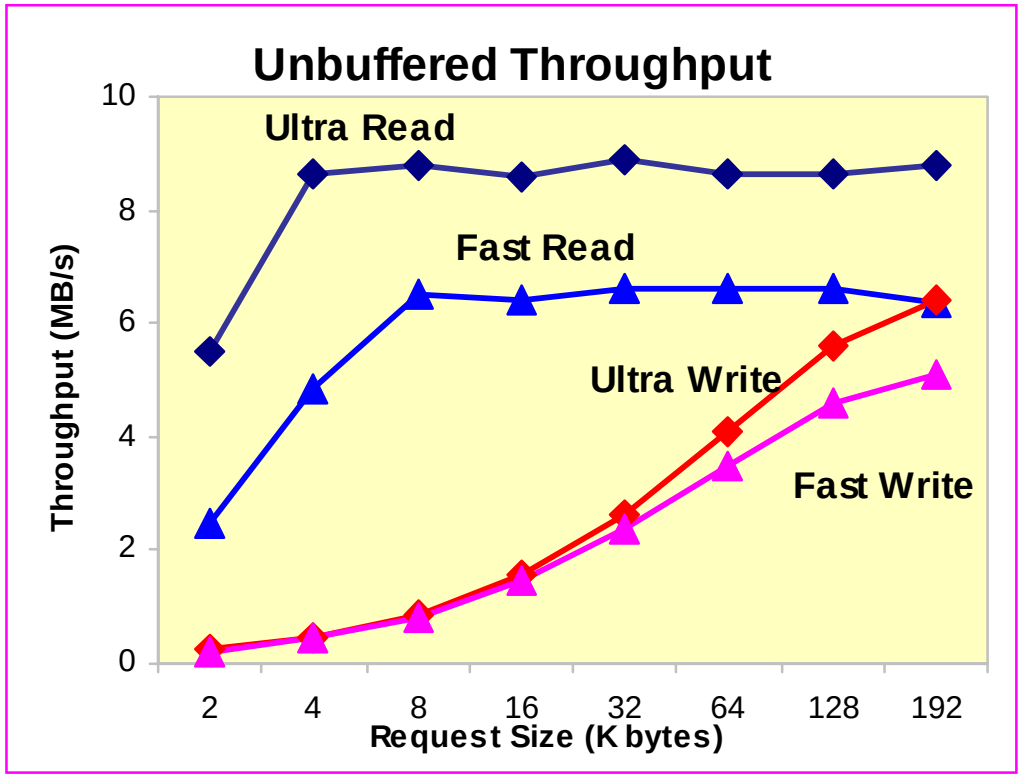
Things To Think About

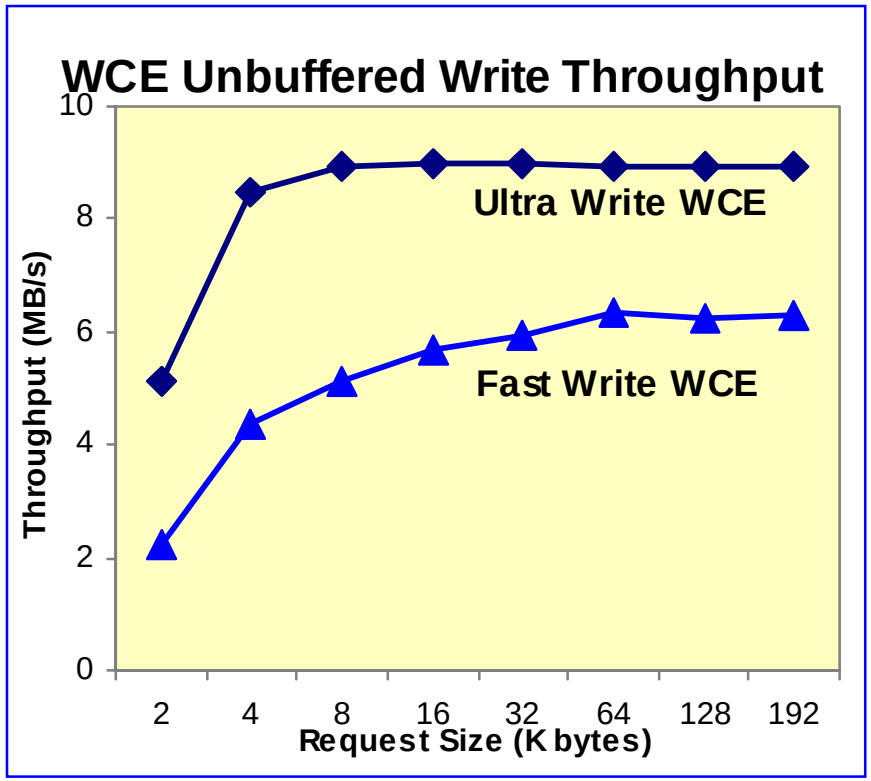
- Read-ahead
- Write-behind (WCE)
- Write coalescing
- Small requests kill
- Problems above 64K
  - » (but those have been fixed)
- Sweet spot at 64K

Raw Performance (no caching)

- Reads do well above 4K
- Writes are terrible
  - » WCE helps

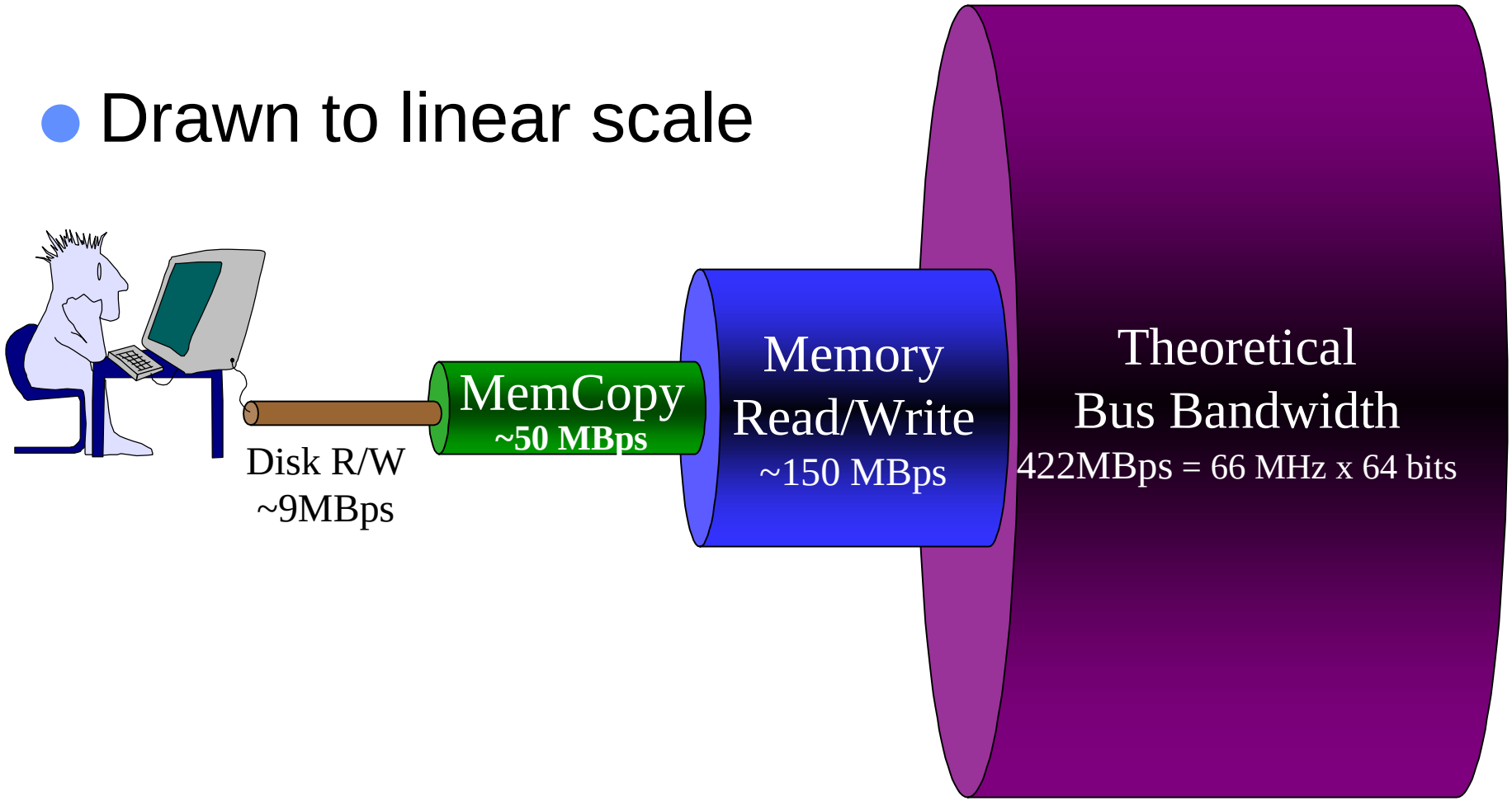
- Half-power point
  - » Read: 4 KB
  - » Write: 64 KB no wce
    4 KB with wce





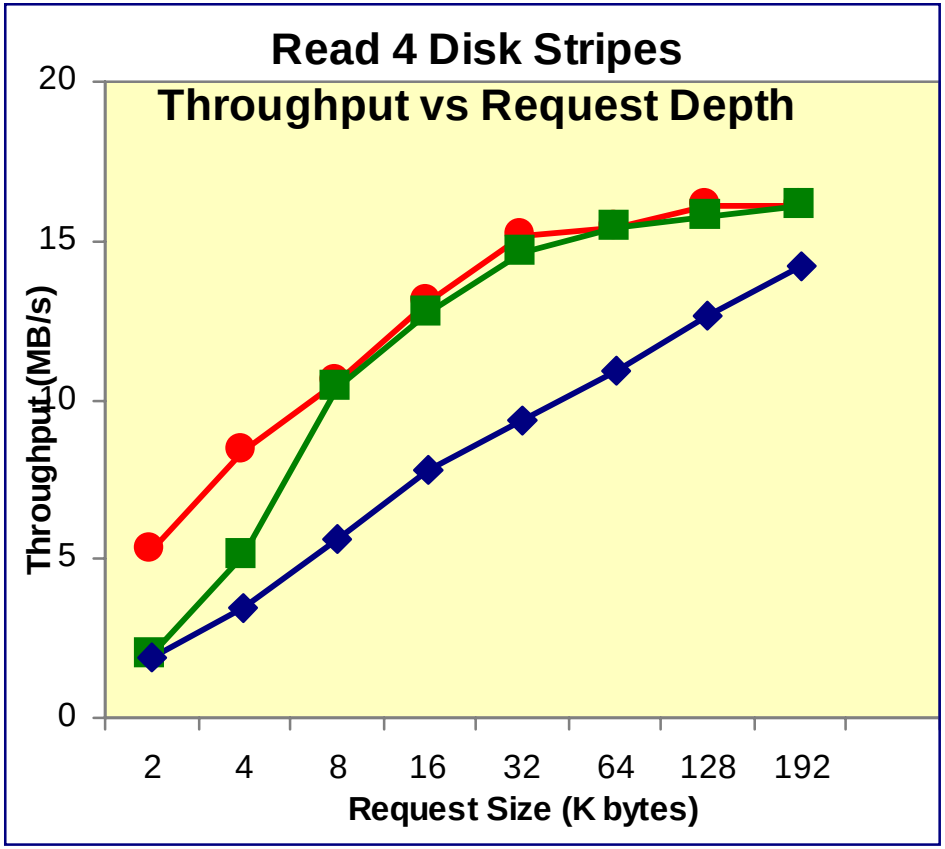
Bottlenecks

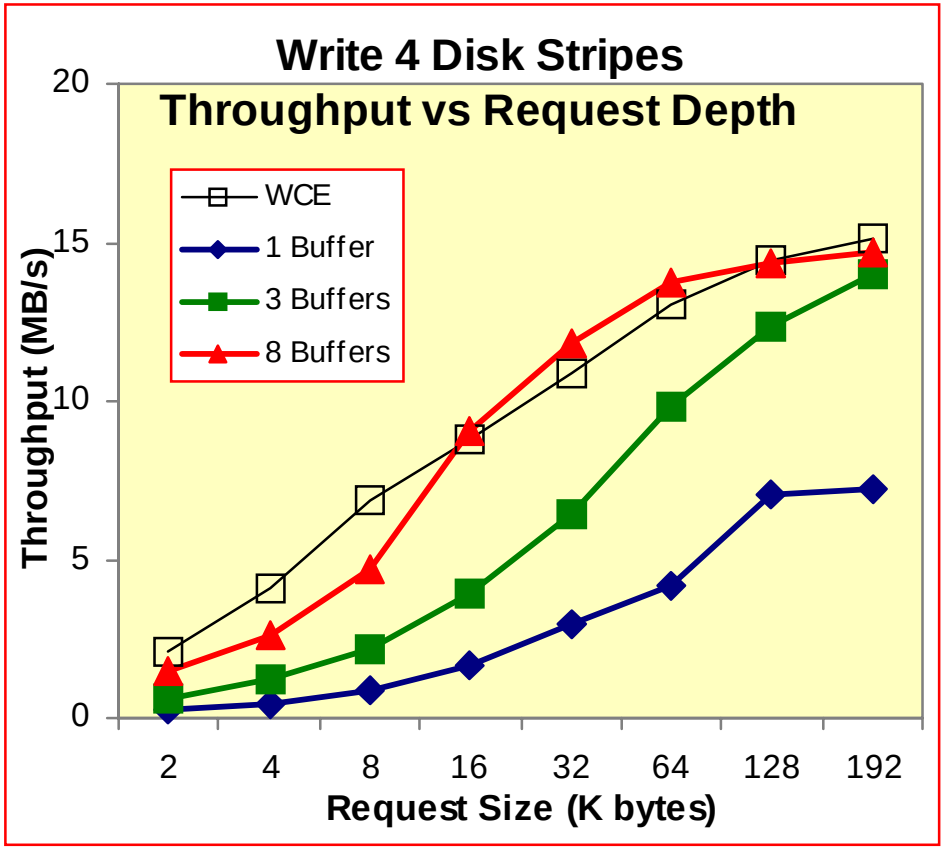
- Drawn to linear scale



Parallelism - Multiple Disks

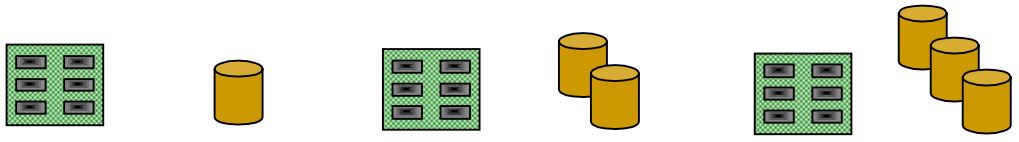
- Stripes NEED parallelism
- 3-deep is probably good enough (*triple-buffered*)
- Asynchronous requests get close to WCE



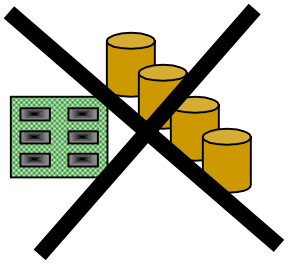


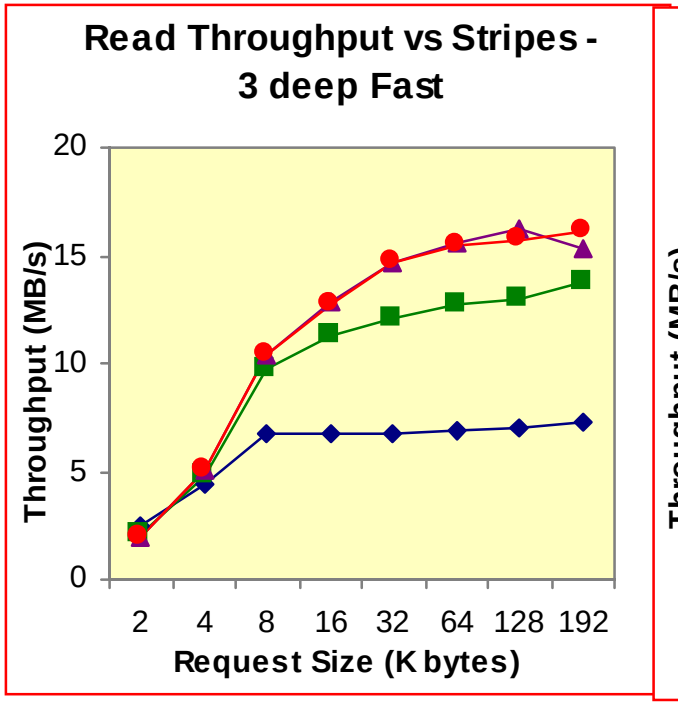
3 Stripes and You're Out!

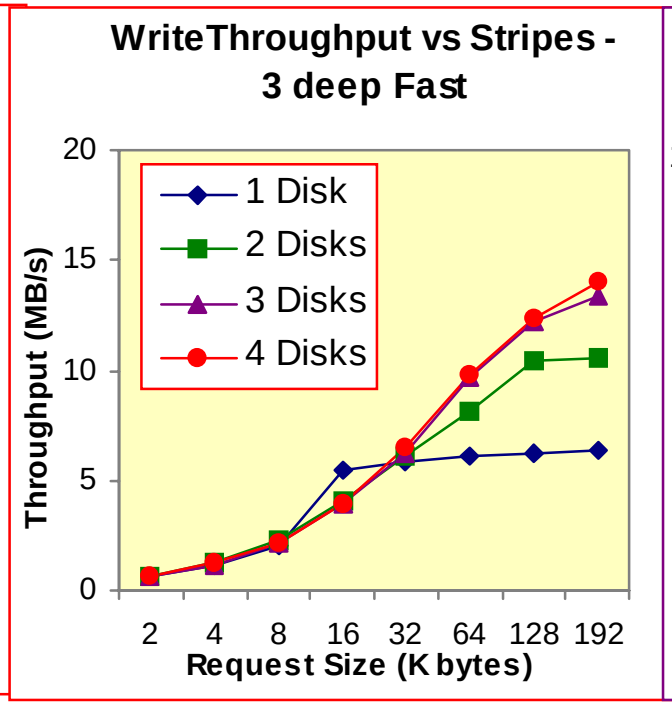
- 3 disks saturate an adapter
  - » both Fast and Ultra

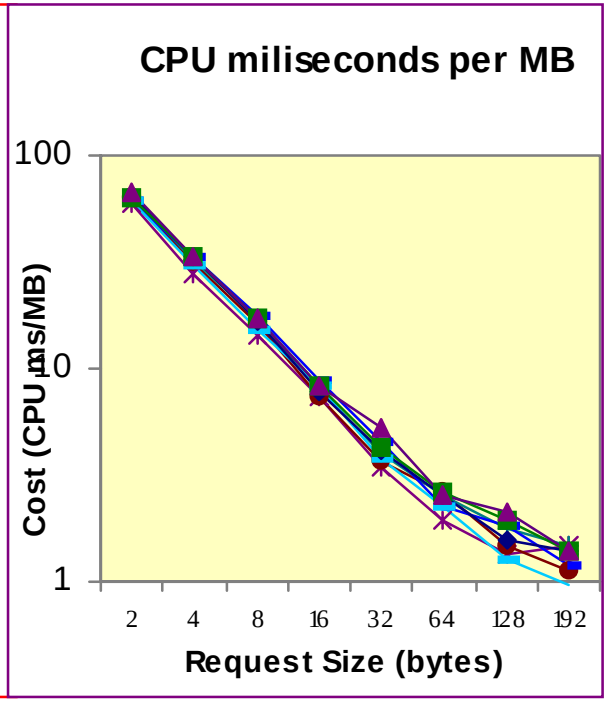


No further benefit



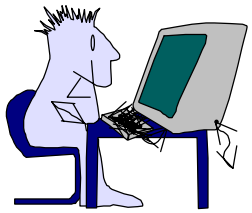


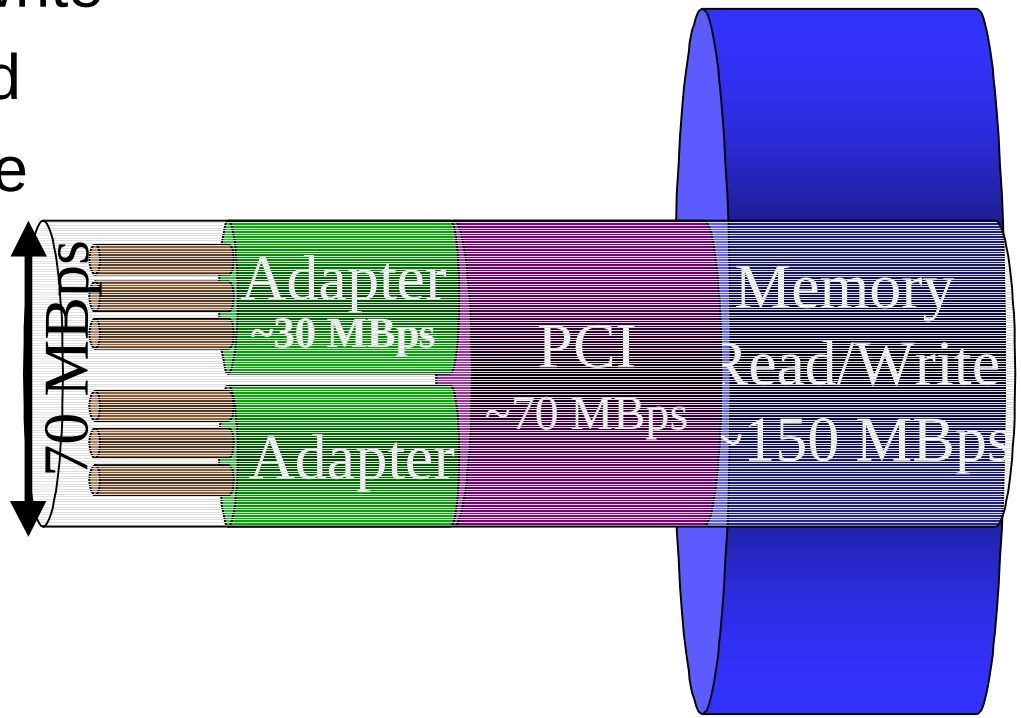




Bottlenecks

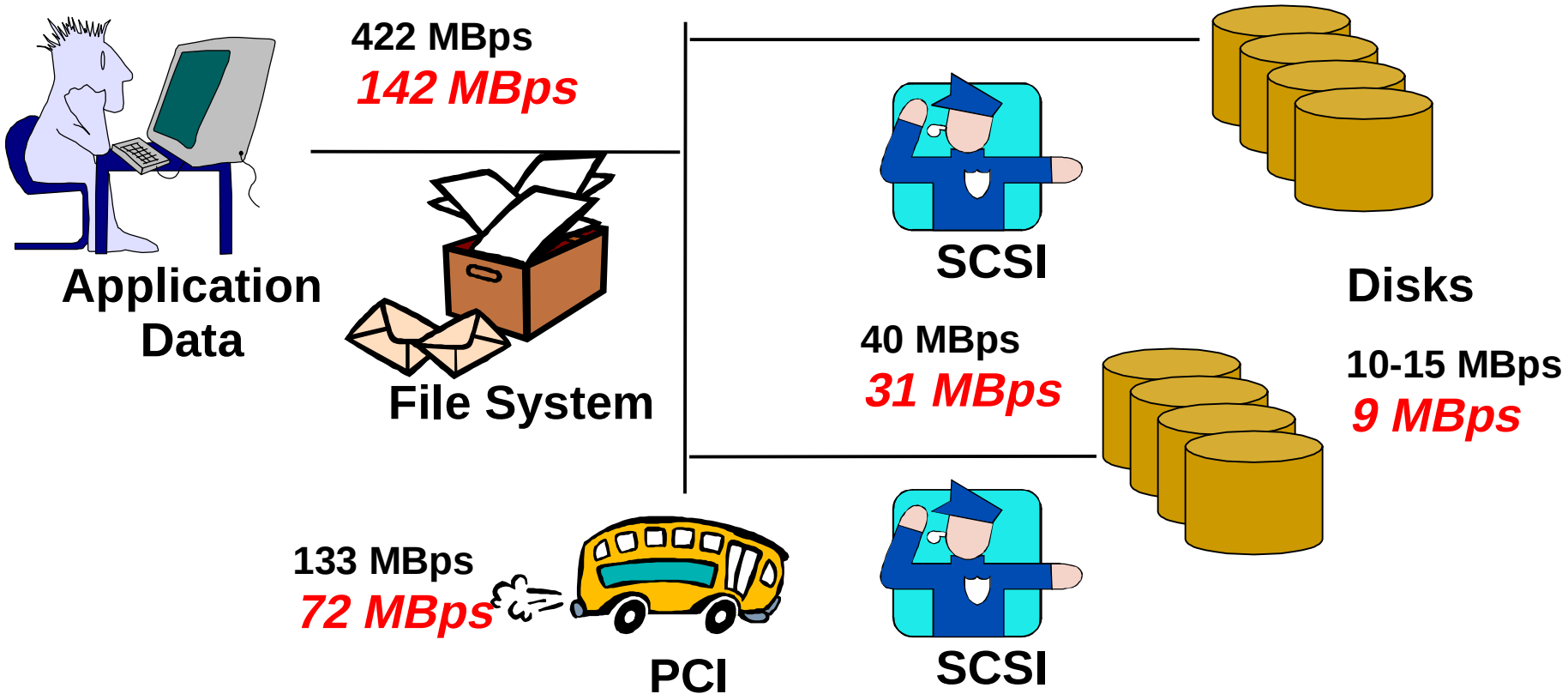
- NTFS Read/Write 9 disk, 2 SCSI bus, 1 PCI
  - ~ 65 MBps Unbuffered read
  - ~ 43 MBps Unbuffered write
  - ~ 40 MBps Buffered read
  - ~ 35 MBps Buffered write





PAP vs *RAP*

- Reads are easy, writes are hard
- Deep asynchronous write can match WCE



Summary

- **Read is easy, write is hard**
  SCSI & FS read prefetch works
  Read PAP ~ .8 RAP
  Write PAP ~ .05 RAP to .8 RAP
- **NTFS buffering good for small IOs**
  coalesces into 64KB requests
- **Bigger is better:** 8KB ok, 64KB best
- **Deep requests help**
  3-deep is good, 8-deep is better
- **WCE is fast but dangerous**
  3-deep writes approximate WCE
  for ≥ 8KB requests
- **3 disks saturate a SCSI bus**
  Fast-Wide (15 MBps) and Ultra-Wide (31 MBps)

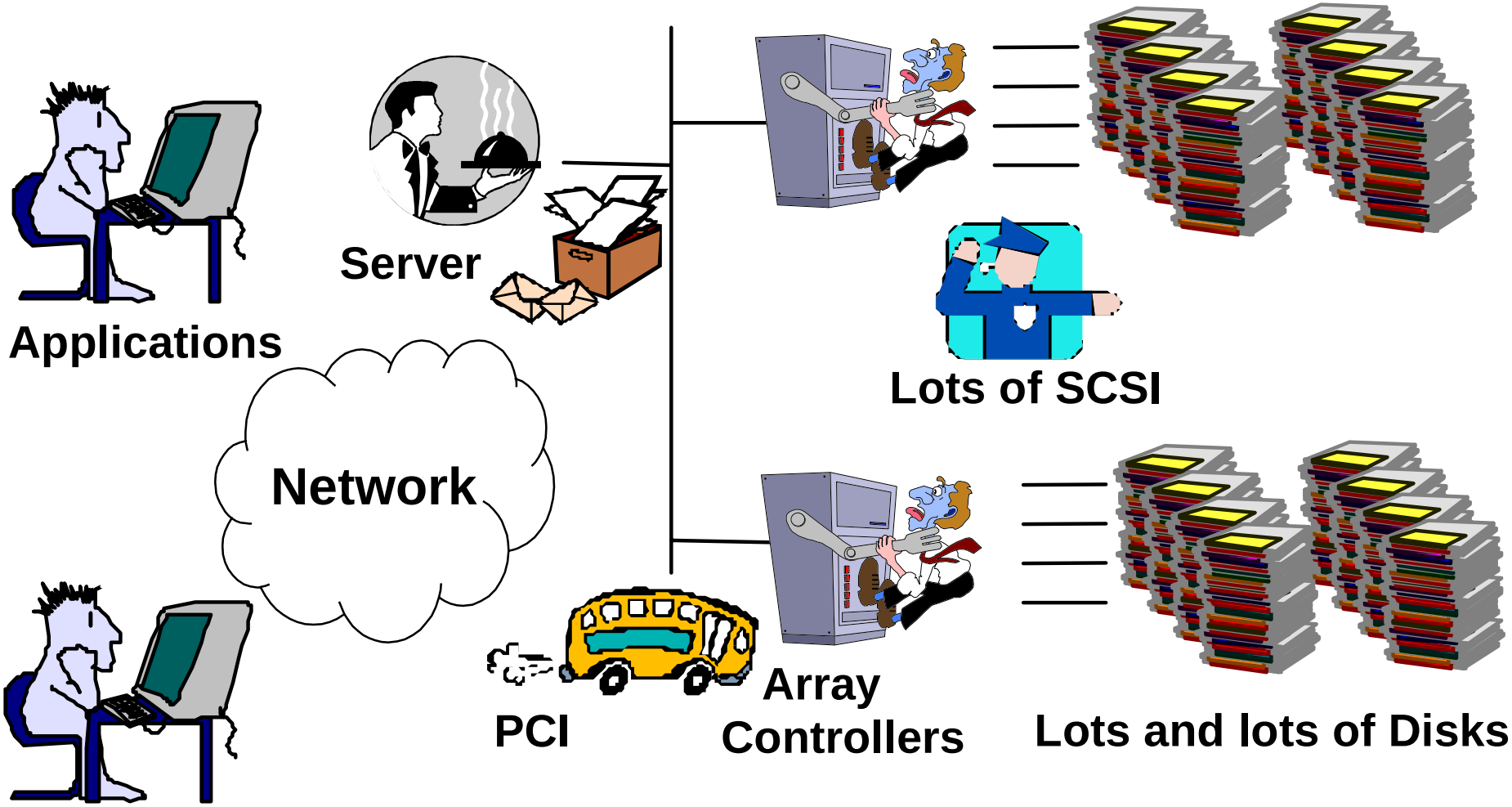
- **Memory speed is ultimate limit**
  with multiple disks, multiple PCI
  50MBps copy, 150 MBps r/w
- **Avoid FS buffering above 16KB**
  costs 20 ms/MB of cpu

- **Pitfalls**
  - read-before-write: 2KB buffered IO
  - allocate/extend: synchronous write
  - zoned disks => 50% speed bump
  - stripe alignment => 20% bump

More Details

- Web site has
  - » Paper
  - » Sample code
  - » Benchmark program
  - » These slides
  - » http://research.Microsoft.com/BARC/Sequential_IO

The More Complicated Picture



- But there *is* hope, help is on the way...

Shameless Plugs

- Network-Attached Secure Disks *- put disks on the network*
  - » www.pdl.cs.cmu.edu/NASD
- SAN/VIA *- make networks fast & cheap*
  - » www.viarch.org
- Active Disks *- take advantage of processing power on all those disks*
  - » www.pdl.cs.cmu.edu/Active

- Microsoft Research
  - » www.research.microsoft.com
- Carnegie Mellon
  - » www.cs.cmu.edu, www.ece.cmu.edu

Microsoft Research

Carnegie Mellon University

Details/Pitfalls/Extras

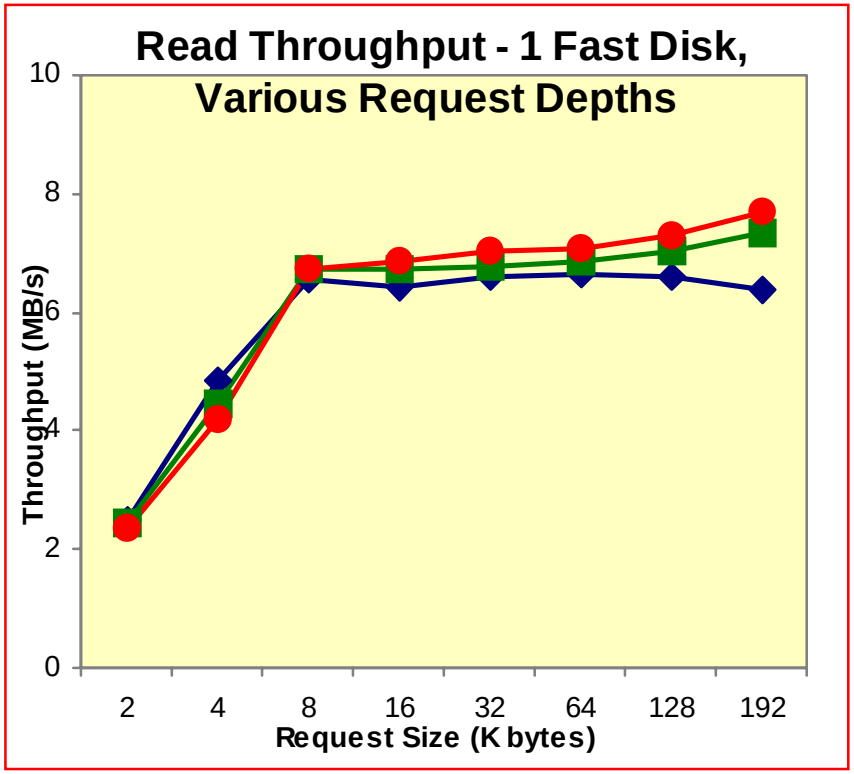
Simplest Possible Code

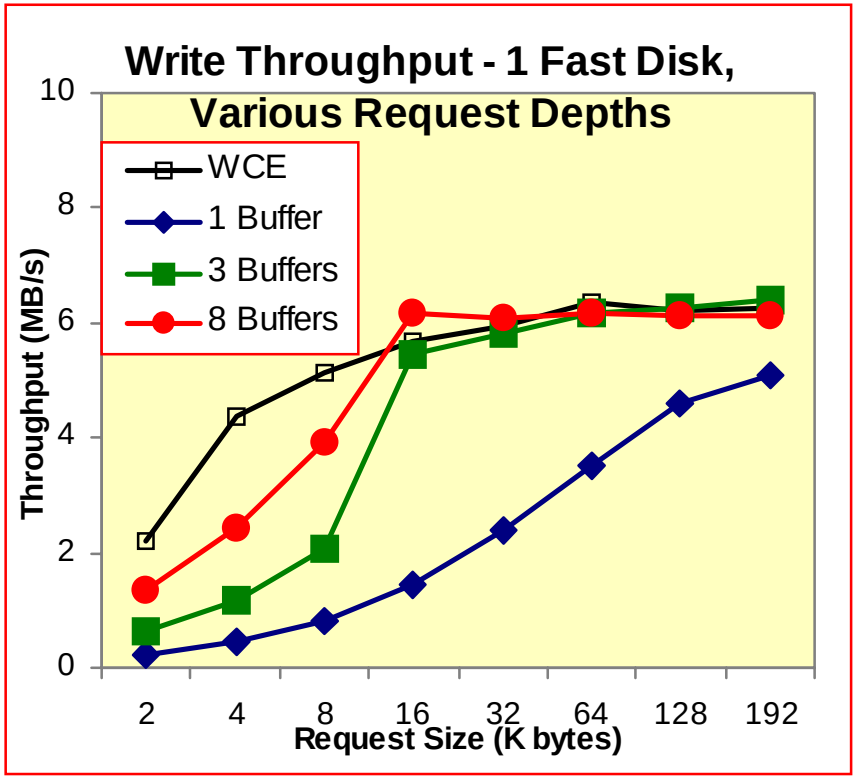
```
#include <stdio.h>
#include <windows.h>

int main()
{ const int iREQUEST_SIZE = 65536;
  char cRequest[iREQUEST_SIZE];
  unsigned long ibytes;

  HANDLE hFile = CreateFile("C:\input.dat",    // name
                      GENERIC_READ,            // desired access
                      0, NULL,                 // share & security
                      OPEN_EXISTING,           // pre-existing file
                      FILE_ATTRIBUTE_TEMPORARY | FILE_FLAG_SEQUENTIAL_SCAN,
                      NULL);                   // file template

  while(ReadFile(hFile,cRequest,iREQUEST_SIZE,&ibytes,NULL) ) // do read
  { if (ibytes == 0) break;     // break on end of file
    /* do something with the data */ };

  CloseHandle(hFile);
  return 0;
  }
```

- Error checking adds some more, but still, its easy

Parallelism - One Disk

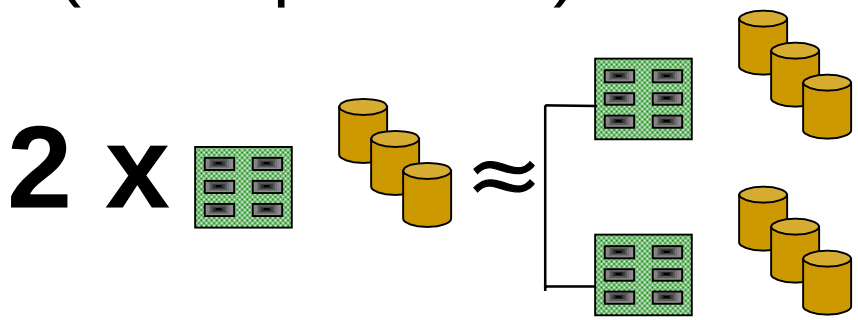
- Not much benefit with single disk
- Asynchronous requests get close to WCE

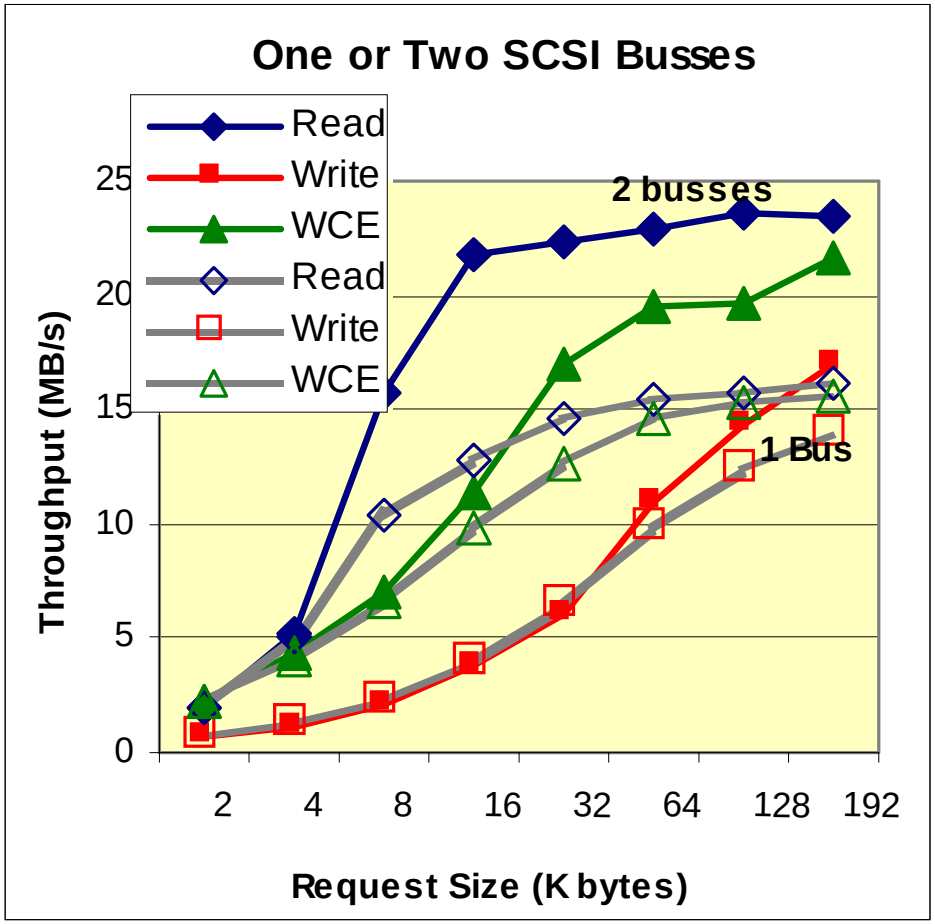




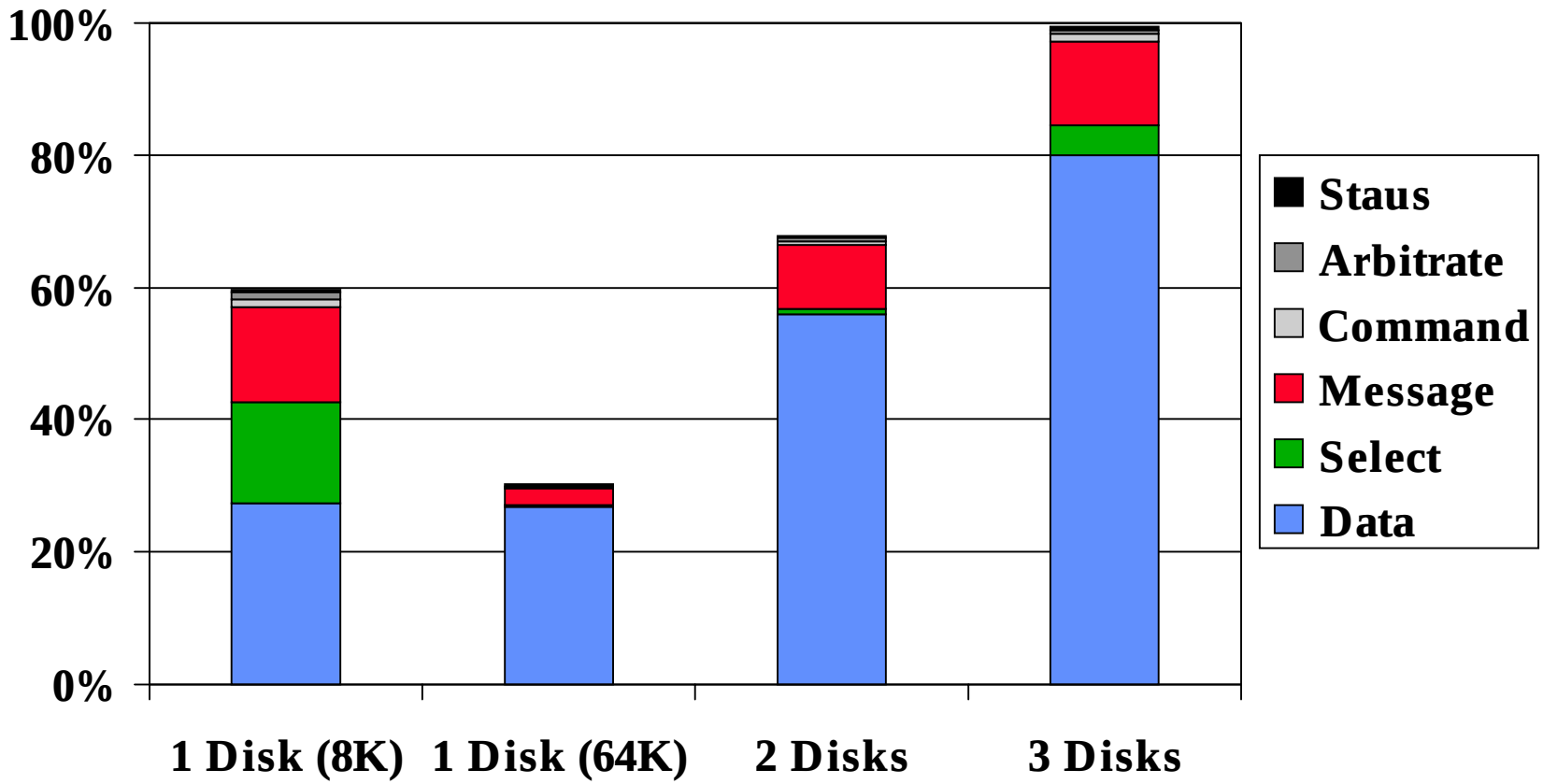
More Busses

- Second SCSI bus nearly doubles throughput
- Writes need deeper buffers
- Experiment is unbuffered (3-deep +WCE)





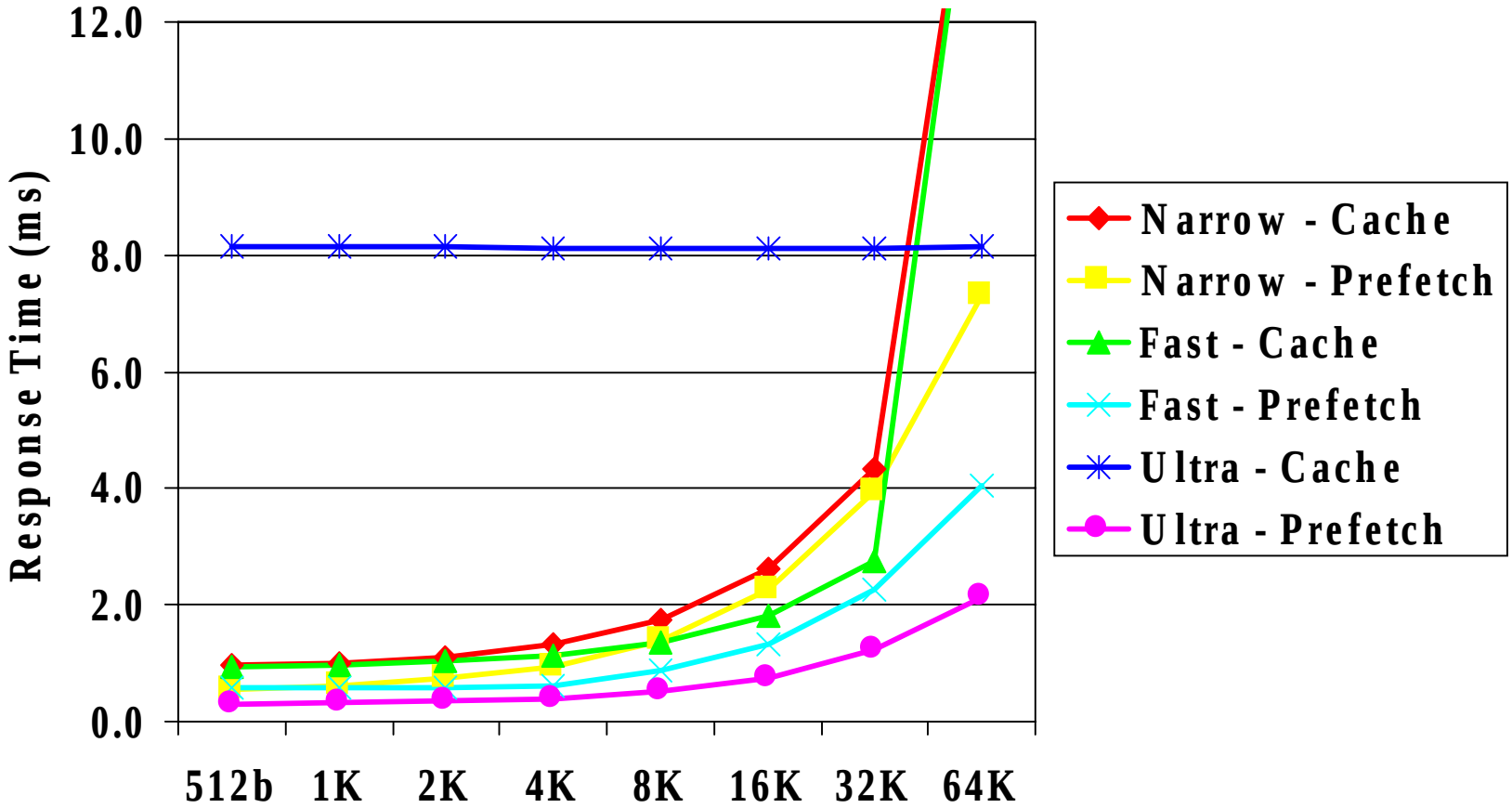
SCSI Bus Traffic



Carnegie Mellon University

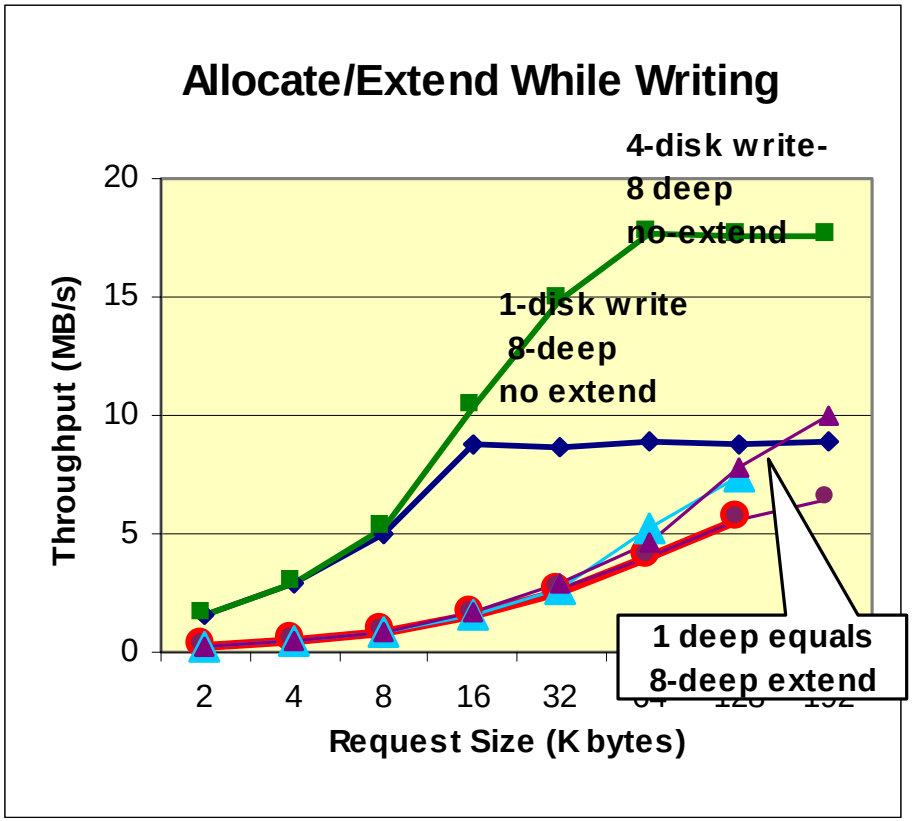
Microsoft Research

Disk Behavior



Allocate / Extend Forces Sync Writes

- When you allocate space NT zeros it (both DRAM and disk)
- Prevents others from reading data you delete
- This “kills” write parallelism
- Solution: pre-allocate or reuse files
- Do VERY large writes

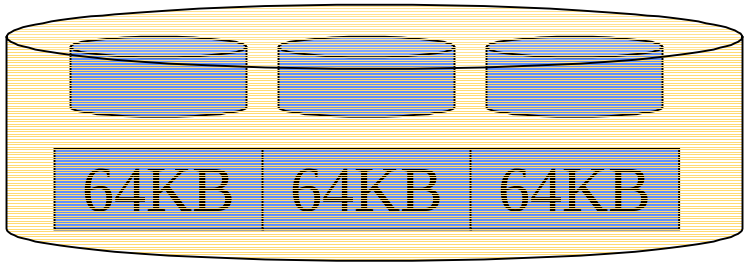


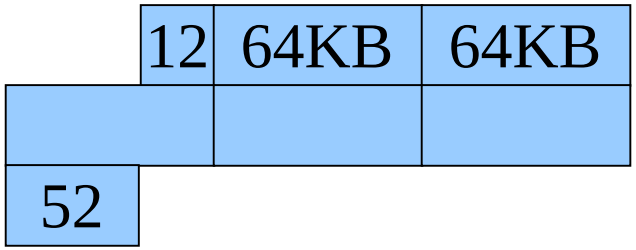
Carnegie Mellon University

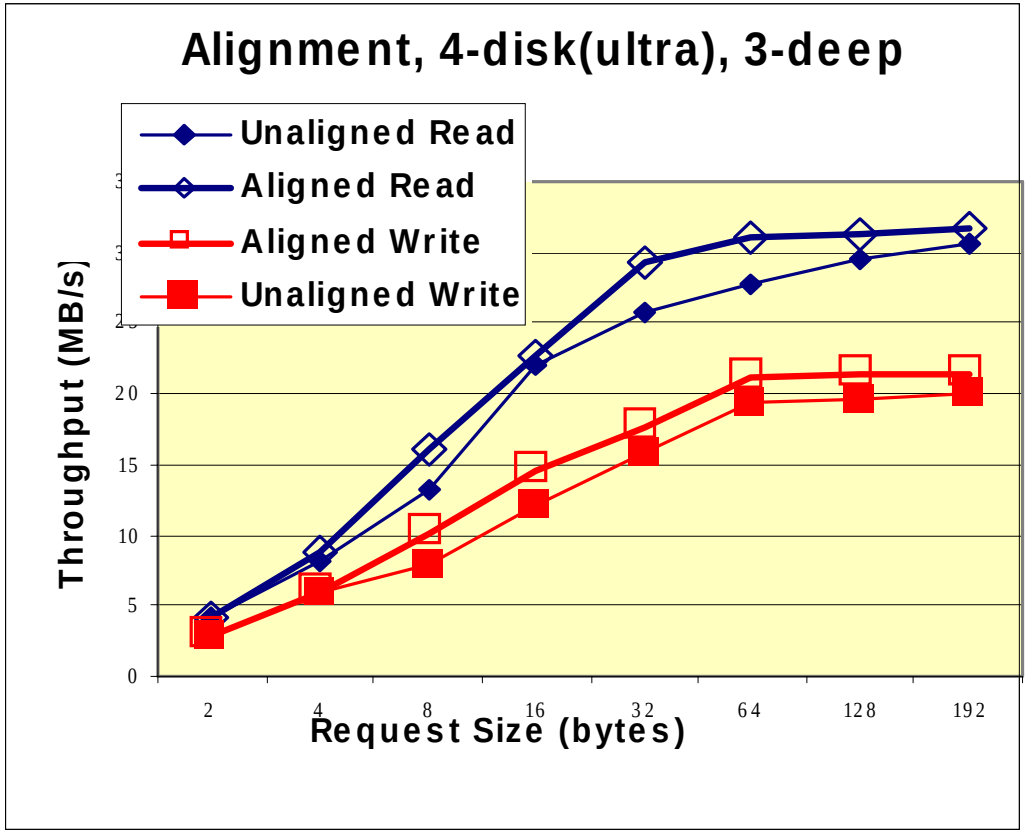
Microsoft Research

Alignment - Chunk vs. Cluster

- Filesystem has *allocation unit*
  - » default is 4 KB
- Stripe *chunk size*
  - » locked at 64 KB







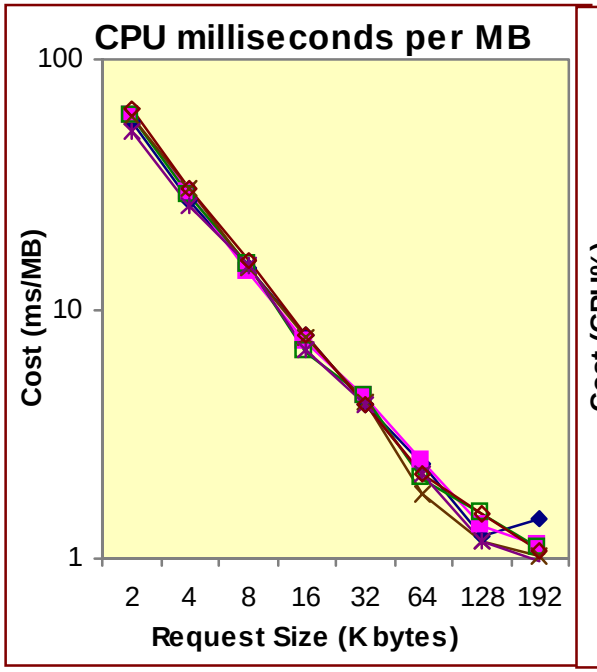
Carnegie Mellon University

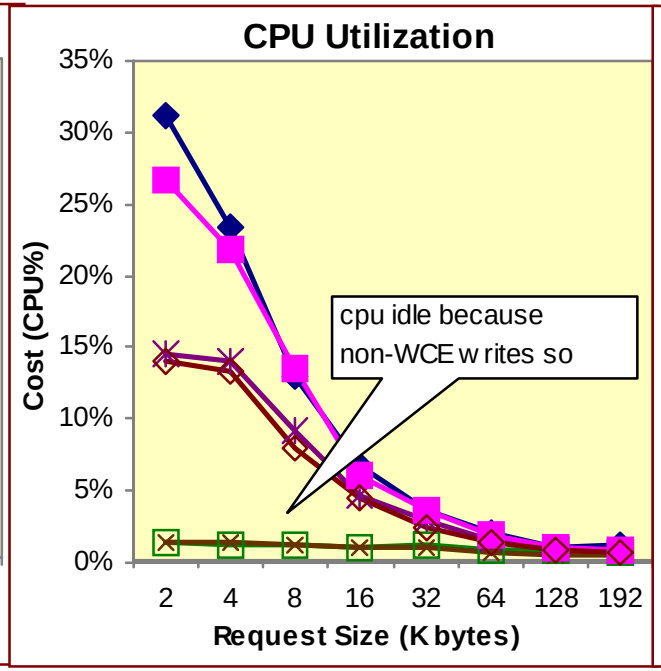
Microsoft Research

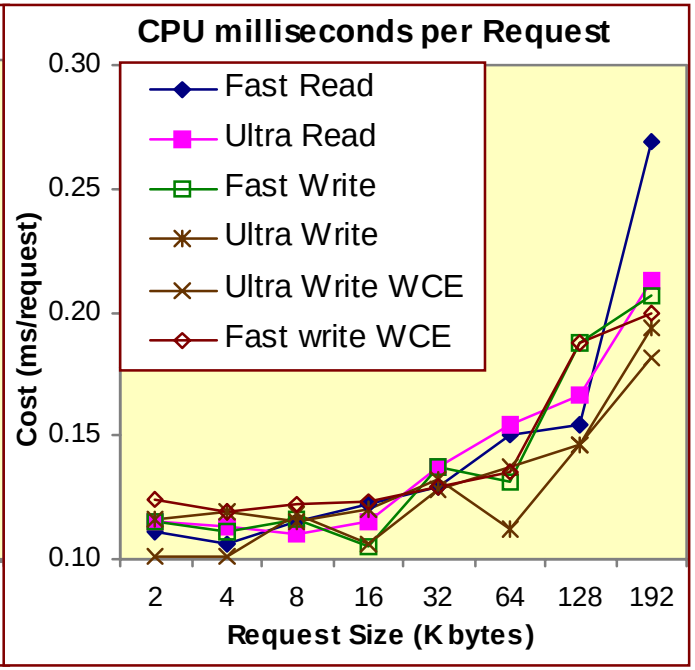
Cost of Unbuffered I/O

- Saves buffer memcpy()
- Was 20 ms/MB, now 2 ms/MB
- Cost/request ~ 120 µs

- Buffered:
  - » saturates CPU at 50 MB/s
- Unbuffered
  - » saturates CPU at 500 MB/s

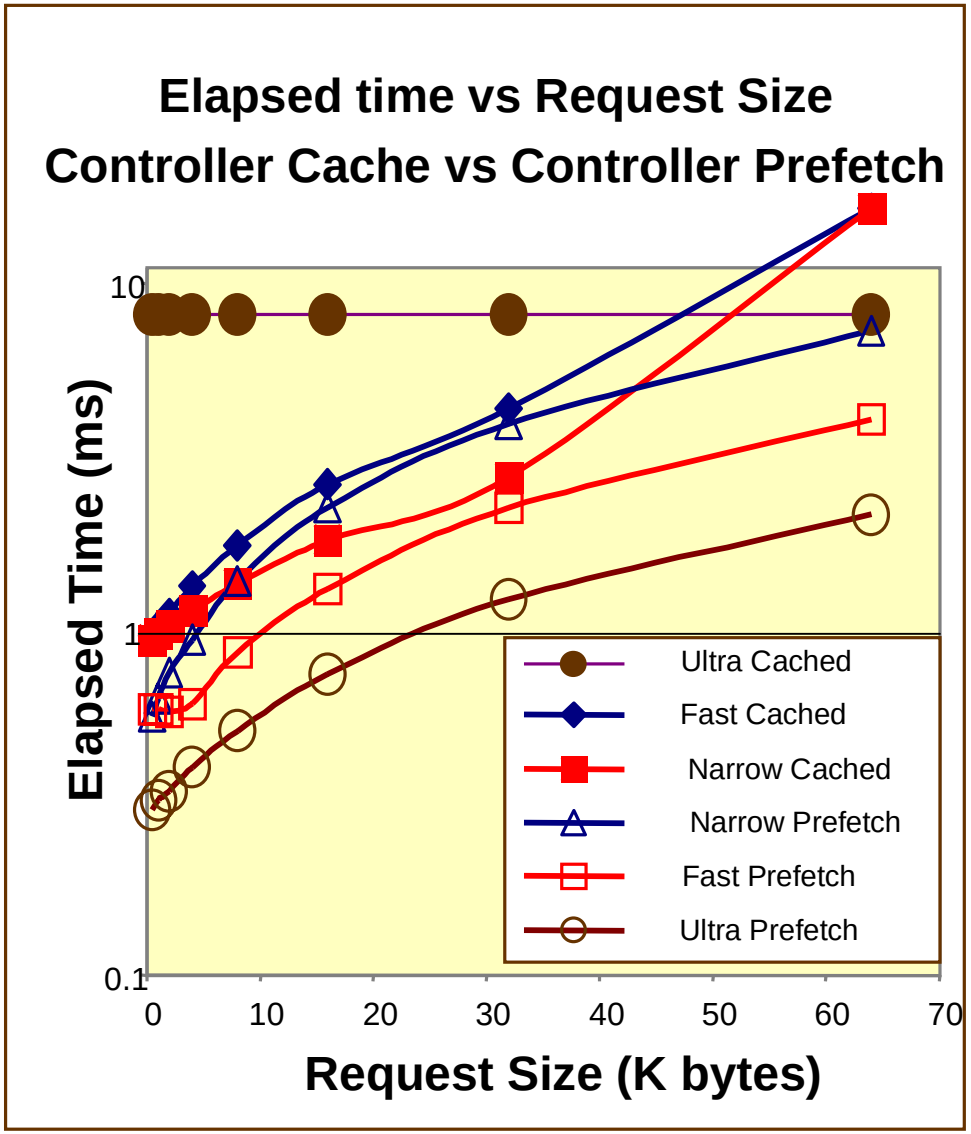






Disks & Adapters are Complex

- Minimum response time is 300 µs
- Typical 1 ms for 8 KB
- A number of “interesting” effects



File System & Stripes

- FS buffering helps small reads
- FS buffered writes peak at 12MBps
- 3-deep async helps