HEC-IWG FS&I/O R&D Workshop

Randy Melen

SLAC/SCCS

August 16, 2005

- Our environment
  - ~ 2PB of active data for BaBar experiment, but growing still
  - Data analysis (mining) done with random reads of small blocks (2KB down to 100 bytes)
  - A researcher typically has several hundred simultaneous analysis streams (in batch)
  - And several hundred concurrent researchers are active

- Our problem
  - So several thousand simultaneous streams of random (unpredictable, readahead doesn’t help) read requests to disk
  - Latency from client request to receiving data is 7000 to 12000 microseconds
  - Data space is probably 10 to 32TB right now, probably 256TB within a few years

- We need latencies between disk and DDR memory latencies
- So why not just buy a very big SMP from the usual vendors with massive memory?
- Because we also need a price point between disk and DDR memory
- We do not need cache coherency for our read-mostly requirement

- To begin exploring this area, we have built a "toy" 64-host 1TB memory cluster using commodity hardware with DDR memory
- DDR memory is still too expensive to scale up to 10 to 32TB that is needed for a real test
- Using xrootd from the HEP world to test usefulness

- Stress tests done with no client computation, just data access
- Measured latency drops to about 200 microseconds

SLAC Scientific Computing Drivers

- BaBar (data-taking ends December 2008)
  - The world's most data-driven experiment
  - Data analysis challenges until the end of the decade
- KIPAC
  - From cosmological modeling to petabyte data analysis
- Photon Science at SSRL and LCLS
  - Ultrafast Science, modeling and data analysis
- Accelerator Science
  - Modeling electromagnetic structures (PDE solvers in a demanding application)
- The Broader US HEP Program (aka LHC)
  - Contributes to the orientation of SLAC Scientific Computing R&D

Future Work: Latency Reduction (All require work with vendors)

- Operating system and TCP stack enhancements
- TCP stack bypass
  - RDMA
  - MPI-optimized service
- Network card driver optimization
- TOE (not good if bandwidth-focused)

Use of Prototype for SLAC Science

- BaBar
  - Host part of the (~30 TB) microDST data
  - Access data via “pointer skims”
  - Both normal production use and intensified tests with ‘real’ access patterns an super-real access rates.
- GLAST
  - Will require a ~2TB intensely accessed database. Have asked to test concepts on the PetaCache Prototype
- LSST Database Prototyping
  - Proposed tests using the PetaCache prototype

Development Machine

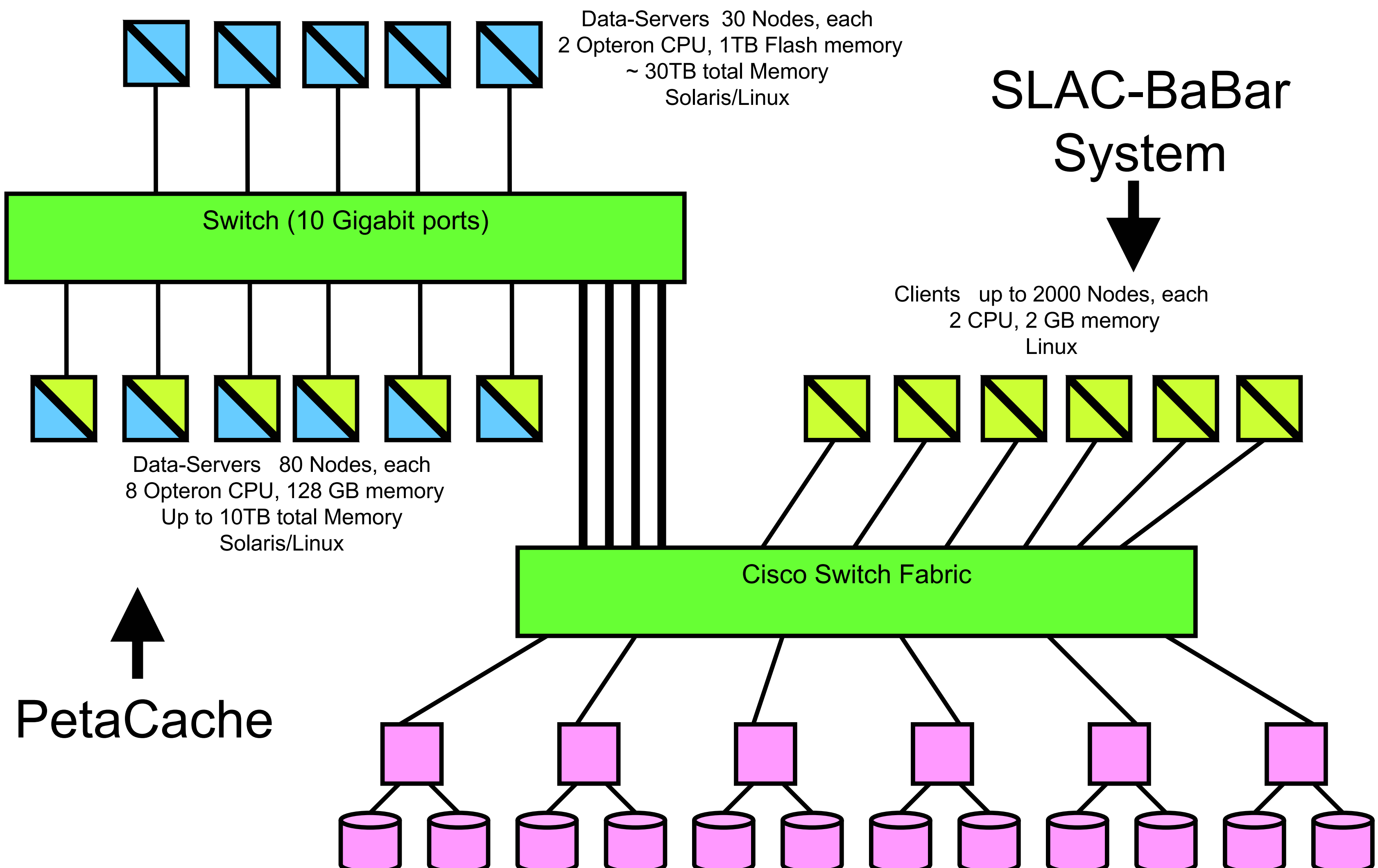
- Ideas for Storage-Class Memory
- Likely configuration

Storage-Class Memory

- New technologies coming to market in the next 3 – 10 years (Jai Menon – IBM)
- Current not-quite-crazy example is flash memory

Flash Memory

Development Machine Plans



Summary

- Data-intensive science increasingly requires low-latency access to terabytes or petabytes
- Memory is one key:
  - Commodity DRAM today (increasing total cost by ~2x)
  - Storage-class memory (whatever that will be) in the future
- Revolutions in scientific data analysis will be another key
  - Current HEP approaches to data analysis assume that random access is prohibitively expensive
  - As a result, permitting random access brings much-less-than-revolutionary immediate benefit
- Use the impressive motive force of a major HEP collaboration with huge data-analysis needs to drive the development of techniques for revolutionary exploitation of an above-threshold machine.

PetaCache

Huge-Memory Architecture
for
Data-Intensive Science

Richard P. Mount

SLAC

August 16, 2005

PetaCache Goals

- The PetaCache architecture aims at revolutionizing the query and analysis of scientific databases with complex structure.
  – Generally this applies to feature databases (terabytes–petabytes) rather than bulk data (petabytes–exabytes)
- The original motivation comes from HEP
  – Sparse (~random) access to tens of terabytes today, petabytes tomorrow
  – Access by thousands of processors today, tens of thousands tomorrow

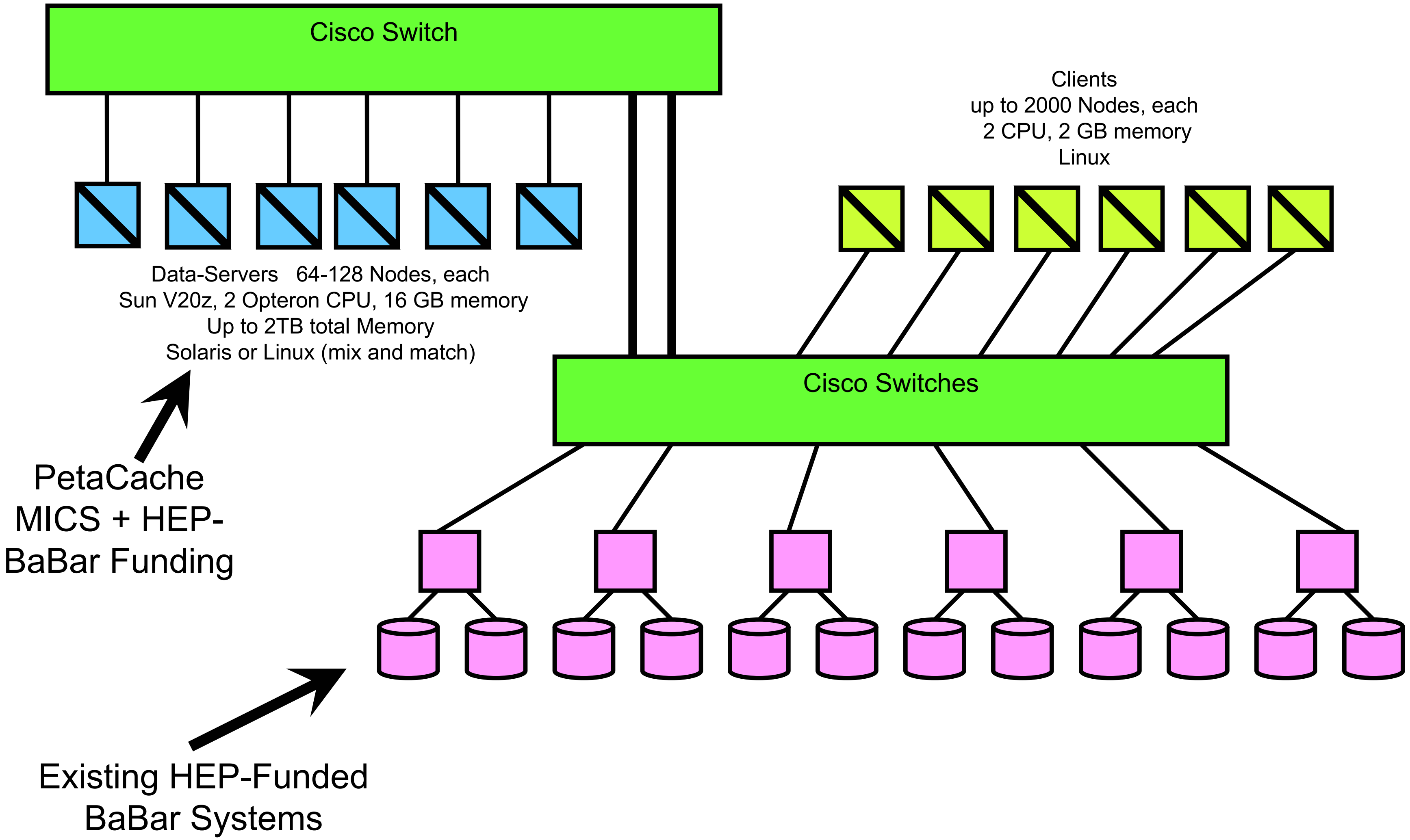
Prototype (Development) Machine Design Goals

- Attractive to scientists
  - Big enough data-cache capacity to promise revolutionary benefits
  - 1000 or more processors
- Processor to (any) data-cache memory latency < 100 μs
- Aggregate bandwidth to data-cache memory > 10 times that to a similar sized disk cache
- Data-cache memory should be 3% to 10% of the working set (approximately 10 to 30 terabytes for BaBar)
- Cost effective, but acceptably reliable
  - Constructed from carefully selected commodity components

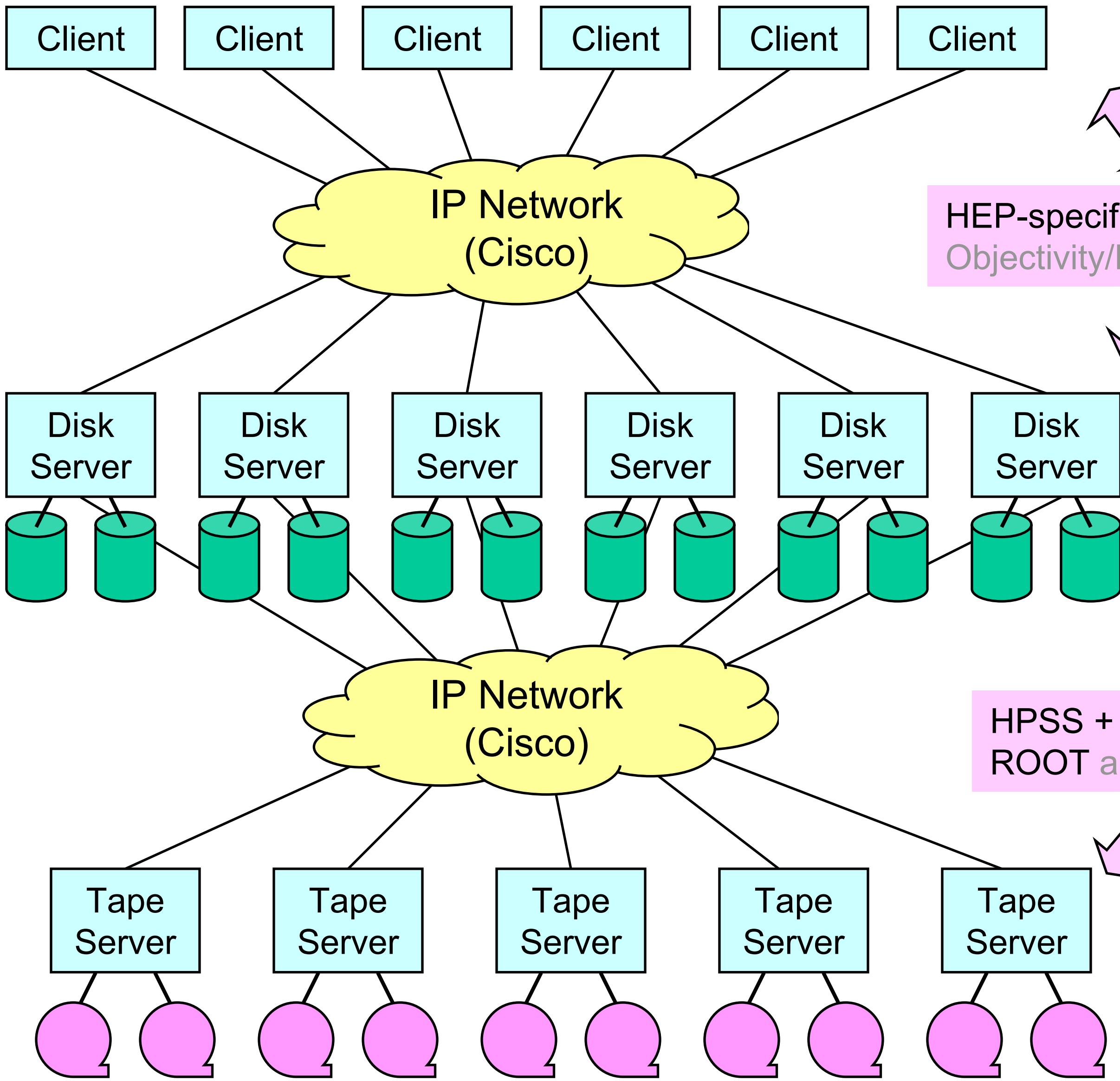
Prototype (Development) Machine Design Choices

- Intel/AMD server mainboards with 4 or more ECC dimm slots per processor
- 2 Gbyte dimms ($550 each)
- 4 Gbyte dimms ($7,000 each) too expensive this year
- 64-bit operating system and processor
  - Favors Solaris and AMD Opteron
- Large (500+ port) switch fabric
  - Large Ethernet switches are most cost-effective
- Use of ($10M+) BaBar disk/tape infrastructure, augmented for any non-BaBar use

Prototype Machine (Operational)



SLAC-BaBar Computing Fabric



1700 dual CPU Linux
400 single CPU
Sun/Solaris

HEP-specific ROOT software (Xrootd) + Objectivity/DB object database

120 dual/quad CPU
Sun/Solaris
~400 TB Sun
FibreChannel RAID
arrays

HPSS + SLAC enhancements to ROOT and Objectivity server code

25 dual CPU
Sun/Solaris
40 STK 9940B
6 STK 9840A
6 STK Powderhorn
over 1 PB of data

Object-Serving Software

- Xrootd/olbd (Andy Hanushevsky/SLAC)
  - Optimized for read-only access
  - File-access paradigm (filename, offset, bytecount)
  - Make 1000s of servers transparent to user code
  - Load balancing
  - Self-organizing
  - Automatic staging from tape
  - Failure recovery
- Allows BaBar to start getting benefit from a new data-access architecture within months without changes to user code
- The application can ignore the hundreds of separate address spaces in the data-cache memory

Making the Server Perform

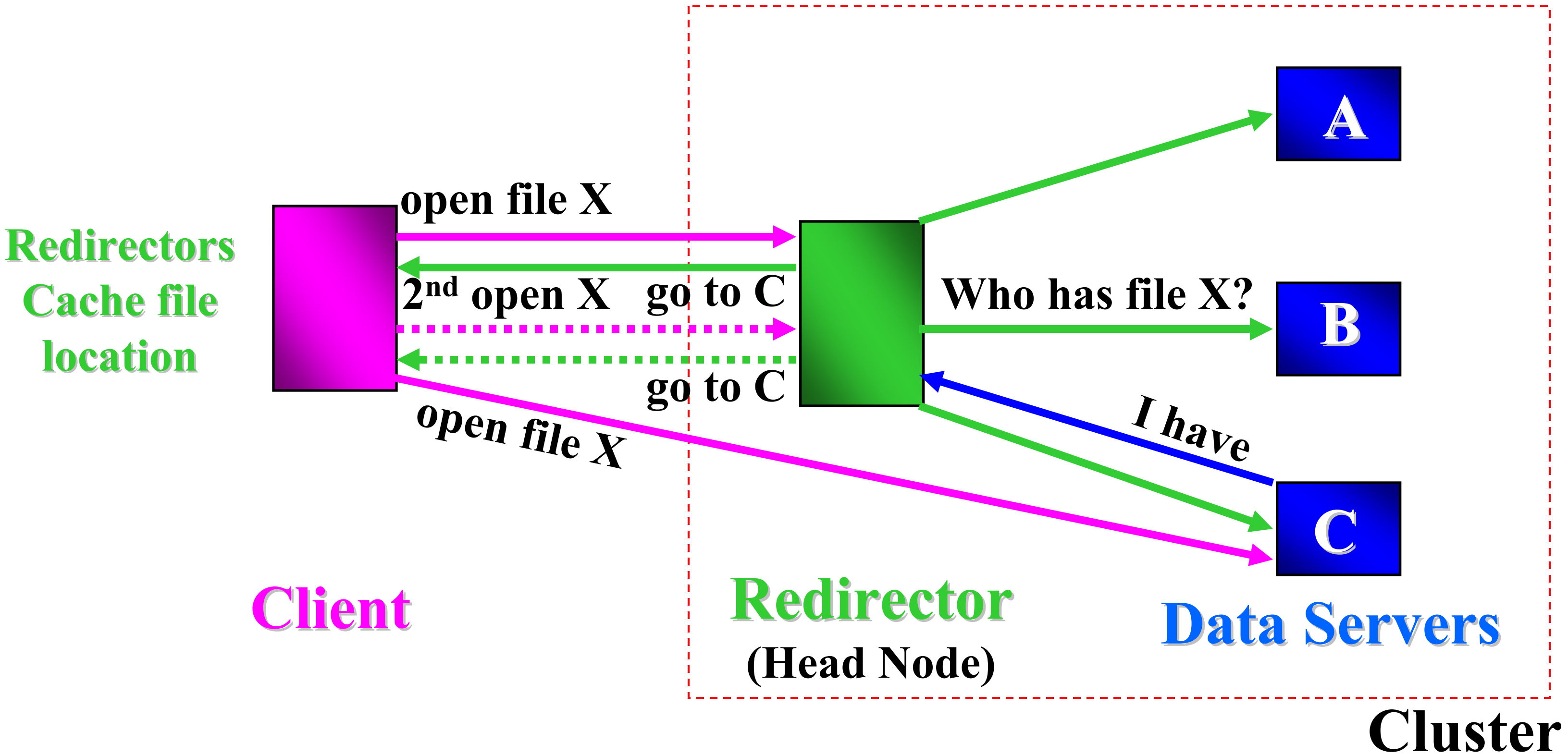
- Solve ***only*** the problem at hand
  - Avoids high overhead but unused features
  - **xrootd** is only a **D**ata **A**ccess **S**ystem
  - It may look like a file system but it is ***not*** one
    - Avoids high overhead consistency semantics
    - Not needed in write once read many applications

This is common sense that is hard to follow

Basic Cluster Architecture

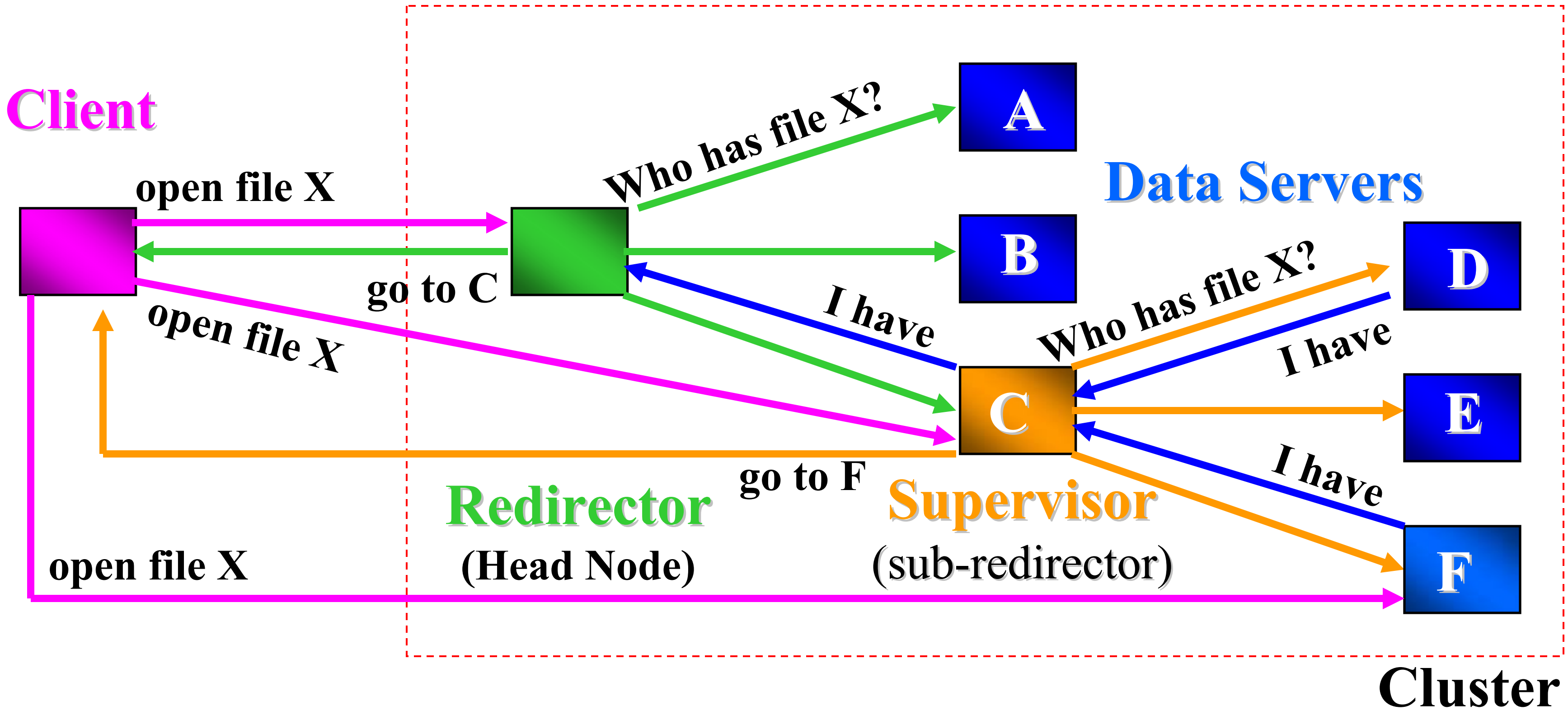
- Software cross bar switch
  - Allows point-to-point connections
    - Client and data server
  - I/O performance not compromised
    - Assuming switch overhead can be amortized
- Scale interconnections by stacking switches
  - Virtually unlimited connection points
    - Switch overhead must be very low

Single Level Switch



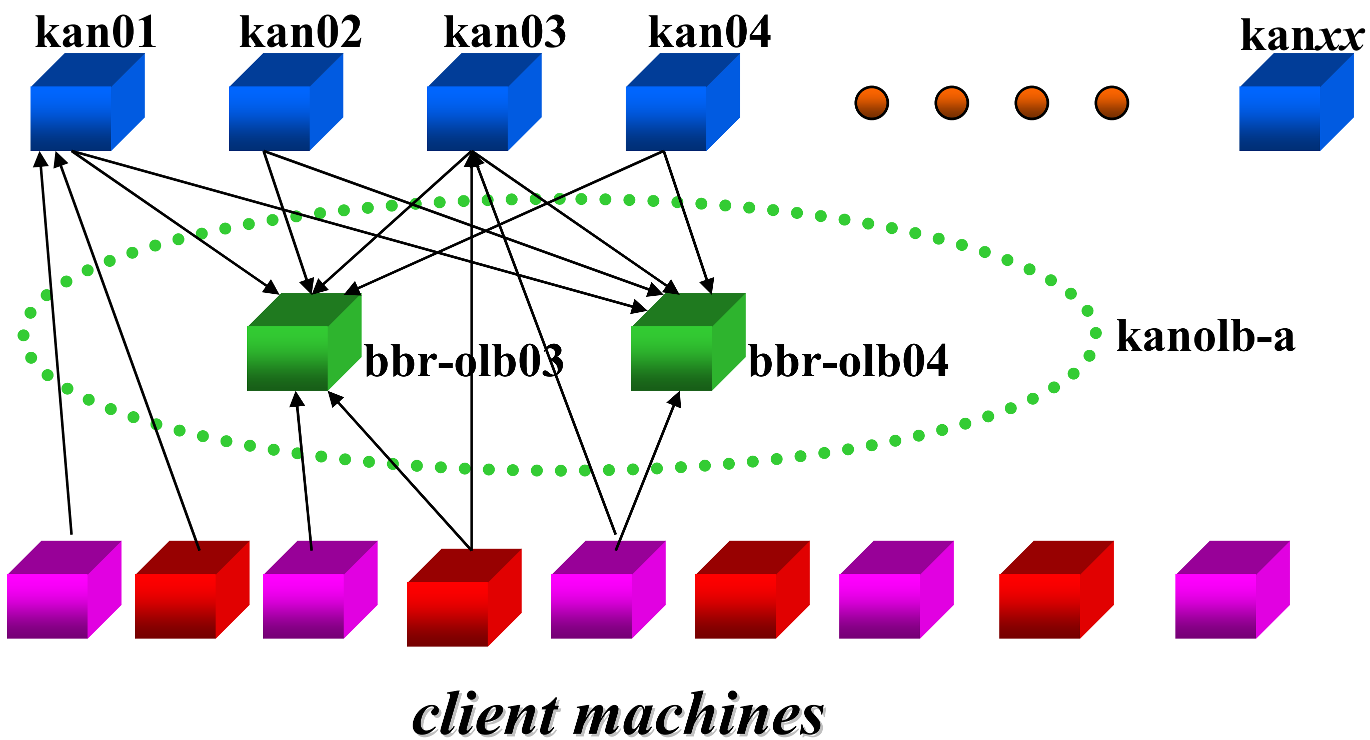
Client sees all servers as xrootd data servers

Two Level Switch



Client sees all servers as xrootd data servers

Example: SLAC Configuration



Hidden Details

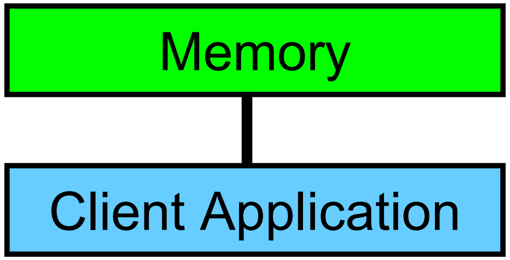
Making Clusters Efficient

- Cell size, structure, & search protocol are critical
  - Cell Size is 64
    - Limits direct inter-chatter to 64 entities
    - Compresses incoming information by up to a factor of 64
    - Can use very efficient 64-bit logical operations
  - Hierarchical structures usually most efficient
    - Cells arranged in a B-Tree (i.e., B64-Tree)
    - Scales 64^h (where h is the tree height)
      - Client needs h-1 hops to find one of 64^h servers (2 hops for 262,144 servers)
      - Number of responses is bounded at each level of the tree
  - Search is a directed broadcast query/rarely respond protocol
    - Provably best scheme if less than 50% of servers have the wanted file
      - Generally true if number of files >> cluster capacity
      - Cluster protocol becomes more efficient as it grows

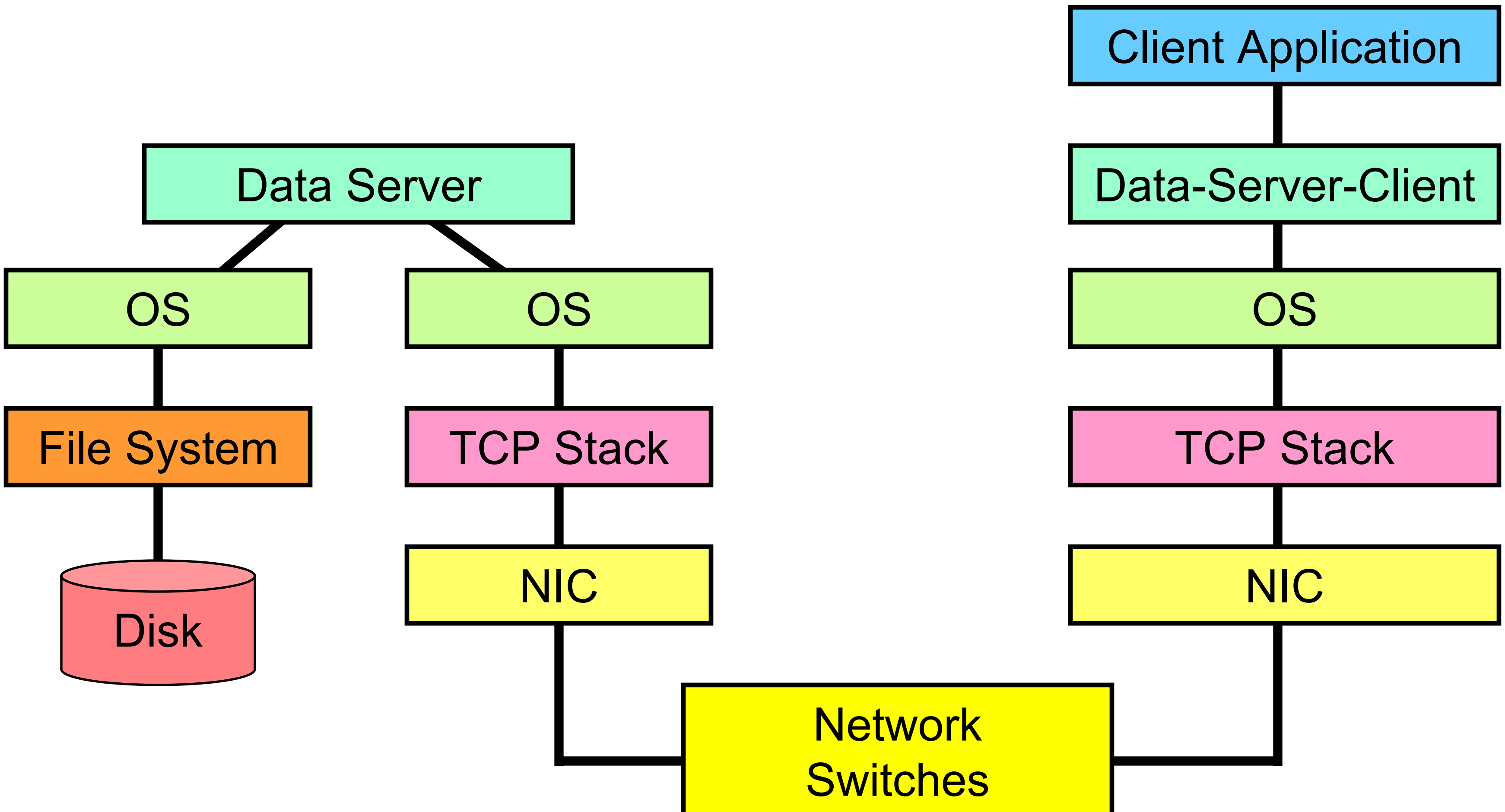
Cluster Scale Management

- Massive clusters must be self-managing
  - Scales 64^n where n is height of tree
    - Scales very quickly ($64^2 = 4096$, $64^3 = 262{,}144$)
    - Well beyond direct human management capabilities
  - Therefore clusters self-organize
    - Uses a minimal spanning tree algorithm
      - 280 nodes self-cluster in about 7 seconds
      - 890 nodes self-cluster in about 56 seconds
    - Most overhead is in wait time to prevent thrashing

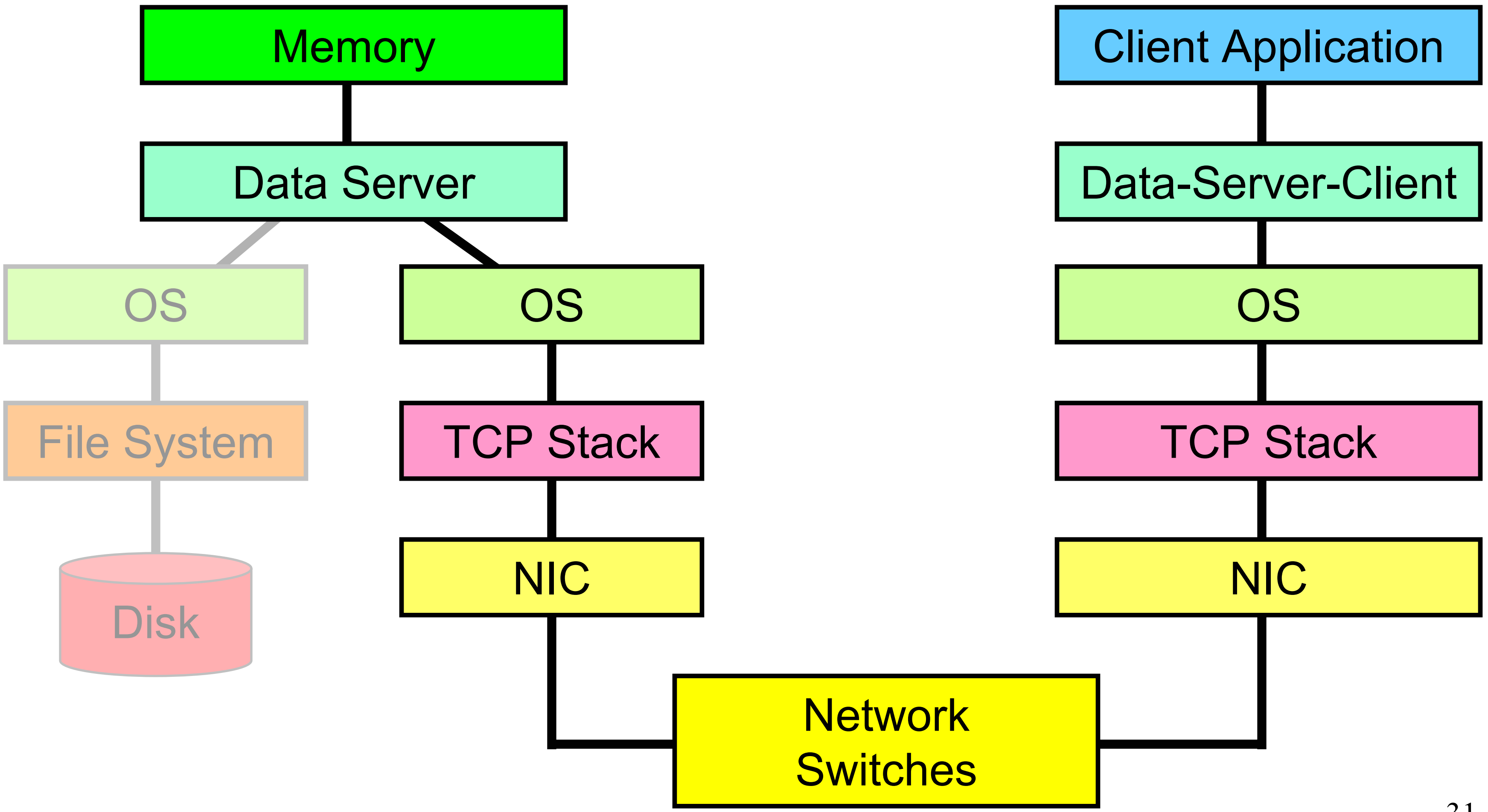
Latency (1)
Ideal



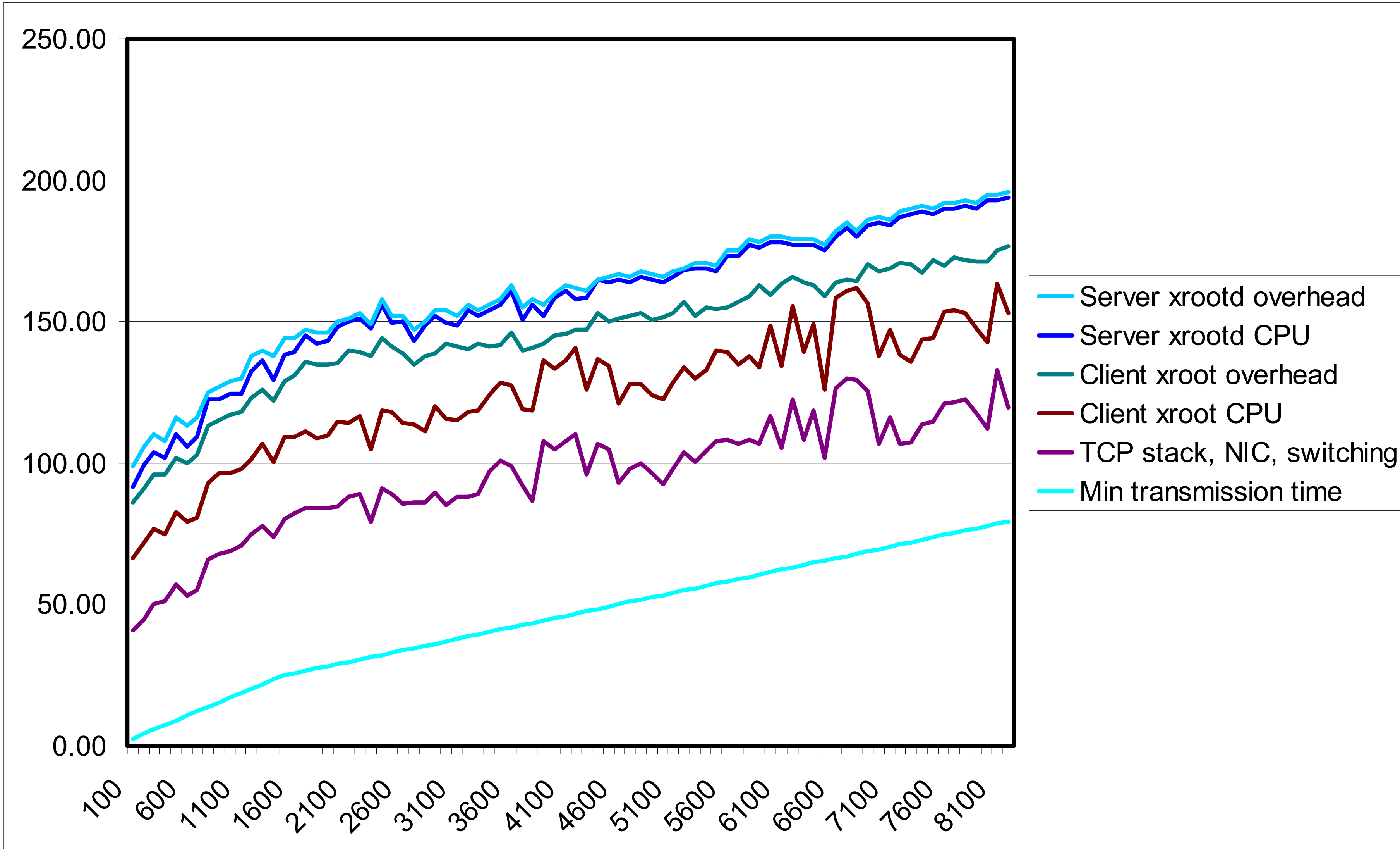
Latency (2)
Current reality for Disk-based Servers



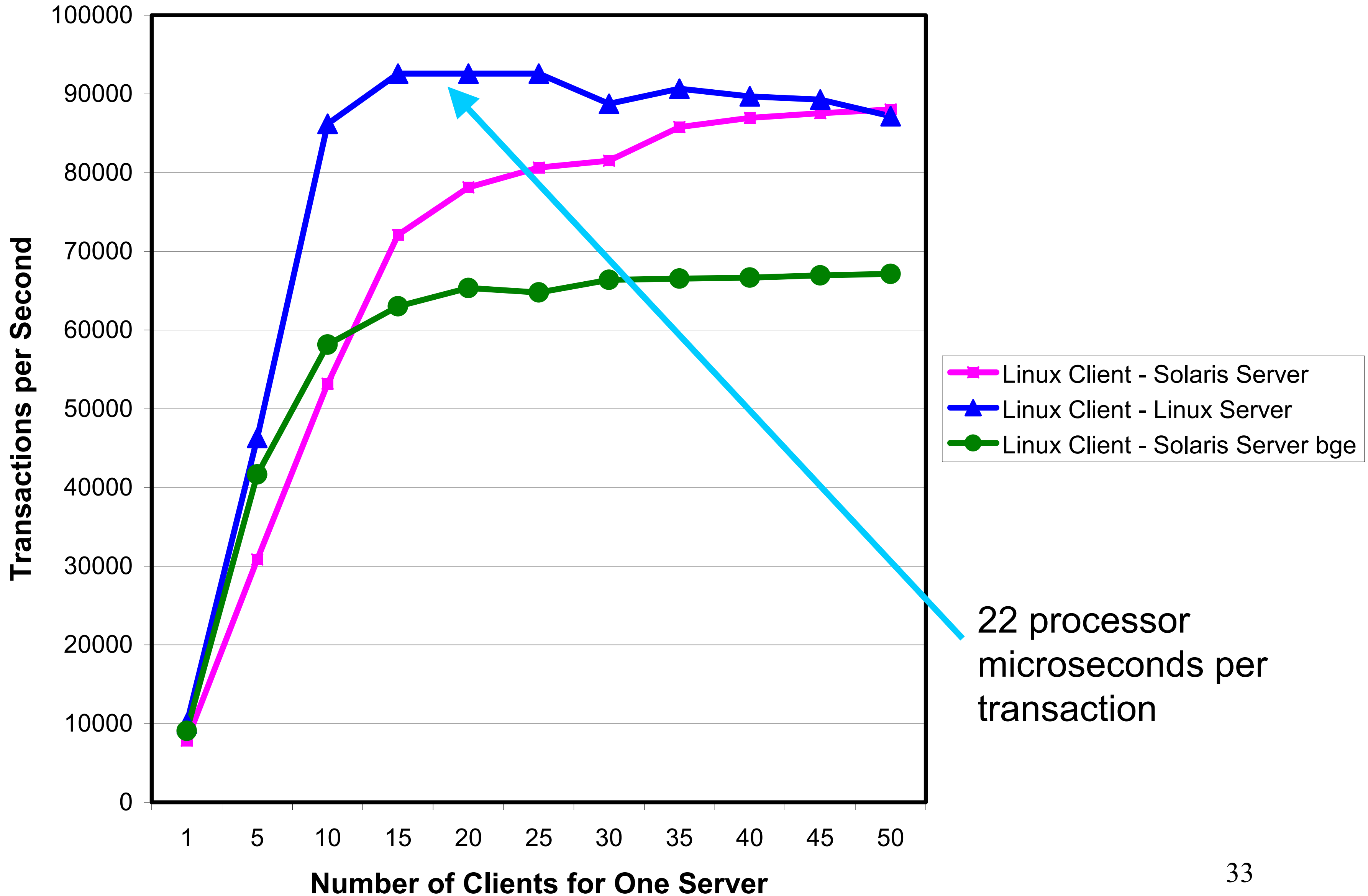
Latency (3)
Practical Goal for Prototype



Latency (microseconds) versus data retrieved (bytes)



Throughput Measurements



xrootd self-organiation

Number of xrootd/olbd servers (n)	Time required to self-organize (seconds)	Time = an^x
280	7	
890	86 (first start to last finish) 56 (last start to last finish)	x = 1.9 x = 2.3