# High-Performance Data Intensive Distributed Computing

Brian L. Tierney
(bltierney@lbl.gov)
Future Technologies Group
Lawrence Berkeley National Laboratory

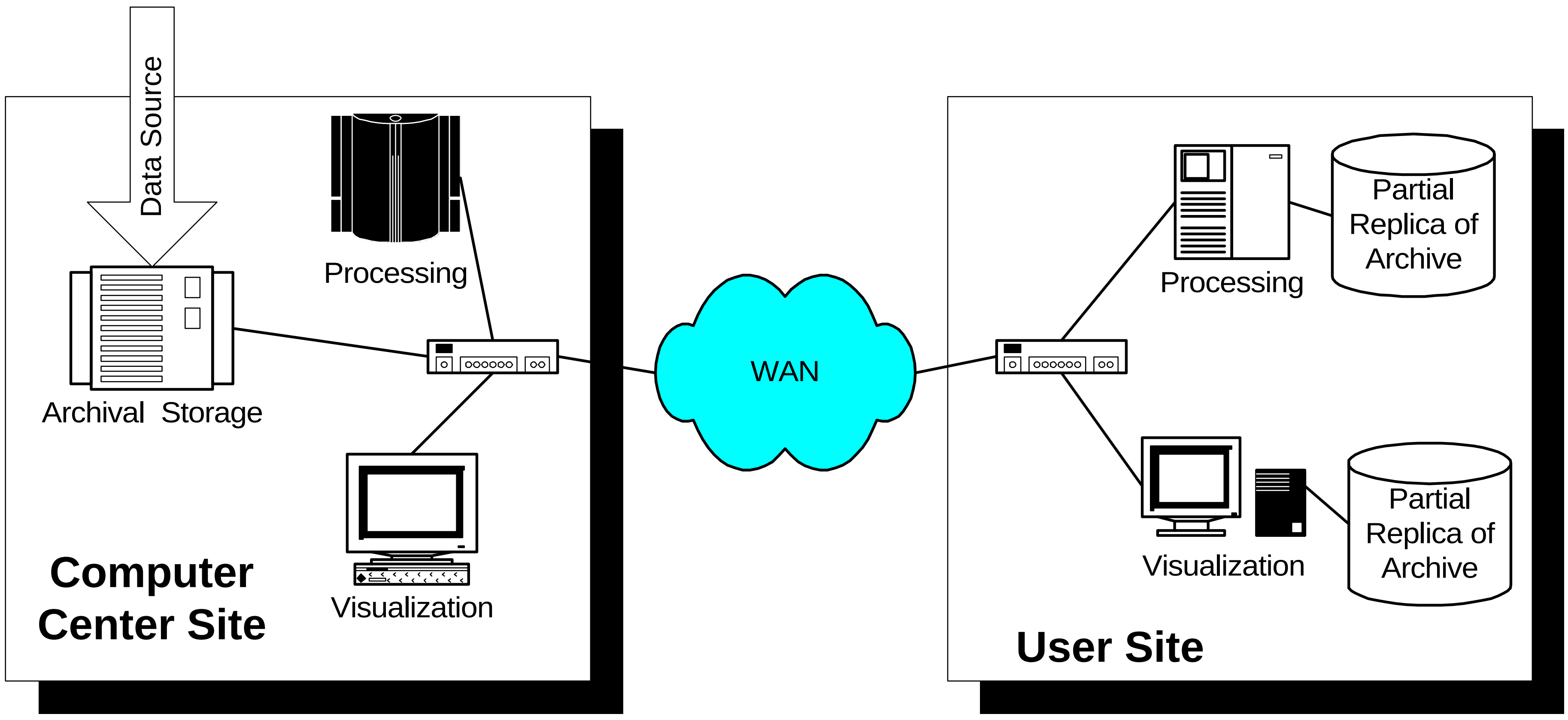
# Outline

- Architectures for Data Intensive Computing
- The LBNL Distributed Parallel Storage System
- China Clipper Experiment
- Performance Analysis Tools: NetLogger
- Current Work: The “Data Grid”

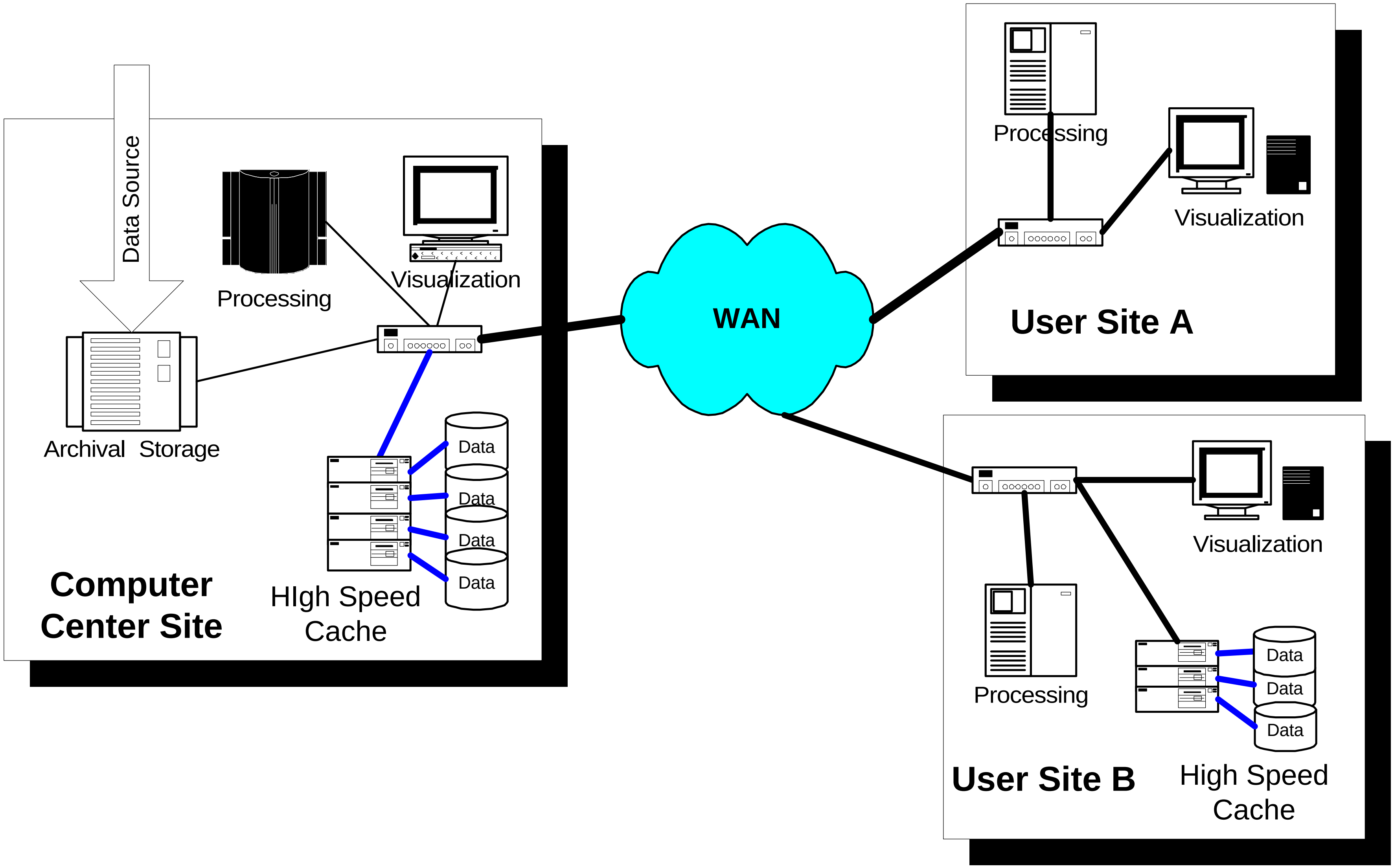
# Why is Remote Storage Important?

- Why is distributed storage important for Data Intensive Computing?
  - Researchers often are not at the same location as the data source
  - Compute cycles are often not at the same location as the data source or the data archive

# Remote Access to a Large Data Archive

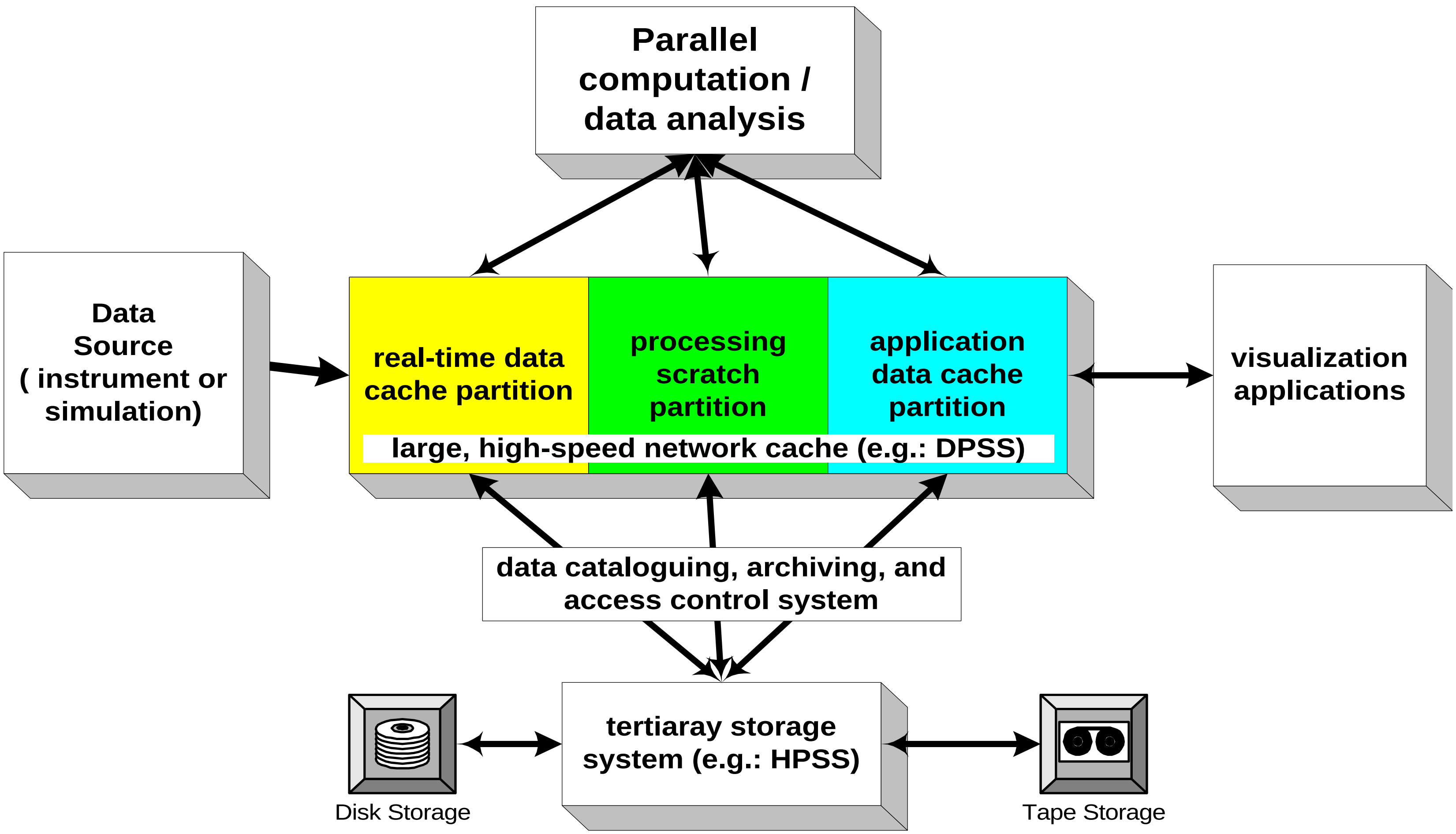


# Remote Access to a Large Data Archive using a Data Cache



# Data Architecture

## Architecture for Data Intensive Distributed Computing



# Key features of the architecture

- Allows for high-speed access to very large scientific data sets using an http-like model
  - — don't download entire web site, only the parts required immediately
  - — don't copy entire data set, only the parts of the data as it is needed
- very high-speed data cache that is distributed, scaleable, and dynamically configurable
- high-speed tertiary storage interface
- data cataloguing system
- access control system

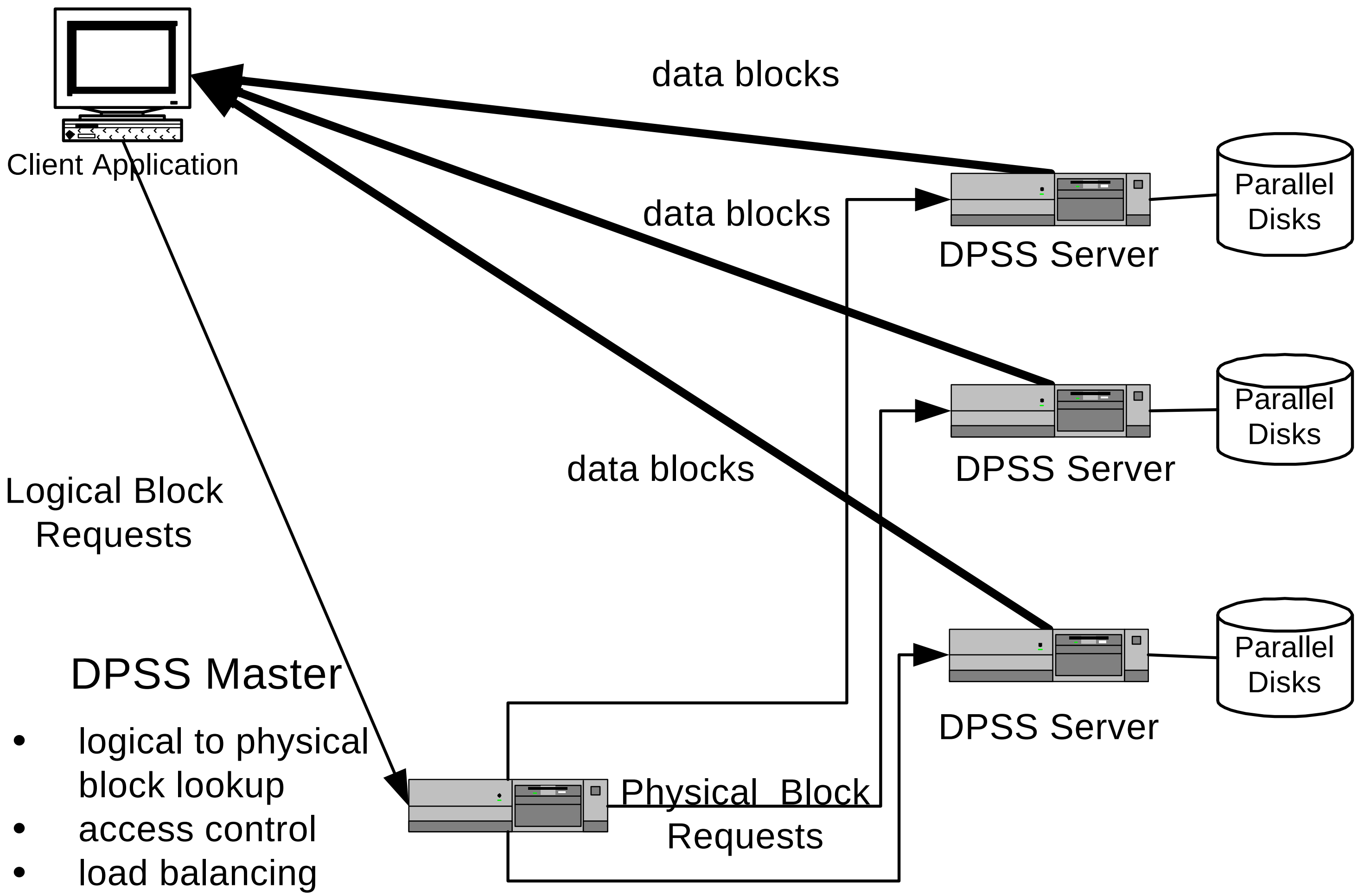
# The Distributed Parallel Storage Server (DPSS)

- Our implementation of this data cache is called the DPSS
  — provides high-speed parallel access to remote data
  — Similar to a striped RAID system, but tuned for WAN access
    - data is striped across both disks and servers
  — On a high-speed network, can actually access remote data faster that from a local disk
    - 57 MB/sec vs 10 MB/sec

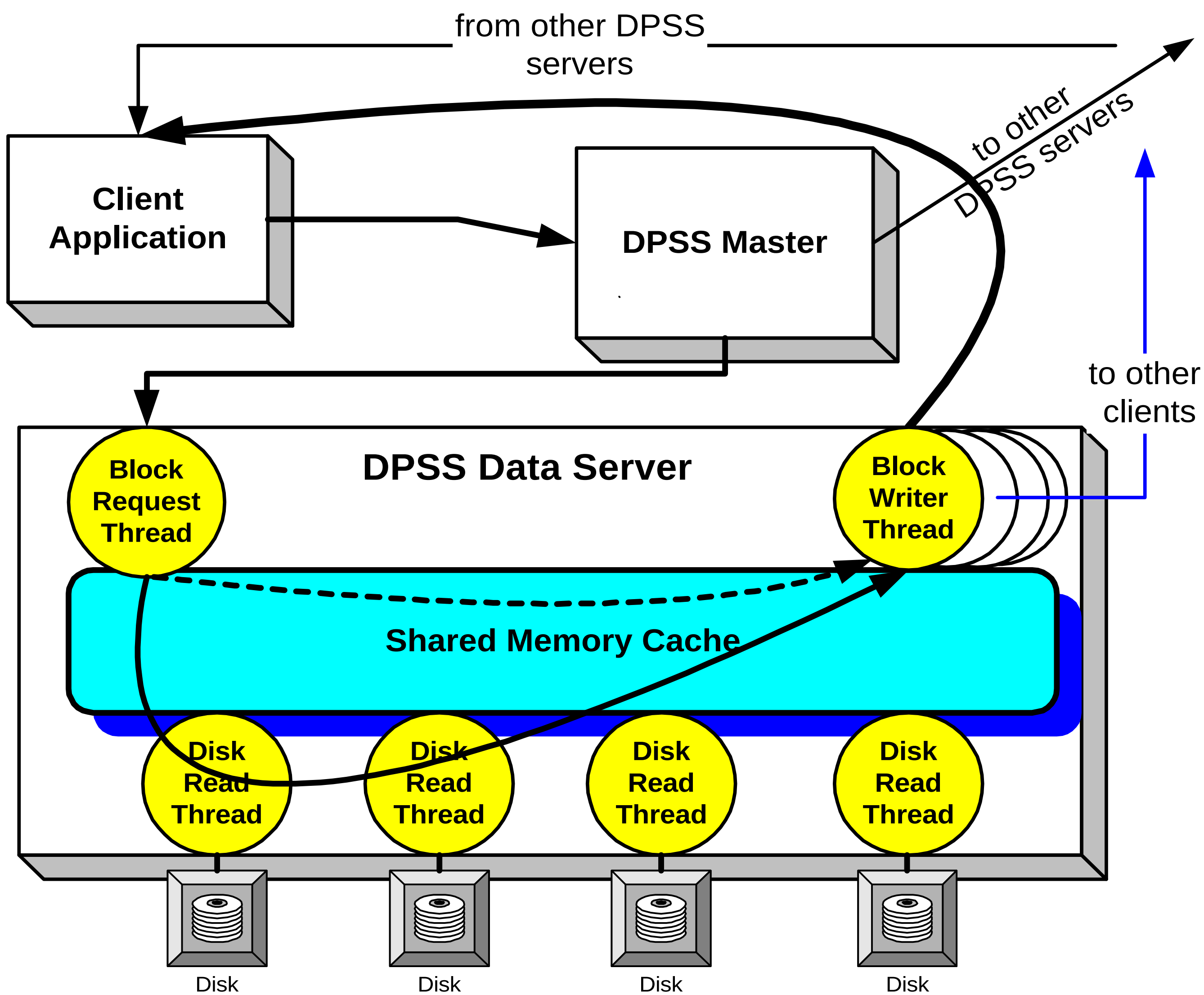
# DPSS Design

- support data-intensive applications
- provide very high data throughput
- parallelism at every level, including disk, SCSI bus, network, and server
- high-speed WAN aware
- scaleable throughput and capacity
- economical
  — use only low-cost commodity hardware components
- location transparency
  — location of DPSS servers is transparent to the application

# DPSS Architecture



# DPSS Server Architecture



# Typical DPSS implementation

- 4 - 5 UNIX workstations (e.g. Sun Ultra I0s, Pentium 400)
  - — 4 - 6 Ultra-SCSI disks on 2 SCSI host adapters
  - — a high-speed network interface (100BT, 1000BT, ATM)
- This configuration can deliver an aggregated data stream to an application at about 500 Mbits/s (62 MBy/s) using these relatively low-cost, “off the shelf” components by exploiting the parallelism of:
  - — five hosts,
  - — twenty disks,
  - — ten SCSI host adapters
  - — five network interfaces

# Sample DPSS Costs

- server host = Sun Ultra 10S or Pentium/Linux: $3-5K
  - throughput = 11 - 14 MB/sec
- disk = 16 GB Ultra-wide SCSI (Seagate): $900
  - might be able to use IDE drives with new PCI card that puts multiple IDE "master" devices on the same PCI card (16 GB IDE disk only $275); waiting for Linux driver
- Cost is mainly dominated by disk price

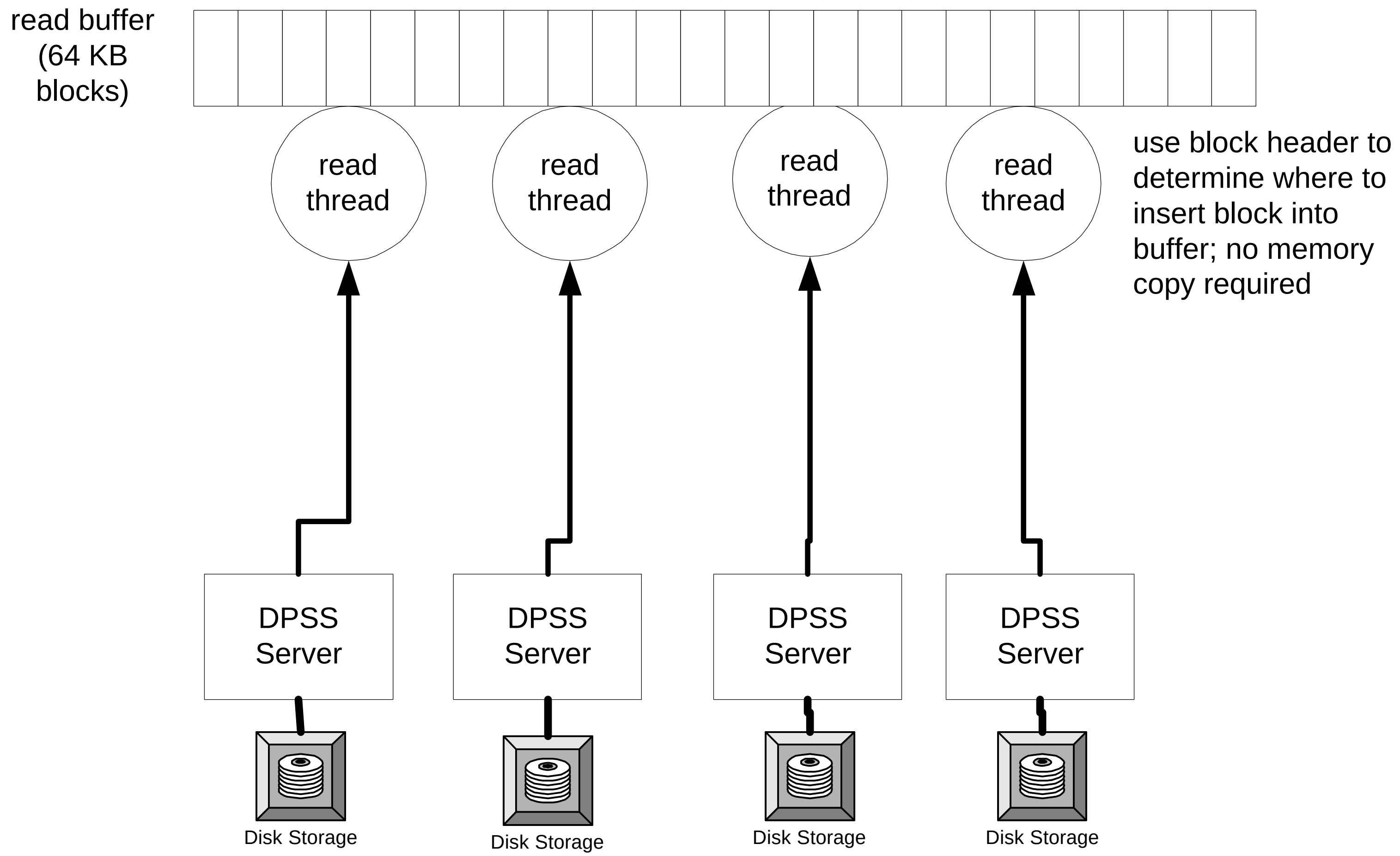
| Throughput | Capacity | Configuration | Cost |
|---|---|---|---|
| 10 MB/sec | 33 GB | 1 server, 2 disks | $6.3K |
| 50 MB/sec | 165 GB | 5 servers, 10 disks | $31.5K |
| 50 MB/sec | 1 TB | 5 servers, 64 disks | $80K |
| 100 MB/sec | 1 TB | 10 servers, 64 disks | $102 K |

# DPSS client API

- Modeled on Unix I/O
- C library with the following routines:
  - — dpssOpen(“x-dpss://hostname/setname”,mode)
  - — dpssAlloc()
  - — dpssRead()
  - — dpssWrite()
  - — dpssLseek()
  - — dpssClose()
- Read/Write calls have a thread per DPSS server
  - — client scales with number of servers

# dpssRead()

dpssRead(dpss_file_descriptor, char *buffer, int size) ;
ie: dpssRead(dpssfd, buffer, 4*1024*1024);



# Importance of TCP Buffer Tuning

- 45 Mbps WAN (latency = 41 ms), some congestion

| | |
|---|---|
| 8 KB data packets, 24 KB TCP buffers | 6.5 Mbps |
| 64 KB data packets, 350 KB TCP buffers | 15.6 Mbps |
| 2 sockets/threads, 64 KB data, 350 KB TCP buffers | 18 Mbps |

- OC12 (622 Mbps) WAN (latency = 45 ms), no congestion

| | |
|---|---|
| 8 KB data packets, 24 KB TCP buffers | 7 Mbps |
| 64 KB data packets, 4 MB TCP buffers | 350 Mbps |
| 2 sockets/threads, 64 KB data, 4 MB TCP buffers | 380 Mbps |

- Congested Internet Path (latency = 80 ms)

| | |
|---|---|
| 8 KB data packets, 24 KB TCP buffers | .8 Mbps |
| 64 KB data packets, 350 KB TCP buffers | .8 Mbps |
| 2 sockets/threads, 64 KB data, 350 KB TCP buffers | 1.6 Mbps |

# Importance of TCP Tuning

| Buffer Tuning | Network | throughput |
|---|---|---|
| Tuned for LAN (64 K) | LAN<br>WAN | 264 Mb/s (33 MB/s)<br>44 Mb/s (5.5 MB/s) |
| Tuned for WAN (512 K) | LAN<br>WAN | 152 Mb/s (19 MB/s)<br>112 Mb/s (14 MB/s) |
| Auto-tuning | LAN<br>WAN | 264 Mb/s (33 MB/s)<br>112 Mb/s (14 MB/s) |

**LAN = OC-12 (rtt = 1ms)**

**WAN = OC-3 (rtt = 44 ms)**

**OS: Solaris 2.6**

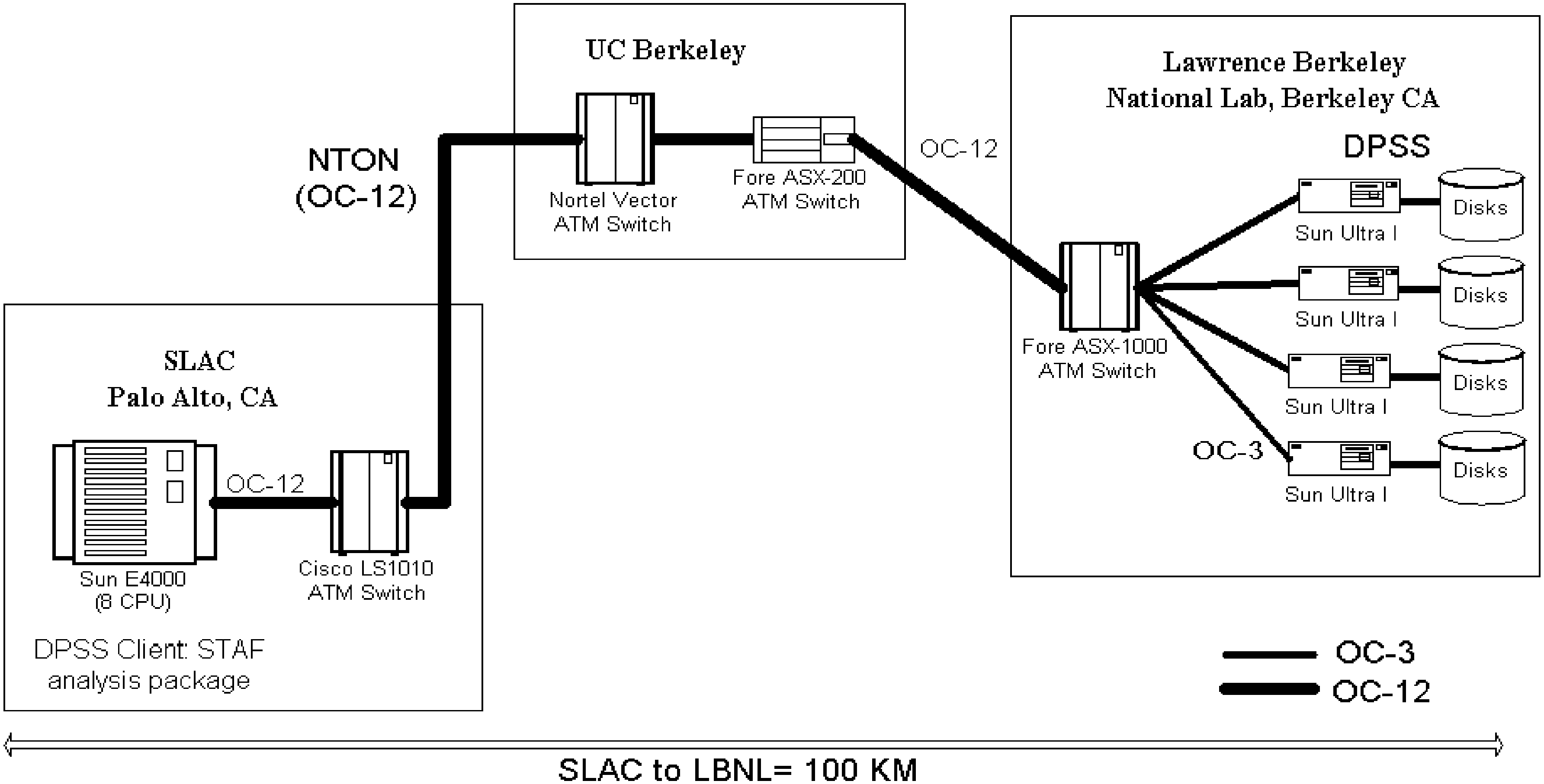
# Clipper Project

- Goals
  - — Develop technologies required for distributed data-intensive applications
  - — Apply to high energy physics (HEP) data analysis
- Participants
  - — Argonne National Laboratory
  - — Lawrence Berkeley National Laboratory
  - — Stanford Linear Accelerator Center (SLAC)

# Clipper Technologies

- Distributed Parallel File System
  - — High-speed, low-cost data cache
- Globus
  - — End-to-end resource management
- ESnet and NTON
  - — OC12 networks
- HPSS and Objectivity
  - — Data archives

# LBNL / SLAC HENP Application Experiment




Achieved 57 MBytes/sec (450 Mbits/sec) of user data delivered to the application

# LBNL/SLAC Performance Results

- Experiments conducted over NTON, July, 1998
  — Application network was IP over OC-12 (622 Mbit/sec) ATM.
- An application (STAF: Physics Analysis package) running on a Sun Enterprise-4000 SMP at SLAC (Palo Alto) read data from four distributed disk servers at LBNL (Berkeley), parsed the XDR records and placed the data into the application memory

# LBNL/SLAC Performance Results

- Each DPSS server transfer rate is 14.25 MBytes/sec
- OC-12 receiver was able read data from 4 servers in parallel at 57 Mbytes/sec
  - — this is the rate of data delivered from datasets in a distributed cache to the remote application memory, ready for analysis algorithms to commence operation.
- This is equivalent to 4.5 TeraBytes/day!

# NetLogger: Distributed System Performance Analysis Tools

# Overview

- The Problem
  — When building distributed systems, we often observe unexpectedly low performance
    - the reasons for which are usually not obvious
  — The bottlenecks can be in any of the following components:
    - the applications
    - the operating systems
    - the disks or network adapters on either the sending or receiving host
    - the network switches and routers, and so on
- The Solution:
    - Highly instrumented systems with precision timing information and analysis tools

# Bottleneck Analysis

- Distributed system users and developers often assume the problem is network congestion
  - — This is often not true
- In our experience tuning distributed applications, performance problems are due to:
  - — network problems: 40%
  - — host problems: 20%
  - — application design problems/bugs: 40%
    - 50% client , 50% server
- Therefore it is equally important to instrument the applications

# NetLogger Toolkit

- We have developed the NetLogger Toolkit
  - A set of tools which make it easy for distributed applications to log interesting events at every critical point
  - NetLogger also includes tools for host and network monitoring

- The approach is novel in that it combines network, host, and application-level monitoring to provide a complete view of the entire system

# Why “NetLogger”?

- The name “NetLogger” is somewhat misleading
  — Should really be called: “Distributed Application, Host, and Network Logger”

- “NetLogger” was a catchy name that stuck

# NetLogger Components

- NetLogger Toolkit contains the following components:
  - — NetLogger message format
  - — NetLogger client library
  - — NetLogger visualization tools
  - — NetLogger host/network monitoring tools

- Additional critical component for distributed applications:
  - — NTP (Network Time Protocol) or GPS clock is required to synchronize the clocks of all systems

# NetLogger Message Format

- We are using the IETF draft standard Universal Logger Message (ULM) format
- Sample NetLogger ULM event:

```
DATE=19980430133038.055784 HOST=foo.lbl.gov
  PROG=testprog LVL=Usage NL.EVNT=SEND_DATA
  SEND.SZ=49332
```

  — This says program named *testprog* on host *foo.lbl.gov* performed event named SEND_DATA, size = 49332 bytes, at the date/time given

- User-defined data elements (any number) are used to store information about the logged event - for example:
  - NL.EVNT=SEND_DATA SEND.SZ=49332
    — the number of bytes of data sent

# NetLogger API

- NetLogger Toolkit includes application libraries for generating NetLogger messages
  - Can send log messages to:
    - file
    - host/port (*netlogd*)
    - syslogd
    - memory, then one of the above
- C, C++, Fortran, Java, Perl, and Python APIs are currently supported

# Sample NetLogger Use

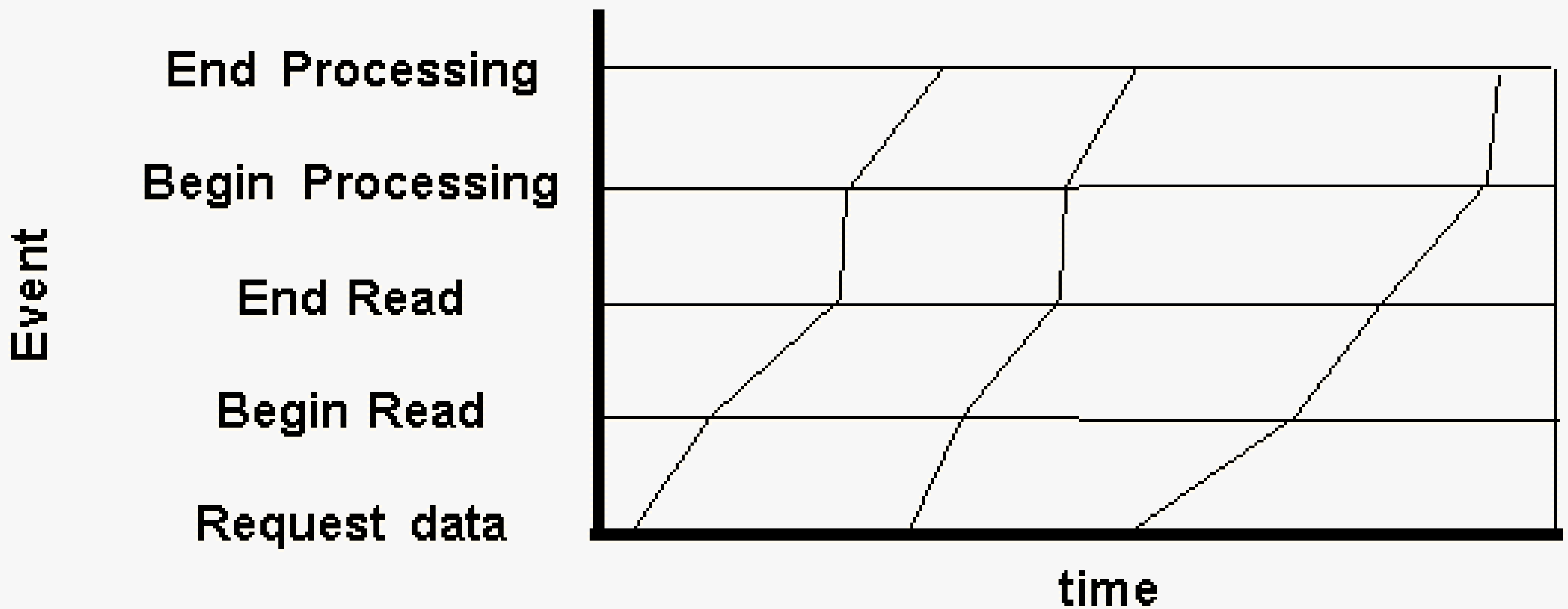
```
lp = NetLoggerOpen(method, progname, NULL,
                   hostname, NL_PORT);

while (!done)
{
      NetLoggerWrite(lp, "EVENT_START",
                     "TEST.SIZE=%d", size);

      /* perform the task to be monitored */
      done = do_something(data, size);

      NetLoggerWrite(lp, "EVENT_END");
}
NetLoggerClose(lp);
```

# NetLogger Host/Network Tools

- Wrapped UNIX network and OS monitoring tools to log “interesting” events using the same log format
  - — *netstat* (TCP retransmissions, etc.)
  - — *vmstat* (system load, paging, etc.)
  - — *iostat* (disk activity)
  - — *ping* (network latency)
  - — *snmp_get* (switch/router stats)

- These tools have been wrapped with Perl or Java programs which:
  - — parse the output of the system utility
  - — build NetLogger messages containing the results

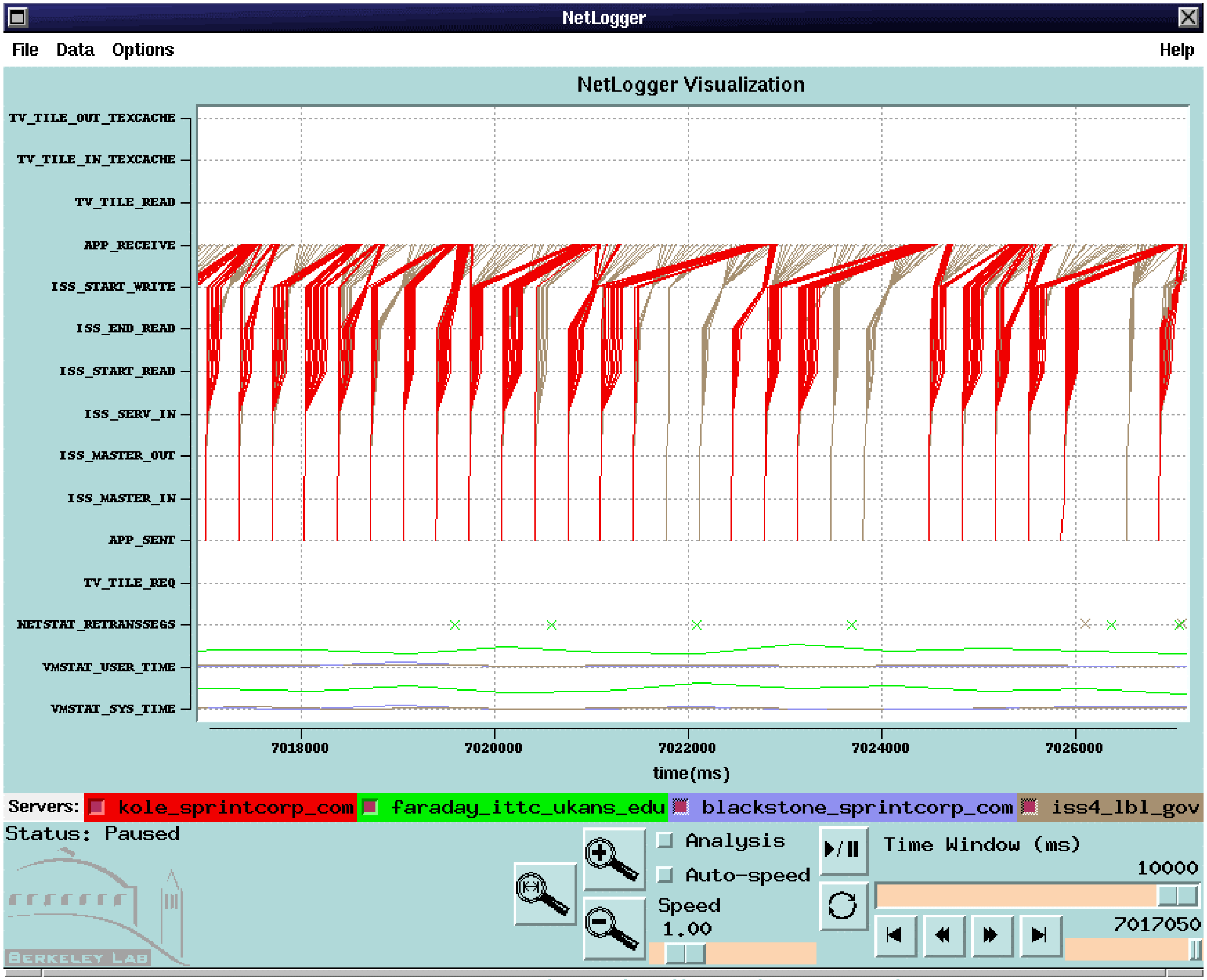
# NetLogger Event “Life Lines”



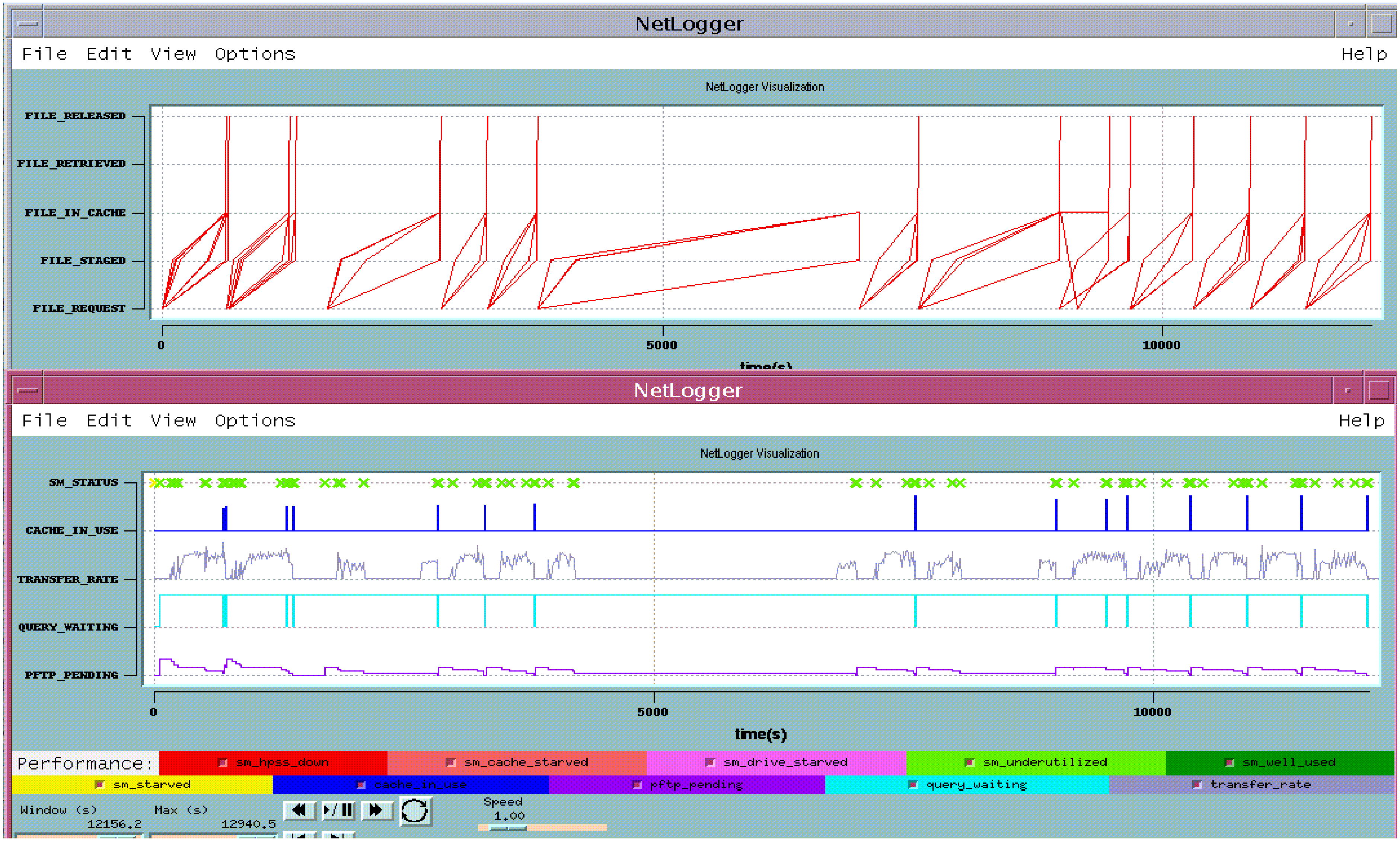
# NetLogger Visualization Tools

- Exploratory, interactive analysis of the log data has proven to be the most important means of identifying problems
  — this is provided by *nlv* (NetLogger Visualization)
- *nlv* functionality:
  — can display several types of NetLogger events at once
  — user configurable: which events to plot, and the type of plot to draw (lifeline, load-line, or point)
  — play, pause, rewind, slow motion, zoom in/out, and so on
  — *nlv* can be run post-mortem or in real-time
    - real-time mode done by reading the output of *netlogd* as it is being written

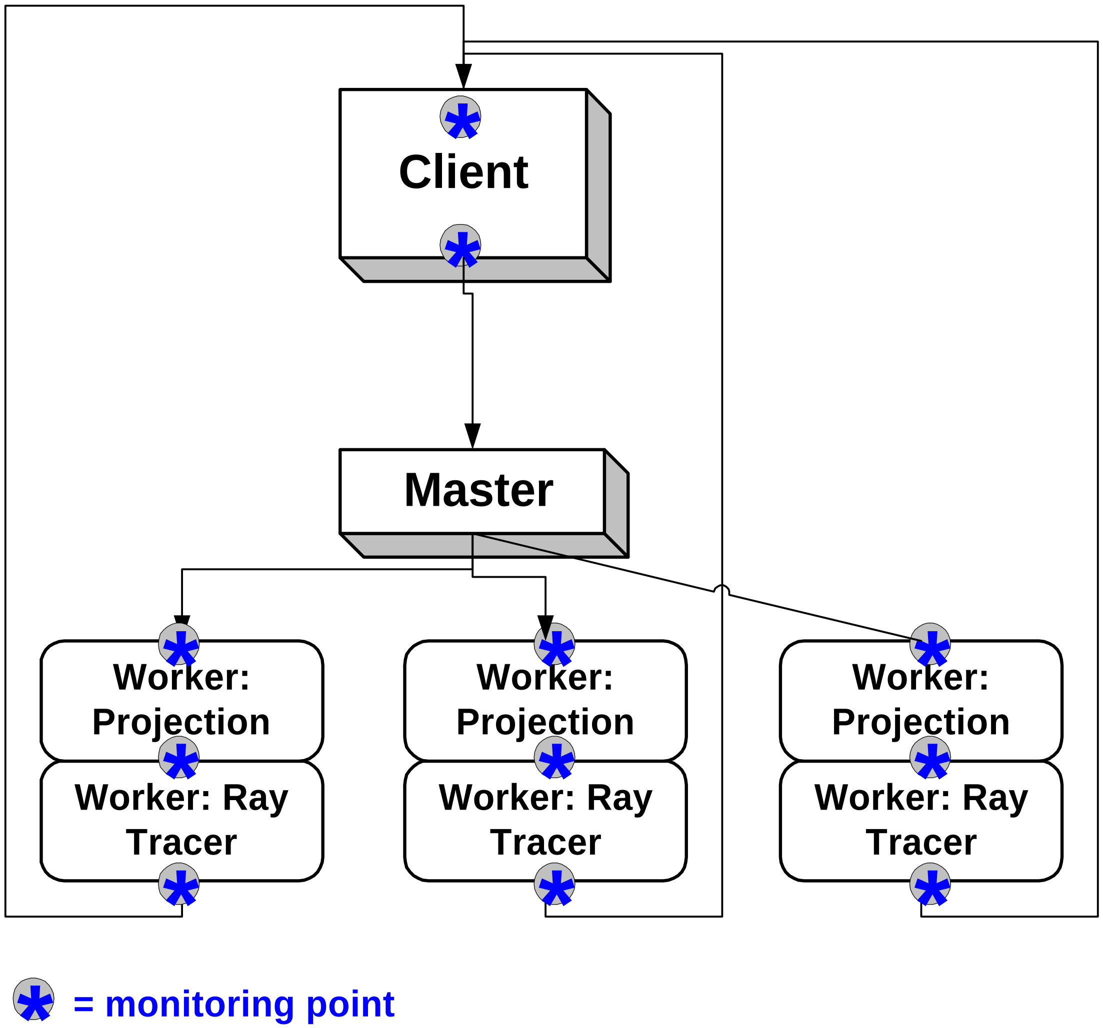
# NLV with lifeline, load-line, and point events



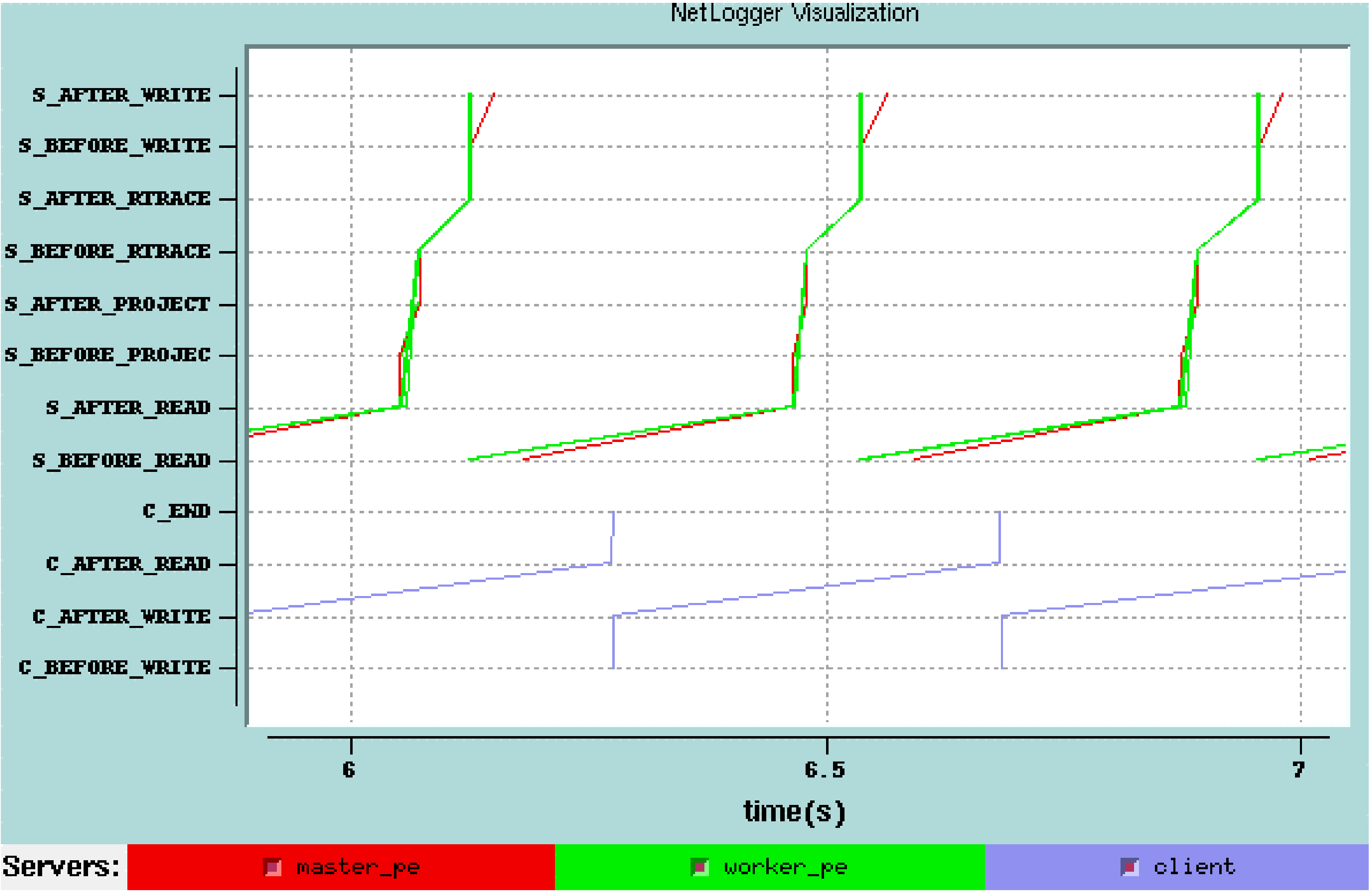
# NLV Example: System Performance



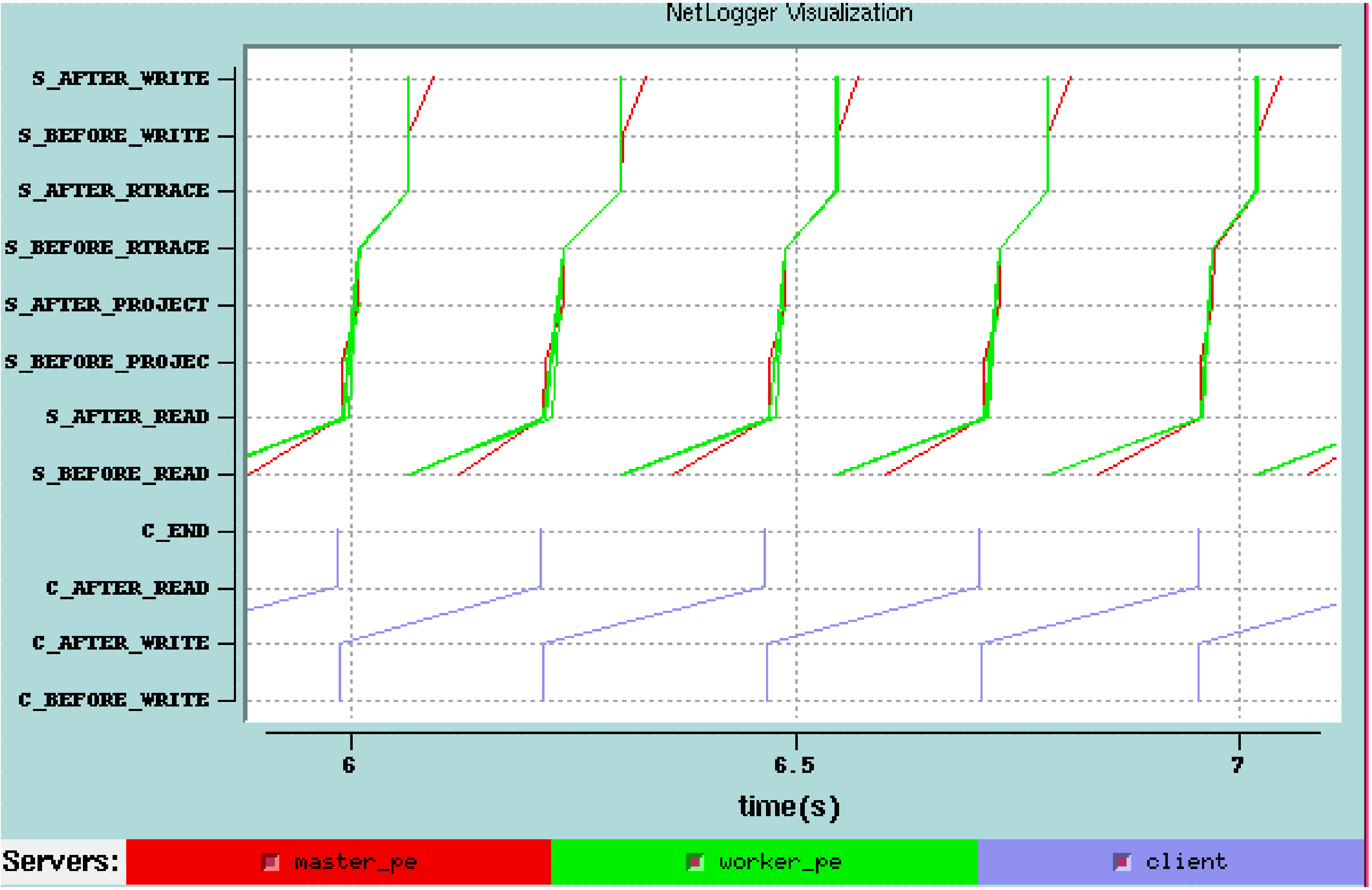
# Parallel Ray Tracing (Radiance): Instrumentation Points



# NetLogger Radiance Results: Before Tuning



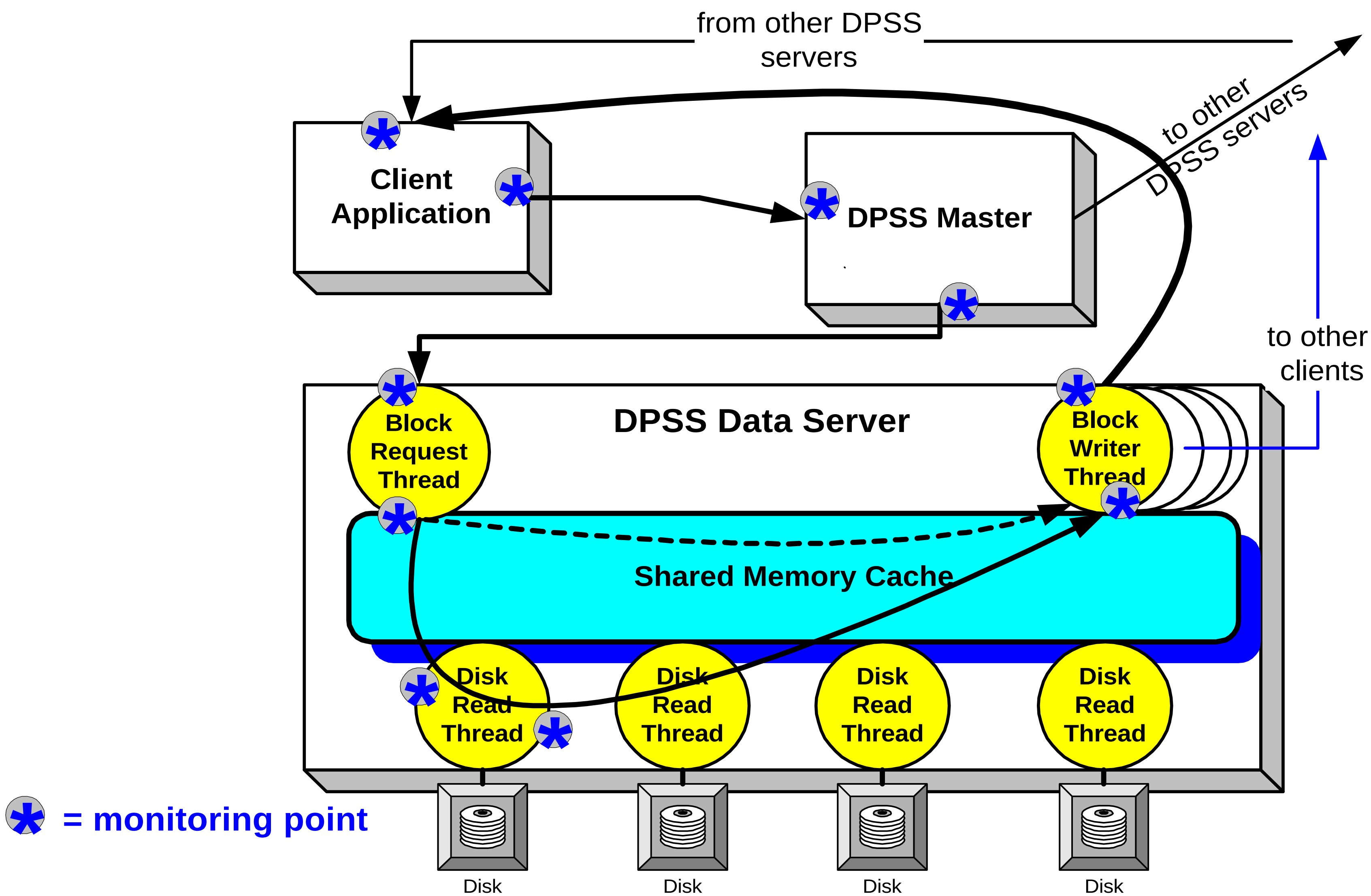
# NetLogger Radiance Results: After Tuning



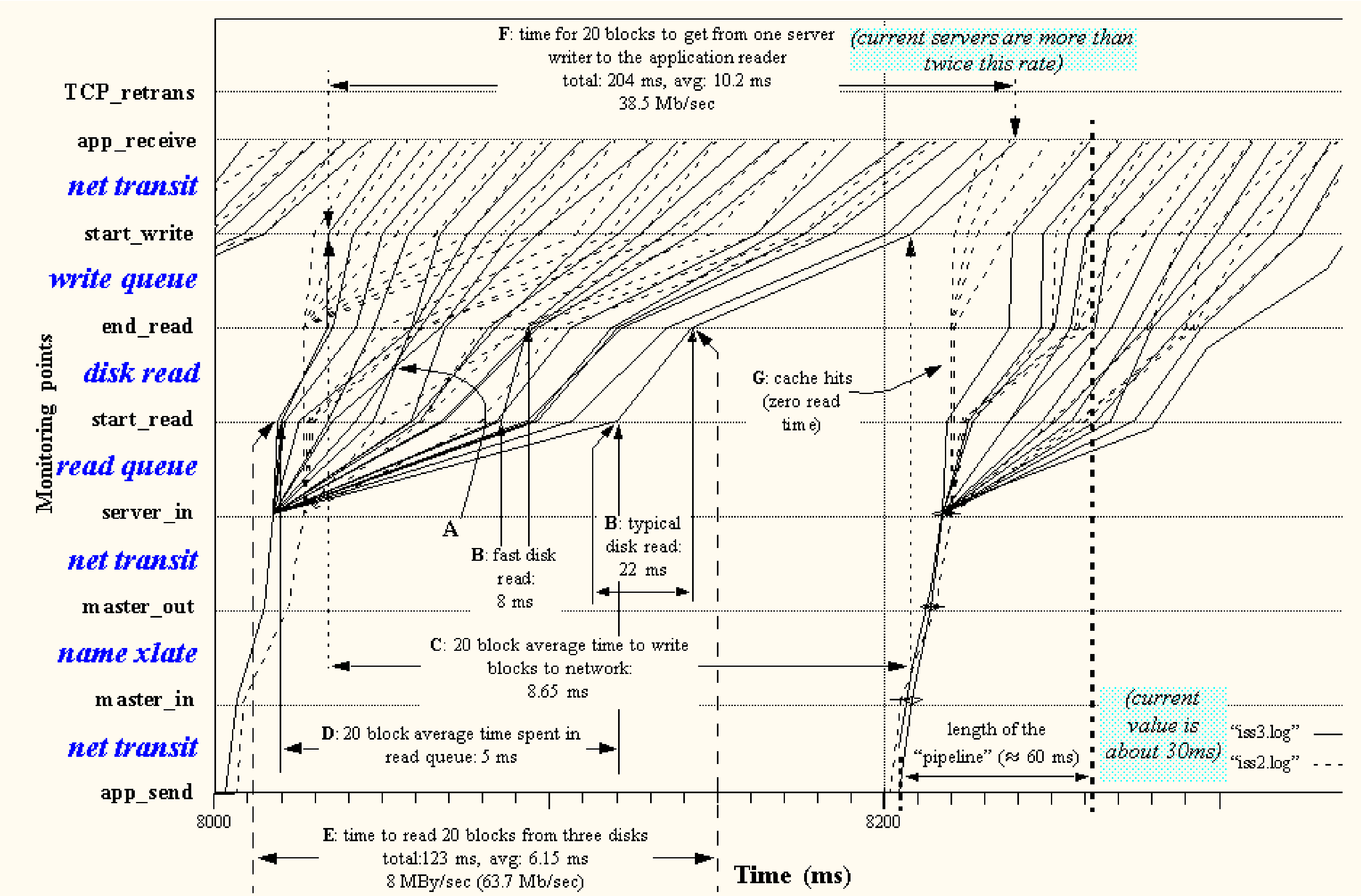
# Example 2: Parallel Data Block Server

- NetLogger was used for performance tuning and debugging of the DPSS and the WAN

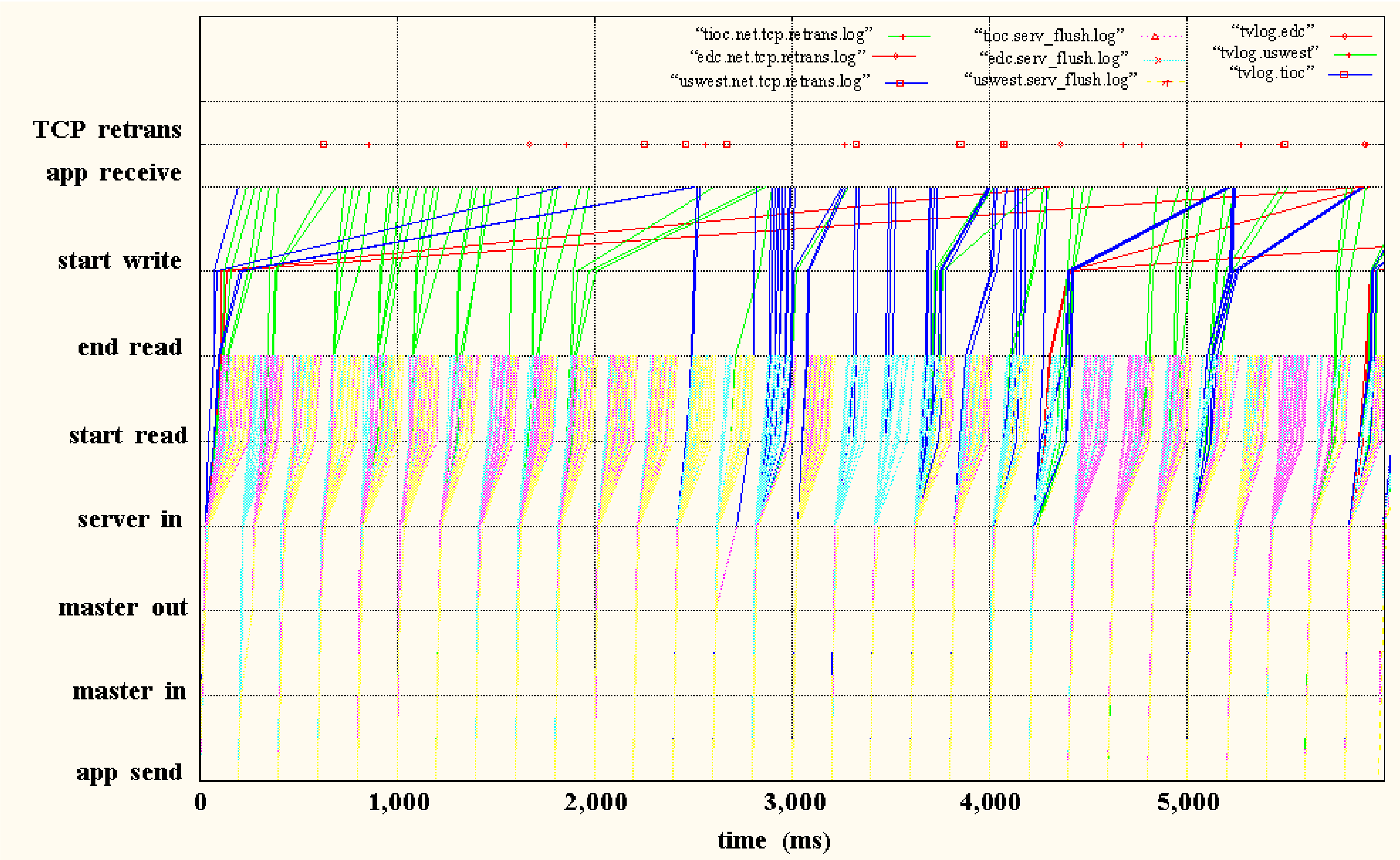
# DPSS Instrumentation



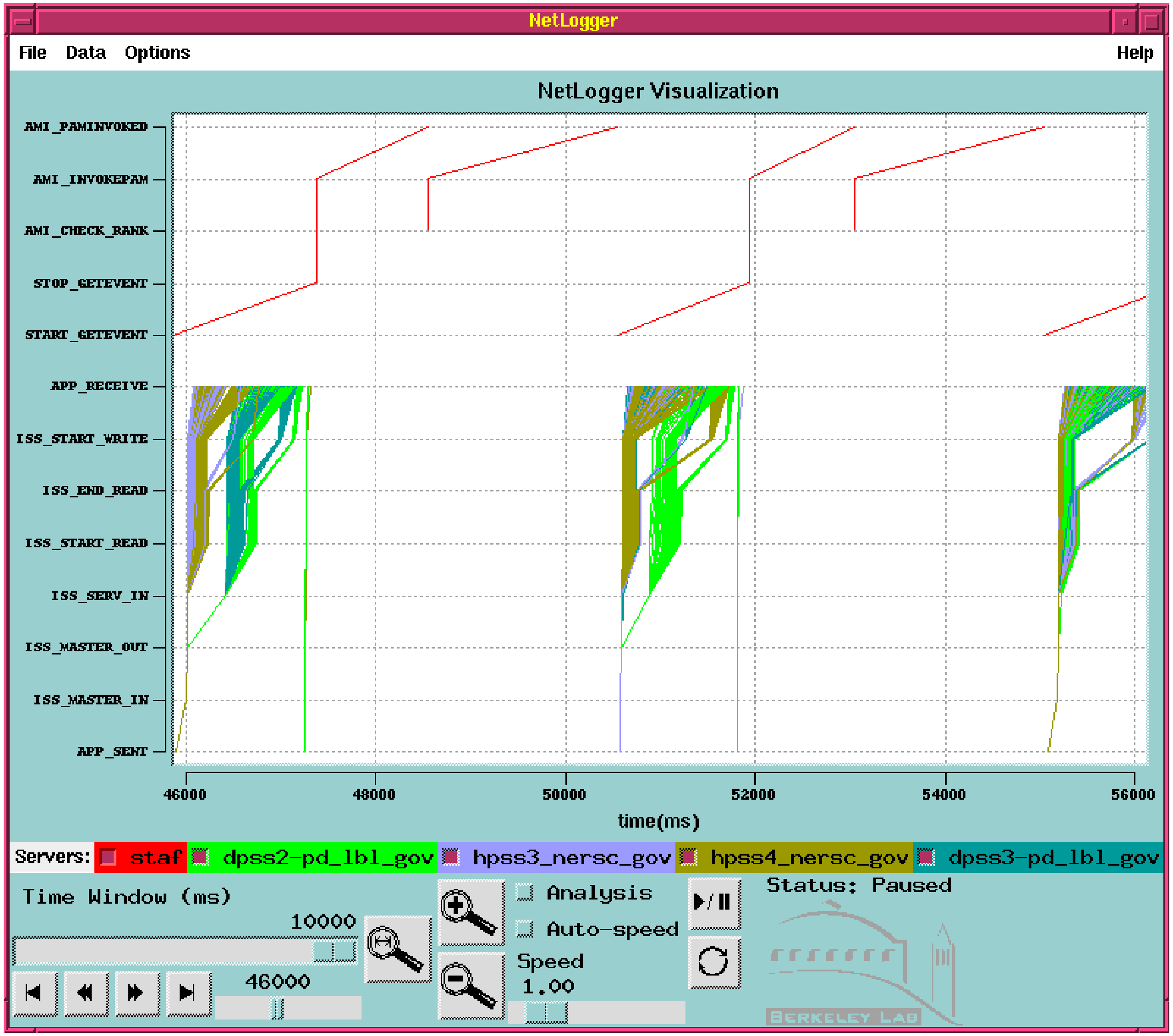
# NetLogger Results for the DPSS



# NetLogger Results for the DPSS over a WAN



# NLV of DPSS with a HENP client



# Data Grids

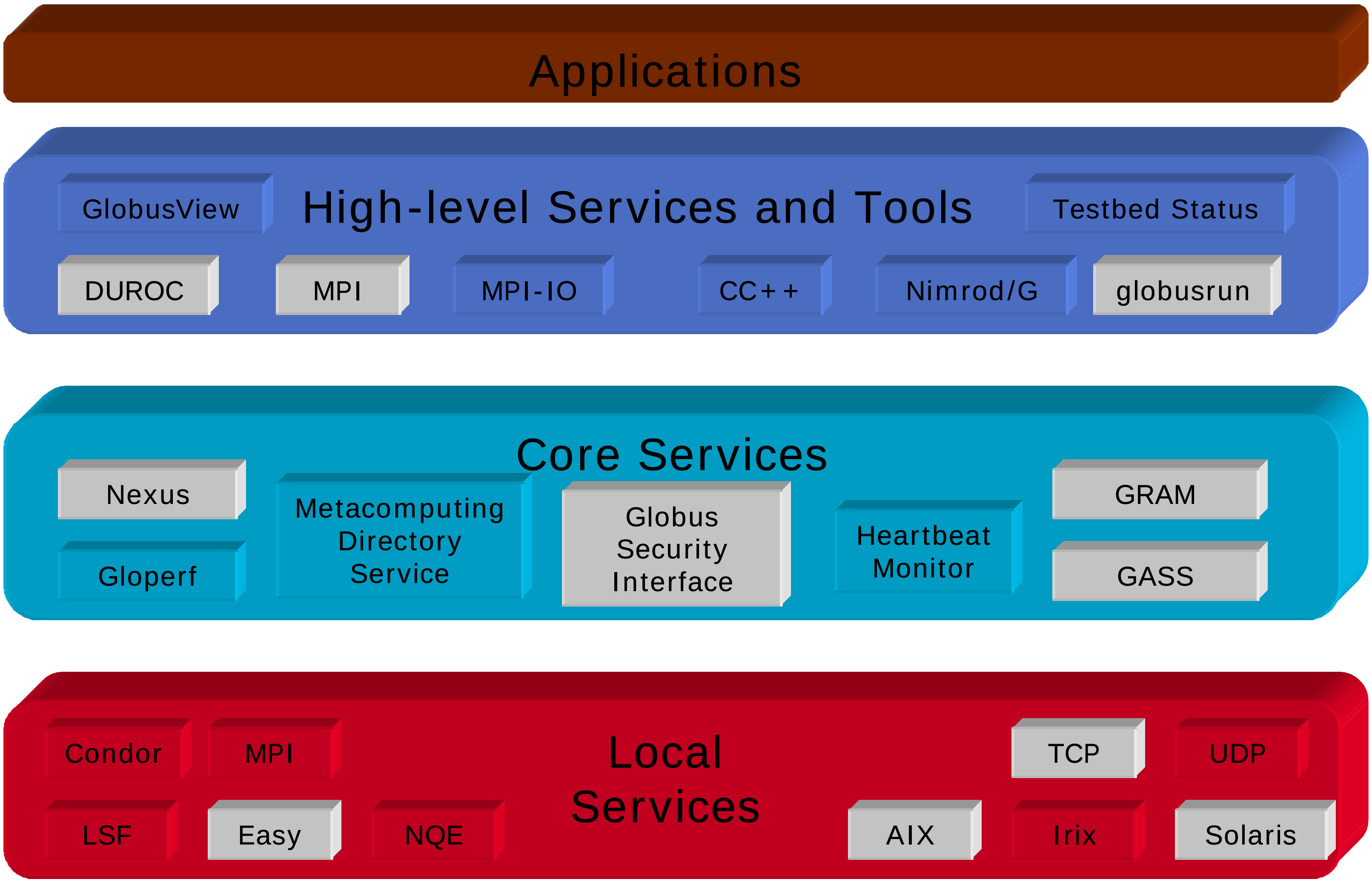
# Computational/Data Grids

- Grid / Computational Grid:
  - — The integration of various approaches used for integrating dispersed resources
  - — analogy with the grid that supplies ubiquitous access to electric power.
  - — Basic grid services are those that locate, allocate, coordinate, utilize these resources
- Data Grid:
  - — services for handling remote access to large data sets in a grid environment
- Working with Globus group at ANL to build “Data Grid” services

# Grid Services

- Grid services include:
  - — authentication
  - — resource location
  - — resource allocation
  - — configuration
  - — communication
  - — remote file access
  - — fault detection
  - — executable management

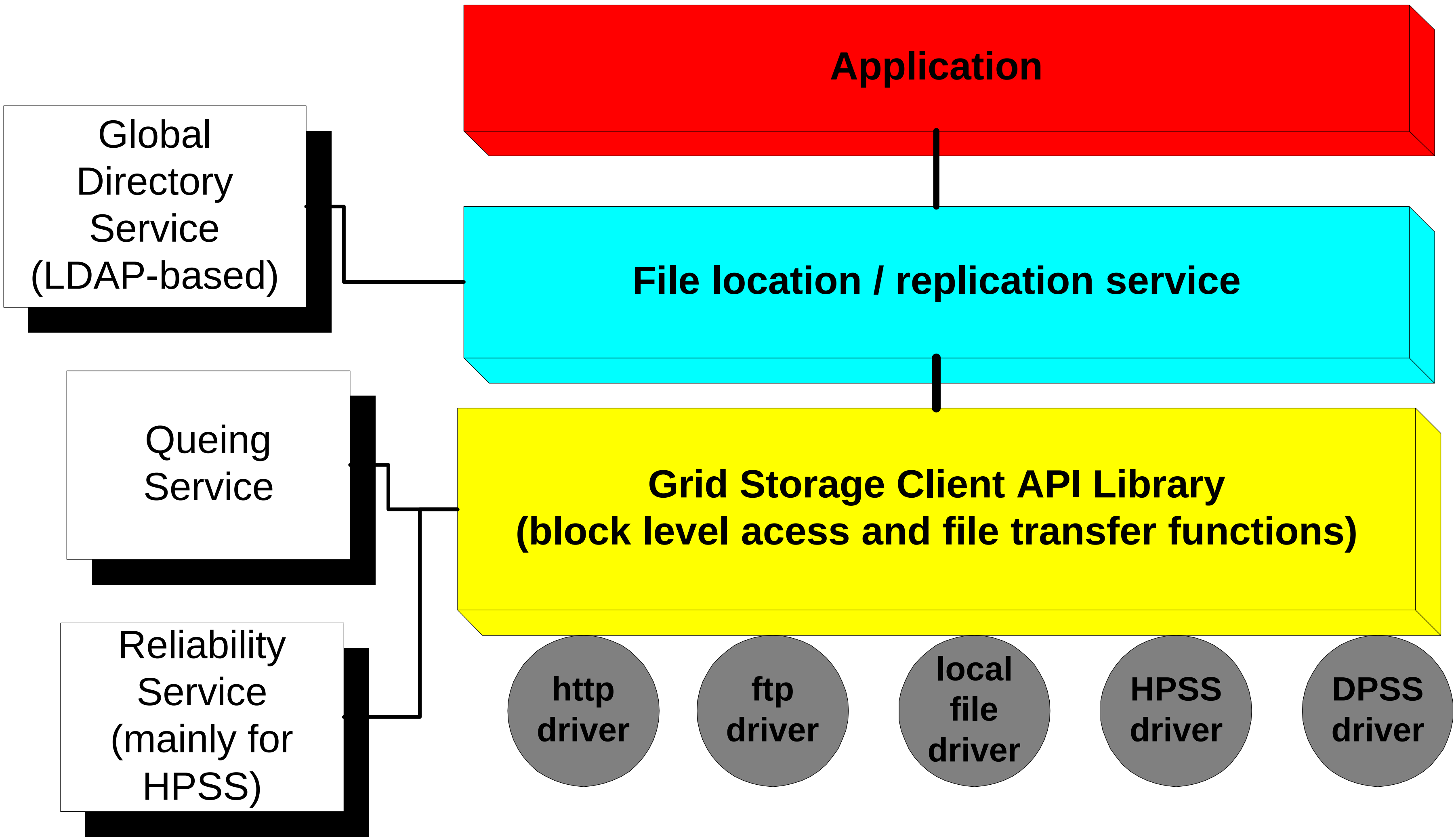
# Layered Architecture (Globus)



# Data Grids

- We use the term “Data Grid” to describe additional services that are unique to data intensive grid applications. These services include:
  - — data migration tools that are optimized for transferring large data sets over WANs
  - — data set discovery and replication tools
  - — data caches / cache management services
  - — metadata service:
    - global name space for data archived at multiple sites
    - file access control
    - file collections (data set = many files)
    - replica management

# Data Grid Architecture



# Storage Client API

- Storage client API
  - — simplifies the implementation of Grid applications by providing a uniform interface to several types of storage systems
  - — The interface is defined so that implementations can exploit techniques to achieve high performance, I.e.:
    - network striping
    - parallel I/O
    - network protocol tuning
- Other Components
  - — *metadata catalog:* stores metadata about each file
  - — *replica catalog:* maps a logical file name to one or more file instance names

# Data Grid Applications

- DOE NGI Applications that will be early users of the Data Grid services
  - — Earth Grid Project
  - — Particle Physics Data Grid (PPDG) Project
    - (Cal Tech, SLAC, LBNL, and many others)

- See: http://www-didc.lbl.gov/NGI

# Another New Project: Grid Monitoring Service

- Our goal is to deploy NetLogger-like host and network monitoring as a standard “grid service”
- Before this can happen, we need to define:
  — archive system
    - standard interface to archive system (probably LDAP?)
  
  — network monitoring system
    - Surveyor, NWS, pingER, OCXmon, GloPerf,…
    - SNMP-based?
- Grid Forum “end to end monitoring” working group
  — http://www.gridforum.org/
- DOE NGI monitoring / instrumentation working group
  — goal is to deploy something by the end of the year

# Summary: How to Achieve High Throughput over a WAN

- Over the past several years we have learned that the following is needed to obtain good TCP throughput over WAN's:
  — Use multiple TCP sockets for the data stream
    - possibly as many as 1 per disk
  — Use a separate thread for each socket
  — Use large block sizes (at least 64 KB)
  — Read and write at least 100 blocks at a time, if possible
  — Use the optimal TCP send and receive buffer sizes
    - too large or too small adversely affects performance
  — Avoid unnecessary data copies
    - manipulate pointers to data blocks instead

# For more information

- http://www-didc.lbl.gov/DPSS
- http://www-didc.lbl.gov/NetLogger
- http://www-didc.lbl.gov/NGI
- http://www.globus.org/

- email: bltierney@lbl.gov