Distributed Storage Caches and Distributed System Performance Analysis Tools

Data Intensive Computing Tutorial

SuperComputing '98

Brian L. Tierney (bltierney@lbl.gov)
Future Technologies Group

Lawrence Berkeley National Laboratory
Berkeley, CA 94720

Outline

- Overview of Distributed Storage
- Data Handling Architecture
- Distributed Storage Applications
  - Terrain Visualization
  - Medical Imaging
  - HENP
- A Distributed Storage System: The DPSS
- Next Generation Architectures
- Distributed Systems Performance Analysis
  - NetLogger Components
  - NTP
- Sample Results

Distributed Storage

Why is distributed storage important for Data Intensive Computing?

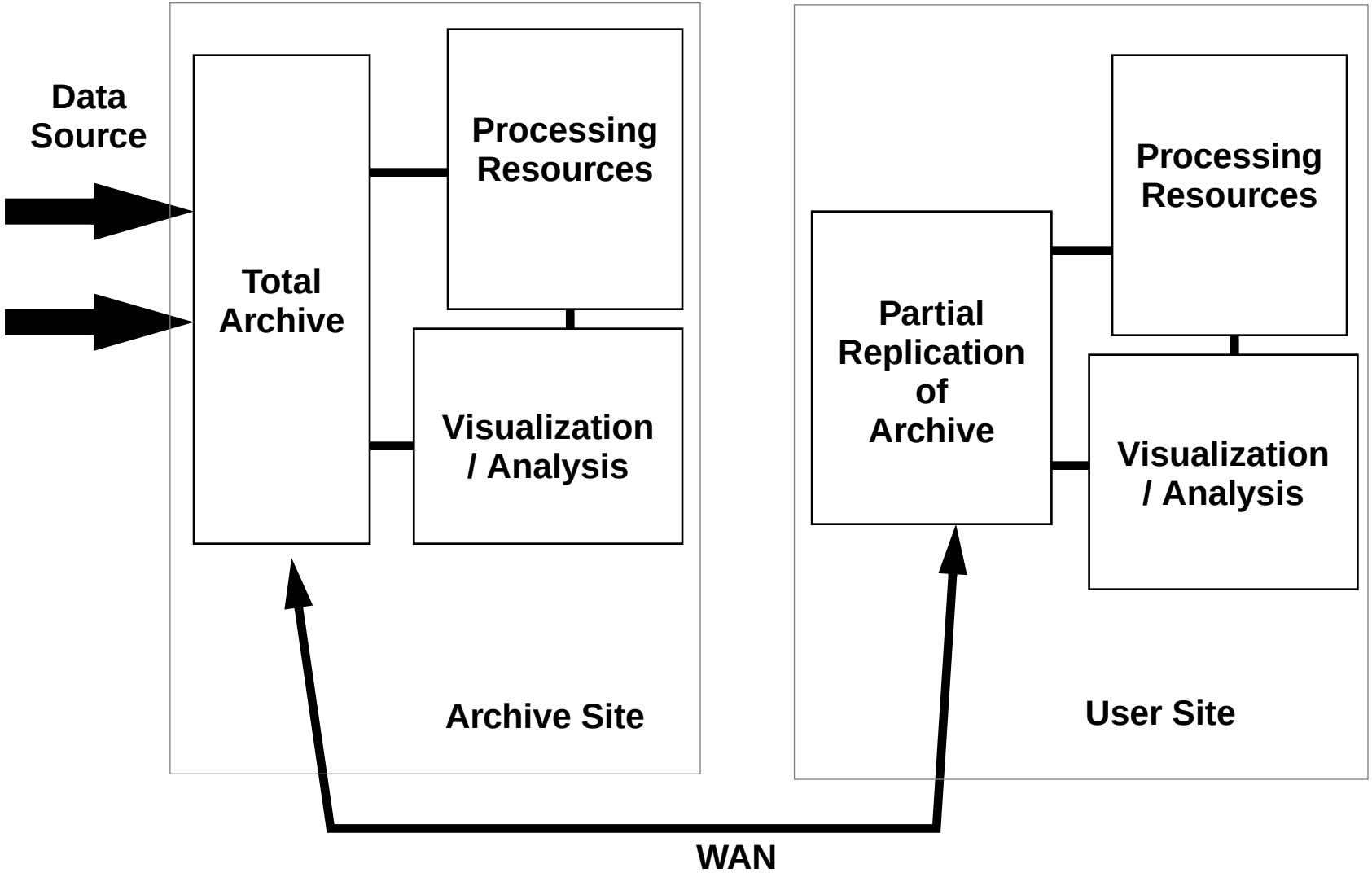
- **Researchers often are not at the same location as the data source**
- **Compute cycles are often not at the same location as the data source or the data archive**

Other Advantages of Distributed Storage:

- sharing of resources
- fault tolerance / load balancing through replicated data at multiple sites, where a fault might be:
  - host failure
  - disk failure
  - network failure
  - software fault
  - network congestion
  - excessive CPU load
- added flexibility: provides the ability to move the data to the compute cycles, or move the compute cycles to the data, depending on network speed

Distributed Storage

Remote Access to Large Data Archives



Example Architecture for On-line Data Sources

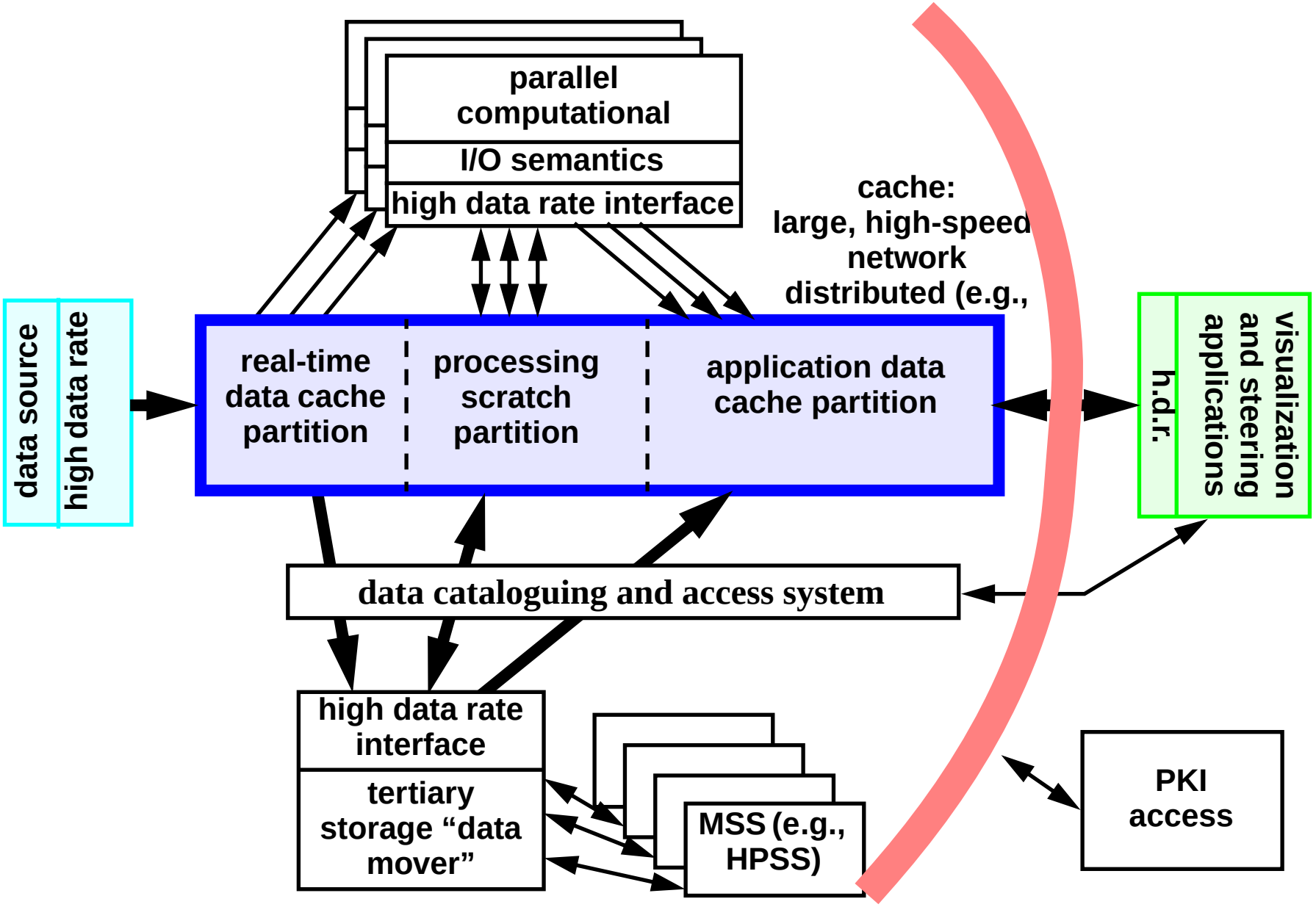
A prototype architecture has evolved from our experiences:

- distributed satellite image processing in MAGIC testbed
- on-line angiography (x-ray video) systems for Kaiser Hospital
- simulated on-line HENP detectors

Key features of the architecture:

- very high-speed cache that is distributed, scalable, and dynamically configurable
- common, low-level, high data rate interface that supports various application I/O semantics
- high-speed tertiary storage interface
- data cataloguing and access system
- distributed management of strong access control

Example Architecture



A Prototype High Volume, High Data Rate, Data Analysis Architecture

Example Architecture

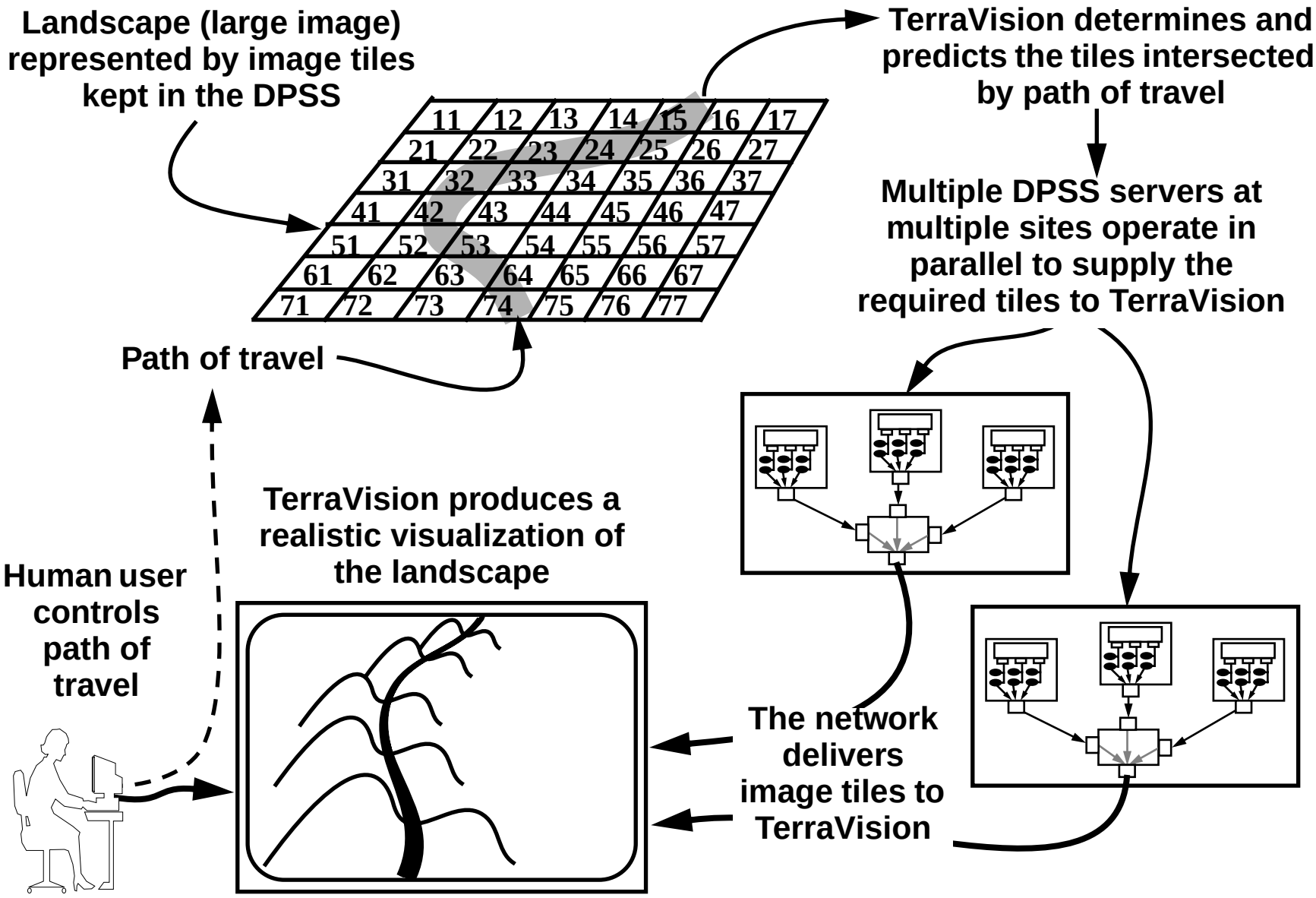
- **Advantages of this architecture:**
  - **first level processing can be done using resources at the collaborators sites (this type of experiment typically involves several major institutions)**
  - **large tertiary storage systems exhibit substantial economies of scale, and so using a large tertiary storage system at, say, a supercomputer center, should result in more economical storage, better access (because of much larger near-line systems - e.g. lots of tape robots) and better media management.**

Distributed Storage Applications

Background:

- **We first developed a distributed storage system, called the DPSS (Distributed Parallel Storage System) as part of the DARPA-sponsored MAGIC Gigabit Network Testbed (see: http://www.magic.net).**
- **The prototype high-speed application for this system was TerraVision, developed at SRI.**
- **TerraVision uses tile images and digital elevation models to produce a 3D visualization of landscape.**

Distributed Storage Applications



TerraVision and the DPSS Cooperate to Visualize Landscape

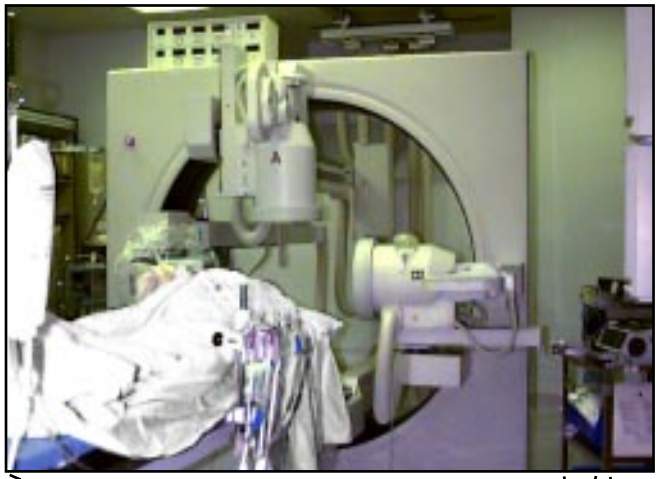
Medical Imaging Data

- The DPSS was used with an LBNL/Kaiser Permanente collaboration focused on connecting remote, on-line instrument systems to “real-time” digital libraries.
- System characteristics:
  - automatic generation of metadata
  - automatic cataloguing of the data and the metadata as the data is received
  - transparent management of tertiary storage systems where the original data is archived (via Unitree NFS interface)
  - facilitation of cooperative research by providing specified users at local and remote sites immediate as well as long-term access to the data

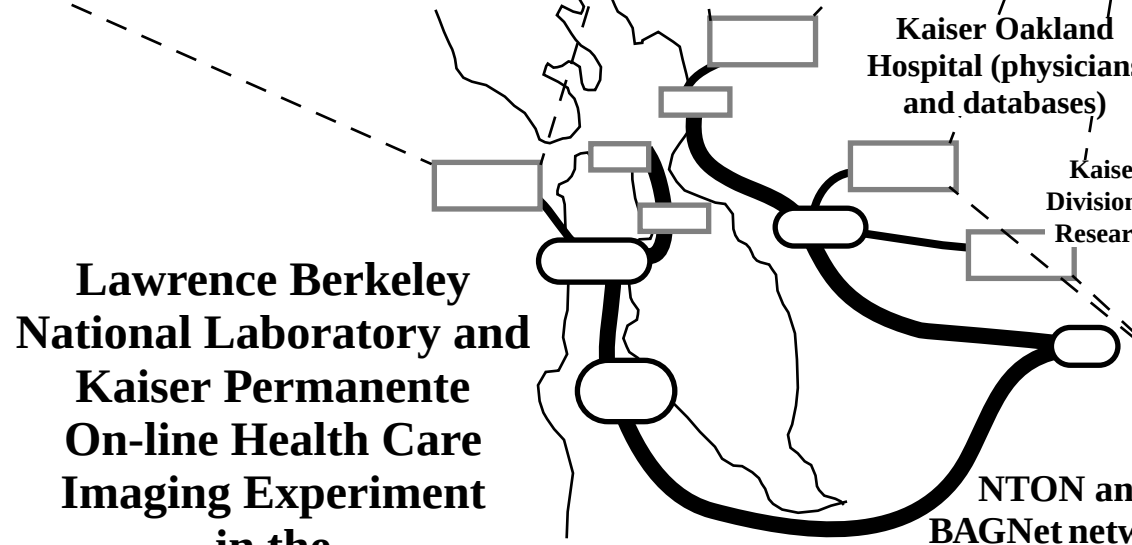
- **System characteristics (cont.):**
  - **mechanisms to incorporate the data into other databases or documents**
  - **cardio-angiography data was collected directly from a Phillips scanner in the San Francisco Kaiser hospital Cardiac Catheterization Laboratory — when the data collection for a patient is complete (about once every 20–40 minutes), 500–1000 megabytes of digital video data was sent across the ATM network to the WALDO system at LBNL which makes the data available to physicians in other Kaiser hospitals**
  - **this automated process goes on 8–10 hours a day**

Distributed Storage Applications

Kaiser San Francisco Hospital Cardiac Catheterization Lab (digital video capture)



LBNL WALDO server and DPSS distributed cache for data processing, cataloguing, and storage



Kaiser Oakland Hospital (physicians and databases)

Kaiser Division of Research

Lawrence Berkeley National Laboratory and Kaiser Permanente On-line Health Care Imaging Experiment in the San Francisco Bay Area

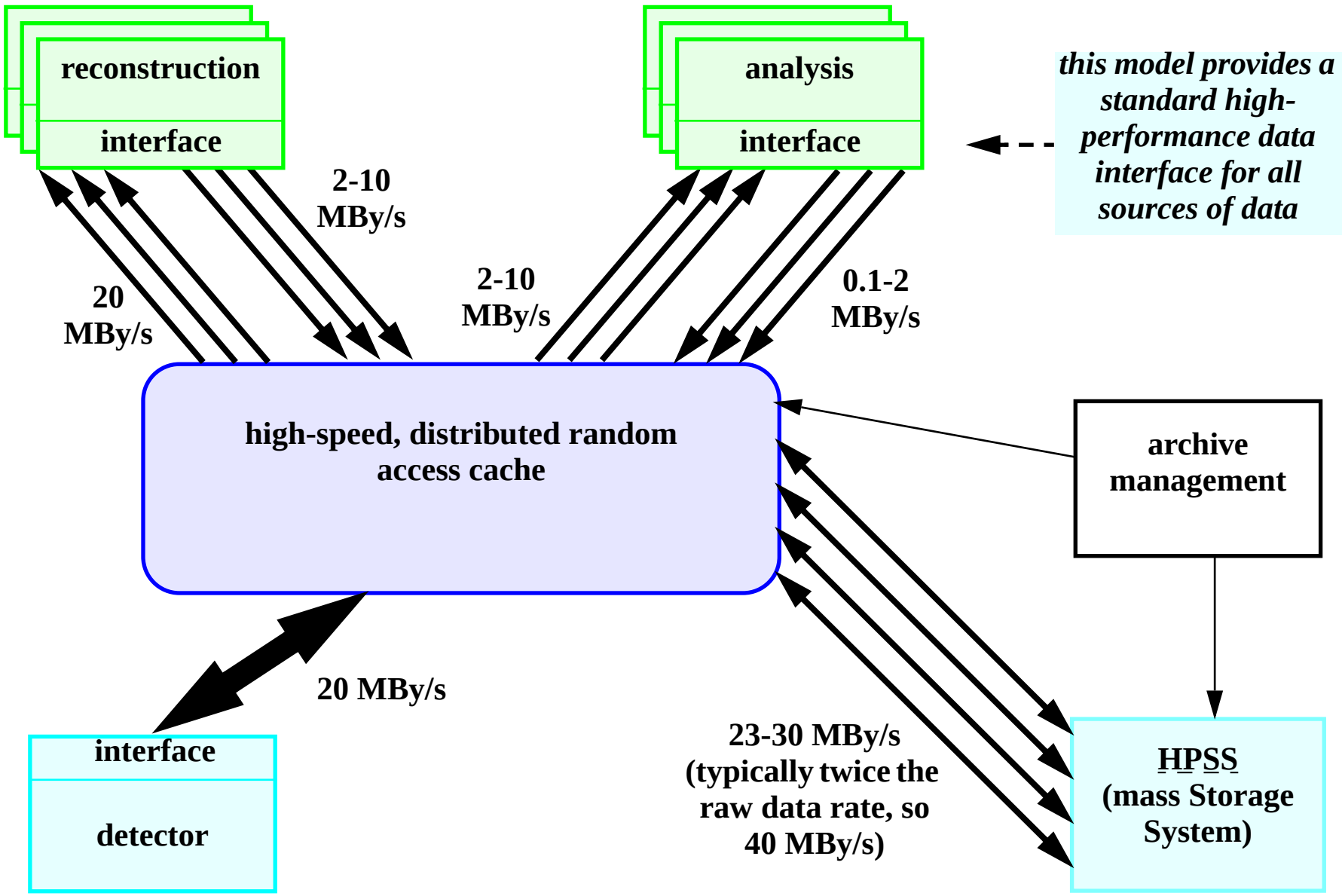
NTON and BAGNet network testbeds

Automatically generated user interfaces providing indexed access to the large data objects (the X-ray video) and to various derived data.

High Energy and Nuclear Physics Data

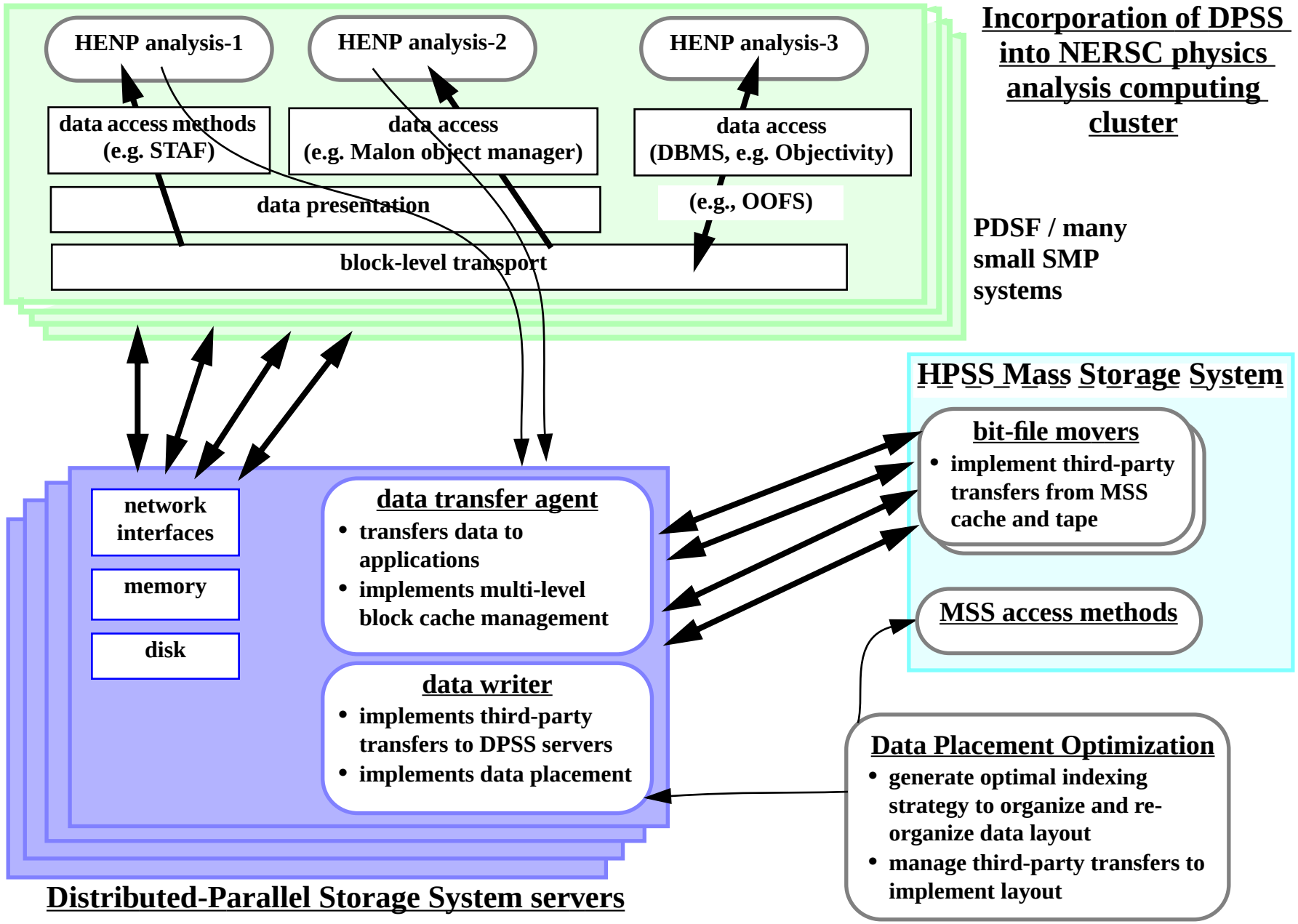
- Data source: The STAR detector at RHIC (Brookhaven National Lab).
  - This detector puts out a steady state data stream of 20-40 MBytes/second.
- This application requires a data handling architecture capable of supporting the processing and storage of over 2 TB / day:

Distributed Storage Applications



STAR data flow characteristics and prototype data handling model

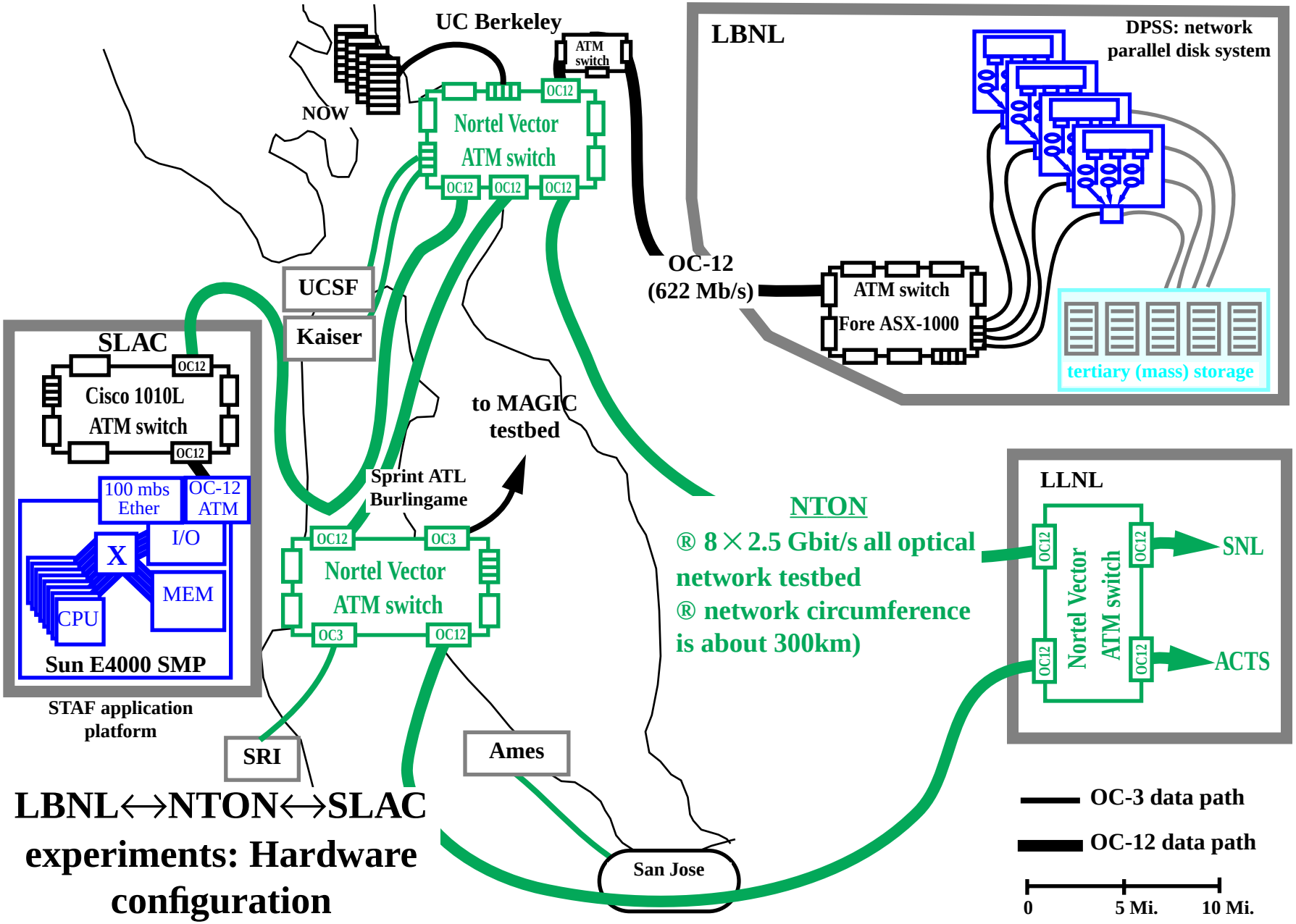
Distributed Storage Applications



Current STAF/DPSS Performance

- A recent set of experiments were conducted over the National Transparent Optical Network testbed — eight 2.4 gigabit/sec data channels around the San Francisco Bay.
- The application network was IP over OC-12 (622 Mbit/sec) ATM.
- An application 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.

Distributed Storage Applications



LBNL↔NTON↔SLAC experiments: Hardware configuration

- **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!**
  - **latency for a single 64 KByte data block is 25 ms, so pipelining is very important**

Distributed Parallel Storage System

Design Goals:

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

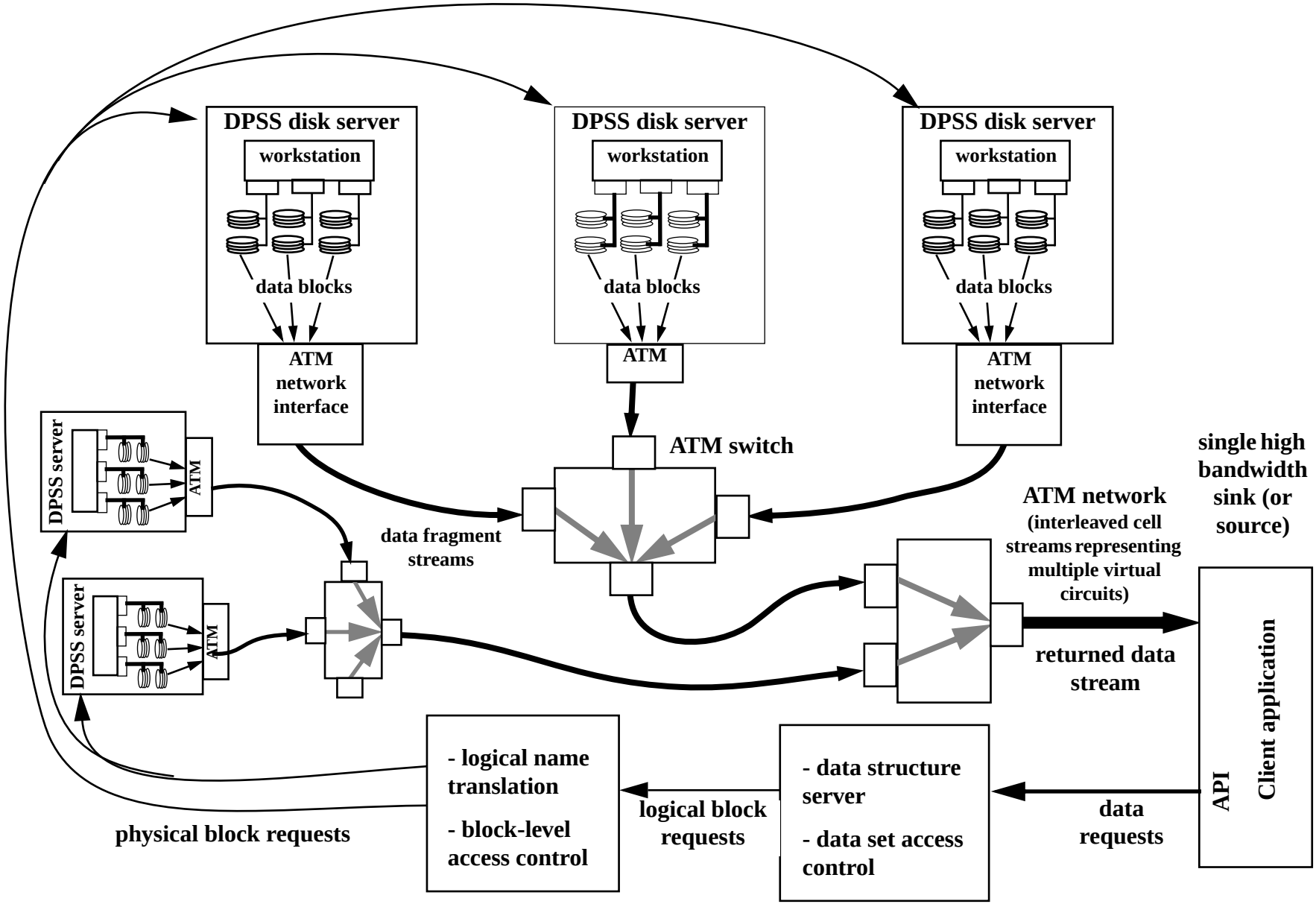
DPSS Architecture:

- A dynamically configurable network-striped disk array designed to supply high speed data streams to other processes in the network
- Provides the functionality of a single, very large, random-access, block-oriented I/O device (i.e.: a large virtual disk)
- At the user level, it is a semi-persistent cache of named data-objects
- At the storage level, it is a logical block server
- Allows client applications complete freedom to determine optimal data layout, replication, and dynamic reconfiguration
- The DPSS is NOT a reliable tertiary storage system

Implementation:

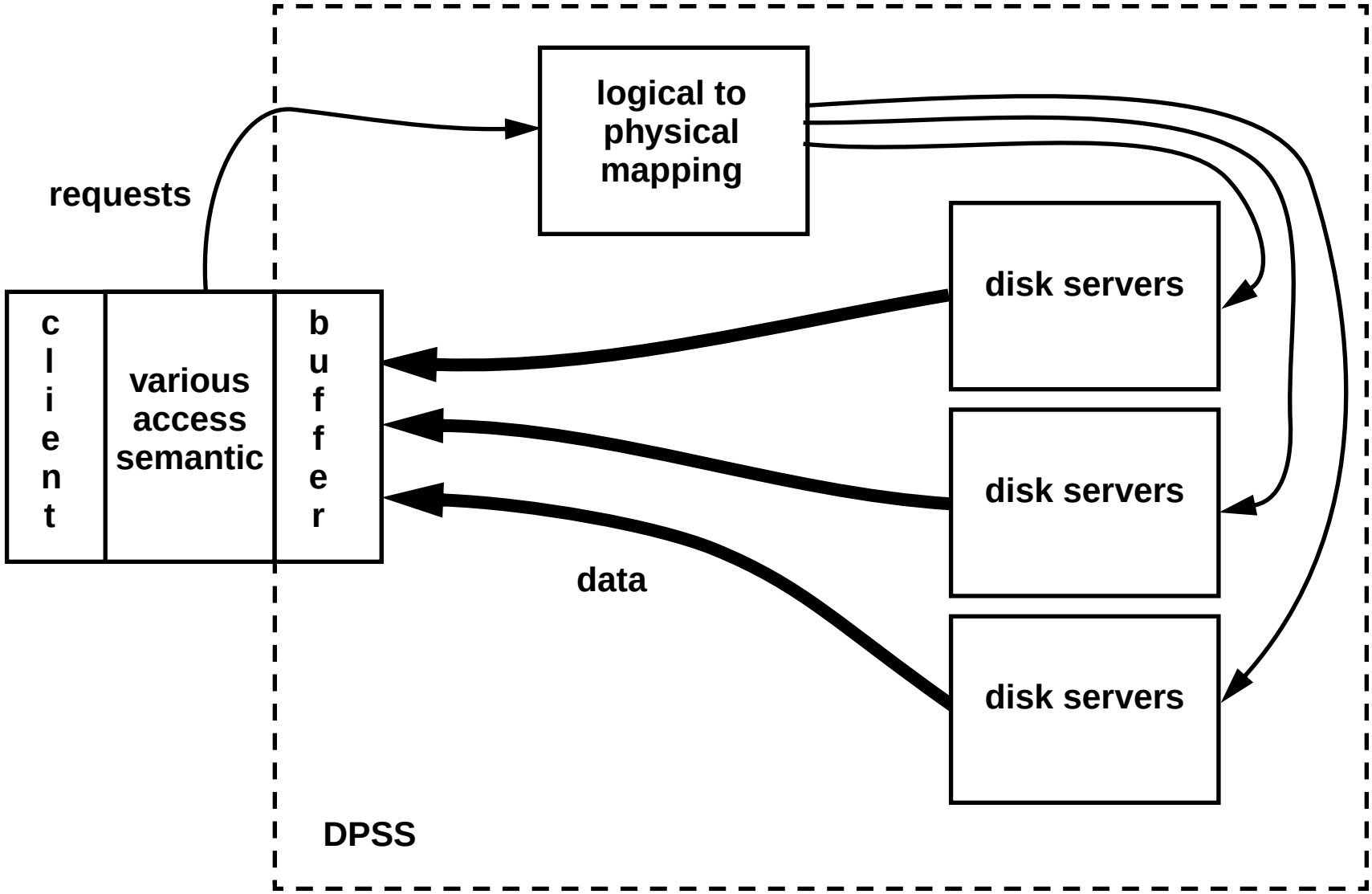
- zero memory copies of data blocks, and is all user level code
- runs on Solaris, IRIX, DEC Unix, Linux, FreeBSD, Solaris X86
- very large logical address/name space (16 bytes)
- highly distributed, a high degree of parallelism at every level, and highly pipelined
- data blocks are declustered across both disks and servers to maximize parallelism
- supports parallel reading and writing
- TCP Transport (UDP data transfers also supported)
- highly instrumented

DPSS



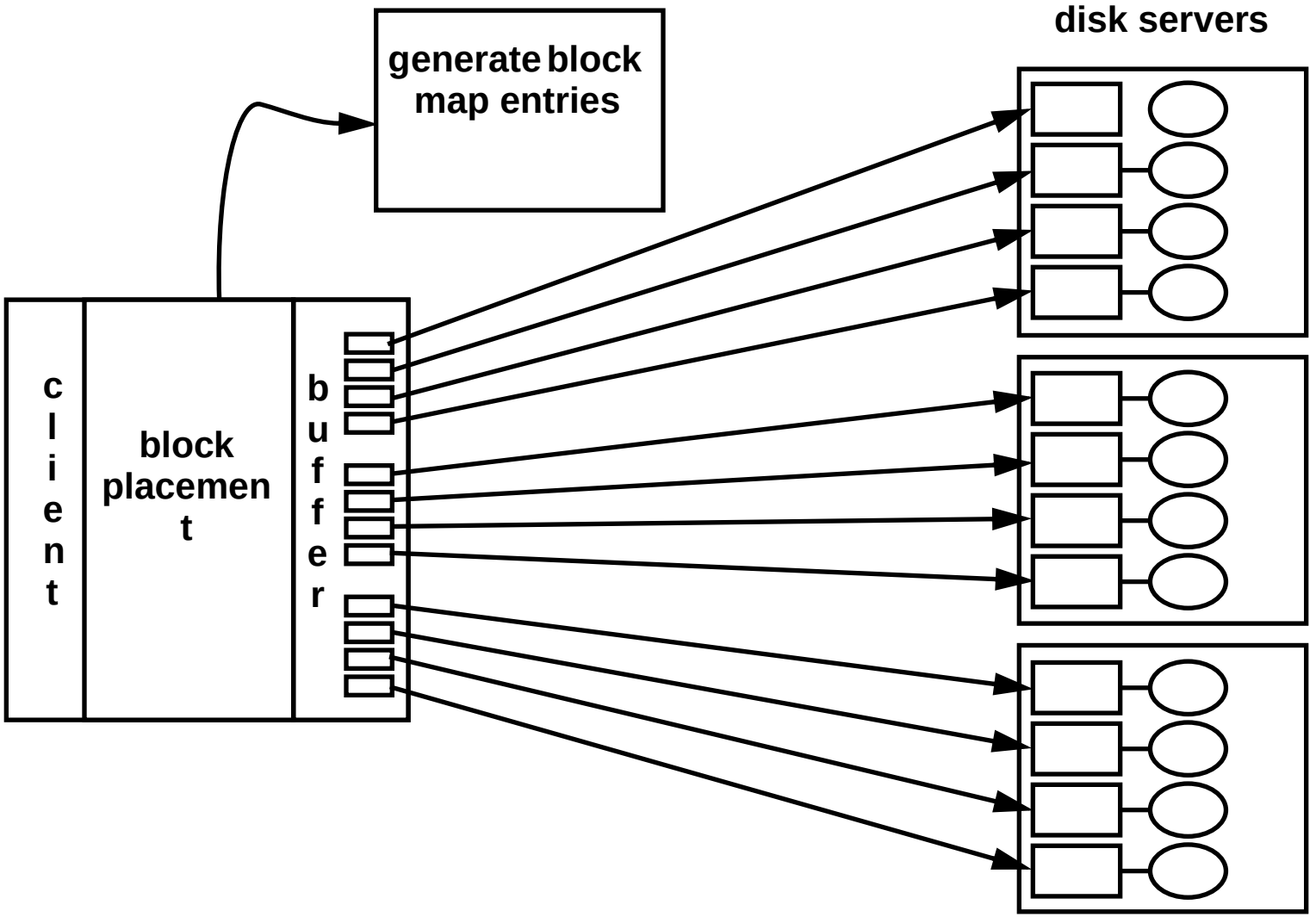
Architecture for Distributed-Parallel Storage System

DPSS



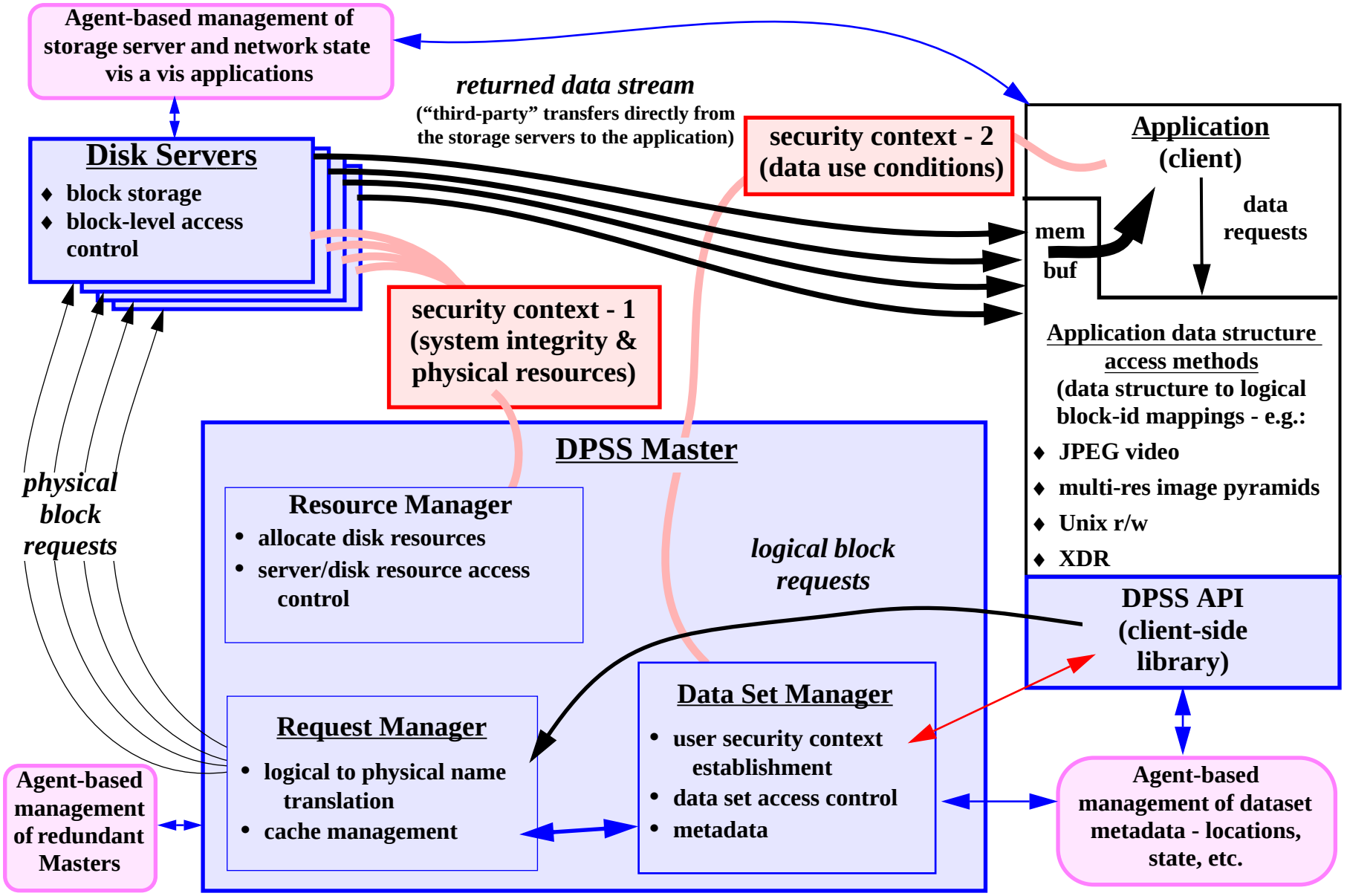
Distributed-Parallel Storage System Model (Reading)

DPSS



DPSS model for high-speed writing

DPSS



Distributed-Parallel Storage System Architecture (data reading illustrated)

DPSS

- **Typical DPSS implementation**
  - **5 UNIX workstations (e.g. Sun Ultra I0s, Pentium 400)**
  - **4 - 6 Ultra-SCSI disks on 2 SCSI host adaptors**
  - **a high-speed network (e.g.: ATM or 100 Mbit ethernet)**
- **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 adaptors**
  - **five network interfaces**

- **Sample Costs:**
  - **server host = Sun Ultra 10S: $5000**
    - **throughput = 11 - 14 MB/sec**
  - **disk = 18 GB Ultra-wide Seagate Cheetah: $1500**

TABLE 1.

Throughput	Capacity	Configuration	Cost
10 MB/sec	33 GB	1 server, 2 disks	$8K
50 MB/sec	165 GB	5 servers, 10 disks	$40 K
50 MB/sec	1 TB	5 servers, 64 disks	$121 K
100 MB/sec	.5 TB	10 servers, 32 disks	$98 K
100 MB/sec	1 TB	10 servers, 64 disks	$146 K

- **Note that cost is mainly dominated by disk price**

Other Types of Distributed Storage

Other systems that provide distributed storage capabilities:

- **AFS**
- **DCE / DFS**

These system provide full file system functionality, plus security and scalablity.

But, they do not provide enough throughput for data intensive applications:

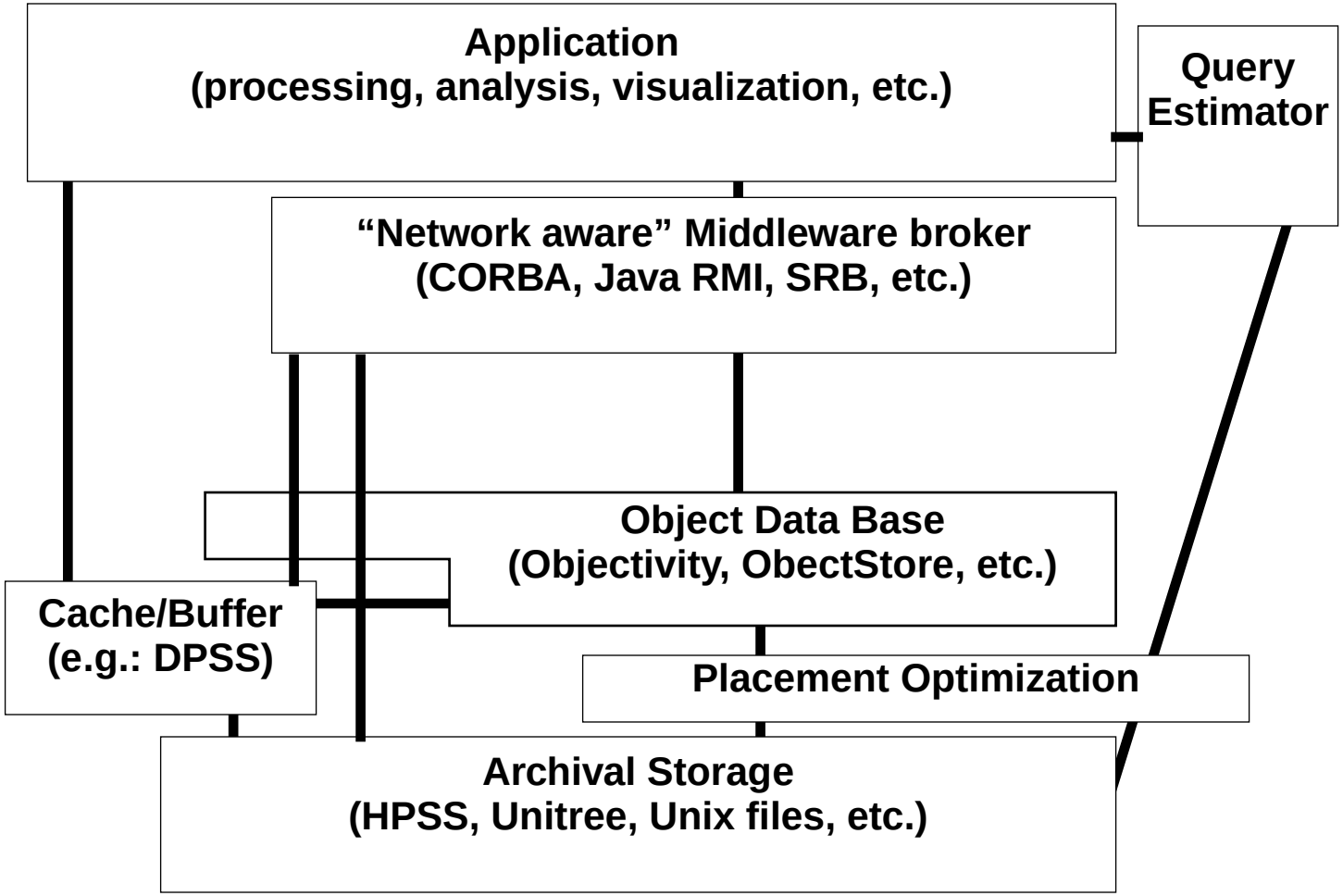
- **can’t stripe a file across several servers**

Next Generation Architecture

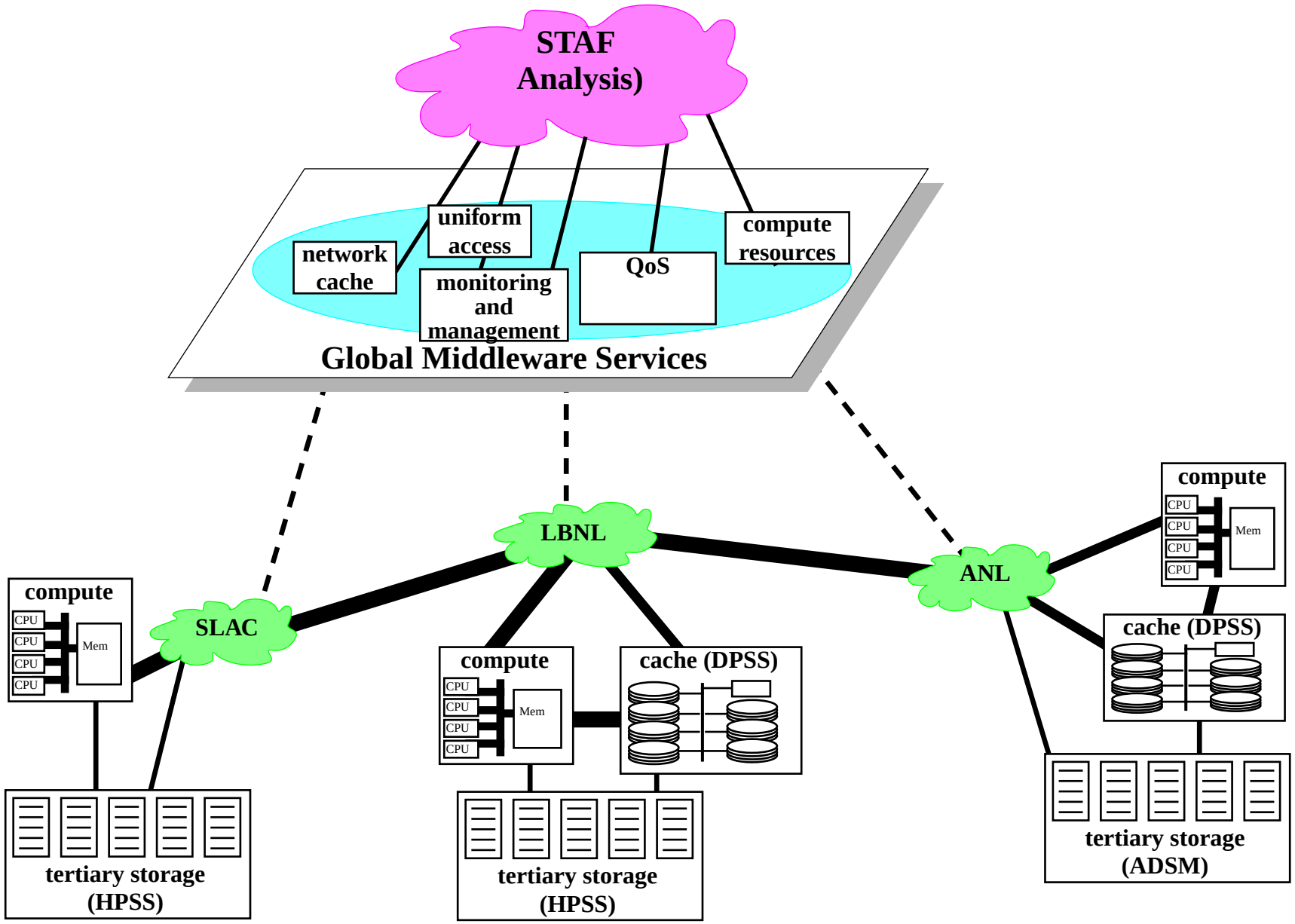
The next step is to add the following components to this data handling architecture:

- **security**
- **middleware**
  - **Globus**
- **global naming**
  - **URN system**
  - **SRB from SDSC**
- **Other “computational grid” components**

High Performance Data Intensive Computing Environment



Next Generation Architecture



Part II: Performance Analysis Tools

Overview

- **The Problem:**

  When building data intensive distributed services, we often observe unexpectedly low network throughput and/or high latency - the reasons for which are usually not obvious.

  The bottlenecks can be in any of the following components:

  - **the applications**
  - **the operating systems**
  - **the device drivers, the network adapters on either the sending or receiving host (or both)**
  - **the network switches and routers, and so on**

- **The Solution:**

  Highly instrumented systems with precision timing information and analysis tools

Motivation

There are virtually no behavioral aspects of widely distributed applications that can be taken for granted - they are fundamentally different from LAN-based distributed applications.

- **Techniques that work in the lab frequently do not work in a WAN environment (even a testbed network)**

To characterize the wide area environment we have developed a methodology for detailed, *end-to-end, top-to-bottom monitoring* and analysis of significant events involved in distributed systems data interchange.

- **This has proven invaluable for isolating and correcting performance bottlenecks, and even for debugging distributed parallel code.**

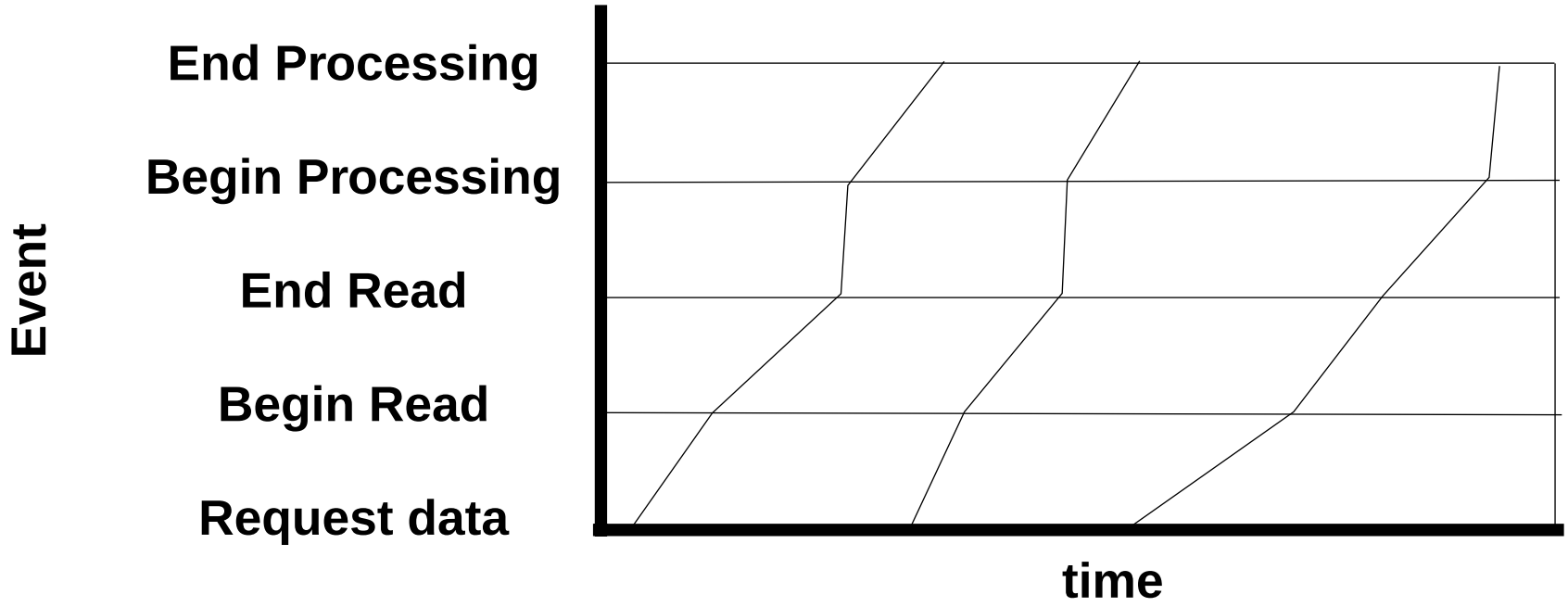
NetLogger

- We have developed the NetLogger Toolkit: a set of tools to aid in creating graphs that trace a data request throughout a distributed system.
  - NetLogger makes it easy to modify distributed applications to log interesting events at every critical point in a distributed system.
  - 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

NetLogger

- **Logged events are correlated with system behavior to characters the performance of the system during actual operation**
  - **facilitates bottleneck identification**
- **Using “life-lines” to visualize the data flow is the key to easy interpretation of the results.**
- **We believe this type of monitoring is a critical component to building reliable high performance data intensive systems**

NetLogger Event "Life-lines"



Netlogger Components

- Common log format
- Application libraries for generating NetLogger Messages
  - Can send log messages to:
    - file
    - syslogd
    - host/port (netlogd)
  - C, C++ and Java are currently supported
- Event Visualization tools
- Management Agents
- Modified Unix network and OS monitoring tools to log "interesting" events using the same log format
    - *netstat*, *vmstat*, and *tcpdump* modified output results in the NetLogger log format

NetLogger log format:

We are using the IETF draft standard Universal Logger Message (ULM) format:

- a list of “field=value” pairs
- required fields: DATE, HOST, PROG, and LVL
  - LVL is the severity level (Emergency, Alert, Error, Usage, etc.)
- followed by optional user defined fields

NetLogger adds these required fields:

- NL.EVNT, a unique identifier for the event being logged. i.e.:
  DPSS_SERV_IN, VMSTAT_USER_TIME, NETSTAT_RETRANSSEG
- NL.SEC, and NL.USEC, which are the seconds and microseconds from the Unix *gettimeofday* system call

Sample NetLogger ULM event:

DATE=19980430133038 HOST=foo.lbl.gov PROG=testprog LVL=Usage NL.EVNT=SEND_DATA NL.SEC=893968238 NL.USEC=55784 SEND.SZ=49332

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

NL.EVNT=NETSTAT_RETRANSSEGS NS.RTS=2

- **the number of TCP retransmits since the previous event**

NetLogger API:

Open calls:

```
NLhandle   *lp = NULL;

/* log to a local file */
lp = NetLoggerOpen(NL_FILE, program_name, log_filename,
                   NULL, 0);
/* log to syslog */
lp = NetLoggerOpen(NL_SYSLOG, program_name, NULL, NULL, 0);

/* log to “netlogd” on the specified host/port */
lp = NetLoggerOpen(NL_HOST, program_name, NULL, hostname,
                   DPSS_NETLOGGER_PORT);
/* log to memory, then flush to host/port */
lp = NetLoggerOpen(NL_HOST_MEM, program_name, NULL, hostname,
                   DPSS_NETLOGGER_PORT);
```

Write the log event:

```
NetLoggerWrite(lp, "EVENT_NAME",  "F1=%d F2=%d F3=%d
      F4=%.2f", data1, data2, string, fdata);
```

Sample Code:

```
/* log to a local file */
lp = NetLoggerOpen(method, progname, log_filename, NULL, 0);

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 Visualization Tools

Exploratory, interactive analysis of the log data has proven to be the most important means of identifying problems.

We have developed a tool called *nlv* (NetLogger Visualization).

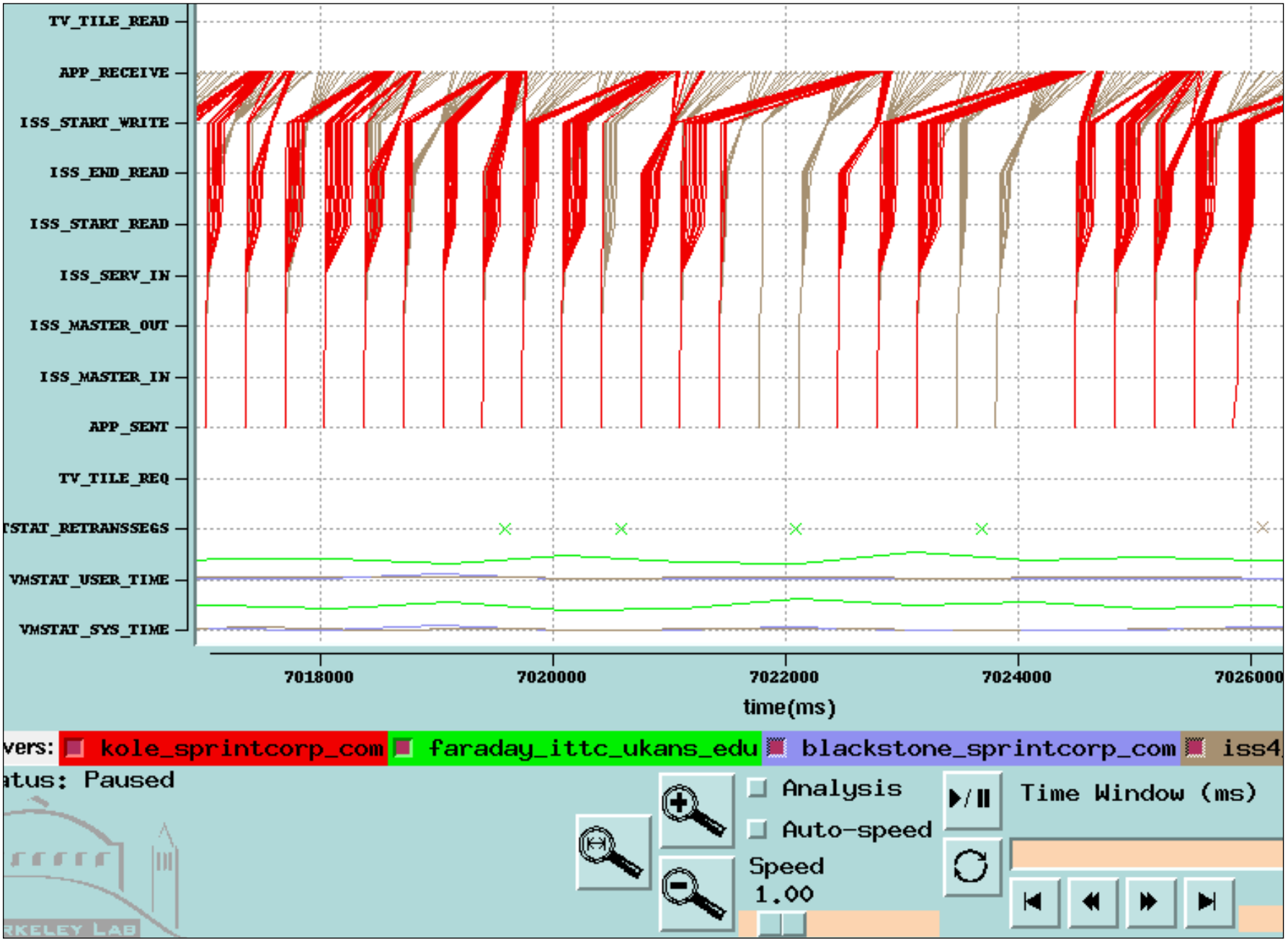
***nlv* functionality:**

- **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"**

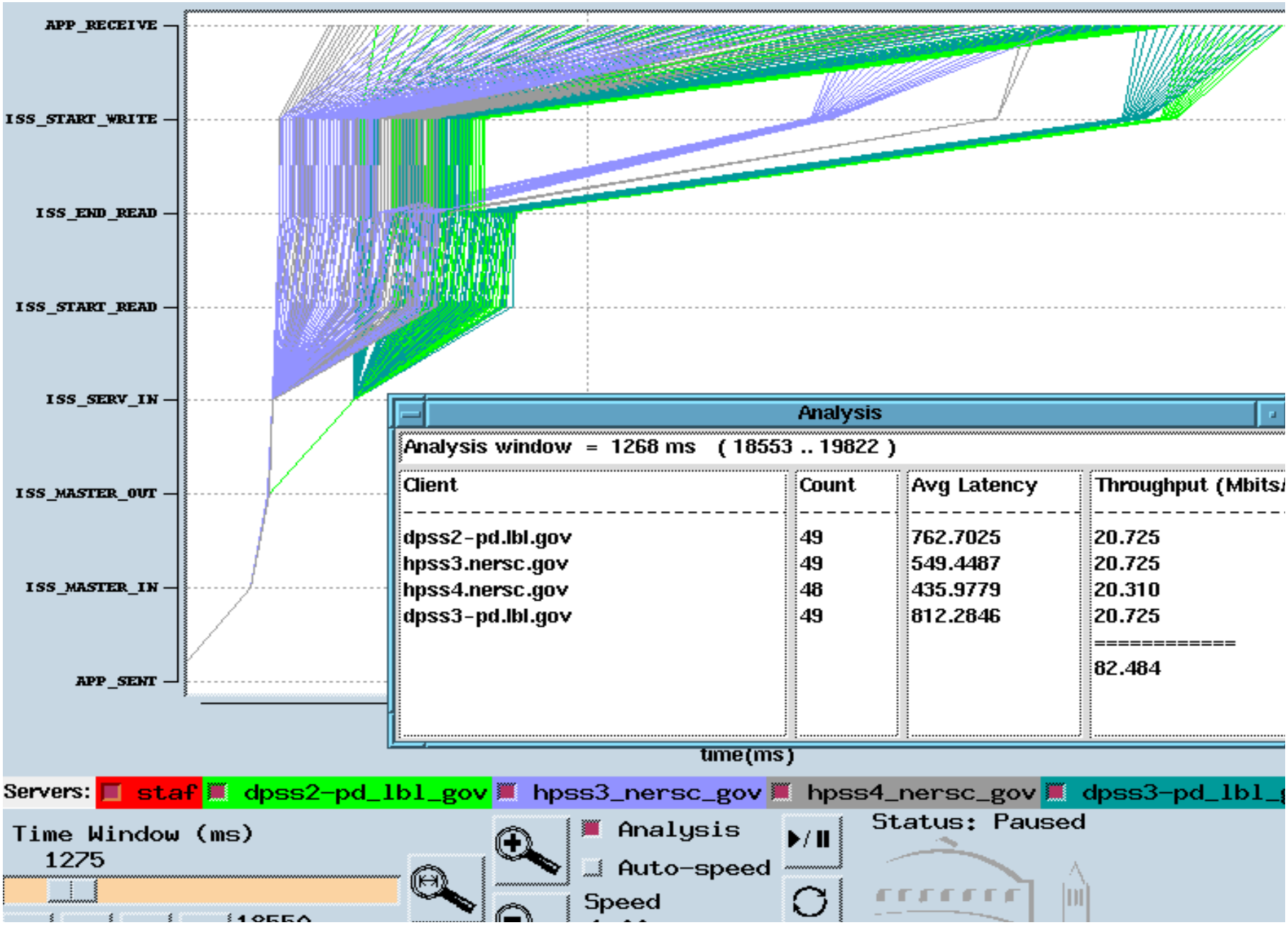
Other NetLogger tools to analyze log files:

- ***perl* scripts to extract information from log files**
- ***gnuplot* to graph the results**

NLV


TV_TILE_READ
APP_RECEIVE
ISS_START_WRITE
ISS_END_READ
ISS_START_READ
ISS_SERV_IN
ISS_MASTER_OUT
ISS_MASTER_IN
APP_SENT
TV_TILE_REQ
TSTAT_RETRANSSEGS
VMSTAT_USER_TIME
VMSTAT_SYS_TIME
7018000
7020000
7022000
7024000
7026000
time(ms)
vers:
kole_sprintcorp_com
faraday_ittc_ukans_edu
blackstone_sprintcorp_com
iss4
tus: Paused
Analysis
Auto-speed
Speed
1.00
Time Window (ms)

NLV


APP_RECEIVE
ISS_START_WRITE
ISS_END_READ
ISS_START_READ
ISS_SERV_IN
ISS_MASTER_OUT
ISS_MASTER_IN
APP_SENT
Analysis
Analysis window = 1268 ms (18553 .. 19822)
Client
Count
Avg Latency
Throughput (Mbits/
dpss2-pd.lbl.gov 49 762.7025 20.725
hpss3.nersc.gov 49 549.4487 20.725
hpss4.nersc.gov 48 435.9779 20.310
dpss3-pd.lbl.gov 49 812.2846 20.725
82.484
time(ms)
Servers: staf dpss2-pd_lbl_gov hpss3_nersc_gov hpss4_nersc_gov dpss3-pd_lbl_g
Time Window (ms)
1275
Analysis
Auto-speed
Speed
Status: Paused

Network Time Protocol

- For NetLogger timestamps to be meaningful, all systems clocks must be synchronized.
- NTP is used to synchronize time of all hosts in the system.
  - NTP is from Dave Mills, U. of Delaware (http://www.eecis.udel.edu/~ntp/)
- Must have NTP running on one or more primary servers, and on a number of local-net hosts, acting as secondary time servers.

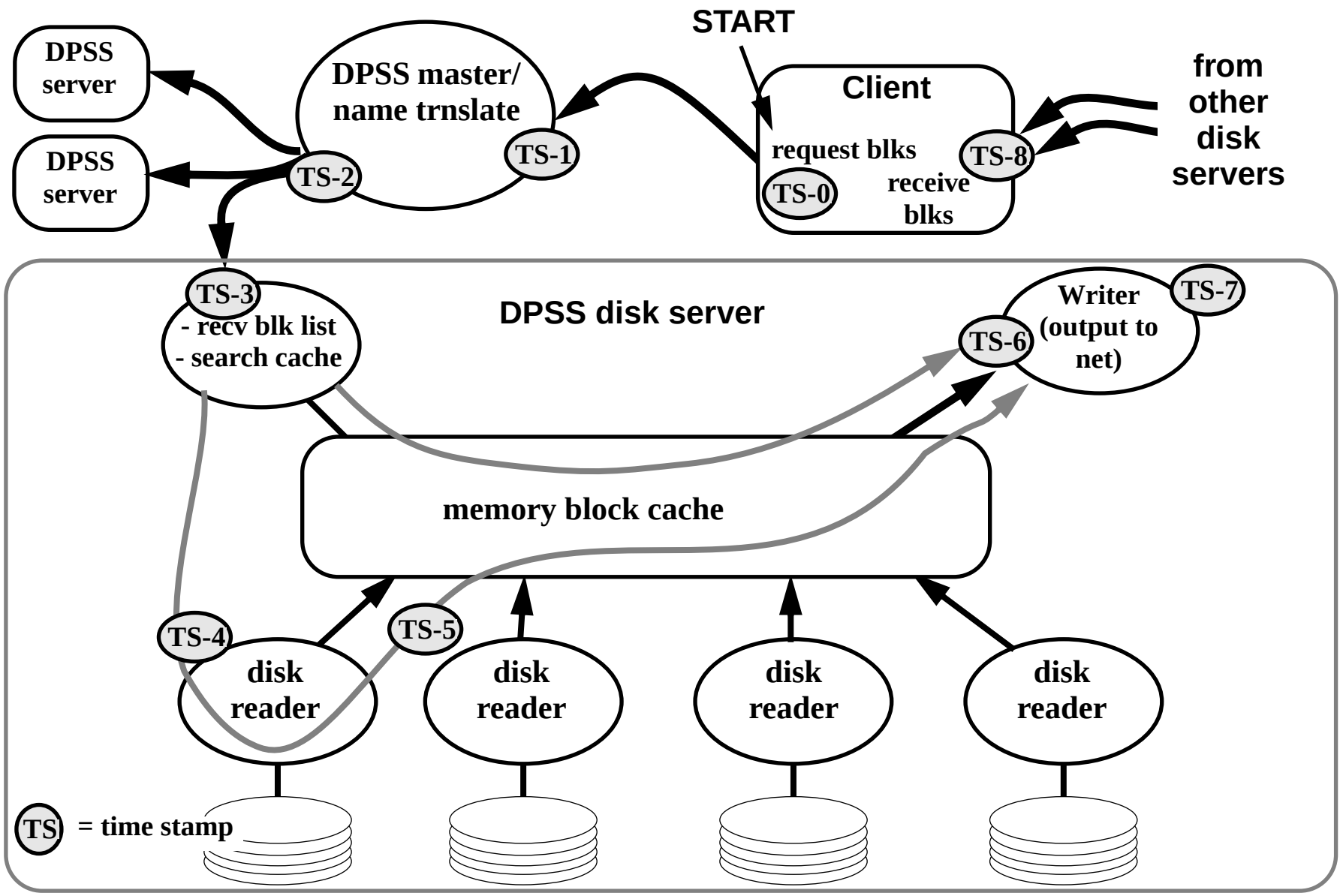
NTP

- **Purpose of NTP**
  - **conveys timekeeping information from the primary servers to other time servers via the Internet**
  - **cross-checks clocks and mitigates errors due to equipment or propagation failures**
- **Host time servers will synchronize via another peer time server, based on the following timing values:**
  - **those determined by the peer relative to the primary reference source of standard time**
  - **those measured by the host relative to the peer**
- **NTP provides not only precision measurements of offset and delay, but also definitive maximum error bounds, so that the user interface can determine not only the time, but the quality of the time as well.**

Running NTP:

- All hosts run the *xntpd* daemon, which synchronizes the clocks of each host both to GPS-based time servers and to each other.
- This allows us to synchronize the clocks of all hosts to within about 250 microseconds of each other, but...
  - systems have to stay up for a significant length of time for the clocks to converge to 250 μs
  - best to have a time server on the same network as all hosts
  - many different sys admins (harder to synchronize than clocks)
- In practice, clock synchronization of 1ms is good enough

DPSS Monitoring Points



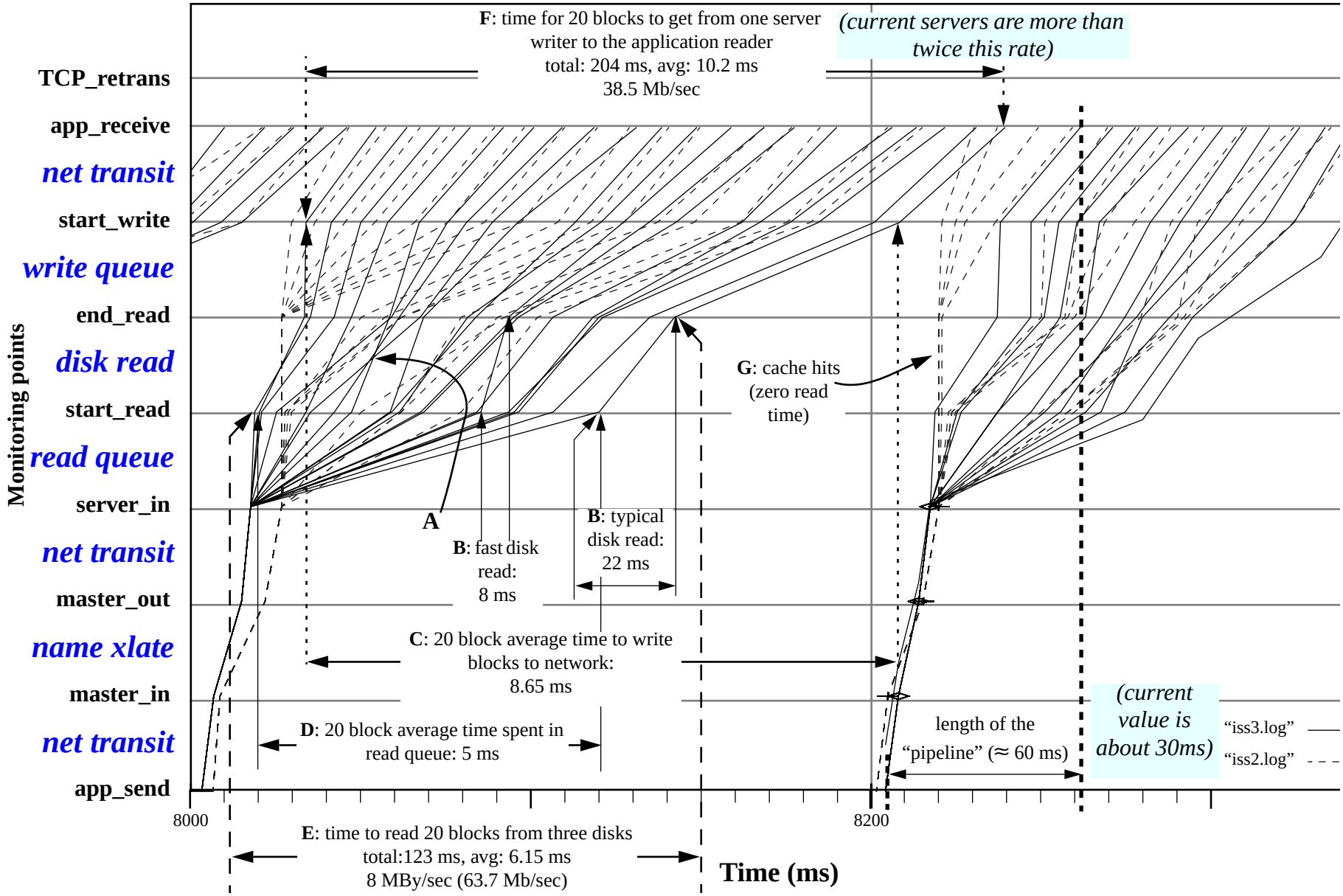
DPSS life-lines:

- each line represents the history of a data block as it moves through the end-to-end path
- data requests are sent from the application every 200 ms (the nearly vertical lines starting at *app_send* monitor point)
- initial single lines fan out as the request lists are resolved into individual data blocks (*server_in*)

NetLogger Analysis of the DPSS

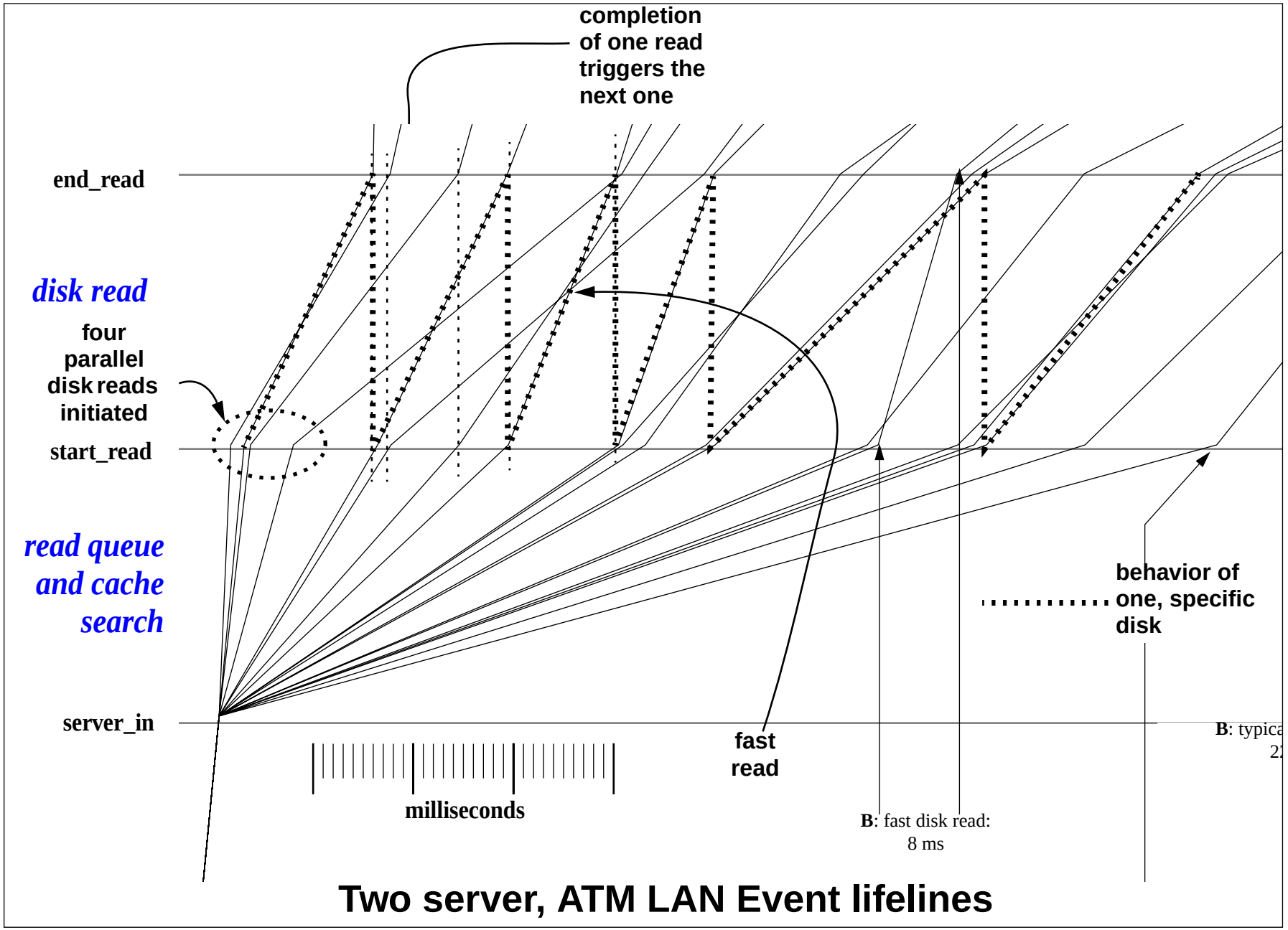
- **Analysis of the data block life-lines shows, e.g. (see next figure):**
  - **A: if two lines cross in the area between *start read* and *end read*, this indicates a read from one disk was faster than a read from another disk**
  - **B: all the disks are the same type, the variation in read times are due to differences in disk seek times**
  - **C: average time to move data from the memory cache into the network interface is 8.65 ms**
  - **D: the average time in disk read queue is 5 ms**
  - **E: the average read rate from four disks is 8 MBy/sec**
  - **F: the average send rate (receiver limited, in this case) is 38.5 Mb/sec.**
  - **G: some requested data are found in the cache (were read from disk, but not sent in previous cycle because arrival of new request list flushes write queue)**

NetLogger Analysis of the DPSS



Two server, ATM LAN

NetLogger Analysis of the DPSS

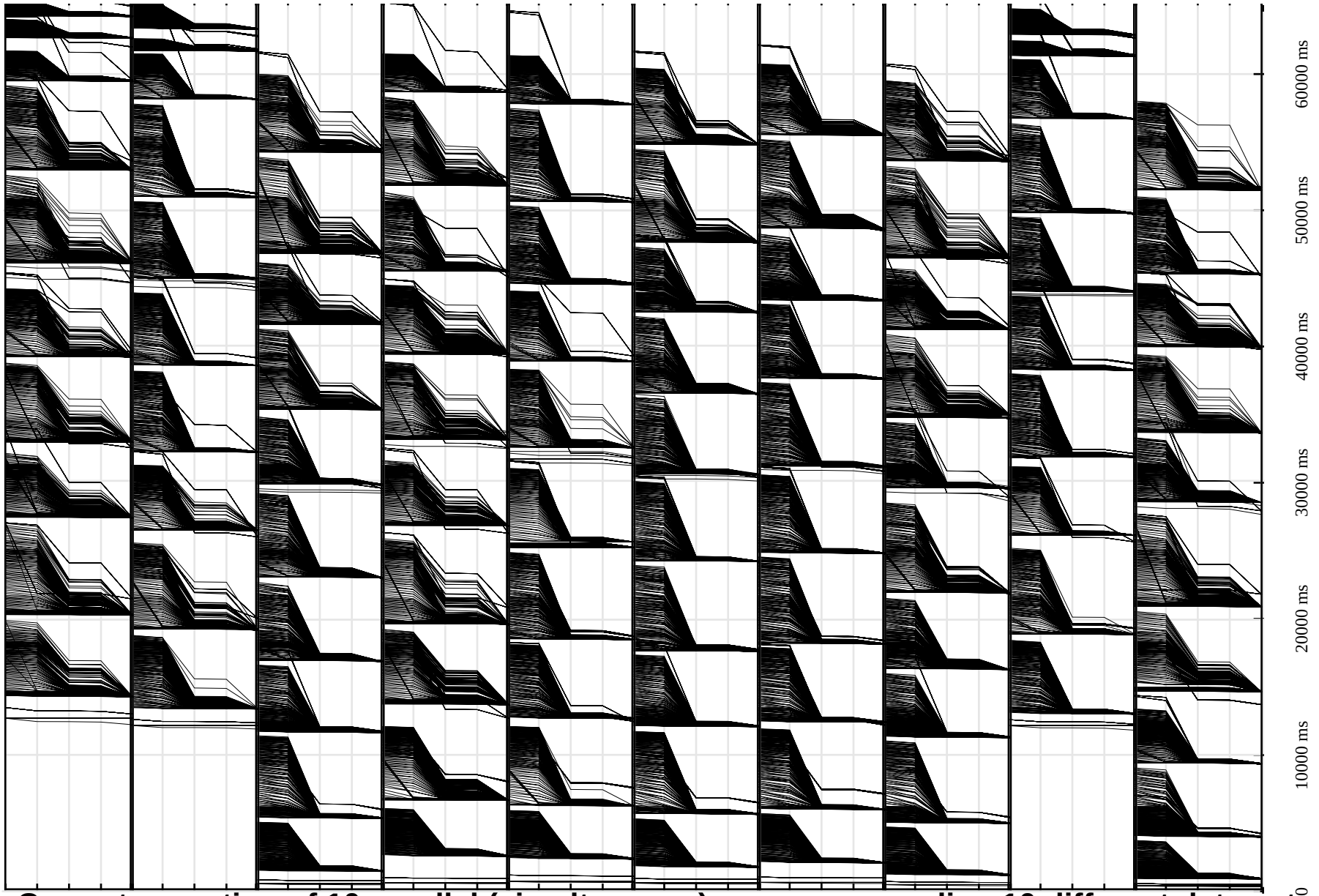


Two server, ATM LAN Event lifelines

Other NetLogger Results: A DPSS scaling experiment:

- 10 clients accessing 10 different data sets simultaneously
- show that all clients get an equal share of the DPSS resources

NetLogger Analysis of the DPSS



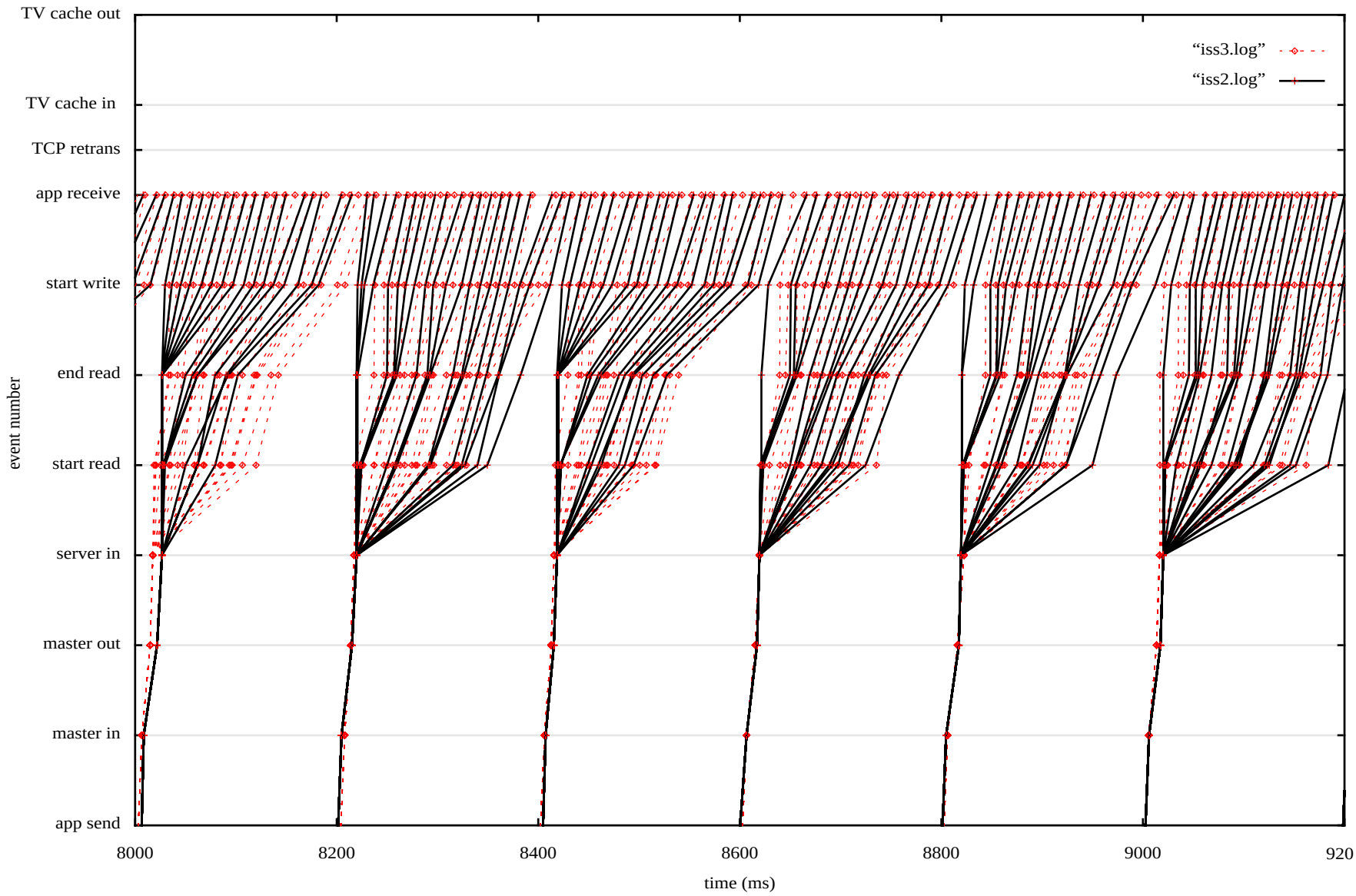
Correct operation of 10 parallel (simultaneous) processes reading 10 different data sets from one DPSS (each row is one process, each group is a request for 10 Mbytes of data,

A Case Study: TCP in the Early MAGIC Testbed

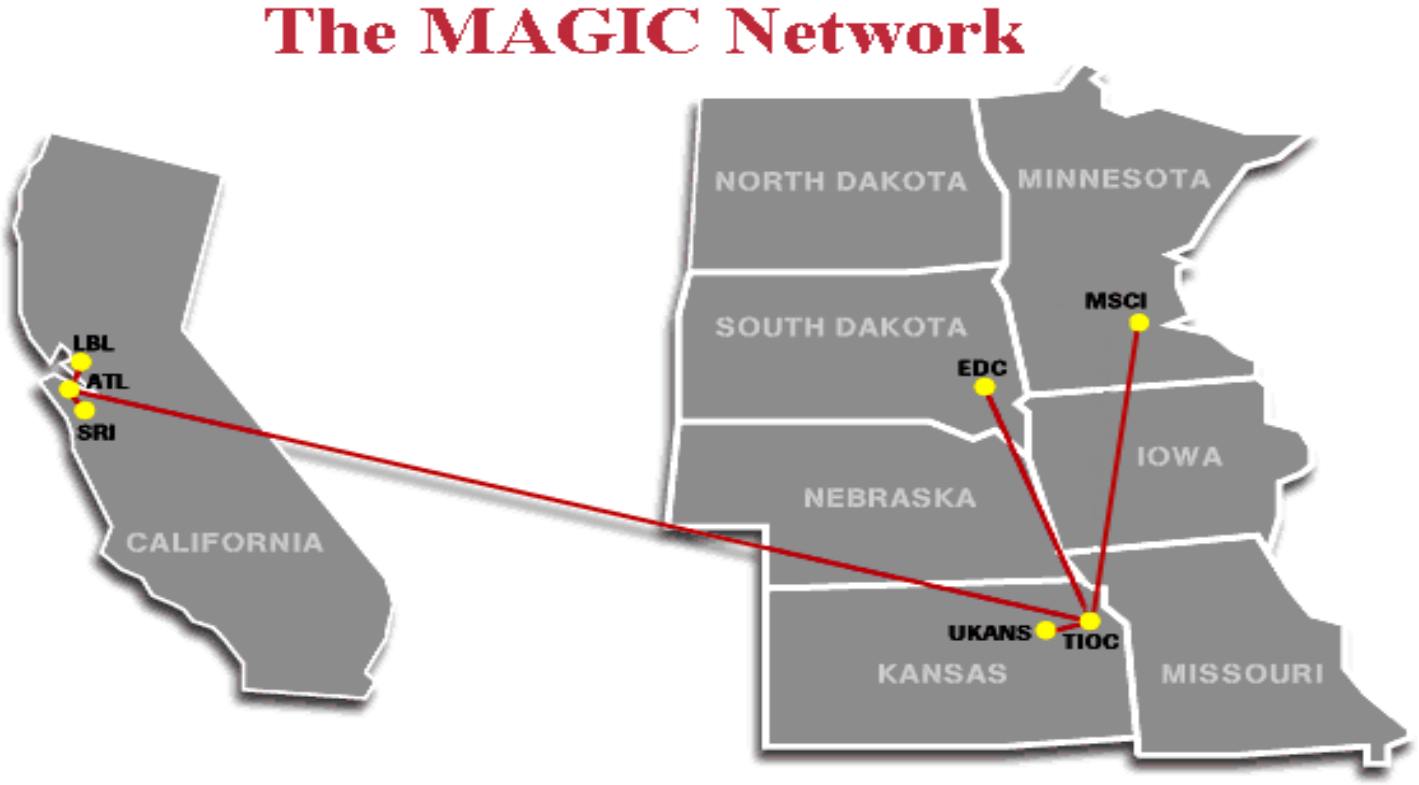
The Problem:

- DPSS and TerraVision were working well in a LAN environment, but failing to deliver high (even medium!) data rates to TerraVision when operated in the MAGIC WAN.
- We suspected that the ATM switches were dropping cells, but they reported no cell loss.
- Network engineers claimed that the network was working "perfectly". For testing they were using network performance tools such at *ttcp* and *netperf*, which are somewhat useful, but don't model real distributed applications, which are complex, bursty, and have more than one connection in and/or out of a given host at one time.

Control test: 2 servers over ATM LAN



- TerraVision image tiles are distributed across DPSS servers on the MAGIC network at the following sites:
  - EROS Data Center, Sioux Falls, SD;
  - Sprint, Kansas City, MO;
  - University of Kansas, Lawrence, KS;
  - SRI, Menlo Park, CA;
  - LBNL, Berkeley, CA

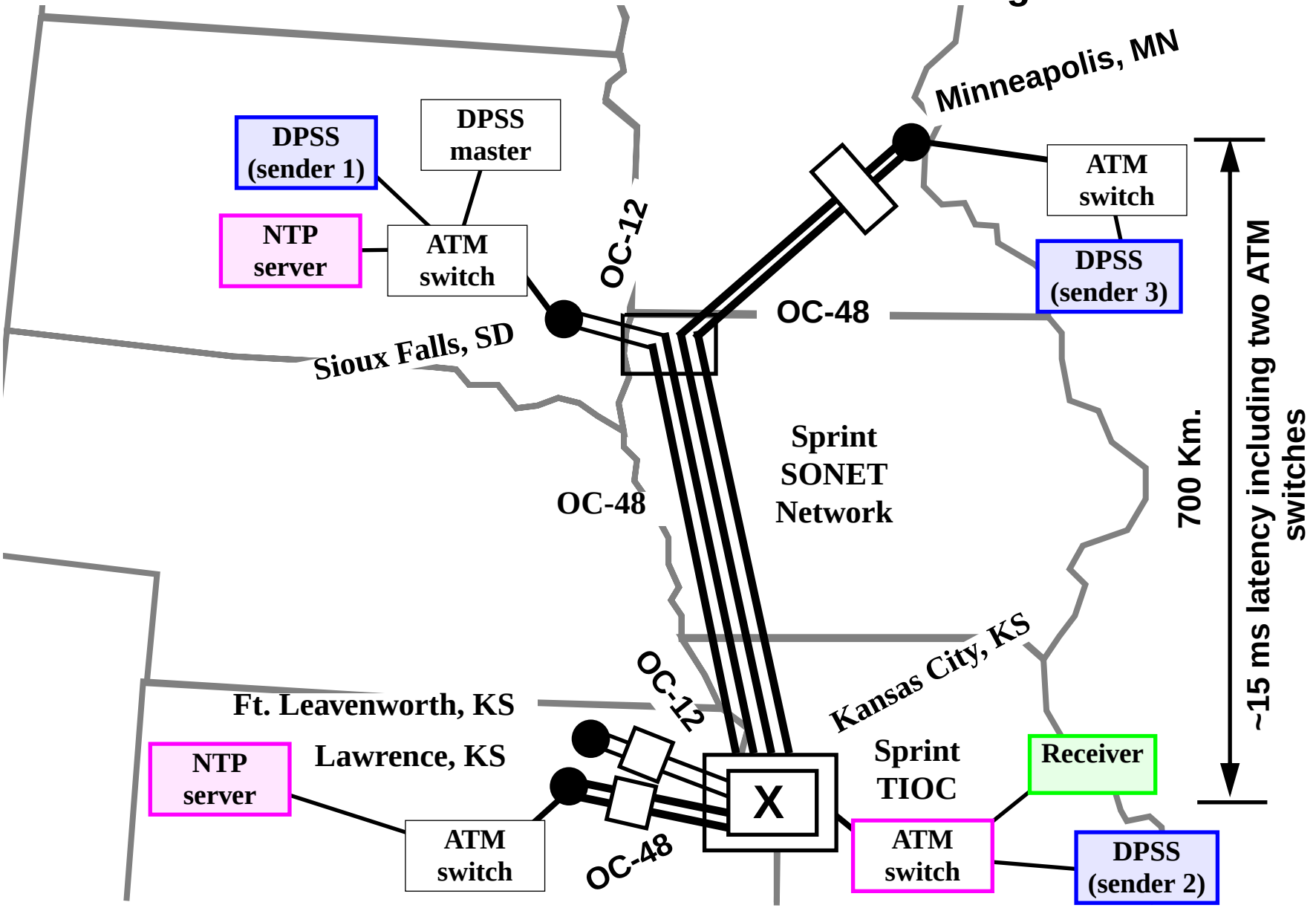


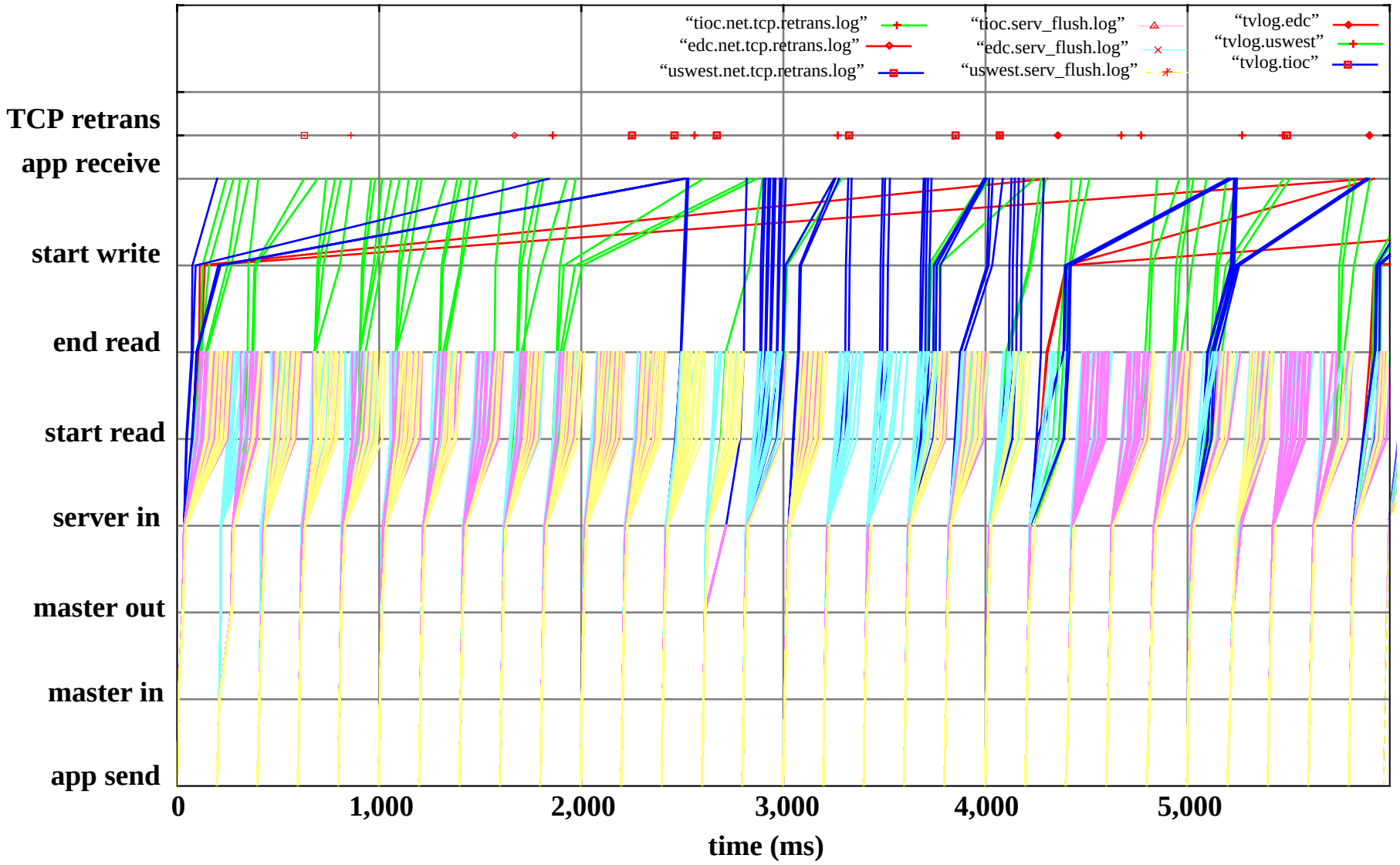
MAGIC WAN experiment:

- Three disk server configuration DPSS gave the results shown below:
- What was happening in this experiment is that TCP's normal ability to accommodate congestion is being defeated by an unreasonable network configuration:
  - the final ATM switch where the three server streams come together had a per port output buffer of only about 13K bytes
  - the network MTU (minimum transmission unit) is 9180 Bytes (as is typical for ATM networks)
  - three sets of 9 KBy IP packets were converging on a link with less than 50% that amount of buffering available, resulting in most of the packets (roughly 65%) being destroyed by cell loss at the switch output port

The next generation of ATM switches (e.g.: Fore LC switch modules) had much more buffering: 32K cells (1500 KB), and this problem went away.

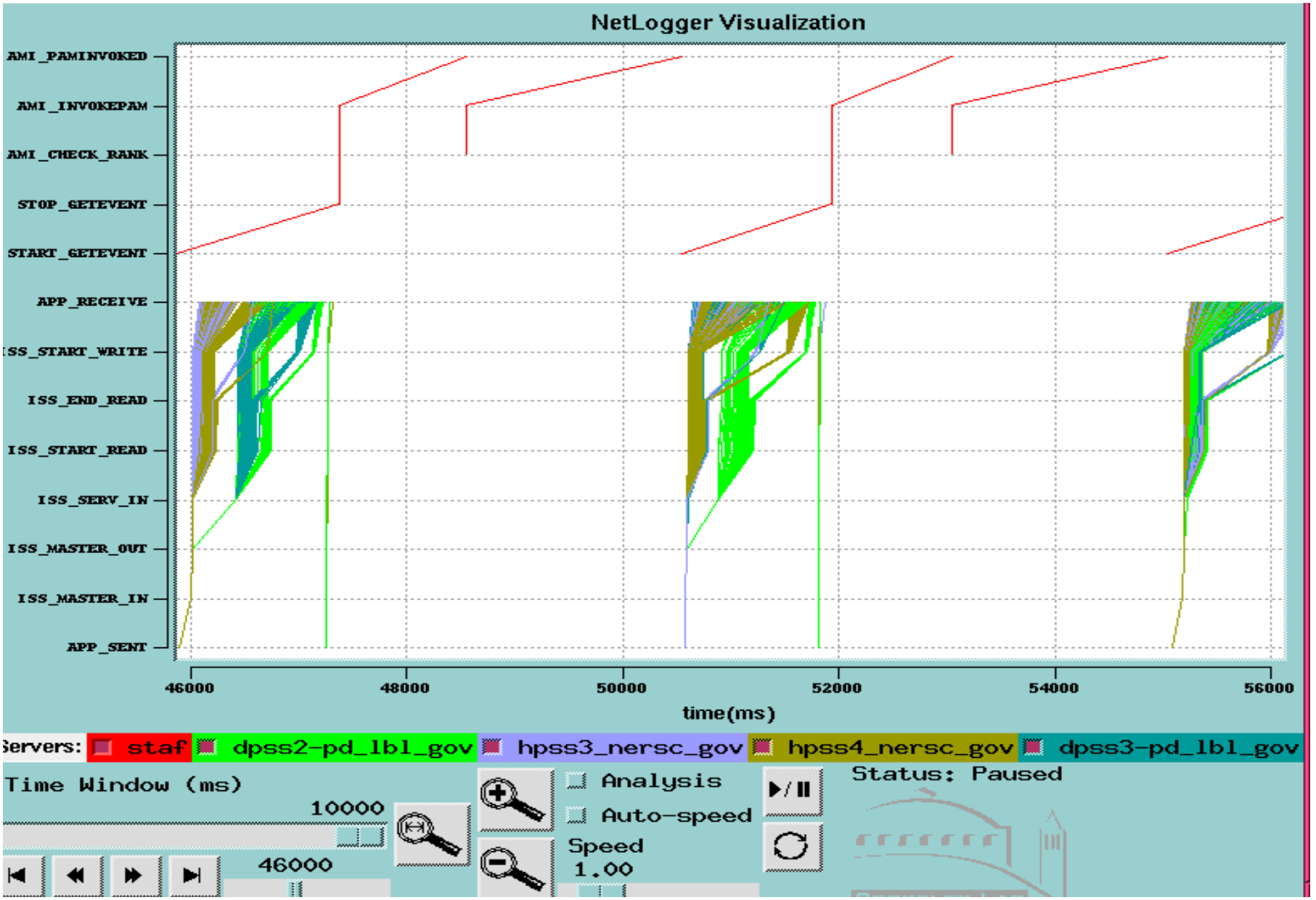
The MAGIC Network Performance Test Configuration





Three servers, ATM WAN, SS-10s as servers, tv_sim on SGI Onyx

NetLogger Analysis of HENP Application



For more information see:

http://www-didc.lbl.gov/DPSS

http://www-didc.lbl.gov/NetLogger