An Overview of The Distributed Parallel Storage System (DPSS)[1]

Brian L. Tierney (bltierney@lbl.gov)
Jason Lee, Gary Hoo, Guojun Jin, Brian Crowley
Data Intensive Distributed Computing Group

Bill Johnston, Mary Thompson
Imaging and Distributed Collaboration Group

(http://www-itg.lbl.gov)
Lawrence Berkeley National Laboratory
Berkeley, CA 94720

1. **This work is jointly supported by DARPA - ITO, and by the U. S. Dept. of Energy, Energy Research Division, Mathematical, Information, and Computational Sciences office, under contract DE-AC03-76SF00098 with the University of California.**

Overview

- **The Problem:**
  Some applications require high-speed (100's of Mbits/sec) access to very large (100's of GBytes) data sets. Having a large, local RAID array is often not feasible.
- **The Solution:**
  The Distributed-Parallel Storage System (DPSS)
- **Sample Applications:**
  - **Terrain Visualization**
  - **Medical Imaging**
  - **HENP data analysis**

Outline

- Introduction
- History: The Magic Gigabit Network Testbed and TerraVision
- DPSS Architecture and Implementation
- DPSS hardware issues
- DPSS Clients
  - HENP
  - Medical Images
- Performance Analysis and Monitoring: NetLogger Toolkit
- Current and Future Work
  - Agents / Brokers

The Distributed-Parallel Storage System (DPSS)

Goal:

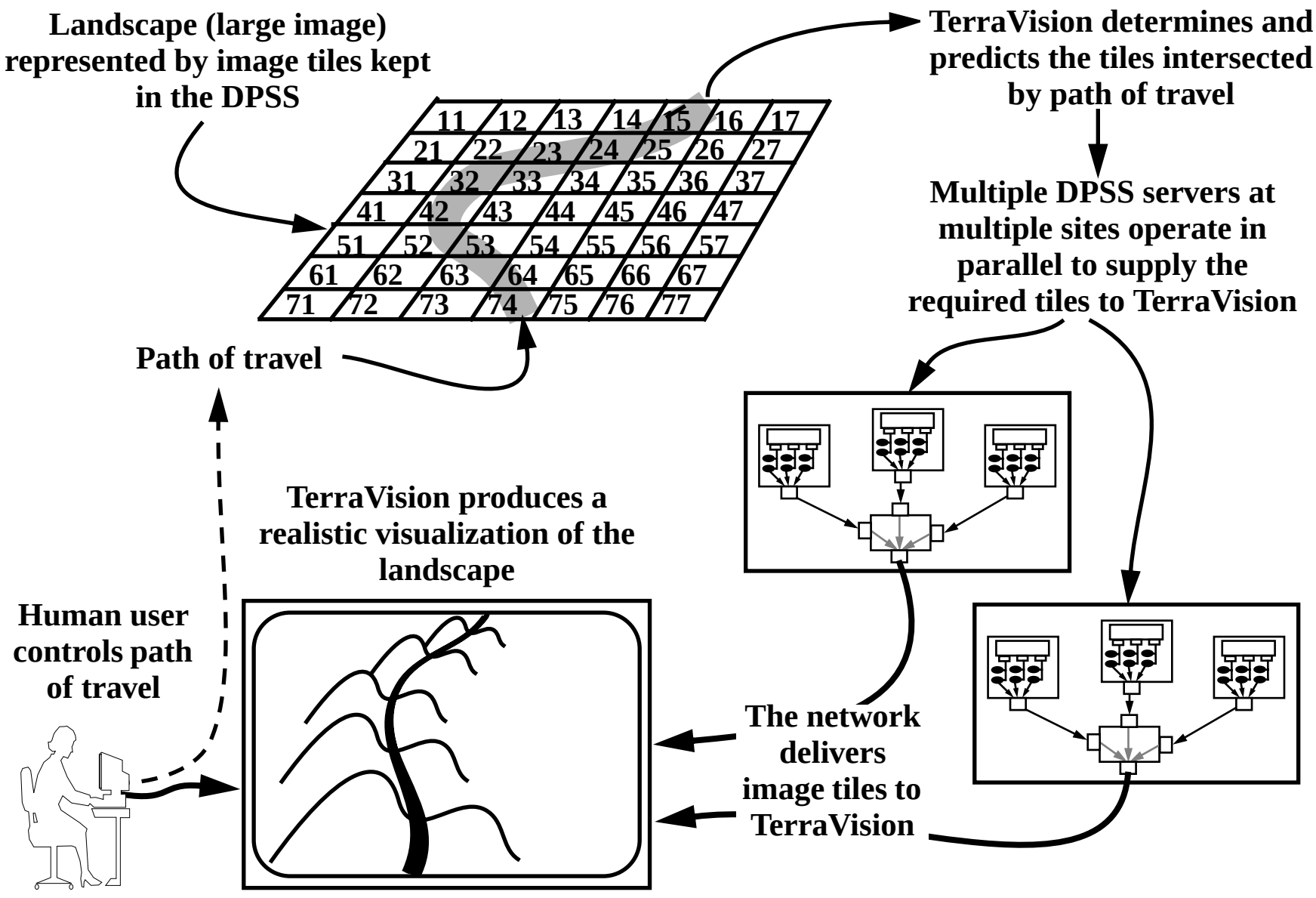
- **To handle massive volumes of data at high data rates through the use of technology that provides complete location and access transparency**

DPSS Architecture:

- **The DPSS is a dynamically configurable collection of widely distributed disk servers which operate in parallel to provide high-speed, random access to very large data sets**
- **The DPSS can provide data streams fast enough to enable various multi-user, “real-time”, virtual reality-like applications in an Internet / ATM environment**
- **At the application level, the DPSS is a persistent cache of named objects, at the storage level it is a logical block server**
- **The DPSS is NOT a reliable tertiary storage system!**

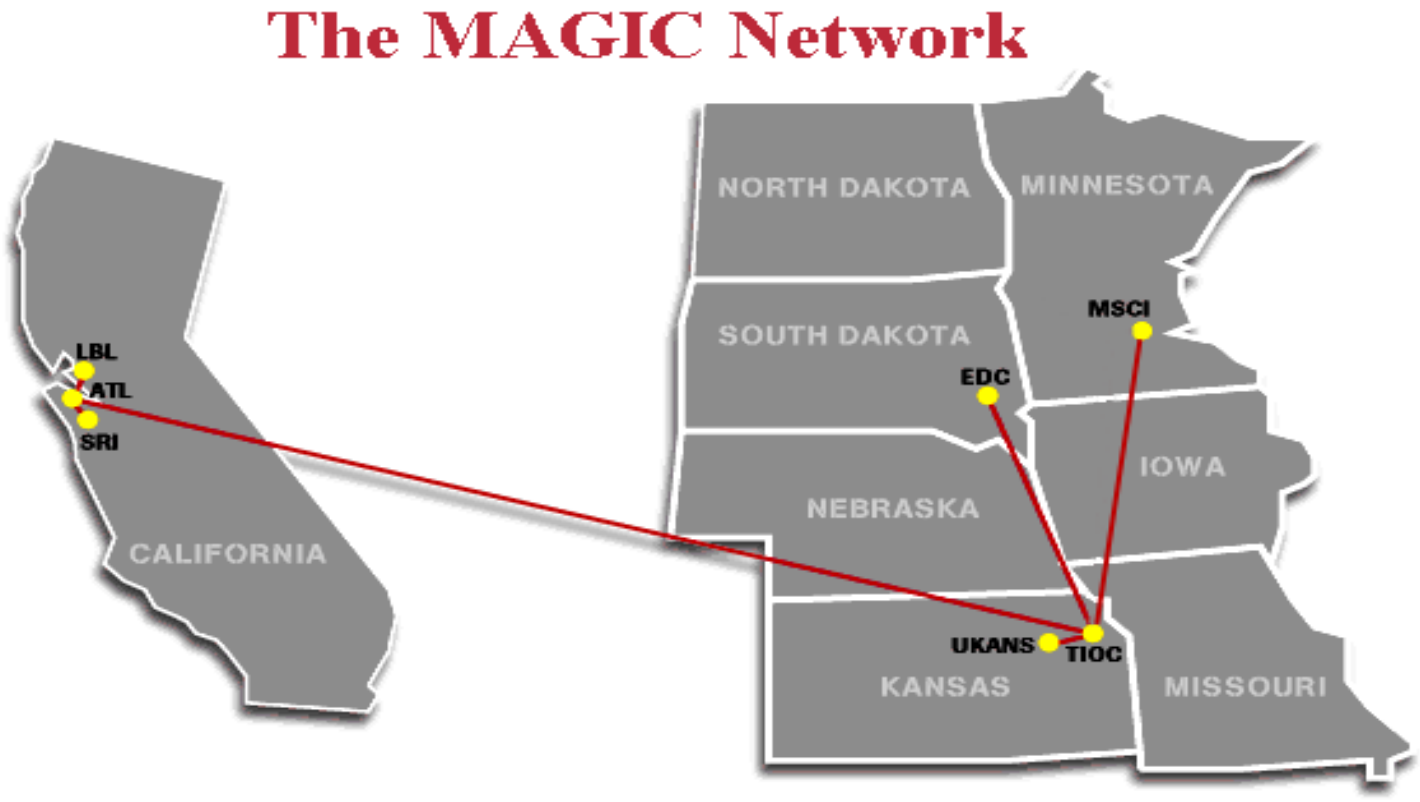
MAGIC

- DPSS is being developed as part of the DARPA-sponsored MAGIC Gigabit Network Testbed (see: http://www.magic.net)
- The prototype high-speed application for the DPSS was TerraVision, developed at SRI
- TerraVision uses tile images and digital elevation models to produce a 3D visualization of landscape.
- Several aspects of the DPSS design were to suite the needs of TerraVision
- DARPA's primary interest in the DPSS and TerraVision was to stress test new ATM OC-3 (and now OC-12) WANs.



TerraVision and the DPSS Cooperate to Visualize Landscape

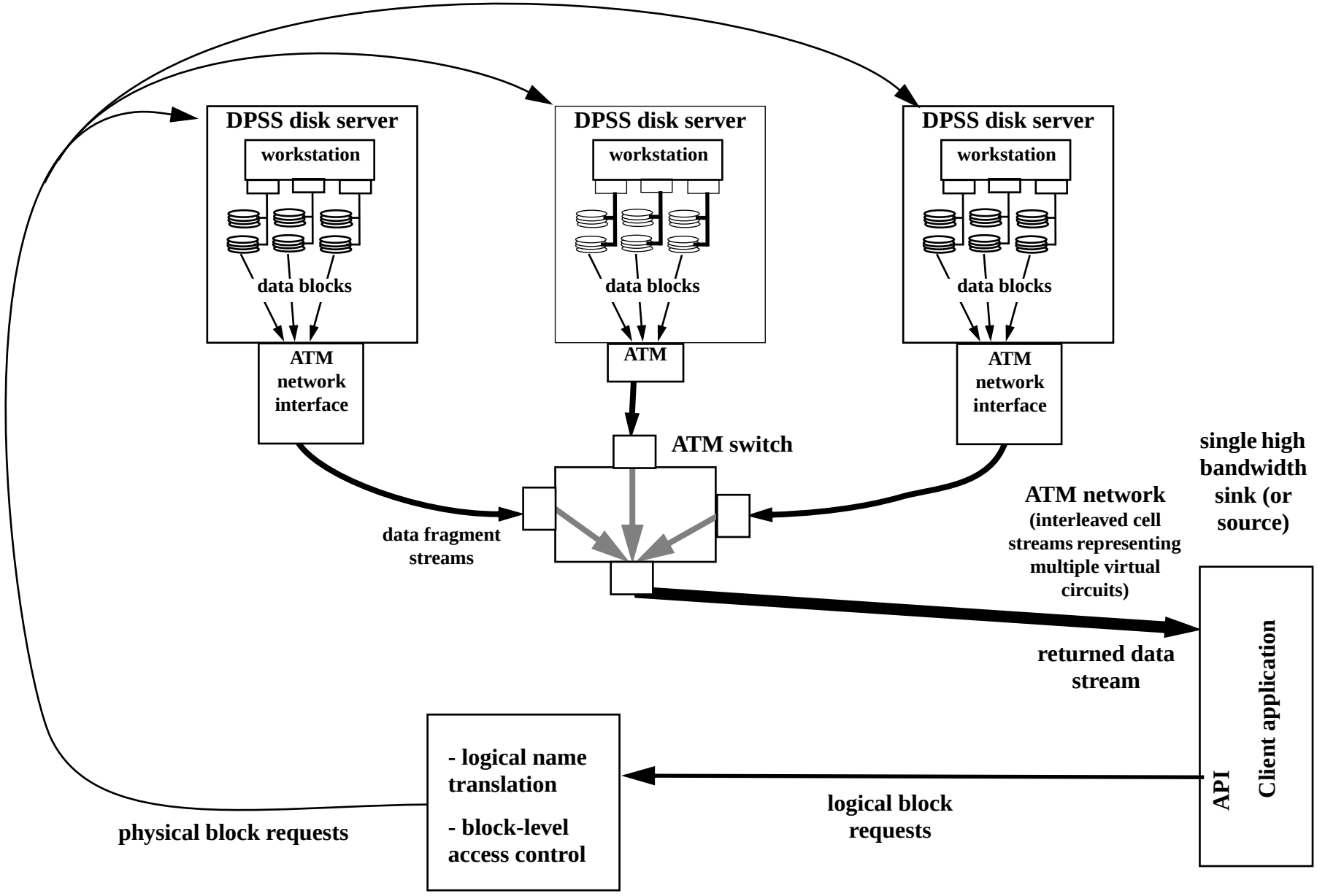
- **TerraVision image tiles are distributed across DPSS servers on the MAGIC network at the following sites:**
  - **EROS Data Center, Sioux Falls, SD;**
  - **Sprint, Kansas City, MO;**
  - **University of Kansas, Lawrence, KS;**
  - **SRI, Menlo Park, CA;**
  - **LBNL, Berkeley, CA**



DPSS Characteristics

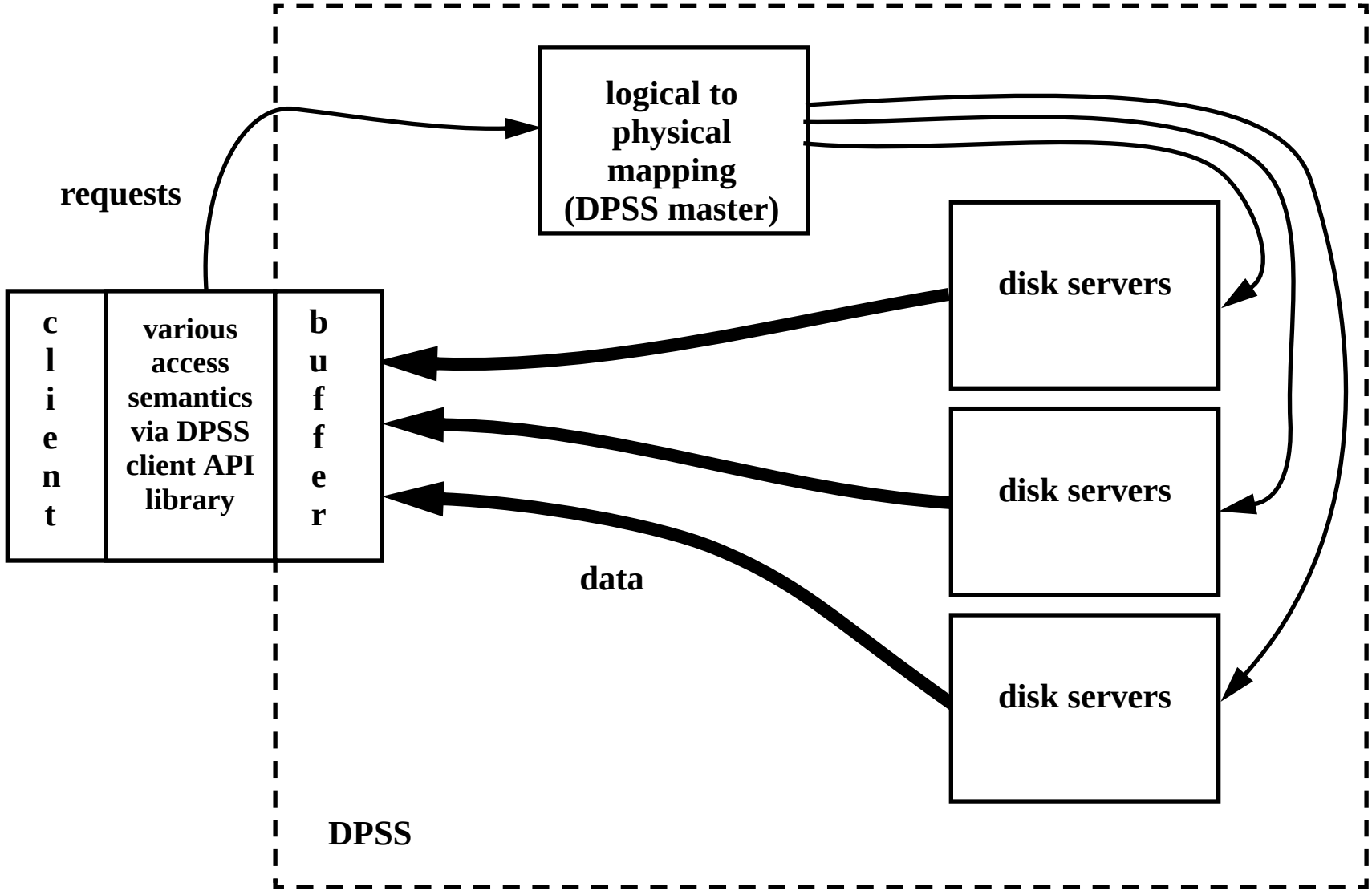
- composed of “off-the-shelf” commodity hardware components
- Ported to Solaris, IRIX, DEC Unix, Linux, FreeBSD, Solaris X86
- supports both reading and writing
- random block server, with very large logical address/name space
- highly distributed, a high degree of parallelism at every level, and highly pipelined
- implementation has zero memory copies of data blocks, and is all user level code
- uses TCP and UDP Transport
- highly instrumented

DPSS Architecture



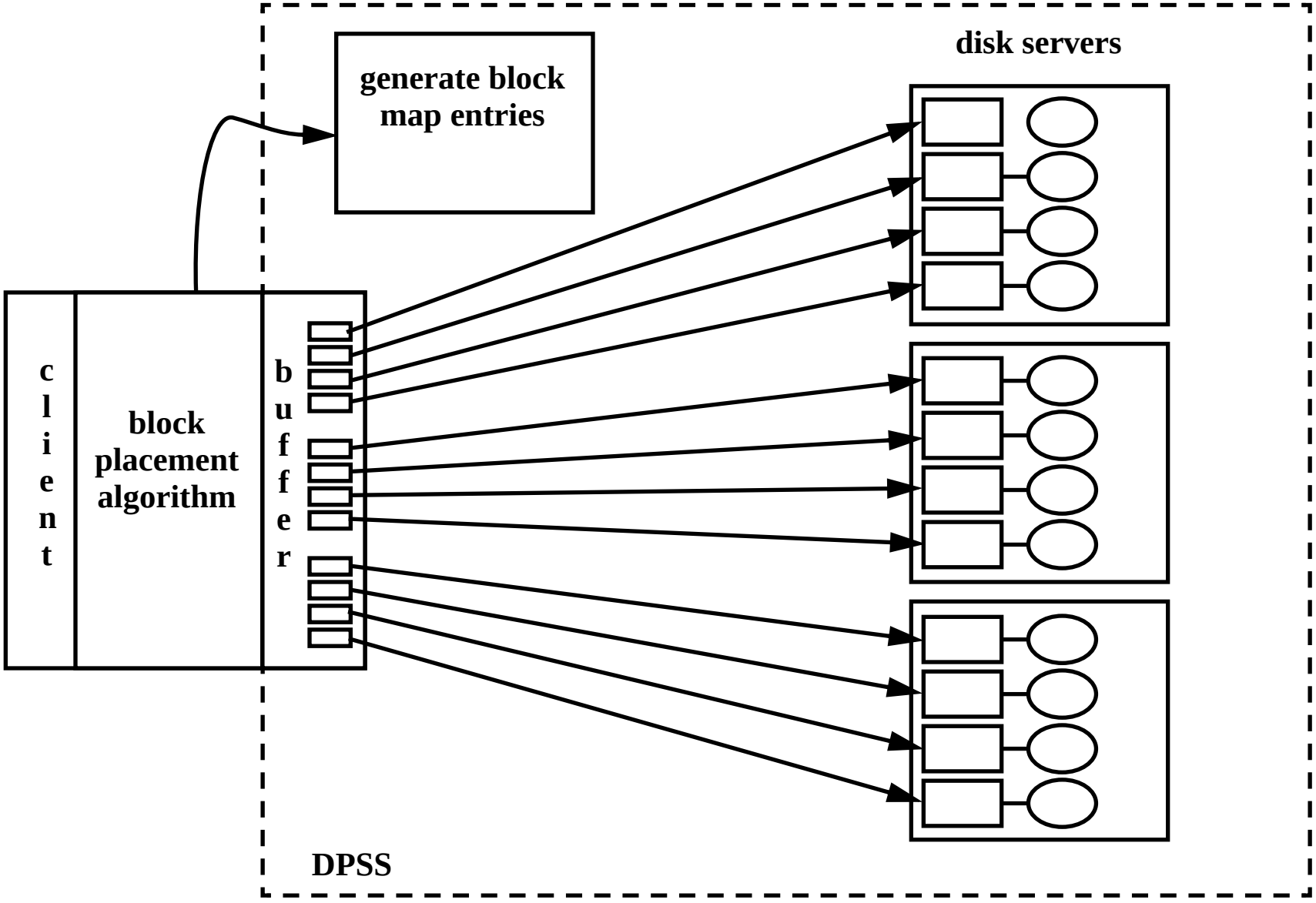
Architecture for Distributed-Parallel Storage System

DPSS Architecture



Distributed-Parallel Storage System Model (Reading)

DPSS Architecture



DPSS model for high-speed writing

DPSS Characteristics cont.

- implementation based on POSIX threads to achieve completely asynchronous I/O
  - Every resource (disk, network interface) on both servers and client libraries has its own thread of control
- Client library provide various application access semantics
  - e.g: (x,y,res,band); (x,y,z,t); frame; offset
- Data Replication for reliability and performance (agent managed)
- three major software components: storage and control, security, agent-based management
- Data block request semantics to support application data prediction (prioritized request lists)
- strong, public-key cryptography based, security an integral part of the design

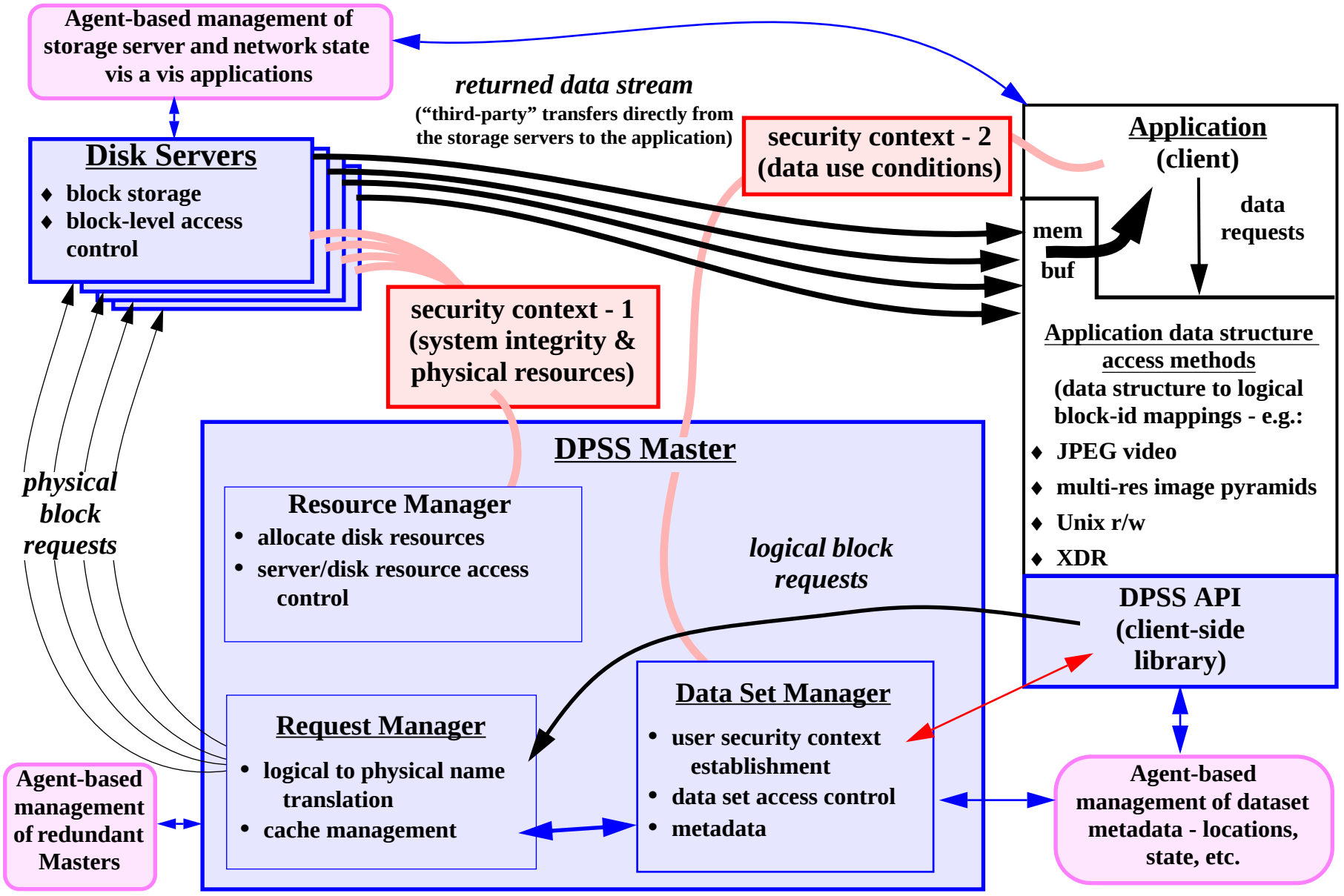
DPSS Architecture

DPSS Characteristics cont.

- new data request cancels currently unsatisfied (flushes disk read and network write queues, but not memory cache)
- writing mechanism provides complete freedom of data layout,
  - a high degree of disk-level parallelism provides a significant fraction of optimal layout latency
- client does not have to wait for the entire file to be transferred from tertiary storage to the DPSS before it can start reading the blocks. It can start to access blocks as soon as they are on the DPSS.

The DPSS is NOT a reliable tertiary storage system!

DPSS Architecture



Distributed-Parallel Storage System Architecture (data reading illustrated)

DPSS Architecture

Typical DPSS implementation

- 4 - 5 UNIX workstations (e.g. Sun Ultra I, Pentium 200)
- 4 - 6 fast-SCSI disks on multiple 2 - 3 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 400 Mbits/s (50 MBy/s) using these relatively low-cost, “off the shelf” components by exploiting the parallelism provided by approximately five hosts, twenty disks, ten SCSI host adaptors, and five network interfaces.

Building a "balanced" DPSS system

- Ideally, there should be enough disks to saturate each SCSI bus, and enough SCSI buses to saturate the network interface.

♦ Example 1:

- Network = Fast Ethernet = 85 Mb/s (10.6 MB/s); SCSI = Fast-SCSI = 8.5 MB/s; disks = Seagate Hawk = 3.5 MB/s
- need 3 disks per SCSI and 2 SCSI buses to guarantee the ability to saturate the network

♦ Example 2:

- Network = ATM OC3 = 110 Mb/s (13.8 MB/s); SCSI = Fast-Wide-SCSI = 18 MB/s; disks = Seagate Barracuda = 5 MB/s
- need 4 disks per SCSI and 1 SCSI bus to saturate the network

DPSS Architecture

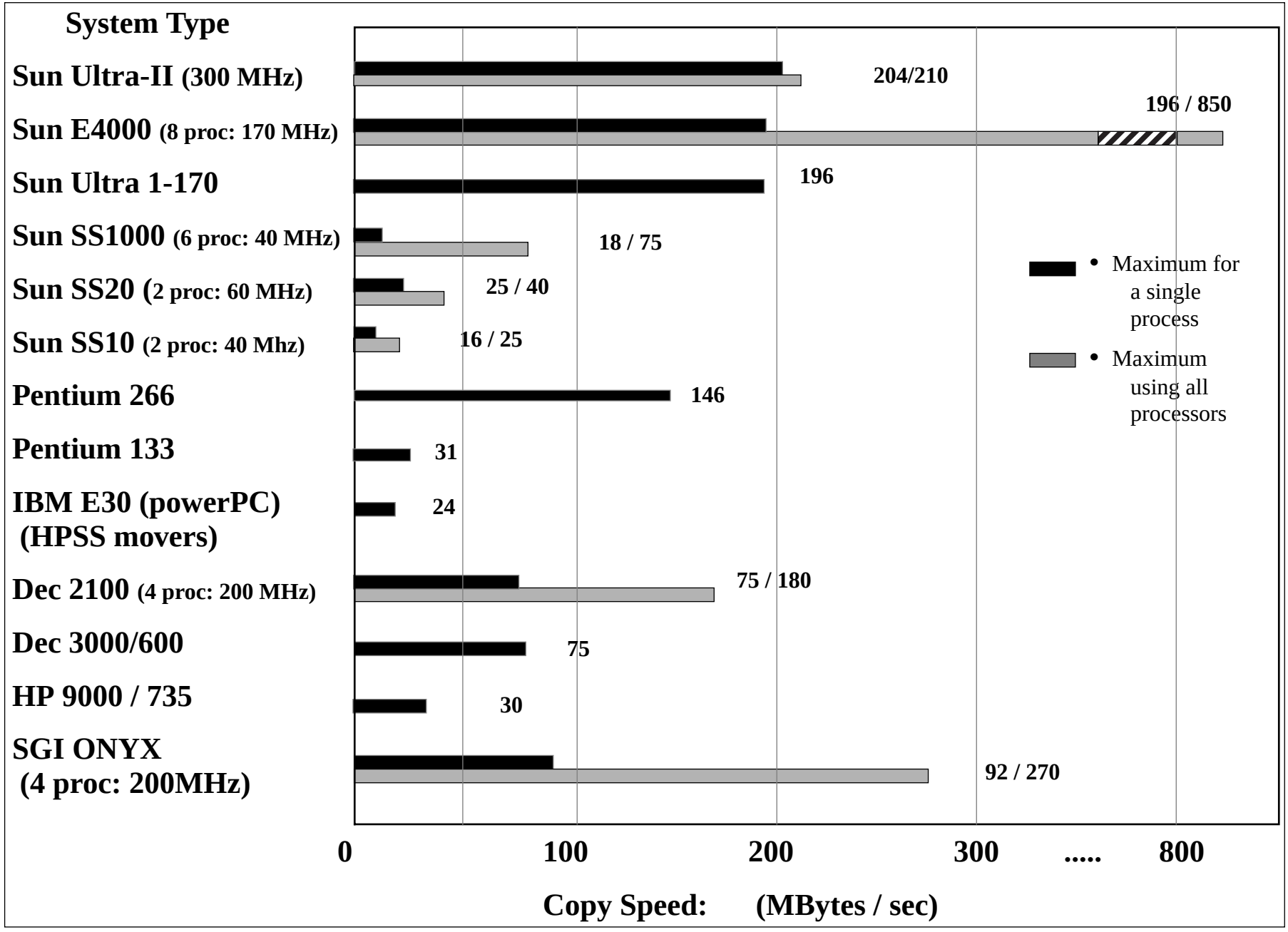
- **Example 3:**
  - **Network = ATM OC-12 = 400 Mb/s (50 MB/s); SCSI = Ultra-wide-SCSI = 35 MB/s; disks = Seagate Cheetah = 10 MB/s**
  - **need 4 disks per SCSI and 2 SCSI buses to guarantee the ability to saturate the network**

- **Note: Can use more disks to increase capacity, but may decrease performance slightly due to overhead of extra threads**

Memory Copy Speed

- In the previous generation of UNIX workstations, memory copy speed was the bottleneck
  - TCP is never better than 50% of the memory-to-memory copy speed (i.e.: a TCP stream in a SS10 will never exceed 8 Mbytes/sec, or 64 Mbits/sec)
- With the newest generation of hardware (i.e: Sun Ultra E4000), this is no longer the case
  - i.e.: a single TCP stream might now be as fast as 780 Mbits/sec, but we are only seeing 400 Mbits/sec, so something else is now the bottleneck (probably CPU)

Hardware Issues



Linux vs. FreeBSD vs. Solaris x86 vs. NT?

- **Performance**
  - **Solaris X86 up to 50% slower than FreeBSD**
  - **Linux threads thrash when heavily loaded**
  - **poor Linux fast ethernet performance**
- **Reliability**
  - **user-level pthreads broken in FreeBSD**
  - **NT crashes often**
- **Portability:**
  - **OpenNT package make porting to NT fairly easy, but does not handle pthreads**

But all are getting better...

Linux seems to be the best at the moment.

Building a low cost Terabyte System

- Costs:
  - can now get 23 GB disks for $4500 each ($200/GB)
  - Assume each system has 128 MB RAM and Fast-Ethernet adaptor; and 2 fast-wide SCSI adaptors
  - Assume 6 disks per server, and 8 servers
  - Total disk cost = 216K
- generic Pentium system: $1300
- Sun Ultra-I: $5800
- Total cost:
  - Pentium system: 8 servers = $10.5K + $216K = $227K
  - Sun System: 8 servers = $46K + $216K = $262K

DPSS Hardware

- **Sun-based system is only about 15% more than the Pentium-based system**
  - **added reliability and stability of Solaris probably make the Sun system a better choice.**

BUT:

- **What if we assume Gigabit ethernet-based systems?**
  - **Might want to use 2 Ultra-II systems with 24 disks each instead?**

- **Determining the optimal DPSS hardware configuration is a constantly moving target!**

Other DPSS Applications

- **Medical Application: video angiography**
  - **CRADA with Kaiser Hospital**
  - **collects data from a remote medical imaging system - operates automatically 10 hours/day, 5-6 days/week with data rates of about 30 Mbits/sec during the data collection phase (about 20 minutes/hour).**
  - **automatically processes, catalogues, and archives 10 GBytes/ day**

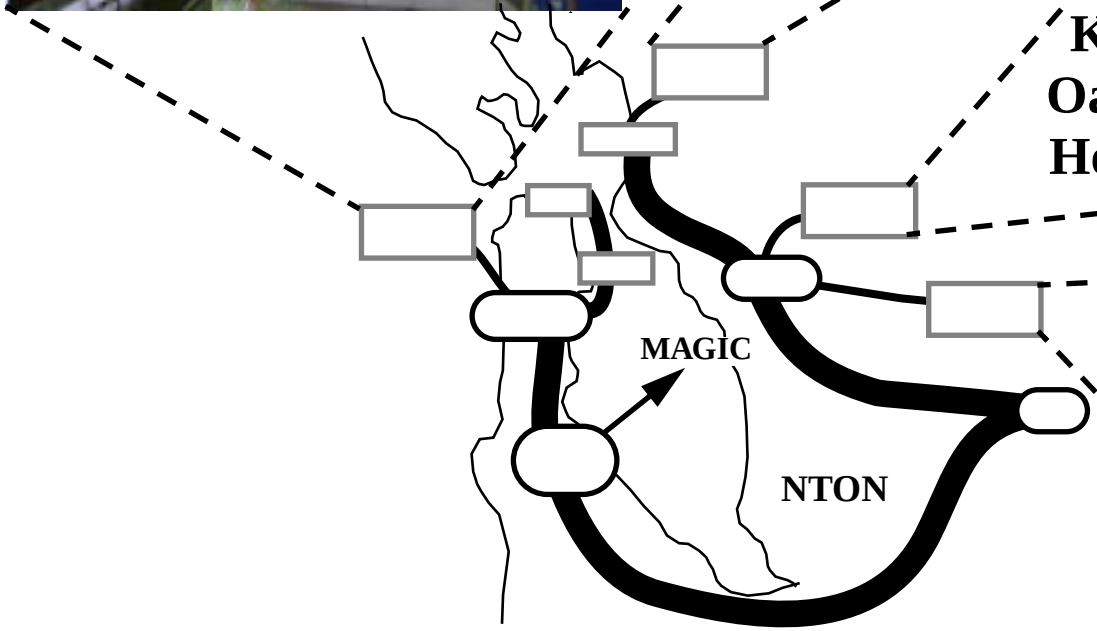
Applications

Kaiser San Francisco Hospital
Cardiac Catheterization Lab

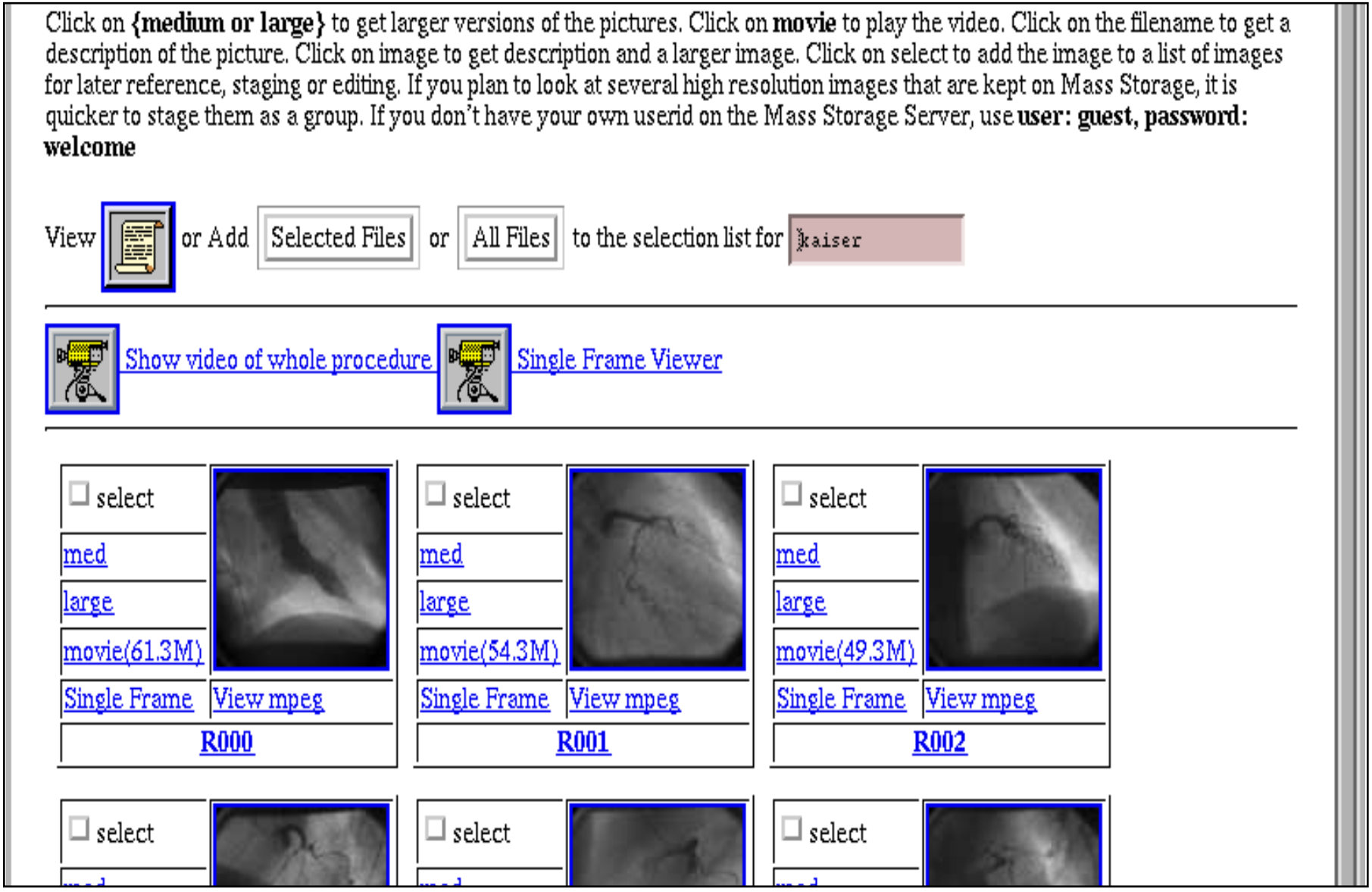
LBNL
WALDO and
DPSS

Kaiser
Oakland
Hospital

Kaiser
Division
of
Research



Applications



A representation of the six sub-objects (about 0.75 GBy total) resulting from a single cycle of operation of a remote, on-line cardio-angiography system

Figure 4

High Energy and Nuclear Physics Distributed Data

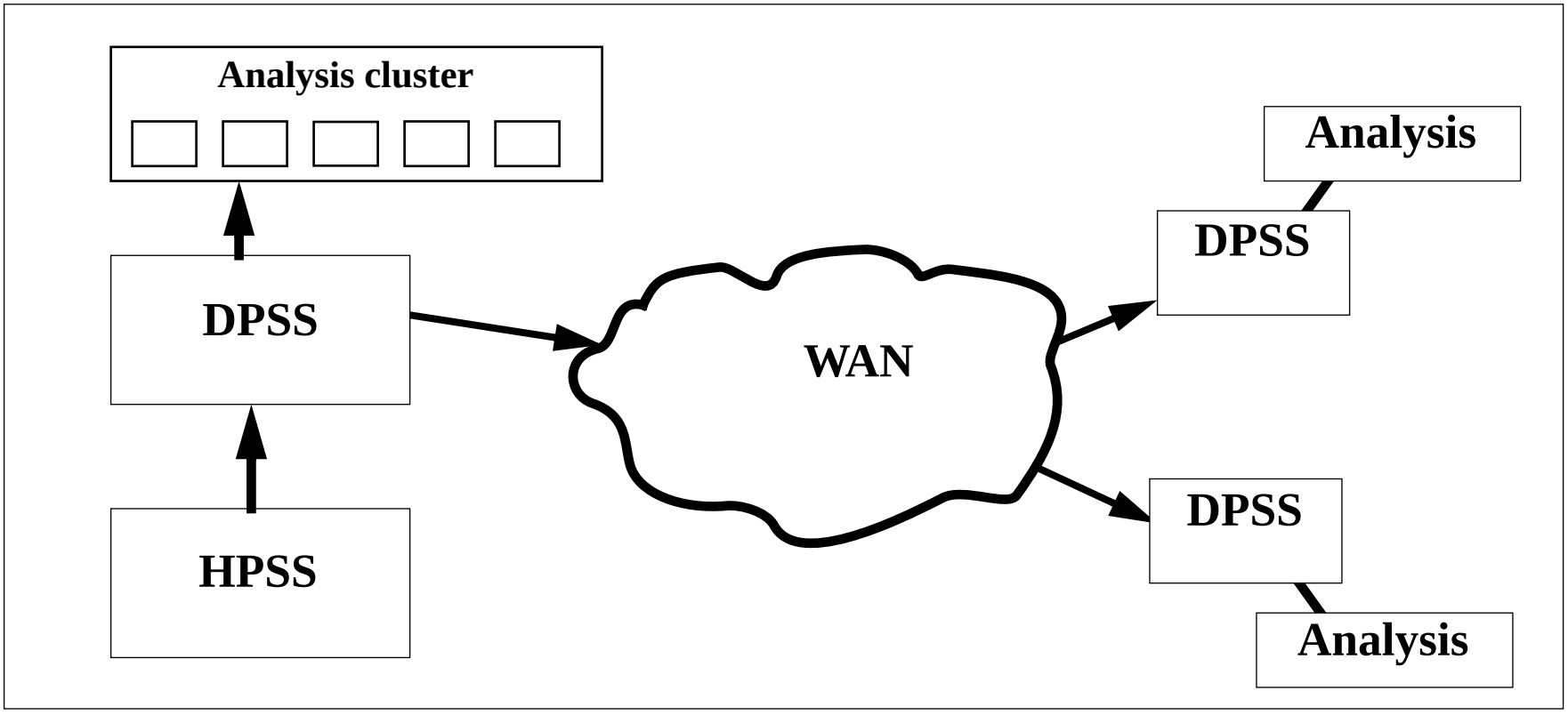
Data source:

- **The STAR detector at RHIC (Brookhaven National Lab). This detector puts out a steady state data stream of 20-40 MBytes/ second. (1.7 TB / day)**

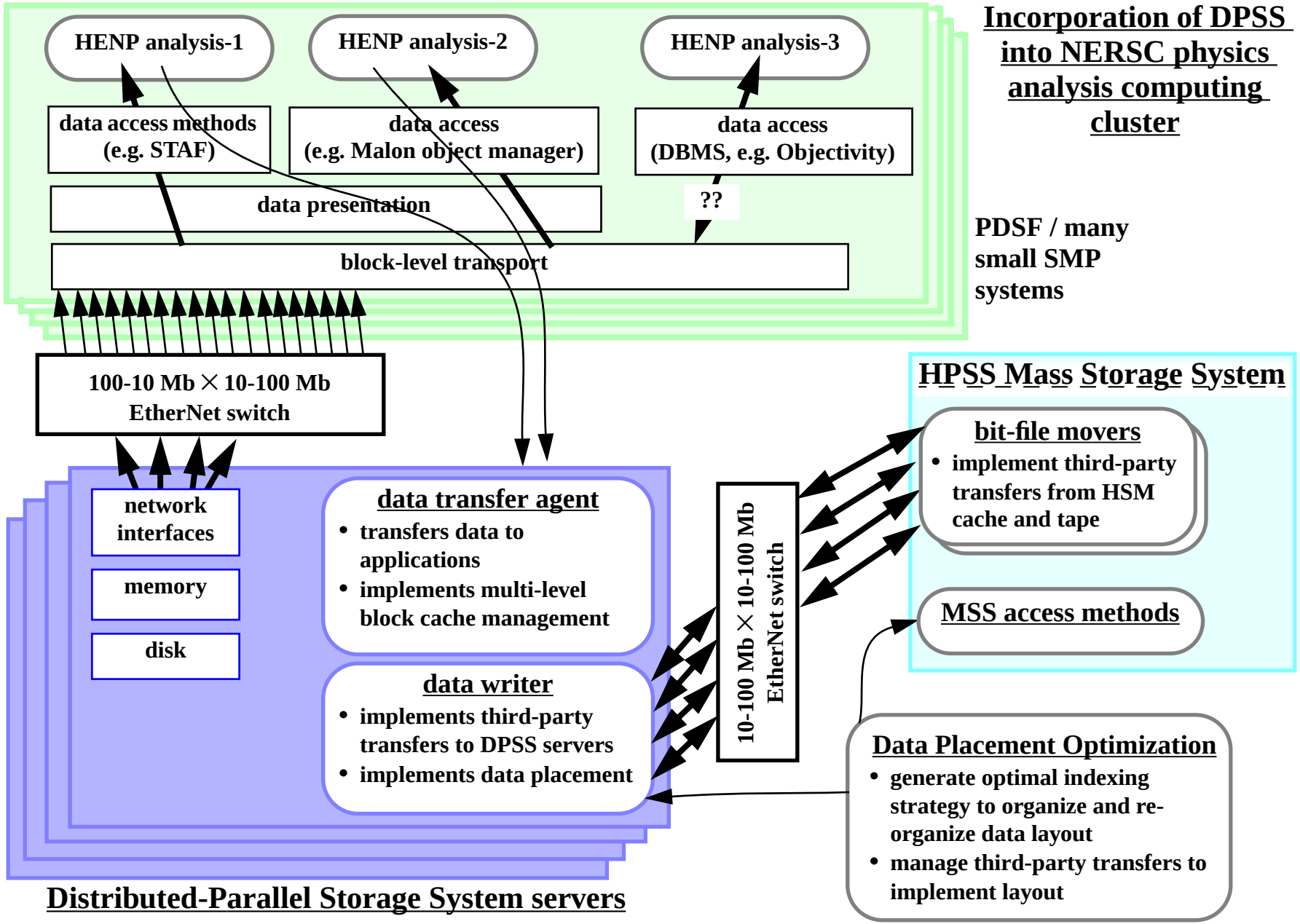
♦ **Application Architecture and Implementation: (see figures)**

DPSS Applications - HENP

- The DPSS is used as a large “window” into the tape-based tertiary storage system
- remote DPSS’s may or may not make sense, depending of the network speed and whether or not the data is used more than once.



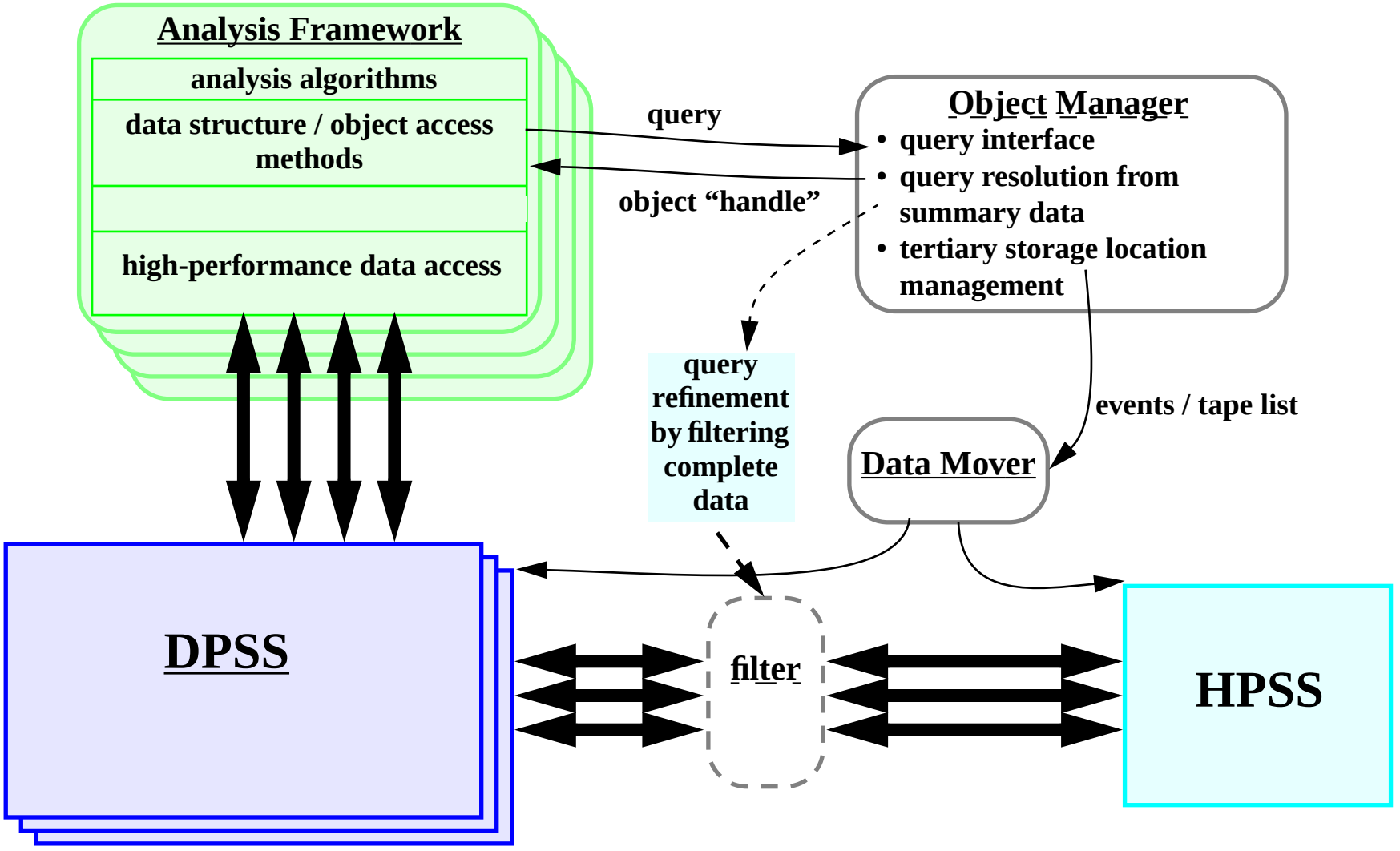
DPSS Applications - HENP



Incorporation of DPSS into NERSC physics analysis computing cluster

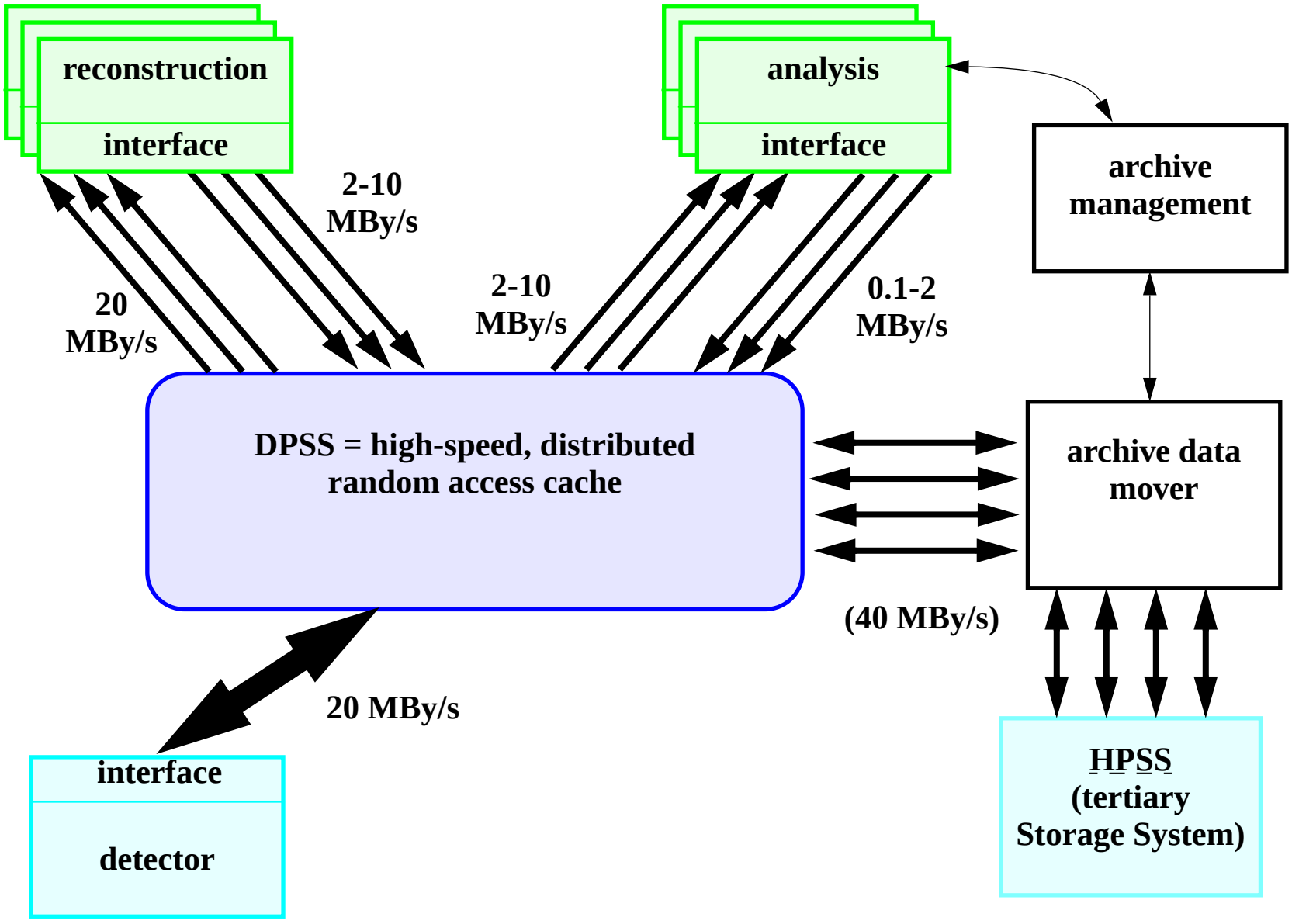
PDSF / many small SMP systems

DPSS Applications - HENP



DPSS based HENP data processing architecture

DPSS Applications - HENP



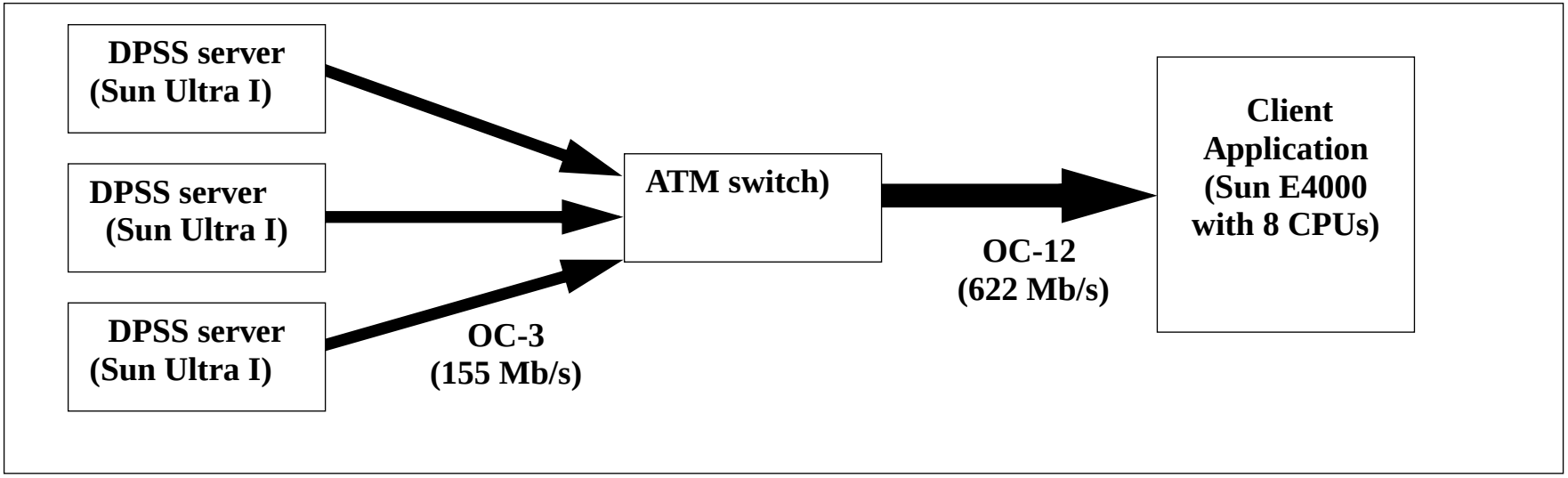
STAR data flow characteristics and prototype data handling model

DPSS Applications - HENP

- **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.**
  - **provides an "impedance" matching function between the coarse-grained nature of parallel tape drives and the fine-grained access of hundreds of applications.**

DPSS Applications - HENP

Prototype Experiment:



- **1 STAF client: 22 MBytes/sec (176 Mbits/sec) for reading data, and 30 MBytes/sec (240 Mbits/sec) for writing data**
- **20 STAF clients: 1 MByte / second per client: 20 MB/sec aggregate data delivery**
- **This system is capable of moving approximately 2 TeraBytes of data per day!**

Other Clients

- **EROS Data Center (USGS)**
  - **very large image viewer: provides the ability to quickly pan and zoom over 1-2 GB images**
- **Mbone session recorder/player**
  - **will demo this at SC '97**

Outline

- Introduction
- History: The Magic Gigabit Network Testbed and TerraVision
- DPSS Architecture and Implementation
- DPSS hardware issues
- DPSS Clients
  - HENP
  - Medical Images
- Performance Analysis and Monitoring: NetLogger Toolkit
- Current and Future Work
  - Agents / Brokers

Performance Monitoring Approach and Experiments

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

  The bottlenecks can (and have been) in any of the 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**

Monitoring

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 wide-area network 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 every significant event involved in distributed systems data interchange.

- **Has proven invaluable for isolating and correcting performance bottlenecks, and even for debugging distributed parallel code**
- **It is difficult to track down performance problems because of the complex interaction between the many distributed system components resulting in problems in one place being most apparent somewhere else.**

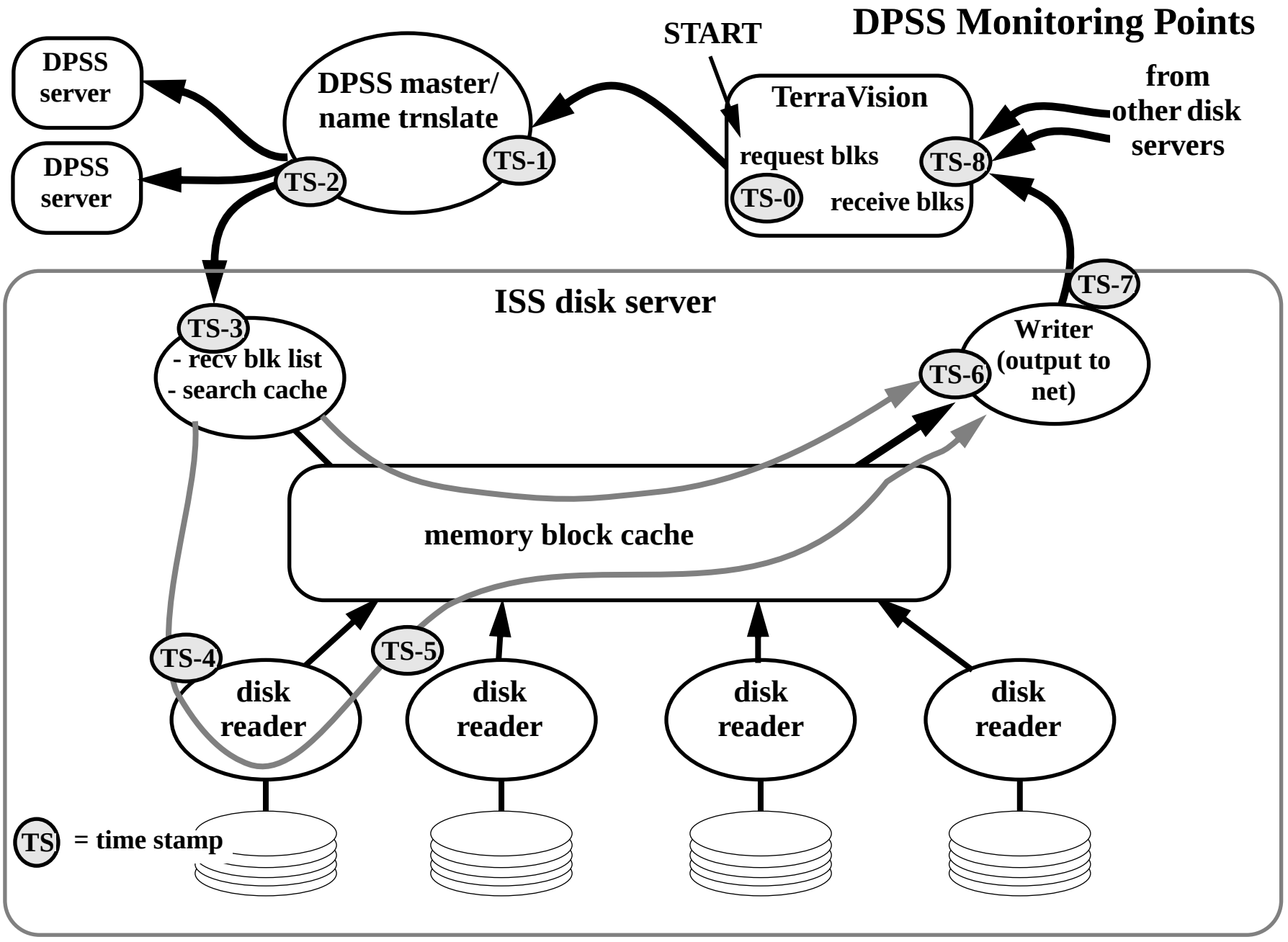
Monitoring

- **Network performance tools such at *ttcp* and *netperf* 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.**
- **To address this, we have developed the NetLogger Toolkit: a methodology and set of tools to aid in creating graphs that trace a data block through a distributed system**
- **This allows us to determine exactly what is happening within this complex system.**

Monitoring

- **Netlogger Components:**
  - **Use a common log format**
  - **Collect state information and time stamps at all critical points in the data path, including instrumenting the clients and applications (see figure).**
  - **Timing information is carried as a defined part of the data block structure, auxiliary information is logged and correlated based on precision timestamps**
  - **Modified some Unix network and OS monitoring tools to log “interesting” events using the same log format**
  - **The results are plotted as detailed, data block transit history “life-lines”**

Monitoring


DPSS Monitoring Points
START
DPSS server
DPSS server
DPSS master/ name trnslate
TS-1
TS-2
TerraVision
request blks
TS-0
receive blks
TS-8
from other disk servers
ISS disk server
TS-3
- recv blk list
- search cache
TS-7
Writer (output to net)
TS-6
memory block cache
TS-4
TS-5
disk reader
disk reader
disk reader
disk reader
TS = time stamp

Monitoring

- **OS and network monitoring include:**
  - **TCP retransmits**
  - **CPU usage (user and system) and CPU interrupts**
  - **AAL 5 information**
  - **ATM switch buffer overflows**
  - **ATM hosts adapter buffer overflow**
- **Tools**
  - ***netstat* and *vmstat* modified to poll and report continuously**
    - **(Currently poll at 100 ms, so data is accurate to +/- 50 ms)**
  - **SNMP queries to switches and network interfaces**
    - **switch buffer overflows**
  - **(unfortunately SNMP time resolution is too coarse to be very useful)**

Monitoring

- **Common logging format:**

  keyword; hostname; seconds; nano-sec; data; data; data;.....;

  - **“keyword” - an identifier describing what is being logged - e.g. a reference to the program that is doing the logging (i.e):**
    DPSS_SERV_IN, VMSTAT_SYS_CALLS,
    NETSTAT_RETRANSSEGS, TV_RQ_TILE
  - **“data” elements (any number) are used to store information about the logged event - for example:**
    - **NETSTAT_RETRANSSEGS events have one data element: the number of TCP retransmits since the previous event**
    - **DPSS_START_WRITE events have data elements containing: the logical block name, the data set ID, a “user session” ID, and an internal DPSS block counter**

Monitoring

NETSTAT_OUTDATASEGS; 806706009; 652917; 4817378;
NETSTAT_RETRANSSEGS; 806706009; 652917; 16395;
NETSTAT_RETRANSBYTES; 806706009; 652917; 111841701;
NETSTAT_OUTDATASEGS; 806706009; 760199; 0;

VMSTAT_INTR; 806706009; 546568; 71612733;
VMSTAT_SYS_CALLS; 806706009; 546568; 2794466576;
VMSTAT_USER_TIME; 806706009; 546568; 2;
VMSTAT_SYS_TIME; 806706009; 546568; 5;
VMSTAT_INTR; 806706009; 646444; 3;

APP_SENT; 824951202; 824949; 34; 78; 45; 0; 6; 5; 198.207.141.6;
DPSS_MASTER_IN; 824951202; 832232; 34; 78; 45; 0; 6; 5; 198.207.141.6;
DPSS_MASTER_OUT; 824951202; 865724; 34; 78; 45; 0; 6; 5; 198.207.141.6;
DPSS_SERV_IN; 824951202; 877494; 34; 78; 45; 0; 6; 5; 198.207.141.6;
DPSS_START_READ; 824951202; 885279; 34; 78; 45; 0; 6; 5; 198.207.141.6;
DPSS_END_READ; 824951202; 909439; 34; 78; 45; 0; 6; 5; 198.207.141.6;
DPSS_START_WRITE; 824951202; 910743; 34; 78; 45; 0; 6; 5; 198.207.141.6; 49264
APP_RECEIVE; 824951210; 914210; 34; 78; 45; 0; 6; 5; 198.207.141.6;
APP_SENT; 824951202; 824949; 34; 76; 45; 0; 6; 3; 198.207.141.6;

Network Time Protocol[1]

- For NetLogger timestamps to be meaningful, all systems clocks must be synchronized
- NTP is used to synchronize time stamps throughout MAGIC
  - All MAGIC hosts run *xntpd*, which synchronizes the clocks of each host both to time servers and to each other
  - The MAGIC backbone segments are used to distribute NTP data, allowing us to synchronize the clocks of all hosts to within about 250 microseconds of each other, but...
    - Many different sys admins (harder to synchronize than clocks)
    - The systems have to stay up for a significant length of time for the clocks to converge to 250 μs

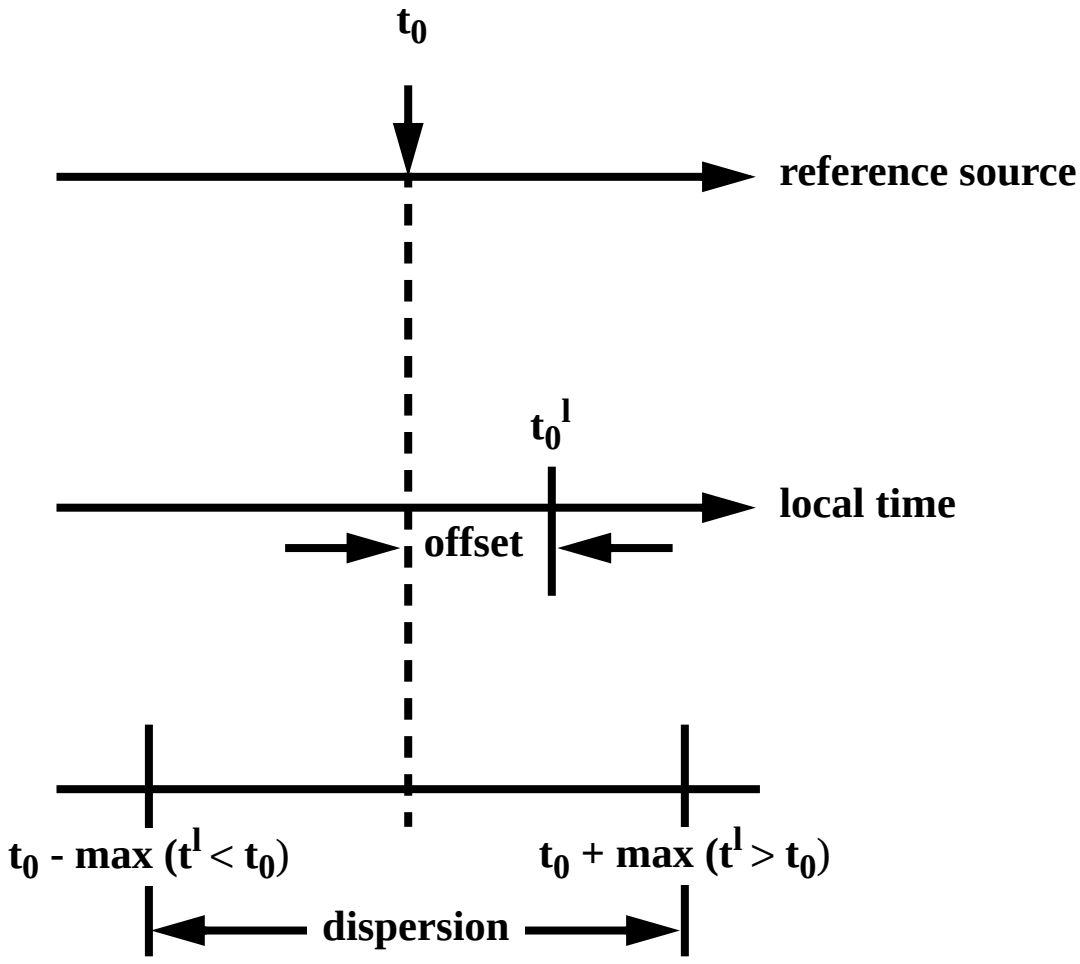
1. "Network Time Protocol: Specification, Implementation and Analysis", D. Mills, U. Delaware, March, 1992. Internet RFC-1305

NTP

- **Purpose of NTP**
  - **To convey timekeeping information from the primary servers to other time servers via the Internet**
  - **To cross-check clocks and mitigate errors due to equipment or propagation failures**
- **Some number of local-net hosts or gateways, acting as secondary time servers, run NTP with one or more of the primary servers**
- **Most 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

- **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.**



NetLogger

- **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

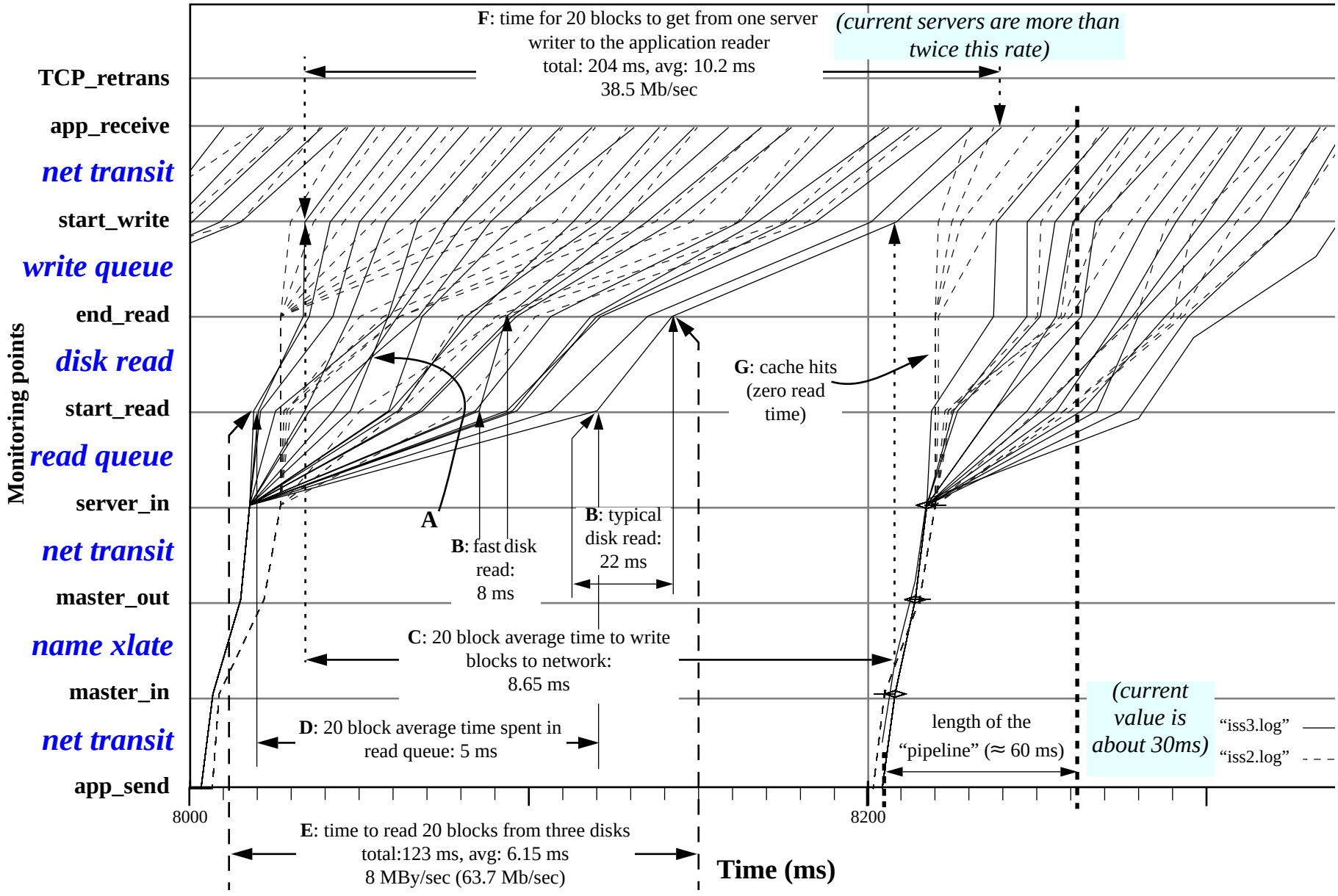
  D: the average time in disk read queue is 5 ms

  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



Two server, ATM *LAN*

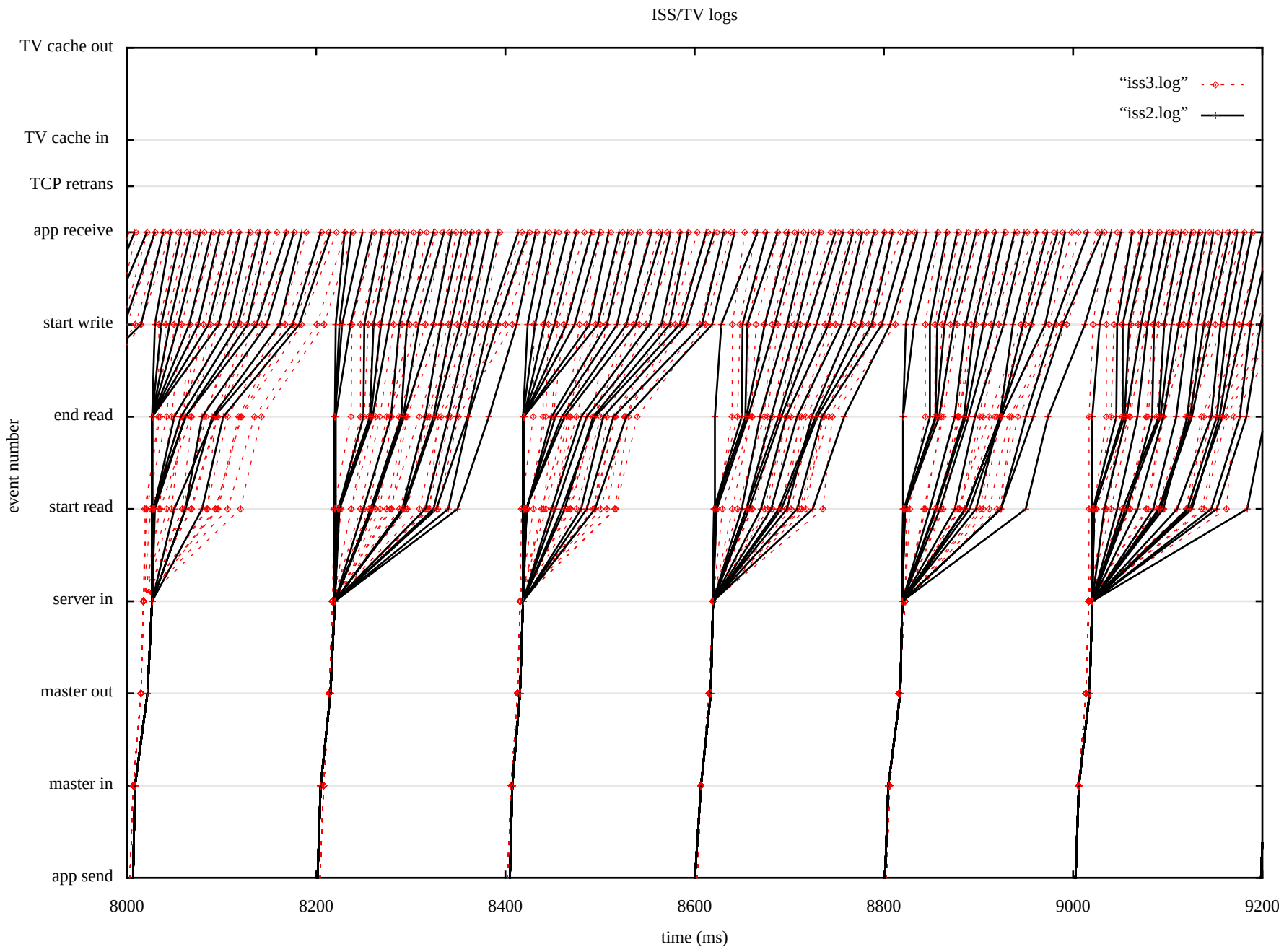
A Case Study: TCP in the Early MAGIC Testbed

- DPSS and TerraVision were working well in a LAN environment, but failing to deliver high (even medium!) data rates to TerraVision when we operated in the MAGIC WAN
- We suspected that the switches were dropping cells, but they reported no cell loss
- Network engineers claimed that the network was working "perfectly"

Performance Analysis

- **Control test: 2 servers over ATM LAN**
  - **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*).**

Performance Analysis

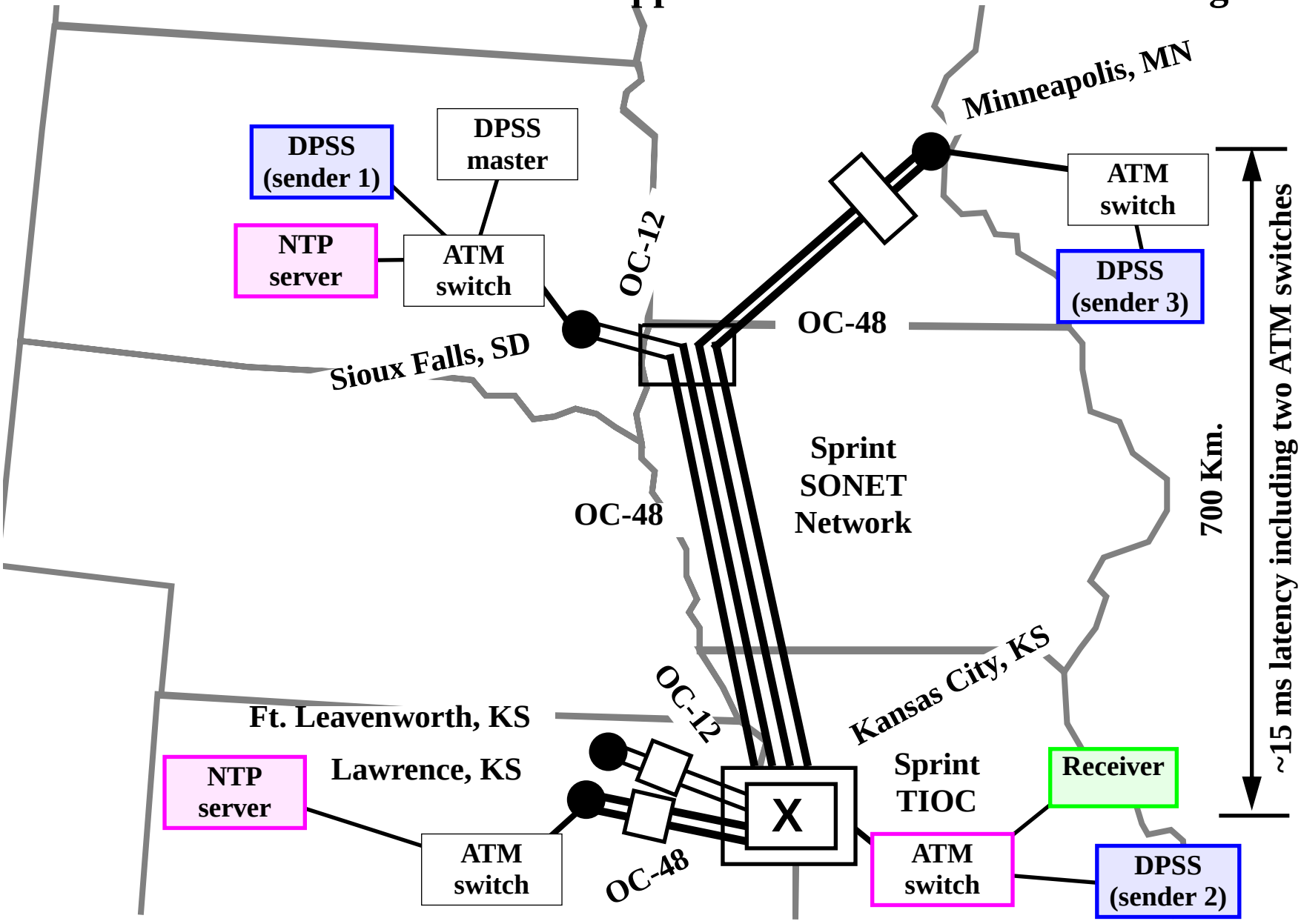


MAGIC WAN experiment:

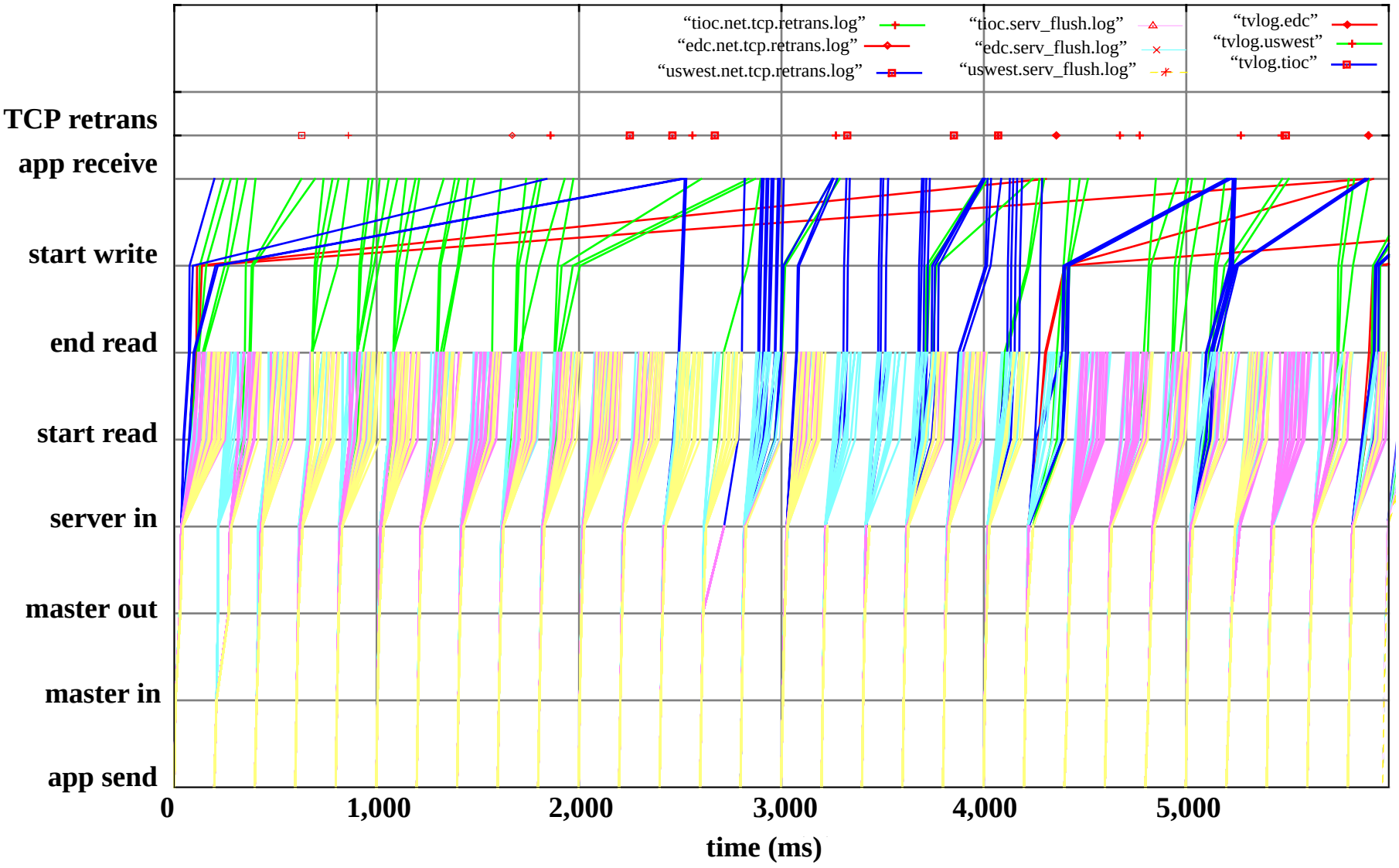
- Three disk server configuration (next figure) DPSS gave results in the following figure:

Performance Analysis

The MAGIC Network and DPSS / Application Performance Test Configuration



Performance Analysis



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

Performance Analysis

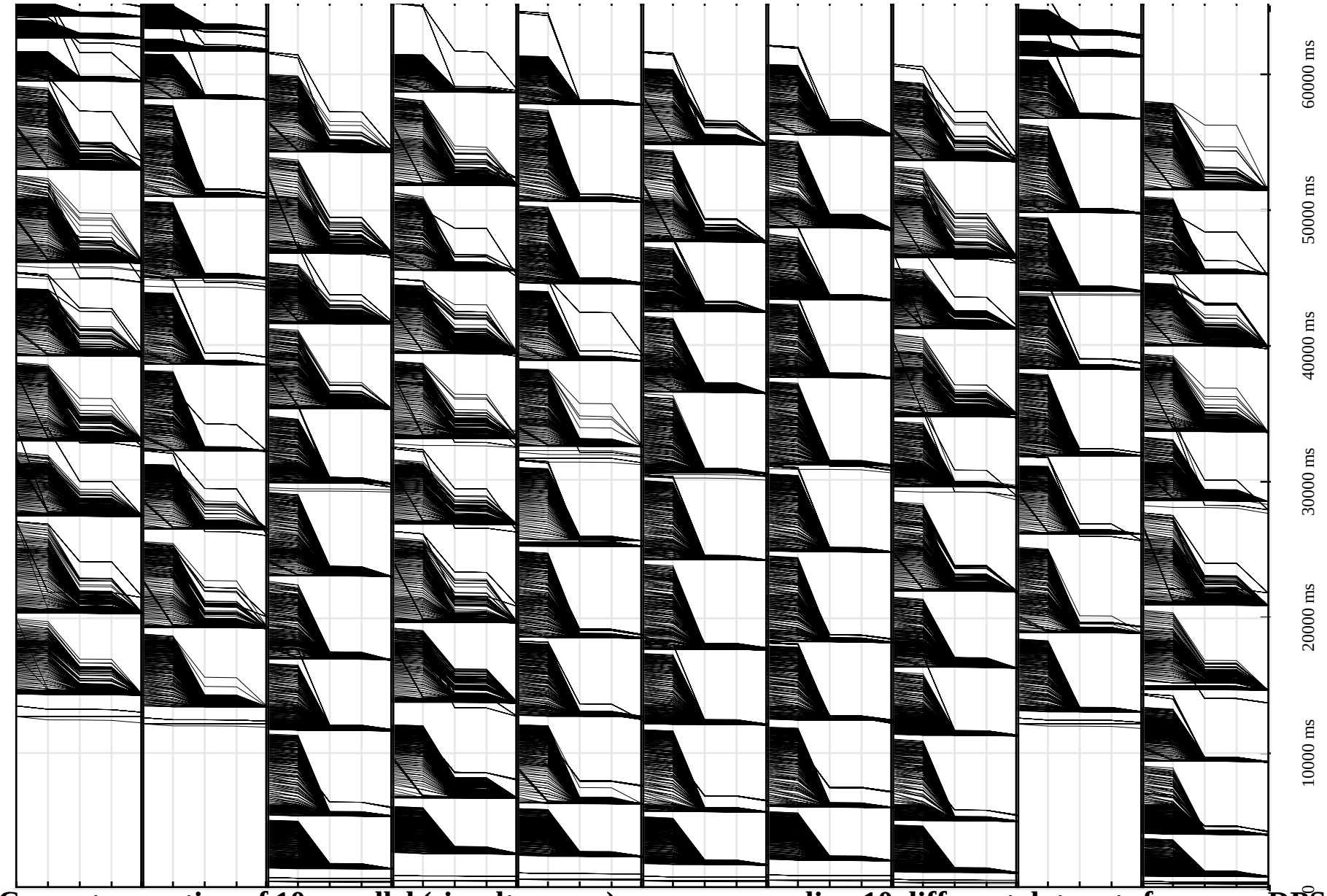
- **What we believe to be happening in this experiment is that TCP's normal ability to accommodate congestion is being defeated by an unreasonable network configuration:**
  - **the final ATM switch where the three server streams come together has a per port output buffer of only about 13K bytes.**
  - **the network MTU (minimum transmission unit) is 9180 Bytes (as is typical for ATM networks)**
  - **the TCP congestion window cannot get smaller than the MTU, and therefore TCP's throttle-back strategy is pretty well defeated: On average, every retransmit fails, even at TCP's "lowest throughput" setting, because this smallest unit of data is still too large for the network buffers.**
  - **Three sets of 9 KBy IP packets are 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 new generation of ATM switches (Fore LC switch modules) have much more buffering: 32K cells (1500 KB), so these switches should not have this problem.

Performance Analysis

- **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**

Performance Analysis



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, total = 1 GBy, *1.5 TBy/day*)

Current Work:

Agent Based Management of Widely Distributed Systems

If comprehensive monitoring is the key to diagnosis, agent based management may be the key to keeping widely distributed systems running reliably.

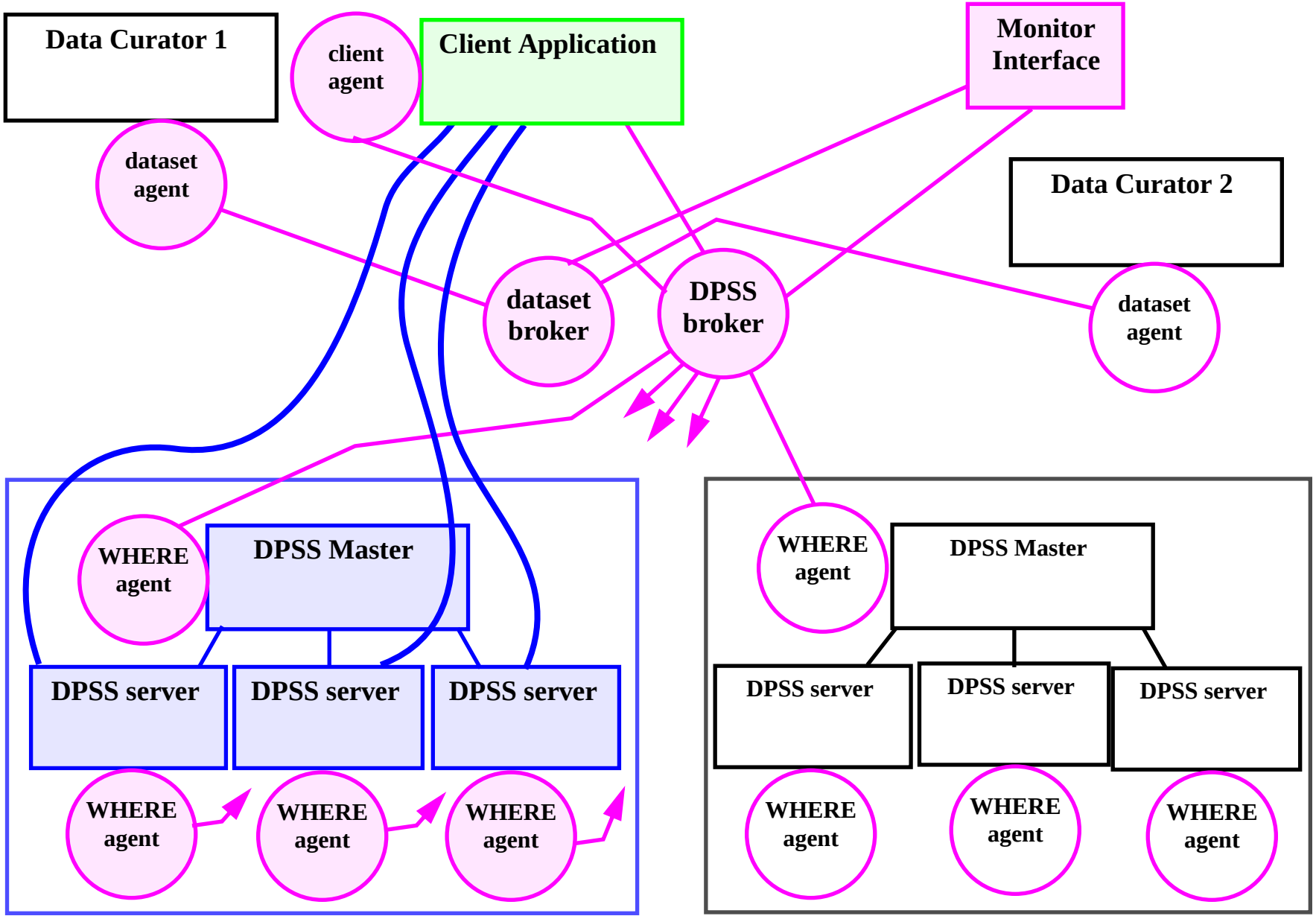
- **"Agents" are**
  - **autonomous**
  - **adaptable**
  - **monitors**
  - **managers**
  - **information aggregators**
  - **KQML based information filters**
  - **implemented in Java**
  - **constantly communicating with peers via IP multicast**

Agent Based Management

Initial use of agents

- Provide structured access to current and historical information regarding the state of the DPSS components
- Keep track of all components within the system and restart any component that has crashed, including one of the other agents (addresses fault tolerance)
- When new components are added, such as a new disk server, the agents do not have to be reconfigured - an agent is started on the new host, and it will inform all other agents about itself and the new server
  - Brokers and agents may discover new agents via SAP (the multicast discovery protocol), when a new agent - and the resource that it represents - is added to the configuration.

Agent Based Management



Agent Based Management

- ***Broker agents*** **manage information from a collection of** ***monitor agents*** **(usually on behalf of a user client)**
  - **agents manage dataset metadata (dynamic state, alternate locations tertiary location) at each storage system**
    - **brokers provide an integrated view of the data**
  - **agents continuously monitor state of all network interfaces and data paths**
    - **brokers analyze this information on behalf of a client to determine which DPSS has the best network connectivity**
  - **agents monitor the load of each DPSS disk server**
    - **broker can analyze to decide which servers to use when data is replicated (addresses high availability)**

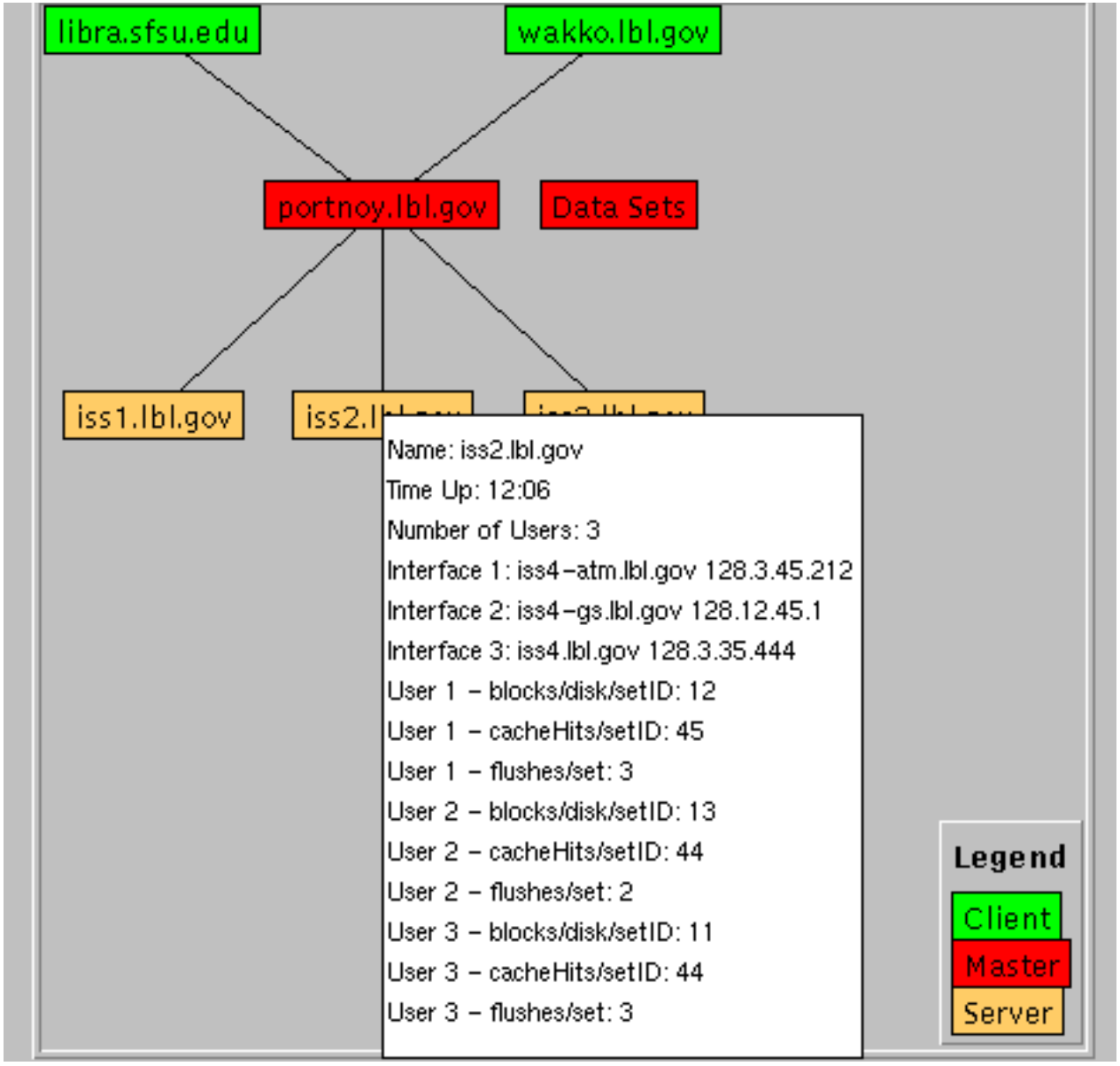
Agent Based Management

- **Brokers can perform actions on behalf of a user**
  - **e.g., if a data set is not currently loaded onto a DPSS (which is typically used as a cache), the broker can cause the dataset to be loaded from tertiary storage**
- **A broker/agent architecture allows the system administrators to separate mechanism from policy**
  - **agent rule-based operation can be used to determine what policies are be enforced while remaining separate from the actual mechanism used to implement these policies**

<u>First demonstration / experiment</u>

An application uses aggregated information from a broker to present an adaptive and dynamic view of the system: data throughput, server state, and dataset metadata as reported by the agents

Agent Based Management



Prototype automatically generated monitor interface (a Web applet) for brokers aggregating information from DPSS system-state and dataset-state monitoring agents

Agent Based Management

- **When you observe that something has gone wrong in a widely distributed system, it is generally too late to react - in fact, you frequently can't even tell what is wrong, because:**
  - **it depends on a history of events**
  - **you can't get at the needed information any more**
  - **it will take too long to ask and answer all of the required questions**

Agents will not only monitor, but keep a state history in order to answer the question "how did we get here?"

- **Active analysis of operational patterns (e.g., pattern analysis of data block lifeline traces) will lead to adapting behavior / configuration to avoid or correct problems**

Other Current Work:

- Integrating the DPSS for use as a cache between the HPSS and the PDSF
- Dynamic reconfiguration of DPSS resources (agent managed)
  - adding and deleting servers and storage resources during operation
- Survivability / Fault Tolerance of DPSS master and servers
- Real-time display and analysis of NetLogger data
- try creating a DPSS client library that is a shared libc replacement for open(), close(), read(), write(), lseek().

- **Transport**
  - **TCP is not the correct answer for many specific applications**
    - **segment retransmission should be optional, as should the ordering (some of this will be addressed with new TCP selective segment re-transmission options)**
  - **Experiment: try various user-level TCP implementations**
    - **e.g.: try using TReno (Mathis/Mahdavi, PSC) implementation of TCP**
      - **build on top of UDP**
      - **does selective acknowledgments (SACK)**
      - **does not retransmit lost packets, only adjusts the window size**

- **For more information see:**

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

  check out our demos at the LBNL booth at SC ‘97