Data Intensive Distributed Computing: A Medical Application Example

Brian L. Tierney (bltierney@lbl.gov)
Jason R. Lee
William E. Johnston

Ernest Orlando Lawrence Berkeley National Laboratory

Outline

- Introduction: Distributed Storage
- Data Architecture
- High Speed Data Cache
  - Distributed Parallel Storage System (DPSS)
- Medical Application
- Physics Application
- How to achieving high TCP throughput

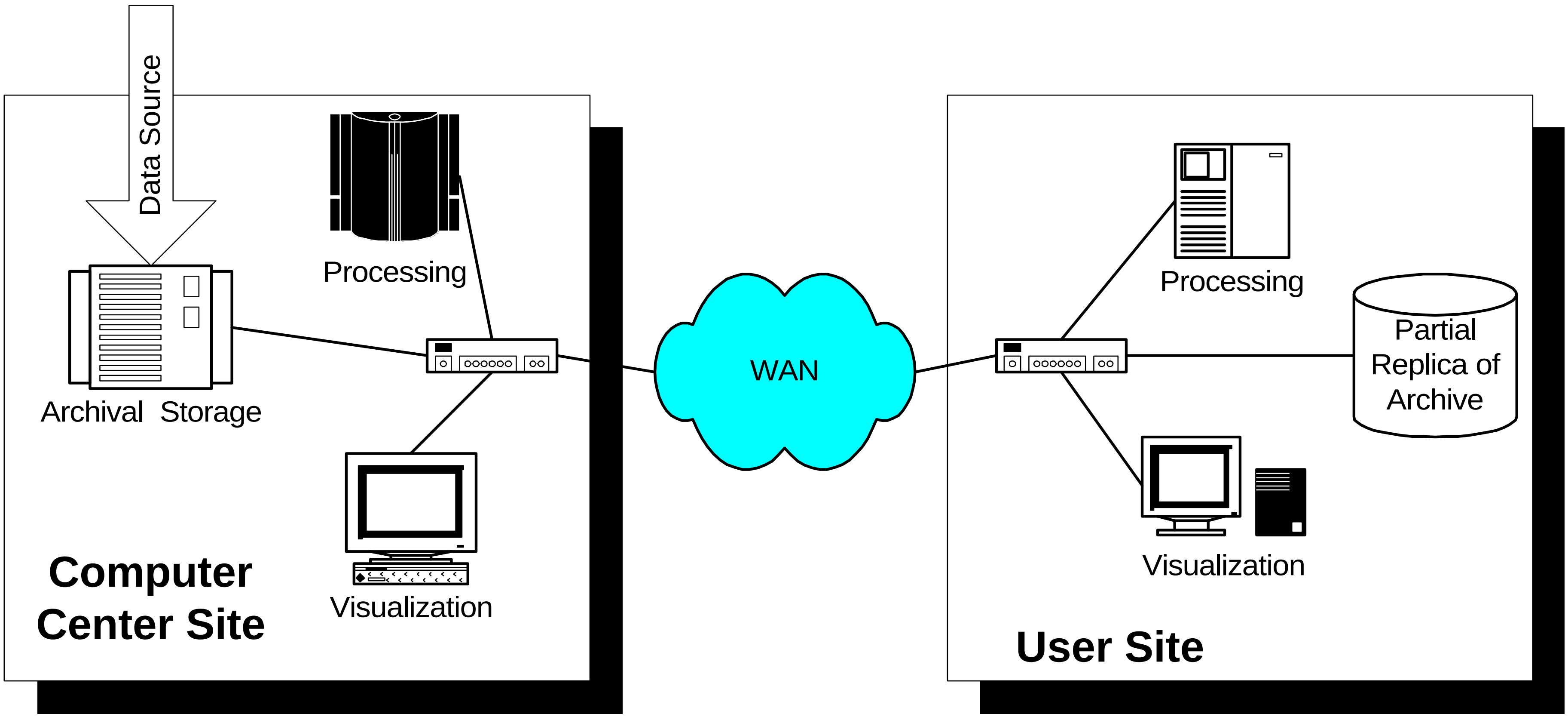
Why Distributed Storage

- **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**

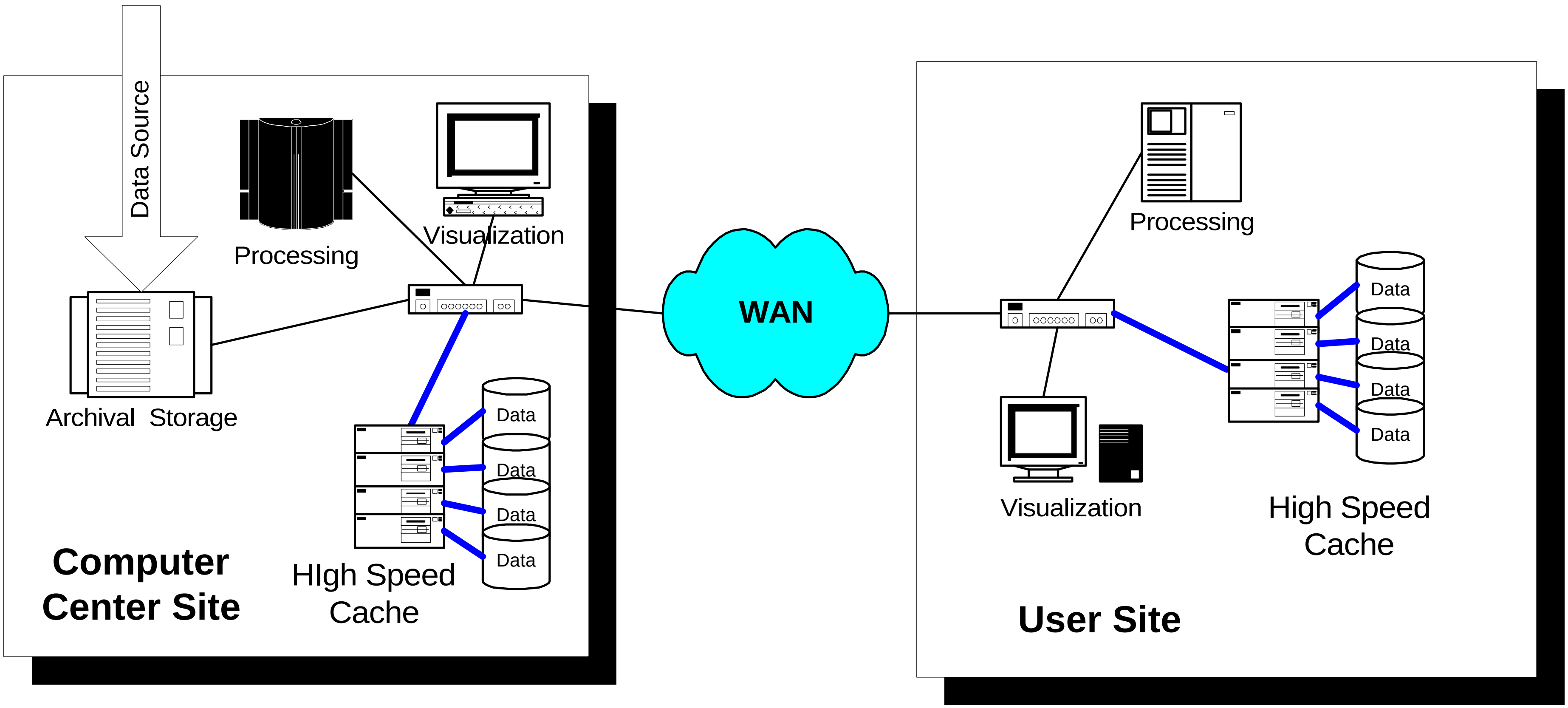
Advantages of Distributed Storage

- 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

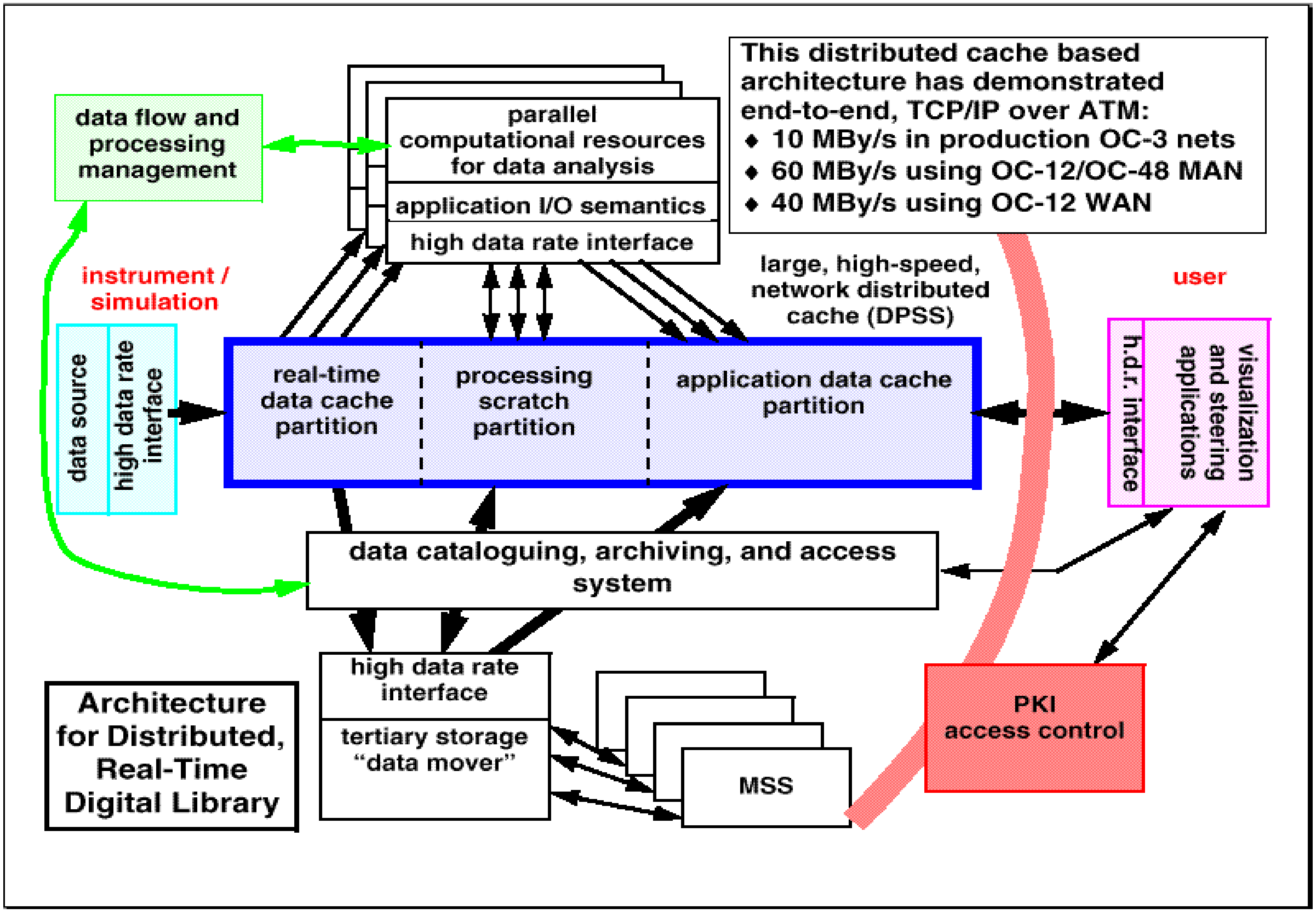
Remote Access to a Large Data Archive



Remote Access to a Large Data Archive using a Cache



Data Architecture



Key Features of the Architecture

- Very high-speed cache that is distributed, scaleable, 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

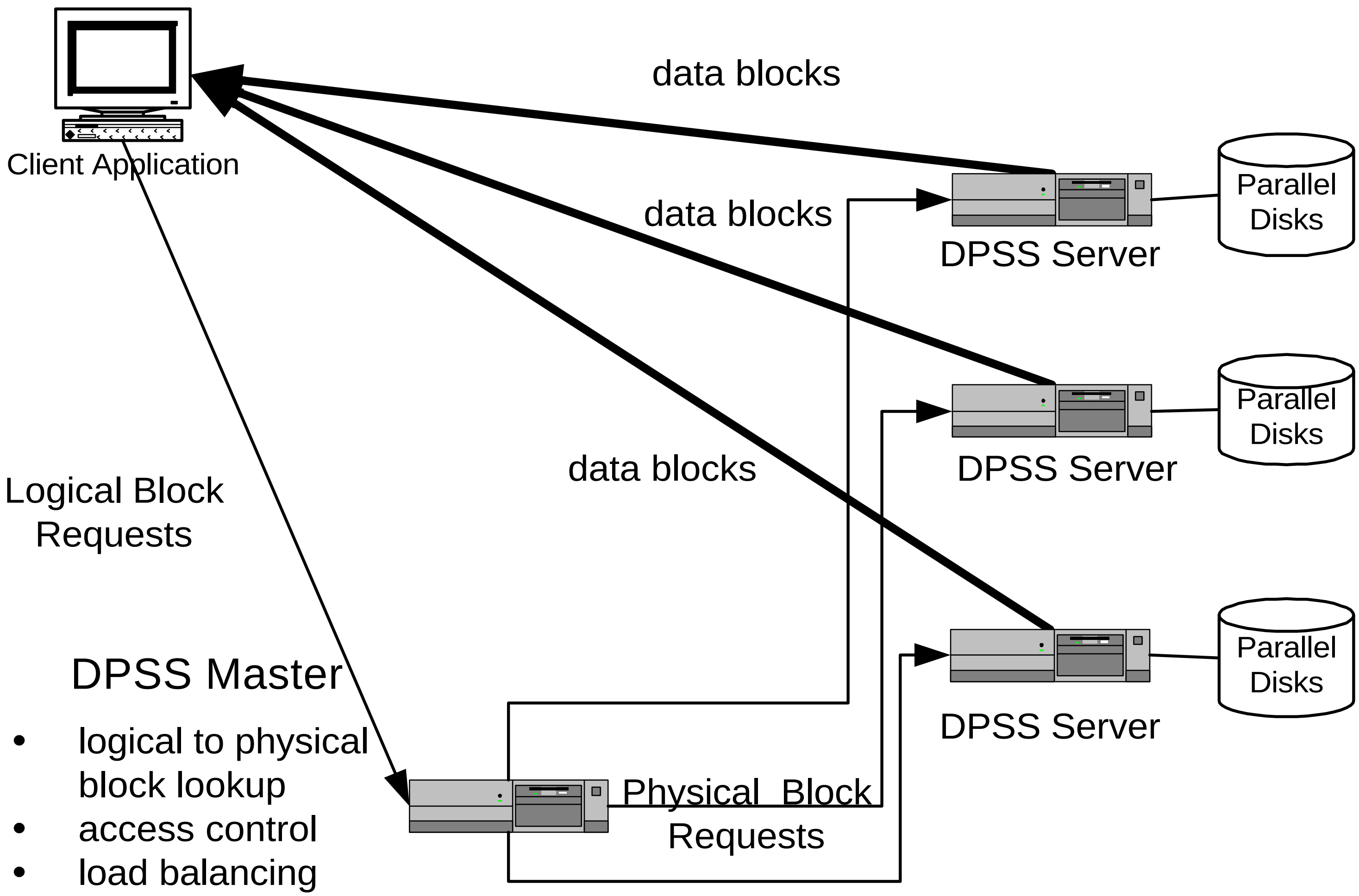
Data Handling Model

- data sources deposit data in cache, and consumers take data from the cache, usually writing processed data back to the cache
- metadata is typically recorded in a cataloguing system as data enters the cache, or after intermediate processing
- a tertiary storage system manager migrates data to and from the cache. The cache can thus serve as a moving window on the object/dataset.
- the native cache access interface is at the logical block level, but client-side libraries implement various access I/O semantics - e.g., Unix I/O

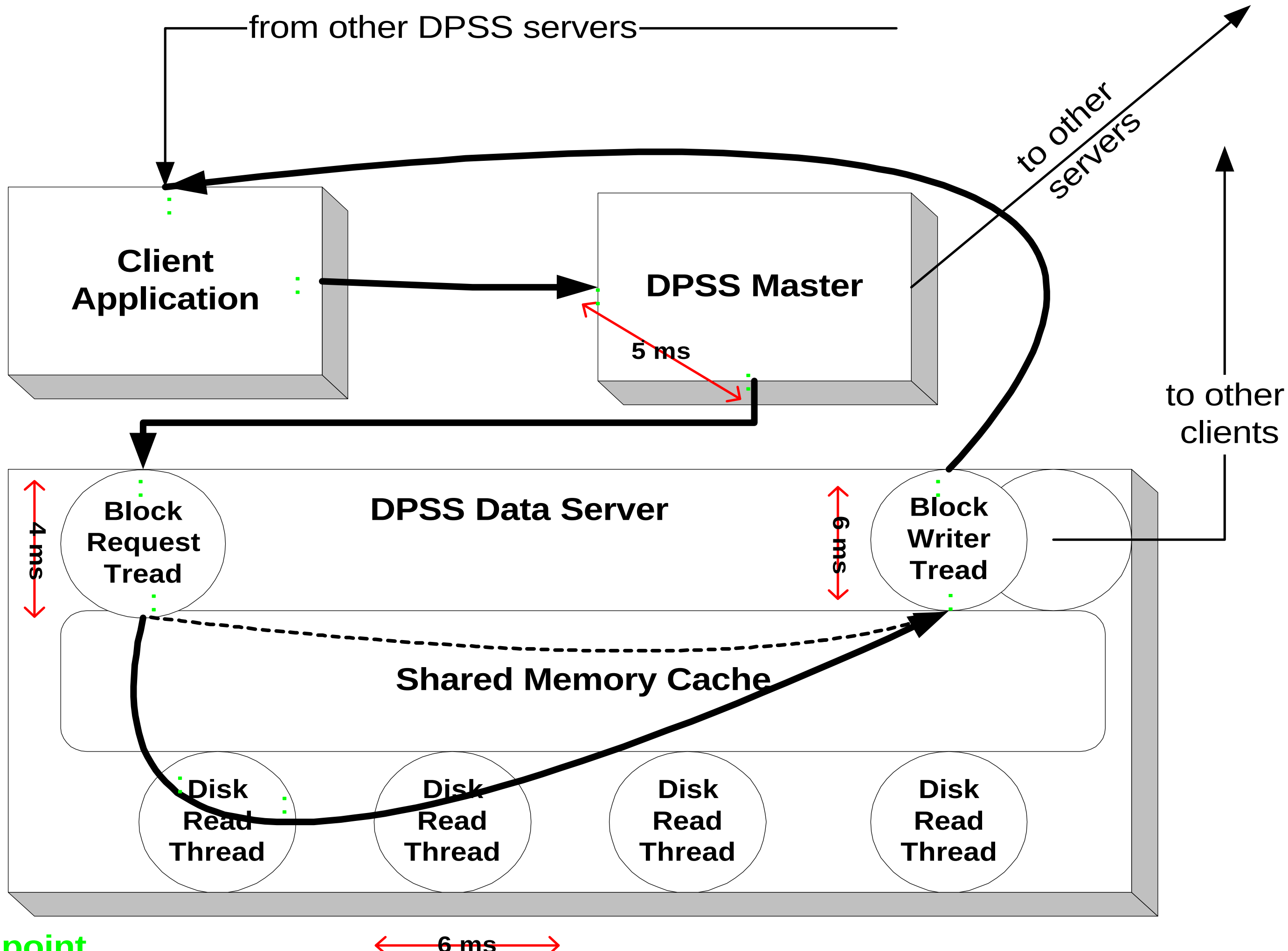
Advantages of this Architecture

- 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
  - using a large tertiary storage system (e.g.: at a supercomputer center) should result in:
    - more economical storage
    - better access (due to more tape robots)
    - better media management

DPSS Cache Architecture



DPSS Architecture



Total Latency for a single block = 25 ms (master + server + network)

: = monitoring point

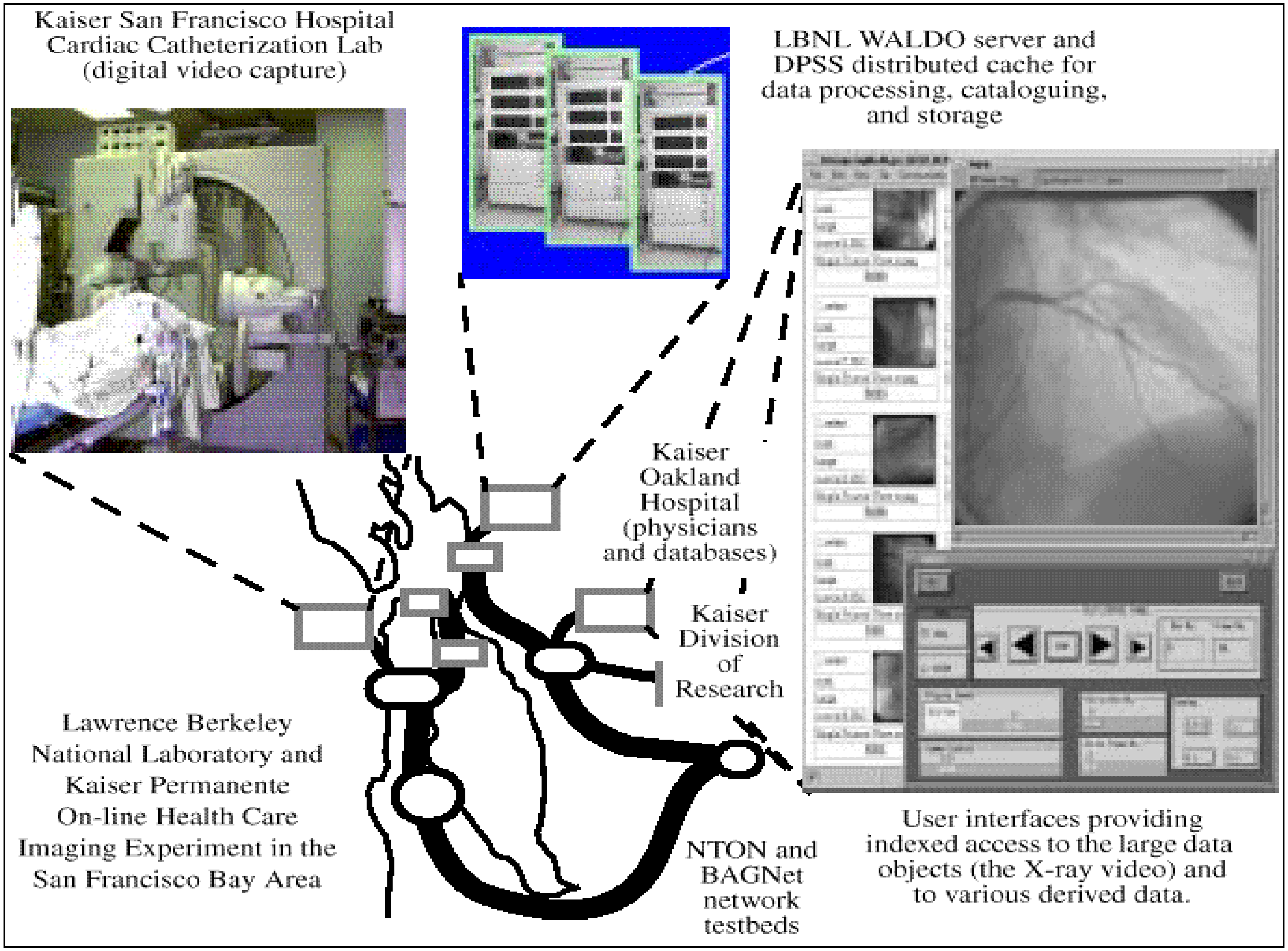
Typical DPSS implementation

- **4 UNIX workstations (e.g. Sun Ultra I0s, Pentium 450)**
  - **4 - 6 Ultra-SCSI disks on 2 SCSI host adapters**
  - **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 MBytes/sec) using these relatively low-cost, “off the shelf” components by exploiting the parallelism of:**
  - **4 hosts**
  - **16 disks**
  - **8 SCSI host adapters**
  - **4 network interfaces**

Medical Imaging Application

- **Cardio-angiography data was collected directly from a Philips 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 system at LBNL**
  - **Data now available to physicians at other hospitals**
- **This automated process goes on 8–10 hours a day**

Medical Imaging Application



Medical Data Handling System

- **LBNL/Kaiser Permanente collaboration focused on connecting remote, on-line instrument systems to "real-time" digital libraries, and provided:**
  - —**automatic generation of metadata**
  - —**automatic cataloguing of the data and the metadata as the data is received (or as close to real time as possible)**
  - —**transparent management of tertiary storage systems where the original data is archived**
  - —**facilitation of cooperative research by providing specified users at local and remote sites immediate as well as long-term access to the data**
  - —**mechanisms to incorporate the data into other databases or documents**

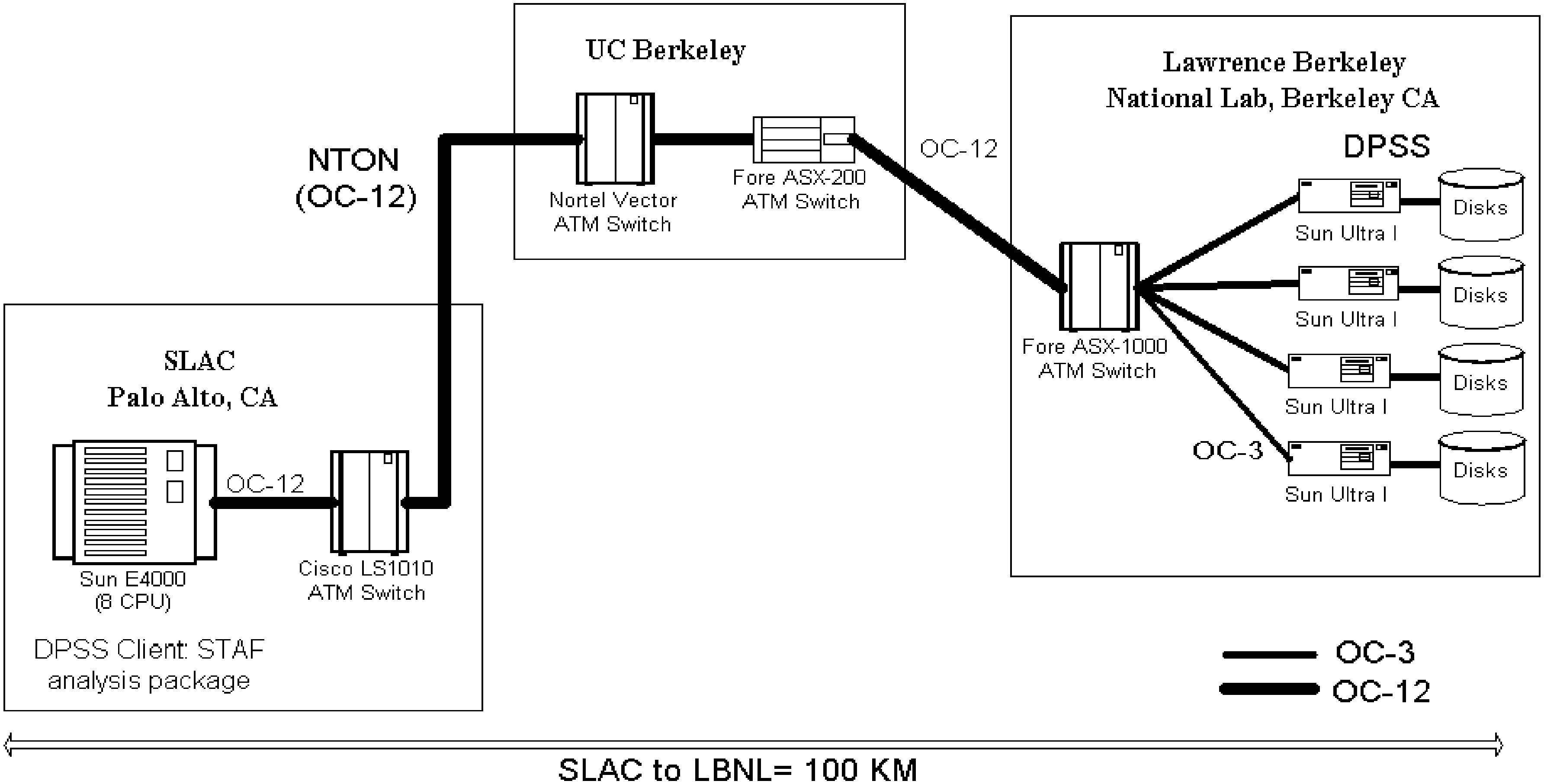
High Energy and Nuclear Physics Data Example

- 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

HENP Application Experiment

- **A set of experiments were conducted over the National Transparent Optical Network (NTON) 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 (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.**

HENP Application Experiment



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

HENP Application 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

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/Clipper/
- http://www-didc.lbl.gov/Imglib/