

# Performance Evaluation of Parallel Input/Output Systems



**Thomas Ludwig**

Ruprecht-Karls-Universität Heidelberg

Computer Science Department

Parallel and Distributed Systems

**[t.ludwig@computer.org](mailto:t.ludwig@computer.org)**



# Storage Facts

- Supercomputers in the TOP500-list
  - Main memory: Dozens of TeraByte
  - Disk space: PetaByte range
- Berkeley Report „How Much Information“
  - Increase of **5** Exabytes (about  $5 \times 10^{18}$  bytes) of information stored in 2002
  - 92% of it on magnetic media, mainly hard disks



# Part I

---

## Introduction to Parallel Input/Output



# Single Disk Performance

## Storage performance

- A single disk makes about 50MByte/s
- To store 1 GByte takes 20 seconds
- To store 1 TByte takes 20.000 seconds (~5 hours)  
Or 20 seconds with 1000 disks

Storage performance is a crucial factor



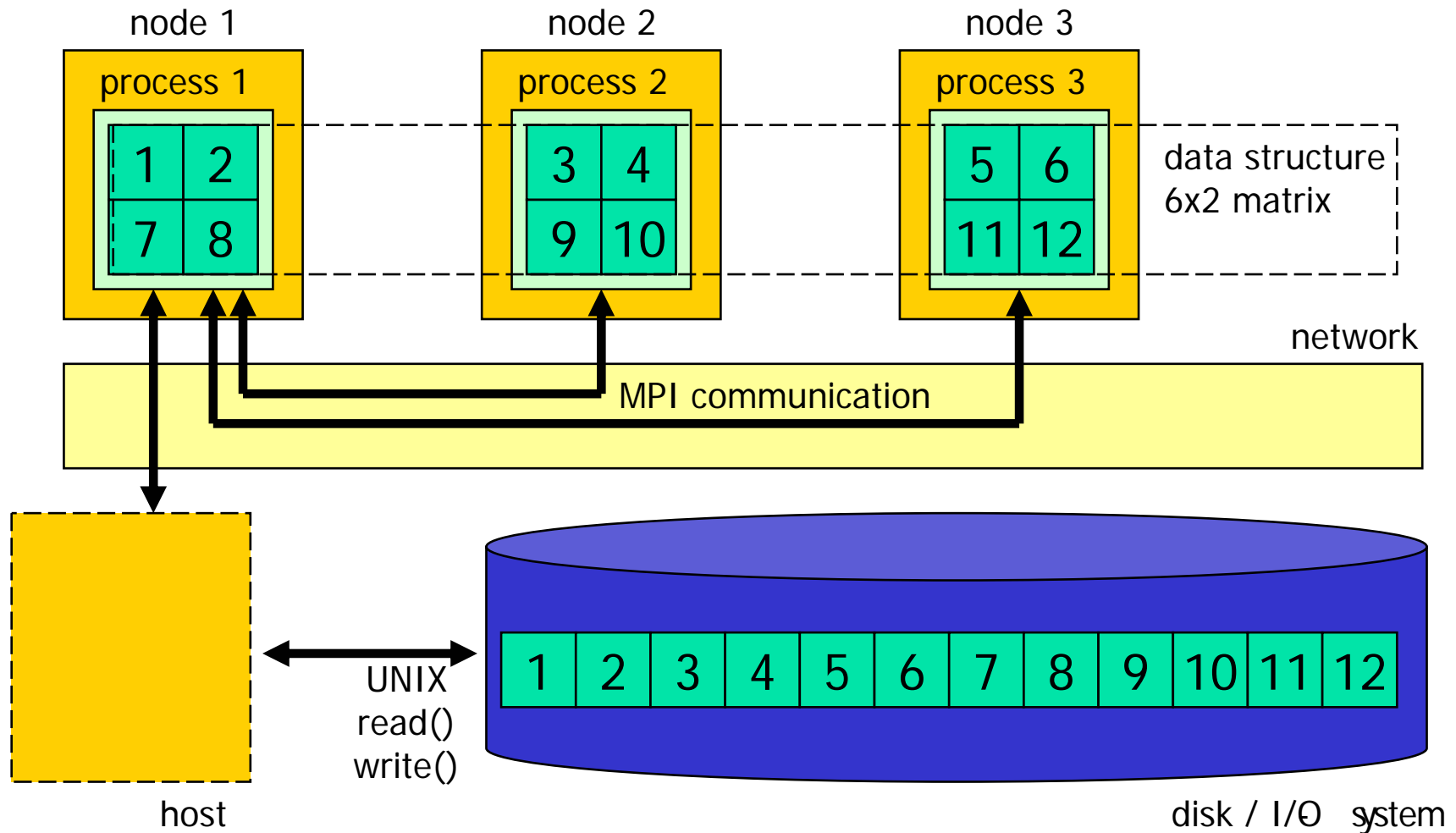
# Abstraction Layers

---

- Program level
  - How do I access my data?
  - Single process / multiple processes
- System level
  - Where is my data?
  - Single disk / multiple disks
  - Powerful modern I/O-systems are available
    - RAID, SAN, NAS
- Concepts at program and at system level are independent of each other!

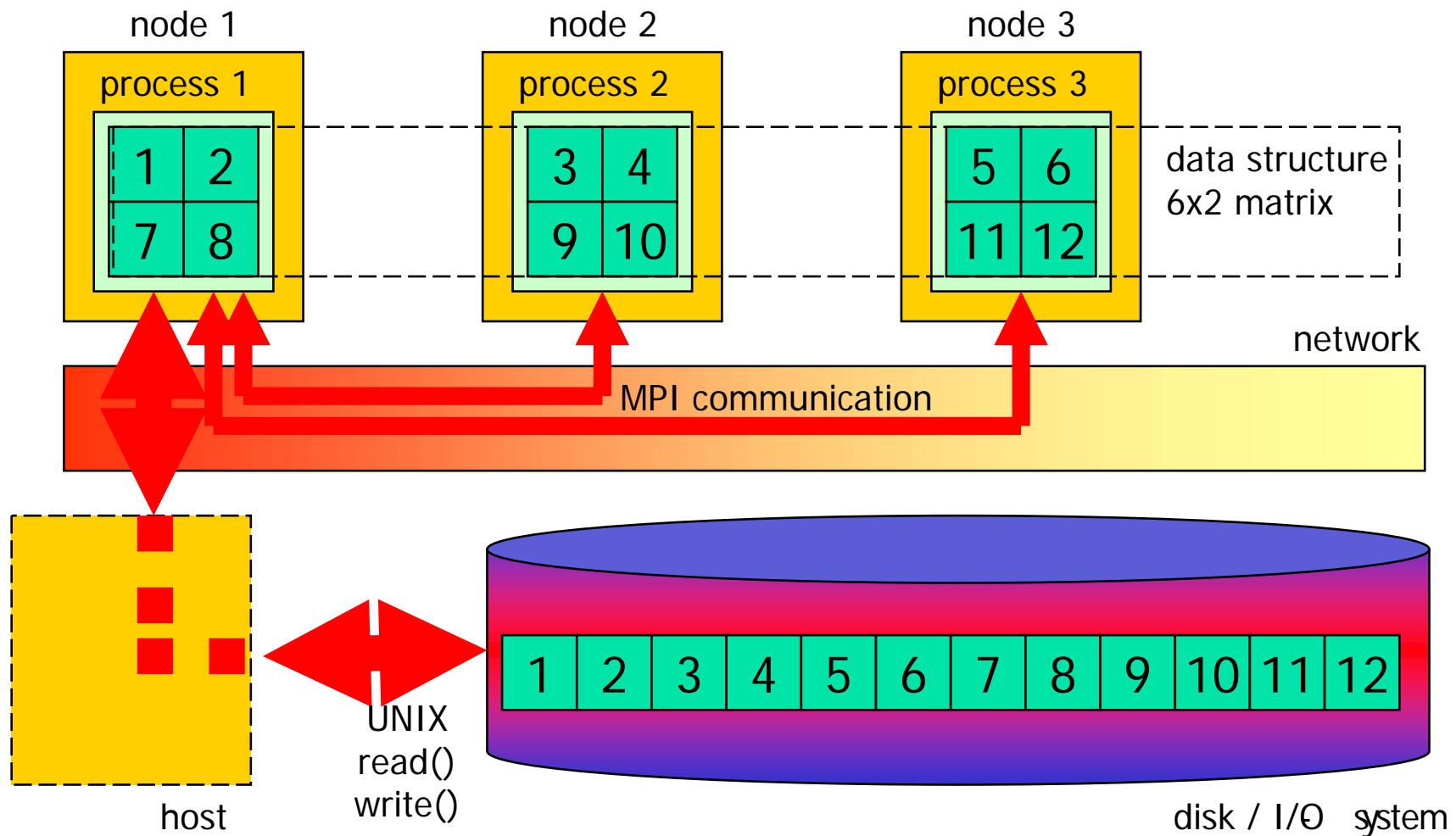
# Traditional Approach

## Parallel program with MPI communication



# Traditional Approach: Problems

## Parallel program with MPI communication





# Modern I/O-Concepts

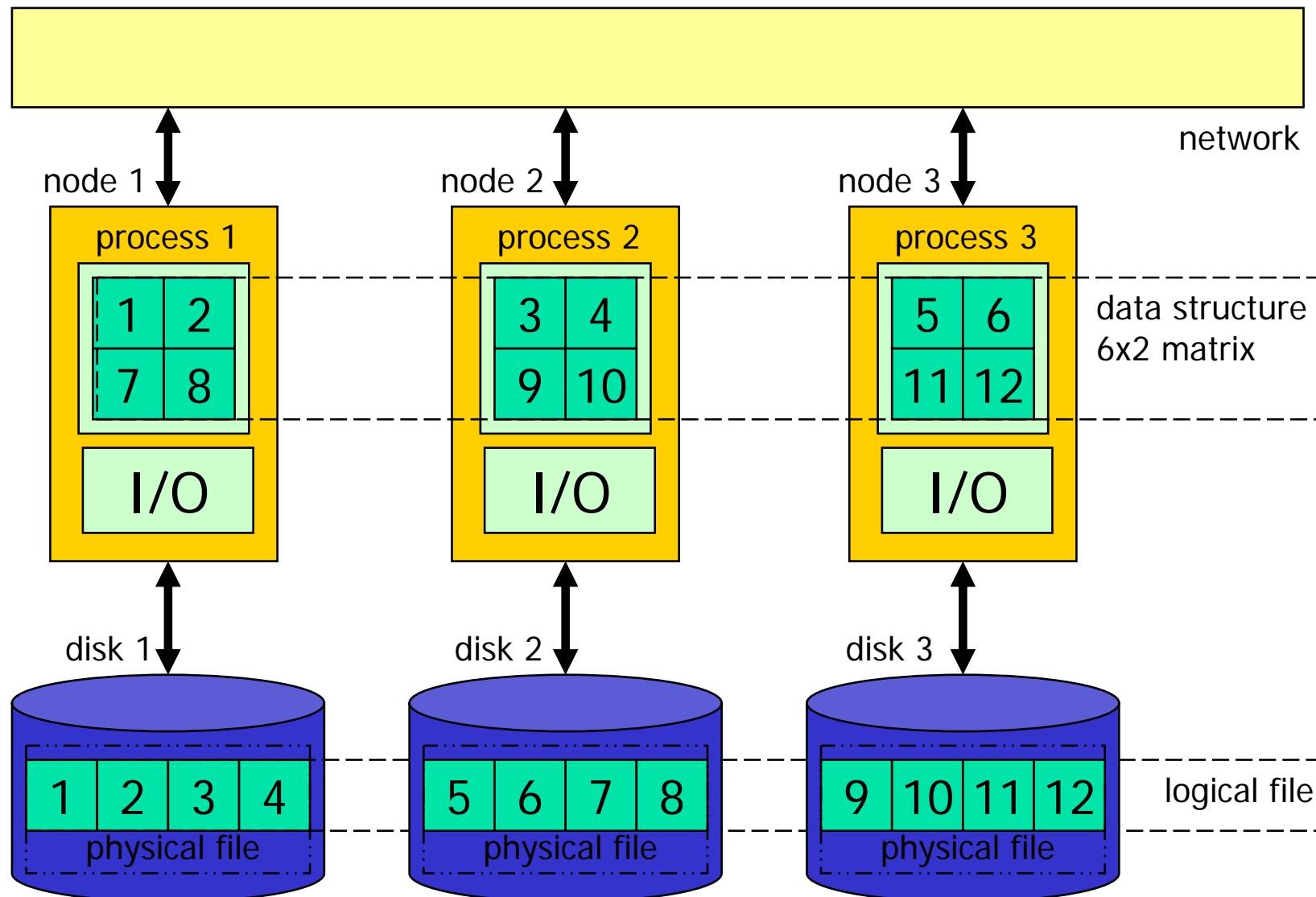
---

## Parallelization of I/O

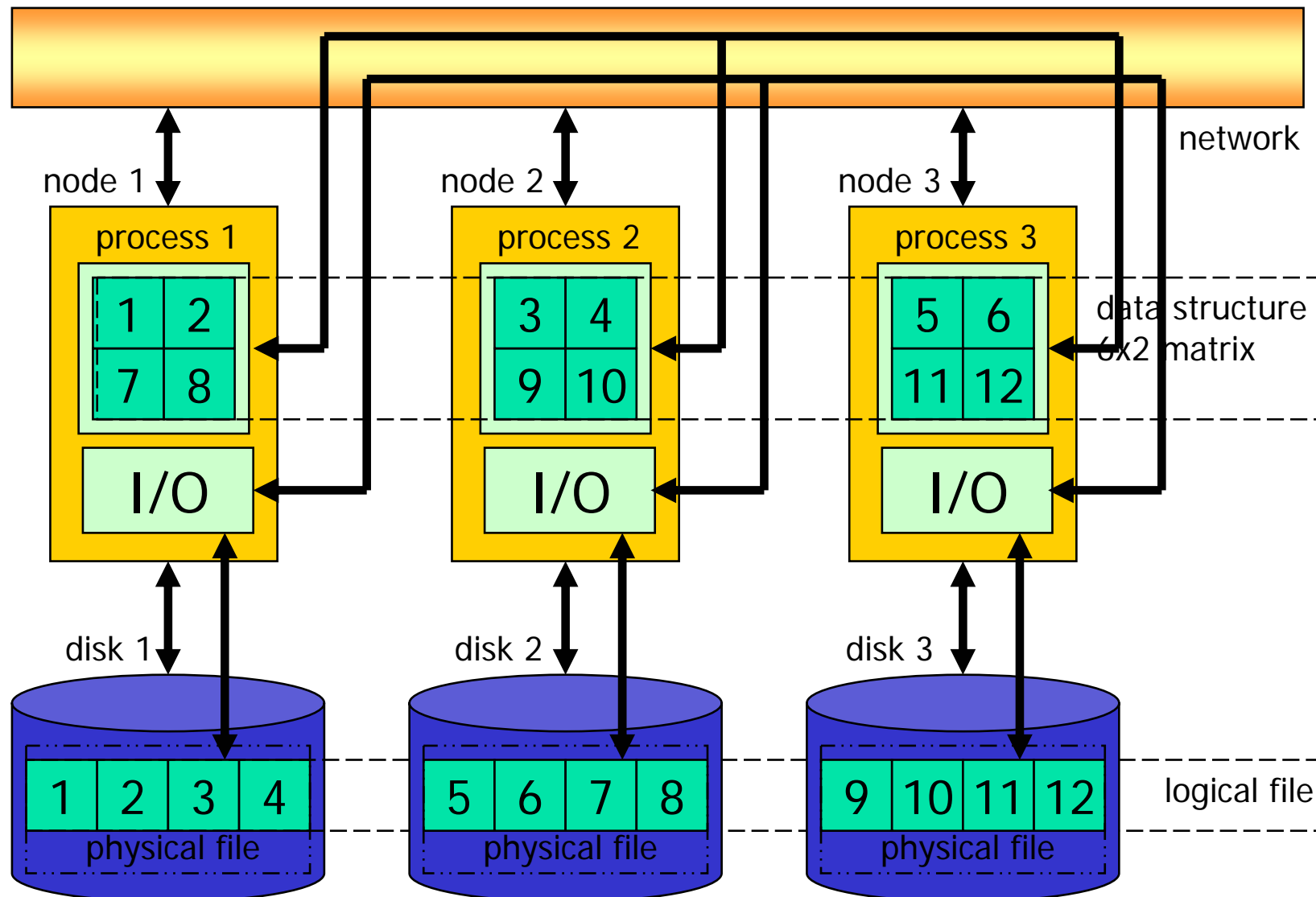
- At program level (Parallel file I/O)
  - Each process can directly make I/O calls
  - I/O calls from different processes may address identical files (even identical bytes)
- At system level (Parallel file system)
  - We use more than one disk
  - A single file may be spread over many disks
  - The size of a file may exceed the size of a disk

Both together is called "Parallel I/O"

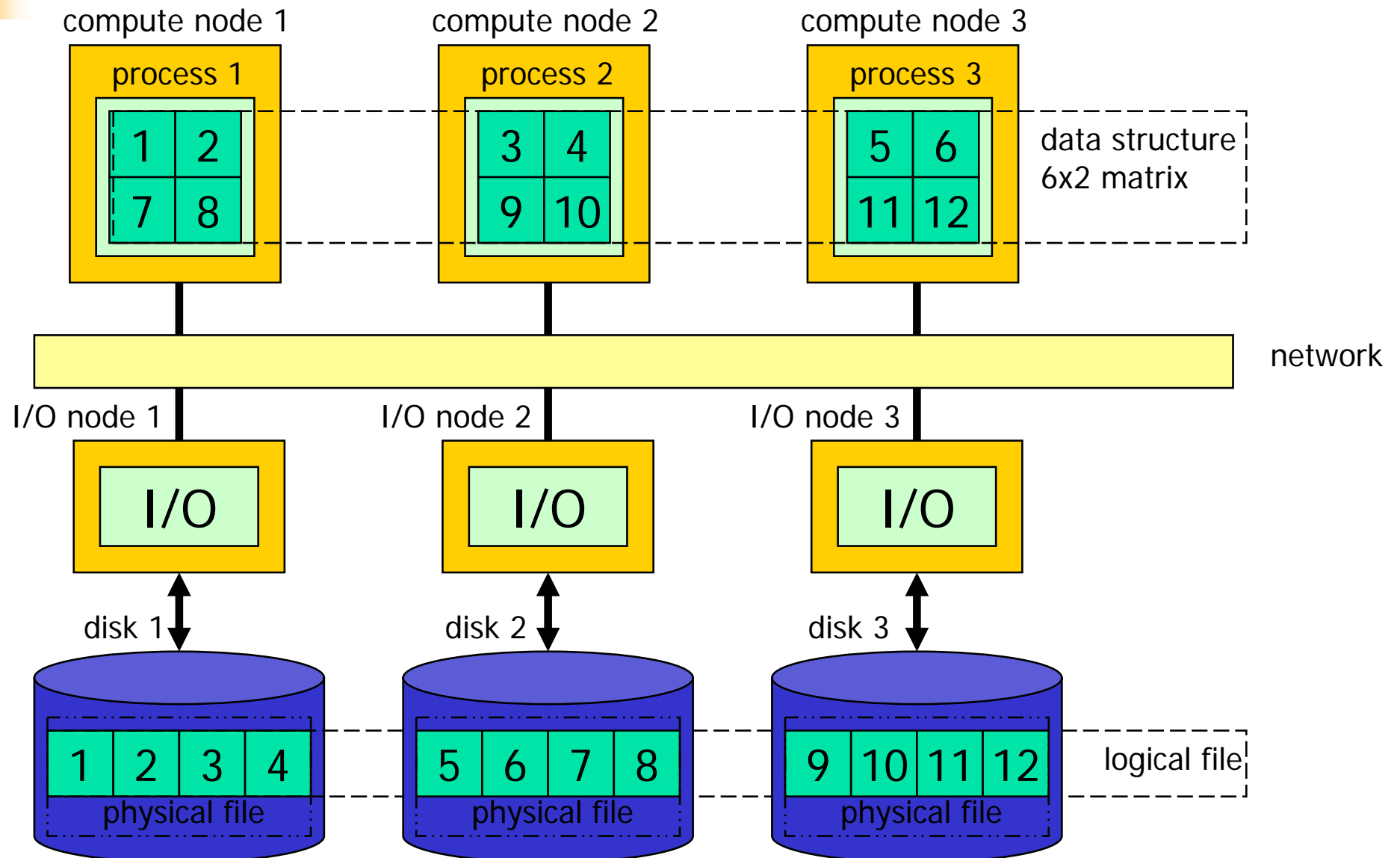
# The Concept of Parallel I/O



# The Concept of Parallel I/O...



# The Concept of Parallel I/O...





# The Concept of Parallel I/O...

---

Distribution function  $d$  :

$$d(\text{offset\_in\_logical\_file}) = (\text{server}, \text{offset\_in\_physical\_file})$$

e.g. default with PVFS2:

RAID-0 striping with 64 KByte stripe size

Distribution function influences overall performance



# Parallel I/O Libraries

---

- Parallel programming with message passing on large supercomputers and clusters
  - Use send()- and receive()-routines
  - MPI is the current standard for this
- Parallel I/O: defined by MPI-IO
  - Defines read()-operation similar to receive()
  - Defines write()-operation similar to send()
  - Keeps most other semantic details of MPI message passing



# Parallel File Systems

---

- Parallel file system
  - Distributes a single file over several disks
  - Allows for concurrent access to a single file from processes of a parallel program
- Not too many systems available
- A selection
  - PVFS (ANL): popular open source system
  - Lustre (CFS): powerful open source system
  - GPFS (IBM): older proprietary approach



# Open Research Questions

---

- Usability
  - Abstraction levels with user libraries
- Increase of performance
  - Metadata server is bottleneck
  - Mapping of logical file to physical files is important (distribution function)
- Improved availability
  - How to keep a file on 1000 disks?
  - Fault tolerance is mandatory



# Own Research

---

- Deployment and optimization of parallel I/O in cluster environments
  - Deployment in production environments
  - Performance measurement
  - Resource management



# Part II

---

## Deployment in Production Environments



# Image Processing in Bioinformatics

---

- Project with the German Cancer Research Center (DKFZ) in Heidelberg
- High volume data comes from microscopy at the EMBL in Heidelberg
- Archived at the DKFZ
- Computed at the DKFZ
  
- Project just started / first investigations



# Structure of Data Set

---

- One picture has approximately 1 MByte
- One experiment has
  - Approximately 400 „Spots“  
= different positions with cells with knocked-out genes
  - Per spot approximately 100 pictures  
Every n minutes
- Results in about 40.000 files/experiment
- We have about 500 experiments
- Alltogether 20MFiles and 20TBytes



# Image Processing

---

- Image processing algorithm takes every file and computes feature vector
  - CPU intensive
- Video generation tool takes all files of one spot (about 100) and makes an MPEG video from it.
  - I/O intensive



# Storage Concept

---

- Use clustering as with CPU
  - i.e. as we usually have clusters with disks attached to the nodes:  
use them as storage facility
- Available cluster nodes
  - IBM: 4 nodes, dual processor (Opteron), 1 free disk
  - HP: 4 nodes, dual processor/dual core (Opteron), 3 free disks with RAID-0 controller







# Bechmark Programs

---





- Already done
  - **b\_eff\_io** synthetic benchmark
  - Stresstest for metadata operations
  
- To be done
  - Image processing
  - Video generation

# Configurations for Parallel I/O





1

Knoten 1	Knoten 2	Knoten 3	Knoten 4
E/A	E/A	E/A	E/A
Metadaten	Metadaten	Metadaten	Metadaten
			





2

Knoten 1	Knoten 2	Knoten 3	Knoten 4
E/A	E/A	E/A	E/A
Metadaten	Metadaten	Metadaten	Metadaten
			

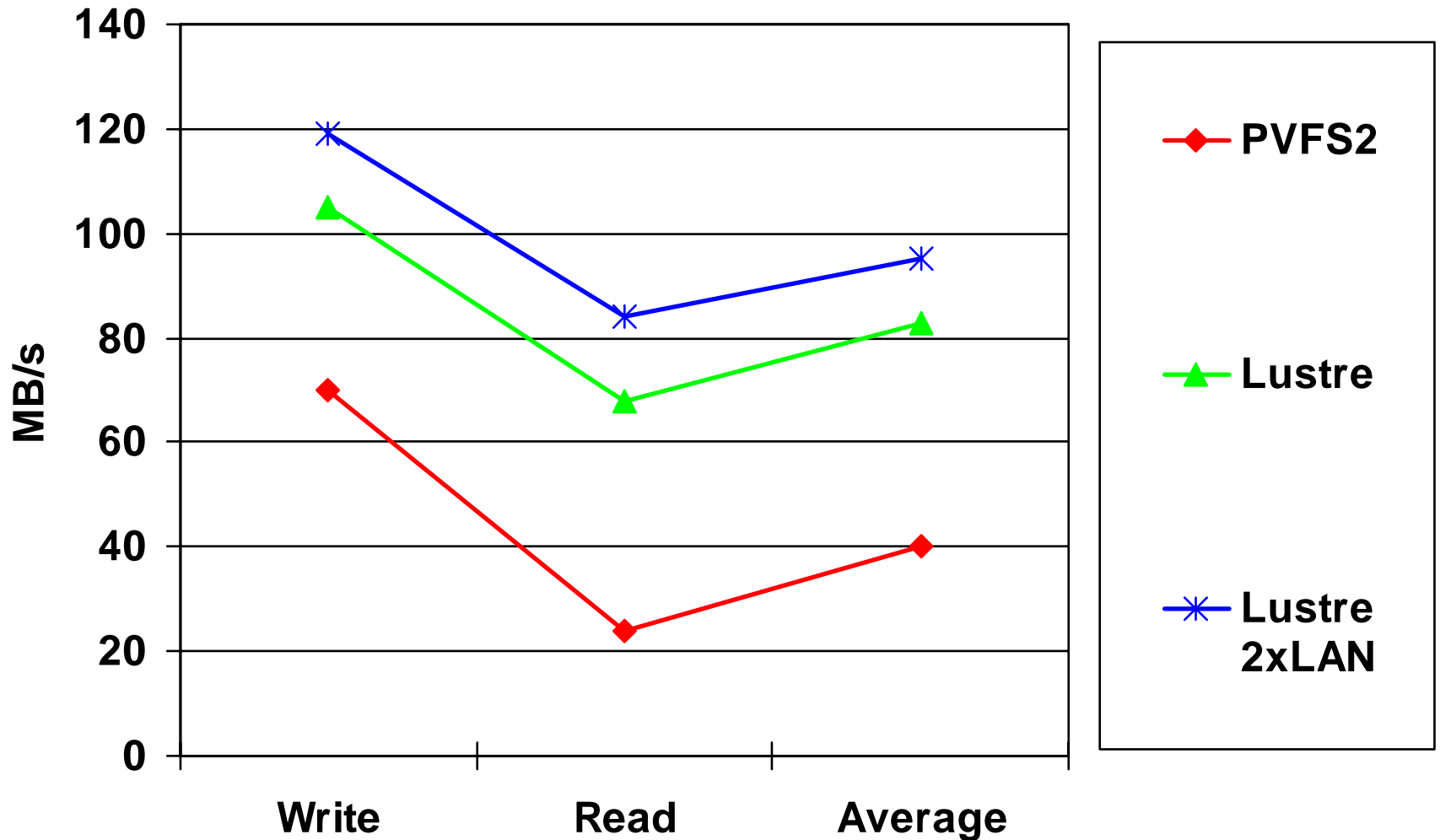
3

Knoten 1	Knoten 2	Knoten 3	Knoten 4
	E/A	E/A	E/A
Metadaten			
			

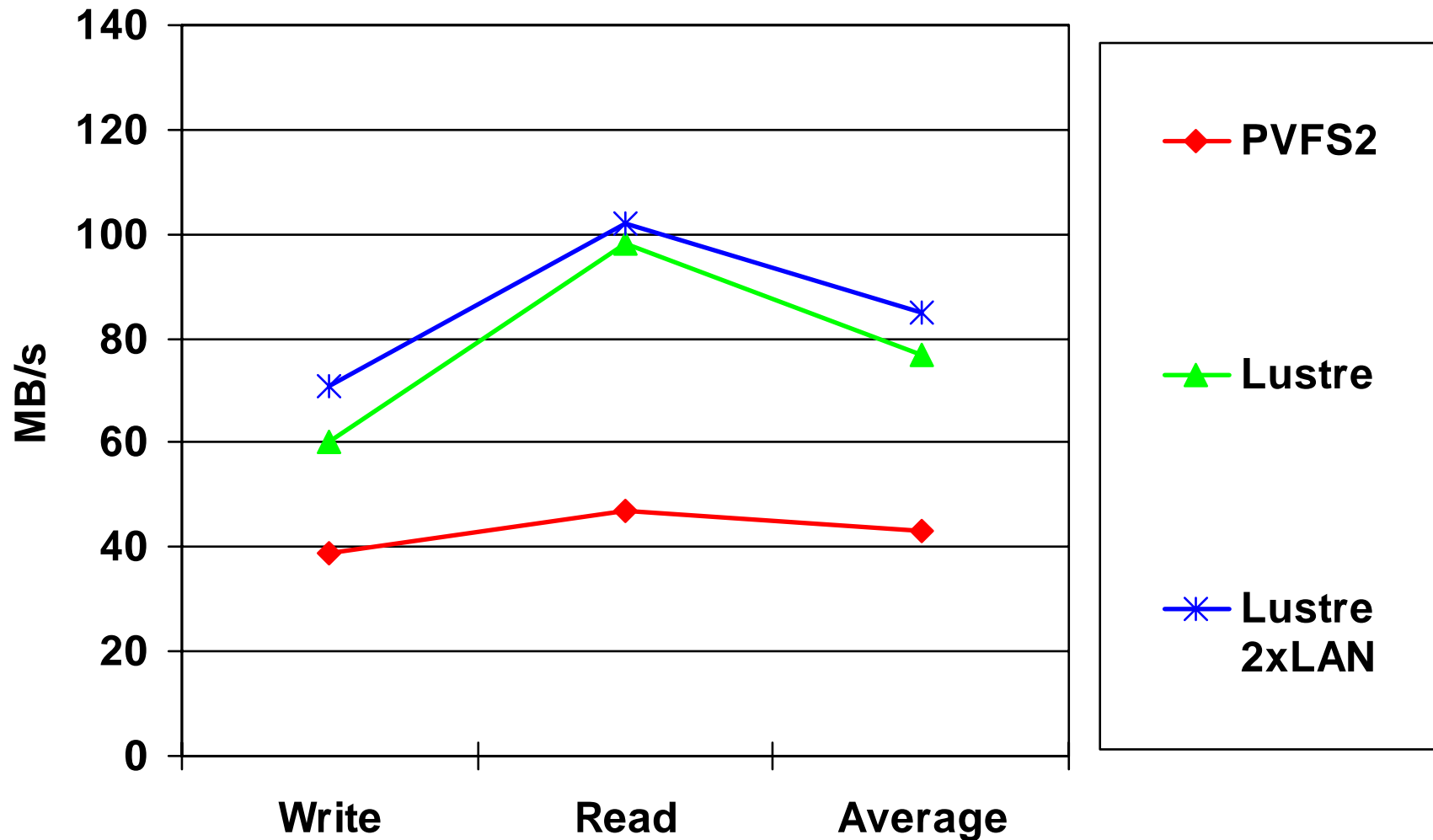
4

Knoten 1	Knoten 2	Knoten 3	Knoten 4
	E/A	E/A	E/A
Metadaten			
			

# Configuration 3: one client



# Configuration 4: four clients

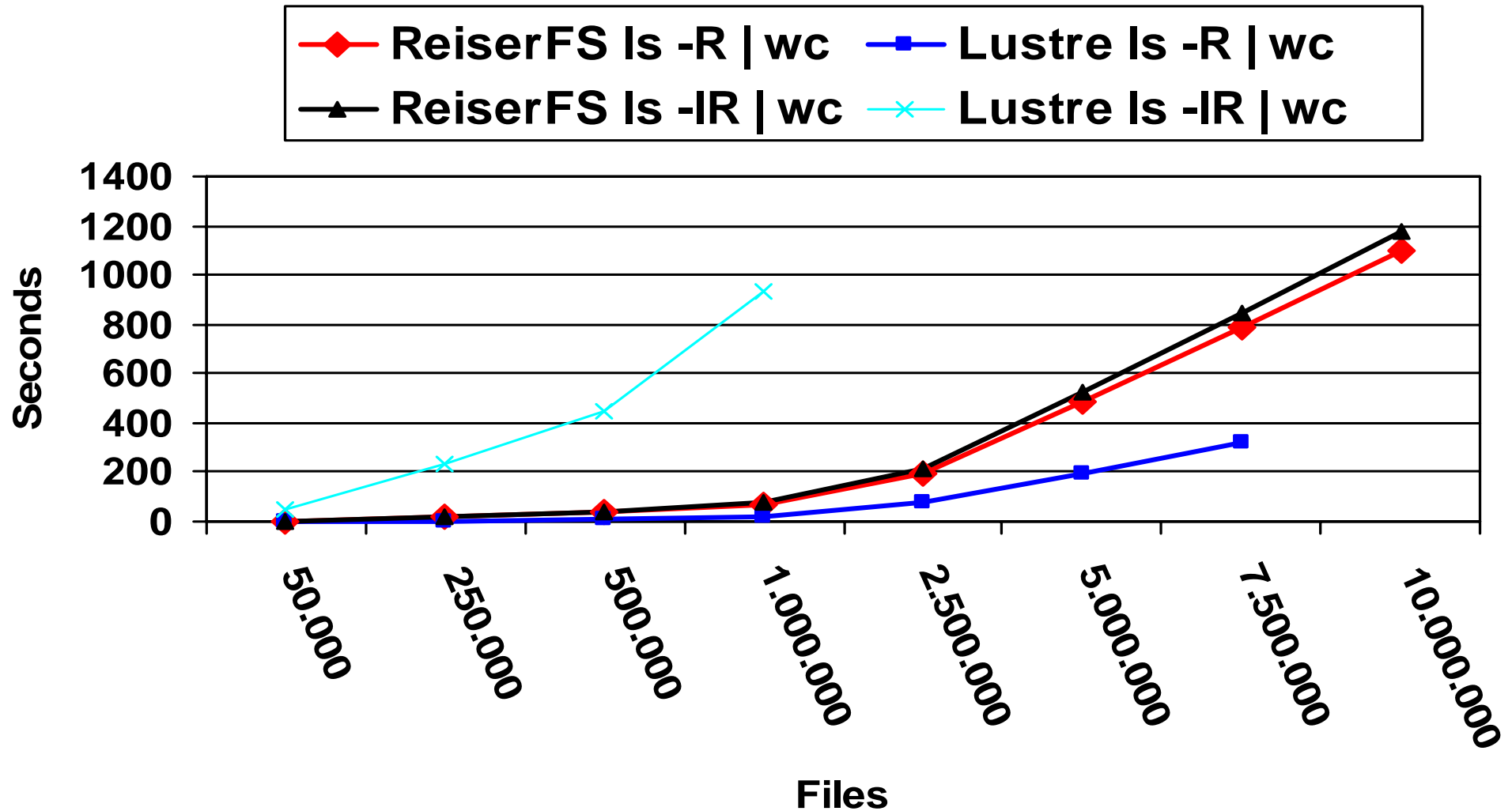


# Stresstest for Metadata Operations

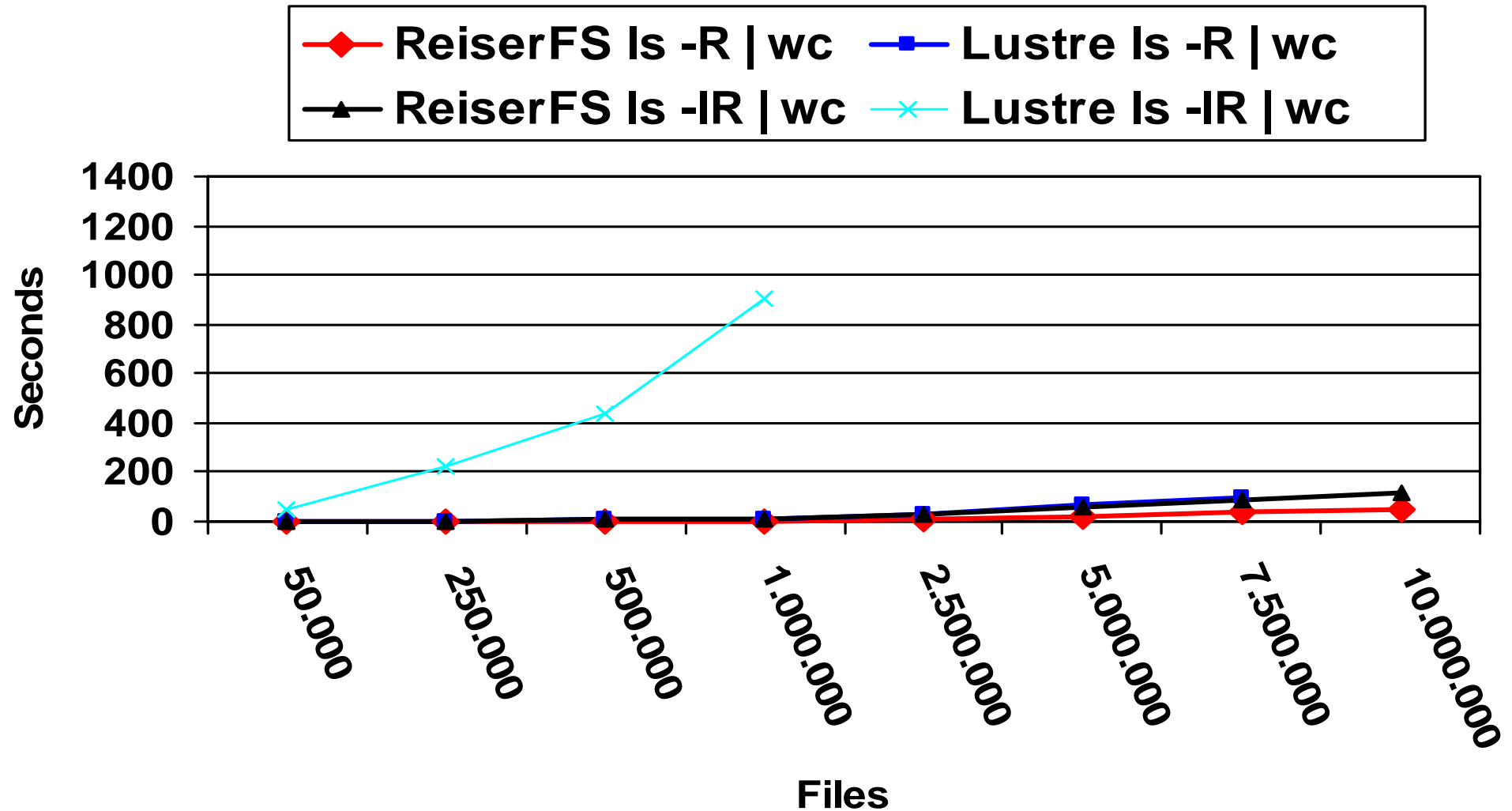


- What's the problem here?
  - Metadata servers cannot be parallelized, i.e. access to them is a potential bottleneck
  - You may have several metadata servers, but usually they handle different files
- Metadata of files in a parallel file system is not stored in regular i-nodes

# Metadata: First Access



# Metadata: Repeated Access





# First Conclusion

---

- Performance evaluation problem
  - No appropriate tool to investigate in detail the performance issues
- Metadata problem
  - Metadata access performance is a crucial issue for parallel file systems with respect to scalability of performance



# Part III

---

## Current Work: Performance Measurement



# The Semantic Gap

---

- Program I/O is done in parallel MPI applications
- System I/O is performed by the servers of the I/O subsystem
- There is no tool that shows the relation between activities on both abstraction levels



# Consequences

---

- No real insight into low level I/O activity in dependency of high level I/O activity
- No easy tuning of the application
- No easy tuning of the servers
- No optimal adaptation to the available resources



# Relevant Resources

---

- Disks: limit what a node can read and write
- Server network connection: limits what a single server can send to/receive from clients
- Client network connection: limits what a client can send to/receive from servers
- CPU: limits what server and client code can use as compute cycles



# Project Goal

---

Design and implement a tool environment that visualizes three aspects:

- Client I/O activity
- Server I/O activity
- The relation of the two of them

## Environment

- MPICH2 for parallel programming
- PVFS2 as a parallel file system

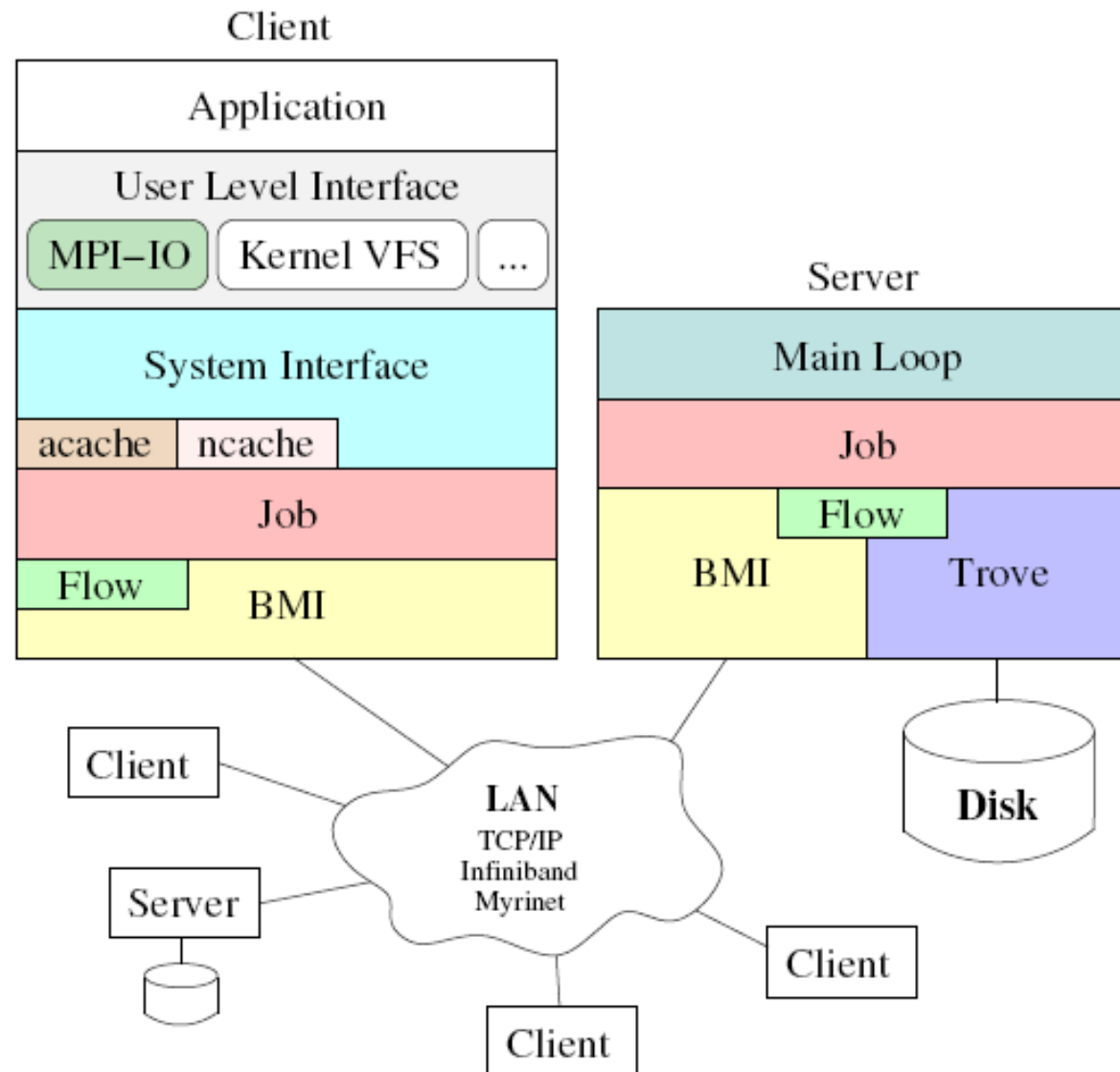


# Starting Point

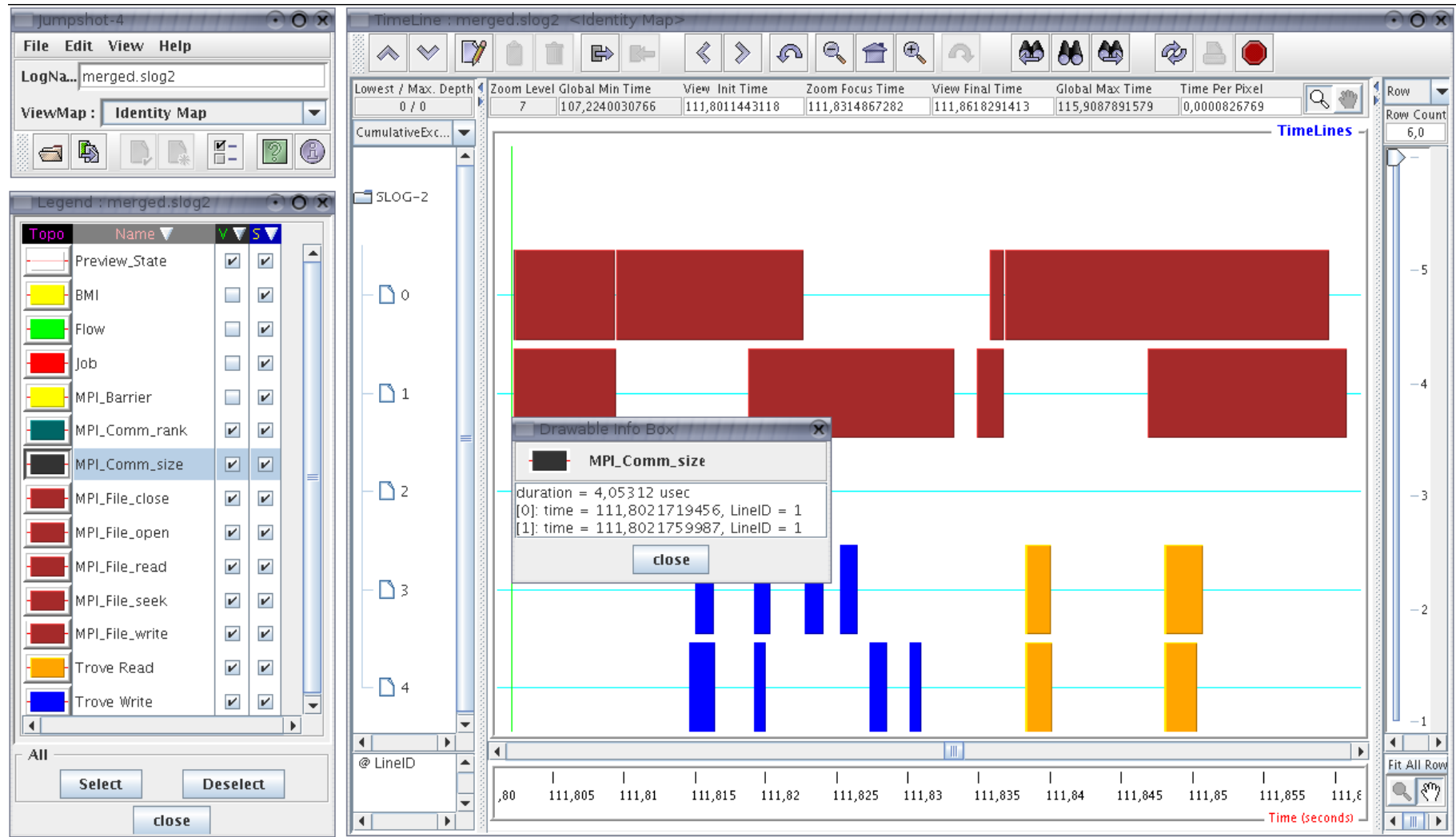
---

- Trace generation in MPICH2 via MPE
- Trace visualization via Jumpshot
- First rudimentary instrumentation in the PVFS2 servers to get relevant performance values

# The PVFS2 Environment



# Jumpshot





# Idea

---

- Generate a trace for the servers
- Merge the server trace with the client trace
- Add arrows from MPI-IO calls to corresponding server activities
- Visualize everything together in Jumpshot





# Trace Generation Problems

---

- Servers run permanently
  - Start and stop tracing
- Client and server traces are not synchronized
  - Add special events to adjust them
- Server has no tracing facility
  - Start servers as MPI program and produce a trace as with an MPI application
- Client and server traces are separate
  - Merge them



# Add Arrows to the Trace

---

- Arrow semantics
  - We want to show which MPI-IO call results in which PVFS server activities
- What do we need?
  - We need a unique ID for every MPI-IO call
- Concept
  - Trace activities at the client
  - Transfer IDs to the servers
  - Trace activities at the servers
  - Connect activities with identical IDs at both levels

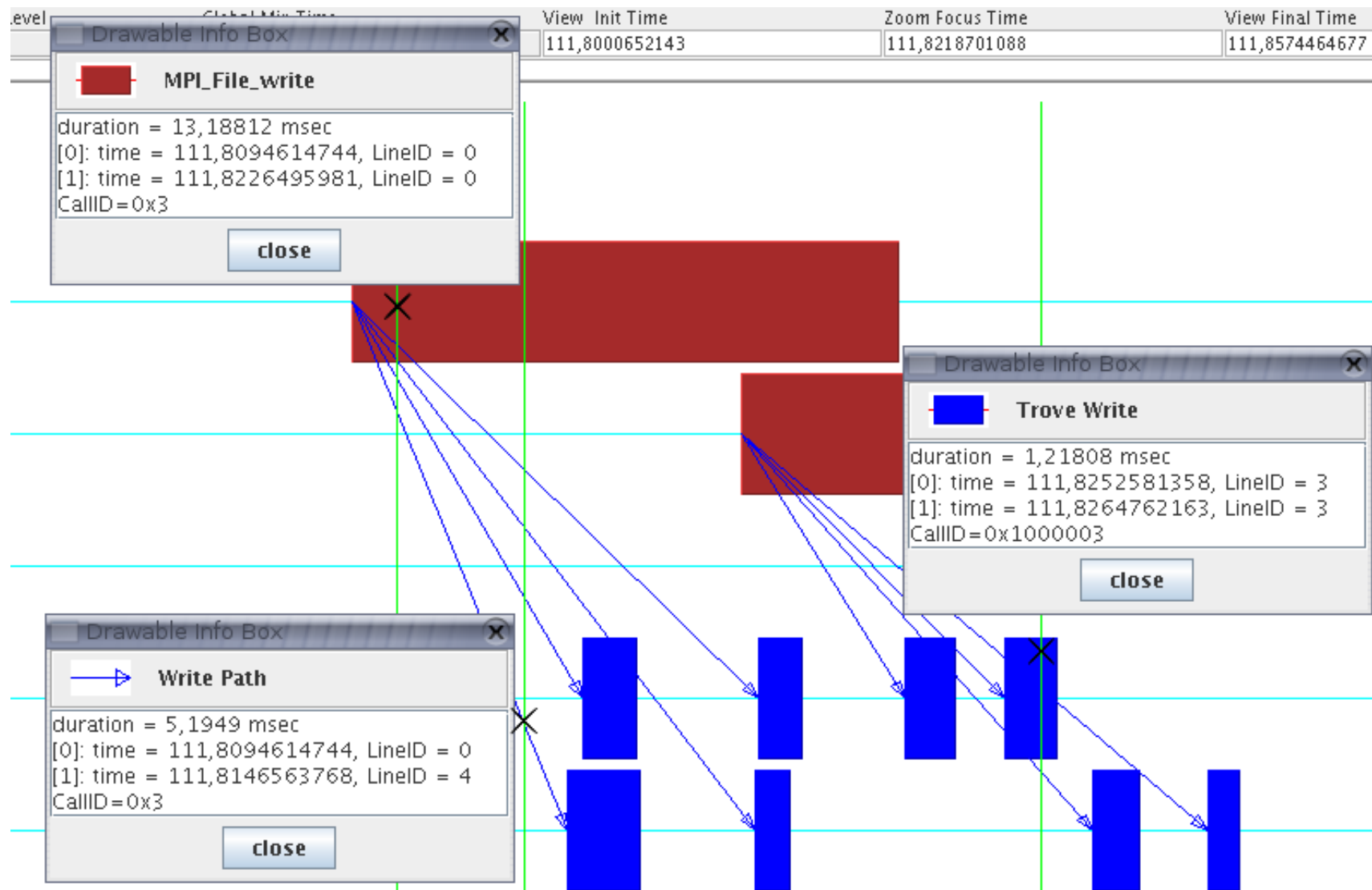


# Visualization

---

- In theory: no problem at all
  - Jumpshot just takes the elements of an slog2 file and visualizes them – has no semantics integrated
- In practice
  - Too much information for the individual time lines (overlapping events because of asynchronous server behavior)
  - Solution: Distribute events to several time lines such that there is no overlap

# An Example: MPI\_File\_write triggers Trove write





# First Results

- We can see all PVFS2 activities from the server modules Job, BMI, Trove, Flow
- We can relate them to the MPI-IO calls that triggered the corresponding activity
- Sources and patches will be on-line in about 2 months



# Next Steps

---

- We will (re-)run various programs and produce traces
  - Synthetic test programs
  - Real programs
- A trace analysis will give more insight into sources of performance problems



# Part IV

---

## Future Work: Ressource Management



# Project Goal: I/O Cluster

---

- Use cluster as a configurable I/O and compute resource
  - Determine number of nodes for computation and for I/O depending on the I/O activity of a program
  - Integrate it with the cluster resource manager (PBS and others)



# Features

---

- Provide medium term storage
- Provide QoS features
  - Single/multiple user partition
  - No fault tolerance / tolerance against disk failures
  - Storage capacity
  - I/O bandwidth
- Use cluster storage as file cache for high throughput computing