

# Analysis of the MPI-IO Optimization Levels with the PIOviz Jumpshot Enhancement



**Thomas Ludwig**

S. Krempel, M. Kuhn, J. Kunkel, C. Lohse

Ruprecht-Karls-Universität Heidelberg

Computer Science Department

Parallel and Distributed Systems

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



# Overview

---

- Software Model
  - MPI, MPI-IO, PVFS, ROMIO
- The PIOviz Project
  - Parallel I/O Visualization
- Analysis of the MPI-IO Levels
- Conclusions, Future Work



# Software Model

---

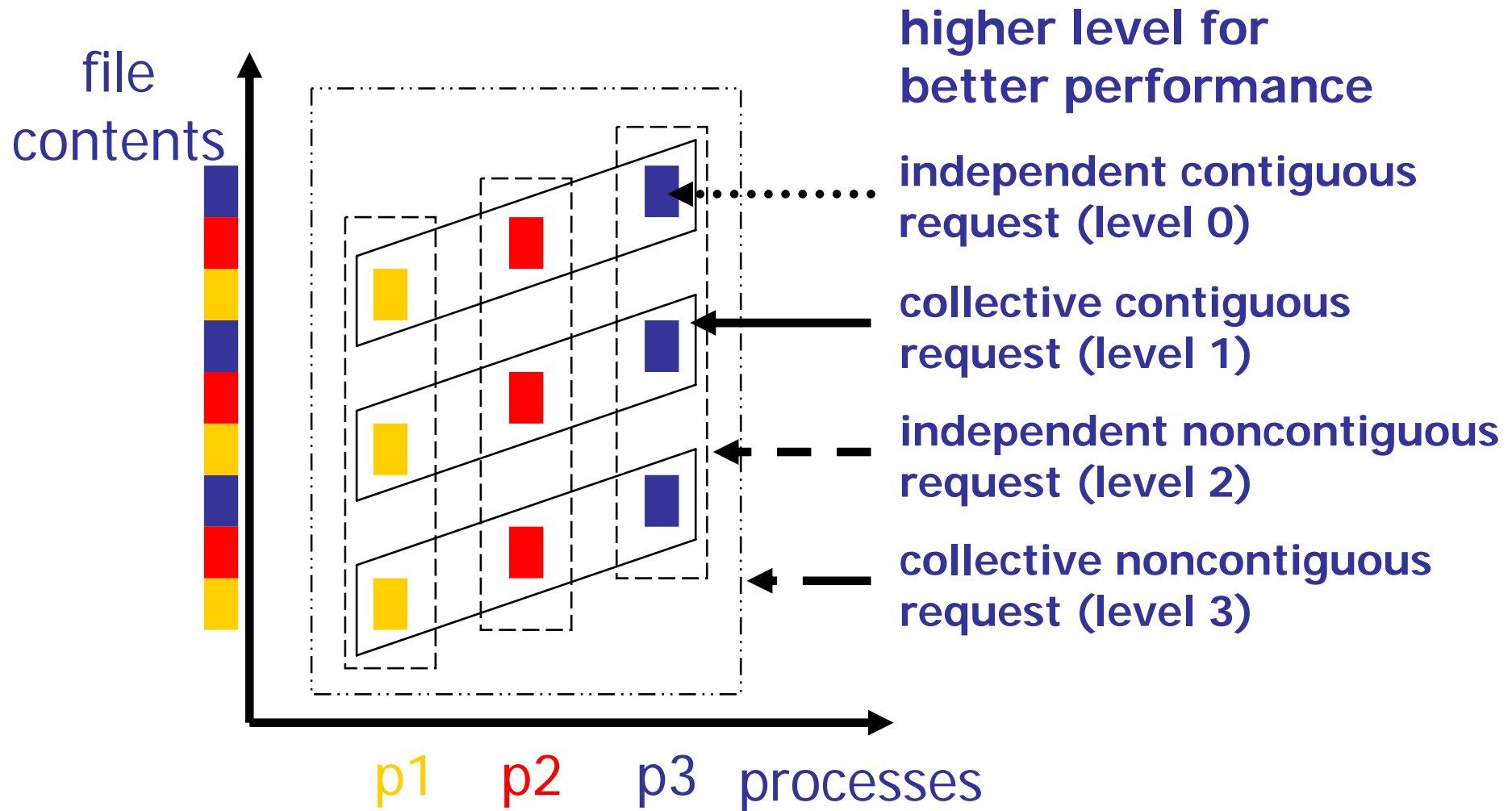
User-level optimizations in MPI-IO

- collective calls of several processes
- access to noncontiguous data regions

Realized by different calls with the API

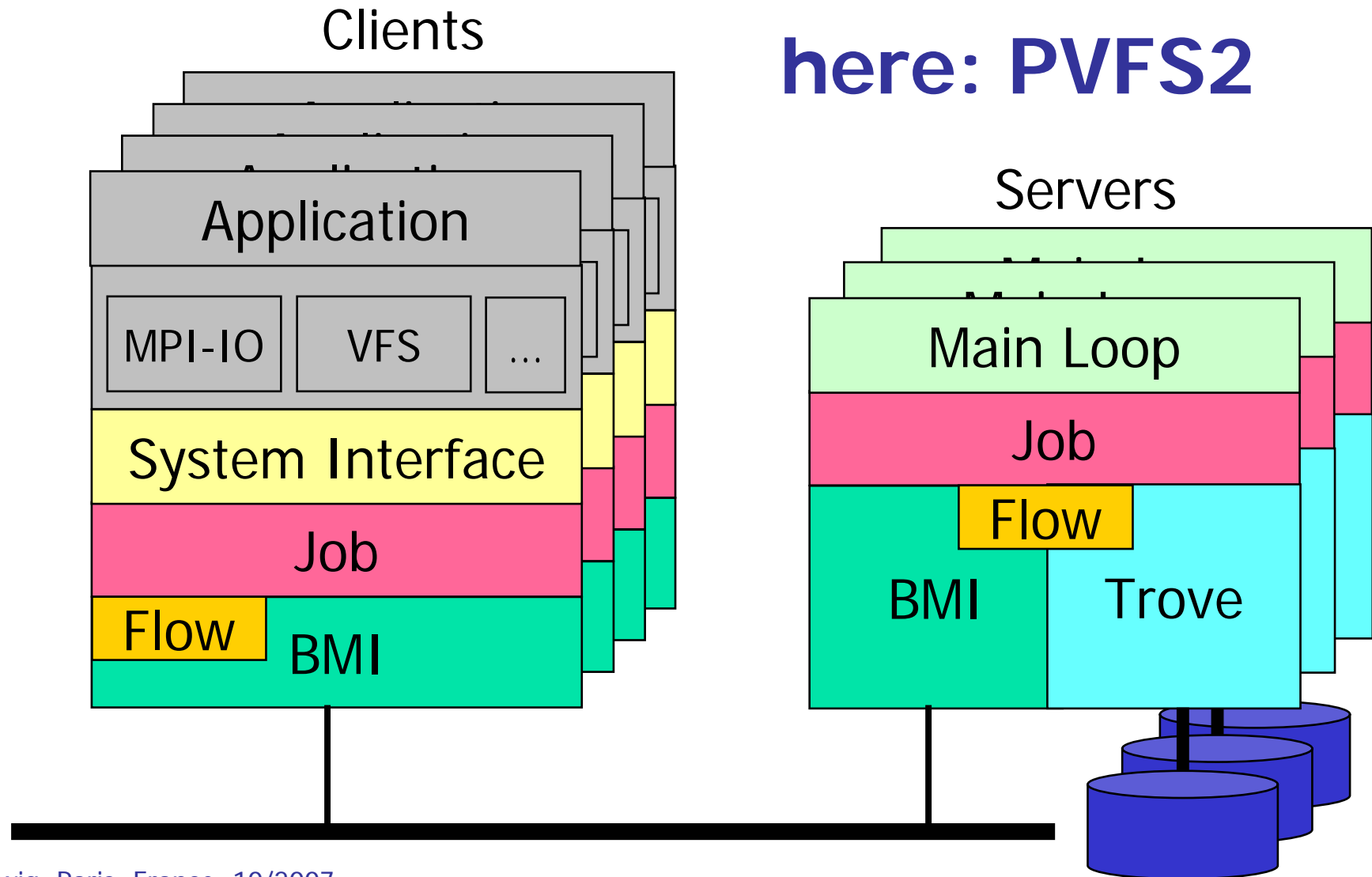
Allows for optimizations in the implementation of the MPI-IO functionality

# MPI-IO API Variations

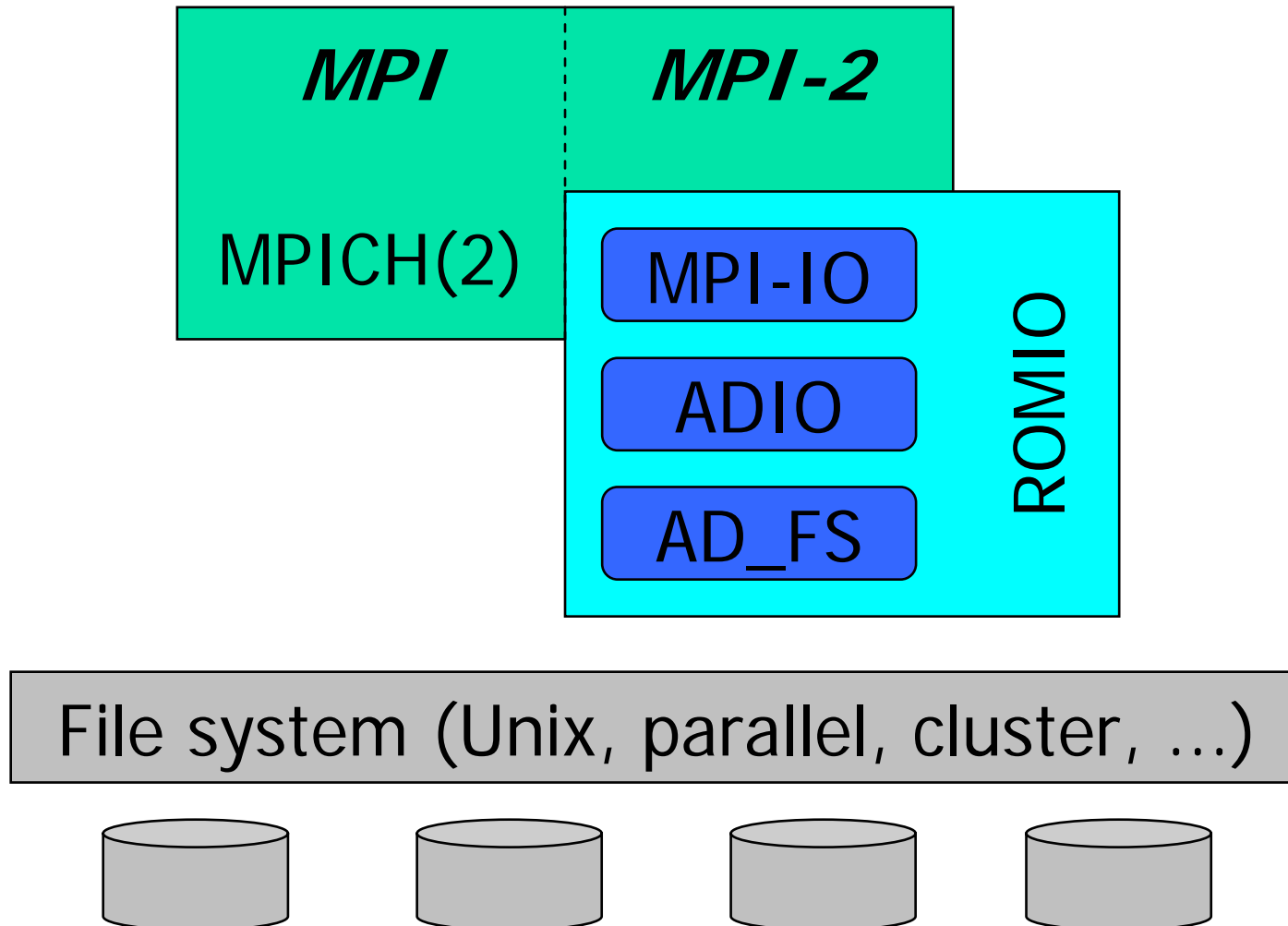


# The Parallel File System

## Server Process + Client Library



# The I/O-Client





# ROMIO-Optimizations

---

- Data Sieving

Level 2 (non-contiguous)

- fetch more data than needed
- throw away parts of it

- Two-Phase Protocol

Level 3 (collective and non-contiguous)

- each client fetches a big contiguous block
- sends data to owner process via internal messages



# The Semantic Gap with I/O

---

- Program I/O is done in parallel applications with MPI-IO
- System I/O is performed by the servers of the I/O subsystem
- There is no tool that shows the causal 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



# Project Goal

Design and implement a trace-based tool environment that visualizes three aspects:

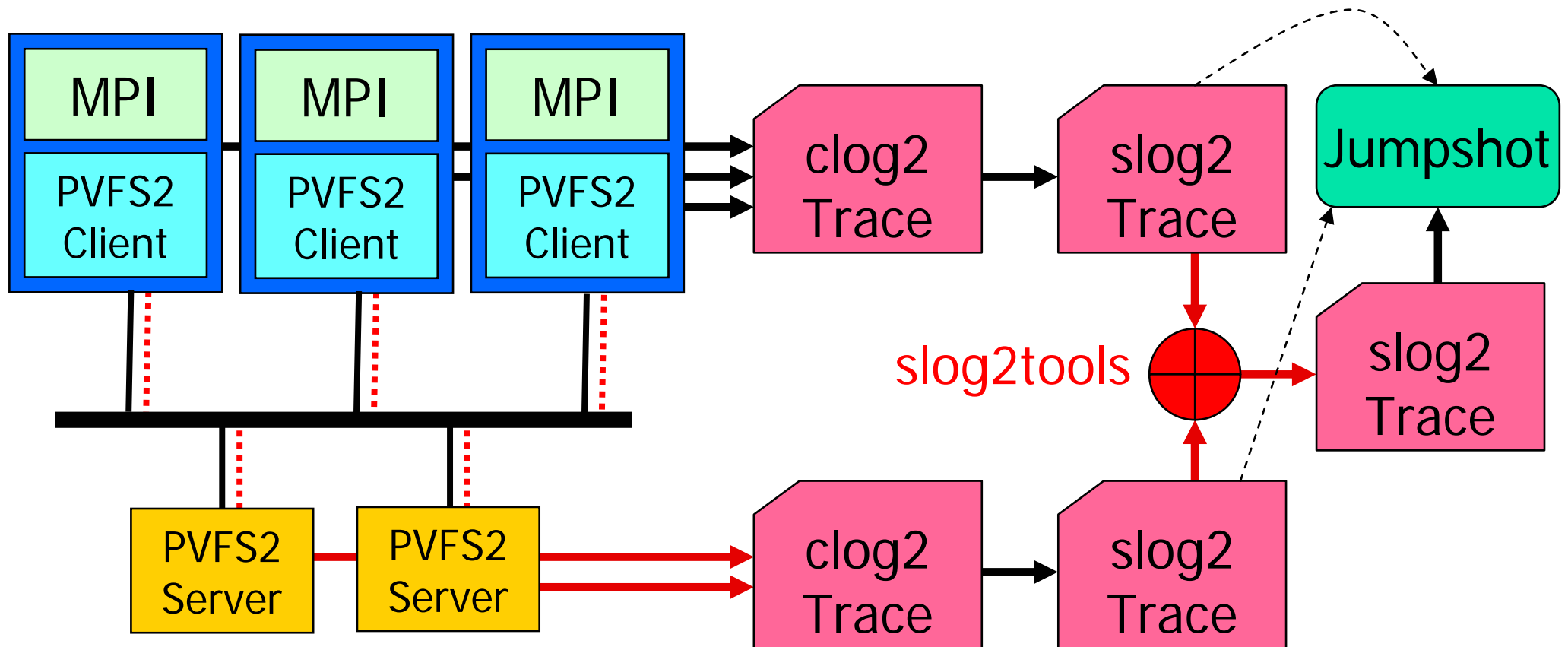
- Client I/O activity
- Server I/O activity
- **The relation between the two of them**

Environment

- MPICH2 for parallel programming
- PVFS2 as a parallel file system
- Jumpshot as visualization tool

# Trace Generation

Use MPICH/MPE clog2/slog2 traces





# Phases of Trace-Generation

---

Already included in MPICH

- Generate traces for clients

New with our project

- Generate traces for servers
- Merge client and server traces
- ...
- Add arrows



# Generate traces for clients

---

- Feature already in original MPICH
- Just link program with MPE logging mechanisms
- Produces single trace file for visualization with original Jumpshot

# 4 clients trace (1x MPI\_File\_open, 10x MPI\_File\_write)

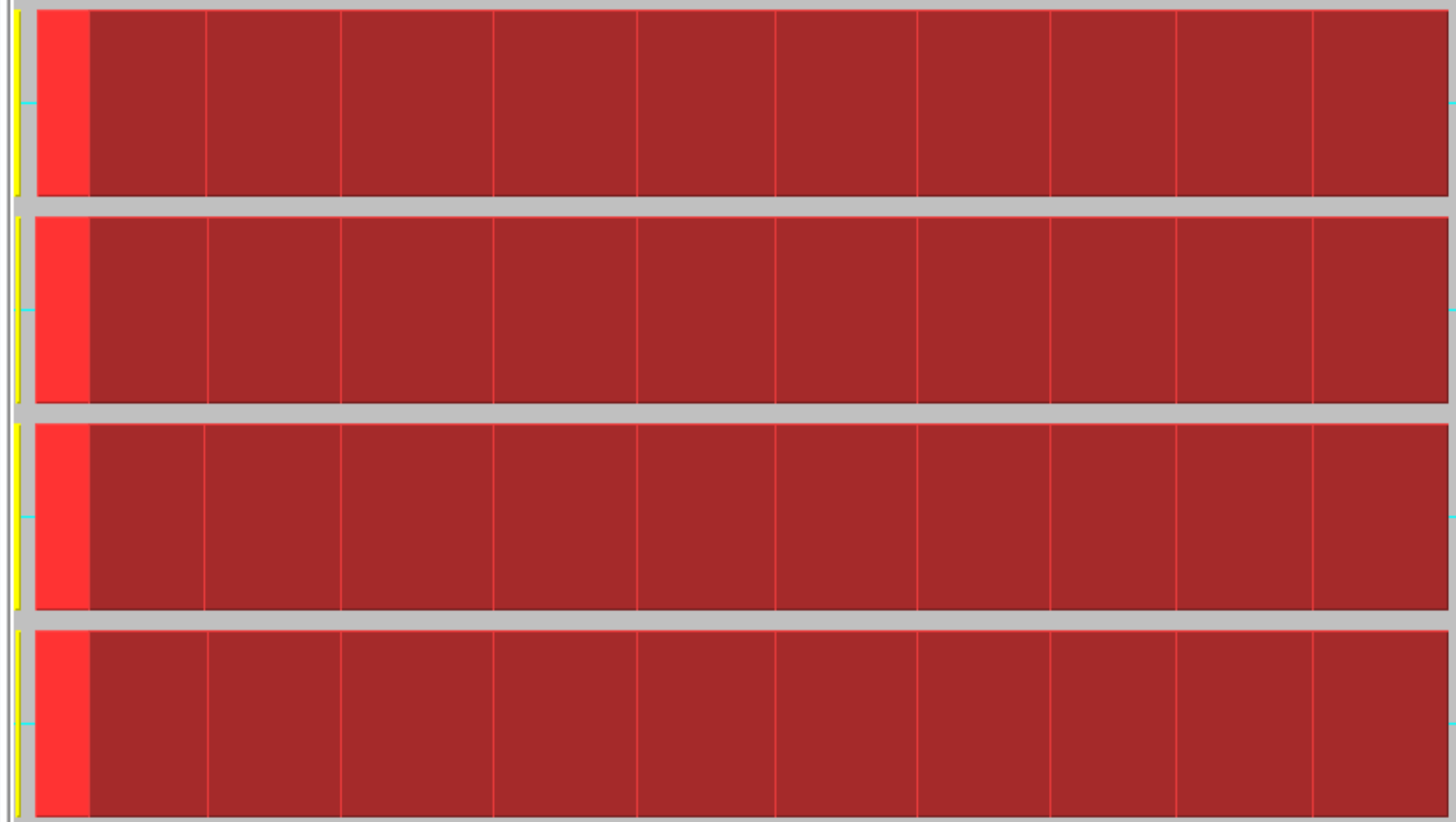
SLOG-2

0

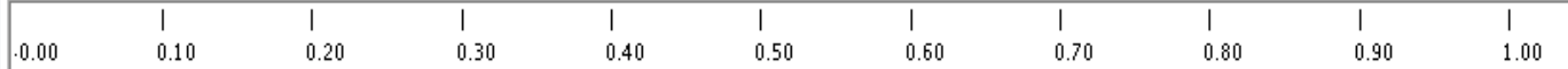
1

2

3



@ world\_rank



Time (seconds)



# Generate traces for servers

---

- Feature originally not available
- Concept: start PVFS as a pseudo MPI-program
  - small modification of PVFS needed
  - MPE is now available for logging
  - add MPE event generation into source code of PVFS
- Produces single trace for visualization with original Jumpshot
- Includes events from server processes and metadata server process



# Merge client and server traces

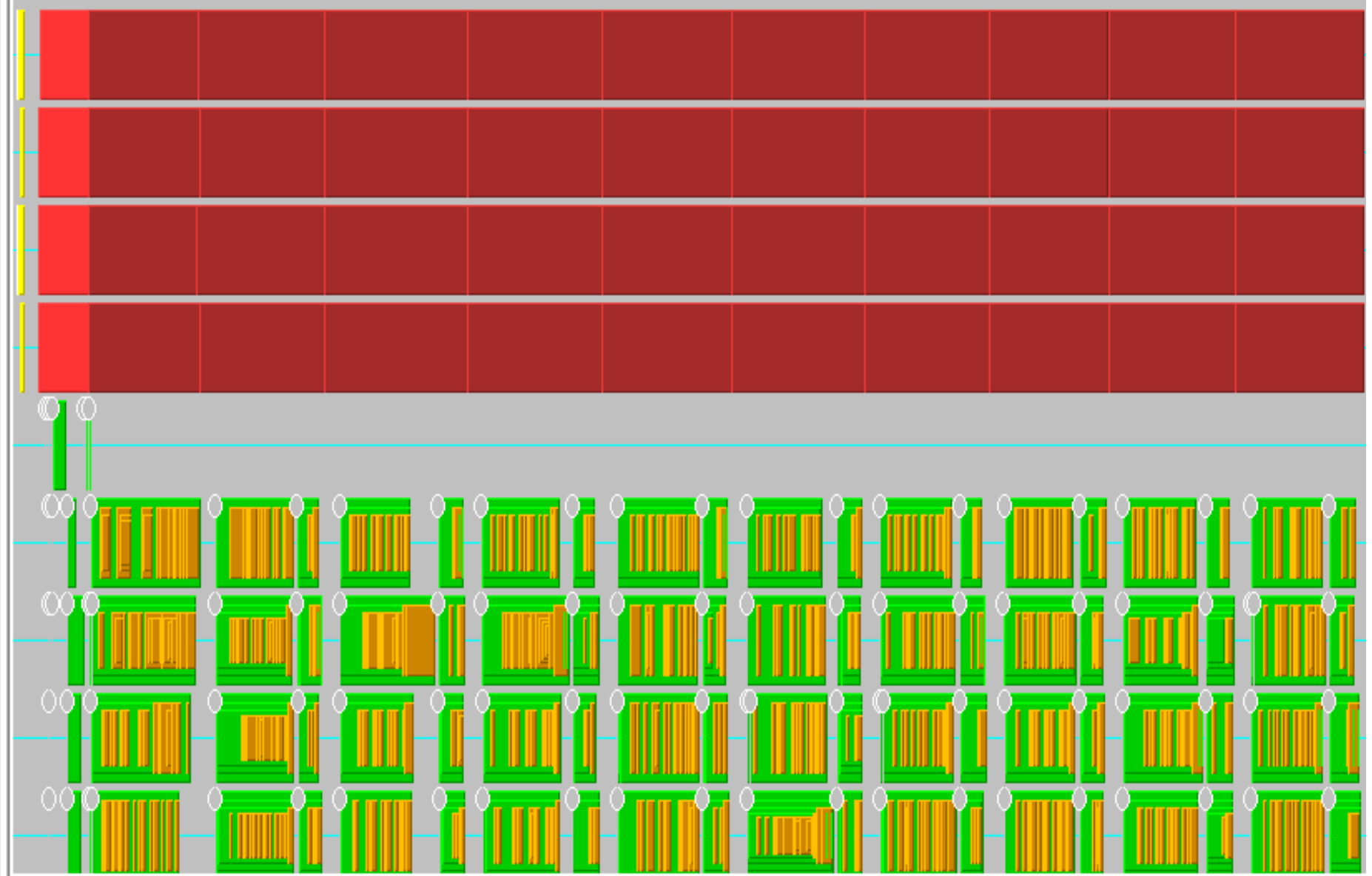
---

- Tool: **MergeSlog2** (Java)
- Purpose
  - merge client and server trace into single trace
  - adjust time information consistently
    - use special events to do this
  - rename ranks for servers

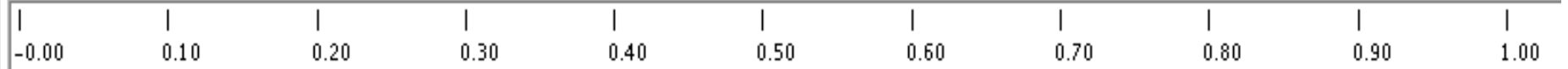
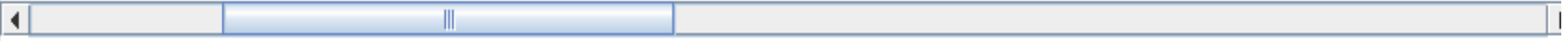
# merged client/server trace

SLOG-2

- 0
- 1
- 2
- 3
- 100
- 101
- 102
- 103
- 104



@ LineID



Time (seconds)



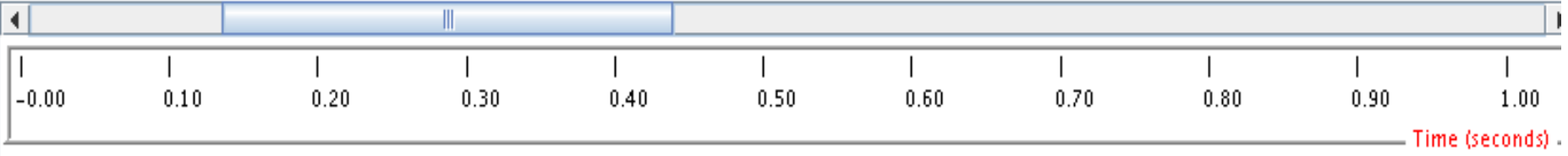
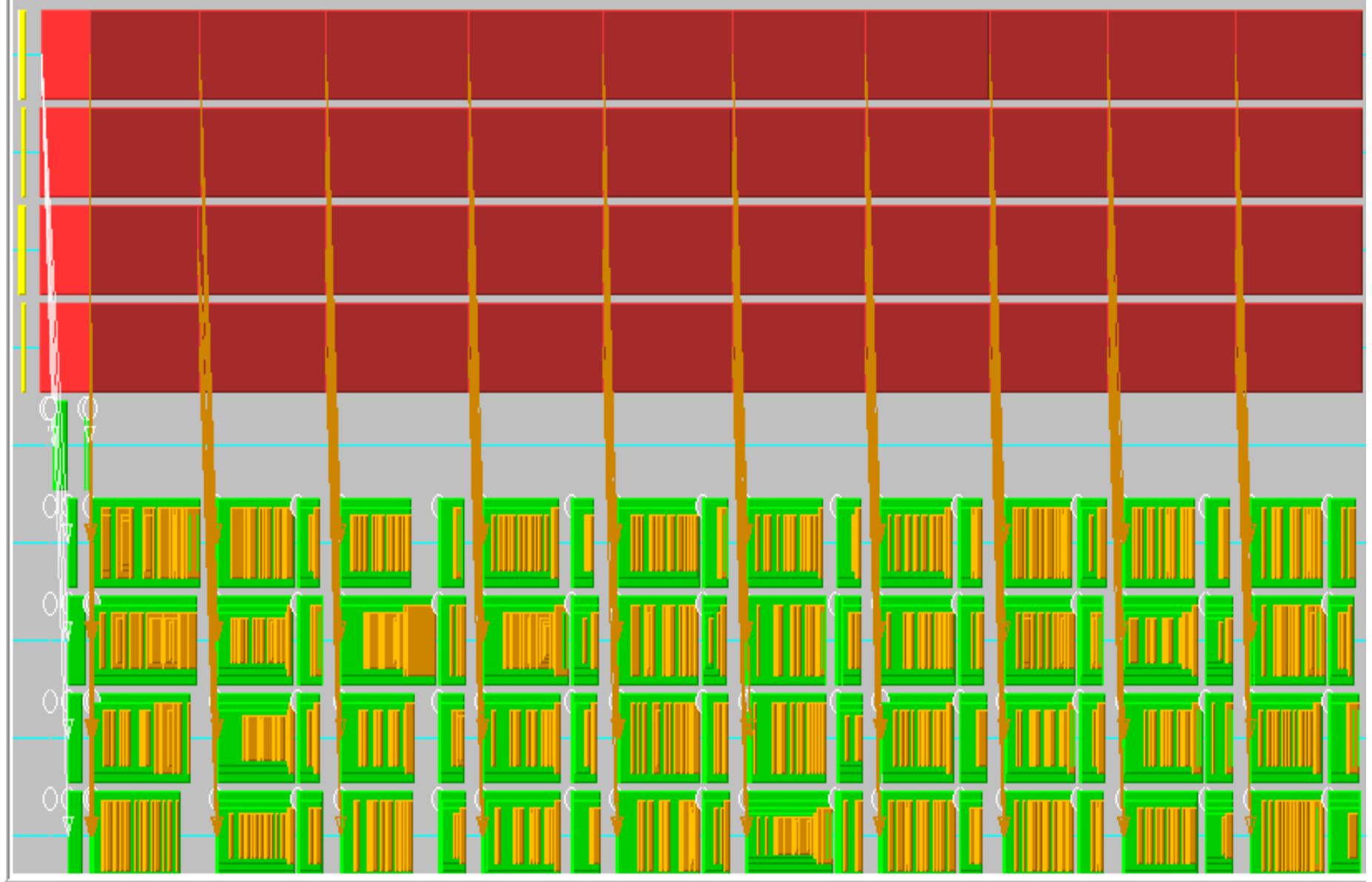
# Add arrows

---

- Tool: **Slog2ToArrowSlog2** (Java)
- Purpose
  - connect corresponding events in client and server trace with arrows
  - we want to see which MPI-IO operation triggers which PVFS Trove activity

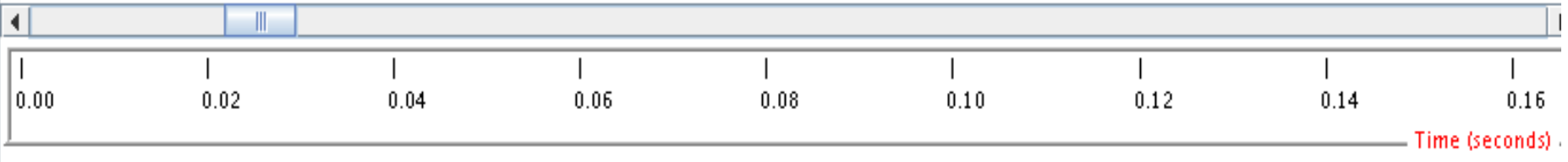
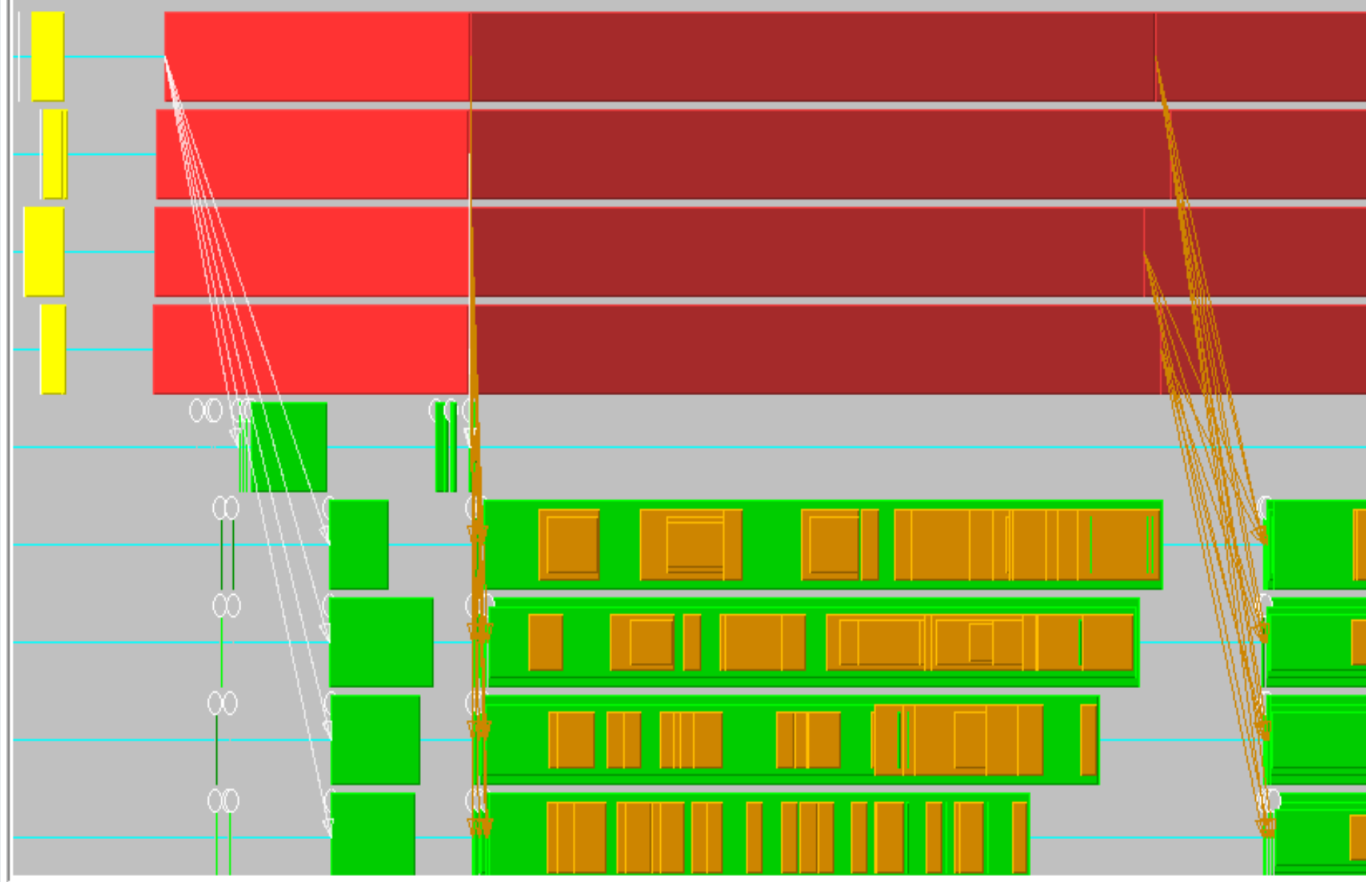
# merged client/server trace with arrows

- SLOG-2
- 0
- 1
- 2
- 3
- 100
- 101
- 102
- 103
- 104



# merged client/server trace with arrows

- SLOG-2
- 0
- 1
- 2
- 3
- 100
- 101
- 102
- 103
- 104



# Evaluation of Optimization Levels

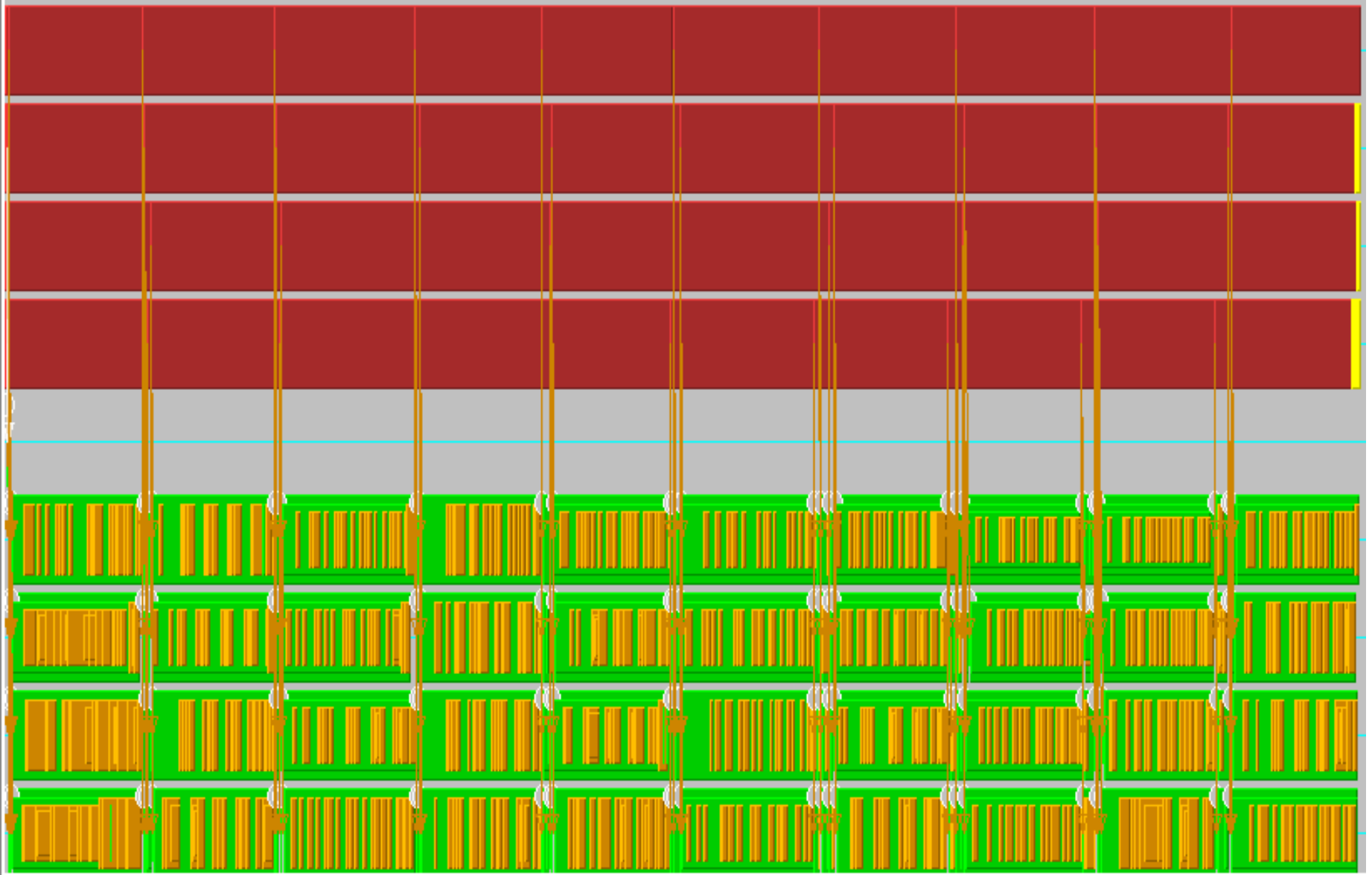


- Configuration: 4 servers, 4 clients
  - each on different nodes
- today only 1 example
  - **write 50MB/client**
- 4 I/O concepts
  - individual vs. collective calls
  - contiguous vs. non-contiguous data regions

4 servers / 4 clients / ind-ctg / write 4\*50MB

SLOG-2

- 0
- 1
- 2
- 3
- 100
- 101
- 102
- 103
- 104



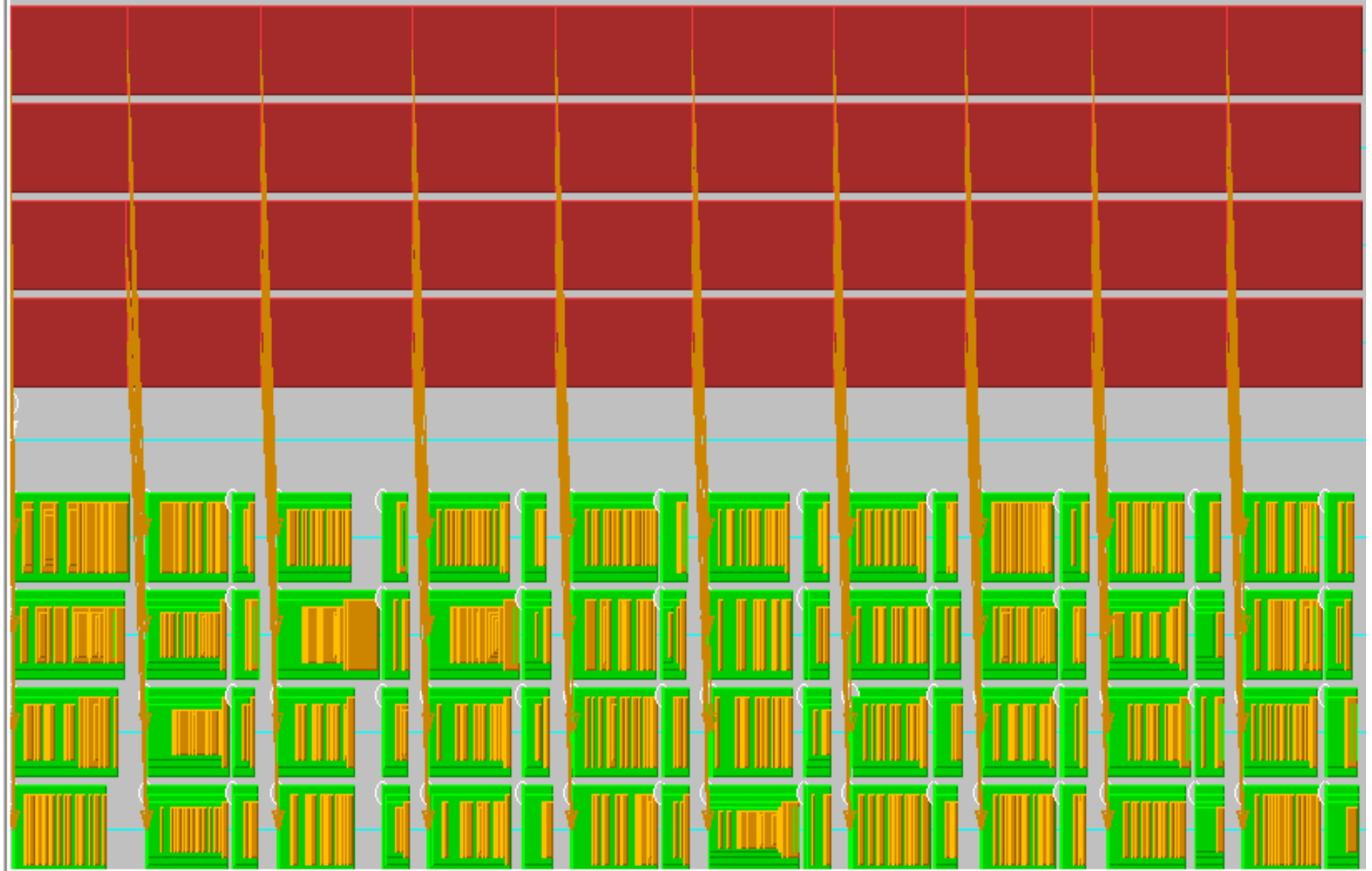
0.72s minimum is 0.53s

0,05 0,10 0,15 0,20 0,25 0,30 0,35 0,40 0,45 0,50 0,55 0,60 0,65 0,70 0,75

Time (seconds)

@ Servers  
@ Spreadlin

4 servers / 4 clients / col-ctg / write 4\*50MB



0.97s

0,10

0,20

0,30

0,40

0,50

0,60

0,70

0,80

0,90

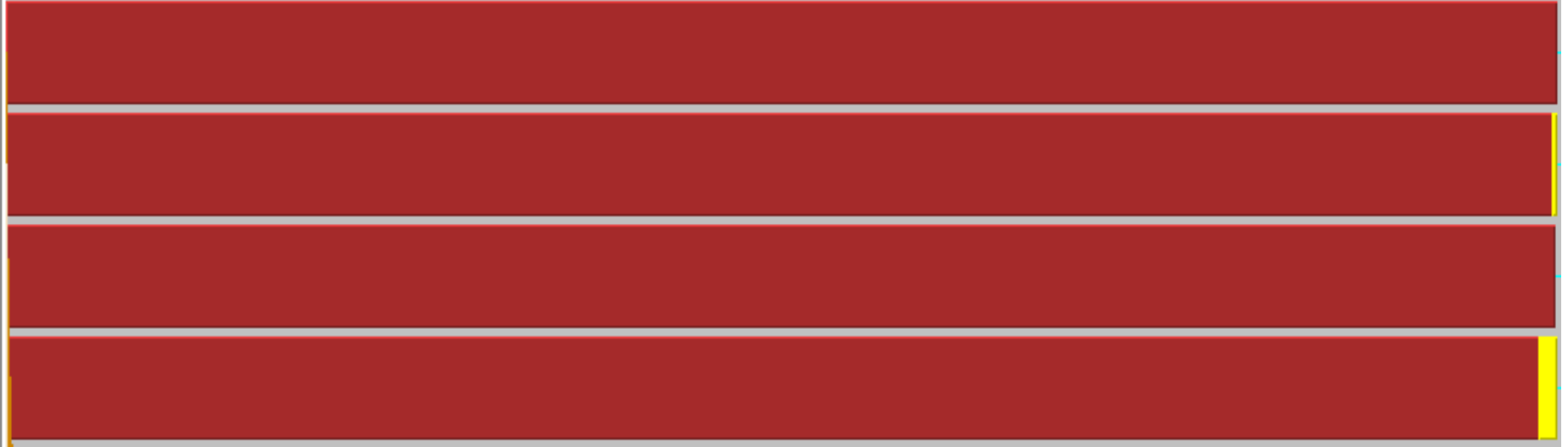
1,00

Time (seconds)

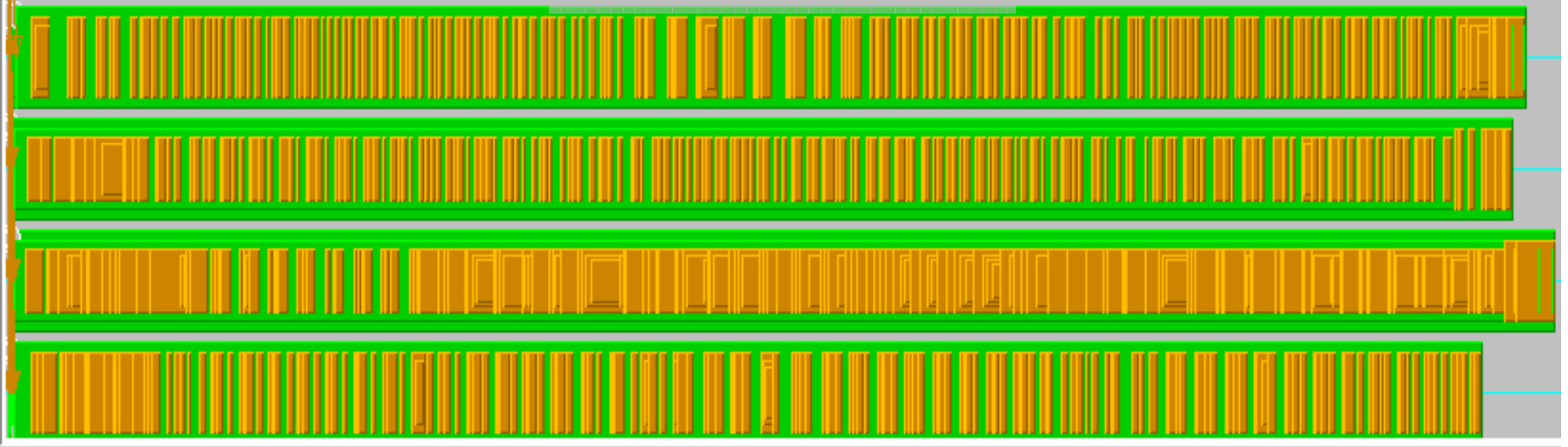
4 servers / 4 clients / ind-nctg / write 4\*50MB

SLOG-2

- 0
- 1
- 2
- 3
- 100
- 101
- 102
- 103
- 104



9 MPI calls less!



0.62s



Time (seconds)

4 servers / 4 clients / col-nctg / write 4\*50MB

SLOG-2

0

1

2

3

100

101

102

103

104

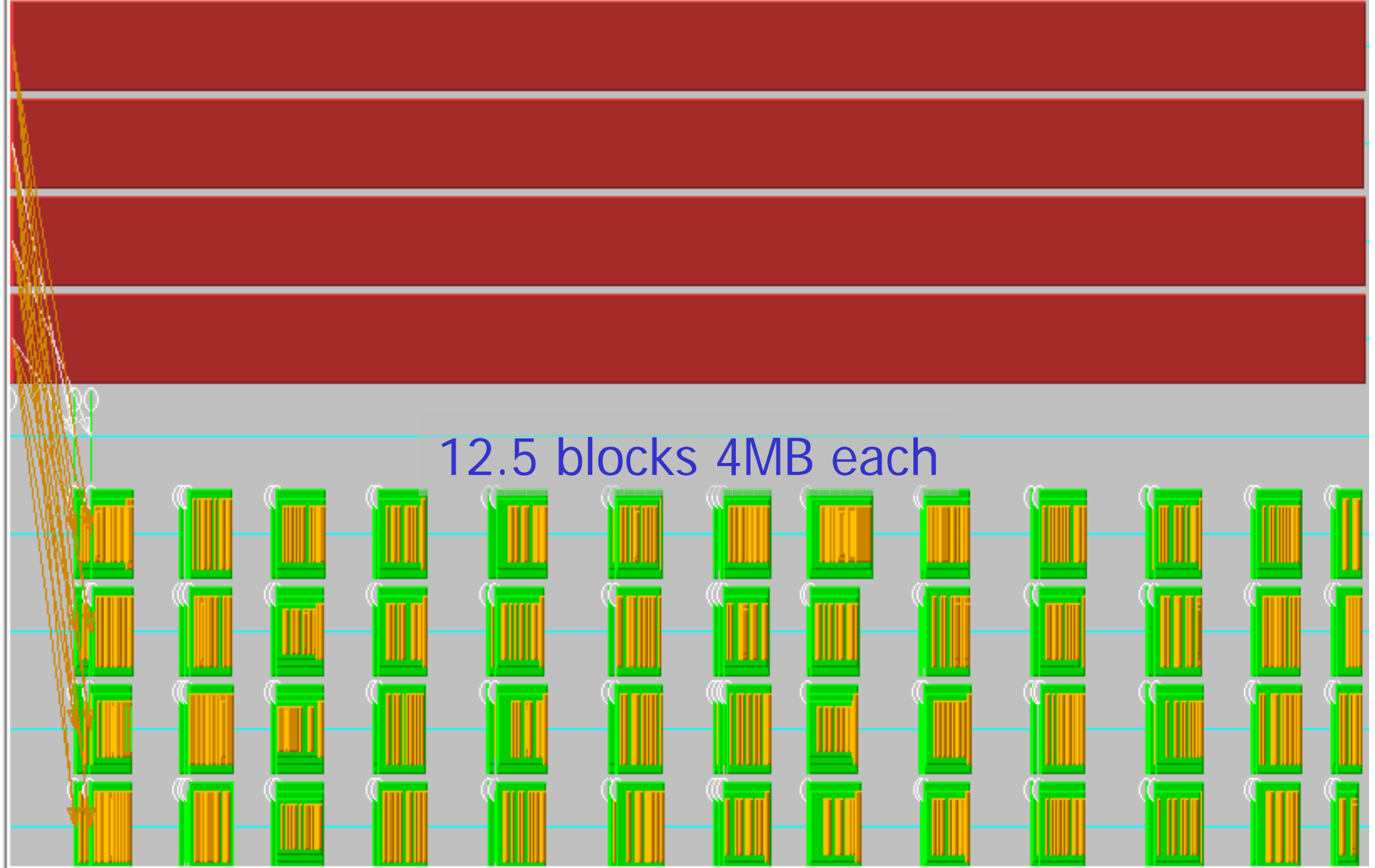
12.5 blocks 4MB each

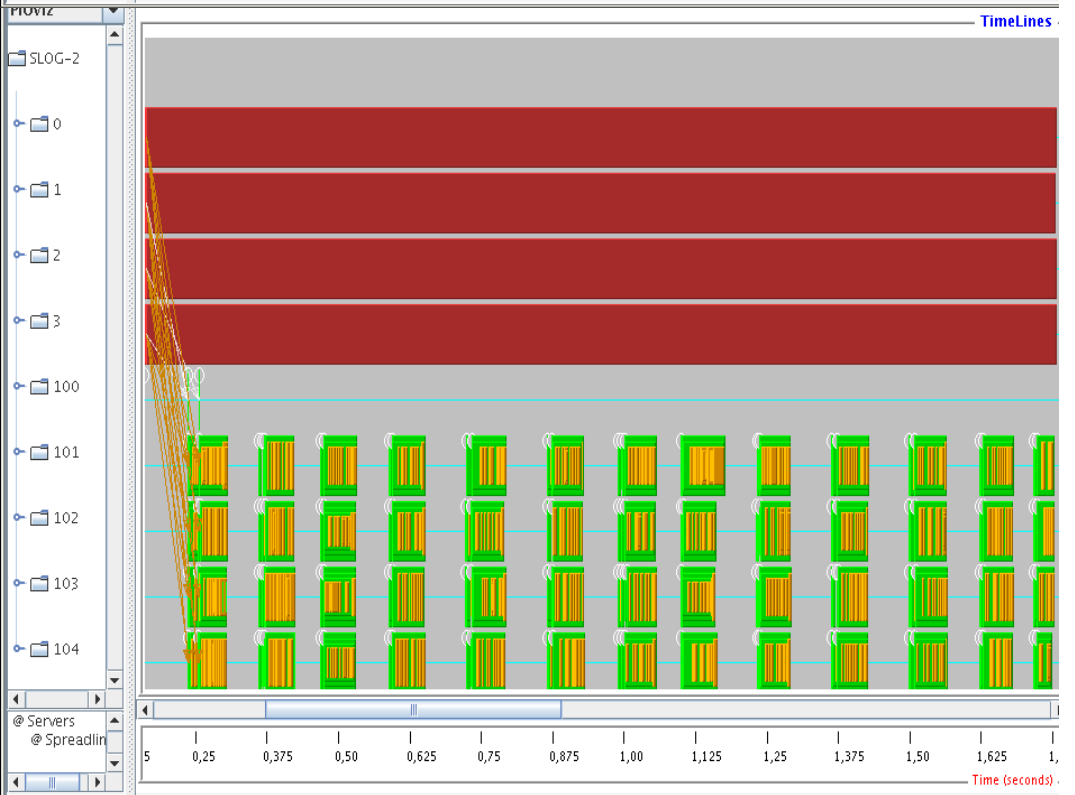
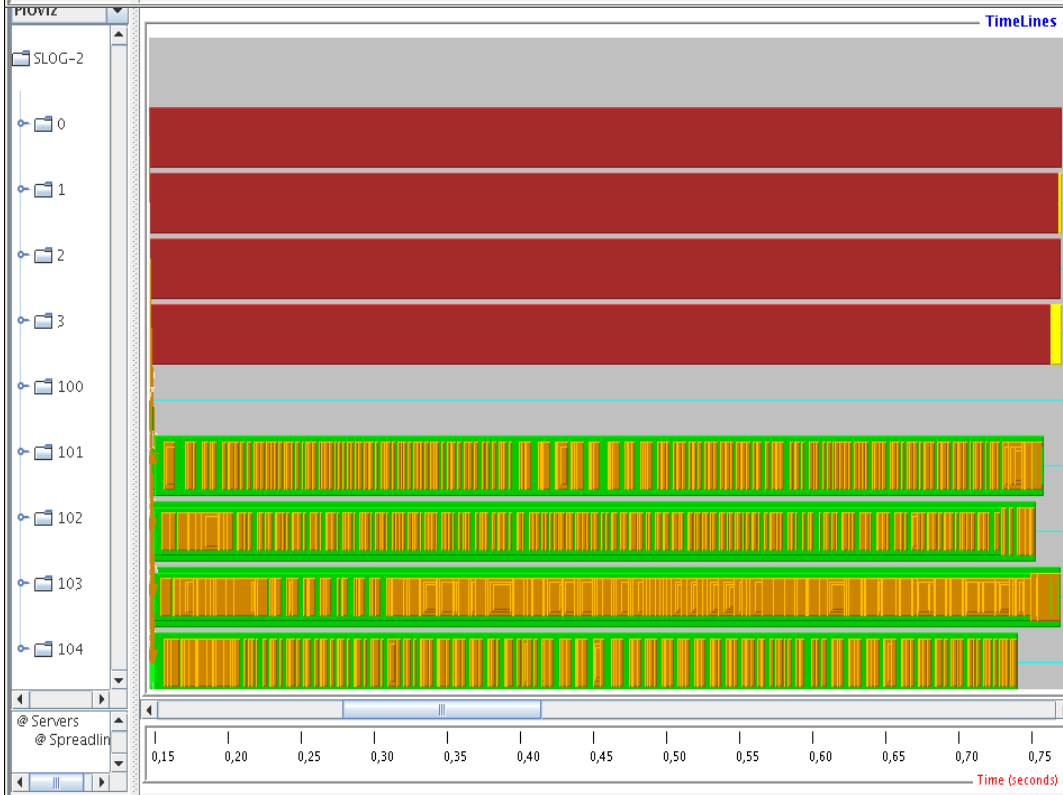
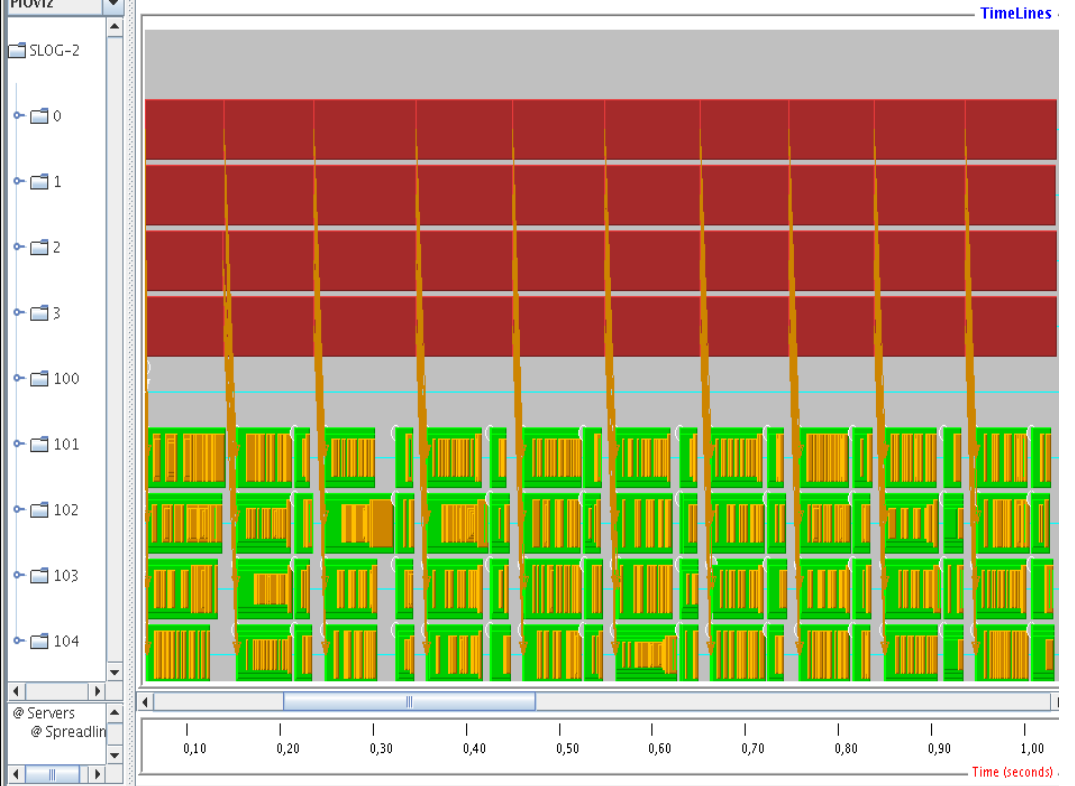
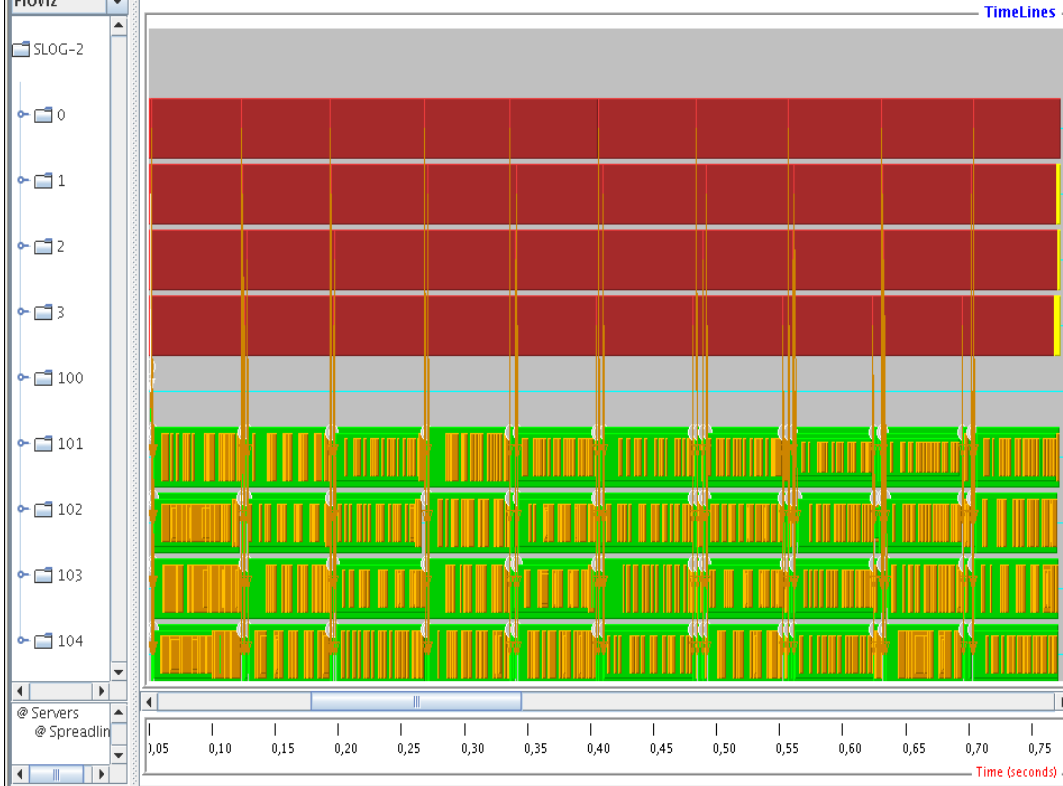
1.59s

5 0,25 0,375 0,50 0,625 0,75 0,875 1,00 1,125 1,25 1,375 1,50 1,625 1,75

Time (seconds)

@ Servers  
@ Spreadlin





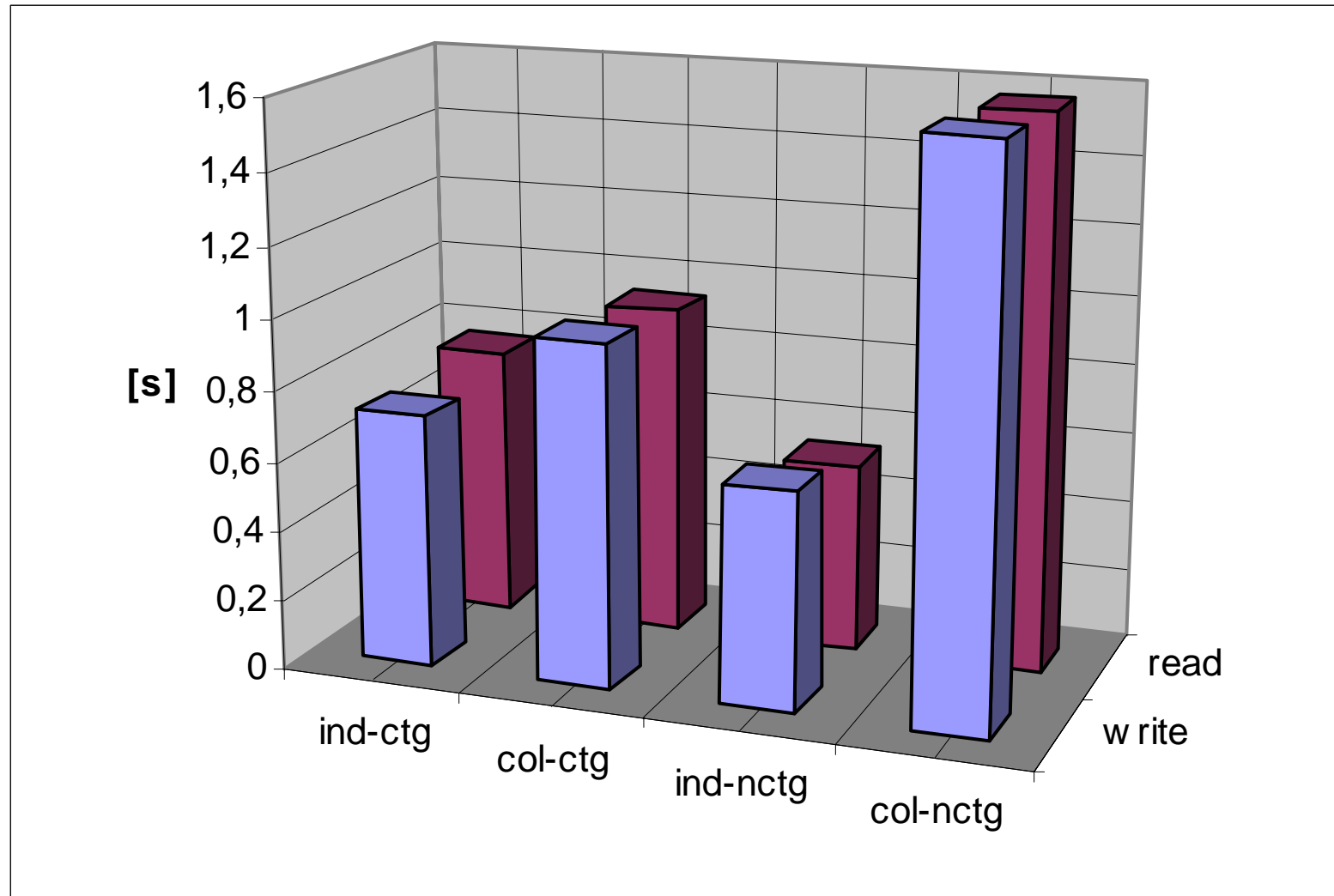


# Observations

---

- Contiguous collective calls last longer because time is determined by the slowest participant
- Non-contiguous collective calls last much too long because the communication in the two-phase protocol is not efficient
- Non-contiguous independent calls with data sieving have the best performance

# Summary 4\*50MB write - read





# What do we Learn from this?

---

It is **very difficult** to measure all these effects and to visualize them

However, it's **extremely difficult** to interpret these effects correctly

We do not know yet how it is to **control** these effects



# Using the Tracing Environment

---

- Test phase was successful
  - We detected some problems in PVFS
  - Several of them were implementation related
- We will now look at real applications
- Package with all components will be released soon in the public domain
  - Package already available – we currently test it



# Ongoing Work

---

## Measure more data

- We integrated performance counter values in our traces
- Can be visualized in the usual way

## Use data for on-line decisions

- We work on load-balancing via data file migration
- Prototype already functional

## Development of I/O simulator



# Finally...

David Bailey

*Twelve Ways to Fool the Masses When  
Giving Performance Results on Parallel  
Computers*

Supercomputer Review, August 1991

**(12) If all else fails, show pretty  
pictures and animated videos, and  
don't talk about performance**