

# Performance Visualization of MPI-IO Operations



**Thomas Ludwig**

Ruprecht-Karls-Universität Heidelberg

Computer Science Department

Parallel and Distributed Systems

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



# Outline

---

- Part I
  - Why parallel I/O and how to do it?
- Part II
  - How to model I/O performance?
- Part III
  - Project goals and general concepts
- Part IV
  - Evaluations of I/O performance



# PART I

---

- I/O and high volume I/O
- Parallel I/O in programs
- Parallel I/O subsystems



# 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



# Modern I/O-Concepts

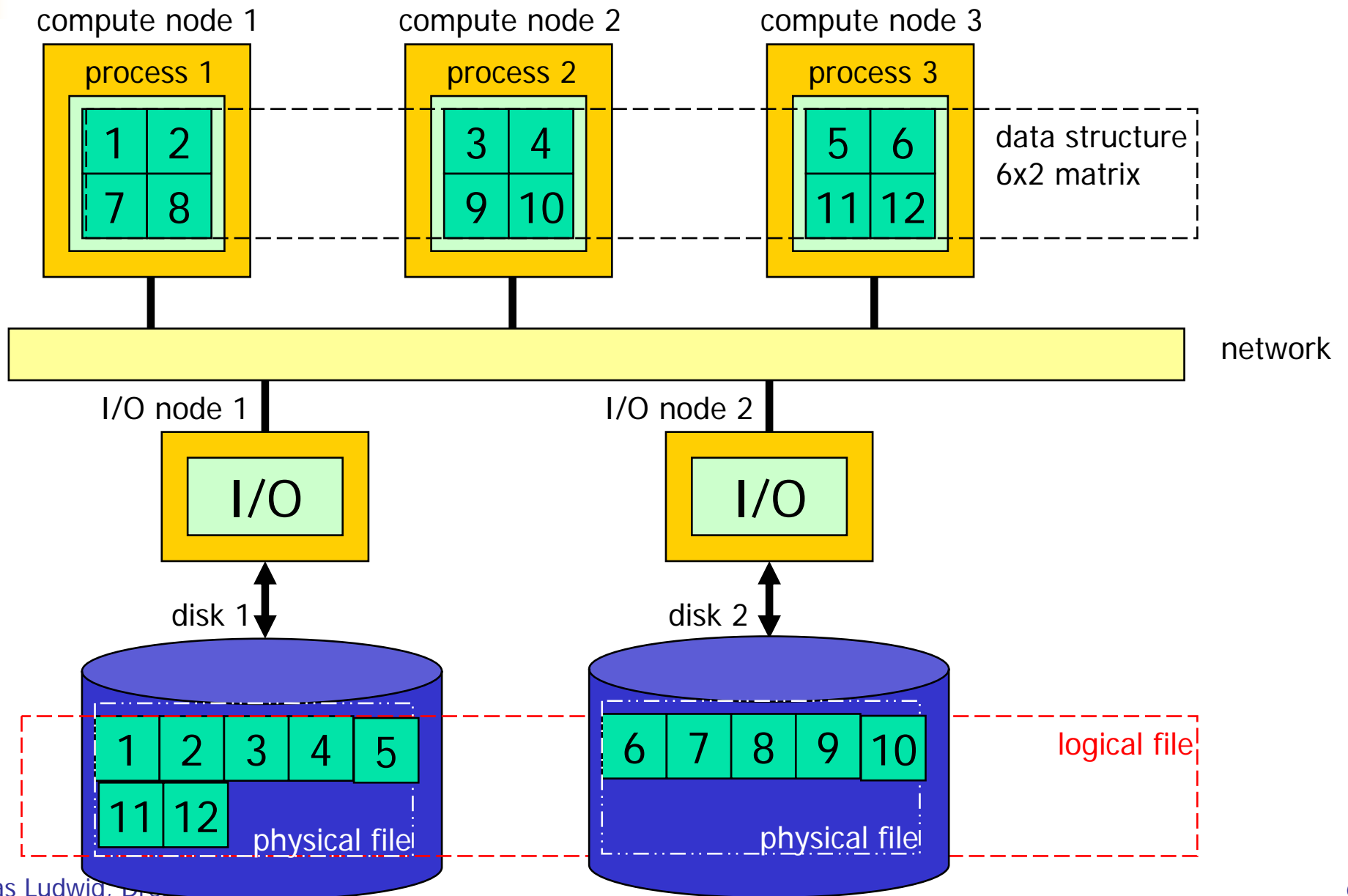
---

## 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





# 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/SUN): powerful open source system
  - GPFS (IBM): older proprietary approach

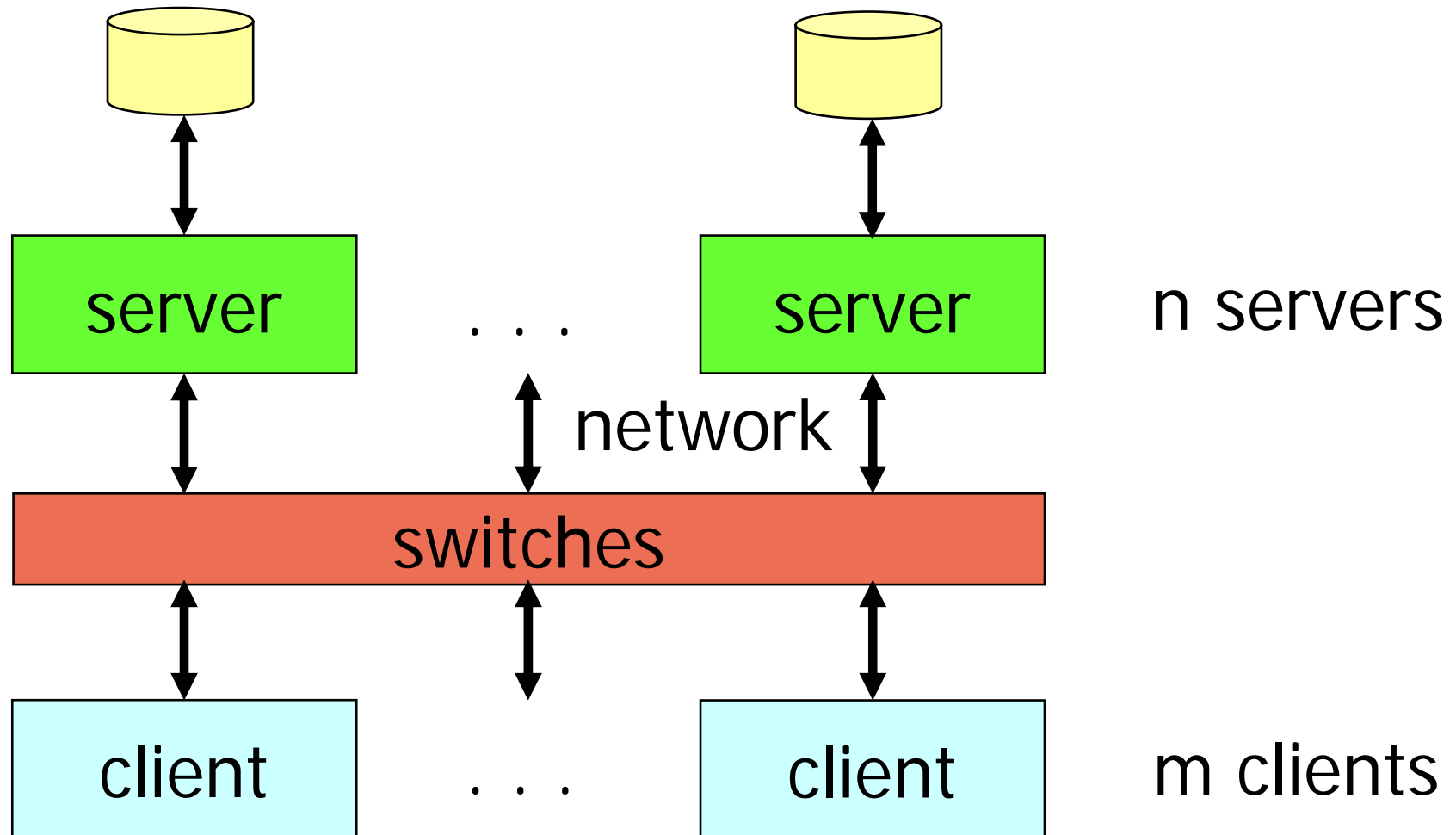


# PART II

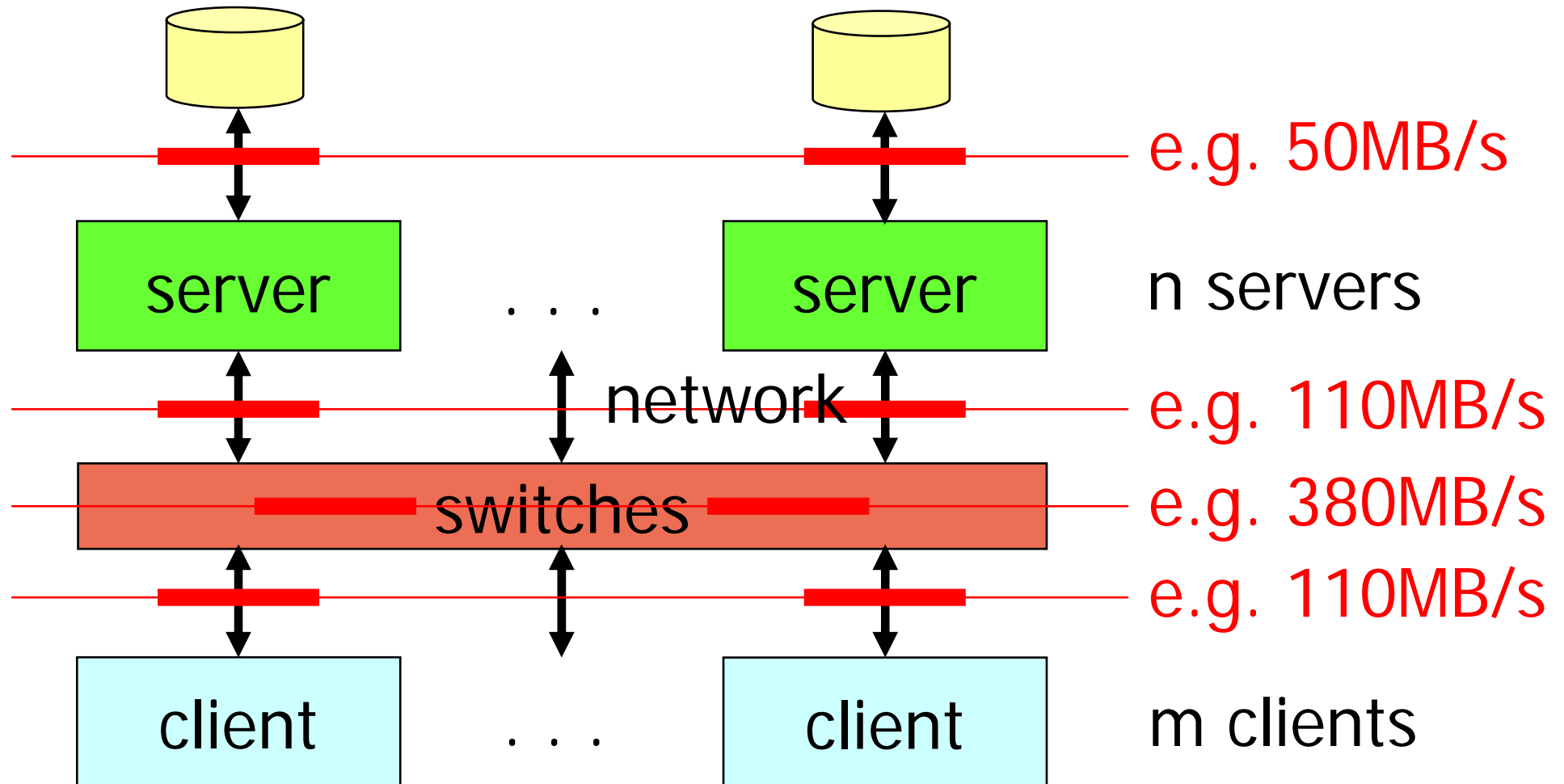
---

- Modelling the I/O performance of the hardware
- What the software will do
- What we need

# Hardware Model



# Hardware Model





# Performance Modelling

---

- Performance of short operations is dominated by latencies
  - disk access latency
  - message transfer latency
- Performance of long operations is dominated by bandwidths
  - disk I/O bandwidth
  - network bandwidth
  - bisection bandwidth
- Performance is influenced by caches
  - In order to avoid effects of caches the amount of data read or written must be high



# Balanced System

---

In a balanced system the client I/O performance requirements and the provided server I/O performance are identical

Usually that would mean:

we have a different number of clients and servers



# Problems in Real Systems

---

- Tuning-mechanisms at all hierarchical layers
  - disk read-ahead and disk caching
  - I/O buffering by the operating system
  - tuning in the message protocol stack
  - optimization in the I/O libraries
- Furthermore
  - we do not know exactly the clients' I/O behaviour and demands



# Software Model

---

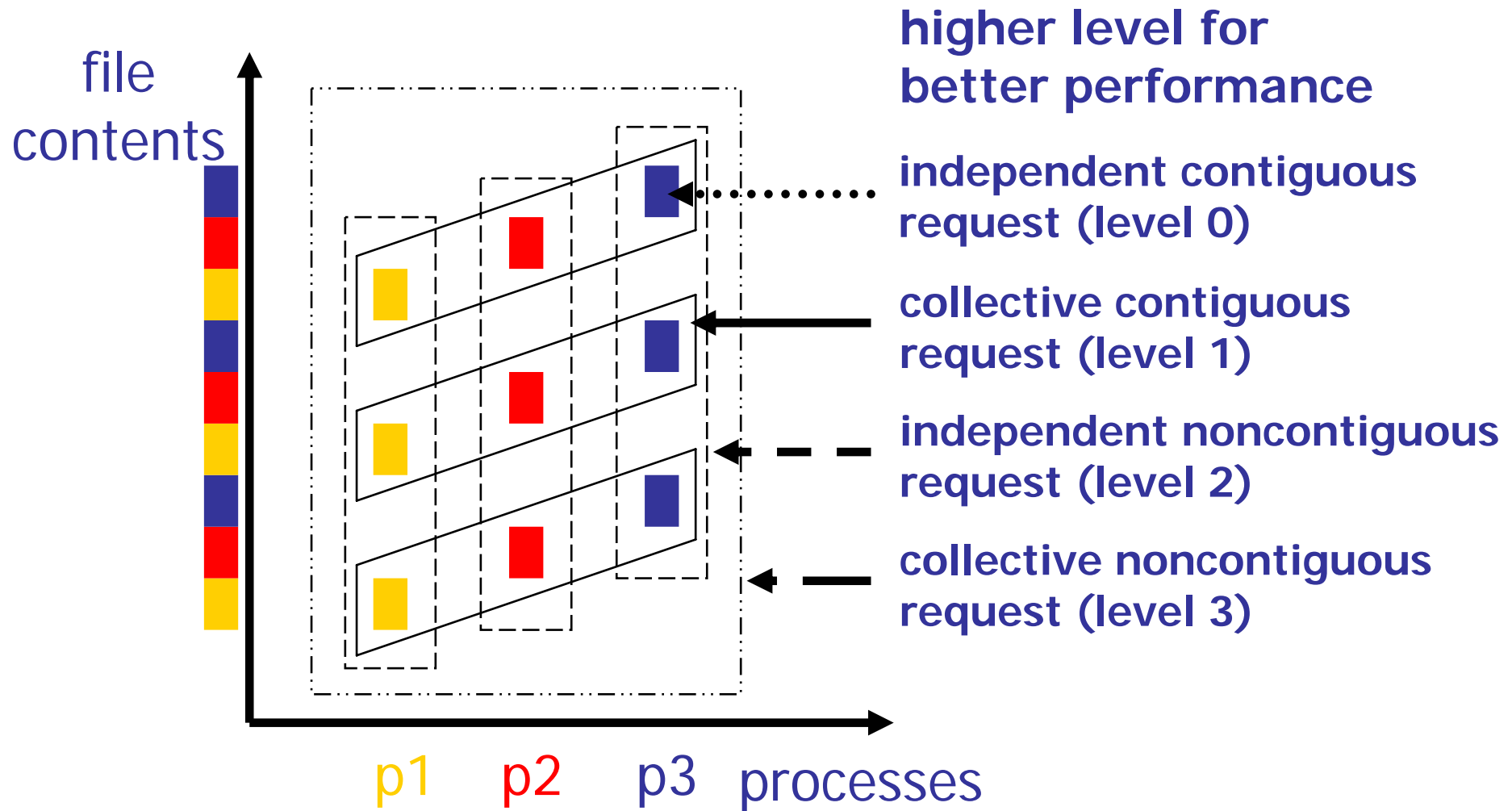
User-level optimizations in MPI-IO

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

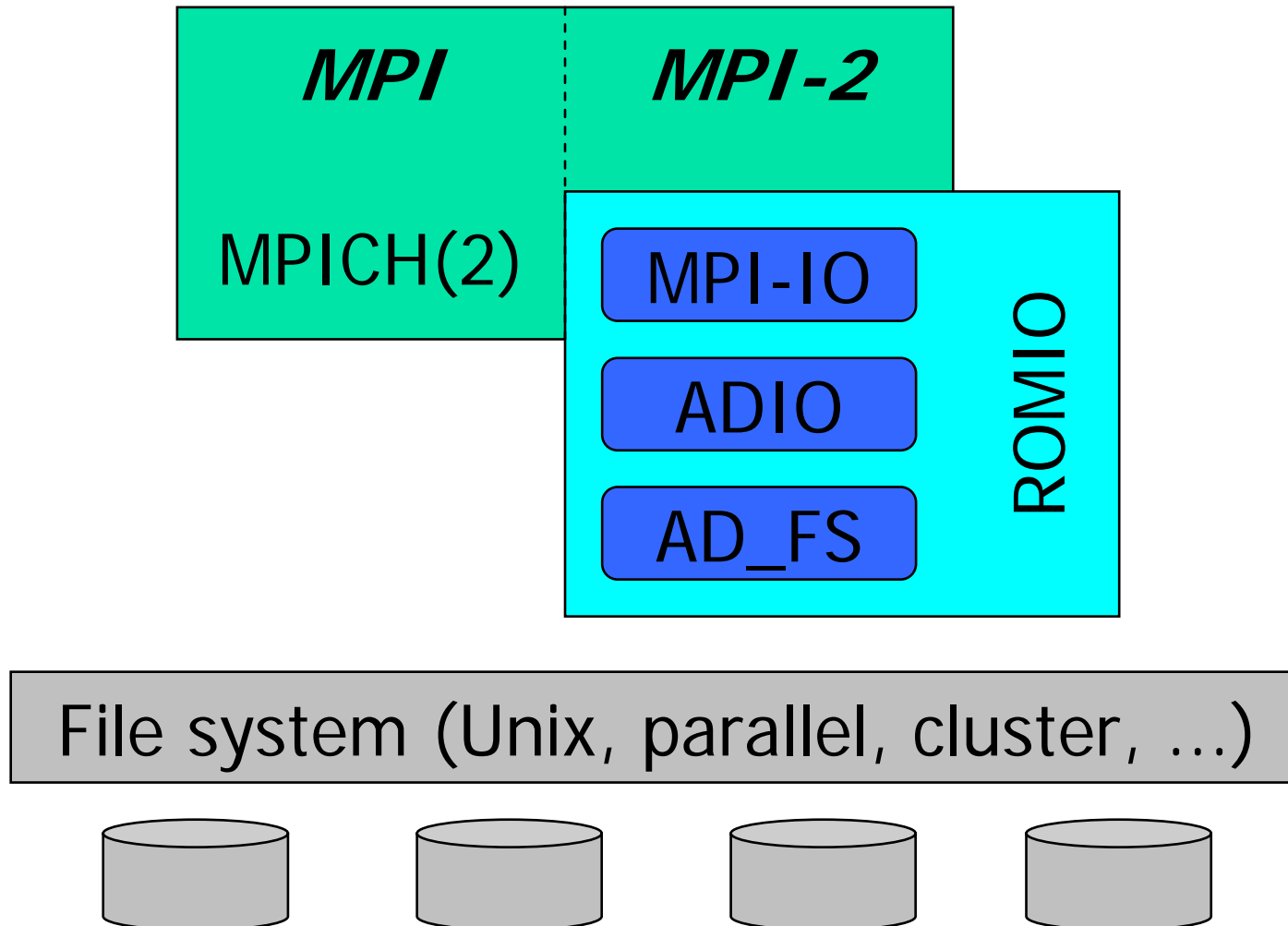
Realized by different calls with the API

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

# MPI-IO API Variations

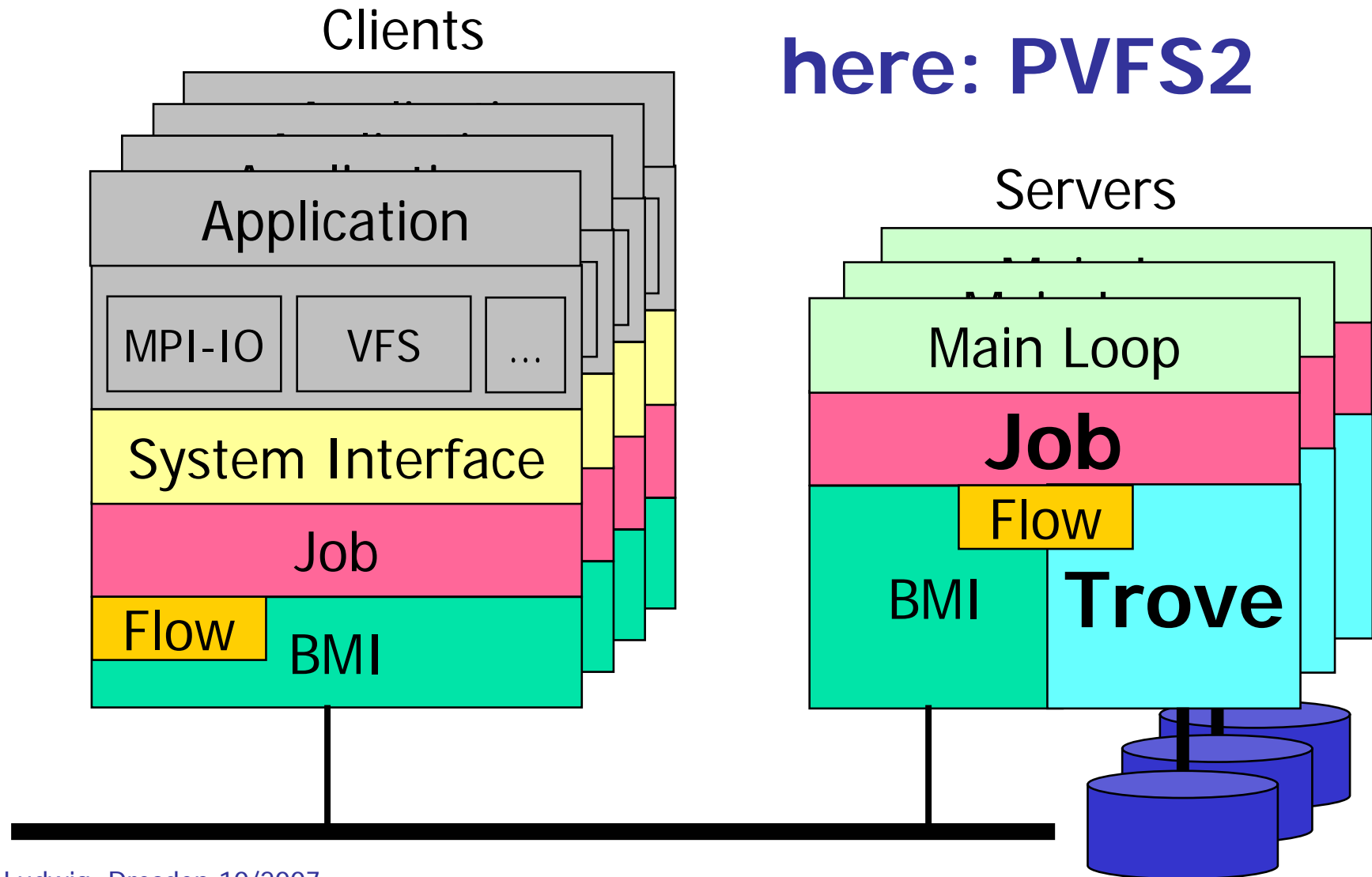


# The I/O-Client



# The Parallel File System

## Server Process + Client Library





# ROMIO-Optimizations

---

- Data Sieving

Level 2 (non-contiguous)

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

- Two-Phase Protocoll

Level 3 (collective and non-contiguous)

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



# What do we Need?

---

**We need detailed insight into client  
and server behaviour**

## **Goals**

- tune application optimally
- tune I/O subsystem optimally



# PART III

---

- The Problem of I/O Performance Evaluation
- Project Goals
- Generation of Trace Data
- Visualization of Trace Data



# 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 of the two of them

## Environment

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



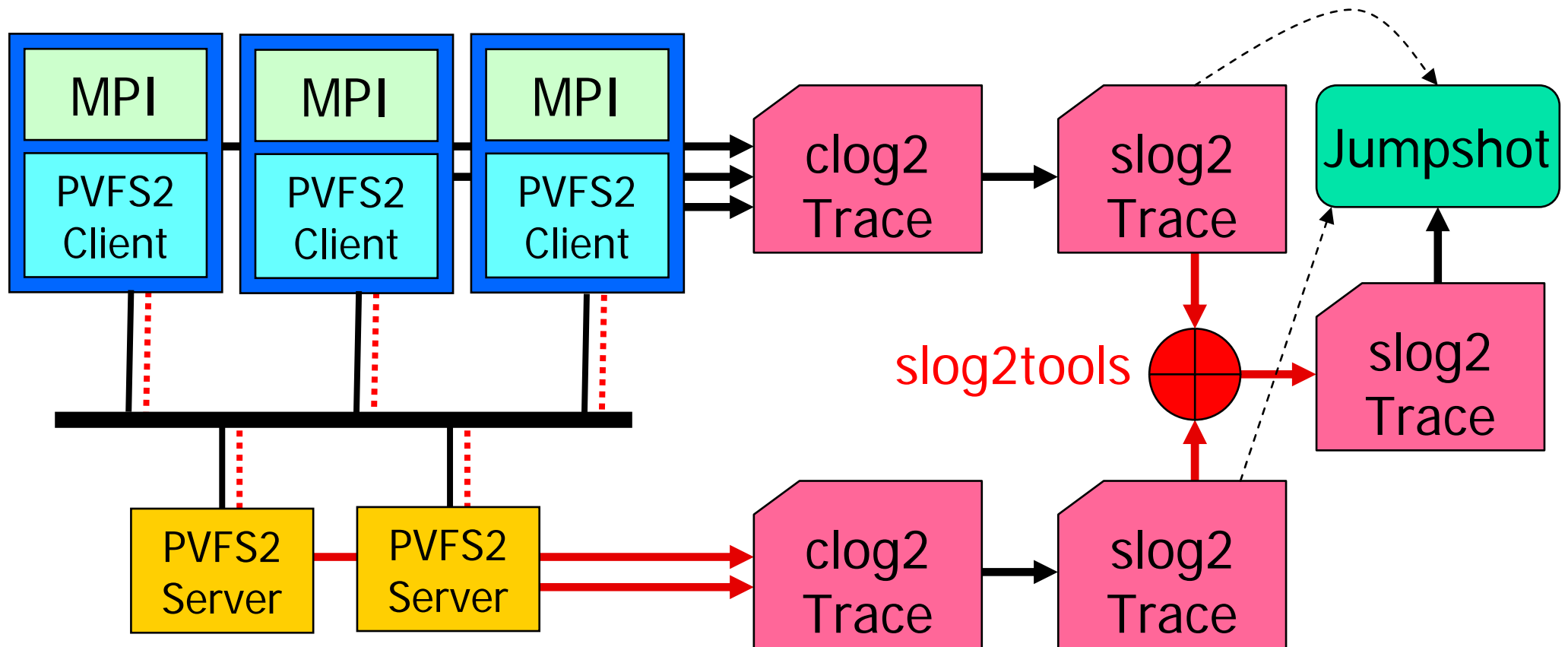
# Concepts

---

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

# 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
- Convert single events to states
- Eliminate state overlapping
- 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)

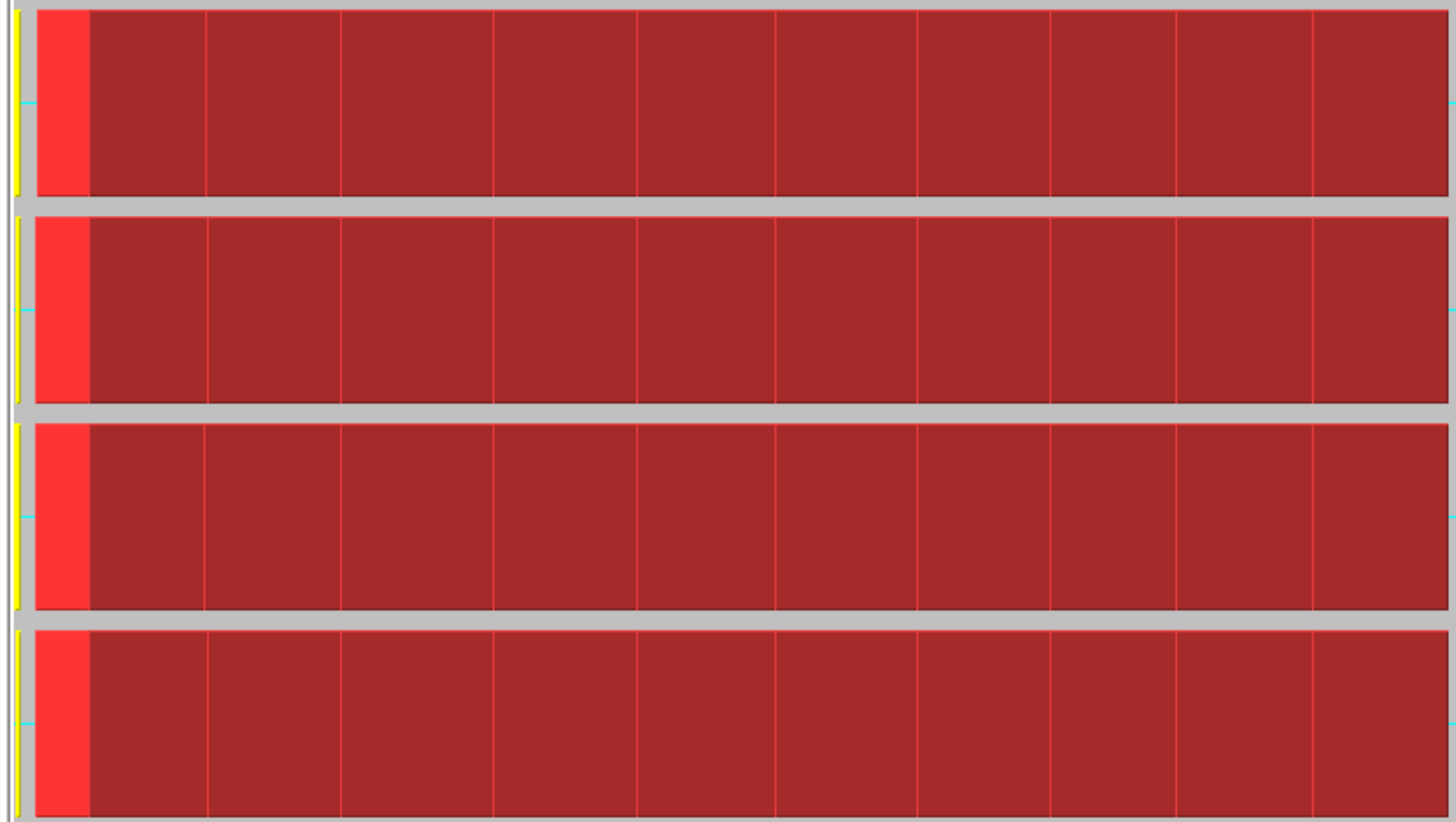
SLOG-2

0

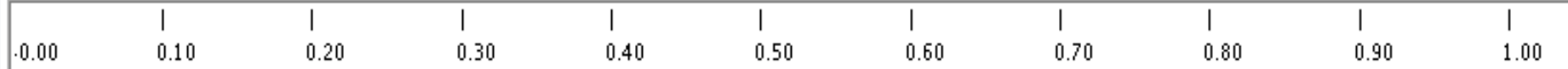
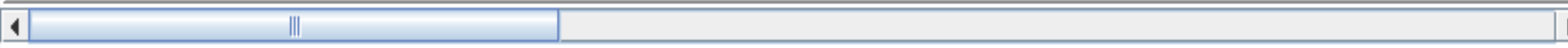
1

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

# 4 servers / 1 MD server trace

SLOG-2

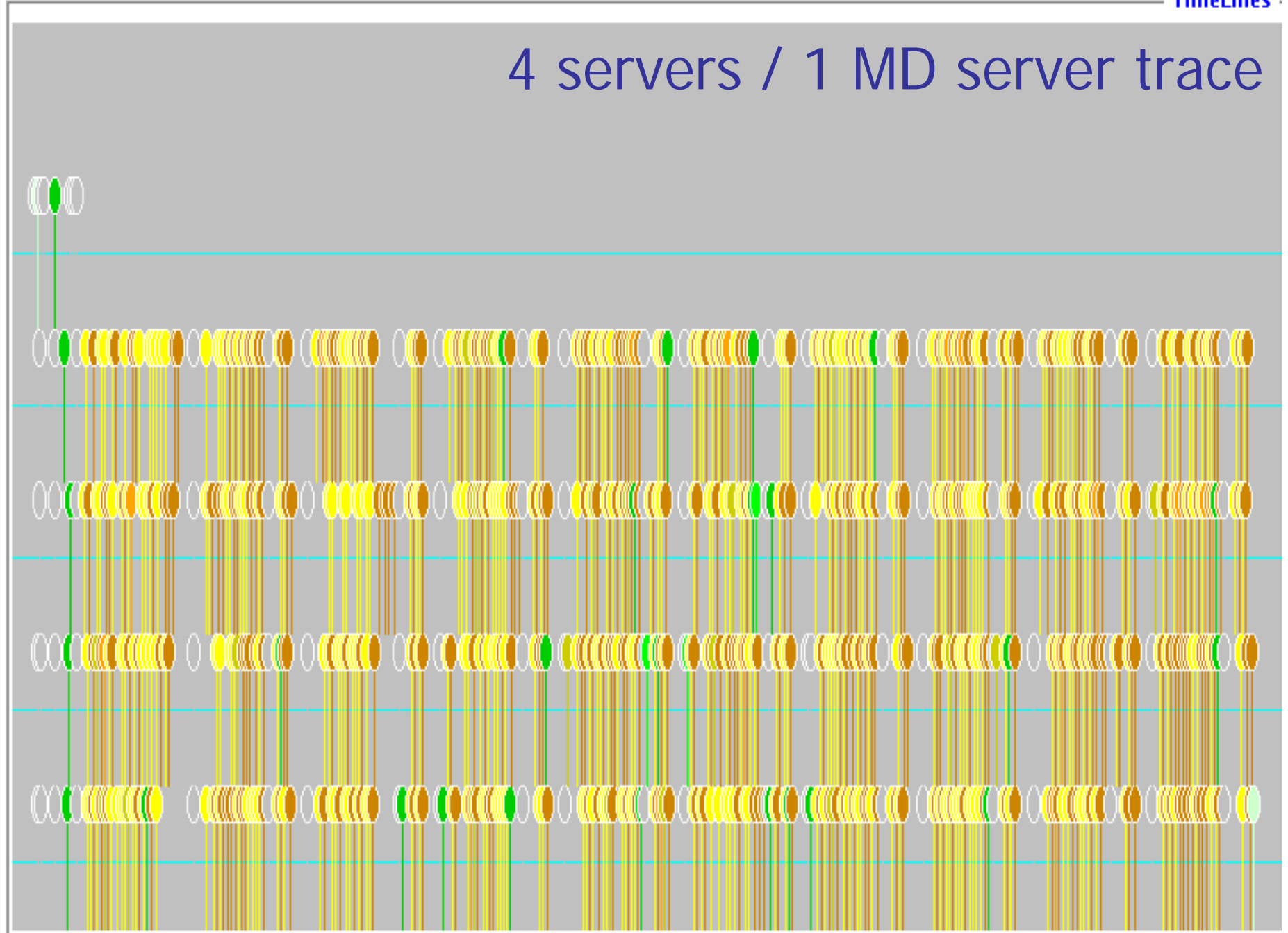
0

1

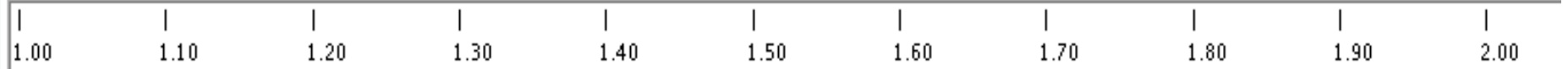
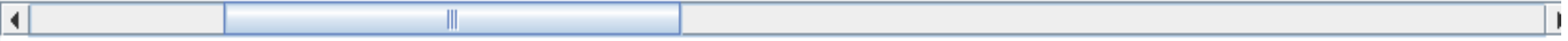
2

3

4



@ world\_rank



Time (seconds)



# Observations for server trace

---

- Time line is shifted because servers were started earlier
  - Problem: adjust time lines for clients and servers
- Information is only available in form of single events
  - Problem: overlapping activities in servers caused by multiple clients' requests
- Process rank value for servers starts with 0
  - Problem: conflict with clients



# Merge client and server traces

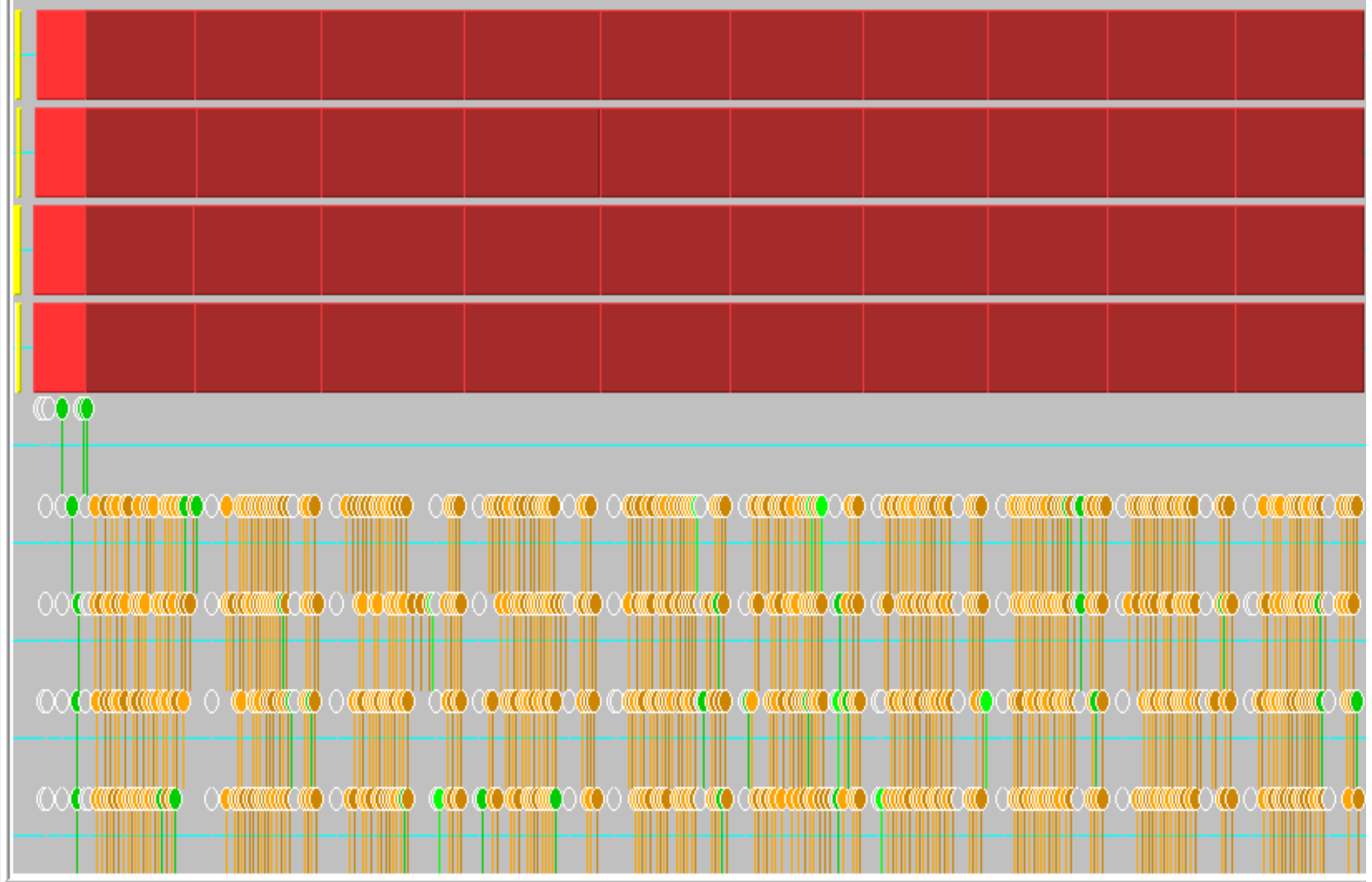
---

- 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

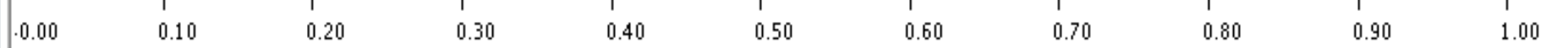
# merged client/server trace

SLOG-2

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



@ LineID



Time (seconds)



## Observations for server trace

---

- Time lines adjusted
- Server processes renamed
- Still no server states visible
- Still no logical relation between events in client and server processes



# Convert single events to states

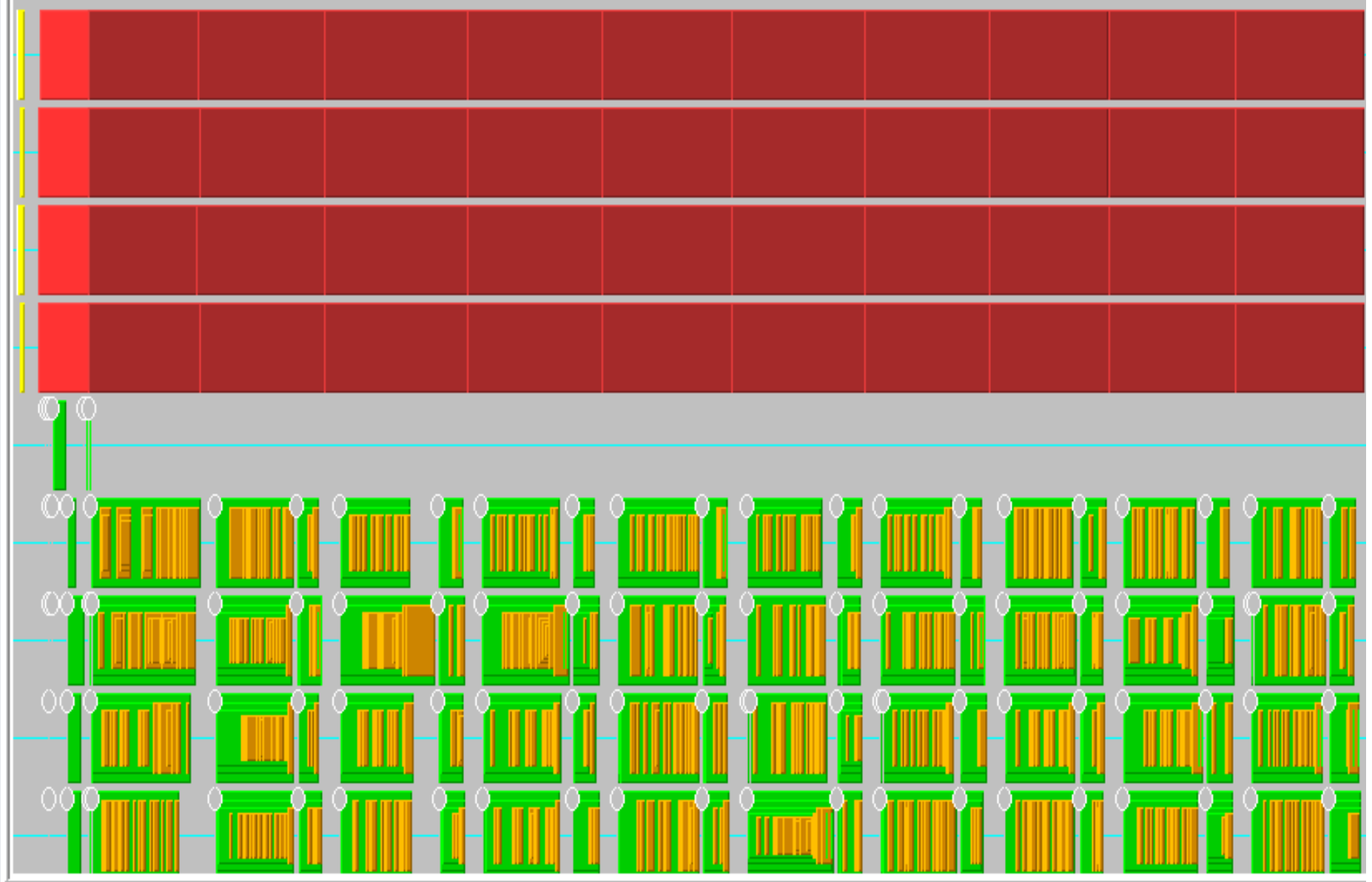
---

- Tool: **EventToState** (Java)
- Purpose
  - identify corresponding start and end events for various server activities
  - replace single events by one state entry in the trace file
  - add information like caller ID, time values etc. (available in end event)

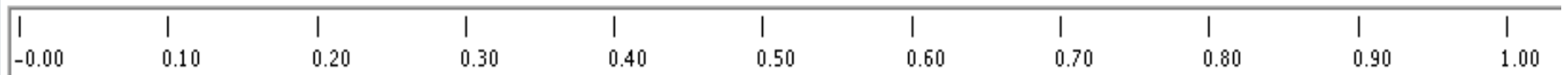
# merged client/server trace with states

SLOG-2

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



@ LineID



Time (seconds)



## Observations for state information

---

- Overlapping states on single time lines
- Yet no real insight into server behaviour
  
- Still no logical relation between events in client and server processes



# Eliminate state overlapping

---

- Tool: **CompositeSlog2ToLineIDMap**  
(Java)
- Purpose
  - distribute states on single time line over several new (sub-)time lines
  - guarantee non-overlapping states

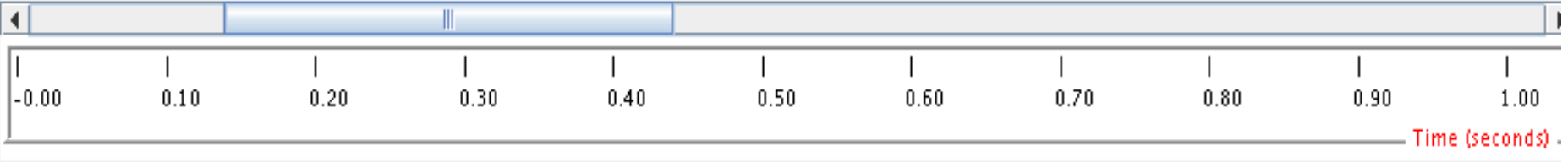
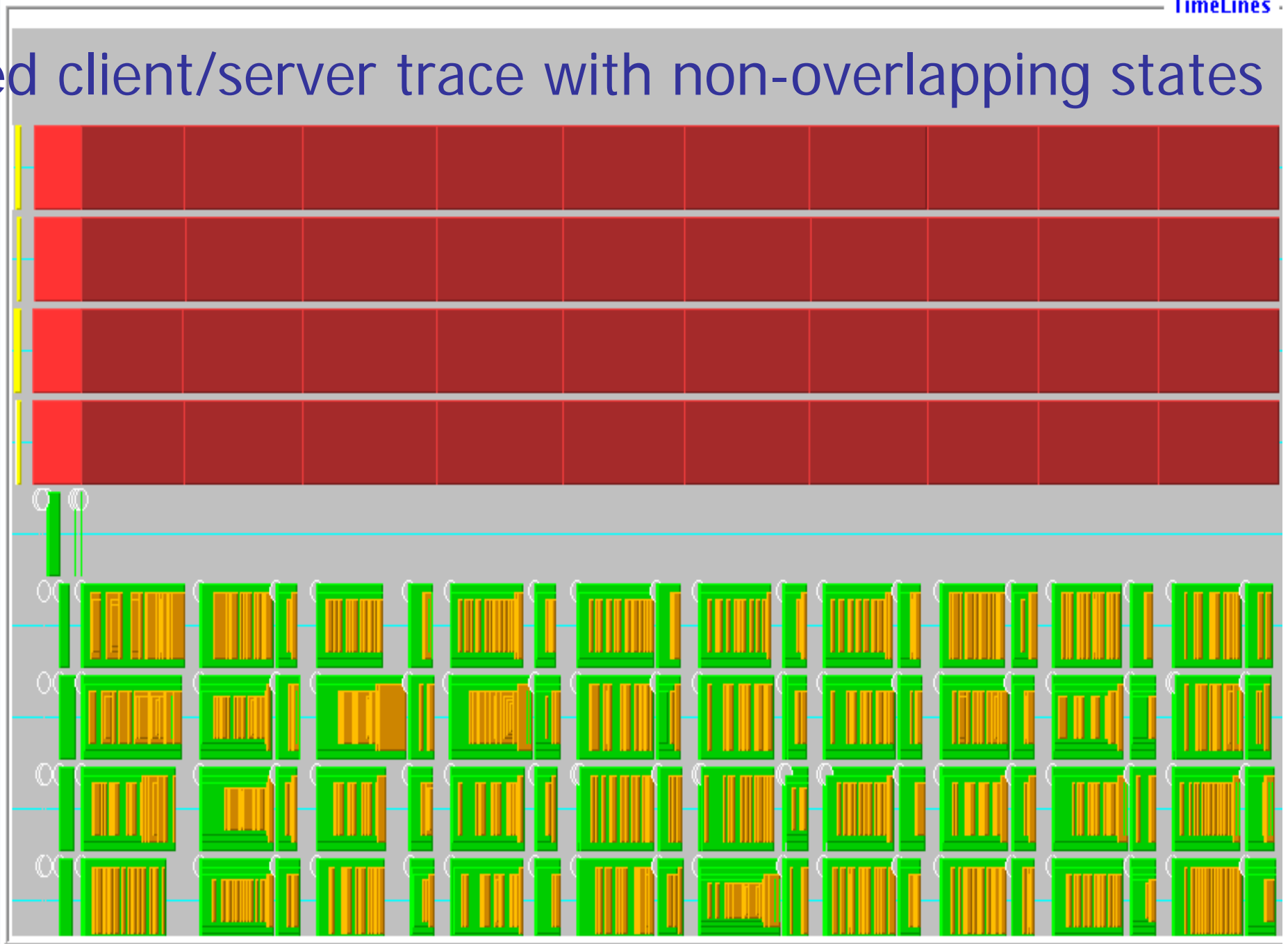
# merged client/server trace with non-overlapping states

NOVIZ

SLOG-2

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

@ Servers  
@ Spreadlin

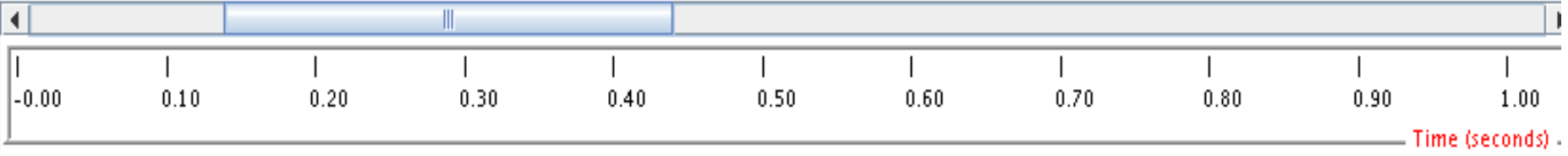


# merged client/server trace with non-overlapping states

NOVIZ

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

@ Servers  
@ Spreadlin





# Observations for multiple sub-time lines

- This is necessary because of
  - several clients contact one server with requests AND
  - request execution is handled via state machines in an interlaced way
- Still no logical relation between events in client and server processes



# 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



# Add Arrows to the Trace

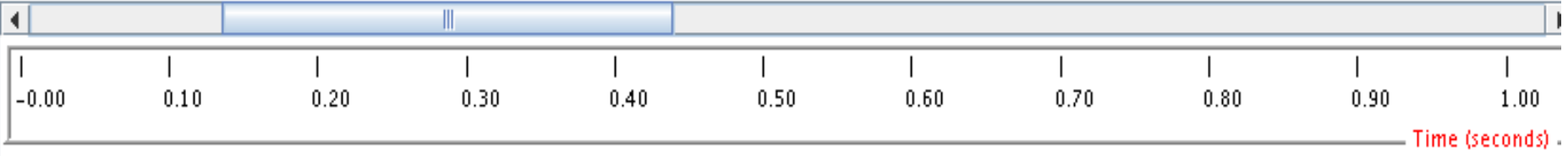
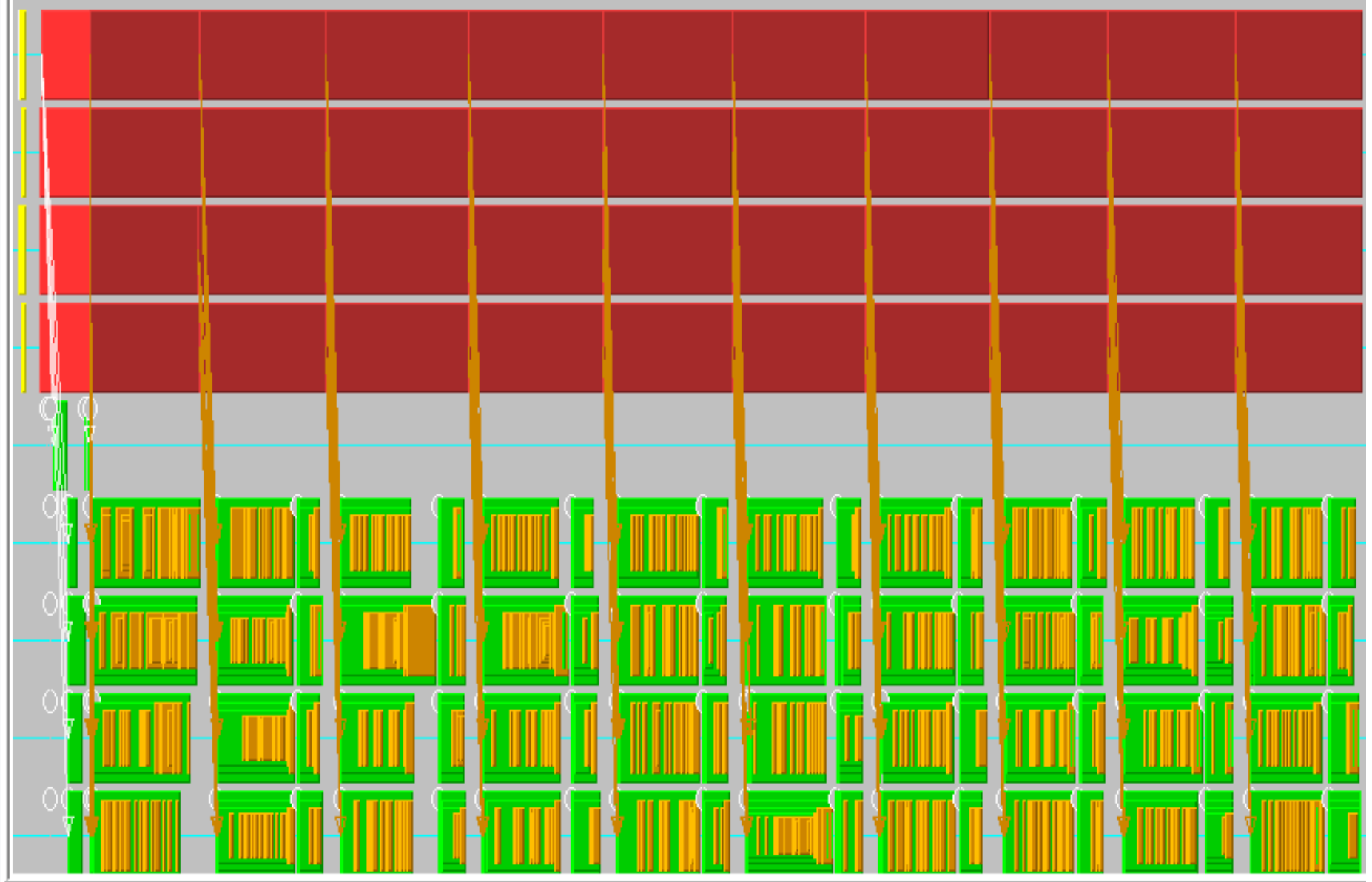
- Arrow semantics
  - We want to show which MPI-IO call results in which PVFS server activities (data, metadata)
  - 1:n relation between program and I/O-system
- What do we need?
  - We need a unique ID for every MPI-IO call
- Concept
  - Trace activities at the client (create IDs)
  - Transfer IDs to the servers
  - Trace activities at the servers (store IDs in trace)
  - Connect activities with identical IDs at both levels

... trace with non-overlapping states and arrows

SLOG-2

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

@ Servers  
@ Spreadlin

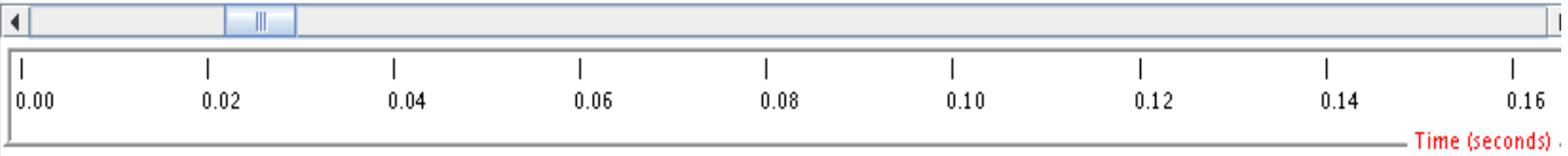
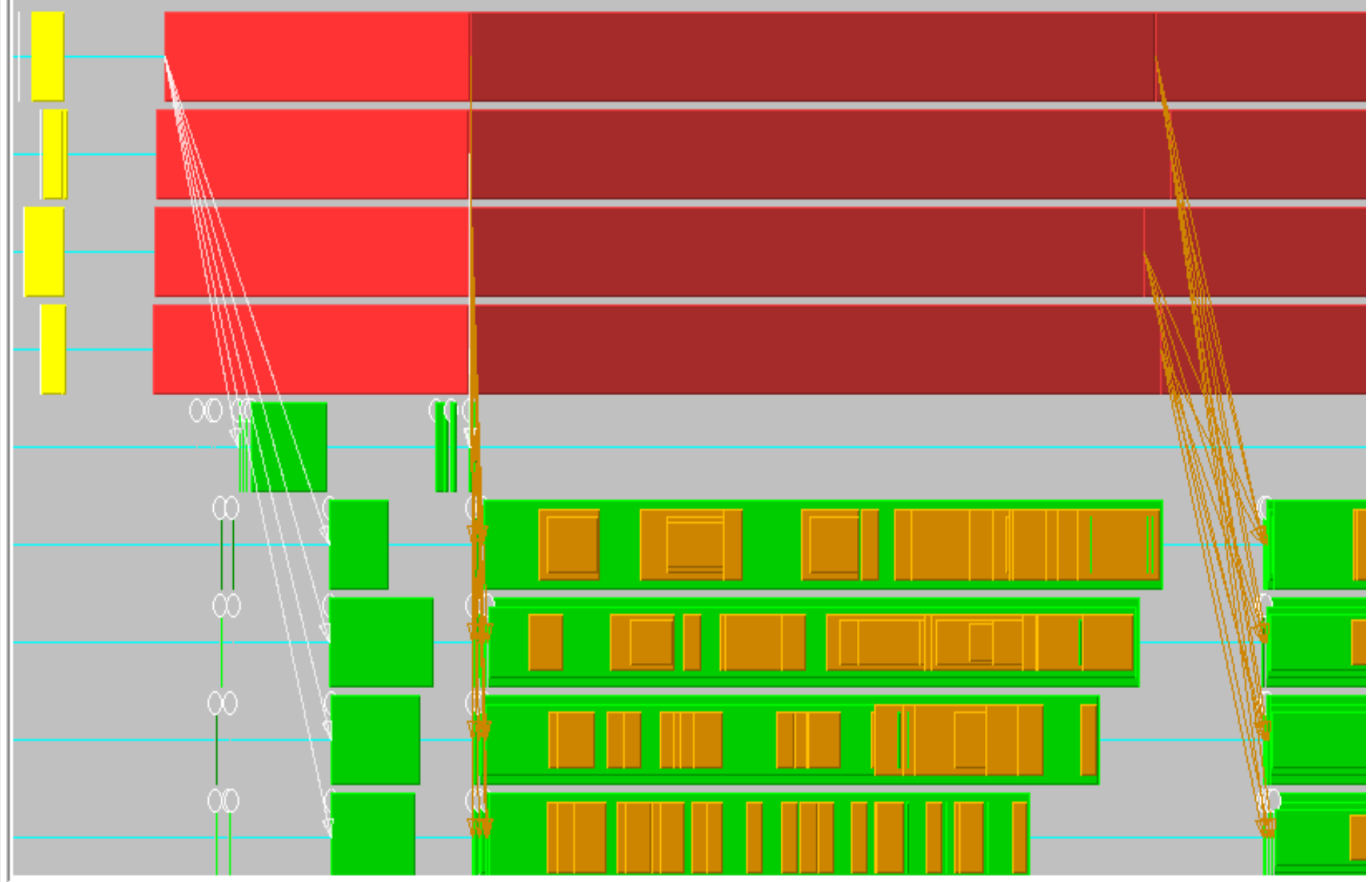


# ... trace with non-overlapping states and arrows

Flowviz

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

@ Servers  
@ Spreadlin





# Observations for traces with arrows

---

- Arrows are used for two purposes
  - Programmer can see which activity in the program triggers which activity on the server
    - gives more insight for program tuning
  - PVFS developer can make a detailed server activity analysis
    - gives more insight for server tuning



# PART IV

---

- Evaluation of File Distribution
- Evaluation of Optimization Levels
  - independent vs. collective calls
  - contiguous vs. non-contiguous data regions



# Evaluation of File Distribution

---

## Configuration:

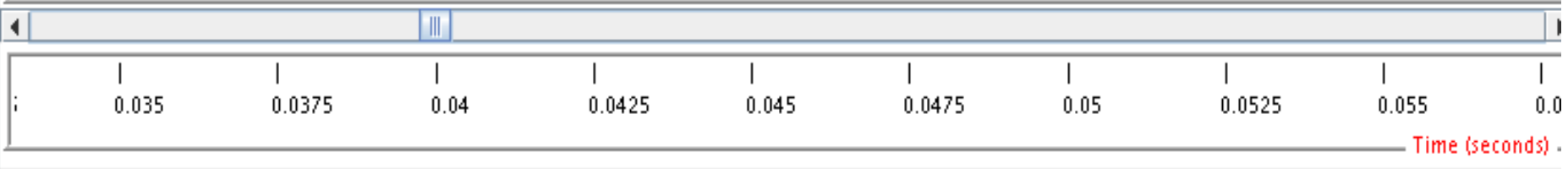
- four servers / one client
- client writes 10x50KB or 1x500KB
- striping has default value 64KB

# 1 client writes 10x50KB to 4 servers with 64KB striping

Fileviz

- 10002
- 0
- 100
- 104
- 101
- 102
- 103

@ Servers  
@ Spreadlin



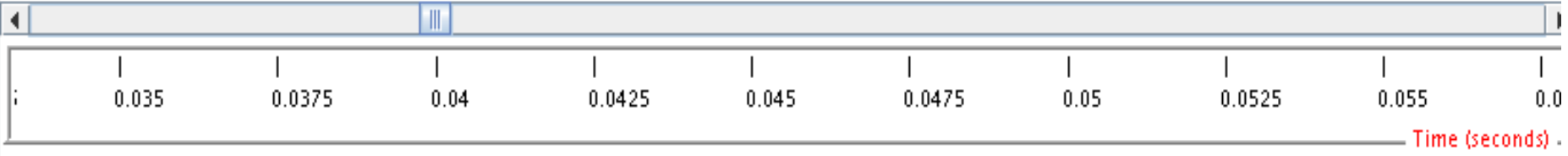
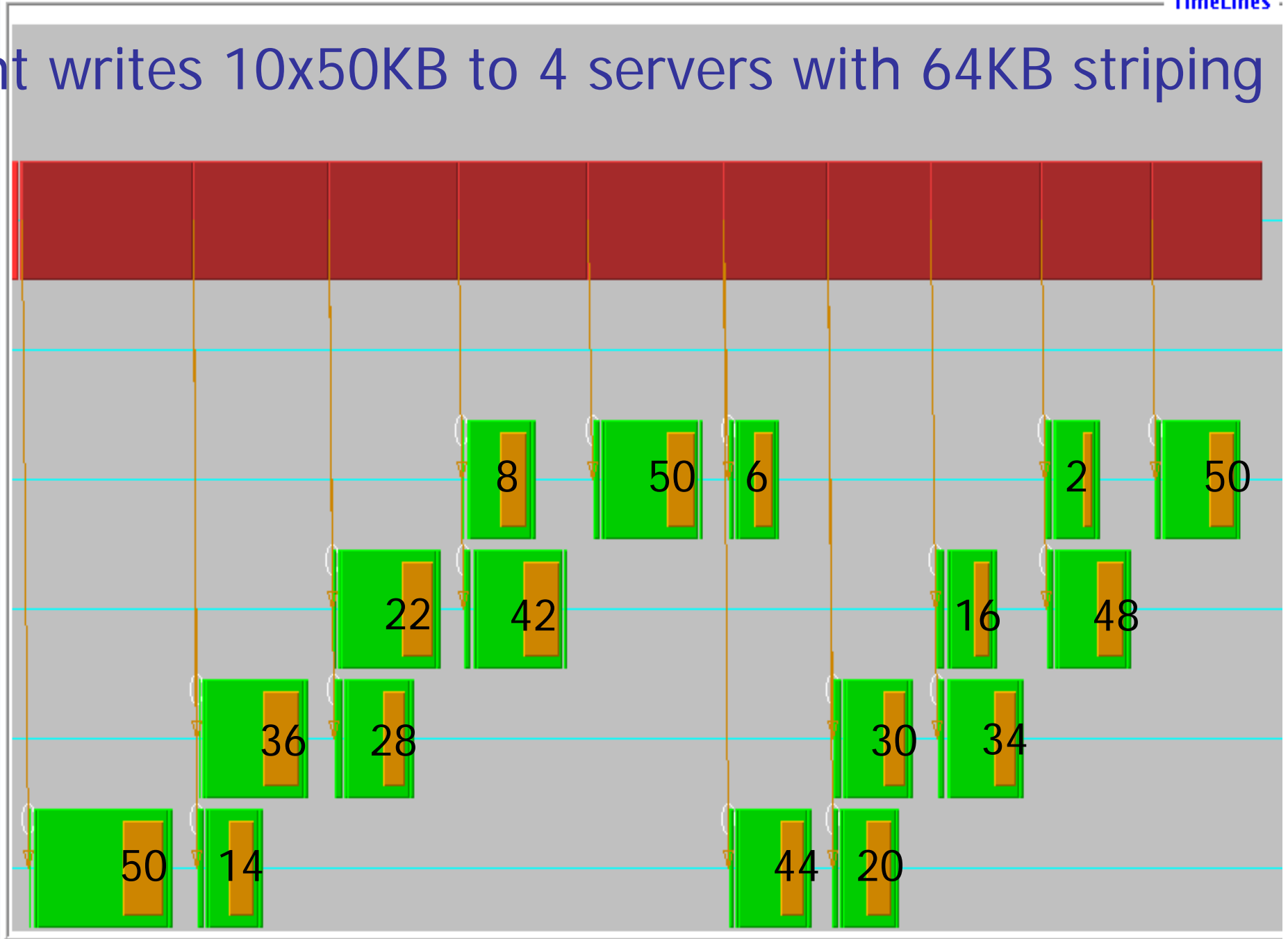
# 1 client writes 10x50KB to 4 servers with 64KB striping

Flowviz interface showing a tree view of nodes:

- 0
- 100
- 104
- 101
- 102
- 103

Bottom panel labels:

- @ Servers
- @ Spreadlin

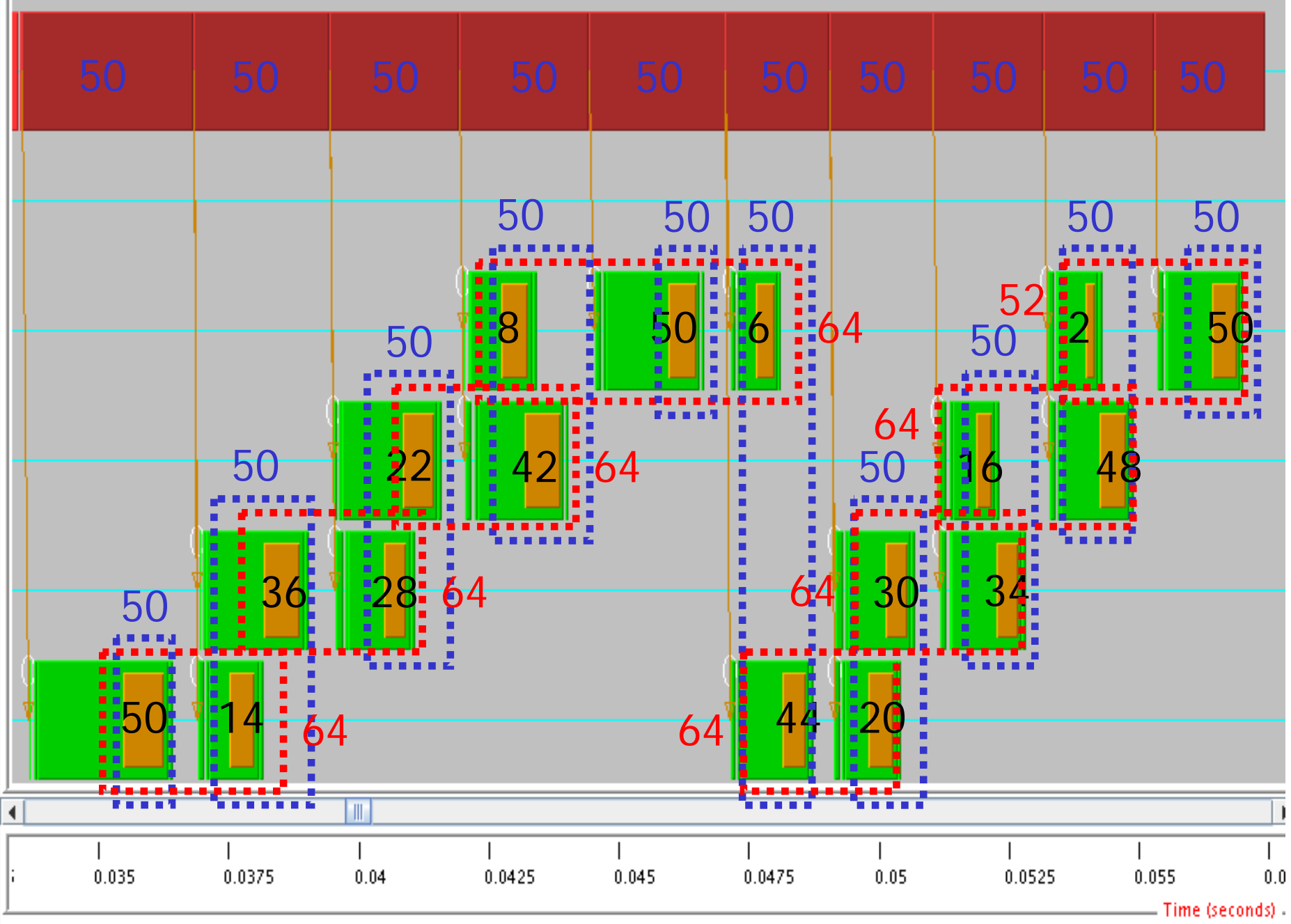


# 1 client writes 10x50KB to 4 servers with 64KB striping

Flowviz interface showing a tree structure of nodes:

- 0
- 100
- 104 (116)
- 101 (128)
- 102 (128)
- 103 (128)

Bottom panel: @ Servers, @ Spreadlin

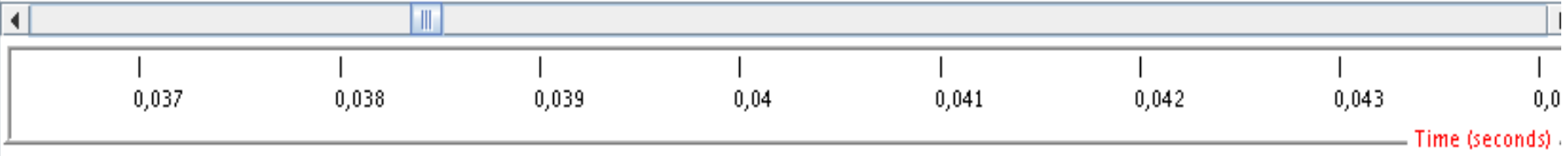
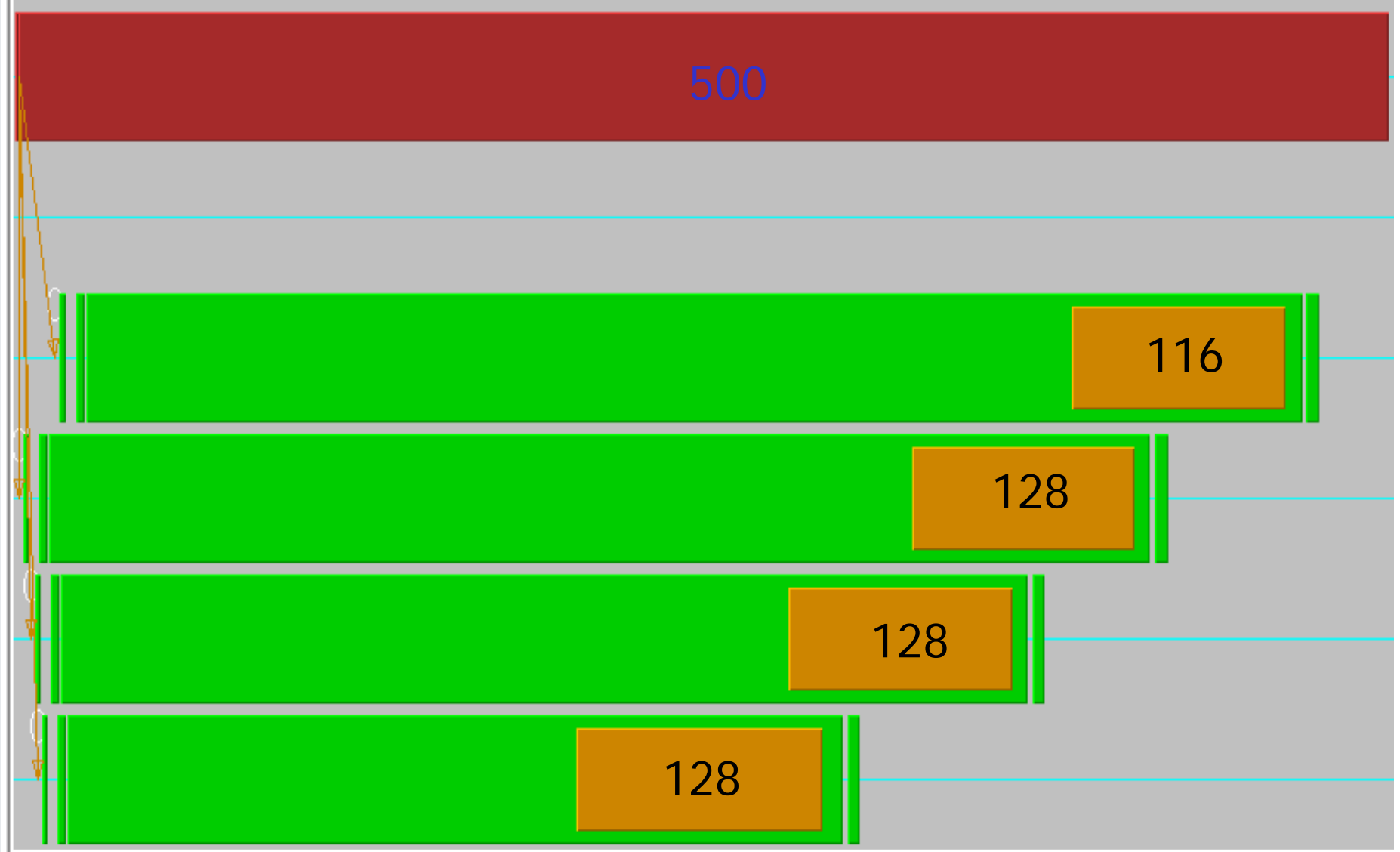


# 1 client writes 1x500KB to 4 servers with 64KB striping

Flowviz

- SLOC-2
- 0
- 100
- 103
- 102
- 101
- 104

@ Servers  
@ Spreadlin



# Evaluation of Optimization Levels

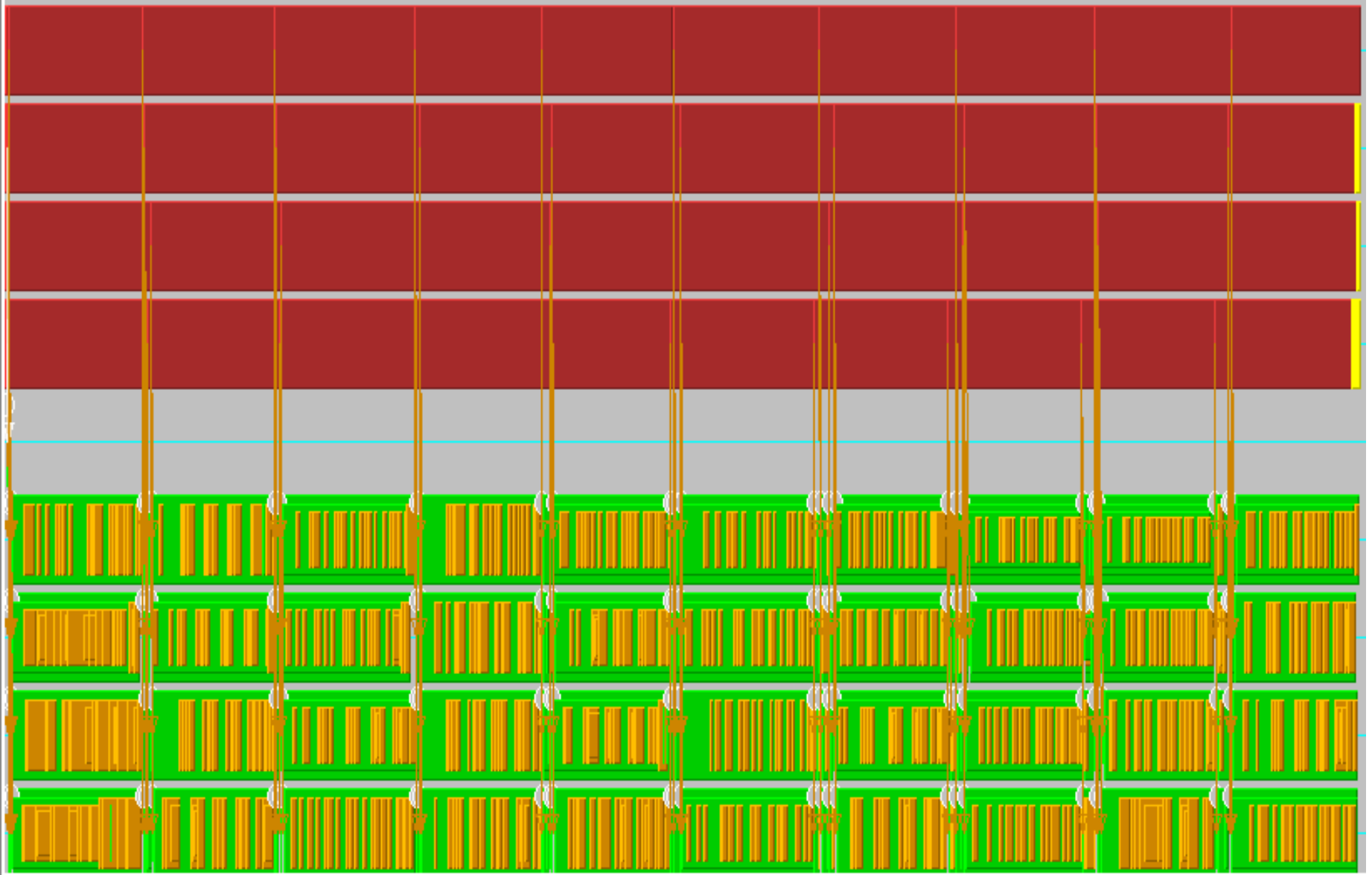


- Configuration: 4 servers, 4 clients
- 2 examples
  - write 50MB (read was similar)
  - write 500KB (read was similar)
- 4 I/O concepts
  - individual vs. collective calls
  - contiguous vs. non-contiguous data regions

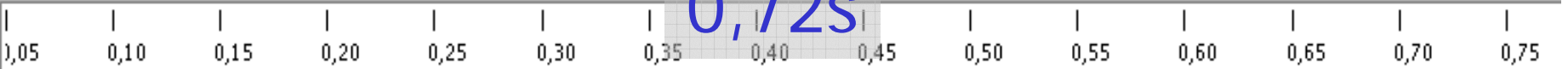
4 servers / 4 clients / ind-ctg / write 4\*50MB

SLOG-2

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

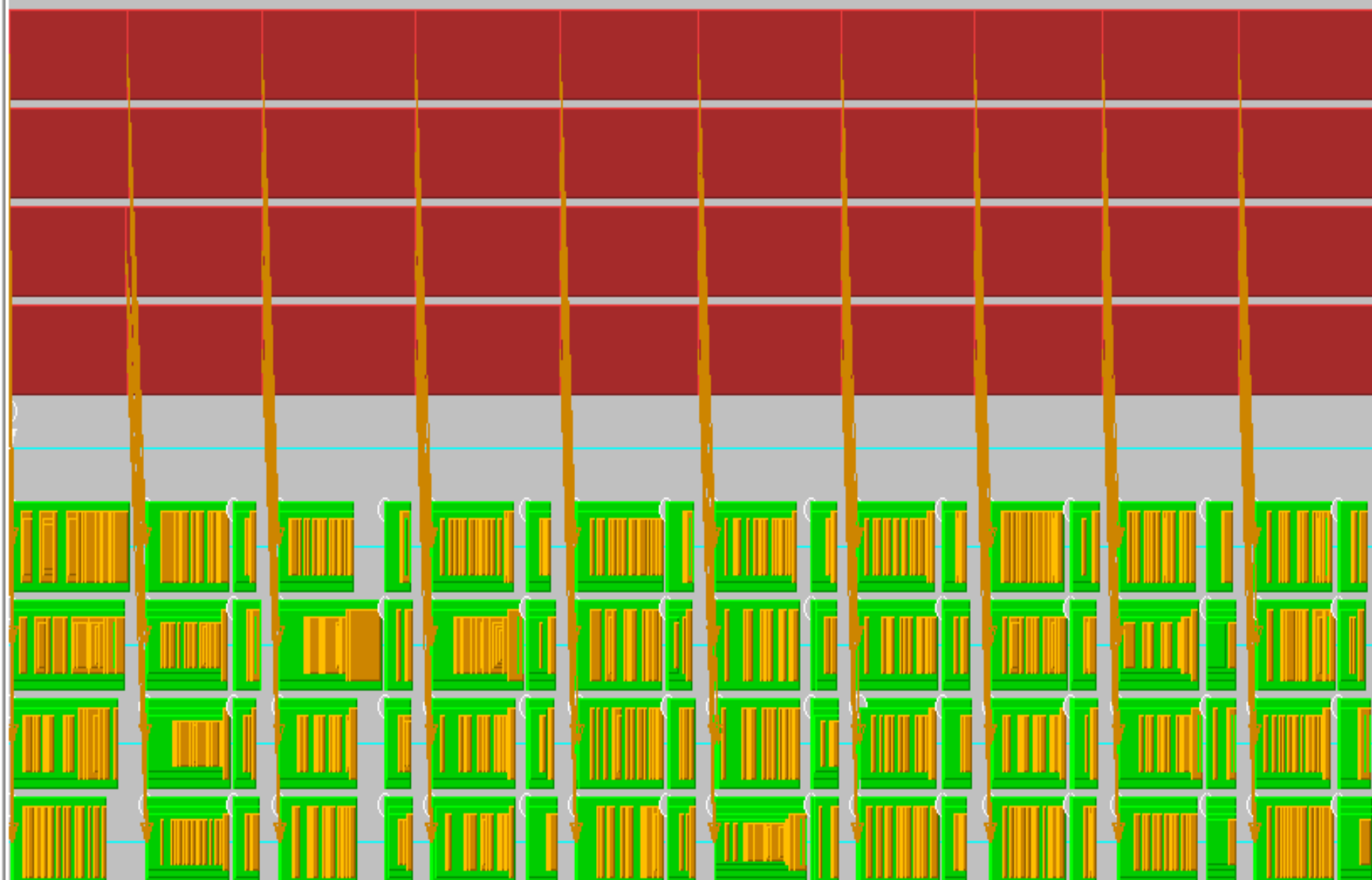


0,72s



Time (seconds)

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

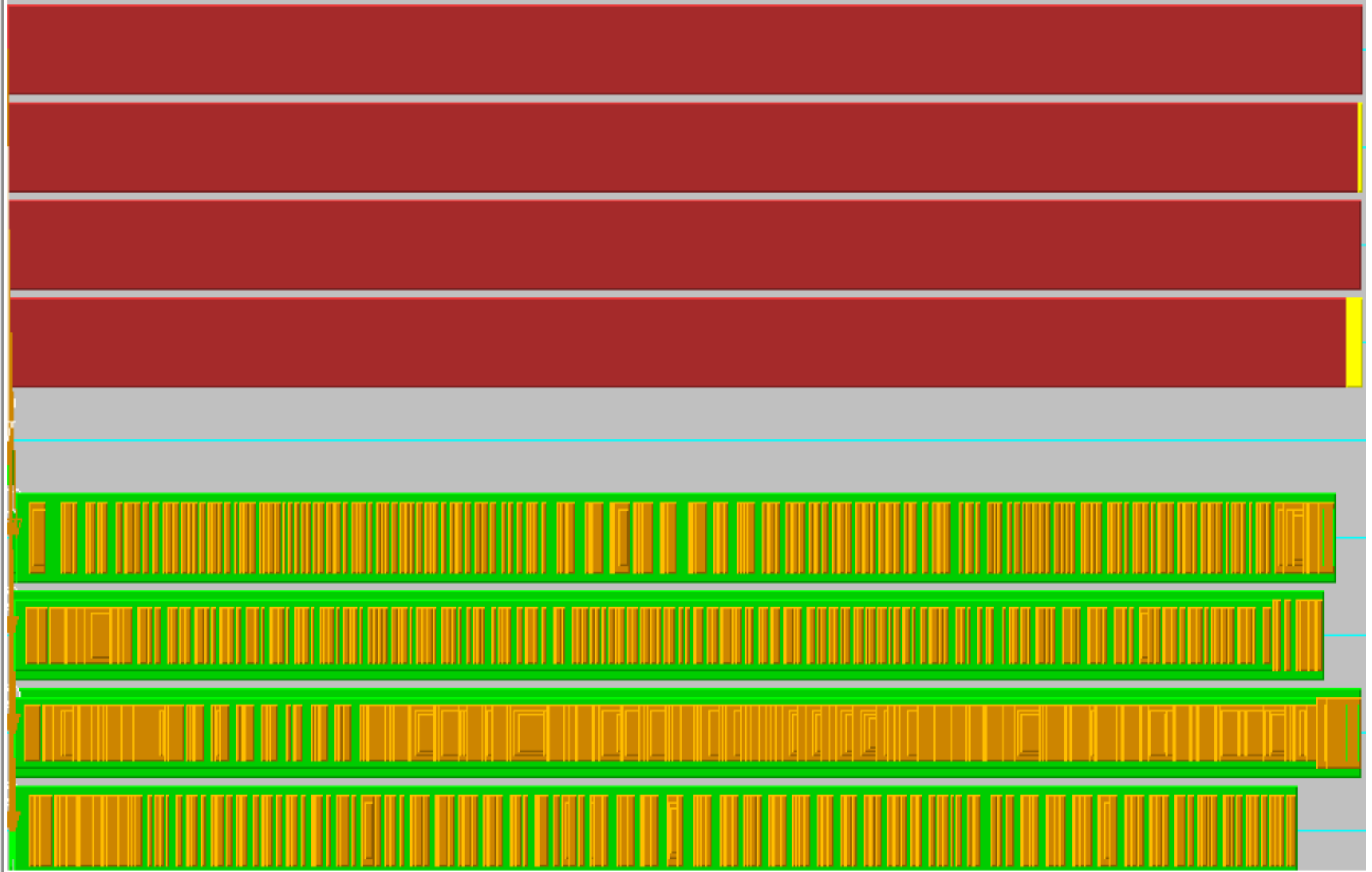
1,00

Time (seconds)

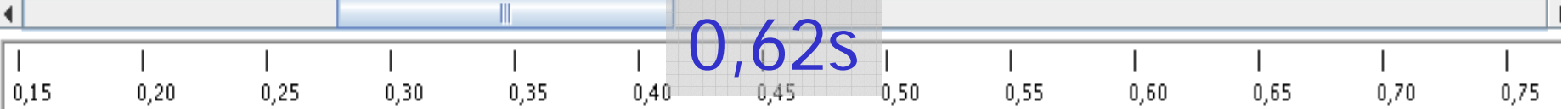
4 servers / 4 clients / ind-nctg / write 4\*50MB

SLOG-2

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



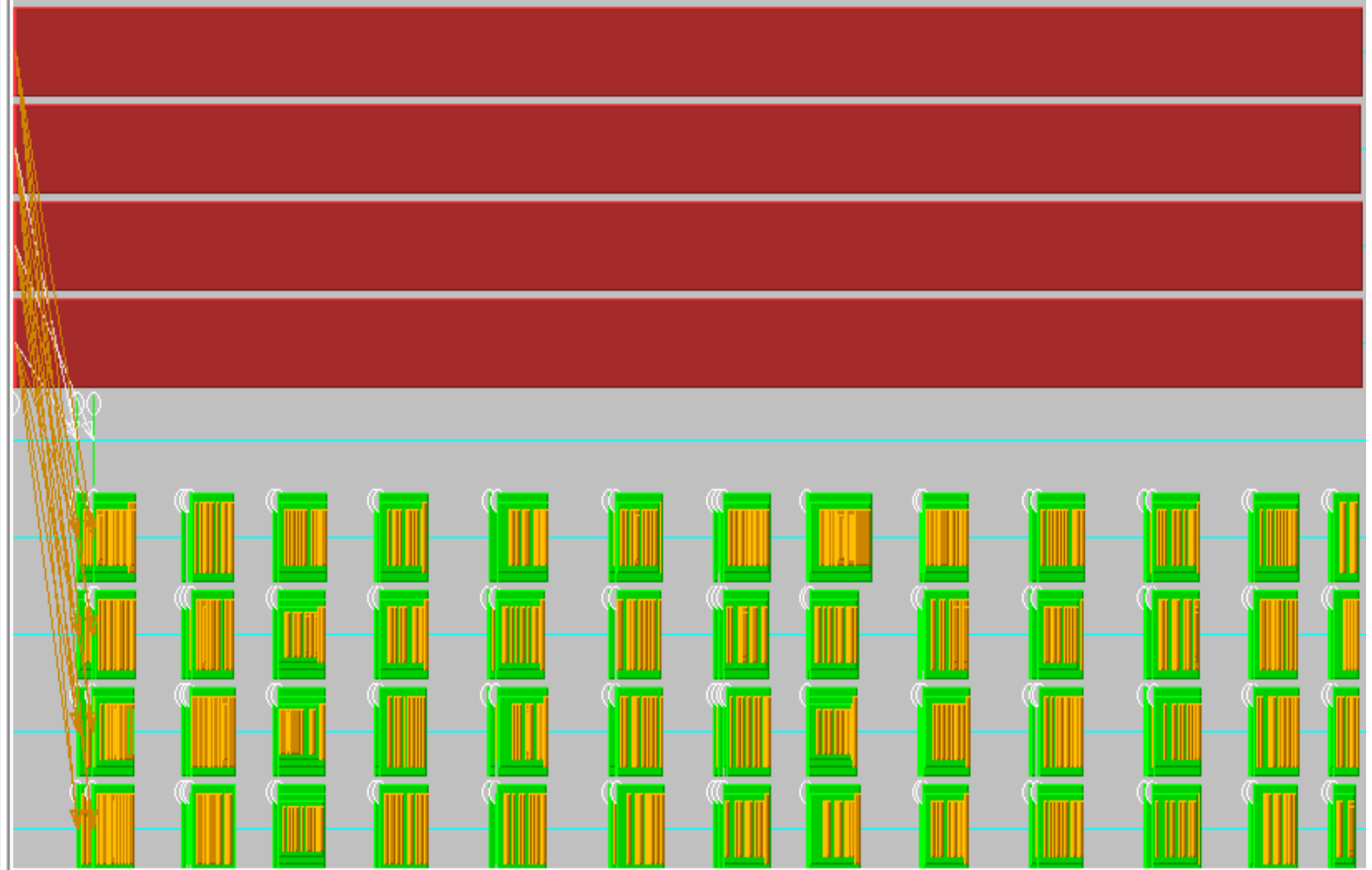
@ Servers  
@ Spreadlin



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

SLOG-2

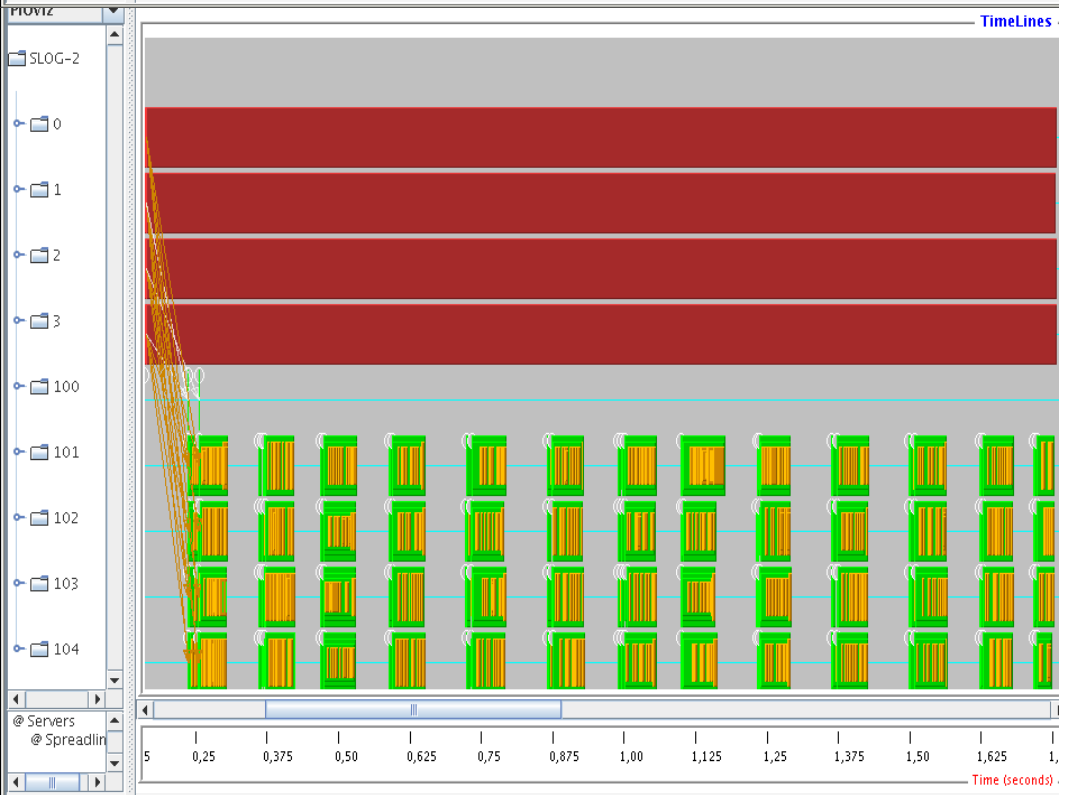
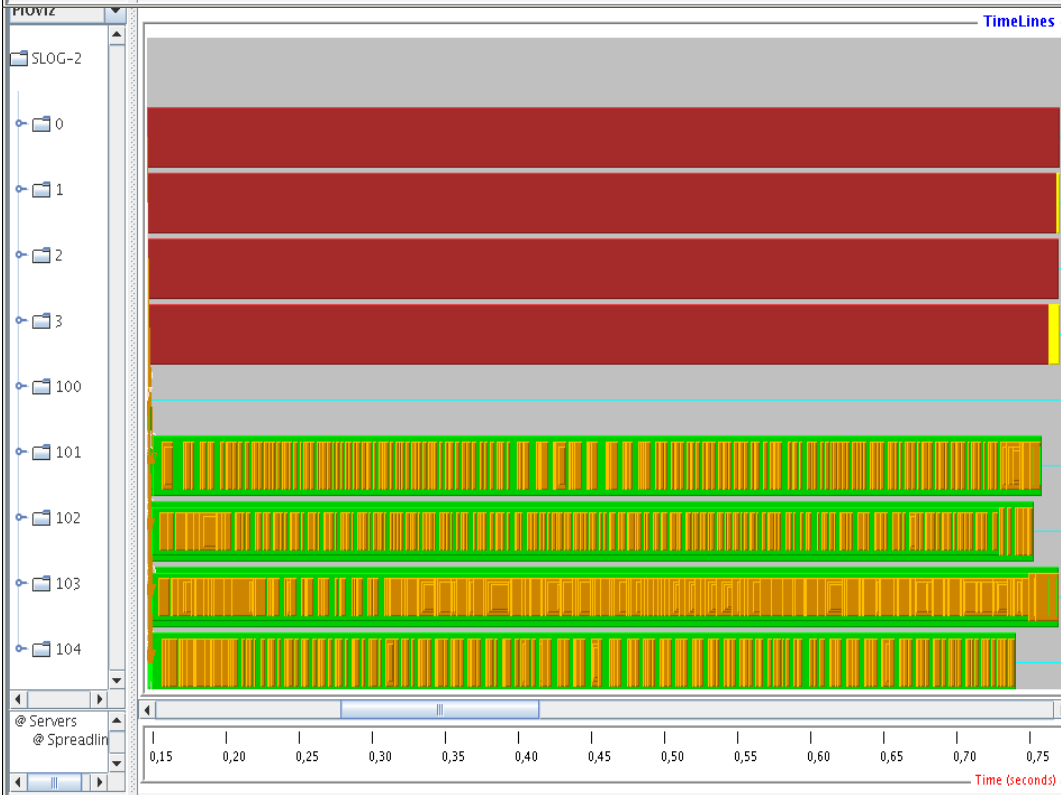
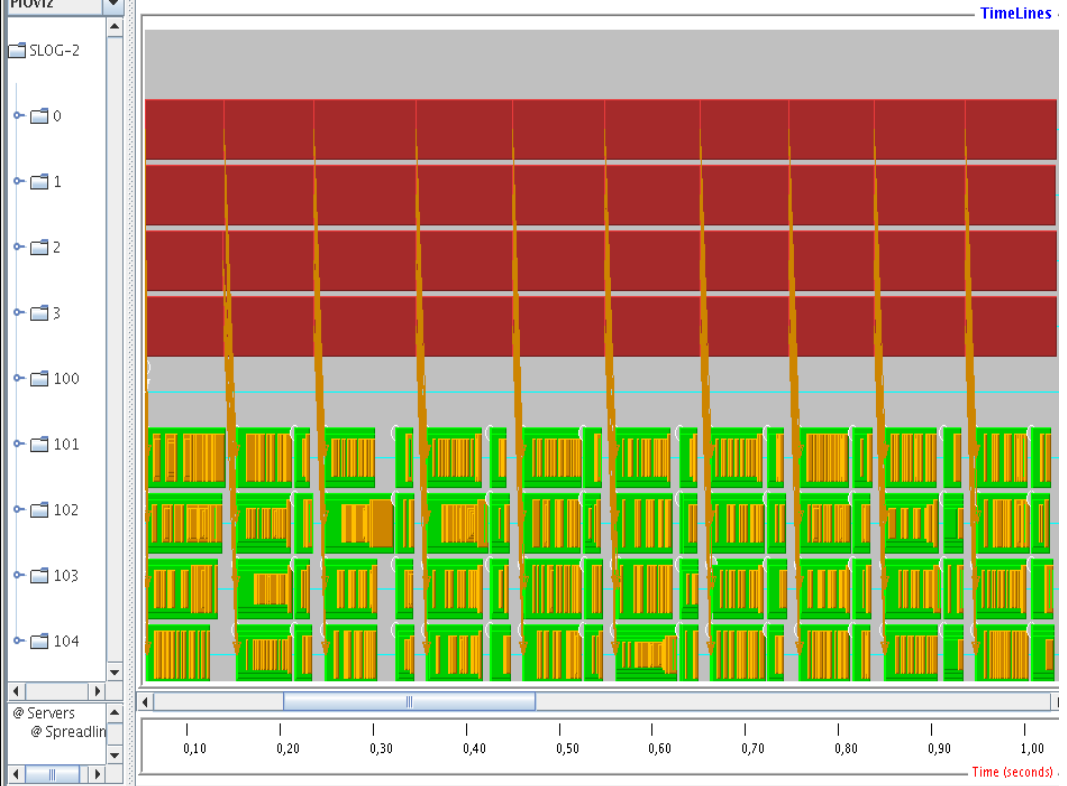
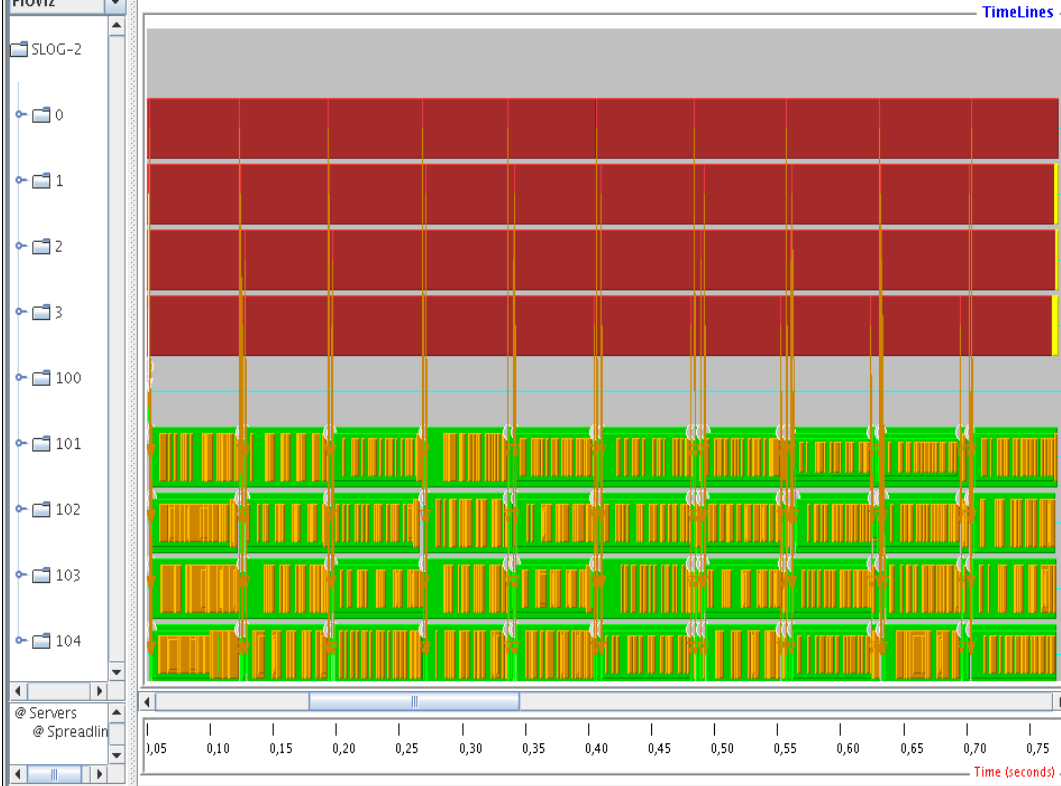
- 0
- 1
- 2
- 3
- 100
- 101
- 102
- 103
- 104



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,

Time (seconds)



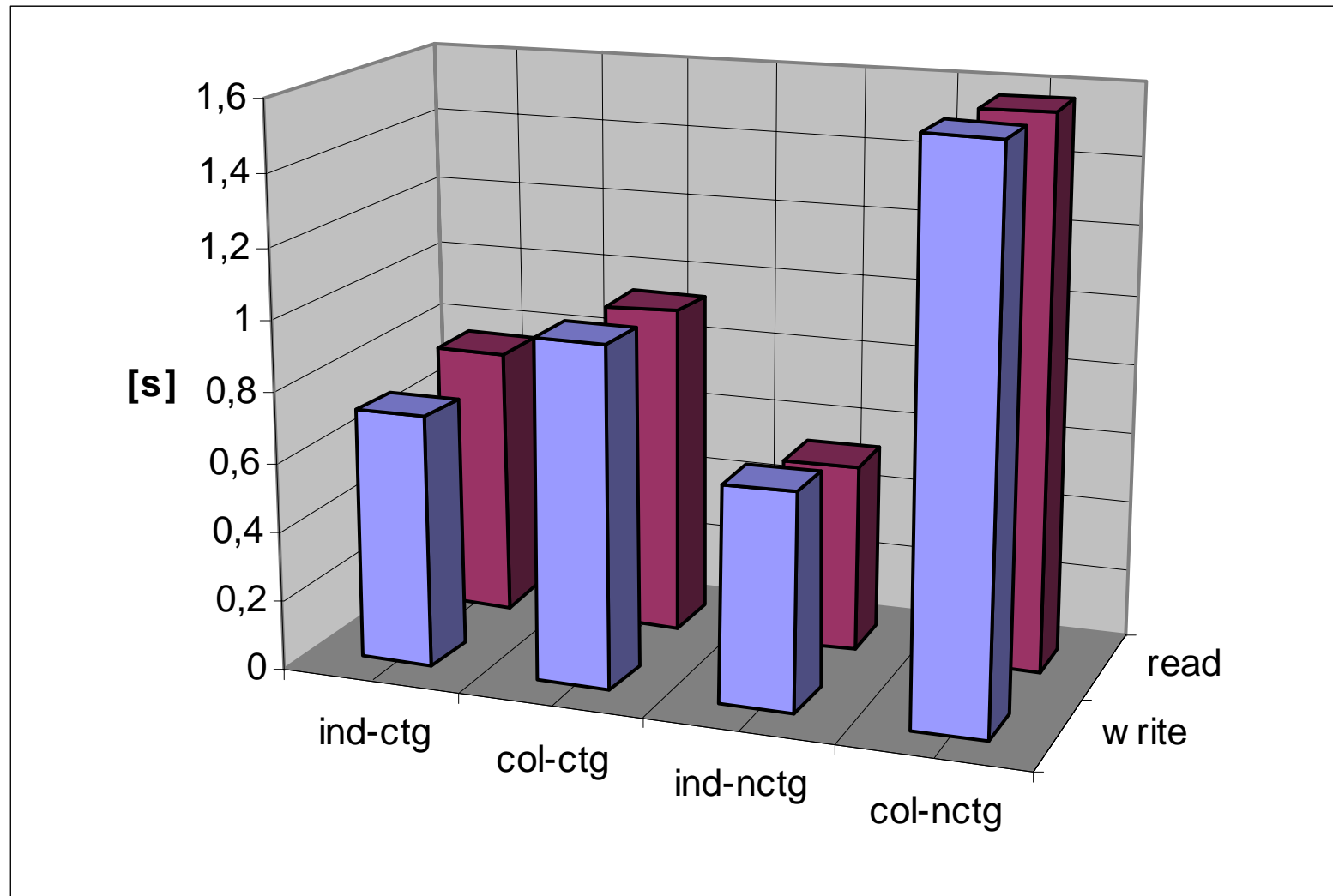


# Observations

---

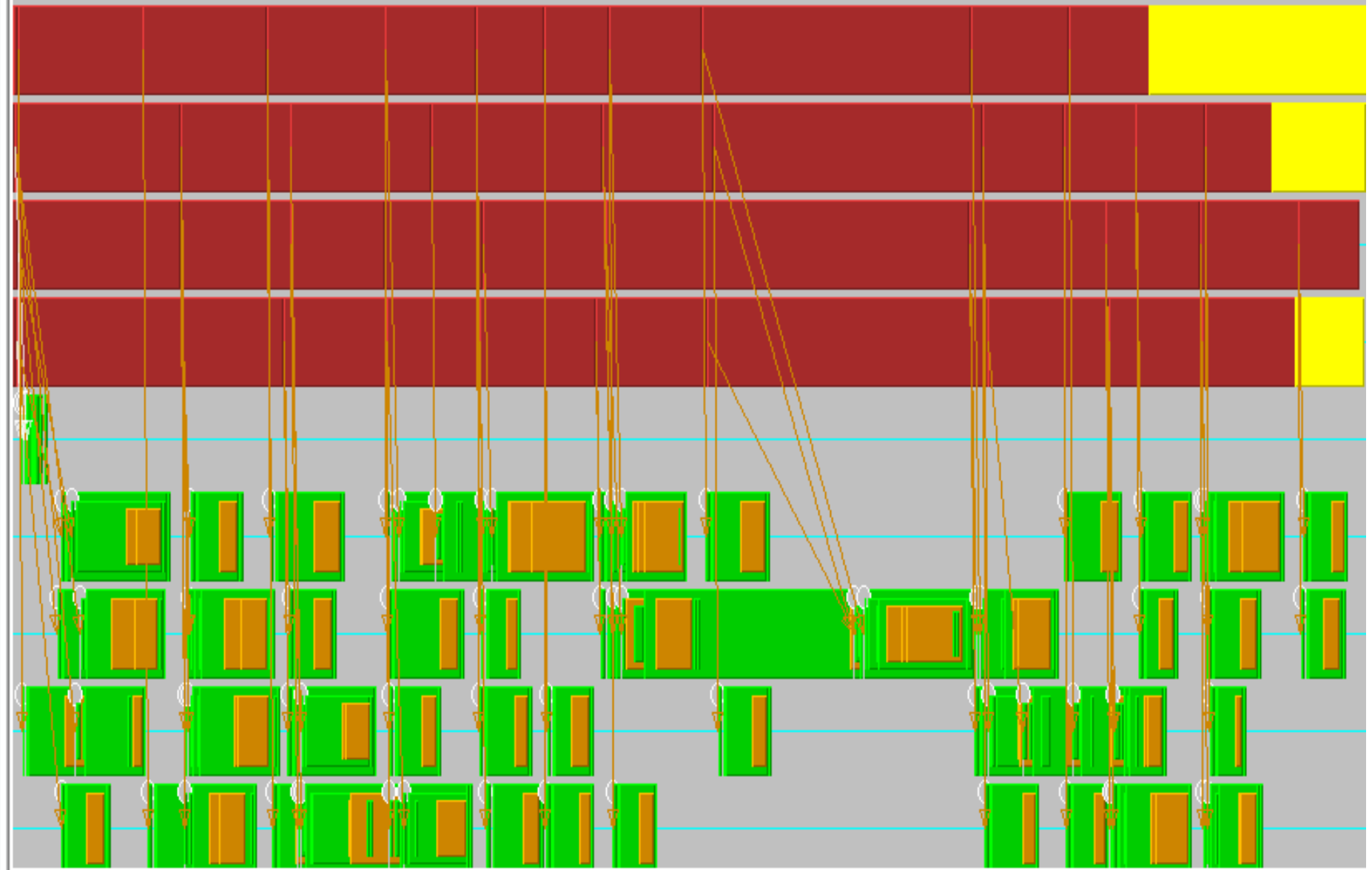
- Contiguous collective calls last longer because time is determined by the slowest participant
- Non-contiguous collective calls last much too long because the communication in the two-phase protocol is not efficient
- Non-contiguous independent calls with data sieving have the best performance

# Summary 4\*50MB



# 4 servers / 4 clients / ind-ctg / write 4\*500KB

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



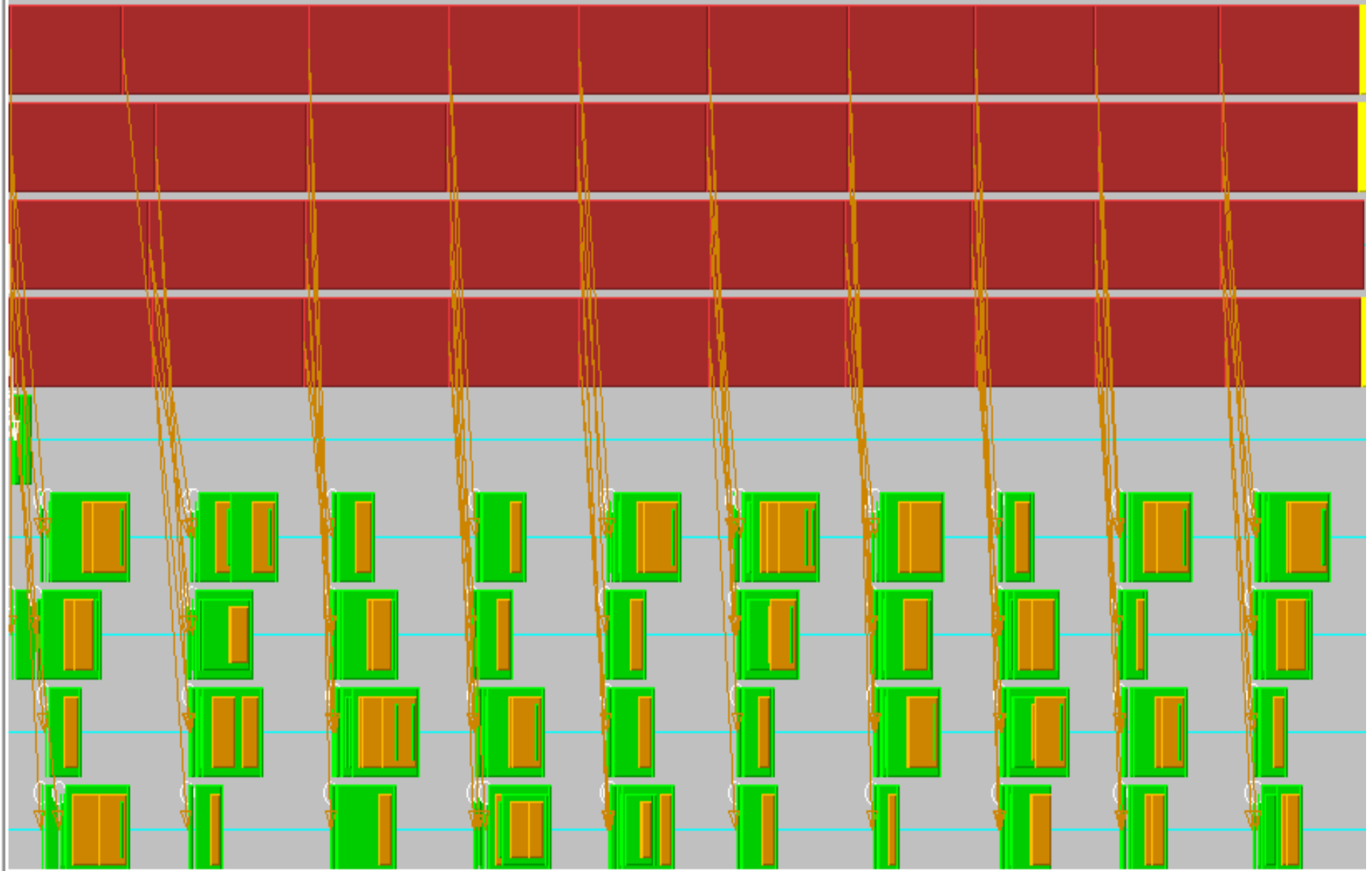
0,039s



4 servers / 4 clients / col-ctg / write 4\*500KB

SLOG-2

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

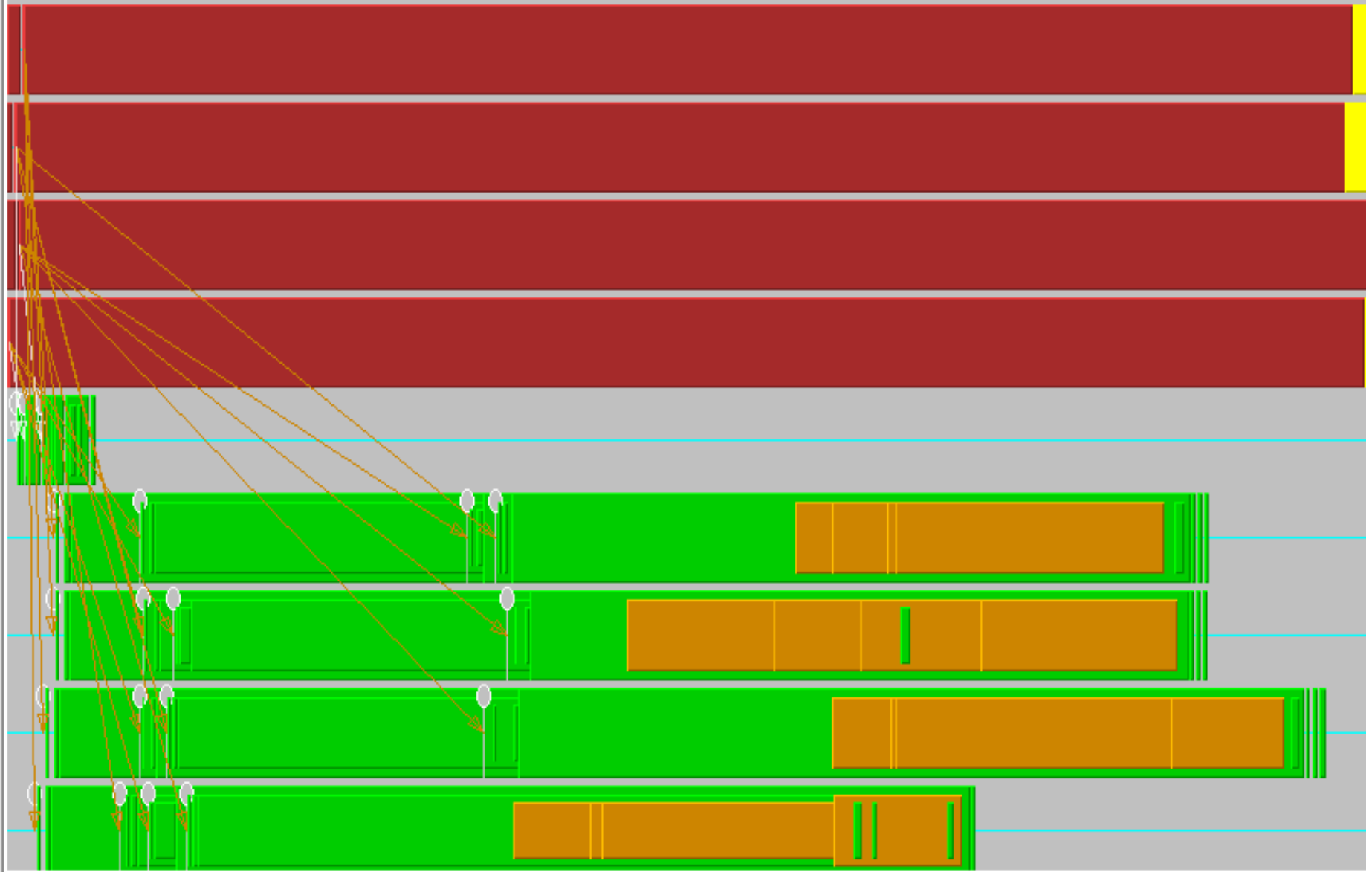


0,052s

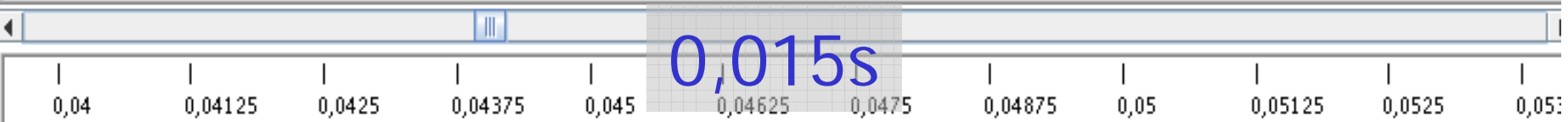
# 4 servers / 4 clients / ind-nctg / write 4\*500KB

SLOG-2

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



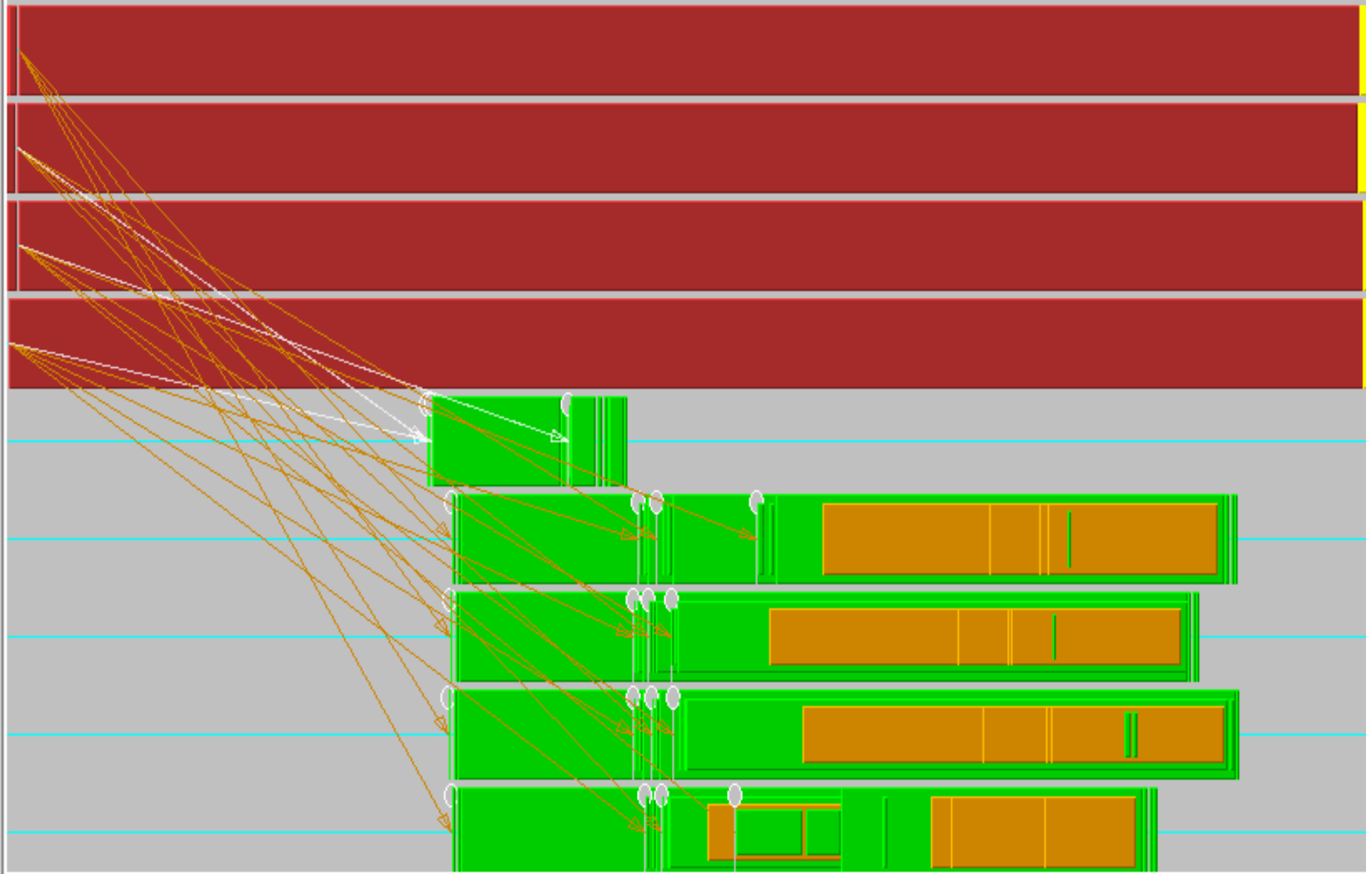
@ Servers  
@ Spreadlin



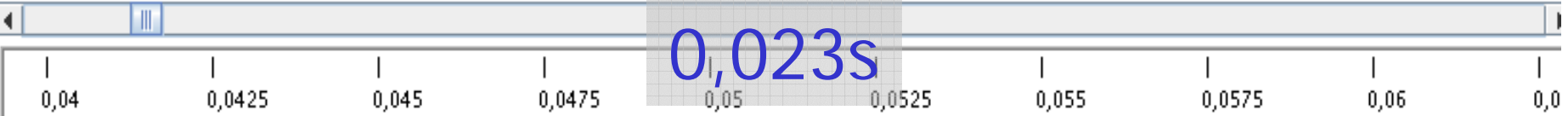
# 4 servers / 4 clients / col-nctg / write 4\*500KB

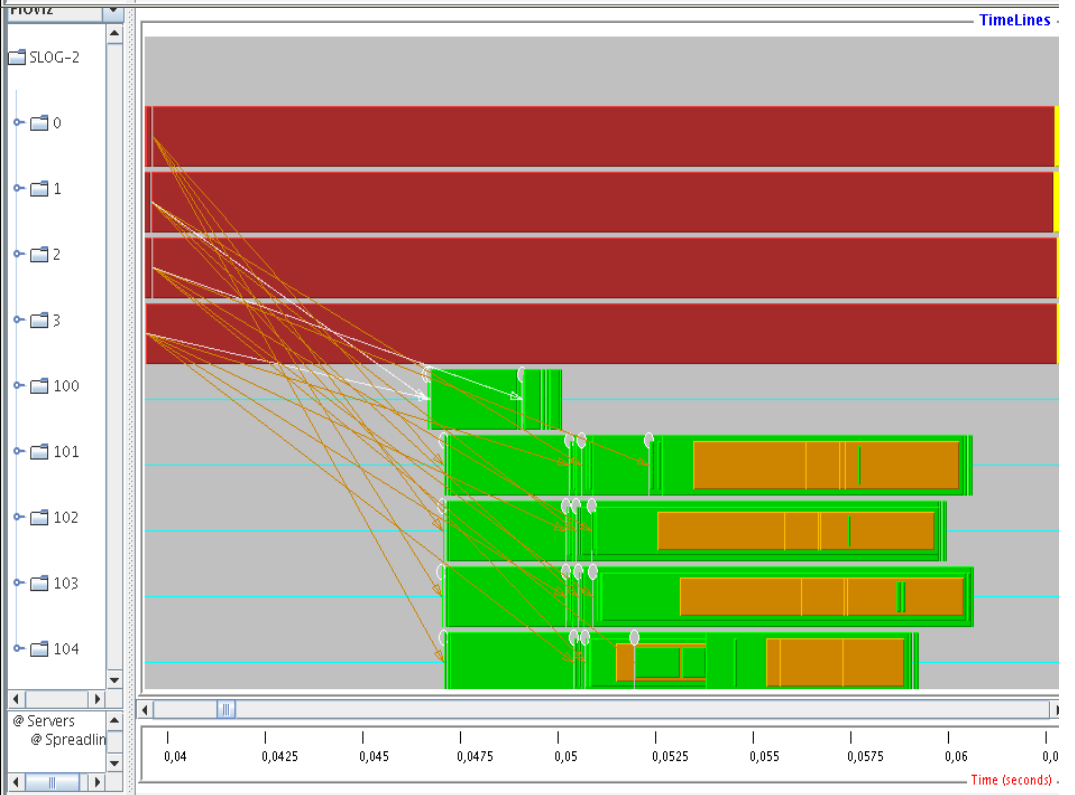
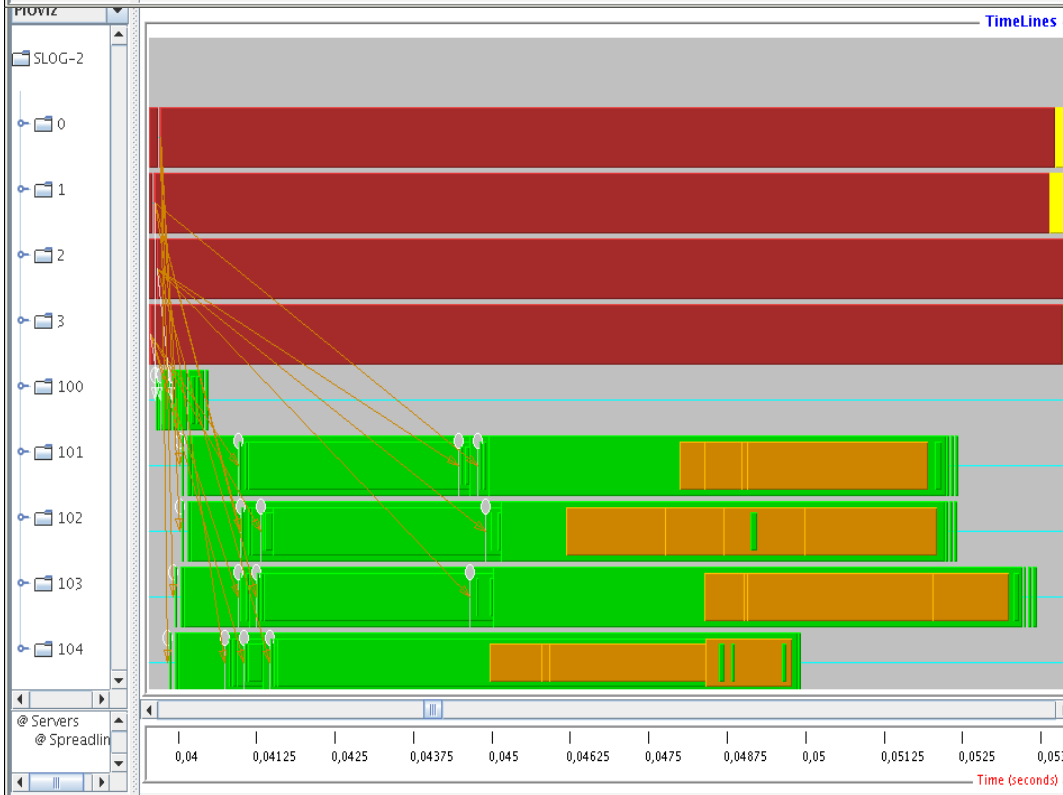
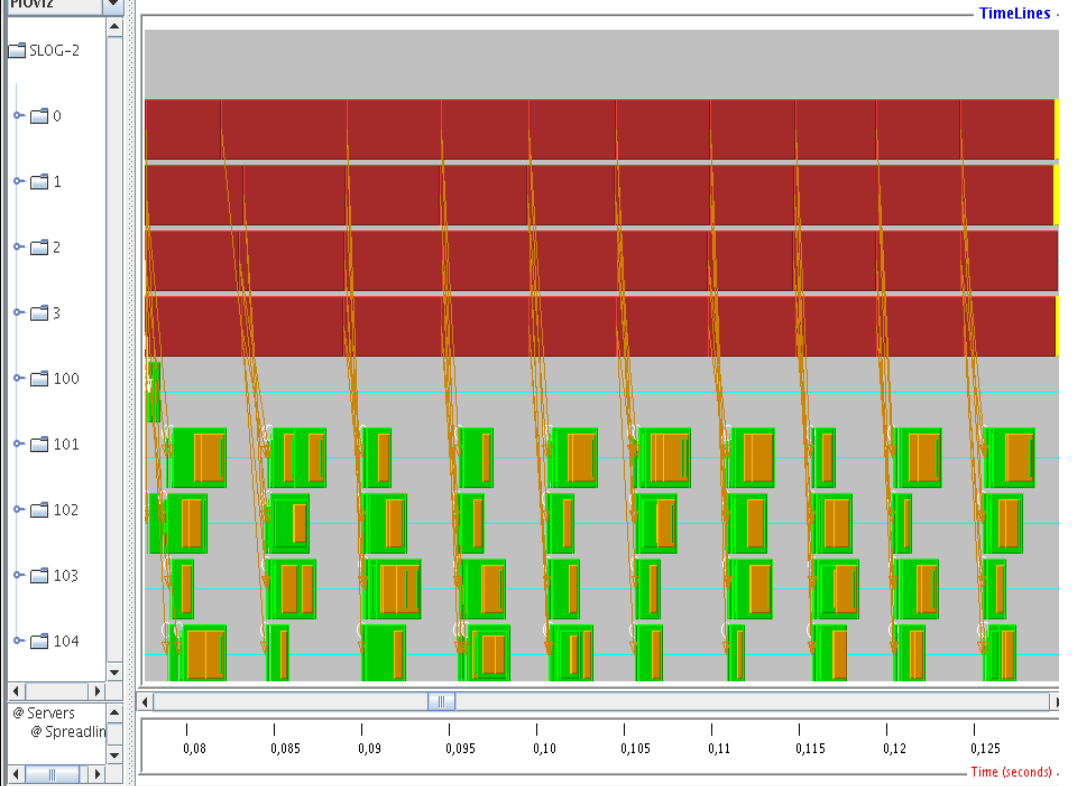
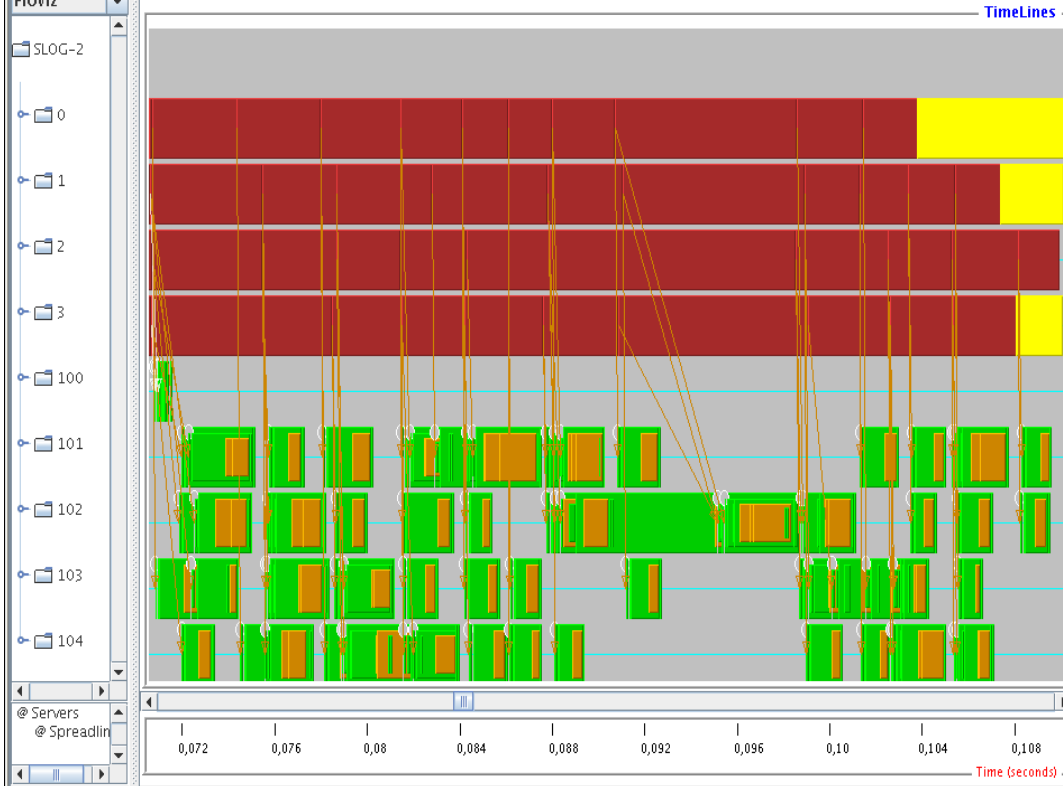
SLOG-2

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



0,023s





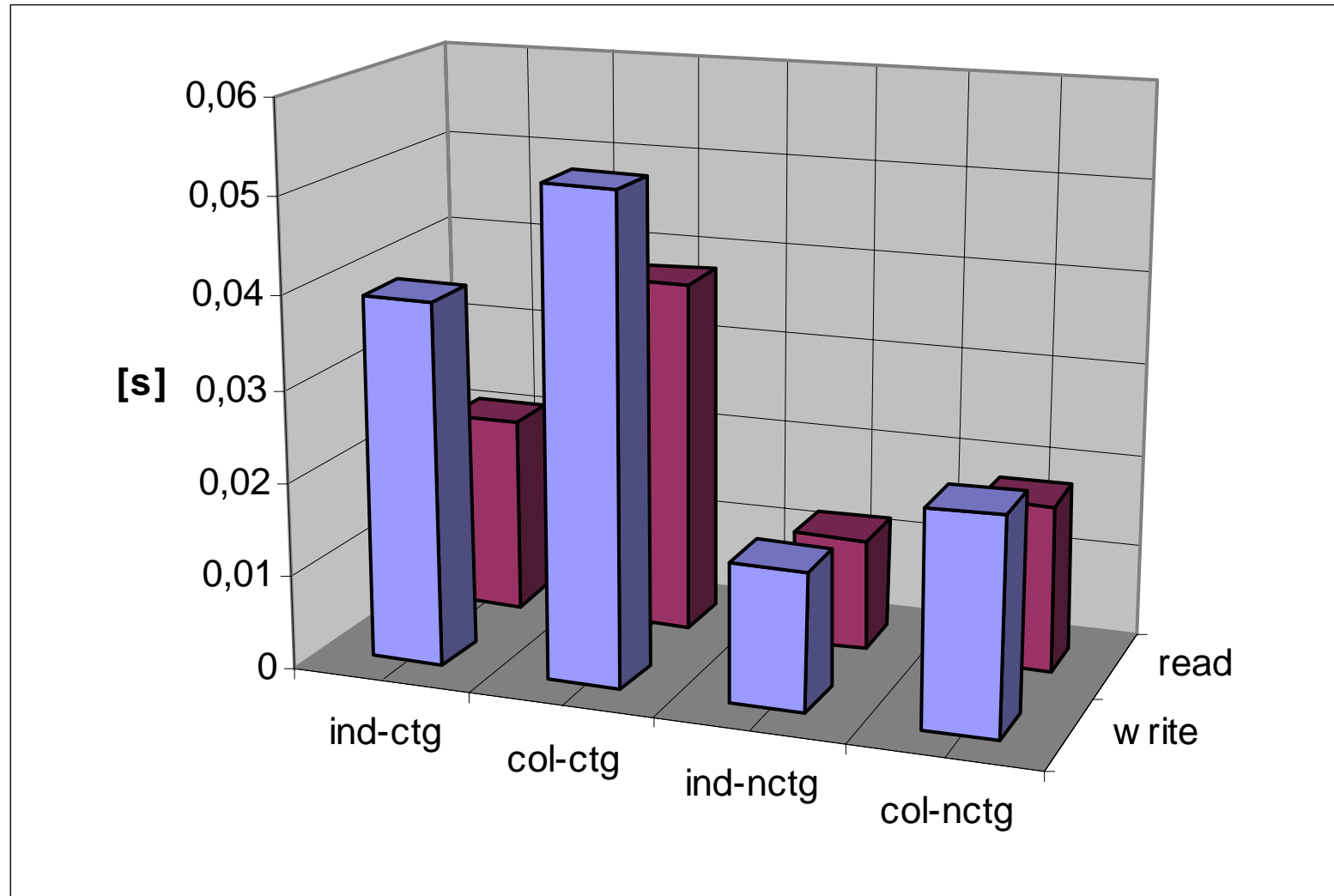


# Observations

---

- Contiguous collective calls last longer because time is determined by the slowest participant
- Non-contiguous collective call is quite efficient; it seems that the inefficiency does not show up with small data amounts
- Non-contiguous independent calls with data sieving have the best performance

# Summary 4\*500KB





# PART V

---

- Conclusions
- Ongoing and Future Work



# 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 positive
  - We detected some problems in PVFS
  - Several of them were implementation errors
- We will now look at real applications
- Package with all components will be released soon in the public domain



# Ongoing Work

---

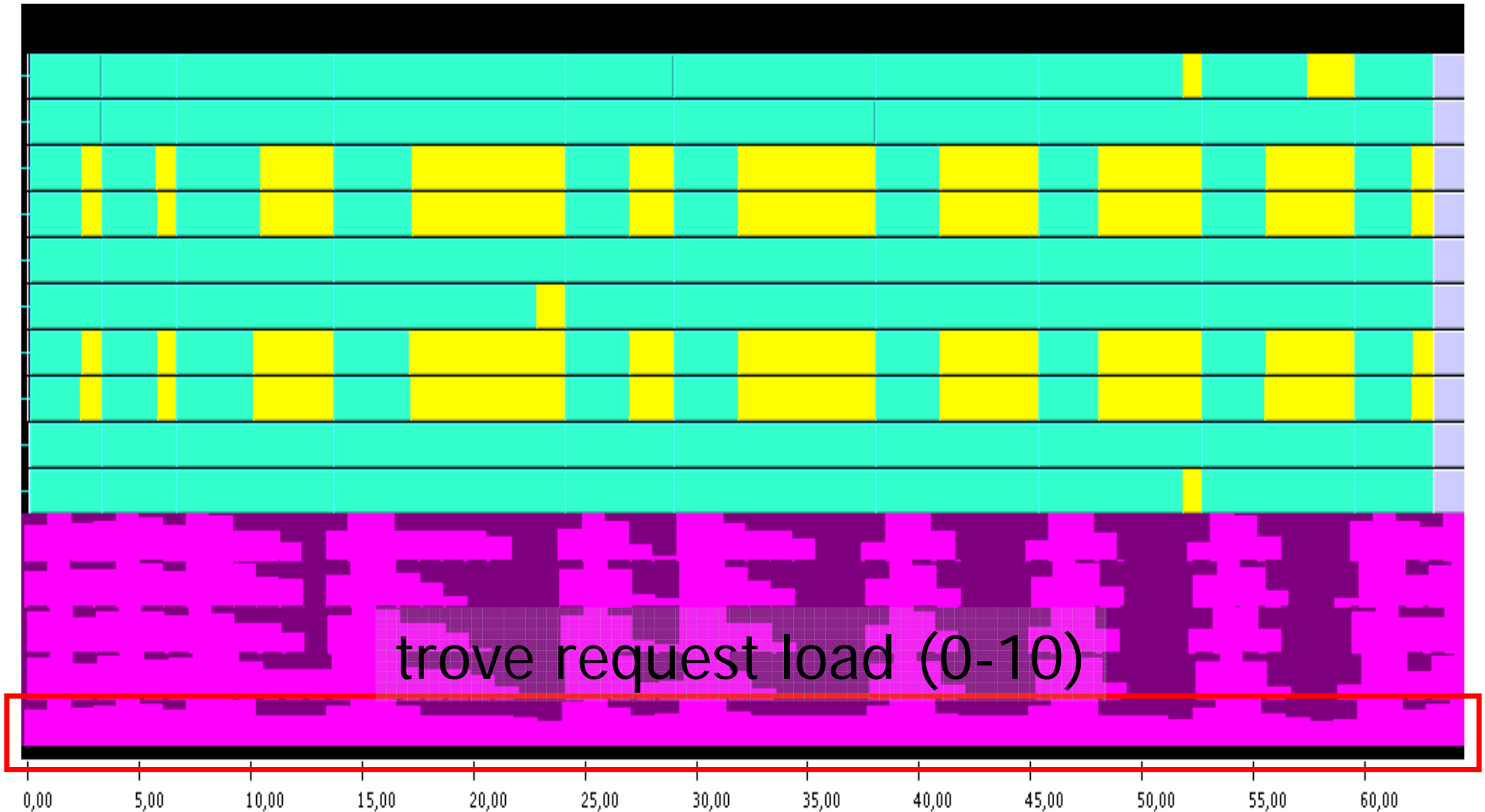
## Measure more data

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

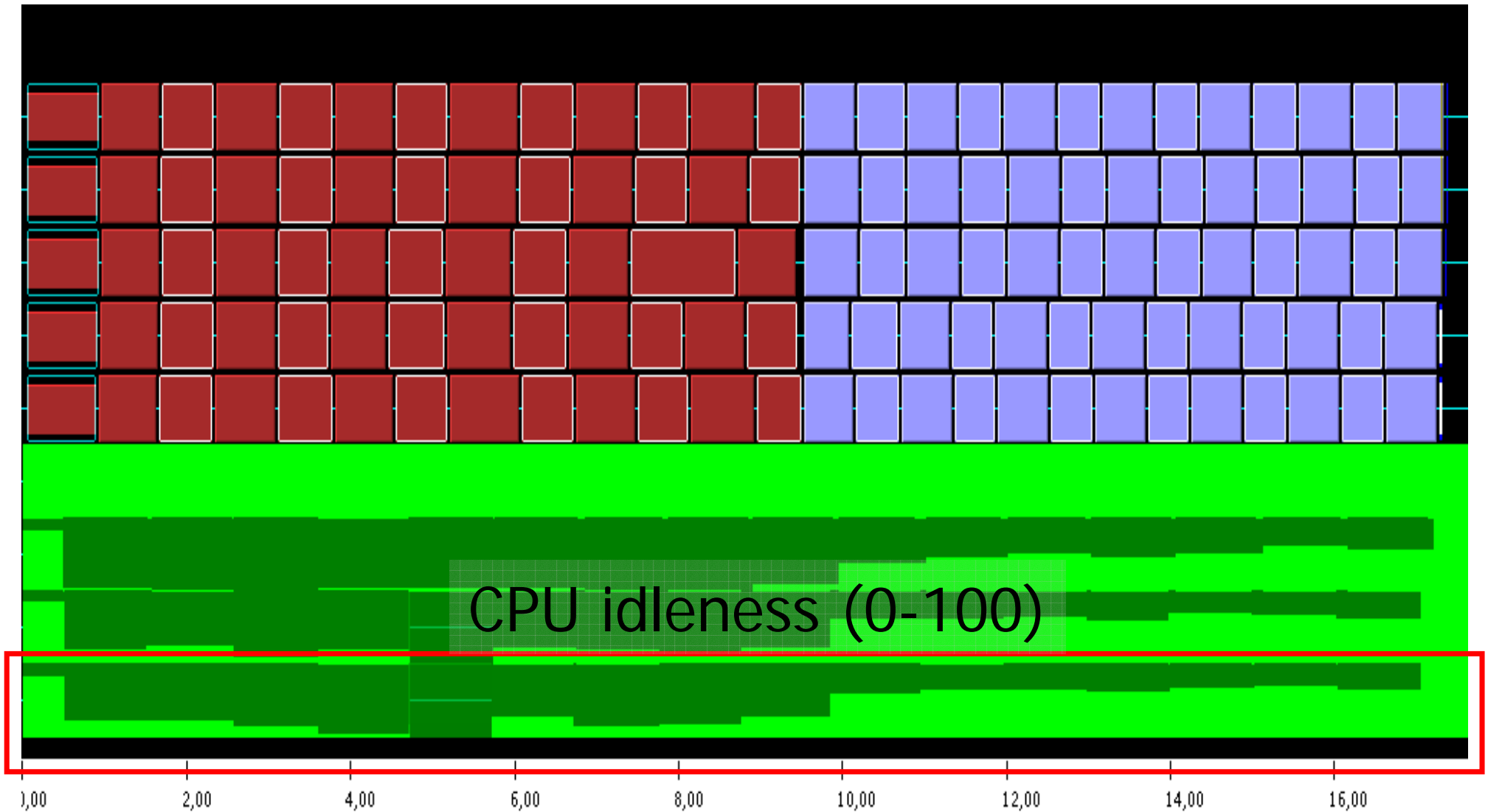
## Use data for on-line decisions

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

# Integrate Performance Counters: Trove-Load



# Integrate Performance Counters: CPU-Load





# 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



# The Team

---

Thomas Ludwig

Stephan Krempel – Michael Kuhn

Julian Kunkel – Christian Lohse

Frank Panse – Dulip Withanage

in cooperation with the PVFS-team of  
Rob Ross at Argonne National Laboratory