INSPECTION OF I/O OPERATIONS FROM SYSTEM CALL TRACES USING DIRECTLY-FOLLOWS-GRAPH

NOV 18, 2024 I PROTOOLS SC-W 24, ATLANTA | **A. SANKARAN**[1], I.ZHUKOV[1], W.FRINGS[1], P.BIENTINESI[2]

Mitglied der Helmholtz-Gemeinschaft

[1] Jülich Supercomputing Center, Germany
[2] Umea Universität, Sweden

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

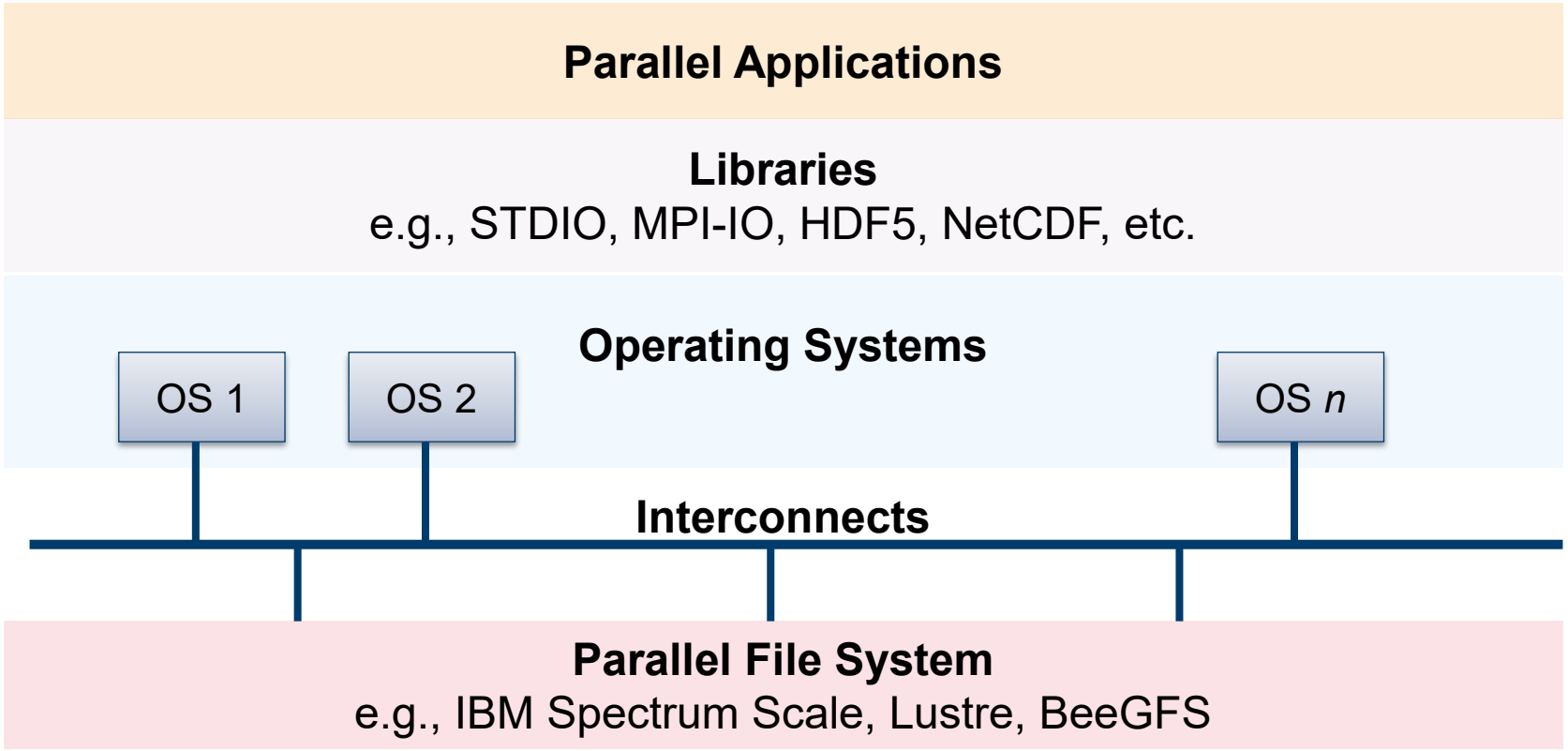
The Objective

Develop a methodology to compare the I/O behavior of multiple user programs resulting from different configuration options.

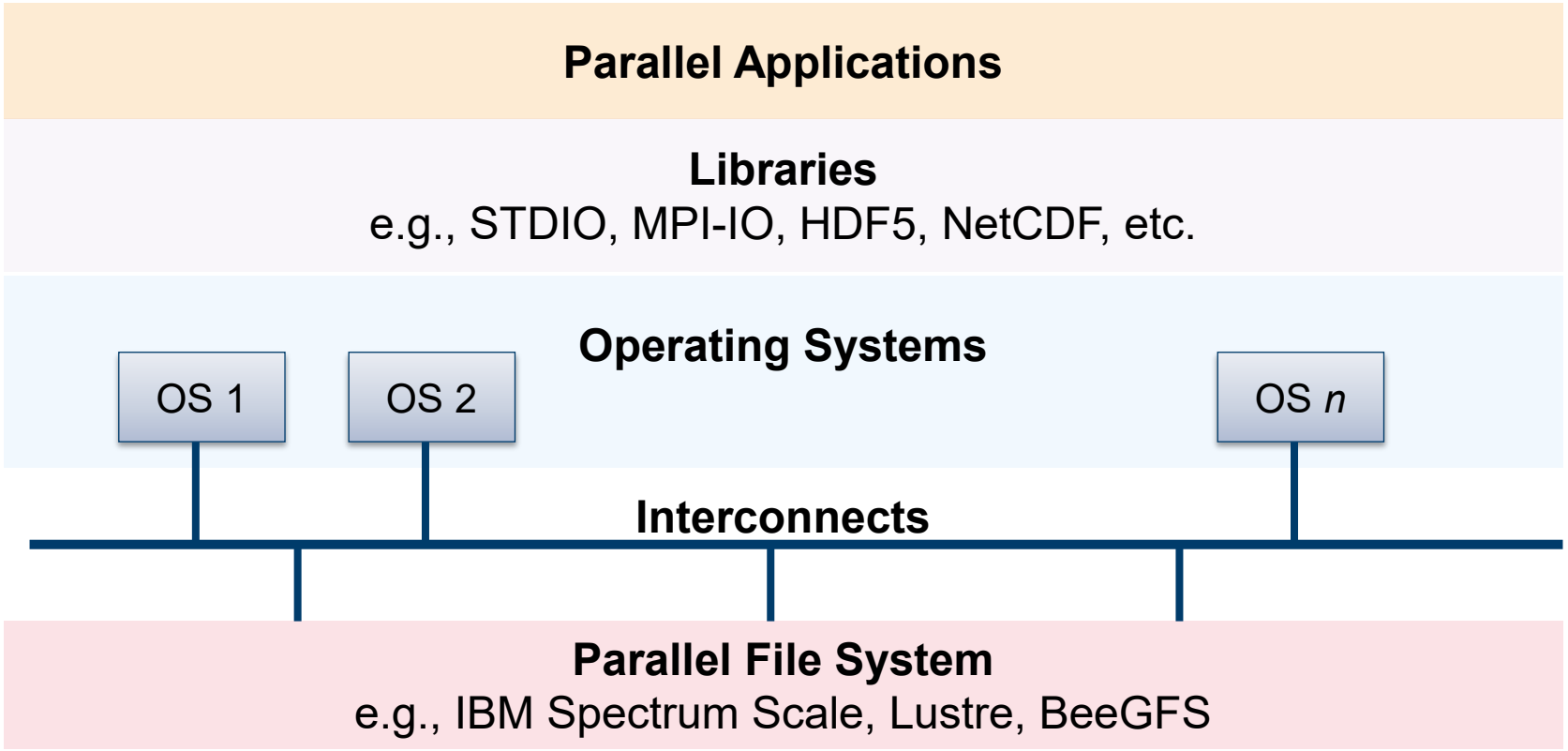
The methodology should work:

- For **arbitrary user programs** without modifying them (e.g., even if the code is within a container, without `MPI_Init`).
- Without necessarily depending on data that only admins can provide (to facilitate **portability between sites**).
- With **capabilities to be integrated with existing frameworks** such as Darshan and Score-P.

The I/O Stack


Parallel Applications
Libraries
e.g., STDIO, MPI-IO, HDF5, NetCDF, etc.
Operating Systems
OS 1
OS 2
OS n
Interconnects
Parallel File System
e.g., IBM Spectrum Scale, Lustre, BeeGFS

The I/O Stack

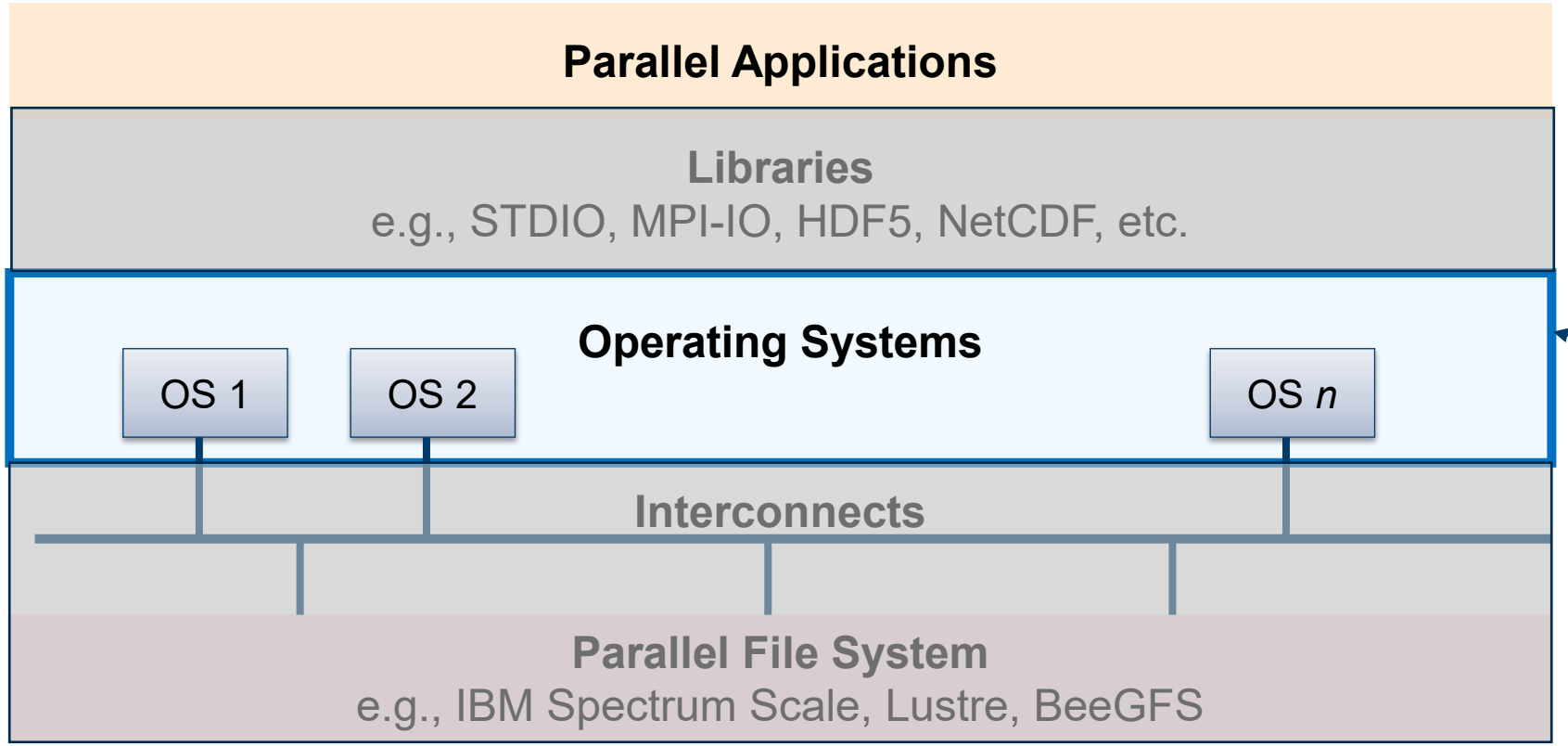


Requires configuration

to

maximize **throughput**.

The Data for Performance Analysis



Tracing System Calls with STrace

Basic Usage:

```
strace [COMMAND]
```

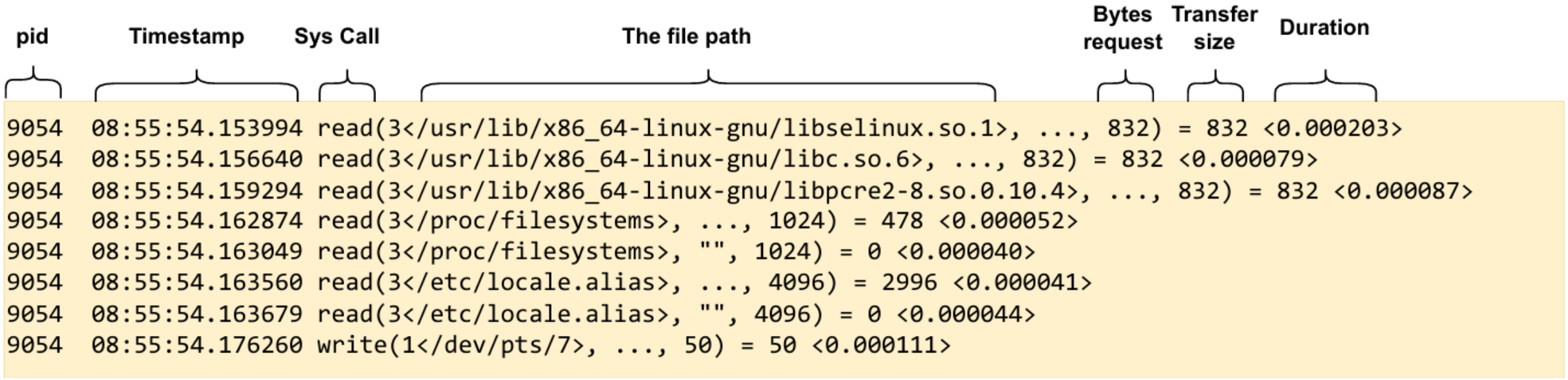
Example:

```
$ strace ls
execve("/usr/bin/ls", ["ls"], 0x7ffd9ef46ac0 /* 36 vars */) = 0
brk(NULL)                               = 0x56490ef01000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc49b7b440) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f9f2c1d8000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=38711, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 38711, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f9f2c1ce000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
```

Tracing System Calls with STrace

Example:

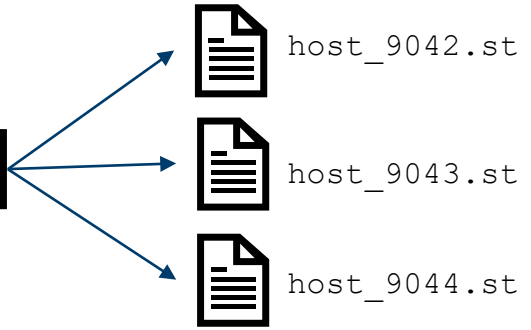
```
strace -f -tt -T -y -e read,write  ls
```



Tracing System Calls with STrace

Example:

```
srun -n 3 strace -f -tt -T -y -e read,write -o $hostname_$$.st ls
```



pid | Timestamp | Sys Call | The file path | Bytes request | Transfer size | Duration

```
9054  08:55:54.153994 read(3</usr/lib/x86_64-linux-gnu/libselinux.so.1>, ..., 832) = 832 <0.000203>
9054  08:55:54.156640 read(3</usr/lib/x86_64-linux-gnu/libc.so.6>, ..., 832) = 832 <0.000079>
9054  08:55:54.159294 read(3</usr/lib/x86_64-linux-gnu/libpcre2-8.so.0.10.4>, ..., 832) = 832 <0.000087>
9054  08:55:54.162874 read(3</proc/filesystems>, ..., 1024) = 478 <0.000052>
9054  08:55:54.163049 read(3</proc/filesystems>, "", 1024) = 0 <0.000040>
9054  08:55:54.163560 read(3</etc/locale.alias>, ..., 4096) = 2996 <0.000041>
9054  08:55:54.163679 read(3</etc/locale.alias>, "", 4096) = 0 <0.000044>
9054  08:55:54.176260 write(1</dev/pts/7>, ..., 50) = 50 <0.000111>
```

Trace file: host_9042.st

The Challenge: Extracting Information from System Traces

	pid	call	start	duration	bytes	fs	case	end
10	22085	read	1900-01-01 18:54:16.207116	0.000015	832	/p/software/fs/jusuf/stages/2024/software/HDF5...	st.jsfc134.22051.log	1900-01-01 18:54:16.207131
33	22085	read	1900-01-01 18:54:16.211619	0.000017	832	/p/software/fs/jusuf/stages/2024/software/psmp...	st.jsfc134.22051.log	1900-01-01 18:54:16.211636
221	22085	read	1900-01-01 18:54:16.246555	0.000008	832	/usr/lib64/libc.so.6	st.jsfc134.22051.log	1900-01-01 18:54:16.246563
231	22085	read	1900-01-01 18:54:16.247709	0.000015	832	/p/software/fs/jusuf/stages/2024/software/IME/...	st.jsfc134.22051.log	1900-01-01 18:54:16.247724
235	22085	read	1900-01-01 18:54:16.248466	0.000016	832	/p/software/fs/jusuf/stages/2024/software/Szip...	st.jsfc134.22051.log	1900-01-01 18:54:16.248482

Challenge: How do you extract useful information from large amounts of information in the system call traces?

The Challenge: Extracting Information from System Traces

	pid	call	start	duration	bytes	fs	case	end
10	22085	read	1900-01-01 18:54:16.207116	0.000015	832	/p/software/fs/jusuf/stages/2024/software/HDF5...	st.jsfc134.22051.log	1900-01-01 18:54:16.207131
33	22085	read	1900-01-01 18:54:16.211619	0.000017	832	/p/software/fs/jusuf/stages/2024/software/psmp...	st.jsfc134.22051.log	1900-01-01 18:54:16.211636
221	22085	read	1900-01-01 18:54:16.246555	0.000008	832	/usr/lib64/libc.so.6	st.jsfc134.22051.log	1900-01-01 18:54:16.246563
231	22085	read	1900-01-01 18:54:16.247709	0.000015	832	/p/software/fs/jusuf/stages/2024/software/IME/...	st.jsfc134.22051.log	1900-01-01 18:54:16.247724
235	22085	read	1900-01-01 18:54:16.248466	0.000016	832	/p/software/fs/jusuf/stages/2024/software/Szip...	st.jsfc134.22051.log	1900-01-01 18:54:16.248482

Typical questions one could ask looking at the above data:

- What is the **total read time** in the **$SOFTWARE** directory, i.e., under /p/software?
- What is the **rate of data movements** in the **$PROJECT** directory, i.e., under /p/project**?**

Idea: Map Trace Events to Activities

Idea:

- **Map each row to** a string that helps answer your question. We call this string "**Activity**".
- **Apply grouping** based on activities and **compute statistics**.
- **Identify the directly-follows relation** between the activities to build a **Directly-Follows-Graph (DFG).**

pid	call	start	duration	bytes	fs
22085	read	1900-01-01 18:54:16.207116	0.000015	832	/p/software/fs/jusuf/stages/2024/software/HDF5...
22085	read	1900-01-01 18:54:16.211619	0.000017	832	/p/software/fs/jusuf/stages/2024/software/psmp...
22085	read	1900-01-01 18:54:16.246555	0.000008	832	/usr/lib64/libc.so.6

Activity

→ read+/p/software

→ read+/p/software

→ read+/usr/lib64

Event-Log in Process Mining

pid	call	start	duration	bytes	fs
22085	read	1900-01-01 18:54:16.207116	0.000015	832	/p/software/fs/jusuf/stages/2024/software/HDF5...
22085	read	1900-01-01 18:54:16.211619	0.000017	832	/p/software/fs/jusuf/stages/2024/software/psmp...
22085	read	1900-01-01 18:54:16.246555	0.000008	832	/usr/lib64/libc.so.6

Activity

- read+/p/software
- read+/p/software
- read+/usr/lib64

- This data is can be formalized as an **event-log** in **Process Mining**.
- Process mining introduces **scalable techniques to translate event logs into** different types of dependency graphs, including **Directly-Follows Graph**.
- Ref: W. M. P. Van Der Aalst, “Foundations of process discovery,” in Process Mining Handbook, DOI: https://doi.org/10.1007/978-3-031-08848-3_2

Construction of Directly-Follows Graph

P0.strace.log

P1.strace.log

Activity
read+$SOFTWARE
read+$SOFTWARE
write+$PROJECT

Activity
read+$SOFTWARE
write+$PROJECT
write+$SCRATCH

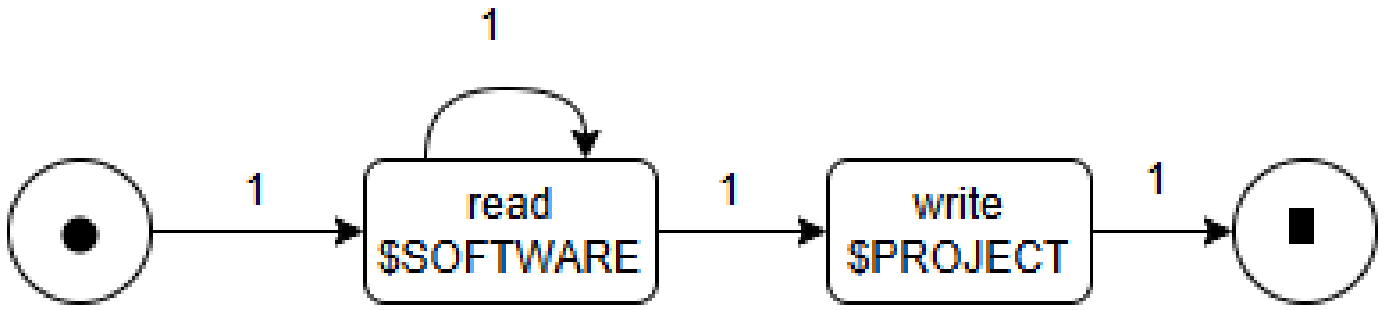
Construction of Directly-Follows Graph

P0

Activity
read+$SOFTWARE
read+$SOFTWARE
write+$PROJECT

P1

Activity
read+$SOFTWARE
read+$PROJECT
write+$PROJECT



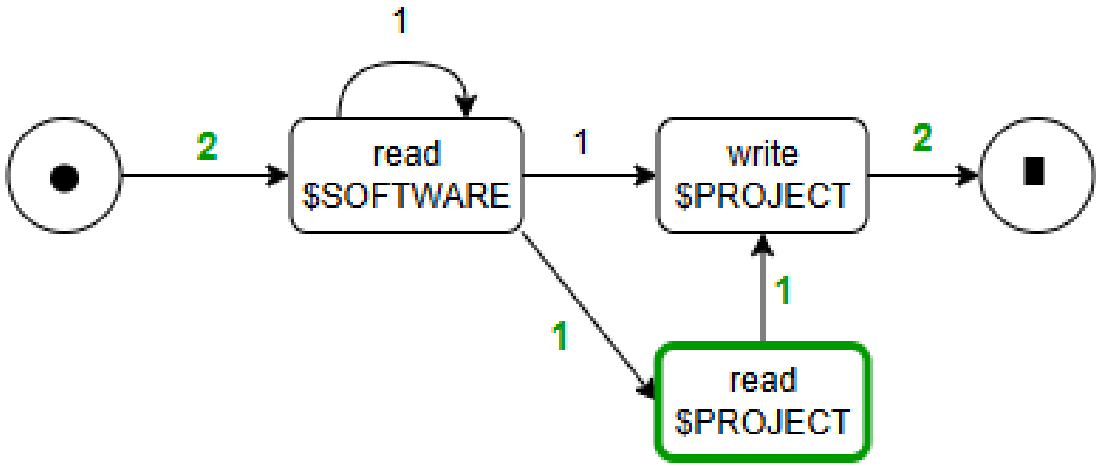
Construction of Directly-Follows Graph

P0

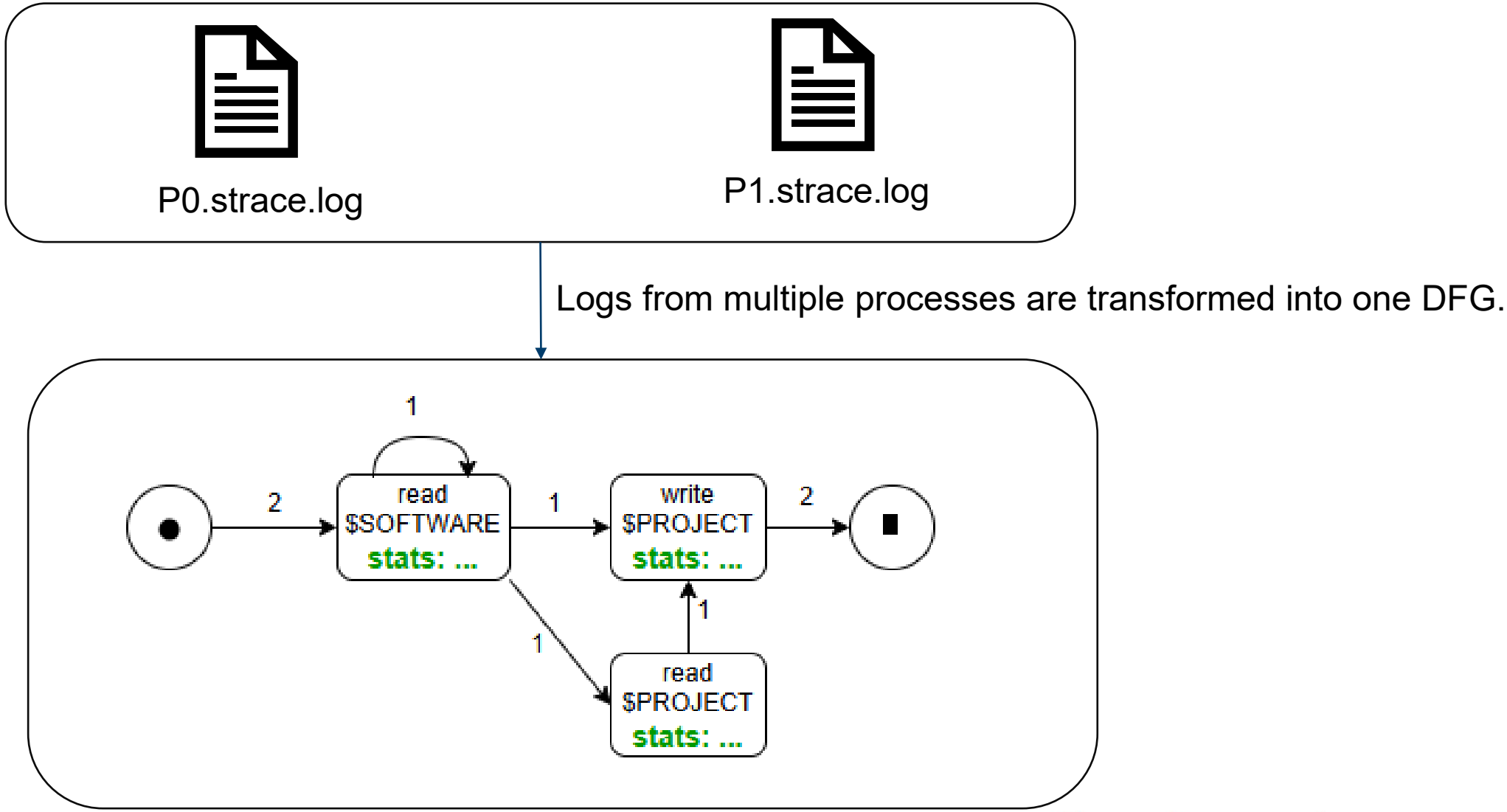
Activity
read+$SOFTWARE
read+$SOFTWARE
write+$PROJECT

P1

Activity
read+$SOFTWARE
read+$PROJECT
write+$PROJECT



Composing Strace Logs into a DFG

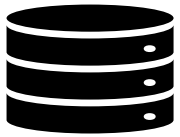


Usage

Prefix your executable with the tracer.

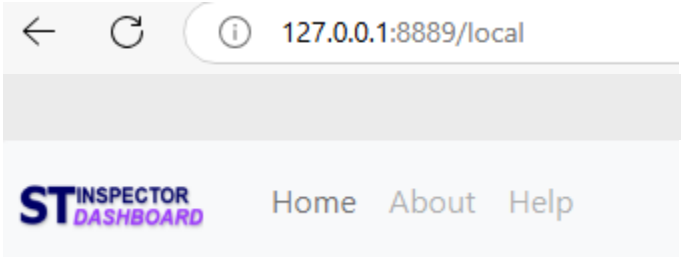
```
srun -n [N] [OPTIONS]  ./strace.sh [COMMAND 1]
srun -n [N] [OPTIONS]  ./strace.sh [COMMAND 2]
```

strace logs stored in user space



ssh tunnel

Interactive dashboard for trace inspection



IOR Example 1: File per Process vs Single Shared File

We trace IOR benchmark (on 96 cores) with the following configurations:

- All process write to a **single shared file** in the directory `$SCRATCH/ssf`
- All process **write to its own file** in the directory `$SCRATCH/fpp`

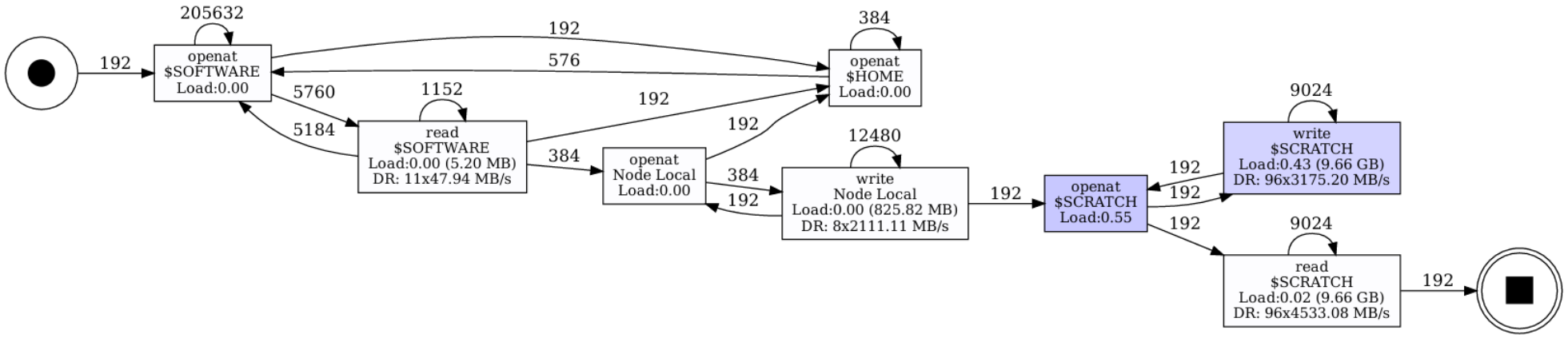
```
#Single Shared File
srun -n 96 ./strace.sh ./ior -t 1m -b 16m -s 3 -w -r -C -e -o $SCRATCH/ssf/test

#One File per Process
srun -n 96 ./strace.sh ./ior -t 1m -b 16m -s 3 -w -r -F -C -e -o $SCRATCH/fpp/test
```

A Peek into the Dashboard

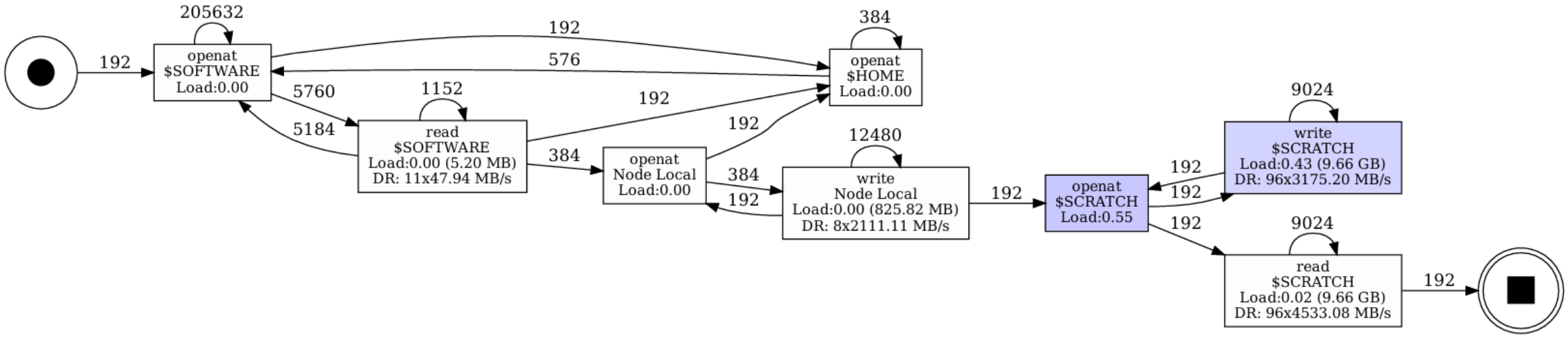
Committed Traces:

Job ID	Start	Cmds	State	Details	Last Update	Actions
i241104102315	2024-11-04 10:23:15	./main 40	Traced	no. of steps: 2 success: 2 errors: 0 run time: 3.0 size: 20.39 MB	2024-11-04 10:23:54	inspect



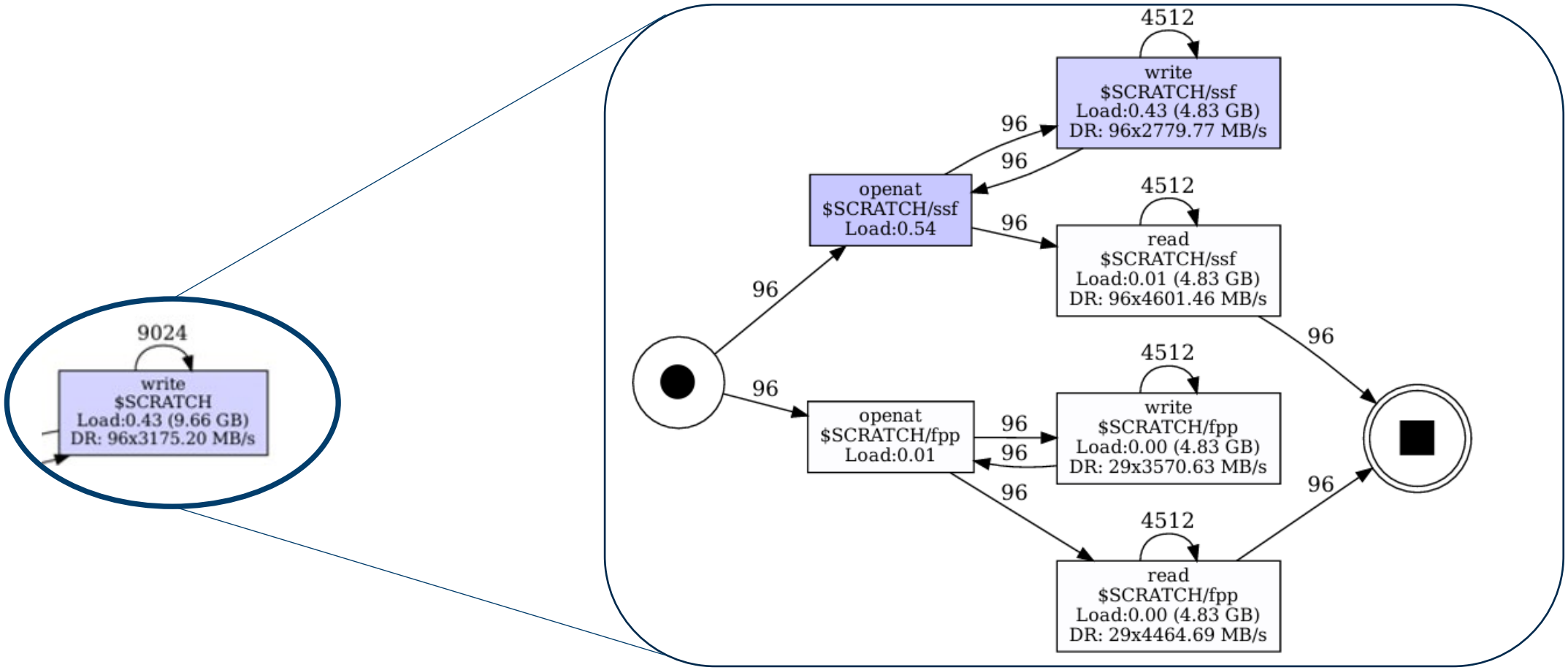
IOR Example 1: File per Process vs Single Shared File

The combined DFG for both runs:



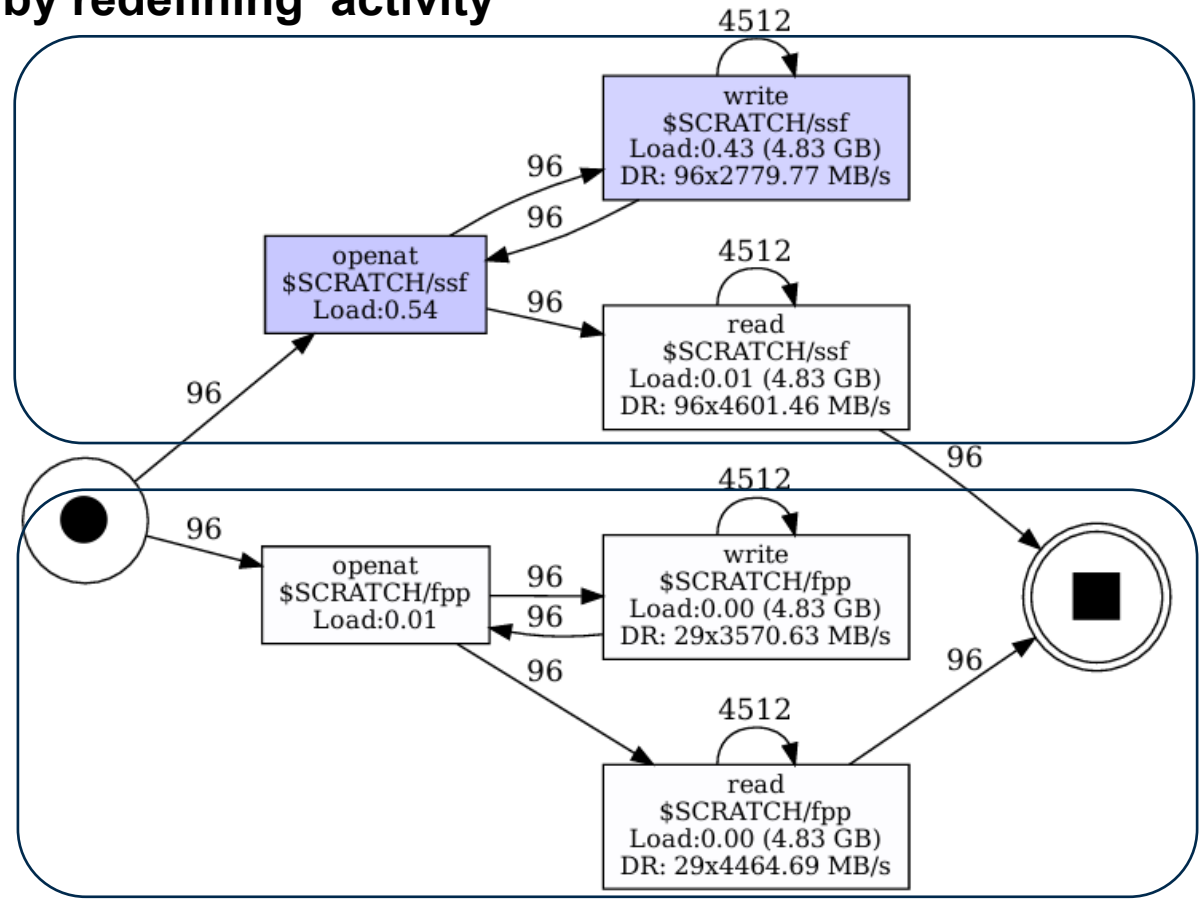
IOR Example 1: File per Process vs Single Shared File

Focus on $SCRATCH by redefining ‘activity’



IOR Example 1: File per Process vs Single Shared File

Focus on $SCRATCH by redefining 'activity'

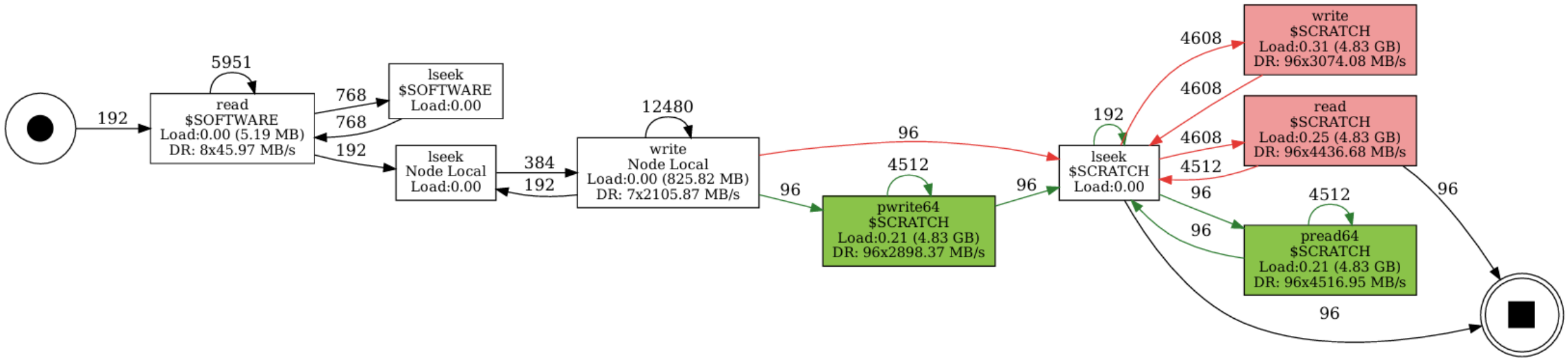


Highlights the performance impacts of configuration choices

IOR Example 2: With and Without MPI-IO

```
#POSIX
srun -n 96 ./strace.sh ./ior -t 1m -b 16m -s 3 -w -r -C -e -o $SCRATCH/ssf/test

#MPIIO
srun -n 96 ./strace.sh ./ior -t 1m -b 16m -s 3 -a mpiio -w -r -C -e -o $SCRATCH/ssf/test
```



Highlights the difference in I/O behavior through graph coloring.

Summary

- We presented a methodology to synthesize a DFG from traces of system calls.
- We showed the usage of the methodology in comparing I/O operations of arbitrary programs.

What next?

- Minimize the overheads in trace collection with *SIONlib*.
- Inspect real applications.

Acknowledgements

- This research was financially supported by the Juelich Supercomputing Center at Forschungszentrum Juelich and the BMBF project 01-1H1-6013 AP6 NRW Anwenderunterstutzung SiVeGCS.

- Additionally, supervision support from RWTH Aachen University through the DFG project IRTG-2379 is gratefully acknowledged.