PRACE PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE

JÜLICH FORSCHUNGSZENTRUM

Parallel I/O strategies

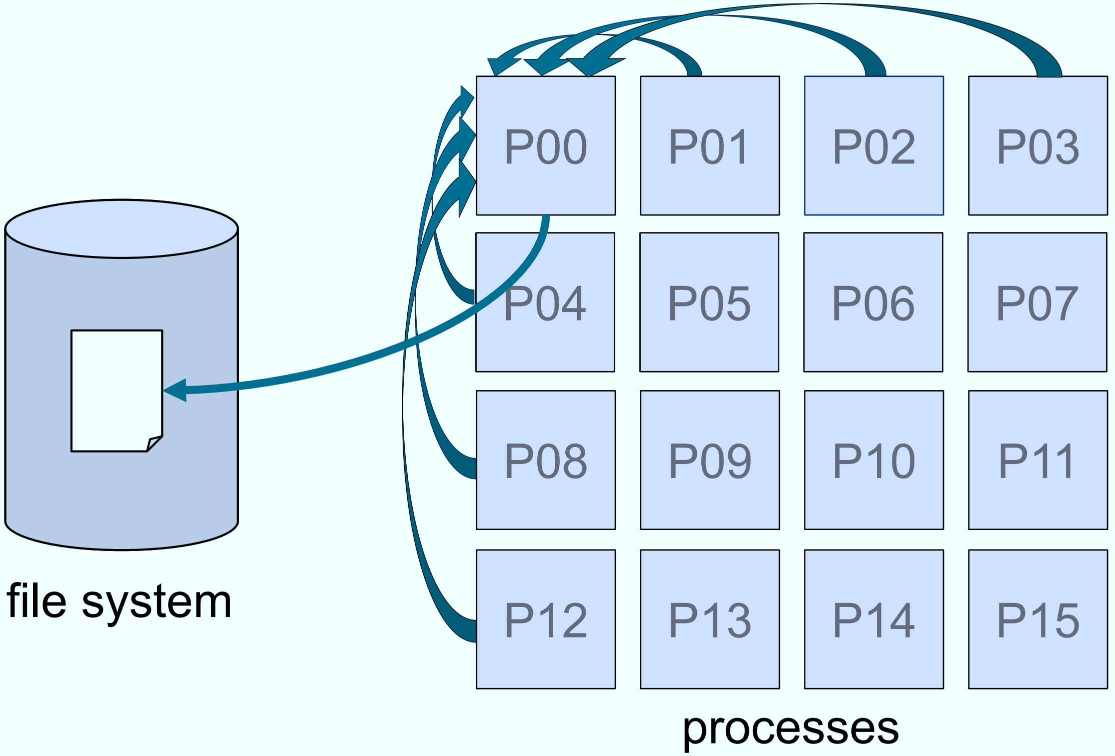
Sebastian Lührs
s.luehrs@fz-juelich.de
Jülich Supercomputing Centre
Forschungszentrum Jülich GmbH

Ostrava, March 22nd, 2018

Outline

- Common I/O strategies
    - One process performs I/O
    - Task-local files
    - Shared files
- I/O workflow
- Pitfalls
- Parallel I/O software stack
- Course exercise description
    - General exercise workflow
    - Mandelbrot set description
    - Exercise API

One process performs I/O



P00 P01 P02 P03

P04 P05 P06 P07

P08 P09 P10 P11

P12 P13 P14 P15

file system

processes

One process performs I/O

+ Simple to implement

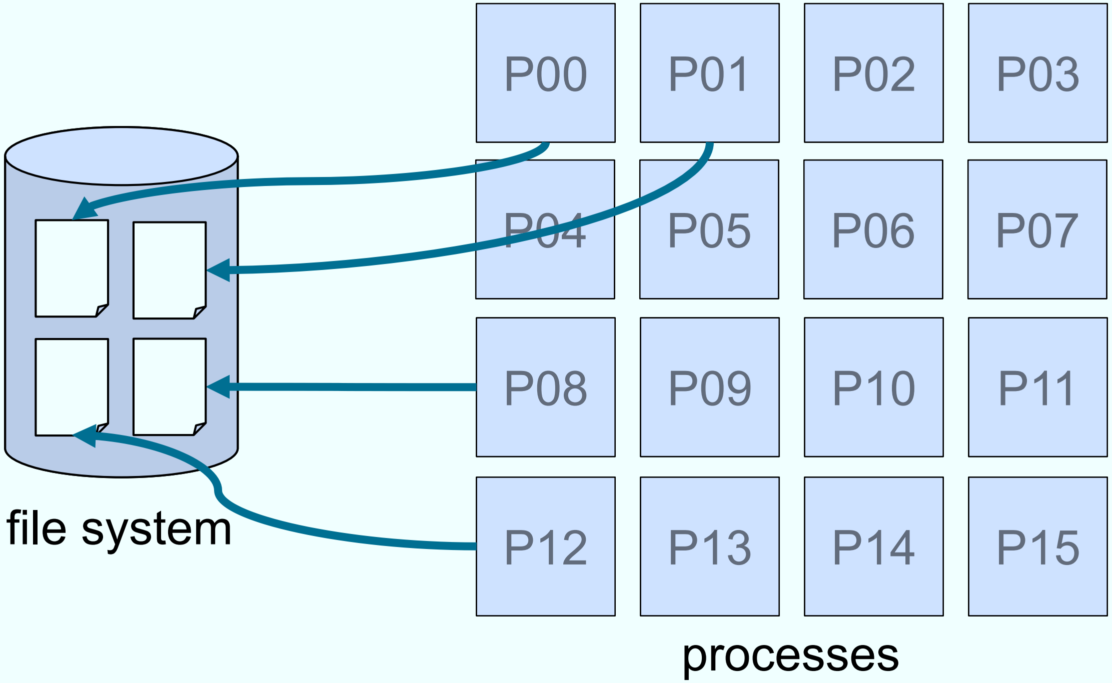
- I/O bandwidth is limited to the rate of this single process
- Additional communication might be necessary
- Other processes may idle and waste computing resources during I/O time

Frequent flushing on small blocks

Pitfall 1

- Modern file systems in HPC have **large file system blocks** (e.g. 4MB)
- A flush on a file handle forces the file system to perform all pending write operations
- If application writes in small data blocks, the same file system block it has to be **read and written multiple times**
- Performance degradation due to the inability to combine several write calls

Task-local files



Task-local files

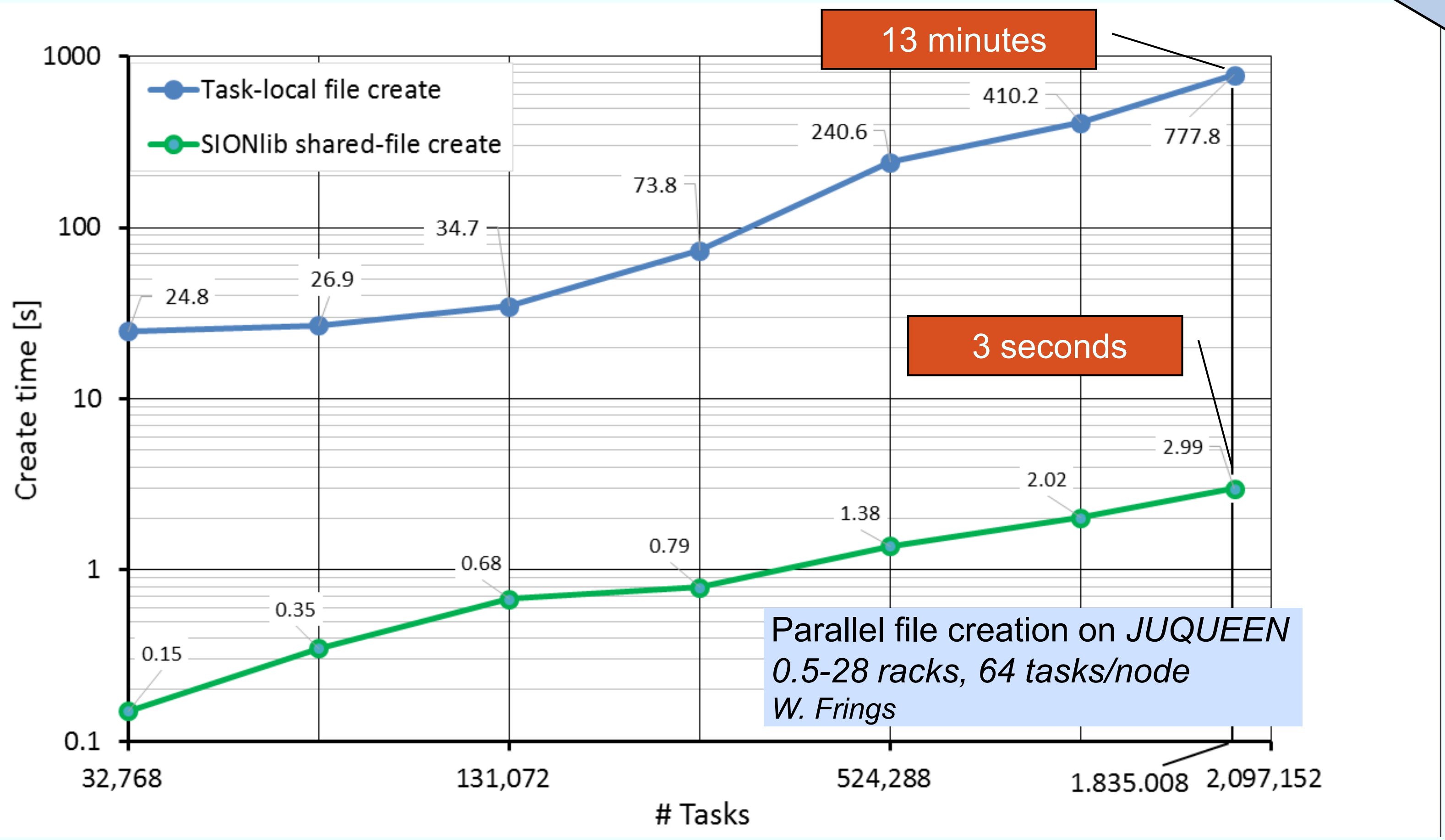
- + Simple to implement
- + No coordination between processes needed
- + No false sharing of file system blocks

- - Number of files quickly becomes unmanageable
- - Files often need to be merged to create a canonical dataset
- - File system might serialize meta data modification

Serialization of meta data modification

Pitfall 2

Example: Creating files in parallel in the same directory



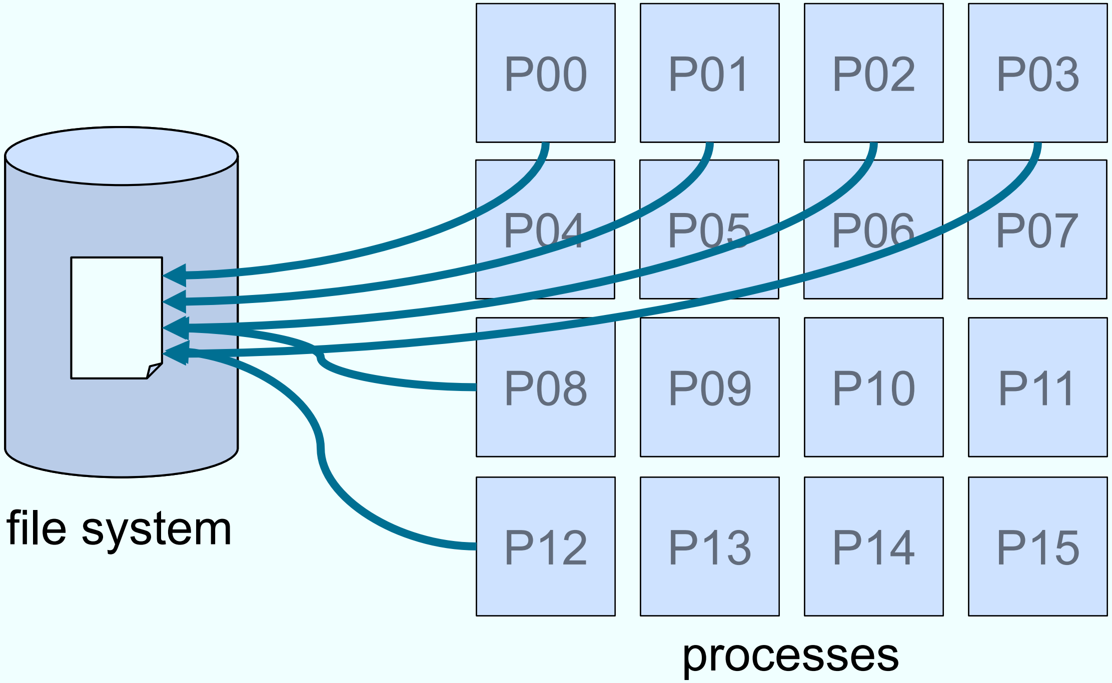
13 minutes

3 seconds

Parallel file creation on *JUQUEEN*
0.5-28 racks, 64 tasks/node
W. Frings

The creation of 2.097.152 files costs 113.595 core hours on JUQUEEN!

Shared files

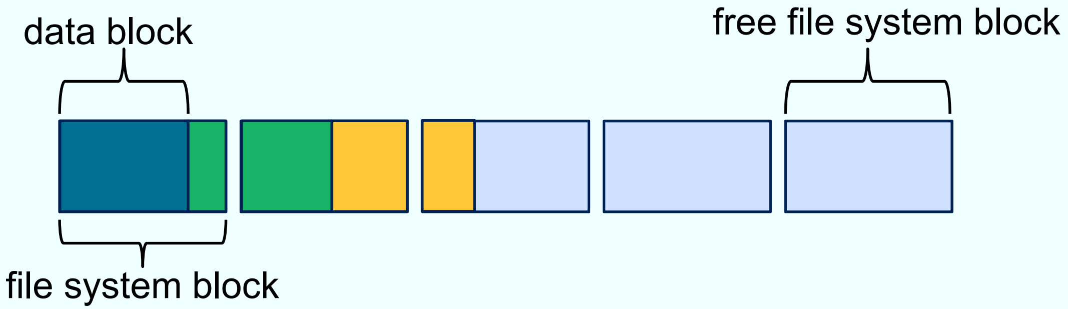


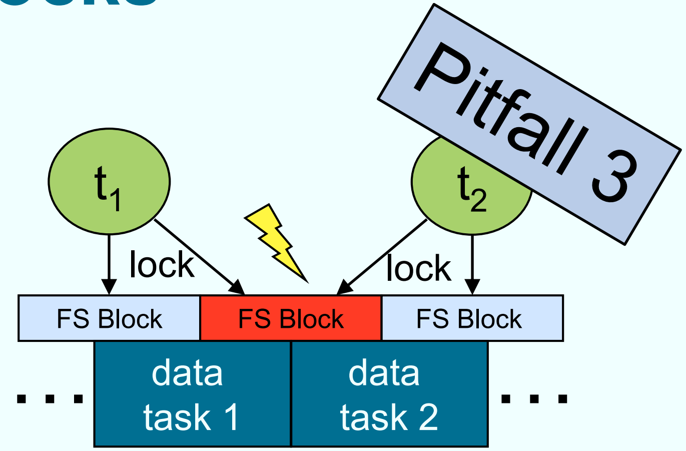
Shared files

+ Number of files is independent of number of processes
+ File can be in canonical representation (no post-processing)

- Uncoordinated client requests might induce time penalties
- File layout may induce false sharing of file system blocks

False sharing of file system blocks



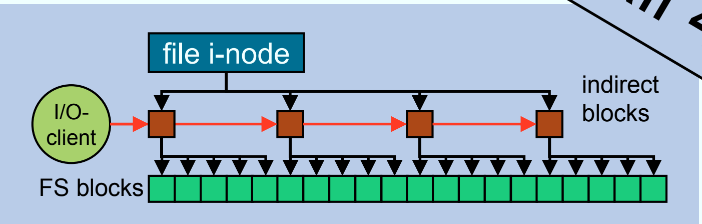


- Data blocks of individual processes **do not fill up a complete file system block**
- Several processes **share a file system block**
- Exclusive access (e.g. write) must be **serialized**
- The more processes have to synchronize the more waiting time will propagate

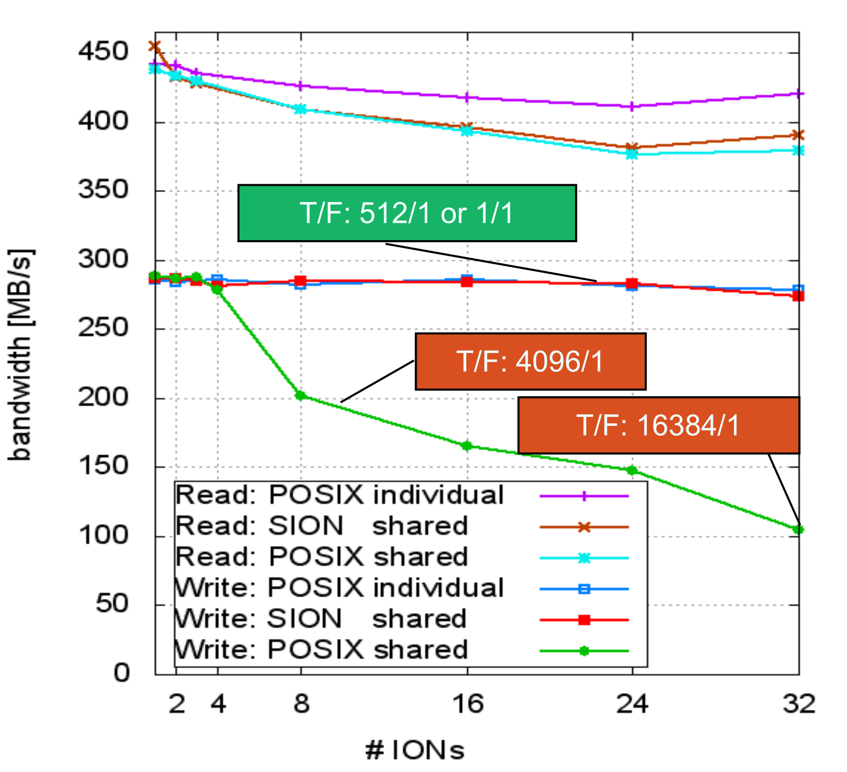
Number of Tasks per Shared File

Pitfall 4

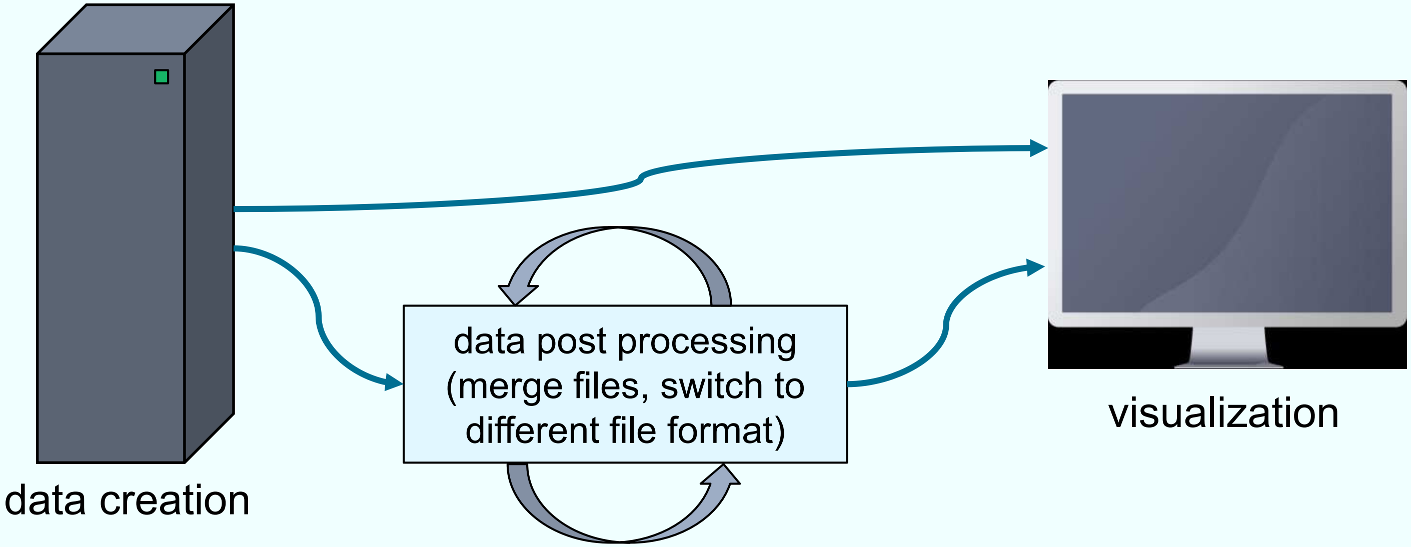
- Meta-data wall on file level
  - File meta-data management
  - Locking



- Example Blue Gene/P
  - Jugene (72 racks)
  - I/O forwarding nodes (ION)
  - GPFS client on ION
  - One file per ION



I/O Workflow

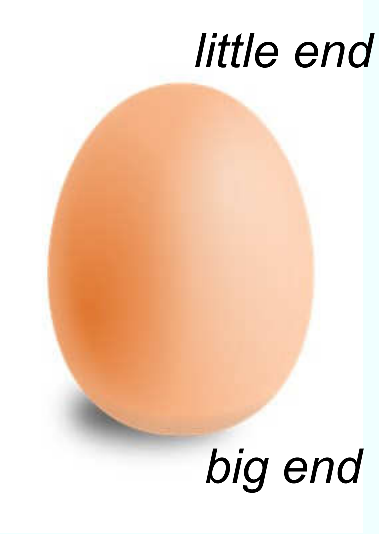

data post processing (merge files, switch to different file format)
data creation
visualization

- Post processing can be very time-consuming (> data creation)
  - Widely used portable data formats avoid post processing
- Data transportation time can be long:
  - Use shared file system for file access, avoid raw data transport
  - Avoid renaming/moving of big files (can block backup)

Portability

Pitfall 5

- Endianness (byte order) of binary data



2,712,847,316

=

10100001 10110010 11000011 11010100

Address	Little Endian	Big Endian
1000	11010100	10100001
1001	11000011	10110010
1002	10110010	11000011
1003	10100001	11010100

- Conversion of files might be necessary and expensive

Portability

Pitfall 6

- Memory order depends on programming language

1	2	3
4	5	6
7	8	9

→

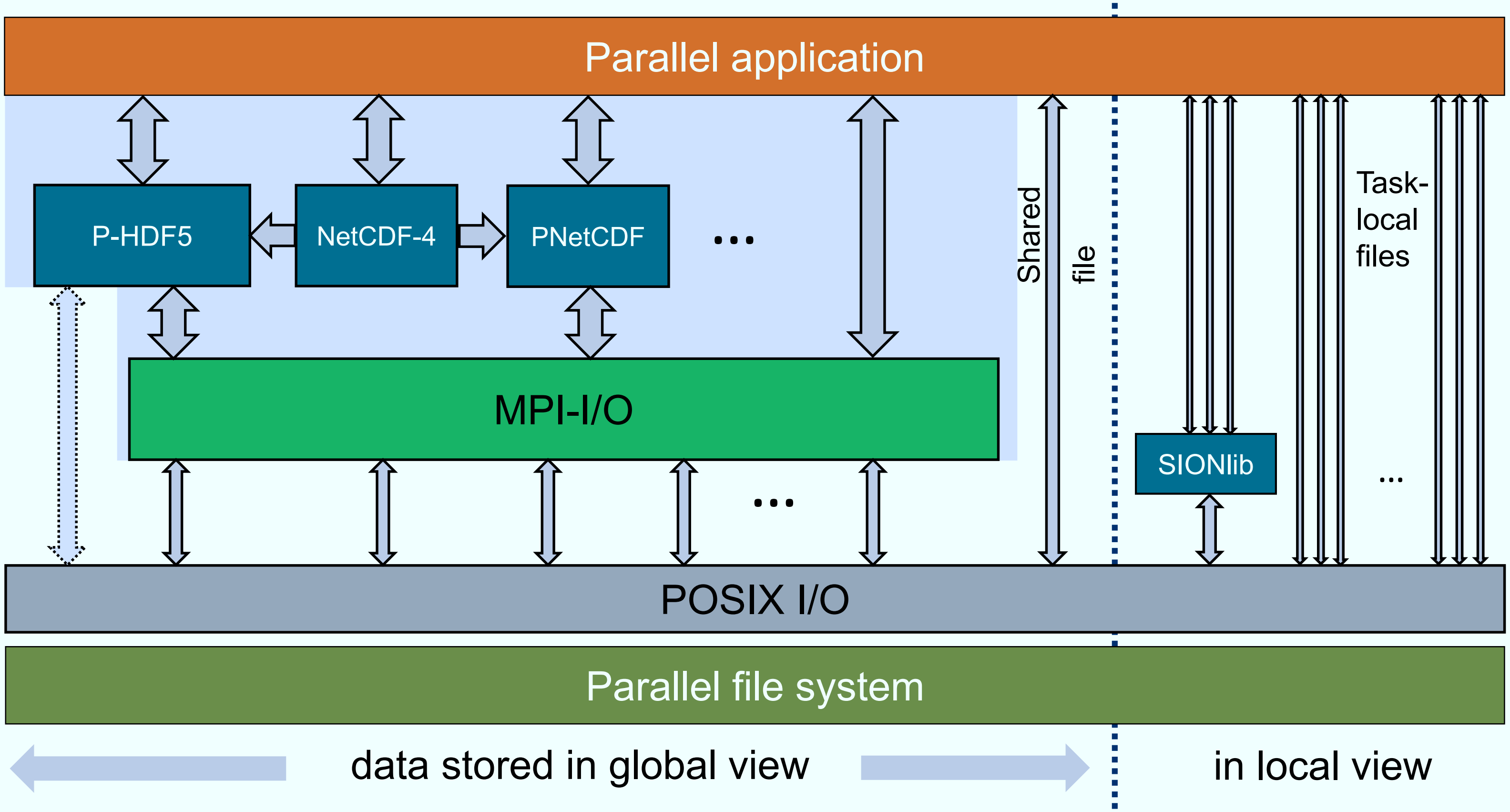
Address	row-major order (e.g. C/C++)	column-major order (e.g. Fortran)
1000	1	1
1001	2	4
1002	3	7
1003	4	2
1004	5	5
…	…	…

- Transpose of array might be necessary when using different programming languages in the same workflow
- Solution: Choosing a portable data format (HDF5, NetCDF)

How to choose the I/O strategy?

- Performance considerations
  - Amount of data
  - Frequency of reading/writing
  - Scalability
- Portability
  - Different HPC architectures
  - Data exchange with others
  - Long-term storage
- E.g. use two formats and converters:
  - Internal: Write/read data “as-is”
    → Restart/checkpoint files
  - External: Write/read data in non-decomposed format
    (portable, system-independent, self-describing)
    → Workflows, Pre-, Post-processing, Data exchange

Parallel I/O Software Stack



I/O Profiling with Darshan

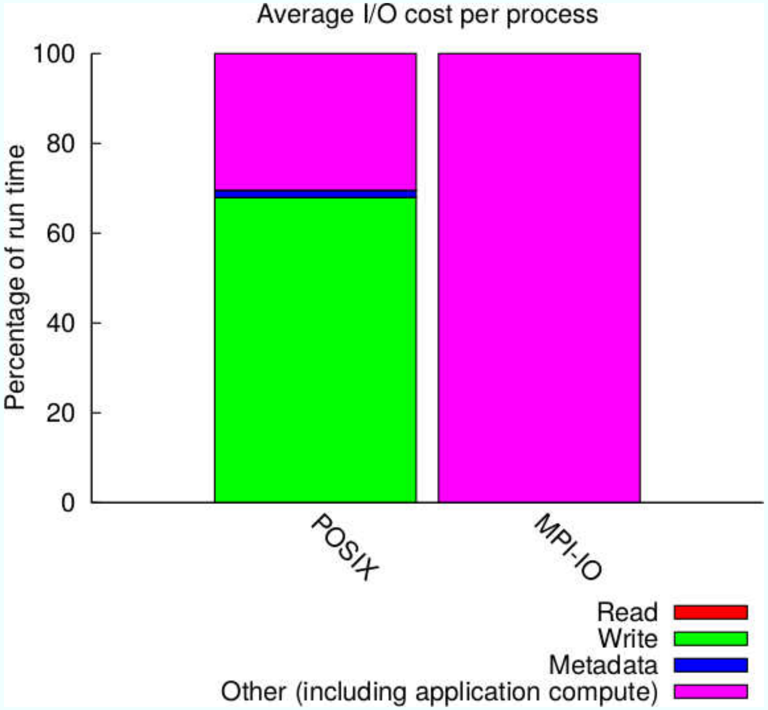
- I/O profiler by Argonne National Lab: http://www.mcs.anl.gov/research/projects/darshan/
- Darshan module (Salomon)
  - module load darshan-runtime
- Tell to use Darshan (in submit script)

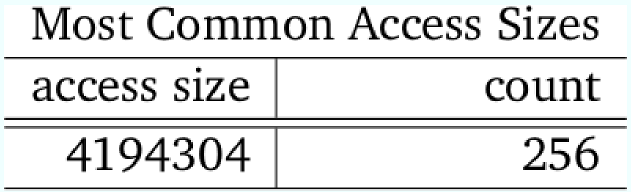
```
export LD_PRELOAD=$EBROOTDARSHANMINRUNTIME/lib/libdarshan.so \
export DARSHAN_LOG_PATH=/path/to/your/logile \
exportDARSHAN_LOGFILE=darshan.log \
mpiexec ./executable
```

- Analyse output
  - module load darshan-util
  - darshan-parser darshan.log
  - darshan-job-summary.pl darshan.log (needs pdflatex)

Darshan: Interpret the summary

- Average and statistical information on I/O patterns
    - Relative time for I/O
    - Most common access sizes
- Additional metrics
    - File count
    - I/O size histogram
    - Timeline for read / write per task
    - …




Most Common Access Sizes

access size	count
4194304	256