JÜLICH
FORSCHUNGSZENTRUM

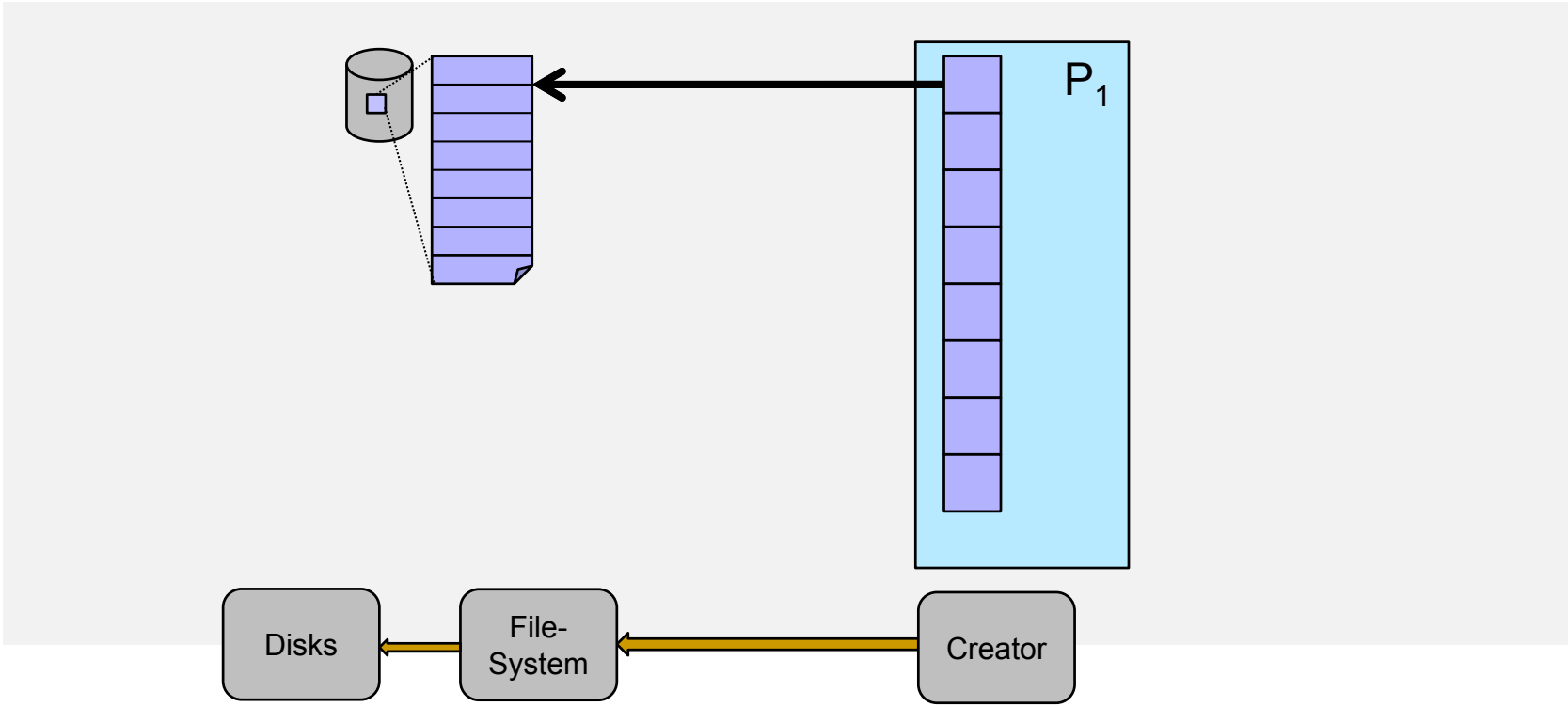
Parallel Task-Local File I/O with SIONlib

Jülich Supercomputing Centre
14 March 2017 | Kay Thust

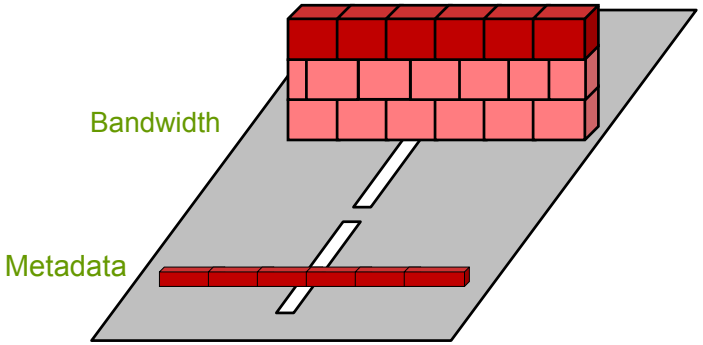
Outline

- Introduction
  - Motivation
  - SIONlib in a Nutshell
  - SIONlib API (parallel, serial)
  - Details
- Interface
- Example
- Tools
- Exercises

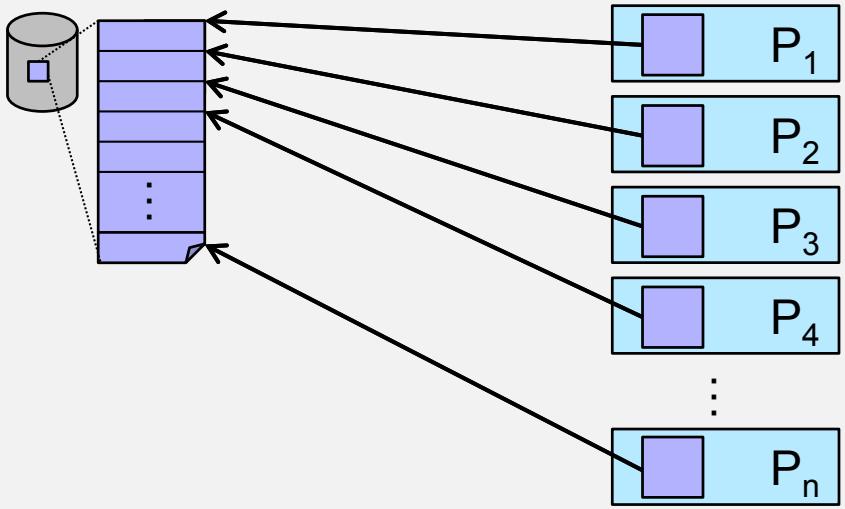
Where are the Bottlenecks?
… using Serial I/O …

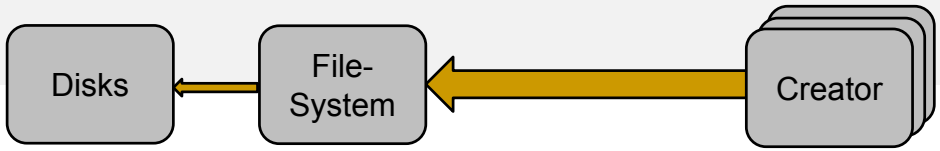



```
Number of Tasks:                    1
Number of Files:                    1
Bandwidth:                ~ 100 MB/s
I/O-Method:                    POSIX
Streams:                            1
Bottleneck Metadata:               no
Bottleneck Bandwidth:  (>>10 GB) yes
```

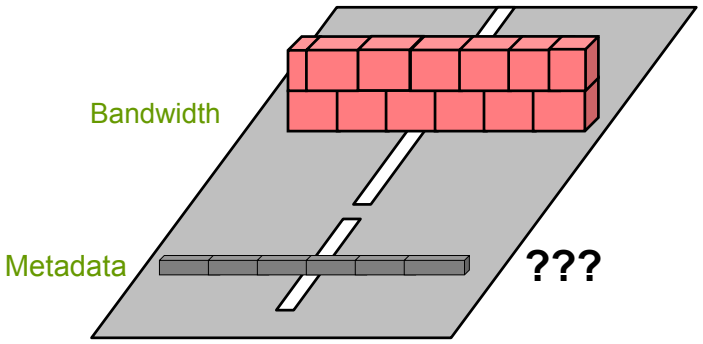


Where are the Bottlenecks?
… using Parallel I/O … Multiple Tasks …

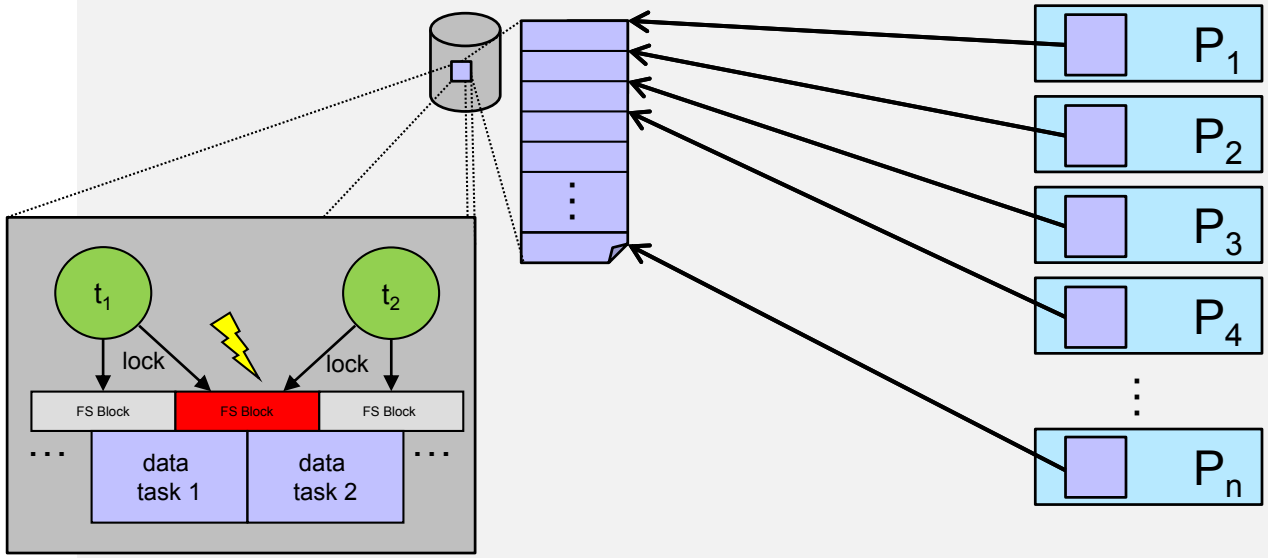


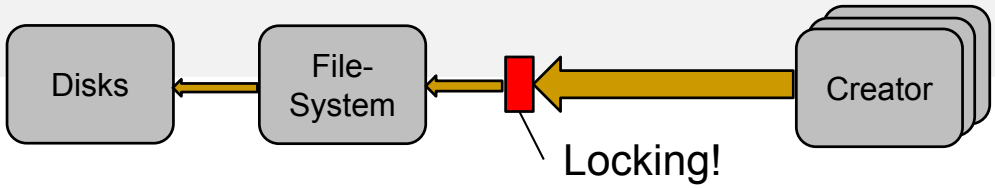


Number of Tasks:	n
Number of Files:	1
Bandwidth:	~ 100 MB/s
I/O-Method:	POSIX
Streams:	n
Bottleneck Metadata:	?
Bottleneck Bandwidth:	(>>10 GB) yes



Where are the Bottlenecks?
… using Parallel I/O … Locking …





Experminent:

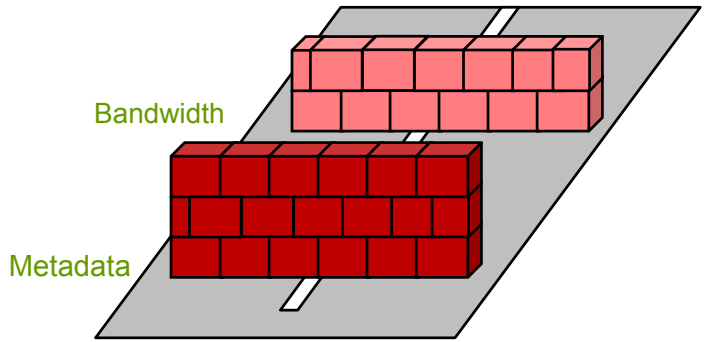
BG/P → GPFS (Write),
32768 tasks, 256 GB

aligned	**3650.2** MB/s
not aligned	**1863.8** MB/s

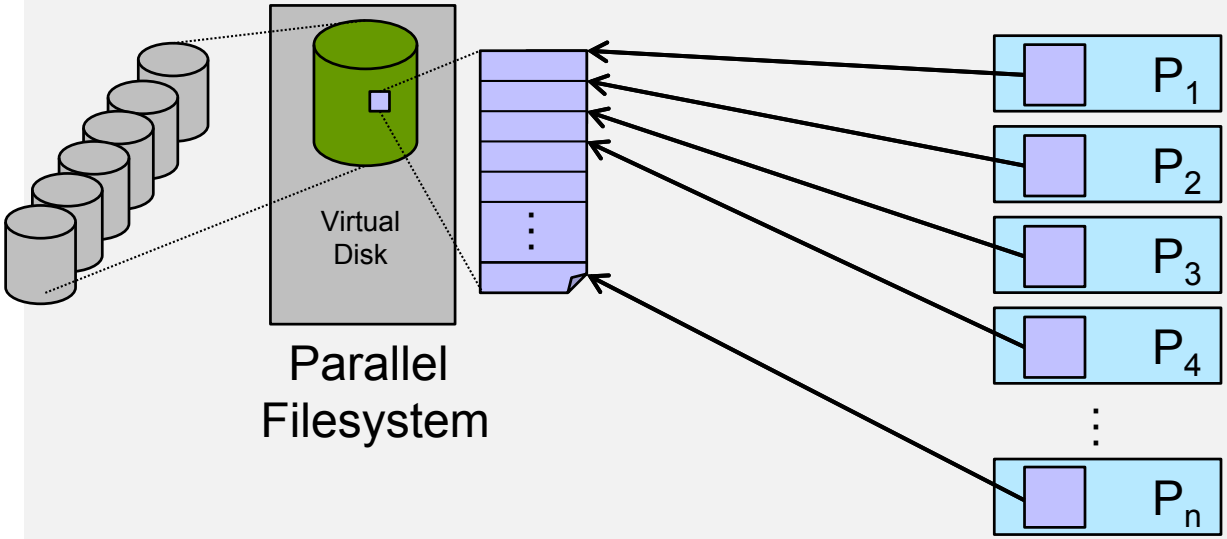
2x faster

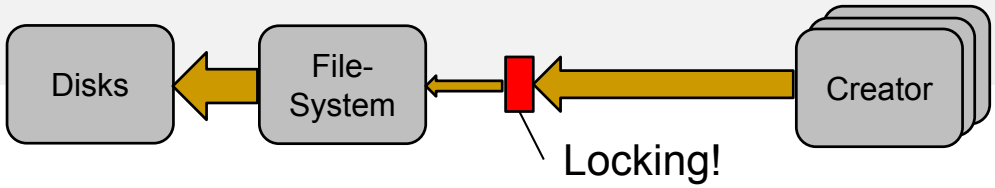
Jugene (JSC, IBM Blue Gene/P, GPFS, fs:work)

Number of Tasks:	n
Number of Files:	1
Bandwidth:	~ 100 MB/s
I/O-Method:	POSIX
Streams:	n
Bottleneck Metadata:	(locking) yes
Bottleneck Bandwidth:	yes

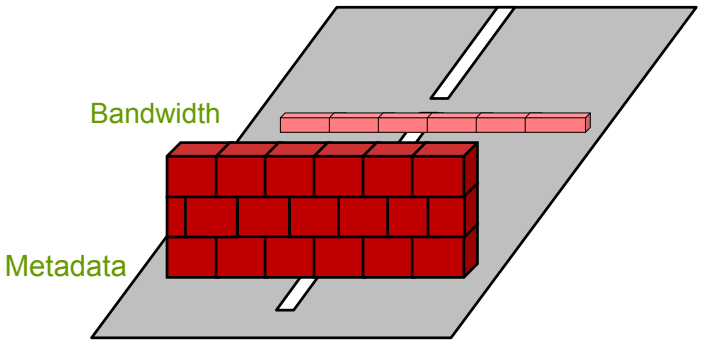


Where are the Bottlenecks?
… using Parallel I/O … Parallel Filesystem …

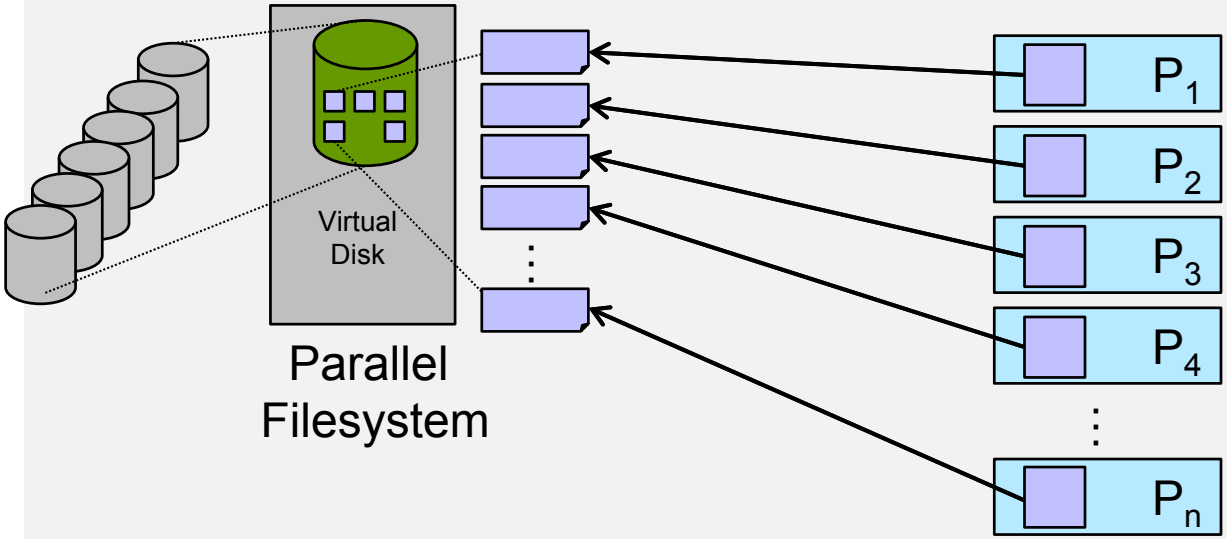


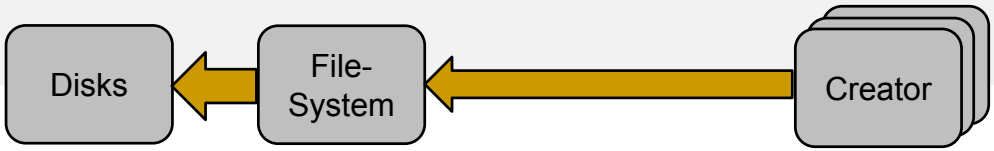



```
Number of Tasks:                    n
Number of Files:                    1
Bandwidth:                  >> 1 GB/s
I/O-Method:                     POSIX
Streams:                            n
Bottleneck Metadata:    (locking) yes
Bottleneck Bandwidth:              no
```

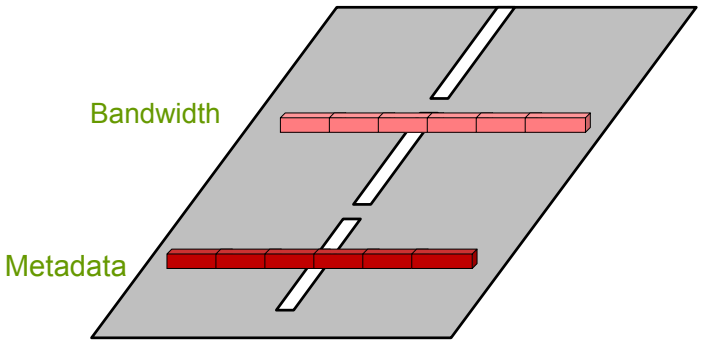


Where are the Bottlenecks?
… using Parallel I/O … Task-local Files …

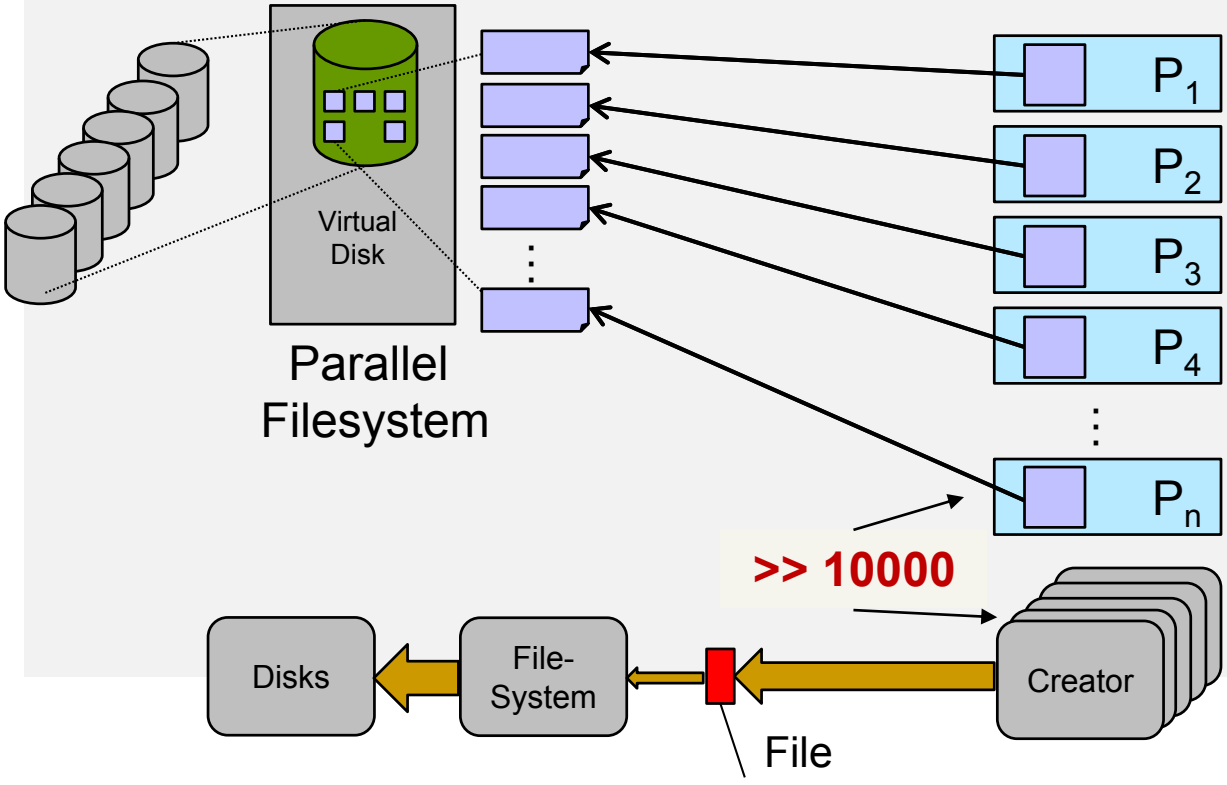


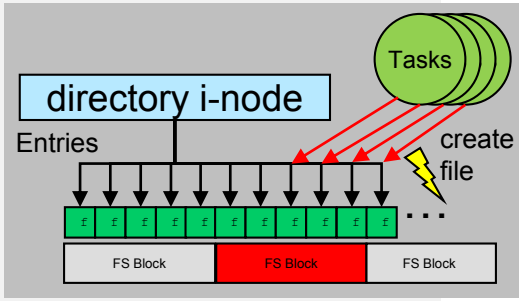



```
Number of Tasks:                        n
Number of Files:                        1
Bandwidth:                      >> 1 GB/s
I/O-Method:                         POSIX
Streams:                                n
Bottleneck Metadata (file create)     yes
Bottleneck Bandwidth:                  no
```

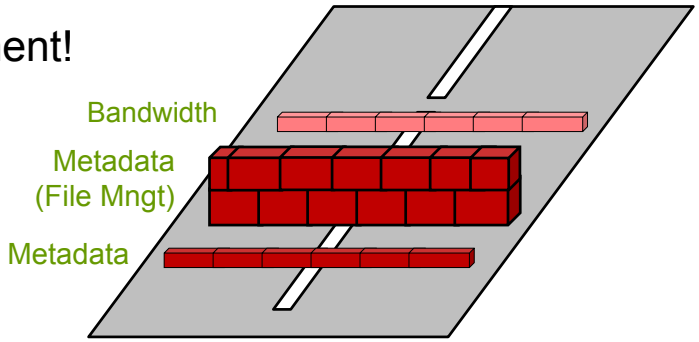


Where are the Bottlenecks?
… Increasing #Tasks … Task-local Files …

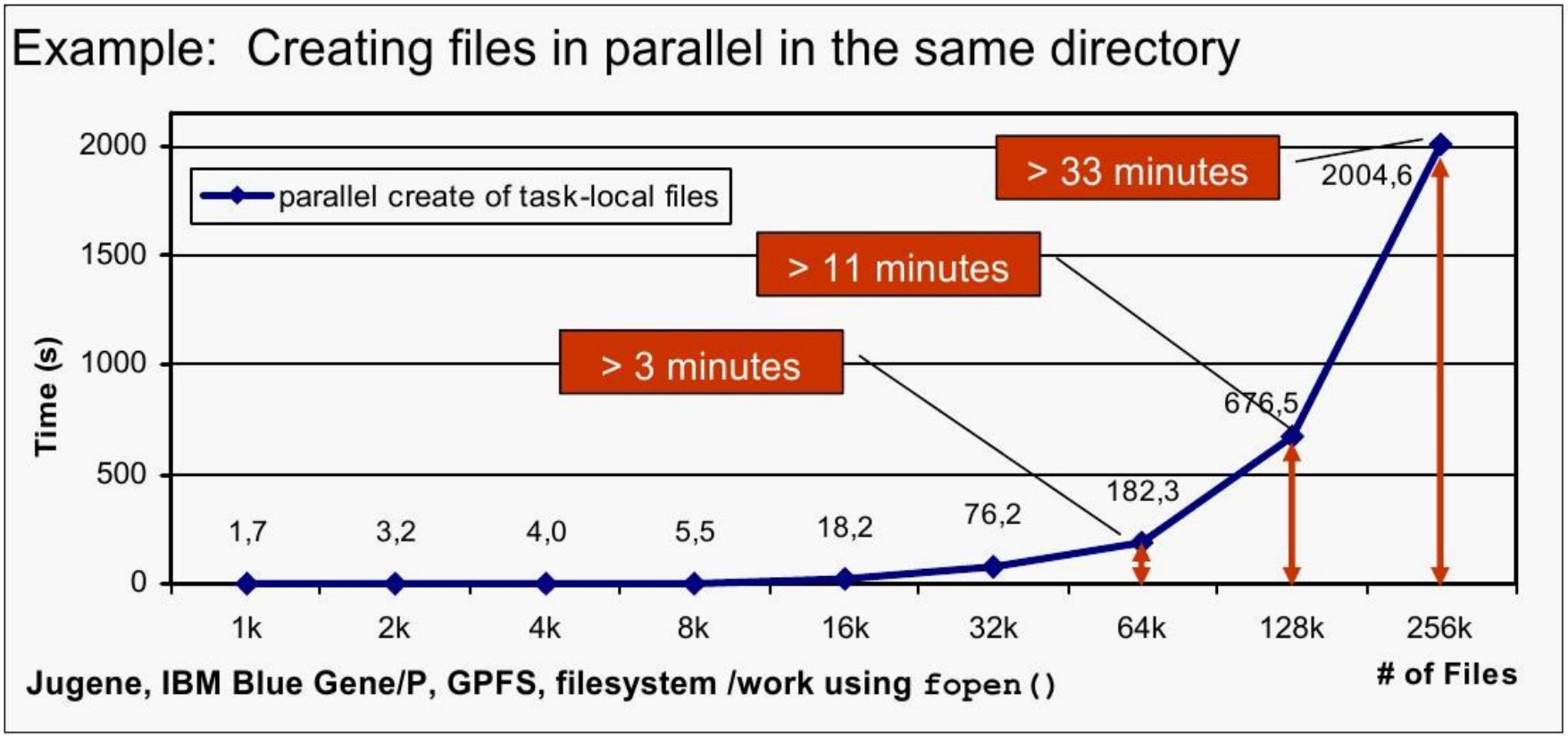




Number of Tasks:	(>> 10000) n
Number of Files:	1
Bandwidth:	>> 1 GB/s
I/O-Method:	POSIX
Streams:	(>> 10000) n
Bottleneck Metadata:	yes
Bottleneck Bandwidth:	no



Limitations of Task-Local I/O: File Creation & Management

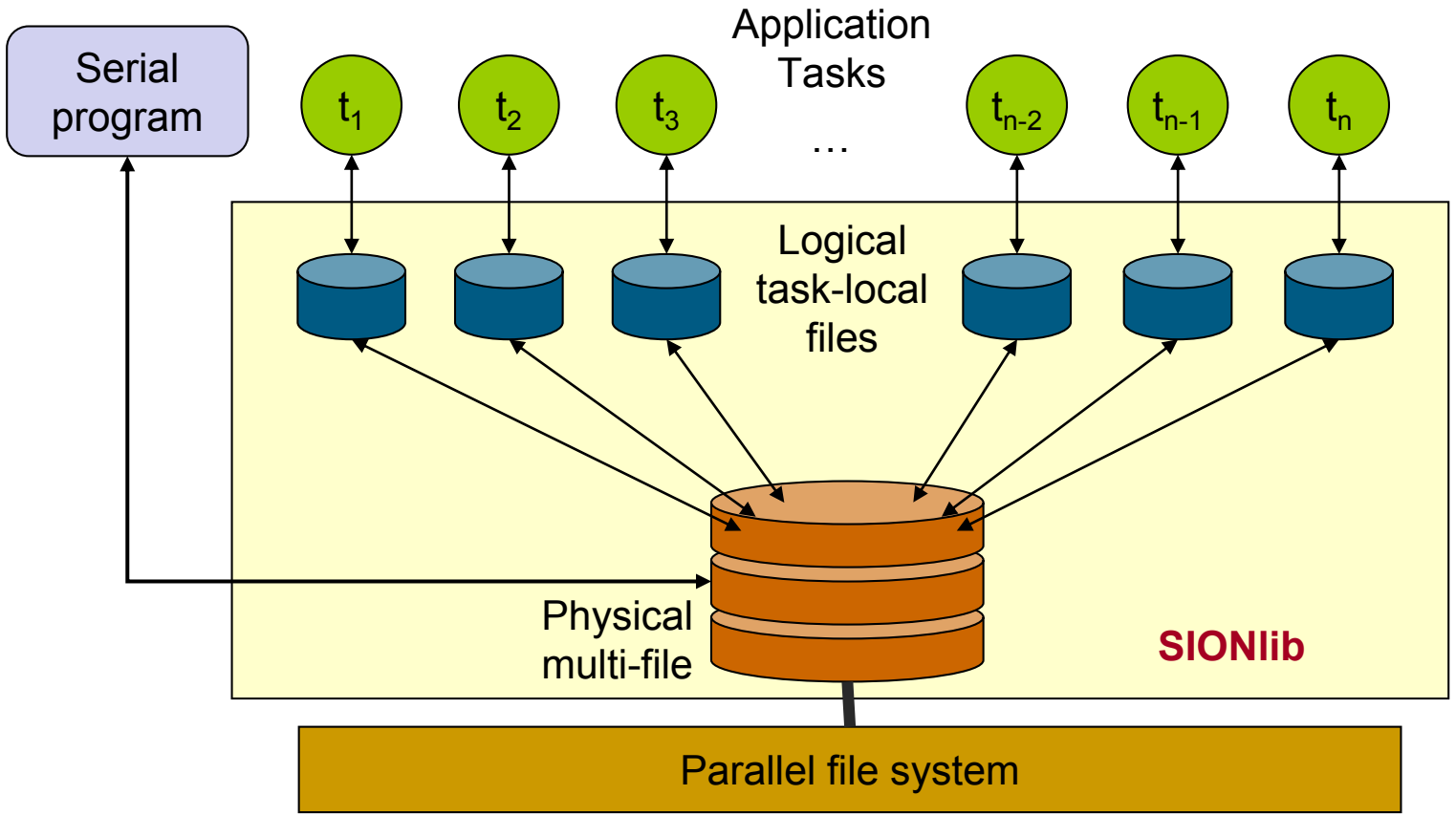


- Contention at the metadata server
- May degrade system I/O performance also for other users
- complicated file handling (e.g. archive)

Motivation: Using Shared Files

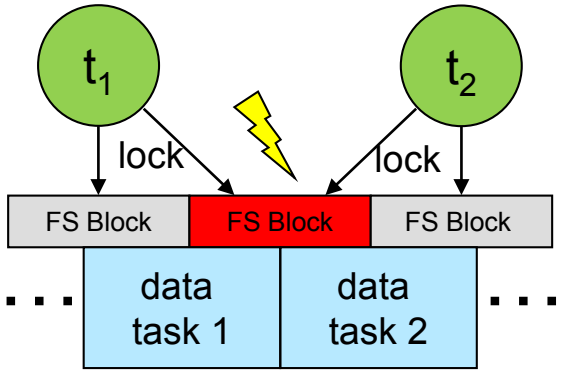
Idea: Mapping many logical files onto one or a few physical file(s)

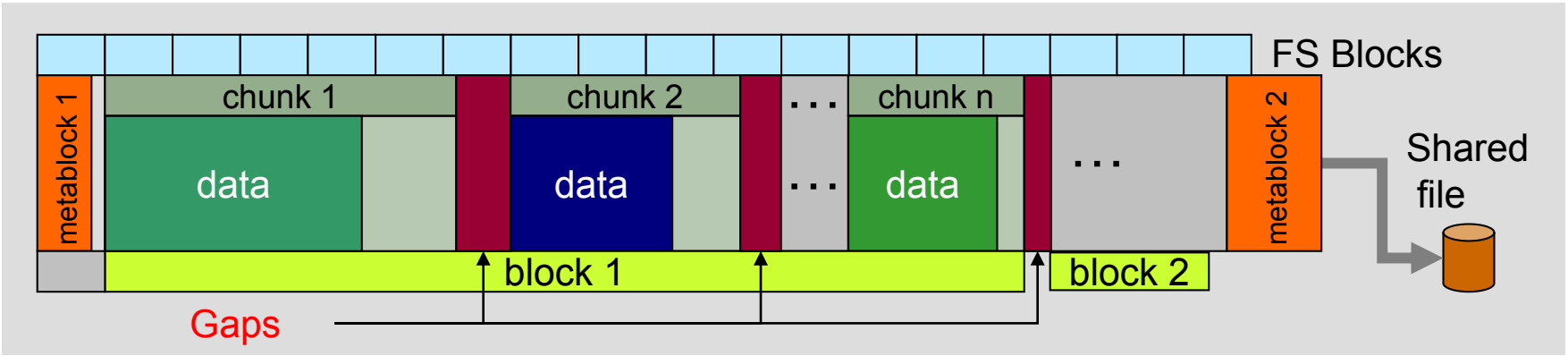
→ Task-local view to local data not changed



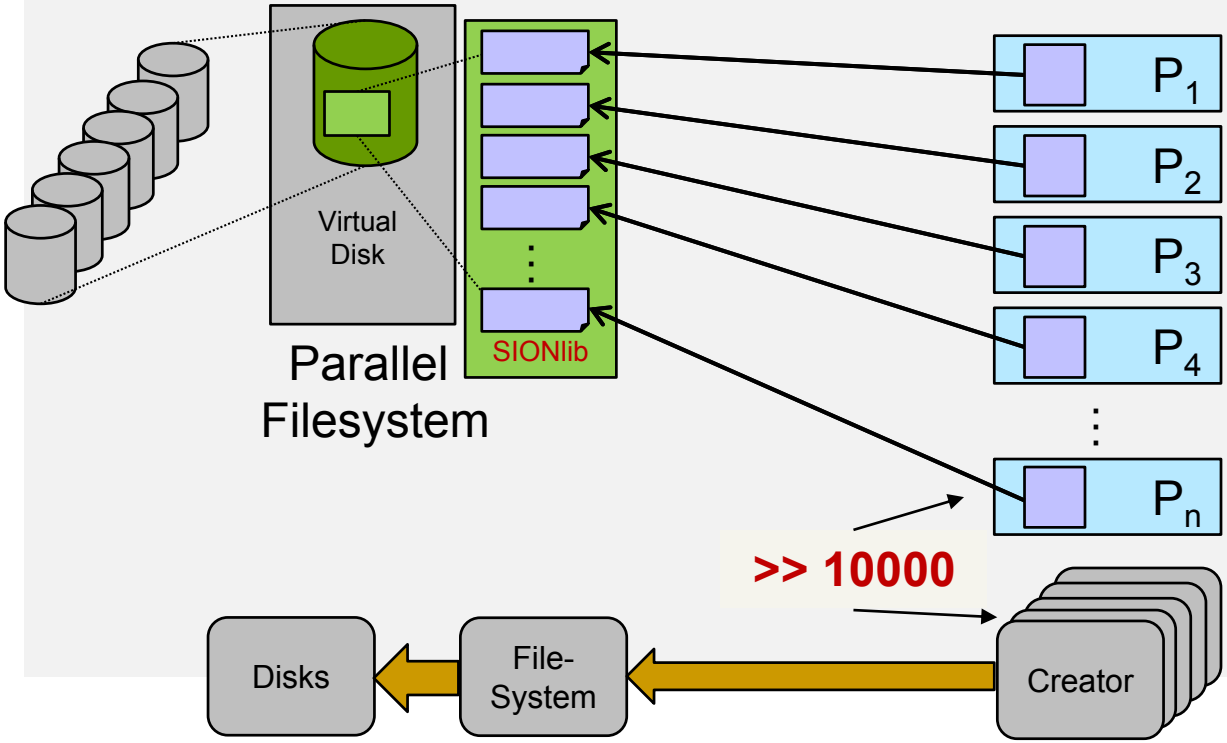
SIONlib File Format

- Logical partitioning of Shared File
- Dedicated data chunks per task, one or more
- Alignment to boundaries of file system blocks (no contention)
- Metadata blocks: e.g. chunk position, size and usage
- Contention when writing to same file-system block in parallel

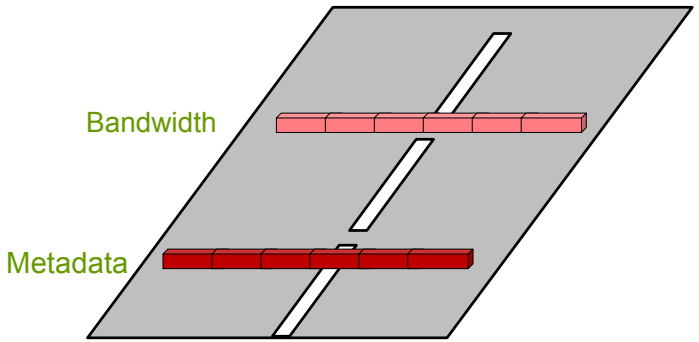




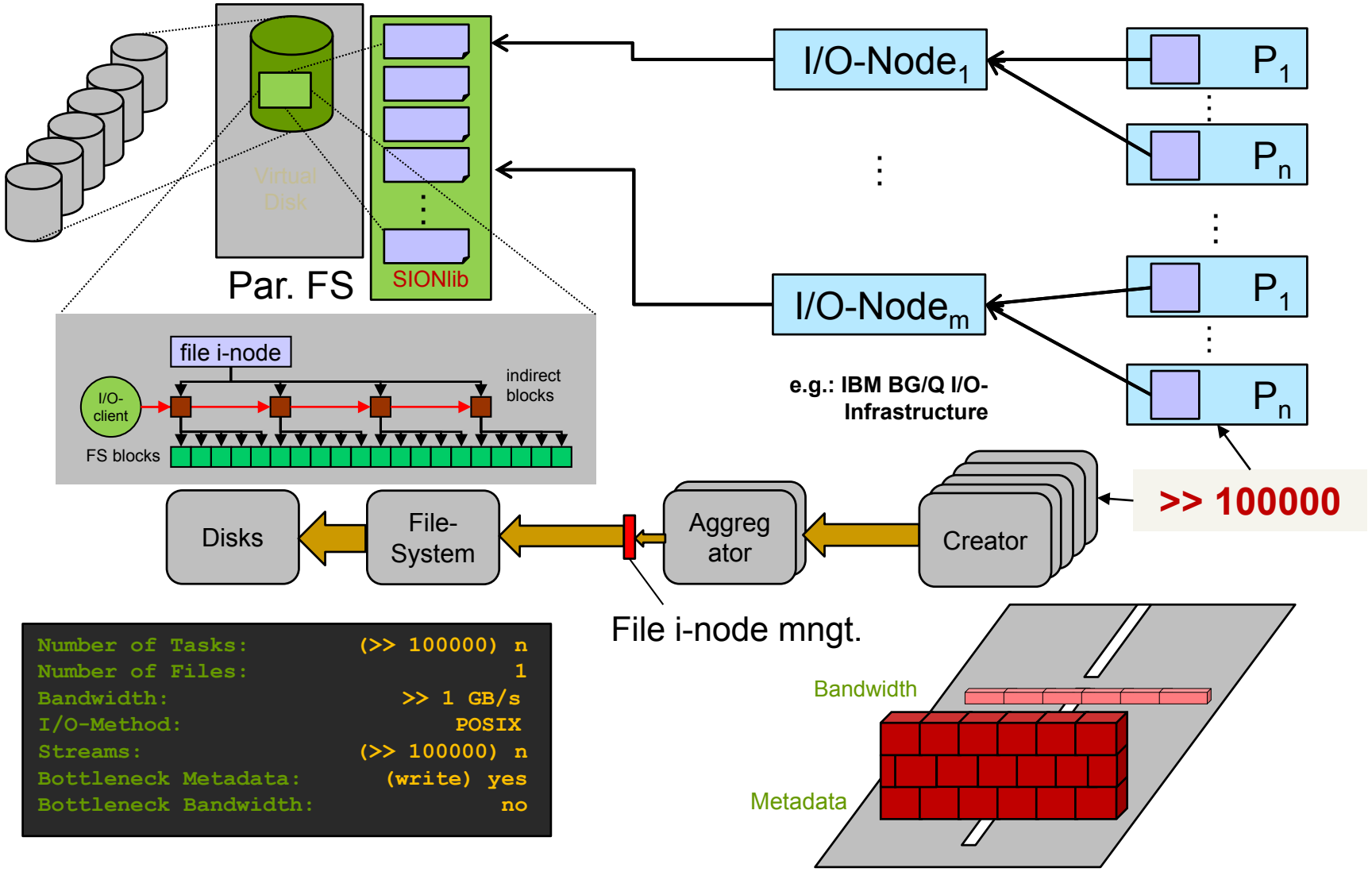
Where are the Bottlenecks?
… Increasing #Tasks … SIONlib File …



Number of Tasks:	(>> 10000) n
Number of Files:	1
Bandwidth:	>> 1 GB/s
I/O-Method:	POSIX
Streams:	(>> 10000) n
Bottleneck Metadata:	no
Bottleneck Bandwidth:	no

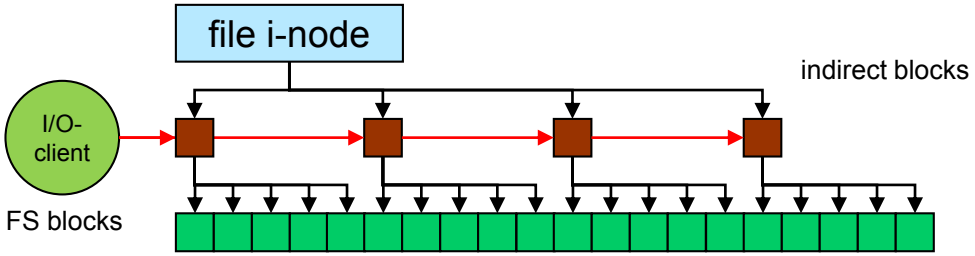


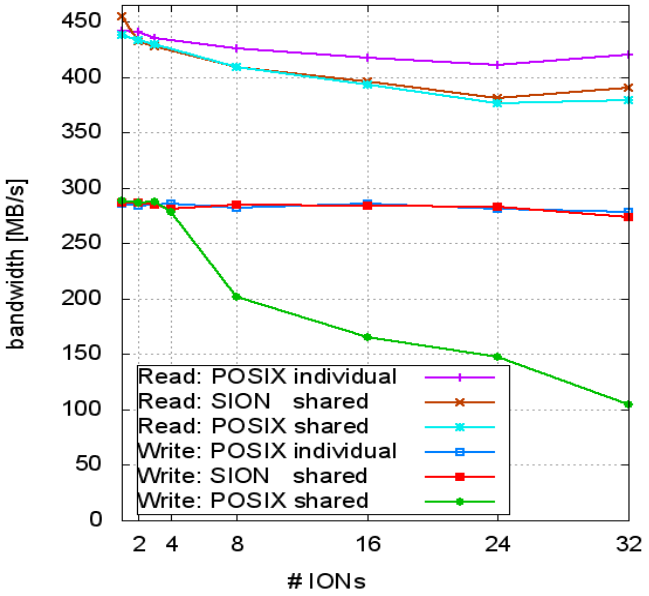
Where are the Bottlenecks?
… Increasing #Tasks further … File Metadata …



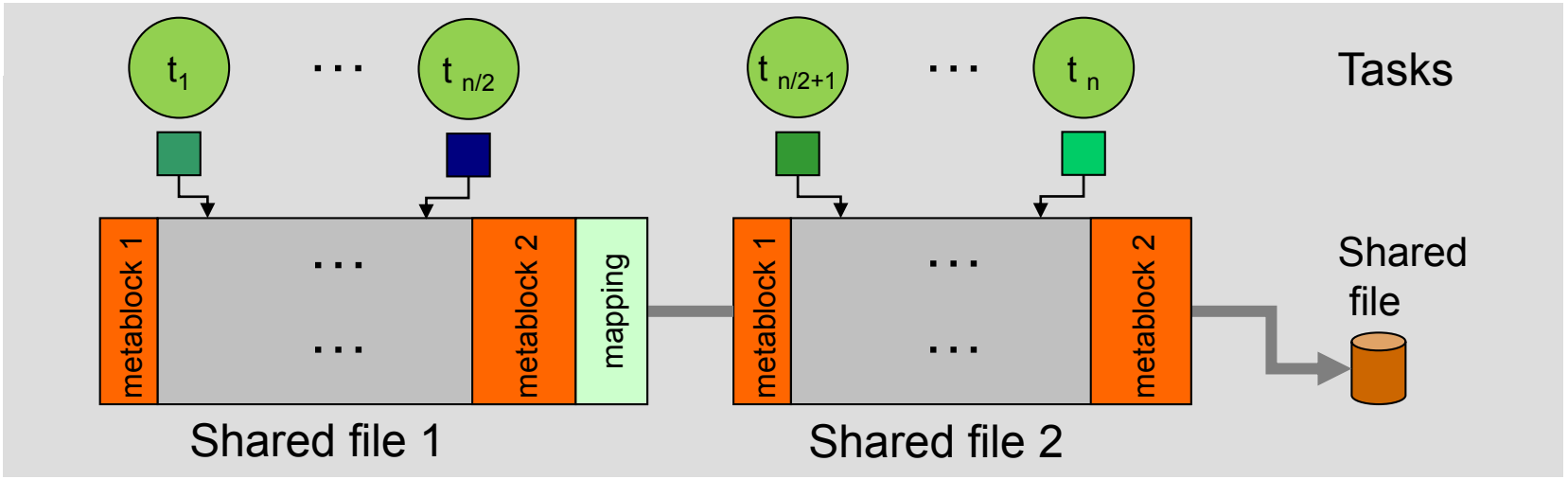
File Format: Multiple Physical Files

- Variable number of underlying physical files
- Bandwidth degradation on GPFS by using a single shared files

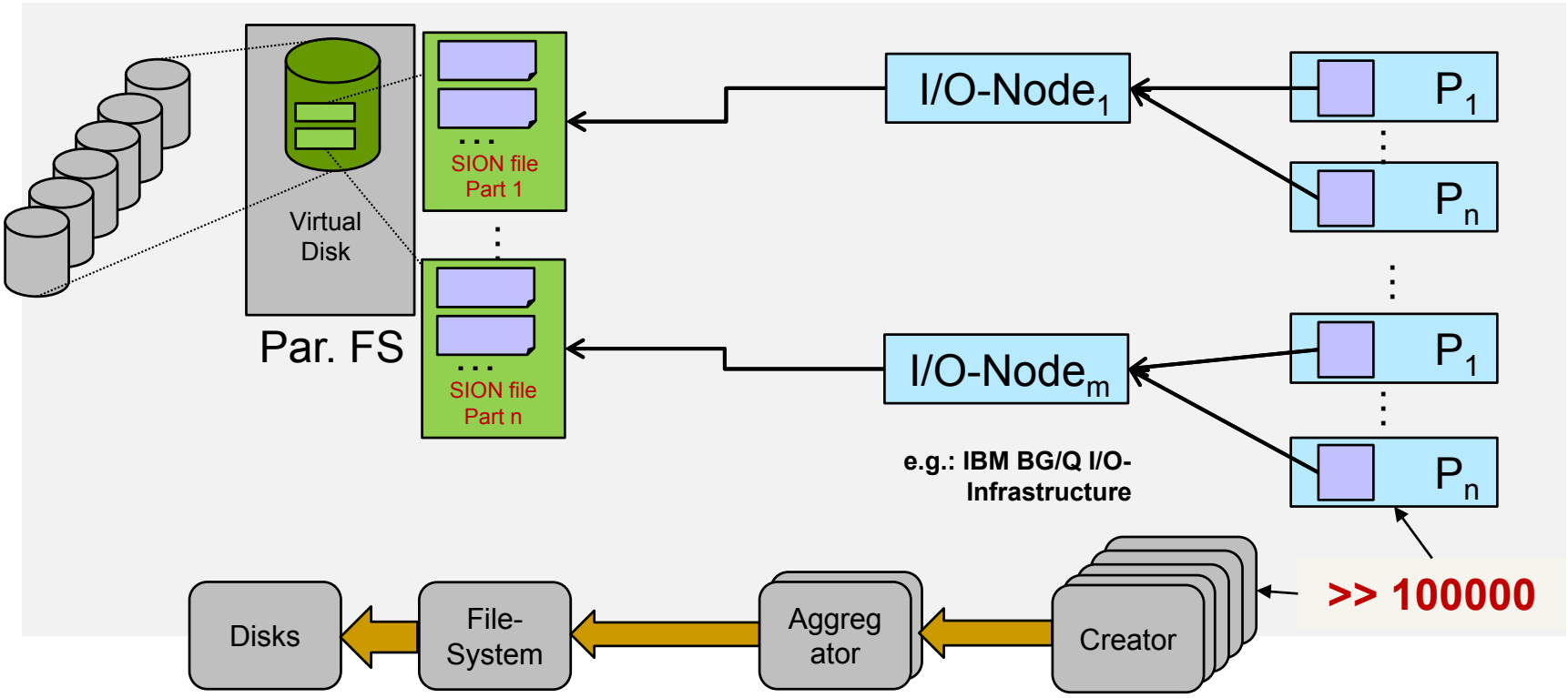




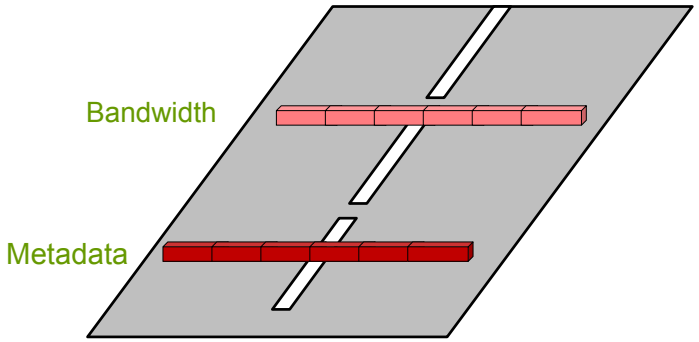
Jugene: Bandwidth per ION, comparison individual files (POSIX), one file per ION (SION) and one shared file (POSIX)



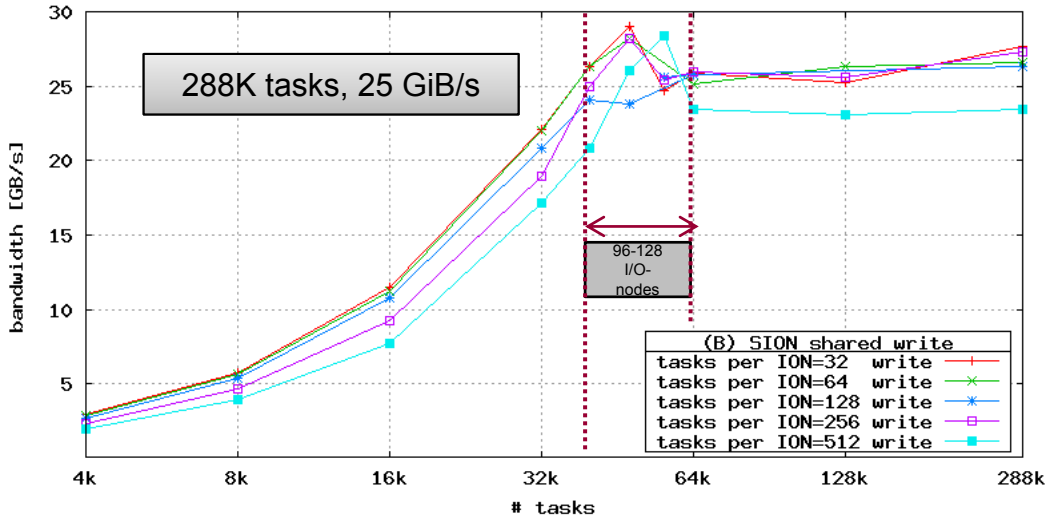
Any more Bottlenecks?
… Increasing #Tasks further … Multi-Shared-File



Number of Tasks:	(>> 100000) n
Number of Files:	1
Bandwidth:	>> 1 GB/s
I/O-Method:	POSIX
Streams:	(>> 100000) n
Bottleneck Metadata:	no
Bottleneck Bandwidth:	no

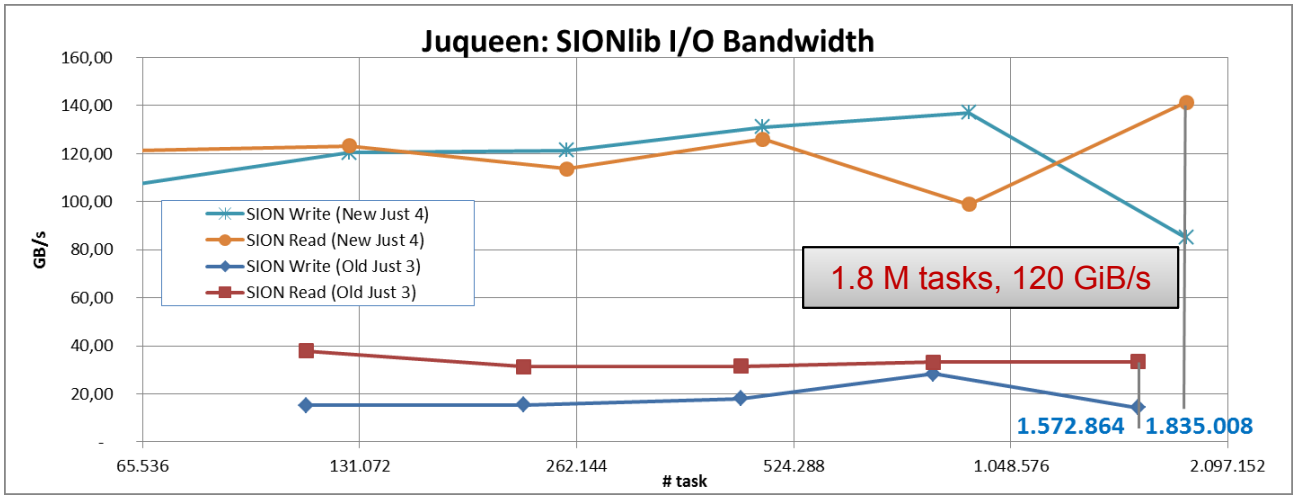


SIONlib: Scaling to Large # of Tasks

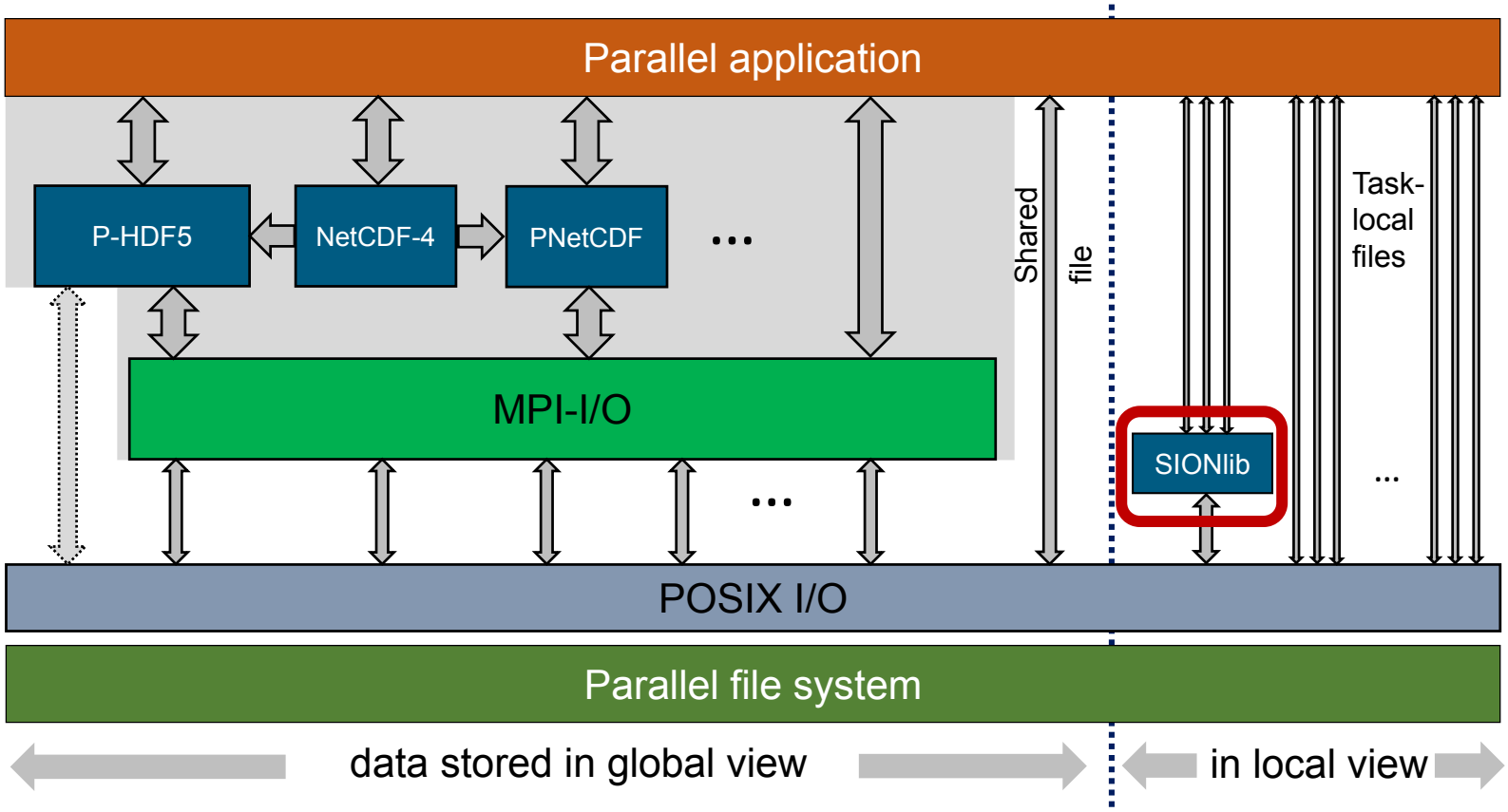


***JUGENE**: Total bandwidth (write), one file per I/O-node (ION), varying the number of tasks doing the I/O*

***JUQUEEN**: Total bandwidth (write/read) , one file per I/O-bridge (IOB), Old (Just3) vs. New (Just4, GSS) GPFS file system*



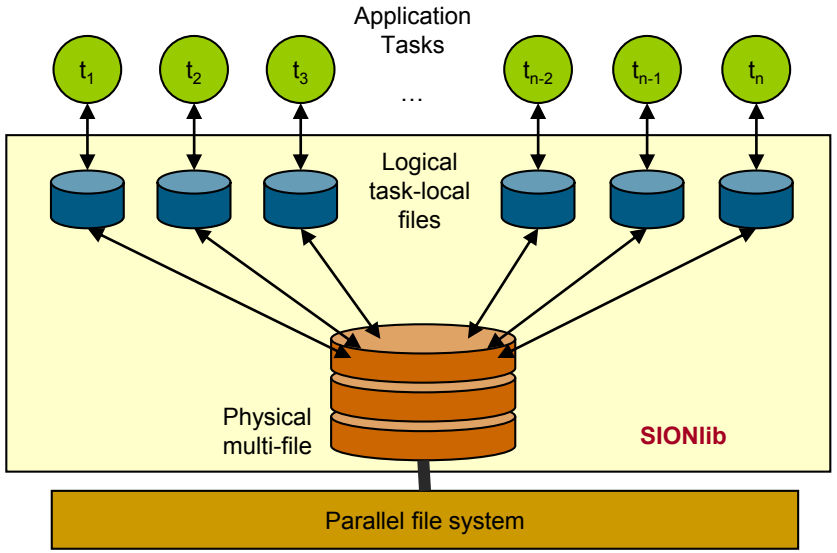
SIONlib in the Parallel I/O Software Stack



Introduction to SIONlib

SIONlib: *Scalable I/O library for parallel access to task-local files*

- Collective I/O to binary shared files
- Logical task-local view to data
- Write and read of binary stream-data
- Metadata header/footer included in file
- Collective open/close, independent read/write



- Read/write: POSIX or ANSI-C calls
- Support for MPI, OpenMP, MPI+OpenMP
- C, C++, and Fortran-wrapper (alpha of Python wrapper)
- Optimized for large processor numbers

Parallel I/O for Large Scale Application

JÜLICH FORSCHUNGSZENTRUM

- External Formats:
  - Exchange data with others → portability
  - Pre- and post-processing on other systems (workflow)
  - Store data without system-dependent structure (e.g. number of tasks)
  - Archive data (long-term readable and self-describing formats)
- Internal Formats:
  - Scratch files, Restart files
  - Fastest I/O has priority over portability and flexibility
  - Read and write data “as-is” (memory dump)
- SIONlib supports I/O of internal formats

SIONlib in a Nutshell: Task-Local I/O

JÜLICH FORSCHUNGSZENTRUM

C

```c
/* Open */
sprintf(tmpfn, "%s.%06d", filename, my_nr);
fileptr = fopen(tmpfn, "bw", ...);
...

/* Write */
fwrite(bindata, 1, nbytes, fileptr);
...

/* Close */
fclose(fileptr);
```

- Original ANSI C version
- No collective operation, no shared files
- Data: stream of bytes

SIONlib in a Nutshell: Add SIONlib

C

```c
/* Collective Open */
nfiles = 1; chunksize = nbytes; fsblksize = -1;
sid = sion_paropen_mpi(filename, "bw",
                       &nfiles,  &chunksize ,
                       &fsblksize ,
                       MPI_COMM_WORLD, &lcomm,
                       &fileptr , ...);
/* Write */
sion_fwrite(bindata,1,nbytes,sid);

...
/* Collective Close */
sion_parclose_mpi(sid);
```

- Includes check for space in current chunk
- Parameter of fwrite: fileptr → sid

Version, Download, Installation

- Version: 1.7.1 (November 2016)
- Version: file format: 5
- Open-source license

  http://www.juelich.de/jsc/sionlib
- Installation

Shell
```
➢ configure –prefix=<INSTPATH> <--enable-debug> …
➢ make
➢ make test
➢ make install
```

- Query version of SIONlib:

Shell
```
➢ sionversion
 SIONlib Version 1.7.1 (svn_rev 2088), fileformat
version 5 (./sionversion)
```

Compiling and Linking own Application

JÜLICH FORSCHUNGSZENTRUM

- Include file:

C
```
#include <sion.h>
```

- SIONlib's installation builds multiple libraries:

Files
```
→ libsion<type><language><bit><loc>.a
  type = com (common routines)  gen (generic interface)
         ser (serial interface) mpi (MPI API)
         omp (OpenMP API)       ompi(hybrid API)
  language = (C), cxx (C++), f77 (Fortran77), f90 (F90)
  bit = 64, 32 loc = fe (Frontend), be (Backend)
```

- **sionconfig**: prints set of options needed for given combination

Shell
```
➢ sionconfig <options>
  (--com|--ser|--omp|--mpi|--ompi|--gen) --> select SIONlib API
  (--cflags|--libs|--path)     --> select output of sionconfig
  [--32|--64]                  --> Precision
  [--be] [--fe] [--mic]        --> Cross compile
  [--gcc]                      --> for GCC Compiler
  [--c|--f77|--f90|--cxx]      --> Language selection
```

- Example Makefile:

File
```
LDFLAGS += `sionconfig --libs --mpi --be --64`
CFLAGS  += `sionconfig --cflags --mpi --be --64`
```

SIONlib Datatypes

- Only used for parameters for SIONlib function calls
- Data written to or read from file is a byte stream and does not need to be declared by special data types

C `sion_int32`

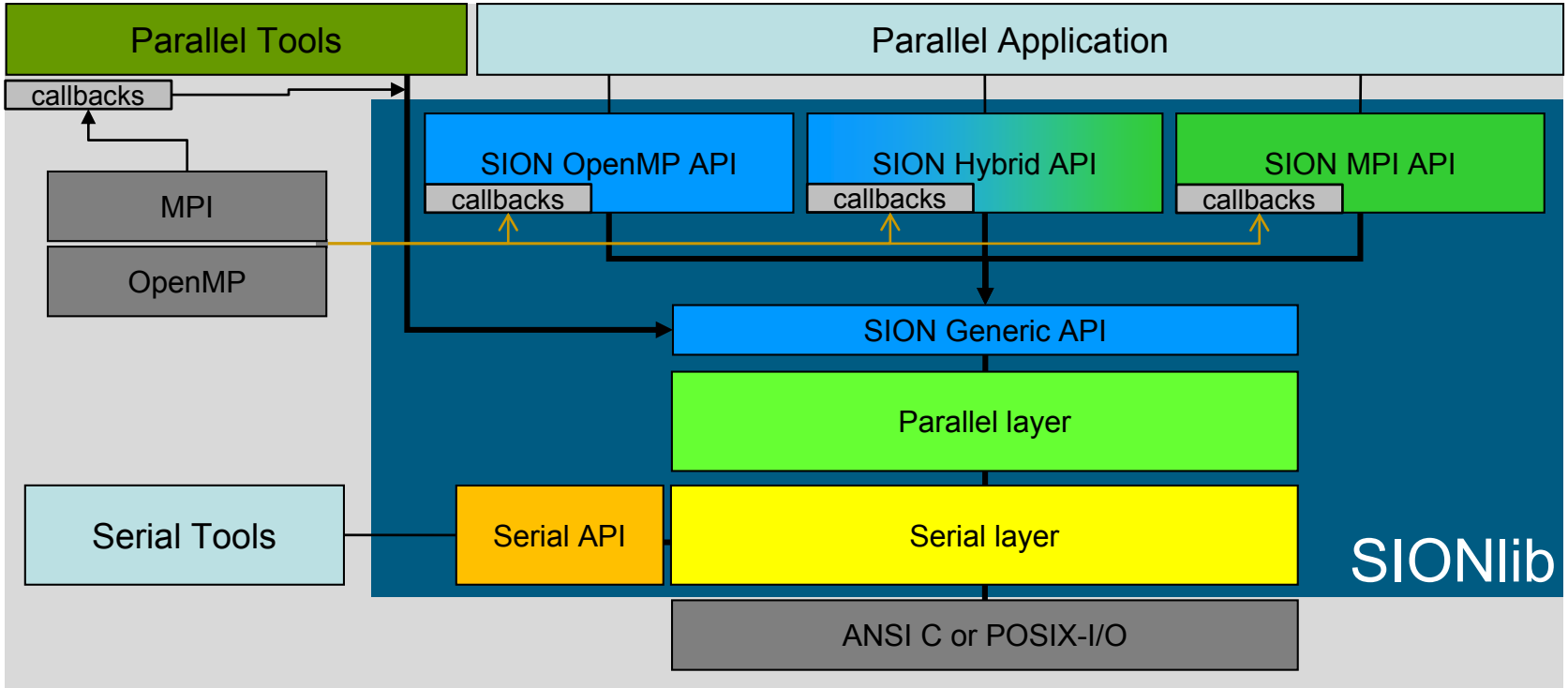
  - 4-byte signed integer (C)
  - INTEGER*4 (Fortran)

C `sion_int64`

  - 8-byte signed integer (C)
  - INTEGER*8 (Fortran)
  - Typically used for all parameters that are used to describe or to compute file positions

SIONlib Architecture (version >= 1.6)

- Implementation of different interfaces on top of generic API
- Separates the communication specific code from the general communication patterns
- Improves maintainability and ease of future development
- Available in SIONlib version 1.6

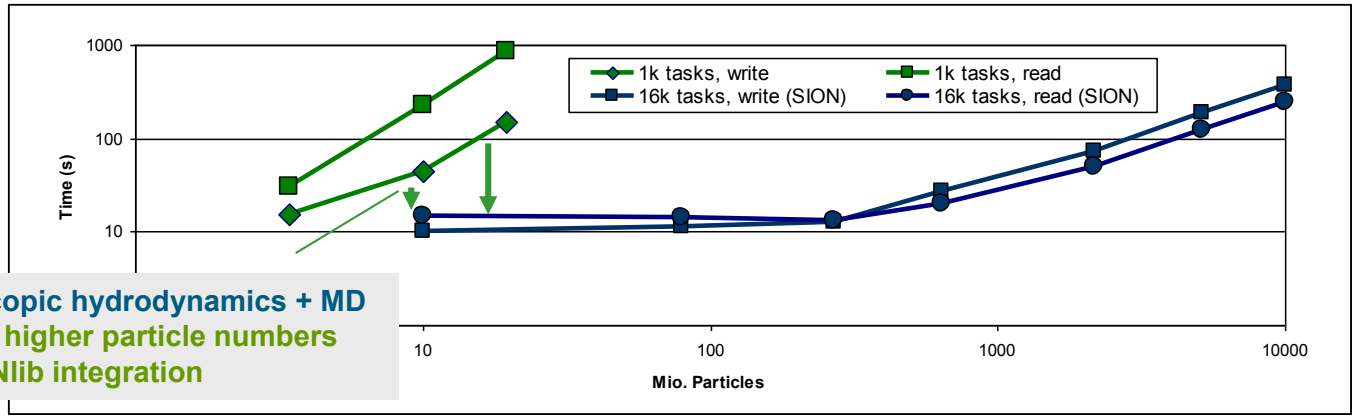


SIONlib: Applications

Applications

DUNE-ISTL (Multigrid solver, Univ. Heidelberg) **ITM** (Fusion-community),
LBM (Fluid flow/mass transport, Univ. Marburg), **PSC** (particle-in-cell code),
OSIRIS (Fully-explicit particle-in-cell code), **PEPC** (Pretty Efficient Parallel C. Solver)
Profasi: (Protein folding and aggr. simulator) **NEST** (Human Brain Simulation)

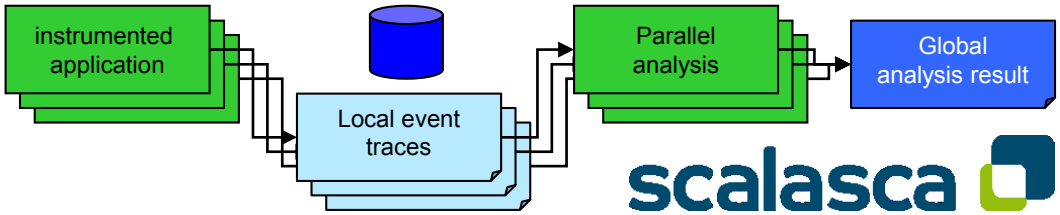
MP2C:



MP2C: Mesoscopic hydrodynamics + MD
Speedup and higher particle numbers through SIONlib integration

Tools/Projects

Scalasca: Performance Analysis



Score-P: Scalable Performance Measurement Infrastructure for Parallel Codes

DEEP-ER: Adaption to new platform and parallelization paradigm

Outline

- Introduction
- Interface
  - API Overview
  - General Parameters
  - Open/Close (Parallel)
  - Open/Close (Serial)
  - Read/Write
  - Get Information
  - Seek, Utility Functions
- Example
- Tools
- Exercises

SIONlib: Parallel Open/Close

- Parallel interface, using MPI

```
C  sion_paropen_mpi(), sion_parclose_mpi()
```

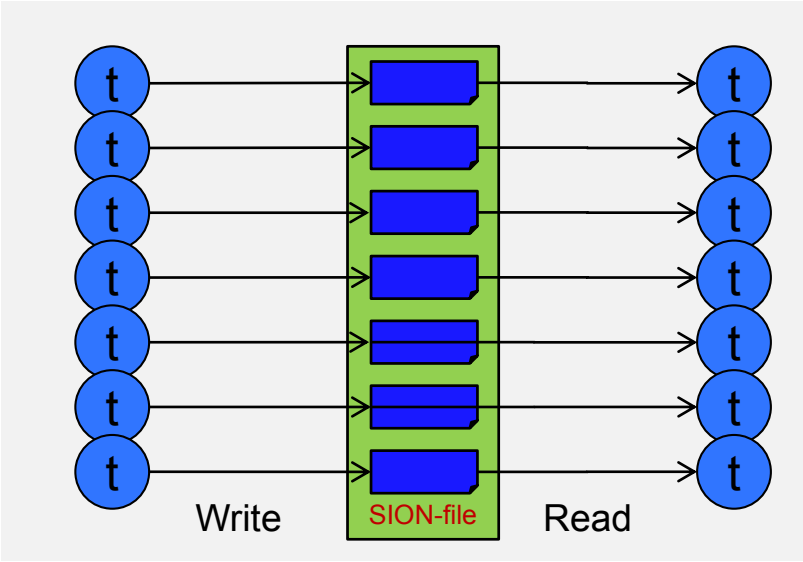
- Parallel interface, using OpenMP

```
C  sion_paropen_omp(), sion_parclose_omp()
```

- Parallel interface, using MPI+OpenMPI

```
C  sion_paropen_ompi(), sion_parclose_ompi()
```

- SIONlib data format does not distinguish data chunks of MPI tasks and OpenMP threads
  → *SIONlib task*



SIONlib: Parallel Mapped Open/Close

- Parallel interface, using MPI

```
sion_paropen_mapped_mpi(), sion_parclose_mapped_mpi()
```

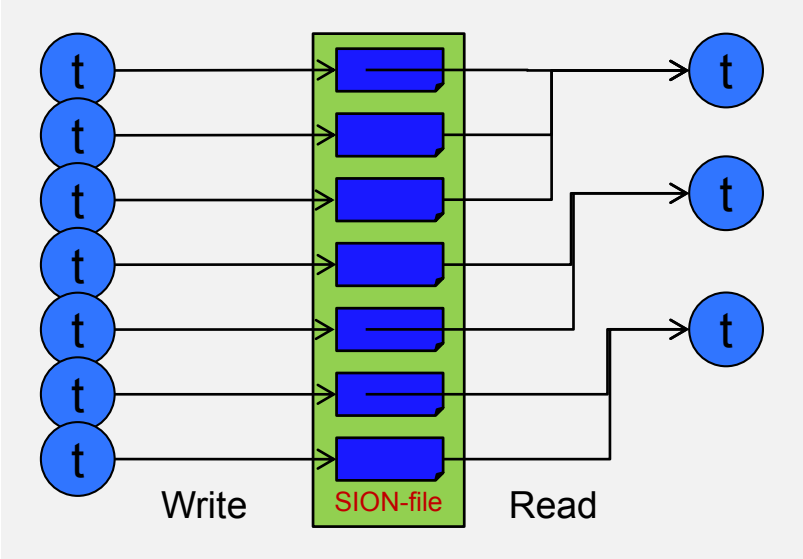
- Parallel interface, using OpenMP

```
sion_paropen_mapped_omp(), sion_parclose_mapped_omp()
```

- Parallel interface, using MPI+OpenMPI

```
sion_paropen_mapped_ompi(), sion_parclose_mapped_ompi()
```

- Use cases:
  - Restart application on a different number of tasks
  - Post-processing of data with small parallel tool



SIONlib: Serial Open/Close

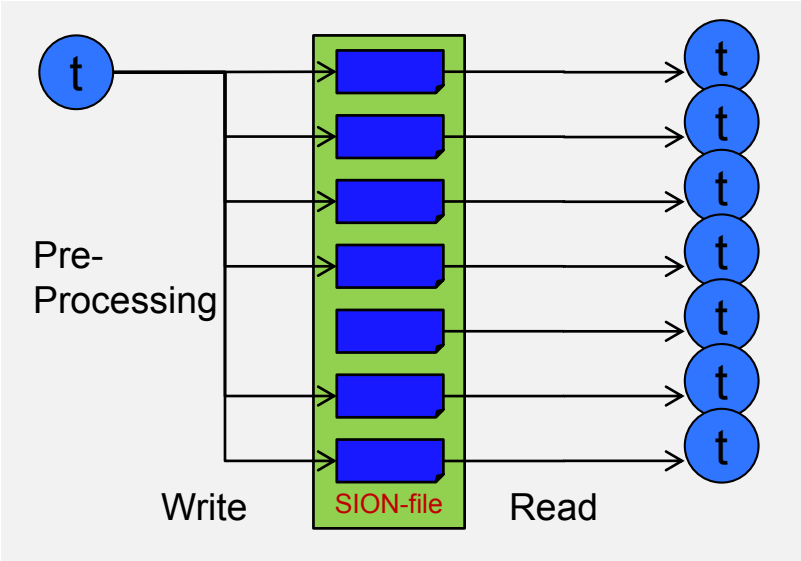
- Serial interface, accessing data of all SIONlib tasks in container

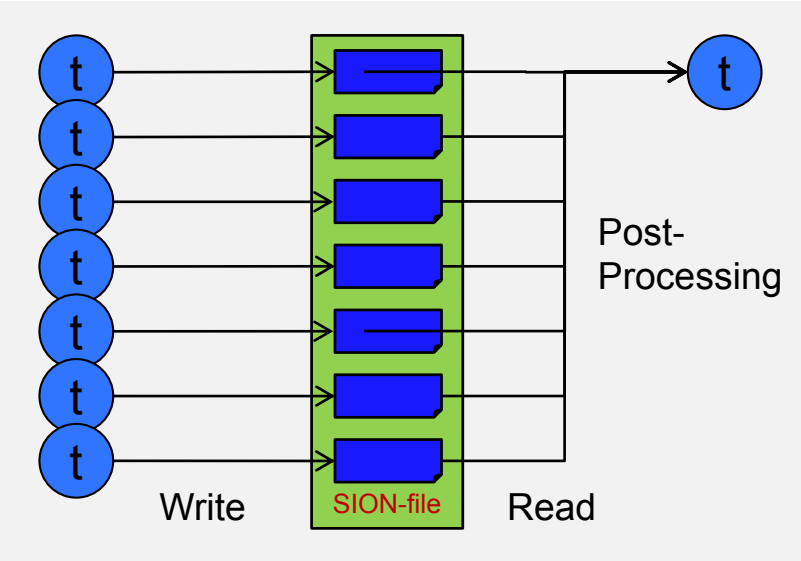
```c
sion_open(), sion_close()
```

- Serial interface, accessing data of one SIONlib task

```c
sion_open_rank(), sion_close()
```

- Use case: pre- and post-processing tools, converters





SIONlib: Write and Read Data

- Wrapper function for writing (internal checks, move to next chunk, …)

```
sion_fwrite()
```

```
sion_coll_fwrite()
```

- Wrapper function for reading (internal check on end of file)

```
sion_fread()
```

```
sion_coll_fread()
```

- Ansi-C and POSIX calls for writing and reading will be removed in future versions

- Utility functions
  - Read: Check end of file:

```
sion_feof()
```

  - Write: Check space in chunk:

```
sion_ensure_free_space()
```

SIONlib: Key-Value Container

- Open file with `sion_paropen…(filename, "…,keyval=inline" …)`
- Writing one key-value pair to current chunk
- Reading data of one key from file

C `sion_fwrite_key()`

C `sion_fread_key()`

- Utility functions
  - Iterator over all keys:

C `sion_fread_key_iterator_reset()`

C `sion_fread_key_iterator_next()`

  - Seek key and position in data:

C `sion_seek_key()`

  - Full scan of metadata:
    → all keys and positions in memory

C `sion_key_full_scan()`

SIONlib: Tool-Support

- Duplicate SIONlib structure
  - e.g. reading in parallel from multiple threads data of one SIONlib task
- Reopen file in parallel mode to adjust chunk sizes

```c
sion_dup()
```

```c
sion_dedup()
```

```c
sion_reinit_mpi()
```

SIONlib: Get Information I

- Byte order: big(1) or little(0) endian
  - Current endianness — C `sion_get_endianness()`
  - File endianness — C `sion_get_file_endianness()`
  - Swap needed — C `sion_endianness_swap_needed()`
- File state
  - Total number of bytes written/task — C `sion_get_bytes_written()`
  - Total number of bytes read/task — C `sion_get_bytes_read()`
  - Remaining bytes in chunk — C `sion_bytes_avail_in_chunk()`
  - Absolute byte-position in file — C `sion_get_position()`

SIONlib: Get Information II

- Multiple physical files:
  - Get mapping (global task-> file,local tasks) `sion_get_mapping()`
  - Number of files `sion_get_number_of_files()`
  - Number of file for this task `sion_get_filenumber()`
- Serial Mode: get information about all tasks:
  - Pointer to internal arrays `sion_get_locations()`
  - Open Mode (serial or parallel) `sion_is_serial_opened()`
- Thread Safety `sion_is_thread_safe()`

SIONlib: Seek, Utility functions

- Change position in SION file:
  - Move to another position (recommended) Utilities — `C sion_seek()`
  - Change endianness of data in memory — `C sion_swap()`
  - Wrapper for POSIX stat(), supporting large file support (file existence) — `C _sion_file_stat_file()`
  - SIONlib Version — `C sion_get_version()`
    - → Main version, sub-version, patch level, file format version

SIONlib: Header File & Return Codes

- Header file (C, C++ only)

C
```
#include <sion.h>
```

  - **Important**: include MPI first

F
```
use sion_f90_mpi
```

- Return codes
  - C, C++: C-preprocessor directives, defined in `sion.h` (`sion_const.h`)

C
```
#define SION_SUCCESS 1
#define SION_NOT_SUCCESS 0
```

  - Fortran: 1 → success, 0 → not success
  - Exception: user-definable callback-routines of generic interface (0 → success, 1 → not success)

SIONlib Parameters of Open Calls I

fname: file name

- Character string describing path and file name
- Will not be extended by SION-specific suffix
- In general multiple physical files are generated
- First file: ***filename***
- All other files: **filename** + "." + 6-digit-number (000001 ...)
- All commands and function calls use the base name

fmode: file mode

- Must specify at least one of the following

`r,rb,br`	(read block), open existing SION file for reading
`w,wb,bw`	(write block), create a new SION file, open for write; overwrite if existing
`posix`	use internally POSIX interface for file access, otherwise ANSI-C

`keyval=(default|inline|unknown)`: key-value container

SIONlib Parameters of Open Calls II

sid: SIONlib file descriptor

C
```
sid = sion_paraopen_mpi(…)
```

F
```
      FSION_PAROPEN_MPI(…,sid)
```

- Unique integer value, referring internally to data structure
- Associated to SION file (internal file handle)
- Allows multiple simultaneously opened files
- C: return code, Fortran: last parameter of open call
- Integer file handle for Fortran necessary

chunksize: size of data per task

- Pointer of type: `sion_int64*` (C), `Integer*8` (Fortran)
- Size of data in bytes written by this tasks (maximum size of single `sion_fwrite` call)
- May be different for each task and must be set if open for writing
- Will be increased internally to the next multiple of the file system block size

SIONlib Parameters of Open Calls III

fsblksize: file system block size

- Size of file system block in bytes
- Read-mode: file system block size at write time
- Write-mode: automatically detected by SIONlib if set to -1

fileptr: ANSI-C file pointer

- Can be replaced by NULL pointer if not needed (recommended)
- Will be removed in future versions
- Only needed if wrapper functions for writing and reading are not used
- Not available in Fortran

newfname:

- File name of physical file assigned to this task

Additional Parameters of sion_paropen_mpi

gComm: MPI Communicator

- Call is collective over all tasks of this communicator
- Each task gets assigned one chunk of SION file
- Read: number of tasks must be equivalent to number tasks written to SION file

lComm: MPI Communicator

- Tasks of the same communicator are writing to the same physical file
- `MPI_Comm_null` if not specified

numfiles: Number of physical files

- If using lComm: set numfiles=-1
- Read-mode: parameters will be set by open call

globalrank:

- Rank of task in global communicator gComm

Collective Open (MPI)

JÜLICH FORSCHUNGSZENTRUM

C

```
int sion_paropen_mpi (char  *fname, const char *file_mode,
                      int *numFiles,
                   MPI_Comm gComm,   MPI_Comm *lComm,
                 sion_int64 *chunksize,
                 sion_int32 *fsblksize,
                      int *globalrank,
                     FILE **fileptr, char **newfname);
```

Fortran

```
FSION_PAROPEN_MPI (FNAME, FILE_MODE, NUMFILES,
                   GCOMM, LCOMM, CHUNKSIZE, FSBLKSIZE,
                   GLOBALRANK, NEWFNAME, SID)
CHARACTER*(*) FNAME, FILE_MODE, NEWFNAME
INTEGER       NUMFILES, FSBLKSIZE, GLOBALRANK, SID
INTEGER       GCOMM, LCOMM
INTEGER*8     CHUNKSIZE
```

- Open a SION file in parallel for reading or writing data
- Collective call, called by each task at the same time
- Accesses one or more physical files of a logical SION file
- Parameters are passed “by reference” to pass back information in read open mode

Collective Open (OpenMP)

C

```
int sion_paropen_omp (char  *fname, const char *file_mode,
                      sion_int64 *chunksize,
                      sion_int32 *fsblksize,
                             int *globalrank,
                          FILE **fileptr, char **newfname);
```

Fortran

```
FSION_PAROPEN_OMP (FNAME, FILE_MODE, NUMFILES,
                   CHUNKSIZE, FSBLKSIZE,
                   GLOBALRANK, NEWFNAME, SID)
CHARACTER*(*) FNAME, FILE_MODE, NEWFNAME
INTEGER       NUMFILES, FSBLKSIZE, GLOBALRANK, SID
INTEGER*8     CHUNKSIZE
```

- Open a SION file in parallel for writing or reading data
- Collective call, has to be called inside a parallel region
- SION file consists of only one physical file
- Parameters are passed “by reference” to pass back information in read open mode
- Thread-number: `globalrank`

Collective Open (MPI+OpenMP)

JÜLICH FORSCHUNGSZENTRUM

C

```
int sion_paropen_ompi(char  *fname, const char *file_mode,
                      int *numFiles,
                   MPI_Comm gComm,   MPI_Comm *lComm,
                 sion_int64 *chunksize,
                 sion_int32 *fsblksize,
                      int *globalrank,
                    FILE **fileptr, char **newfname);
```

Fortran

```
FSION_PAROPEN_OMPI(FNAME, FILE_MODE, NUMFILES,
                   GCOMM, LCOMM, CHUNKSIZE, FSBLKSIZE,
                   GLOBALRANK, NEWFNAME, SID)
CHARACTER*(*) FNAME, FILE_MODE, NEWFNAME
INTEGER       NUMFILES, FSBLKSIZE, GLOBALRANK, SID
INTEGER       GCOMM, LCOMM
INTEGER*8     CHUNKSIZE
```

- Open a SION file in parallel for reading or writing data
- Collective call, call from each task/thread at the same time, has to be called inside a parallel region
- Accesses one or more physical files of a logical SION file
- Parameters are passed “by reference” to pass back information in read open mode

Collective Close

C

```
int sion_parclose_mpi(int sid)
int sion_parclose_omp(int sid)
int sion_parclose_ompi(int sid)
```

F

```
FSION_PARCLOSE_MPI (SID, IERR)
FSION_PARCLOSE_OMP (SID, IERR)
FSION_PARCLOSE_OMPI (SID, IERR)
INTEGER SID, IERR
```

- Closes a SION file in parallel on all tasks/threads
- Collective call, called by each task/thread at the same time
- Metadata will be collected from each tasks
- Metadata blocks of SION file will be written in this call
- Currently no fault tolerant handling of the metadata, call has to executed successfully

Serial Open

C

```
int sion_open (char         *fname,
               const char   *file_mode,
               int          *ntasks,   int *nfiles,
               sion_int64 **chunksizes,
               sion_int32   *fsblksize,
               int **globalranks,  FILE **fileptr);
```

Fortran

```
FSION_OPEN (FNAME, FILE_MODE, NTASKS, NUMFILES,
            CHUNKSIZES, FSBLKSIZE, GLOBALRANKS, SID)
CHARACTER*(*) FNAME, FILE_MODE
INTEGER       NUMFILES, NTASKS, FSBLKSIZE, SID
INTEGER       GLOBALRANKS(NTASKS)
INTEGER*8     CHUNKSIZES(NTASKS)
```

- Open a SION file in serial mode
- All chunks of all tasks can be selected, via `sion_seek`
- Multiple physical files can be handled
- Designed for serial pre- and post-processing tools
- Reads all metadata of all tasks into memory

Serial Open of one Rank

C

```
int sion_open_rank (char         *fname,
                    const char   *file_mode,
                    sion_int64   *chunksize,
                    sion_int32   *fsblksize,
                    int          *rank,
                    FILE        **fileptr);
```

Fortran

```
FSION_OPEN_RANK (FNAME, FILE_MODE, CHUNKSIZE,
                 FSBLKSIZE,RANK, SID)
CHARACTER*(*) FNAME, FILE_MODE
INTEGER       RANK, FSBLKSIZE, SID
INTEGER*8     CHUNKSIZE
```

- Open SION file for one rank in serial mode
- Multiple physical files can be handled
- Designed for parallel program if collective open/close is not possible
- Reads only metadata of this task into memory

Serial Close

C
```
int sion_close(int sid)
```

F
```
FSION_CLOSE(SID, IERR)
INTEGER SID, IERR
```

- Closes a SION file in serial mode
- Metadata blocks of SION file will be written in this call
- Currently no fault tolerant handling of the metadata, call has to be executed successfully

Read Data

C

```
size_t sion_fread (void  *data,
                   size_t size,
                   size_t nmemb,
                   int    sid);
```

F

```
FSION_READ (DATA, SIZE, NMEMB, SID, IERR)
INTEGER SIZE, NMEMB, SID, IERR
```

- Read `size*nmemb` bytes from current position in chunk
- Internally this function reads in a while loop until all data is read from file
- Reading more data than stored in one chunk is possible with this wrapper
- Returns number of elements read

End of File

C
```
int sion_feof(int sid);
```

F
```
FSION_FEOF (SID, IERR)
INTEGER SID, IERR
```

- Equivalent to POSIX feof which cannot be used for shared SION files
- Internally this function flushes all buffer and checks current positions against chunk boundaries
- Moves file pointer to next chunk if end of current chunk is reached
- The function is a task-local function, which can be called independently from other MPI tasks.
- Returns 1 if pointer is behind last byte of data for this task

Write Data

C

```
size_t sion_fwrite (void *data,
                    size_t size,
                    size_t nmemb,
                    int    sid);
```

F

```
FSION_WRITE (DATA, SIZE, NMEMB, SID, IERR)
INTEGER SIZE, NMEMB, SID, IERR
```

- Write `size*nmemb` chunk, beginning from current position
  - This size must not exceed the chunksize defined in open call
- Internally this function uses **`sion_ensure_free`** space to check whether there is enough space is available
- Returns number of elements written

Ensure Free Space in Chunk

C

```
int sion_ensure_free_space (int         sid,
                            sion_int64 bytes);
```

F

```
FSION_ENSURE_FREE_SPACE (SID, BYTES, IERR)
INTEGER    SID, IERR
INTEGER*8 BYTES
```

- Ensures that there is enough space available for writing
- A new chunk will be allocated if not enough space is left in the current chunk
- The function is a task-local function, which can be called independently from other MPI tasks
- The function moves the file pointer to a new position in some cases and flushes also the local file pointer
- Returns 1 if space could be ensured, there is currently no indicator if a new chunk was allocated

Get Byte Ordering (Endianness)

JÜLICH FORSCHUNGSZENTRUM

C
```
int sion_get_file_endianness (int sid);
int sion_get_endianness ();
int sion_endianness_swap_needed (int sid);
```

F
```
FSION_GET_FILE_ENDIANNESS(SID, ENDIANESS)
FSION_GET_ENDIANNESS(ENDIANESS)
FSION_ENDIANNESS_SWAP_NEEDED(SID, ENDIANESS) (TBD)
INTEGER   SID, ENDIANESS
```

- Return endianness (1 ! big endian, 0 ! little endian)
- For current file (sid), or
- For current runtime environment
- Bytes have to be reordered if:

  `sion_get_file_endianness`() != `sion_get_endianness`()
  - Convenience function `sion_endianness_swap_needed`
- Utility for reordering: see **`sion_swap`**

Seek: Change File Position

C

```
int sion_seek (int         sid,
               int         rank,
               int         chunknr,
               sion_int64 posinchunk );
```

F

```
FSION_SEEK (SID, RANK, CHUNKNUM, POSINCHUNK, IERR)
INTEGER     SID, RANK, CHUNKNUM, IERR
INTEGER*8   POSINCHUNK
```

- Sets the file pointer to a new position
- Seek parameters:
  - `rank`: rank number (0,...), or SION_CURRENT_RANK
  - `chunknum` : chunk number (0,...), or SION_CURRENT_BLK
  - `posinchunk`: position (0,...), or SION_CURRENT_POS
- In parallel write mode seeking is currently not supported
- For serial opened file please use **`sion_seek_fp`**, because physical file pointer could change.

Utility: Swap bytes

JÜLICH FORSCHUNGSZENTRUM

C

```
void sion_swap (void *target,
                void *source,
                int   size,
                int   n,
                int   aflag);
```

F

```
FSION_SWAP (TARGET, SOURCE, SIZE, N, AFLAG, IERR)
INTEGER     SIZE, N, AFLAG, IERR
```

- Perform byte-order swapping for arrays of `n` units of `size` bytes
- Bytes are swapped if and only if `aflag` ==0
- Data is copied from `source` to `target`
- In-place swapping (`target` == `source`) is allowed;
  if `target` != `source` , the buffers must not overlap
- `aflag` can be initialized as follows:
  (`sion_get_file_endianness()== sion_get_endianness()`)
  or
  (`!sion_endianness_swap_needed()`)

Outline

- Introduction
- Interface
- Example
  - Sample program writing data (C, Fortran)
  - Blue Gene/Q: using local communicator
- Tools
- Exercises

Example Code: sion_par_write (Part 1)

JÜLICH FORSCHUNGSZENTRUM

C

```c
/* SIONlib header file */
#include <sion.h>

/* SION parameters */
int         sid, numFiles, globalrank;
MPI_Comm    lComm;
sion_int64 chunksize, left, bwrote;
sion_int32 fsblksize;
char        fname[256], *newfname=NULL;

/* initialize MPI */
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

/* open parameters */
chunksize = 10; globalrank = my_rank;
numFiles = 1;   fsblksize = -1;
strcpy(fname, "parfile.sion");
```

Example Code: sion_par_write (Part 2)

JÜLICH FORSCHUNGSZENTRUM

C

```c
/* create a new file */
sid = sion_paropen_mpi(fname, "bw",     &numFiles,
                       MPI_COMM_WORLD, &lComm,
                       &chunksize,     &fsblksize,
                       &globalrank,
                       NULL, &newfname);

/* write buffer to file */
p = (char *) fname; /* filename as sample data */
sion_fwrite(p, 1, chunksize, sid);

/* close file */
sion_parclose_mpi(sid);

/* finalize MPI */
MPI_Finalize();
```

Example Code: sion_par_write (Part 1)

Fortran

```
! SION parameters
integer*8 :: chunksize
integer :: gComm,lComm,sid,globalrank,ierr
integer :: fsblksize, nfiles
character(len=255) :: filename = 'parfile.sion'
character(len=255) :: newfname
integer*4,dimension(:),allocatable :: buffer

! MPI initialization
call MPI_Init(ierr)
call MPI_Comm_size(MPI_COMM_WORLD,nranks,ierr)
call MPI_Comm_rank(MPI_COMM_WORLD,my_rank,ierr)

! open parameters
gcomm=MPI_COMM_WORLD
globalrank=my_rank
fsblksize=-1
chunksize=10
nfiles=1
```

Example Code: sion_par_write (Part 2)

Fortran

```
! create a new file
call fsion_paropen_mpi(trim(filename),'bw',    &
                       nfiles, gComm,          &
                       lComm, chunksize,       &
                       fsblksize, globalrank, &
                       newfname, sid)

! write buffer to file
buffer = …
call fsion_write(buffer,4,veclen,sid,ierr)

! close file
call fsion_parclose_mpi(sid,ierr)

! MPI finalization
call MPI_Finalize(ierr)
```

Blue Gene/Q I/O-node Task Mapping

JÜLICH FORSCHUNGSZENTRUM

C

```c
/* communicator consists of all task working with
   the same I/O-node */
MPI_Comm      commSame;
MPIX_Pset_same_comm_create(&commSame);
MPI_Comm_size(commSame, &sizeSame);
MPI_Comm_rank(commSame, &rankSame);
numFiles = -1;

/* create a new file */
sid = sion_paropen_mpi(fname, "bw", &numFiles,
                       MPI_COMM_WORLD, &commSame,
                       &chunksize, &fsblksize,
                       &globalrank, &fileptr, &newfname);
```

Outline

- Introduction
- Interface
- Example
- Tools
  - Managing SION files
  - Parallel Benchmark
- Exercises

Tools: sionsplit

JÜLICH FORSCHUNGSZENTRUM

Command

```
➢ sionsplit [options] sionfile prefix

Parameters & Options:
  sionfile    Input file in SIONlib format
  prefix      directory and/or filename-prefix for
              task-local files
  [-v]        verbose mode
  [-g]        use global rank for numbering files
  [-d <num>] number of digits for filename
              generation (default 5)
```

- Generates task-local files from SION-file
- Serial tool

Tools: sioncat

Command

➢ **sioncat** [options] *sionfile*

Parameters & Options:

sionfile	Input file in SIONlib format
[-v]	verbose mode
[-t <tasknum>]	write only data of task <tasknum>
[-b <chnknum>]	write only data of chunk <chnknum>
[-o <outfile>]	file data will be written to, if not specified stdout will used

- Extract all or selected data from a SION file
- Serial tool

Tools: siondump

JÜLICH FORSCHUNGSZENTRUM

Command

➢ **siondump** [options] *sionfile*

Parameters & Options:

sionfile	Input file in SIONlib format
[-a]	print all information about all blocks
[-m]	print all mapping information
[-l]	print all sizes in number of bytes
[-k]	print key-value statistic for each task
[-K]	print key-value list for each task
[-S]	print size of internal data structure in memory
[-V]	show version number of SIONlib
[-v]	verbose mode

- Print metadata information of a SION file
- Serial tool

Tools: siondefrag

JÜLICH FORSCHUNGSZENTRUM

Command

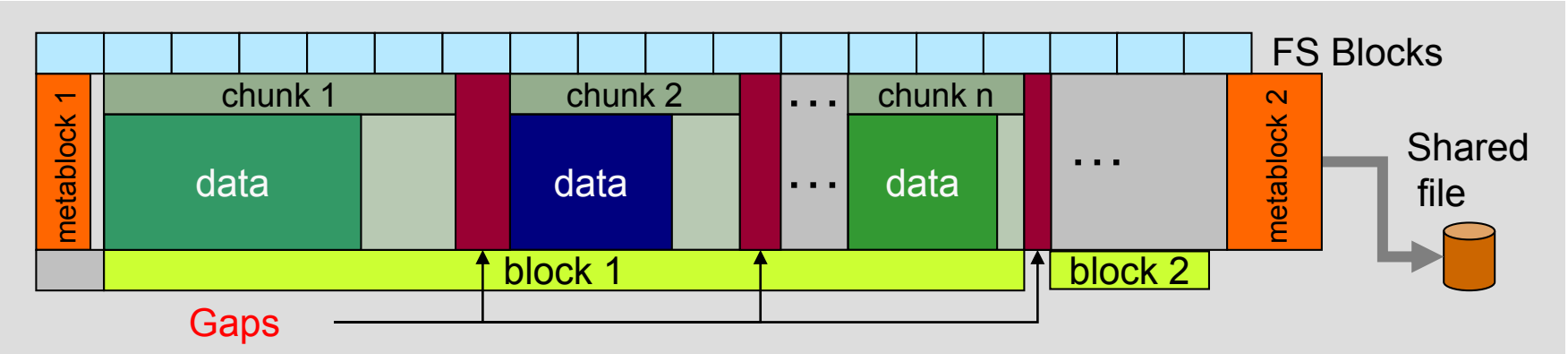
```
➢ siondefrag [options] sionfile new_sionfile

Parameters & Options:
 sionfile           Input file in SIONlib format
 new_sionfile       Output file in SIONlib format
 [-Q <fsblksize>] filessystem blocksize for new
                    sion file in MB (default: input file)
 [-q <fsblksize>] filessystem blocksize for new
                    sion file in bytes
 [-V]               show version number of SIONlib
 [-v]               verbose mode
```

- Generates new SION file from existing one
- Changes the underlying file system block size
- Can be used to remove empty gaps, caused by
  - Not completely filled chunks
  - Alignment to file system blocks
- Serial tool

Notes on Sparse Files

- In general, SION files can be sparse, i.e. the file contains unused space (Gaps).
- This usually is handled efficiently by the underlying file system using sparse files
- Transferring files using **`rsync`** with the `-S` or `--sparse` option preserves the sparsity, without this option the resulting files are dense



Tools: sionconfig

JÜLICH FORSCHUNGSZENTRUM

Command

```
➢ sionconfig [options]

Options:
 (--com|--ser|--omp|
  --mpi|--ompi|--gen)        select SIONlib API
 (--cflags|--libs|--path)    select output of sionconfig
 [--32|--64]                 Precision
 [--be] [--fe] [--mic]       Cross compile
 [--gcc]                     for GCC Compiler
 [--c|--f77|--f90|--cxx]     Language selection
 [-V]                        show version number
 [-v]                        verbose mode
```

- Returns flags and options for compiler and cinker on stdout
- Serial tool

Tools: sionversion

Command

```
➢ sionversion

→ SIONlib Version 1.7.1 (svn_rev 2088), \
    fileformat version 5 (./sionversion)
```

- Current version of SIONlib
  - Major, minor version number
  - Patch level
  - File format version
  - SVN revision number
- Serial tool

Tools: partest, Parallel I/O benchmark I

JÜLICH FORSCHUNGSZENTRUM

Command

```
➢ partest [options]

Options: (file settings)
[-f filename] (--filename[=])      data filename
[-n <numfiles>]  (--numfiles[=])   number of files
[-r <chunksize>] (--chunksize[=]) sion chunk size (*)
[-q <fsblksize>] (--fsblksize[=]) size of filesystem
                                  blocks (*)
(*) Size Formats: <d>[g,G,Gb,GB,
                      m,M,Mb,MB,
                      k,K,Kb,KB, GiB, MiB, KiB ]
…
```

- Parallel benchmark program of SIONlib
- Output-Format: Task-Local, SIONlib, MPI-I/O

Tools: partest, Parallel I/O benchmark II

JÜLICH FORSCHUNGSZENTRUM

Command

```
…
Options: (test configuration)

[-T <type>](--testtype[=]) testtype (0:SION,collective)
                                (1:SION, independent read)
                                (2:MPI IO) (3:Task-Local)
[-b <bufsize>](--bufsize[=]) blocksize written by
                                ONE fwrite (*)
[-g <totalsize>](--totalsize[=]) global total size of
                                   data written(*)
[-s <localsize>](--localsize[=]) size of local data
                                   for each task(*)
[-F <factor>] (--factor[=]) random factor
                               (0.0 to 1.0, def: 0.0)
[-R (0|1)] (--read[=])      switch read off/on
[-W (0|1|2)] (--write[=])   switch write off, on,
                                        or 2x write
```

Tools: partest, Parallel I/O benchmark III

JÜLICH FORSCHUNGSZENTRUM

Command

```
…
Options: (test specific configuration)
[-v] (--verbose[=](0|1))     verbose print info for each task
[-C] (--nochecksum[=](0|1)) suppress checksum
[-d] (--debugtask[=](0|1))  debug task 0
[-D] (--Debugtask[=](0|1))  debug task n
[-L] (--posix[=](0|1))      use POSIX calls instead of ANSI
[-M] (--collwrite[=](0|1))  use collective write if possible
[-m] (--collread[=](0|1))   use collective read if possible
[-Z <offset>](--taskoffset[=]) shift tasks numbering for
                               reading by offset to ommit
                               data caching of file-system (0)
[-O <bytes>] (--byteoffset[=]) start offset, write <bytes>
                               first before using blksize  (0)
[-j <#tasks>] (--serialized[=])serialize I/O, only I/O of
                              #tasks are running in parallel
                              (-1 -> all tasks in parallel,
                              -2 -> use transactions, def: -1)
[-X] (--unlinkfiles[=](0|1))   remove files after test
```

Tools: partest, Parallel I/O benchmark IV

JÜLICH FORSCHUNGSZENTRUM

Command

```
…
Options: (Blue Gene/L, Blue Gene/P, Blue Gene/Q)
[-P] (--bgionode[=](0|1))     order tasks by BG I/O-node
[-p <numtasks>]
       (--bgtaskperionode[=]) number of tasks per BG I/O-node
                              (default: all)

Options: (Blue Gene/L, Blue Gene/P, Blue Gene/Q)
[-w] (--hintlargeblock[=](0|1)) I BM, Large Block IO
[-Q <size>] (--hintiobufsize[=]) IBM, IO bufsize in KB
[-x] (--hintsparseacess[=](0|1)) IBM, sparse access

…
```

Outline

- Introduction
- Interface
- Example
- Tools
- Exercises
  - Simple Write/Read Test
  - Parallel Computation of Mandelbrot

Exercises on JURECA

- The solutions to the exercises can be found in
  - /work/hpclab/train001

Exercise I

Exercise 1 – Write/Read

1.1 Write

- Create a SION data set, where each task writes local data to a corresponding chunk of the SION-file
- A local vector of 10000 integers should be allocated and initialized with the task number
- Each task should write the vector to the SION data set

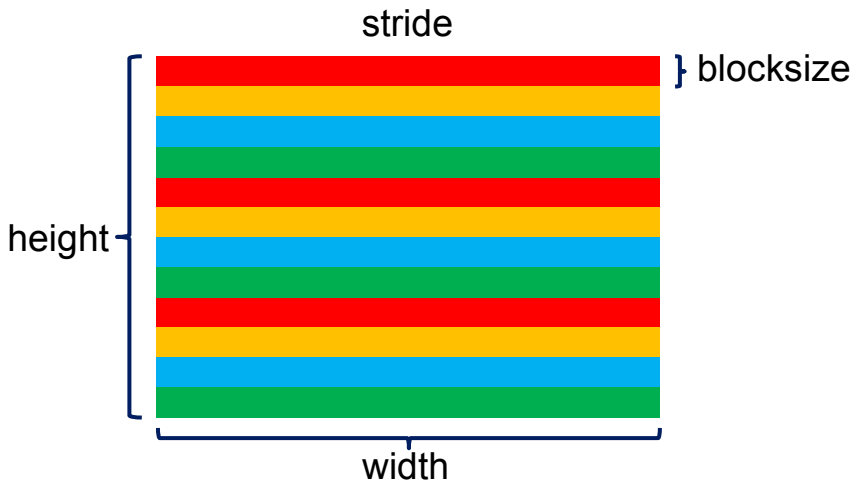
1.2 Read

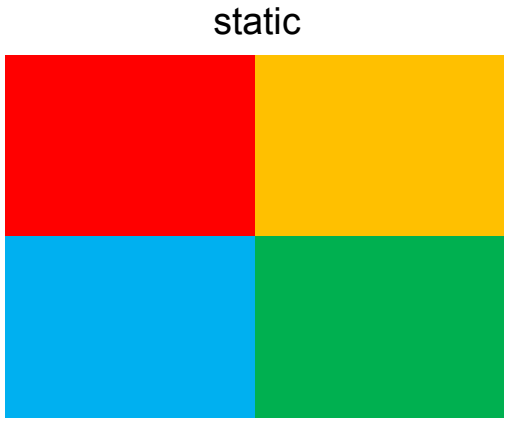
- Read the SION data of the corresponding chunk into memory
- Check if the data is consistent (task number)

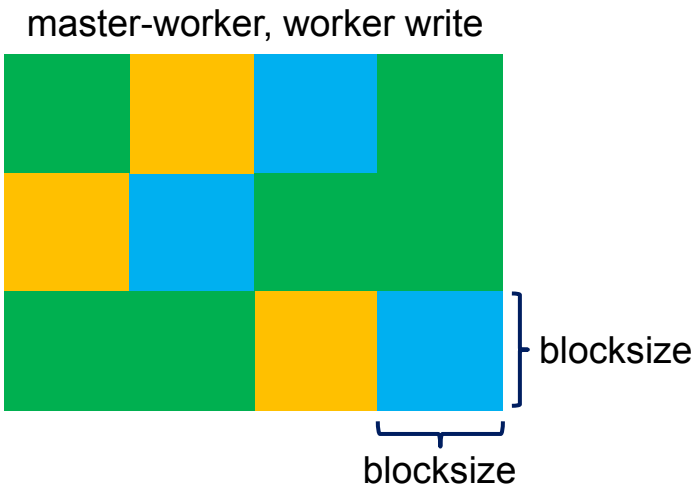
1.3 Tools

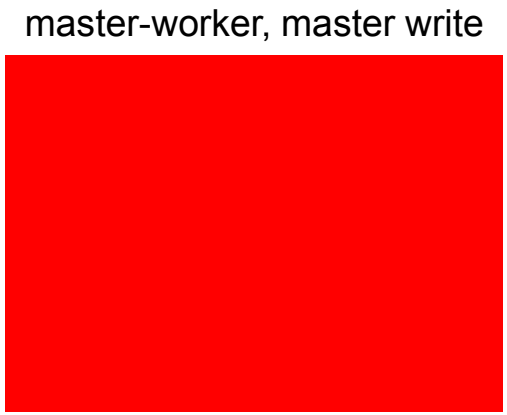
- Run siondump on the SION-file to check the metadata
- Use siondefrag to create a dense version of the SION file, and check again the metadata of the new file
- Extract the chunks of the SION file into task-local files

Exercise II: Mandelbrot set









Mandelbrot exercise API

JÜLICH FORSCHUNGSZENTRUM

C

```c
typedef struct _infostruct
{
  int type; int width; int height;
  int numprocs;
  double xmin; double xmax; double ymin; double ymax;
  int maxiter;
} _infostruct;
```

Fortran

```fortran
type :: t_infostruct
  integer :: type, width, height
  integer :: numprocs
  real :: xmin, xmax, ymin, ymax
  integer :: maxiter
end type t_infostruct
```

Mandelbrot exercise API

JÜLICH FORSCHUNGSZENTRUM

C

```
void open_sion(int *fid, _infostruct *infostruct,
               int *blocksize, int *start, int rank)
```

Fortran

```
open_sion(fid, info, blocksize, start, rank)
  integer, intent(out) :: fid
  type(t_infostruct), intent(in) :: info
  integer, dimension(2), intent(in) :: blocksize
  integer, dimension(2), intent(in) :: start
  integer, intent(in) :: rank
```

`fid`	lib specific file_id (can occurs twice if multiple ids needed)
`info`	global information structure
`blocksize`	chosen (or calculated) blocksizes (C: [y,x], Fortran: [x,y])
`start`	calculated start point (C: [y,x], Fortran: [x,y], starting at `0`)
`rank`	process MPI rank

Mandelbrot exercise API

JÜLICH FORSCHUNGSZENTRUM

C

```
void close_sion(int *fid, _infostruct *infostruct,
                int rank)
```

Fortran

```
close_sion(fid, info, rank)
  integer, intent(inout) :: fid
  type(t_infostruct), intent(in) :: info
  integer, intent(in) :: rank
```

`fid`	lib specific file_id (can occurs twice if multiple ids needed)
`info`	global information structure
`rank`	process MPI rank

Mandelbrot exercise API

JÜLICH FORSCHUNGSZENTRUM

C

```
void write_to_sion_file(
  int *fid, _infostruct *infostruct, int *iterations,
  int width, int height, int xpos, int ypos)
```

Fortran

```
write_to_sion_file(fid, info, iterations, width, height,
                   xpos, ypos)
  integer, intent(in) :: fid
  type(t_infostruct), intent(in) :: info
  integer, dimension(:), intent(in) :: iterations
  integer, intent(in) :: width
  integer, intent(in) :: height
  integer, intent(in) :: xpos
  integer, intent(in) :: ypos
```

`iterations`	data array
`width, height`	size of current data block (pixel coordinates)
`xpos, ypos`	position of current data block (pixel coordinates starting at `0`)

Mandelbrot exercise API

JÜLICH FORSCHUNGSZENTRUM

C

```
void collect_sion(
  int **iterations, int **proc_distribution,
  _infostruct *infostruct)
```

Fortran

```
collect_sion(iterations, proc_distribution, info)
  integer, dimension(:), pointer :: iterations
  integer, dimension(:), pointer :: proc_distribution
  type(t_infostruct), intent(inout) :: info
```

`iterations`	data array
`proc_distribution`	process distribution array (only in SIONlib)
`info`	global information structure