Parallel I/O for SwissTx

SFIO parallel I/O system implementation for Swiss-Tx. MPI-II I/O interface support for SFIO.

LSP DI EPFL

1.0 Introduction

1.1 Requirements for parallel I/O

1.1.1 Scalable Throughput

1.1.2 Multiple Access

We try to design the system adapted to multiple access.

1.2 Striped Files Parallel Access Solution

Cyclical distribution of logical file block by block over the set of striped files located on different disksk and different computers.

1.3 Interface

1.3.1 SFIO

1.3.2 MPI-II I/O

1.4 Network communication and Disk I/O access optimisation

This is data flow optimisation problems over the communication network and between memory and physical disk

1.4.1 Data fragmentation problem for noncollective operations

For noncollective operations, data fragmentation problem is resolved and maximaly optimized as for the communication over the network as well as for writing andreading optimization.

1.4.2 Data fragmentation problem for collective operations

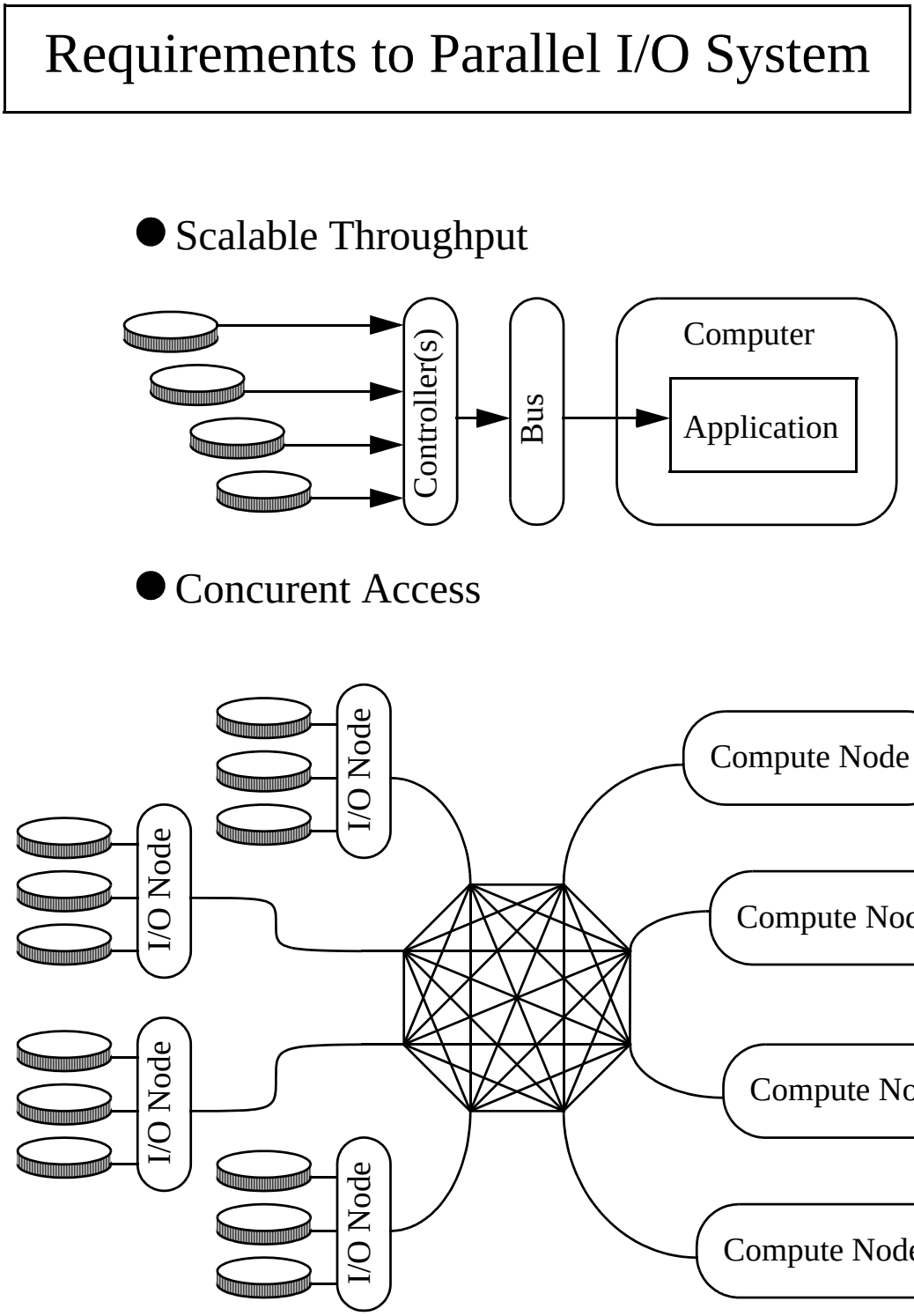
The optimization is realized for noncollective operations do his job here and resloves half of optimisation problems, but there are space of optimization specialy for the case of collective communications.

Fig. 01.

Parallel I/O for Swiss-Tx

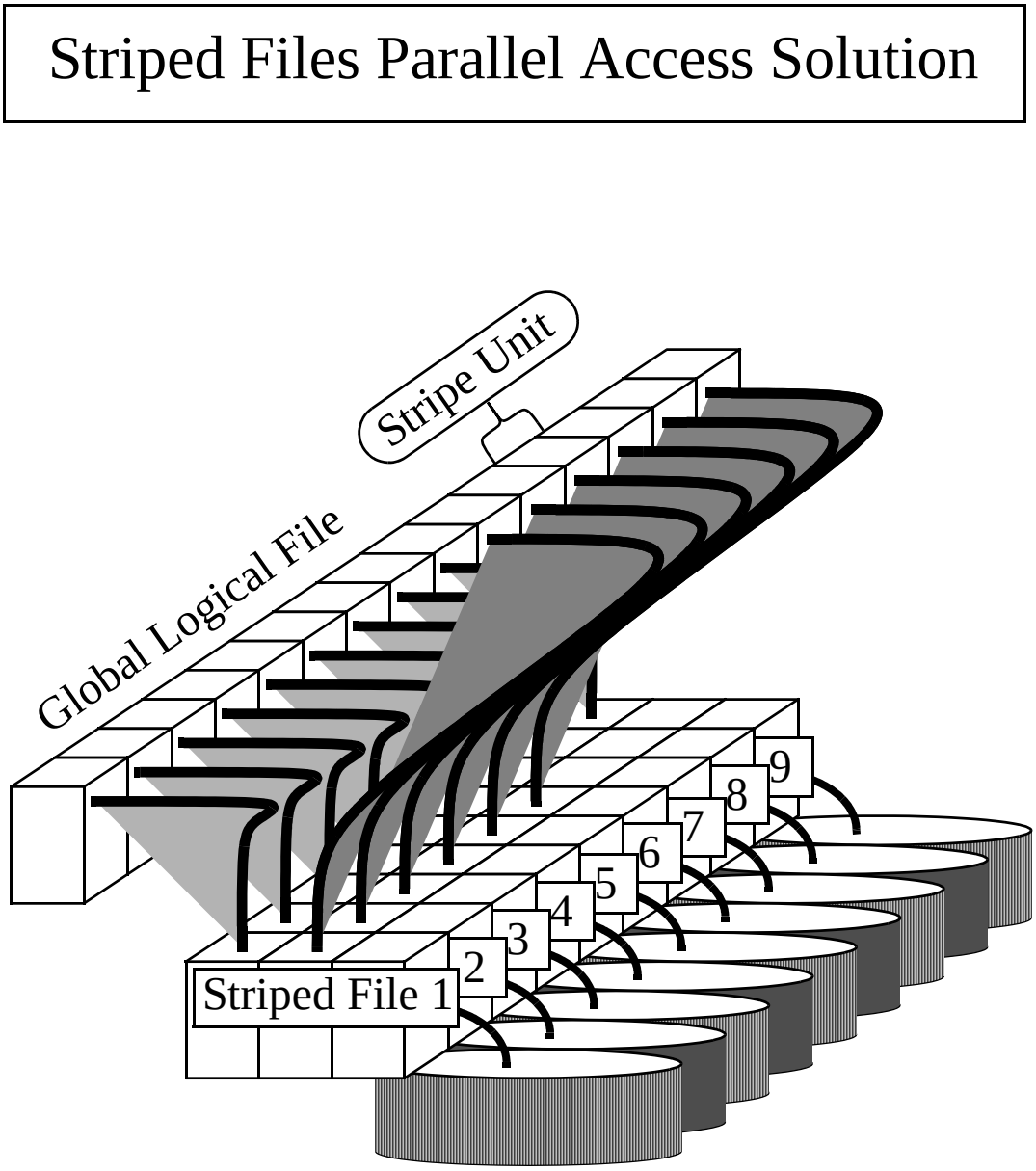
- Requirements for Parallel I/O
- Striped Files Parallel Access Solution
- Interface
- Network Communication and Disk Access optimisation

Fig. 02. We are trying to be close to the linear dependance of throughput from the



number of I/O devices. The upper limit of throughput can be the controller(s)/bus bandwidth or the TOTAL network bandwidth.

Fig. 03. The basic idea of our implementation is cyclical distribution of logical file



accross the set of striped files. Access to single part of logical file require parallel access to set of striped files.

Fig. 04. The native interface is SFIO, but we work to provide also MPI-II I/O inter-

face.

Interface

● SFIO

MFILE* **mopen**(char *s, int unitsz); // “t0-p1,/tmp/a;t0-p2,/tmp/a”

void **mclose**(MFILE *f);

void **mchsize**(MFILE *f, long size);

void **mdelete**(char *s);

void **mcreate**(char *s);

void **mread**(MFILE *f, long offset, char *buf, unsigned count);

void **mwrite**(MFILE *f, long offset, char *buf, unsigned count);

void **mreadb**(MFILE *f, unsigned bcount,
long Offset[], char *Buf[], unsigned Count[]);

void **mwriteb**(MFILE *f, unsigned bcount,
long Offset[], char *Buf[], unsigned Count[]);

● MPI-II I/O

MPI_File_open	MPI_File_write_all
MPI_File_set_view	MPI_File_read_all
MPI_File_write	MPI_File_write_at_all
MPI_File_read	MPI_File_read_at_all
MPI_File_write_at	MPI_File_close
MPI_File_read_at	MPI_File_delete

Fog. 05.

Network and Disk Optimisation

- Network optimisation for noncollective access
- Disk optimisation for noncollective access
- Collective access optimisation

Pg. 06. Fragmented data optimisation for communication

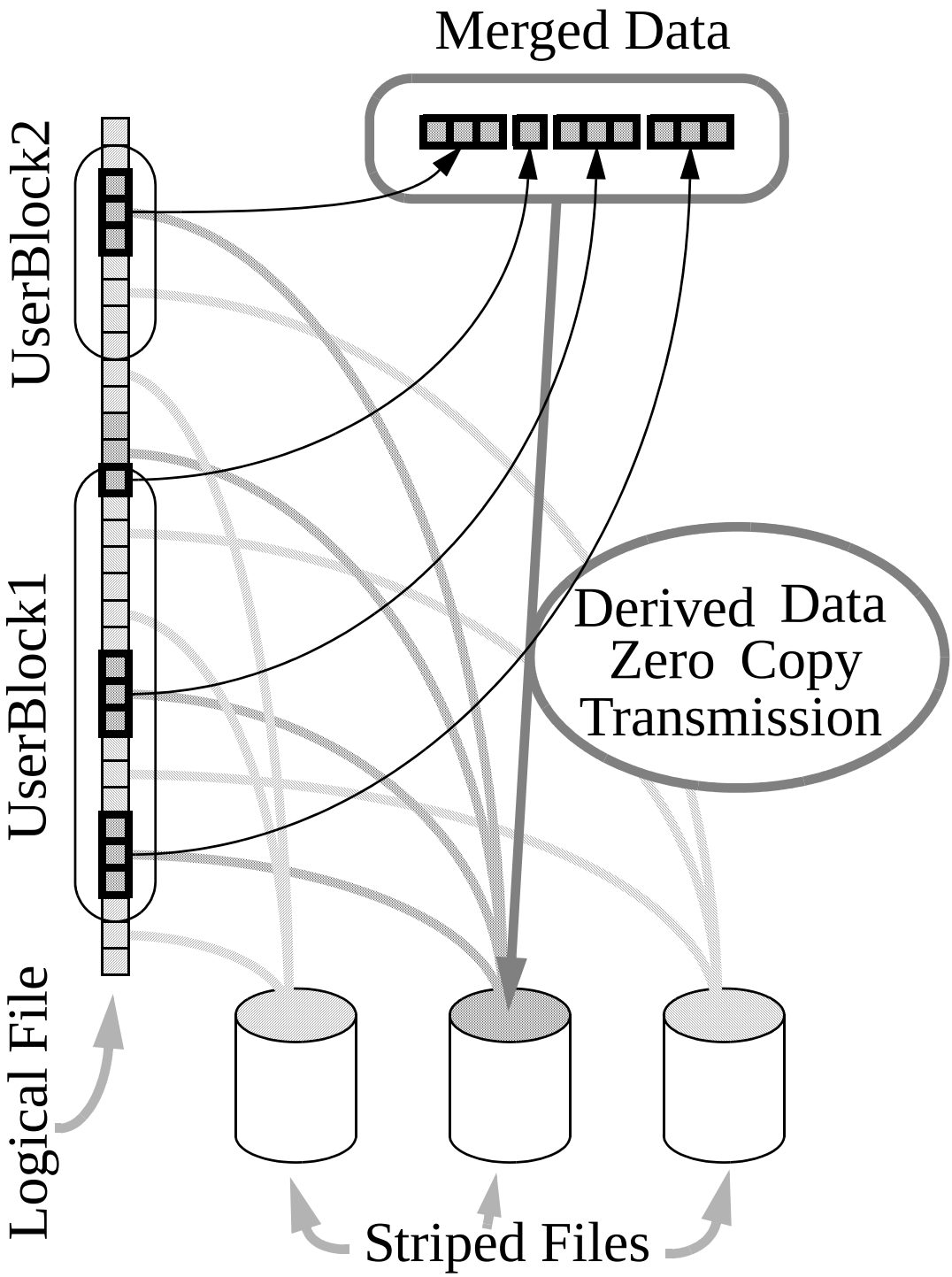


Fig. 07. Data fragmentation is resolved for network communication

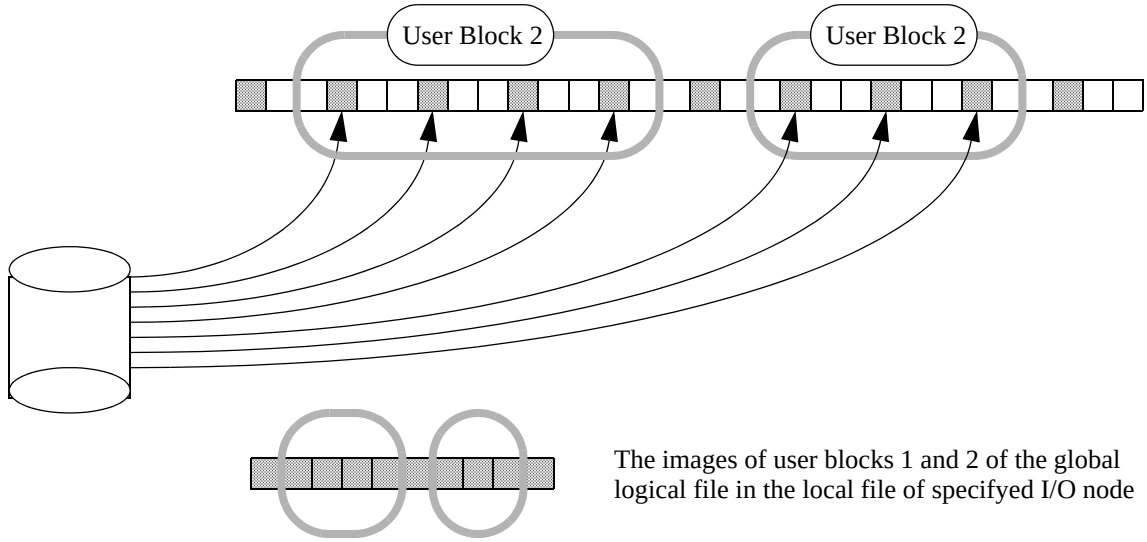


Fig. 08. Data fragmentation still exist at disk access

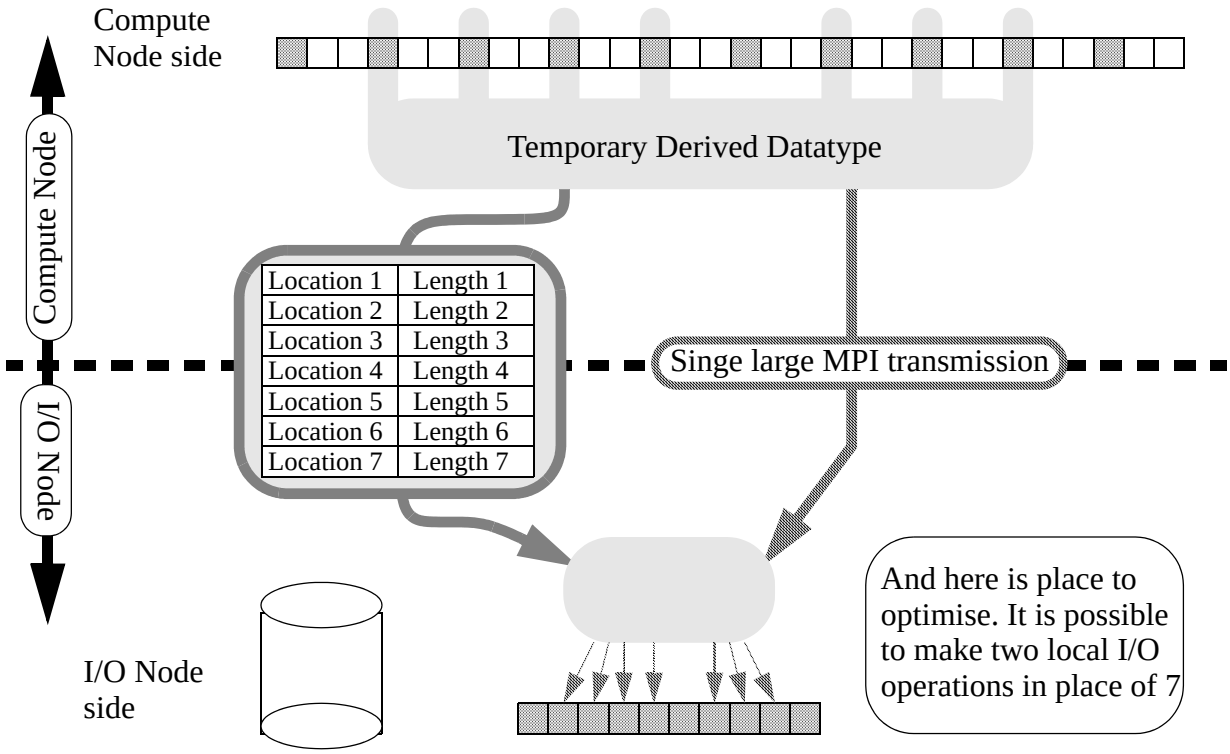


Fig. 09. Data fragmentation is resolved on disk level also

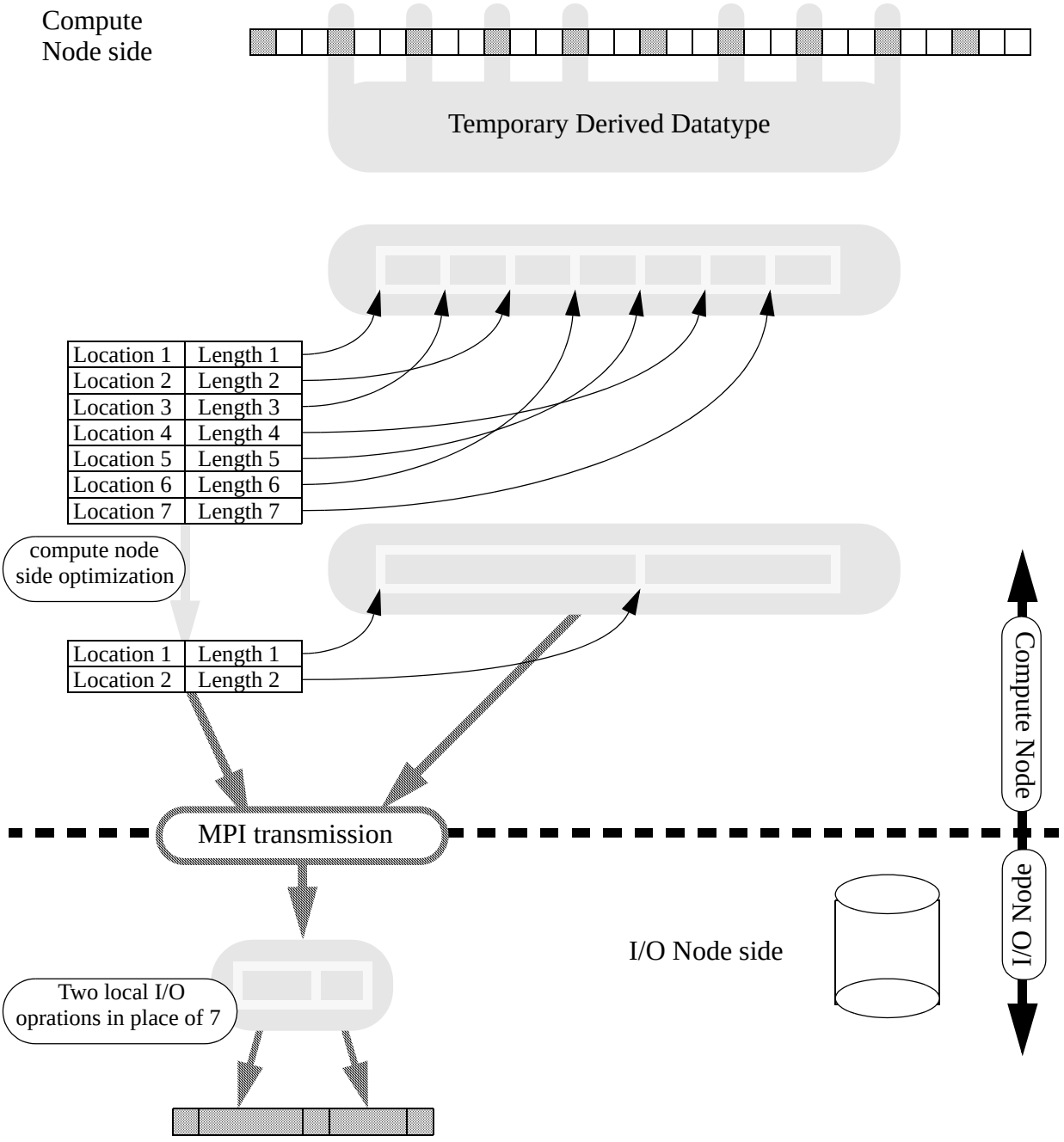


Fig. 10.

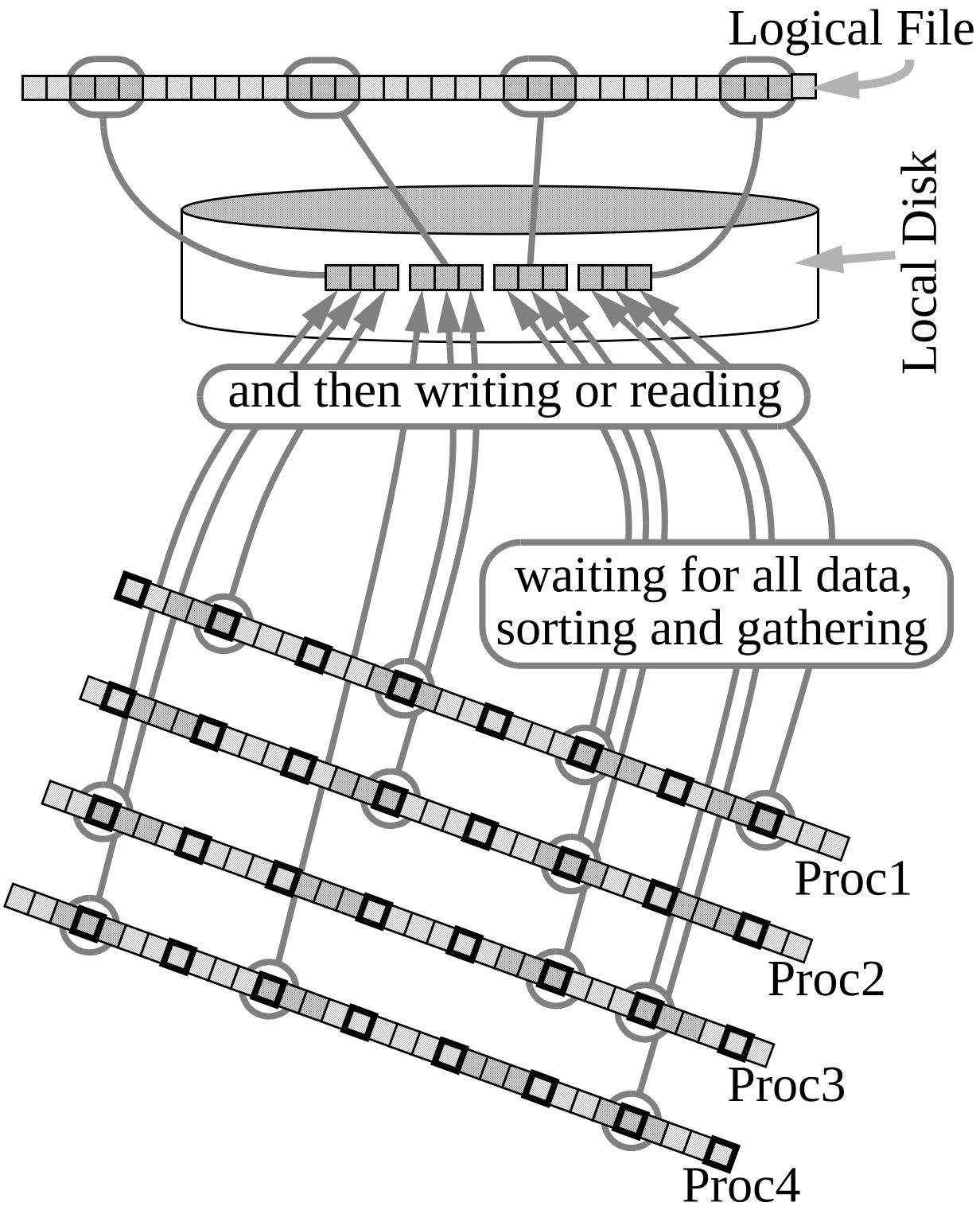
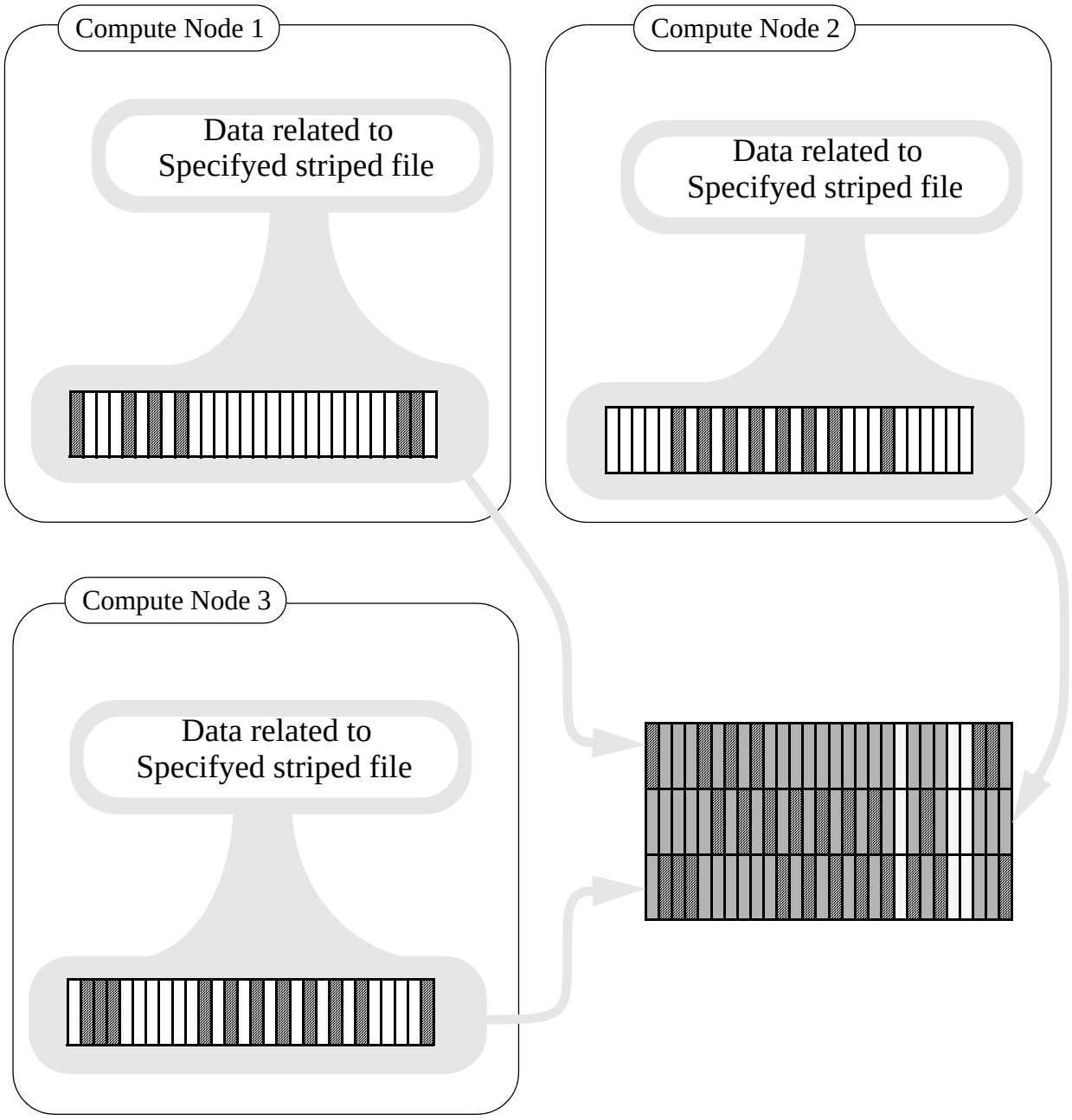

Logical File
Local Disk
and then writing or reading
waiting for all data,
sorting and gathering
Proc1
Proc2
Proc3
Proc4

Fig. 11. Another example when disk access optimisation is required at collective



operations. Here each compute node tryes to access some part of logical file, but in the picture is shown the translation of this part to the view of some specifyed striped file. For each of compute node the striped file is same.

2.0 SFIO software architecture

2.1 Encapsulation over MPI

2.2 Functionality

2.3 Data organisation and optimisation

Fig. 12.

- Encapsulation over MPI
- Functionality
- Data organisation and optimisation

Fig. 13. Encapsulation over MPI

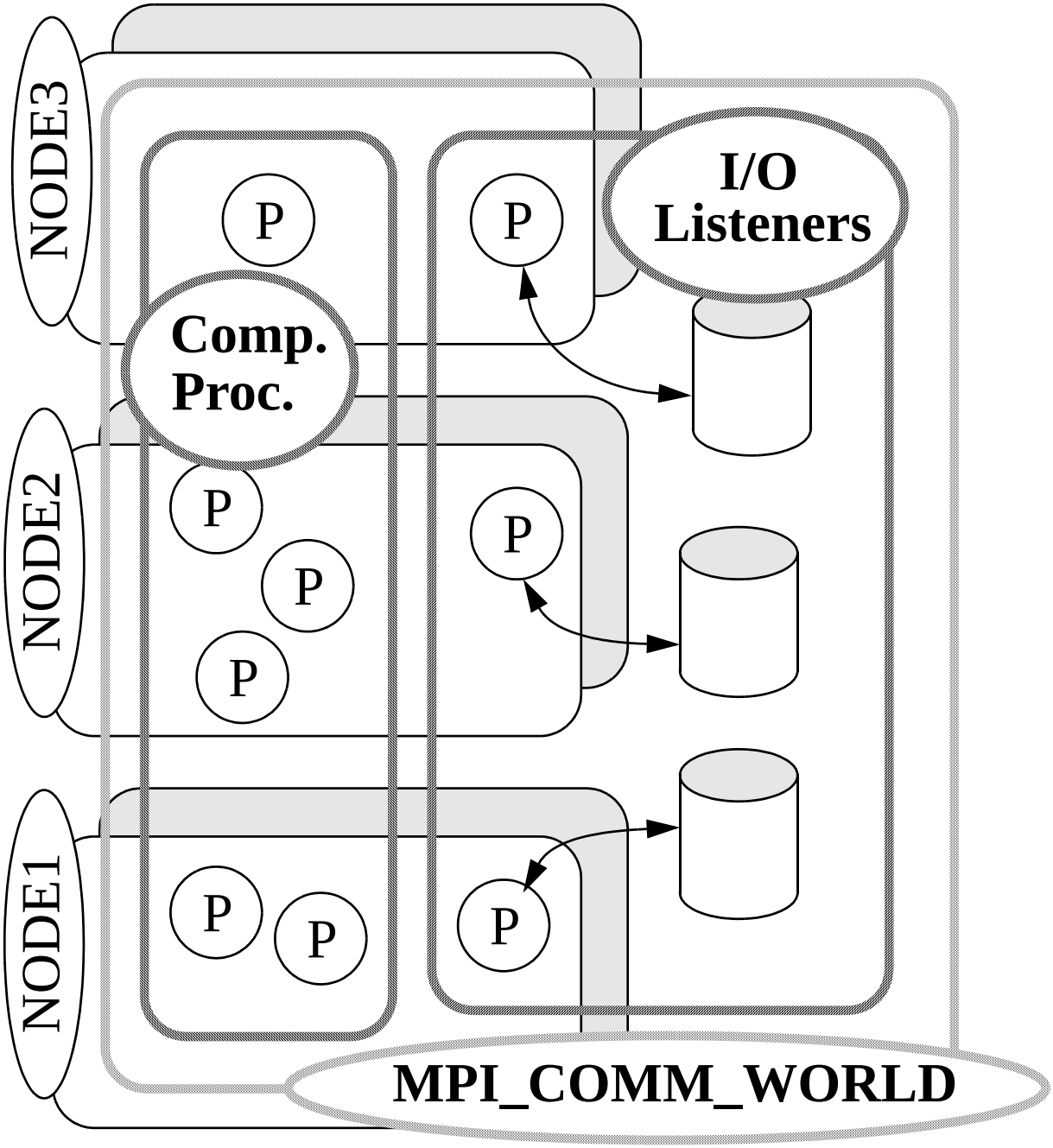
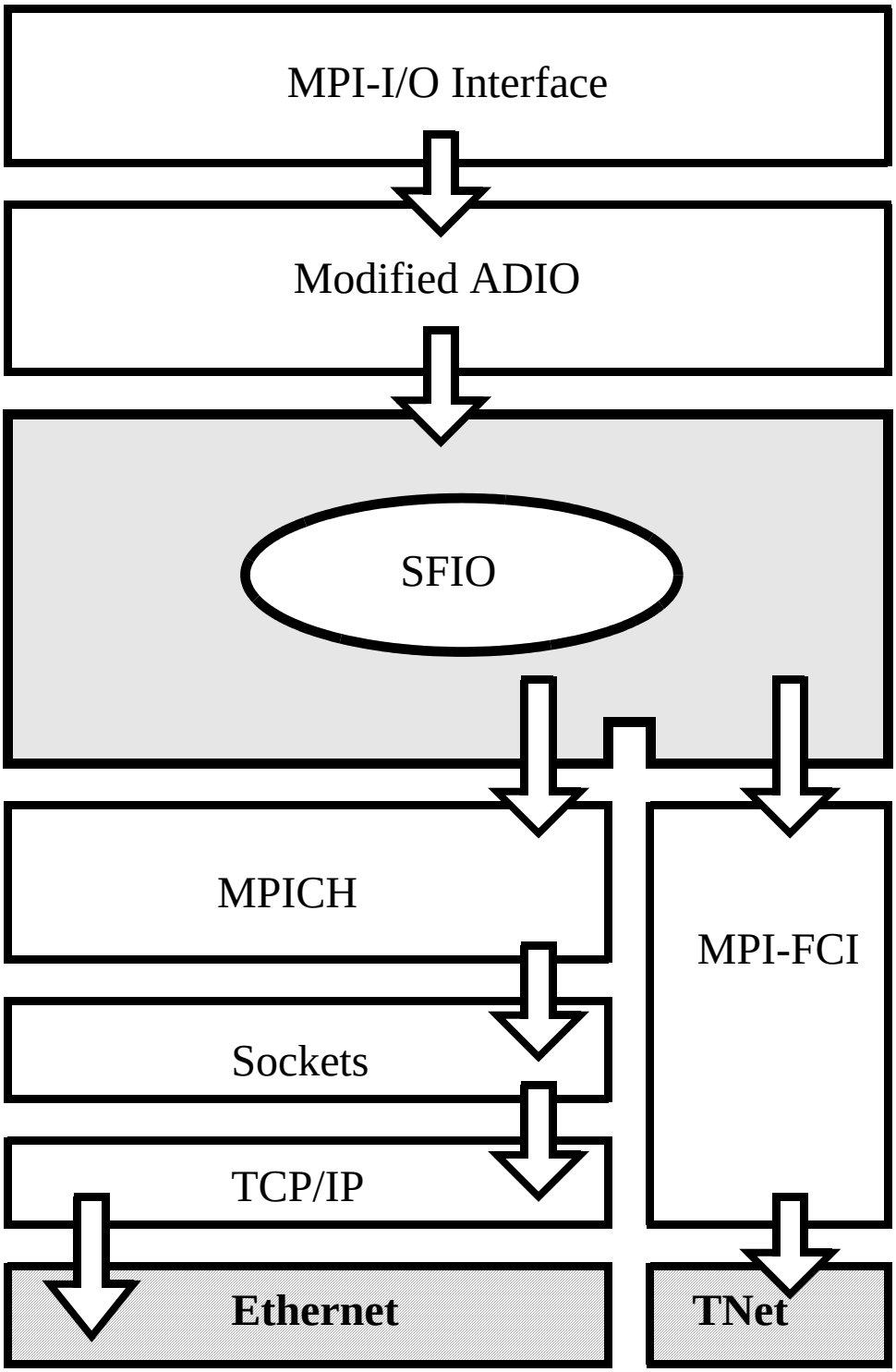
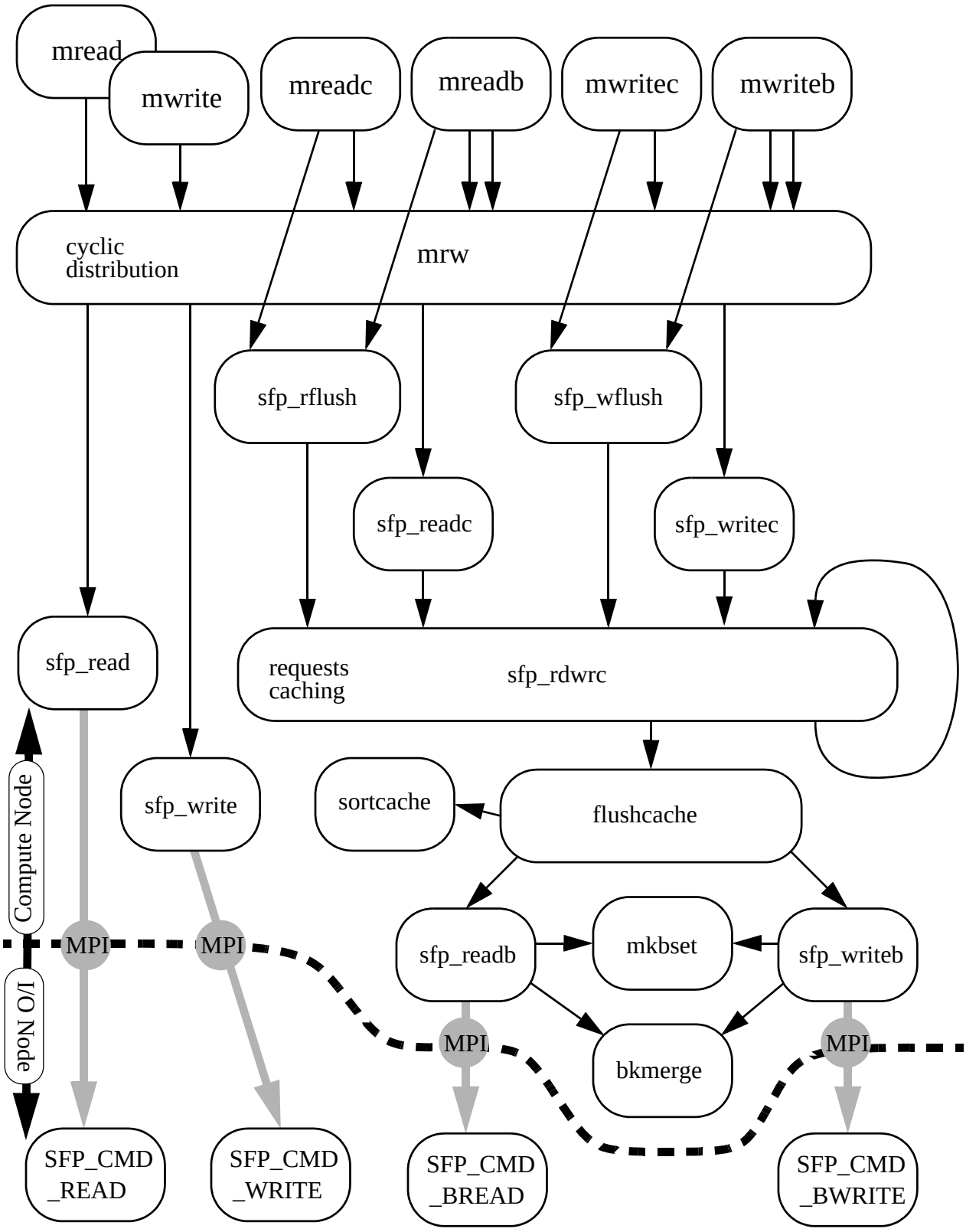


Fig. 14. Functionality. SFIO is implemented and tested with Digital Unix



(MPICH, FCI) and Intel Windos NT (MPICH).

Fig. 15. Functionality, data organisation and optimisation



3.0 SFIO interface

3.1 Single Block Access interface

3.2 Multiple Block Access interface

Fig. 16.

- Single Block Access Interface

- Single Block Access Interface

Fig. 17. If there is one compute node

Single Block Access Interface

```
#include "/usr/p5/gabriely/mpi/lib/mio.h"

int _main(int argc, char *argv[])
{
   char buf[]="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
   MFILE *f;

   f=mopen( "t0-p1.epfl.ch,/tmp/aa;t0-p2.epfl.ch,/tmp/aa" , 10 );

   mwrite( f , 0 , buf , 26 );

   mclose(f);
   return 0;
}
```

Offset in Logical File

Names of Striped Files

Stripe Unit Size

Fig. 18. If there are more than one compute nodes

Single Block Access Interface

```
#include "/usr/p5/gabriely/mpi/lib/mio.h"
int _main(int argc, char *argv[])
{
        char buf[]="abcdefghijklmnopqrstuvwxyz";
        MFILE *f;
        int rank;
        MPI_Comm_rank(_MPI_COMM_WORLD,&rank);

        f=mopen( "t0-p1.epfl.ch,/tmp/aa;t0-p2.epfl.ch,/tmp/aa" , 10 );

        mwrite(f, rank*26 , buf , 26 );

        mclose(f);
        printf("rank=%d\n",rank);
        return 0;
}
```

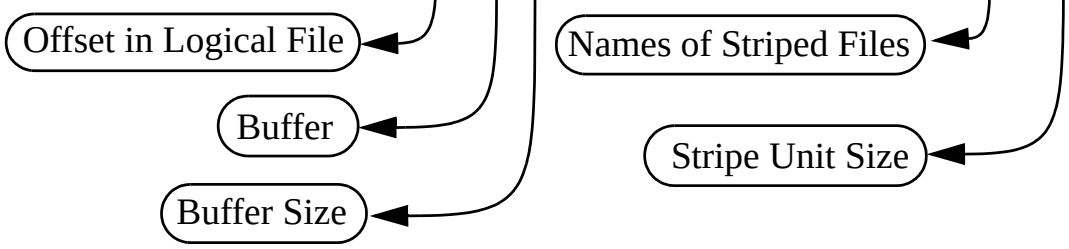


Fig. 19.

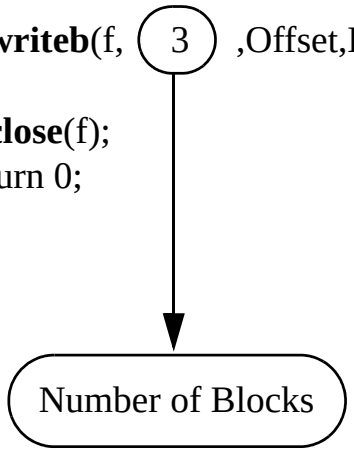
Multiple Block Access Interface

```
#include "/usr/p5/gabriely/mpi/lib/mio.h"
int _main(int argc, char *argv[])
{
      char buf1[]="abcdefghijklmnopqrstuvwxyz";
      char buf2[]="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
      char buf3[]="0123456789";
      char* Buf[]={buf1,buf2,buf3};
      long Offset[]={0,26,52};
      unsigned Count[]={26,26,10};
      MFILE *f;

      f=mopen("t0-p1.epfl.ch,/tmp/aa;t0-p2.epfl.ch,/tmp/aa",10);

      mwriteb(f, 3 ,Offset,Buf,Count);

      mclose(f);
      return 0;
}
```



4.0 Performance measurements

Data Access Performance on Windows NT cluster (4 I/O nodes). Effect of Optimisation

Data Access Performance on Swiss-Tx SFIO over MPICH with 100Mbps ethernet HUB (7 I/O nodes). Effect of Optimisation

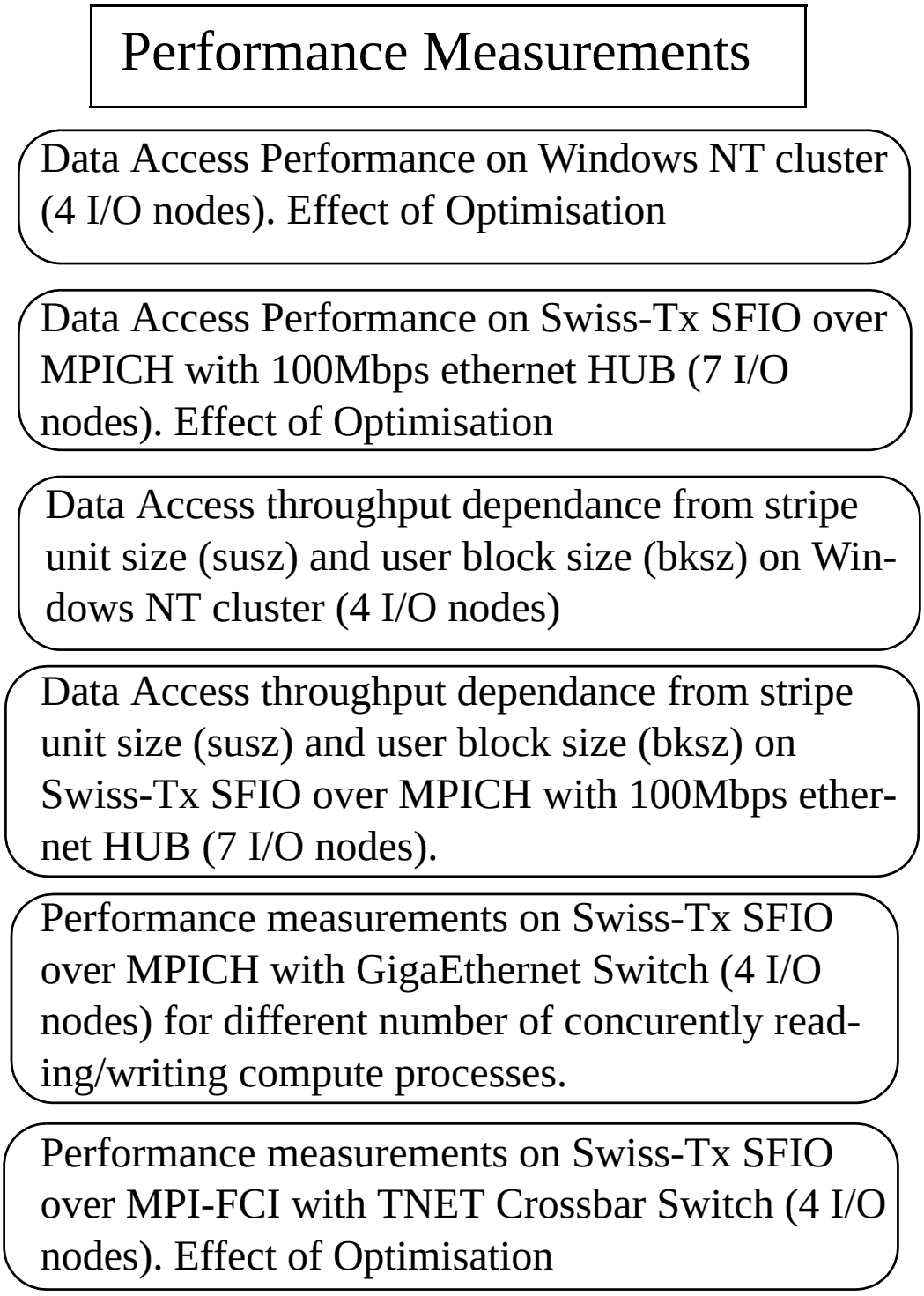
Data Access throughput dependance from stripe unit size (susz) and user block size (bksz) on Windows NT cluster (4 I/O nodes)

Data Access throughput dependance from stripe unit size (susz) and user block size (bksz) on Swiss-Tx SFIO over MPICH with 100Mbps ethernet HUB (7 I/O nodes).

Performance measurements on Swiss-Tx SFIO over MPICH with GigaEthernet Switch (4 I/O nodes) for different number of concurently reading/writing compute processes.

Performance measurements on Swiss-Tx SFIO over MPI-FCI with TNET Crossbar Switch (4 I/O nodes). Effect of Optimisation

Fig. 20.


Performance Measurements
Data Access Performance on Windows NT cluster (4 I/O nodes). Effect of Optimisation
Data Access Performance on Swiss-Tx SFIO over MPICH with 100Mbps ethernet HUB (7 I/O nodes). Effect of Optimisation
Data Access throughput dependance from stripe unit size (susz) and user block size (bksz) on Windows NT cluster (4 I/O nodes)
Data Access throughput dependance from stripe unit size (susz) and user block size (bksz) on Swiss-Tx SFIO over MPICH with 100Mbps ethernet HUB (7 I/O nodes).
Performance measurements on Swiss-Tx SFIO over MPICH with GigaEthernet Switch (4 I/O nodes) for different number of concurently reading/writing compute processes.
Performance measurements on Swiss-Tx SFIO over MPI-FCI with TNET Crossbar Switch (4 I/O nodes). Effect of Optimisation

5.0 Integration into MPI-II I/O

Portable MPI Model Implementation of Argonne National Laboratory

Fig. 21

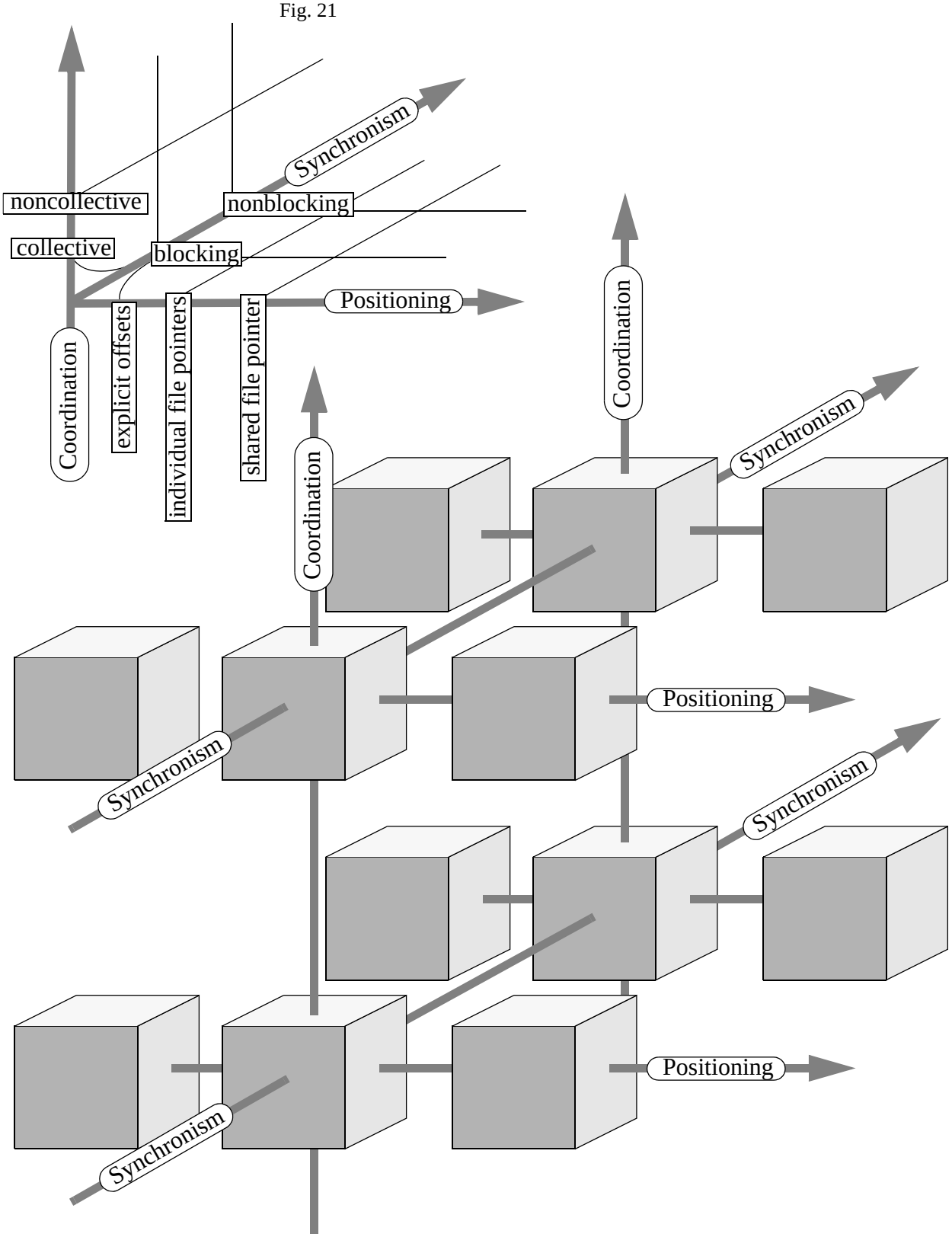

noncollective
collective
blocking
nonblocking
Synchronism
Positioning
Coordination
explicit offsets
individual file pointers
shared file pointer
Coordination
Coordination
Synchronism
Positioning
Synchronism
Synchronism
Positioning
Synchronism

Fig. 22.

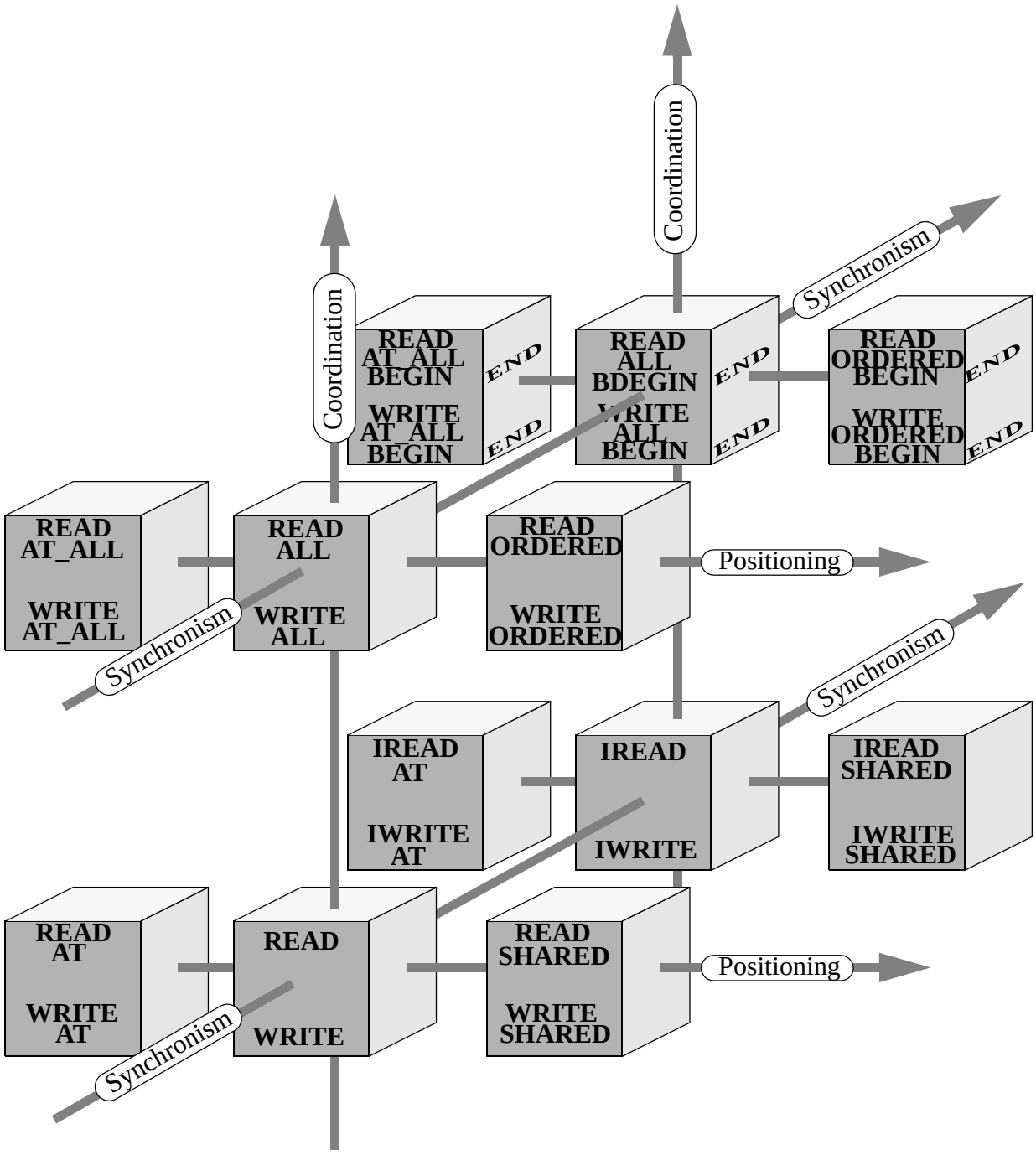

Coordination
Coordination
Synchronism
READ AT_ALL BEGIN
END
WRITE AT_ALL BEGIN
END
READ ALL BDEGIN
END
WRITE ALL BEGIN
END
READ ORDERED BEGIN
END
WRITE ORDERED BEGIN
END
READ AT_ALL
WRITE AT_ALL
READ ALL
WRITE ALL
READ ORDERED
WRITE ORDERED
Positioning
Synchronism
Synchronism
IREAD AT
IWRITE AT
IREAD
IWRITE
IREAD SHARED
IWRITE SHARED
READ AT
WRITE AT
READ
WRITE
READ SHARED
WRITE SHARED
Positioning
Synchronism

Fig. 23.

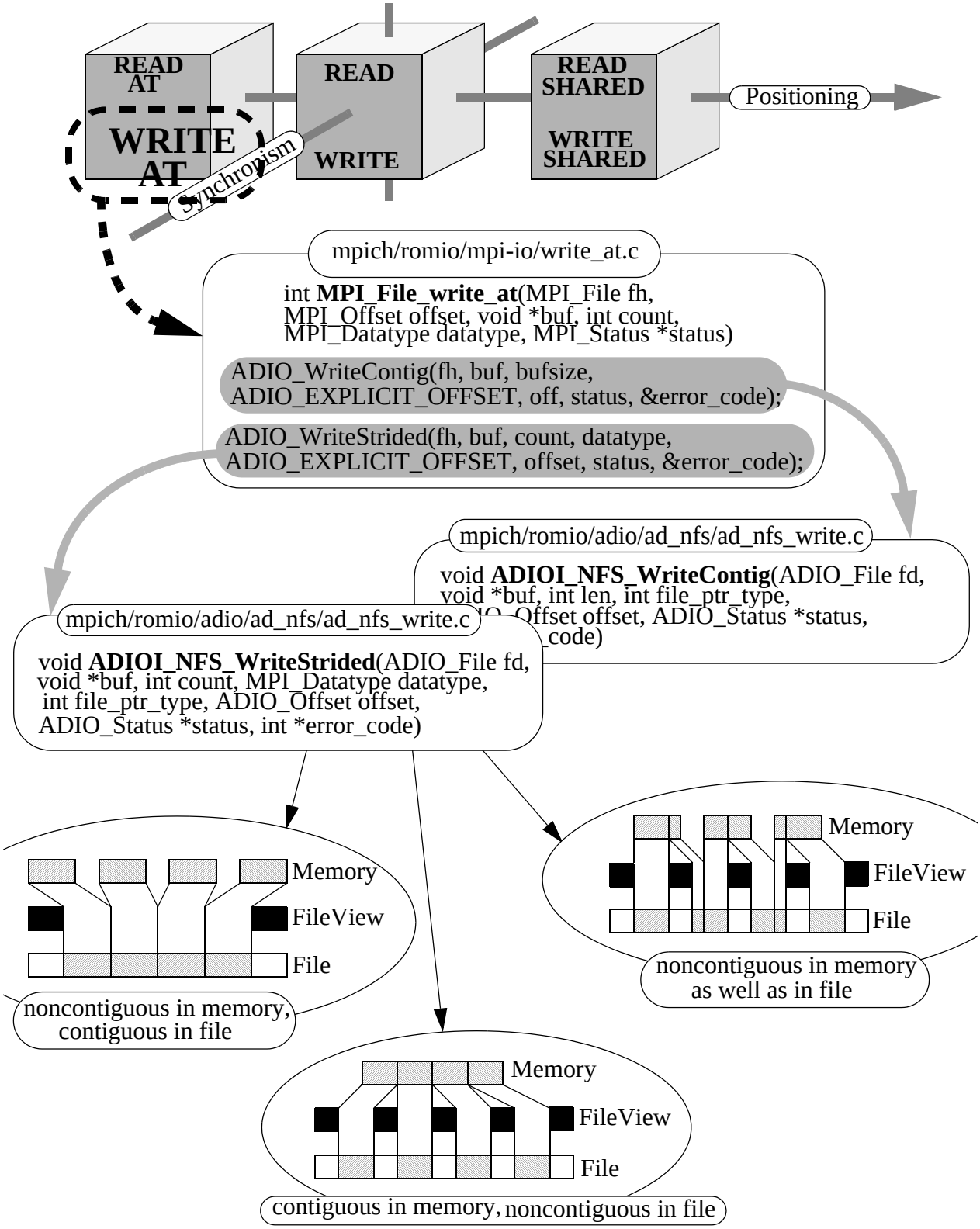

READ AT
WRITE AT
READ
WRITE
READ SHARED
WRITE SHARED
Positioning
Synchronism
mpich/romio/mpi-io/write_at.c
int MPI_File_write_at(MPI_File fh,
MPI_Offset offset, void *buf, int count,
MPI_Datatype datatype, MPI_Status *status)
ADIO_WriteContig(fh, buf, bufsize,
ADIO_EXPLICIT_OFFSET, off, status, &error_code);
ADIO_WriteStrided(fh, buf, count, datatype,
ADIO_EXPLICIT_OFFSET, offset, status, &error_code);
mpich/romio/adio/ad_nfs/ad_nfs_write.c
void ADIOI_NFS_WriteContig(ADIO_File fd,
void *buf, int len, int file_ptr_type,
code)
mpich/romio/adio/ad_nfs/ad_nfs_write.c
void ADIOI_NFS_WriteStrided(ADIO_File fd,
void *buf, int count, MPI_Datatype datatype,
int file_ptr_type, ADIO_Offset offset,
ADIO_Status *status, int *error_code)
Memory
FileView
File
noncontiguous in memory,
contiguous in file
Memory
FileView
File
noncontiguous in memory
as well as in file
Memory
FileView
File
contiguous in memory, noncontiguous in file

Fig. 24.

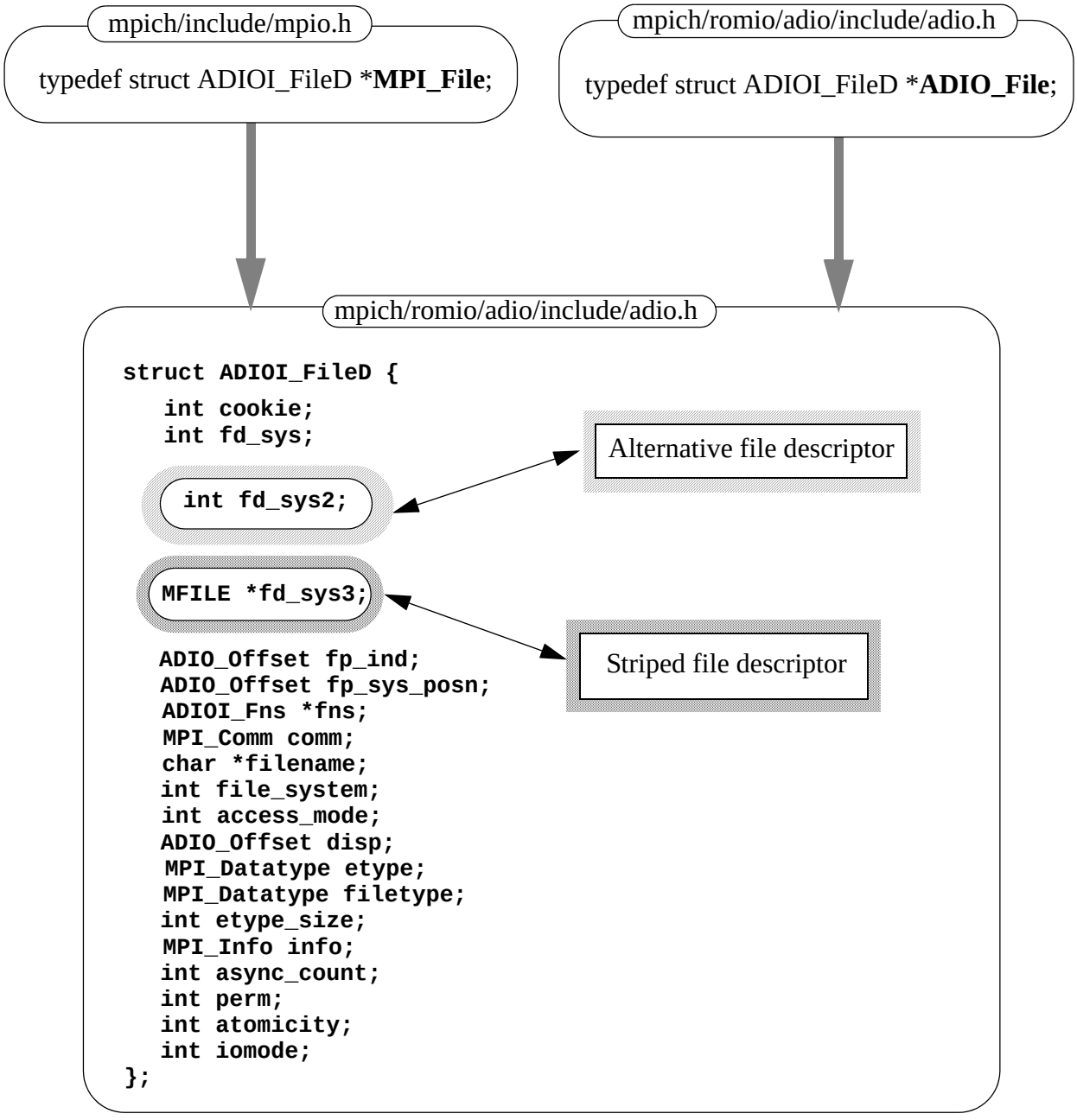
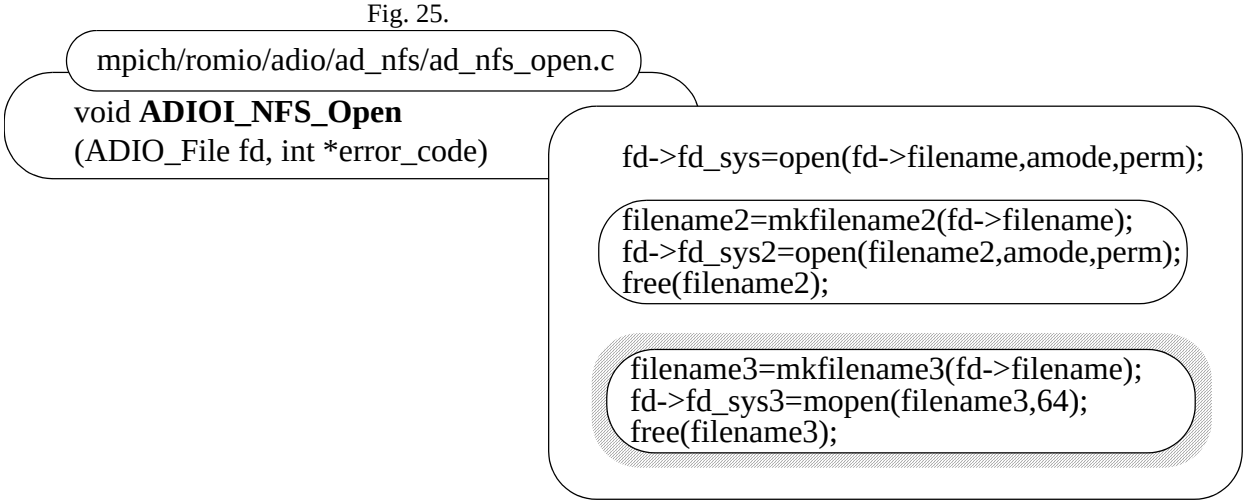

mpich/include/mpio.h
typedef struct ADIOI_FileD *MPI_File;
mpich/romio/adio/include/adio.h
typedef struct ADIOI_FileD *ADIO_File;
mpich/romio/adio/include/adio.h
struct ADIOI_FileD {
int cookie;
int fd_sys;
int fd_sys2;
Alternative file descriptor
MFILE *fd_sys3;
Striped file descriptor
ADIO_Offset fp_ind;
ADIO_Offset fp_sys_posn;
ADIOI_Fns *fns;
MPI_Comm comm;
char *filename;
int file_system;
int access_mode;
ADIO_Offset disp;
MPI_Datatype etype;
MPI_Datatype filetype;
int etype_size;
MPI_Info info;
int async_count;
int perm;
int atomicity;
int iomode;
};

Fig. 25.




```
char *mkfilename3(char *filename)
{
    char *filename3;
    const int devnum=8;
    char *prefix[8]=
    {
        "t0-p1.epfl.ch,/scratch/p1/gabriely/sfp:",
        "t0-p2.epfl.ch,/scratch/p2/gabriely/sfp:",
        "t0-p3.epfl.ch,/scratch/p3/gabriely/sfp:",
        "t0-p4.epfl.ch,/scratch/p4/gabriely/sfp:",
        "t0-p5.epfl.ch,/scratch/p5/gabriely/sfp:",
        "t0-p6.epfl.ch,/scratch/p6/gabriely/sfp:",
        "t0-p7.epfl.ch,/scratch/p7/gabriely/sfp:",
        "t0-p8.epfl.ch,/scratch/p8/gabriely/sfp:"
    };
    char *suffix[8]={ ":01;",":02;",":03;",":04;",
                ":05;",":06;",":07;",":08;" };
    char* name;
    int i;
    int len;
    for(i=0,name=filename;filename[i]!=0;i++)
    {
        if(filename[i]=='/')
            name=filename+i+1;
    }
    for(i=0,len=0;i<devnum;i++)
    {
        len+=strlen(prefix[i]);
        len+=strlen(name);
        len+=strlen(suffix[i]);
    }
    filename3=(char*)malloc(len+1);
    for(i=0,filename3[0]=0;i<devnum;i++)
    {
        strcat(filename3,prefix[i]);
        strcat(filename3,name);
        strcat(filename3,suffix[i]);
    }
    return filename3;
}
```

Fig. 26.

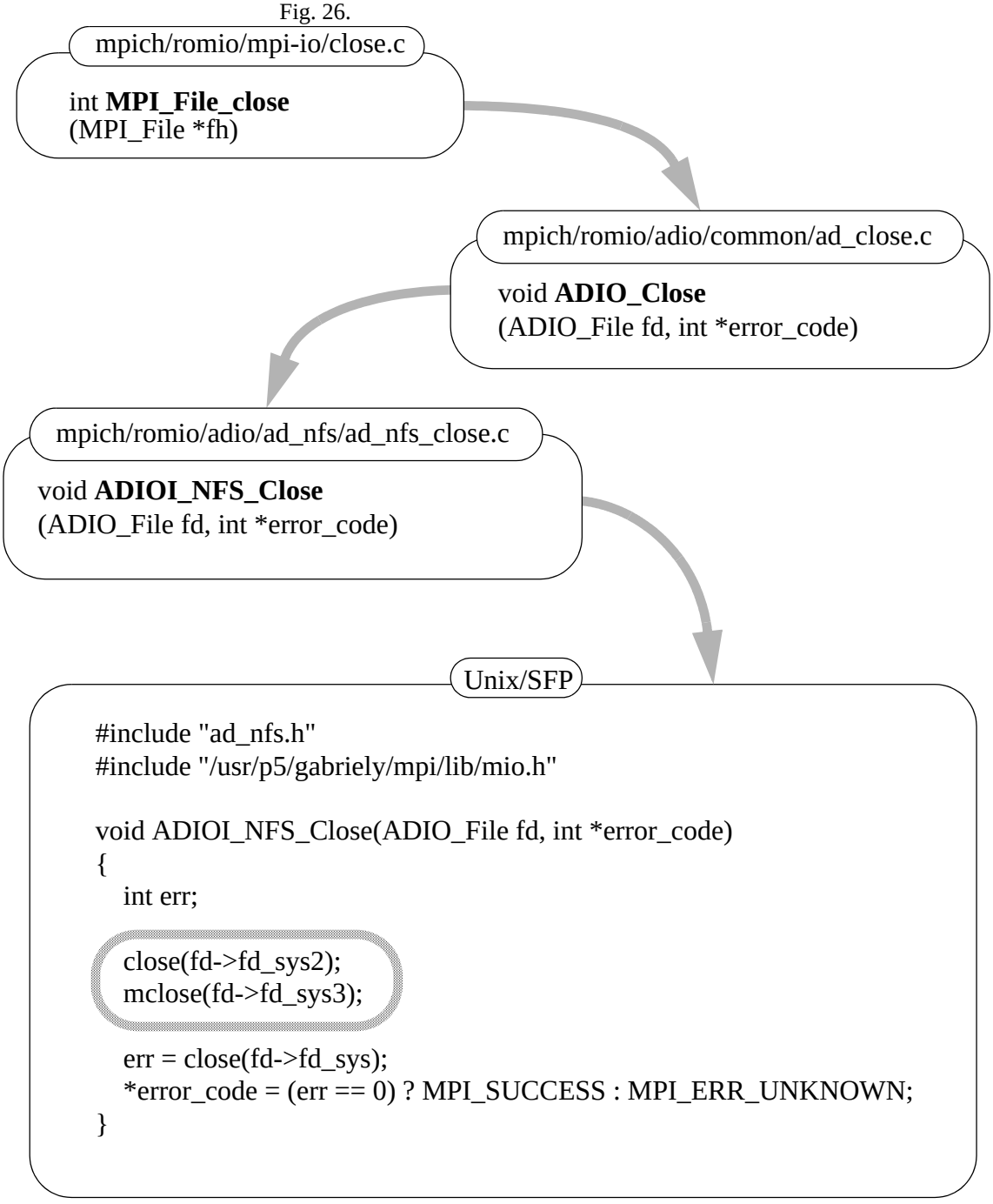

mpich/romio/mpi-io/close.c
int **MPI_File_close**
(MPI_File *fh)
mpich/romio/adio/common/ad_close.c
void **ADIO_Close**
(ADIO_File fd, int *error_code)
mpich/romio/adio/ad_nfs/ad_nfs_close.c
void **ADIOI_NFS_Close**
(ADIO_File fd, int *error_code)
Unix/SFP
#include "ad_nfs.h"
#include "/usr/p5/gabriely/mpi/lib/mio.h"
void ADIOI_NFS_Close(ADIO_File fd, int *error_code)
{
int err;
close(fd->fd_sys2);
mclose(fd->fd_sys3);
err = close(fd->fd_sys);
*error_code = (err == 0) ? MPI_SUCCESS : MPI_ERR_UNKNOWN;
}

Fig. 27.

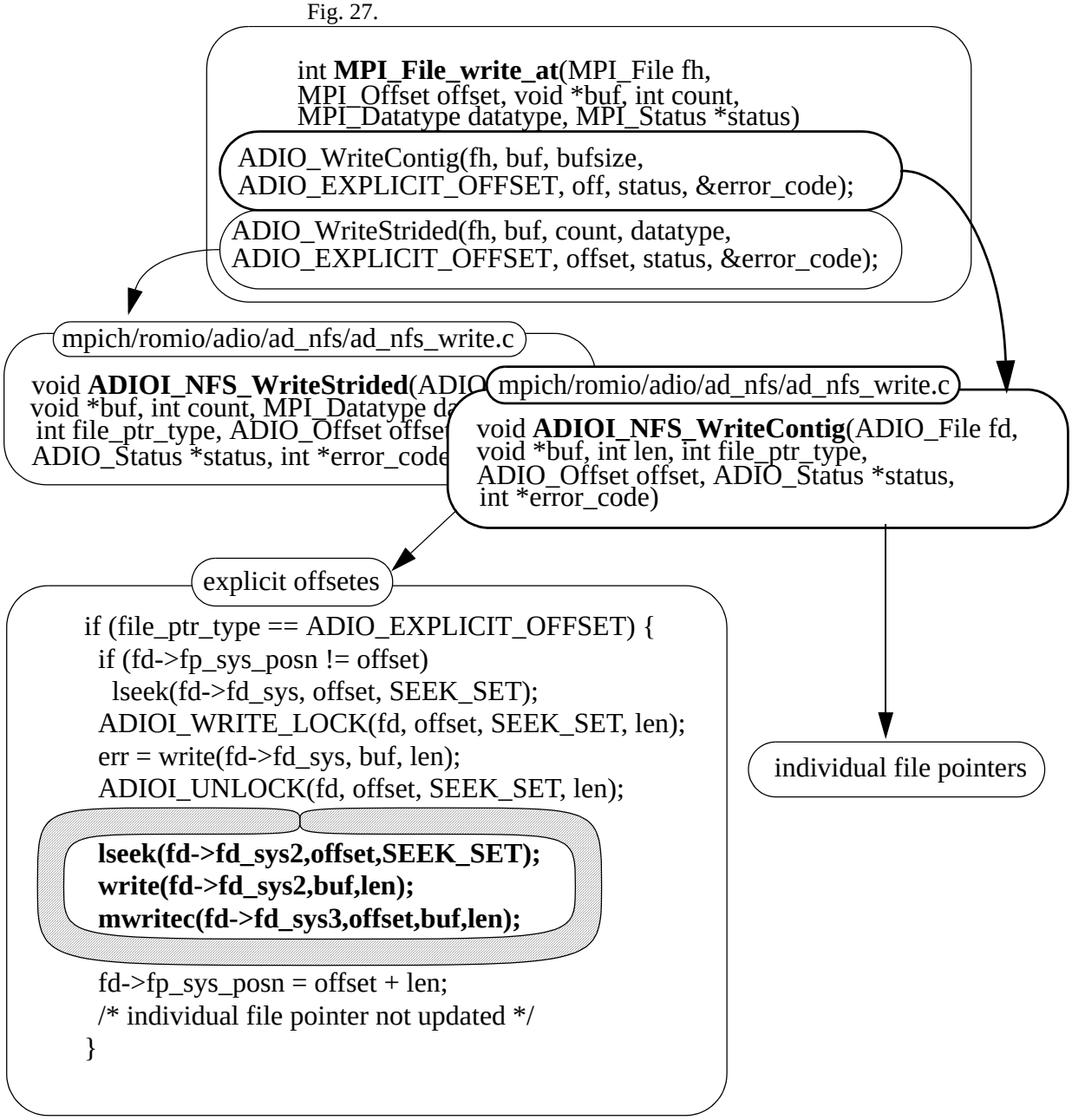

int MPI_File_write_at(MPI_File fh,
MPI_Offset offset, void *buf, int count,
MPI_Datatype datatype, MPI_Status *status)
ADIO_WriteContig(fh, buf, bufsize,
ADIO_EXPLICIT_OFFSET, off, status, &error_code);
ADIO_WriteStrided(fh, buf, count, datatype,
ADIO_EXPLICIT_OFFSET, offset, status, &error_code);
mpich/romio/adio/ad_nfs/ad_nfs_write.c
void ADIOI_NFS_WriteStrided(ADIO
void *buf, int count, MPI_Datatype da
int file_ptr_type, ADIO_Offset offse
ADIO_Status *status, int *error_code
mpich/romio/adio/ad_nfs/ad_nfs_write.c
void ADIOI_NFS_WriteContig(ADIO_File fd,
void *buf, int len, int file_ptr_type,
ADIO_Offset offset, ADIO_Status *status,
int *error_code)
explicit offsetes
if (file_ptr_type == ADIO_EXPLICIT_OFFSET) {
if (fd->fp_sys_posn != offset)
lseek(fd->fd_sys, offset, SEEK_SET);
ADIOI_WRITE_LOCK(fd, offset, SEEK_SET, len);
err = write(fd->fd_sys, buf, len);
ADIOI_UNLOCK(fd, offset, SEEK_SET, len);
lseek(fd->fd_sys2,offset,SEEK_SET);
write(fd->fd_sys2,buf,len);
mwritec(fd->fd_sys3,offset,buf,len);
fd->fp_sys_posn = offset + len;
/* individual file pointer not updated */
}
individual file pointers

Fig. 28.

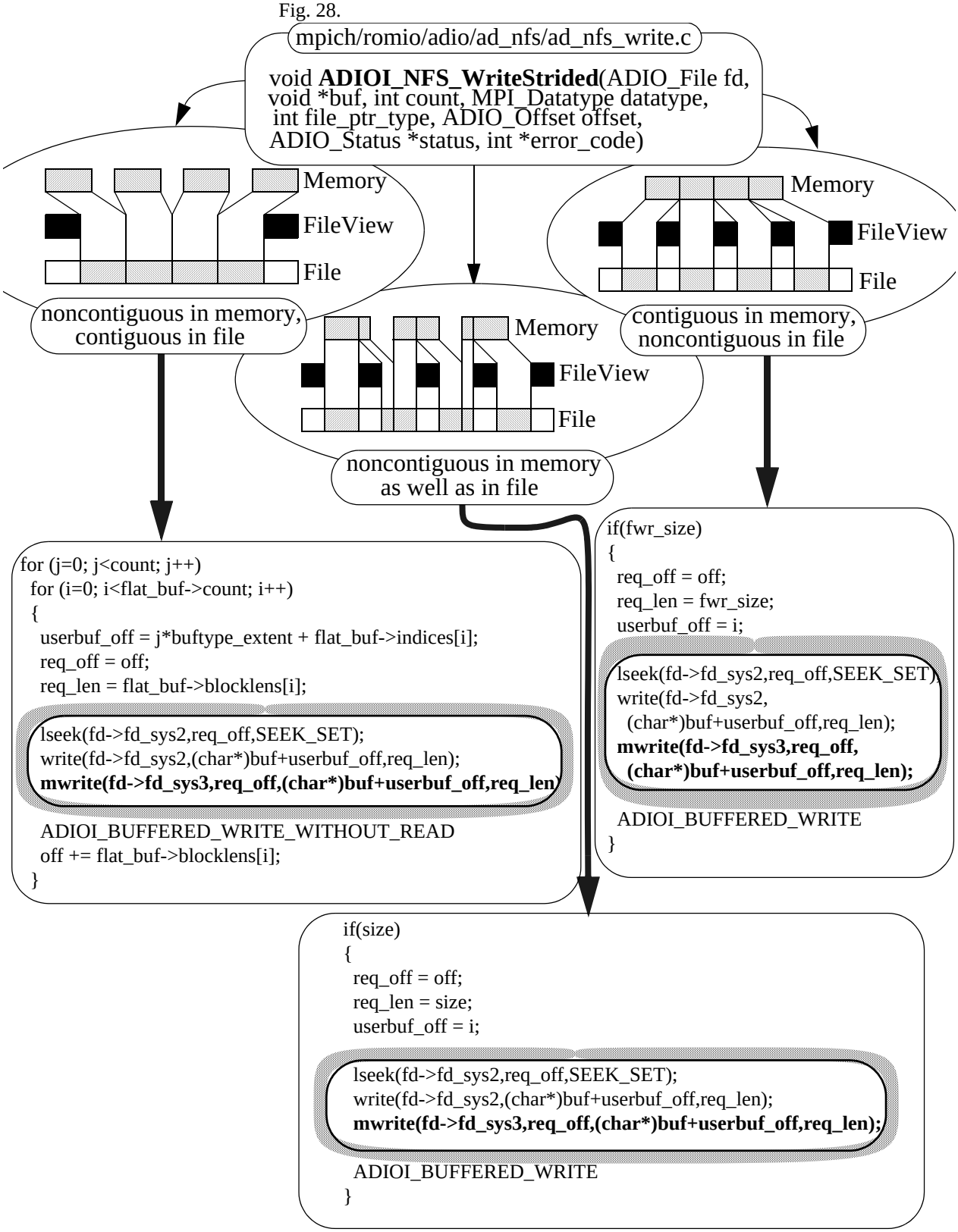

mpich/romio/adio/ad_nfs/ad_nfs_write.c
void ADIOI_NFS_WriteStrided(ADIO_File fd, void *buf, int count, MPI_Datatype datatype, int file_ptr_type, ADIO_Offset offset, ADIO_Status *status, int *error_code)
Memory
FileView
File
noncontiguous in memory, contiguous in file
Memory
FileView
File
contiguous in memory, noncontiguous in file
Memory
FileView
File
noncontiguous in memory as well as in file
for (j=0; j<count; j++)
for (i=0; i<flat_buf->count; i++)
{
userbuf_off = j*buftype_extent + flat_buf->indices[i];
req_off = off;
req_len = flat_buf->blocklens[i];
lseek(fd->fd_sys2,req_off,SEEK_SET);
write(fd->fd_sys2,(char*)buf+userbuf_off,req_len);
mwrite(fd->fd_sys3,req_off,(char*)buf+userbuf_off,req_len);
ADIOI_BUFFERED_WRITE_WITHOUT_READ
off += flat_buf->blocklens[i];
}
if(fwr_size)
{
req_off = off;
req_len = fwr_size;
userbuf_off = i;
lseek(fd->fd_sys2,req_off,SEEK_SET);
write(fd->fd_sys2,
(char*)buf+userbuf_off,req_len);
mwrite(fd->fd_sys3,req_off,
(char*)buf+userbuf_off,req_len);
ADIOI_BUFFERED_WRITE
}
if(size)
{
req_off = off;
req_len = size;
userbuf_off = i;
lseek(fd->fd_sys2,req_off,SEEK_SET);
write(fd->fd_sys2,(char*)buf+userbuf_off,req_len);
mwrite(fd->fd_sys3,req_off,(char*)buf+userbuf_off,req_len);
ADIOI_BUFFERED_WRITE
}

Fig. 29.

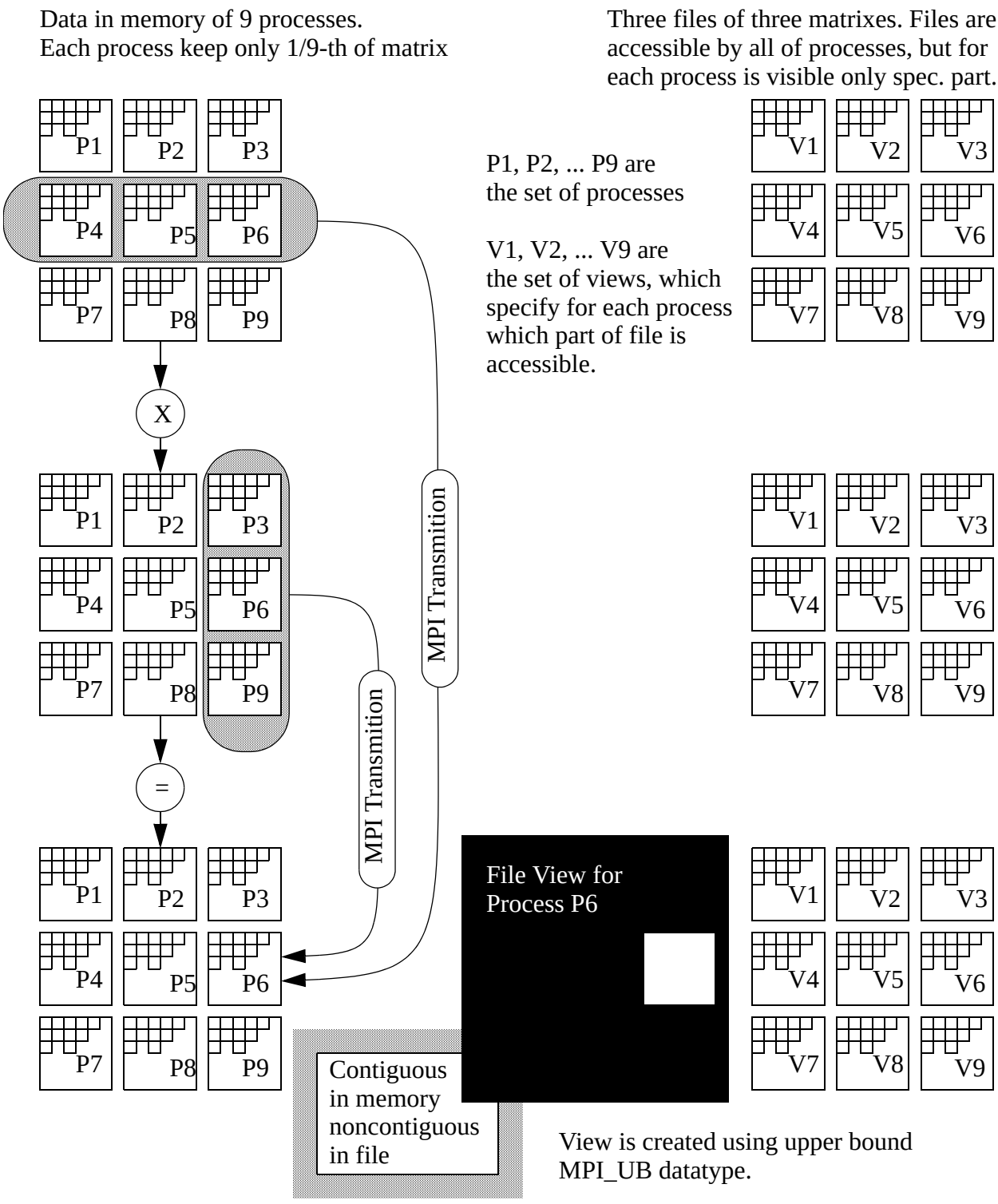
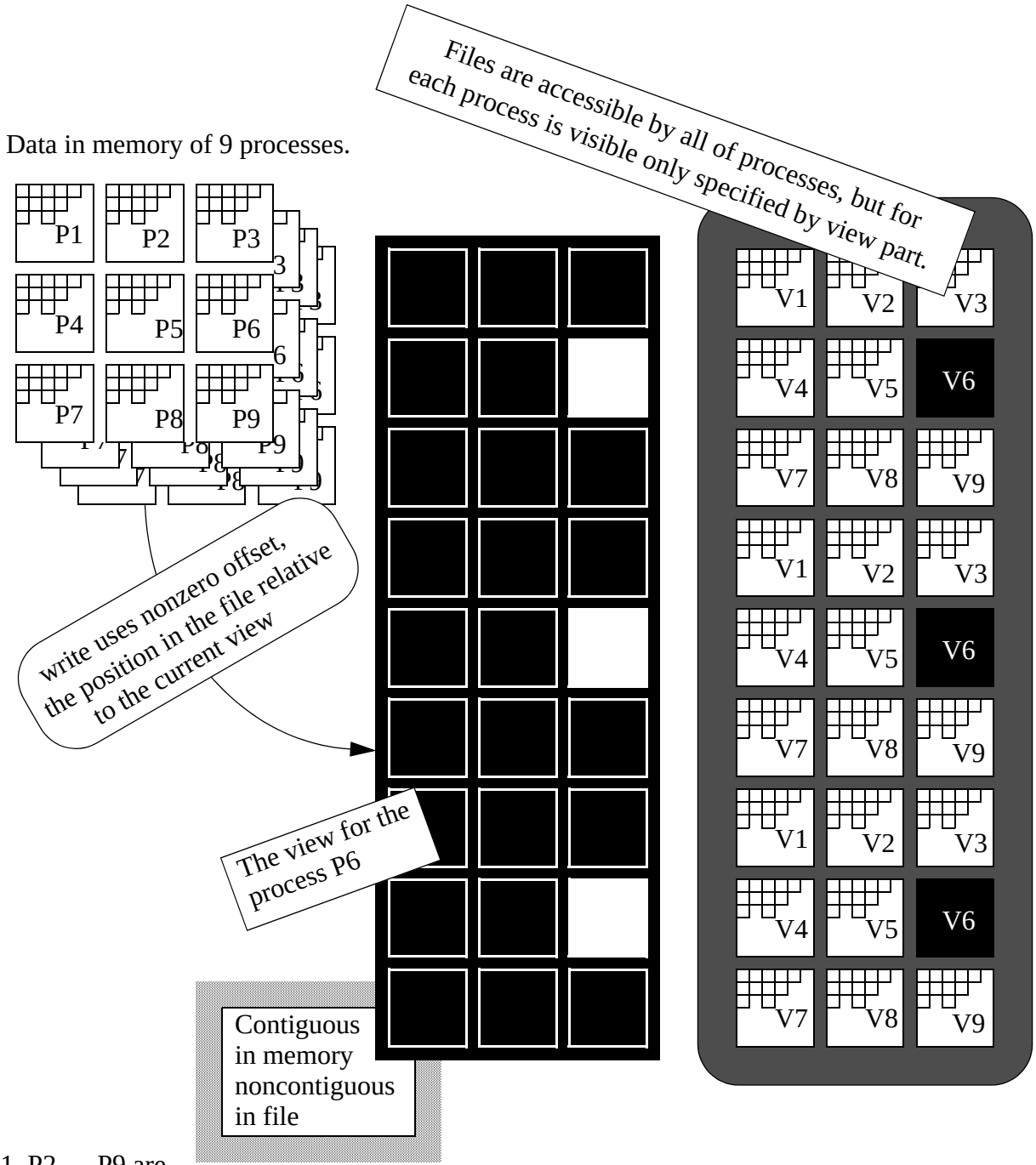

Data in memory of 9 processes.
Each process keep only 1/9-th of matrix
Three files of three matrixes. Files are accessible by all of processes, but for each process is visible only spec. part.
P1, P2, ... P9 are the set of processes
V1, V2, ... V9 are the set of views, which specify for each process which part of file is accessible.
P1
P2
P3
P4
P5
P6
P7
P8
P9
V1
V2
V3
V4
V5
V6
V7
V8
V9
X
=
MPI Transmition
MPI Transmition
File View for Process P6
Contiguous in memory noncontiguous in file
View is created using upper bound MPI_UB datatype.

Fig. 30.



P1, P2, ... P9 are the set of processes

V1, V2, ... V9 are the set of views

View is created using upper bound and lower bound MPI_UB, MPI_LB datatype.

6.0 Conclusion

This is a cheap way of obtaining high performance I/O