Parallel I/O system for Swiss-Tx

Striped file based parallel I/O system with the NAFS user interface. Realisation of the system using the MPI communications as the transport layer.

LSP-DI EPFL

1.0 Intruduction

The goal of whole project is to supplay MPI-I/O to the fast MPI-FCI of Swiss-Tx. The realisation is divided into the two different works.

First is the MPI-I/O interface support. This includes the correct mapping of the defined by user derived data set from the local process memory into the logical file through the file view, which is also defined by derived datatypes. So this is the translation of MPI-I/O calls into the multiple block oriented calls. As the intermediate level for the multiple block oriented calls is taken NAFS interface.

Second part is the actual realization of the I/O system with the intermediate NAFS block-interface. Here the basic rule in the realisation of I/O system is to increase performance using paralelism. And thereafter the striped file distributed system is taken as a basic idea of this layer implementation.

To the Swiss-Tx is ported the NAFS realised on top of Ps^2 of CAP. But because the CAP is socket based, this have performance limitations defined by TCP/IP.

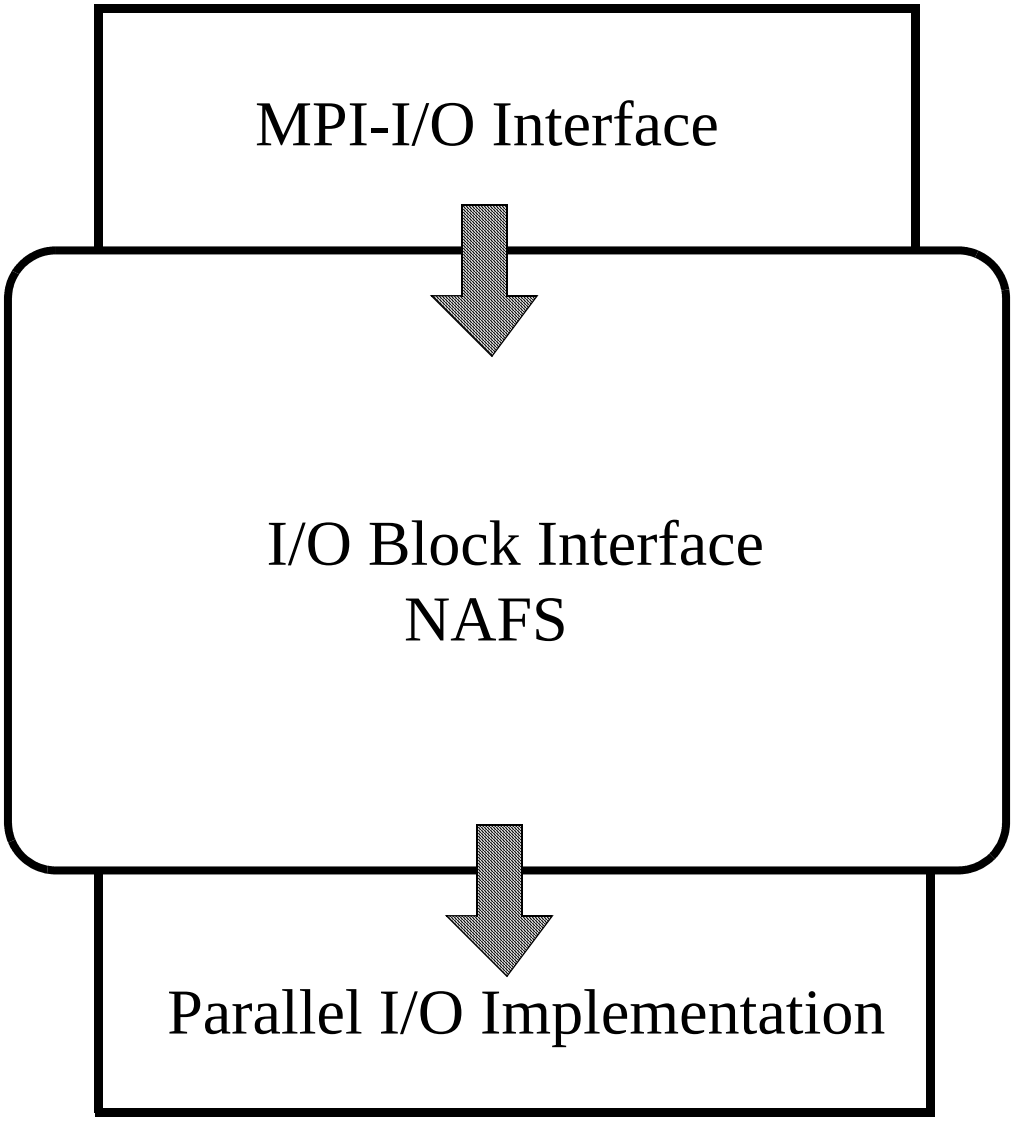
Another alternative for realization of striped file based parallel I/O is the usage of fast MPI communications as the transport layer. This version also have NAFS block-interface but directly based on the MPI communications and the operating system I/O calls.

Pg. 01

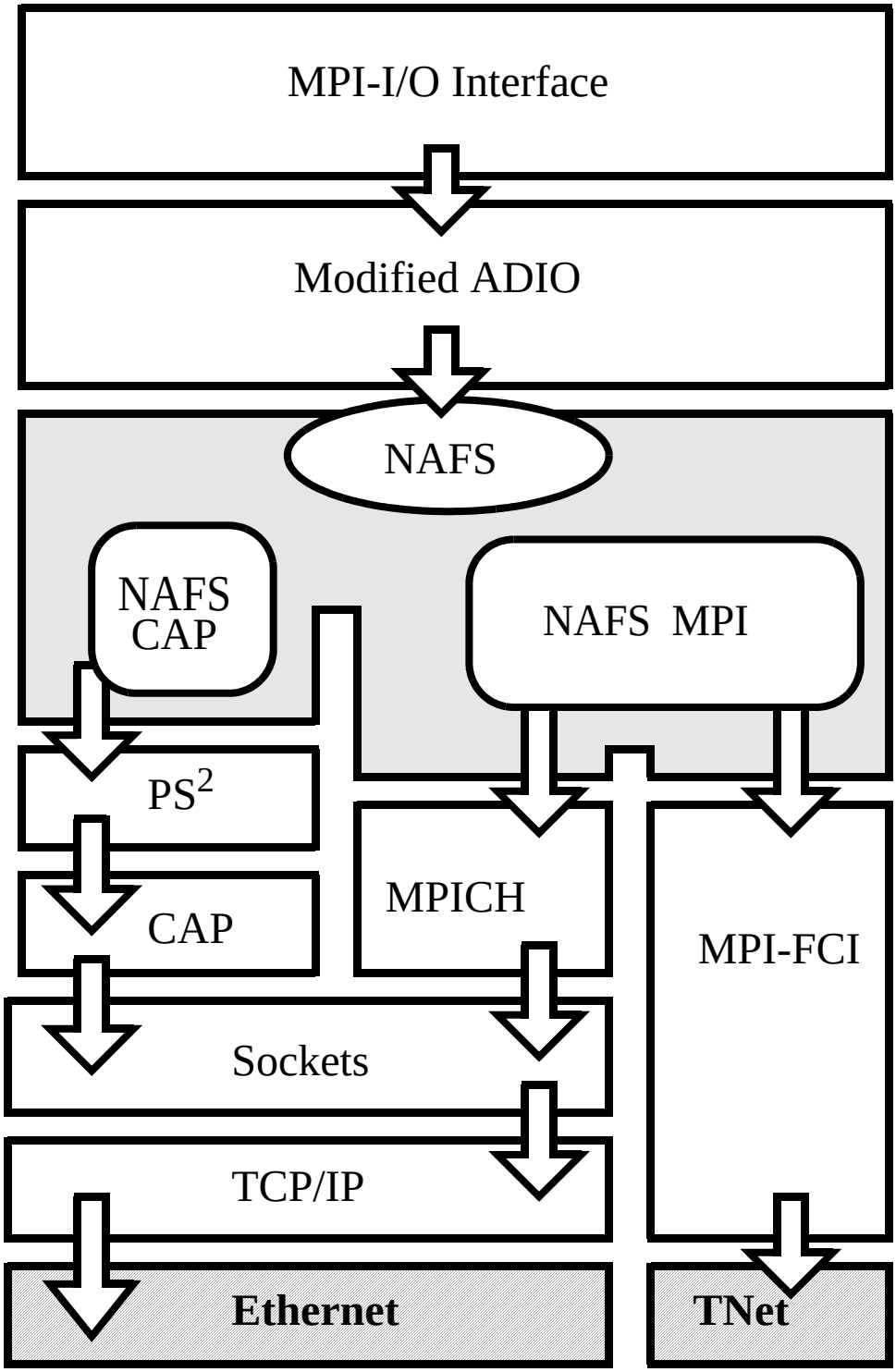
MPI-I/O for Swiss-Tx

- MPI-I/O Interface realization
- Intermediate layer NAFS
- The I/O implementation realization

Pg. 02


MPI-I/O Interface
I/O Block Interface
NAFS
Parallel I/O Implementation

Pg. 03


MPI-I/O Interface
Modified ADIO
NAFS
NAFS CAP
NAFS MPI
PS2
CAP
MPICH
MPI-FCI
Sockets
TCP/IP
Ethernet
TNet

2.0 NAFS on top of MPI

How multiple access oriented I/O system realised on top of MPI, and how paralellism of I/O is reached ?

The idea is following to divide the logical file into the blocks and cyclicaly distribute this blocks into the striped files. Then distribute the striped files accross the I/O nodes,- computers in the network which whill perform disk I/O operations.

After MPI program is launched and number of MPI processes start running on the computers, then the striped file parallel I/O system selects from the set of MPI processes those which will perfom I/O operations and configures them as I/O listener. Those processes, after this will not be available for user computation.

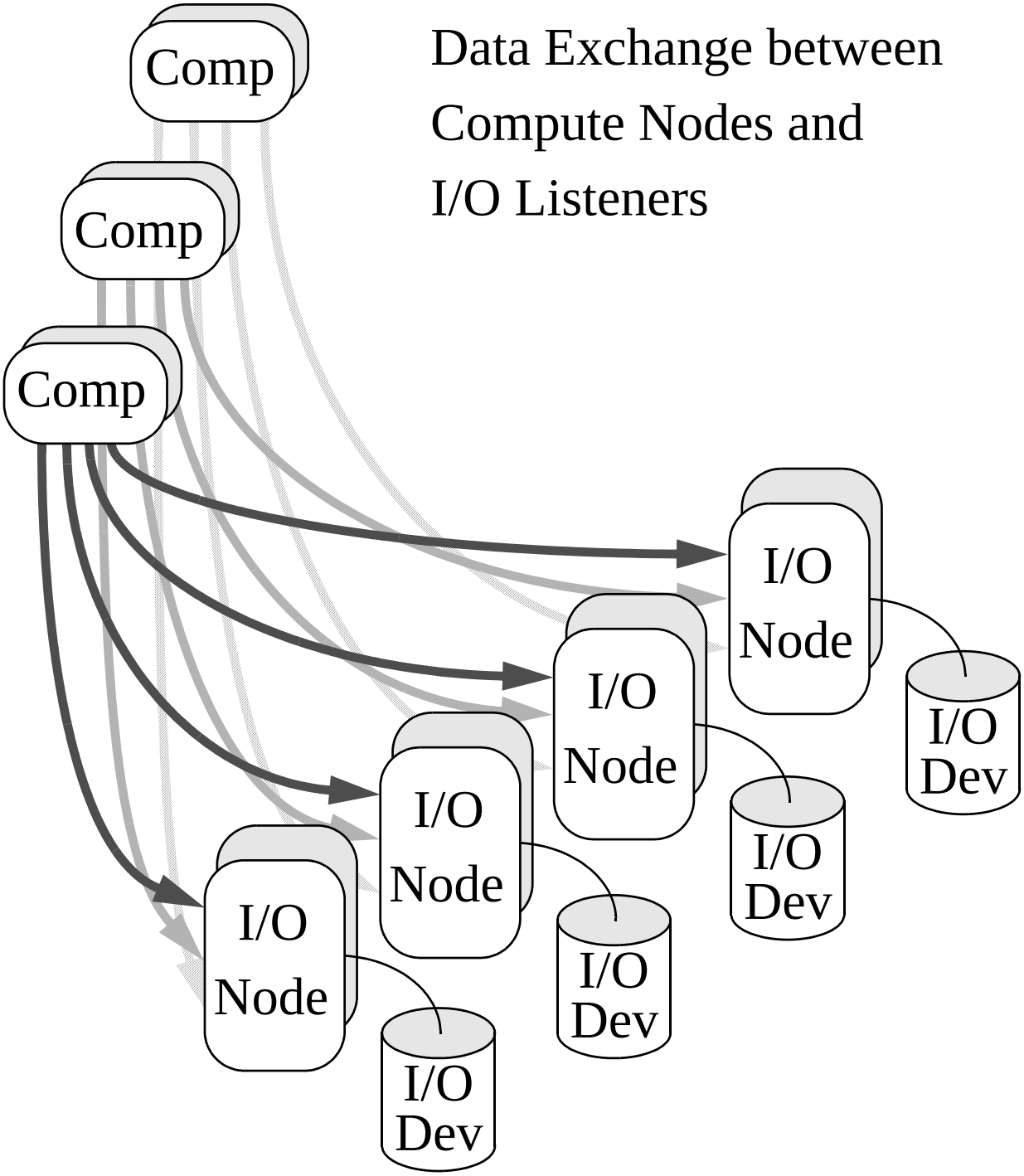
When from user process is called NAFS I/O function to perform I/O operations, the user process computes the amount of subsets of local data which must be processed on different I/O nodes and geterates related requests to the specific I/O listeners using the MPI communications.

Pg. 04

How is I/O parallelism obtained ?

- Division of file into the blocks
- Local files per I/O node
- Cyclic distribution over striped files

Pg. 05

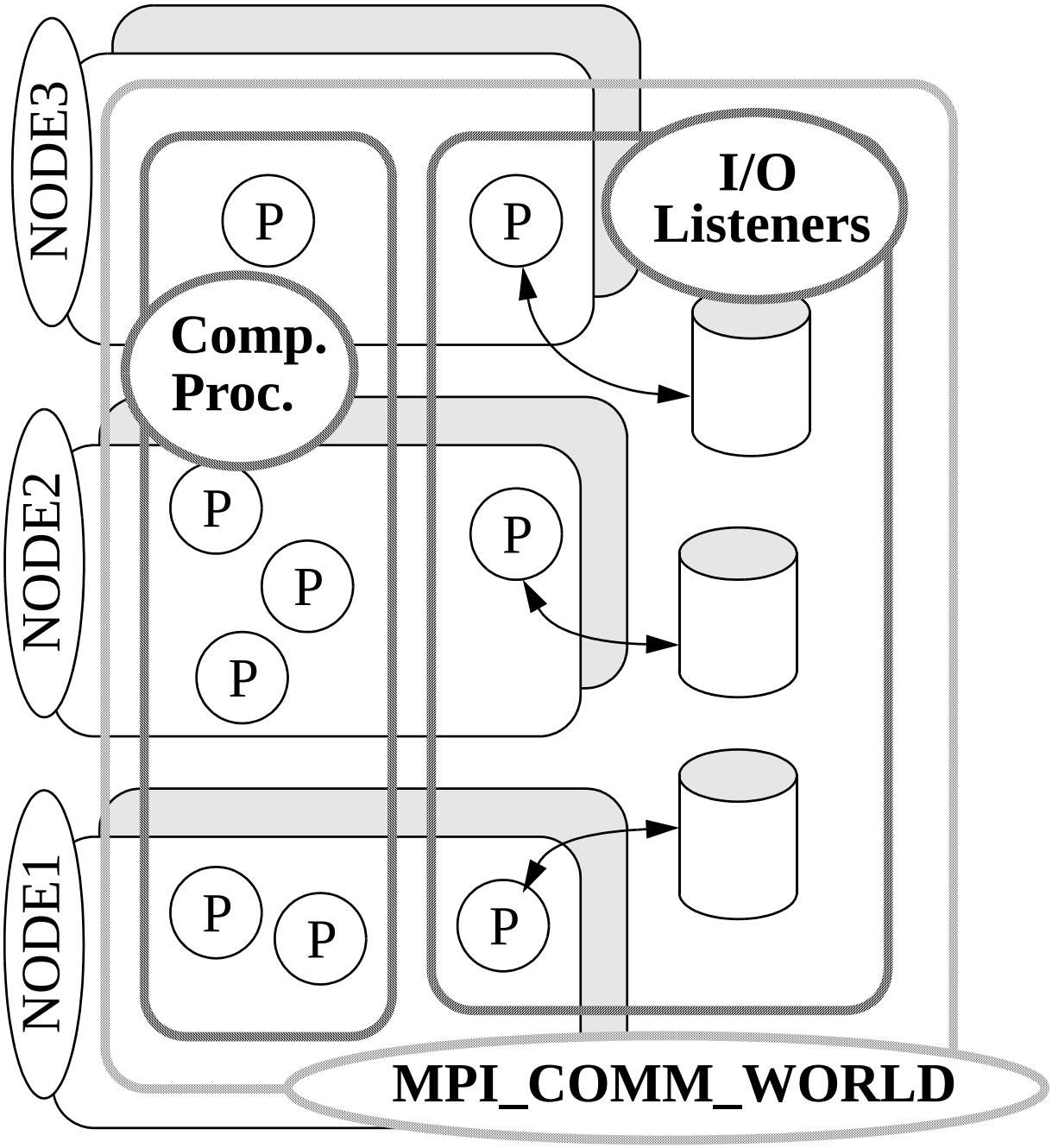

Data Exchange between
Compute Nodes and
I/O Listeners
Comp
Comp
Comp
I/O
Node
I/O
Node
I/O
Node
I/O
Node
I/O
Dev
I/O
Dev
I/O
Dev
I/O
Dev

Pg. 06

Initialisation of NAFS/MPI

- Fixing one MPI process per I/O node
- Running I/O listeners
- The rest processes grouping into new MPI communicator

pg. 07


NODE3
NODE2
NODE1
P
P
I/O
Listeners
Comp.
Proc.
P
P
P
P
P
P
P
P
MPI_COMM_WORLD

3.0 Optimisations in realisation

When user generates request for I/O access of the large contiguose piece of data within the logical file, cyclicaly striped across the I/O nodes, it is possible that this will generate single block I/O access per I/O node more than once. Especialy the number of single block access requests per I/O node will be high in the case of small stripe unit sizes.

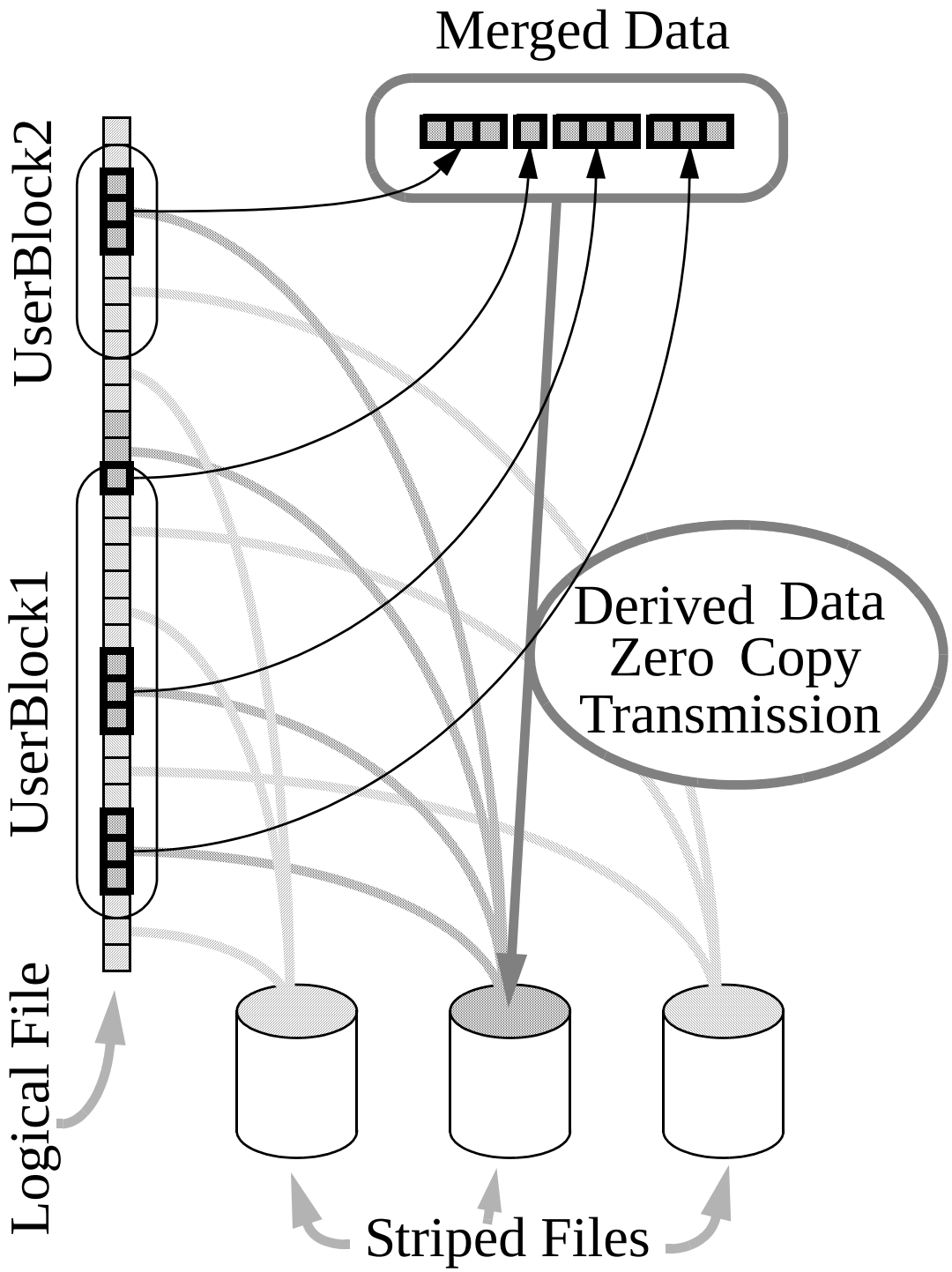
It is known that the cost of communication with transmission of set of small blocks are higher than the cost of transmission of large single block. So it is object of optimisation the gathering together the small blocks into the large one before making communication.

Another fragmentation level is the user multiple block access interface over the striping factor. So here in optimisation should be counted also this factor, especialy for the small sizes of the user blocks.

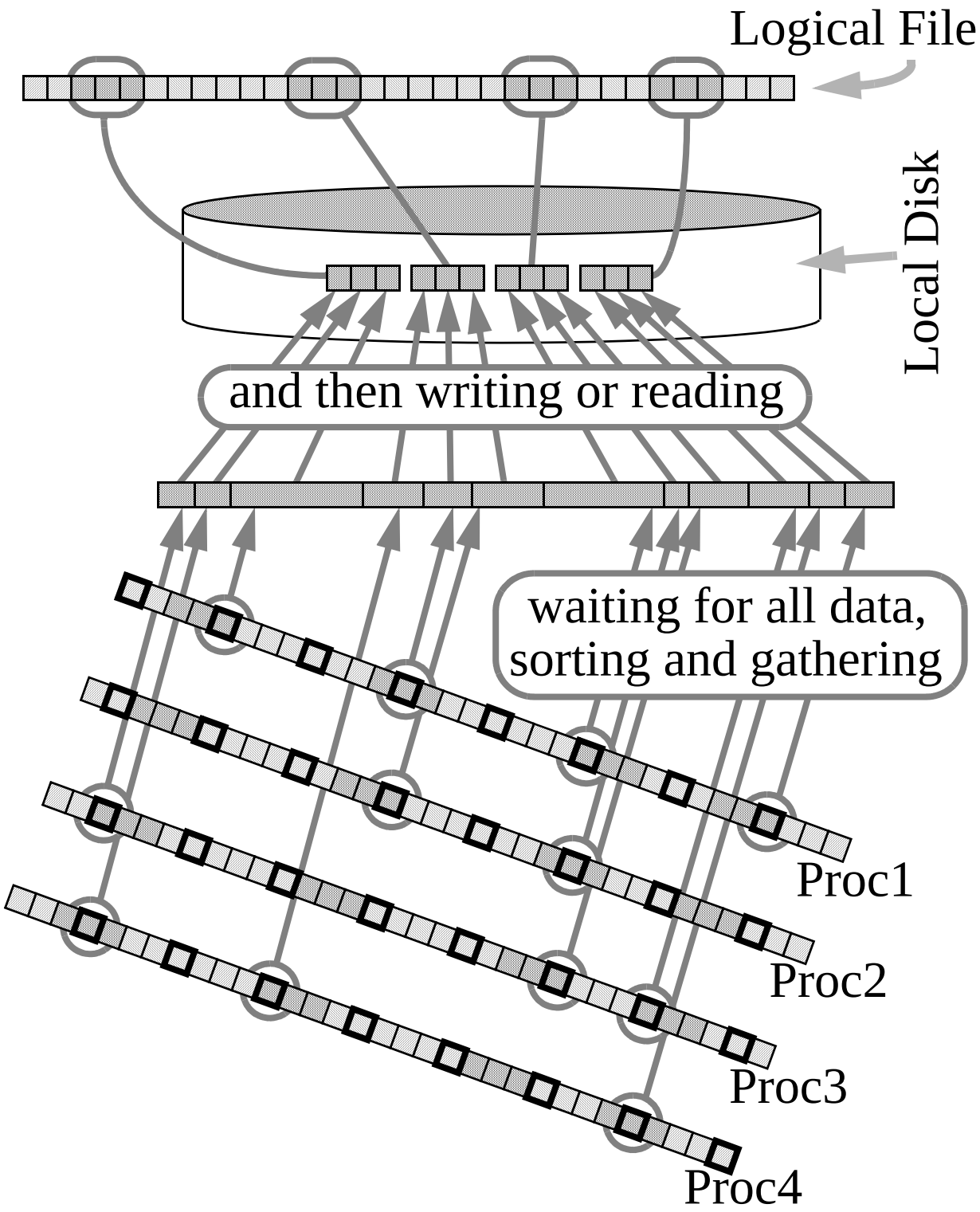
This two kinds of optimisations are realised in the current version.

In the collective communications important think is optimisation on the I/O node level. When you make collective I/O operation, then I/O nodes receiving the access requests from several compute nodes. It is known that contiguous large block I/O access are more efficient than random small block access. So optimisation is here the gathering of all of requests from the all compute nodes involved into the collective I/O operation. Then before making the actual I/O operations create from the set of small I/O requests gathered from different sources, a request containing as much as possible large and contiguous block of data necessary for I/O processing on the local device.

Pg. 08


Merged Data
UserBlock2
UserBlock1
Derived Data
Zero Copy
Transmission
Logical File
Striped Files

Pg. 09


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