Dynamic and Adaptive optimization techniques to enhance the performance of MPI applications by using **HECToR** and **Eddie** clusters.

Author: Rosa Filgueira Vicente
University of Edinburgh

edinburgh data-intensive research

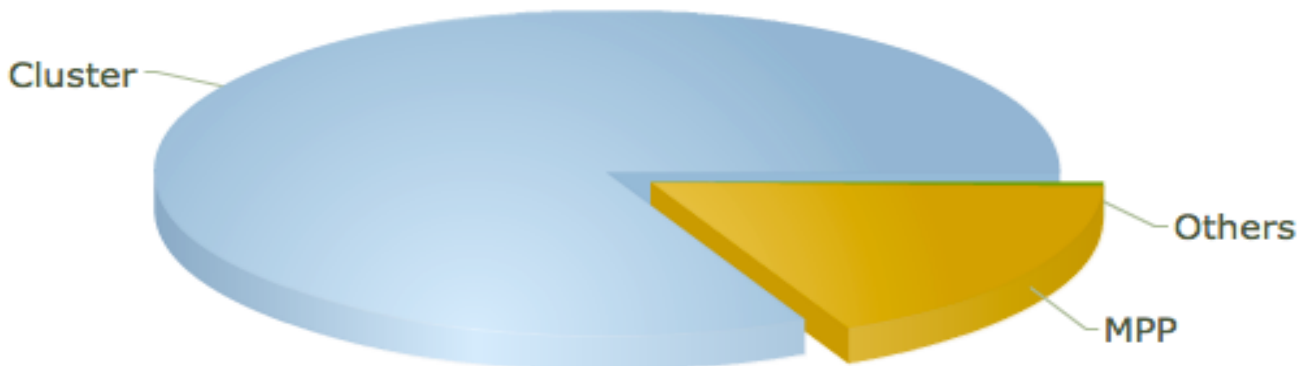
Summary

1. Problem description
2. Main Objectives
3. Strategies for improve the performance of collective I/O operation
4. Strategies for improve the performance communication operations
5. Evaluations

Summary

Problem description

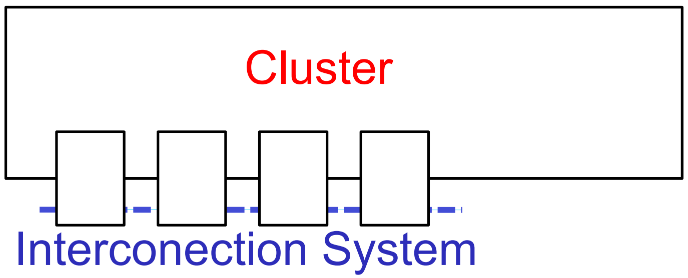
- Parallel computation on cluster has become the most common solution for HPC application.



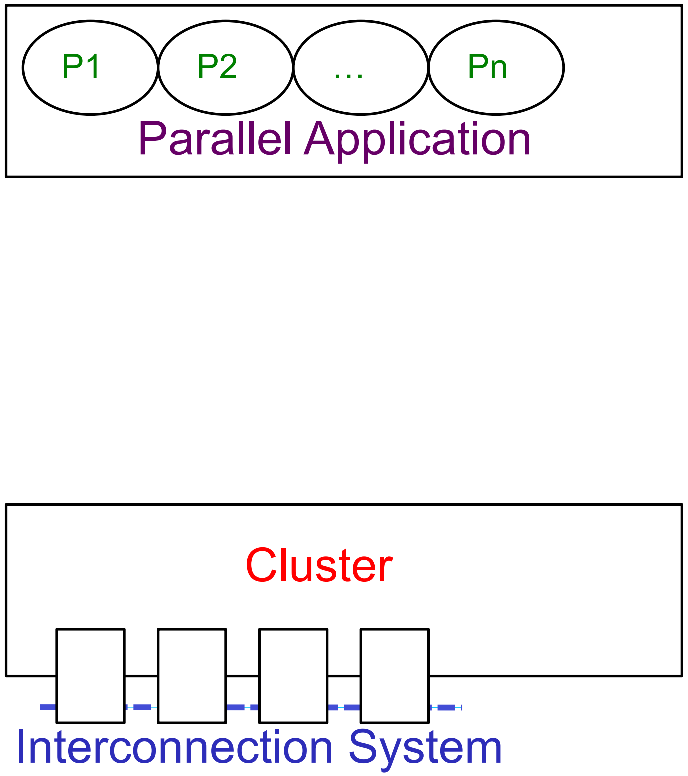
TOP 500
June2011

Problem description

- Cluster is a group of linked computers, working together closely thus in many respects forming a single computer.

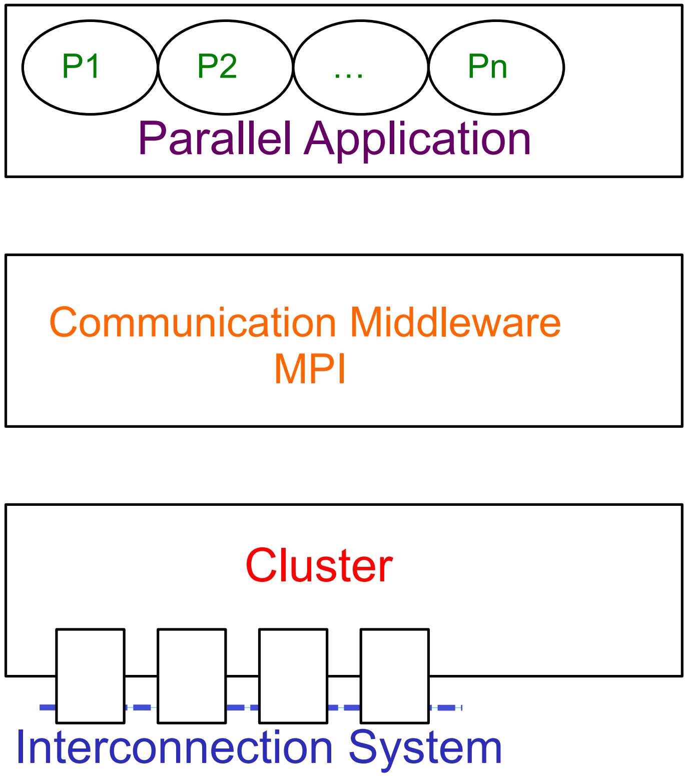


Problem description



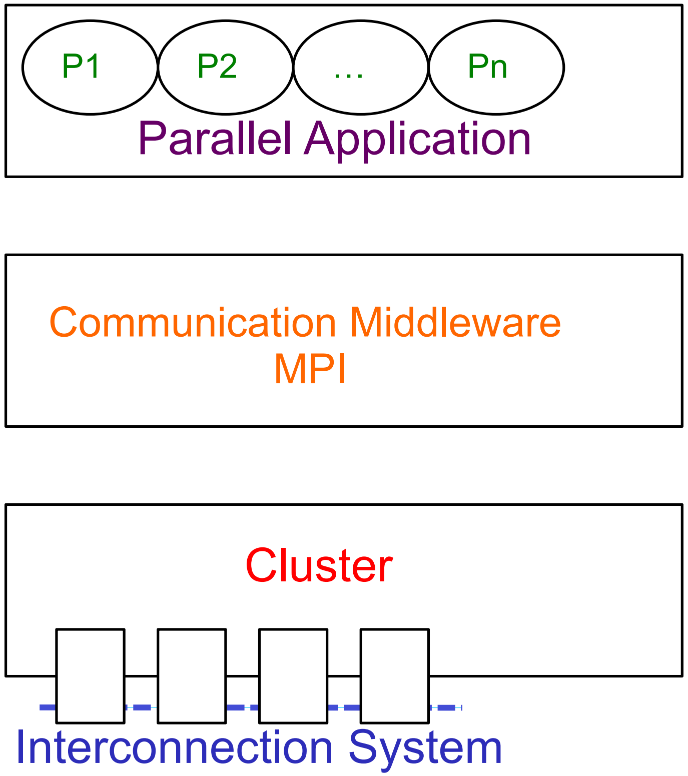
- On a cluster, one/many parallel applications could be running.

Problem description



- One of most communication middleware used is the standard **MPI** (Message Passing Interface).

Problem description

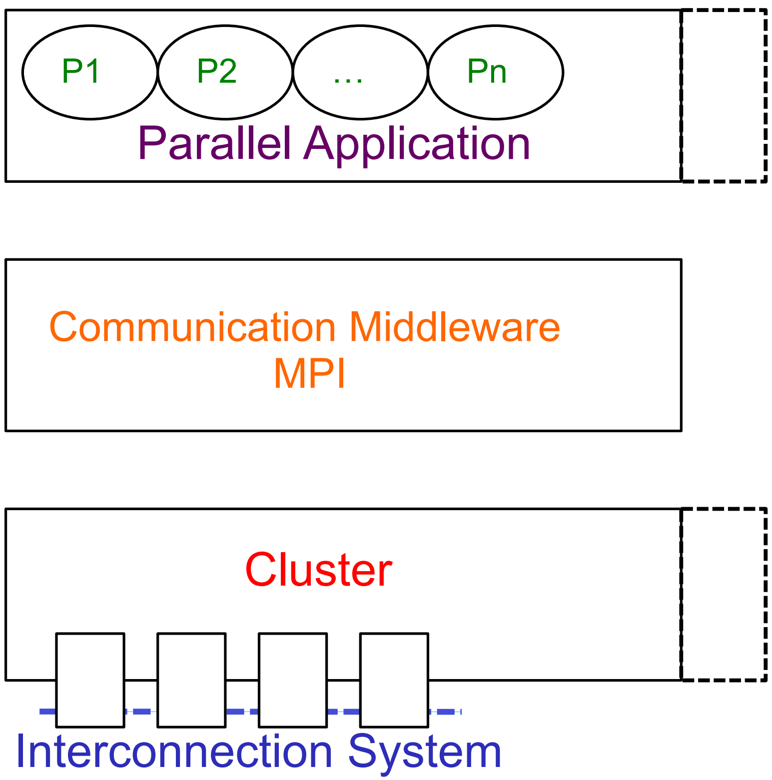


- One of most communication middleware used is the standard **MPI** (Message Passing Interface).
- Different implementations:
  - **MPICH**
  - OpenMPI
  - LAM
  - CHIMP-MPI …

Phd.Thesis:
MPICH 1.2

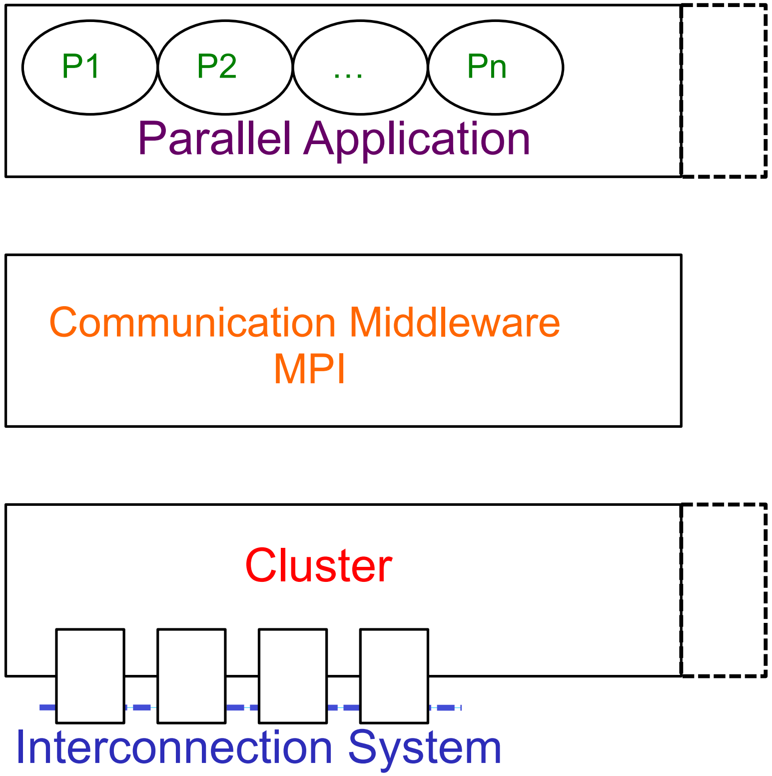
Now : MPICH 2

Problem description: trend



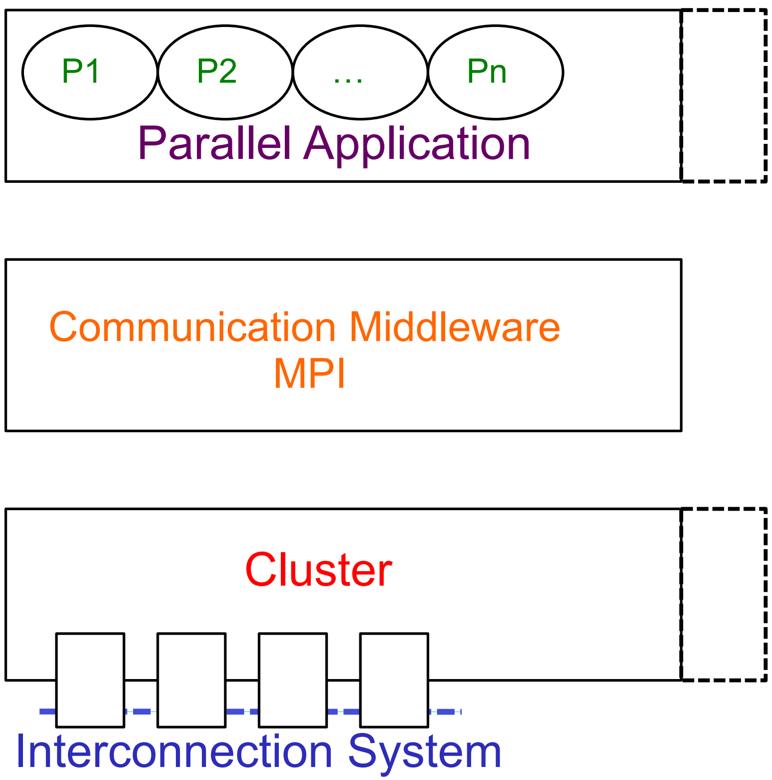
- Multicore processors provides a flexible way to increase the computation capability of cluster
- System performance may be improved with multicore but bottleneck from other components could reduce the scalability.

Problem description: Bottleneck



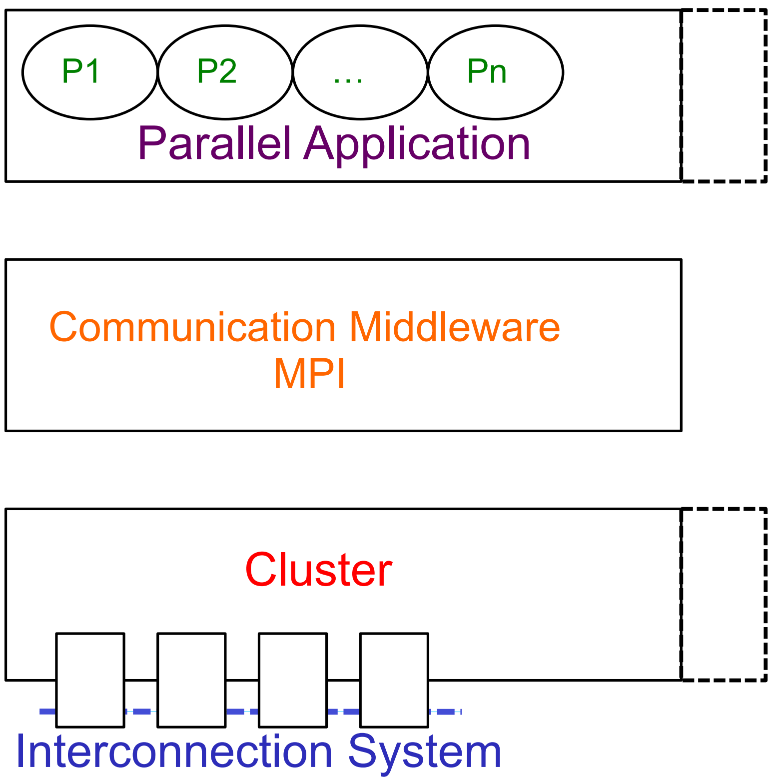
- Bottleneck :
  - I/O subsystem
  - Communication subsystem

Problem description: Bottleneck



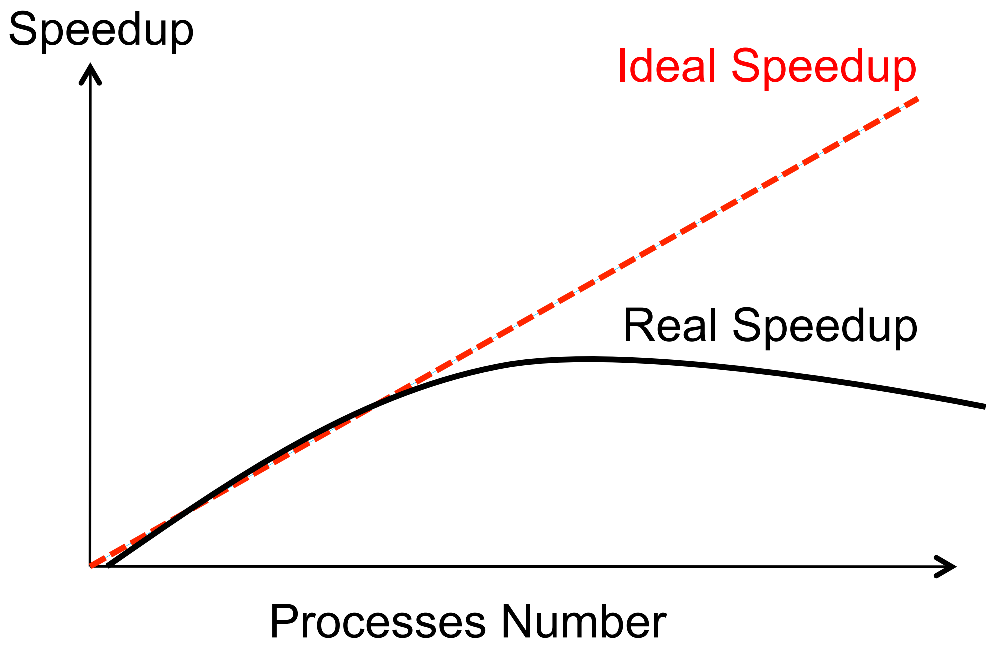
- I/O Bottleneck:
  - I/O requests initiated by multiple cores and besides when non-contiguous disk accesses is used

Problem description: Bottleneck



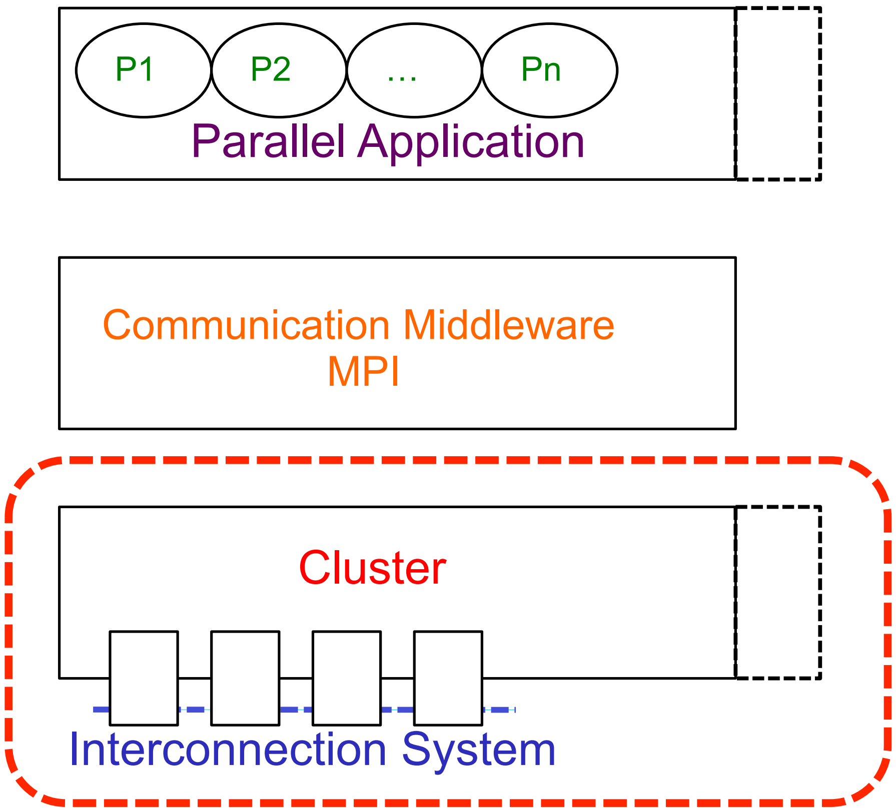
- Communication Bottleneck:
  - Network used are very fast and low latency.
  - Computational capability in multicore very high.
  - The frequency of message increase a lot → insufficient the increase of the bandwidth and latency

Problem description: Bottleneck



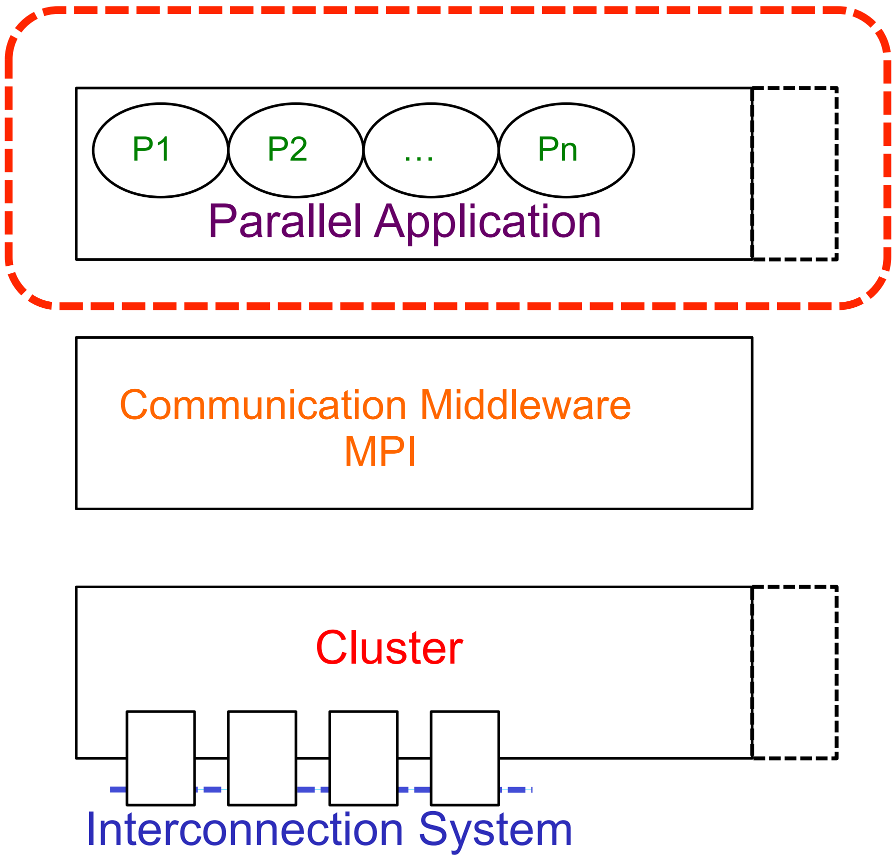
- Communication and I/O saturation:
  - Scalability problem
  - Performance problem

Problem description: possible solutions



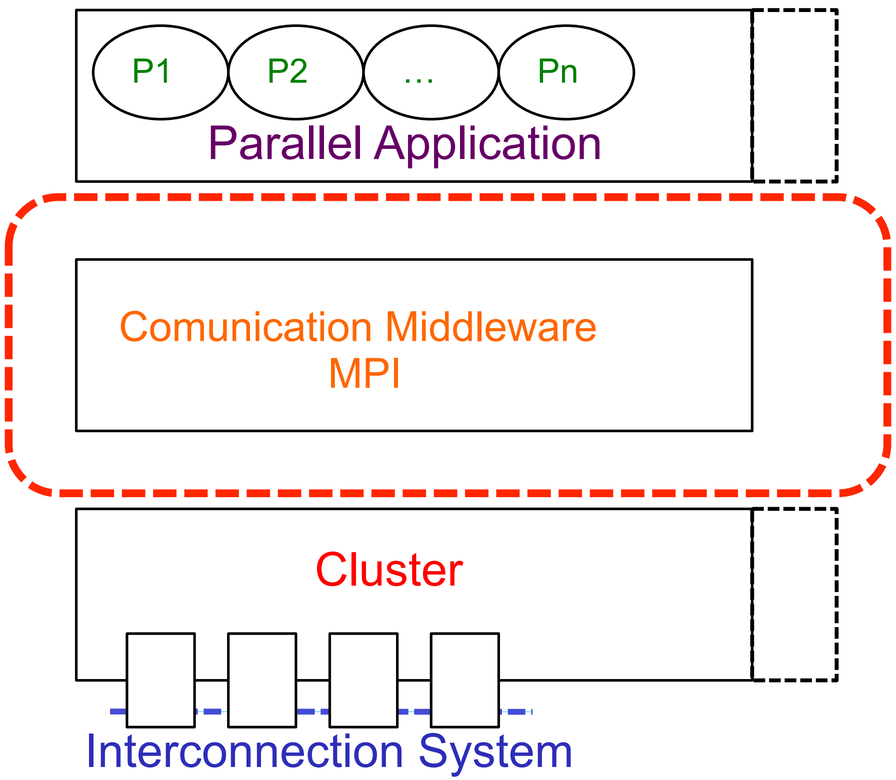
- 1) Improve network
  - Expensive.
  - Limited to current technology

Problem description: possible solutions



- 2) Improve the applications:
  - More effort in the design
  - Not always possible
  - The improvement affects few users

Problem description: possible solutions

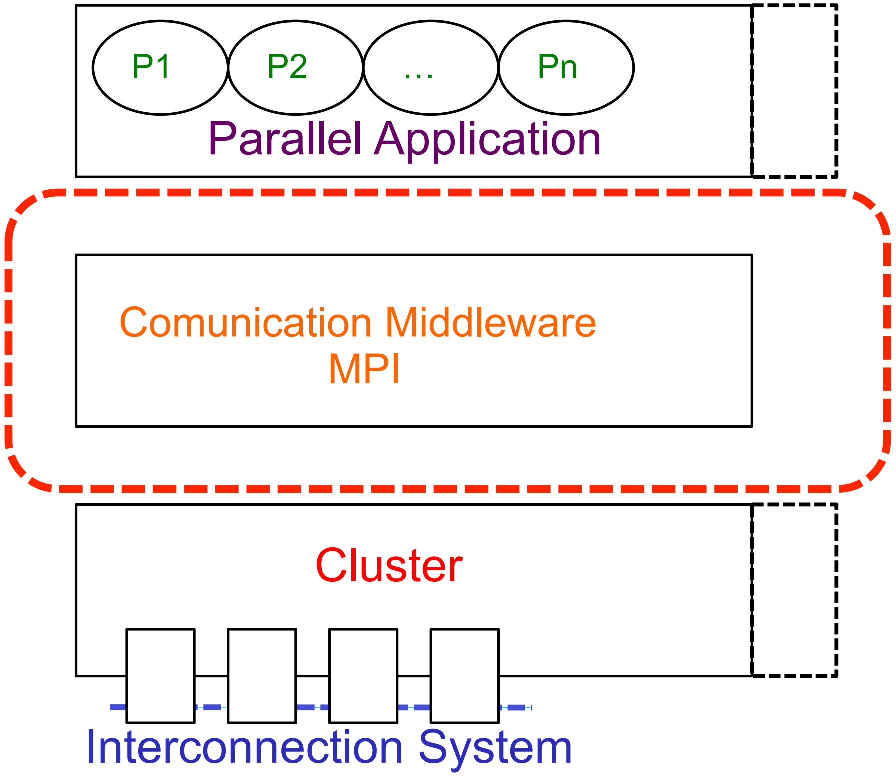


- 3) Improve the communication middleware
  - Portability
  - Greater user benefited
  - Lower Cost
  - Transparent:
    - Users
    - Applications

Summary

1. Problem description
2. Main Objectives
3. Strategies for improve the performance of collective I/O operation
4. Strategies for improve the performance of communication operations
5. Evaluations

Main objetives



- Improve the scalability and the performance of MPI based applications executed on Multicore cluster
- How? Improving the Middleware MPI

Specific objectives

1. Reduction of the number of communications in collective I/O operations:
   a) Dynamic and Adaptive I/O aggregator pattern

2. Reduction of transferred data volume in communications.
   a) Compression techniques by using message passing interface profiling (PMPI)

Summary

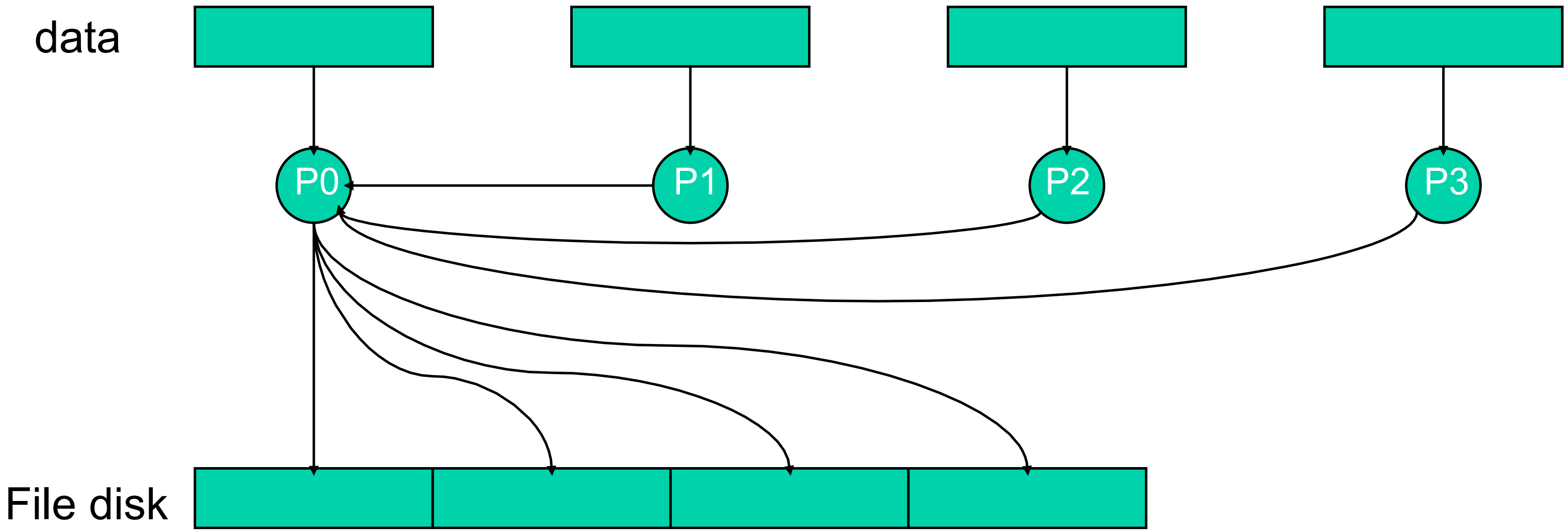
1. Problem description
2. Main Objectives
3. Strategies for improve the performance of collective I/O operation
4. Strategies for improve the performance of communication operations
5. Evaluations

Strategies for improve the performance of collective I/O operation

- Parallel scientific applications generate a lot of data to write/ read in/from disk.
- Access pattern:
  - Sequential access
  - Individual access
  - Collective access

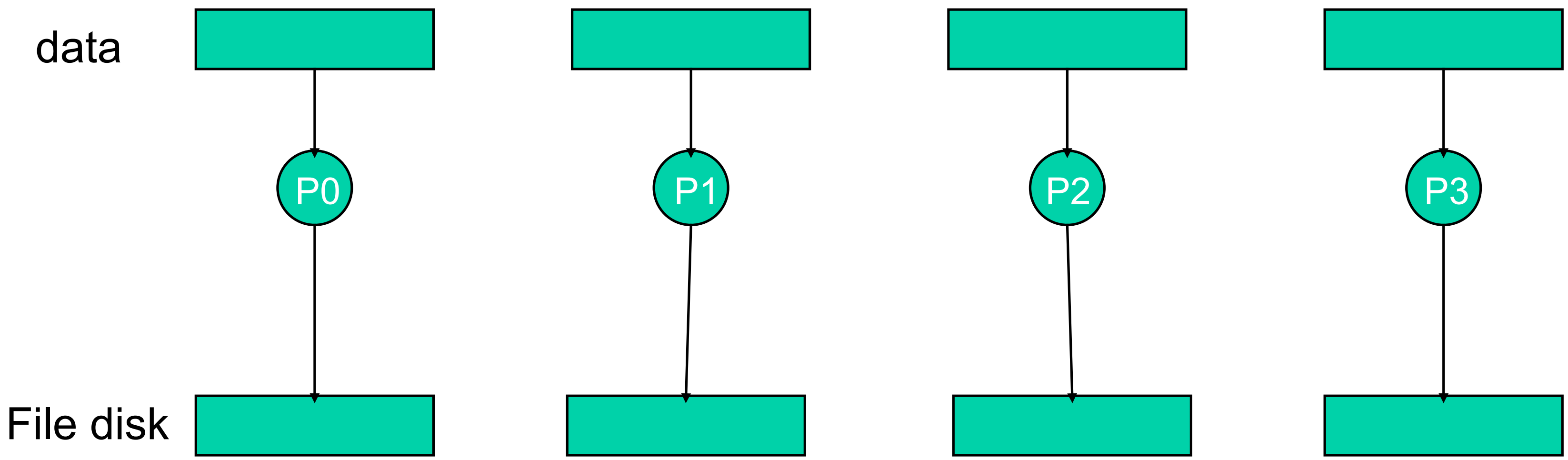
Sequential I/O

- All processes send data to one process (usually process 0), and this process writes it to the file



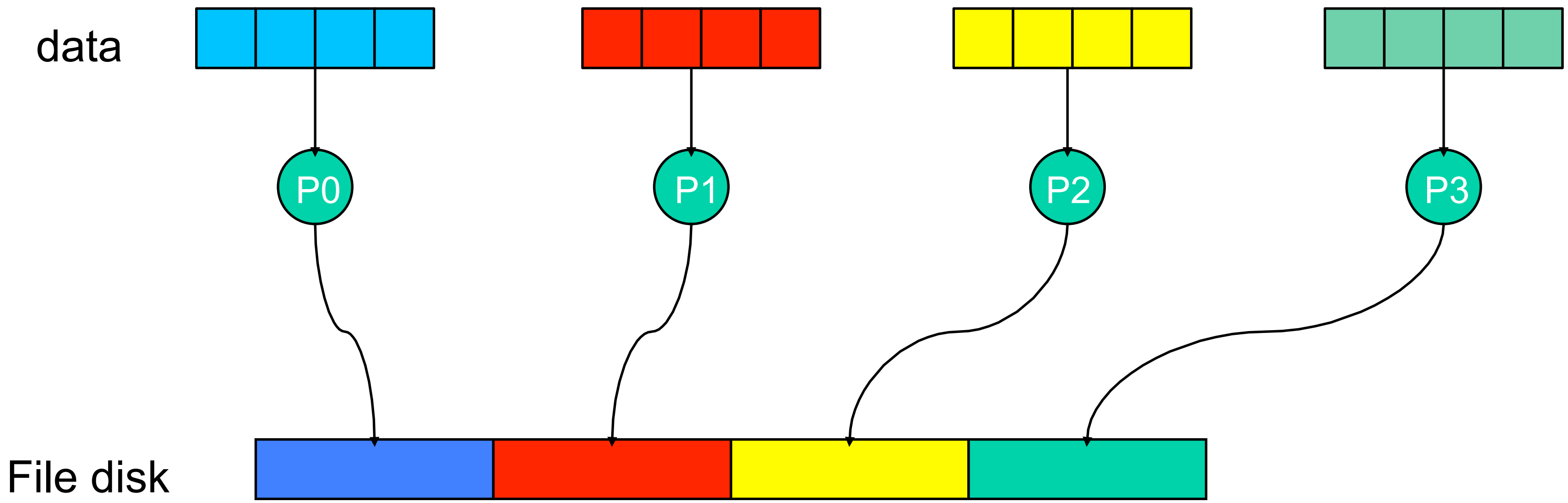
Individual I/O

- Each process writes to a separate file



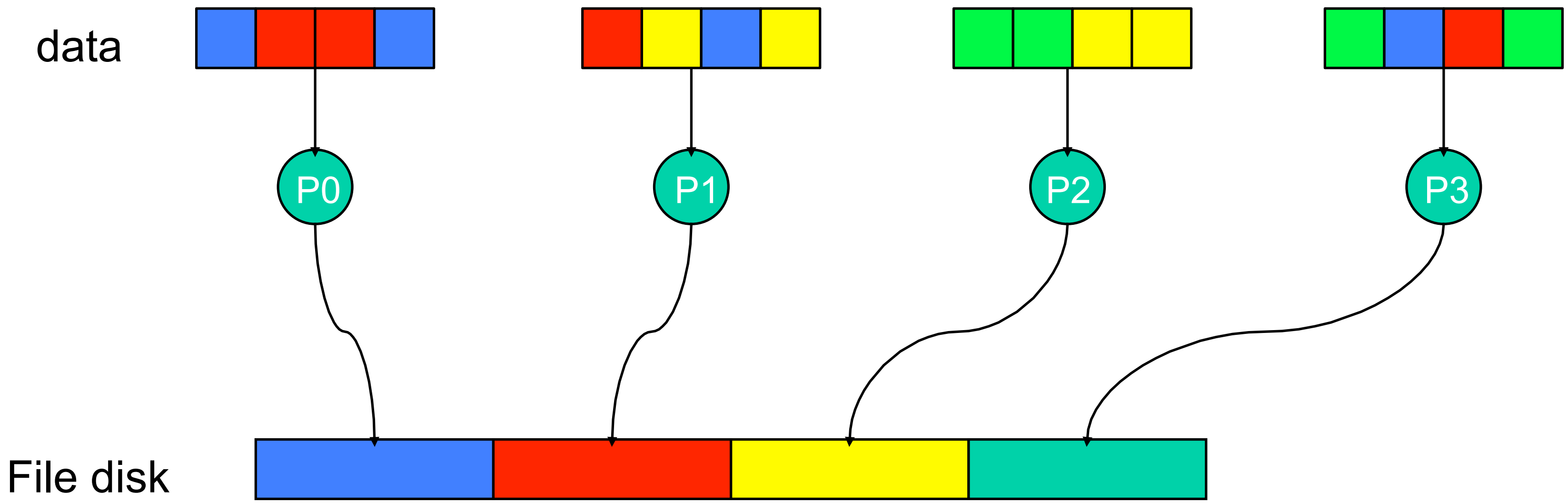
Collective I/O

- Processes write to shared file
- Contiguous data
- Two Phase I/O technique



Collective I/O

- Processes write to shared file
- Non-Contiguous data
- Two Phase I/O technique



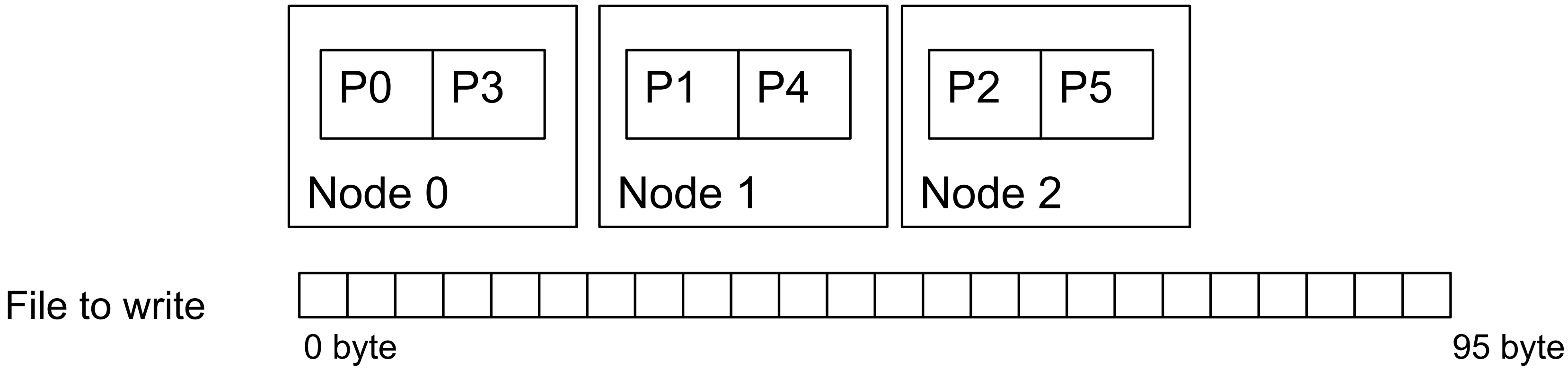
Two_Phase I/O

- Two-Phase I/O phases:
  - Shuffle: aggregate data into contiguous buffers.
  - I/O: transfer contiguous buffer to file system.
- Before these two phases:
  - File region is divided into equal contiguous regions
    - Called File Domains (FD).
  - Each FD is assigned to a subset of compute nodes (aggregators).
    - Each aggregator is responsible for transferring all data from its FD to the file system.
- Cause of inefficiency: The assignment of FD to aggregators is independent of data distribution.

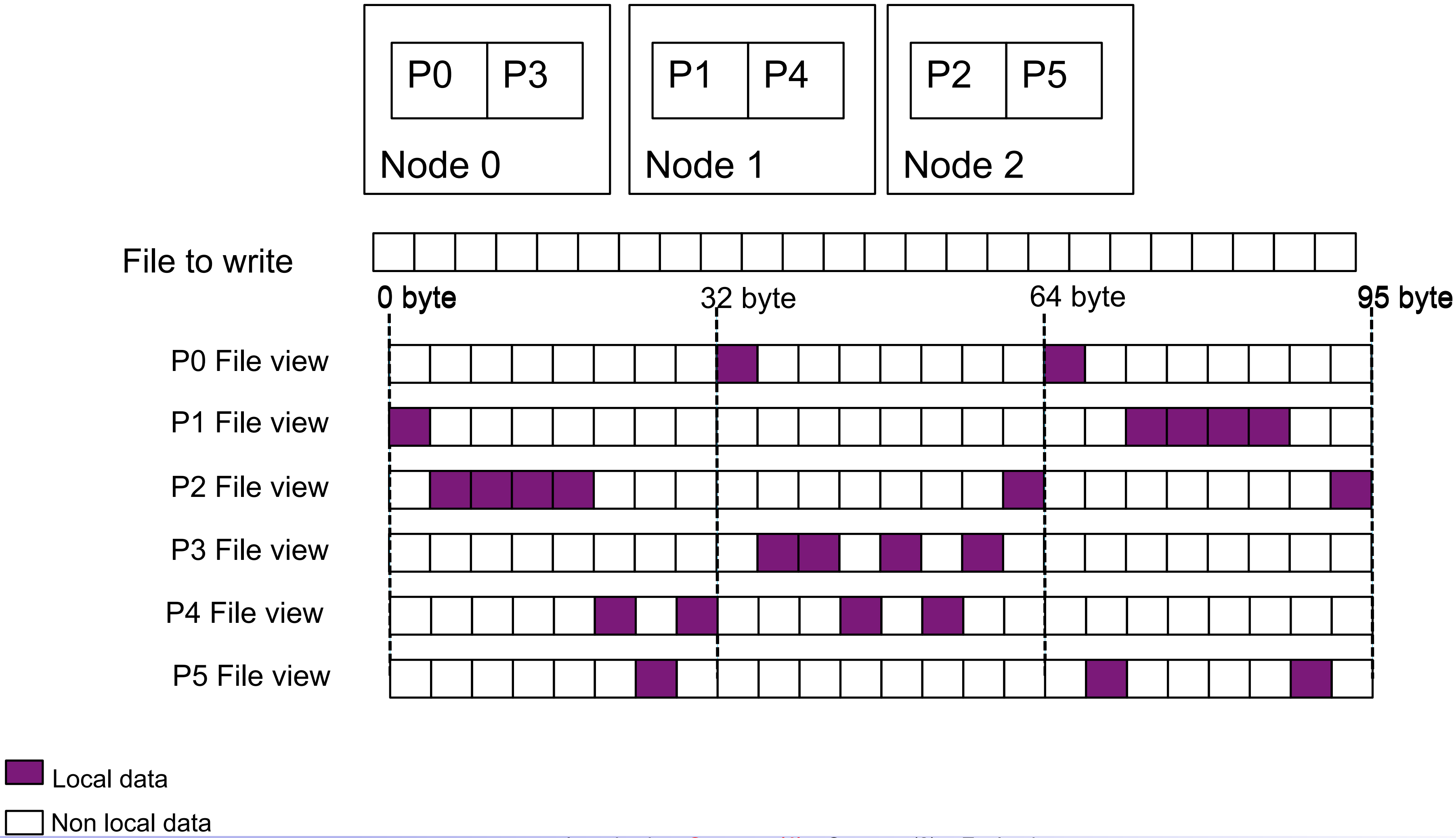
Two_Phase I/O "problem"

- The assignment of FD to aggregators is fixed.
- Independent of data distribution.
- Default aggregator Pattern:
  - So many aggregator as nodes.
  - Create an array of nodes, ordered by rank (process identifier in MPI), and assign each aggregator one process.
  - When there are more than process per node:
    - The process with the smallest rank is chosen as aggregator.

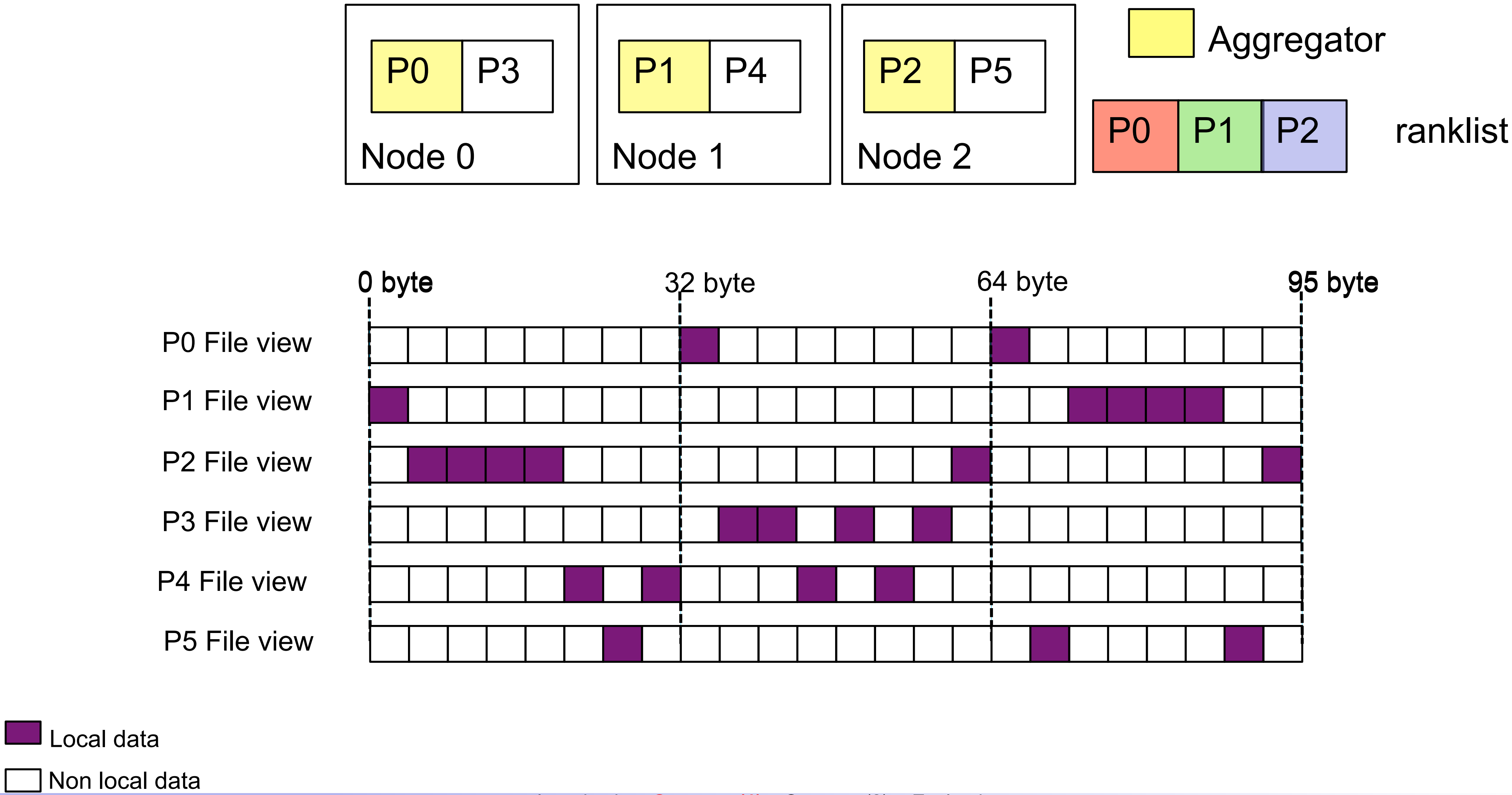
Example of Two-Phase I/O



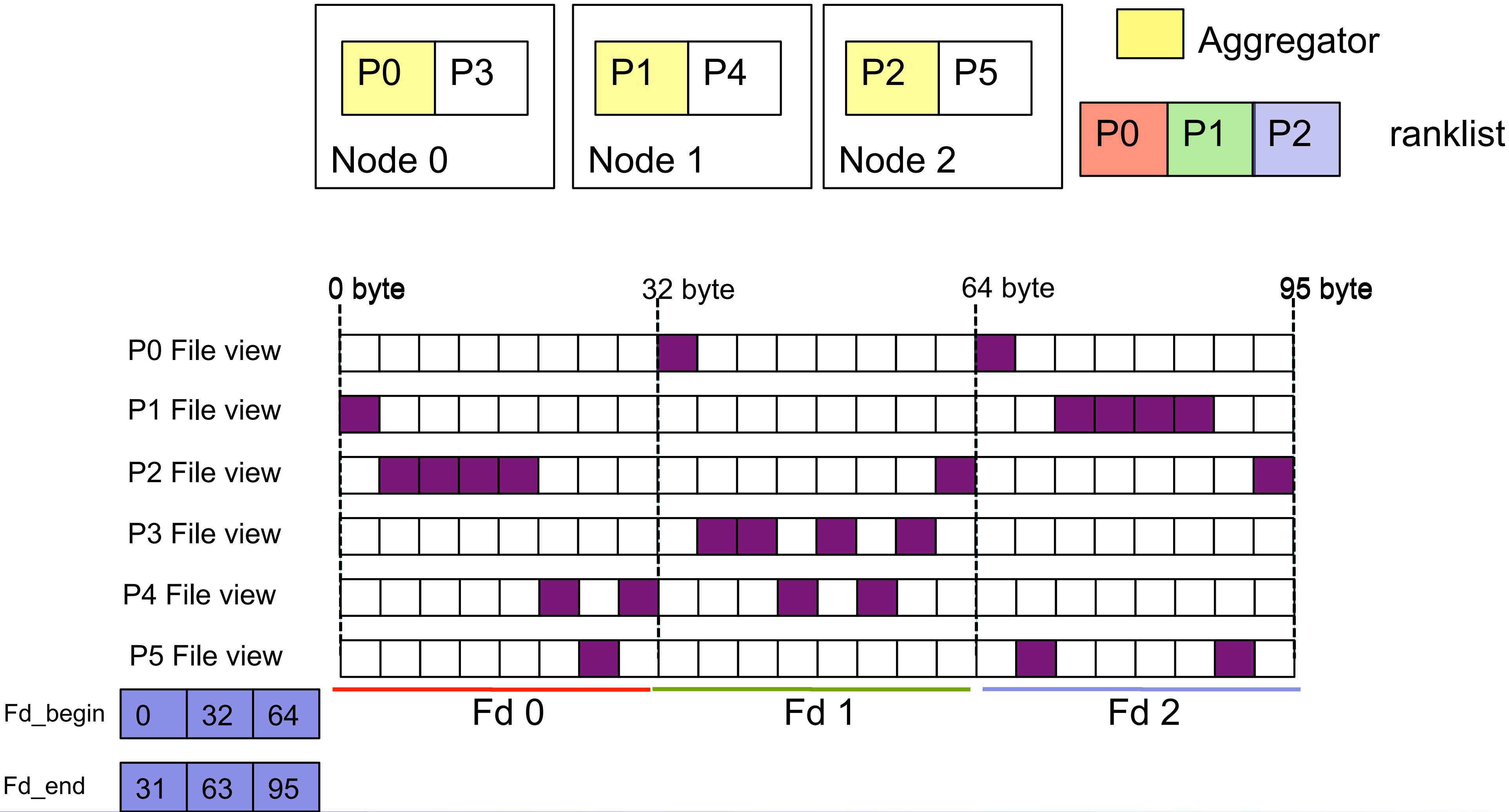
Example of Two-Phase I/O



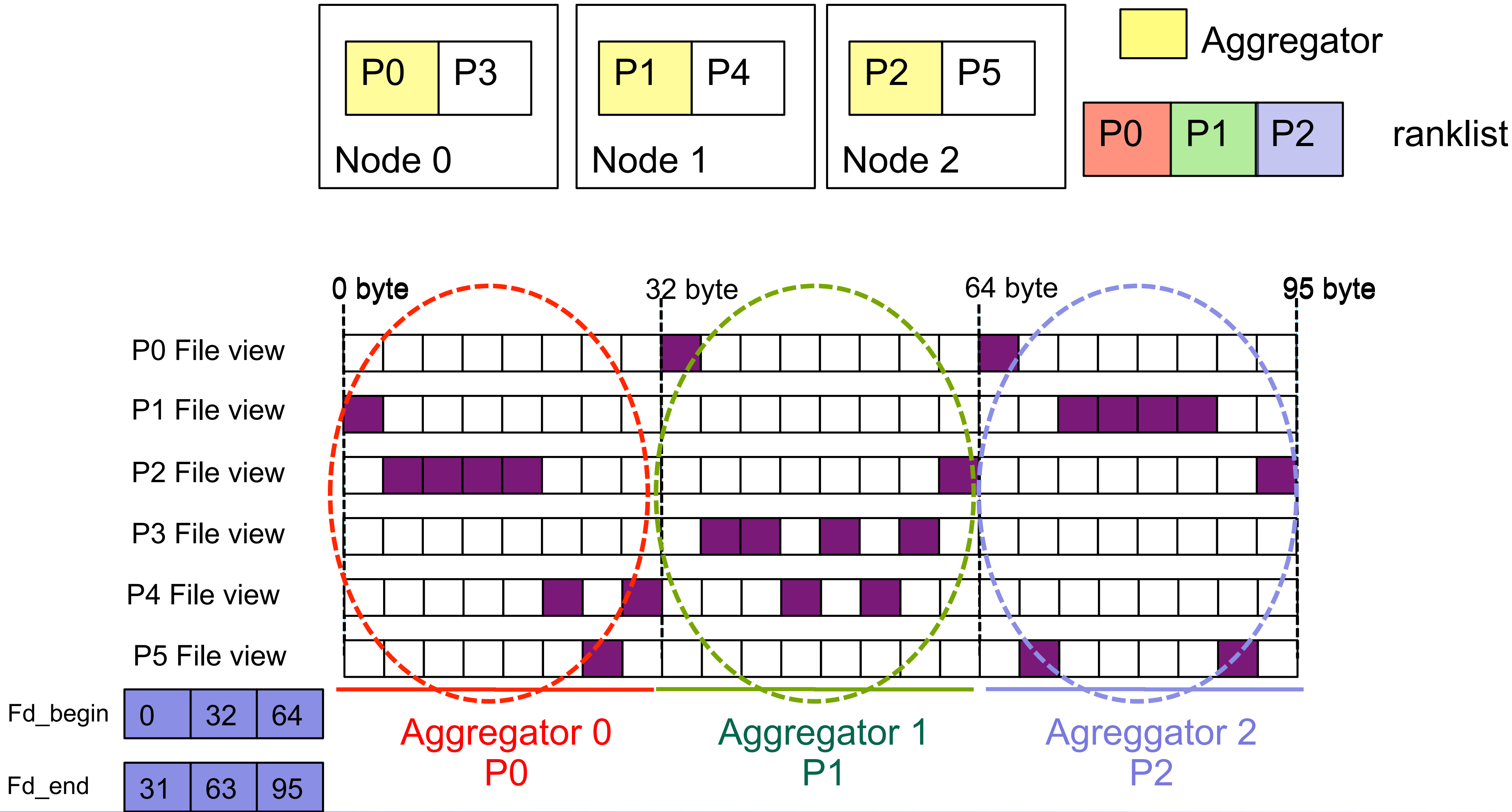
Example of Two-Phase I/O



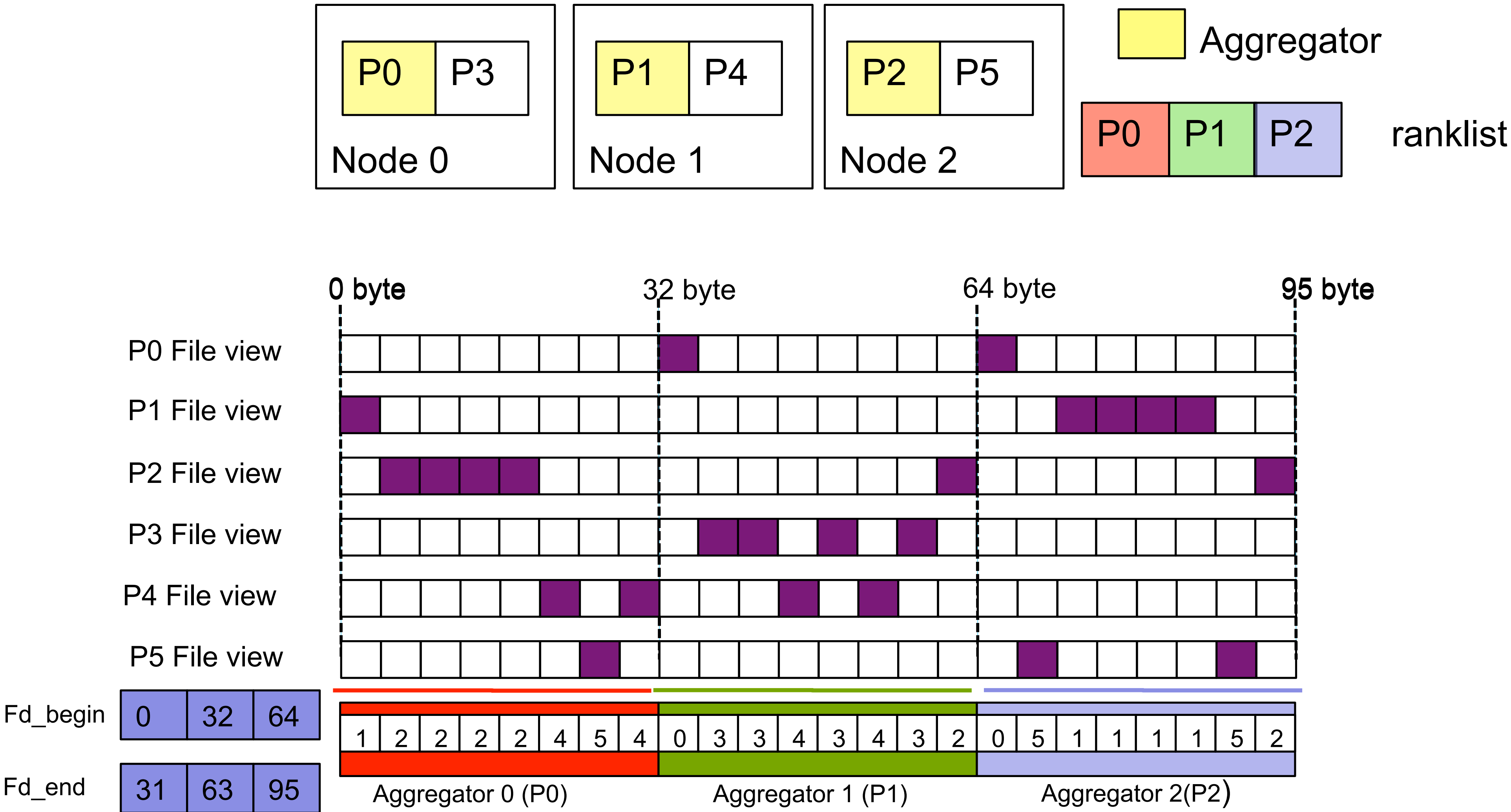
Example of Two-Phase I/O



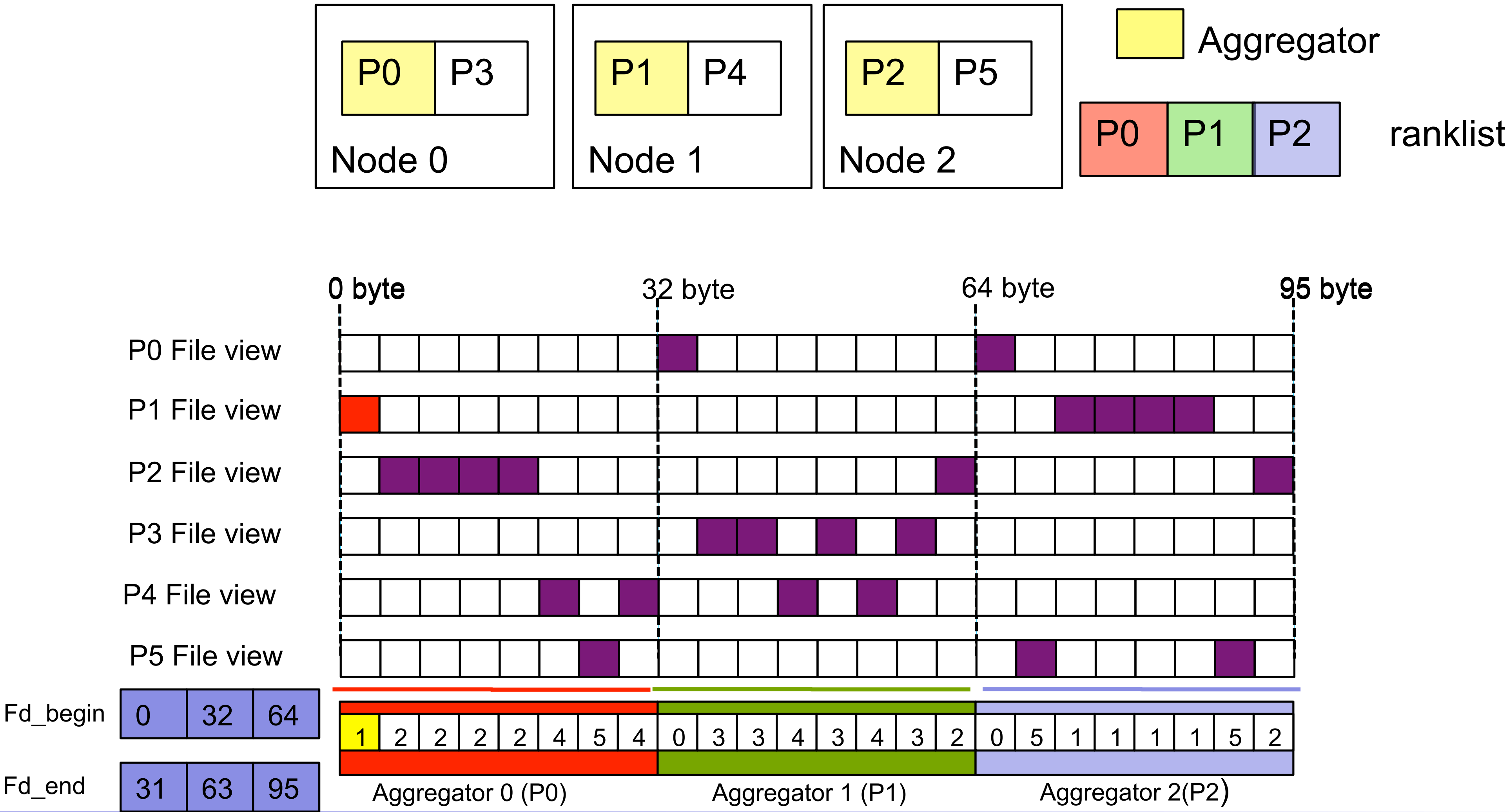
Example of Two-Phase I/O



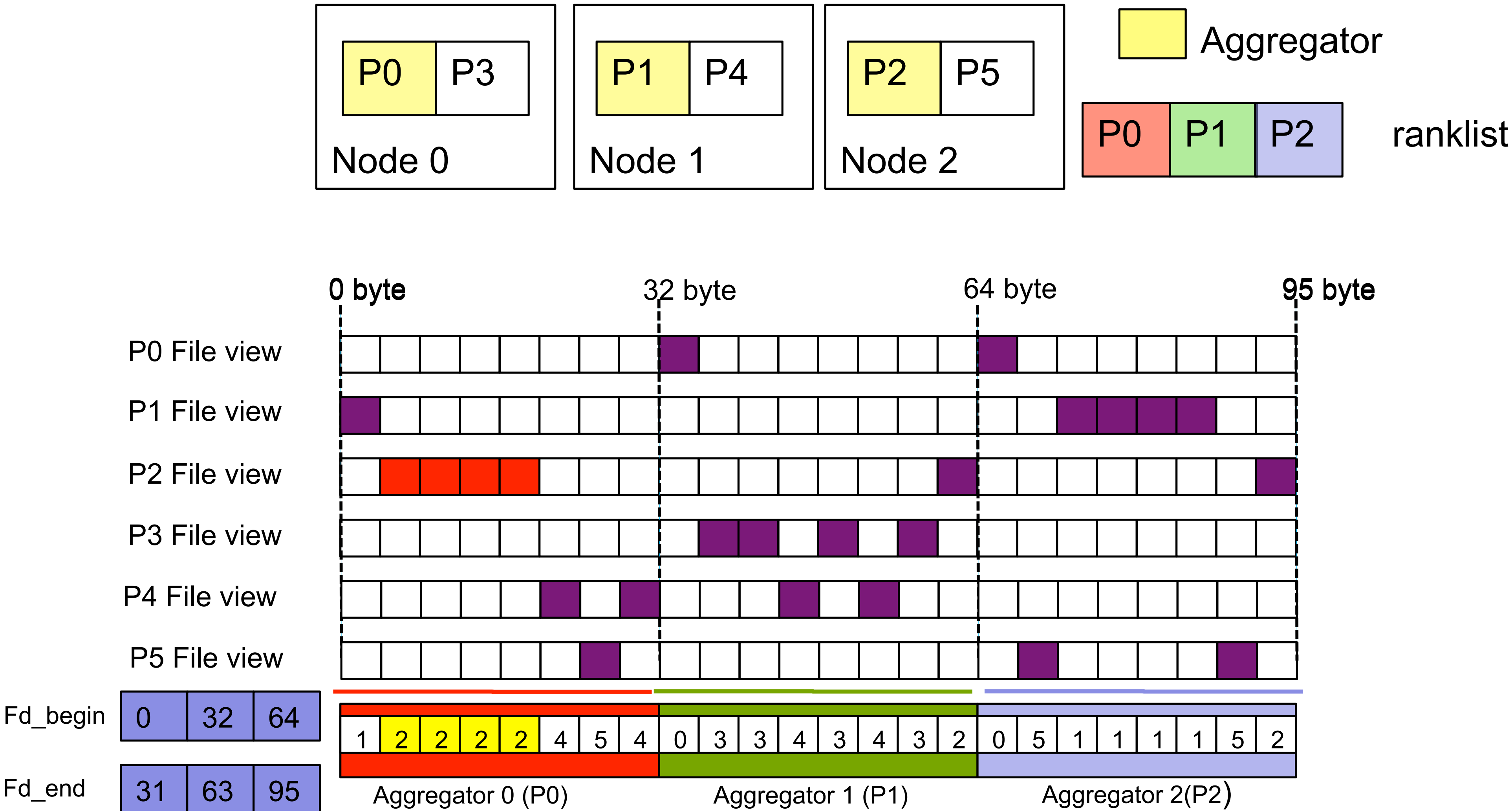
Example of Two-Phase I/O



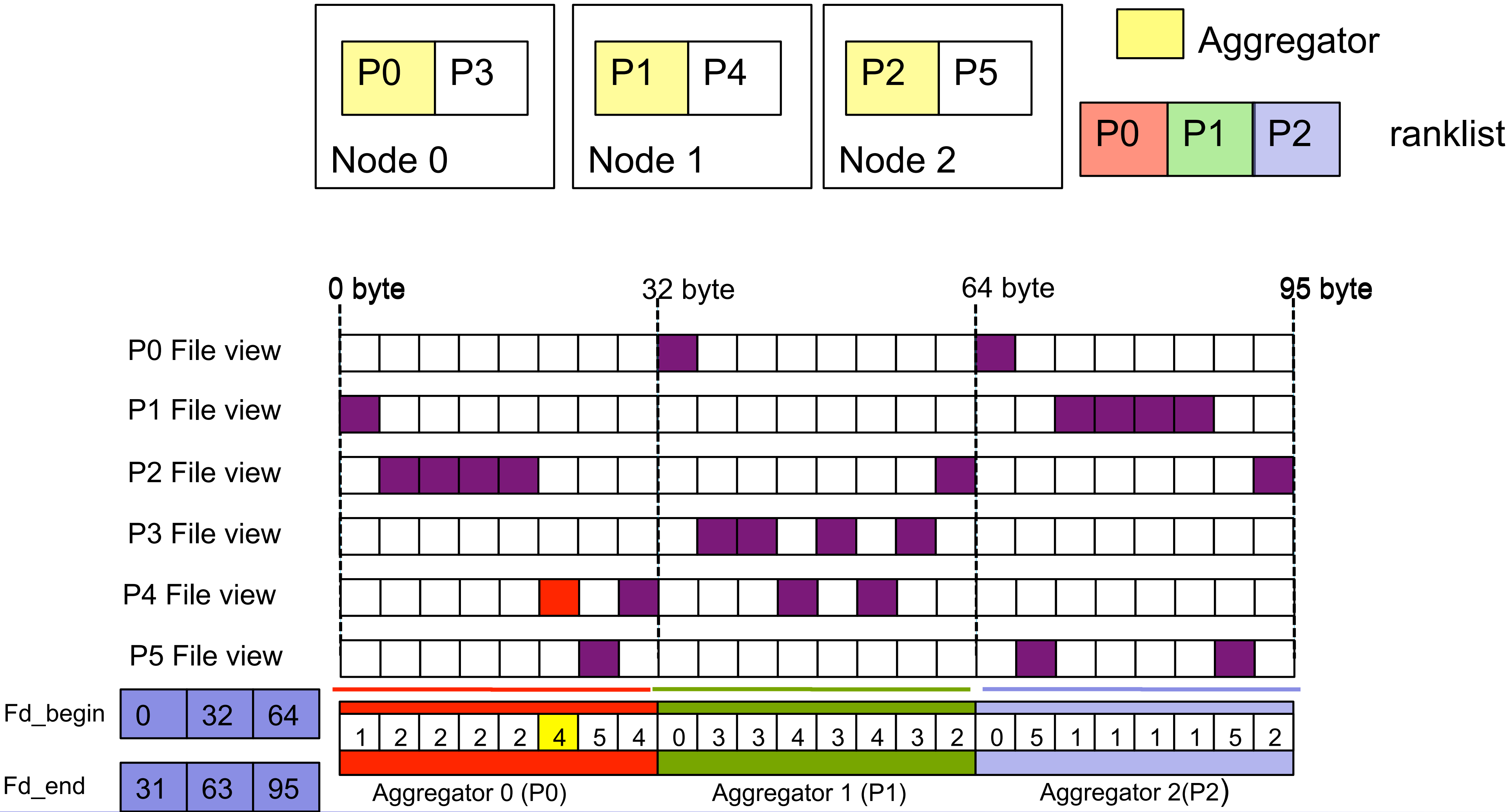
Example of Two-Phase I/O



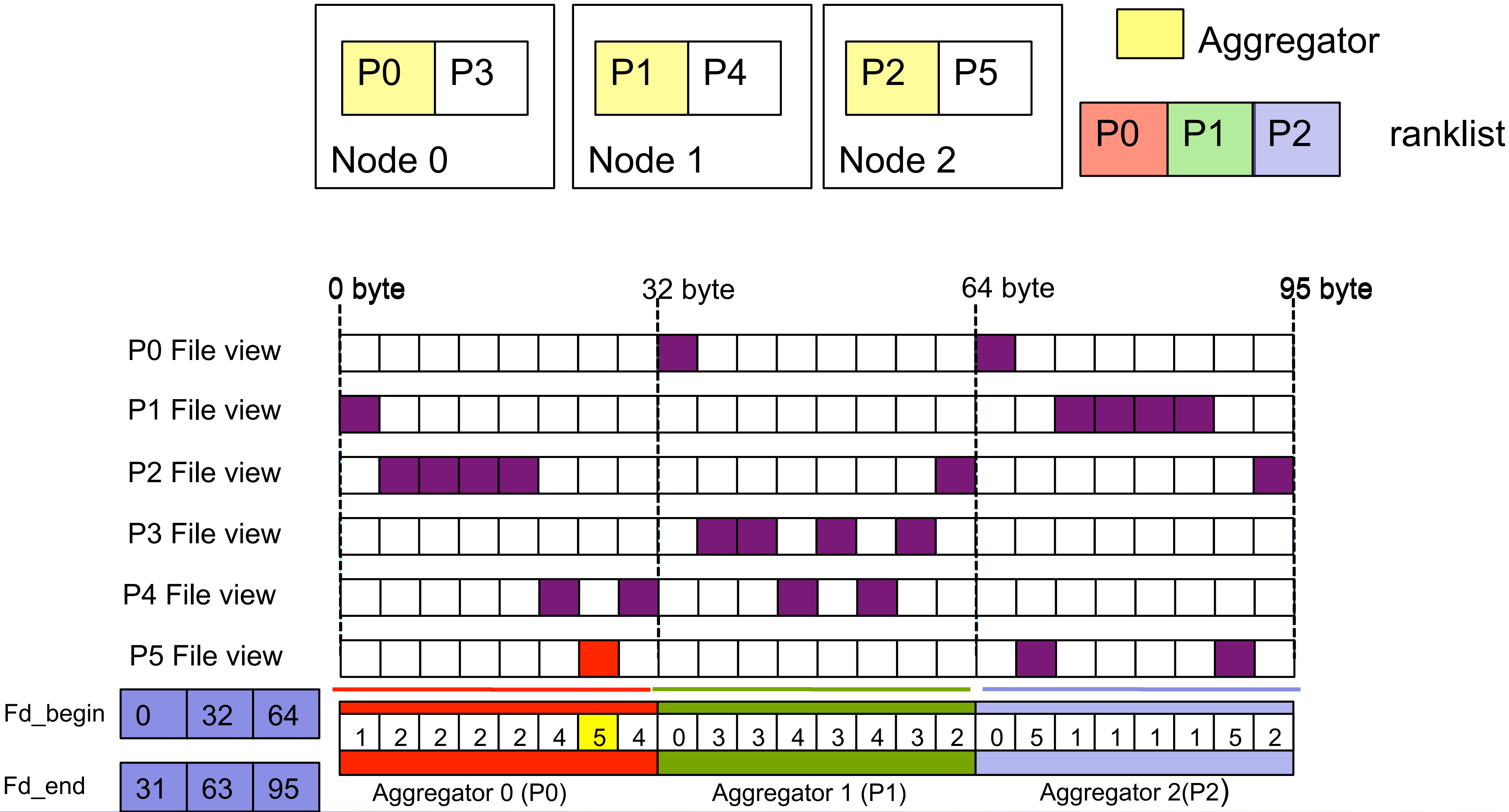
Example of Two-Phase I/O



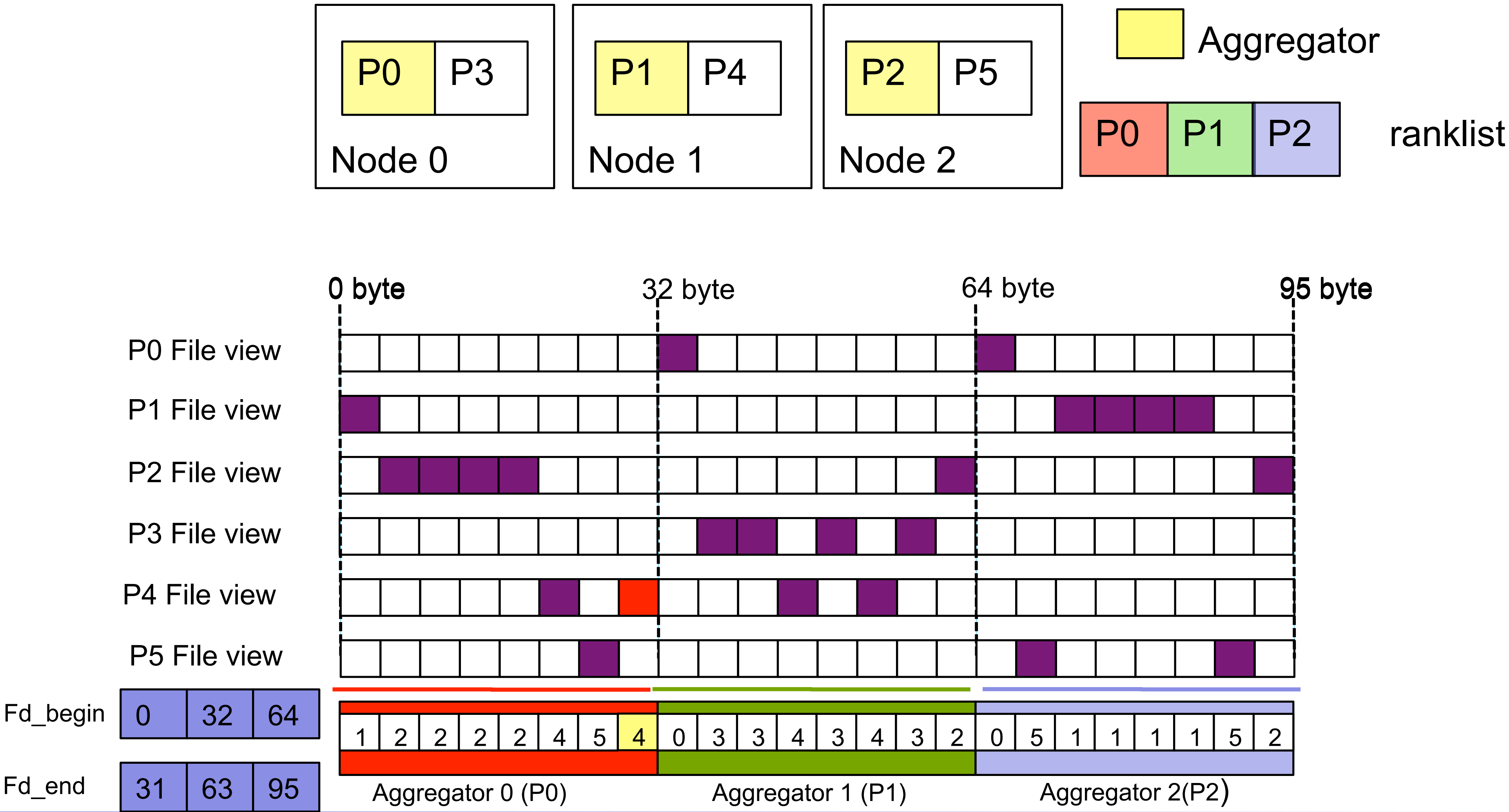
Example of Two-Phase I/O



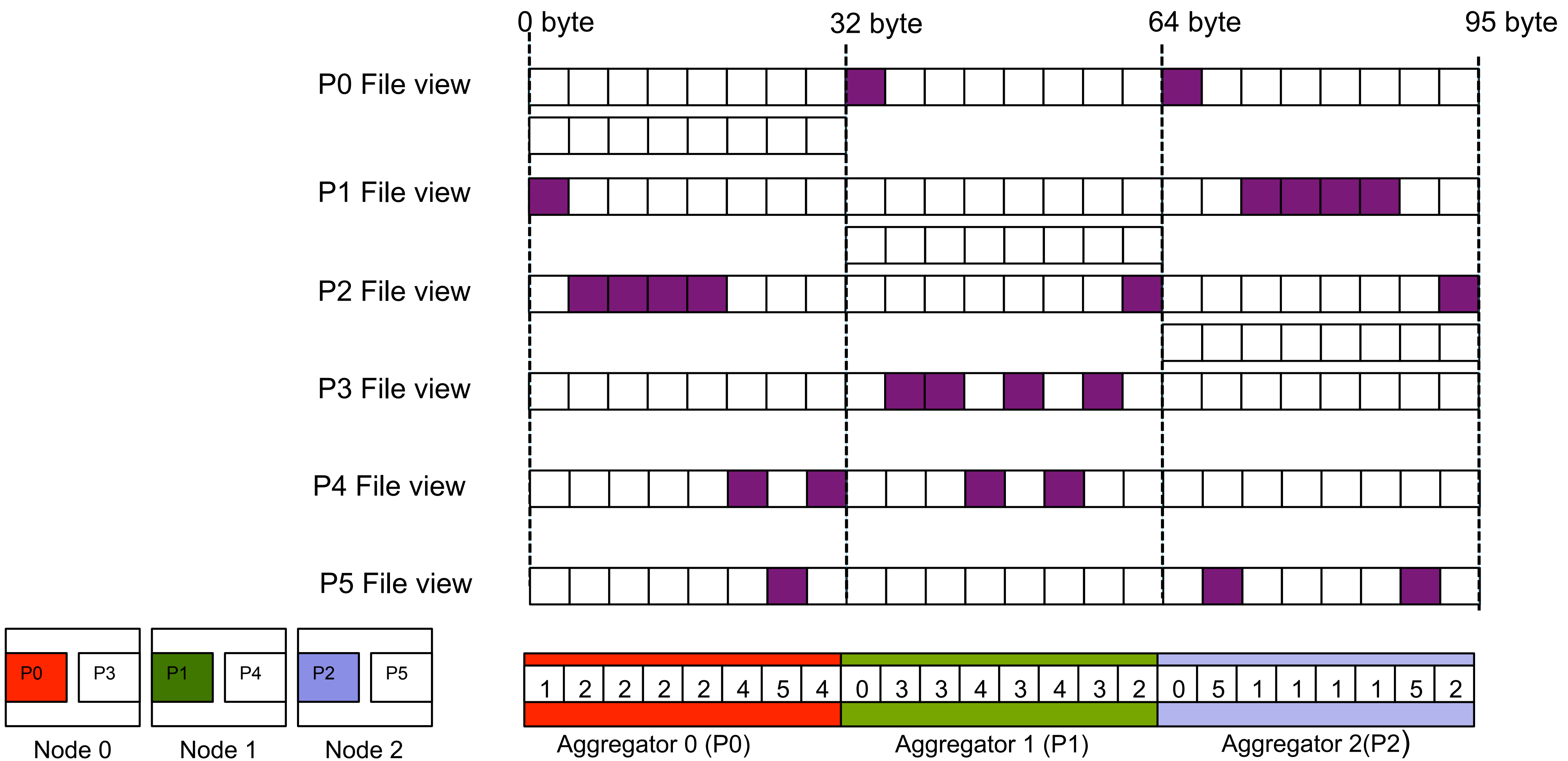
Example of Two-Phase I/O



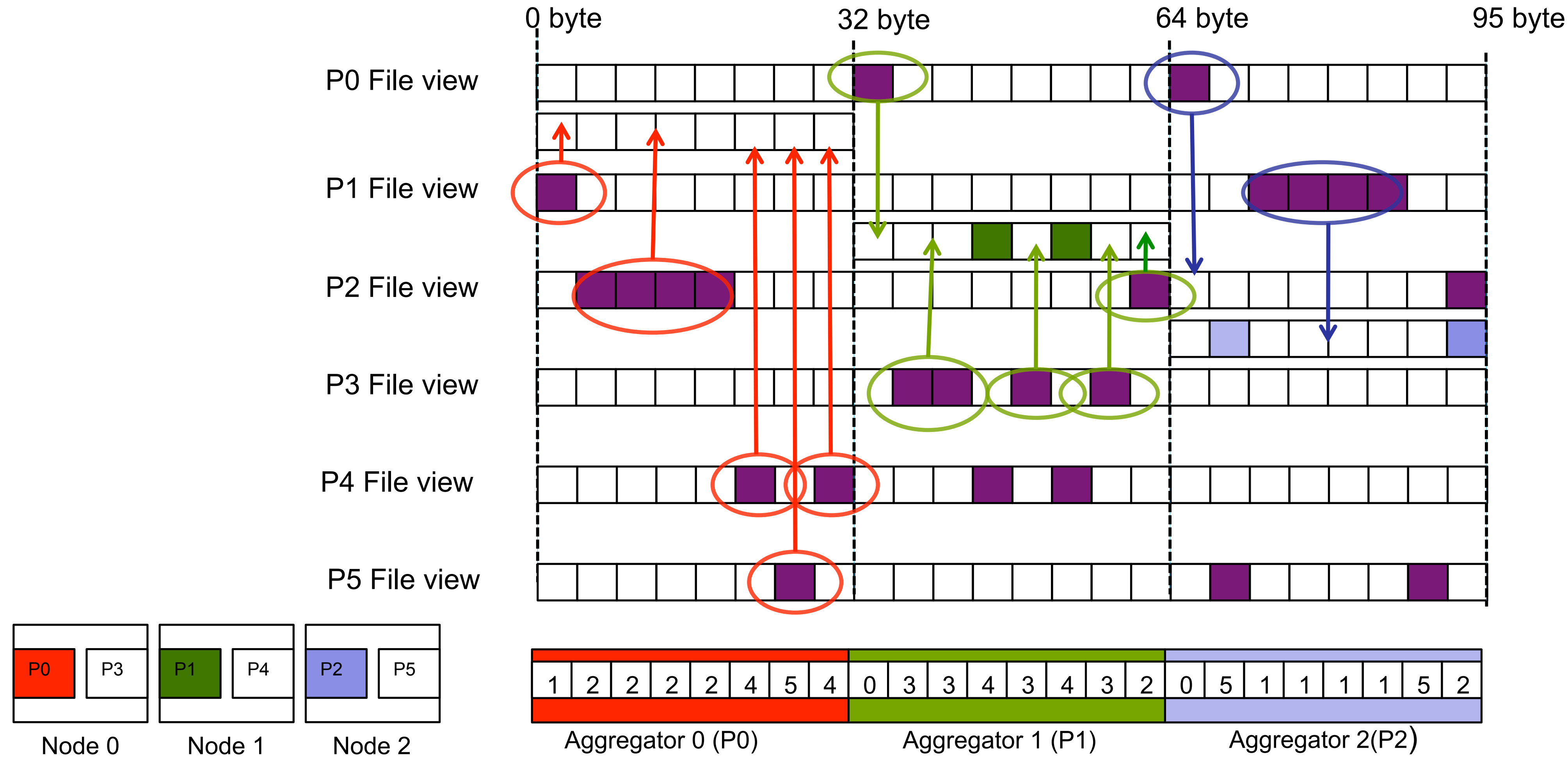
Example of Two-Phase I/O



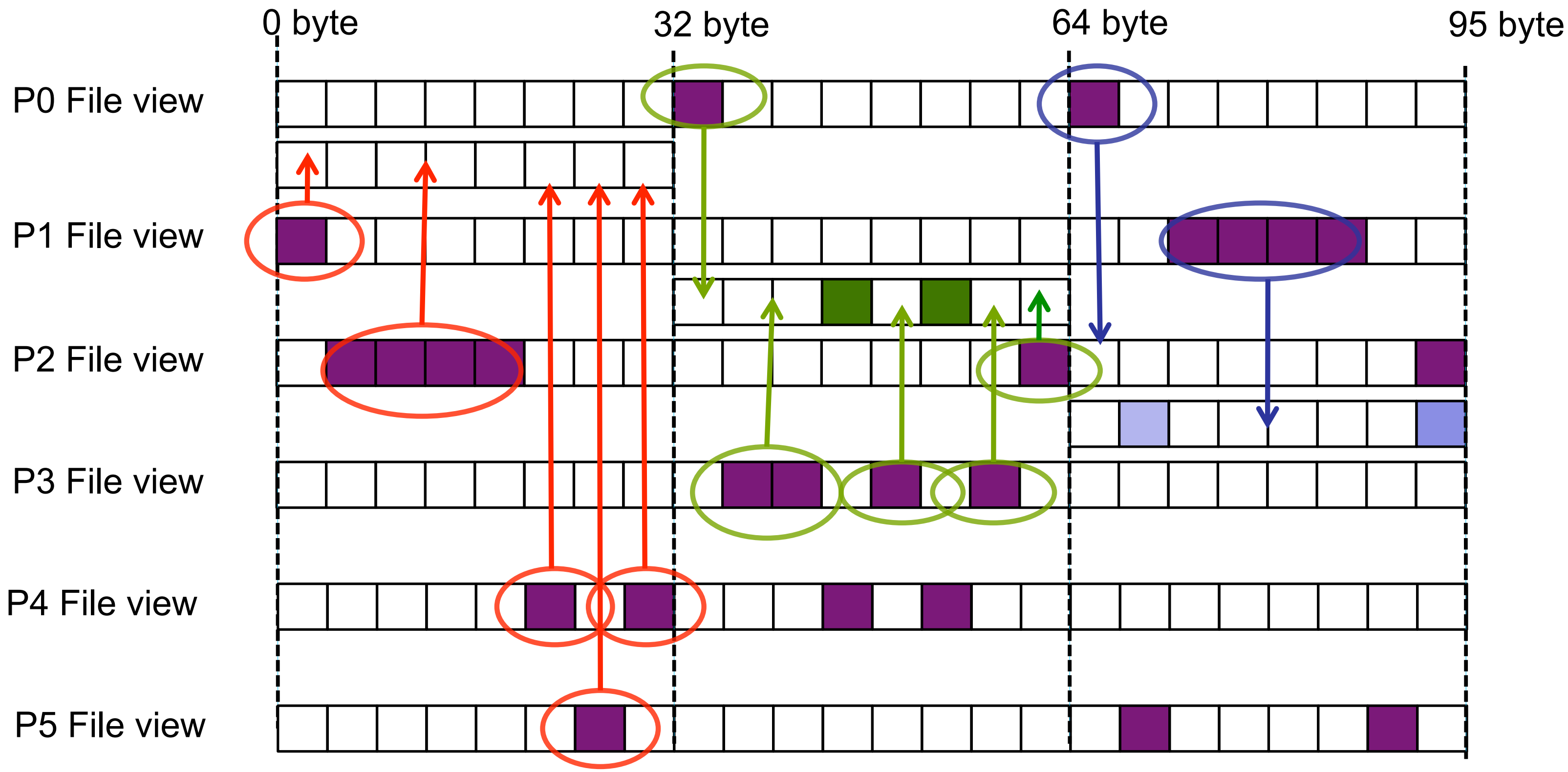
Example of Two-Phase I/O



Example of Two-Phase I/O

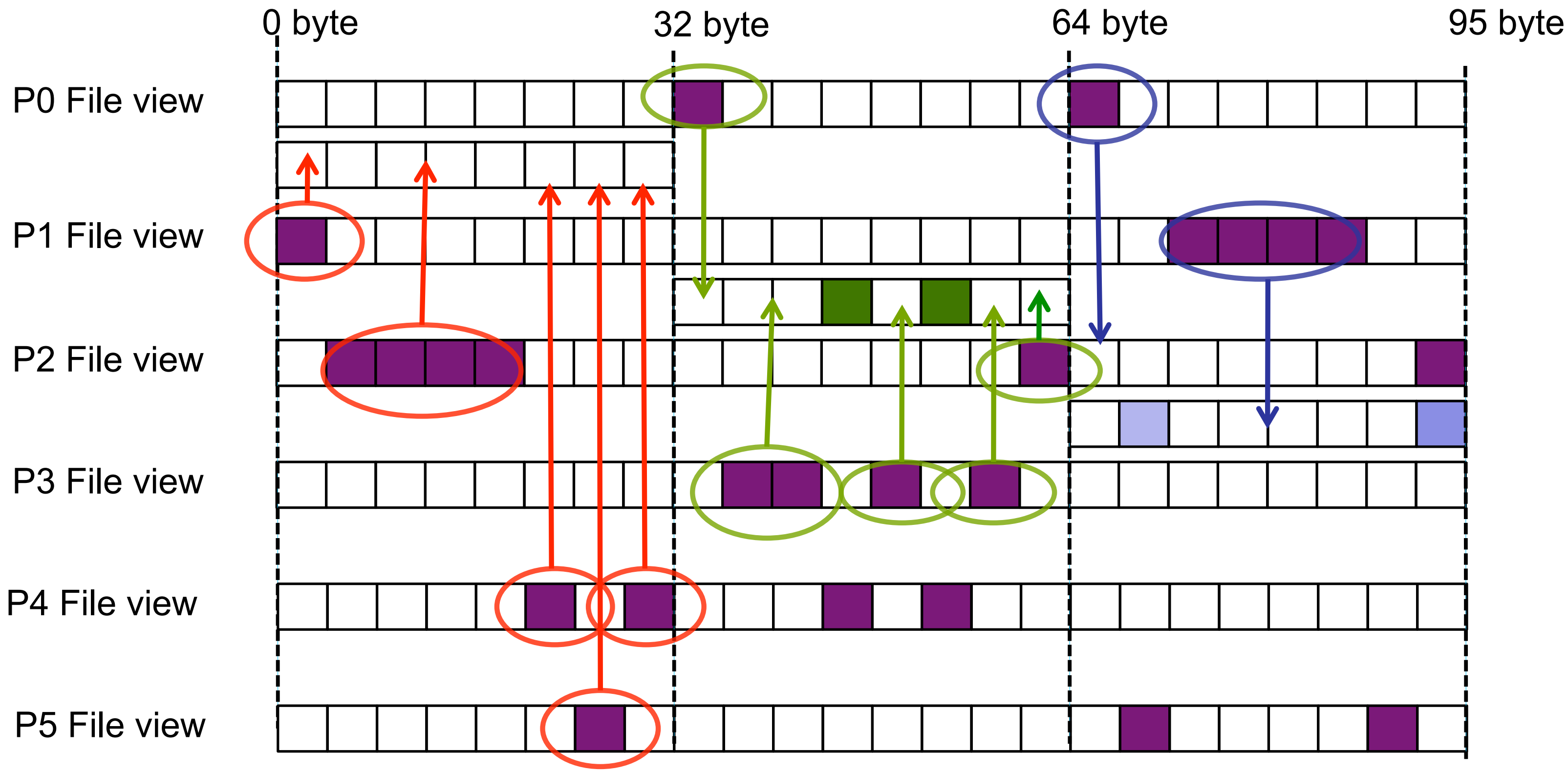


Example of Two-Phase I/O



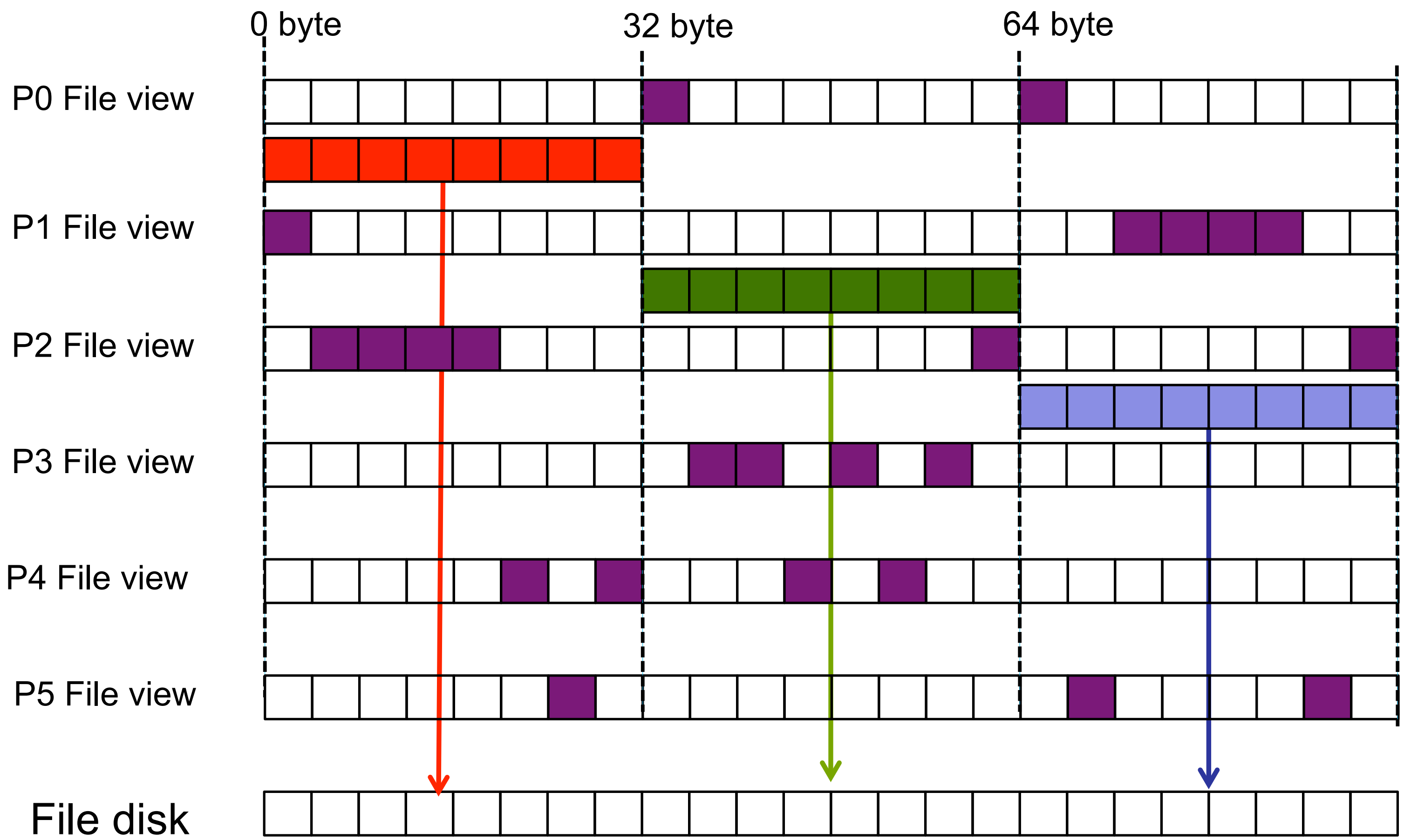
Communication: 12 message
72bytes of 96 bytes are transferred among the processes.

Example of Two-Phase I/O

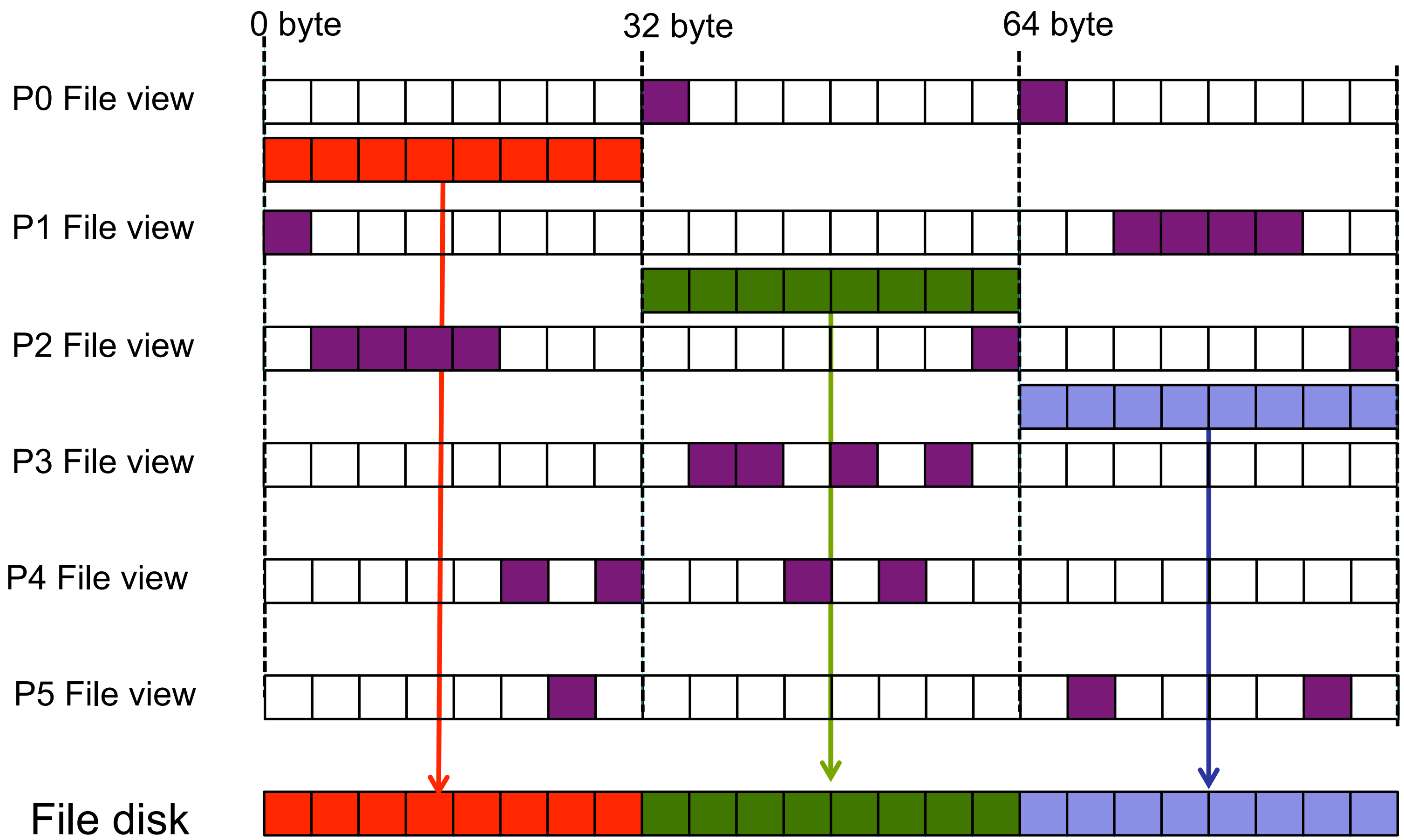


The number of communications could be reduced if each aggregator is assigned to the process **more adequate**.

Example of Two-Phase I/O



Example of Two-Phase I/O



Phd. Thesis Proposal

- Replace the rigid assignment of aggregator by new one based on the next aggregation-criteria:
  - Aggregation by communication number (ACN):
    - Each aggregator is assigned to the **process** who has the highest number of contiguous data blocks.
- MPICH1.2

HPC-Programme Proposal (I)

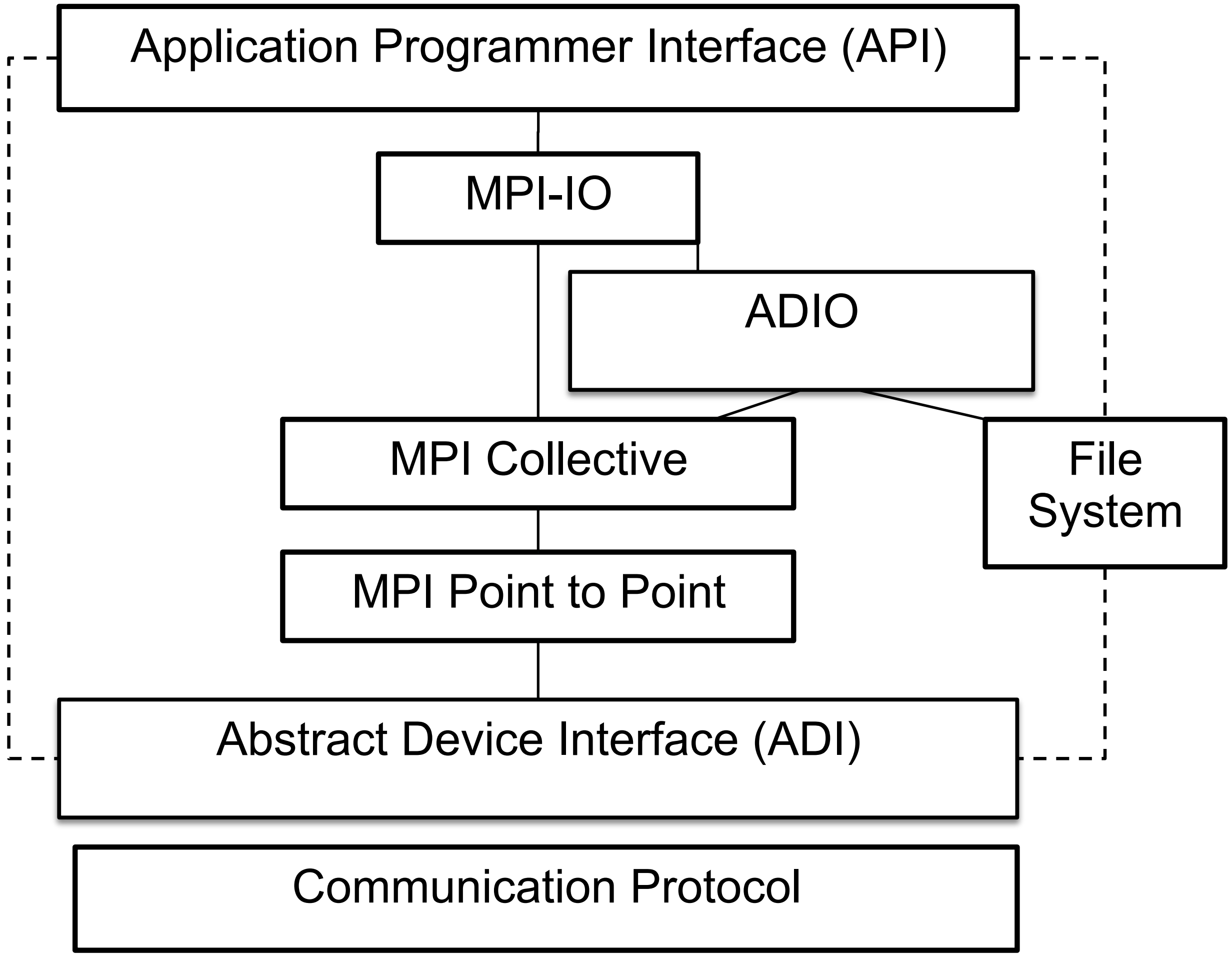
- Replace the rigid assignment of the aggregators by one of the next two aggregation-criteria:
  - Aggregation by communication number (ACN):
    - Each aggregator is assigned to the (New!!) **node** who has the highest number of contiguous data blocks.
  - (New!!) Aggregation by volume number (AVN):
    - Each aggregator is assigned to the node who has more volume of data.

- (New!!) MPICH2:
  - Communication among the cores of the same node by shared memory.

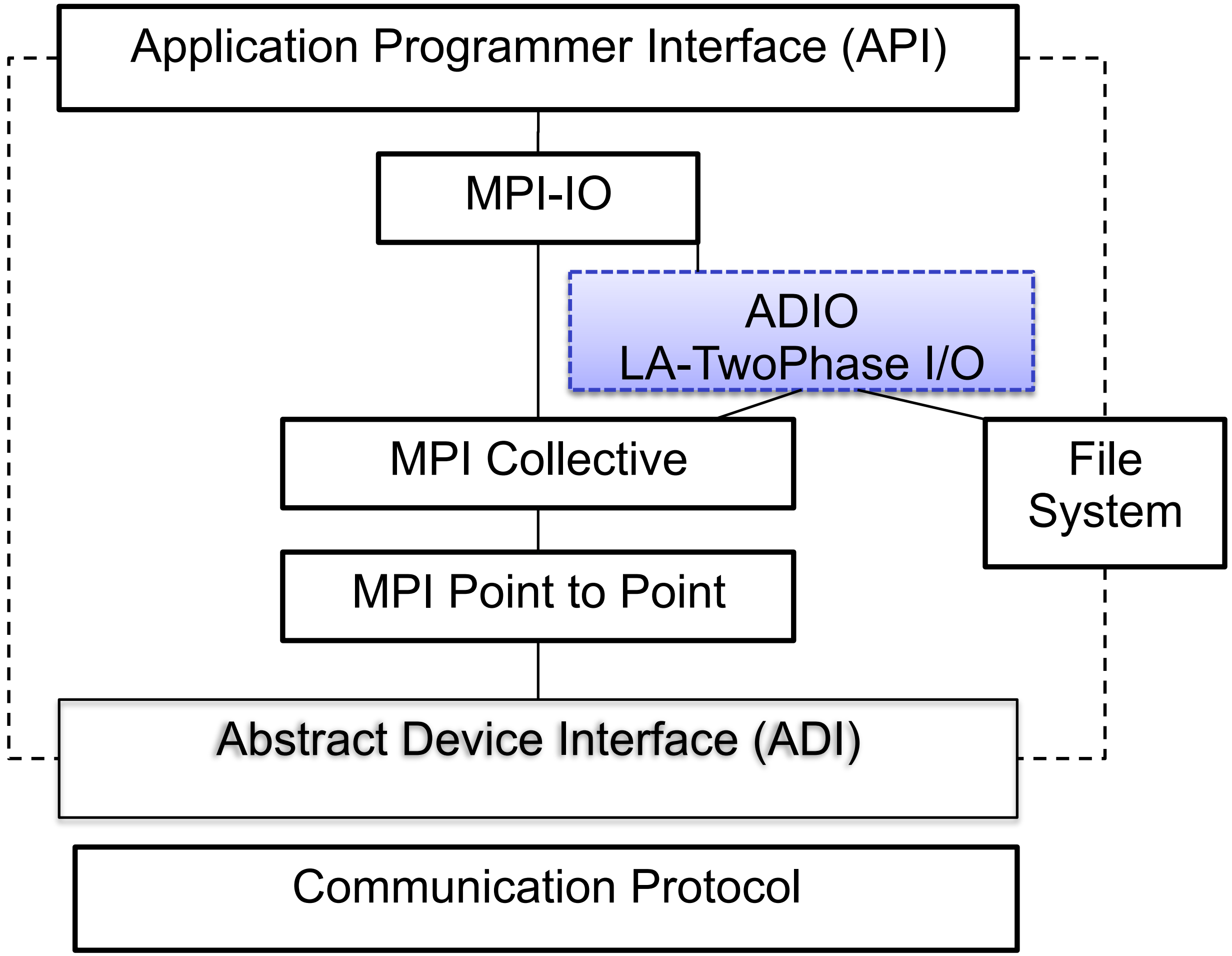
Implementation of the proposal

- The new two dynamic aggregator patters are implemented in two different ways:
  - New version of Two_Phase I/O in MPICH2:
    - Locality_Aware_Two_Phase I/O (LA_TwoPhase I/O)
  - Library at application level:
    - Aggregation_Pattern_Calculation
    - The MPI based application call this library to calculate the aggregators.
    - Use MPI-IO Hint (**variables can be used to control the behavior of collective operations**) to modified the default aggregator pattern:
      - cb_config_list: Provides explicit control over aggregators.

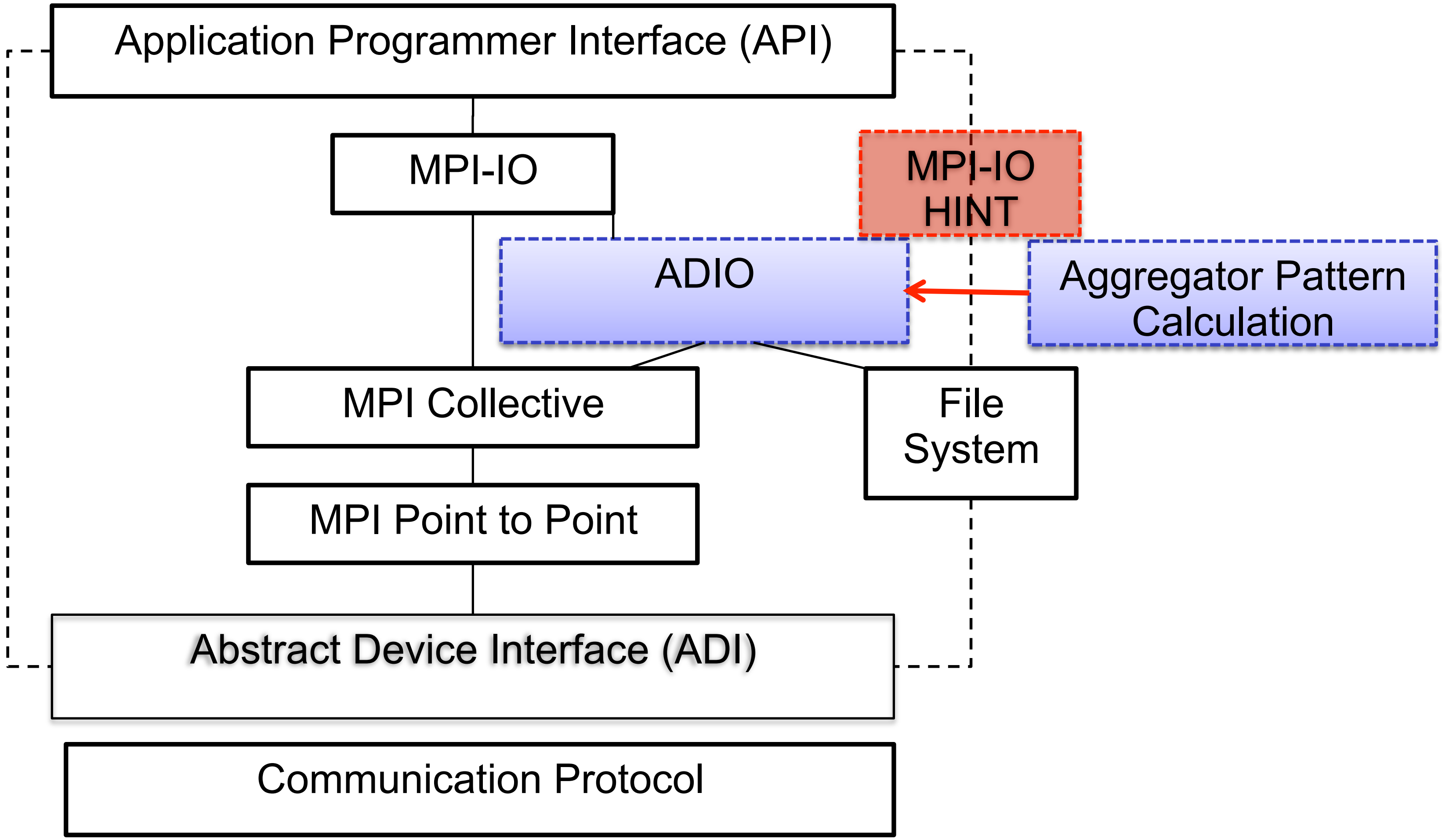
MPICH2 Architecture



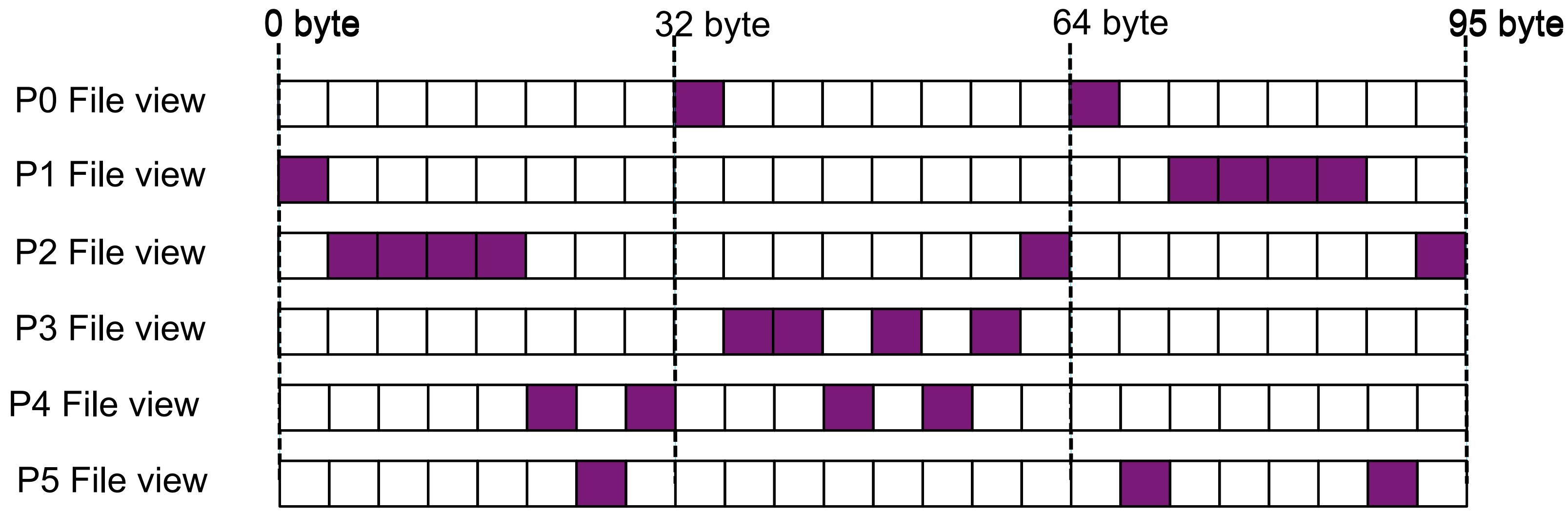
MPICH2 Modification (I)



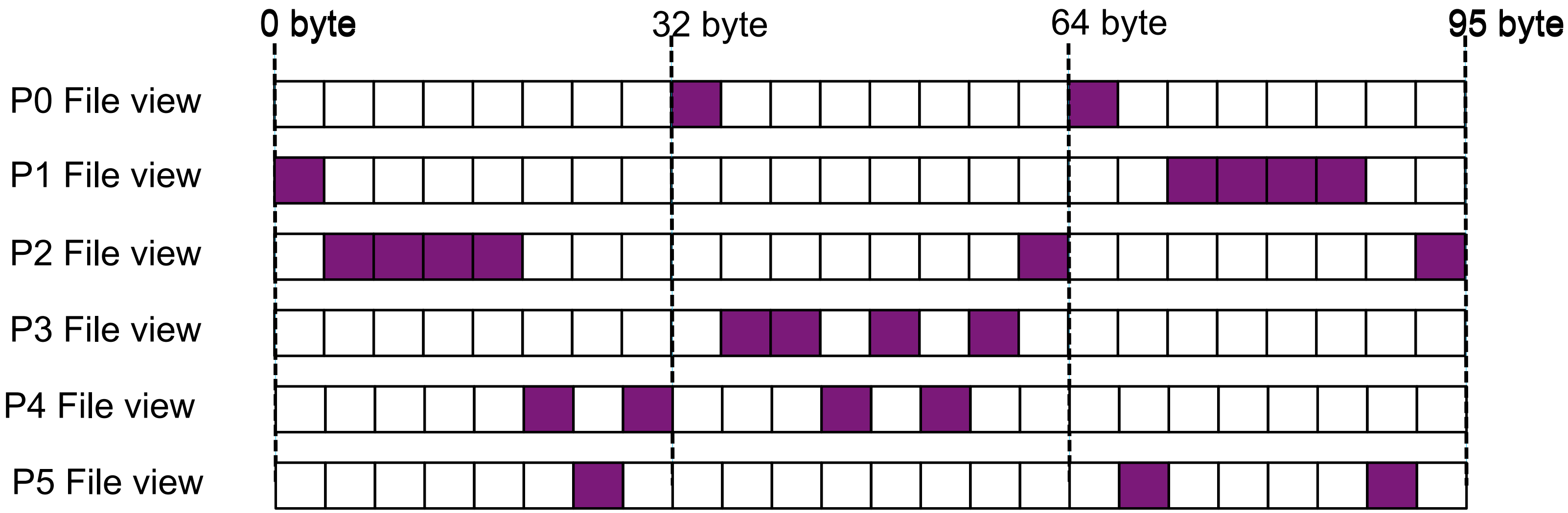
MPICH2 Modification (II)



Example of ACN aggregator pattern



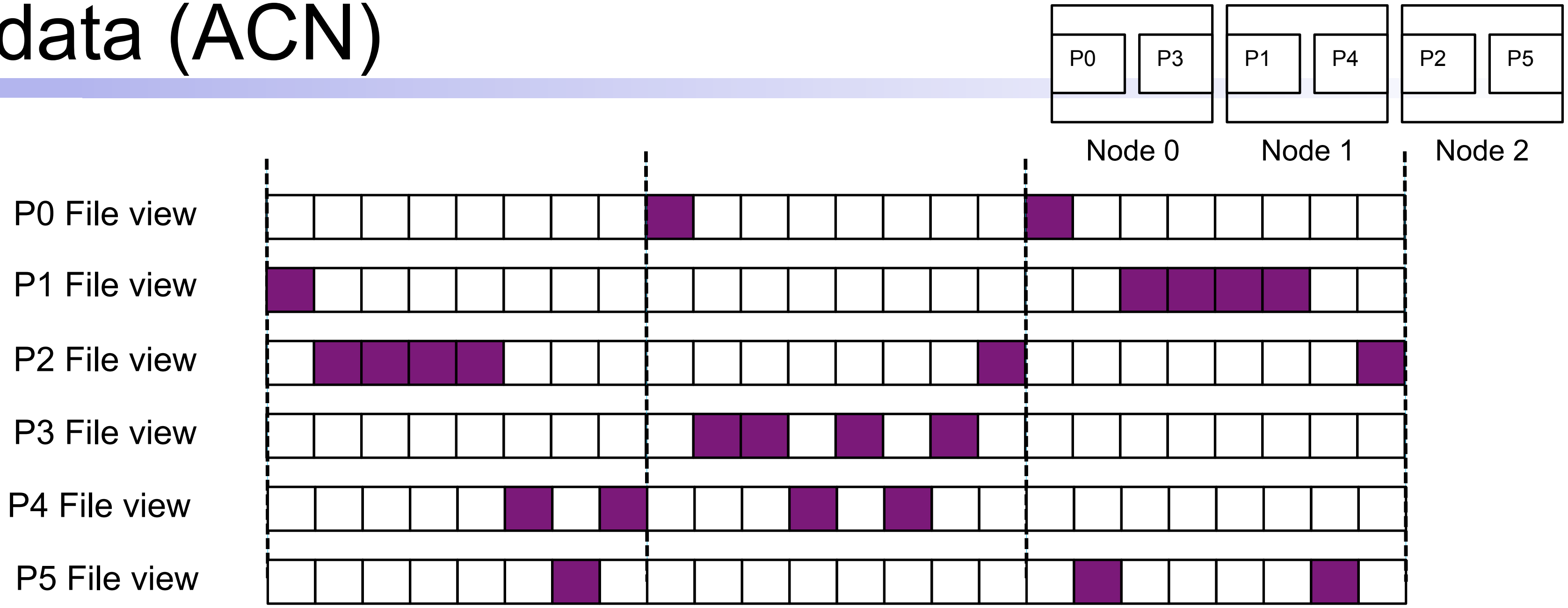
Calculation and recollection of distribution data (ACN)

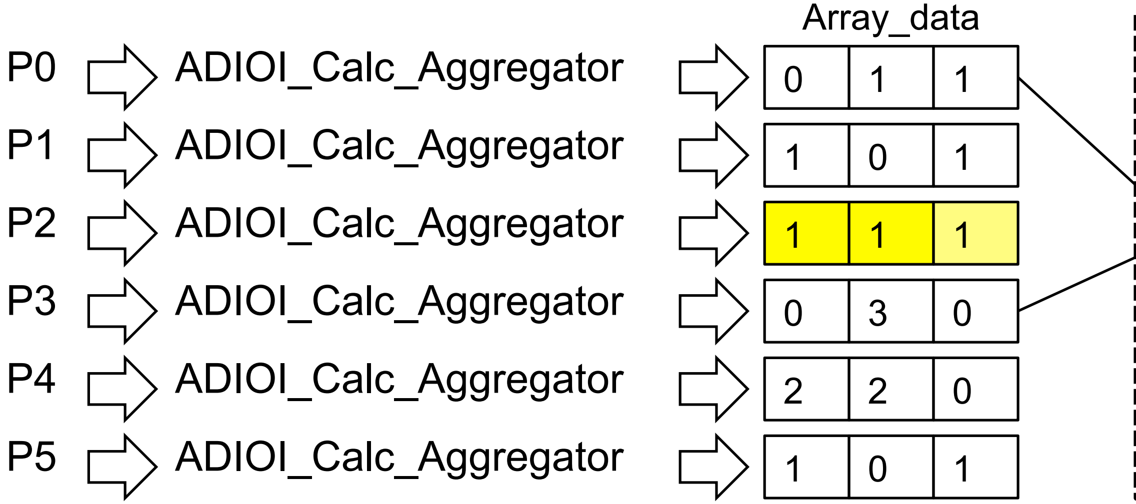


Array_data

P0	ADIOI_Calc_Aggregator	0	1	1
P1	ADIOI_Calc_Aggregator	1	0	1
P2	ADIOI_Calc_Aggregator	1	1	1
P3	ADIOI_Calc_Aggregator	0	3	0
P3	ADIOI_Calc_Aggregator	2	2	0
P3	ADIOI_Calc_Aggregator	1	0	1

Calculation and recollection of distribution data (ACN)





	FD0	FD1	FD2
N0	0	4	1
N1	3	2	1
N2	2	1	3

Calculation of ACN pattern

P0

Assigment_matrix_ACN

LAP ⇨

	FD0	FD1	FD2
N0	0	4	1
N1	3	2	1
N2	2	1	3

Aj=Max(Pi,FD)

Calculation of ACN pattern

P0

Assigment_matrix_ACN

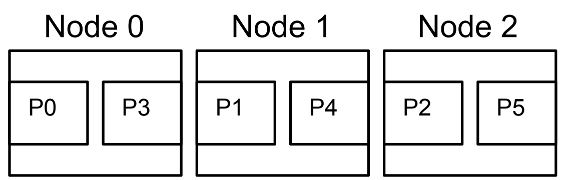
LAP ⇨

	FD0	**FD1**	**FD2**
N0	0	**4**	1
N1	**3**	2	1
N2	2	1	**3**

Aj=Max(Pi,FD)

New ranklist

P1	P0	P2



Calculation of ACN pattern

P0

Assigment_matrix_ACN

LAP ⇨

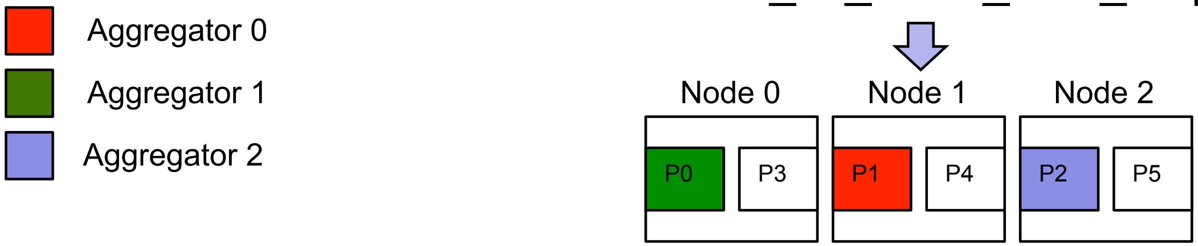
	FD0	FD1	FD2
N0	0	**4**	1
N1	**3**	2	1
N2	2	1	**3**

Aj=Max(Pi,FD)

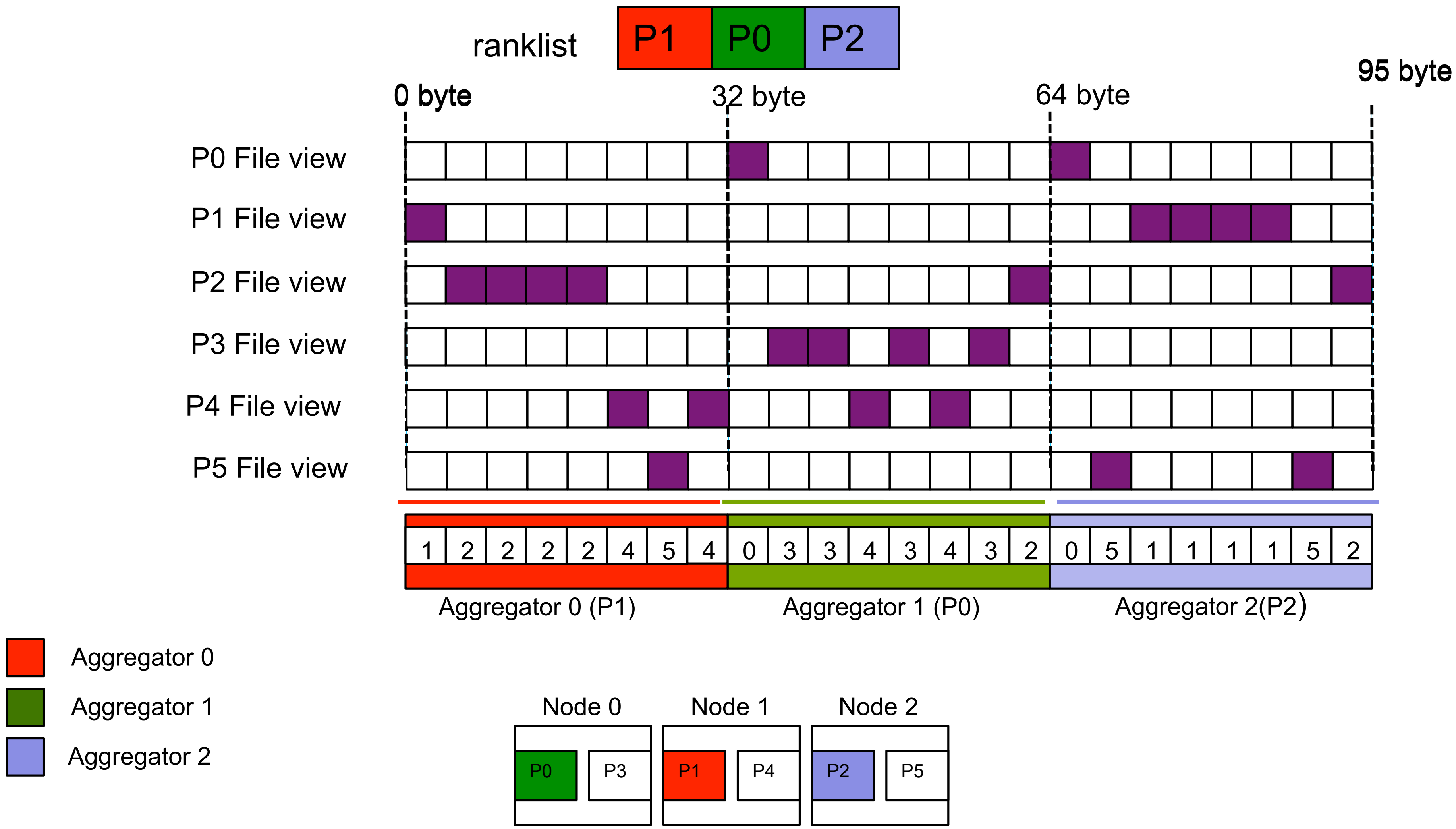
New ranklist

P1	P0	P2

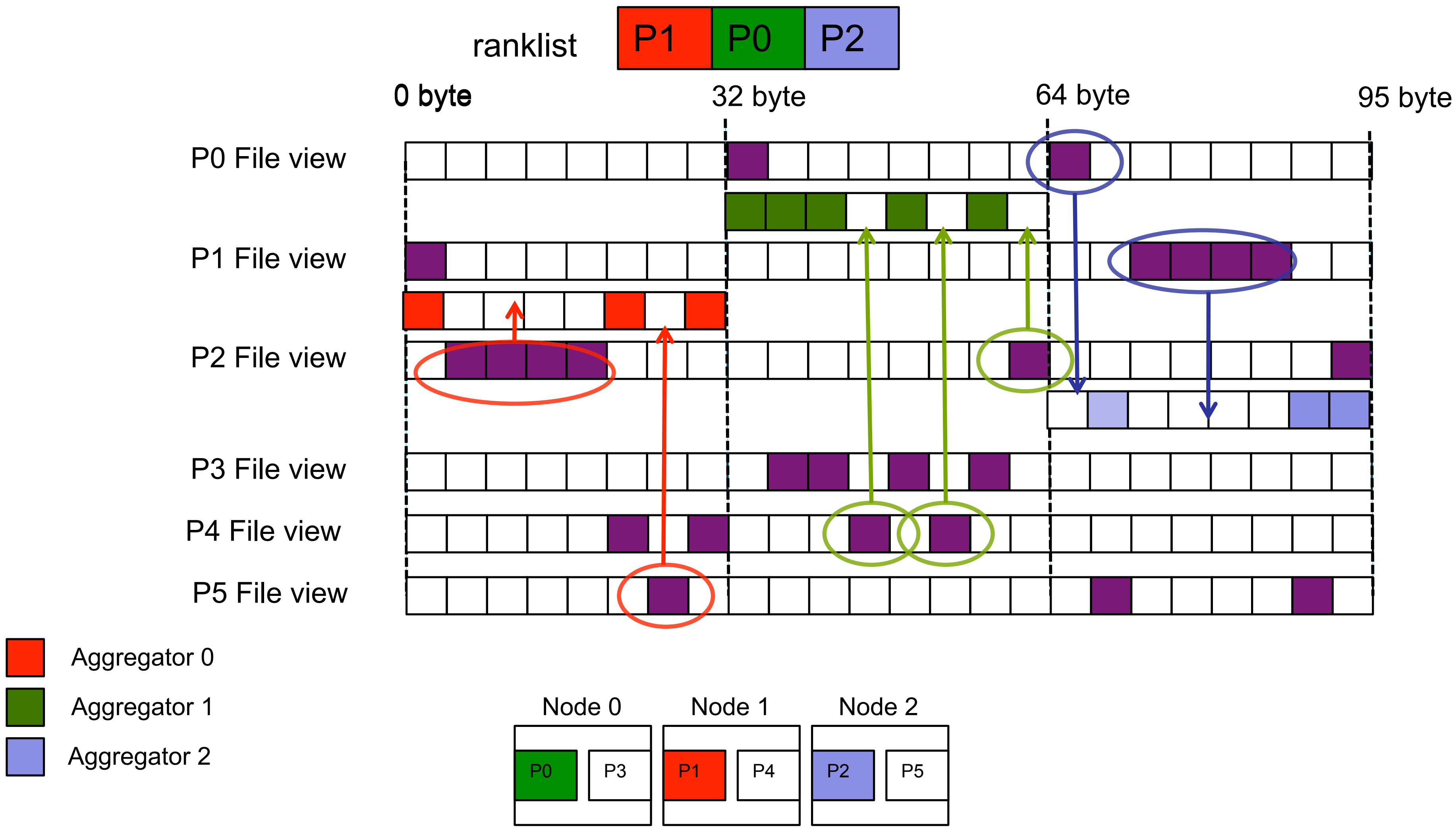
ADIOI_cb_bcast_rank_map



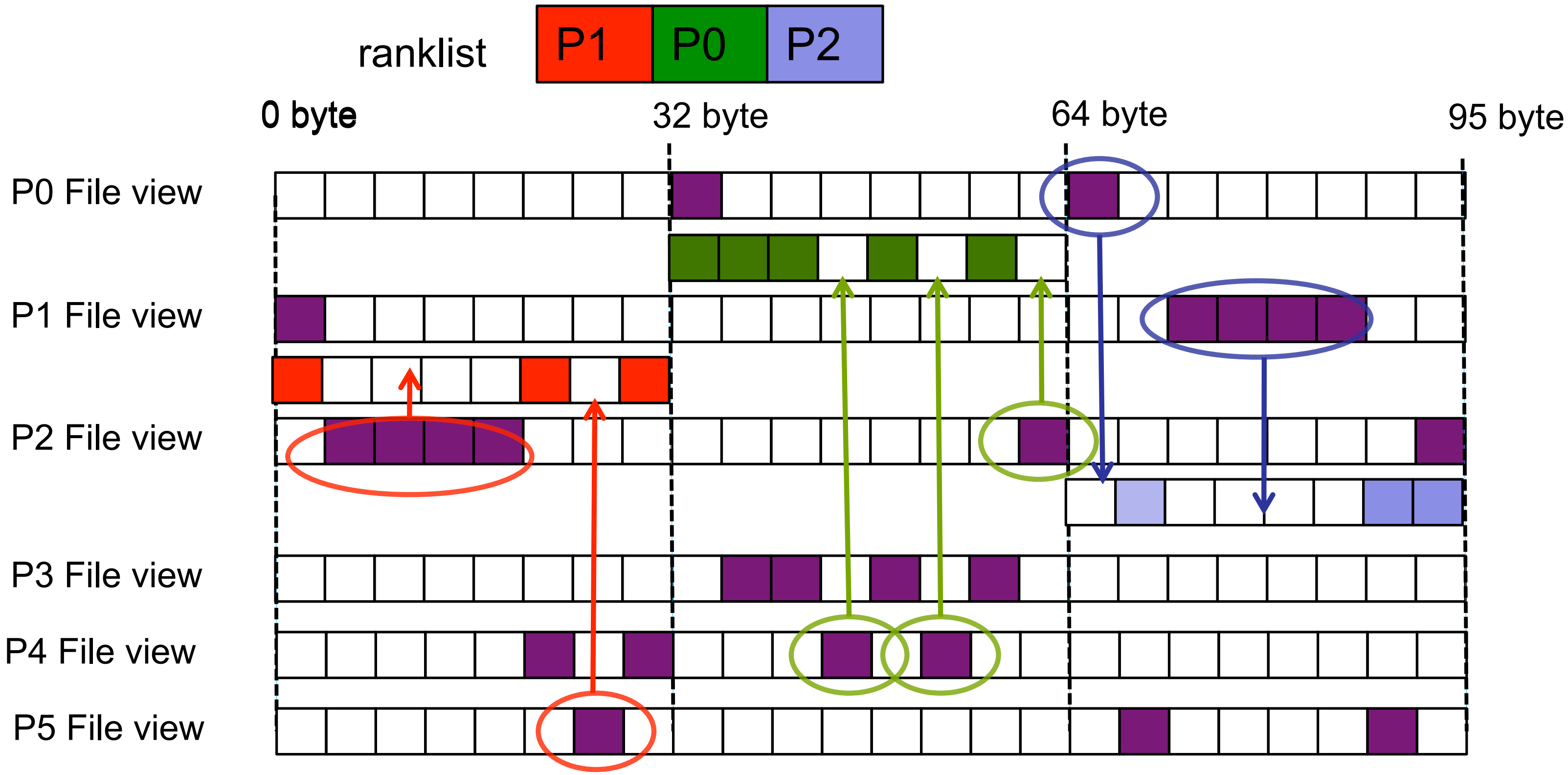
Reduction in the number of communication with the ACN pattern.



Reduction in the number of communication with the ACN pattern.



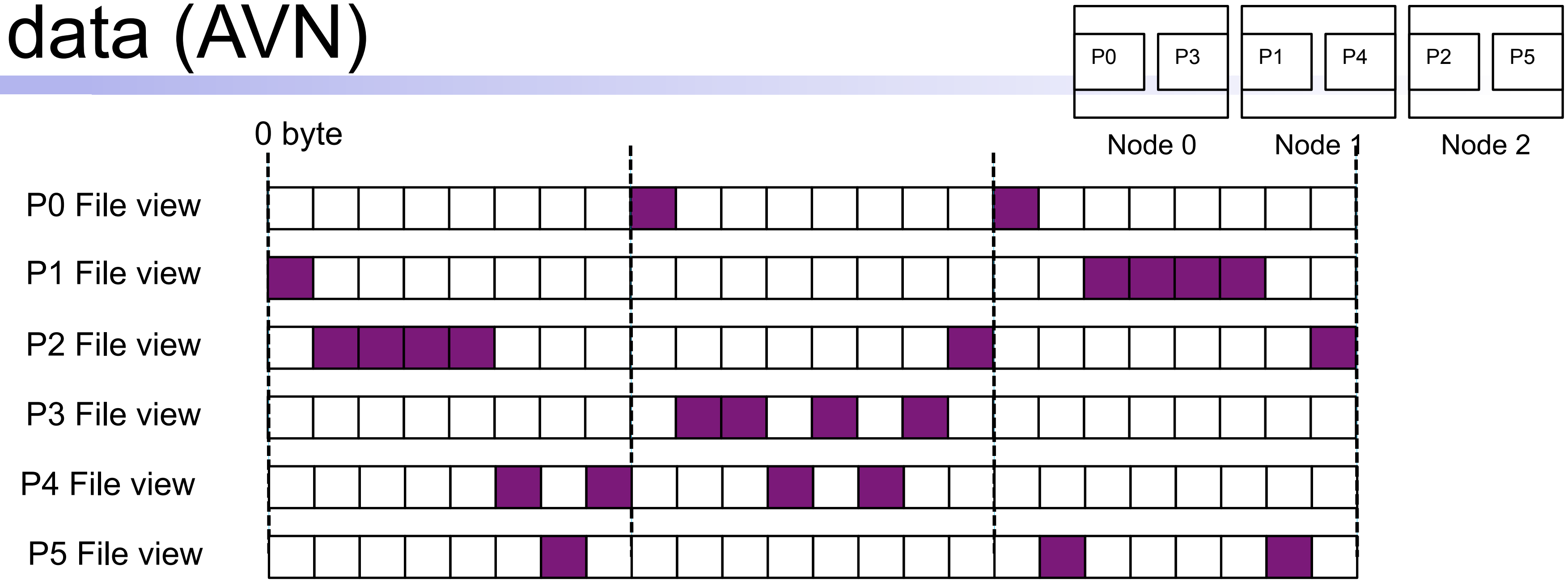
Reduction in the number of communication with the ACN pattern.

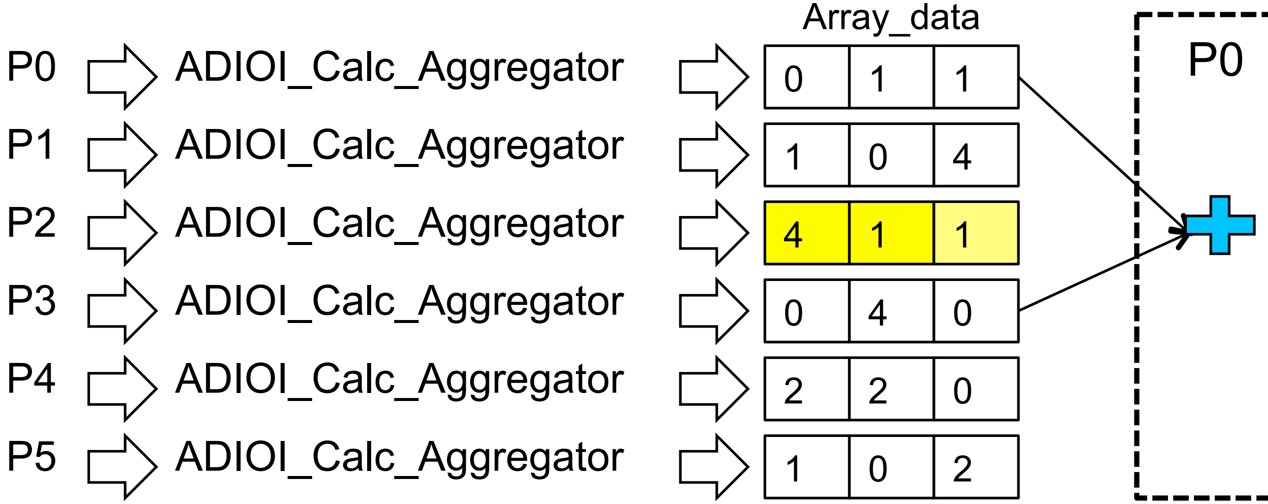


Two_Phase	ACN
12 messages	7 messages

ACN target:
Reduce the number of communications

Calculation and recollection of distribution data (AVN)





P0 Assigment_matrix_AVN

	FD0	FD1	FD2
N0	0	5	1
N1	3	2	4
N2	5	1	3

Calculation of AVN pattern

P0

Assigment_matrix_AVN

LAP ⇨

	FD0	FD1	FD2
N0	0	**5**	1
N1	3	2	**4**
N2	**5**	1	3

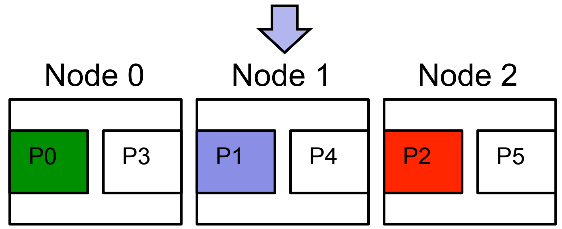
Aj=Max(Pi,FD)

New ranklist

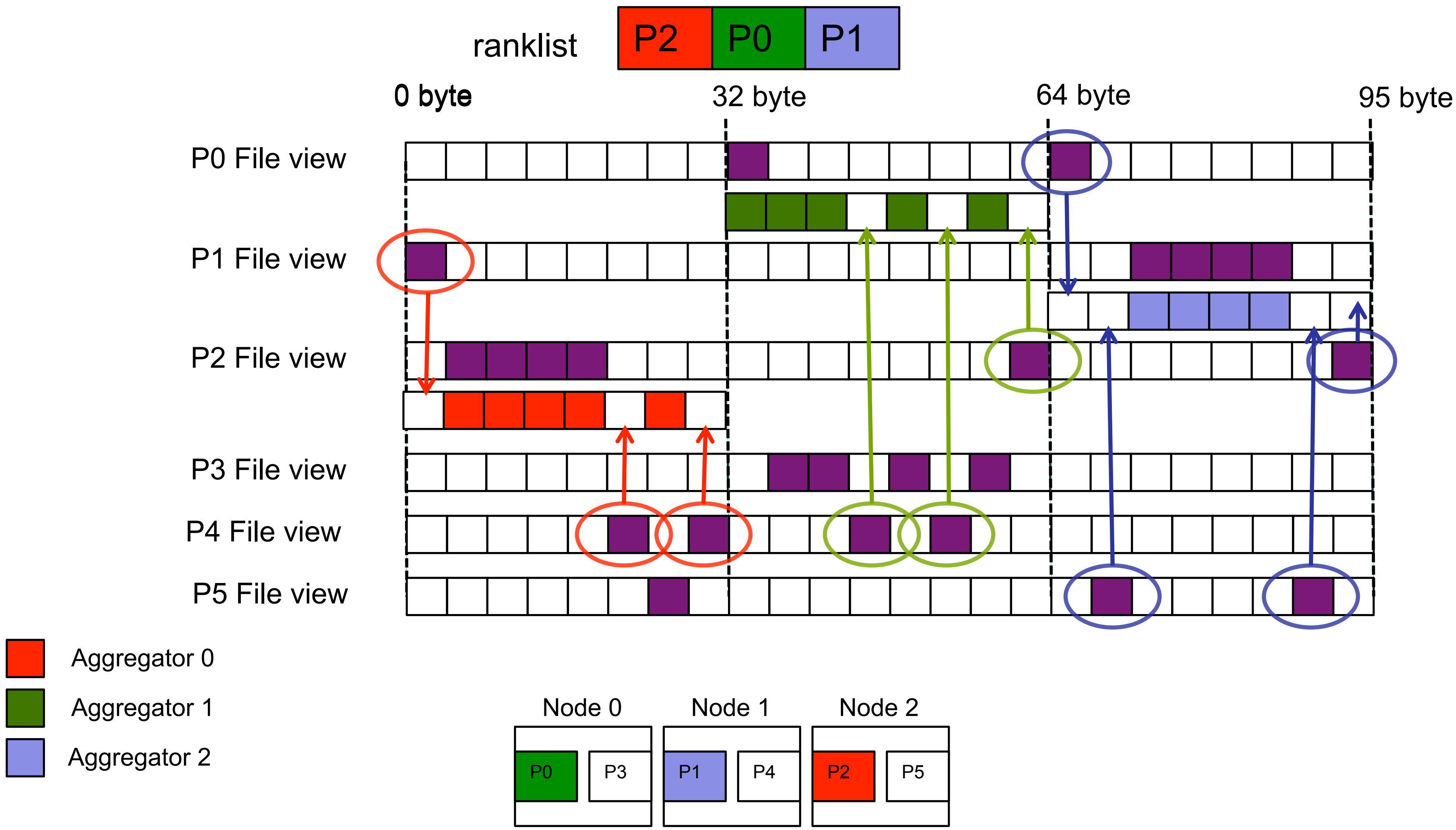
P2	P0	P1

ADIOI_cb_bcast_rank_map

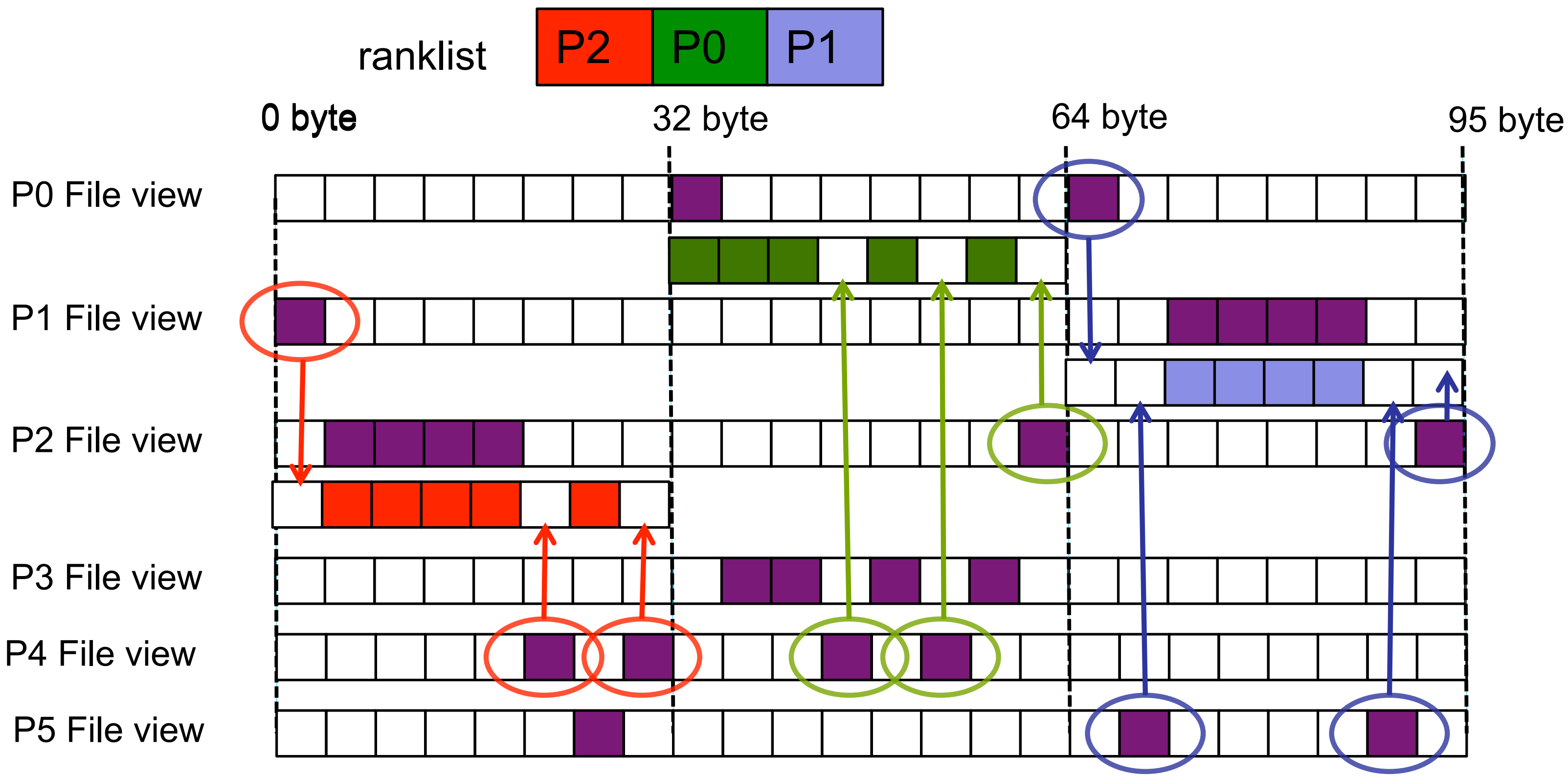
- Aggregator 0
- Aggregator 1
- Aggregator 2



Reduction in the number of communication with the AVN pattern.



Reduction in the number of communication with the AVN pattern.



Two_Phase	AVN
72 bytes	40 bytes

AVN target:
Reduce the volume of communications

ACN vs AVN

Two_Phase	ACN
12 messages	7 messages

Two_Phase	AVN
72 bytes	40 bytes

ACN target:
Reduce the number of communications

AVN target:
Reduce the volume of communications

- Which is the best ??
  - Depends of the data distribution among the processes.
    - Data very distributed, small contiguous data blocks :
      - High number of communications
      - Best aggregator pattern: **ACN**
    - Data less distributed, big contiguous data blocks:
      - High volume of data in the communications
      - Best aggregator pattern: **AVN**

New Proposal: Intelligent aggregator pattern for collective I/O pattern

- Select the aggregation-criterion that the reduce more the communication phase.
- Open question

Why two implementation of ACN and AVN?

- Aggregator patters are implemented in two different ways:
  - New version of Two_Phase I/O in MPICH2:
    - Locality_Aware_Two_Phase I/O (LA_TwoPhase I/O)
  - Library at application level:
    - Aggregation_Pattern_Calculation
    - The MPI based application call this library to calculate the aggregator list.
    - The application use one MPI-IO Hint to modified the default aggregator list:
      - cb_config_list: Provides explicit control over aggregators

Why two implementation of ACN and AVN? (II)

- First idea:
  - Install my own library of MPICH2 in HECToR.
  - Modify the source code of Two_Phase IO (inside MPICH2) to implement both aggregator patterns (Locality_Aware_Two_Phase I/O)
  - Problem: HECToR→ No install own MPI library.
- Second idea:
  - MPI-IO hints are used to provide information to a MPI program to assist the performance of the MPI file I/O routines.
  - Use **cb_config_list** hint to indicate to XT MPI my "new aggregator list".
  - Implement the aggregator patterns at application level. (Aggregation_Pattern_Calculation)

Lists MPI-IO hints supported on Cray XT systems.

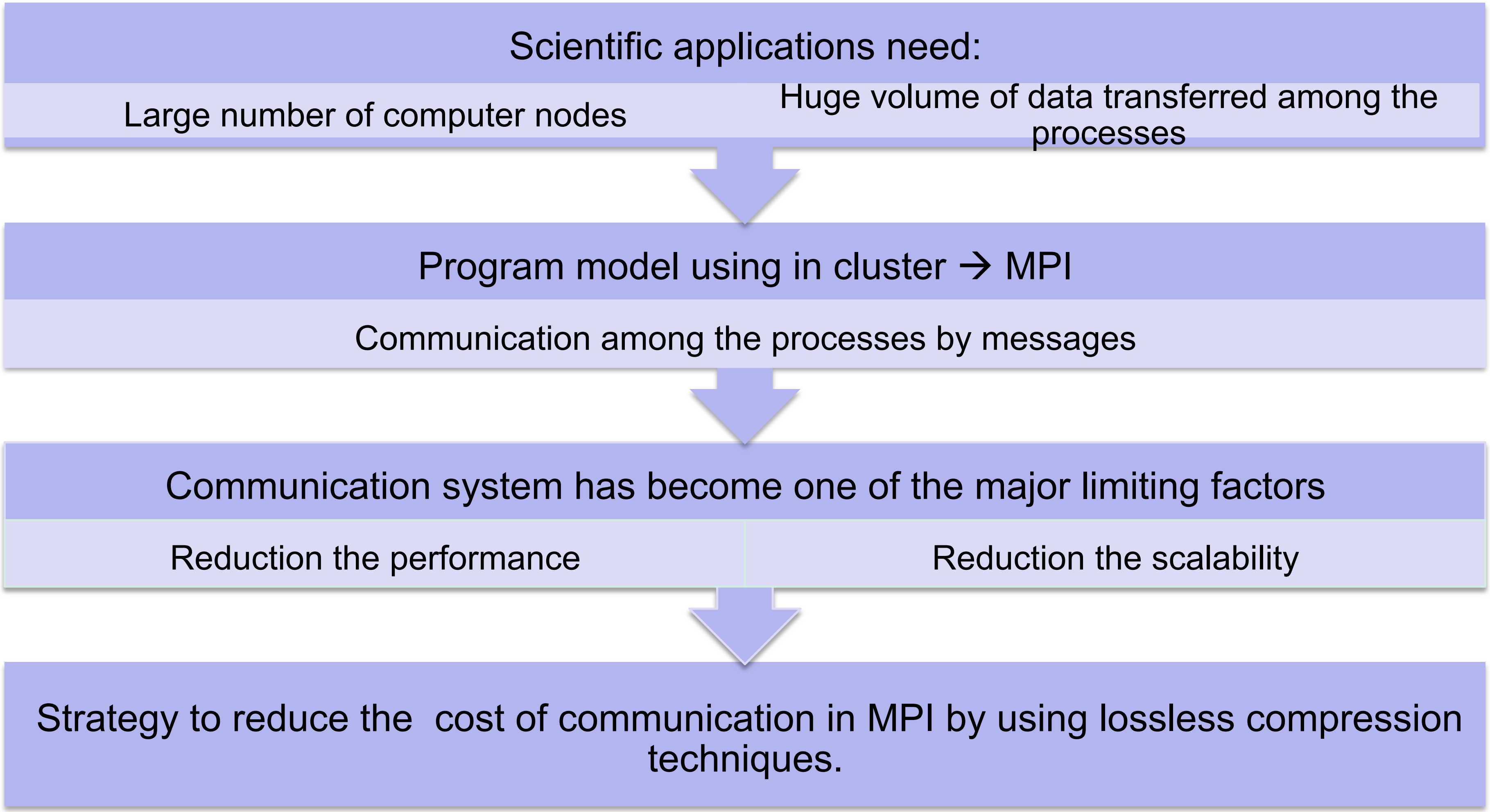
direct_io	cb_config_list	ind_rd_buffer_size
romio_cb_read	romio_no_indep_rw	ind_wr_buffer_size
romio_cb_write	romio_ds_read	cb_nodes
cb_buffer_size	romio_ds_write	

- Problem: cb_confing_list “seems” not work in XT MPI
  - Cray developer responsible for MPI-IO recognised my problem:
    - "We have raised a bug on the issue however right now and there doesn't seem to be an obvious workaround".
- Solution: Implement both ideas by using Eddie cluster.

Summary

1. Problem description
2. Main Objectives
3. Strategies for improve the performance of collective I/O operation
4. Strategies for improve the performance communication operations
5. Evaluations

Strategies for improve the performance of communication operations.



Phd. Thesis Proposal: Communication Compression

- Reduce the cost of communications:
  - By MPI messages compression in run-time
- Lossless compressions algorithms
- Compress all MPI primitives.
- We have developed three different strategies:
  - Runtime Compression (RC)
  - Runtime Adaptive Compression (RAC)
  - Guided Strategy (GS)
- Implementation of the strategies by modifying the source code of MPICH1.2

HPC-Programme Proposal (II)

- Implement Runtime Adaptive Compression (RAC) by using (New!!) message passing interface profiling (PMPI).
- Why ??
  - HECToR → No install own MPICH
  - DEISA/PRACE Spring School: Tools and Techniques for Extreme Scalability
    - The Scalasca Performance Analysis Toolset → PROFILING!!
    - New idea: Use the MPI standard profiling interface (PMPI) to implement the adaptive compression strategy

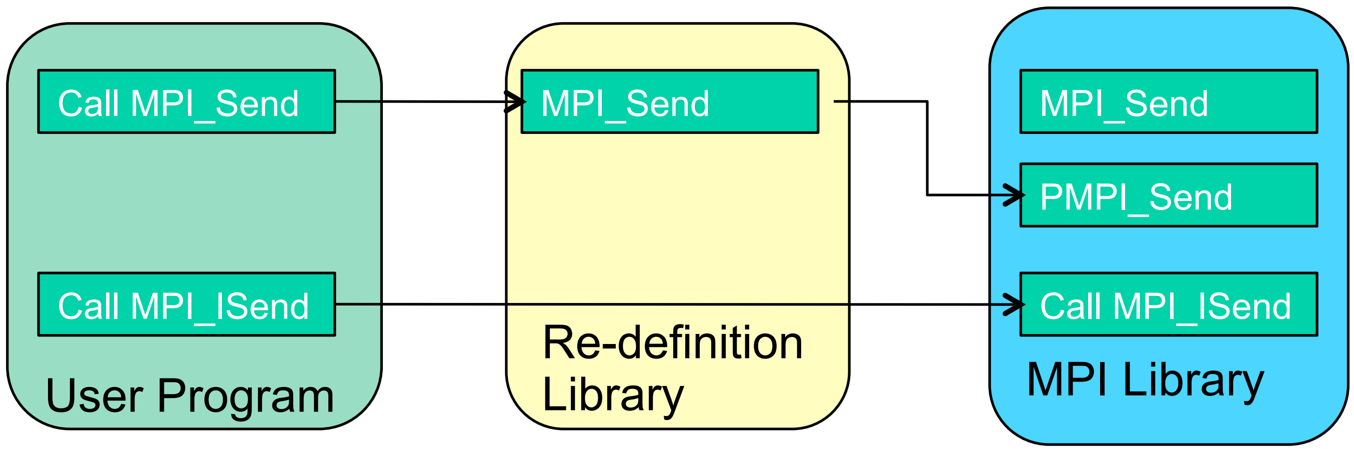
HPC-Programme Proposal (II)

- PMPI allows replacement of MPI routines at link time (not need to recompile)
  - No modification of the source code of the MPI implementation
  - No modification of the source code of the application

- Portable, independent of the MPI implementation (XT MPI, MPICH2, OPENMPI …).

PMPI

- Each standard MPI function can be called with an MPI_ or PMPI_ prefix.
- PMPI such wrapper functions to customize MPI behavior: implement RAC strategy



Example of Use of Profiling Interface


```
// extern.c
int MPI_Send( void *start, int count, MPI_Datatype datatype,
int dest, int tag, MPI_Comm comm )
{
    printf ("Before send the message to process %d\n",dest);
    return PMPI_send(start, count, datatype, dest, tag, comm);
}
```

```
// my_application.c
if (my_rank==0)
{   for (i=0;i<5;i++)
        array[i]=i;
    for (j=1;j<num_processes;j++)
        MPI_Send( array,5,MPI_INT,i,tag,MPI_COMM_WORLD);
 }
```

```
> mpicc -c extern.c
> mpicc -c my_application.c

> mpicc -g my_application.o extern.o -o executable
```

Example of Use of Profiling Interface


```
> mpirun –np 10 ./executable > output.txt
> cat output.txt

Before send the message to process 1
Before send the message to process 2
Before send the message to process 3
Before send the message to process 4
Before send the message to process 5
Before send the message to process 6
Before send the message to process 7
Before send the message to process 8
Before send the message to process 9
```

Runtime Adaptive Compression

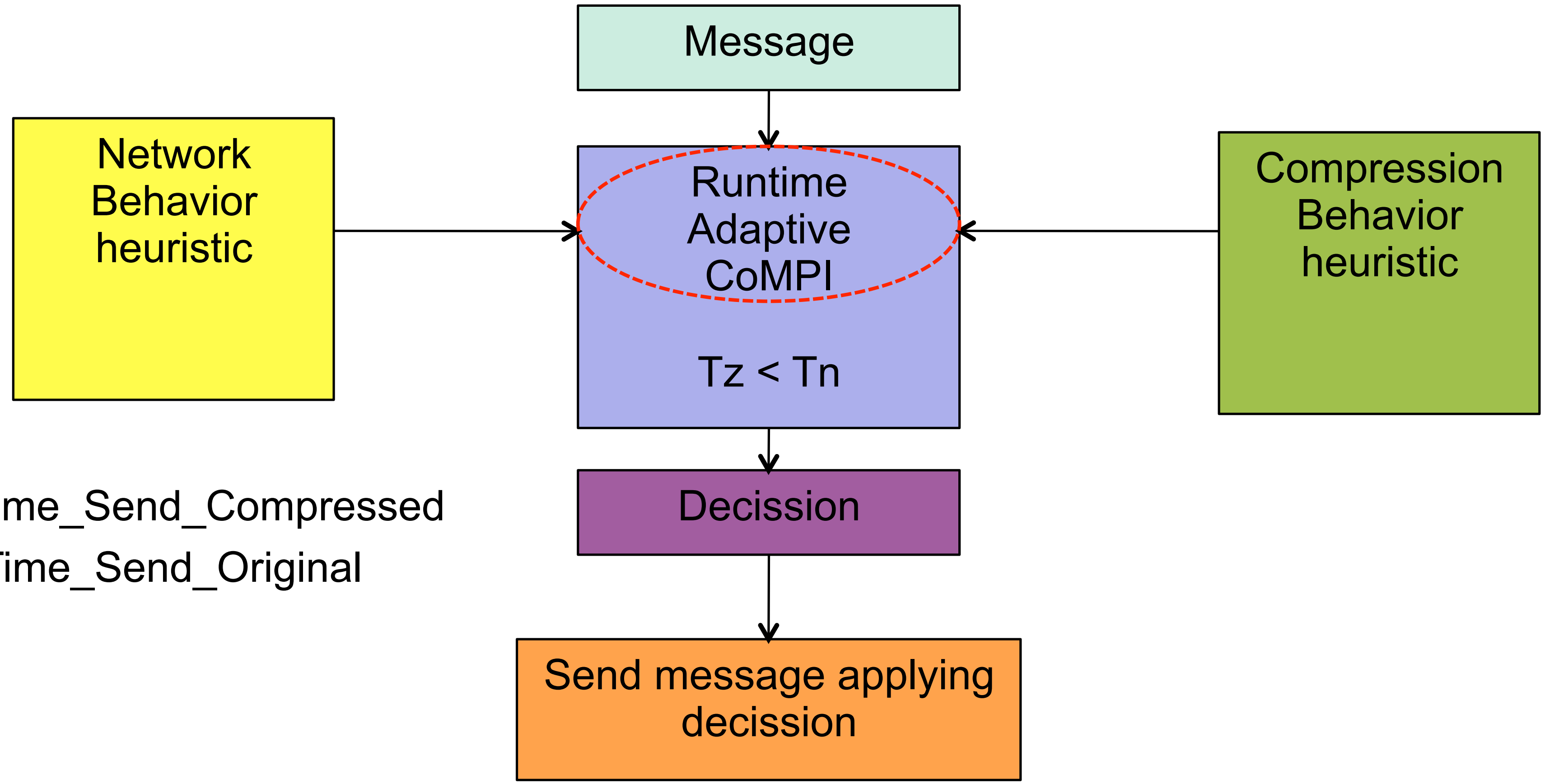
- Runtime Adaptive Compression Strategy (RAC), per message transferred takes two decision:
  - Turn on and off the compression.
  - Select itself the best compression algorithm:
    - LZO, RLE, HUFFMAN, RICE, FPC.
- Learn in run-time from previous messages
- Decision depending on:
  - Message feature:
    - Datatype and length
  - Network performance:
    - Latency and bandwidth
  - Compression algorithms

Decisions → Speedup

- Speedup to decide if send the message with/without compression.
- So the decission depends:
  - Original message transmisision time
  - Compressed messaged transmission time
  - Compression and decompression time

$$Speedup = \frac{Time\_Sent\_Orig.}{(Time\_Sent\_Compr. + time\_compress. + time\_decompr.)}$$

Decisions → Speedup

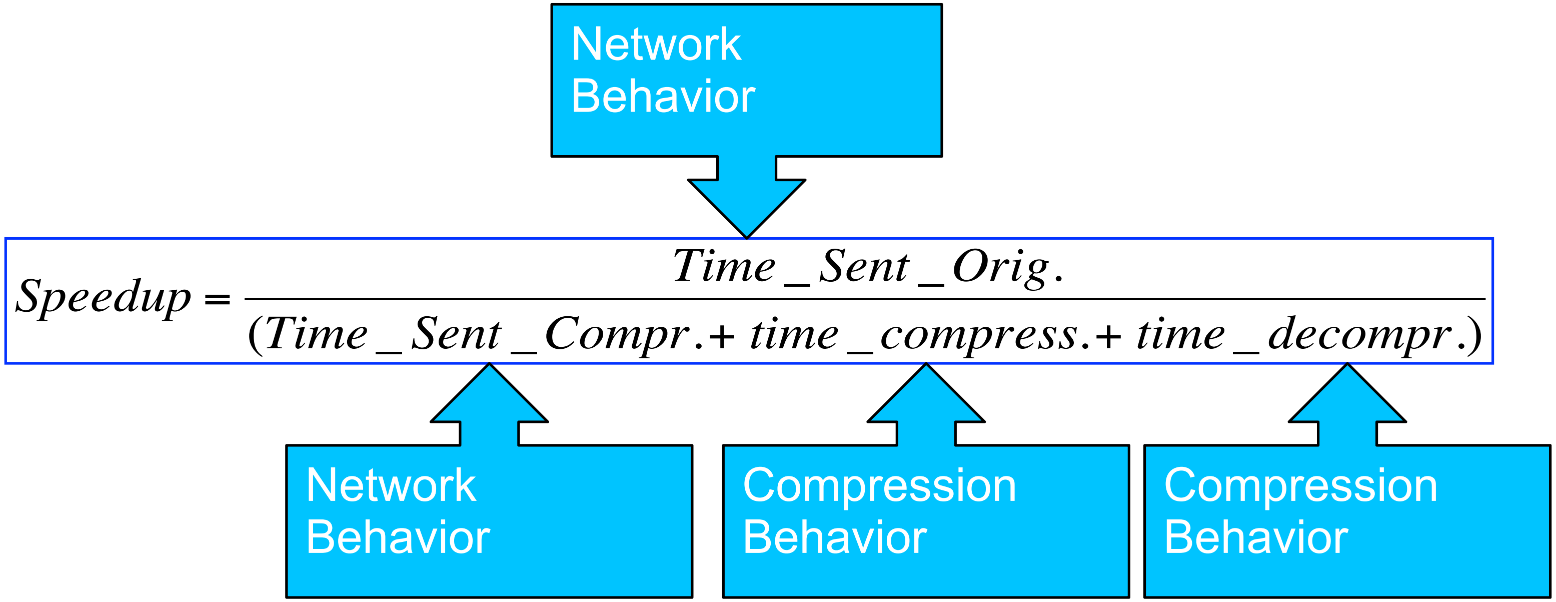


Tz=Time_Send_Compressed

Tn=Time_Send_Original

Compression behavior

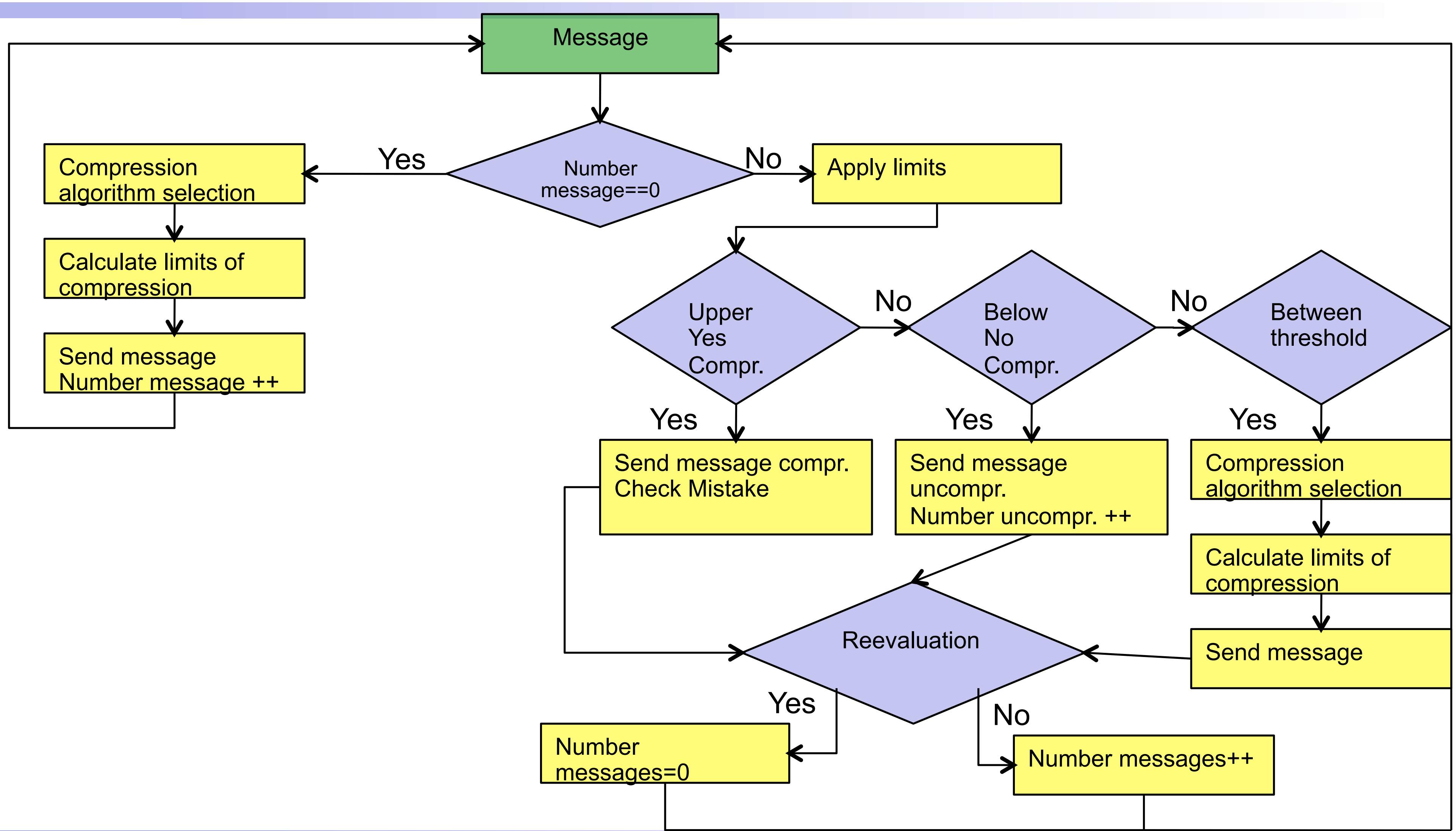
$$Speedup = \frac{Time\_Sent\_Orig.}{(Time\_Sent\_Compr. + time\_compress. + time\_decompr.)}$$



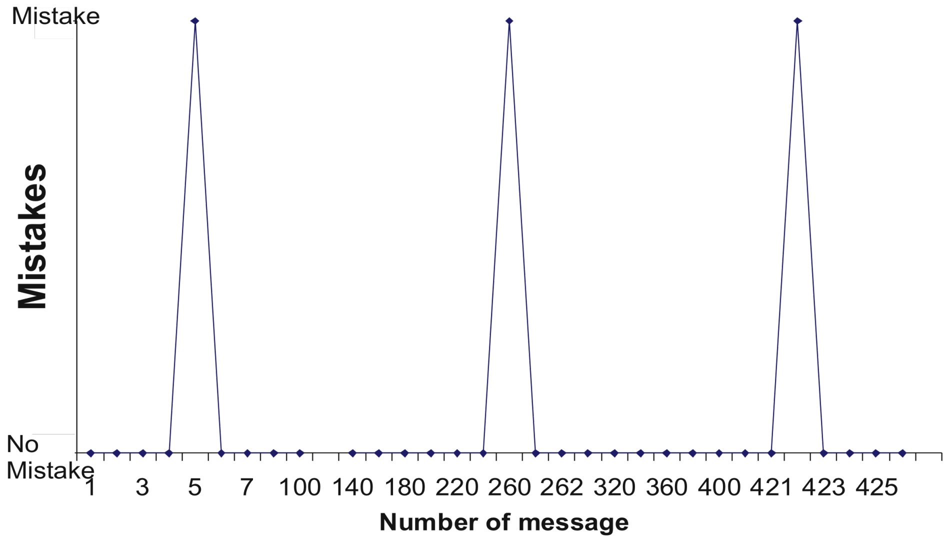
Decision Methodology

- Calculate the speedup per message? No → high overhead computation time
- According to Compression Behavior and Network data Behavior, RAS decides:
  - Datatype:
    - Integer y Float → LZO
    - Double → LZO or FPC
    - Others → LZO, RLE, RICE or HUFFMAN
  - Message size → Decision Threshold:
    - Each datatype has its thresholds
      - Length_yes_compression
      - Length_no_compression

Decision Methodology

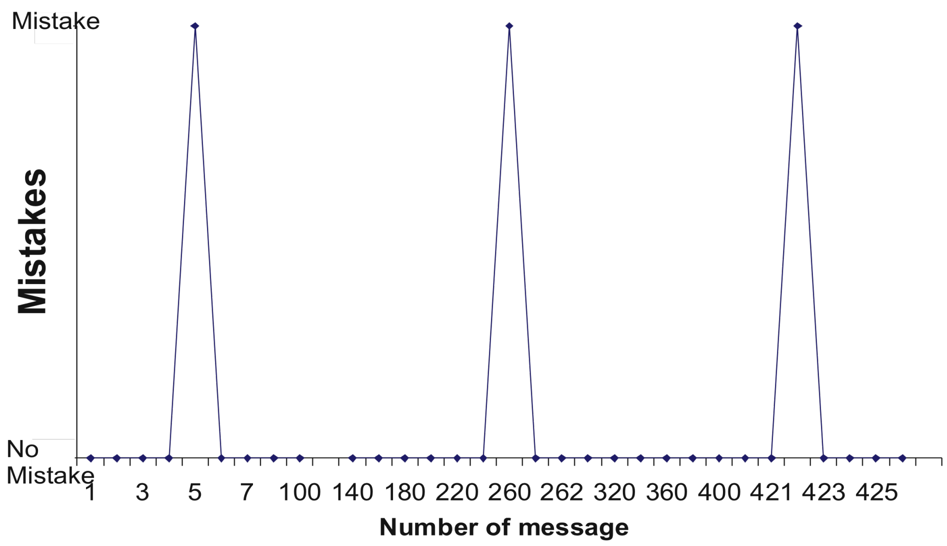


Different cases of re-evaluation

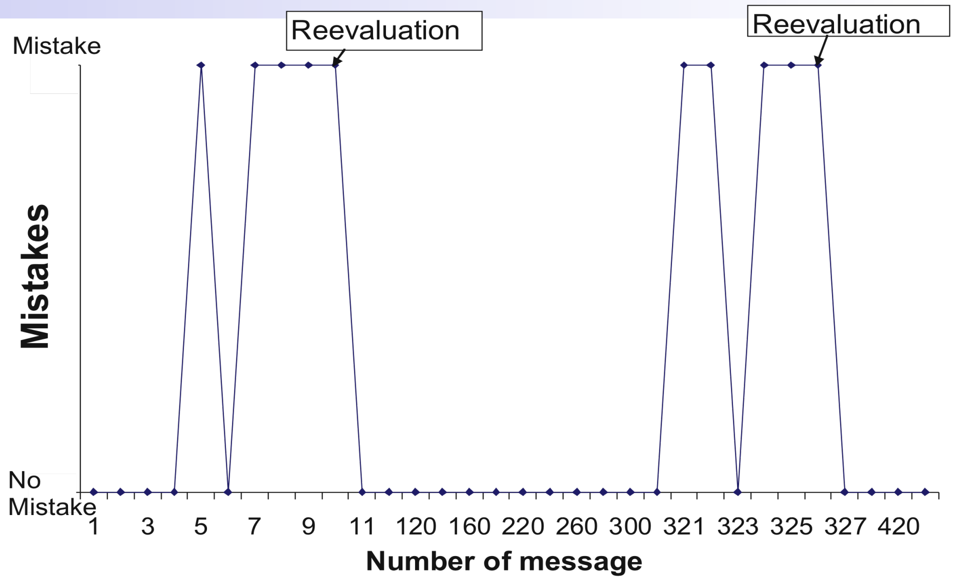


Low frequency mistakes

Different cases of re-evaluation

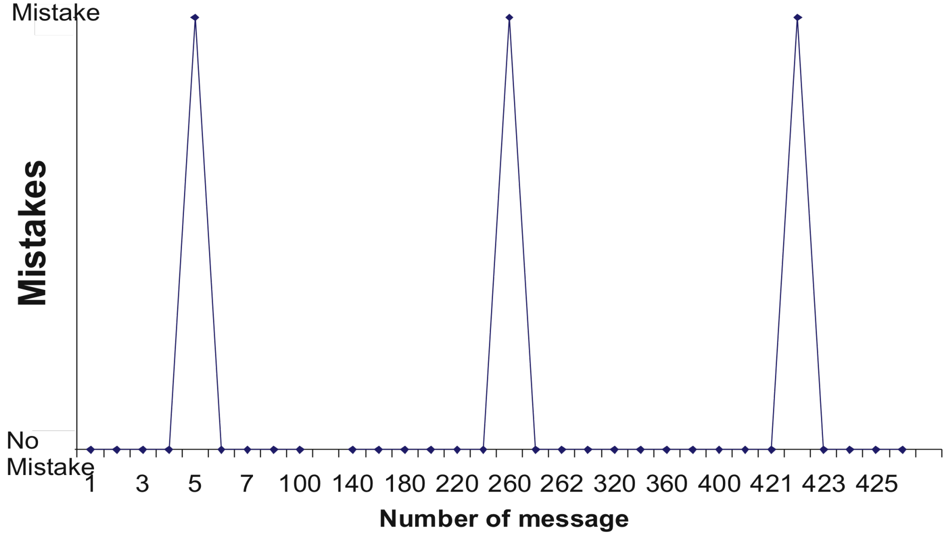


Low frequency mistakes

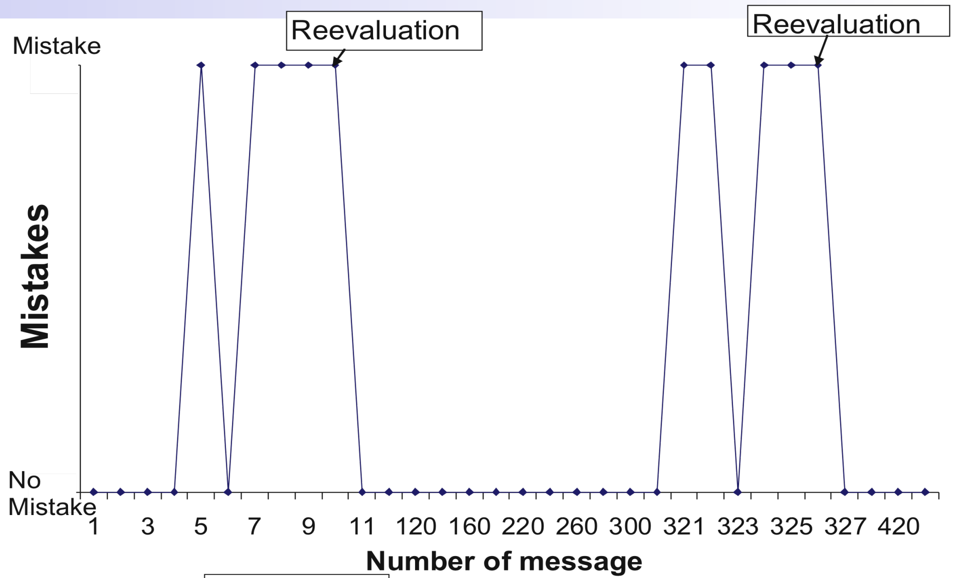


High frequency mistakes

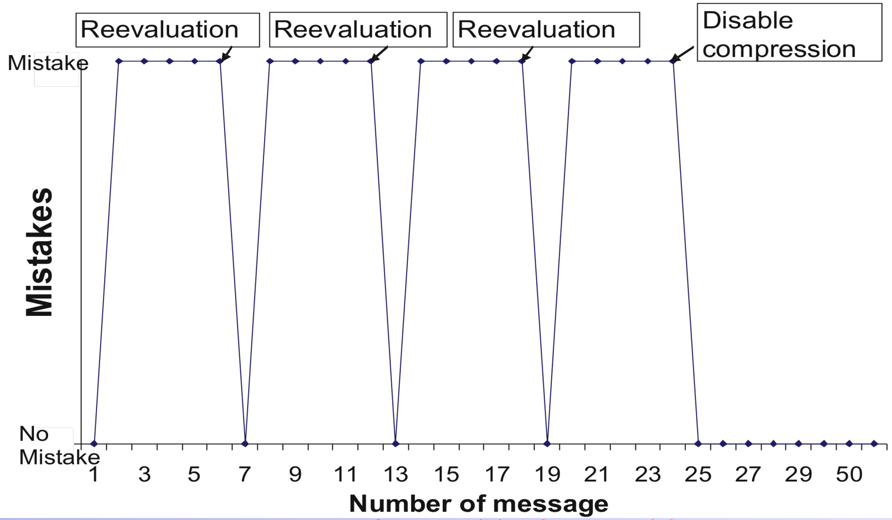
Different cases of re-evaluation



Low frequency mistakes



High frequency mistakes



Many reevaluations
in a short time

New Proposal: PRAcTICaL-MPI

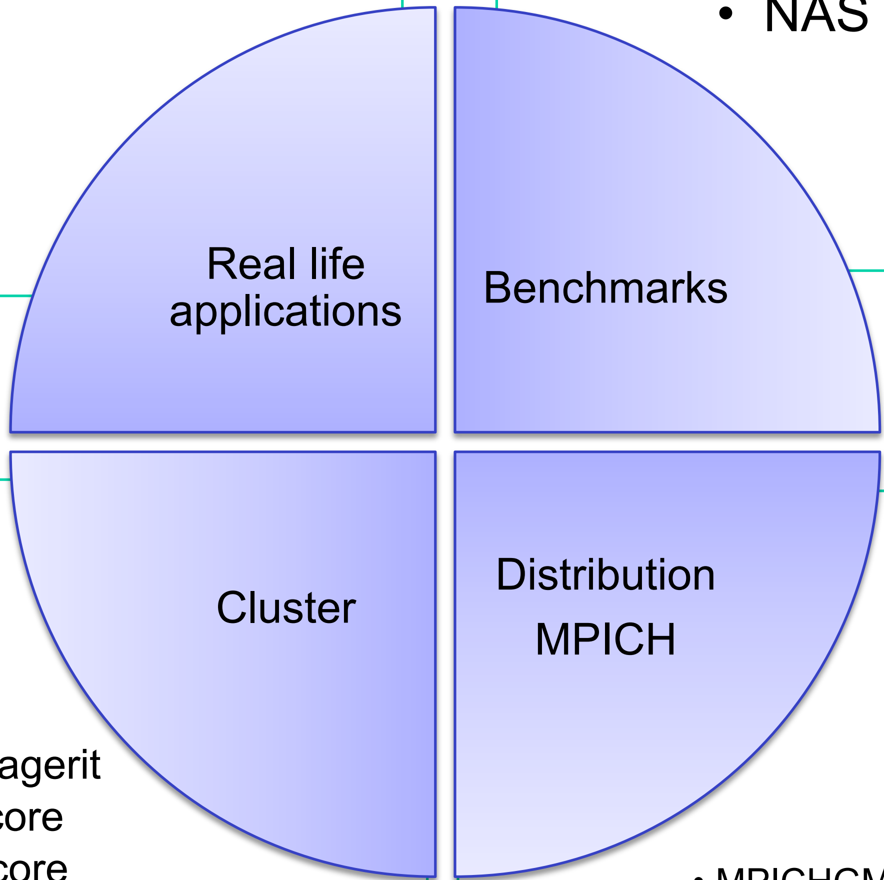
- PRAcTICaL-MPI: PoRtable AdpaTIve Compression Library
- New compression algorithms:
  - Snappy or PFOR
- Evaluate using OPEN MPI, XT MPI.
- Open question

Summary

Phd. Thesis evaluation tools

- BISP3D
- PSRG
- STEM-II

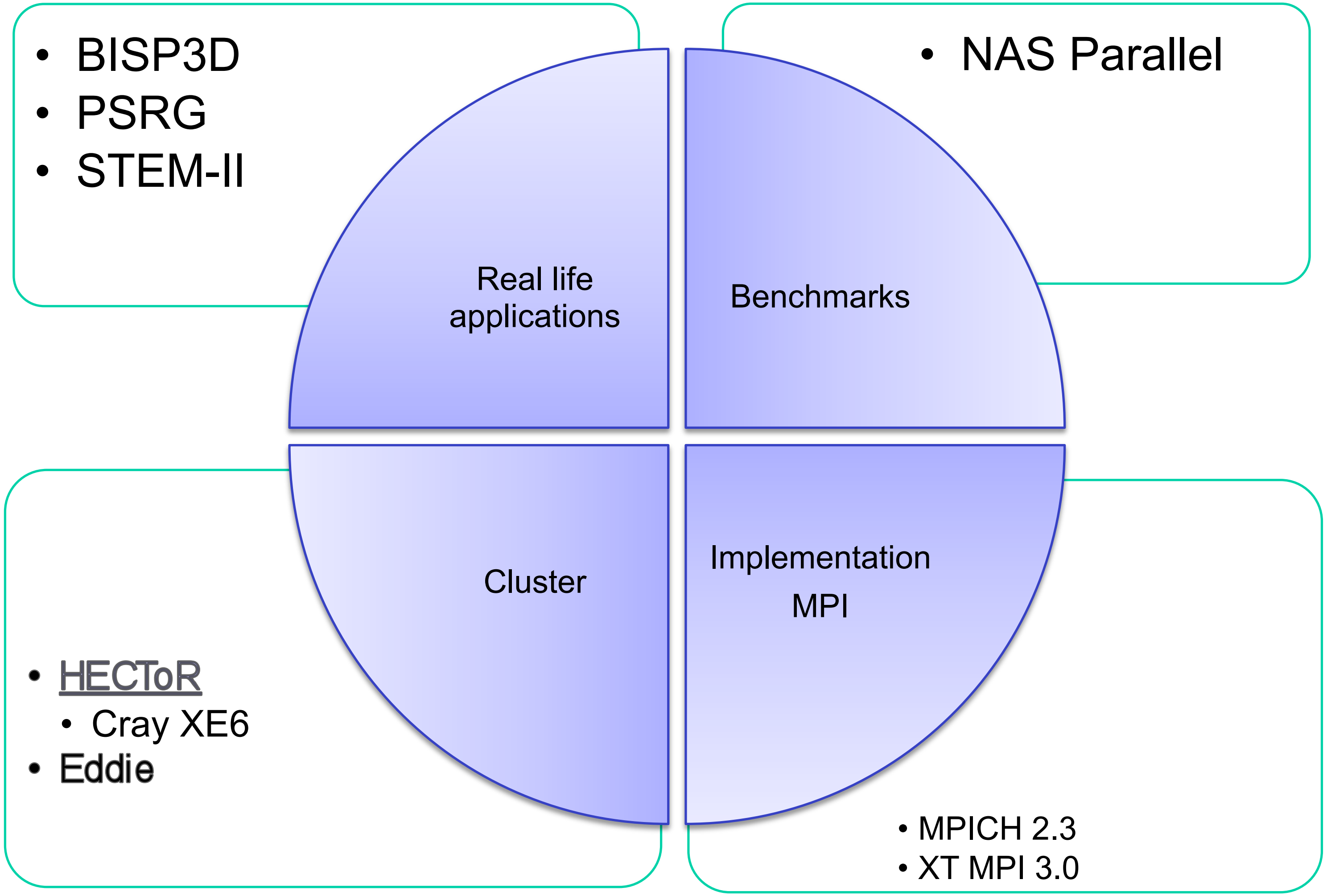
- NAS Parallel



- CESVIMA: Magerit
- UC3M: dual core
- UC3M: tetra core
- UC3M: dual core2

- MPICHGM-1.2.7.15NOGM

HPC Programme



HECToR

High End Computing Terascale Resource

"HECToR is the UK's high-end computing resource, funded by the UK Research Councils. It is available for use by academia and industry in the UK and Europe."

Features on HECToR Phase 2b

- 1856 compute nodes which contain two AMD 2.1 GHz 12-core Opteron processors => 44,544 cores

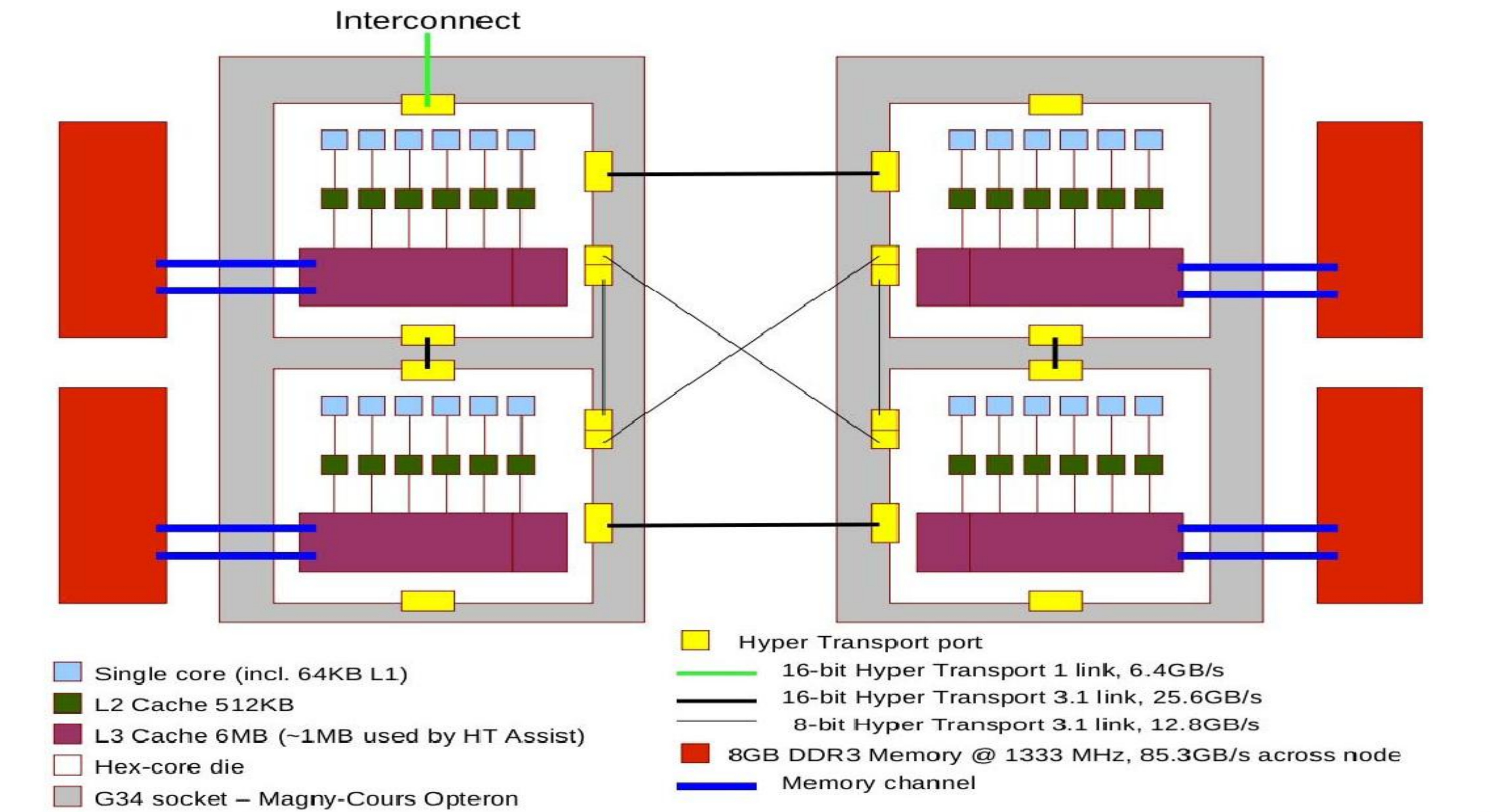
- Theoretical peak performance of 373 Tflops

- 32 GB main memory per processor, shared between 24 cores => total memory of 58 TB

- Gemini interconnect

- 12 IO nodes

XE6 24-core Magny Cours node



Running Jobs in HECToR


```
#!/bin/bash –login
#PBS -N My_rosa
#PBS -l mppwidth=20
#PBS -l mppnppn=2
#PBS -l walltime=00:20:00
#PBS -A x01-rfil

# Change to the direcotry that the job was
 submitted from cd $PBS_O_WORKDIR

# Launch the parallel job
aprun –n 20 -N 2 ./bisp3d arg1 arg2
```

- mppwidth: Request the total number of MPI tasks for your job.
- mppnppn: Tells the scheduler how many processes to place on a node

My example: 20 MPI processes, 2 processes per Node → 10 Nodes

EDDIE

- The compute component of Edinburgh Compute and Data Facility (ECDF), known as Eddie, offers a number of services aimed to satisy as best as possible all researchers' computational requirements.

Features of EDDIE on Mark2Phase1

- My experiments by using Mark2Phase1
- 130 IBM dx360M3 iDataPlex servers:
  - Two Intel Xeon E5620 quad-core processors (8 cores per node).
- Gigabit Ethernet network.
- Now it is available Mark2Phase2:
  - 156 IBM dx360M3 iDataPlex servers:
    - Two Intel Xeon E5645 six-core processors (12 cores per node).
    - Gigabit Ethernet network.

Running Jobs in EDDIE


```
#!/bin/sh
#$ -N My_rosa
#$ -cwd
#$ -pe openmpi_smp8_mark2 20
#$ -l h_rt=00:30:00


# Launch the parallel job
mpirun –np $NSLOTS ./bisp3d arg1 arg2
```

•NSLOTS: Request the total number of MPI tasks for your job.
•But you can not configure the number of nodes per core.

My example: 20 MPI processes

Running Jobs in EDDIE


```
#!/bin/sh
#$ -N My_rosa
#$ -cwd
#$ -pe openmpi_smp8_mark2 20
#$ -l h_rt=00:30:00


# Launch the parallel job
mpirun –np $NSLOTS ./bisp3d arg1 arg2
```

The file contained in $PE_HOSFILE have this:

eddie296.ecdf.ed.ac.uk **1**
ecdf@eddie296.ecdf.ed.ac.uk UNDEFINED

eddie328.ecdf.ed.ac.uk **2**
ecdf@eddie328.ecdf.ed.ac.uk UNDEFINED

eddie335.ecdf.ed.ac.uk **2**
ecdf@eddie335.ecdf.ed.ac.uk UNDEFINED

eddie336.ecdf.ed.ac.uk **2**
ecdf@eddie336.ecdf.ed.ac.uk UNDEFINED

eddie341.ecdf.ed.ac.uk **3**
ecdf@eddie341.ecdf.ed.ac.uk UNDEFINED

eddie350.ecdf.ed.ac.uk **3**
ecdf@eddie350.ecdf.ed.ac.uk UNDEFINED

eddie353.ecdf.ed.ac.uk **4**
ecdf@eddie353.ecdf.ed.ac.uk UNDEFINED

eddie364.ecdf.ed.ac.uk **3**
ecdf@eddie364.ecdf.ed.ac.uk UNDEFINED

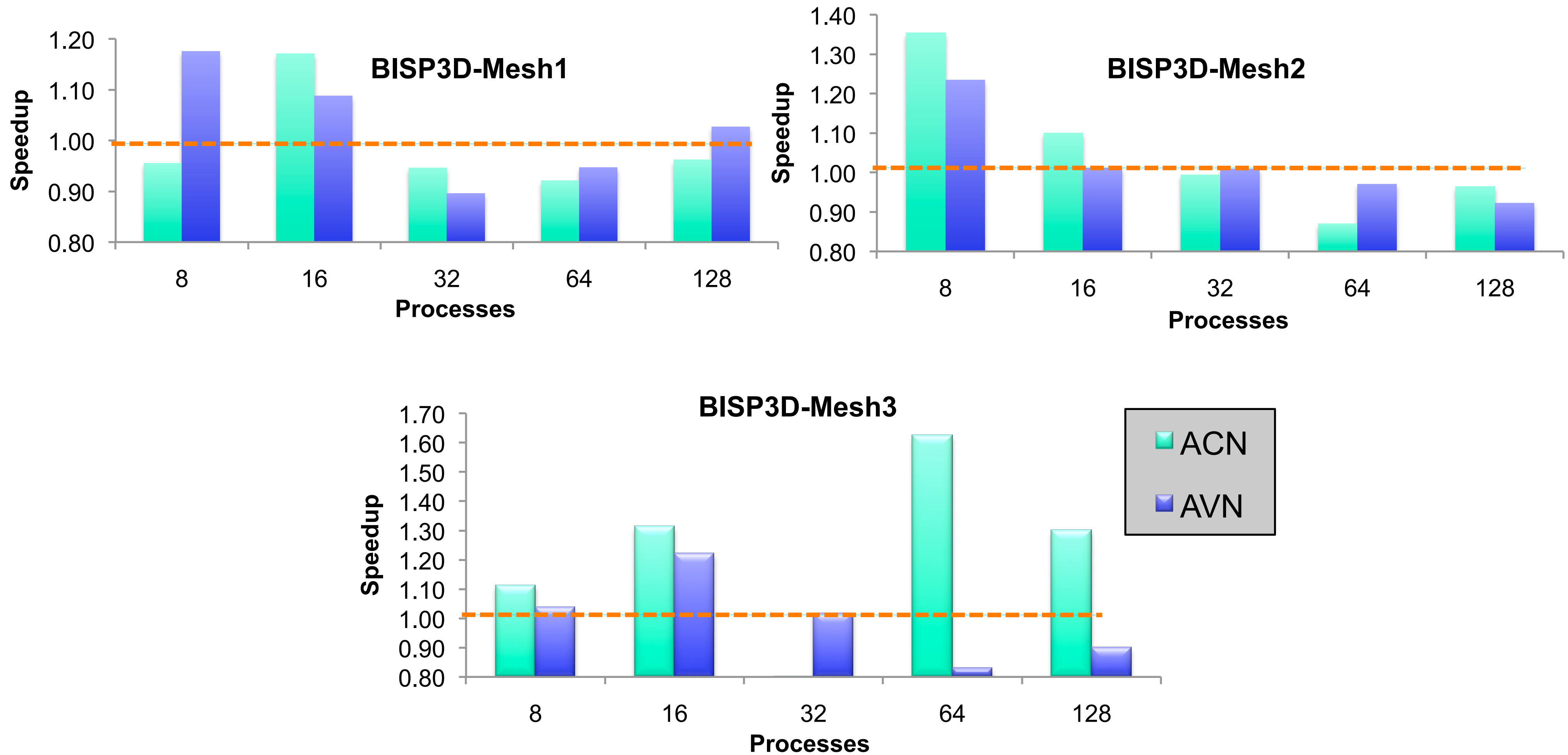
•NSLOTS: Request the total number of MPI tasks for your job.
•But you can not configure the number of nodes per core.

My example: 20 MPI processes

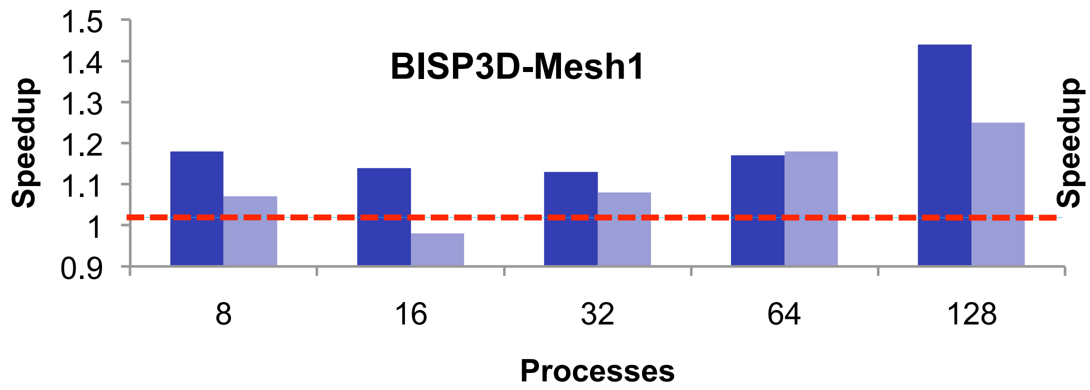
BISP3D

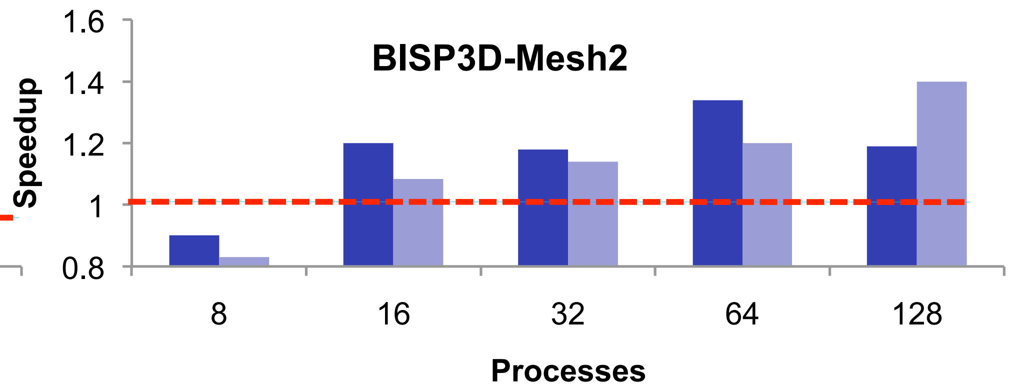
- 3-Dimensional simulator of BJT and HBT bipolar devices:
  - The goal is to relate electrical characteristics of the device with its physical and geometrical parameters.
- We have use 3 different different devices
  - Each bipolar device it is represented by a mesh.
  - Load represent the number of elements per node (in a mesh).
- Uses Two_Phase IO to write results in disk
  - Original BISP3D sequential writes.
- Irregular application (non contiguous data).

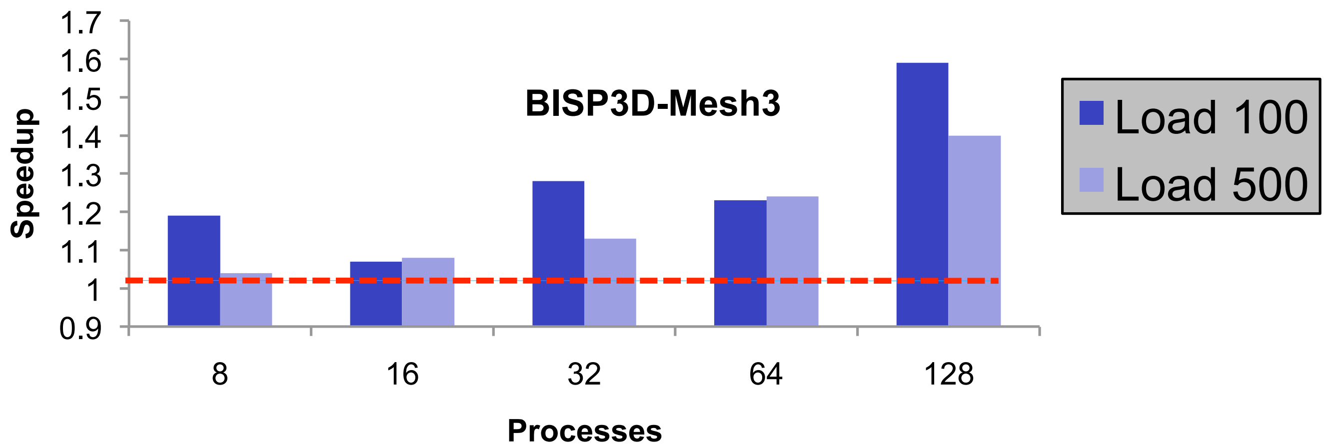
Evaluations of IO aggregator patterns (EDDIE)



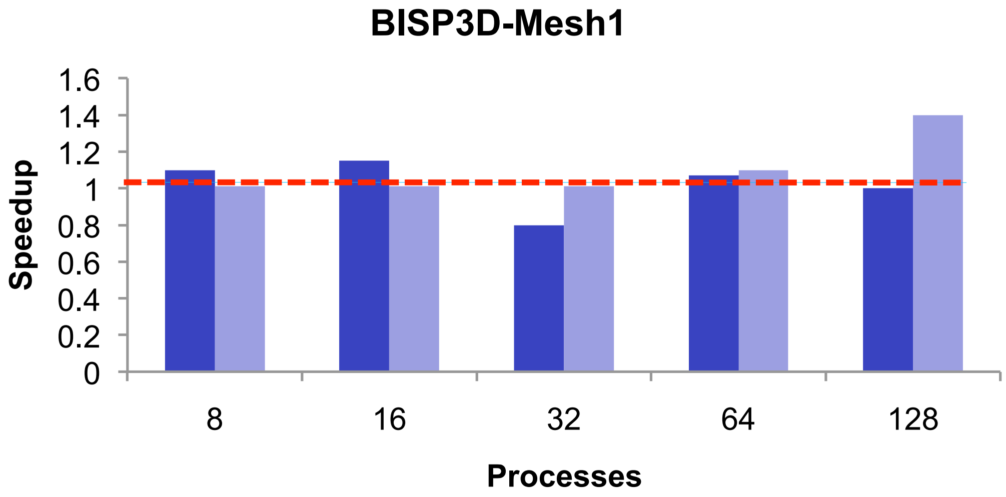
Evaluations of Run Adaptive Compression (EDDIE)

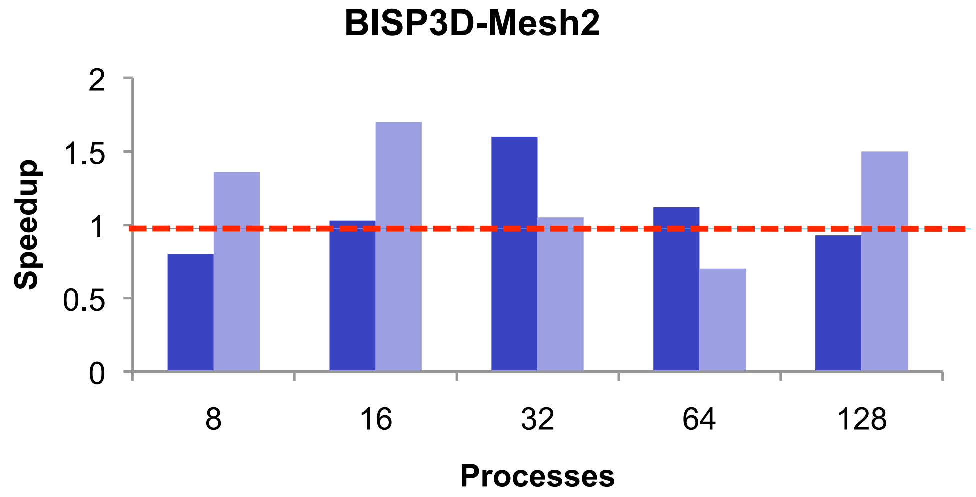


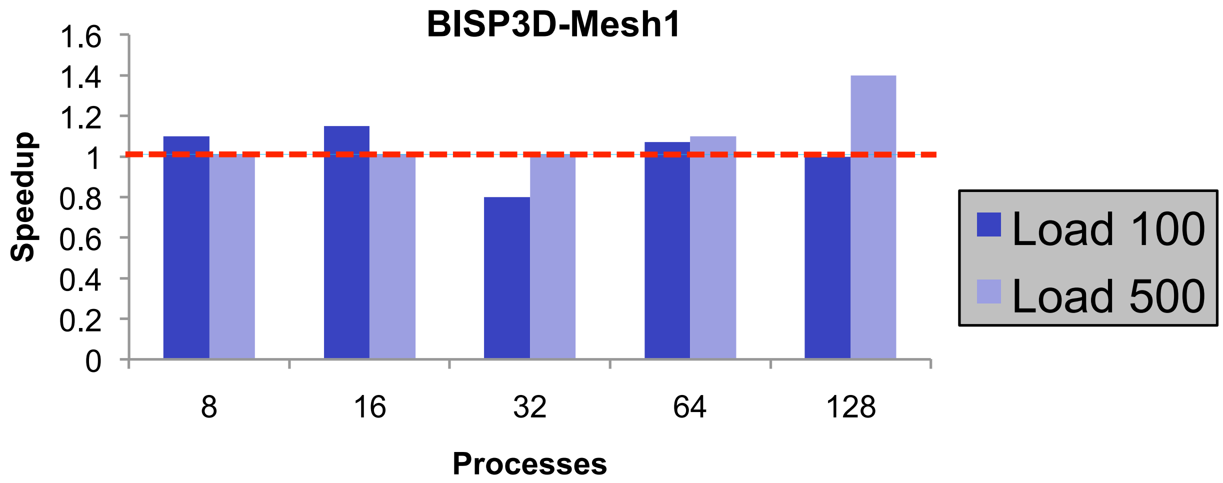




Evaluations of Run Adaptive Compression (HECToR)







Questions??

- Thanks!!

Dynamic and Adaptive optimization techniques to enhance the performance of MPI applications by using **HECToR** and **Eddie** clusters**.**

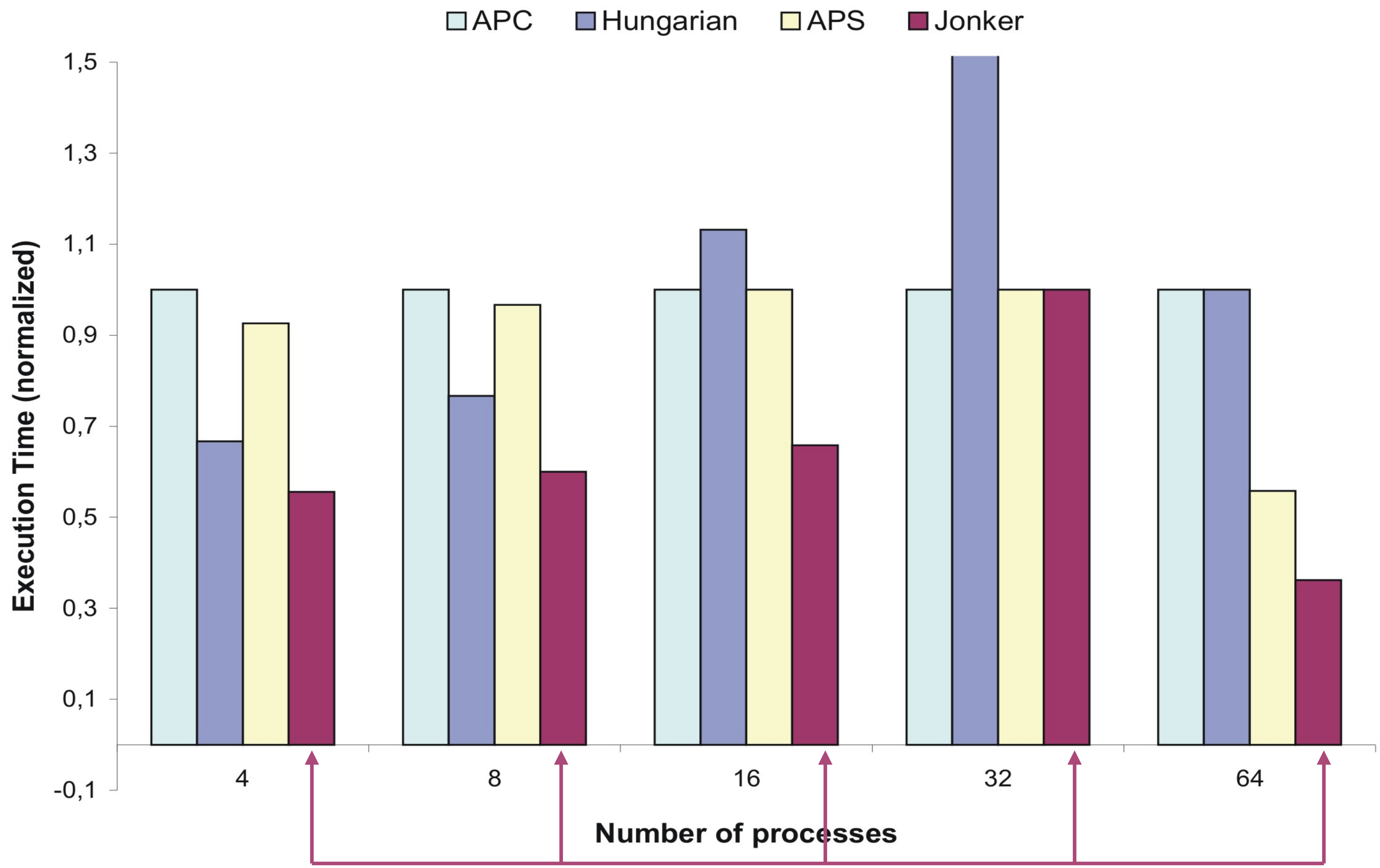
Author: Rosa Filgueira Vicente
University of Edinburgh

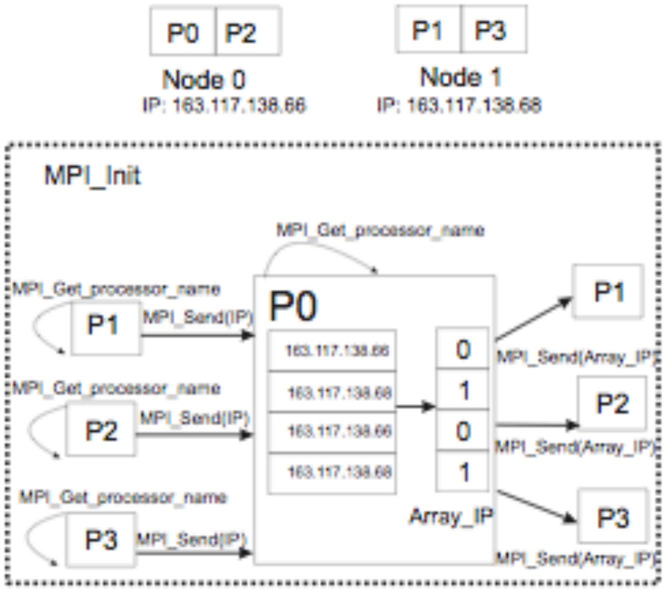
Linear Assignment Problem (I)

- LAP computes the optimal assignment of m items to n elements given an m x n cost matrix.

- Several algorithms have been developed for LAP:
  - Hungarian algorithm.
  - Jonker and Volgenant algorithm.
  - APC and APS Algorithms.
- All algorithms produce the same assignment.
- The difference is the time to compute the optimal allocation.

3.1 Linear Assigment Problem (II)




P0
P2
P1
P3
Node 0
IP: 163.117.138.66
Node 1
IP: 163.117.138.68
MPI_Init
MPI_Get_processor_name
MPI_Get_processor_name
P1
MPI_Send(IP)
MPI_Get_processor_name
P2
MPI_Send(IP)
MPI_Get_processor_name
P3
MPI_Send(IP)
P0
163.117.138.66
163.117.138.68
163.117.138.66
163.117.138.68
0
1
0
1
Array_IP
P1
MPI_Send(Array_IP)
P2
MPI_Send(Array_IP)
P3
MPI_Send(Array_IP)