Dynamic optimization techniques to enhance scalability and performance of MPI-based applications on Multi-core cluster.

Author: Rosa Filgueira Vicente
University: University Carlos III of Madrid

Summary

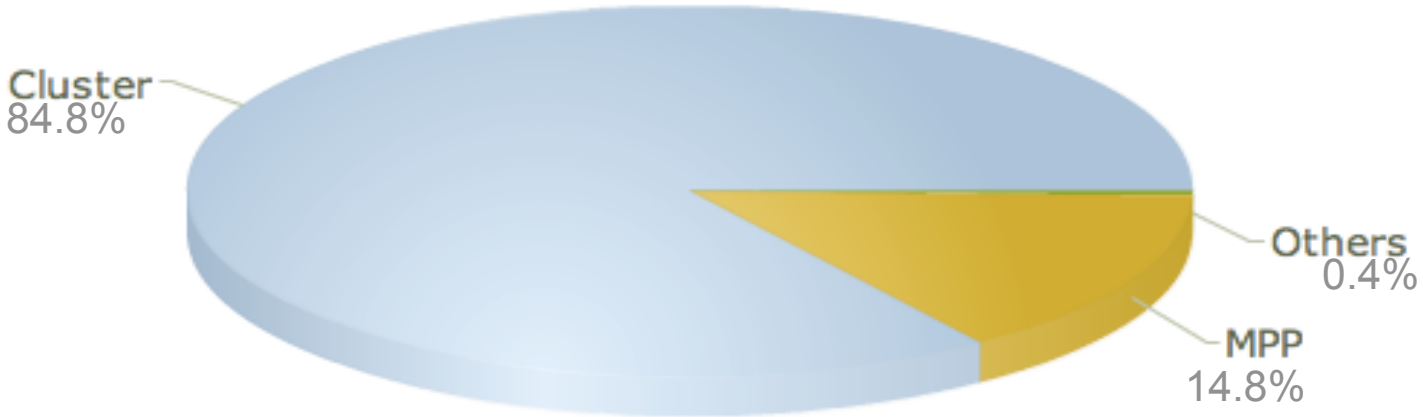
1. Problem description
2. Main Objectives
3. Strategies for improve the performance of collective I/O operation
4. Strategies for improve the performance communication operations.
5. Evaluation tools.

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. Evaluation tools

Problem description

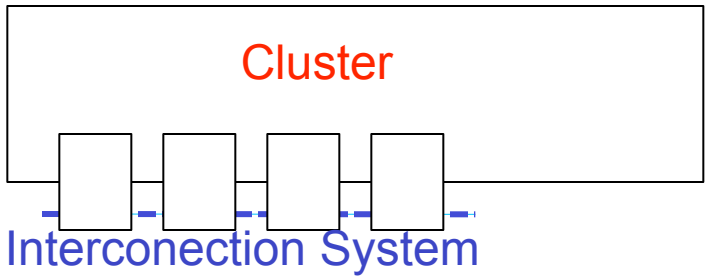
- Parallel computation on cluster has become the most common solution for HPC application.



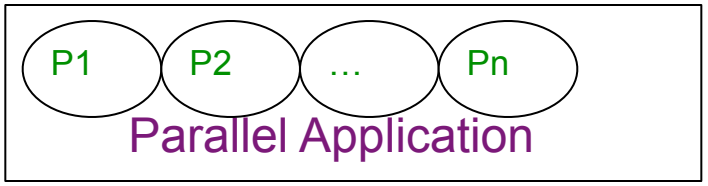
TOP 500
February 2011

Problem description

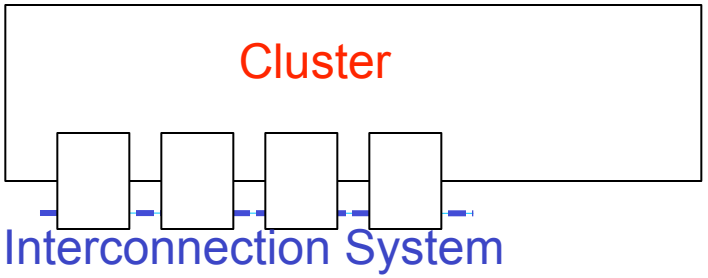
- Cluster is a group of linked computers, working together closely thus in many respects forming a single computer. The components of a cluster are commonly, but not always, connected to each other through fast local area networks



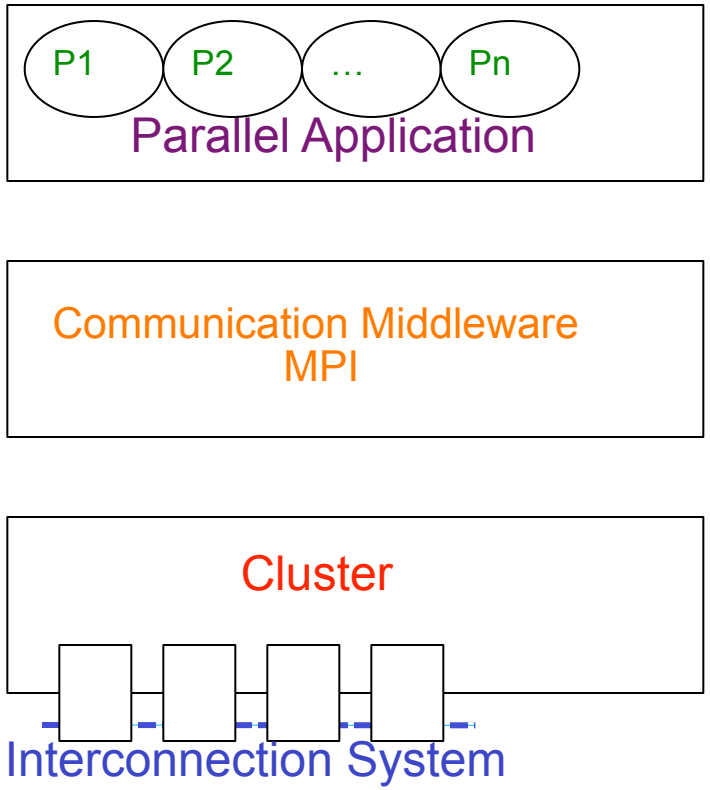
Problem description



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



Problem description



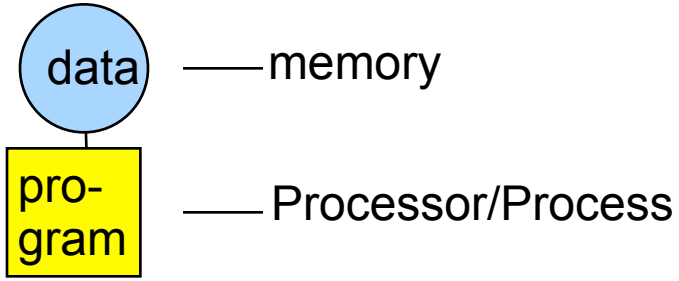
- One of most communication middleware used is the standard **MPI** (Message Passing Interface)

What is MPI (Message Passing Interface)?

- Standardized message passing library specification (IEEE)
  - for parallel computers, clusters and heterogeneous networks
  - not a specific product, compiler specification etc.
  - many implementations, MPICH, LAM, OpenMPI …
- Portable, with Fortran and C/C++ interfaces.
- Many functions
- Real parallel programming

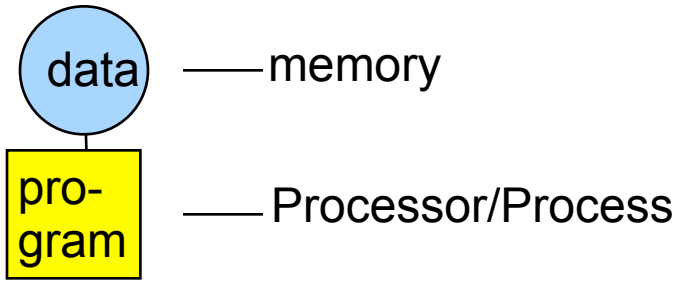
The Message-Passing Programming Paradigm

- Sequential Programming Paradigm

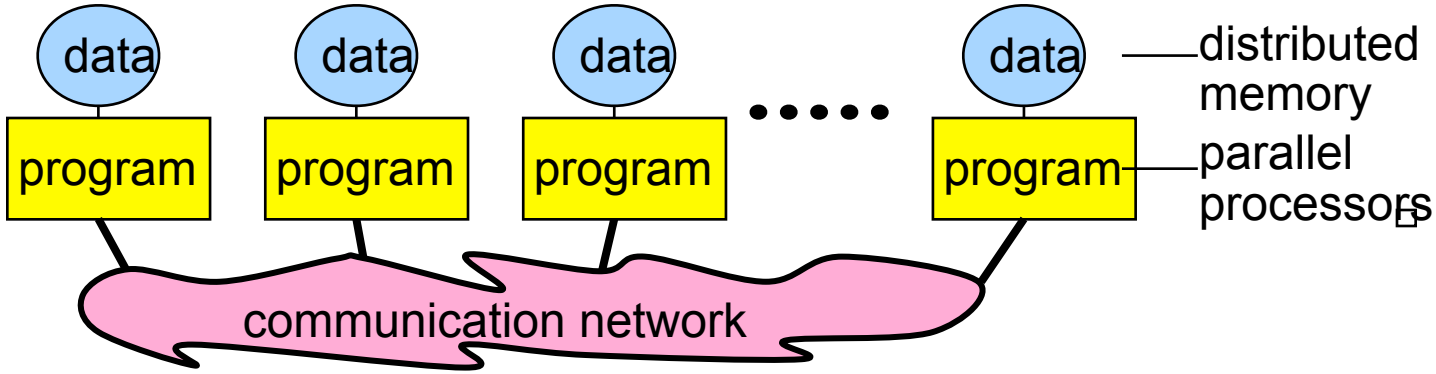


The Message-Passing Programming Paradigm

- Sequential Programming Paradigm

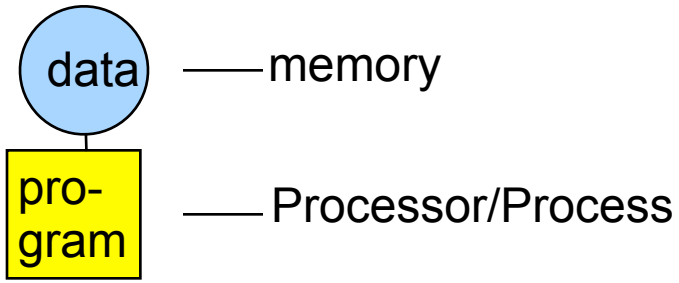


- Message-Passing Programming Paradigm



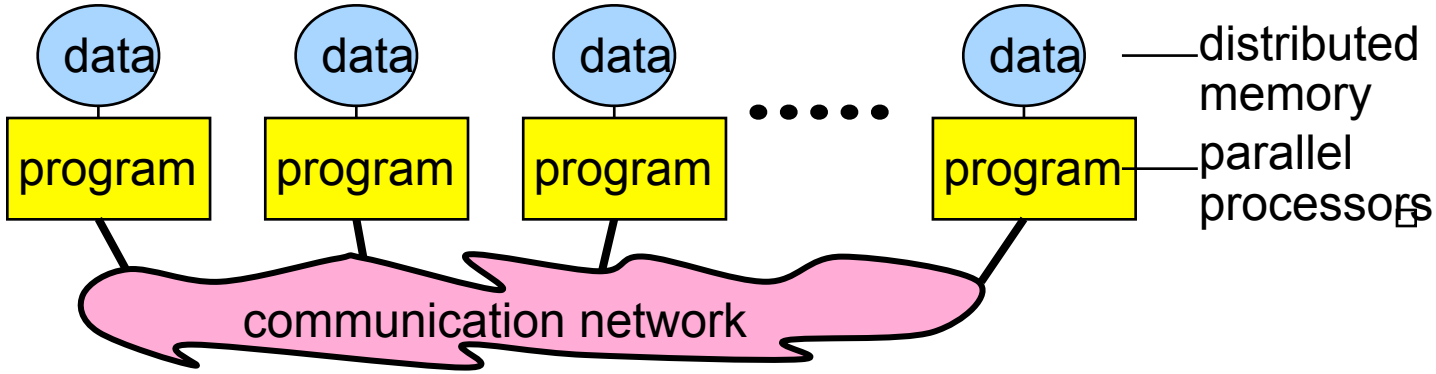
The Message-Passing Programming Paradigm

- Sequential Programming Paradigm



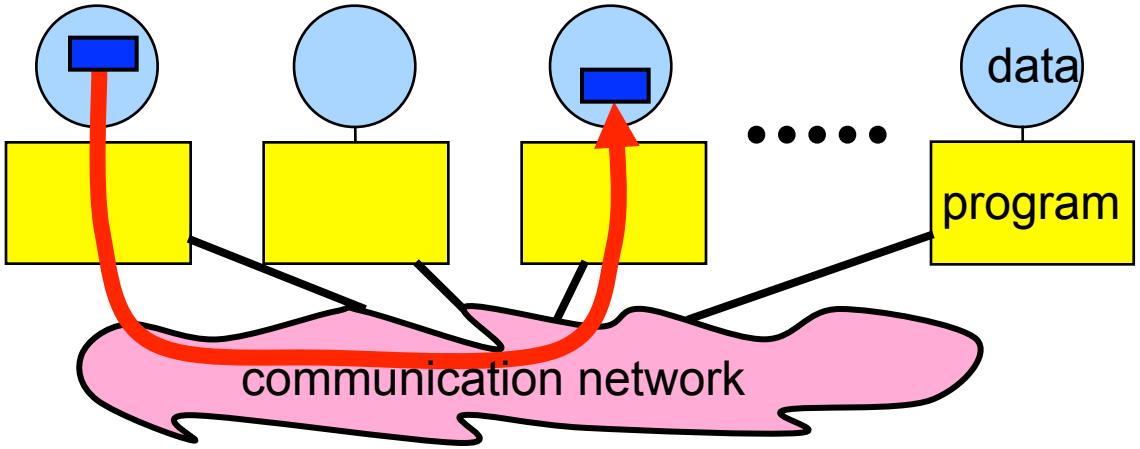
A processor may run many processes

- Message-Passing Programming Paradigm

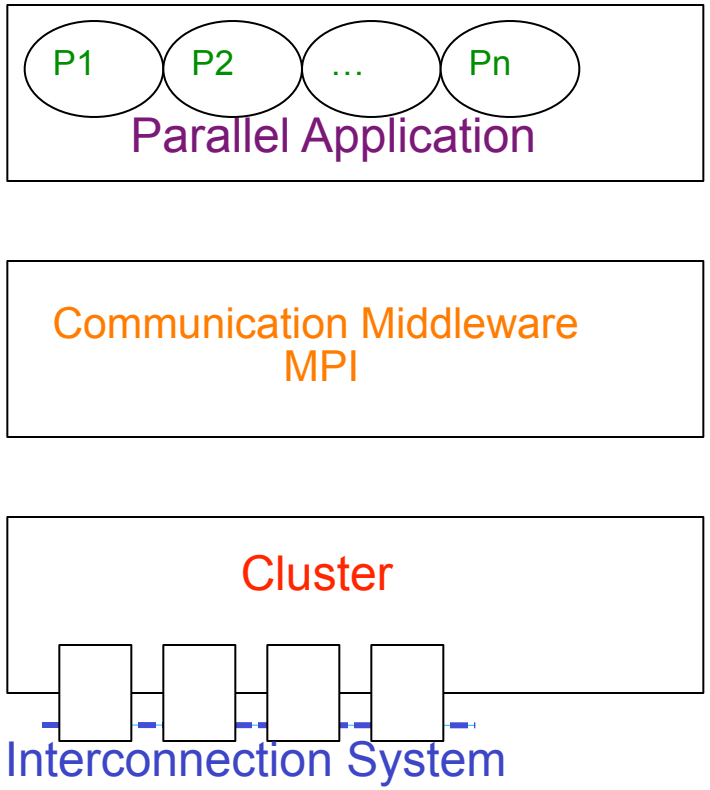


Message passing

- Messages are packets of data moving between sub-programs
- Necessary information for the message passing system:
  - sending process
  - source location
  - source data type
  - source data size
  - receiving process
  - destination location
  - destination data type
  - destination buffer size



Problem description

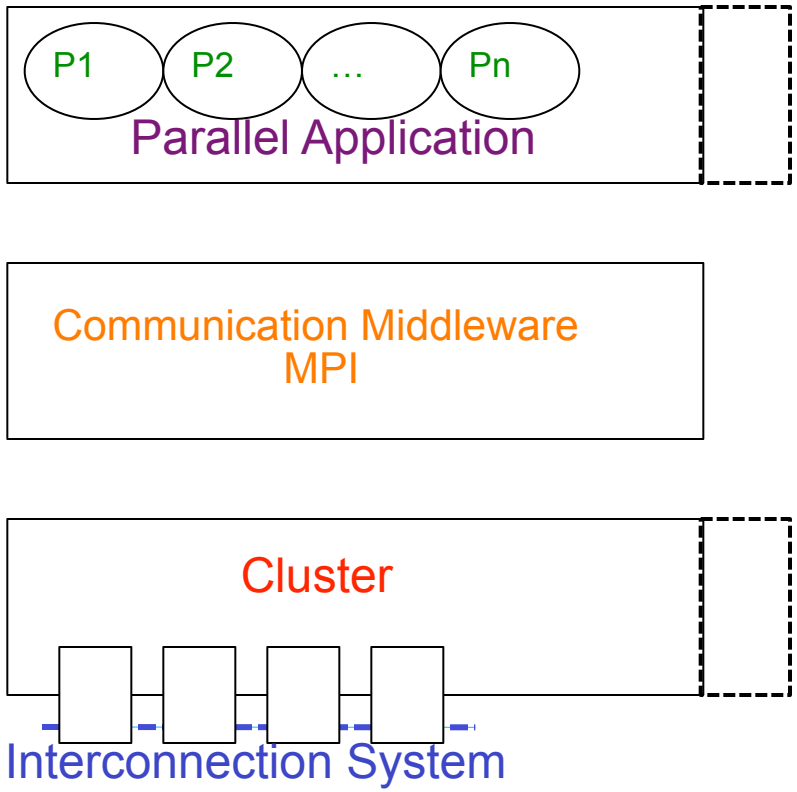


- One of most communication middleware used is the standard **MPI** (Message Passing Interface)
- Different implementations:
  - **MPICH**
  - OpenMPI
  - LAM
  - CHIMP-MPI …

Phd.Thesis: MPICH 1.2

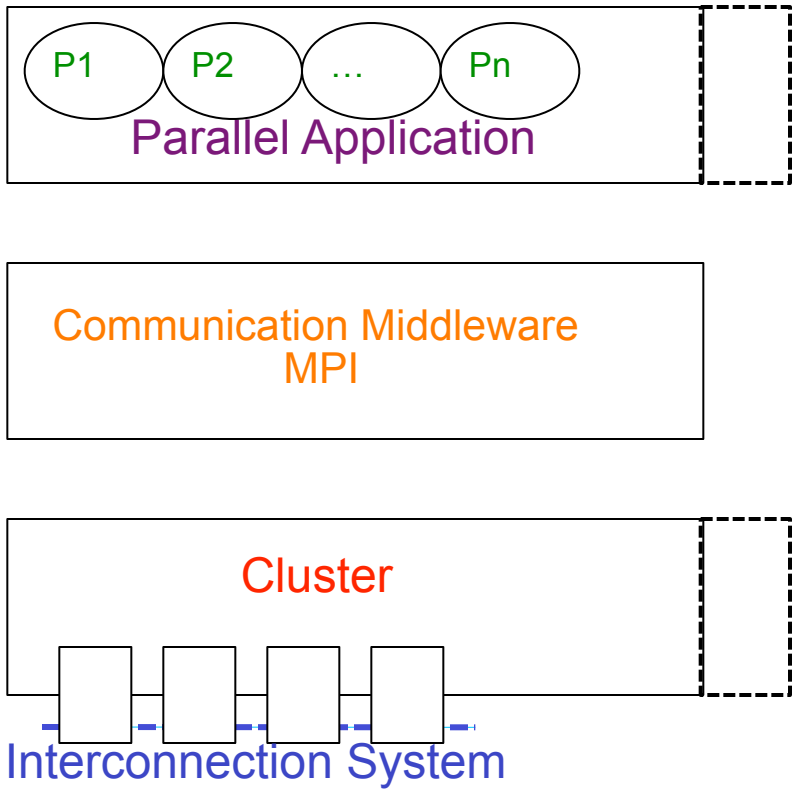
Now : MPICH 2

Problem description: trend



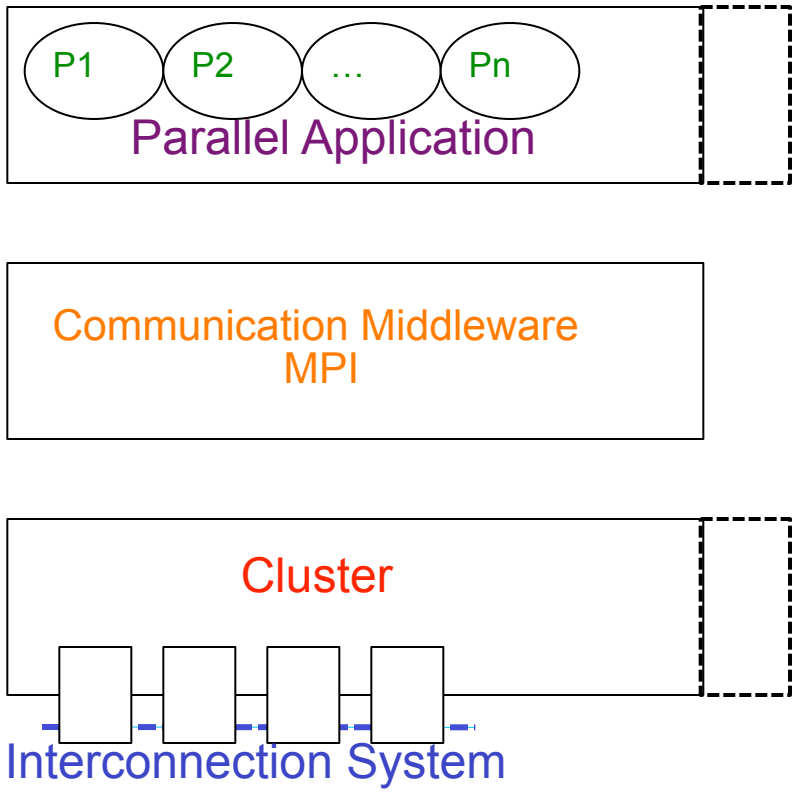
- Multicore processors provides a flexible way to increase the computation capability of cluster
- System performance may be improved with multicore but bottleneck from other components could reduce the scalability.

Problem description: Bottleneck



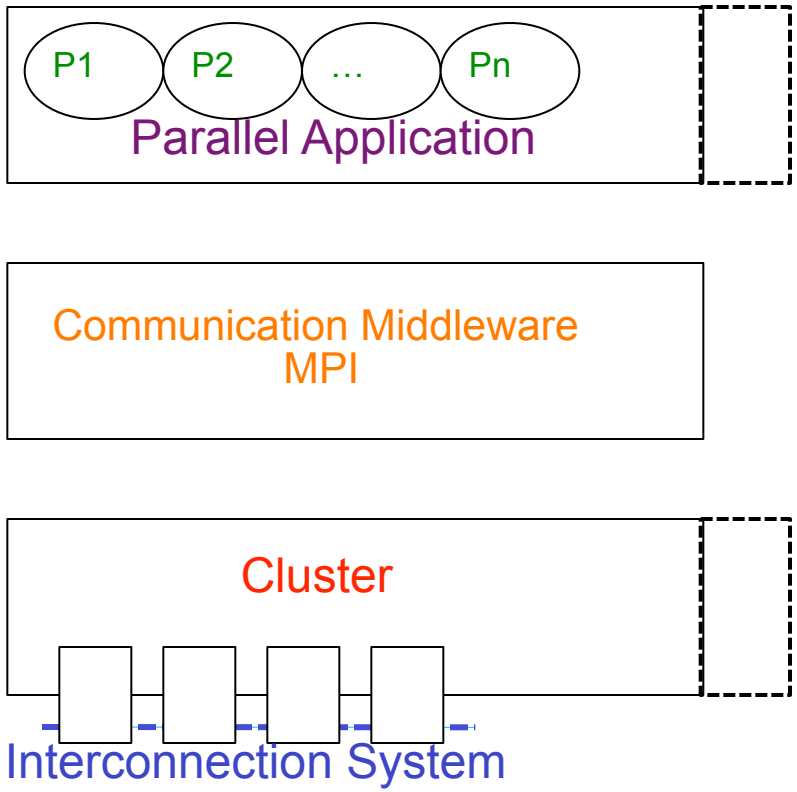
- Bottleneck :
  - I/O subsystem
  - Communication subsystem

Problem description: Bottleneck



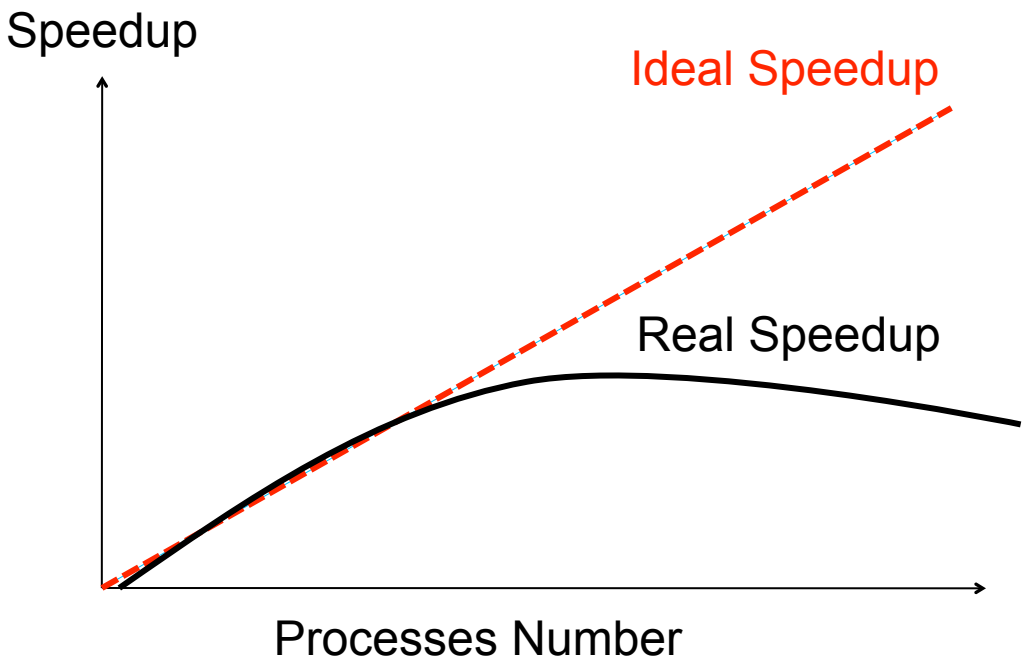
- I/O Bottleneck:
  - I/O requests initiated by multiple cores and besides when non-contiguous disk accesses is used.

Problem description: Bottleneck



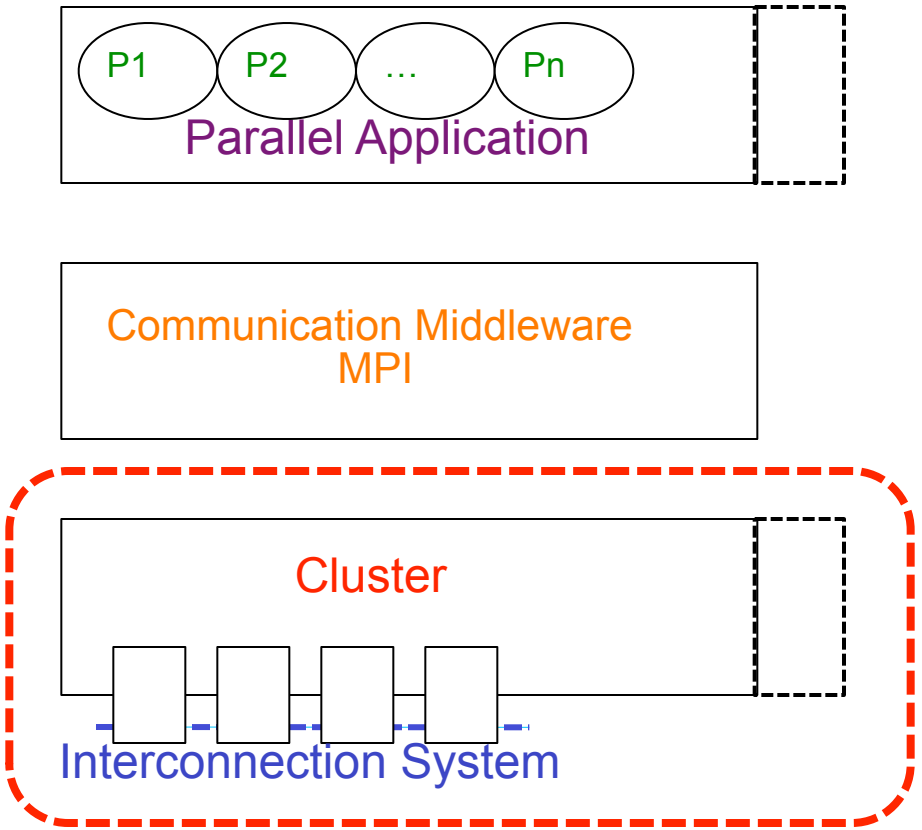
- Communication Bottleneck:
  - Network used are very fast and low latency.
  - Computational capability in multicore very high.
  - The frequency of message increase a lot → insufficient the increase of the bandwidth and latency

Problem description: Bottleneck



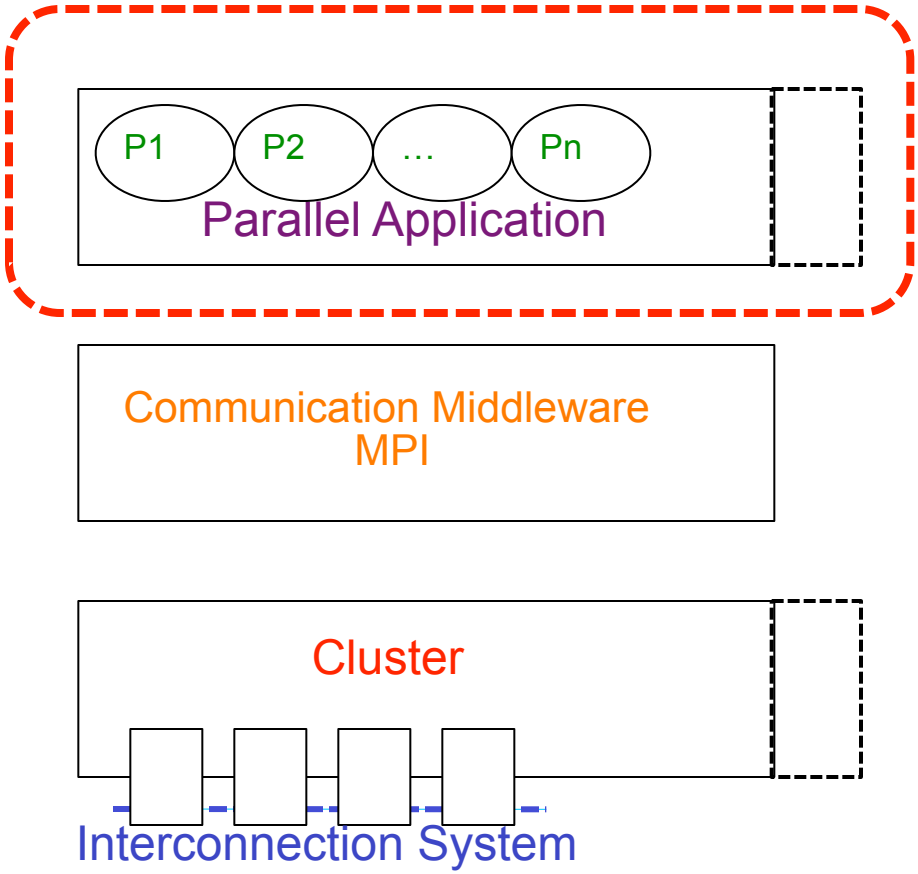
- Communication and I/O saturation:
  - Scalability problem
  - Performance problem

Problem description: possible solutions



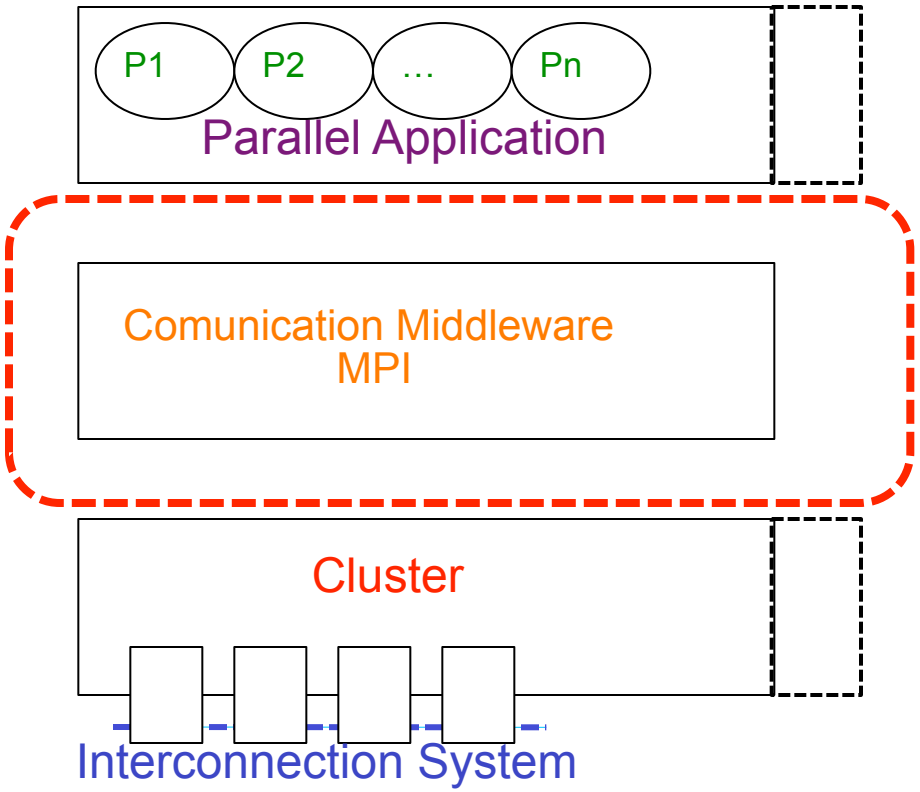
- 1) Improve network
  - Expensive.
  - Limited to current technology

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

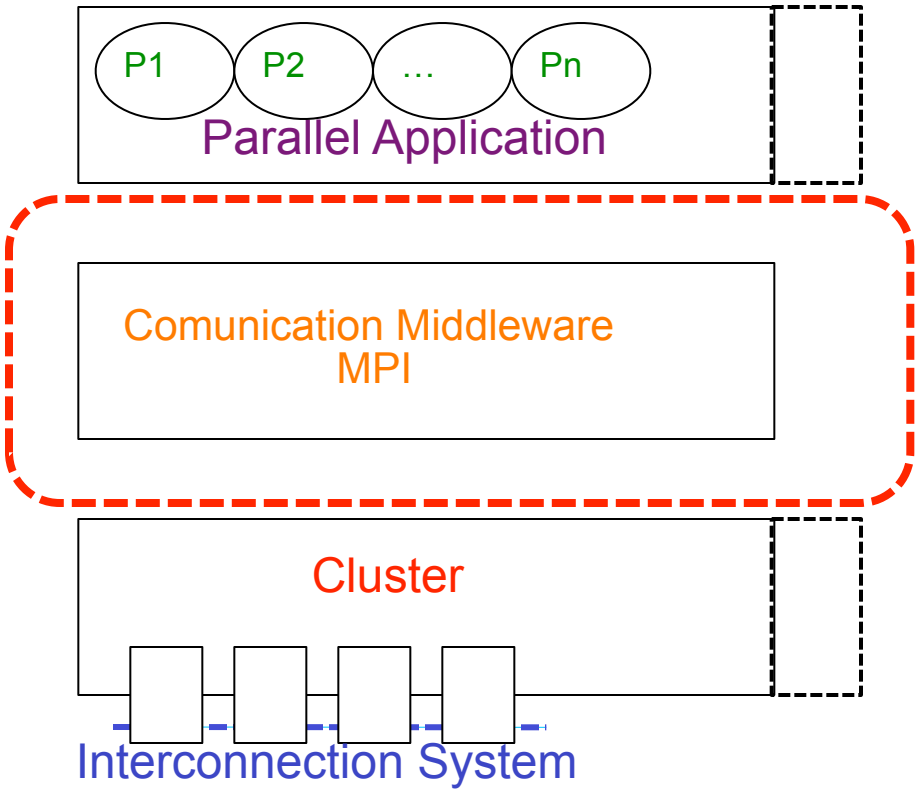


- 3) Improve the communication middleware
  - Portability
  - Greater user benefited
  - Lower Cost
  - Transparent:
    - Users
    - Applications

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 I/O and communication operations.
5. Evaluation

Main objetives



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

Specific objetives

1. Reduction of the number of communications in collective I/O operations.

2. Reduction of transferred data volume in communications.

Summary

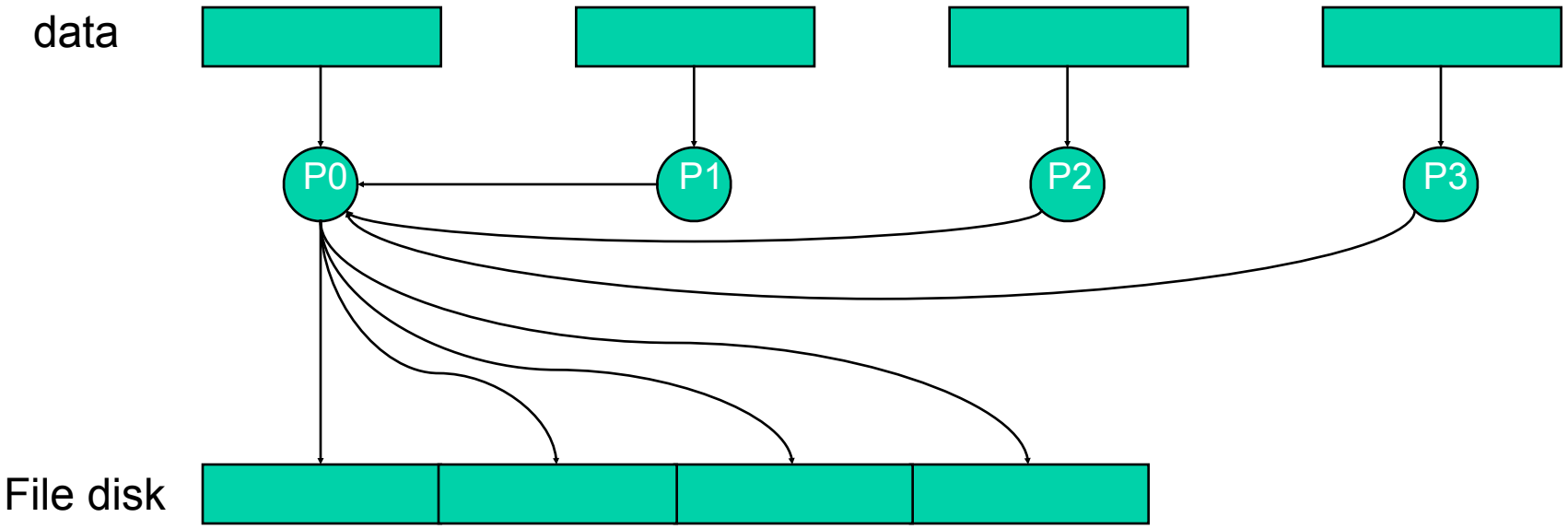
1. Problem description
2. Main Objectives
3. Strategies for improve the performance of collective I/O operation
4. Strategies for improve the performance of I/O and communication operations.
5. Evaluation tools

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

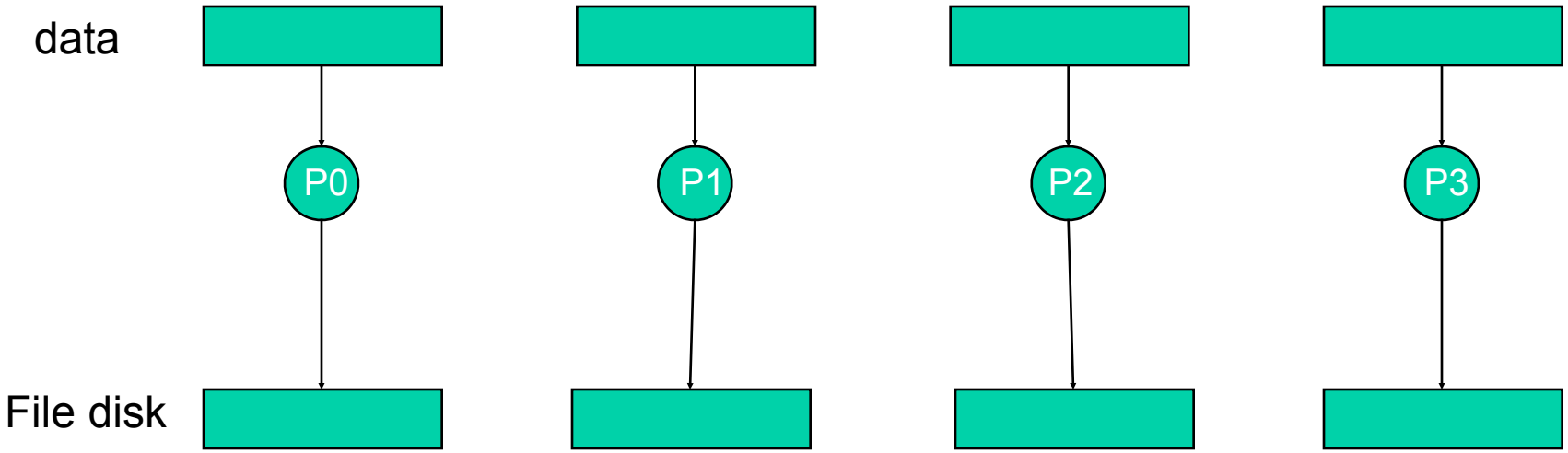
Sequential I/O

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



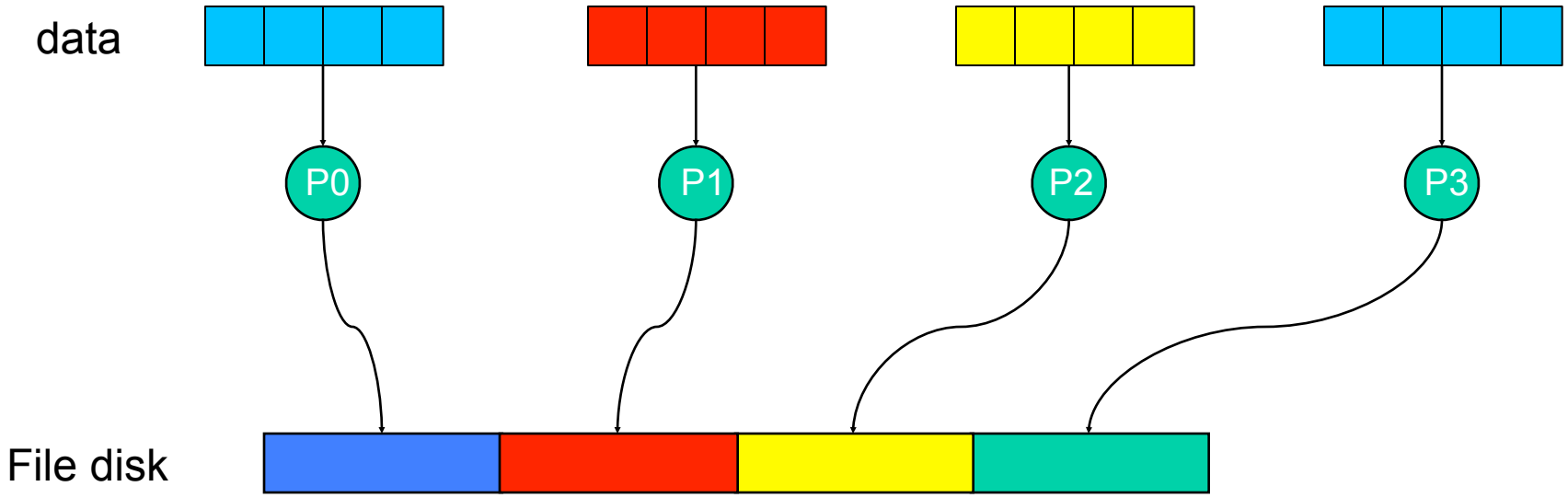
Individual I/O

- Each process writes to a separate file



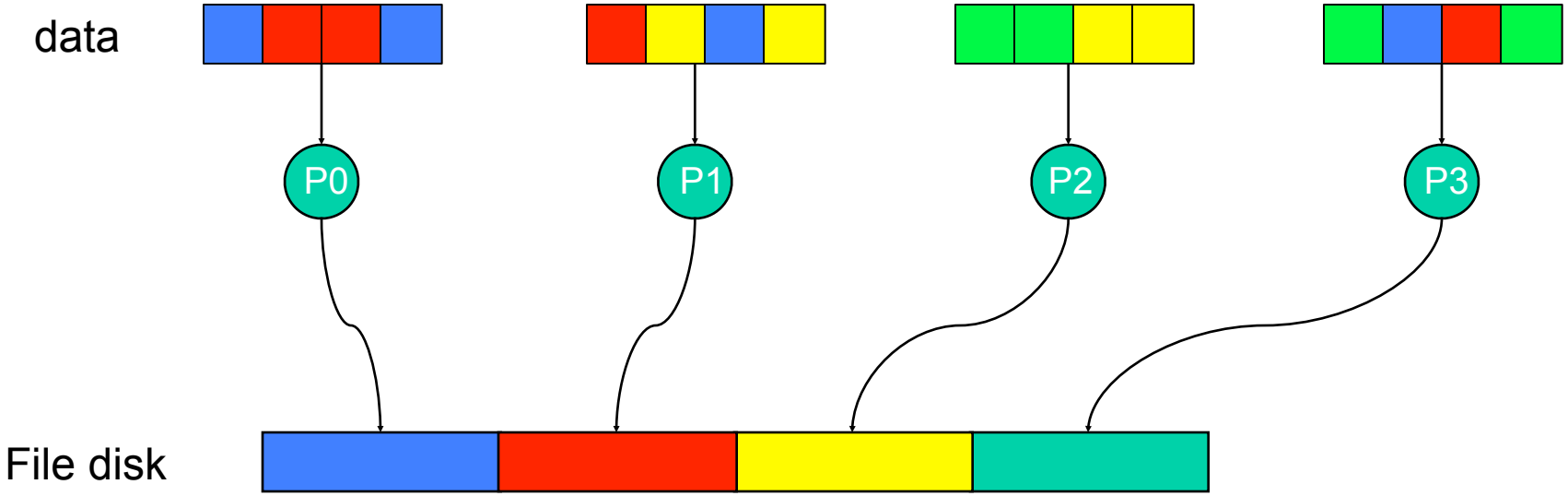
Collective I/O

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



Collective I/O

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



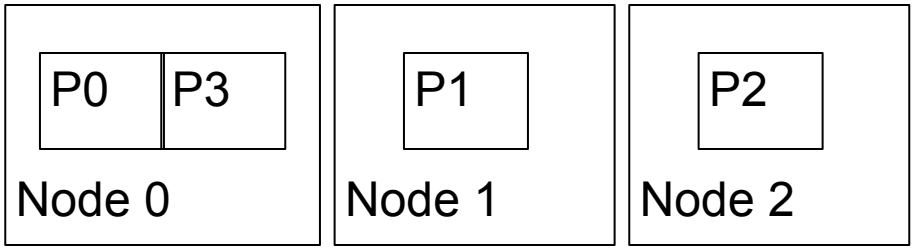
Two_Phase I/O

- 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.

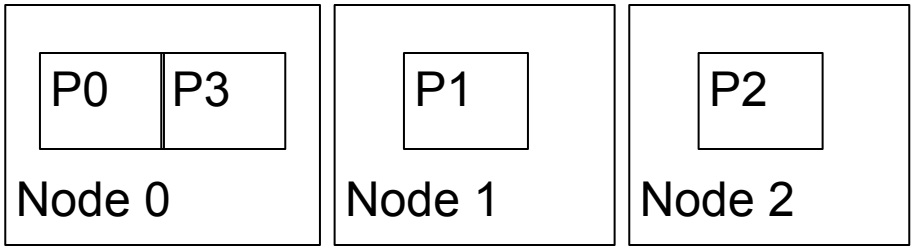
Two_Phase I/O “problem”

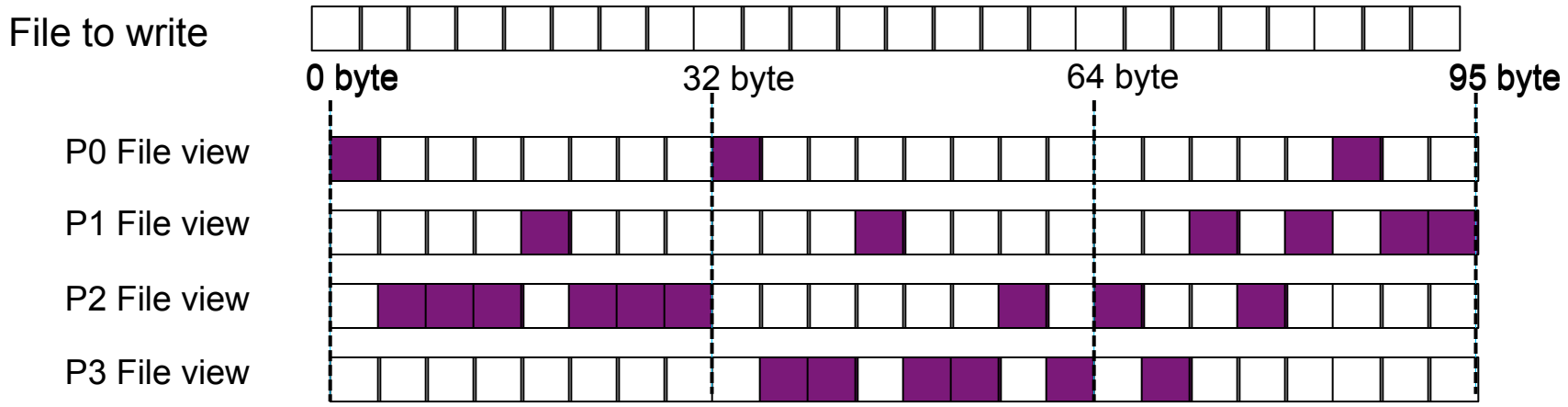
- The assignment of FD to aggregators is fixed.
- Independent of data distribution.
- Default aggregator Pattern:
  - So many aggregator as nodes.
  - Create an array of nodes, ordered by rank (process identifier in MPI), and assign each aggregator one process.
  - When there are more than process per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

Example of Two-Phase I/O

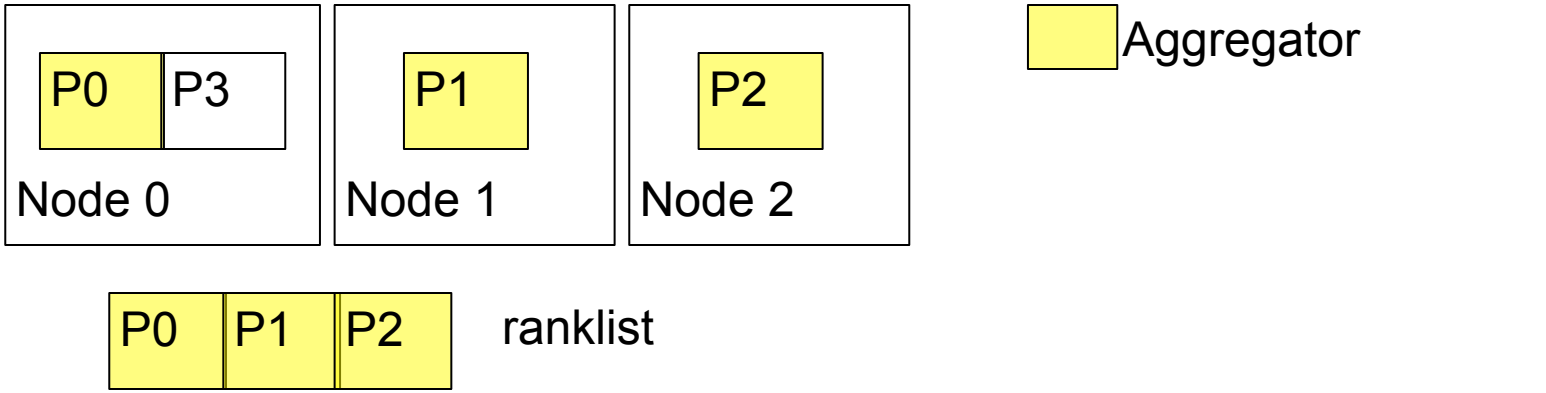


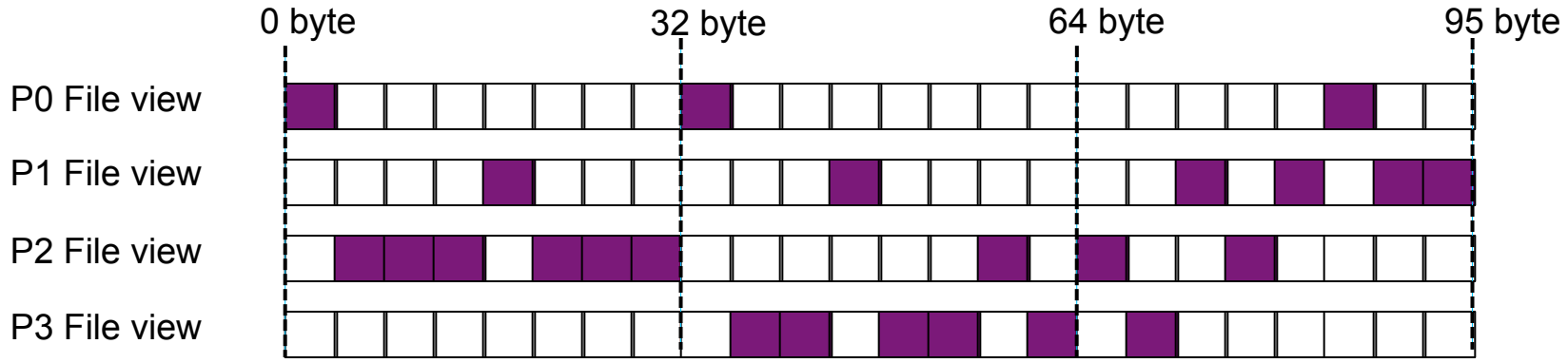
Example of Two-Phase I/O





Example of Two-Phase I/O

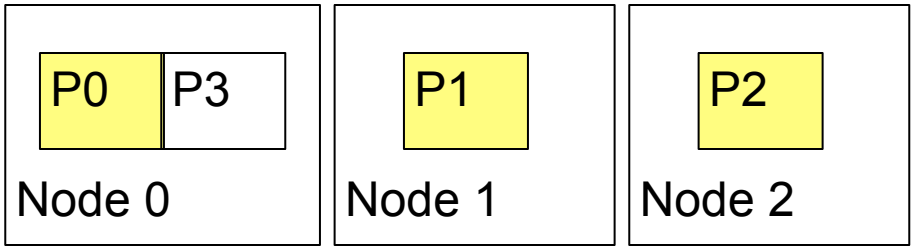




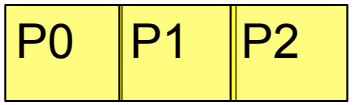
Local data

Non local data

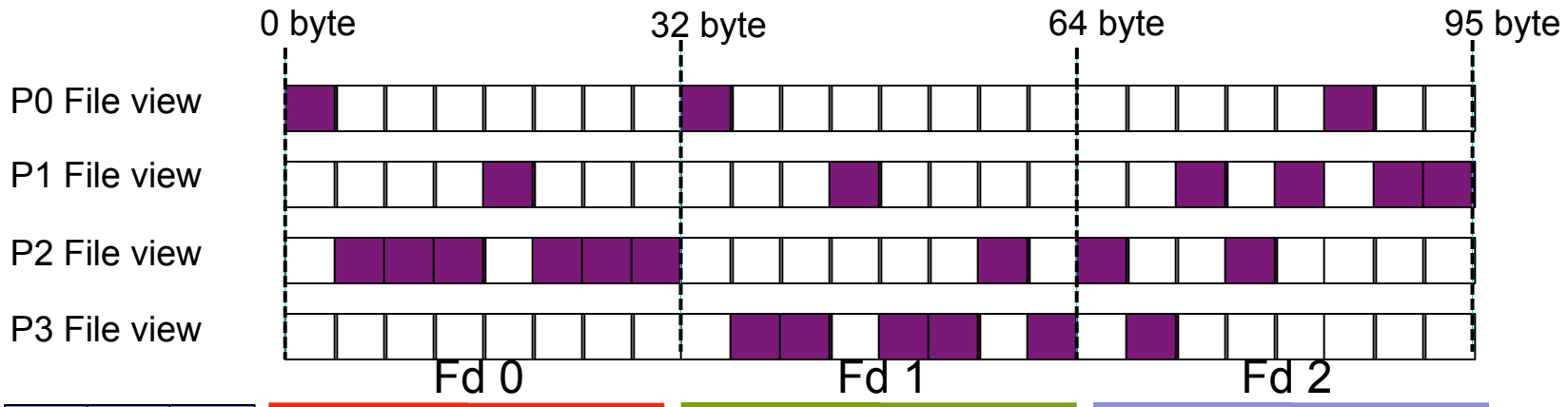
Example of Two-Phase I/O



Aggregator

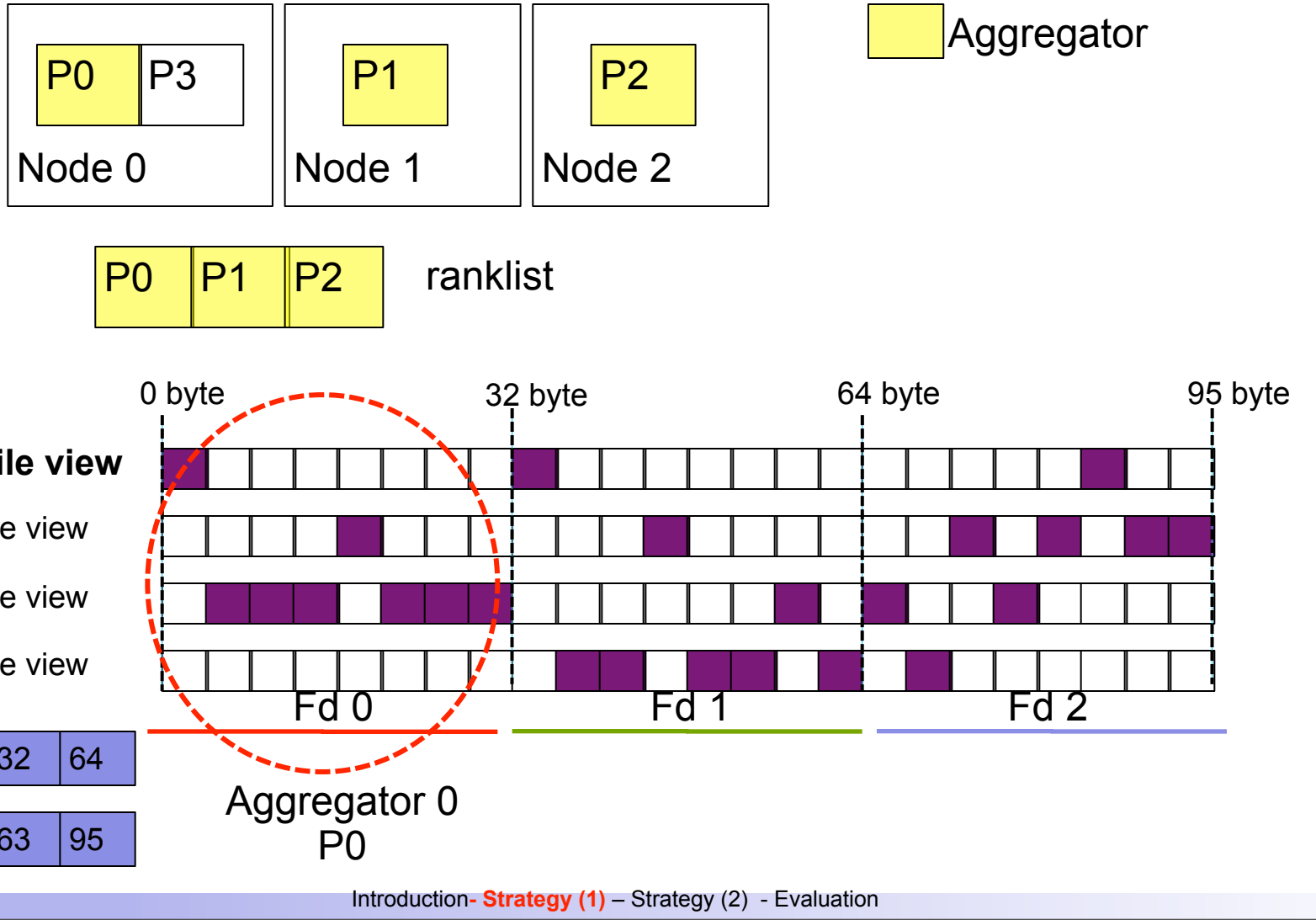


ranklist

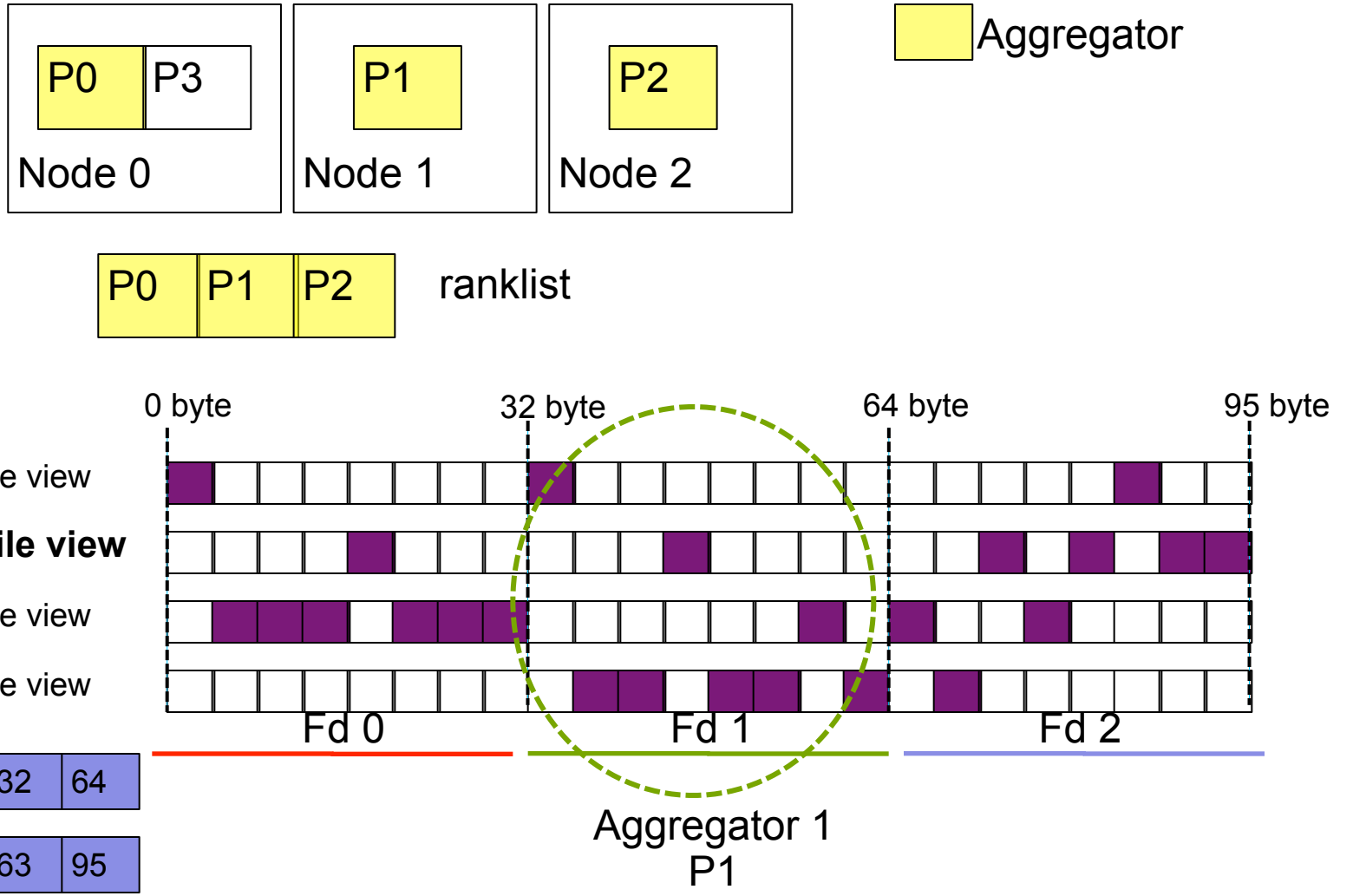


Fd_begin	0	32	64
Fd_end	31	63	95

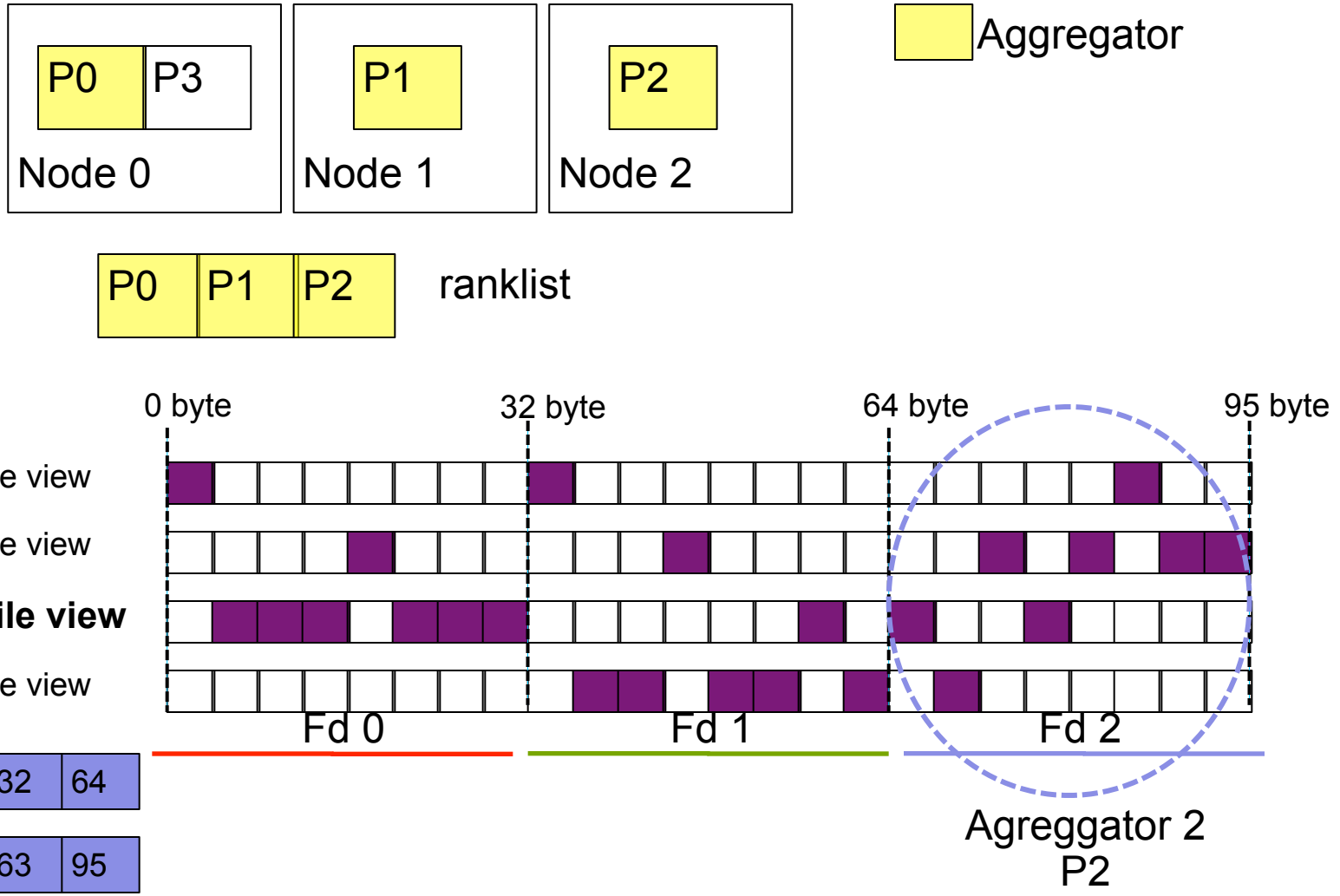
Example of Two-Phase I/O



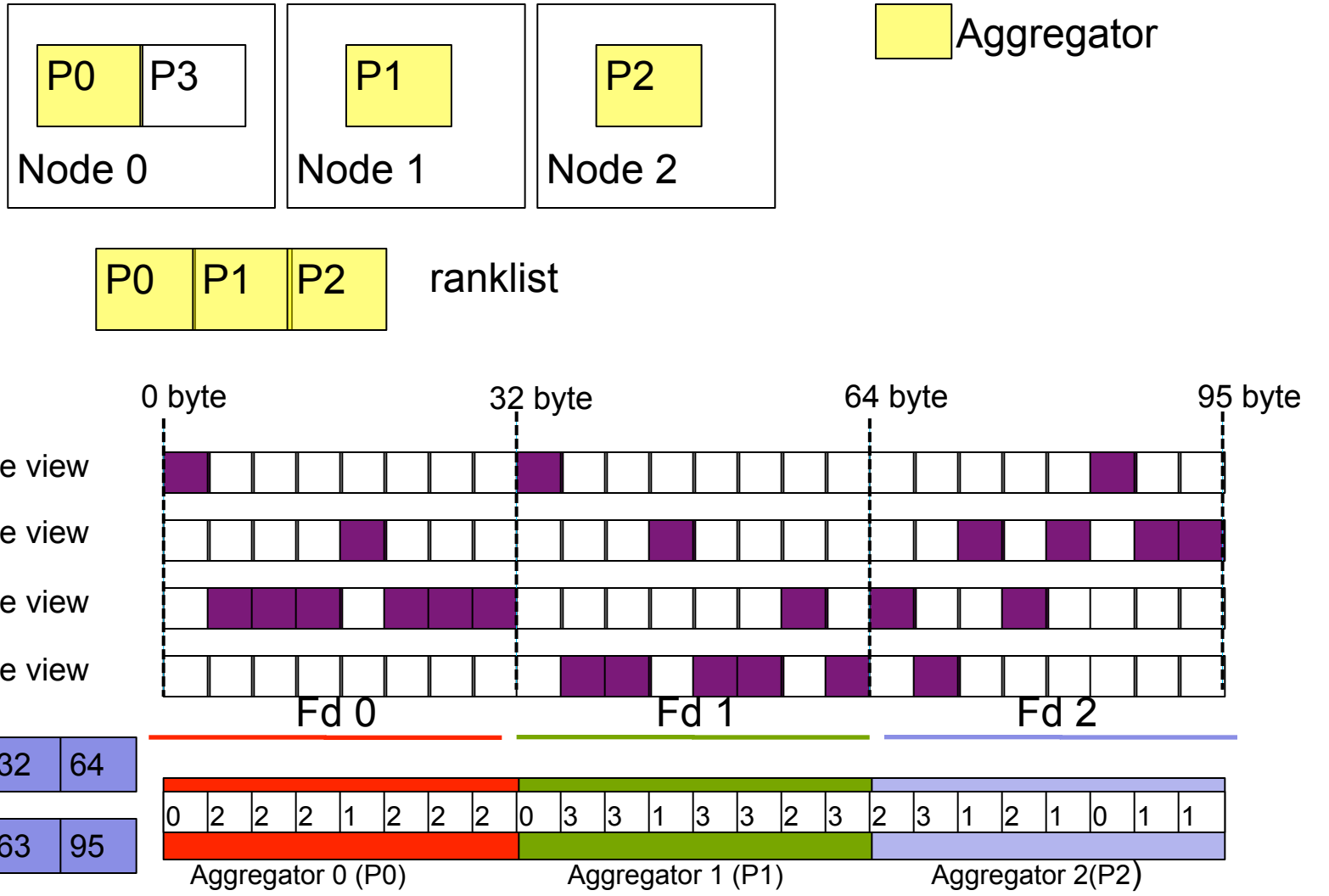
Example of Two-Phase I/O



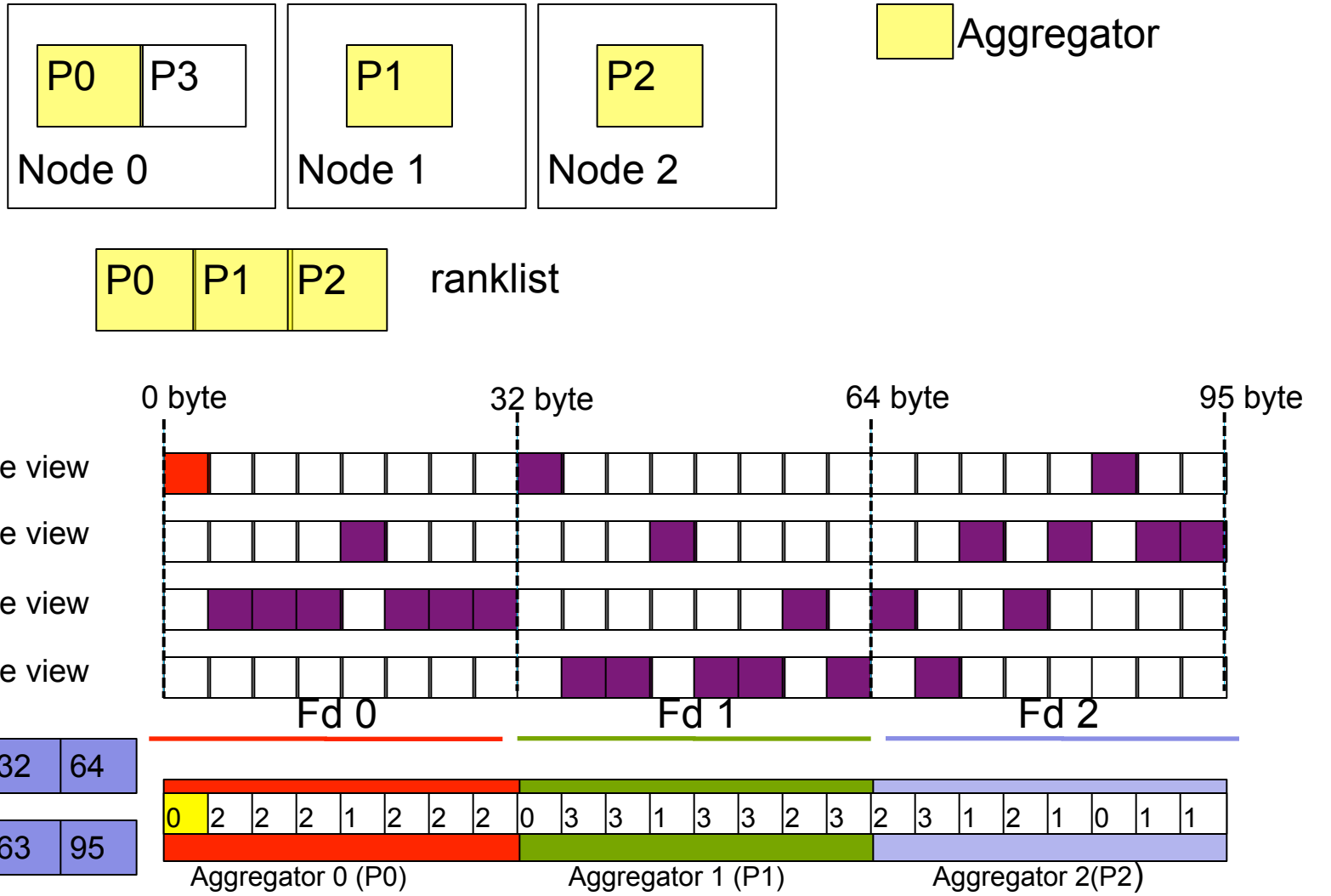
Example of Two-Phase I/O



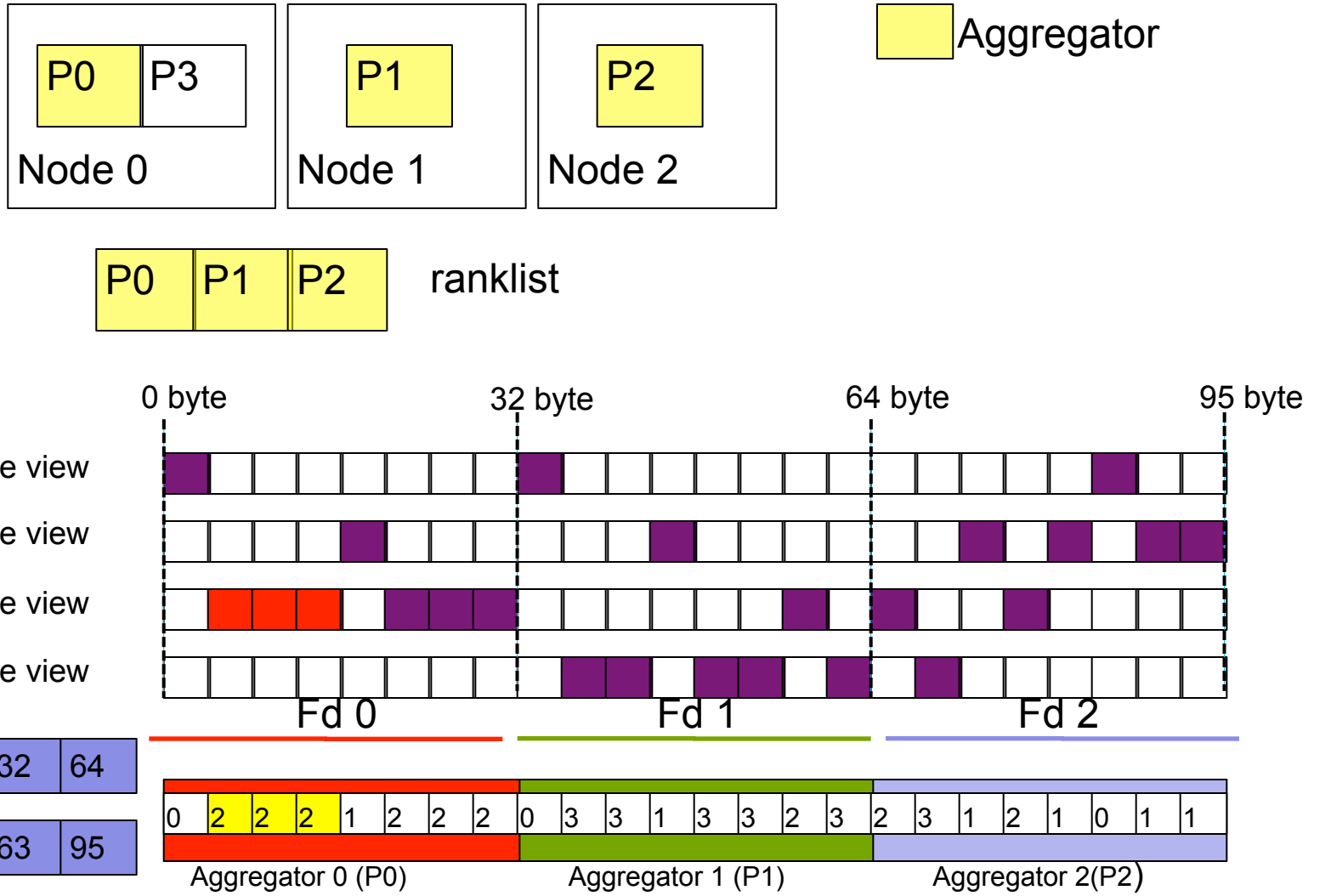
Example of Two-Phase I/O



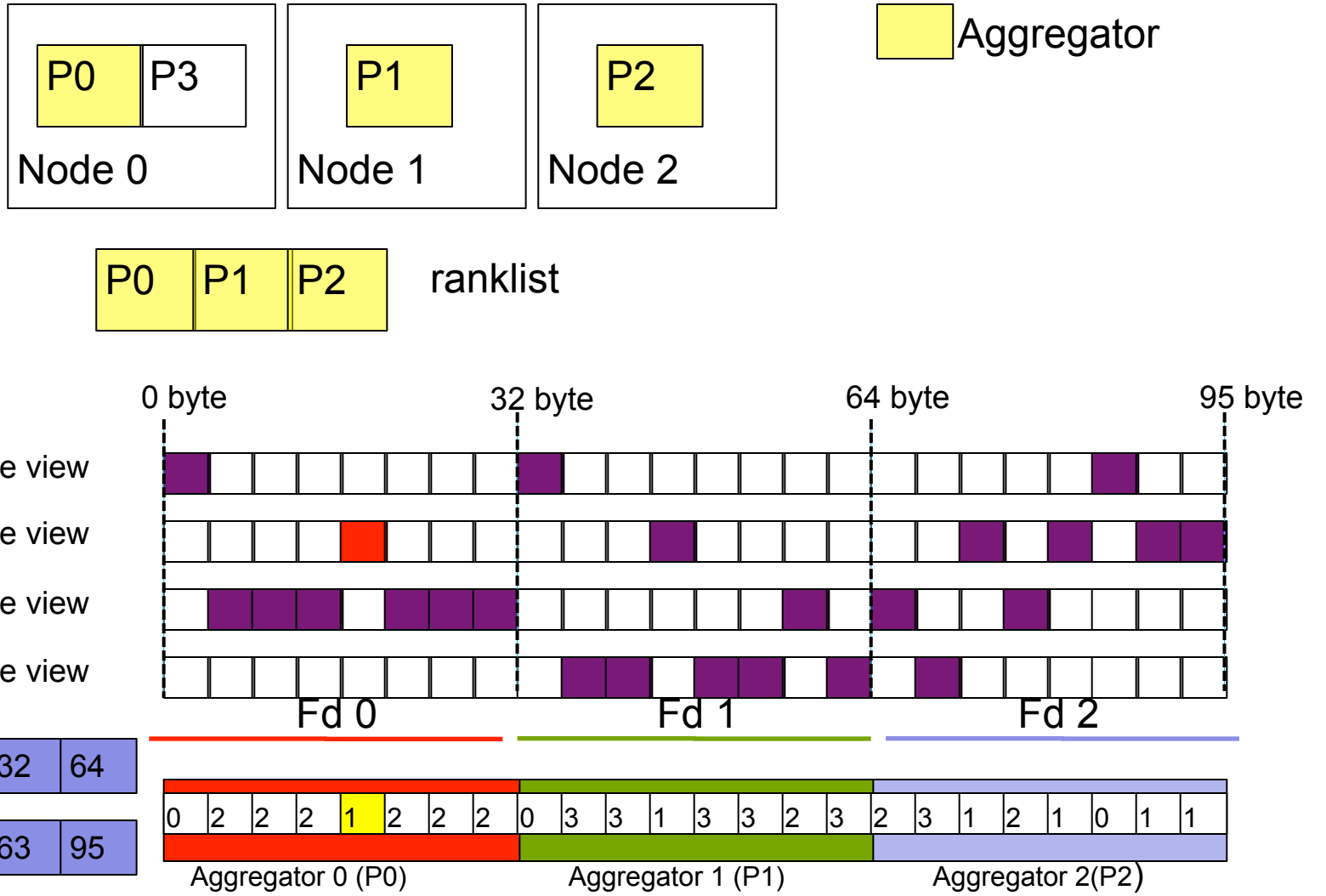
Example of Two-Phase I/O



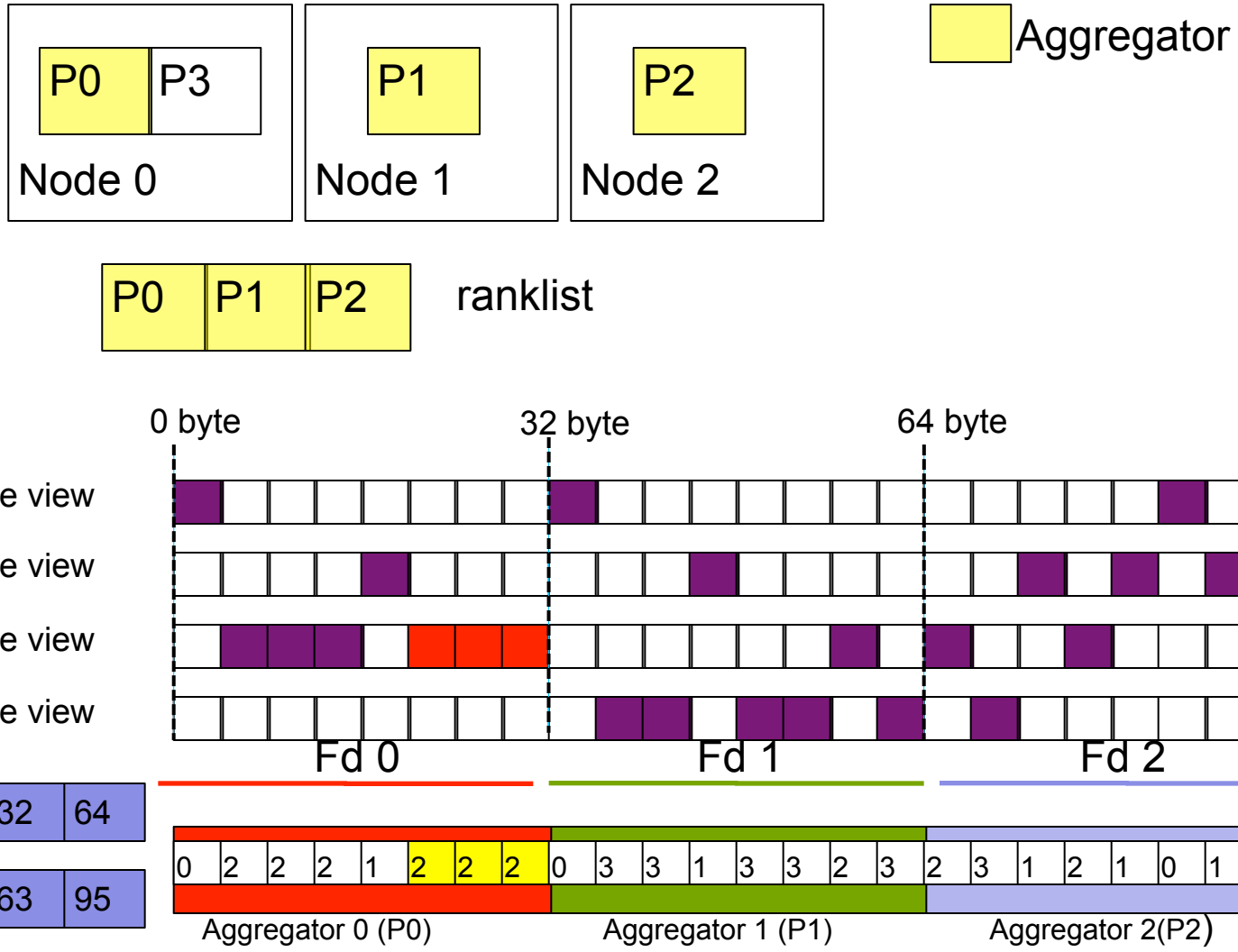
Example of Two-Phase I/O



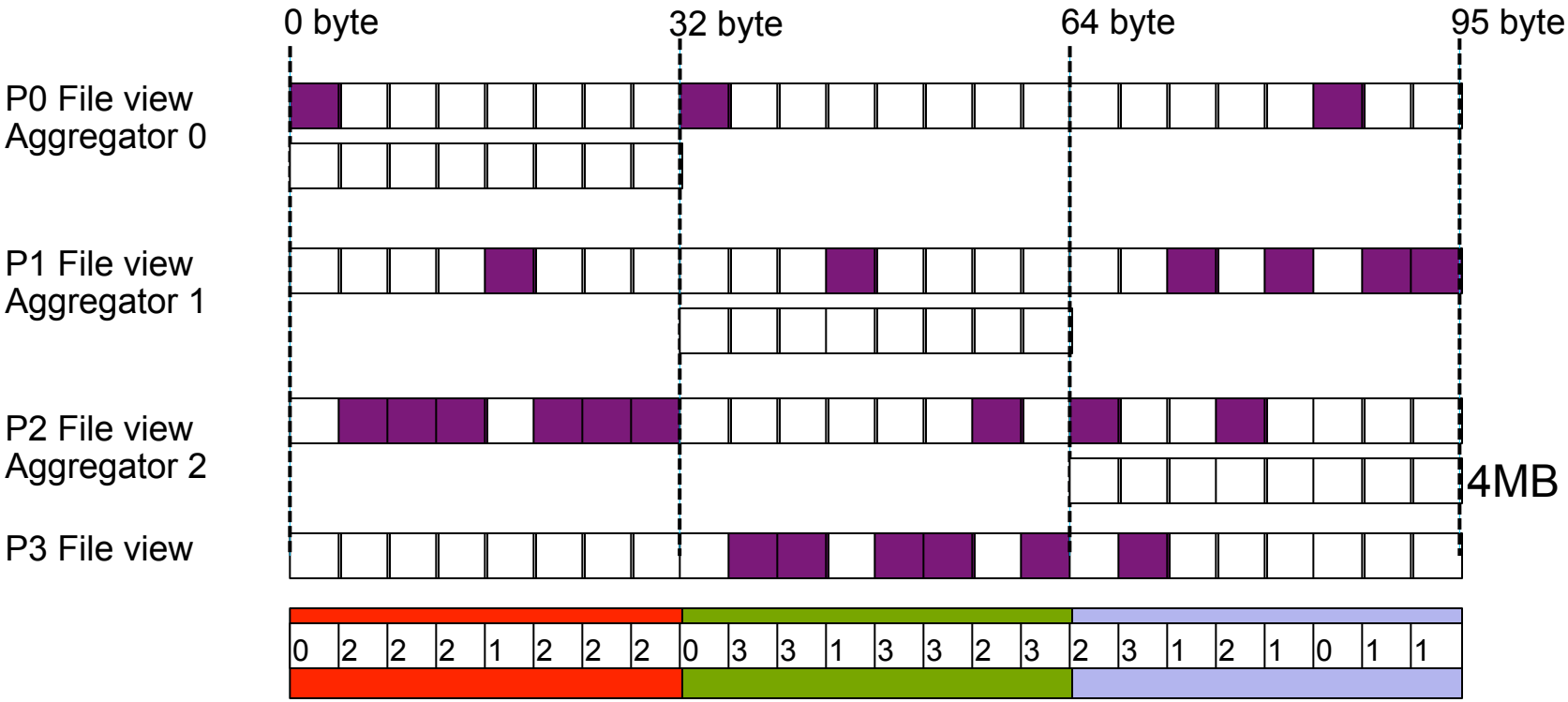
Example of Two-Phase I/O



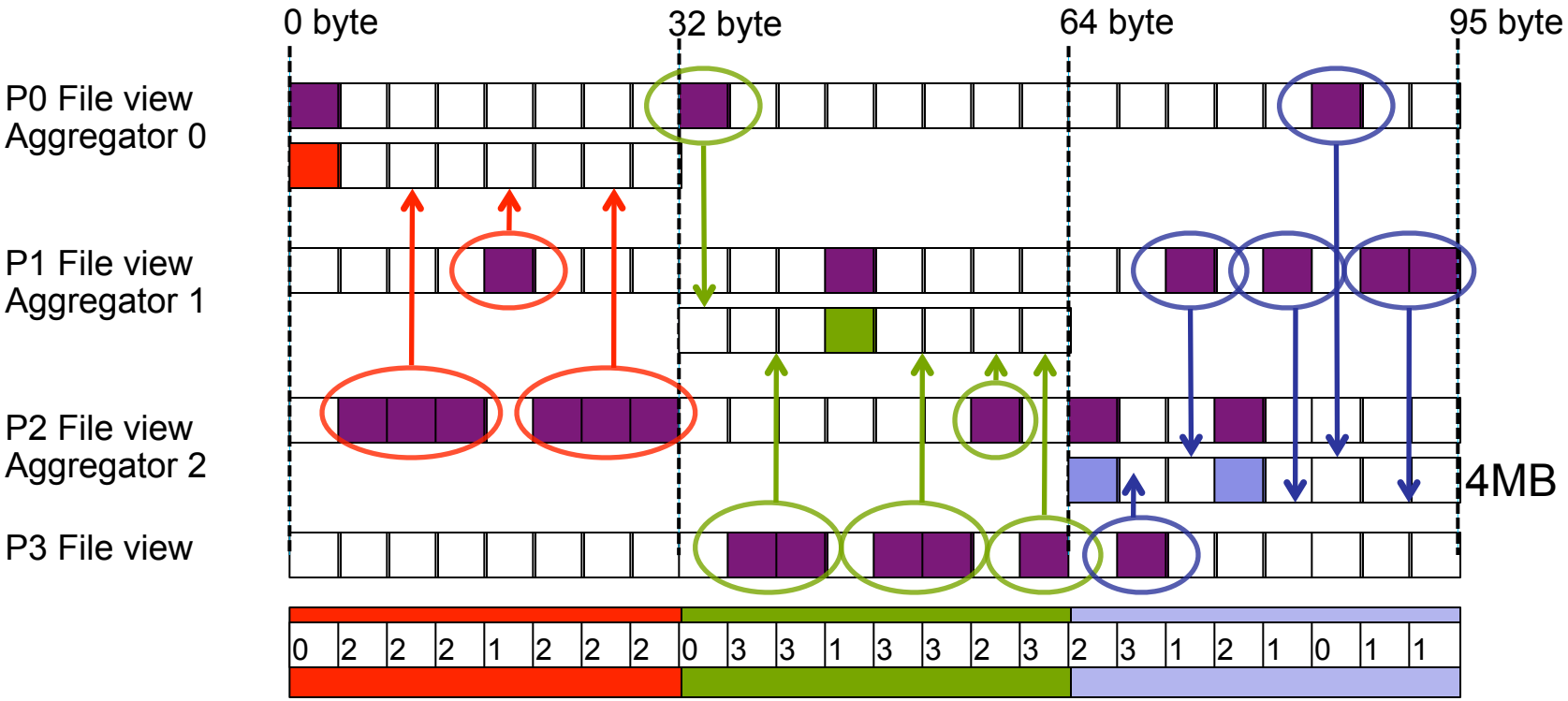
Example of Two-Phase I/O



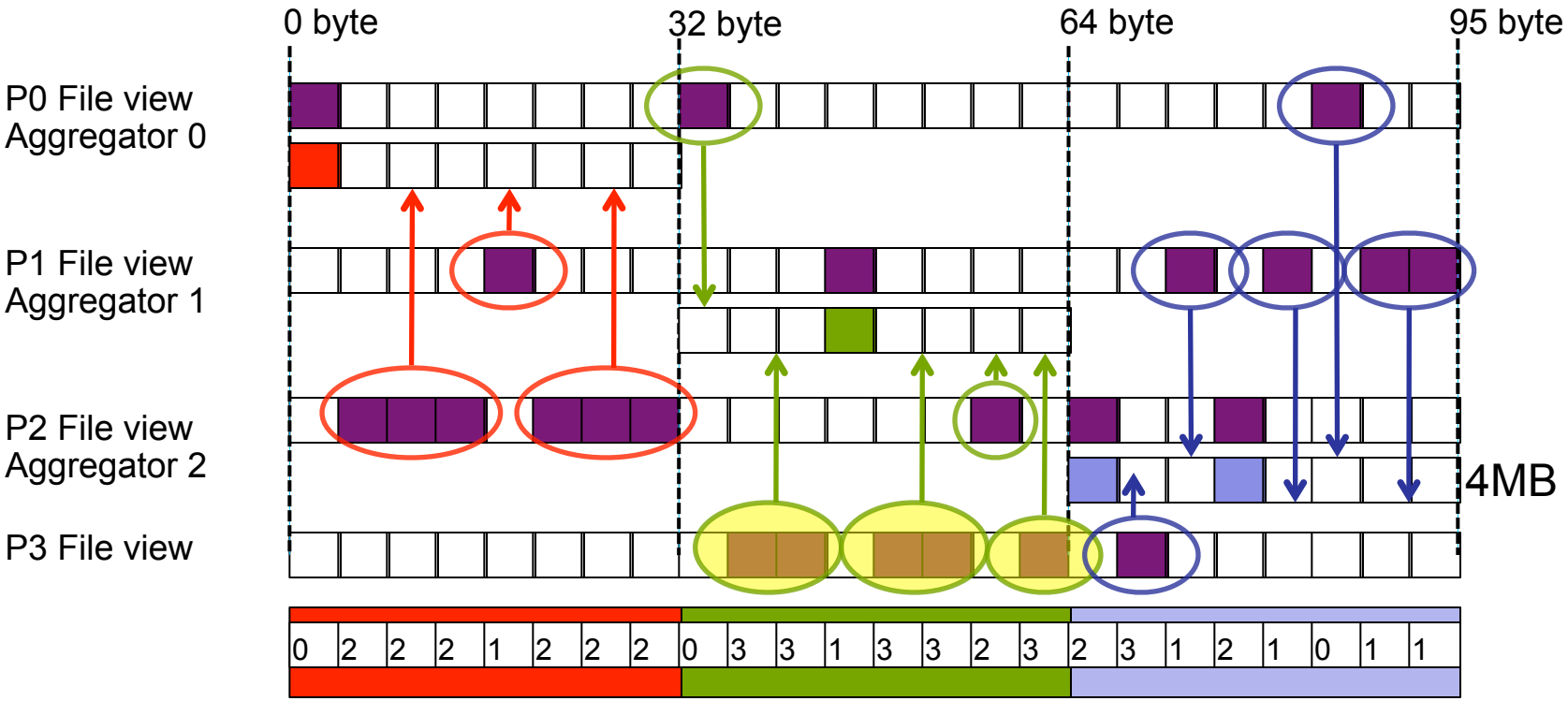
Example of Two-Phase I/O



Example of Two-Phase I/O

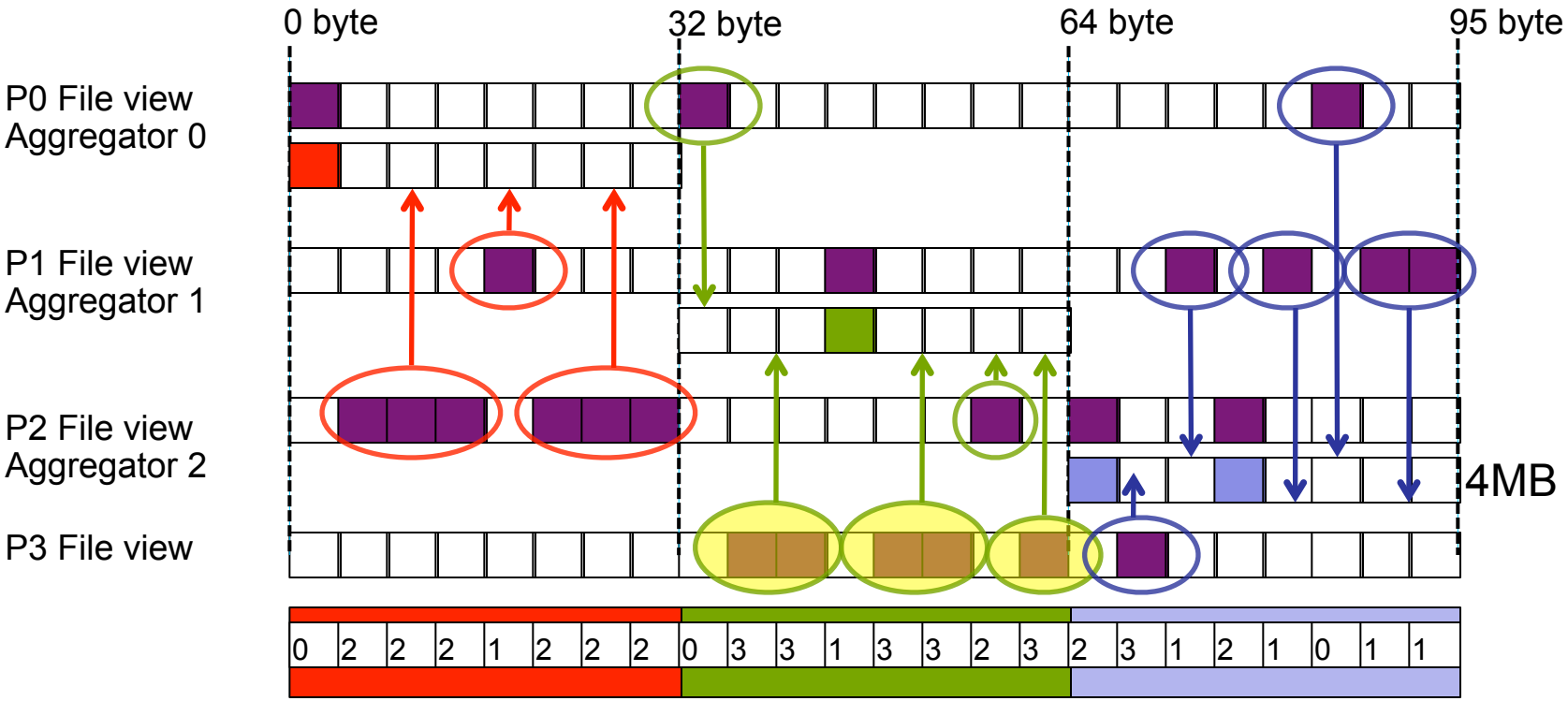


Example of Two-Phase I/O



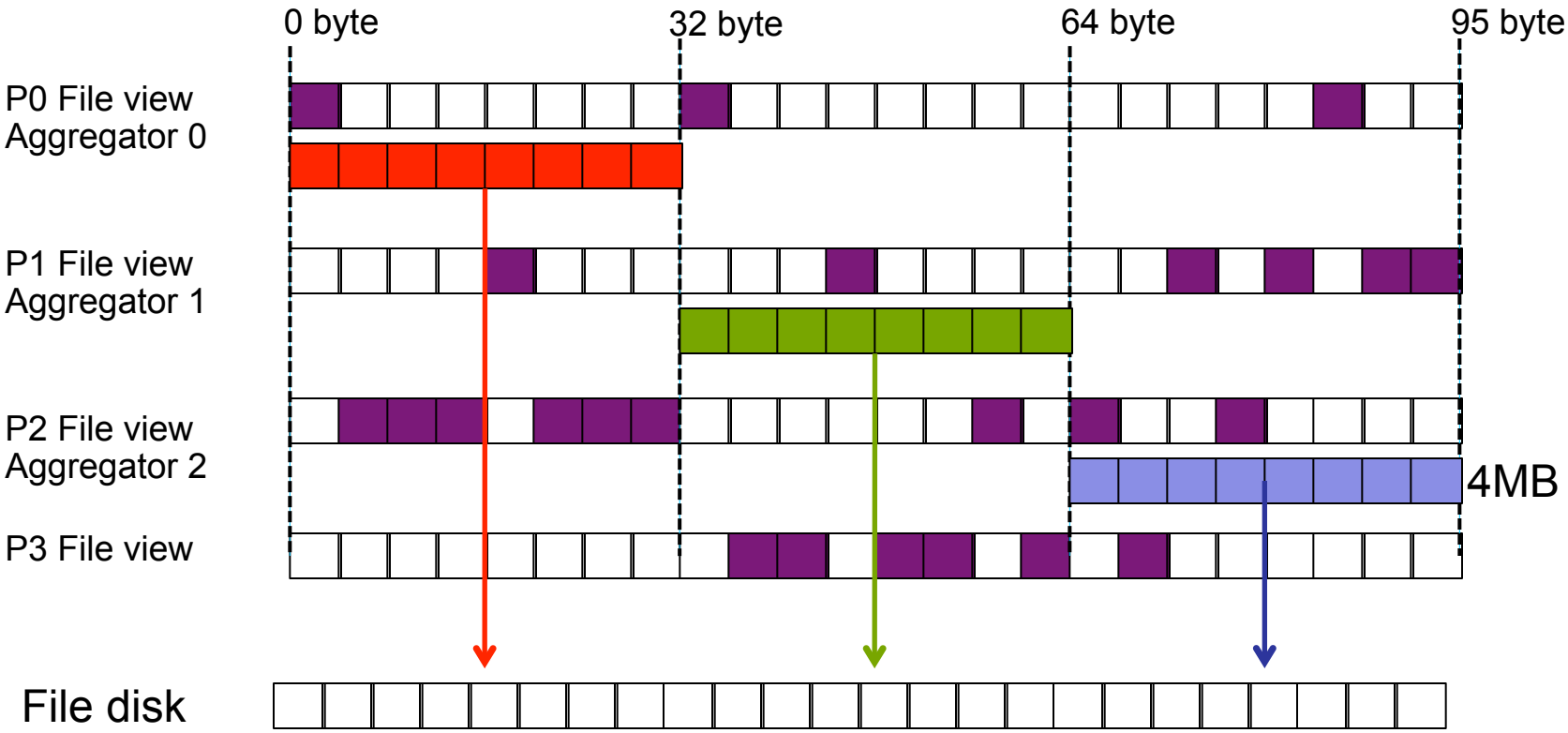
Communication: 13 message
76 bytes of 96 bytes are transferred among the processes.

Example of Two-Phase I/O

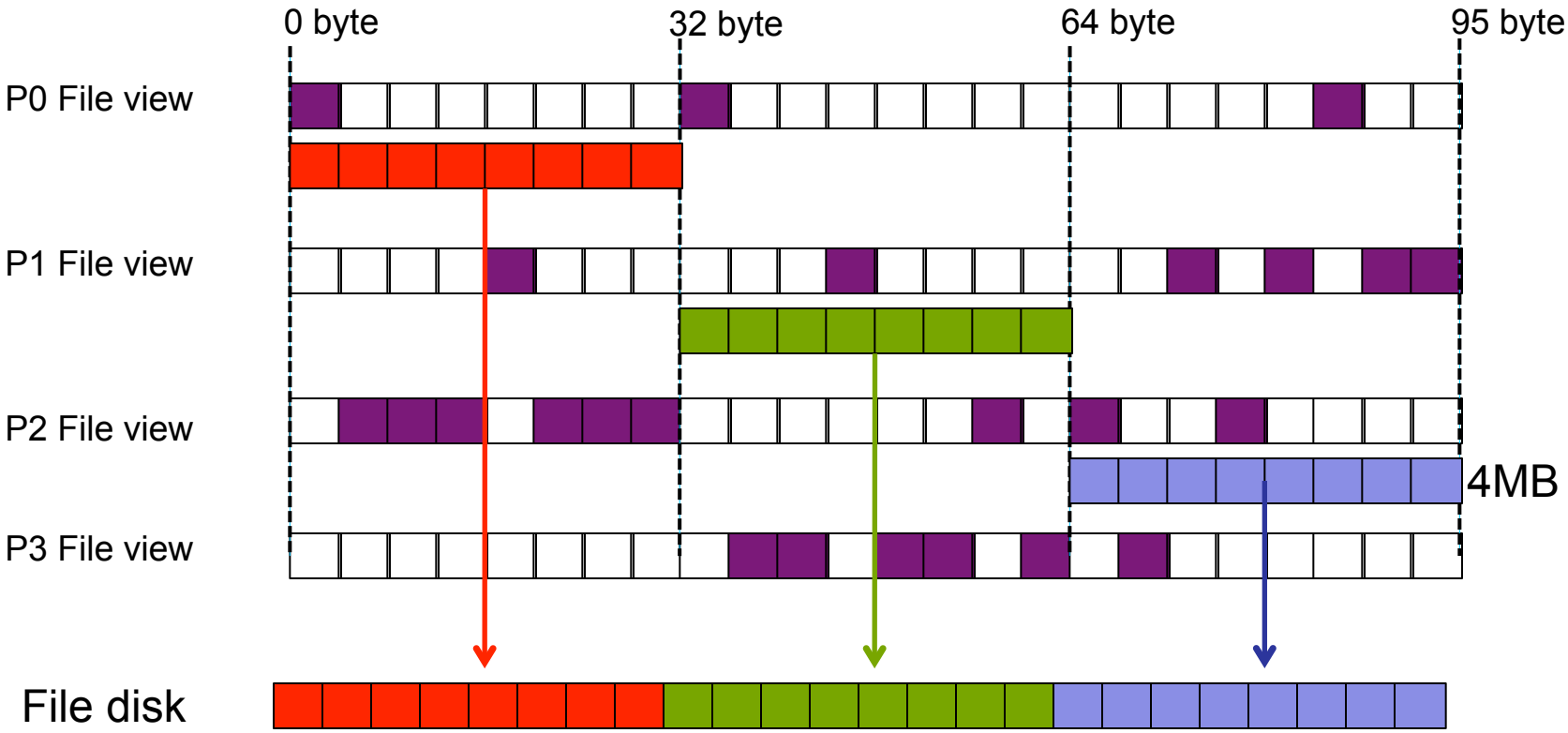


The number of communications could be reduced if each aggregator is assigned to the process more adequate.

Example of Two-Phase I/O



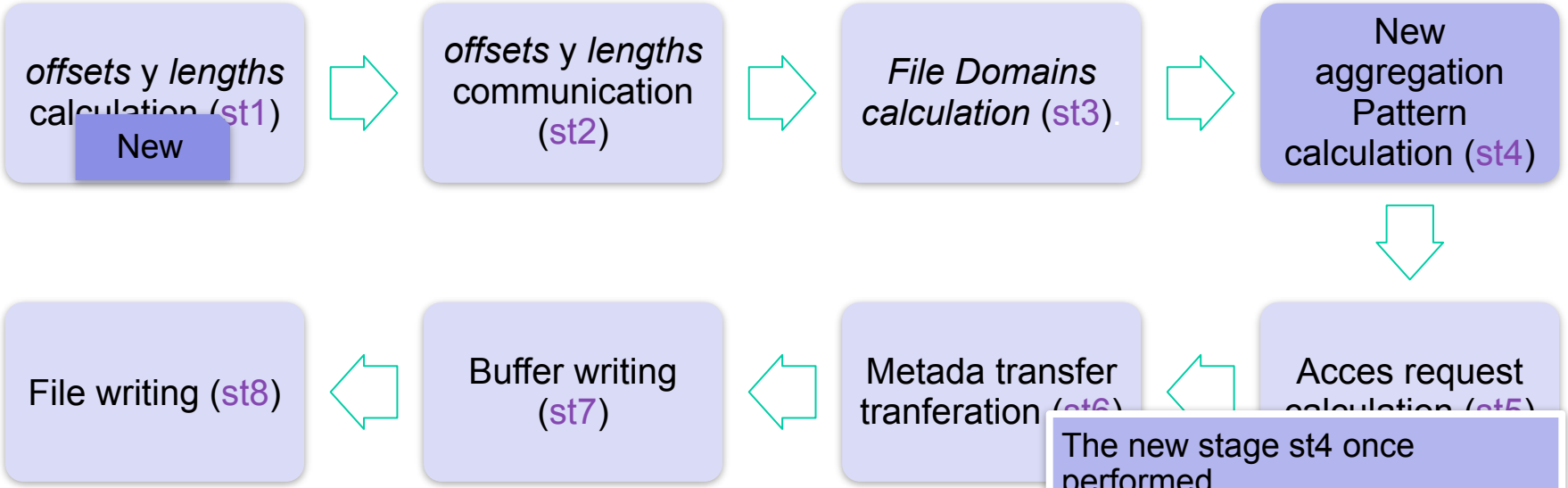
Example of Two-Phase I/O



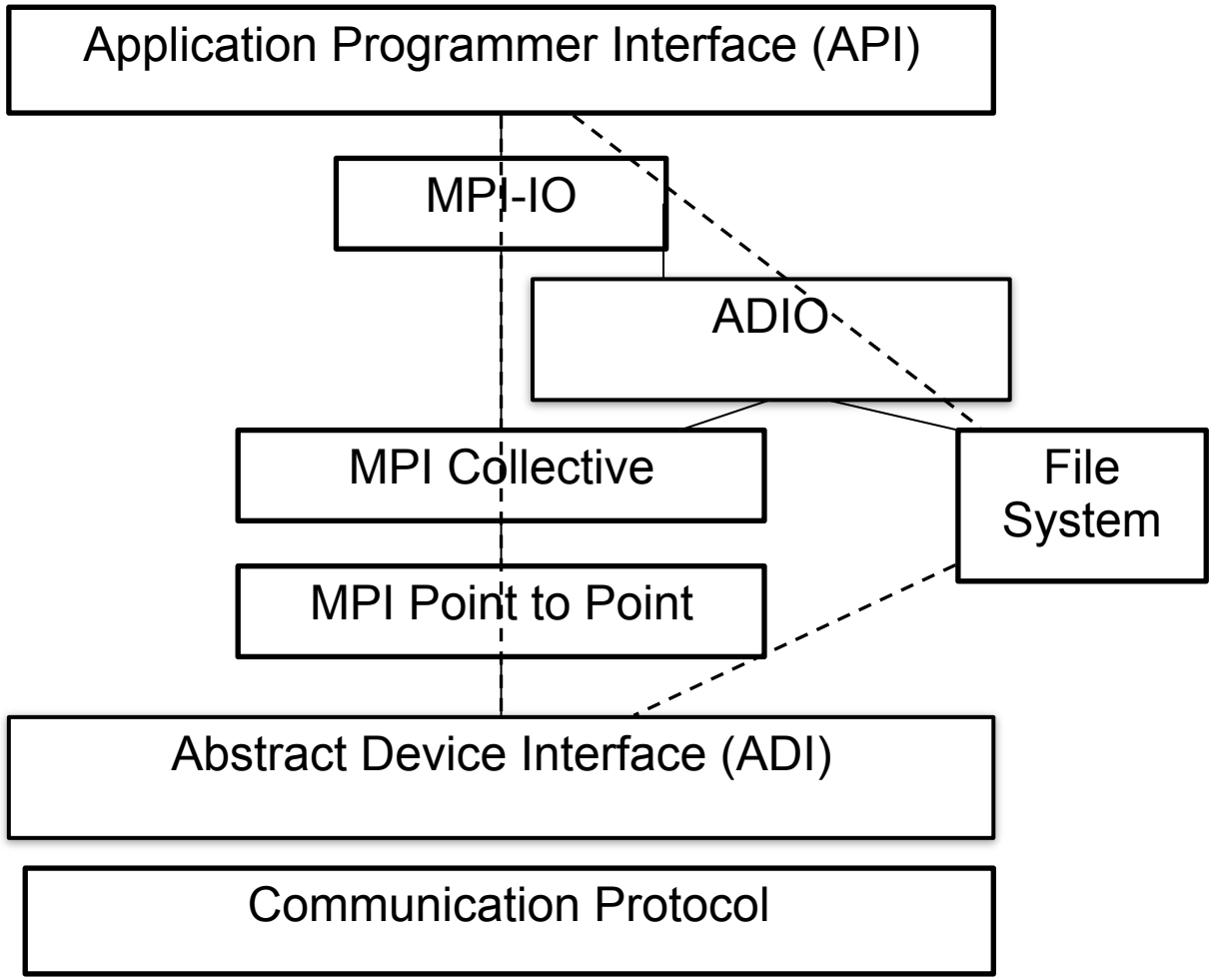
Phd. Thesis Proposal

- Replace the rigid assignment of aggregator by new one based on the next aggregation-criteria:
  - Reduce the number of communications:
    - Each aggregator is assigned to the process who has the highest number of contiguous data blocks.
    - Uses the initial distribution of data over the processes.
    - Employs the Linear Assignment Problem (LAP)
- The implementation of the new dynamic aggregator pattern → new version of Two_Phase I/O:
  - Locality_Aware_Two_Phase I/O (LA_TwoPhase I/O)

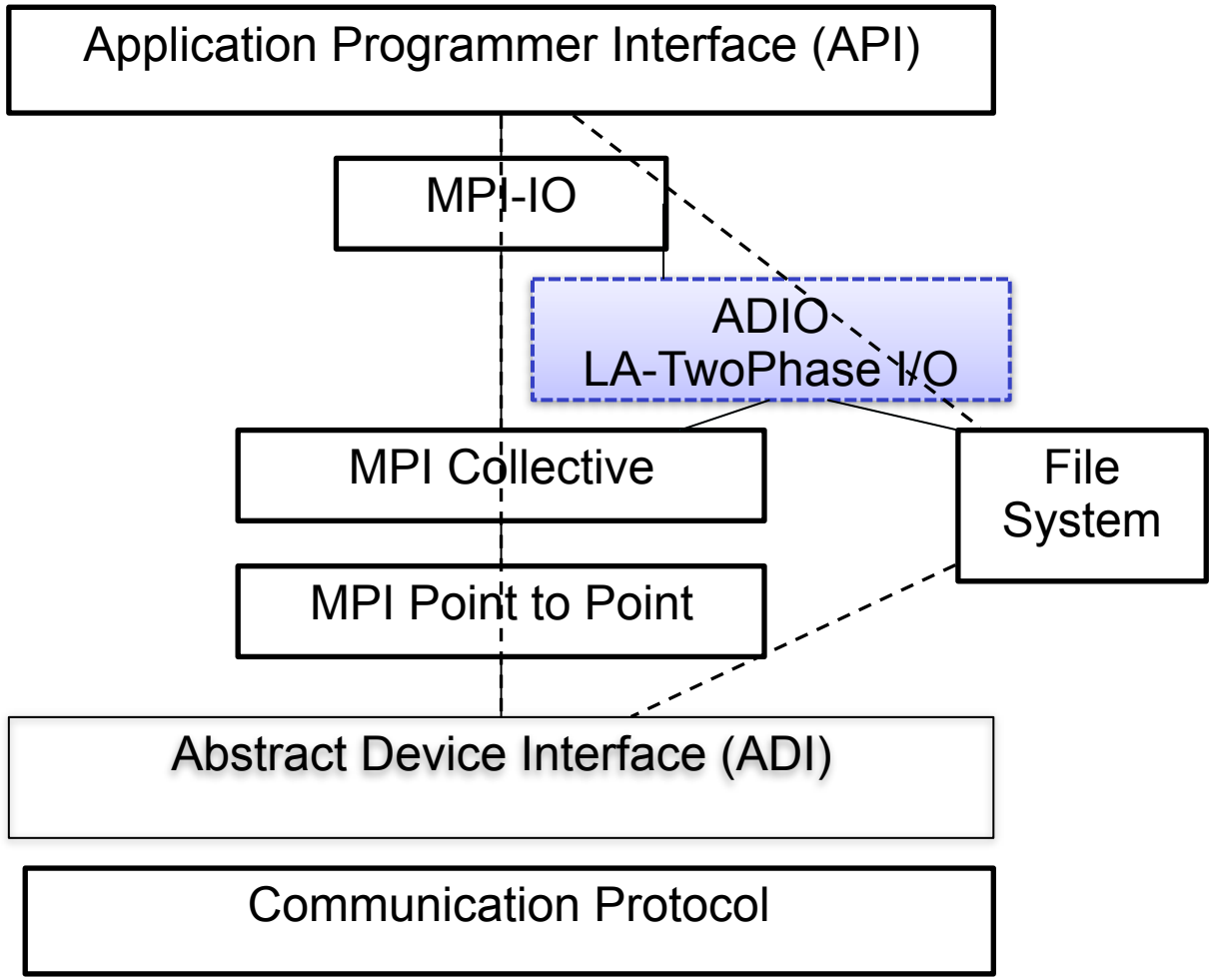
LA_Two_Phase I/O



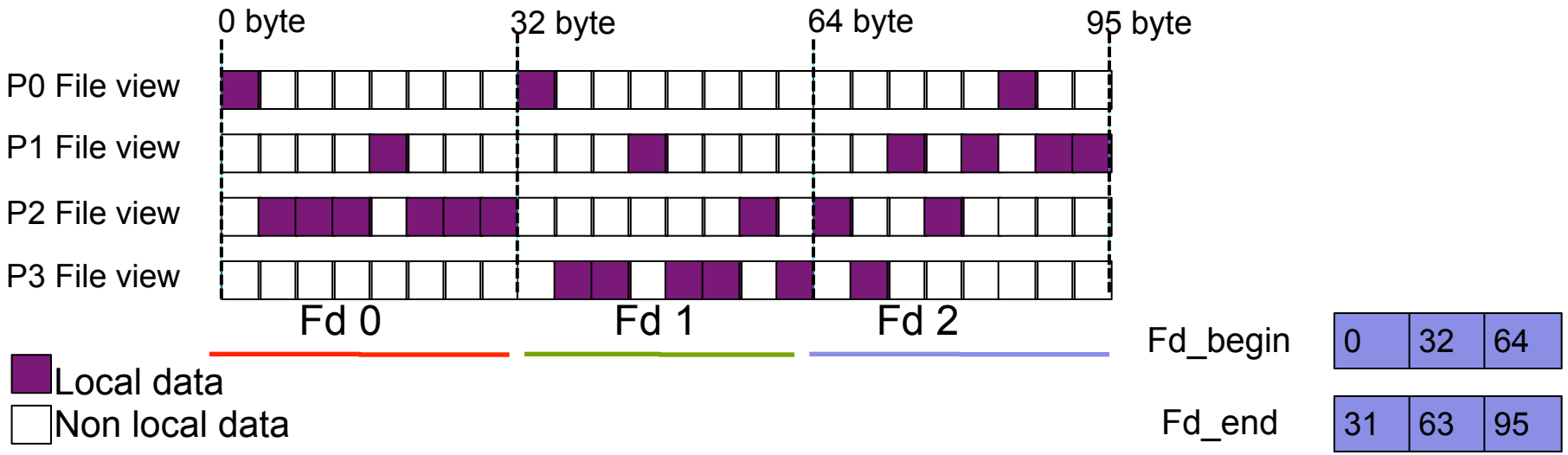
MPICH1.2 Architecture



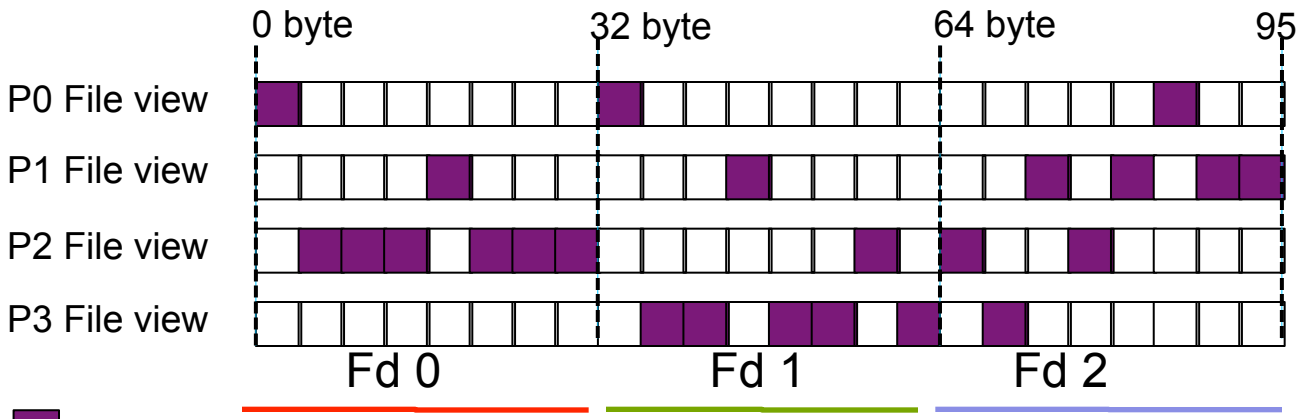
MPICH1.2 Modification



Example of LA_Two_Phase I/O



Calculation and recollection of distribution data

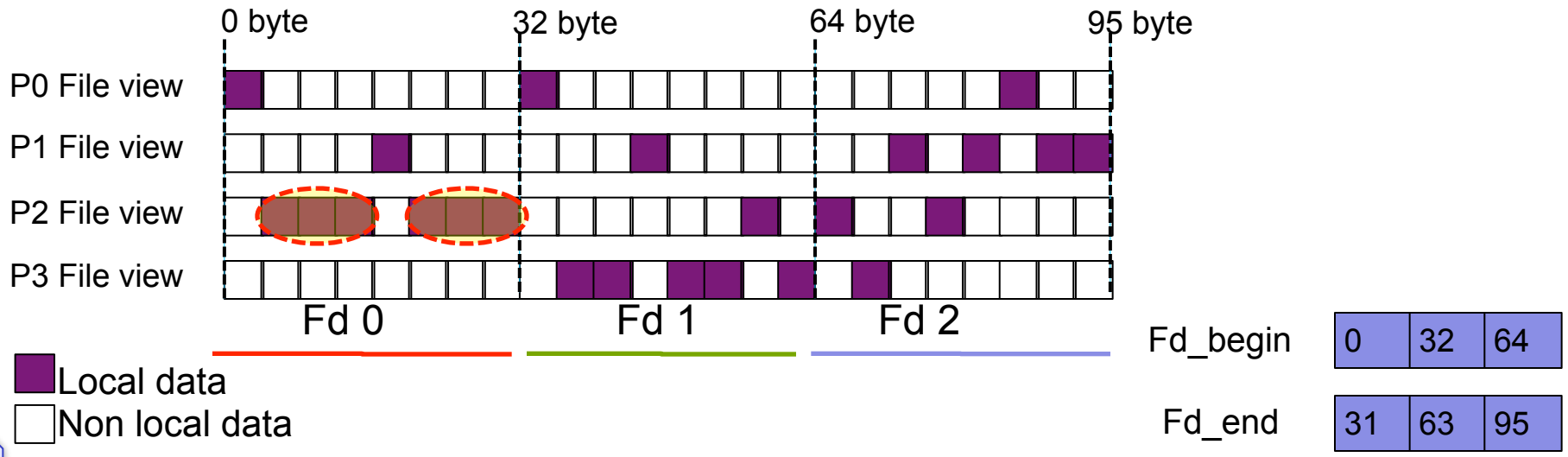


Fd_begin	0	32	64
Fd_end	31	63	95

Stage:
St4.1

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

Calculation and recolection of distribution data

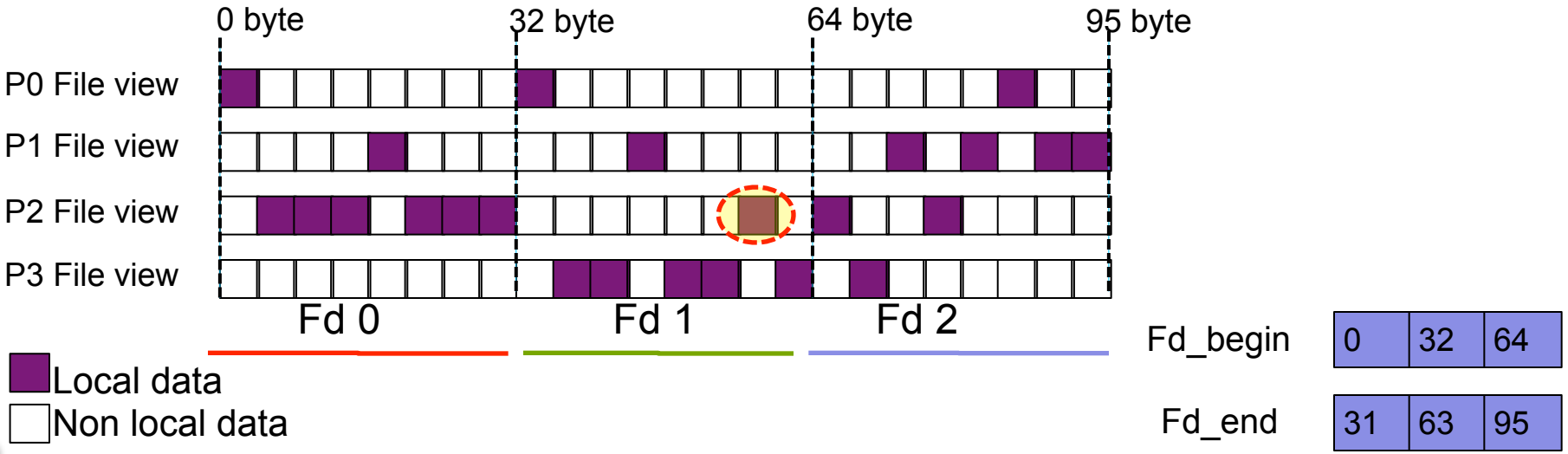


Fd_begin	0	32	64
Fd_end	31	63	95

Stage:
St4.1

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

Calculation and recolection of distribution data

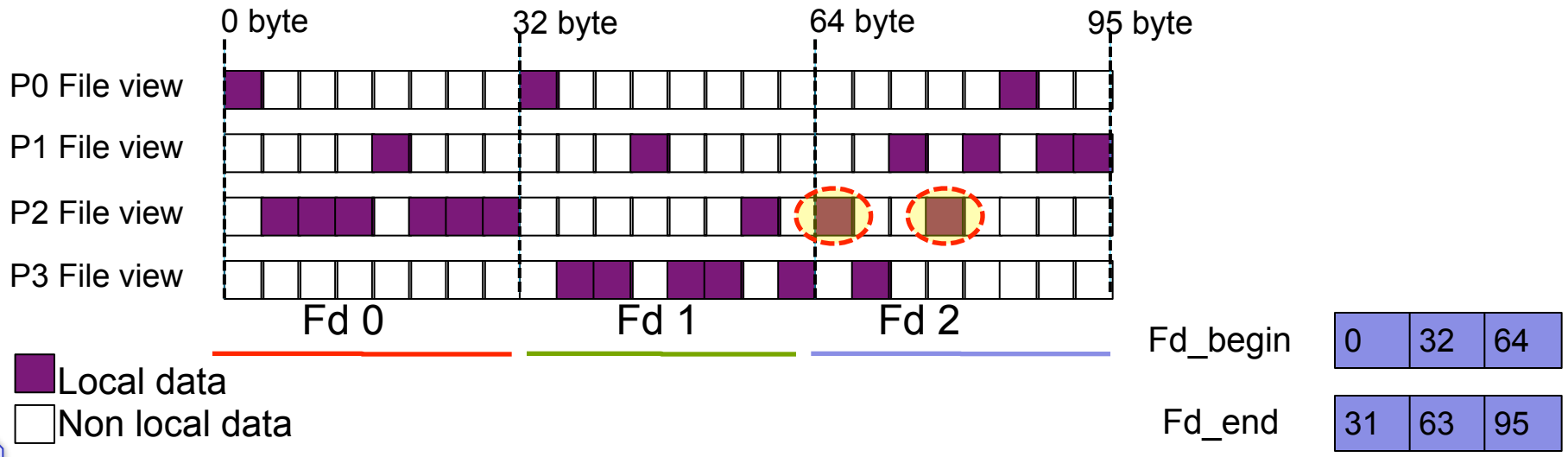


Fd_begin	0	32	64
Fd_end	31	63	95

Stage:
St4.1

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

Calculation and recolection of distribution data

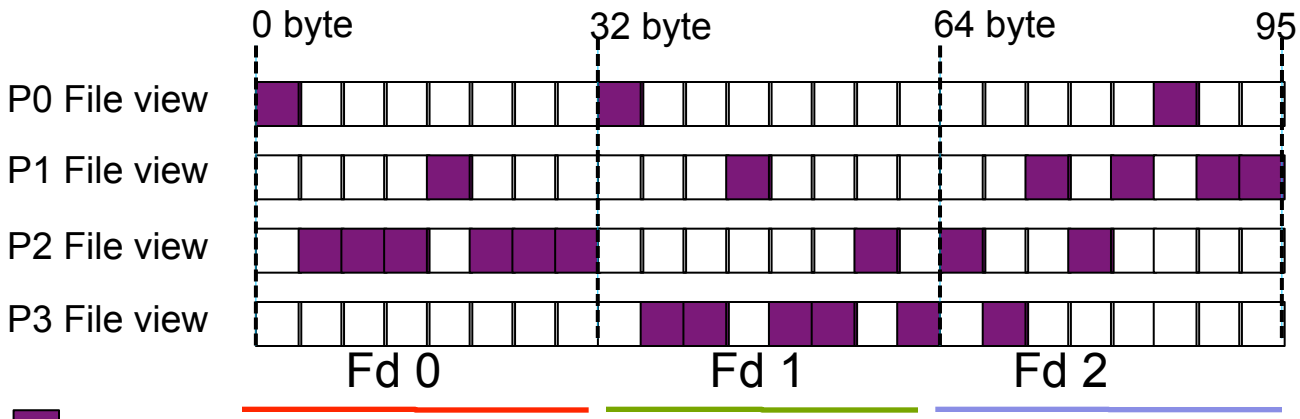


Fd_begin	0	32	64
Fd_end	31	63	95

Stage:
St4.1

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

Calculation and recolection of distribution data



Fd_begin	0	32	64
Fd_end	31	63	95

Stage :
St4.2

P0 ⇨ ADIOI_Calc_Aggregator
P1 ⇨ ADIOI_Calc_Aggregator
P2 ⇨ ADIOI_Calc_Aggregator
P3 ⇨ ADIOI_Calc_Aggregator

Array_data

1	1	1
1	1	3
2	1	2
0	3	1

P0 Assigment_matrix

	FD0	FD1	FD2
P0	1	1	1
P1	1	1	3
P2	2	1	2
P3	0	3	1

Calculation of new aggregator pattern

P0

LAP

Assigment_matrix

	FD0	FD1	FD2
P0	1	1	1
P1	1	1	3
P2	2	1	2
P3	0	3	1

Aj=Max(Pi,FD)

Stage:
St4.3

Calculation of new aggregator pattern

P0

LAP

Assigment_matrix

	FD0	FD1	FD2
P0	1	1	1
P1	1	1	3
P2	2	1	2
P3	0	3	1

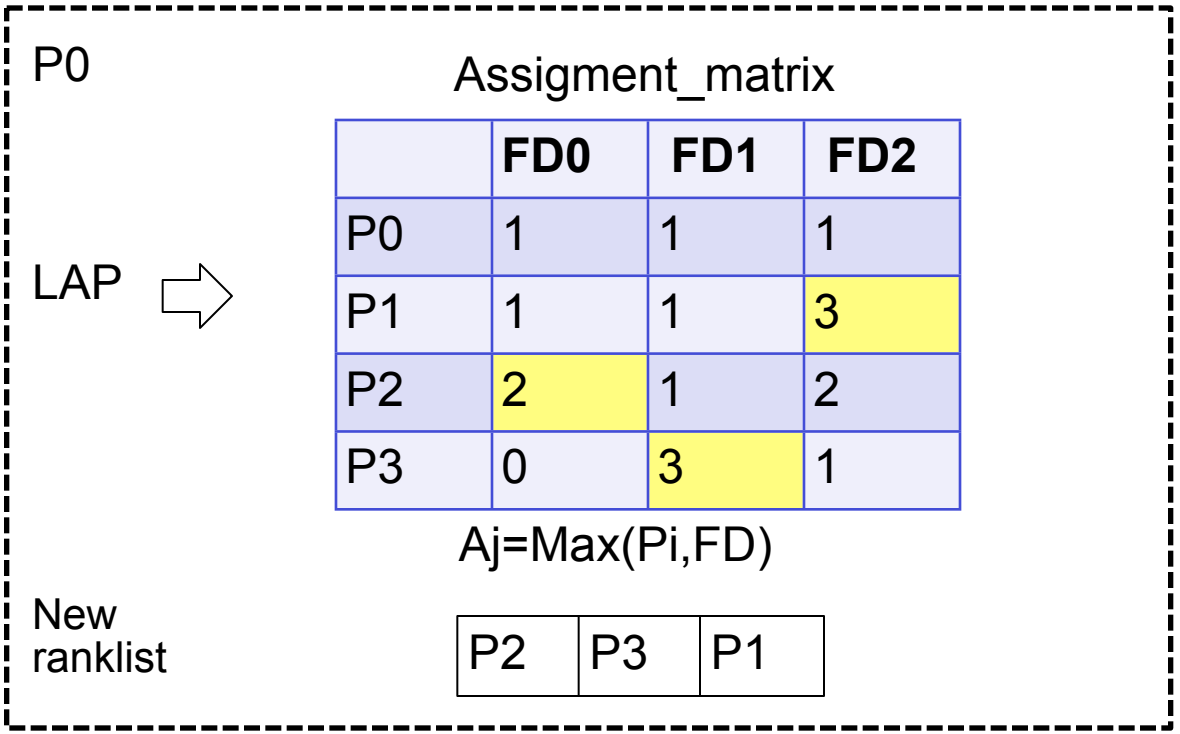
Aj=Max(Pi,FD)

New ranklist

P2	P3	P1

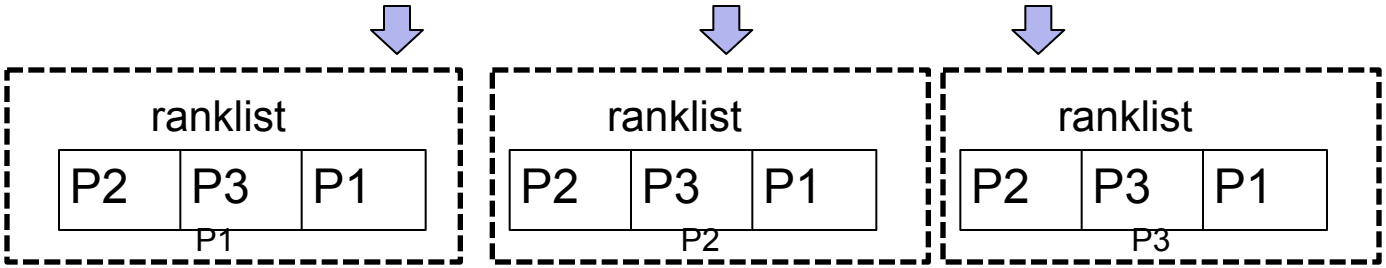
Stage:
St4.3

Distribution of new aggregator pattern

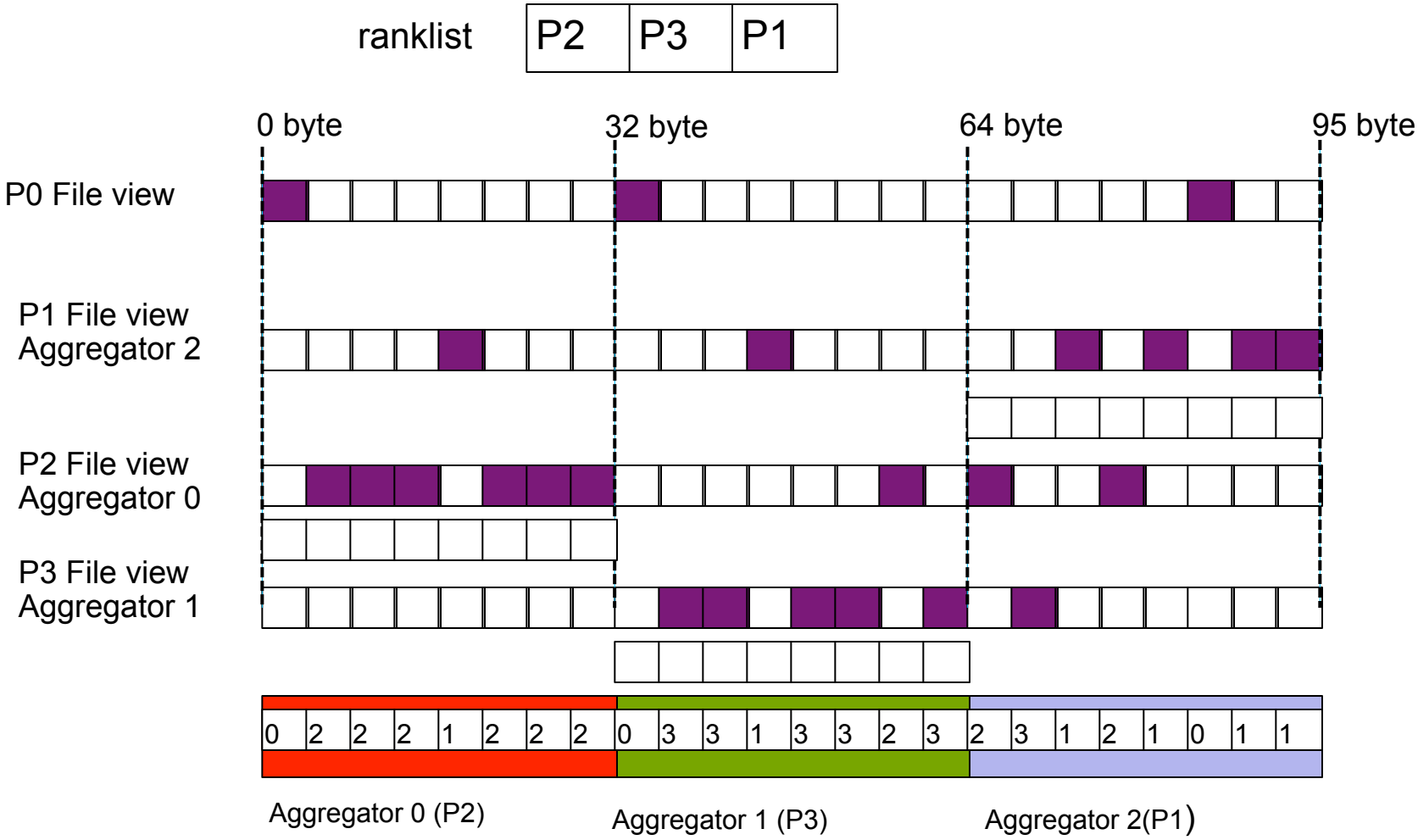


Stage: St4.3

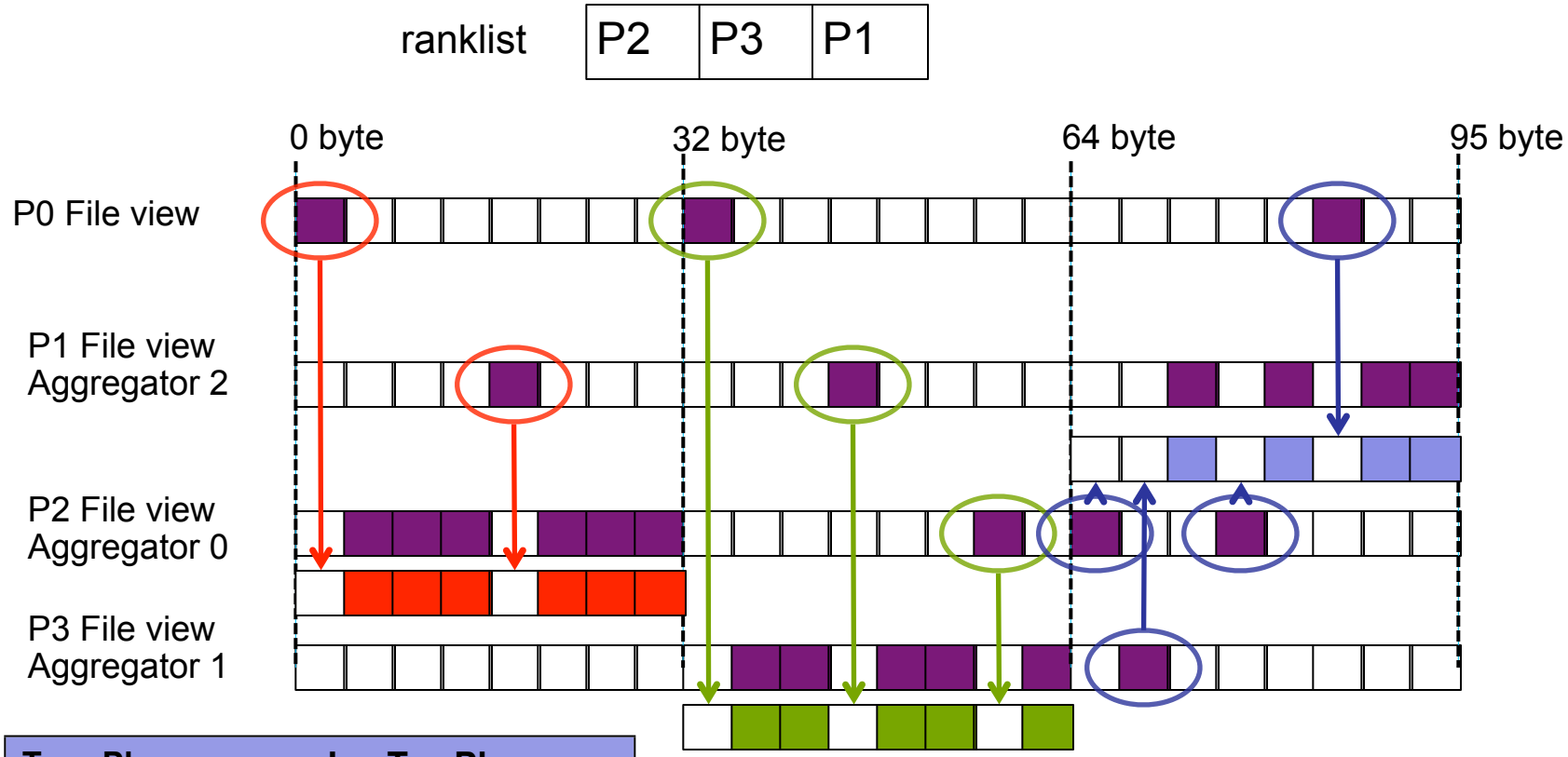
ADIOI_cb_bcast_rank_map



Reduction in the number of communication with the new pattern.

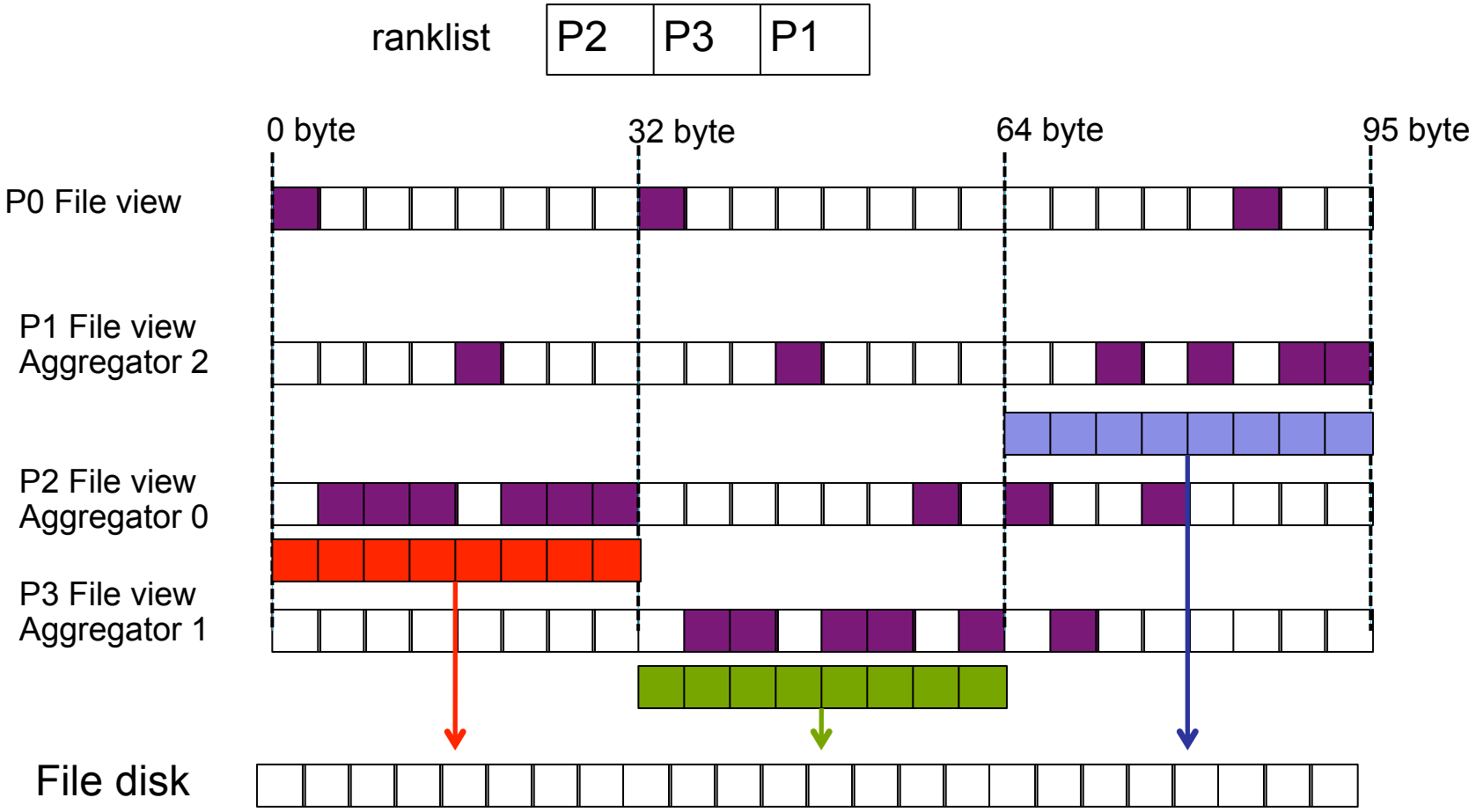


Reduction in the number of communication with the new pattern.

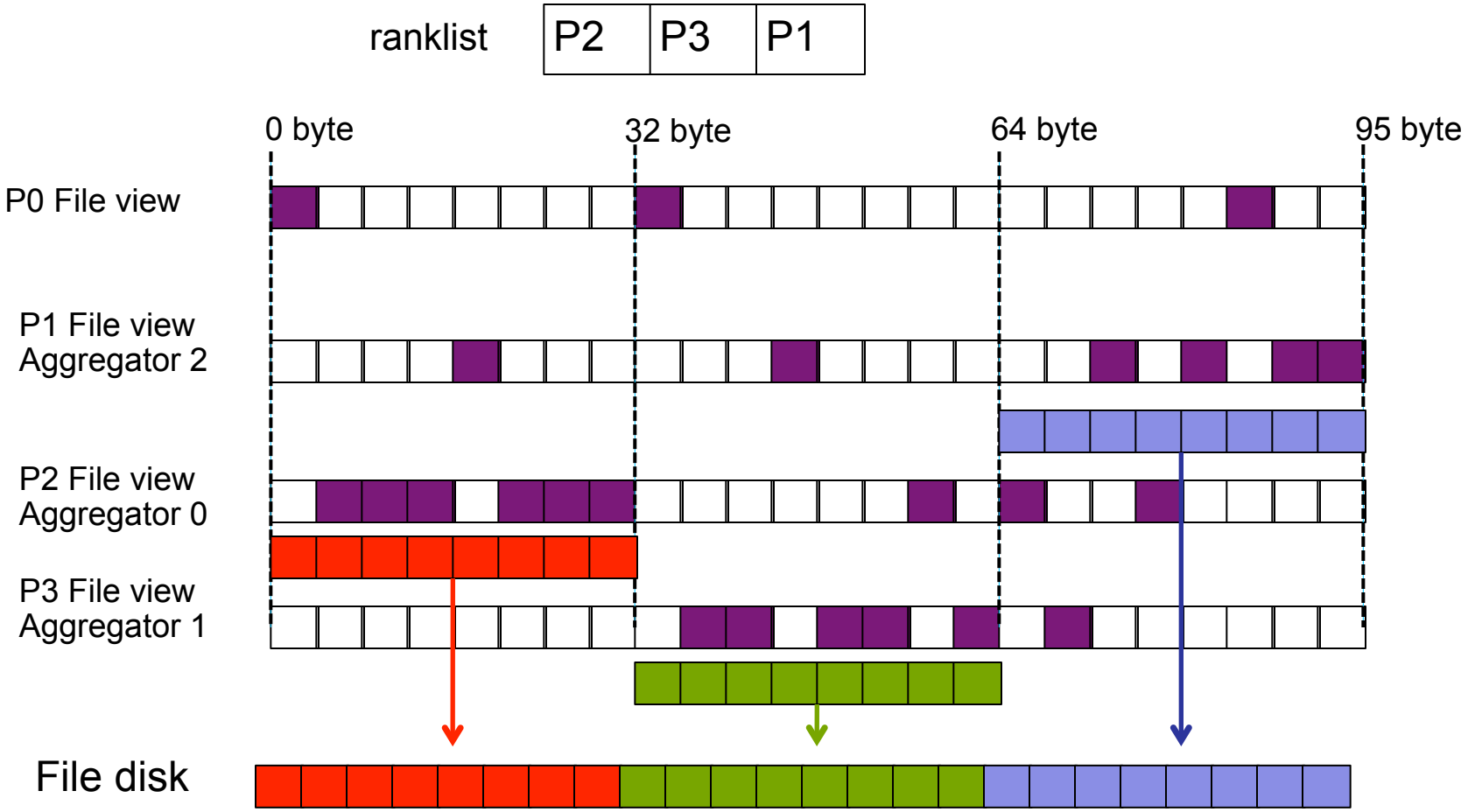


Two_Phase	La_TwoPhase
13 messages	9 messages
76 bytes	36 bytes

Collective write

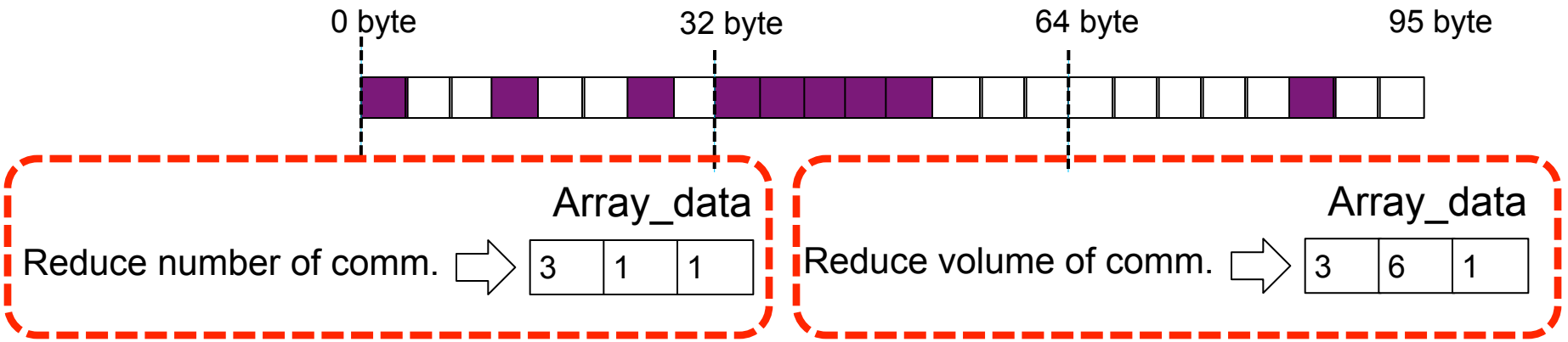


Collective write



New Proposal: Dynamic and adaptive I/O aggregator pattern

- We propose to use two aggregation-criteria:
  - Reduce the number of communications:
    - Each aggregator to the node who has more highest number of contiguous data blocks.
  - Reduce the volume the communications:
    - Each aggregator to the node who has more data of file domain associated with the aggregator.



New Proposal: Dynamic and adaptive I/O aggregator pattern

- Is dynamic because is calculated at runtime
- Is adaptive because could select the aggregation-criterion that the reduce more the communication phase.
- Before: each aggregator was assigned according local data that each cores stores.
- Now: MPICH2 → according local data of each node stores.

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. Evaluation

Strategies for improve the performance of I/O and communication operations.

Scientific applications need:
- Large number of computer nodes
- Huge volume of data transferred among the processes

Program model using in cluster → MPI
- Communication among the processes by messages

Communication system has become one of the major limiting factors
- Reduction the performance

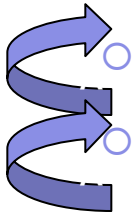
Strategy to reduce the cost of communication in MPI by using lossless compression techniques.

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 (RAS)
  - Guided Strategy (GS)

Phd. Thesis Proposal: Communication Compression

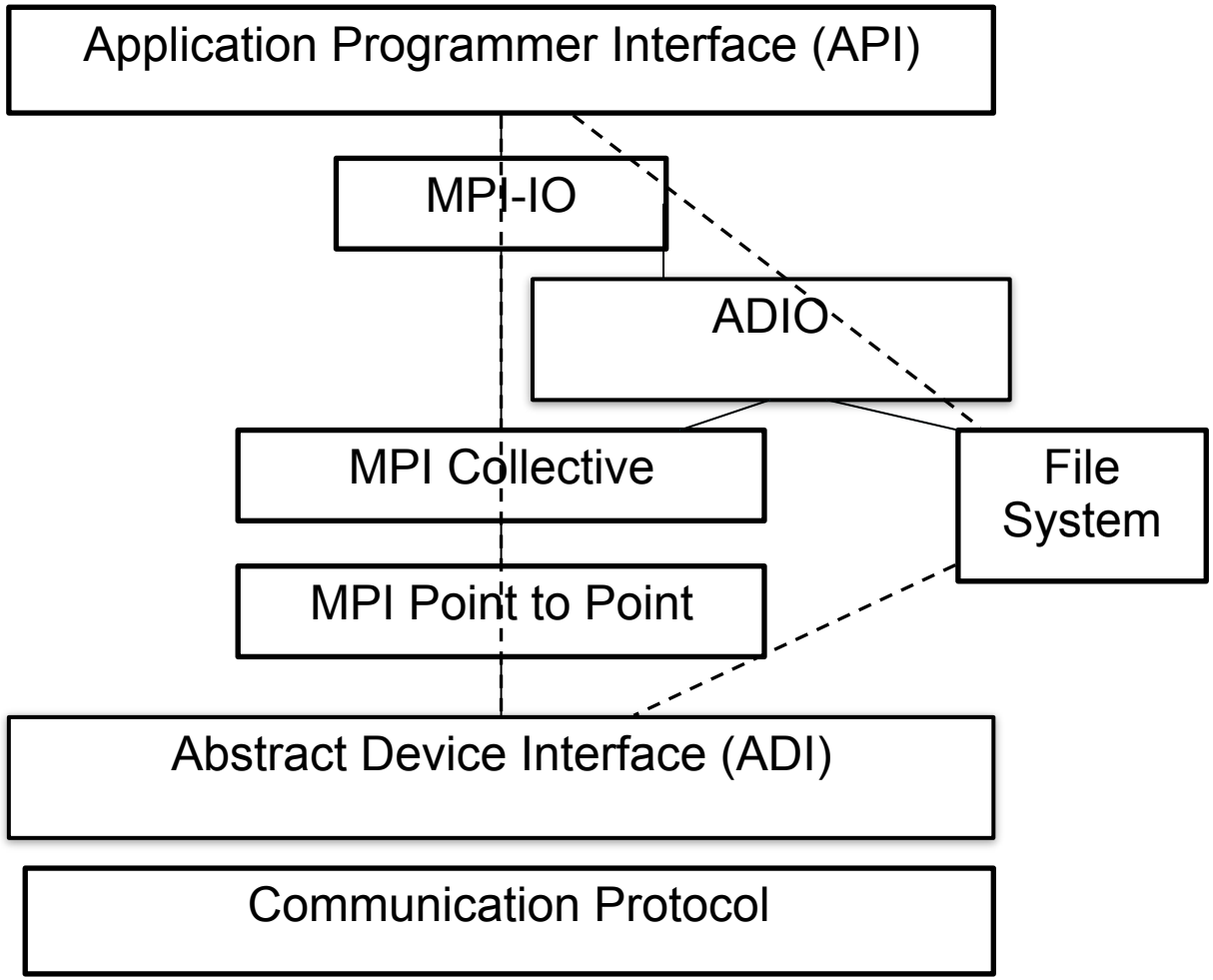
- Reduce the cost of communications:
  - By MPI messages compression in run-time
- Lossless compressions algorithms
- Compress all MPI primitives.
- We have developed three different strategies:
  - Runtime Compression (RC)
  - Runtime Adaptive Compression (RAS)
  - Guided Strategy (GS)



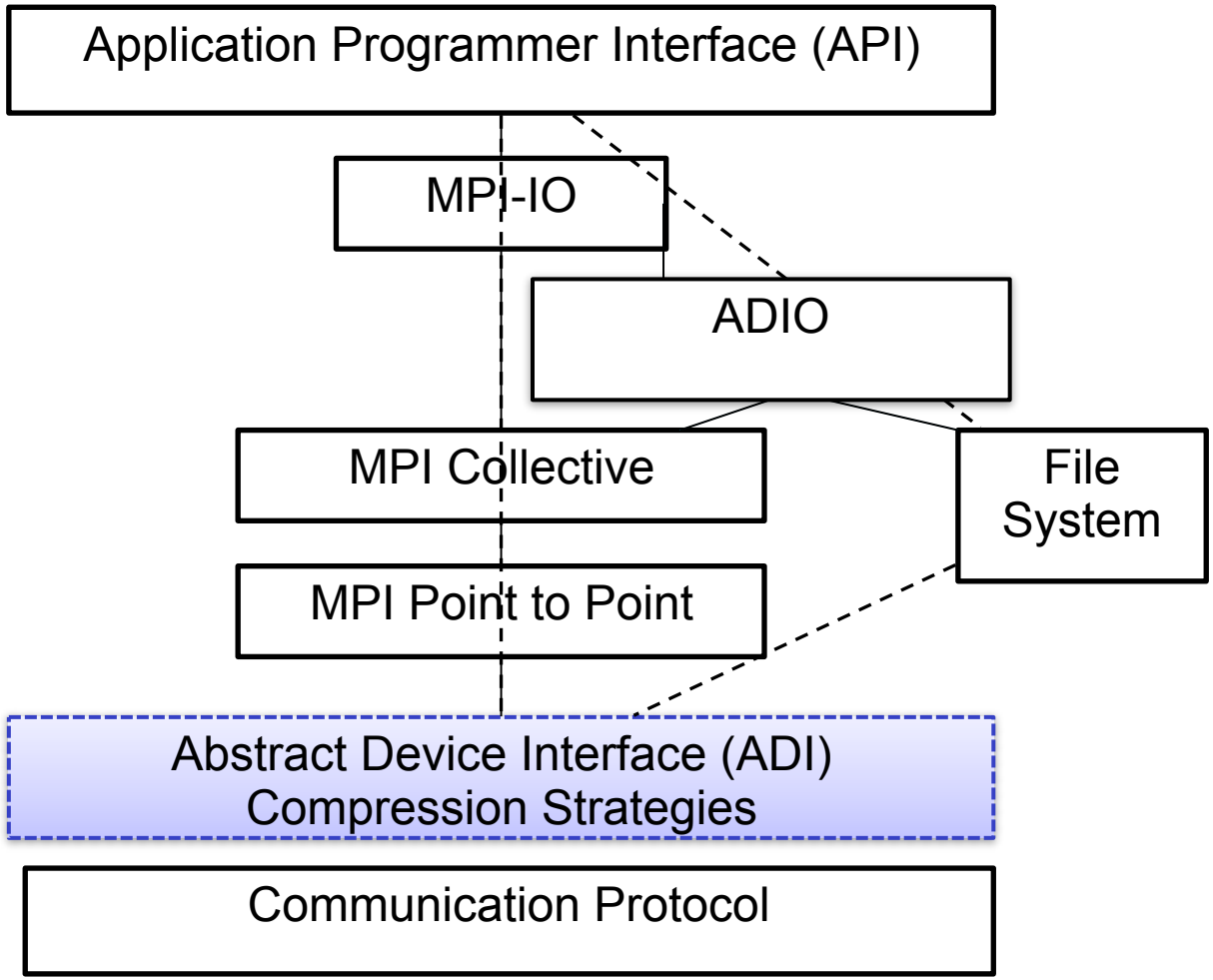
1 Compression Strategy: Runtime Compression

- Runtime Compression (RC) uses different compression algorithms:
  - LZO, RLE, HUFFMAN, RICE, FPC.
- Compress all the transferred message (> 2KB) with the algorithm indicated by the user (MPI_hint)
- Compression not can be disable
- Compress all MPI primitivies:
  - Messages: point-to-point and collectives.
  - Communications: blocking and no blocking
  - Datatypes: contiguous y non-contiguous

MPICH1.2 Architecture



MPICH1.2 Architecture



Modification over ADI layer

- Header in the exchanged message to inform:
  - Compression used or not, algorithm and length
  - All compression algorithm are included in a single Compression Library:
    - We can add new compression algorithms easily.

Compression stages
- Message size

Decompression stages
- Header

2 Compression Strategy: Runtime Adaptive Compression

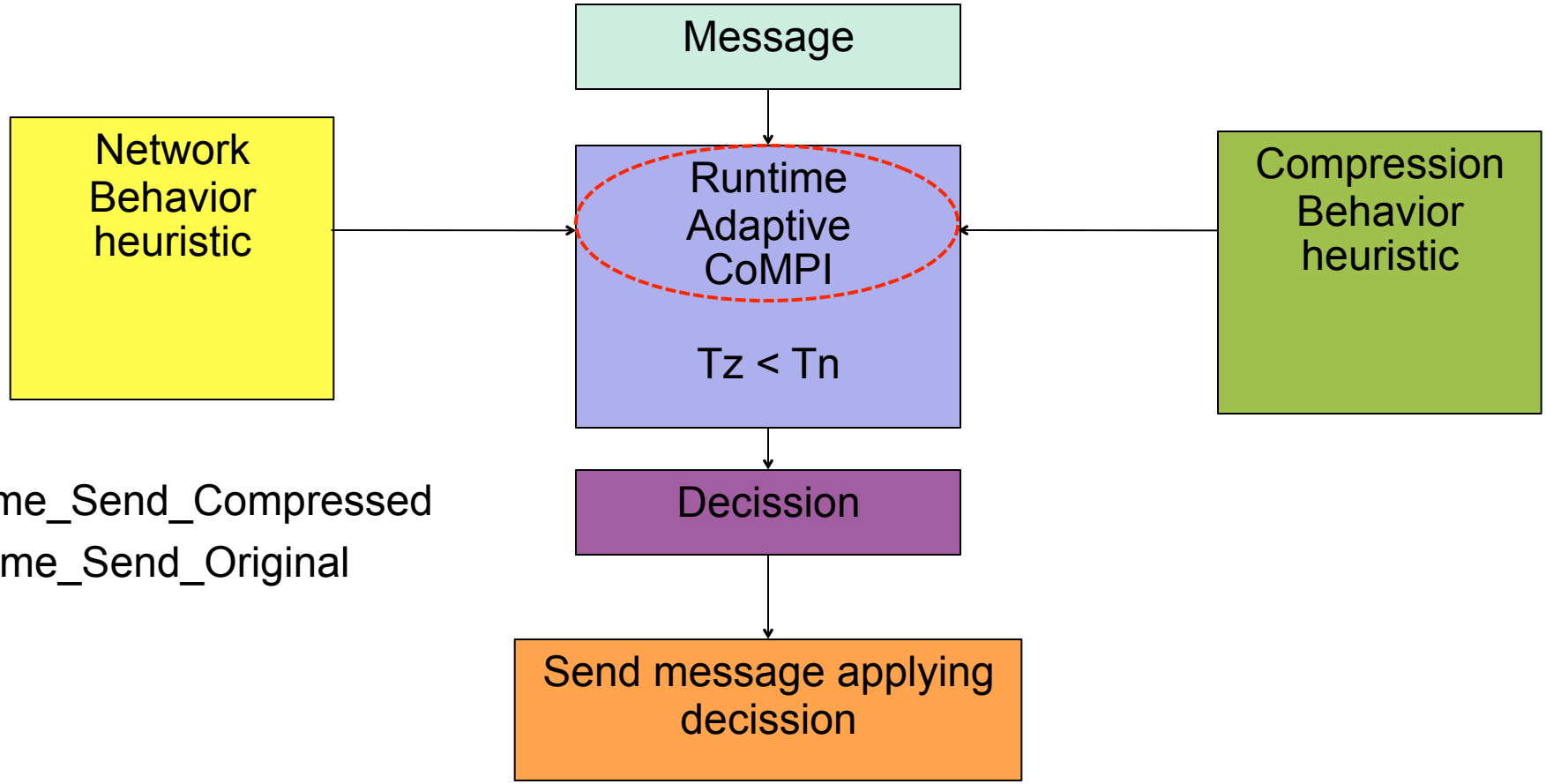
- Runtime Adaptive Compression Strategy (RAS), per message transferred takes two decision:
  - Turn on and off the compression.
  - Select itself the best compression algorithm.
- Learn in run-time from previous messages
- Decision depending on:
  - Message feature:
    - Datatype and length
  - Network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

Decisions → Speedup (1/2)

- 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

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

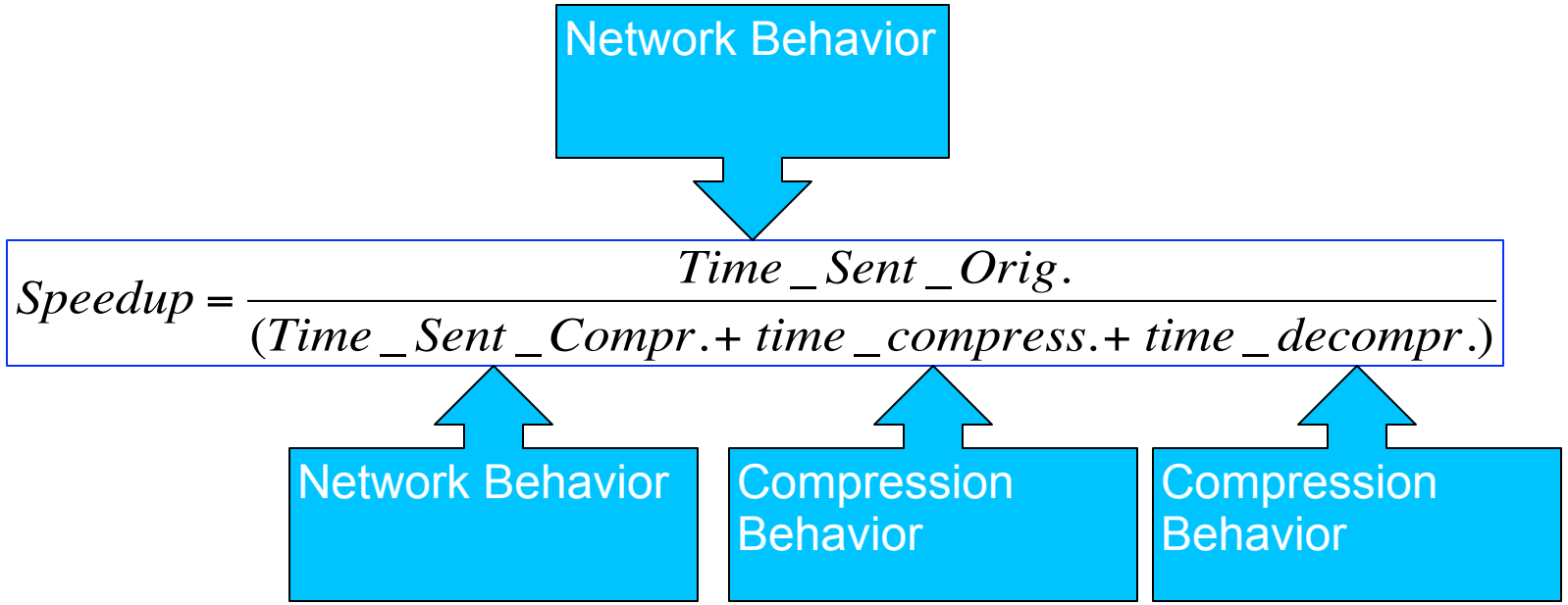
Decisions → Speedup



Tz=Time_Send_Compressed

Tn=Time_Send_Original

Compression behavior

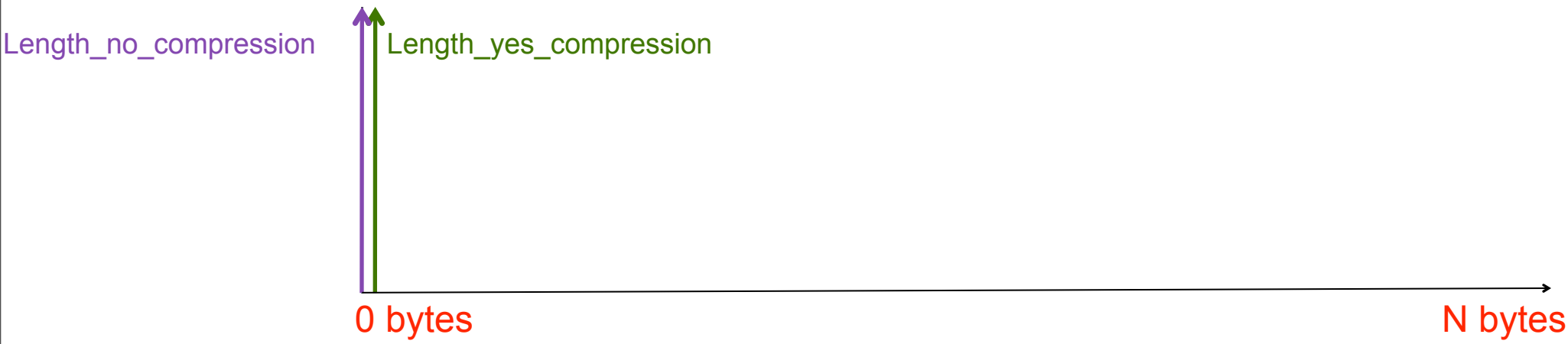


Decision Methodology

- 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

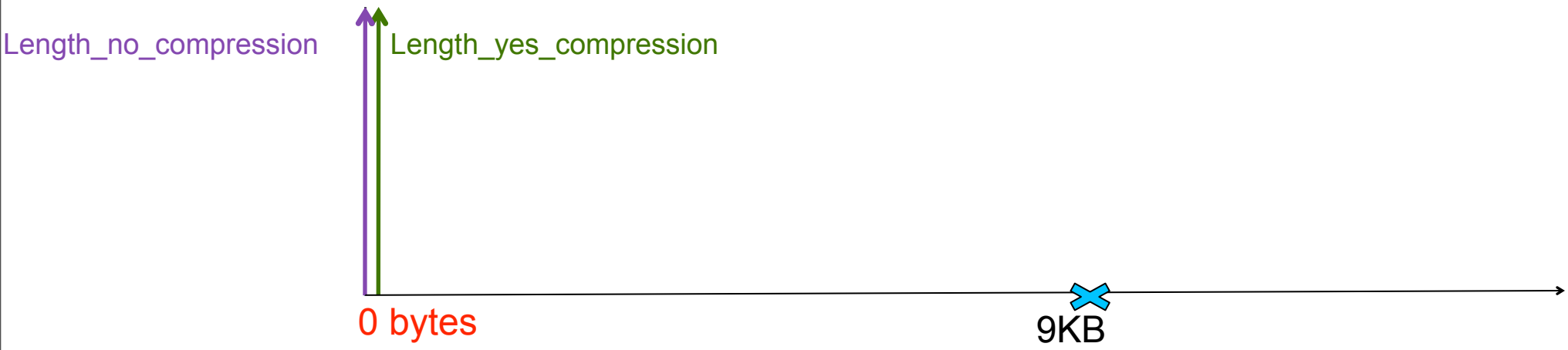
Process 0



Decision Methodology

Process 0

Message Integer
Size 9KB

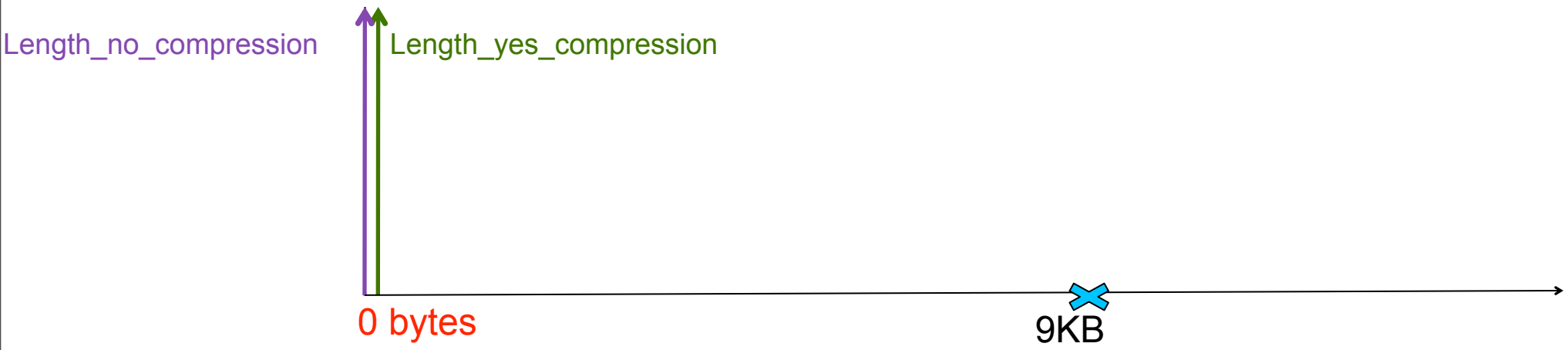


Decision Methodology

Process 0

Message Integer
Size 9KB

Study size message
And thresholds



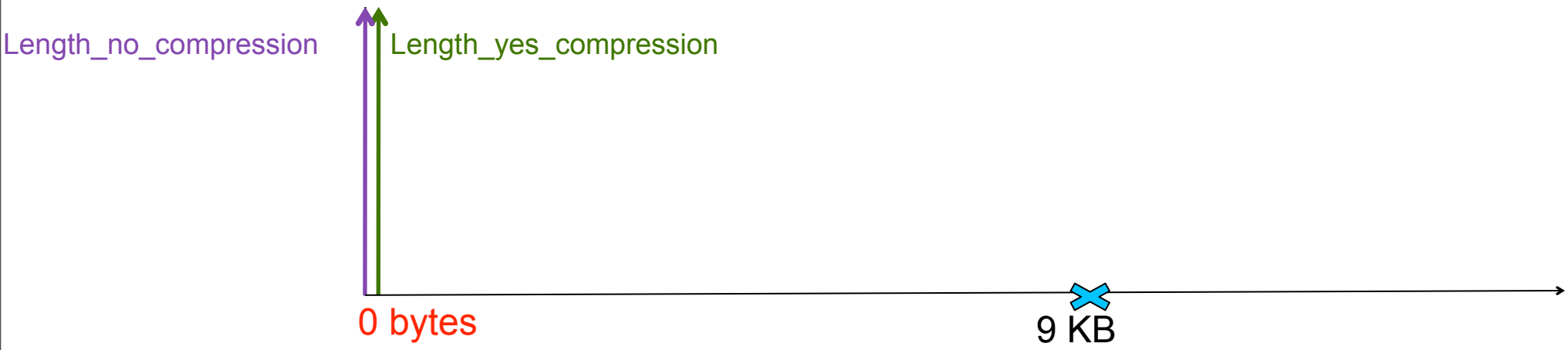
Decision Methodology

Process 0

Message Integer
Size 9KB

Study size message
And thresholds

IF (size > length_yes_compression)
&& (length_yes_compression !=0)
Compress message



Decision Methodology

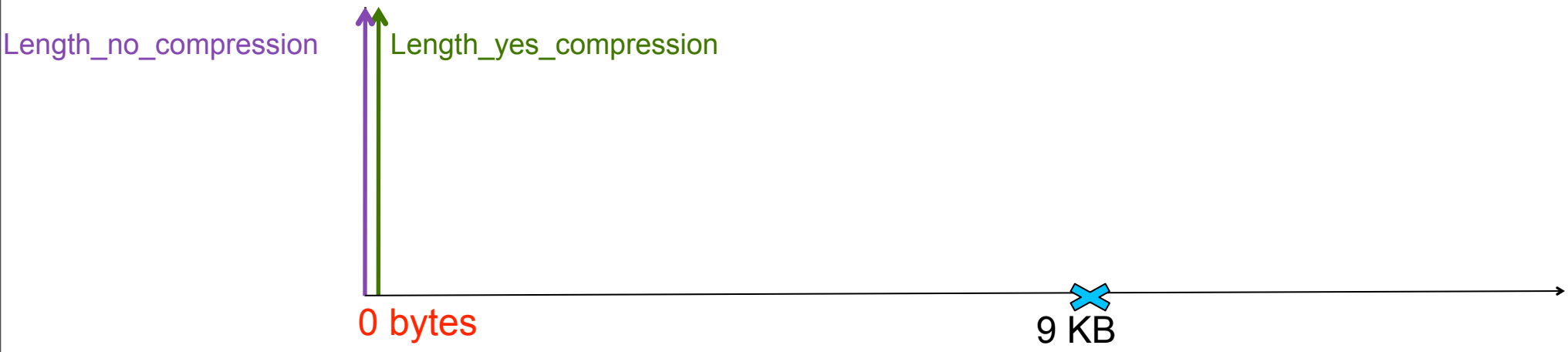
Process 0

Message Integer
Size 9KB

Study size message
And thresholds

IF (size > length_yes_compression)
&& (length_yes_compression !=0)
Compress message

IF (size < length_no_compression)
&& (length_no_compression !=0)
No compress message



Decision Methodology

Process 0

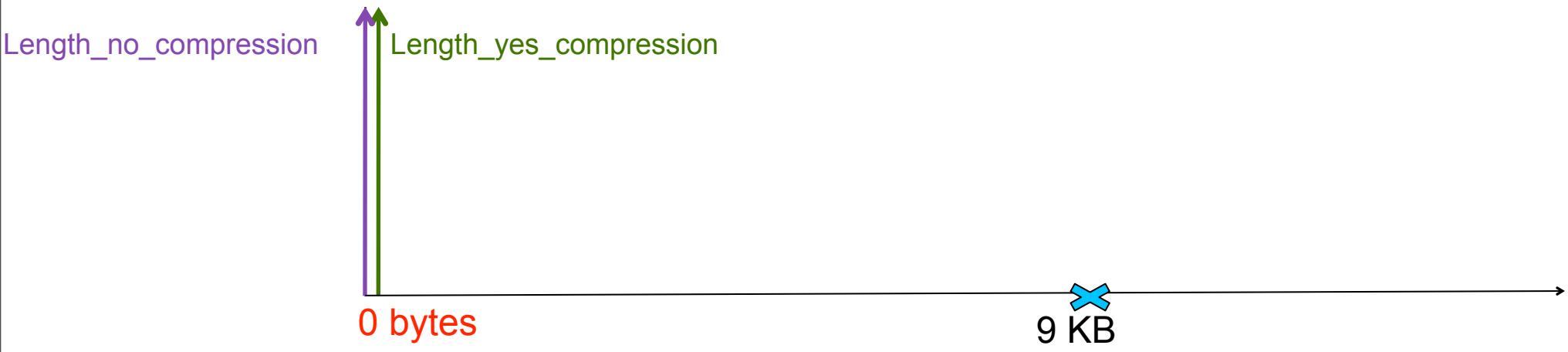
Message Integer
Size 9KB

Study size message
And thresholds

IF (size > length_yes_compression)
&& (length_yes_compression !=0)
Compress message

IF (size < length_no_compression)
&& (length_no_compression !=0)
No compress message

IF (Between thresholds)
|| (thresholds == 0)
Speedups calculation



Decision Methodology

Process 0

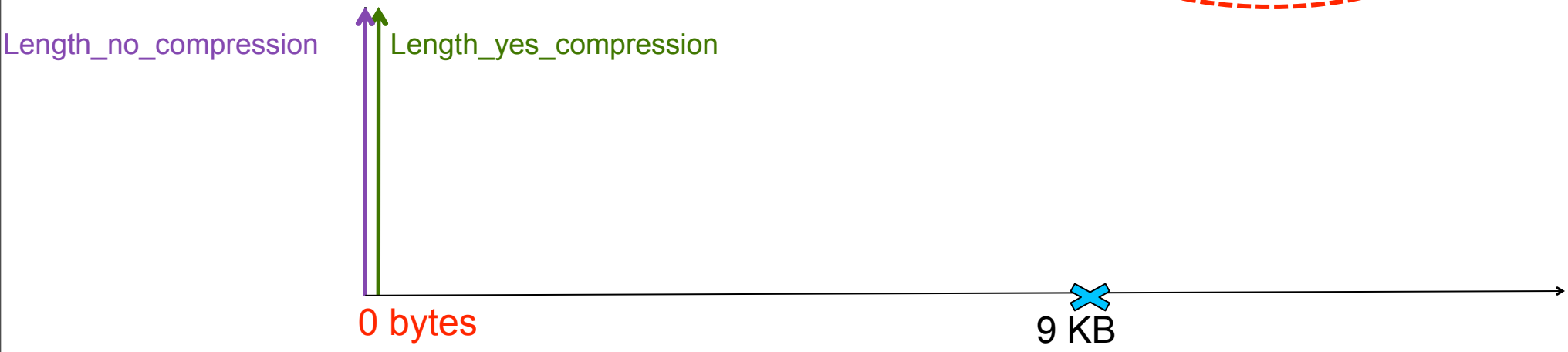
Message Integer
Size 9KB

Study size message
And thresholds

IF (size > length_yes_compression)
&& (length_yes_compression !=0)
Compress message

IF (size < length_no_compression)
&& (length_no_compression !=0)
No compress message

IF (Between thresholds)
|| (thresholds == 0)
Speedups calculation



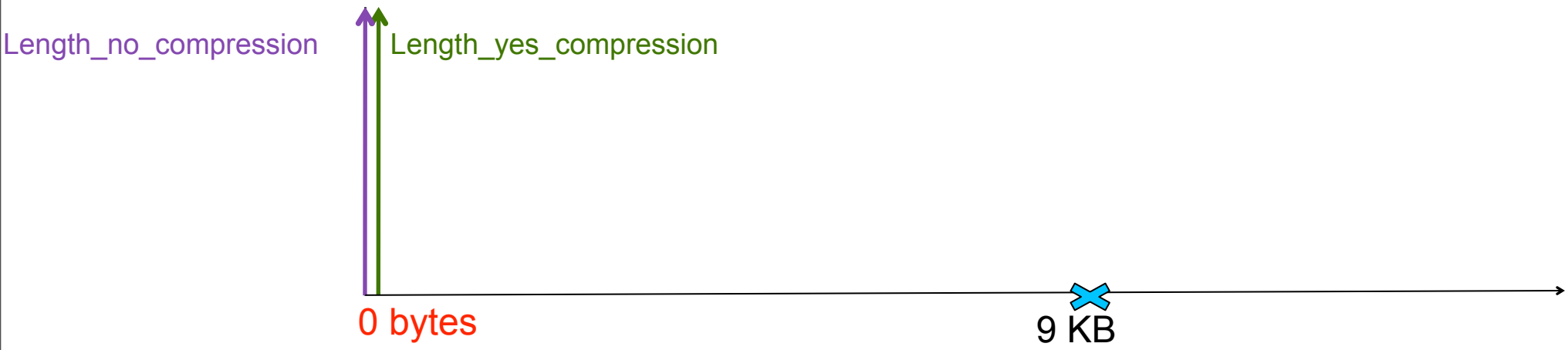
Decision Methodology

Process 0

Message Integer Size 9KB

Speedups Calculation

I. Select compressors
II. Speedups calculation
III. Select compressor with greater speedup
IV. Or (if all speedup less or equal 0) No compress

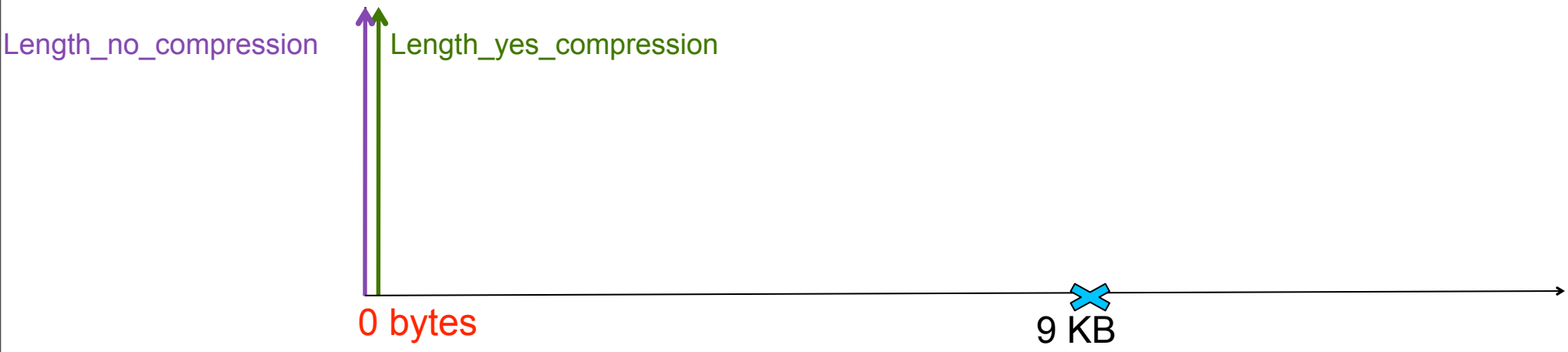


Decision Methodology

Process 0

Message Integer
Size 9KB

Speedups
Calculation → Compress with LZO



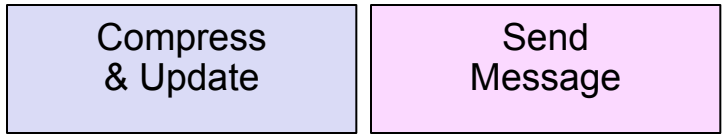
Decision Methodology

Process 0

Compress
& Update

Decision Methodology

Process 0



Decision Methodology

Process 0

Message Integer
Size 16KB

Decision Methodology

Process 0

Message Integer
Size 16KB

Study size message
And thresholds

IF (size > length_yes_compression)
&& (length_yes_compression !=0)
Compress message

IF (size < length_no_compression)
&& (length_no_compression !=0)
No compress message

IF (Between thresholds)
|| (thresholds == 0)
Speedups Calculation

Decision Methodology

Process 0

Message Integer
Size 16KB

Study size message
And thresholds

IF (size > length_yes_compression)
&& (length_yes_compression !=0)
Compress message

IF (size < length_no_compression)
&& (length_no_compression !=0)
No compress message

IF (Between thresholds)
|| (thresholds == 0)
Speedups Calculation

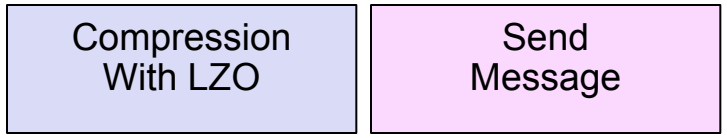
Decision Methodology

Process 0

Compression
With LZO

Decision Methodology

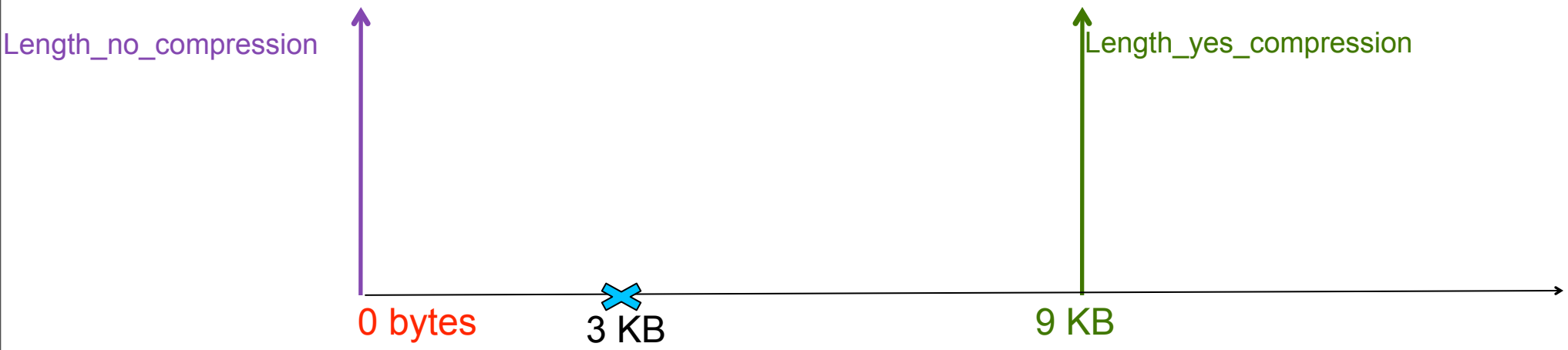
Process 0



Decision Methodology

Process 0

Message Integer
Size 3KB



Decision Methodology

Process 0

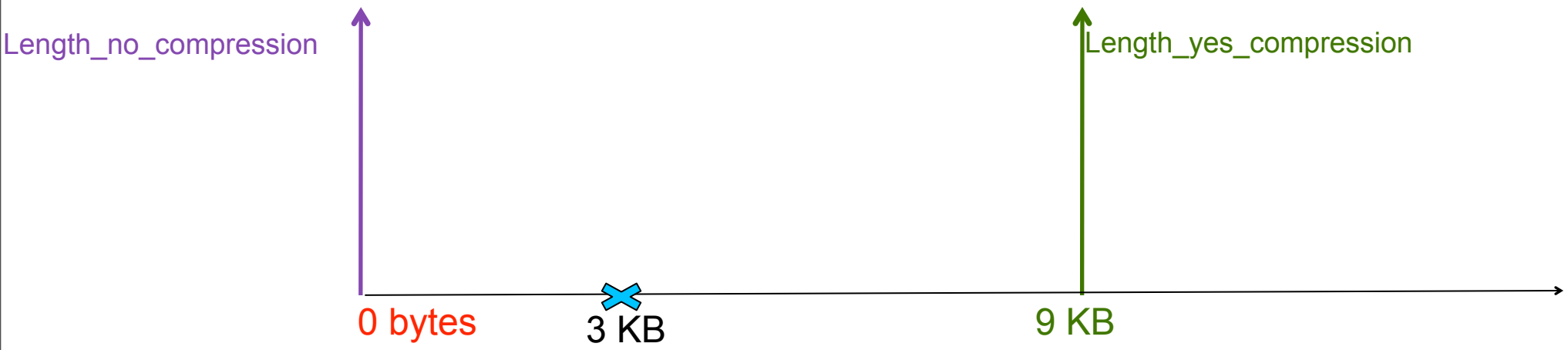
Message Integer
Size 3KB

Study size message
And thresholds

IF (size > length_yes_compression)
&& (length_yes_compression !=0)
Compress message

IF (size < length_no_compression)
&& (length_no_compression !=0)
No compress message

IF (Between thresholds)
|| (thresholds == 0)
Speedups Calculation



Decision Methodology

Process 0

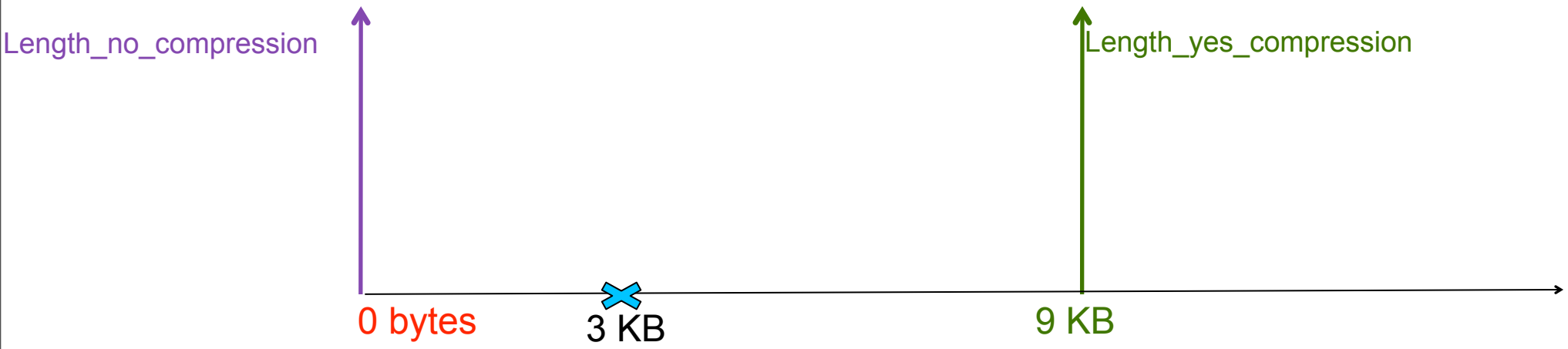
Message Integer
Size 3KB

Study size message
And thresholds

IF (size > length_yes_compression)
&& (length_yes_compression !=0)
Compress message

IF (size < length_no_compression)
&& (length_no_compression !=0)
No compress message

IF (Between thresholds)
|| (thresholds == 0)
Speedups Calculation

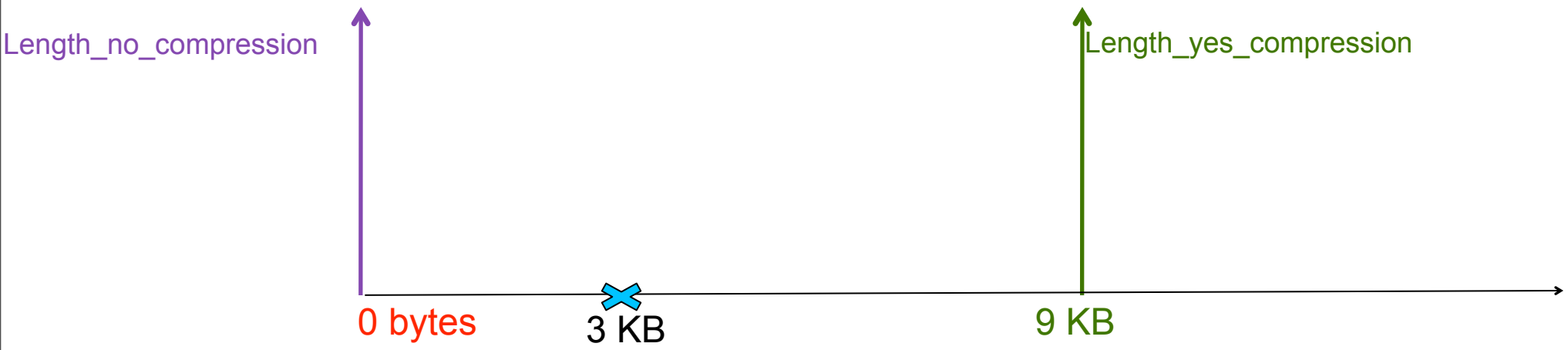


Decision Methodology

Process 0

Message Integer Size 3KB

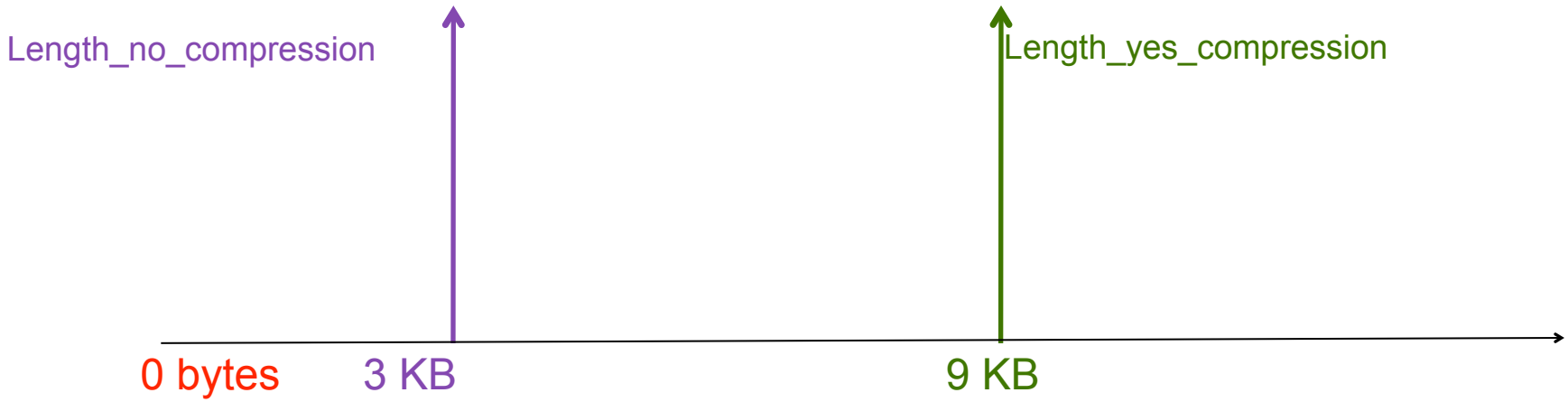
Speedups Calculation → No Compress



Decision Methodology

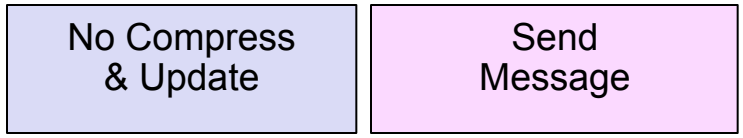
Process 0

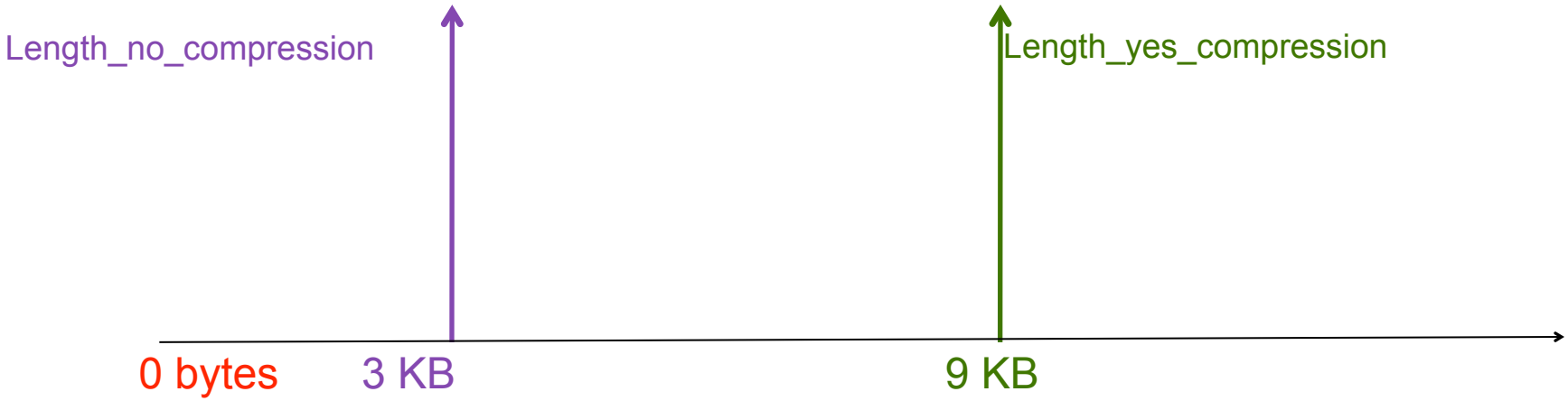
No Compress
& Update



Decision Methodology

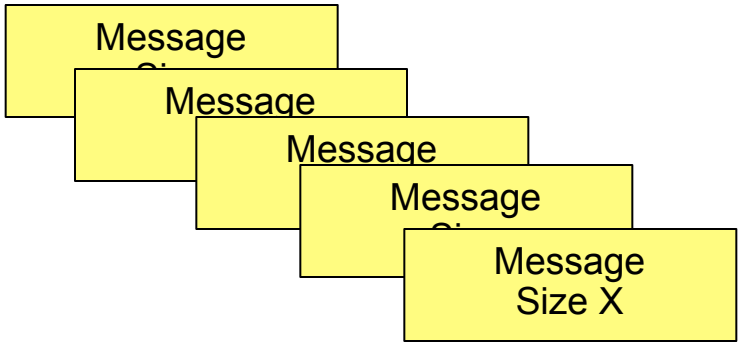
Process 0

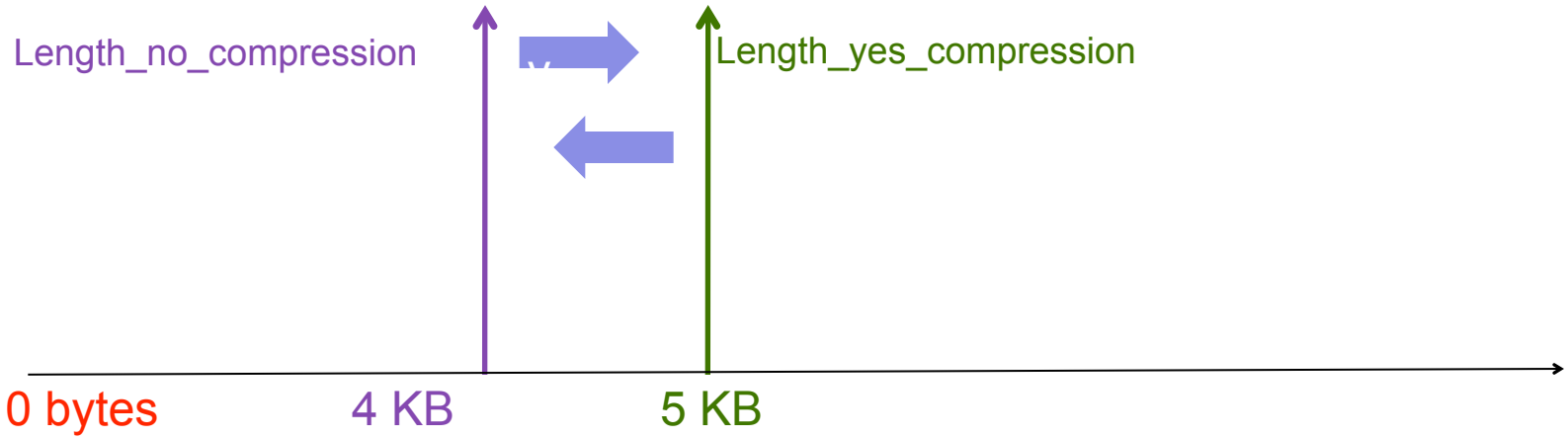




Decision Methodology

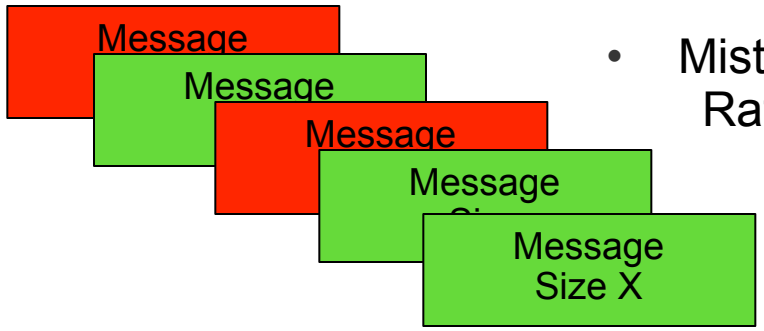
Process 0



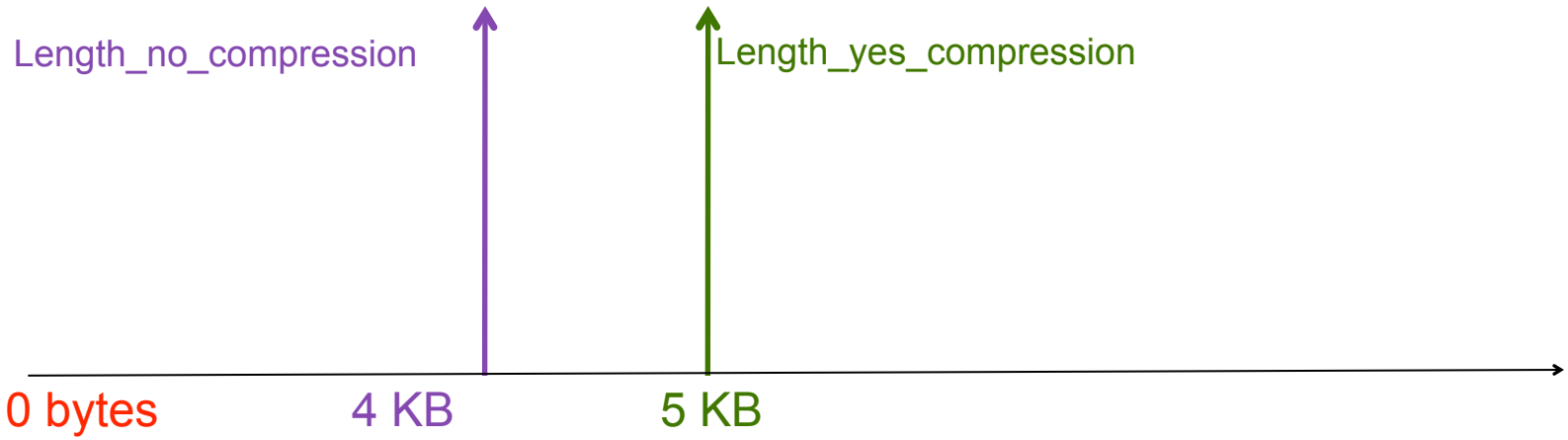


Re-evaluation Methodology

Process 0

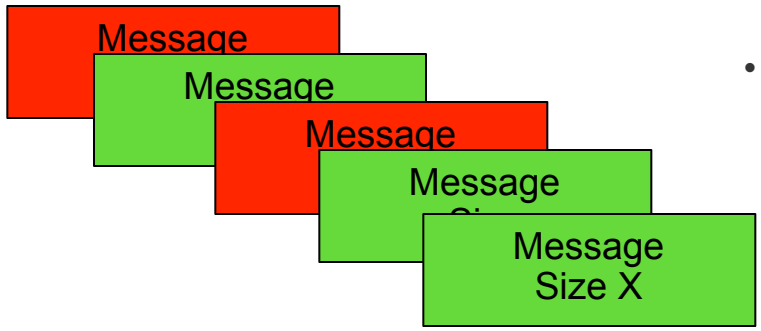


- Mistake:
  Ratio compress message < MIN_RATIO

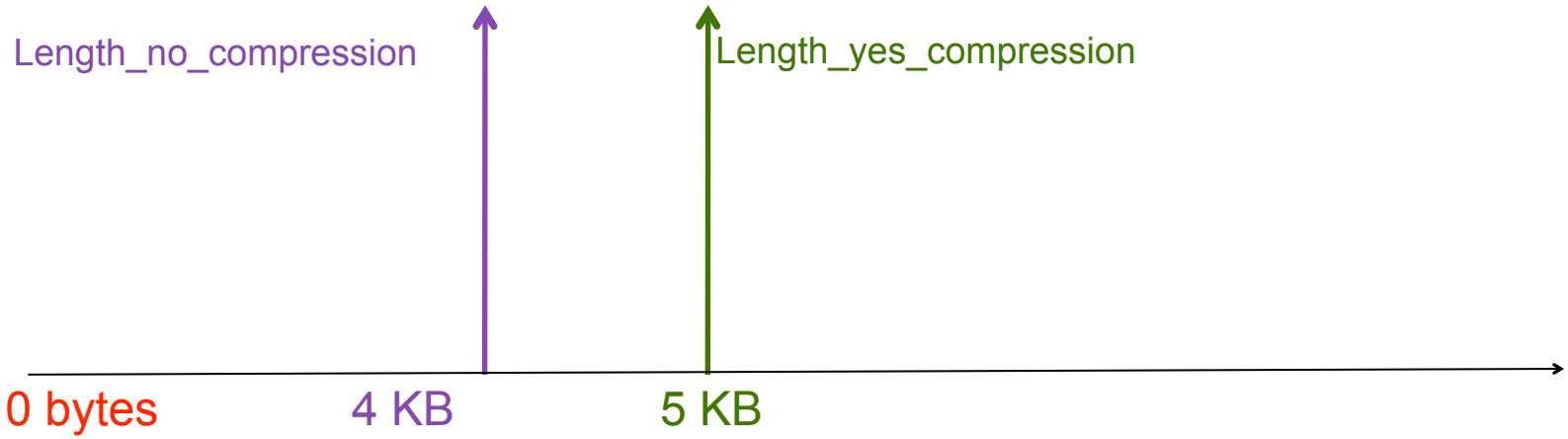


Re-evaluation Methodology

Process 0

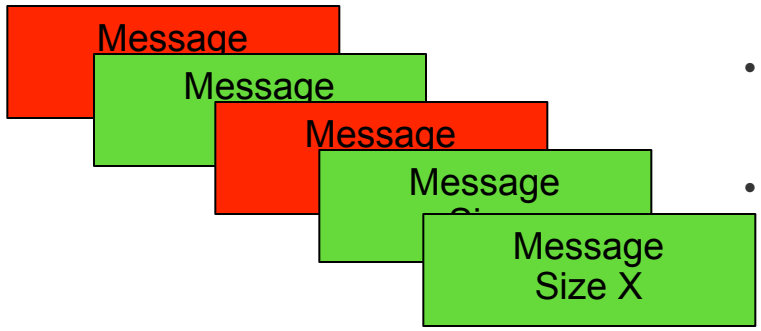


- Number of message without compress > MAX_MESSAGE

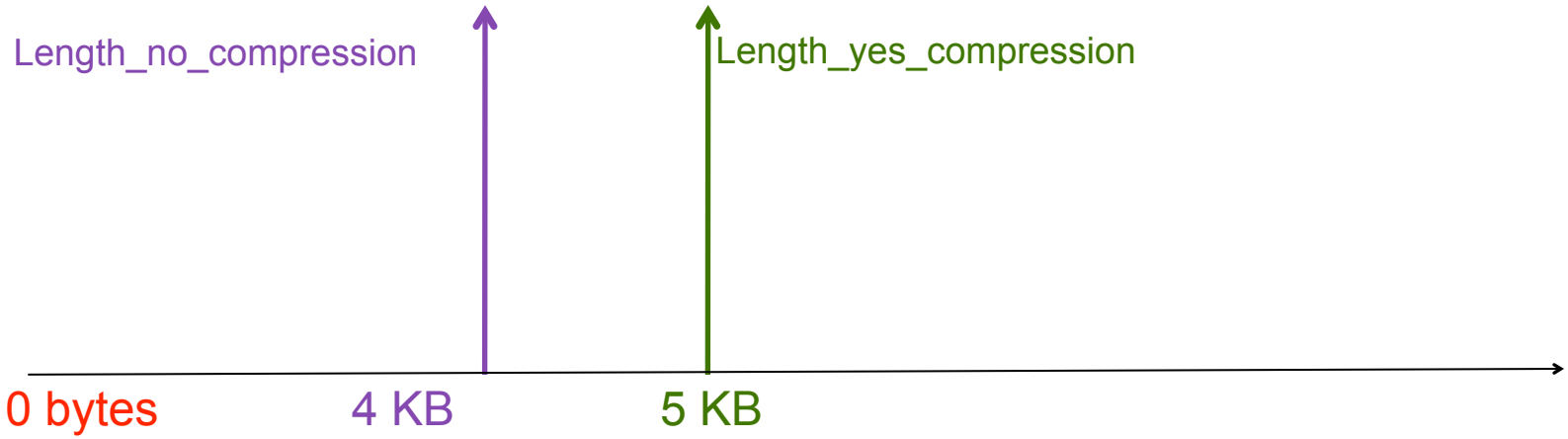


Re-evaluation Methodology

Process 0

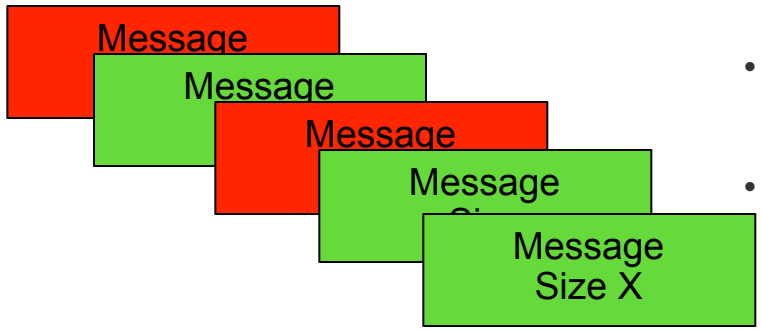


- Number of message without compress > MAX_MESSAGE
- Number Mistakes > MAX_MISTAKES && Frecuency Mistakes < MIN_FRECUENCY



Re-evaluation Methodology

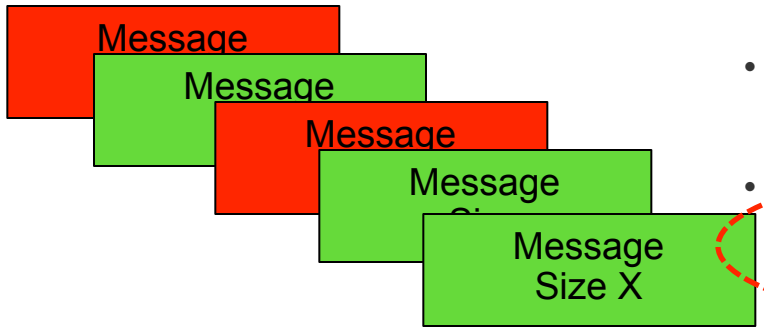
Process 0



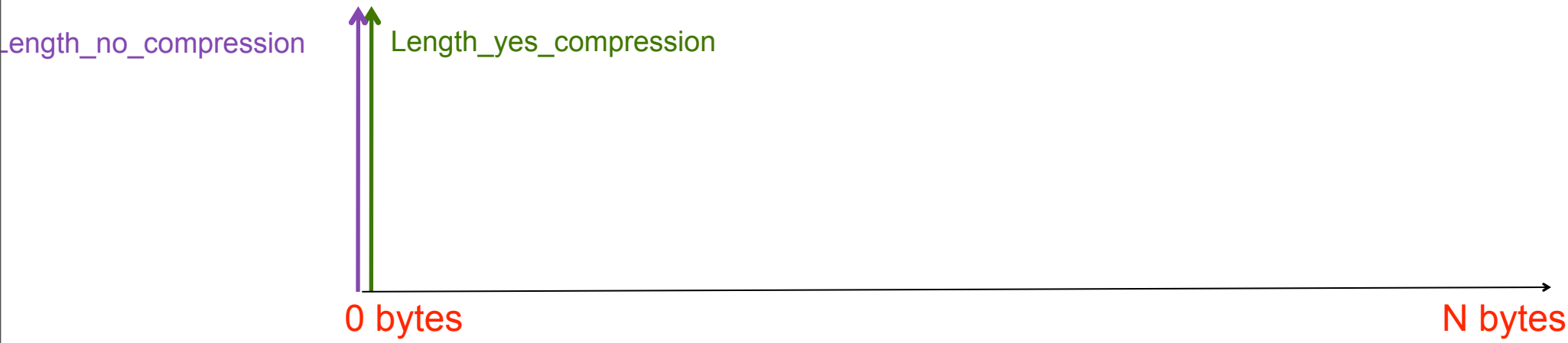
- Number of message without compress > MAX_MESSAGE
- Number Mistakes > MAX_MISTAKES && Frecuency Mistakes < MIN_FRECUENCY

Re-evaluation Methodology

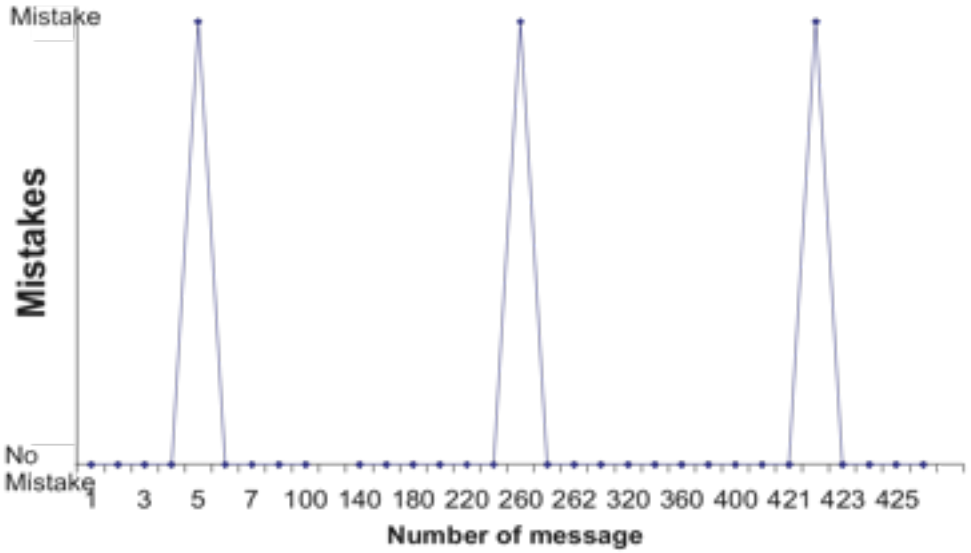
Process 0



- Number of message without compress > MAX_MESSAGE
- Number Mistakes > MAX_MISTAKES && Frecuency Mistakes < MIN_FRECUENCY

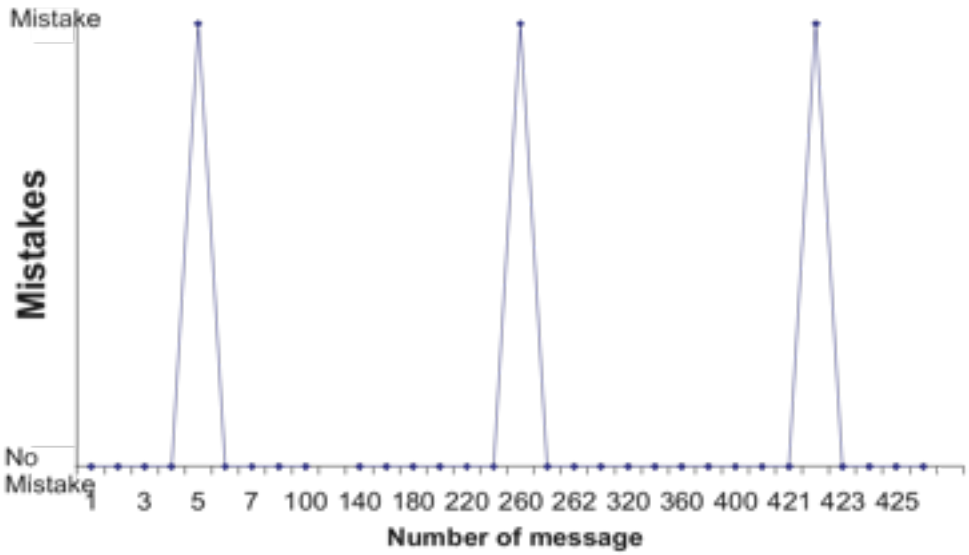


Different cases of re-evaluation

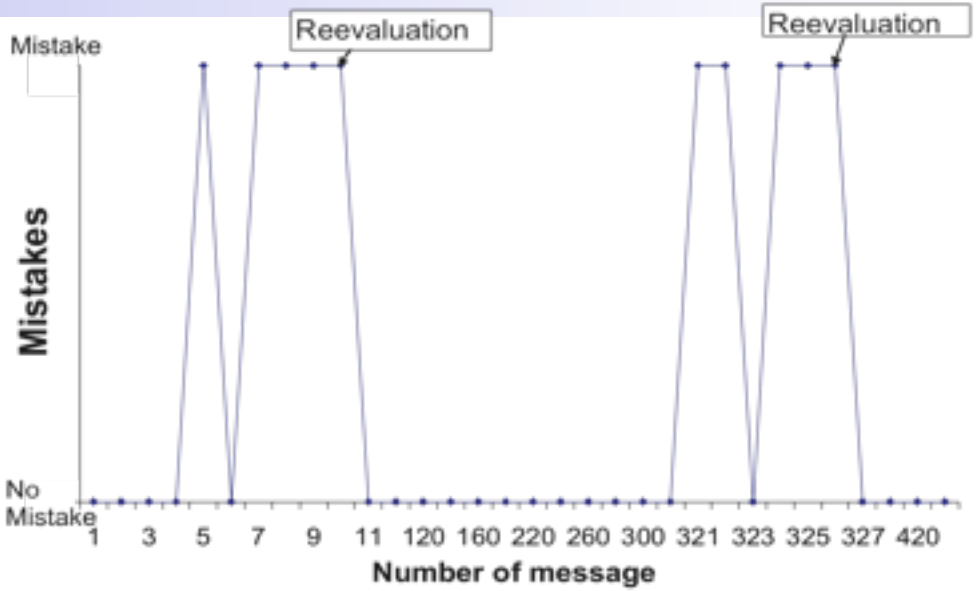


High frequency mistakes

Different cases of re-evaluation

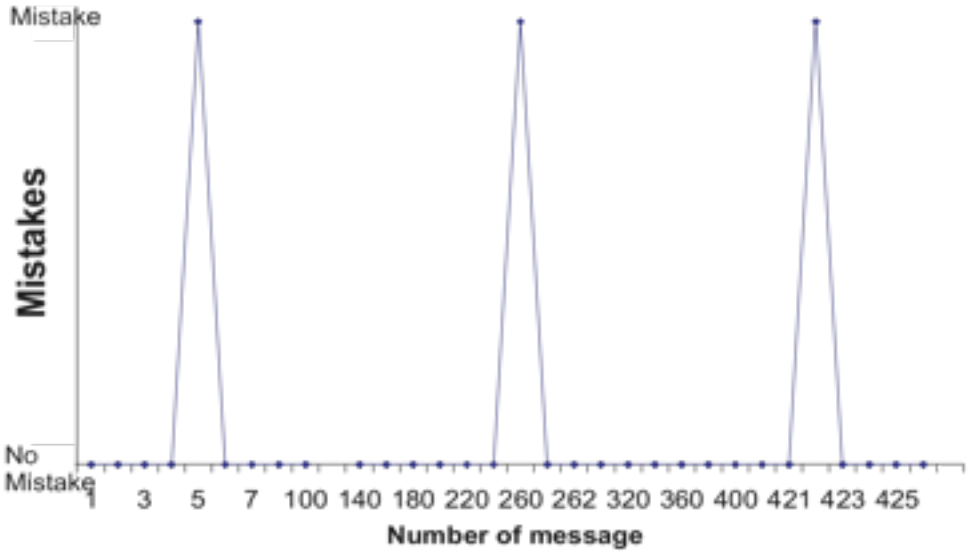


High frequency mistakes

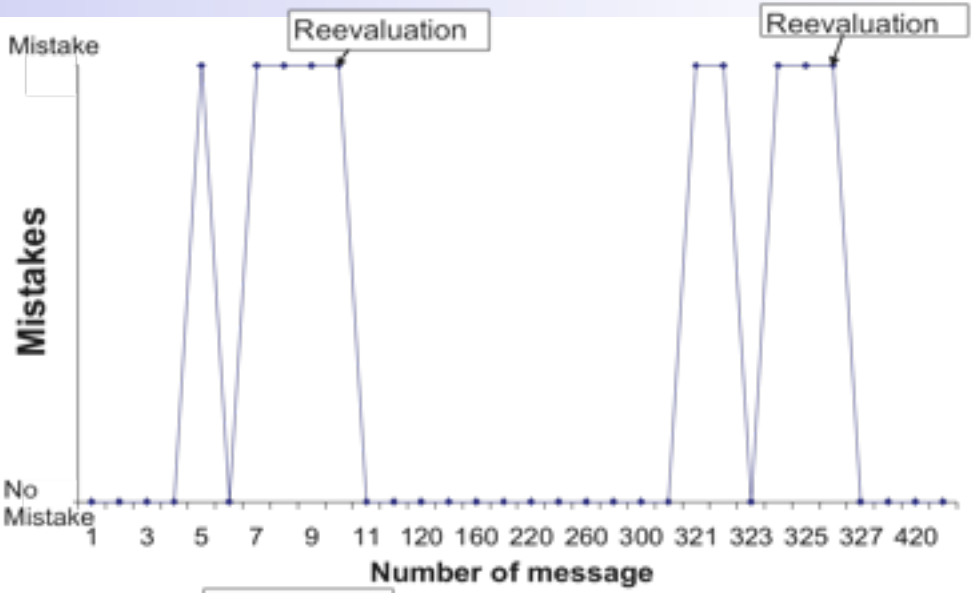


Low frequency mistakes

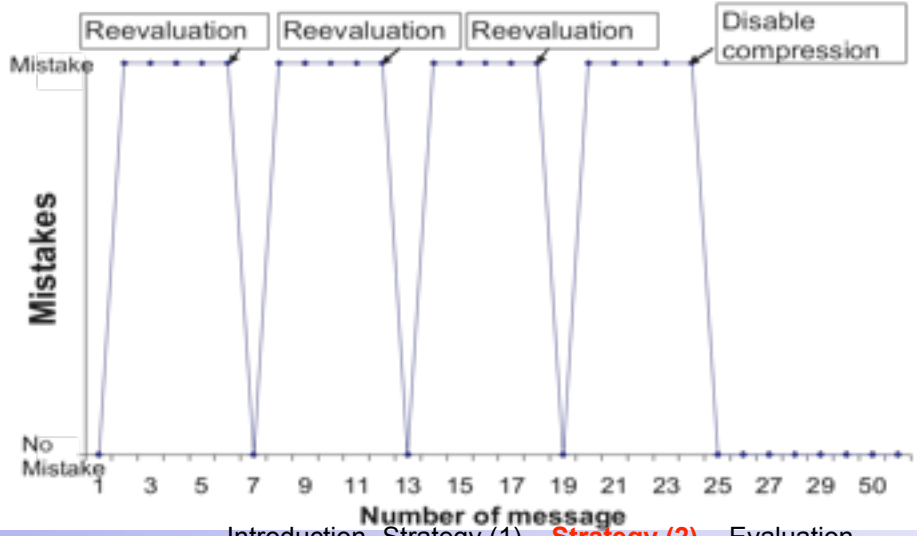
Different cases of re-evaluation



High frequency mistakes



Low frequency mistakes



Many reevaluations in a short time

3 Compression Strategy: Guided Strategy

- Scientific Application:
  - Are executed many times in the same cluster
  - With same input parameters.
- Guided strategy (GS)→ takes the decisions off-line by storing all message from the first execution.
- Decisions (compress or not, and which algorithm) are stored in a file called Decision Rules
- Subsequent executions of the same application and same input parameters, GS applies rules of Decision Rules.

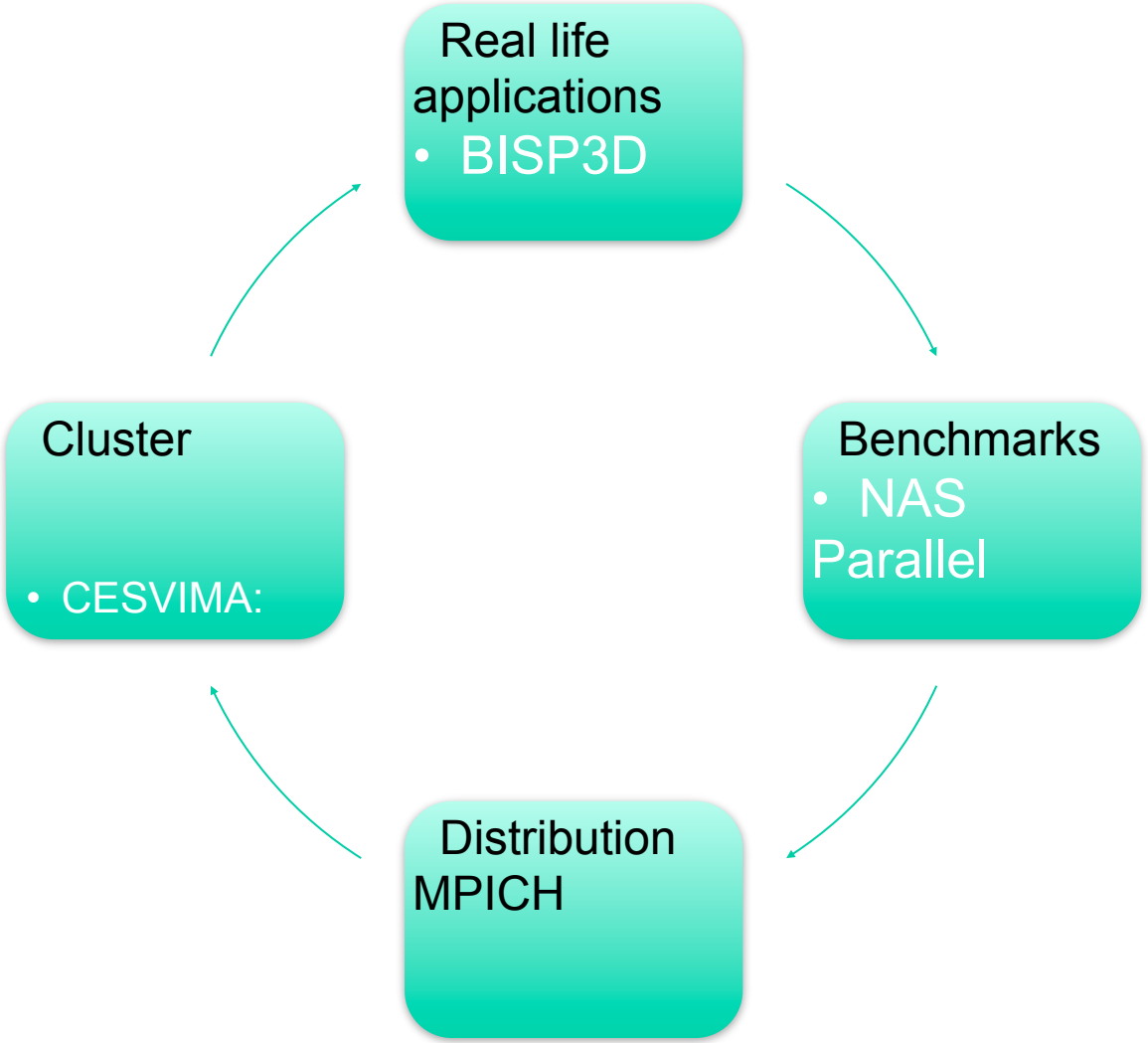
And now??

- Use MPICH2 instead MPICH1.2 and adapt the different compression strategies.
- No apply compression when the processes are in the same node
- New compression algorithms
- New ideas:
  - Take account the features of communications
    - MPI_Broadcast → Apply compression once per operation.

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 I/O and communication operations.
5. Evaluation tools

Phd. Thesis evaluation tools



Clusters used in our experiments

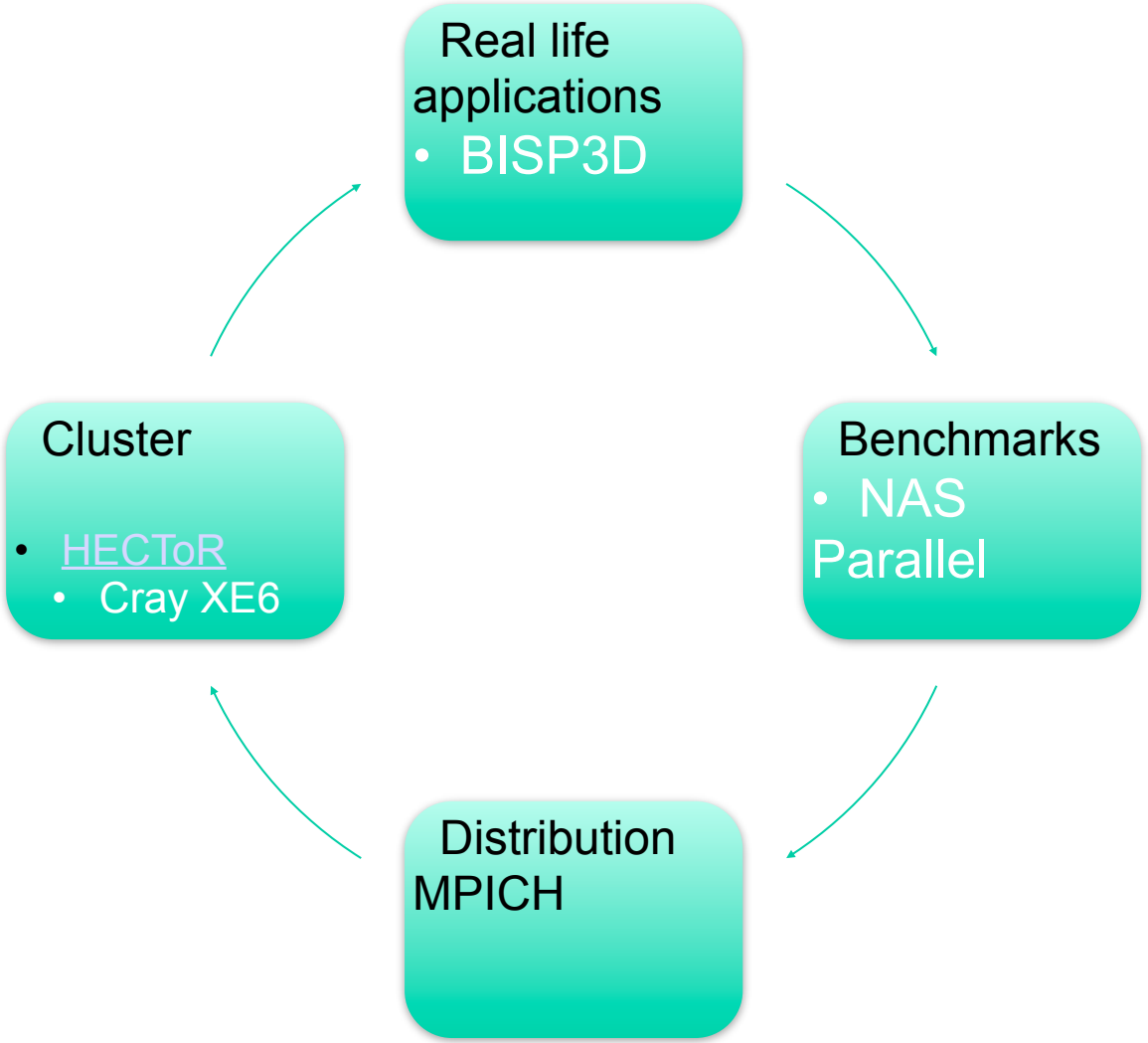
Clusters A
- Magerit Cluster (CESVIMA)
- 1200eServer BladeCenter nodes.
- **Myrinet**.
- Node 2 processor

Cluster B
- University Carlos III Labs.
- 22 **Dual Core**
- **Gigabit**
- Each node
- Intel(R) Xeon(TM)

Cluster C
- University Carlos III Labs.
- 4 **Tetra Core**
- **Gigabit**
- Each node
- Intel(R) Xeon(TM)

Cluster D
- University Carlos III Labs.
- 64 **Dual Core**
- **Ethernet**
- Each node
- AMD with 512GB of

New evaluation tool: HECToR

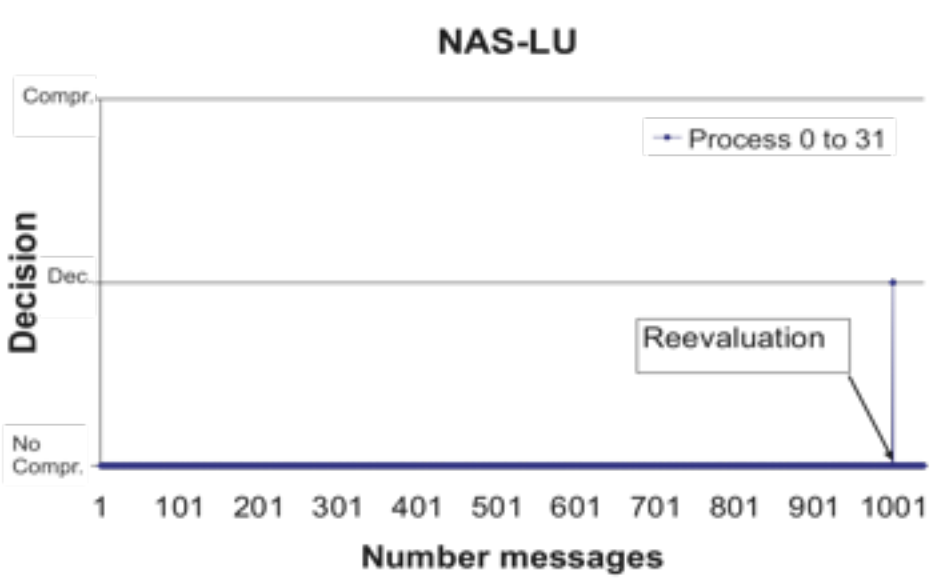


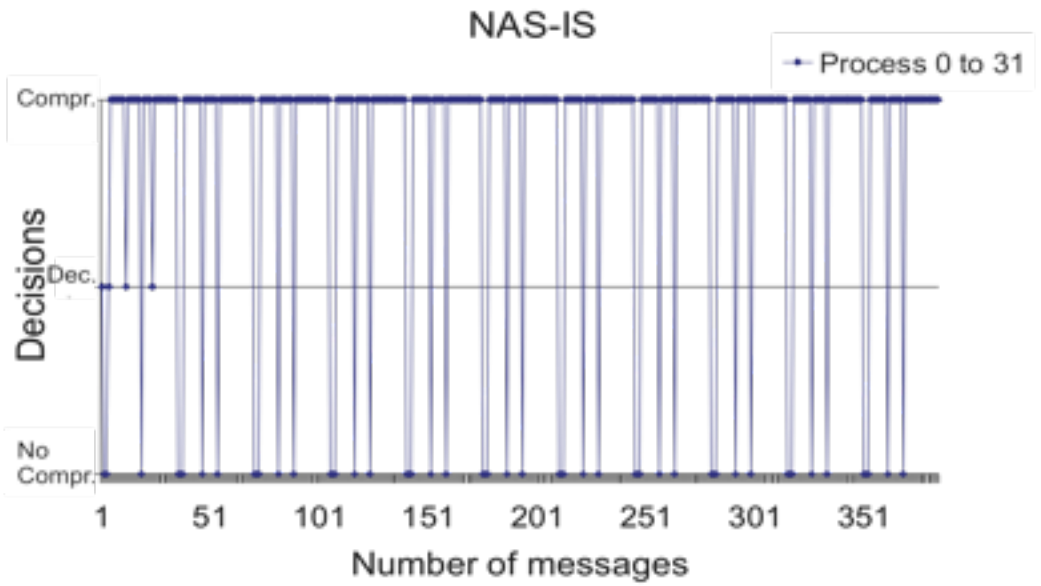
HECToR Hardware

Clusters HECToR

- Cray XE6
- 1800 compute nodes
- **Gemini interconect**.
- Each Node 2 AMD 2.1 GHz 12 core Opteron processors →44,544 cores
- Lustre ….

Runtime Adaptive Compression (RAS) Compression decisions.

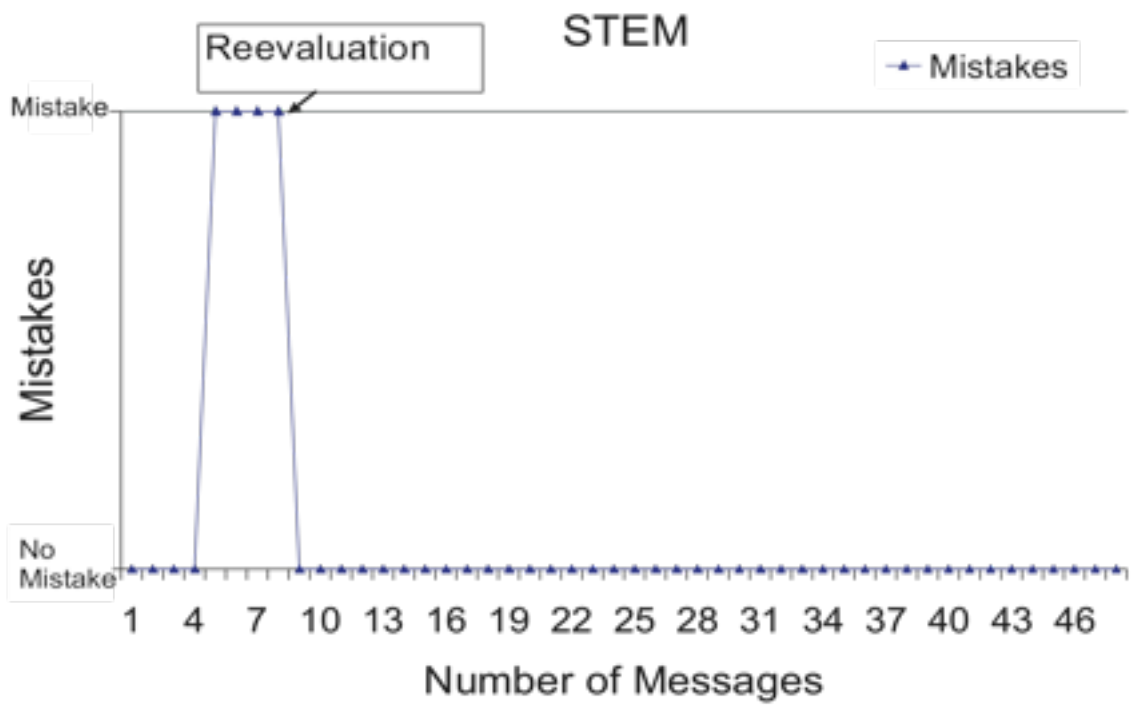




Benchmarks, which different kind of data, datatype and message size.

RAS adapts itself at run-time to the application behavior and takes different compression decisions.

Reevaluation case with STEM



- Several mistake in a short period of time:
  STEM. Initially selects RLE, and before the re-evaluation selects LZO.

-Number of messages sent without compression is higher than defined ratio. (NAS-LU)

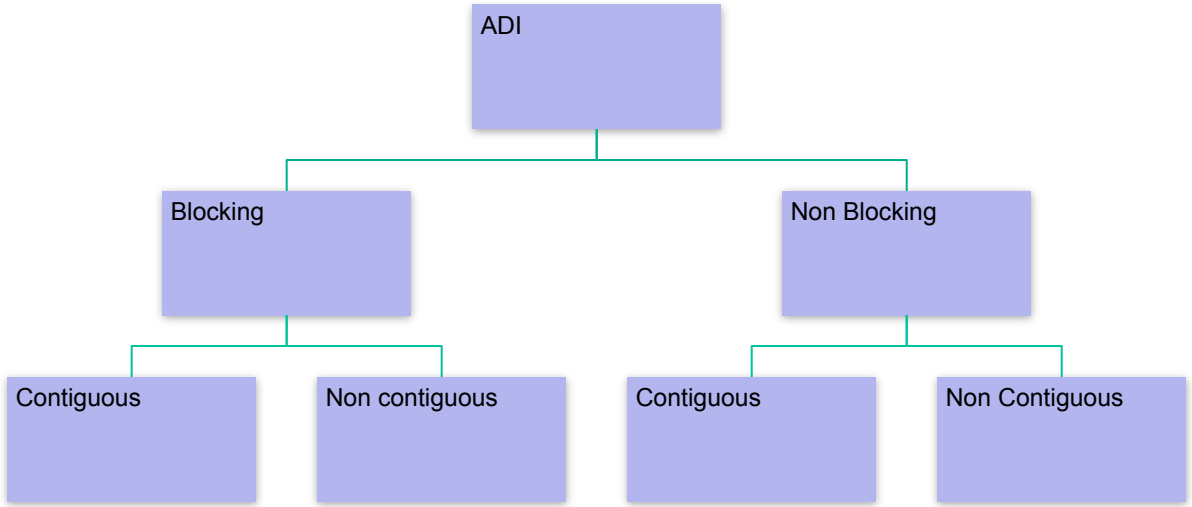
Questions??

- Thanks!!
- http://www.arcos.inf.uc3m.es/~rosaf/

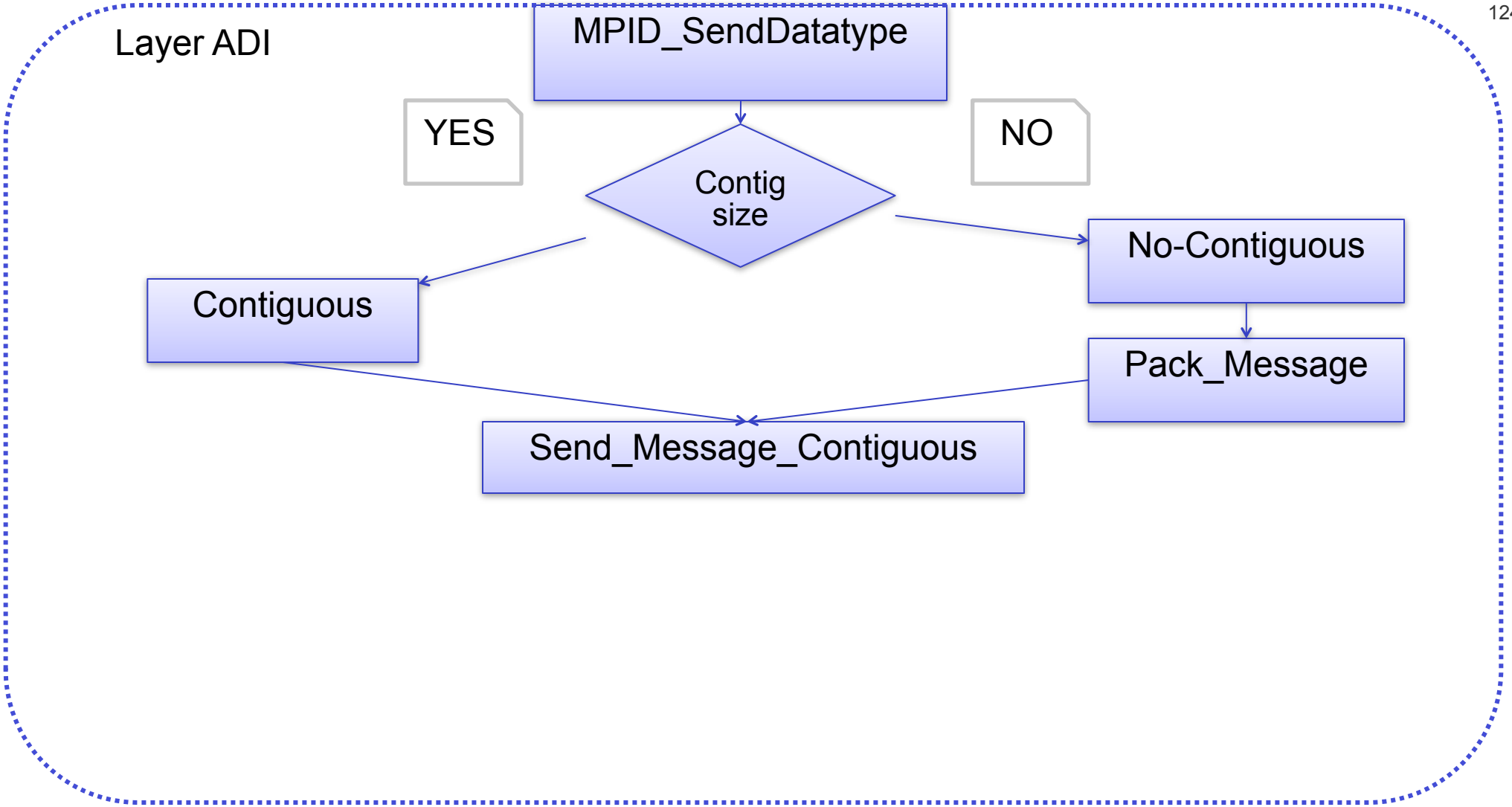
Dynamic optimization techniques to enhance scalability and performance of MPI-based applications on Multi-core cluster.

Author: Rosa Filgueira Vicente

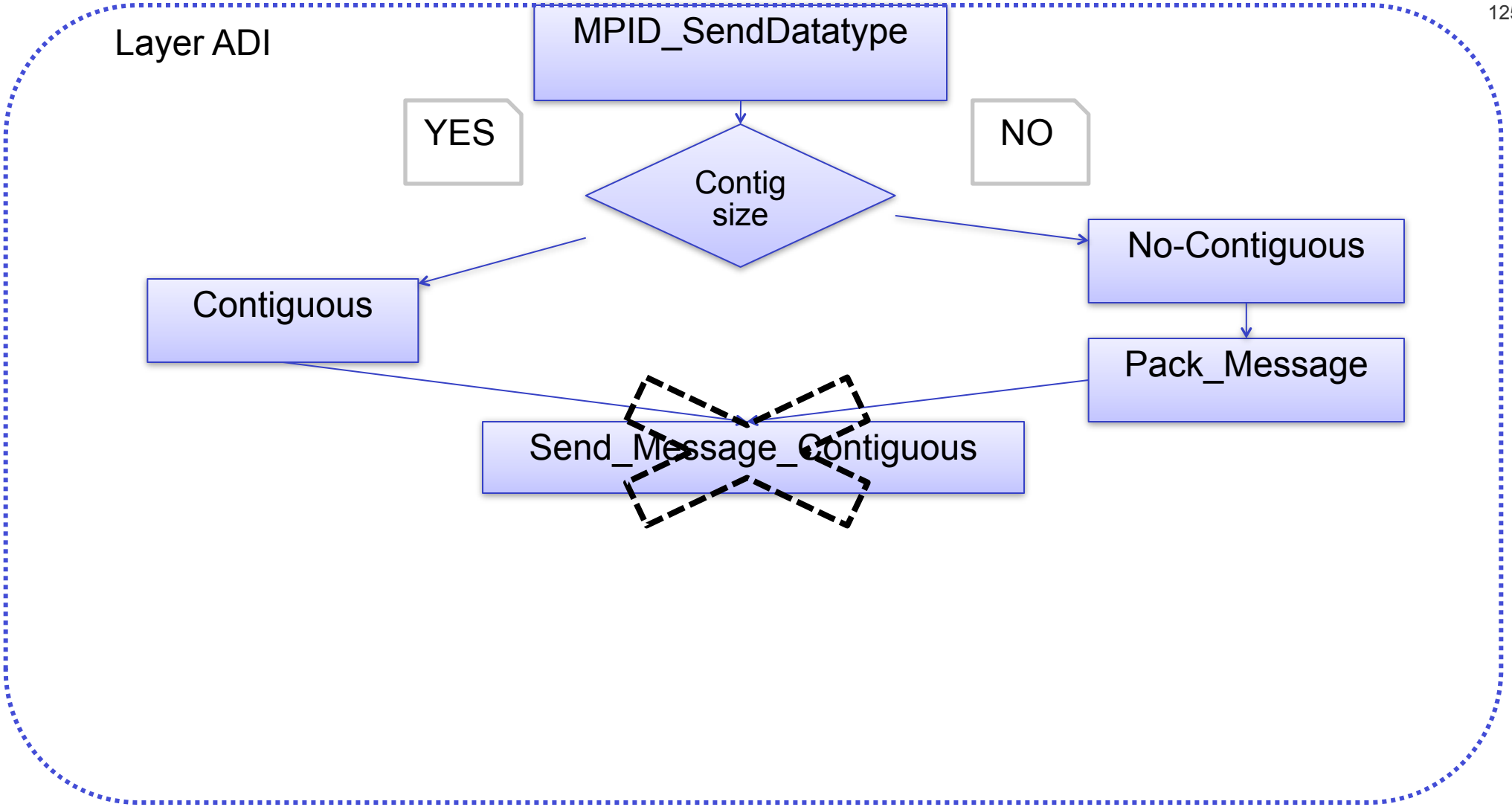
ADI Layer structure



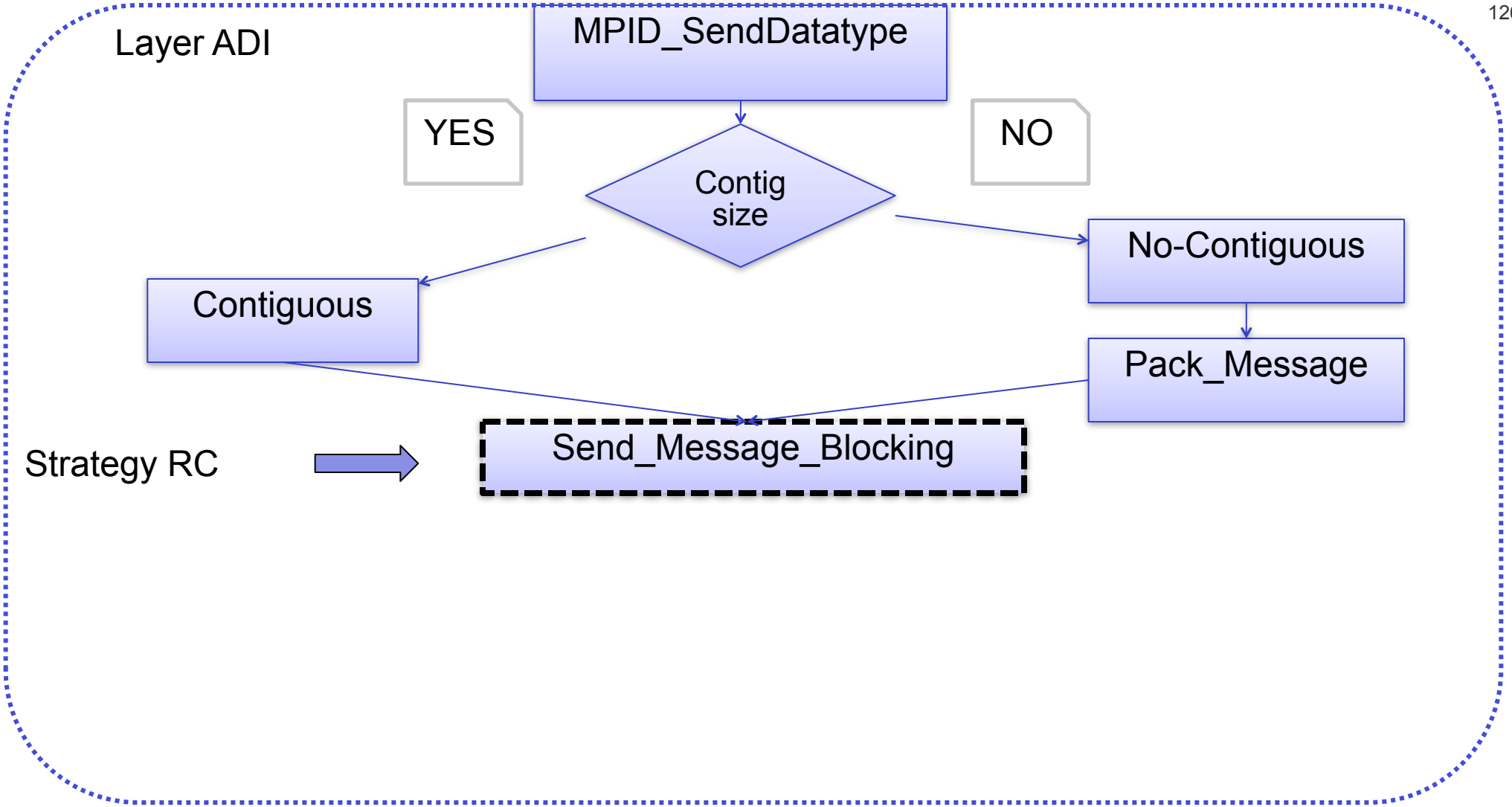
Send Message Blocking (MPICH)



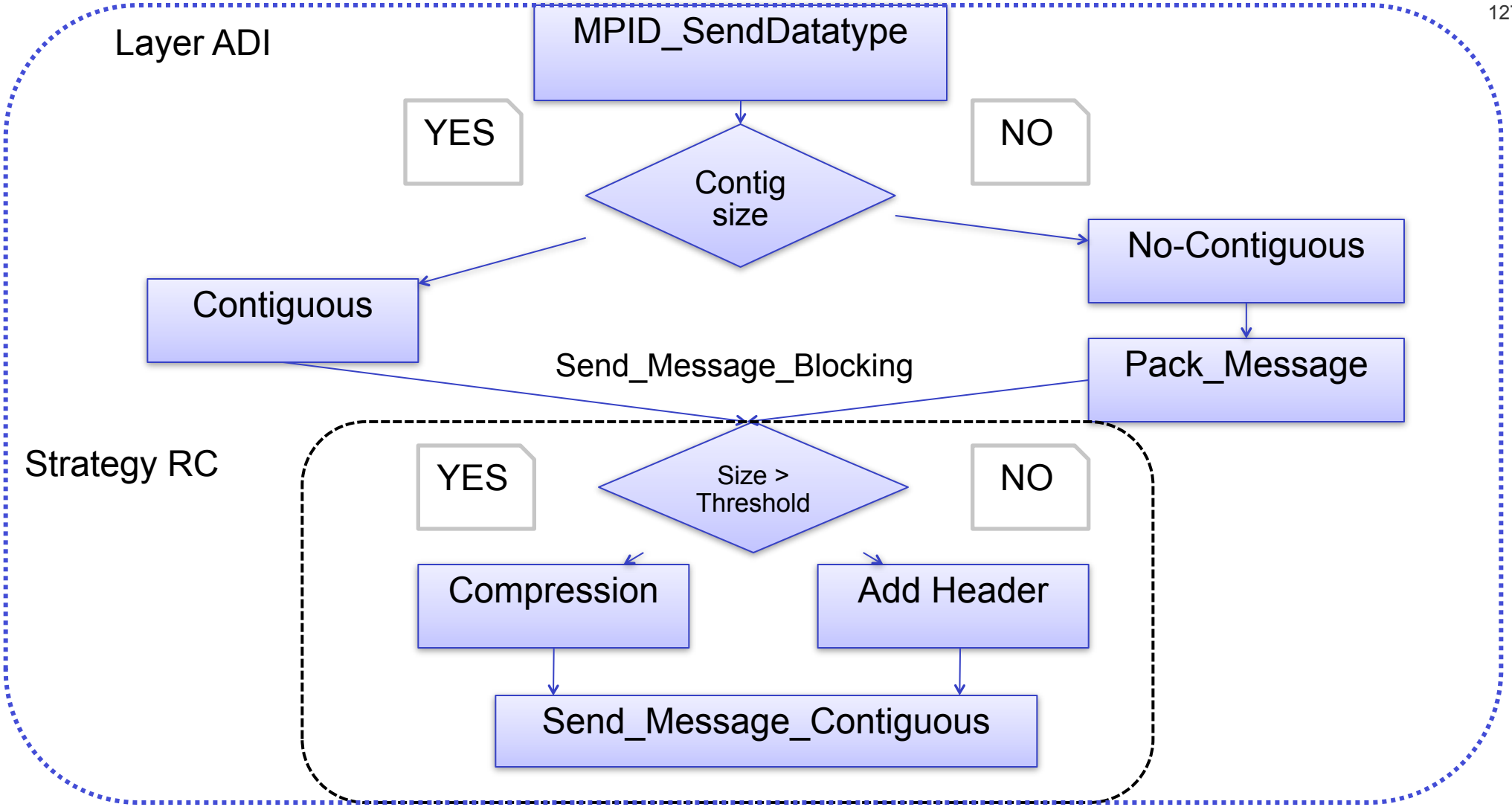
Send Message Blocking (RC)



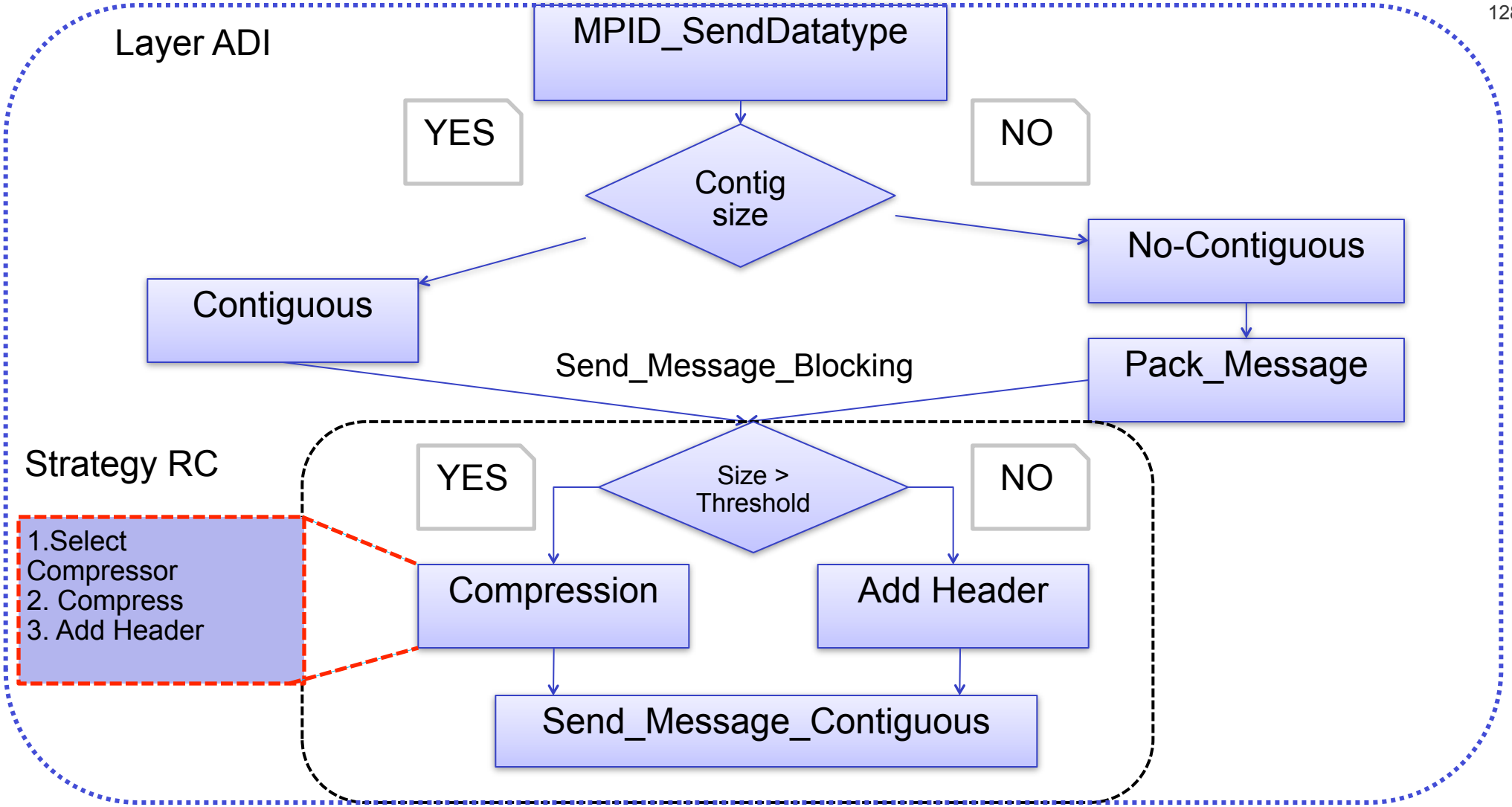
Send Message Blocking (RC)



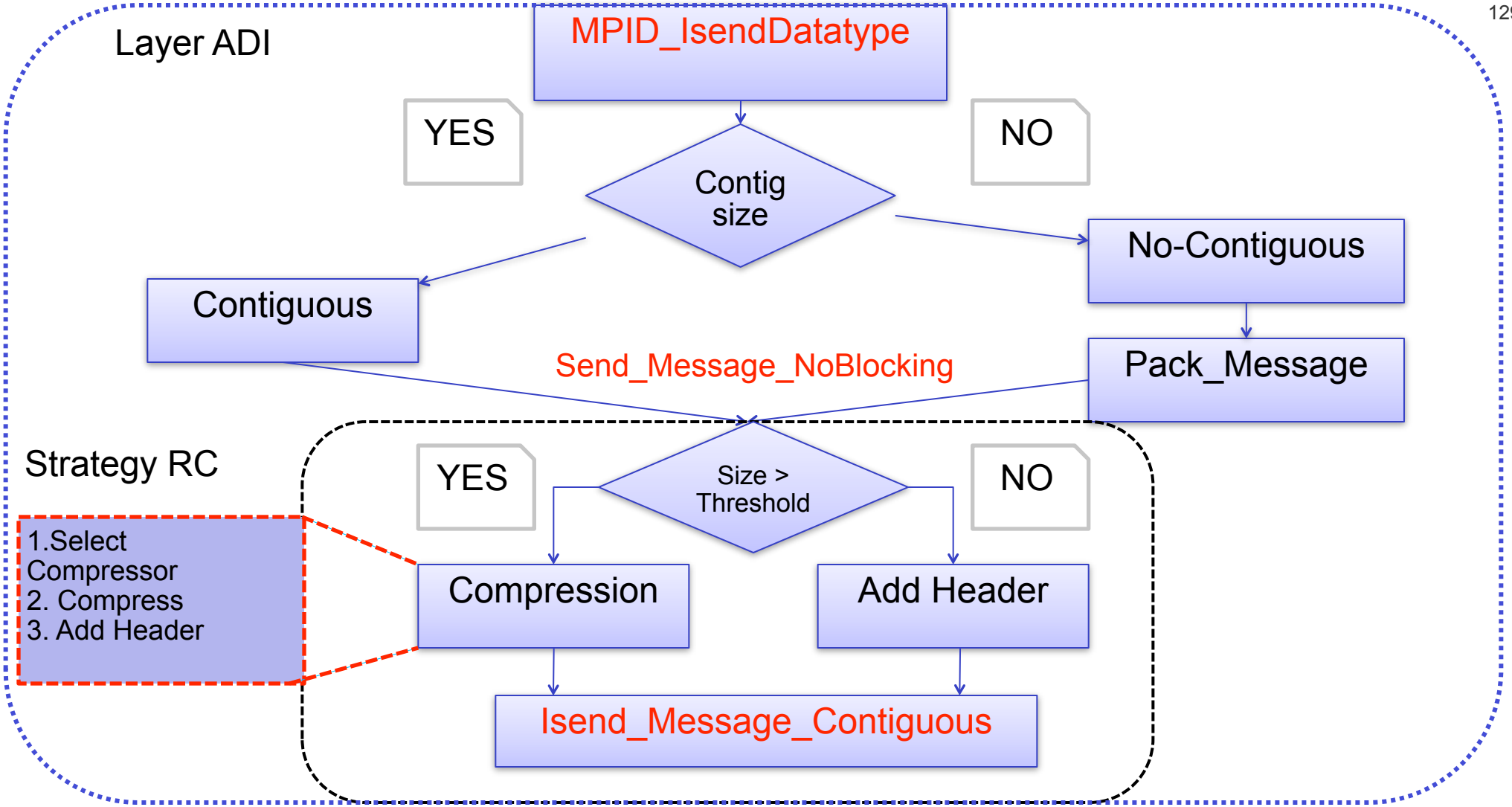
Send Message Blocking (RC)



Send Message Blocking (RC)

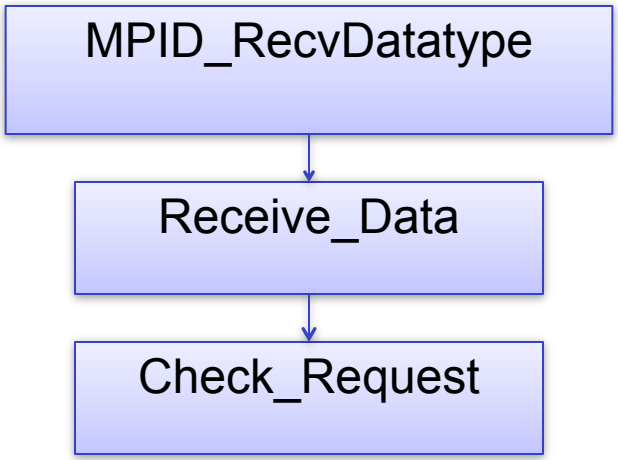


Send Message No Blocking (RC)



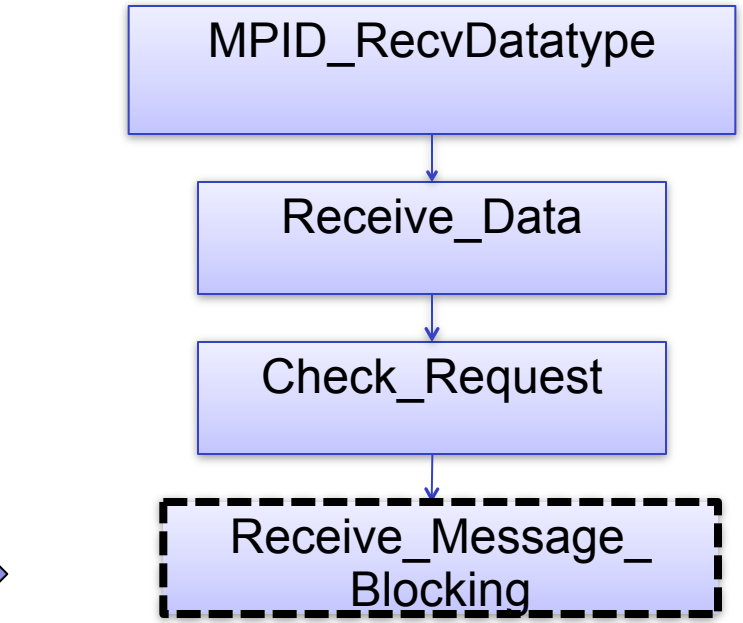
Receive message blocking (MPICH)

Layer ADI



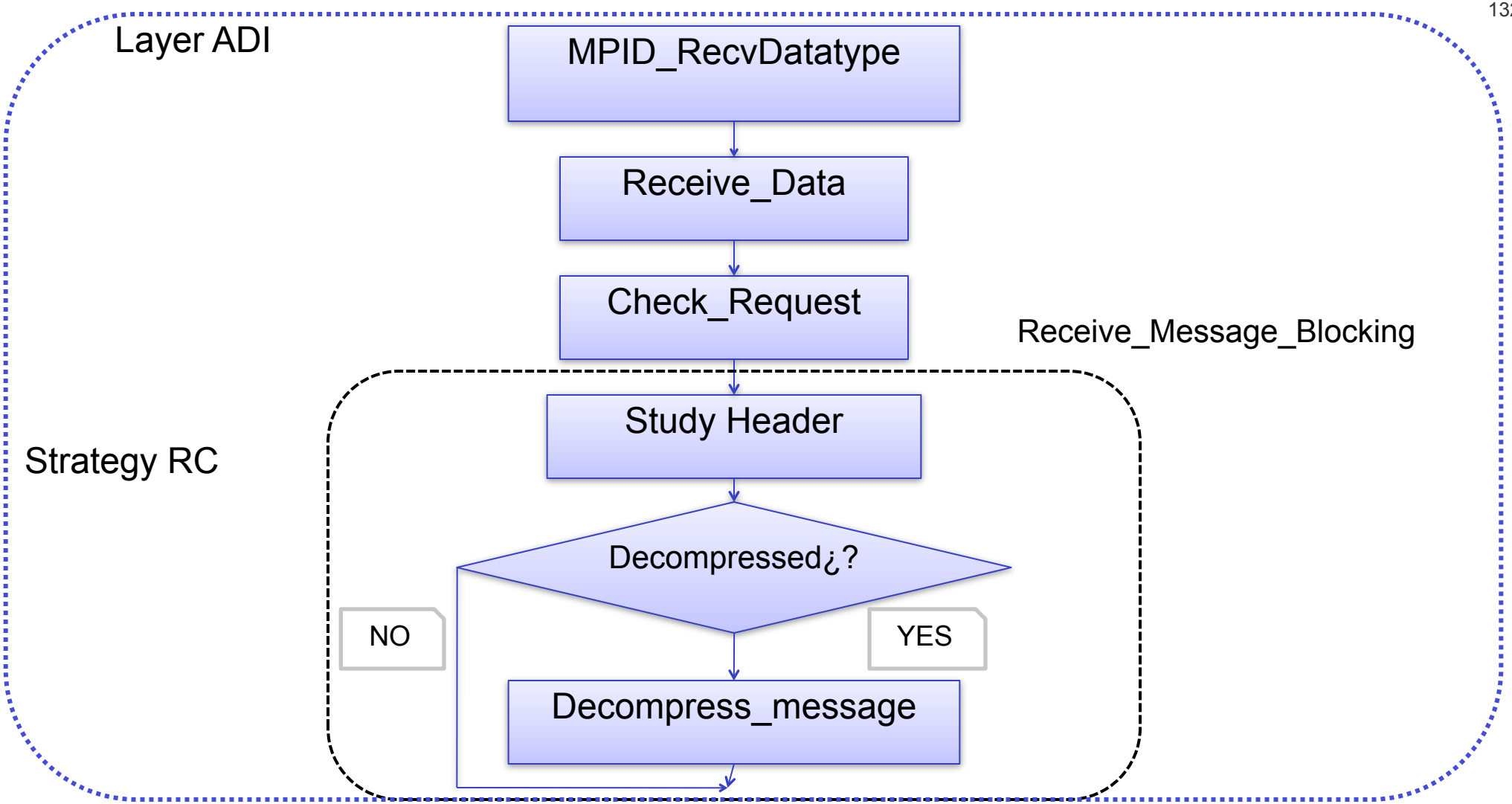
Receive message blocking (RC)

Layer ADI

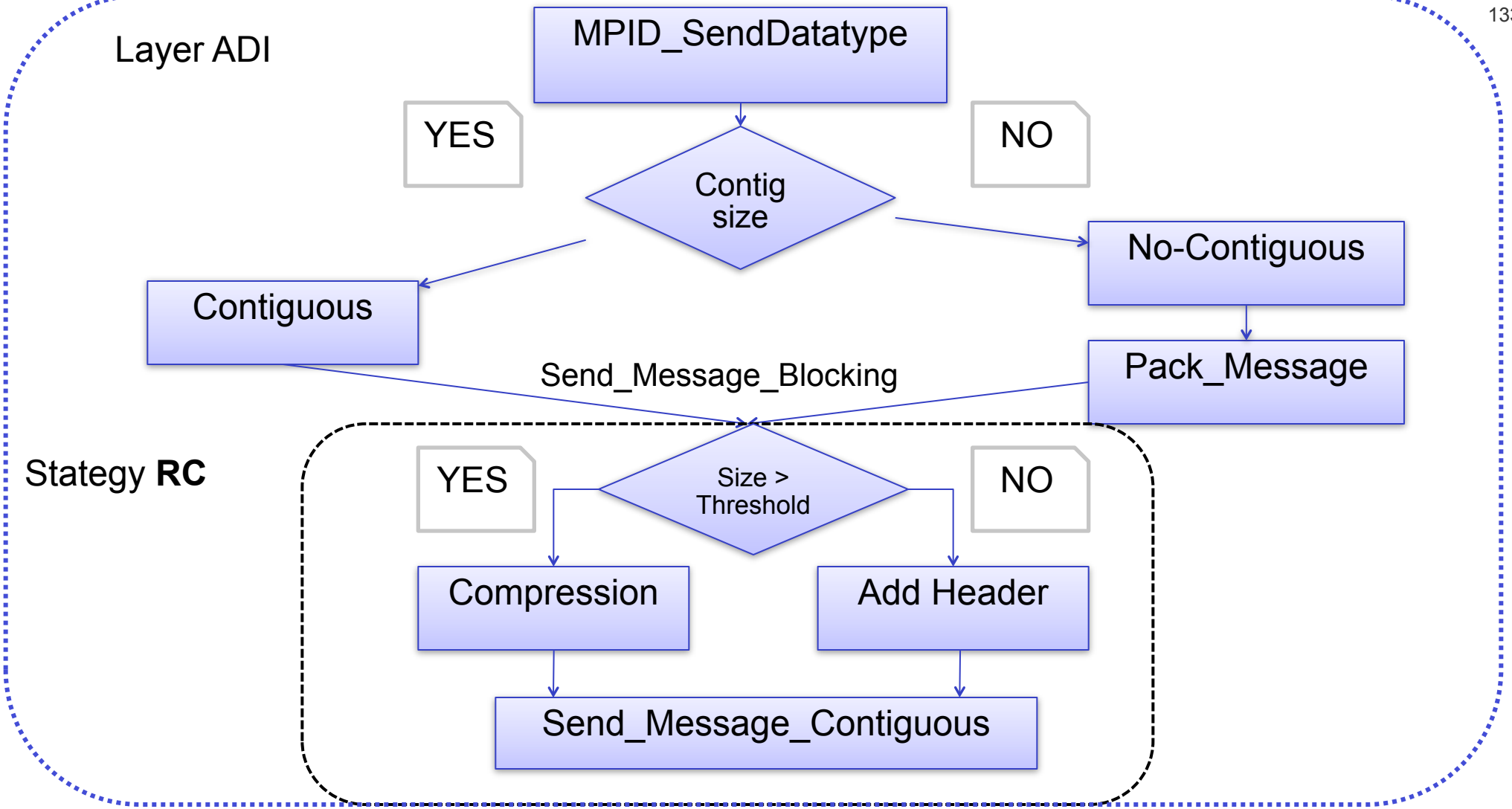


Stratey RC

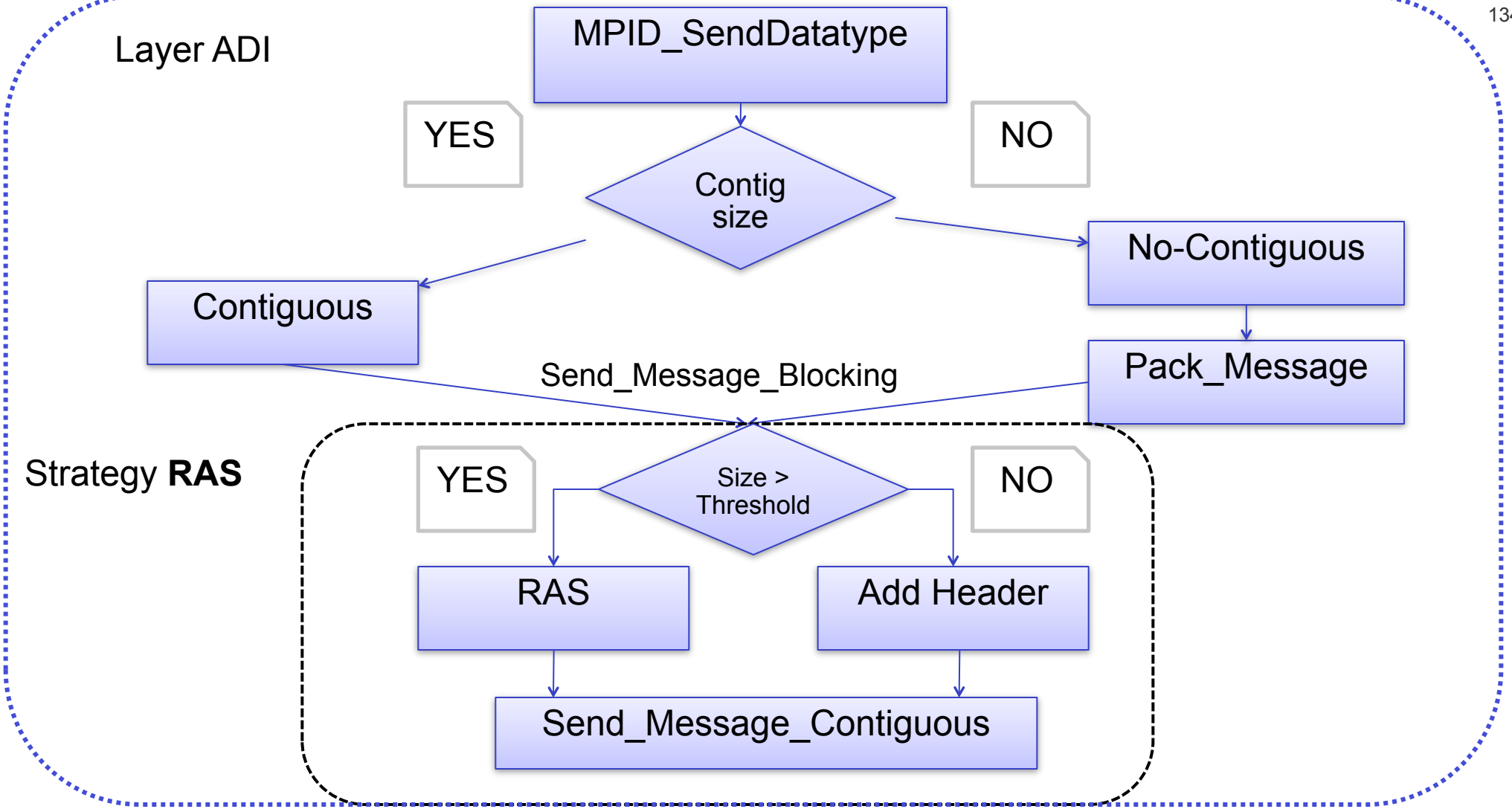
Receive message blocking (MPICH)



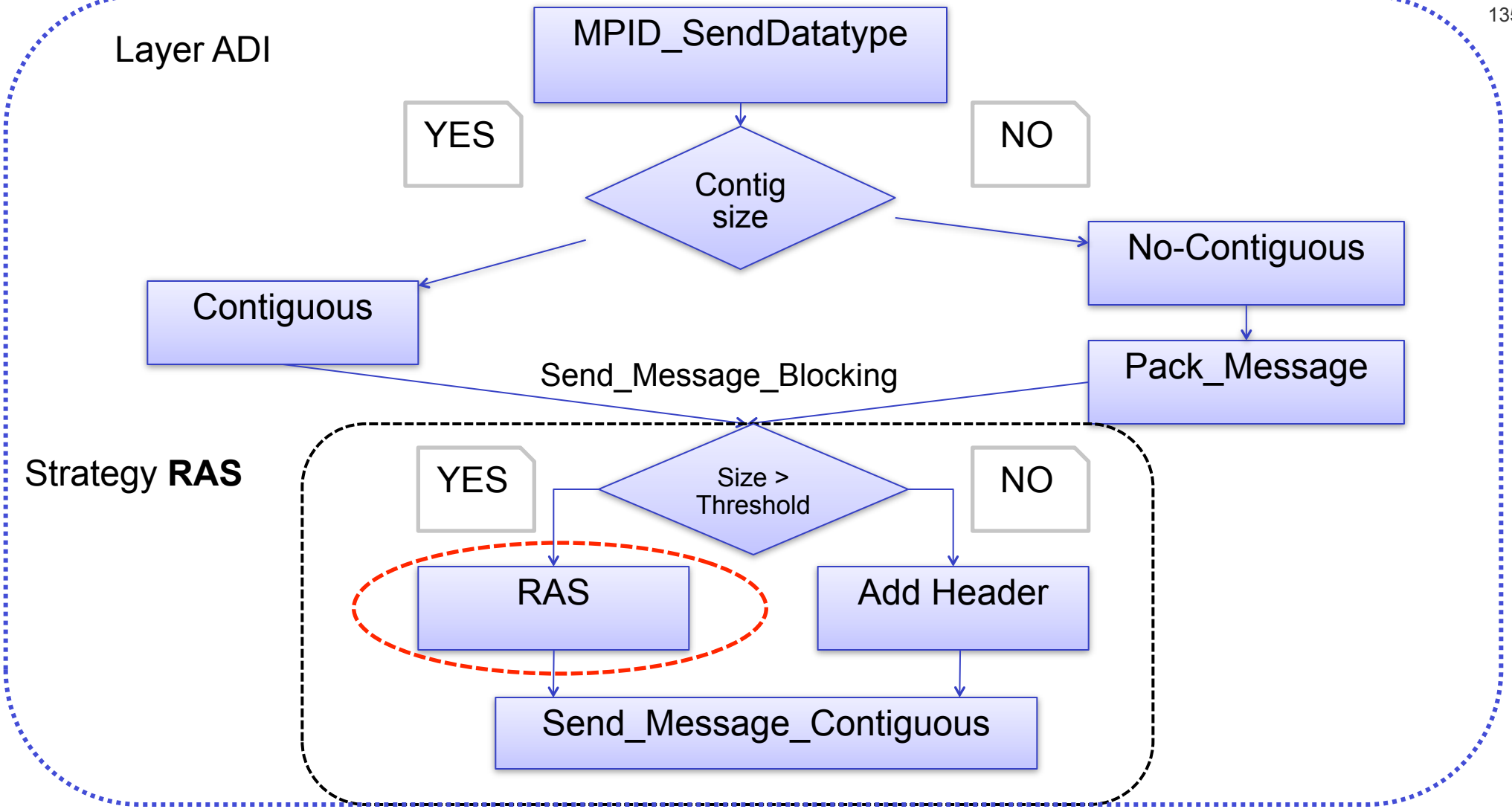
Send_Message_Blocking
Runtime Compression



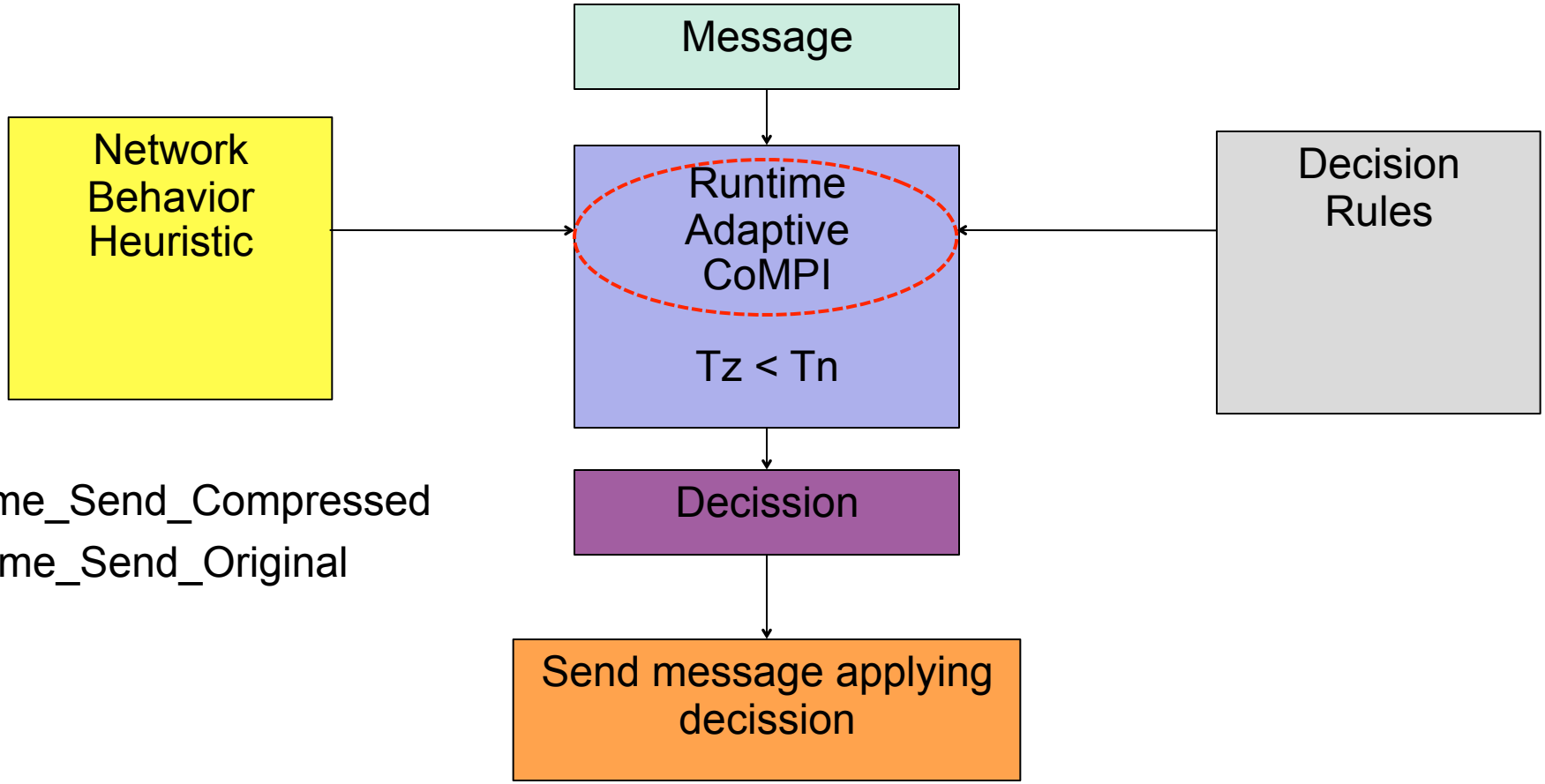
Send_Message_Blocking
Runtime Adaptive Compression



Send_Message_Blocking
Runtime Adaptive Compression



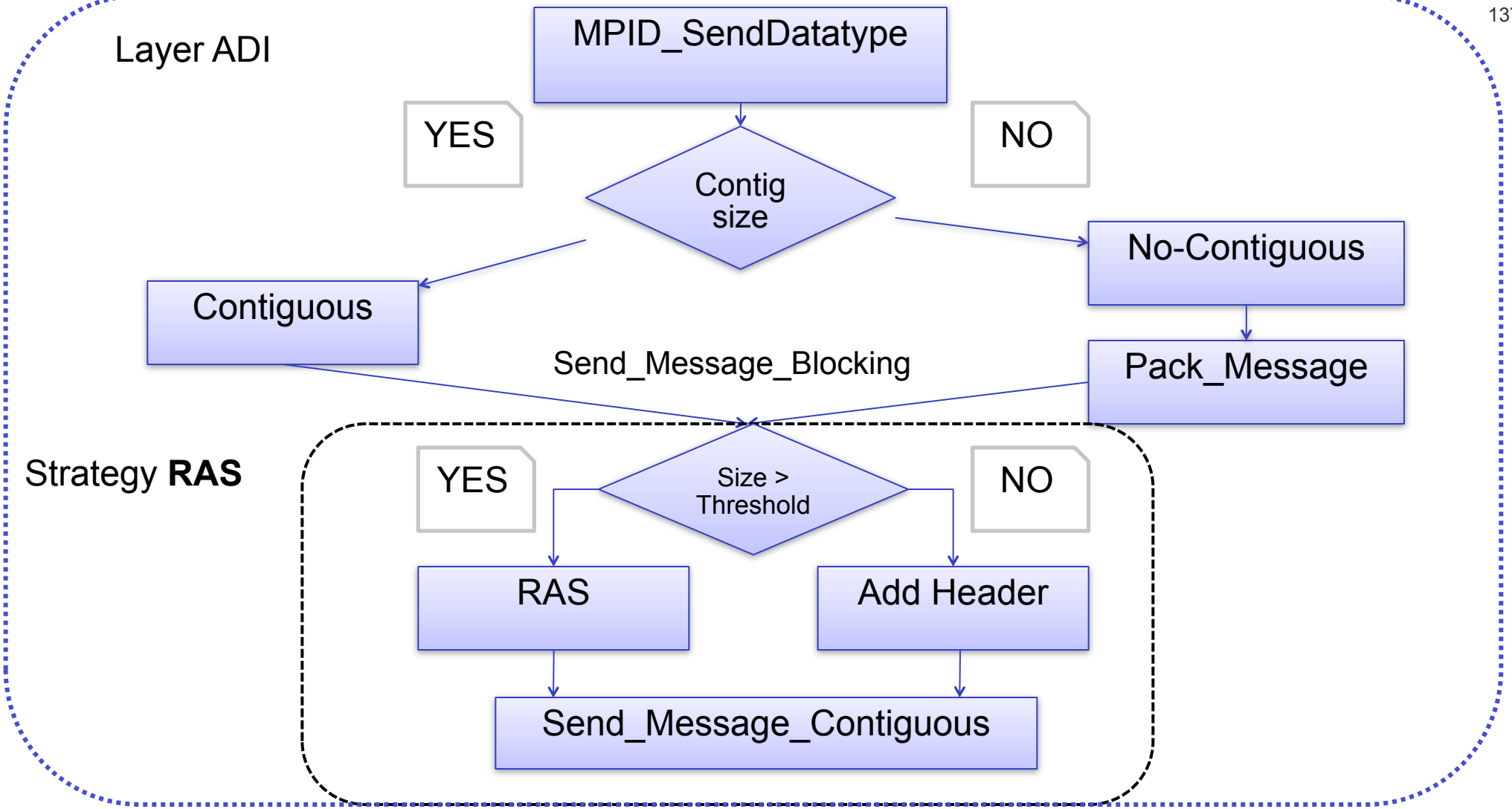
Guided Strategy



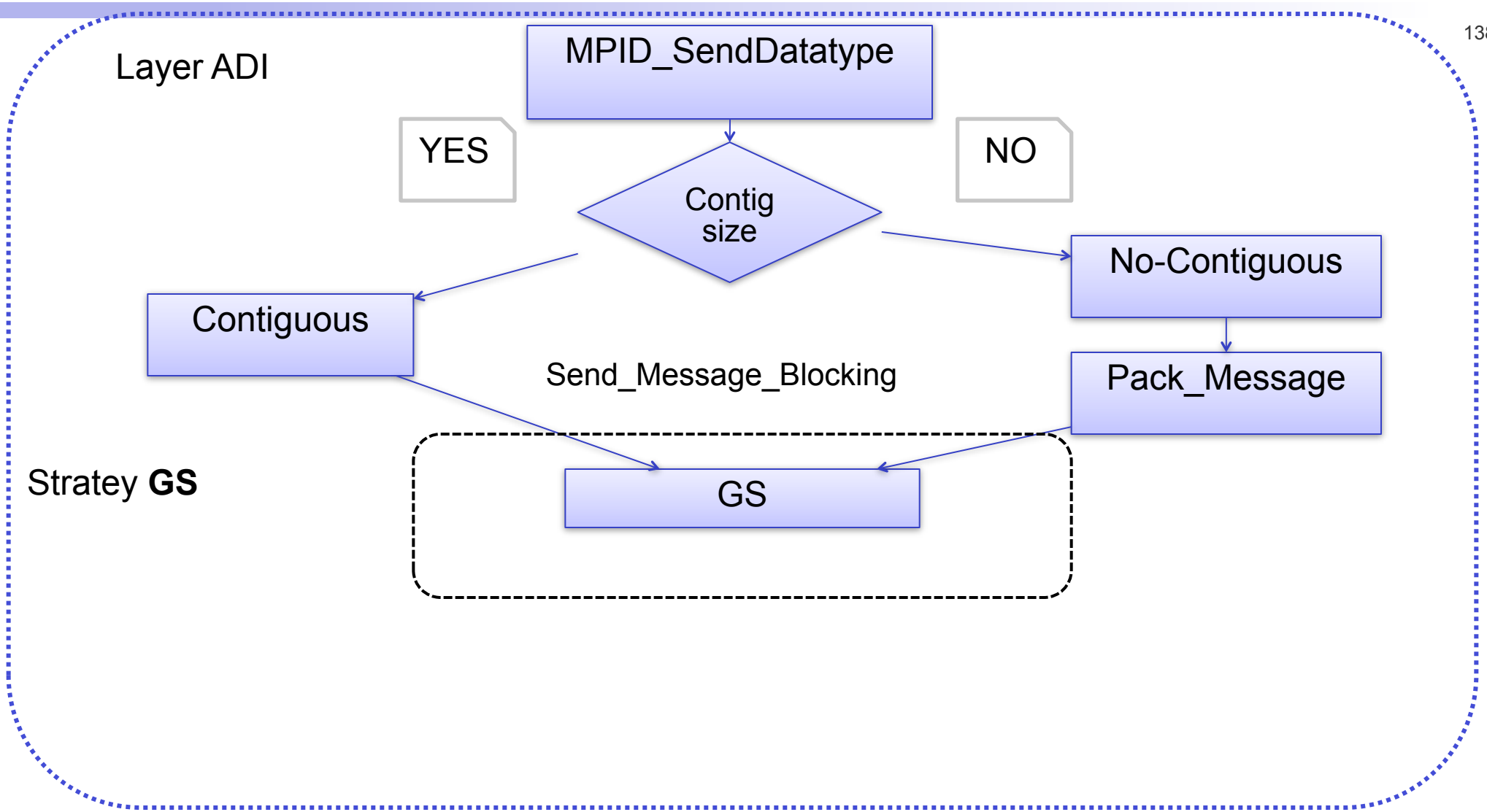
Tz=Time_Send_Compressed

Tn=Time_Send_Original

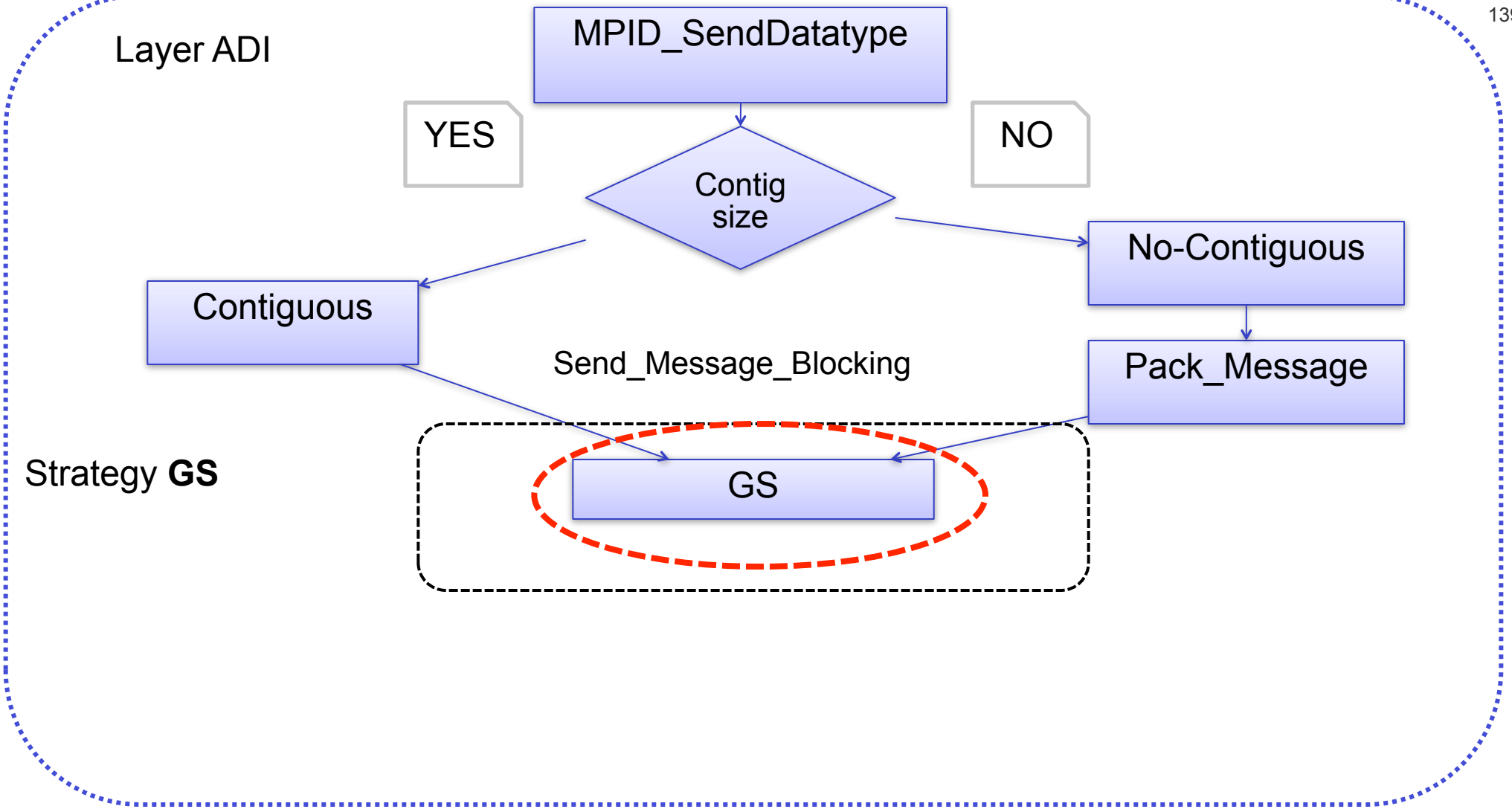
Send_Message_Blocking
Runtime Adaptive Compression



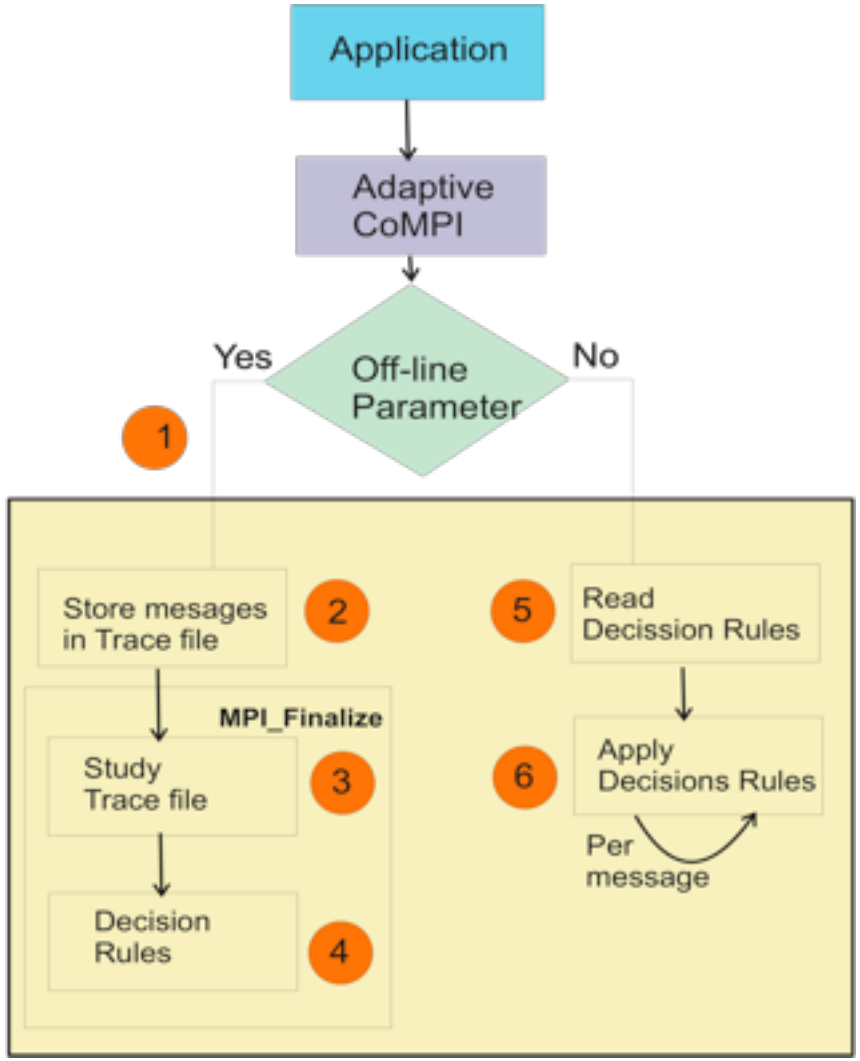
Send_Message_Blocking
Guided Strategy



Send_Message_Blocking
Guided Strategy

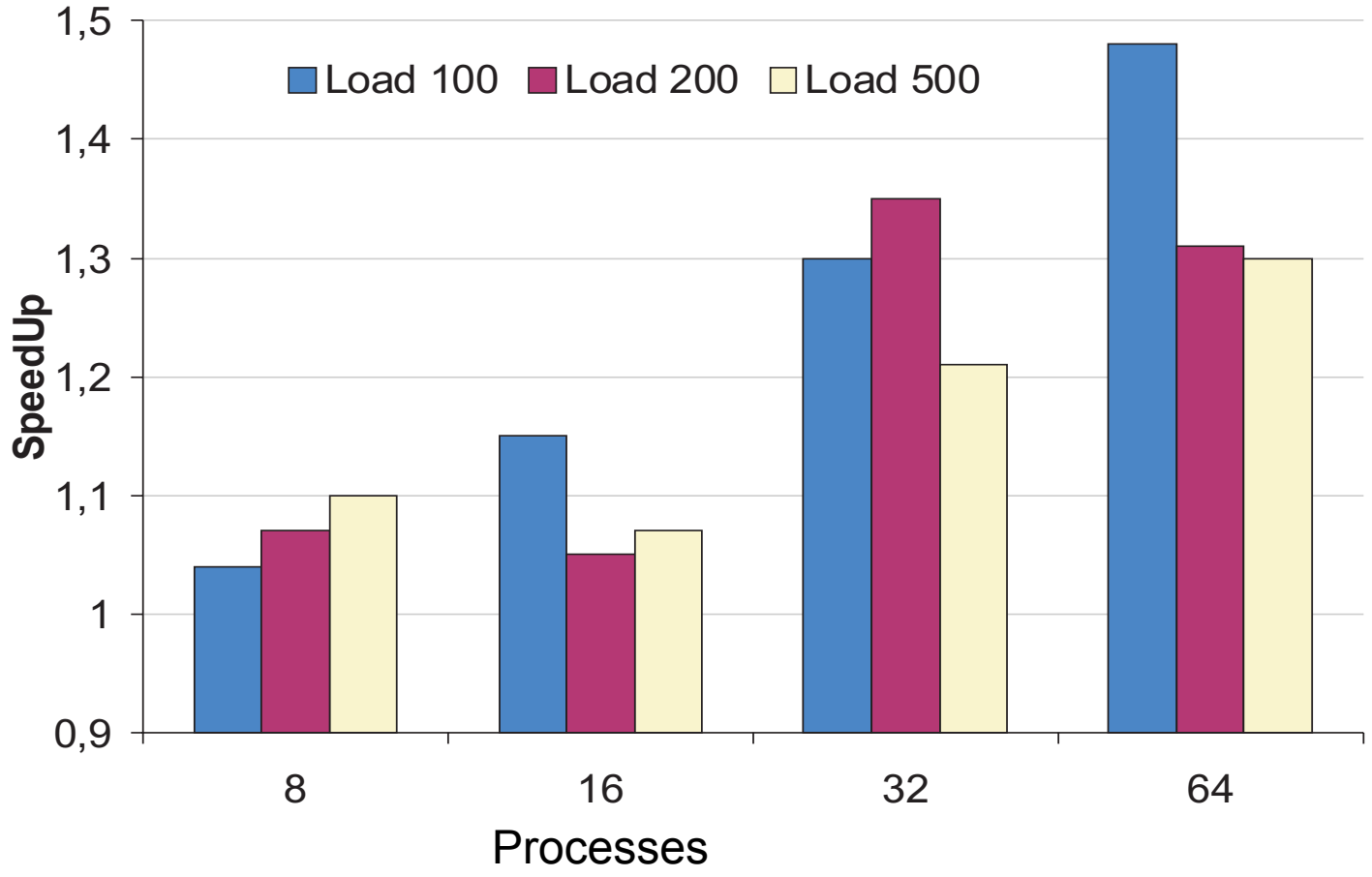


Guided Strategy



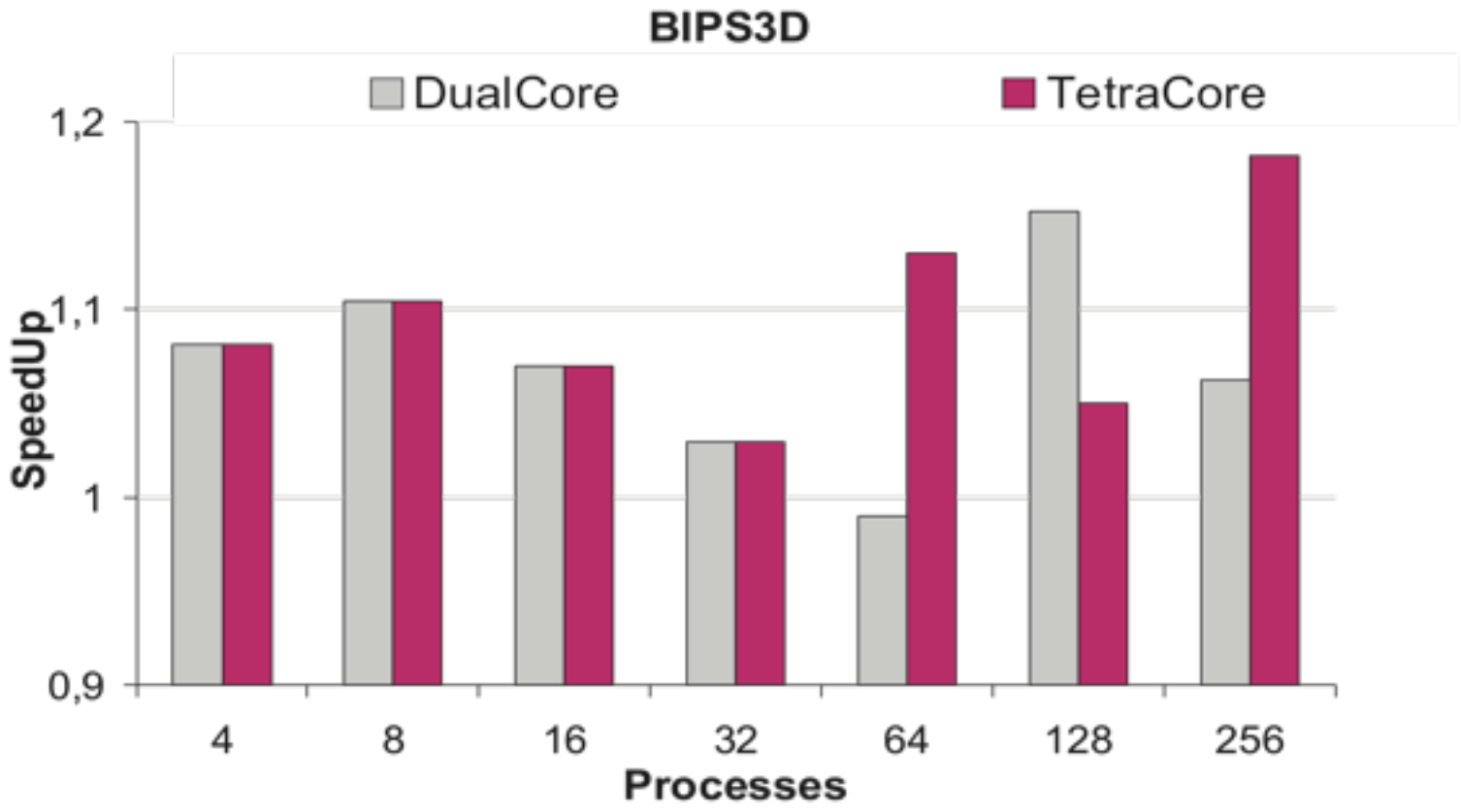
Evaluation
LA_TwoPhase I/O BISP3D with cluster A

Mesh 1



Evaluation LA_TwoPhase I/O BIPS3D with cluster B and cluster C

Mesh 4



RAS desactiva compresión cuando no obtiene beneficio (NAS).

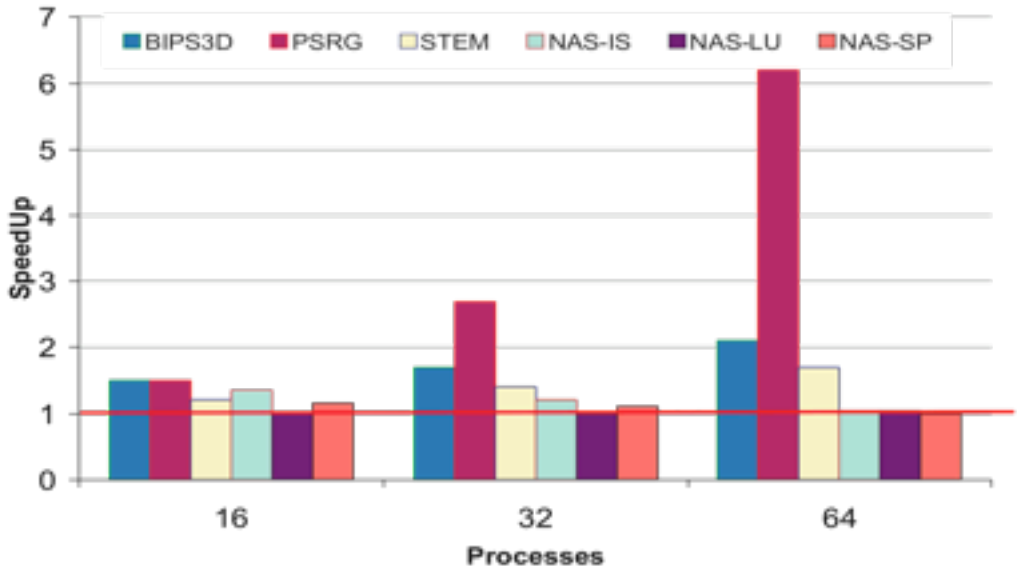
RAS elige el mejor algoritmo por mensaje.

Ejemplo PRSG utiliza LZO y RLE para comprimir en función de características de mensajes.

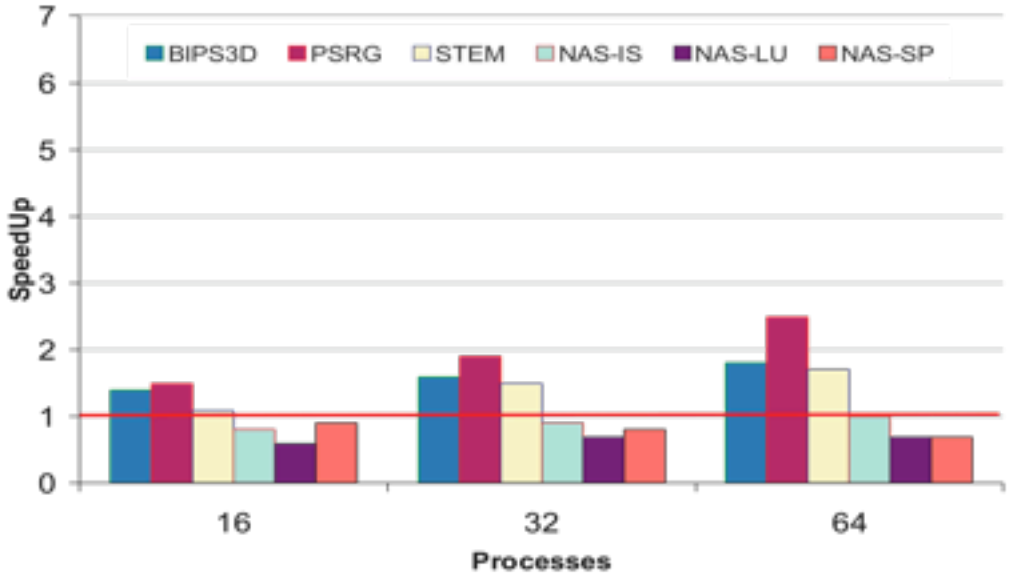
Mejor dejar tomar decisiones al algoritmo.

RAS no penaliza

Adaptive Compression Strategy (RAS)

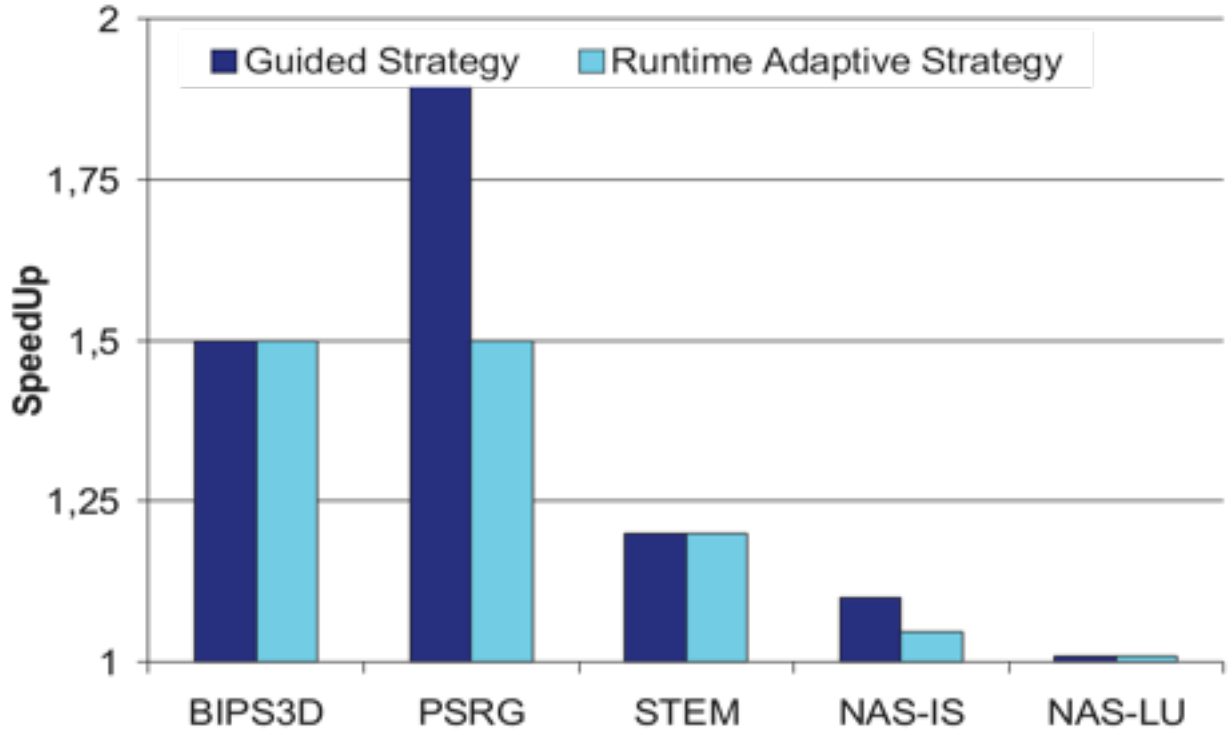


NO adaptive (RC)



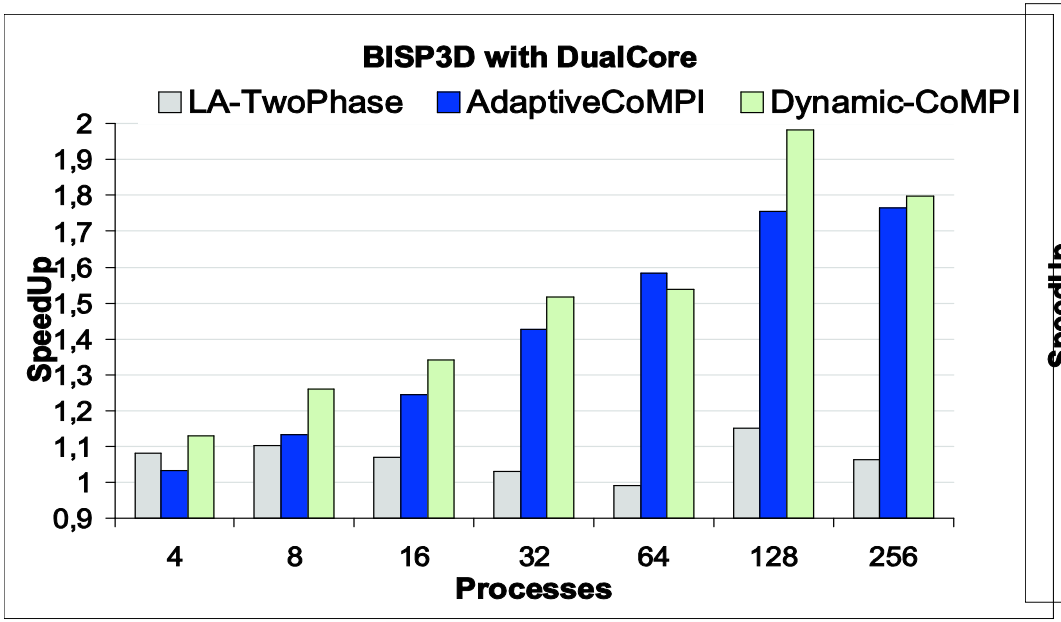
Guided Strategy (GS)

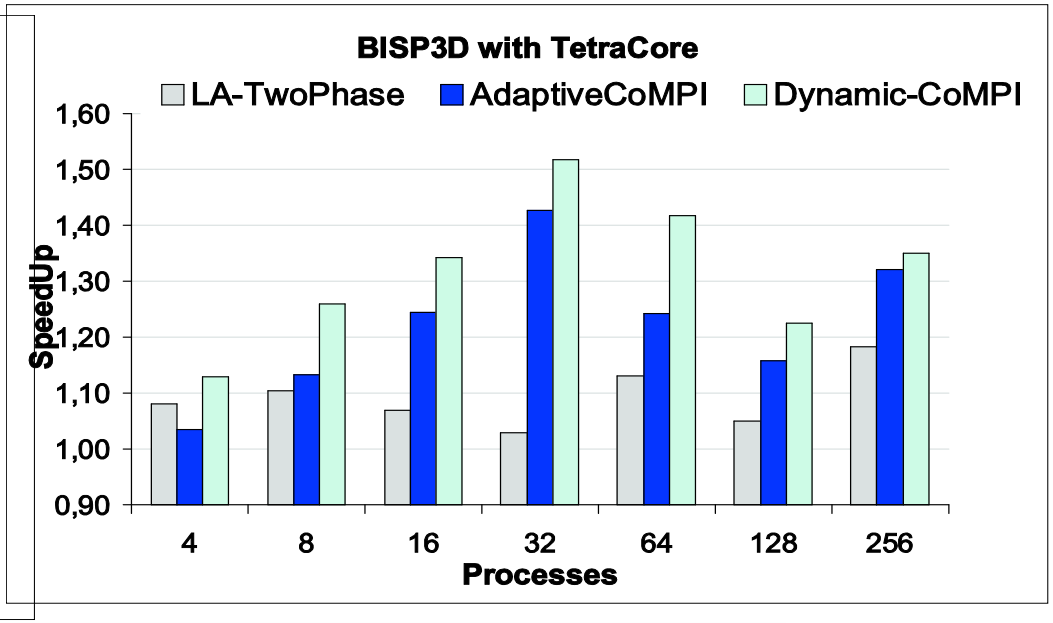
16 processes



RAS es tan buena estrategia como GS, ya que obtienen resultados similares.

Comparación de técnicas con BISP3D





-Speedup LA-TwoPhase mayor en TetraCore→ Mayor impacto el patrón de agregadores.

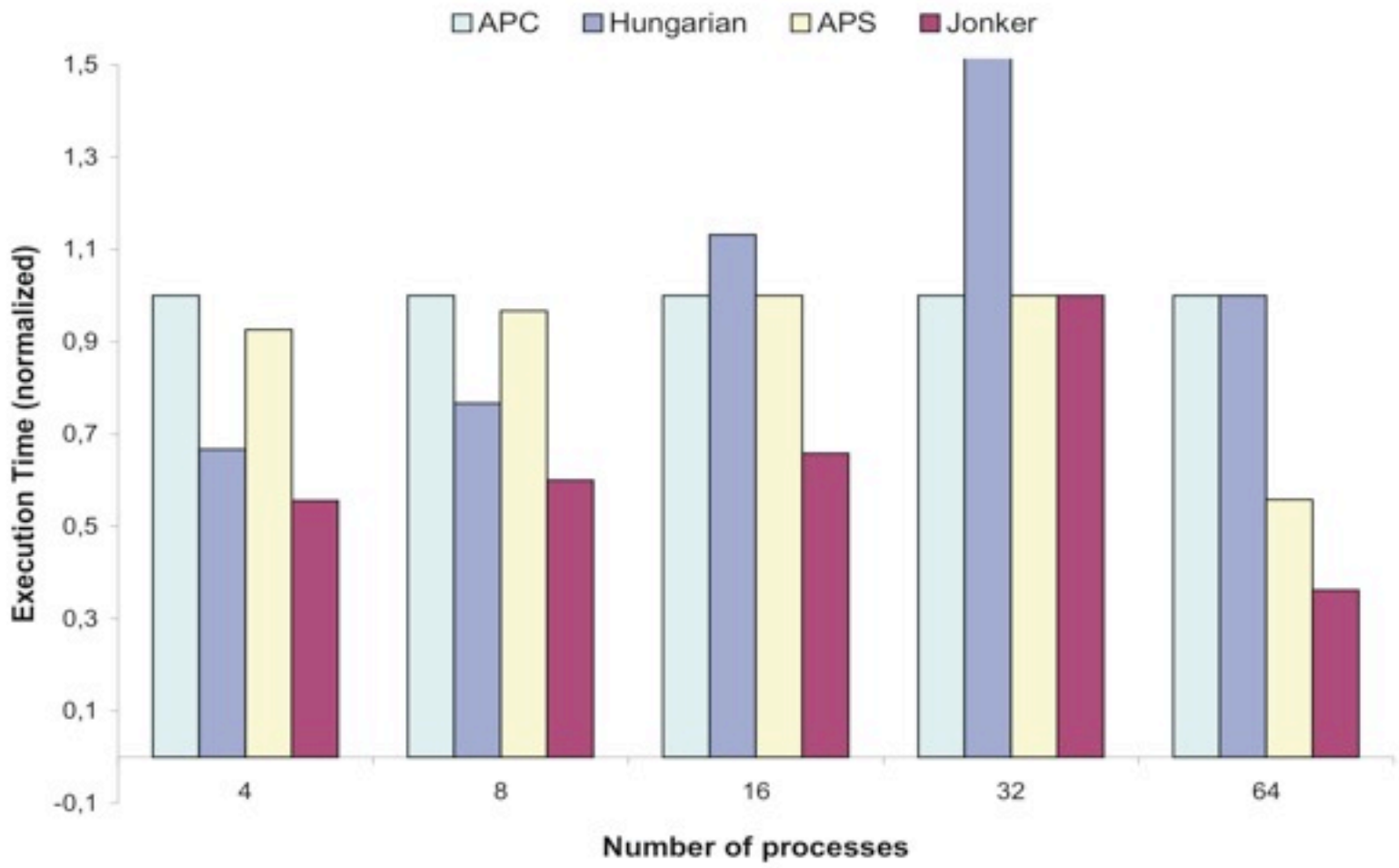
-Speedup Adaptive-CoMPI mayor en DualCore que TetraCore → Se mandan más mensajes comprimidos.

-Dynamic-CoMPI Speedup= LA-TwoPhase + Adaptive-CoMPI → Son complementarias ambas técnicas.

Linear Assigment 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)



3.1 Linear Assigment Problem (II)