


Comparing archival policies for BlueWaters

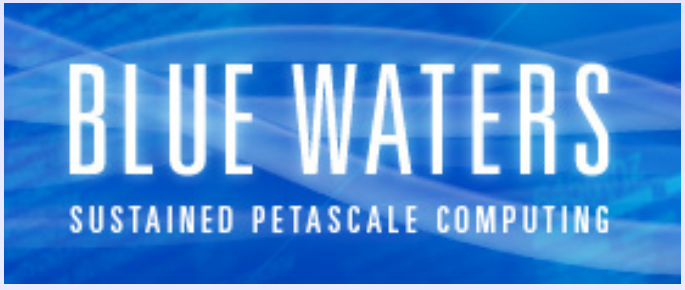
Franck Cappello, Mathias Jacquelin, Loris Marchal,
Yves Robert and Marc Snir

INRIA – CNRS – École Normale Supérieure de Lyon, France
NCSA – University of Illinois at Urbana–Champaign

4th Workshop of the UIUC-INRIA
Joint Laboratory for Petascale Computing,
Urbana, November 22, 2010.

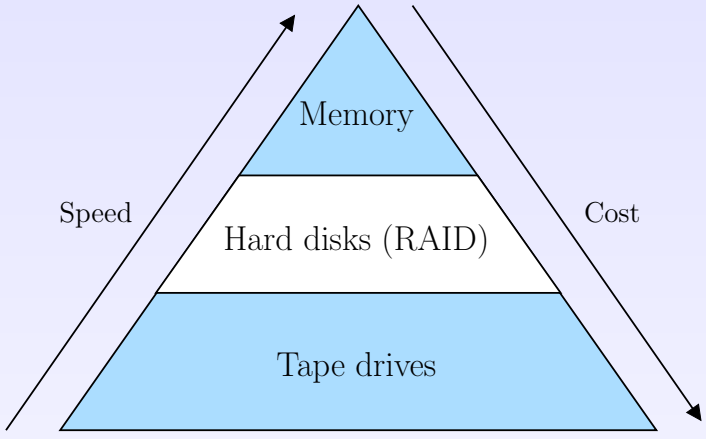
Outline

Motivation: BlueWaters



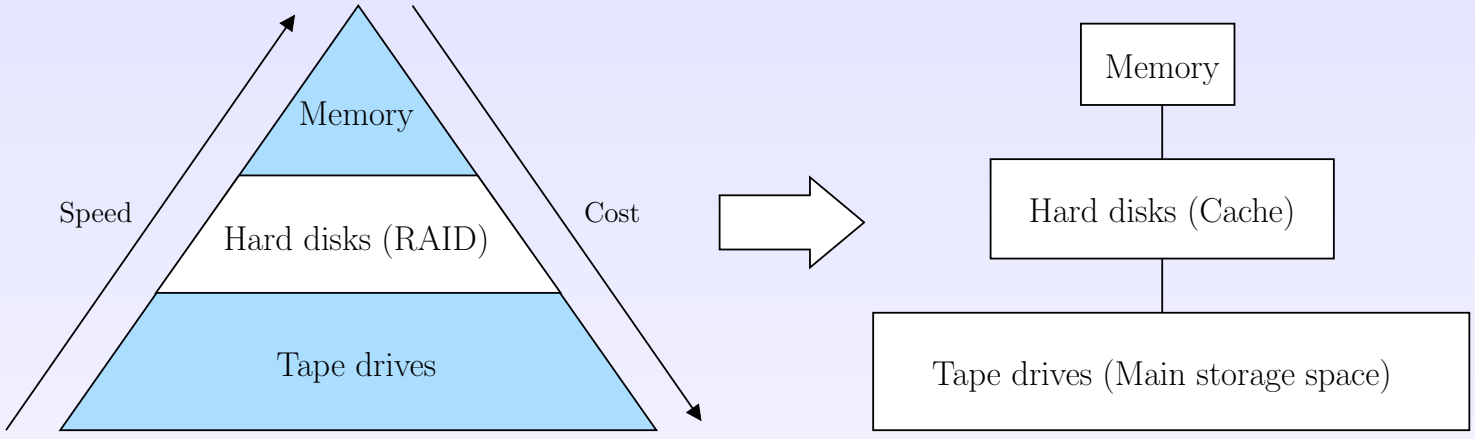
- Sustained Petaflops/s for general applications
- Hierarchical storage system
  - Hard disk drives used as "cache"
  - Tape drives used as actual permanent storage media
- Main Objective: Design an efficient disk management policy
- Today: Writing data to tapes efficiently

Motivation: BlueWaters



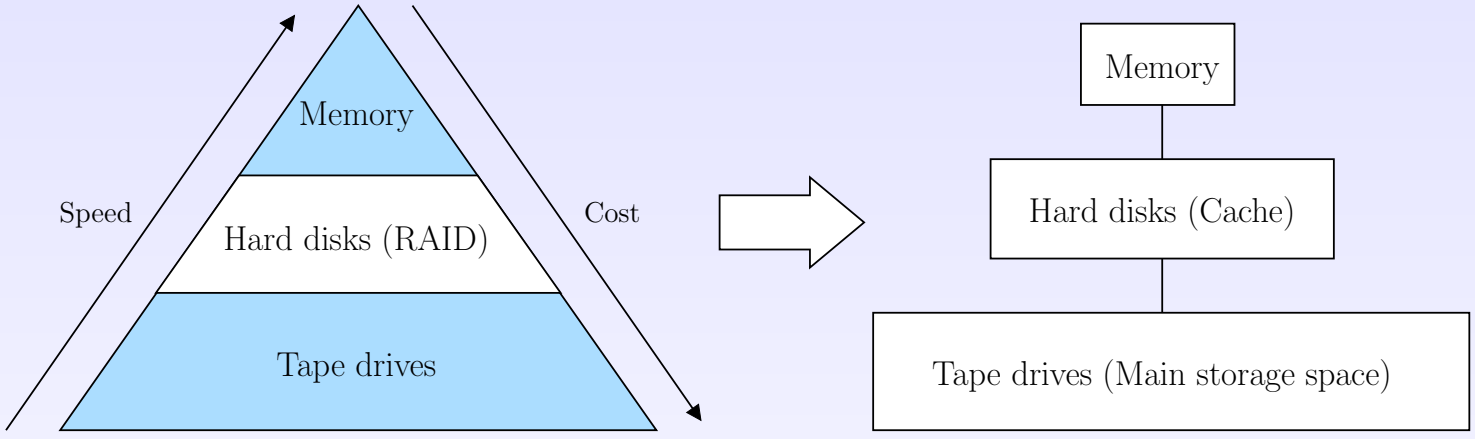
- Sustained Petaflops/s for general applications
- Hierarchical storage system
  - Hard disk drives used as ''cache''
  - Tape drives used as actual permanent storage media
- Main Objective: Design an efficient disk management policy
- Today: Writing data to tapes efficiently

Motivation: BlueWaters



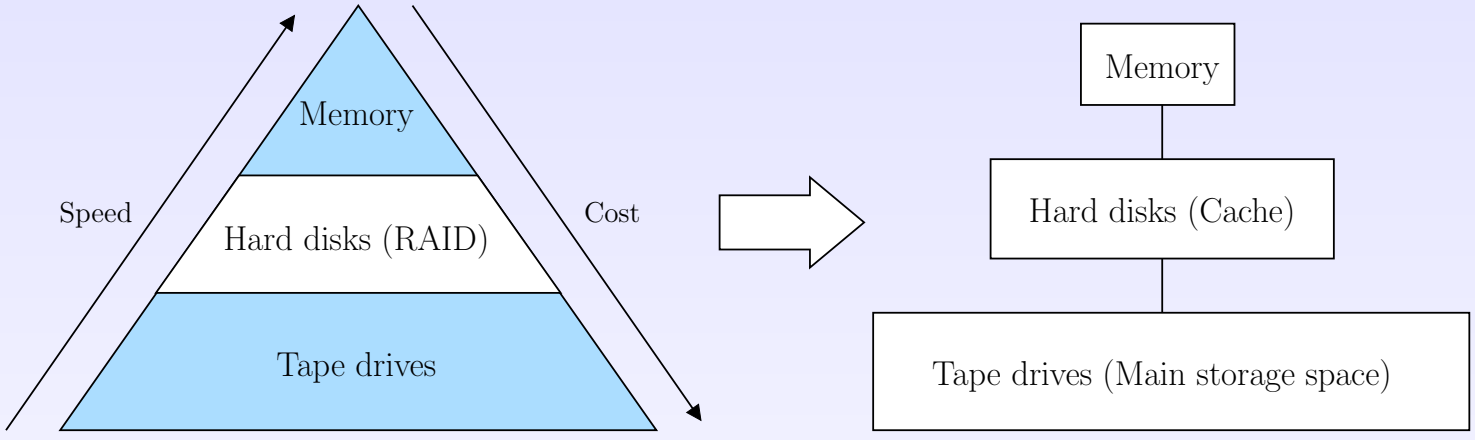
- Sustained Petaflops/s for general applications
- Hierarchical storage system
  - Hard disk drives used as ''cache''
  - Tape drives used as actual permanent storage media
- Main Objective: Design an efficient disk management policy
- Today: Writing data to tapes efficiently

Motivation: BlueWaters



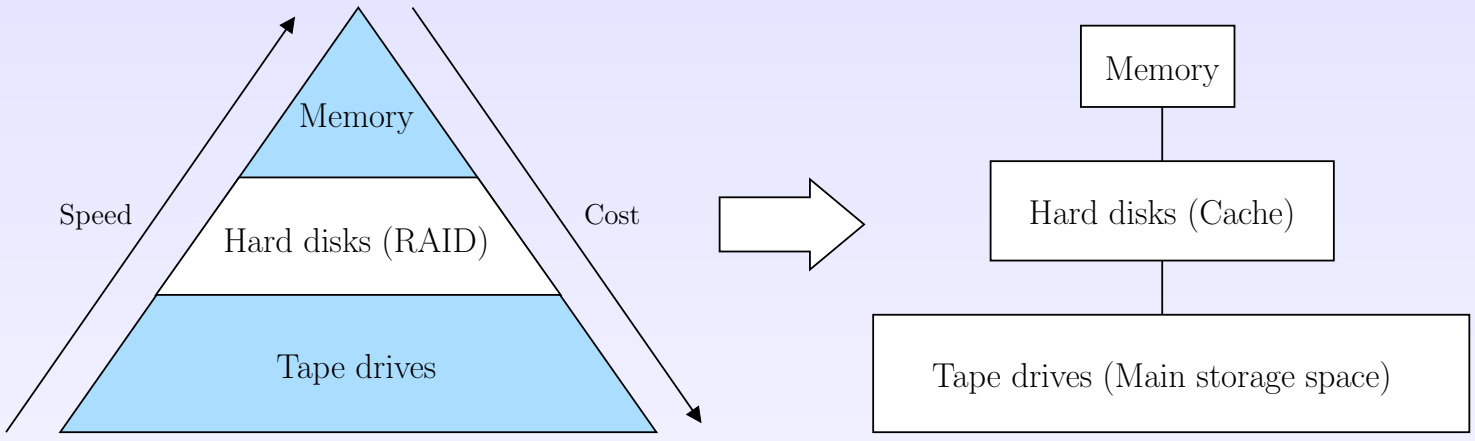
- Sustained Petaflops/s for general applications
- Hierarchical storage system
  - Hard disk drives used as ''cache''
  - Tape drives used as actual permanent storage media
- Main Objective: Design an efficient disk management policy
- Today: Writing data to tapes efficiently

Motivation: BlueWaters



- Sustained Petaflops/s for general applications
- Hierarchical storage system
  - Hard disk drives used as ''cache''
  - Tape drives used as actual permanent storage media
- Main Objective: Design an efficient disk management policy
- Today: Writing data to tapes efficiently

Motivation: BlueWaters



- Sustained Petaflops/s for general applications
- Hierarchical storage system
  - Hard disk drives used as ''cache''
  - Tape drives used as actual permanent storage media
- Main Objective: Design an efficient disk management policy
- Today: Writing data to tapes efficiently

Why new tape access policies?

Main issue

Entire storage space is seamlessly exposed to the user

Why new tape access policies?

Main issue

Entire storage space is seamlessly exposed to the user

Today :

Tape I/O Management

From disk to tape $\stackrel{?}{=}$ from memory to disk
Write resilient data efficiently on the parallel tape storage architecture of BlueWaters

Outline

Tape Related Hardware

- Tapes [≈ 500000]
  - Serpentine tapes
  - Each tape stores up to 1TB of uncompressed data
- Tape Drives [≈ 500]
- Tape Libraries [≈ 3]
  - Manage the tapes
  - Each library has robotic arms moving tapes to/from tape drives
  - Passthrough to transfer tapes between different libraries
- Mover Nodes [≈ 50]
  - 24 cores, 96 GB of RAM
  - Connected to up to 10 tape drives
  - Compute parity
  - Forward I/O requests to tape drives

Tape Related Hardware

- Tapes [≈ 500000]
  - Serpentine tapes
  - Each tape stores up to 1TB of uncompressed data
- Tape Drives [≈ 500]
- Tape Libraries [≈ 3]
  - Manage the tapes
  - Each library has robotic arms moving tapes to/from tape drives
  - Passthrough to transfer tapes between different libraries
- Mover Nodes [≈ 50]
  - 24 cores, 96 GB of RAM
  - Connected to up to 10 tape drives
  - Compute parity
  - Forward I/O requests to tape drives

Tape Related Hardware

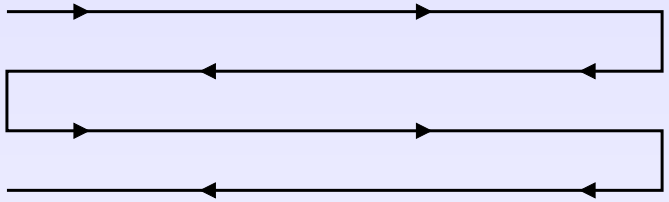
- Tapes [$\approx$ 500000]
  - Serpentine tapes
  - Each tape stores up to 1TB of uncompressed data
- Tape Drives [$\approx$ 500]
- Tape Libraries [$\approx$ 3]
  - Manage the tapes
  - Each library has robotic arms moving tapes to/from tape drives
  - Passthrough to transfer tapes between different libraries
- Mover Nodes [$\approx$ 50]
  - 24 cores, 96 GB of RAM
  - Connected to up to 10 tape drives
  - Compute parity
  - Forward I/O requests to tape drives

Tape Related Hardware

- Tapes [$\approx$ 500000]
  - Serpentine tapes
  - Each tape stores up to 1TB of uncompressed data
- Tape Drives [$\approx$ 500]
- Tape Libraries [$\approx$ 3]
  - Manage the tapes
  - Each library has robotic arms moving tapes to/from tape drives
  - Passthrough to transfer tapes between different libraries
- Mover Nodes [$\approx$ 50]
  - 24 cores, 96 GB of RAM
  - Connected to up to 10 tape drives
  - Compute parity
  - Forward I/O requests to tape drives

Characteristics of tape drives

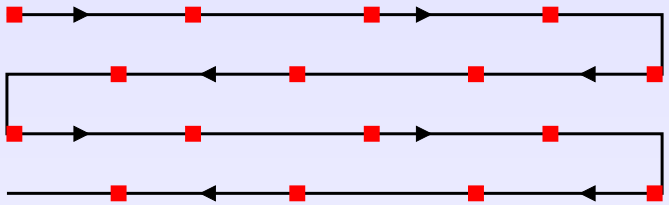
- Serpentine tapes



  - Regular and anti-directional tracks
  - Head can only be positioned on key points
  - No direct relationship between logical addresses and physical positions on tape
- Hardware compression: up to 3x
- Sequential accesses
- Random accesses

Characteristics of tape drives

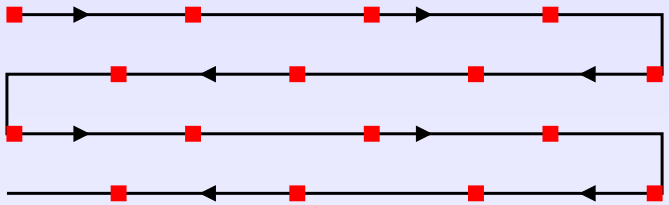
- Serpentine tapes



  - Regular and anti-directional tracks
  - Head can only be positioned on key points
  - No direct relationship between logical addresses and physical positions on tape
- Hardware compression: up to 3x
- Sequential accesses
- Random accesses

Characteristics of tape drives

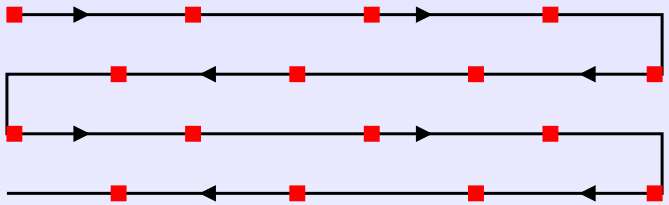
- Serpentine tapes



  - Regular and anti-directional tracks
  - Head can only be positioned on key points
  - No direct relationship between logical addresses and physical positions on tape
- Hardware compression: up to 3x
- Sequential accesses
- Random accesses

Characteristics of tape drives

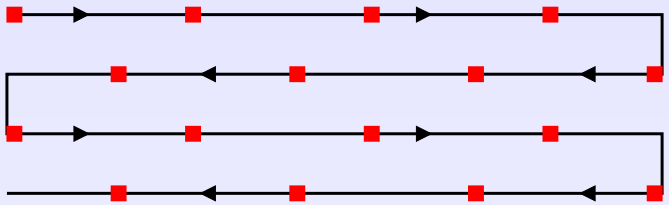
- Serpentine tapes



  - Regular and anti-directional tracks
  - Head can only be positioned on key points
  - No direct relationship between logical addresses and physical positions on tape
- Hardware compression: up to 3x
- Sequential accesses
- Random accesses

Characteristics of tape drives

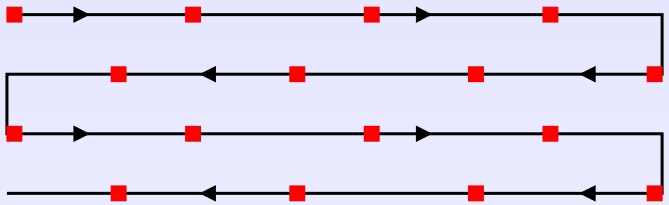
- Serpentine tapes



  - Regular and anti-directional tracks
  - Head can only be positioned on key points
  - No direct relationship between logical addresses and physical positions on tape
- Hardware compression: up to 3x
- Sequential accesses
- Random accesses

Characteristics of tape drives

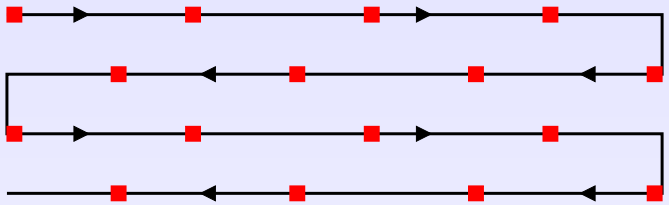
- Serpentine tapes



  - Regular and anti-directional tracks
  - Head can only be positioned on key points
  - No direct relationship between logical addresses and physical positions on tape
- Hardware compression: up to 3x
- Sequential accesses ☺
  - Up to 160 MB/s of uncompressed data
- Random accesses

Characteristics of tape drives

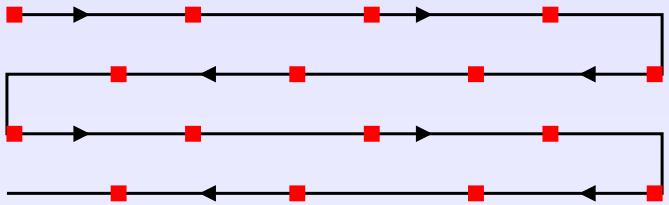
- Serpentine tapes



  - Regular and anti-directional tracks
  - Head can only be positioned on key points
  - No direct relationship between logical addresses and physical positions on tape
- Hardware compression: up to 3x
- Sequential accesses ☺
  - Up to 160 MB/s of uncompressed data
- Random accesses

Characteristics of tape drives

- Serpentine tapes



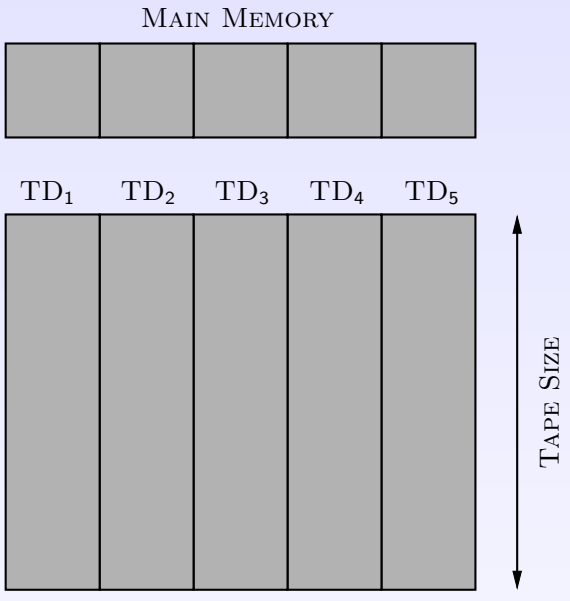
  - Regular and anti-directional tracks
  - Head can only be positioned on key points
  - No direct relationship between logical addresses and physical positions on tape
- Hardware compression: up to 3x
- Sequential accesses ☺
  - Up to 160 MB/s of uncompressed data
- Random accesses ☹
  - High latencies: average latency is 36.5s

Outline

Request model

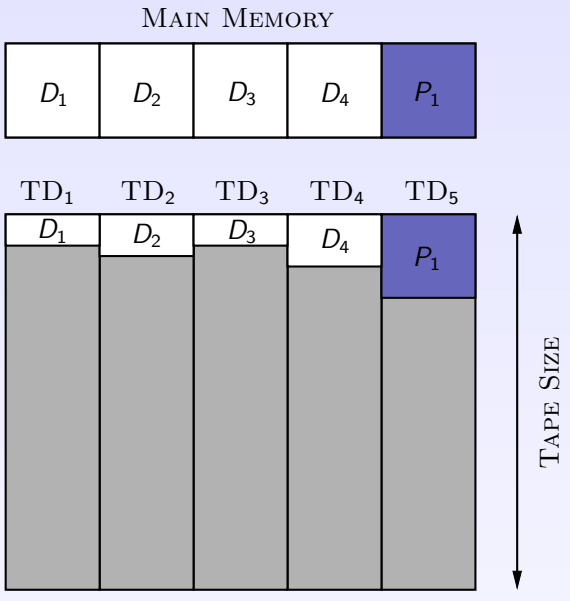
- Requests could be sent by:
  - Several users for a classical archival system
  - The disk management system, or the batch scheduler
- An I/O request is defined by:
  - An associated file
  - Its size
  - An I/O policy
  - The resiliency scheme X+Y where:
    - X denotes the number of data blocks
    - Y denotes the number of parity blocks
    - For instance : 4+1, 4+2, 8+2...

The RAIT policy



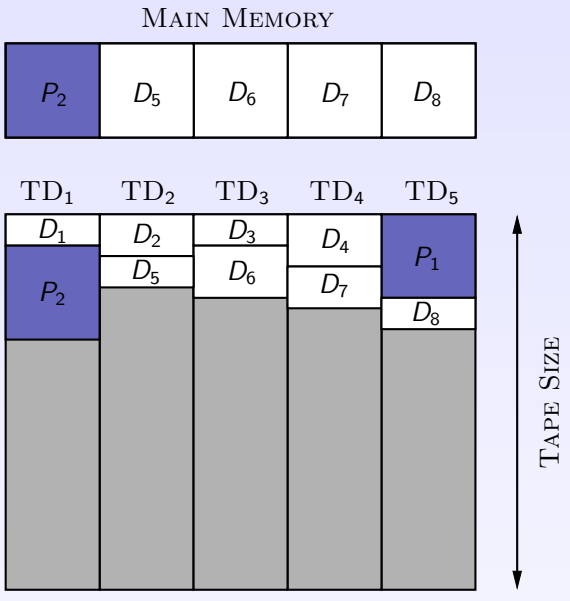
- Redundant Array of Independent Tapes, inspired by RAID
- Rationale: Writing data in parallel while balancing tape usage

The RAIT policy



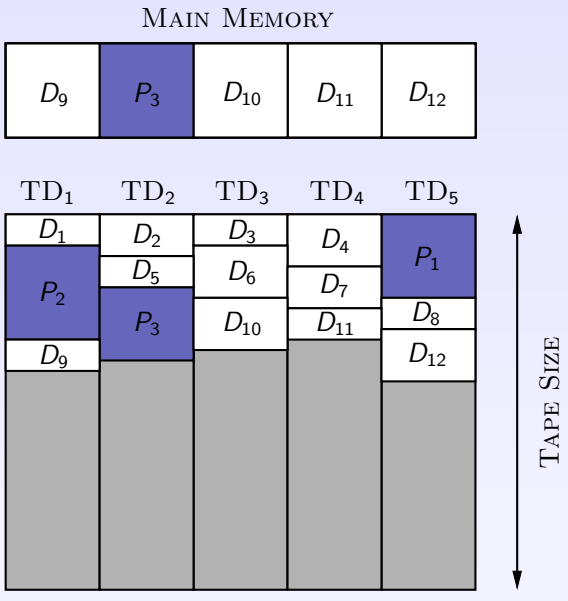
- Redundant Array of Independent Tapes, inspired by RAID
- Rationale: Writing data in parallel while balancing tape usage

The RAIT policy



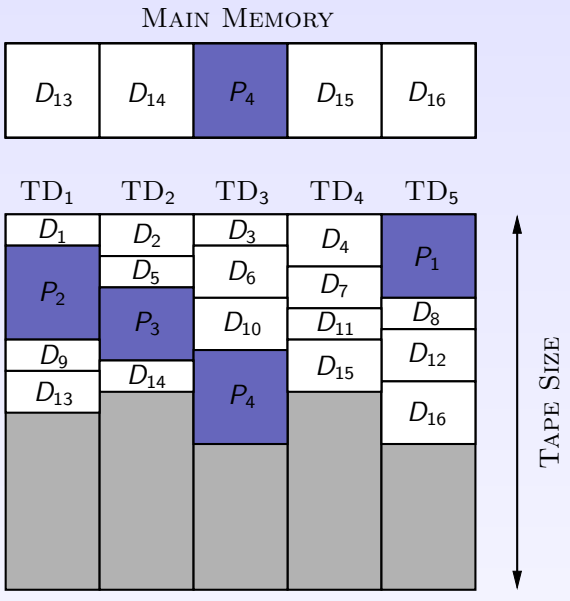
- Redundant Array of Independent Tapes, inspired by RAID
- Rationale: Writing data in parallel while balancing tape usage

The RAIT policy



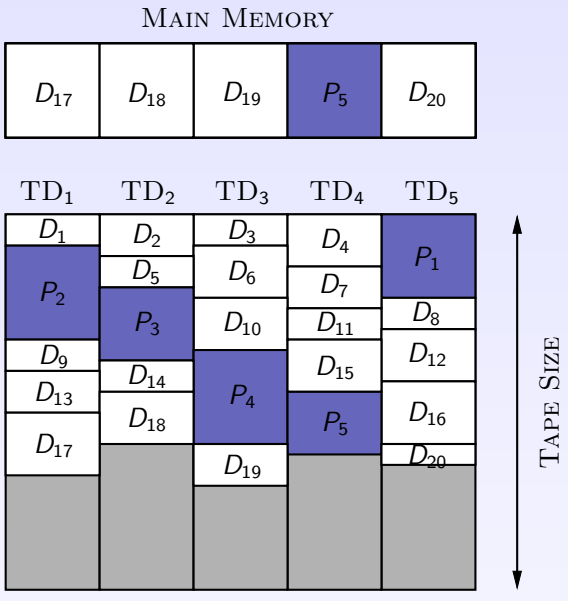
- Redundant Array of Independent Tapes, inspired by RAID
- Rationale: Writing data in parallel while balancing tape usage

The RAIT policy



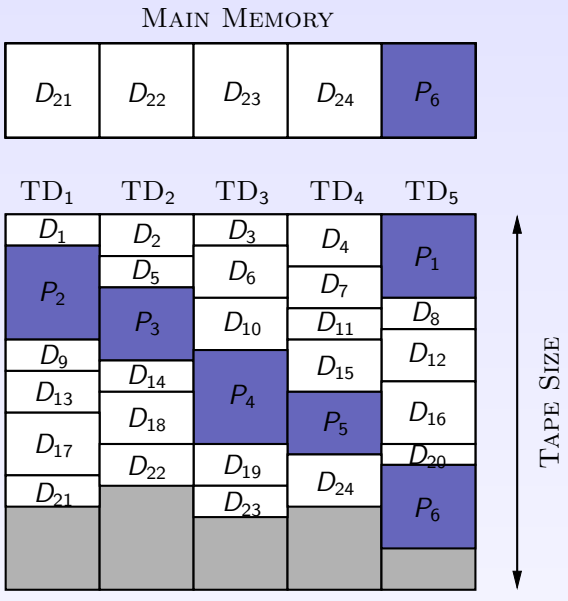
- Redundant Array of Independent Tapes, inspired by RAID
- Rationale: Writing data in parallel while balancing tape usage

The RAIT policy



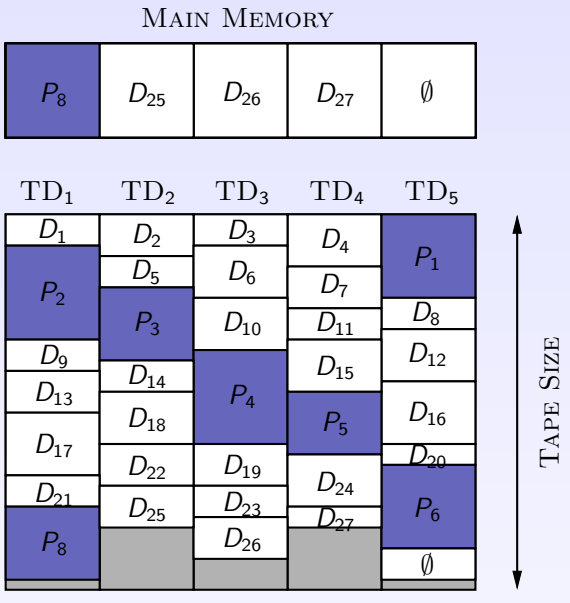
- Redundant Array of Independent Tapes, inspired by RAID
- Rationale: Writing data in parallel while balancing tape usage

The RAIT policy



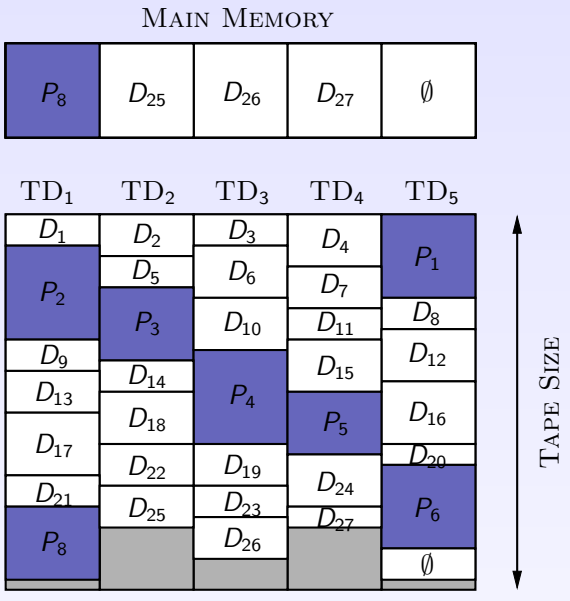
- Redundant Array of Independent Tapes, inspired by RAID
- Rationale: Writing data in parallel while balancing tape usage

The RAIT policy



- Redundant Array of Independent Tapes, inspired by RAID
- Rationale: Writing data in parallel while balancing tape usage

The RAIT policy



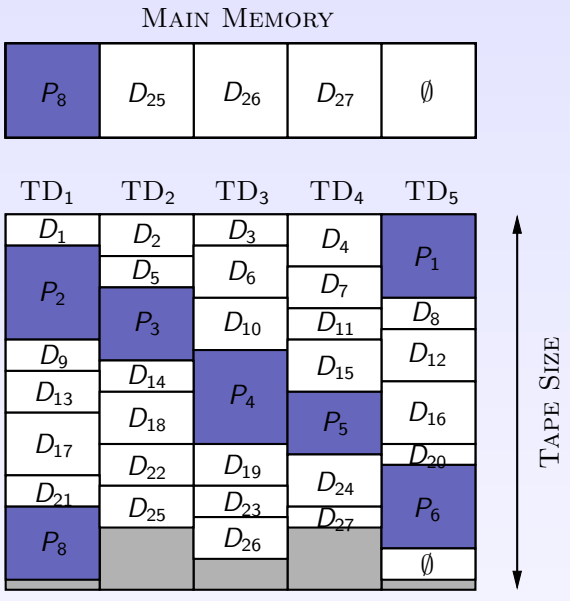
- Redundant Array of Independent Tapes, inspired by RAID
- Rationale: Writing data in parallel while balancing tape usage

\+ Parity blocks computed ''on the fly'' in RAM

\+ Data accessed in parallel

\+ Tape occupancy balanced

The RAIT policy



- Redundant Array of Independent Tapes, inspired by RAID
- Rationale: Writing data in parallel while balancing tape usage

\+ Parity blocks computed ''on the fly'' in RAM

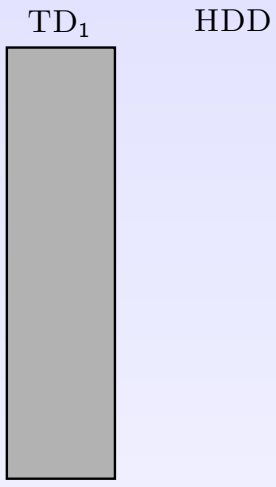
\+ Data accessed in parallel

\+ Tape occupancy balanced

– Data is fragmented

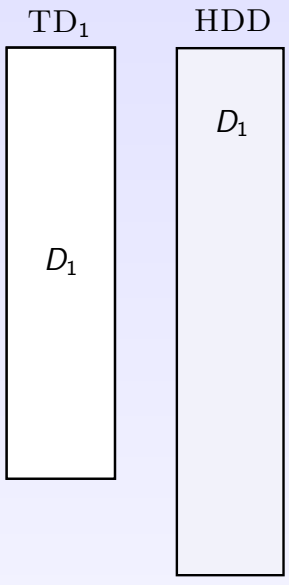
– Big impact on the level of parallelism of the system

The VERTICAL policy



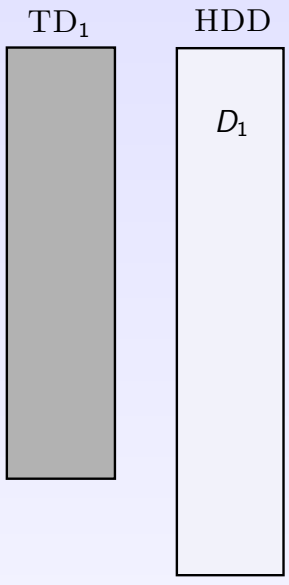
- Rationale: Writing data contiguously while maintaining the level of parallelism of the system

The VERTICAL policy



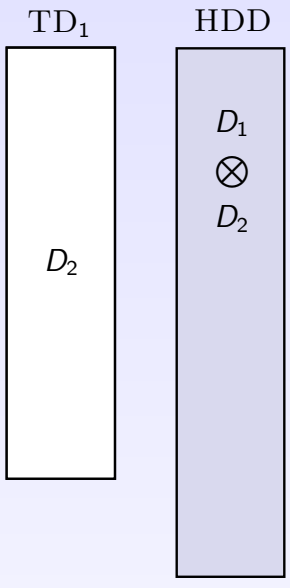
- Rationale: Writing data contiguously while maintaining the level of parallelism of the system

The VERTICAL policy



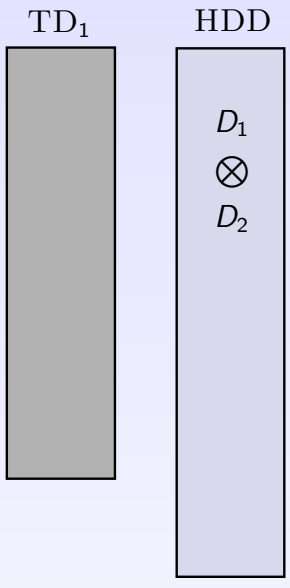
- Rationale: Writing data contiguously while maintaining the level of parallelism of the system

The VERTICAL policy



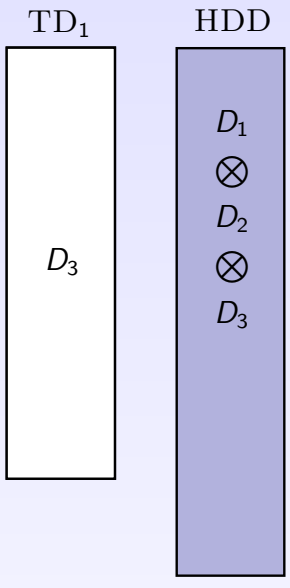
- Rationale: Writing data contiguously while maintaining the level of parallelism of the system

The VERTICAL policy



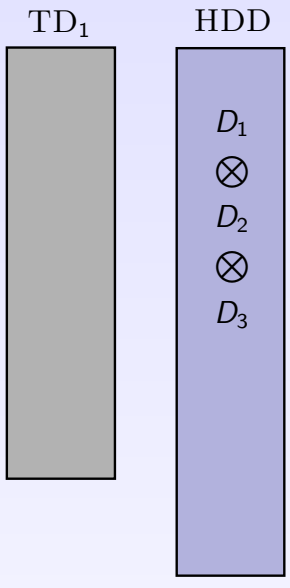
- Rationale: Writing data contiguously while maintaining the level of parallelism of the system

The VERTICAL policy



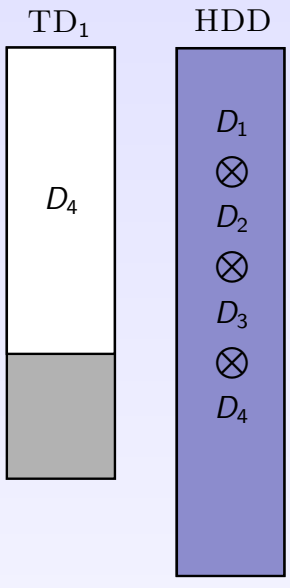
- Rationale: Writing data contiguously while maintaining the level of parallelism of the system

The VERTICAL policy



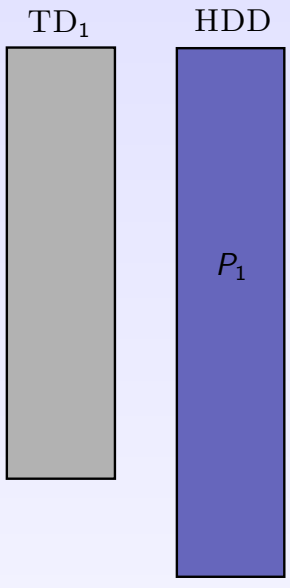
- Rationale: Writing data contiguously while maintaining the level of parallelism of the system

The VERTICAL policy



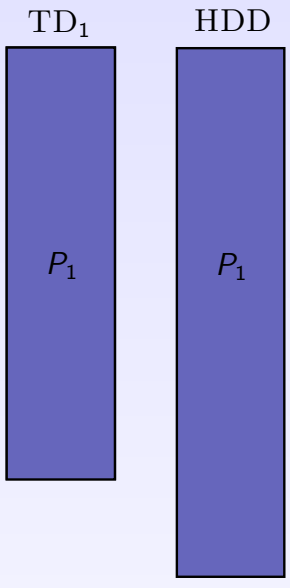
- Rationale: Writing data contiguously while maintaining the level of parallelism of the system

The VERTICAL policy



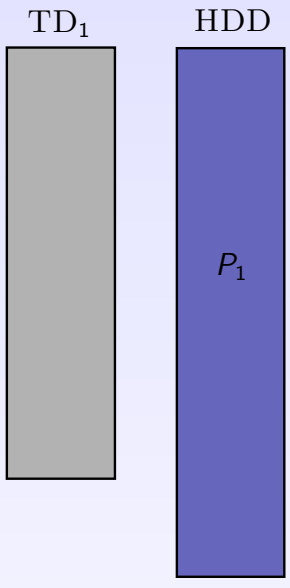
- Rationale: Writing data contiguously while maintaining the level of parallelism of the system

The VERTICAL policy



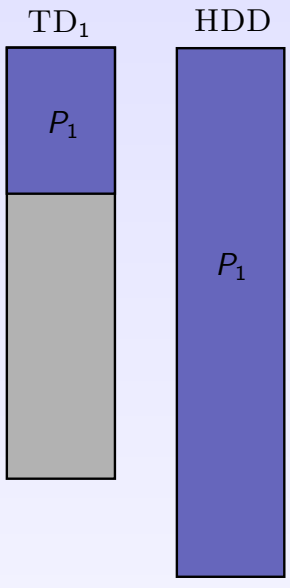
- Rationale: Writing data contiguously while maintaining the level of parallelism of the system

The VERTICAL policy



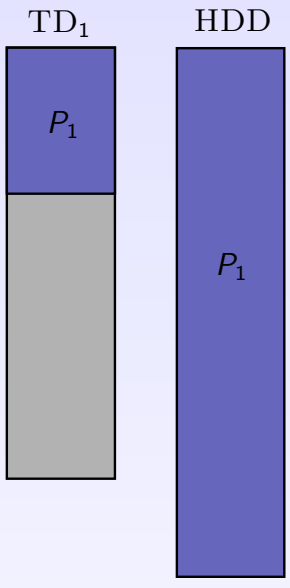
- Rationale: Writing data contiguously while maintaining the level of parallelism of the system

The VERTICAL policy



- Rationale: Writing data contiguously while maintaining the level of parallelism of the system

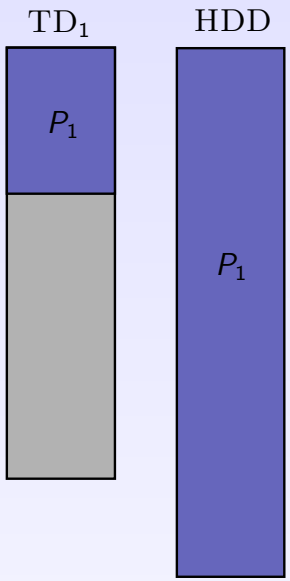
The VERTICAL policy



- Rationale: Writing data contiguously while maintaining the level of parallelism of the system

+ Data accessed sequentially
+ Keep entire level of parallelism of the system

The VERTICAL policy

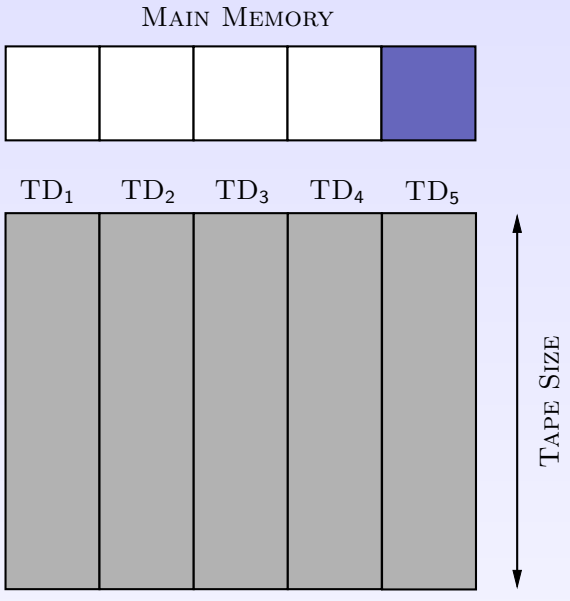


- Rationale: Writing data contiguously while maintaining the level of parallelism of the system

+ Data accessed sequentially
+ Keep entire level of parallelism of the system

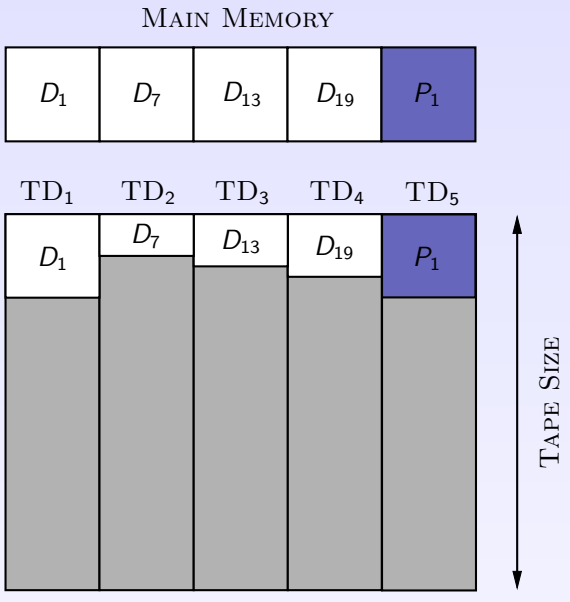
– No parallelism for a single request
– Large local storage required for parities
– Dedicated parity tapes, occupancy is less balanced

The PARALLEL policy



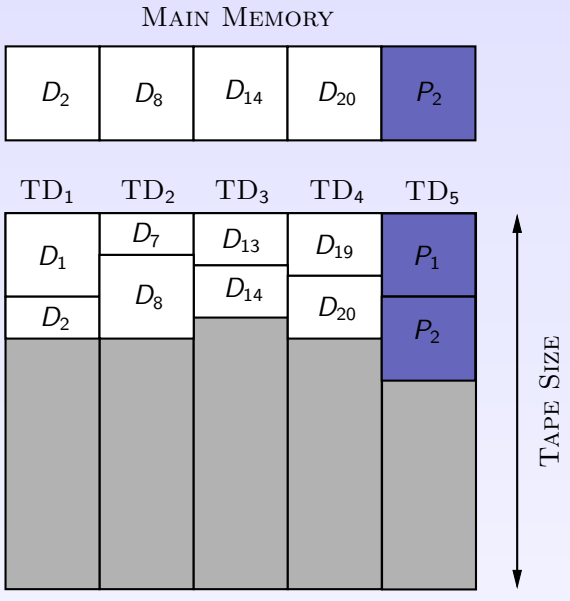
- Rationale: Writing data in parallel while enforcing contiguous access to tapes

The PARALLEL policy



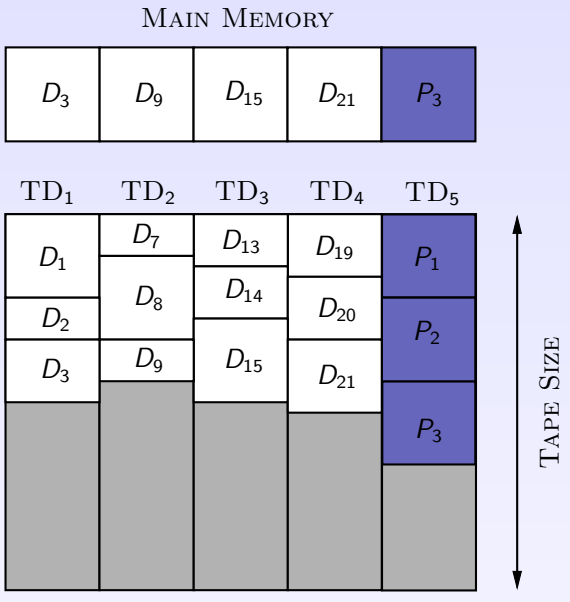
- Rationale: Writing data in parallel while enforcing contiguous access to tapes

The PARALLEL policy



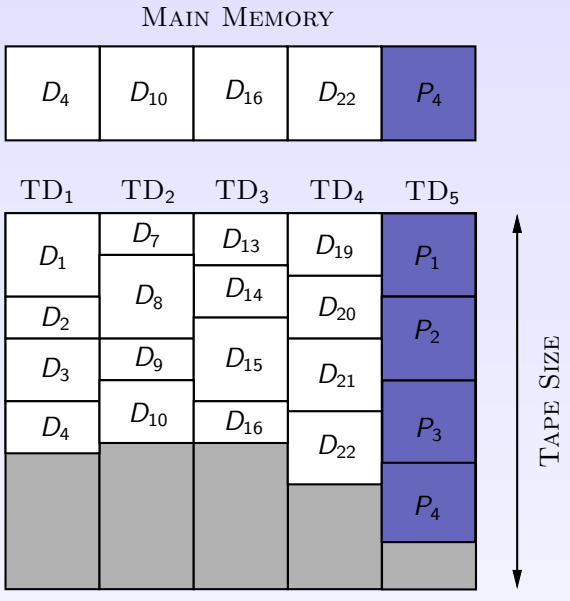
- Rationale: Writing data in parallel while enforcing contiguous access to tapes

The PARALLEL policy



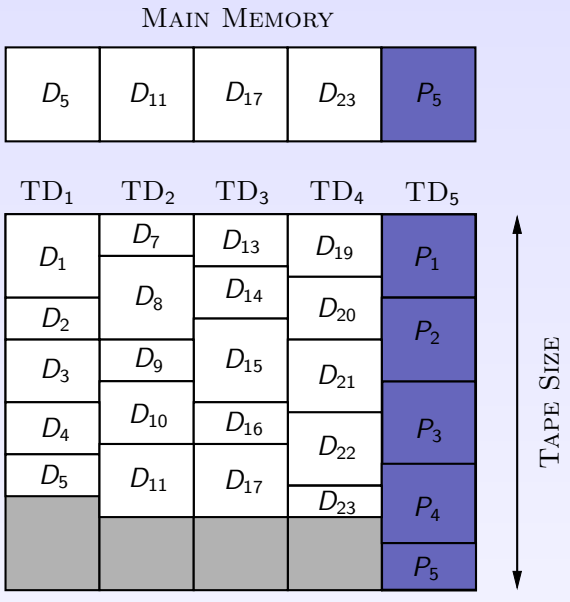
- Rationale: Writing data in parallel while enforcing contiguous access to tapes

The PARALLEL policy



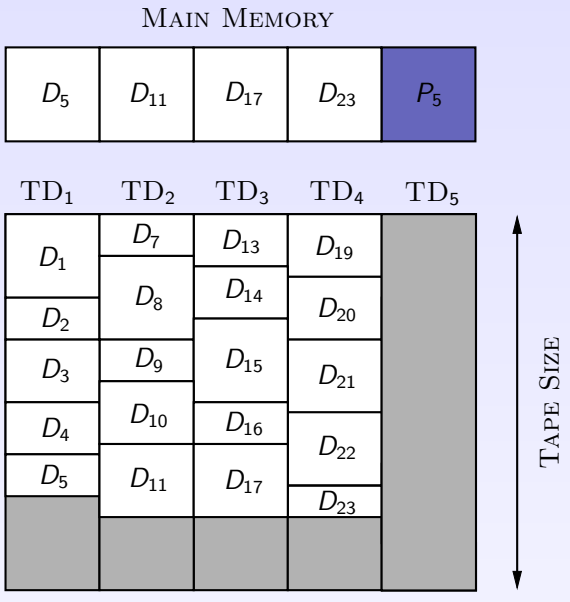
- Rationale: Writing data in parallel while enforcing contiguous access to tapes

The PARALLEL policy



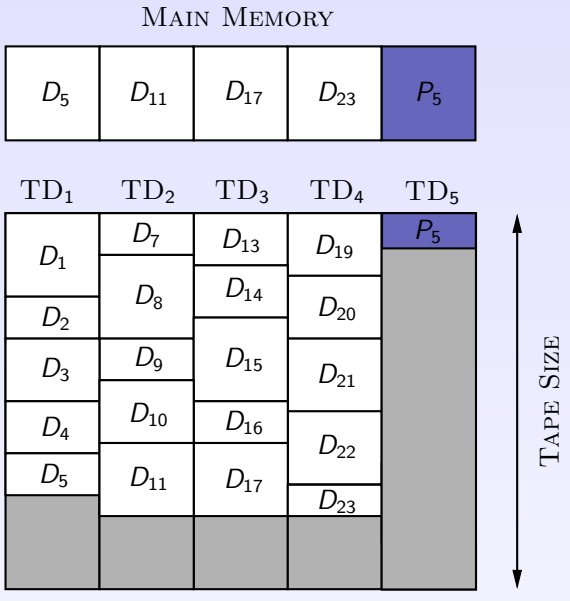
- Rationale: Writing data in parallel while enforcing contiguous access to tapes

The PARALLEL policy



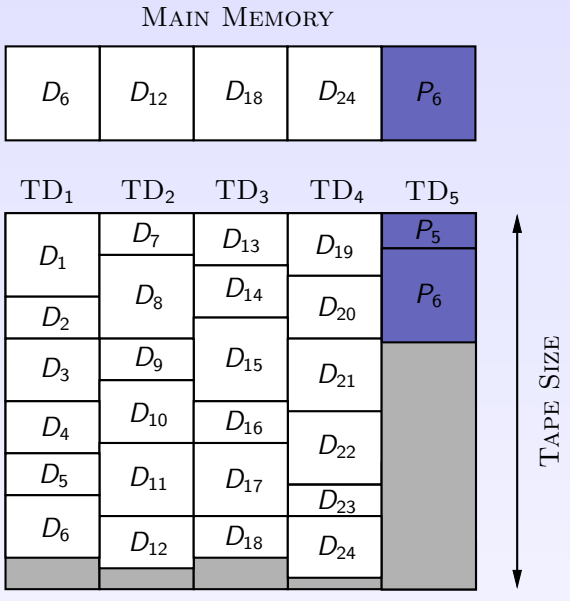
- Rationale: Writing data in parallel while enforcing contiguous access to tapes

The PARALLEL policy



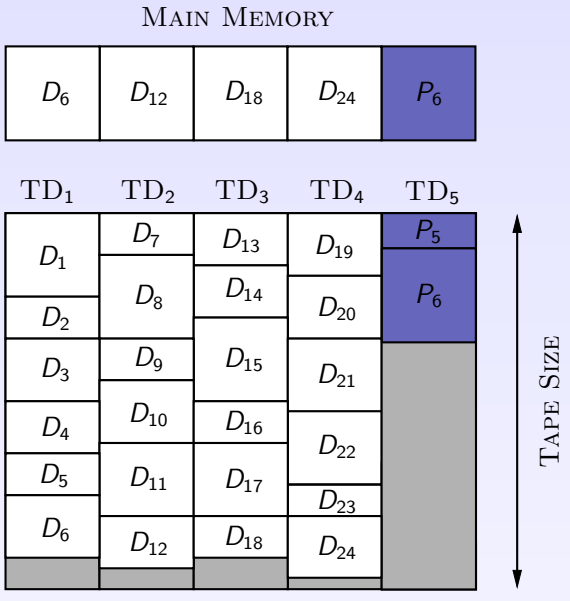
- Rationale: Writing data in parallel while enforcing contiguous access to tapes

The PARALLEL policy



- Rationale: Writing data in parallel while enforcing contiguous access to tapes

The PARALLEL policy



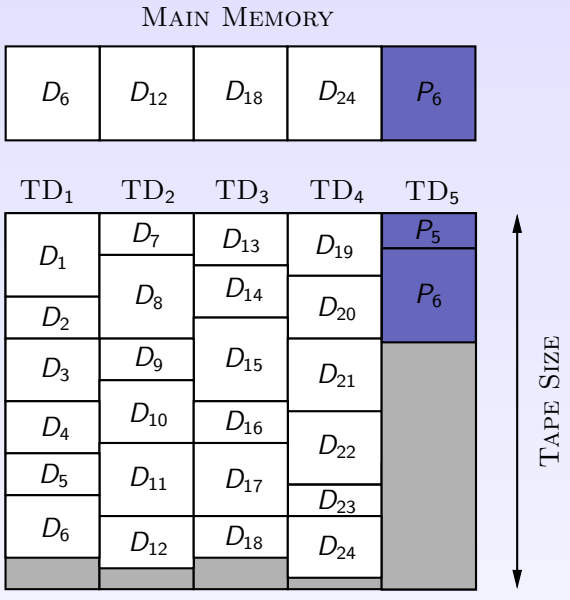
- Rationale: Writing data in parallel while enforcing contiguous access to tapes

\+ Parity computed in RAM

\+ Data accessed in parallel, only X tape drives when reading

\+ Larger contiguous data chunks than RAIT

The PARALLEL policy



- Rationale: Writing data in parallel while enforcing contiguous access to tapes

\+ Parity computed in RAM

\+ Data accessed in parallel, only X tape drives when reading

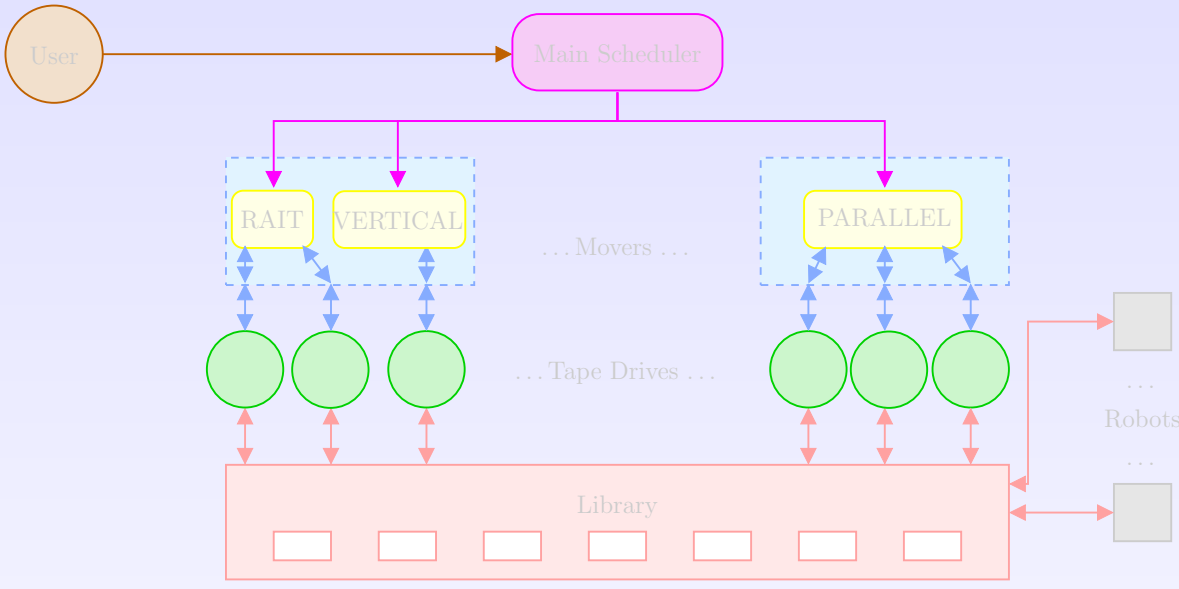
\+ Larger contiguous data chunks than RAIT

– Parity on dedicated tapes, occupancy less balanced

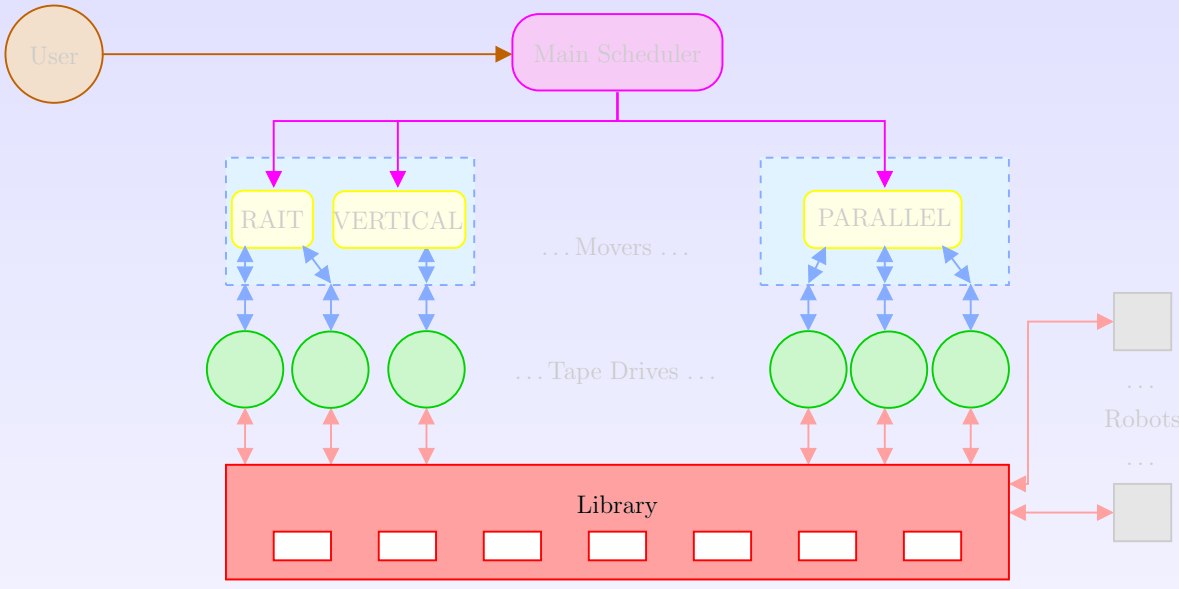
– Big impact on the level of parallelism of the system

Outline

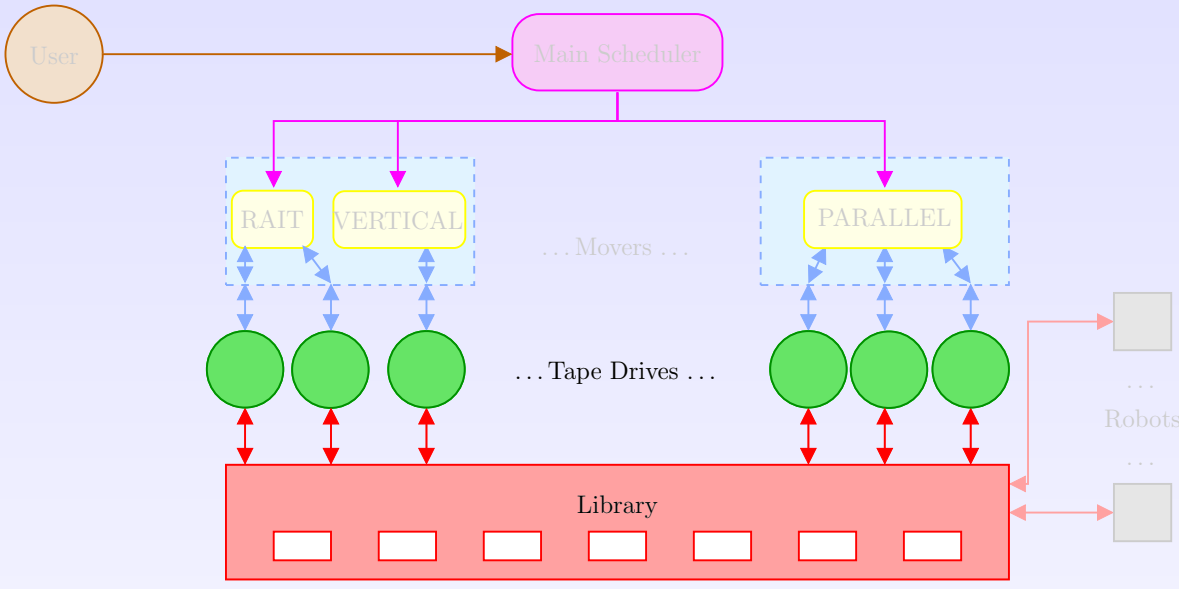
Simulation model



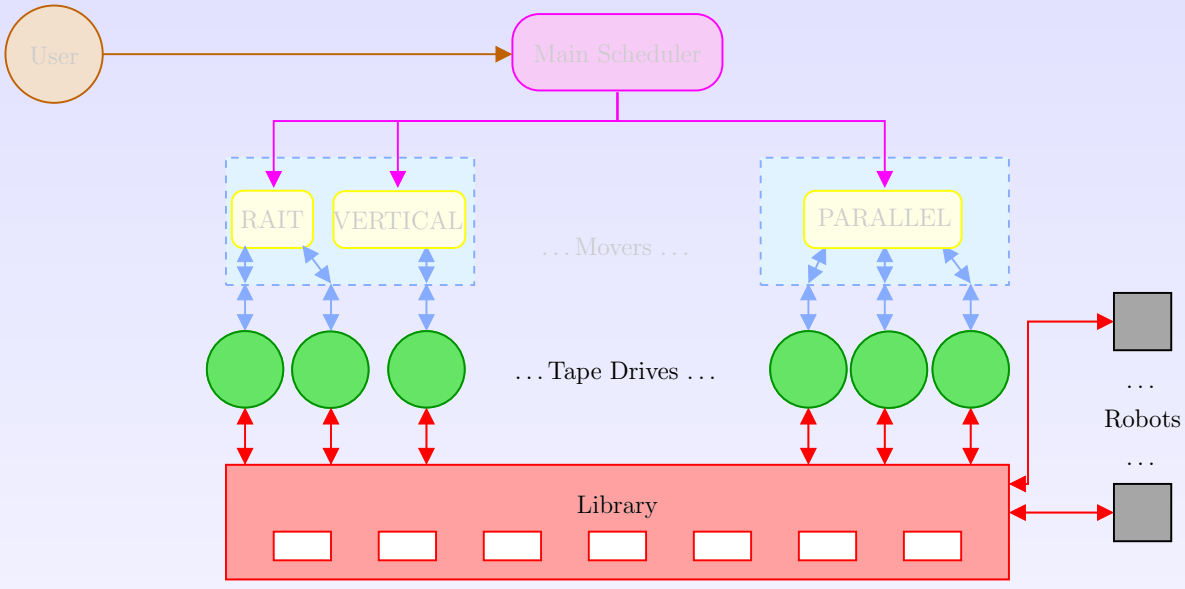
Simulation model



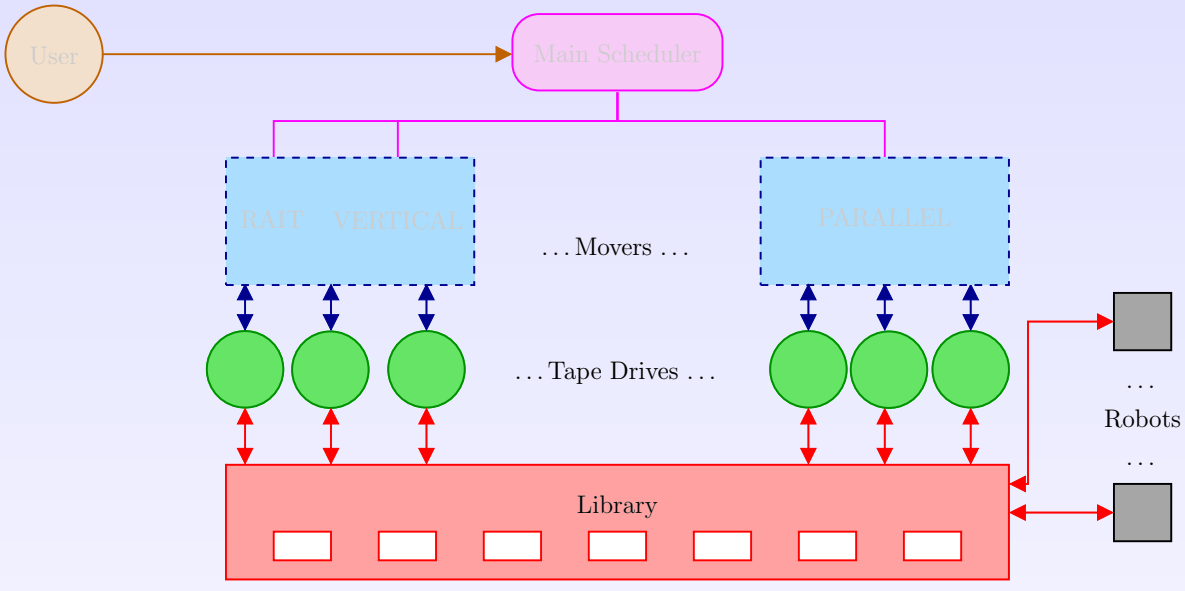
Simulation model



Simulation model

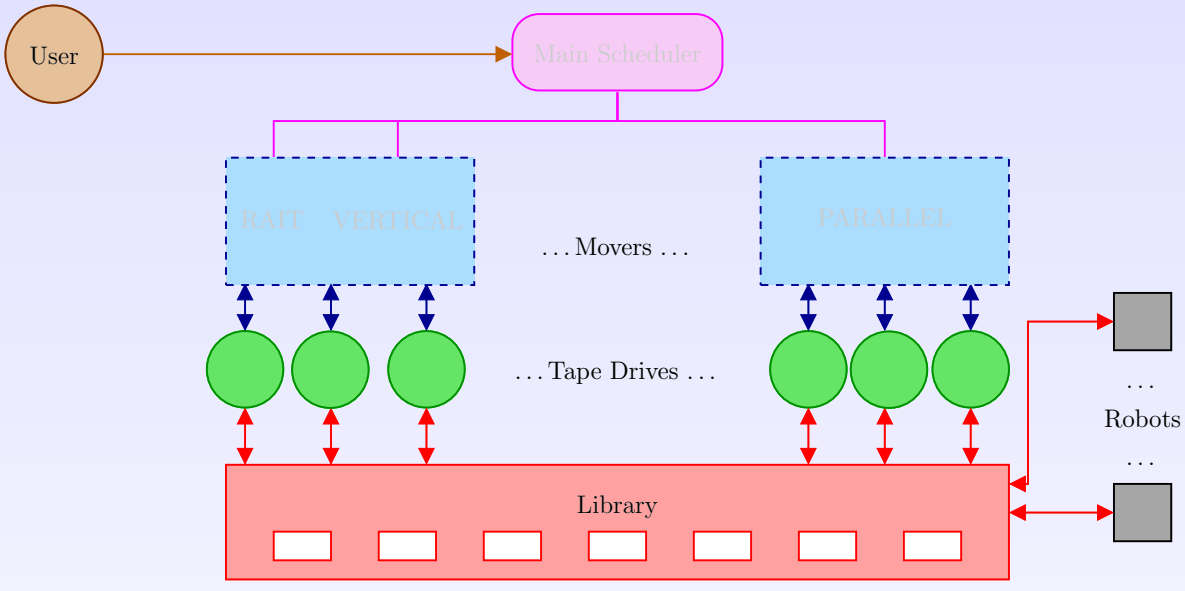


Simulation model



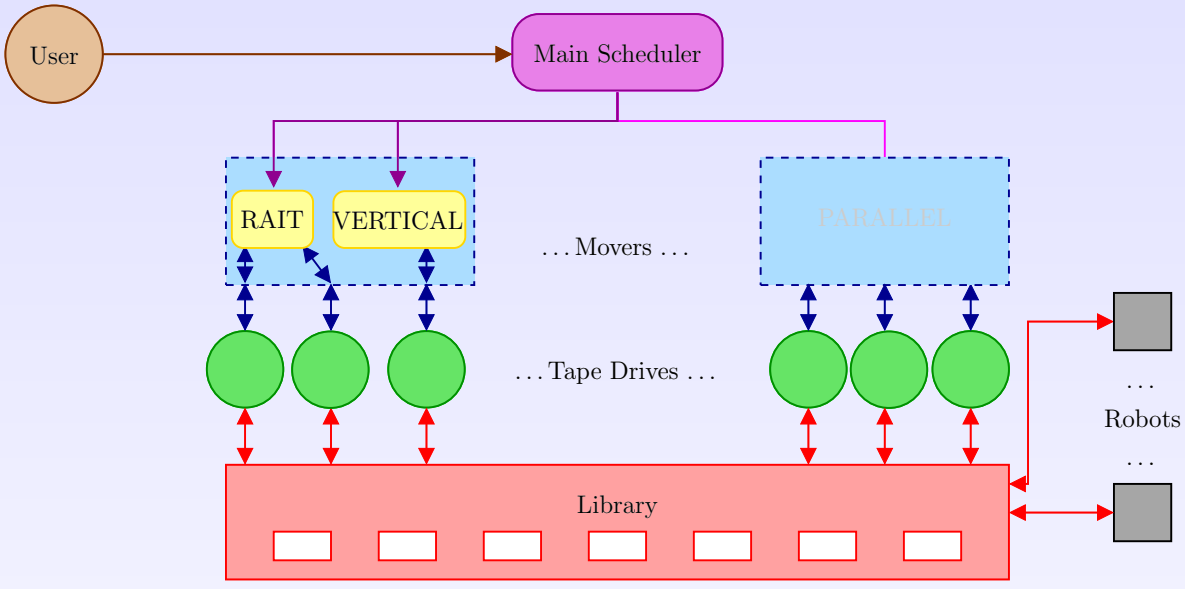
- Mover nodes can compute parity ''on the fly''
- Mover nodes have enough storage space to hold parities (for VERTICAL policy)

Simulation model



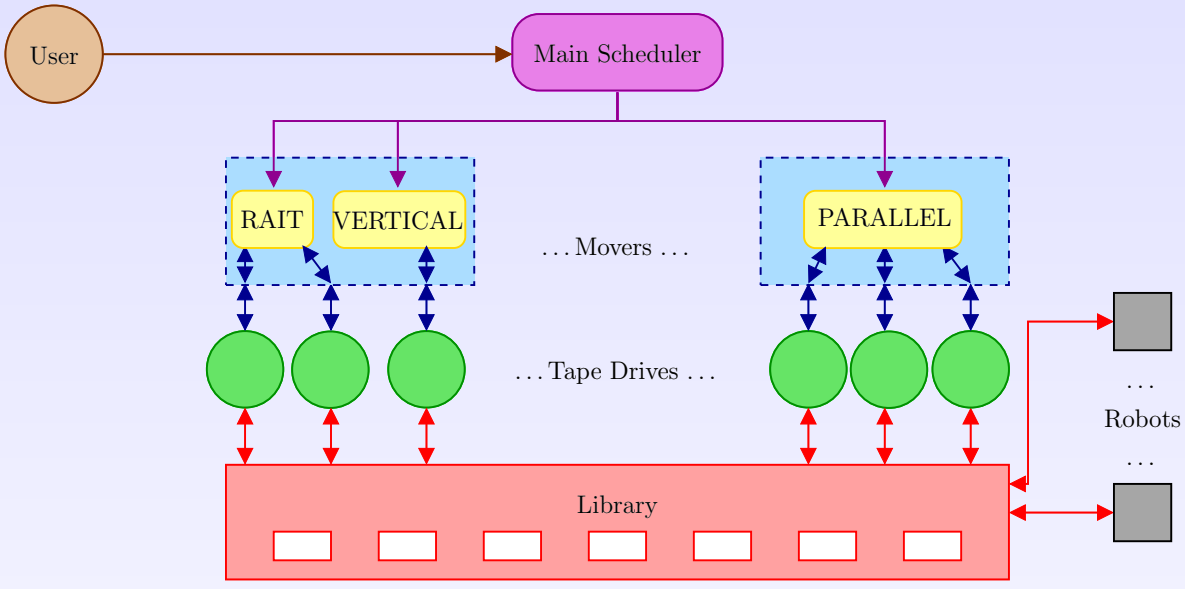
- Mover nodes can compute parity ''on the fly''
- Mover nodes have enough storage space to hold parities (for VERTICAL policy)

Simulation model



- Main scheduler, ''FIFO First Fit'' policy
  - Foreach mover:
    - Find unassigned tape drives holding tapes matching policy
    - Find other unassigned tape drives
    - If enough tape drives, assign request to current mover, starting with its matching tape drives
  - If the system is unable to handle the request, wait

Simulation model



- Main scheduler, ''FIFO First Fit'' policy
  - Foreach mover:
    - Find unassigned tape drives holding tapes matching policy
    - Find other unassigned tape drives
    - If enough tape drives, assign request to current mover, starting with its matching tape drives
  - If the system is unable to handle the request, wait

Random workload generation

- Arrival dates: Poisson process with arrival rate λ
- File size: random log uniform distribution
- I/O type chosen uniformly
- File reuse probability: 0.15
- Resiliency class $X + Y$ chosen among predefined classes with probability p_{X+Y}
- Data compression rate C_D in $[1, 3]$
- Parity compression rate C_P in $[1, C_D]$

Experimental setup

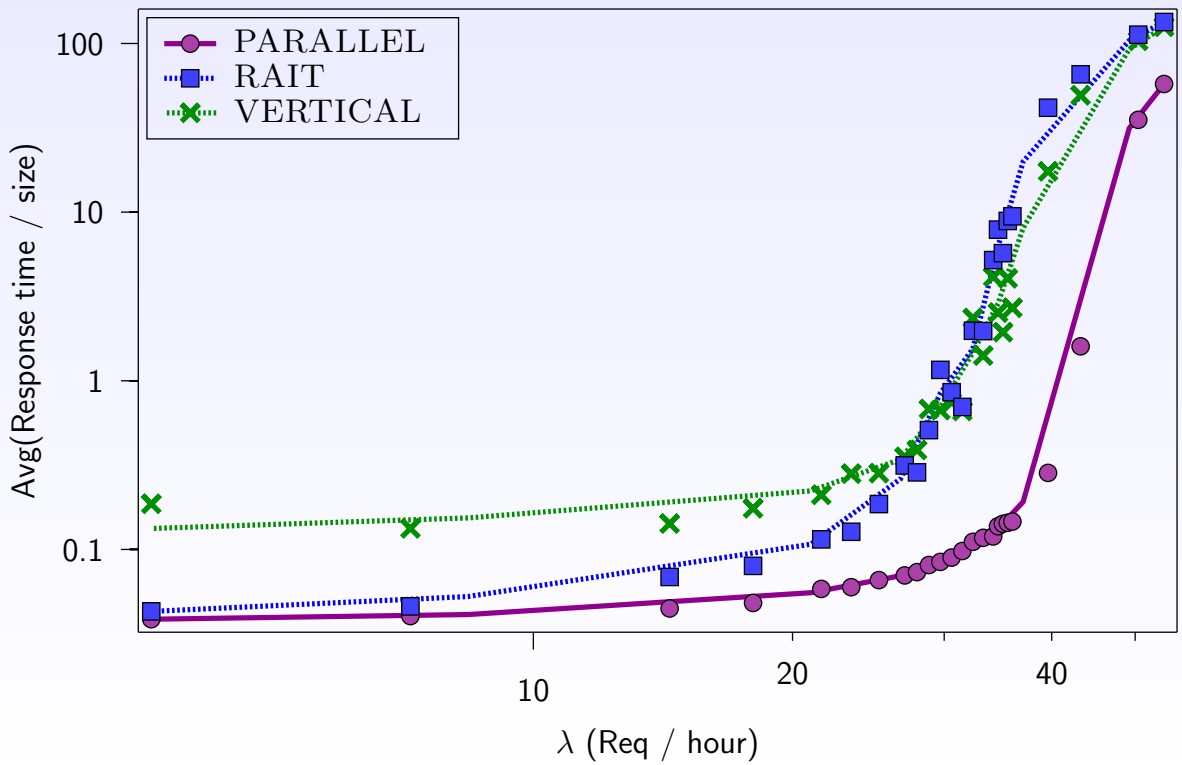
- Experiments performed using a discrete event based simulator: SimGrid
- Homogeneous policies
- Platform:
  - 1 tape library with 10 robotic arms
  - 50 mover nodes
  - 500 tape drives (10 per mover node)
- Resiliency schemes:

$X+Y$	$1+0$	$1+1$	$2+1$	$3+1$
p_{X+Y}	0.05	0.05	0.1	0.2

$X+Y$	$4+1$	$4+2$	$6+2$	$8+2$
p_{X+Y}	0.2	0.2	0.2	0.2

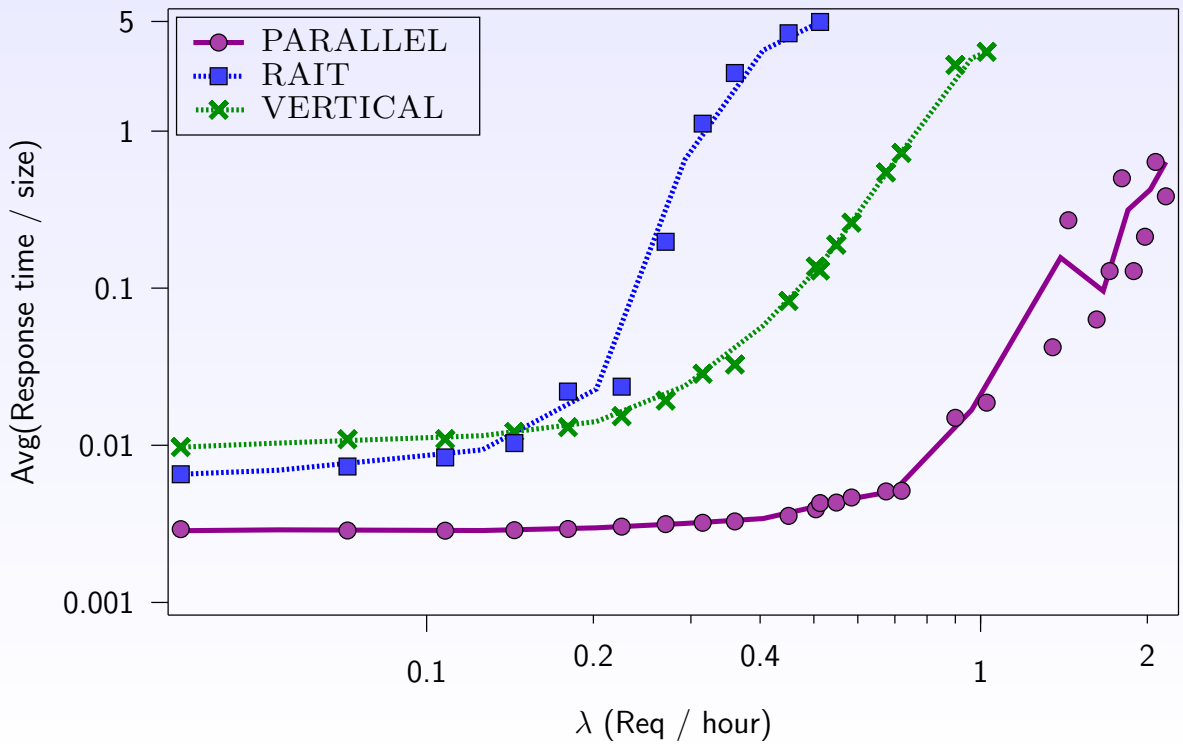
Avg(Response time / size) for small files

- File sizes range from 10^3 to 10^6 MB
- PARALLEL best solution
- RAIT close to PARALLEL on low arrival rates
- VERTICAL similar to RAIT on high arrival rates



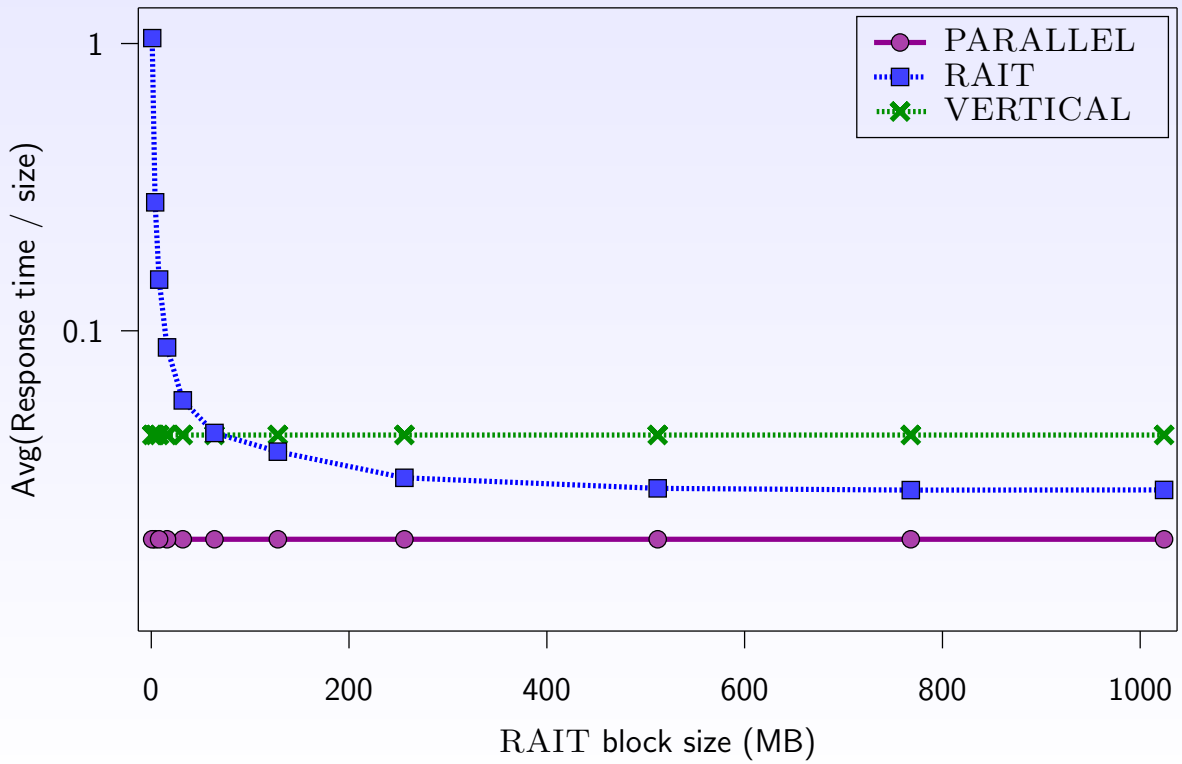
Avg(Response time / size) for big files

- File sizes range from 10^6 to 10^8 MB
- PARALLEL still best solution
- VERTICAL handles higher arrival rates than RAIT



Influence of RAIT block size

- Fixed λ value, various RAIT block sizes
- Significant impact on average response time only for small values



Tape occupancy and influence of RAIT block size

- Tape occupancy is a good indicator of data repacking need
- As expected, VERTICAL policy is the best solution

Policy	RAIT	PARALLEL	VERTICAL
Avg. Occupancy	79.3%	71.4%	83.9%
Std. Dev.	0.2	0.4	0.1

- RAIT block size has only a minor impact on tape occupancy:
  - 7% difference between 1MB blocks and 1GB blocks.

Outline

Conclusion and perspectives

Contributions

- VERTICAL and PARALLEL I/O policies
- Experimental performance evaluation of every policy through simulation using randomized workloads
- Evaluation of the impact of RAIT block size

Ongoing & future work

- Pipelining I/O requests
- Study heterogeneous policies:
  - with static policy allocation on mover nodes
  - with dynamic policy allocation on mover nodes
- Handle hardware failures and data reconstruction, and assess the impact of data reconstruction and repacking on performance