中国科学技术大学
University of Science and Technology of China

Fast Reconstruction for Large Disk Enclosures Based on RAID2.0

Qiliang Li, Min Lyu, Liangliang Xu, Yinlong Xu and Wei Wang

University of Science and Technology of China

ICPP 2021

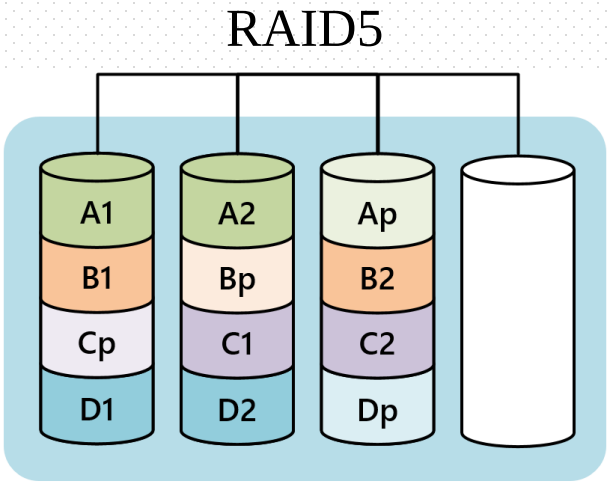
Large Disk Enclosures and RAID

- Large disk enclosures are widely used to provide large capacity storage

OceanStor 18000F V5

Dell PowerVault MD3060e

- Limitations of conventional RAID
  - Slow reconstruction



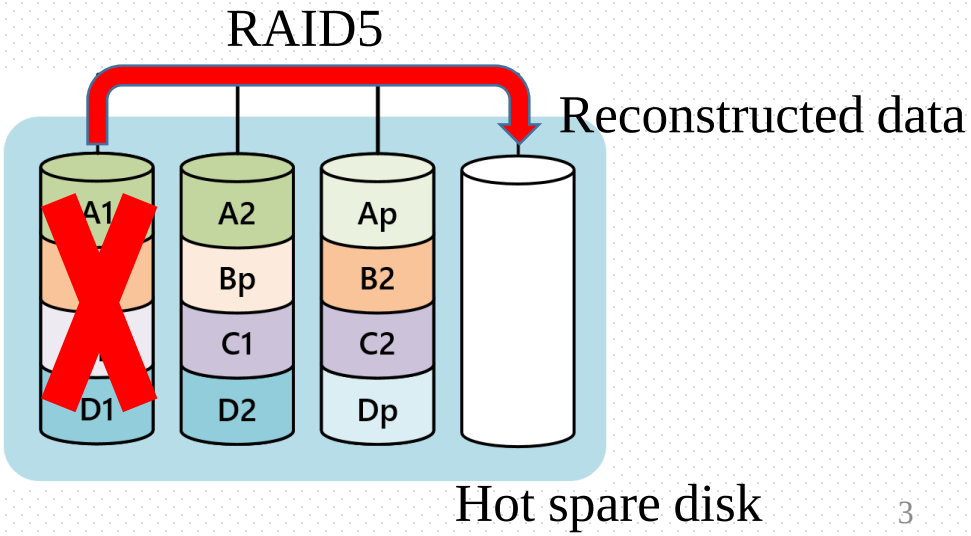
Large Disk Enclosures and RAID

- Large disk enclosures are widely used to provide large capacity storage

OceanStor 18000F V5

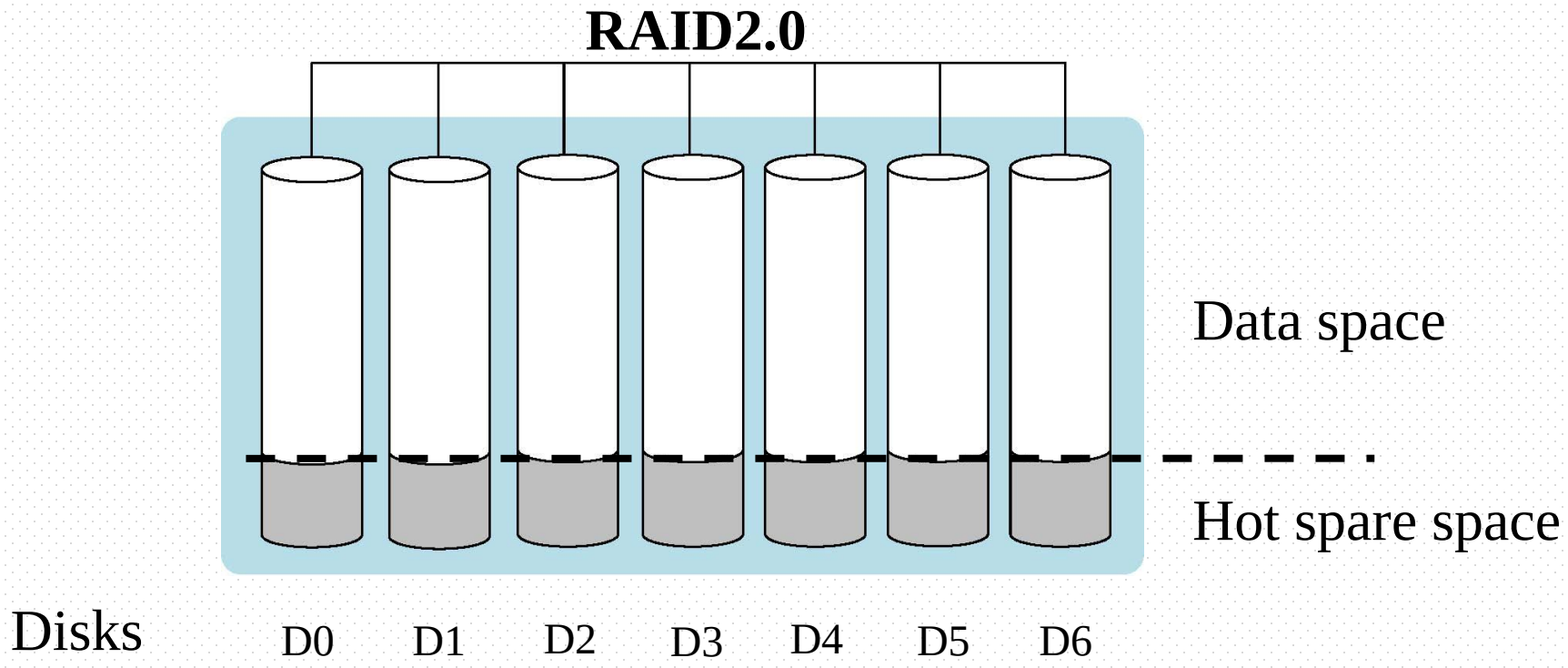
Dell PowerVault MD3060e

- Limitations of conventional RAID
  - Slow reconstruction



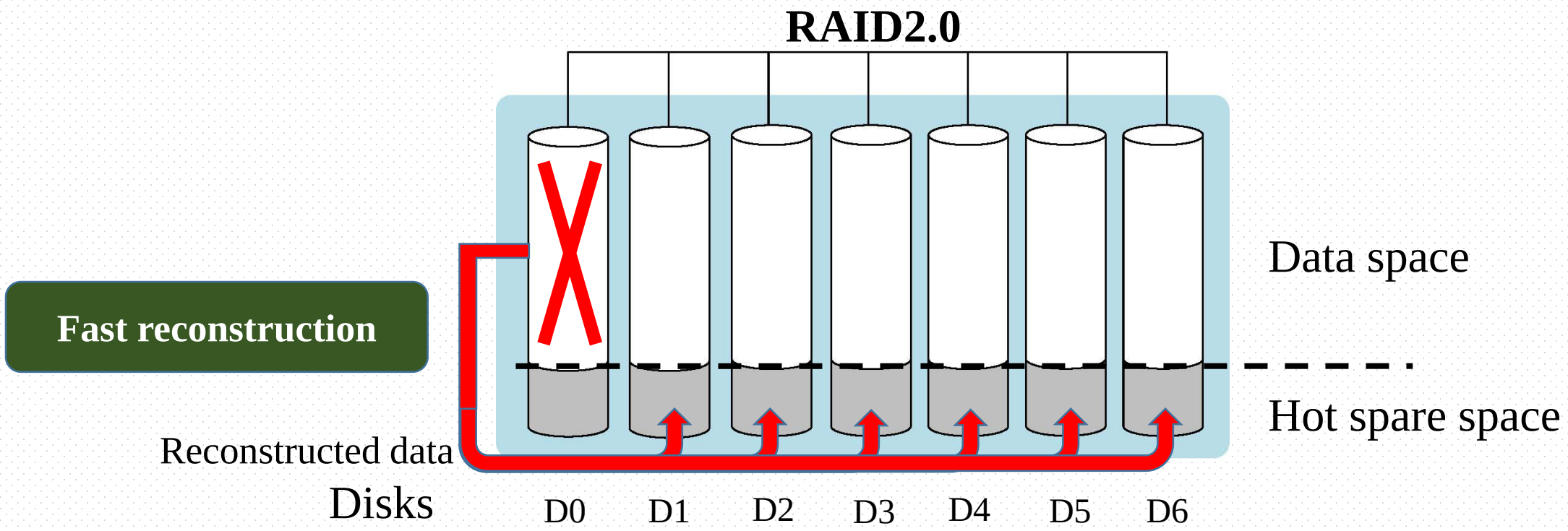
RAID2.0 Architecture

- Storage space contains data space and hot spare space
  - Data space: storing data in normal state



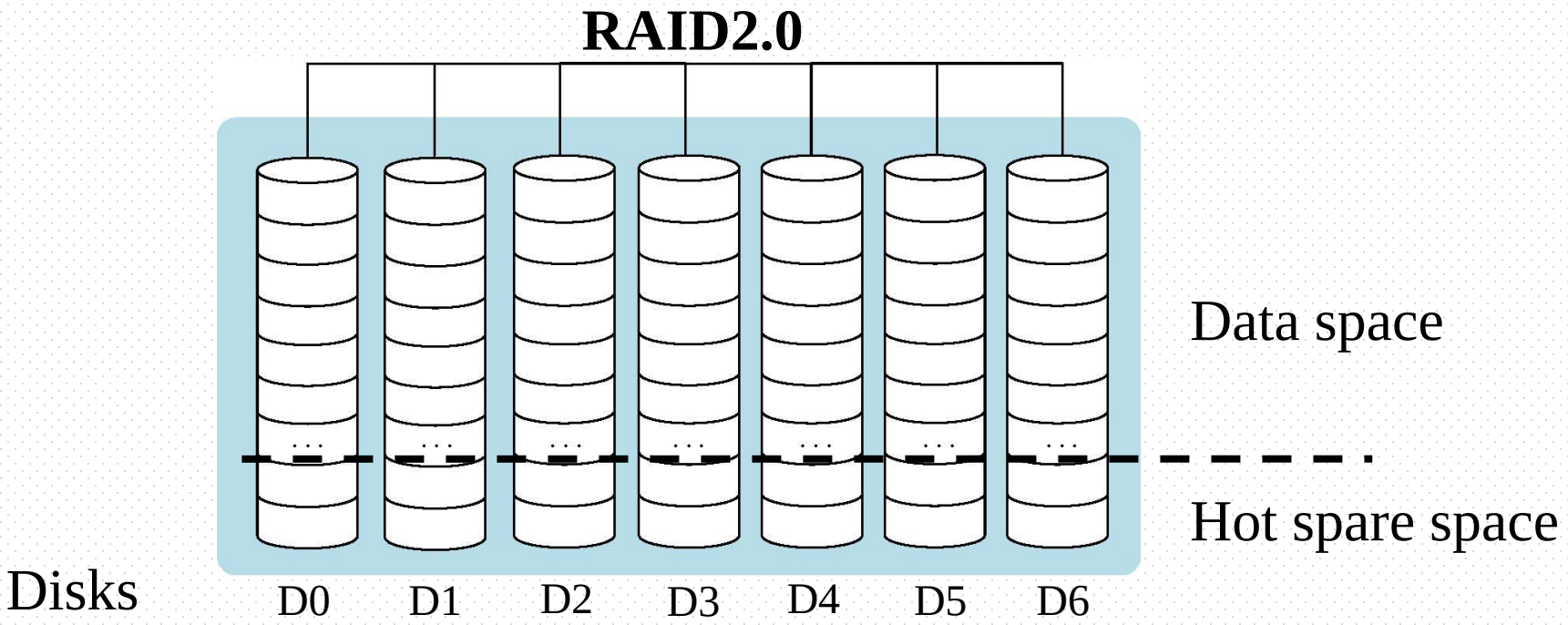
RAID2.0 Architecture

- Storage space contains data space and hot spare space
  - Data space: storing data in normal state
  - Hot spare space: storing reconstructed data in case of failure



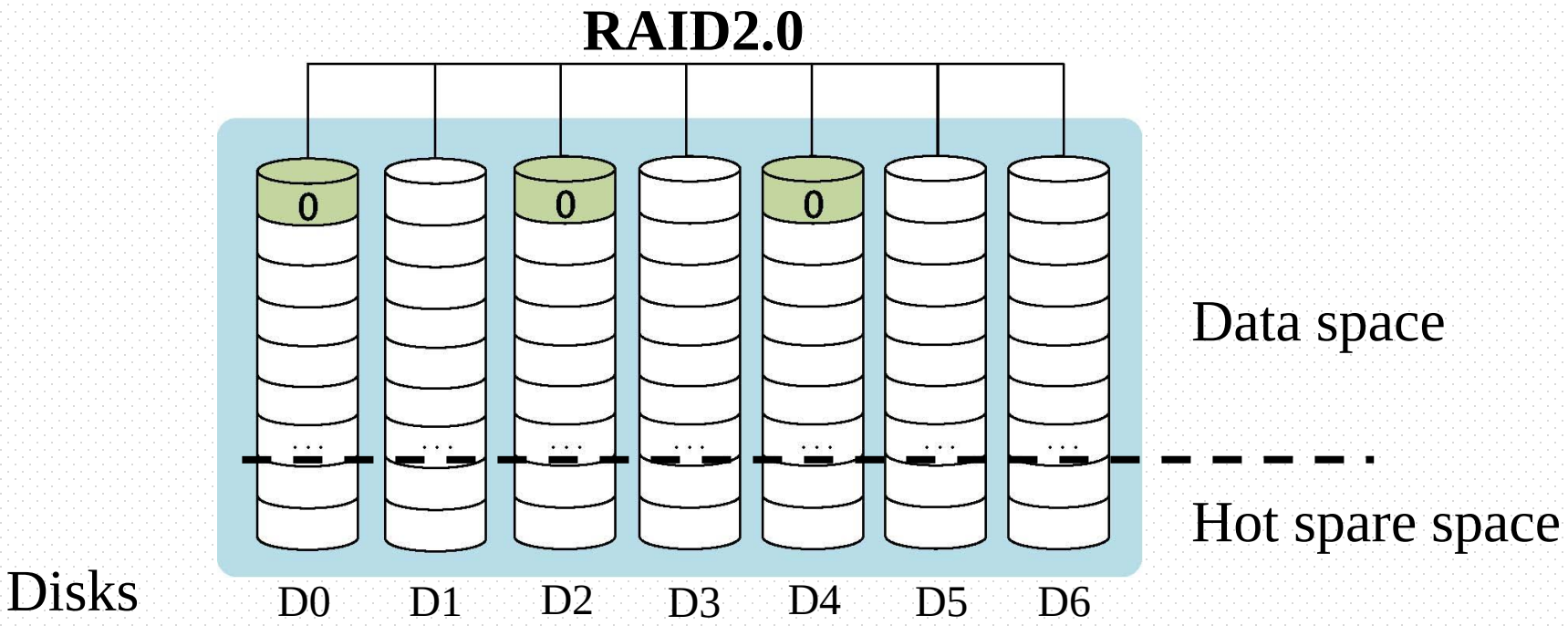
RAID2.0 Architecture

- Divide storage space into chunks
  - Each chunk usually not smaller than 64MB



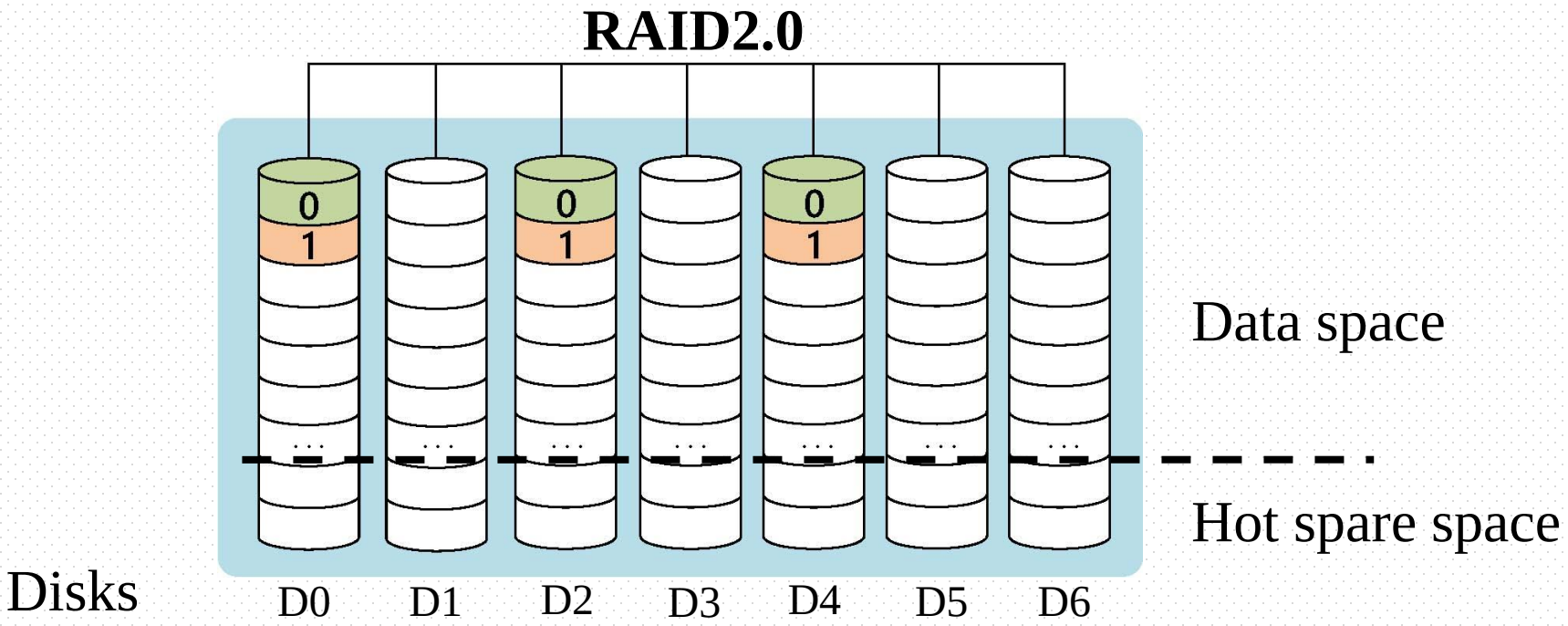
RAID2.0 Architecture

- Divide storage space into chunks
- Construct RAID group by randomly selecting chunks
  - E.g., 2+1 RAID group: group 0



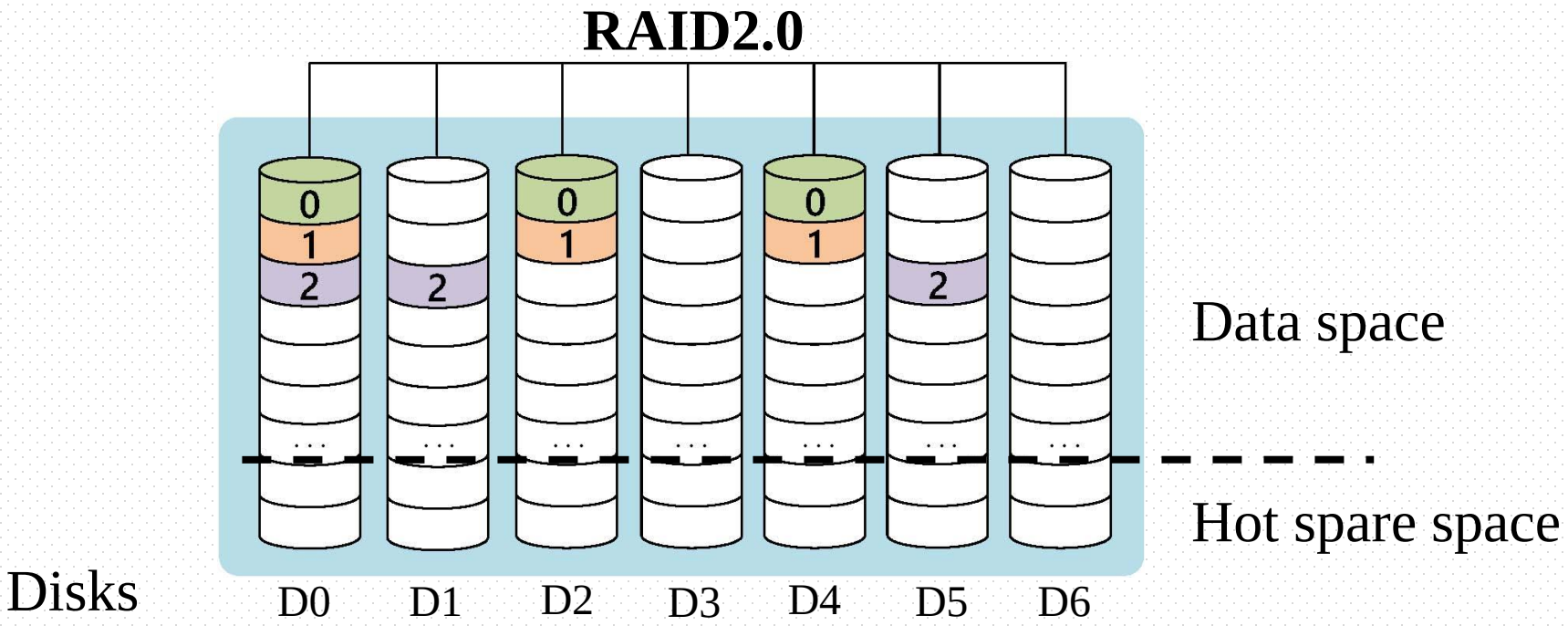
RAID2.0 Architecture

- Divide storage space into chunks
- Construct RAID group by randomly selecting chunks
  - E.g., 2+1 RAID group: group 0, group 1



RAID2.0 Architecture

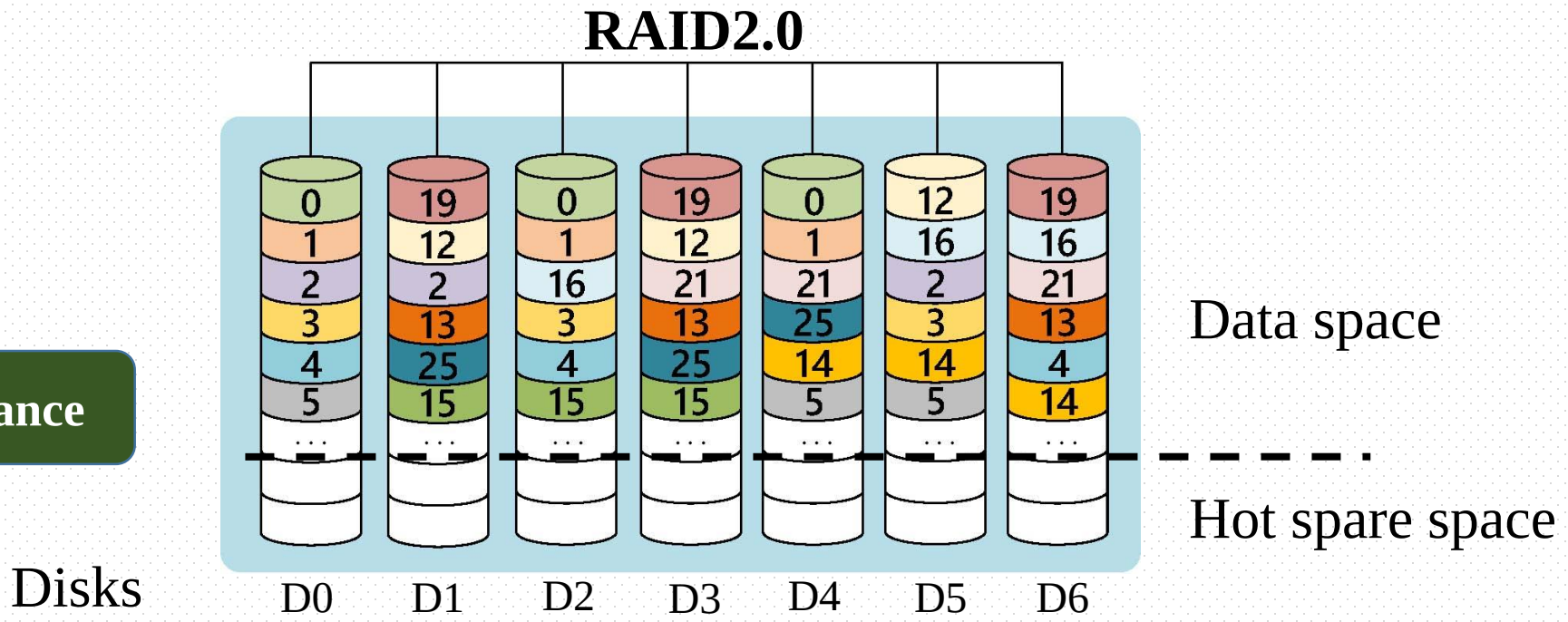
- Divide storage space into chunks
- Construct RAID group by randomly selecting chunks
  - E.g., 2+1 RAID group: group 0, group 1, group 2



RAID2.0 Architecture

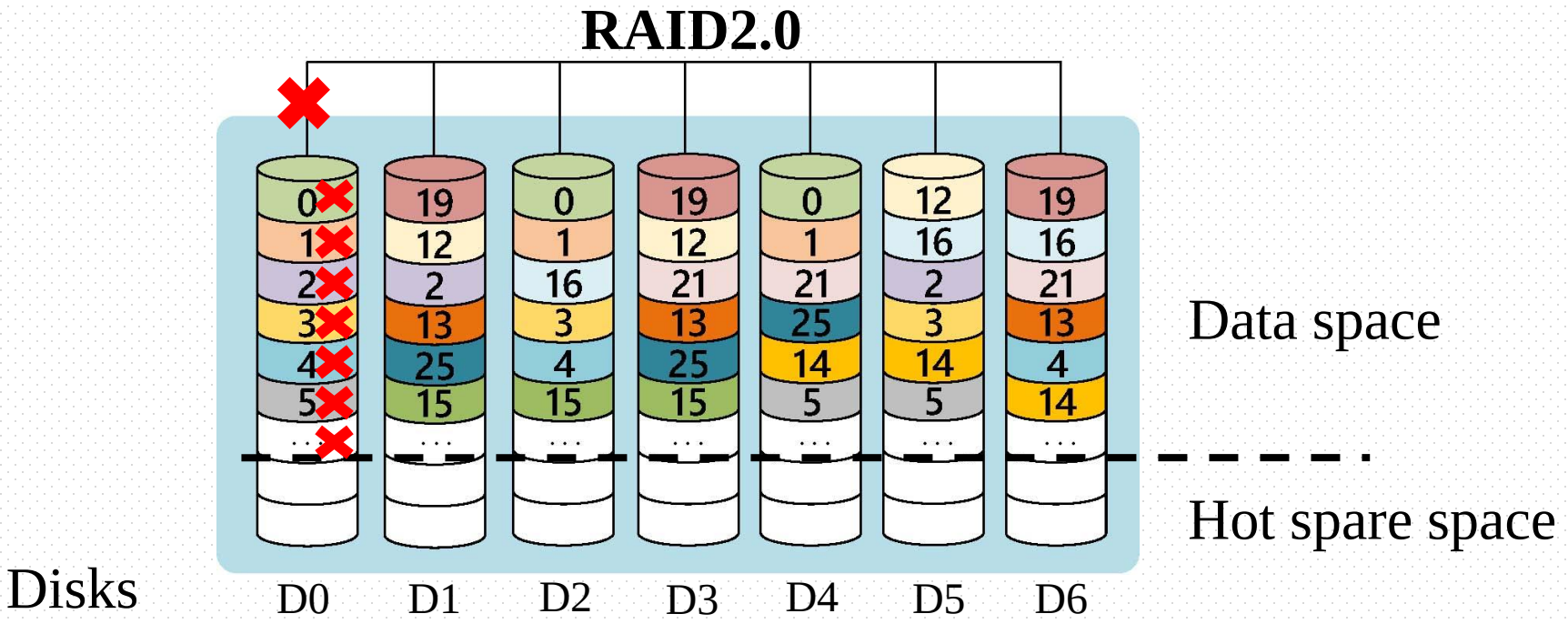
- Divide storage space into chunks
- Construct RAID group by randomly selecting chunks
  - E.g., 2+1 RAID group: group 0, group 1, group 2, etc.

Storage load balance



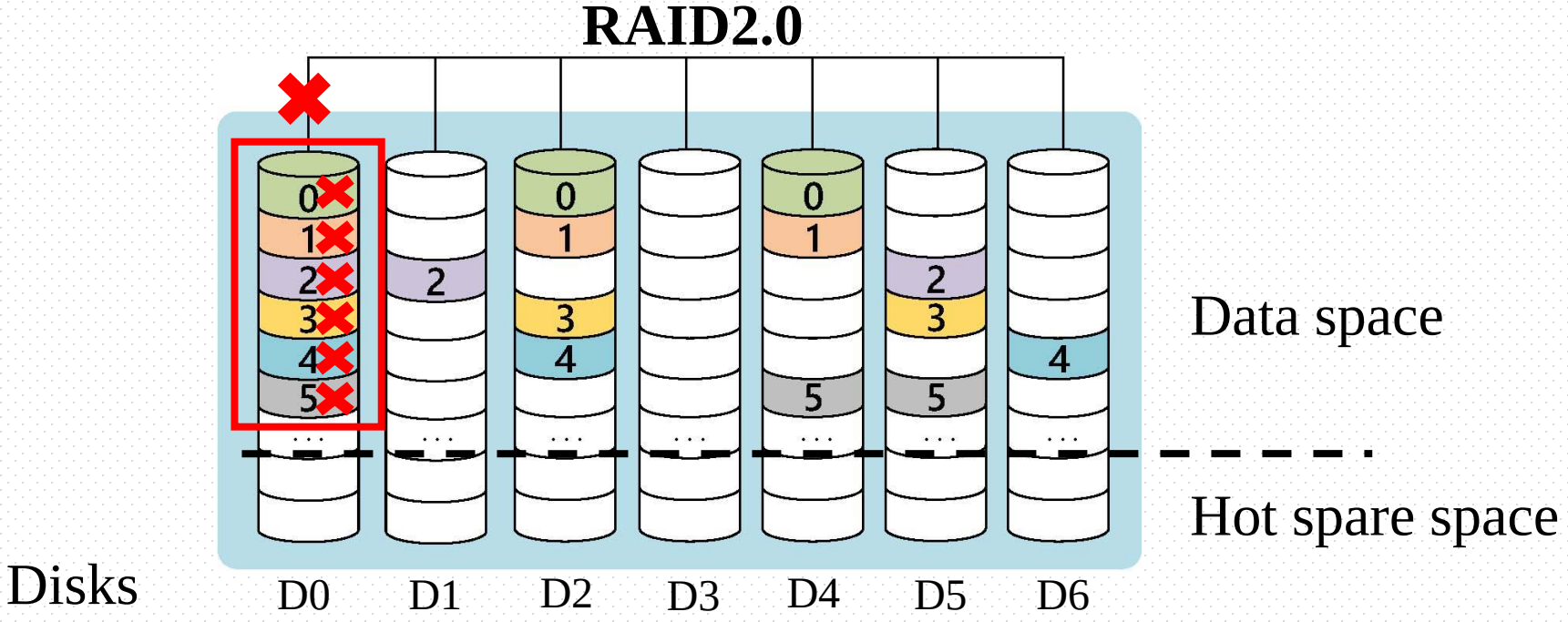
Reconstruction in RAID2.0

- When a disk fails, all lost chunks denoted as *recovery tasks* are pending for being reconstructed



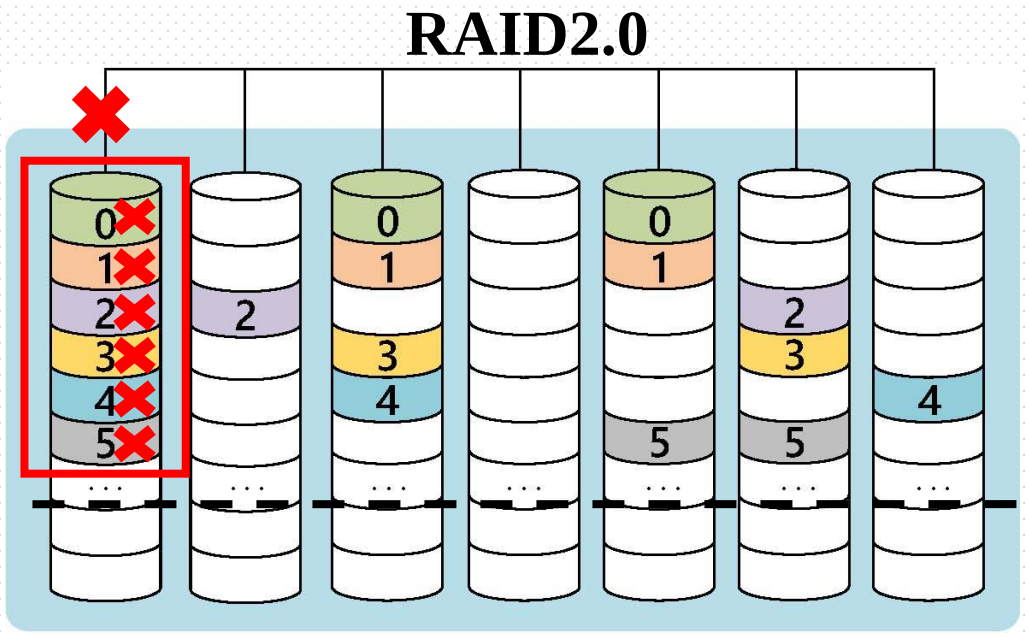
Reconstruction in RAID2.0

- Perform reconstruction in batches
  - Limited memory capacity and CPU resource
- Each batch sequentially reconstructs a limited number of groups
- Select disks randomly for reconstructed chunks



Imbalanced Data Layout Within a Batch

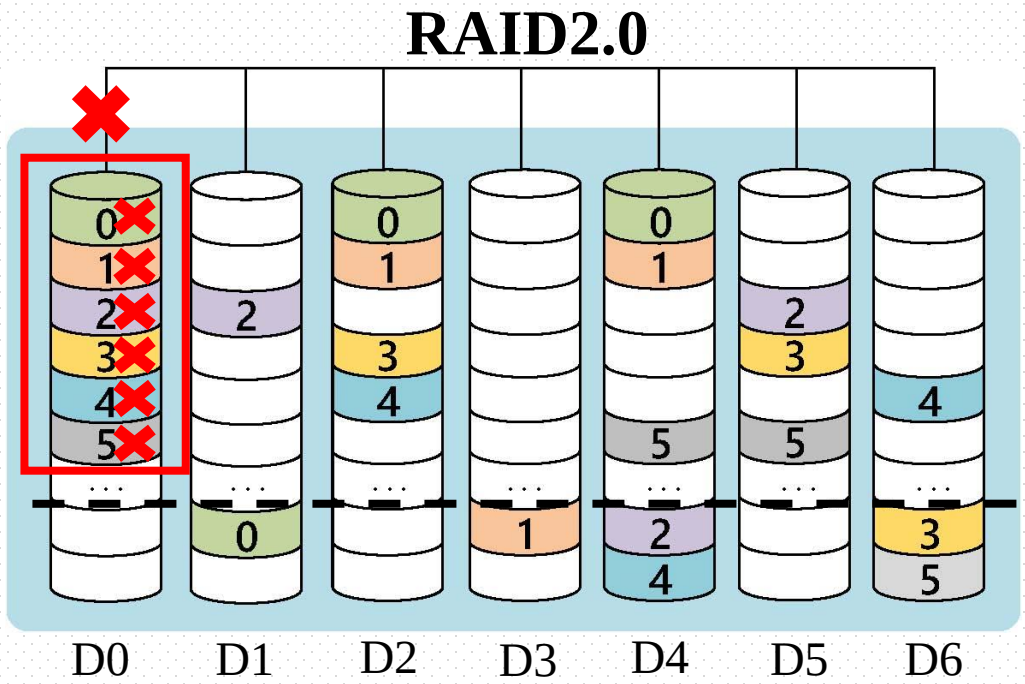
Imbalanced data layout
➡ Imbalanced read load



Random Selection of Surviving Disks

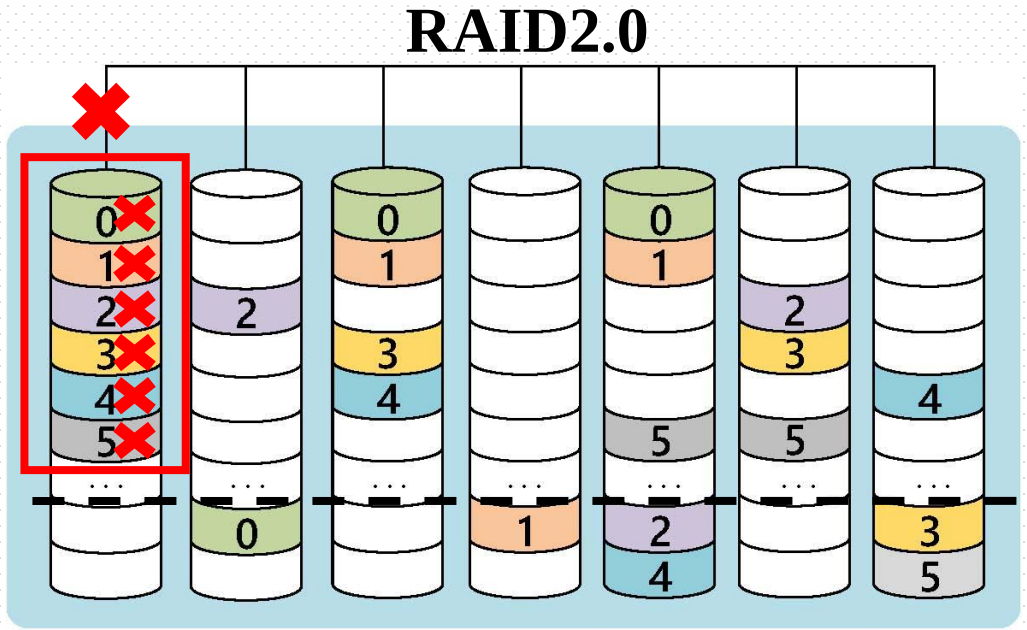
Imbalanced data layout
➡ Imbalanced read load

Random selection of disks
➡ Imbalanced write load



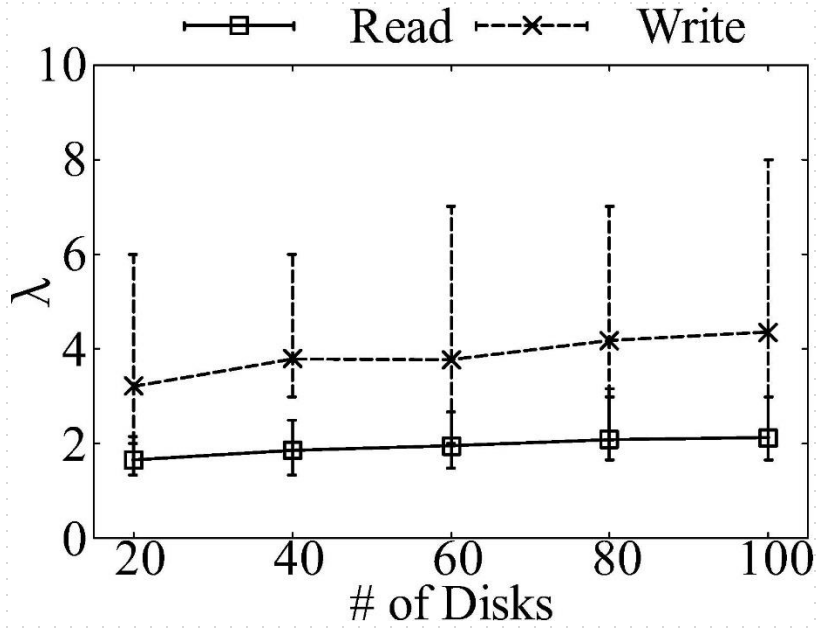
Local Load Imbalance

- Definition of load balance rate λ
  - $\lambda_{r/w}$: Ratio of the maximum number of chunks read from/written to a disk to the average in a batch



Evaluate Local Load Imbalance

- Simulations of $RAID_R$
  - Use the system time as random seed
  - 6+1 RAID group, CDF of 100 batches
  - Single disk failure



Different disk numbers

Popular Approach: Design Data Layout

- Guarantee uniform data distribution for balanced failure recovery I/O [1]
- Limitations
  - Relocation costs
    - Long used storage status —Relocation→ Designed dedicated data layout
  - Maintenance costs
    - Data layout after recovery or scaling —Relocation→ Normal data layout
  - Heterogeneous cases
    - Existence of hot spot disks with less rebuilding bandwidth

[1] RAID+: Deterministic and balanced data distribution for large disk enclosures - Zhang et al., at FAST'18

Objectives

Goal 1: Achieving local read and write load balance in reconstruction

Goal 2: Applicable to heterogeneous rebuilding bandwidth

Design of DR-RAID

Balance read load

- Step 1 – Batching tasks out of order

Balance write load

- Step 2 – Selecting surviving disks based on matching theory

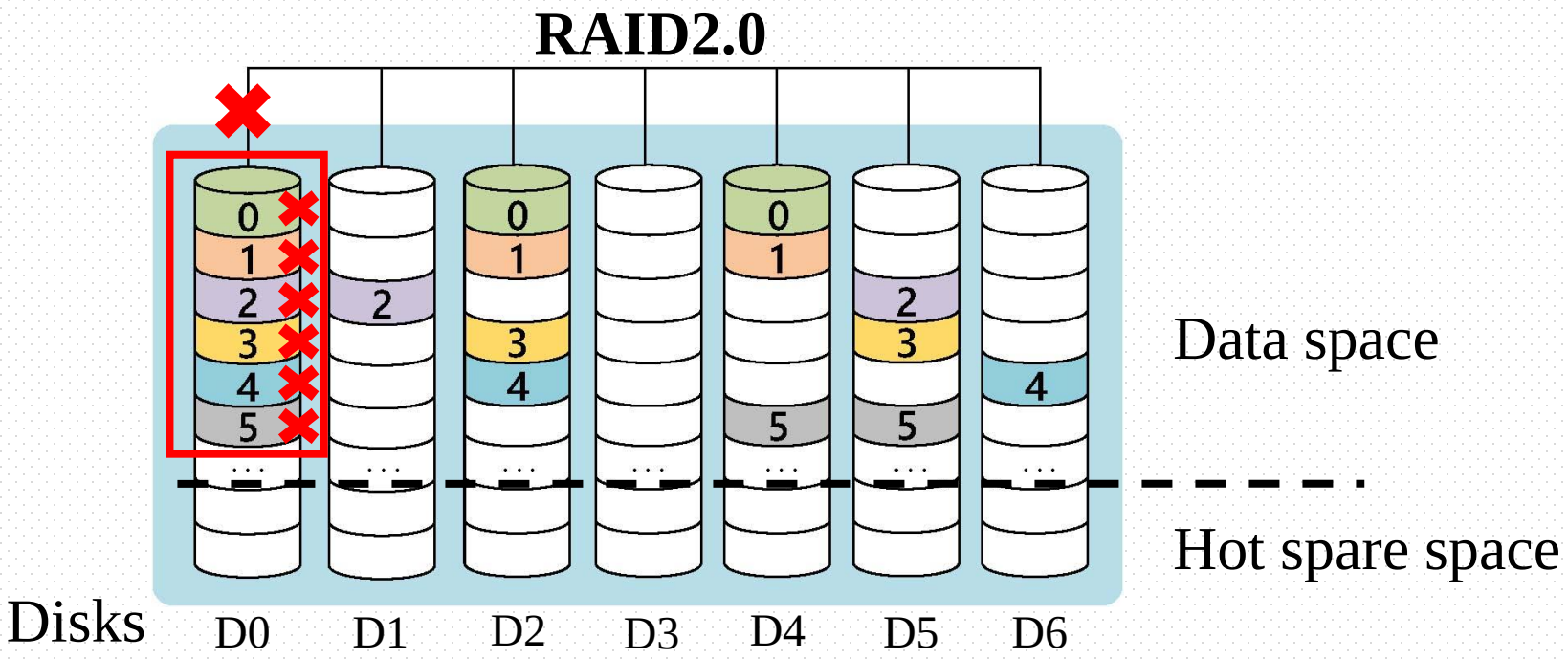
Step 1 – Batching Tasks Out of Order

- Initialize the batch with successive serial numbers
- Get read load of each disk and calculate λ_r
- If $\lambda_r \neq 1$
  - Select a task t_r from the current batch to be replaced
  - Select a task t_l from the remaining pending tasks
  - Replace t_r with t_l
- Keep replacing until $\lambda_r = 1$ or there are no satisfying replacements.

Note: The replacement rule is explained with the following example.

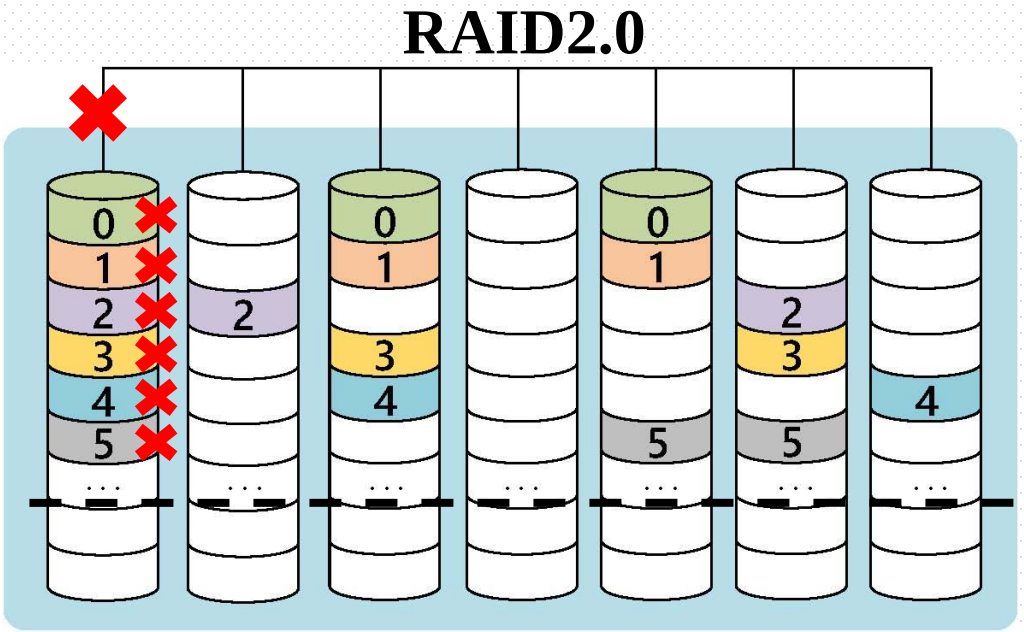
Step 1 – Batching Tasks Out of Order

- Initialize the batch with successive serial numbers



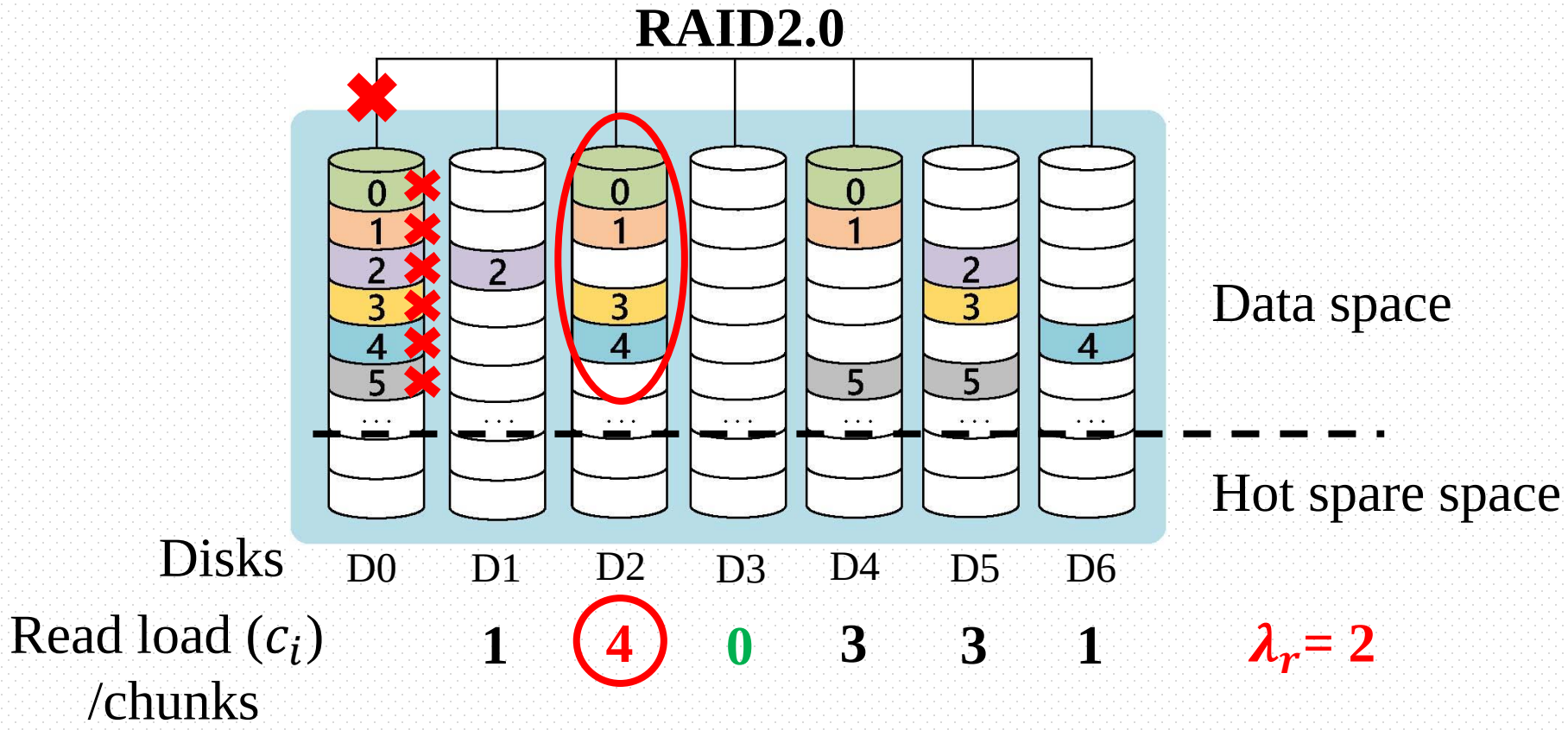
Step 1 – Batching Tasks Out of Order

- Get read load of each disk and calculate λ_r



Step 1 – Batching Tasks Out of Order

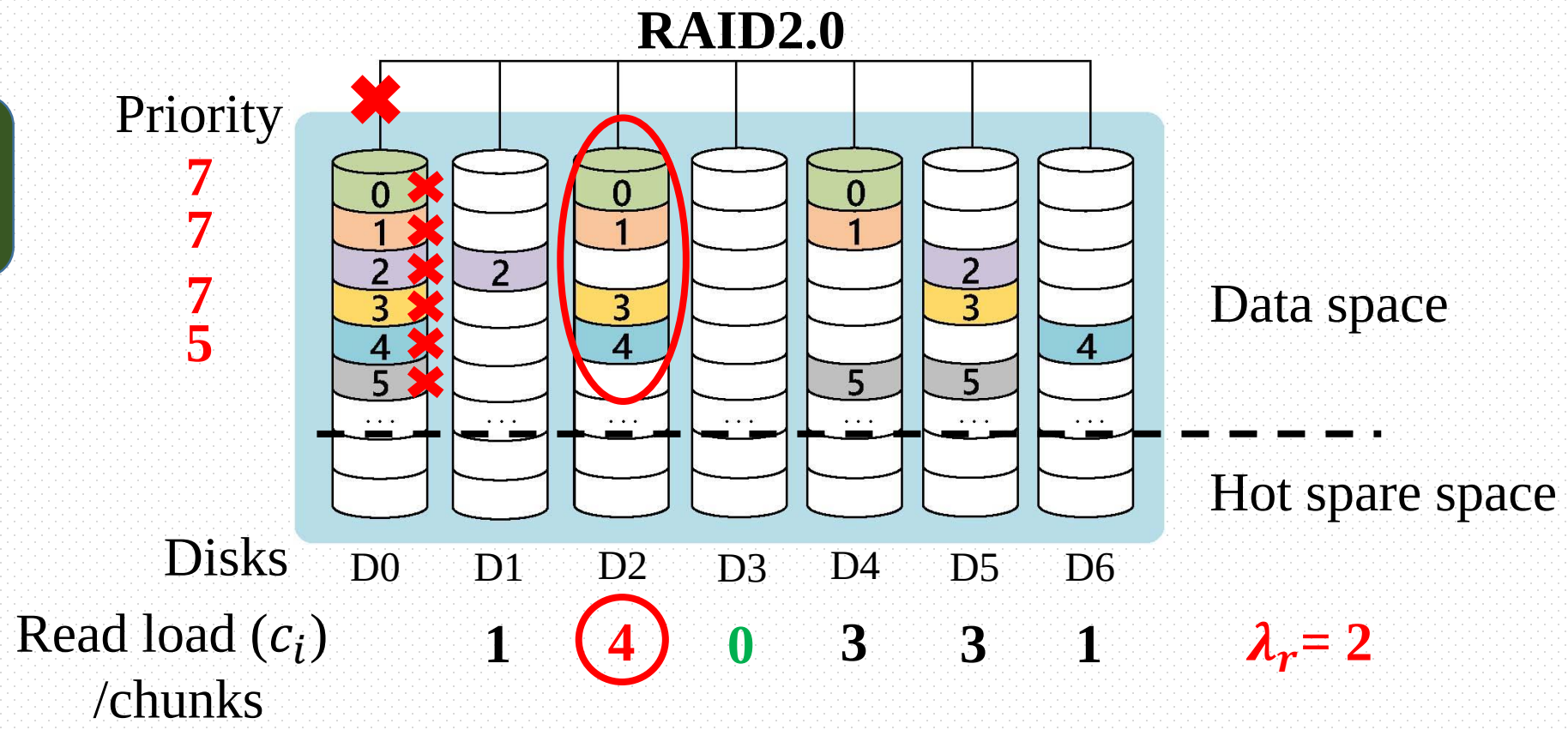
- Select a task t_r from the current batch to be replaced



Step 1 – Batching Tasks Out of Order

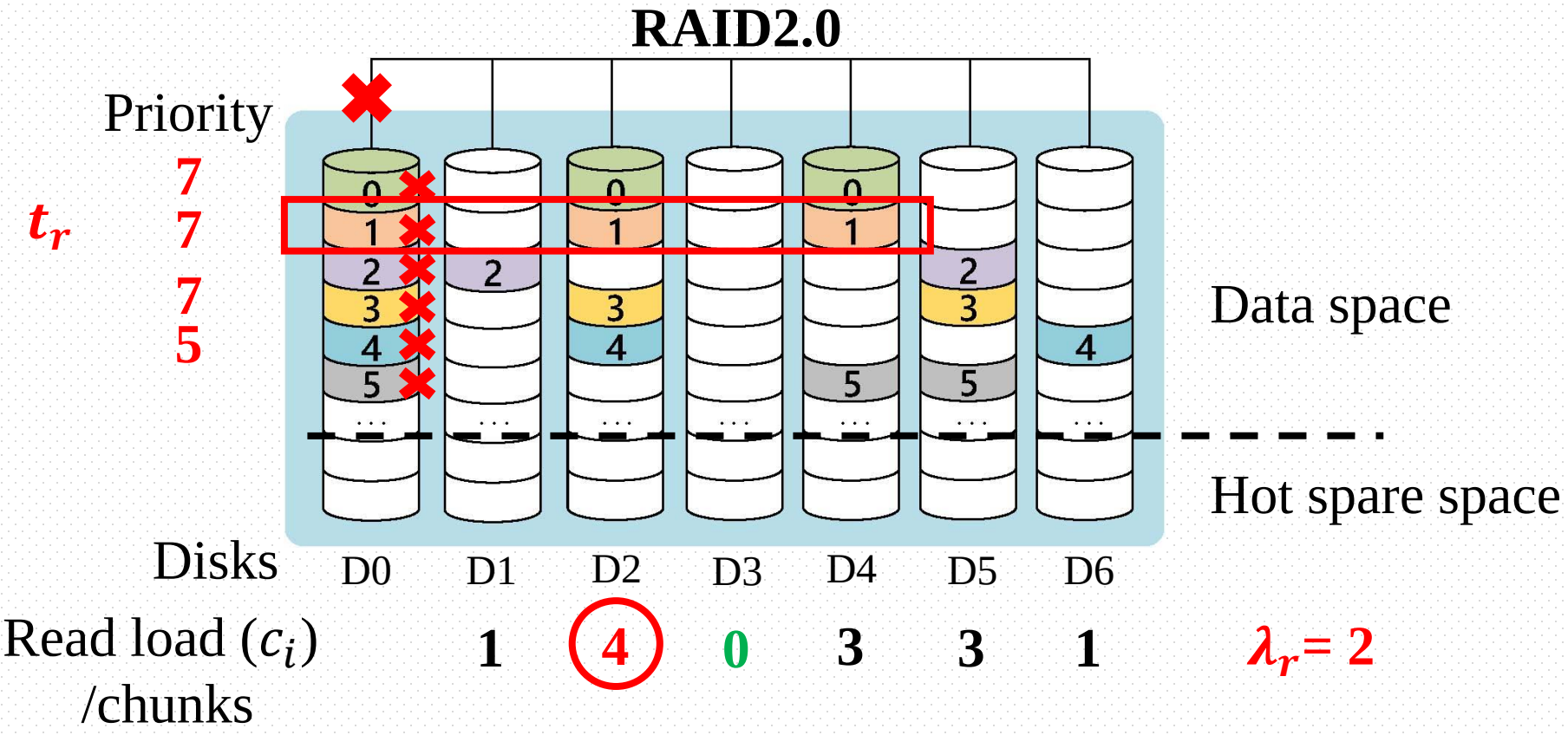
- Select a task t_r from the current batch to be replaced

$$p_j = \sum_{i=1}^{k} c_{j_i}$$



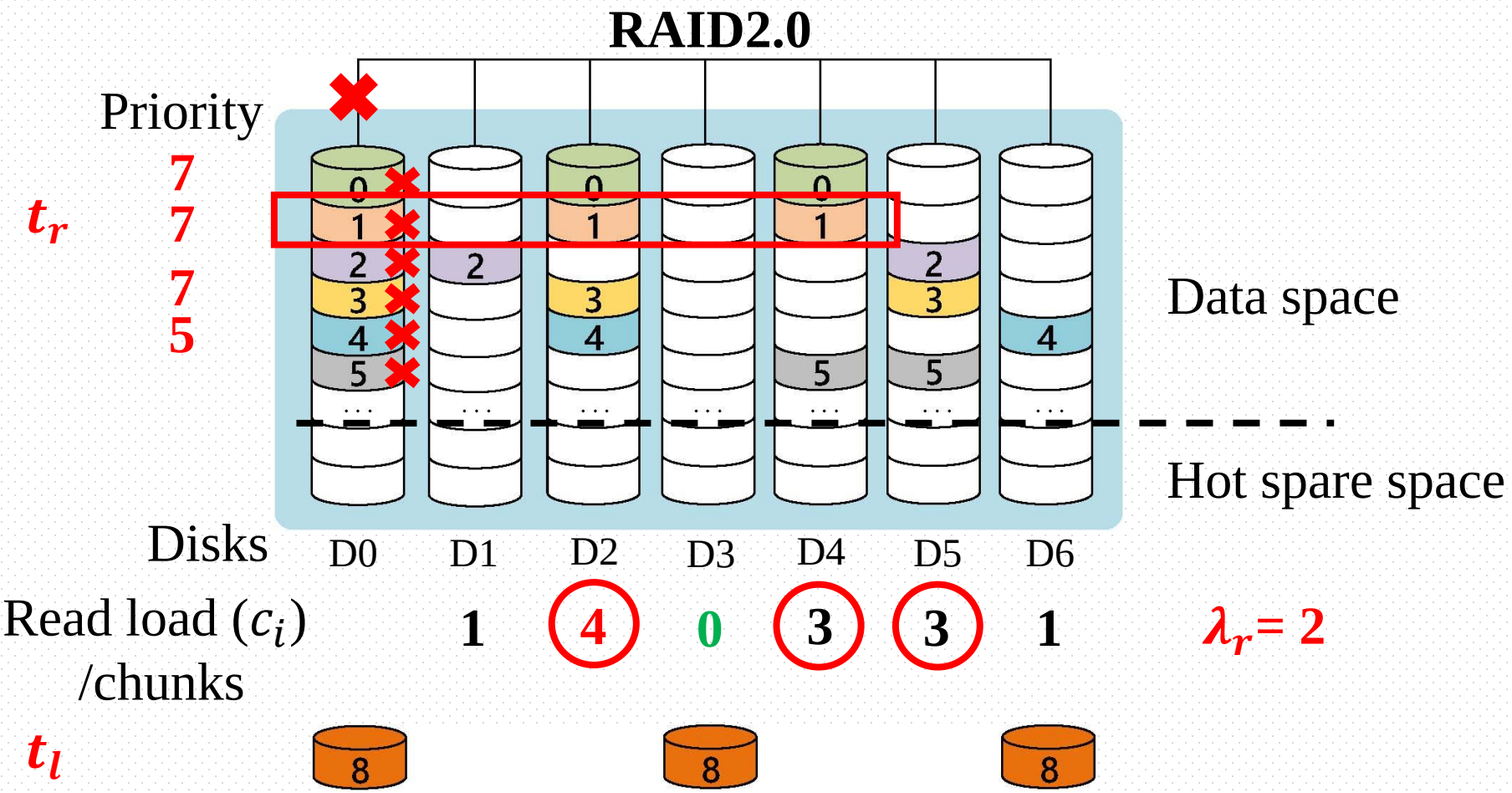
Step 1 – Batching Tasks Out of Order

- Select a task t_r from the current batch to be replaced



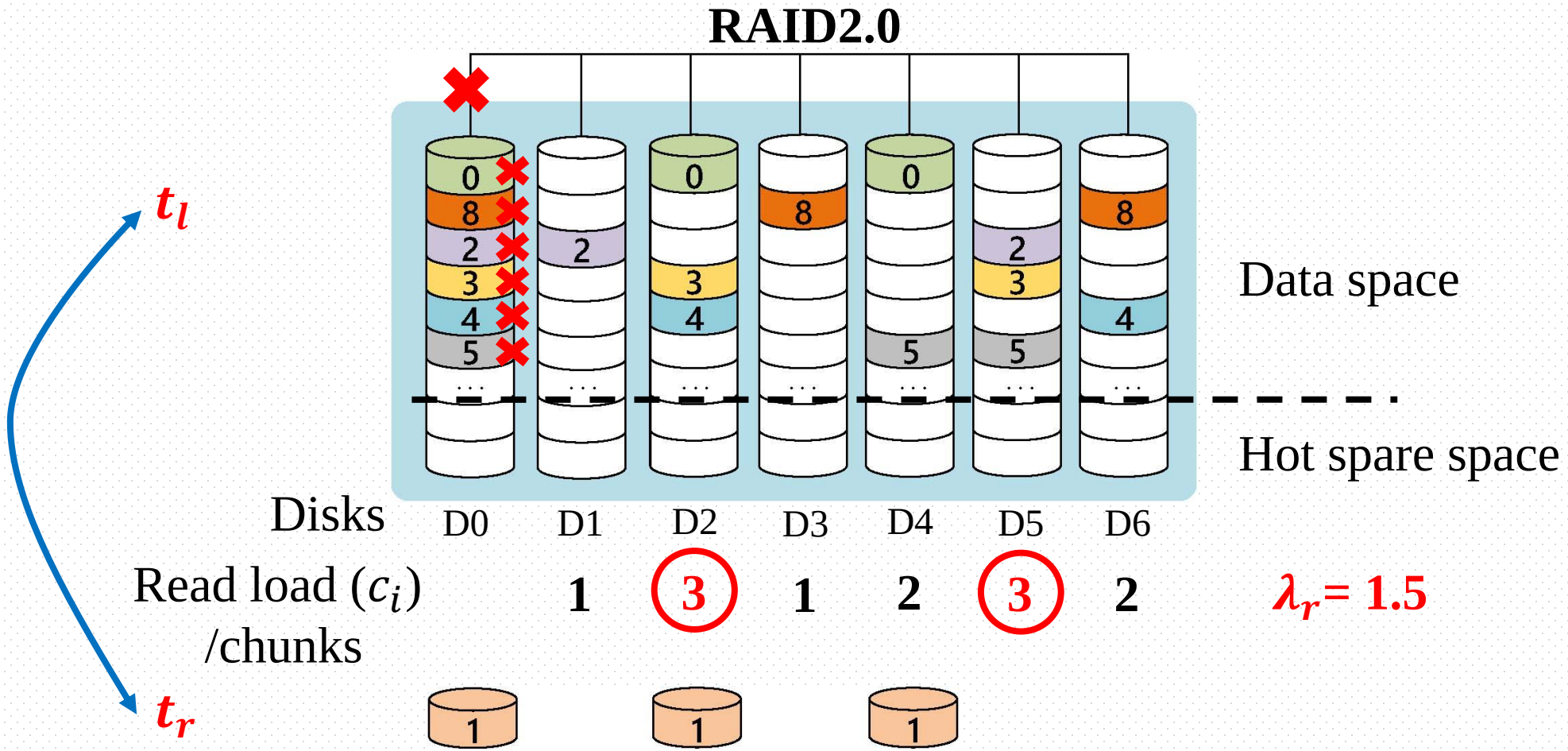
Step 1 – Batching Tasks Out of Order

- Select a task t_l from the remaining pending tasks



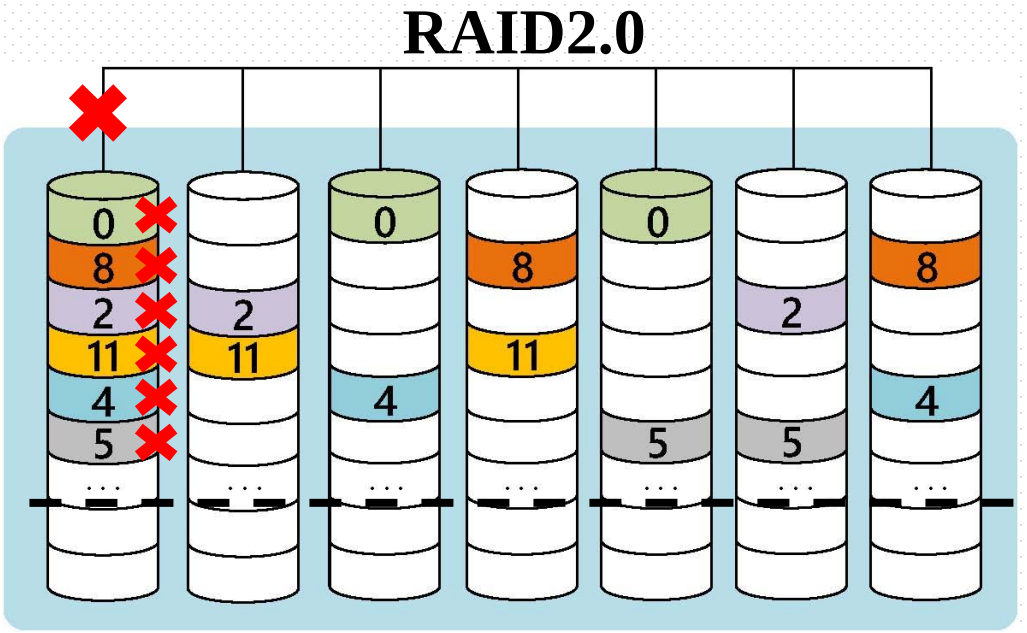
Step 1 – Batching Tasks Out of Order

- Replace t_r with t_l



Step 1 – Batching Tasks Out of Order

- Keep replacing until $\lambda_r = 1$ or there are no satisfying replacements

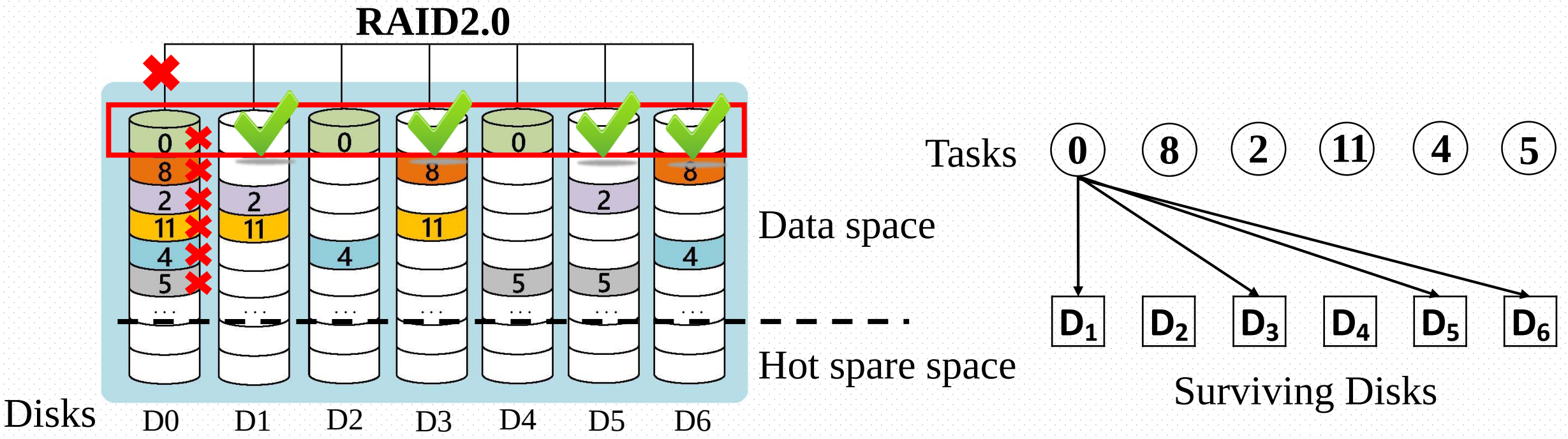


Step 2 – Selecting Surviving Disks

- Construct the bipartite graph based on the selected batch of tasks
- Find a maximum matching
- Distribute the reconstructed chunks to disks based on the maximum matching

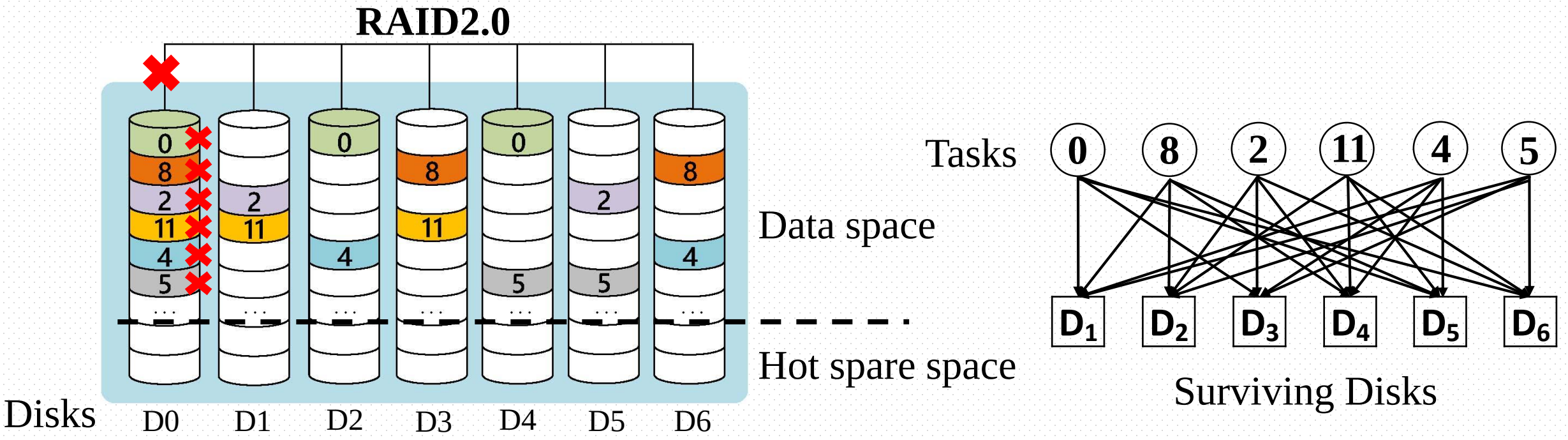
Step 2 – Selecting Surviving Disks

- Construct the bipartite graph based on the selected batch of tasks



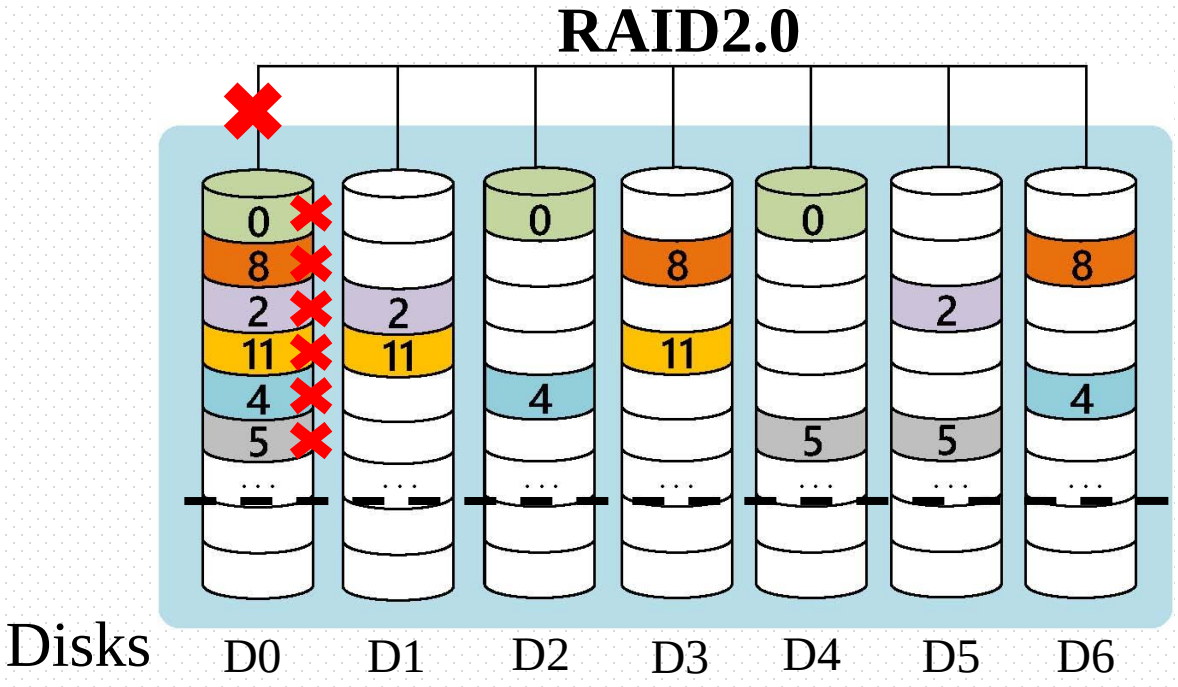
Step 2 – Selecting Surviving Disks

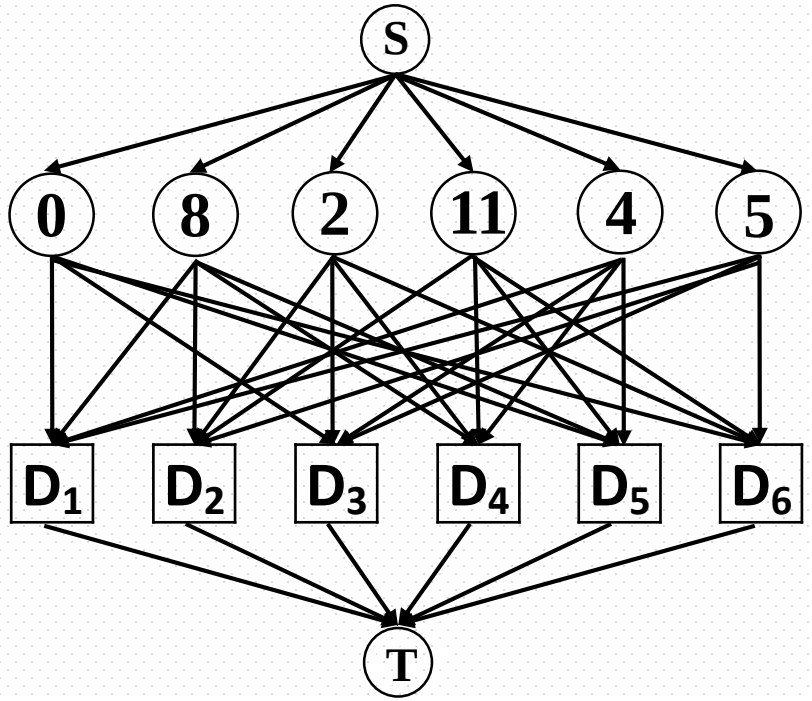
- Construct the bipartite graph based on the selected batch of tasks



Step 2 – Selecting Surviving Disks

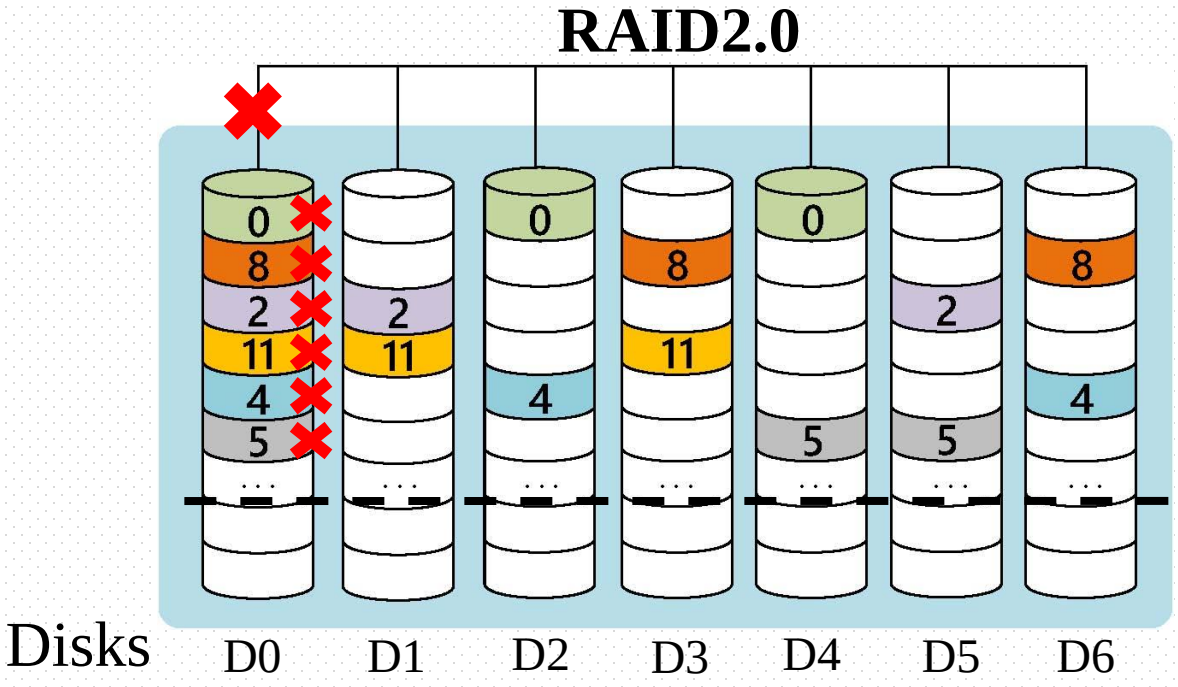
- Find a maximum matching

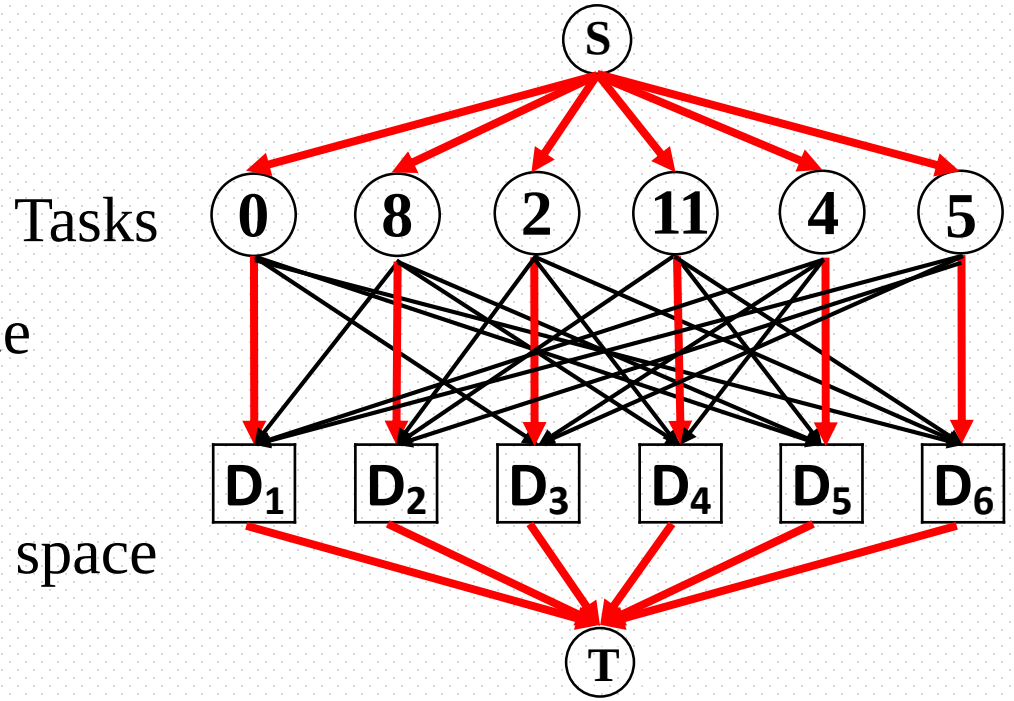




Step 2 – Selecting Surviving Disks

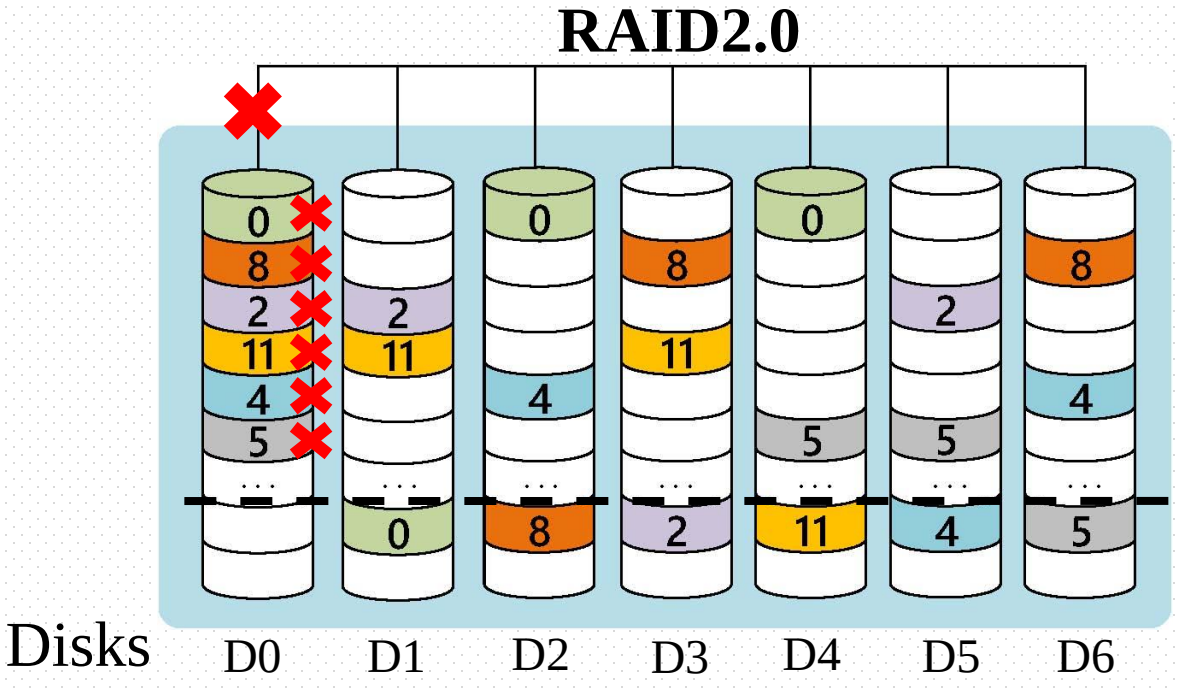
- Find a maximum matching

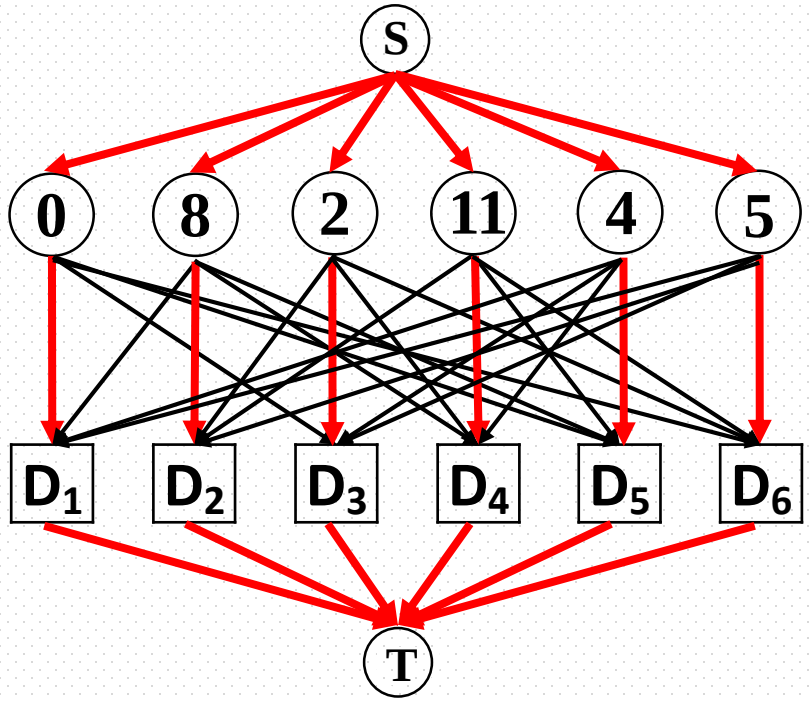




Step 2 – Selecting Surviving Disks

- Distribute the reconstructed chunks to disks based on the maximum matching





Heterogeneous Rebuilding Bandwidth

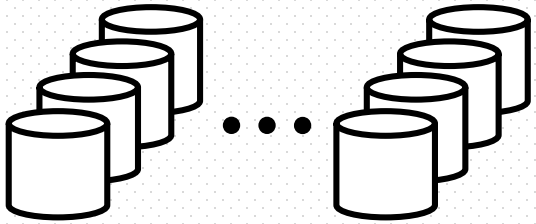
- Get the available rebuilding bandwidth b_i of disk D_i
- Add weights based on b_i into Step 1 and Step 2
  - Substitute the counter c_i for c_i/b_i in Step 1
  - Change the capacity of edges connecting to sink T in Step 2

Evaluation

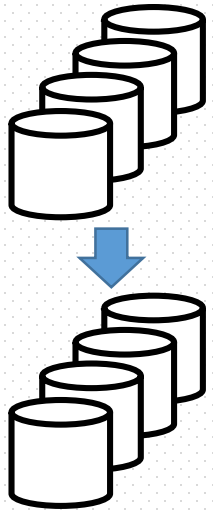
- Hardware setup
  - Local cluster of 11 nodes interconnected through a InfiniBand Switch
  - Use iSER to implement the connection of large-scale disk pool
  - Hardware details of each node
    - 64GB memory, 48 cores, a 7200-RPM 1TB SATA hard disk
- Methodology
  - Default configuration
    - 50 disks, 6+1 RAID group, chunk size of 64MB, batch size of 49
  - Single failure reconstruction

Baselines

- 2 randomized data placement schemes: $RAID_R$ and $RAID_H$
  - $RAID_R$: Use system time as random seed
  - $RAID_H$: Randomize based on Jenkins hash function
- 2 conventional organizations: RAID50 and RAID5C



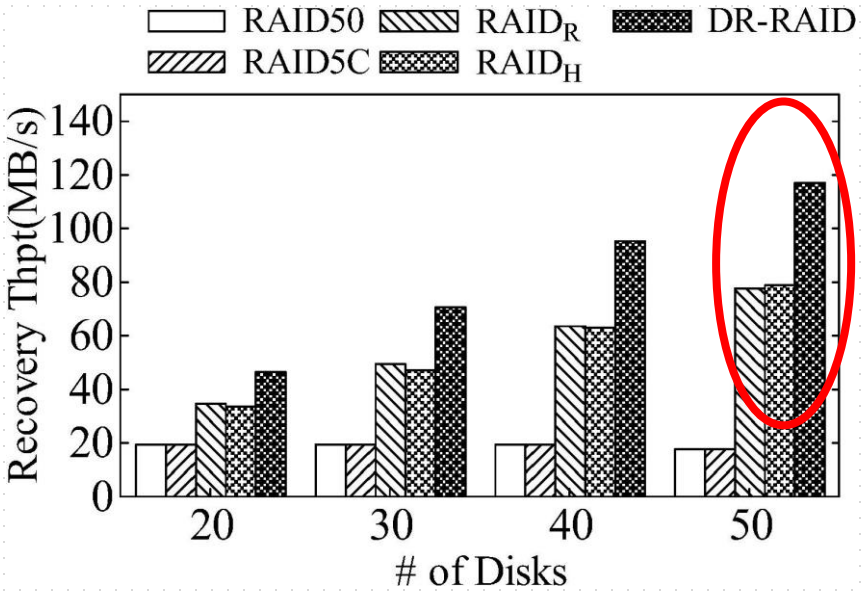
RAID50



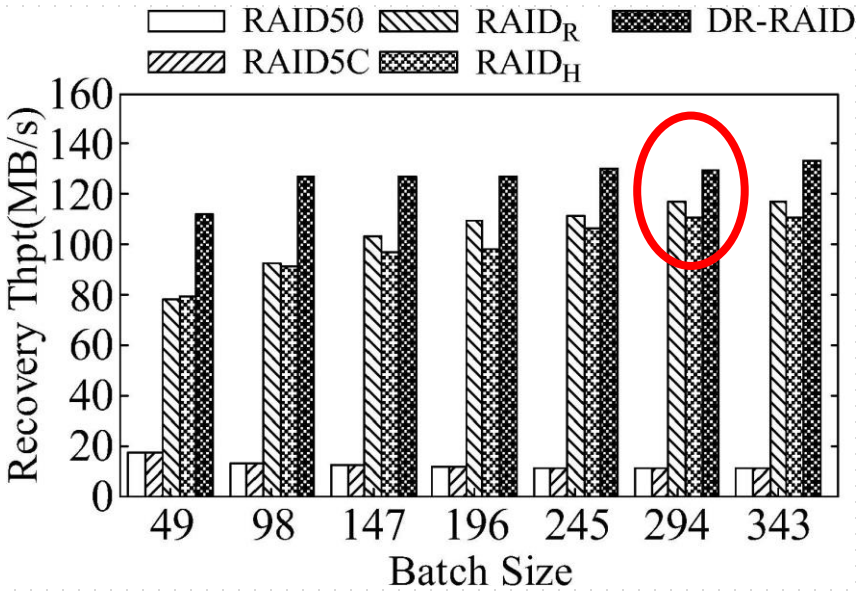
RAID5C

Offline Rebuilding

- Recovery Throughput
  - Varying disk pool sizes
    - 51% improvement of $RAID_R$ & 6.6x of RAID5C for 50 disks
  - Varying batch sizes
    - 11% improvement of $RAID_R$ even when batch size is 294



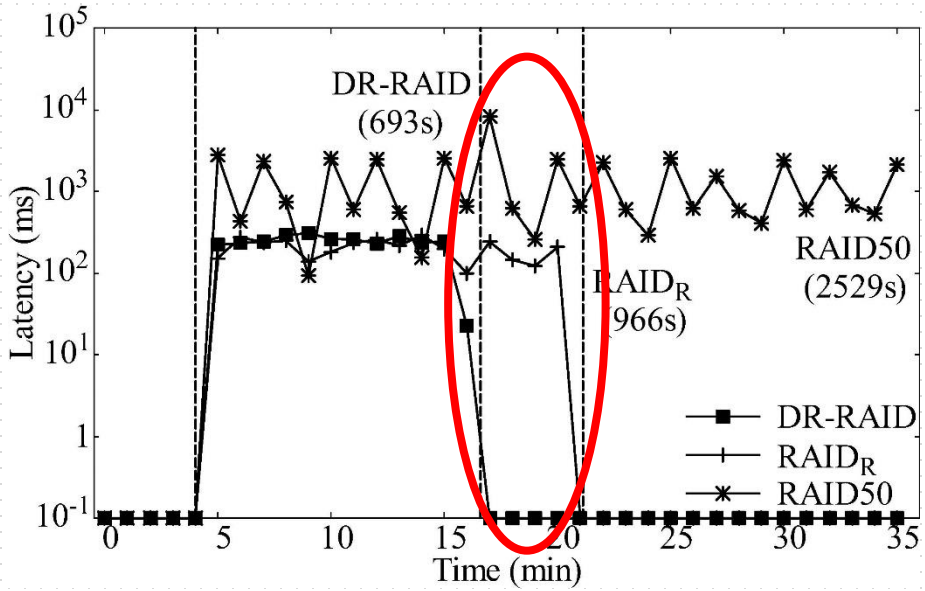
Varying disk pool sizes



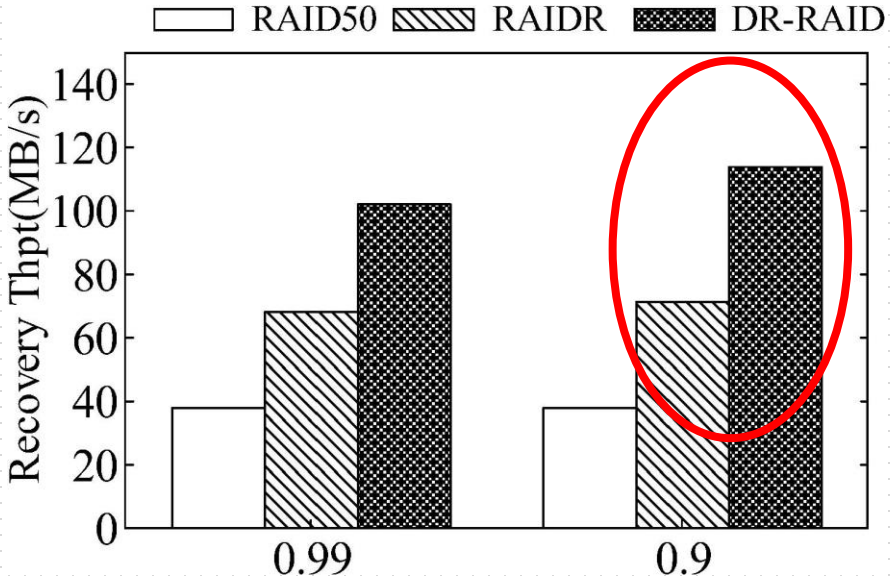
Varying batch sizes

Online Rebuilding

- Request latency
  - The period of performance degradation is shortened by 28%
- Rebuilding throughput with varied rebuilding bandwidth
  - 59% improvement of $RAID_R$ & 2.0x of RAID50



Request latency of replaying WebSearch2



Heterogeneous rebuilding

Conclusion

- Local load imbalance within a recovery batch slows down reconstruction process
- DR-RAID selectively packs recovery tasks into a batch to achieve reconstruction load balance
- DR-RAID increases rebuilding throughput compared with the random data placement scheme in offline rebuilding and varied rebuilding bandwidth

Thanks for your attention!

Q&A

Qiliang Li@USTC
leeql@mail.ustc.edu.cn