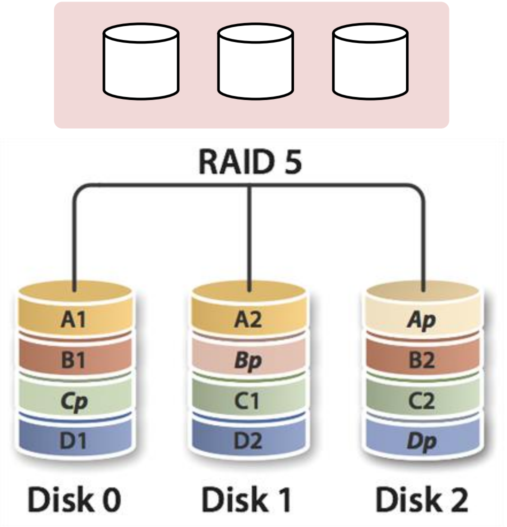
RAID+: Deterministic and Balanced Data Distribution for Large Disk Enclosures

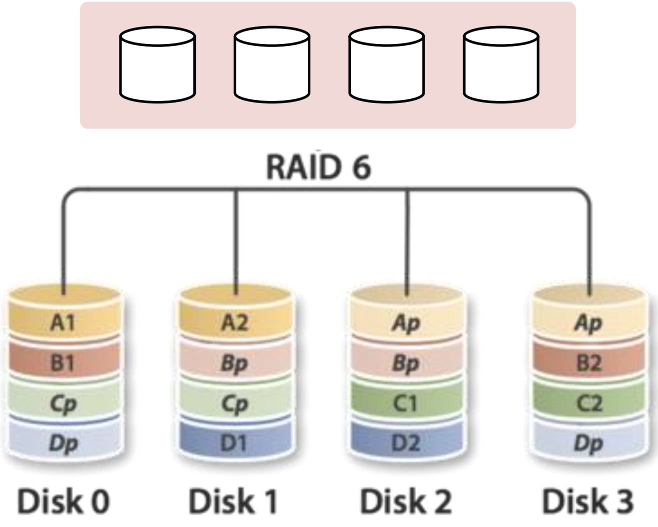
Guangyan Zhang, Zican Huang, *Xiaosong Ma*
SonglinYang, Zhufan Wang, Weimin Zheng

Tsinghua University
Qatar Computing Research Institute, HBKU

RAID with Large Disk Pools

- RAID (Redundant Array of Inexpensive Disks) widely used





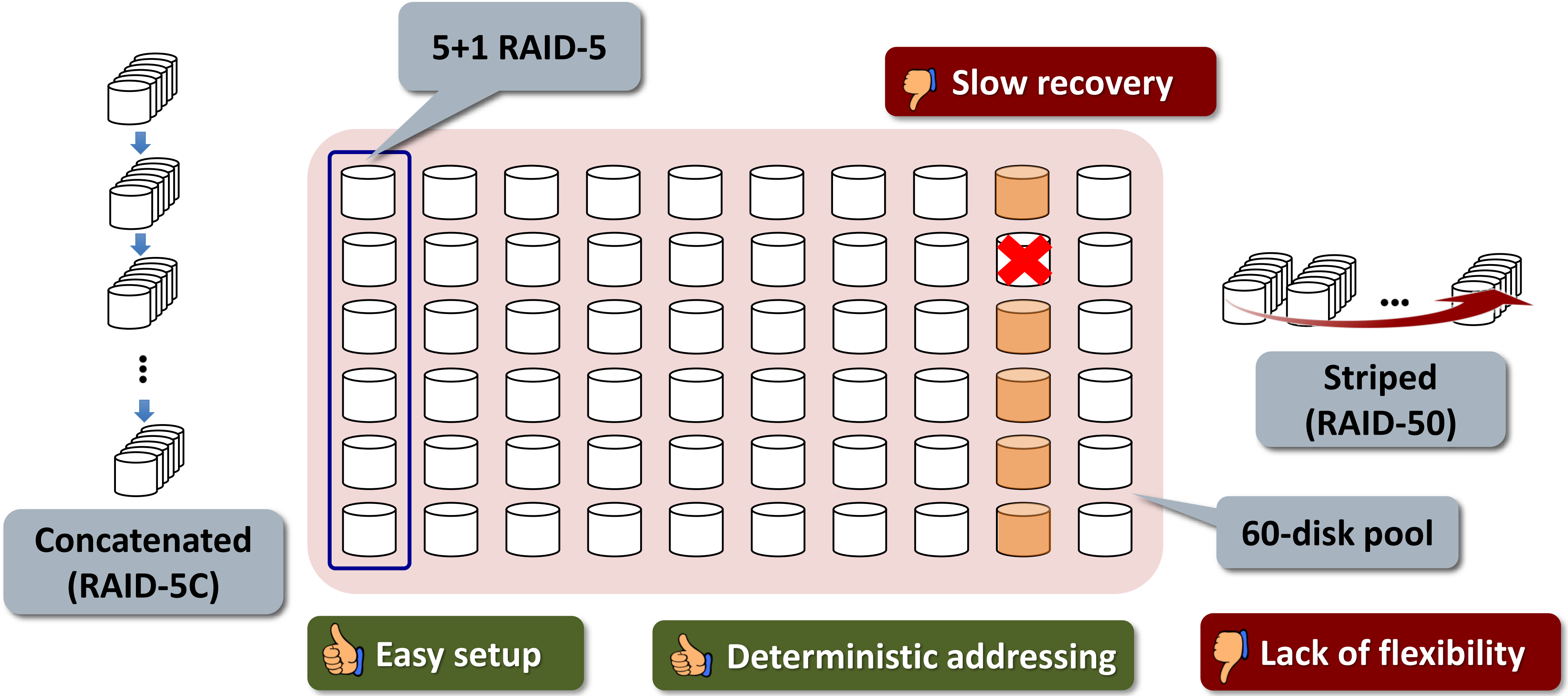
- Not designed to work directly on large arrays

HGST 4U60G2

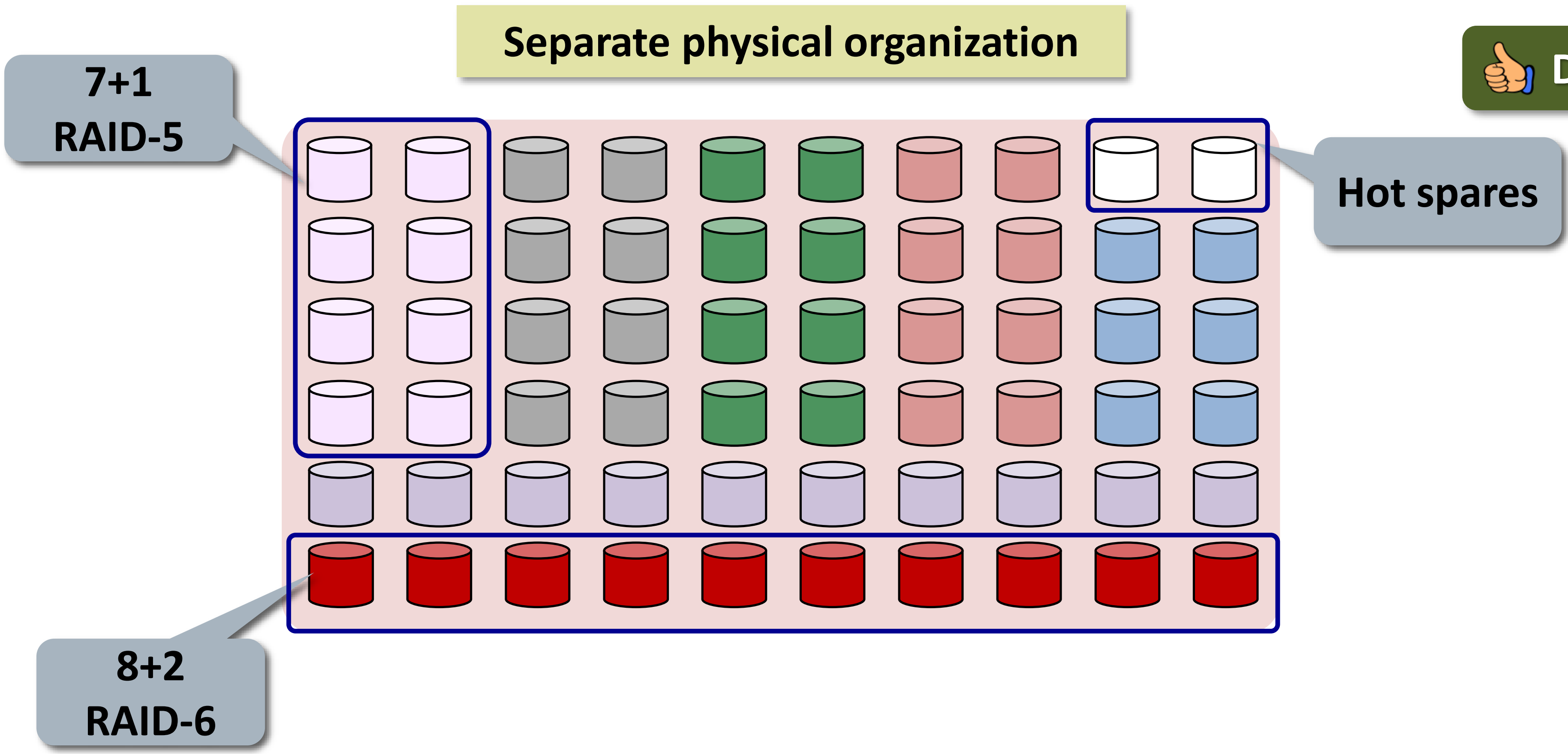
EMC VMAX3

Dell PowerVault MD3060e

Existing Ways of Using Large Disk Pools (1): Integrating Homogeneous RAID Groups



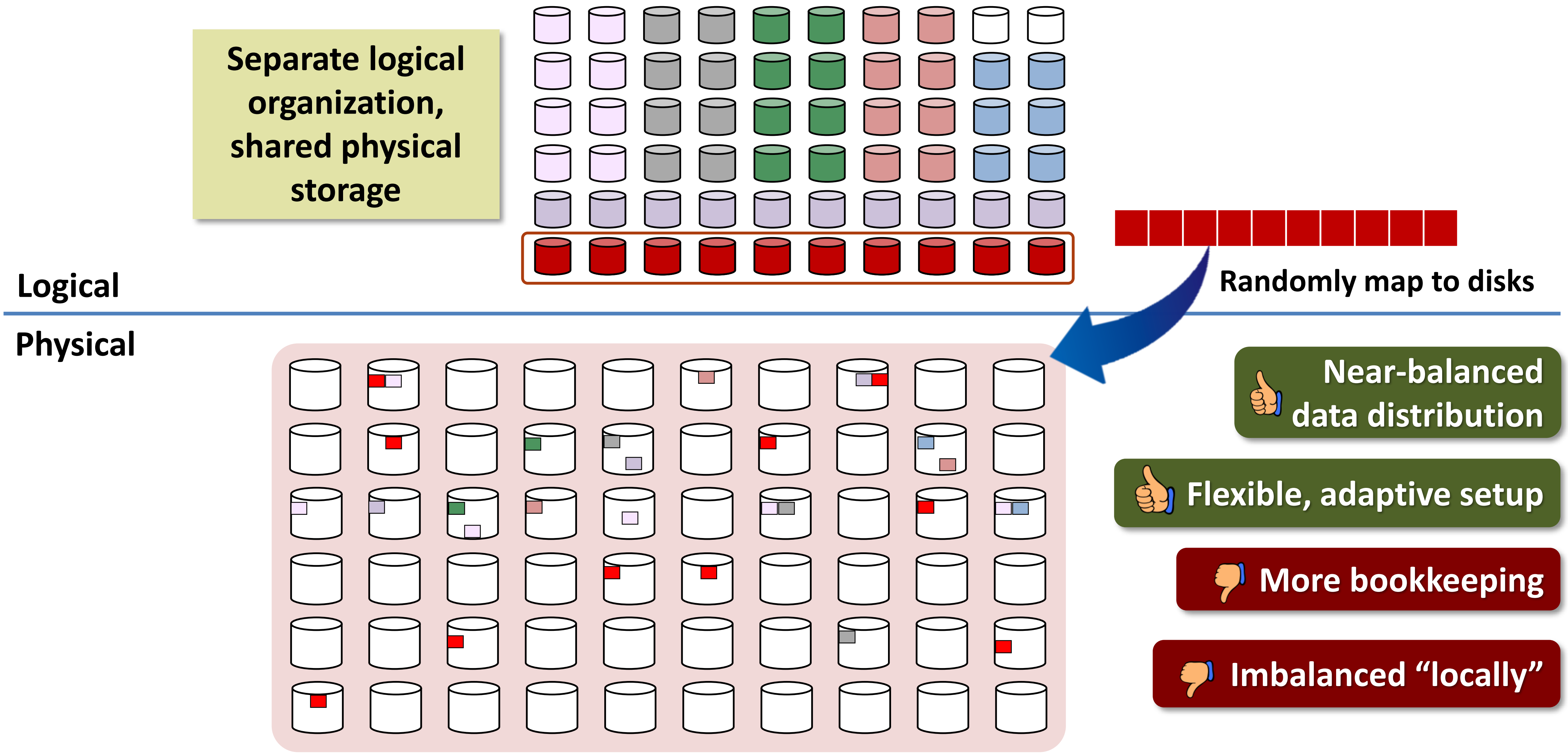
Existing Ways of Using Large Disk Pools (2): Building Heterogeneous RAID Groups



- 👍 Easy setup
- 👍 Deterministic addressing
- 👍 Strong isolation
- 👎 Low peak performance
- 👎 Slow recovery
- 👎 Lack of flexibility

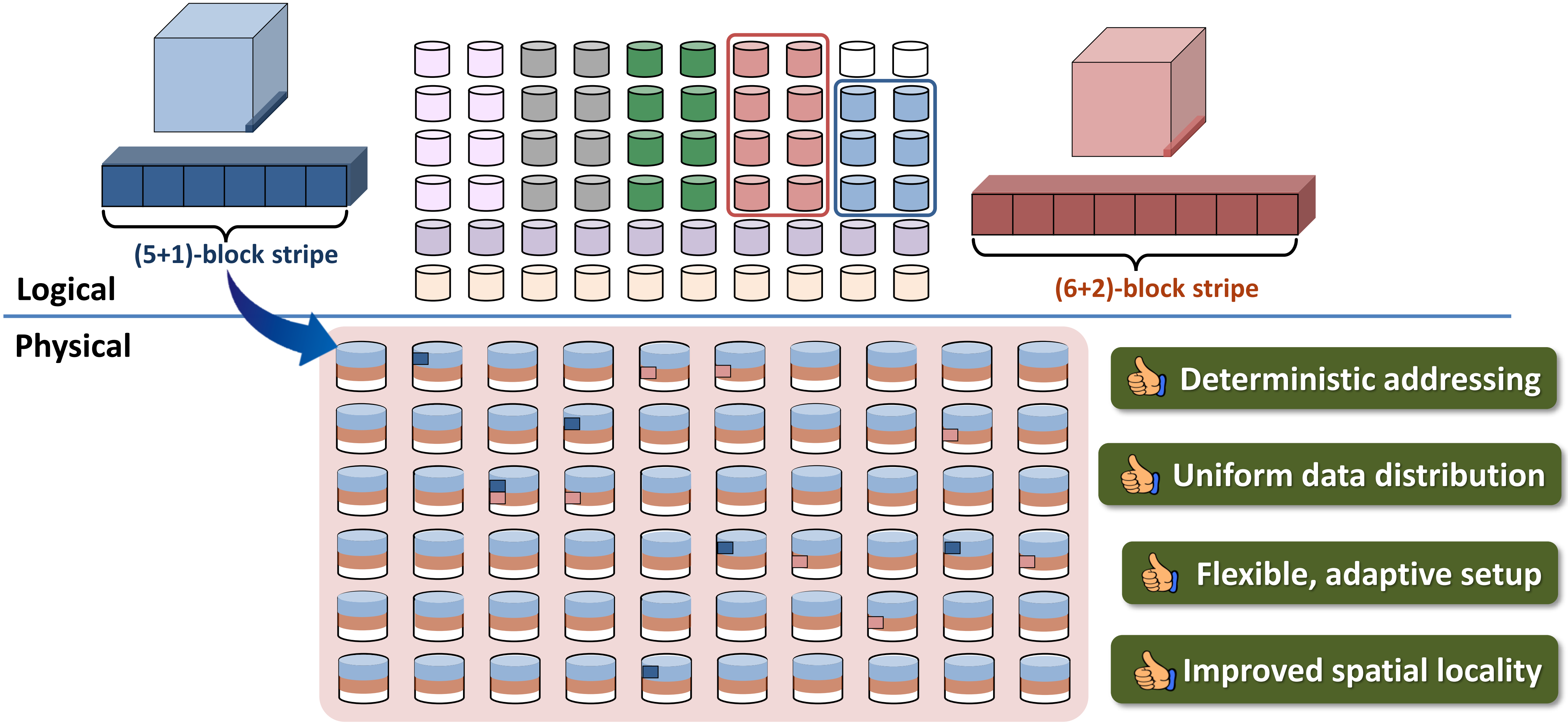
Existing Ways of Using Large Disk Pools (3): Random Block Placement

Separate logical organization, shared physical storage

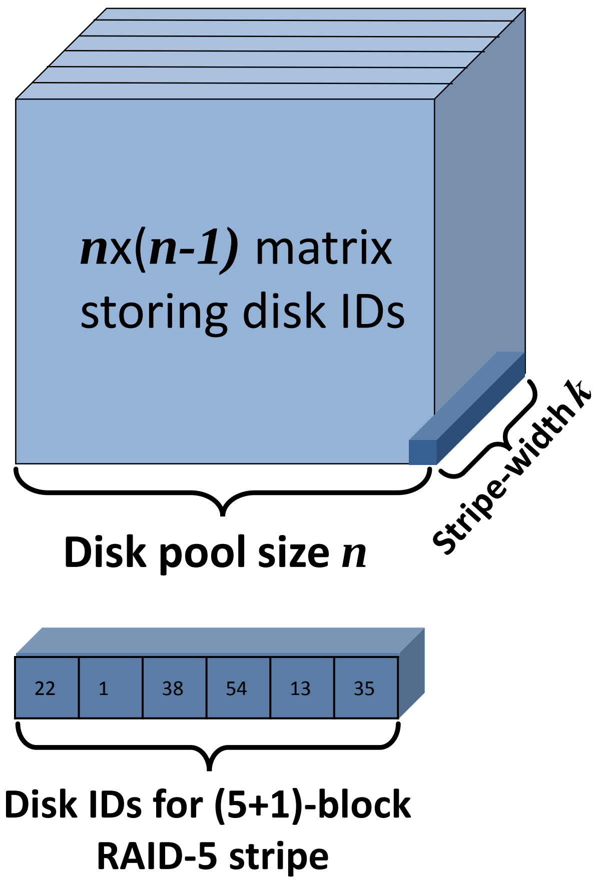


- 👍 Near-balanced data distribution
- 👍 Flexible, adaptive setup
- 👎 More bookkeeping
- 👎 Imbalanced “locally”

RAID+: Guaranteed Uniform Distribution using 3D templates

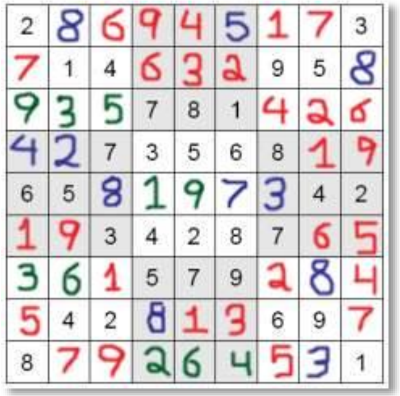


RAID+ 3D Template



- Deterministic mapping of ***k***-block stripes to ***n***-disk pool
  - ***k*** decoupled from ***n***
- Properties (see paper for details and proofs)
  - Fault tolerance of RAID
    - ✓ No blocks in stripe placed on same disk
  - *Guaranteed* load balance
    - ✓ Uniform distribution of data blocks, for *both application I/O and recovery*
  - Ease of maintenance
    - ✓ Deterministic, computable addressing
- **Achieved by using *Mutually Orthogonal Latin Squares (MOLS)***

Introducing Latin Squares



- ***Latin square*** of order ***n***
  - $n \times n$ array filled with n different symbols
  - Each symbol occurring exactly once in each row and column

- 2 Latin squares ***orthogonal***, if when superimposed, each of n^2 ordered pairs appears exactly once

1	2	3	4	5
2	3	4	5	1
3	4	5	1	2
4	5	1	2	3
5	1	2	3	4

+

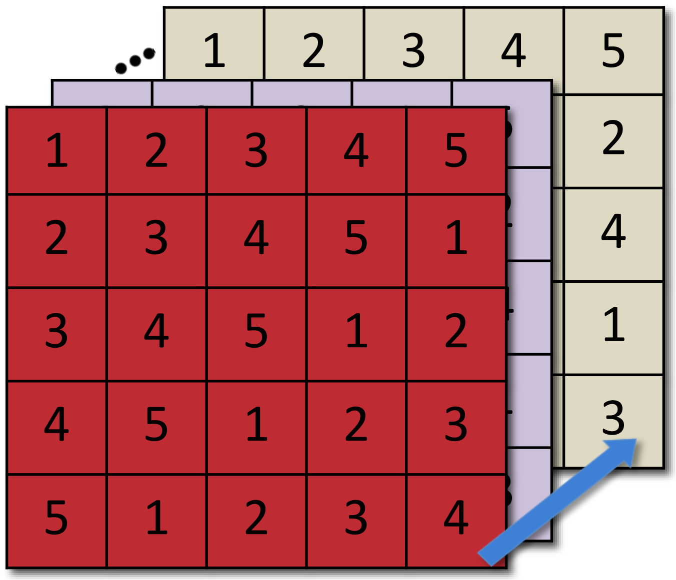
1	2	3	4	5
3	4	5	1	2
5	1	2	3	4
2	3	4	5	1
4	5	1	2	3

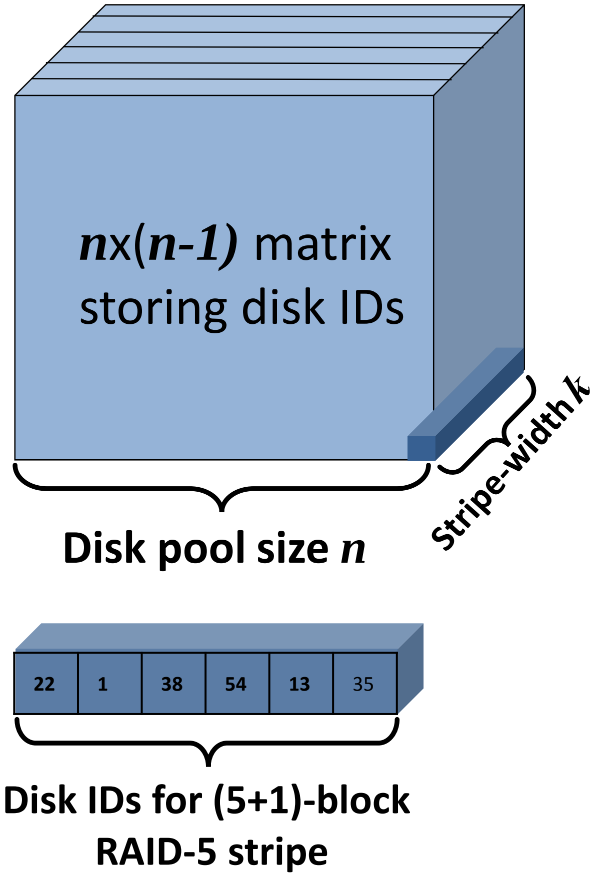
→

1,1	2,2	3,3	4,4	5,5
2,3	3,4	4,5	5,1	1,2
3,5	4,1	5,2	1,3	2,4
4,2	5,3	1,4	2,5	3,1
5,4	1,5	2,1	3,2	4,3

- ***Mutually Orthogonal Latin Squares (MOLS)***
  - Set of Latin squares, *mutually orthogonal* ***pair wise***

Stacking MOLS into RAID+ Template





- Do we have enough MOLS?
  - *Number of MOLS* open question
  - n-1 when n is power of prime number
    - ***Valid RAID+ pool sizes***
  - For the i^{th} square L_i, $L_i[x,y] = i \cdot x + y$

MOLS-based Addressing

$L_i[x,y] = i \cdot x + y$

L_1

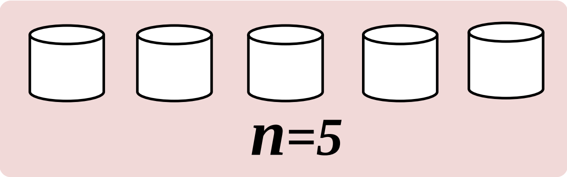
L_2

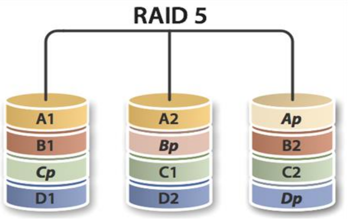
L_3

$k=3$ **MOLS**

a	1	2	3
b	2	3	4
c	3	4	0
d			
e			
f			
g			
h			
i			
j			
k			
l			
m			
n			
o			
P			
q			
r			
s			
t			

$n(n-1)=20$ **stripes**





$k=3$

D_0	D_1	D_2	D_3	D_4
c	a	a	a	b
		b	b	c
			c	

$(n-1)k=12$ **blocks per disk**

MOLS-based Addressing (n=5, k=3)

0	1	2	3	4
1	2	3	4	0
2	3	4	0	1
3	4	0	1	2
4	0	1	2	3

0	1	2	3	4
2	3	4	0	1
4	0	1	2	3
1	2	3	4	0
3	4	0	1	2

0	1	2	3	4
3	4	0	1	2
1	2	3	4	0
4	0	1	2	3
2	3	4	0	1

***K*=3 MOLS**

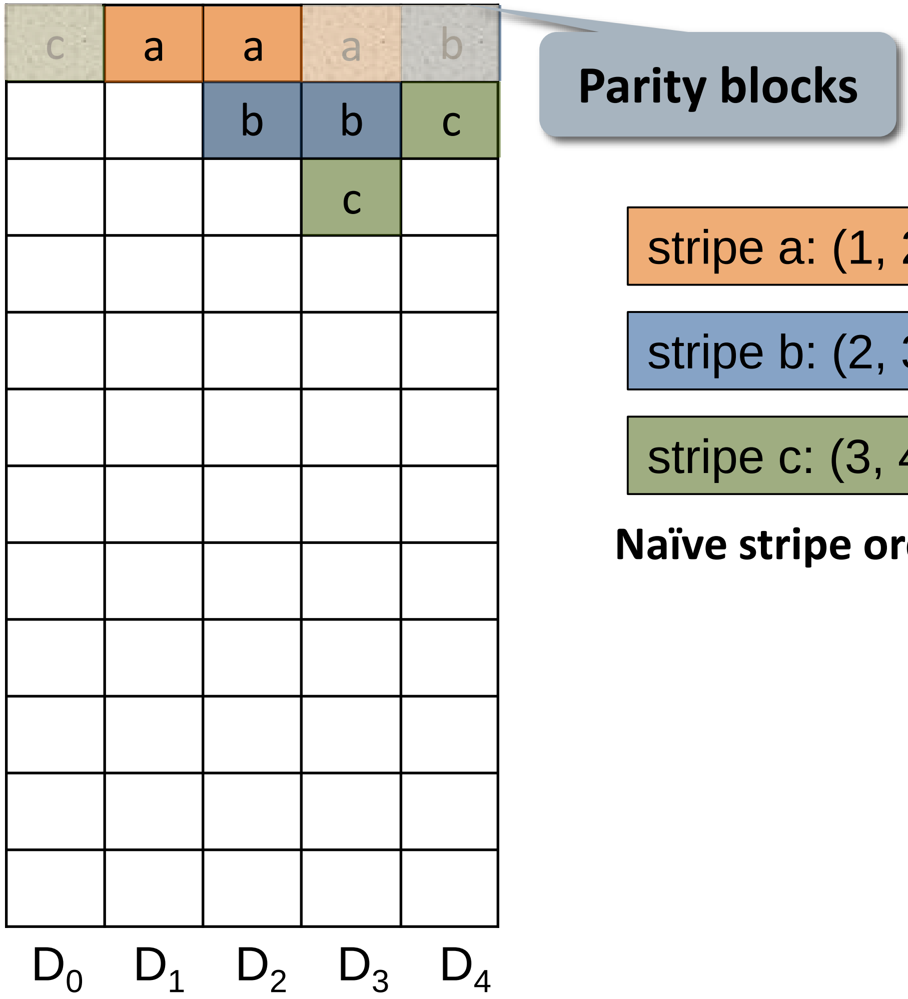
a	1	2	3
b	2	3	4
c	3	4	0
d	4	0	1
e	0	1	2
f	2	4	1
g	3	0	2
h	4	1	3
i	0	2	4
j	1	3	0
k	3	1	4
l	4	2	0
m	0	3	1
n	1	4	2
o	2	0	3
P	4	3	2
q	0	4	3
r	1	0	4
s	2	1	0
t	3	2	1

***n(n-1)=20* stripes**

c	a	a	a	b
d	d	b	b	c
e	e	e	c	d
g	f	f	g	f
i	h	g	h	h
j	j	i	j	i
l	k	l	k	k
m	m	n	m	l
o	n	o	o	n
q	r	p	q	q
r	s	s	q	q
s	t	t	t	r
D_0	D_1	D_2	D_3	D_4

***(n-1)k=12* blocks per disk**

Impact of Stripe Ordering



(n-1)k=12 blocks per disk

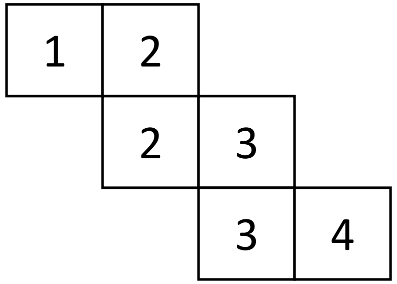
stripe a: (1, 2, 3)

stripe b: (2, 3, 4)

stripe c: (3, 4, 0)

Naïve stripe ordering

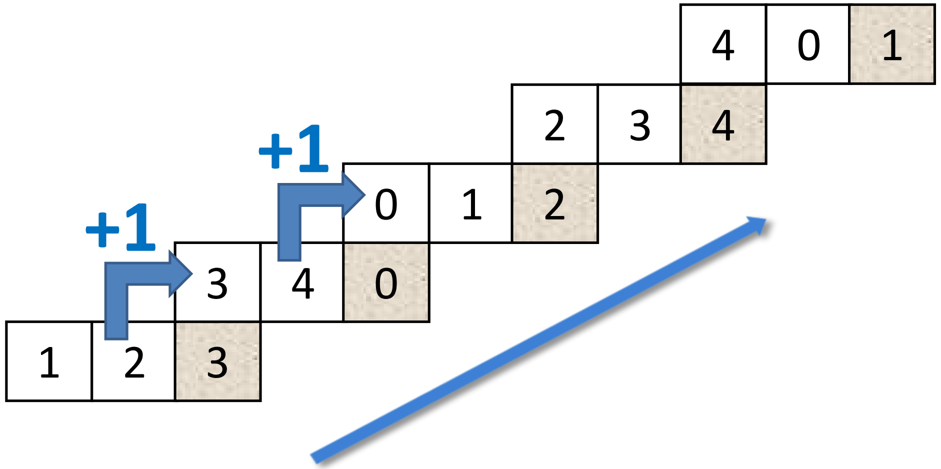
Sequential read of 6 blocks =>
Disk access sequence:



Highly overlapped

Sequential Access Friendly Stripe Ordering (I)

- <u>Sequential read</u> friendly ordering
  - ***Intuition***: traverse all disks (<u>ignoring</u> parity blocks)



0	1	2	3	4
1	2	3	4	0
2	3	4	0	1
3	4	0	1	2
4	0	1	2	3

0	1	2	3	4
2	3	4	0	1
4	0	1	2	3
1	2	3	4	0
3	4	0	1	2

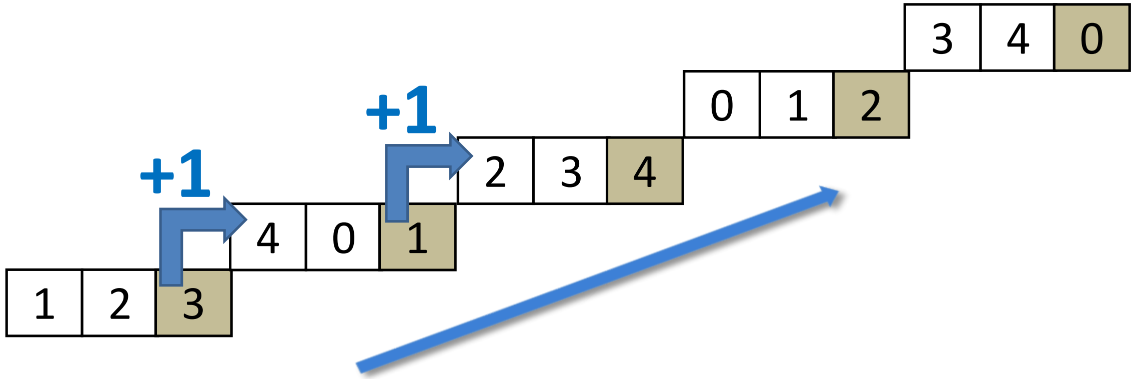
0	1	2	3	4
3	4	0	1	2
1	2	3	4	0
4	0	1	2	3
2	3	4	0	1

a	1	2	3
b	2	3	4
c	3	4	0
d	4	0	1
e	0	1	2
f	2	4	1
g	3	0	[illegible]
h	4	1	[illegible]
i	0	2	[illegible]
j	1	3	0
k	3	1	4
l	4	2	0
m	0	3	1
n	1	4	2
o	2	0	3
p	4	3	2
q	0	4	3
r	1	0	4
s	2	1	0
t	3	2	1

R-friendly ordering:
A-C-E-B-D

Sequential Access Friendly Stripe Ordering (II)

- <u>Sequential write</u> friendly ordering
  - ***Intuition***: traverse all disks (<u>including</u> parity blocks)



0	1	2	3	4
1	2	3	4	0
2	3	4	0	1
3	4	0	1	2
4	0	1	2	3

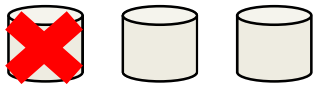
0	1	2	3	4
2	3	4	0	1
4	0	1	2	3
1	2	3	4	0
3	4	0	1	2

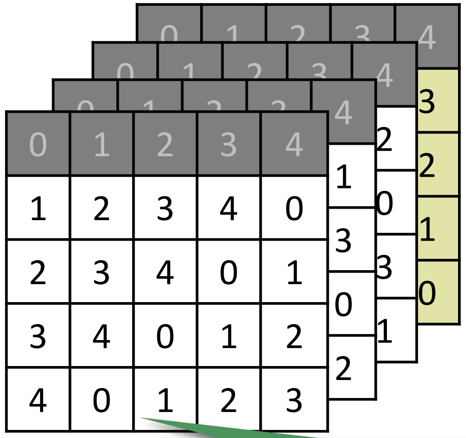
0	1	2	3	4
3	4	0	1	2
1	2	3	4	0
4	0	1	2	3
2	3	4	0	1

a	1	2	3
b	2	3	4
c	3	4	0
d	4	0	1
e	0	1	2
f	2	4	1
g	3	0	[illegible]
h	4	1	[illegible]
i	0	2	[illegible]
j	1	3	0
k	3	1	4
l	4	2	0
m	0	3	1
n	1	4	2
o	2	0	3
p	4	3	2
q	0	4	3
r	1	0	4
s	2	1	0
t	3	2	1

W-friendly ordering:
A-D-B-E-C

When Disk Failure Happens





Still orthogonal!

0	1	2	3	4
4	0	1	2	3
3	4	0	1	2
2	3	4	0	1
1	2	3	4	0

Spare square L_4

a	1	2	3
b	2	3	4
c	3	4	0
d	4	0	1
e	0	1	2
f	2	4	1
g	3	0	2
h	4	1	3
i	0	2	4
j	1	3	0
k	3	1	4
l	4	2	0
m	0	3	1
n	1	4	2
o	2	0	3
p	4	3	2
q	0	4	3
r	1	0	4
s	2	1	0
t	3	2	1

a	1	2	3	4
b	2	3	4	0
c	3	4	0 →	1
d	4	0	1	2
e	0	1	2	3
f	2	4	1	3
g	3	0	2	4
h	4	1	3	0
i	0	2	4	1
j	1	3	0	2
k	3	1	4	2
l	4	2	0	3
m	0	3	1	4
n	1	4	2	0
o	2	0	3	1
p	4	3	2	1
q	0	4	3	2
r	1	0	4	3
s	2	1	0	4
t	3	2	1	0

Stripe sees a disk only once
=> replace with disk ID in L_4

RAID+ Interim Layout

a	1	2	3	4
b	2	3	4	0
c	3	4	0	1
d	4	0	1	2
e	0	1	2	3
f	2	4	1	3
g	3	0	2	4
h	4	1	3	0
i	0	2	4	1
j	1	3	0	2
k	3	1	4	2
l	4	2	0	3
m	0	3	1	4
n	1	4	2	0
o	2	0	3	1
p	4	3	2	1
q	0	4	3	2
r	1	0	4	3
s	2	1	0	4
t	3	2	1	0

c	a	a	a	b
d	d	b	b	c
e	e	e	c	d
g	f	f	g	f
i	h	g	h	h
j	j	i	j	i
l	k	l	k	k
m	m	n	m	l
o	n	o	o	n
q	r	p	p	p
r	s	s	q	q
s	t	t	t	r

5 x12 Normal layout

c	a	a	a	b
d	d	b	b	c
e	e	e	c	d
g	f	f	g	f
i	h	g	h	h
j	j	i	j	i
l	k	l	k	k
m	m	n	m	l
o	n	o	o	n
q	r	p	p	p
r	s	s	q	q
s	t	t	t	r
	c	d	e	g
	i	j	l	m
	o	q	r	s

4 x15 interim layout (after loss of 1 disk)

RAID+ Recovery

c	a	a	a	b
d	d	b	b	c
e	e	e	c	d
g	f	f	g	f
i	h	g	h	h
j	j	i	j	i
l	k	l	k	k
m	m	n	m	l
o	n	o	o	n
q	r	p	p	p
r	s	s	q	q
s	t	t	t	r
	c	d	e	g
	i	j	l	m
	o	q	r	s

Pre-allocated space

interim layout
(after loss of 1 disk)

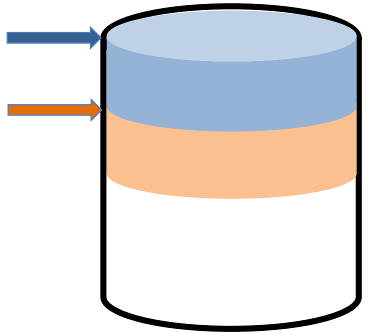
- **Guaranteed** uniform load distribution
  - Read and write evenly on all survivors
  - Fast all-to-all data migration
  - Copy in background to replacement/hotspare
- Repeat for more failures, without hot-spares
  - N >> k in most cases!
- Pre-allocated recovery area
  - Trading capacity for recovery performance
  - Optimized for 1-disk failure
- Maintaining RAID level fault tolerance
  - Optimization for 2+ simultaneous failures

Addressing Metadata Management

c	a	a	a	b
d	d	b	b	c
e	e	e	c	d
g	f	f	g	f
i	h	g	h	h
j	j	i	j	i
l	k	l	k	k
m	m	n	m	l
o	n	o	o	n
q	r	p	p	p
r	s	s	q	q
s	t	t	t	r

RAID+ template
(w. reserved recovery area)

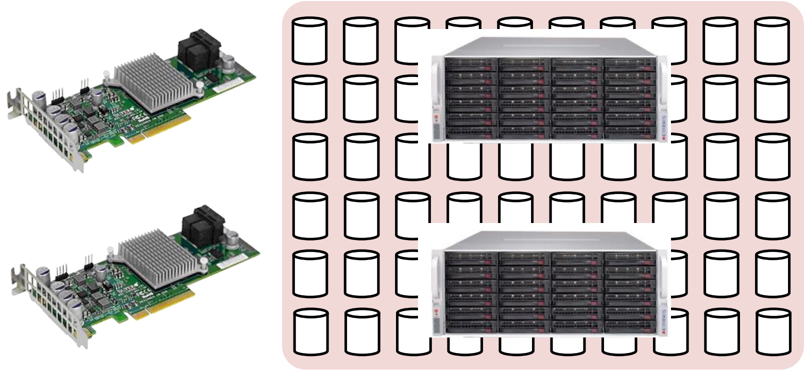
- Only records "template base address"



  - Small overhead with typical template sizes
- In-template addressing easily computable
  - Implemented in <20 lines of code

Evaluation

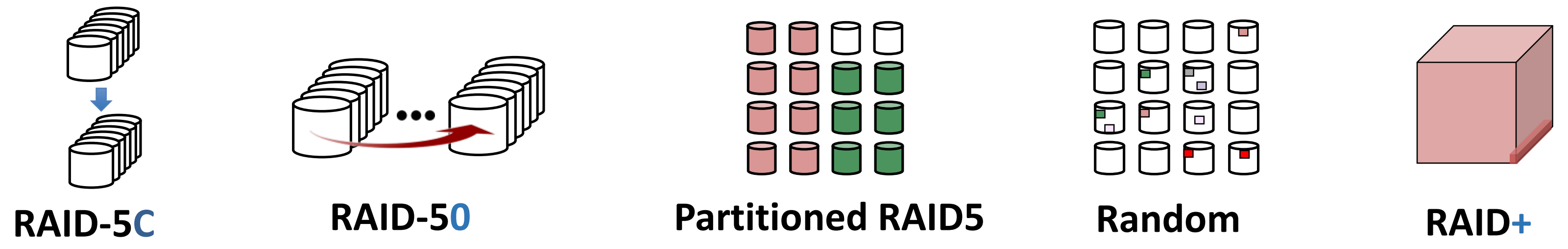
- Implementation
  - MD (Multiple Devices) driver for software RAID, in Linux kernel 3.14.35
- Platform
  - SuperMicro 4U storage server, 2 12-core Intel E5-2650 processors, 128GB DDR4
  - 2 AOC-S3008L- L8I SAS JBOD adapters, each connected to 30-bay SAS3 expander backplane via 2 channels
  - 60 Seagate Constellation 7200RPM 2TB HDDs
  - Aggregate I/O channel bandwidth: 24GB/s



- Workloads
  - Synthetic: fio benchmark
  - 8 public block-level I/O traces, from MSR Cambridge and SPC
  - I/O-intensive applications: GridGraph, TPC-C, Facebook-like, MongoDB

Evaluation Methodology

- Systems
  - Common base RAID setting: (6+1) RAID-5
  - $RAID_H$: CRUSH-style, hashing based random mapping
  - $RAID_R$: Pseudo-random number generation based random mapping

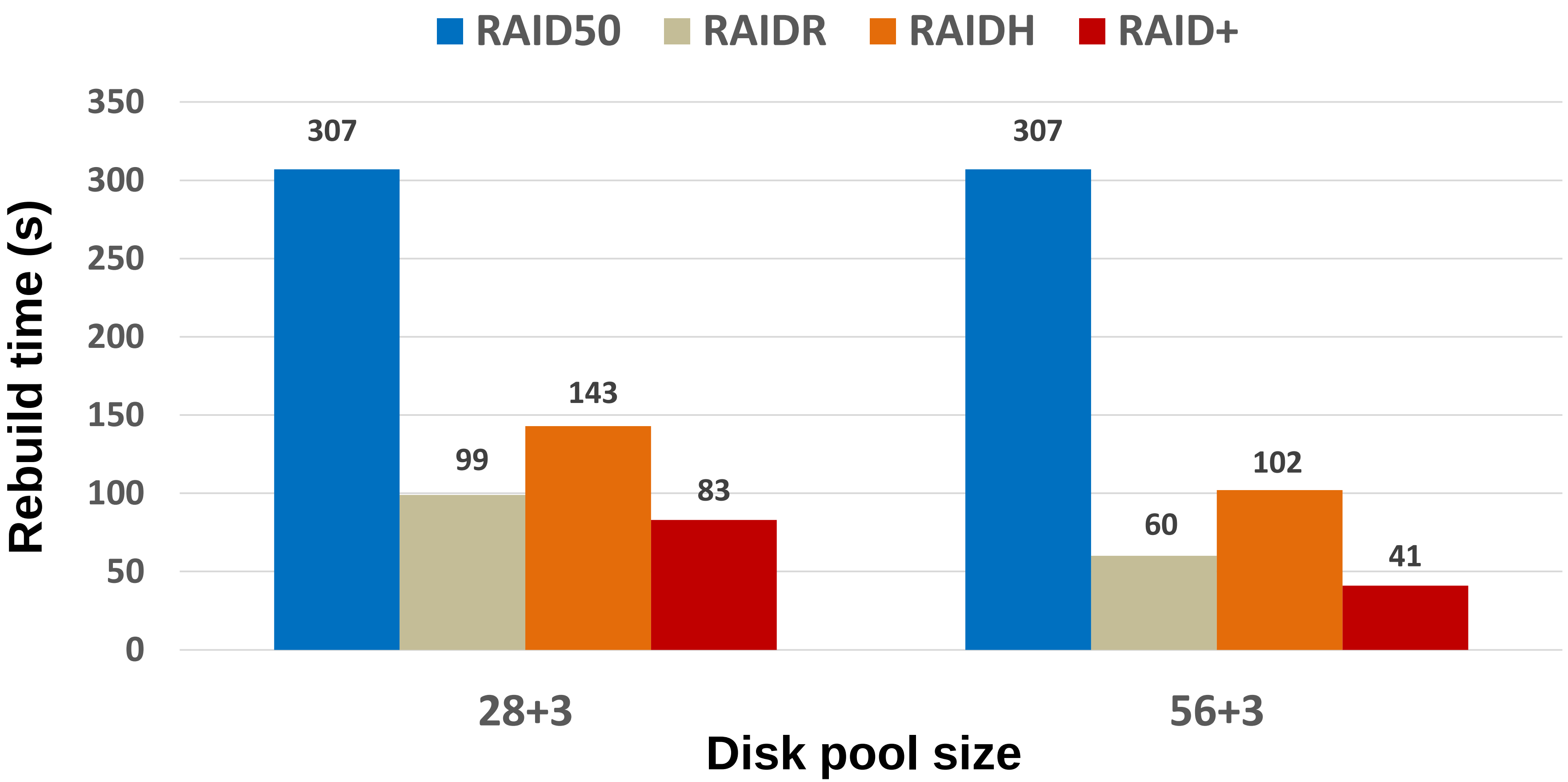


- Metrics
  - Recovery speed (online and offline), interference to application performance
  - Data re-distribution after failures
  - Normal performance: synthetic and application, single- and multi-workload

- More results in paper

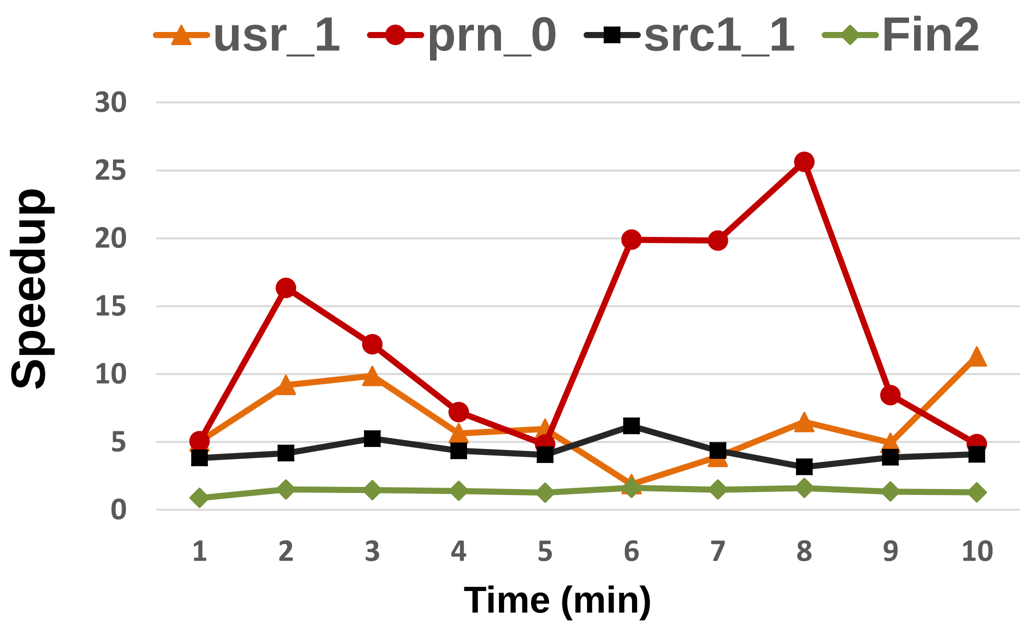
Reconstruction Performance: Offline Rebuild

- Two disk pool sizes: (56+3), (28+3)
- Data size per disk: 50GB

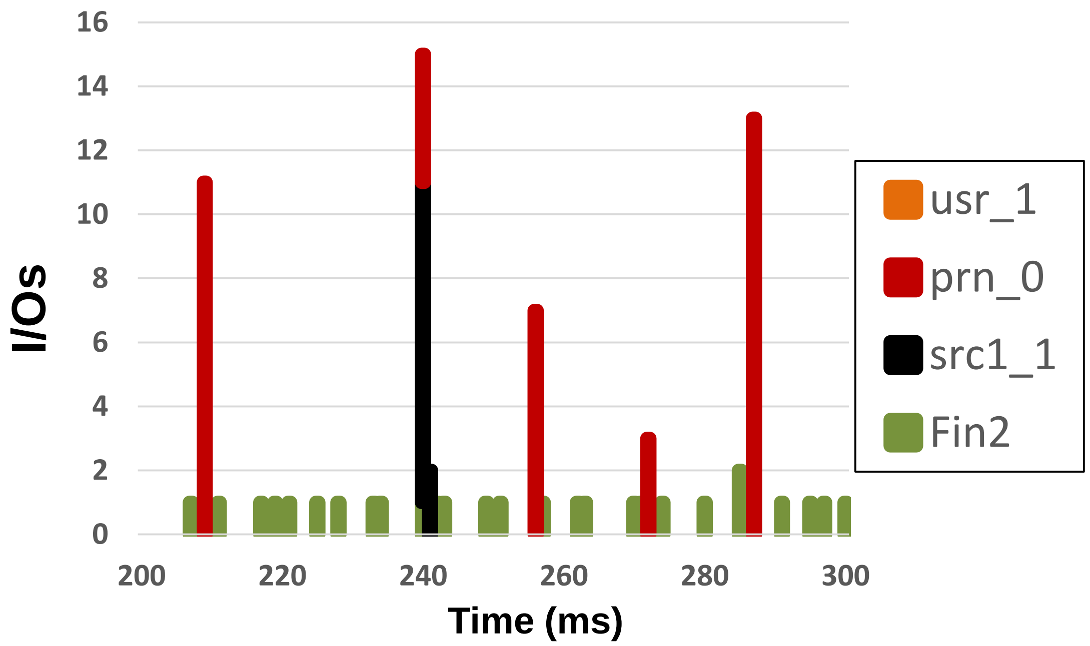


Normal I/O Performance : Multi-workload

- Running 4-app mix selected from 8 storage traces
- Evaluating all 70 mixes
  - RAID+ brings avg. 2.05 weighted speedup over partitioned RAID-5, vs. 1.83 by $RAID_H$
  - Only 39 out of 280 runs experienced slowdown



RAID+ speedup over partitioned RAID-5



Sample load variation

Conclusions

- RAID+: new RAID architecture
  - Flexible and efficient
  - Virtual RAID groups on large disk enclosures

- Enabled by Latin-square based 3D templates
  - Guaranteed uniform data distribution for both normal and 1-disk failure recovery I/O
  - Deterministic addressing
  - Utilizing all disks evenly while maintaining data locality

- Especially appealing to shared large disk enclosures
  - Enterprise server, cloud, and datacenter

THANKS!

Q&A