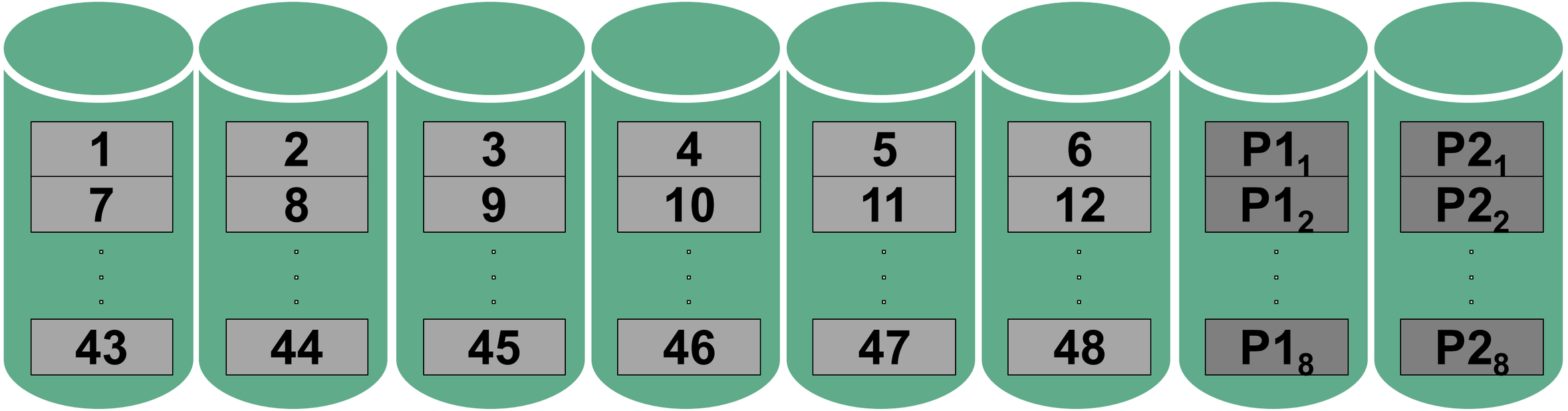
Okapi
Decoupling Data Stripes and Redundancy Groups

Sanjith Athlur

Timothy Kim, Saurabh Kadekodi*,
Francisco Maturana, Xavier Ramos,
Arif Merchant*, Rashmi Vinayak, Greg Ganger

Carnegie Mellon University, *Google

Stripes ≠ Groups



Data Striping
Based on workload access pattern

Erasure Coding: *k*-of-*n* scheme
Based on reliability / cost

Okapi: Decouple Data Stripes and EC Groups

Okapi decouples stripes and groups

- Okapi adds flexibility to tune stripes and groups
  - Extracts best of both worlds
  - Improves read performance and optimizes space savings
  - Without sacrificing reliability

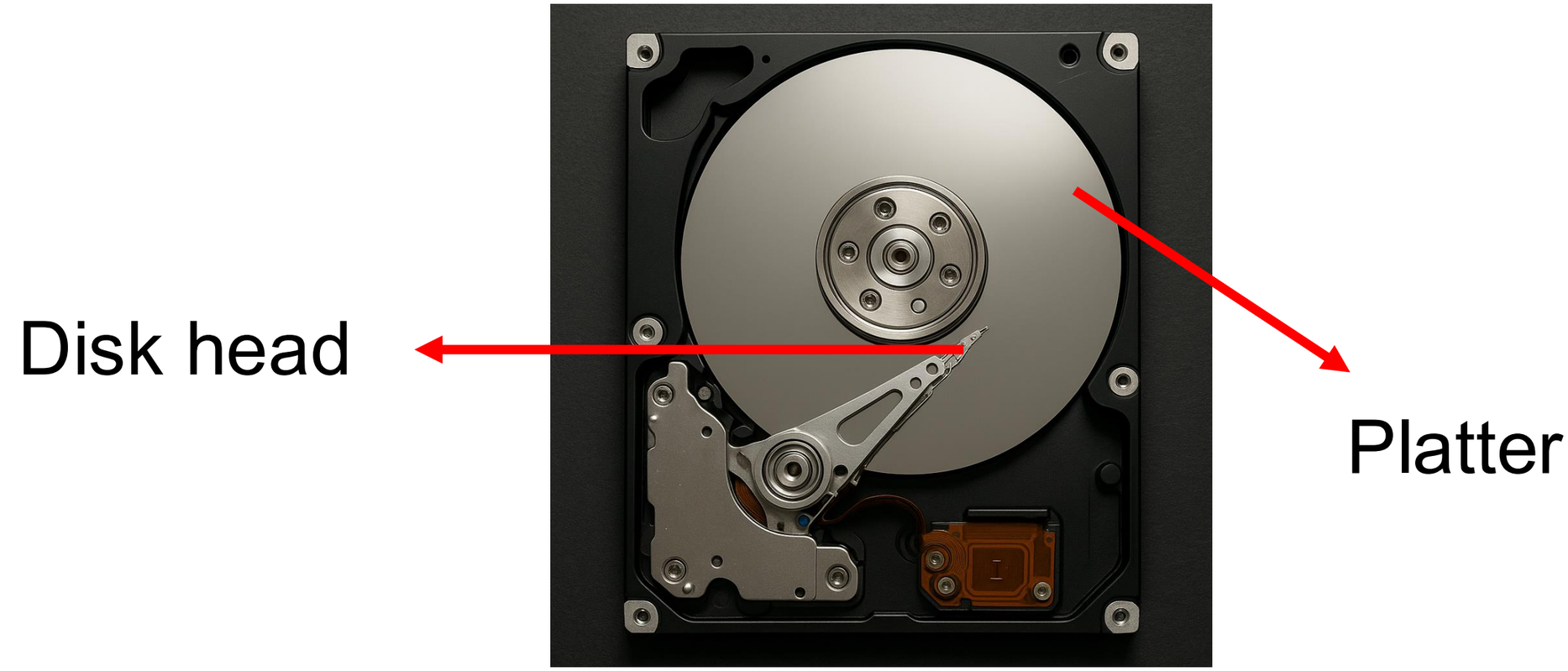
- Okapi improves transcoding IO overheads

Outline

- How coupled systems stripe/group data?
- How Okapi stripes and groups data?
- Okapi improves read performance
- Okapi reduces transcoding IO overheads
- Okapi overcomes challenges with simple design ideas

HDDs: Amortize repositioning costs

Read/Write = Repositioning (seek + rotation) + Transfer



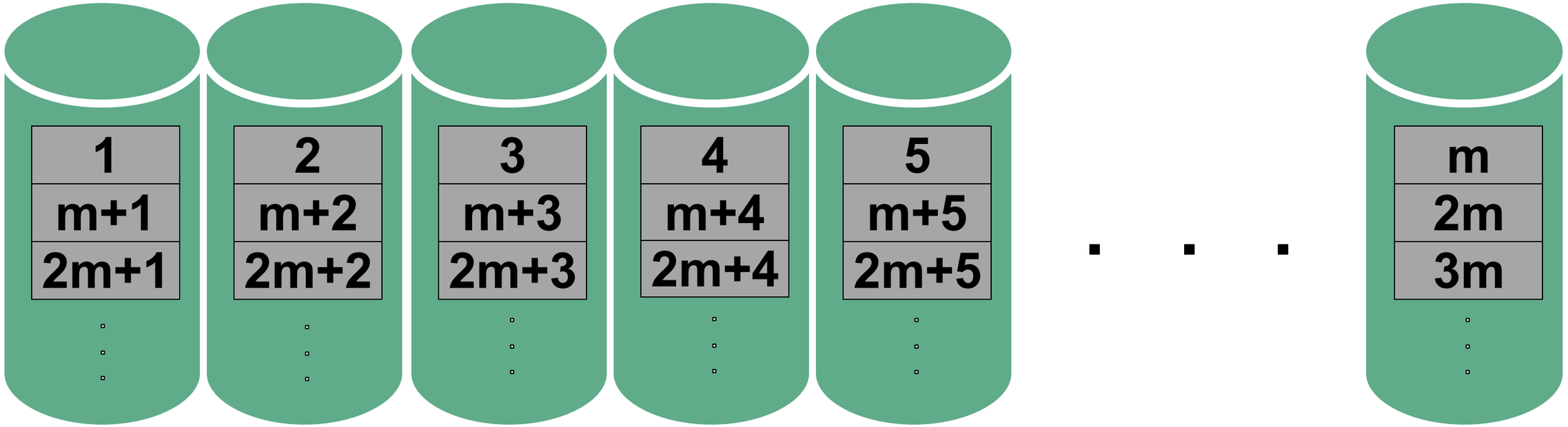
HDDs: Amortize repositioning costs

Read/Write = Repositioning (seek + rotation) + Transfer

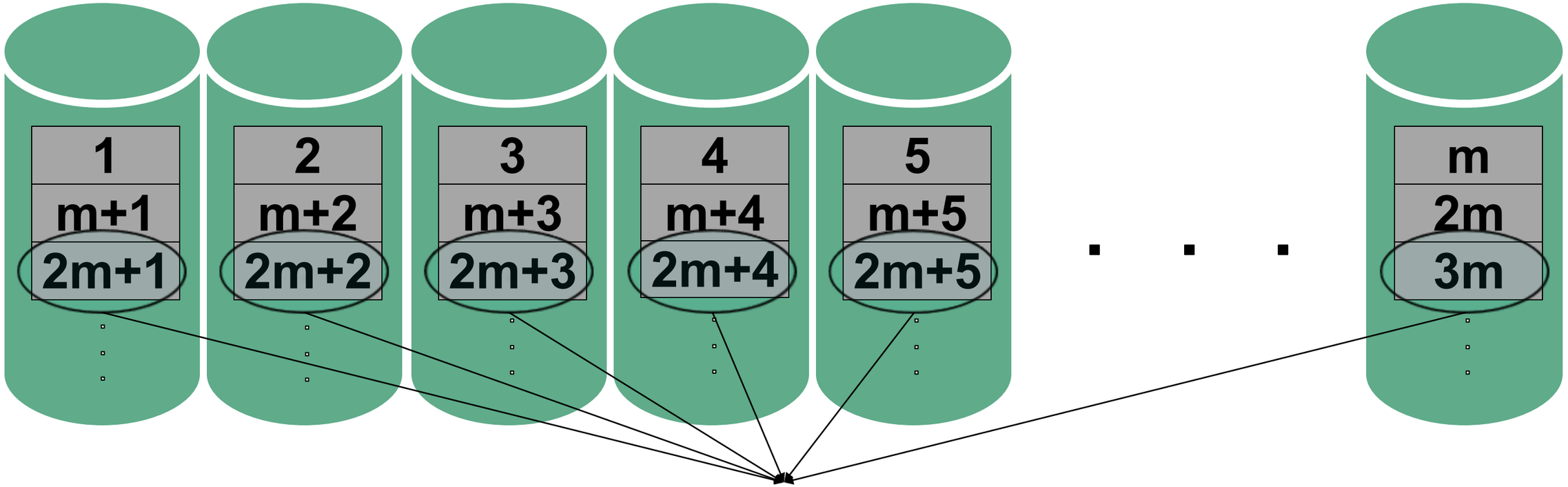


Few large reads is better than many small reads

Stripe for IO performance



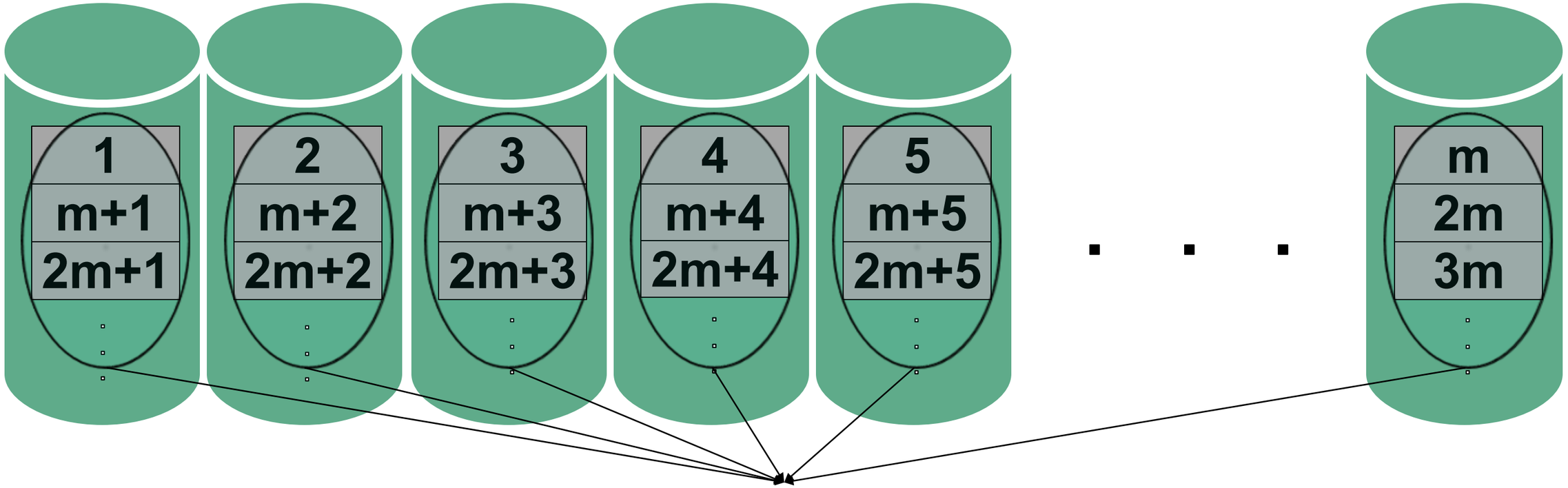
Stripe for IO performance



Cells

Default cell size = 1 MB

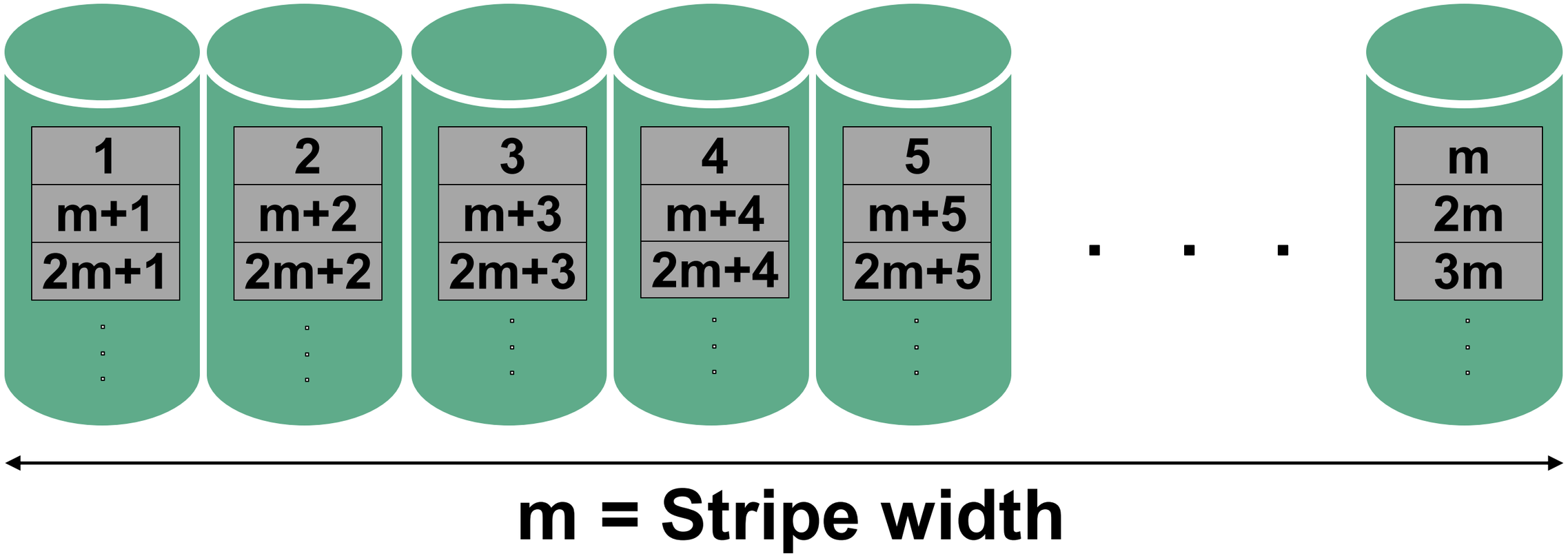
Stripe for IO performance



Blocks

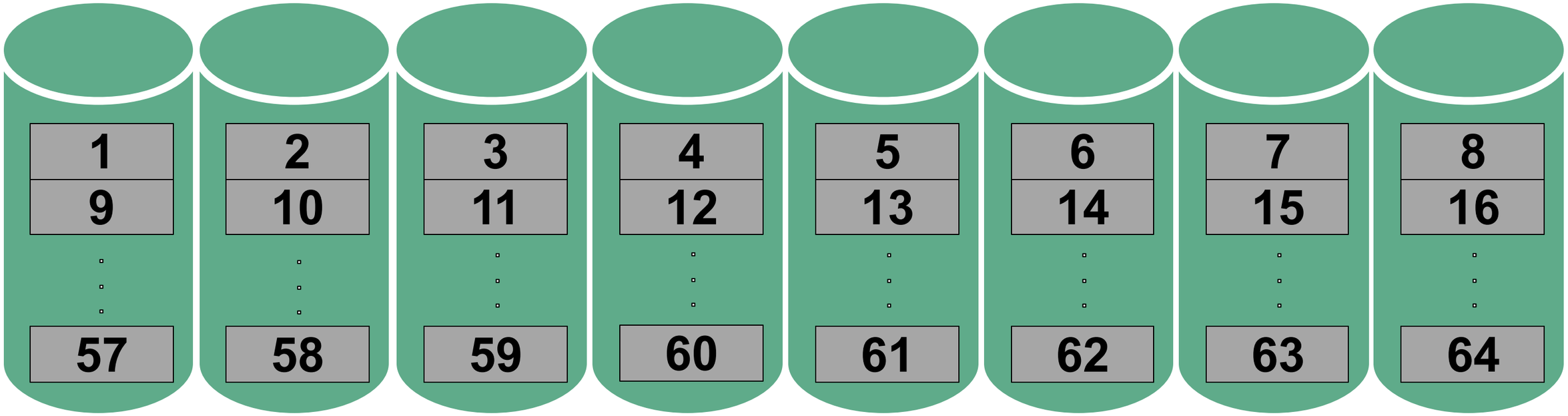
Default block size = 8 MB

Stripe for IO performance



Large streaming read? Stripe wide

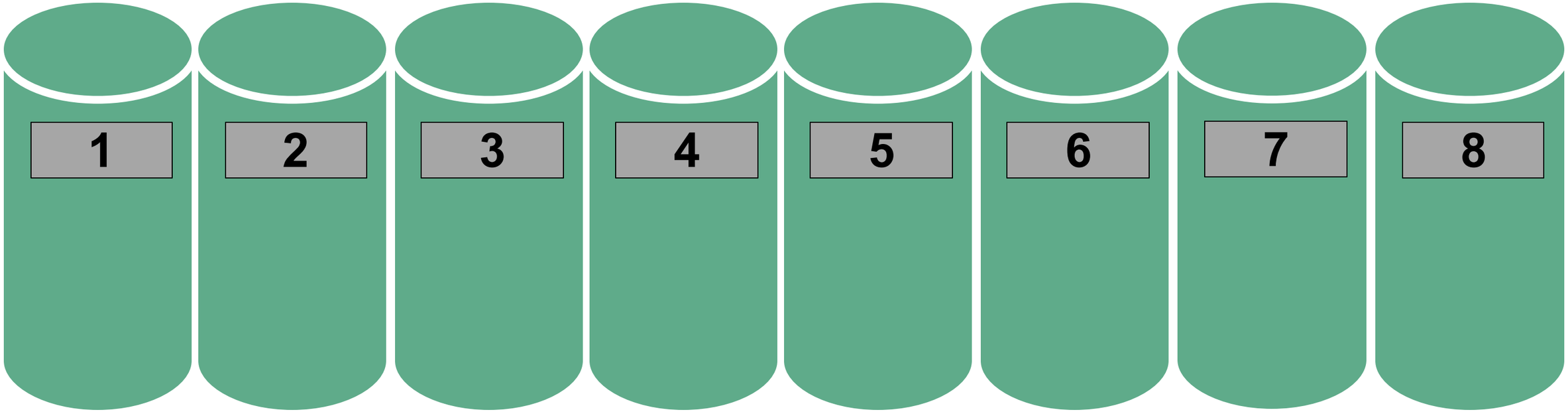
Read size = 64 MB. Stripe width = 8



High parallelism. High throughput.

Small read? Wide stripes can hurt

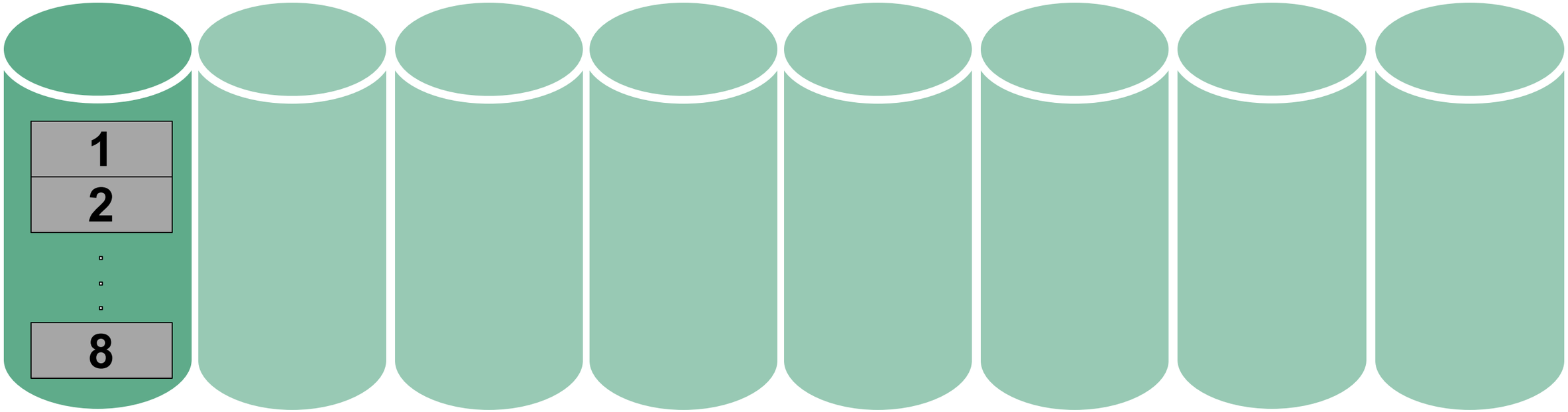
Read size = 8 MB. Stripe width = 8



Total disk time spent / byte read is high

Small read? Stripe narrow

Read size = 8 MB. Stripe width = 1



Low stripe width → high disk efficiency

Best striping depends on the workload

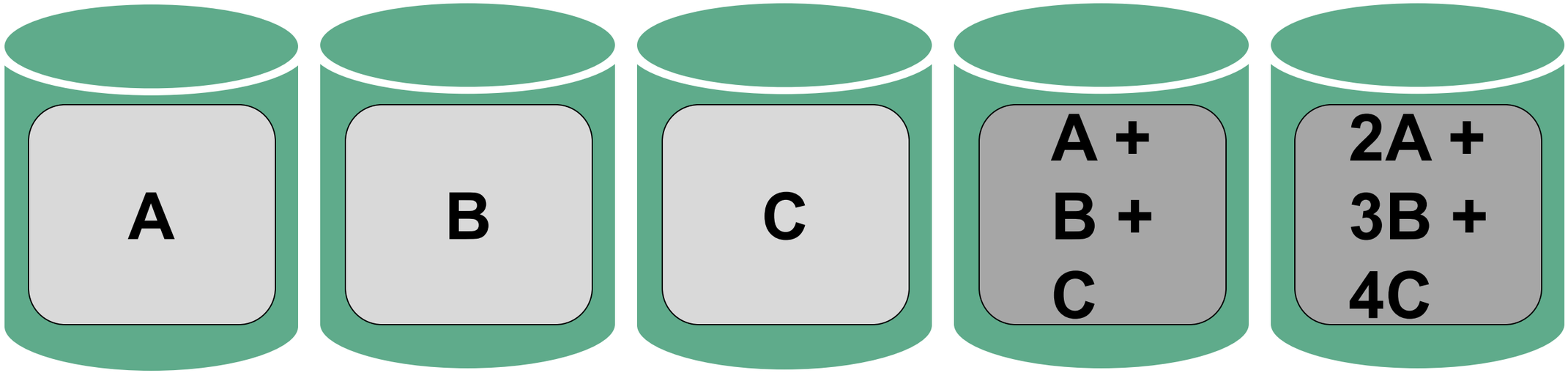
- Striping width depends on size of the read request

- Small reads → narrower stripes
- Large reads → wider stripes

- Developers analyze access patterns to tune files

Group for reliability

Any k of n blocks required to recover missing block(s)



Data blocks
#blocks = k

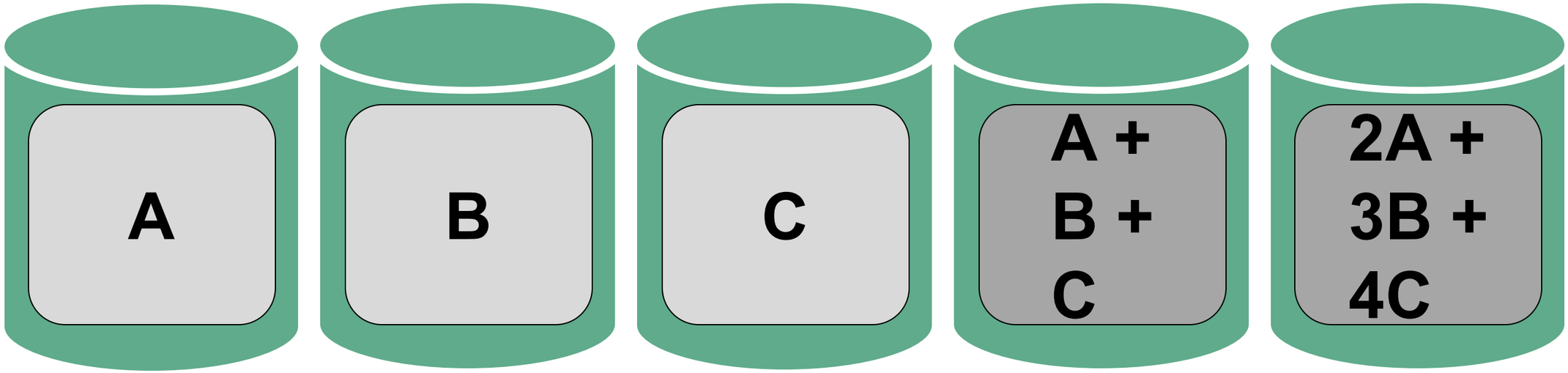
Parity blocks
#blocks = n-k

k-of-n erasure coded group

k → Group width

Group for reliability

Any k of n blocks required to recover missing block(s)



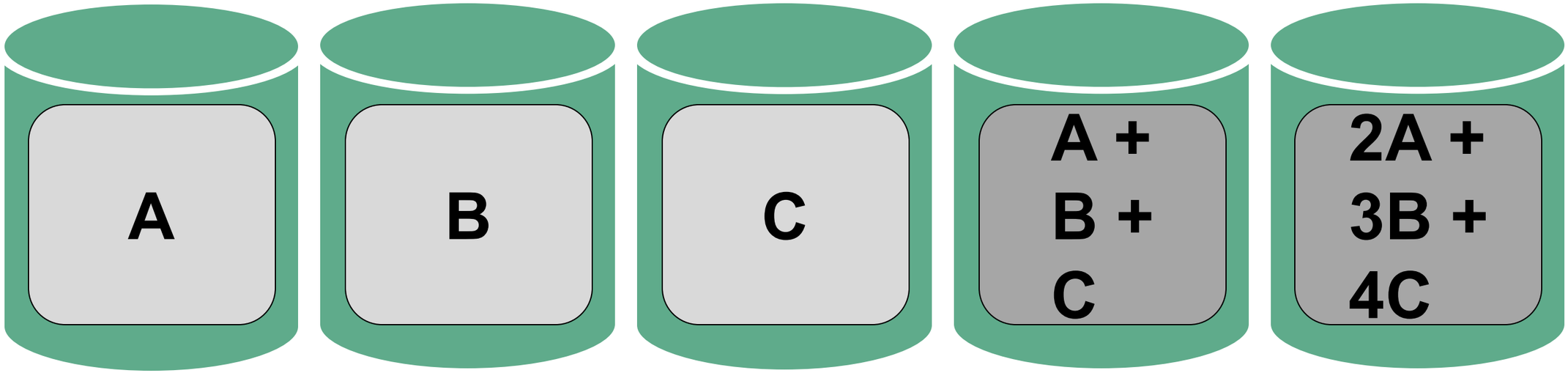
Data blocks
#blocks = k

Parity blocks
#blocks = n-k

Depends on: **Reliability requirements (Target MTTDL)**

Group for reliability

Any k of n blocks required to recover missing block(s)



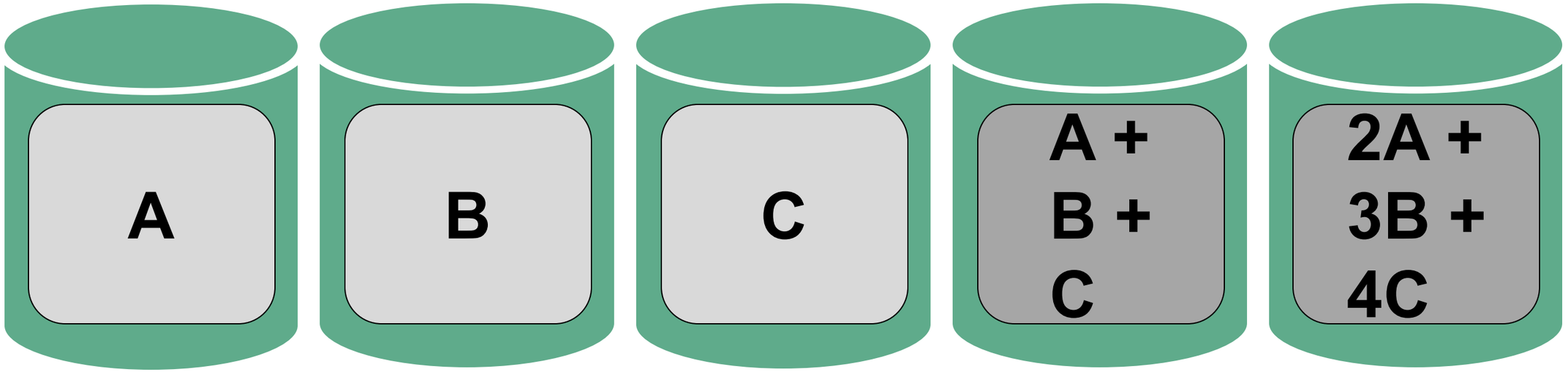
Data blocks
#blocks = k

Parity blocks
#blocks = n-k

Depends on: **Storage overhead (cost) =** $\frac{n-k}{k}$

Group for reliability

Any k of n blocks required to recover missing block(s)



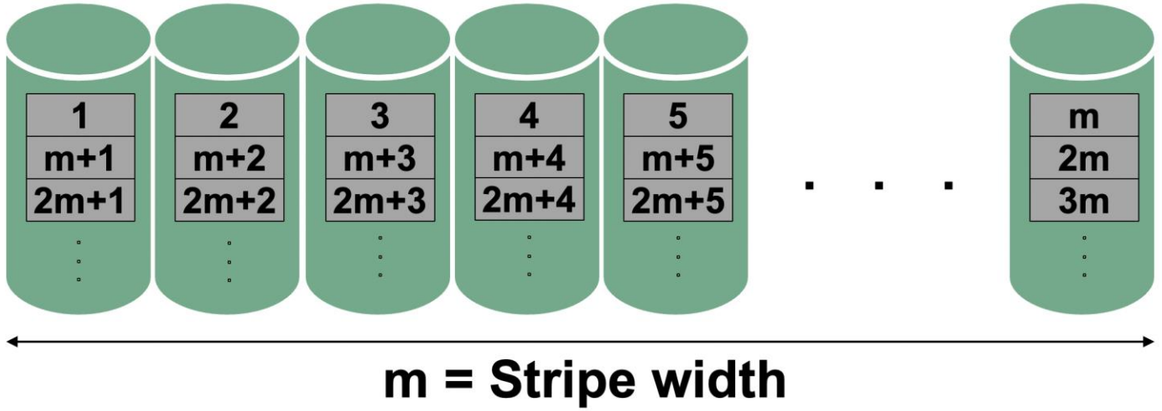
Data blocks
#blocks = k

Parity blocks
#blocks = n-k

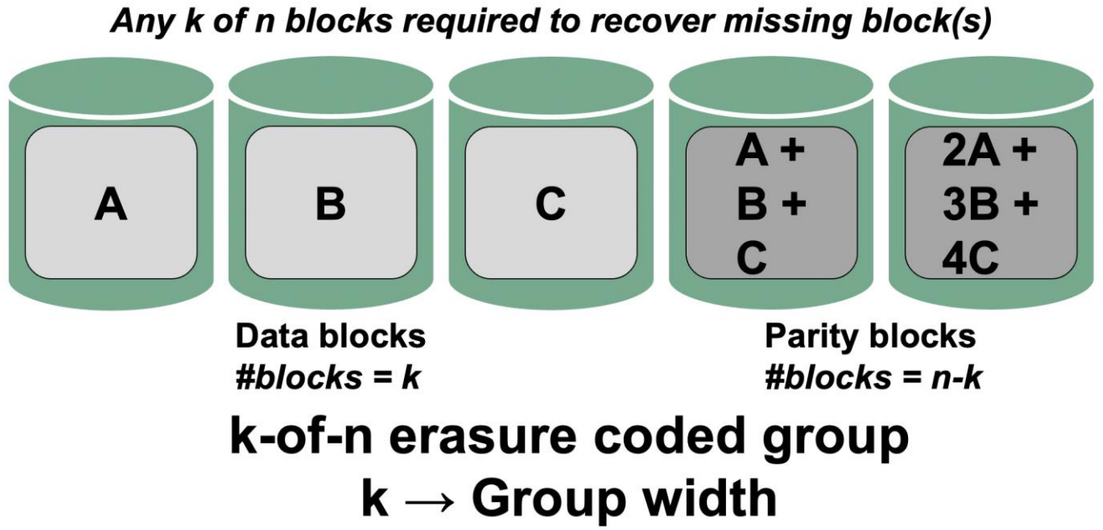
Depends on: Reconstruction traffic quota α *k*

Stripes and groups are independent

Stripe for IO performance



Group for reliability



Depends on

- Workload access patterns

Depends on

- Target MTTDL
- Storage overhead
- Reconstruction costs

Stripes and groups are strictly coupled

File size = 36 MB. Block size = 3 MB.
Stripe width = 4 ⇒ Group width = 4 (*4-of-6* scheme)

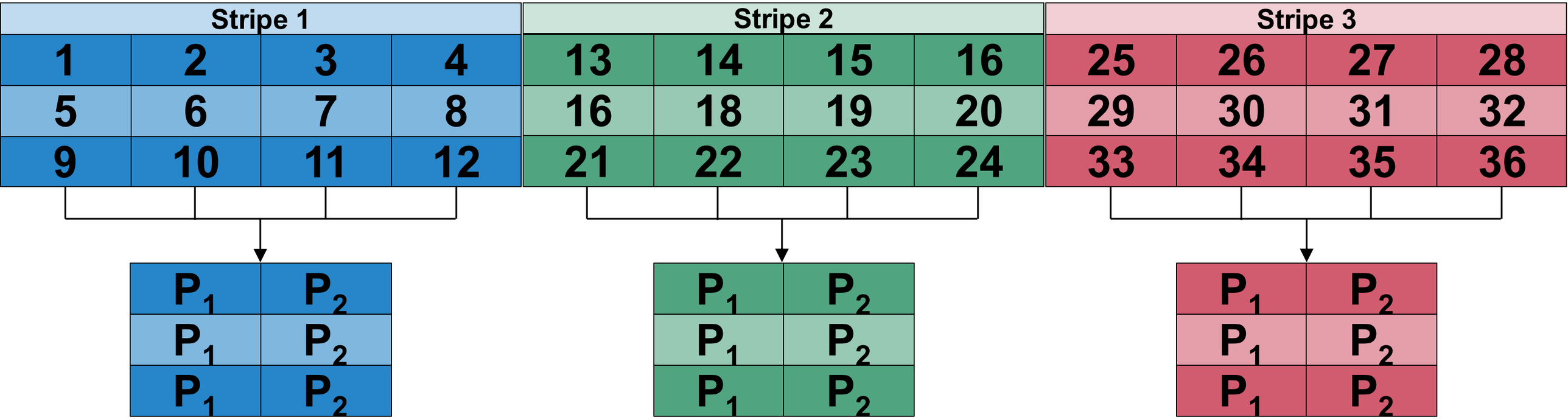
Stripe 1			
1	2	3	4
5	6	7	8
9	10	11	12

Each number is a cell (1 MB)
Each column is a block (3 MB)
Each block is stored in a distinct disk
A block is stored sequentially on a disk

Stripes and groups are strictly coupled

File size = 36 MB. Block size = 3 MB.

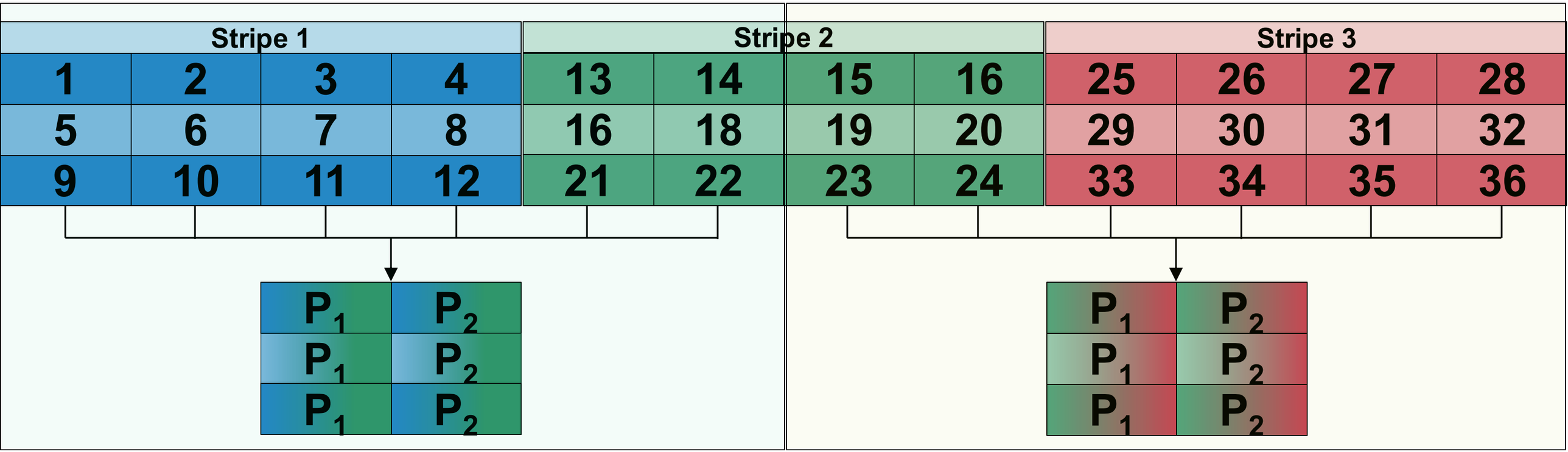
Stripe width = 4 ⇒ Group width = 4 (*4-of-n* scheme)



Okapi allows tuning stripes *and* groups

File size = 36 MB. Block size = 3 MB.

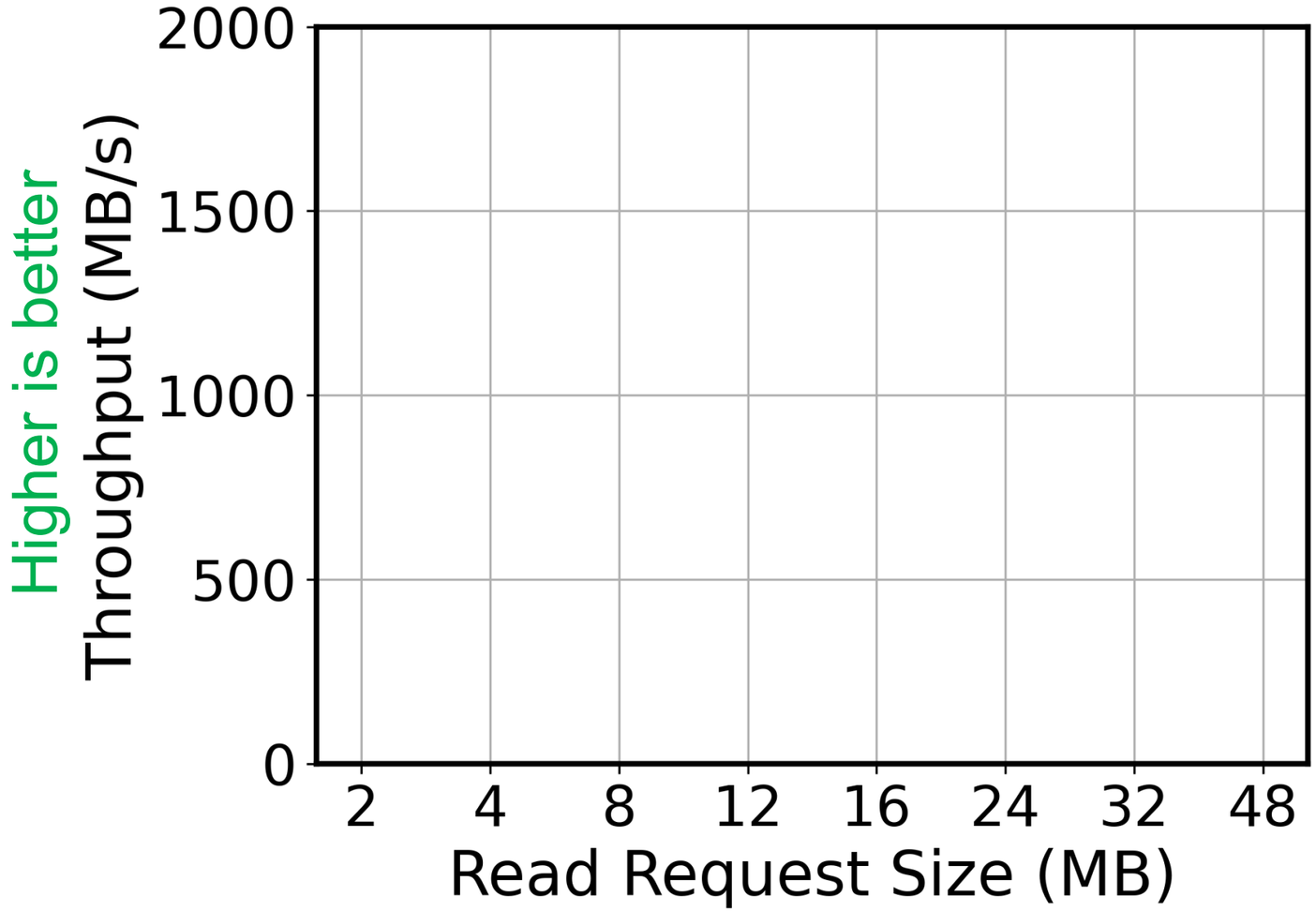
Stripe width = 4. Group width = 6 (*6-of-8* scheme)



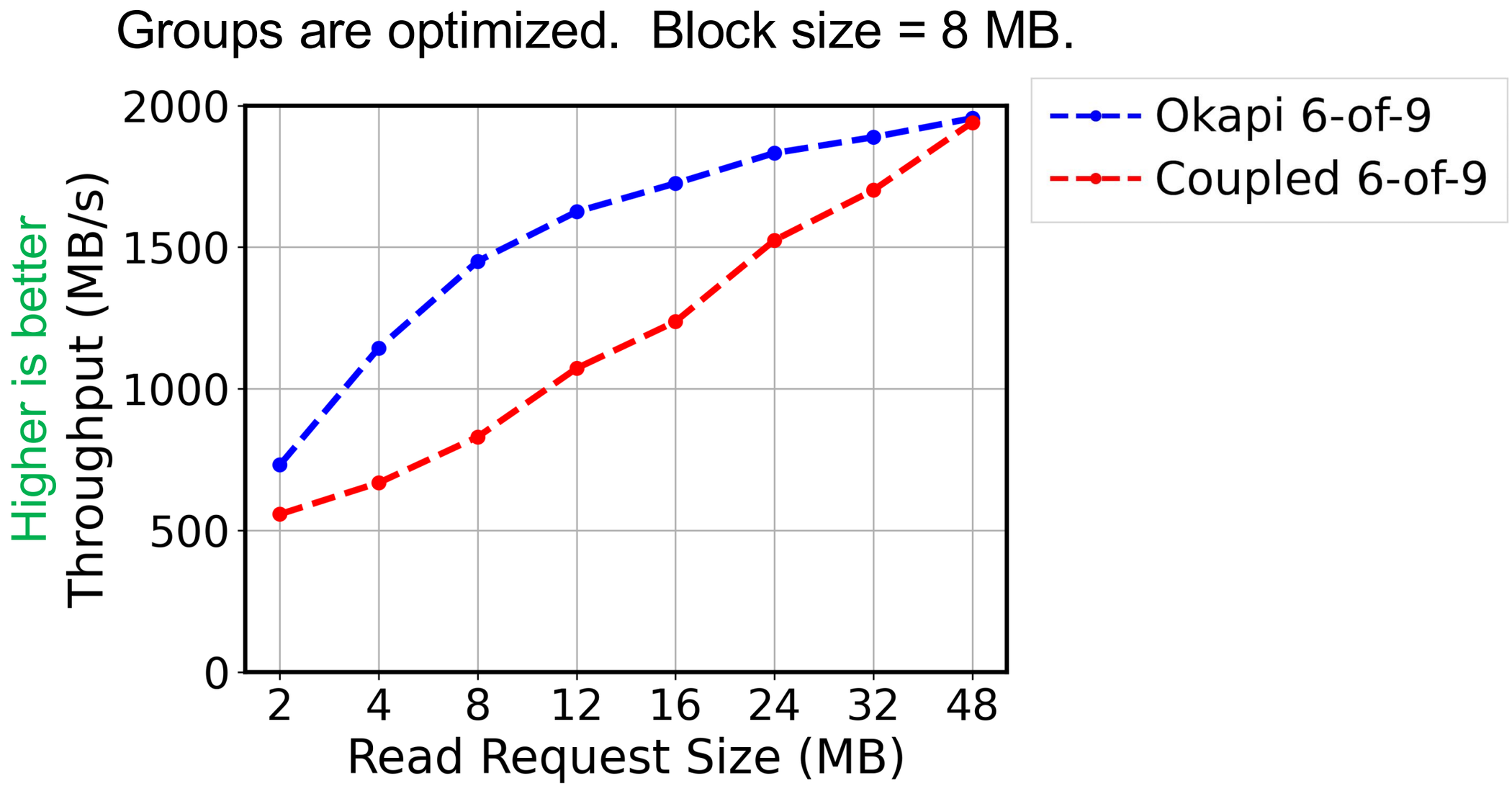
Group formed with 6 *consecutive* blocks

Okapi improves tput by up to 80%

Clients make read requests of size *x* to a 20-node DFS cluster



Okapi improves tput by up to 80%



Files have a consistent access pattern

- All reads of same size for 64-94% of files
  (Measured across Google clusters, for 150 days)

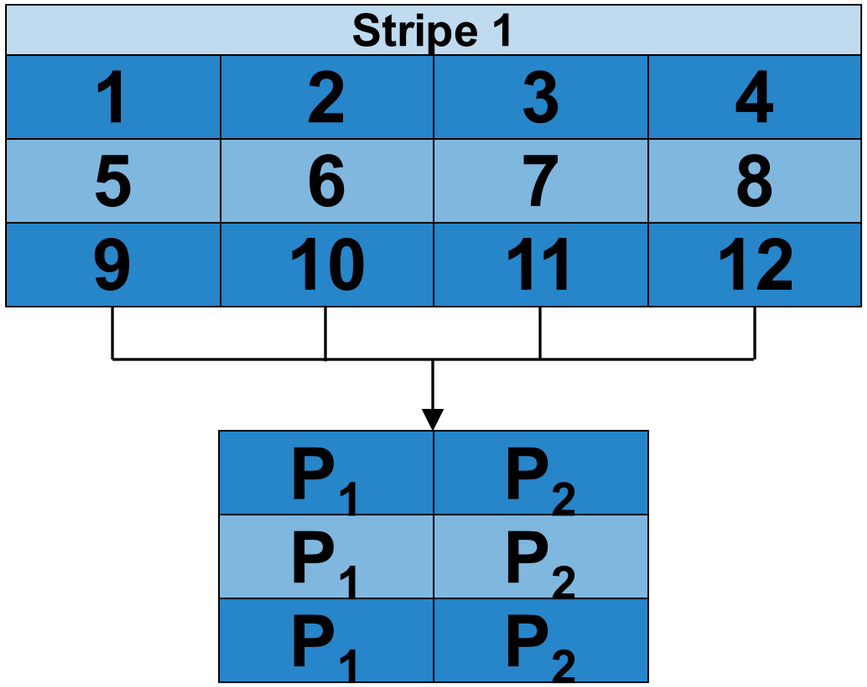
- Per-file read request sizes show low variance

Okapi for transcoding

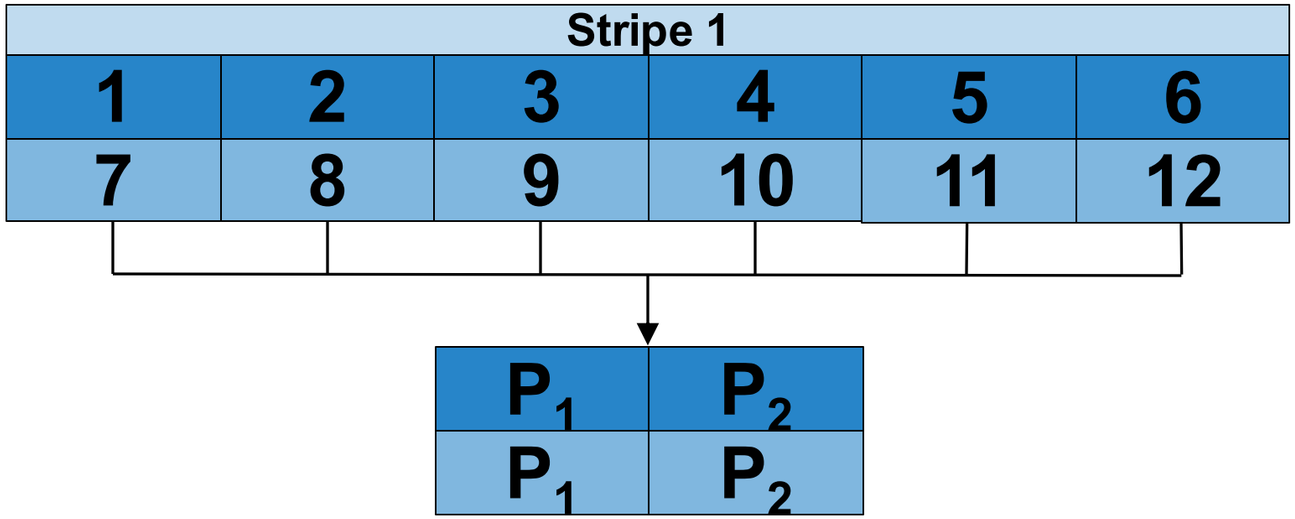
- Transcoding : changing EC scheme of a file
  - from k1-of-n1 to k2-of-n2
- Common and useful
  - Petabytes of IO per day in large clusters
- But SOTA: Read – Re-encode – Write

Transcoding is ugly

From *4-of-6* scheme

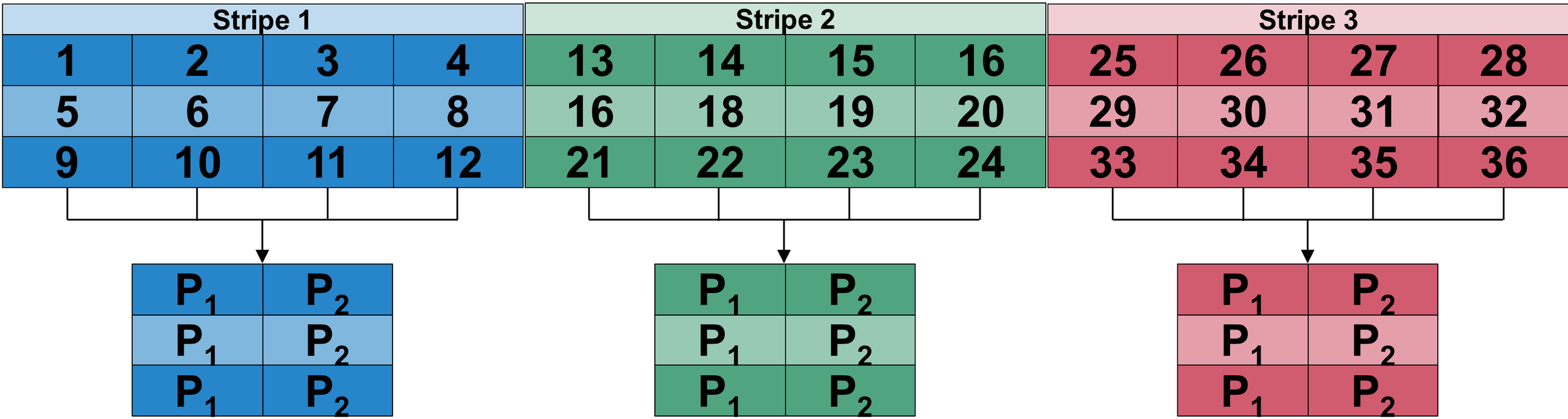


to *6-of-8* scheme



Okapi's transcoding is good

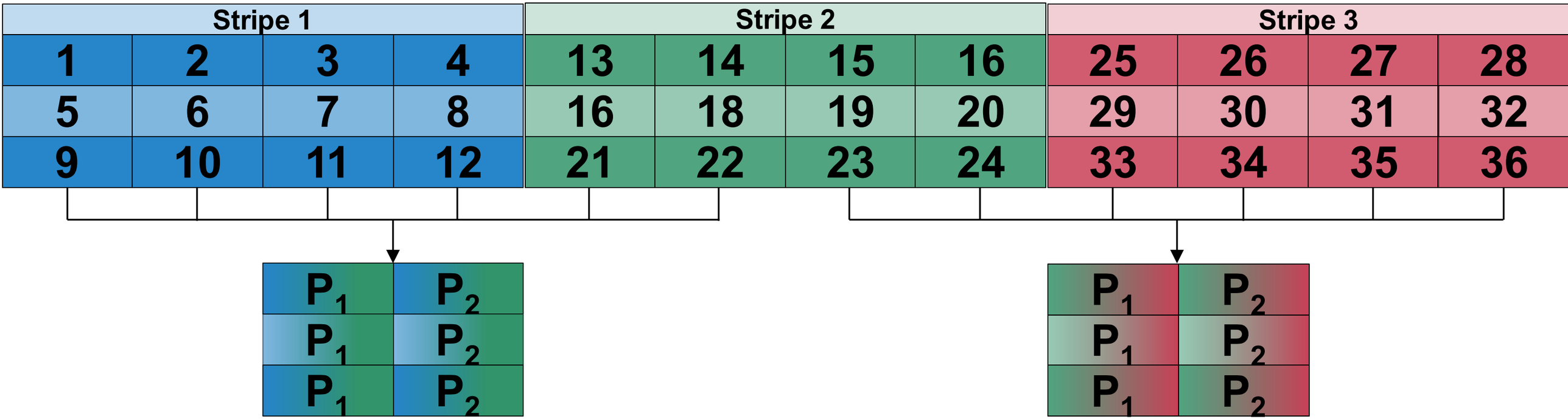
From *4-of-6* scheme



Don't re-stripe. Only re-group

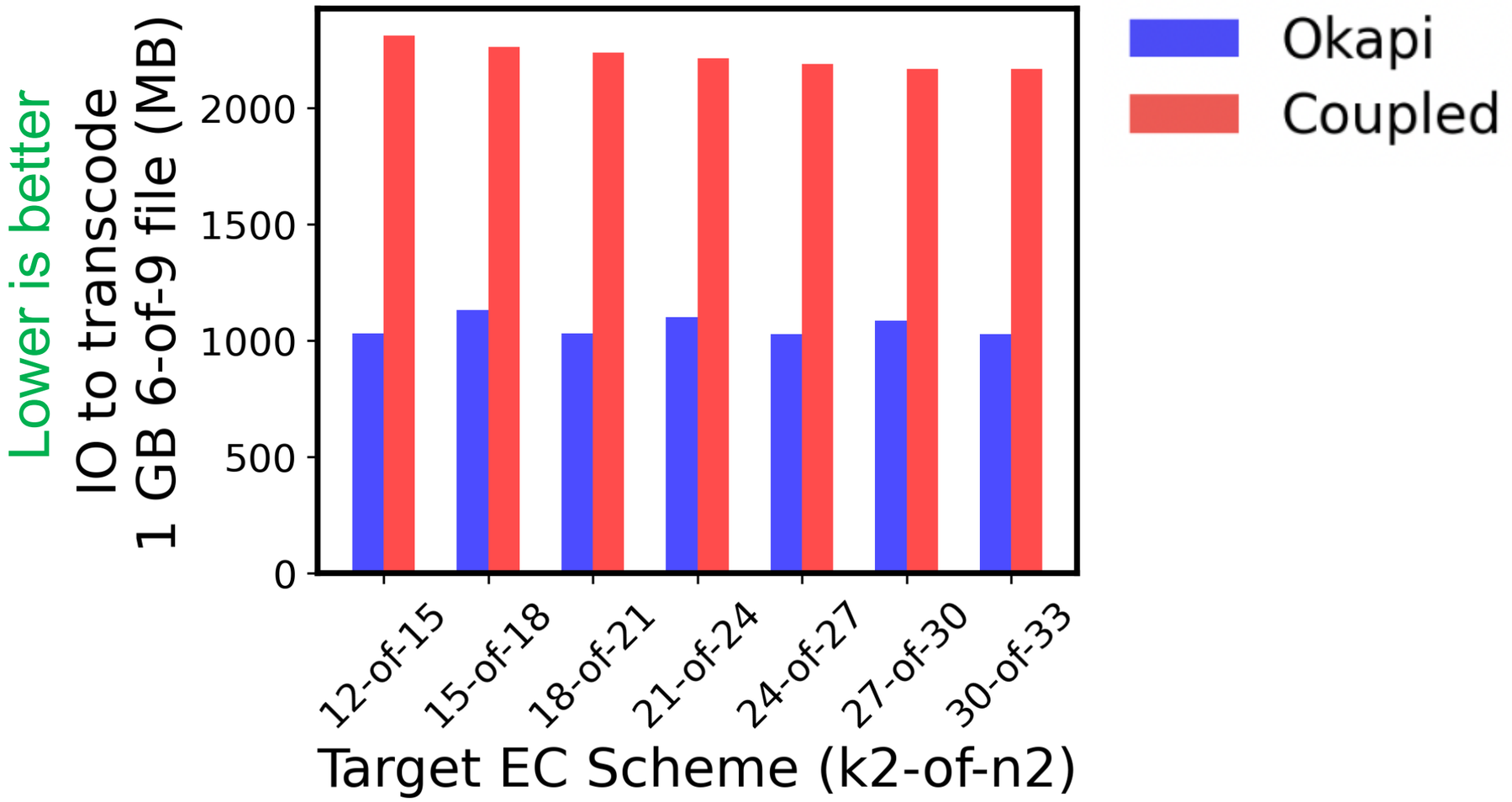
Okapi's transcoding is good

To *6-of-8* scheme



Okapi eliminates data movement

Okapi reduces transcoding IO by 50%



Challenges

- Metadata overhead
  - Might require two data structures for striping and grouping
  - Instead, we infer grouping from striping information
- Write memory overhead
  - Might need to buffer more bytes to compute parities
  - Instead, we use partial parities to buffer just parities
- Read amplification for degraded reads may be high
  - We use caching/prefetching to curb the overhead

Stripes ≠ Groups

- Okapi adds flexibility to tune stripes and groups
  - Extracts best of both worlds
  - Improves read performance and optimizes space savings
  - Without sacrificing reliability

- Okapi improves transcoding IO overheads