RAID-Z Expansion

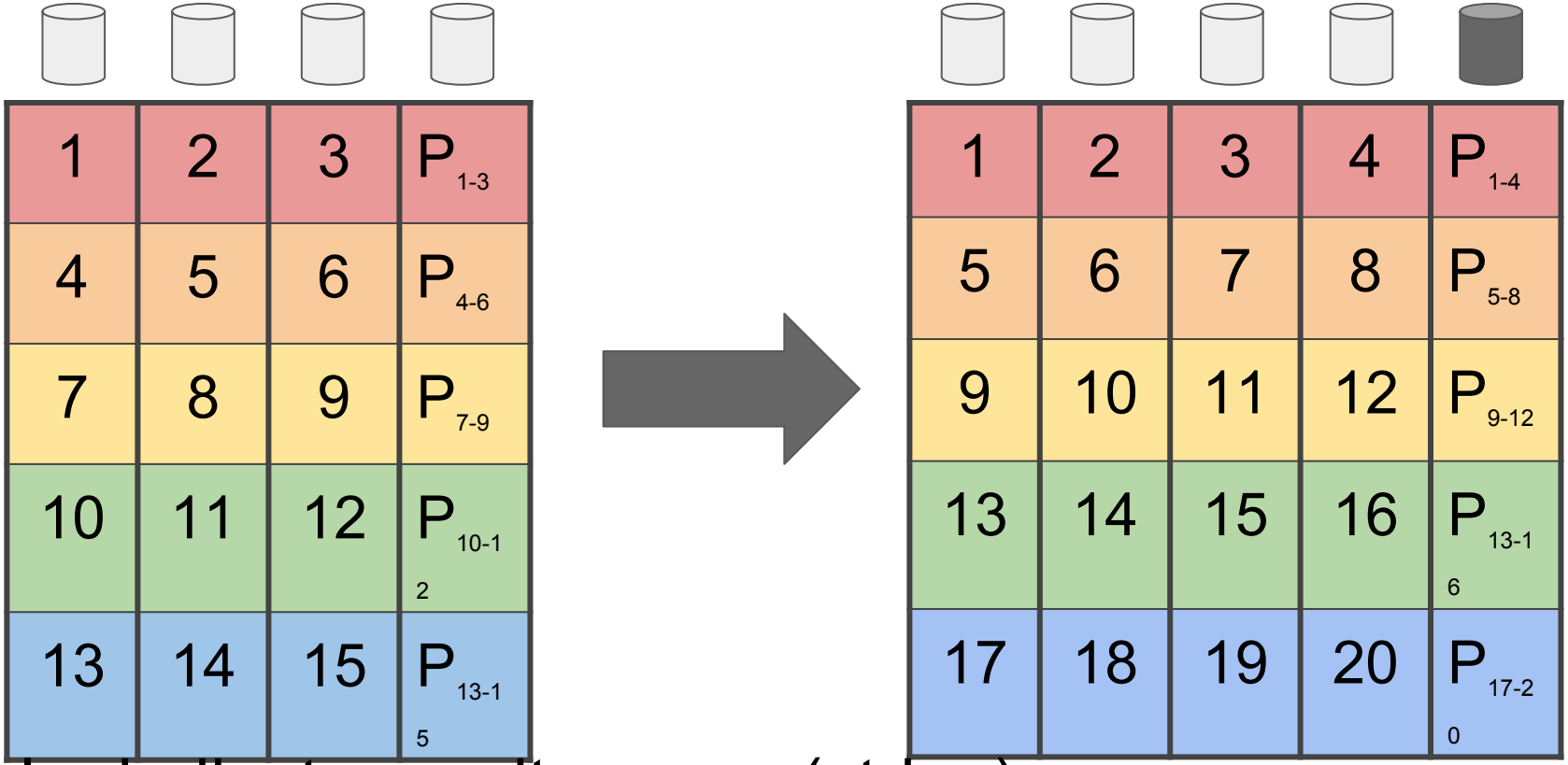
Matt Ahrens

Problem:

You have a RAID-Z pool with 4 disks in it

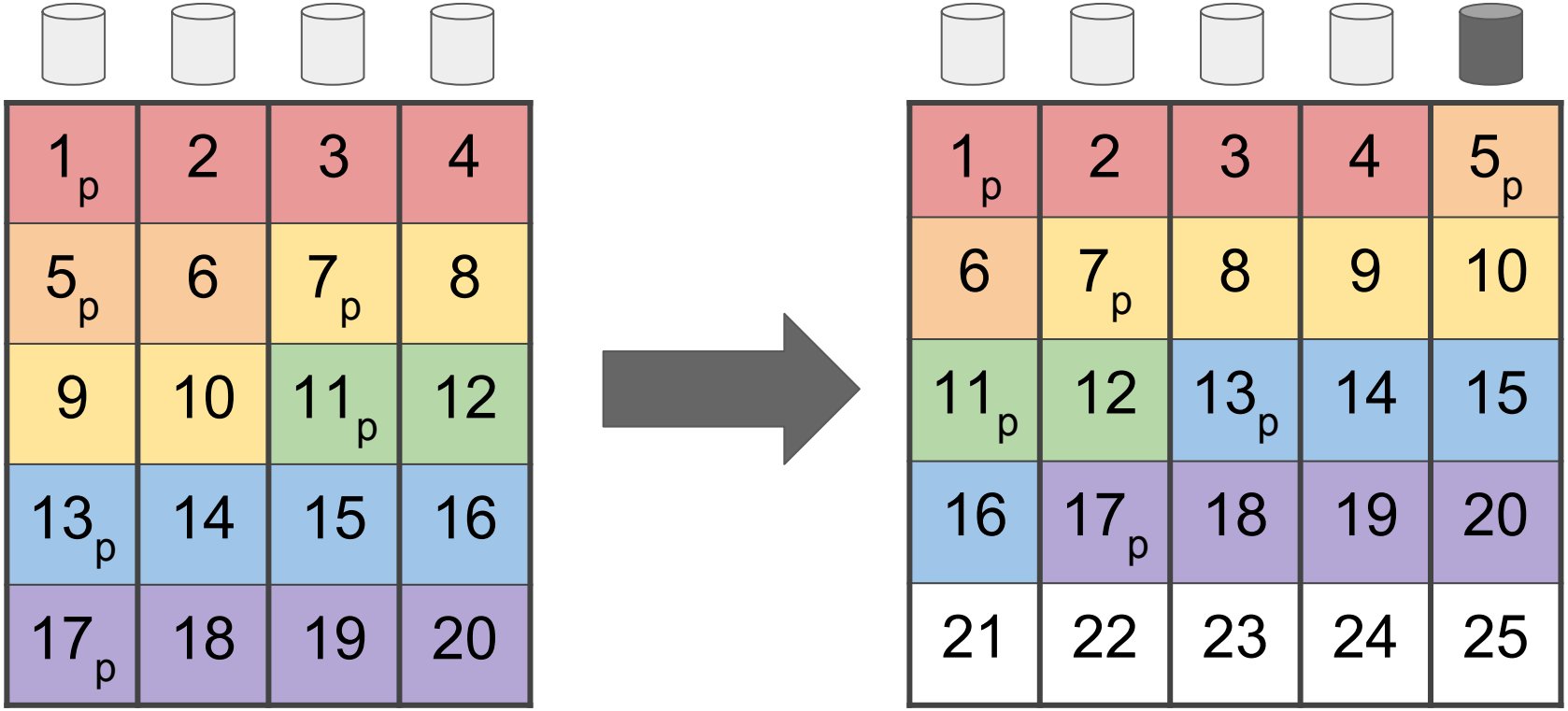
You want to add a 5th disk

How does traditional RAID 4/5/6 do it?



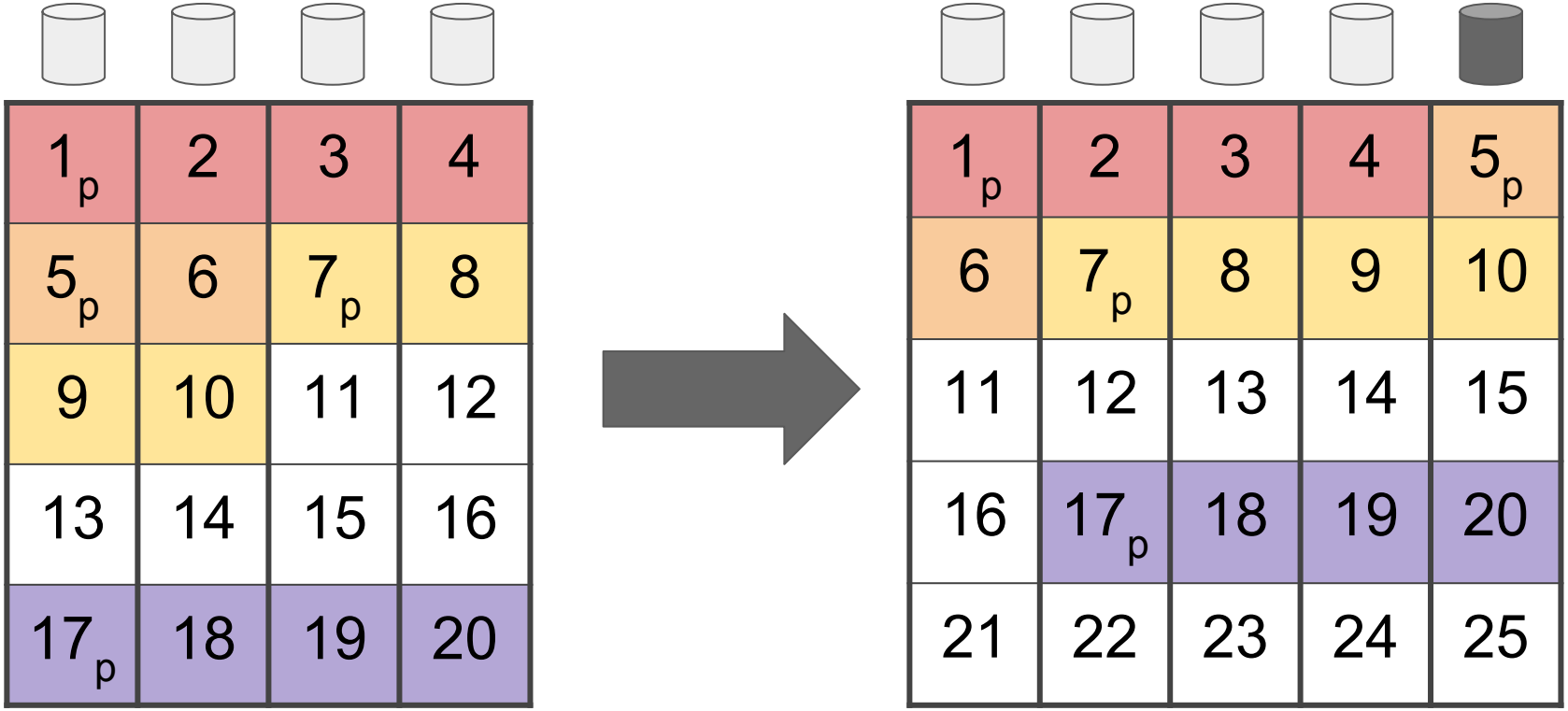
Color indicates parity group (stripe)

RAID-Z Expansion: Reflow



Color indicates parity group (logical stripe)

RAID-Z Expansion: Reflow copies allocated data

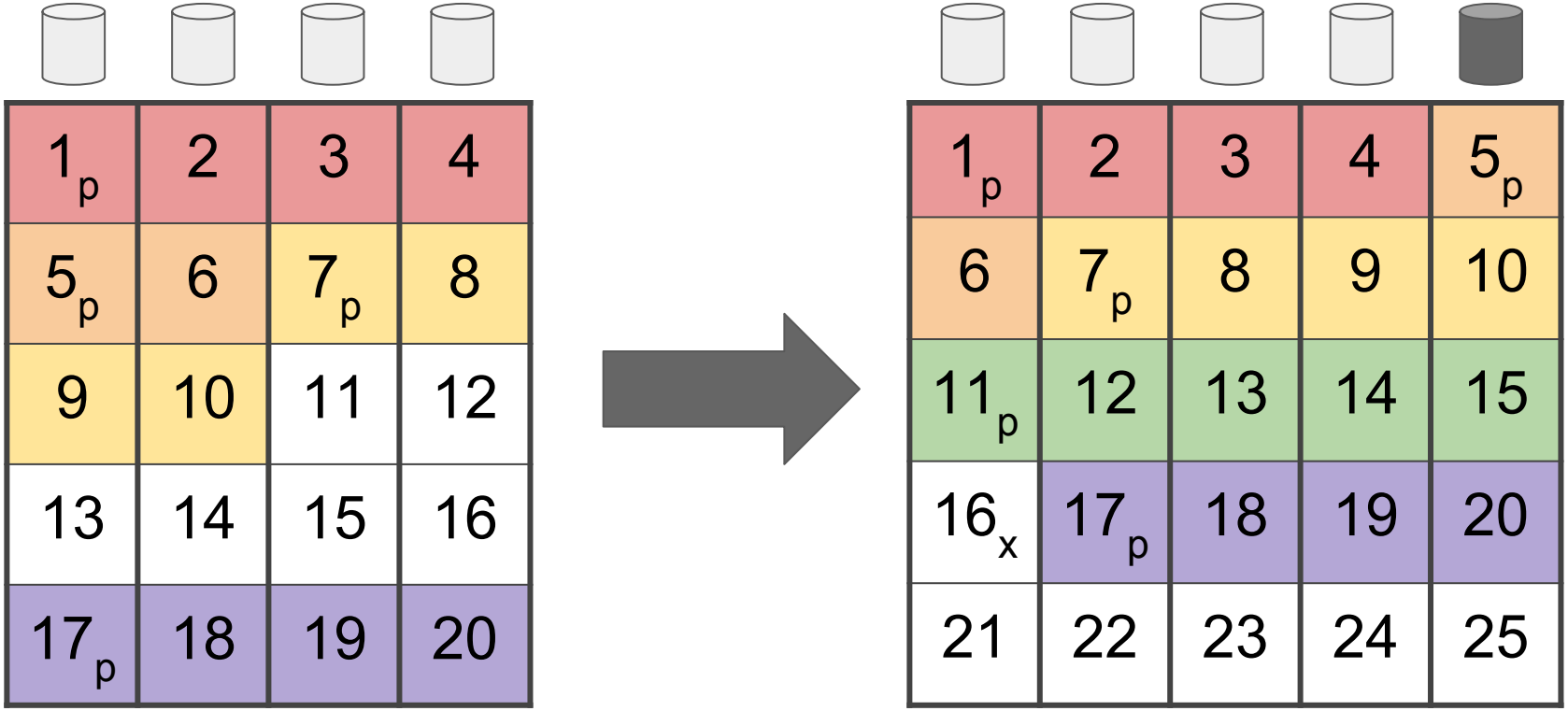


Color indicates parity group (logical stripe)

This works!

- Doesn’t change block pointers
- Reads / Writes sequentially
- Spacemaps tell us what we need to copy
- Each logical stripe is independent
  - Don’t need to know where parity is
  - Segments still on different disks, so redundancy is preserved
  - (contraction couldn’t work this way)

RAID-Z Expansion: new writes, new stripe width

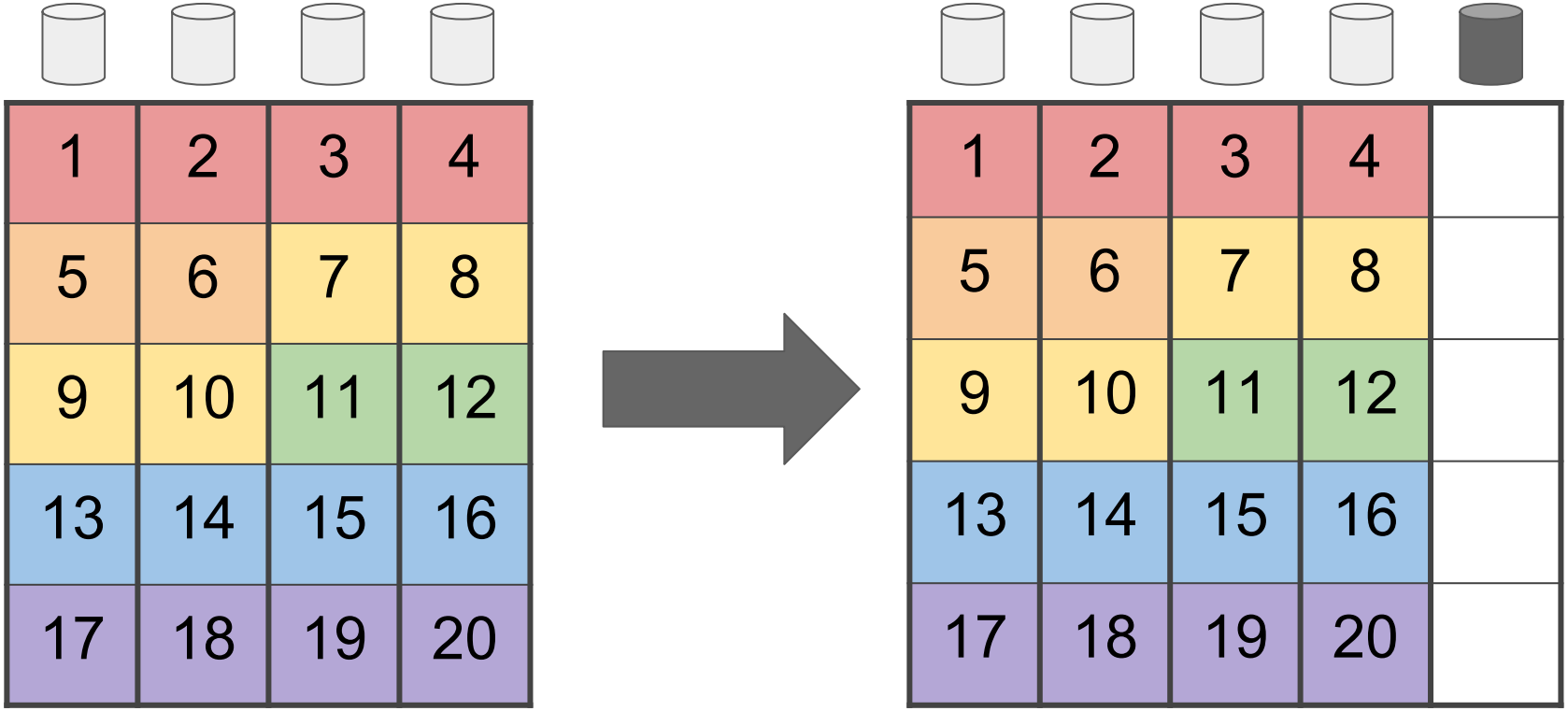


Color indicates parity group (logical stripe)

Logical vs Physical stripe width

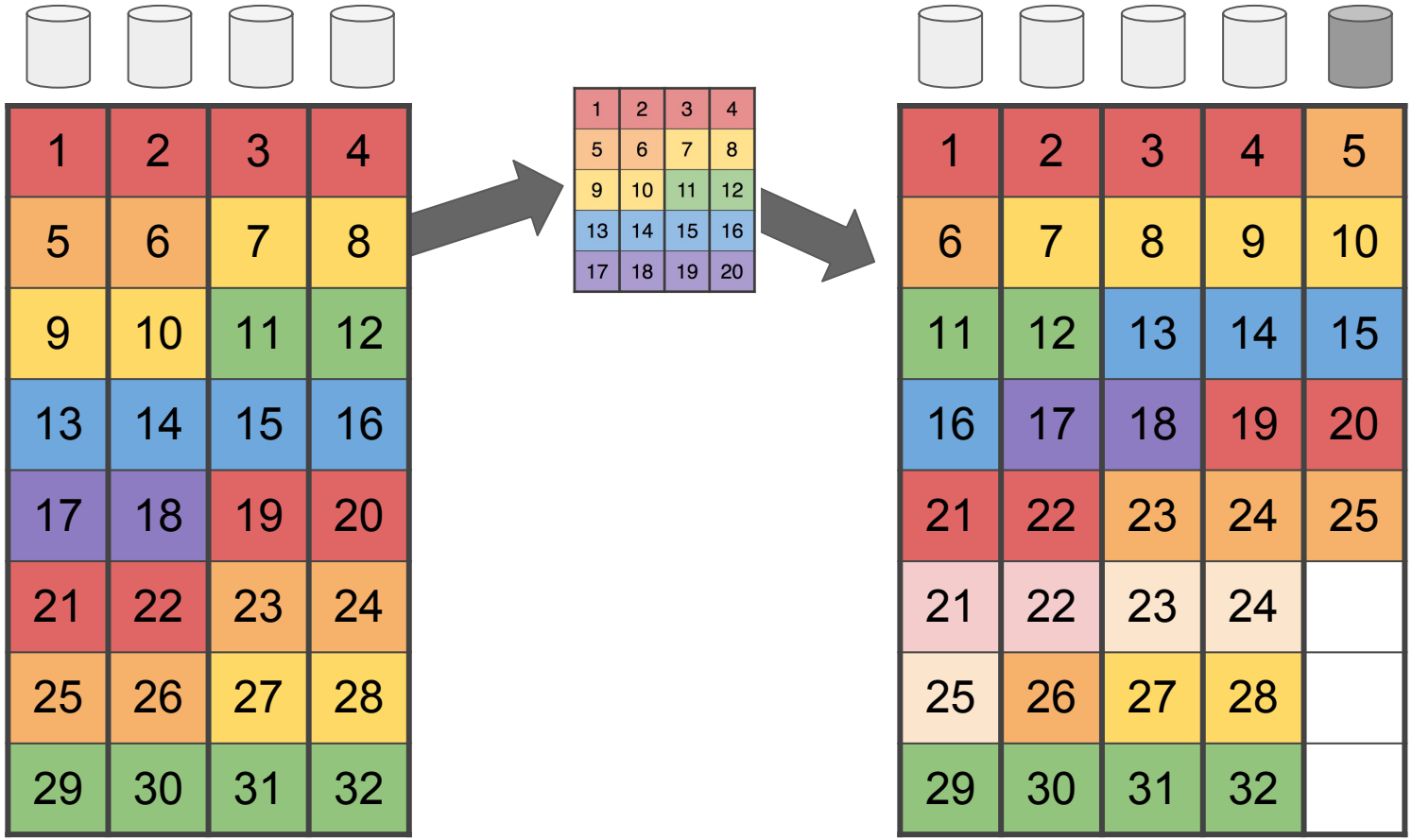
- After conversion, physical stripe width is 5
- Old blocks still have logical stripe width of 4
- New blocks have logical stripe width of 5
  - Improved data : parity ratio (4:1 instead of 3:1)
- When reading, need to know logical stripe width
  - Use block’s birth time (+ expansion time) to determine

Hard to get started

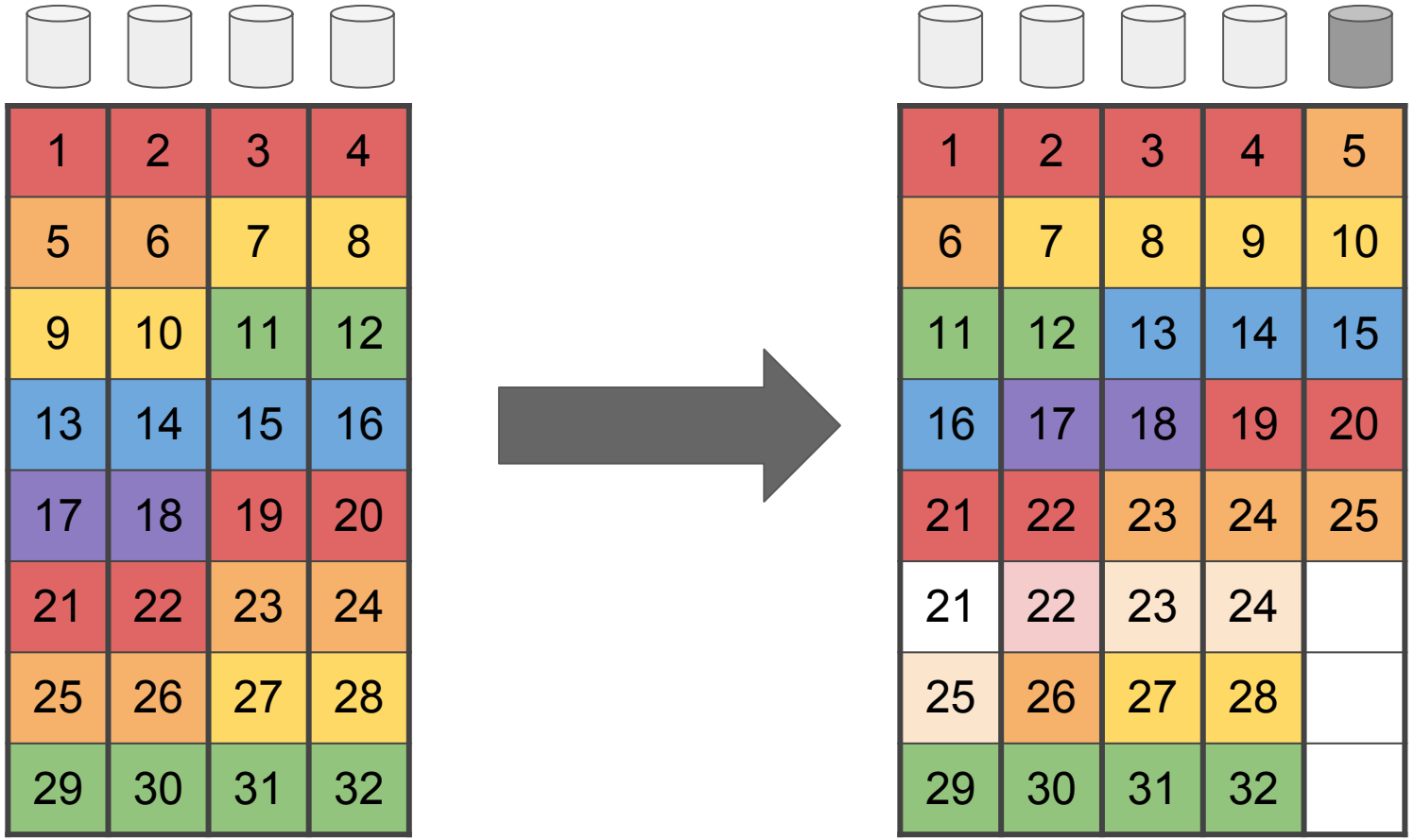


Color indicates logical stripe; white=unused

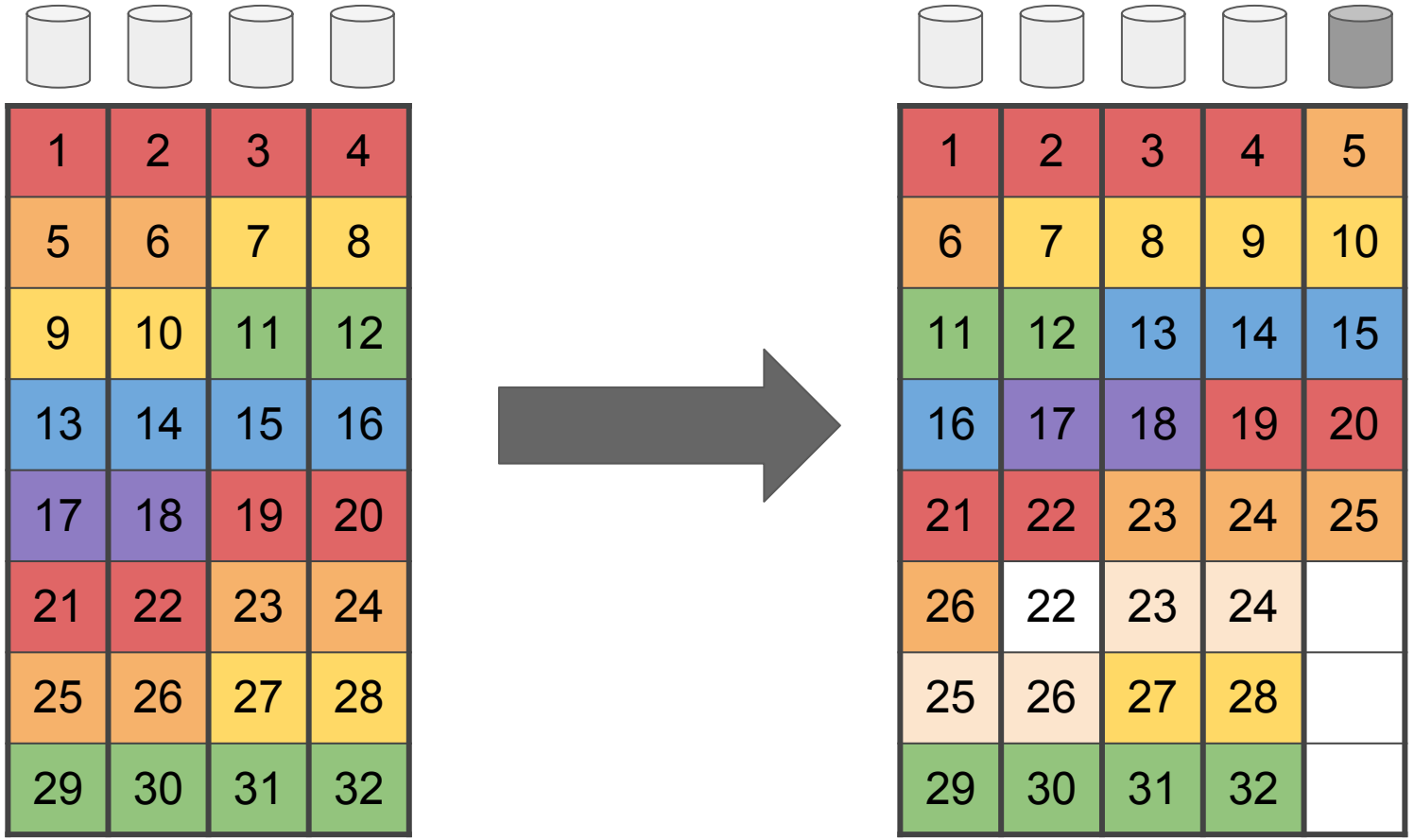
Read at least 5*5=25 sectors into scratch object



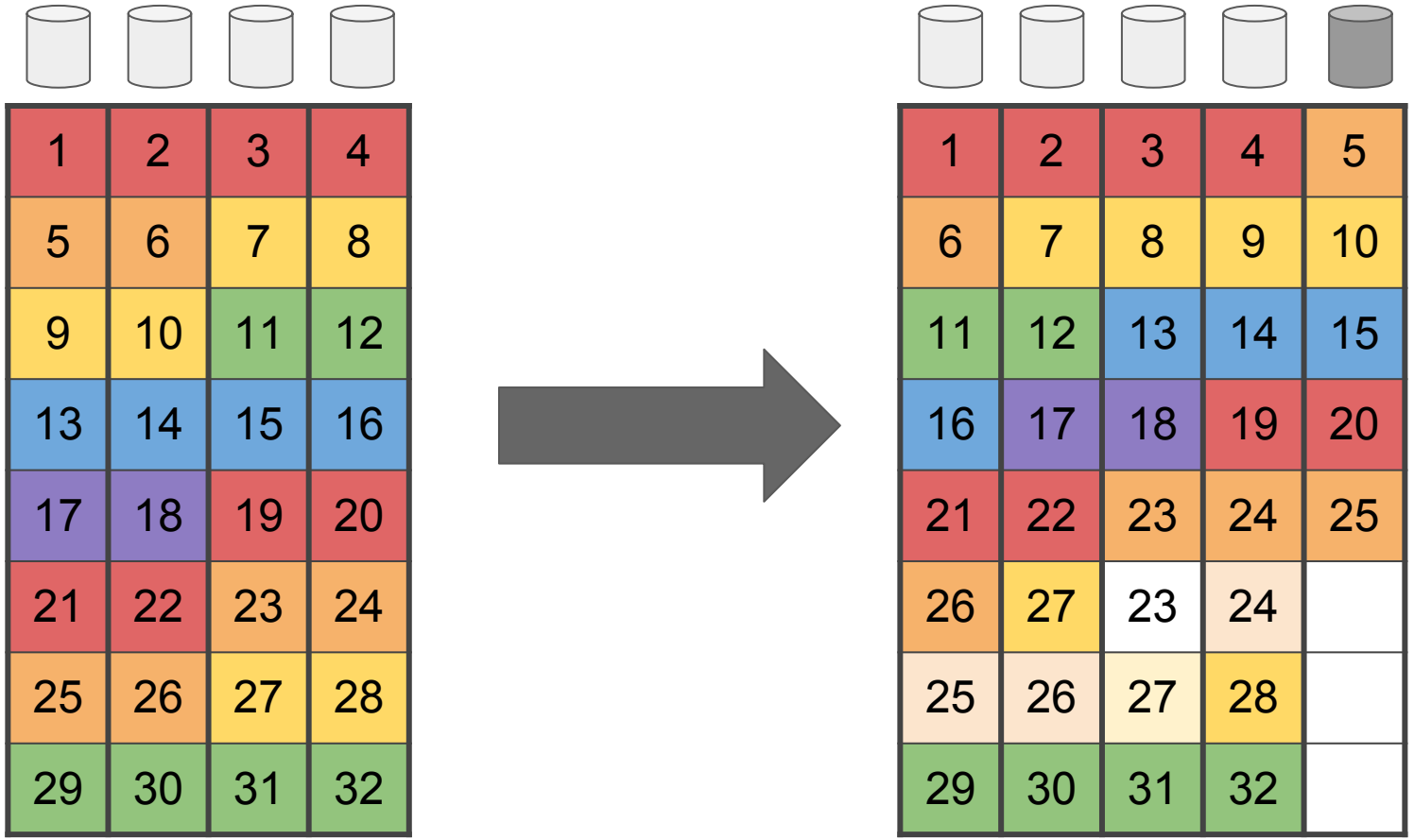
Reflow progress = 25; separation=6; chunk size = 1



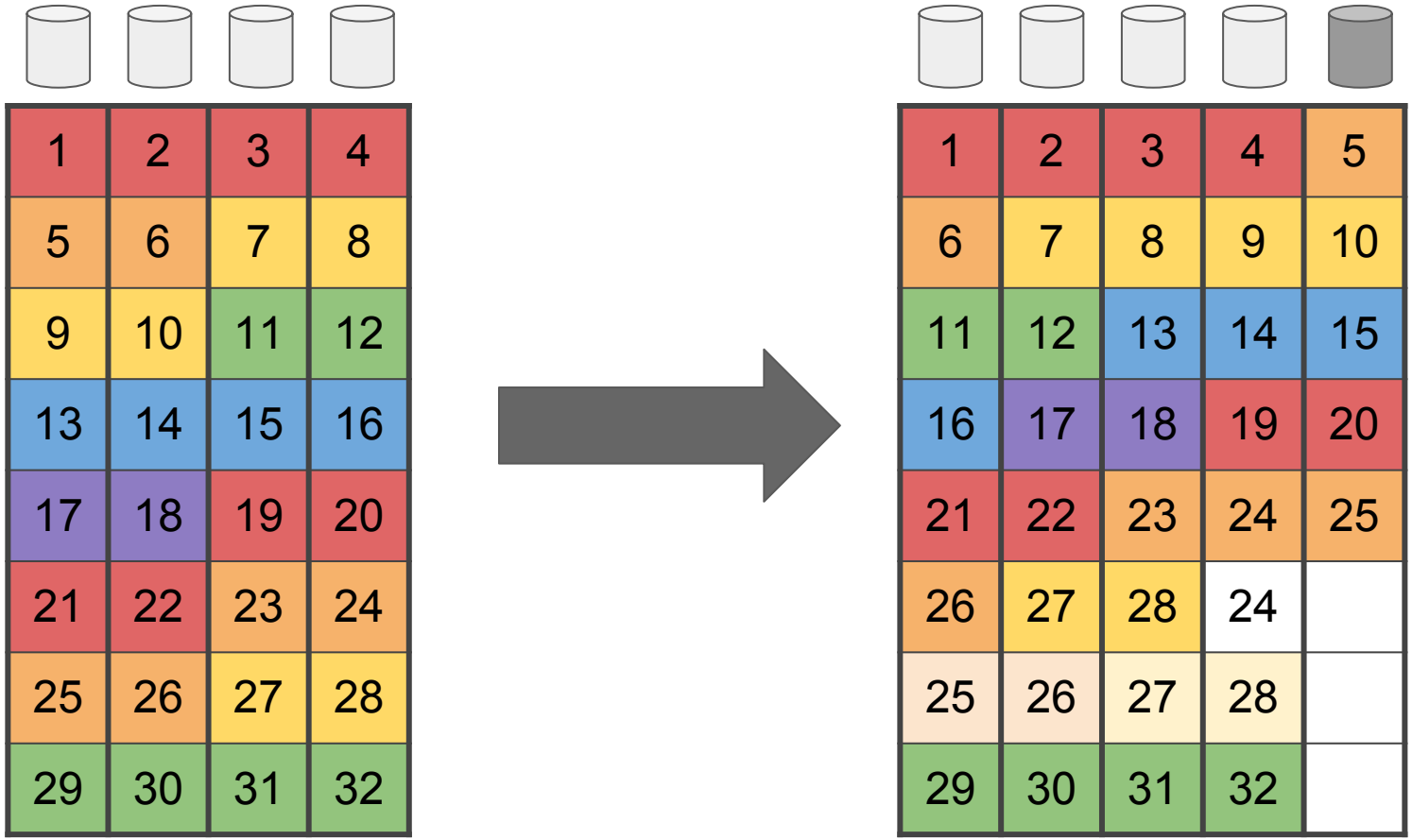
Reflow progress = 26; separation=6; chunk size = 1



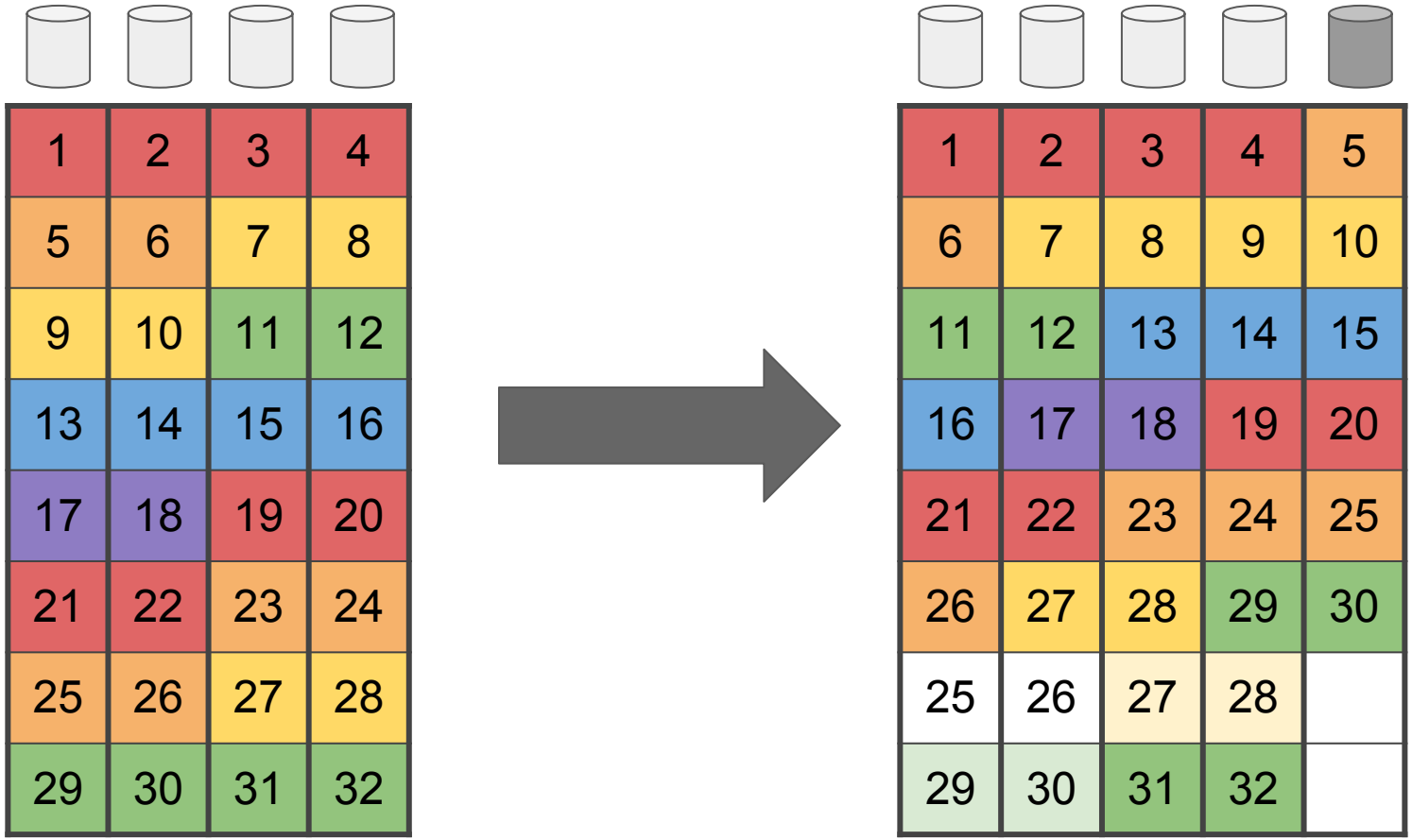
Reflow progress = 27; separation=6; chunk size = 1



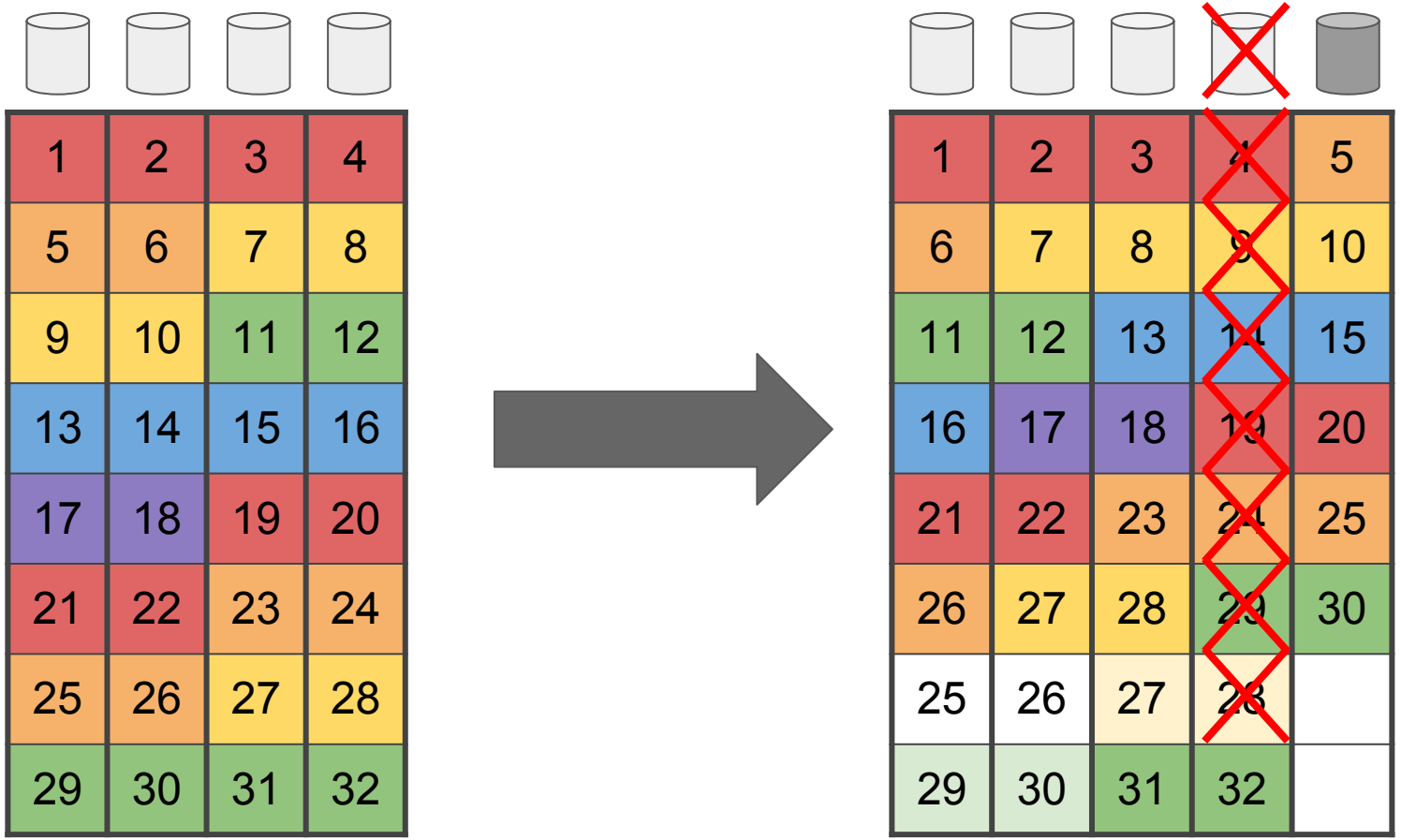
Reflow progress = 28; separation=7; chunk size = 2



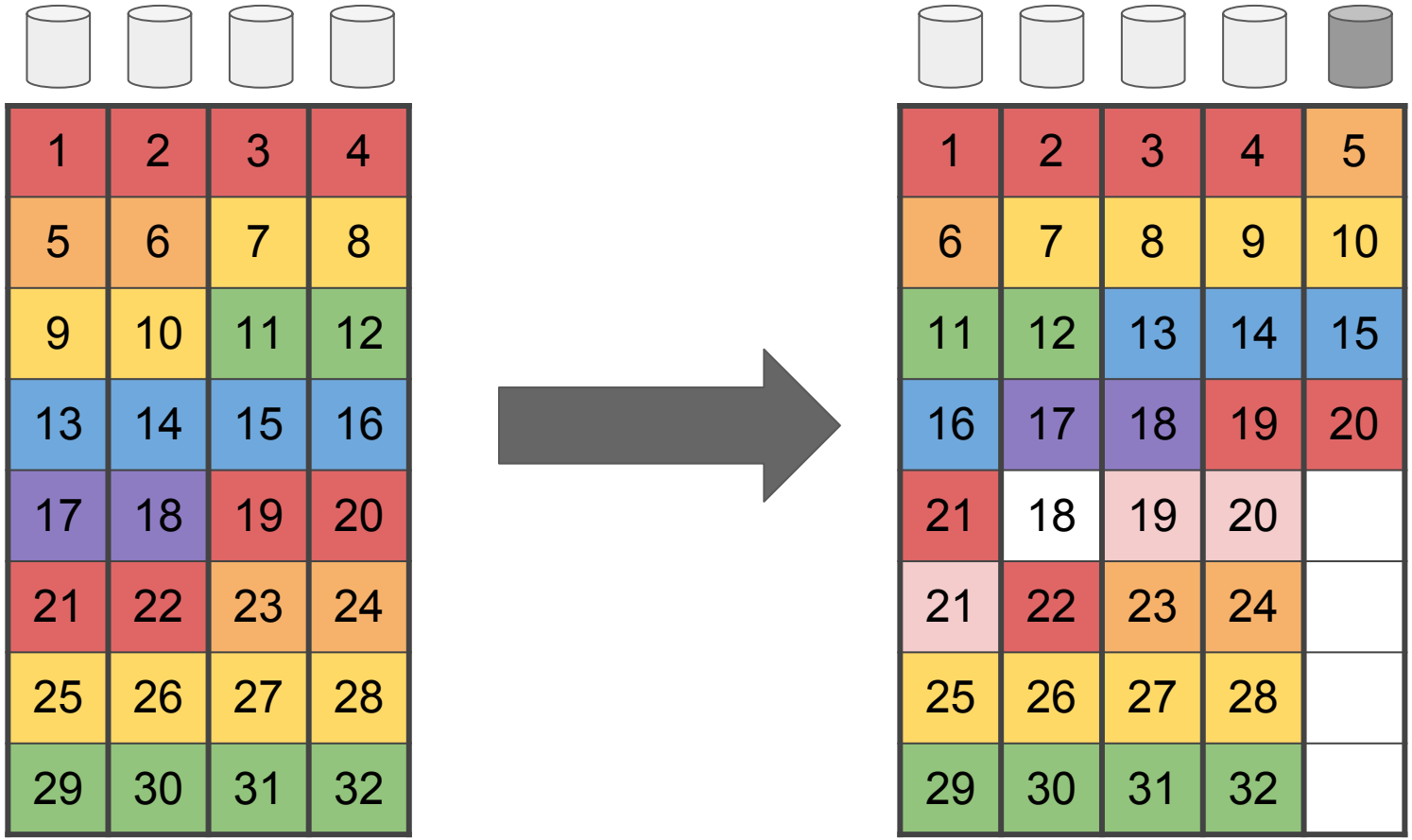
Reflow progress = 28; separation=7; chunk size = 2



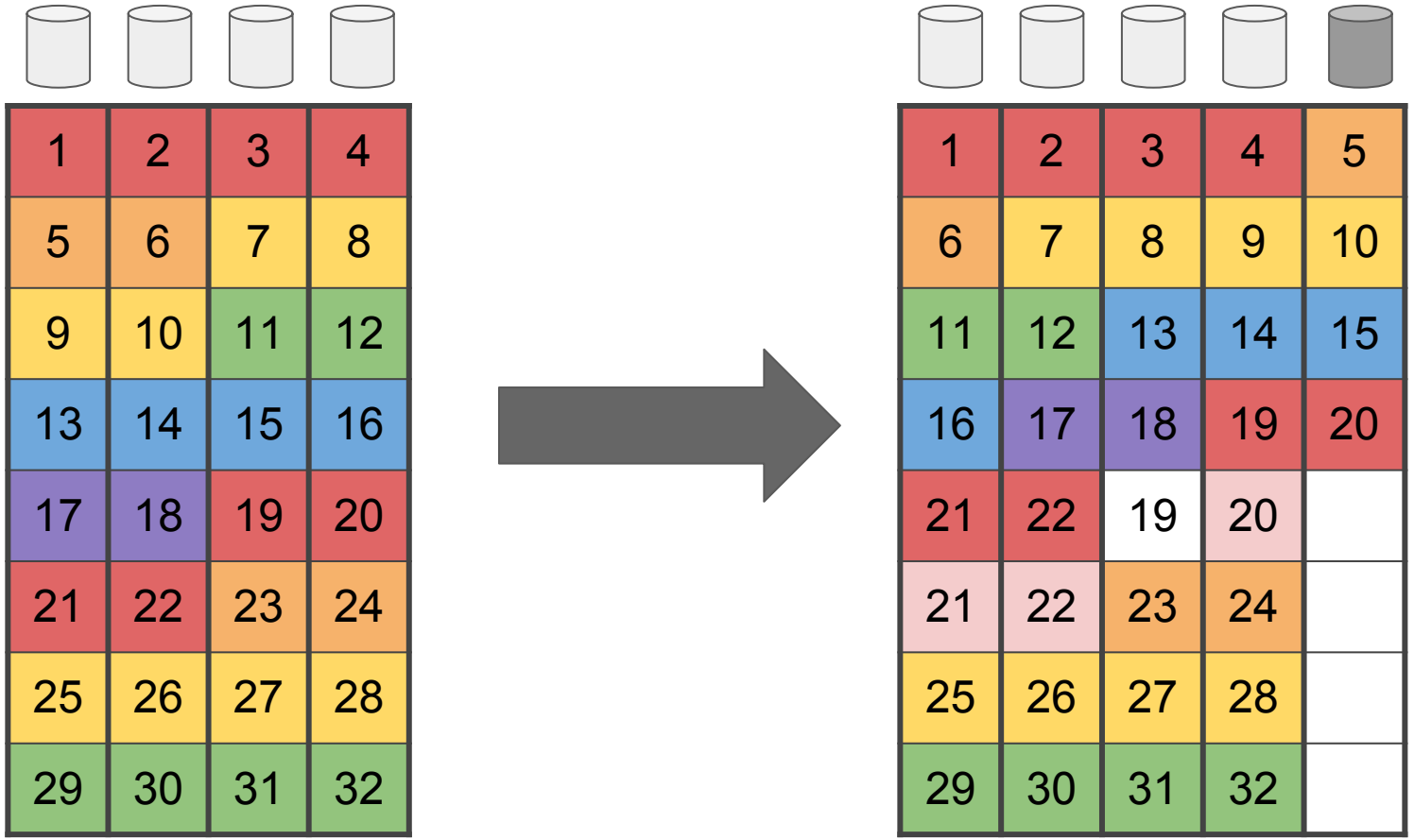
Reflow progress = 28; separation=7; chunk size = 2



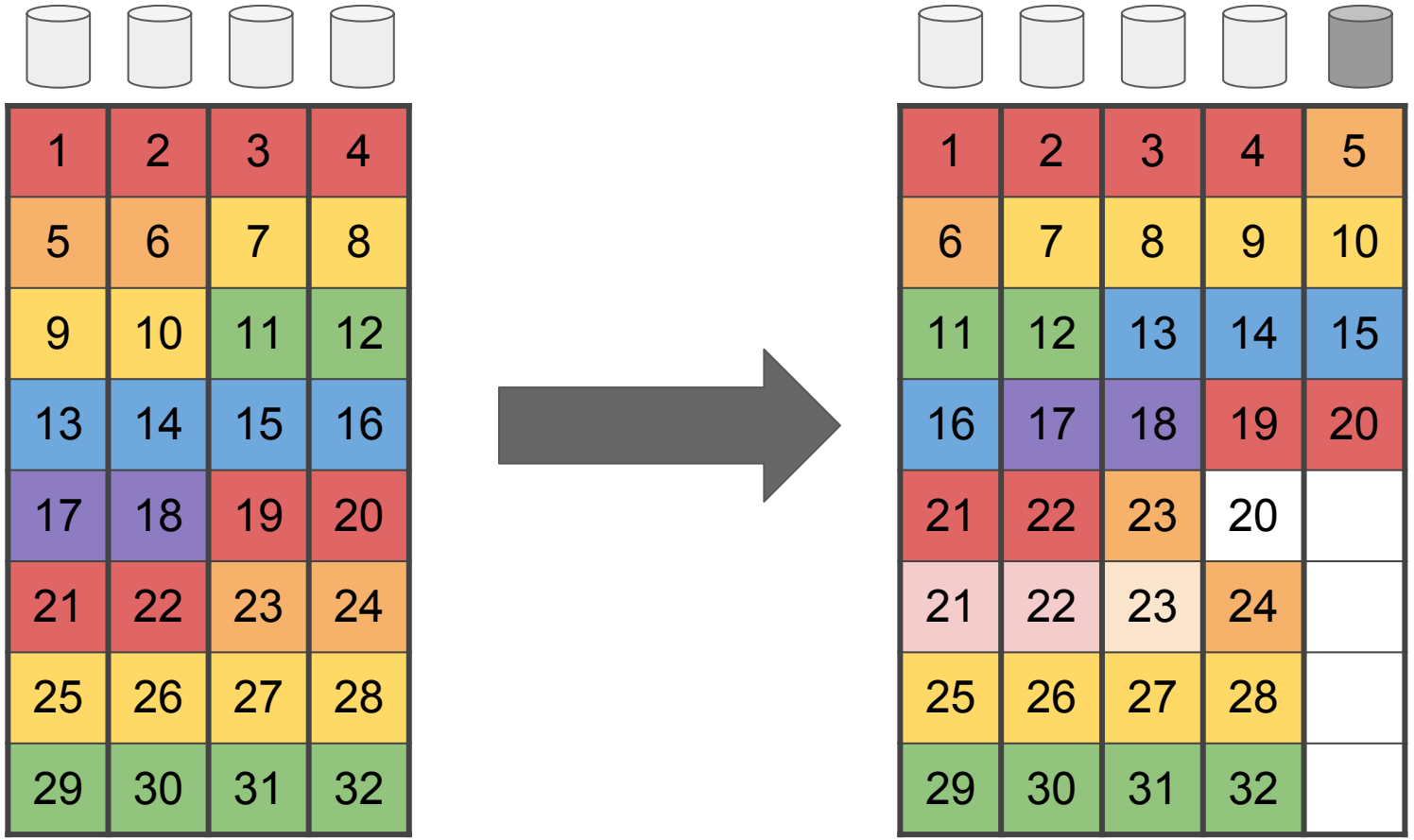
Reflow progress = 21; separation=5; chunk size = 1



Reflow progress = 22; separation=5; chunk size = 1

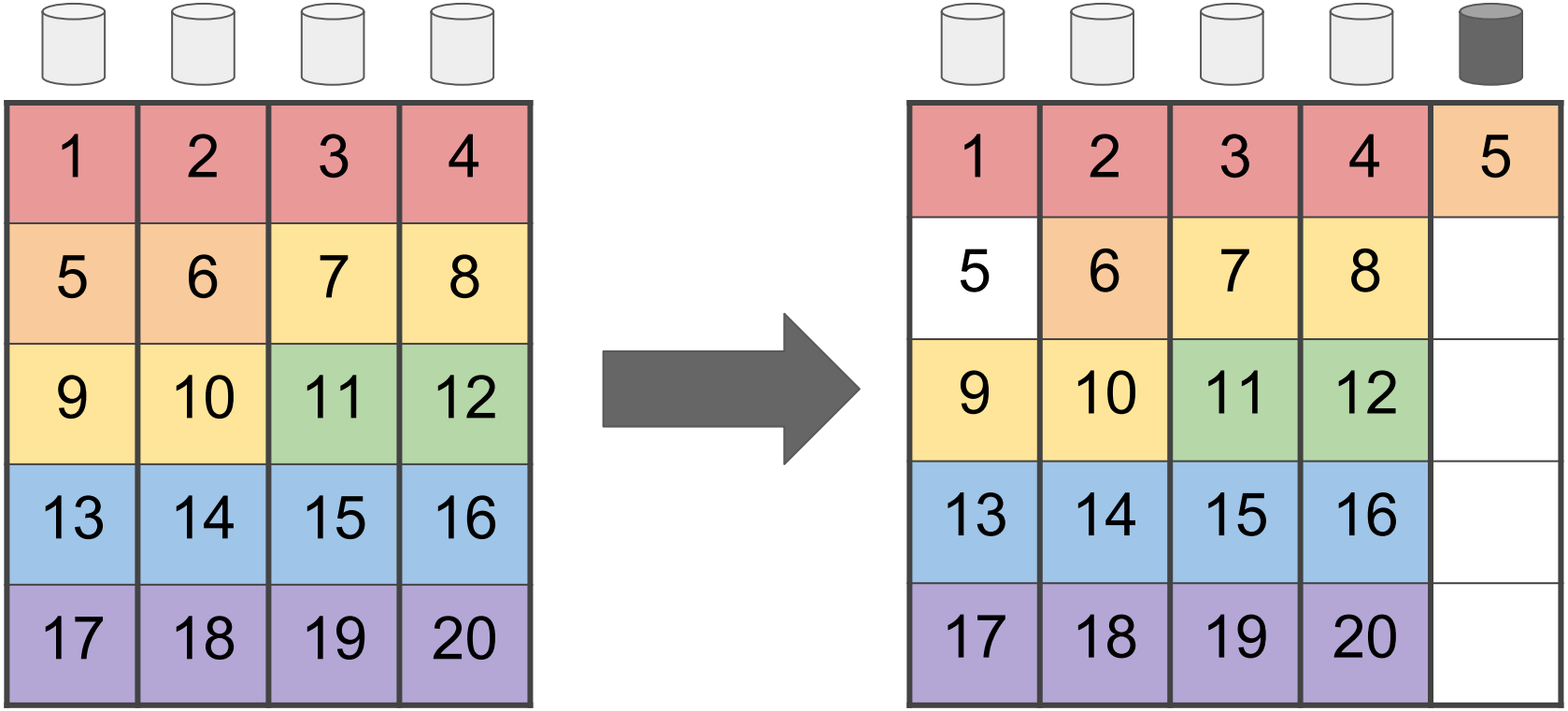


Reflow progress = 23; separation=5; chunk size = 1



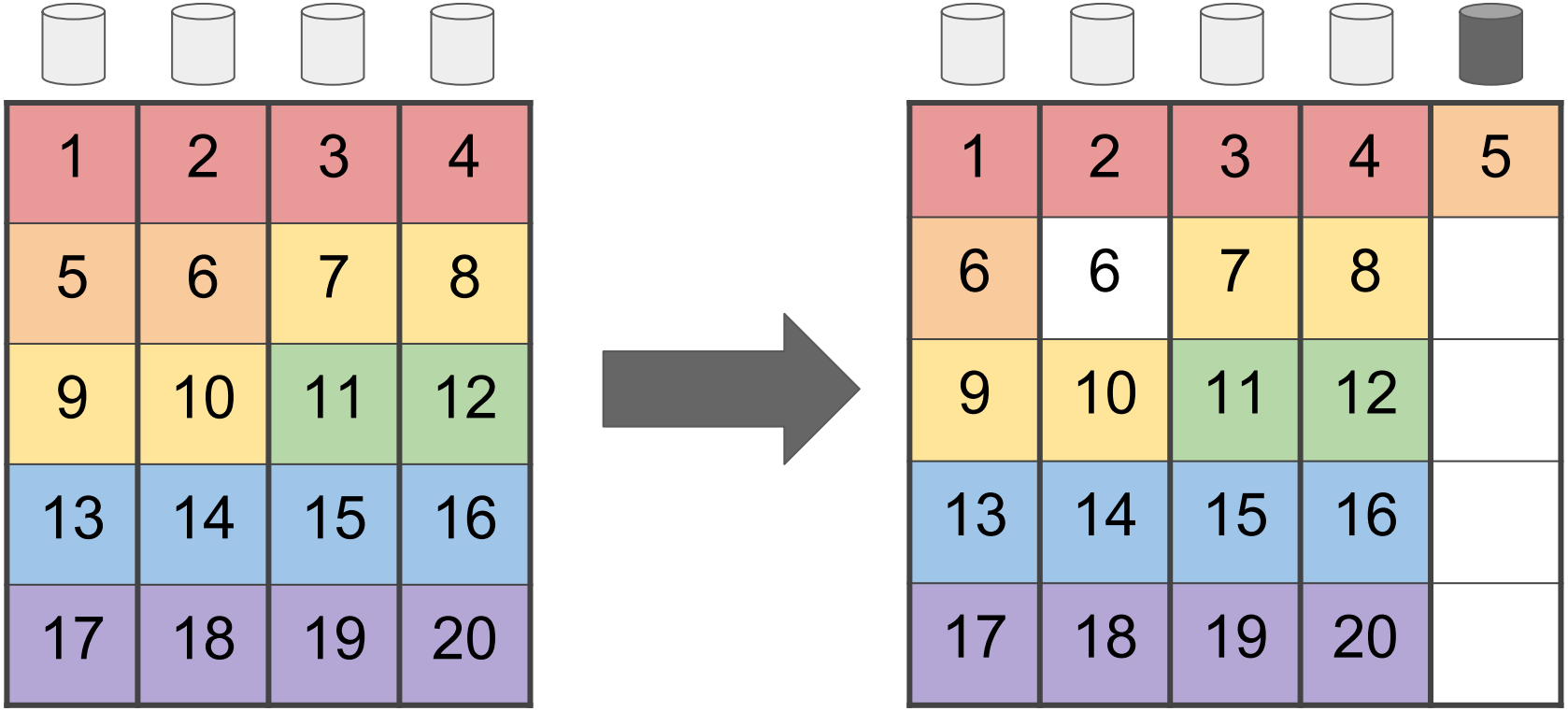

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20

Reflow Progress = 5; Chunk Size = 1



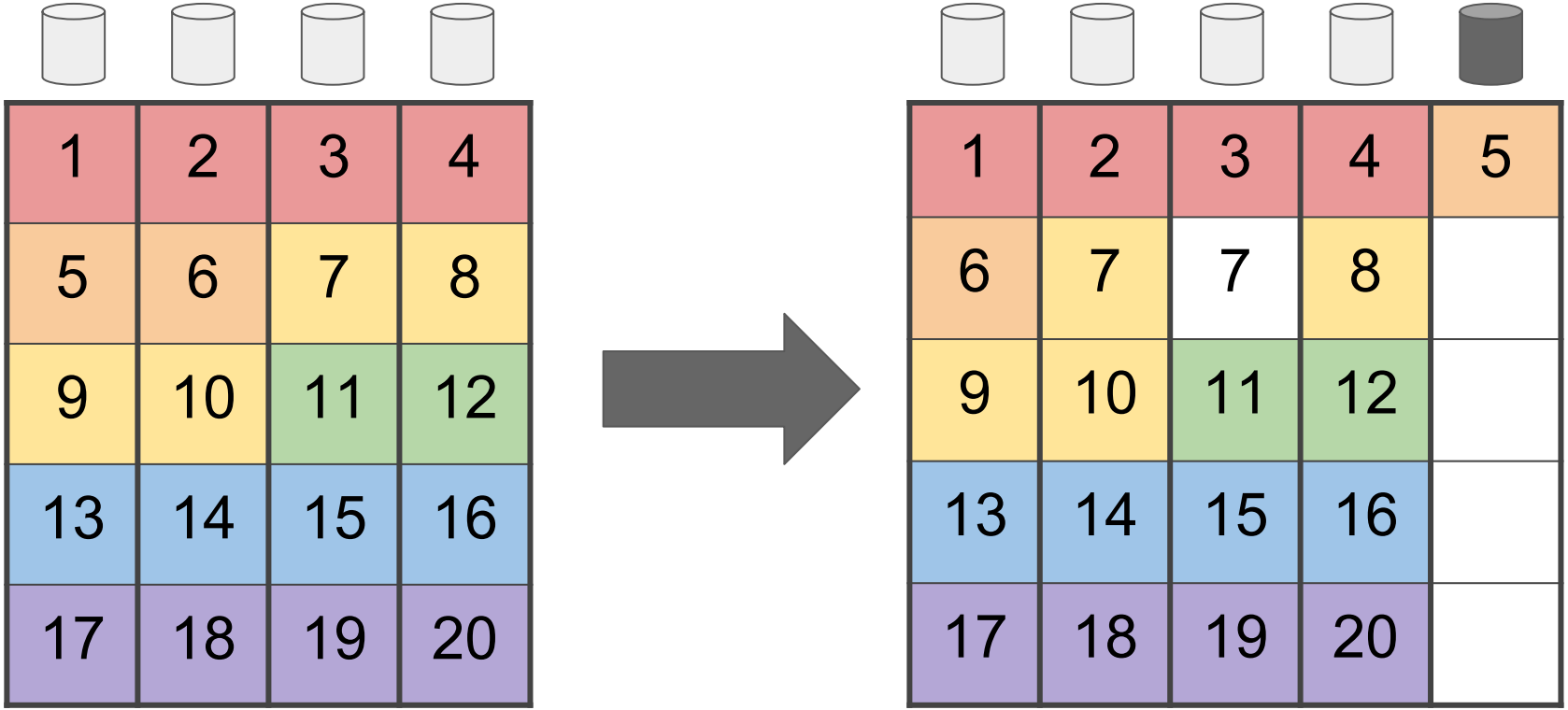
Color indicates logical stripe; white=unused

Reflow Progress = 6; Chunk Size = 1



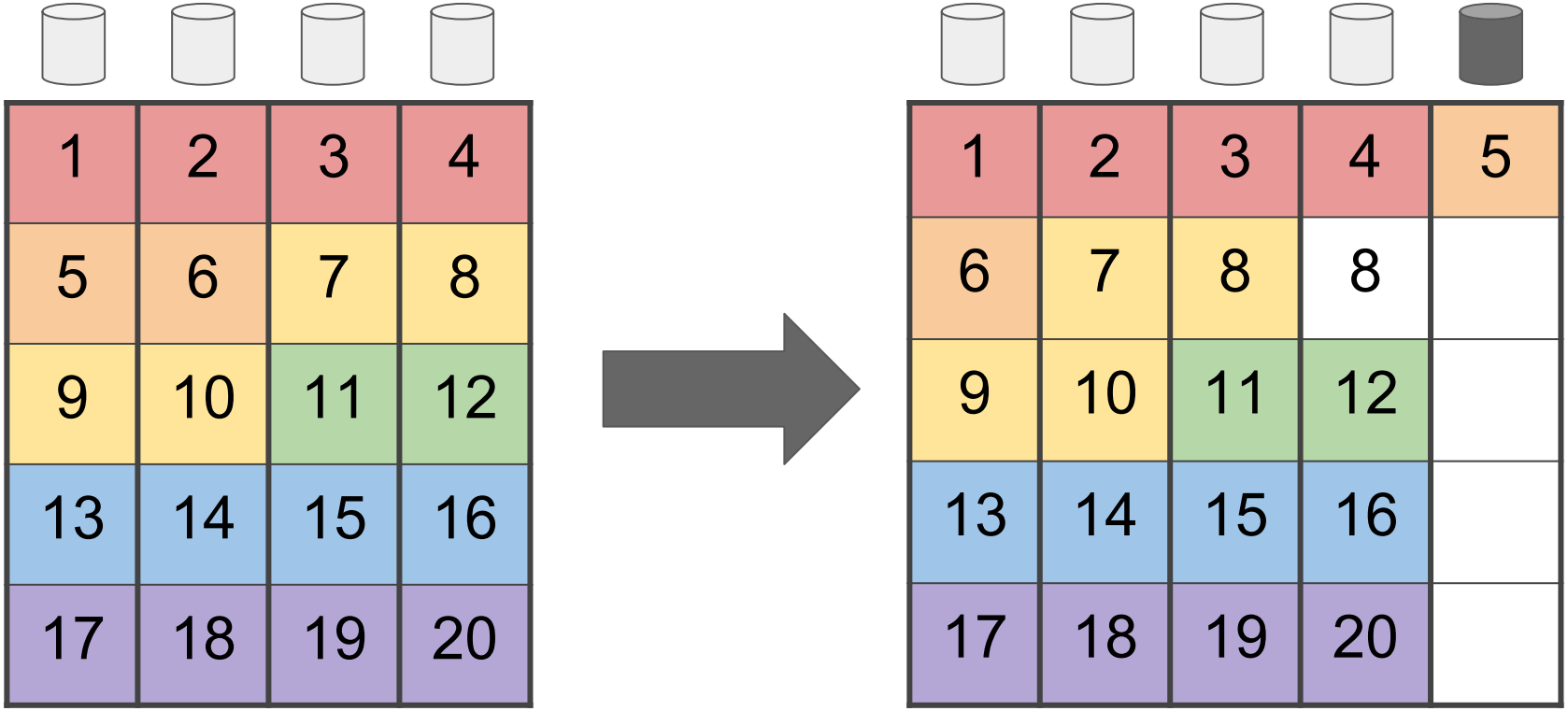
Color indicates logical stripe; white=unused

Reflow Progress = 7; Chunk Size = 1



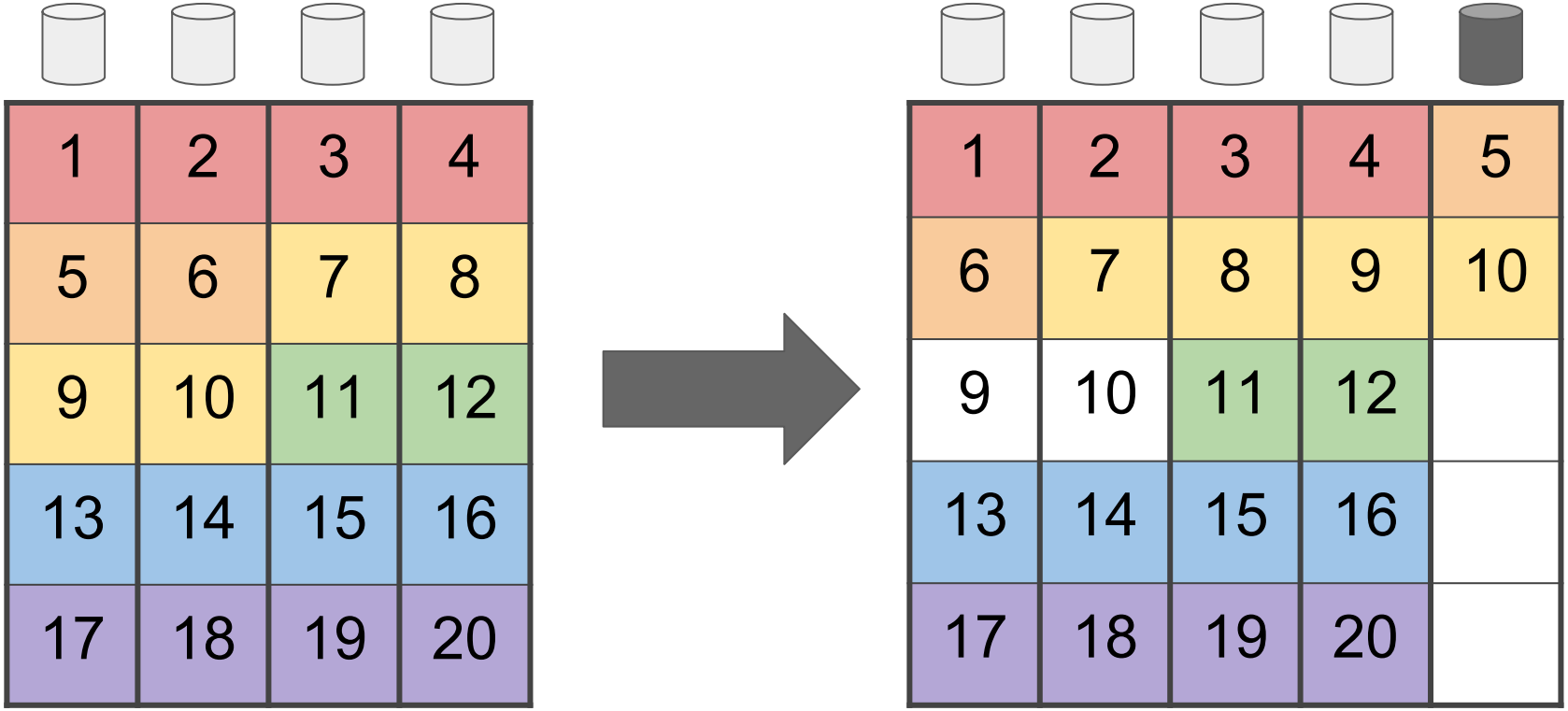
Color indicates logical stripe; white=unused

Reflow Progress = 8; Chunk Size = 2



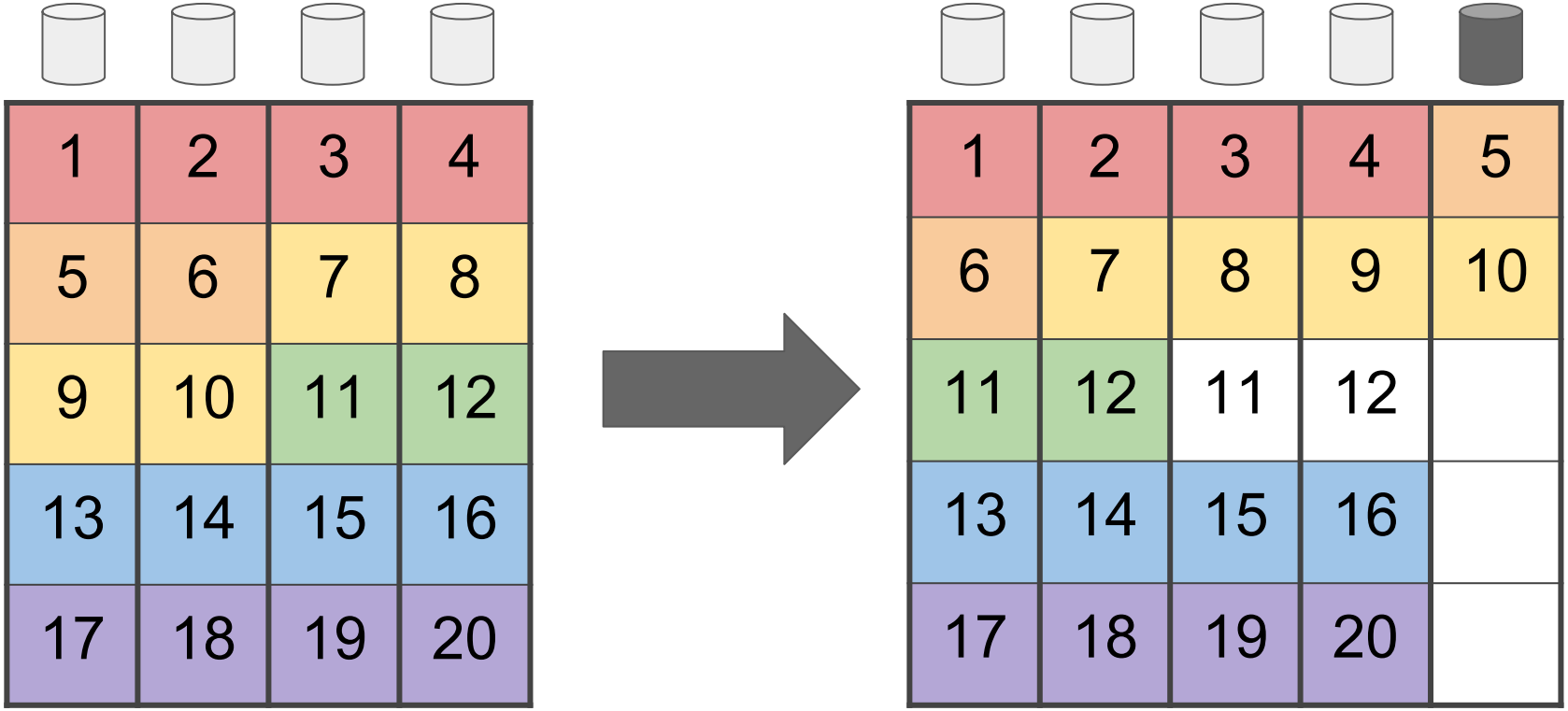
Color indicates logical stripe; white=unused

Reflow Progress = 10; Chunk Size = 2



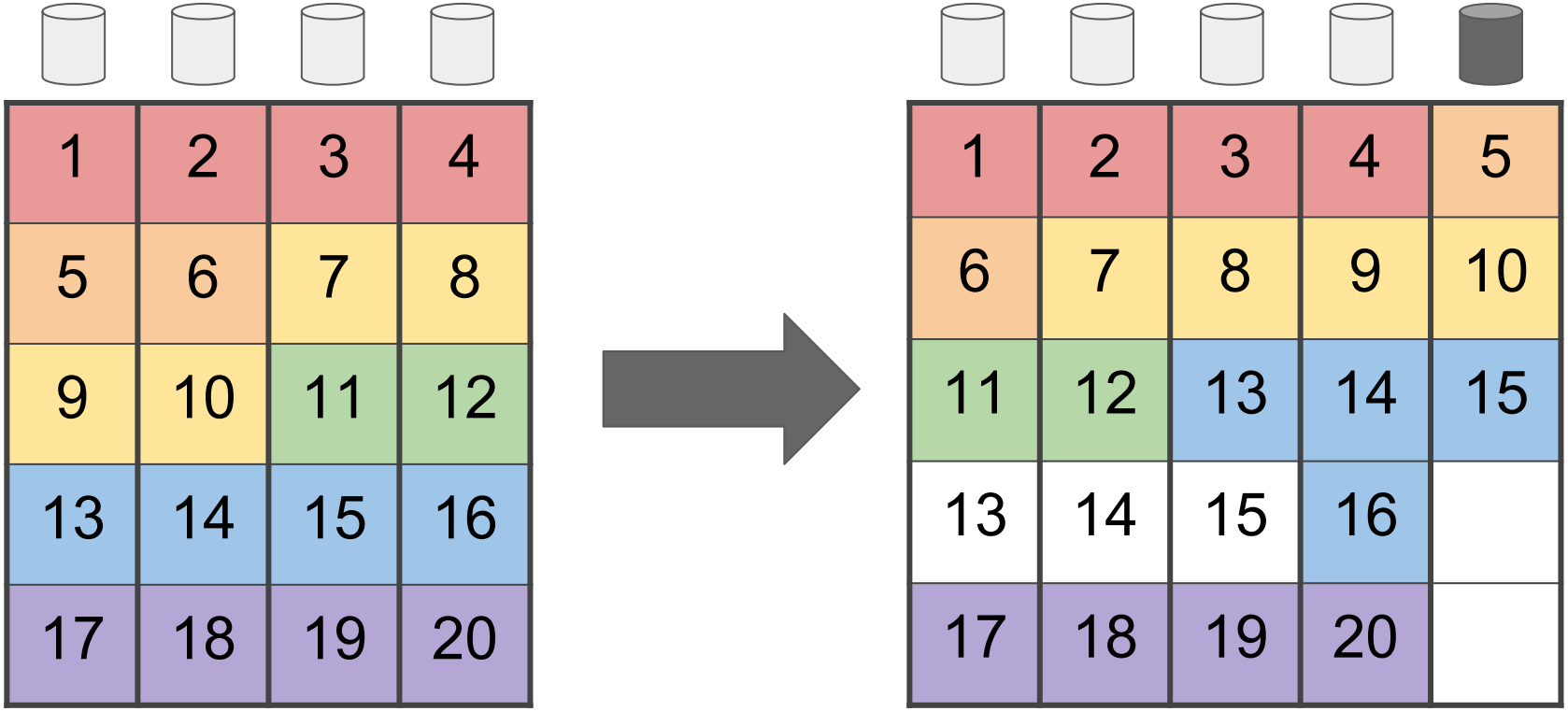
Color indicates logical stripe; white=unused

Reflow Progress = 12; Chunk Size = 3



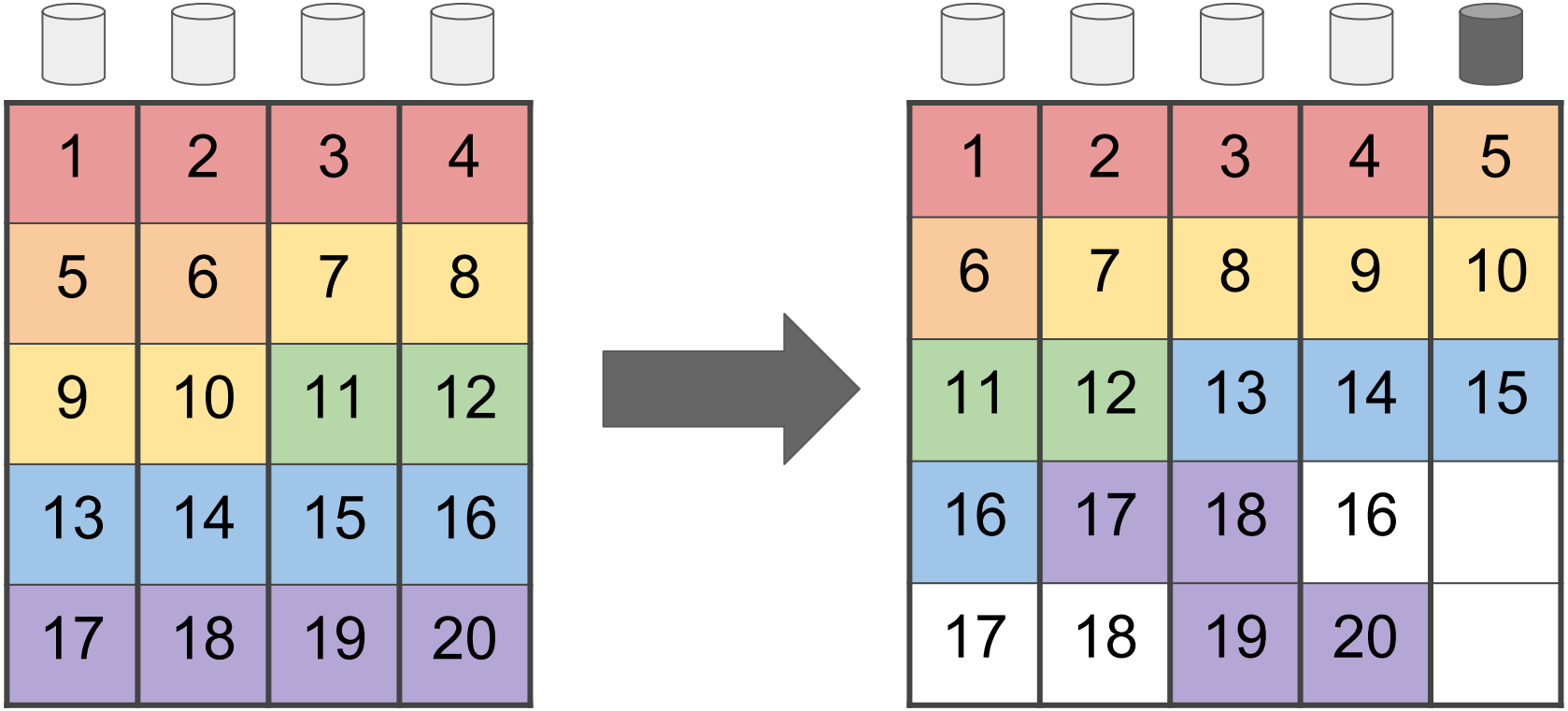
Color indicates logical stripe; white=unused

Reflow Progress = 15; Chunk Size = 3



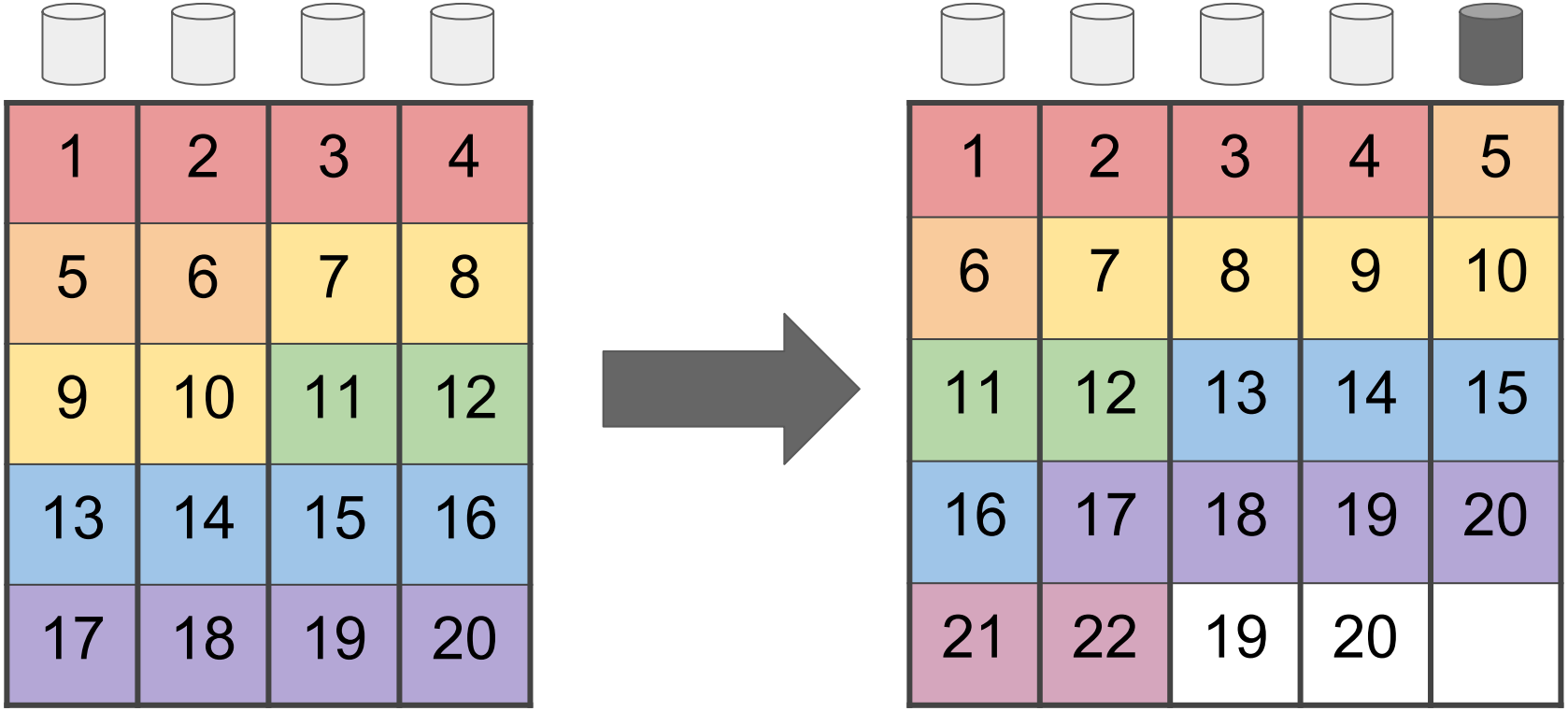
Color indicates logical stripe; white=unused

Reflow Progress = 18; Chunk Size = 4



Color indicates logical stripe; white=unused

Reflow Progress = 22; Chunk Size = 5



Color indicates logical stripe; white=unused

Reflow process

- Need to track progress to know the physical stripe width
- Each TXG can only overwrite what’s previously unused
- Exponential increase in amount that can be copied per TXG
  - 41 TXG’s for 1MB
  - 113 TXG’s for 1GB
  - 186 TXG’s for 1TB
  - 258 TXG’s for 1PB

Design implications

- Works with RAIDZ-1/2/3
- Can expand multiple times (4-wide -> 5 wide -> 6 wide)
- Old data has old Data : Parity ratio
- New data has new Data : Parity ratio
- RAIDZ must be healthy (no missing devices) during reflow
  - If disk dies, reflow will pause and wait for it to be reconstructed
- Reflow works in the background

Thank you!

Status

- High level design complete
- Detailed design 80% complete
- Zero lines of code written
- Expect significant updates at BSDCAN (June 2018) and next year's DevSummit (~October 2018)