I/O, Files, & Storage Devices (Part II)

Professor Travis Peters
CSCI 460 Operating Systems
Fall 2019

Some slides & figures adapted from Stallings instructor resources.

Some slides adapted from Adam Bates's F'18 CS423 course @ UIUC
https://courses.engr.illinois.edu/cs423/sp2018/schedule.html

Some content adapted from the Disk Scheduling Algorithms tutorial
http://www.cs.iit.edu/~cs561/cs450/disksched/disksched.html

Today

Announcements

- Project Proposals Due **TONIGHT @ 10PM**!
- Exam #2 in class **NEXT FRIDAY**!
- PA2 Due **NEXT MONDAY (11/11)!** — *thank Parker for requesting this deadline be pushed…*
- Guest Lecture with Will Peteroy on Monday

  Read before class!

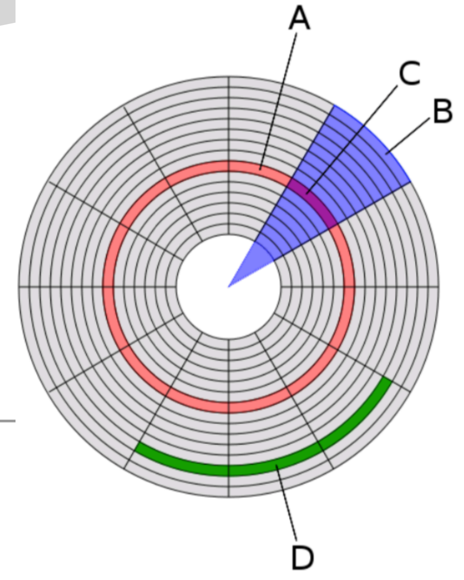
  - https://msrc-blog.microsoft.com/2010/12/08/on-the-effectiveness-of-dep-and-aslr/
  - https://msrc-blog.microsoft.com/2013/08/12/mitigating-the-ldrhotpatchroutine-depaslr-bypass-with-ms13-063/
  - https://arstechnica.com/information-technology/2019/08/armed-with-ios-0days-hackers-indiscriminately-infected-iphones-for-two-years/

Goals & Learning Objectives

- Understand key concepts behind I/O devices and functions
- Understand some of the key issues in the design of OS support for I/O
- Understand basics of secondary storage (emphasis on disks and disk scheduling)
- Understand basics behind files and file systems

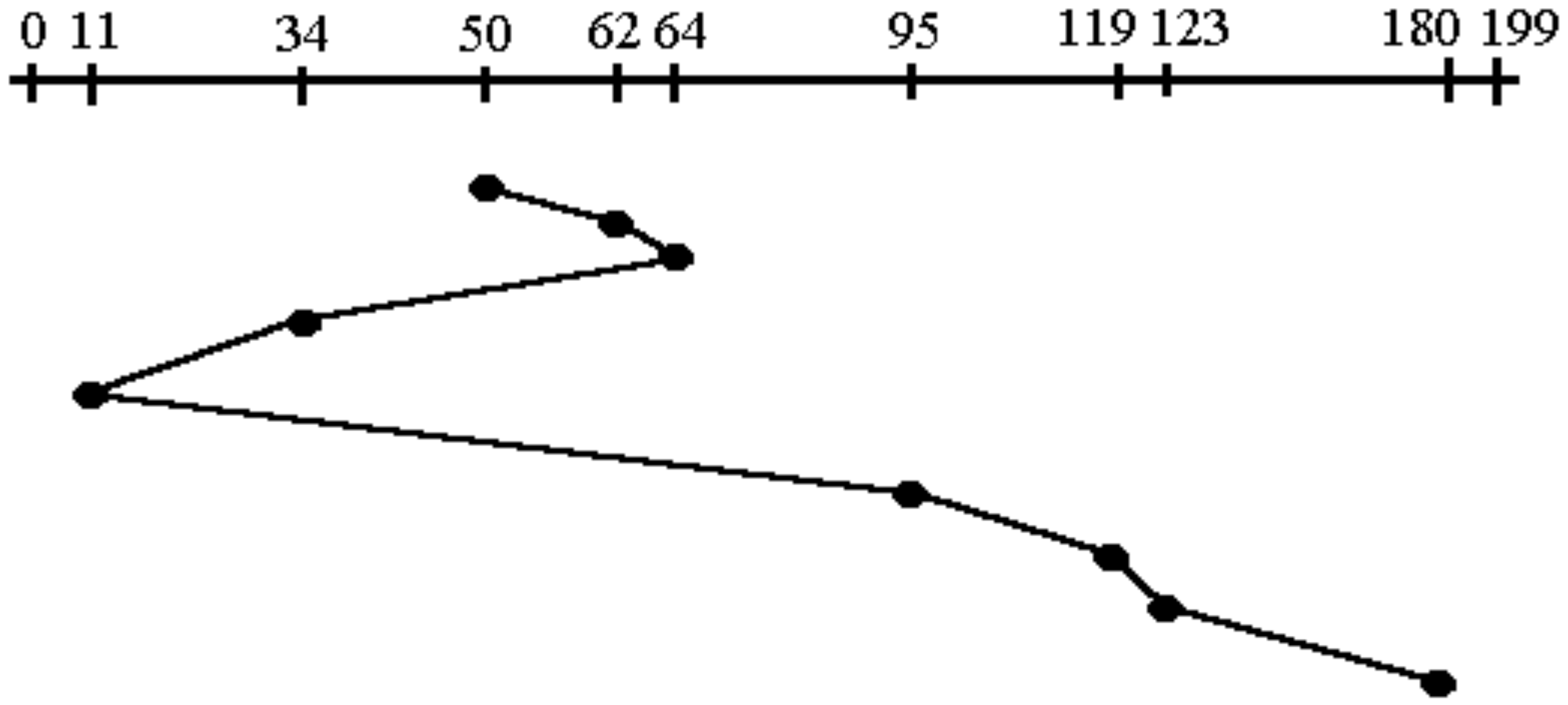
Disk Scheduling Policies *(cont.)*

A = Track / B = Sector / C = Sector of Track / D = File

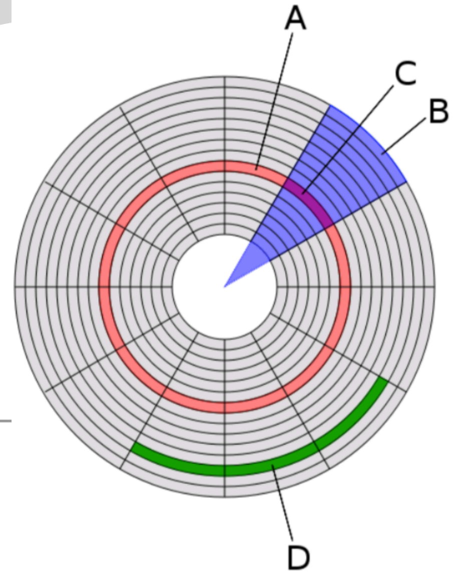


Shortest-Service-Time-First (SSTF)

- Select I/O ops that require the least amount of movement from disk arm's current position
- Choose next op that incurs minimum seek time
- ***Pros?***
  - Try to minimize seek time…
- ***Cons?***
  - Is SSTF optimal?
  - Are we worried about overhead of sorting?
  - Can we avoid starvation?



Disk Scheduling Policies *(cont.)*

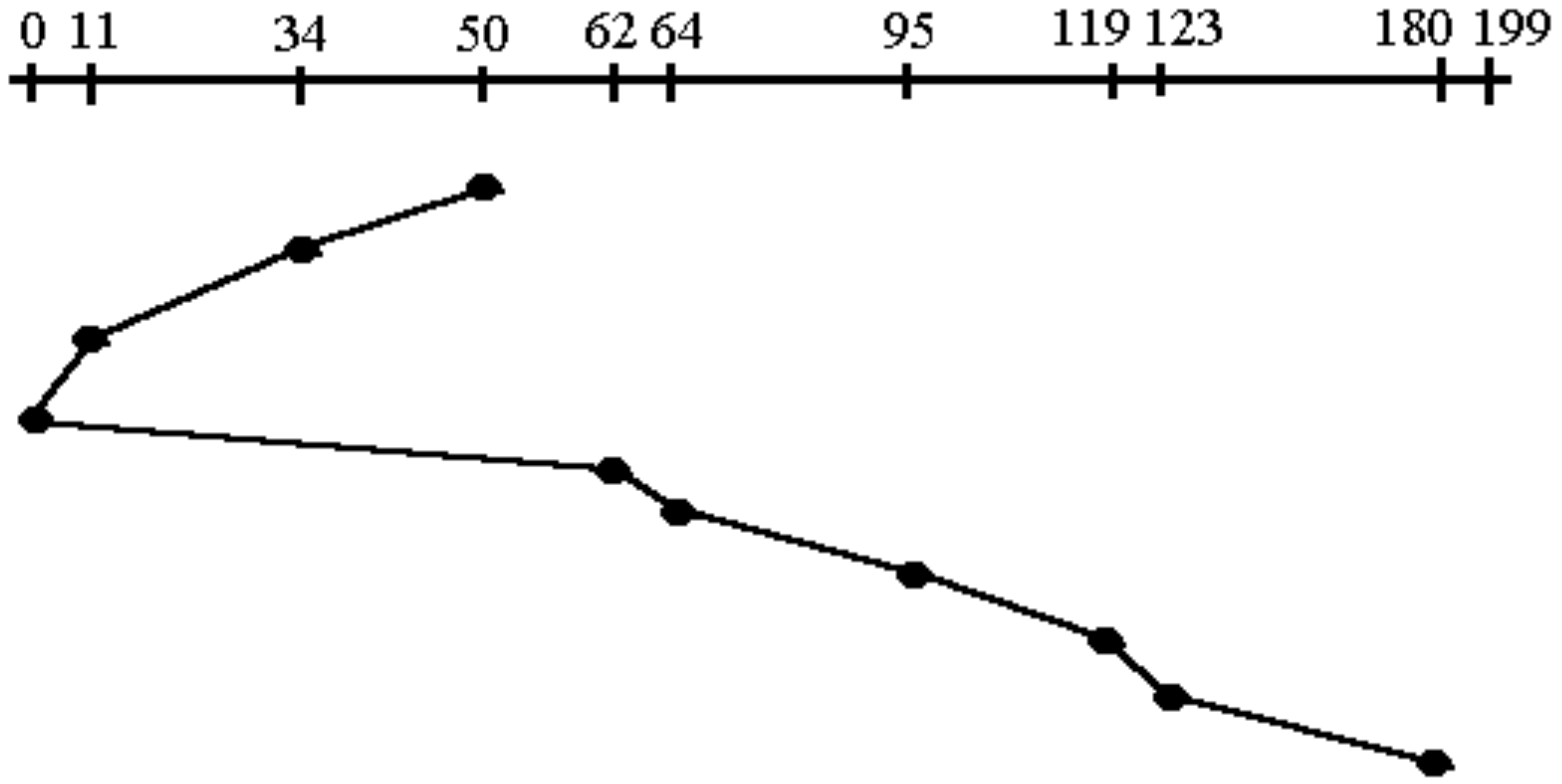


A = Track / B = Sector / C = Sector of Track / D = File

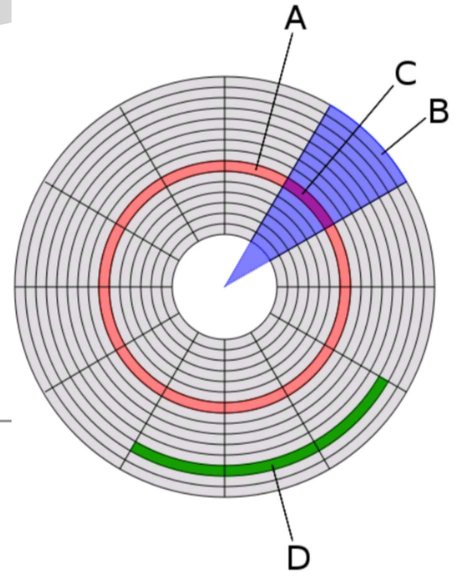
SCAN *(a.k.a. The Elevator Algorithm)*

- The disk arm moves in one direction, servicing the closest request in the direction of travel
- Continue in that direction until…
  - last track is reached, or
  - there are no more requests to service in that direction **(→ LOOK policy)**
- **Pros?**
  - Bounded time for each request
- **Cons?**
  - Requests at the opposite end will take a while…
  - Does not exploit locality…
  - Which sectors have shorter wait times?

    (How to fix this?)



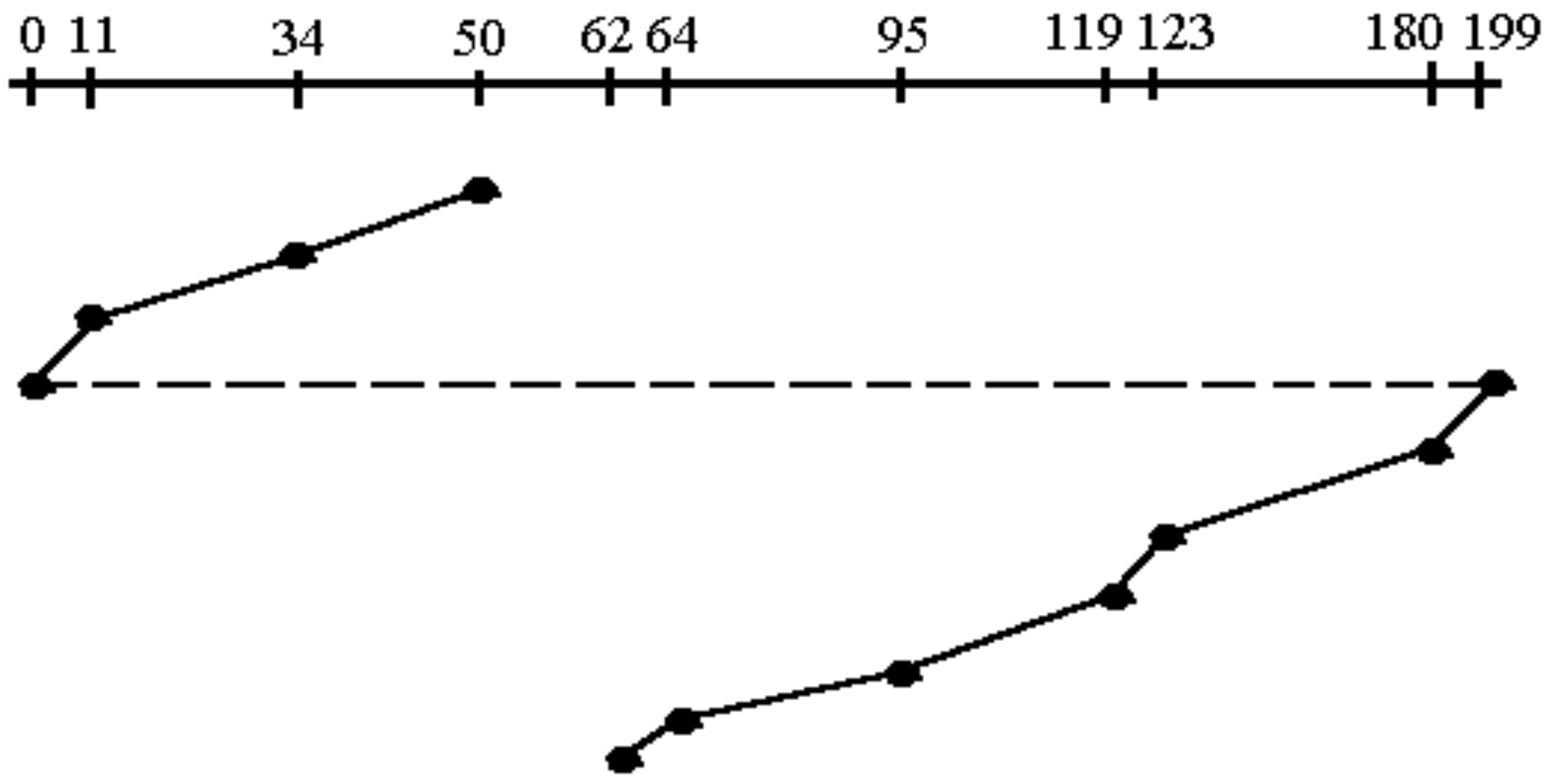
Disk Scheduling Policies *(cont.)*



A = Track / B = Sector / C = Sector of Track / D = File

C-SCAN *(Circular Scan)*

- Similar to SCAN…
  - The disk arm moves in one direction only…
  - …but *NEVER* actually changes directions!
  - When the last track is reached, reset the head ("wrap around") back to the opposite end of the disk and begin again
- **Pros?**
  - Uniform service time
  - Addresses issues with max delay that can occur in regular SCAN
- **Cons?**
  - Do nothing on the return



+ Many more… *N-Step-SCAN, FSCAN, C-LOOK, etc…*

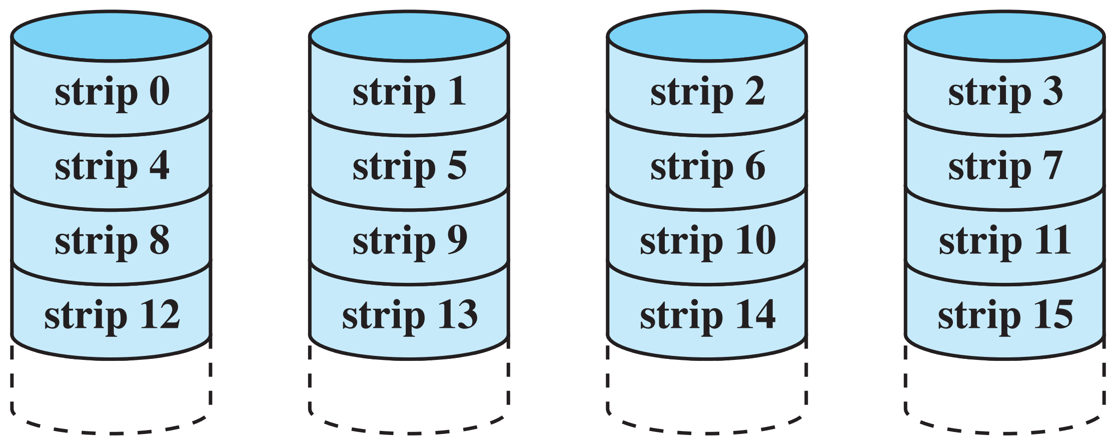
Disks… a bit more on disks!!!

RAID (Redundant Array of Independent Disks)

- Array of independent disks that work **cooperatively** and can be **used in parallel**
  → *Logically, treated as a single disk drive by the OS*
  → *Separate I/O requests can be handled in parallel*
  → *Data distributed/replicated across disks improves reliability **(striping)***

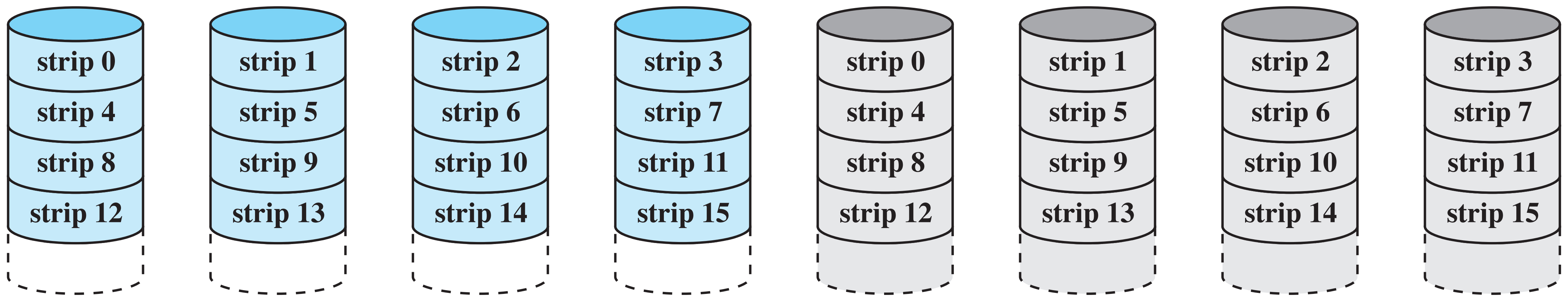
- Various widely-accepted schemes (**RAID0**, **RAID1**, ~~RAID2~~, ~~RAID3~~, ~~RAID4~~, **RAID5**, **RAID6**)
  - Primary contribution was aimed at addressing issues with redundancy
  - Only 4/7 levels are commonly used

RAID Levels (Redundant Array of Independent Disks)



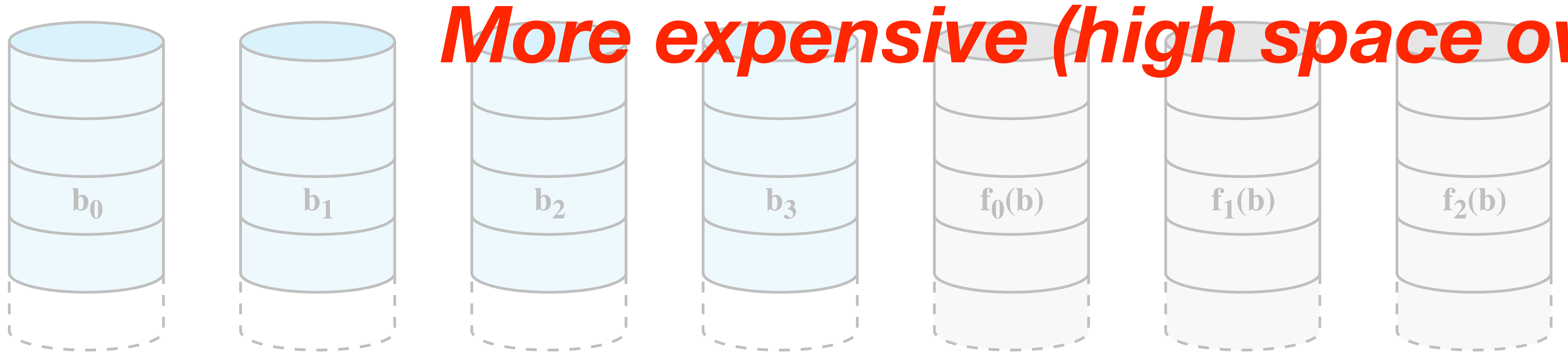
(a) RAID 0 (non-redundant)

Non-redundant
Files are "striped" across disks
Improves performance
Data loss is possible though…

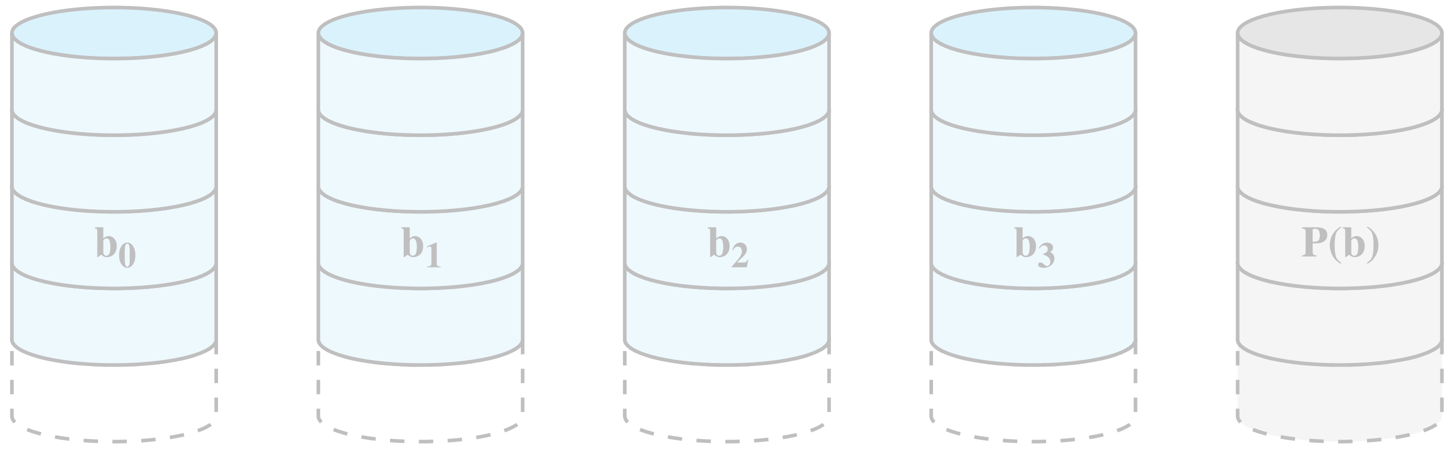


(b) RAID 1 (mirrored)

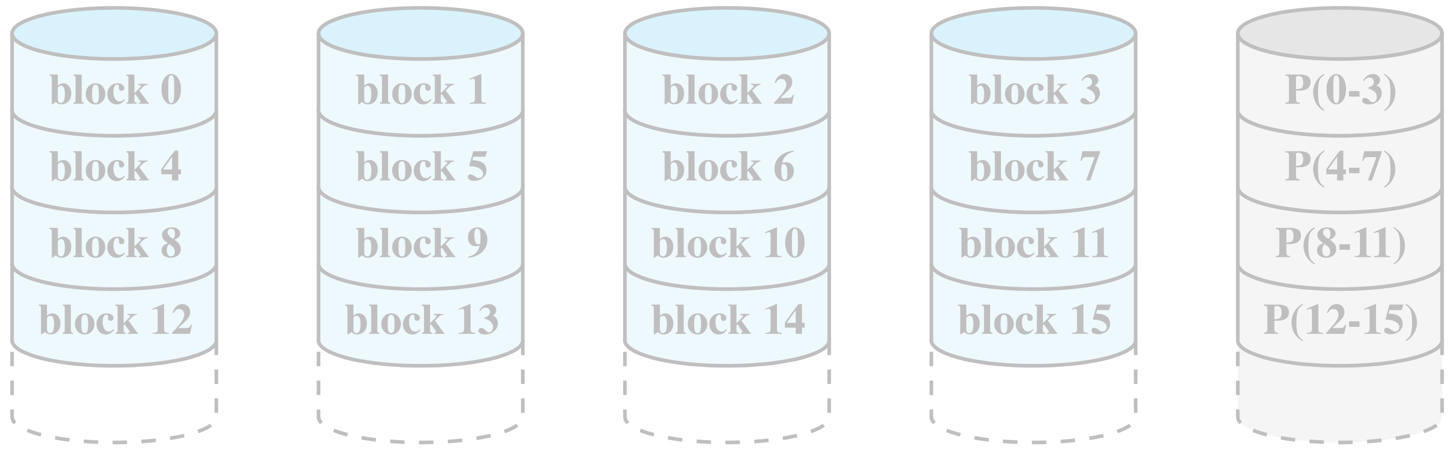
Improves performance + reliability
More expensive (high space overhead)…



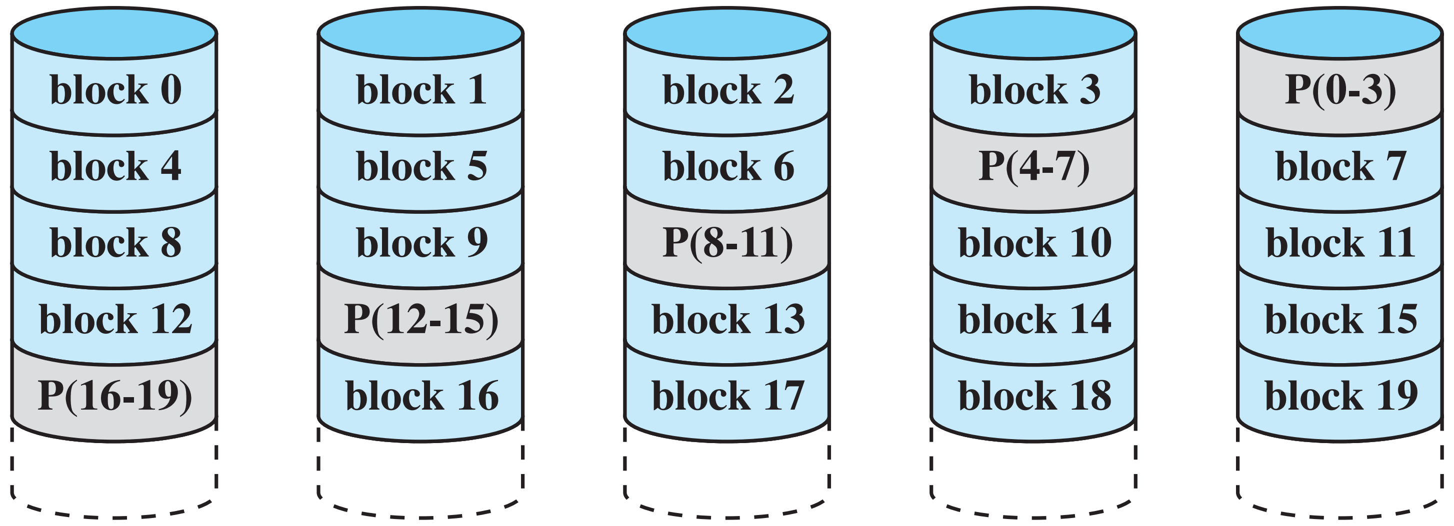
(c) RAID 2 (redundancy through Hamming code)



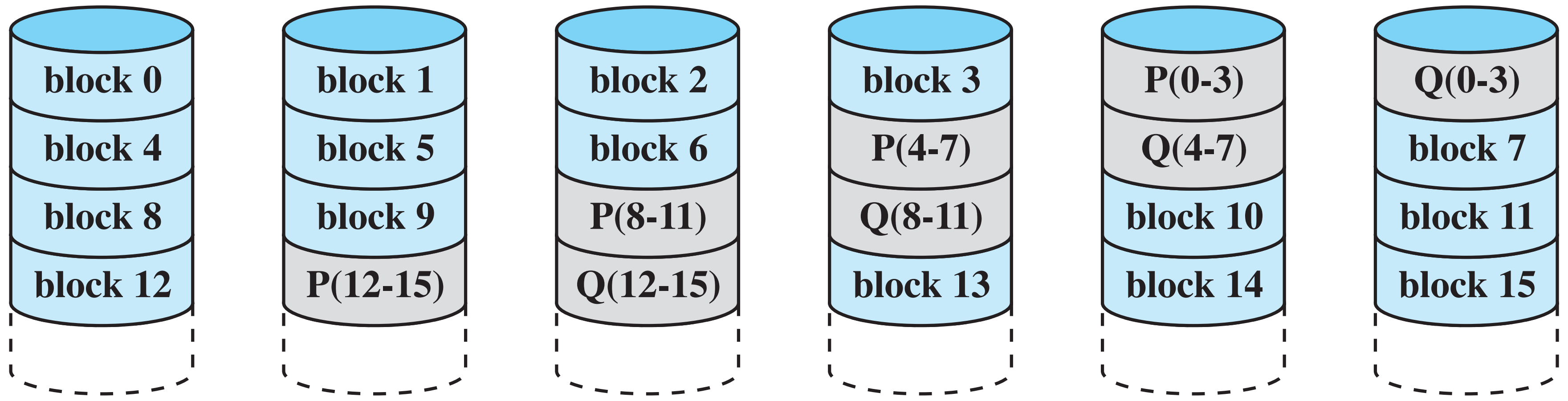
(d) RAID 3 (bit-interleaved parity)



(e) RAID 4 (block-level parity)



(f) RAID 5 (block-level distributed parity)



(g) RAID 6 (dual redundancy)

Less redundancy, less overhead, varying reliability…

File Allocation

When considering allocating space for a file…

- Is the maximum space required for the file allocated at once?
  - Preallocation vs. Dynamic Allocation
  - *Is it reasonable to assume we know how big a file can grow to be?*

- When allocating units of space, should the units be contiguous **(“portions")**? And how big should those portions be?
  - Locality → better **performance**
  - Fewer portions → **smaller data structures** (e.g., tables) for managing allocations
  - Fixed-sized blocks → **simplifies** allocation space & allocation policies
  - **Variable-size** / **small fixed-size** portions → **minimizes waste**

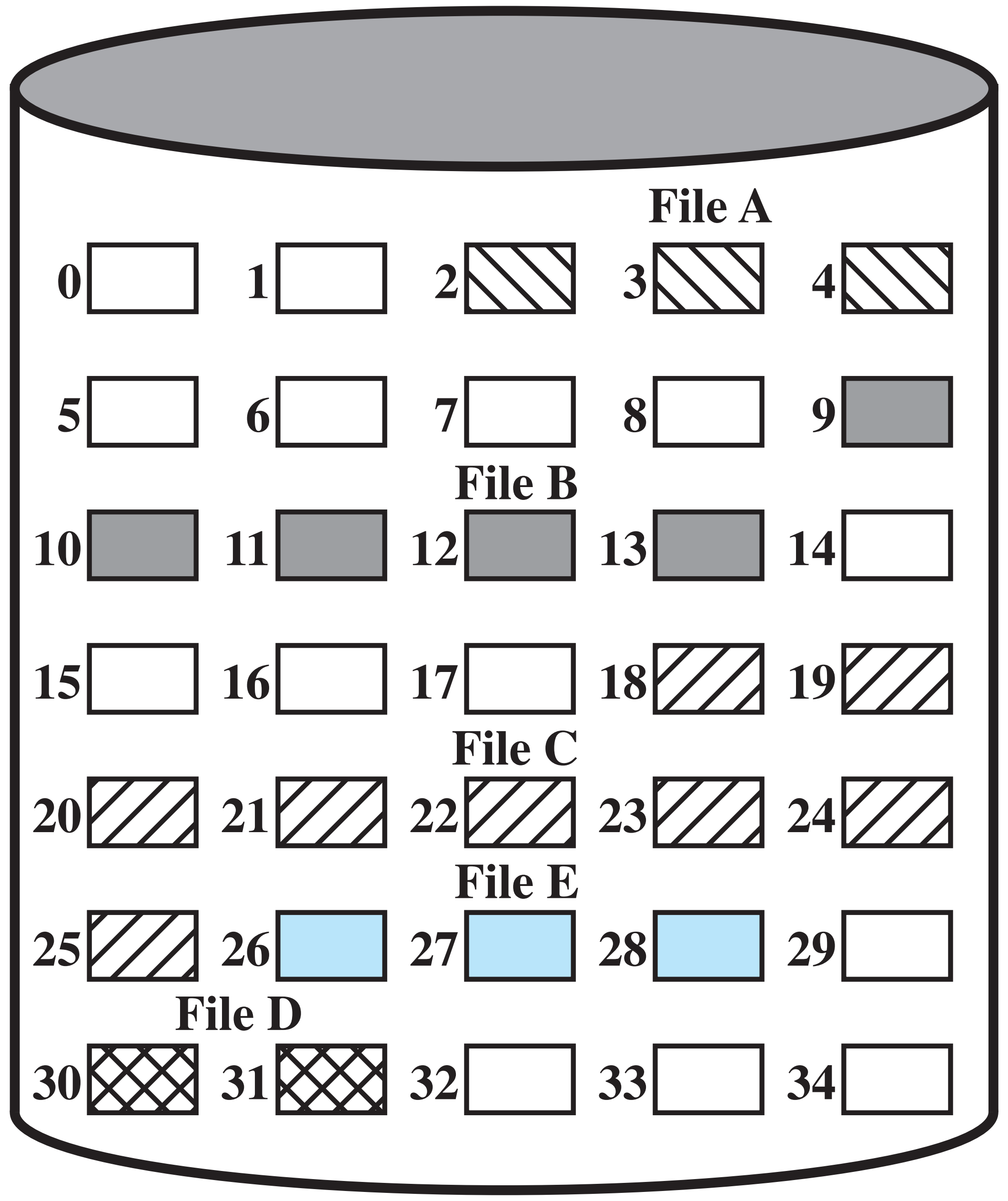
Recall fragmentation issues from virtual memory
→ *Strategies: first-fit, best-fit, next-fit, etc.*

- What sort of data structure(s) should be used to keep track of allocations?
  - Ex. File Allocation Table (FAT)
  - In general, 1 entry holds 1 file unit
    - Ex. File unit = entire file (1 contiguous set of blocks)
    - Ex. File unit = a fixed-size piece of the total file (blocks are scattered around)

Example: Contiguous File Allocation

Allocate a single contiguous block to a file at the time of creation.

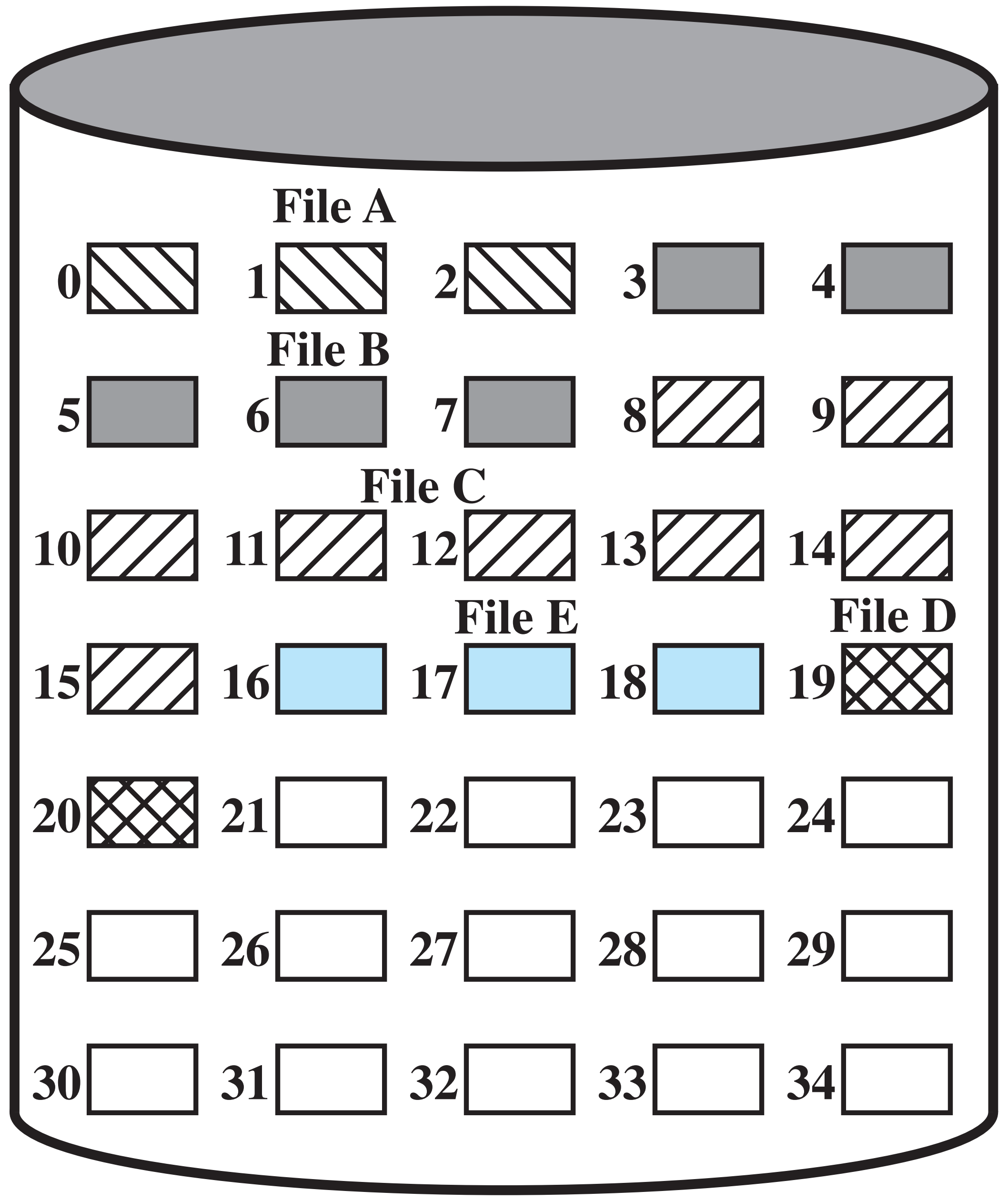
CFA is best from the perspective of a single sequential file!
…not so much for the system as a whole…



File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Before Compaction



File Allocation Table

File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

After Compaction

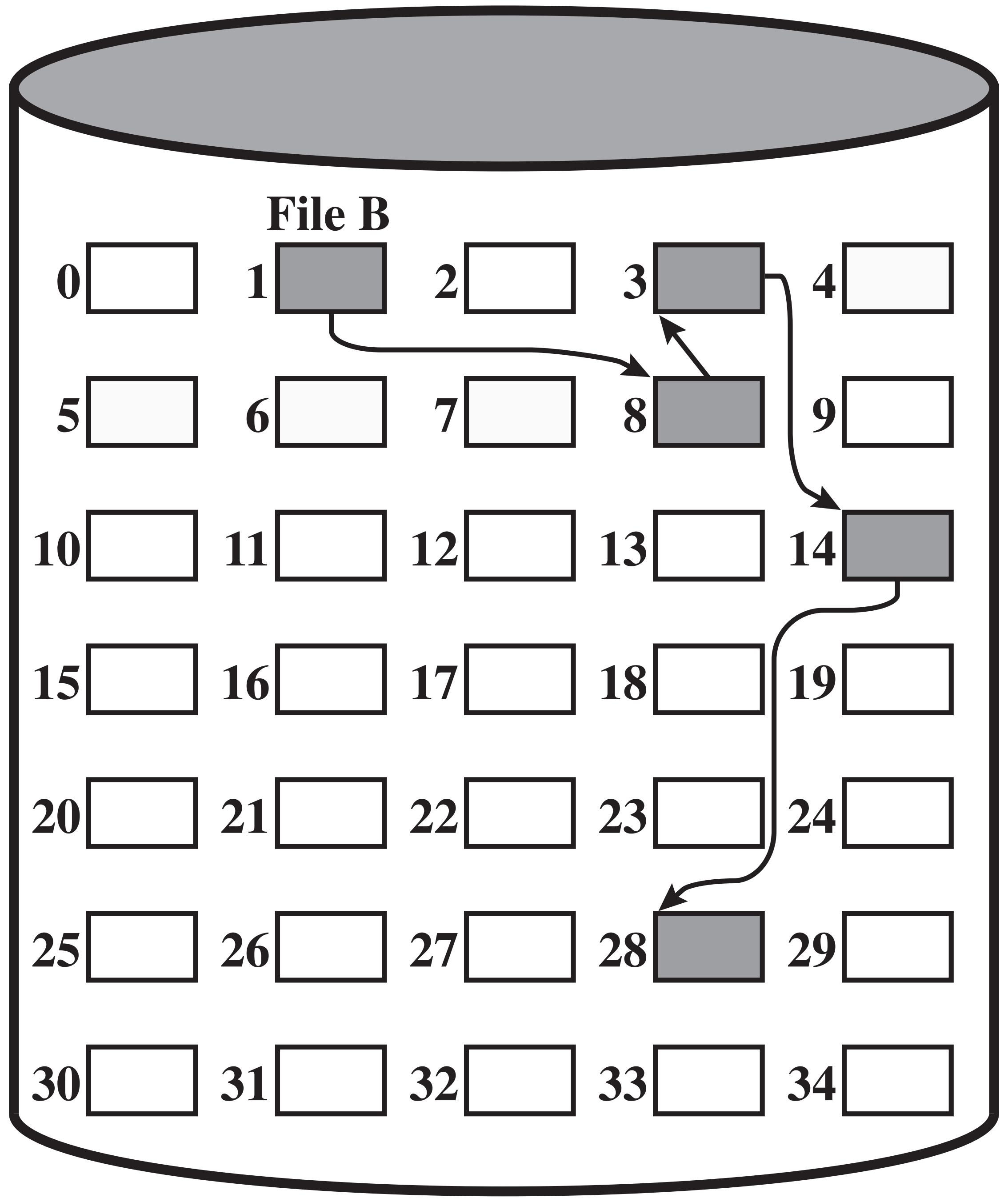
Example: Chained Allocation

Allocate individual, fixed-size blocks until enough space is allocated for the file.

Recall: each block contains a pointer to the next.

CA is best for managing space for sequential files!

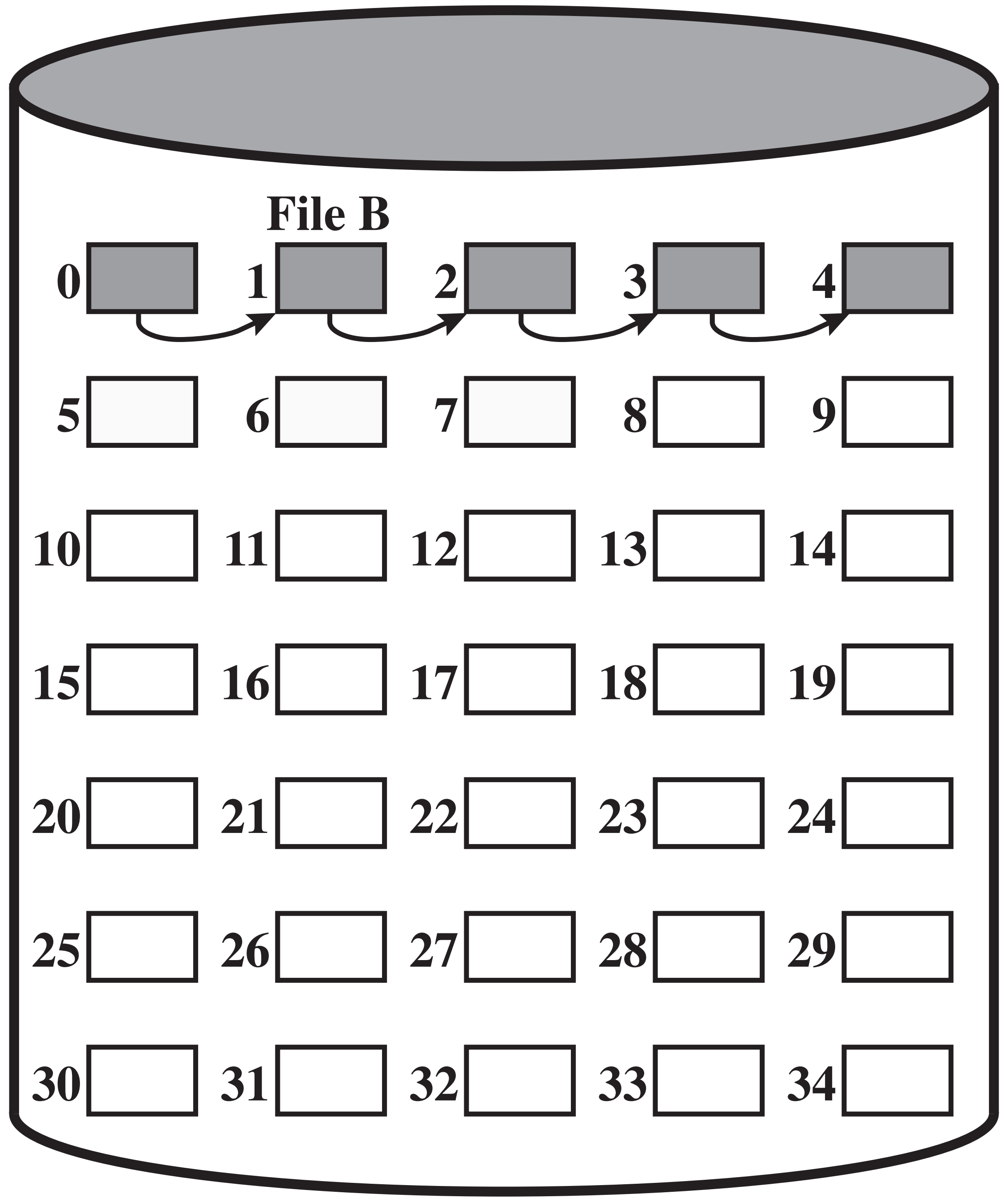
…no utilization of locality though…



File Allocation Table

File Name	Start Block	Length
• • •	• • •	• • •
File B	1	5
• • •	• • •	• • •

Before Periodic Consolidation



File Allocation Table

File Name	Start Block	Length
• • •	• • •	• • •
File B	0	5
• • •	• • •	• • •

After Periodic Consolidation

Example: Indexed Allocation

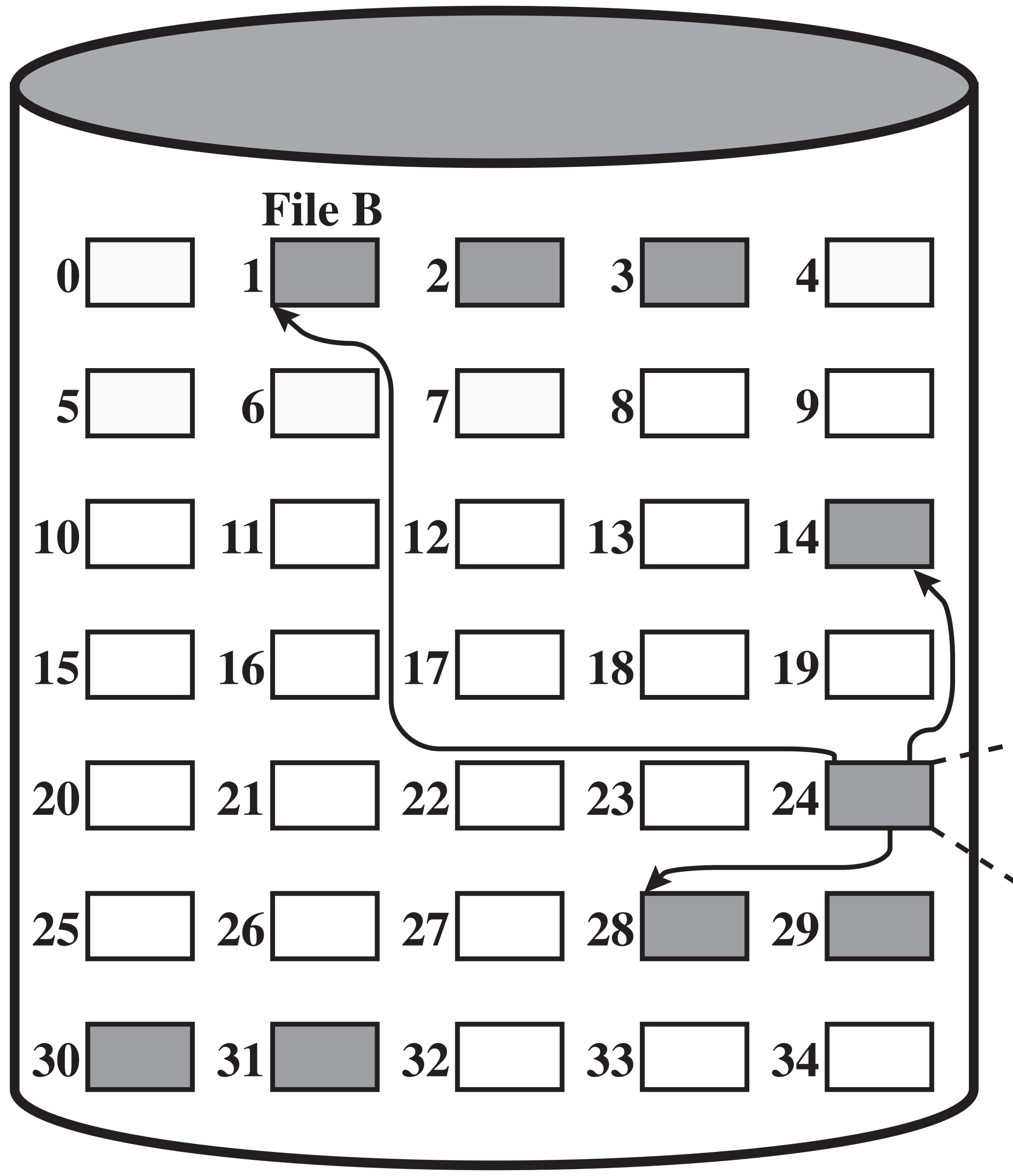
Allocate individual, fixed-size blocks until enough space is allocated for the file.

Maintain separate one-level index for each file in a separate block;

FAT points to block w/ the file's index;

Index has one entry per portion allocated to the file;

Addresses most problems from CFA and CA; still need to periodically run consultation.

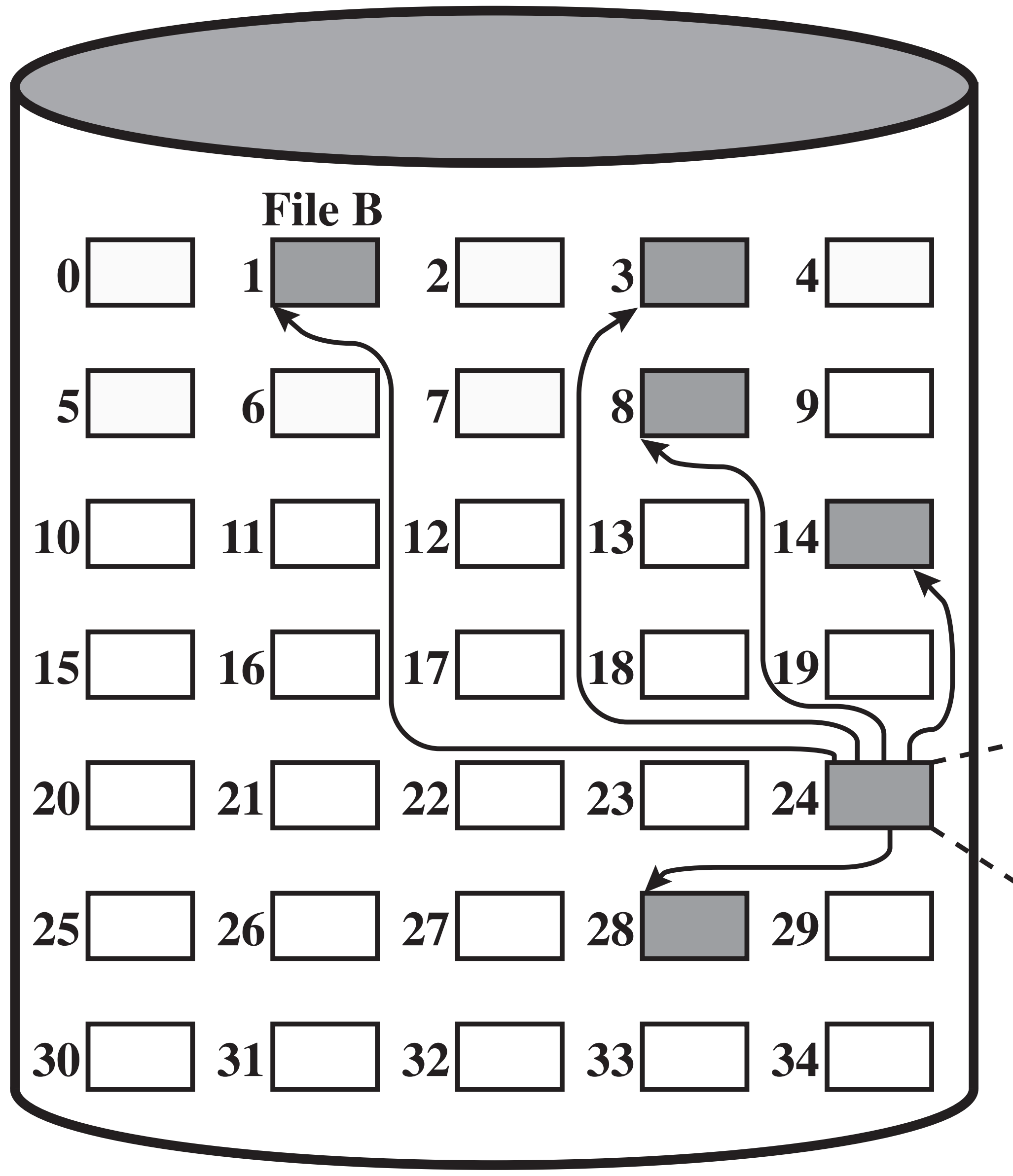


File Allocation Table

File Name	Index Block
• • •	• • •
File B	24
• • •	• • •

Start Block	Length
1	3
28	4
14	1

IA / Block Portions



File Allocation Table

File Name	Index Block
• • •	• • •
File B	24
• • •	• • •

1
8
3
14
28

IA w/ Variable-Length Portions

Extras

Things alluded to in class but not fully covered

Example: Free Space Management

- We have to manage all of the free space as well…
  → *Disk Allocation Table (DAT)*

- Bit Tables
  - Maintain vector of bits; 1 bit per block (0=free, 1=used)
  - Easy to find a free block
  - Easy to find a contiguous sequence of free blocks
  - Compact representation for free space management

…0011110000011100001111111111110110001….

$$\frac{\text{disk size in bytes}}{8 \times \text{file system block size}} = \frac{16\text{GB}}{8 \times 512} = \text{4MB bit table}$$

With disk sizes today, complete bit tables can be large…

→ create a summary table—break into subranges;
calculate # free blocks in subrange, max # of contiguous blocks, etc.

See text for more examples of other strategies and their trade-offs…

Volumes — *A Logical Disk*

- A volume is nothing more than a logical representation for a physical disk. Indeed…
  - a volume can be a subset of a disk,
  - or represent a combination of disks,
  - Or even a subset/combination of another volume(s)
- Analogous to the relationship between physical memory & virtual memory