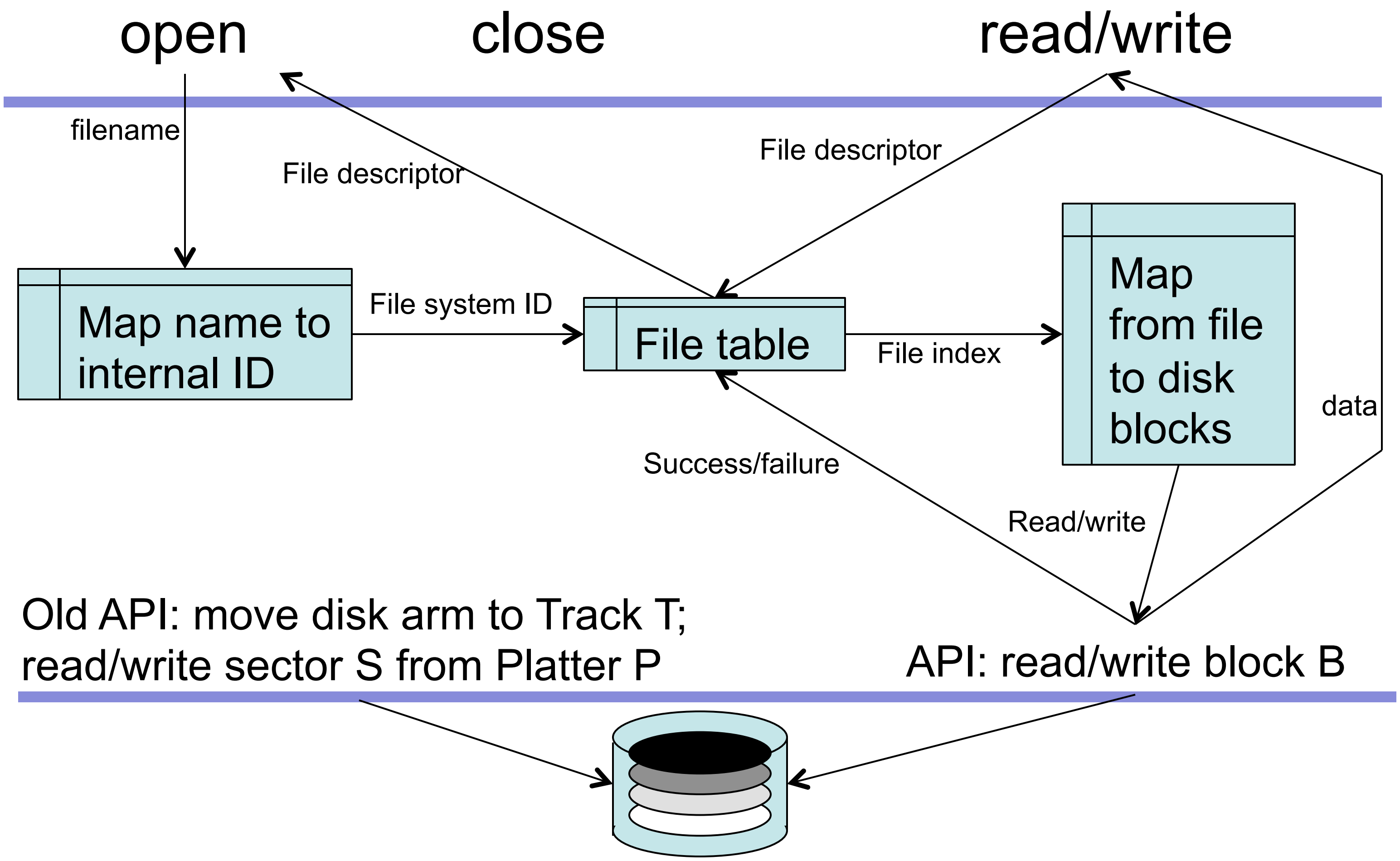
File Systems: Introduction

- Learning Objective
  - Describe the layers of software between the file system system call API and the disk.
  - Decompose those layers in a collection of independent problems.
  - Derive solutions to the key problems of:
    - File representation
    - Naming & Name Spaces
    - Disk Allocation
    - Recovery
- Topics:
  - From "open/close/read/write" to spinning media.
  - File representation
  - Naming
  - Allocation

From Syscall API to Disk



open

close

read/write

filename

File descriptor

File descriptor

Map name to internal ID

File system ID

File table

File index

Map from file to disk blocks

data

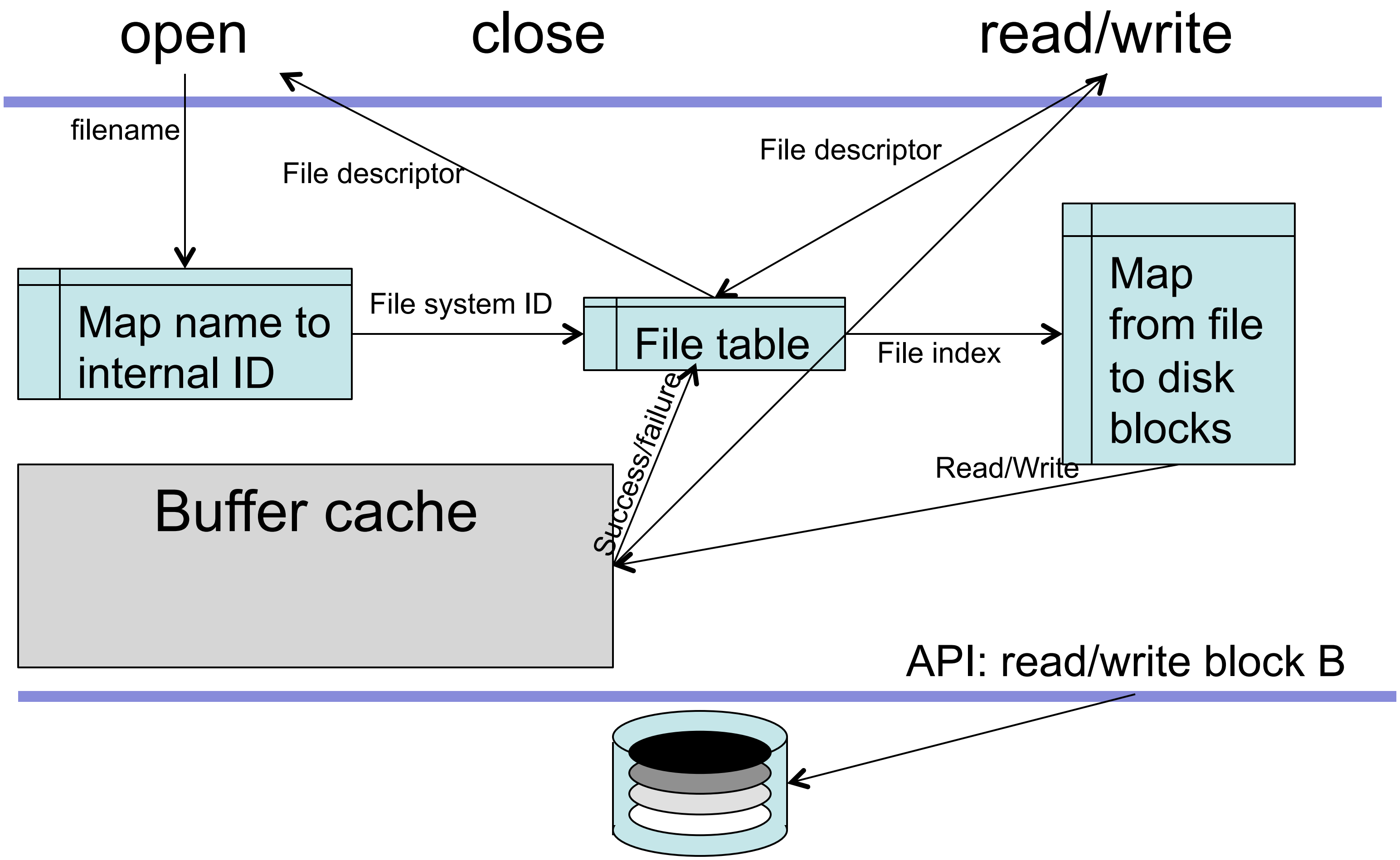
Success/failure

Read/write

Old API: move disk arm to Track T; read/write sector S from Platter P

API: read/write block B

From Syscall API to Disk



Components of a File System

- Directory: maps names to internal IDs
- File table: keeps track of file state
- File index: maps from a file to a collection of disk blocks
- Buffer cache: keeps copies of recently used blocks in memory.
- What kinds of design parameters are likely to be important?
  - Transfer sizes: how much do you move to/from disk?
  - Allocation size: in what unit to you allocate disk blocks?
  - Placement: Where do you place files on disk?

Exercise 1: File Representation

- How might you represent a file?
  - Must support sequential and random access to a file.
  - Must be reasonably efficient.
  - Address the following two questions:
    1. In what size pieces will you allocate disk space to files?
    2. What metadata (data that describes the data) do you need?
  - Questions to think about:
    - Where will you store metadata?
    - What is the ratio of metadata to data for your representation?
    - What kind of *internal fragmentation* can your representation support?
    - What are the advantages/disadvantages of the approach you picked?

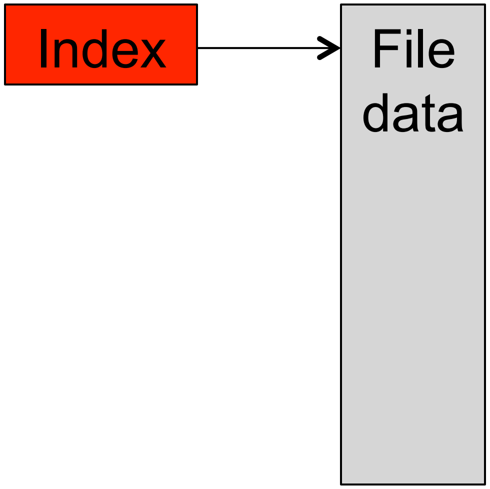
Allocation Units

- Allocation units
  - Fixed sized block
  - A small number of fixed size blocks
  - Variable sizes blocks (called extents)
- Tradeoffs:

File Representation

- Single extent: Metadata is a single address (and perhaps a length)
- A small (fixed) number of extents: Metadata is a few disk addresses (perhaps with length)
- File is a large number of blocks
  - Put blocks together in a linked list: metadata is an address
  - Build a large flat index: Metadata is a large array of one address per block/extent
  - Build a multi-level index (like a multi level page table)

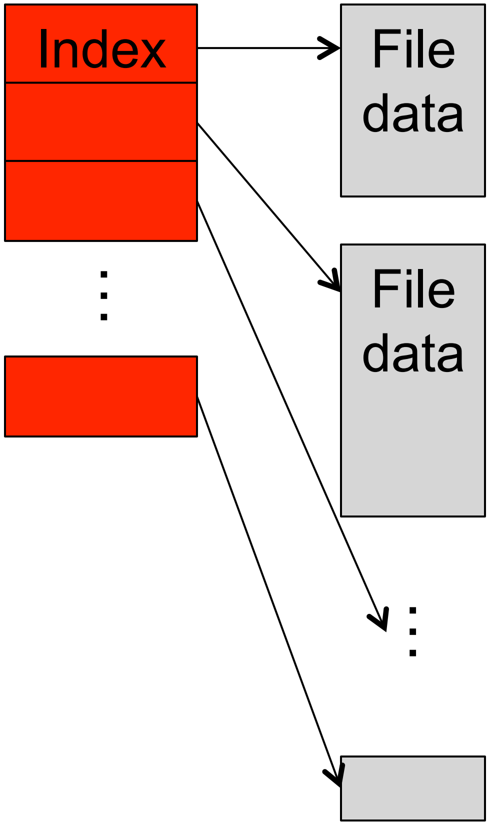
File Representation: Single Extent



- Pros:

- Cons:

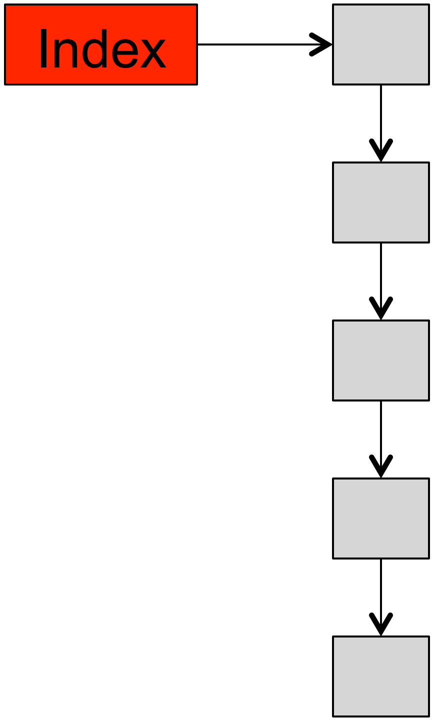
File Representation: A Few Extents



- Pros:

- Cons:

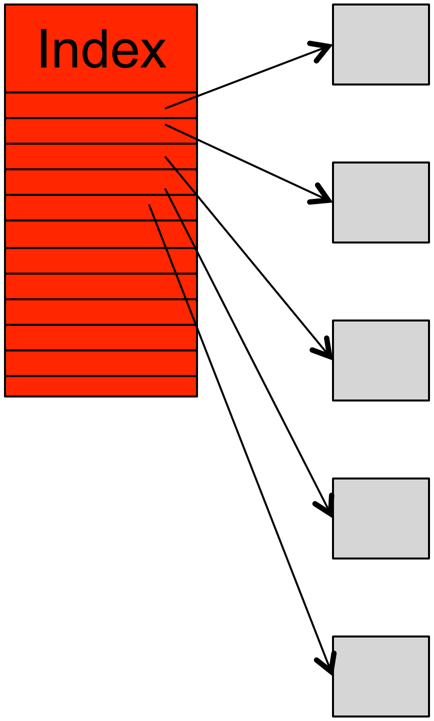
File Representation: Linked Blocks



- Pros:

- Cons:

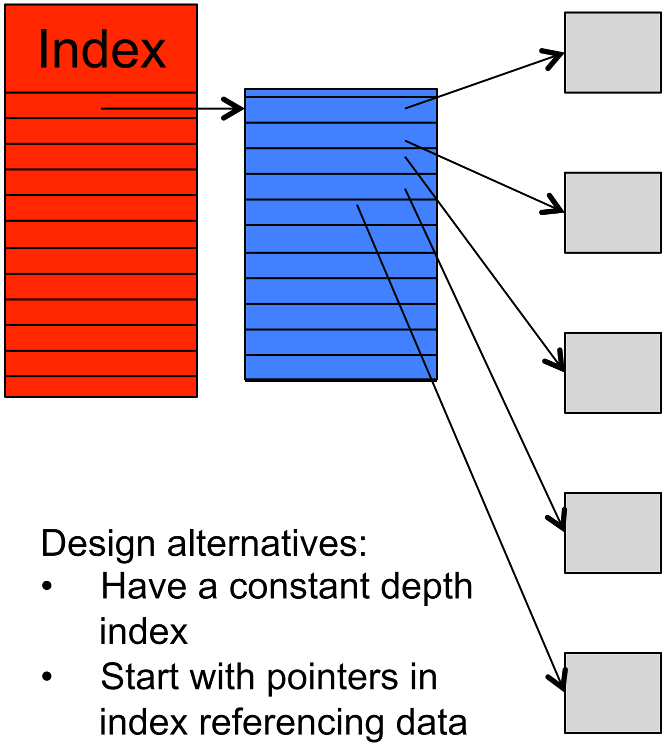
File Representation: Flat Index



- Pros:

- Cons:

File Representation: Multi-level Index



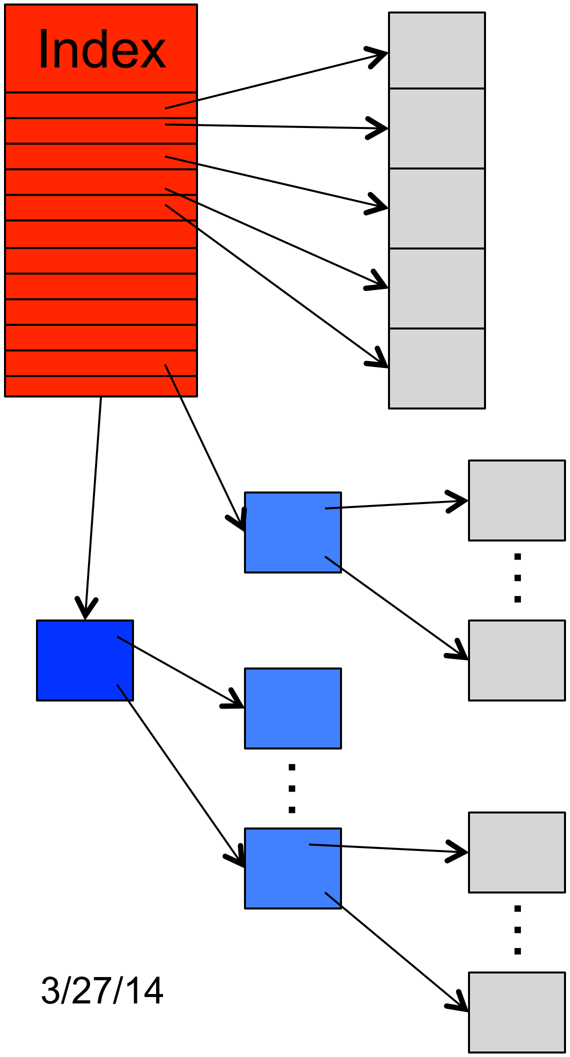
- Pros:

- Cons:

Design alternatives:

- Have a constant depth index
- Start with pointers in index referencing data (direct pointers) When that fills, add first level of indirection and copy pointers, repeat

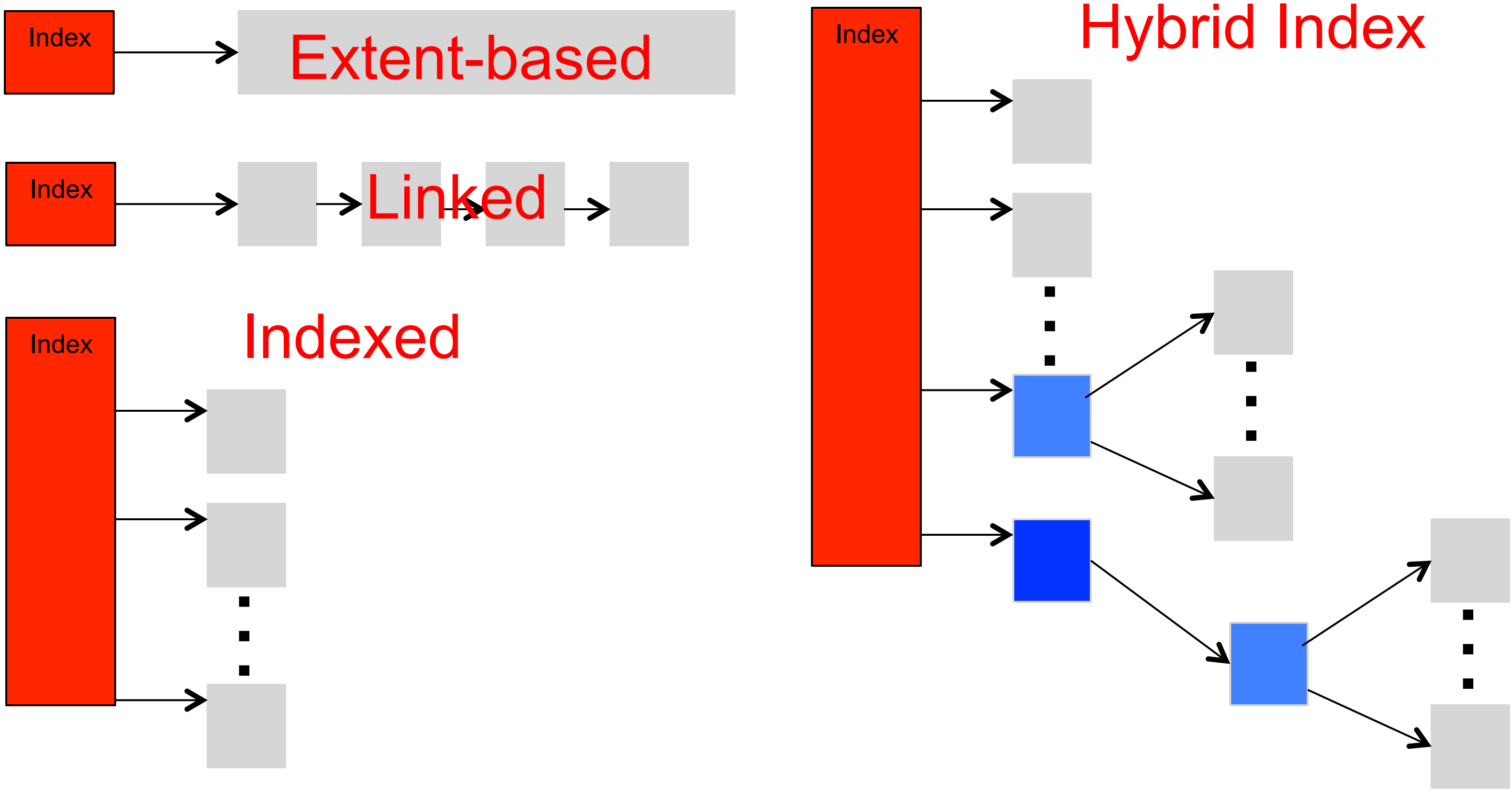
File Representation: Hybrid Index



- Pros:

- Cons:

File Structure Summary



Exercise 2: Free Space Management

- Assume you allocate in fixed size blocks:
    - How do you keep track of free space?
    - How do you select which blocks to allocate to a particular file?

- Assume that you allocate variable size extents:
    - How do you select the extent size?
    - How do you manage free space?
    - Where do you allocate extents?