File Systems I

COMS W4118

References: Operating Systems Concepts (9e), Linux Kernel Development, previous W4118s

Typical file access patterns

- Sequential Access
  - Data read or written in order
    - Most common access pattern
      - E.g., copy files, compiler read and write files,
  - Can be made very fast (peak transfer rate from disk)

- Random Access
  - Randomly address any block
    - E.g., update records in a database file
  - Difficult to make fast (seek time and rotational delay)

Disk management

- Need to track where file data is on disk
  - How should we map logical sector # to surface #, track #, and sector #?
    - Order disk sectors to minimize seek time for sequential access

- Need to track where file metadata is on disk

- Need to track free versus allocated areas of disk
  - E.g., block allocation bitmap (Unix)
    - Array of bits, one per block
    - Usually keep entire bitmap in memory

Allocation strategies

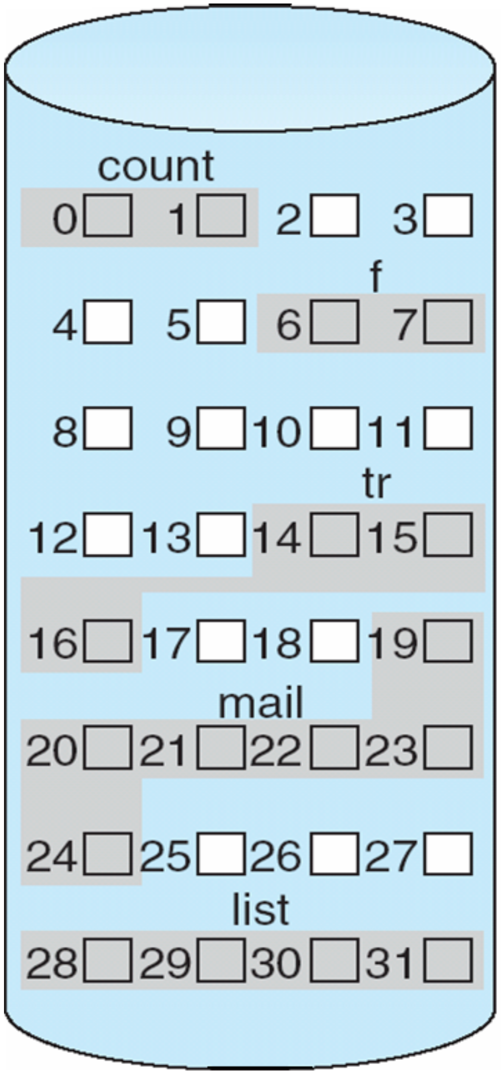
- Various approaches (similar to memory allocation)
    - Contiguous
    - Extent-based
    - Linked
    - FAT tables
    - Indexed
    - Multi-Level Indexed

- Key metrics
    - Fragmentation (internal & external)?
    - Grow file over time after initial creation?
    - Fast to find data for sequential and random access?
    - Easy to implement?
    - Storage overhead?

Contiguous allocation

- Allocate files like continuous memory allocation (base & limit)
  - User specifies length, file system allocates space all at once
  - Can find disk space by examining bitmap
  - Metadata: contains starting location and size of file

Contiguous allocation example



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Pros and cons

- Pros
  - Easy to implement
  - Low storage overhead (two variables to specify disk area for file)
  - Fast sequential access since data stored in continuous blocks
  - Fast to compute data location for random addresses. Just an array index
- Cons
  - Large external fragmentation
  - Difficult to grow file

Extent-based allocation

- Multiple contiguous regions per file (like segmentation)
  - Each region is an extent
  - Metadata: contains small array of entries designating extents
    - Each entry: start and size of extent

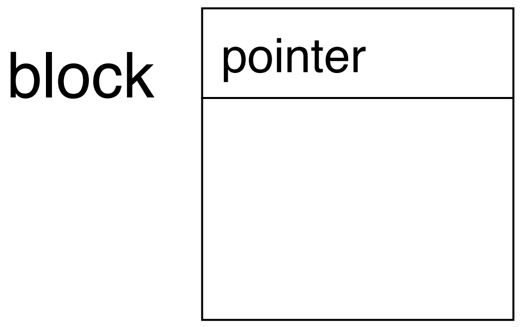
Pros and cons

- Pros
  - Easy to implement
  - Low storage overhead (a few entries to specify file blocks)
  - File can grow overtime (until run out of extents)
  - Fast sequential access
  - Simple to calculate random addresses

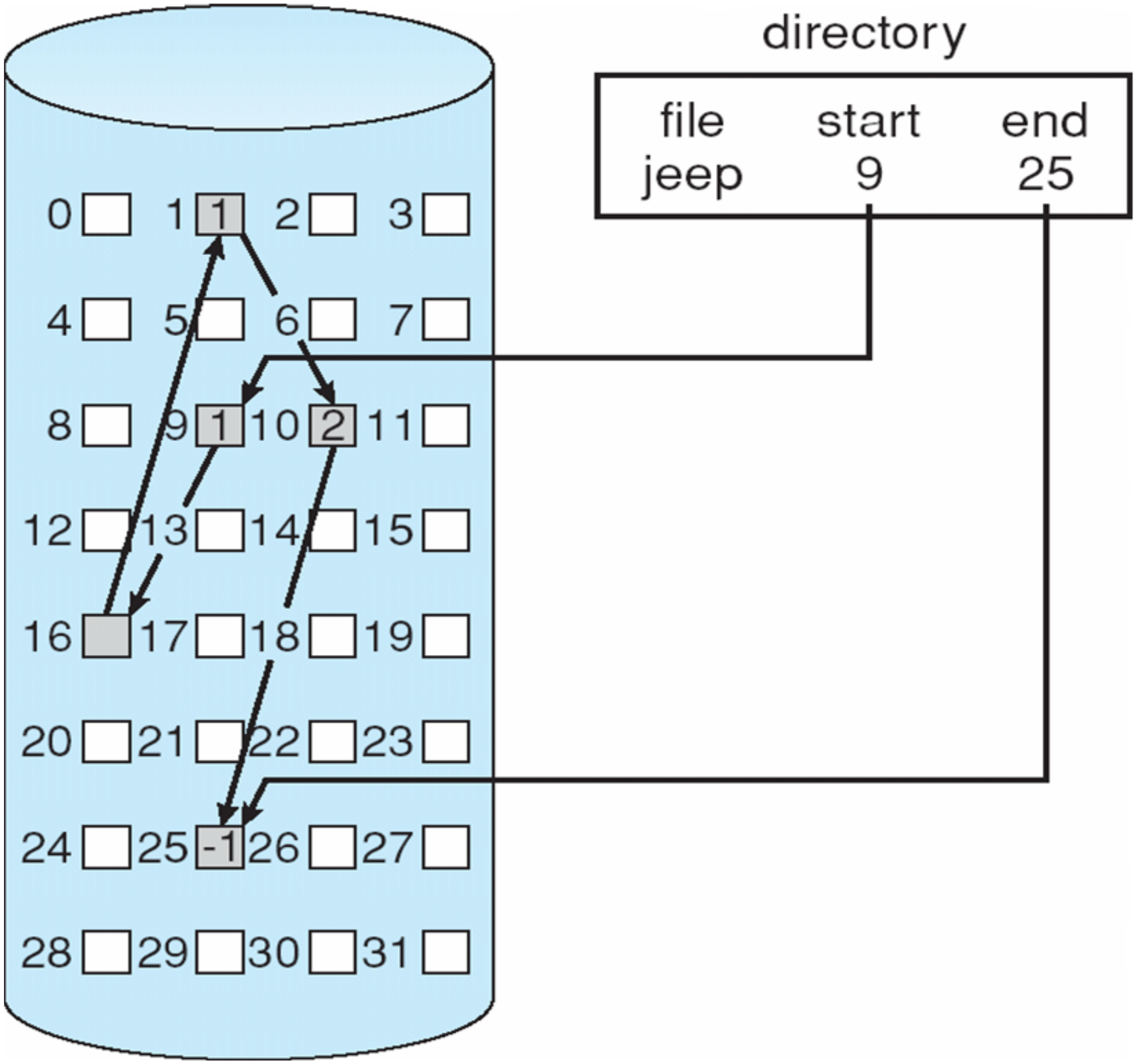
- Cons
  - Help with external fragmentation, but still a problem

Linked allocation

- All blocks (fixed-size) of a file on linked list
  - Each block has a pointer to next
  - Metadata: pointer to the first block



Linked allocation example



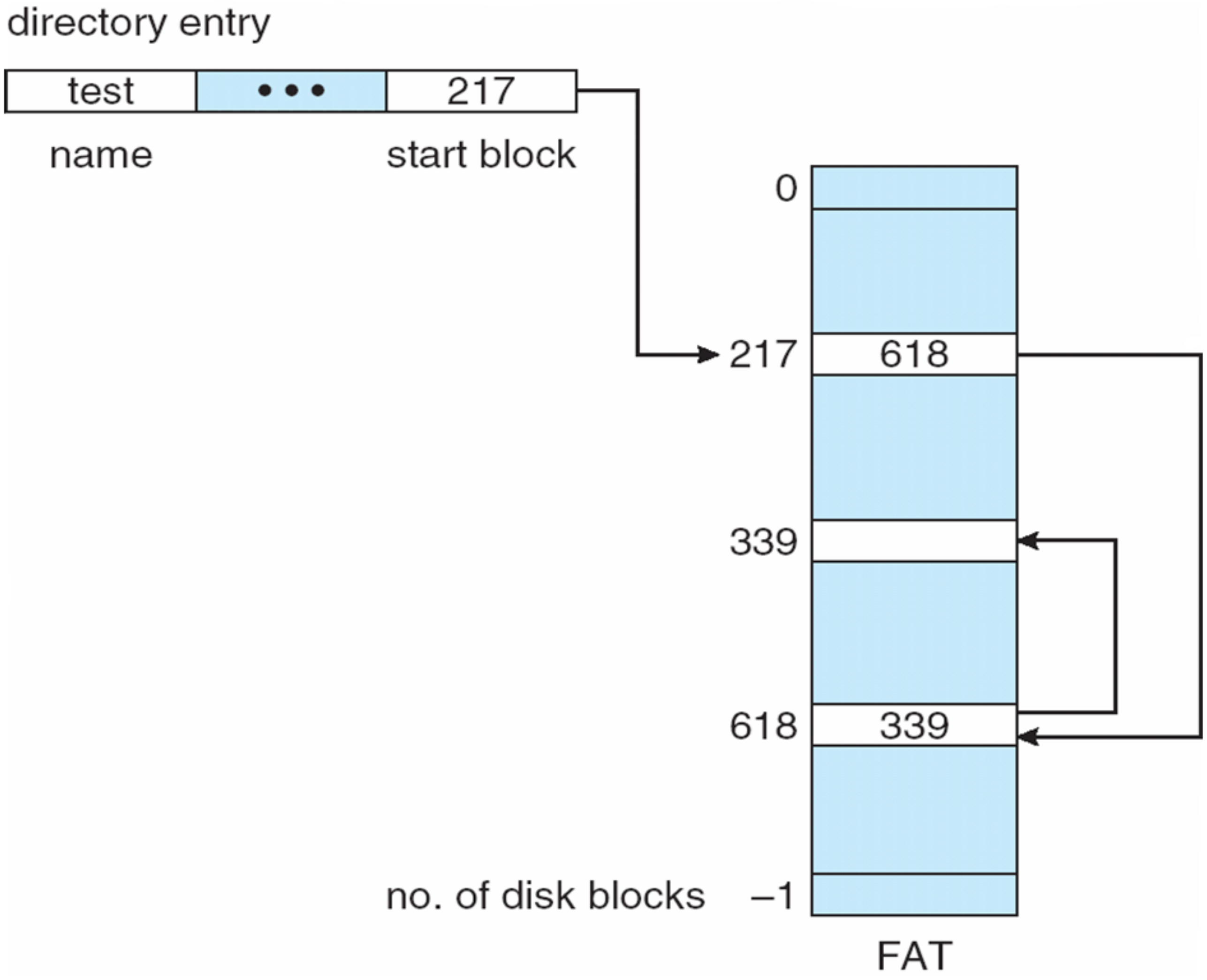
Pros and cons

- Pros
  - No external fragmentation
  - Files can be easily grown with no limit
  - Also easy to implement, though awkward to spare space for disk pointer per block
- Cons
  - Large storage overhead (one pointer per block)
  - Potentially slow sequential access
  - Difficult to compute random addresses

Variation: FAT table

- Store linked-list pointers outside block in File-Allocation Table
  - One entry for each block
  - Linked-list of entries for each file
- Used in MSDOS and Windows operating systems

FAT example

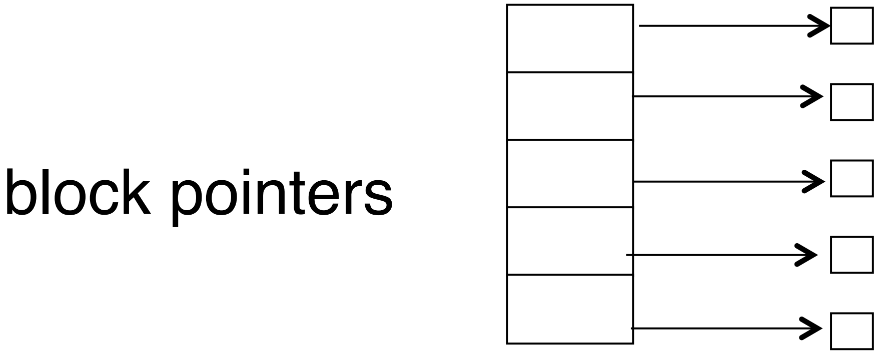

directory entry
test
• • •
217
name
start block
0
217
618
339
618
339
no. of disk blocks –1
FAT

Pros and cons

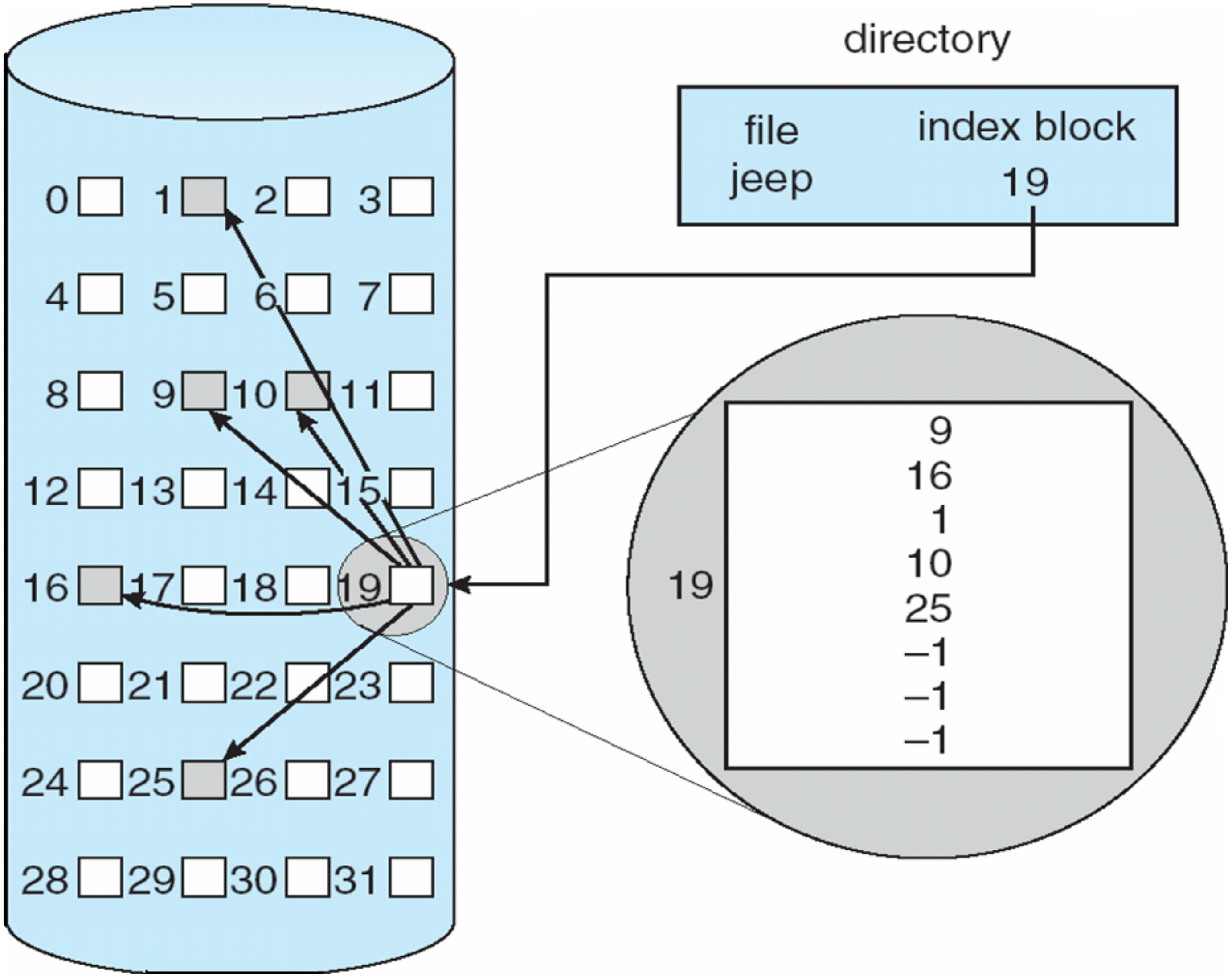
- Pros
  - Fast random access. Only search cached FAT
- Cons
  - Large storage overhead for FAT table
  - Potentially slow sequential access

Indexed allocation

- File has array of pointers (index) to block
  - Allocate block pointers contiguously in metadata
    - Must set max length when file created
    - Allocate pointers at creation, allocate blocks on demand
    - Cons: fixed size, same overhead as linked allocation
  - Maintain multiple lists of block pointers
    - Last entry points to next block of pointers
    - Cons: may need to access a large number of pointer blocks



Indexed allocation example

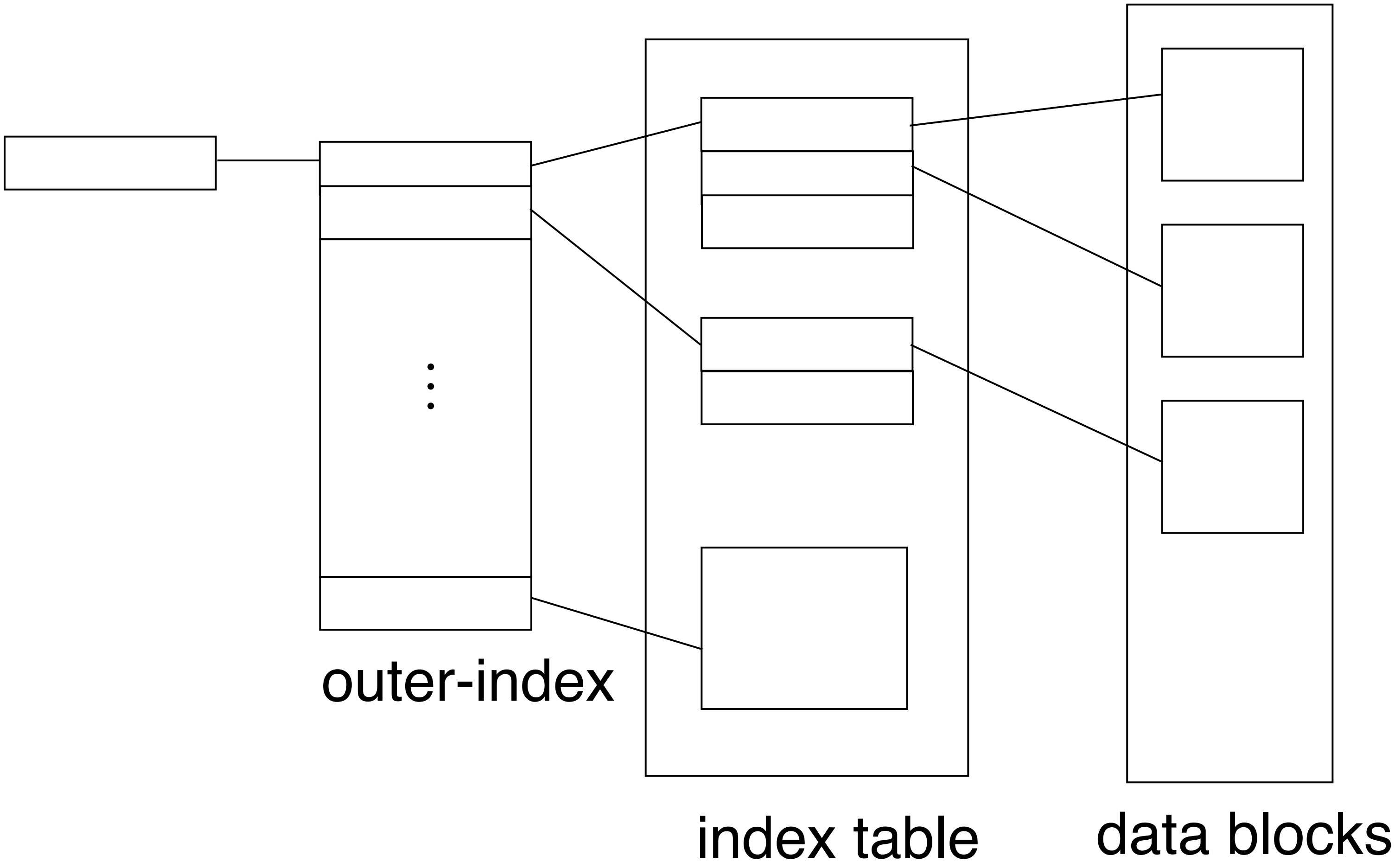


Pros and cons

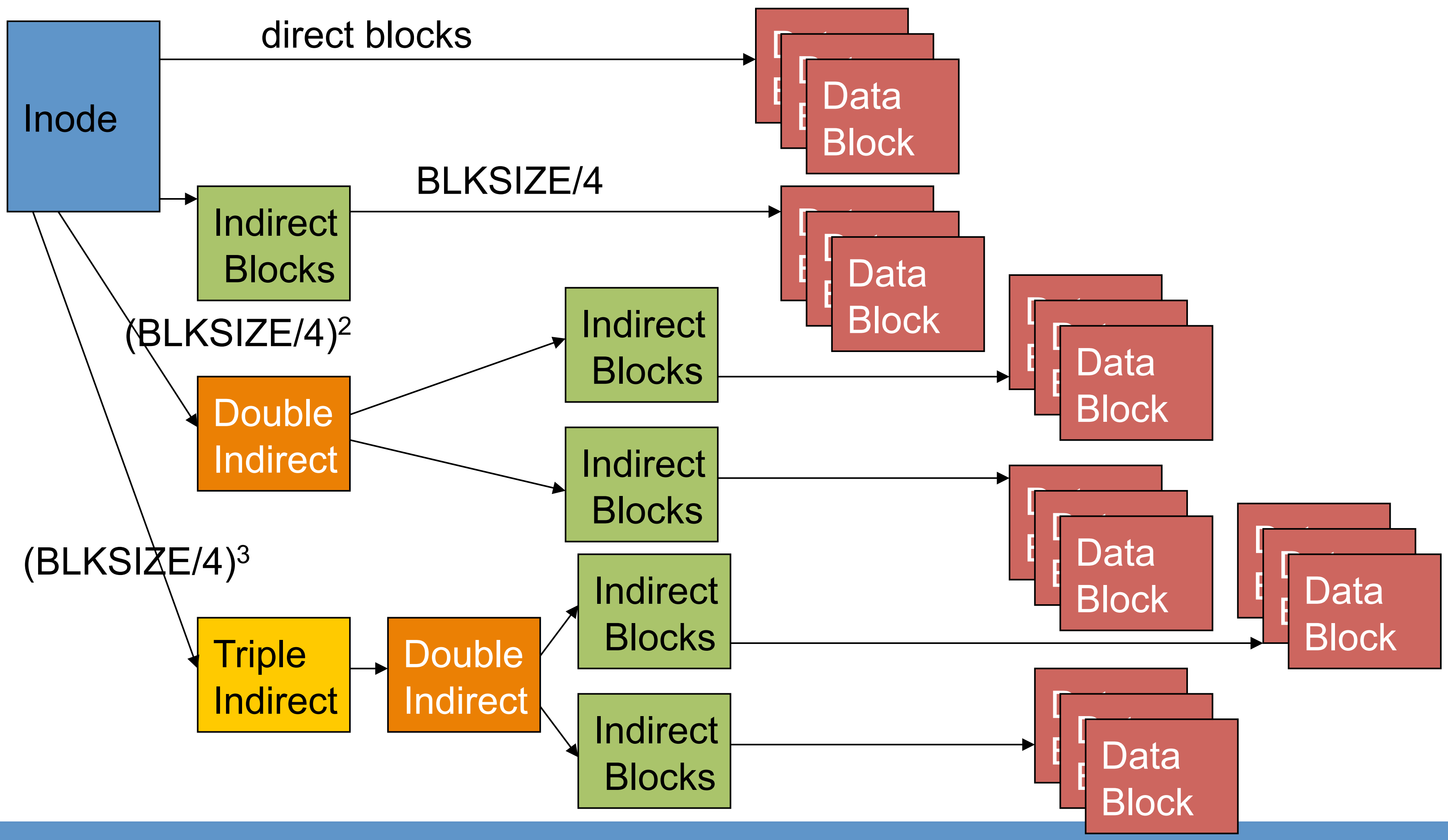
- Pros
  - Easy to implement
  - No external fragmentation
  - Files can be easily grown with the limit of the array size
  - Fast random access. Use index
- Cons
  - Large storage overhead for the index
  - Sequential access may be slow.
    - Must allocate contiguous block for fast access

Multi-level indexed files

- Block index has multiple levels



Multi-level indexed allocation (UNIX FFS, and Linux ext2/ext3)



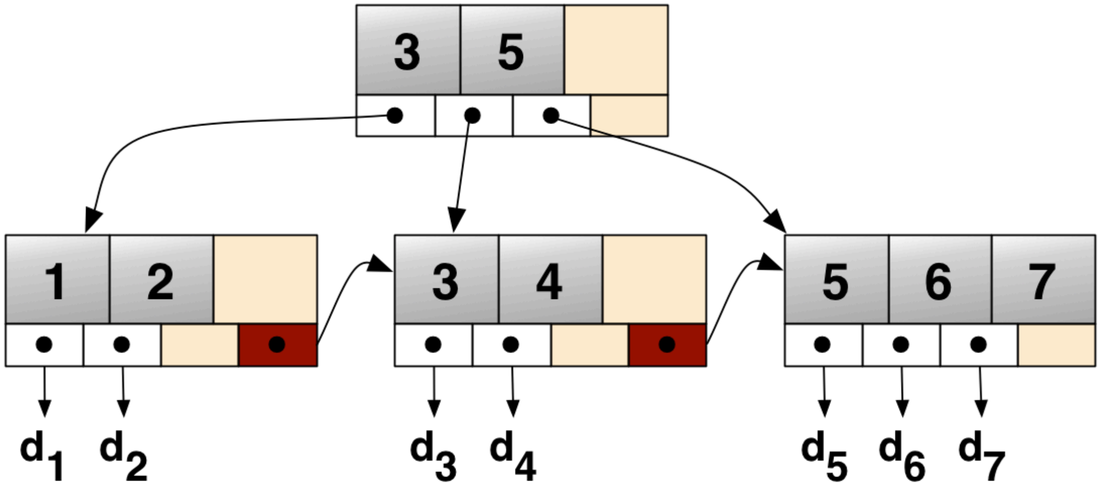
Pros and cons

- Pros
  - No external fragmentation
  - Files can be easily grown with much larger limit compared to one-level index
  - Fast random access. Use index

- Cons
  - Large space overhead (index)
  - Sequential access may be slow.
    - Must allocate contiguous block for fast access
  - Implementation can be complex

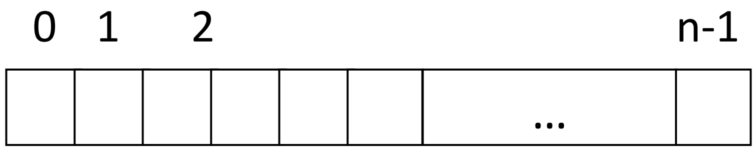
Advanced Data Structures

- Combine Indexes with extents/multiple cluster sizes
- More sophisticated data stuctures
- B+ Trees
  - Used by many high perf filesystems for directories and/or data
  - E.g., XFS, ReiserFS, ext4, MSFT NTFS and ReFS, IBM JFS, brtfs
  - Can support very large files (including sparse files)
  - Can give very good performance (minimize disk seeks to find block)

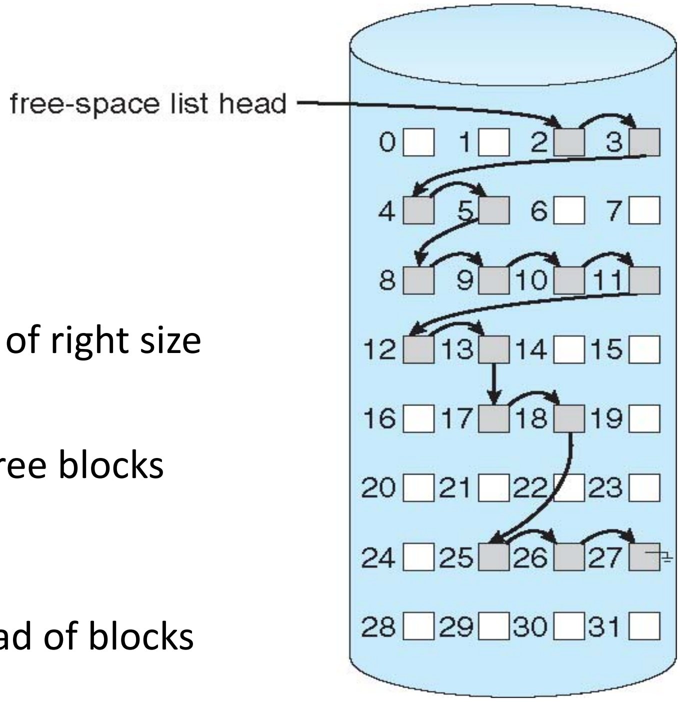


Free Space Management

- File system maintains **free-space list** to track available blocks/clusters
- **Free bitmap** stored in the superblock

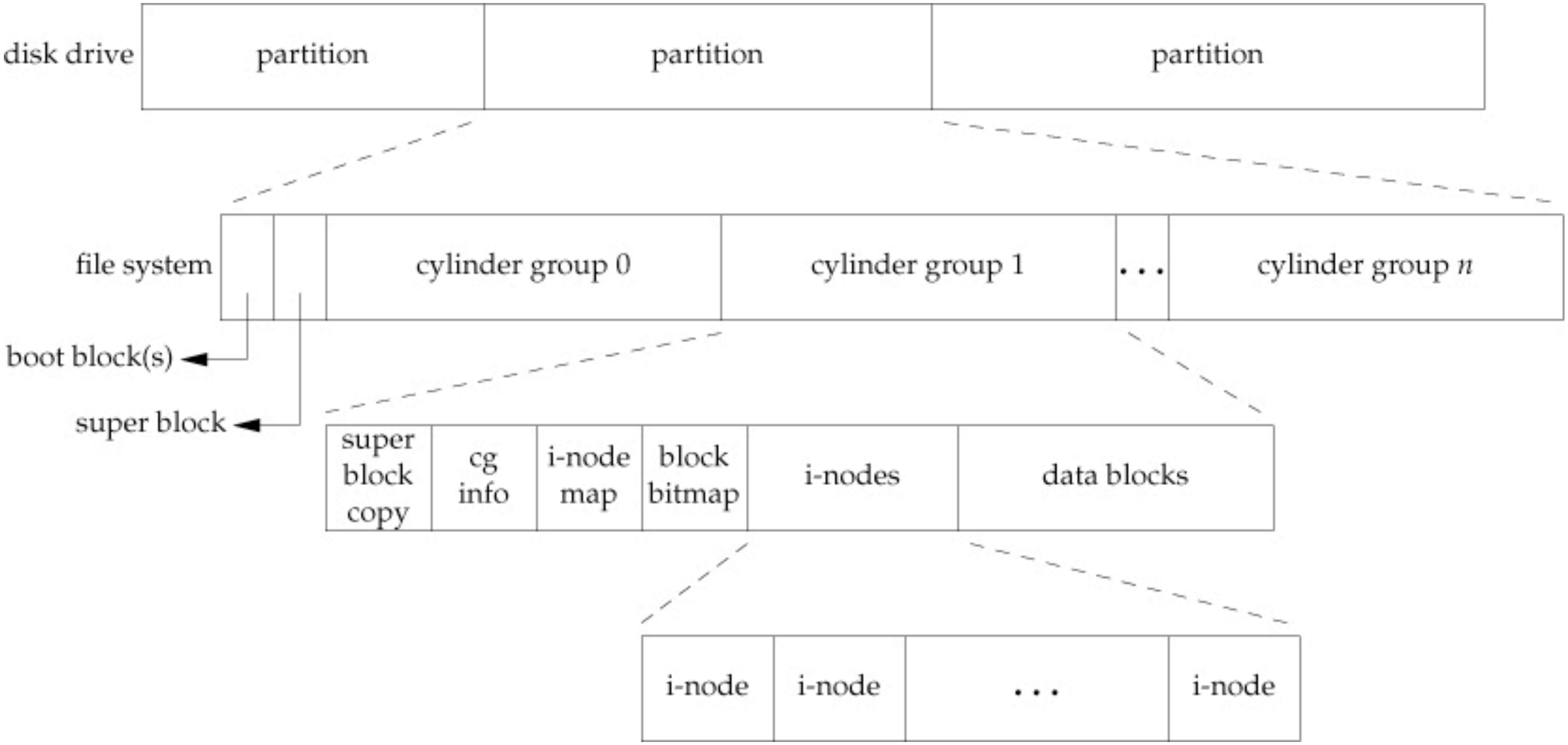


$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{block}[i] \text{ free} \\ 0 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

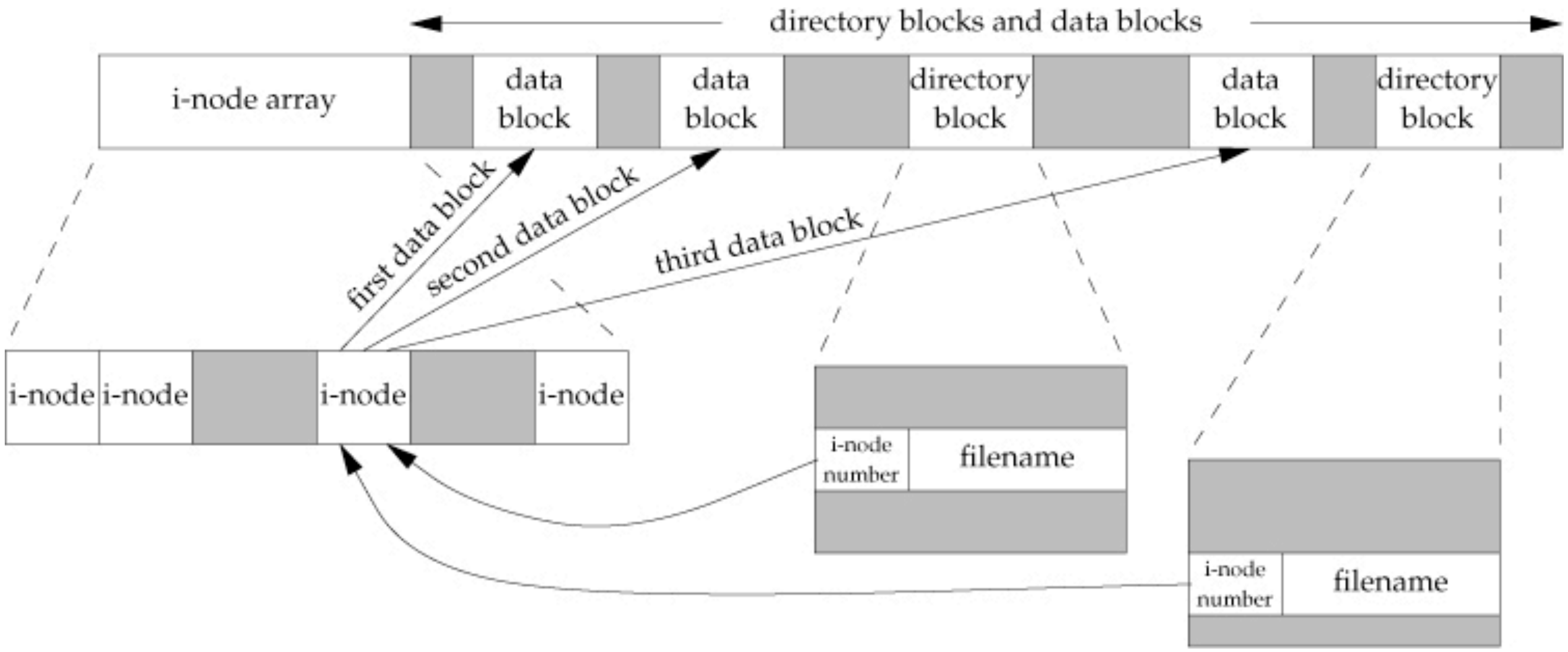


- Linked free list in free blocks
  - Pros: space efficient
  - Cons: requires many disk reads to find free cluster of right size
- Grouping
  - Use a free index-block containing n-1 pointers to free blocks and a pointer to the next free index-block
- Counting
  - Free list of variable sized contiguous clusters instead of blocks
  - Reduces number of free list entries

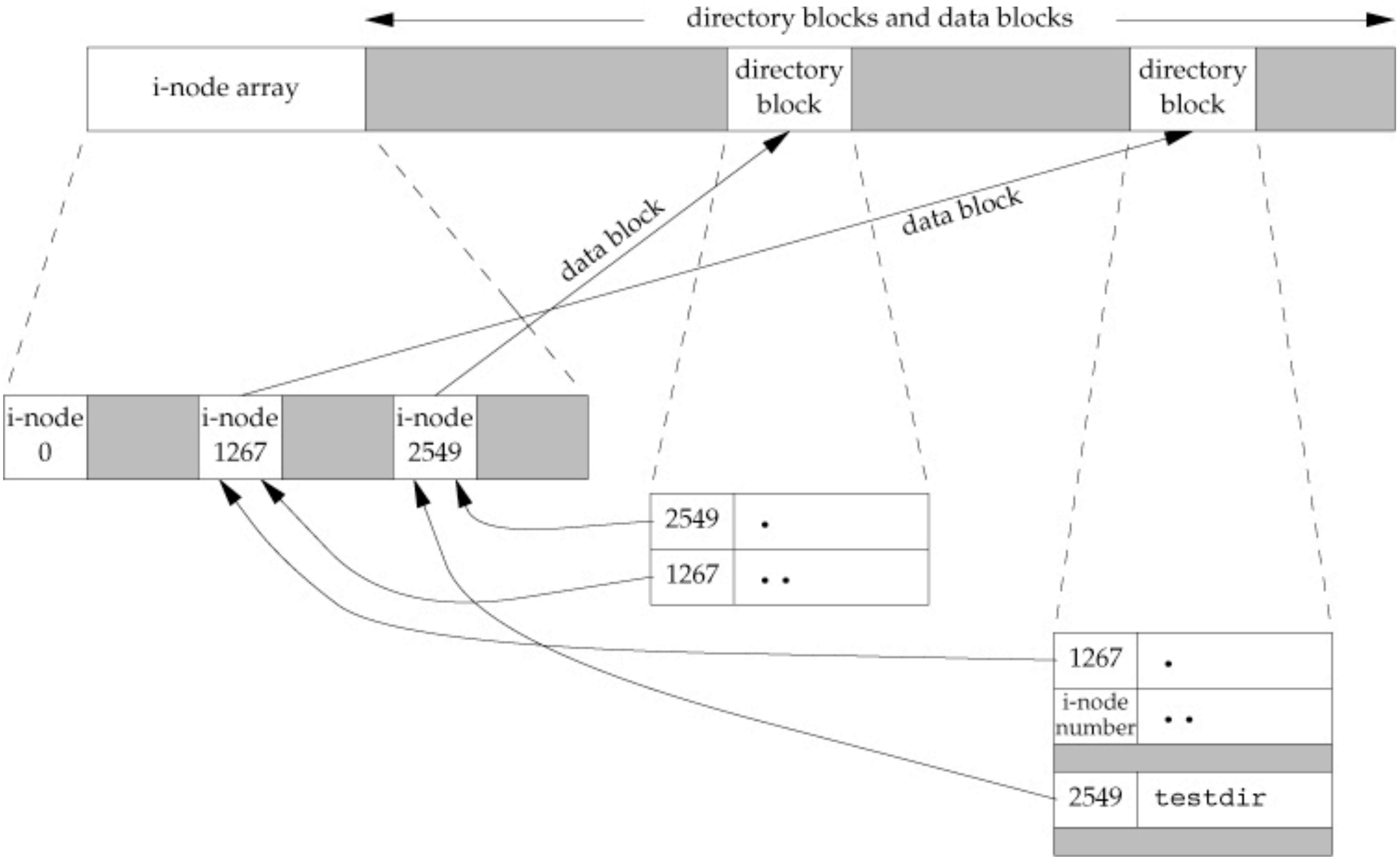
Berkeley Fast File System (FFS) Layout



Inode and data block in cylinder group



After “mkdir testdir”



Hard links v. Symlinks

- Two types of links
  - Symbolic link
    - Special file, designated by bit in meta-data
    - File data is name to another file
  - Hard link
    - Multiple directory entries point to same file
    - All hard-links are equal: no primary
    - Store reference count in file metadata
    - Cannot refer to directories; why?