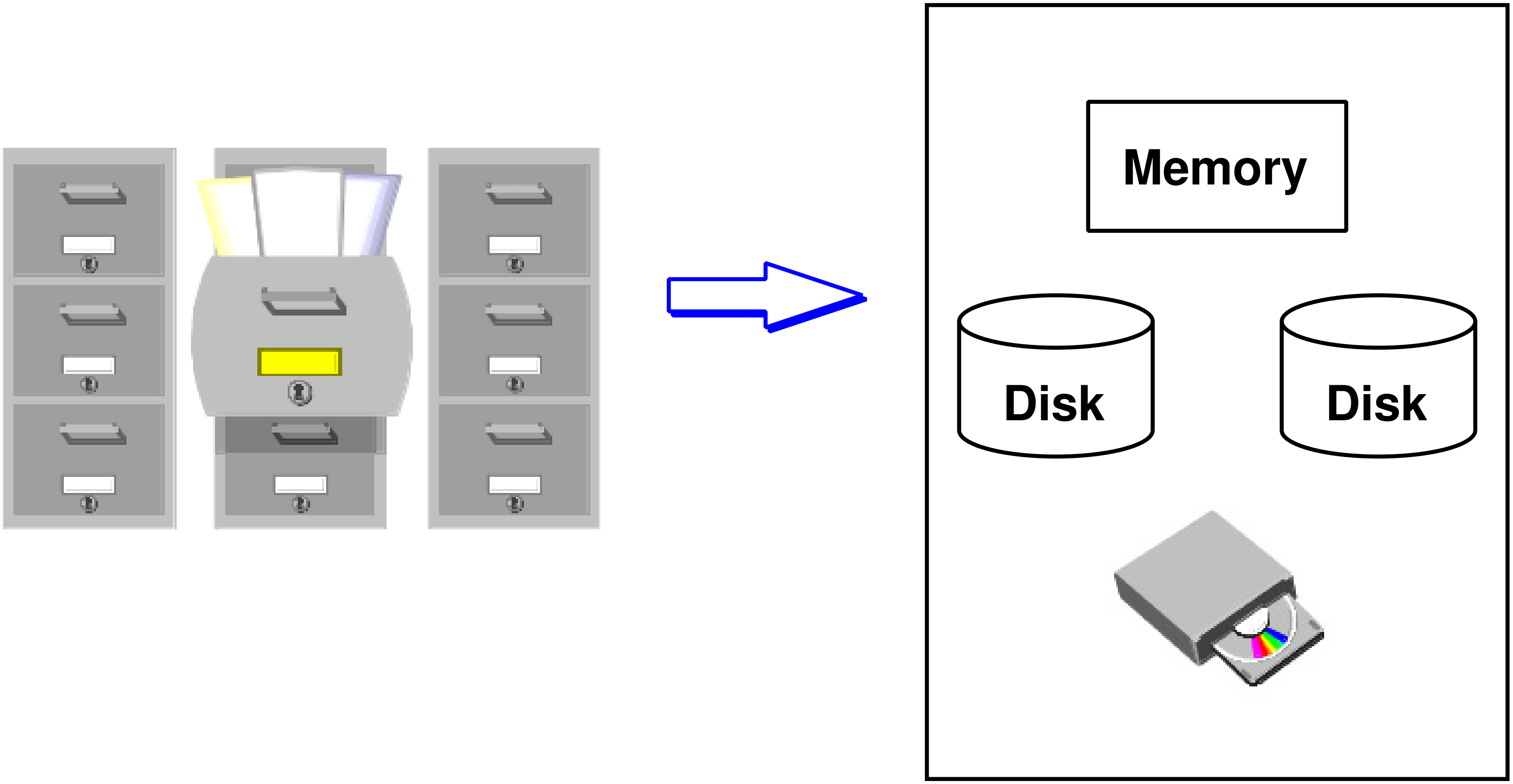
Ch 6: File Systems

Bill Cheng

http://merlot.usc.edu/william/usc/

Files



Requirements

- Permanent storage
  - resides on disk (or alternatives)
  - survives software and hardware crashes
    - (including loss of disk?)
- Quick, easy, and efficient
  - satisfies needs of most applications
    - how do applications use permanent storage?

Applications

- Software development
  - text editors
  - linkers and loaders
  - source-code control
- Document processing
  - editing
  - browsing
- Web stuff
  - serving
  - browsing
- Program execution
  - paging

Needs

- Directories
  - convenient naming
  - fast lookup
- File access
  - sequential is very common!
  - "random access" is relatively rare

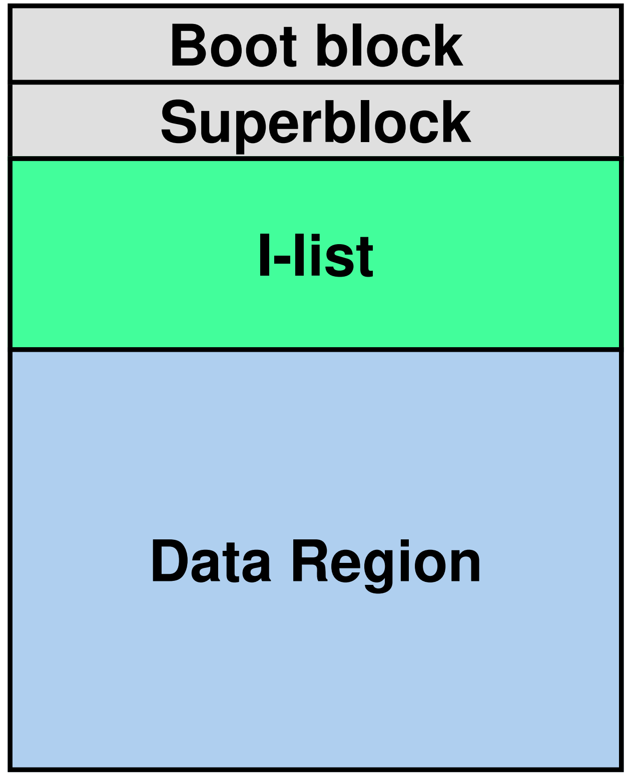
6.1 The Basics of File Systems

- ***UNIX's S5FS***
- **Disk Architecture**
- **Problems with S5FS**
- **Improving Performance**

S5FS

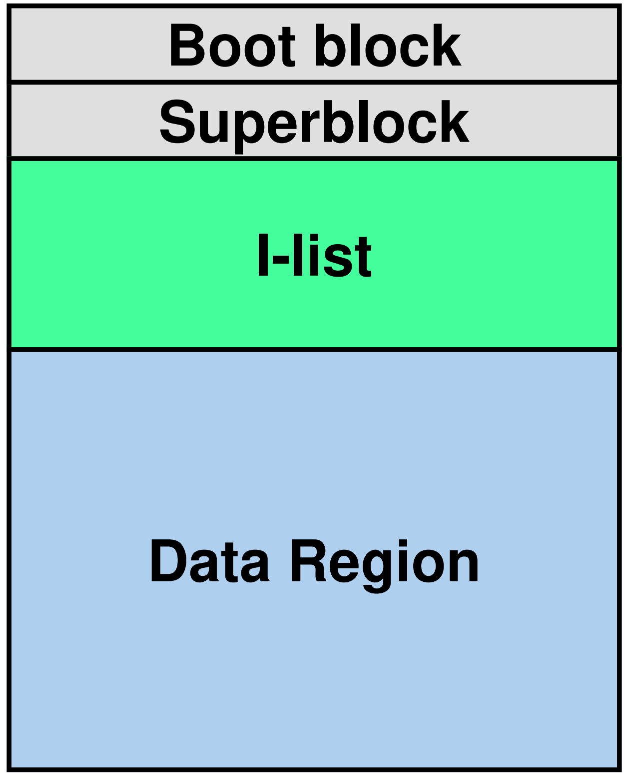
- A simple file system
  - slow
  - not terribly tolerant to crashes
  - reasonably efficient in space
    - no compression
- Concerns
  - on-disk data structures
    - file representation
      - recall that AFS translates an inode number to disk address
    - free space

S5FS Layout



- A disk is simply an array of blocks of 1KB each (old Unix: 512B)
  - a disk block is a *logical* unit (usually multiple of page size)
- A "linear view" (1-D array of blocks) of the disk

S5FS Layout



where are the "free space"?
- inodes can be free or not
- data block can be free or not

- The *superblock*
  - describes the layout of the rest of the file system
  - contains the *head* of the *free list*
- The *i-list* is an *array* of *index nodes (inodes)*
  - each representing a *file*

S5FS: Inode

Device
Inode Number
Mode (File Type)
Link Count
Owner, Group
File Size
Disk Map

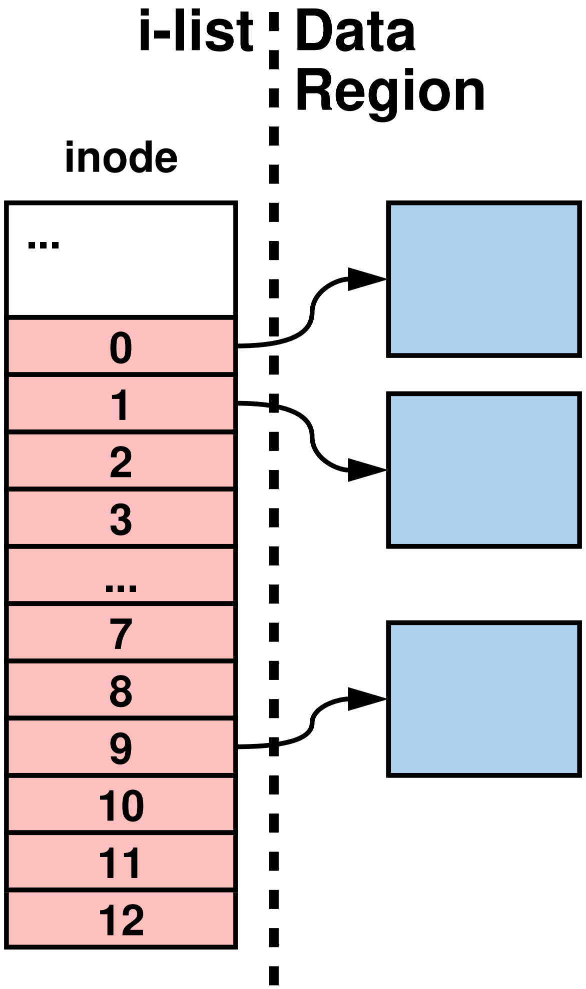
Disk Map

inode
...
0
1
2
3
...
7
8
9
10
11
12

- assuming blocksize = 1KB
  - to make math easier

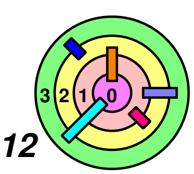
- Disk Map contains 13 disk block pointers
  - a disk block pointer is just a block number
    - pointers 0 through 9 are direct pointers
    - pointer 10 is an indirect pointer
    - pointer 11 is an double-indirect pointer
    - pointer 12 is an triple-indirect pointer

Disk Map

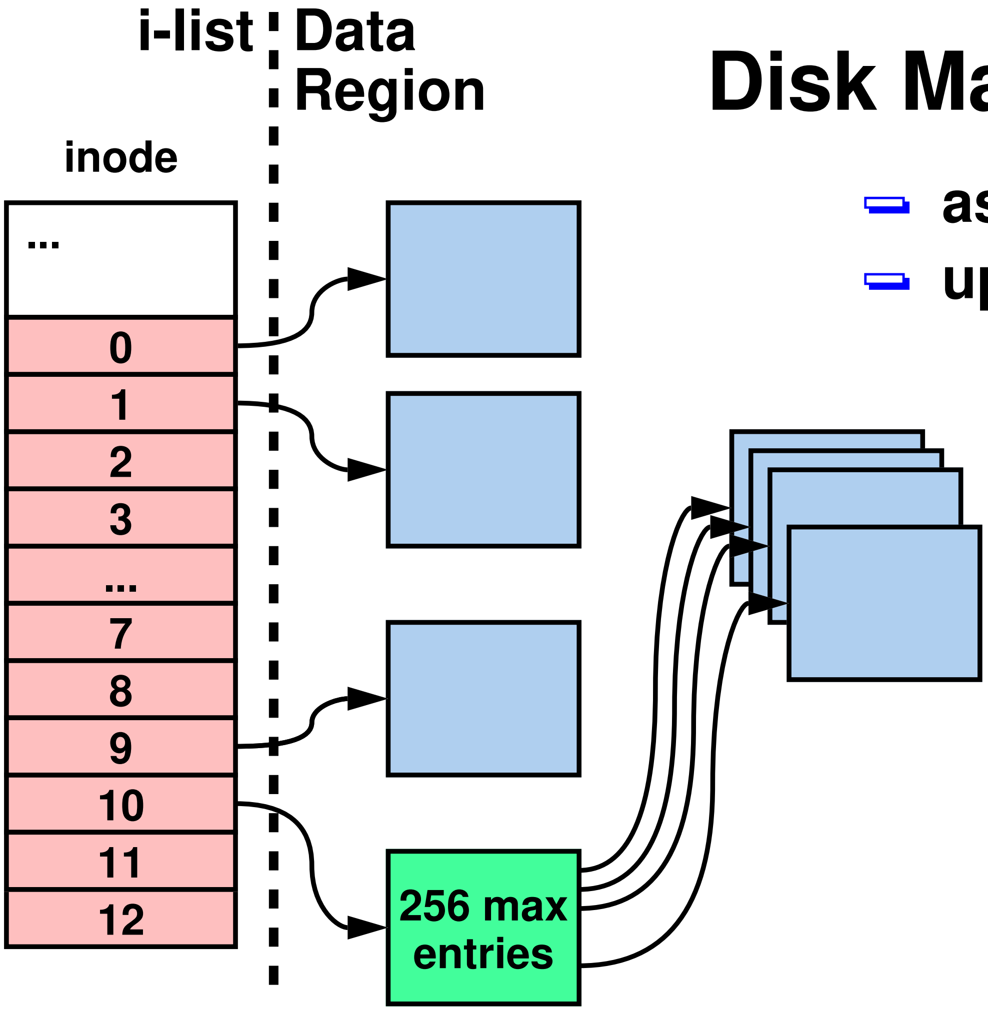


- assuming blocksize = 1KB
- up to 10KB

- Disk Map contains 13 disk block pointers
  - a disk block pointer is just a block number
- Some blocks in the data region contains data



Disk Map

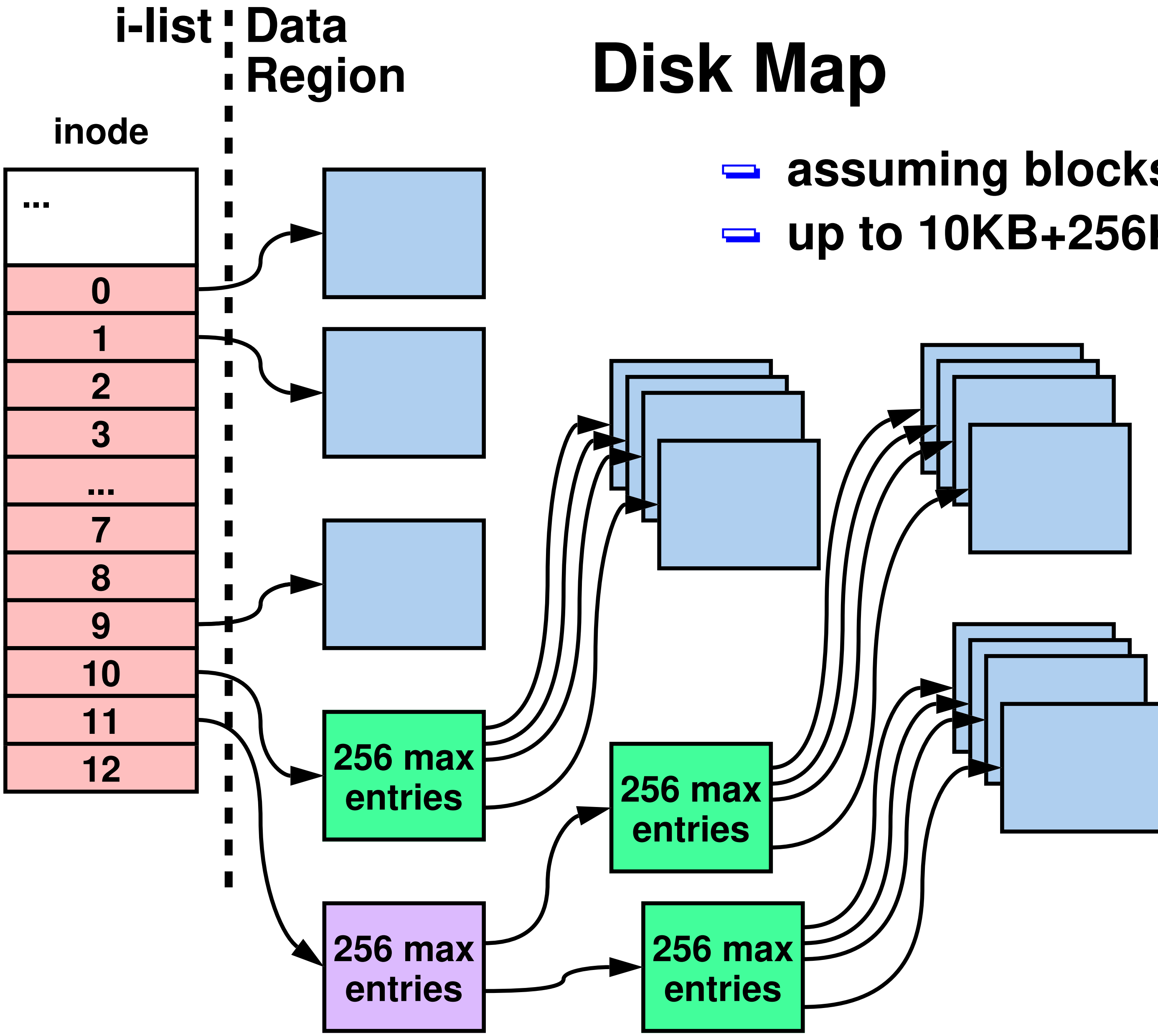


- assuming blocksize = 1KB
- up to 10KB+256KB

- Disk Map contains 13 disk block pointers
  - a disk block pointer is just a block number
- Some blocks in the data region contains data
  - some blocks in the data region contains data structures

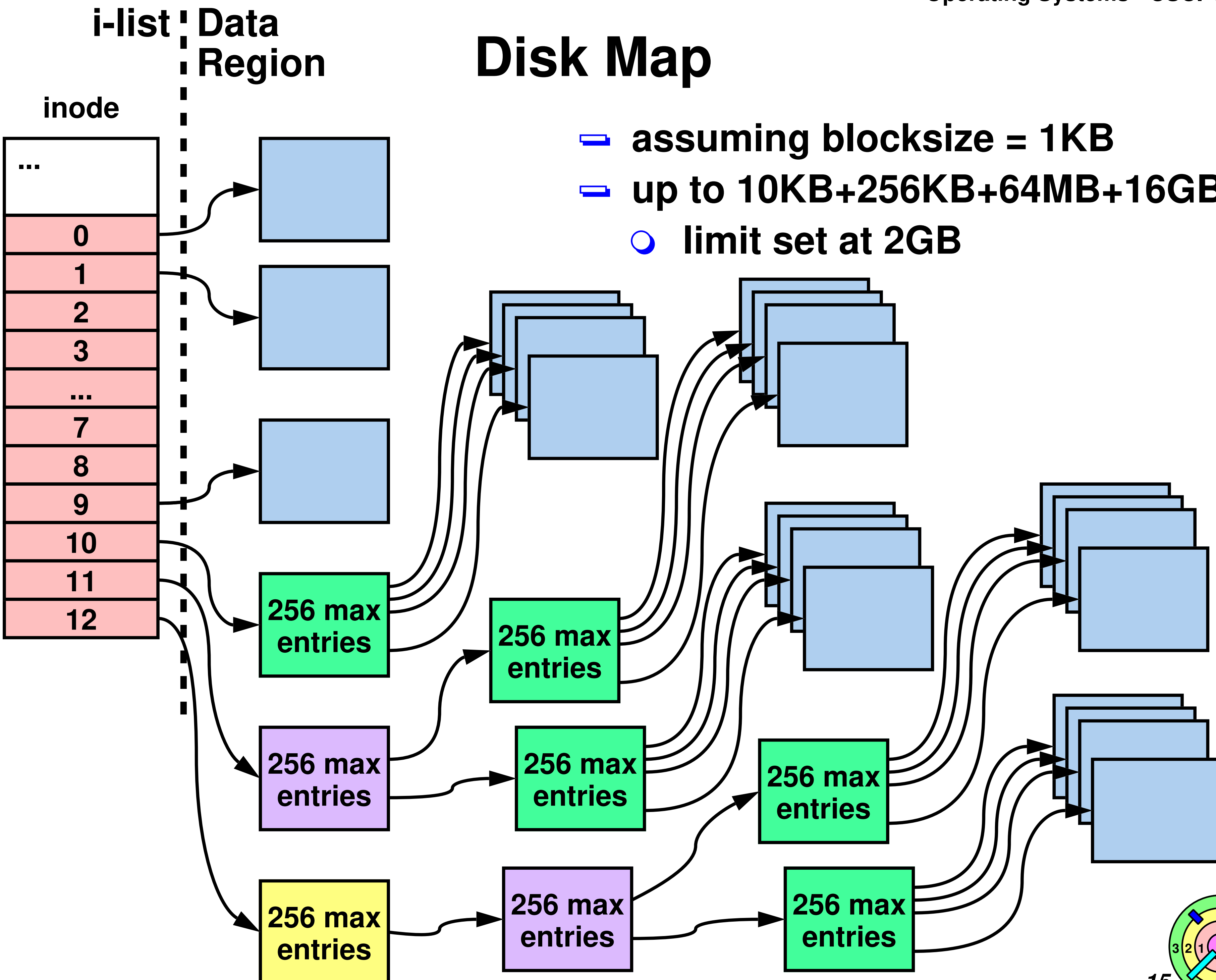
Disk Map

- assuming blocksize = 1KB
- up to 10KB+256KB+64MB

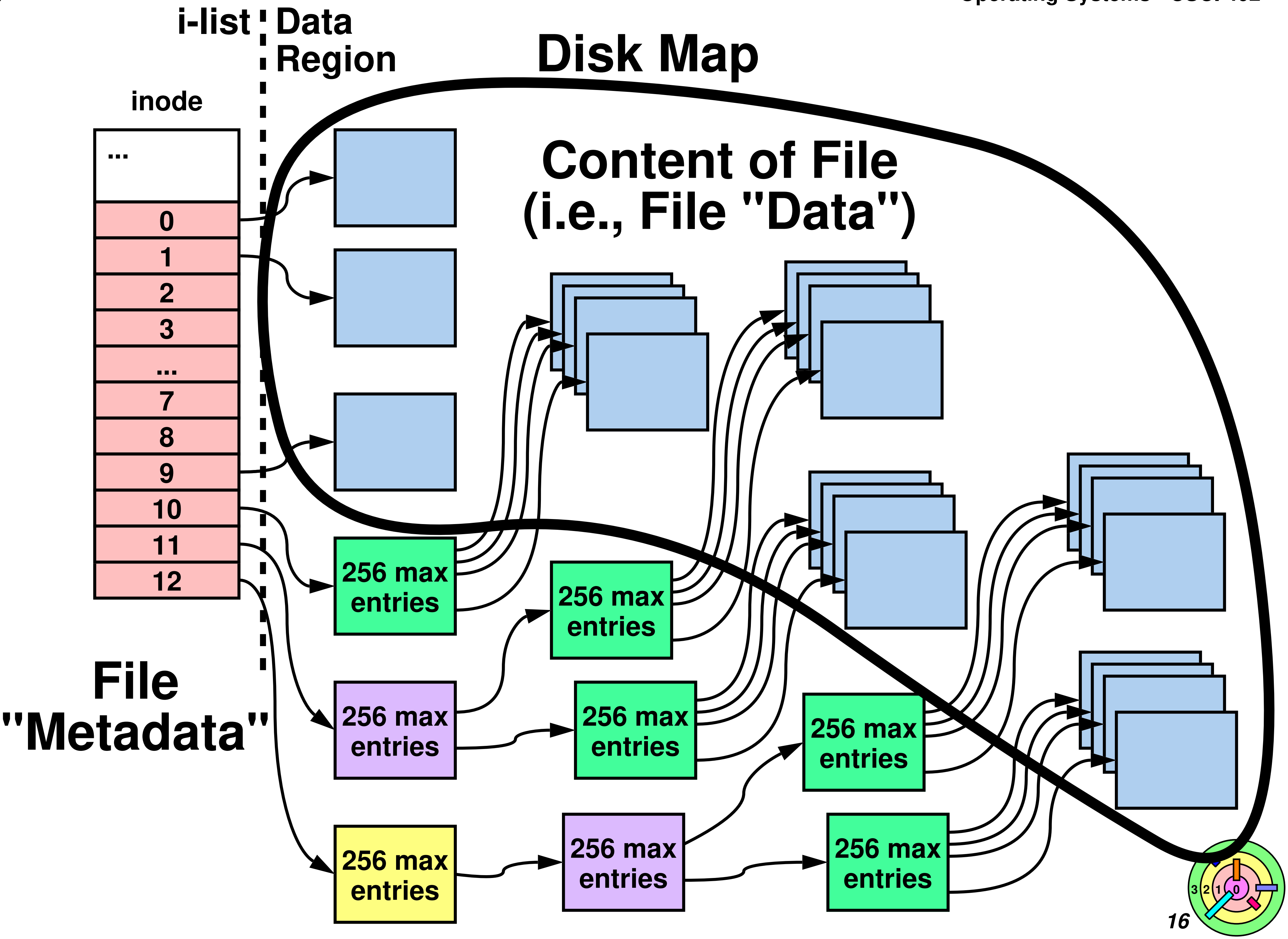


Disk Map

- assuming blocksize = 1KB
- up to 10KB+256KB+64MB+16GB
  - limit set at 2GB



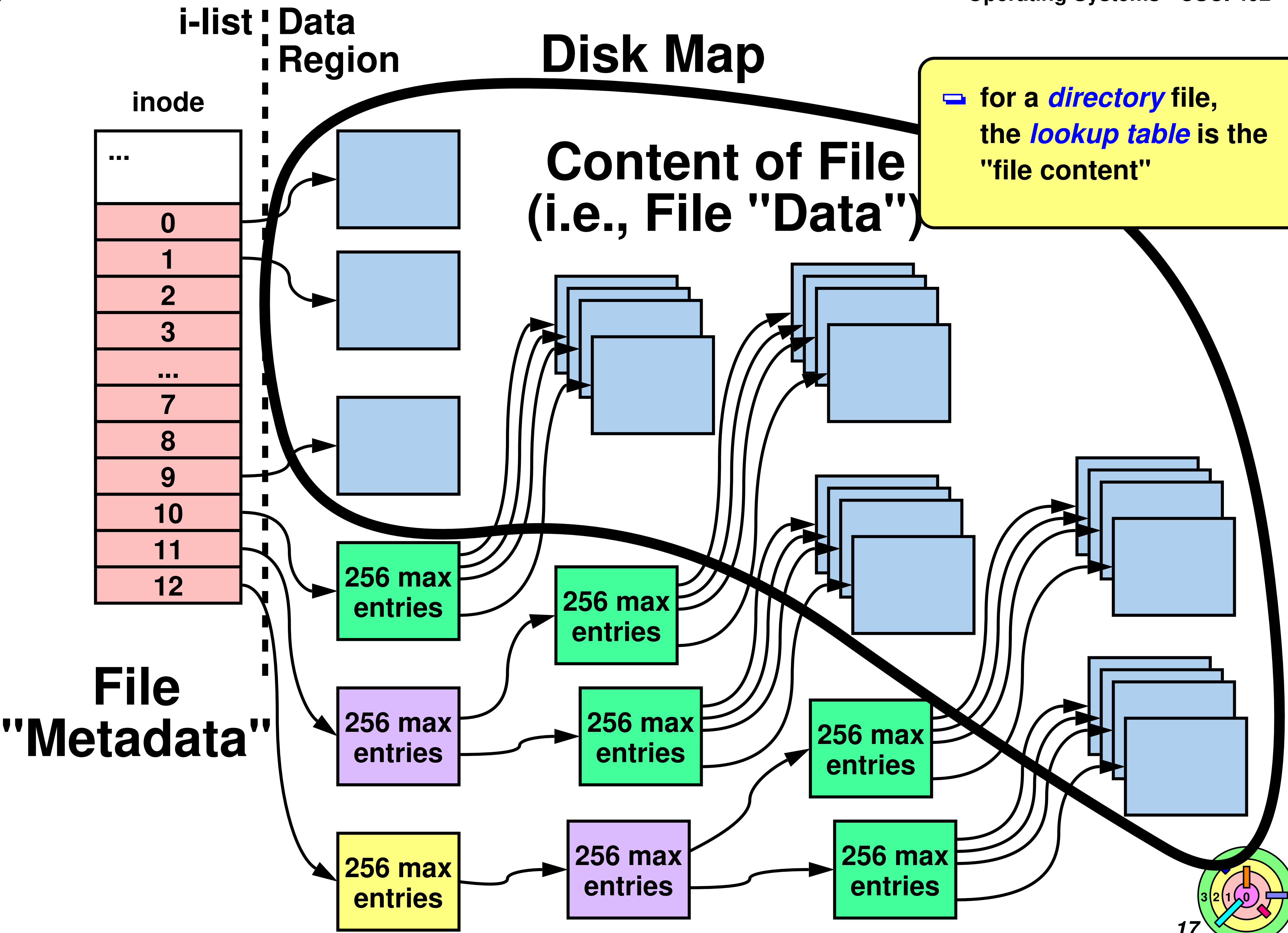
Disk Map



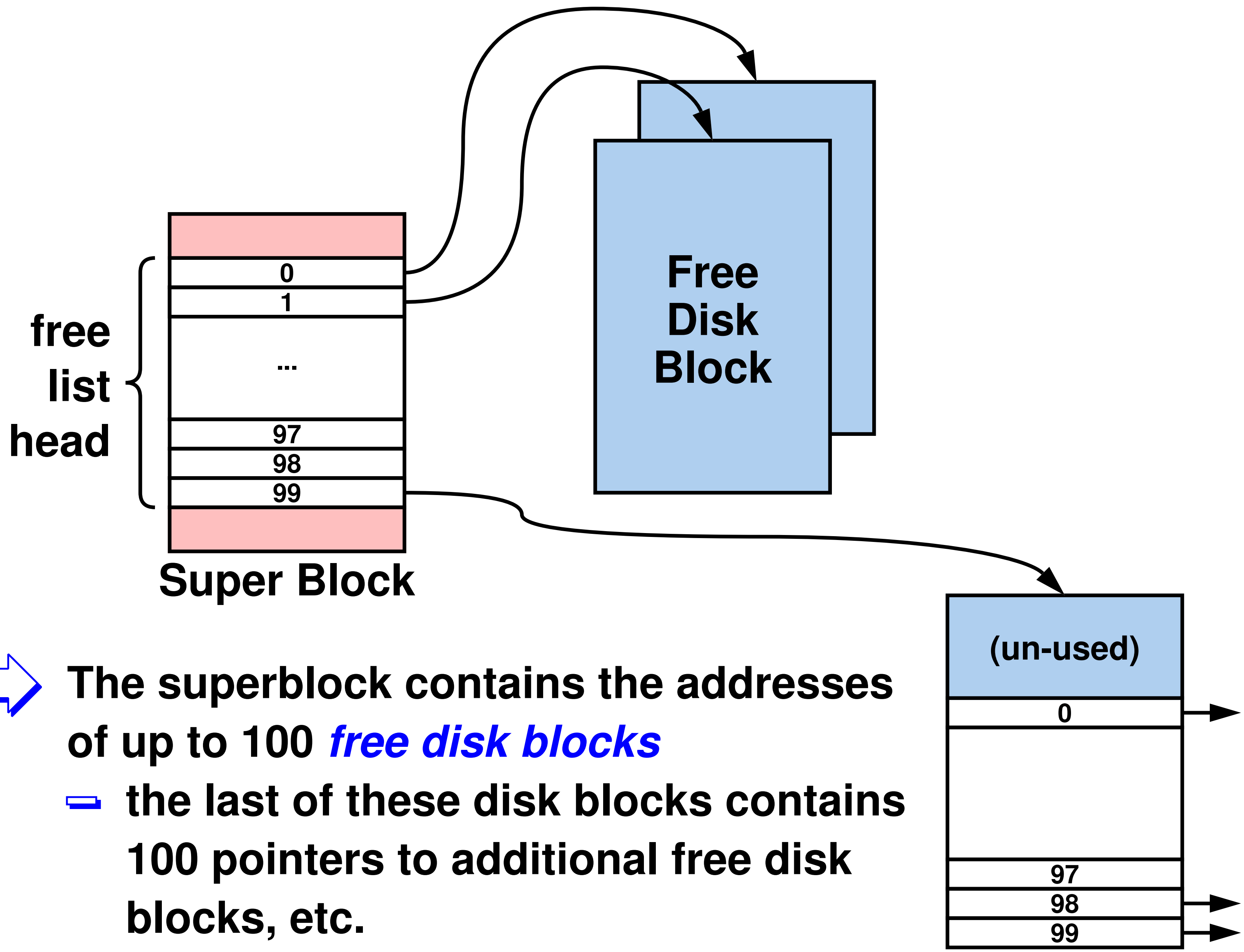
i-list
Data Region
inode
...
0
1
2
3
...
7
8
9
10
11
12
Content of File (i.e., File "Data")
256 max entries
256 max entries
256 max entries
256 max entries
256 max entries
256 max entries
256 max entries
256 max entries
File "Metadata"

Disk Map

- for a *directory* file, the *lookup table* is the "file content"

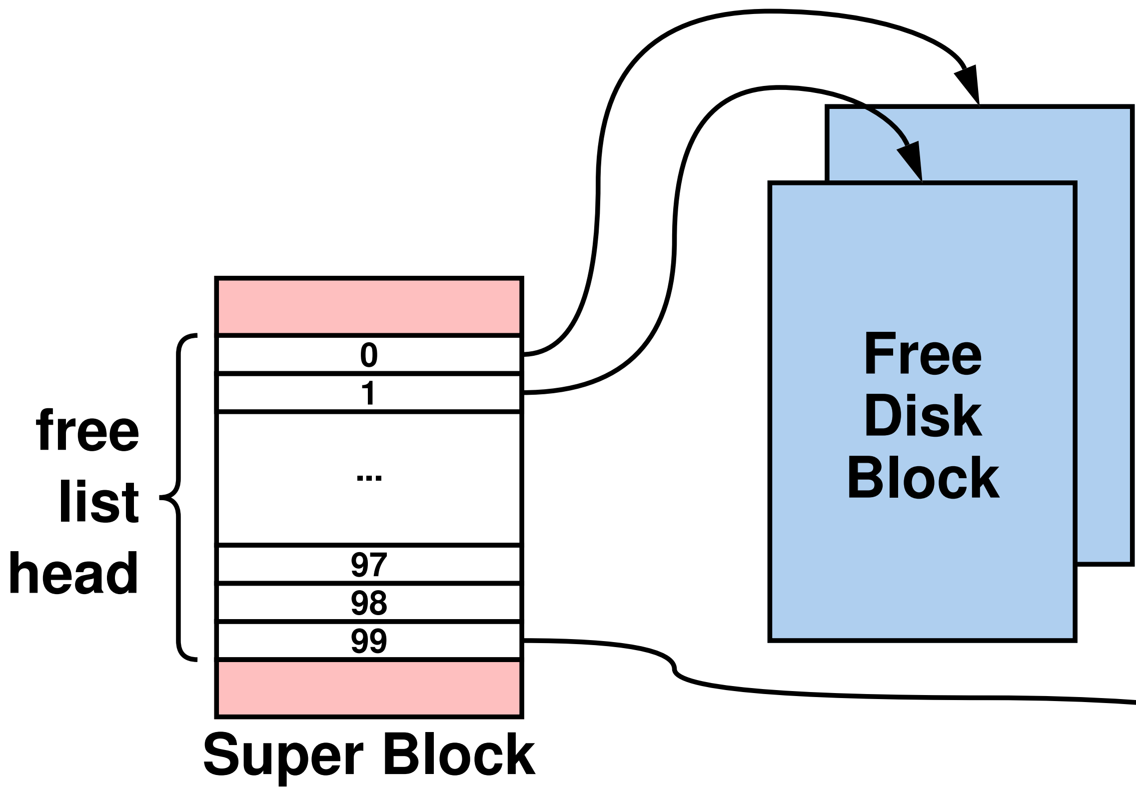


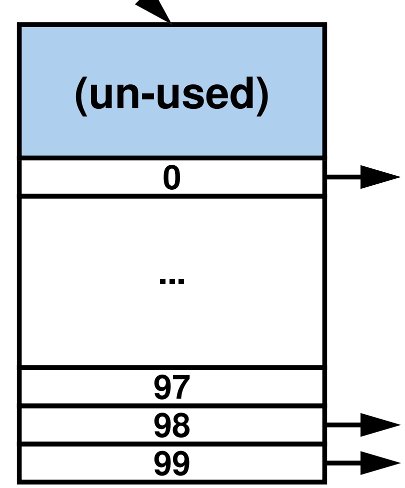
S5FS Free List



- The superblock contains the addresses of up to 100 *free disk blocks*
  - the last of these disk blocks contains 100 pointers to additional free disk blocks, etc.
  - can find *all* the free disk blocks this way

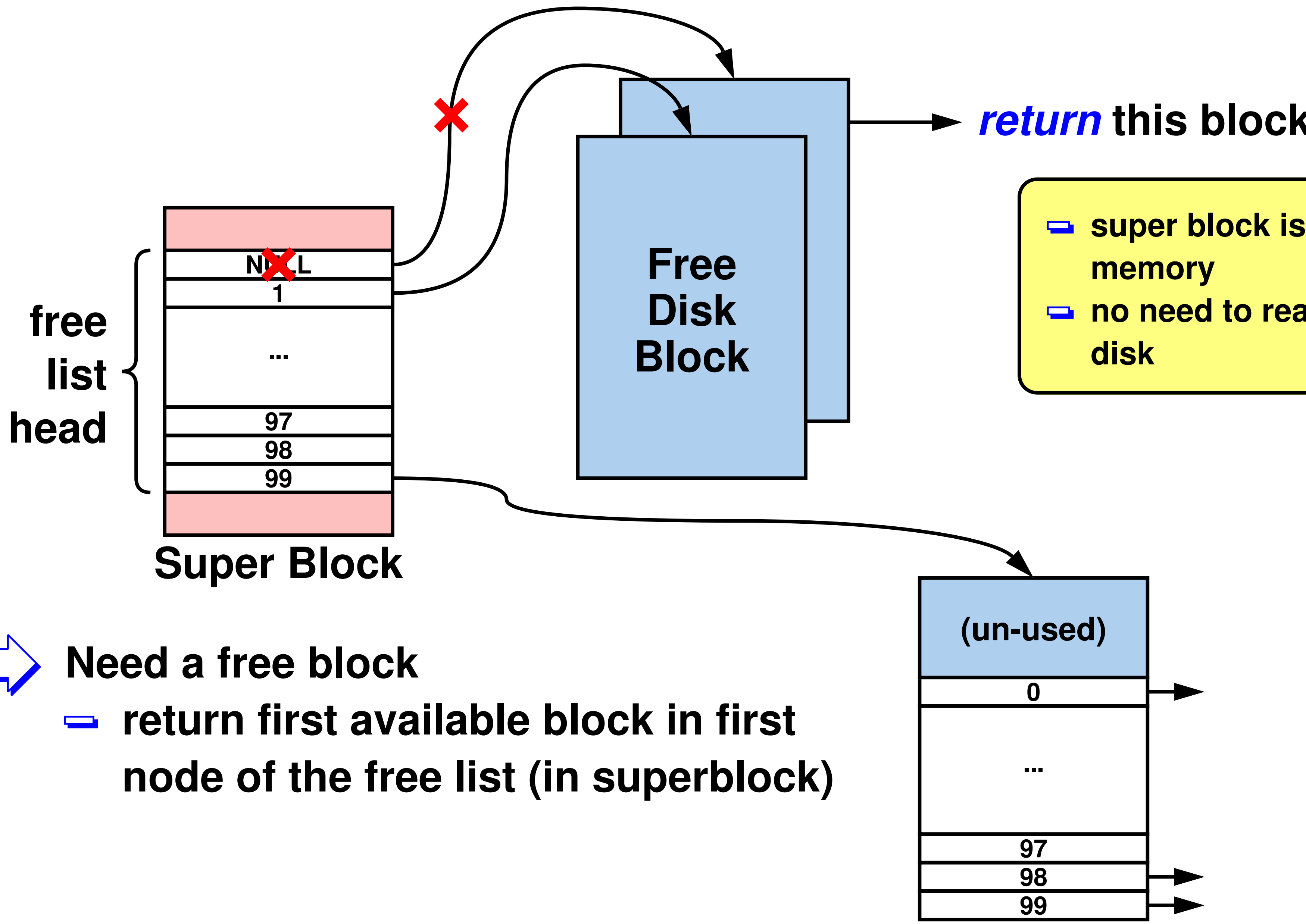
S5FS Free List: Allocation - Case (1)





- Need a free block
  - return first available block in first node of the free list (in superblock)

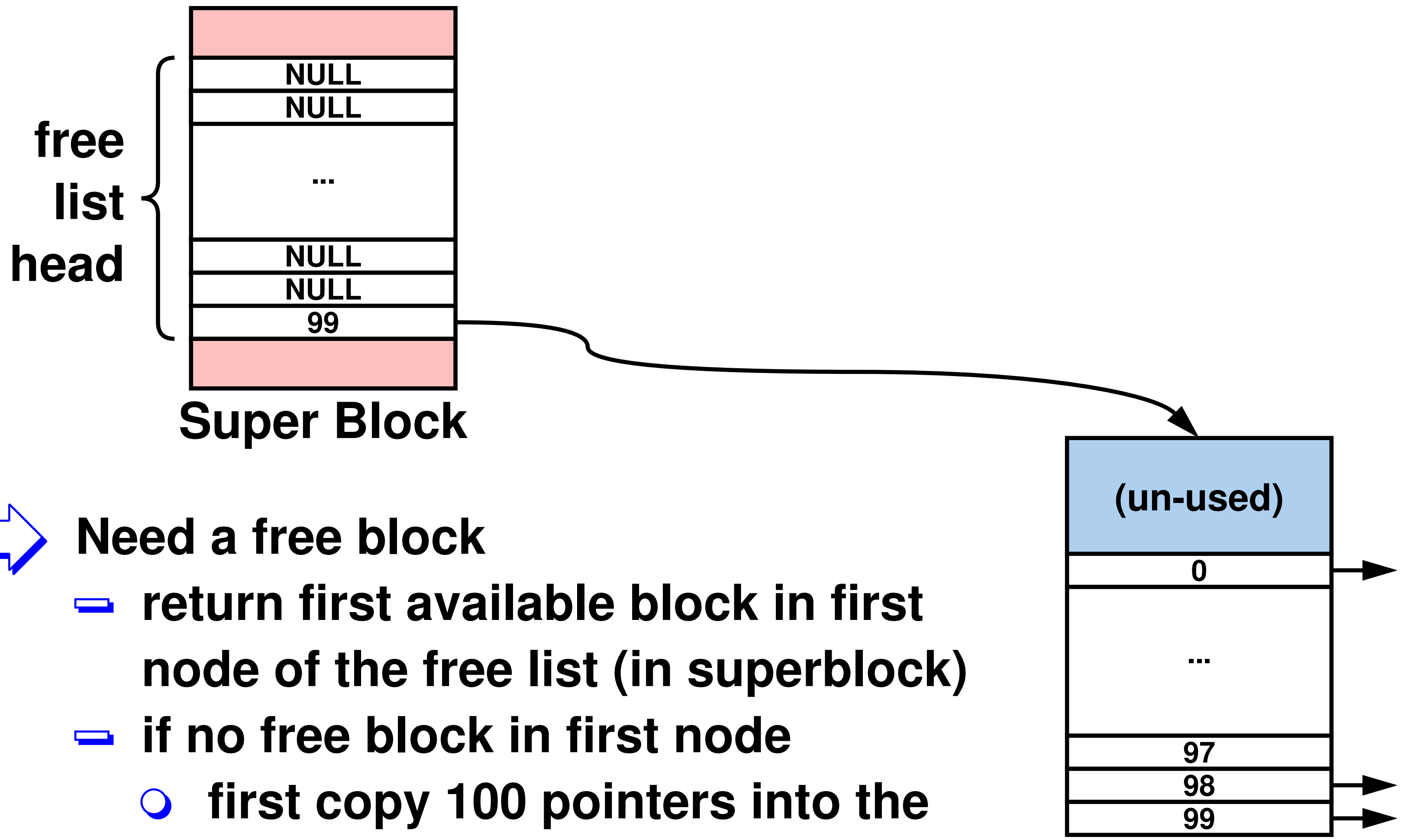
S5FS Free List: Allocation - Case (1)



- super block is cached in memory
- no need to read from disk

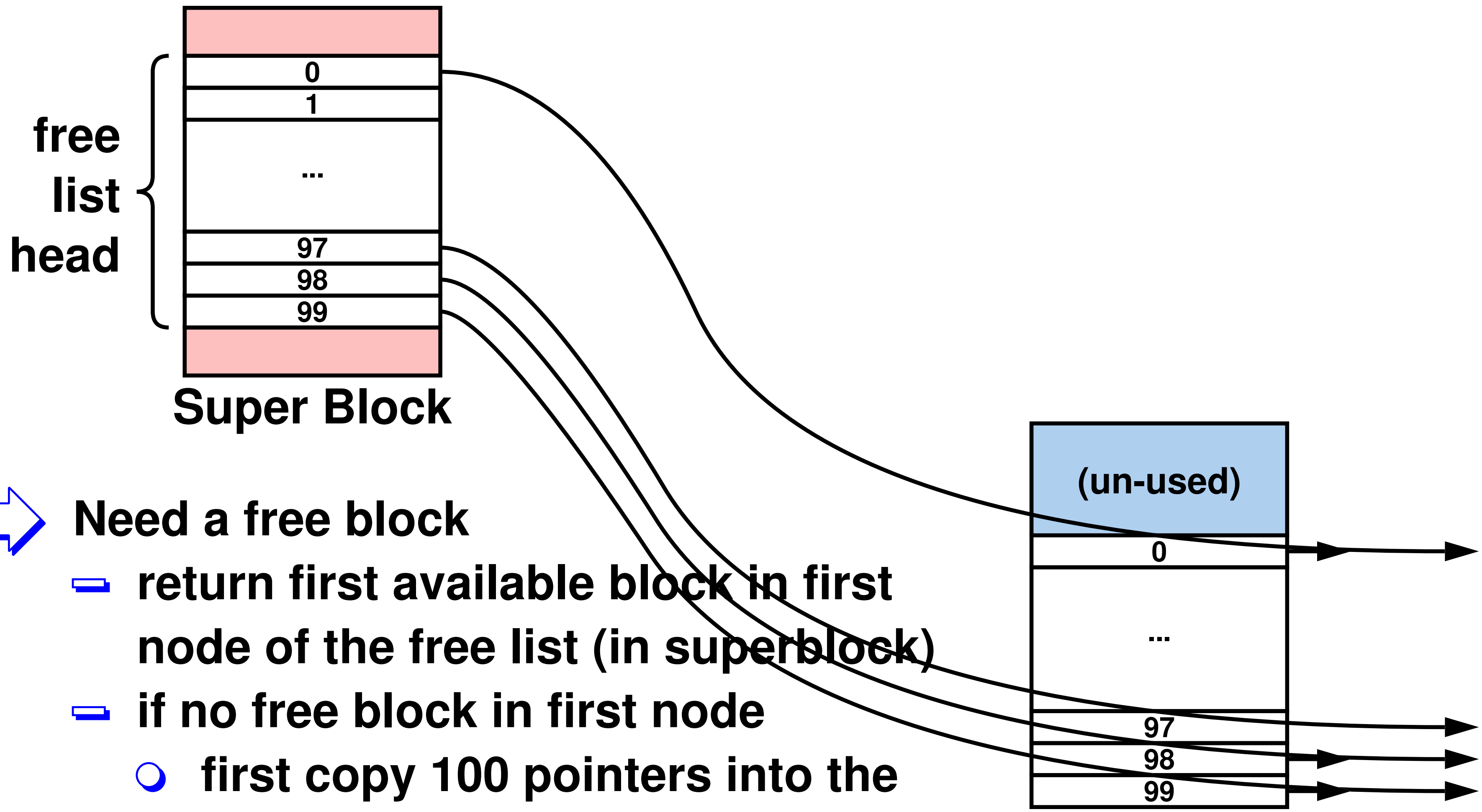
- Need a free block
  - return first available block in first node of the free list (in superblock)

S5FS Free List: Allocation - Case (2)



- Need a free block
  - return first available block in first node of the free list (in superblock)
  - if no free block in first node
    - first copy 100 pointers into the superblock with one disk access

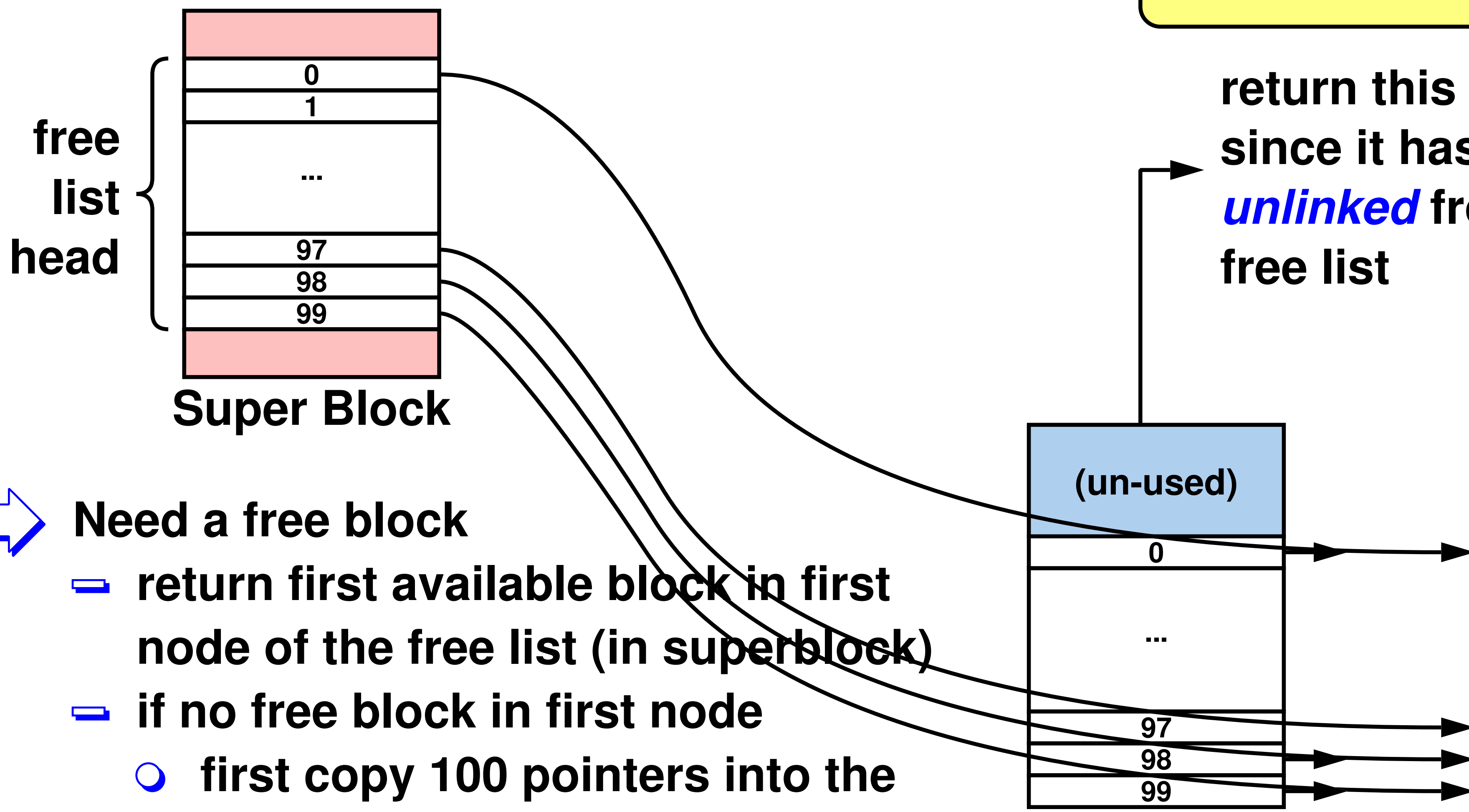
S5FS Free List: Allocation - Case (2)



- Need a free block
  - return first available block in first node of the free list (in superblock)
  - if no free block in first node
    - first copy 100 pointers into the superblock with one disk access

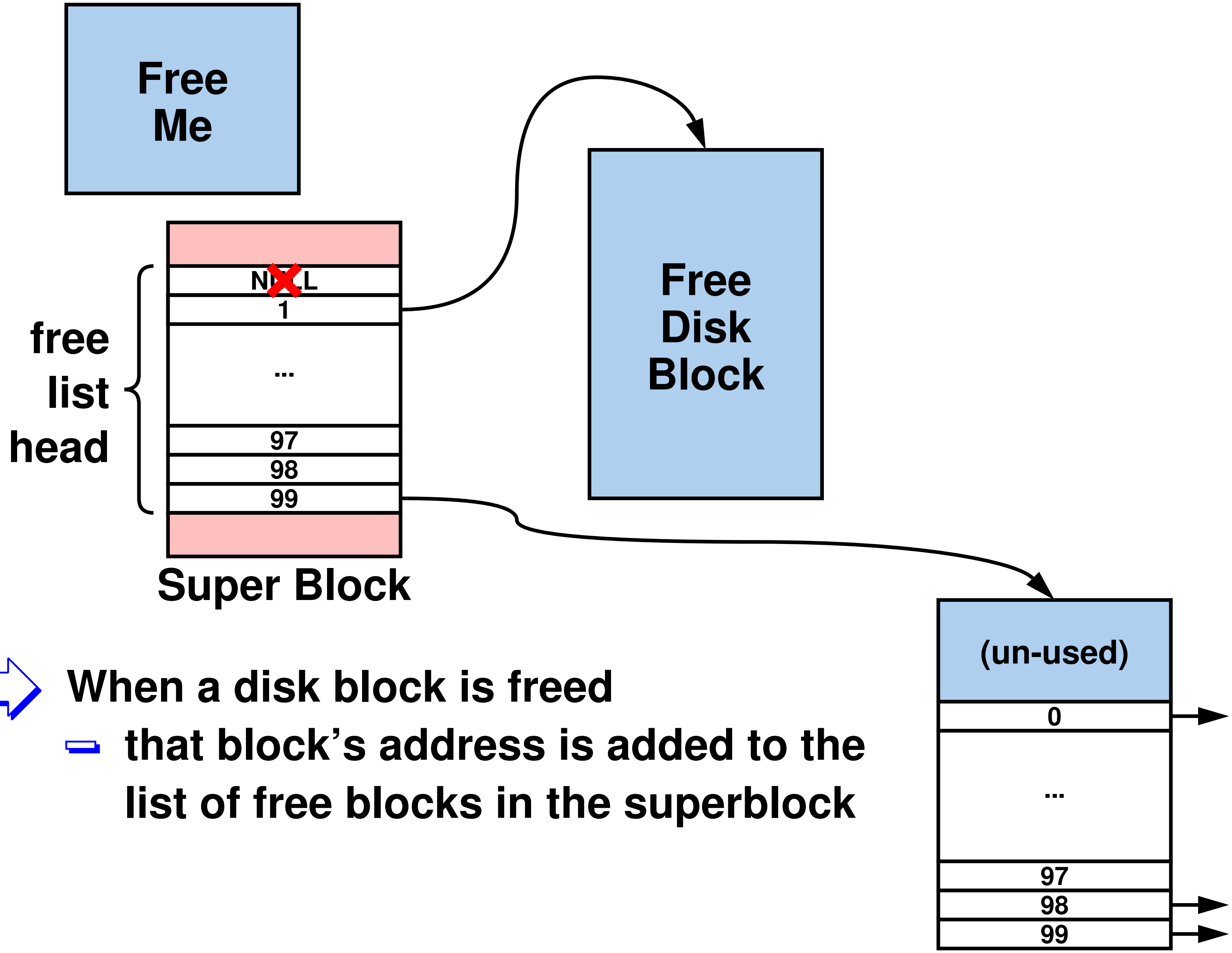
S5FS Free List: Allocation - Case (2)

- one disk read to copy disk block pointers
- only happens 1 out of 100 allocations



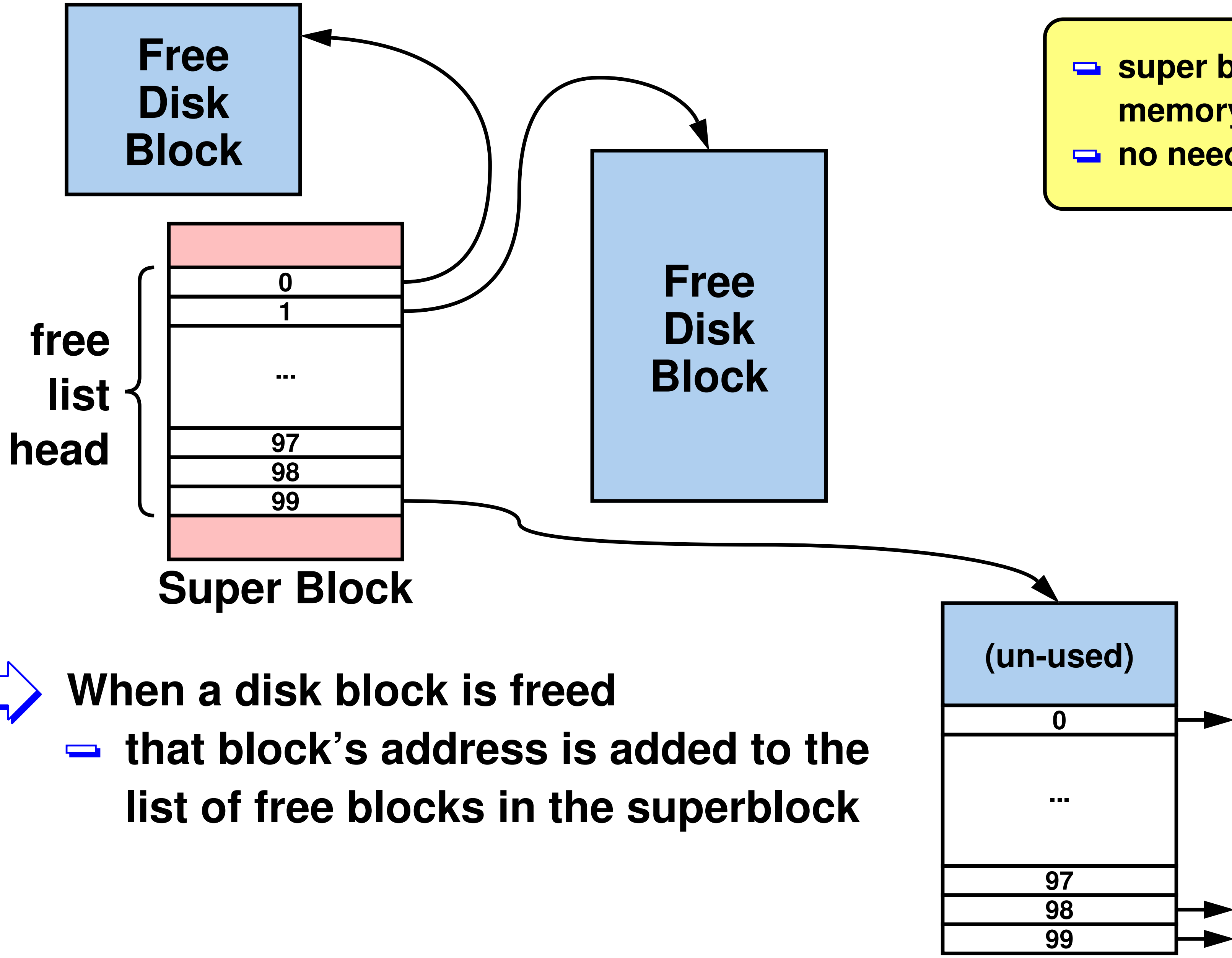
- Need a free block
  - return first available block in first node of the free list (in superblock)
  - if no free block in first node
    - first copy 100 pointers into the superblock with one disk access

S5FS Free List: Deallocation - Case (1)



- When a disk block is freed
  - that block's address is added to the list of free blocks in the superblock

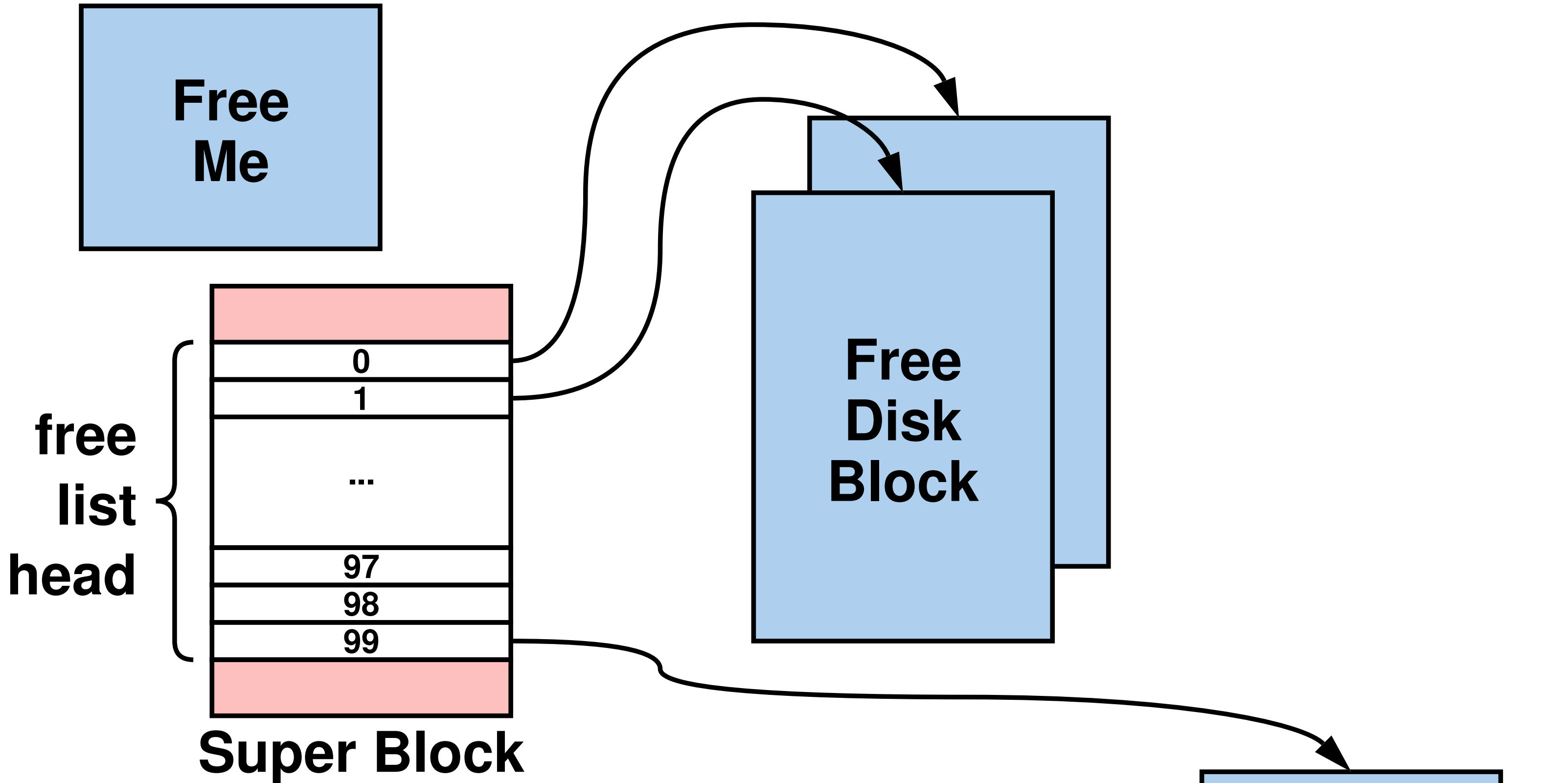
S5FS Free List: Deallocation - Case (1)

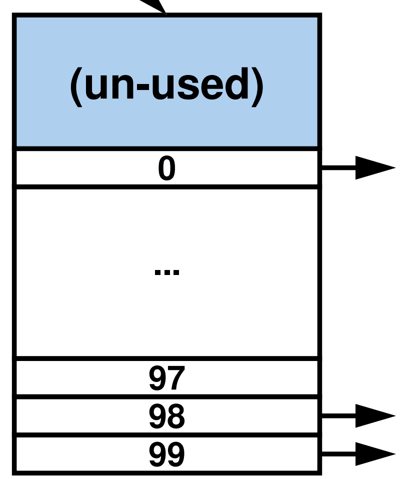


- super block is cached in memory
- no need to write to disk

- When a disk block is freed
  - that block's address is added to the list of free blocks in the superblock

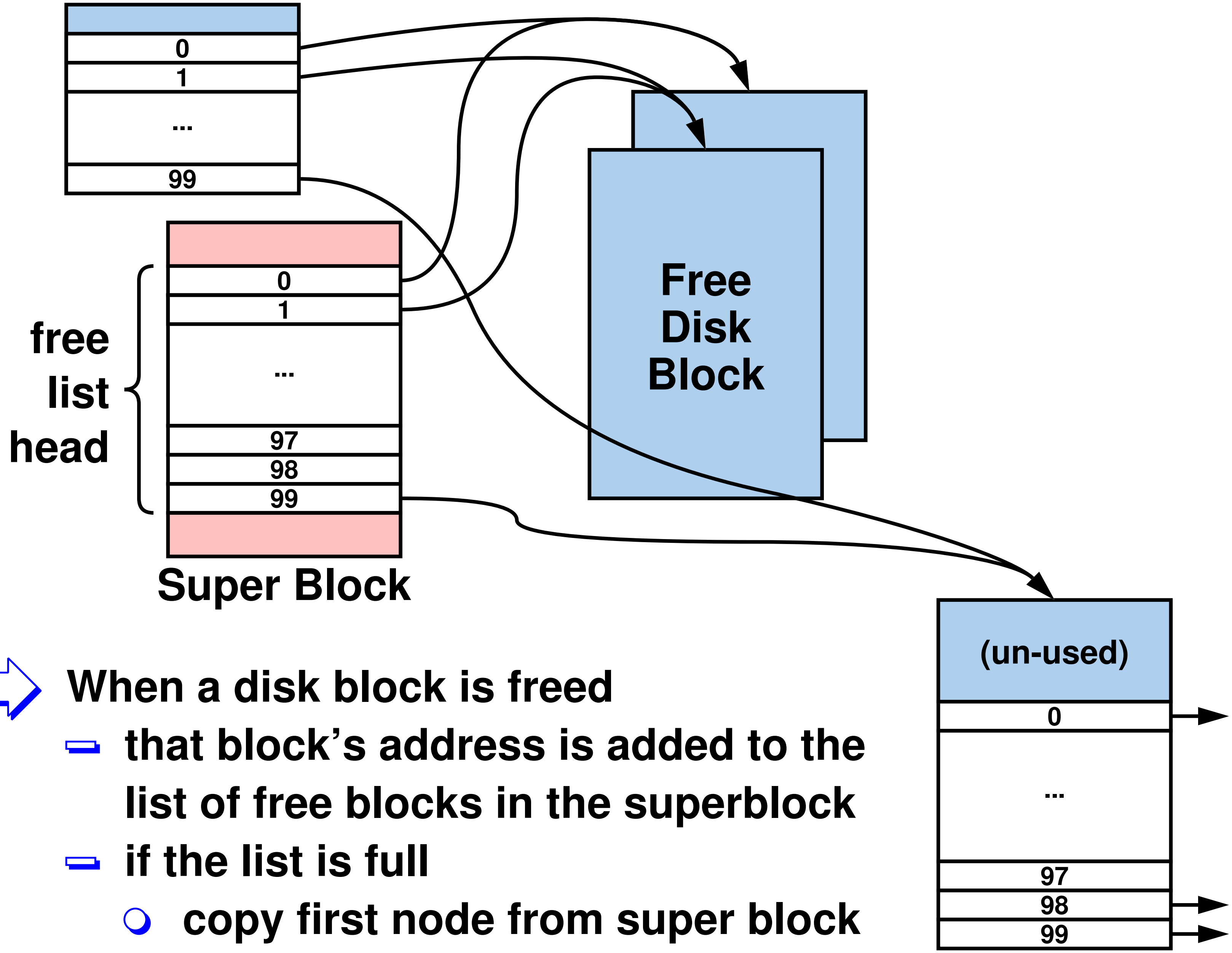
S5FS Free List: Deallocation - Case (2)





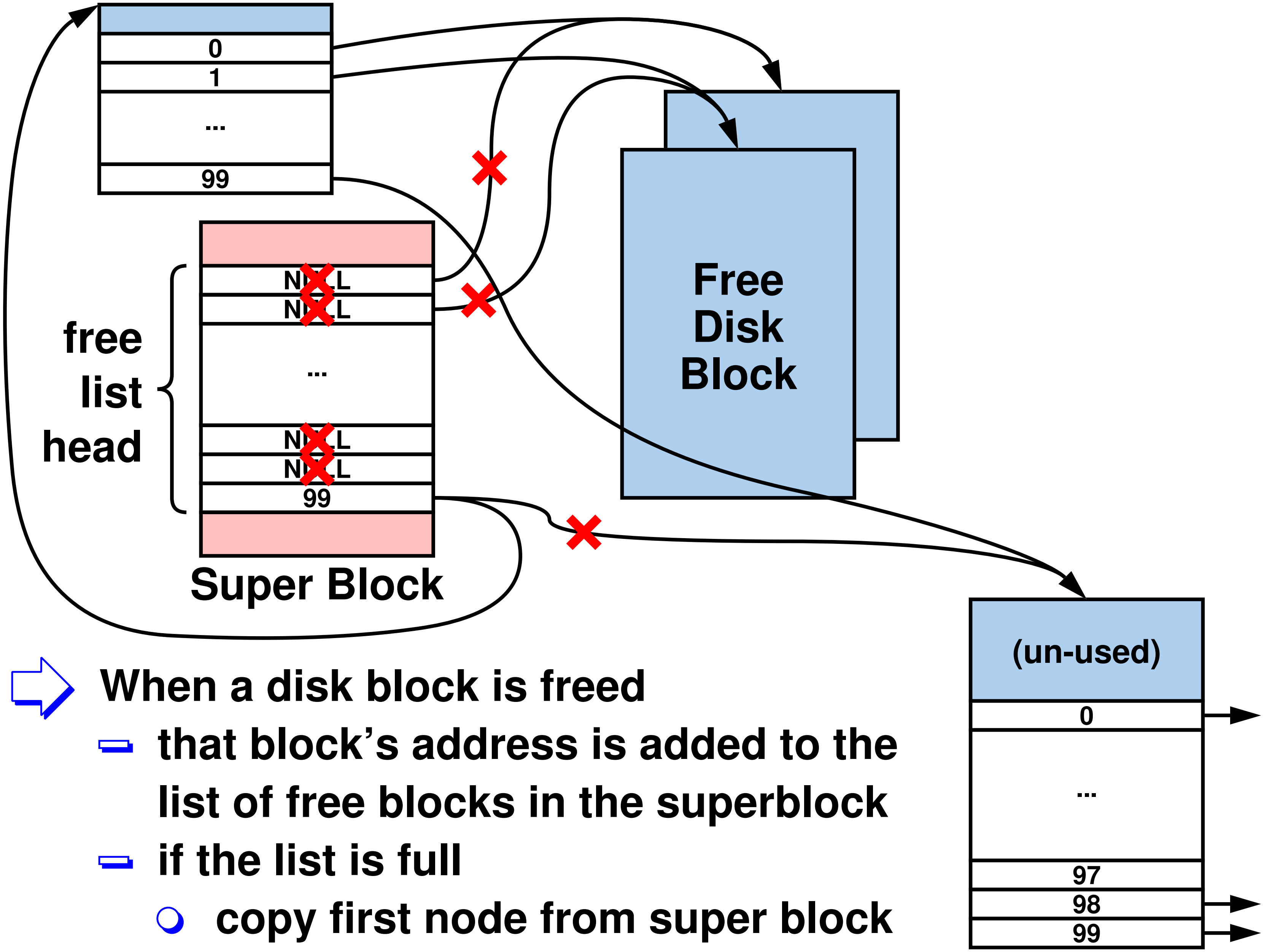
- When a disk block is freed
  - that block's address is added to the list of free blocks in the superblock
  - if the list is full
    - copy first node from super block into the newly freed block and update superblock

S5FS Free List: Deallocation - Case (2)



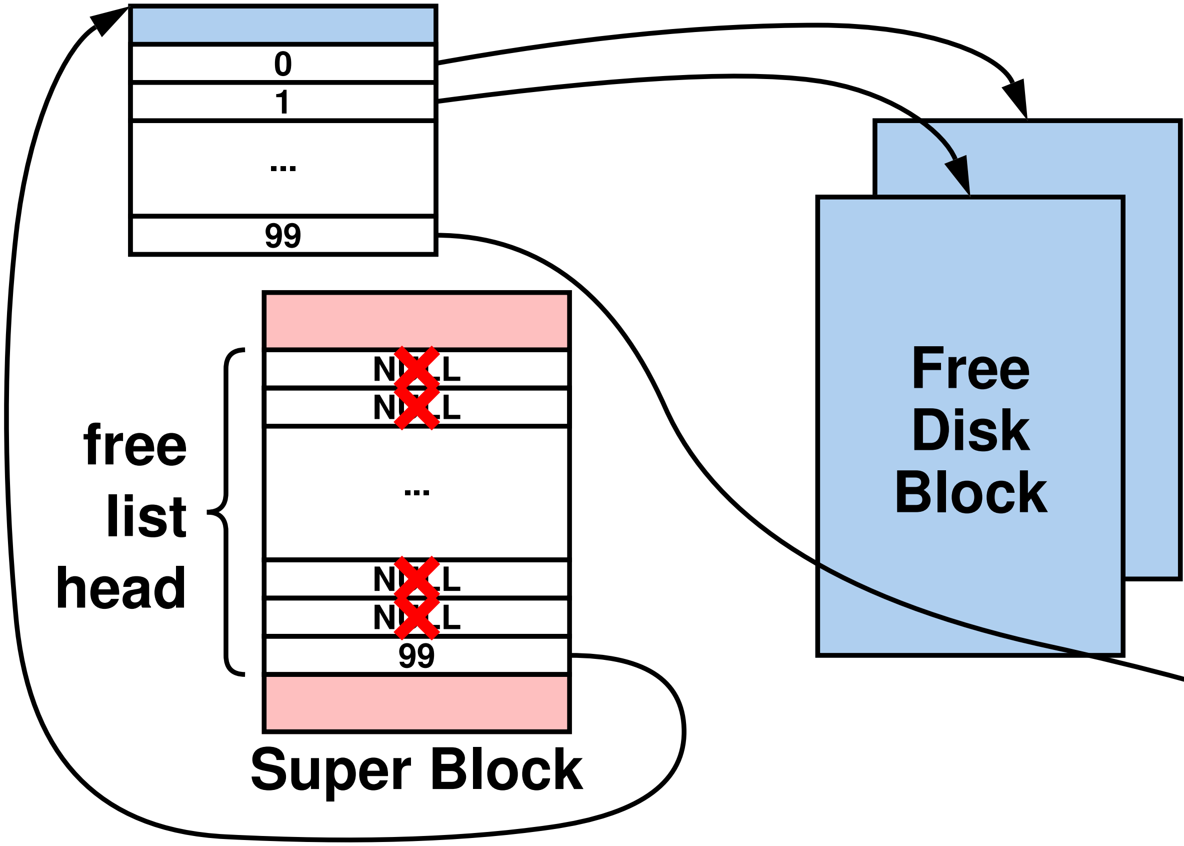
- When a disk block is freed
  - that block's address is added to the list of free blocks in the superblock
  - if the list is full
    - copy first node from super block into the newly freed block and update superblock

S5FS Free List: Deallocation - Case (2)

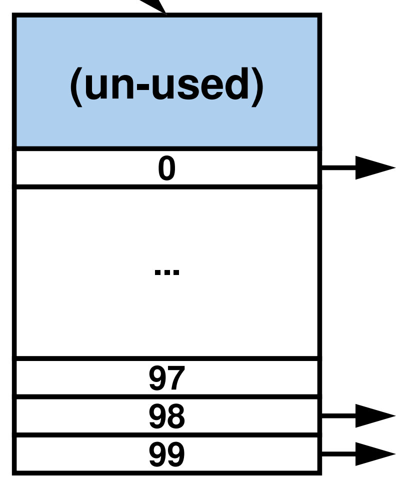


- When a disk block is freed
  - that block's address is added to the list of free blocks in the superblock
  - if the list is full
    - copy first node from super block into the newly freed block and update superblock

S5FS Free List: Deallocation - Case (2)



- one disk write to copy disk block pointers
- only happens 1 out of 100 frees

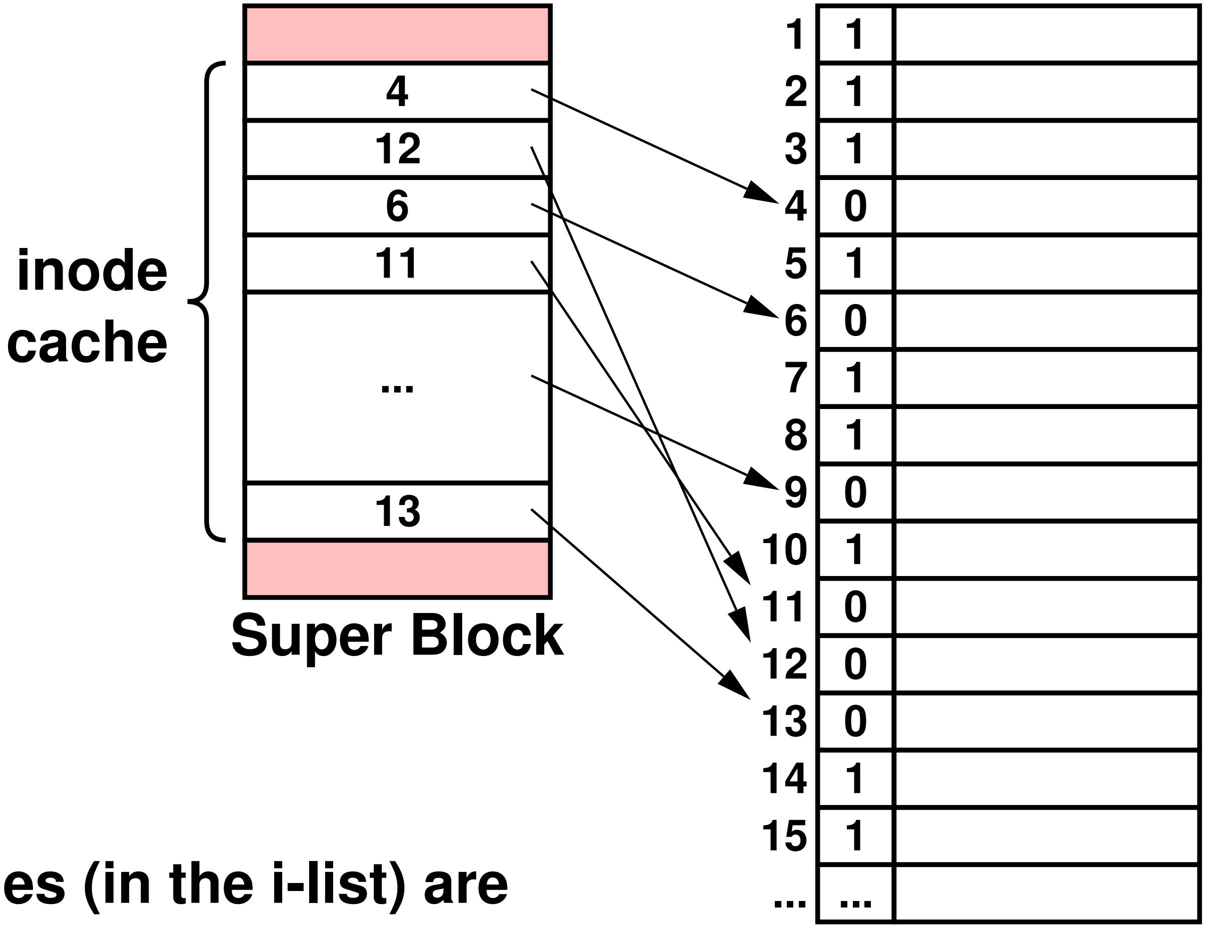


- When a disk block is freed
  - that block's address is added to the list of free blocks in the superblock
  - if the list is full
    - copy first node from super block into the newly freed block and update superblock

S5FS Free List

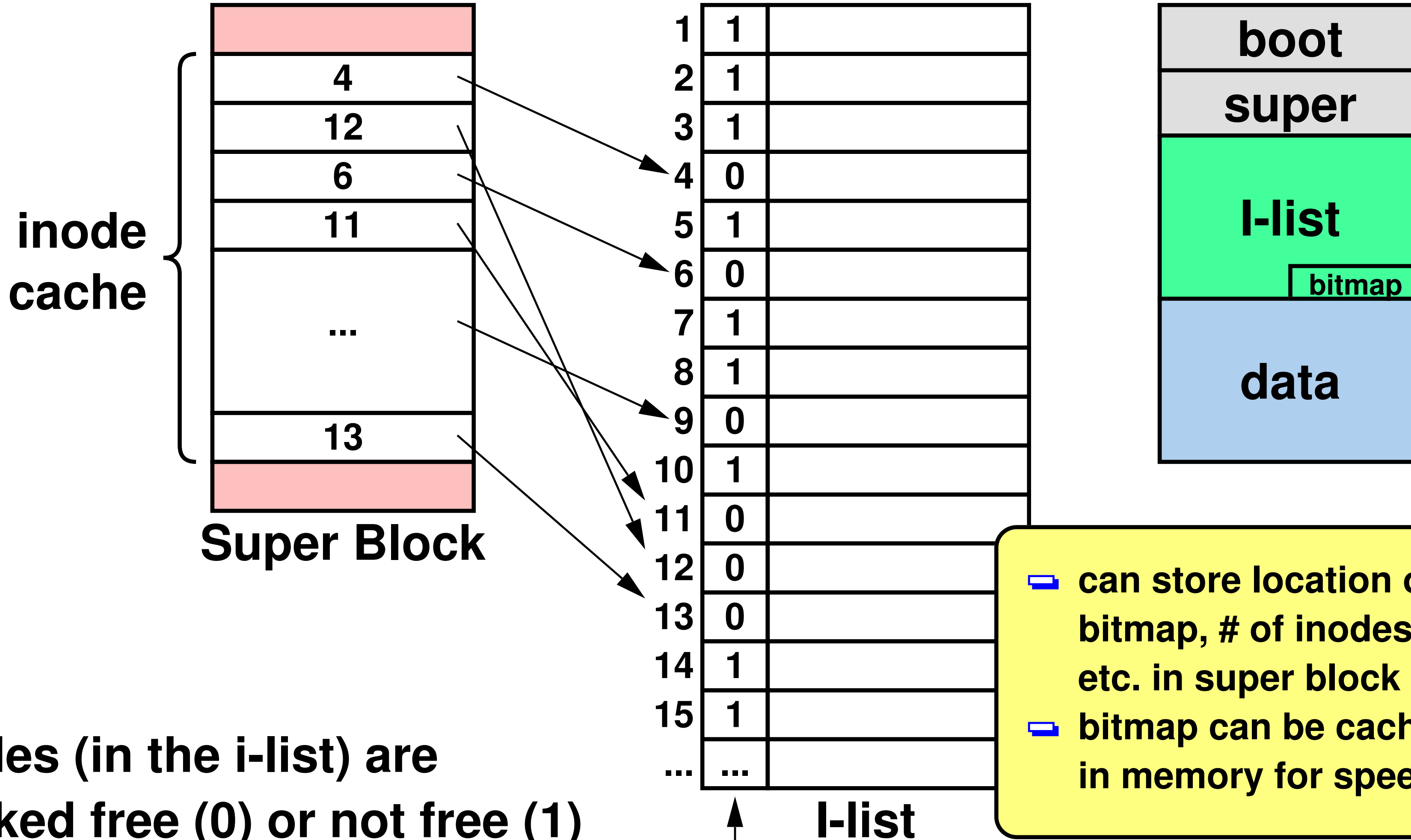
- Why 100 disk pointers in each free block?
  - to *reduce the number of disk accesses*
    - for every 100 times you need to allocate/deallocate a free block, you only need to go to the disk 1 time (on the average)
      - effective allocation/deallocation time is 100 times faster!
  - is the reason also to reduces the length of the free list?
    - no
    - but reducing the length of the free list reduces the number of disk accesses
      - correct, but that's a secondary effect
      - it's important to be able to distinguish between what's primary and what's secondary effects

S5FS Free Inode List



- Inodes (in the i-list) are marked free (0) or not free (1)
  - use a "bitmap" to store this
  - no additional organization in the i-list
  - the superblock *caches* *free* inodes *(i.e., in the inode cache)*

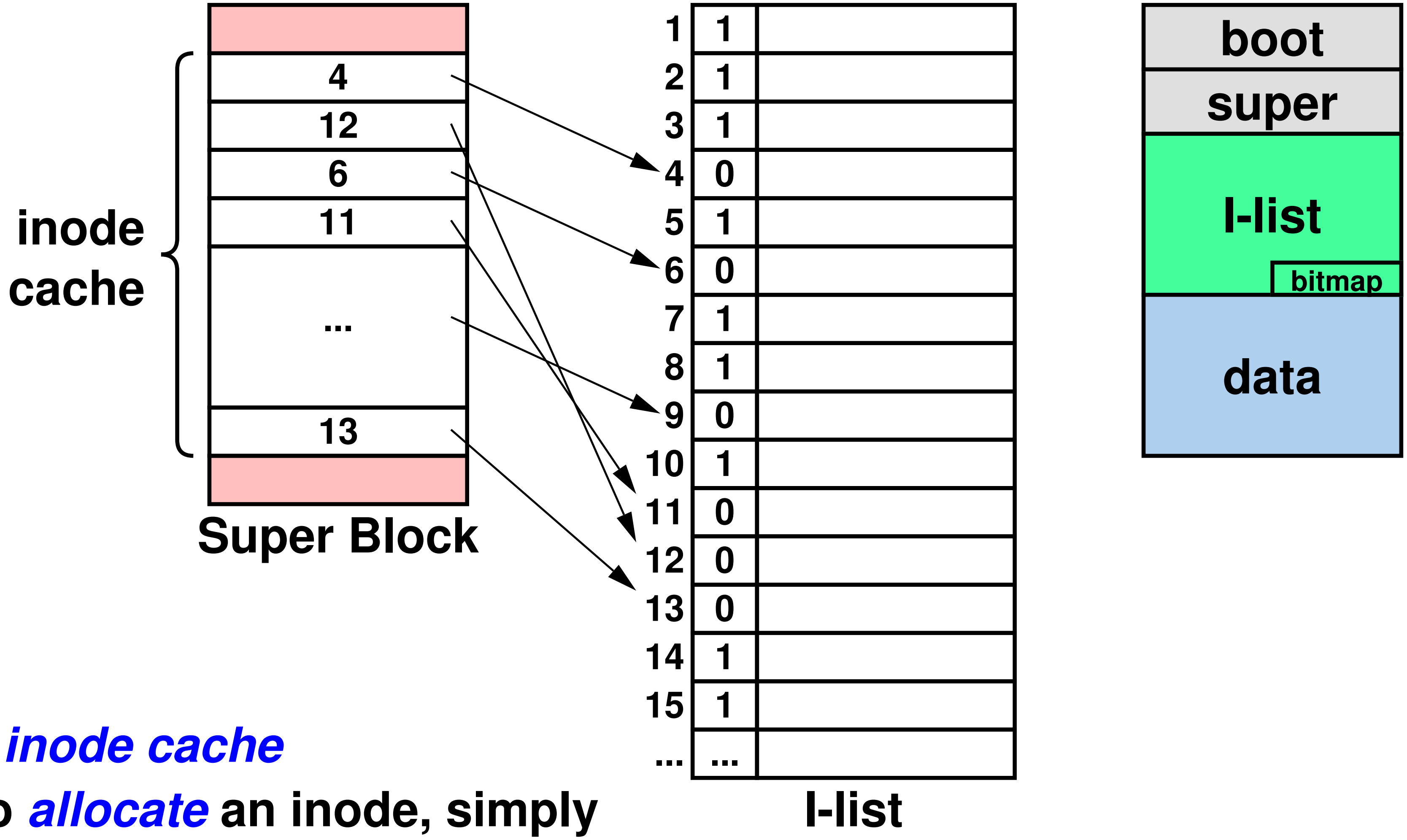
S5FS Free Inode List



- can store location of bitmap, # of inodes, etc. in super block
- bitmap can be cached in memory for speed

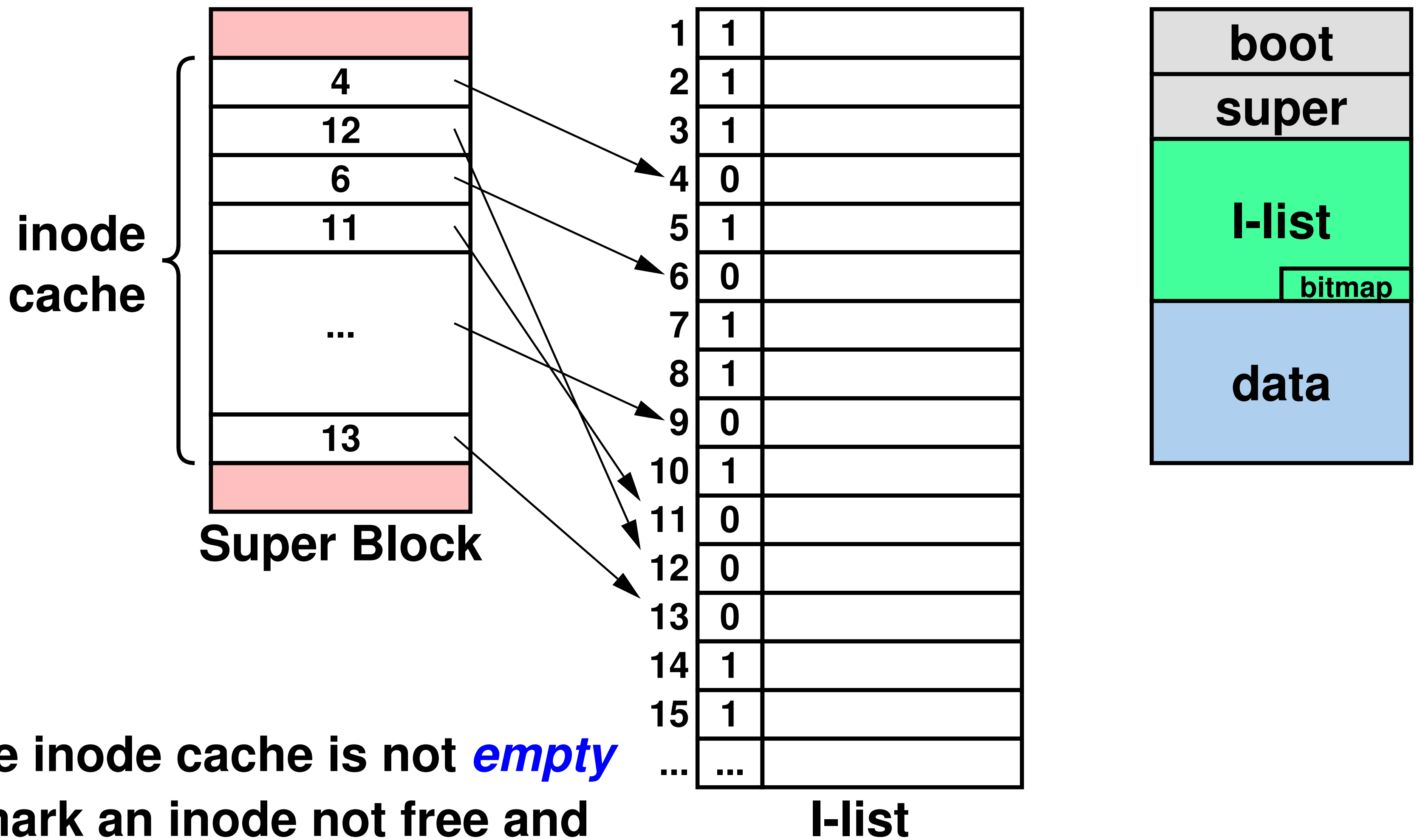
- Inodes (in the i-list) are marked free (0) or not free (1)
  - use a "bitmap" to store this
  - no additional organization in the i-list
  - the superblock *caches free* inodes *(i.e., in the inode cache)*

S5FS Free Inode List



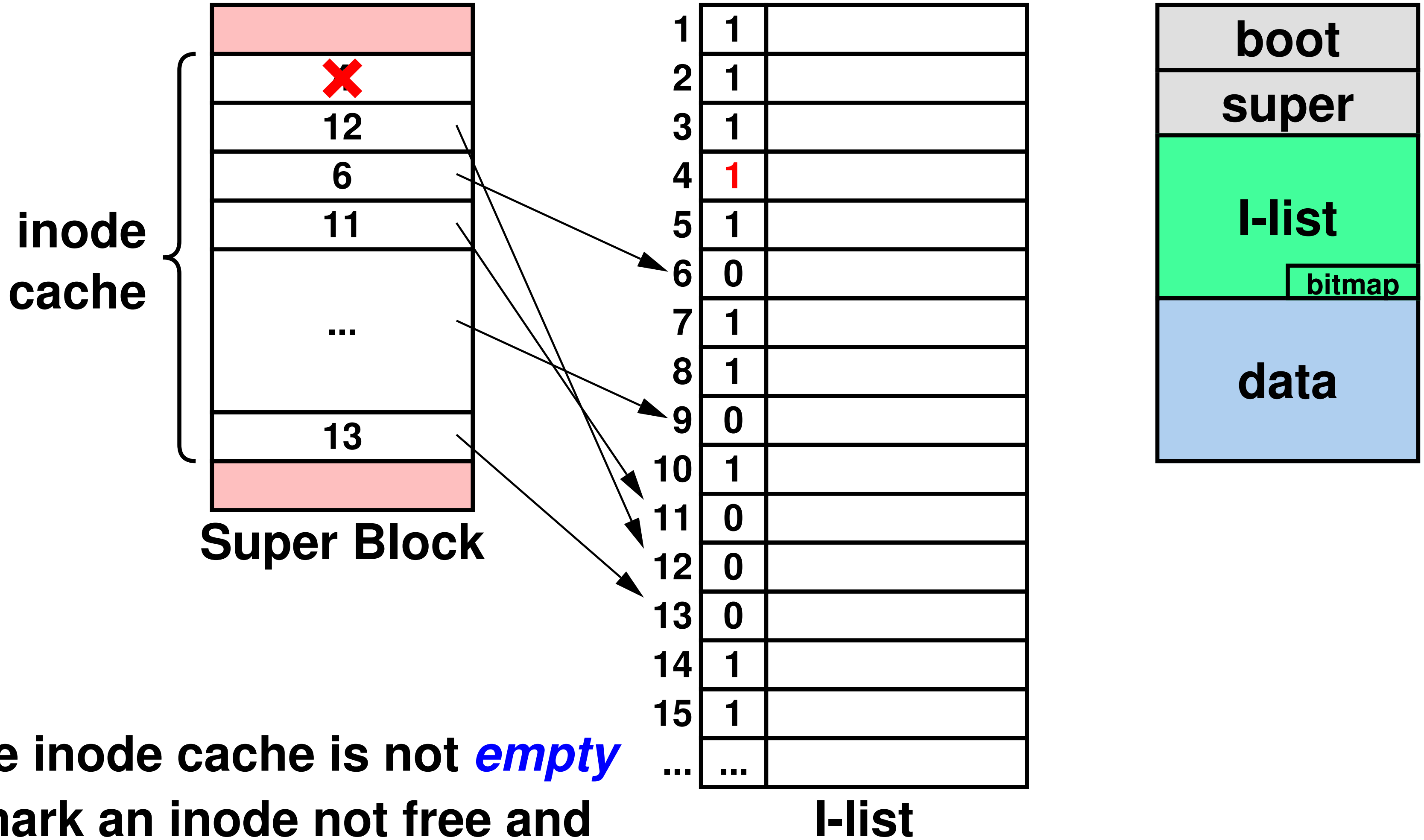
- The *inode cache*
  - to *allocate* an inode, simply mark it not free and remove it from the inode cache
  - to *free* an inode, simply mark it free and add to the inode cache if there is room

S5FS Free Inode List: Allocation - Case (1)



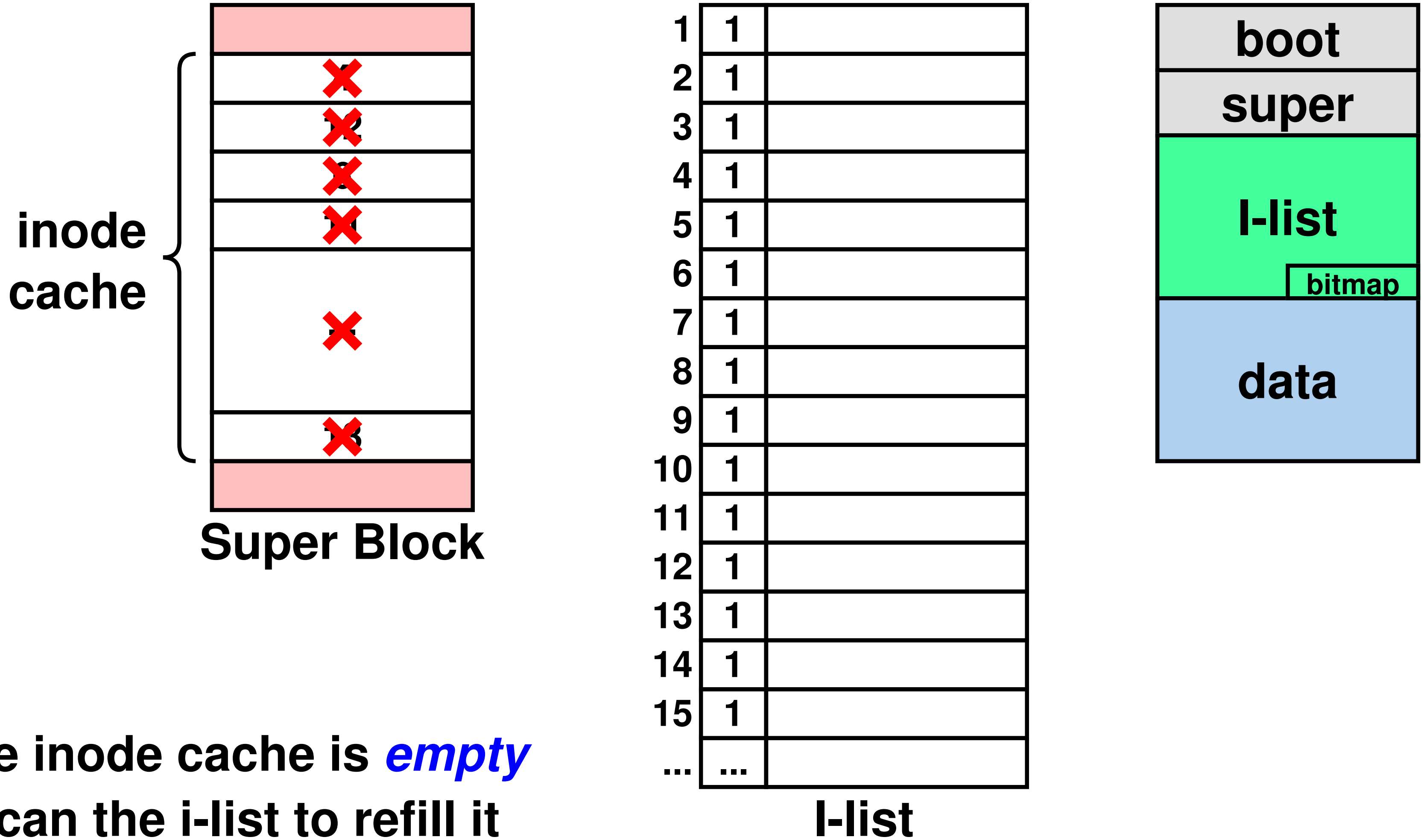
- If the inode cache is not *empty*
  - mark an inode not free and remove it from the inode cache
    - ex: need a free inode

S5FS Free Inode List: Allocation - Case (1)



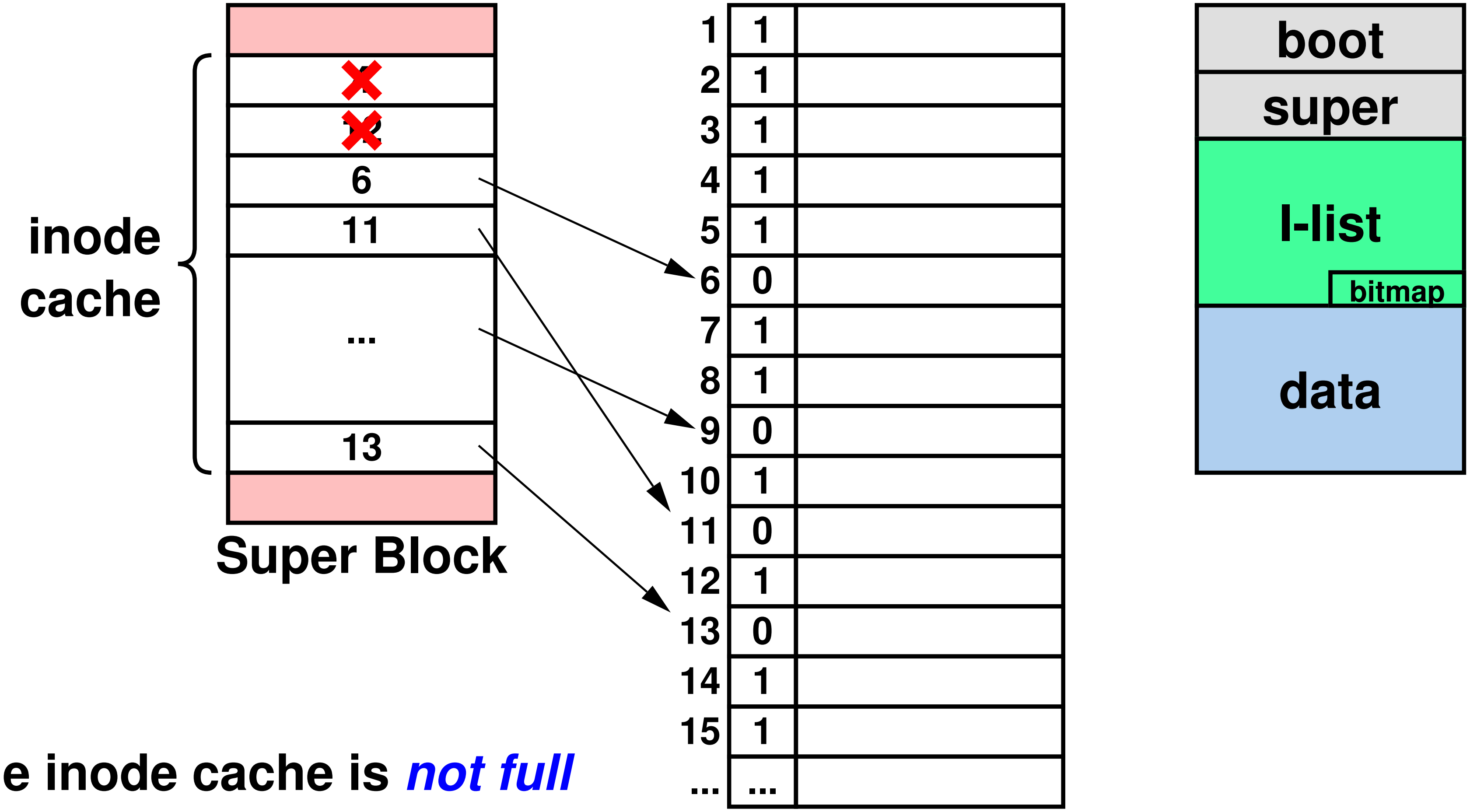
- If the inode cache is not *empty*
  - mark an inode not free and remove it from the inode cache
    - ex: need a free inode
      - inode 4 is now allocated (i.e., not free)

S5FS Free Inode List: Allocation - Case (2)



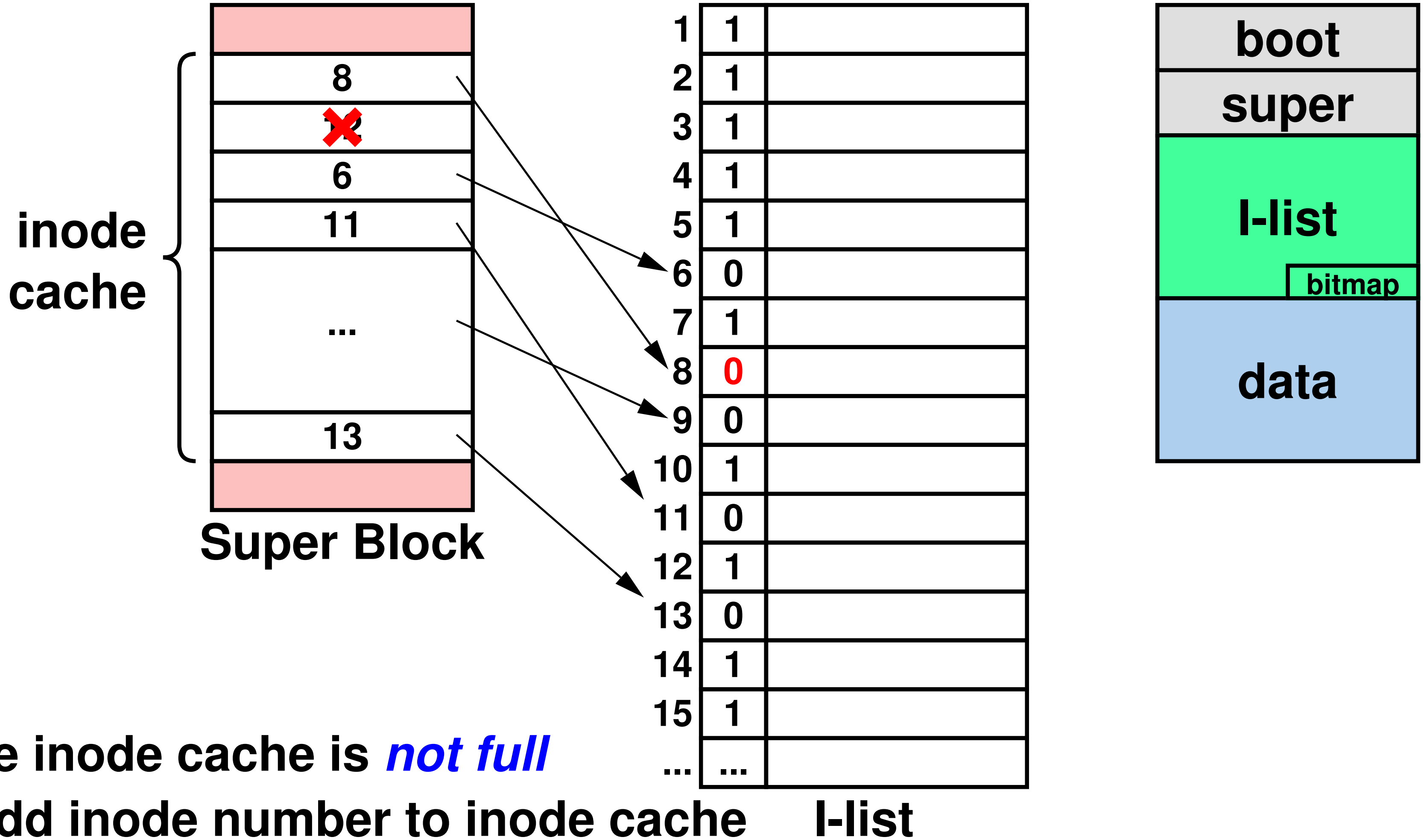
- If the inode cache is *empty*
  - scan the i-list to refill it
    - to help out with the scan, the super block keeps the first free inode in the i-list ("low water mark")
      - need to maintain this entry when necessary

S5FS Free Inode List: Deallocation - Case (1)



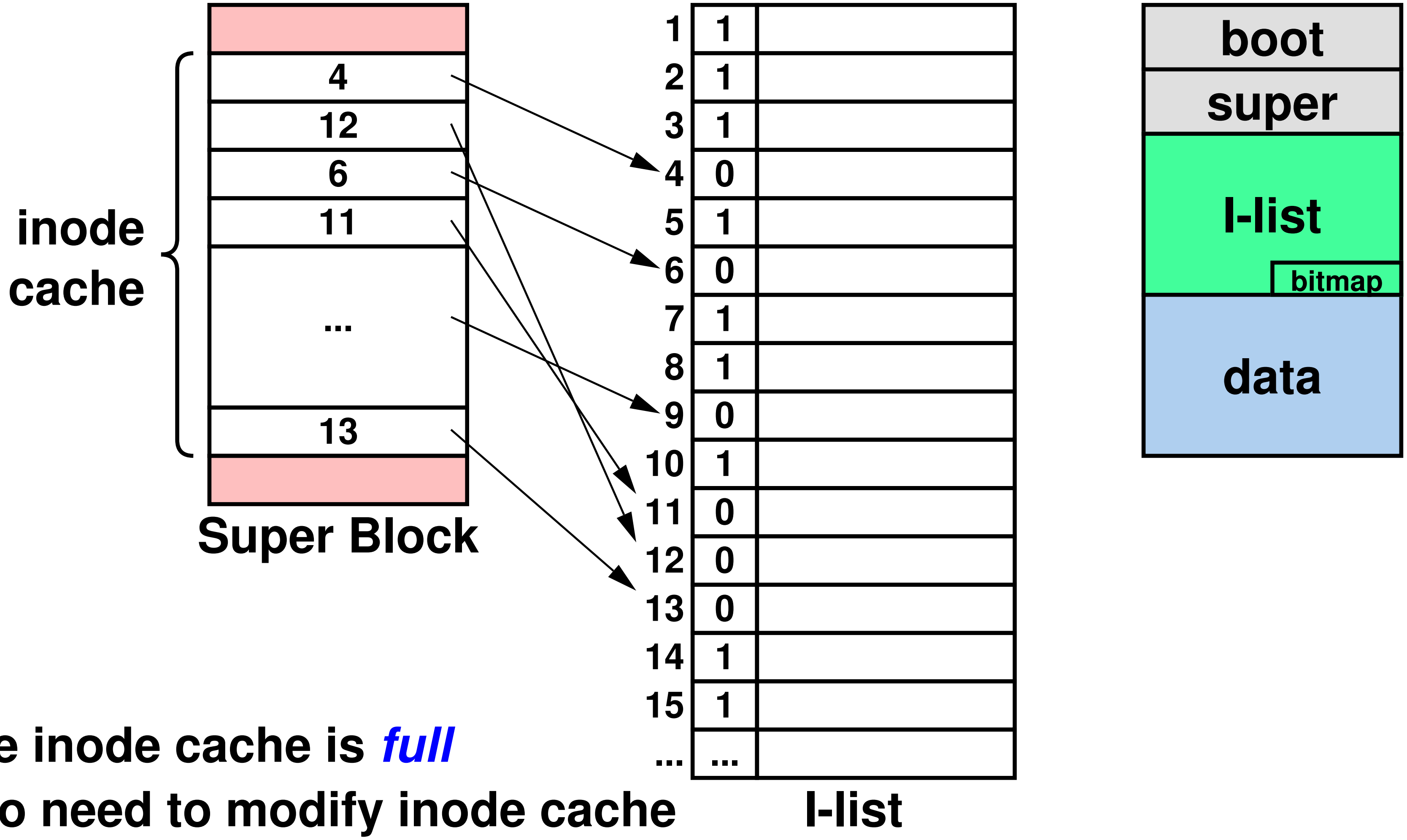
- If the inode cache is ***not full***
  - add inode number to inode cache
  - update bitmap to indicate that the inode is now free
  - update "low water mark" in super block if necessary
    - ex: free inode 8

S5FS Free Inode List: Deallocation - Case (1)



- If the inode cache is ***not full***
  - add inode number to inode cache
  - update bitmap to indicate that the inode is now free
  - update "low water mark" in super block if necessary
    - ex: free inode 8

S5FS Free Inode List: Deallocation - Case (2)



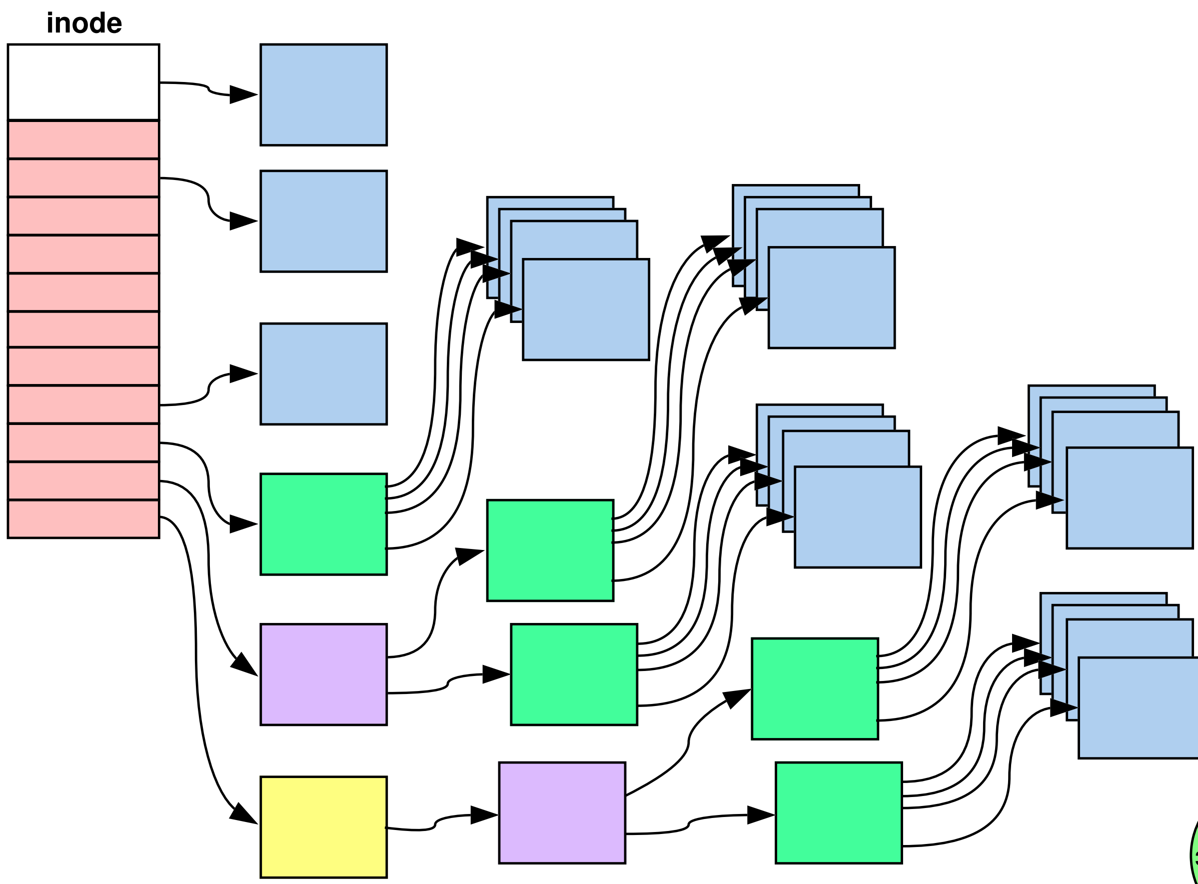
- If the inode cache is ***full***
  - no need to modify inode cache
  - update bitmap to indicate that the inode is now free
  - update "low water mark" in super block if necessary

S5FS Summary

- To *create a file*
  - get a *free inode*
    - update *i-list*, *inode cache*, and "low water mark"
- *First write to a file*
  - get a *free block*
    - update *free list*
- More writes to the file
  - may need to get a free block and update free list
- To *delete a file*
  - add disk block(s) to *free list*
  - mark inode free in *i-list*, update *inode cache* and "low water mark"

S5FS Summary

- In designing a file system, one tries to ***minimize*** the number of ***disk operations***
  - read vs. write
  - sequential access vs. random access
    - S5FS gives O(1) number of disk operations for random access

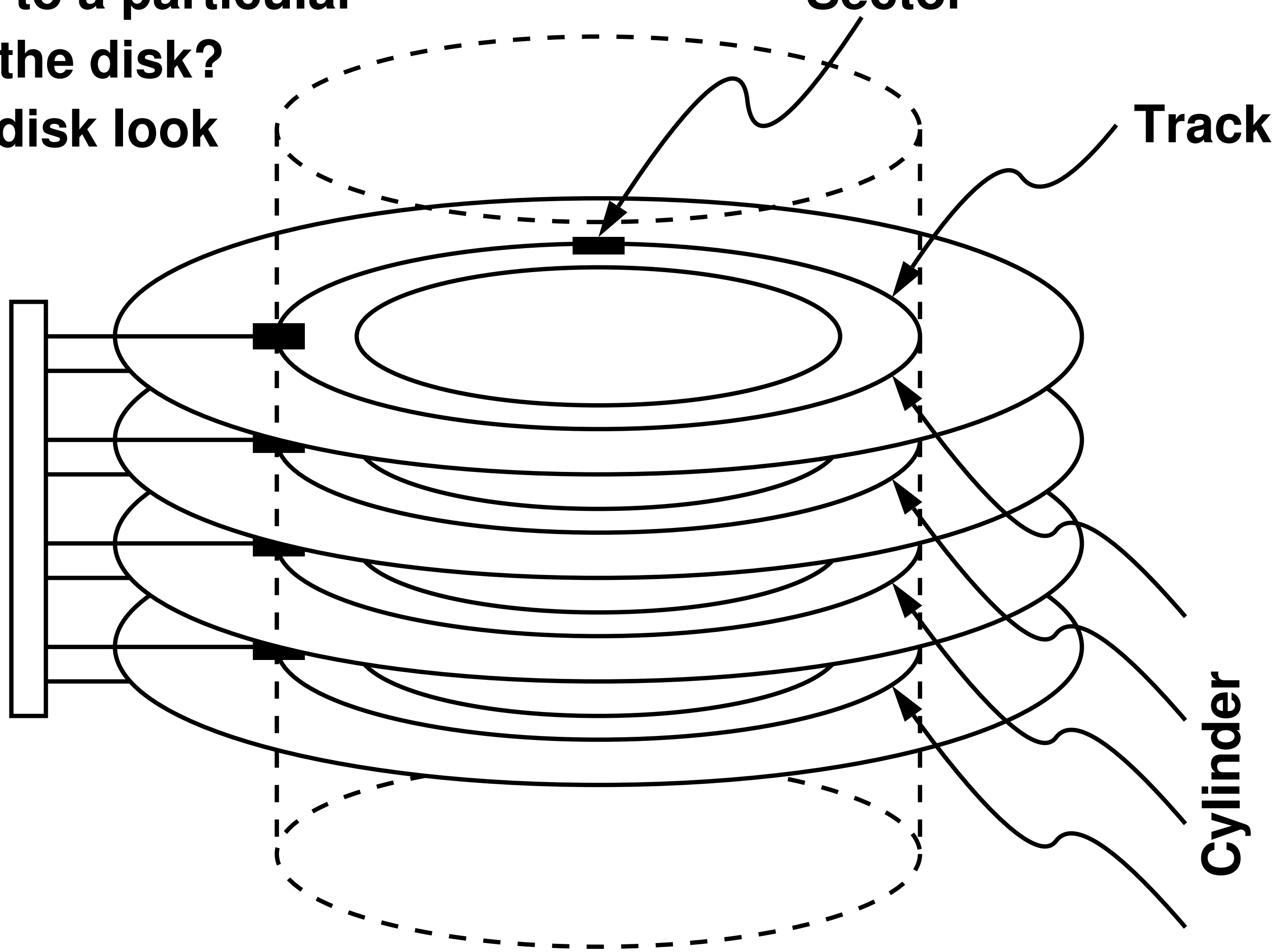


6.1 The Basics of File Systems

- UNIX's S5FS
- *Disk Architecture*
- Problems with S5FS
- Improving Performance

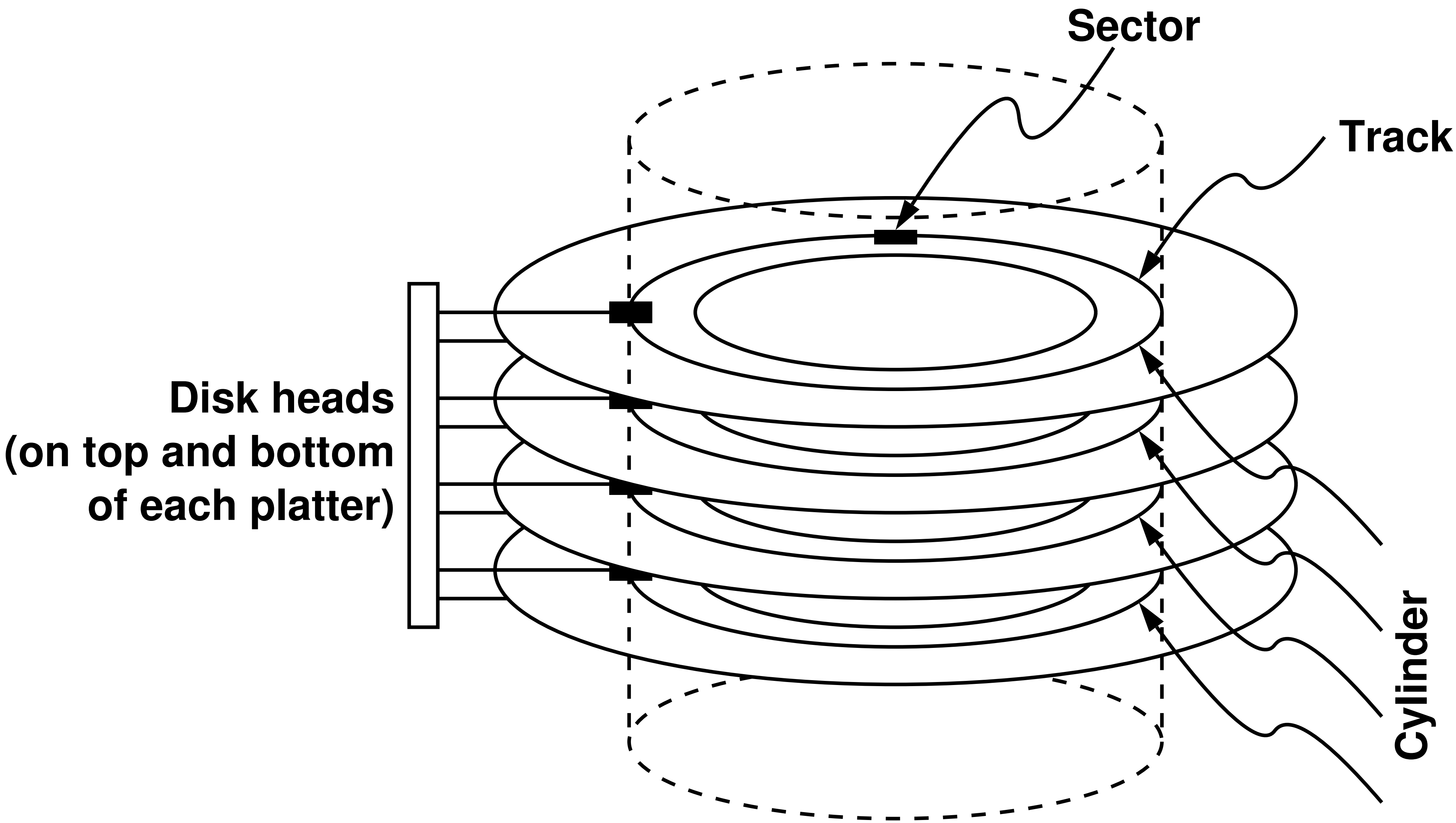
Disk Architecture

- How do you get to a particular byte of data on the disk?
  - what does a disk look like?



- Clearly, this looks nothing like the S5FS layout
  - or any logical file system layout

Disk Architecture



- Smallest addressable unit is a *sector*
  - disk address = (*head/surface#*, *cylinder/track#*, *sector#*)

Rhinopias Disk Drive

Rotation speed	10,000 RPM
Number of surfaces	8
Sector size	512 bytes
Sectors/track	500-1000; 750 average
Tracks/surface	100,000
Storage capacity	307.2 billion bytes
Average seek time	4 milliseconds
One-track seek time	.2 milliseconds
Maximum seek time	10 milliseconds

- ***Disk access time:*** time to copy a sector from disk to controller
  - *access time = seek time + rotational latency + data transfer time*
    - some people would use the term "response time" to mean "access time", but we should avoid the use of the term "response time"