File System Implementation Issues

- What data/metadata is stored in the device?
- How is that information arranged?
- How is that information used by the OS during open(), read(), write(), close(), etc.
- What happens upon crash or outage?
- What details are left to the device itself?

File System Implementation Issues

- What data/metadata is stored in the device?
- How is that information arranged?
- How is that information used by the OS during open(), read(), write(), close(), etc.
- What happens upon crash or outage?
- What details are left to the device itself?

It all depends on the fs implementation

Disk Space Management

How to keep track of allocated vs free blocks

What strategy to use for allocating free blocks to a file

Keeping Track of Free Blocks

Approach #1:
- Keep a bitmap
- 1 bit per disk block

Approach #2
- Keep a free list

Keeping Track of Free Blocks

Approach #1:

- Keep a bitmap
- 1 bit per disk block

  Example:

  - 1 KB block size
  - 16 GB Disk $\Rightarrow$ 16M blocks = 2^{24} blocks

  Bitmap size = 2^{24} bits $\Rightarrow$ 2048 blocks

  - 1/8192 space lost to bitmap

Approach #2

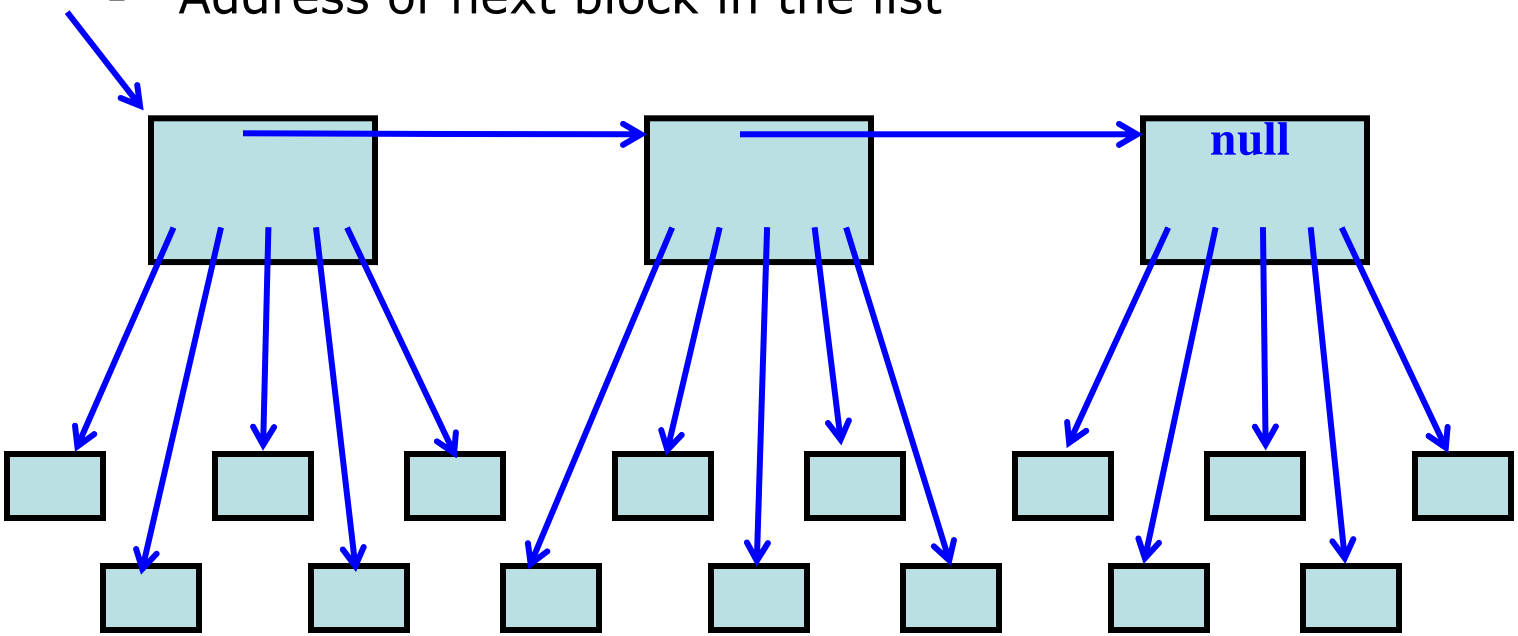
- Keep a free list

List of Free Disk Blocks

Linked list of free blocks

Each list block on disk holds

- A bunch of addresses of free blocks
- Address of next block in the list



Free List of Disk Blocks

Two kinds of blocks:

- Free Blocks
- Blocks containing pointers to free blocks

Always keep one block of pointers in memory for fast allocation and freeing

- Ideally this block will be partially full

Comparison: Free List vs Bitmap

Goal: keep all the blocks in one file close together

Comparison: Free List vs Bitmap

Goal: keep all the blocks in one file close together

Free Lists:

- Free blocks could be anywhere
- Allocation comes from (almost) random location

Comparison: Free List vs Bitmap

Goal: keep all the blocks in one file close together

Free Lists:

- Free blocks could be anywhere
- Allocation comes from (almost) random location

Bitmap:

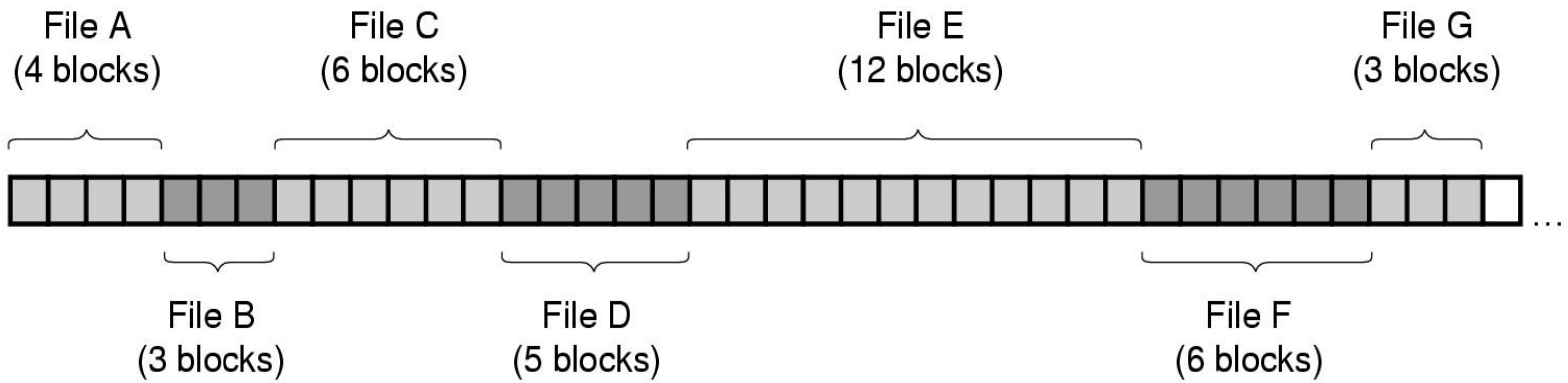
- Much easier to find a free block "close to" a given position
- Bitmap implementation:
  - Easier to keep entire bitmap in memory

Allocation Strategies

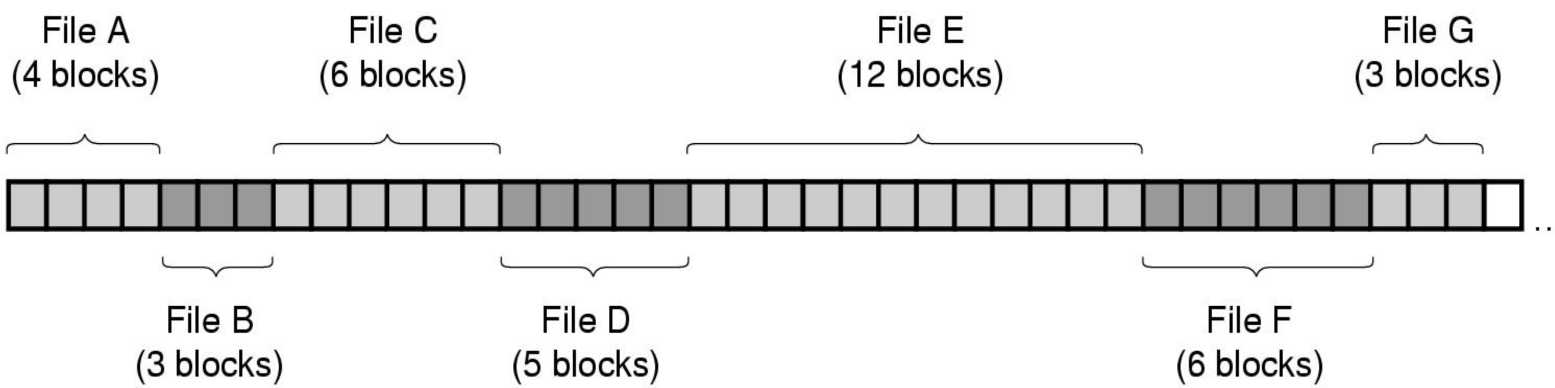
Determining which blocks make up a file:

- Contiguous allocation
- Linked allocation
- FAT file system
- Unix I-nodes

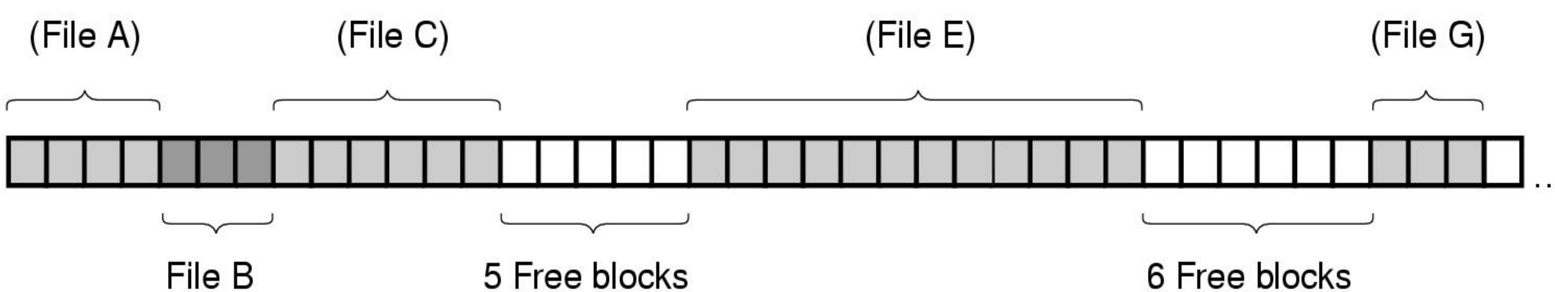
Contiguous Allocation



Contiguous Allocation



After deleting D and F...



Contiguous Allocation

Advantages:

- Simple to implement (Need only starting sector & length of file)
- Performance is good (... for sequential reading)

Contiguous Allocation

Advantages:

- Simple to implement (Need only starting sector & length of file)
- Performance is good (... for sequential reading)

Disadvantages:

- After deletions, disk becomes fragmented
- Will need periodic compaction (time-consuming)
- Will need to manage free lists
- If new file put at end of disk... no problem
- If new file is put into a “hole” we must know maximum file size *... at the time it is created!*

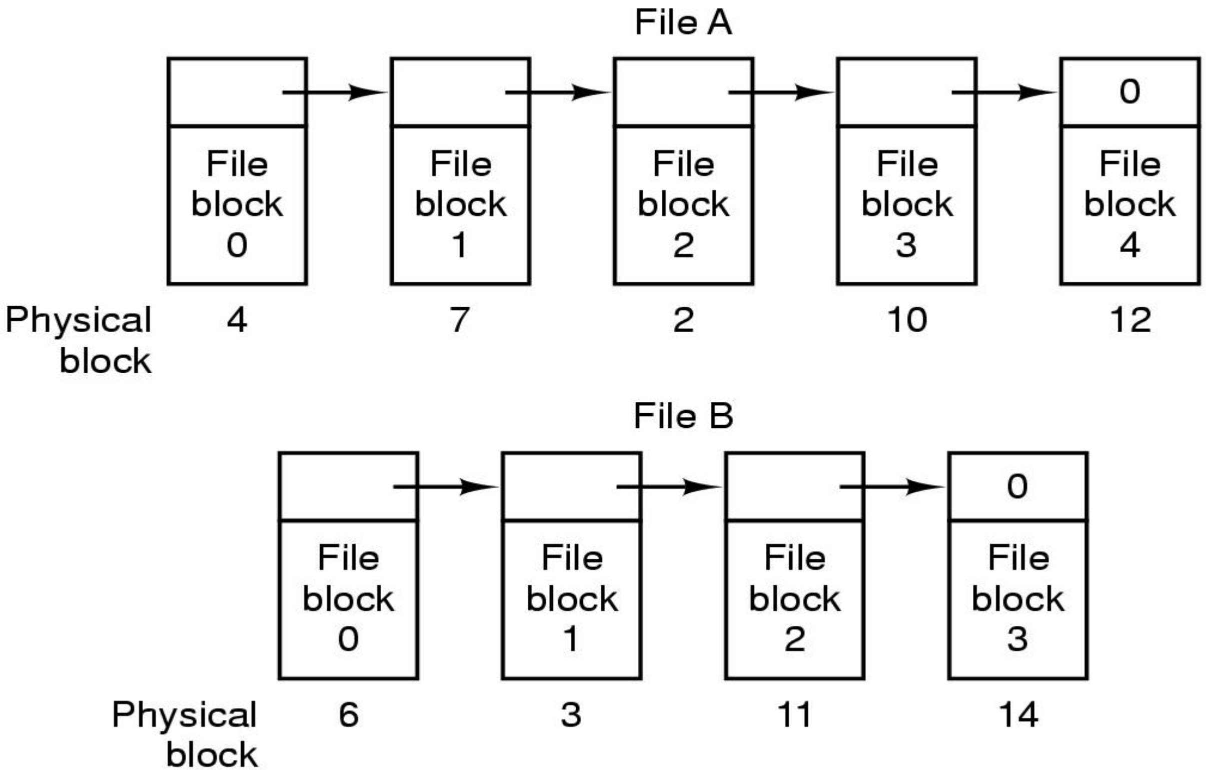
Contiguous Allocation

Good for Write-Once storage devices

- All file sizes are known in advance
- Files are never deleted

- e.g., backup storage devices

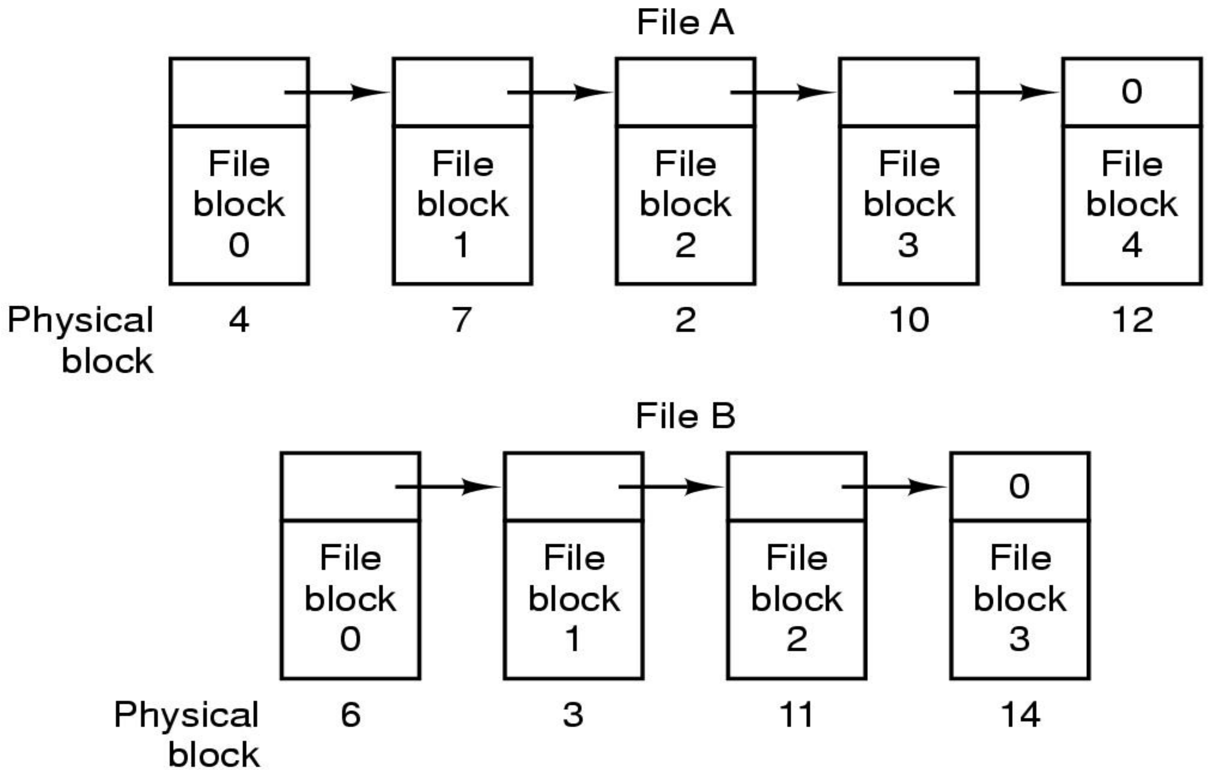
Alternative: Linked List Allocation



Each file is a sequence of blocks

First word in each block contains the number of the next block

Linked List Allocation



Random access into the file is slow!

File Allocation Table (FAT)

Keep the link information in a table in memory

One entry per block on the disk

Each entry contains the address of the “next” block

- End of file marker (-1)
- A special value (-2) indicates the block is free

File Allocation Table (FAT)

Random access...

- Searching the linked list is fast because it is all in memory

Directory entry needs only one number:

- The starting block number

Disadvantage:

- Entire table must be in memory all at once!
- This is a problem for large capacity file systems

File Allocation Table (FAT)

Disadvantage:

- Entire table must be in memory all at once!
- Example:

  200 GB = device capacity

  1 KB = block size

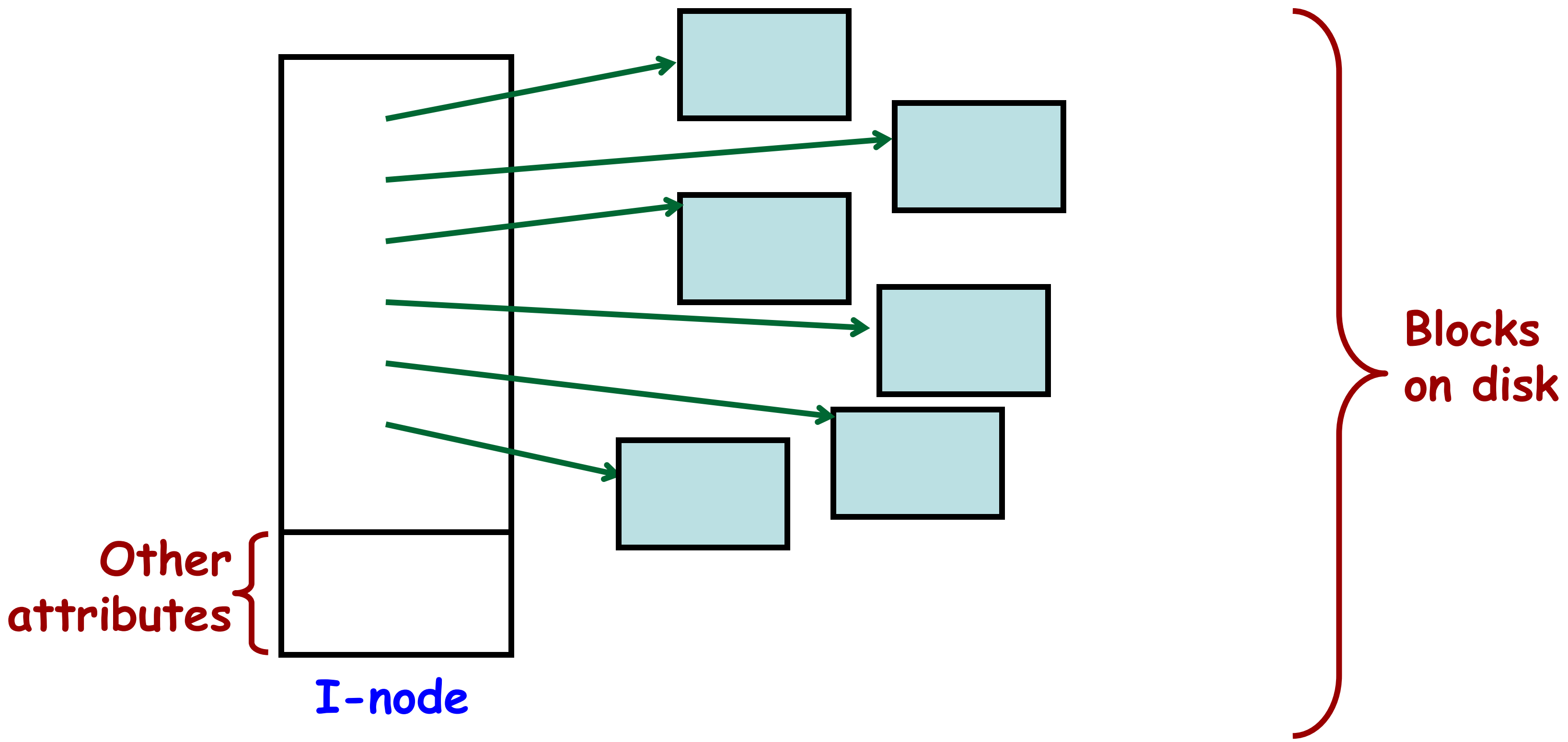
  4 bytes = FAT entry size

  800 MB of memory used to store the FAT

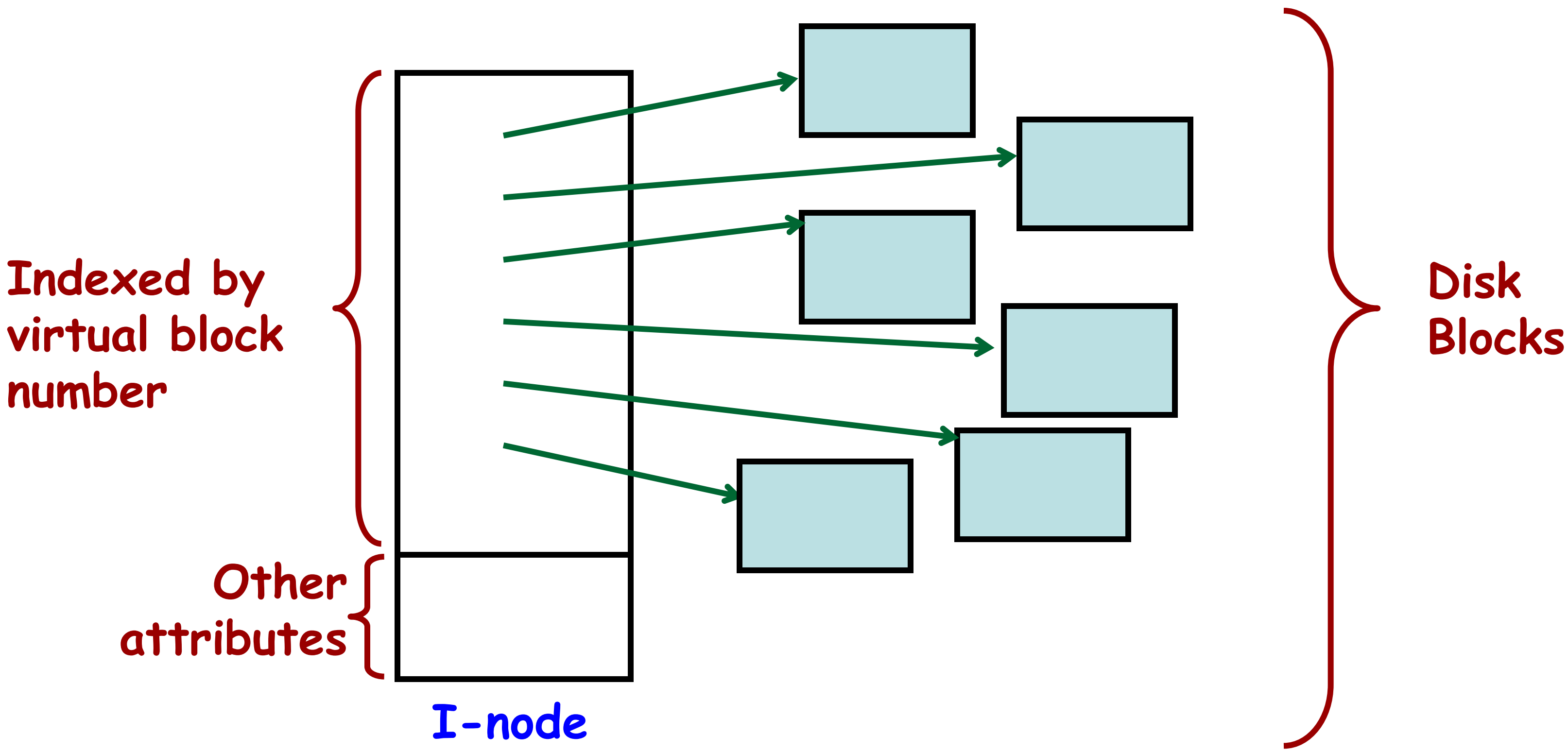
I-nodes

Each I-node ("index-node") is a structure containing info about the file

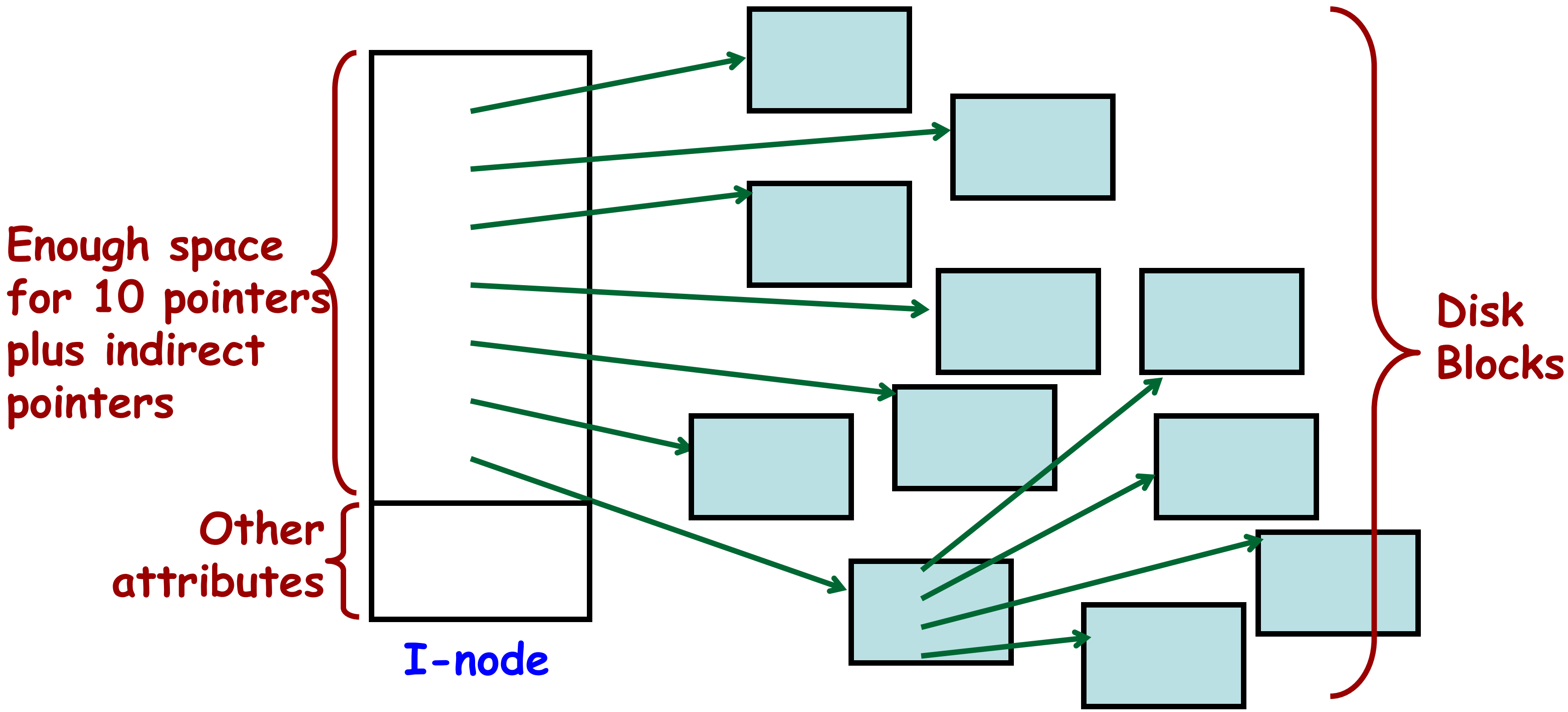
- Attributes and location of the blocks containing the file



I-nodes


Indexed by virtual block number
Other attributes
I-node
Disk Blocks

I-nodes


Enough space for 10 pointers plus indirect pointers
Other attributes
I-node
Disk Blocks

The UNIX I-node