File system internals
Tanenbaum, Chapter 4

COMP3231
Operating Systems

UNIX storage stack

Syscall interface:
```
creat
open
read
write
...
```
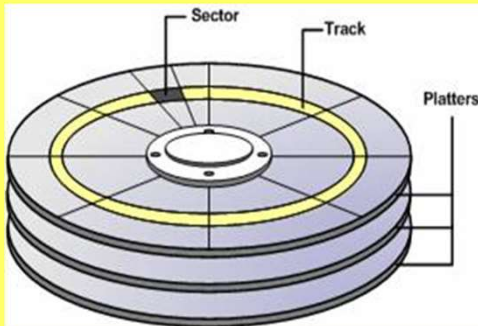
Application

FD table
OF table
VFS
FS
Buffer cache
Disk scheduler
Device driver
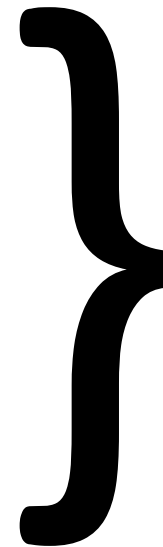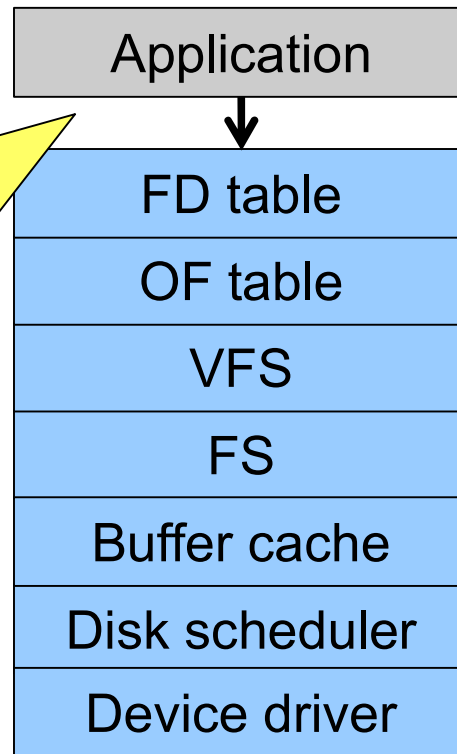
} Operating System

UNIX storage stack

Application

Hard disk platters:
 tracks
 sectors

| FD table |
| OF table |
| VFS |
| FS |
| Buffer cache |
| Disk scheduler |
| Device driver |

UNSW
SYDNEY

UNIX storage stack

Disk controller:

Hides disk geometry,
 bad sectors
Exposes linear
 sequence of blocks

0   N

Application
FD table
OF table
VFS
FS
Buffer cache
Disk scheduler
Device driver

UNSW
SYDNEY

UNIX storage stack

Application

FD table
OF table
VFS
FS
Buffer cache
Disk scheduler
Device driver
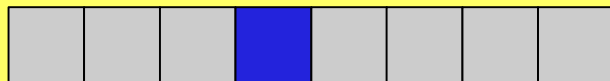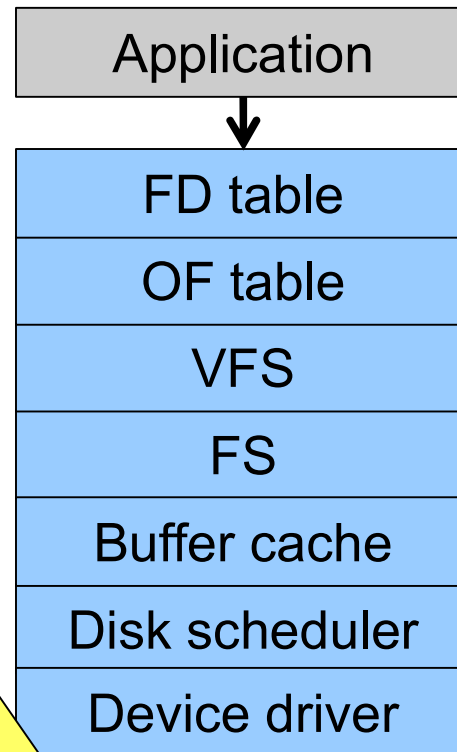
Device driver:

Hides device-specific protocol
Exposes block-device Interface (linear sequence of blocks)

0   N

UNIX storage stack

Application

FD table

OF table

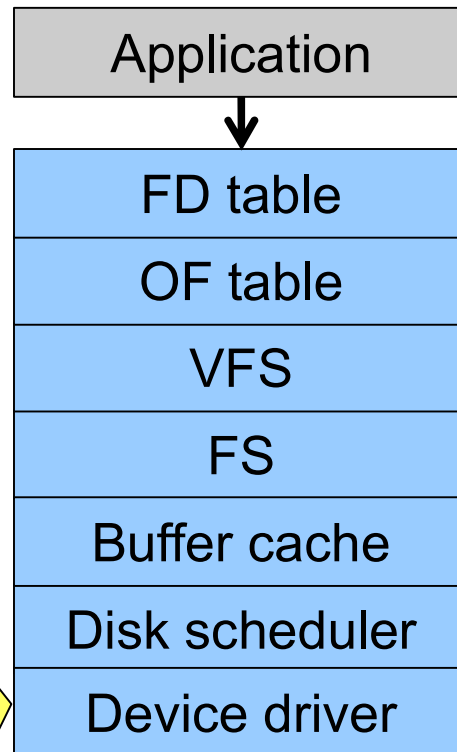VFS

FS

Buffer cache

Disk scheduler

Device driver
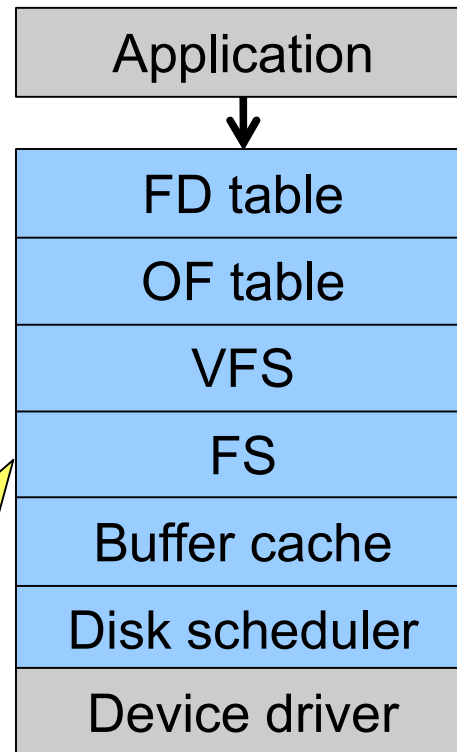
File system:

Hides physical location of data on the disk

Exposes: directory hierarchy, symbolic file names, random-access files, protection

UNSW
SYDNEY

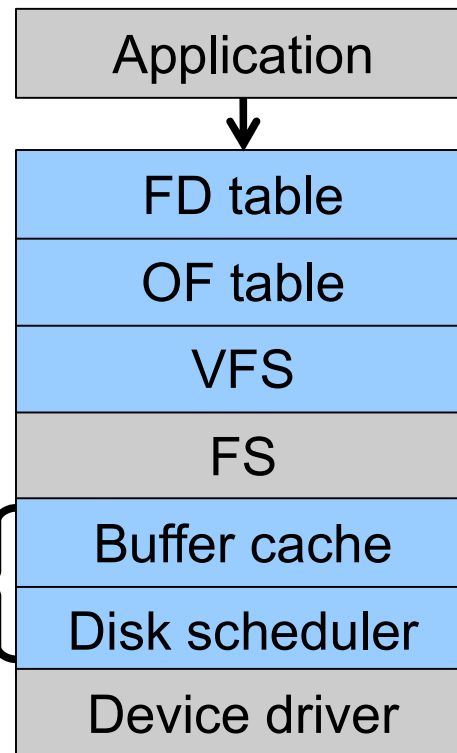# UNIX storage stack

Optimisations:

Keep recently accessed disk blocks in memory

Schedule disk accesses from multiple processes for performance and fairness

Application
FD table
OF table
VFS
FS
Buffer cache
Disk scheduler
Device driver

UNIX storage stack

Virtual FS:

Unified interface to multiple FSs

Application

FD table
OF table

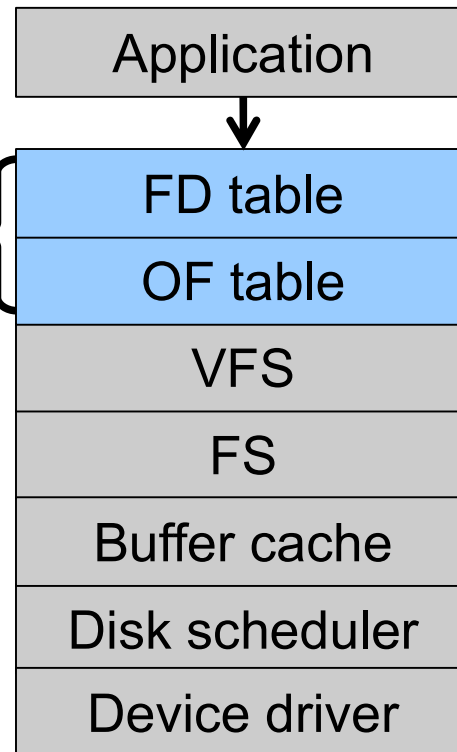| VFS	
FS	FS2
Buffer cache	
Disk scheduler	Disk scheduler
Device driver	Device driver
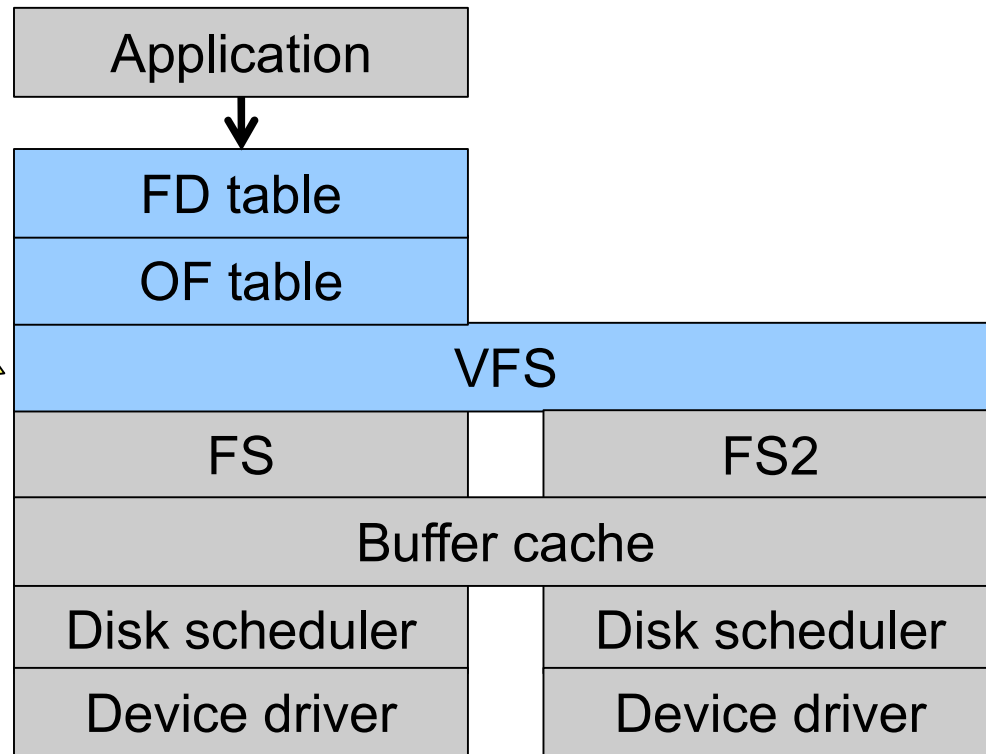
UNIX storage stack

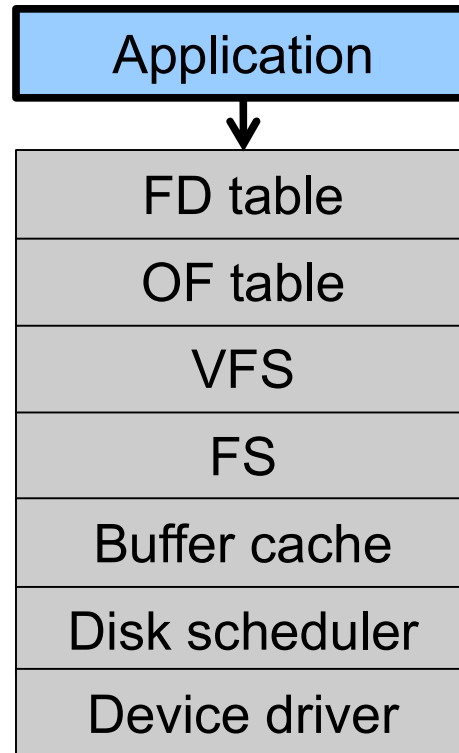File desctriptor and
Open file tables:

Keep track of files
opened by user-level
processes

Matches syscall interface
to VFS Interface

Application
FD table
OF table
VFS
FS
Buffer cache
Disk scheduler
Device driver

UNIX storage stack

Application

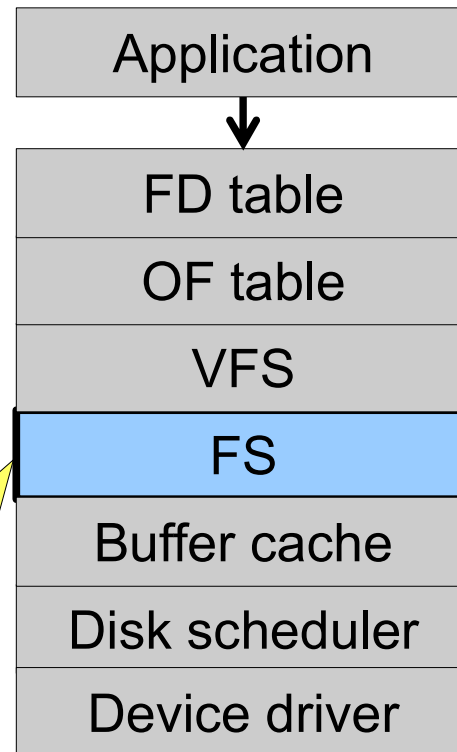| FD table |
| --- |
| OF table |
| VFS |
| FS |
| Buffer cache |
| Disk scheduler |
| Device driver |

Architecture of the OS storage stack

File system:

- Hides physical location of data on the disk

- Exposes: directory hierarchy, symbolic file names, random-access files, protection

Application
FD table
OF table
VFS
FS
Buffer cache
Disk scheduler
Device driver

UNSW
SYDNEY

Some popular file systems

- FAT16
- FAT32
- NTFS
- Ext2
- Ext3
- Ext4
- ReiserFS
- XFS
- ISO9660

- HFS+
- UFS2
- ZFS
- JFS
- OCFS
- Btrfs
- JFFS2
- ExFAT
- UBIFS

Question: why are there so many?

Why are there so many?

- Different physical nature of storage devices
  - Ext3 is optimised for magnetic disks
  - JFFS2 is optimised for flash memory devices
  - ISO9660 is optimised for CDROM
- Different storage capacities
  - FAT16 does not support drives >2GB
  - FAT32 becomes inefficient on drives >32GB
  - ZFS, Btrfs is designed to scale to multi-TB disk arrays
- Different CPU and memory requirements
  - FAT16 is not suitable for modern PCs but is a good fit for many embedded devices
- Proprietary standards
  - NTFS may be a nice FS, but its specification is closed

Assumptions

- In this lecture we focus on file systems for magnetic disks
  - Seek time
    - ~15ms worst case
  - Rotational delay
    - 8ms worst case for 7200rpm drive
  - For comparison, disk-to-buffer transfer speed of a modern drive is ~10µs per 4K block.
- Conclusion: keep blocks that are likely to be accessed together close to each other

Implementing a file system

- The FS must map symbolic file names into a collection of block addresses

- The FS must keep track of
  - which blocks belong to which files.
  - in what order the blocks form the file
  - which blocks are free for allocation

- Given a logical region of a file, the FS must track the corresponding block(s) on disk.
  - Stored in file system metadata
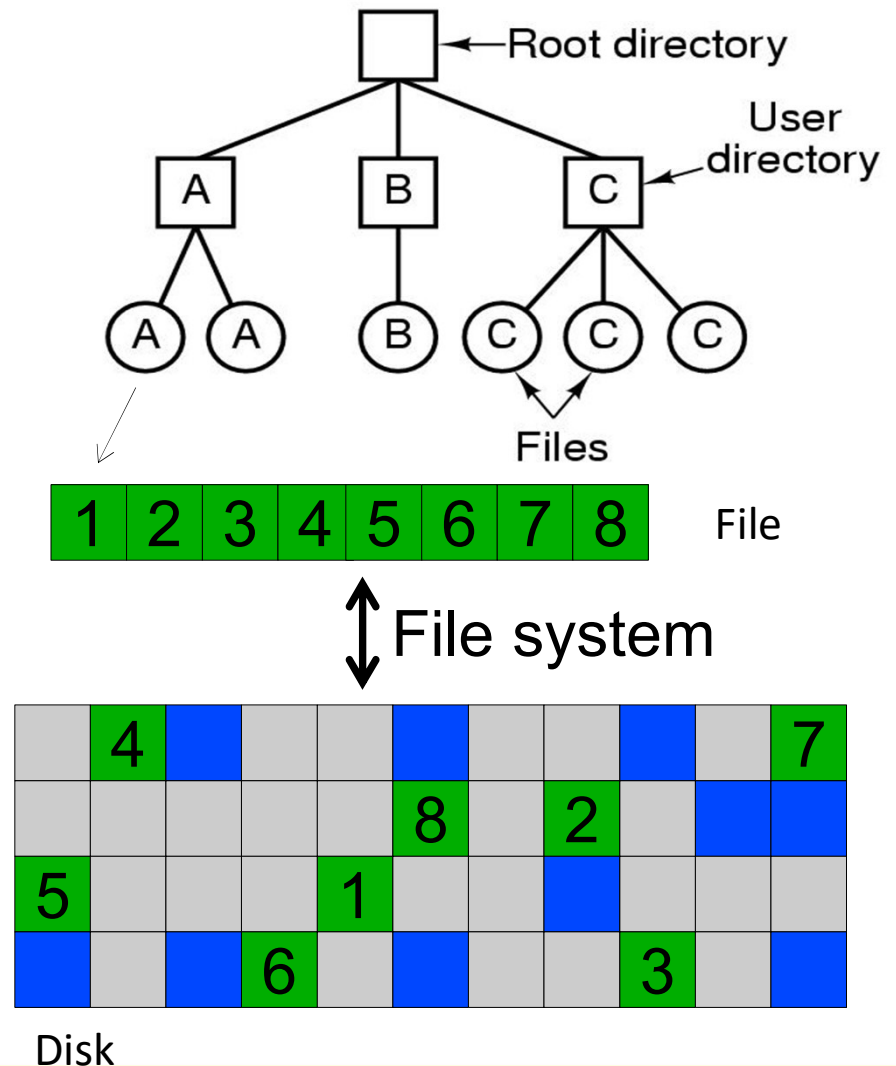
File Allocation Methods

- A file is divided into "blocks"
    - the unit of transfer to storage
- Given the logical blocks of a file, what method is used to choose were to put the blocks on disk?

File

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Disk
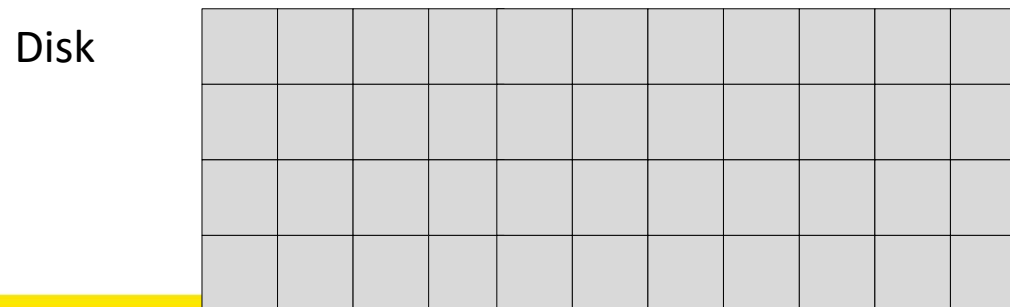
Contiguous Allocation

✓ Easy bookkeeping (need to keep track of the starting block and length of the file)

✓ Increases performance for sequential operations

✗ Need the maximum size for the file at the time of creation

✗ As files are deleted, free space becomes divided into many small chunks (external fragmentation)

Example: ISO 9660 (CDROM FS)

Metadata
(also stored
on disk)

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
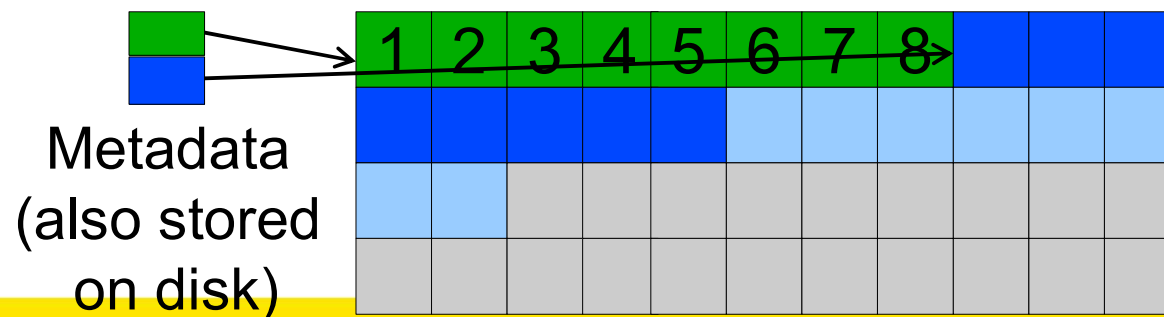
Dynamic Allocation Strategies

- – Disk space allocated in portions as needed
- – Allocation occurs in fixed-size blocks
- ✔ No external fragmentation
- ✔ Does not require pre-allocating disk space
- ✗ Partially filled blocks (internal fragmentation)
- ✗ File blocks are scattered across the disk
- ✗ Complex metadata management (maintain the collection of blocks for each file)
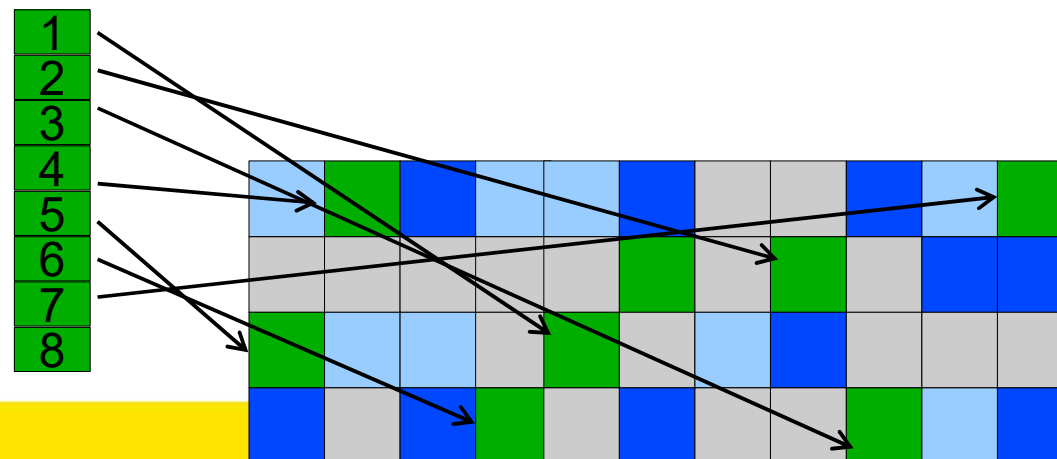
External and internal fragmentation

- **External fragmentation**
  - The space wasted external to the allocated memory regions
  - Memory space exists to satisfy a request but it is unusable as it is not contiguous

- **Internal fragmentation**
  - The space wasted internal to the allocated memory regions
  - Allocated memory may be slightly larger than requested memory; this size difference is wasted memory internal to a partition

Dynamic allocation: Linked list allocation

- Each block contains a pointer to the next block in the chain. Free blocks are also linked in a chain.
  - ✓ Only single metadata entry per file
  - ✓ Best for sequential files

Disk

1 4 2 3
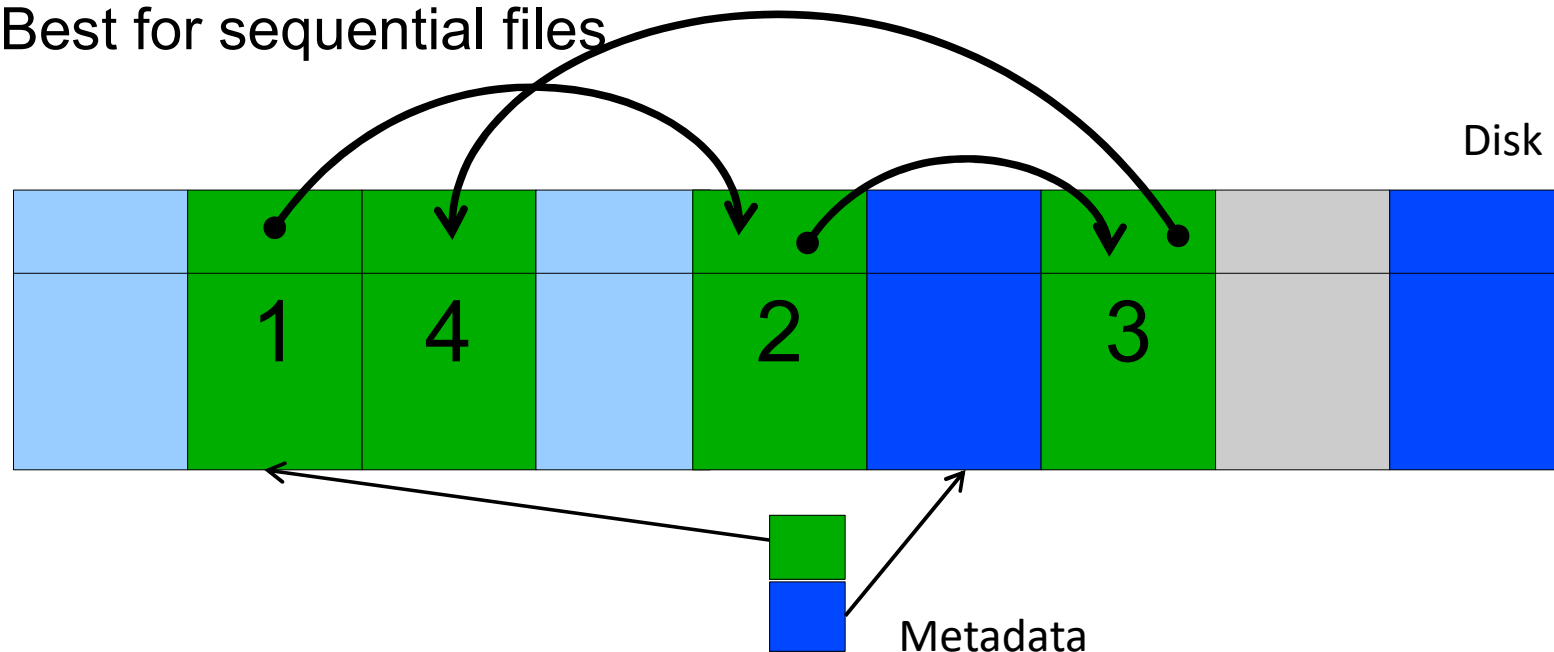
Metadata

Question: What are the downsides?

UNSW
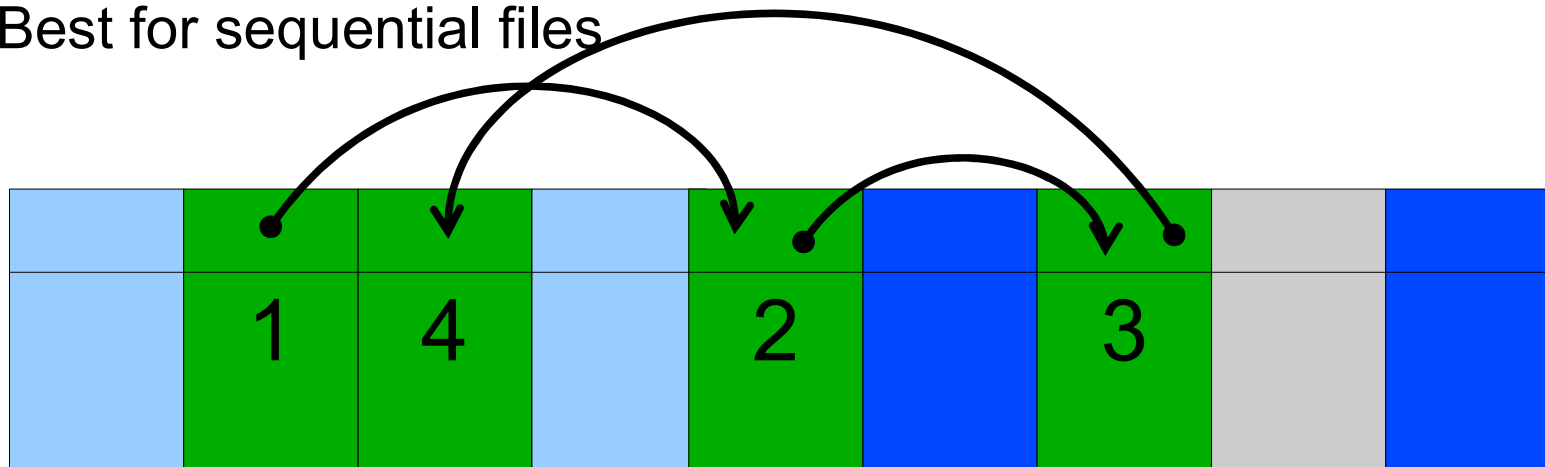SYDNEY

Linked list allocation

- Each block contains a pointer to the next block in the chain. Free blocks are also linked in a chain.
  - ✔ Only single metadata entry per file
  - ✔ Best for sequential files

  - ✗ Poor for random access
  - ✗ Blocks end up scattered across the disk due to free list eventually being randomised

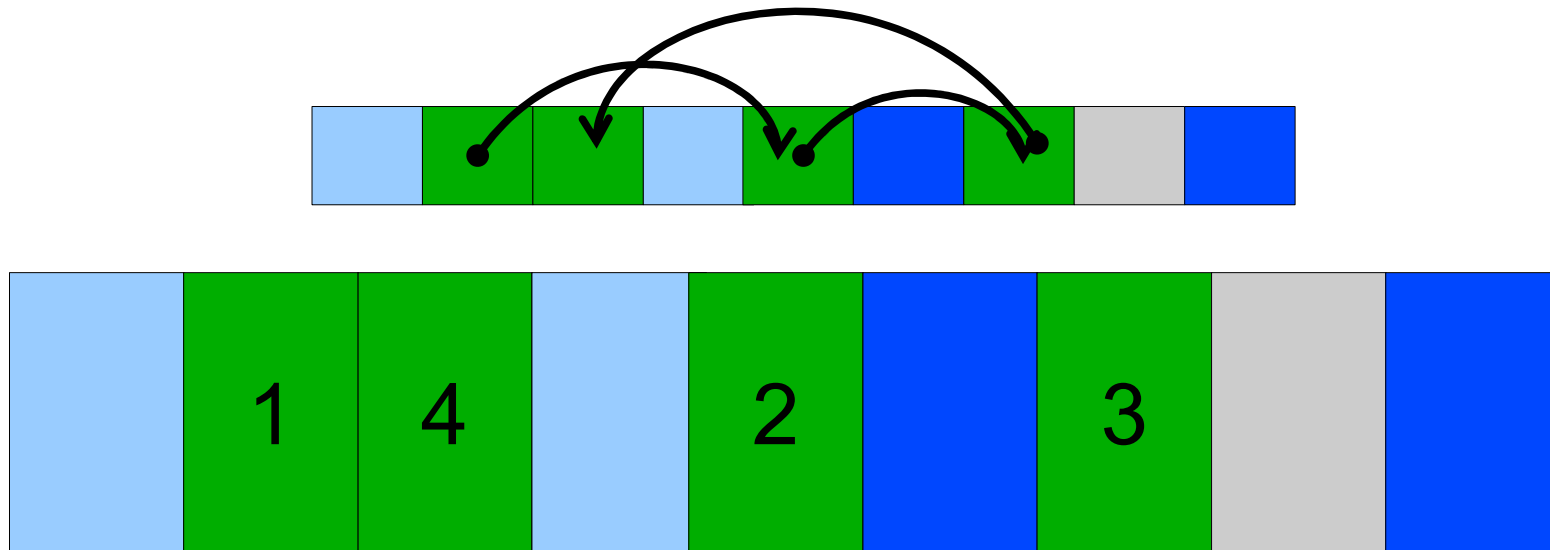# Dynamic Allocation: File Allocation Table (FAT)

- Keep a map of the entire FS in a separate table
  - A table entry contains the number of the next block of the file
  - The last block in a file and empty blocks are marked using reserved values
- The table is stored on the disk and is replicated in memory
- Random access is fast (following the in-memory list)

RAM

Disk

| 1 | 4 | 2 | 3 |

UNSW
SYDNEY
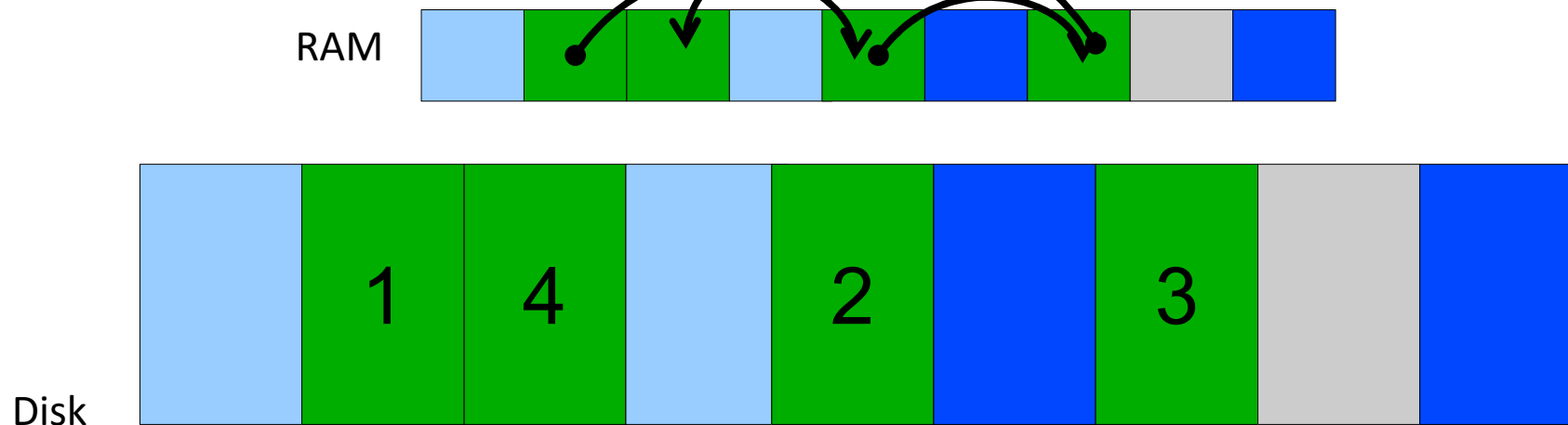
File allocation table

- Issues
  - Requires a lot of memory for large disks
    - 200GB = 200*10^6 * 1K-blocks ==>

      200*10^6 FAT entries = 800MB
  - Free block lookup is slow
    - searches for a free entry in table

File allocation table disk layout

- Examples
  - FAT12, FAT16, FAT32

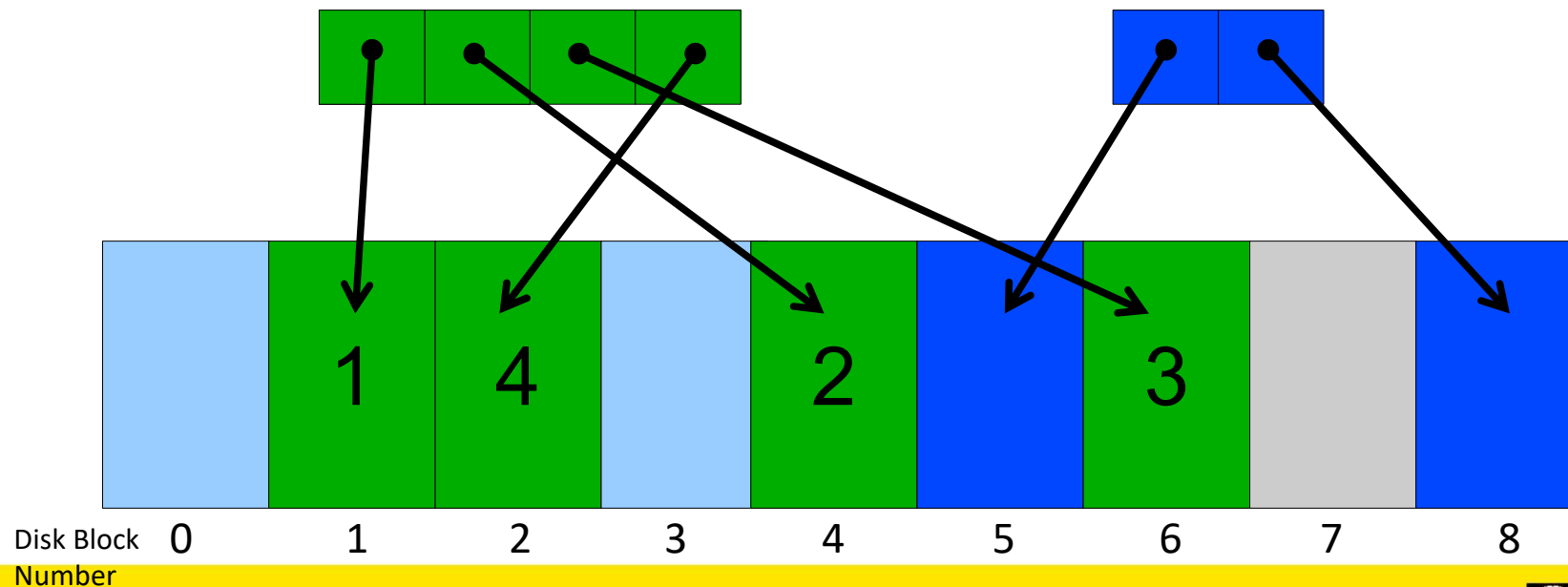reserved   FAT1   FAT2   data blocks

Two copies of FAT for
redundancy

Dynamical Allocation: inode-based FS structure

- Idea: separate table (index-node or i-node) for each file.
  - Only keep table for open files in memory
  - Fast random access
- The most popular FS structure today

Disk Block Number   0   1   2   3   4   5   6   7   8
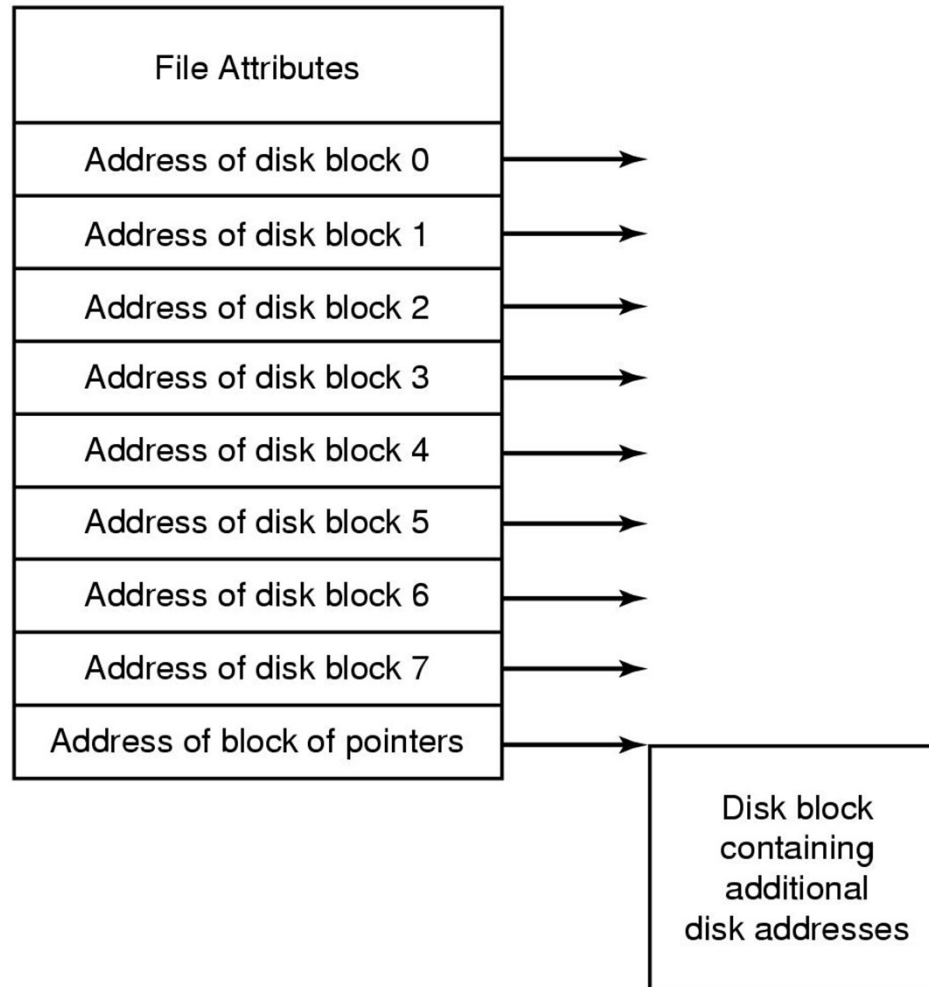
i-node implementation issues

- i-nodes occupy one or several disk areas

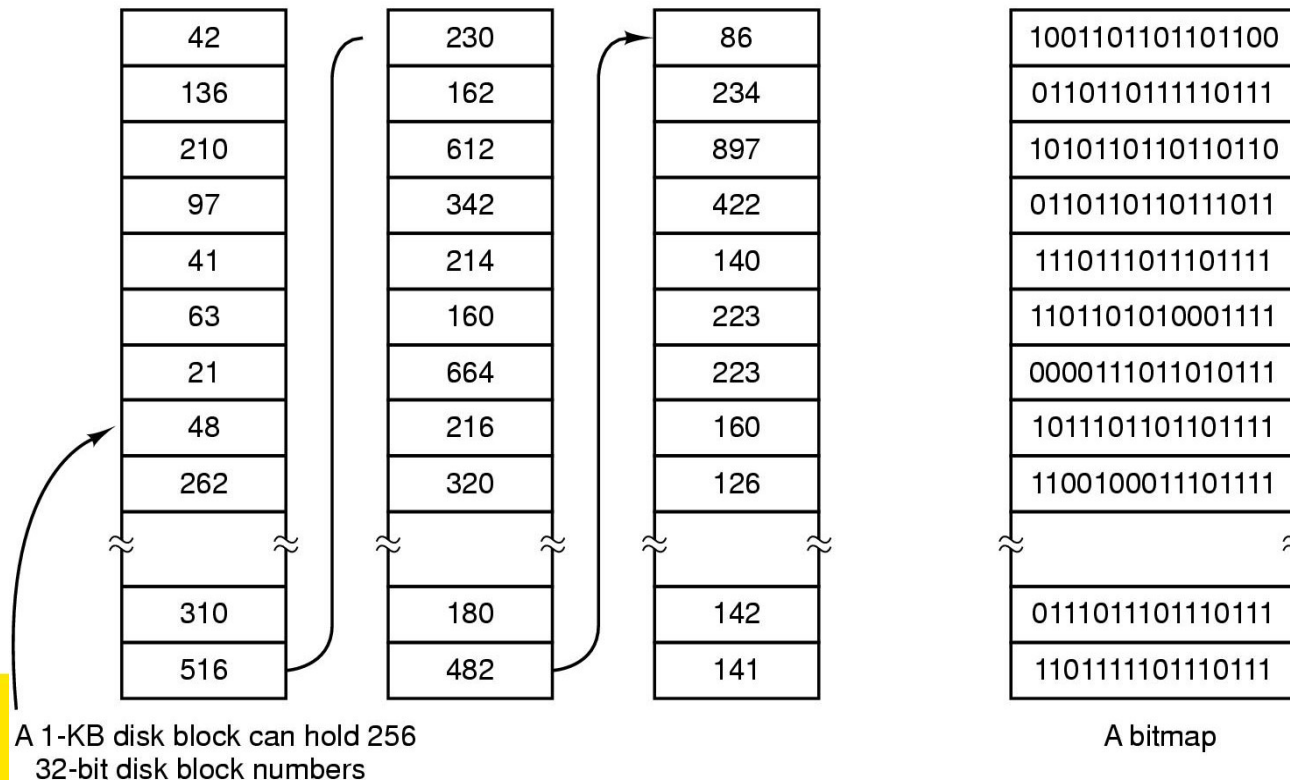i-nodes   data blocks

- i-nodes are allocated dynamically, hence free-space management is required for i-nodes

  – Use fixed-size i-nodes to simplify dynamic allocation

  – Reserve the last i-node entry for a pointer (a block number) to an extension i-node.

i-node implementation issues

i-node implementation issues

- Free-space management
  - Approach 1: linked list of free blocks in free blocks on disk
  - Approach 2: keep bitmaps of free blocks and free i-nodes on disk

42	230	86	1001101101101100
136	162	234	0110110111110111
210	612	897	1010110110110110
97	342	422	0110110110111011
41	214	140	1110111011101111
63	160	223	1101101010001111
21	664	223	0000111011010111
48	216	160	1011101101101111
262	320	126	1100100011101111
≈	≈	≈	≈
310	180	142	0111011101110111
516	482	141	1101111101110111

A 1-KB disk block can hold 256
32-bit disk block numbers

A bitmap

UNSW
SYDNEY

Free block list

- List of all unallocated blocks

- Background jobs can re-order list for better contiguity

- Store in free blocks themselves
  - Does not reduce disk capacity

- Only one block of pointers need be kept in the main memory

Bit tables

- Individual bits in a bit vector flags used/free blocks

- 16GB disk with 512-byte blocks --> 4MB table

- May be too large to hold in main memory

- Expensive to search
  - Optimisations possible, e.g. a two level table

- Concentrating (de)allocations in a portion of the bitmap has desirable effect of concentrating access

- Simple to find contiguous free space

Implementing directories

- Directories are stored like normal files
  - directory entries are contained inside data blocks
- The FS assigns special meaning to the content of these files
  - a directory file is a list of directory entries
  - a directory entry contains file name, attributes, and the file i-node number
    - maps human-oriented file name to a system-oriented name

UNSW
SYDNEY

Fixed-size vs variable-size directory entries

- Fixed-size directory entries
  - Either too small
    - Example: DOS 8+3 characters
  - Or waste too much space
    - Example: 255 characters per file name
- Variable-size directory entries
  - Freeing variable length entries can create external fragmentation in directory blocks
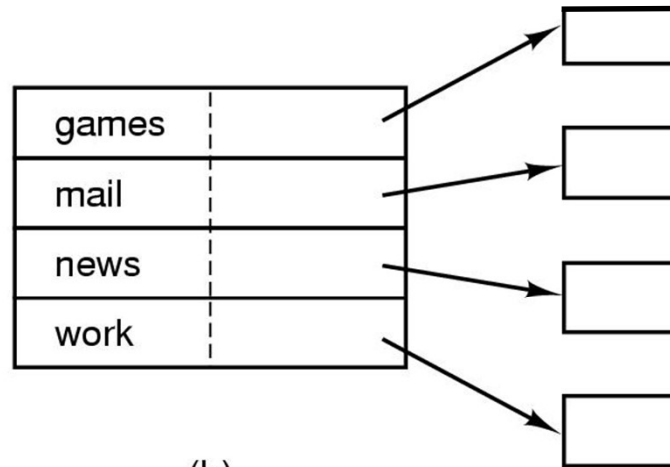   - Can compact when block is in RAM

Searching Directory Listings

- Locating a file in a directory
  - Linear scan
    - Implement a directory cache in software to speed-up search
  - Hash lookup
  - B-tree (100's of thousands entries)

Storing file attributes

(a)disk addresses and attributes in directory entry

–FAT

(b) directory in which each entry just refers to an i-node

–UNIX

Trade-off in FS block size

- File systems deal with 2 types of blocks
    - Disk blocks or sectors (usually 512 bytes)
    - File system blocks 512 * 2^N bytes
    - What is the optimal N?

- Larger blocks require less FS metadata
- Smaller blocks waste less disk space (less internal fragmentation)
- Sequential Access
    - The larger the block size, the fewer I/O operations required
- Random Access
    - The larger the block size, the more unrelated data loaded.
    - Spatial locality of access improves the situation
- Choosing an appropriate block size is a compromise

UNSW
SYDNEY