# Operating system concepts

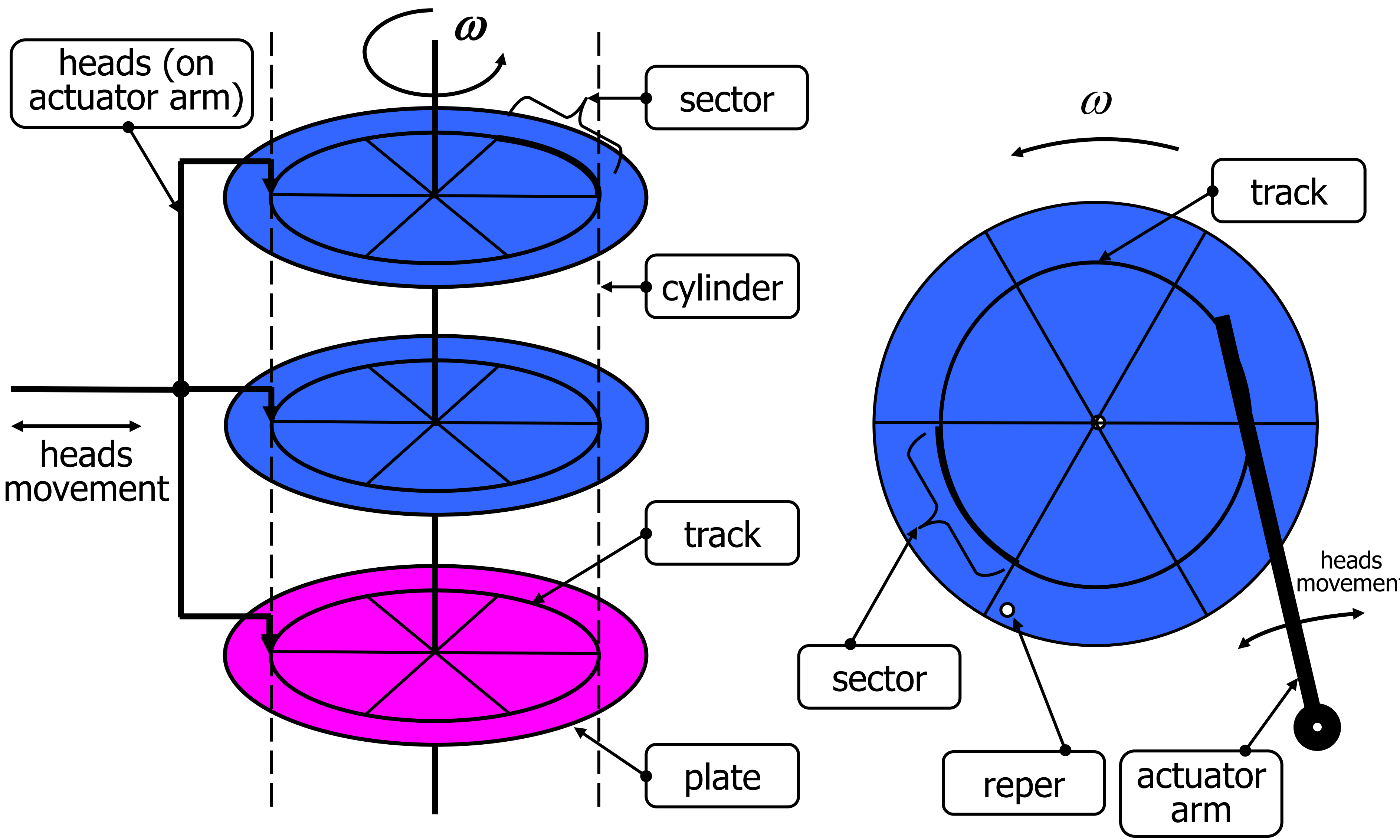
***File Systems***

# Hard disk

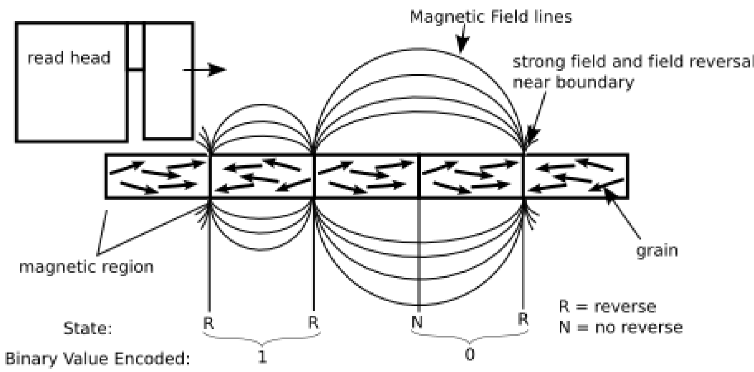
Hard disk is used for:

- permanent data storage
    - when system goes down, its data are saved on disk
    - when system boots, operating system is loaded from disk
    - (compiled) programs and data are permanently saved on disk, from there loaded into memory and stored back
- secondary memory
    - disk is used as auxiliary memory (in addition to RAM)
    - due to significant difference in speed (nanoseconds vs milliseconds), special techniques are used (described previously in memory management)
- How are data stored on disk?
    - physical data organization on disk ?
    - logical representation (as used from operating system) ?

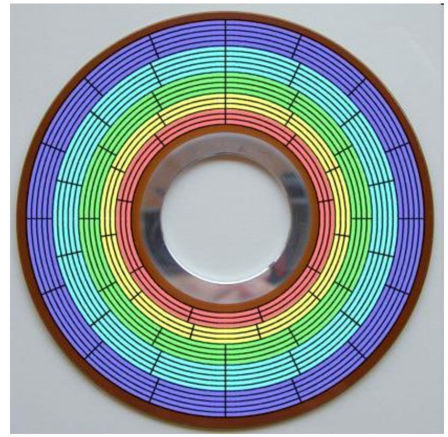
# Physical data organization on disk (in theory)



# Physical data organization on disk (in practice)

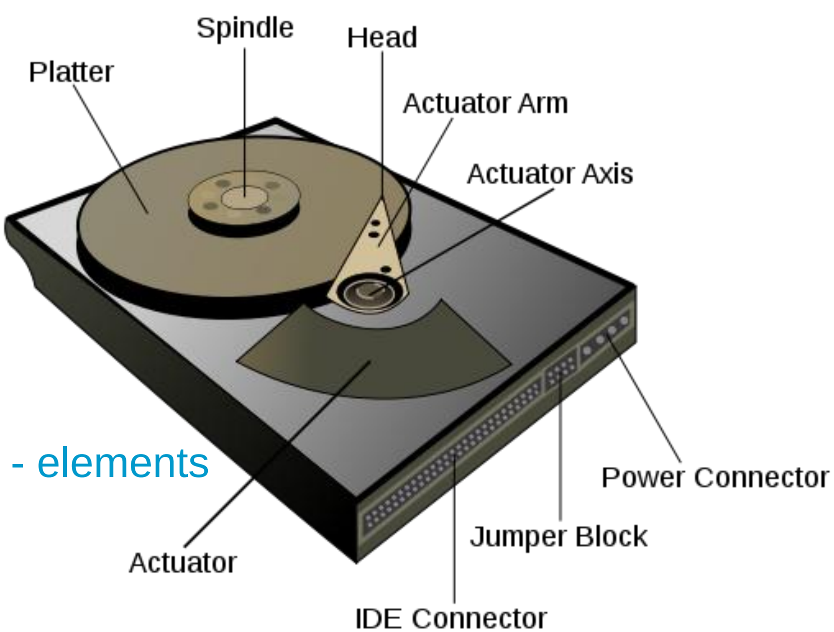



Magnetic plate – storing bits



Tracks and sectors

Disk - uncovered




Disk - elements

Disk - detail

# Addressing data on disk

- Data unit = sector
  - sector sizes: 512 B, 1024 B
- Sector's physical "address":
  - plate identifier
  - track identifier
  - sector identifier
- Sector's logical address:
  - electronic that manipulates with mechanical hard drive components "translates" physical address into logical one
  - all sectors represented as a linear array of sectors/blocks – linear "address" sector space
    - *Logical block addressing* – LBA

# Hard drives characteristics (typical)

- Capacity: ~100 GB to 2 TB
- Physical size 3.5”, 2.5”, 1.8”, 1”, 0.85”
- Rotation: ~5000 rpm to 15000 rpm (5400, 7200, 10000)
- Data transfer rate: ~70 MB/s
- Seek time: 2 to 15 ms; typical ~9 ms
  - time to move heads over 1/3 of tracks
  - “average” random disk access head movement

# Files

- Data on hard disk (and other media) are organized into ***files***
- File: set of data/information that are united somehow
- File may contain:
  - program (code and data)
    - e.g. executable file or script (.exe; .out; .bat; .sh; …)
    - e.g. extensions, dynamic library (.dll; .so; …)
  - data (input or output)
    - documents (word, text files, HTML, …)
    - multimedia (pictures, videos, music, …)
    - …
  - other
    - OS data (like pagefile), …

# File system

- How to organize files on disk?
  - Physical placement: where (in which sectors)?
  - Logical organization: directories
  - Access, security, fragmentation, … ?

  - ➔ use of a File System
- File system defines how to place data on disk and how to retrieve it
  - File table – data that defines a particular disk, its files and free space
    - each file has its *descriptor* in the file table
- Operating system uses file system and provides operations like:
    - “create file”, “open file”, “close file”, “delete file”, “move file”
    - “write to file”, “read from file”

# File descriptor

- Every file on disk has a descriptor

- Descriptor contents (mainly):
  - file name
  - directory (logical placement)
  - file type, file size
  - creation time, modification time, last access time
  - owner information
  - security information
  - access rights
  - …
  - **data placement description** (in which sectors/blocks)
    - data unit – block (cluster)
      - block size: 1, 2, 4, 8, 16, … consecutive sectors

# File system examples (file placement descr.)

- Disk may be divided into partitions
  - Every partition is managed separately (has its own file system)
    - e.g. part1: from block 0 to 10000, part2: from 10001 to 20000

- NTFS
- UNIX

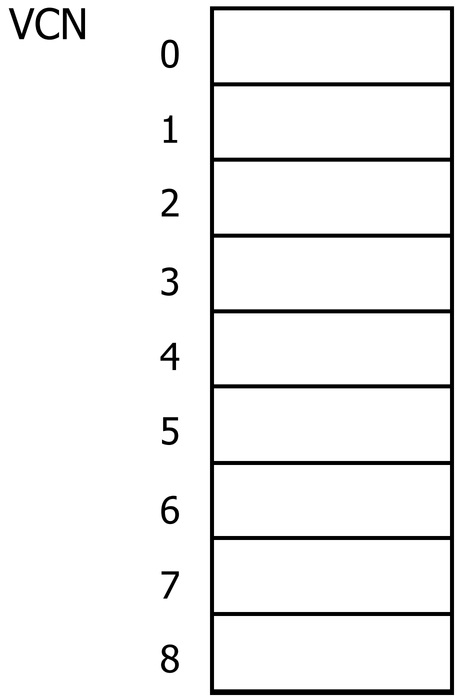
# File system examples – NTFS

- MFT (Master File Table)
  - every file has a file descriptor in the MFT (even MFT itself)

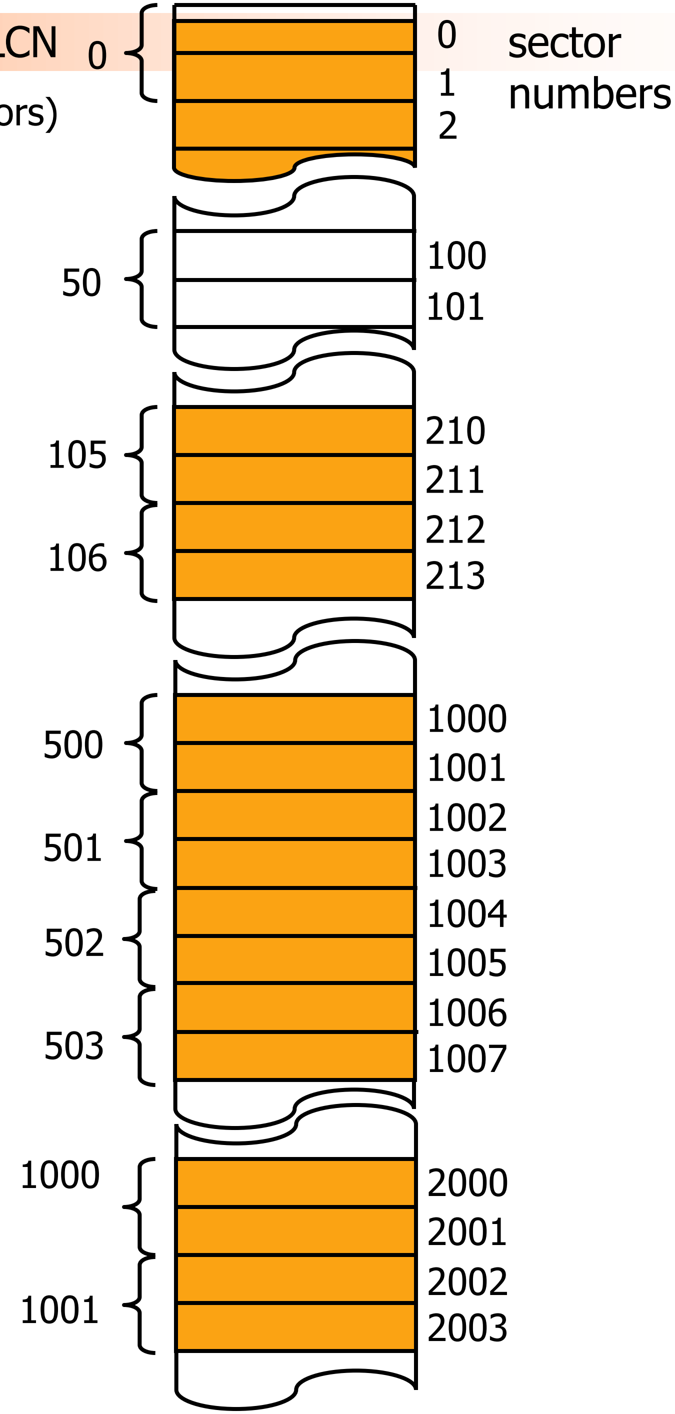
- Cluster numbering in NTFS:
  - LCN – Linear Cluster Number
    - partition is divided into clusters (blocks), e.g. starting with LCN=0

  - VCN – Virtual Cluster Number
    - each file uses a number of blocks
    - VCN represents virtual address *inside the file*: first part of the file is at VCN=0, second in VCN=1, …

  - Mapping from VCN to LCN is defined in the file descriptor




(1 block = 2 sectors)

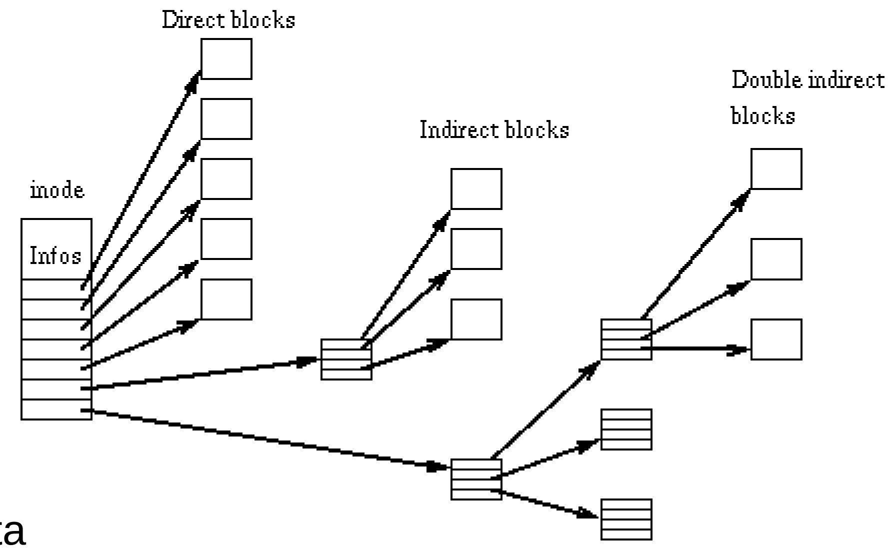



Translation from VCN to LCN (in file descriptor)

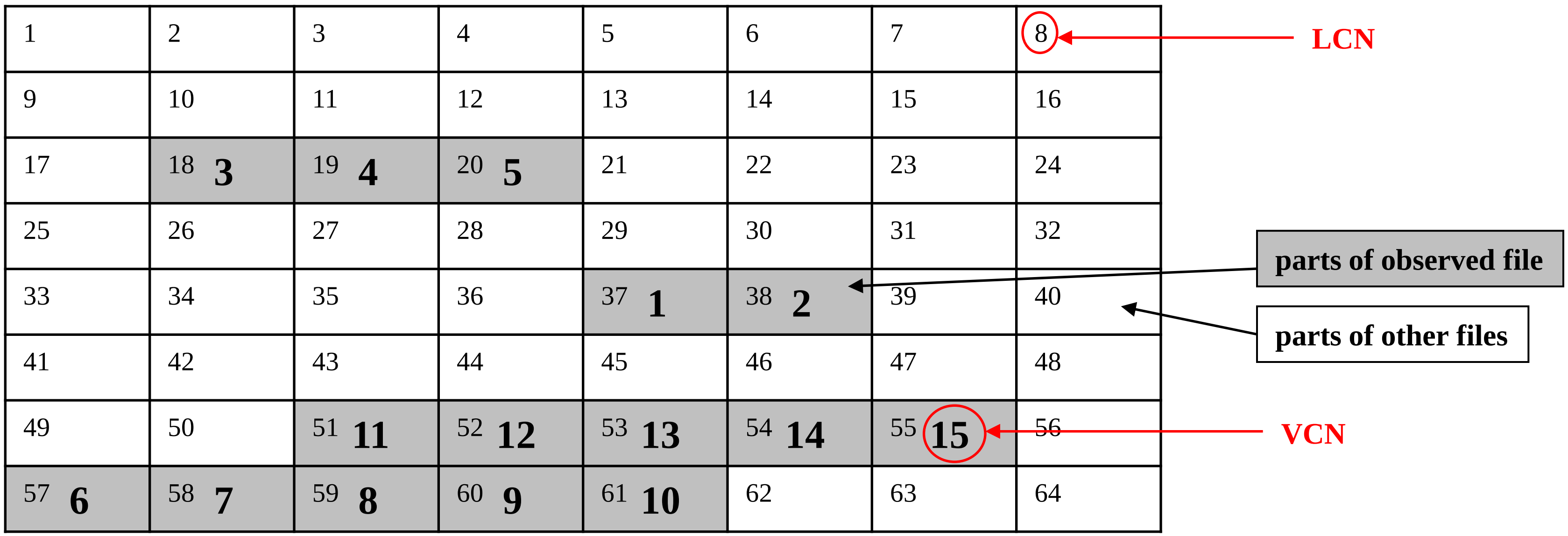
| VCN | Starting LCN | blocks |
|---|---|---|
| 0 | 50 | 1 |
| 1 | 500 | 4 |
| 5 | 105 | 2 |
| 7 | 1000 | 2 |

# File system examples – UNIX i-node

- File descriptor = i-node
- For referencing file clusters, there are a dozen pointers in :
  - Ten (or twelve) direct pointers
    - pointers that directly point to blocks of the file's data
    - e.g. 5th pointer points to block on disk that holds 5th data block of file (file is divided into blocks)
  - one singly indirect pointer
    - a pointer that points to a block of pointers that then point to blocks of the file's data
  - one doubly indirect pointer
    - a pointer that points to a block of pointers that point to other blocks of pointers that then point to blocks of the file's data
  - one triply indirect pointer
    - a pointer that points to a block of pointers that point to other blocks of pointers that point to other blocks of pointers that then point to blocks of the file's data (*not shown here*)



# File system examples – NTFS & UNIX




**NTFS**

| VCN | LCN | blocks |
|---|---|---|
| 1 | 37 | 2 |
| 3 | 18 | 3 |
| 6 | 57 | 5 |
| 11 | 51 | 5 |

**"UNIX"** – first 13 pointers (in file descriptor)

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 37 | 38 | 18 | 19 | 20 | 57 | 58 | 59 | 60 | 61 | (x) | | |

(x) →

| | | | | | | |
|---|---|---|---|---|---|---|
| 51 | 52 | 53 | 54 | 55 | (unused pointers) | |

# File system examples – FAT (idea)

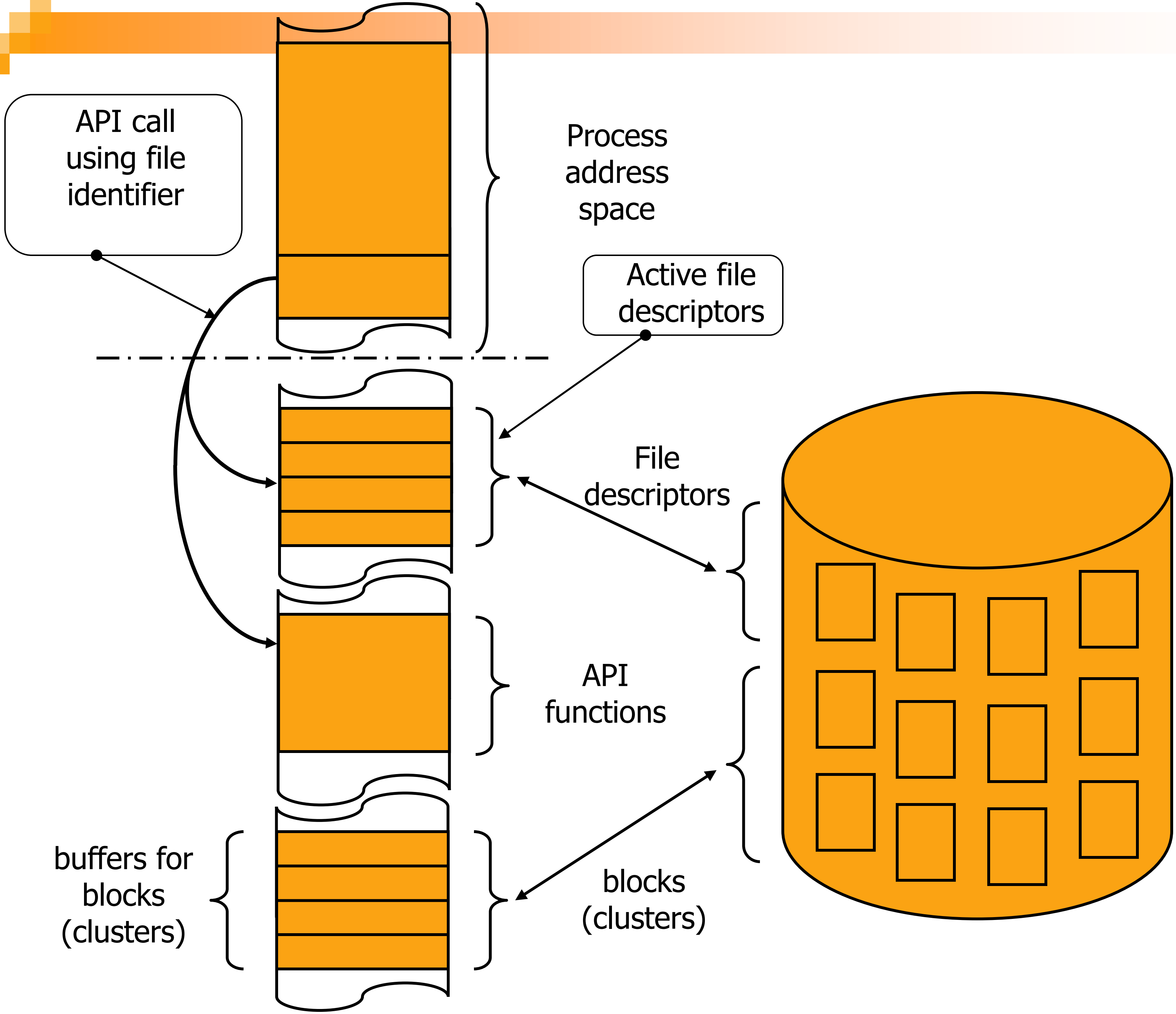
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 **3** | 19 **4** | 20 **5** | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 33 | 34 | 35 | 36 | 37 **1** | 38 **2** | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 **11** | 52 **12** | 53 **13** | 54 **14** | 55 **15** | 56 |
| 57 **6** | 58 **7** | 59 **8** | 60 **9** | 61 **10** | 62 | 63 | 64 |

**FAT**; in **directory table** is number **37** as starting block

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| | 19 | 20 | 57 | | | | |
| | | | | | | | |
| | | | | 38 | 18 | | |
| | | | | | | | |
| | | 52 | 53 | 54 | 55 | -1 | |
| 58 | 59 | 60 | 61 | 51 | | | |

# File subsystem

- Working with files requires several operations from the operating system
- OS must:
  - make a copy of file table in RAM
  - for every file in use
    - load file descriptor and extend it with location pointer
    - create buffers
    - …
- Files are manipulated through location pointer, also known as *file pointer*
  - at file open, file pointer is set to the start of the file
  - with reading or writing data, file pointer is moved forward


API call using file identifier
Process address space
Active file descriptors
File descriptors
API functions
buffers for blocks (clusters)
blocks (clusters)

# Using files

- OS provides interface for file access
- Examples:

```
int open  (char *filename, int access, int perm);
int close (int handle);
int read  (int handle, void *buffer, int nbyte);
int write (int handle, void *buffer, int nbyte);

int lseek (int fildes, int offset, int whence);
```

# Operating system concepts

***Distributed Systems***

# Distributed, parallel computing

- Distributed computing:
  - more than one node – more than one computer, interconnected with appropriate communication mechanism (net, internet)
- Parallel computing:
  - more than one processing element
    - in single multiprocessor (or multi core) computer system
    - in distributed systems
  - *the usual definition* for "parallel computing" implies a **single system** with shared memory, not distributed!
- Distributed application
  - application whose parts are executing on more than one node (using communication mechanisms provided by network subsystem) for performing requested operation

# Distributed architectures examples

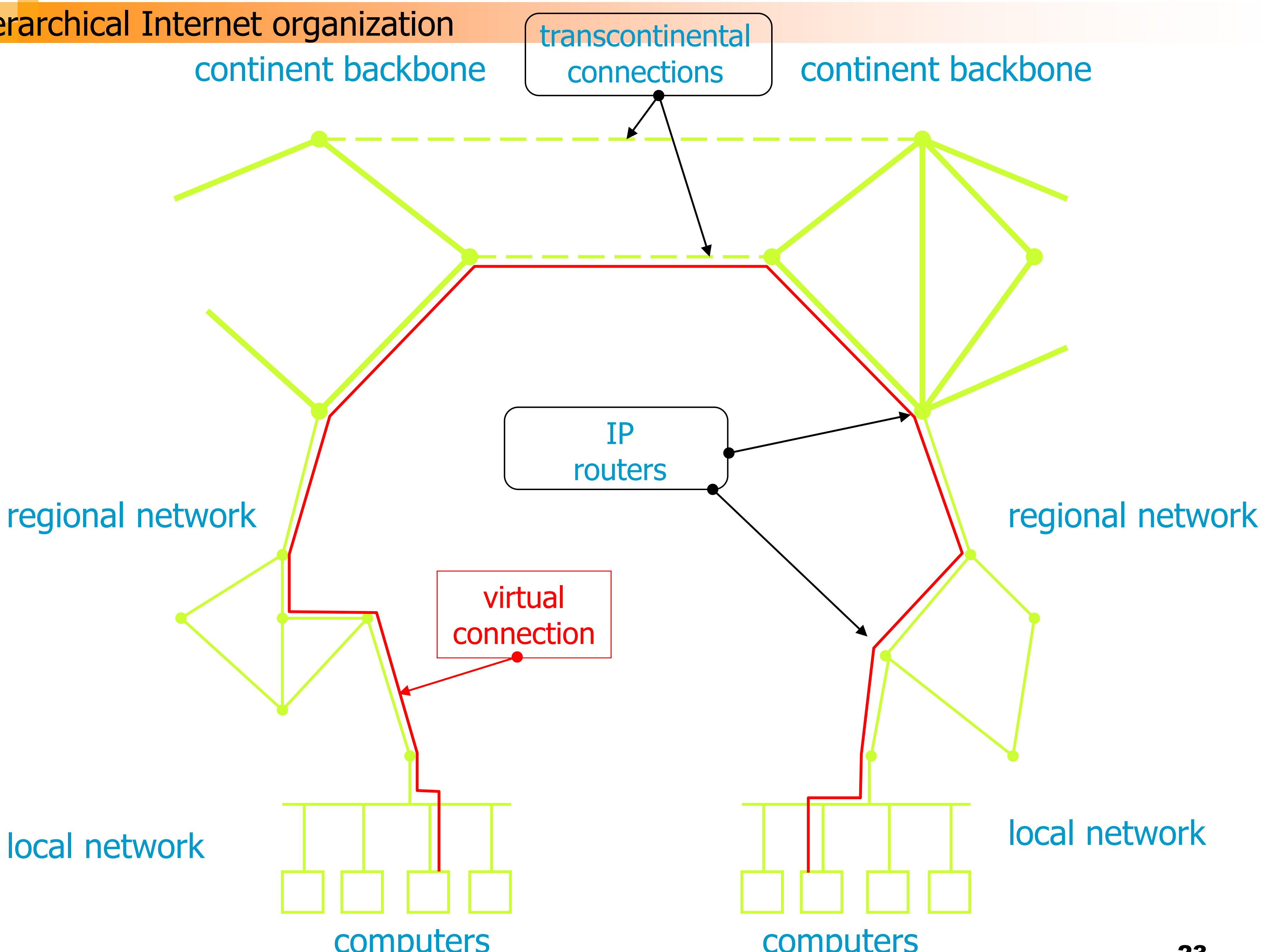
- Client-server architecture
  - server provides services for various clients
  - clients use services from various servers to perform requested operations
  - mostly used architecture today
    - examples: Web (HTTP), mail (SMTP/IMAP/POP), FTP, instant messaging, database access
- 3-tier, n-tier architecture
  - client-server approach where client and/or server functionality is divided into *tiers* (e.g. user-logic-data)
- Peer-to-peer architecture
  - there is no master node, each node can act as client and server
  - e.g. SMTP (exchanging messages between mail servers), DNS, routers, P2P file sharing

# Operating system – network subsystem

- Network subsystem is very complex!
  - layered architecture is used (required)
- ISO defines 7 layer referent model: OSI-RM
- In real world, **TCP/IP** model is used
  - only 4 layers:
    - application layer
      - interpret (give meaning) to received data
      - e.g. HTTP: GET /index.html HTTP/1.1
    - transport layer
      - connect send/received data with a socket (application that uses it)
    - internet layer (network layer from the picture)
      - forward IP packets through nodes toward destination node
    - link layer (data link + physical layers)
      - send/receive data between two nodes

| The OSI Model |
| --- |
| **7. Application Layer** |
| NNTP · SIP · SSI · DNS · FTP · Gopher · HTTP · NFS · NTP · SMPP · SMTP · SNMP · Telnet · (more) |
| **6. Presentation Layer** |
| MIME · XDR · TLS · SSL |
| **5. Session Layer** |
| Named Pipes · NetBIOS · SAP |
| **4. Transport Layer** |
| TCP · UDP · SCTP · DCCP |
| **3. Network Layer** |
| IP · ICMP · IPsec · IGMP · IPX · AppleTalk |
| **2. Data Link Layer** |
| ARP · CSLIP · SLIP · Ethernet · Frame relay · ITU-T G.hn DLL · L2TP · PPP · PPTP |
| **1. Physical Layer** |
| RS-232 · RS-449 · V.35 · V.34 · I.430 · I.431 · T1 · E1 · POTS · SONET/SDH · OTN · DSL · 802.11a/b/g/n PHY · ITU-T G.hn PHY · Ethernet · USB · Bluetooth |

# Hierarchical Internet organization



# Using network subsystem

- Two basic communication mechanisms:
  - message passing interface
    - send message to node
    - wait for message from node (and read it upon receiving) or process pending (received) messages
    - mostly used when exchanged data is small
    - e.g. UDP (Universal Datagram Protocol)

  - virtual connections
    - create virtual connection channel
    - communicate through that channel with read/write (like with files and pipes)
    - protocol control transfer – ordering packets, data integrity, …
    - mostly used for file transfer protocols (e.g. HTTP, SMTP, FTP, …)
    - e.g. TCP (Transmission Control Protocol)

# Data sharing, synchronization ?

- No real shared memory!
- Virtual shared memory?
  - identical memory segment at all nodes
  - data change must be propagated (how?! - complex)

- Synchronization?
  - “disable interrupts” or “Test and Set” won’t work
  - no shared memory – original Dekker and Lamport won’t do
  - new mechanisms are required based only on message exchange mechanisms

- Synchronization (and data sharing) can be divided into:
  - centralized mechanisms
  - distributed mechanisms

# Centralized synchronization of distributed nodes

- Central node decides who may enter critical sections
- All nodes send requests to central node
  - when they receive response, they enter C.S.
- Upon exiting from C.S. node sends a message to central node which then signals the next node
- Protocol is highly dependent on central node
- An variation of this protocol uses *token* as C.S. object
  - token is passed in circular manner among nodes
  - when a node receives the token it can enter its C.S.
  - when leaving C.S. or if node doesn't require the token, it passes the token to the next node in chain

# Distributed synchronization of distributed nodes

- Idea:
  - all nodes that wants to enter C.S. send request to all other nodes
  - all nodes have request queue sorted by request time
  - first request from queue is granted entrance – all nodes confirm that by response message to corresponding node
  - when leaving C.S. node sends message to all other nodes – this request is then removed from queues and the next one is allowed to enter
- Problems:
  - different nodes may use different clocks
  - messages don't arrive instantly, and even not in the *same order* as they were sent
  - => local "time" can't be used, but another mechanism must be built: global events ordering

# Global logical time

- To achieve global event ordering (not necessary equivalent with *time* ordering!) a few rules must be implemented
- Every node keeps track of its local logical time
- Every time a node sends a message to another node it adds its local time-stamp to the message
- Every time a node receives a message – it updates its logical time to a value that is larger than its *previous local time* and the *received time stamp*
- When comparing events – compare their attached time stamps!
- Algorithms that use those principles:
  - *Lamport's Distributed Mutual Exclusion Algorithm*
  - *Ricart-Agrawala algorithm*