Exam: see LMS for details

Exam cover sheet

- *similar format to the mid-semester test*

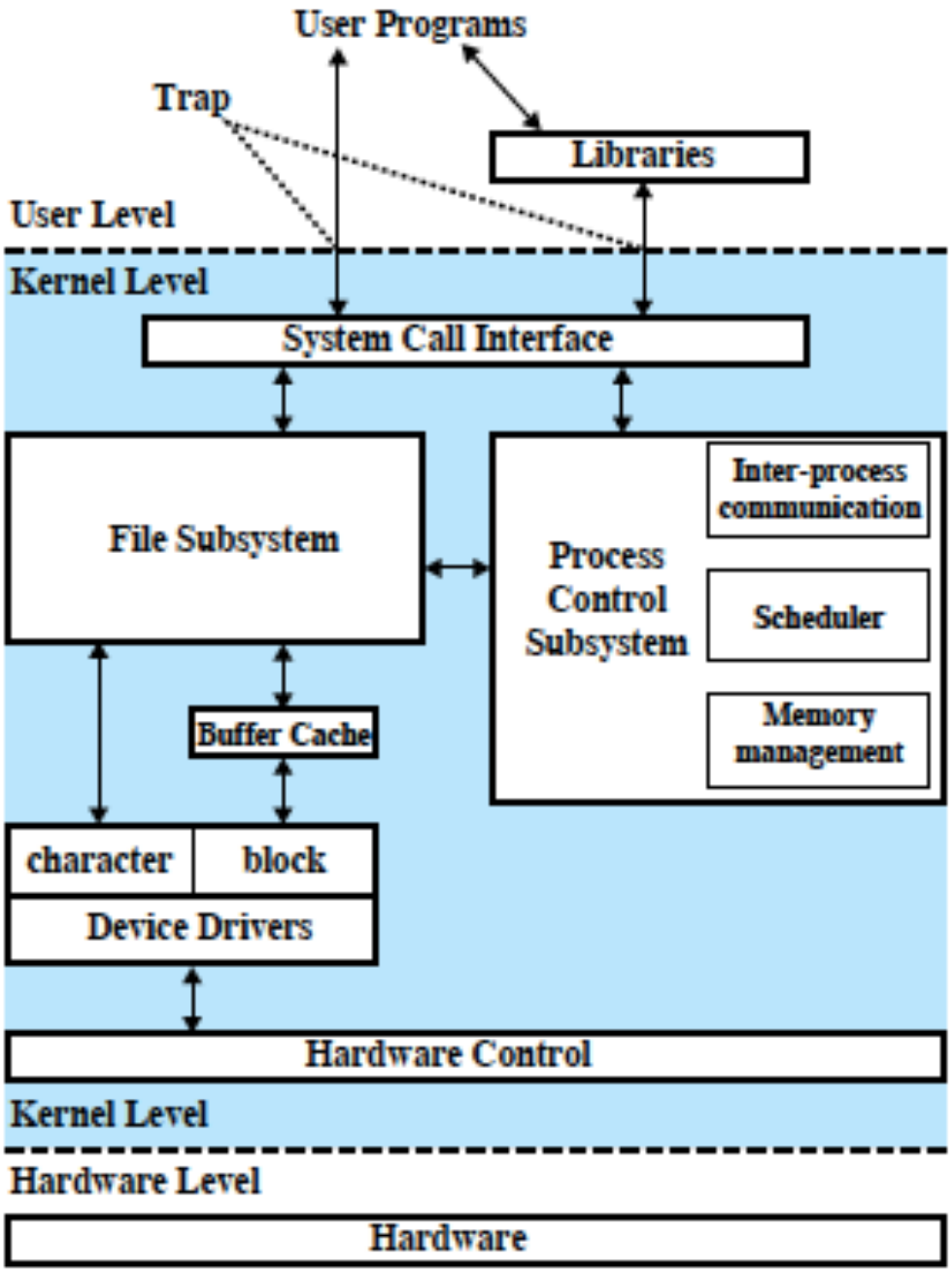
Tutorial questions are the best guide

- *solutions available on the LMS*

Sample short answer exams questions are available on the LMS

Lab tasks

- *key concepts will be examined*
- *you do not have to write C code on the exam*


User Programs
Trap
Libraries
User Level
Kernel Level
System Call Interface
File Subsystem
Inter-process communication
Process Control Subsystem
Scheduler
Memory management
Buffer Cache
character
block
Device Drivers
Hardware Control
Kernel Level
Hardware Level
Hardware

Learning outcomes

On completion of this section, your are expected to demonstrate an understanding of:

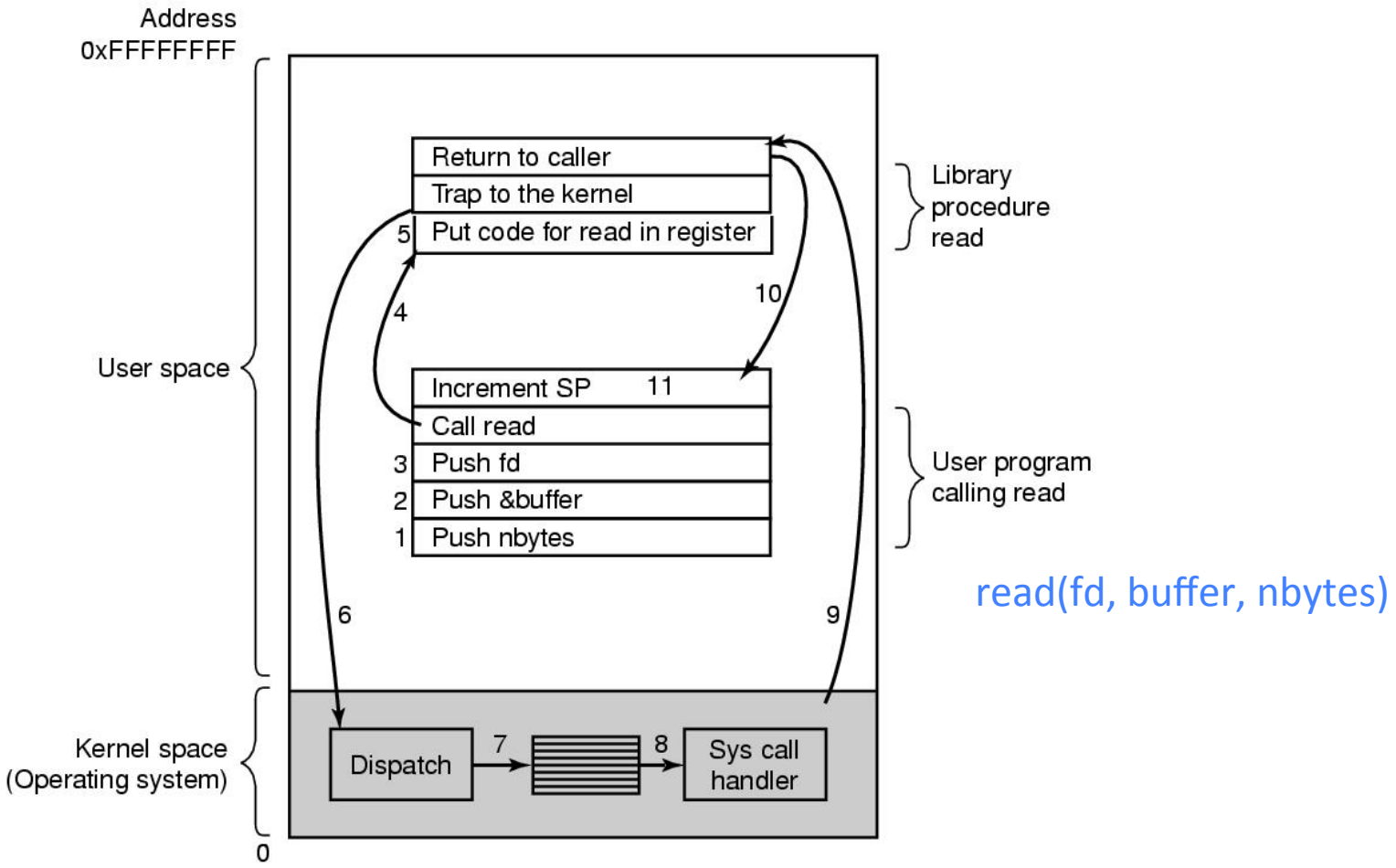
- Processes and multi-programming
  - managing processes in Unix/Linux
- The user – kernel distinction
- System calls
  - Unix/Linux examples
- Threads

System calls for process management

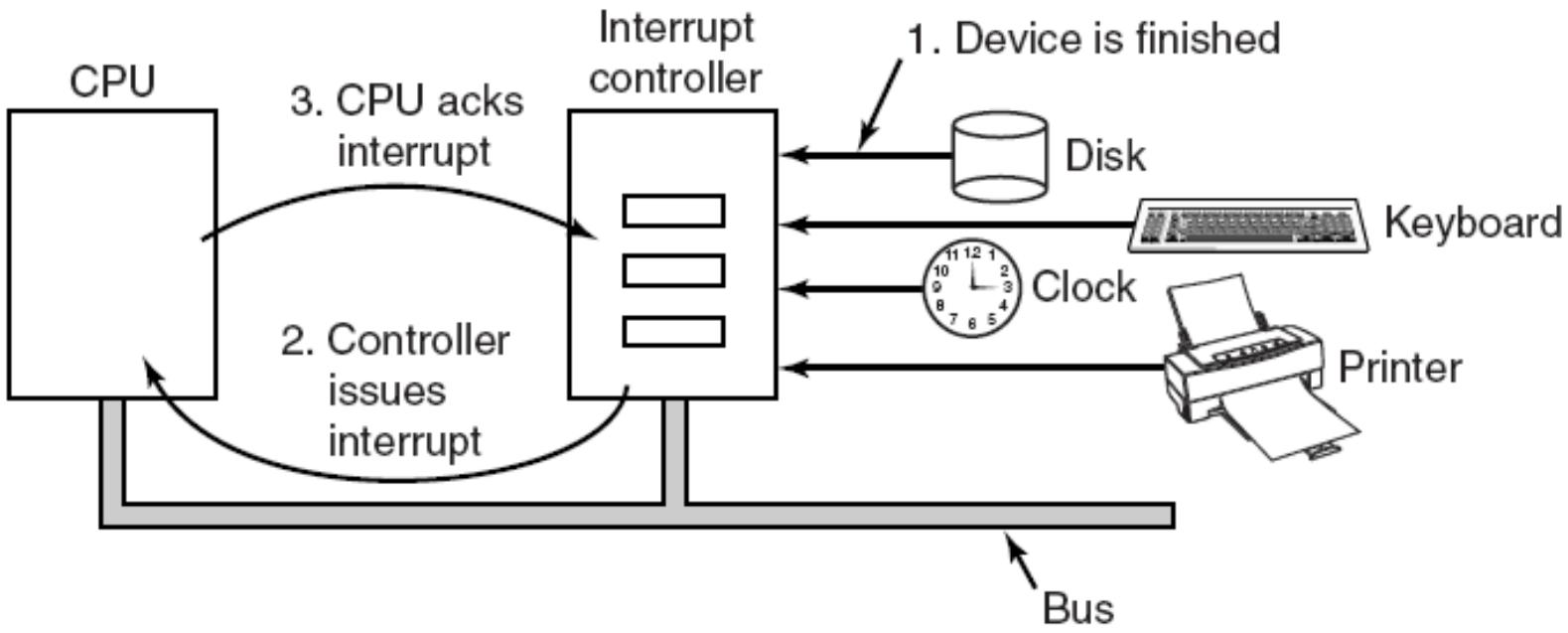
Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

System call example



Interrupts

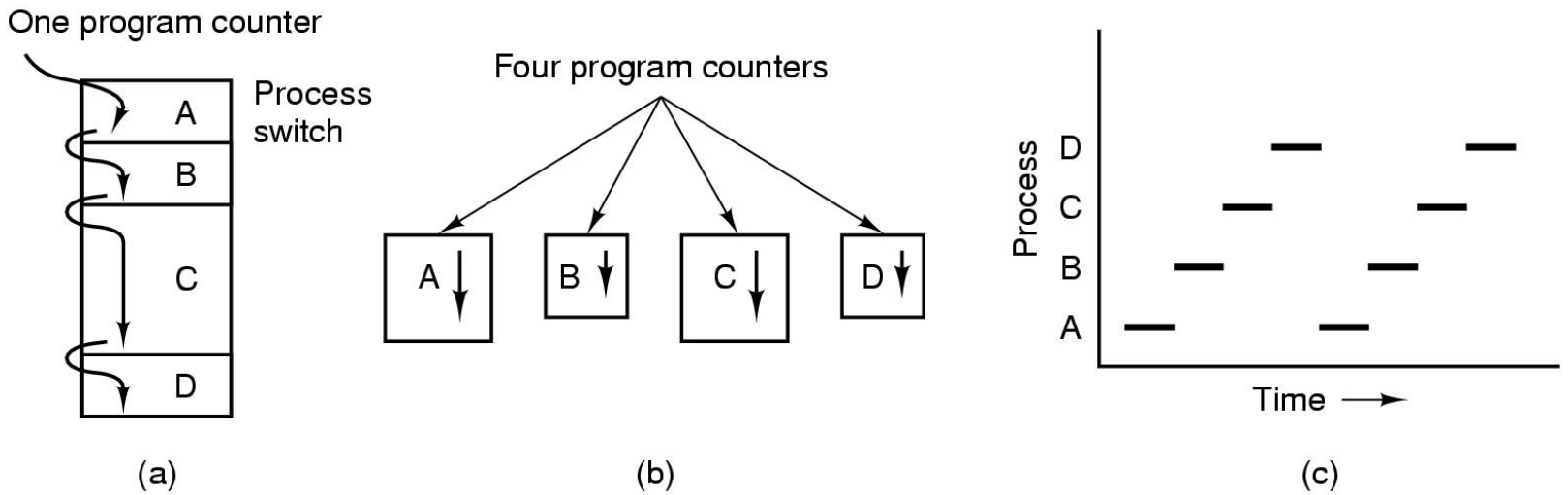

CPU
3. CPU acks interrupt
Interrupt controller
1. Device is finished
Disk
Keyboard
Clock
Printer
2. Controller issues interrupt
Bus

Learning outcomes

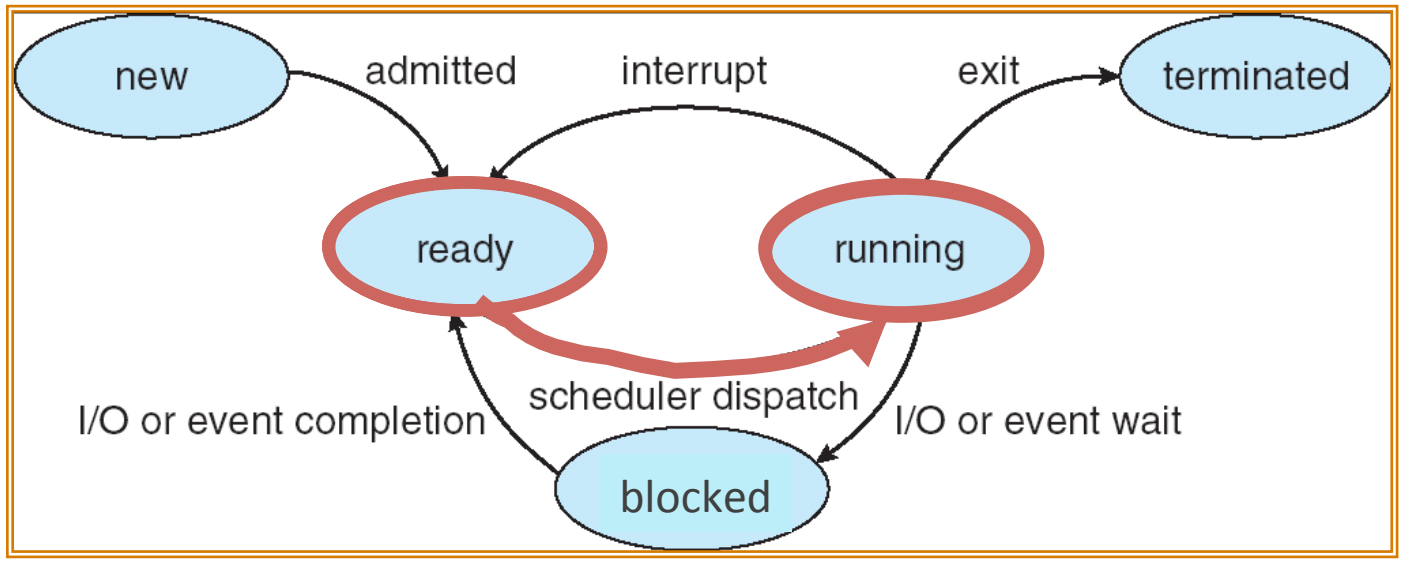
On completion of this section, your are expected to demonstrate an understanding of:

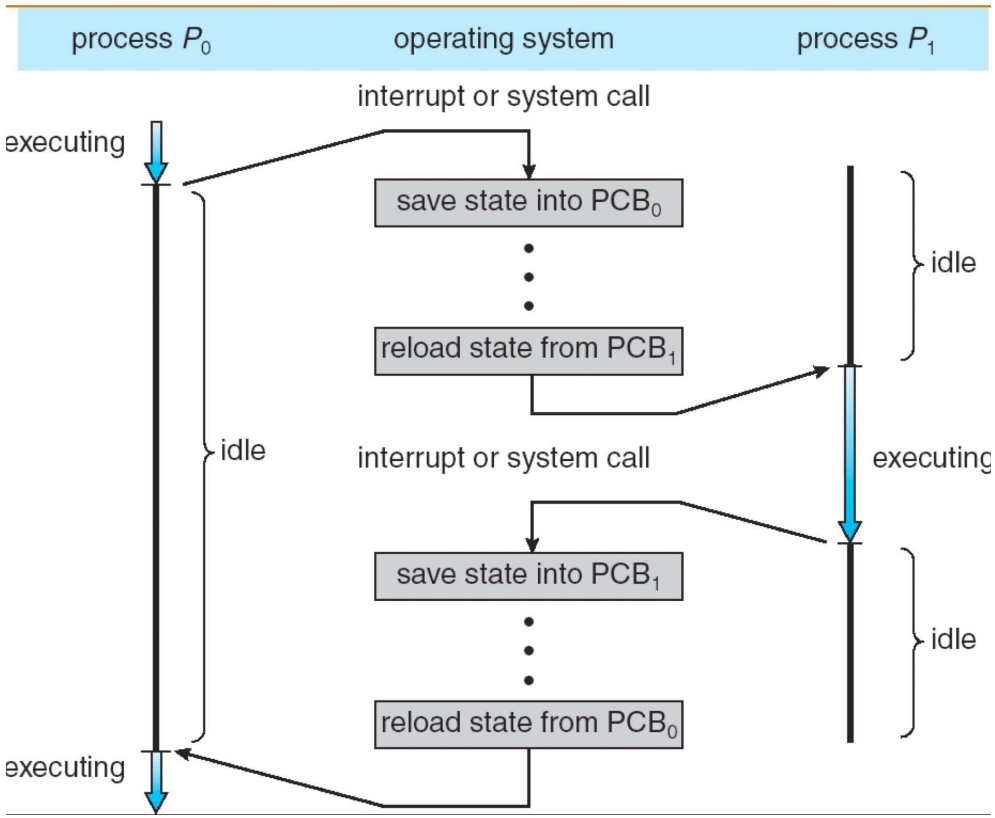
- Process states (and the use of PCB)
- Context switching
- The use of scheduling algorithms
  - Explain the performance characteristics of particular algorithms

Process model



Process life cycle




process P_0
operating system
process P_1
interrupt or system call
executing
save state into PCB_0
idle
reload state from PCB_1
idle
interrupt or system call
executing
save state into PCB_1
idle
reload state from PCB_0
executing

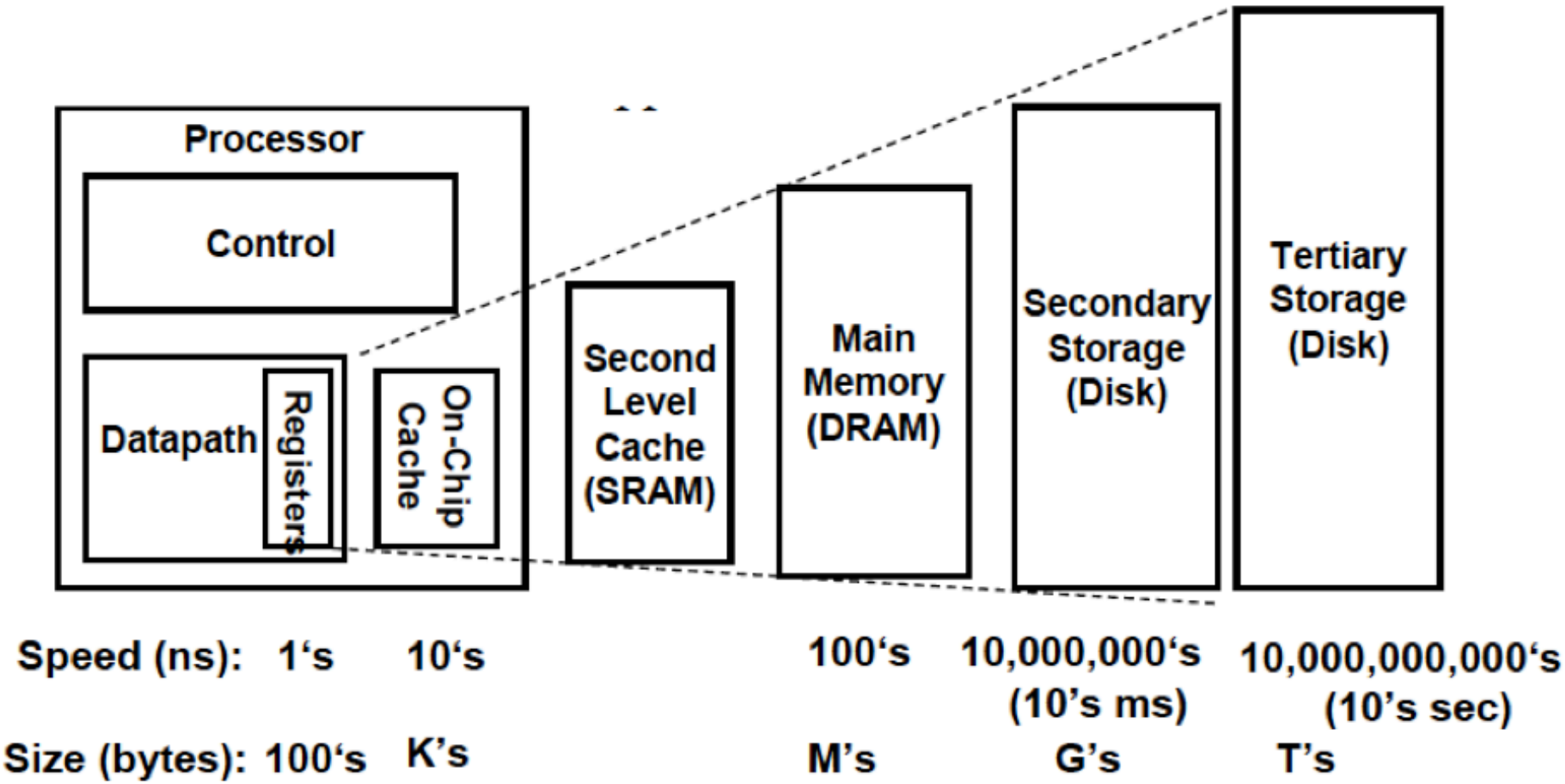
Process scheduling

Learning outcomes

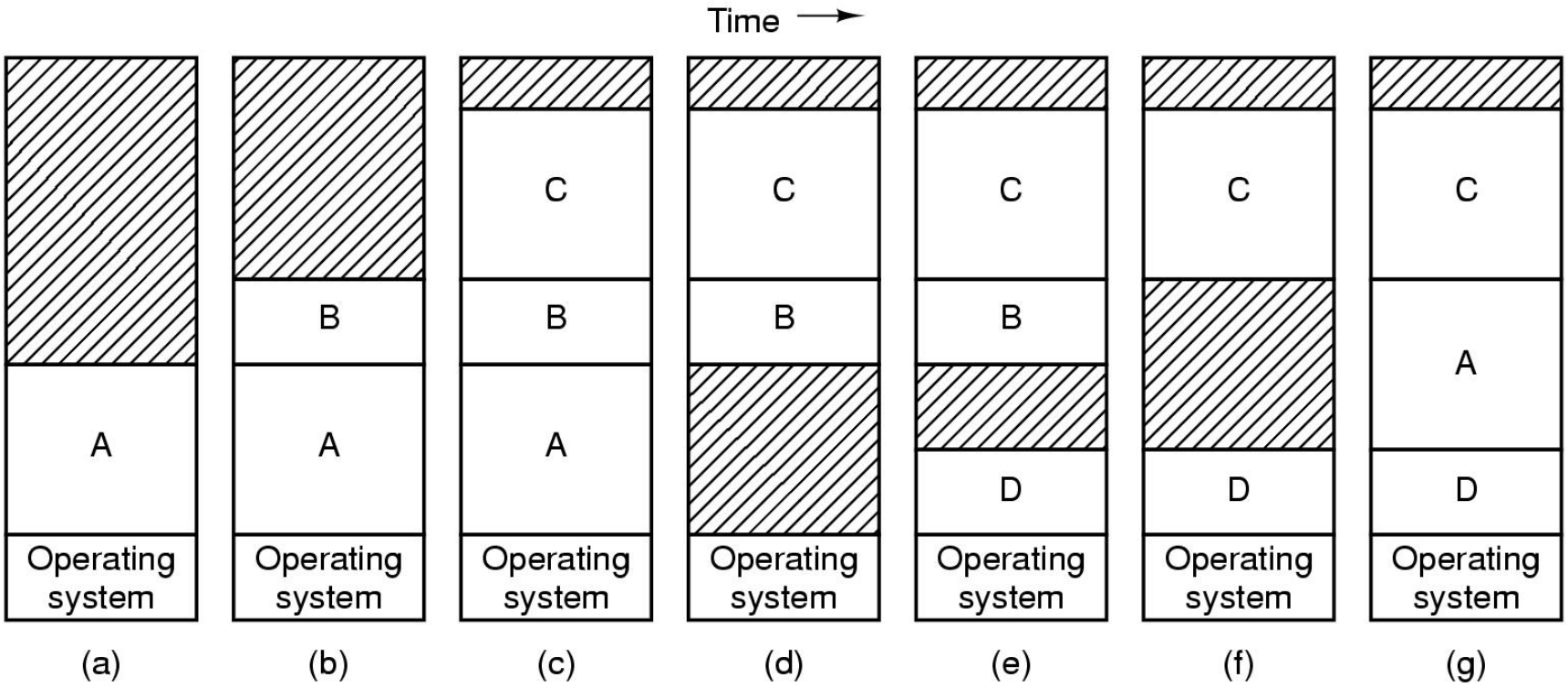
On completion of this section, your are expected to demonstrate an understanding of:

- Fragmentation
- The mapping between physical and virtual memory
- Paging
  - page faults, page table,
  - page replacement algorithms
- Working set

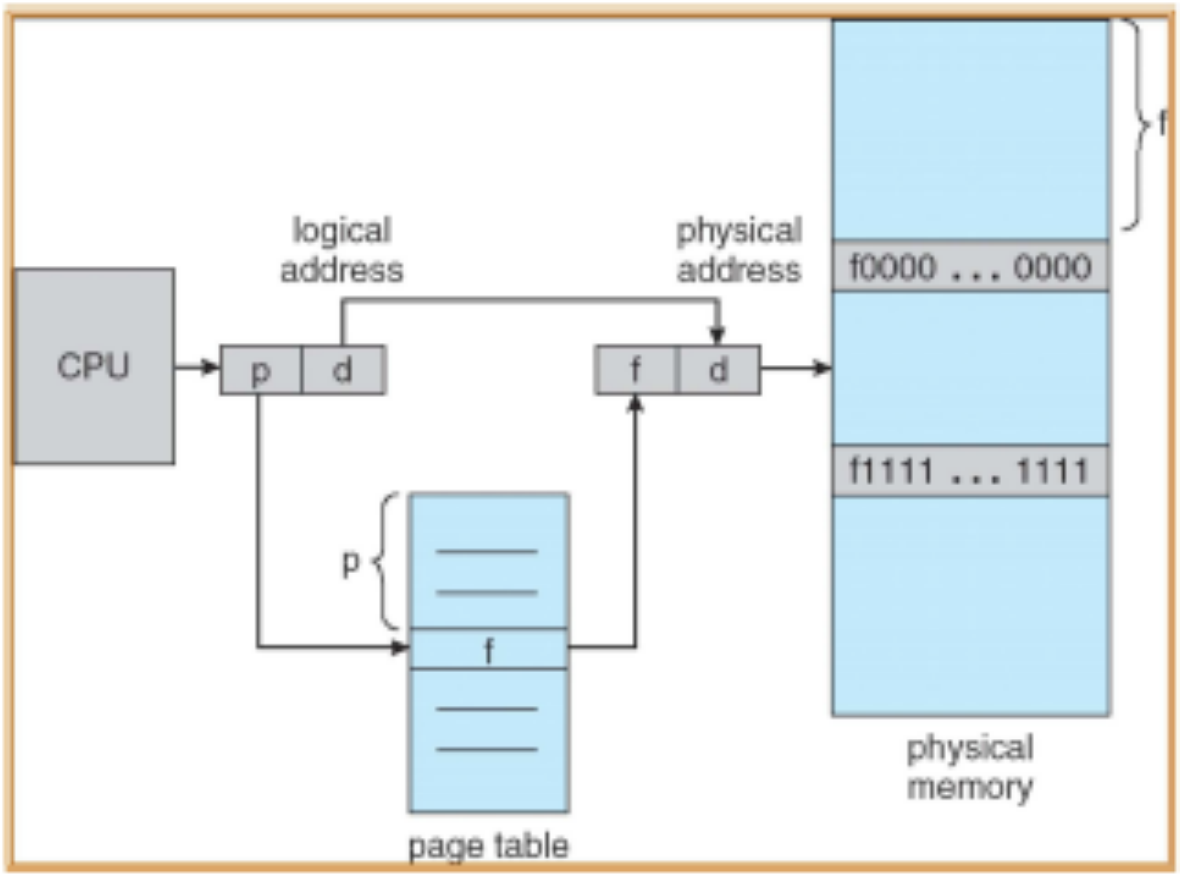
Memory hierarchy



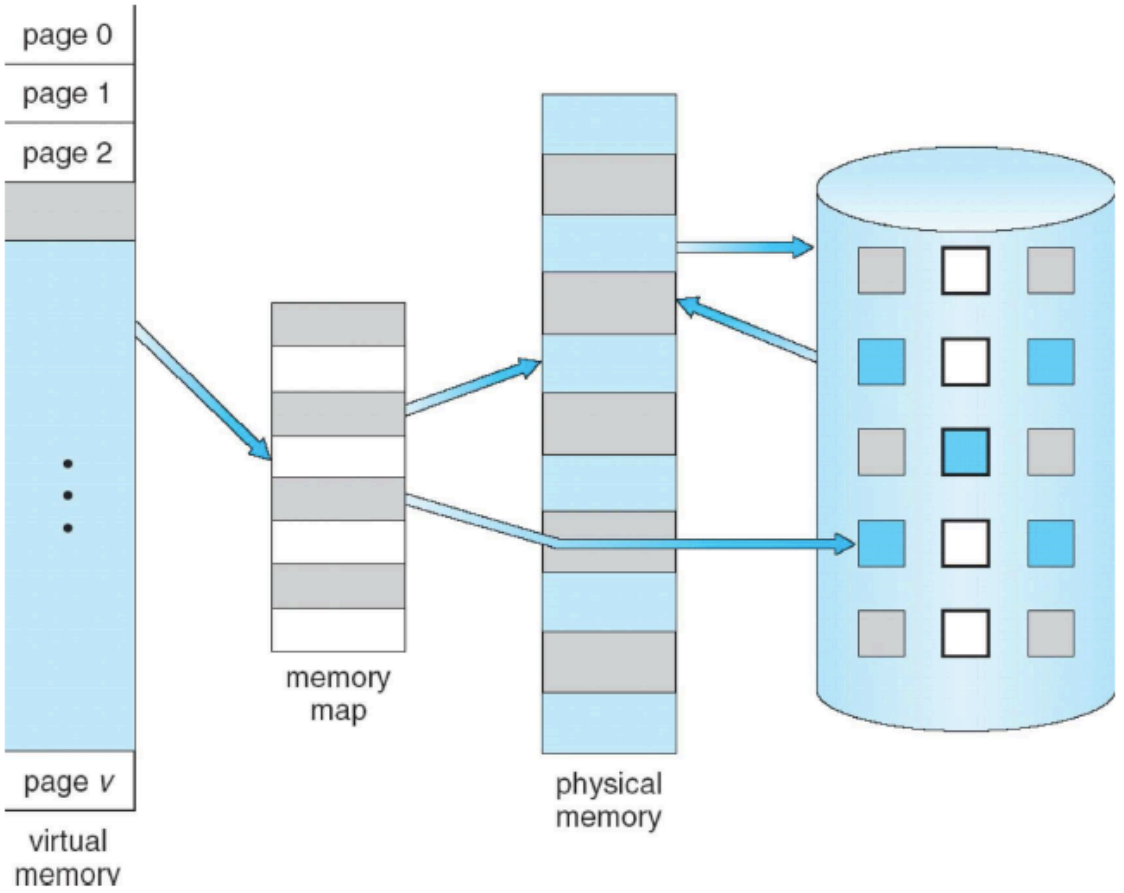
Swapping



Page based memory management



Page based memory management

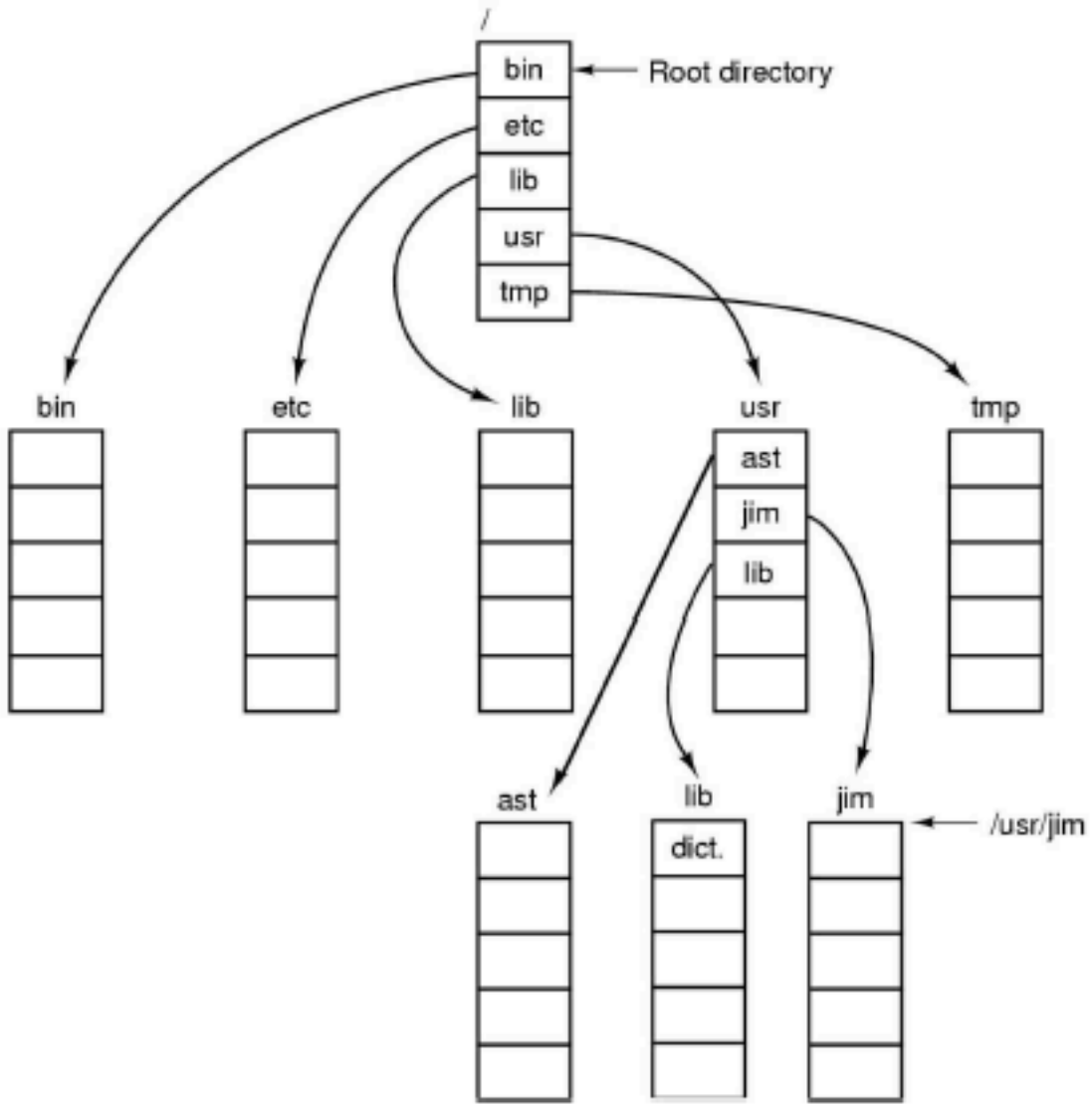


Learning outcomes

On completion of this section, your are expected to demonstrate an understanding of:

- The purpose of a file system
  - application in Linux/Unix
- File permissions (in Linux/Unix)
- The use of file descriptors
- File allocation methods
  - Including FAT and indexed (use of i-node)

File system


/
bin
Root directory
etc
lib
usr
tmp
bin
etc
lib
usr
tmp
ast
jim
lib
ast
lib
jim
/usr/jim
dict.

Unix stat

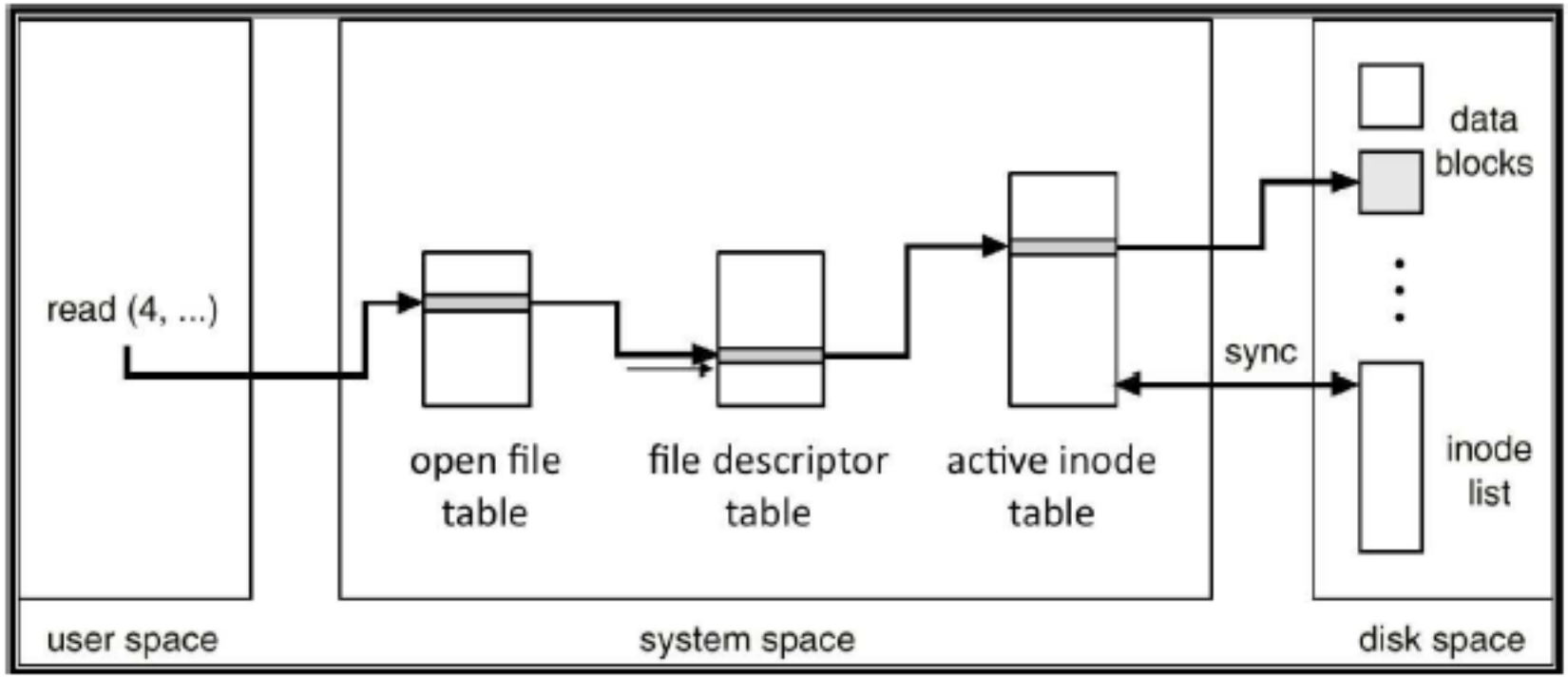
```
struct stat{
      mode_t          st_mode; // file type and mode(permission)
       ino_t          st_ino; //inode number
       dev_t          st_dev;  //device number
       dev_t          st_rdev;  //device number (special)
       nlink_t        st_nlink; //number of links
       uid_t          st_uid;  // user ID of owner
       gid_t          st_gid;  // group ID of owner
       off_t          st_size;  //size in bytes
       time_t         st_atime; //time of last access
   ……}
```

System calls for file management

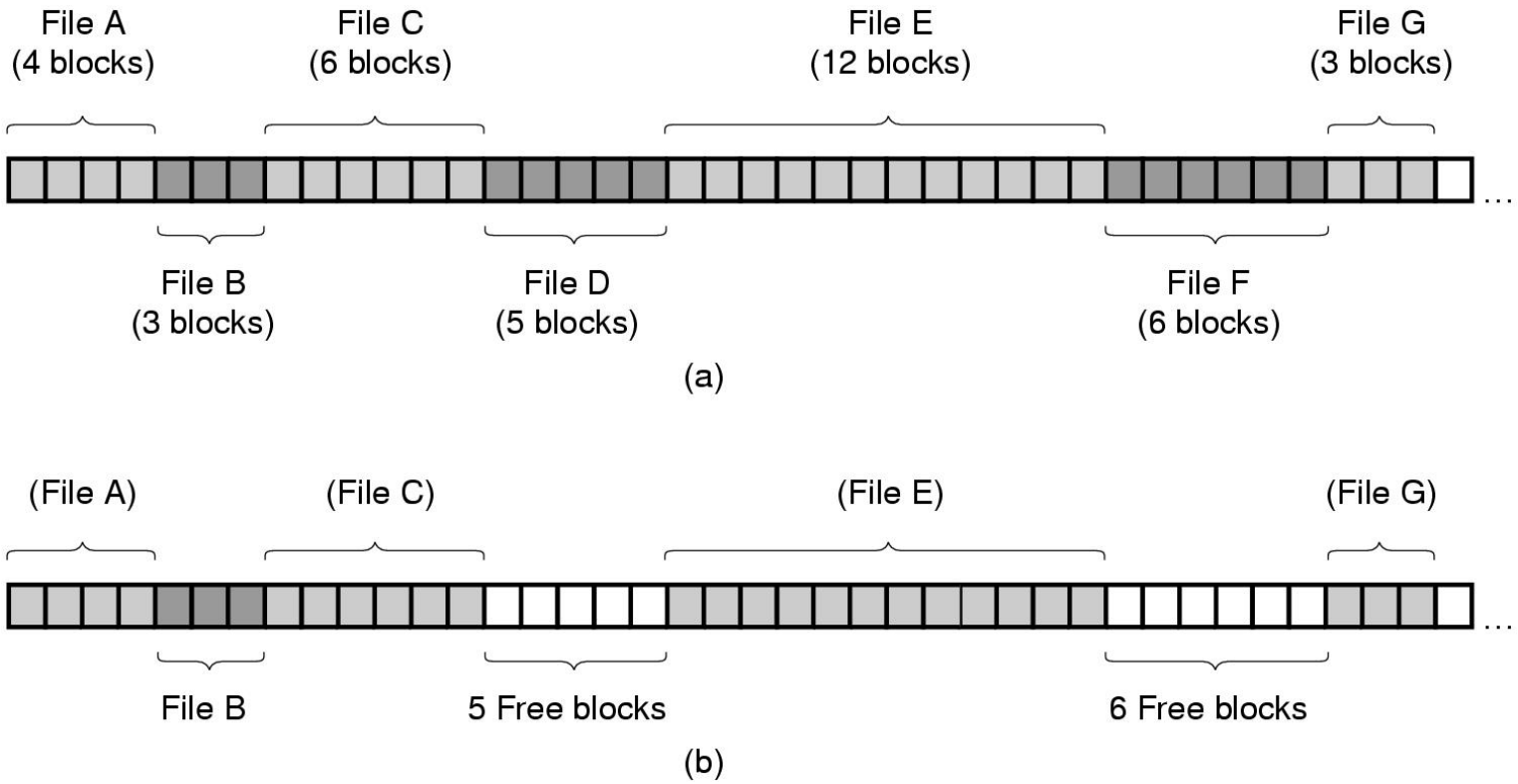
File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Using files in Unix



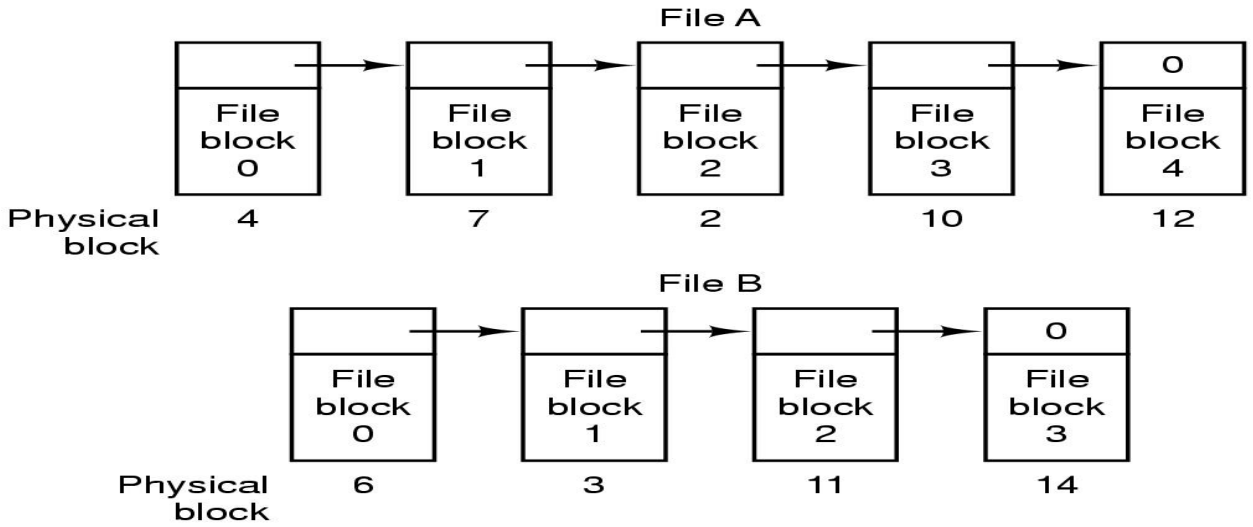
Contiguous allocation scheme



(a) Contiguous allocation of disk space for 7 files.

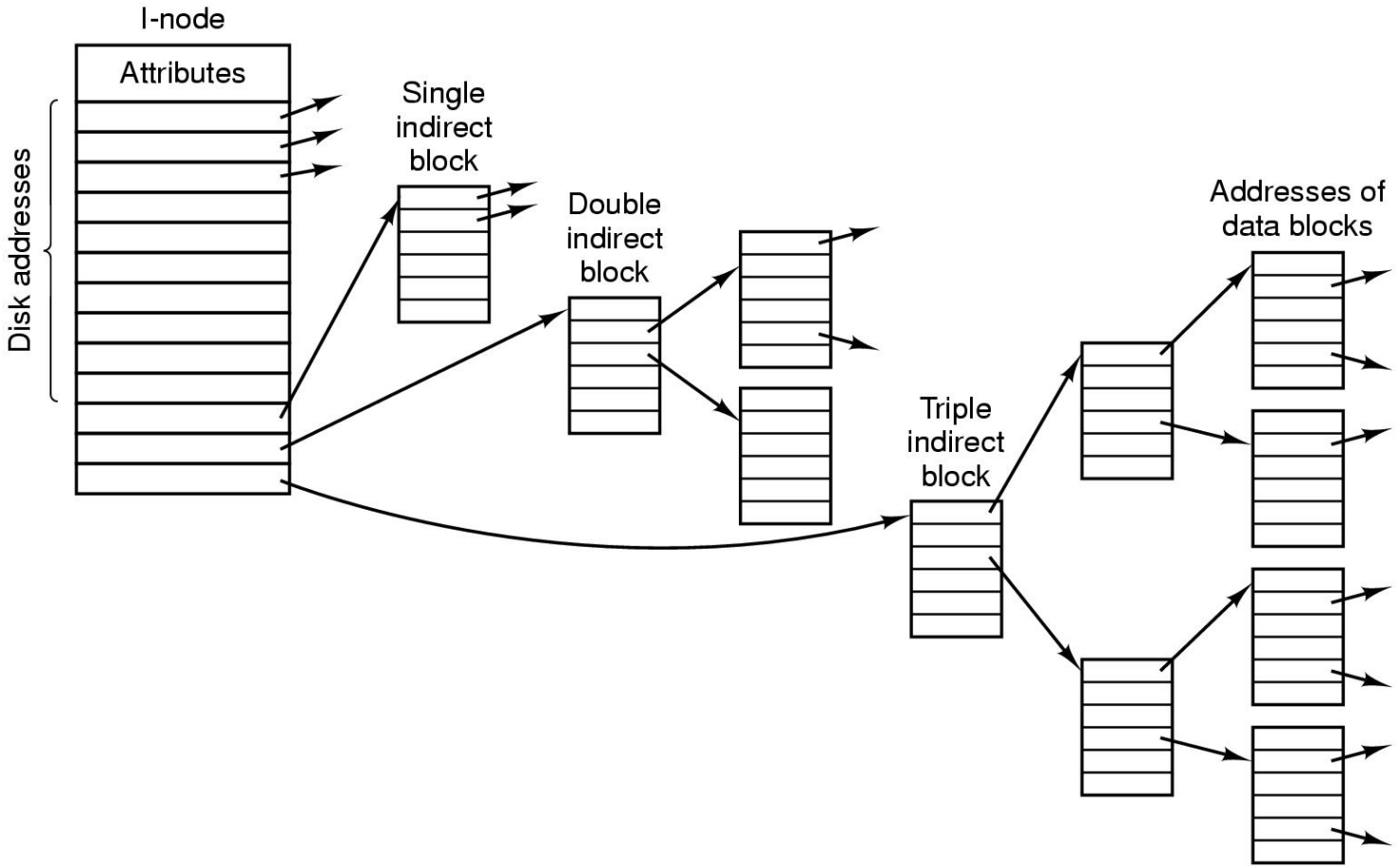
(b) The state of the disk after files D and F have been removed.

Linked list allocation scheme / FAT



- FAT 16
- FAT 32
  - Differ in how many bits a disk address contains
  - Block size also varies: 512B, 1K, 2K,4K,…,32K

UNIX file system

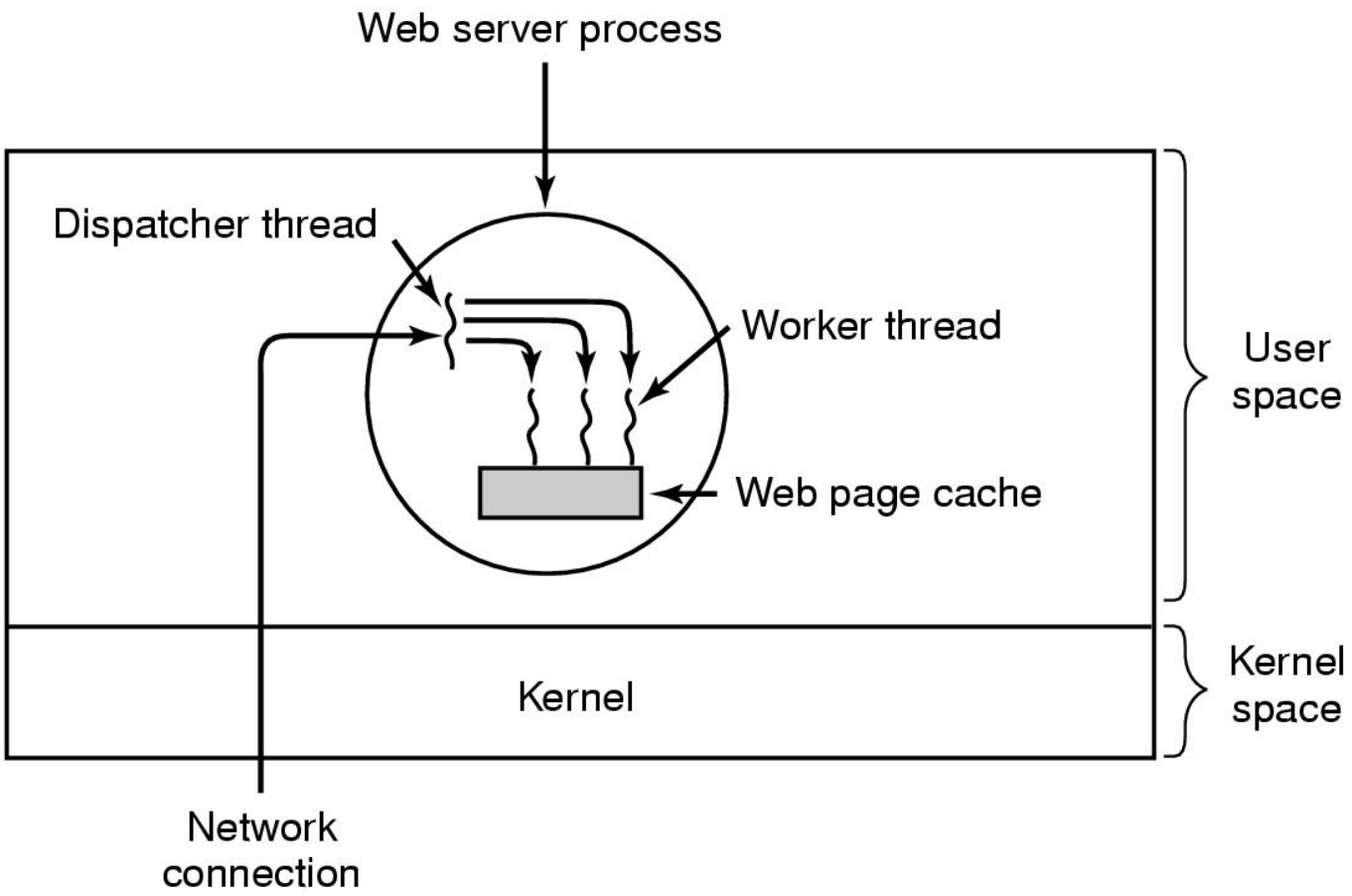


Learning outcomes

On completion of this section, your are expected to demonstrate an understanding of:

- Race conditions
- Mutual exclusion (protecting the critical section)
- Techniques to deal with mutual exclusion
  - Algorithms; Semaphores; Pthread mutex
- Practical examples (producer-consumer problem)

Threads


Web server process
Dispatcher thread
Worker thread
Web page cache
User space
Kernel
Kernel space
Network connection

Threads

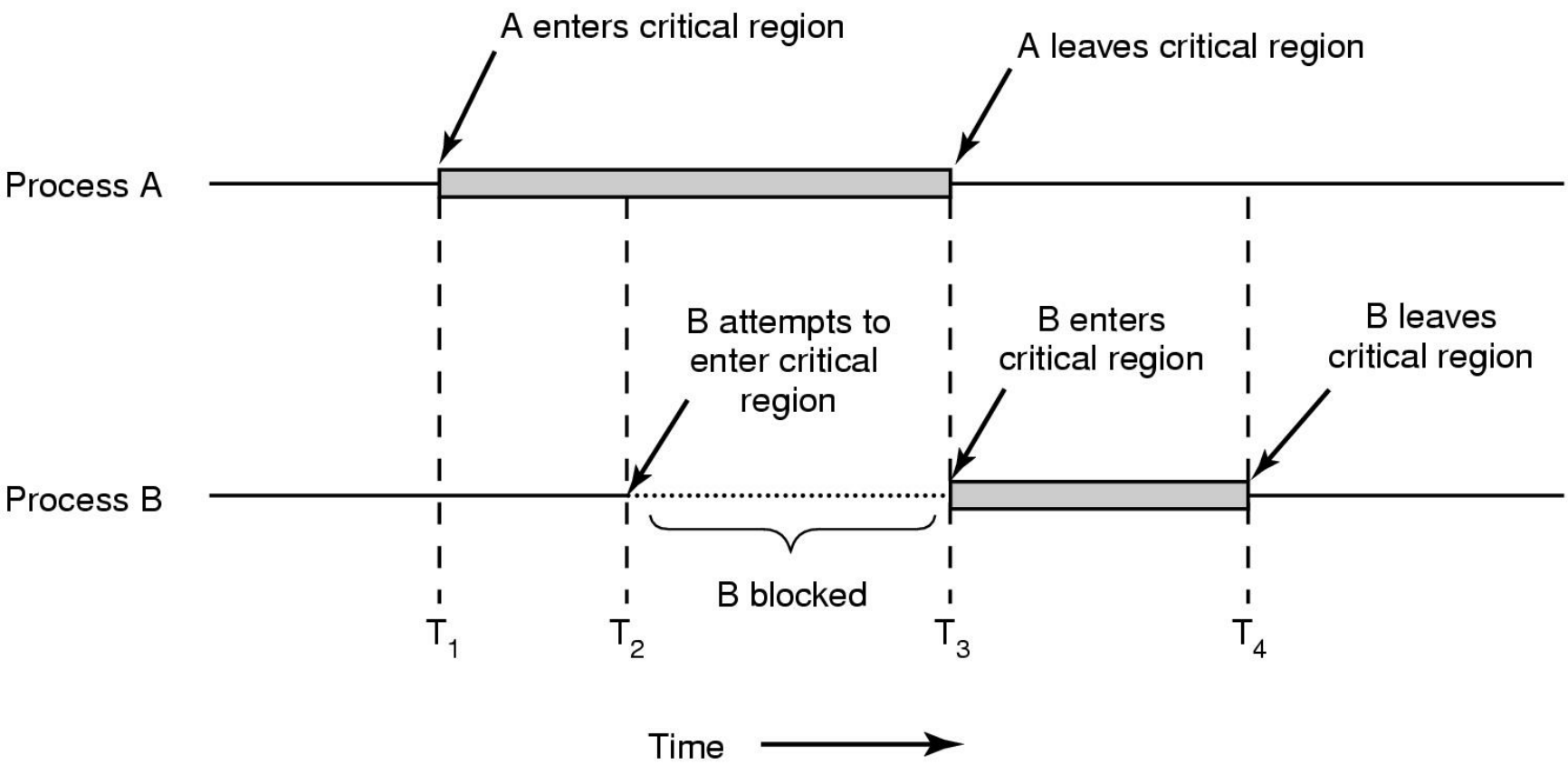
```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}
```

(a)

```
while (TRUE) {
  wait_for_work(&buf)
  look_for_page_in_cache(&buf, &page);
  if (page_not_in_cache(&page)
       read_page_from_disk(&buf, &page);
  return_page(&page);
}
```

(b)

Critical region / mutual exclusion



Critical region / mutual exclusion

```
while (TRUE) {
    while (turn != 0)        /* loop */ ;
    critical_region( );
    turn = 1;
    noncritical_region( );
}
```

(a)

```
while (TRUE) {
    while (turn != 1)        /* loop */ ;
    critical_region( );
    turn = 0;
    noncritical_region( );
}
```

(b)

Learning outcomes

On completion of this section, your are expected to demonstrate an understanding of:

- Key *terms* related to network security
- Symmetric key and public key cryptography
- Use of cryptography hash functions
- Digital signatures and certificates
- The use of SSL/TLS

Security definitions

- **Confidentiality**: only the sender and intended receiver should "understand" the message contents
- **Authentication:** the sender and receiver want to confirm the identity of each other
- **Message integrity**: the sender and receiver want to ensure that the message is not altered (in transit, or afterwards) without detection
- **Access and availability**: services must be accessible and available to users

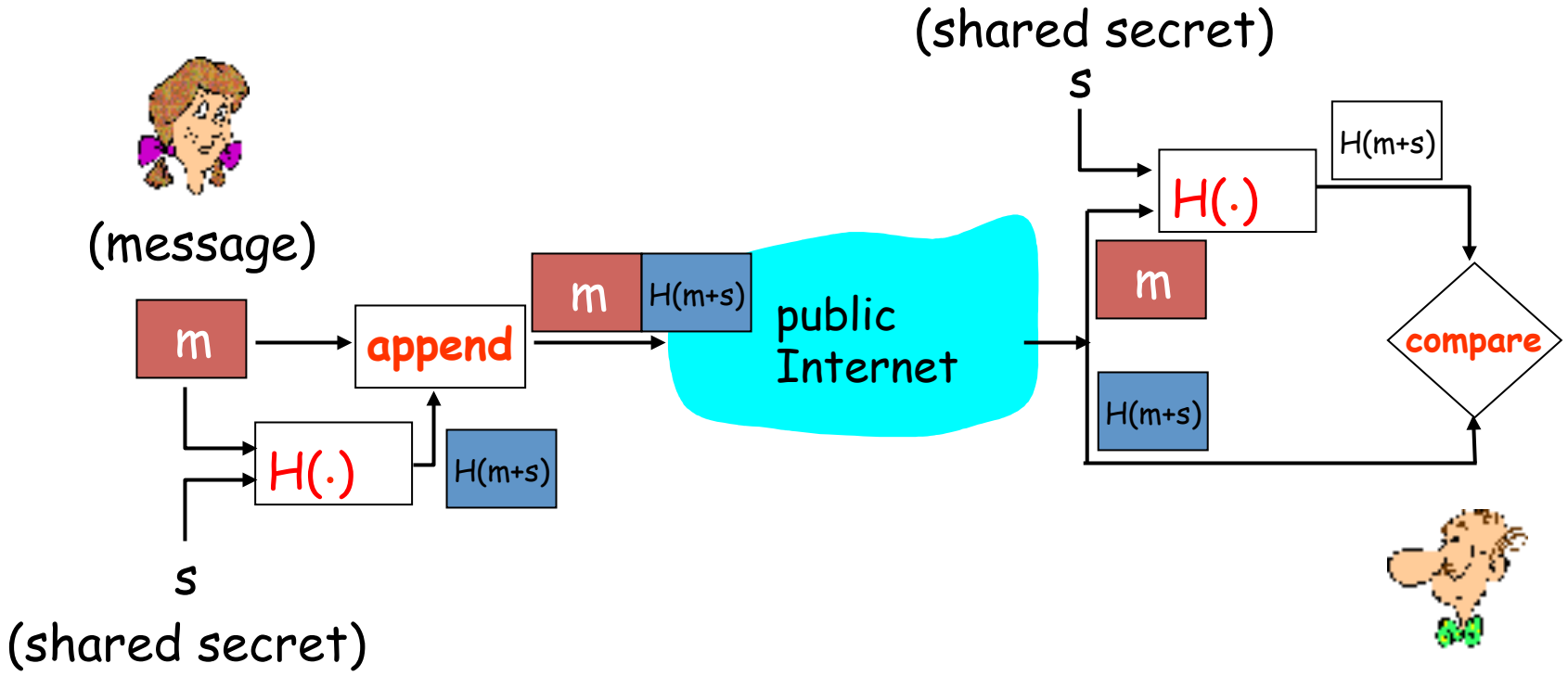
Secret key crypto

- Both the sender and the receiver had to agree on the secret key in advance
  - they had to “meet” somehow
- Encrypting and decrypting used the same key
  - these are still used, e.g. **AES**
  - choose a key from a huge set
    - it is not feasible to guess

Public key crypto

- The receiver generates two keys:
  - a public key **E** (for encrypting), and
  - a private key **D** (for decrypting)
- Publicises the public key **E**
  - people use this for encrypting messages
- Keeps the private key **D** secret
  - receiver uses this for decrypting messages

... using digital signatures

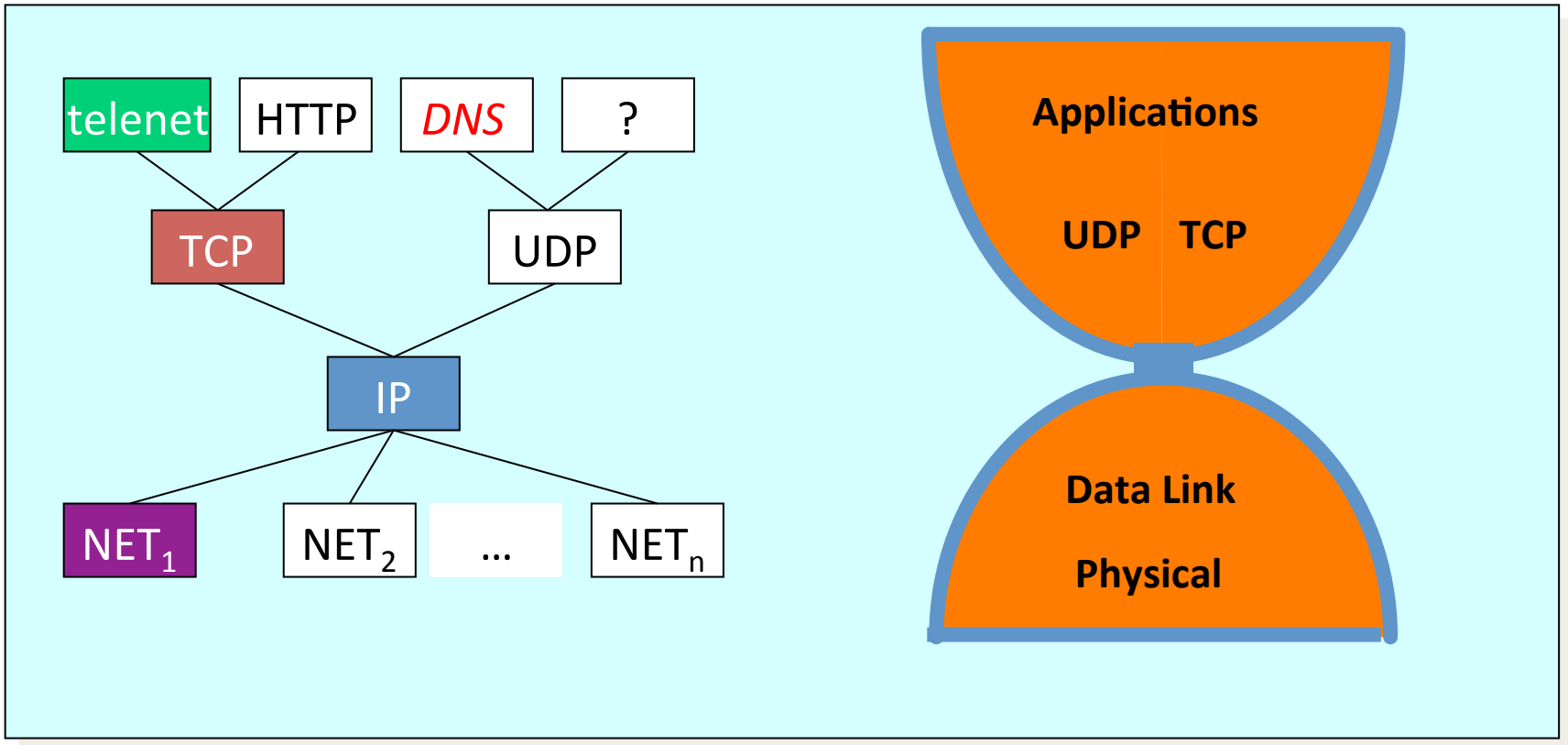


Learning outcomes

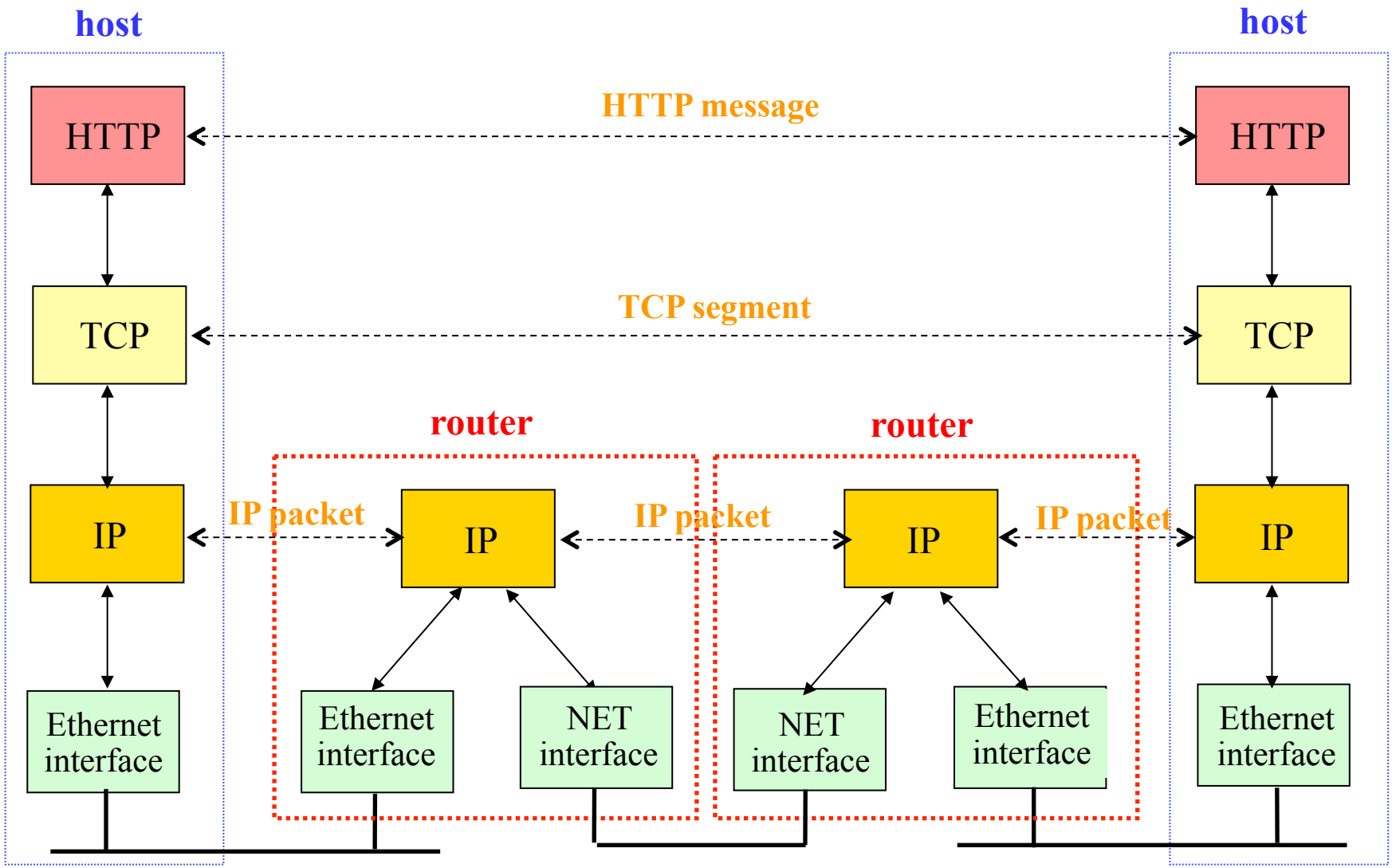
On completion of this section, your are expected to demonstrate an understanding of:

- *Network definitions*
- Packet switching *vs* Circuit Switching
- Network protocol hierarchies
- Relationship b/w a service and a protocol
- OSI model

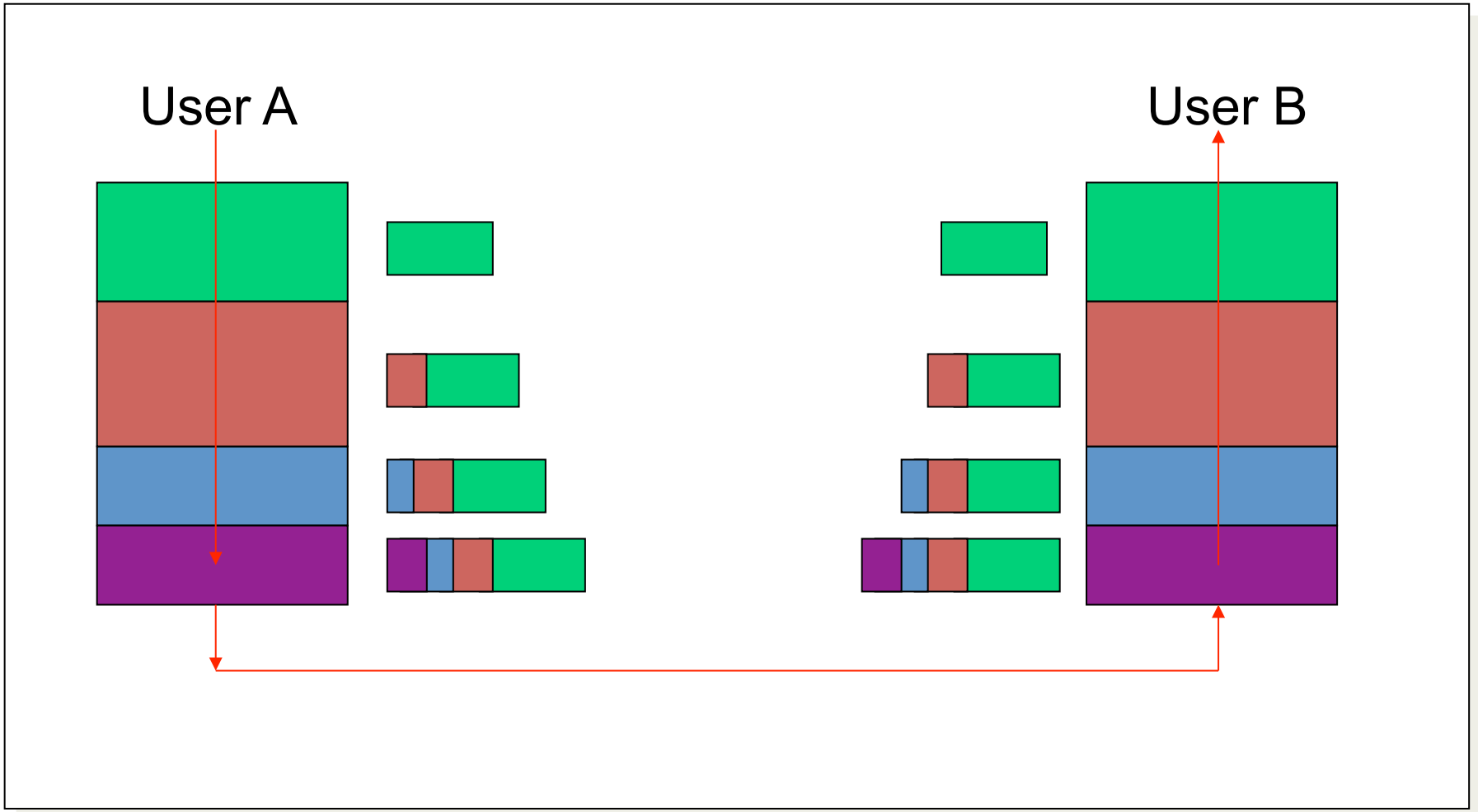
Internet protocol suite



OSI Model :: TCP/IP Model



Layer encapsulation: use of headers

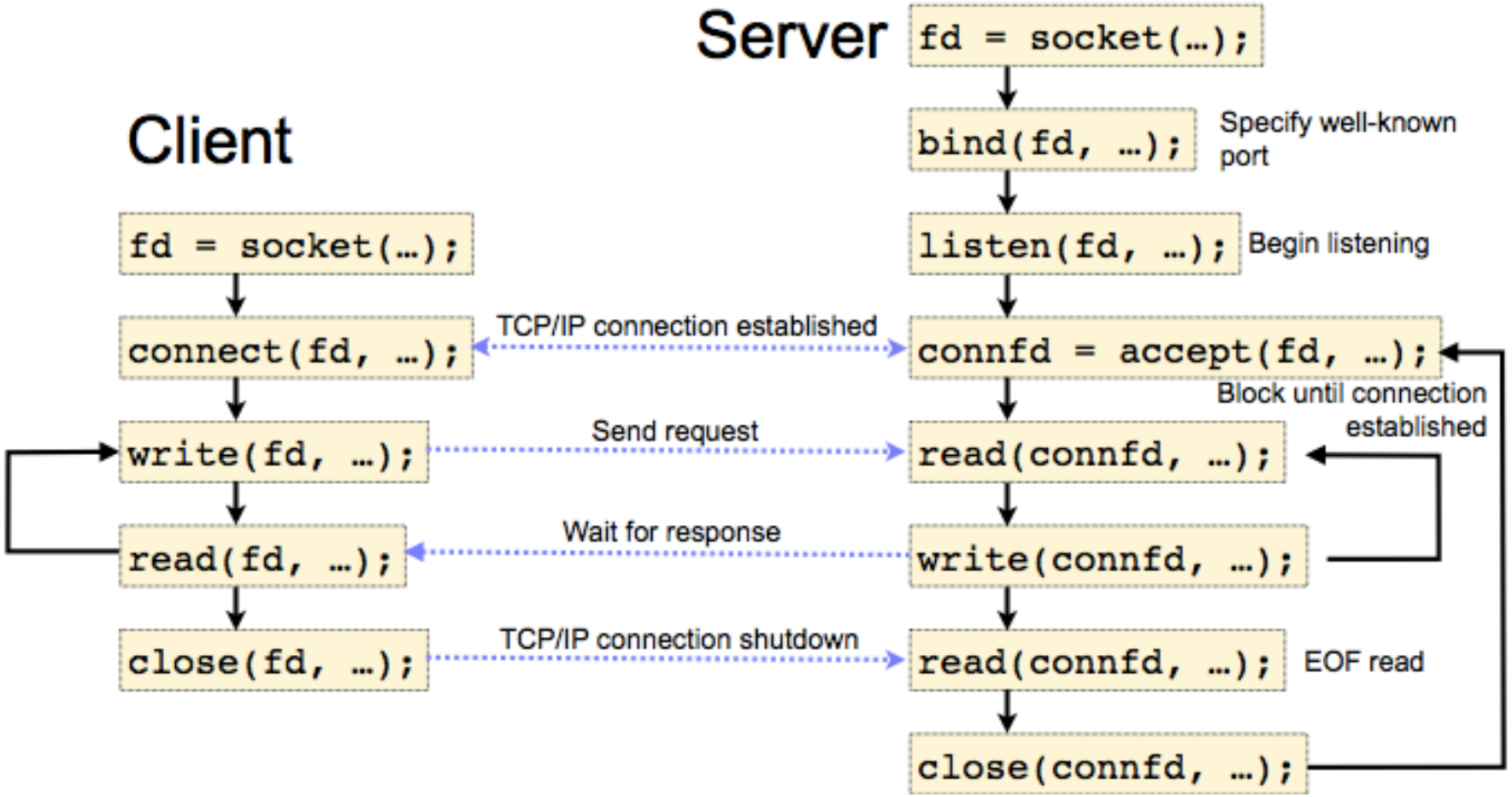


Learning outcomes

On completion of this section, your are expected to demonstrate an understanding of:

- The use of sockets in network programming
- The differences between TCP and IP sockets
  - stream vs datagram sockets
- Socket systems call
  - socket(), bind(), accept(), “reading” and “writing”
- Implement client and server programs
- How to write a server, which uses TCP sockets, that is capable of handling concurrent requests

Sockets (TCP)

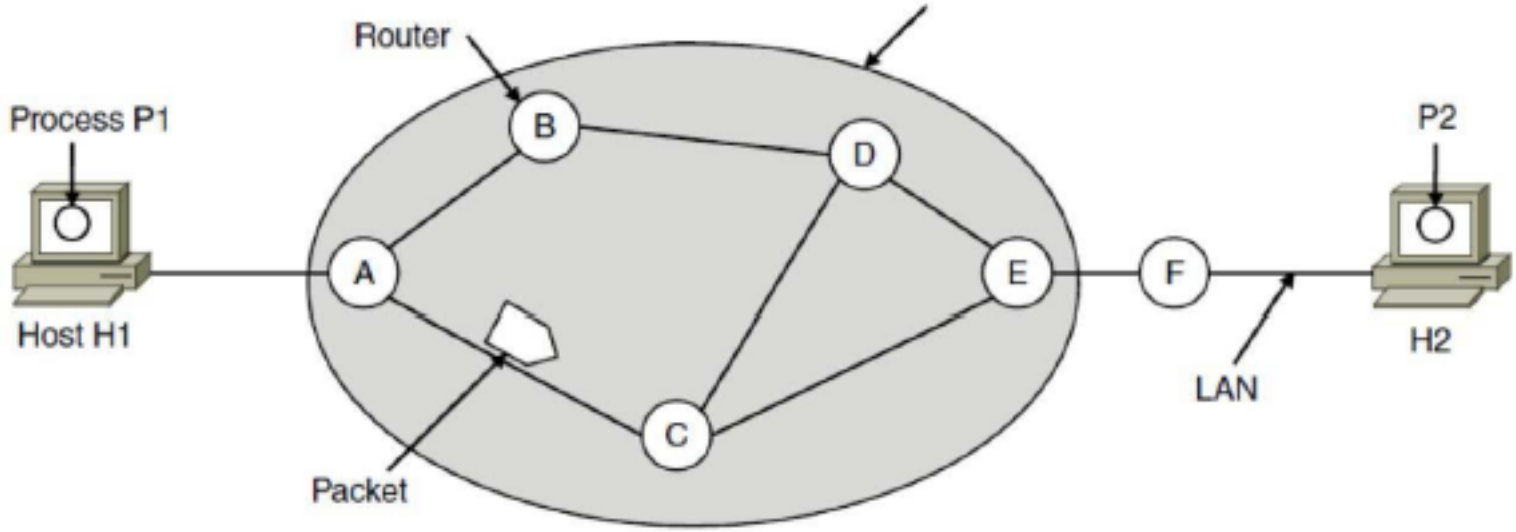

Server
fd = socket(…);
bind(fd, …);
Specify well-known port
listen(fd, …);
Begin listening
connfd = accept(fd, …);
Block until connection established
read(connfd, …);
write(connfd, …);
read(connfd, …);
EOF read
close(connfd, …);
Client
fd = socket(…);
connect(fd, …);
TCP/IP connection established
write(fd, …);
Send request
read(fd, …);
Wait for response
close(fd, …);
TCP/IP connection shutdown

Learning outcomes

On completion of this section, your are expected to demonstrate an understanding of:

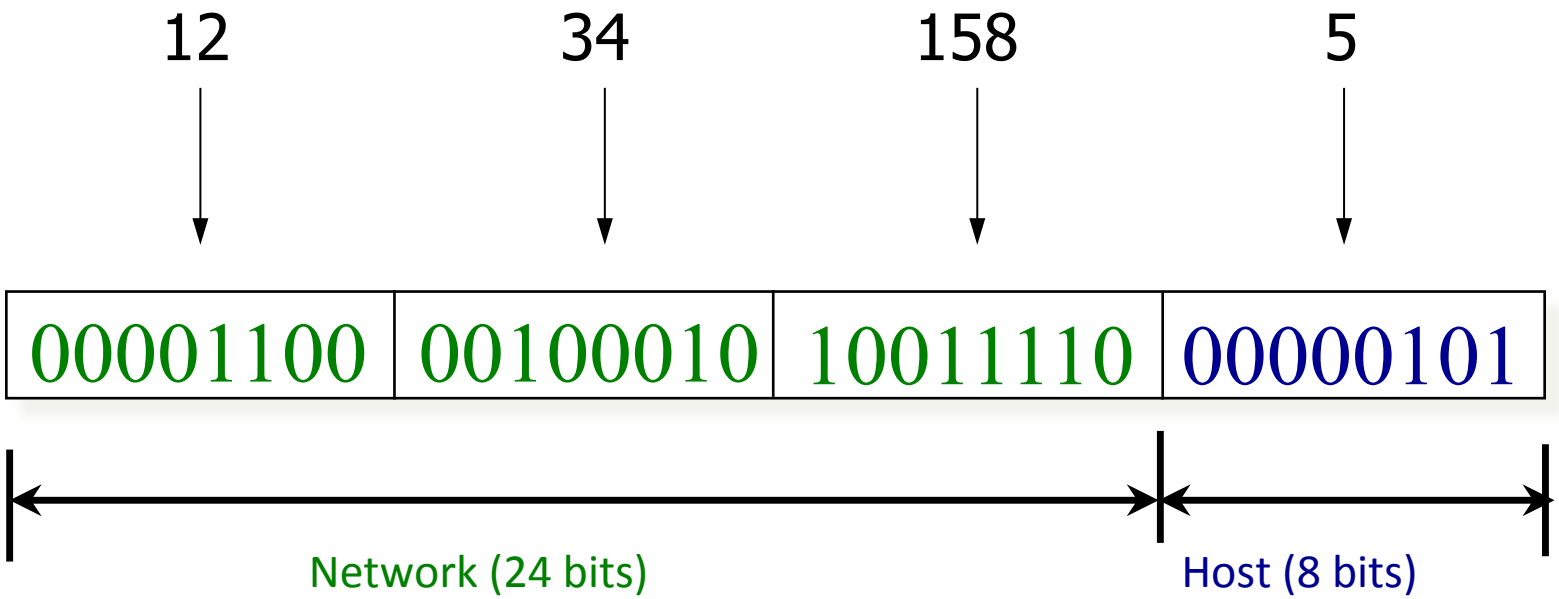
- Network layer protocols
  - IP address; subnets; routing; fragmentation
- Transport layer protocols
  - TCP, UDP
  - TCP connection; congestion control
- Format and use of protocol headers
  - how Wireshark can be used
- ARP

IP: “Best-Effort” packet delivery

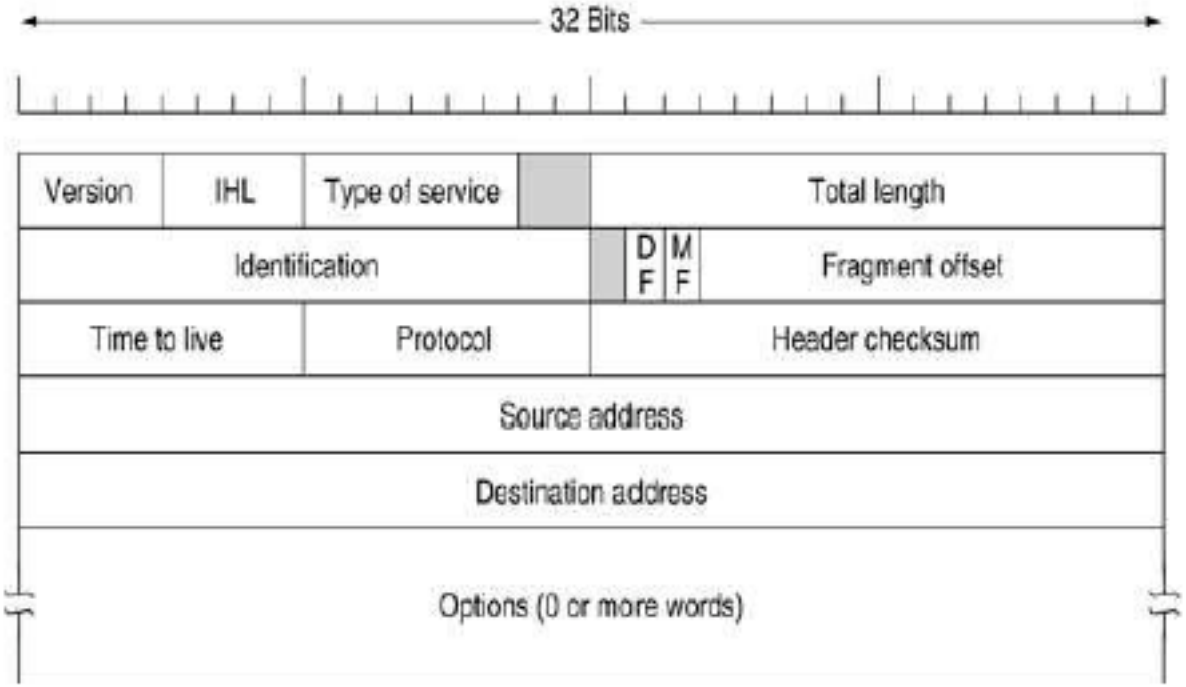


Hierarchical addressing

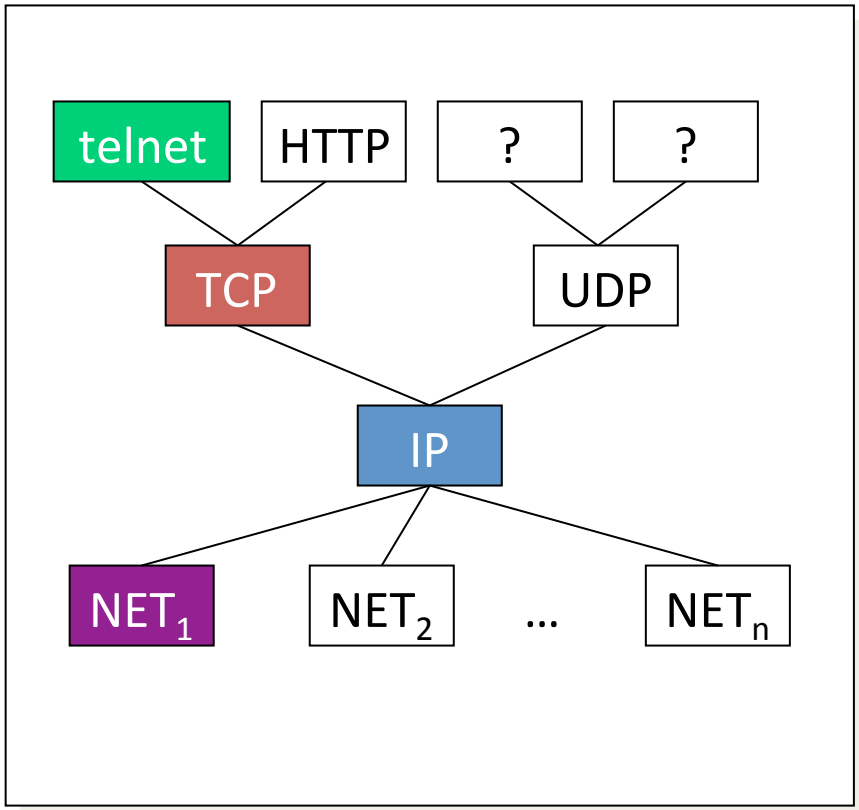
12.34.158.0/24 is a 24-bit **prefix**

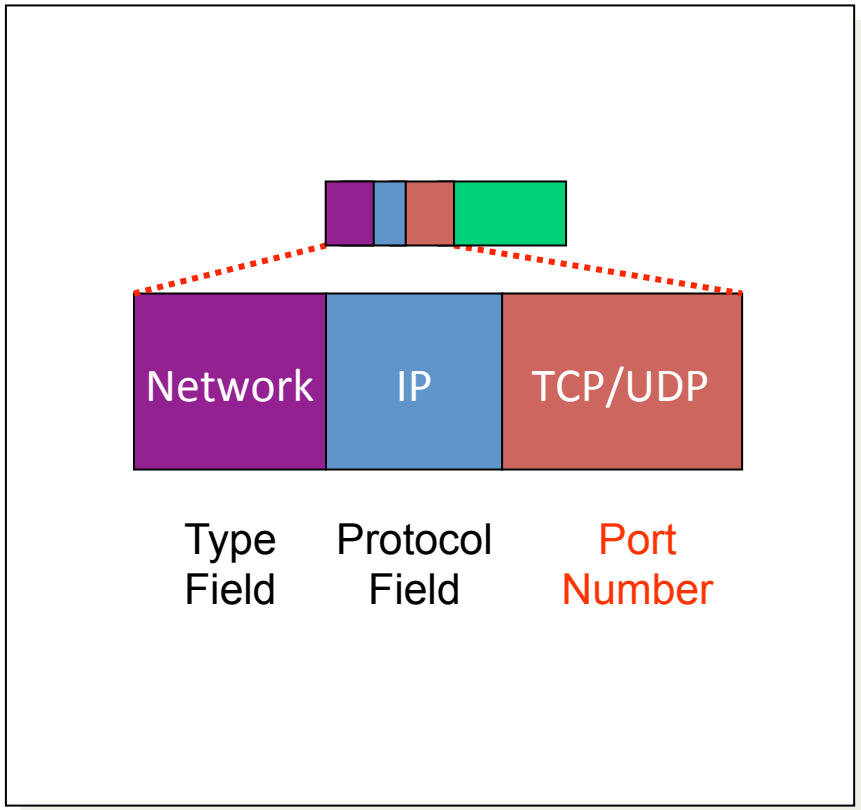


IP packet header

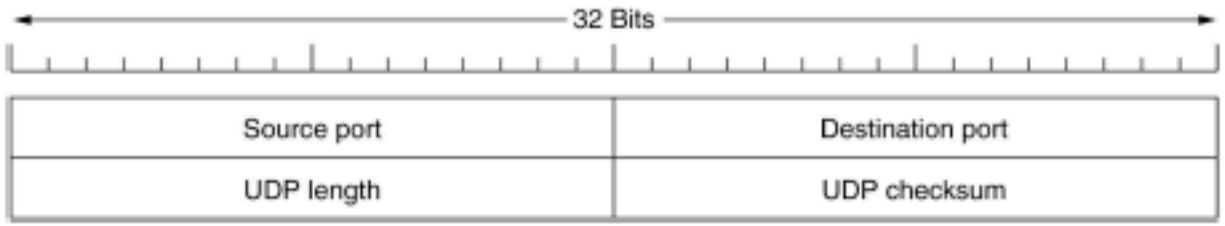


Protocol demultiplexing

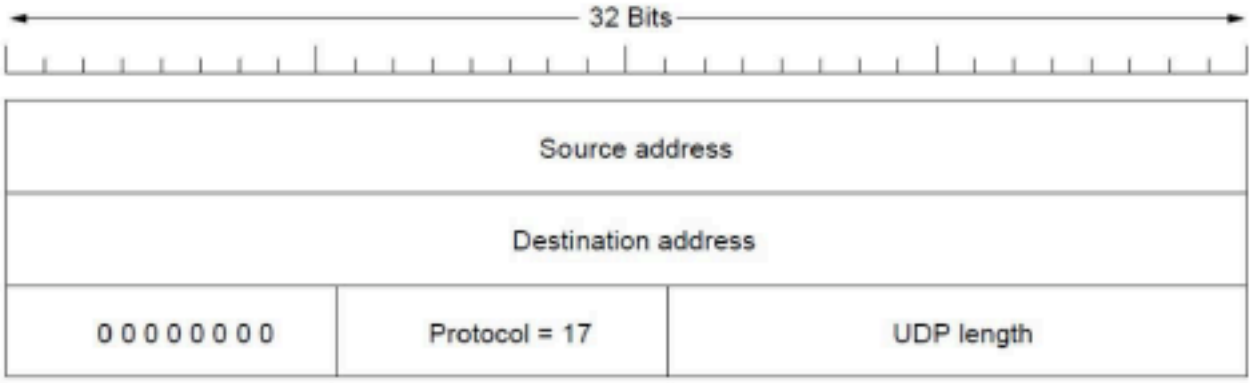




UDP header

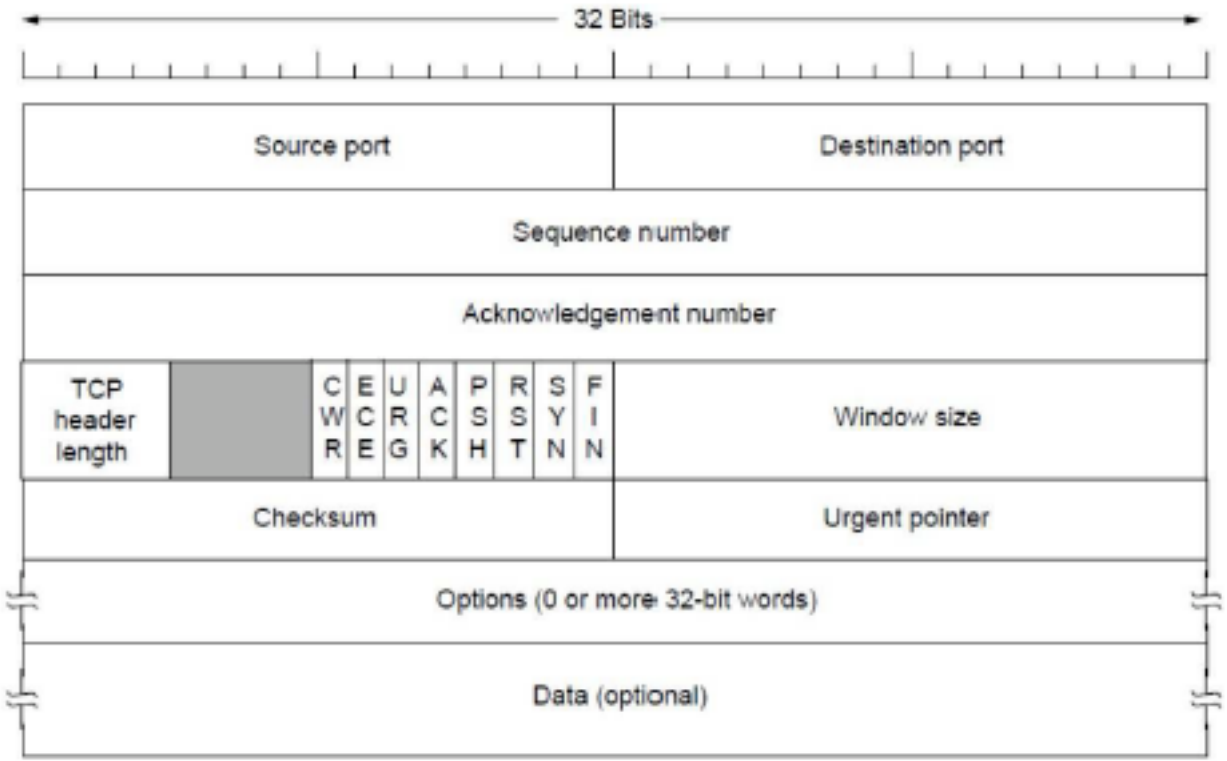


(top) UDP header

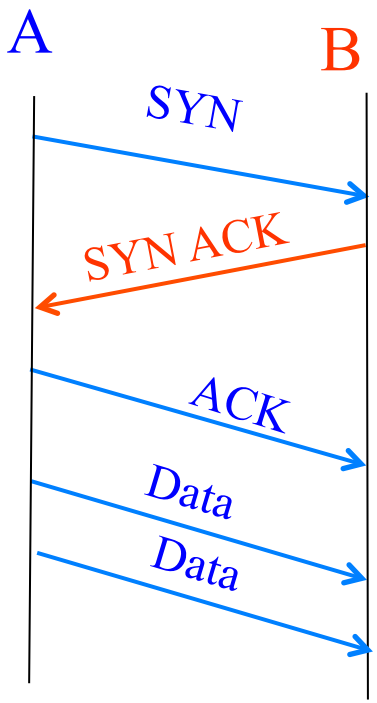


(bottom) The IPv4 pseudoheader included in the UDP checksum.

TCP header


32 Bits
Source port
Destination port
Sequence number
Acknowledgement number
TCP header length
CWR
ECE
URG
ACK
PSH
RST
SYN
FIN
Window size
Checksum
Urgent pointer
Options (0 or more 32-bit words)
Data (optional)

TCP connection



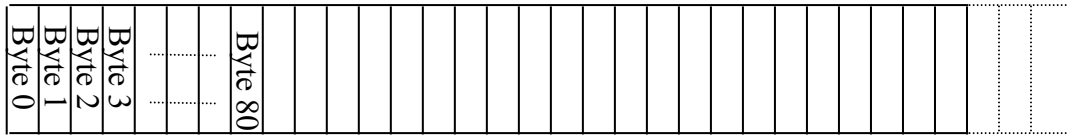
Host A sends a SYNchronize (open) to the host B

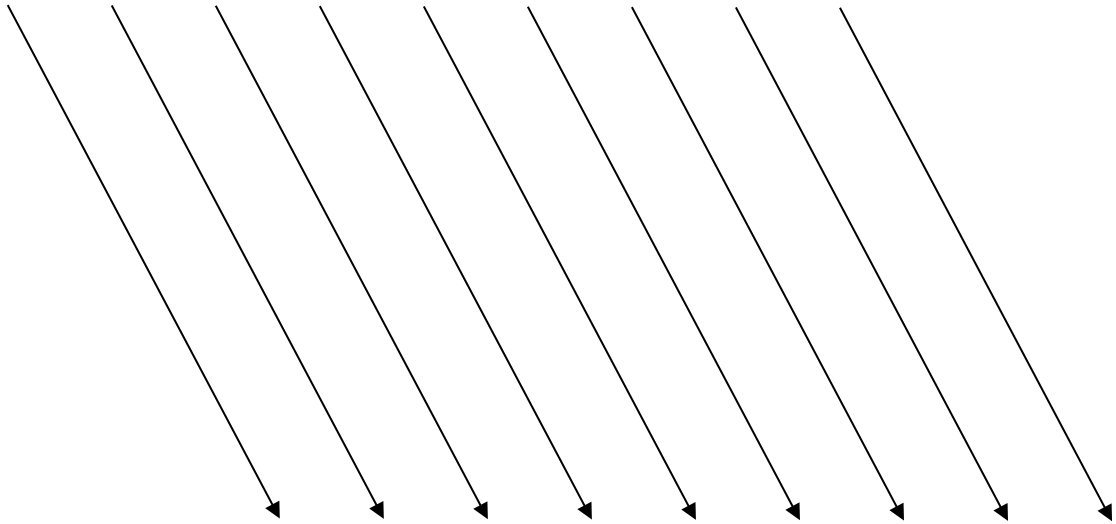
Host B returns a SYN ACKnowledgment (SYN ACK)

Host A sends an ACK to acknowledge the SYN ACK

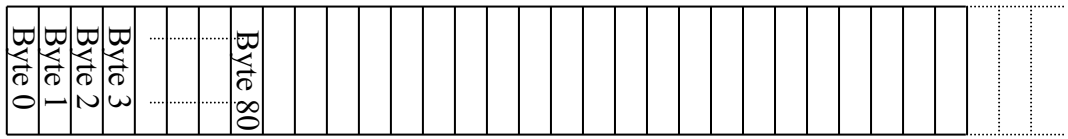
TCP "stream of bytes"

Host A

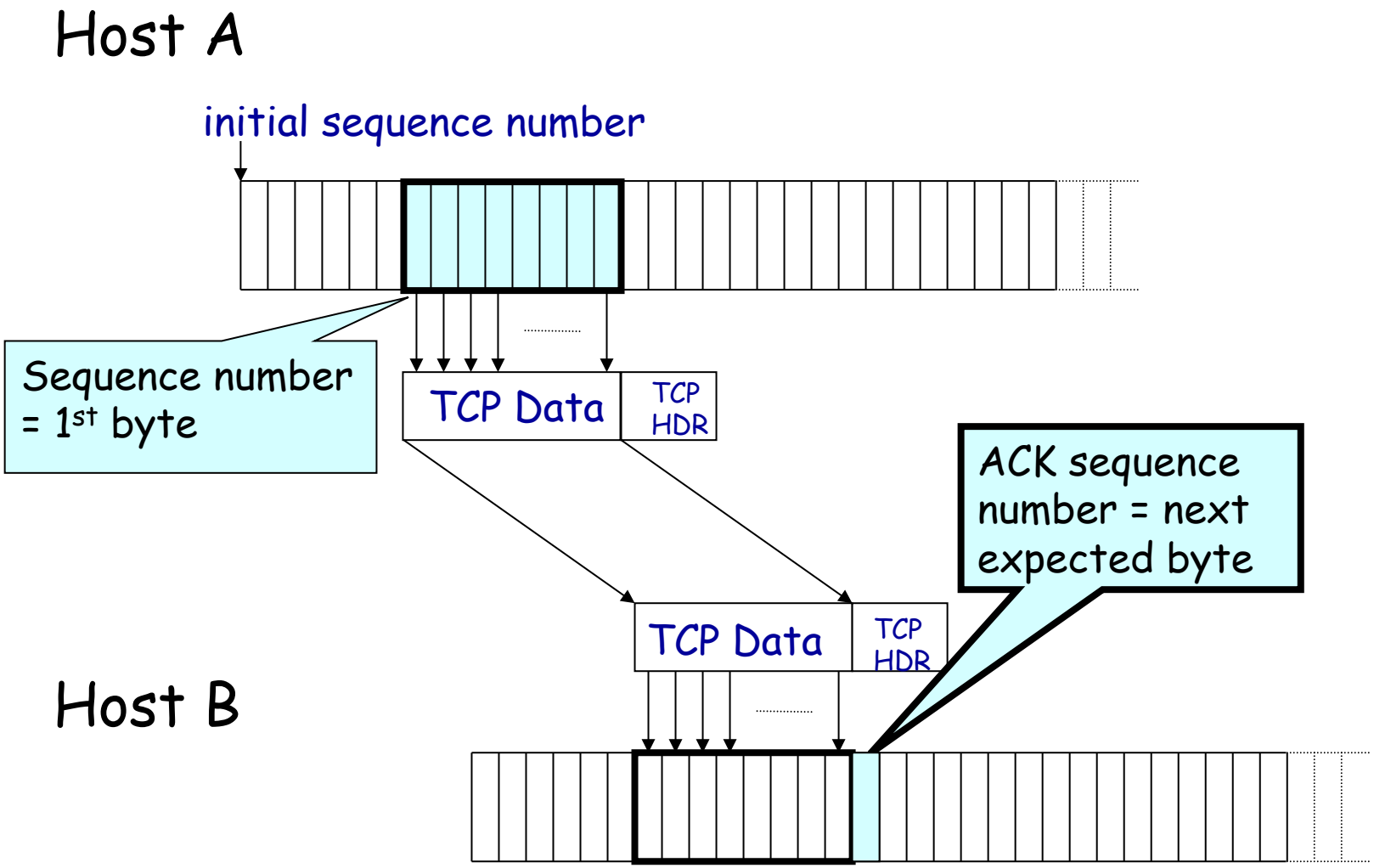




Host B



TCP acknowledgments



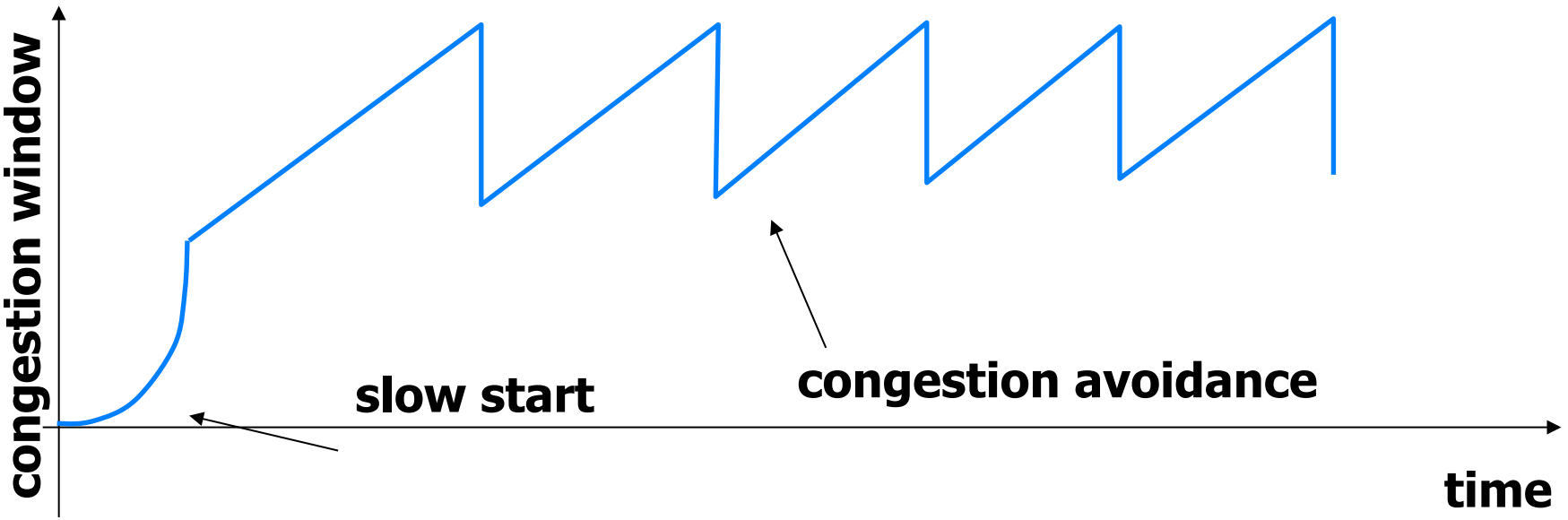
TCP congestion control

Flow control: window-based

- Sender limits number of outstanding bytes (window size)
- *Receiver window* ensures data does not overflow receiver

Congestion control: adapting to packet losses

- *Congestion window* tries to avoid overloading the network (increase with successful delivery, decrease with loss)
- TCP connection starts with small initial congestion window



Learning outcomes

On completion of this section, your are expected to demonstrate an understanding of:

- Difference between a protocol and a service
- DNS (domain name system)
- Email service and protocols
  - POP3, SMTP
- HTTP
  - including persistent connections; pipelining
  - role of proxy servers

Domain Name System (DNS)

Properties of DNS

- Hierarchical name space divided into zones
- Translation of names to/from IP addresses
- Distributed over a collection of DNS servers

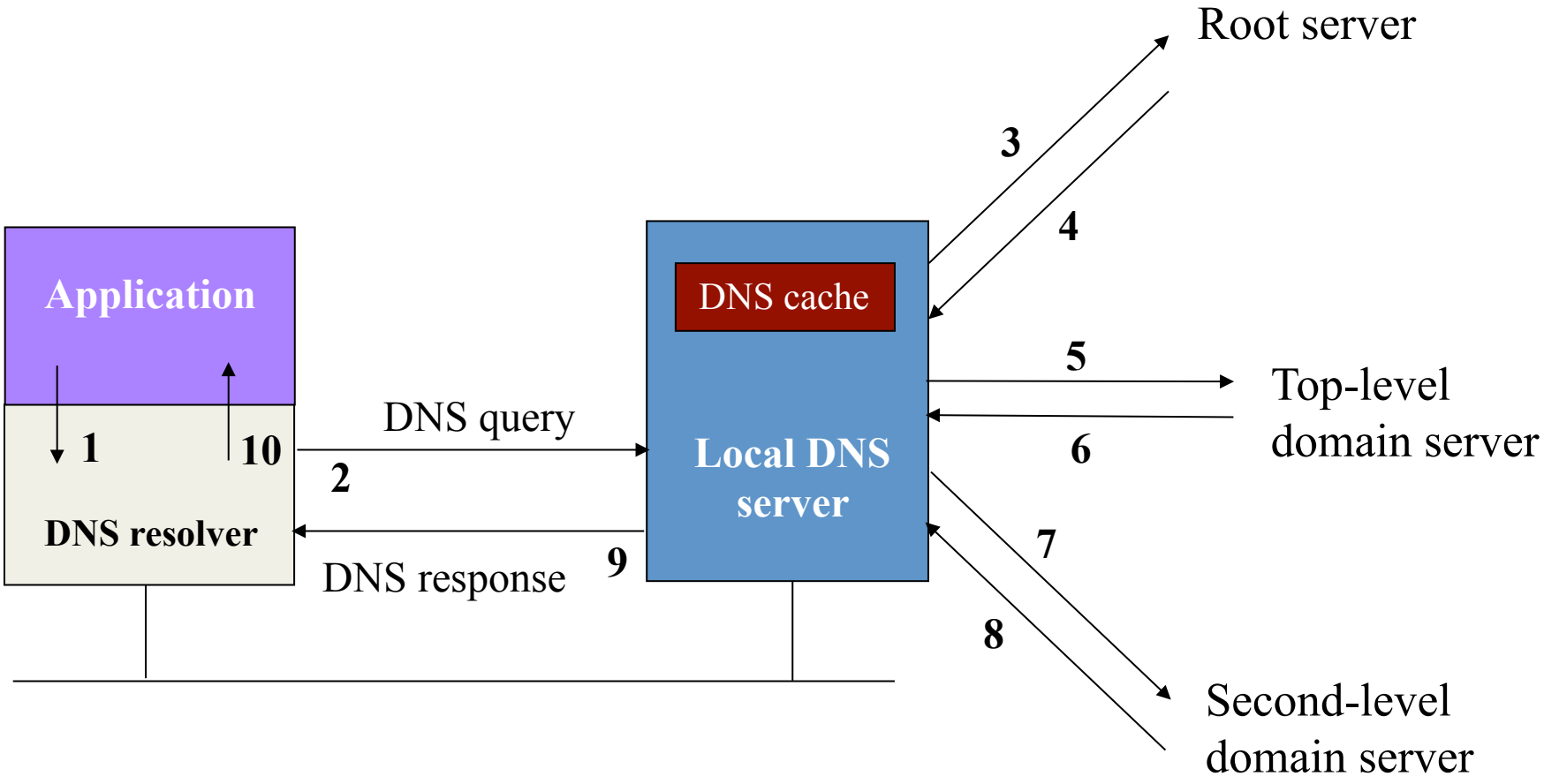
Client application

- Extract server name (e.g., from the URL)
- Invoke system call to trigger DNS resolver code
  - E.g., *gethostbyname()* on "www.csse.unimelb.edu.au"

Server application

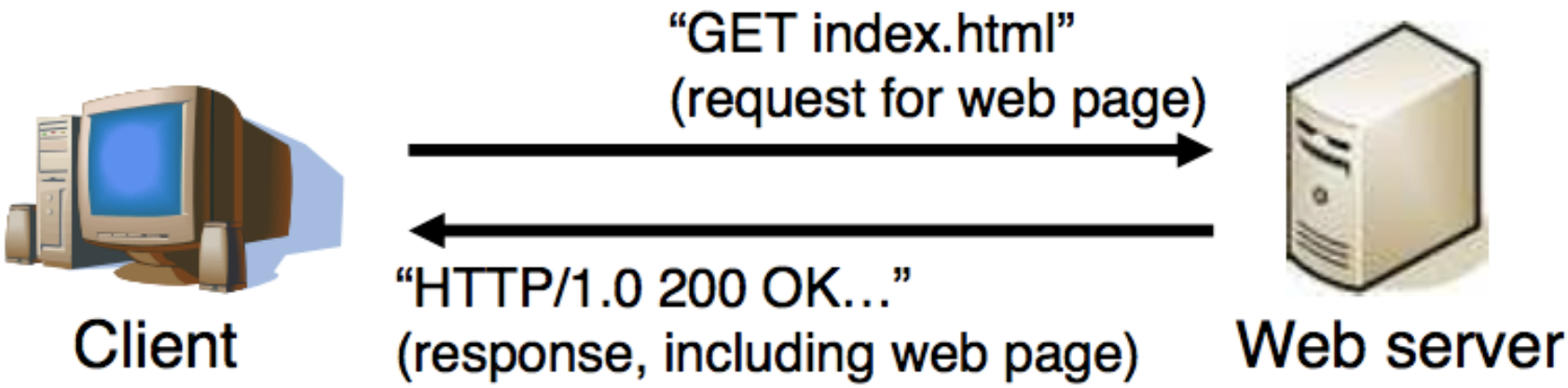
- Extract client IP address from socket
- Optionally invoke system call to translate into name
  - E.g., *gethostbyaddr()* on "128.250.37.110"

DNS Resolver and Local DNS Server



Caching based on a time-to-live (TTL) assigned by the DNS server responsible for the host name to reduce latency in DNS translation.

HTTP example: Client – Server app



Using HTTP

```
GET /index.html HTTP/1.0
Host: www.unimelb.edu.au
```

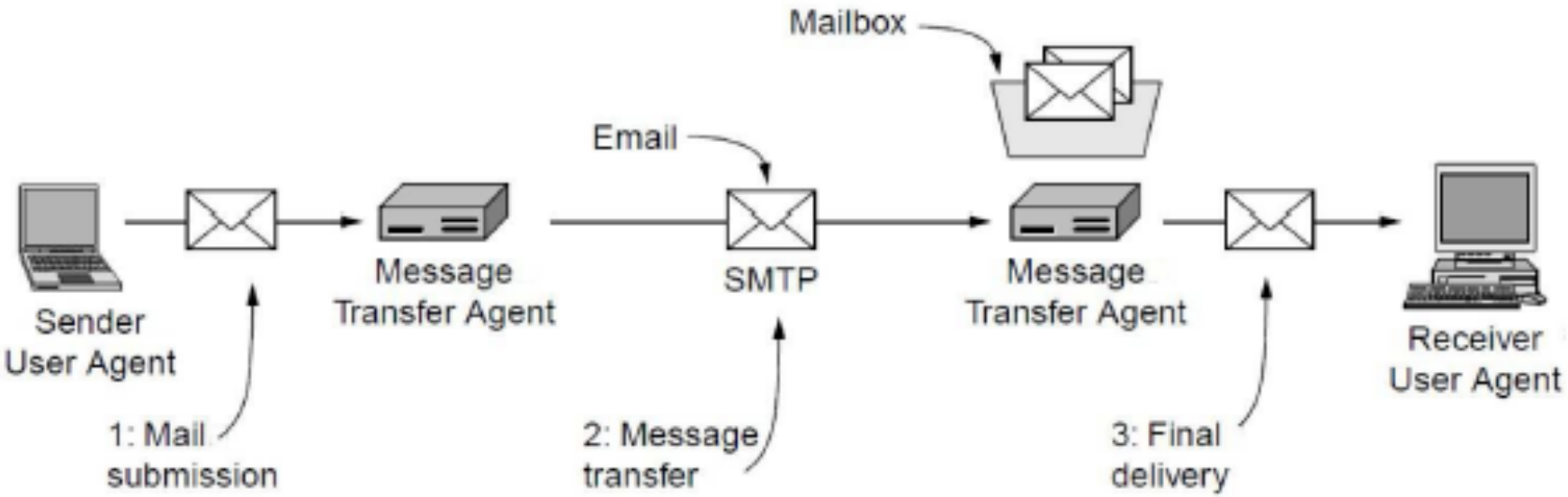
Request

Response

```
HTTP/1.0  200
Date:
Server:
Location:
Content-Length:
Content-Type:

….details
```

Email protocols



Lab tasks

- SVN repo, shell (fork, exec), using Linux
- Buffer overflow (exploring the use of gdb)
  - memory address
- File systems
  - chmod, navigating directories
- Threads (including synchronization using mutex)
  - producer/consumer problem
- Socket programming
  - system calls (socket, bind, listen, connect)
- Wireshark
  - Transport/Network headers