Operating Systems
Design and Function

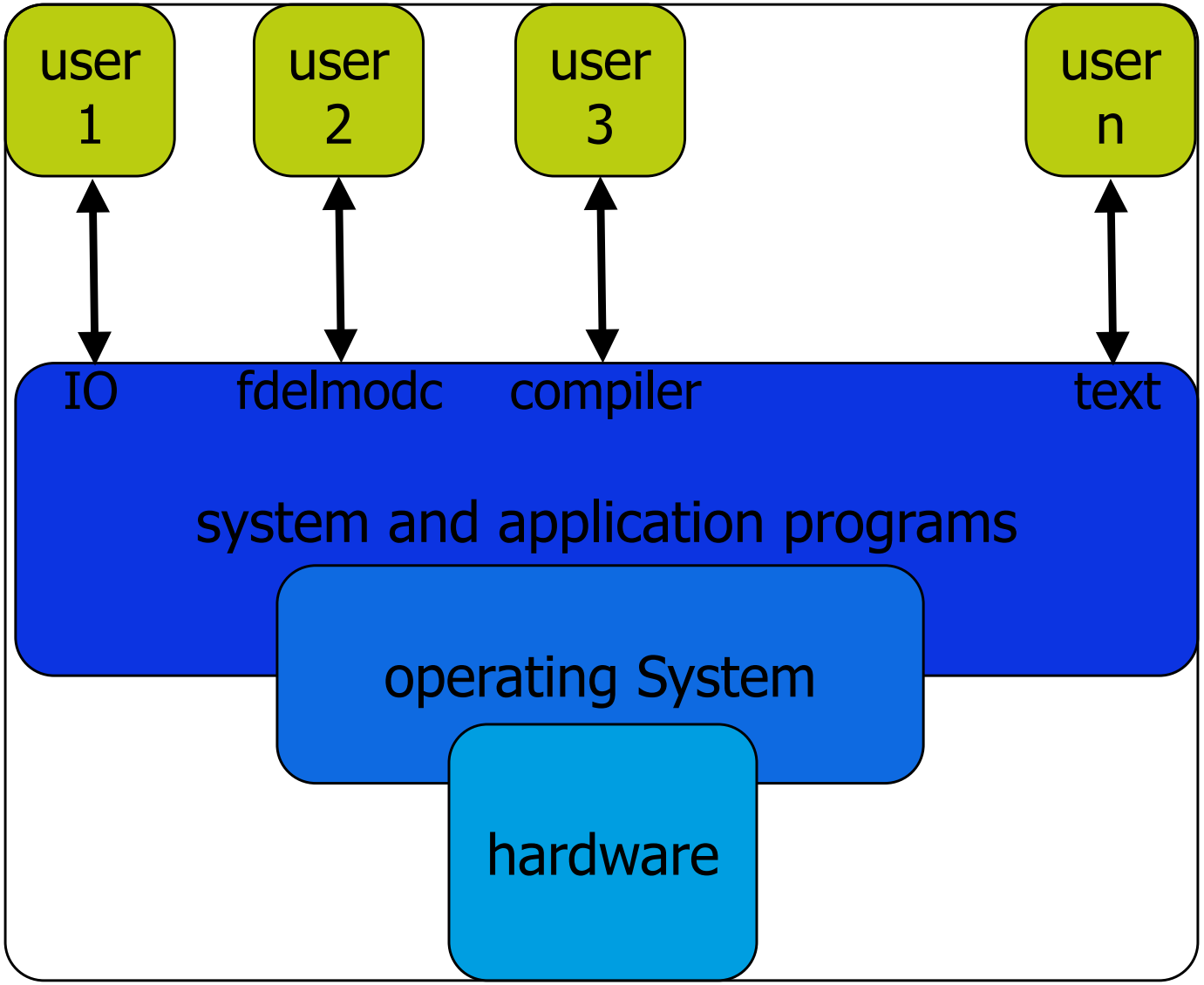
Jan Thorbecke

Operating System (OS) Contents

- What is and does and OS
- system and kernel functions
- **Processes**
- **File systems**
- Storage
- Unix and Linux
- Windows

- Handy commands in Linux

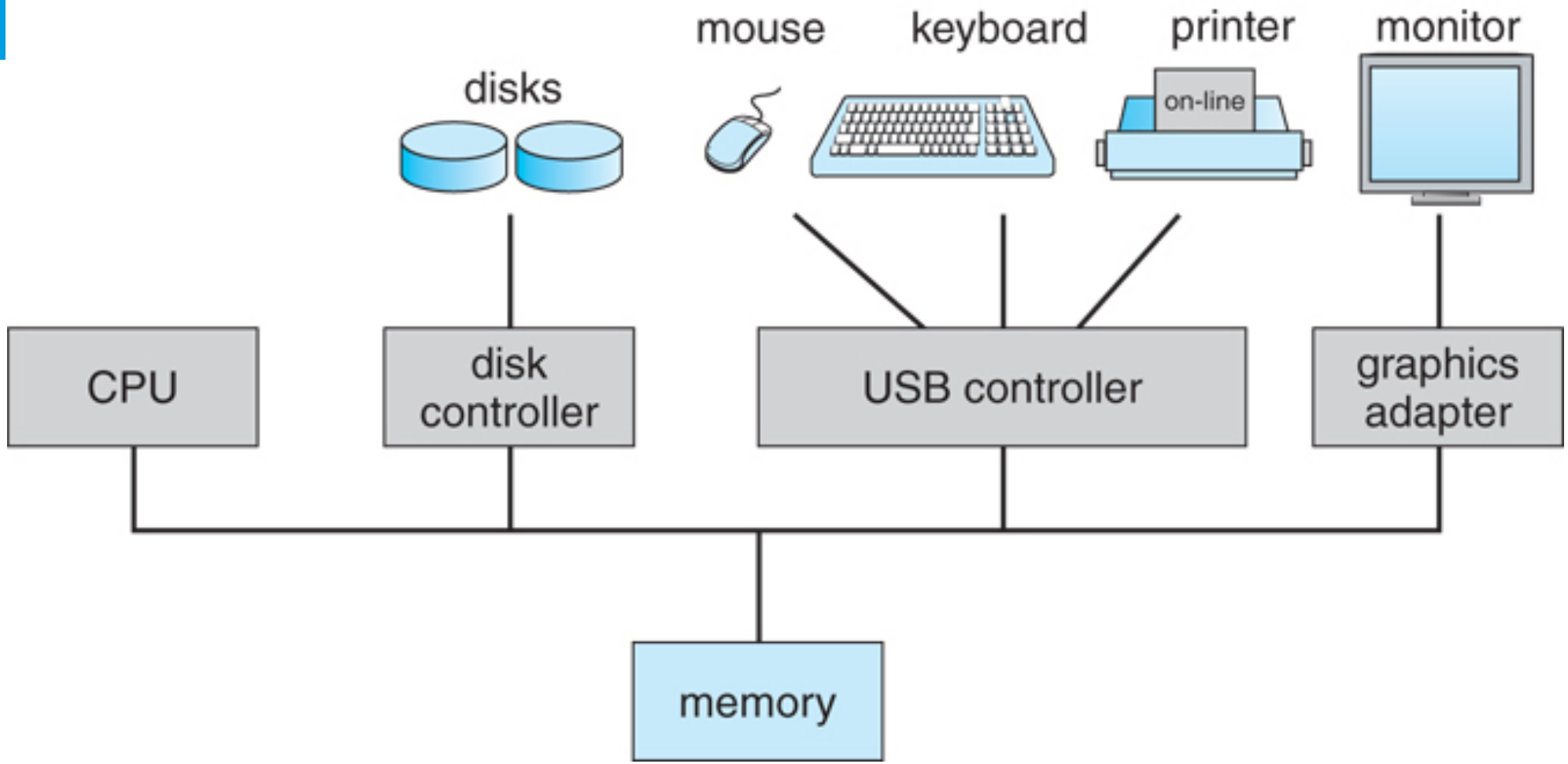
Four Components Computer System



Operating System Definition

- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer
- OS for **protection and security**
  - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other

Managing resources

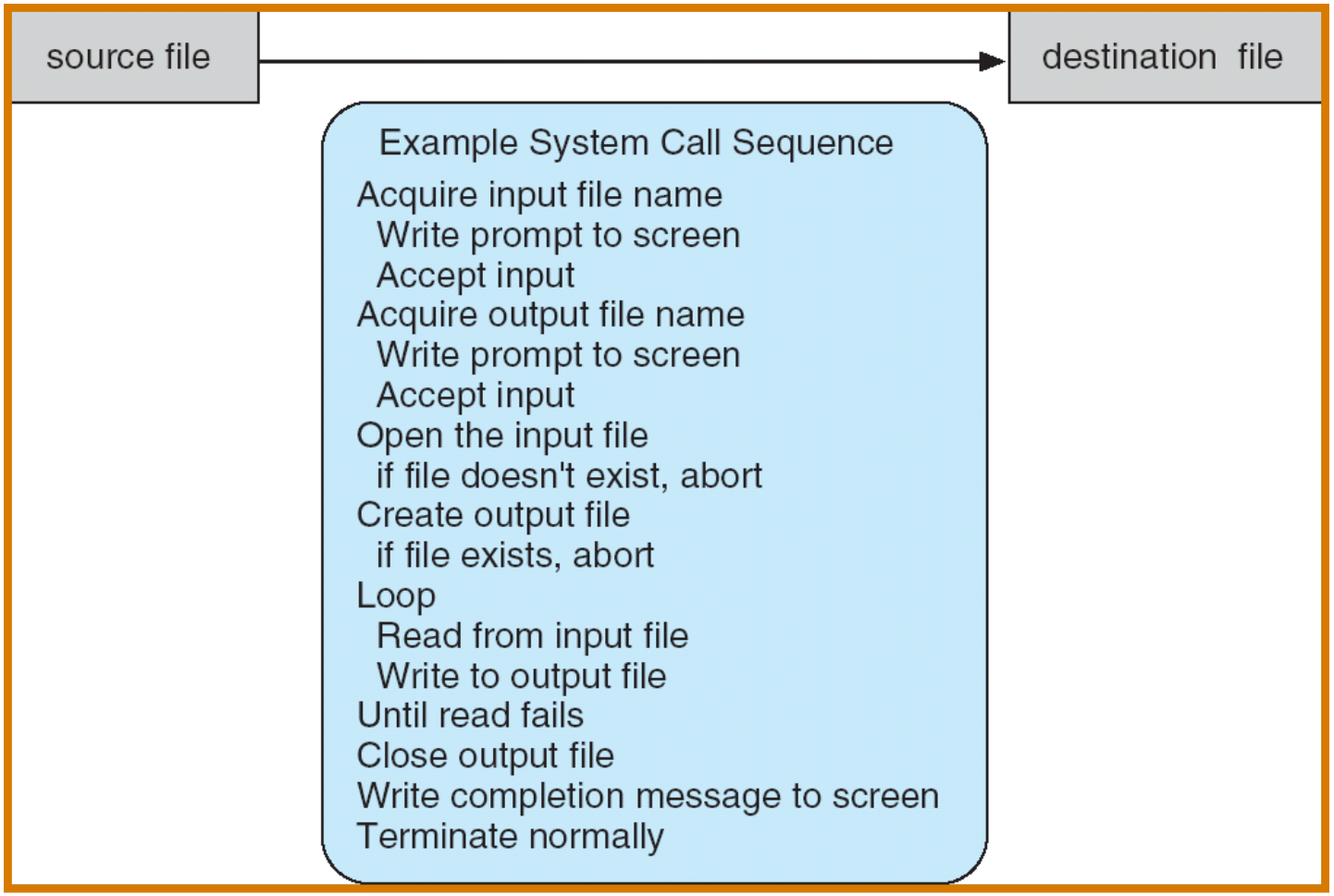


System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use
- Three most common APIs are
  - Win32 API for Windows,
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and
  - Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?

Example of System Calls

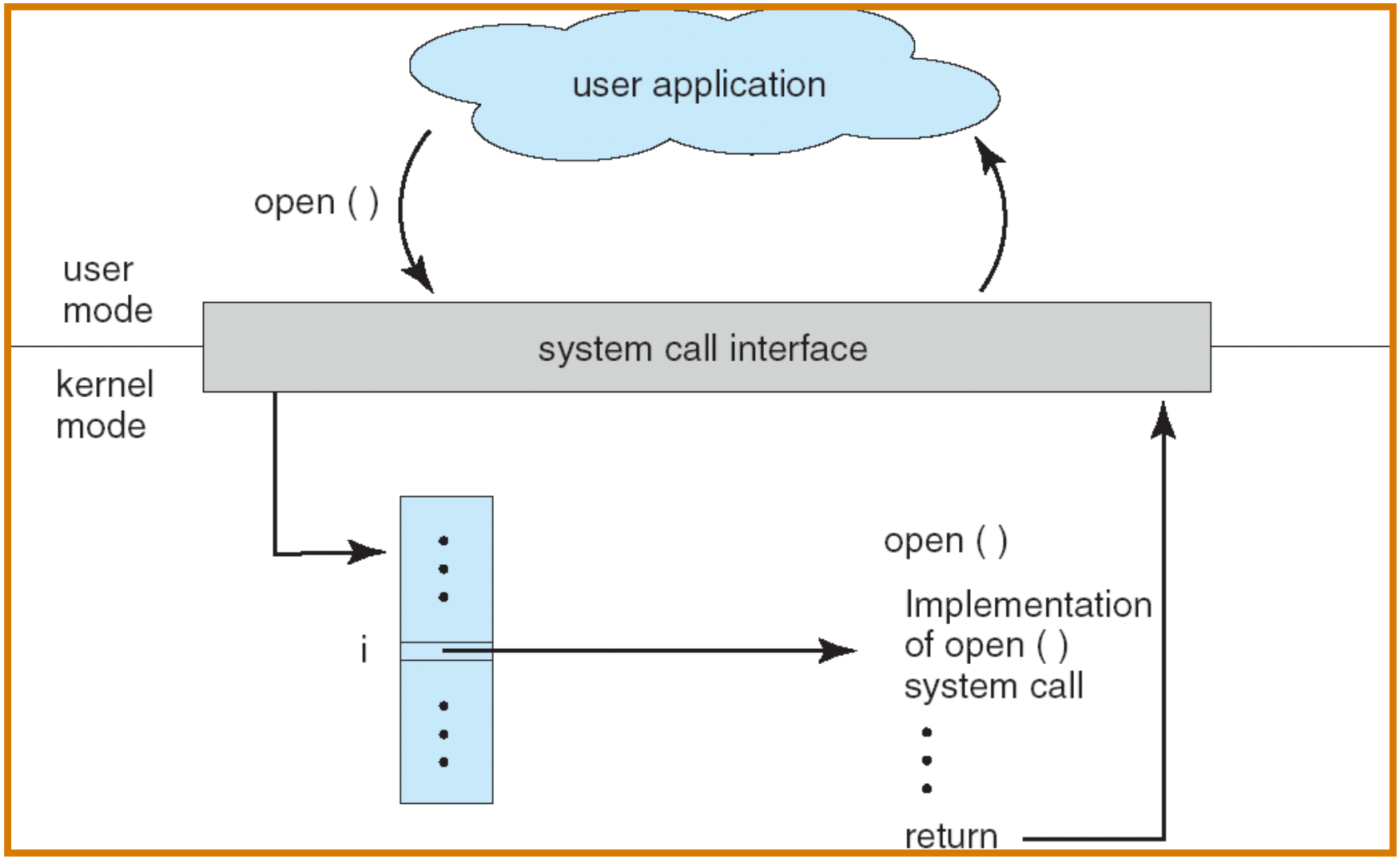
- System call sequence to copy the contents of one file to another file



System Call Implementation

- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
  - look for unistd.h in linux or http://www.linfo.org/system_call_number.html
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - **Most details of OS interface hidden from programmer by API**
    - Managed by run-time support library (set of functions built into libraries included

API – System Call – OS Relationship



Types of System Calls

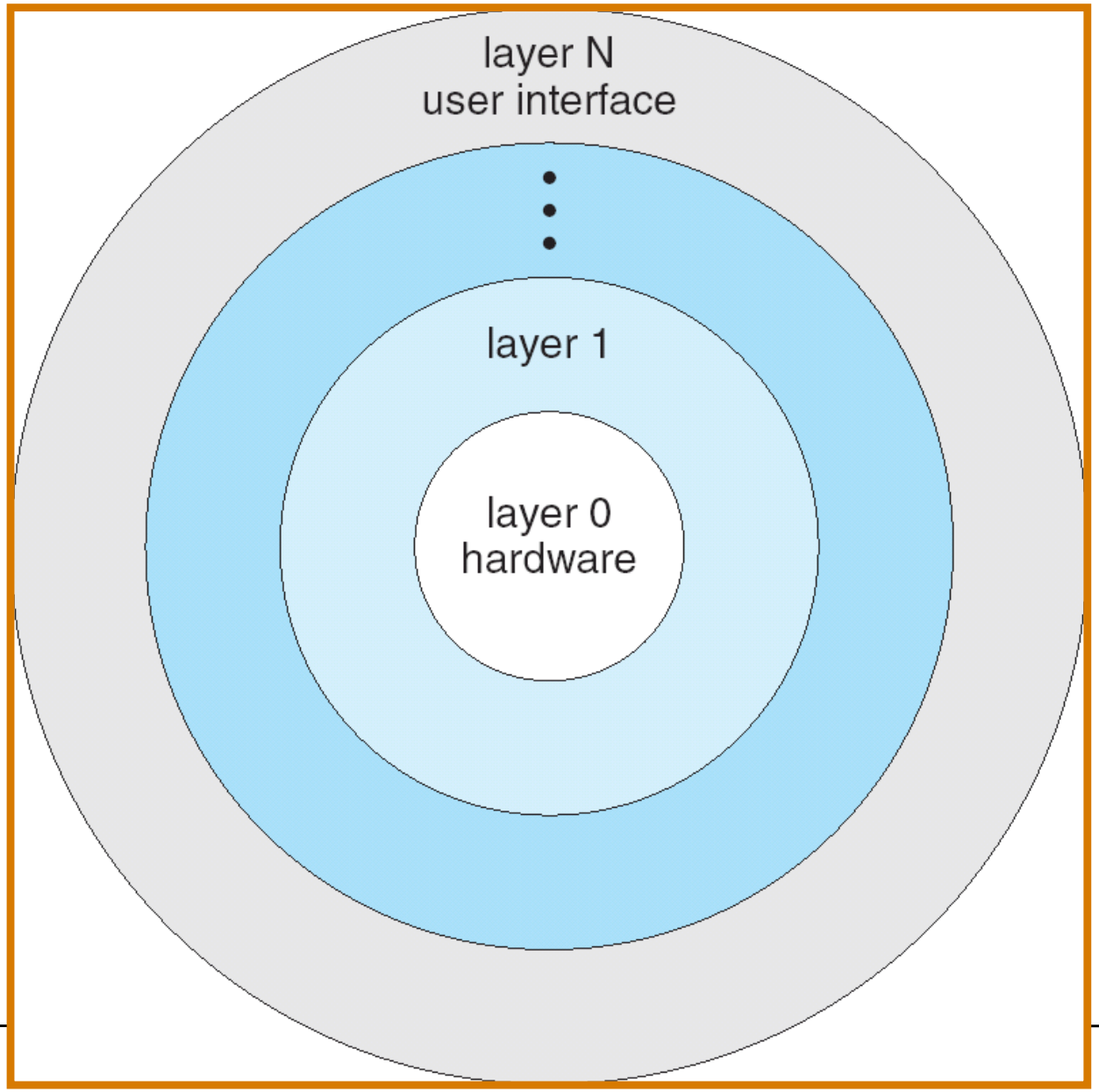
- Process control
  - load execute, wait, allocate/free memory, create process, ..
- File management
  - create, open, close, read, write,
- Device management
  - request device, read write, get set, attach detach
- Information maintenance
  - get set time/date, process file attributes
- Communications
  - create connection, send receive, transfer,

OS Implementations

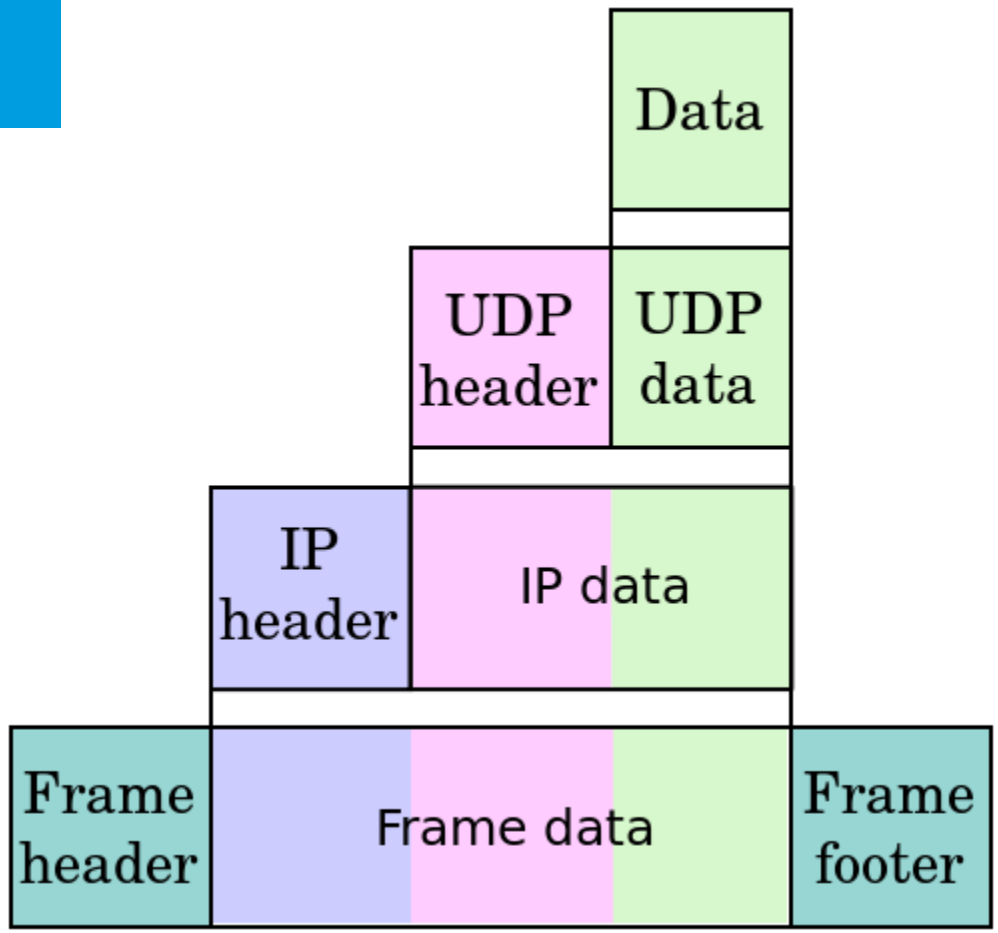
Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

Layered Operating System



Example: layered TCP/IP stack



Application

Application layer protocols:
DHCP, DNS, FTP, HTTP, POP3, SMTP, TELNET, SSL, ...

Transport

Transport layer protocols:
TCP, UDP, ...

Internet

Network layer protocols:
IP, ICMP, IGMP, ARP, RARP, ...

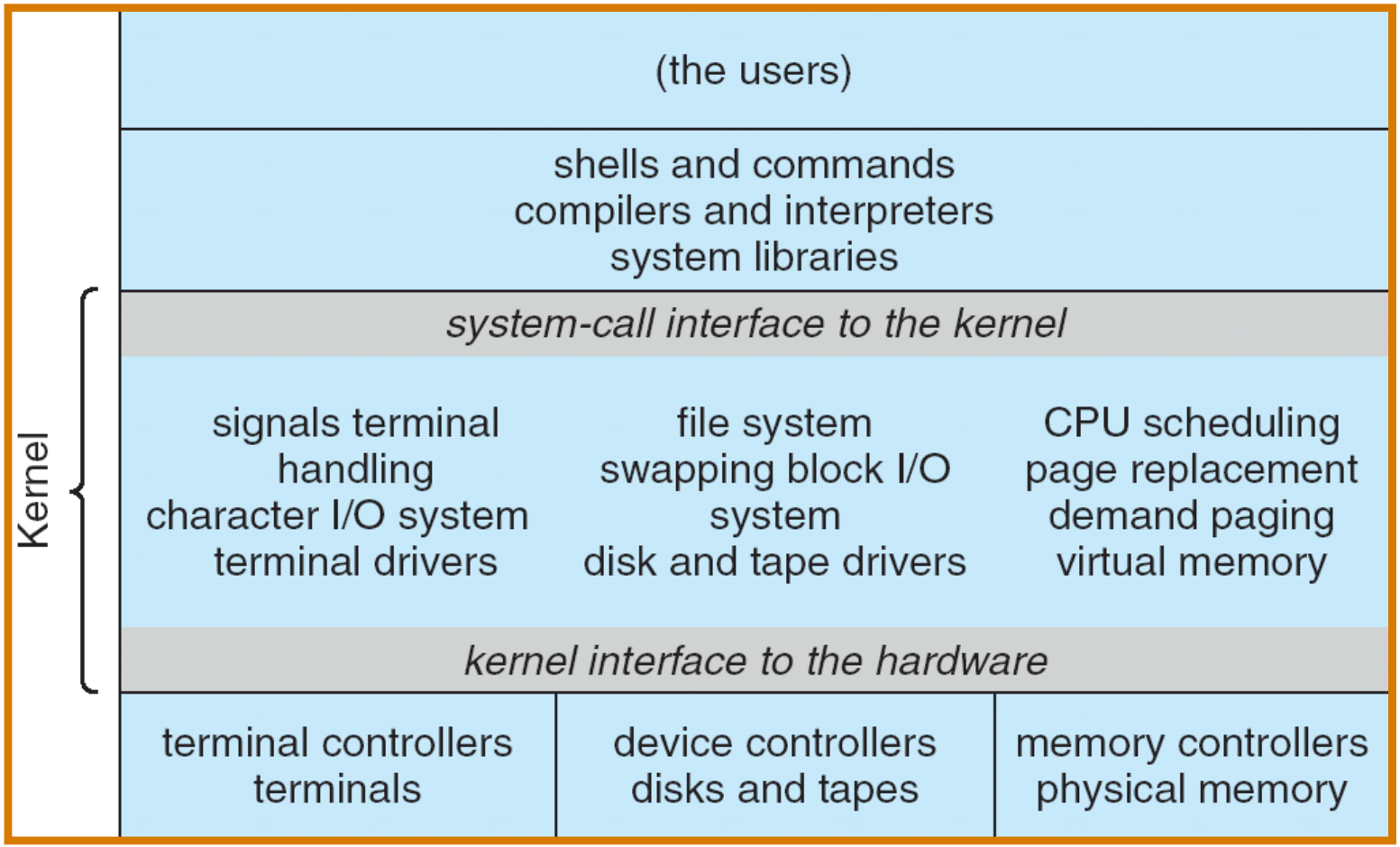
Link

Link layer protocols:
Ethernet (IEEE 802.3), ...

UNIX

- The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

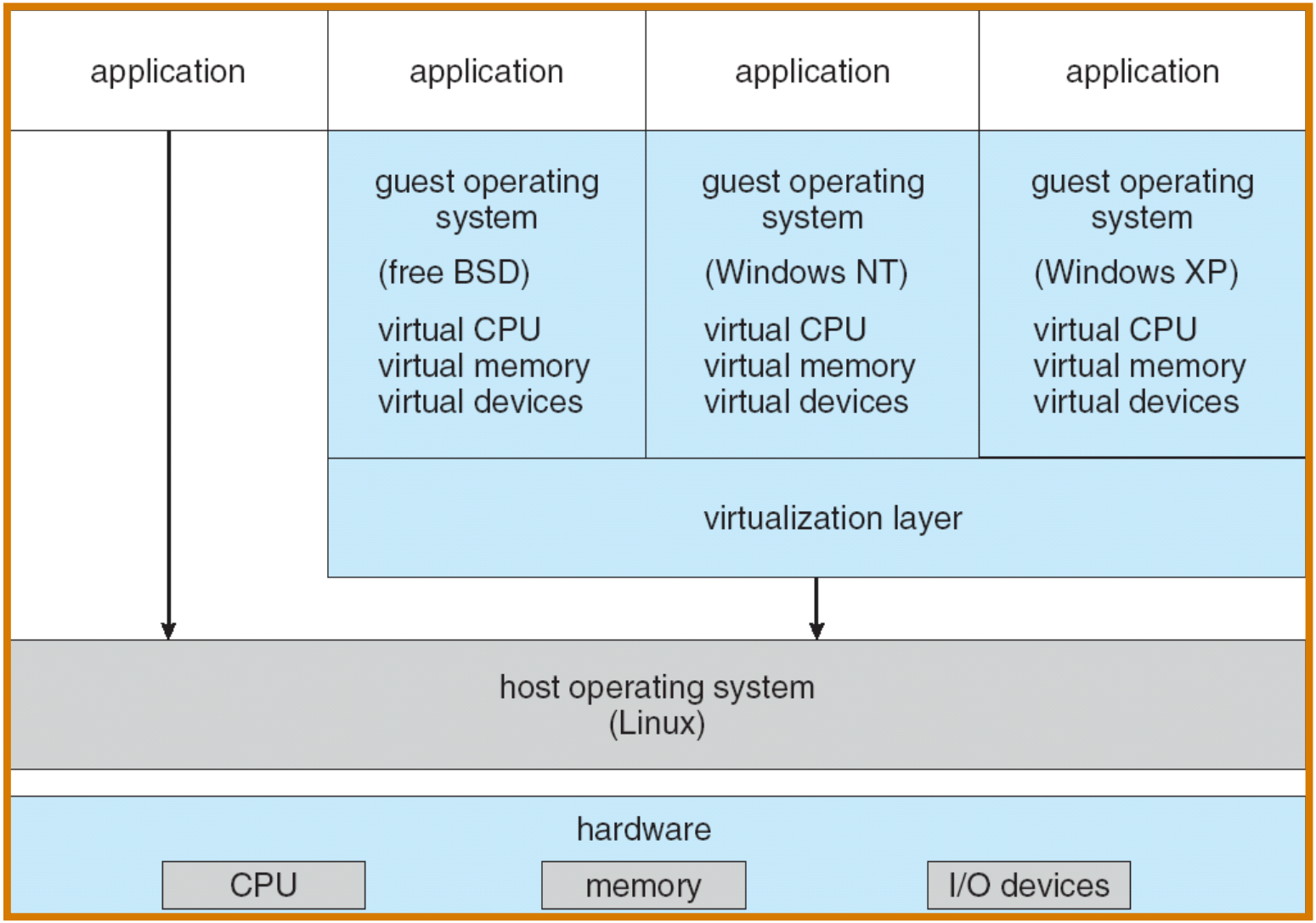
UNIX System Structure



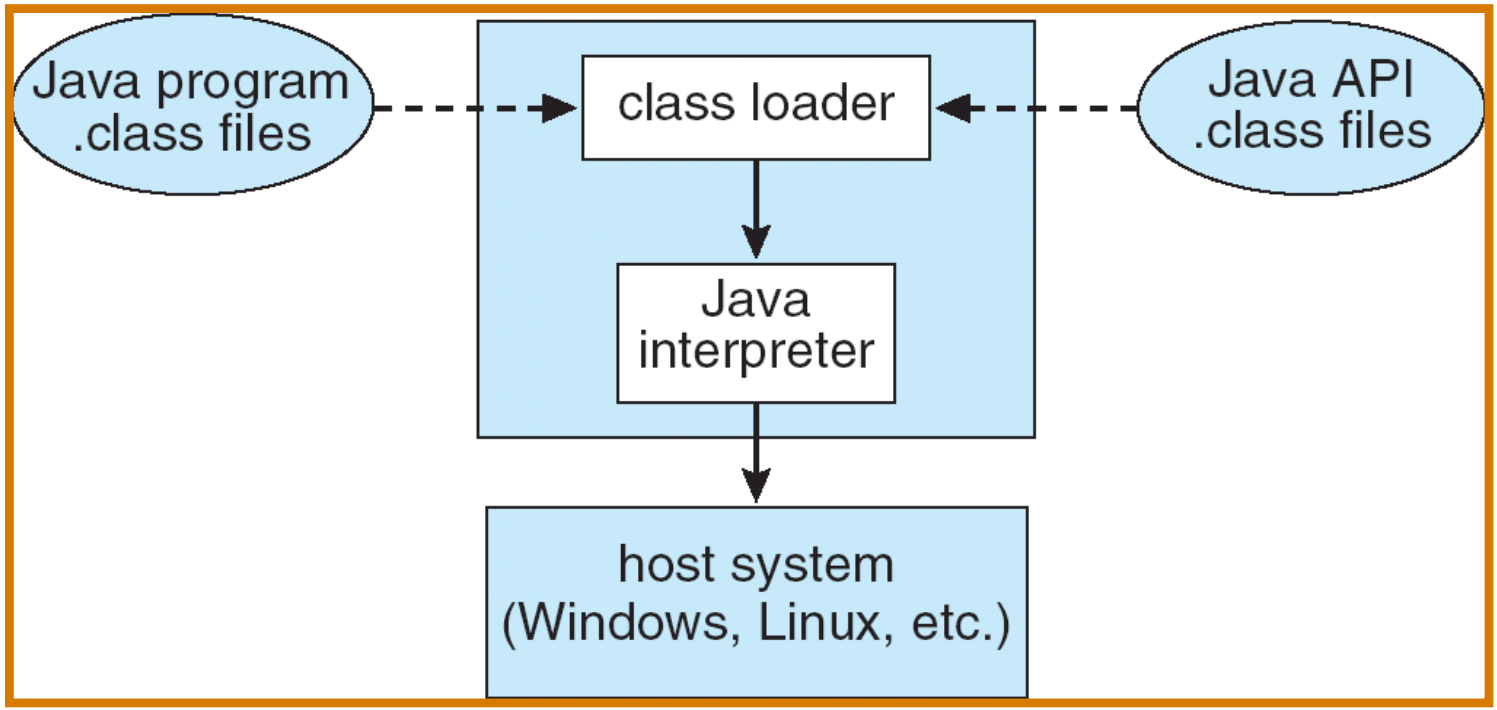
Virtual Machines

- A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface *identical* to the underlying bare hardware
- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory

Virtual Machines



The Java Virtual Machine



Service handling by OS

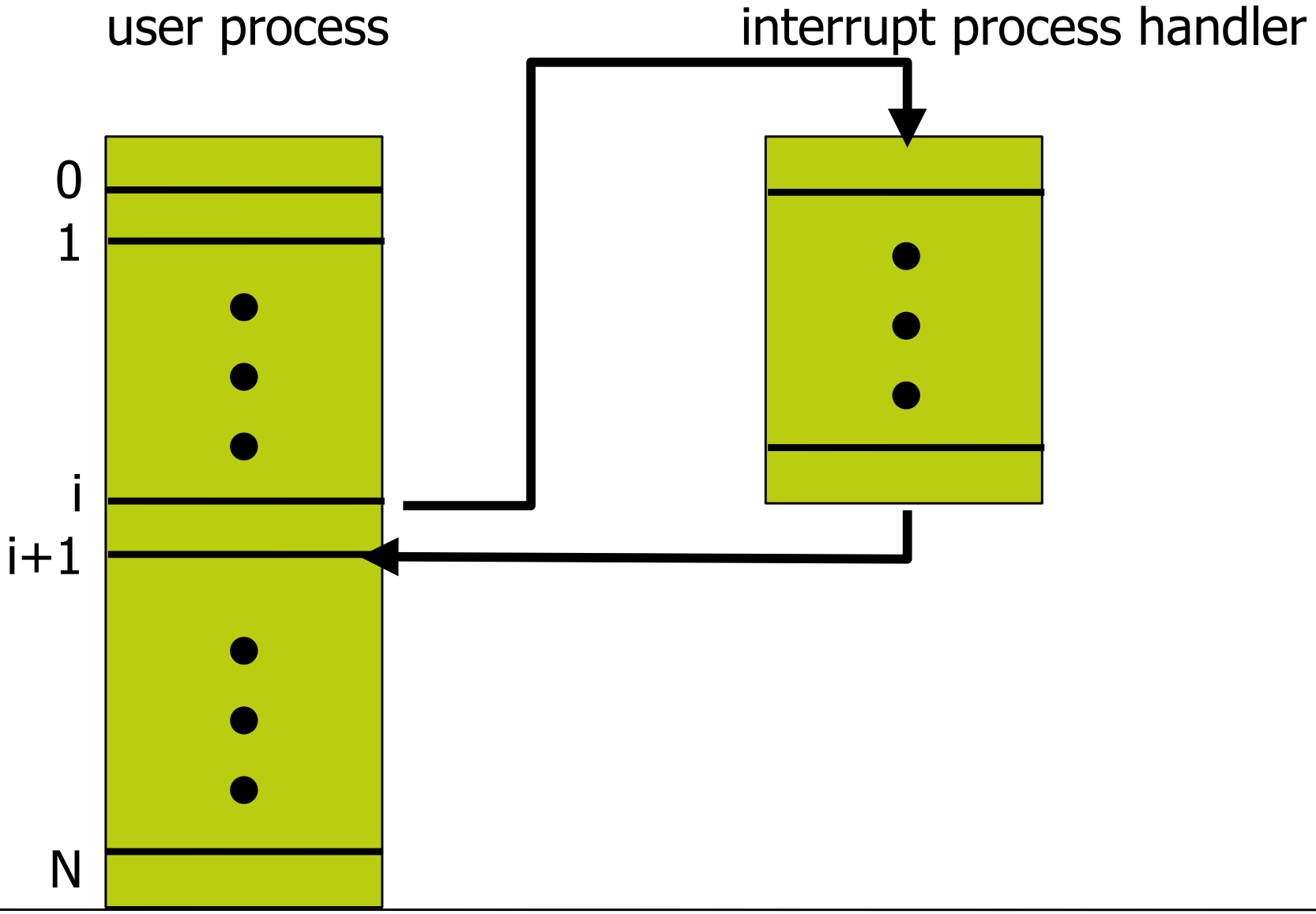
- External devices (keyboard, printer, mouse, harddisk, …) ask for a service from the OS

- How does the OS know it has something to do?
  - **Polling**
    Operating system periodically checks each device if it needs a service
    - extra overhead, while it might not be needed
  - **Interrupt**
    Each device can signal the OS. When interrupt signalled, processor executes a routine called an interrupt handler to deal with the interrupt
    - No overhead when no requests pending

Polling vs Interrupts

- "Polling is like picking up your phone every few seconds to see if you have a call. Interrupts are like waiting for the phone to ring."
- Interrupts win if processor has other work to do and event response time is not critical.
- Polling can be better if processor has to respond to an event ASAP.
  - May be used in device controller that contains dedicated secondary processor.

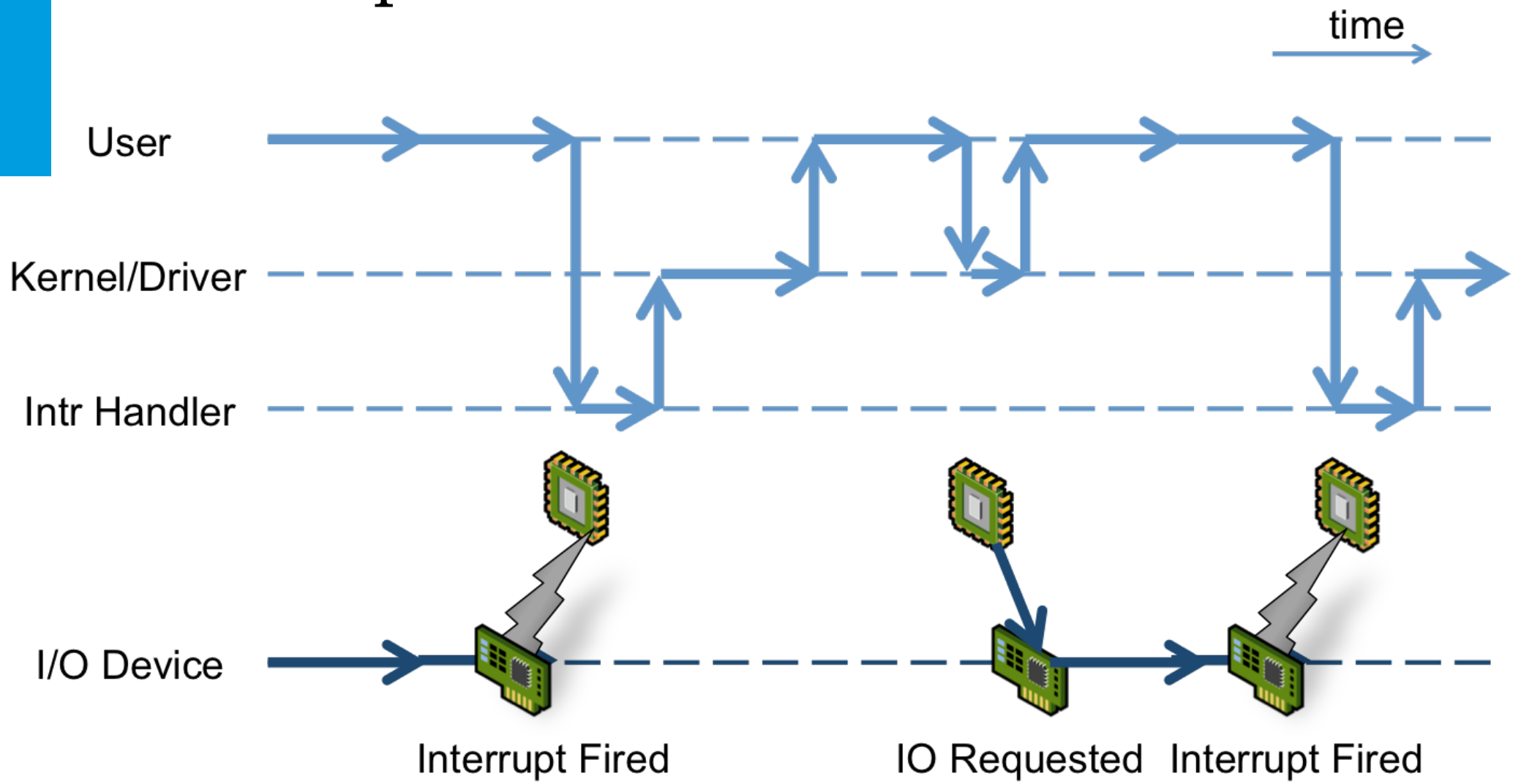
Transfer of Control via Interrupt



Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the *interrupt vector*, which contains the addresses of all the service routines.
- Interrupt architecture must save the address of the interrupted instruction.
- Incoming interrupts are *disabled* while another interrupt is being processed to prevent a *lost interrupt*.
- A *trap* is a software-generated interrupt caused either by an error or a user request.
- Most operating system are *interrupt* driven.

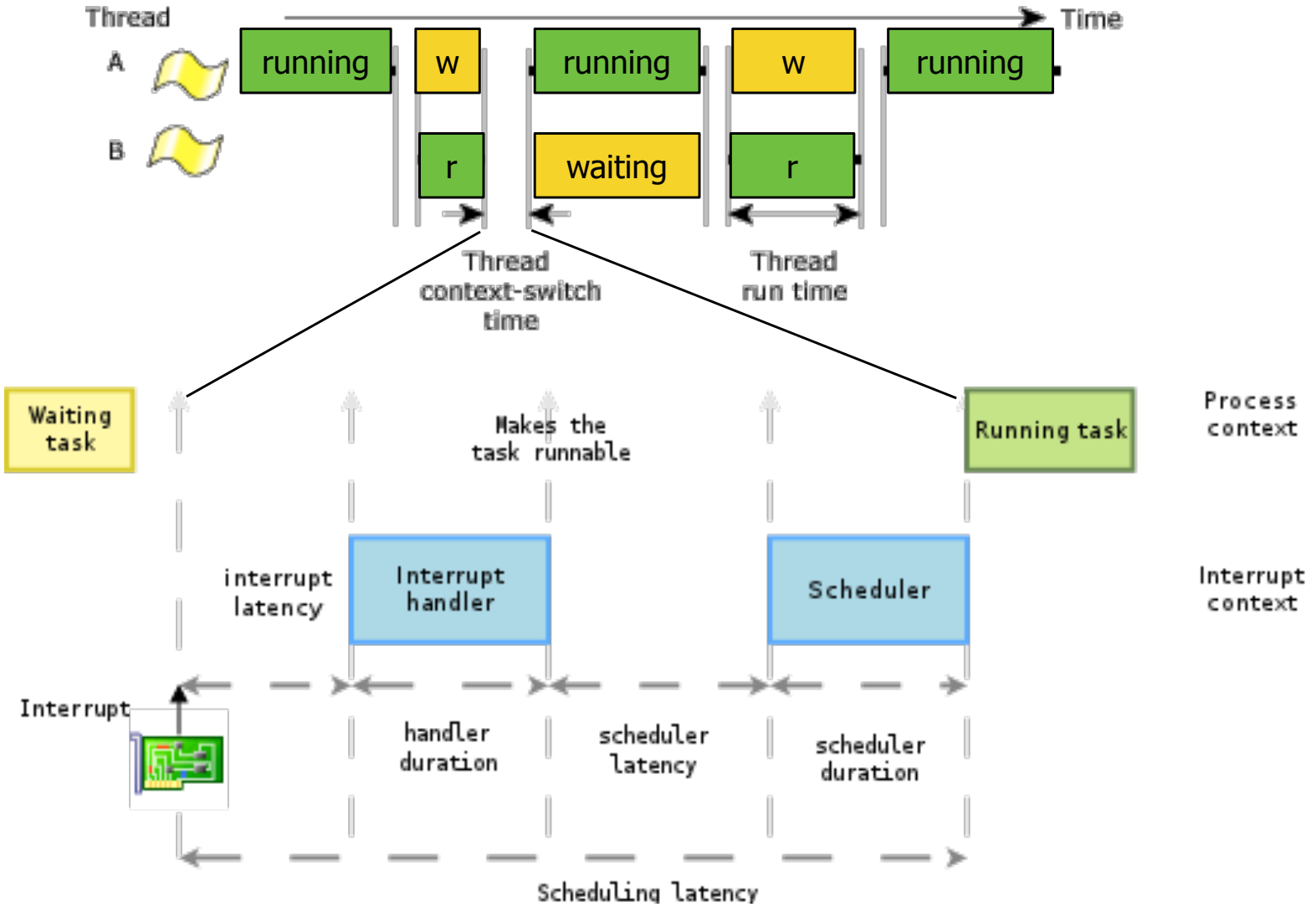
Interrupt Timeline



Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support

Context Switch



Small side-step: Interrupts and parallelization

Why specialized massive parallel system engineers (10.000+ cores) pay special attention to kernel interrupts?

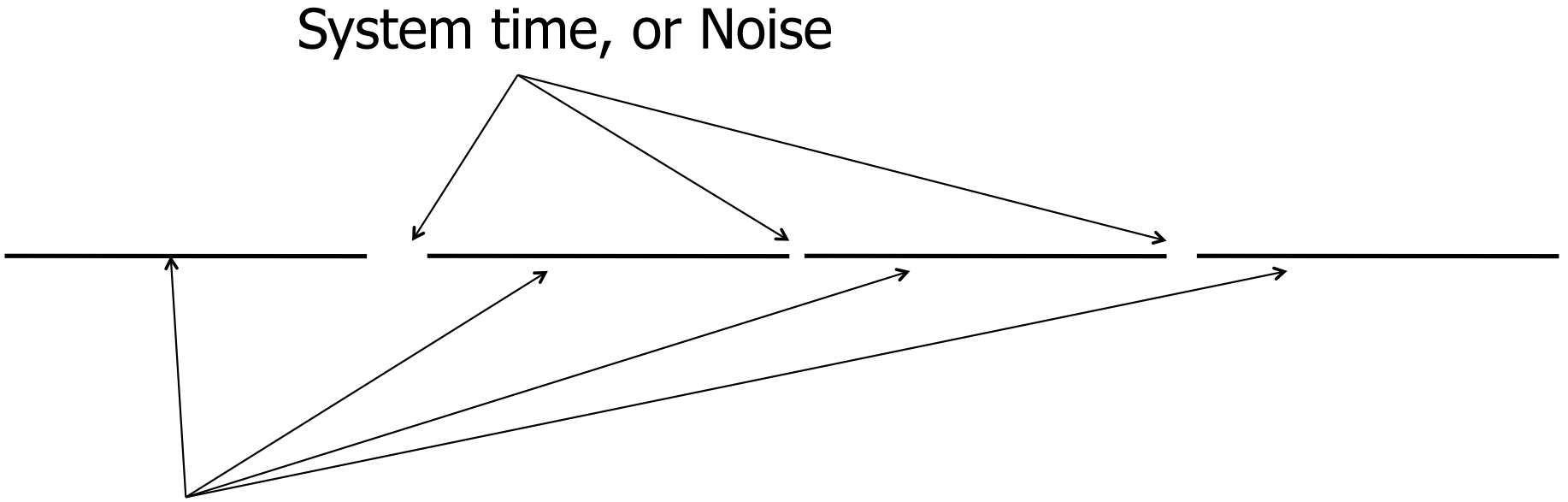
"The Case of the Missing Supercomputer Performance."

http://hpc.pnl.gov/people/fabrizio/papers/sc03_noise.pdf

Process and Context Switching

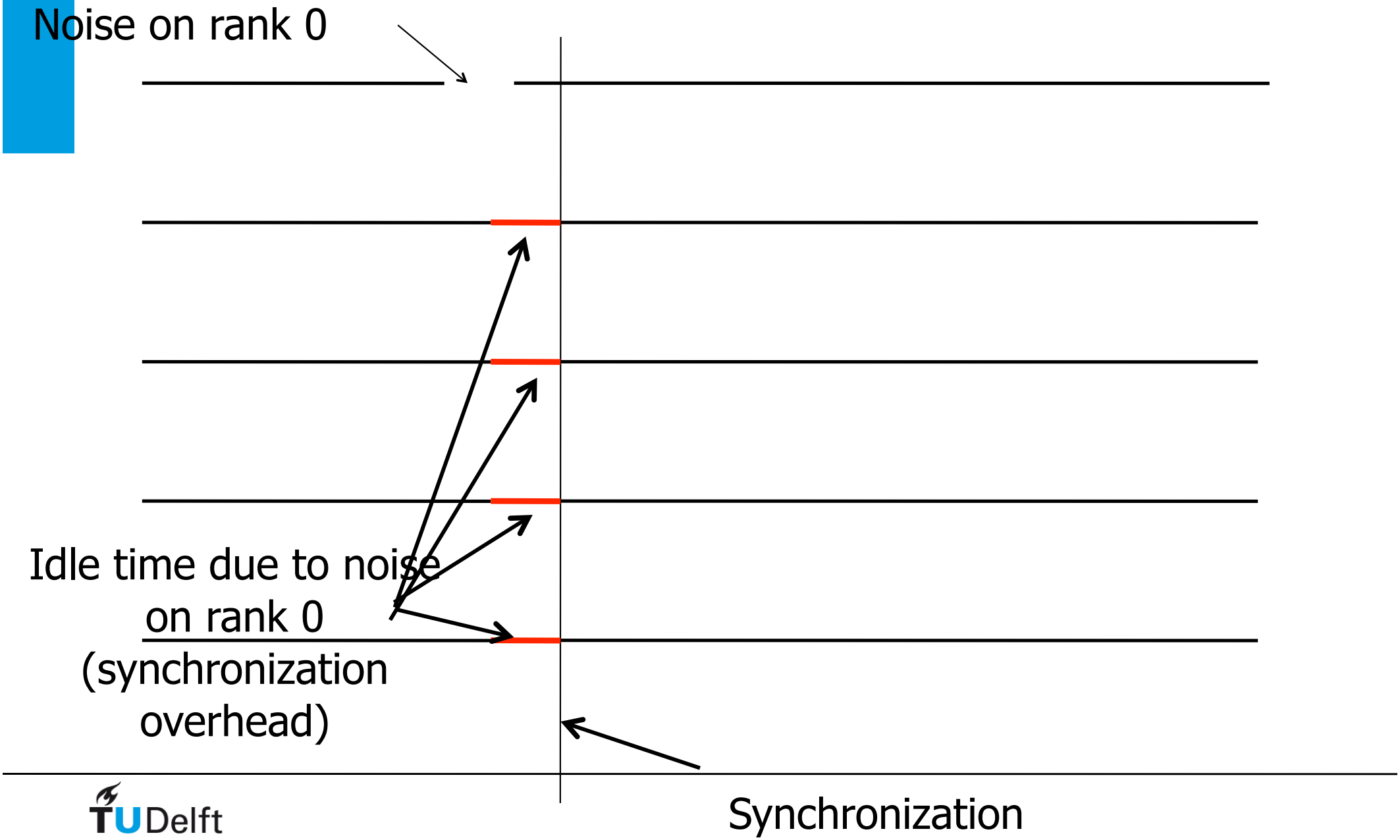
- Processor (CPU) runs one process at a time
  - To run a process, set the program counter (PC).
  - When the processor switches from one process to another it performs a context switch
  - Context switch: swap out the context of currently running process and swap in the context of the process to start running
- The operating system is made up of processes.
  - When an application is running, the OS is not in the CPU.
  - The OS only runs
    - When the timer goes off because an application has been running long enough and it is someone else's turn.
    - When it is explicitly called from the application to do something.
    - When an interrupt has occurred (e.g. page fault)

Noise Illustration for a Serial Application

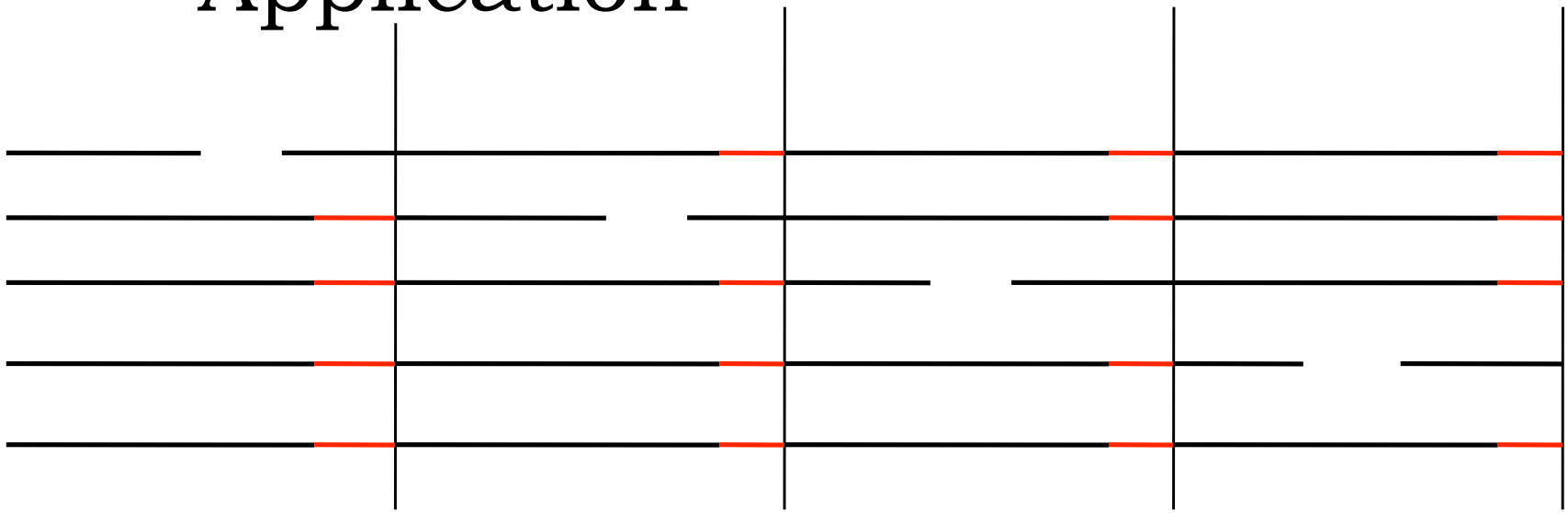


For a serial job, the performance "degradation" caused by noise is simply the additional elapsed time due to time spent on system activity

Effect of Noise in a Parallel Application



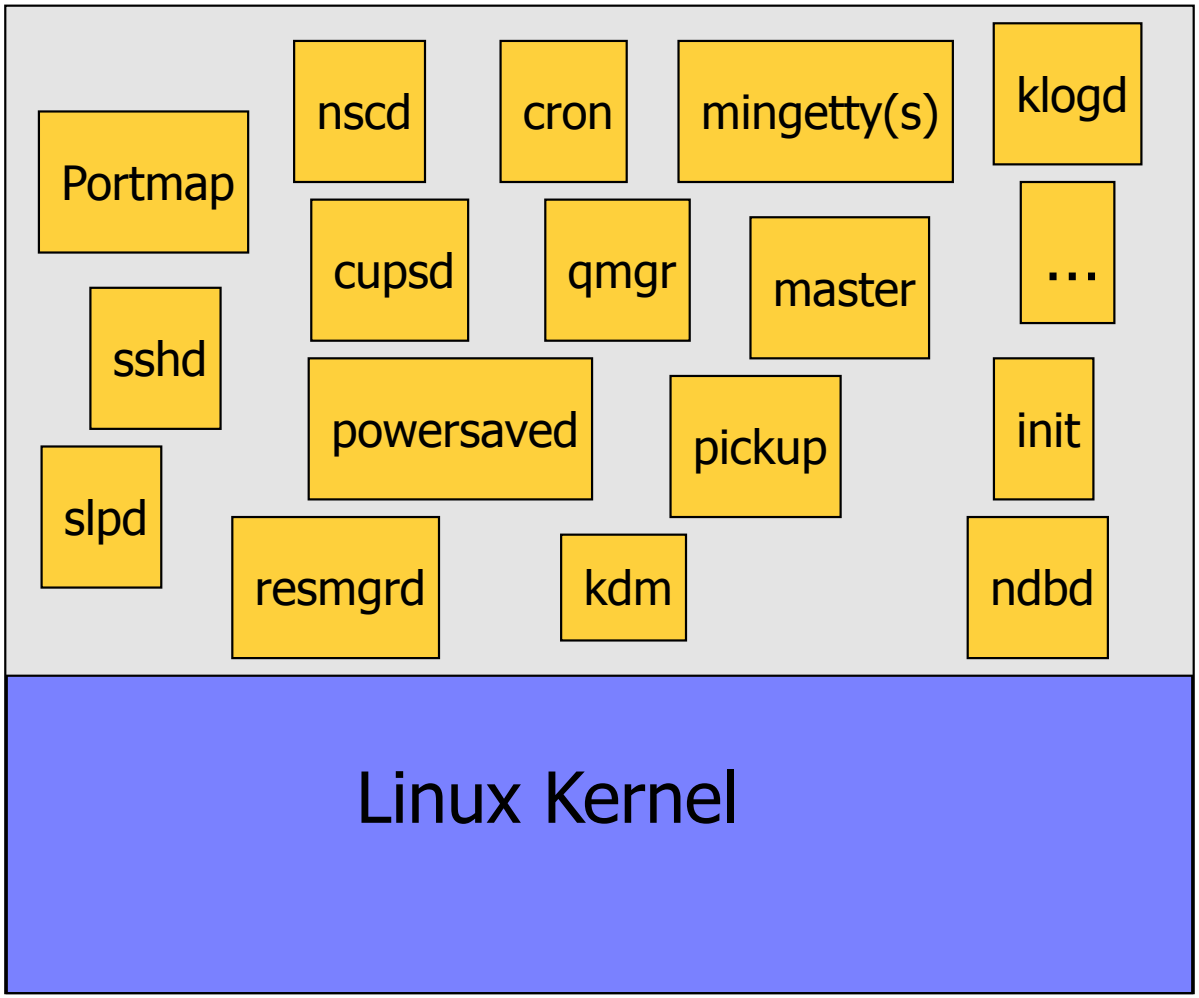
Pathological Noise in a Parallel Application



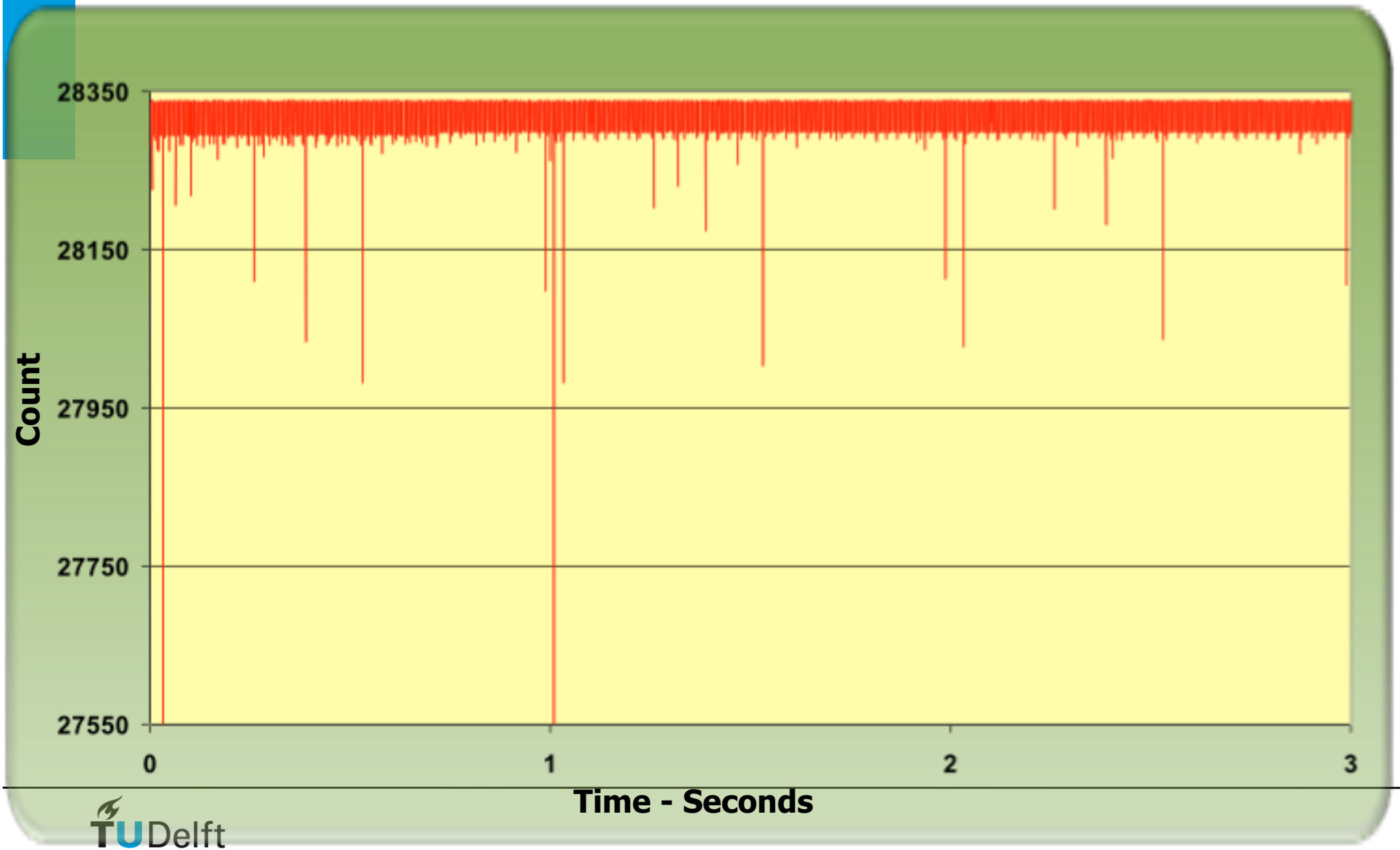
- In each synchronization interval, one rank experiences a noise delay
- Because the ranks synchronize, all ranks experience all of the delays
- In this worst-case situation, the performance degradation due to the noise is multiplied by the number of ranks.
- Noise events that occur infrequently on any one rank (core) occur frequently in the parallel job

Synchronization overhead amplifies the noise

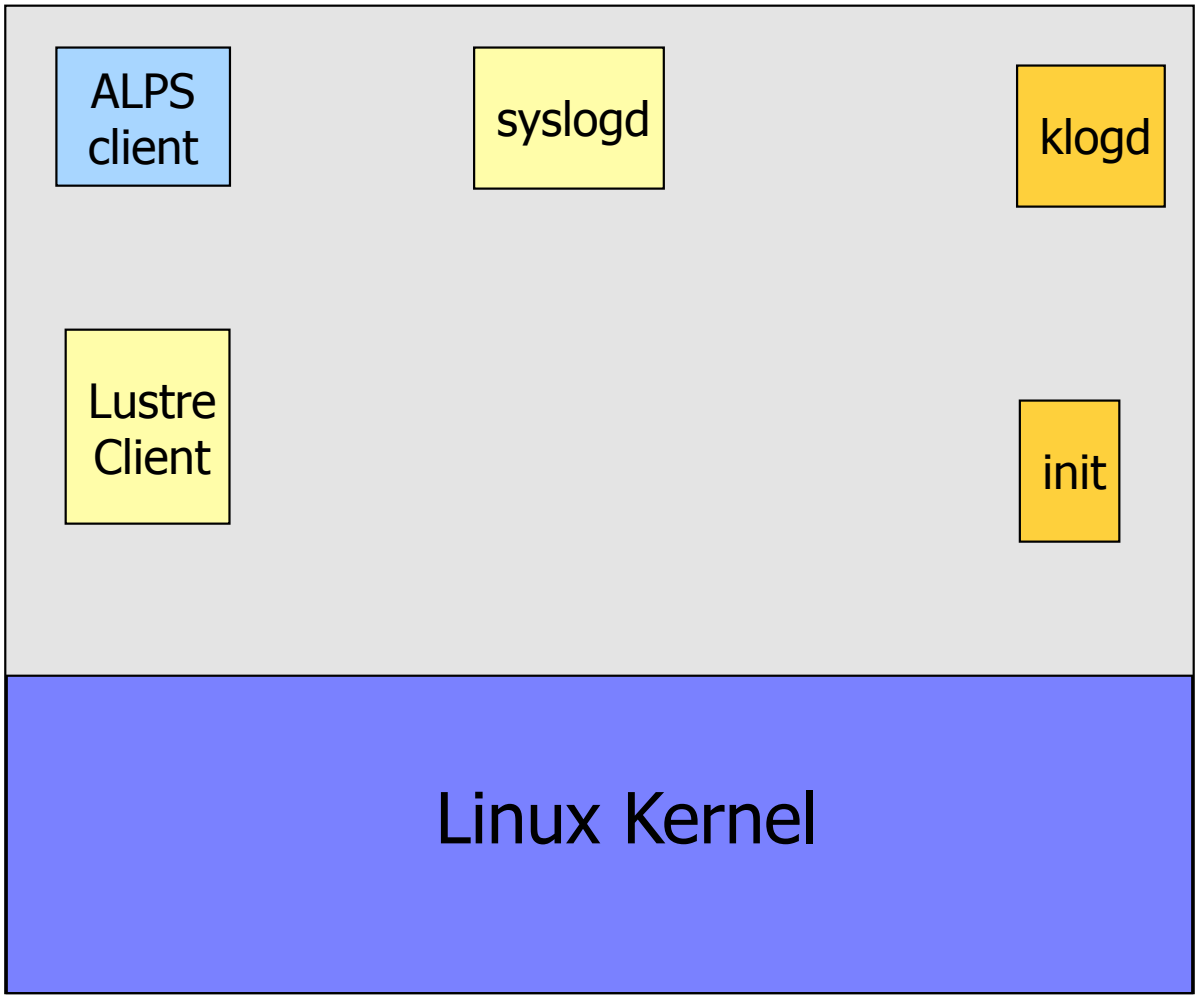
Trimming OS – *Standard Linux Server*



FTQ Plot of Stock SuSE (most daemons removed)



Linux on a Diet – *CNL*



FTQ plot of CNL

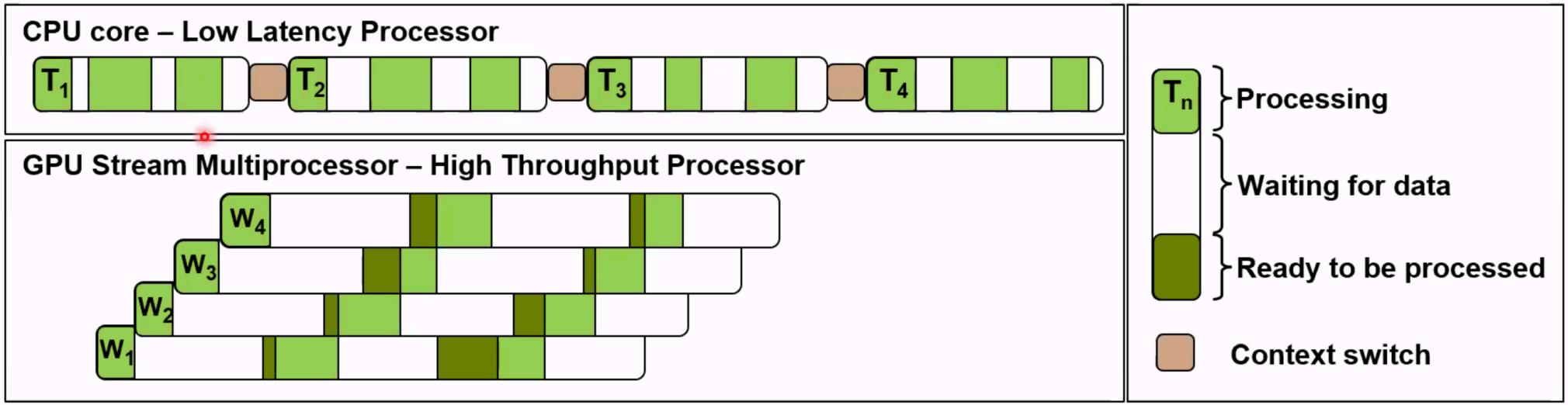
https://rt.wiki.kernel.org/index.php/FTQ

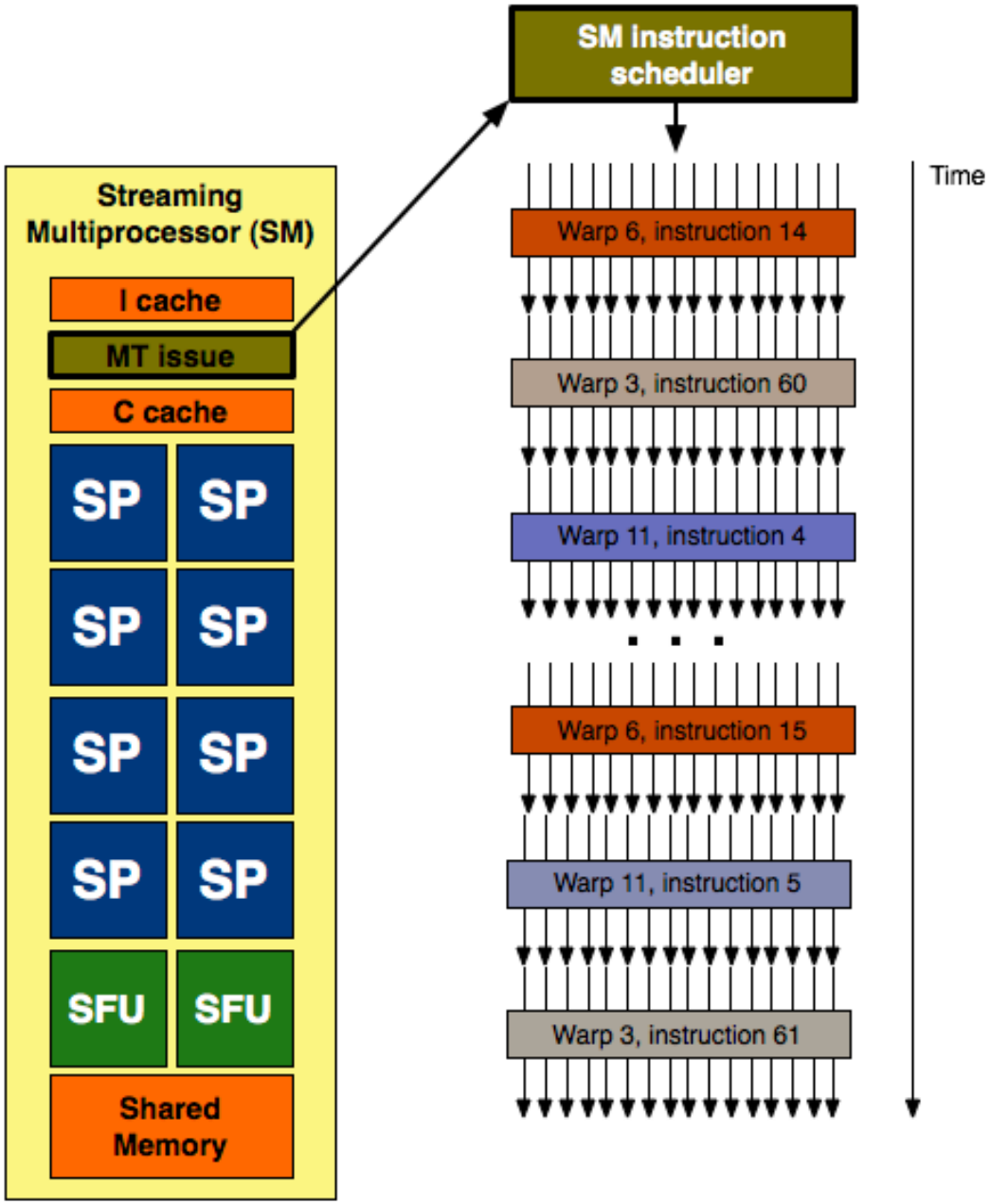
Another small side step: GPU's

- Context switch and GPU's
- GPU's have thousands of cores and have to work on 100 of thousands tasks at the same time to be efficient.
  - How can they switch between tasks?
  - If a wavefront / warp block stalls (e.g. data dependency) CU's can quickly context switch to another wavefront / warp.

Low Latency or High Throughput?

- CPU architecture must minimize latency within each thread
- GPU architecture hides latency with computation from other thread warps




SM instruction scheduler
Streaming Multiprocessor (SM)
I cache
MT issue
C cache
SP
SP
SP
SP
SP
SP
SP
SP
SFU
SFU
Shared Memory
Time
Warp 6, instruction 14
Warp 3, instruction 60
Warp 11, instruction 4
Warp 6, instruction 15
Warp 11, instruction 5
Warp 3, instruction 61

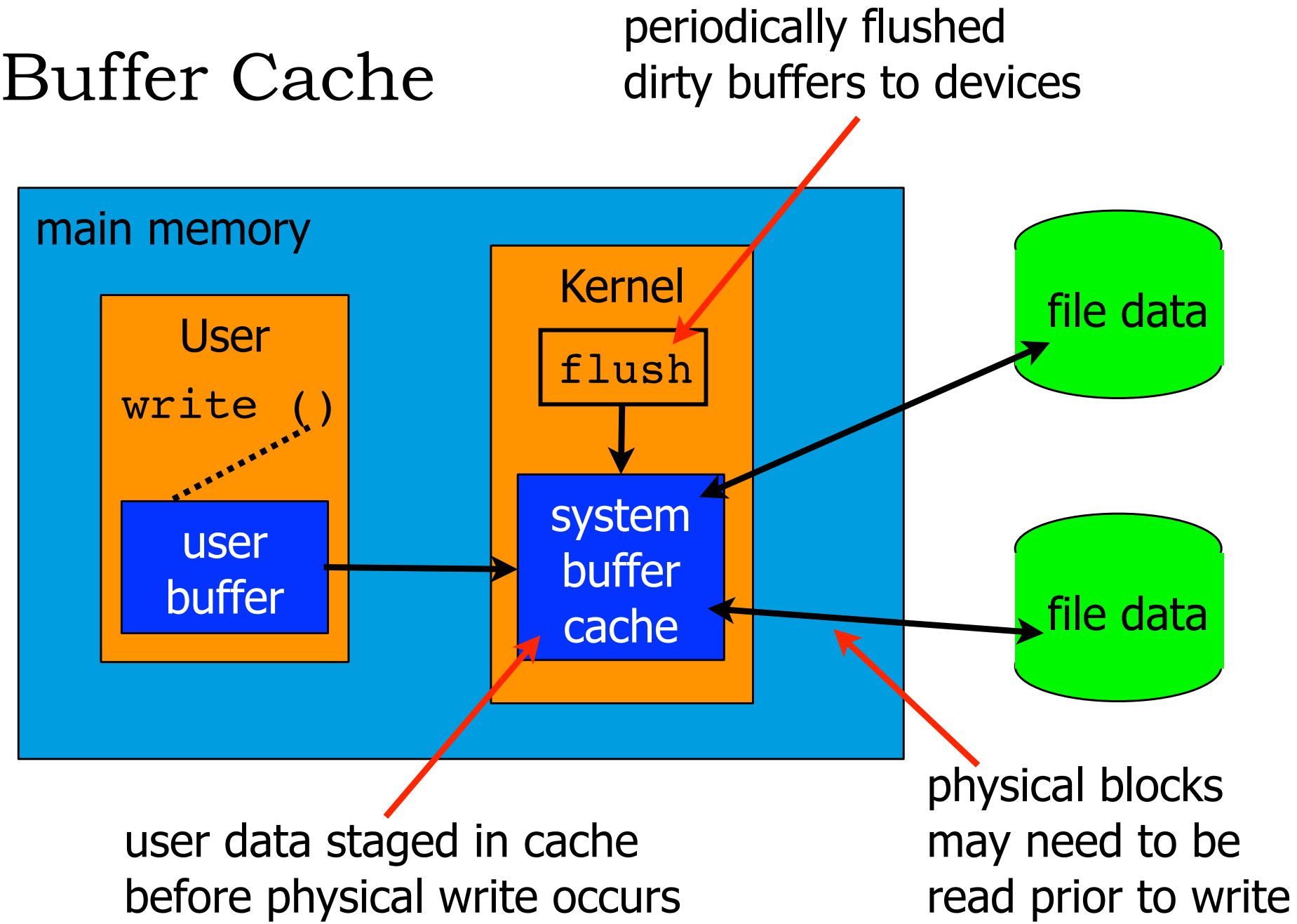
I/O Performance

- I/O can be a major factor in system performance:
  - Demands CPU to execute device driver, kernel I/O code
  - Context switches due to interrupts
  - Data copying
  - Network traffic especially stressful

Kernel I/O Subsystem

- Buffering - store data in memory while transferring between devices
  - To cope with device speed mismatch
  - To cope with device transfer size mismatch
  - To maintain “copy semantics”

Buffer Cache



Buffer Cache

- Advantages of buffered IO requests
  - A system call IO request can access data already stored in system cache buffers much faster than it can read from disk.
  - The system buffer cache will buffer 'ill-formed' user IO requests. which makes IO programming much easier
  - Allows true parallel access to files from multiple threads without the disk ever being accessed
- Disadvantages of buffered IO requests
  - buffer management creates system overhead
  - user data is vulnerable until flushed to the device(s).

I/O optimization

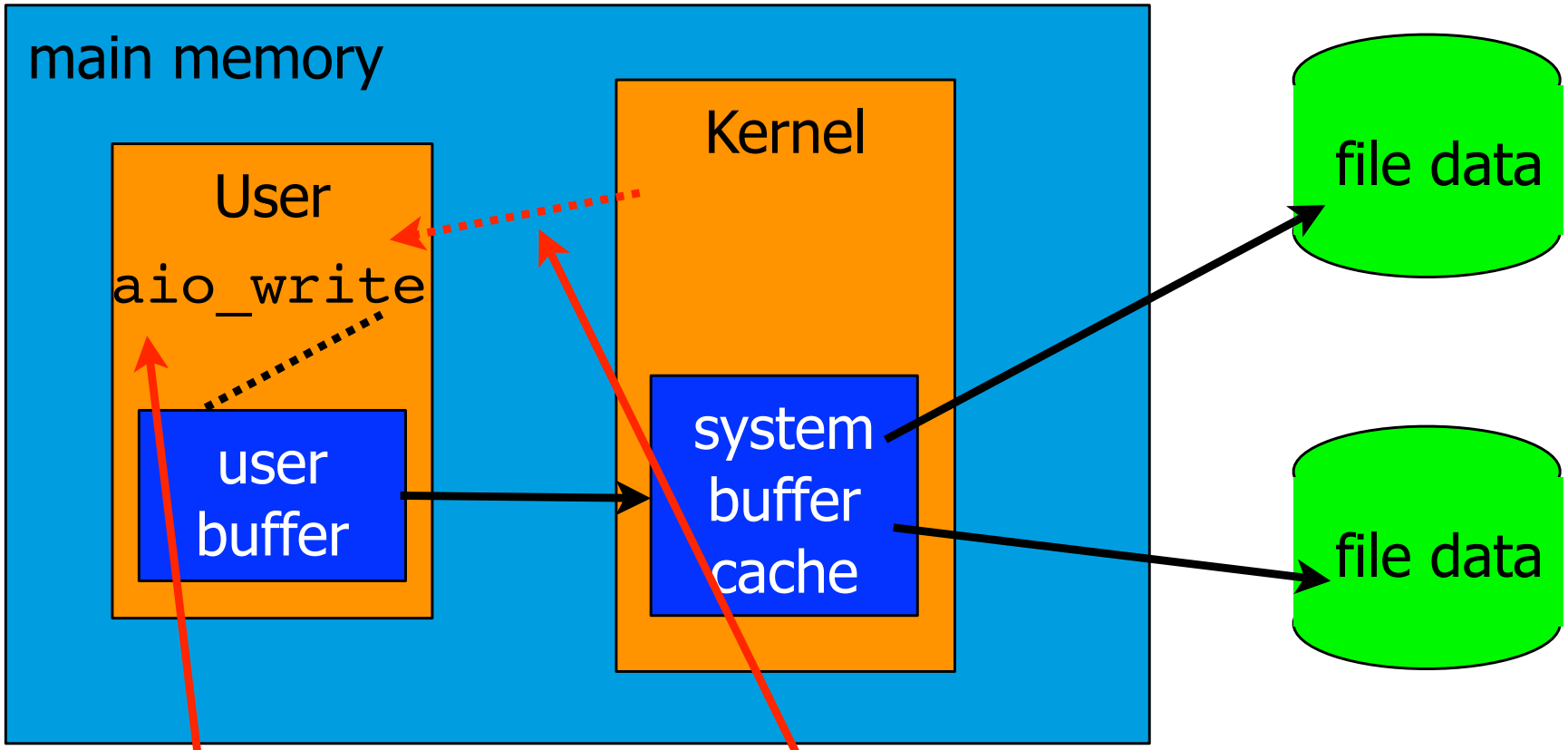
- Application tuning
  - A-synchronous IO
- System tuning
  - striped disks
  - RAID (discussed later)

Blocking and Nonblocking I/O

- **Blocking** - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs

- **Asynchronous** - process runs while I/O executes
  - A bit more difficult to use
  - I/O subsystem signals process when I/O completed
  - IO can be fully overlapped with processing
  - see `man aio`

Asynchronous IO

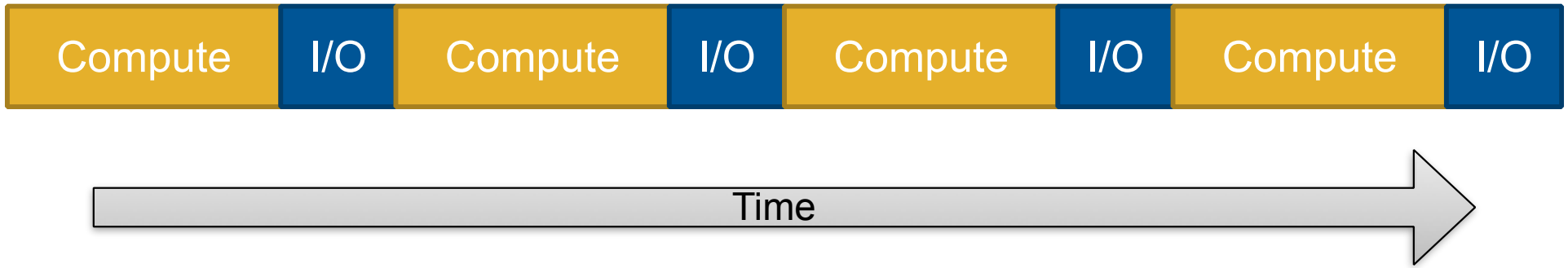


process continues immediately without waiting for IO completion

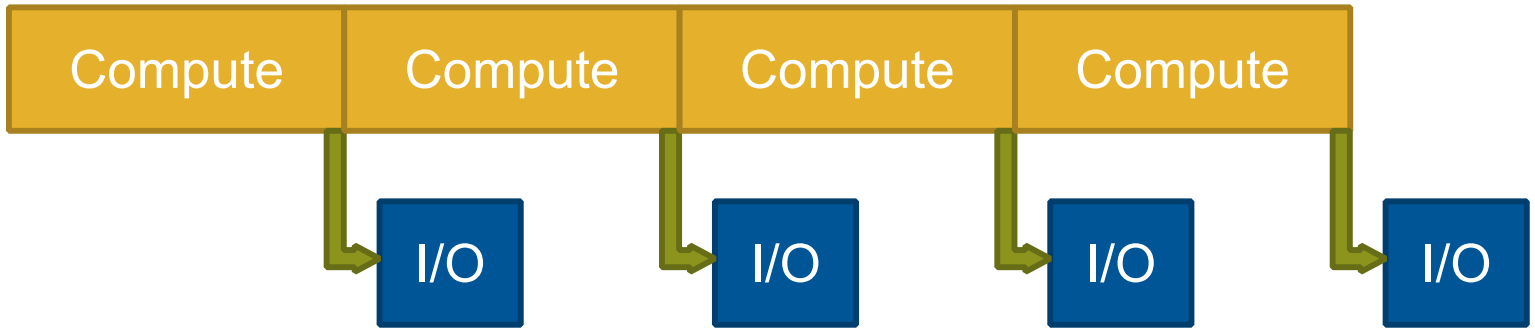
kernel can send signal notifying user of IO completion

Asynchronous IO

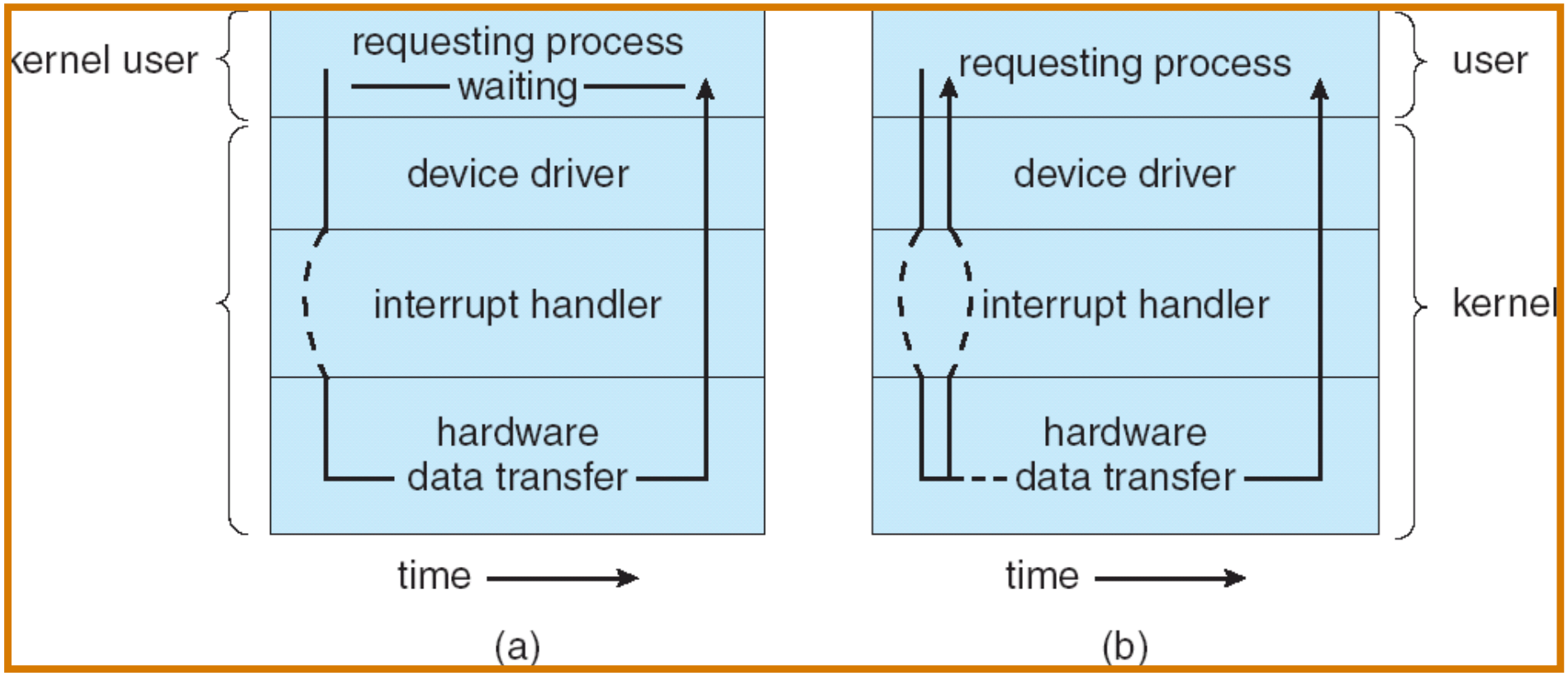
Standard Sequential I/O



Asynchronous I/O



Two I/O Methods



Synchronous

Asynchronous

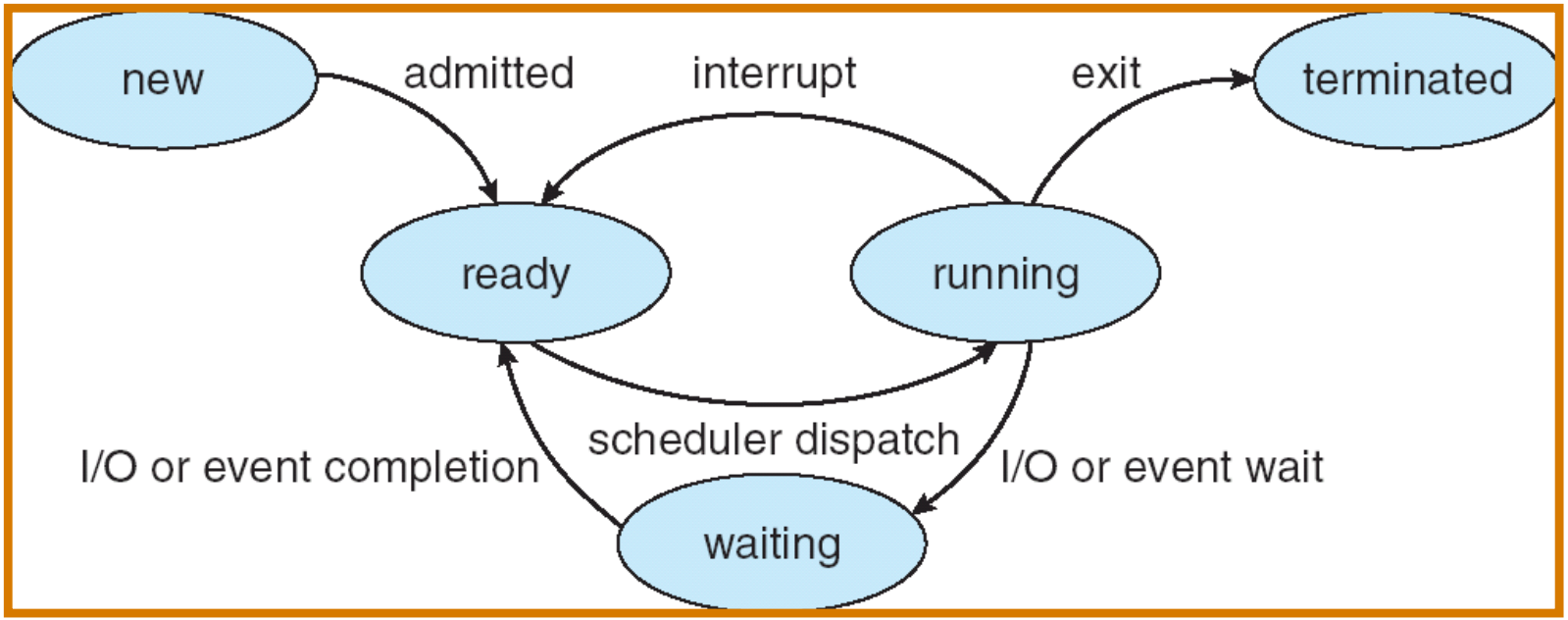
Processes

Process State

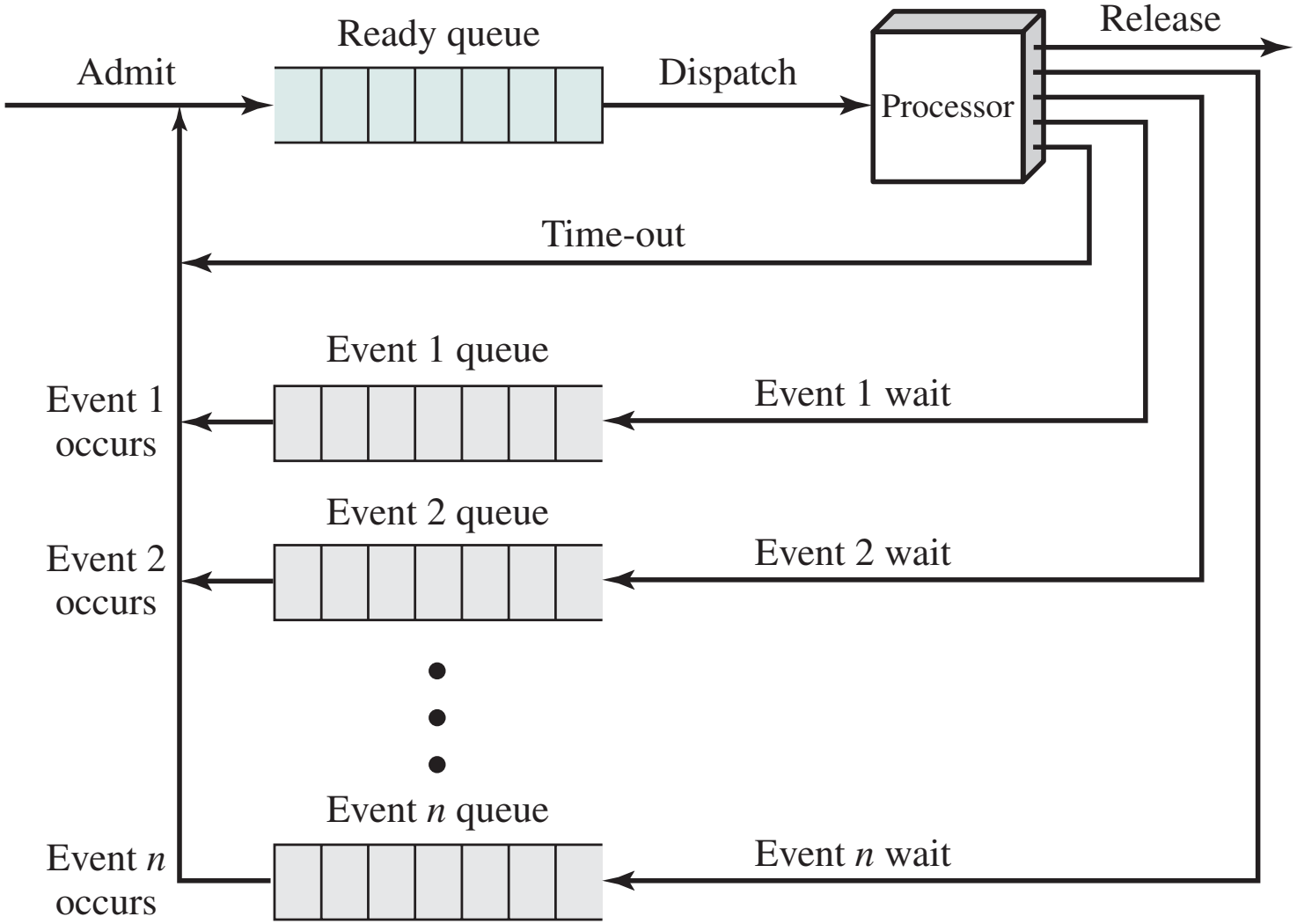
- As a process executes, it changes state
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution

- top and ps command

Diagram of Process State



Process queue's



Operating Systems - Internals and Design Principles 7th ed - W. Stallings (Pearson, 2012)

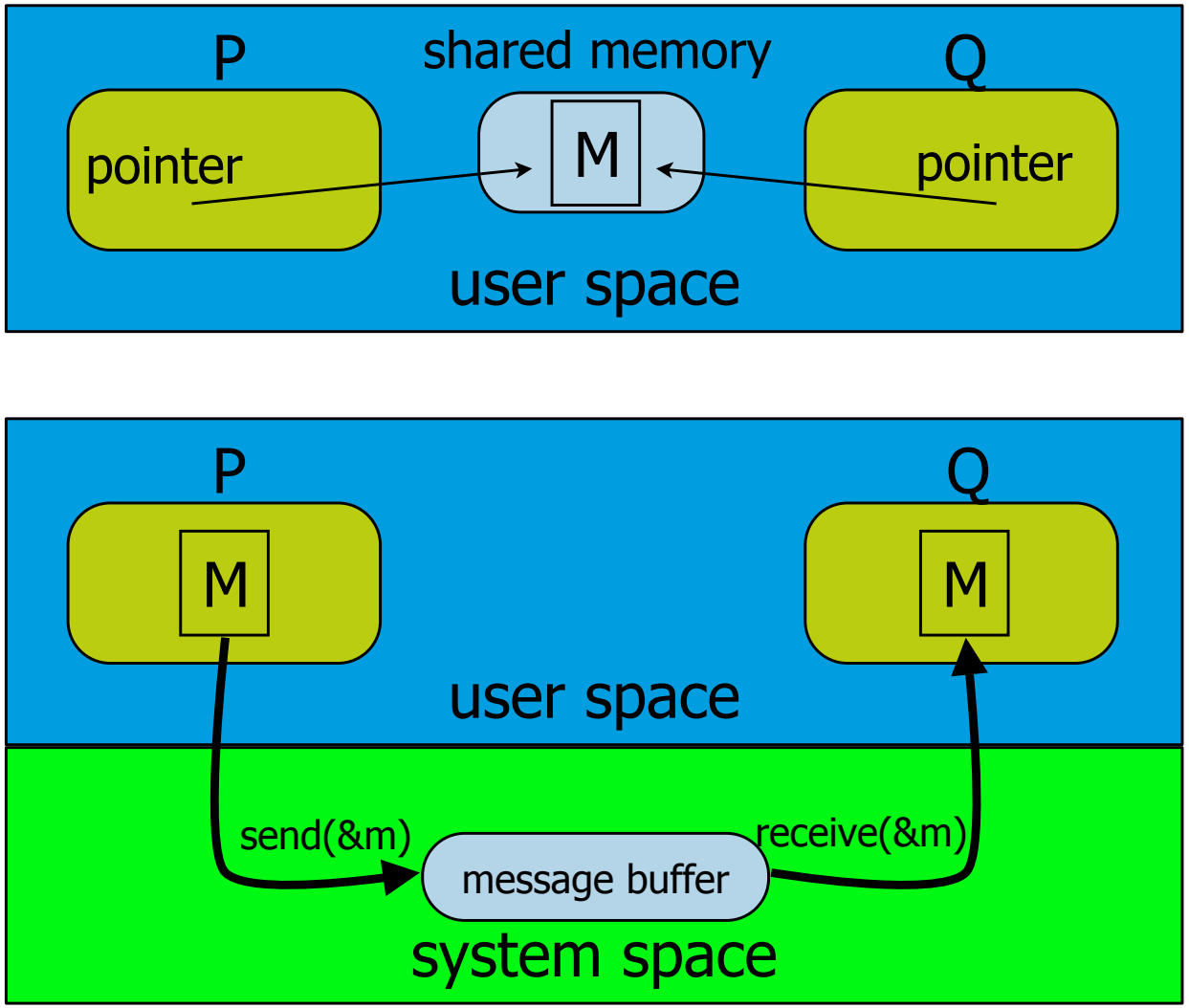
Show on command line

- top with ID of parents (PPID) and ID of processes (PID(PGRP))

Communication

- between processes within the same kernel
- between processes running in different kernels

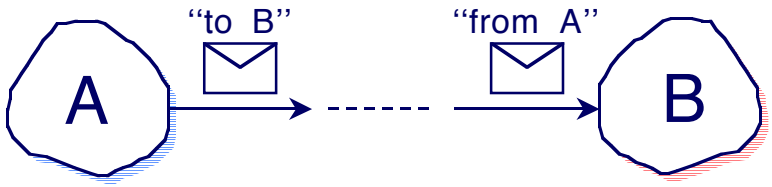
Communications approaches

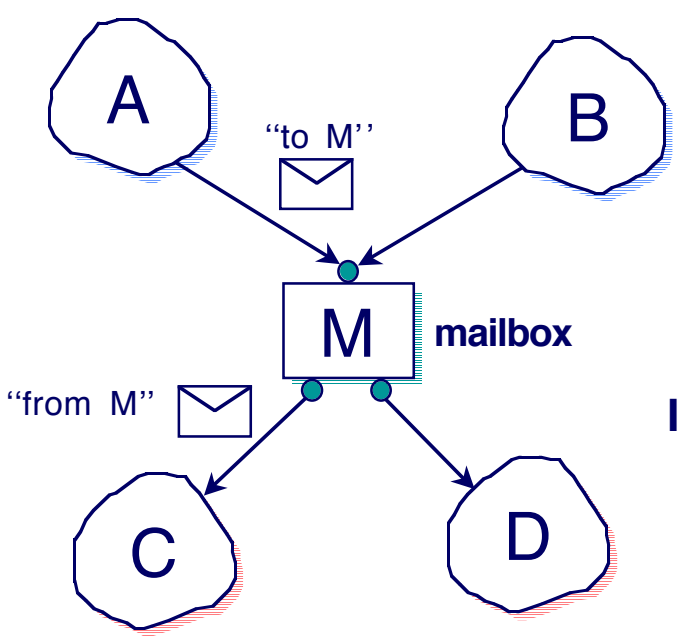


Forms of Communication

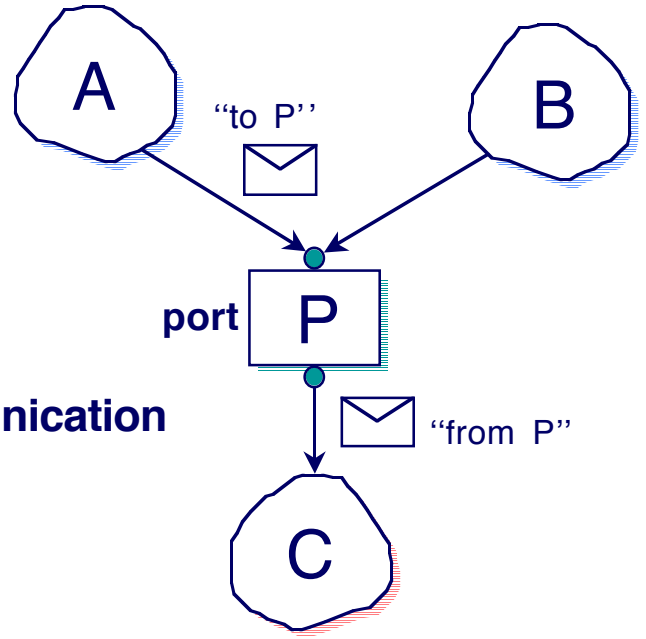
- Direct
  - message goes from sender to receiver
- Indirect
  - message goes through an intermediate named object

Direct communication





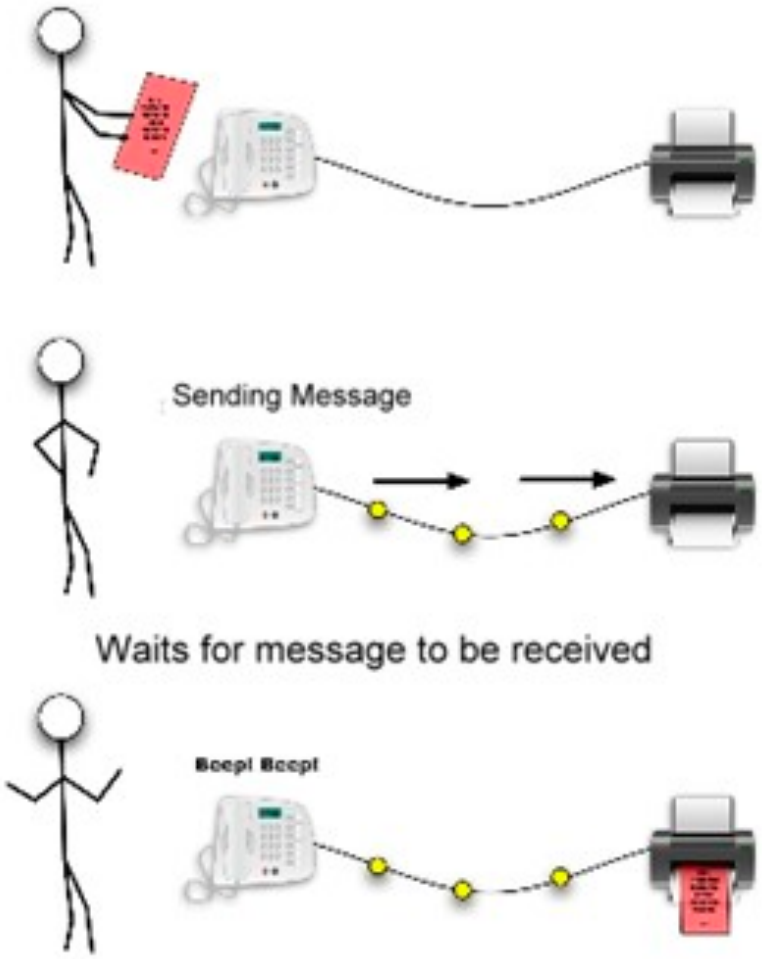
Indirect communication



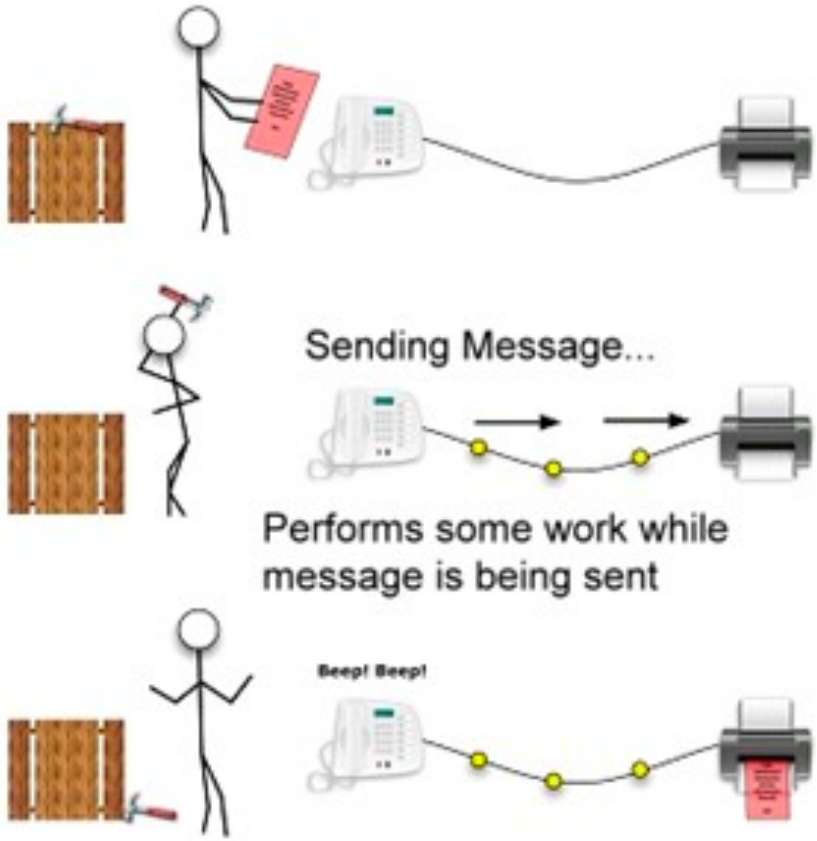
Synchronization with messages

- Message passing may be either blocking or non-blocking
- **Blocking** is considered synchronous
  - Blocking send has the sender block until the message is received
  - Blocking receive has the receiver block until a message is available
- **Non-blocking** is considered asynchronous
  - Non-blocking send has the sender send the message and continue
  - Non-blocking receive has the receiver receive a valid message or null

blocking

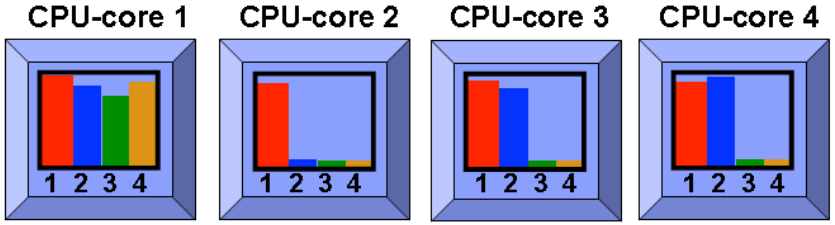


non-blocking



Process Placement

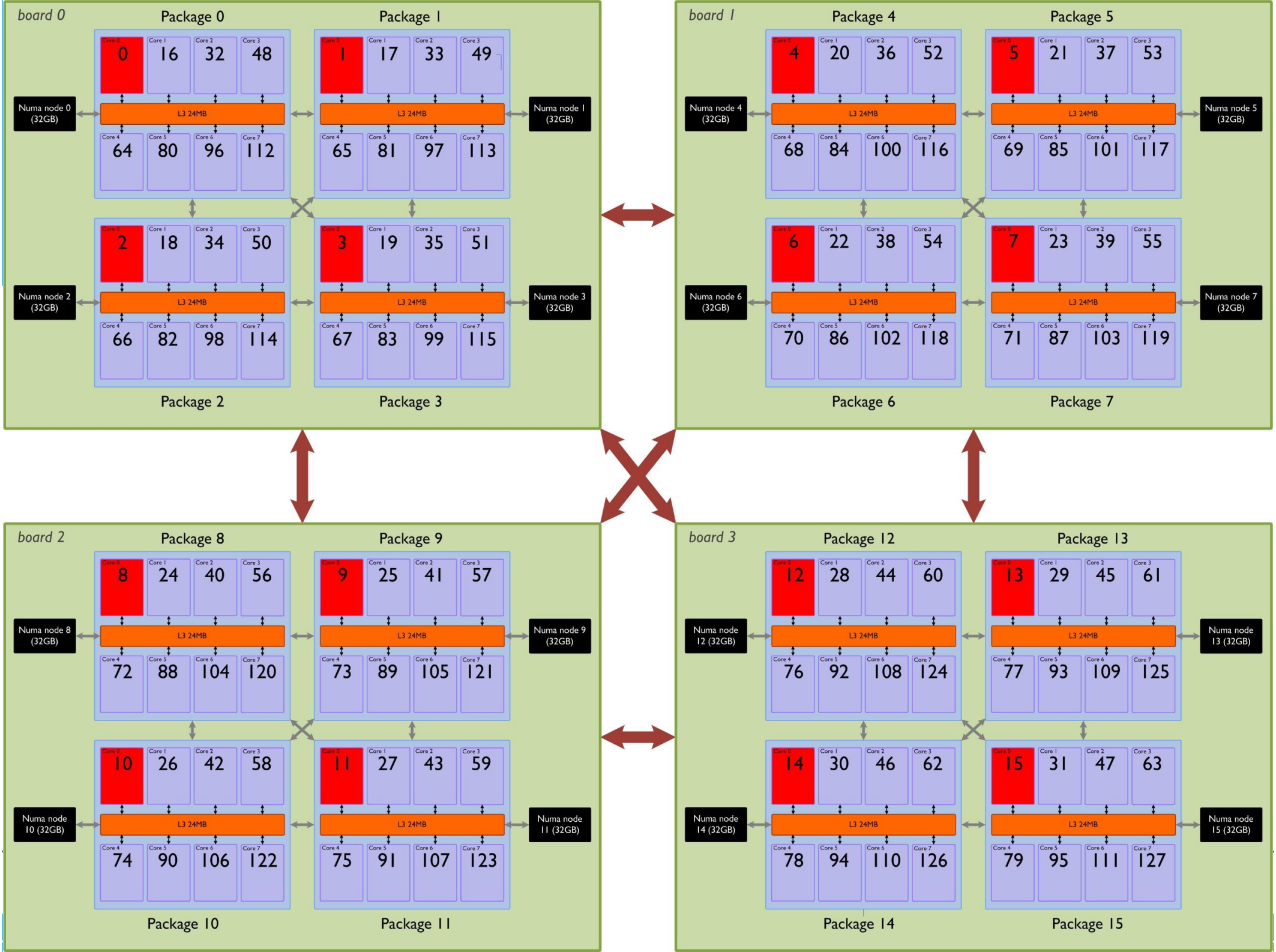
- Many hardware cores are available ,much more process are running on the system:



- How to place the workload?

Algorithm 1. The Placement Distribution Model

1: $\mathbb{N} \leftarrow \{node_1, node_2, \ldots, node_i\}$ *The set of all processor nodes*
2: $\mathbb{M} \leftarrow \{mem_1, mem_2, \ldots, mem_j\}$ *The set of memory nodes*
3: $\mathbb{L} \leftarrow \{link_1, link_2, \ldots, link_k\}$ *The set of all links between nodes*
4: $\mathbb{T} \leftarrow \{data_1, data_2, \ldots, data_l\}$ *The set of data quantities*
5: $\mathbb{E} \leftarrow \mathbb{N}$ x $\mathbb{M}$ *Cartesian product denoting data movement*
6: $\mathbb{G} \leftarrow <\mathbb{E}, \mathbb{L}>$ *Graph $\mathbb{G}$ representing memory and processor layout*
7: $\mathbb{D} \leftarrow \{<x, y> \mid x \in \mathbb{T}, y \in \mathbb{M}\}$ *A data quantity x resides in memory location y*
8: $\mathbb{I} \leftarrow \mathbb{E}$ x $\mathbb{D}$ *Set of inputs for thread, memory placement*
9: $\mathbb{I} \equiv \{<e, f> \mid e = <n, m> \in \mathbb{E}, f = <x, y> \in \mathbb{D}\}$
10: $W(l) \mid l \in \mathbb{L}$ *Weight matrix W*
11: $C(x, y)$ *Cost matrix C*


board 0
Package 0
Package 1
Package 2
Package 3
Numa node 0 (32GB)
Numa node 1 (32GB)
Numa node 2 (32GB)
Numa node 3 (32GB)
L3 24MB
0 16 32 48
1 17 33 49
64 80 96 112
65 81 97 113
2 18 34 50
3 19 35 51
66 82 98 114
67 83 99 115
board 1
Package 4
Package 5
Package 6
Package 7
Numa node 4 (32GB)
Numa node 5 (32GB)
Numa node 6 (32GB)
Numa node 7 (32GB)
4 20 36 52
5 21 37 53
68 84 100 116
69 85 101 117
6 22 38 54
7 23 39 55
70 86 102 118
71 87 103 119
board 2
Package 8
Package 9
Package 10
Package 11
Numa node 8 (32GB)
Numa node 9 (32GB)
Numa node 10 (32GB)
Numa node 11 (32GB)
8 24 40 56
9 25 41 57
72 88 104 120
73 89 105 121
10 26 42 58
11 27 43 59
74 90 106 122
75 91 107 123
board 3
Package 12
Package 13
Package 14
Package 15
Numa node 12 (32GB)
Numa node 13 (32GB)
Numa node 14 (32GB)
Numa node 15 (32GB)
12 28 44 60
13 29 45 61
76 92 108 124
77 93 109 125
14 30 46 62
15 31 47 63
78 94 110 126
79 95 111 127

Scheduling

- OS measures performance (Performance Monitoring Unit, LowLevelCache)
- Based on performance migrates jobs to different part of the machine.
- NonUniformMemoryAccess machines: memory placement is as important: place process as close as possible to the memory they use.

Memory placement policies

- First fit: first available node
- Best fit: node with smallest available memory
- Worst fit: node with largest available memory
- First touch placement
- Round Robin

Placement tools for users

```
numactl --hardware
list of available hardware

taskset —-cpu-list <cpus> <command>
places jobs on requested cpu's

numactl --membind=2 <program>
==> places on memory of node 2

numactl —-interleave=all
==> allocate all memory dimms(4k block)
```

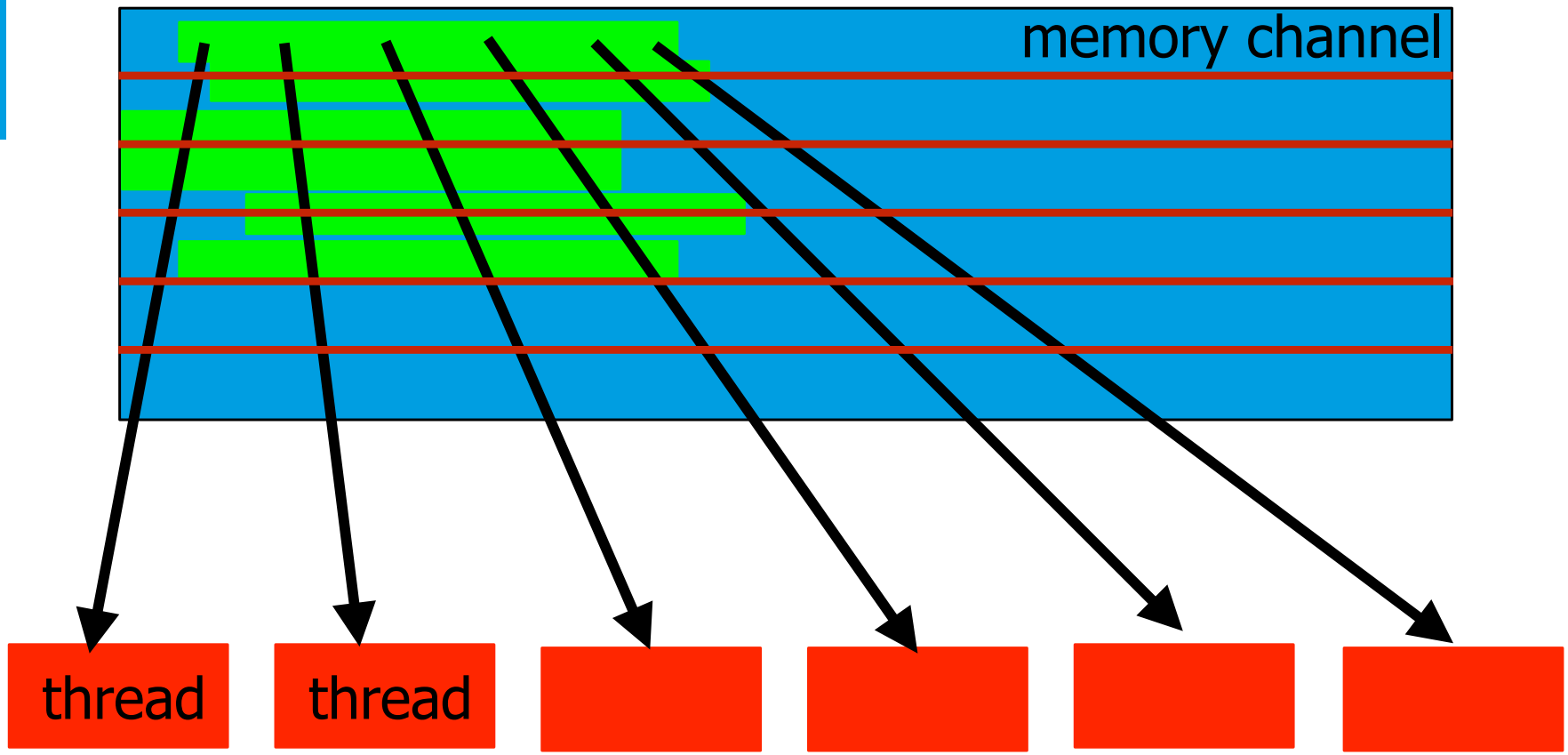
First touch example

```
g1_m1 = (float ***)calloc(ny, sizeof(float **));
for (i1=0; i1<ny; i1++){
  g1_m1[i1] = (float **)calloc(nx, sizeof(float *));
    for (i2=0; i2<nx; i2++) {
      g1_m1[i1][i2] = (float *)calloc(nz, sizeof(float));
}

……

#pragma omp parallel
for (i1=0; i1<ny; i1++){
for (i2=0; i2<nx; i2++)
for (i3=0; i3<nz; i3++)
     a[i1][i2][i3] = p[i1][i2][i3] + C1* p[i1-1][i2][i3+1]
                   + …
```

First touch example



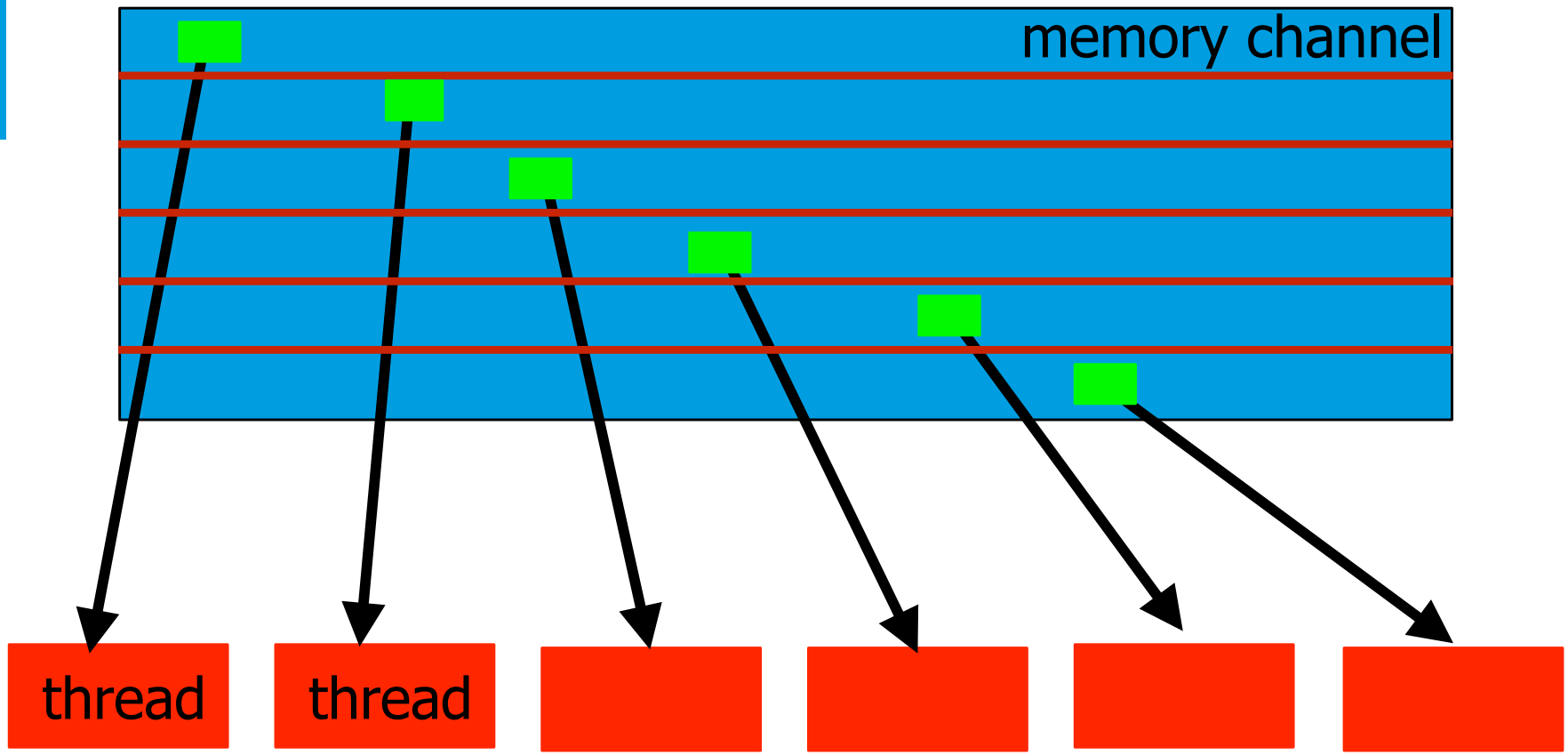
First touch example

```
g1_m1 = (float ***)calloc(ny, sizeof(float **));
#pragma omp parallel
for (i1=0; i1<ny; i1++){
  g1_m1[i1] = (float **)calloc(nx, sizeof(float *));
    for (i2=0; i2<nx; i2++) {
      g1_m1[i1][i2] = (float *)calloc(nz, sizeof(float));
}

……

#pragma omp parallel
for (i1=0; i1<ny; i1++){
for (i2=0; i2<nx; i2++)
for (i3=0; i3<nz; i3++)
     a[i1][i2][i3] = p[i1][i2][i3] + C1* p[i1-1][i2][i3+1]
                     + …
```

First touch example



runtime improved from 576 to 80 seconds using 128 threads

File-System organization

File Attributes (inode metadata)

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on the disk

Linux command line for inode info

- ls -i /proc/cpuinfo (prints inodes number)

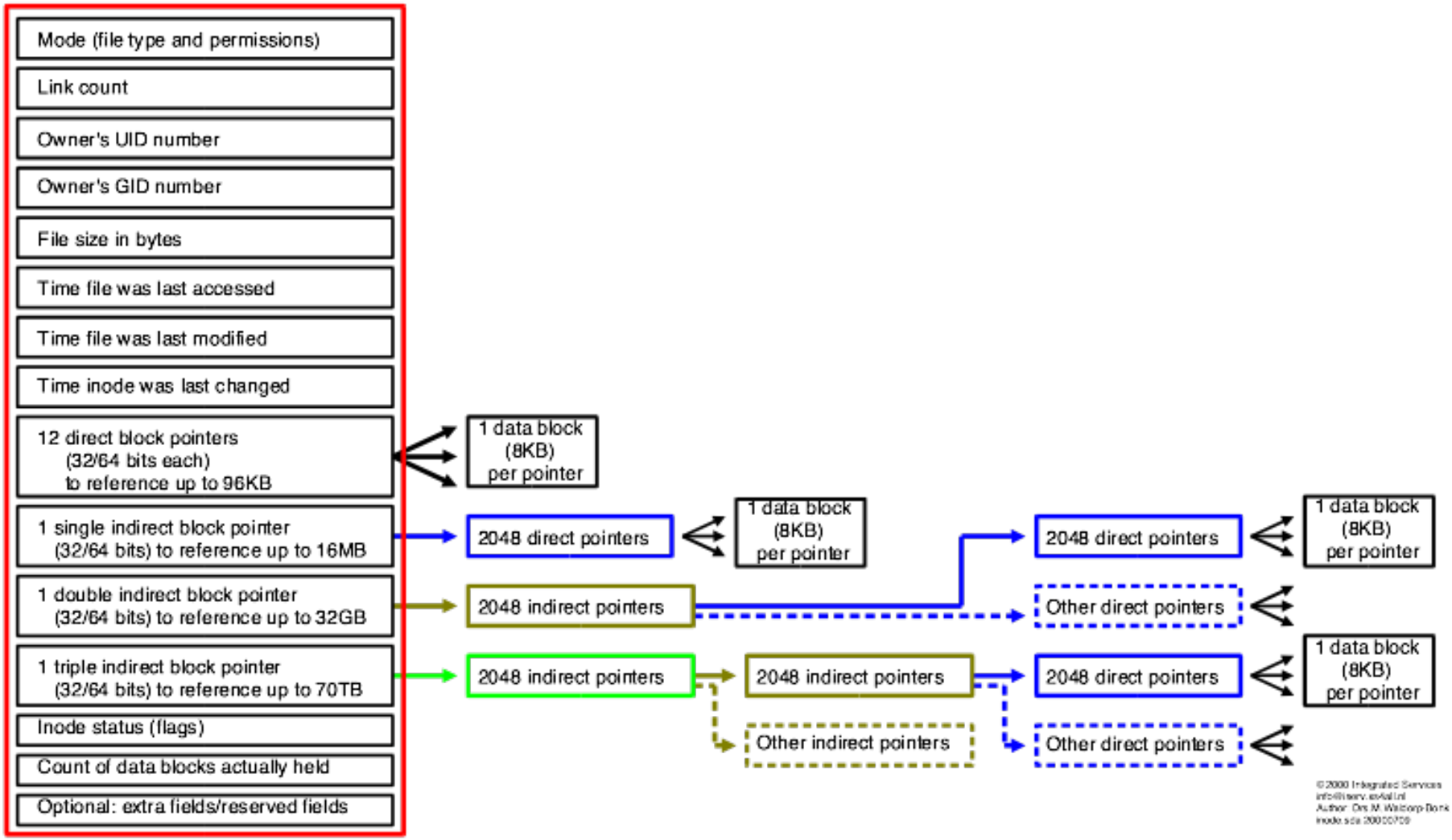
```
4026531851 /proc/cpuinfo
```

- stat /usr/bin/gcc

```
  File: `/usr/bin/gcc' -> `gcc-4.3'
  Size: 7               Blocks: 0          IO Block: 1048576 symbolic link
Device: fh/15d   Inode: 15745695    Links: 1
Access: (0777/lrwxrwxrwx)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2011-04-28 17:14:19.000000000 -0500
Modify: 2010-08-30 11:44:57.000000000 -0500
Change: 2010-08-30 11:44:57.000000000 -0500
```

- df -ih

UNIX inode structure (POSIX)



File Operations

- File is an **abstract data type** (defined by operations)
- **Create**
- **Write**
- **Read**
- **Reposition within file**
- **Delete**
- **Truncate**
- *Open(F_i)* – search the directory structure on disk for entry F_i, and move the content of entry to memory
- *Close (F_i)* – move the content of entry F_i in memory to directory structure on disk

File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (**NFS**) is a common distributed file-sharing method

File Sharing – Multiple Users

- **User IDs** identify users, allowing permissions and protections to be per-user
- **Group IDs** allow users to be in groups, permitting group access rights

Protection

- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

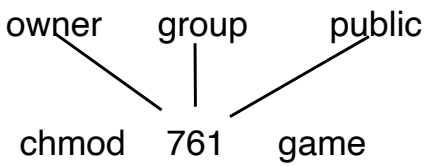
Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users

			RWX
a) **owner access**	7	⇒	1 1 1
			RWX
b) **group access**	6	⇒	1 1 0
			RWX
c) **public access**	1	⇒	0 0 1

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

Attach a group to a file



chgrp G game

Example of permissions

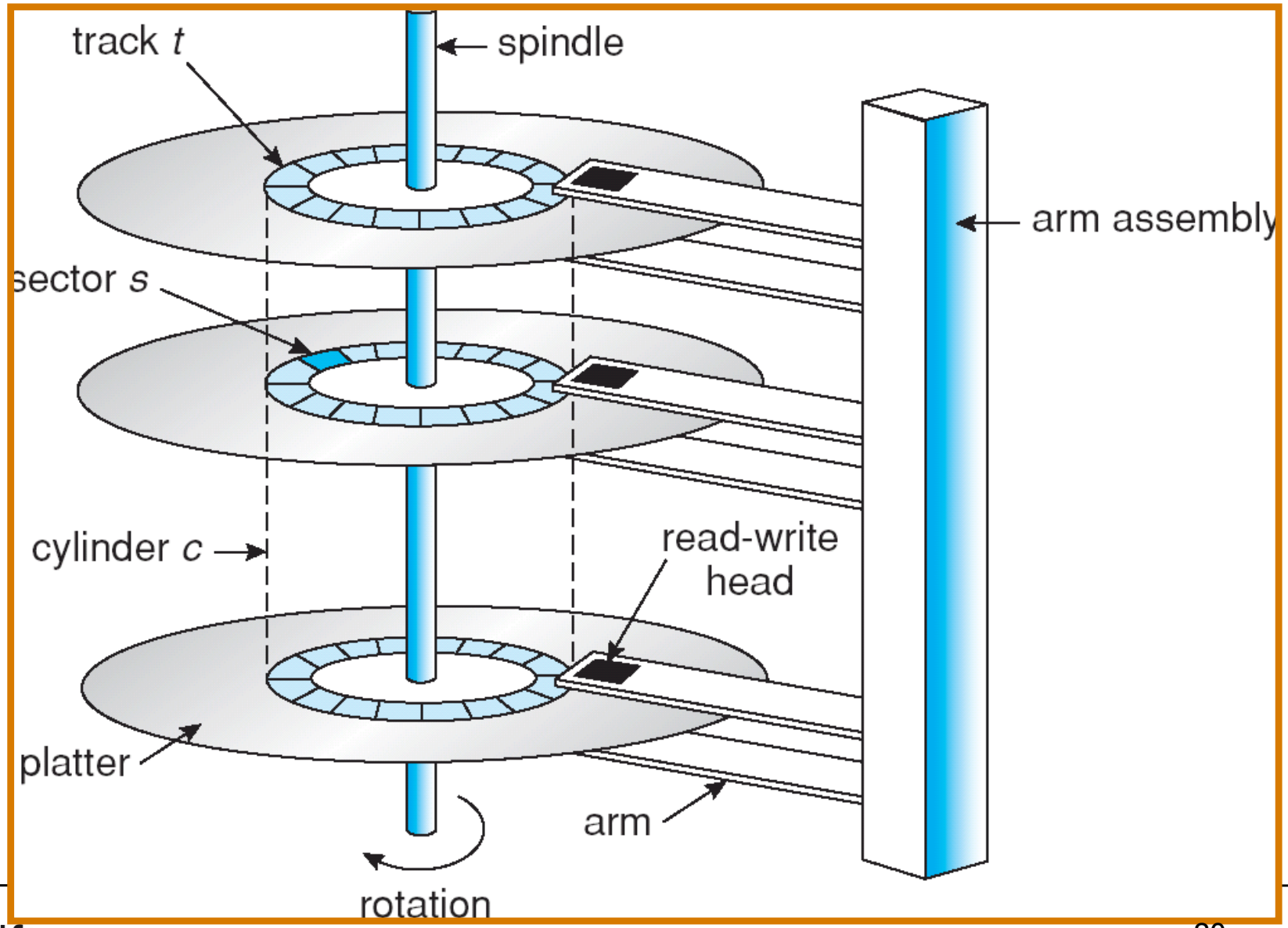
- Go to Unix shell.

Mass-Storage Systems

Mass-Storage Systems

- striped disks
- RAID Structure
- Performance Issues

Moving-head Disk Mechanism



Striped Disks

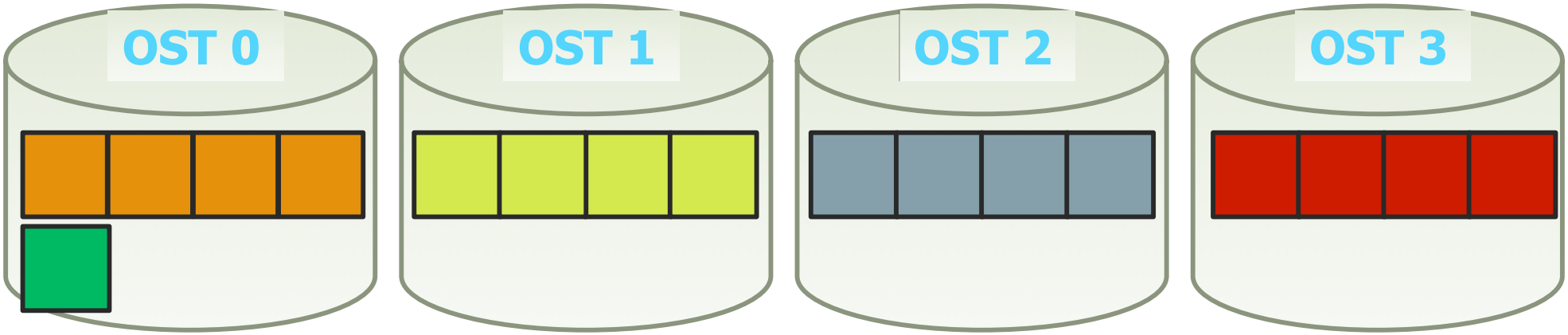
Stripe data across disks to create logical volumes with greater throughput

- Each stripe unit in a stripe can be read and written simultaneously
- Choose an appropriate stripe unit and IO size
- Application must do a large data transfer which access all disks in a stripe group

Striping:
Logical and Physical View of a File

- Logically, a file is a linear sequence of bytes :

- Physically, a file consists of data distributed across OSTs.



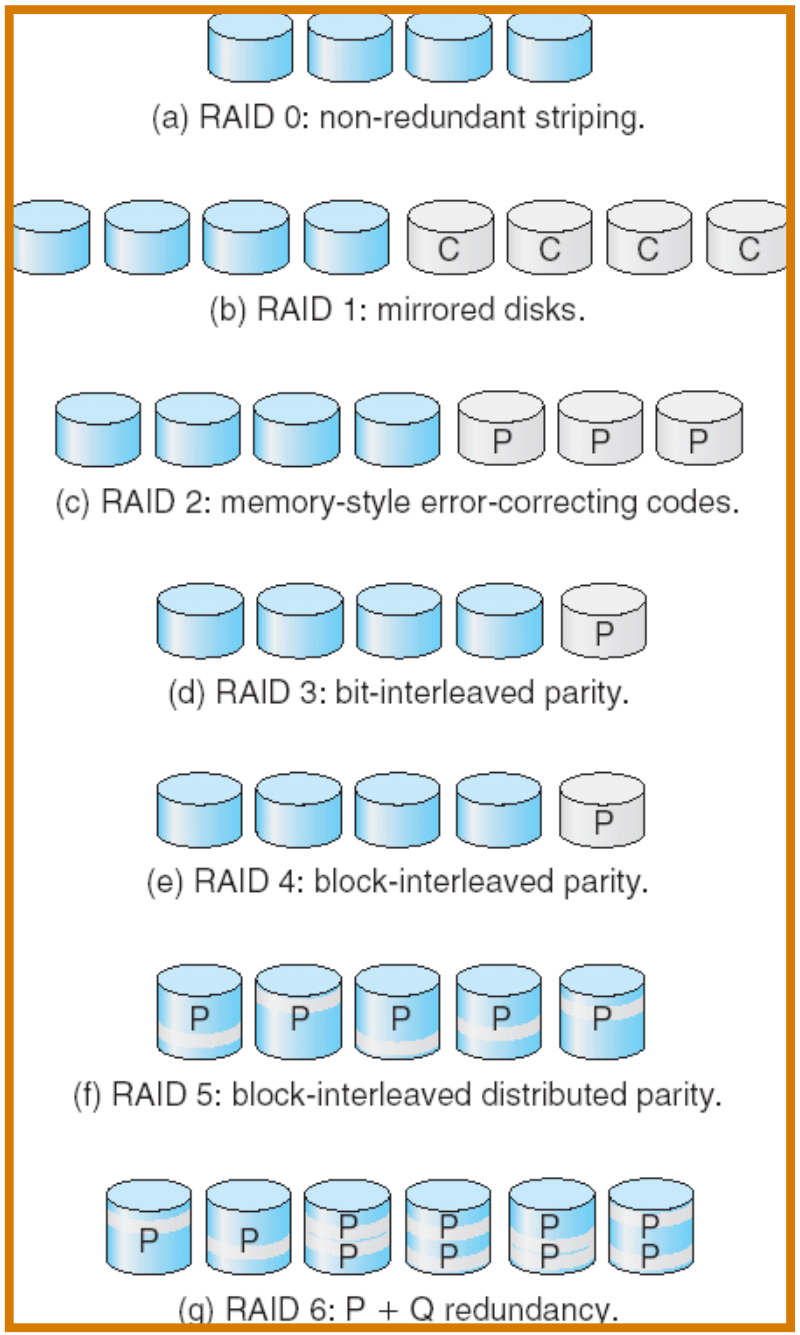
RAID Structure

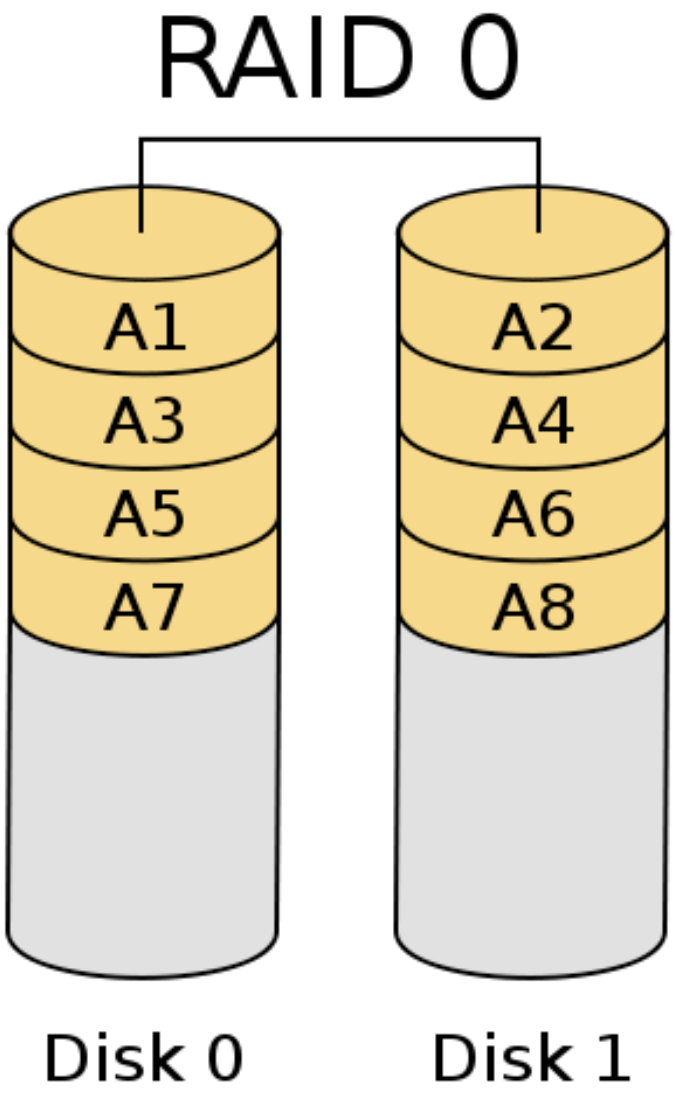
- **R**edundant **A**rray of **I**ndependent **D**isks
- **RAID** – multiple disk drives provides **reliability** via **redundancy**.
- **RAID** is arranged into six different levels.
- http://en.wikipedia.org/wiki/RAID

RAID

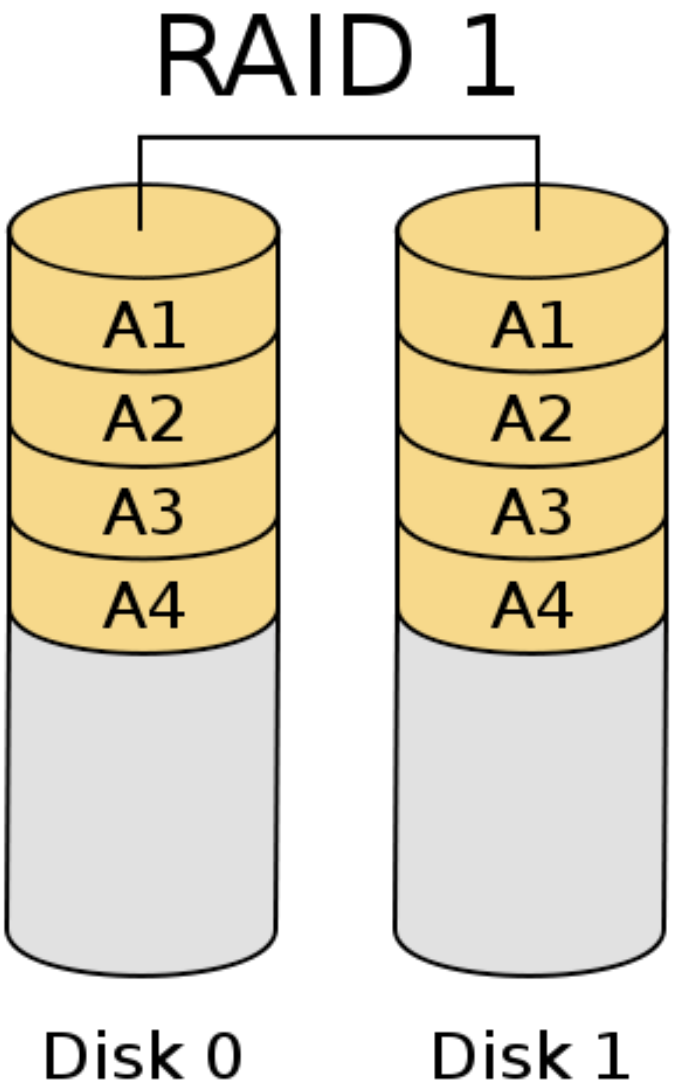
- Several improvements in disk-use techniques involve the use of multiple disks working co-operatively.
- Disk striping uses a group of disks as one storage unit.
- RAID schemes improve performance and improve the reliability of the storage system by storing redundant data.
  - *Mirroring* or *shadowing* keeps duplicate of each disk.
  - *Block interleaved parity* uses much less redundancy.

RAID Levels

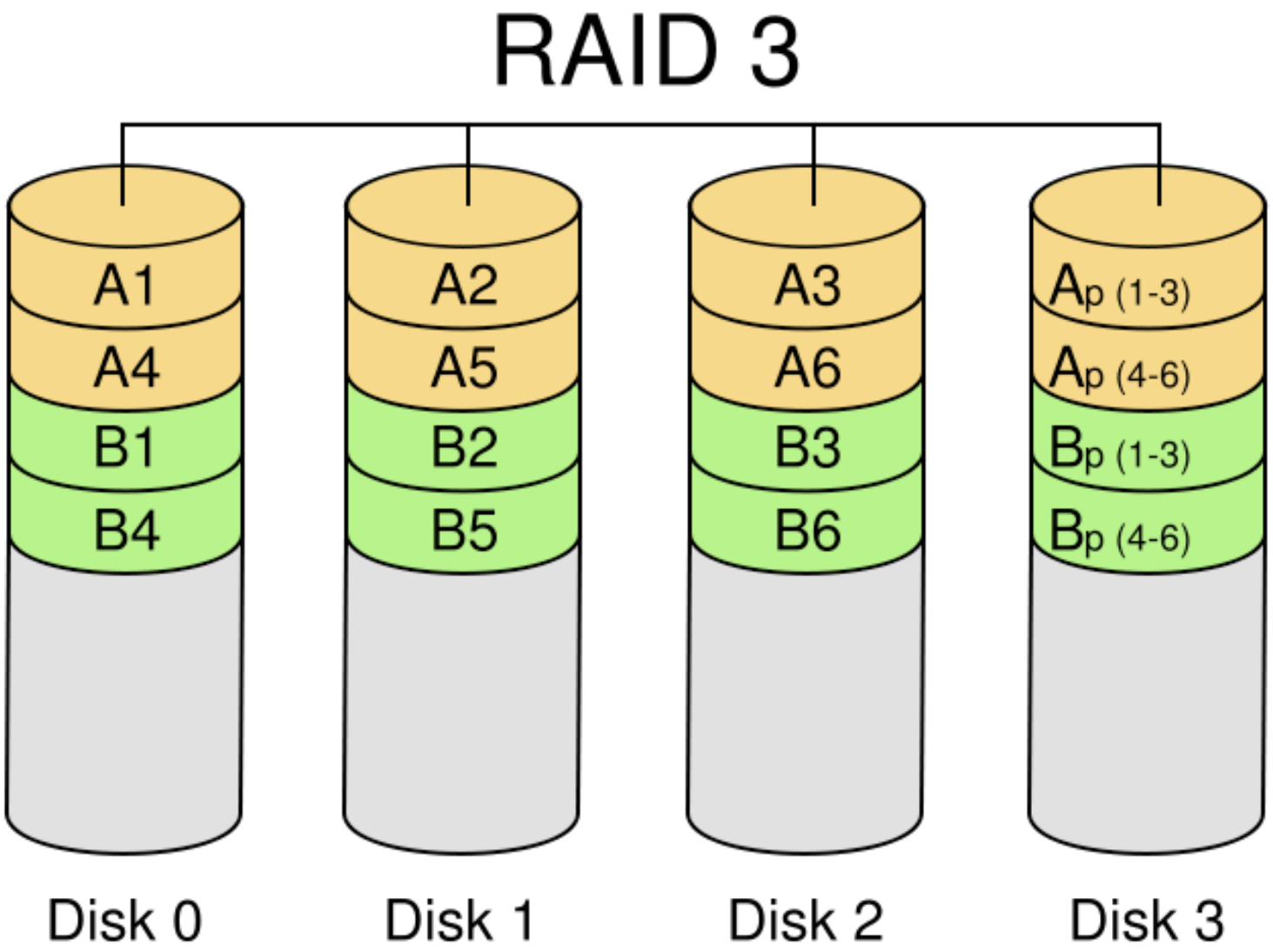

(a) RAID 0: non-redundant striping.
C
(b) RAID 1: mirrored disks.
P
(c) RAID 2: memory-style error-correcting codes.
P
(d) RAID 3: bit-interleaved parity.
P
(e) RAID 4: block-interleaved parity.
P
(f) RAID 5: block-interleaved distributed parity.
P
(g) RAID 6: P + Q redundancy.



"**Striping**". Provides improved performance and additional storage but no redundancy or fault tolerance. Any disk failure destroys the array, which has greater consequences with more disks in the array. The data is broken into fragments.

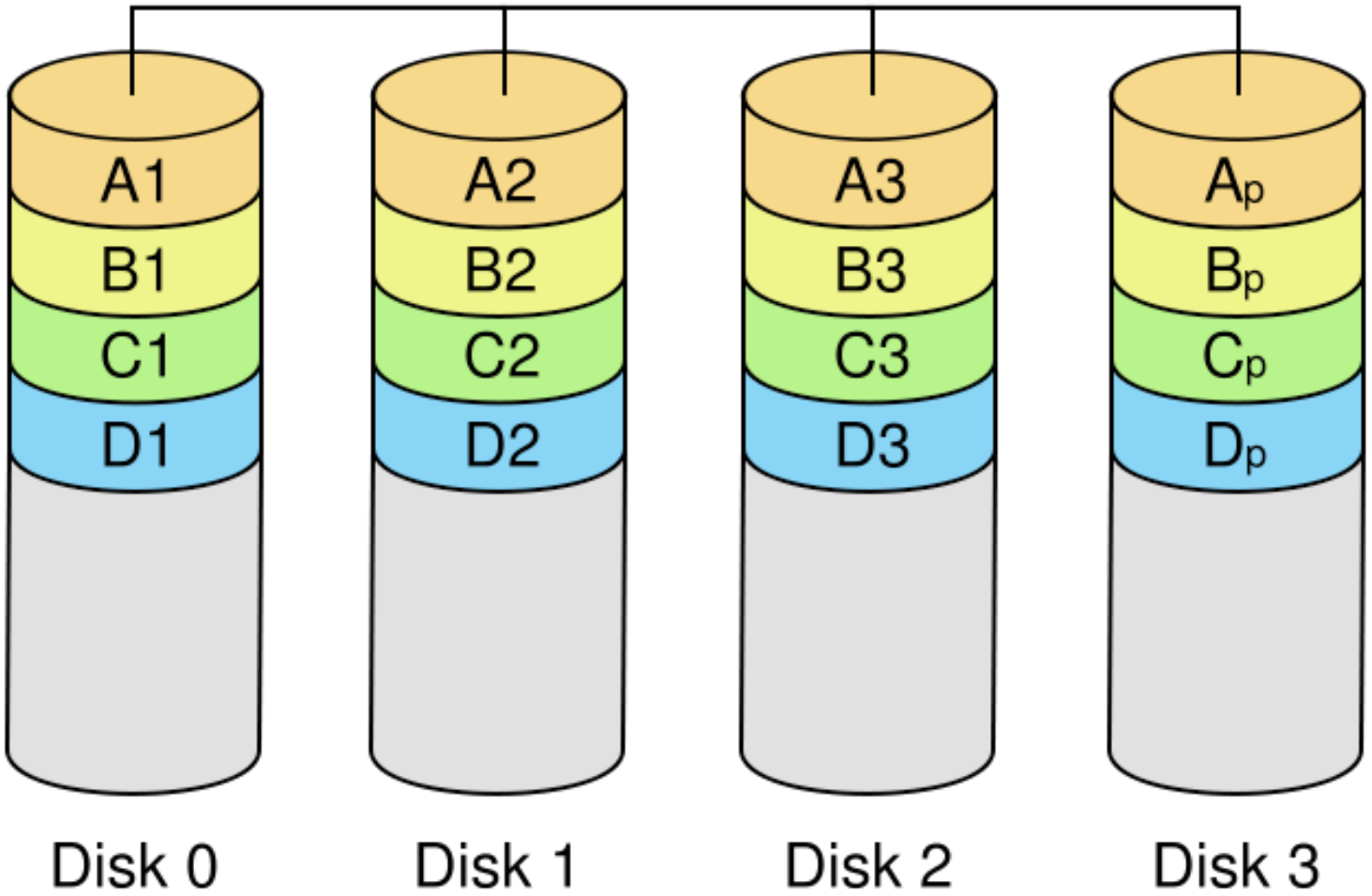


'**Mirroring**'. Provides fault tolerance from disk errors and failure of all but one of the drives. Increased read performance occurs when using a multi-threaded operating system that supports split seeks, very small performance reduction when writing.



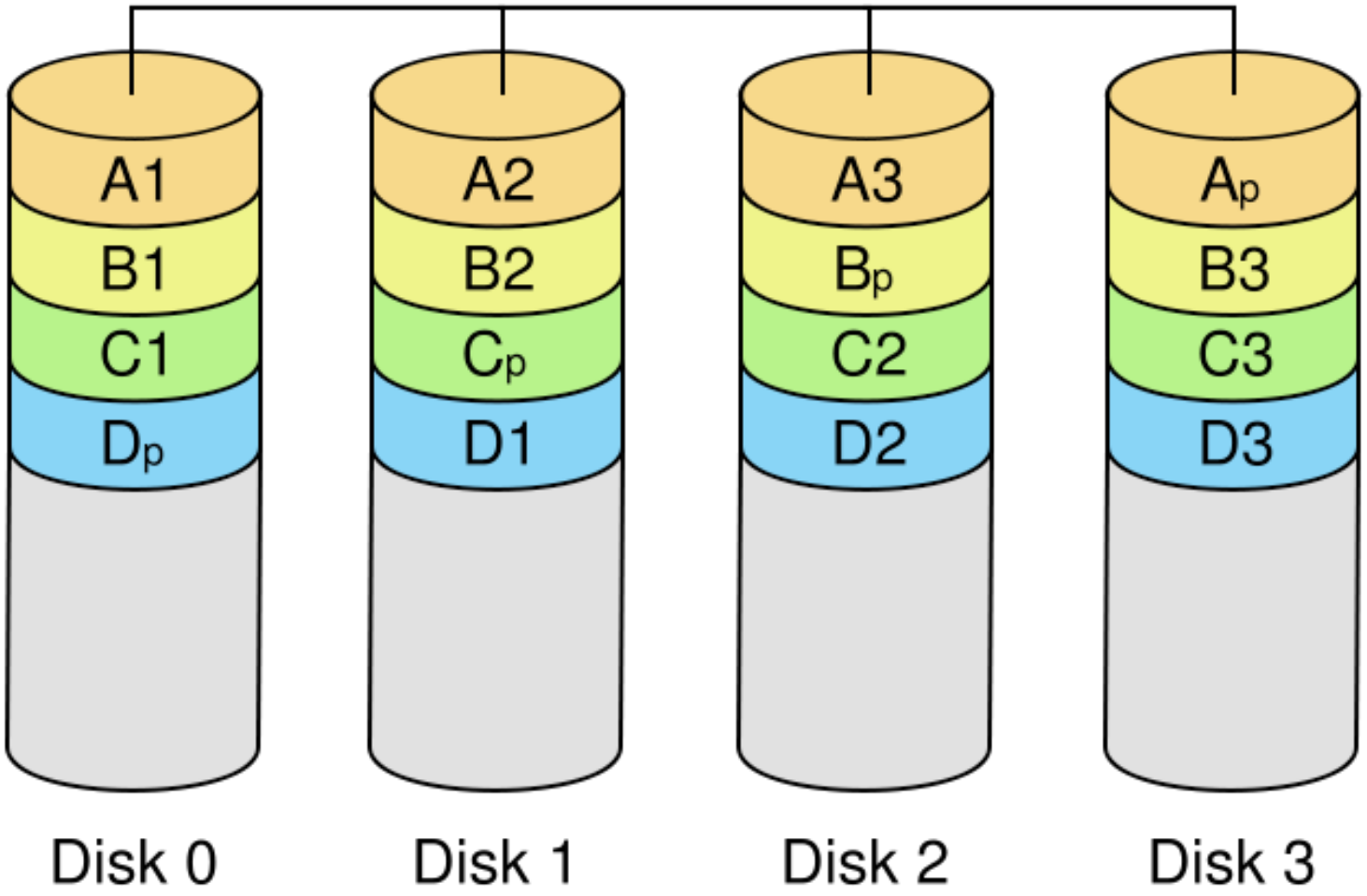
Striped set with dedicated parity or **bit interleaved parity** or **byte level parity**.
This mechanism provides fault tolerance similar to RAID 5. However, reads and writes to the array perform like a single drive. For this to work properly, the drives must have **synchronised** rotation. If one drive fails, the performance doesn't change.

RAID 4



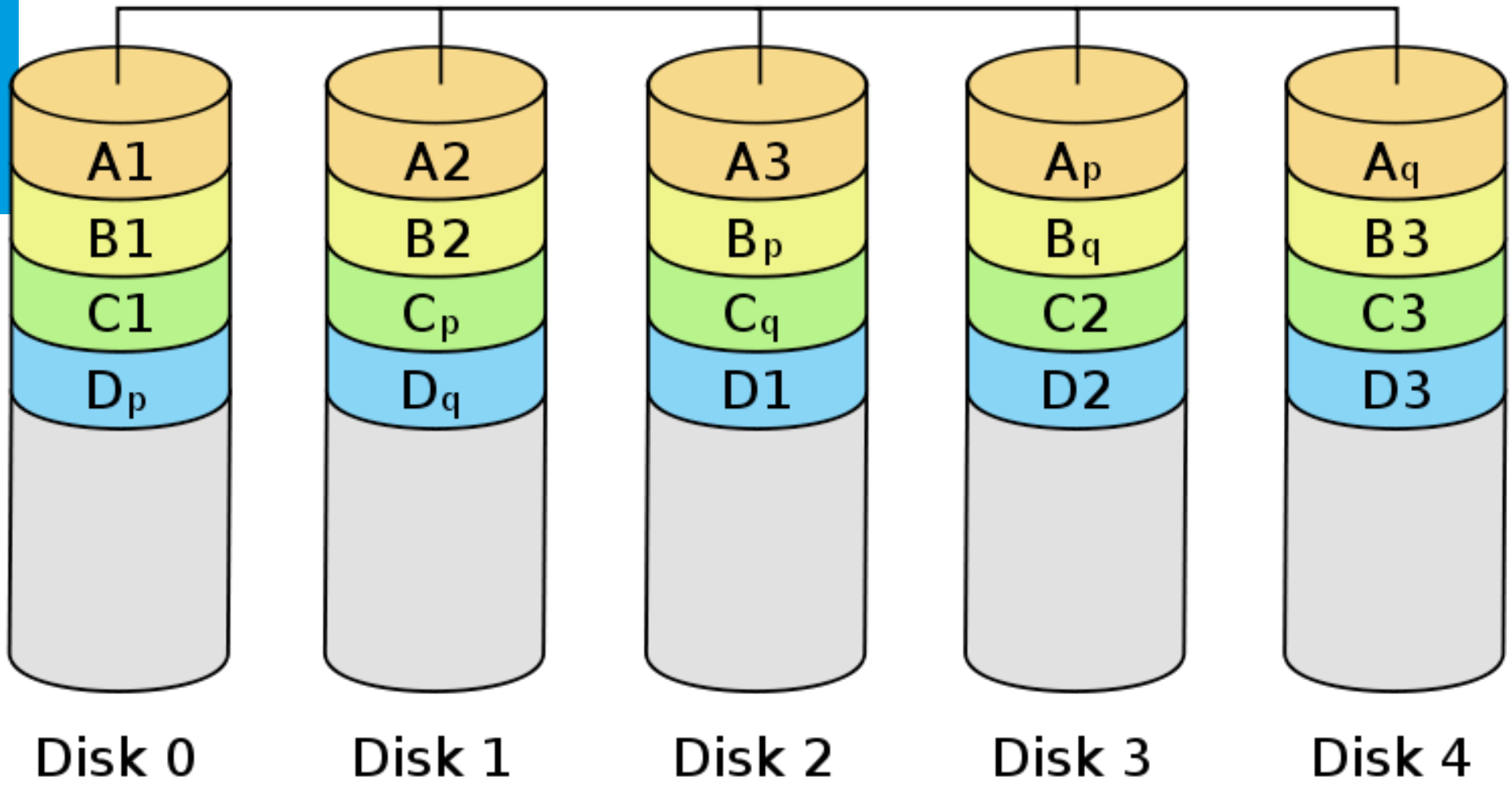
Block level parity. Identical to RAID 3, but does block-level striping instead of byte-level striping. In this setup, files can be distributed between multiple disks. Each disk operates independently which allows I/O requests to be performed in parallel, though data transfer speeds can suffer due to the type of parity. The error detection is achieved through dedicated parity and is stored in a separate, single disk unit.

RAID 5



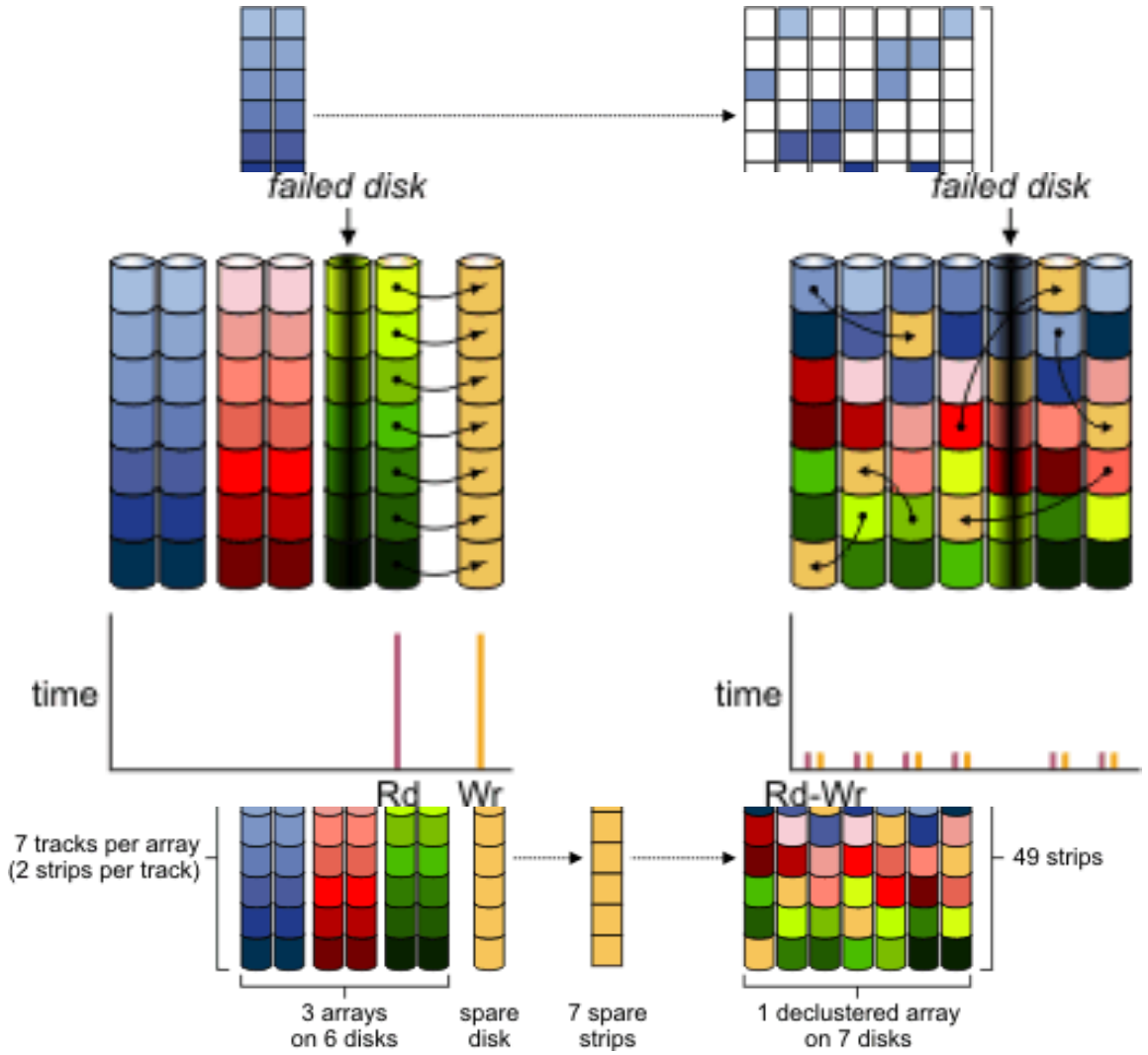
Striped set with distributed parity or **interleave parity**. Drive failure requires replacement, but the array is not destroyed by a single drive failure. Upon drive failure, any subsequent reads can be calculated from the distributed parity such that the drive failure is masked from the end user. A single drive failure in the set will result in reduced performance of the entire set until the failed drive has been replaced and rebuilt.

RAID 6



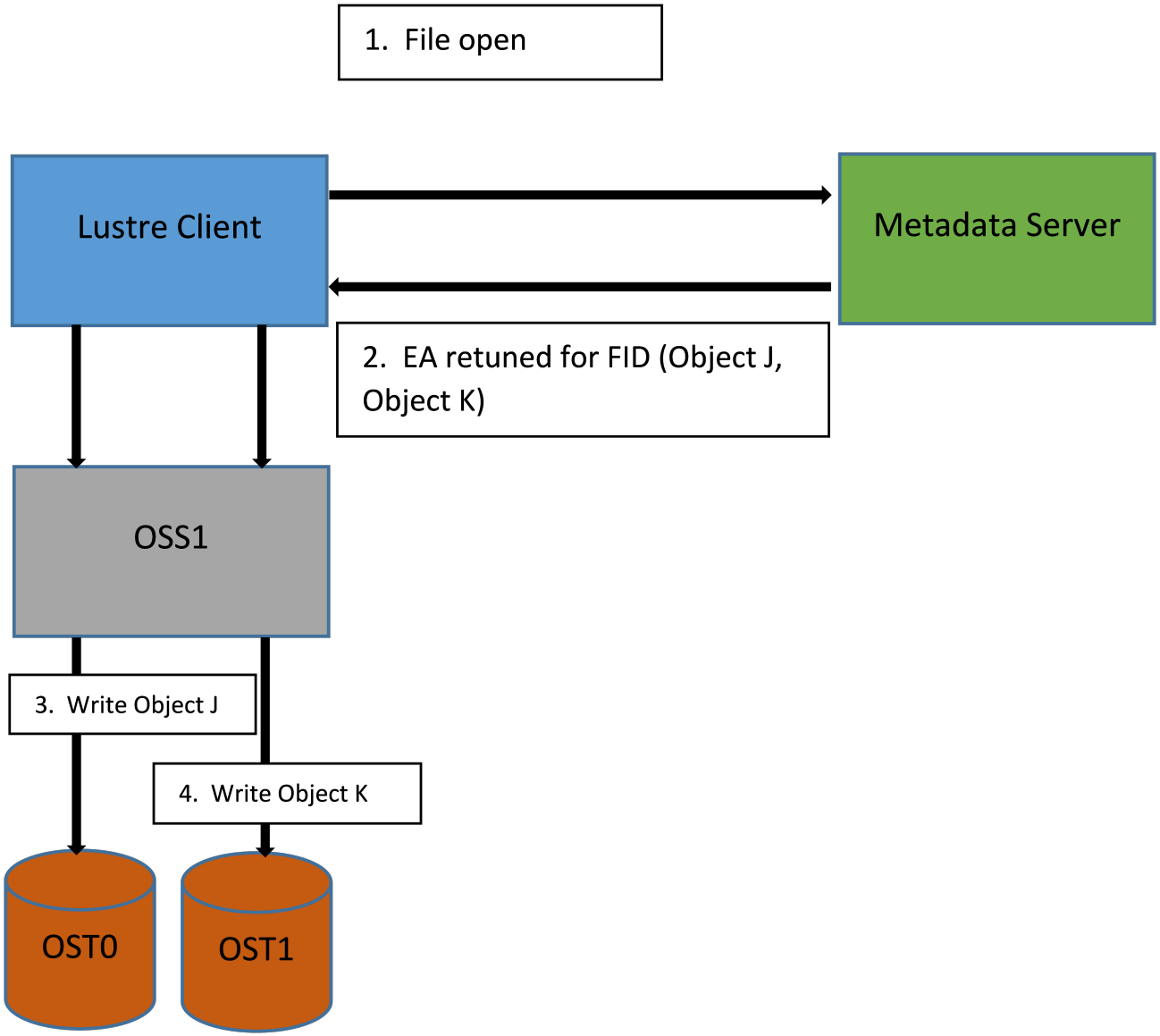
Striped set with dual distributed parity. Provides fault tolerance from two drive failures; array continues to operate with up to two failed drives. This becomes increasingly important because large-capacity drives lengthen the time needed to recover from the failure of a single drive.

Rebuilding failing disk with RAID

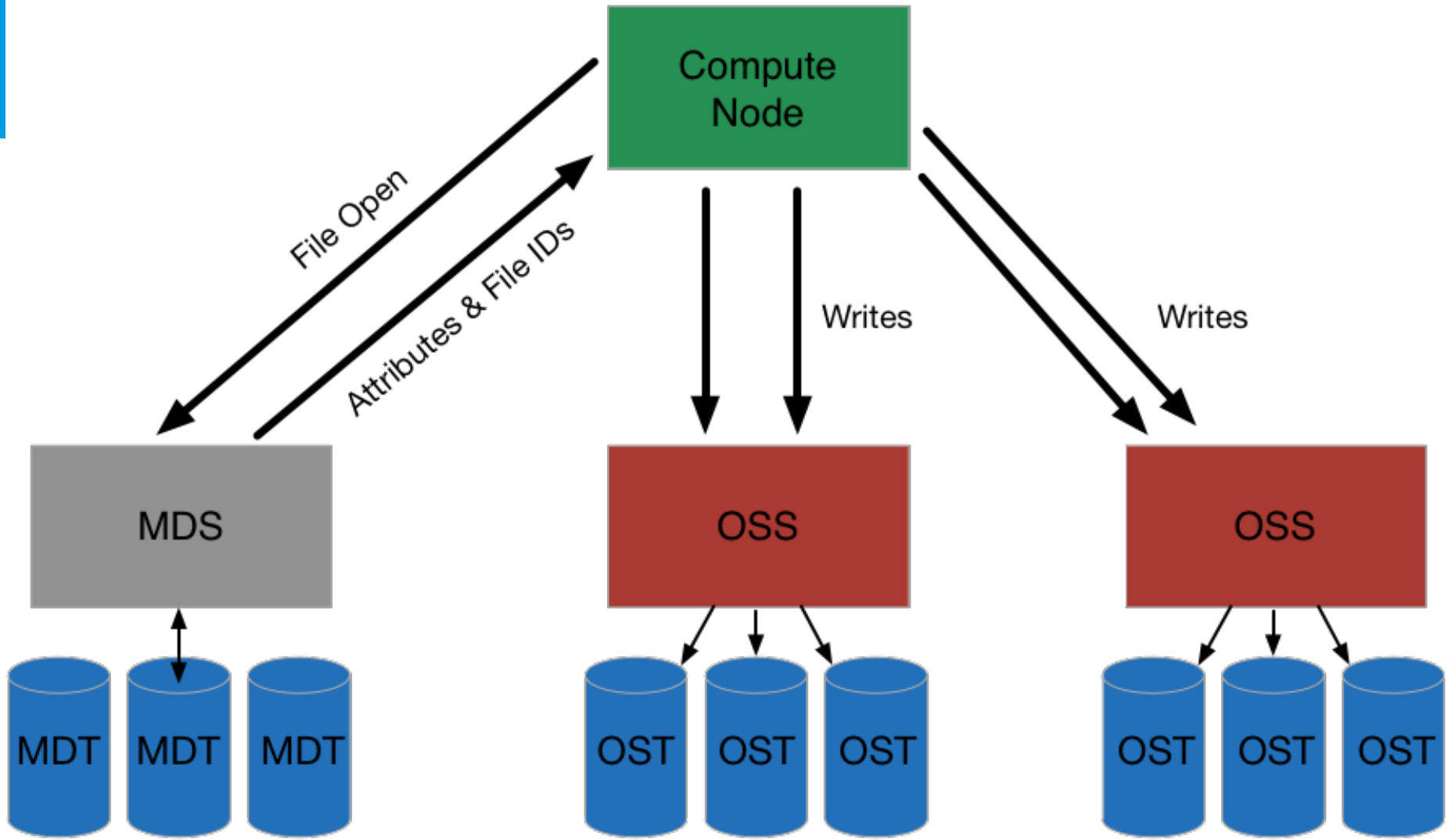


space disk blocks are distributed along the other disks

MetaData server



MetaData server

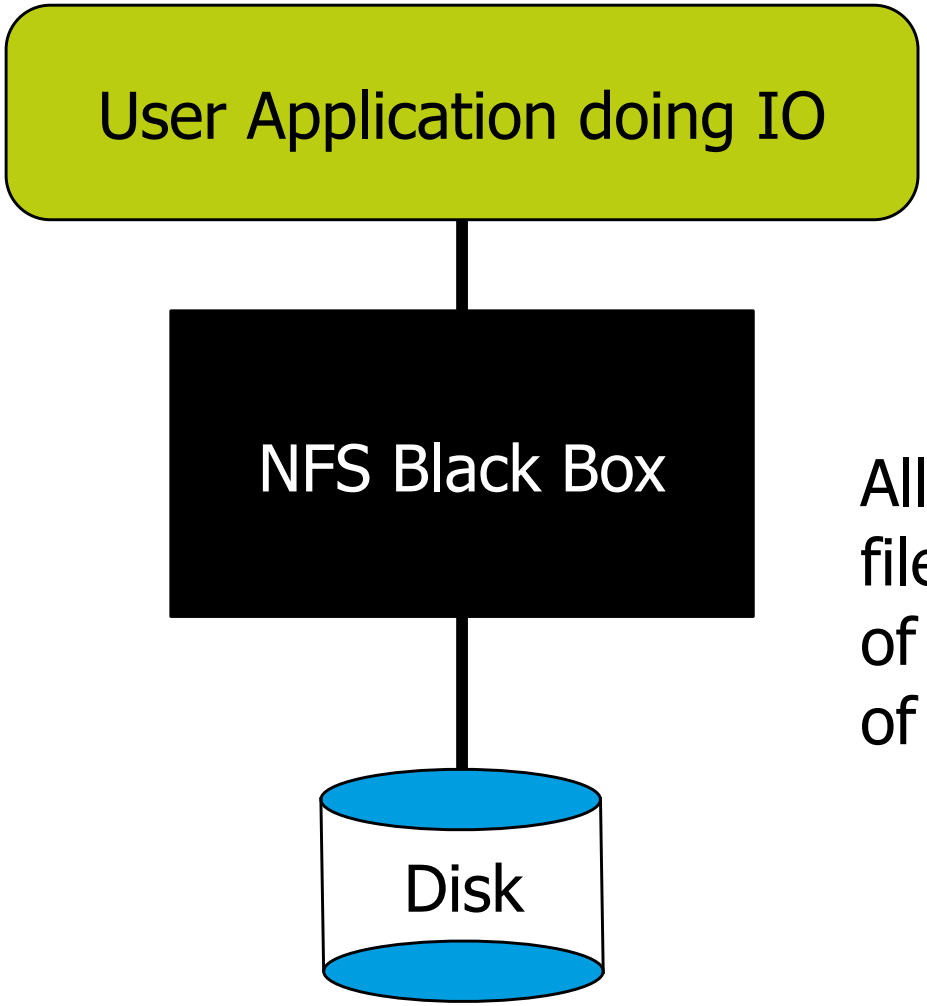


NFS

NFS

- Network File System
  - in 1985 created by SUN
- Transparency
  - server exports local filesystem to clients (exports)
  - clients mount the filesystem (fstab)
- Adds overhead due to communication protocol between server and client
- Concurrency: locking
- http://en.wikipedia.org/wiki/Network_File_System_(protocol)

What is NFS?



Goal:
Allow access to file independent of the location of the file

What is in the NFS Box?

- NFS is a suite of protocols on top of TCP or UDP, and IP.
- NFS is stateless - the server does not keep track or requests made by the client. Evert request must be self contained.
- NFS is not a good (fast) protocol for busy networks. Do not write large or many output file to NFS.
- try `mount` command

Implementation of OS's

Unix origin

- 1969 From Bell labs by Ken Thompson and Dennis Ritchie.
- Ideas from Multics on PDP-7
- UNiplexed Information and Computing Service => UNIX
- Thompson wanted to play a game he had written called Space Travel (a science-fiction simulation that involved navigating a rocket through the solar system).
- They knew from experience that the essence of communal computing, as supplied by remote-access, time-shared machines, is not just to type programs into a terminal instead of a keypunch, but to encourage close communication".
- 1969 was also the year the ARPANET (the direct ancestor of today's Internet) was invented.
- more info: http://en.wikipedia.org/wiki/Unix

History Unix

- During the development of Unix C was also developed (from B)
- Unix completely written in C and not in assembler
- During the 70 many universities contributed to Unix
- Berkeley developed open source BSD (Software Distribution) and vi

History Unix

- 1980, the Defense Advanced Research Projects Agency (DARPA) needed a team to implement its brand-new TCP/IP protocol stack on the VAX under Unix. DARPA chose Berkeley Unix as a platform — explicitly because its source code was available and unencumbered.
- After TCP/IP, everything changed. The ARPANET and Unix cultures began to merge at the edges, a development that would eventually save both from destruction.
- Then a disaster happened; the rise of Microsoft.

Standardization of Unix

- Mainly three parts:
  - ISO C
    - Support for a set of C-functions
  - IEEE POSIX
    - Support for a set of C-functions
  - The Single UNIX Specification
    - A superset of POSIX.1
    - Support SUS to get The UNIX trademark

Unix Implementations

- UNIX System V Release 4
  - A product of AT&T / Bell

    4.4BSD
  - The Berkley Software Distribution

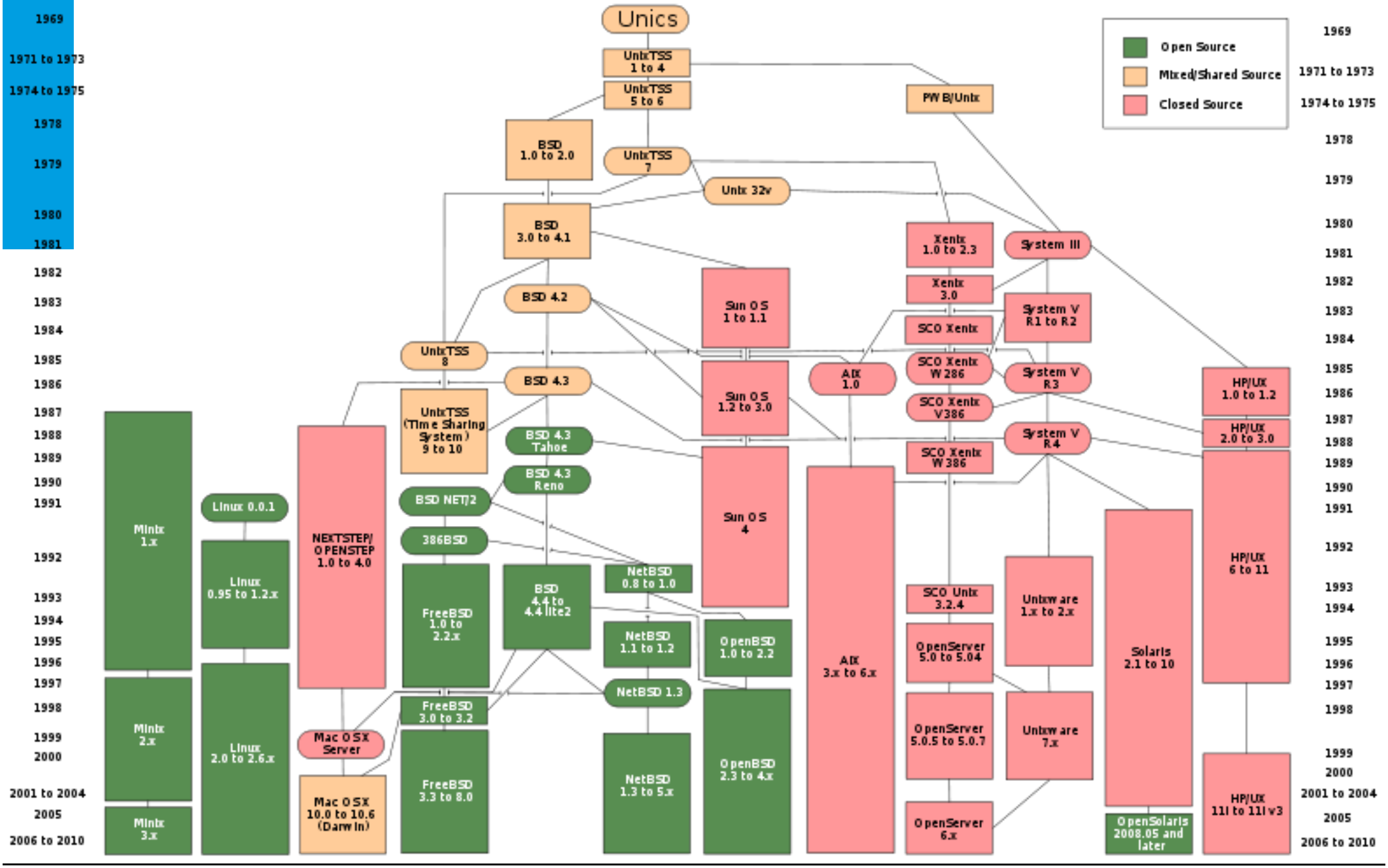
    FreeBSD
  - A 4.4BSD-Lite operating System

    Linux
  - Gnu/Linux operating system

    Solaris
  - Sun OS, formally certified as UNIX

    Mac OS X
  - Core system is ”Darwin”, partly built on FreeBSD

    more and variants
  - Lookout for POSIX-compliant...


Unics
UnixTSS 1 to 4
UnixTSS 5 to 6
UnixTSS 7
UnixTSS 8
UnixTSS (Time Sharing System) 9 to 10
PWB/Unix
BSD 1.0 to 2.0
BSD 3.0 to 4.1
BSD 4.2
BSD 4.3
BSD 4.3 Tahoe
BSD 4.3 Reno
BSD NET/2
386BSD
FreeBSD 1.0 to 2.2.x
FreeBSD 3.0 to 3.2
FreeBSD 3.3 to 8.0
BSD 4.4 to 4.4 lite2
Unix 32v
Sun OS 1 to 1.1
Sun OS 1.2 to 3.0
Sun OS 4
AIX 1.0
AIX 3.x to 6.x
Xenix 1.0 to 2.3
Xenix 3.0
SCO Xenix
SCO Xenix W 286
SCO Xenix V386
SCO Xenix W 386
SCO Unix 3.2.4
OpenServer 5.0 to 5.04
OpenServer 5.0.5 to 5.0.7
OpenServer 6.x
System III
System V R1 to R2
System V R3
System V R4
Unixware 1.x to 2.x
Unixware 7.x
Solaris 2.1 to 10
OpenSolaris 2008.05 and later
HP/UX 1.0 to 1.2
HP/UX 2.0 to 3.0
HP/UX 6 to 11
HP/UX 11i to 11i v3
NetBSD 0.8 to 1.0
NetBSD 1.1 to 1.2
NetBSD 1.3
NetBSD 1.3 to 5.x
OpenBSD 1.0 to 2.2
OpenBSD 2.3 to 4.x
NEXTSTEP/ OPENSTEP 1.0 to 4.0
Mac OS X Server
Mac OS X 10.0 to 10.6 (Darwin)
Linux 0.0.1
Linux 0.95 to 1.2.x
Linux 2.0 to 2.6.x
Minix 1.x
Minix 2.x
Minix 3.x
Open Source
Mixed/Shared Source
Closed Source
1969
1971 to 1973
1974 to 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 to 2004
2005
2006 to 2010

Handy Linux commands

- od -f -Ad ....binary file — inspects binary files
- nm -aA *.o — inspects object files
- ls -lart — list directory latest change at the bottom
- grep — looks for strings in files
- ldd — shows used dynamic libraries
- nohup — keeps program in background running
- ps — a snapshot of the process table
- top — snapshots of the process table
- bash scripts
- find . -name "*.f90" -exec grep alloc {} \; -print

Bash scripts

- Calculations and numbers

```
pi=$(echo "scale=10; 4*a(1)" | bc -l)
dxsrc=$(echo "scale=4; 25/2"| bc -l)

pfldr=$(printf %03d $fldr)
file=${file_base}${pfldr}.su

setenv x2 5
setenv aper `echo $x2 | gawk '{a=$1*2 ;
printf("%.8f\n",a)}'`
```

Bash scripts

- Loops

```
xsrc1=0
xsrc2=6000
xsrc=$xsrc1
while (( xsrc <= xsrc2 ))
do
     echo -n $xsrc
     (( xsrc = $xsrc + $dxsrc))
done
```

Bash scripts

- Loops

```
i=9109
while (( i <= 9294 ));
do qdel $i; (( i += 1 )); done;

for file in ref*;
do
     echo $file
     filename=${file%.*ps}
     convert $file ${filename}.EPS
done
```

Bash scripts

- Loops

```
for i in $( ls ); do echo item: $i; done

for i in `seq 1 10`;
do
     echo $i
done
```

Bash SU

```
sumax < modtmp.su outpar=nep
Mmin=`cat nep | awk '{print $2}'`
Mmax=`cat nep | awk '{print $1}'`
echo min=$Mmin max=$Mmax
```

Bash SU

```
fldr=1
for file in shotRS_A100_F30*;
do
     echo $file;
     pfldr=$(printf %03d $fldr);
     suwind  < $file key=tracl min=11 max=191 | \
     sunormalize > normA_${pfldr}.su;
          (( fldr += 1 ));
done
```

Bash

```
case "$src_type" in
    6)
        file_shot=data/shotB_Fx${xsrc}.su;
         ;;
    7)
        file_shot=data/shotB_Fz${xsrc}.su;
         ;;
    *)
        file_shot=data/shotB_${xsrc}.su;
         ;;
esac
```

Exercise: file IO

- The code measures writing and reading to file using different approaches.
- On your git clone: cd HPCourse/IO
- Check the README for instructions.
- The program produces a list with the data rate in MB/s written to and read from file.
- Cygwin users change size on line 55 to 32*1024*1024

Memory-Mapped Files

- Memory-mapped file I/O allows file I/O to be treated as routine memory access by **mapping** a disk block to a page in memory

- A file is initially read using demand paging. A page-sized portion of the file is read from the file system into a physical page. Subsequent reads/writes to/from the file are treated as ordinary memory accesses.

- Simplifies file access by treating file I/O through memory rather than **`read()`** **`write()`** system calls

- Also allows several processes to map the same file allowing the pages in memory to be shared

Exercise: Accuracy

- Simple programs to show the accuracy of floating point numbers.
- On your git clone: cd HPCourse/FloatPrecision
- Check the README for instructions.
- What have you learned from these examples?