Textbook:
Operating Systems – Internals and Design Principles,
by ***William Stallings.***
Publisher: Pearson.

Chapter 2
Operating Systems Overview

Part B

Textbook Section 2.3 **Process**

- **Fundamental to the understanding of OS**

A *process* can be defined, alternatively, as:

- A program in execution
- An instance of a running program
- The entity that can be assigned to, and executed on, a processor
- A unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources

Philosophical Remarks

It is philosophically debatable whether the entire world is ***ultimately*** made of “**things**” or whether the world is made of “**processes**” (and “events”).

The branches of philosophy in which such questions are discussed are **Metaphysics** and **Ontology**.

One of the most well-known and profound metaphysical-ontological works on this topic is ***“Process and Reality”*** (1929) by **Alfred North Whitehead**, who was both a mathematician and a modern philosopher. (*difficult stuff* !!!)

In a process-oriented ontology, “things” are temporal phenomena that only seem to exist (from our perspective), but which are in reality brought about by underlying processes.
Whereas “western” ontology was quite often “*substance*”-oriented, “eastern” ontology was quite often rather “*flow*”-oriented.

Process versus Algorithm

A recipe in a cook-book is not the cooking; it is only a description. The cook-book-recipe itself does not bring a meal onto the table. For the activity of cooking, much more than the recipe is needed: Edible ingredients, a kitchen with utensils, and a competent cook.

Also an Algorithm is merely a description – NOT a “happening”. For the Algorithm to become a Process, which is a “happening”, we need computer’s hardware with its internal electronic activities and resources, and we need the Operating System by which the resources are allocated and managed.

Process versus Algorithm

A recipe in a cook-book is not the cooking; it is only a description. The cook-book-recipe itself does not bring a meal onto the table. For the activity of cooking, much more than the recipe is needed: Edible ingredients, a kitchen with utensils, and a competent cook.

Also an Algorithm is merely a description – NOT a "happening". For the Algorithm to become a Process, which is a "happening", we need computer's hardware with its internal electronic activities and resources, and we need the Operating System by which the resources are allocated and managed.

For the definition of the term **Algorithm** please recapitulate **Chapter 8** of the COS**151** book, *Foundations of Computer Science*, studied in Sem.1

Process versus Algorithm

As there are many user application programs (each of which is described by its own algorithm), there are also many processes in the system.
To keep track of all these processes, the Operating System keeps a kind of "log book" which lists all currently registered processes and their corresponding resource allocations.

Because this "log book" is a **data structure**, the Operating System needs a specific section of the memory in which this "log book" is safely kept. Moreover, because the running Operating System is a process in the system, too, the Operating System's "log book" must also contain process information about itself !

For further explanations of **Data Structures** please remember **Chapter 11** of the COS**151** book, *Foundations of Computer Science*, studied in Sem.1

Attention! Confusion is looming!

Though **a process is** – strictly speaking – *a happening in time* – and **NOT a thing** – the textbook often speaks about processes somewhat sloppily **as if** they were things.

Occasionally it conflates the operating system's management information (in the "log book") **about** a process with the process itself – or it conflates the resources held by a process with that process as such.

For example: in Figure **2.8**, it seems as if two processes (A and B) would somehow "exist" (like things) in the memory – whereas all that actually exists in memory is data, whilst (with the availability of only 1 CPU) maximally 1 of A or B can be a "happening" in time.

"Components" of a "*Process*"

- A process "needs" three "components":
  - Executable **algorithm**
  - The associated **data** needed by the program (variables, work space, buffers, etc.)
  - The execution **context** (or "process status") of the running program

- The execution **context** is essential:
  - It is the **management data** by which the OS is able to supervise and control the process
  - Includes the **contents of the various process registers**
  - Includes **information about the priority** of the process and whether the process is waiting for the completion of a particular I/O event

Please keep in mind the confusion that was mentioned on the previous slide

"Components" of a "*Process*"

At this point,
Please press PAUSE,
and look once again at
Sub-Section 5.6.7 – Cycles – in the **COS151** book "**Foundations of Computer Science**".
Thereafter you can continue with this Lecture.

- The execution context is essential:
  - It is the management data by which the OS is able to supervise and control the process
  - Includes the contents of the various **process registers**
  - Includes information about the priority of the process and whether the process is waiting for the completion of a particular I/O event

Process Management

- The entire state of the process at any instant is contained in its context
- New features can be designed and incorporated into the OS by expanding the context to include any new information needed to support the feature

Thus: Process Management also **requires Memory Management**

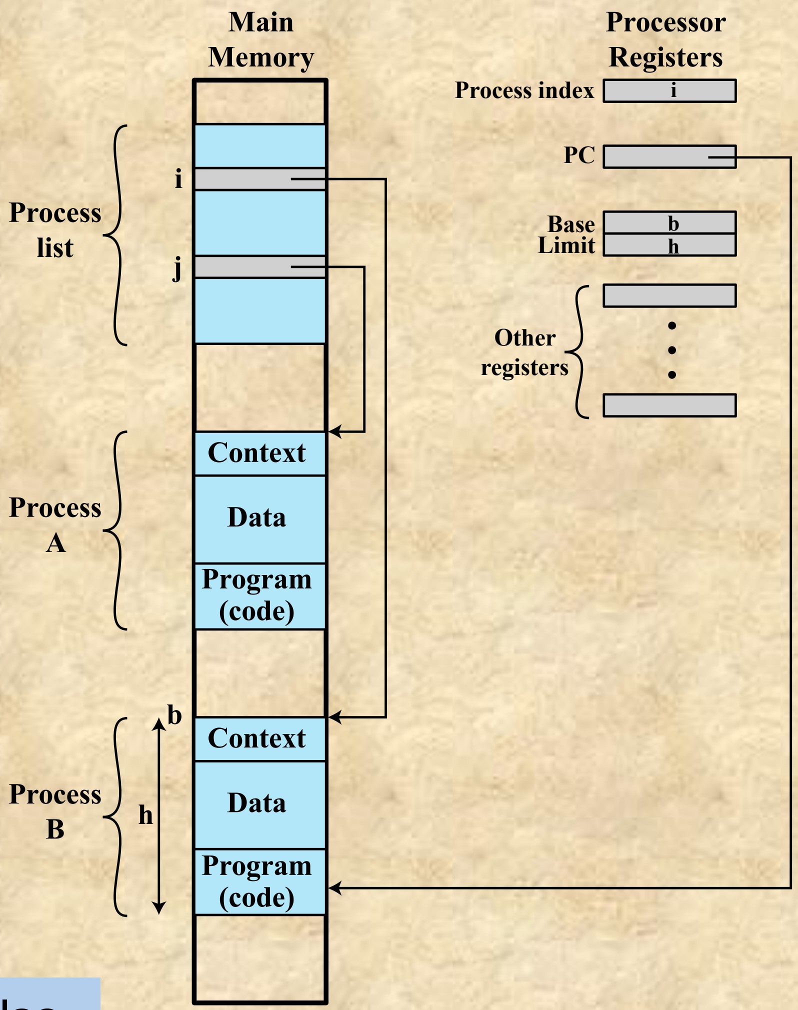


Figure 2.8 Typical Process Implementation

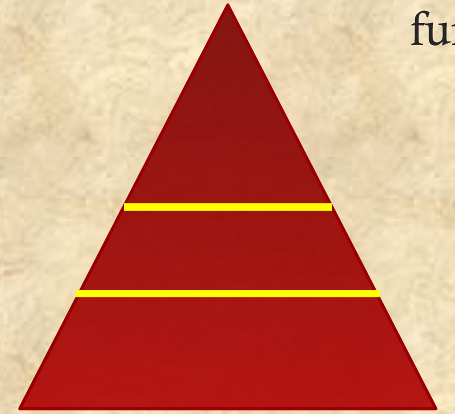
Memory Management

- The OS has several storage management responsibilities:

- Process isolation
- Automatic allocation and management
- Support of modular programming
- Protection and access control
- Long-term storage

"*Memory Hierarchy*"

- Recall **Sub-section 5.3.3**, and therein particularly **Figure 5.4**, of our COS**151 Introduction** book "***Foundations of Computer Science***"
- Throughout this Operating Systems course COS122 **it is presumed that you are already familiar** with those fundamental concepts



Important OS Terminology: "*Transparent*"

- In POLITICS = "visible",
- For example such that citizens can "see" what is going on in governance and public administration

- In **COMPUTER SCIENCE** = "**in**visible" (like a perfectly clean glass window)!
- **For example: "Network Transparency"** means that the users are **not** aware that they are working on a complicated network infrastructure: all they can see are the available resources and the other users – as if they would be working on one monolithic device.

Virtual Memory

- An OS implementation technique that allows programs to address memory from an abstract point of view, without regard to the amount of main memory physically available
  - Here is, again, "*transparency*": the user application programs do **not see** whether they access the main memory or the external storage
- Was conceived to enable multiple user processes holding portions of main memory at the same time

The Memory Management Technique of *Paging*

- Allows processes own a number of fixed-size memory blocks, called **pages**
- Program references a word by means of a *virtual address,* consisting of a page number and an offset within the page
- Each page of a process may be located anywhere in memory
- The paging system provides for a dynamic mapping between the virtual address used in the program and a *real address* (or physical address) in main memory
- Because I/O operations are "expensive" (slow), entire pages (not single bytes) are transferred from disc to main memory when a virtual address is translated to a real address outside the range of the main memory. **Note that access to the H-Disc is an I/O operation!**

The mapping between Virtual Addresses and Real Addresses must happen very often, whenever a process accesses a Virtual Address.

Hence it is important that the Address Translation Mechanism does not consume a lot of time.

For this reason, the Mapping between Virtual and Real Addresses is Implemented in Hardware.

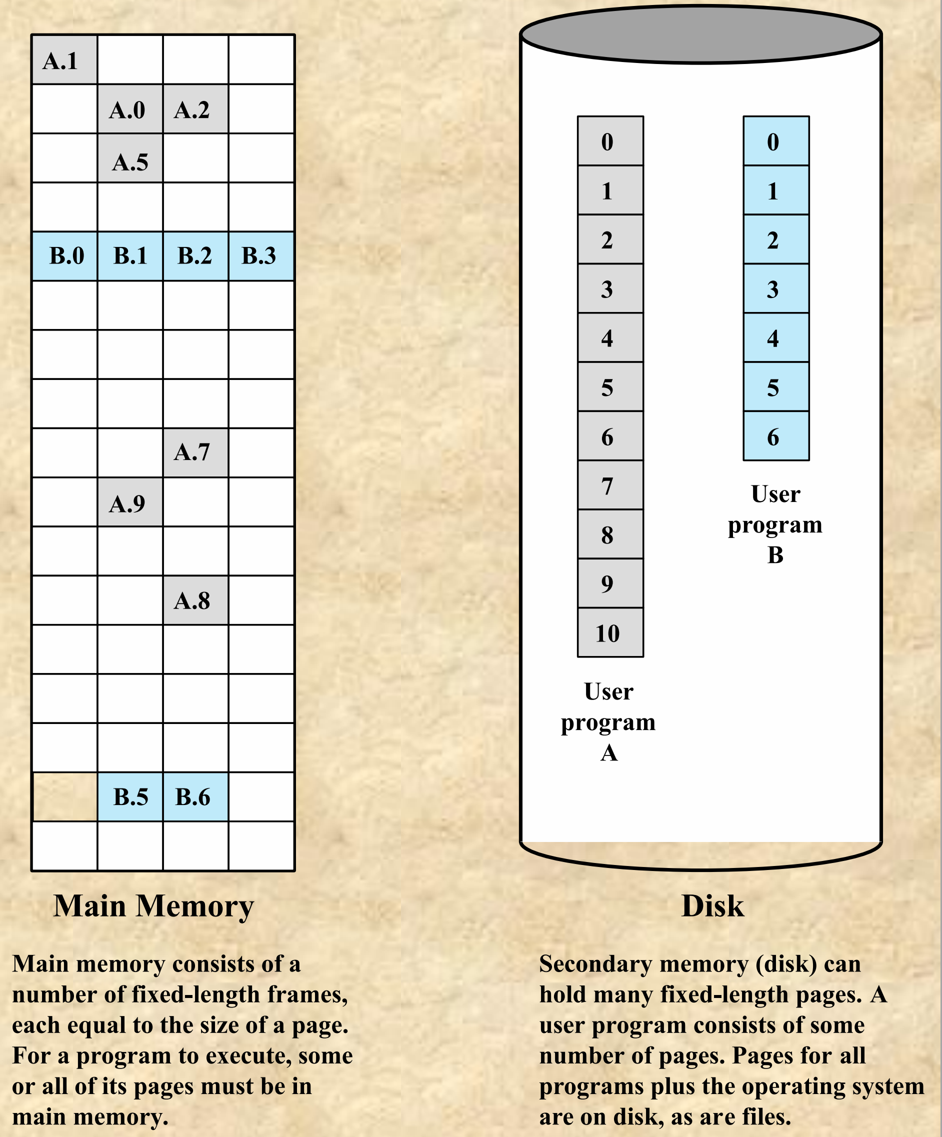
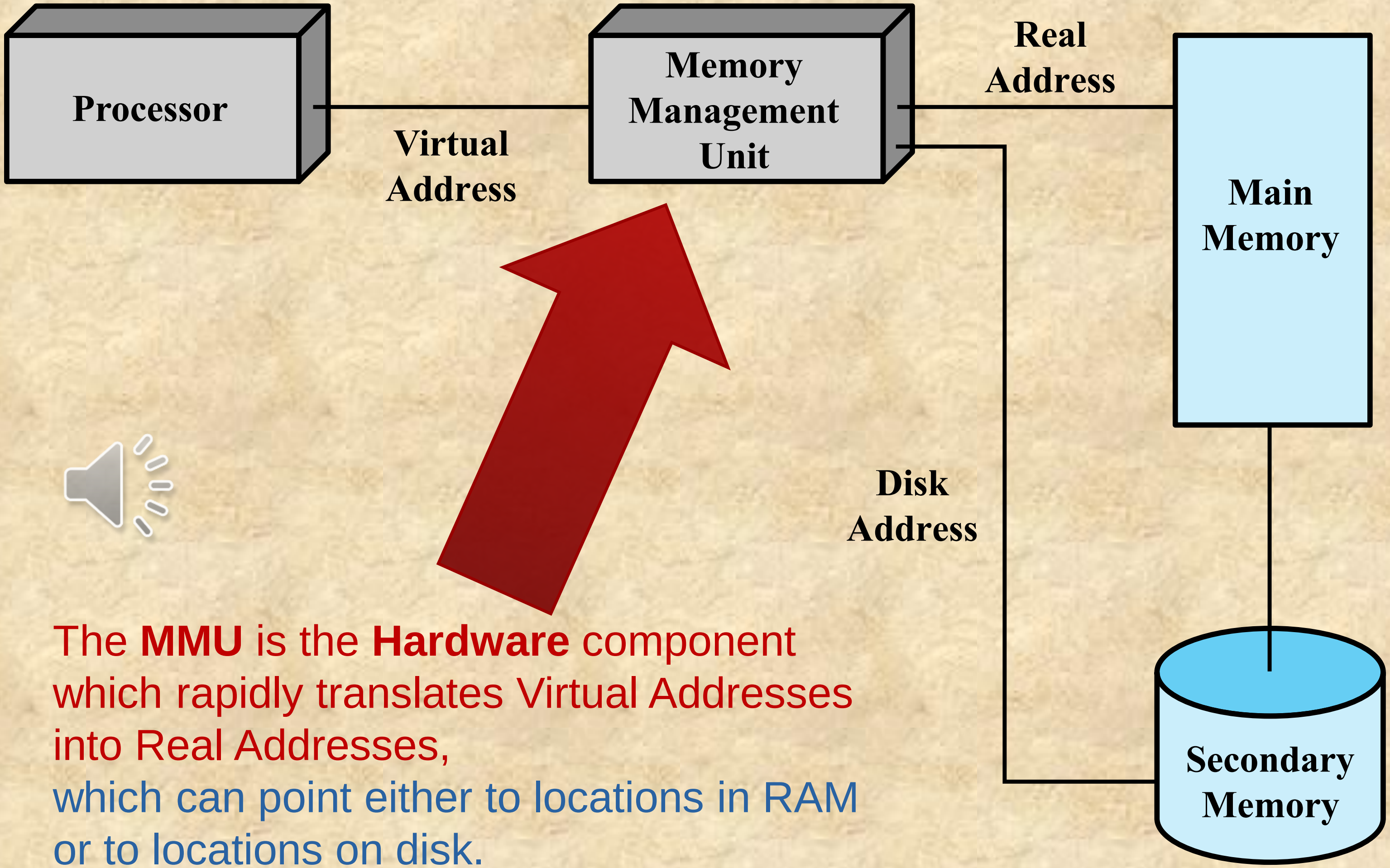

A.1
A.0
A.2
A.5
B.0
B.1
B.2
B.3
A.7
A.9
A.8
B.5
B.6
Main Memory
Main memory consists of a number of fixed-length frames, each equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.
0 1 2 3 4 5 6 7 8 9 10
User program A
0 1 2 3 4 5 6
User program B
Disk
Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages for all programs plus the operating system are on disk, as are files.

Figure 2.9 Virtual Memory Concepts

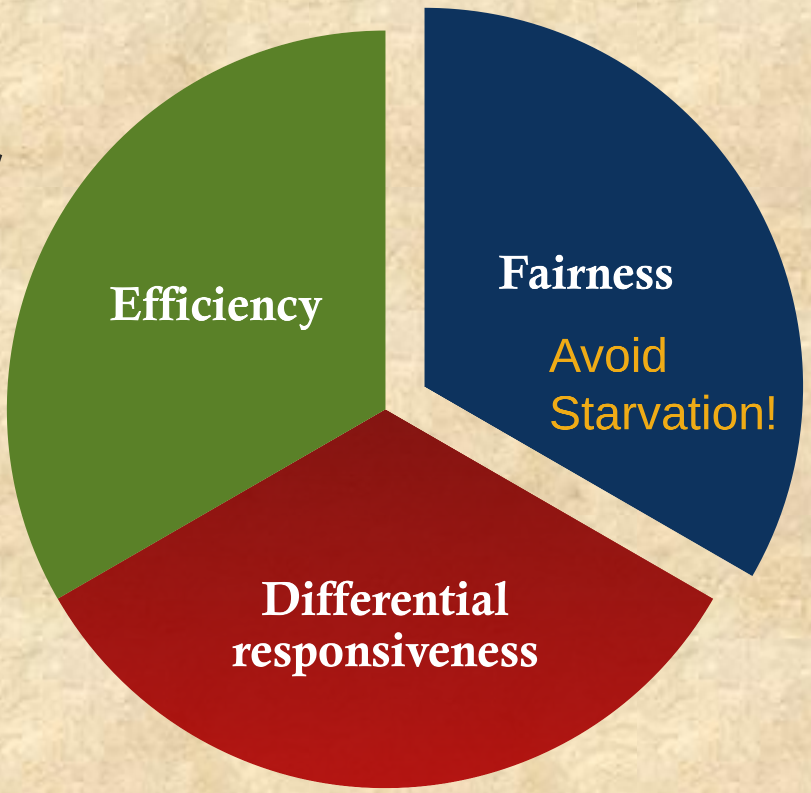


The **MMU** is the **Hardware** component which rapidly translates Virtual Addresses into Real Addresses,
which can point either to locations in RAM or to locations on disk.
Page-Swapping is triggered when a disk Address was identified as the real location of a Virtual Address

Figure 2.10 Virtual Memory Addressing

Scheduling and Resource Management

- Key responsibility of an OS is managing resources
- Resource allocation policies must consider:



2.4 Different Architectural Approaches

History of Computing is going on…

- Demands on operating systems require new ways of organizing the OS

Different approaches and design elements have been tried:

- Microkernel architecture
- Multithreading
- Symmetric multiprocessing
- Distributed operating systems
- Object-oriented design

Microkernel Architecture

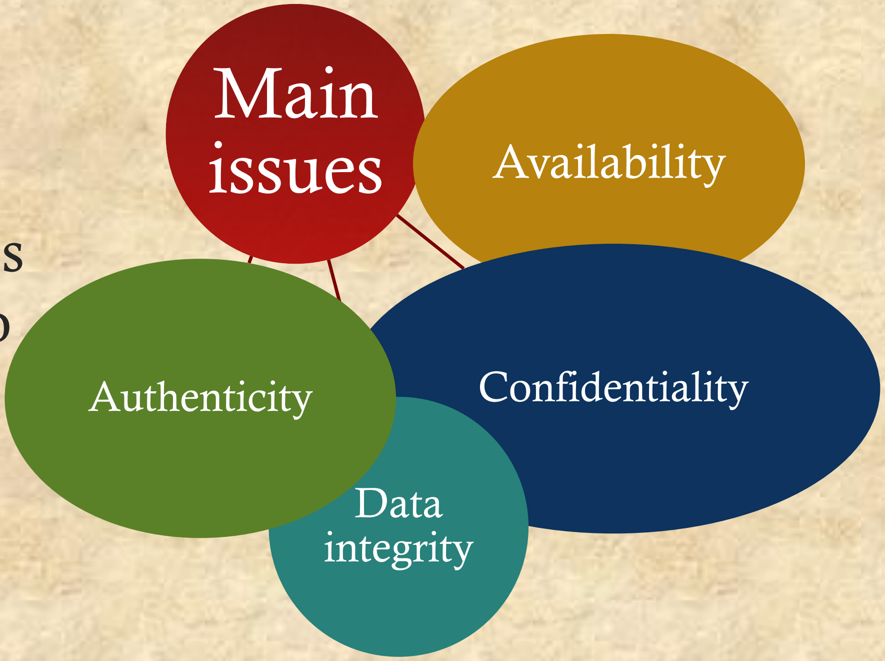
- Assigns only a few essential functions to the kernel:
  - Address space management
  - Interprocess communication (IPC)
  - Basic scheduling
- The approach:
  - Simplifies implementation
  - Provides flexibility
  - Well suited to a distributed environment

2.5 Fault Tolerance

- Refers to **the ability of a system or component to continue normal operation despite the presence of hardware or software faults**
- Typically involves some degree of **redundancy (duplication)**
- Intended to **increase the reliability** of a system
  - Typically comes with a cost in financial terms or performance
- The extent of adoption of fault tolerance measures must be determined by how critical the resource is: **Safety-Critical Systems must be highly fault tolerant**: Think, for example, of software in robots, software that controls industrial factories, etc.

Some Aspects of Fault Tolerance: Protection and Security

- The problem involves **controlling access** to computer systems and the information stored in them, in order to prevent harm.



A compromised system must be "fixed" and is then not available

Possible *Internal* Causes of Faults (some Examples)

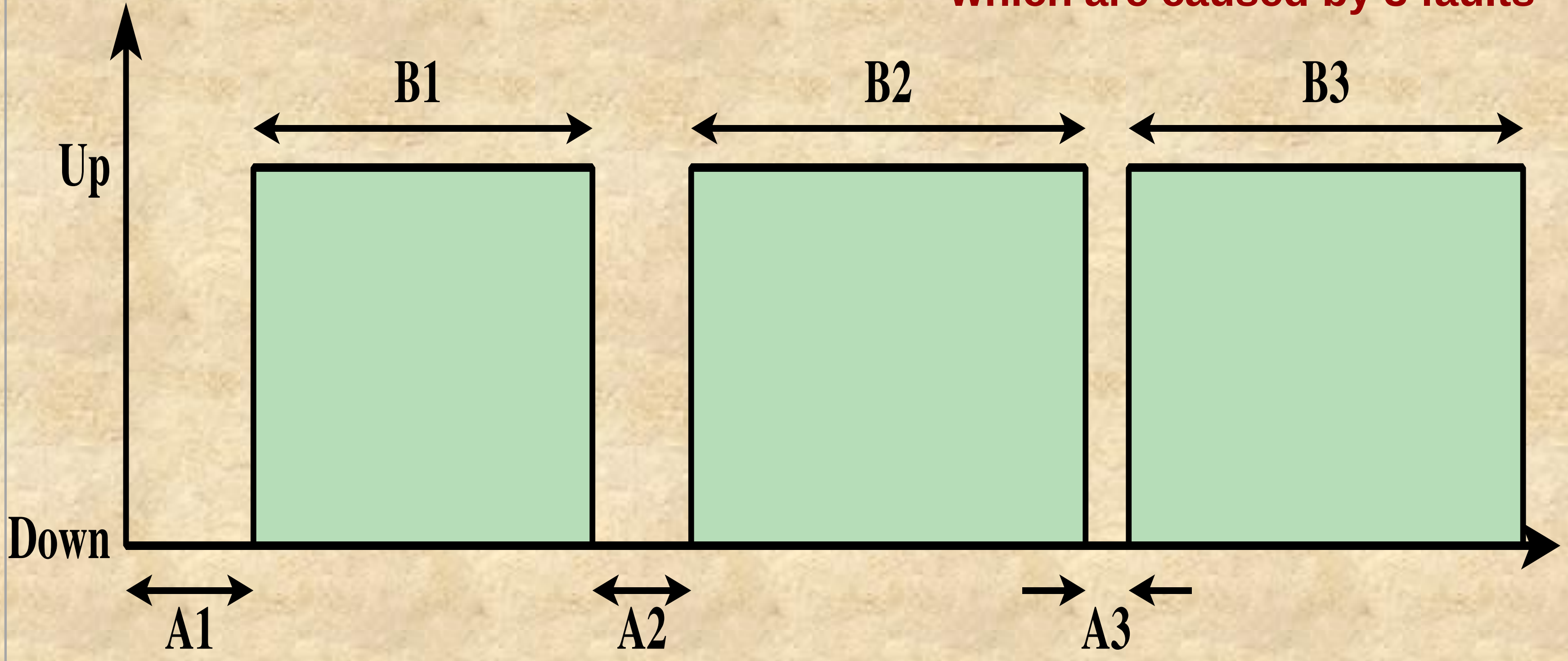
- **Improper synchronization**
  - It is often the case that a routine must be suspended awaiting an event elsewhere in the system
  - Improper design of the signaling mechanism can result in loss or duplication
- **Failed mutual exclusion**
  - More than one user or program attempts to make use of a shared resource at the same time
  - There must be some sort of mutual exclusion mechanism that permits only one routine at a time to perform an update against the file
- **Nondeterminate program operation**
  - When programs share memory, and their execution is interleaved by the processor, they may interfere with each other by overwriting common memory areas in unpredictable ways
  - The order in which programs are scheduled may affect the outcome of any particular program
- **Deadlocks**
  - It is possible for two or more programs to be hung up waiting for each other

➔ **Chapters 5 and 6 of this Course**

Fundamental *Notions* in *Reasoning about Faults*

- The basic **measures** are:
  - **Reliability**
    - *R(t)*
    - Defined as **the probability of a system's correct operation up to time *t* given that the system was operating correctly at time *t=0***
  - Mean time to failure (MTTF), closely related to MTTR:
    - Mean time to repair (**MTTR**) is the **average time it takes to repair or replace a faulty element**
  - **Availability**
    - Defined as **the fraction of time the system is available to service users' requests**

Example: **a system-run with 3 up-times and 3 down-times which are caused by 3 faults**



$$MTTF = \frac{B1 + B2 + B3}{3} \qquad MTTR = \frac{A1 + A2 + A3}{3}$$

Figure 2.14

Class	Availability	Annual Downtime
Continuous	1.0	0
Fault Tolerant	0.99999	5 minutes
Fault Resilient	0.9999	53 minutes
High Availability	0.999	8.3 hours
Normal Availability	0.99 - 0.995	44-87 hours

Table 2.4 Availability Classes

Figure 2.14

Class	Availability	Annual Downtime
Continuous	1.0	0
Fault Tolerant	0.99999	5 minutes
Fault Resilient	0.9999	53 minutes
High Availability	0.999	8.3 hours
Normal Availability	0.99 - 0.995	44-87 hours

Homework Exercise:
Calculate whether you have "normal"
Availability of Electricity in your home where you live,
whereby "**normality**" is defined as
maximally 90 hours of down-time per year.

“Faults”

- Are **defined** by the IEEE Standards Dictionary as **an erroneous hardware or software state** resulting from:
  - Component failure
  - Operator error
  - Physical interference from the environment
  - Design error
  - Program error
  - Data structure error

- The standard also states that **a fault manifests itself** as:
  - A **defect** in a hardware device or component
  - An **incorrect step**, process, or data definition in a computer program

Fault Categories

- **Permanent**
  - A fault that, after it occurs, is always present
  - The fault persists until the faulty component is replaced or repaired
- **Temporary**
  - A fault that is not present all the time for all operating conditions
  - Can be classified as
    - **Transient** – a fault that occurs only once
    - **Intermittent** – a fault that occurs at multiple, unpredictable times

Fault Tolerance by means of Redundancy:

Methods of Redundancy

Spatial (physical) redundancy

Involves the use of multiple components that either perform the same function simultaneously or are configured so that one component is available as a backup in case of the failure of another component

Temporal redundancy

Involves repeating a function or operation when an error is detected

Is effective with temporary faults but not useful for permanent faults

Information redundancy

Provides fault tolerance by replicating or coding data in such a way that bit errors can be both detected and corrected

➔ Section **11.6**

Operating System's Support Mechanisms

- A number of techniques can be incorporated into OS software to **support fault tolerance:**
  - **Process Isolation**: ➔ Ch.7 (memory management)
  - **Concurrency Controls**: ➔ Ch.5-6
  - **"Sand-box" Virtual Machines**: ➔ Ch.14
  - **Checkpoints and Rollbacks** ➔ ***similar*** *to the after-crash recovery techniques* in **Database Management** Systems (**DBMS**): **see 2nd-Year and 3rd-Year COS courses** on this topic.

Hardware with *many* CPUs: → Symmetric Multiprocessing (SMP)

- Term that refers to a computer hardware architecture and also to the OS behavior that exploits that architecture
- The OS of an SMP schedules processes across all of the available CPUs
- The OS must provide tools and functions to exploit the parallelism in an SMP system
- An attractive feature of an SMP is that the existence of multiple processors is **transparent** (*invisible*) to the user

SMP Advantages

Performance	More than one process can be running simultaneously, each on a different processor
Availability	Failure of a single process does not halt the system
Incremental Growth	Performance of a system can be enhanced by adding an additional processor
Scaling	Vendors can offer a range of products based on the number of processors configured in the system

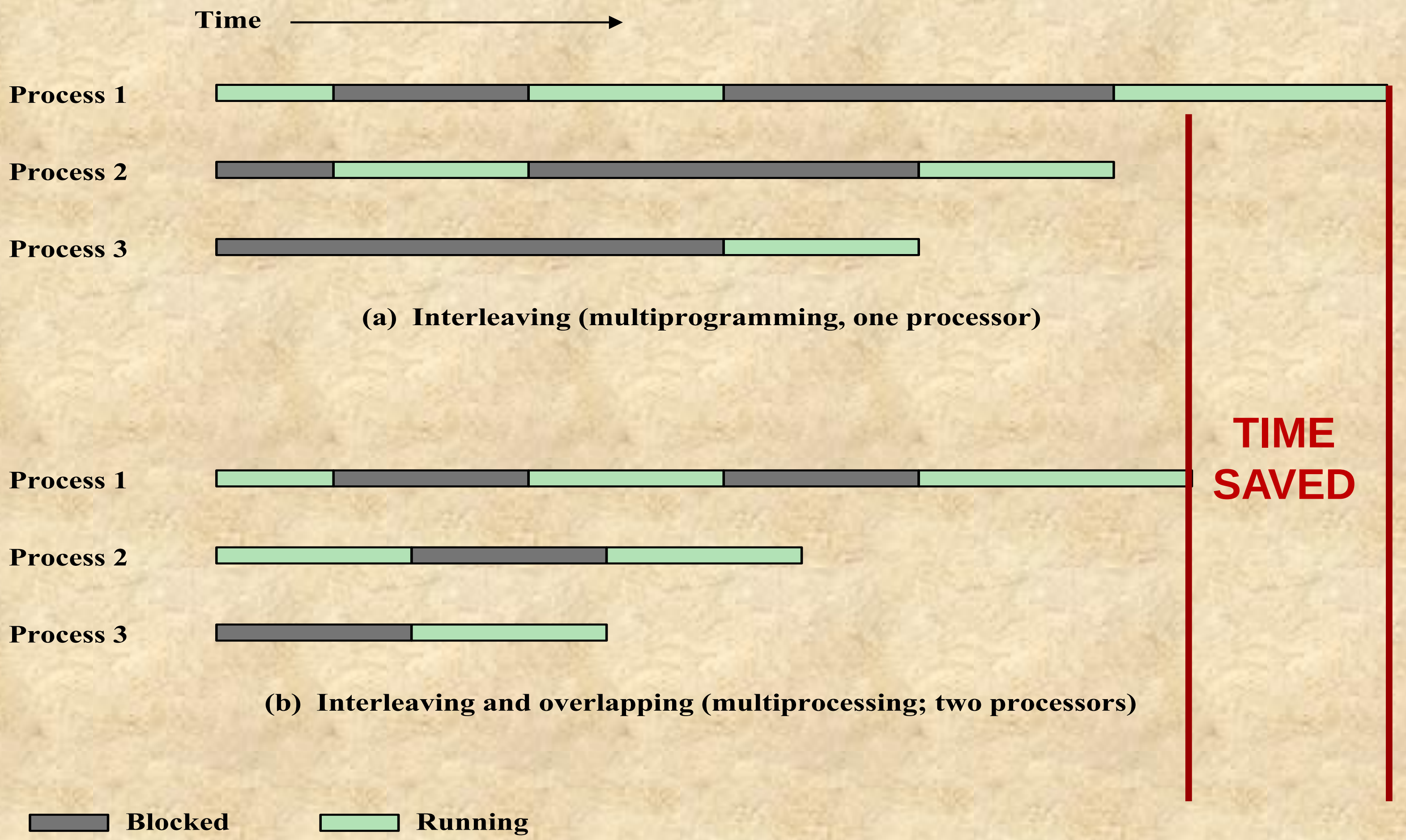

Time
Process 1
Process 2
Process 3
(a) Interleaving (multiprogramming, one processor)
TIME SAVED
Process 1
Process 2
Process 3
(b) Interleaving and overlapping (multiprocessing; two processors)
Blocked
Running

Figure 2.12 Multiprogramming and Multiprocessing

Symmetric Multiprocessor OS Design Considerations

- A multiprocessor OS must provide all the functionality of a multiprogramming system plus additional features to accommodate multiple processors
- **Key design issues:**

Simultaneous concurrent processes or threads

Kernel routines need to be reentrant to allow several processors to execute the same kernel code simultaneously

Scheduling

Any processor may perform scheduling, which complicates the task of enforcing a scheduling policy

Synchronization

With multiple active processes having potential access to shared address spaces or shared I/O resources, care must be taken to provide effective synchronization

Memory management

The reuse of physical pages is the biggest problem of concern

Reliability and fault tolerance

The OS should provide graceful degradation in the face of processor failure

Conclusion of Part B

- With this lecture **we have reached the end of Section 2.6** of our **Operating Systems** Textbook,
  - and you are now asked to **study the details from the book**.

- Moreover, our discussions thus far have been based on several ***fundamental concepts***, which you are also asked to recapitulate:
  - ***Algorithms*** from Chapter **8** of COS**151**,
  - ***Data Structures*** from Chapter **11** of COS**151**,
  - ***Computer Organisation*** from Chapter **5** of COS**151**.