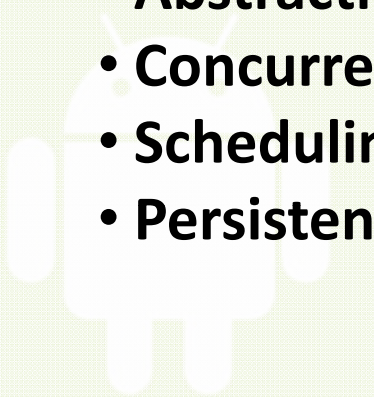
Operating Systems COMS(3010A) Introduction

Branden Ingram

branden.ingram@wits.ac.za

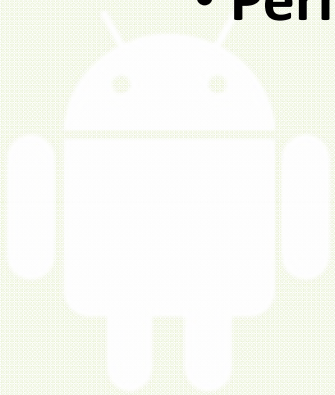
CONTENT

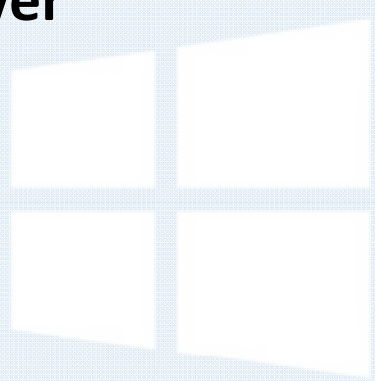
- Introduction
- Abstraction/Virtualisation
- Concurrency
- Scheduling
- Persistence



Why is COMS3010 important

- Some of you will actually design and build operating systems or components of them.
  - Perhaps more now than ever





Why is COMS3010 important

- Some of you will actually design and build operating systems or components of them.
  - Perhaps more now than ever

- Many of you will create systems that utilize the core concepts in operating systems.
  - Whether you build software or hardware
  - The concepts and design patterns appear at many levels

Why is COMS3010 important

- Some of you will actually design and build operating systems or components of them.
  - Perhaps more now than ever

- Many of you will create systems that utilize the core concepts in operating systems.
  - Whether you build software or hardware
  - The concepts and design patterns appear at many levels

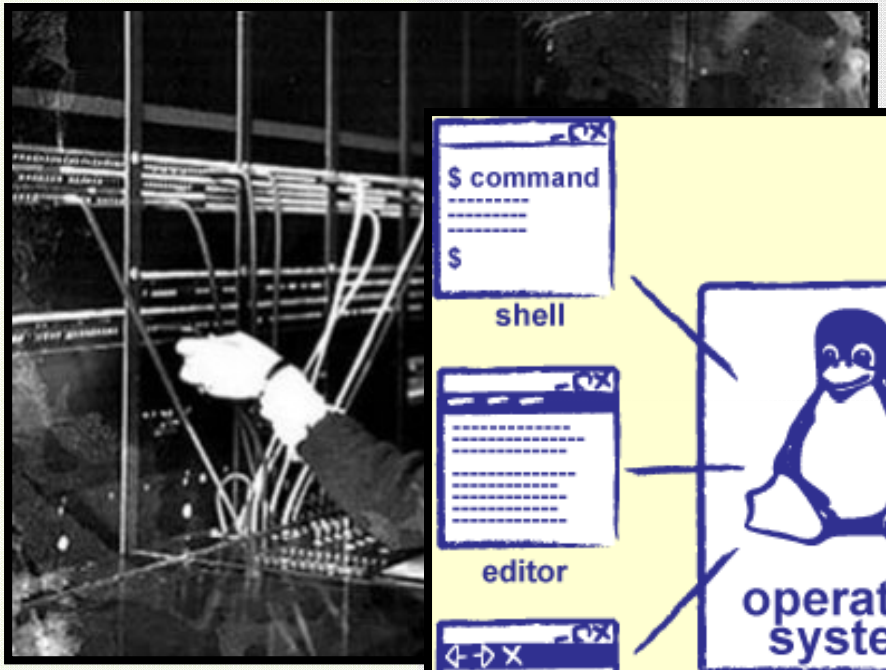
- All of you will build applications, etc. that utilize operating systems
  - The better you understand their design and implementation, the better use you'll make of them.

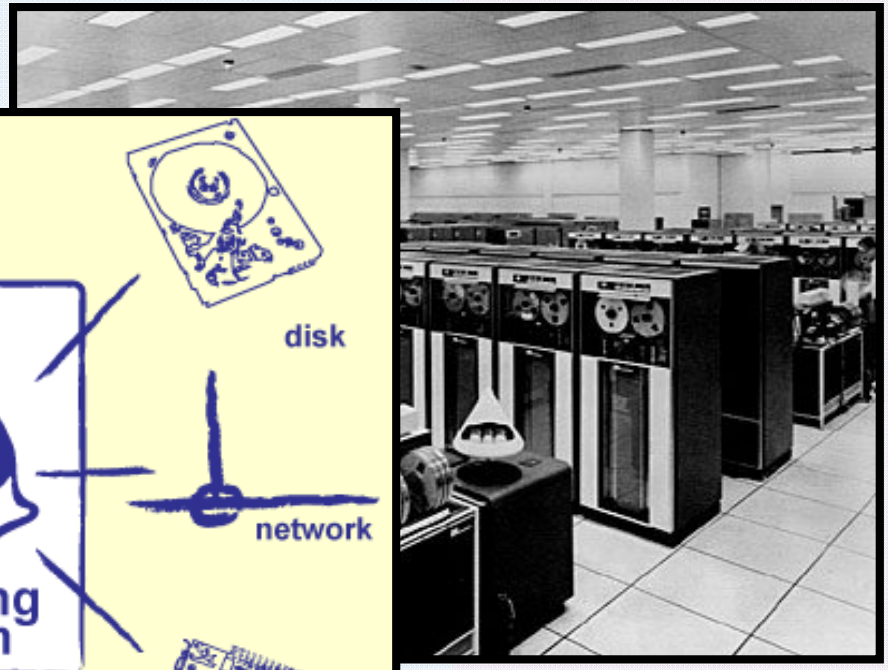
OS ? What is it ?

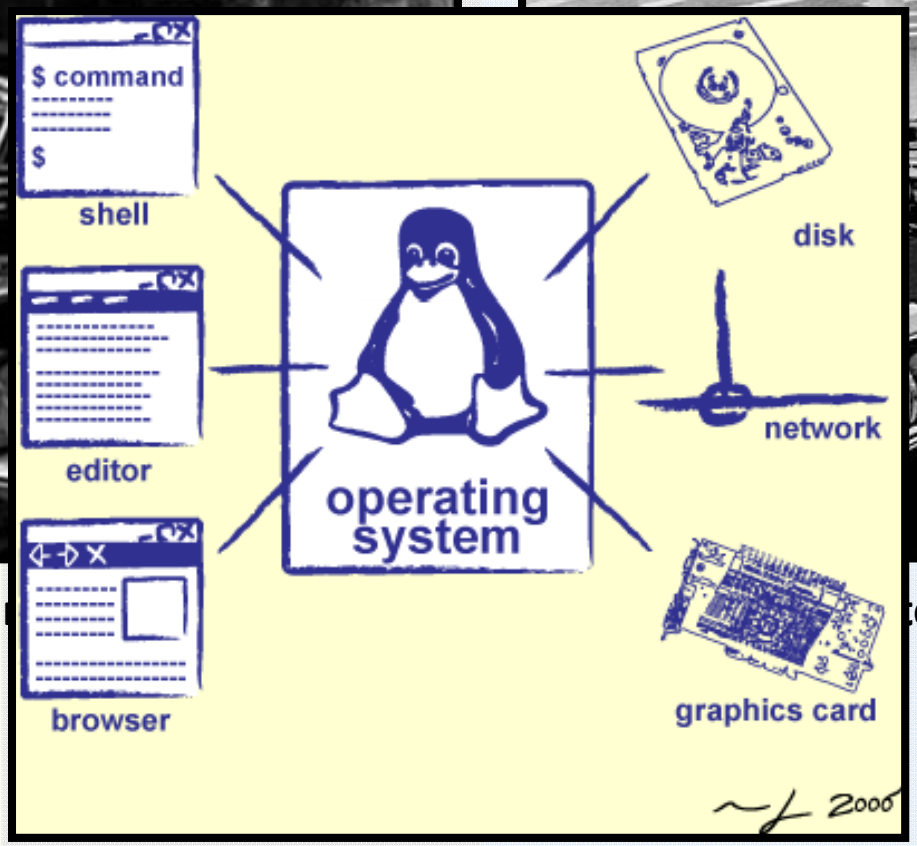
Switchboard Operator

Computer Operator

OS ? What is it ?







Switchboa

er Operator

Software Operator

OS ? What is it ?

- OK, well what is a system then

OS ? What is it ?

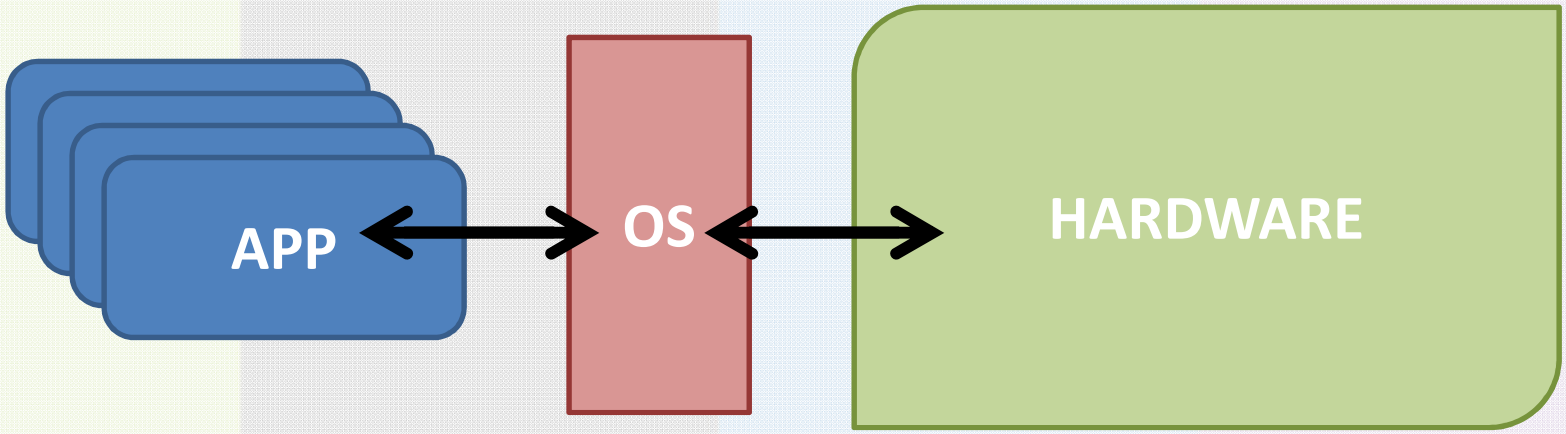
- OK, well what is a system then?

- A set of things working together as parts of a mechanism or a interconnecting network

OS ? What is it ?

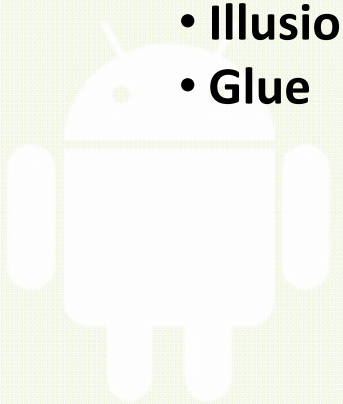
- Special layer of software that provides application software access to hardware resources

  - Convenient abstraction of complex hardware devices
  - Protected access to shared resources
  - Security and authentication
  - Communication amongst logical entities



ROLES of OS's

- There are 3 MAIN roles of any OS
  - Referee
  - Illusionist
  - Glue



ROLES of OS's

- Referee
  - Manage sharing of resources between different applications running on the same machine
    - Stopping and Starting of programs
    - Isolation of applications from each other
    - Deciding which applications get resources when
    - Protect itself and other programs from malicious software

ROLES of OS's

- Illusionist
  - Provide clean, easy to use abstractions of physical resources
    - Provides the illusion that a program has infinite memory
    - Provides the illusion that a program has the computers CPU all to itself
    - Allows programs to be written independently of the resources available
    - Since applications are written at a higher level of abstraction, the operating system can invisibly change the amount of resources allocated to each application

ROLES of OS's

- Glue
  - Provide a set of common services that facilitate sharing among applications
    - Cut/Paste works uniformly across the system
    - A file written by one application can be read by another
    - Common user interface routines which give applications a similar look and feel
    - Provides a layer separating applications from I/O devices

ROLES of OS's

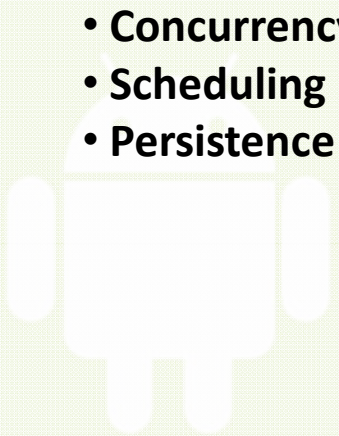
- **Referee**
  - **Manage sharing of resources, Protection**
    - Resource allocation, isolation, communication

- **Illusionist**
  - **Provide clean, easy to use abstractions of physical resources**
    - Infinite memory, dedicated machine
    - Higher level objects: files, users, messages
    - Masking limitations, virtualization

- **Glue**
  - **Common services**
    - Storage, Window system, Networking
    - Sharing, Authorization
    - Look and feel

So how does the OS perform these roles?

- Abstraction
- Concurrency
- Scheduling
- Persistence



Abstraction

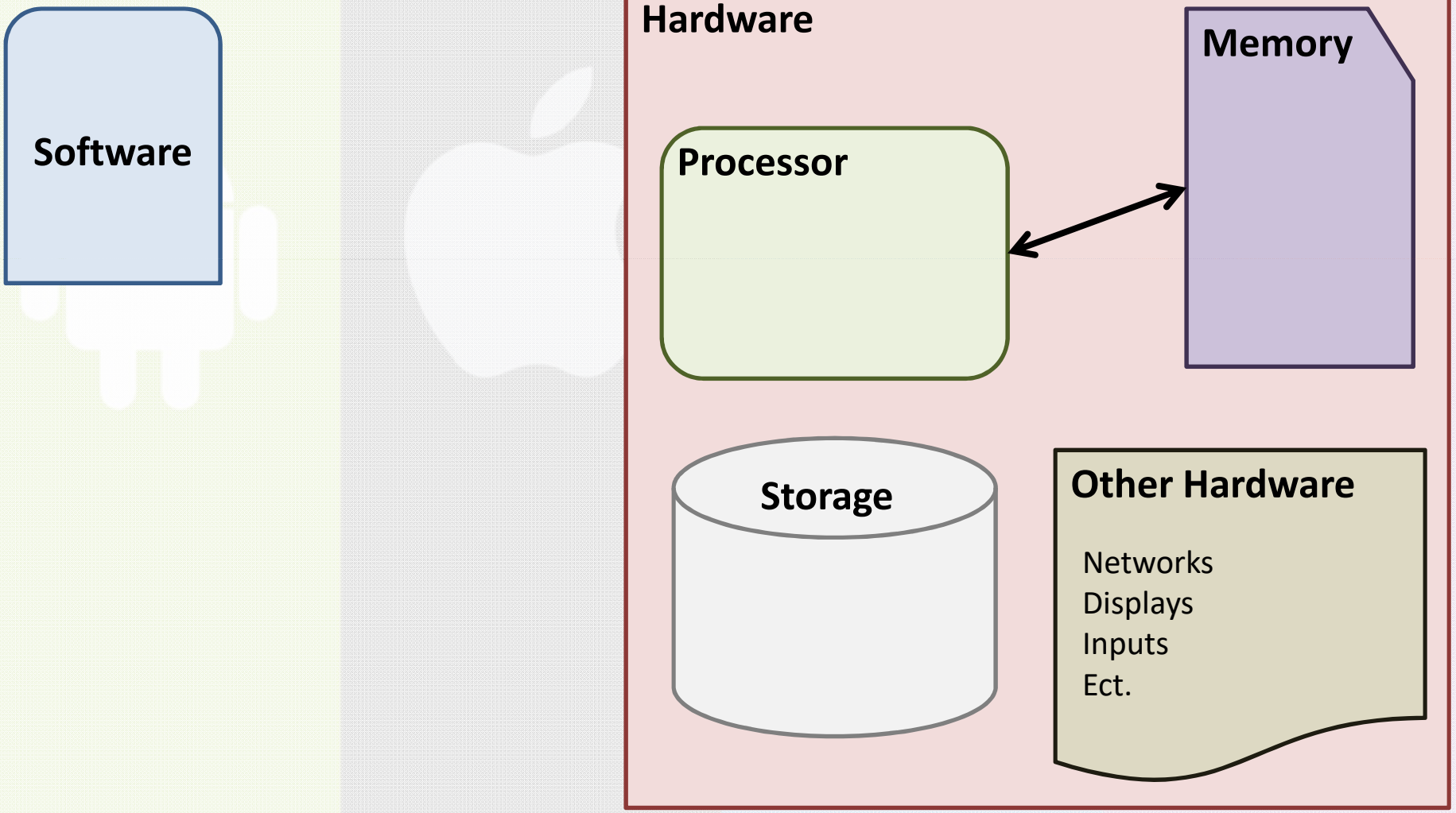
• Abstraction in general is a logical view of something. You don’t look at the (technical) details, but only at the principle.

Abstraction

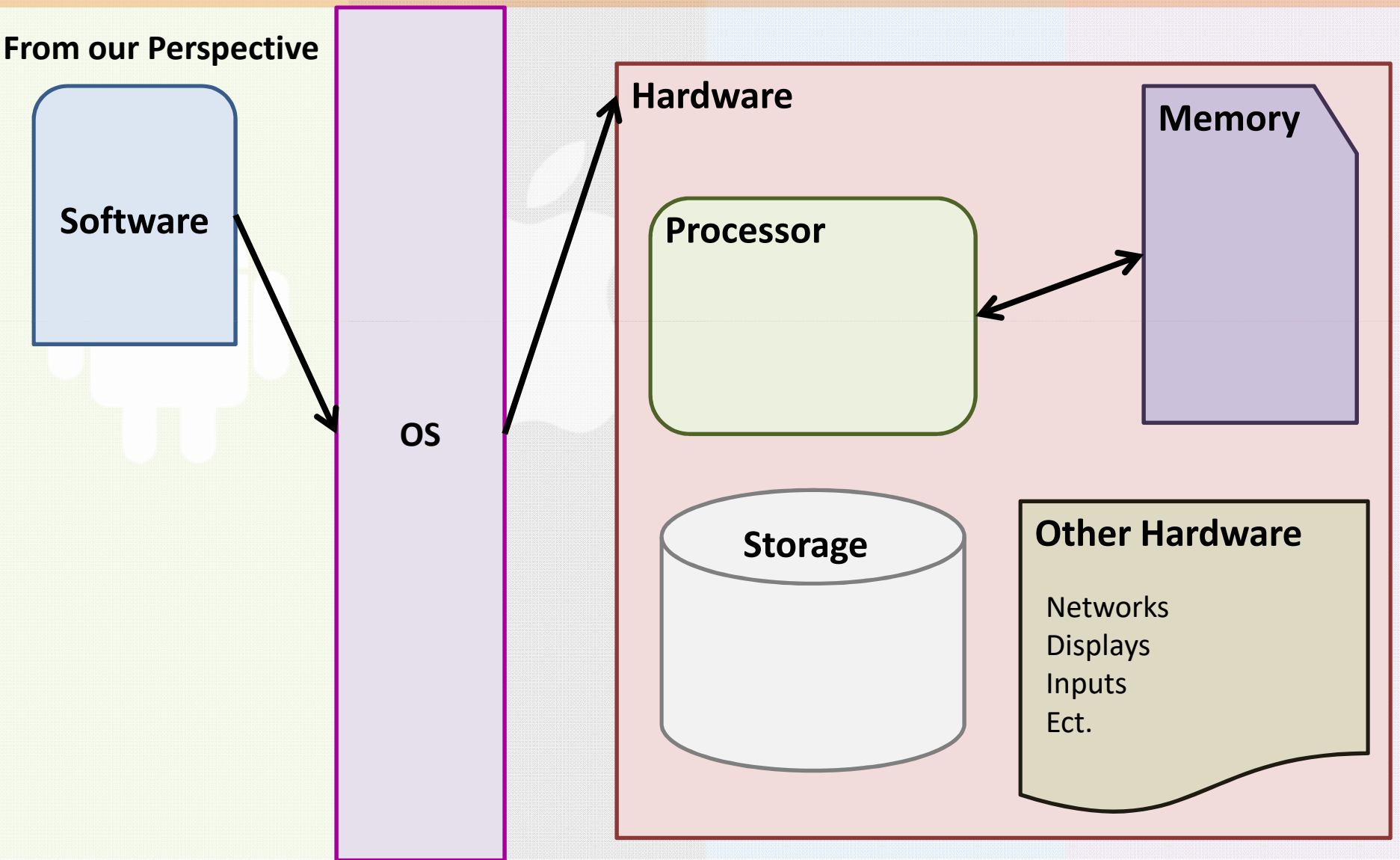
• Abstraction in general is a logical view of something. You don't look at the (technical) details, but only at the principle.

• OS's provide an abstract view of the computer HW. It doesn't matter whether your sound card is from one manufacturer or another; the OS provides a unified way for applications to use it, they don't need to know how does it work *internally*.

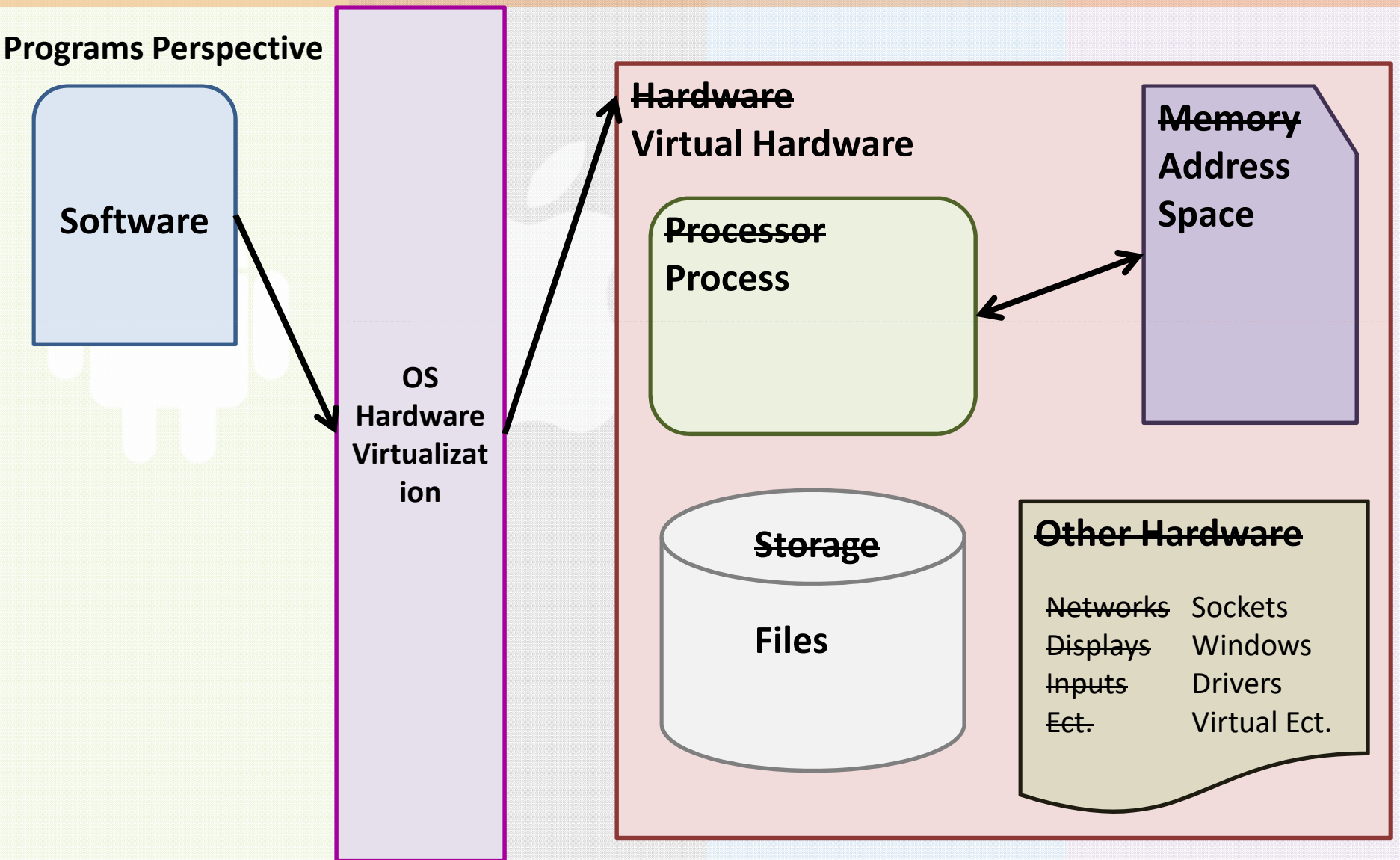
Abstraction



Abstraction



Abstraction



Concurrency

• Concurrency is the tendency for things to happen at the same time in a system. Concurrency is a natural phenomenon, of course. In the real world, at any given time, many things are happening simultaneously.

Concurrency

• Concurrency is the tendency for things to happen at the same time in a system. Concurrency is a natural phenomenon, of course. In the real world, at any given time, many things are happening simultaneously.

• When dealing with concurrency issues in software systems, there are generally two aspects that are important:

- being able to detect and respond to external events occurring in a random order,
- ensuring that these events are responded to in some minimum required interval.

Concurrency

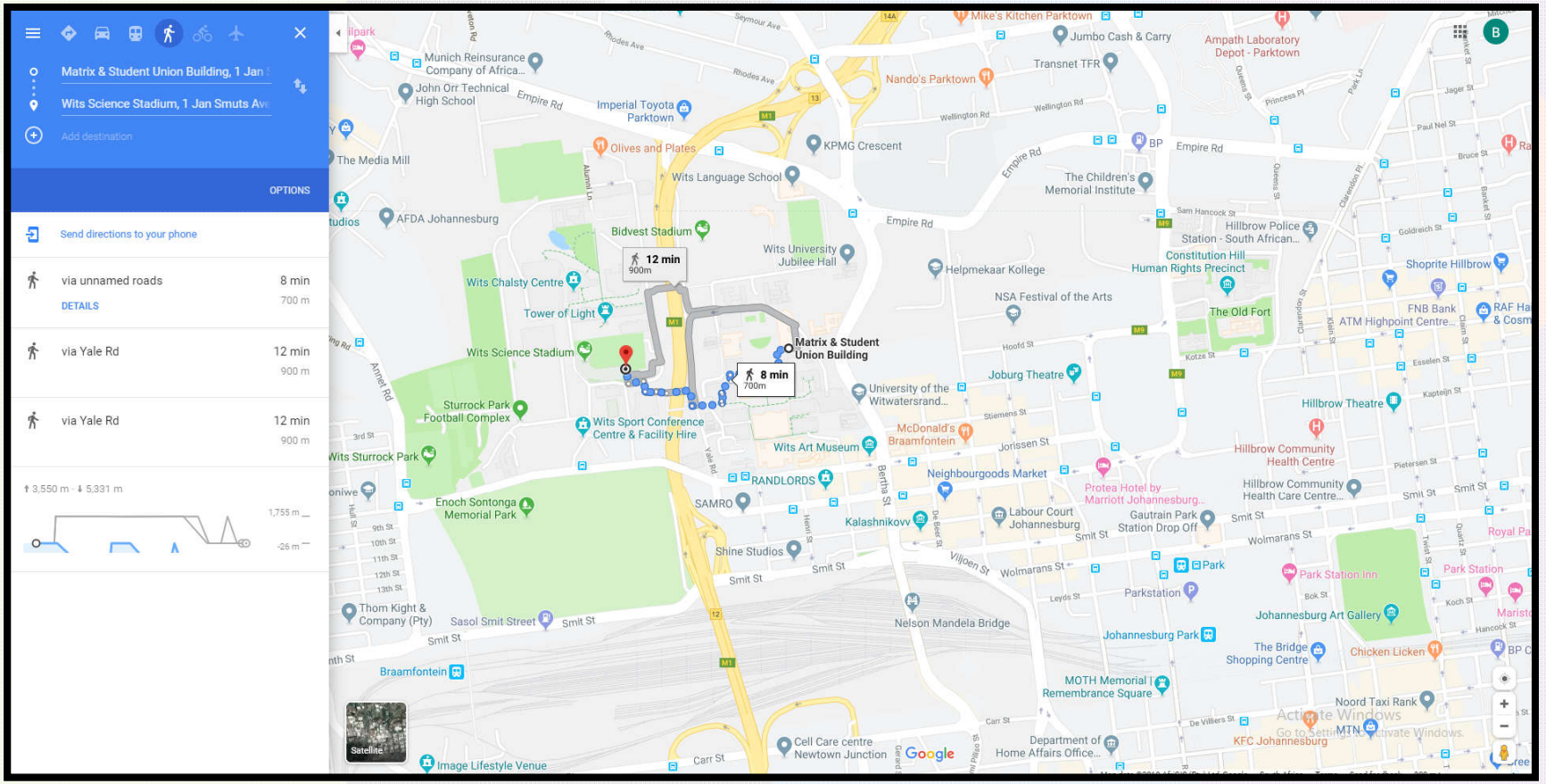
• Concurrency is the tendency for things to happen at the same time in a system. Concurrency is a natural phenomenon, of course. In the real world, at any given time, many things are happening simultaneously.

• When dealing with concurrency issues in software systems, there are generally two aspects that are important:
    • being able to detect and respond to external events occurring in a random order,
    • ensuring that these events are responded to in some minimum required interval.

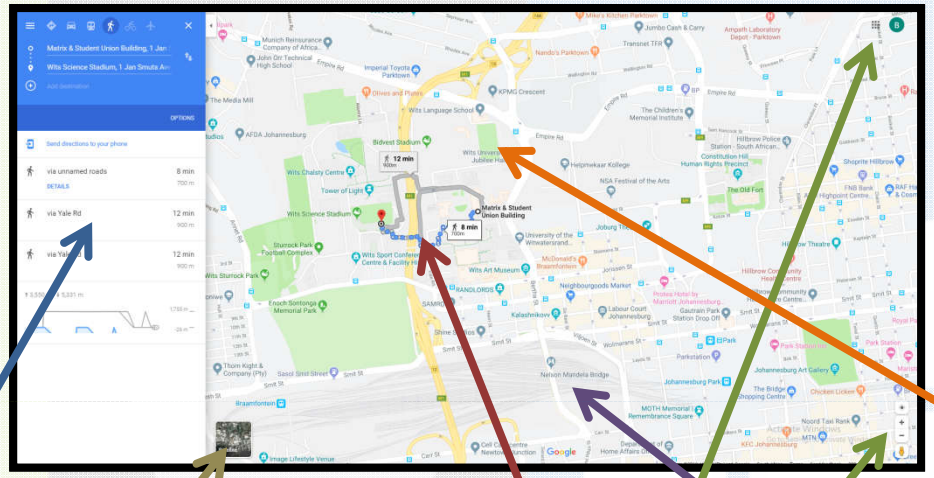
• If each concurrent activity evolved independently, in a truly parallel fashion, this would be relatively simple: we could simply create separate programs to deal with each activity.
• The challenges of designing concurrent systems arise mostly because of the interactions which happen between concurrent activities. When concurrent activities interact, some sort of coordination is required.

Concurrency

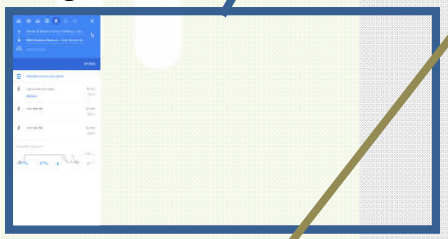
- Google Maps



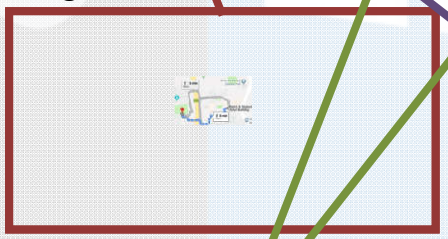
Concurrency



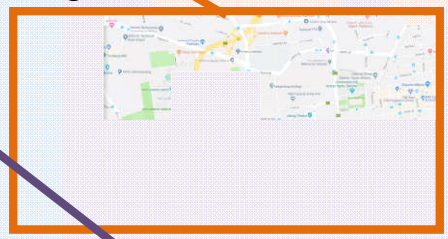
Programs 1



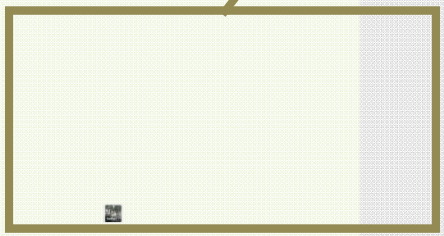
Programs 2



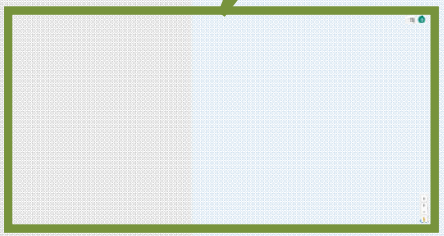
Programs 3



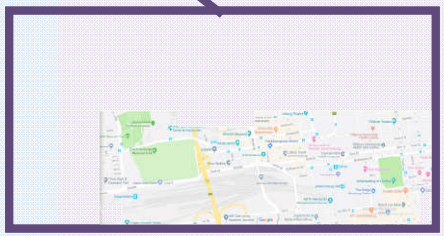
Programs 4



Programs 5



Programs 6



Scheduling

- In computing, scheduling is the method by which work is assigned to resources that complete the work.
- The work may be virtual computation elements such as threads, processes or data flows, which are in turn scheduled onto hardware resources such as processors, network links or expansion cards.

Scheduling

• In computing, scheduling is the method by which work is assigned to resources that complete the work.

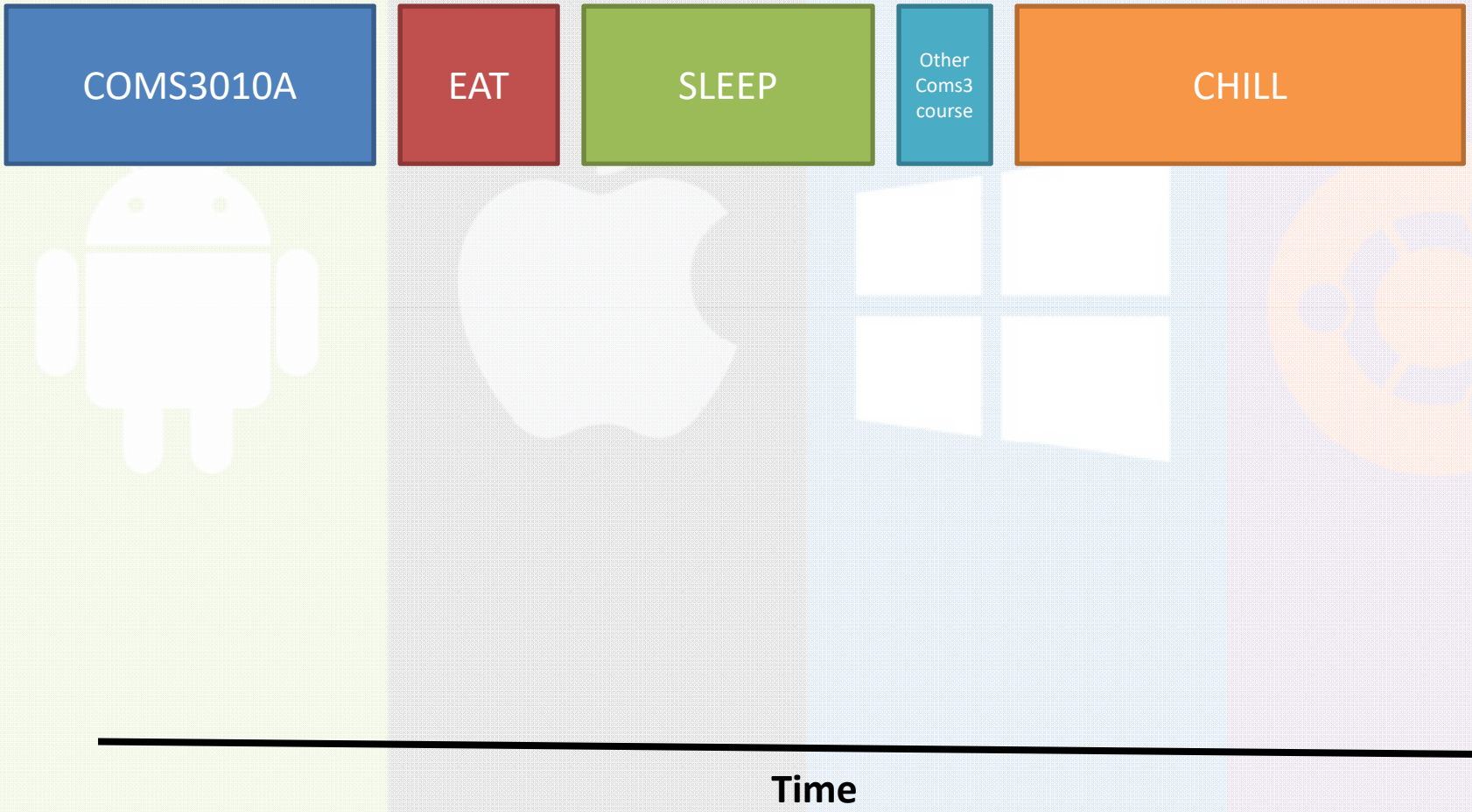
• The work may be virtual computation elements such as threads, processes or data flows, which are in turn scheduled onto hardware resources such as processors, network links or expansion cards.

Scheduling

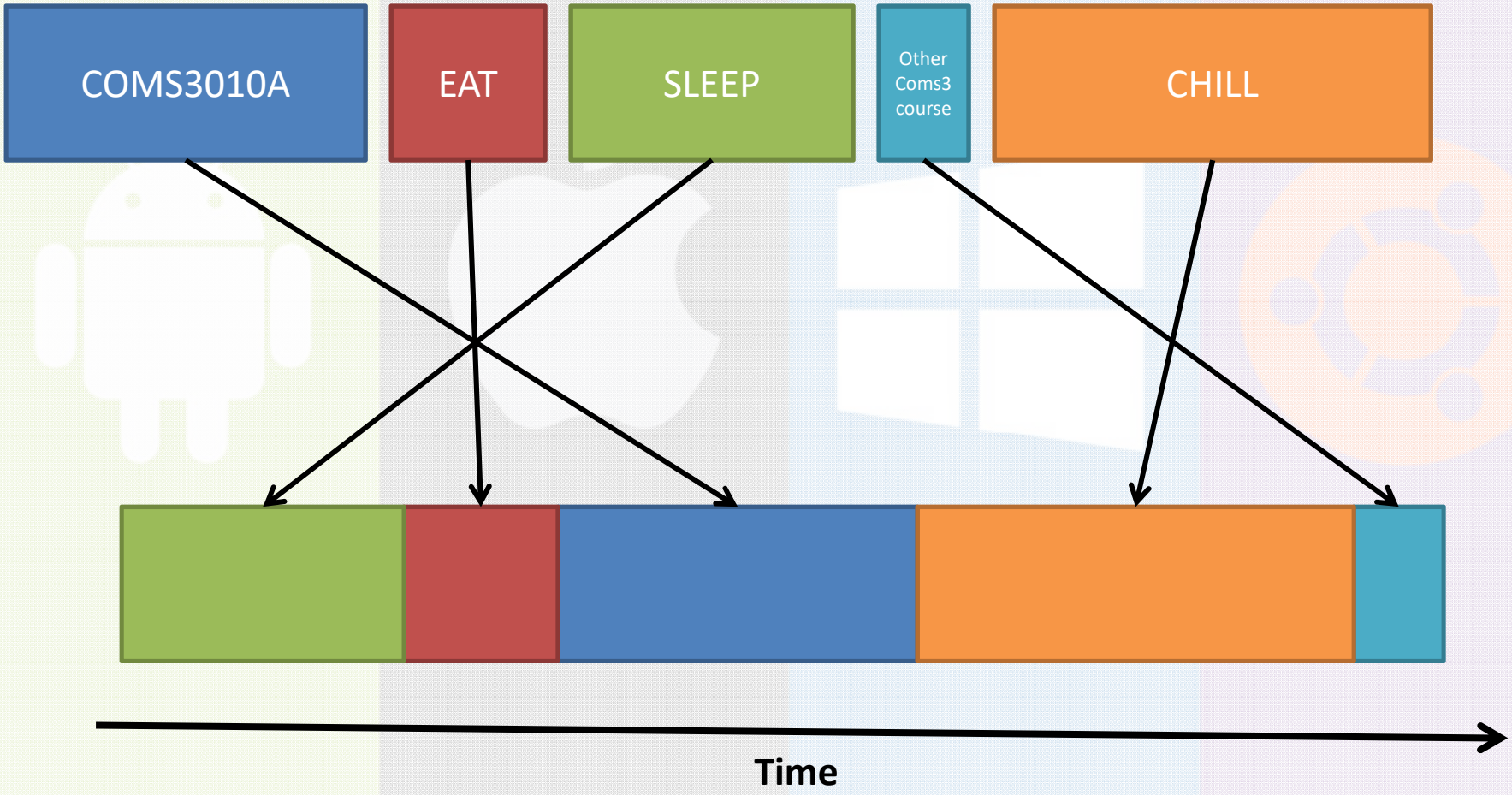
• In computing, scheduling is the method by which work is assigned to resources that complete the work.
• The work may be virtual computation elements such as threads, processes or data flows, which are in turn scheduled onto hardware resources such as processors, network links or expansion cards.

• A scheduler is what carries out the scheduling activity.
• Schedulers are often implemented so they keep all computer resources busy, allow multiple users to share system resources effectively, or to achieve a target quality of service.

Scheduling



Scheduling



Persistence

- In computer science, persistence refers to the characteristic of state that outlives the process that created it.

Persistence

• In computer science, persistence refers to the characteristic of state that outlives the process that created it.

• This is achieved in practice by storing the state as data in computer data storage. Programs have to transfer data to and from storage devices and have to provide mappings from the native programming-language data structures to the storage device data structures.

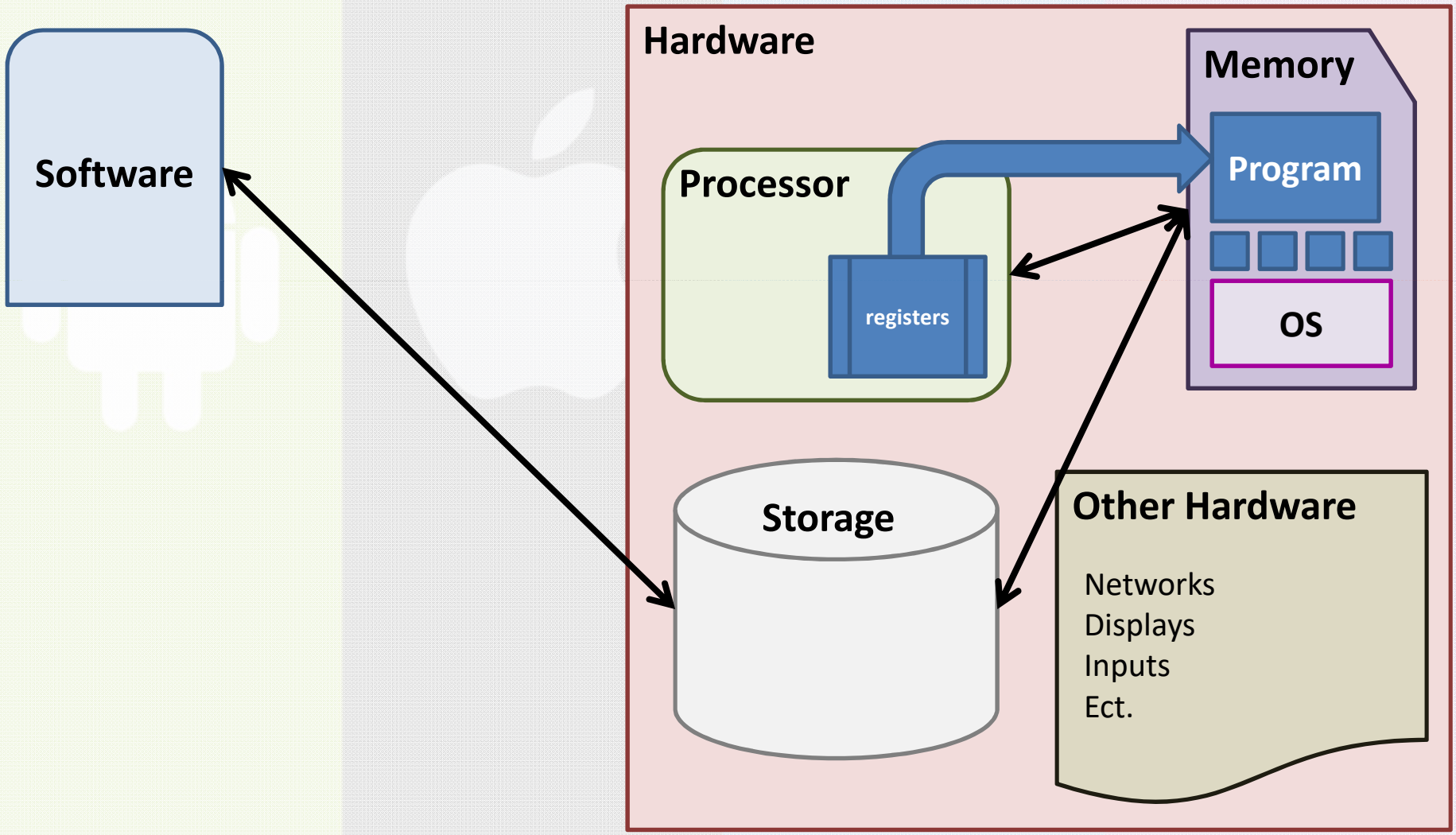
Persistence

• In computer science, persistence refers to the characteristic of state that outlives the process that created it.

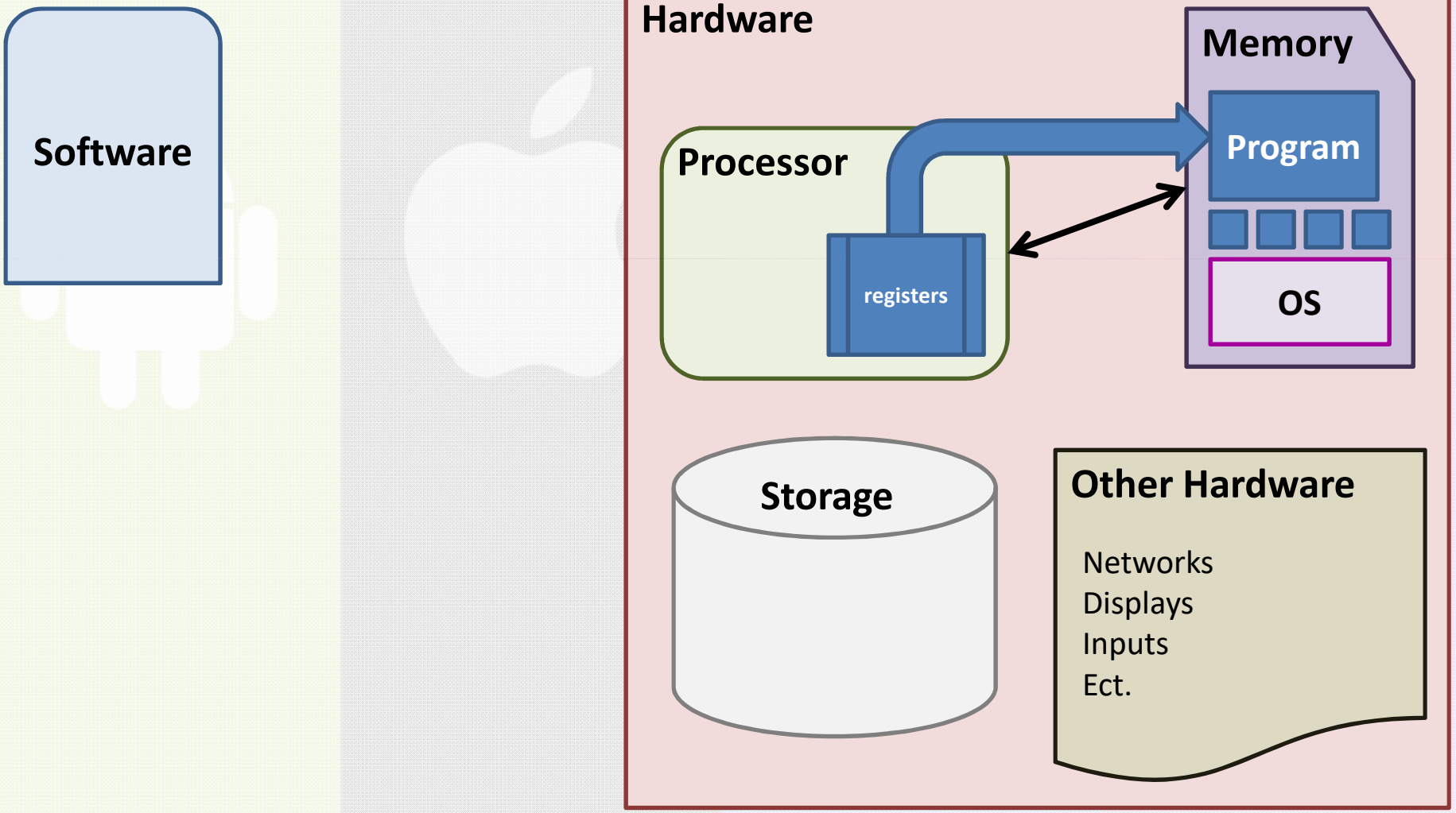
• This is achieved in practice by storing the state as data in computer data storage. Programs have to transfer data to and from storage devices and have to provide mappings from the native programming-language data structures to the storage device data structures.

• Picture editing programs or word processors, for example, achieve state persistence by saving their documents to files.

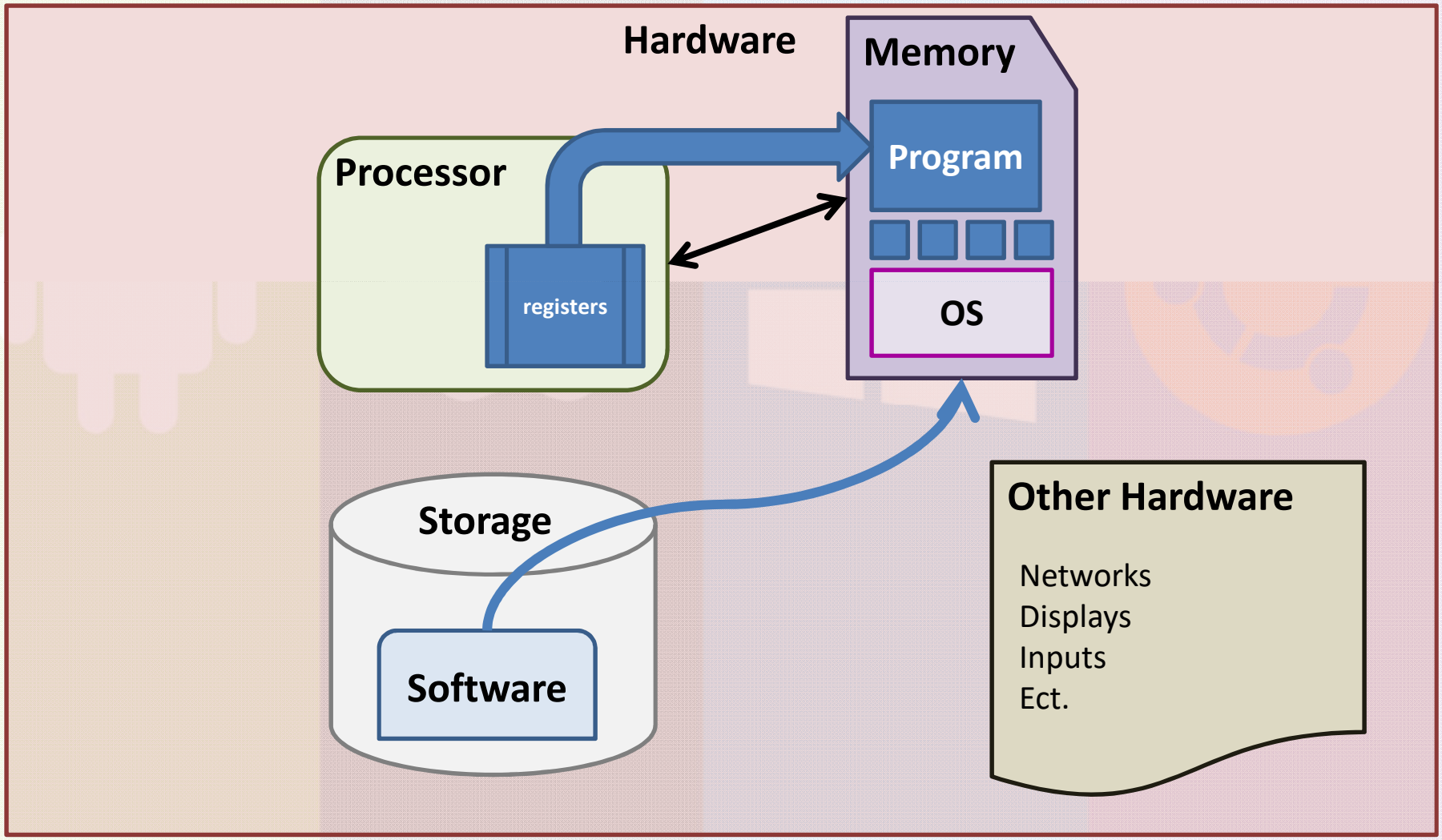
Persistence



OS Basics : Loading



OS Basics : Loading



Questions?

What happens when multiple applications share resources?

- Sharing is central to the functionality of most computers
- The OS must somehow keep all of the activities separate and yet allow all access to the full capacity of the machine

What happens when multiple applications share resources?

- Sharing is central to the functionality of most computers
- The OS must somehow keep all of the activities separate and yet allow all access to the full capacity of the machine

- Job of the referee
  - Resource Allocation
    - The OS must allocate resources to each application when required
  - Isolation
    - An error in one application should not disrupt other applications or the OS itself
  - Communication
    - The OS must setup communication boundaries but also allow them to be crossed when appropriate

What happens when applications need more resources than the hardware provides?

- A computer only has a finite amount of resources
  - Limited processors, physical memory, network bandwidth, disk space
- The OS must mask the restrictions inherent in computer hardware

What happens when applications need more resources than the hardware provides?

- A computer only has a finite amount of resources
  - Limited processors, physical memory, network bandwidth, disk space
- The OS must mask the restrictions inherent in computer hardware

- Job of the illusionist
  - Virtualization
    - Provides the application with the illusion of infinite resources

- Taking Virtualization further we can virtualize the entire computer
  - Running an OS as an application on top of another OS

What happens when applications need more resources than the hardware provides?

- A computer only has a finite amount of resources
  - Limited processors, physical memory, network bandwidth, disk space
- The OS must mask the restrictions inherent in computer hardware

- Job of the illusionist
  - Virtualization
    - Provides the application with the illusion of infinite resources

- Taking Virtualization further we can virtualize the entire computer
  - Running an OS as an application on top of another OS

- This is called a Virtual Machine (VM)
  - The Guest OS thinks it is running on physical devices but this is another illusion generated by the OS underneath

What is the purpose of a virtual machine?

- One benefit is Portability
  - Consider a program that only works on a specific system, we could still use that program on a VM running that system
- Debugging
  - If an OS can be run as an application, developers can set breakpoints and step through the OS as if it were a regular application

Can we raise the level of abstraction above bare hardware?

- YES

Can we raise the level of abstraction above bare hardware?

- **YES**
- **By providing a set of common, standard services to applications, allows developers to simplify and standardize their design**
- **By providing common services we can also allow for communication between applications**

Can we raise the level of abstraction above bare hardware?

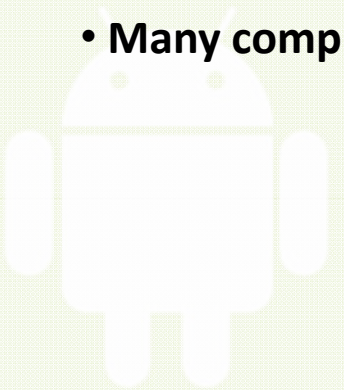
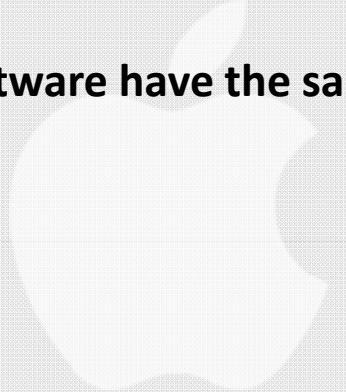
- YES
- By providing a set of common, standard services to applications, allows developers to simplify and standardize their design
- By providing common services we can also allow for communication between applications
- Job of the Glue

Are the roles of the referee, illusionist and the glue unique to operating systems?

- NO

Are the roles of the referee, illusionist and the glue unique to operating systems?

- NO
- Many complex software have the same challenges an OS has

Are the roles of the referee, illusionist and the glue unique to operating systems?

- NO
- Many complex software have the same challenges an OS has
- Cloud Computing
  - Referee – How are resources allocated between competing applications running in the cloud
  - Illusionist – Cloud resources are always changing, what abstractions isolate developers from these hardware changes
  - Glue – Clouds often distribute work across different machines, what abstractions should the cloud provide to coordinate and share data between these machines

Are the roles of the referee, illusionist and the glue unique to operating systems?

- NO
- Many complex software have the same challenges an OS has
- Multiplayer Games
  - Referee – Many games offload work onto client machines to reduce server load, How do game designers set limits for extensions to prevent unfair play
  - Illusionist – If objects in the game are spread across client and server, is that distinction evident to extensions or is it abstracted
  - Glue – How should a developer design their API's to make it easier to foster a community of developers

Are the roles of the referee, illusionist and the glue unique to operating systems?

- NO
- Many complex software have the same challenges an OS has
- Database System
  - Referee –
  - Illusionist –
  - Glue –

Are the roles of the referee, illusionist and the glue unique to operating systems?

- NO
- Many complex software have the same challenges an OS has
- Database System
  - Referee – How should resources be allocated to various users of the database
  - Illusionist –
  - Glue –

Are the roles of the referee, illusionist and the glue unique to operating systems?

- NO
- Many complex software have the same challenges an OS has
- Database System
  - Referee – How should resources be allocated to various users of the database
  - Illusionist – How does the database mask machine failures to ensure data integrity
  - Glue –

Are the roles of the referee, illusionist and the glue unique to operating systems?

- NO
- Many complex software have the same challenges an OS has
- Database System
  - Referee – How should resources be allocated to various users of the database
  - Illusionist – How does the database mask machine failures to ensure data integrity
  - Glue – What common services make it easier to develop database applications

What design goals should we look for in an Operating System?

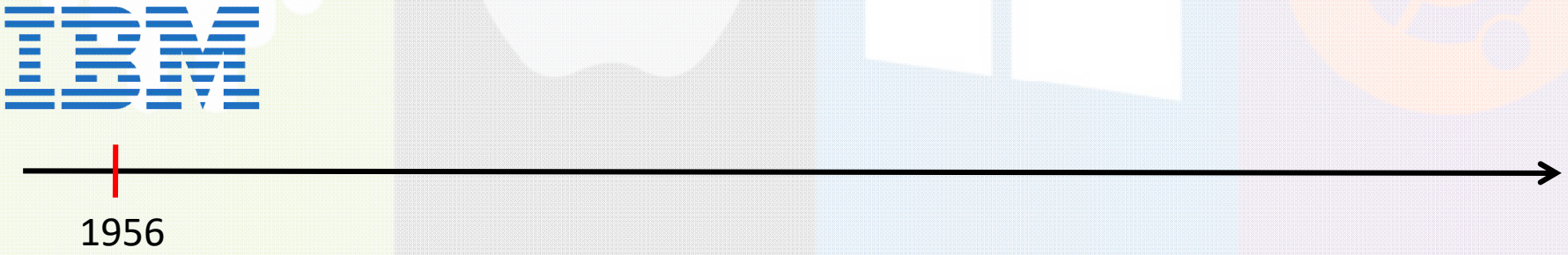
- Reliability and Availability
- Security
- Portability
- Performance
- Adoption

History

- In the Beginning

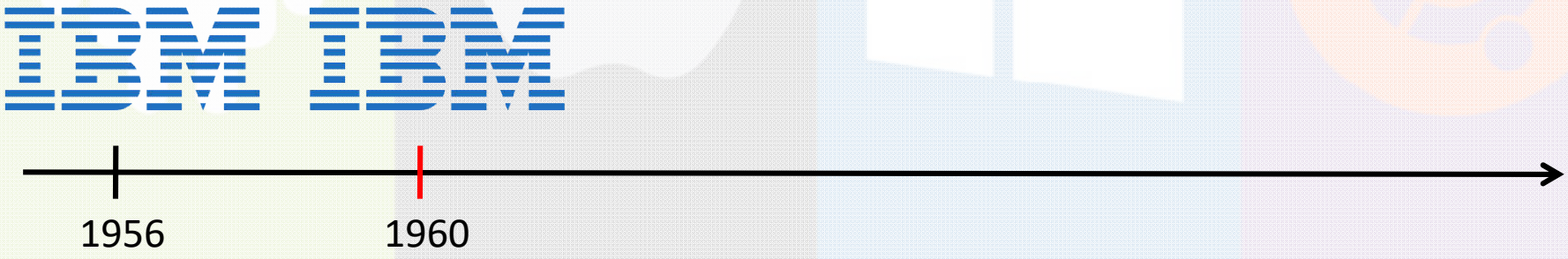
History

- **The first operating system was created by General Motors in 1956 to run a single IBM mainframe computer.**



History

- **In the 1960s, IBM was the first computer manufacturer to take on the task of operating system development and began distributing operating systems with their computers**

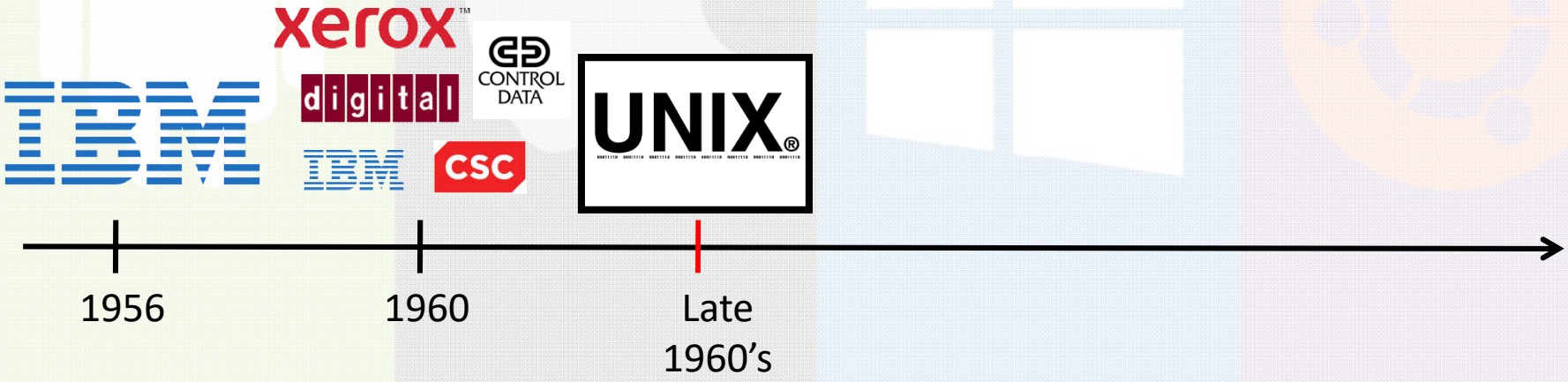


History

- However, IBM wasn't the only vendor creating operating systems during this time. Control Data Corporation, Computer Sciences Corporation, Burroughs Corporation, GE, Digital Equipment Corporation, and Xerox all released mainframe operating systems in the 1960s as well

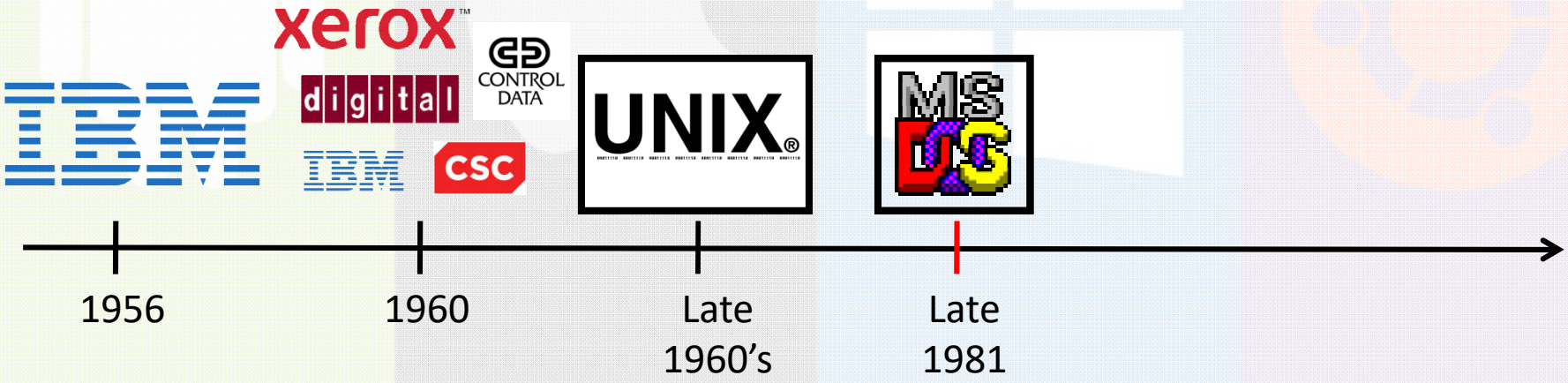
History

- In the late 1960s, the first version of the Unix operating system was developed. Written in C, and freely available during it's earliest years, Unix was easily ported to new systems and rapidly achieved broad acceptance.



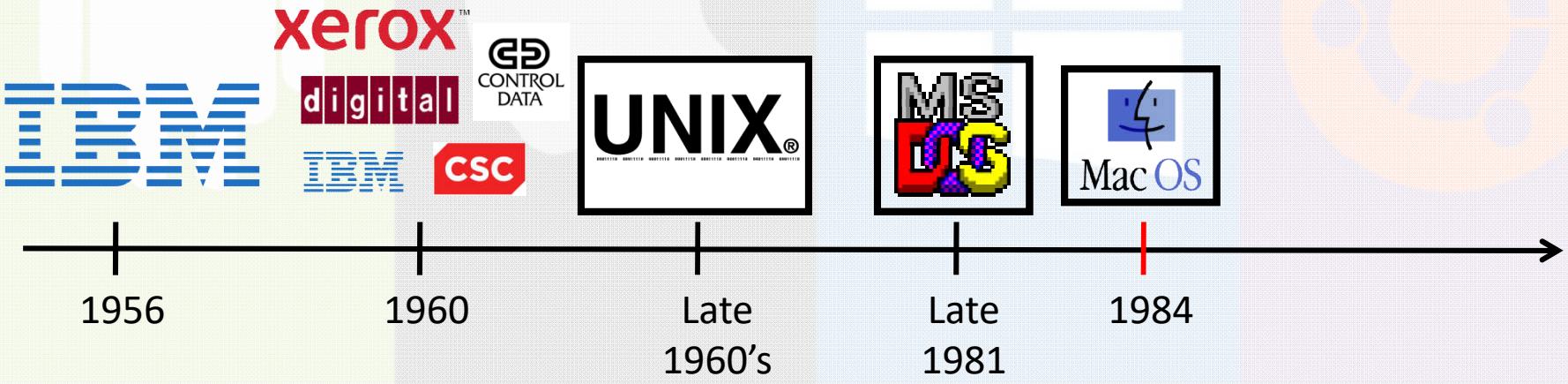
History

- **The first OS built by Microsoft wasn't called Windows, it was called MS-DOS and was built in 1981 by purchasing the 86-DOS operating system from Seattle Computer Products and modifying it to meet IBM's requirements.**



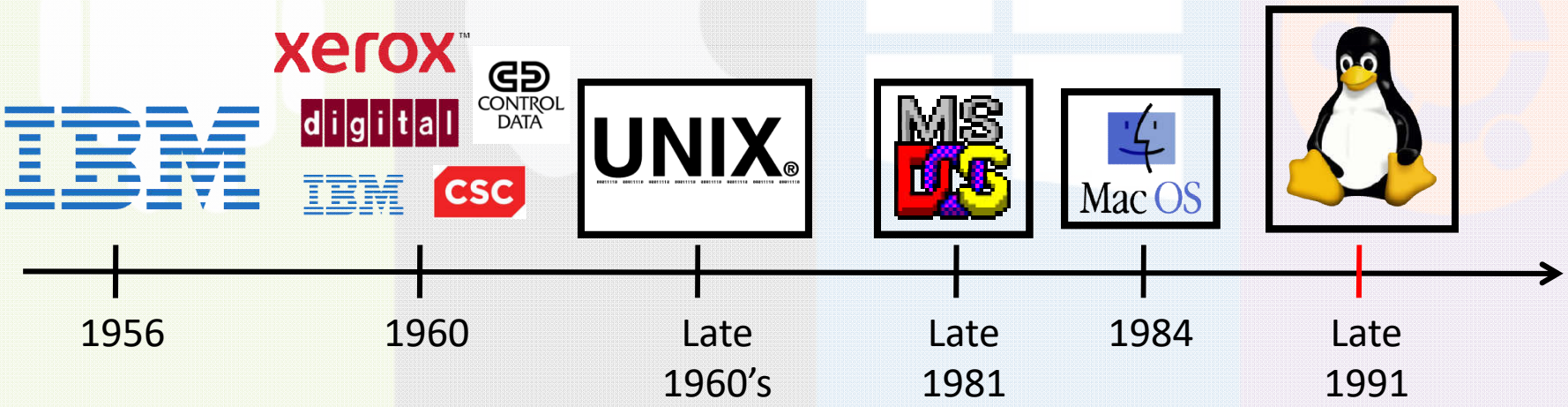
History

- **Apple released the original Macintosh on January 24, 1984. This was the first version of the system software which would later go on to be called MacOS**

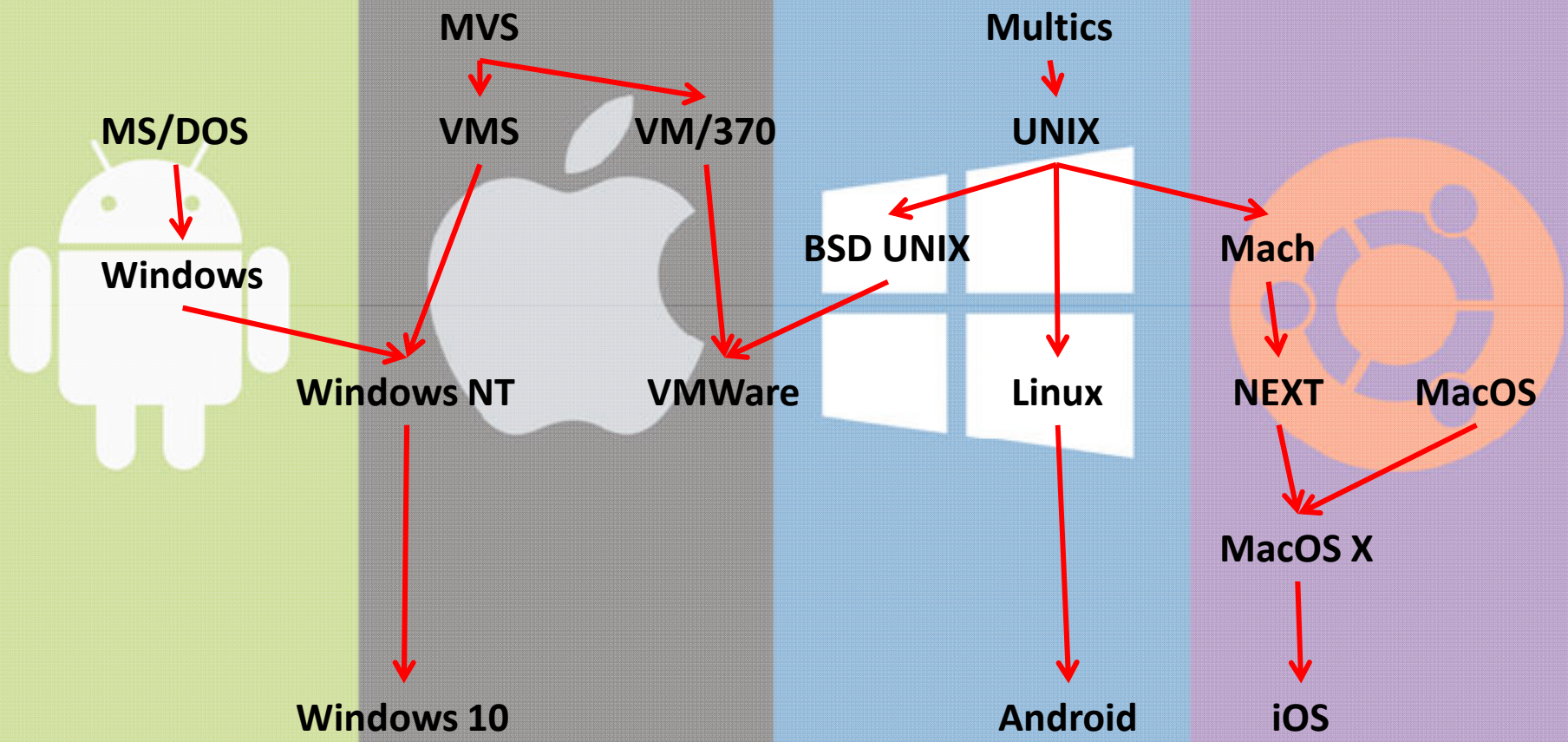


History

- **Linux was first released on September 17, 1991, by Linus Torvalds.**



Where did we come from? Where did we go?



Genealogy of Operating Systems