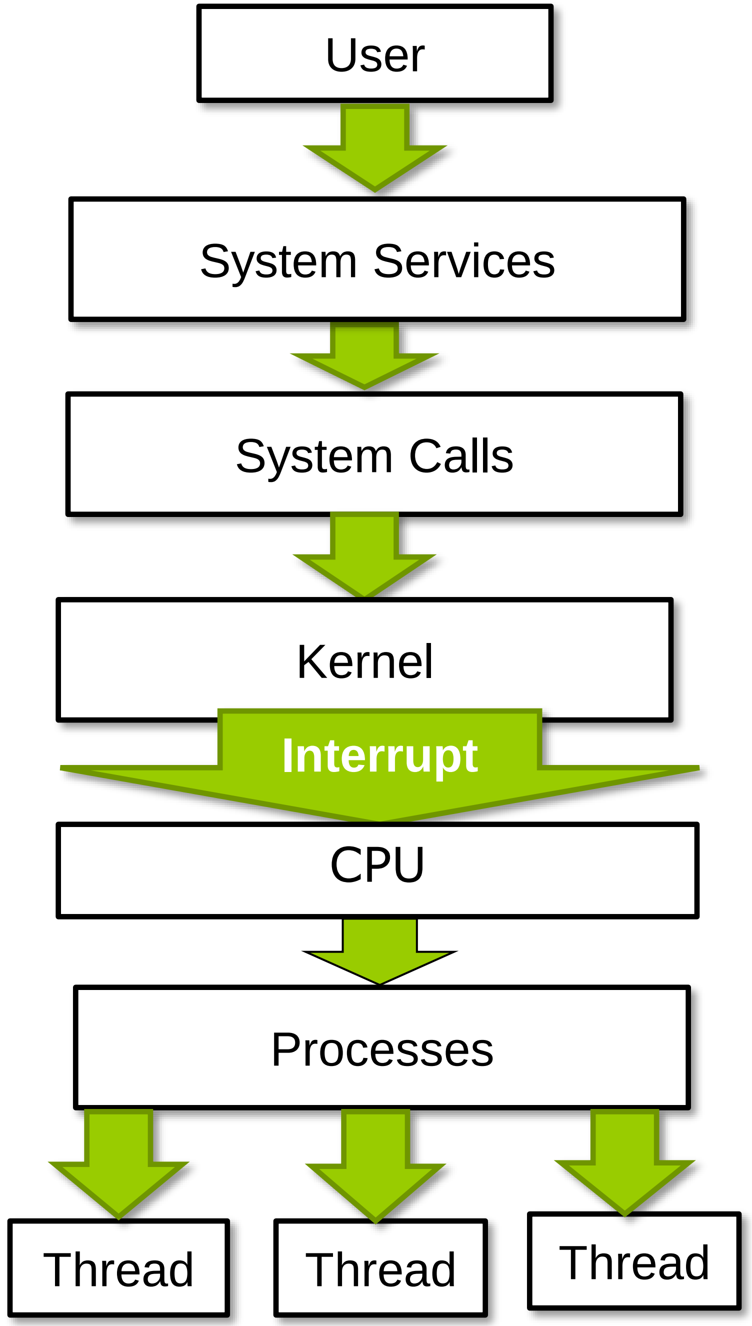
Part 5

The Big Pic of OS



System Services

- System programs provide a convenient environment for program **development and execution.** They can be divided into:
  - File manipulation
  - Status information sometimes stored in a file
  - Programming language support
  - Program loading and execution
  - Communications
  - Background services
  - Application programs
- **Most users' view of the operation** system is defined by system programs, not the actual system calls

System Services (cont.)

- Provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
- **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- **Status information**
  - Some ask the system for info - date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
  - Typically, these programs format and print the output to the terminal or other output devices
  - Some systems implement a **registry** - used to store and retrieve configuration information

System Services (Cont.)

- **File modification**
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided
- **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

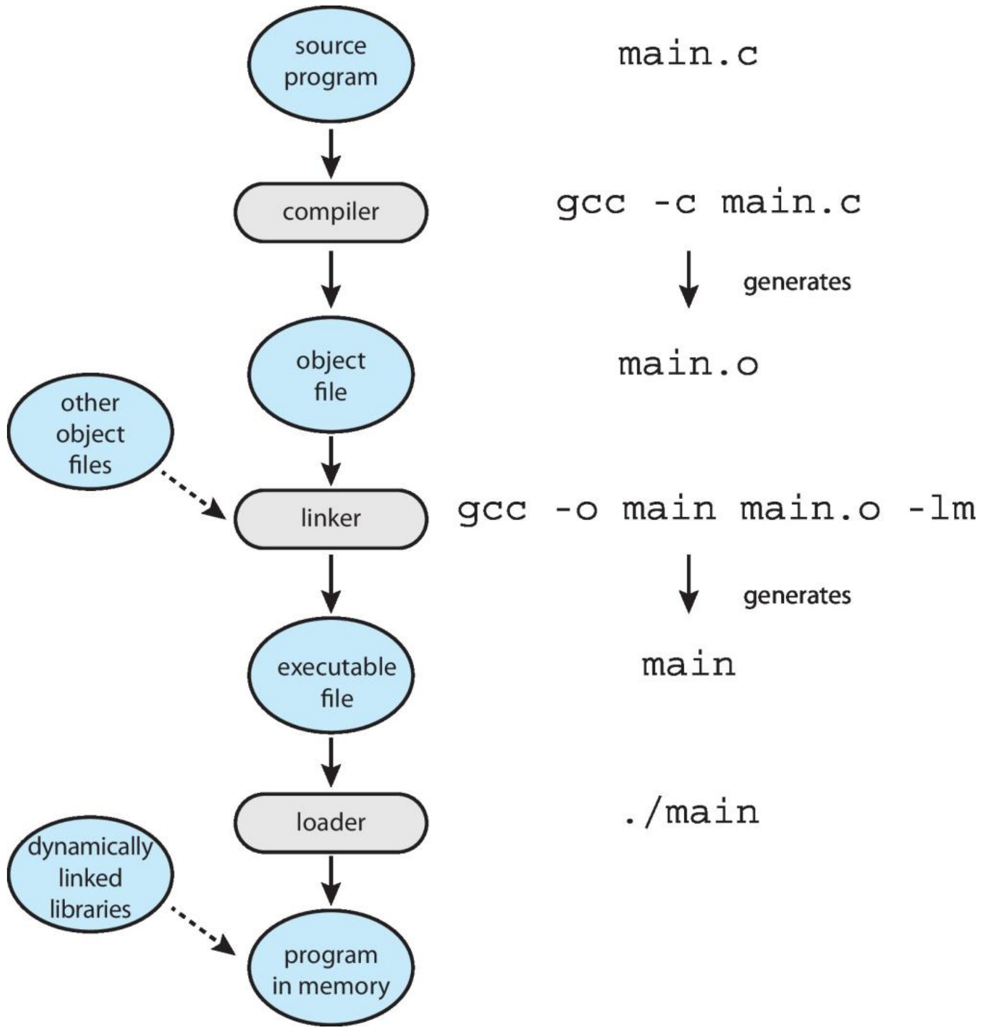
System Services (Cont.)

- **Background Services**
  - Launch at boot time
    - Some for system startup, then terminate
    - Some from system boot to shutdown
  - Provide facilities like disk checking, process scheduling, error logging, printing
  - Run in user context not kernel context
  - Known as **services**, **subsystems**, **daemons**
- **Application programs**
  - Don’t pertain to system
  - Run by users
  - Not typically considered part of OS
  - Launched by command line, mouse click, finger poke

Linkers and Loaders

- Source code compiled into object files designed to be loaded into any physical memory location – **relocatable object file**
- **Linker** combines these into single binary **executable** file
  - Also brings in libraries
- Program resides on secondary storage as binary executable
- Must be brought into memory by **loader** to be executed
  - **Relocation** assigns final addresses to program parts and adjusts code and data in program to match those addresses
- Modern general purpose systems don’t link libraries into executables
  - Rather, **dynamically linked libraries** (in Windows, **DLLs**) are loaded as needed, shared by all that use the same version of that same library (loaded once)
- Object, executable files have standard formats, so operating system knows how to load and start them

The Role of the Linker and Loader



Why Applications are Operating System Specific

- **Apps compiled on one system usually not executable on other operating systems**
- Each operating system provides its own unique system calls
  - Own file formats, etc
- **Apps can be multi-operating system**
  - Written in interpreted language like **Python, Ruby**, and **interpreter** available on multiple operating systems
  - App written in language that includes a VM containing the running app (like Java)
  - Use standard language (like C), compile separately on each operating system to run on each
- Application Binary Interface (ABI) **is architecture equivalent of API, defines how different components of binary code can interface for a given operating system on a given architecture, CPU, etc**

Operating System Design and Implementation

- Design and Implementation of OS not “solvable”, but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- **Start the design by defining goals and specifications**
- Affected by choice of hardware, type of system
- **User** goals and **System** goals
  - **User goals** – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - **System goals –** operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

Operating System Design and Implementation (Cont.)

- Important principle to separate

  Policy: ***What*** will be done?
  Mechanism: ***How*** to do it?

- **Mechanisms determine how to do something, policies decide what will be done**
- The separation of policy from mechanism is a very important principle, it allows **maximum flexibility** if policy decisions are to be changed later (example – timer)
- **Specifying and designing an OS is highly creative task of software engineering**

Implementation

- Much variation
  - Early OSes in **assembly language**
  - Then system programming languages like Algol, PL/1
  - **Now C, C++**
- Actually usually a mix of languages
  - Lowest levels in assembly
  - Main body in C
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- More high-level language easier to **port** to other hardware
  - But slower
- **Emulation** can allow an OS to run on non-native hardware

Operating System Structure

- General-purpose OS is very large program
- Various ways to structure ones
  - Simple structure – MS-DOS
  - More complex -- UNIX
  - Layered – an abstraction
  - Microkernel -Mach

MAJOR DESIGN ISSUES

- Transparency
- Flexibility
- Reliability
- Performance
- Scalability
- Naming
- Replication
- Synchronization
- Communication Model
- Security

Transparency

Multiple Computers are used but the user gets a view of only single system being used. Makes the network invisible to user/applications.

Various degrees of transparency

- Access Transparency
- Location Transparency
- Name Transparency
- Data Transparency
- Execution Transparency
- Performance Transparency

Flexibility

- Flexible operating systems are taken to be those whose designs have been motivated to some degree by the desire to allow the system to be tailored, either statically or dynamically, to the requirements of specific applications or application domains.

- The use of **object orientation** is a common feature of many flexible operating systems.

Reliability

- In general, **reliability** is the ability of a person or system to perform and maintain its functions in routine circumstances, as well as hostile or unexpected circumstances.

- Reliability is generally considered **important by end users**.

- Not all companies making operating systems have a similar standard. Even among operating systems where reliability is a priority, there is a range of quality.

- Also, an operating system may be extremely reliable at one kind of task and extremely unreliable at another.

- Current operating systems have two characteristics that make them unreliable and insecure.

  1> They are huge

  2> They have very poor fault isolation.

- The Linux kernel has more than 2.5 million lines of code; the Windows XP kernel is more than twice as large.

Performance

- The performance of computer hardware typically increases **monotonically with time.**

- Even if the same *could* be said of software, the rate at which software performance improves is usually very slow compared to that of hardware.

- In fact, many might opine that there is plenty of software whose performance has *deteriorated* consistently with time.

- Moreover, it is rather difficult to establish an objective performance metric for software as complex as an operating system: a *"faster OS"* is a *very* subjective, context dependent phrase.

- How to increase performance of an Operating System ?

- **Apple** computers **do not hibernate**. Rather, when they "sleep", enough devices (in particular, the dynamic RAM) are kept alive (at the cost of some battery life, if the computer is running on battery power).

- Consequently, upon wakeup, the user perceives instant-on behavior: a very desirable effect.

- Similarly, by default the system tries to keep network connections alive even if the machine sleeps.

- For example, if you login (via SSH, say) from one PowerBook to another, and both of them go to sleep, your login should stay alive within the constraints of the protocols.

Scalability

- **Scalability** is a desirable property of a system, a network, or a process, which indicates its ability to either handle growing amounts of work in a graceful manner, or to be readily enlarged.

- A system whose performance improves after adding hardware, proportionally to the capacity added, is said to be a **scalable system.**

Naming

- The resources in a distributed system are spread across different computers and a naming scheme has to be devised so that users can discover and refer to the resources that they need.

- An example of such a naming scheme is the URL (Uniform Resource Locator) that is used to identify WWW pages. If a meaningful and universally understood identification scheme is not used then many of these resources will be inaccessible to system users.

Replication

- Replication is one of the oldest and most important topics in the overall area of distributed systems.
- Whether one replicates data or computation, the objective is to have some group of processes that handle incoming events.
- If we replicate data, these processes are passive and operate only to maintain the stored data, reply to read requests, and apply updates.
- When we replicate computation, the usual goal is to provide fault-tolerance.

Synchronization

- Processes require coordination to achieve synchronization
- Types of synchronization:
  - Barrier synchronization

    Process must reach a common synchronization point before they can continue.
  - Condition coordination

    A process must wait for a condition that will be set asynchronously by other interacting processes to maintain some ordering of execution.
  - Mutual exclusion

    Concurrent processes must have mutual exclusion when accessing a critical shared resource.

Communication Model

- Lower level: message passing
- Higher level logical communication provides transparency.
  - Client/server model communication.
    - All system communication are seen as a pair of message exchanges between the client and server.
  - Remote Procedure Call, (RPC), communication.
    - RPC built on top of client/server model.
- Request/reply message passing as used in programming procedure-call concept.

SECURITY

- **Authentication**

  Clients , Servers and messages must be authenticated.

- **Authorization**

  Access control has to be performed across a physical network with heterogeneous components under different administrative units using different security models.

- Operating system authentication (OS authentication) is a way of using operating system login credentials to authenticate database users.

- One aspect of OS authentication can be used to authenticate database administrators.

- OS authentication is needed because there must be a way to identify administrative users even if the database is shut down.

Types of Operating System:

- Normal Operating System

  An operating system, or OS, is a software program that enables the computer hardware to communicate and operate with the computer software.

- Distributed Operating Systems

  Integration of system services presenting a transparent view of a multiple computer system with distributed resources and control. Consisting of concurrent processes accessing distributed shared or replicated resources through message passing in a network environment.

- Multiprocessor Operating Systems

  For the most part, multiprocessor operating systems are just regular operating systems. They handle system calls, do memory management, provide a file system, and manage I/O devices. Nevertheless, there are some areas in which they have unique features. These include process synchronization, resource management, and scheduling.

- Database Operating Systems

  Database systems place increased demands on an operating system to efficiently support: concept of a transactions, manage large volumes of data, concurrency control, system failure control.

- Real-time Operating Systems

  A real-time operating system is a multitasking operating system intended for real-time applications.

  Such applications include embedded systems (programmable thermostats, household appliance controllers, mobile telephones), industrial robots, spacecraft, industrial control.

Operating System Structure

- General-purpose OS is very large program
- Various ways to structure ones
  - Simple structure – MS-DOS
  - More complex -- UNIX
  - Layered – an abstraction
  - Microkernel -Mach

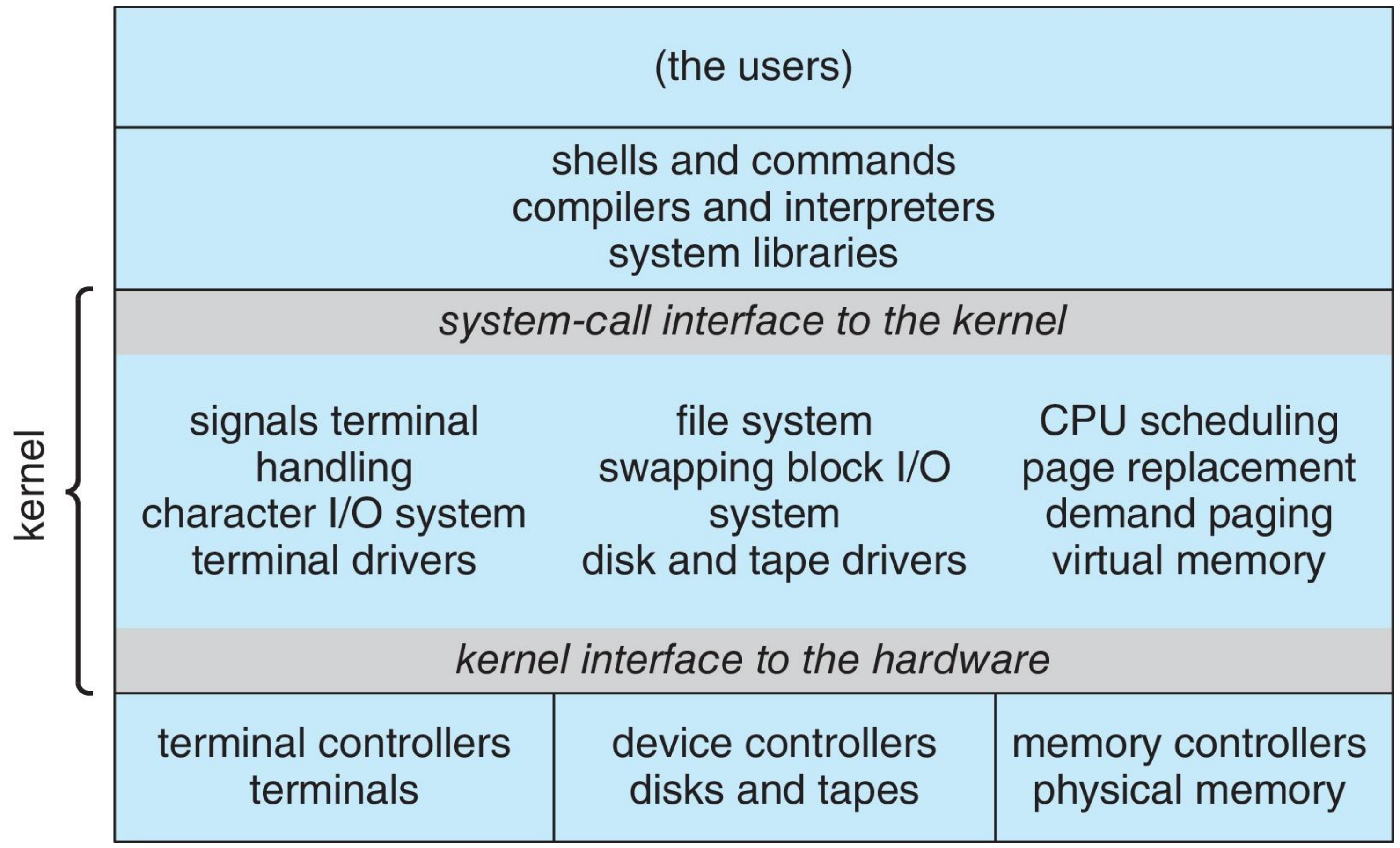
Monolithic Structure – Original UNIX

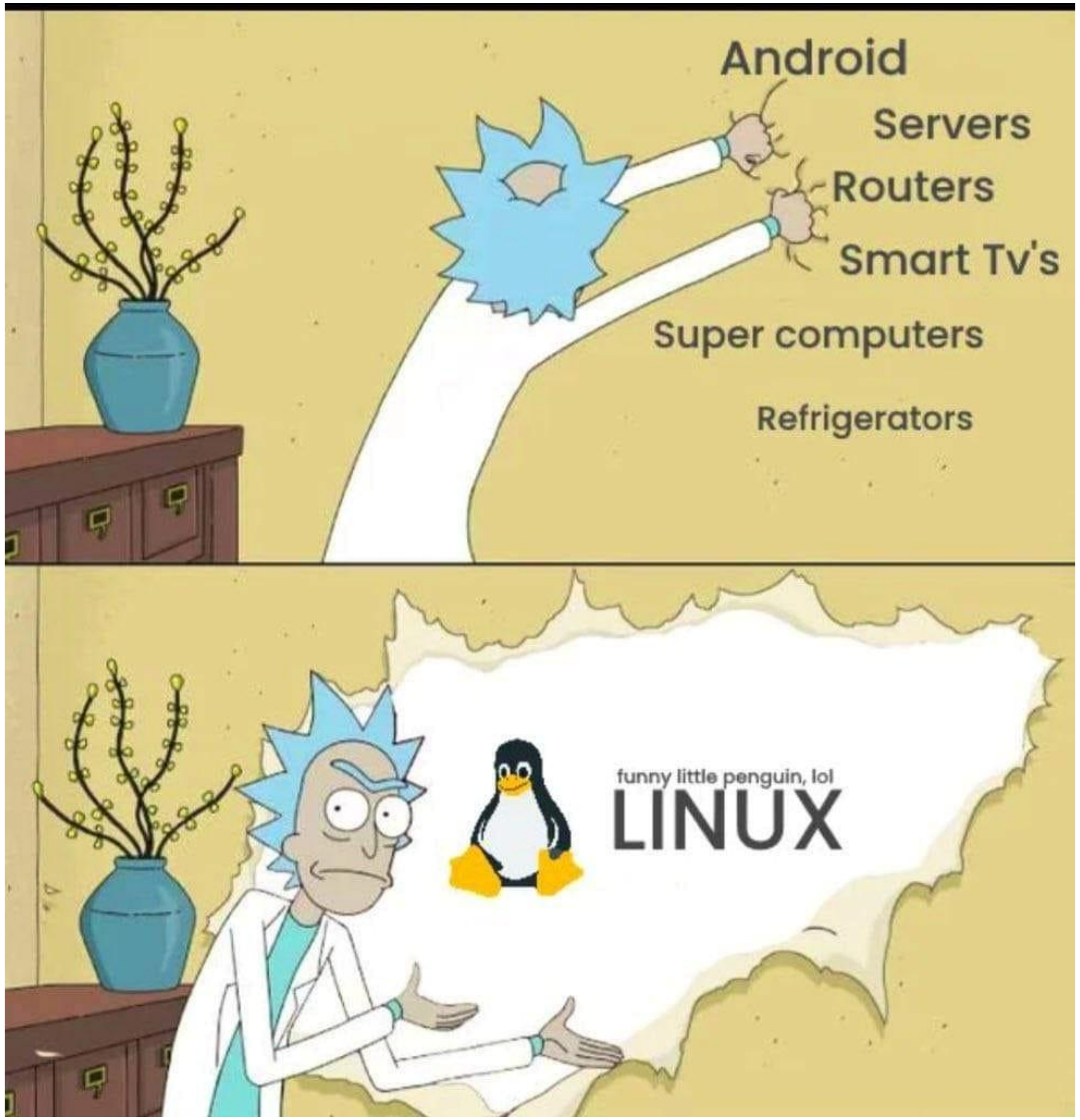
UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts

- 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

Traditional UNIX System Structure

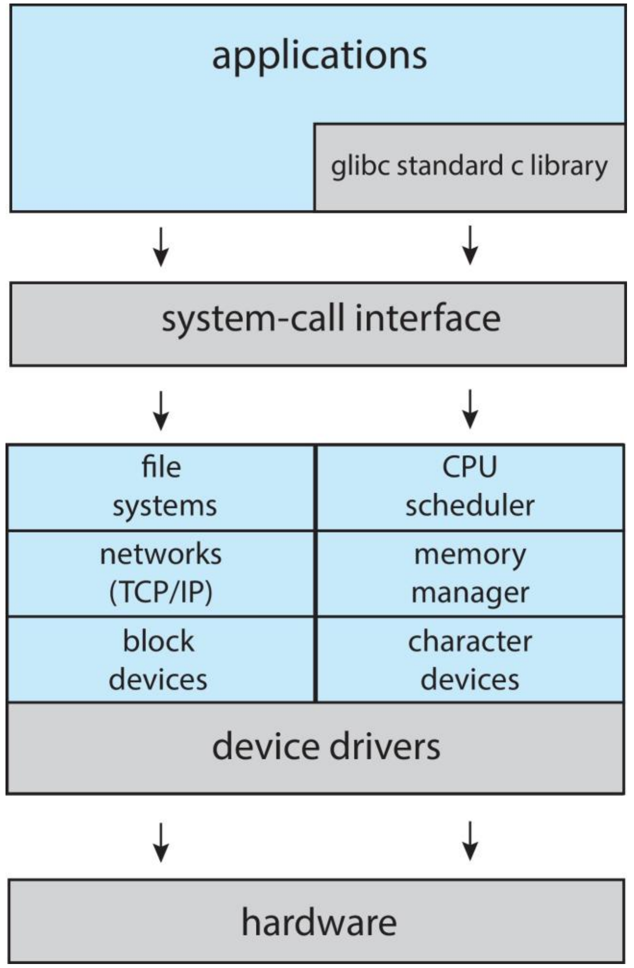
Beyond simple but not fully layered



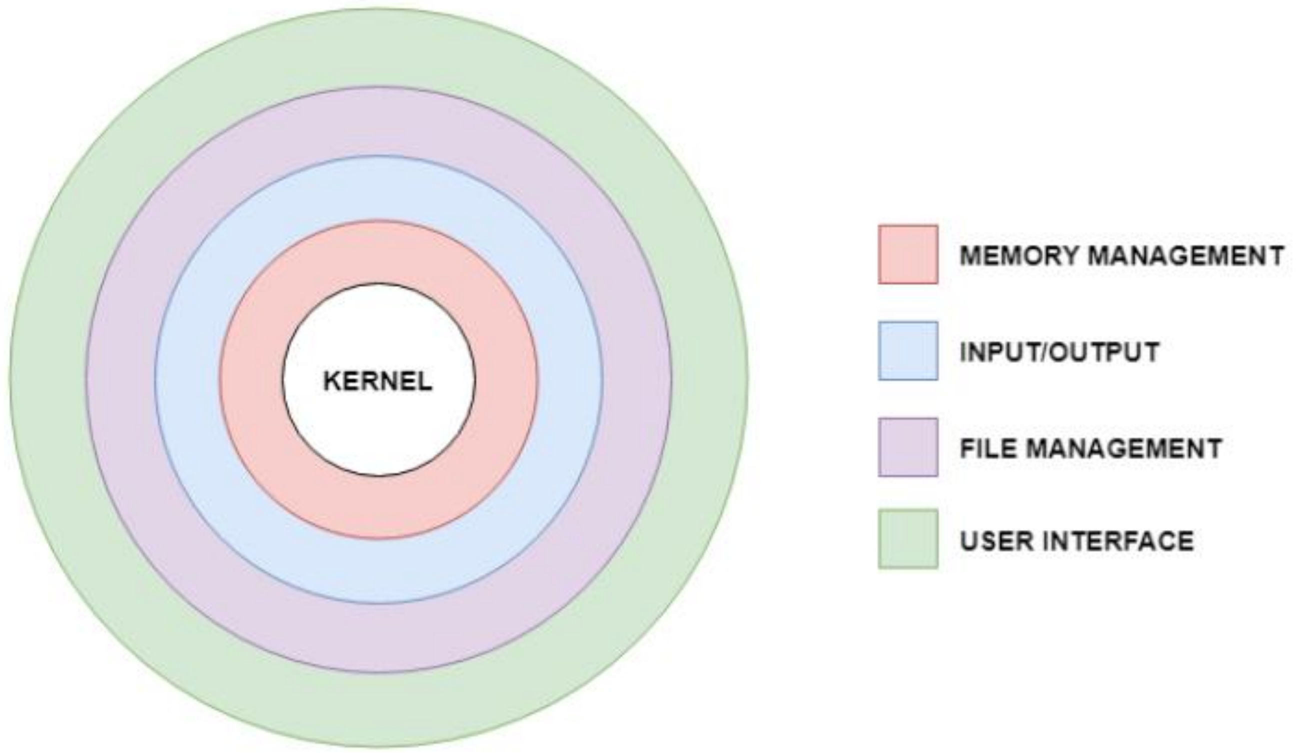

Android
Servers
Routers
Smart Tv's
Super computers
Refrigerators
funny little penguin, lol
LINUX

Linux System Structure

Monolithic plus modular design



Layered Approach

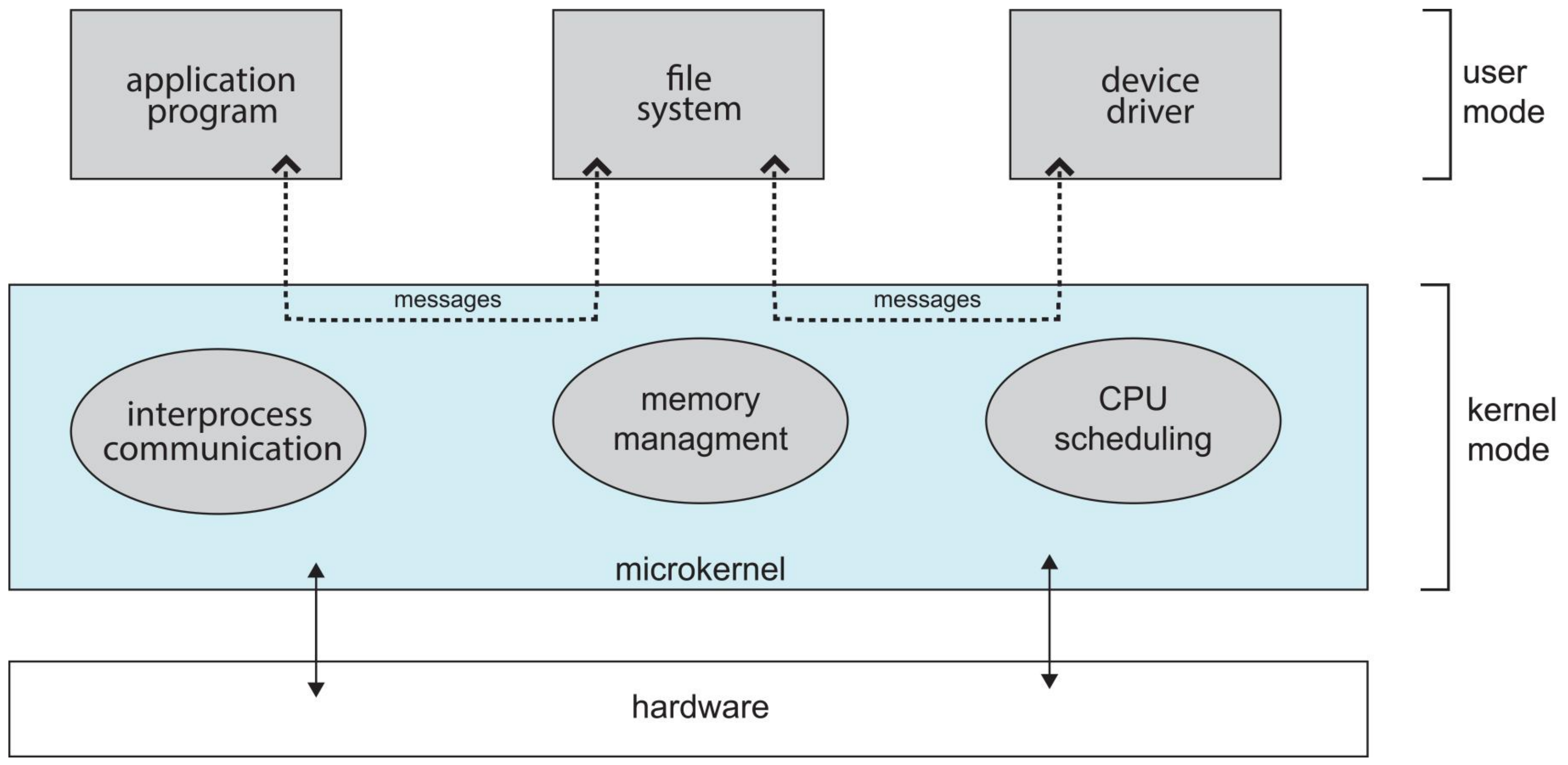


Layered Operating System Design

Microkernels

- Moves as much from the kernel into user space
- **Mach** example of **microkernel**
  - Mac OS X kernel (**Darwin**) partly based on Mach
- Communication takes place between user modules using **message passing**
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

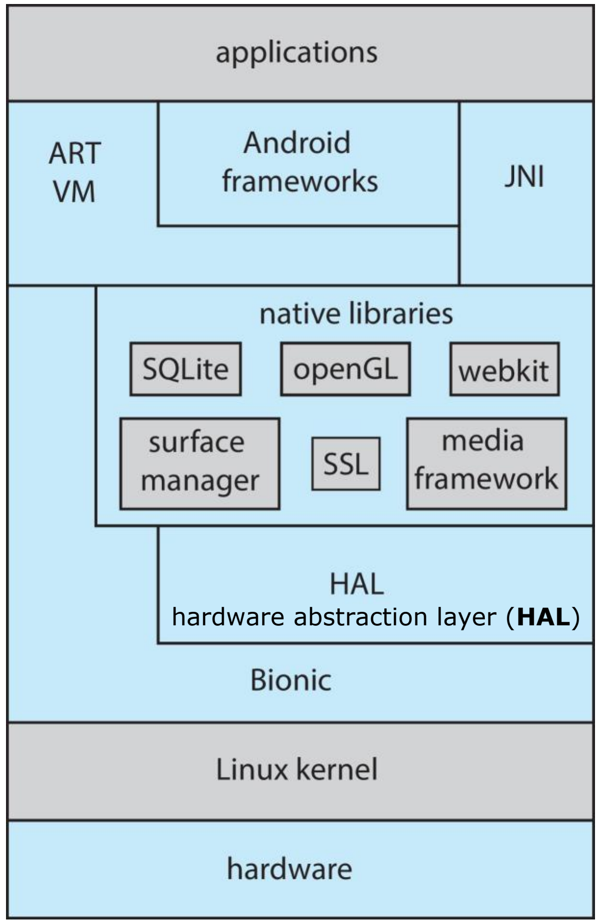
Microkernel System Structure



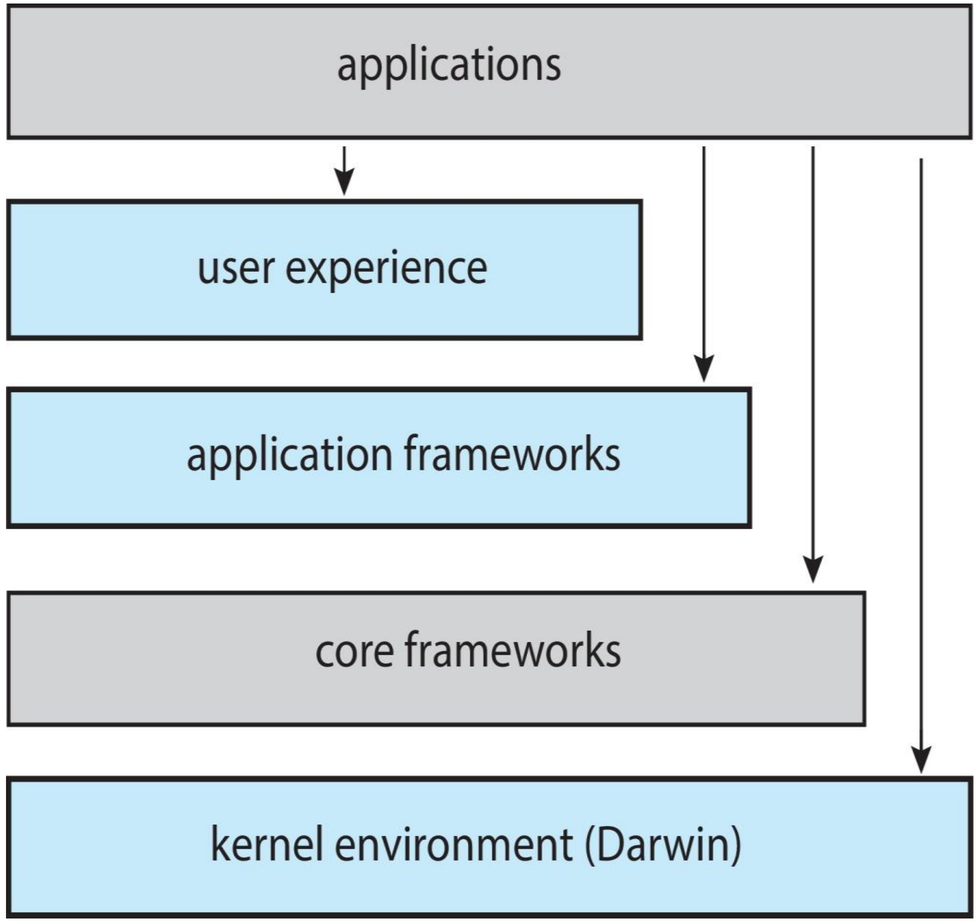
Modules

- Many modern operating systems implement **loadable kernel modules** (**LKMs**)
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible
  - Linux, Solaris, etc

Android vs iOS



Android Architecture



macOS and iOS Structure

Building and Booting Linux

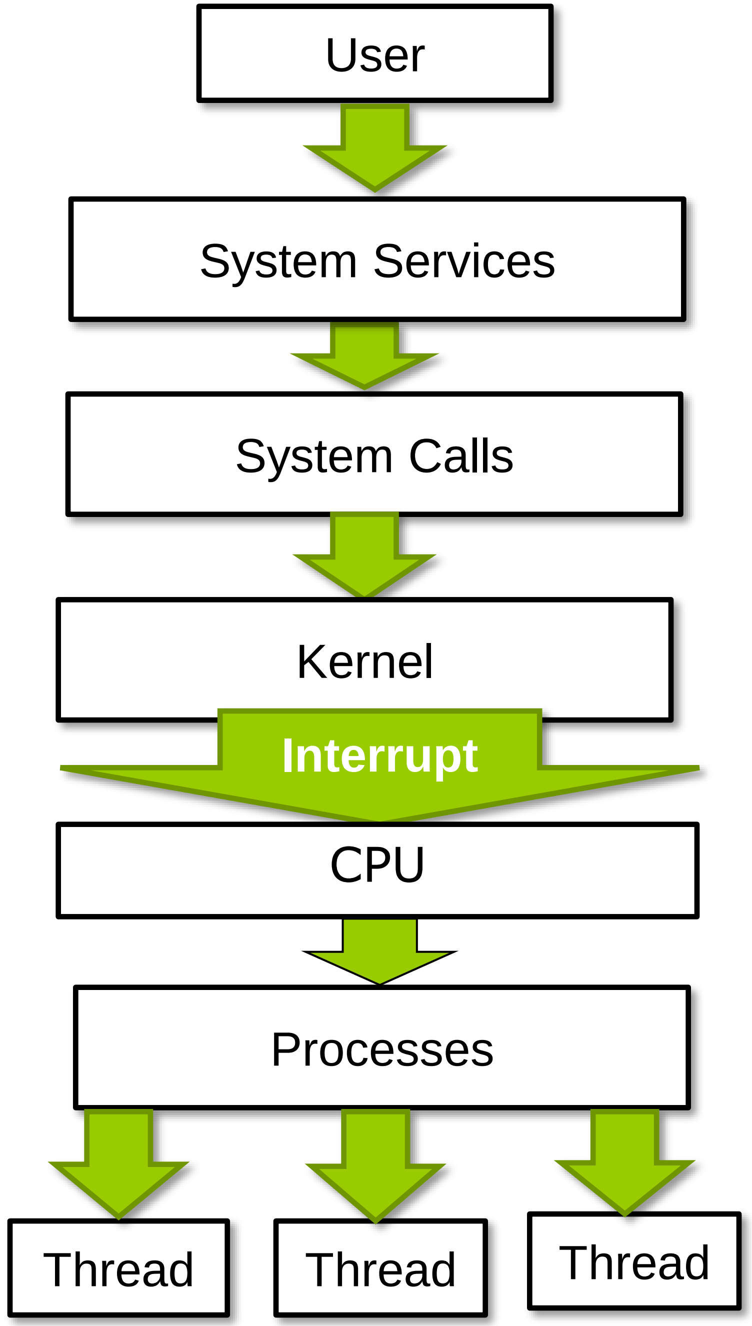
- Download Linux source code (http://www.kernel.org)
- Configure kernel via "`make menuconfig`"
- Compile the kernel using "`make`"
  - Produces `vmlinuz`, the kernel image
  - Compile kernel modules via "`make modules`"
  - Install kernel modules into `vmlinuz` via "`make modules_install`"
  - Install new kernel on the system via "`make install`"

Operating-System Debugging

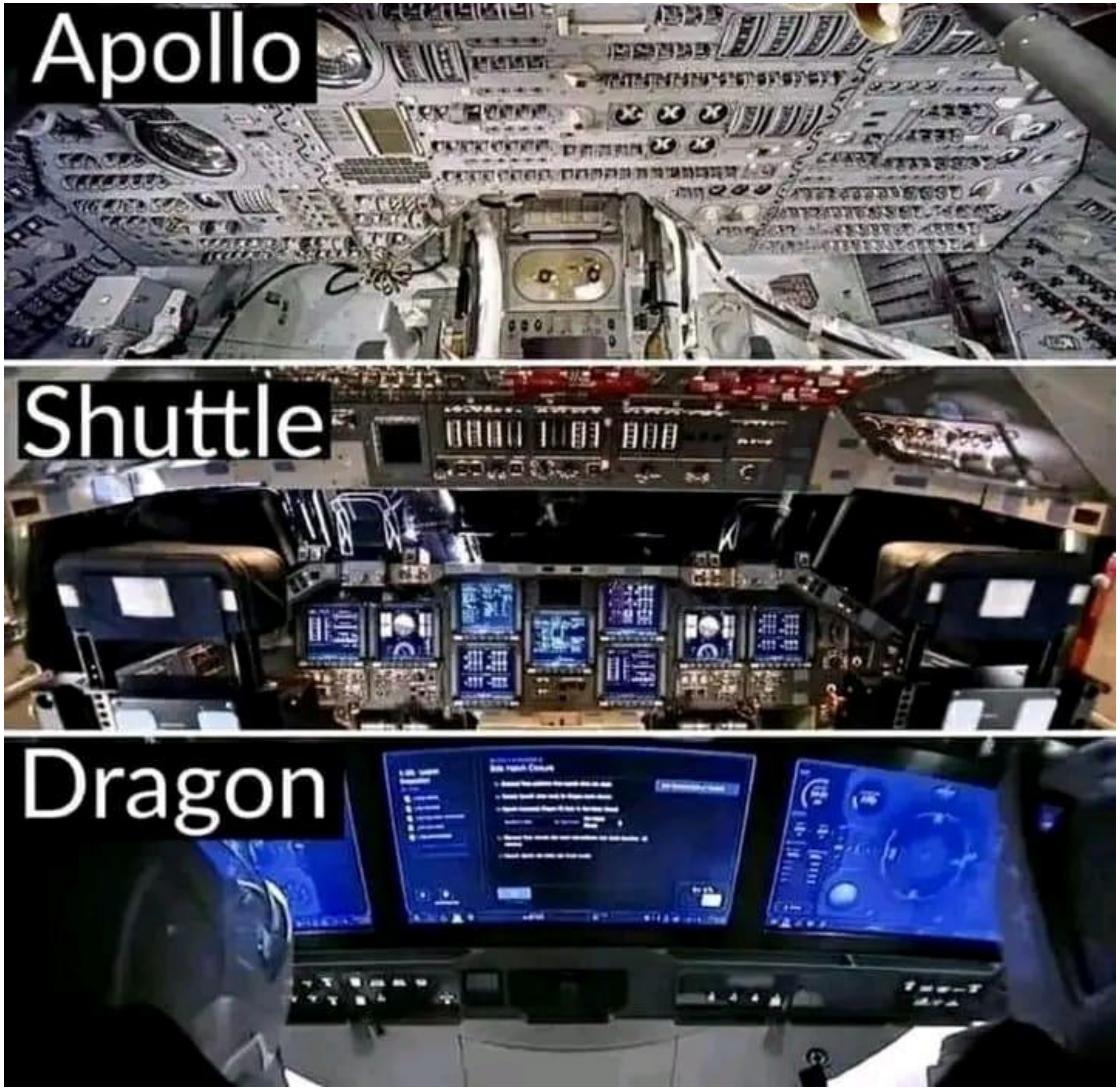
- **Debugging** is finding and fixing errors, or **bugs**
- Also **performance tuning**
- OS generate **log files** containing error information
- **Failure of an application can** generate **core dump** file capturing memory of the process
- **Operating system** failure can generate **crash dump** file **containing kernel memory**
- Beyond crashes, performance tuning can optimize system performance
    - Sometimes using ***trace listings*** of activities, recorded for analysis
    - **Profiling** is periodic sampling of instruction pointer to look for statistical trends

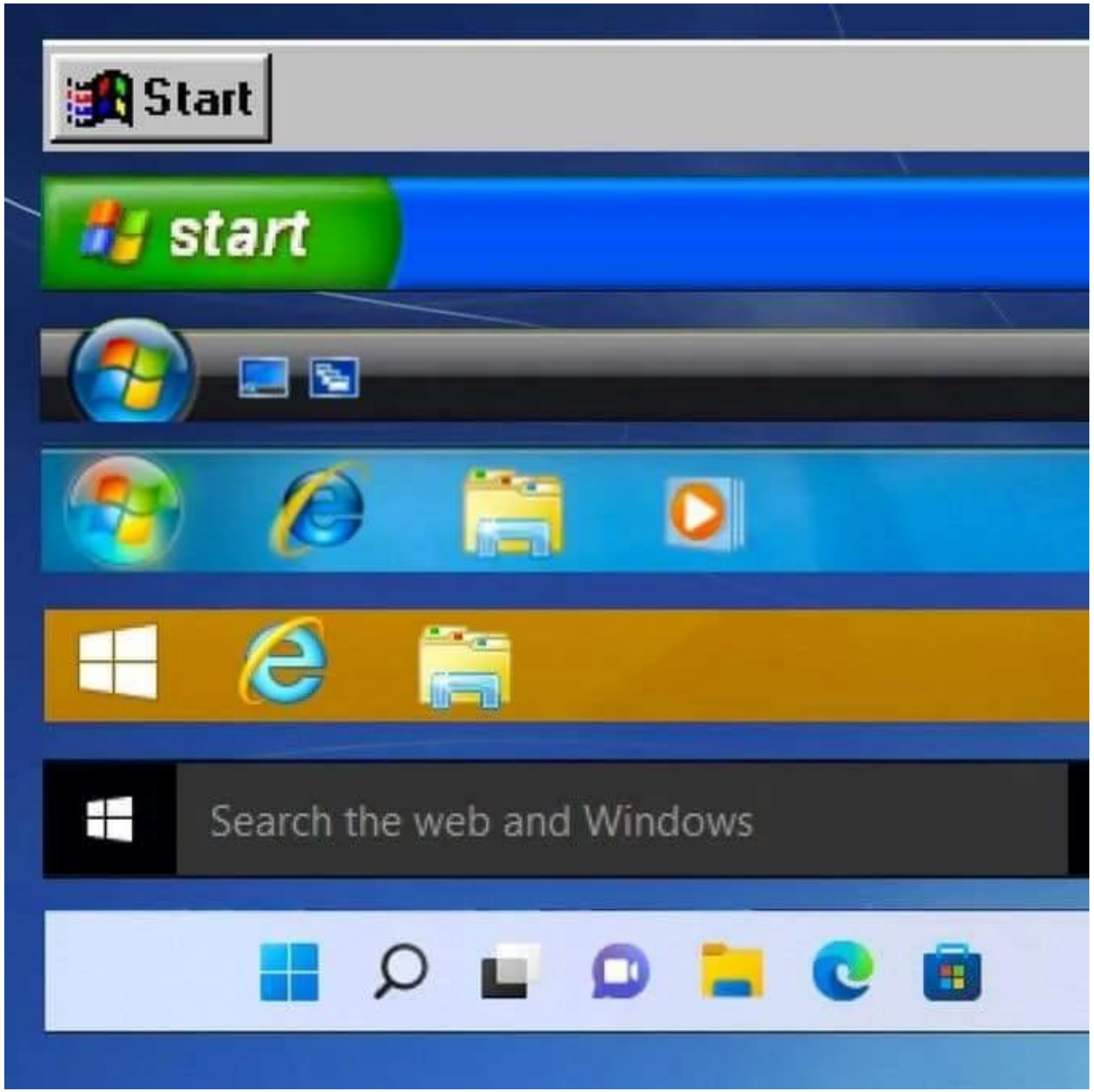
Kernighan's Law: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

The Big Pic of OS



UX vs. UI


Apollo
Shuttle
Dragon


Start
start
Search the web and Windows

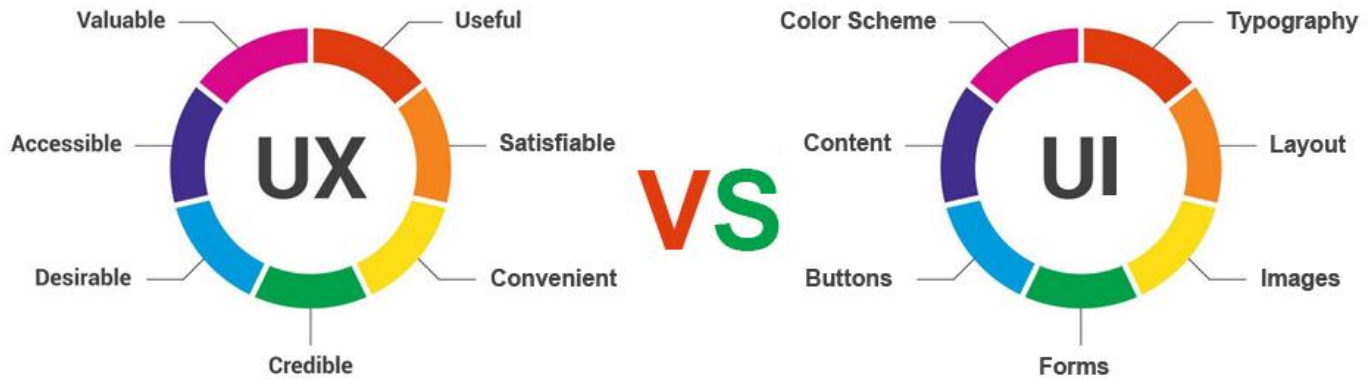
A list of UX/UI Words

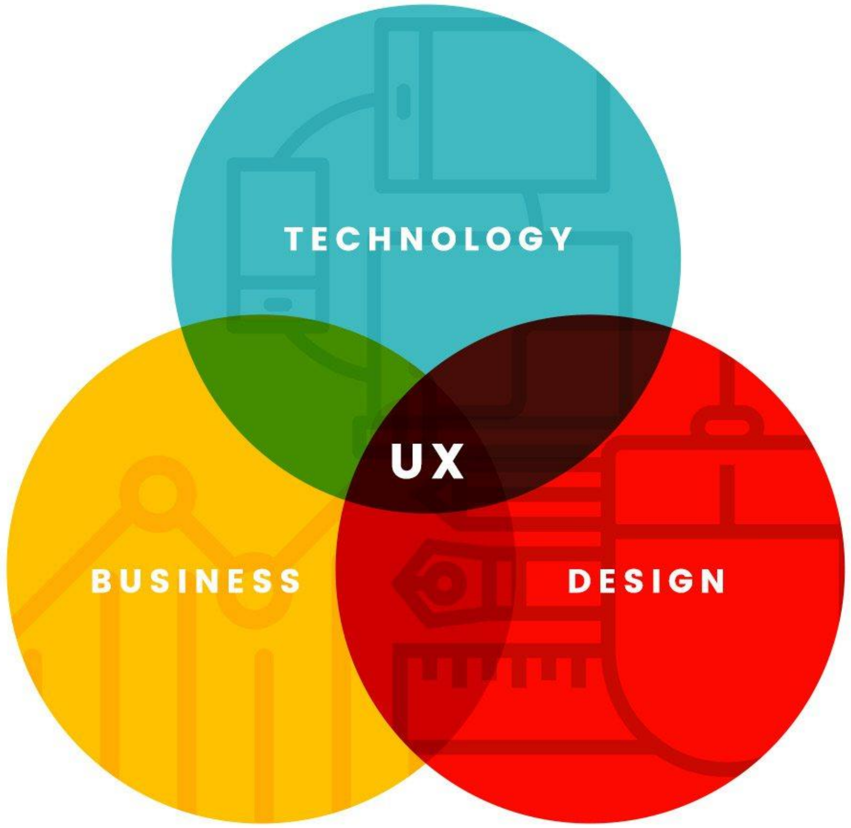
UX

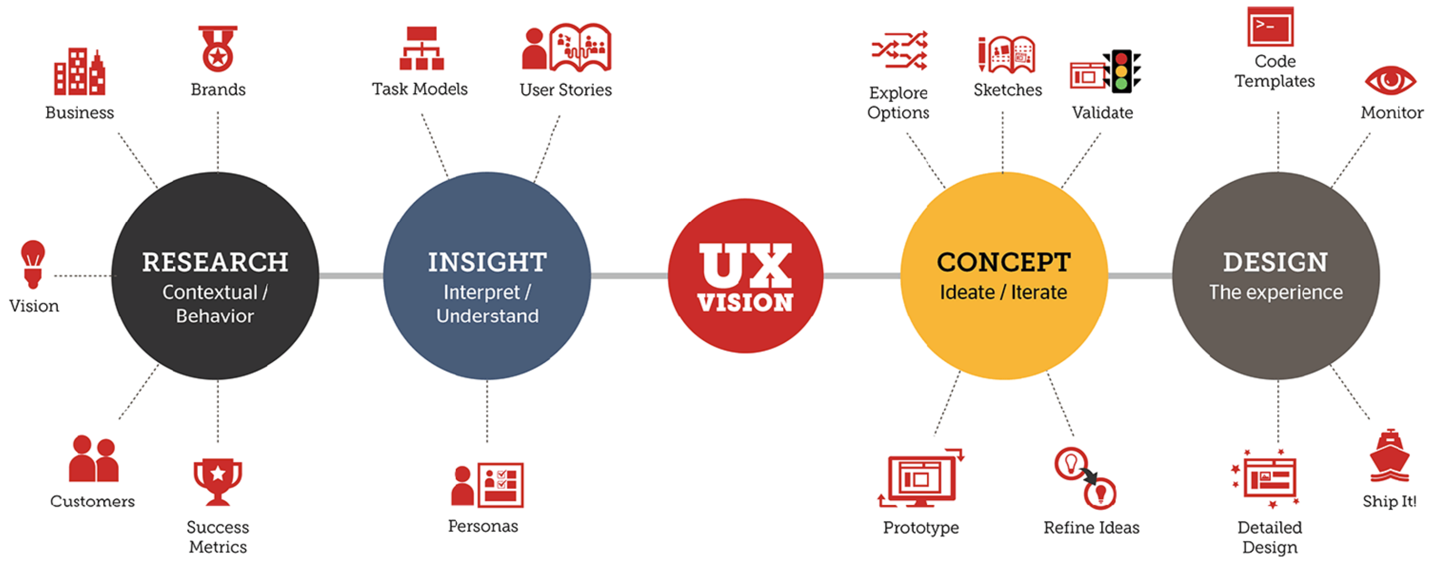
- UX Pyschology
- UX Methods
- UX Strategy
- Design Process
- Design Research
- Inclusive Design
- Mental Models
- Empathy Mapping
- Dark Patterns
- Universal Design
- Accessibility
- User Reserch
- User Interviews
- User Journey
- User Personas
- Affinity Diagram
- UX Microcopy
- Gamification
- UX Deliverables

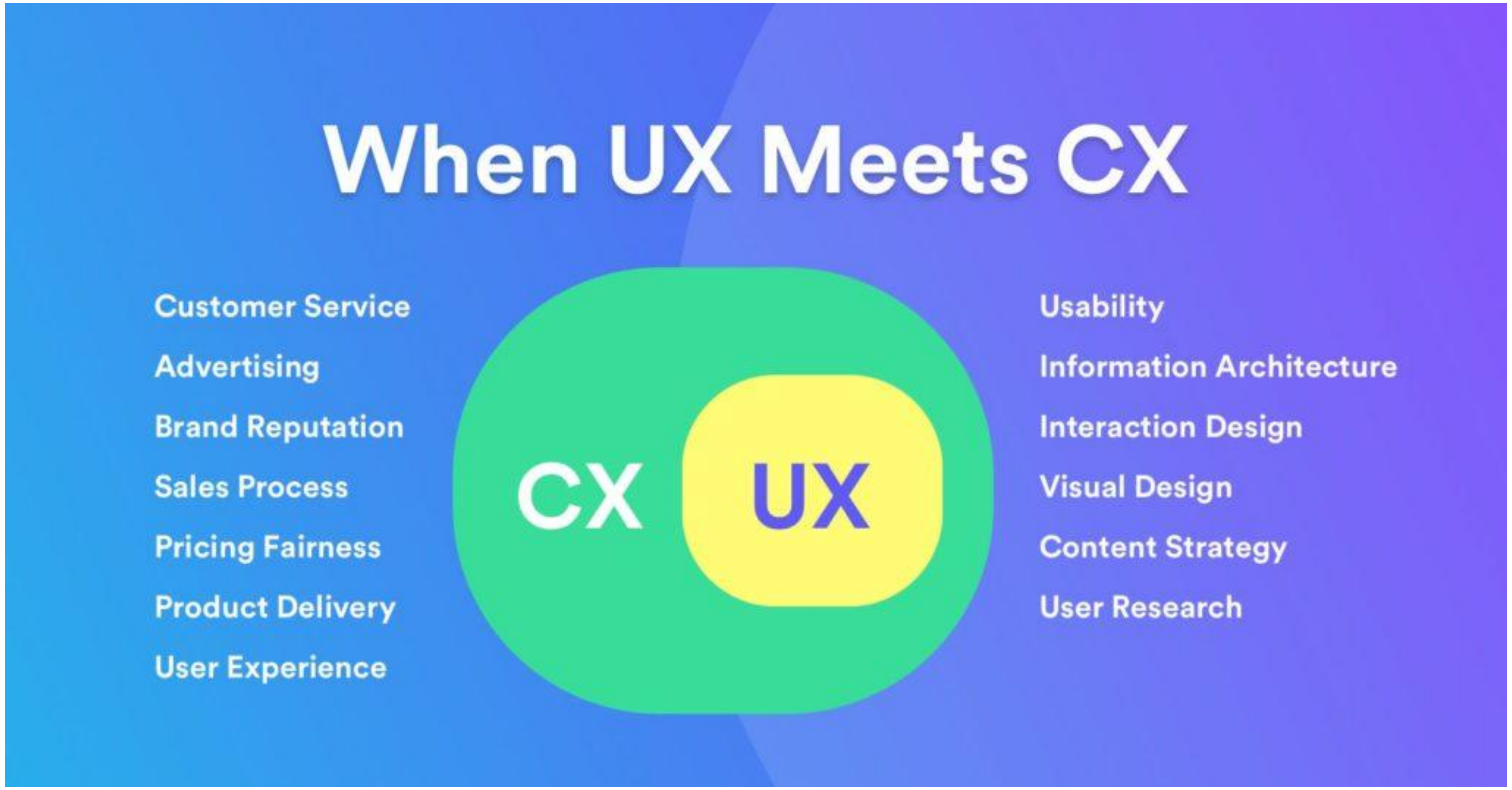
UI

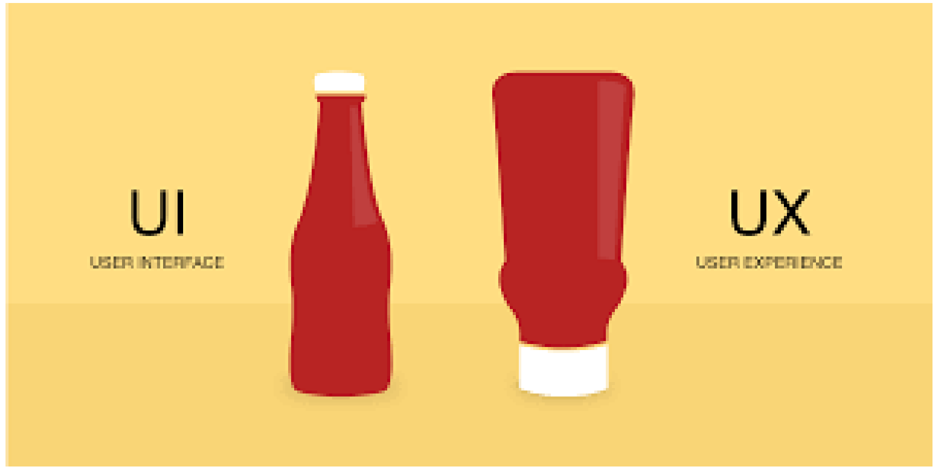
- Visual Design
- Design System
- Grids
- Information Architecture
- Icons
- Typography
- Empty States
- Motion Design
- Prototyping
- App Design
- Interaction Design
- Heuristic Evaluation


Valuable
Useful
Accessible
UX
Satisfiable
Desirable
Convenient
Credible
VS
Color Scheme
Typography
Content
UI
Layout
Buttons
Images
Forms


TECHNOLOGY
UX
BUSINESS
DESIGN


Business
Brands
Task Models
User Stories
Explore Options
Sketches
Validate
Code Templates
Monitor
Vision
RESEARCH
Contextual / Behavior
INSIGHT
Interpret / Understand
UX VISION
CONCEPT
Ideate / Iterate
DESIGN
The experience
Customers
Success Metrics
Personas
Prototype
Refine Ideas
Detailed Design
Ship It!


When UX Meets CX
Customer Service
Advertising
Brand Reputation
Sales Process
Pricing Fairness
Product Delivery
User Experience
CX
UX
Usability
Information Architecture
Interaction Design
Visual Design
Content Strategy
User Research


UI
USER INTERFACE
UX
USER EXPERIENCE