CS 202
Advanced Operating Systems

Winter 26

Lecture 3: OS Architecture

Instructor: Chengyu Song

What is an OS?

- Operating System
  - (of a person) control the functioning of (a machine, process, or system)

    "a shortage of workers to operate new machines"
  - a set of things working together as parts of a mechanism or an interconnecting network

    "the state railroad system"
  - a set of principles or procedures according to which something is done

    "a multiparty system of government"

- The software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals.

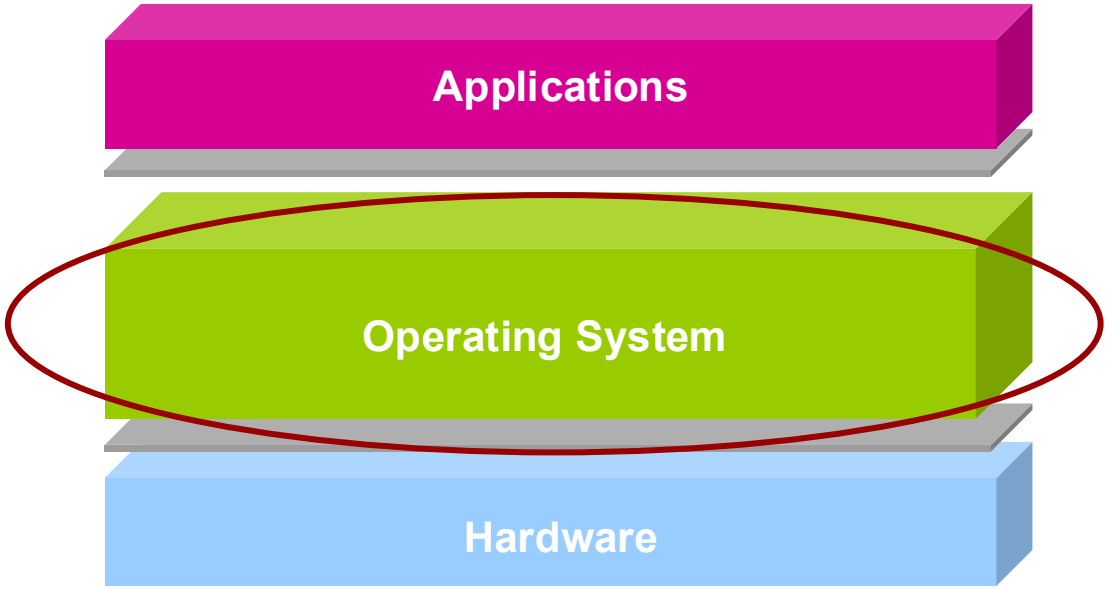
Definitions from Oxford Language

Topics for today

- What is an operating system?
- What is OS architecture / organization?
- Why does OS architecture / organization matter?
- Typical OS architectures

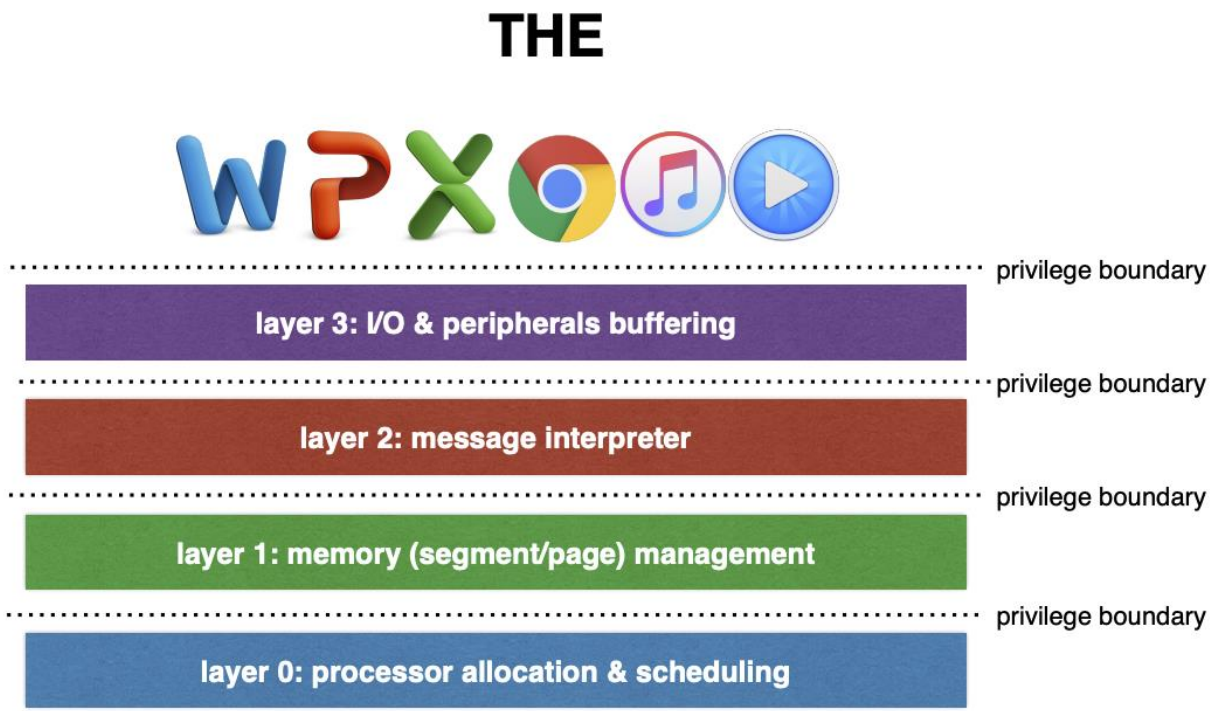
What is OS architecture?

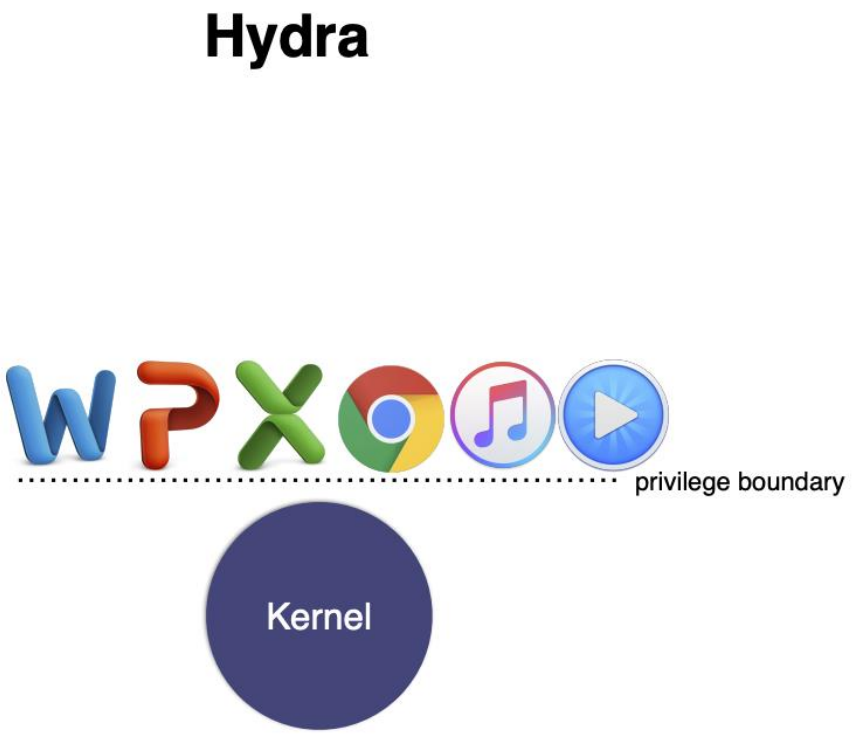
- The design and organization of the software layer that defines the interface between applications and physical resources



OS architecture

- How OS functionalities are organized, and how components interact
- An earlier example:





Topics for today

- What is an operating system?
- What is OS architecture / organization?
- Why does OS architecture / organization matter?
- Typical OS architectures

Fundamental Issues

- The fundamental issues in OS designs are:
  - **Functionality**: how to enable the users/programs to do more?
  - **Performance**: how to do better?
  - Management: how to allocate and schedule resources for tasks?
  - **Protections**: how to make sure things won't go wrong?
  - Communication: how to enable collaboration?
  - Security: how to create a safe environment?
  - Reliability and fault tolerance: how to mask failures?

Why does OS architecture matter?

- Flexibility / Extensibility
  - How to support different hardware?
  - How to enable new hardware to applications?
  - How to adapt to different needs of applications?
- Performance
  - Does the structure facilitate good performance?
  - Scalability (to more applications / hardware resources)
  - Responsiveness (latency of event responses)
- Protection
  - User (mistakes, attacks)
  - System (bugs, faults, attacks)

U/U	U/S
S/U	S/S

Extensibility

- How to support different hardware?
  - CPU / ISA
  - MMU / memory access
  - Disk
  - Devices
- How to enable new hardware to applications?
  - Known hardware types / functionalities
  - New hardware types / functionalities

Extensibility

- How to adapt to different needs of applications?
  - Compile-time customization
    - » Linux kconfig
  - Boot-time configuration
    - » Boot options
  - Runtime reconfiguration
    - » /sys, /proc/sys, prctl, personality, priority
  - Programmable kernel extensions
    - » bpf, ebpf

One size does not fit all!

Performance

- What is bad for performance?
  - Copy (data transfer)
  - Wait (dependencies)
  - Cache misses
  - Checks

- Examples
  - Syscall: CPU context switch (saving / loading), copying args, checking args
  - Process switch: address space switch (TLB / predictors)
  - Swap / thrashing

Protection

- How to protect users and system?
  - Isolation (privilege mode, address spaces)
  - Verification (eliminate bugs / mistakes)
  - Checks (access control, defensive programming)

Fundamental Issues

operation

- The fundamental issues in ~~OS designs~~ are:
  - **Functionality**: how to enable the users/programs to do more? ➡ **add more code**
  - **Performance**: how to be faster? ➡ **remove the boundaries, eliminate checks**
  - Management: how to allocate and schedule resources for tasks?
  - **Protections**: how to make sure things won't go wrong? ➡ **simple, isolations, checks**
  - Communication: how to enable collaboration?
  - Security: how to create a safe environment?
  - Reliability and fault tolerance: how to mask failures?

Fundamental Trade-offs

- How to serve more customers?
  - Offer more products / services
  - Hire more people
- How to teach more students?
  - Offer more degrees
  - Establish more departments
  - Hire more people
- How to serve more citizens?
  - Offer more services
  - Establish more departments
  - Hire more people

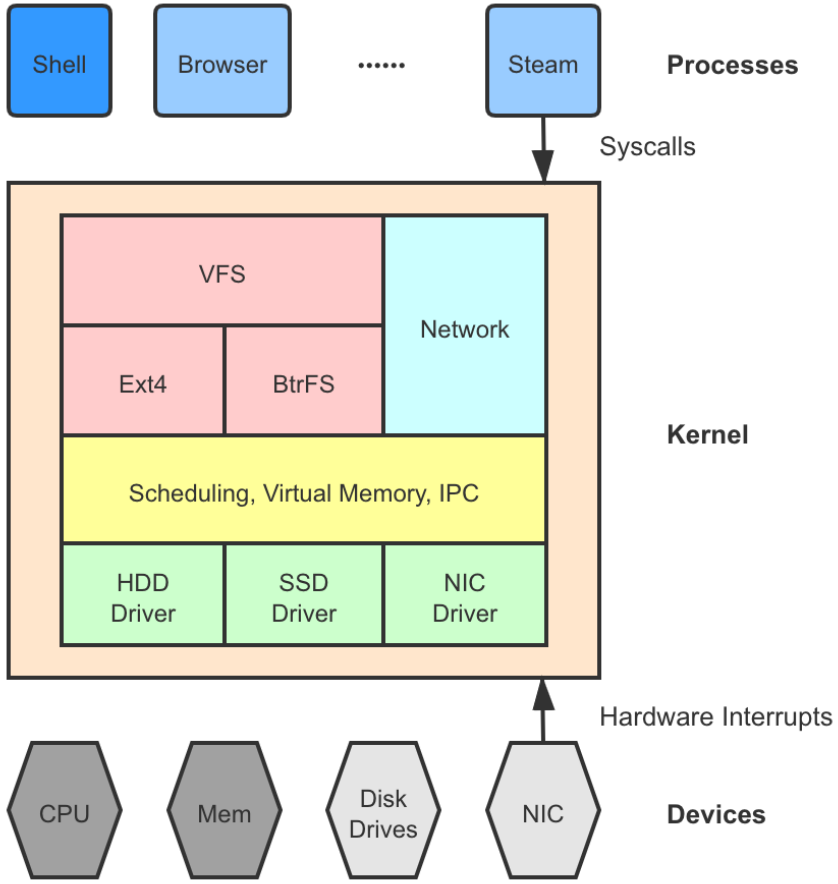
- **Bureaucracy**
- **Bloat**
- **Inefficiency**
- **Slow to adapt**
- **Corruptions**

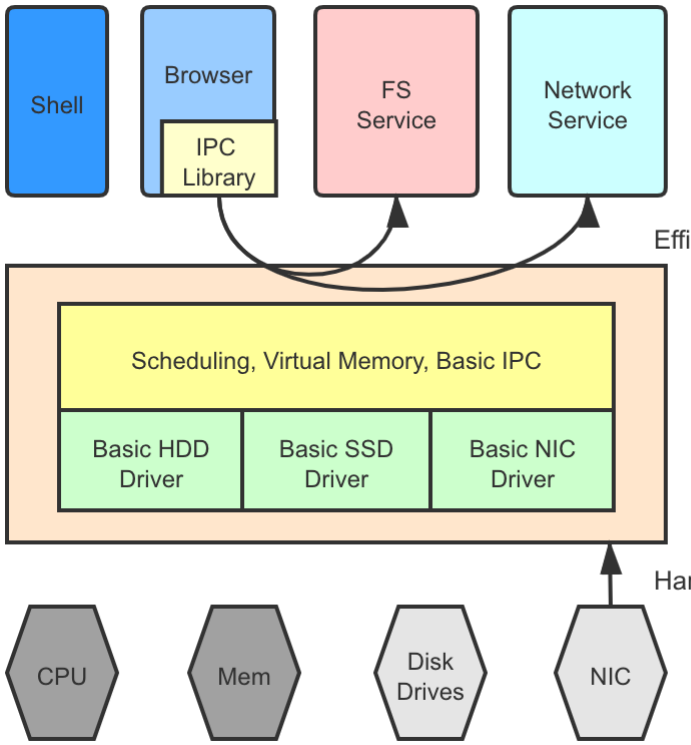
Topics for today

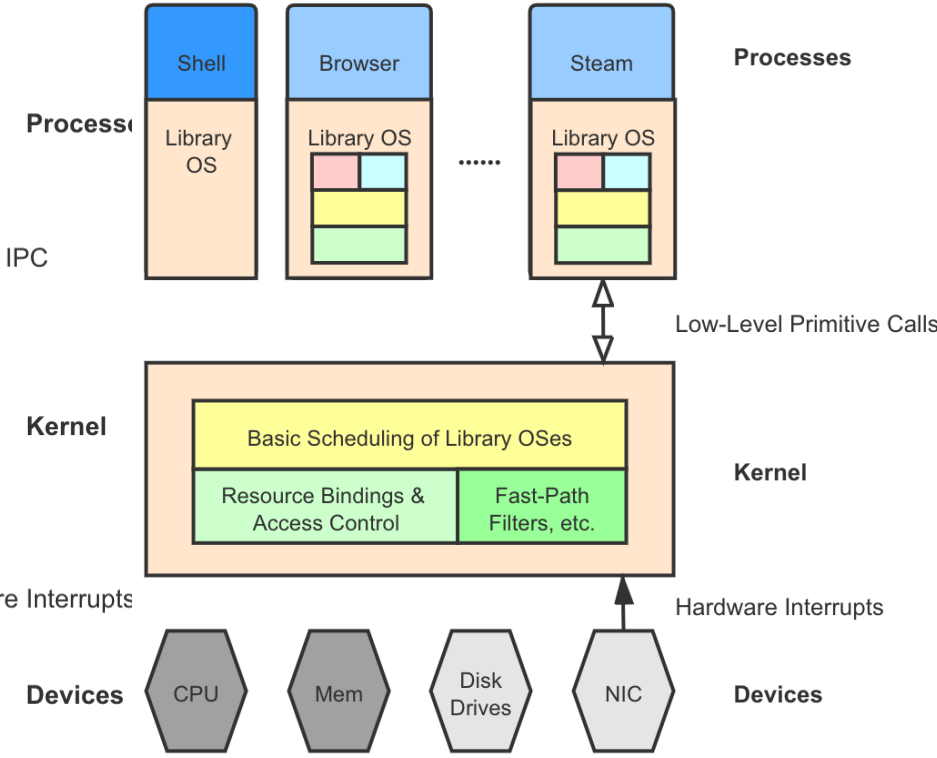
- What is an operating system?
- What is OS architecture / organization?
- Why does OS architecture / organization matter?
- Typical OS architectures

Typical OS architectures

- Monolithic
  - UNIX, Linux, Windows
- Micro-kernel
  - L4, Mach, QNX
- LibraryOS
  - exokernel, graphene



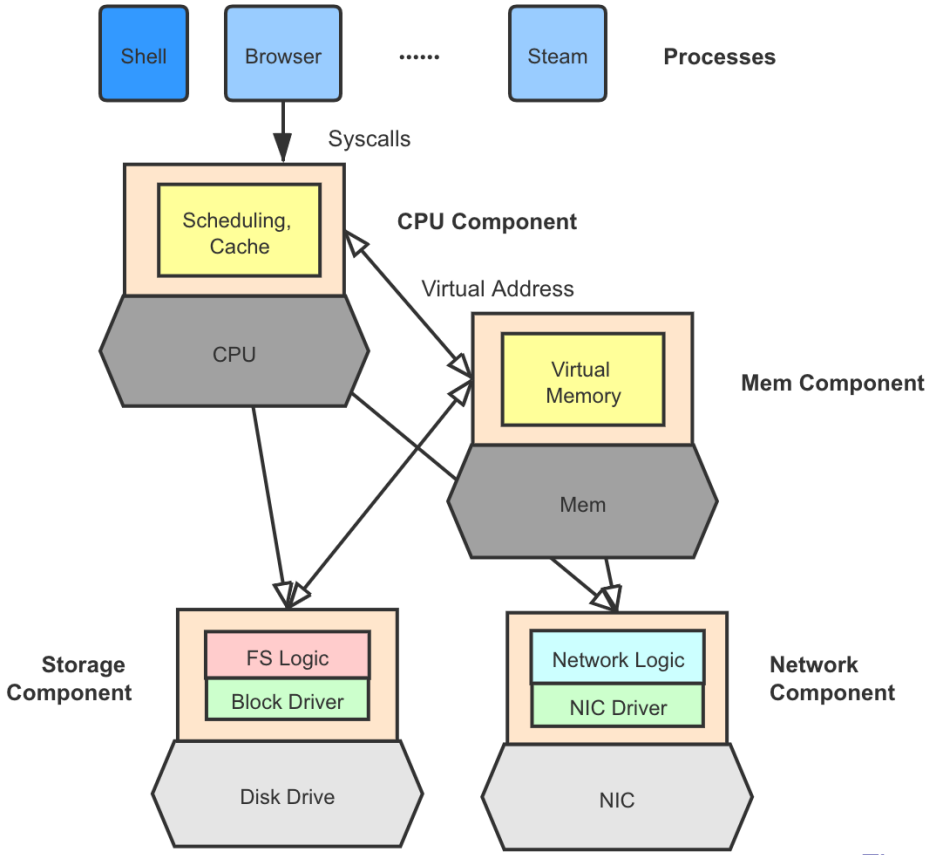


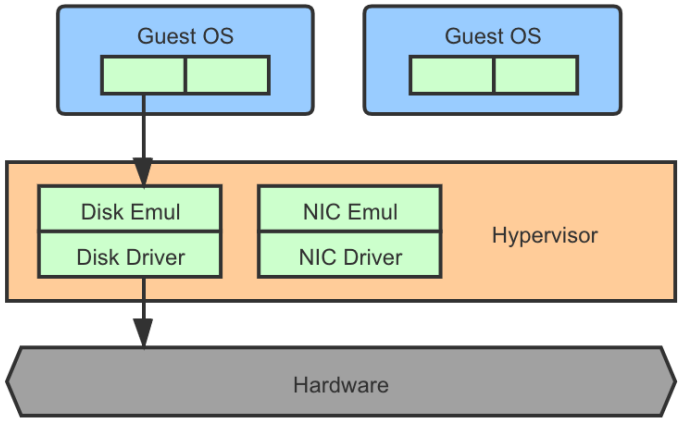


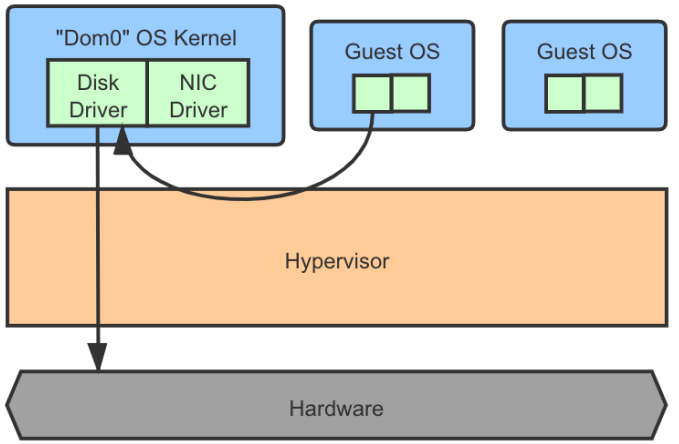
Figures from https://www.josehu.com/technical/2021/05/24/os-kernel-models.html

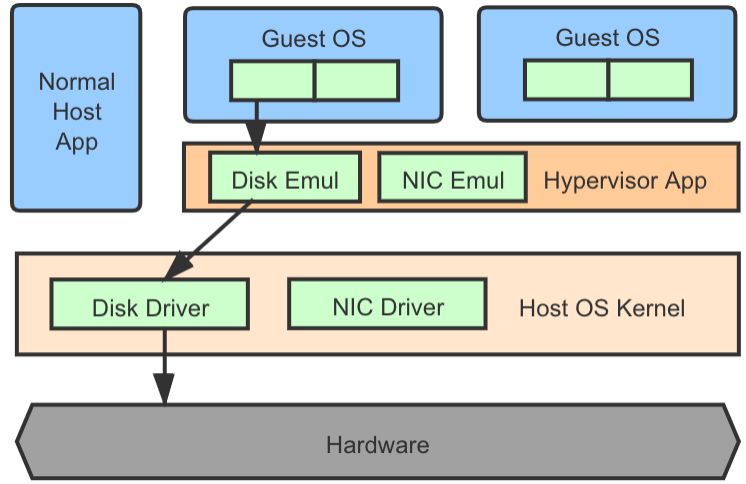
Typical OS architectures

- Multi-kernels
  - Barrelfish, LegoOS
- Hypervisors
  - VMware ESX, Xen, KVM



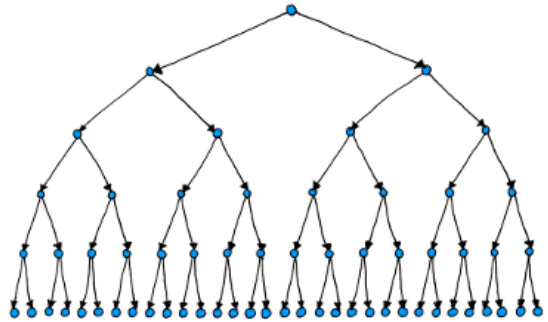




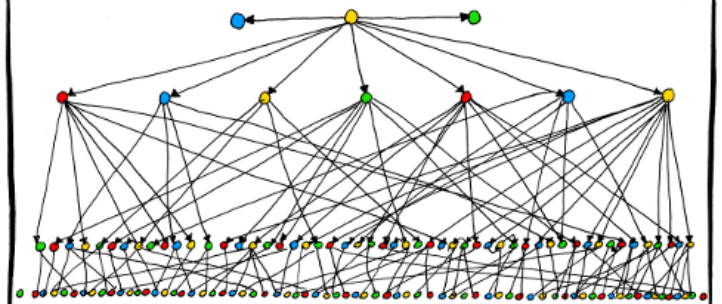


Figures from https://www.josehu.com/technical/2021/05/24/os-kernel-models.html

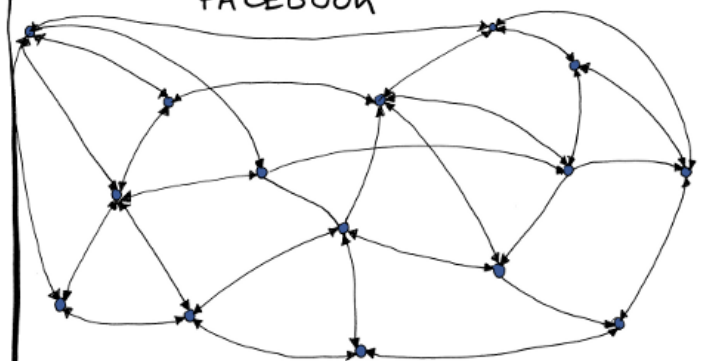
AMAZON



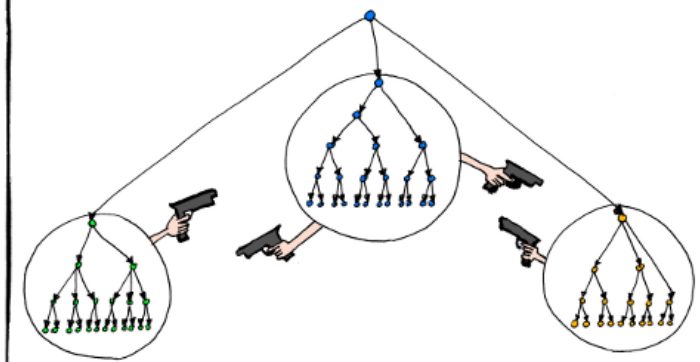
GOOGLE



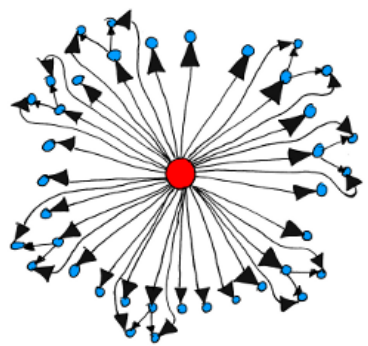
FACEBOOK



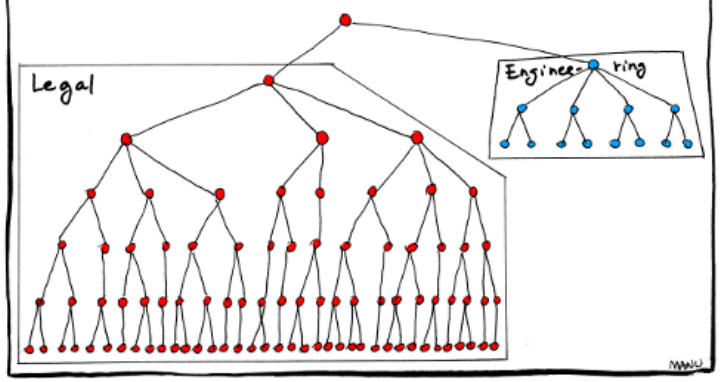
MICROSOFT



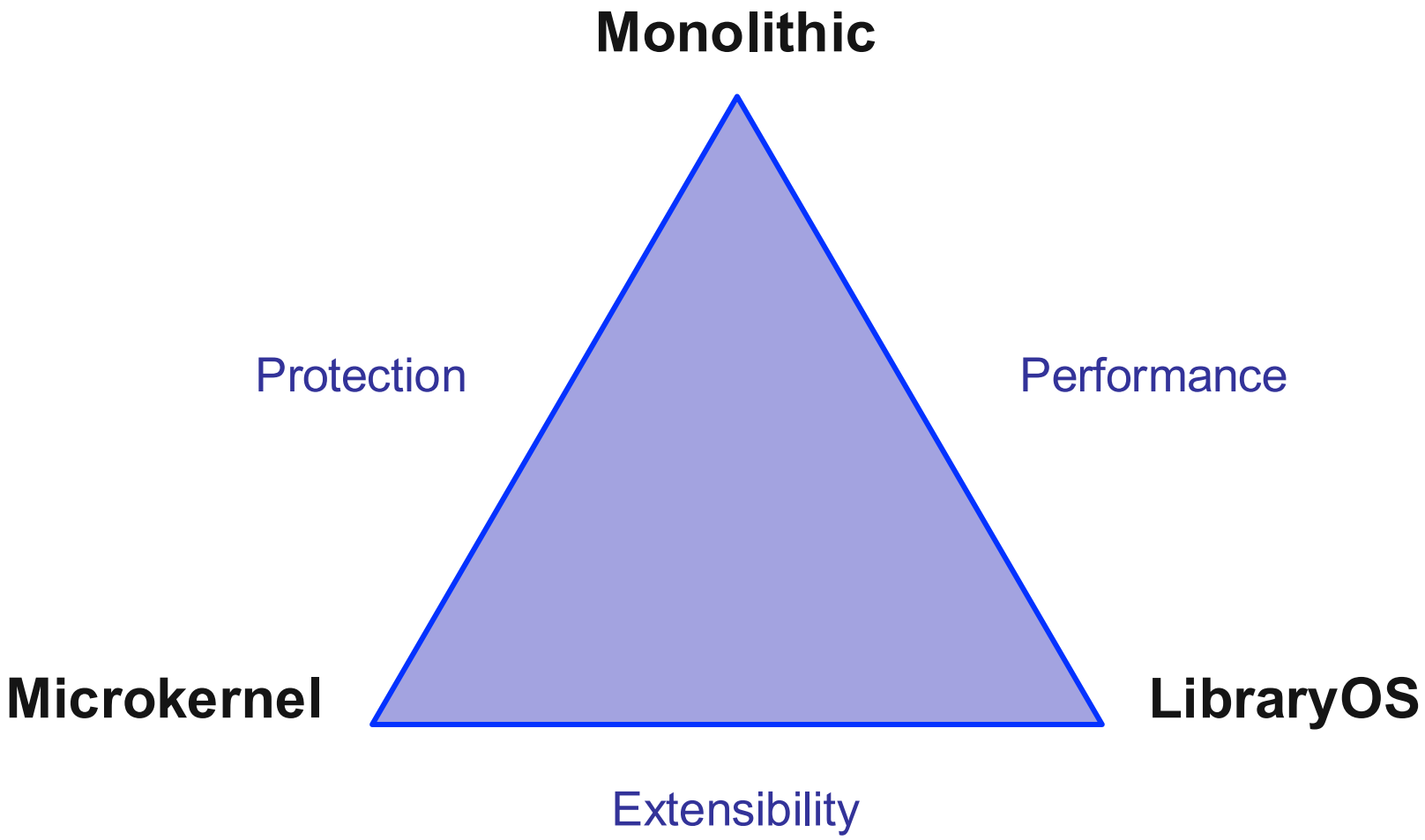
APPLE



ORACLE



Trade-offs of OS architectures



Trade-offs of OS architectures

- Monolithic kernels
  - Good performance
  - Okay protections (has isolation, too big to be bug free, hard to verify)
  - Bad extensibility
- Microkernels
  - Very good protection (isolation, small, verifiable)
  - Okay extensibility
  - Bad performance
- LibraryOS
  - Very good extensibility
  - Good performance
  - Bad protection (no isolation between U/libOS, exposes everything)

Exokernel

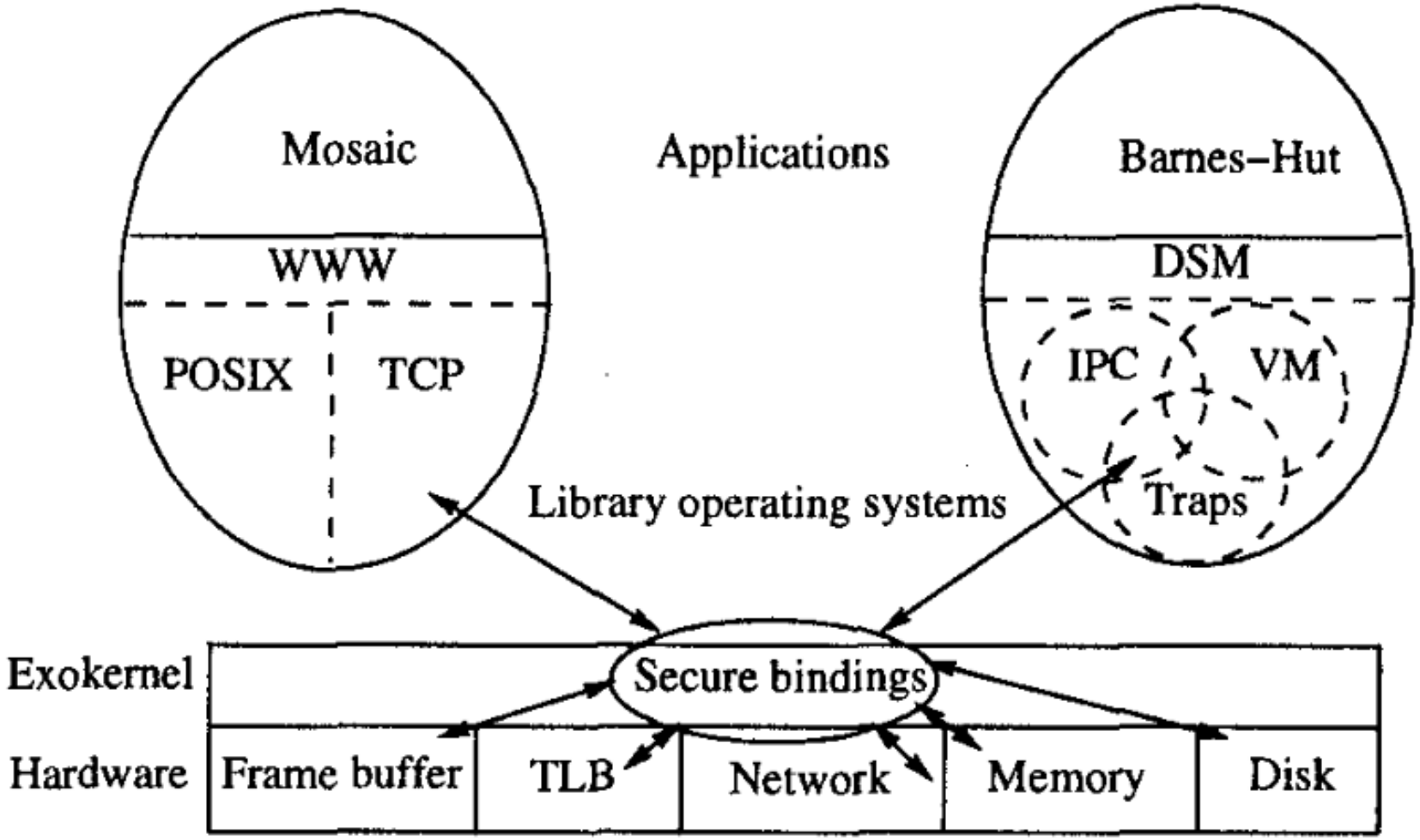
- Motivations: fixed high-level abstractions are hurting applications
  - Fixed high-level abstractions hurt application performance by denying domain-specific optimizations
    - » Scheduler, cache eviction policy
    - » Evidences: Cao et al. [1 O] measured that application-controlled file caching can reduce application running time by as much as 45%.
  - Fixed high-level abstractions discourages changes to existing abstractions and limit the functionality of applications
    - » Syscalls rarely change, if ever
    - » Evidences: few research ideas have been adopted
  - Fixed high-level abstractions hide information from applications and prevent it from making good decisions
    - » Low-level exceptions, interrupts, raw device I/O
    - » Evidences: database implementors must struggle to emulate random-access record storageon top of tile systems [47].

Exokernel

- Core idea: **Separating Protection from Management**
  - Exokernel: provides protection
  - LibraryOS: implements management and high-level abstractions

- Solution: securely multiplexes available hardware resources through a low-level interface
  - Low-level: even more flexible / aggressive than microkernel
  - No emulation: direct exposing hardware (including naming): much lower overhead than virtual machine monitors
  - Securely: trust but verify

Exokernel: architecture



Exokernel: Design Princincple

- **Securely expose hardware**
  - An exokernel strives to safely export all privileged instructions, hardware DMA capabilities, and machine resources
- **Expose allocation**
  - An exokernel should allow library operating systems to request specific physical resources
- **Expose names**
  - An exokernel should export physical names
- **Expose Revocation**
  - An exokemel should utilize a visible resource revocation protocol so that an well-behaved libraryOS can perform effective application-level resource management
  - All revocations, including CPU (i.e., explicit context switch)
- **Arbitration**: same as other OS

Exokernel: secure bindings

- Tracks resource ownership
  - Allocation to libraryOS at bind time
  - Free to use afterwards, no more authorization
- Protection (ownership) checks are simple operations performed by the kernel
  - Hardware mechanisms (if available, e.g., capability tokens)
    - » Memory: TLB, page tables, DMA
  - Software caching (bookkeeping)
  - Downloadable application code
    - » Packet filter
- Allows protection without understanding

Exokernel: secure binding example (1)

- Virtual-to-Physical address translation
  - Protecting which physical pages can be accessed by a libraryOS
- TLB-based
  - LibraryOS decide the virt-to-phy mapping
  - LibraryOS decide the TLB replacement policy
  - Binding presented to exokernel
  - Exokernel checks ownership, if passed, installs/removes entries in hardware TLB
  - LibraryOS can access physical memory freely
- Page-table based
  - More challenging, Xen, hardware virtualization

Exokernel: secure binding example (2)

- Packet filtering
  - Handle packet distributing (who handles?) and processing (e.g., firewall)
  - In-kernel processing is much faster than polling / querying
- Manage packet handling from library (app policy)
  - Packet filter code provided by library (app)
  - In the form of downloadable code
  - Binding (code) presented to exokernel
  - Exokernel verifies the code
  - Exokernel installs app's packet filter code (e.g., to hardware NIC)

Exokernel: hypotheses

- The design and implementation of exokernel is built upon four hypotheses that must be validated through empirical evaluation
  - Exokemels can be very efficient
  - Low-level, secure multiplexing of hardware resources can be implemented efficiently
  - Traditional operating system abstractions can be implemented efficiently at application level
  - Applications can create special-purpose implementations of these abstractions
- Q: how to design experiments?