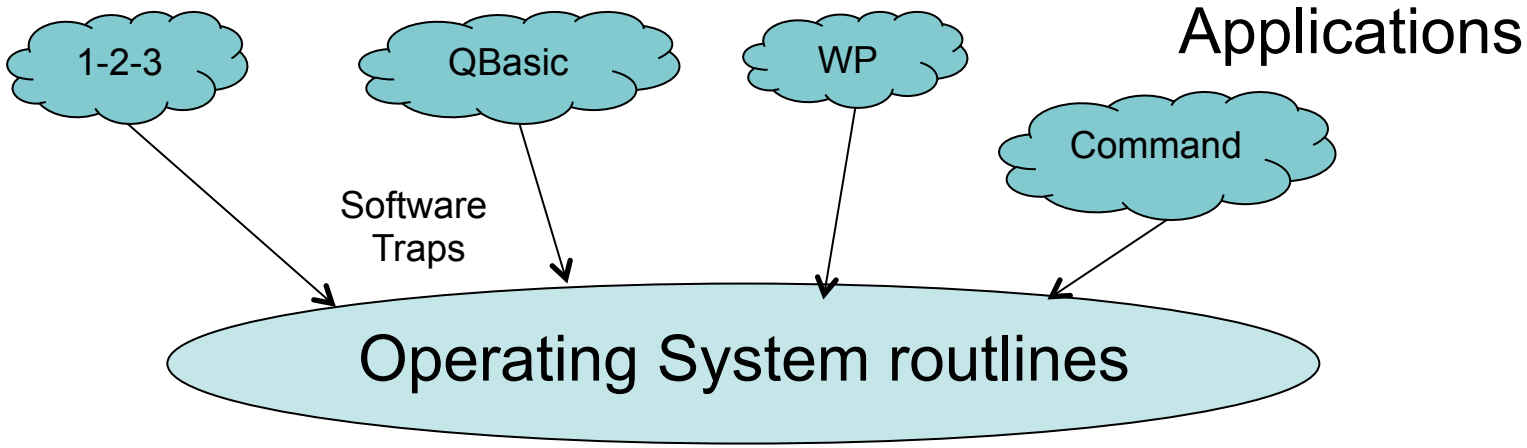
Operating System Architectures

- Learning objectives:
  - Explain how OS functionality is orthogonal to where you place services relative to processor modes.
  - Describe some alternative ways to structure the operating system.
- Operating systems evolve over time, but that evolution is frequently in terms of their architecture: how they structure functionality relative to protection boundaries.
- We'll review some of the basic architectures:
  - Executives
  - Monolithic kernels
  - Micro kernels
  - Exo kernels
  - Extensible operating systems

OS Executives

- Think MS-DOS: With no hardware protection, the OS is simply a set of services:
  - Live in memory
  - Applications can invoke them
  - Requires a software trap to invoke them.
  - Live in same address space as application



Monolithic Operating System

- Traditional architecture
  - Applications and operating system run in different address spaces.
  - Operating system runs in privileged mode; applications run in user mode.

Applications

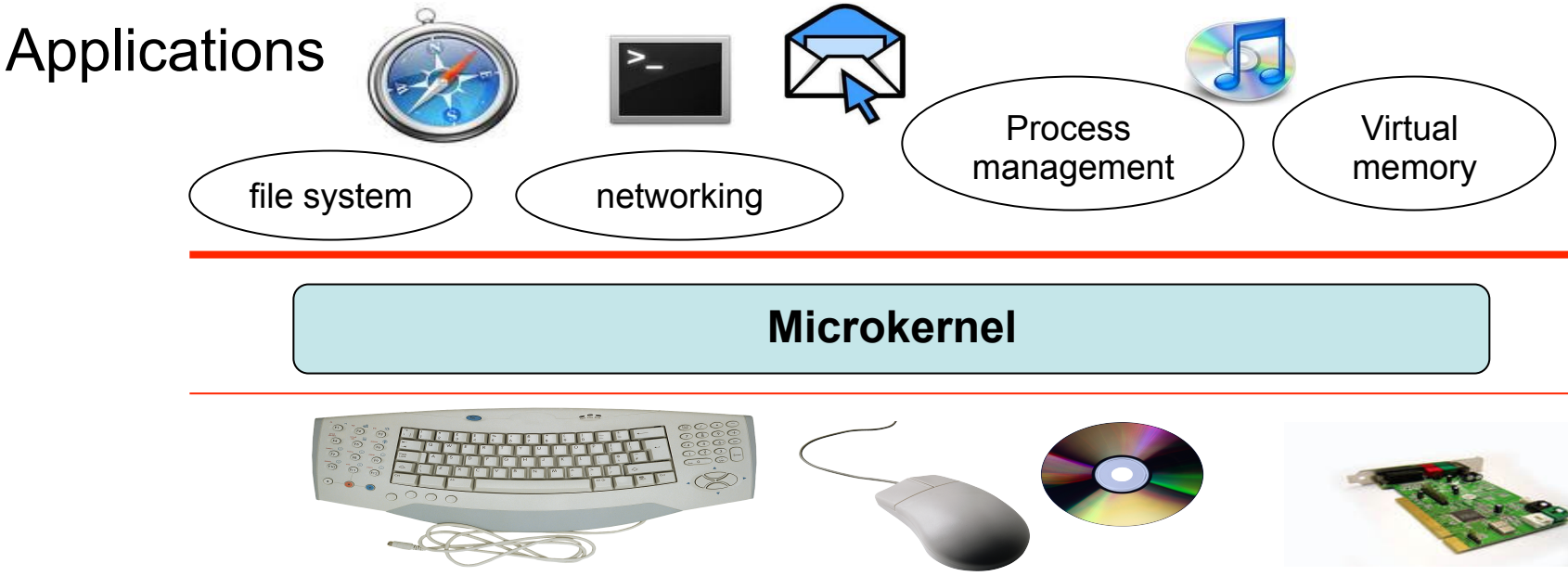
Operating System

file system processes networking

Device drivers virtual memory

Microkernels (late 80's and on)

- Put as little of OS as possible in privileged mode (the microkernel).
- Implement most core OS services as user-level servers.
  - Only microkernel really knows about hardware
  - File system, device drivers, virtual memory all implemented in unprivileged servers.
  - Must use IPC (interprocess communication) to communicate among different servers.

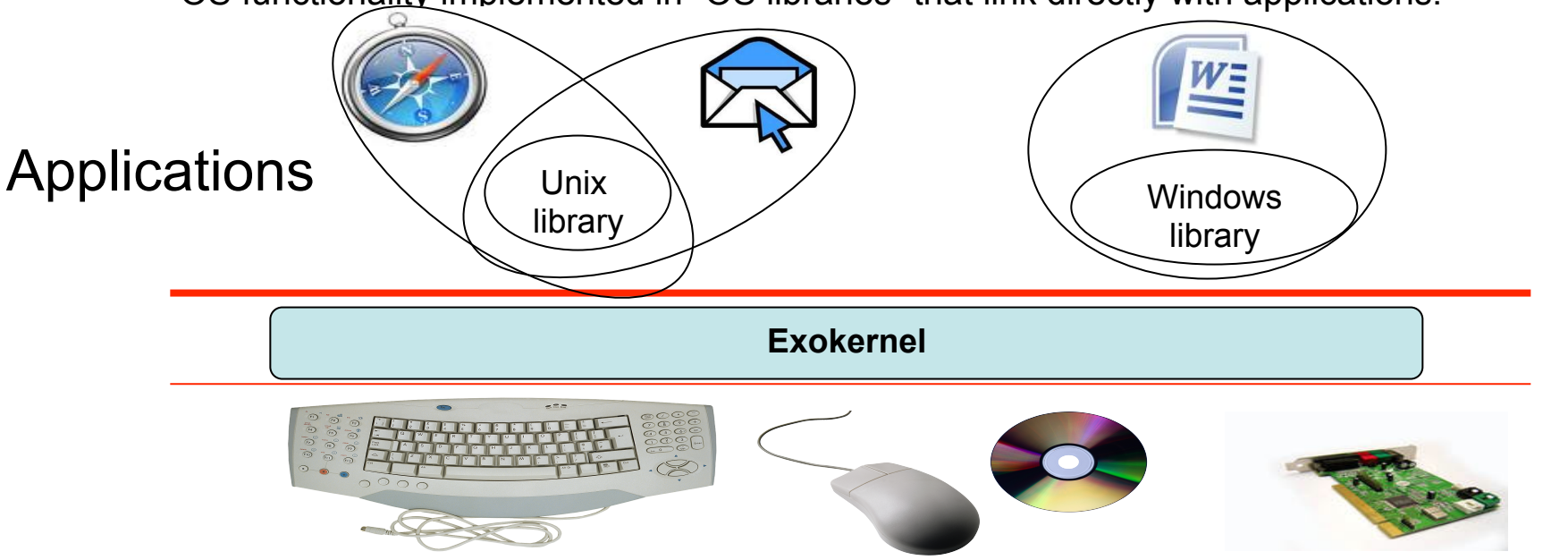


Microkernels: Past and Present

- Much research and debate in late 80's early 90's
  - Pioneering effort in Mach (CMU).
  - Real goal was a new OS that could run UNIX applications.
  - Huge debates over microkernel versus monolithic kernel.
- Windows NT used "modified microkernel"
  - Mostly monolithic
  - Different APIs are user-level services (DOS, Win3.1, Win32, POSIX)
- Mac OS X started as a hybrid architecture, although overtime it has become increasingly a traditional, monolithic architecture.
- Secure Microkernel Project (seL4)
  - Builds on the L4 microkernel to create a small, secure kernel.
  - Provides mechanisms to enforce security guarantees at the OS and application levels.

Exo-Kernels (1995-2000)

- Take microkernels to the extreme.
- Rather than export OS abstractions from kernel, export hardware more directly.
  - Lots of research effort in designing interfaces for exporting hardware so it can be safely multiplexed.
  - Interesting results in safe disk sharing
- OS functionality implemented in “OS libraries” that link directly with applications.



Extensible operating systems

- Mid to late 90's: Lots of research in how to add functionality to the operating system safely.
  - Many fancy mechanisms
    - Expose rich interfaces and use transactions to recover (VINO).
    - Use a safe language (modula3) for extensions (SPIN).
    - Use microkernels and simply write new servers (L4).
    - Binary rewriting...
- In practice:
  - People just wanted to be able to add stuff.
  - Didn't care too much about protection of "stuff."
  - Loadable kernel modules won.