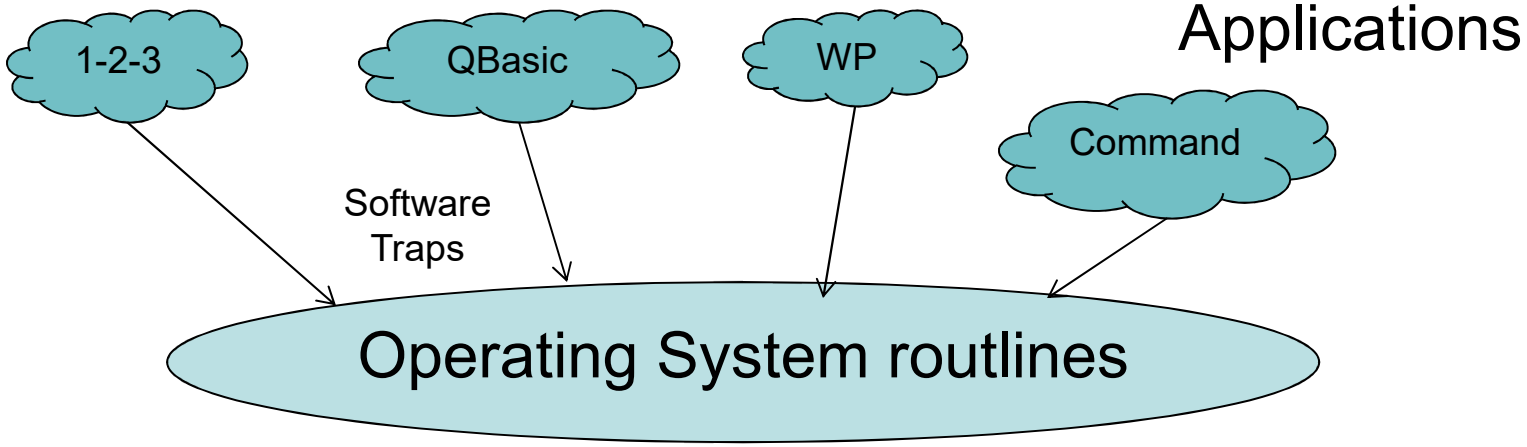
Operating System Architectures: Past, Present, Future

- 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.
  - Pontificate about the future of operating systems.
- 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
  - Exokernels
  - Extensible operating systems
- And then brainstorm about what we might expect to see in the future.

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

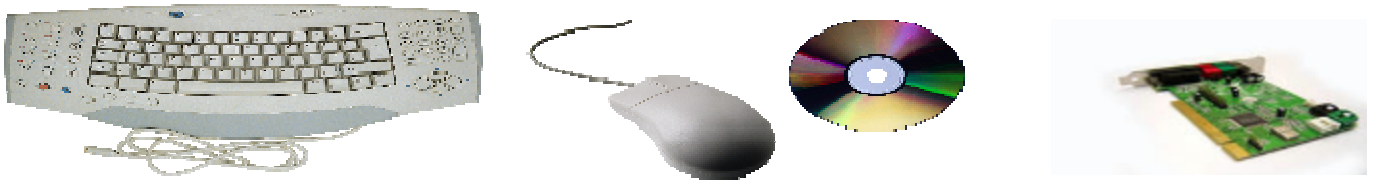
- Traditional architecture
  - Applications have own address space.
  - OS either has own address space or runs in a reserved part of application's address space.
  - Operating system runs in privileged mode; applications run in user mode.

Applications

Operating System

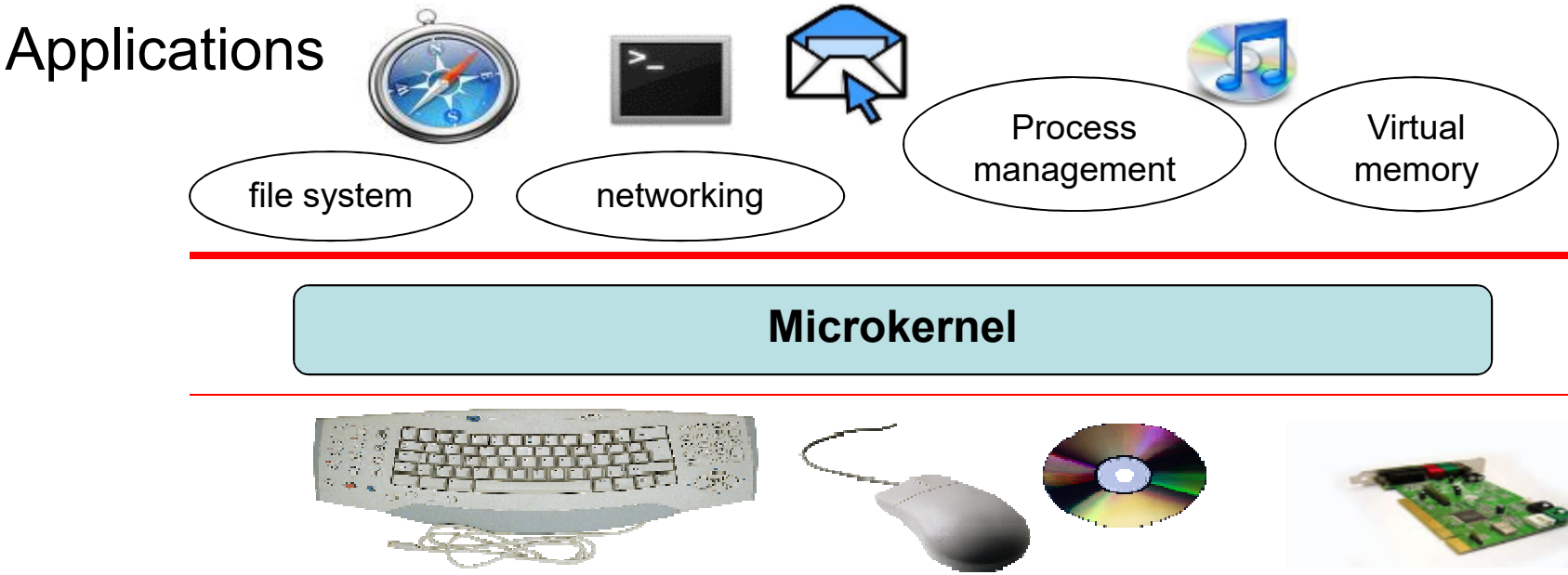
file system processes networking

Device drivers virtual memory



Microkernels (late 80's and on)

- Put as little of the 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 with 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.

Example: Mach VM

- Two interesting features:
  1. Machine independent VM system
  2. VM implemented (mostly) at user-level
- Achieving machine independence
  - Two different abstractions: HW pages and SW pages
    - The VM system is built on SW pages
    - HW pages reflect the actual page structure supported by the platform
  - The only machine dependent piece of the VM system is the pmap module that A) maps between HW and SW pages, and B) interfaces with the underlying HW.
    - Examples: enter mappings, change page permissions, return a virtual-to-physical mapping

Machine Independent virtual memory management for paged uniprocessor and multiprocessor architectures, Rashid et al, 1987.

Structure of a Mach pmap

- pmap structure – represents the HW-specific VM system (like the sfs_fs struct represents the file system specific representation of a file system).
- Implementing a VM functions:
  - Machine independent VM system allocates and frees pages (and references them in address spaces) in units of SW pages.
  - The pmap functions have to set PTEs (or TLB entries) in units of HW pages.
  - Let's imagine that we have 4 KB HW pages and 8 KB software pages.

Implement: pmap_remove_range

```
pmap_remove_range(pmap, start_addr, end_addr) {
```

What might you want to KASSERT?

What is the overall structure of this routine?

```
}
```

Implement: pmap_remove_range

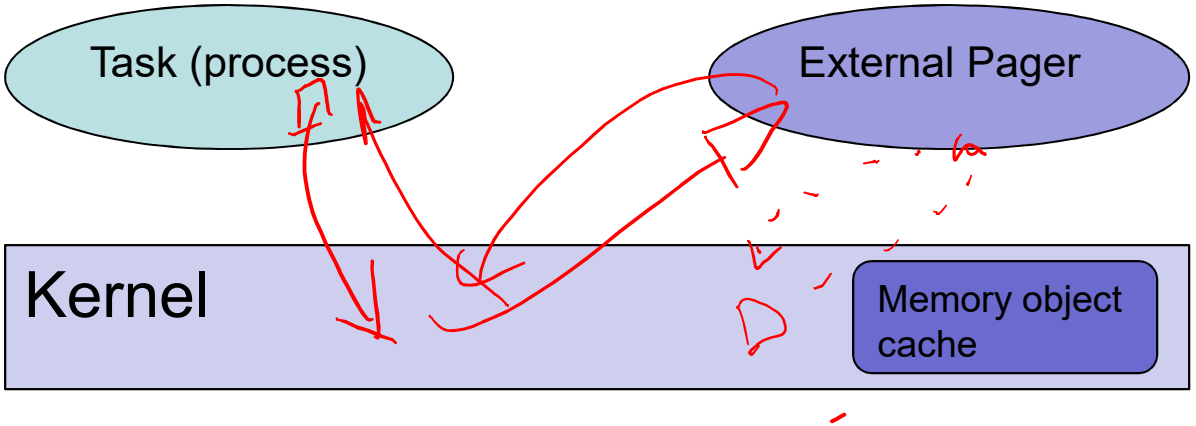
```
pmap_remove_range(pmap, start_addr, end_addr) {
   KASSERT(IS_SW_PAGE_ALIGNED(start_addr));
   KASSERT(IS_SW_PAGE_ALIGNED(end_addr));

   while (start_addr < end_addr) {
      hw_unmap(pmap, start_addr);
      start_addr += HW_PAGE_SIZE;
   }

}
```

VM at User Level

- In Mach, an address space is composed of a collection of memory objects (think segments of your MIPS address spaces).
  - To a first approximation, a memory object is represented by an element in a linked list and the corresponding page table for the object.
- A user-level process, called a pager, is responsible for moving the contents of a memory object in and out of memory.
- Kernel communicates with pagers via messages.



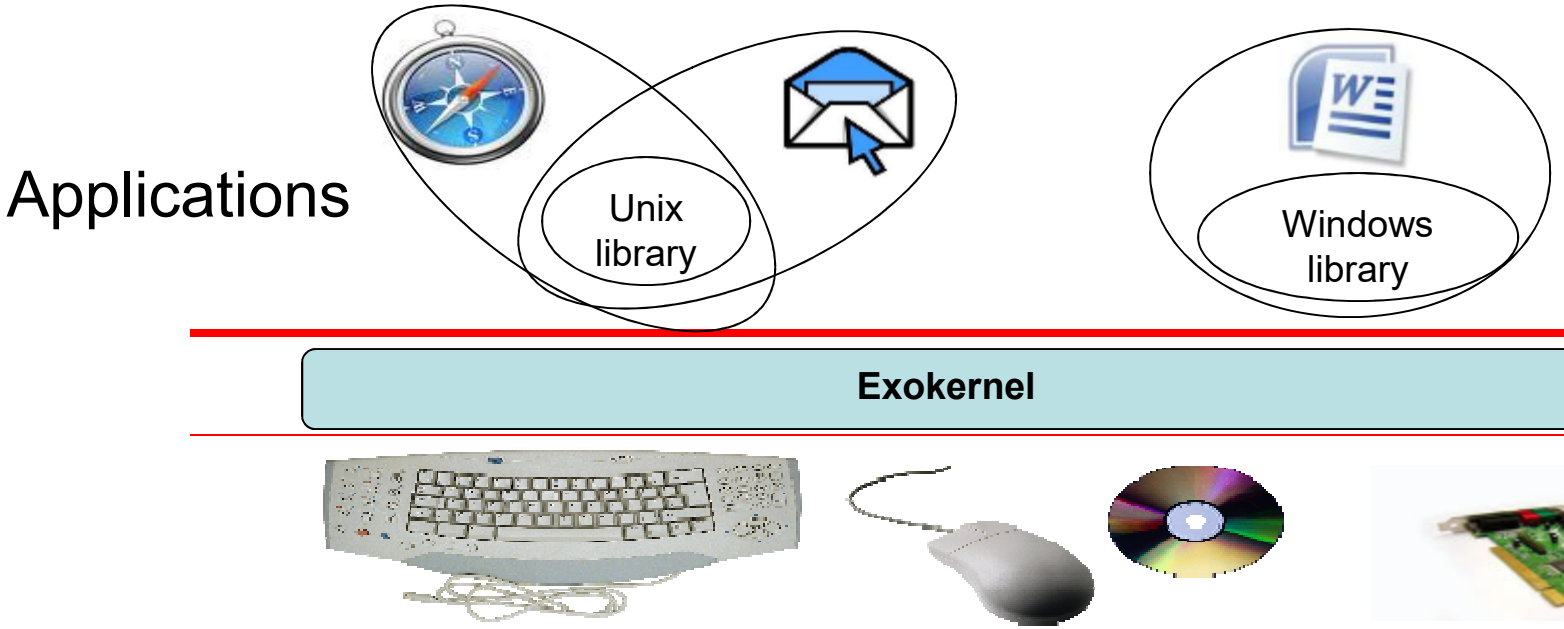
Single-Address Space Operating Systems

- All processes share (and the OS) share a global 64-bit virtual address space.
  - Enhance sharing
  - Simplify integration
  - Improve reliability and performance
- Key idea: Separate protection and addressing.
  - Argue that address spaces are big, thick walls that make sharing hard (i.e., pointers have no meaning outside the program that created it).
  - Results in a lot of copying of data.
- The Opal approach:
  - Address space composed of segments.
  - Once allocated, a segment's VA never changes.
  - Protection domain is the unit of protection; a thread executes in a specific protection domain.
  - Protection domains implemented by capabilities (256-bit references that grant a thread access and rights to a segment via attach)
- Opal implemented as a single server on top of Mach (segments == memory objects).

Sharing and Protection in a Single-address Space Operating System, Chase et. al, 1994.

Exokernels (1995-2000)

- Take microkernels to the extreme.
- Rather than export OS abstractions from kernel, export hardware more directly.
  - Research addressed designing safe interfaces for exporting hardware.
  - Interesting results in safe disk sharing
- OS functionality implemented in “OS libraries” that link directly with applications.



The Exokernel: An Operating System Architecture for Application-Level Resource Management, Engler et. al, 1995

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.

Extensibility, Safety, and Performance in the SPIN Operating System, Bershad et. al., 1995
Dealing with Disaster: Surviving Misbehaved Kernel Extensions, Seltzer, et. al., 1996

Virtual Machines

- Re-inventing operating systems all over again!
- In the 1960's, IBM developed a family of machines (System/360, System/370) and an operating system called VM/360.
  - Enable multiple users to share a single machine by giving each user his/her own machine.
  - Real hardware partitioned into per-"machine" components:
    - Disk
    - Processors
    - Memory
  - Compared to operating systems, it's just a different way of sharing resources.

	Conventional OS	**Virtual Machine Monitor**
Share CPU	Processes	(virtual) machines
Share disk	Global namespace	virtual disk and own file system
Share memory	Virtual memory	Divided among machines