上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

OS Structure

Simplify the kernel

Yubin Xia

Previously on CSP

- Why learn system? For fun and profit
  - Case: guess a password by timing attack
  - Case: guess a PIN by motion sensor
- KISS: worse is better
  - Case: interrupt during system call, return error to user?
  - Case: MapReduce. A major step backwards?
- KISS: the price
  - Case: ShellShock! Text stream as interface

Previously on CSP

- KISS in kernel
  - Case: micro-kernel VS. monolithic kernel
  - Case: Exokernel

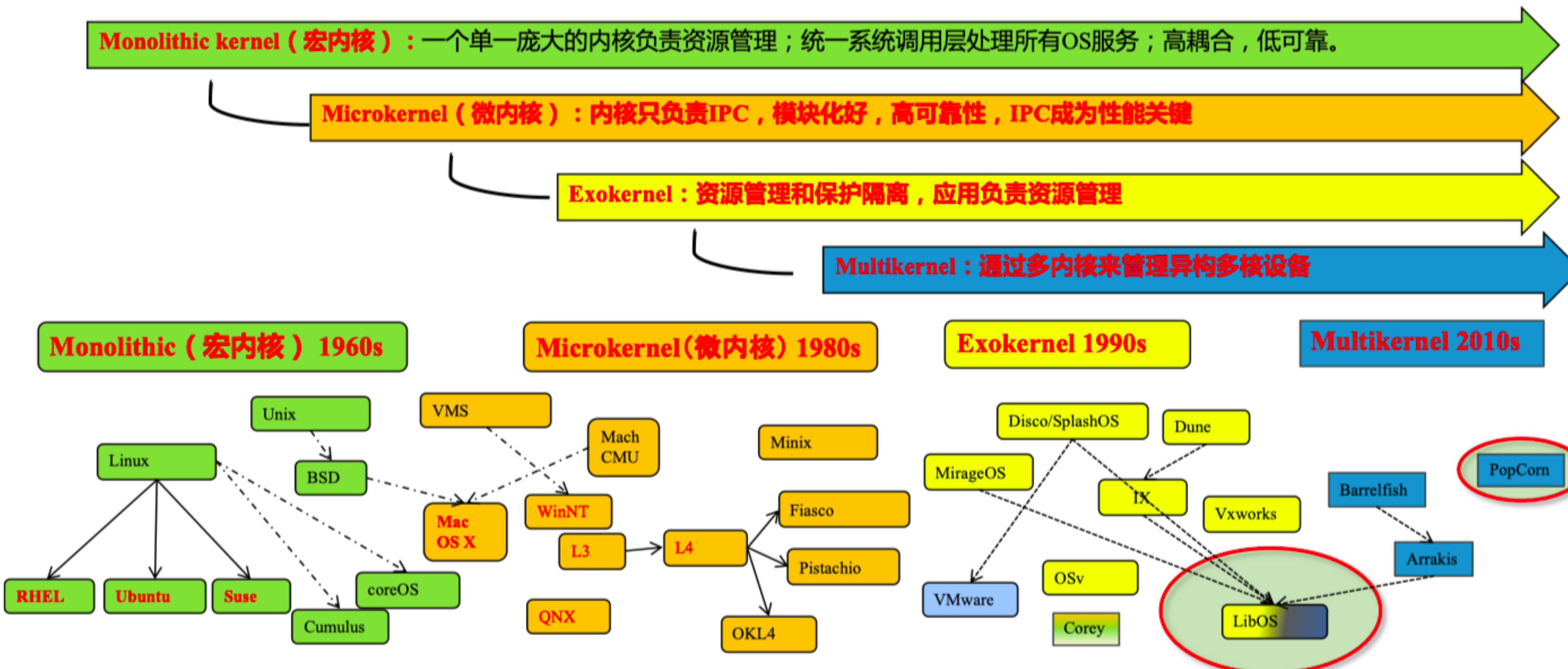
Vasa: Sink on the first sail

Goals of Operating System

- From user
  - Easy to use
  - Easy to learn
  - Full-fledged
  - Security
  - Smooth performance
  - …

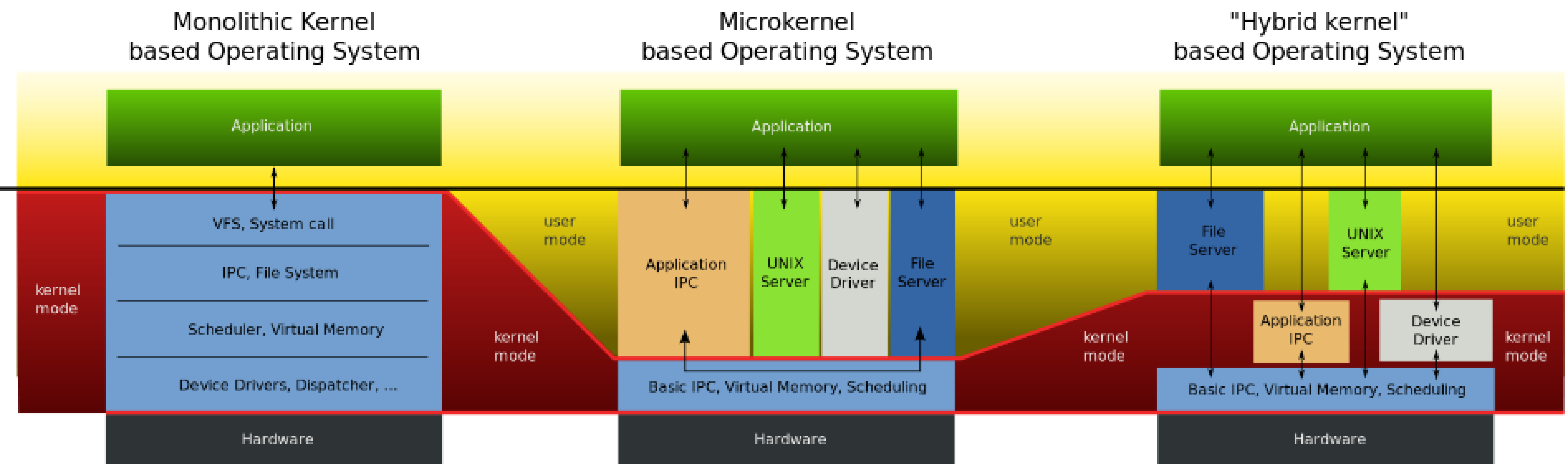
- From system
  – Easy to design and impl.
  – Easy to maintain
  – Flexibility
  – Dependability
  – Efficiency
  – …

Operating System Evolving

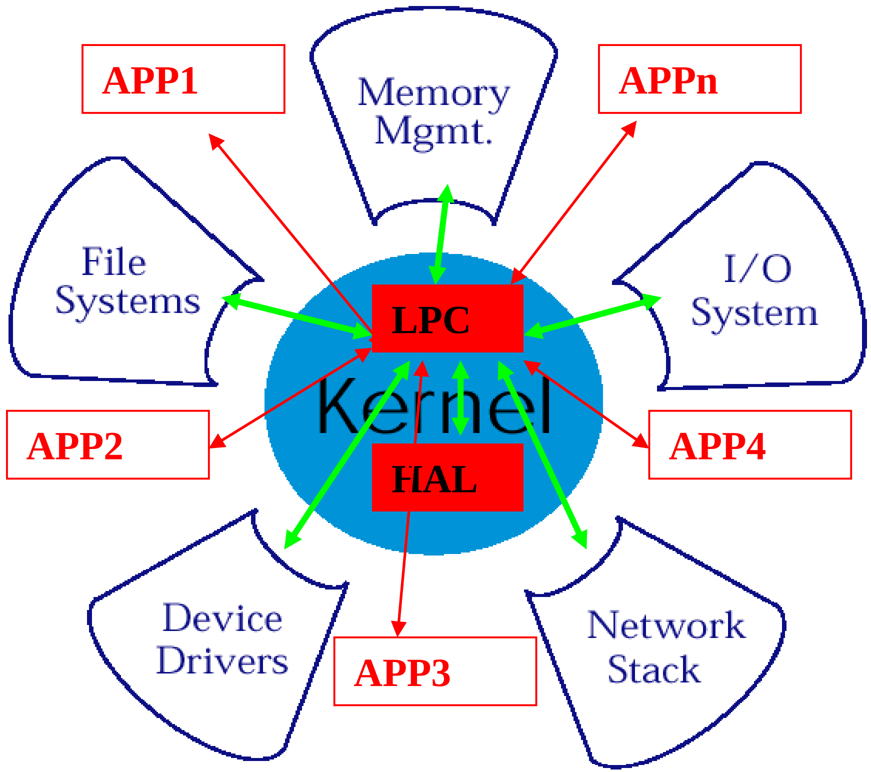


EXOKERNEL ON X86: XOK & EXOS

Comparison


Monolithic Kernel
based Operating System
Microkernel
based Operating System
"Hybrid kernel"
based Operating System
Application
VFS, System call
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
Hardware
kernel
mode
user
mode
Application
Application
IPC
UNIX
Server
Device
Driver
File
Server
Basic IPC, Virtual Memory, Scheduling
Hardware
kernel
mode
user
mode
Application
File
Server
UNIX
Server
Application
IPC
Device
Driver
Basic IPC, Virtual Memory, Scheduling
Hardware
kernel
mode
user
mode

Micro-Kernel Structure & Instances

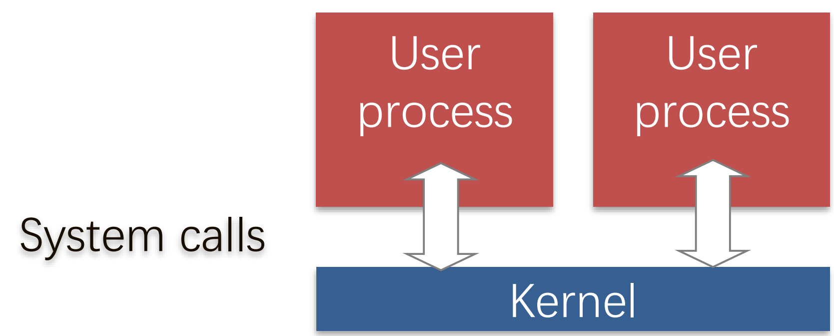


Traditional OS

- Only privileged servers and the kernel can manage system resources
- Both resource management & protection are done by kernel
  - Centralized control
- Untrusted applications are limited to the interface
  - Limited functionality
  - Hurt application performance
  - Hide information (page fault etc.)

Traditional OS

- An interface designed to accommodate **every** application must anticipate all possible needs
  - Flawed!
- Solution:
  - Allow applications enough control over resources by **separating protection from management**
  - "Exokernel" does this



Exokernel

- Separates resource management from protection
  - Normal kernel does both
- **Kernel**: only protect the resources
- **Application**: manage the resources
  - Virtual memory, file system etc., are in application libraries
  - Gives untrusted software as much control over hardware and software resources as possible
  - Specialized applications can gain high performance without sacrificing the unmodified UNIX program

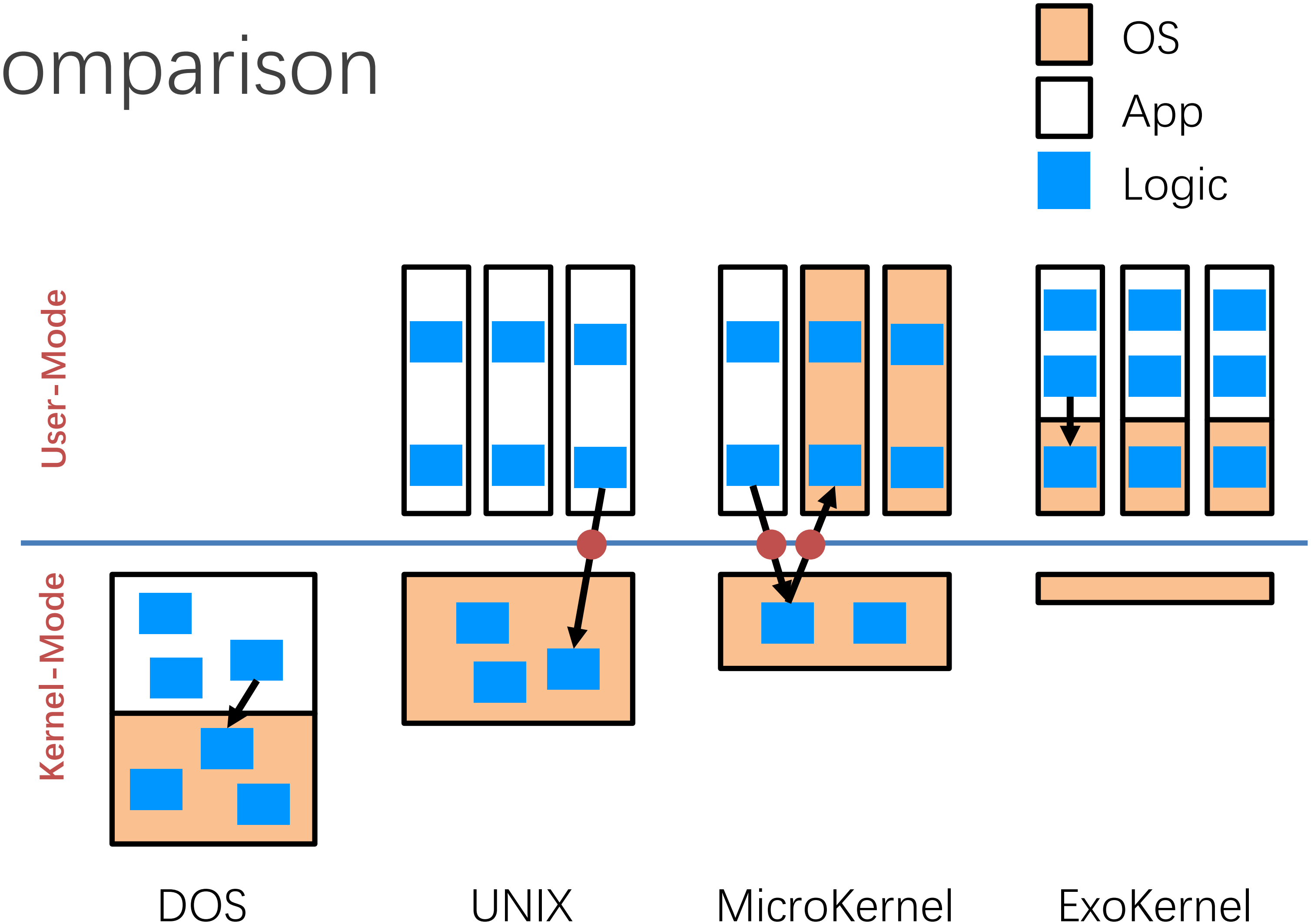
Specialization

- Application specialization
  - E.g., millions of apps on phone
- OS specialization
  - E.g., for server, desktop, phone, IoT devices, …
- Hardware specialization
  - E.g., CPU, GPU, TPU, NPU, DSP, SSE, …

OS as a Library

- "Unix as a library"
  - Can implement traditional OS abstraction
  - Most application programs will be linked to libOSes of their choices instead of communicating with the Exokernel
- Unprivileged libraries
  - Can be modified or replaced at will
- Different libOSes can coexist on the top of same Exokernel
  - Allows system to emulate behaviors of several conventional OSs

Comparison



OS
App
Logic
User-Mode
Kernel-Mode
DOS
UNIX
MicroKernel
ExoKernel

Exokernel Design Challenge

Kernel's new role

- Tracking ownership of resources
- Ensuring resource protection
- Revoking resource access

- Three techniques
  - **Secure binding**
  - **Visible revocation**
  - **Abort protocol**

Secure Binding

It is a protection mechanism that decouples authorization from actual use of a resource

Can improve performance

- The protection checks involved in enforcing a secure binding are expressed in terms of simple operations that the kernel can implement quickly
- A secure binding performs authorization only at bind time, which allows management to be decoupled from protection

Three techniques

- Hardware mechanism, software caching, and downloading application code

Visible Resource Revocation

A way to reclaim resources and break their (application & resources) secure binding

An exokernel uses visible revocation for most resources

- Traditionally, OS have performed revocation invisibly, de-allocating resources without application involvement

Dialogue between an exokernel and a library OS

- Library OS should organize resource lists

The Abort Protocol

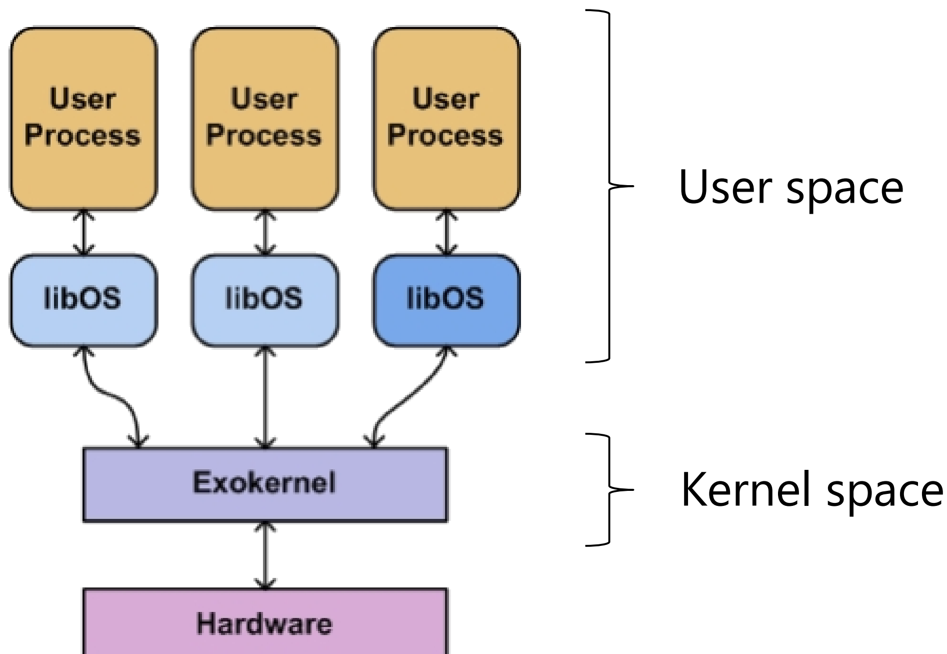
An exokernel must also be able to take resources from library operating systems that fail to respond satisfactorily to revocation requests

If a library OS fails to respond quickly, the secure bindings need to be broken "**by force**"

An exokernel simply breaks all secure bindings to the resource and informs the library operating system

Exokernel: Example on Disk Management

- Application manages its disk-block cache and kernel allows cached pages to be shared securely between applications



Extensible OS

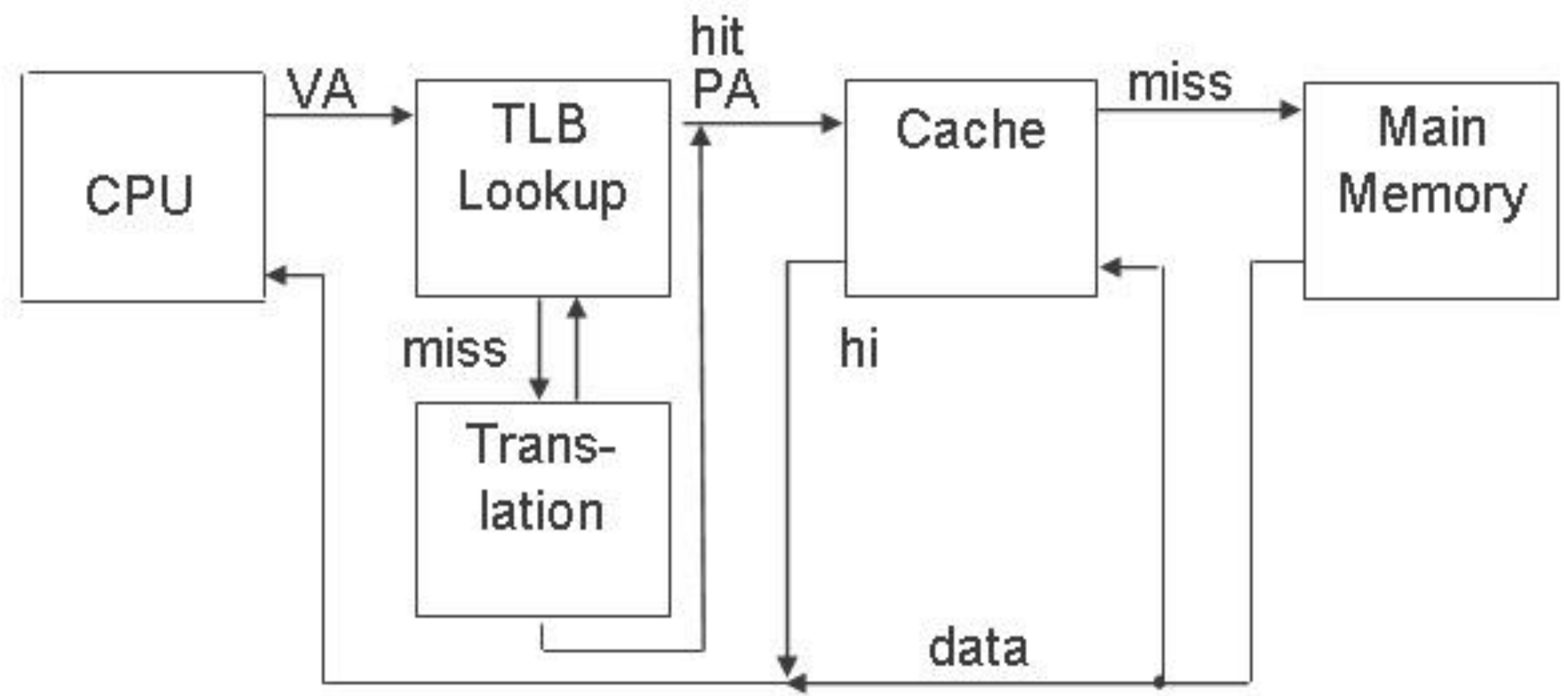
- Extensibility lets new functionalities to be included in the OS
- Let applications safely modify system behavior for its own need
- Different approaches to extensible OS:
  - Exokernel (MIT)
  - SPIN (UW)
  - VINO (Harvard)
  - L4 (IBM)
  - Fluke/OSKit (Utah)

Exokernel's Memory Protection

- Using **software TLB**
  - ExoKernel manages the TLB
  - libOS manages the page table
- Using **page table** only
  - libOS can read but not arbitrarily write the page table
  - Page table itself is set as read-only
  - Modification or changing of page table will trap to ExoKernel

Software managed TLB: when there is a TLB miss, the CPU will trigger an exception and let the kernel to fill TLB, instead of doing so in hardware directly.

Review: TLB & Cache



Exokernel Principles

- **Separate resource protection and management**
  - Exokernel and libOSes
  - Minimum resource management as required by protection (allocation, revocation etc.)
- **Expose allocation**
  - Applications allocate resources
  - Kernel allows the allocation requests

Exokernel Principles

- **Expose names**
  - Exokernels use physical names wherever possible
- **Expose revocation**
  - Let application choose which instance of resource is to give up
- **Expose information**
  - Expose all system info and collect data that application can not easily derive locally

Exokernel: Benefits

- Exposing kernel data structure
  - Can be accessed without system call overhead
- Flexibility
  - libOSes can be modified and debugged more easily
  - "Edit, compile, debug" cycle of applications is considerably faster than the "edit, compile, reboot, debug" cycle of kernel
- Performance
  - Aggressive applications may gain speed up to **10x**

Exokernel: Drawbacks

- Exokernel interface design is not simple
  - Most of the major Exokernel interfaces have gone through multiple designs over several years
- The ease of creation and mixing of libOSes could lead to code messes
  - nightmare for maintenance coders and system administrators
- It is theoretically possible to provide libOSes that enable applications to run simultaneously on the same system, that would also mean different look & feels for each of them
  - Different libOSes may have varying levels of compatibility and interoperability
  - Poorly chosen abstractions may cause lose of information
  - Self-paging libOSes is difficult

Criticisms of Exokernel

- Customer-Support:
  - "*Extensibility has its problems. For example, it makes the customer-support issues a lot more complicated, because you no longer know which OS each of your customers is running*" (Milojicic, 1999).

Linux as a LibOS?

- Use Linux as a libOS or unikernel
  - E.g., LKL - Linux kernel library (https://github.com/lkl)
  - Change all the system calls to function calls
  - Offer certain compatibility, avoid re-implementation
- New problems
  - Is Linux suitable to serve as a libOS/unikernel?
  - How to handle fork()?
  - Still in research stage

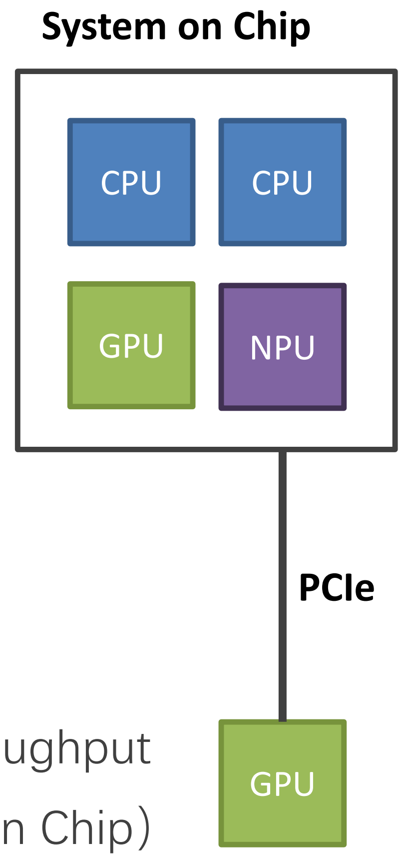
Summary

- Exokernel Architecture:
  - Goal: safe application control of all resources
  - How: by separating resource management from protection
- Results found promising:
  - Unmodified applications run same or **4x** better
  - Customized applications can run up to **8x** better
  - Global performance is similarly good like UNIX

MULTI-KERNEL

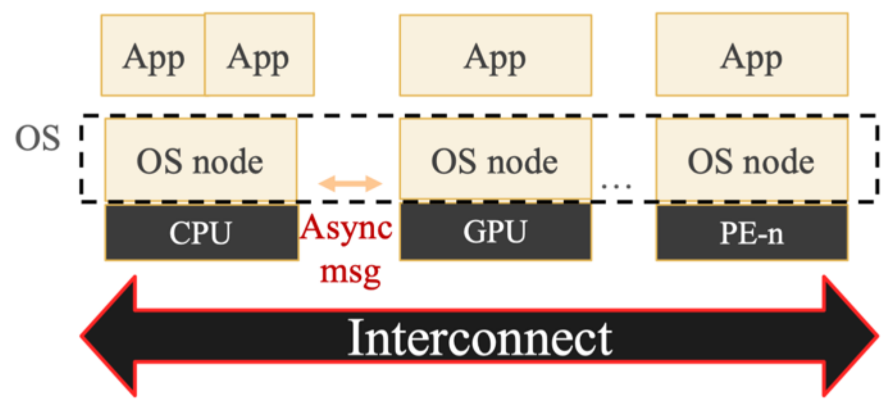
Multikernel

- Multicore and Heterogenous cores
  - OS maintains many shared status
    - Cache coherence becomes harder
    - Scalability issues: more cores, less performance
  - More smart devices like GPU
    - Each device has its own core
    - Connected with PCIe, long latency and poor throughput
    - Connected with system bus, e.g., SoC （System on Chip）



The Design of Multikernel

- Multikernel's idea
  - The default configuration is partitioning, not sharing
  - Maintaining multiple copies, instead of one
  - Explicit communication between different cores
- Multikernel design
  - One kernel per core
    - Including CPU, GPU etc.
  - OS works as a distributed system
  - Applications still run on OS

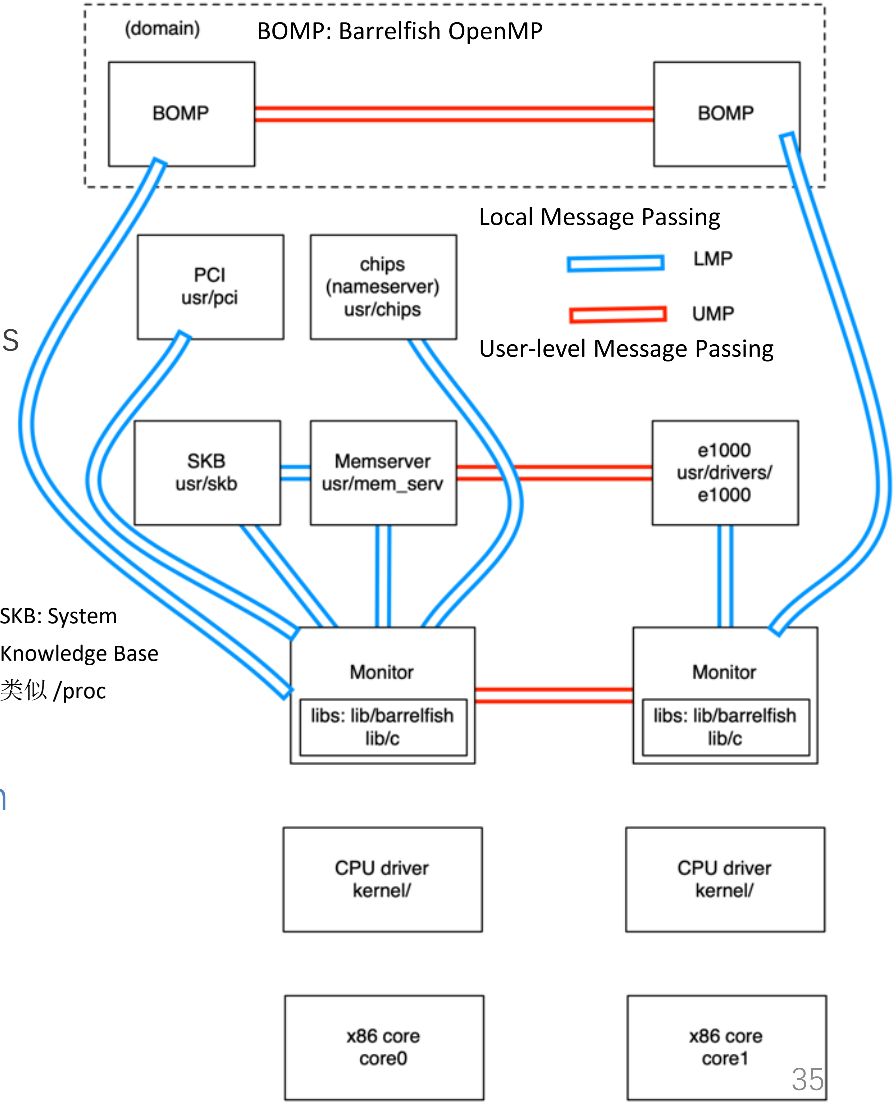


Barrelfish Multikernel

- Barrelfish OS
  - From ETH Zurich and MSR
  - Support heterogenous CPU
  - Use message abstraction between CPU and devices
  - Around 10,000 C code and 500 assembly

Barrelfish Structure

- Kernel: Per core
  - Like CPU driver, adapt to different CPUs
  - System call, interrupt/exception
  - Event-driven, single-thread
  - Uninterruptable
  - Schedules and runs the dispatcher
- Dispatcher (like thread)
  - Multiple dispatcher becomes a Domain
- Domain (like process)

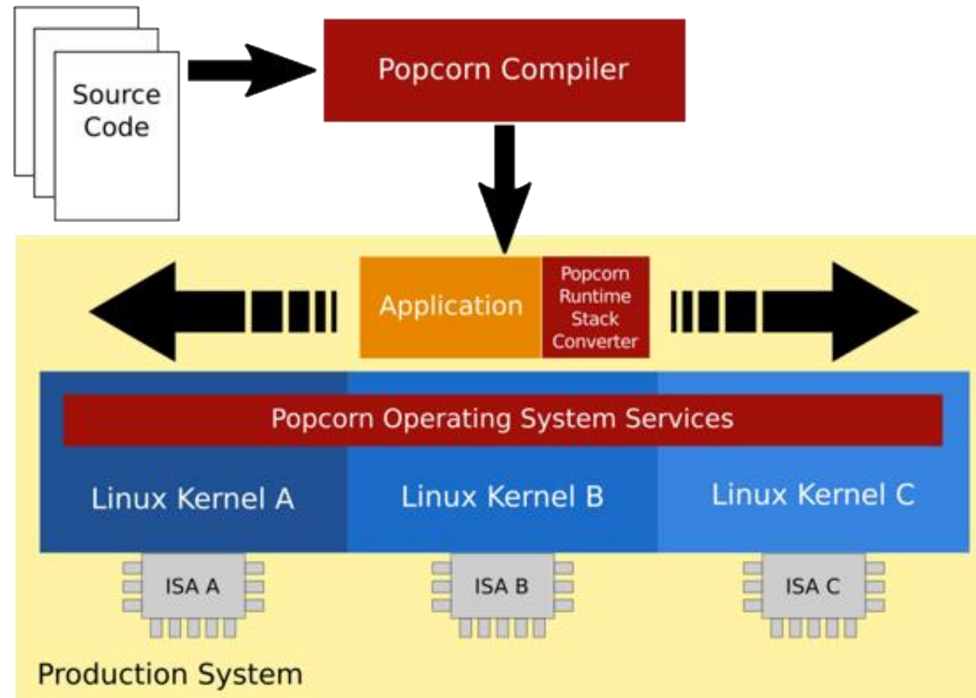


SKB: System
Knowledge Base
类似 /proc

http://www.barrelfish.org/documentation.html

Popcorn Linux

- Support multiple architectures
  - ARM, x86, etc.
- Multiple Linux kernel replicas
  - With the same code
  - One replica per ISA
  - Offer OS services at the same time



http://www.popcornlinux.org/

Flexible System Call Scheduling with Exception-Less System Calls, OSDI'10

FLEX-SC

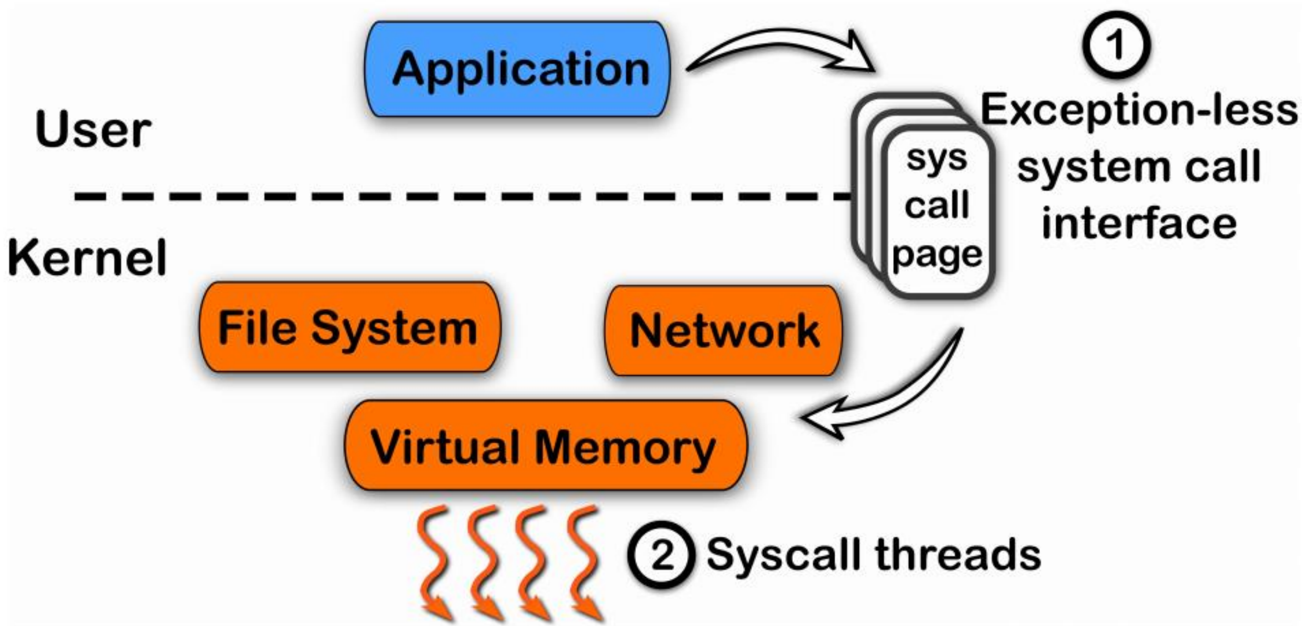
The Motivation

- How to further reduce the latency of syscall?
  - Not only for gettimeofday()
- Where does the latency come from?
  - Mostly for the state switch
    - Save and restore states
    - Privilege checking
  - Cache pollution
- Could we do syscall without state switching?

Flexible System Call

- New syscall mechanism – Flexible System Call
  - Introduce **system call page** that is shared by user & kernel
  - User threads can **push** the system call requests into the system call page
  - kernel threads will **poll** the system call requests out the system call page
- Exception-less syscall
  - Remove synchronicity by decoupling invocation from execution

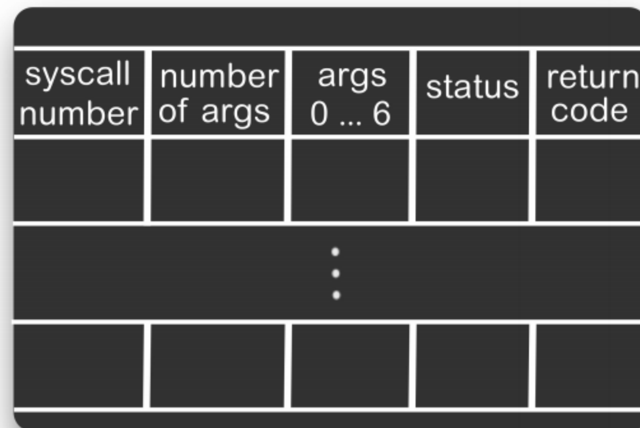
Another Way for System Call



Exception-less System Call

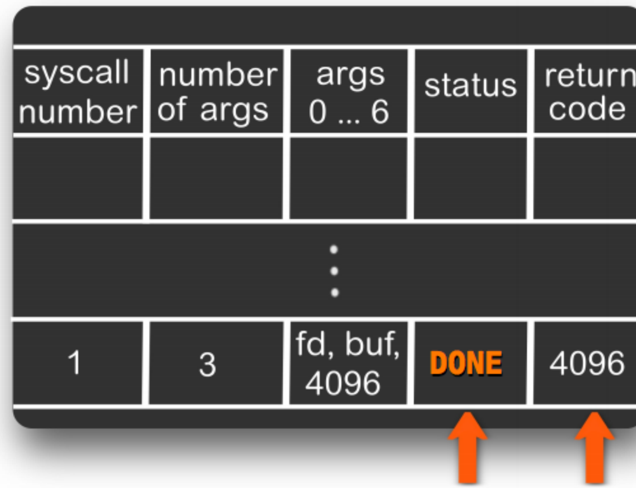
```
write(fd, buf, 4096);
```

```
entry = free_syscall_entry();

/* write syscall */
entry->syscall = 1;
entry->num_args = 3;
entry->args[0] = fd;
entry->args[1] = buf;
entry->args[2] = 4096;
entry->status = SUBMIT;

while (entry->status != DONE)
   do_something_else();

return entry->return_code;
```

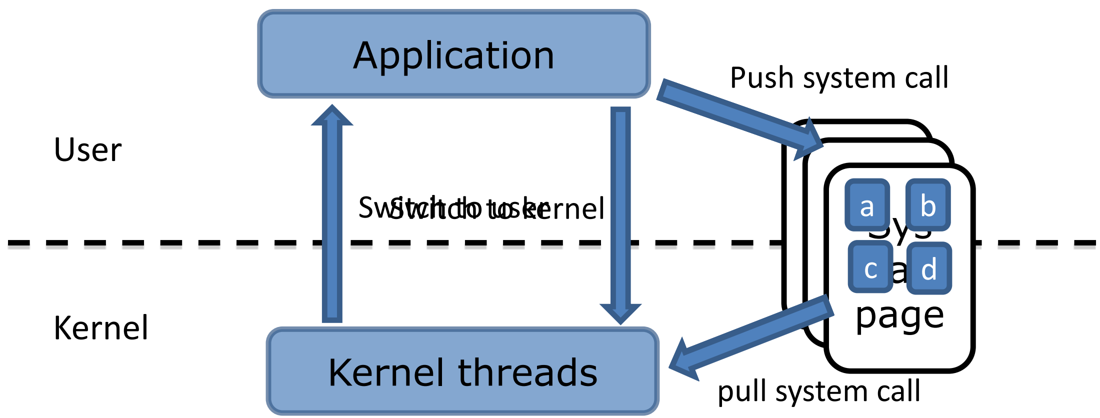


Kernel Fill the Results

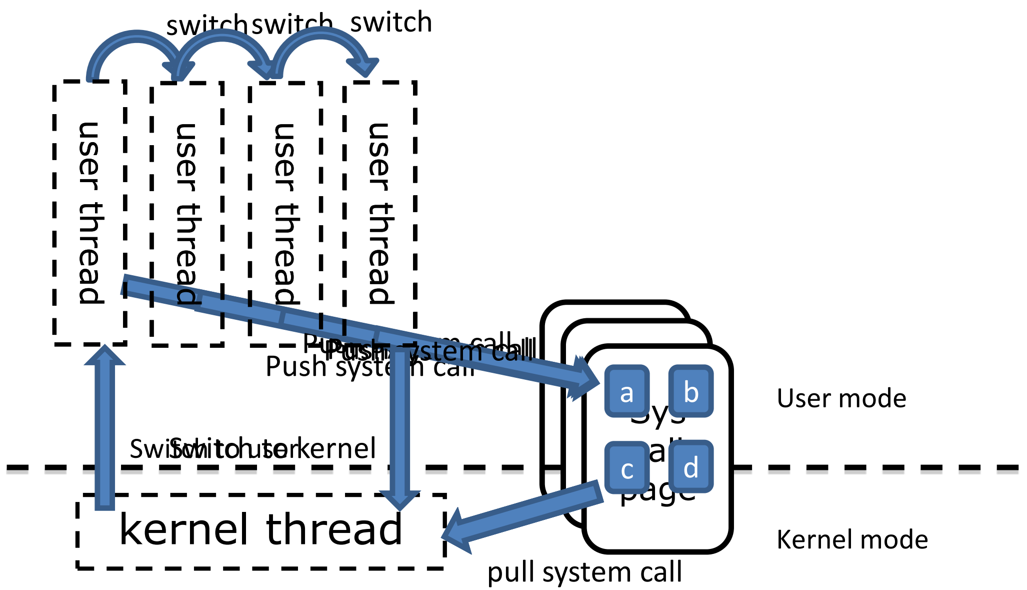
```c
write(fd, buf, 4096);

entry = free_syscall_entry();

/* write syscall */
entry->syscall = 1;
entry->num_args = 3;
entry->args[0] = fd;
entry->args[1] = buf;
entry->args[2] = 4096;
entry->status = SUBMIT;

while (entry->status != DONE)
   do_something_else();

return entry->return_code;
```



On a Single Core: Single Threads



On a Single Core: Multiple Threads



FlexSC: Flexible System Call Scheduling with Exception-Less System Calls

The art of virtualization

CASE: XEN

VMM: A Brief History

- **Virtual Machine Monitor**: a software-abstraction layer that partitions the HW into one or more virtual machines
- 1960s: used for multiplexing the scarce general purpose computing platforms among multiple applications
- 1980s: multitasking OSes + low HW costs
  - Rendered VMMs obsolete
  - Consequently, no hardware support for virtualization in the CPU architectures of the time (e.g., x86)

Why This Revival?

- **Virtual Machine Monitor**: a software-abstraction layer that partition the HW into one or more virtual machines
- 1960s: used for multiplexing the scarce general purpose computing platforms among multiple applications
- 1980s: multitasking OSes + low cost hardware
  - Rendered VMMs obsolete
- 2000s: multitasking OSes + low cost hardware
  - Revived VMMs

Challenges to Build Virtual Machines

- Performance isolation
  - Scheduling priority
  - Memory demand
  - Network traffic
  - Disk accesses
- Support for various OS platforms
- Small performance overhead

Xen's Goals

- Multiplexes resources at the granularity of an entire OS
  - As opposed to process-level multiplexing
  - Price: higher overhead
- Target: 100 virtual OSes per machine

Xen: Approach and Overview

- Conventional approach: **Full virtualization**
  - Cannot access the hardware
  - Problematic for certain privileged instructions (e.g., traps)
  - No real-time guarantees
- Xen: **Para-virtualization**
  - Provides some exposures to the underlying HW
  - Better performance
  - Need modifications to the OS
  - No modifications to applications

Memory Management

- Depending on the hardware supports
    - Software managed TLB
        - Associate address space IDs with TLB tags
        - Allow coexistence of OSes
        - Avoid TLB flushing across OS boundaries

Memory Management

- X86 does not have software managed TLB
  - Xen exists at the top 64MB of every address space
  - Avoid TLB flushing when an guest OS enter/exist Xen
  - Each OS can only map to memory it owns
  - Writes are validated by Xen

CPU

- X86 supports 4 levels of privileges
  - Ring-0 for OS, and ring-3 for applications
  - Xen downgrades the privilege of OSes to ring-1
  - System-call and page-fault handlers registered to Xen
  - "fast handlers" for most exceptions, Xen isn't involved

Device I/O

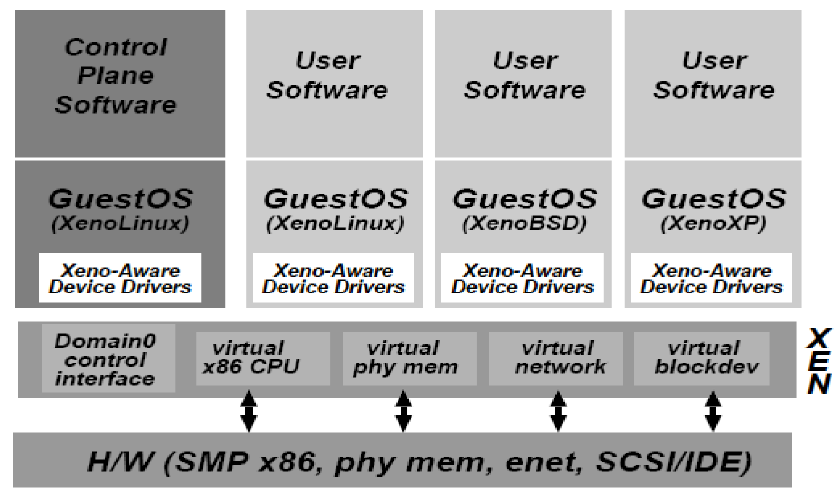
- Xen exposes a set of simple device abstractions
- One driver for each type of device
  - Network
  - Disk
  - Console
  - …

The Cost of Porting an OS to Xen

- Privileged instructions
- Page table access
- Network driver
- Block device driver
- <2% of code-base

Control Management

- Separation of policy and mechanism
- Domain-0 hosts the application-level management software
  - Creation and deletion of virtual network interfaces and block devices



Control Transfer: Hypercalls and Events

- Hypercall: synchronous calls from a domain to Xen
  - Analogous to system calls
- Events: asynchronous notifications from Xen to domains
  - Replace device interrupts