Are VMMs microkernels done right?

- List three ways VMMs are superior to microkernels
  - Include one benchmark class where VMMs “beat” microkernels
- List one piece of computing infrastructure best enabled by VMMs
- List three ways microkernels are superior to VMMs
  - Include one benchmark class where microkernels “beat” VMMs
- List one piece of computing infrastructure best enabled by microkernels
- How similar is microkernels vs VMMs to the RISC vs CISC debate?
- What is liability inversion and why is it bad?

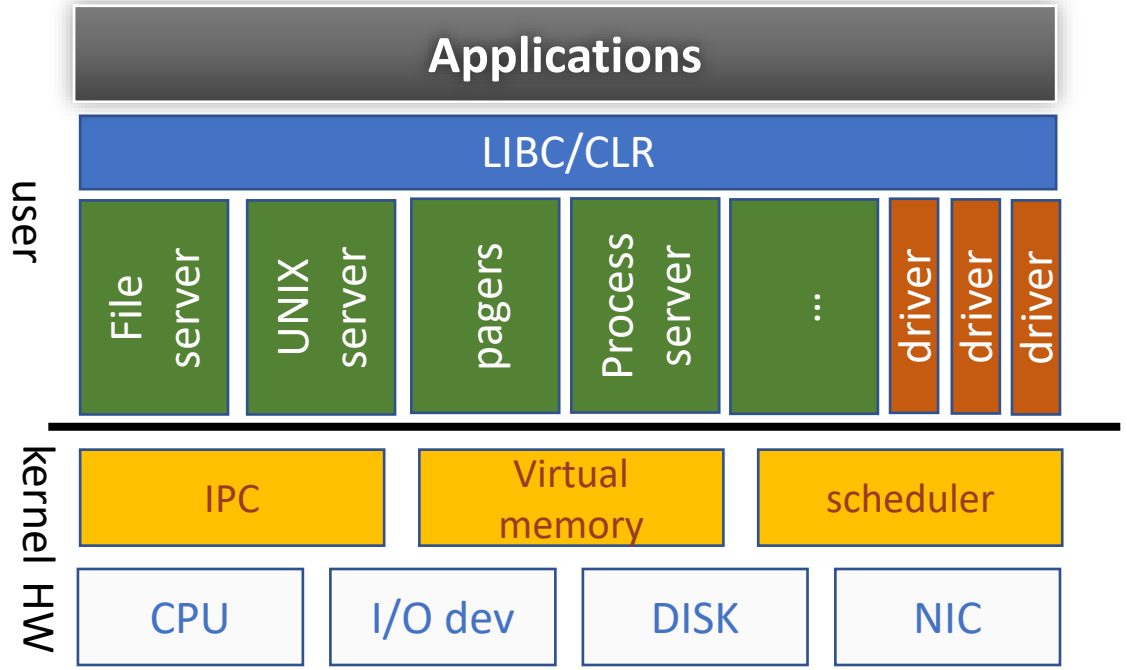
Are VMMs Microkernels Done Right?

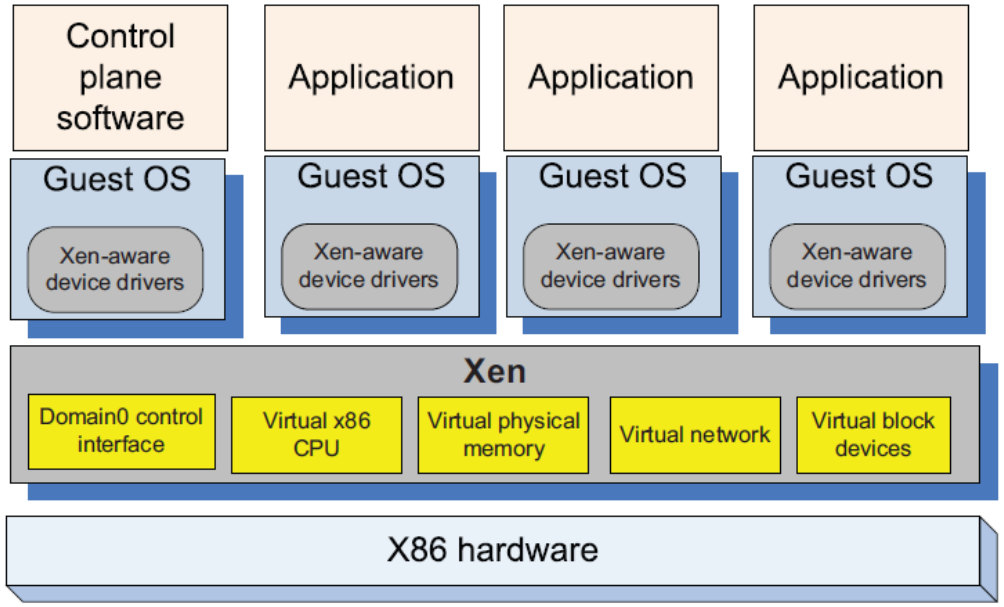
Goldberg [Gol74] defines a virtual machine monitor as

> *"[...] software which transforms the single machine interface into the illusion of many. Each of these interfaces (virtual machines) is an efficient replica of the original computer system, complete with all of the processor instructions [...]."*

Liedtke [Lie96] describes the microkernel approach as

> *"... to minimize the kernel and to implement whatever possible outside of the kernel."*





Microkernel mistakes

- Liability inversion
  - critical system-wide components in user-space
  - kernel can wait on the pager to evict a page before it can proceed.
- System performance depends on IPC performance
  - Not true in a VMM
  - VMM looks more like a device (async control, fixed-format bulk data).
- uKernel→Provides new programming interface
  - or much effort spent in emulation

VMM pros/cons

- VMM Pros
  - *“We have determined that a clear separation between control and data path operations allows us to optimize for the common case...we can perform potentially expensive permission and safety checks at initialization time and then elide validation during more frequent data path operations.”*
    - Similar to secure binding in exokernel.
  - OS is a component. Trend in library OS is to specialize.
    - Is OS as a component a good thing or a compatibility thing?
- VMM mistakes/limitations
  - Are VMMs inherently un-portable across architectures?
  - Do system calls trap?

IPC in microkernels

IPC is the mechanism for *everything*

- kernel-controlled control transfer between protection domains;
- kernel-controlled data transfer between protection domains;
- resource delegation between protection domains
  - amongst mutually distrustful parties

The counter-arguments

- Different goals
  - Faithful emulation vs minimization, privilege separation
- VMMs have many more core primitives
  - IPC only for u-kernel
  - VMMs:
    - synchronous switch between privilege layers (both directions)
    - Async VM → VM communication
    - Resource management, hypercall interface
    - …more
- Xen has liability inversion too (FS example, but dom0 too)
- Xen perf-critical mechanisms are essentially IPC
- L4, DROPS → u-kernel is a *fine* virtual machine

The counter-arguments

- Different goals
  - Faithful emulation vs minimization, privilege separation
- VMMs have many more core primitives
  - IPC only for u-kernel
  - VMMs:
    - synchronous switch between privilege layers (both dire
    - Async VM → VM communication
    - Resource management, hypercall interface
    - …more
- Xen has liability inversion too (FS example, but
- Xen perf-critical mechanisms are essentially IPC
- L4, DROPS → u-kernel is a *fine* virtual machine

Key takeaways:

- Great treatment of tradeoff space
- Many apparently disparate mechanisms boil down to the same thing
- Heiser makes great points. VMs still seem more prevalent though. (L4 pretty popular though)