Legacy Reuse and Improved System Dependability via Virtual Machines

Joshua LeVasseur

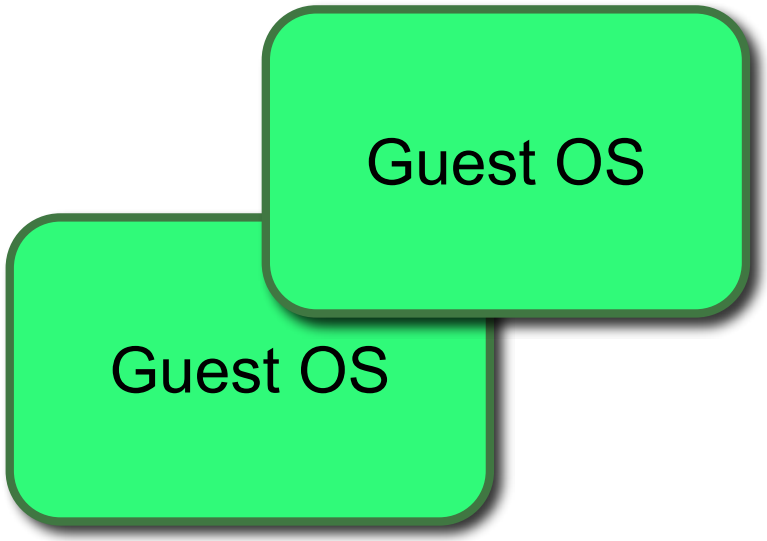
University of Karlsruhe, Germany

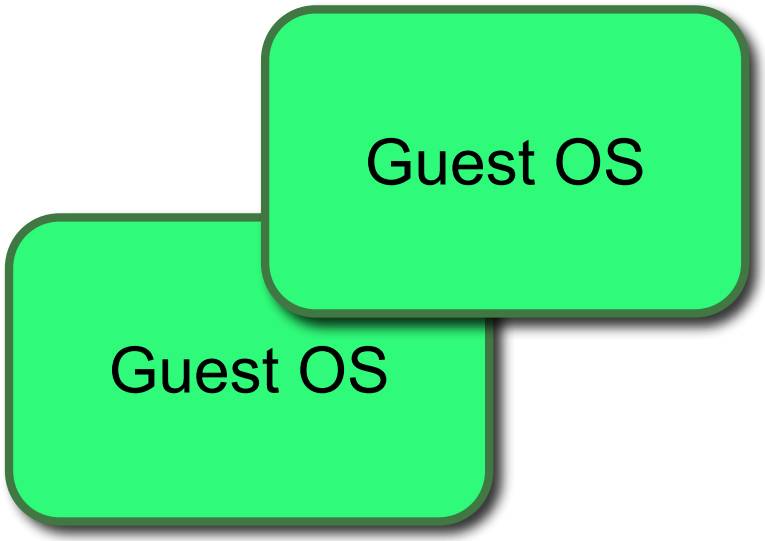
July 8, 2005

Virtual Machine (VM)

- A software duplicate of the hardware
  - Indistinguishable from real hardware
    - Except for timing
- Statistically, most instructions execute directly on real CPU
  - Faster than full emulation

Basic VM Structure



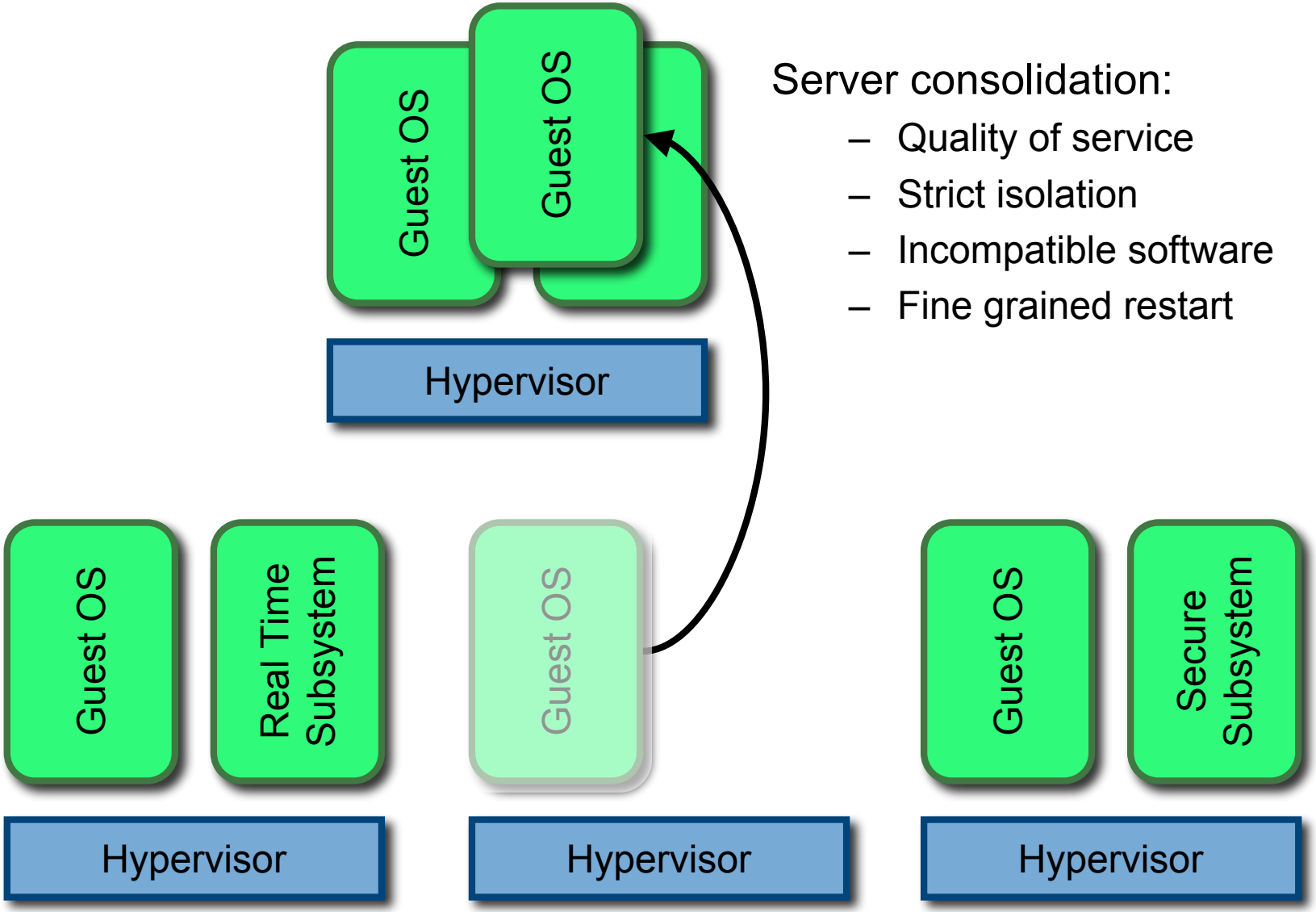


Hypervisor + Virtual Machine Monitor

User-level VMM

L4 Microkernel

Uses of VMs



Server consolidation:

- Quality of service
- Strict isolation
- Incompatible software
- Fine grained restart

Legacy Reuse

- VMs enhance legacy code
- Modular encapsulation
  - Guest OS is nearly a black box
  - Well defined interface (but sometimes buggy)
  - Communication with the black box via platform interfaces (network, disk, …)
- Traditionally, enhancement is via loadable kernel modules

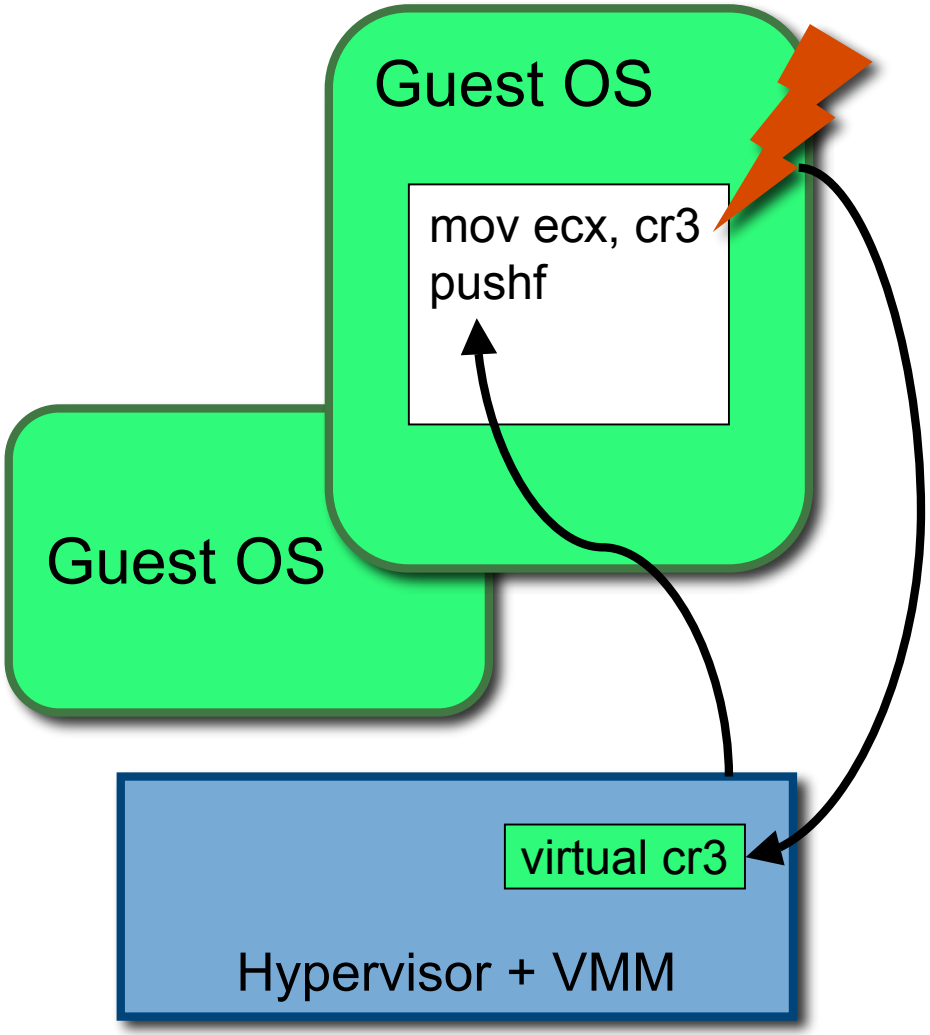
Current OS design is flawed.

The VM is a hack to fix the problems.

Virtualization Definitions

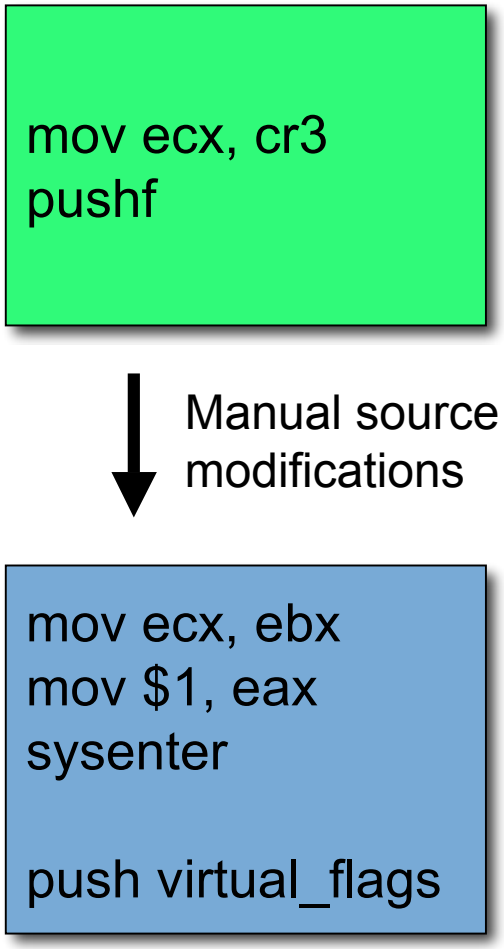
- Sensitive instruction:
  - Destroys the illusion of virtualization
- Innocuous instruction:
  - Safe to execute within a VM
- Privileged instruction:
  - No side effects when executed at user level; raises a fault
- Virtualizable ISA: all sensitive instructions are privileged

Pure Virtualization



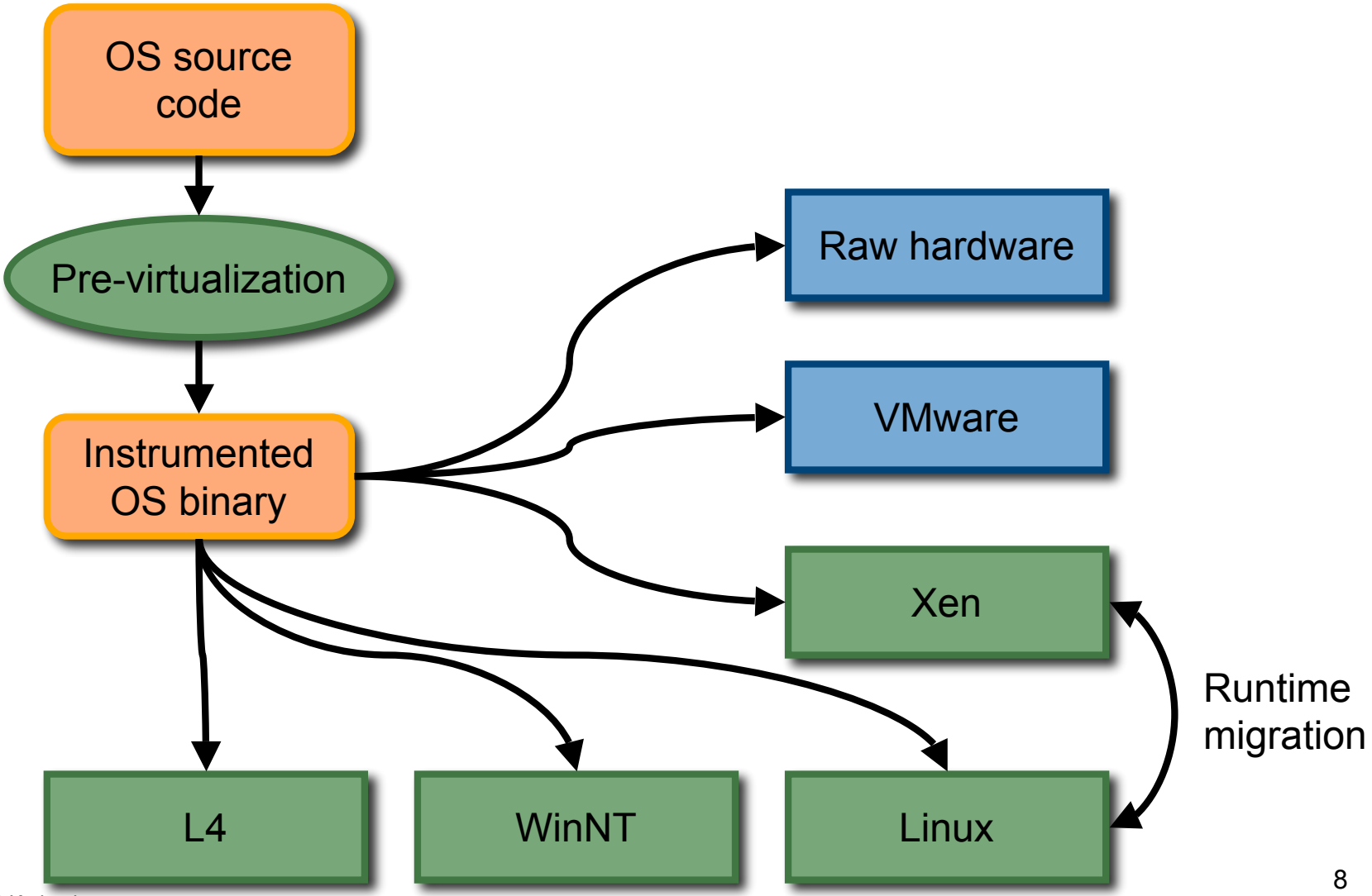
- Problems:
  - Trapping is costly (cycles, pipe flush)
  - x86 isn't fully virtualizable
- VMware's solution:
  - Dynamic code rewriting
  - Difficult

Para-virtualization

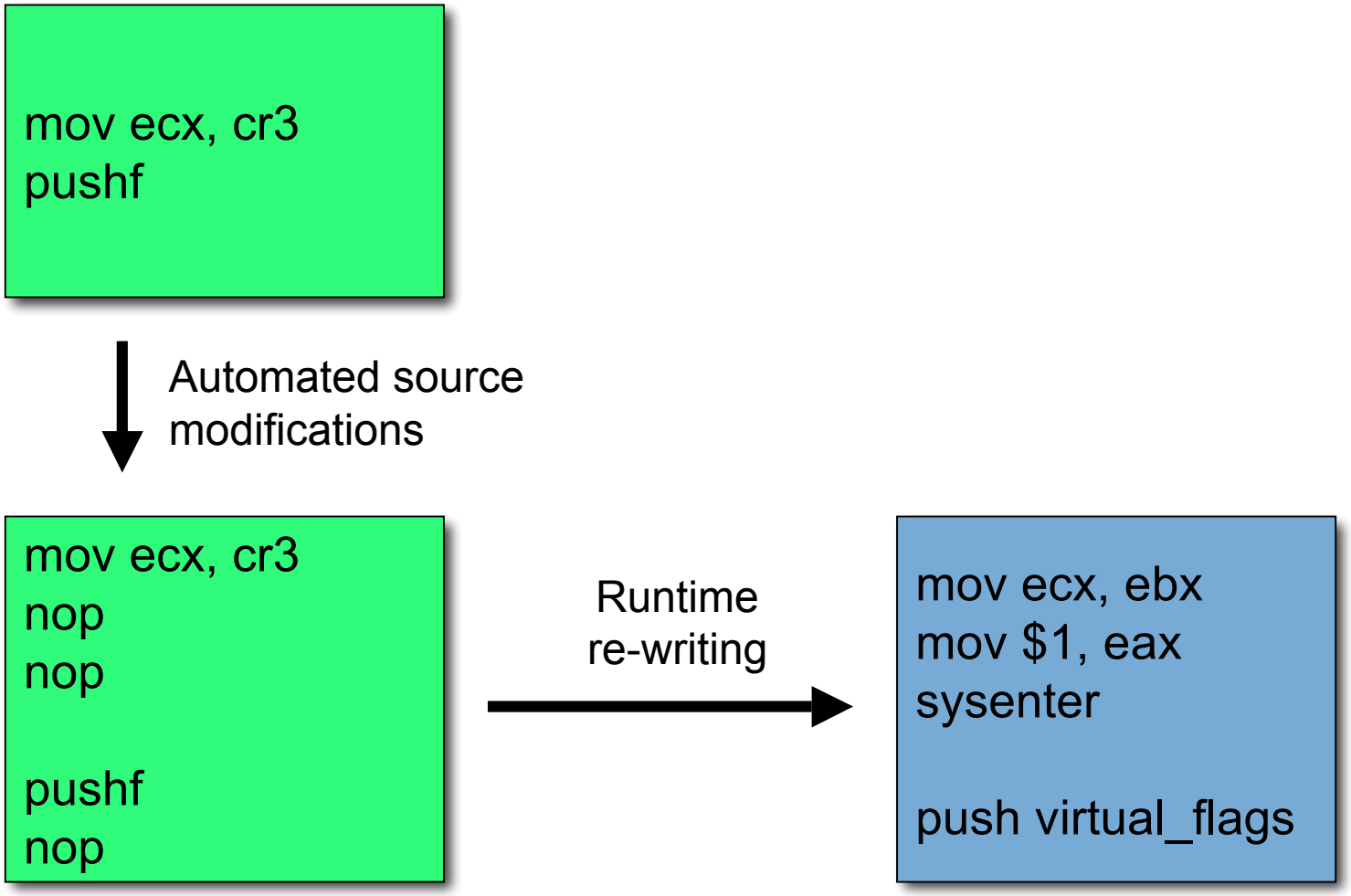


- L4Linux, Denali, Xen
- Replace sensitive instructions with hypercalls
  - Avoids costs of trapping
  - Batch state changes into single hypercall
  - Use apps unmodified
- Problems:
  - Engineering effort
  - Reduces trustworthiness of guest OS
  - Ties guest OS directly to a single hypervisor

Pre-virtualization



Pre-virtualization Basics

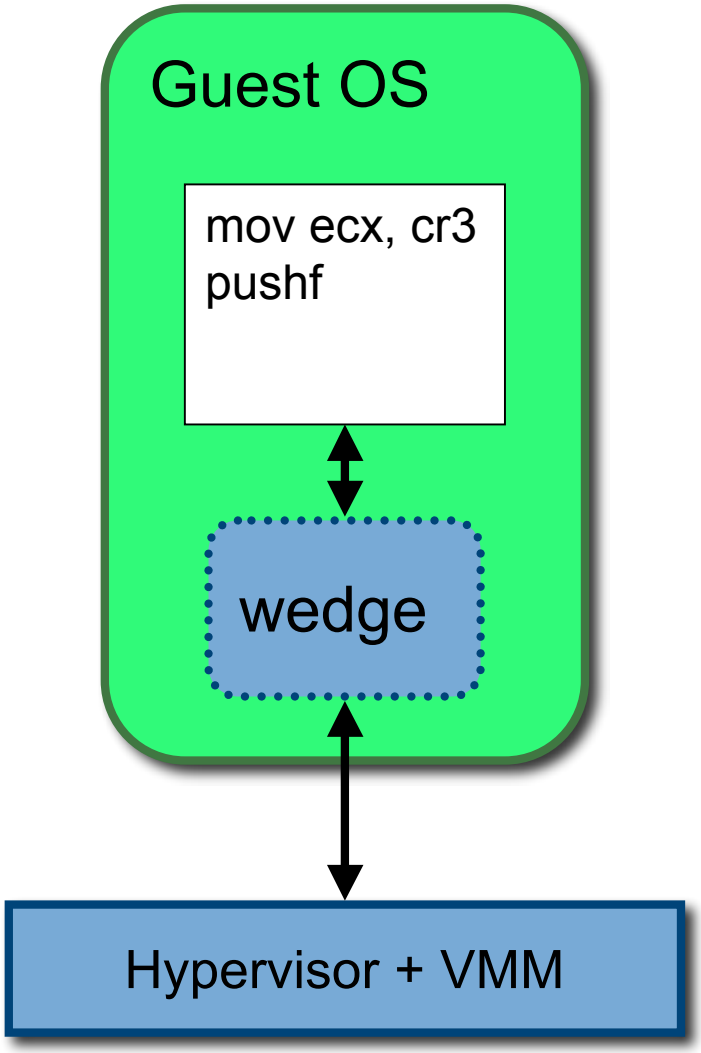


Sensitive Memory and Pre-virtualization

- Virtualization-sensitive state in memory:
  - page tables, tss, idt, gdt, etc.
  - memory mapped devices

- Use a profile-feedback loop to detect
  - Instrument and annotate the instructions that access sensitive memory
  - Or compiler data-flow analysis?

In-Place Virtualization



- The wedge creates a virtual CPU
- Invokes hypercalls only when necessary
- Frequent operations, such as cli/sti, emulated in the wedge

Driver Reuse

Support New OS Endeavors

- Unmodified device driver reuse:
  - Binary or recompiled source
  - Combine drivers from different OS's
- Reuse without legacy constraints
- Isolate drivers
  - Protect new OS from drivers
- Ex: 64+ CPU scalability

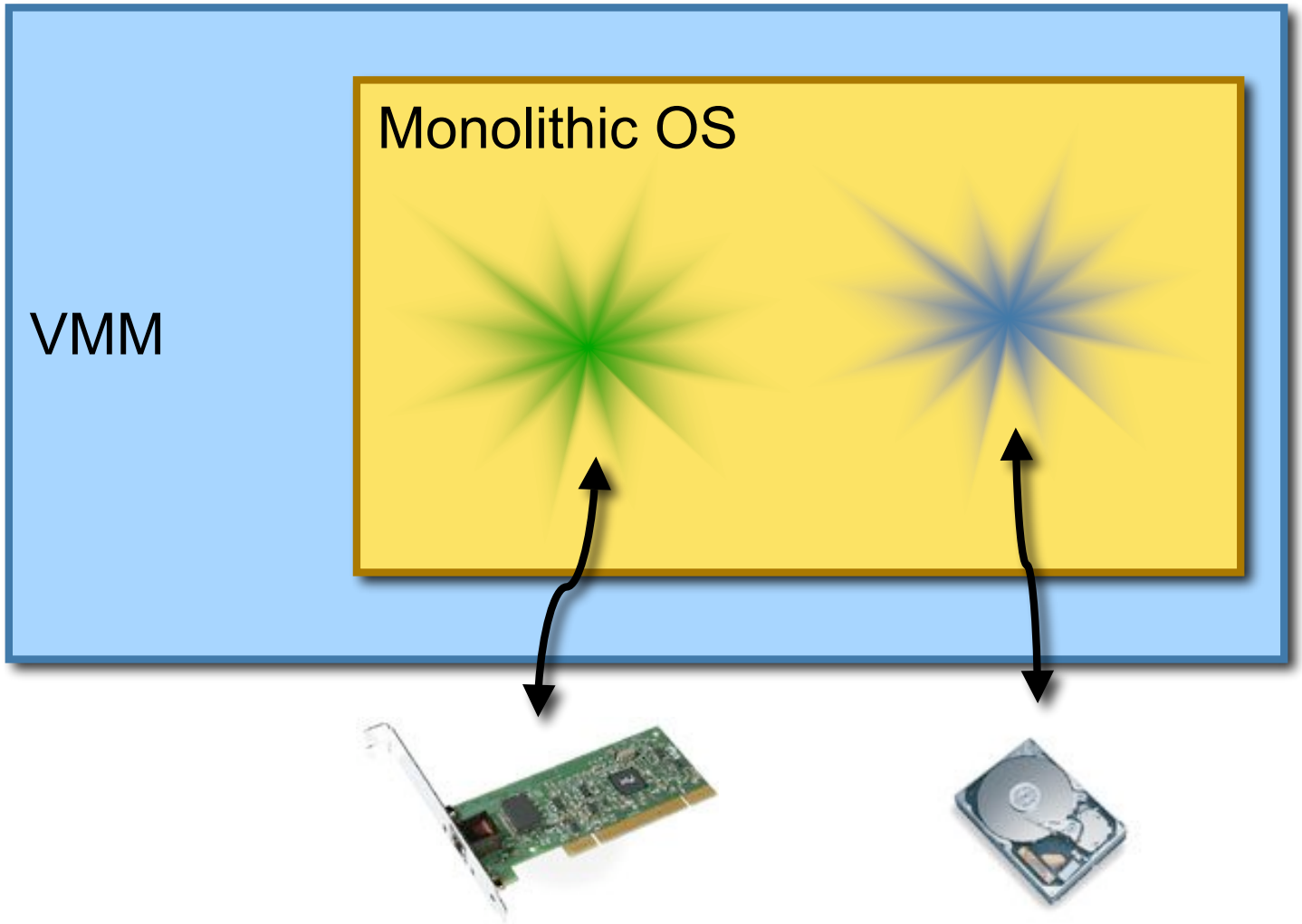
Overview

1. Motivation
2. Our solution
3. Virtualization issues
4. Evaluation

Motivation

- Building a new OS
  - It needs device drivers
- Drivers are major component of OS
  - Linux 2.4.21: 70% of code for ia32
- Implement new drivers
  - Linux, BSD: device documentation
  - Windows: many 3rd party drivers, no code
  - Exotic or out-of-production hardware

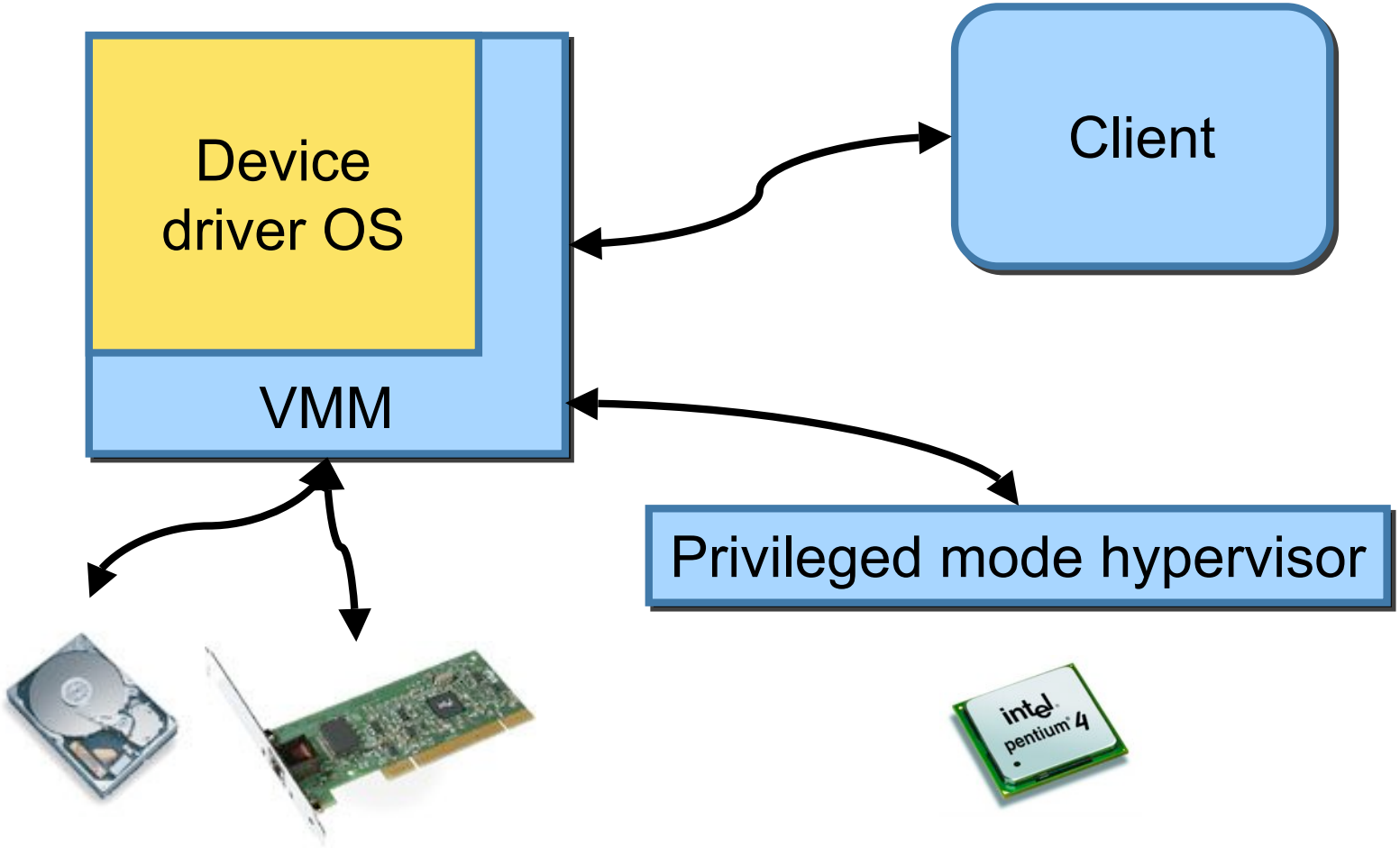
Driver State Machine



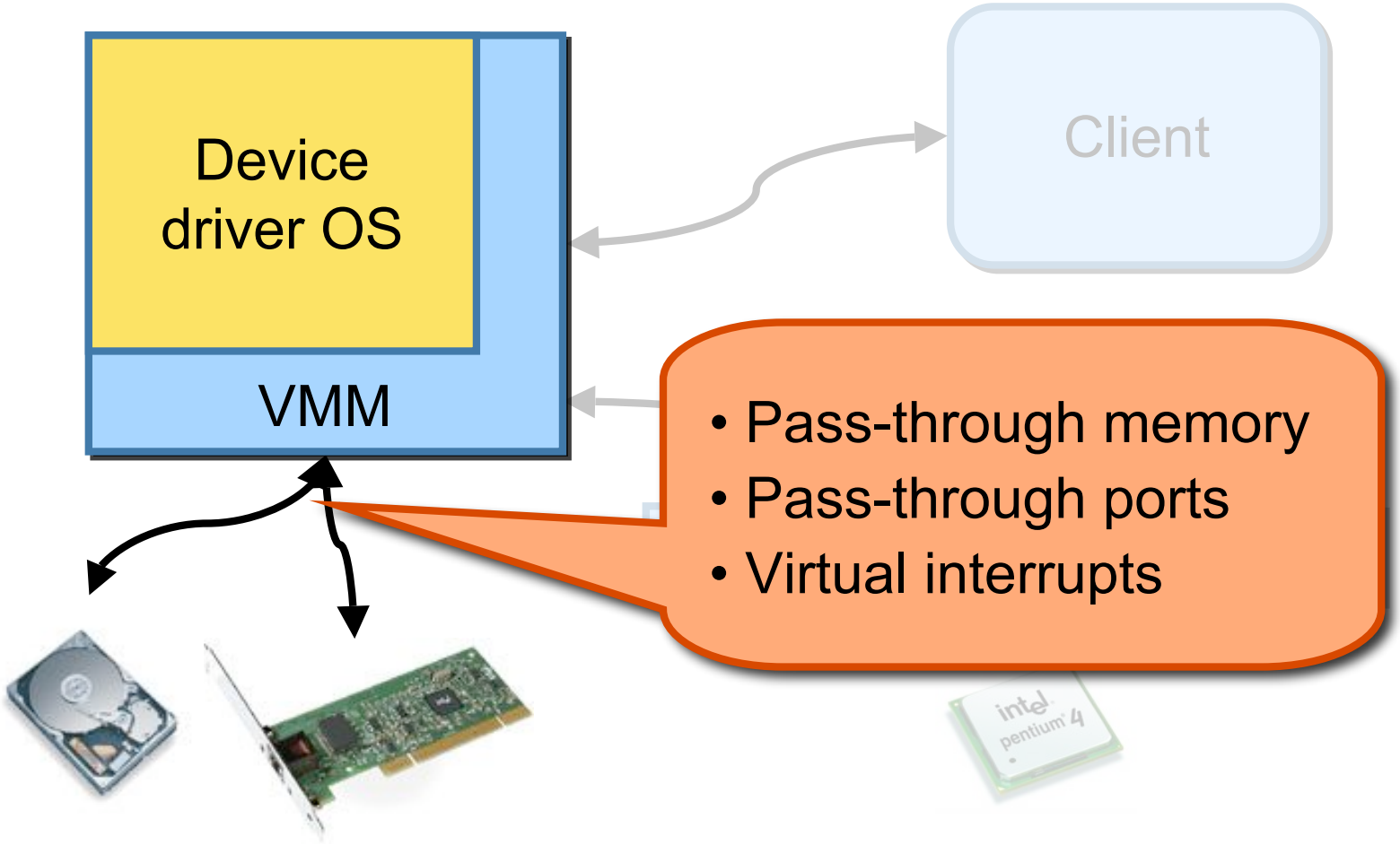
Interface Categories

- Providing resources to drivers
  - Pure virtualization API (e.g. VMware)
  - Para-virtualization API (e.g. Xen)
- Passing requests to drivers
  - Translation modules

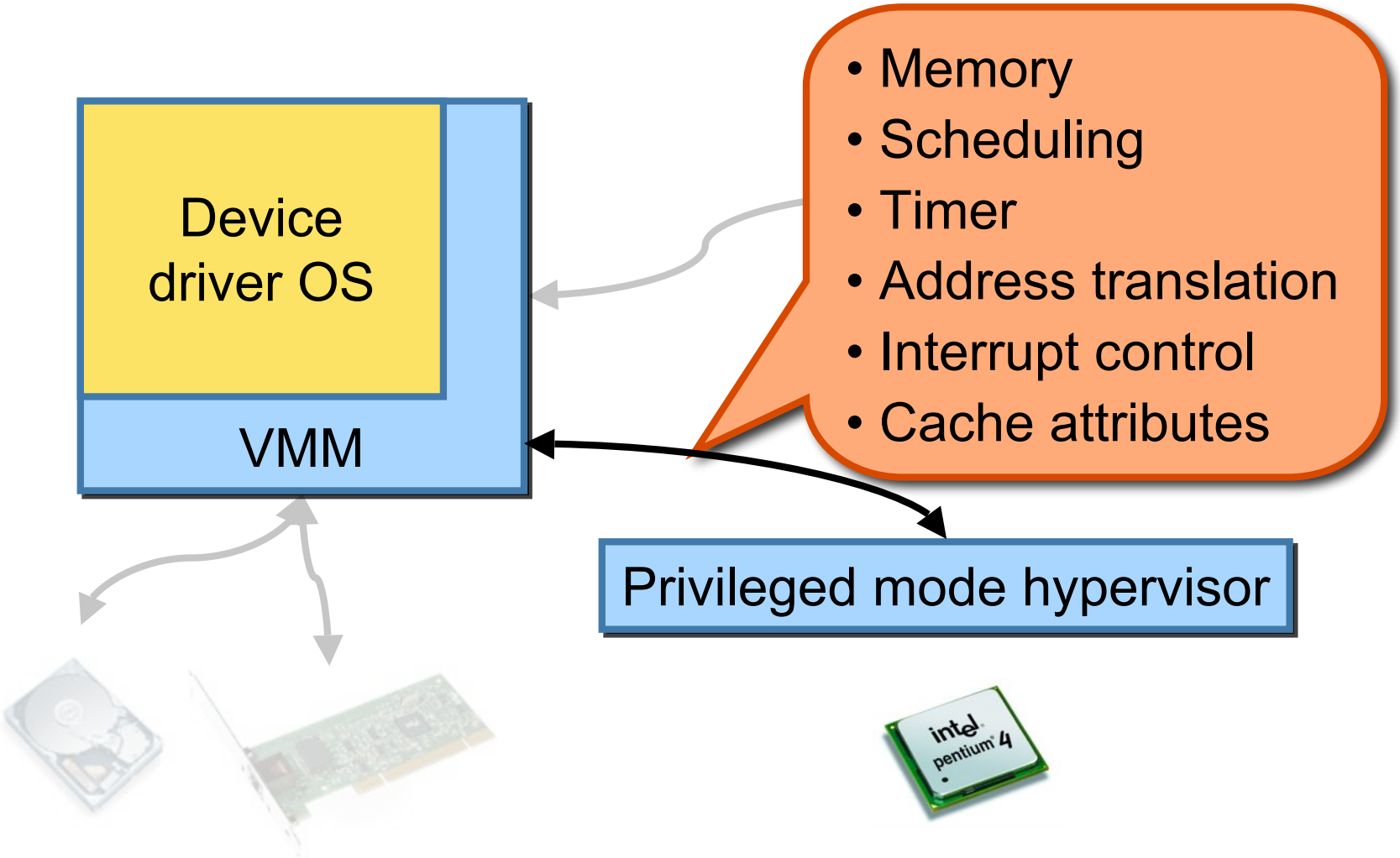
Resource Interfacing



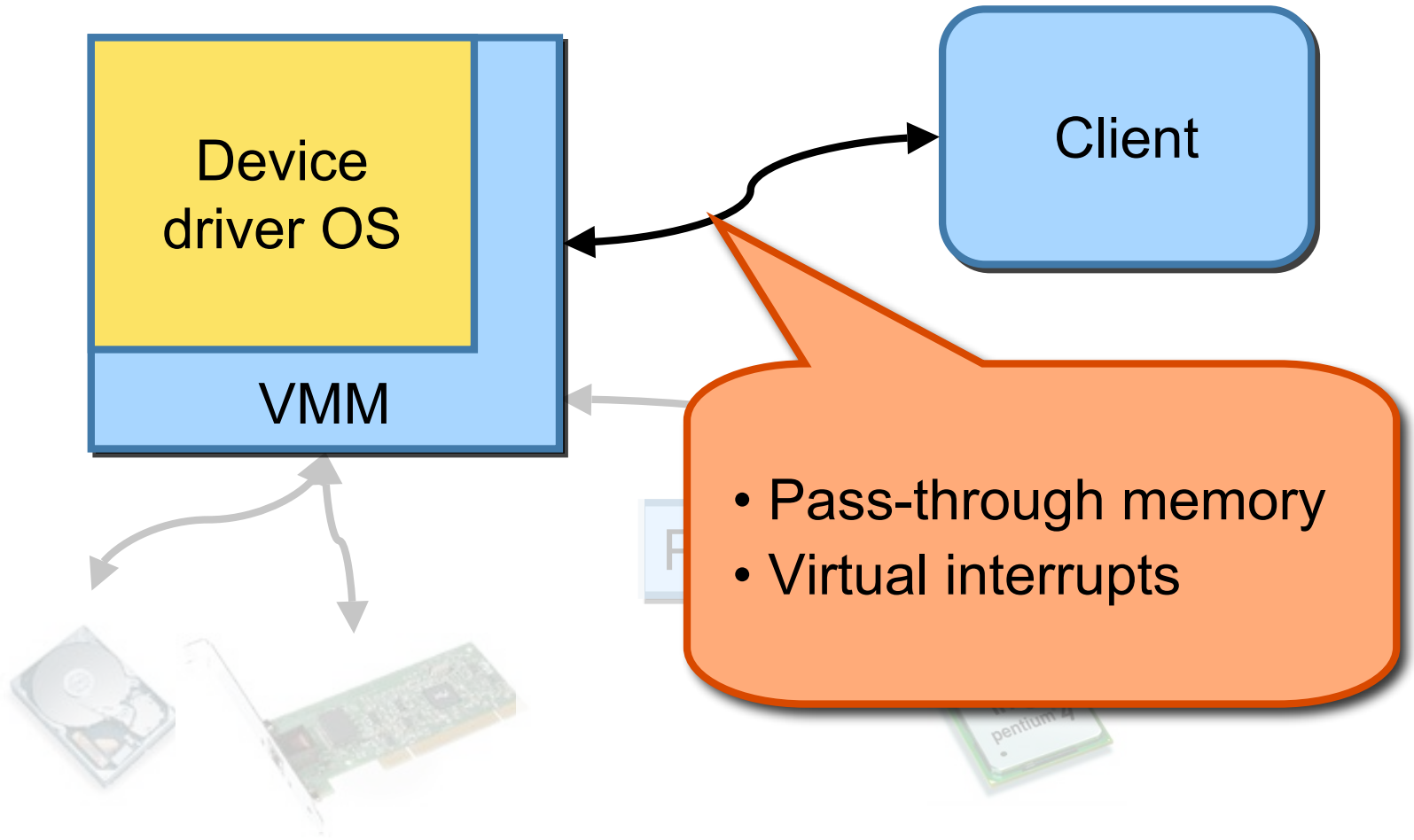
Resource Interfacing



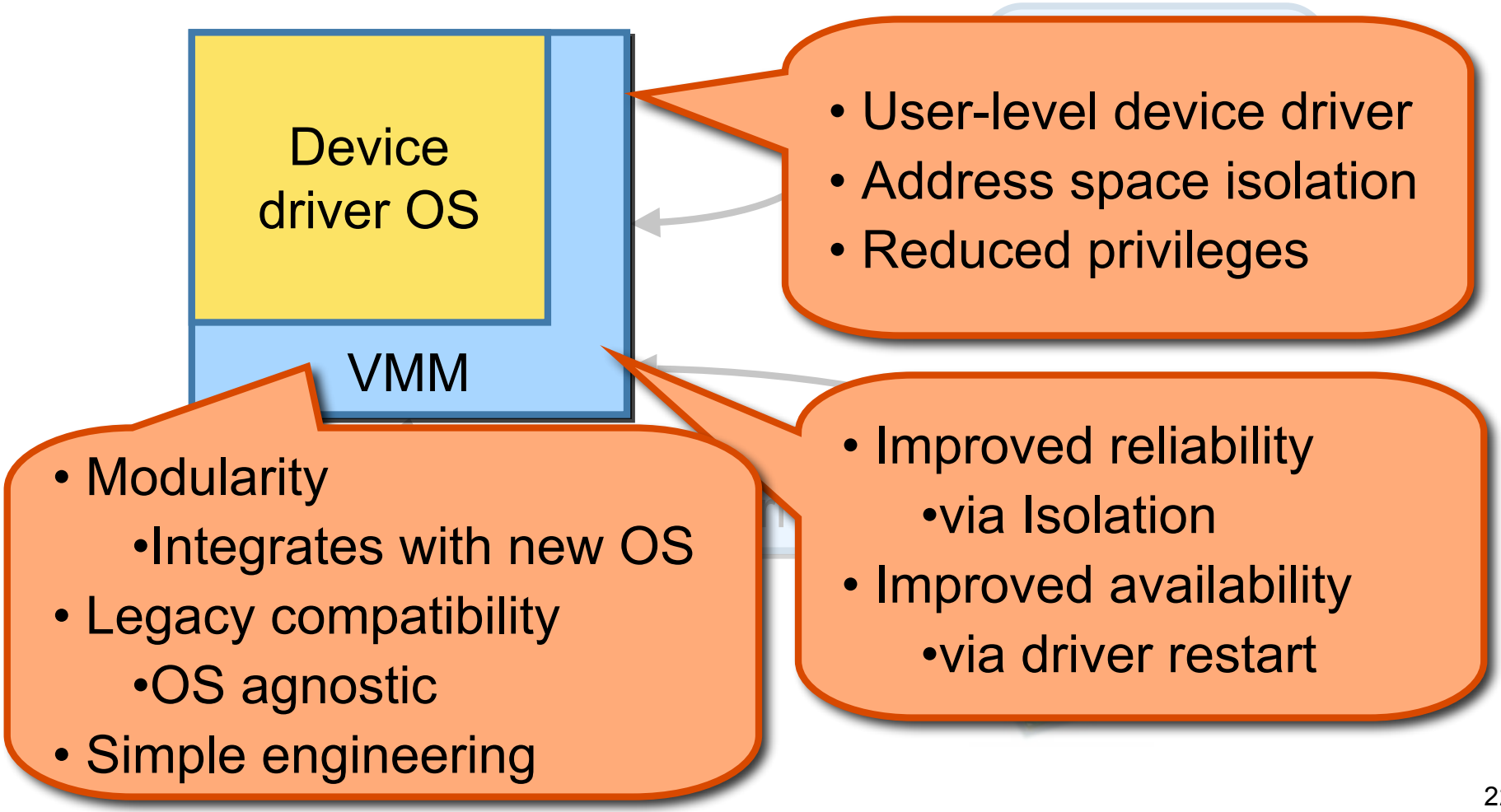
Resource Interfacing



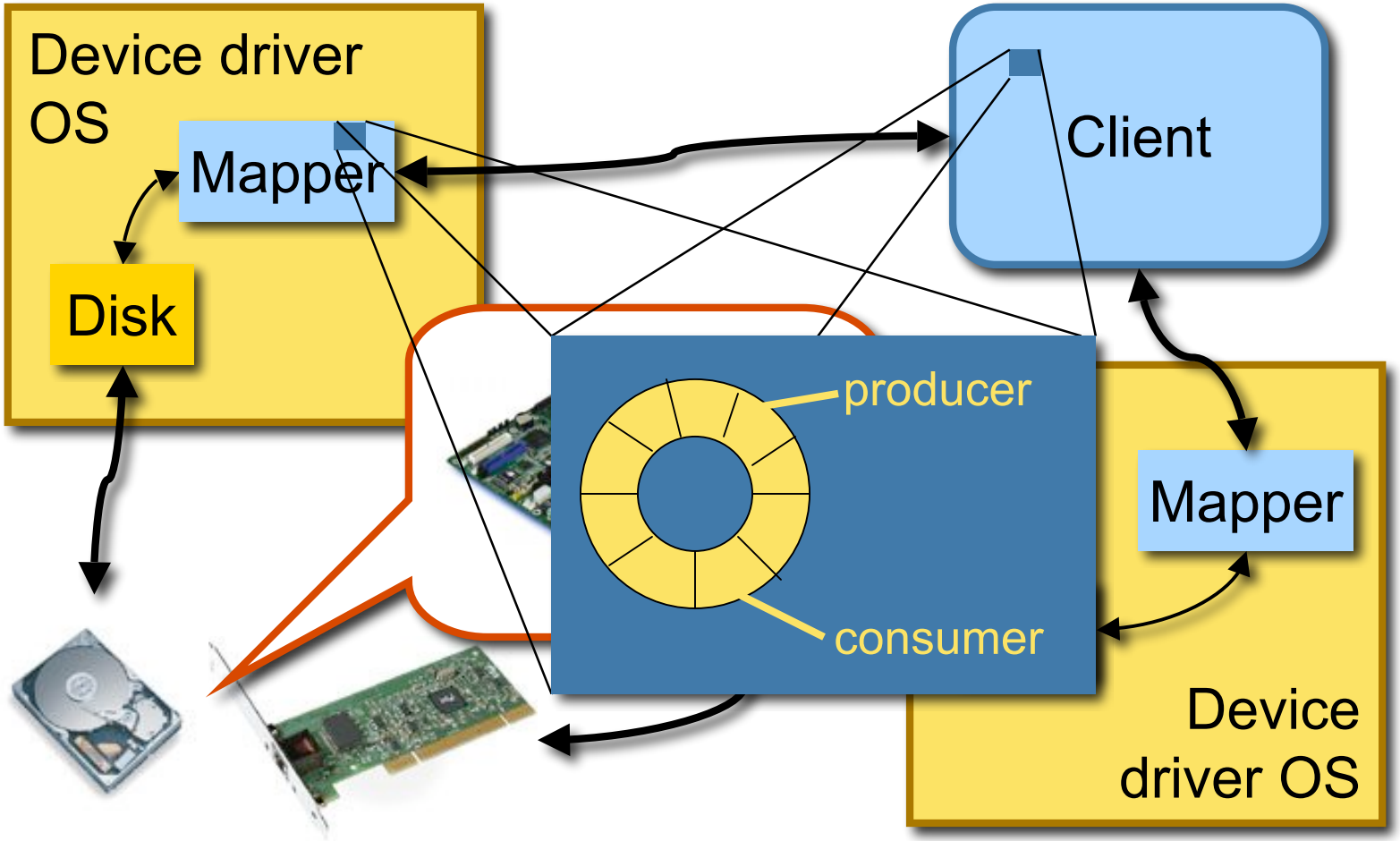
Resource Interfacing



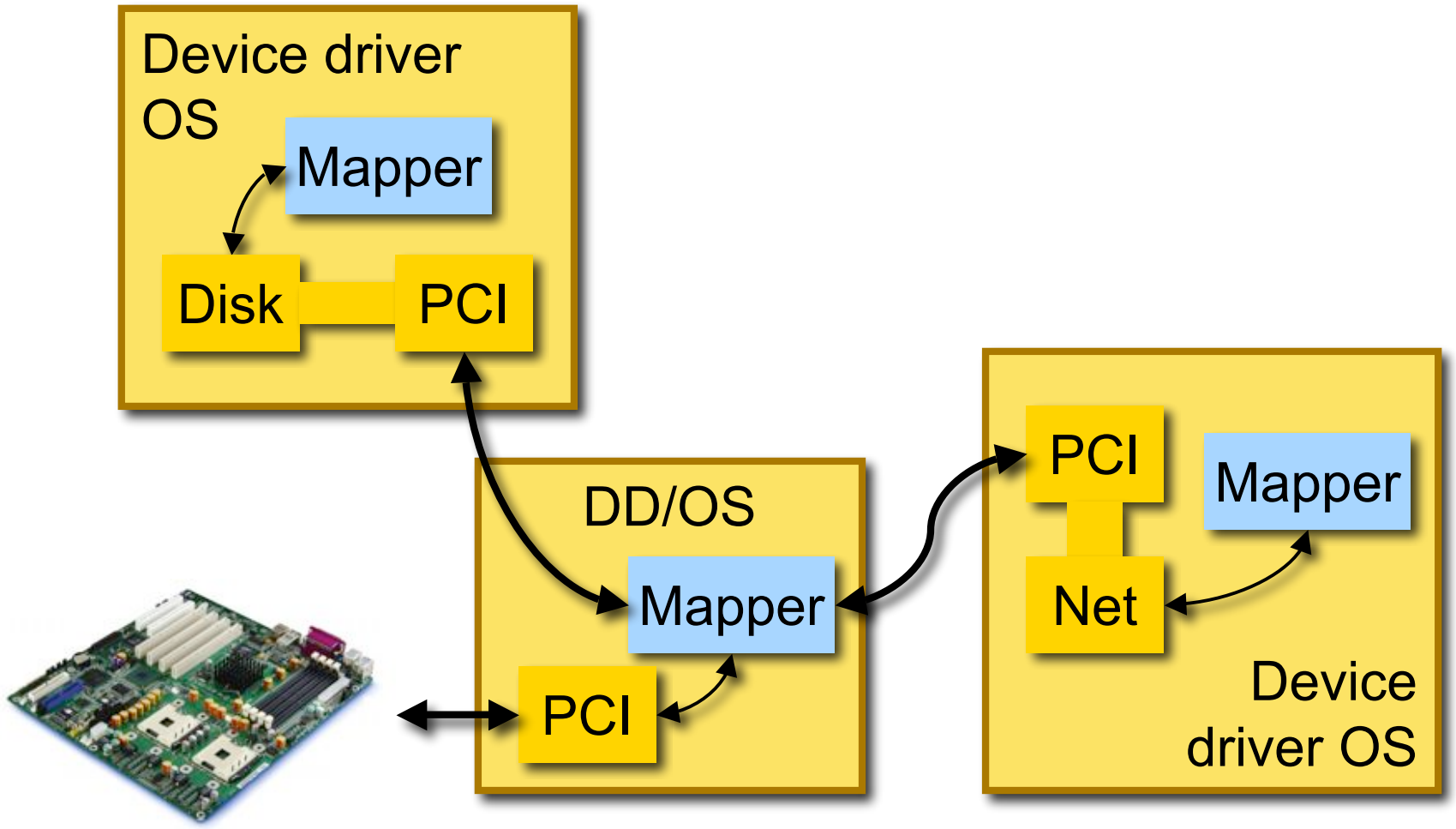
Resource Interfacing



Driver Interfacing



Recursive Driver Interfacing



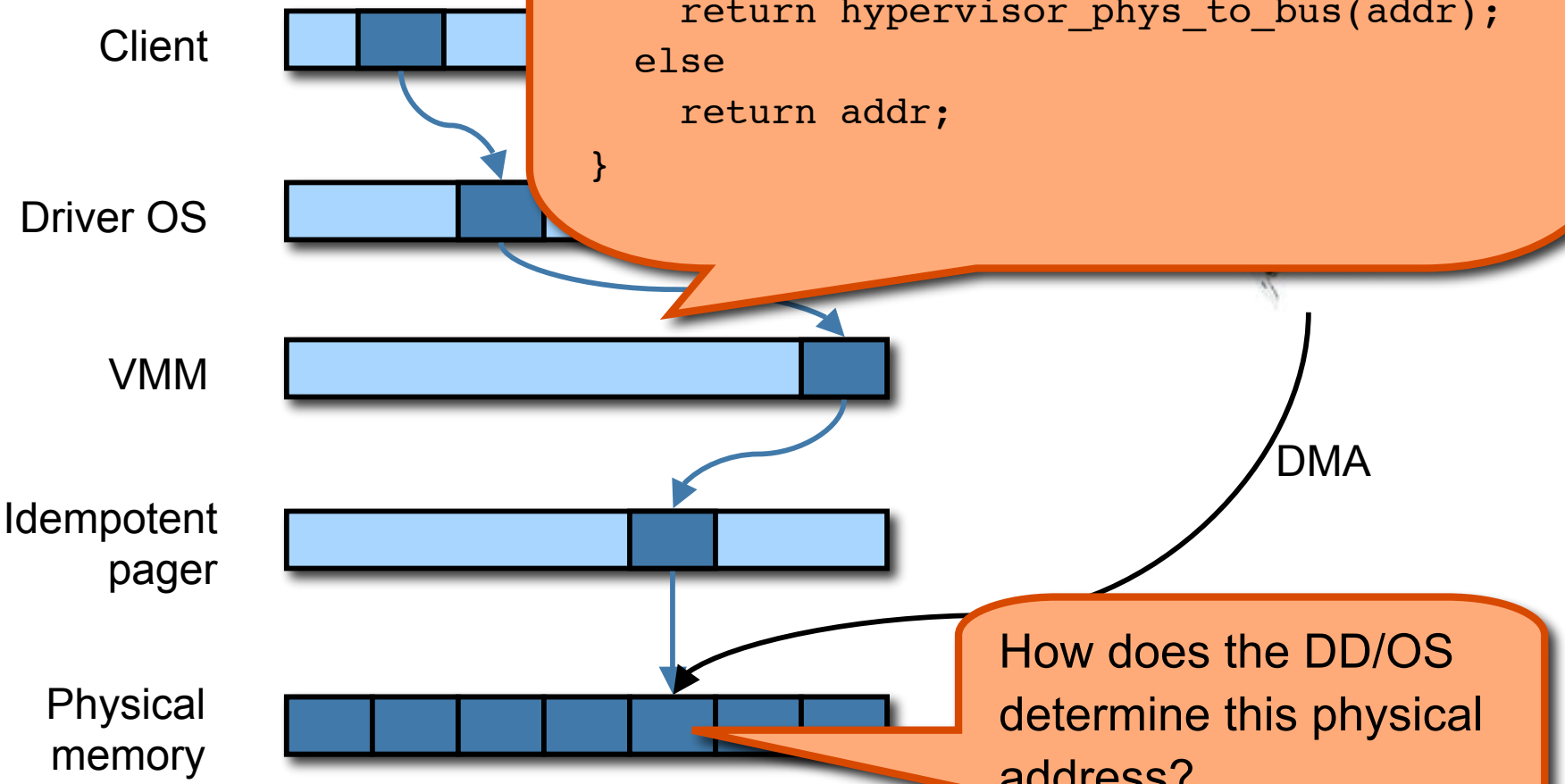
Virtualization Issues

1. DMA address translation
2. DMA and trust
3. Resource consumption
4. Timing
5. Shared hardware & recursion

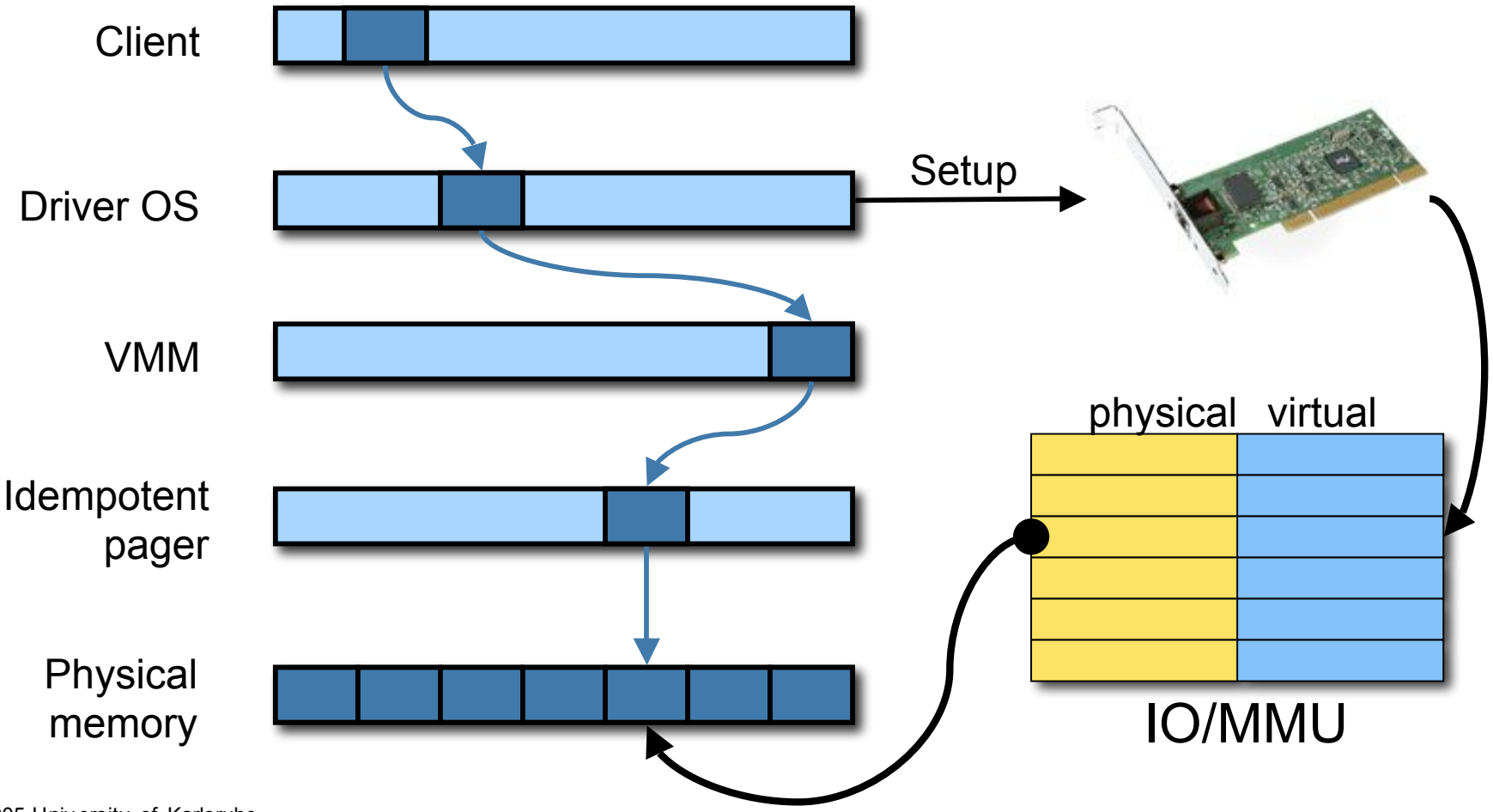
DMA A

Simple with *para-virtualization*: queries VMM for translation.

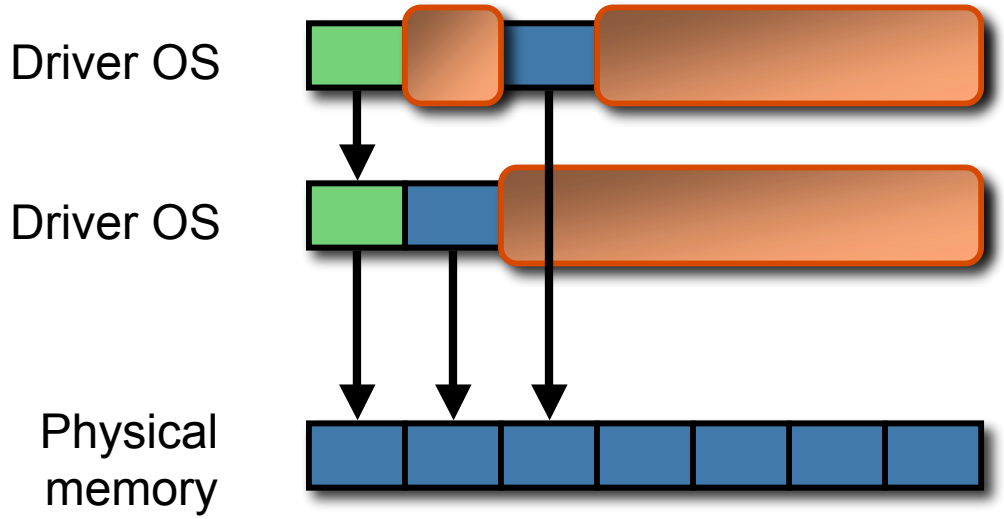
```
phys_to_bus(addr) {
  if( paravirtualization )
    return hypervisor_phys_to_bus(addr);
  else
    return addr;
}
```



IO/MMU Address Translation



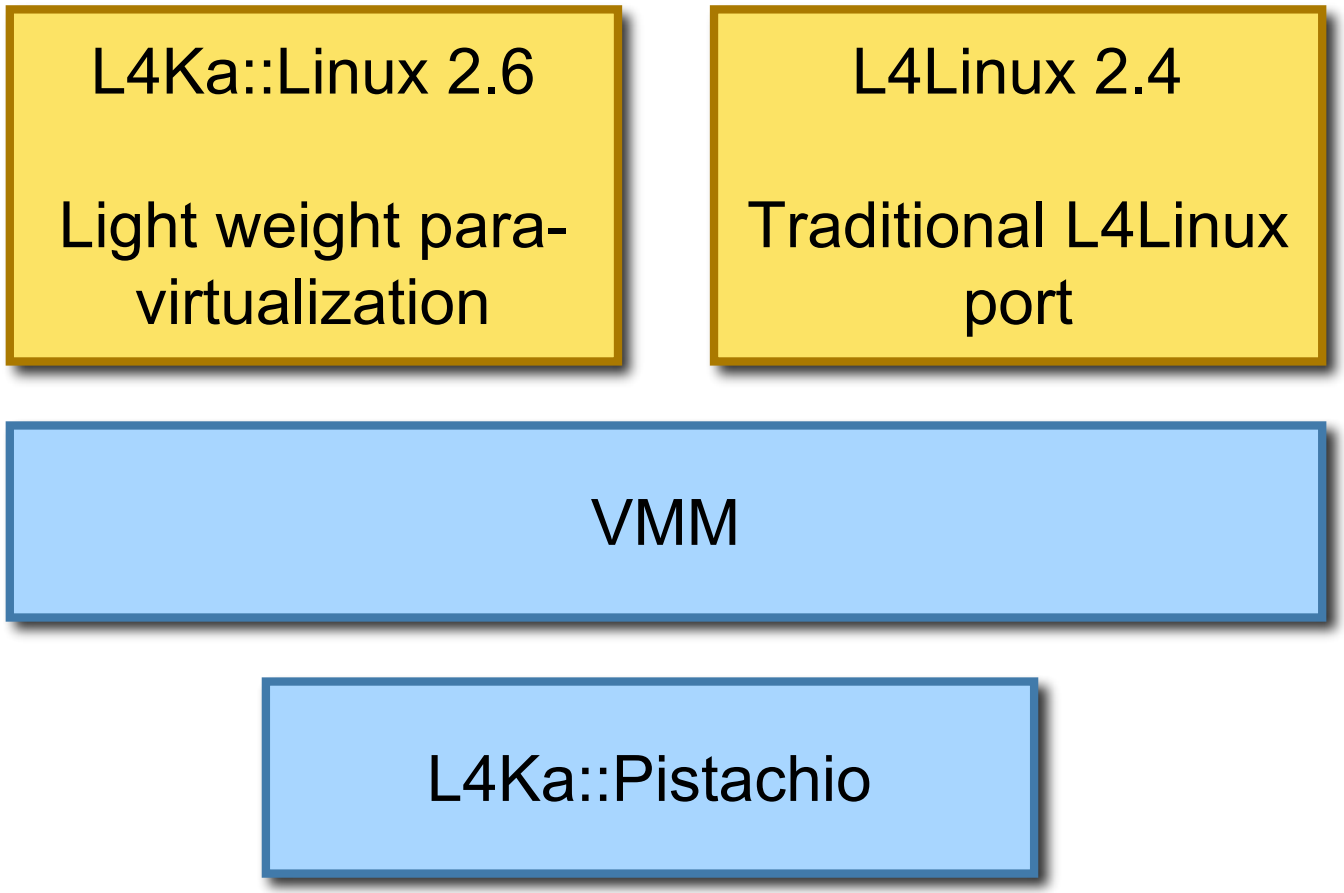
Careful DMA Address Translation



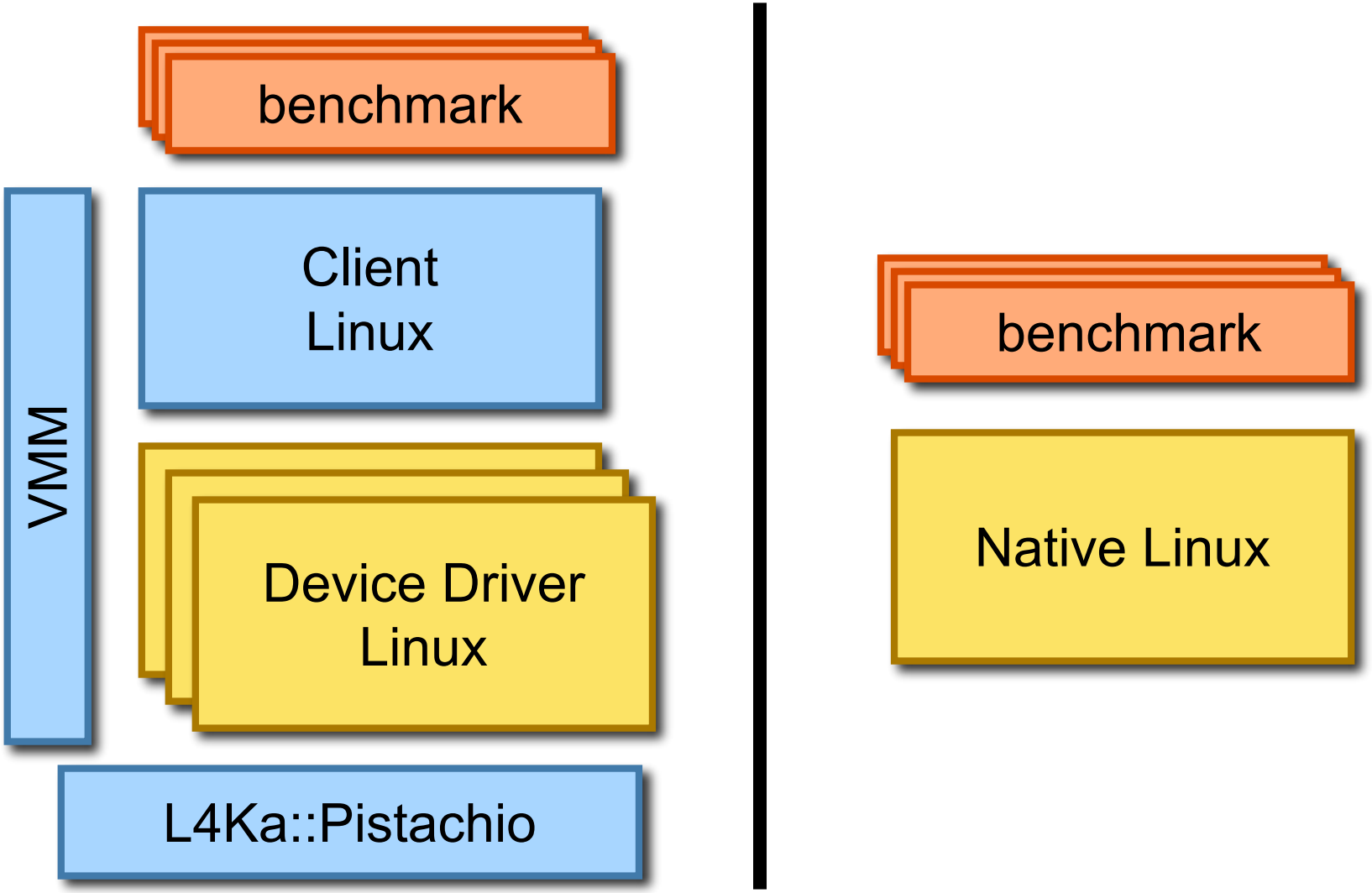
Timing

- DD/OS's have virtualized time
- Devices unaware of virtual time
- Preemption could violate driver timing
- Heuristic: "obey" disabled interrupts
  - Delay DD/OS preemption
  - Hard preemption to avoid denial of service

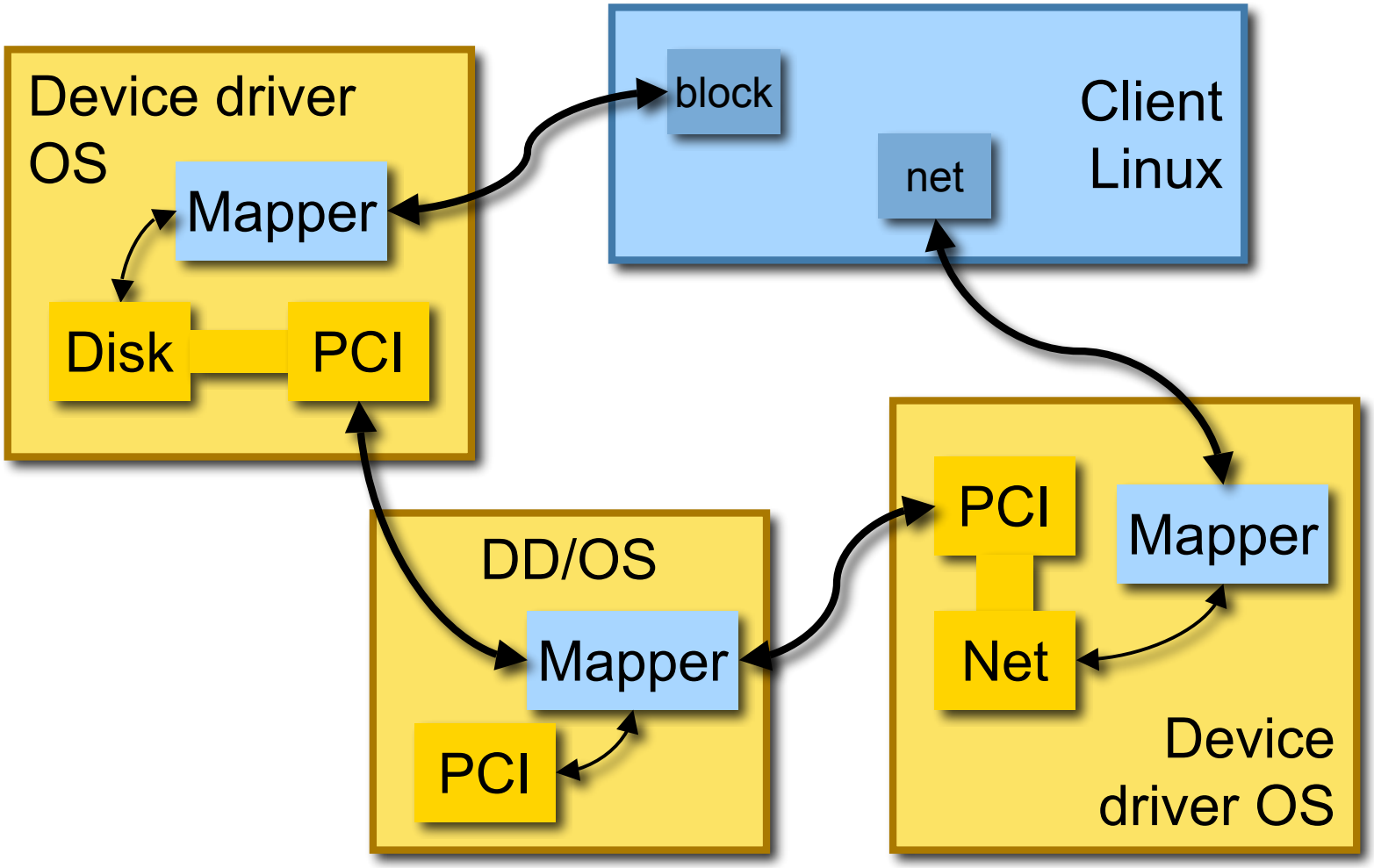
Evaluation (w/ Para-virtualization)



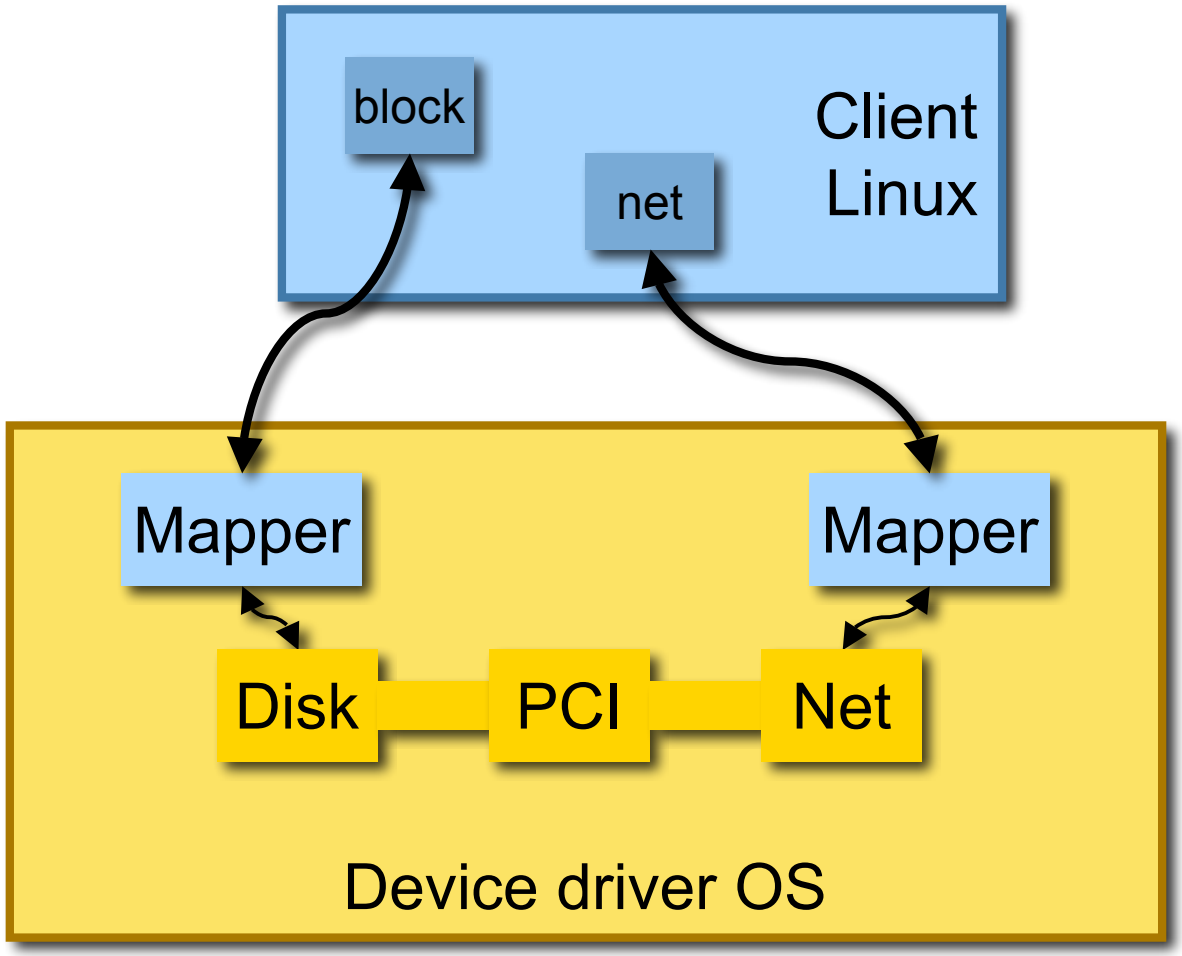
Comparative Benchmarking



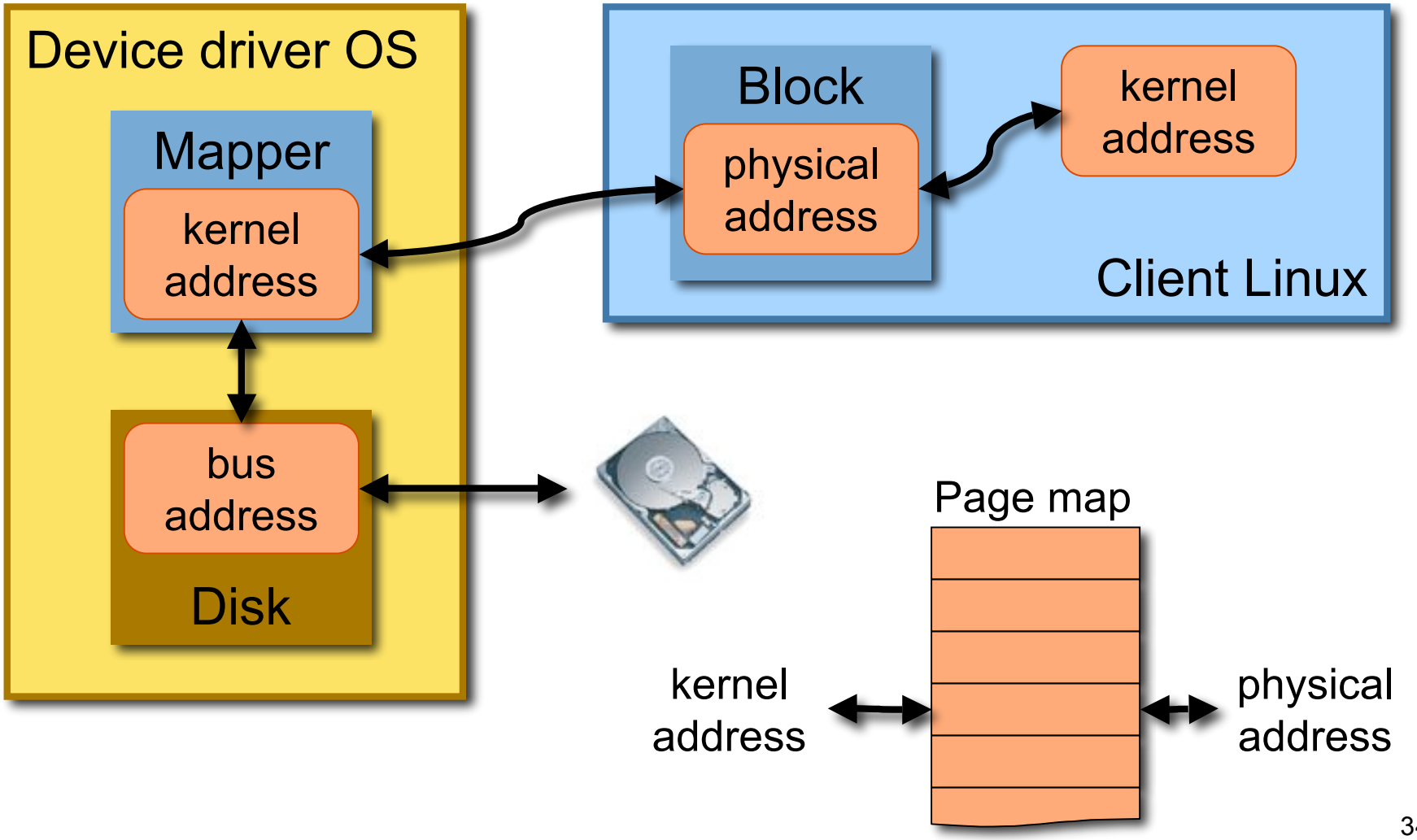
Isolated Architecture



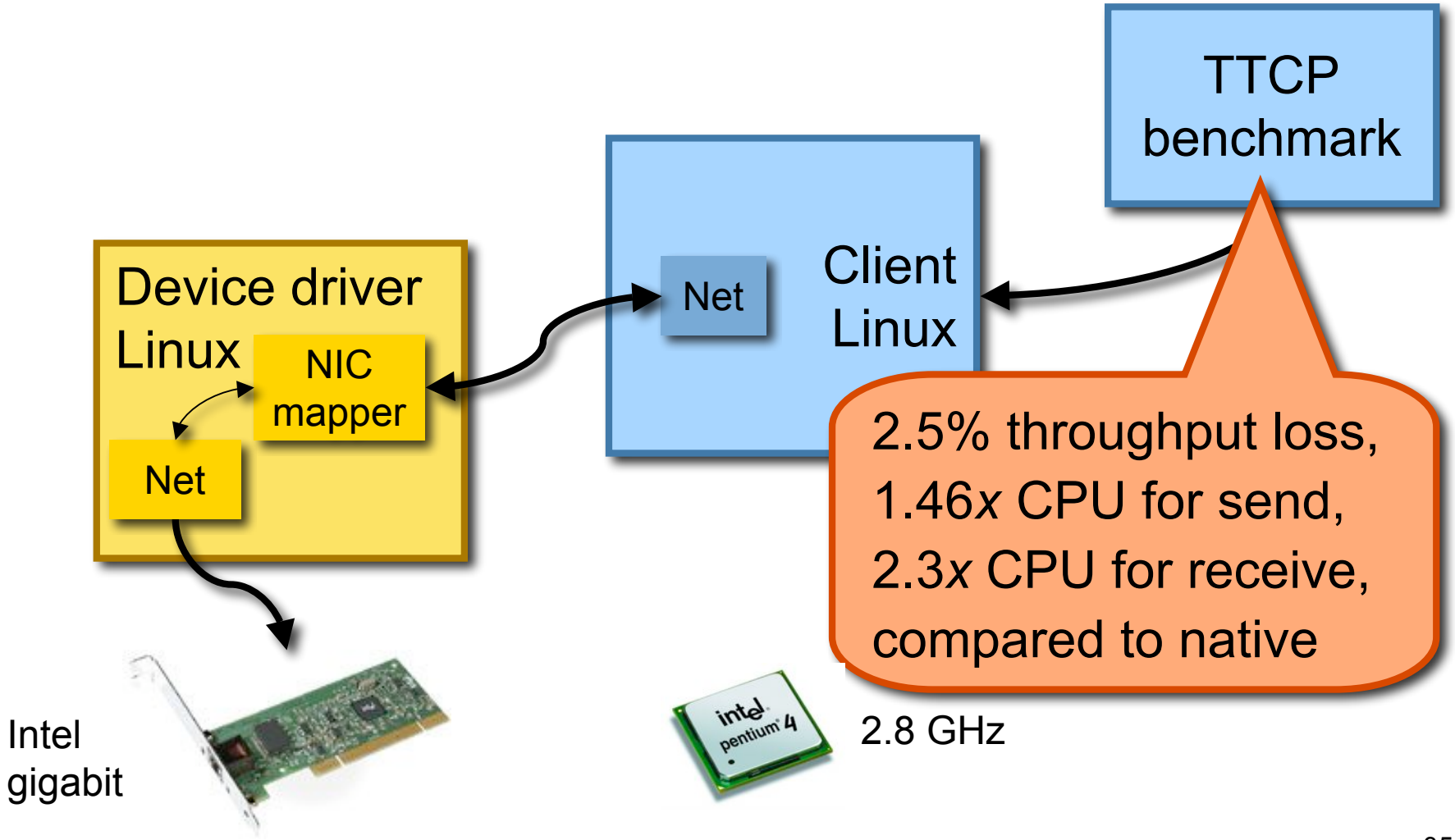
Consolidated Architecture



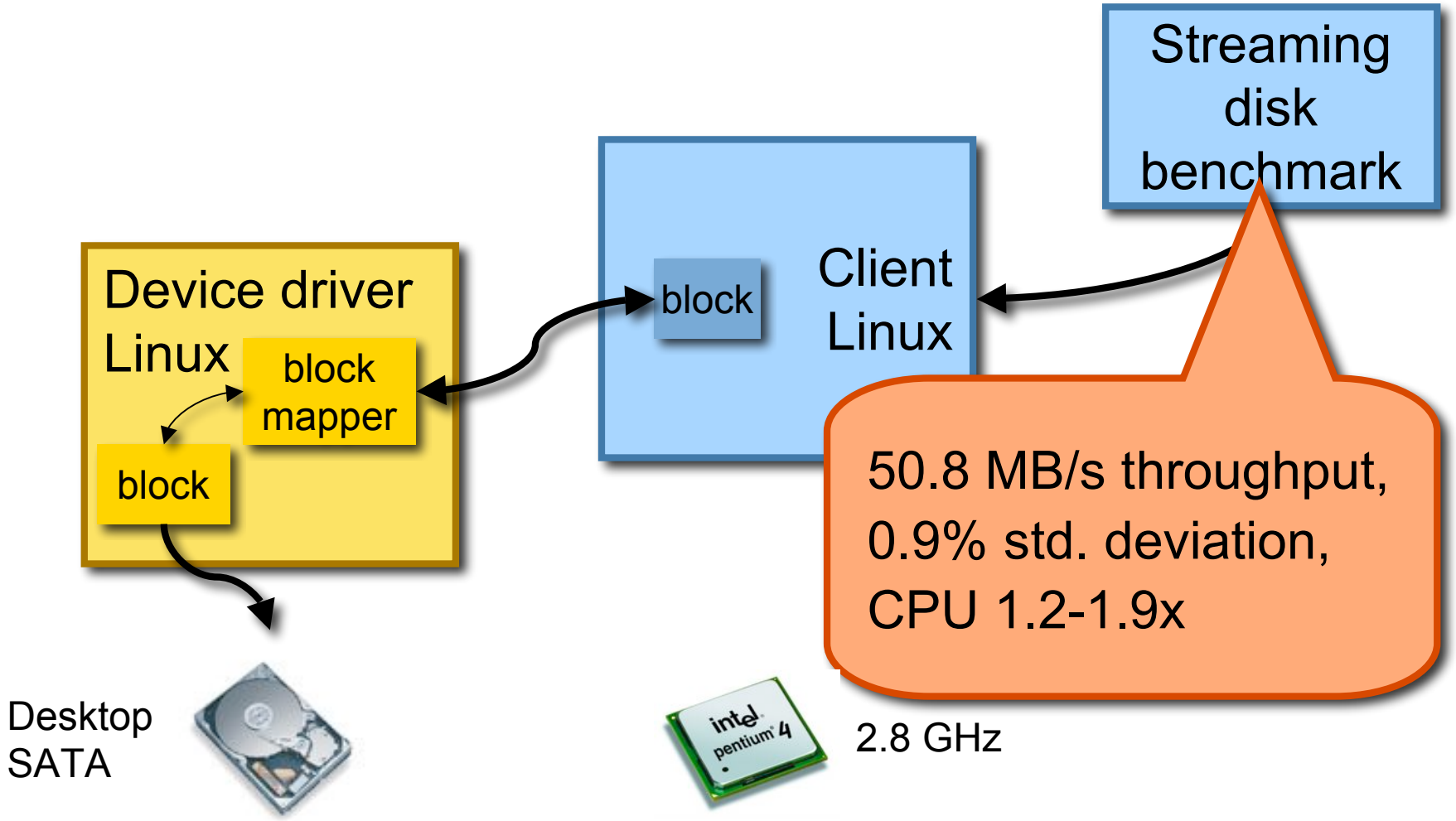
Address Mapping



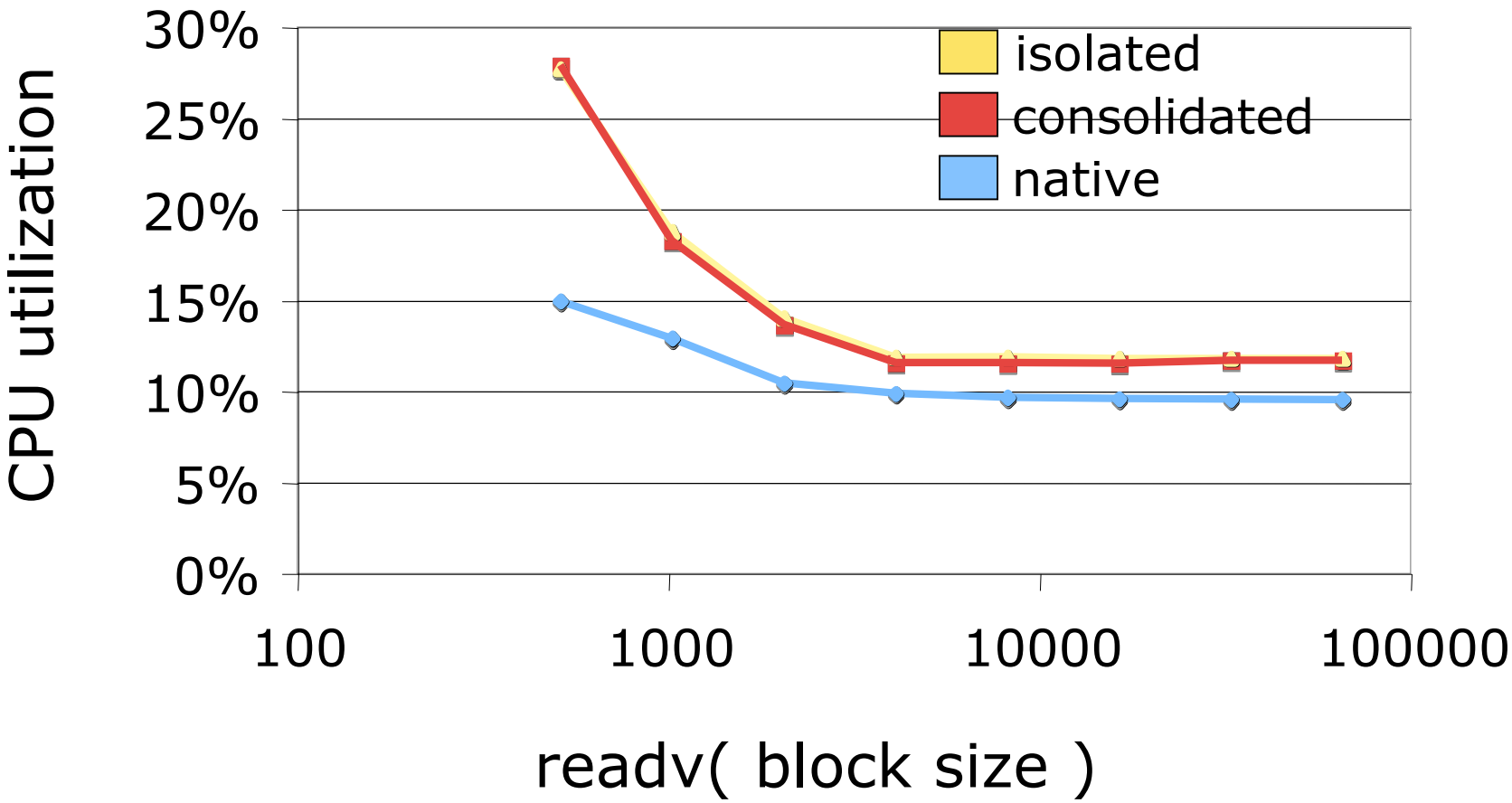
Network Performance



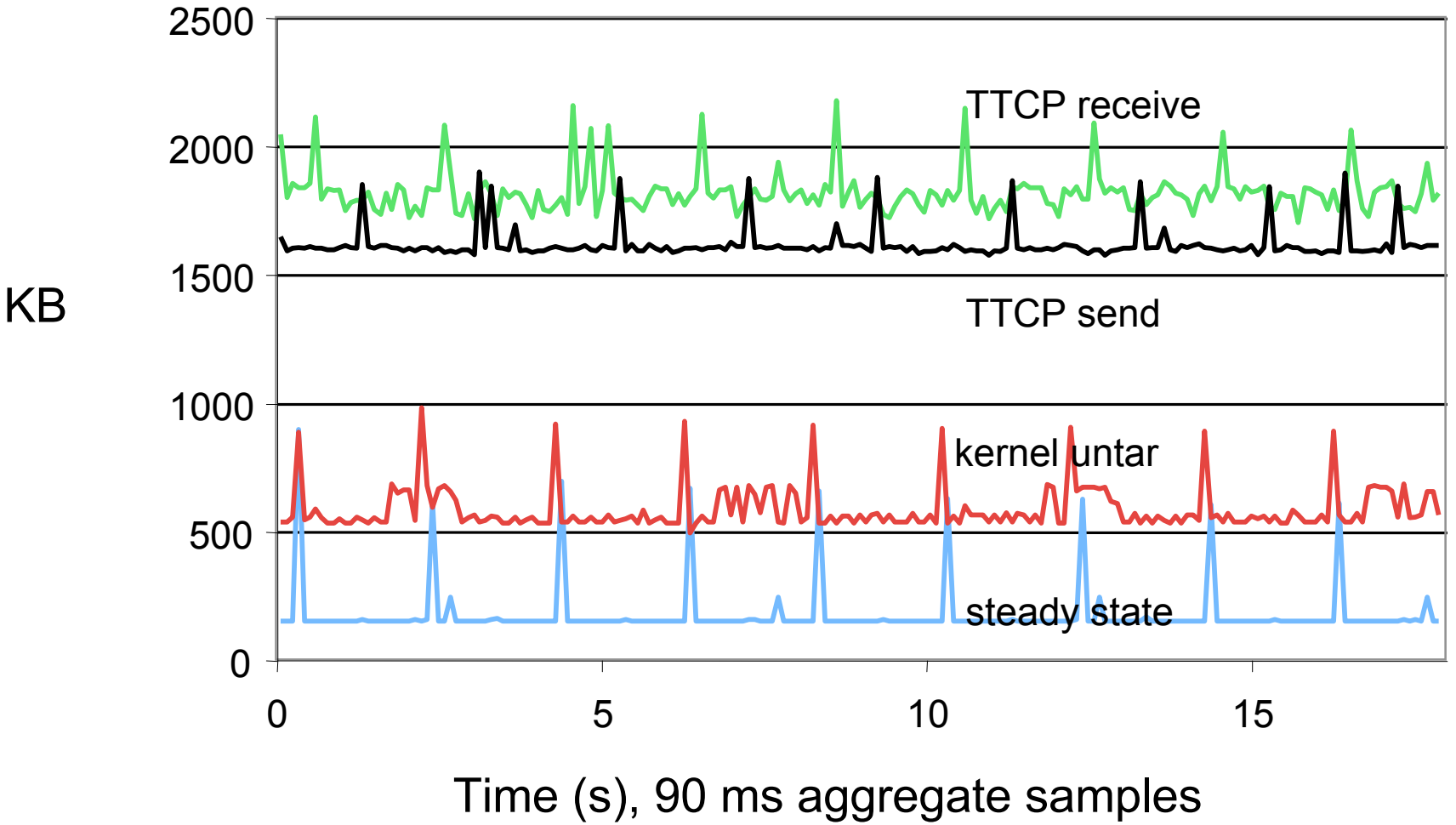
Disk Performance



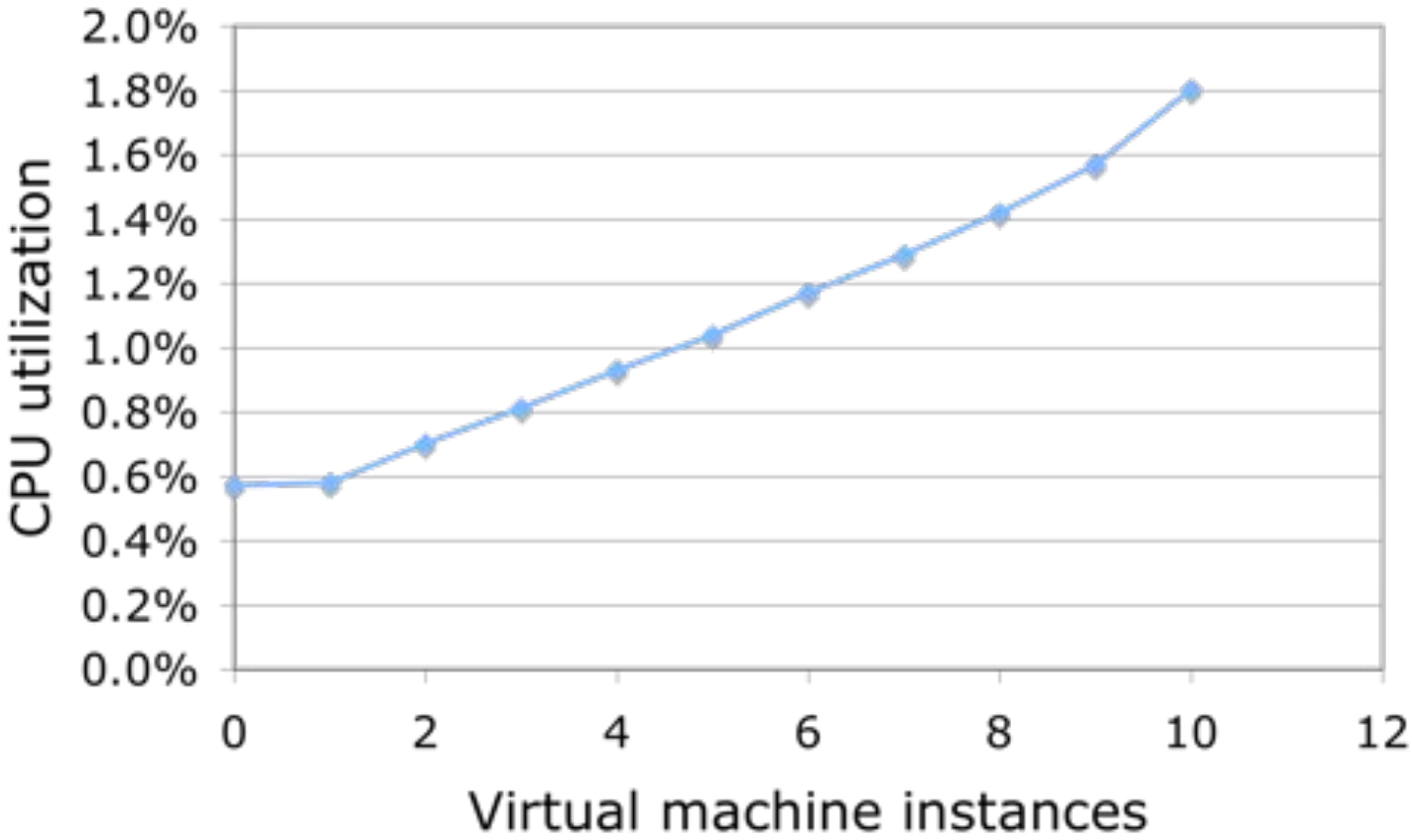
Disk CPU Utilization



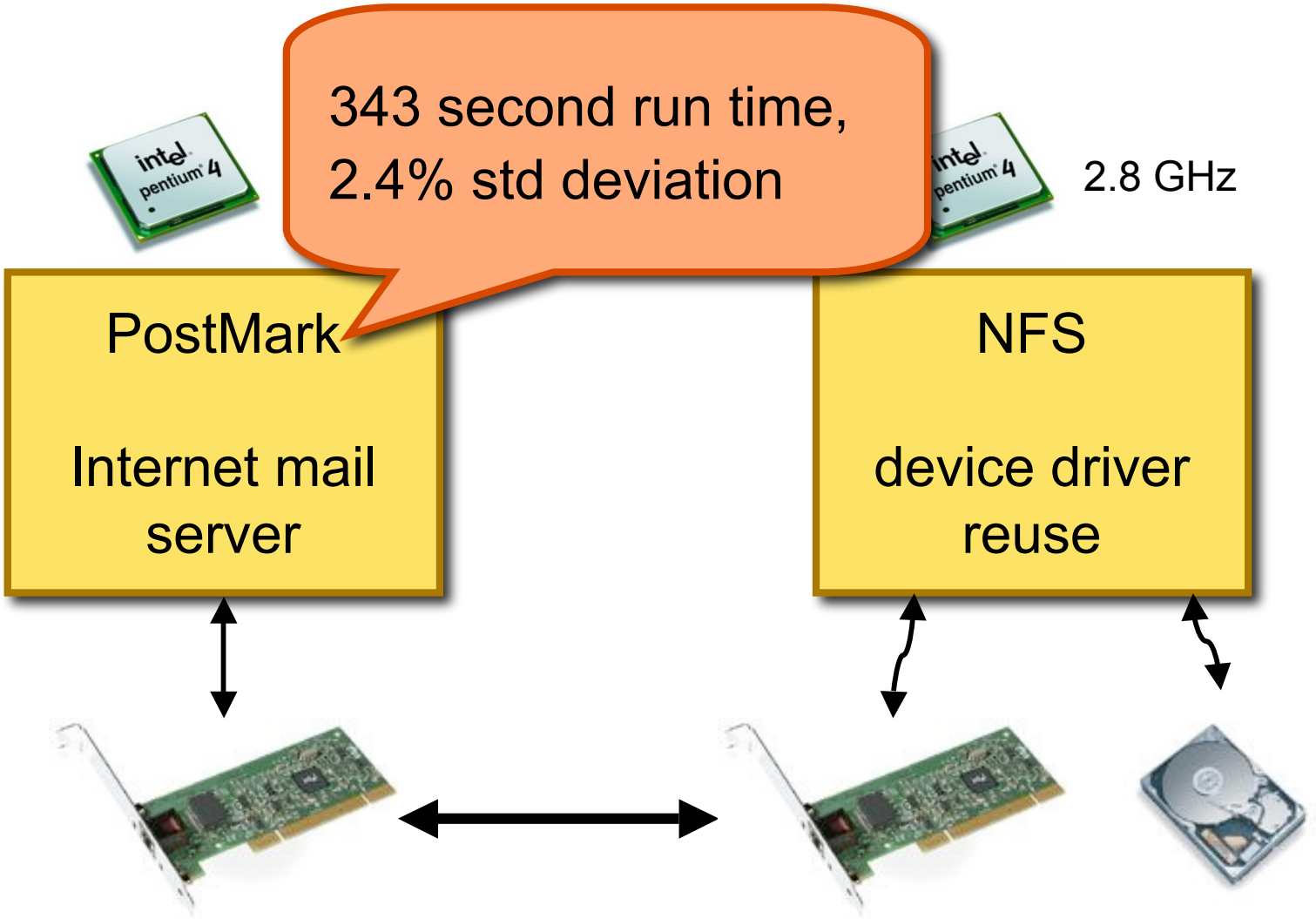
Memory Working Set



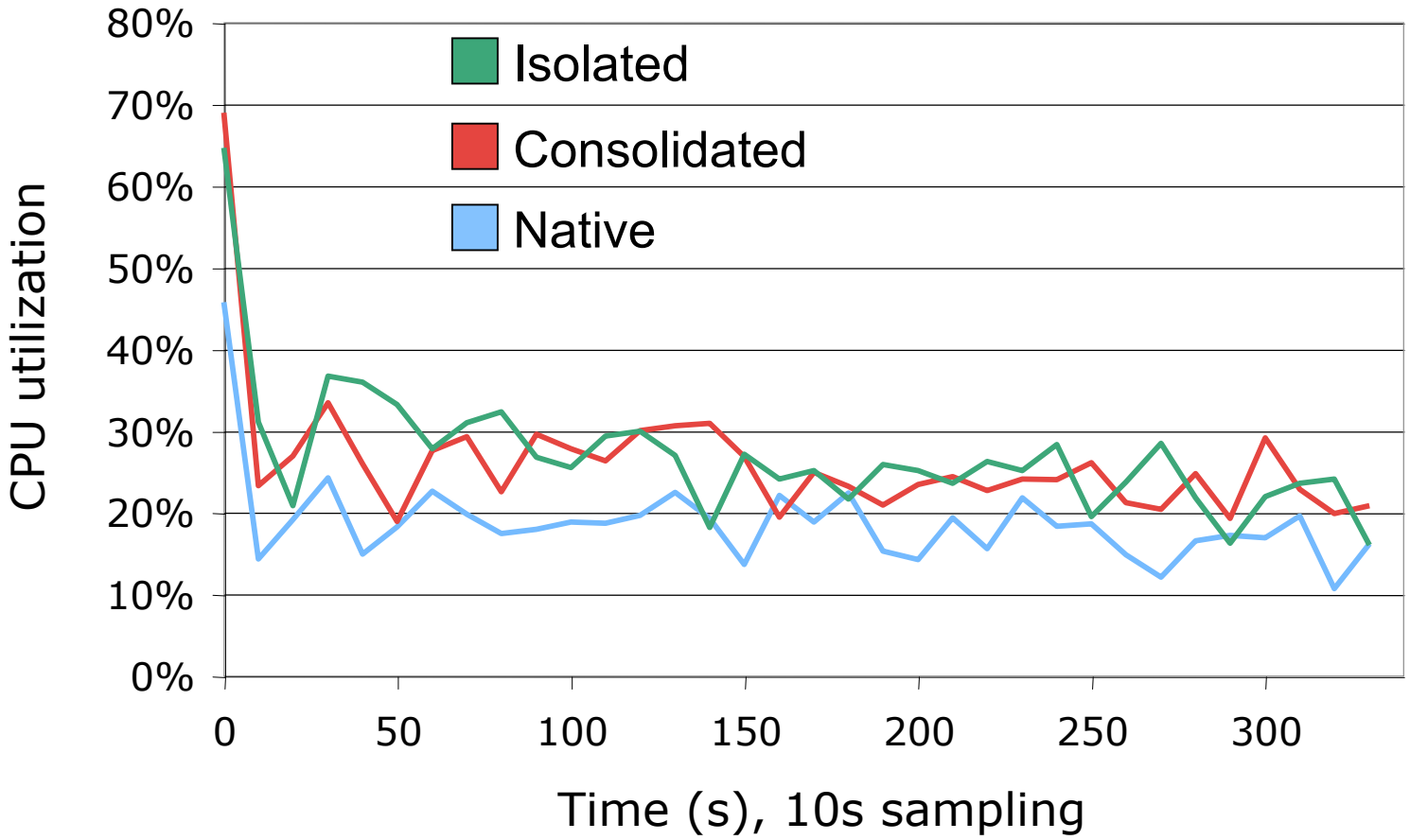
Inherent CPU Utilization



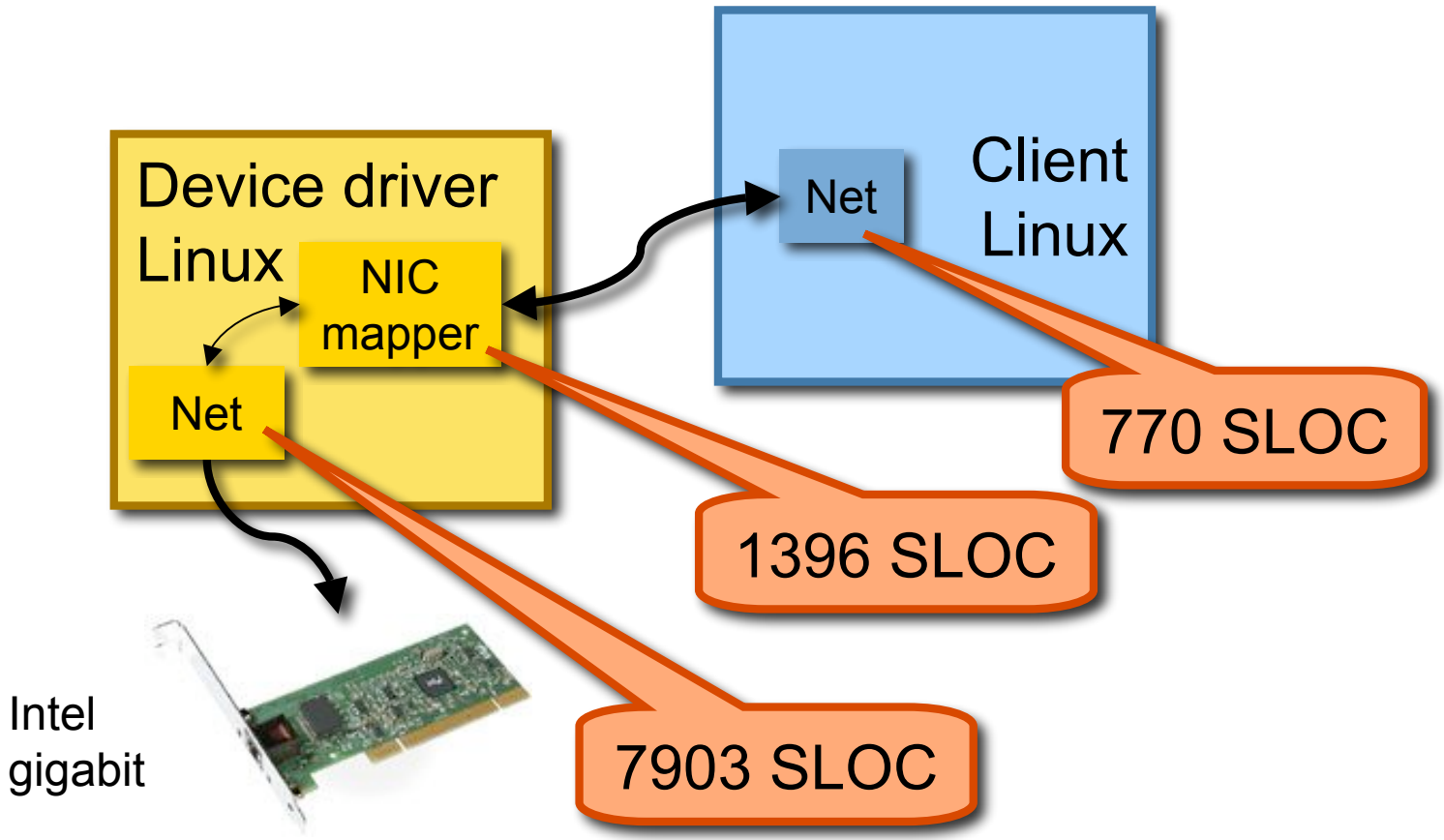
PostMark Benchmark



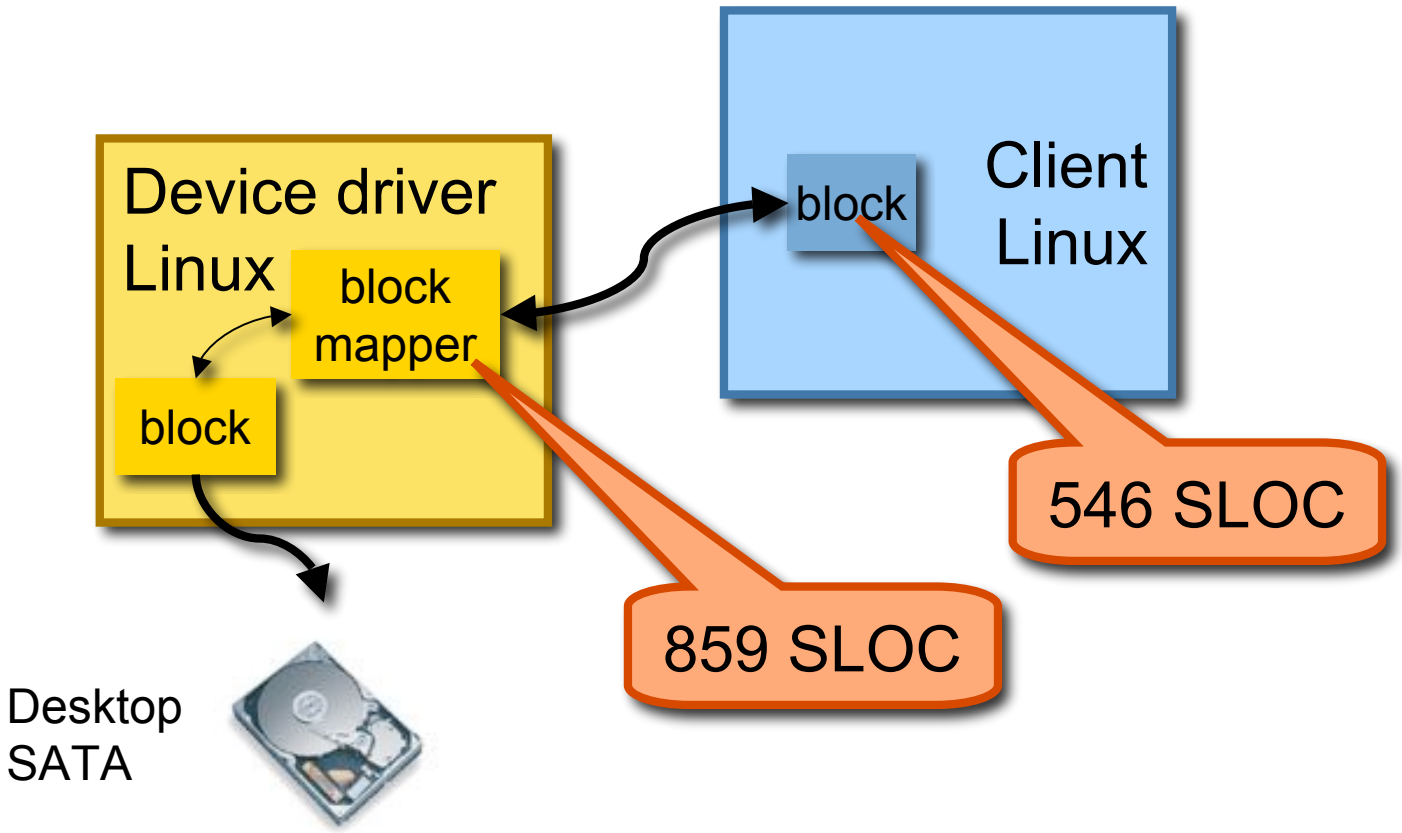
PostMark NFS CPU Utilization



Engineering Effort: Network



Engineering Effort: Disk



Conclusion

- Device driver reuse
  - Binary drivers
  - Source code drivers
- Run the original driver OS in a VM
  - Isolated drivers
  - Orthogonal design
- Little engineering effort for massive reuse
- Enhanced dependability