



Pre-Virtualization with Compiler Afterburning

Joshua LeVasseur, Volkmar Uhlig

University of Karlsruhe, Germany

September 22, 2005

<http://l4ka.org>



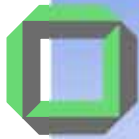
Overview

- Virtualization background
- Basics of pre-virtualization
- Virtualization kit
- Analysis

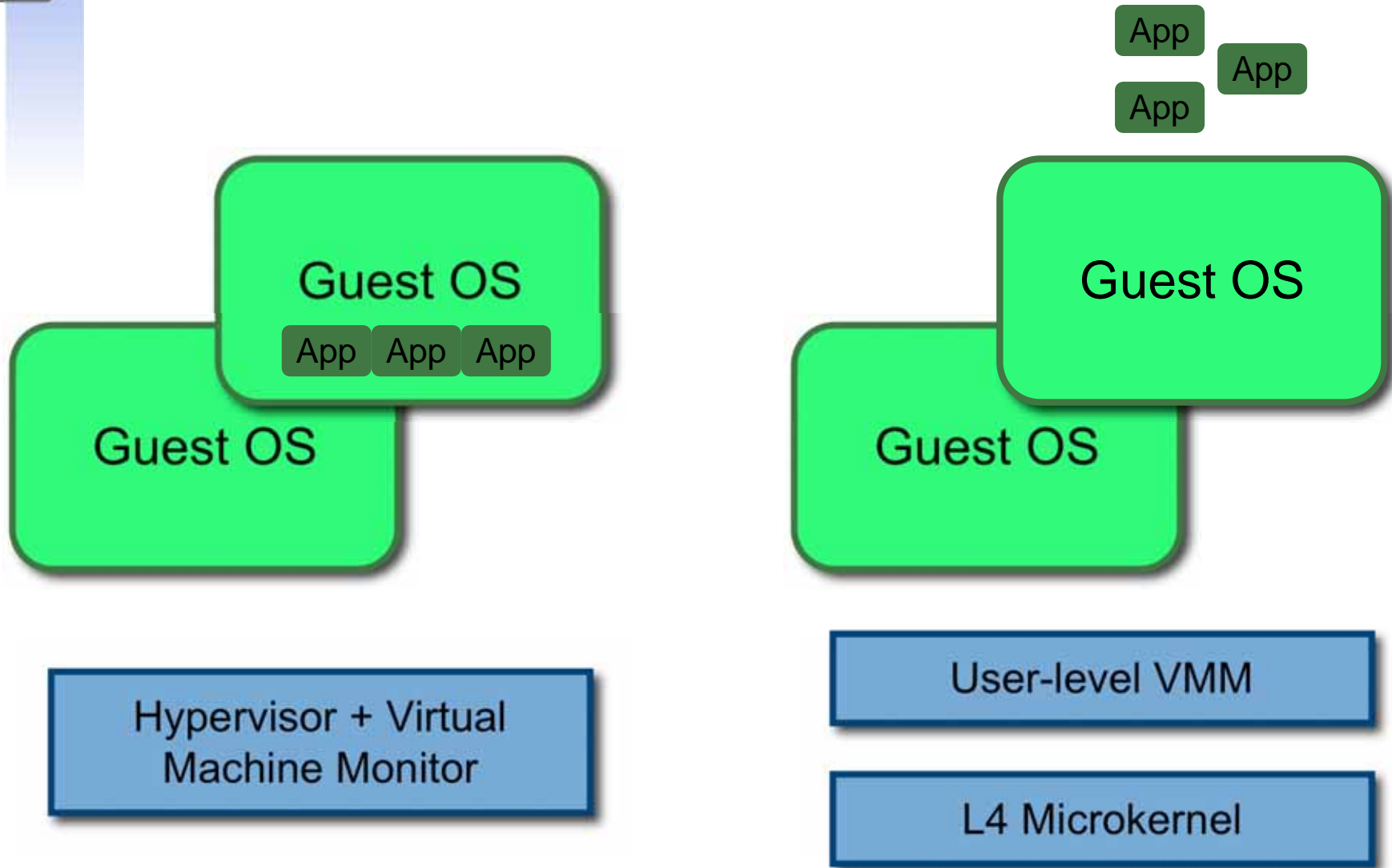


Virtual Machine (VM)

- 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

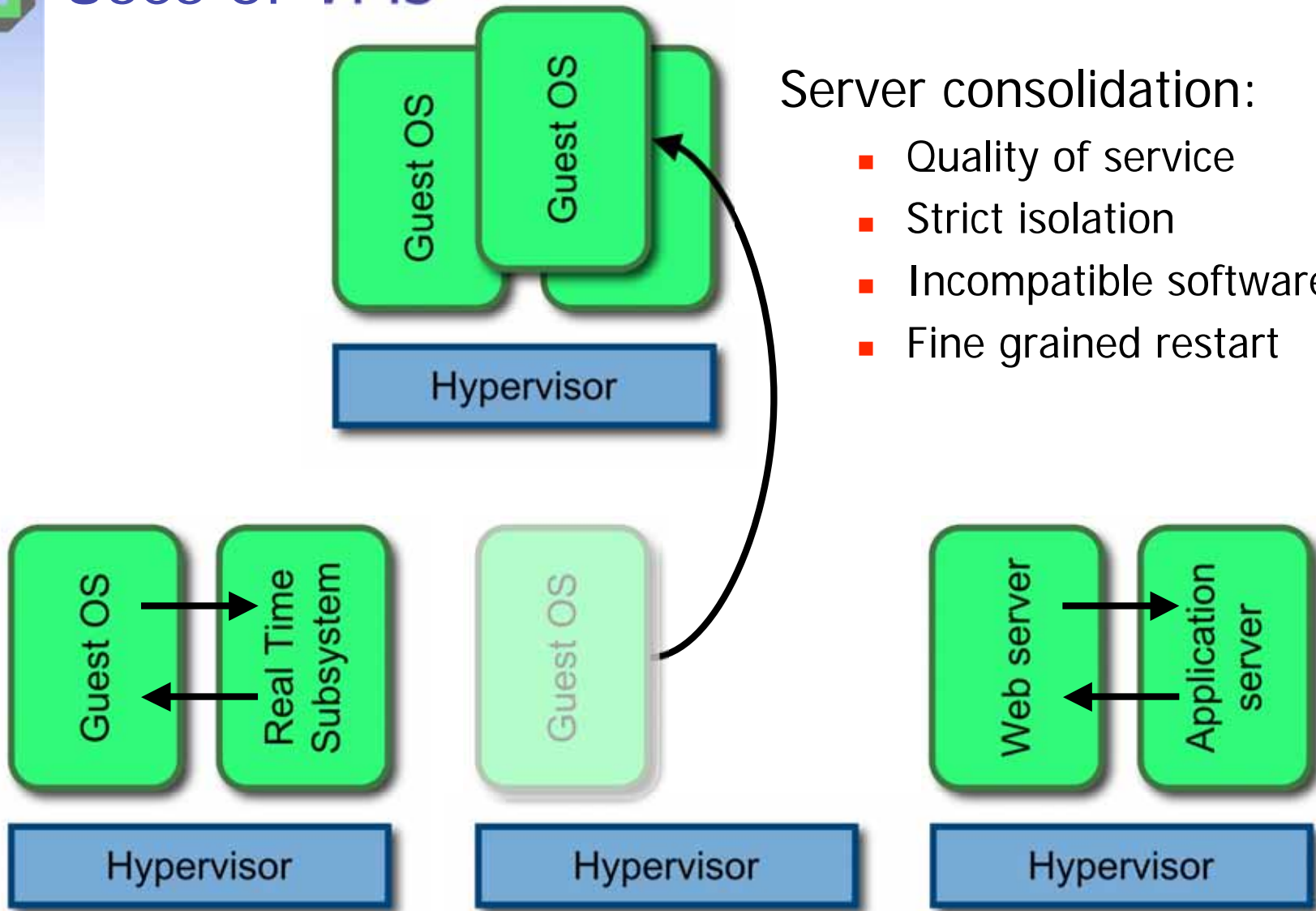


Basic VM Structure





Uses of VMs





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, ...**)

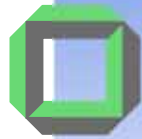
■ Tr...table
ke...
Current OS design is deficient.

The VM is a hack to fix the problems.

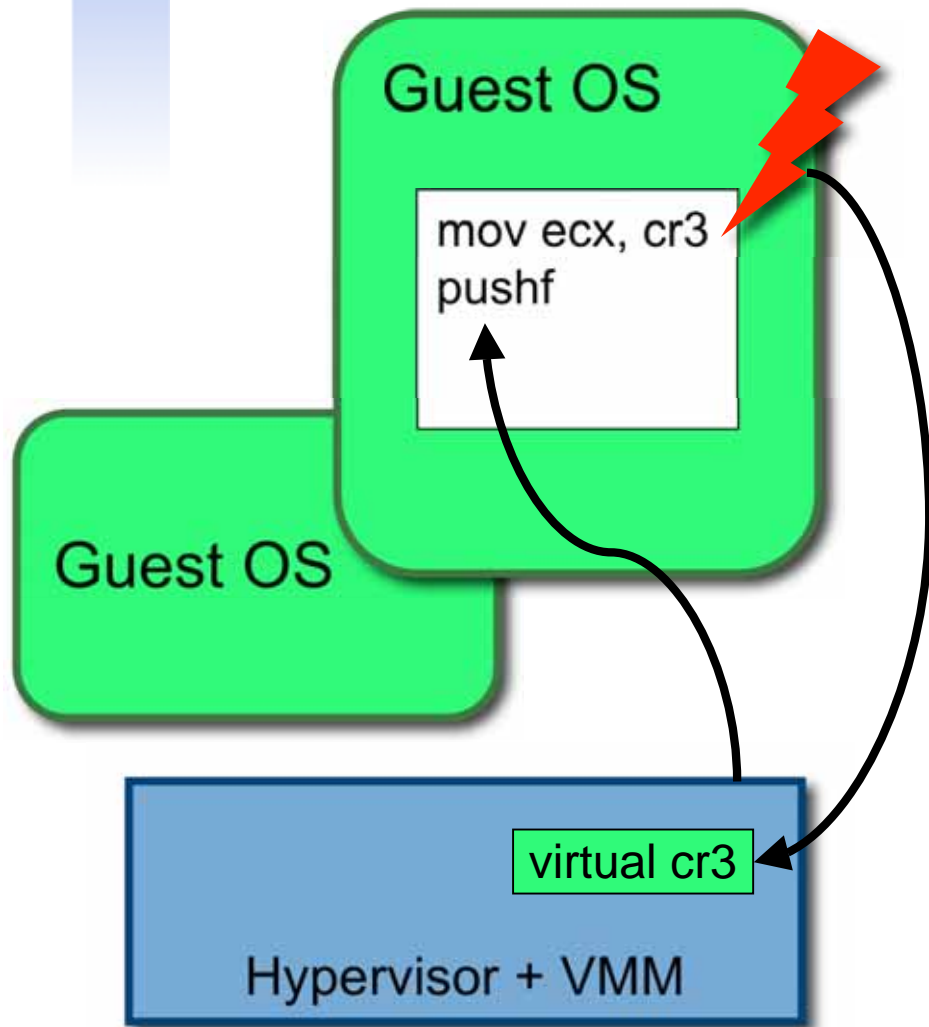


Virtualization Definitions

- **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

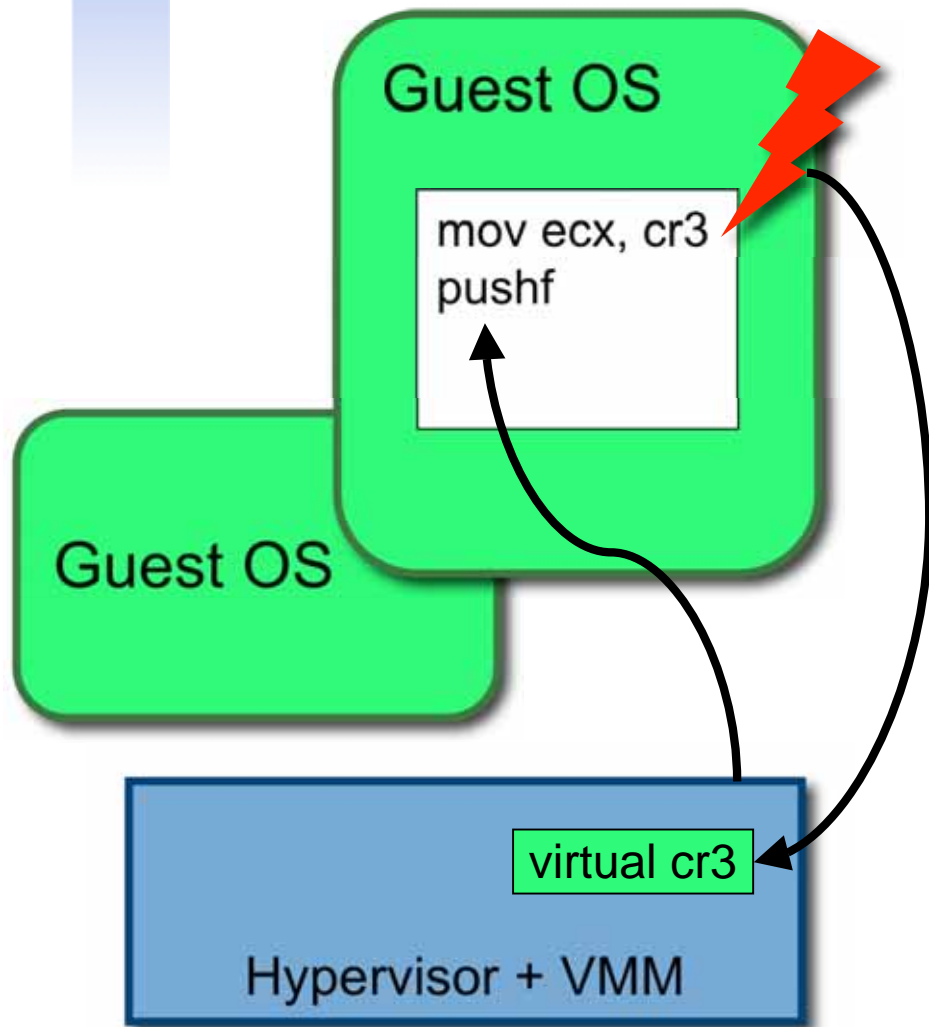


Advantages:

- ✓ Trustworthy emulation
- ✓ Hypervisor diversity
 - Raw hardware
 - VMware
 - VirtualPC
- ✓ Guest OS diversity
 - 1 x engineering effort
 - N x guest OS



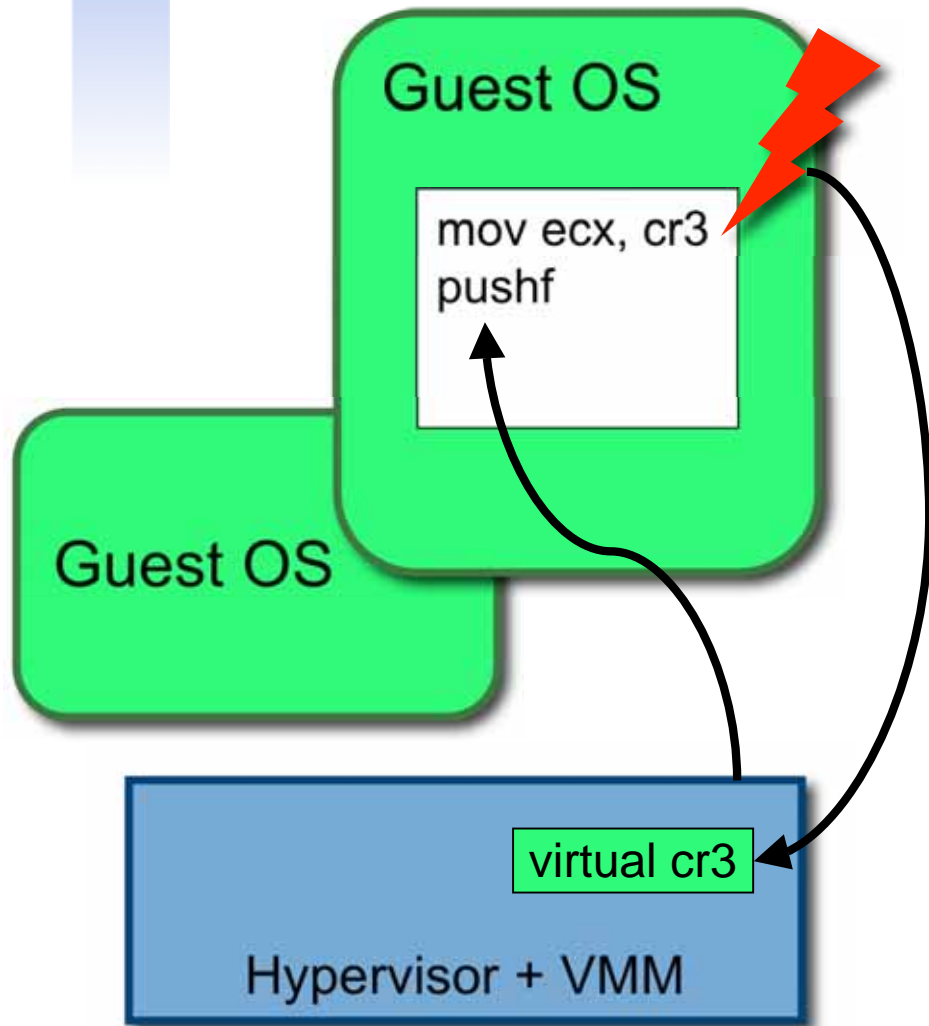
Pure Virtualization



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



Pure Virtualization

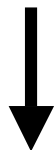


- Harsh requirements on hypervisor:
 - ✗ 4GB address space
 - ✗ Hide differences in CPUs [MMX vs. SSE2]
 - ✗ Burdensome legacy emulation
- High perf. devices:
 - ✗ Custom device drivers



Para-virtualization

```
mov ecx, cr3  
pushf
```



Manual source
modifications

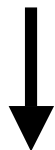
```
mov ecx, ebx  
mov $1, eax  
sysenter  
  
push virtual_flags
```

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



Para-virtualization

```
mov ecx, cr3  
pushf
```



Manual source
modifications

```
mov ecx, ebx  
mov $1, eax  
sysenter  
  
push virtual_flags
```

- Problems:
 - ✗ Migrations
 - Only possible between ABI compatible hypervisors
 - MMX vs. SSE2
 - ✗ No support for raw hardware
 - ✗ High perf. devices:
 - Custom device drivers



Transparent virtualization

```
mov ecx, cr3
pushf
```



Manual source
modifications

```
if (VMM) {
  call VMM_load_cr3
  call VMM_pushf
} else {
  mov ecx, cr3
  pushf
}
```

- VMware's VMI
- Replace sensitive instructions with hooks
 - ✓ Avoids costs of trapping
 - ✓ Hypervisor diversity
 - ✓ Potential industry standard
- Problems:
 - ✗ Manual engineering effort
 - ✗ Reduces trustworthiness of guest OS [unless done by kernel architects]



The Interface Difference

- **Pure virtualization**: lowest level interface
- **Para-virtualization**: high-level interface
 - ✓ Semantic assists
 - ✓ Higher performance
 - ✓ Supports unfriendly architectures
 - ✗ But hypervisor specific
- The semantic-assists cost modularity
 - The ISA is the modular interface



What is Virtualization About?

- Customization:
 - I want my features
 - You want your features
- Decentralized development:
 - Enhance a guest OS without the architect's permission
- Modularity:
 - Layering
 - Software reuse
- Conclusion: **para-virtualization is inferior**

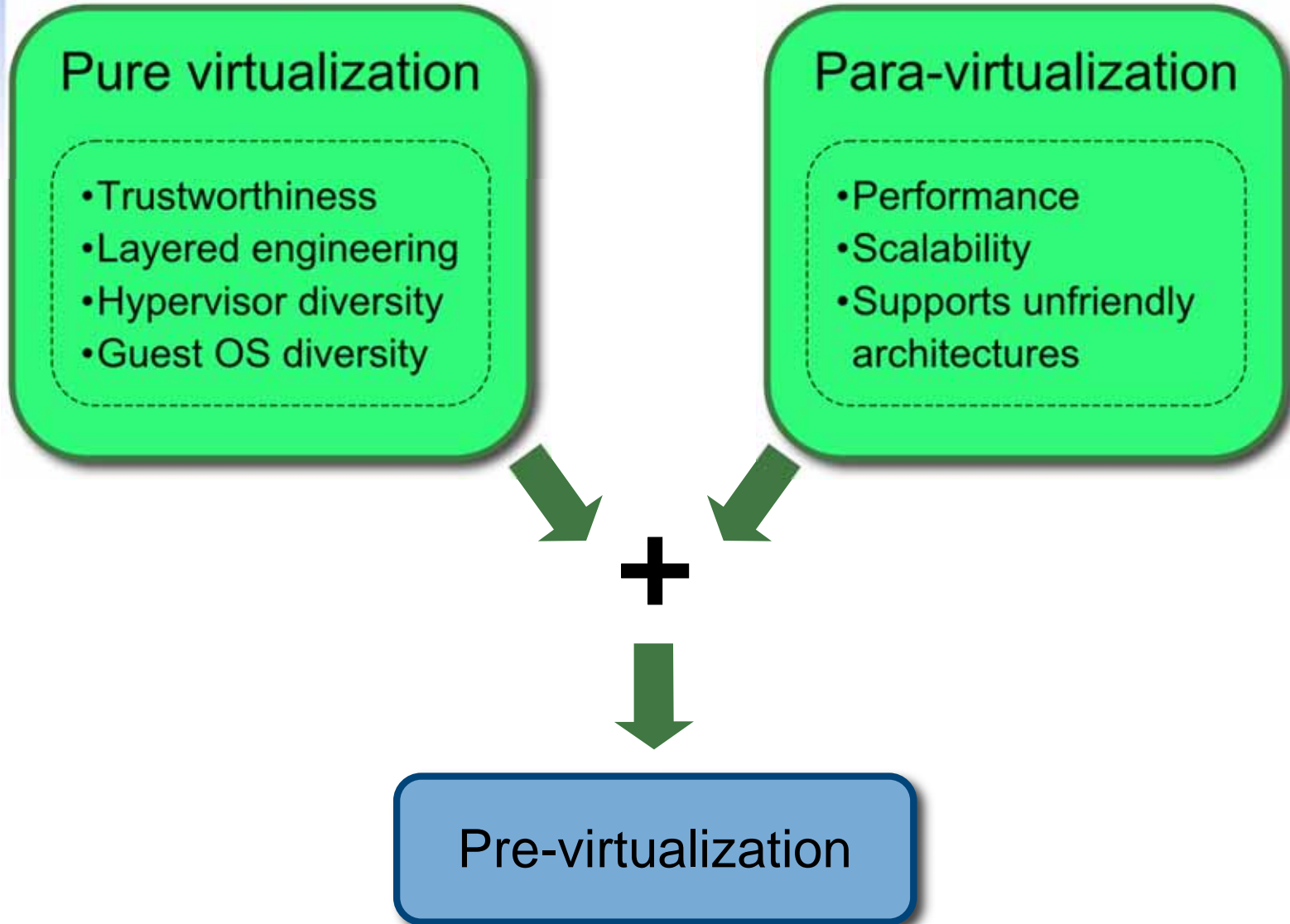


Why not standardize Para-virtualization?

- What is the perfect interface?
 - Ex: Xen changed their API: 2.0 \Rightarrow 3.0. But wasn't 2.0 already the ultimate?
- Domain specific trade-offs:
 - Classic: **security vs. performance**
- Different philosophies:
 - Ex: Run guest kernel at
 - user-level
 - intermediate privilege via x86's segments
- Conclusion: a standardized para-virtualization interface is not possible
 - **The platform ISA is a standard.**

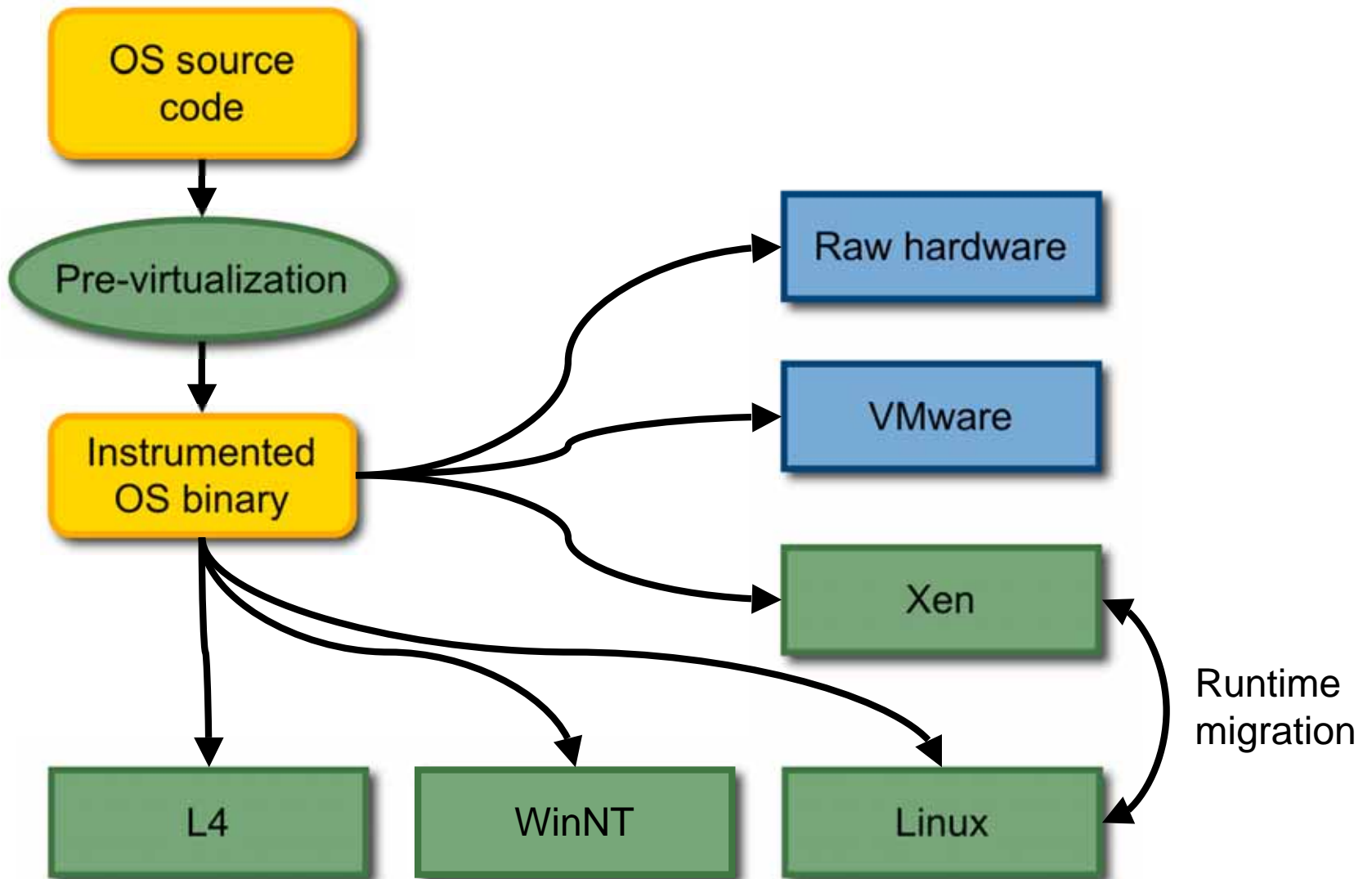


Uniting Two Worlds





Pre-virtualization Overview





Pre-virtualization Basics

```
mov ecx, cr3  
pushf
```

↓ Automated source
modifications [based on
Eiraku and Shinjo]

```
mov ecx, cr3  
nop  
nop  
  
pushf  
nop
```

Runtime
re-writing

```
mov ecx, ebx  
mov $1, eax  
sysenter  
  
push virtual_flags
```



Pre-Virtualization: Eiraku and Shinjo

- Static
 - Replace sensitive ops with emulation code
 - Supports a single hypervisor
- Trapping
 - Replace sensitive ops with trapping instructions
 - Via trapping, hypervisor diversity



Pre-Virtualization: Dynamic Rewriting

- Runs on raw hardware
 - Hypervisor diversity
1. Instruction padding
 - Pad sensitive ops with NOPs
 - Rewrite instruction at runtime
 - Special sequences: `sti ; hlt`
 2. Basic-block padding
 - Pad basic-blocks with NOPs
 - Rewrite entire basic block at runtime



Pre-Virtualization: Dynamic Rewriting

3. Basic-block bypass

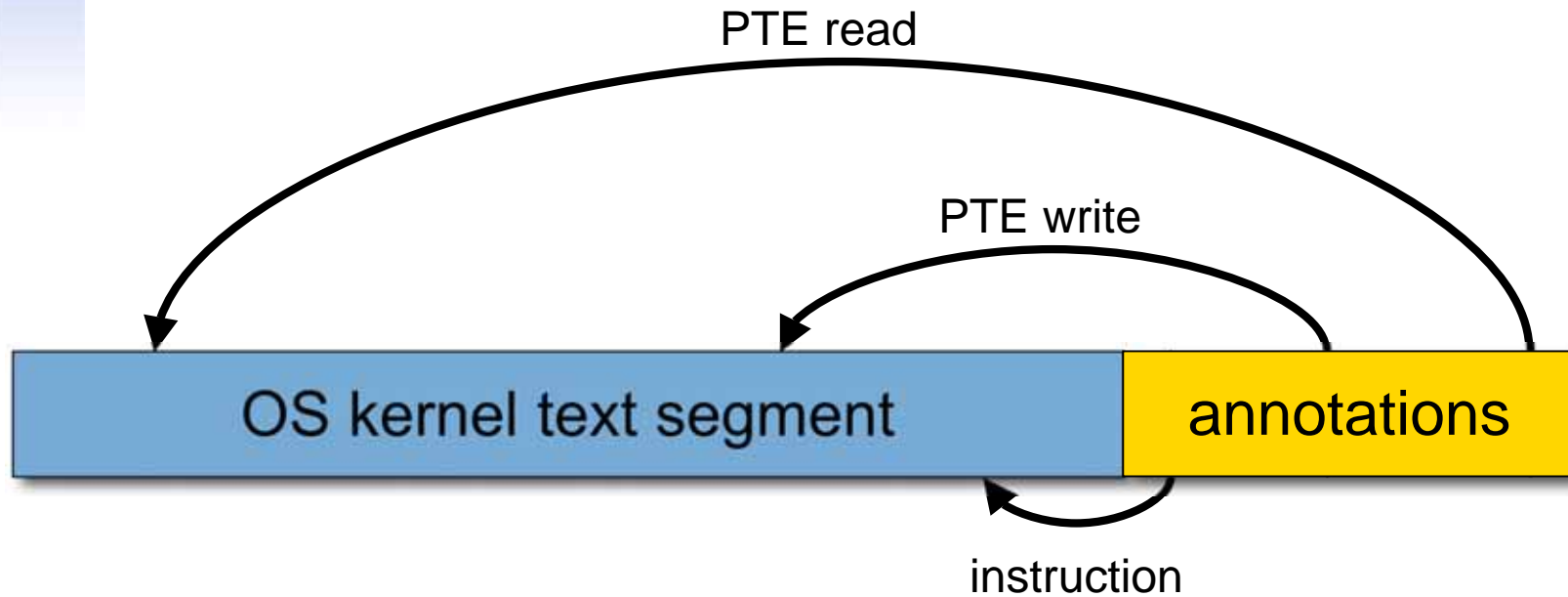
- Write a **jump** instruction at beginning of b.b., and jump to emulation code

4. Basic-block dynamic link

- Rewrite all jump sources to the basic-block, so that they jump to the replacement basic-block

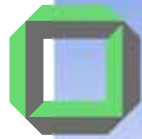


Dynamic Rewriting



Rich annotations possible:

- Register liveness
- Basic block boundaries
- Function boundaries



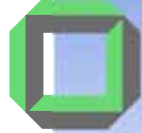
Phase 1: Afterburner



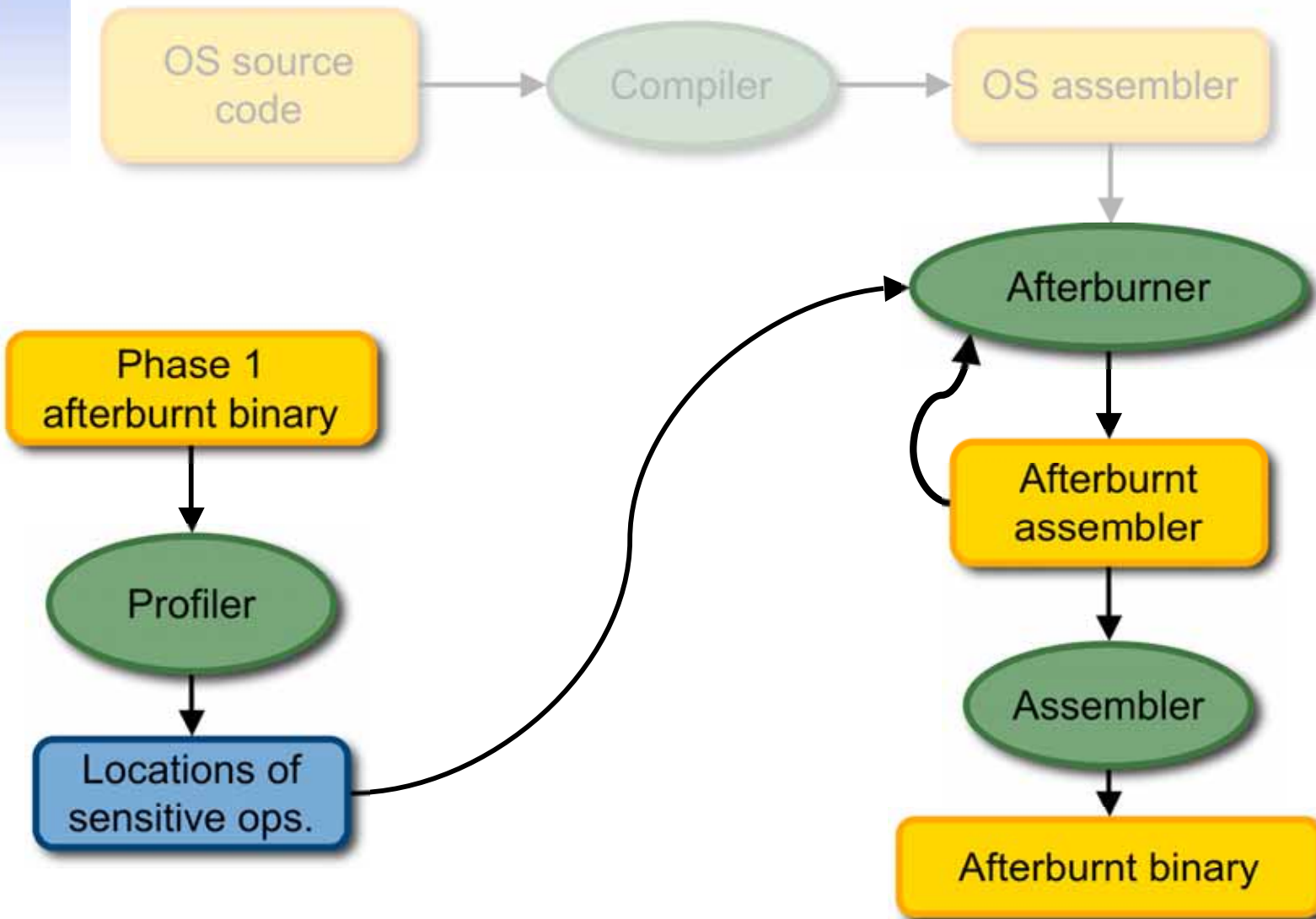


Phase 2: Profiler Feedback

- Virtualization-sensitive memory:
 - Page tables, tss, idt, gdt, etc.
 - Memory mapped devices
 - Important for performance
- Use a profile-feedback loop
 - Instrument and annotate the instructions that access sensitive memory
 - Or compiler data-flow analysis?



Phase 2: Profiler Feedback





Profiler Problems

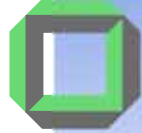
- Profiling is imperfect
 - Incomplete coverage
 - Captures only the common case
- In production environment:
 - Guest kernel could access sensitive memory
 - To detect sensitive accesses, use page faults
 - Easy to implement for devices
 - Suboptimal for guest's page tables
- Long term solution: data flow analysis



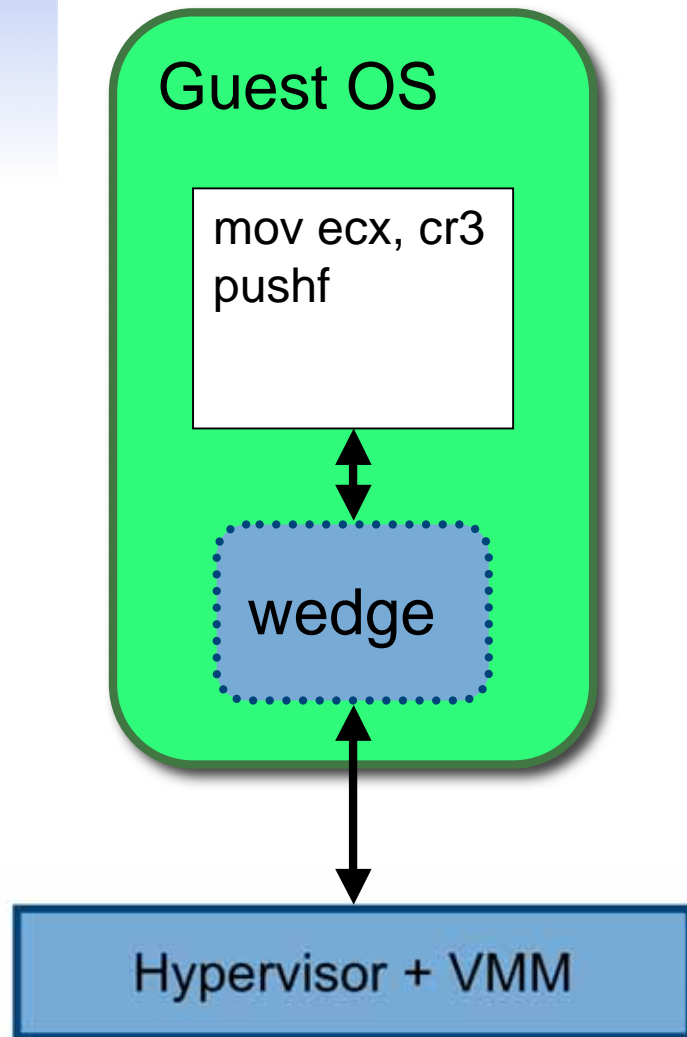
Is Pre-virtualization Good Enough?

- Pre-virtualization:
 - Instruction-level transformations
 - At best, basic-block transformations
- Para-virtualization:
 - High-level interface
 - Semantic assists

We need more from pre-virtualization!



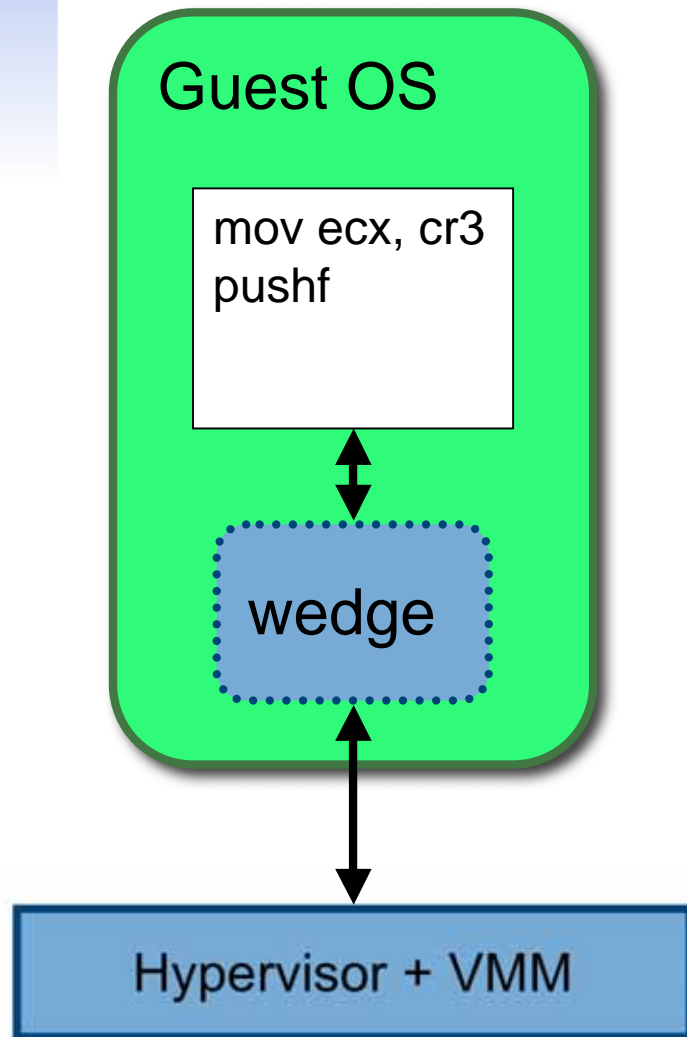
In-Place Virtualization: Approximating Para-virtualization



- The wedge creates a virtual CPU
- Invokes hypercalls only when necessary
- Frequent operations, such as cli/sti, emulated in the wedge
- Loose state consistency
- Use heuristics
- No licensing issues



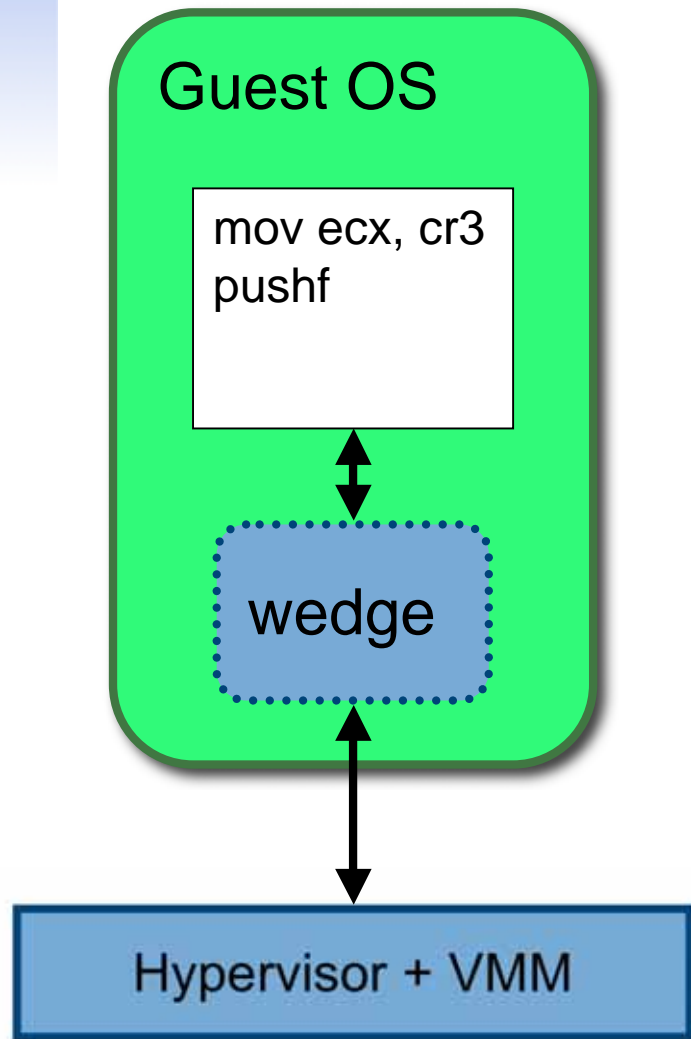
In-Place Virtualization: Examples



- Interrupts: accumulate in wedge, delivered on guest's `iret`, thus **cheap masking**
- PTE updates: synchronize page tables on TLB flushes
- PTE scanning: read referenced+dirty bits in batches
- `fork()` + `exec()` : heuristics



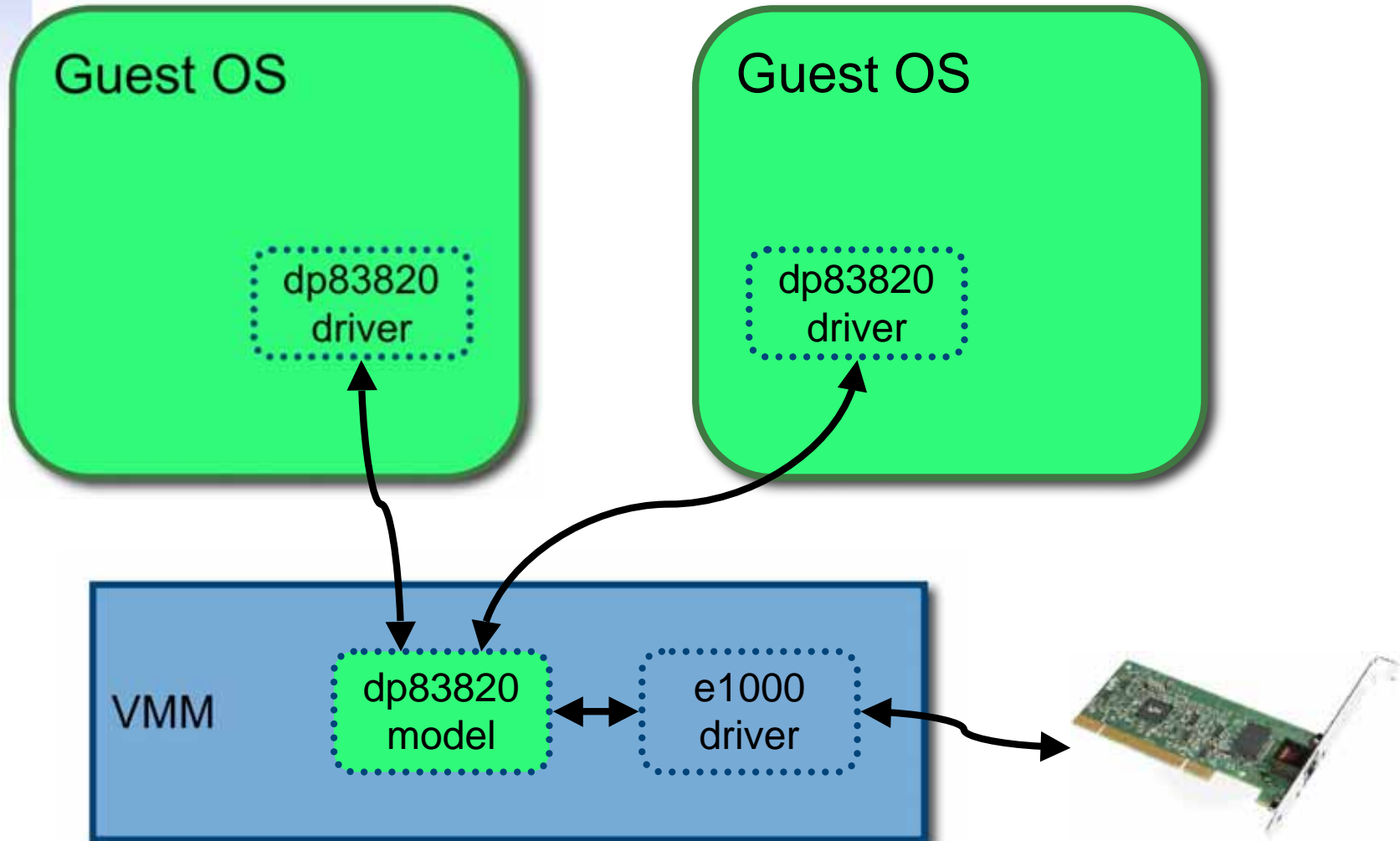
In-Place Virtualization: Problems



- Share the address space
 - May require support from guest kernel
- Large wedges:
 - User-level on Linux/BSD: 1GB
 - User-level on WinNT: 2GB
 - Requires relinking of the guest kernel
- Assumes valid guest stack

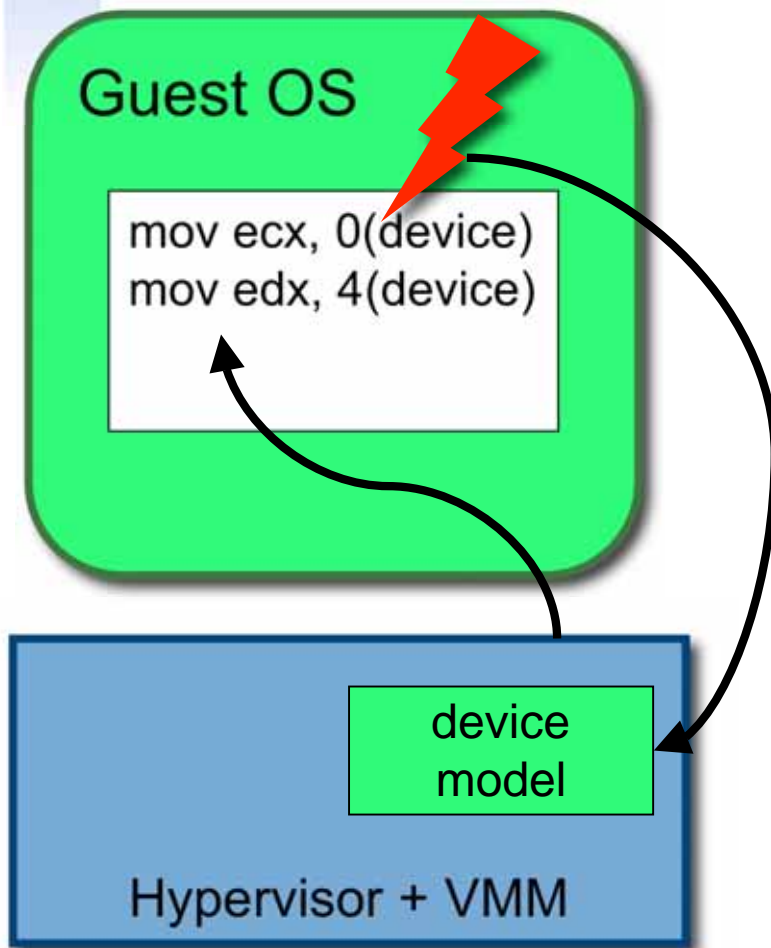


Device Access: Multiplexing





Device Emulation: Brute Force



- Trapping is expensive
- Trapping is frequent
 - Many small state updates



Device Emulation: Custom Drivers

Guest OS

```
mov ecx, ebx  
mov edx, ecx  
mov $10, eax  
sysenter
```

device
model

Hypervisor + VMM

- Traditional solution
- High performance
- Problems:
 - Specific to hypervisor
 - Different config for hypervisor and hardware
 - Unable to migrate between different hypervisors



Device Emulation: Pre-virtualization

- Rewrite sensitive ops
 - Replace with calls to the wedge
- Choose well behaved device interface:
 - network: [dp83820](#)
 - disk: [SATA?](#) [SCSI?](#)
- Can migrate between different hypervisors
- Good engineering leverage:
 - 1x engineering
 - Multiple guests



Device Emulation: Batching

- Device ops cause synchronization:
 - ✗ Hyper call
 - ✗ Potential address space switch
 - ✗ Expensive
- Solution:
 - ✓ Batching
 - ✓ Producer consumer rings
- Batching is difficult:
 - No semantic information
 - OS might send a file, but we see only packets



Device Emulation: Batching Heuristics

- We know:
 - When guest OS performs low-level ops
- Map low-level ops to high-level ops?
 - High device activity in interrupt handlers
 - When interrupt handler completes, device activity is paused briefly
 - Use `hlt` and `iret`



Transparent Virtualization: Safety Net

- Pre-virtualization is unfinished
 - Use manual annotations for memory operations
- Some hypervisors need high-level hooks for performance:
 - L4, Linux/BSD/WinNT user level
- Performance problems:
 - Signals
 - Process delete
 - Access to user memory
 - Thread local data

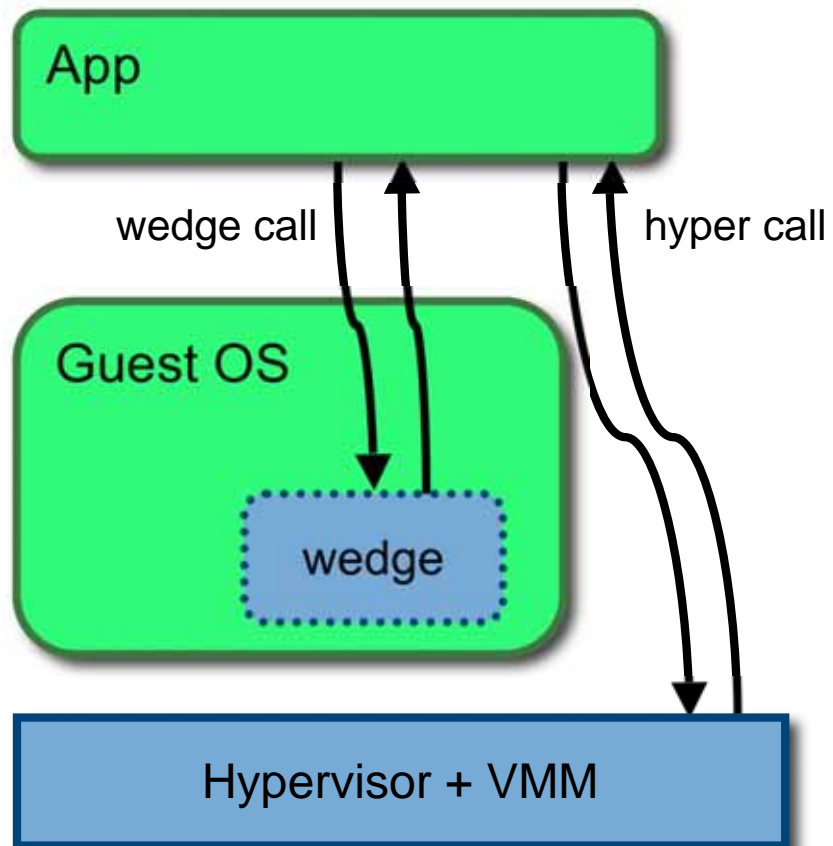


Transparent Virtualization: Safety Net

- Bypass platform init
 - BIOS
 - 16-bit code
- Make guest kernel relocatable
- Limited address space
 - Linux/BSD/WinNT user



Extensibility

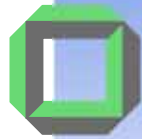


- Wedge call:
 - Management utilities
 - Introspection
 - Filters
 - Enhancements
- Hyper call
 - IPC
 - Management utilities
 - Enhancements



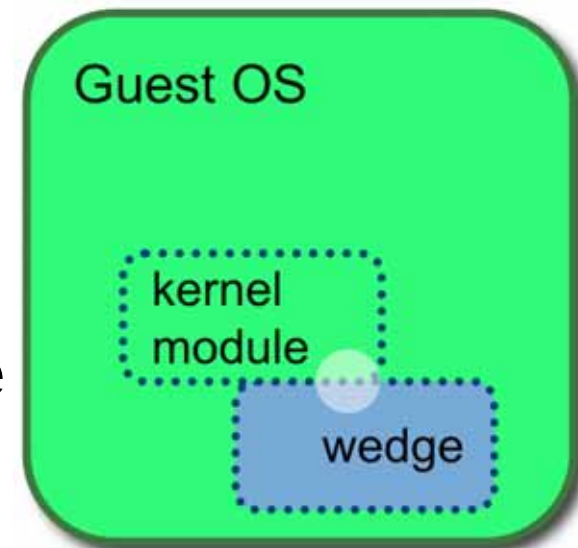
Loadable Kernel Modules

- Kernel modules are pre-virtualized
 - When are they rewritten?
 - The rewriter needs the annotations ...
- A. **User-level utility**
 1. Rewrites the module via wedge calls
 2. Guest kernel loads the module
- B. **Transparent kernel hook**
 - Unmodified user-level tool chain
 - Guest kernel invokes rewrite hook



Loadable Wedge Modules

- Efficient interaction with the wedge:
 - Permit kernel modules to link against the wedge
 - Avoids expensive hyper calls
- Requirement:
 - Wedge symbol resolution
- Uses:
 - Memory ballooning module
 - Virtual device drivers





Evaluation

- Hypervisors:
 - L4Ka::Pistachio microkernel
 - Xen 2.0.2
- Guest OS: Linux 2.6.9
- Comparison:
 - afterburnt Linux on hypervisors
 - para-virtualized Linux: [L4Linux](#), [XenoLinux](#)
 - native Linux
 - afterburnt Linux on hardware

(all execute with direct device access)



Implementation

- Simple implementation in C++
 - Contrast para-virtualization: must use language of the guest OS
- Reserved 64MB of address space
- Hooks added to guest OS:
 - page table manual annotations
 - DMA translation manual hooks
 - thread-local-storage
 - thread exit
 - user access

} L4 optimizations



Benchmark: Netperf

- System-under-test:
 - 2.8 GHz Pentium 4
 - gigabit ethernet
 - 256 MB of RAM, 64 MB RAM disk
- Client system
 - 1.4 GHz Pentium 4
 - gigabit ethernet
 - native Linux 2.4
 - 256 MB of RAM, 64 MB RAM disk
- Netperf send + receive: one gigabyte



Netperf Results

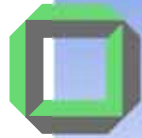
	Send		Receive	
	MB/s	CPU	MB/s	CPU
hardware	108.3	28.8%	97.6	35.4%
afterburnt hardware	108.5	27.4%	97.5	33.8%
XenoLinux	108.4	33.8%	97.6	41.6%
afterburn Xen	107.9	33.5%	96.6	41.1%
L4Ka::Linux	96.98	34.4%	97.5	35.6%
afterburnt L4	108.3	30.1%	97.5	37.2%

95% confidence interval is $\pm 0.21\%$

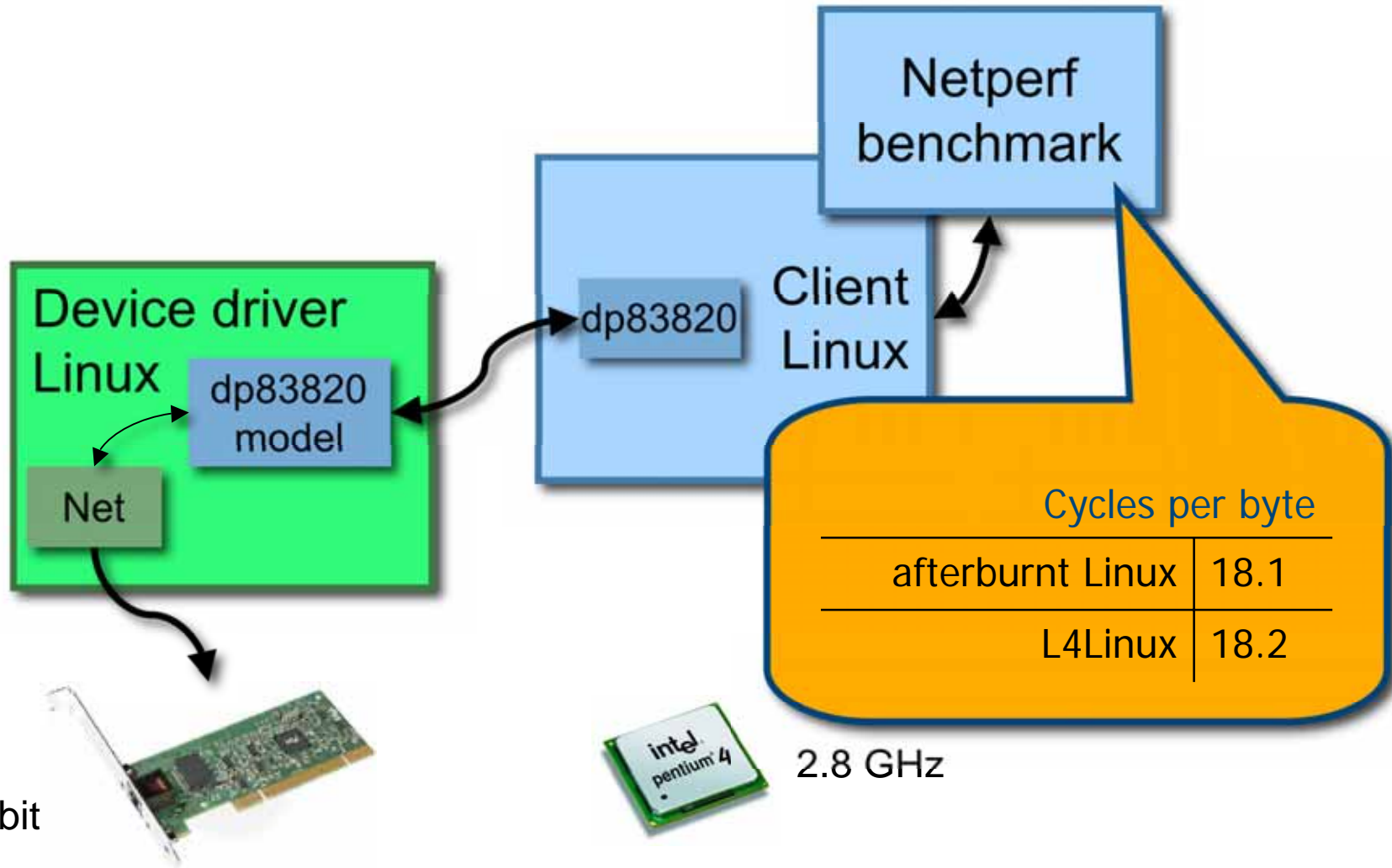


Netperf Instruction Expansion

operation	count	per interrupt
cli	6772992	74
sti	1572769	17
pushf	6715828	73
popf	5290060	58
hlt	91528	1
iret	184760	2
write ds	369520	4
write es	369520	4
read ds	184760	2
read es	184760	2
read fs	182866	2
read gs	182866	2
port out	278737	3



Device Pre-Virtualization Performance





Implementation: Afterburner

- Assembler macros
 - Walks the AST
 - But too inflexible (ex: sti ; hlt)
- Added new assembler operator: !
 - Prevents recursive macro expansion

```
.macro pushfl
9998: !pushfl ; nop ; nop ; nop ; nop ; nop
9999:
.pushsection .afterburn
.long 9998b
.long 9999b
.popsection
.endm
```



Implementation: Architecture

- Front-ends:
 - Architecture specific
 - x86, ia32e, Itanium, PowerPC
- Back-ends:
 - Hypervisor specific
 - L4, Xen, Linux-user, WinNT-user
- Code reuse



Lines of Code

type	headers	source
common	686	1380
device	650	1476
x86	815	4436
L4	615	3655
Xen	677	2670
Linux	168	3405



Join the Fun

- Lots of pre-virtualization projects
- <http://l4ka.org>
- Source code available on the web
 - includes a 'make world'
 - retrieves all packages
 - configures all packages
 - builds everything
 - prepares for QEMU emulator