



Integrating Paravirtualization into the Linux Kernel

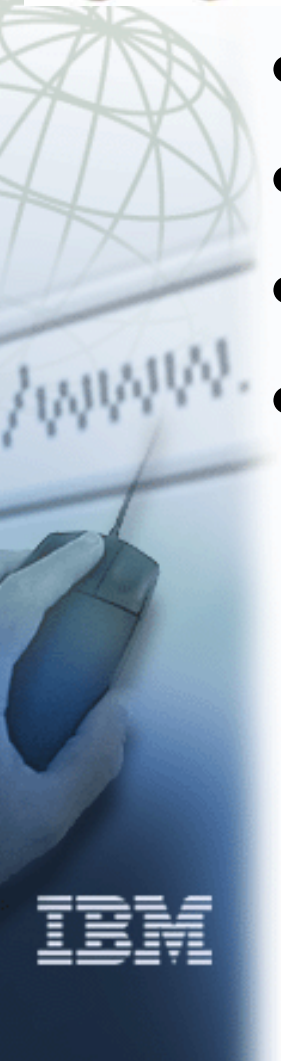
Rusty Russell
IBM Linux Technology Center
(OzLabs)

IBM



Contents

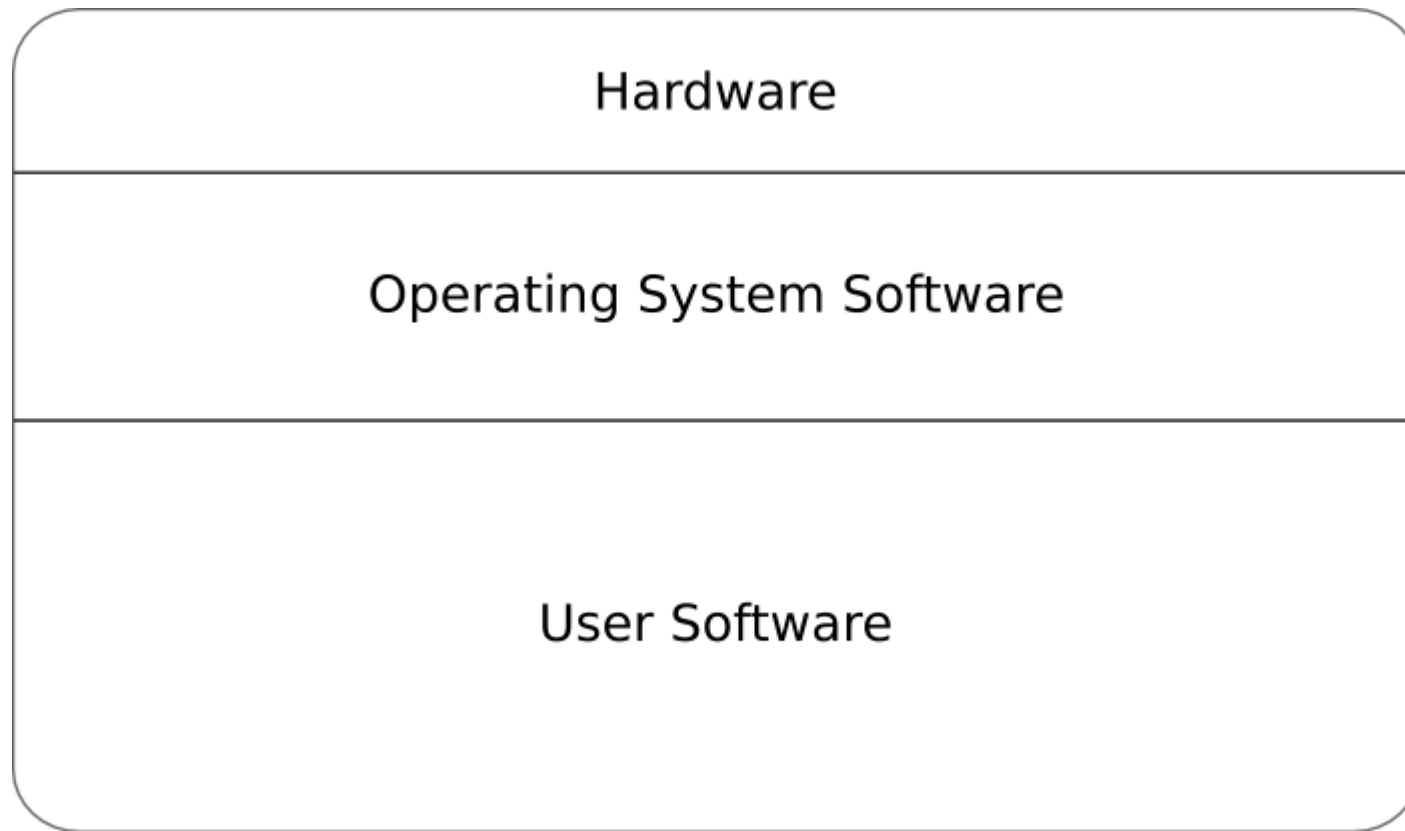
- Introduction to Paravirtualization
- The Choices: Xen, VMI, Native...
- The Solution
- The Implementation
- Future Work





Introduction to Paravirt.

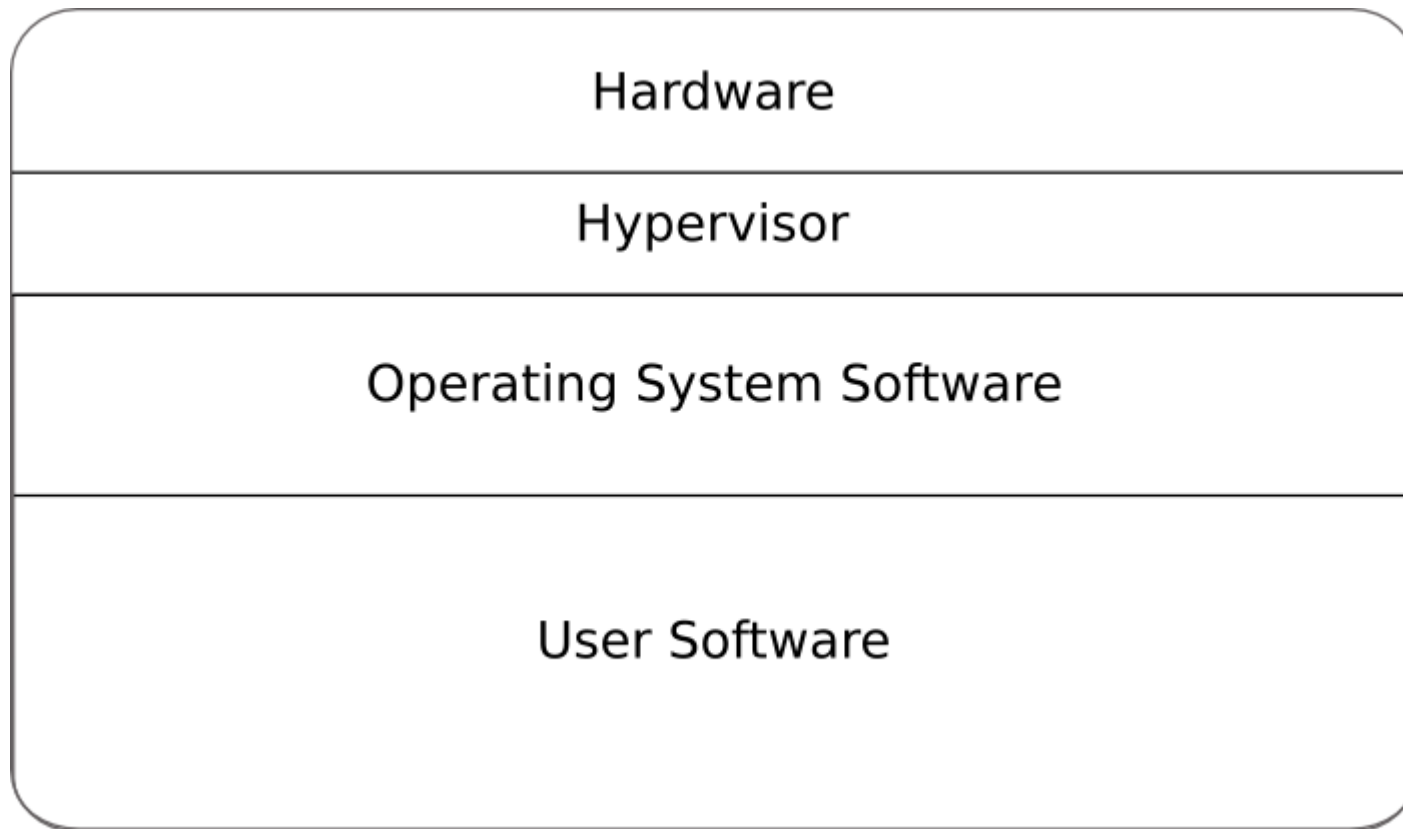
- Operating System on normal hardware:





Introduction to Paravirt.

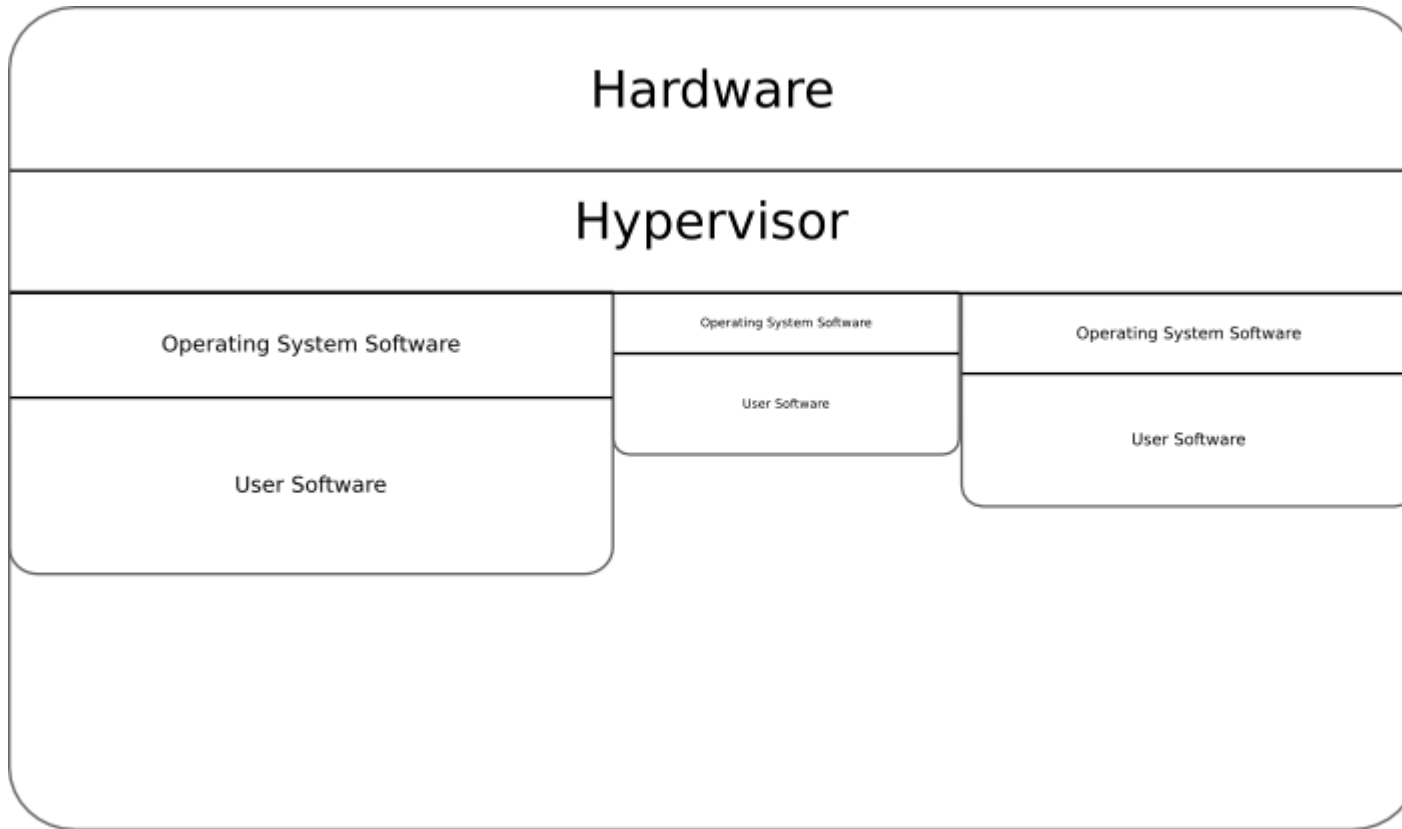
- Operating System “virtualized” under hypervisor:





Introduction to Paravirt.

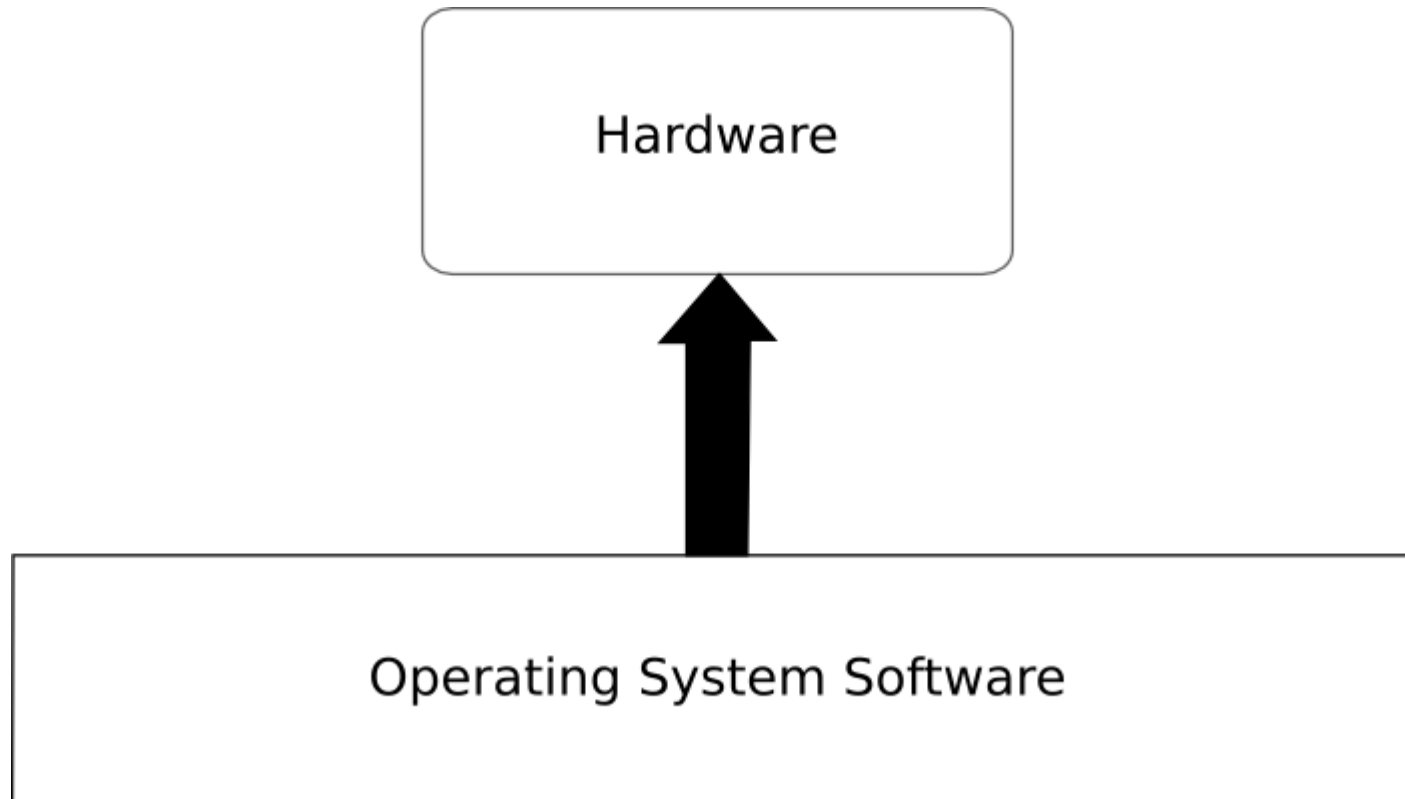
- Operating System “virtualized” under hypervisor:





Introduction to Paravirt.

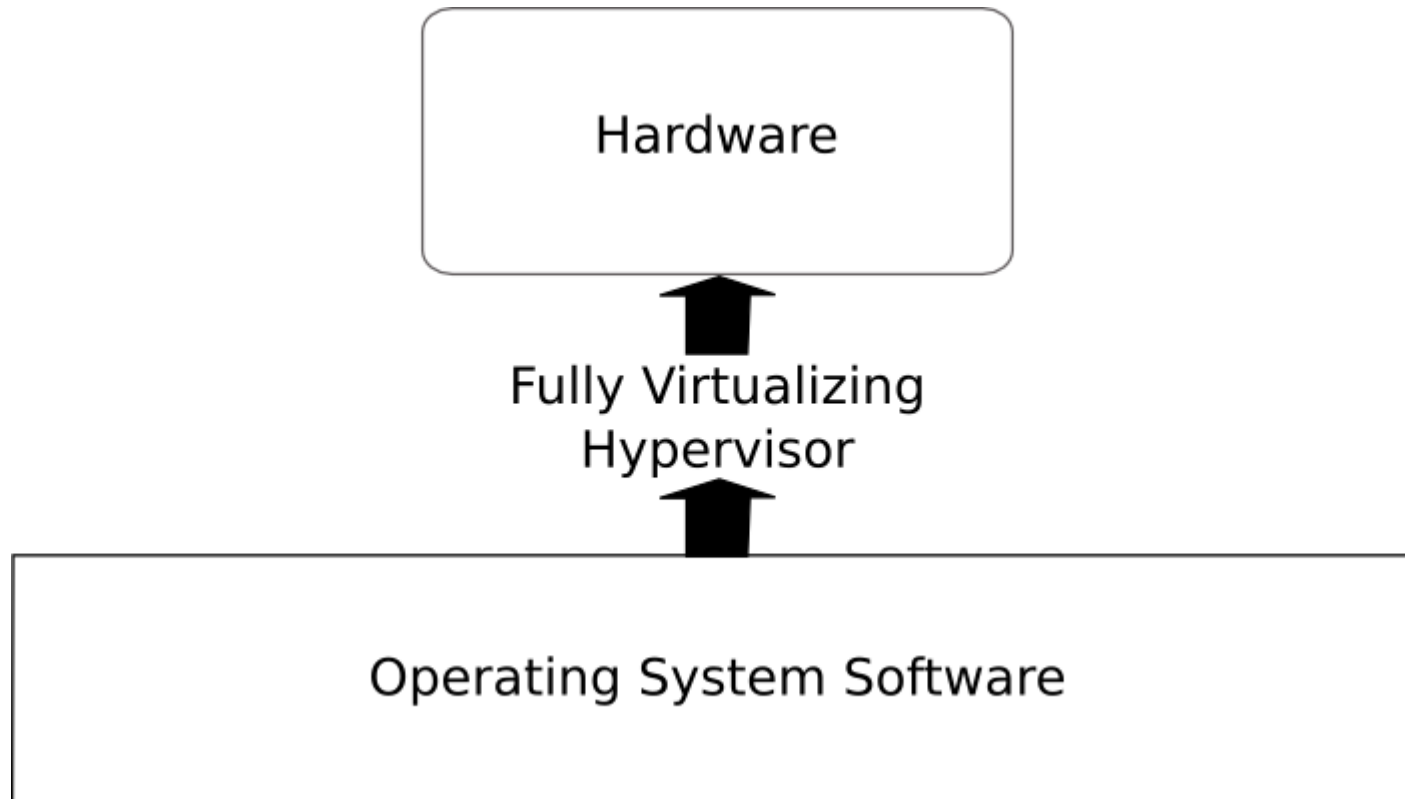
- Full virtualization is easy (for Linux!)





Introduction to Paravirt.

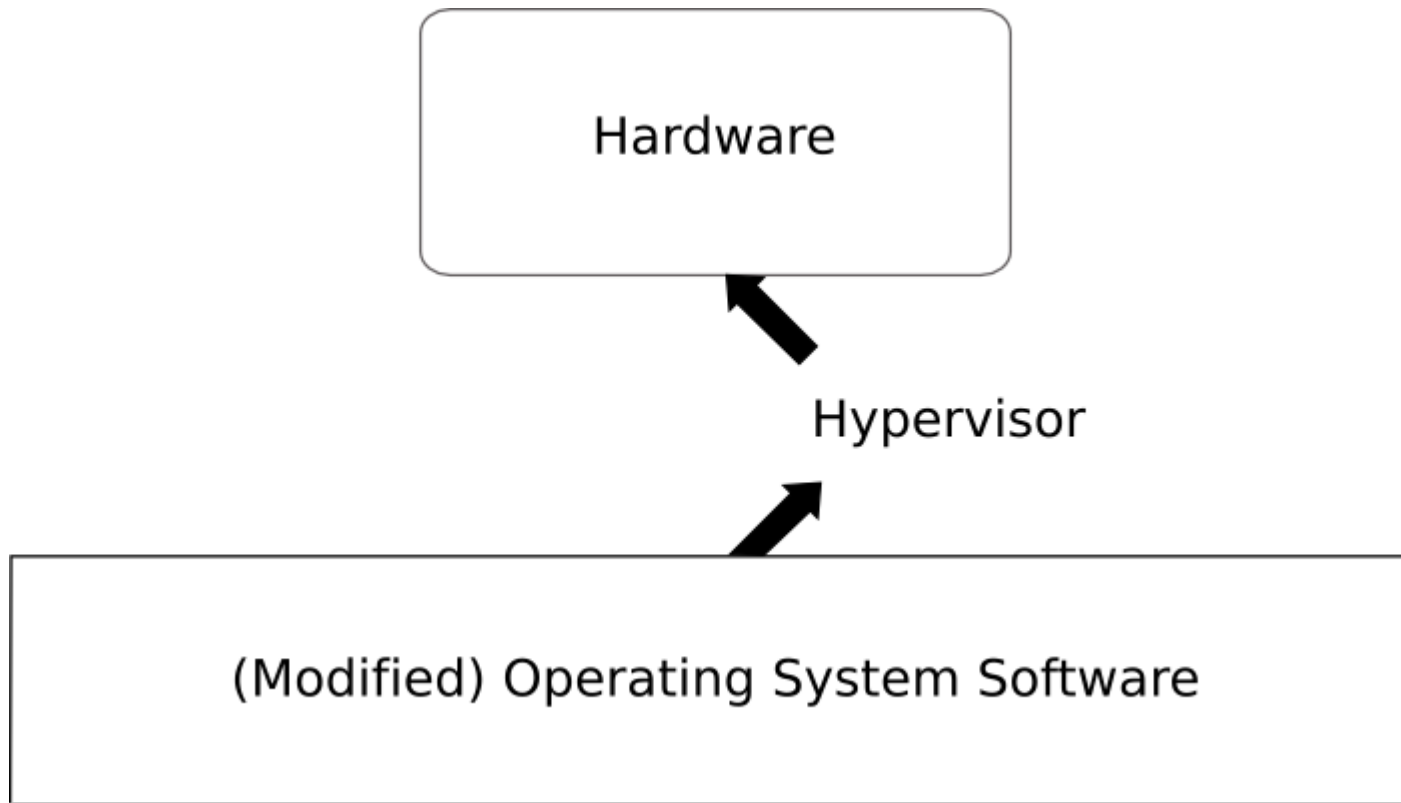
- Full virtualization is easy (for Linux!)





Introduction to Paravirt.

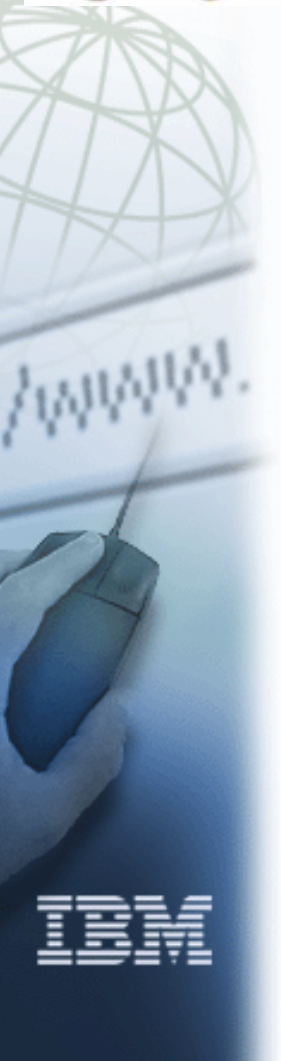
- *Paravirtualization* is more efficient because Operating System cooperates:





The Choices

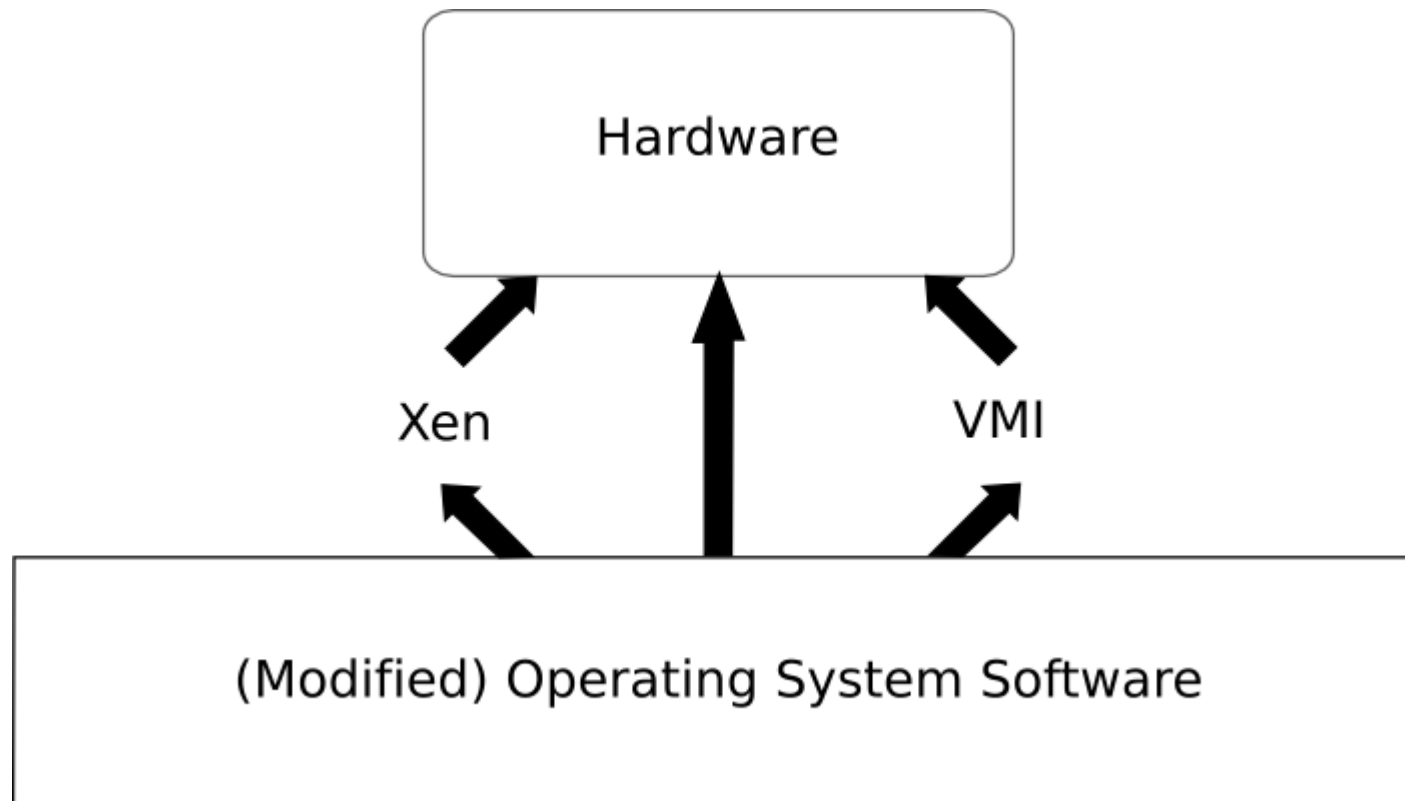
- Various x86 hypervisors exist for Linux:
 - Xen
 - Xen-specific Linux modifications
 - VMWare
 - “VMI” proposed generic interface
 - Native hardware
 - Linux must also support this!





The Choices

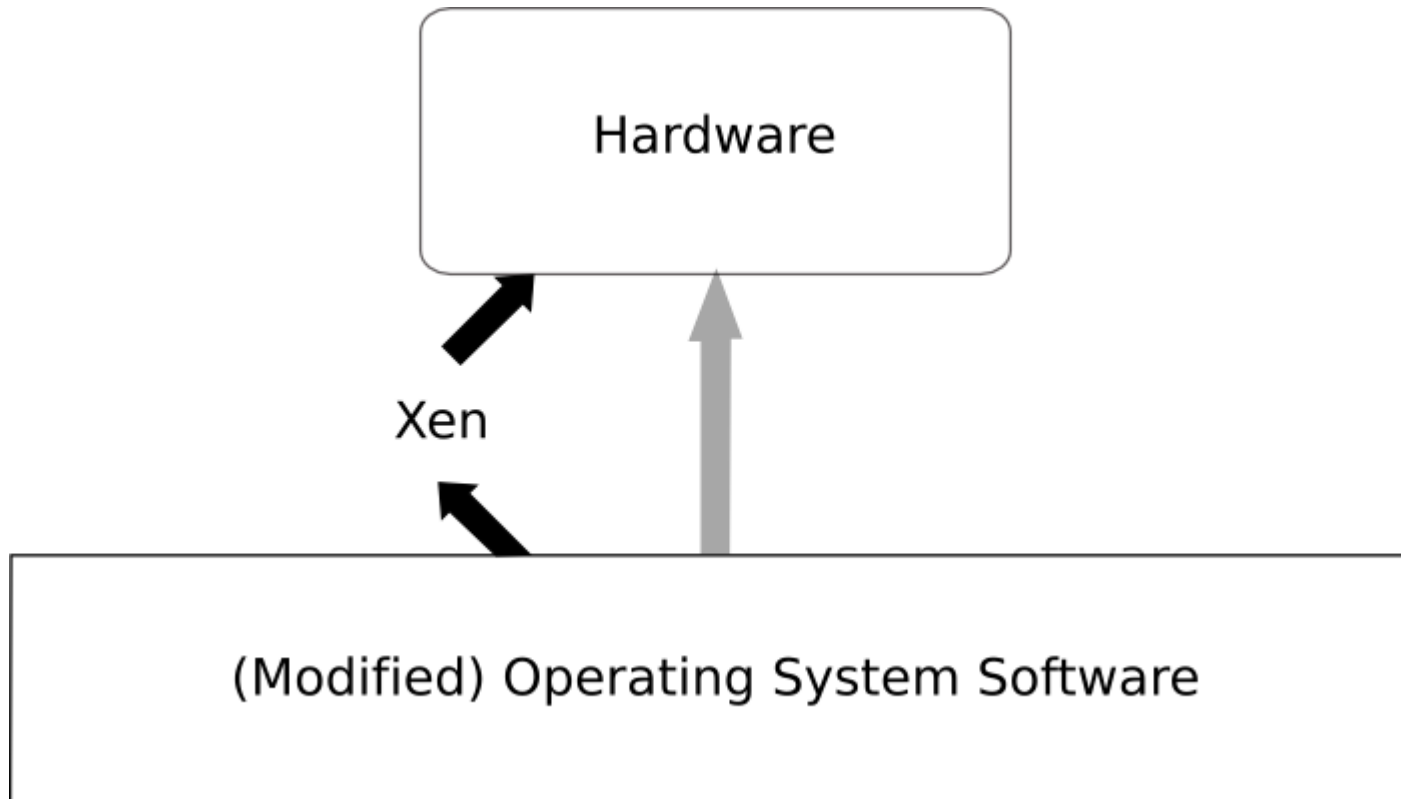
- Various x86 hypervisors exist for Linux: Xen, VMI, Native





The Choices: Xen Patches

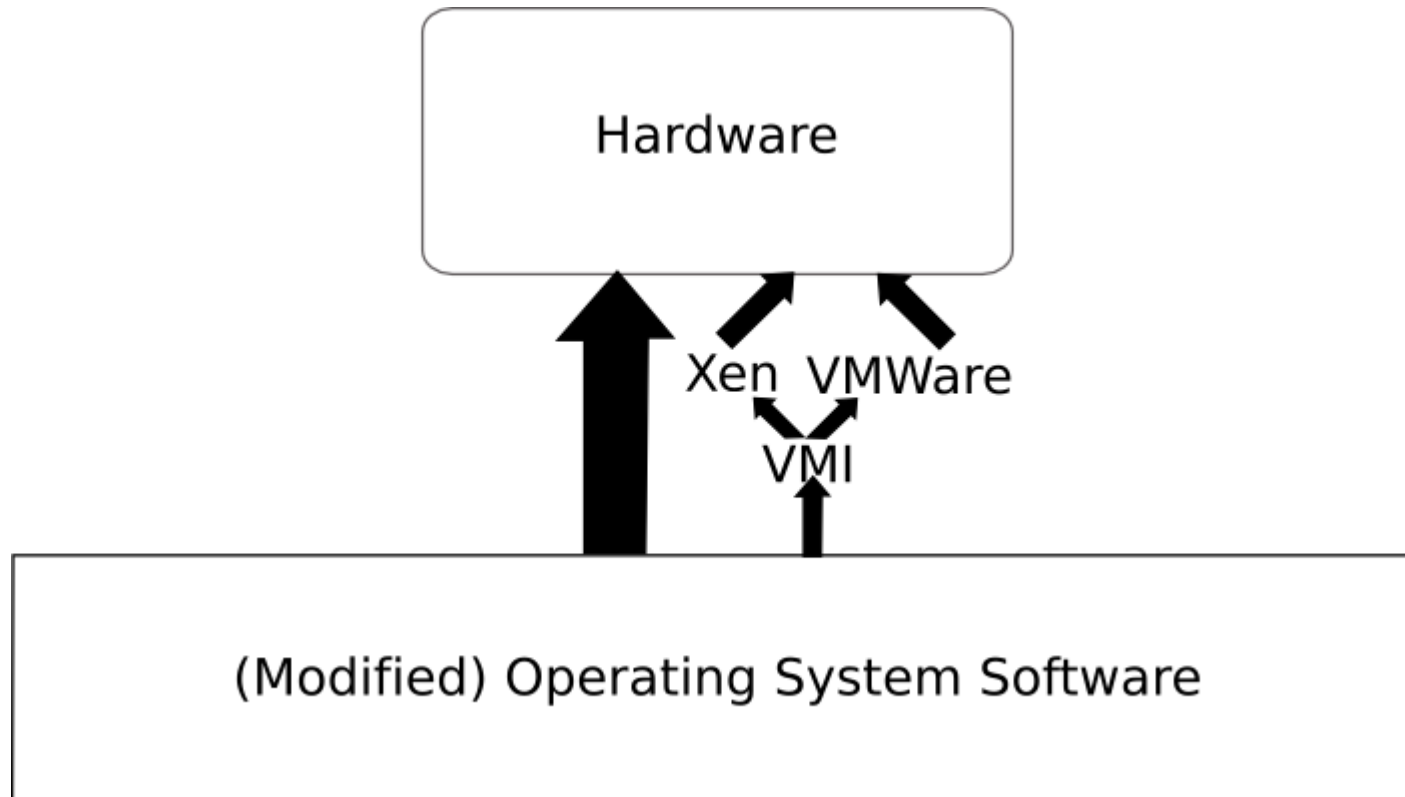
- Chris Wright's CONFIG_XEN patches:





The Choices: VMI Patches

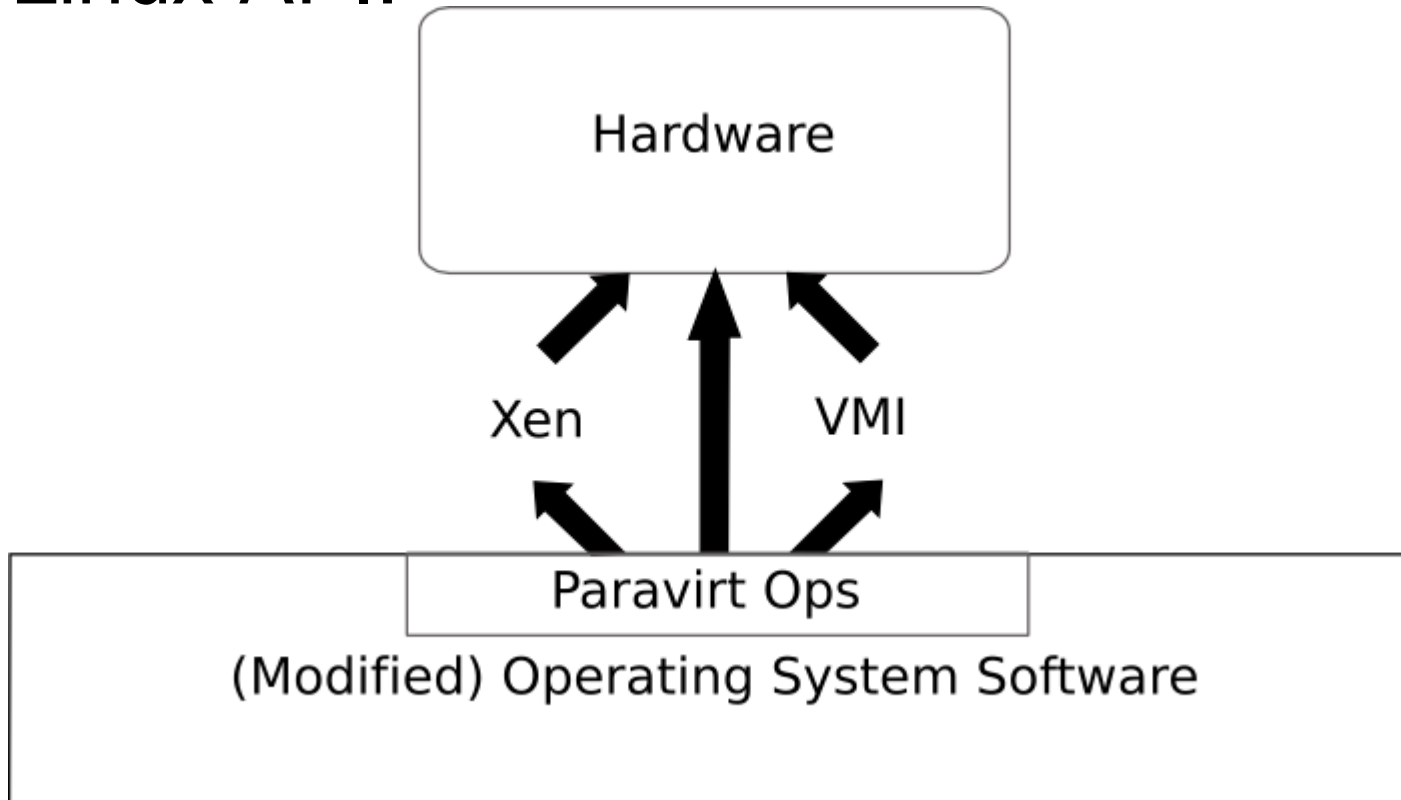
- Zach Amsden's VMI patches:





The Solution

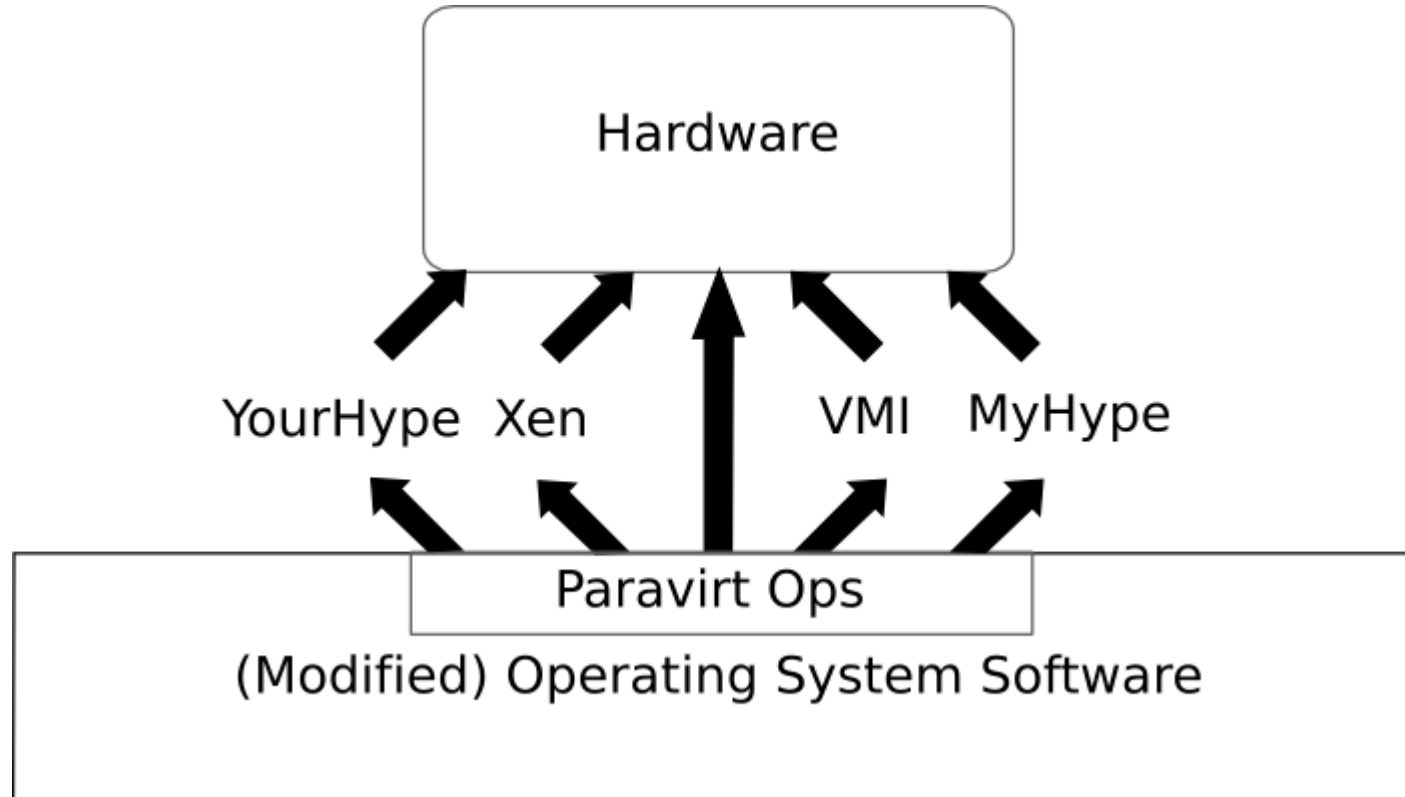
- If we can't agree on a single ABI for all hypervisors, at least we can have a Linux API:





The Solution: Paravirt Ops

- I expect you all to write hypervisors!





The Implementation

- “struct paravirt_ops”
 - Function pointers for every sensitive instruction.
 - Similar to PowerPC's “ppc_md”
- Each hypervisor replaces these with the operations for that interface.
 - Designed to be as easy to port Linux to a new hypervisor as writing new Linux driver.





The Implementation

- struct paravirt_ops contains function pointers, eg:
 - unsigned long (fastcall *save_fl)(void);
void (fastcall *restore_fl)(unsigned long);
void (fastcall *irq_disable)(void);
void (fastcall *irq_enable)(void);
 - Currently 75 functions
 - Not all hypervisors need all of them!



The Implementation

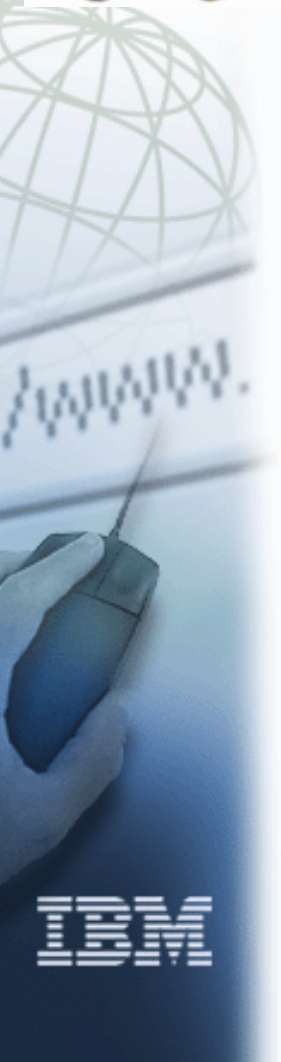
- But indirect function calls can be slow:
 - up to 20% slowdown on microbenchmarks (Imbench) on 3GHz P4.
- We can regain this speed by using runtime patching of call sites.
 - Only need to patch interrupt operations
 - These are well over 90% of operations
 - We already do something similar for SMP kernels on single processor systems.





The Implementation

- “struct paravirt_ops” provides a method for patching:
 - Returns length of patch, so remainder can be padded with NOOPs.
 - unsigned (*patch)(u8 type, u16 clobber, void *firstinsn, unsigned len);
 - type describes which operation
 - clobber indicates what registers you can use
 - firstinsn is pointer to instructions
 - len is length of instructions





The Implementation

- Unpatched code looks like this:

```
– asm volatile("pushl %%ecx; pushl %%edx; "  
               "call *%0; "  
               "popl %%edx; popl %%ecx",  
               : : "m" (paravirt_ops.irq_disable)  
               : "memory", "eax", "cc");
```

- This calls `paravirt_ops.irq_disable()`.
- It is allowed to use the `%eax` register.
- It is 10 bytes long:
 - 2 push (2x1 bytes)
 - 1 indirect call (6 bytes)
 - 2 pops (2x1 bytes)



The Implementation

- **Example: (native)**

```
#define DEF_NATIVE(name, code) \
    extern const char start_##name[], end_##name[]; \
    asm("start_" #name ": " code "; end_" #name ":")
DEF_NATIVE(cli, "cli");
DEF_NATIVE(sti, "sti");
DEF_NATIVE(popf, "push %eax; popf");
DEF_NATIVE(pushf, "pushf; pop %eax");
DEF_NATIVE(pushf_cli, "pushf; pop %eax; cli");
DEF_NATIVE(iret, "iret");
DEF_NATIVE(sti_sysexit, "sti; sysexit");

static const struct native_insns
{
    const char *start, *end;
} native_insns[] = {
    [PARAVIRT_IRQ_DISABLE] = { start_cli, end_cli },
    [PARAVIRT_IRQ_ENABLE] = { start_sti, end_sti },
    ... };
```



The Implementation

- **Example: (native)**

```
unsigned native_patch(u8 type, u16 clobbers,
                    void *insns, unsigned len)
{
    unsigned int insn_len;

    /* Don't touch it if we don't have a replacement */
    if (type >= ARRAY_SIZE(native_insns) ||
        !native_insns[type].start)
        return len;

    insn_len = native_insns[type].end
               - native_insns[type].start;
    if (len < insn_len)
        return len;

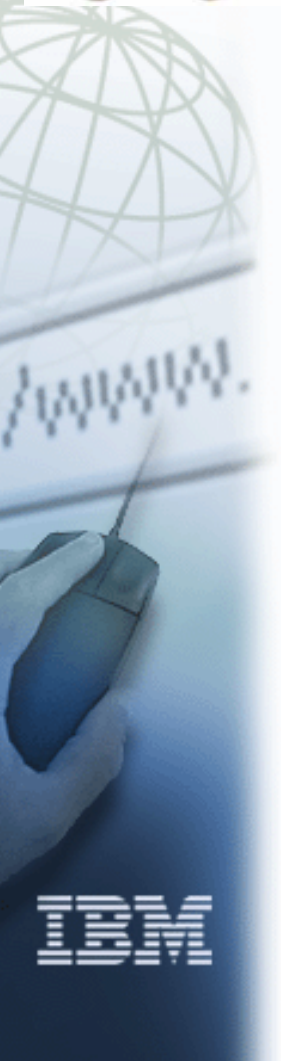
    memcpy(insns, native_insns[type].start, insn_len);
    return insn_len;
}
```



The Implementation

- Example: (Xen)

```
unsigned native_patch(u8 type, u16 clobbers,  
                     void *insns, unsigned len)  
{  
    return len;  
}
```





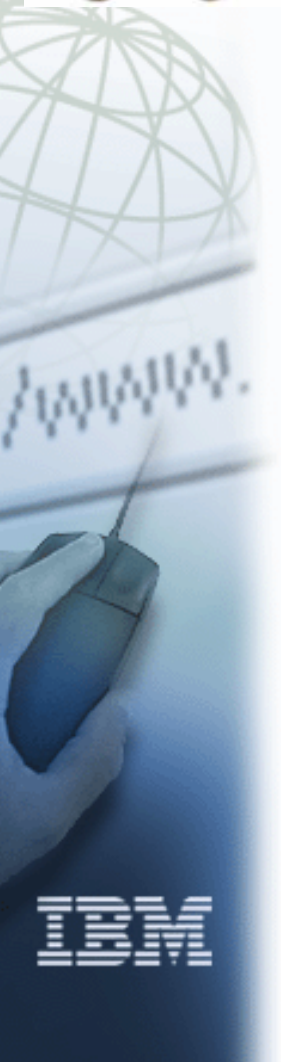
The Implementation

- Example: (Xen)

- Why no patching?

- Their current `local_irq_disable()`:

```
vcpu_info_t *_vcpu;  
preempt_disable();  
_vcpu = &HYPERVISOR_shared_info->vcpu_info  
        [smp_processor_id()];  
_vcpu->evtchn_upcall_mask = 1;  
preempt_enable_no_resched();
```





The Implementation

- Example: (Xen)
 - If we use %gs to point to per-cpu data, we could do this in one instruction:
 - `movb $1, %gs:offset`
 - 8 bytes, no registers needed!
 - Kernel does not use %gs register currently.
 - Jeremy Fitzhardinge has patches to change this





The Implementation

- We also need a way to recognize which hypervisor Linux is booting under.
 - We branch early in vmlinux entry if we are not in ring0.
 - We then call “probe” functions for each hypervisor type we support.
 - Xen uses its own entry point at the moment, but branches to same probing routine.
 - probe routine populates paravirt_ops struct.





Future Work

- Kernel inclusion in 2.6.20?
- I expect many more hypervisors to be implemented in the future.
 - Linux will be the simplest “real” operating system to port to such hypervisors.
- Some work is being done on an in-tree example hypervisor (“lhype”)
 - Should lead to better understanding of hypervisor issues by Linux Kernel community.



Referenes

- Patches can be found here:
 - <http://ozlabs.org/~rusty/paravirt>
- Main contributors (so far!):
 - Zach Amsden (VMWare)
 - Jeremy Fitzhardinge (XenSource)
 - Chris Wright (RedHat)
 - Rusty Russell (IBM)





Legal Statement

This work represents the views of the author(s) and does not necessarily reflect the views of IBM Corporation.

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries: IBM (logo). A full list of U.S. trademarks owned by IBM may be found at <http://www.ibm.com/legal/copytrade.shtml>.

Linux is a registered trademark of Linus Torvalds.

Other company, product, and service names may be trademarks or service marks of others.

