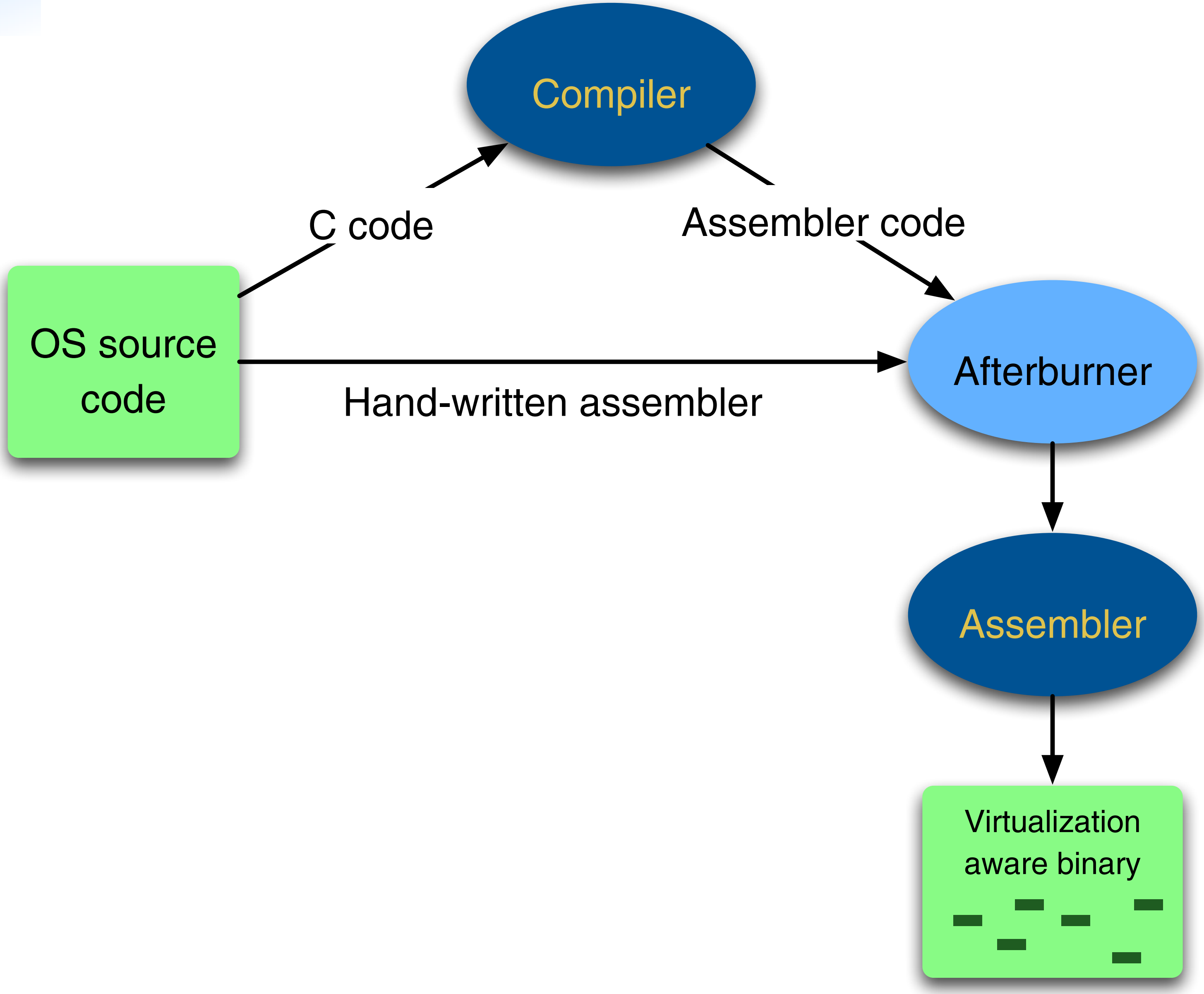
Pre-virtualization internals

Joshua LeVasseur
3 March 2006

L4Ka.org

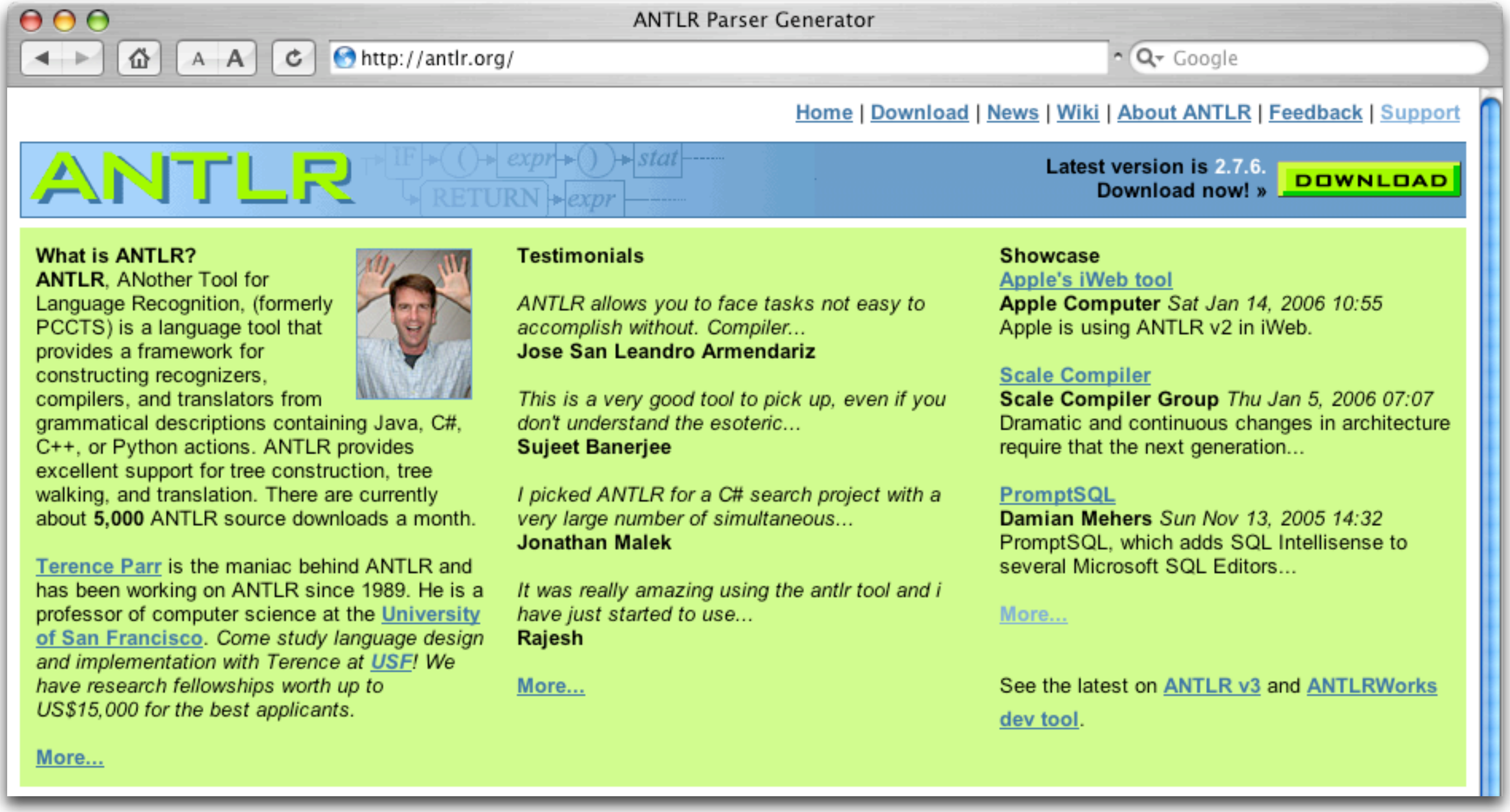
Universität Karlsruhe (TH)

Compile time overview



Afterburner assembler parser

Built using an Antlr grammar



Home | Download | News | Wiki | About ANTLR | Feedback | Support

Latest version is 2.7.6.
Download now! »

What is ANTLR?

ANTLR, ANother Tool for Language Recognition, (formerly PCCTS) is a language tool that provides a framework for constructing recognizers, compilers, and translators from grammatical descriptions containing Java, C#, C++, or Python actions. ANTLR provides excellent support for tree construction, tree walking, and translation. There are currently about **5,000** ANTLR source downloads a month.

Terence Parr is the maniac behind ANTLR and has been working on ANTLR since 1989. He is a professor of computer science at the University of San Francisco. *Come study language design and implementation with Terence at USF! We have research fellowships worth up to US$15,000 for the best applicants.*

More...

Testimonials

ANTLR allows you to face tasks not easy to accomplish without. Compiler...
Jose San Leandro Armendariz

This is a very good tool to pick up, even if you don't understand the esoteric...
Sujeet Banerjee

I picked ANTLR for a C# search project with a very large number of simultaneous...
Jonathan Malek

It was really amazing using the antlr tool and i have just started to use...
Rajesh

More...

Showcase

Apple's iWeb tool
Apple Computer *Sat Jan 14, 2006 10:55*
Apple is using ANTLR v2 in iWeb.

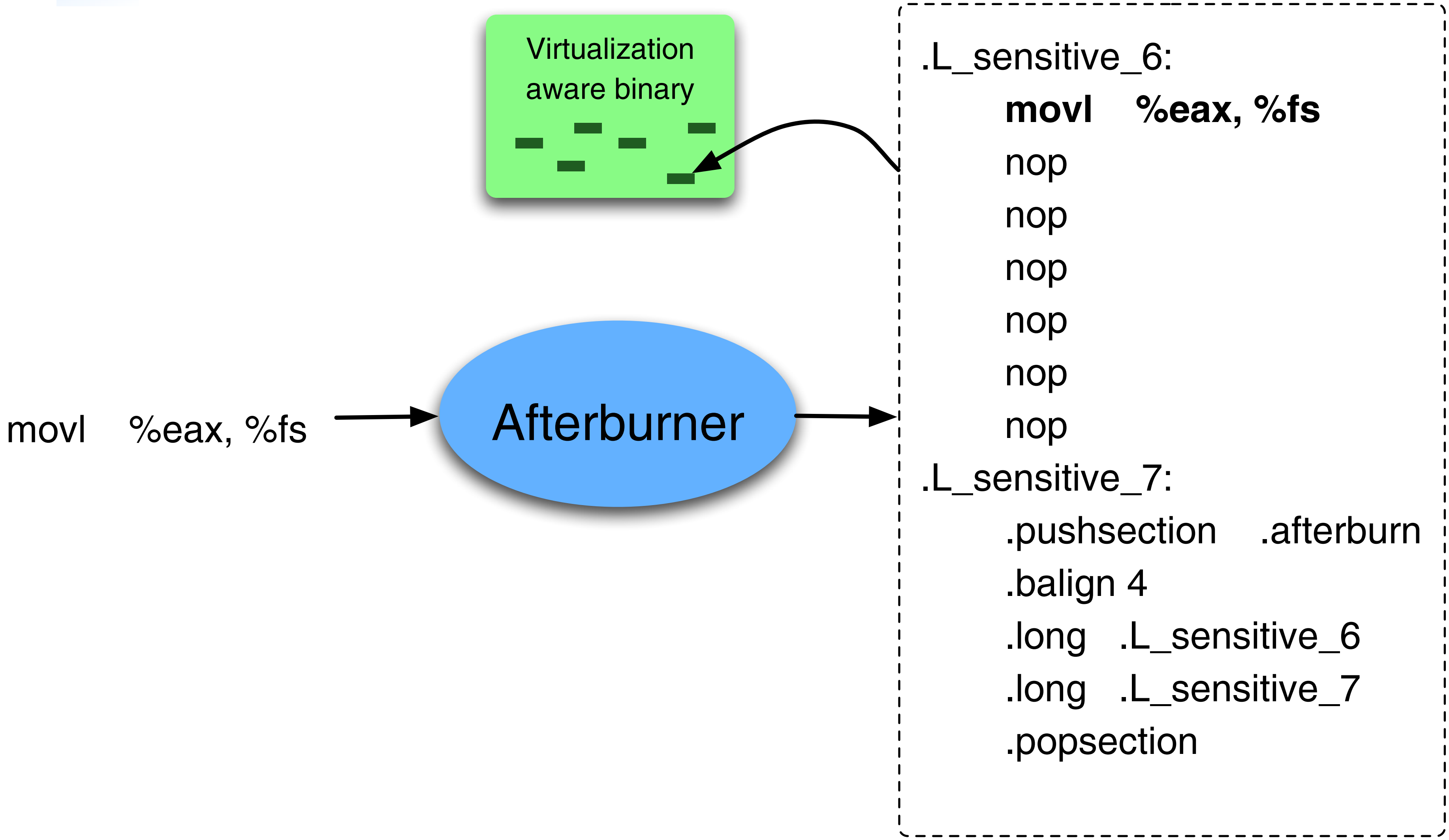
Scale Compiler
Scale Compiler Group *Thu Jan 5, 2006 07:07*
Dramatic and continuous changes in architecture require that the next generation...

PromptSQL
Damian Mehers *Sun Nov 13, 2005 14:32*
PromptSQL, which adds SQL Intellisense to several Microsoft SQL Editors...

More...

See the latest on ANTLR v3 and ANTLRWorks dev tool.

Instruction padding



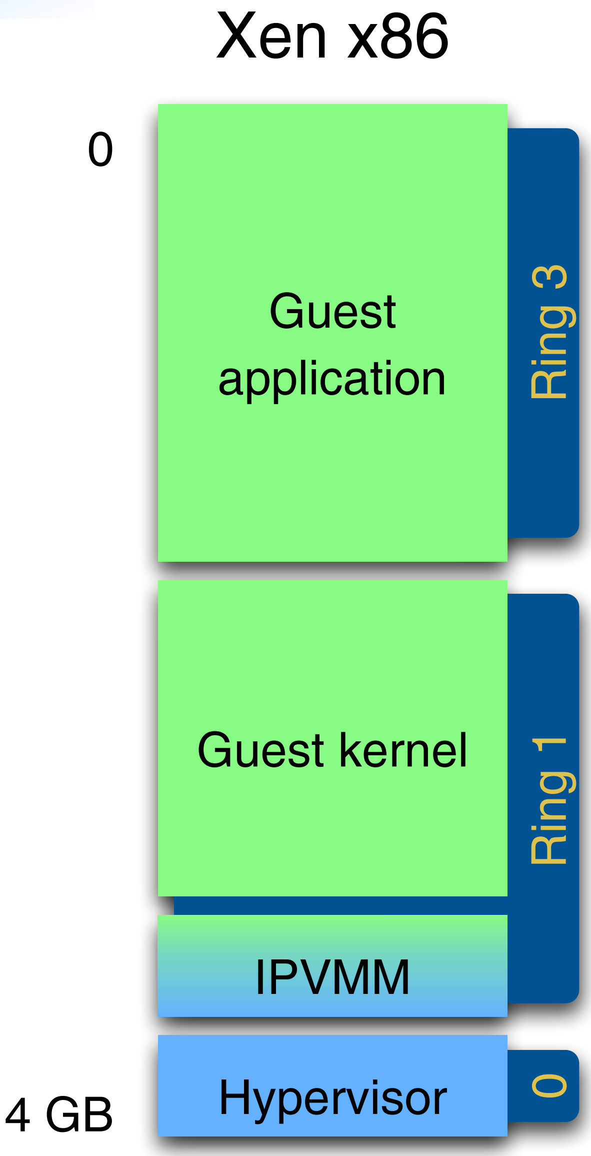
Virtualization aware binary

movl %eax, %fs

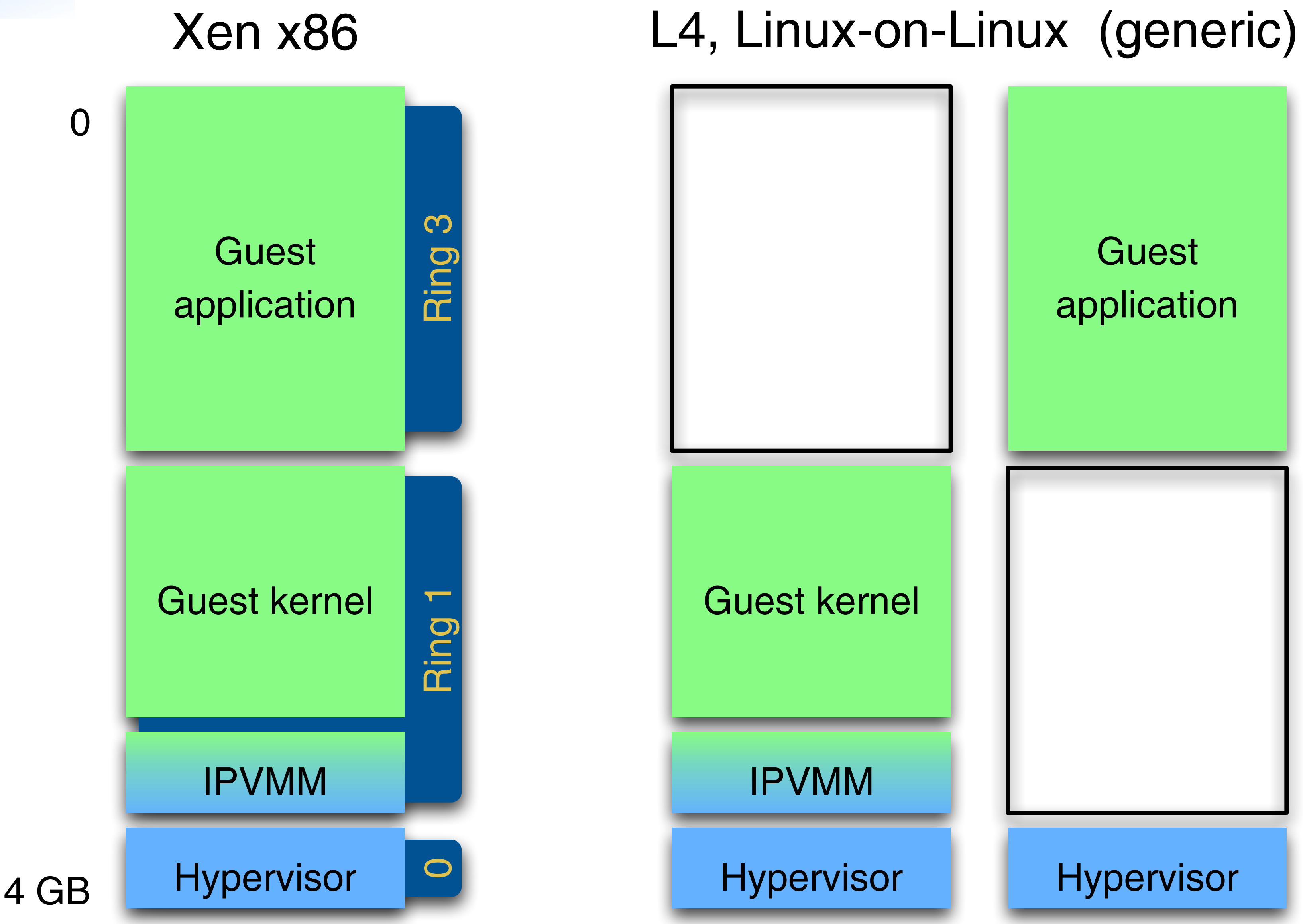
Afterburner

```
.L_sensitive_6:
        movl    %eax, %fs
        nop
        nop
        nop
        nop
        nop
        nop
.L_sensitive_7:
        .pushsection    .afterburn
        .balign 4
        .long   .L_sensitive_6
        .long   .L_sensitive_7
        .popsection
```

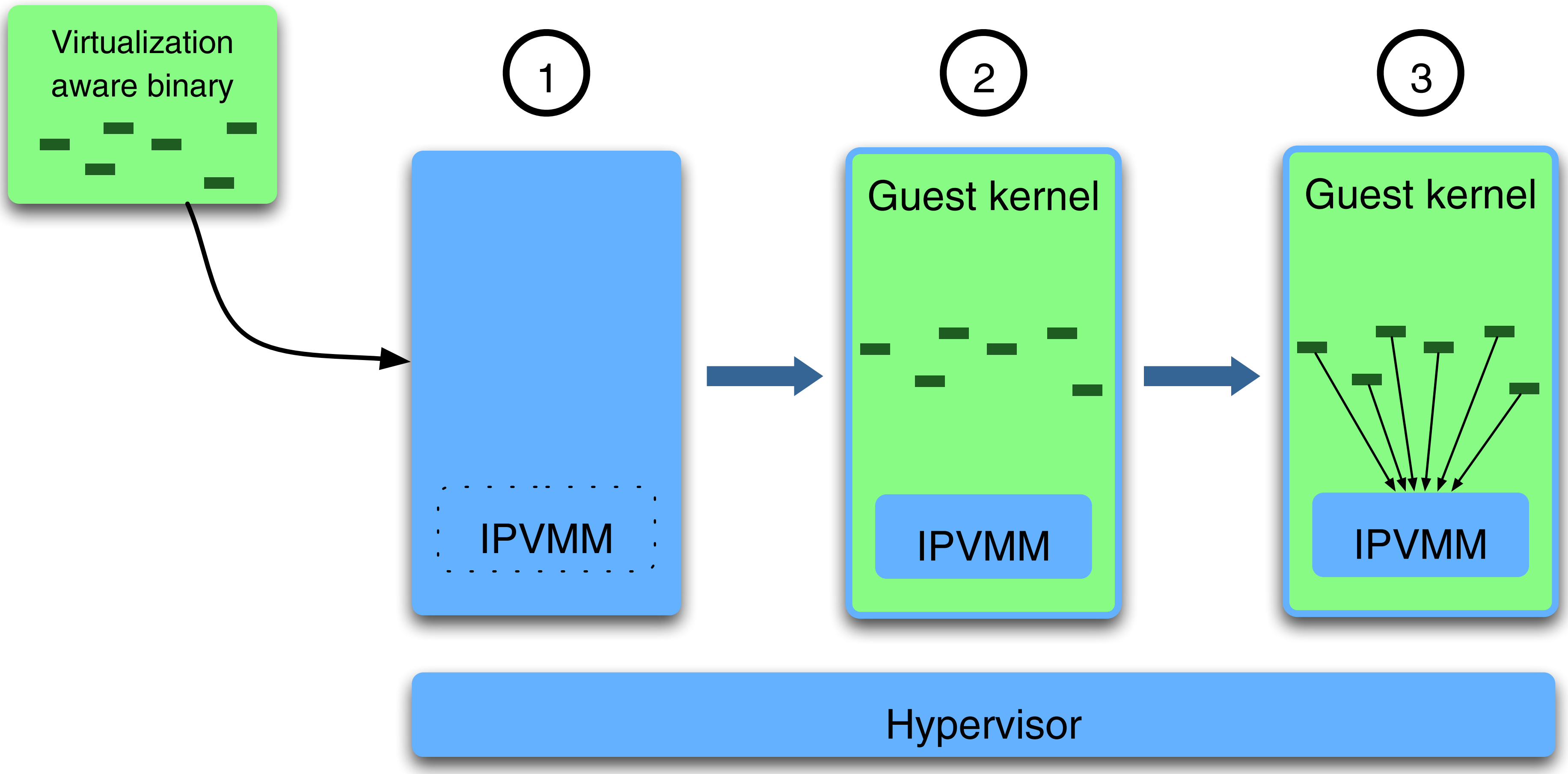
Address space



Address space

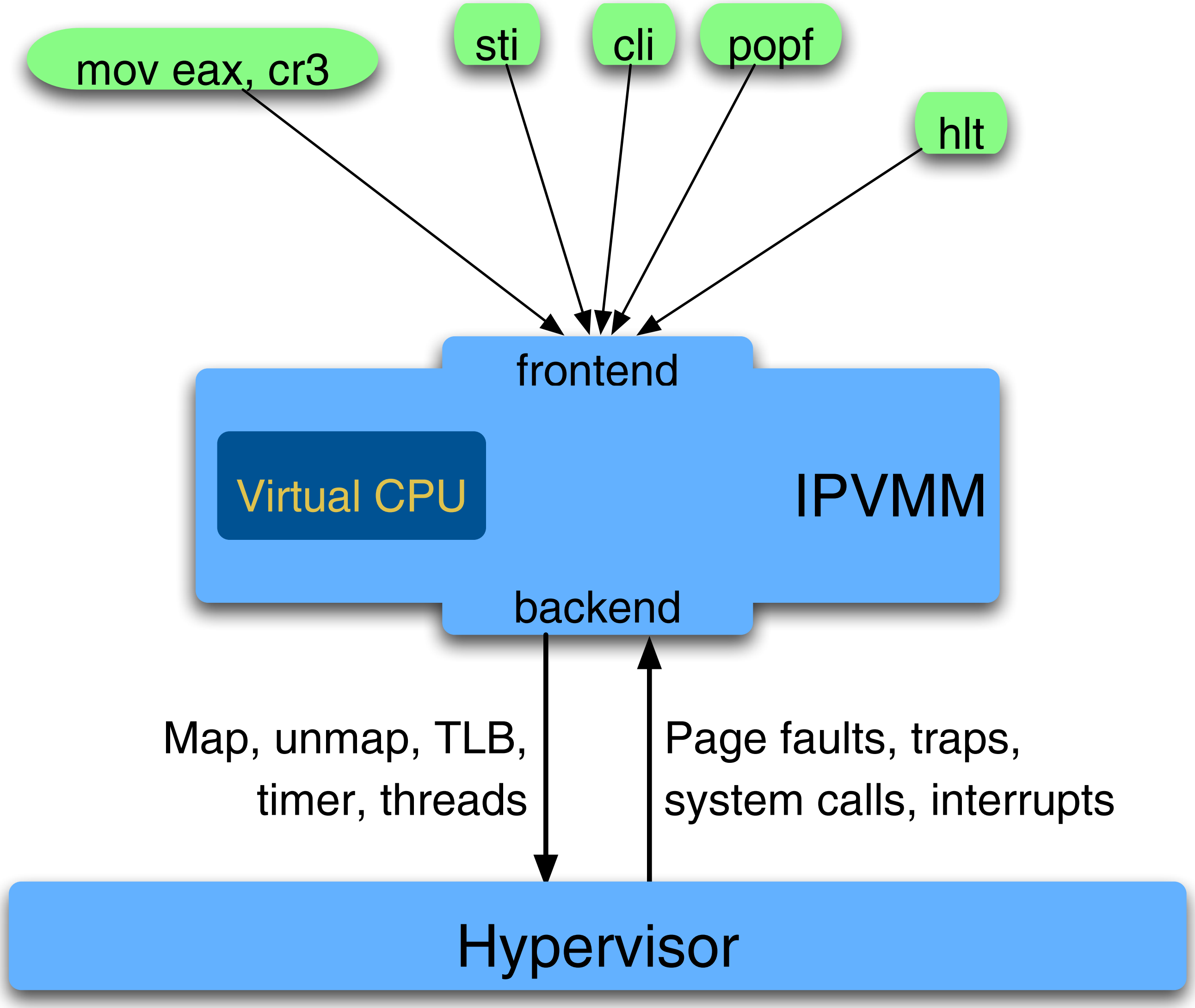


Loading the guest kernel



© 2006 Universität Karlsruhe — *Pre-virtualization internals* — Joshua LeVasseur

In-Place VMM


mov eax, cr3
sti
cli
popf
hlt
frontend
Virtual CPU
IPVMM
backend
Map, unmap, TLB, timer, threads
Page faults, traps, system calls, interrupts
Hypervisor

© 2006 Universität Karlsruhe

Constraints

Code expansion:

- Timing?
- Interrupts?
- Simple state machine?
- Efficiency?

Constraints

- Guest kernel is a sequential process
  - Important: forward progress
  - Unimportant: rate of forward progress
- We provide a virtual CPU
  - Illusion of continuous time

Code expansion:

- Timing?
- Interrupts?
- Simple state machine?
- Efficiency?

© 2006 Universität Karlsruhe — *Pre-virtualization internals* — Joshua LeVasseur

Constraints

Code expansion:

- Timing?
- Interrupts?
- Simple state machine?
- Efficiency?

- Guest kernel is a sequential process
  - Important: forward progress
  - Unimportant: rate of forward progress
- We provide a virtual CPU
  - Illusion of continuous time

- Delay delivery
  - Synchronous (sti, popf)
    - Optimize common case
  - Asynchronous
    - Avoid IPVMM reentrance

Constraints

Code expansion:

- Timing?
- Interrupts?
- Simple state machine?
- Efficiency?

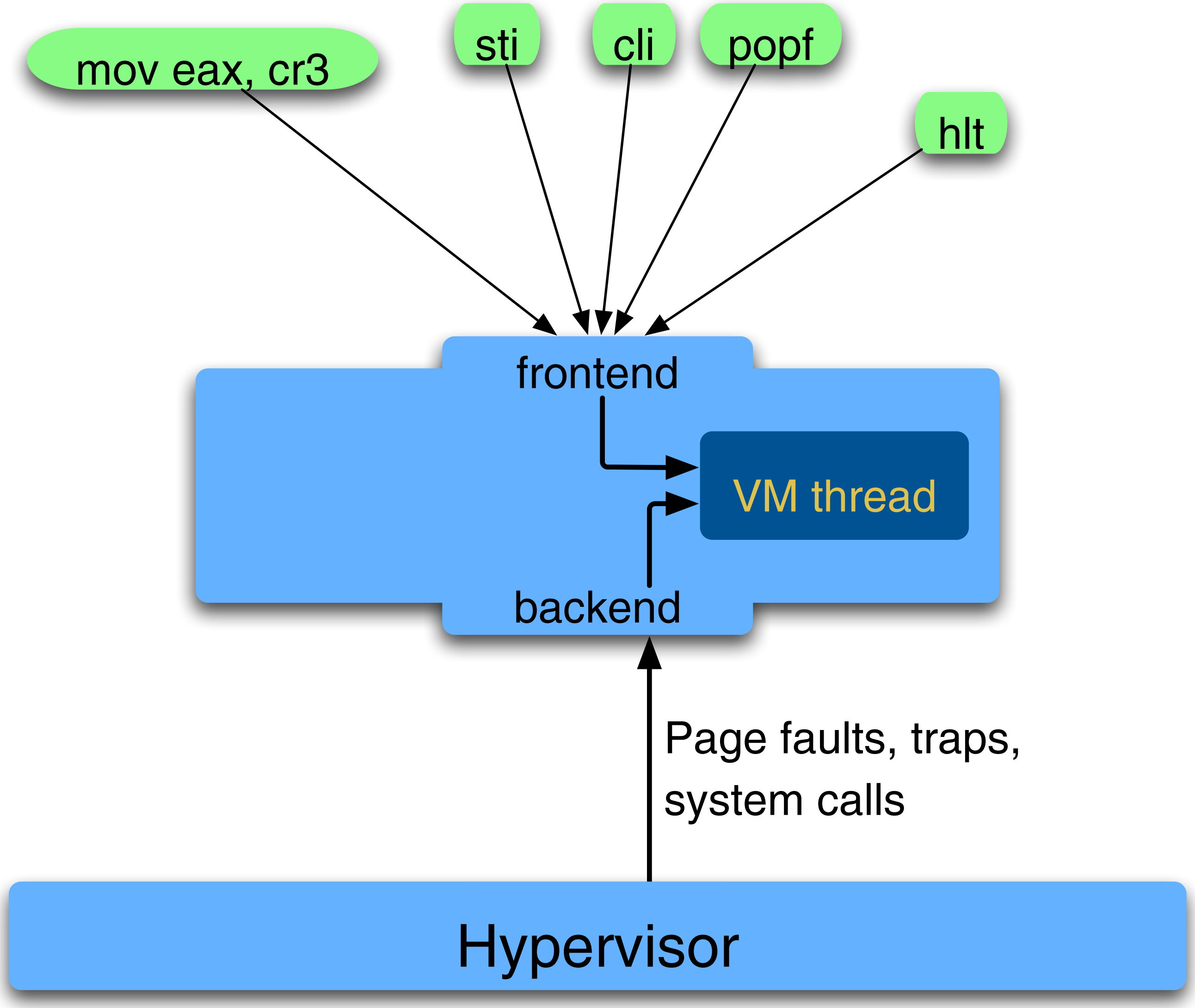
Thread model

- VM thread
- Interrupt thread

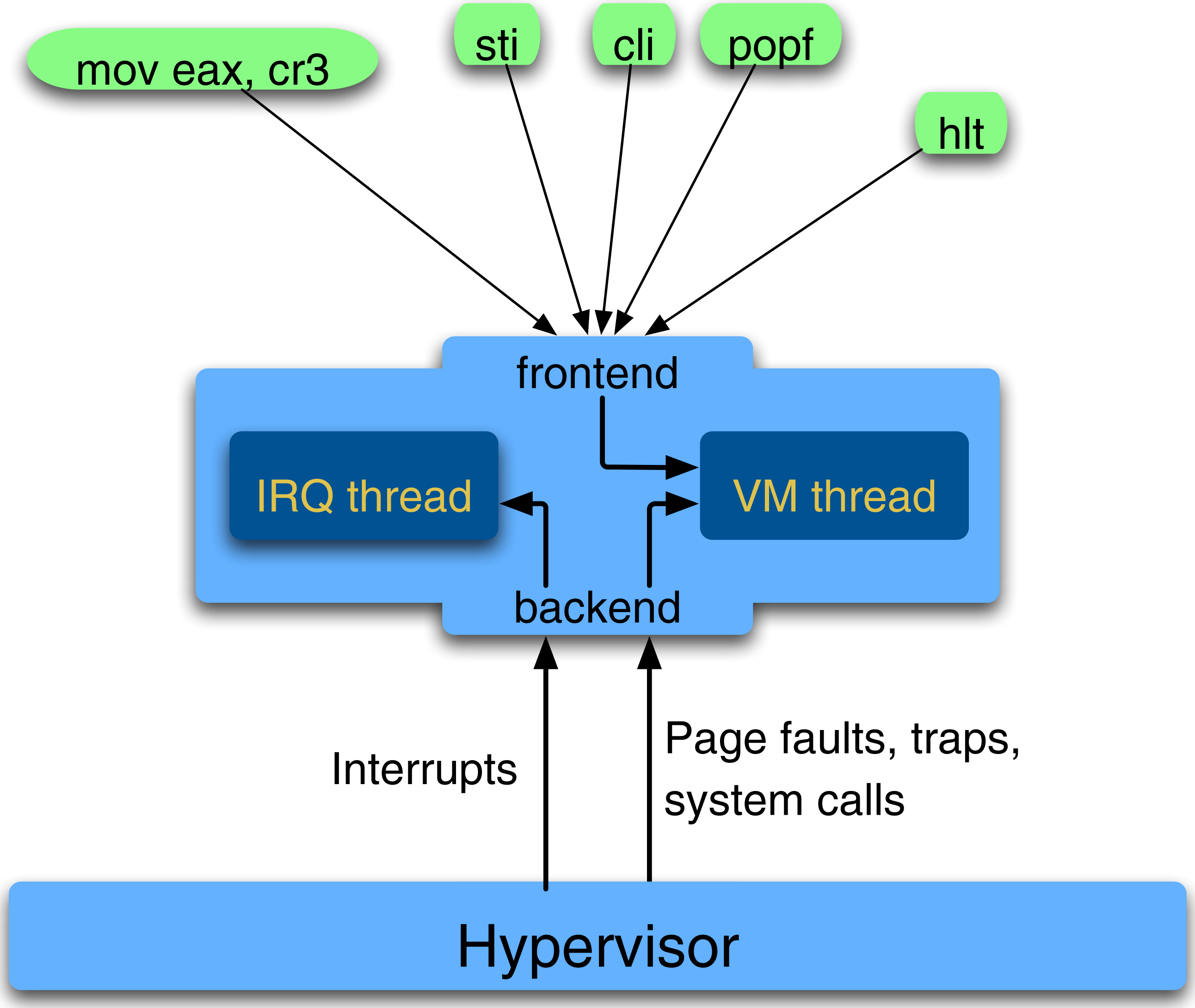
- Guest kernel is a sequential process
  - Important: forward progress
  - Unimportant: rate of forward progress
- We provide a virtual CPU
  - Illusion of continuous time

- Delay delivery
  - Synchronous (sti, popf)
    - Optimize common case
  - Asynchronous
    - Avoid IPVMM reentrance

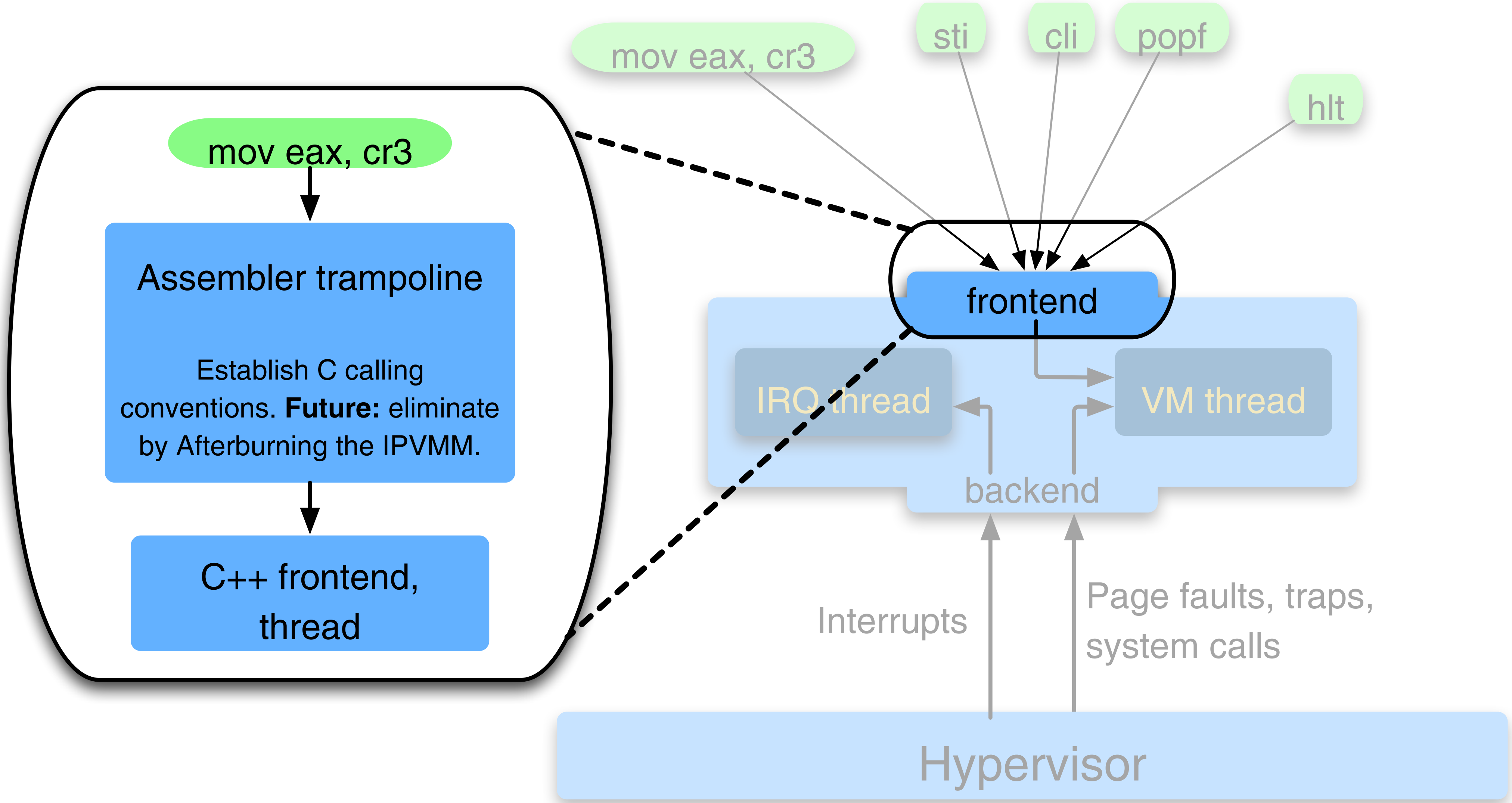
Threads


mov eax, cr3
sti
cli
popf
hlt
frontend
VM thread
backend
Page faults, traps, system calls
Hypervisor

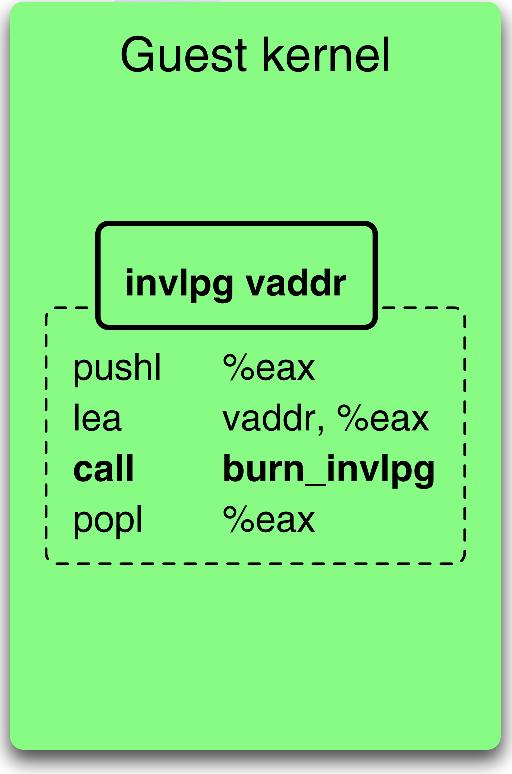
Threads


mov eax, cr3
sti
cli
popf
hlt
frontend
IRQ thread
VM thread
backend
Interrupts
Page faults, traps, system calls
Hypervisor

Frontend entry



Boundary transitions

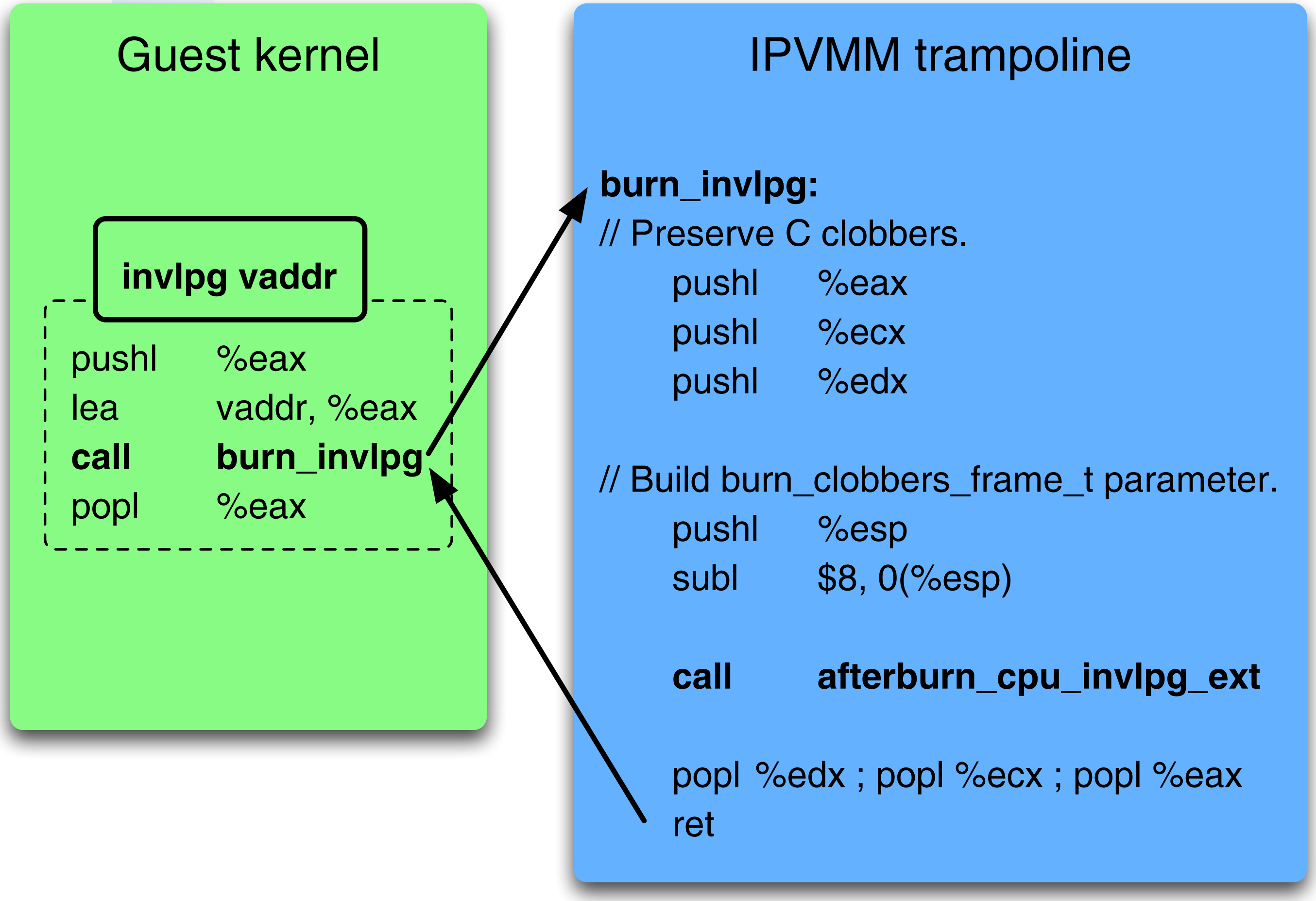


Guest kernel

invlpg vaddr

```
pushl   %eax
lea     vaddr, %eax
call    burn_invlpg
popl    %eax
```

Boundary transitions (generic)



Guest kernel

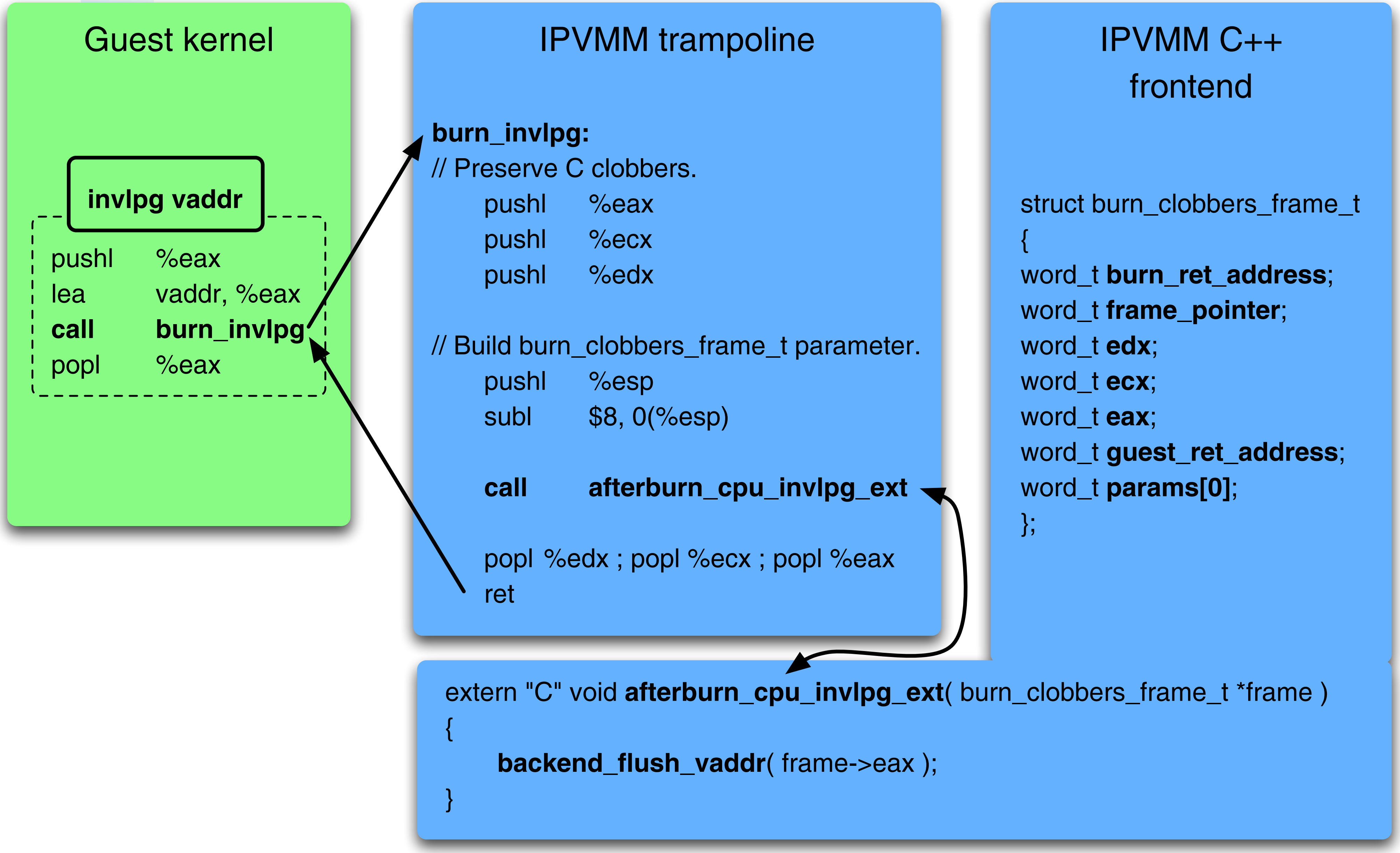
invlpg vaddr

```
pushl   %eax
lea     vaddr, %eax
call    burn_invlpg
popl    %eax
```

IPVMM trampoline

```
burn_invlpg:
// Preserve C clobbers.
        pushl   %eax
        pushl   %ecx
        pushl   %edx

// Build burn_clobbers_frame_t parameter.
        pushl   %esp
        subl    $8, 0(%esp)

        call    afterburn_cpu_invlpg_ext

        popl  %edx ; popl %ecx ; popl %eax
        ret
```

Boundary transitions (generic)



Guest kernel

```
invlpg vaddr

pushl   %eax
lea     vaddr, %eax
call    burn_invlpg
popl    %eax
```

IPVMM trampoline

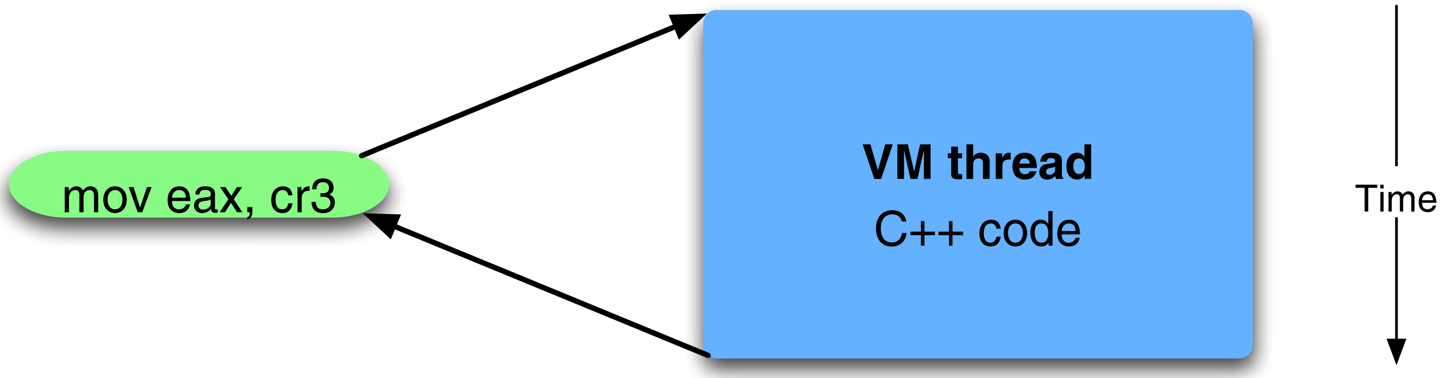
```
burn_invlpg:
// Preserve C clobbers.
        pushl   %eax
        pushl   %ecx
        pushl   %edx

// Build burn_clobbers_frame_t parameter.
        pushl   %esp
        subl    $8, 0(%esp)

        call    afterburn_cpu_invlpg_ext

        popl %edx ; popl %ecx ; popl %eax
        ret
```

IPVMM C++ frontend

```
struct burn_clobbers_frame_t
{
word_t burn_ret_address;
word_t frame_pointer;
word_t edx;
word_t ecx;
word_t eax;
word_t guest_ret_address;
word_t params[0];
};
```

```
extern "C" void afterburn_cpu_invlpg_ext( burn_clobbers_frame_t *frame )
{
        backend_flush_vaddr( frame->eax );
}
```

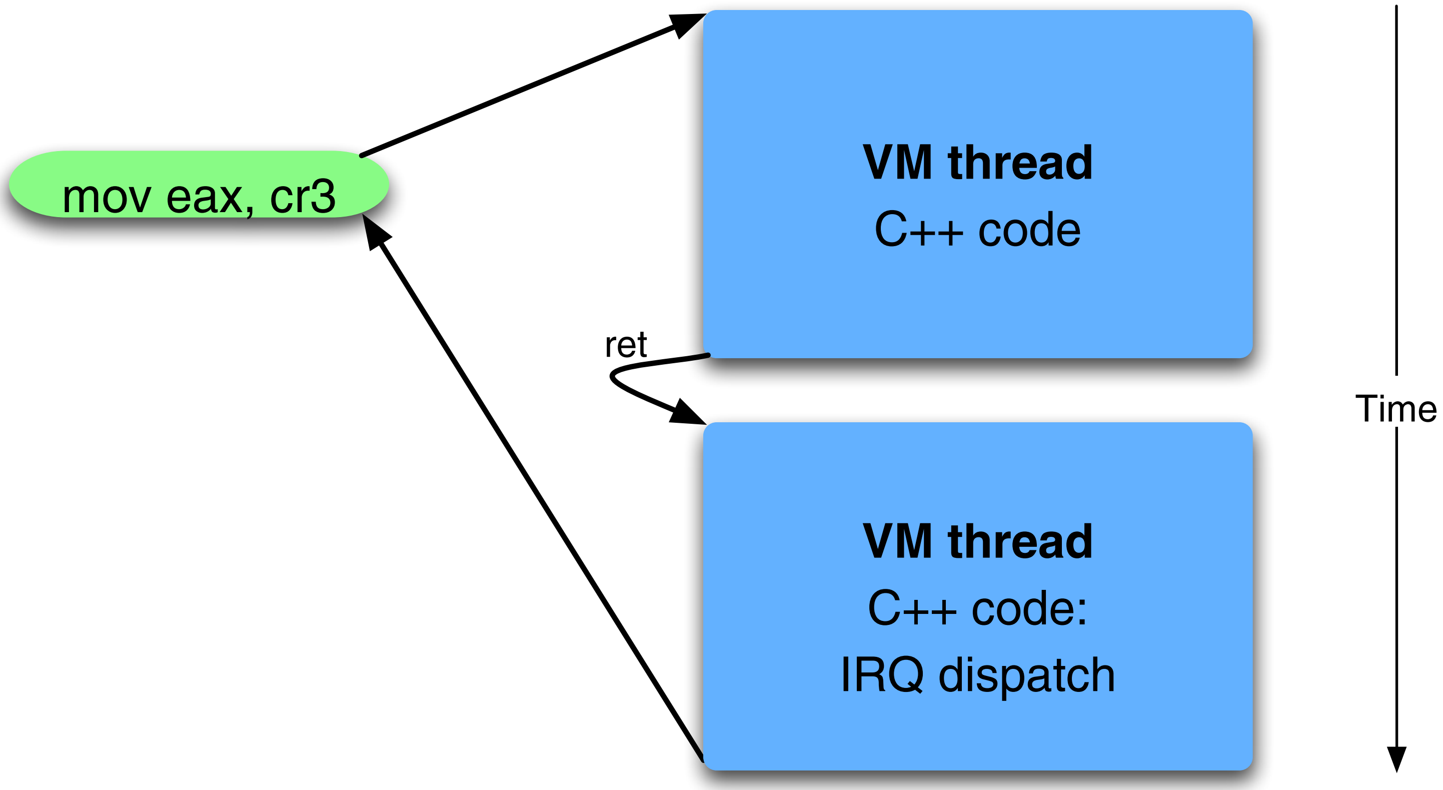
Atomic instructions



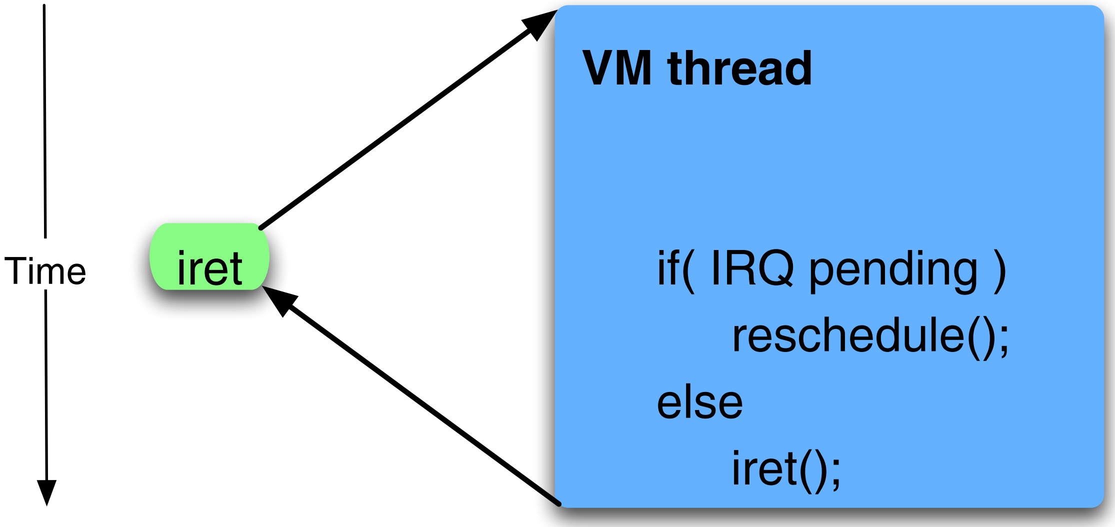
Interrupts

- During IPVMM code?
- During a hypercall?

Thread reschedule



Race conditions



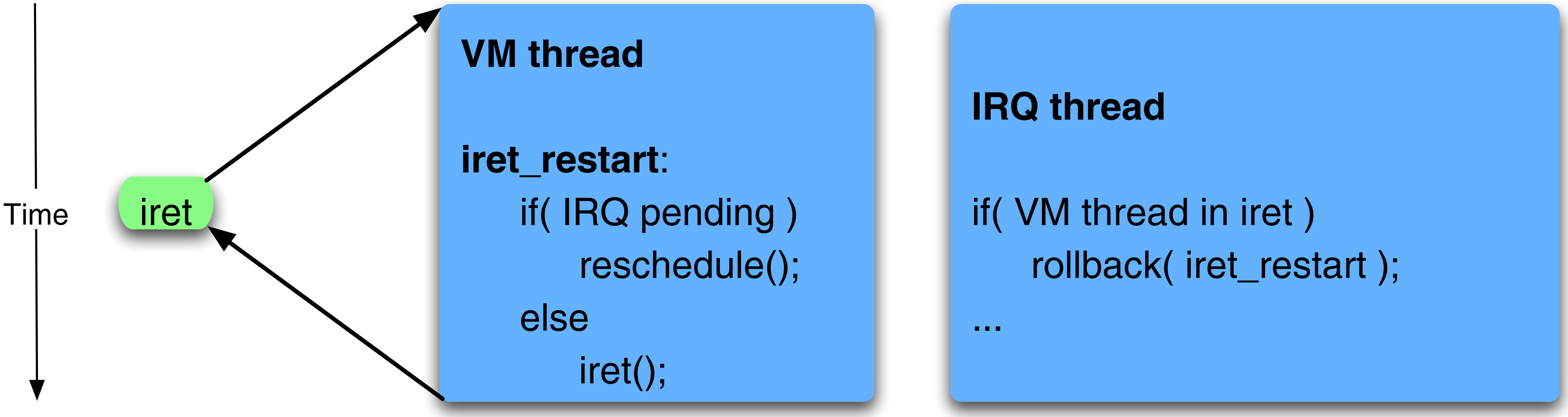
Time

iret

VM thread

```
if( IRQ pending )
    reschedule();
else
    iret();
```

Race conditions



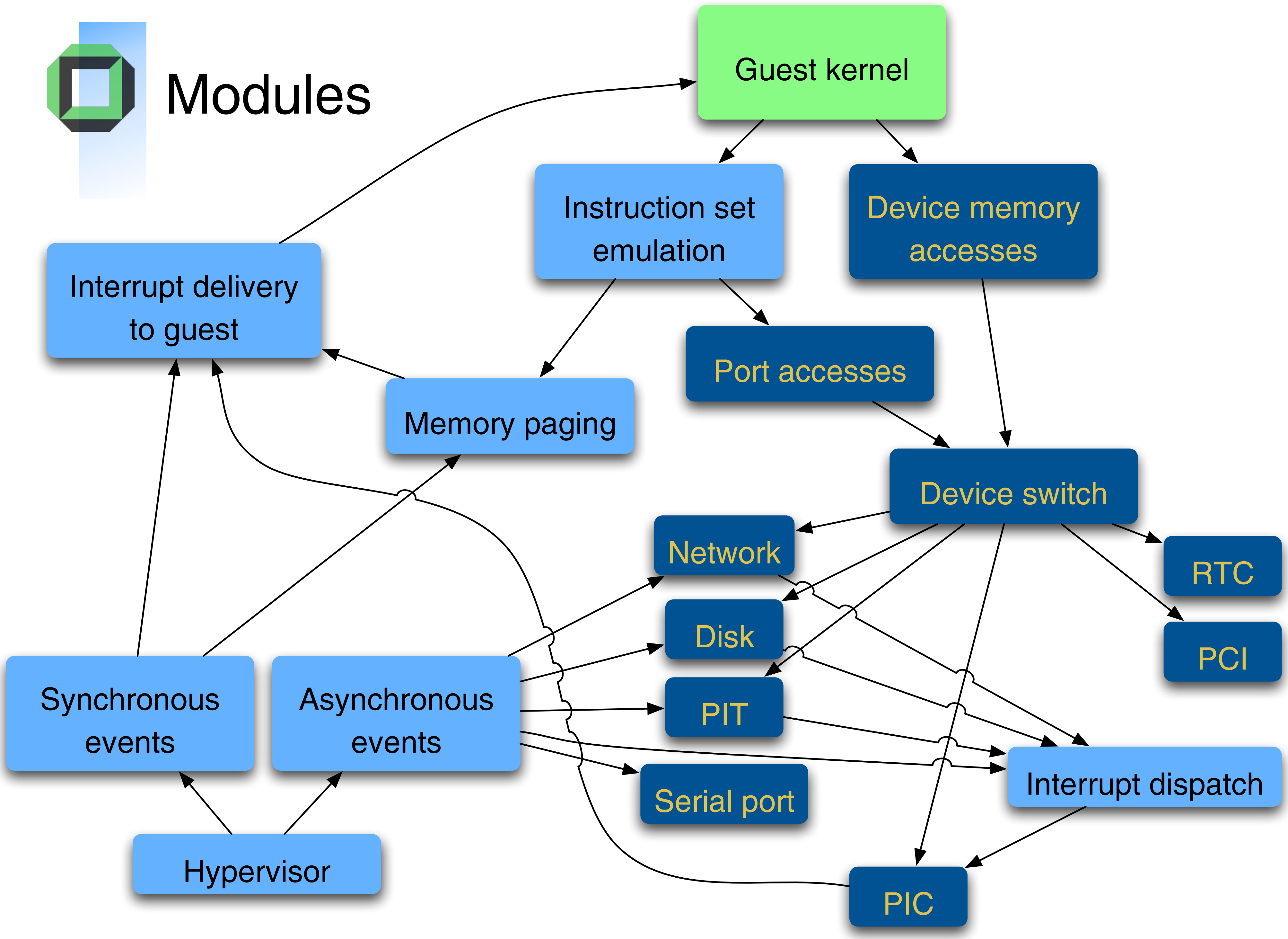
VM thread

```
iret_restart:
    if( IRQ pending )
        reschedule();
    else
        iret();
```

IRQ thread

```
if( VM thread in iret )
    rollback( iret_restart );
...
```

Modules


Guest kernel
Instruction set emulation
Device memory accesses
Interrupt delivery to guest
Port accesses
Memory paging
Device switch
Network
RTC
Disk
PCI
PIT
Synchronous events
Asynchronous events
Interrupt dispatch
Serial port
Hypervisor
PIC

Adding a device model

1. Define a device class
2. Define its interfaces:
   - Port accesses
   - Memory-mapped registers
3. Define its PCI registers:
   - Static structure created at compile-time
4. Raise interrupts with the intlogic_t class
5. In some cases, define a virtual IRQ handler

Productivity-focused coding

Minimize errors:

- **Simple code**
  - Avoid purely quantitative increase in work
- Use assertions
- Avoid reentrancy
- C++
  - Modularity (quickly understood code)
  - Compile-time features only
  - Fast (world's fastest kernel, L4Ka::Pistachio, is written in C++)
- No dynamic memory allocation
  - Construct data structures at compile time
    - Ex: big switch() statement for port access, PCI device access
- **Heavily typed**

Productivity-focused coding

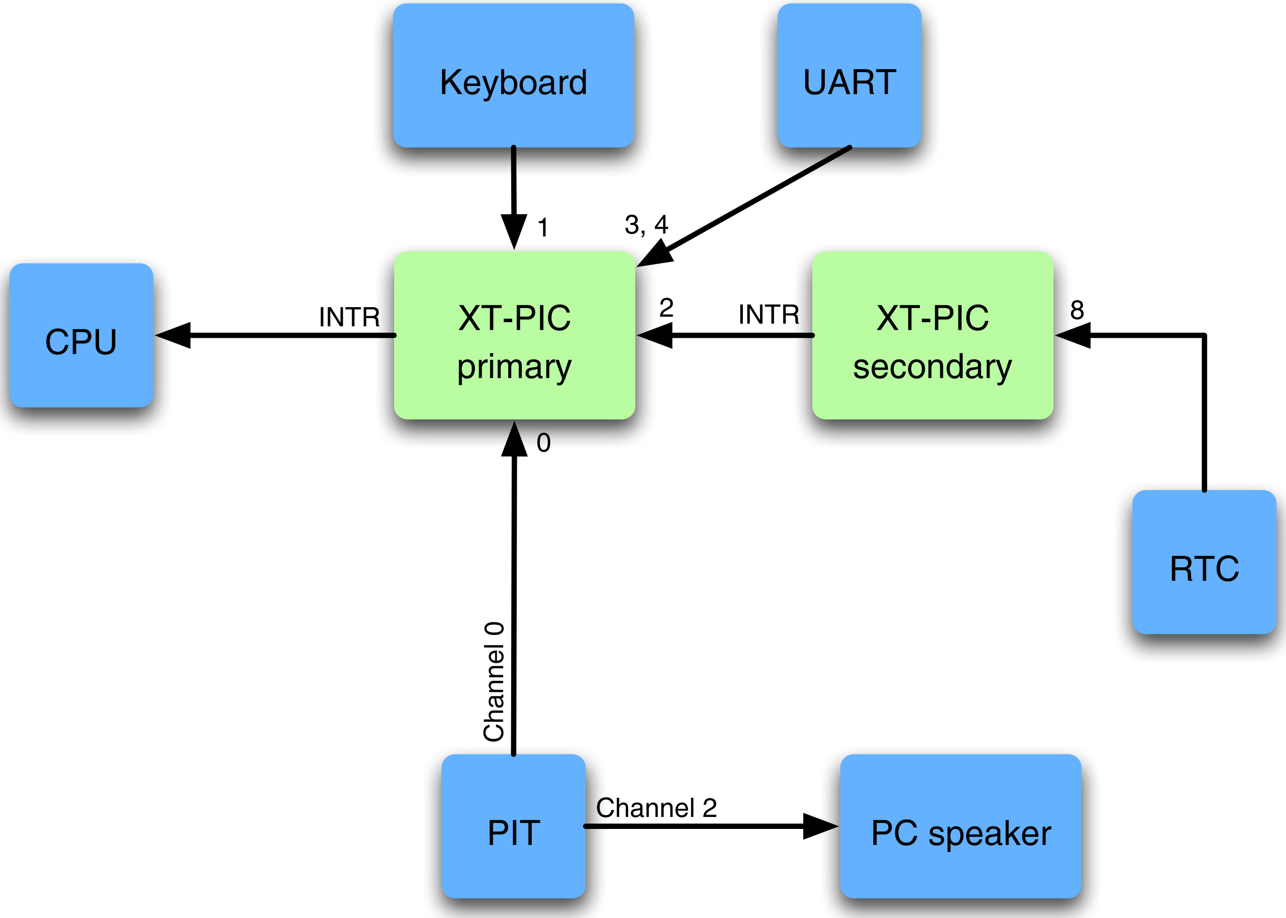
Code maintenance:

- Minimize C Preprocessor
  - We want structured code
- Use CML2 configuration system
  - Feature management

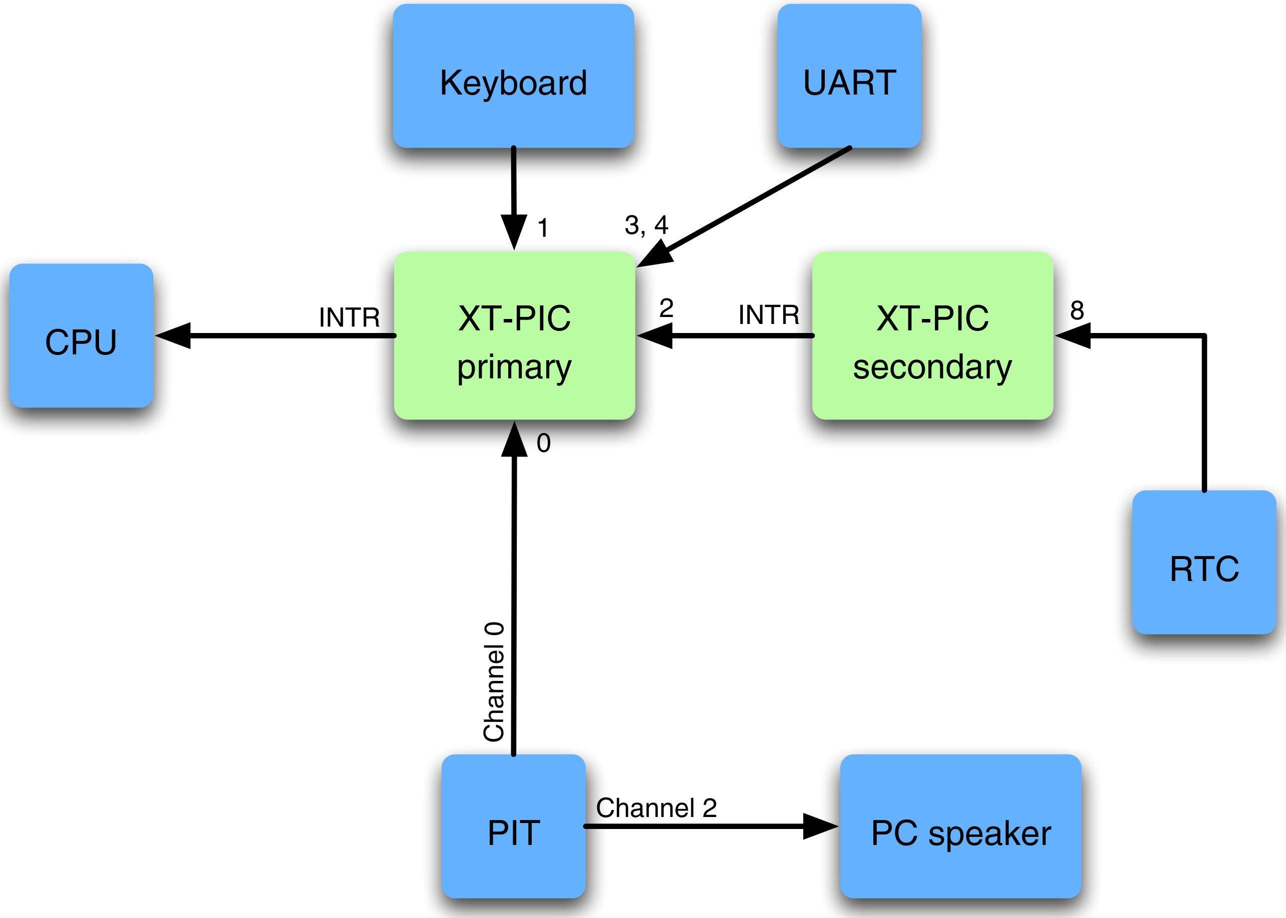
Code reuse:

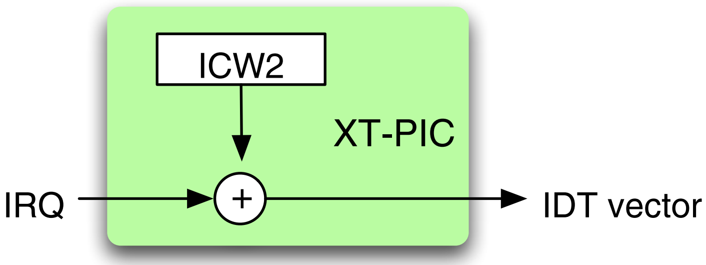
- Abstract frontends and backends
- Reuse frontend across multiple backends

Legacy devices

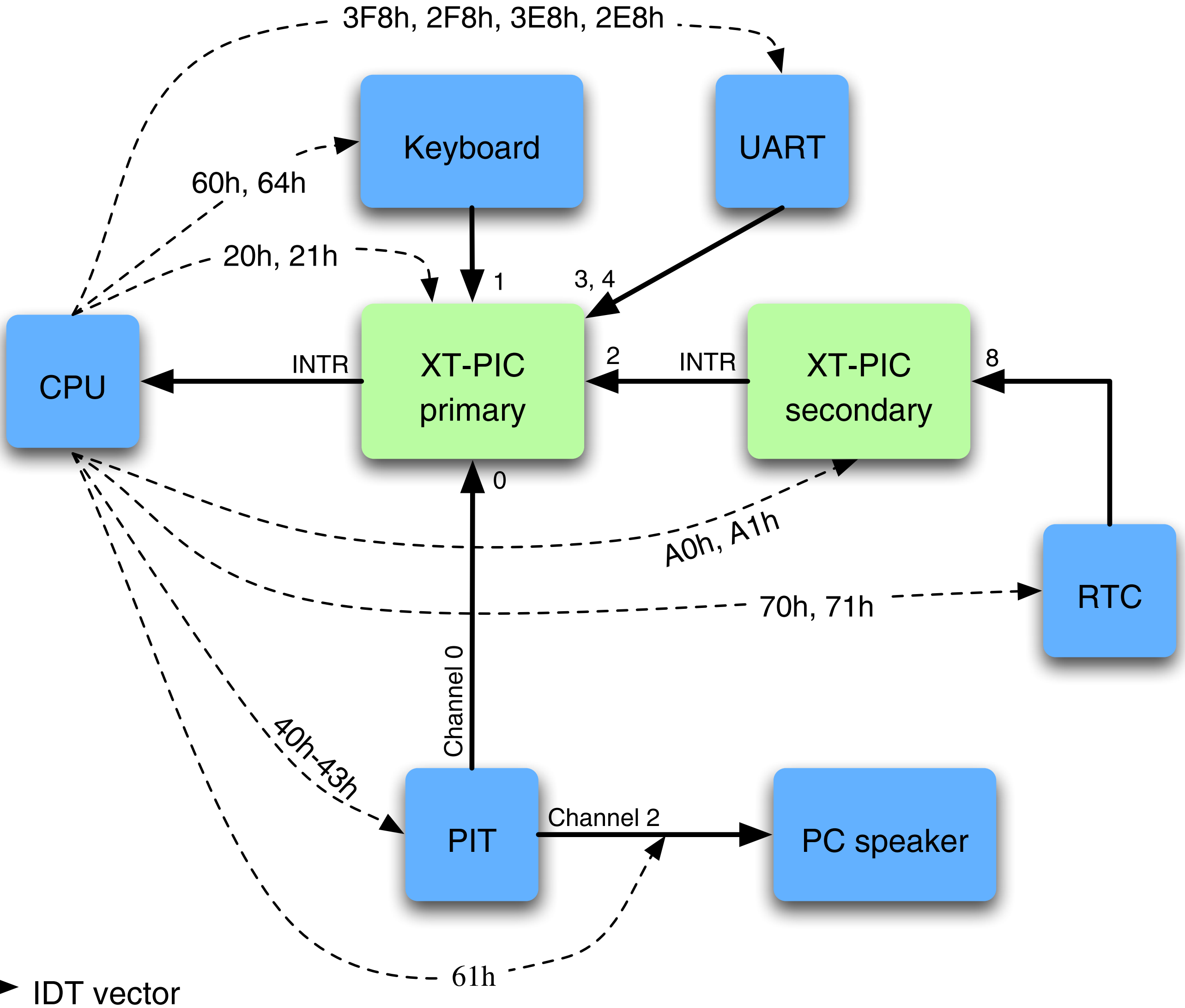


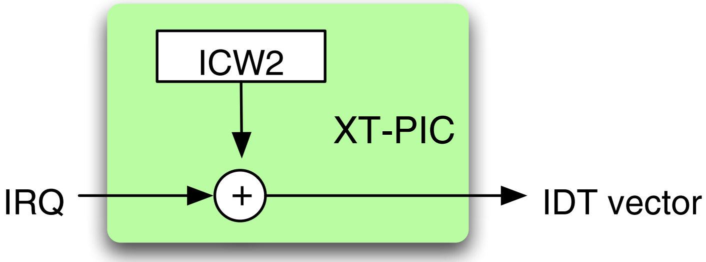
Legacy devices: IDT vector





Legacy devices: Ports





Further information

http://l4ka.org/projects/virtualization/

Publications:

- Full paper: *Pre-Virtualization: Slashing the Cost of Virtualization*
- Quick read: *Pre-Virtualization: Uniting Two Worlds*
- Quick read: White paper