COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

Virtual Open Systems

ARM VIRTUALIZATION "FOR THE MASSES"

Christoffer Dall <c.dall@virtualopensystems.com>
<cdall@cs.columbia.edu>

ARM

ARM®

TEXAS INSTRUMENTS

SAMSUNG

QUALCOMM®

Smartphones

Smartphones

Tablets


0.9 mi
E 47th St
E 48th St
E 49th St
E 50th St
E Hyde Park Blvd
E Madison Park
E 52nd St
E 53rd St
E 54th St
E 55th St
E 56th St
S Woodlawn Ave
19 min

Tablets


SAVE
Vignette
Contrast
Shadows
Vibrance
Sharpness
Curves

But now also...

ARM Servers

But now also...

ARM Servers

Laptops

Even in cars

... and toasters!

So why virtualize?

So why virtualize?

Server virtualization brings high availability, load balancing, and more..

So why virtualize?

- Phone virtualization
  - iOS on Android
  - Home-phone Work-phone
  - Throw-away virtual phones

So why virtualize?

Rich OS in embedded space with legacy OS support

So why virtualize?

```
<1>Unable to handle kernel NULL pointer dereference at virtual address 00000019
[    4.640000] Unable to handle kernel NULL pointer dereference at virtual address 0
<1>pgd = c0004000
[    4.660000] pgd = c0004000
<1>[00000019] *pgd=00000000[    4.660000] [00000019] *pgd=00000000

<0>Internal error: Oops: 5 [#1] PREEMPT
[    4.670000] Internal error: Oops: 5 [#1] PREEMPT
<d>Modules linked in:[    4.670000] Modules linked in:

CPU: 0    Not tainted  (3.0.38-1-ARCH #1)
[    4.670000] CPU: 0    Not tainted  (3.0.38-1-ARCH #1)
PC is at regulator_get_voltage+0x8/0x34
[    4.670000] PC is at regulator_get_voltage+0x8/0x34
LR is at sun4i_cpufreq_initcall+0xac/0x124
[    4.670000] LR is at sun4i_cpufreq_initcall+0xac/0x124
pc : [<c0217b18>]    lr : [<c000e96c>]    psr: a0000013
sp : e783bf98  ip : 00000000  fp : 00000000
[    4.670000] pc : [<c0217b18>]    lr : [<c000e96c>]    psr: a0000013
[    4.670000] sp : e783bf98  ip : 00000000  fp : 00000000
r10: c000e8c0  r9 : 00000000  r8 : 00000000
[    4.670000] r10: c000e8c0  r9 : 00000000  r8 : 00000000
r7 : 00000013  r6 : c0034bd8  r5 : c00279b4  r4 : fffffed
[    4.670000] r7 : 00000013  r6 : c0034bd8  r5 : c00279b4  r4 : fffffed
r3 : 00000000  r2 : 00000000  r1 : 00000001  r0 : fffffed
[    4.670000] r3 : 00000000  r2 : 00000000  r1 : 00000001  r0 : fffffed
Flags: NzCv  IRQs on  FIQs on  Mode SVC_32  ISA ARM  Segment kernel
[    4.670000] Flags: NzCv  IRQs on  FIQs on  Mode SVC_32  ISA ARM  Segment kernel
Control: 10c5387d  Table: 40004019  DAC: 00000015
[    4.670000] Control: 10c5387d  Table: 40004019  DAC: 00000015
```

Or debug events leading up to this kernel crash using GDB?

People

- Marc Zyngier from ARM: VGIC + Timers
- Antonios Motakis from Virtual Open Systems: VFP
- Rusty Russell from Linaro: User space register interface
- Peter Maydell from Linaro: QEMU and arch. compliance
- Christoffer Dall from Virtual Open Systems: Maintainer

KVM/ARM

KVM/ARM

- ARM is not virtualizable

KVM/ARM

- ARM is not virtualizable
- Requires Hardware Virtualization Extensions

KVM/ARM

- ARM is not virtualizable
- Requires Hardware Virtualization Extensions
- Runs on Cortex-A15 and Cortex-A7

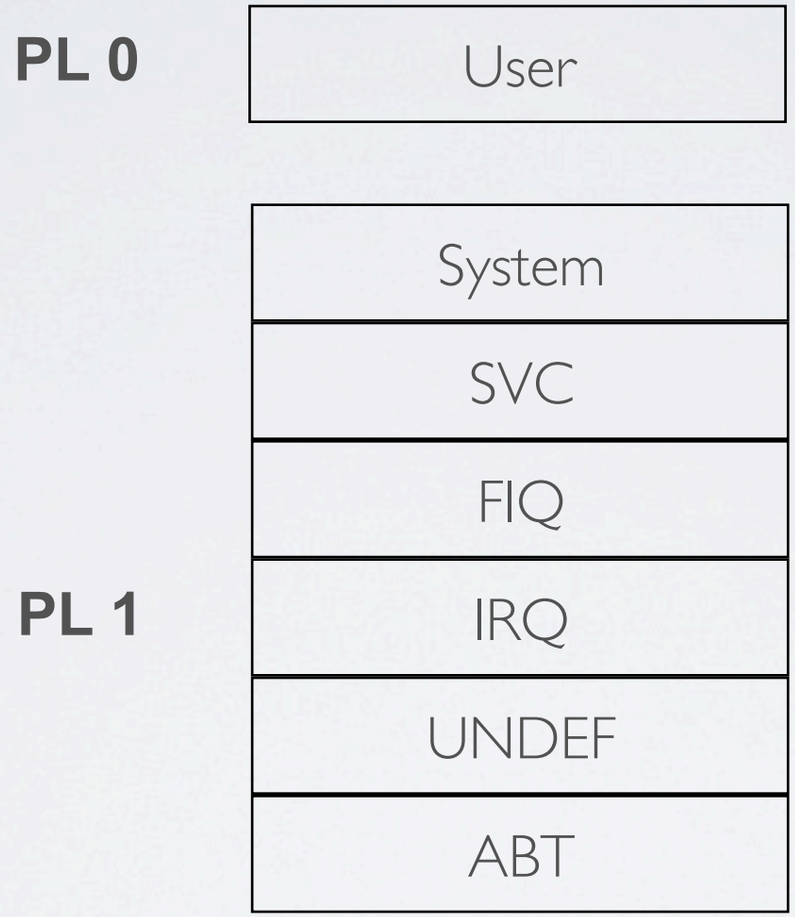
KVM/ARM upstreaming

- Being upstreamed in mainline
- 14th patch series revision
- Waiting for ARM ack's
- Upstreamed through kvm/next or Russell King (ARM)

ARM Architecture

- RISC-style load/store architecture
- Two instruction modes: ARM and Thumb-2
- Co-processors: An extension to the instruction set
- CP15 holds CPU control state

ARMv7 Processor Modes



ARM Architecture

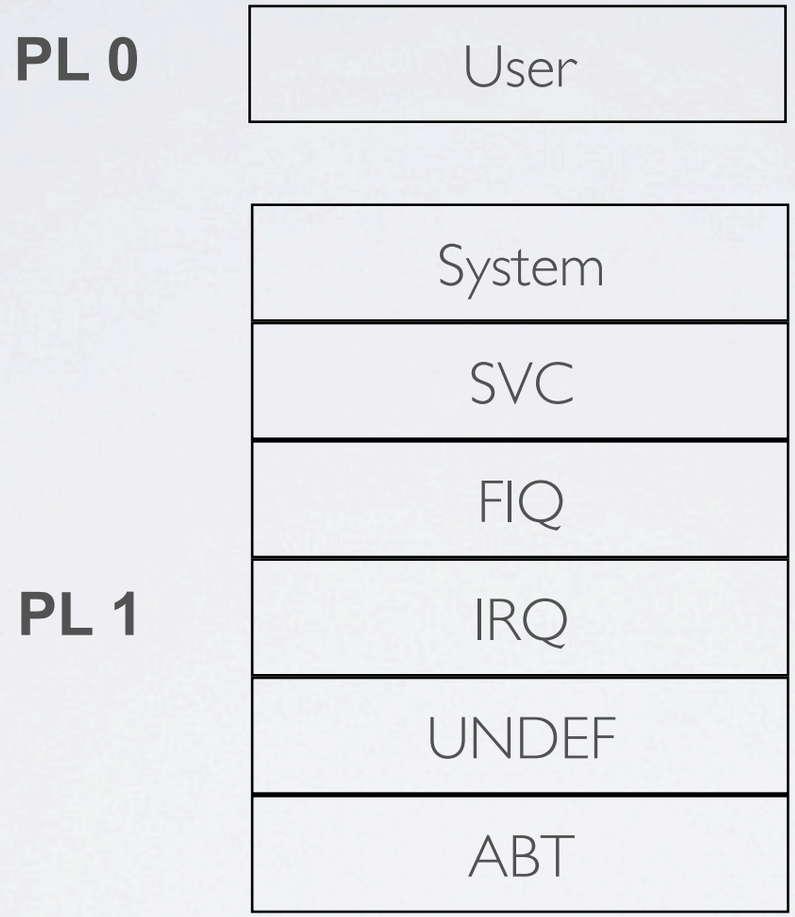
Registers

User	System	SVC	ABT	UND	IRQ	FIQ
R0						
R1						
R2						
R3						
R4						
R5						
R6						
R7						
R8						
R9						
R10						
R11						
R12						
SP_usr		SP_svc	SP_abt	SP_und	SP_irq	SP_fiq
LR_usr		LR_svc	LR_abt	LR_und	LR_irq	LR_fiq
PC						
CPSR						
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

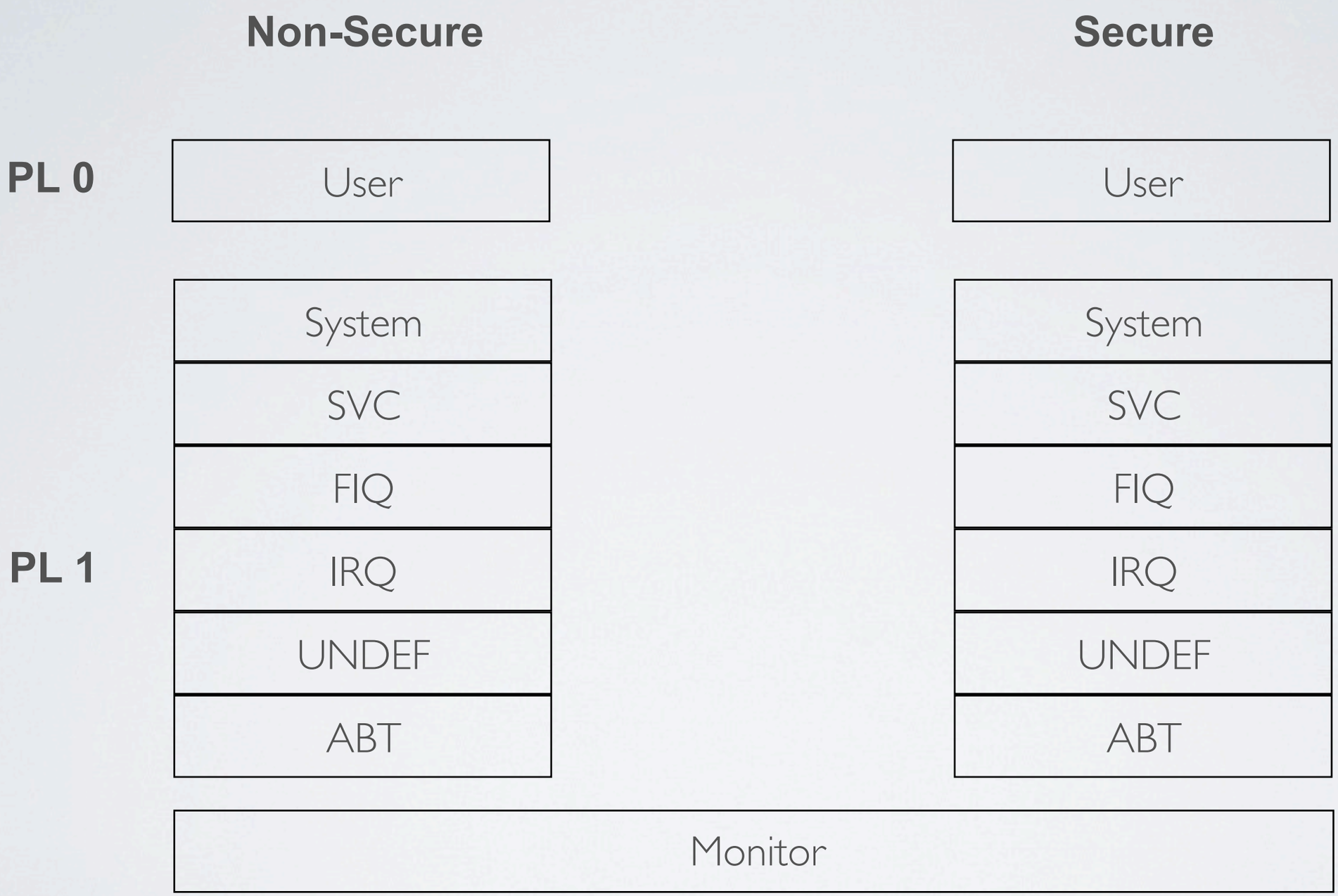
ARM Virtualization Extensions

- Hardware support for virtualization
- Hyp-mode
- Stage-2 memory translation

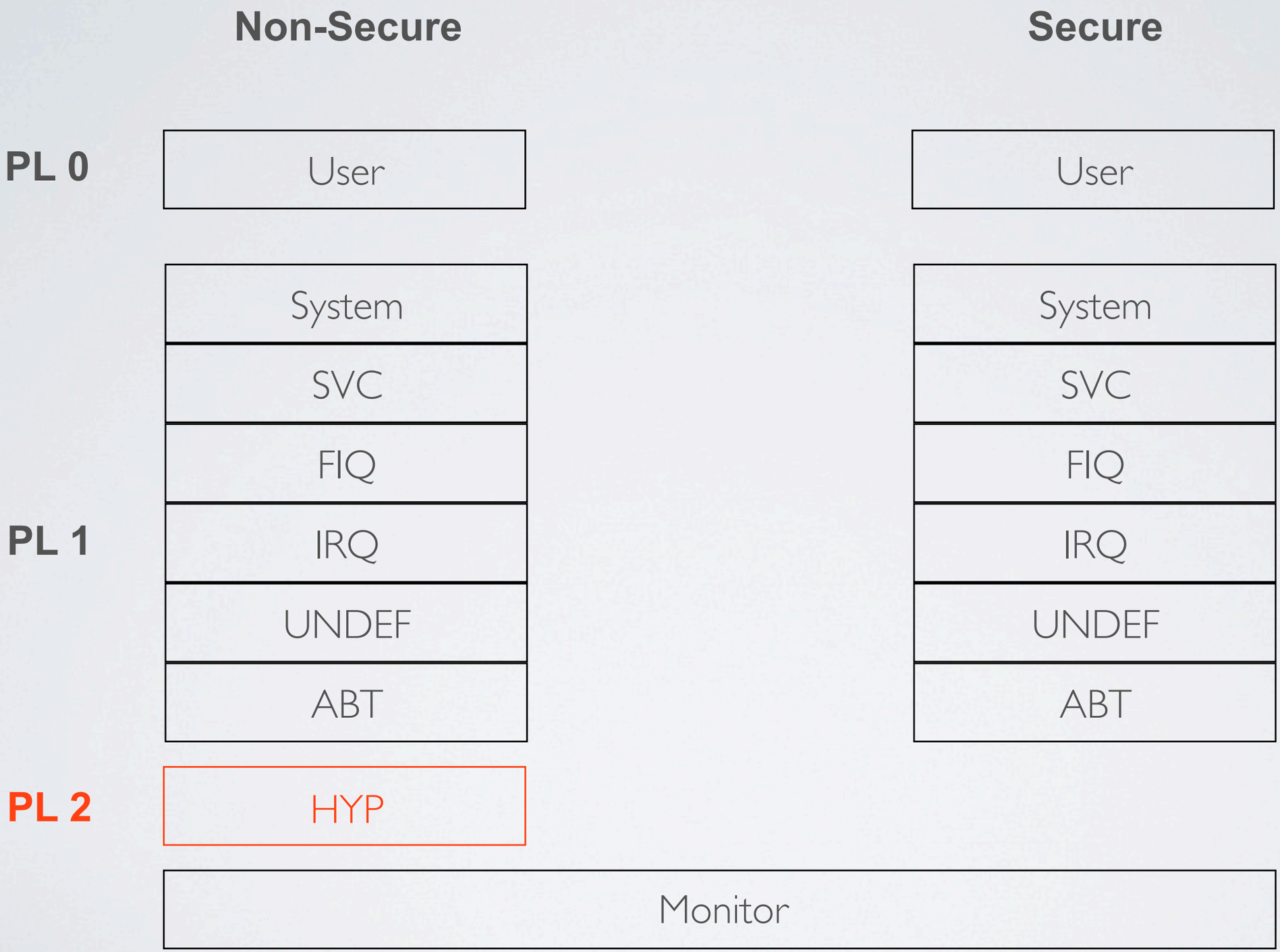
ARMv7 Processor Modes



Security Extensions (TrustZone)



Virtualization Extensions



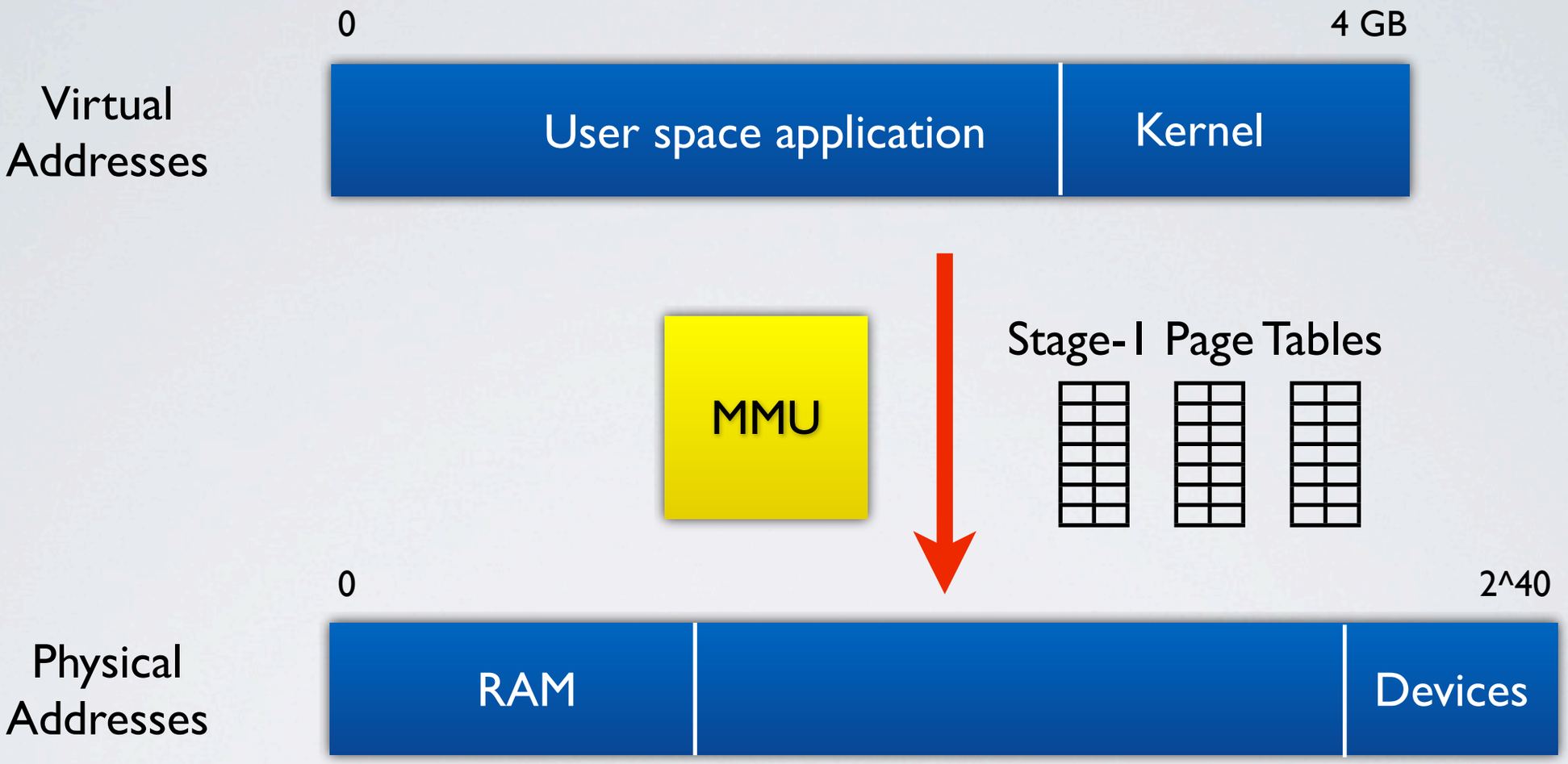
Hyp mode

- Access to hypervisor control registers
- Controls stage-2 translations
- Entry through exception (ex. HVC instruction)
- Exit by exception return

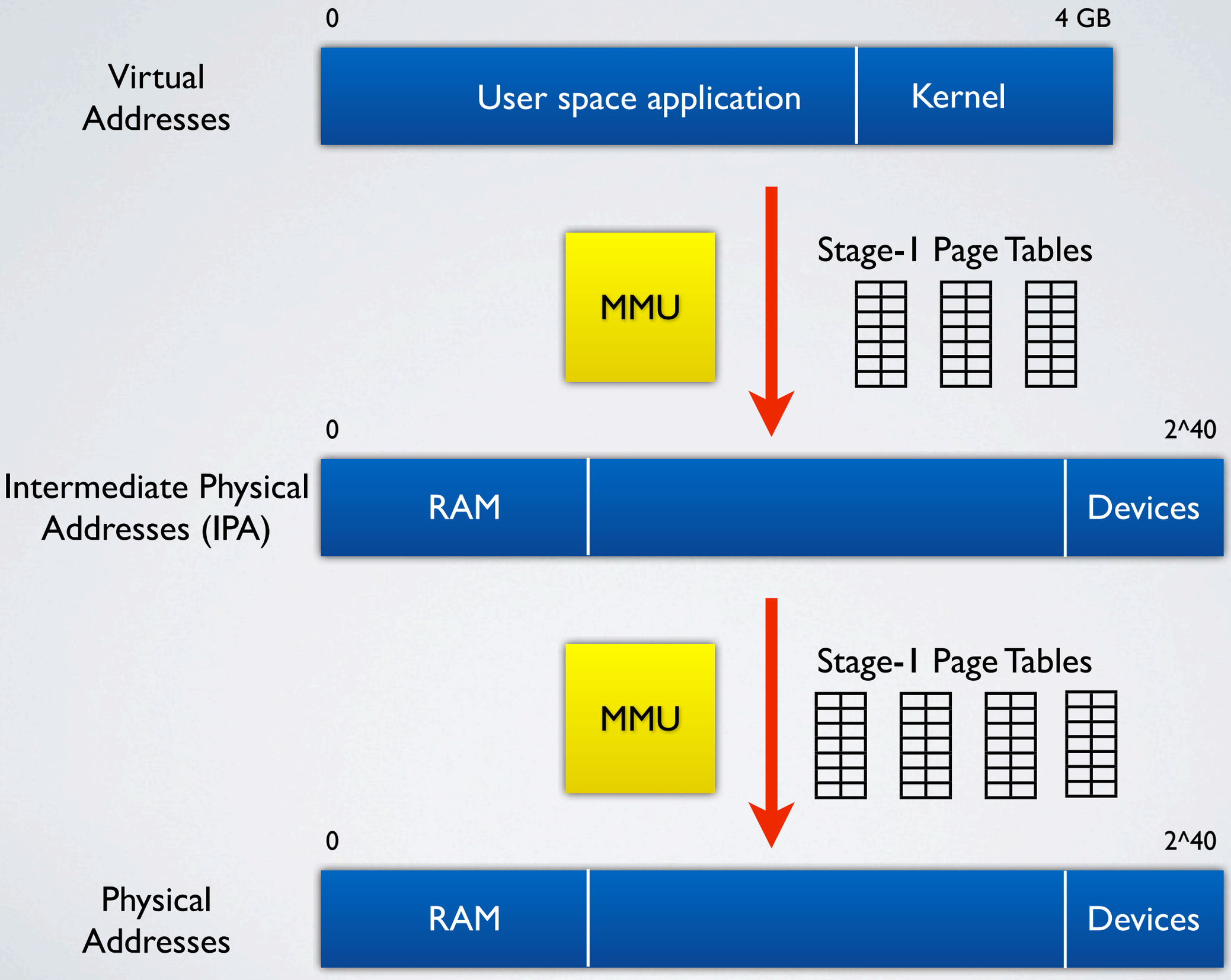
LPAE

- Large-Physical-Address-Extension (LPAE)
- 40-bit physical addresses
- 64-bit page table descriptors
- Page table format used for stage-2 translations

Memory Translation with LPAE



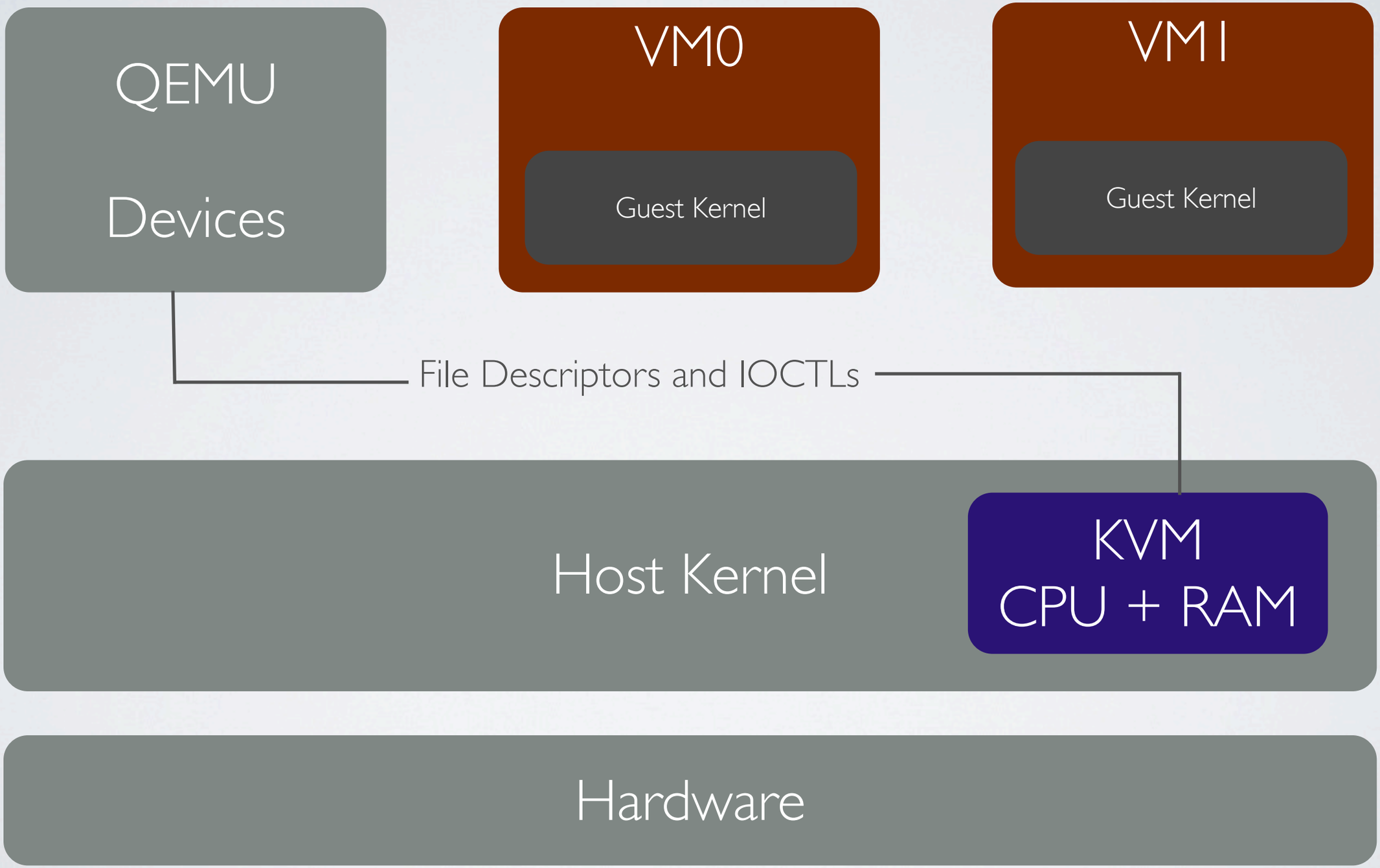
Stage-2 Page Tables



Compared to x86

- No concept of VMCS or VMRESUME / VMEXIT
- Instead, control registers must be programmed in Hyp mode
- Hyp mode is not a superset of normal privileged mode!
- We cannot run the host kernel in Hyp mode.
- Hyp mode was designed with classic embedded hypervisor in mind

KVM

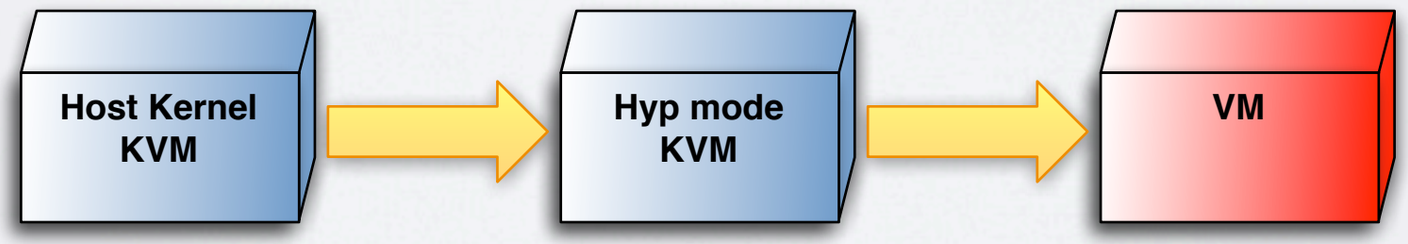

QEMU
Devices
VM0
Guest Kernel
VM1
Guest Kernel
File Descriptors and IOCTLs
Host Kernel
KVM
CPU + RAM
Hardware

User space code

```
void *mem;

kvm_fd = open("/dev/kvm");
vm_fd = ioctl(kvm_fd, KVM_CREATE_VM, 0);

posix_memalign(&mem, PAGE_SIZE, 0x20000000);
ioctl(kvm_fd, KVM_SET_USER_MEMORY_REGION, mem);

vcpu1_fd = ioctl(vm_fd, KVM_CREATE_VCPU, 0);
vcpu2_fd = ioctl(vm_fd, KVM_CREATE_VCPU, 0);

thread1_start(run_vcpu(vcpu1_fd));
thread1_start(run_vcpu(vcpu2_fd));

while (1)
    process_io();
```

KVM and Hyp mode

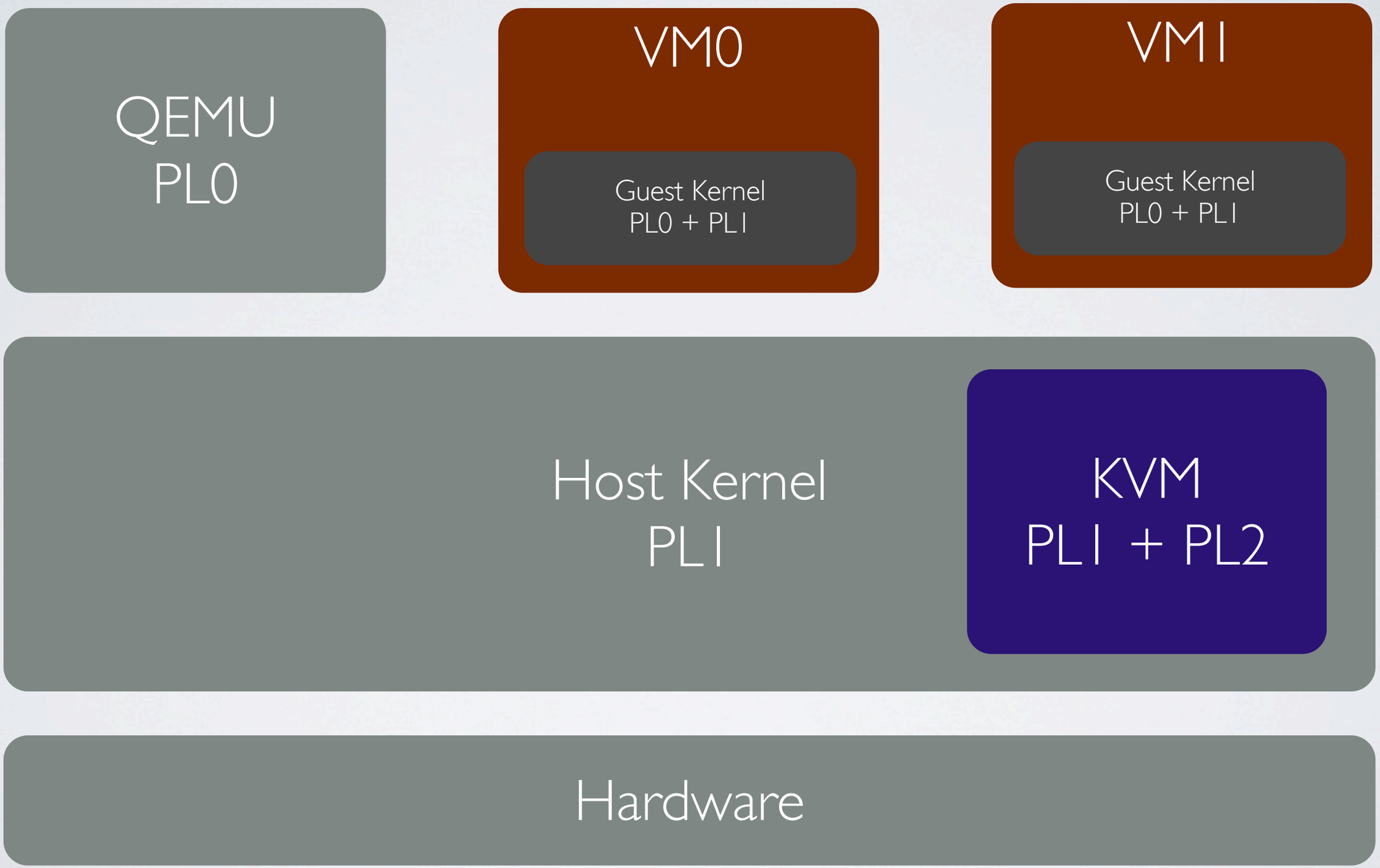
- Running VMs have to go through Hyp mode
- VM data structures must be mapped in Hyp mode



Owning Hyp mode

- HVC instruction enters Hyp mode
- CPU jumps to HVBAR + offset
- Host kernel must control HVBAR and HTTBR
- Boot the kernel in Hyp mode, not SVC!

KVM/ARM Architecture



KVM/ARM Architecture

- KVM virtualizes CPU and Memory
- QEMU virtualizes platform and devices
- Two exceptions: VGIC and Timer

CPU virtualization

- Context switching guest accessible registers
- Trap access to other registers and emulate in software

Memory Virtualization

- Built on hardware support
- VMID tagged caches and TLB
- Support swapping through MMU notifiers
- Support freeing stage-2 page tables
- Support hugetlbfs and THP (waiting on ARM support)

Handling stage-2 aborts

Handling stage-2 aborts

1. Assign a blank stage-2 PGD to the VM

Handling stage-2 aborts

1. Assign a blank stage-2 PGD to the VM

2. Run vcpu

Handling stage-2 aborts

1. Assign a blank stage-2 PGD to the VM

2. Run vcpu

3. VM generates stage-2 fault to Hyp mode

Handling stage-2 aborts

1. Assign a blank stage-2 PGD to the VM

2. Run vcpu

3. VM generates stage-2 fault to Hyp mode

4. Capture fault information

Handling stage-2 aborts

1. Assign a blank stage-2 PGD to the VM

2. Run vcpu

3. VM generates stage-2 fault to Hyp mode

4. Capture fault information

5. Create stage-2 mapping

Handling stage-2 aborts

1. Assign a blank stage-2 PGD to the VM

2. Run vcpu

3. VM generates stage-2 fault to Hyp mode

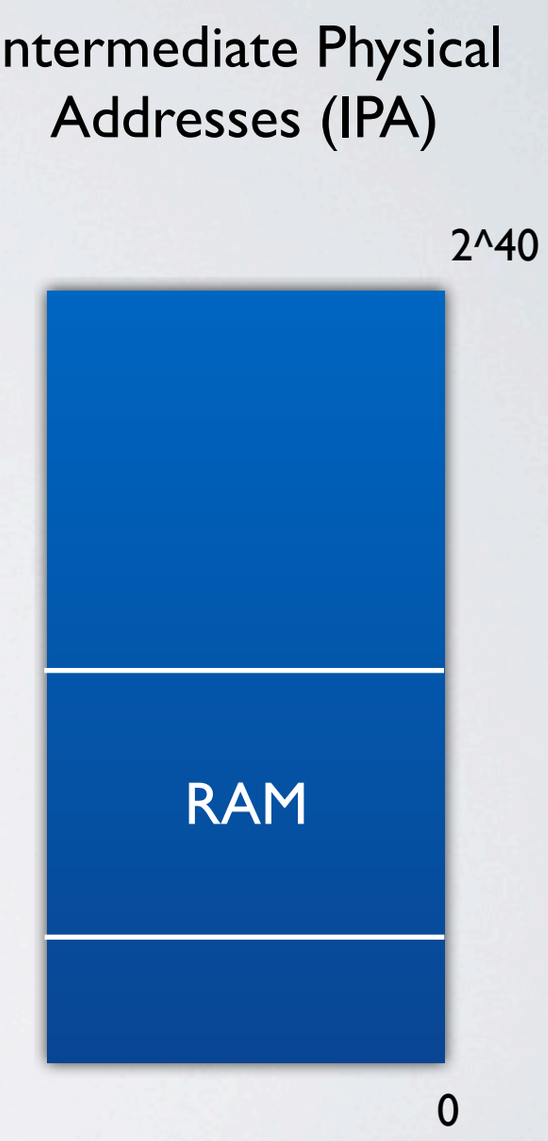
4. Capture fault information

5. Create stage-2 mapping

6. Run vcpu

I/O Virtualization

- All I/O is MMIO
- A stage-2 abort not in RAM considered I/O
- Emulation hints in HSR



Load/Store instruction decoding

Load/Store instruction decoding

```
ldr sp, {r0, #4}!
```

Load/Store instruction decoding

```
ldr sp, {r0, #4}!
```

```
addr = reg[0] + 4;
reg[sp] = mem[addr];
reg[0] = addr;
```

Load/Store instruction decoding

```
ldr sp, {r0, #4}!
```

```
addr = reg[0] + 4;
reg[sp] = mem[addr];
reg[0] = addr;
```

- No decode hints in HSR!

Load/Store instruction decoding

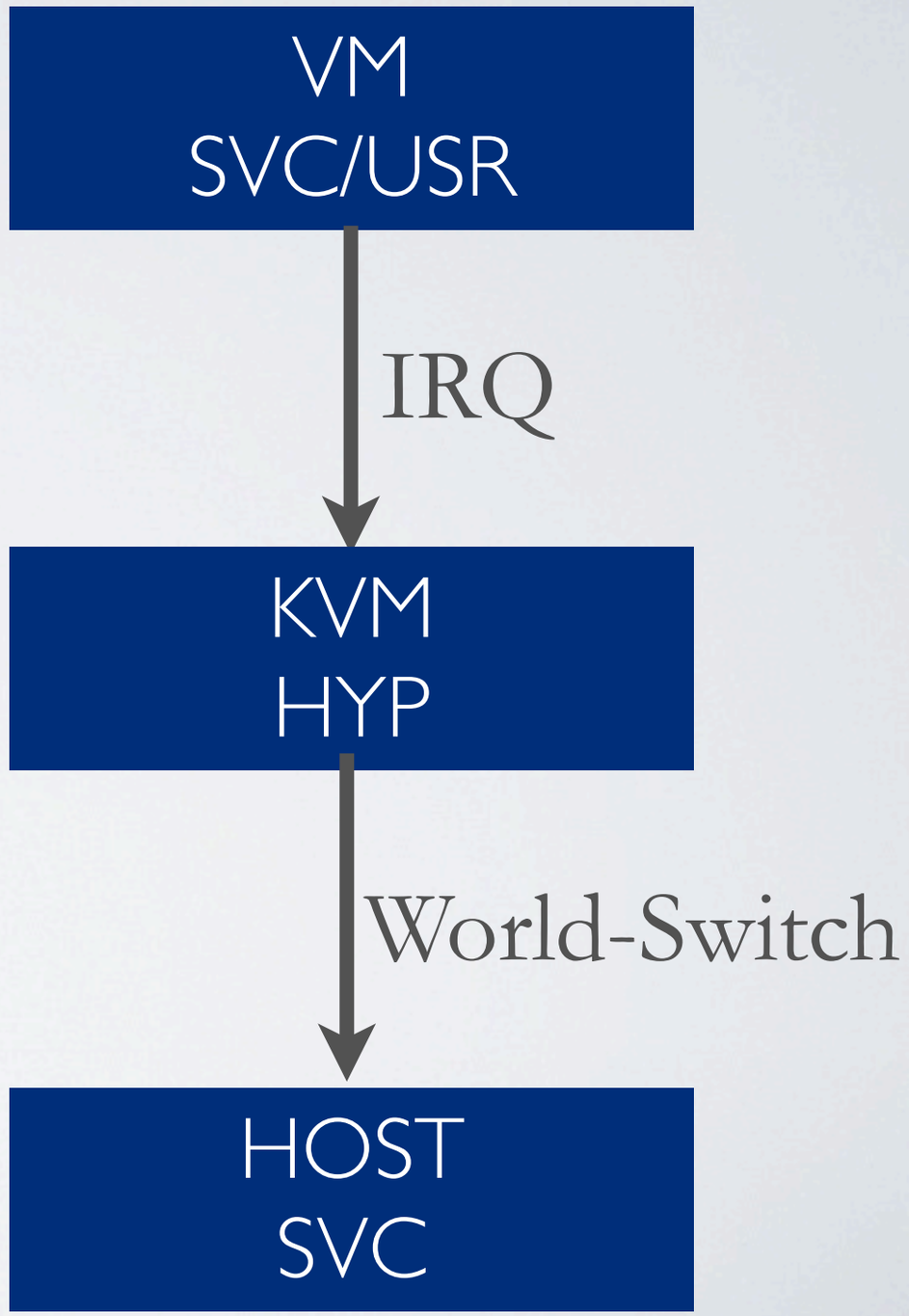
```
ldr sp, {r0, #4}!
```

```
addr = reg[0] + 4;
reg[sp] = mem[addr];
reg[0] = addr;
```

- No decode hints in HSR!
- Avoided in newer kernels

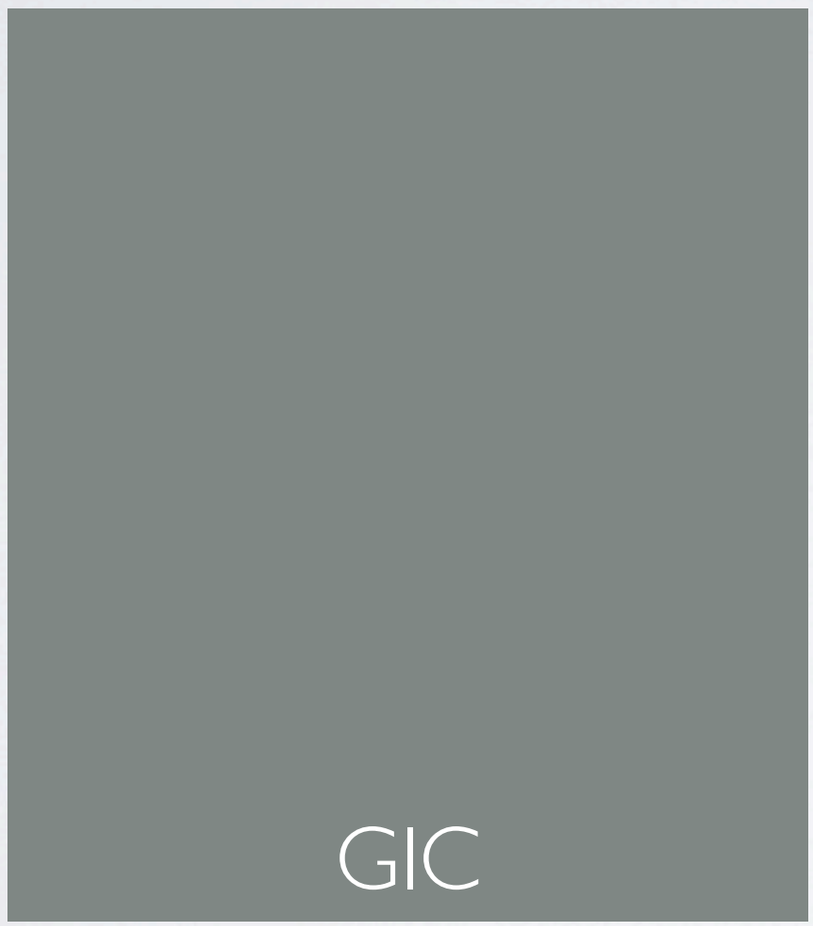
Hardware Interrupts

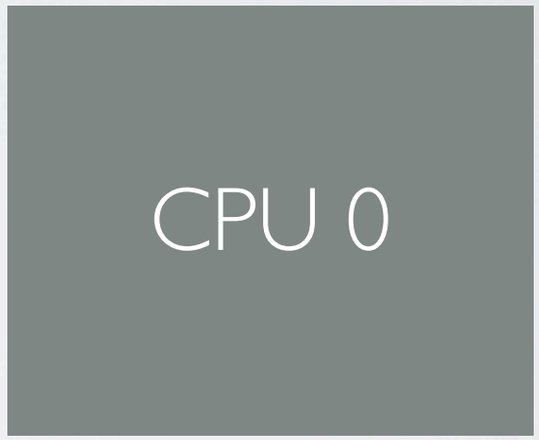
- Configured to trap to Hyp mode
- World-Switch back into host
- Re-enable IRQs

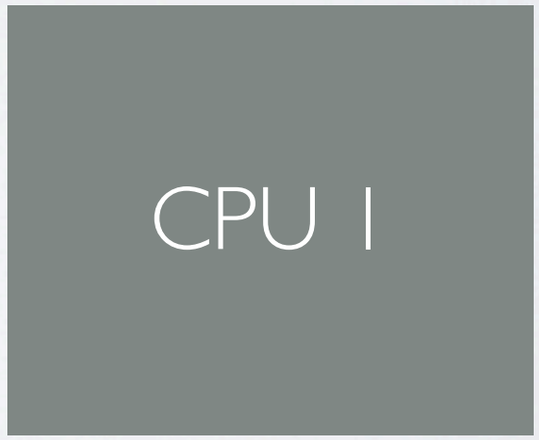


Generic Interrupt Controller

Devices

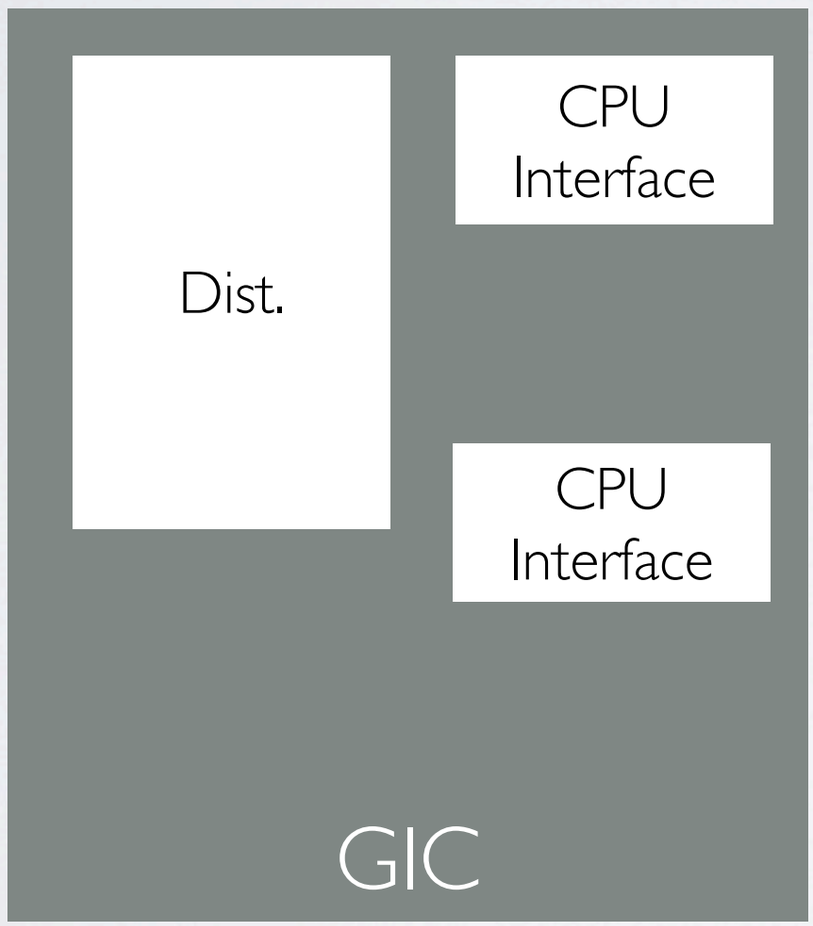


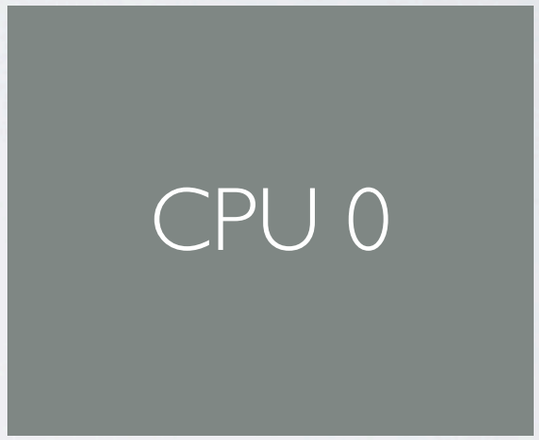


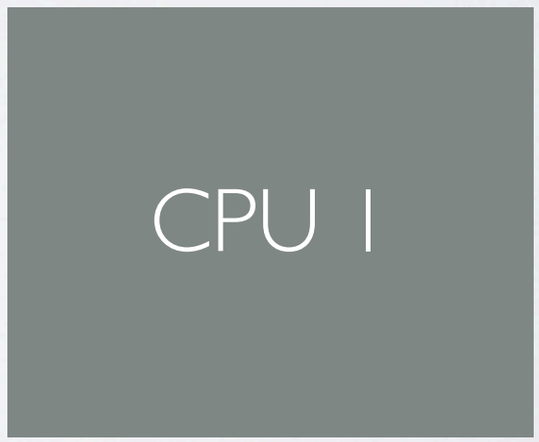


Generic Interrupt Controller

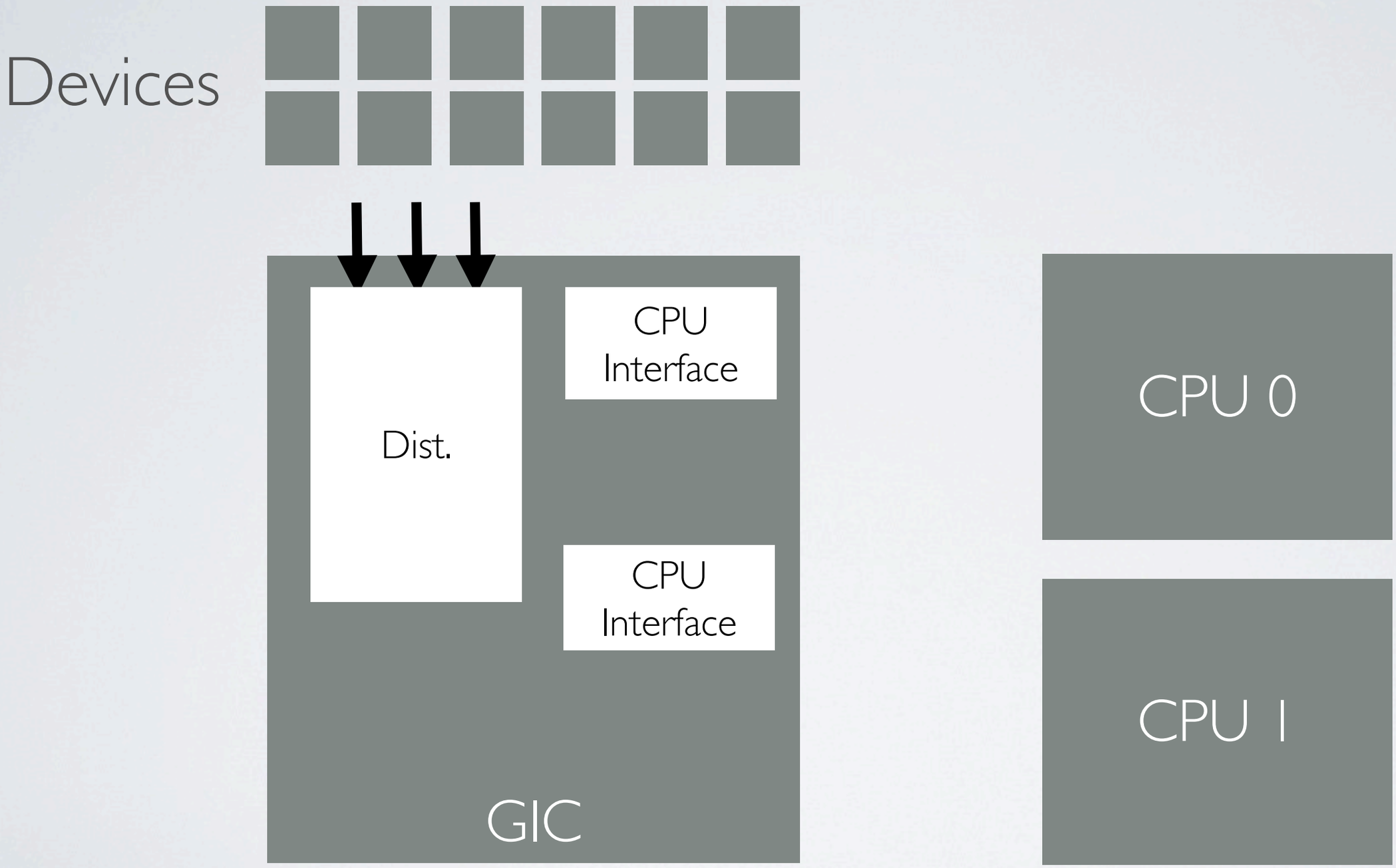
Devices



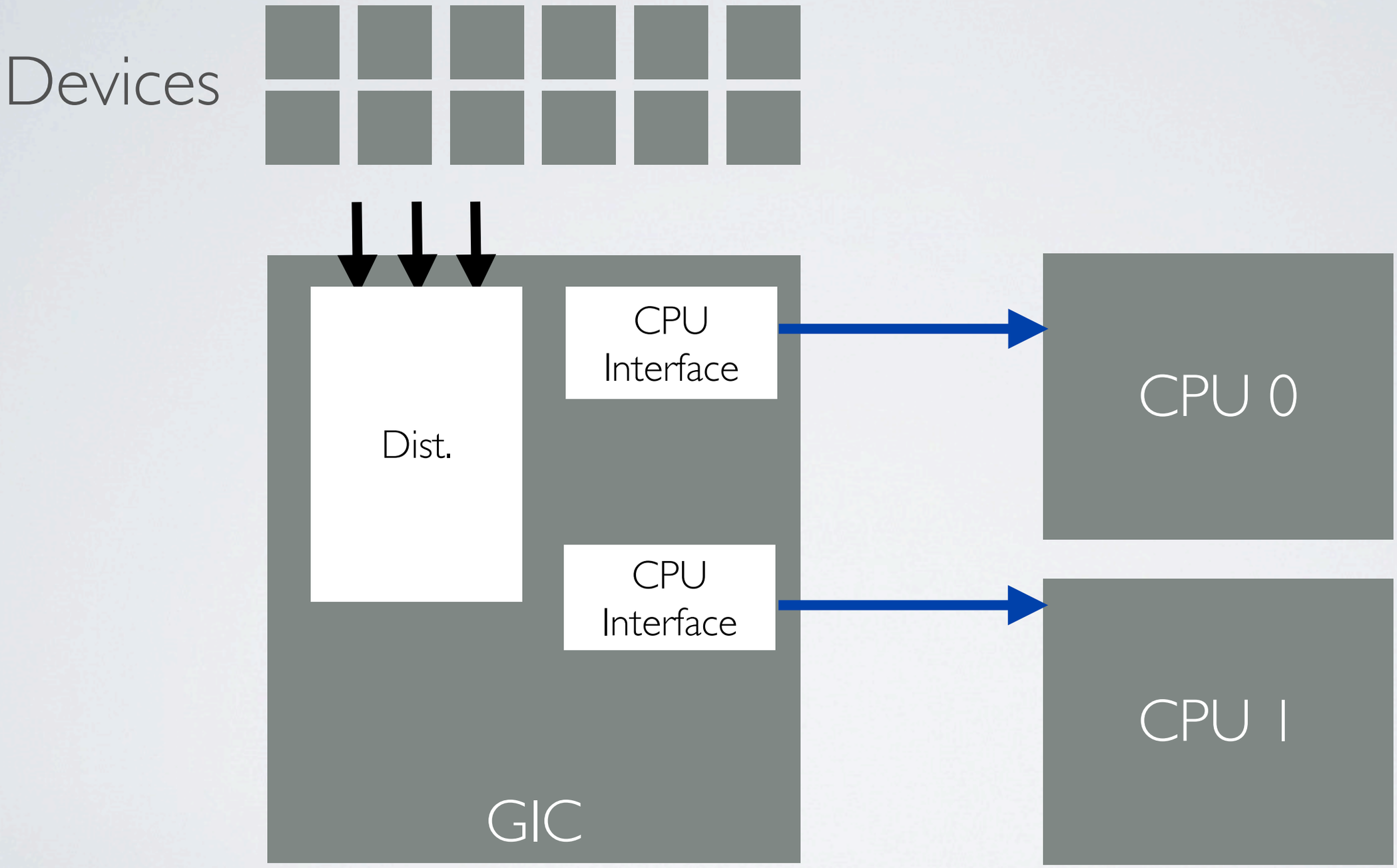




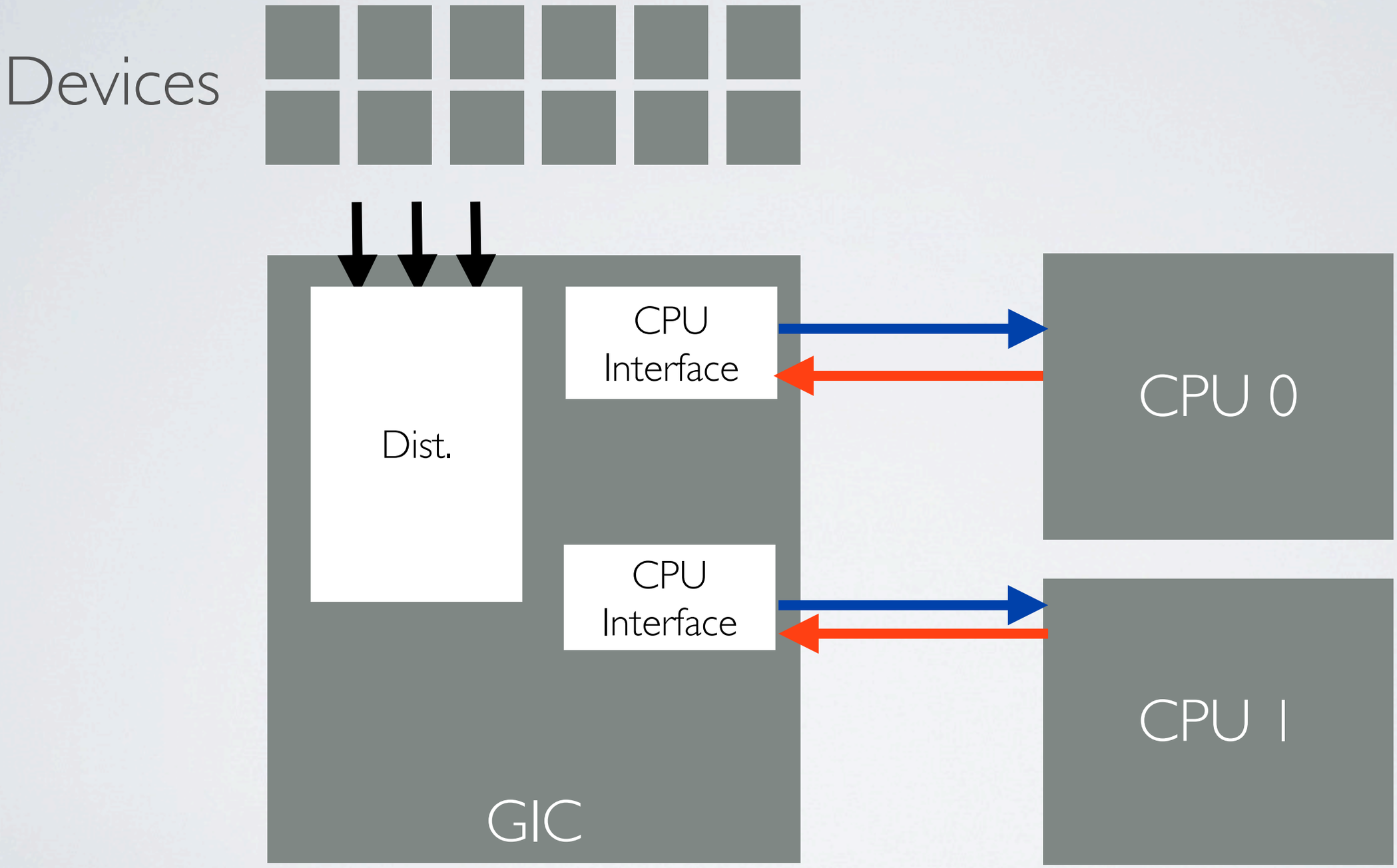
Generic Interrupt Controller



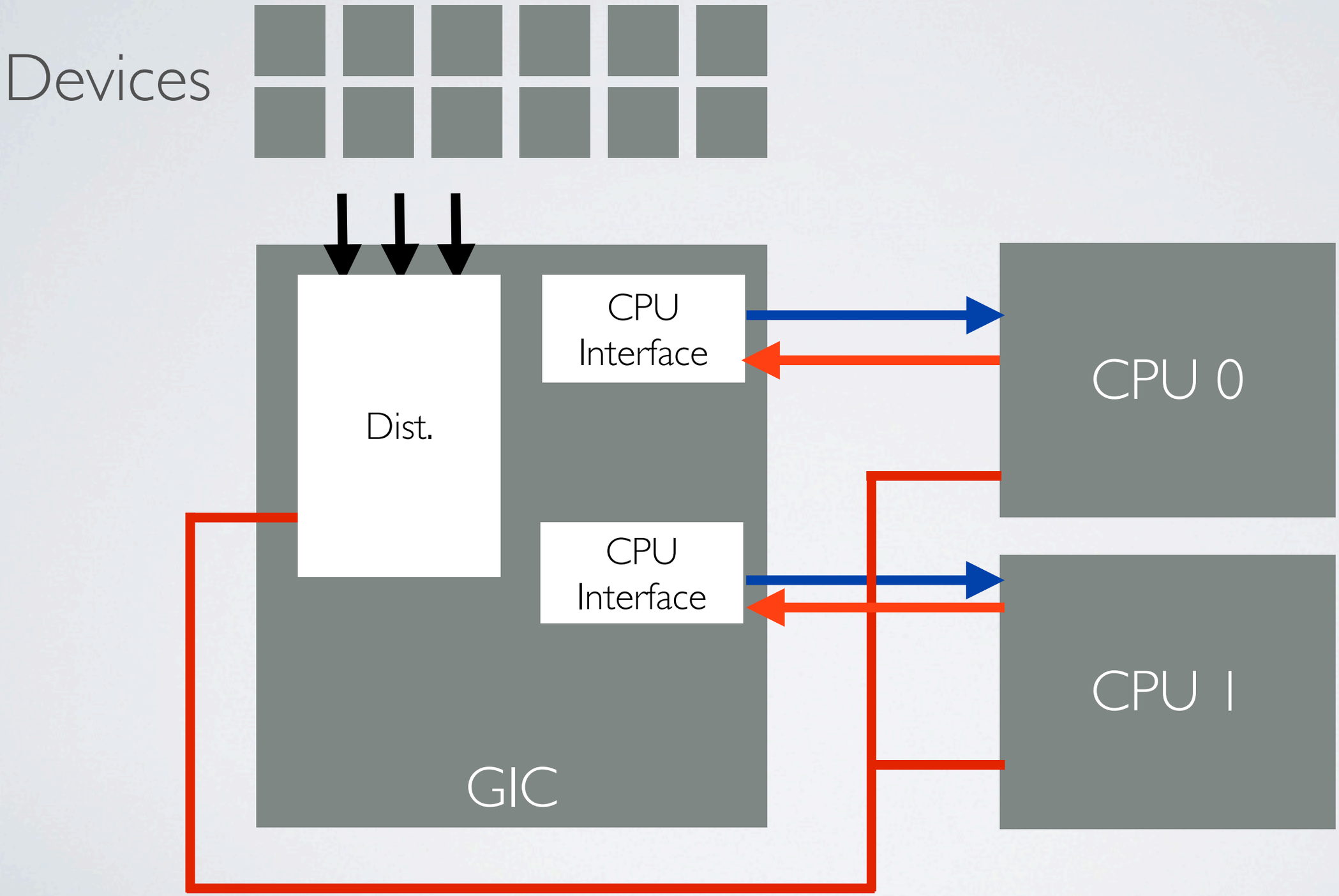
Generic Interrupt Controller



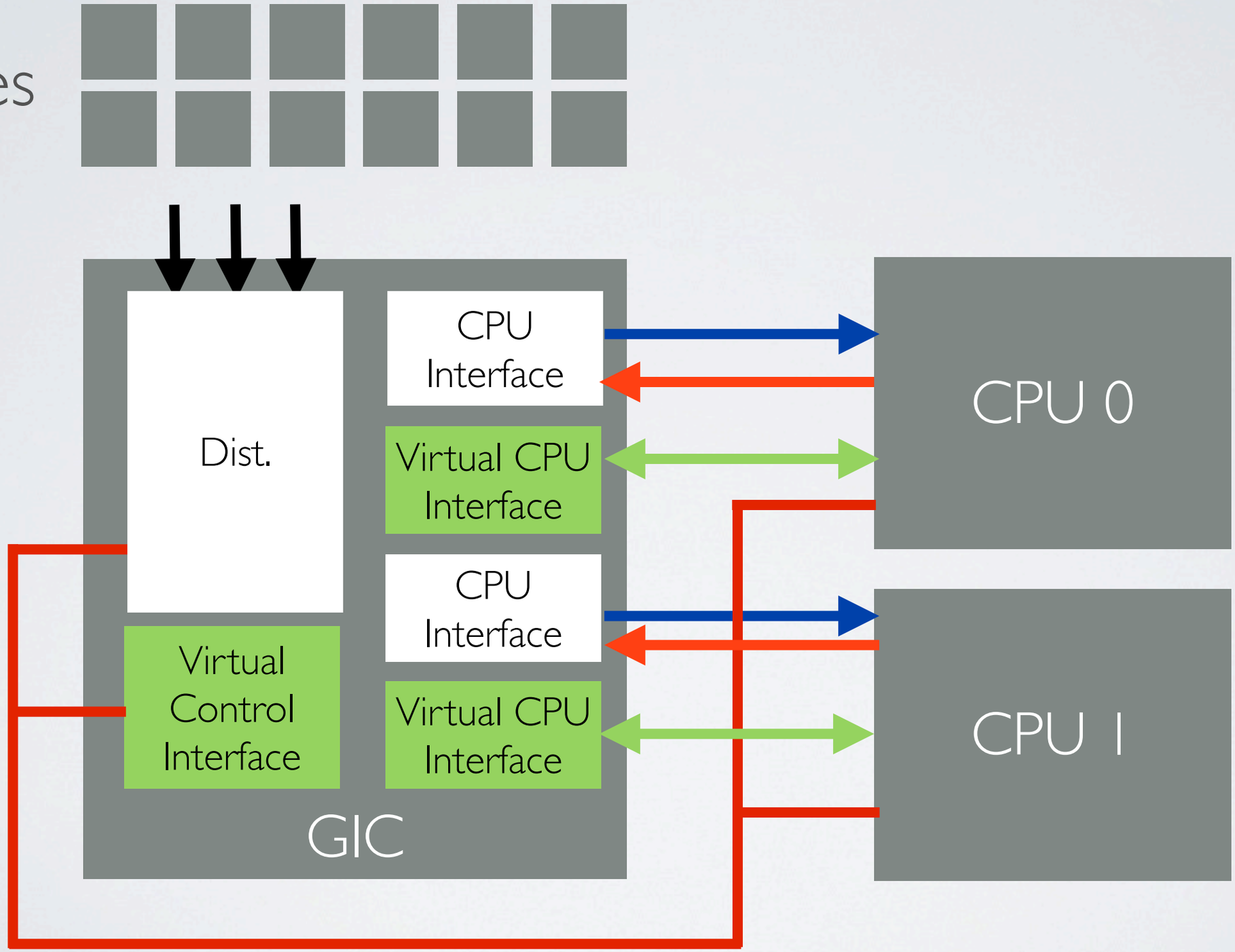
Generic Interrupt Controller



Generic Interrupt Controller



Generic Interrupt Controller



Virtual Interrupts

- Generated by QEMU devices
- KVM_IRQ_LINE
- Programmed in VGIC list registers
- No virtual distributor interface

Architected Generic Timers

- Defines physical + virtual timer and counter
- Virtual counter = Physical counter + offset
- Counter reads from guest without trap
- Timer interrupts do trap

DEMO

Voodoo Bug

Voodoo Bug

```
page = __get_user_pages(...);

...

if (map_writable)
    SetPageDirty(page);
stage2_set_pte(page, ...);
put_page(page);
```

Voodoo Bug

```
page = __get_user_pages(...);

...

SetPageDirty(page);
stage2_set_pte(page, ...);
put_page(page);
```

Clues?

Questions?