

Kernel Interrupt: A Major Overhaul

- APIC Initialization

&

- Vector Allocation

Dou Liyang douly.fnst@cn.fujitsu.com

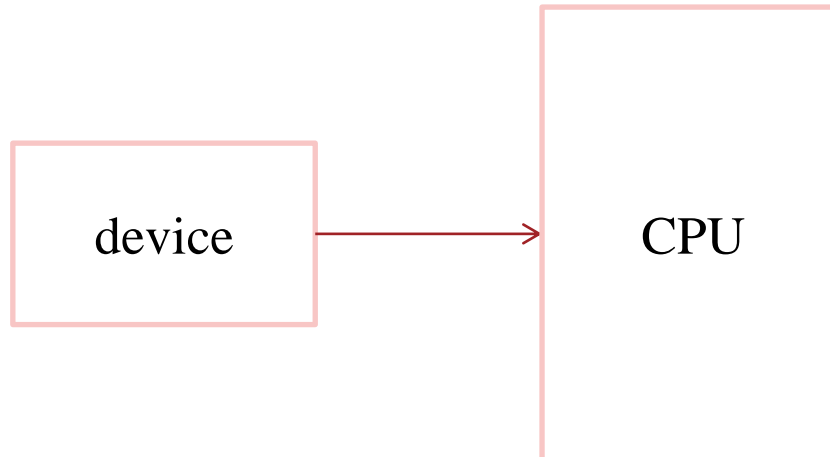
June 20 2018

What's next?

- *Basics of an interrupt*
- Overhaul of interrupt
 - APIC Initialization
 - Vector Allocation
- Future work

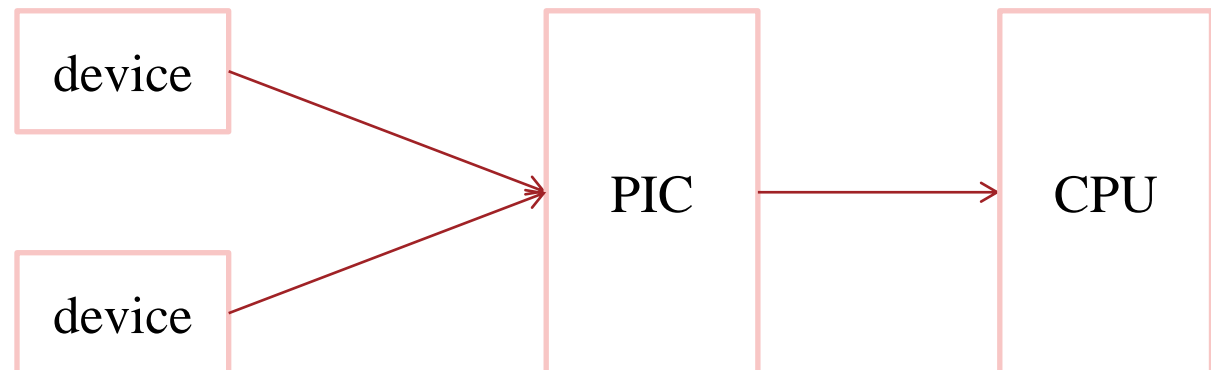
What is An Interrupt?

- A hardware signal
- Emitted from a peripheral to a CPU
- Indicating that a device-specific condition has been satisfied



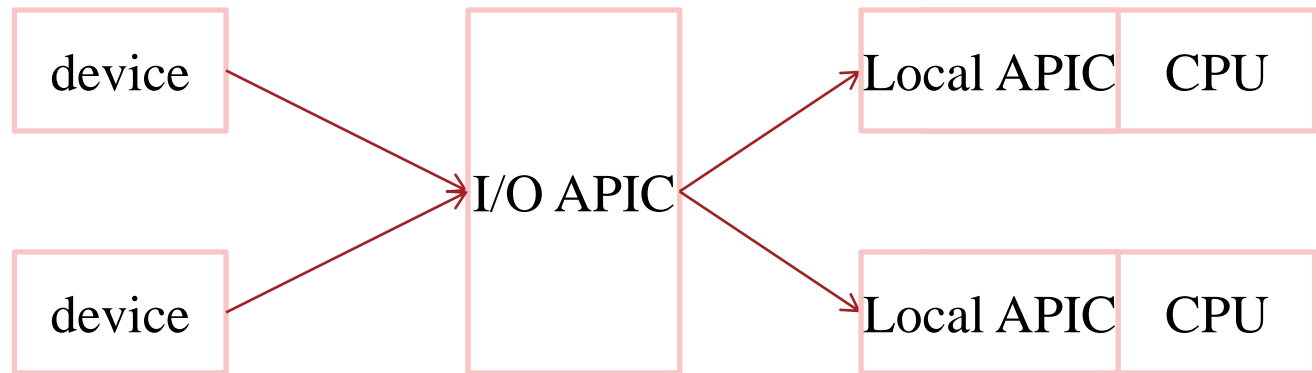
Multiplexing Interrupts

- Having a single interrupt for the CPU is usually not enough
- Most systems have tens, hundreds of them
- ***An interrupt controller*** allows them to be multiplexed
- Very often architecture or platform specific
- In old x86 machine, there was a PIC called 8259A
 - a chip responsible for sequentially processing multiple interrupt requests from multiple devices
 - Called ***PIC Mode***

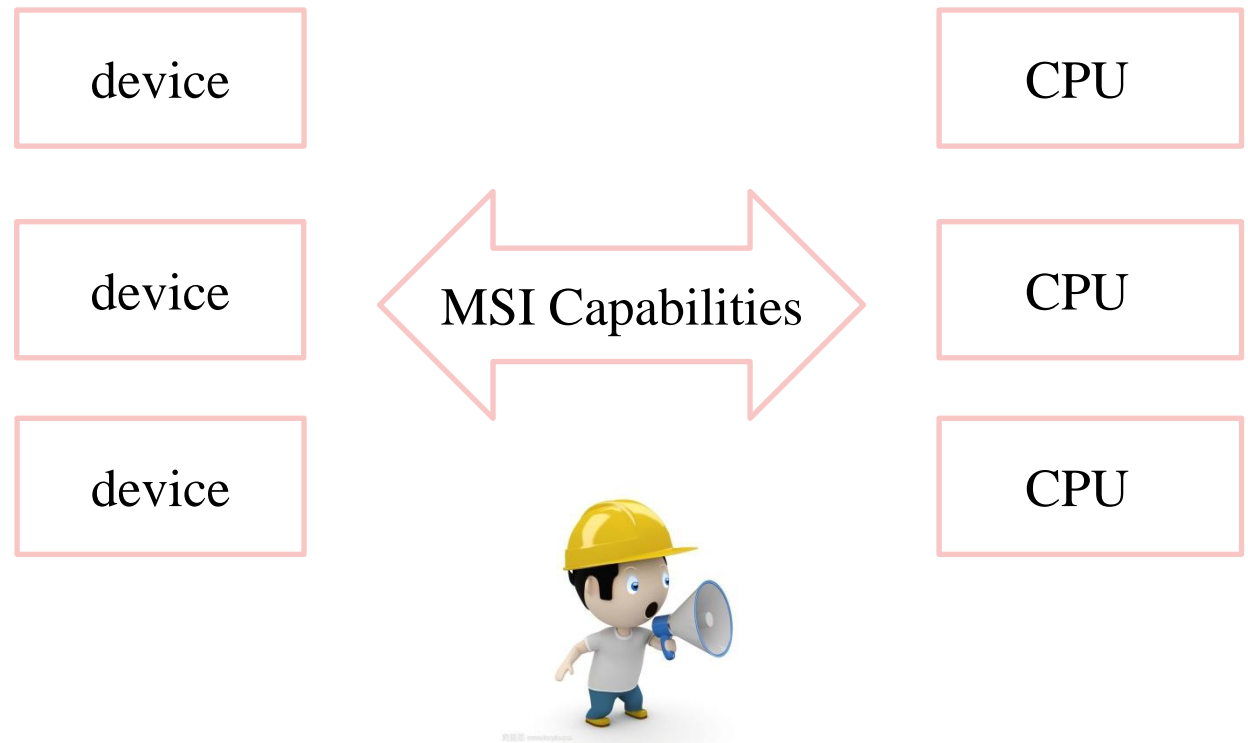


Multiplexing Interrupts in SMP System

- Only a CPU is usually not enough
- Most systems have tens, hundreds of CPUs
- An new interrupt controller should be used
- In x86 machine, there is an **APIC**
 - Local APIC is located on each CPU core, handles the CPU-specific interrupt configuration
 - I/O APIC distribute external interrupts from multiple devices to multiple CPU cores
 - Called *Symmetric I/O Mode*

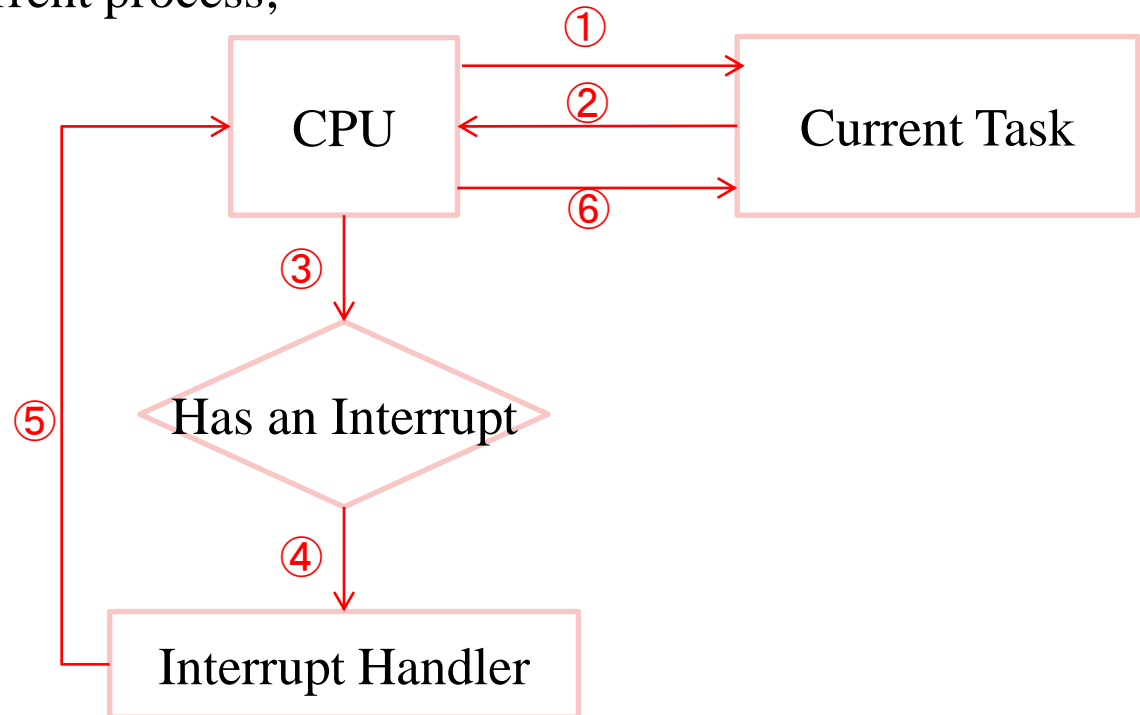


- ***Message Signaled Interrupts*** are as an alternative to line-based interrupts
 - Trigger an interrupt by writing a value to a particular memory
 - Allow the use of the same buses as the data



Handle an Interrupt

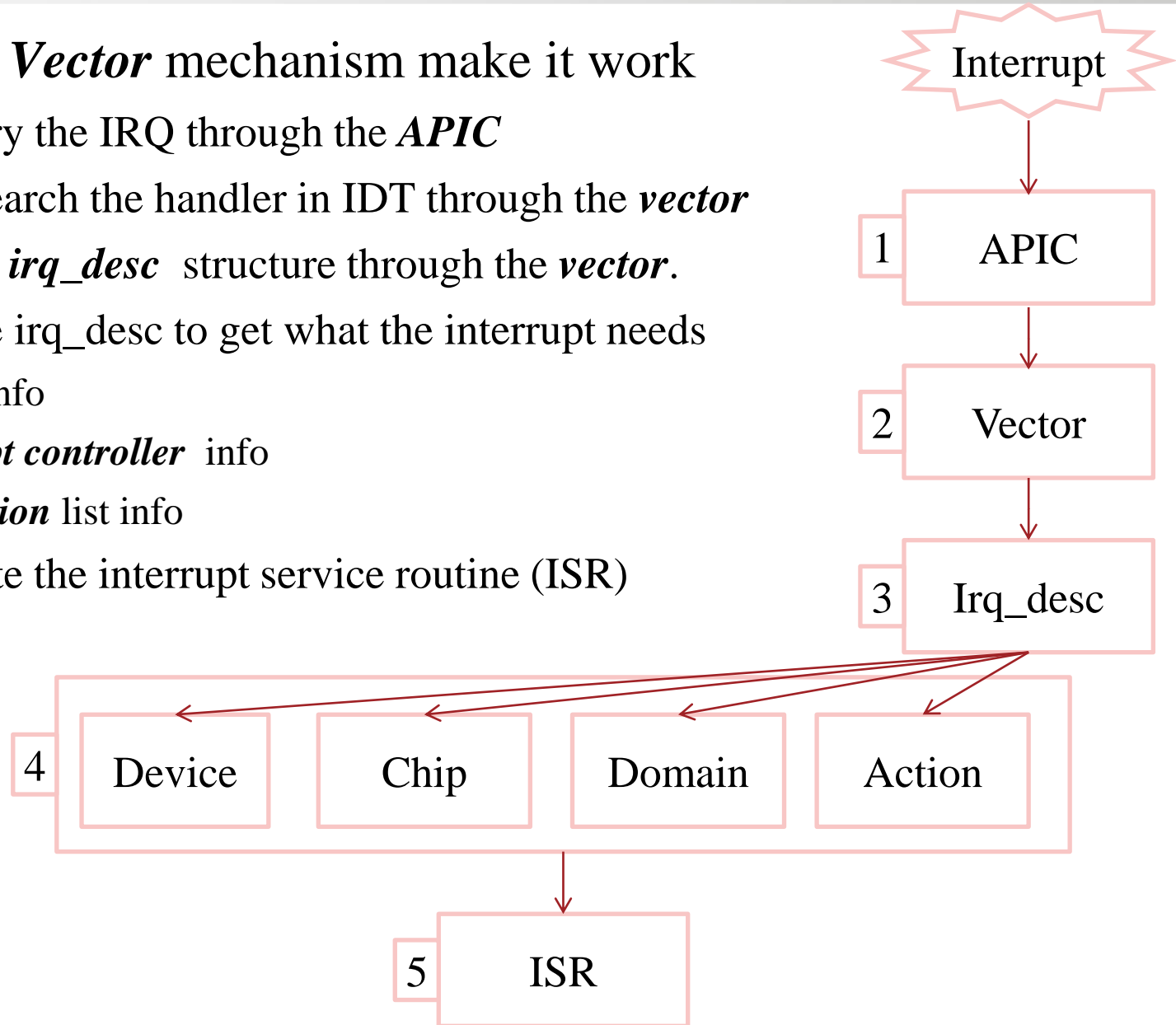
- Preempt current task ① ②
 - *Pause execution* of the current process.
- Execute interrupt handler ③ ~ ⑤
 - Search for *the handler of the interrupt* and transfer control
- Resume the task ⑥
 - *Return to execute* the current process;



How Does “Handle an Interrupt” Work?

■ *APIC* and *Vector* mechanism make it work

1. Delivery the IRQ through the *APIC*
2. CPU search the handler in IDT through the *vector*
3. Get the *irq_desc* structure through the *vector*.
4. Use the *irq_desc* to get what the interrupt needs
 - *device* info
 - *interrupt controller* info
 - *IRQ action* list info
5. Execute the interrupt service routine (ISR)

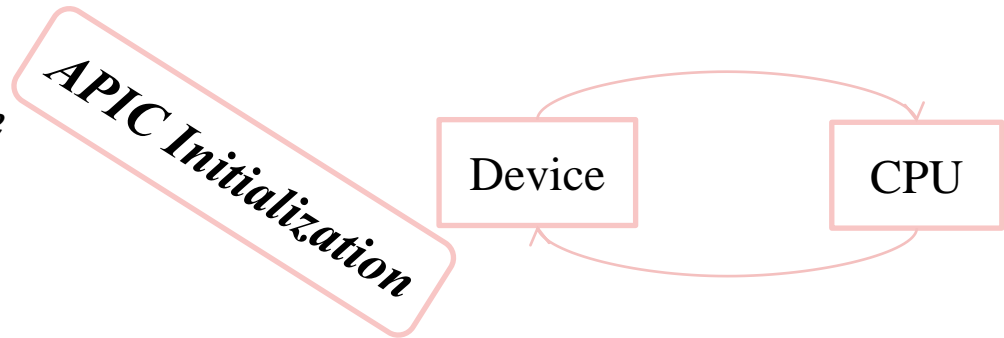


Why “APIC and Vector” Can Work?

■ *Do many initialization and setup works* when Linux boots up

■ For the interrupt delivery

- Initialize 8259A
- Switch *interrupt delivery mode*
- Initialize APIC
 - Local APIC setup
 - I/O APIC setup

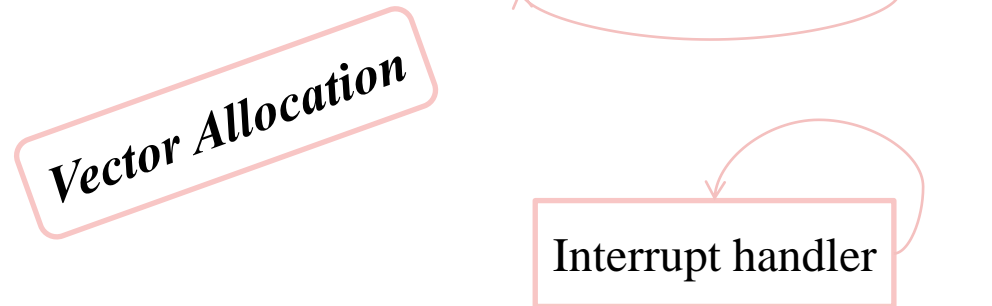


■ For IDT table,

- Initialize the mapping of Vector and Handler

■ For each Interrupt,

- Allocate an IRQ
- Allocate an irq_desc
- Assign a vector



What's next?

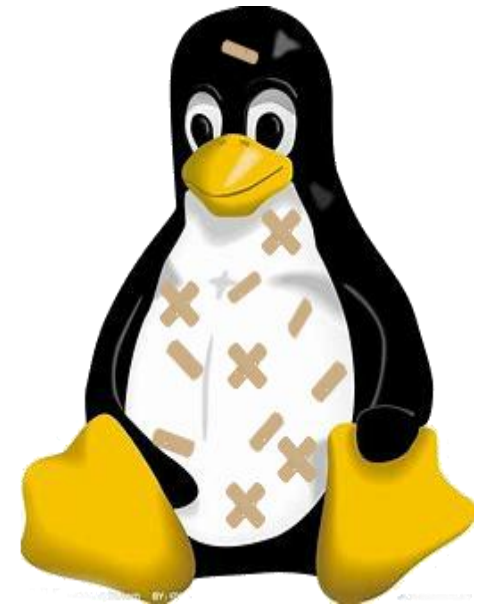
- Basics of an interrupt

- *Overhaul of interrupt*
 - APIC Initialization
 - Vector Allocation

- Future work

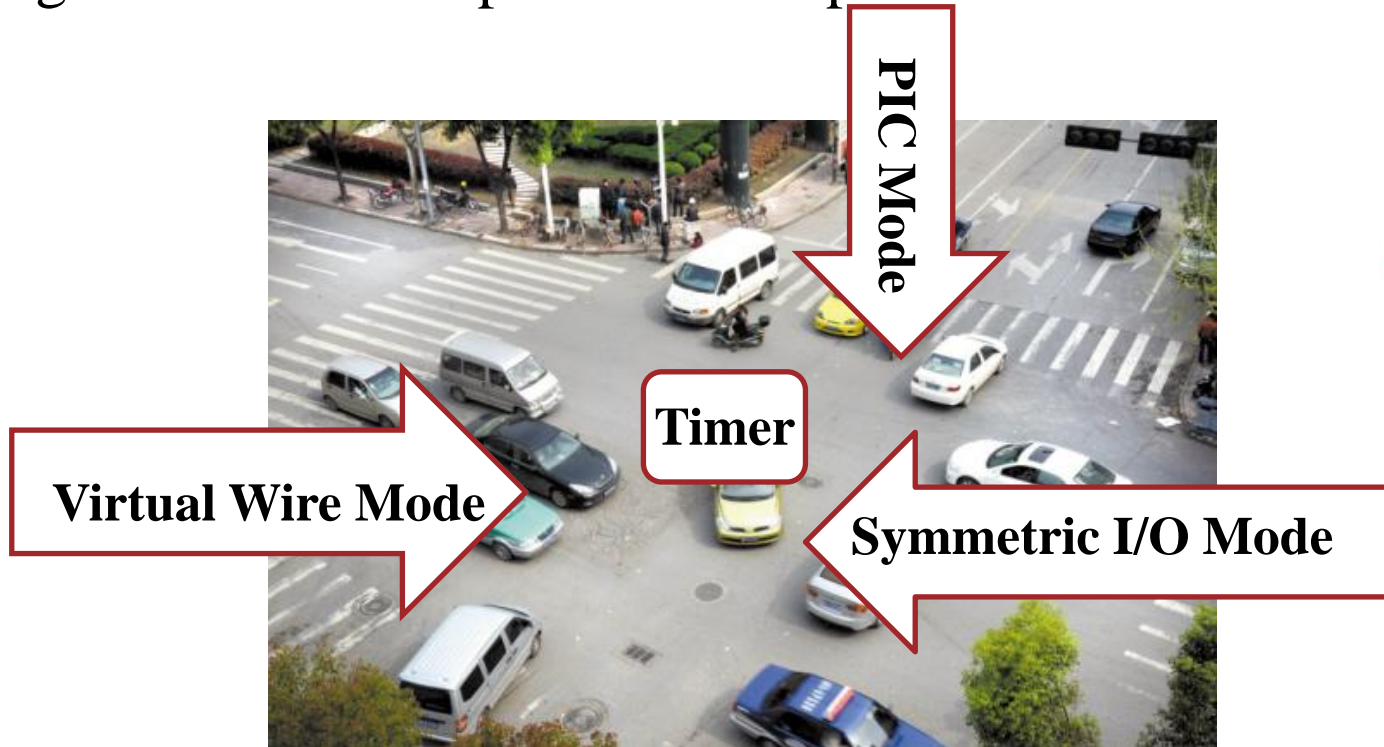
- Interrupt in x86 is a *conglomerate* of ancient bits and pieces
 - Subject to '*modernization*' and *features* over the years
 - *Kdump*
 - CPU Hotplug/System hibernation
 - Multi-queue devices

- It looks like a penguin full of band-aids
 - Can work, but can't see how it works easily.



Problems of APIC Initialization

- Horrible interrupt mode setup
 - Setup the mode **at random places**
 - Run the kernel with **the potentially wrong mode**
- Tangle the timer setup with interrupt initialization



Overhaul of APIC Initialization

- 1. Unify the APIC and interrupt mode setup
 - Construct a **selector** for the interrupt delivery mode

Kconfig

- CONFIG_X86_64
- CONFIG_X86_LOCAL_APIC
- CONFIG_x86_IO_APIC
- CONFIG_SMP

CPU Capability

- boot_cpu_has(X86_FEATURE_APIC)

MP table

- smp_found_config

ACPI table

- acpi_lapic
- acpi_ioapic
- nr_ioapic

Command line options

- disable_apic
- skip_ioapic_setup
- nolapic/noapic/apic=



PIC Mode

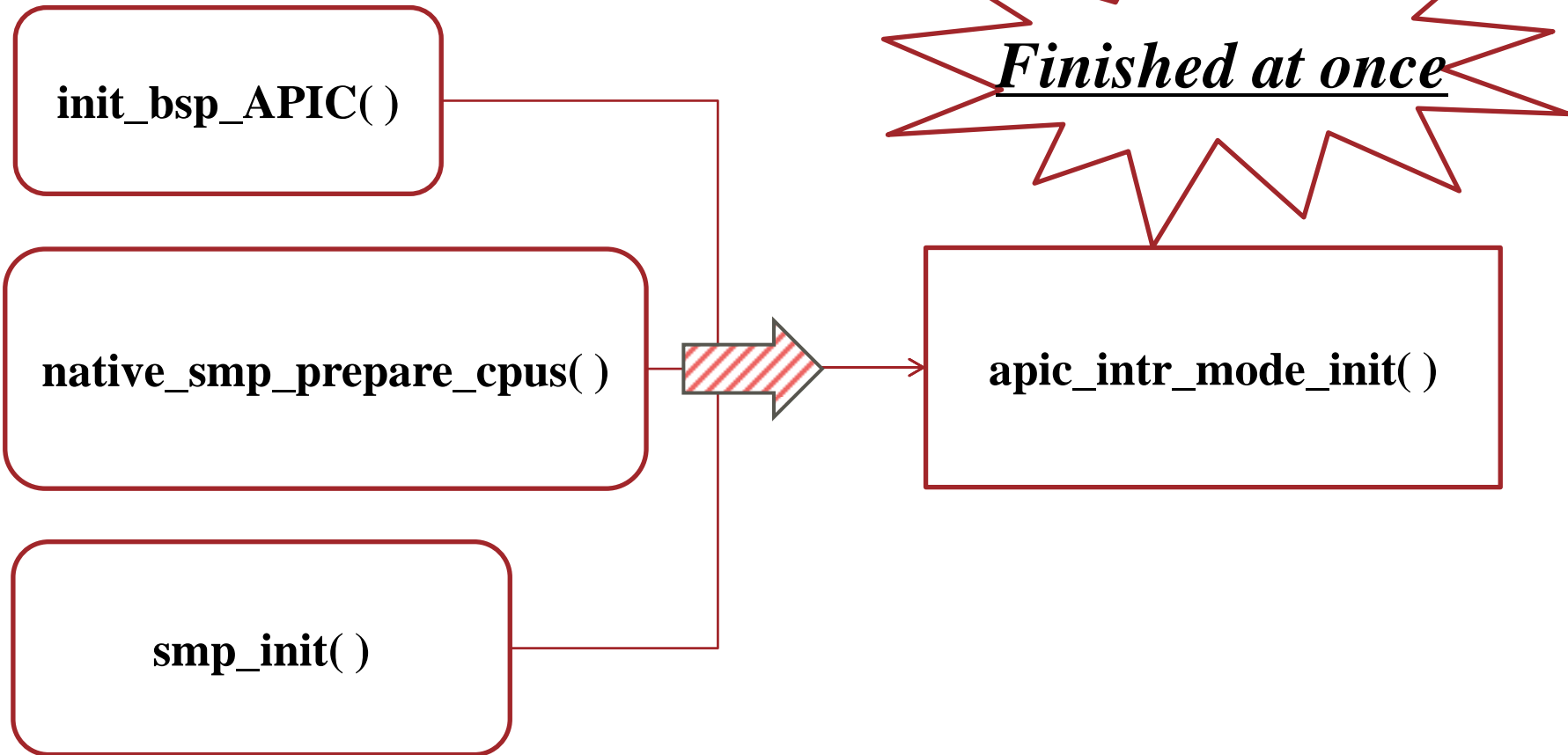
Virtual Wire Mode

Symmetric I/O Mode

See arch/x86/kernel/apic/apic.c apic_intr_mode_select()

Overhaul of APIC Initialization

- 1. Unify the APIC and interrupt mode setup
 - Provide a single function

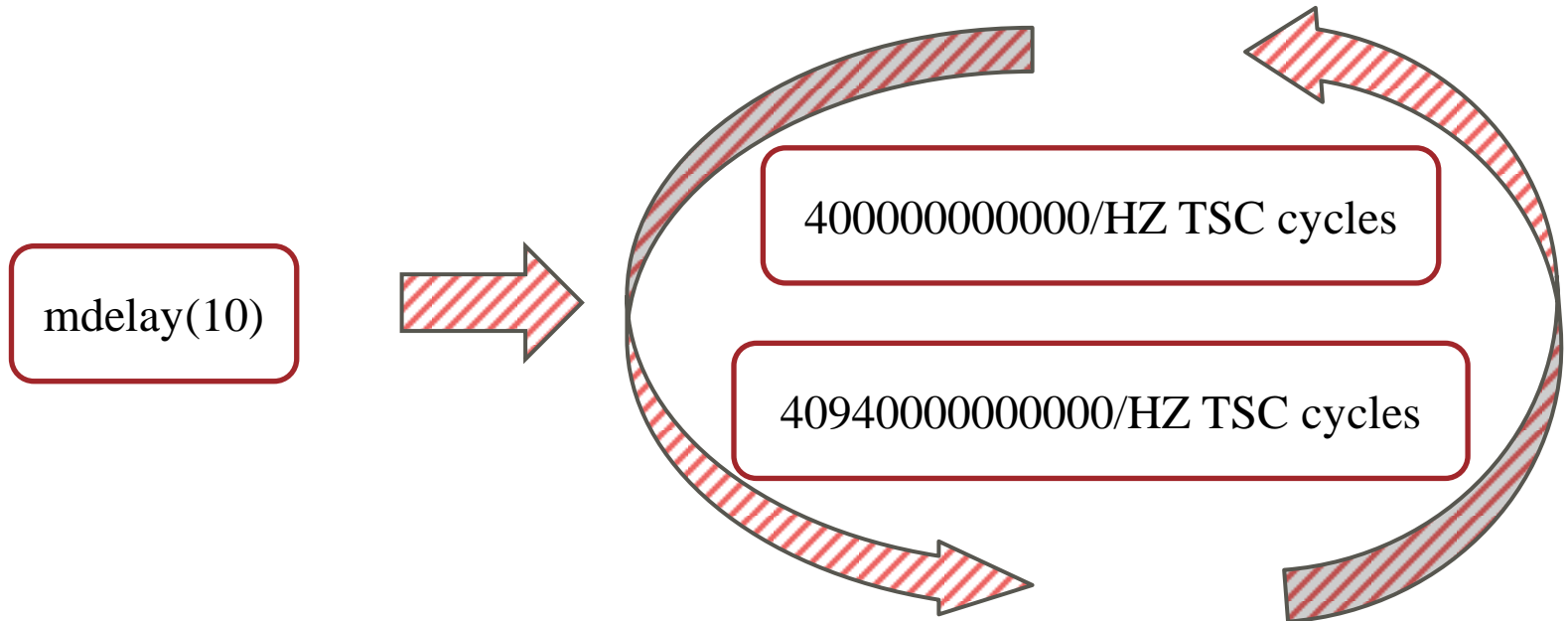


See arch/x86/kernel/apic/apic.c apic_intr_mode_init()

■ 2. Disentangle the timer setup from the APIC initialization

■ Refactor **the delay logic** during APIC initialization process.

- Either use TSC or a simple delay loop to make a rough delay estimate



■ Split local APIC timer setup from the APIC setup

■ 3. Reorganize the interrupt initialization

■ Set up the final interrupt delivery mode **as soon as possible**.

1) Set up the legacy timer(PIT/HPET)

`x86_init.timers.timer_init()`

2) Set up APIC/IOAPIC

`x86_init.irqs.intr_mode_init()`

3) TSC calibration

`tsc_init()`

4) Local APIC timer setup

`x86_init.timers.setup_percpu_clockev()`

■ 4. Some others

- Refactor some common APIC function
- Compatible with ACPI initialization
- Bypass the hypervisor, Such as KVM and Xen

■ 5. Can check which mode the interrupt is by ‘dmesg’:

```
[ 0.000000] Kernel command line: BOOT_IMAGE=/vmlinuz-4.16.0 root=UUID=10f10326-c923-4098-86aa-4
[ 0.000000] Memory: 1465920K/2096616K available (12300K kernel code, 2367K rwdats, 3948K rodat
[ 0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=4, Nodes=1
[ 0.000000] ftrace: allocating 35971 entries in 141 pages
[ 0.000000] Hierarchical RCU implementation.
[ 0.000000] RCU restricting CPUs from NR_CPUS=8192 to nr_cpu_ids=4.
[ 0.000000] Tasks RCU enabled.
[ 0.000000] RCU: Adjusting geometry for rcu_fanout_leaf=16, nr_cpu_ids=4
[ 0.000000] NR_IRQS: 524544, nr_irqs: 456, preallocated irqs: 16
[ 0.000000] Offload RCU callbacks from CPUs: .
[ 0.000000] Console: colour VGA+ 80x25
[ 0.000000] console [tty0] enabled
[ 0.000000] console [ttyS0] enabled
[ 0.000000] ACPI: Core revision 20180105
[ 0.000000] ACPI: 1 ACPI AML tables successfully acquired and loaded
[ 0.000000] clocksource: hpet: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 191126044
[ 0.003000] APIC: Switch to symmetric I/O mode setup
[ 0.006000] ..TIMER: vector=0x30 apic1=0 pin1=2 apic2=-1 pin2=-1
[ 0.011000] tsc: Fast TSC calibration using PIT
[ 0.012000] tsc: Detected 3292.164 MHz processor
[ 0.013000] tsc: Marking TSC unstable due to TSCs unsynchronized
[ 0.014334] Calibrating delay loop (skipped), value calculated using timer frequency.. 6584.32
```

What's next?

- Basics of an interrupt

- *Overhaul of interrupt*
 - APIC Initialization
 - *Vector Allocation*

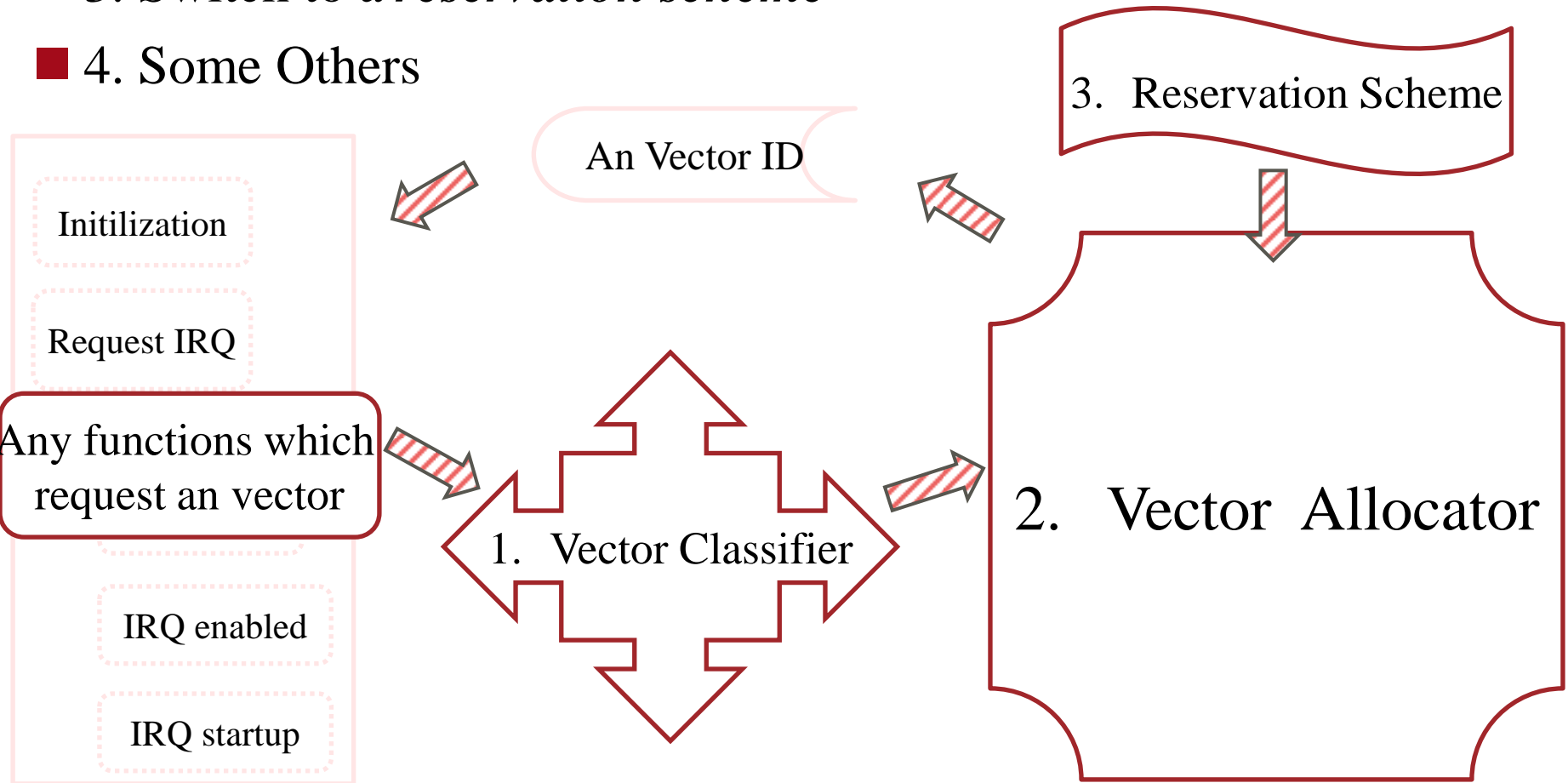
- Future work

- Horrible worst vector management mechanism
 - *Abuse* the interrupt allocation for different type interrupts
 - Serve all different use cases *in one go*
 - Based on *nested loops* to search
 - Cause vector space *exhaustion*
 - Allocate vectors at the *wrong* time and on the *wrong* place
- Some dubious properties, causes *high complexity*
 - Multi CPU affinities for an IRQ
 - Priority level spreading
- Lack of instrumentation
 - All of this is a black box which allows no insight into the actual vector usage



Overhaul of *Vector Allocation*

- 1. Classify the *types* of vectors
- 2. Refactor the *vector allocation mechanism*
- 3. Switch to a *reservation scheme*
- 4. Some Others



- 1. Classify the *types* of vectors
- Each CPU has 256 vectors, But some are *fixed*

- **1. System Vector**

- * *Vectors 0 ... 31*

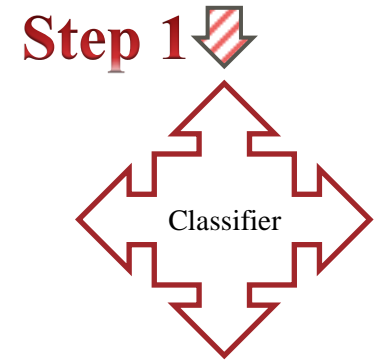
- * *Vector 128*

- * *Vectors INVALIDATE_TLB_VECTOR_START ... 255*

- **2. Legacy Vector**

- * *Vectors 0x30 ... 0x3f*

- **Others** are allocated dynamically for *normal* and *managed* interrupts.



■ 1. Classify the *types* of vectors

- For external interrupts

- Depend on *Interrupt Affinity* (the set of CPUs that can handle this interrupt)

■ 3. *Normal Vector*



■ 4. *Managed Vector*



	<i>Normal Interrupt</i>	<i>Managed Interrupt</i>
At setup time	<ul style="list-style-type: none"> - Affinity may be NULL - A subset of the online CPUs 	<ul style="list-style-type: none"> - Affinity must have been setup - the possible CPUs may be included
User space	<ul style="list-style-type: none"> - Affinity can be modified 	<ul style="list-style-type: none"> - Affinity is fixed
When migration	<ul style="list-style-type: none"> - IRQ can be moved to any online CPUs - Affinity can be even reset 	<ul style="list-style-type: none"> - IRQ can move only in the affinity. - But, can be shutdown and restarted. - Affinity can't be reset

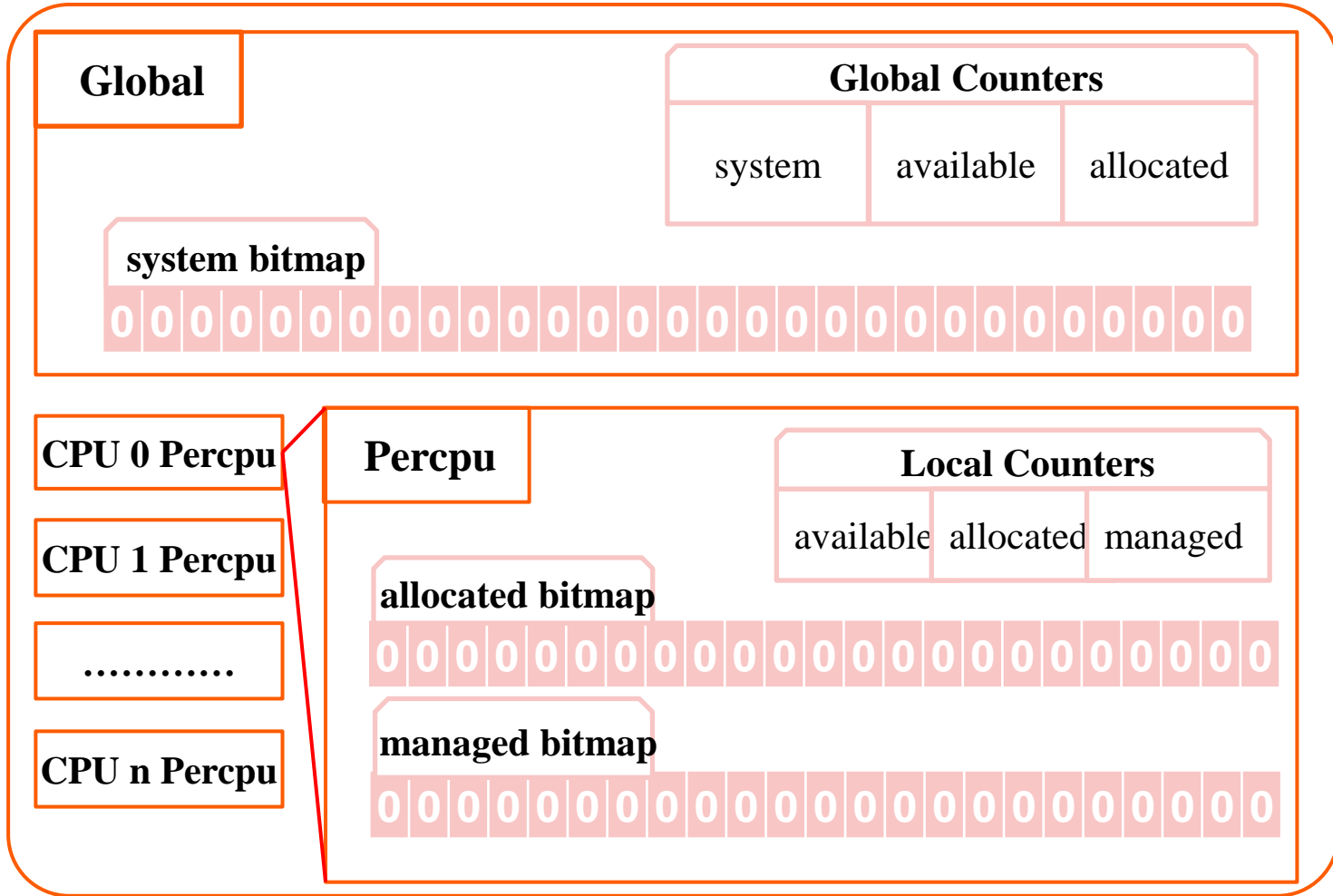
Overhaul of *Vector Allocation*

■ 2. Refactor the *vector allocation mechanism*

Step 2

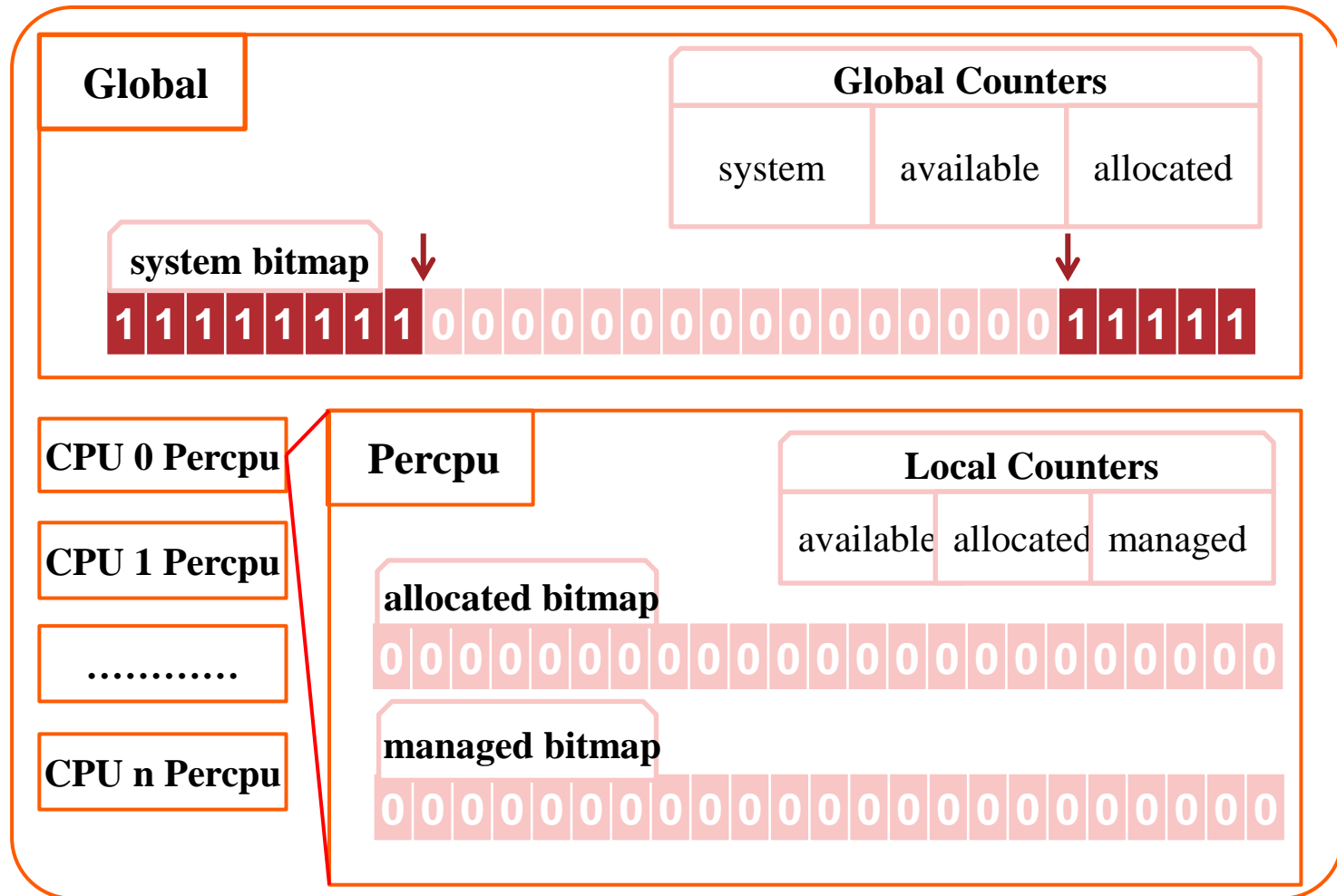
- Create a new bitmap matrix allocator——*IRQ Matrix*

Allocator



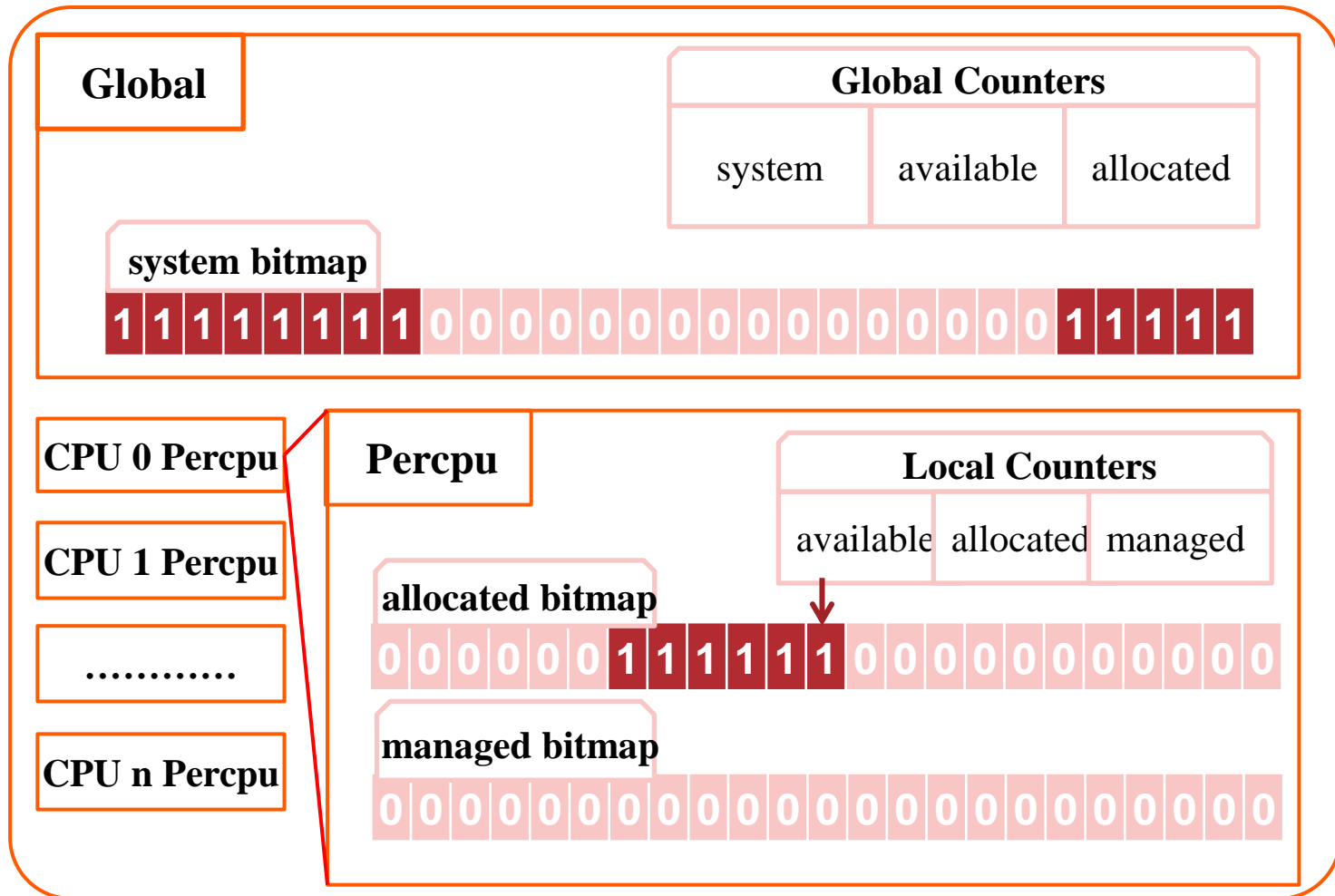
Overhaul of *Vector Allocation*

- 2. Refactor the vector allocation mechanism
 - Use the matrix for *System* vector



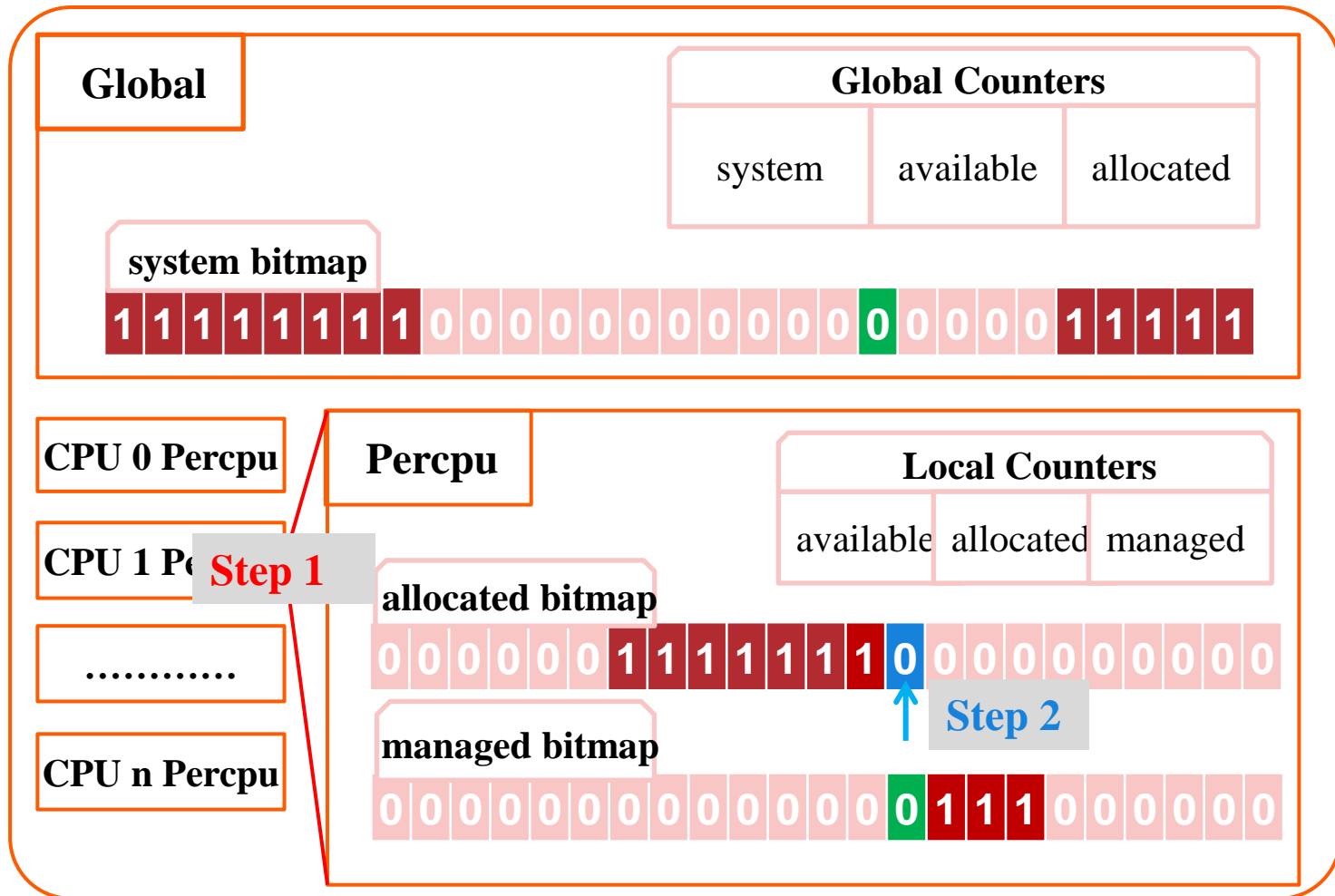
Overhaul of *Vector Allocation*

- 2. Refactor the vector allocation mechanism
 - Use the matrix for *Legacy* vector



Overhaul of *Vector Allocation*

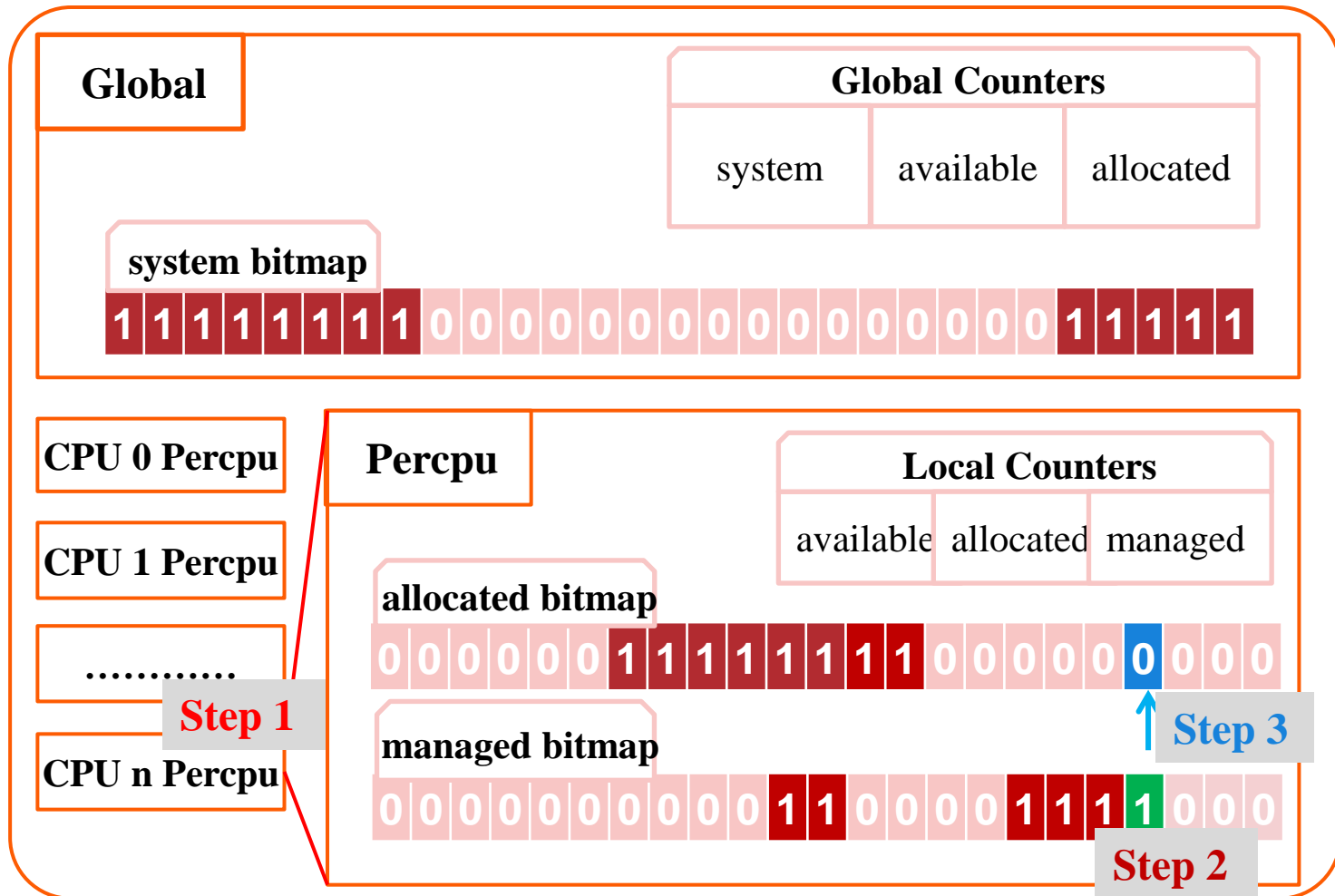
- 2. Refactor the vector allocation mechanism
 - Use the matrix for *Normal* vector



Overhaul of *Vector Allocation*

2. Refactor the vector allocation mechanism

- Use the matrix for *Managed* vector



Overhaul of *Vector Allocation*

■ 3. Switch to *reservation scheme*

- Reserve a *new system vector* , just in case

Step 3 ↓

Reservation

3.1 When the interrupt is allocated and initialized:

Previously



Assign a *real vector* for each interrupts

Now

1. Update the *reservation request counter*

2. Assign *the reserved vector* for each interrupts

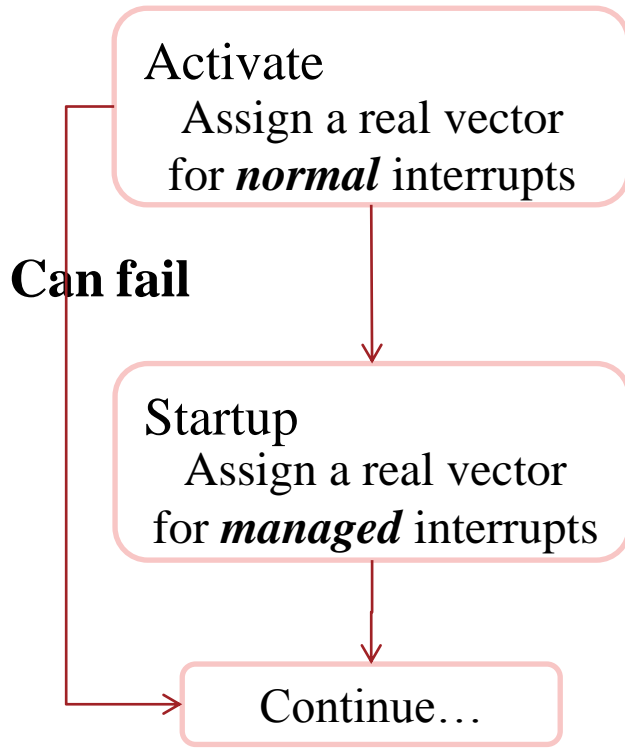
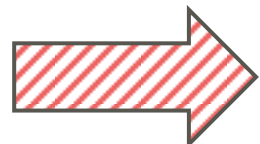
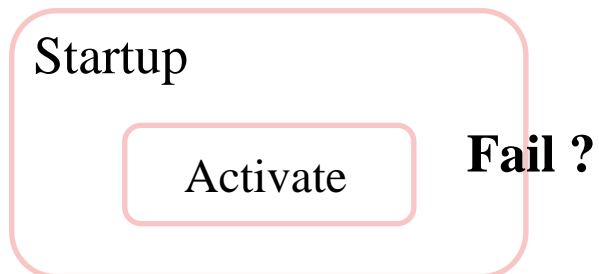
Overhaul of *Vector Allocation*

■ 3. Switch to *reservation scheme*

- *Separate* activation and startup
- Assign the *real* vector



3.2 When the interrupt is *requested*:



Overhaul of *Vector Allocation*

■ Some Others:

- Change from Multi CPU targets to *single interrupt targets*.
- Remove priority level spreading
- Simplify hotplug vector accounting
- Equip with trace points and detailed debugfs information

■ Can see the Vector Allocation by:

- `cat /sys/kernel/debug/irq/irqs/$N`
- `cat /sys/kernel/debug/irq/domains/$N`

```
[root@guest_64bit ~]# cat /sys/kernel/debug/irq/domains/*
name: VECTOR
size: 0
mapped: 25
flags: 0x00000041
Online bitmaps: 2
Global available: 377
Global reserved: 4
Total allocated: 21
System: 43: 0-19,32,50,128,236-255
| CPU | avl | man | act | vectors
| 0 | 190 | 4 | 11 | 34-40,42-44,48
| 1 | 187 | 4 | 10 | 33-37,40-42,45-46
name: IO-APIC-0
size: 24
mapped: 15
flags: 0x00000041
parent: VECTOR
name: VECTOR
size: 0
mapped: 25
flags: 0x00000041
Online bitmaps: 2
Global available: 377
Global reserved: 4
Total allocated: 21
System: 43: 0-19,32,50,128,236-255
| CPU | avl | man | act | vectors
| 0 | 190 | 4 | 11 | 34-40,42-44,48
| 1 | 187 | 4 | 10 | 33-37,40-42,45-46
```

What's next?

- Basics of an interrupt

- Overhaul of interrupt
 - APIC Initialization
 - Vector Allocation

- *Future work*

- Kernel's notion of possible CPU count should be realistic
 - Once the kernel initialized:

Make the possible CPU count realistic

- The vector allocation is a *generic* mechanism
 - Can be used to other architectures

Thank you !