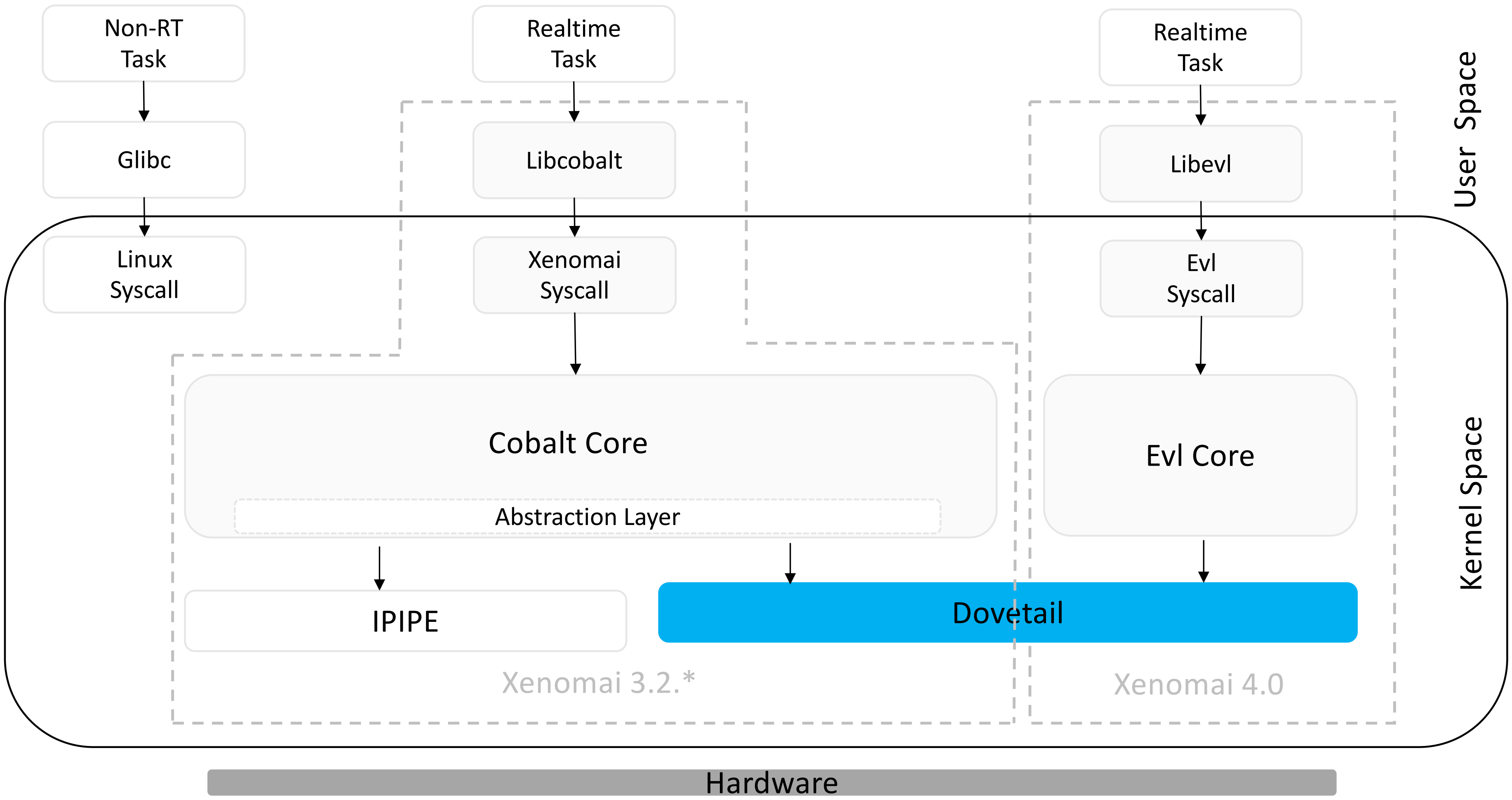
Dovetail Interrupt Pipeline

Agenda

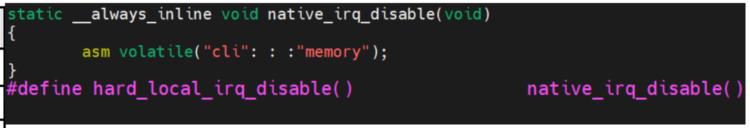
- Dovetail interface -- the successor to the I-IPIPE
- Two-stage IRQ pipeline
- Dovetail Interrupt Flow on X86
- Case Study on SD/MMC CD IRQ Storm
- Cascaded IRQ handling process
- Fixing up the IRQ chip driver
- Summary

Dovetail Interface



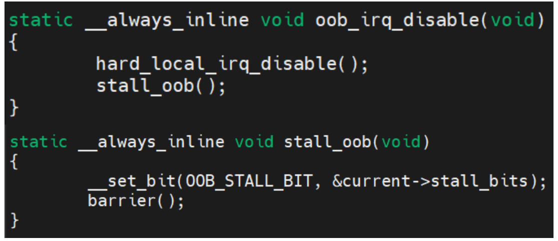
Two-stage IRQ pipeline

Non-virtualized call
flags = hard_local_save_flags()
hard_local_irq_disable()
hard_local_irq_enable()
flags = hard_local_irq_save()
hard_local_irq_restore(flags)
hard_irqs_disabled()
hard_irqs_disabled_flags(flags)

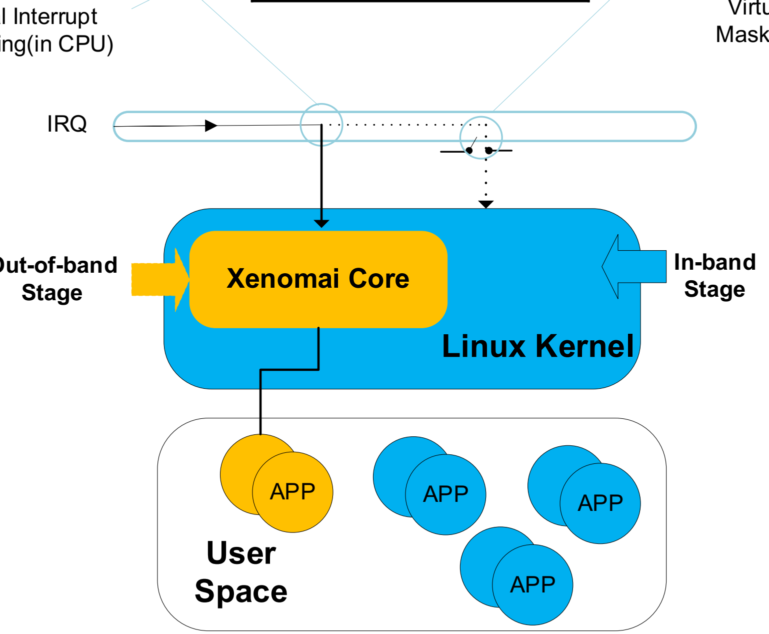


```
static __always_inline void native_irq_disable(void)
{
        asm volatile("cli": : :"memory");
}
#define hard_local_irq_disable()                native_irq_disable()
```

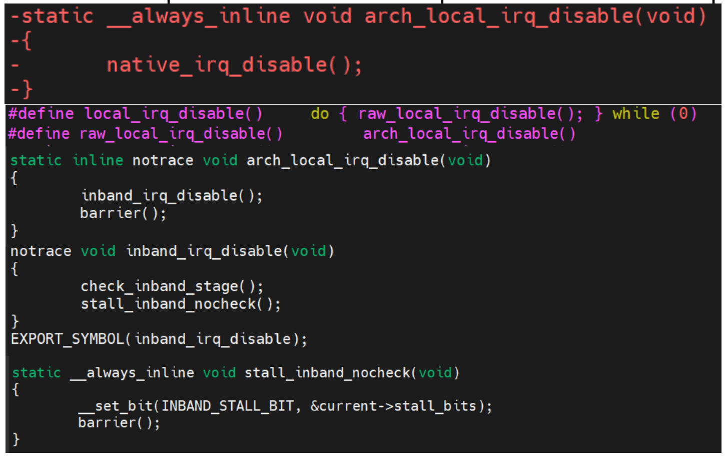
OOB stage operation(Stall/Unstall)
oob_irq_disable()
oob_irq_enable()
flags = oob_irq_save()
oob_irq_restore(flags)
oob_irqs_disabled()



```
static __always_inline void oob_irq_disable(void)
{
        hard_local_irq_disable();
        stall_oob();
}
static __always_inline void stall_oob(void)
{
        __set_bit(OOB_STALL_BIT, &current->stall_bits);
        barrier();
}
```

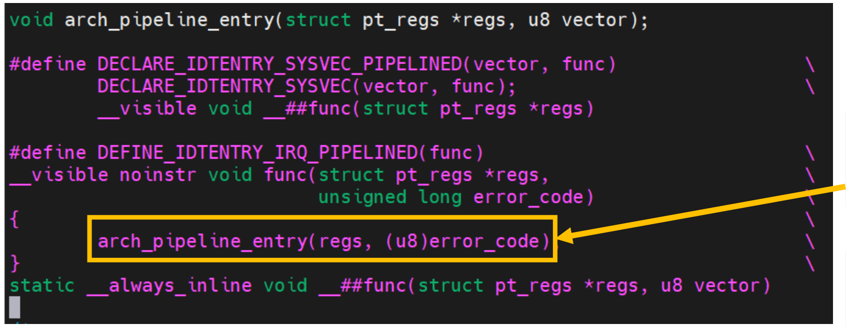


Inband Operations(stall/uninstall)	
Original Linux/Virtual	**Inband stage operation**
local_save_flags(flags)	-none-
local_irq_disable()	Inband_irq_disable()
local_irq_enable()	inband_irq_enable()
local_irq_save(flags)	flags=inband_irq_save()
local_irq_restore(flags)	inband_irq_restore(flags)
irqs_disabled()	inband_irqs_disabled()
irqs_disabled_flags(flags)	-none-

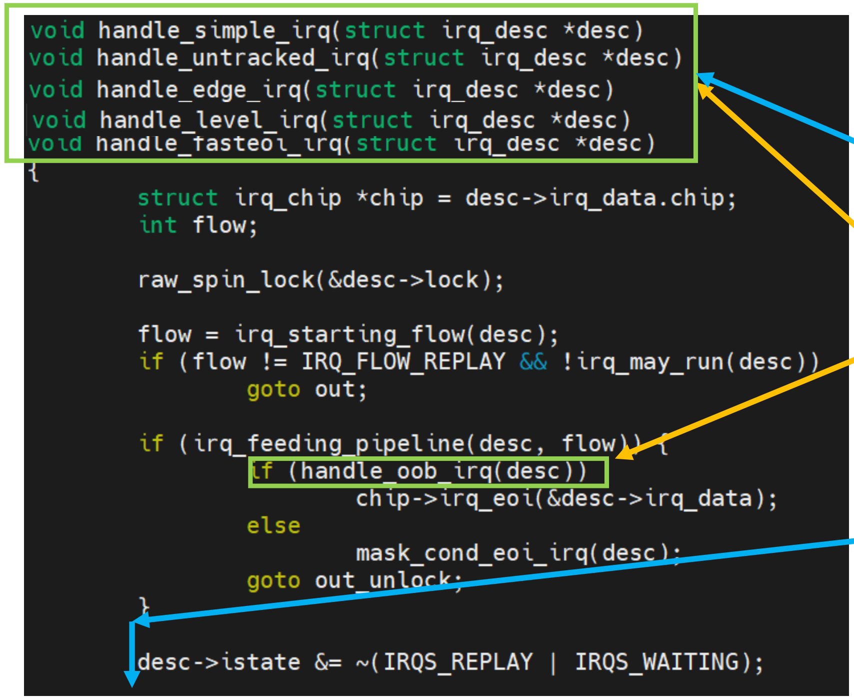


```
-static __always_inline void arch_local_irq_disable(void)
-{
-        native_irq_disable();
-}
#define local_irq_disable()     do { raw_local_irq_disable(); } while (0)
#define raw_local_irq_disable()         arch_local_irq_disable()
static inline notrace void arch_local_irq_disable(void)
{
        inband_irq_disable();
        barrier();
}
notrace void inband_irq_disable(void)
{
        check_inband_stage();
        stall_inband_nocheck();
}
EXPORT_SYMBOL(inband_irq_disable);

static __always_inline void stall_inband_nocheck(void)
{
        __set_bit(INBAND_STALL_BIT, &current->stall_bits);
        barrier();
}
```

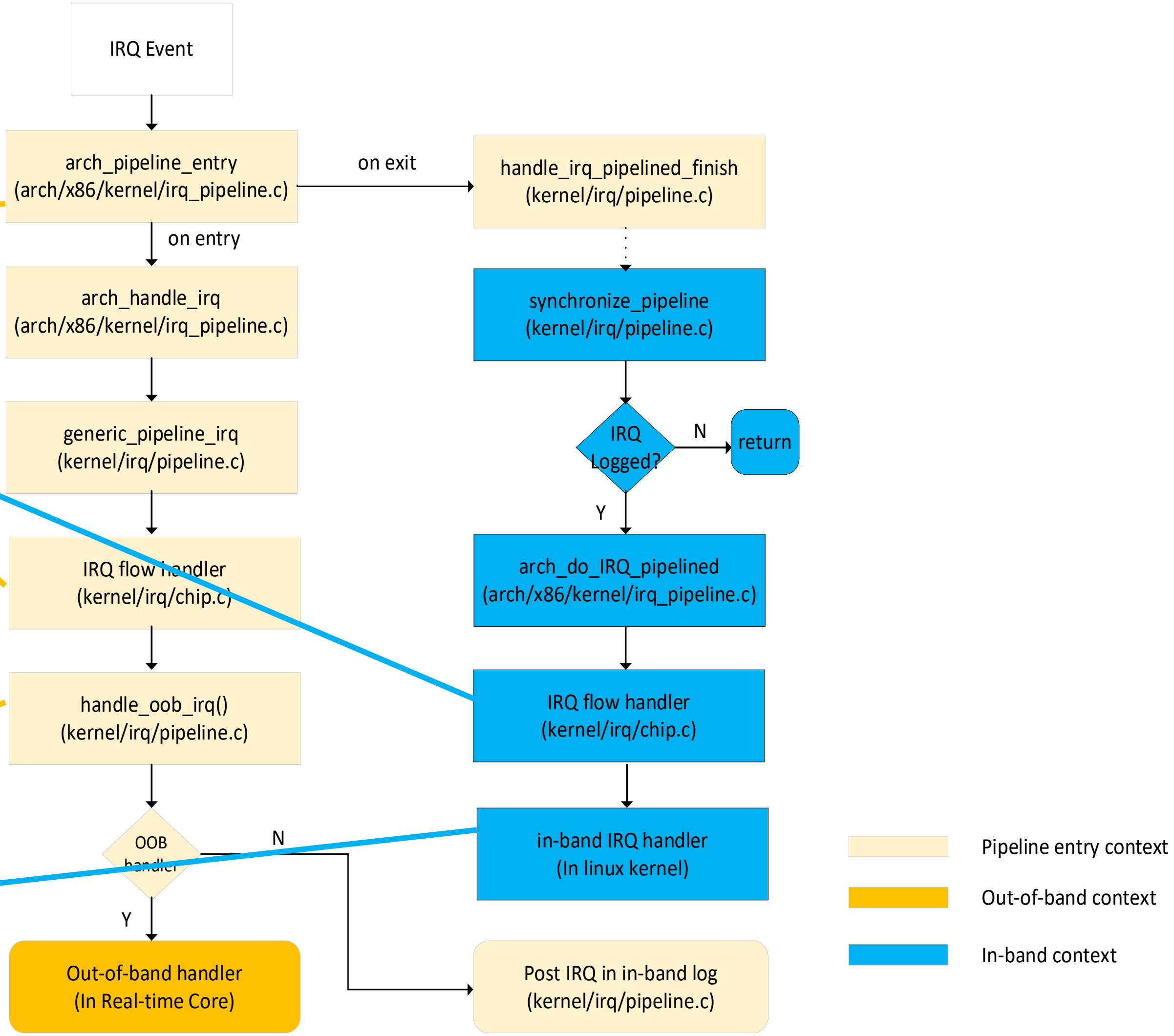
Dovetail Interrupt Flow on X86



```
void arch_pipeline_entry(struct pt_regs *regs, u8 vector);

#define DECLARE_IDTENTRY_SYSVEC_PIPELINED(vector, func)                  \
        DECLARE_IDTENTRY_SYSVEC(vector, func);                           \
        __visible void __##func(struct pt_regs *regs)

#define DEFINE_IDTENTRY_IRQ_PIPELINED(func)                              \
__visible noinstr void func(struct pt_regs *regs,                        \
                            unsigned long error_code)                    \
{                                                                        \
        arch_pipeline_entry(regs, (u8)error_code)                        \
}                                                                        \
static __always_inline void __##func(struct pt_regs *regs, u8 vector)
```



```
void handle_simple_irq(struct irq_desc *desc)
void handle_untracked_irq(struct irq_desc *desc)
void handle_edge_irq(struct irq_desc *desc)
void handle_level_irq(struct irq_desc *desc)
void handle_fasteoi_irq(struct irq_desc *desc)
{
        struct irq_chip *chip = desc->irq_data.chip;
        int flow;

        raw_spin_lock(&desc->lock);

        flow = irq_starting_flow(desc);
        if (flow != IRQ_FLOW_REPLAY && !irq_may_run(desc))
                goto out;

        if (irq_feeding_pipeline(desc, flow)) {
                if (handle_oob_irq(desc))
                        chip->irq_eoi(&desc->irq_data);
                else
                        mask_cond_eoi_irq(desc);
                goto out_unlock;
        }

        desc->istate &= ~(IRQS_REPLAY | IRQS_WAITING);
```



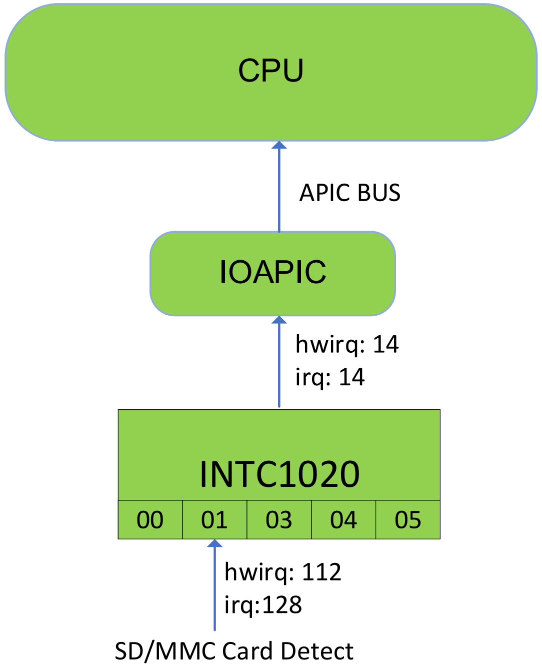
Case Study on SD/MMC CD IRQ Storm

- Xenomai 3.2.2 + Linux 5.10.179:

```
cat /proc/interrupts
           CPU0     CPU1      CPU2    CPU3
  14:   2440190        0         0         0  IR-IO-APIC   14    -fasteoi   INTC1020:00,
INTC1020:01, INTC1020:03, INTC1020:04, INTC1020:05
 128:   2440190        0         0         0  INTC1020:01  112      0000:00:1a.1 cd
```

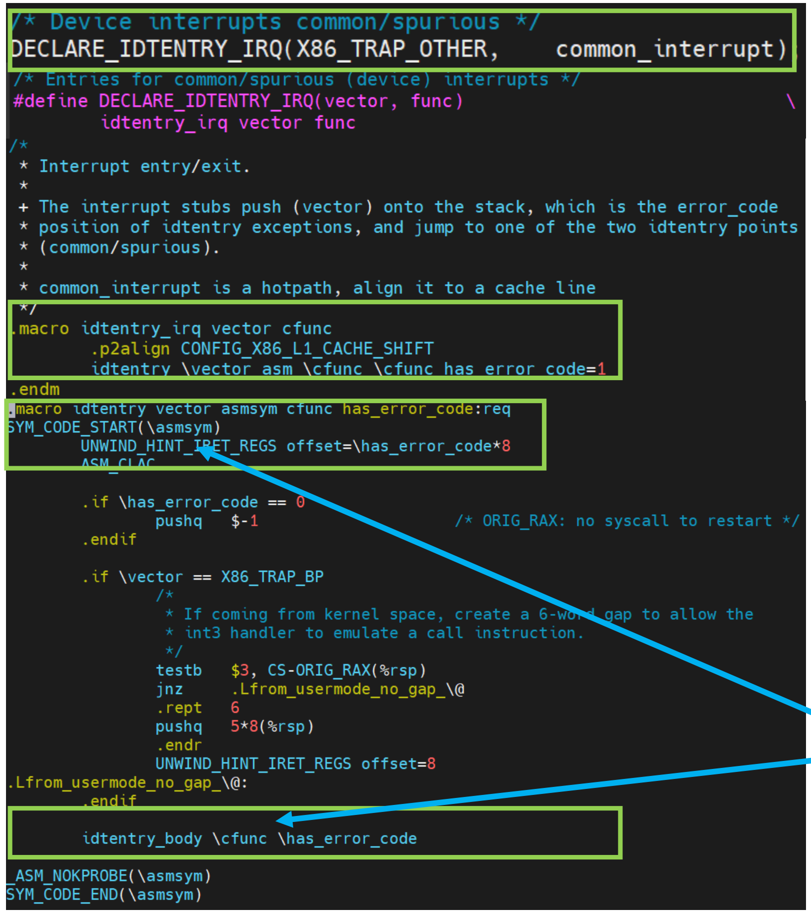
- But Linux 5.10.179: Can not reproduce the issue

```
cat /proc/interrupts
           CPU0     CPU1      CPU2    CPU3
  14:   1         0         0         0  IR-IO-APIC   14    -fasteoi   INTC1020:00, INTC1020:01,
INTC1020:03, INTC1020:04, INTC1020:05
 128:   1         0         0         0  INTC1020:01  112      0000:00:1a.1 cd
```

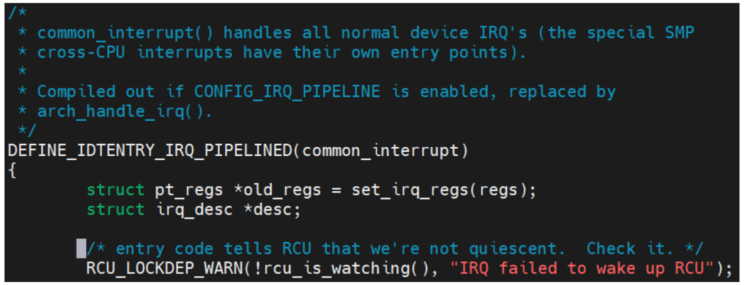


Get stack to further analysis

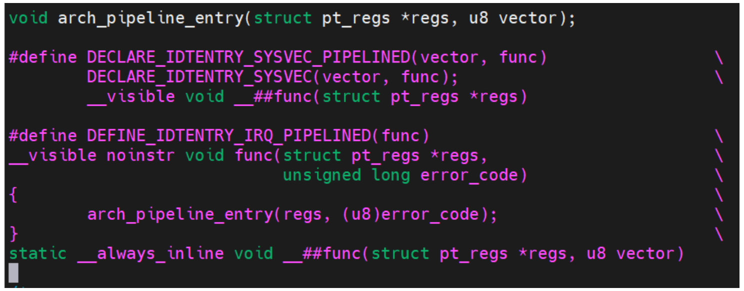
handle_edge_irq
generic_handle_irq
intel_gpio_irq
__handle_irq_event_percpu
handle_irq_event
handle_fasteoi_irq
asm_call_irq_on_stack
</IRQ>
arch_do_IRQ_pipelined
sync_current_irq_stage
handle_irq_pipelined_finish
arch_pipeline_entry
asm_common_interrupt



```
/* Device interrupts common/spurious */
DECLARE_IDTENTRY_IRQ(X86_TRAP_OTHER,	common_interrupt)
/* Entries for common/spurious (device) interrupts */
#define DECLARE_IDTENTRY_IRQ(vector, func)				\
	idtentry_irq vector func
/*
 * Interrupt entry/exit.
 *
 + The interrupt stubs push (vector) onto the stack, which is the error_code
 * position of idtentry exceptions, and jump to one of the two idtentry points
 * (common/spurious).
 *
 * common_interrupt is a hotpath, align it to a cache line
 */
.macro idtentry_irq vector cfunc
	.p2align CONFIG_X86_L1_CACHE_SHIFT
	idtentry \vector asm_\cfunc \cfunc has_error_code=1
.endm
.macro idtentry vector asmsym cfunc has_error_code:req
SYM_CODE_START(\asmsym)
	UNWIND_HINT_IRET_REGS offset=\has_error_code*8
	ASM_CLAC

	.if \has_error_code == 0
		pushq	$-1			/* ORIG_RAX: no syscall to restart */
	.endif

	.if \vector == X86_TRAP_BP
		/*
		 * If coming from kernel space, create a 6-word gap to allow the
		 * int3 handler to emulate a call instruction.
		 */
		testb	$3, CS-ORIG_RAX(%rsp)
		jnz	.Lfrom_usermode_no_gap_\@
		.rept	6
		pushq	5*8(%rsp)
		.endr
		UNWIND_HINT_IRET_REGS offset=8
.Lfrom_usermode_no_gap_\@:
	.endif

	idtentry_body \cfunc \has_error_code

_ASM_NOKPROBE(\asmsym)
SYM_CODE_END(\asmsym)
```



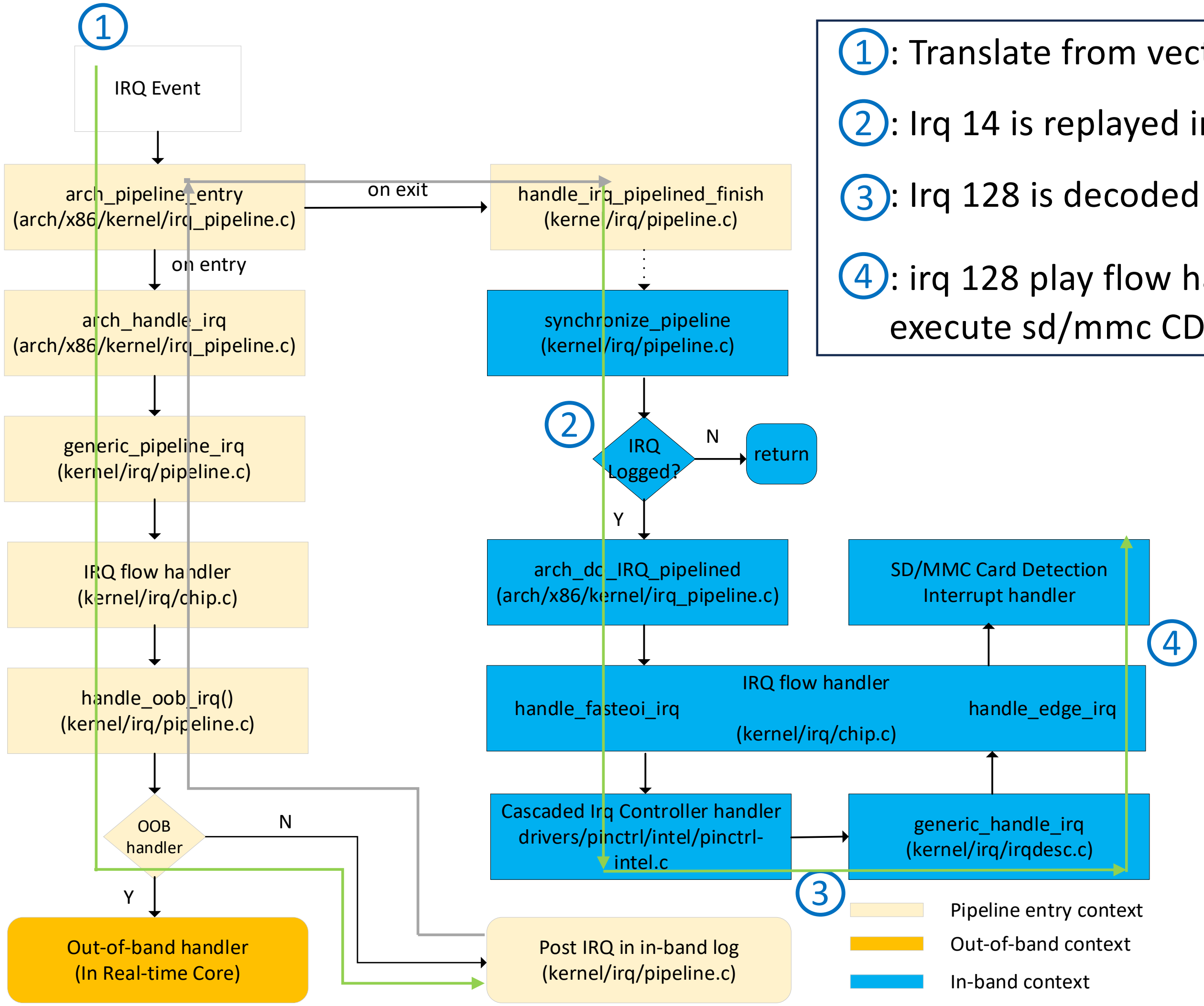
```
/*
 * common_interrupt() handles all normal device IRQ's (the special SMP
 * cross-CPU interrupts have their own entry points).
 *
 * Compiled out if CONFIG_IRQ_PIPELINE is enabled, replaced by
 * arch_handle_irq().
 */
DEFINE_IDTENTRY_IRQ_PIPELINED(common_interrupt)
{
	struct pt_regs *old_regs = set_irq_regs(regs);
	struct irq_desc *desc;

	/* entry code tells RCU that we're not quiescent.  Check it. */
	RCU_LOCKDEP_WARN(!rcu_is_watching(), "IRQ failed to wake up RCU");
```



```
void arch_pipeline_entry(struct pt_regs *regs, u8 vector);

#define DECLARE_IDTENTRY_SYSVEC_PIPELINED(vector, func)			\
	DECLARE_IDTENTRY_SYSVEC(vector, func);				\
	__visible void __##func(struct pt_regs *regs)

#define DEFINE_IDTENTRY_IRQ_PIPELINED(func)				\
__visible noinstr void func(struct pt_regs *regs,			\
			    unsigned long error_code)			\
{									\
	arch_pipeline_entry(regs, (u8)error_code);			\
}									\
static __always_inline void __##func(struct pt_regs *regs, u8 vector)
```

1. Define asm_common_interrupt
2. Call common_interrupt

Cascaded IRQ handling process



①

IRQ Event

arch_pipeline_entry
(arch/x86/kernel/irq_pipeline.c)

on exit

handle_irq_pipelined_finish
(kernel/irq/pipeline.c)

on entry

arch_handle_irq
(arch/x86/kernel/irq_pipeline.c)

synchronize_pipeline
(kernel/irq/pipeline.c)

②

IRQ Logged?

N

return

Y

generic_pipeline_irq
(kernel/irq/pipeline.c)

IRQ flow handler
(kernel/irq/chip.c)

arch_do_IRQ_pipelined
(arch/x86/kernel/irq_pipeline.c)

SD/MMC Card Detection
Interrupt handler

④

handle_oob_irq()
(kernel/irq/pipeline.c)

IRQ flow handler

handle_fasteoi_irq

handle_edge_irq

(kernel/irq/chip.c)

OOB handler

N

Y

Cascaded Irq Controller handler
drivers/pinctrl/intel/pinctrl-intel.c

generic_handle_irq
(kernel/irq/irqdesc.c)

③

Out-of-band handler
(In Real-time Core)

Post IRQ in in-band log
(kernel/irq/pipeline.c)

Pipeline entry context

Out-of-band context

In-band context

①: Translate from vector to irq num 14 & first play then posted

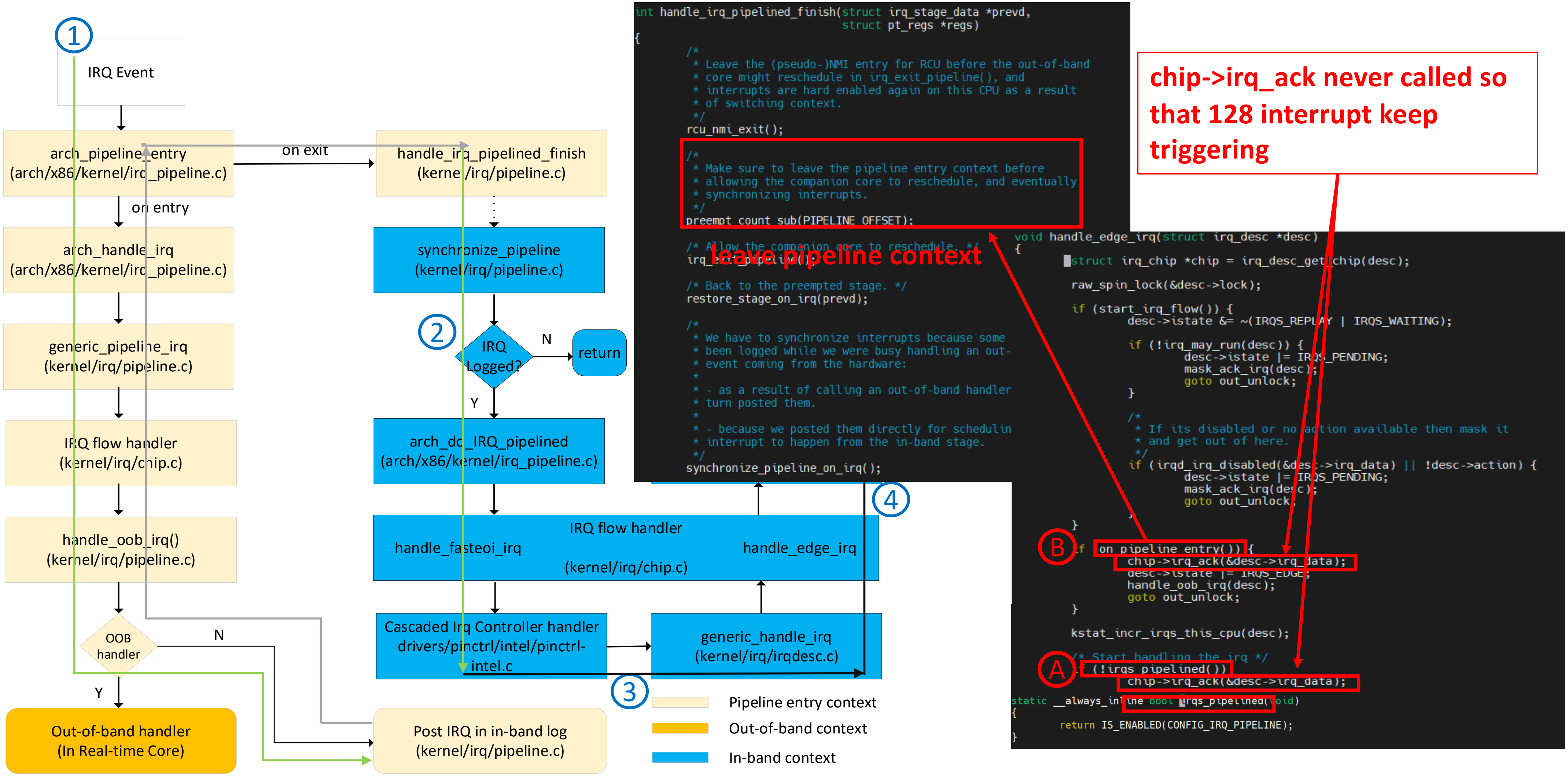
②: Irq 14 is replayed in in-band

③: Irq 128 is decoded after INTC1020 irq handler is executed

④: irq 128 play flow handler handle_edge_irq and then execute sd/mmc CD interrupt handler

handle_edge_irq	④
generic_handle_irq	
intel_gpio_irq	③
__handle_irq_event_percpu	
handle_irq_event	
handle_fasteoi_irq	②
asm_call_irq_on_stack	
</IRQ>	
arch_do_IRQ_pipelined	
sync_current_irq_stage	②
handle_irq_pipelined_finish	
arch_pipeline_entry	①
asm_common_interrupt	

Find the root cause of irq storm



1

IRQ Event

arch_pipeline_entry (arch/x86/kernel/irq_pipeline.c)

on entry

arch_handle_irq (arch/x86/kernel/irq_pipeline.c)

generic_pipeline_irq (kernel/irq/pipeline.c)

IRQ flow handler (kernel/irq/chip.c)

handle_oob_irq() (kernel/irq/pipeline.c)

OOB handler

Y

N

Out-of-band handler (In Real-time Core)

on exit

handle_irq_pipelined_finish (kernel/irq/pipeline.c)

synchronize_pipeline (kernel/irq/pipeline.c)

2

IRQ Logged?

N

return

Y

arch_do_IRQ_pipelined (arch/x86/kernel/irq_pipeline.c)

IRQ flow handler

handle_fasteoi_irq

handle_edge_irq

(kernel/irq/chip.c)

Cascaded Irq Controller handler drivers/pinctrl/intel/pinctrl-intel.c

generic_handle_irq (kernel/irq/irqdesc.c)

3

4

Post IRQ in in-band log (kernel/irq/pipeline.c)

Pipeline entry context

Out-of-band context

In-band context

```
int handle_irq_pipelined_finish(struct irq_stage_data *prevd,
                                struct pt_regs *regs)
{
        /*
         * Leave the (pseudo-)NMI entry for RCU before the out-of-band
         * core might reschedule in irq_exit_pipeline(), and
         * interrupts are hard enabled again on this CPU as a result
         * of switching context.
         */
        rcu_nmi_exit();

        /*
         * Make sure to leave the pipeline entry context before
         * allowing the companion core to reschedule, and eventually
         * synchronizing interrupts.
         */
        preempt_count_sub(PIPELINE_OFFSET);

        /* Allow the companion core to reschedule. */
        irq_exit_pipeline();

        /* Back to the preempted stage. */
        restore_stage_on_irq(prevd);

        /*
         * We have to synchronize interrupts because some
         * been logged while we were busy handling an out-
         * event coming from the hardware:
         *
         * - as a result of calling an out-of-band handler
         * turn posted them.
         *
         * - because we posted them directly for schedulin
         * interrupt to happen from the in-band stage.
         */
        synchronize_pipeline_on_irq();
```

leave pipeline context

chip->irq_ack never called so that 128 interrupt keep triggering

```
void handle_edge_irq(struct irq_desc *desc)
{
        struct irq_chip *chip = irq_desc_get_chip(desc);

        raw_spin_lock(&desc->lock);

        if (start_irq_flow()) {
                desc->istate &= ~(IRQS_REPLAY | IRQS_WAITING);

                if (!irq_may_run(desc)) {
                        desc->istate |= IRQS_PENDING;
                        mask_ack_irq(desc);
                        goto out_unlock;
                }

                /*
                 * If its disabled or no action available then mask it
                 * and get out of here.
                 */
                if (irqd_irq_disabled(&desc->irq_data) || !desc->action) {
                        desc->istate |= IRQS_PENDING;
                        mask_ack_irq(desc);
                        goto out_unlock;
                }
        }
        if (on_pipeline_entry()) {
                chip->irq_ack(&desc->irq_data);
                desc->istate |= IRQS_EDGE;
                handle_oob_irq(desc);
                goto out_unlock;
        }

        kstat_incr_irqs_this_cpu(desc);

        /* Start handling the irq */
        if (!irqs_pipelined())
                chip->irq_ack(&desc->irq_data);
static __always_inline bool irqs_pipelined(void)
{
        return IS_ENABLED(CONFIG_IRQ_PIPELINE);
}
```

B

A

IRQ handling Process after fixed

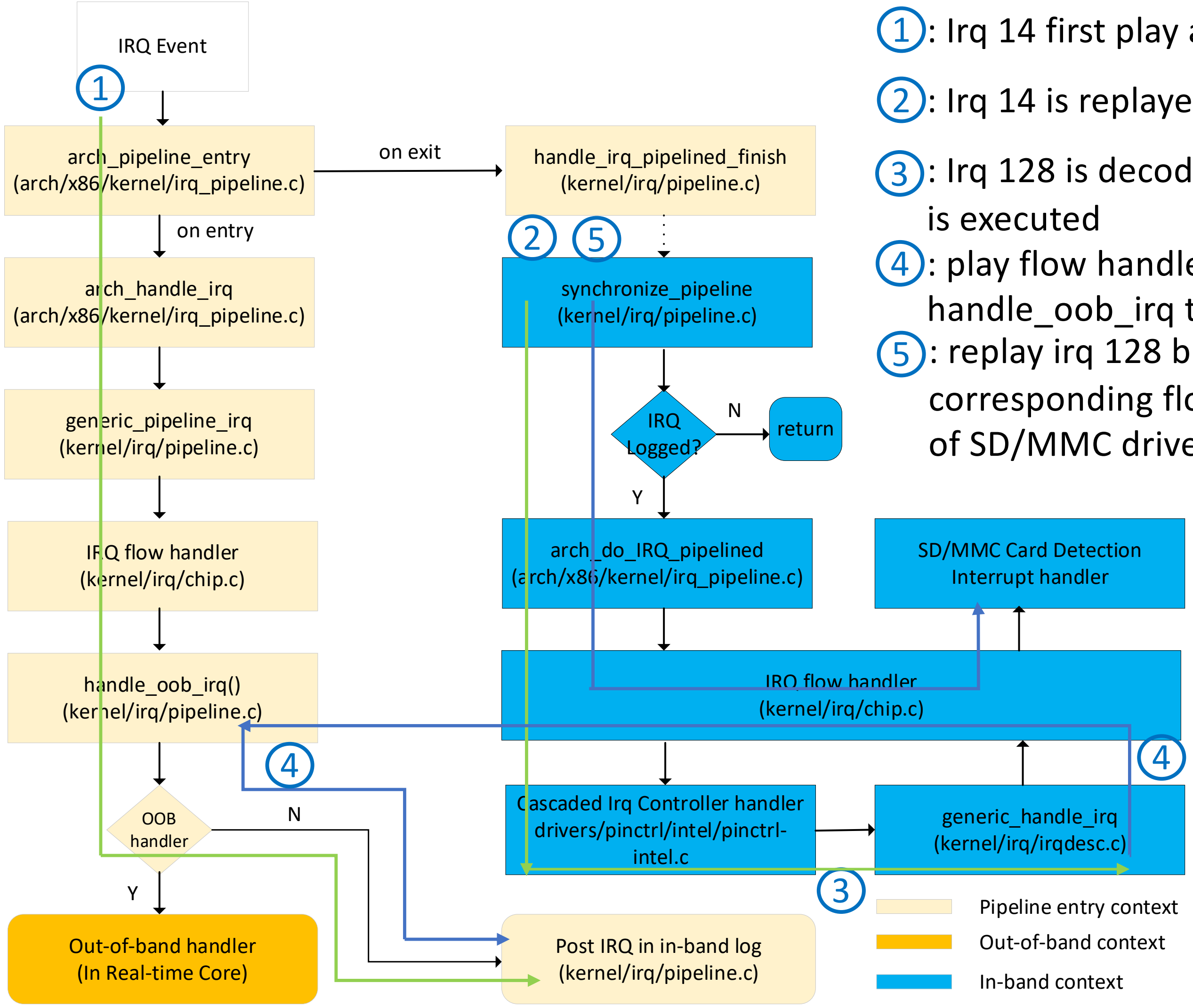
①: Irq 14 first play and posted

②: Irq 14 is replayed in in-band

③: Irq 128 is decoded after irq handler of INTC1020 driver is executed

④: play flow handler to ack irq 128 and execute handle_oob_irq to post the irq inband

⑤: replay irq 128 because it is logged and execute corresponding flow handler and interrupt handler of CD of SD/MMC driver



IRQ Event

arch_pipeline_entry (arch/x86/kernel/irq_pipeline.c)

on exit

handle_irq_pipelined_finish (kernel/irq/pipeline.c)

on entry

arch_handle_irq (arch/x86/kernel/irq_pipeline.c)

synchronize_pipeline (kernel/irq/pipeline.c)

generic_pipeline_irq (kernel/irq/pipeline.c)

IRQ Logged?

N

return

Y

IRQ flow handler (kernel/irq/chip.c)

arch_do_IRQ_pipelined (arch/x86/kernel/irq_pipeline.c)

SD/MMC Card Detection Interrupt handler

handle_oob_irq() (kernel/irq/pipeline.c)

IRQ flow handler (kernel/irq/chip.c)

OOB handler

N

Cascaded Irq Controller handler drivers/pinctrl/intel/pinctrl-intel.c

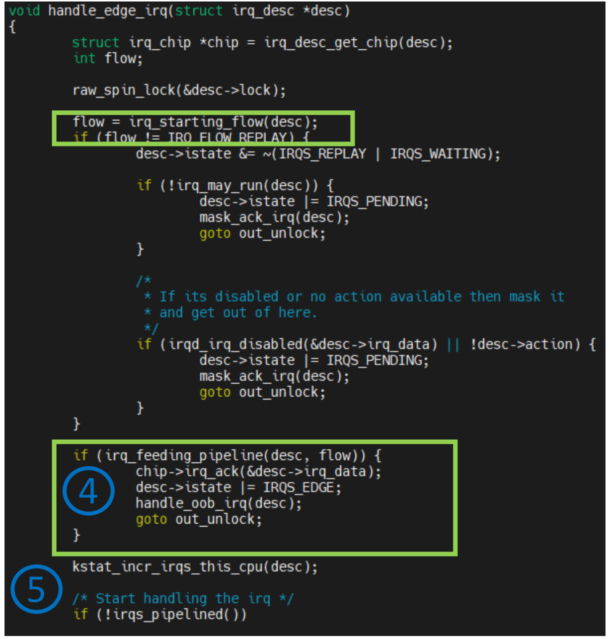
generic_handle_irq (kernel/irq/irqdesc.c)

Y

Out-of-band handler (In Real-time Core)

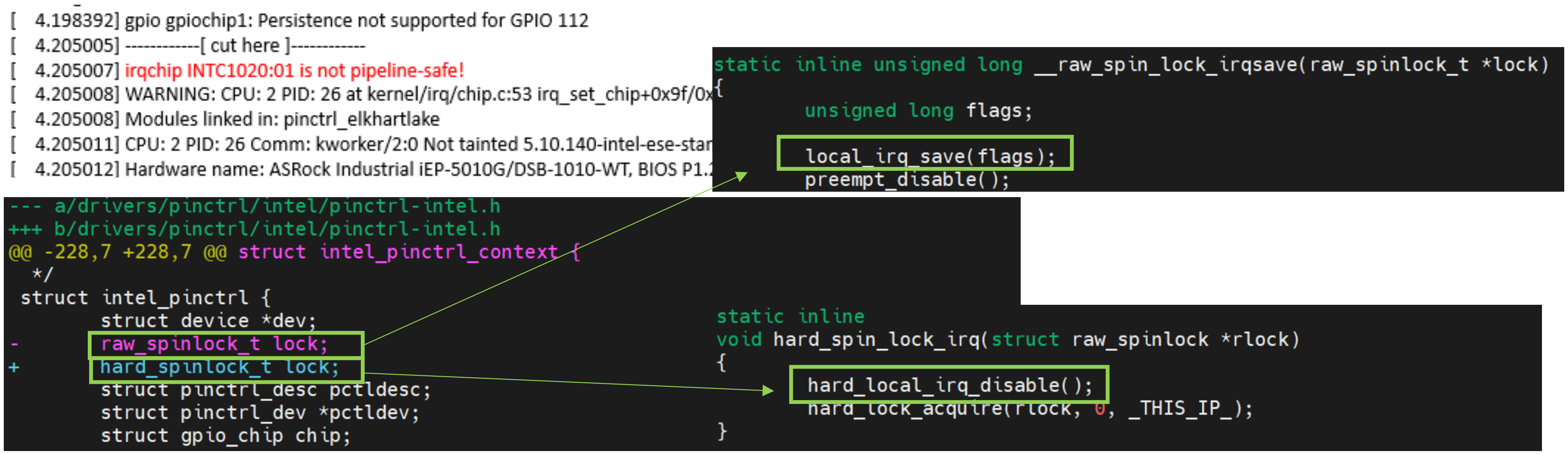
Post IRQ in in-band log (kernel/irq/pipeline.c)

▭	Pipeline entry context
▬	Out-of-band context
▬	In-band context



```
void handle_edge_irq(struct irq_desc *desc)
{
        struct irq_chip *chip = irq_desc_get_chip(desc);
        int flow;

        raw_spin_lock(&desc->lock);

        flow = irq_starting_flow(desc);
        if (flow != IRQ_FLOW_REPLAY) {
                desc->istate &= ~(IRQS_REPLAY | IRQS_WAITING);

                if (!irq_may_run(desc)) {
                        desc->istate |= IRQS_PENDING;
                        mask_ack_irq(desc);
                        goto out_unlock;
                }

                /*
                 * If its disabled or no action available then mask it
                 * and get out of here.
                 */
                if (irqd_irq_disabled(&desc->irq_data) || !desc->action) {
                        desc->istate |= IRQS_PENDING;
                        mask_ack_irq(desc);
                        goto out_unlock;
                }
        }

        if (irq_feeding_pipeline(desc, flow)) {
                chip->irq_ack(&desc->irq_data);
                desc->istate |= IRQS_EDGE;
                handle_oob_irq(desc);
                goto out_unlock;
        }
        kstat_incr_irqs_this_cpu(desc);

        /* Start handling the irq */
        if (!irqs_pipelined())
```

Fixing up the IRQ chip driver

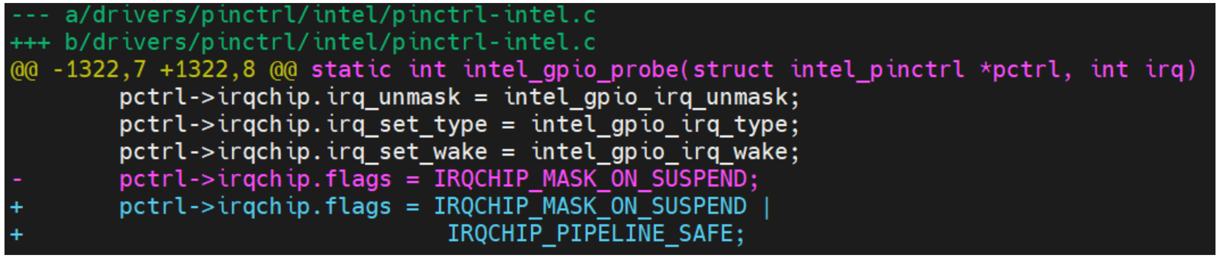


```
[    4.198392] gpio gpiochip1: Persistence not supported for GPIO 112
[    4.205005] ------------[ cut here ]------------
[    4.205007] irqchip INTC1020:01 is not pipeline-safe!
[    4.205008] WARNING: CPU: 2 PID: 26 at kernel/irq/chip.c:53 irq_set_chip+0x9f/0x
[    4.205008] Modules linked in: pinctrl_elkhartlake
[    4.205011] CPU: 2 PID: 26 Comm: kworker/2:0 Not tainted 5.10.140-intel-ese-star
[    4.205012] Hardware name: ASRock Industrial iEP-5010G/DSB-1010-WT, BIOS P1.2
```

```
static inline unsigned long __raw_spin_lock_irqsave(raw_spinlock_t *lock)
{
        unsigned long flags;

        local_irq_save(flags);
        preempt_disable();
```

```
--- a/drivers/pinctrl/intel/pinctrl-intel.h
+++ b/drivers/pinctrl/intel/pinctrl-intel.h
@@ -228,7 +228,7 @@ struct intel_pinctrl_context {
  */
 struct intel_pinctrl {
        struct device *dev;
-       raw_spinlock_t lock;
+       hard_spinlock_t lock;
        struct pinctrl_desc pctldesc;
        struct pinctrl_dev *pctldev;
        struct gpio_chip chip;
```

```
static inline
void hard_spin_lock_irq(struct raw_spinlock *rlock)
{
        hard_local_irq_disable();
        hard_lock_acquire(rlock, 0, _THIS_IP_);

}
```



```
--- a/drivers/pinctrl/intel/pinctrl-intel.c
+++ b/drivers/pinctrl/intel/pinctrl-intel.c
@@ -1322,7 +1322,8 @@ static int intel_gpio_probe(struct intel_pinctrl *pctrl, int irq)
        pctrl->irqchip.irq_unmask = intel_gpio_irq_unmask;
        pctrl->irqchip.irq_set_type = intel_gpio_irq_type;
        pctrl->irqchip.irq_set_wake = intel_gpio_irq_wake;
-       pctrl->irqchip.flags = IRQCHIP_MASK_ON_SUSPEND;
+       pctrl->irqchip.flags = IRQCHIP_MASK_ON_SUSPEND |
+                               IRQCHIP_PIPELINE_SAFE;
```

Thank you