SMP Implementation for OpenBSD/sgi

Takuya ASADA<syuu@openbsd.org>

Introduction

- I was working to add SMP & 64bit support to a BSD-based embedded OS at 
  The target device was MIPS64
- There's no complete *BSD/MIPS SMP implementation at that time, so I implemented it
- The implementation was proprietary, but I wanted to contribute something to BSD community
- I decided to implement SMP from scratch, tried to find a suitable MIPS SMP machine

Finding MIPS/SMP Machine(1)

- Broadcom SiByte BCM1250 looks nice - 2core 1GHz MIPS64, DDR DRAM, GbE, PCI, HT
- Cisco 3845 integrated this processor
- How much is it?

Finding MIPS/SMP Machine(1)

- Broadcom SiByte BCM1250 looks nice - 2core 1GHz MIPS64, DDR DRAM, GbE, PCI, HT
- Cisco 3845 integrated this processor
- How much is it?

from $14,200!
Totally unacceptable!!!

Finding MIPS/SMP Machine(2)

- Some antique SGI machines are available on eBay
- These are basically very cheap
- I realized SGI Octane 2 has 2 cores, already supported by OpenBSD

Finding MIPS/SMP Machine(2)

- Some antique SGI machines are available on eBay
- These are basically very cheap
- I realized SGI Octane 2 has 2 cores, already supported by OpenBSD

Just $33!

Do you remember how much was it?

This is my Octane2

Processors	MIPS R12000 400MHz x2
Memory	1GB SDRAM
Graphics	3D Graphics Card
Sound	Integrated Digital Audio
Storage	35GB SCSI HDD
Ethernet	100BASE-T


030-1467-001 REV D


SO 9832


NEC JAPAN
D30700LRS-250
VR10000
9812K9021 ES3.4
2A
SARONIX 9820K
100.0000 MHz
S1149BAC
SiliconGraphics
Computer Systems
NEC JAPAN
D30700LRS-250
VR10000
9812K9021 ES3.4
2A


NEC JAPAN
D30700LRS-250
VR10000
9812K9021 ES3.4
2A

Become an OpenBSD developer

- I started working on Octane2 since Apr 2009, wrote about on my blog
- Miod Vallat discovered it, suggested my code be merged into OpenBSD main tree
- I become an OpenBSD developer in Sep 2009, started merging the code, joined hackathon, worked with Miod and Joel Sing

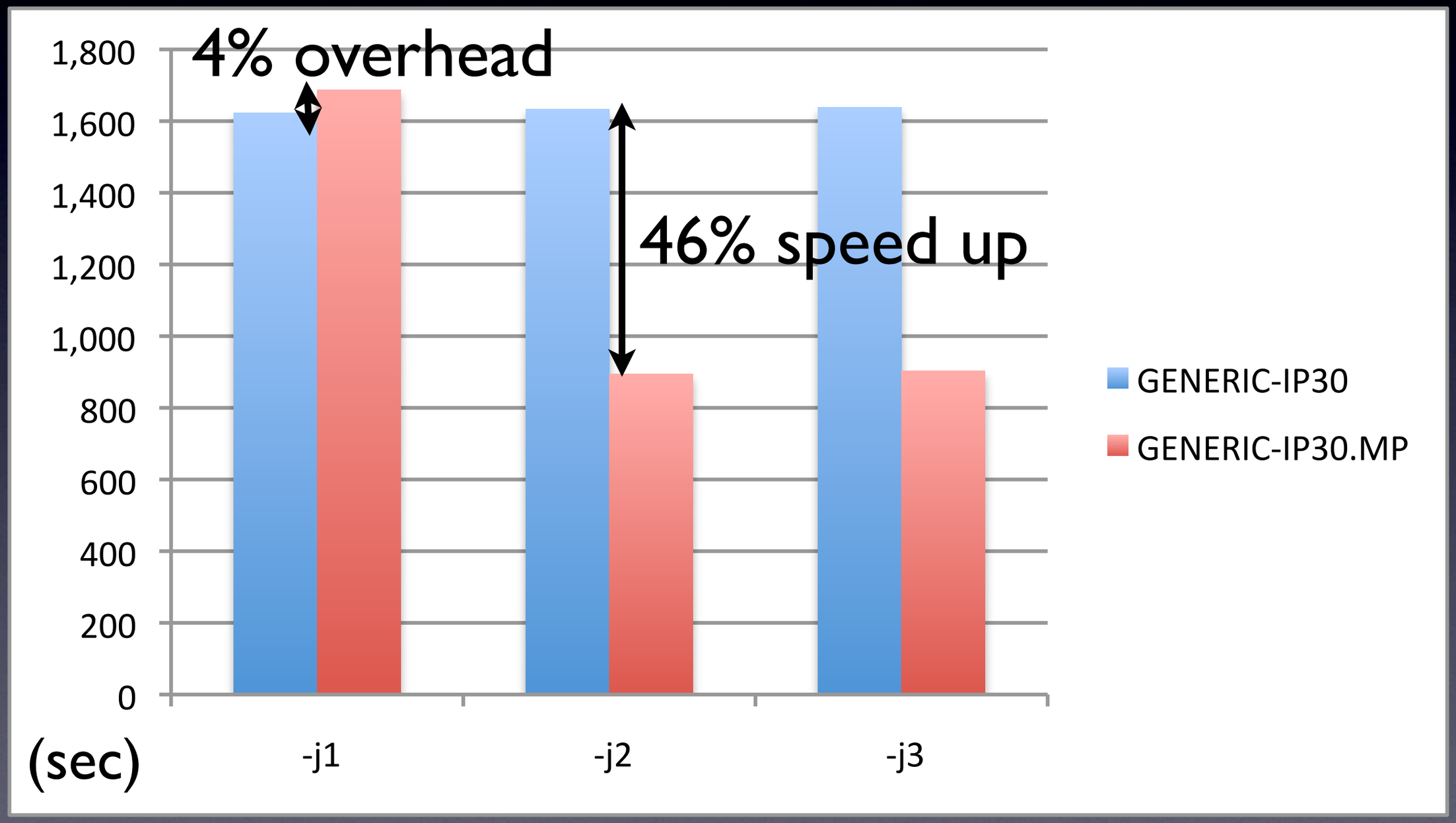
NOW IT WORKS!!!!

- Merged into OpenBSD main tree
- You can try it now!
- It seems stable -
  I tried “make build” again and again over a day, it keeps working

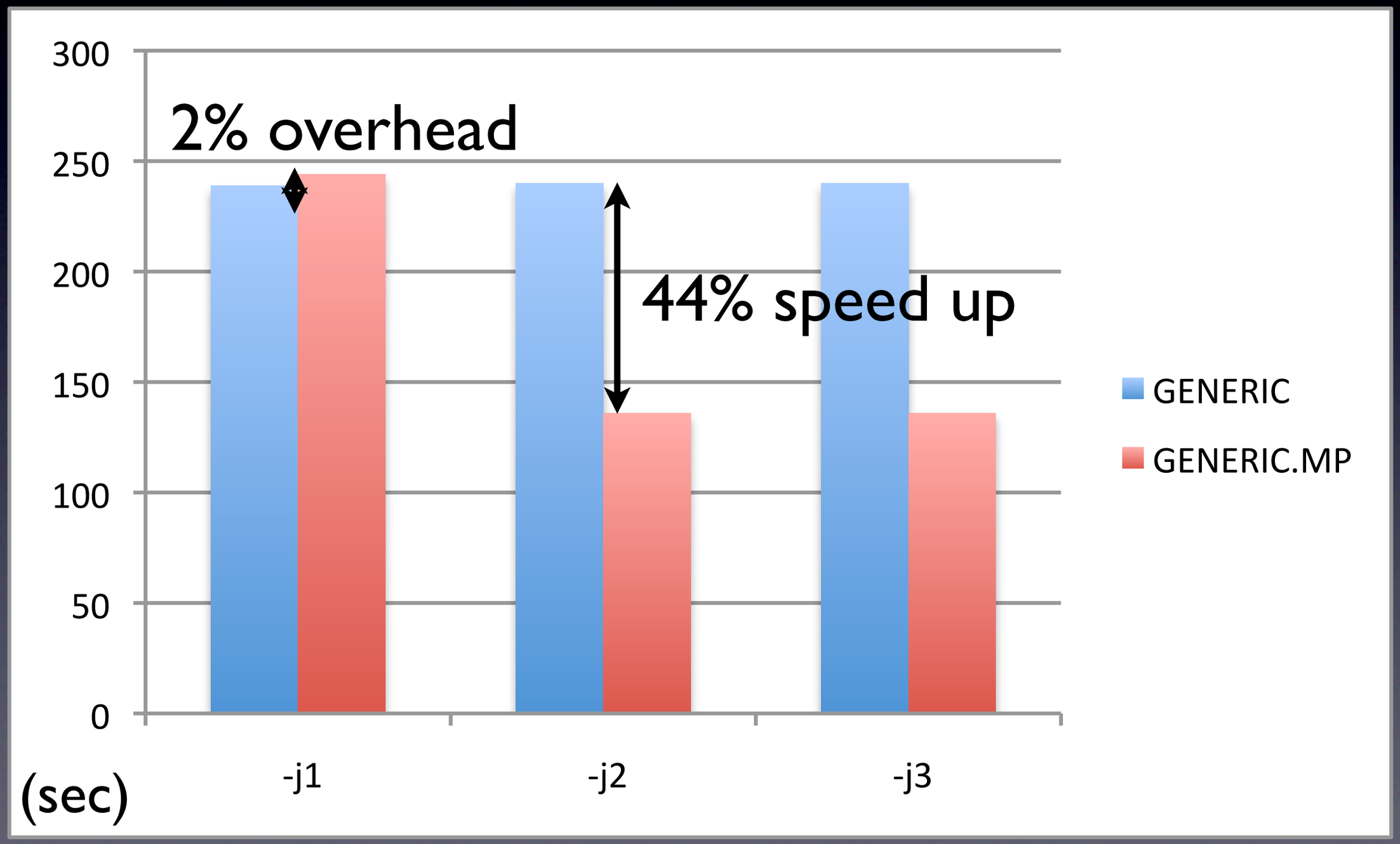
Demo

Open Youtube movie by click “Demo”

Performance - kernel build benchmark



Performance - OpenBSD/amd64 for comparison



What did we need to do for the SMP work

There were lots of works...

- Support multiple cpu_info and processor related macros
- Move per-processor data into cpu_info
- Implement lock primitives
- Acquiring giant lock
- Implement atomic operations
- Spin up secondary processors
- Secondary processor entry point
- IPI: Inter-Processor Interrupt
- Per-processor ASID management
- TLB shootdown
- Lazy FPU handling
- Per-processor clock

Describe it more simply

We can classify tasks into three kind of problems:

- Restructuring per-processor informations
- Implement and use lock/atomic primitives
- Hardware related problems

Restructuring per-processor informations

- In the original sgi port, some informations which related the processor are allocated only one
- Simple case:
  Information is stored into global variable
  Move it into “cpu_info”, per-processor structure
- Complex case:
  In pmap, we need to maintain some information per-processor * per-process

Simple case - clock.c: original code

```
// defined as global variables
        u_int32_t cpu_counter_last;
        u_int32_t cpu_counter_interval;
        u_int32_t pendingticks;

uint32_t clock_int5(uint32_t mask, struct trap_frame *tf)
...
        clkdiff = cp0_get_count() - cpu_counter_last;
        while (clkdiff >= cpu_counter_interval) {
                cpu_counter_last += cpu_counter_interval;
                clkdiff = cp0_get_count() - cpu_counter_last;
                pendingticks++;
```

Simple case - clock.c: modified code

```
uint32_t clock_int5(uint32_t mask, struct trap_frame *tf)
...
        struct cpu_info *ci = curcpu();
        clkdiff = cp0_get_count() - ci->ci_cpu_counter_last;
        while (clkdiff >= ci->ci_cpu_counter_interval) {
                ci->ci_cpu_counter_last +=
                    ci->ci_cpu_counter_interval;
                clkdiff =
                   cp0_get_count() - ci->ci_cpu_counter_last;
                ci->ci_pendingticks++;
```

Complex case: pmap

- MIPS TLB entries are tagged with 8bit process id called ASID, used for improve performance
  MMU skips different process TLB entries on lookup
  We won't need to flush TLB every context switches
- Need to maintain process:ASID assign information because it's smaller than PID, we need to rotate it
- The information should keep beyond context switch
- We maintain ASID individually per-processor

- What we need is:
  ASID information per-processor * per-process

Complex case - pmap.c: original and modified code

```
uint pmap_alloc_tlbpid(struct proc *p)
...
                tlbpid_cnt = id + 1;
                pmap->pm_tlbpid = id;
```

```
uint pmap_alloc_tlbpid(struct proc *p)
...
                tlbpid_cnt[cpuid] = id + 1;
                pmap->pm_tlbpid[cpuid] = id;
```

Implement and use lock/atomic primitives

- Needed to implement
  - lock primitives: mutex, mp_lock
  - atomic primitives: CAS, 64bit add, etc..
- Acquiring giant lock prior to entering the kernel context
  - hardware interrupts
  - software interrupts
  - trap()

Acquiring giant lock on clock interrupt handler

```
uint32_t clock_int5(uint32_t mask, struct trap_frame *tf)
...
        if (tf->ipl < IPL_CLOCK) {
#ifdef MULTIPROCESSOR
                __mp_lock(&kernel_lock);
#endif
                while (ci->ci_pendingticks) {
                        clk_count.ec_count++;
                        hardclock(tf);
                        ci->ci_pendingticks--;
                }
#ifdef MULTIPROCESSOR
                __mp_unlock(&kernel_lock);
#endif
```

Acquiring giant lock on clock interrupt handler

```
uint32_t clock_int5(uint32_t mask, struct trap_frame *tf)
...
        if (tf->ipl < IPL_CLOCK) {
#ifdef MULTIPROCESSOR
                __mp_lock(&kernel_lock);
#endif
                while (ci->ci_pendingticks) {
                        clk_count.ec_count++;
                        hardclock(tf);
                        ci->ci_pendingticks--;
                }
#ifdef MULTIPROCESSOR
                __mp_unlock(&kernel_lock);
#endif
```

Actually, it causes a bug... described later

Hardware related problems

- Spin up secondary processor
- Keeping TLB consistency by software
- Cache coherency

Spin up secondary processor

- We need to launch secondary processor
  - Access hardware register on controller device to power on the processor
- Secondary processor needs own bootstrap code
  - Has to be similar to primary processor's one
  - But we won't need kernel, memory, and device initialization
  - Because primary processor already did it

Keeping TLB consistency by software

- MIPS TLB doesn't have mechanism to keep consistency, we need to do it by software Just like the other typical architectures
- Invalidate/update other processor's TLB using TLB shootdown
  - It implemented by IPI (Inter-Processor Interrupt)

Cache coherency

- MIPS R10000/R12000 processors have full cache coherency
- We don’t have to care about it on Octane
- But, some other processors haven’t full cache coherency

Ideas implementing SMP

- We have faced a number of issues while implementing SMP
  - Fighting against deadlock
  - Dynamic memory allocation without using virtual address
  - Reduce frequency of TLB shootdown

Fighting against deadlock

- It's hard to find the cause of deadlocks because both processors runs concurrently
- It causes timing bugs - conditions are depend on timing
- Need to be able to determine what happened on both processors at that time
- There are tools for debugging it

JTAG ICE

- Very useful for debugging
- We can get any data for debugging on desired timing, even after kernel hangs
- I used it when I was implementing SMP for the embedded OS
- Not for Octane, there's no way to connect

ddb

- OpenBSD kernel has in-kernel debugger, named ddb
- We can get similar data for JTAG ICE, but kernel need to alive - because it's part of the kernel
- Missing features:
  We hadn't implemented "machine ddbcpu<#>" - which is processor switching function on ddb
  Without this, we can only debug one processor which invoked ddb on a breakpoint
- Not always useful

printf()

- Most popular kernel debugging tool
- Just write printf(message) on your code, Easy to use ;)
- Unfortunately, it has some problems
  - printf() wastes lot of cycles, changes timing between processors
    We may miss timing bug because of it
  - Some point of the code are printf() unsafe
    causes kernel hang
- We use it anyway

Divide printf output for two serial port

- There's two serial port and two processors
- If we have lots of debug print, it's hard to understand which lines belongs to what processor
- I implemented dirty hack code which output strings directly to secondary serial port, named combprintf()
- Rewrite debug print to
  primary processor outputs primary serial port,
  secondary processor outputs secondary serial port

What do we need to print for debugging?

- To know where the kernel running roughly
  - put debug print everywhere the point kernel may running through
  - dump all system call using SYSCALL_DEBUG
- How can we determine a deadlock point?

Determine a deadlock point

- Deadlocks are occurring on spinlocks
- It loops permanently until a condition become available, but that condition never comes up
- At least to know which lock primitives we stuck on, we need to stop permanent loop by implementing timeout counter and print debug message

Adding timeout counter into mutex

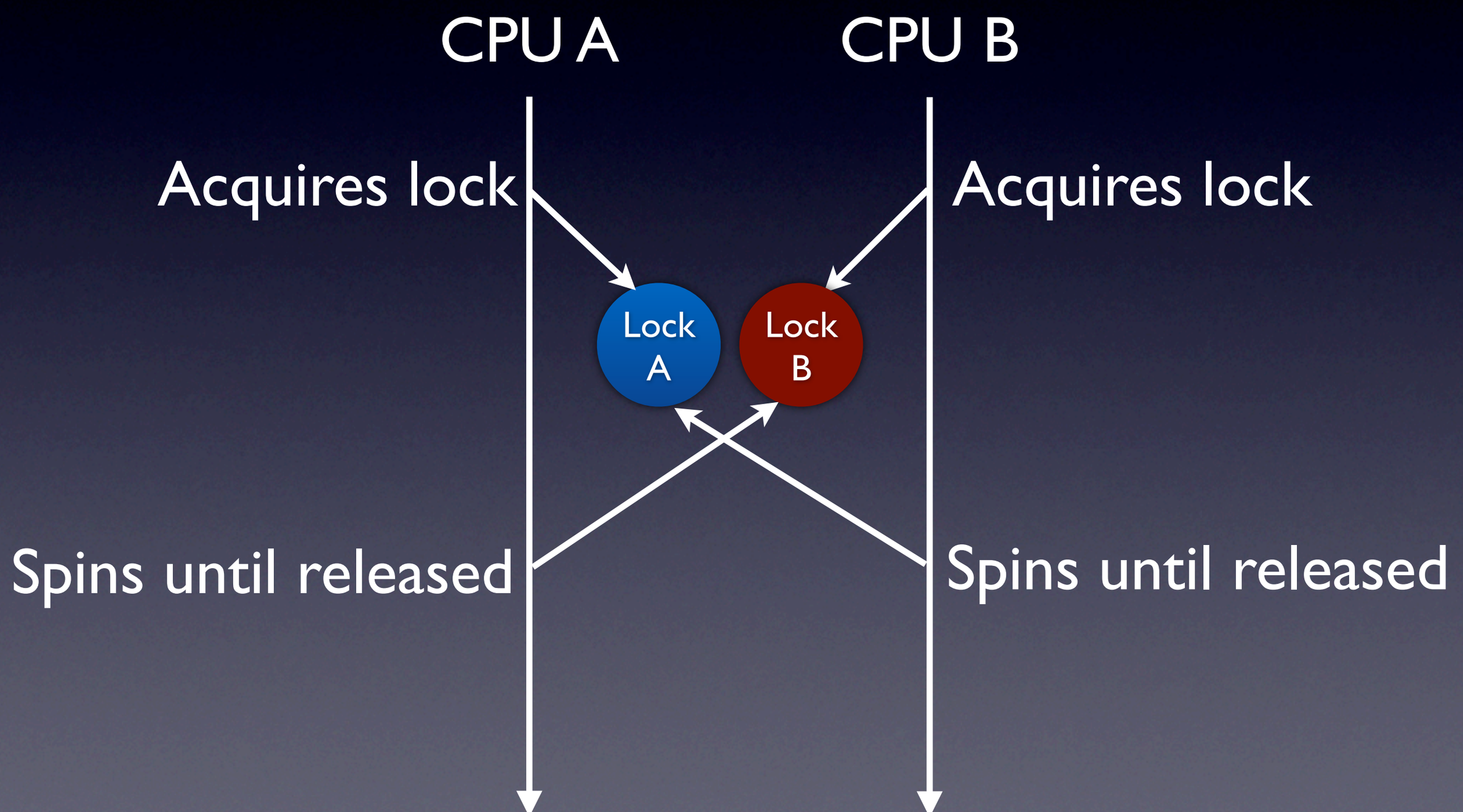
```
void mtx_enter(struct mutex *mtx)
...
        for (;;) {
                if (mtx->mtx_wantipl != IPL_NONE)
                        s = splraise(mtx->mtx_wantipl);
                if (try_lock(mtx)) {
                        if (mtx->mtx_wantipl != IPL_NONE)
                                mtx->mtx_oldipl = s;
                        mtx->mtx_owner = curcpu();
                        return;
                }
                if (mtx->mtx_wantipl != IPL_NONE)
                        splx(s);
                if (++i > MTX_TIMEOUT)
                        panic("mtx deadlocked\n”);
        }
```

Adding timeout counter into mutex

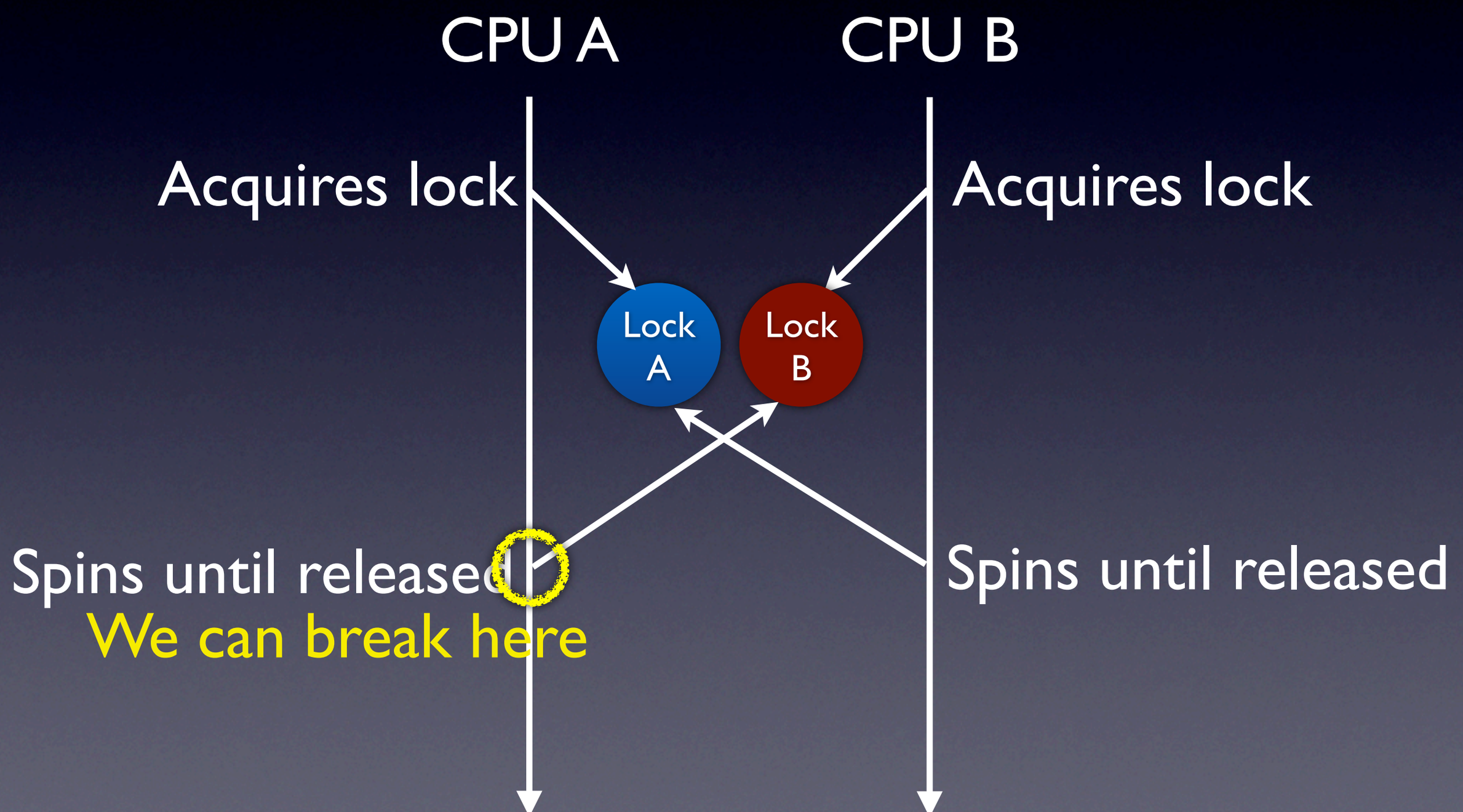
```
void mtx_enter(struct mutex *mtx)
...
        for (;;) {
                if (mtx->mtx_wantipl != IPL_NONE)
                        s = splraise(mtx->mtx_wantipl);
                if (try_lock(mtx)) {
                        if (mtx->mtx_wantipl != IPL_NONE)
                                mtx->mtx_oldipl = s;
                        mtx->mtx_owner = curcpu();
                        return;
                }
                if (mtx->mtx_wantipl != IPL_NONE)
                        splx(s);
                if (++i > MTX_TIMEOUT)
                        panic("mtx deadlocked\n”);
        }
```

But, this is not enough

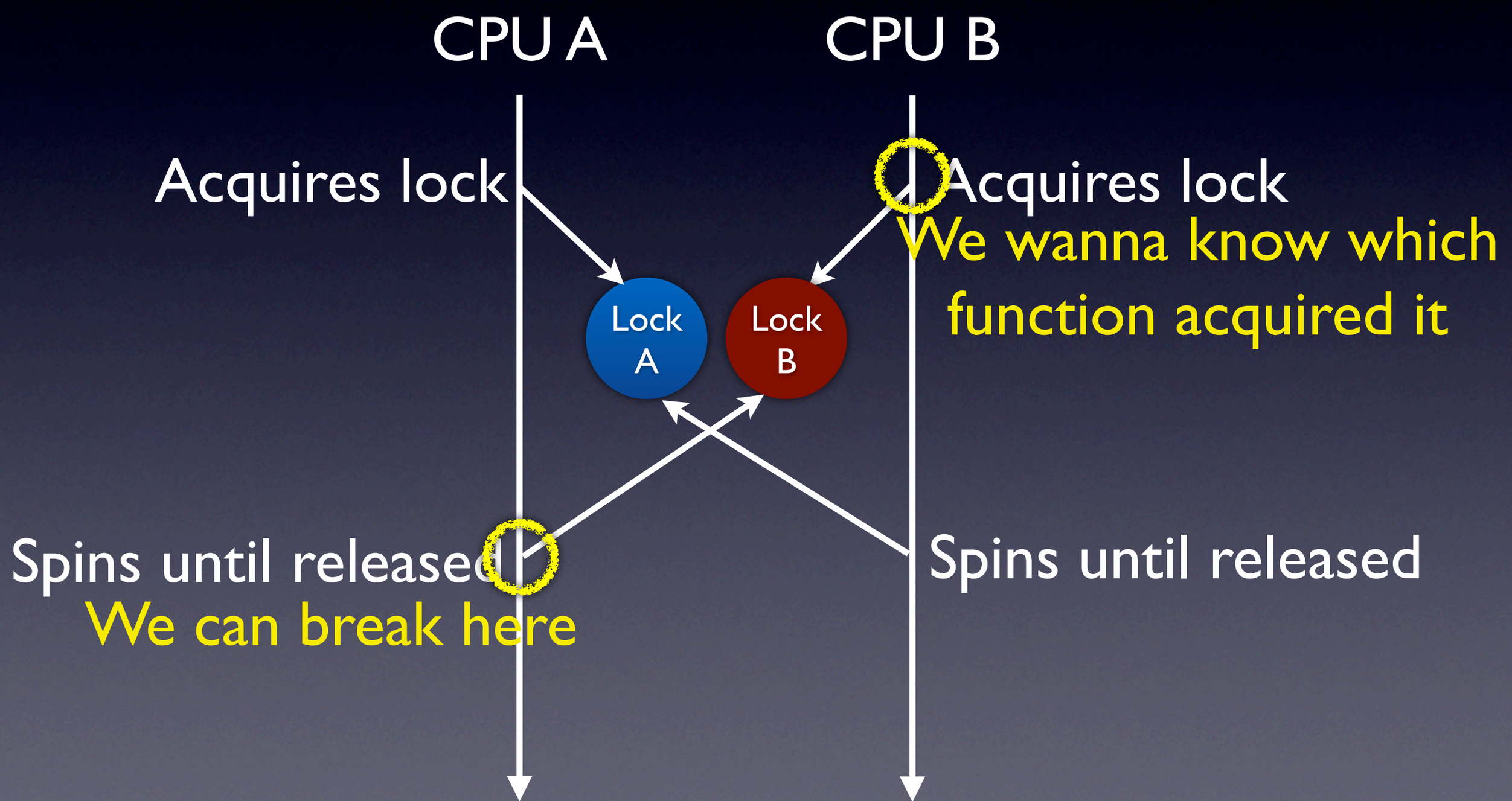
Why it's not enough



Why it's not enough



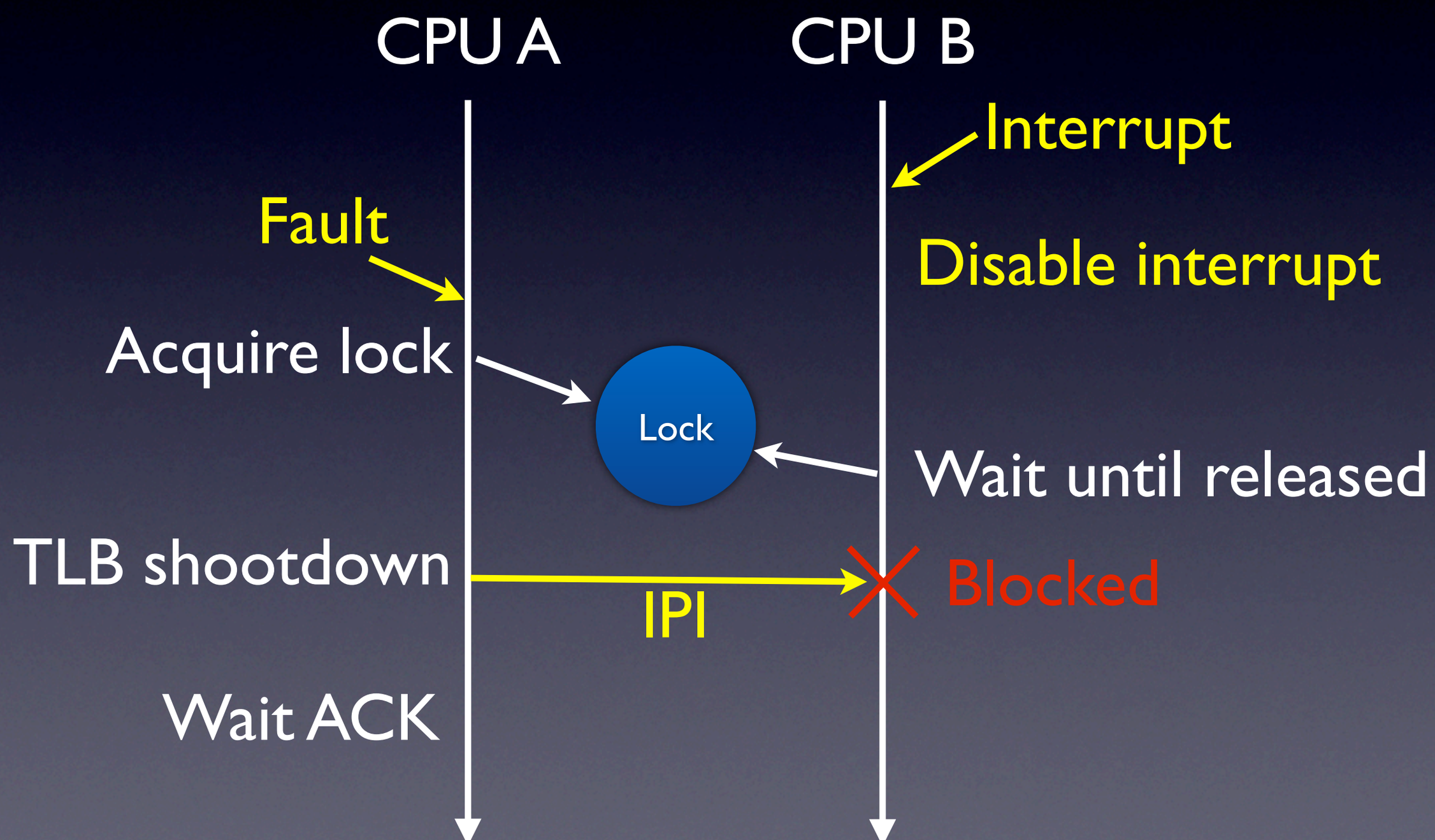
Why it's not enough



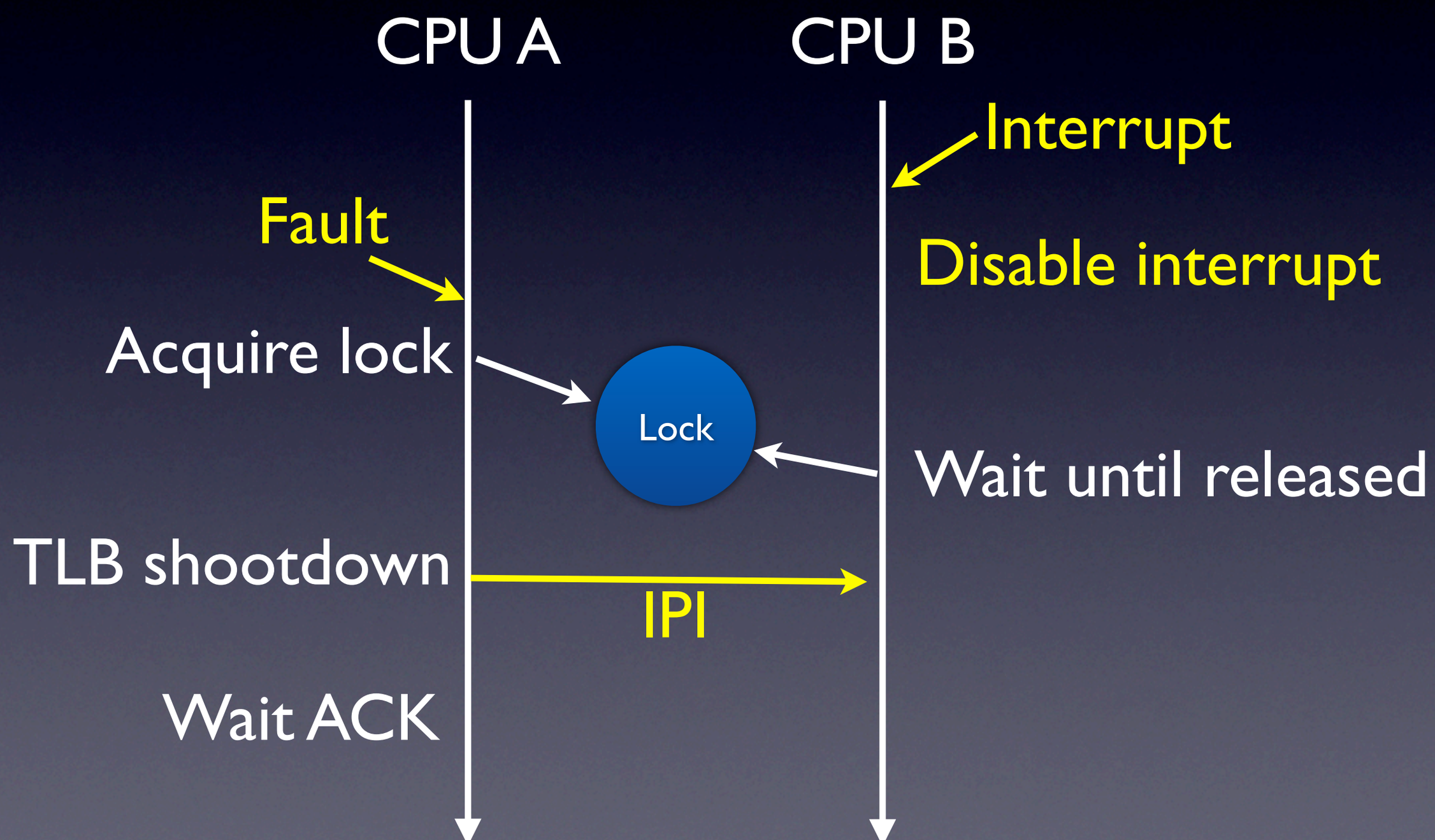
Remember who acquired it

```
void mtx_enter(struct mutex *mtx)
...
        for (;;) {
                if (mtx->mtx_wantipl != IPL_NONE)
                        s = splraise(mtx->mtx_wantipl);
                if (try_lock(mtx)) {
                        if (mtx->mtx_wantipl != IPL_NONE)
                                mtx->mtx_oldipl = s;
                        mtx->mtx_owner = curcpu();
                        mtx->mtx_ra =
                            __builtin_return_address(0);
                        return;
                }
                if (mtx->mtx_wantipl != IPL_NONE)
                        splx(s);
                if (++i > MTX_TIMEOUT)
                        panic("mtx deadlocked ra:%p\n",
                                mtx->mtx_ra);
        }
```

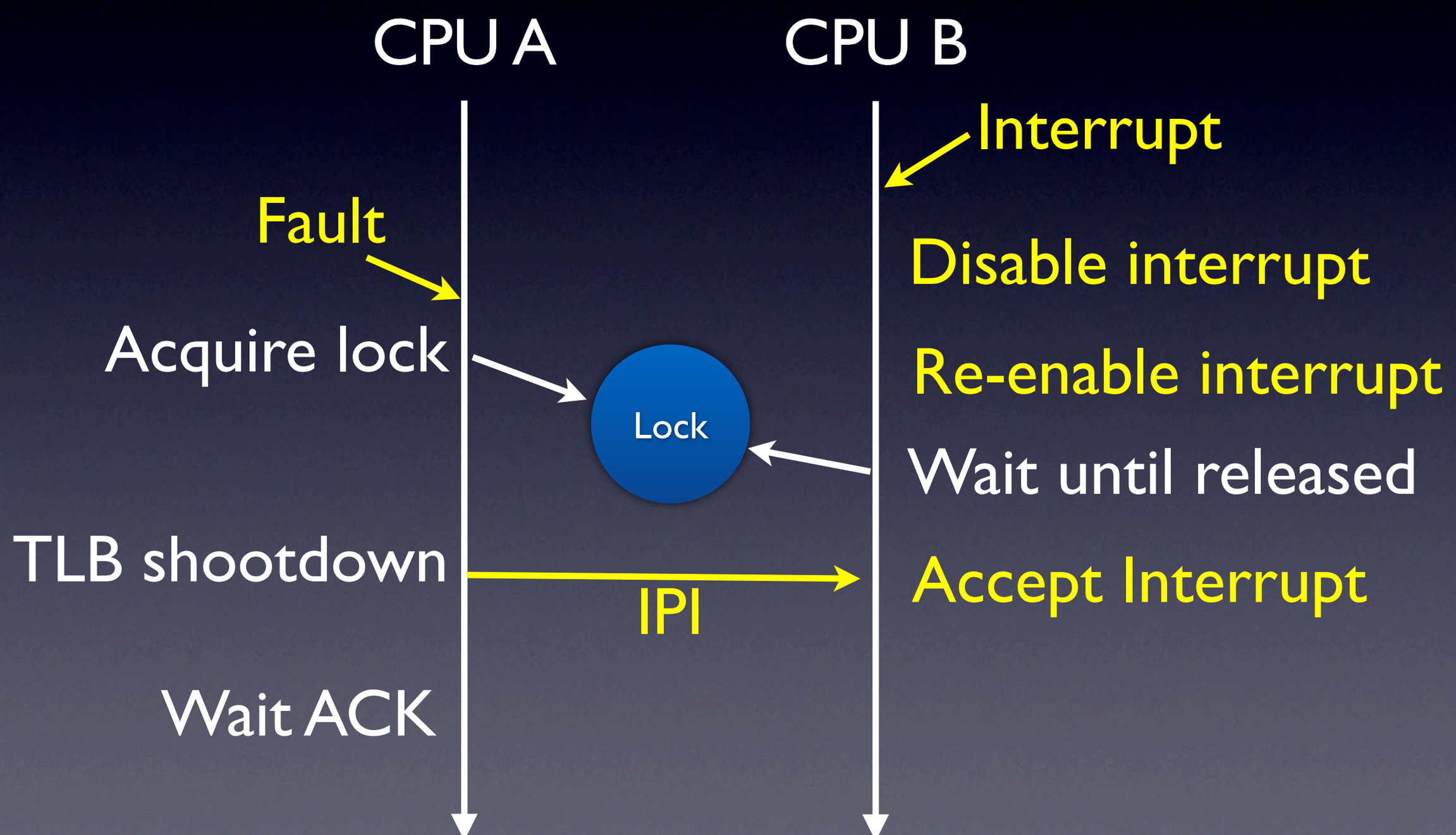
Interrupt blocks IPI



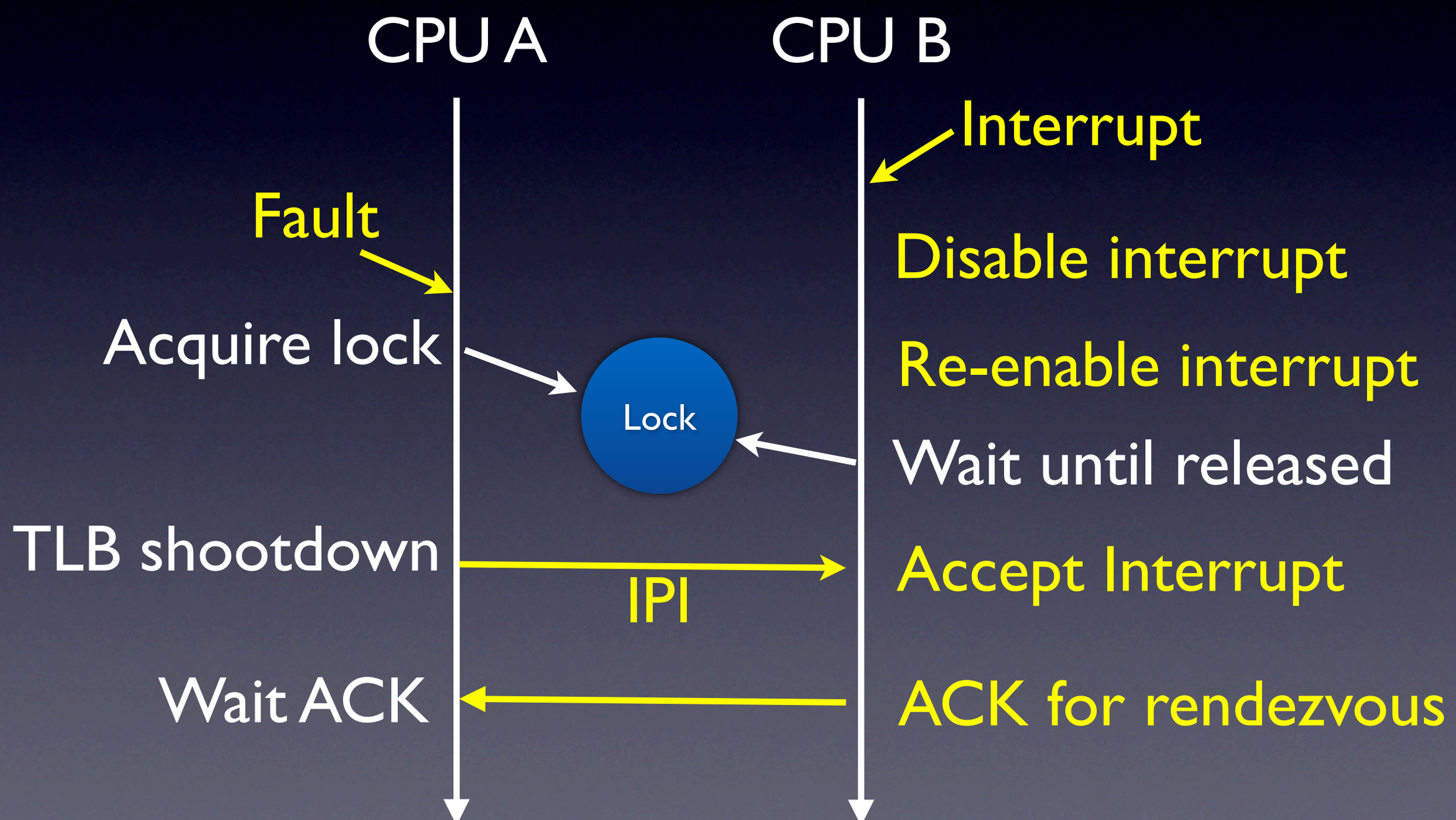
Interrupt blocks IPI



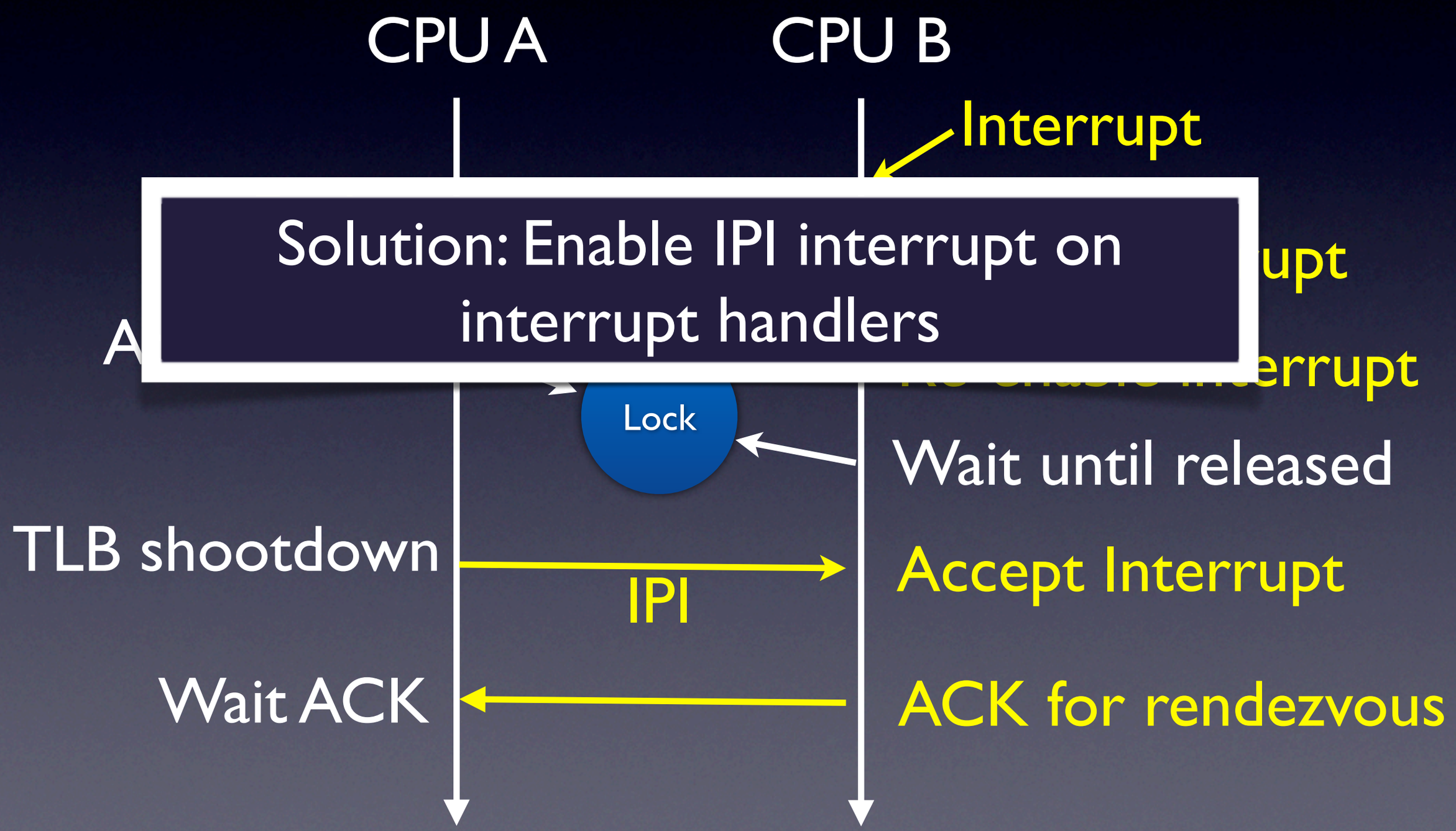
Interrupt blocks IPI



Interrupt blocks IPI



Interrupt blocks IPI



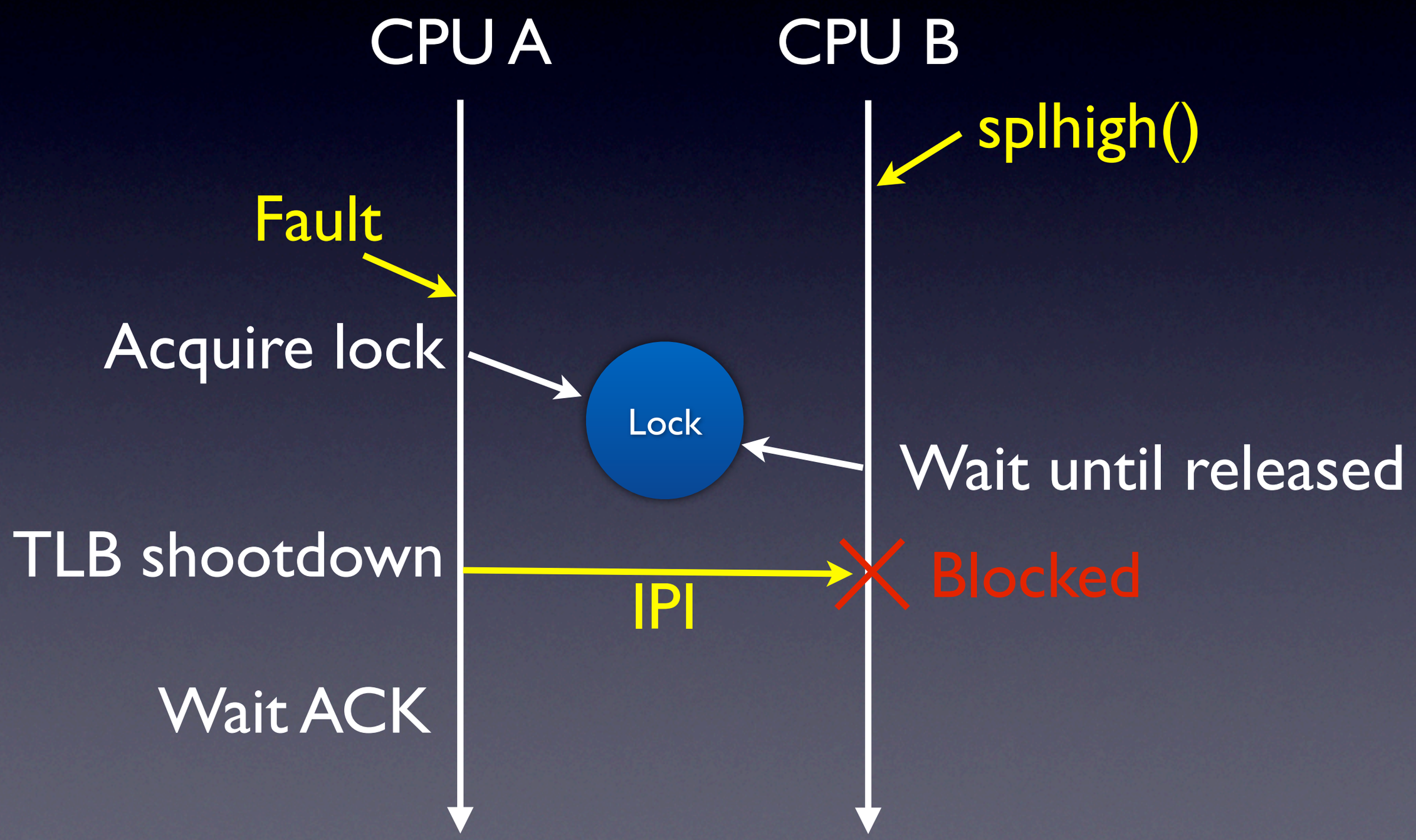
Solution: Enable IPI interrupt on interrupt handlers

Fixing clock interrupt handler

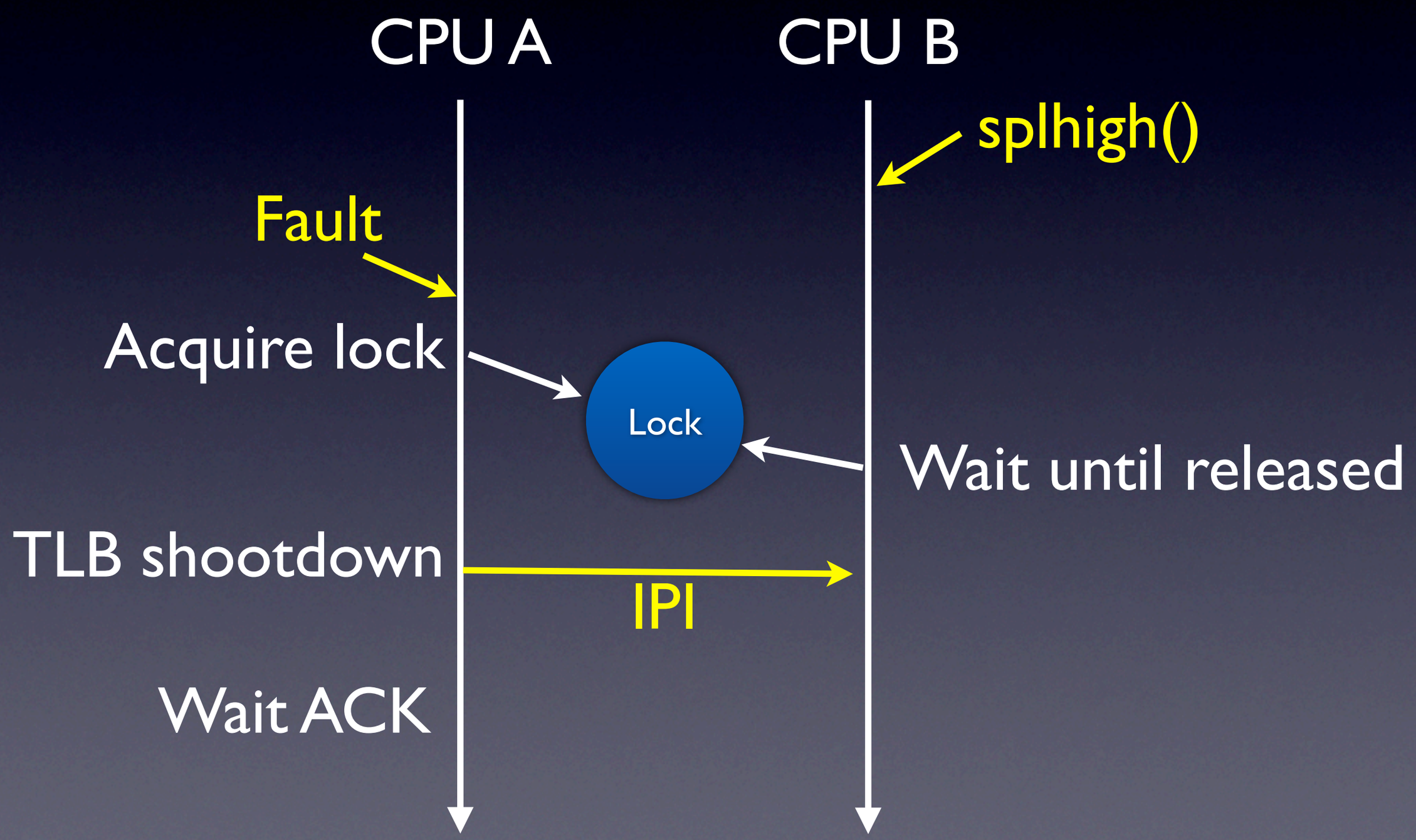
```
uint32_t clock_int5(uint32_t mask, struct trap_frame *tf)
...
        if (tf->ipl < IPL_CLOCK) {
#ifdef MULTIPROCESSOR
                u_int32_t sr;

                sr = getsr();
                ENABLEIPI();
                __mp_lock(&kernel_lock);
#endif
                while (ci->ci_pendingticks) {
                        clk_count.ec_count++;
                        hardclock(tf);
                        ci->ci_pendingticks--;
                }
#ifdef MULTIPROCESSOR
                __mp_unlock(&kernel_lock);
                setsr(sr);
#endif
```

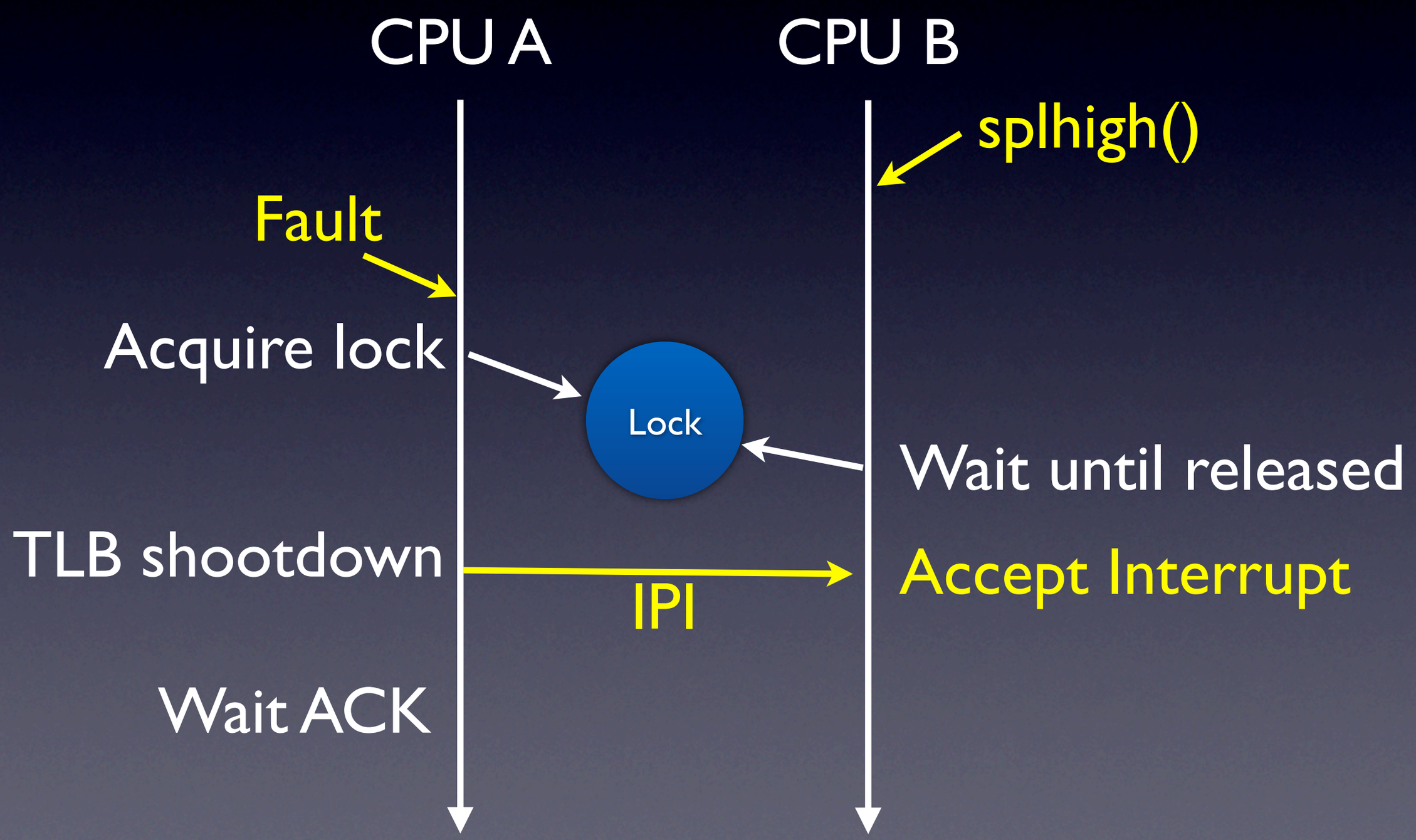
splhigh() blocks IPI



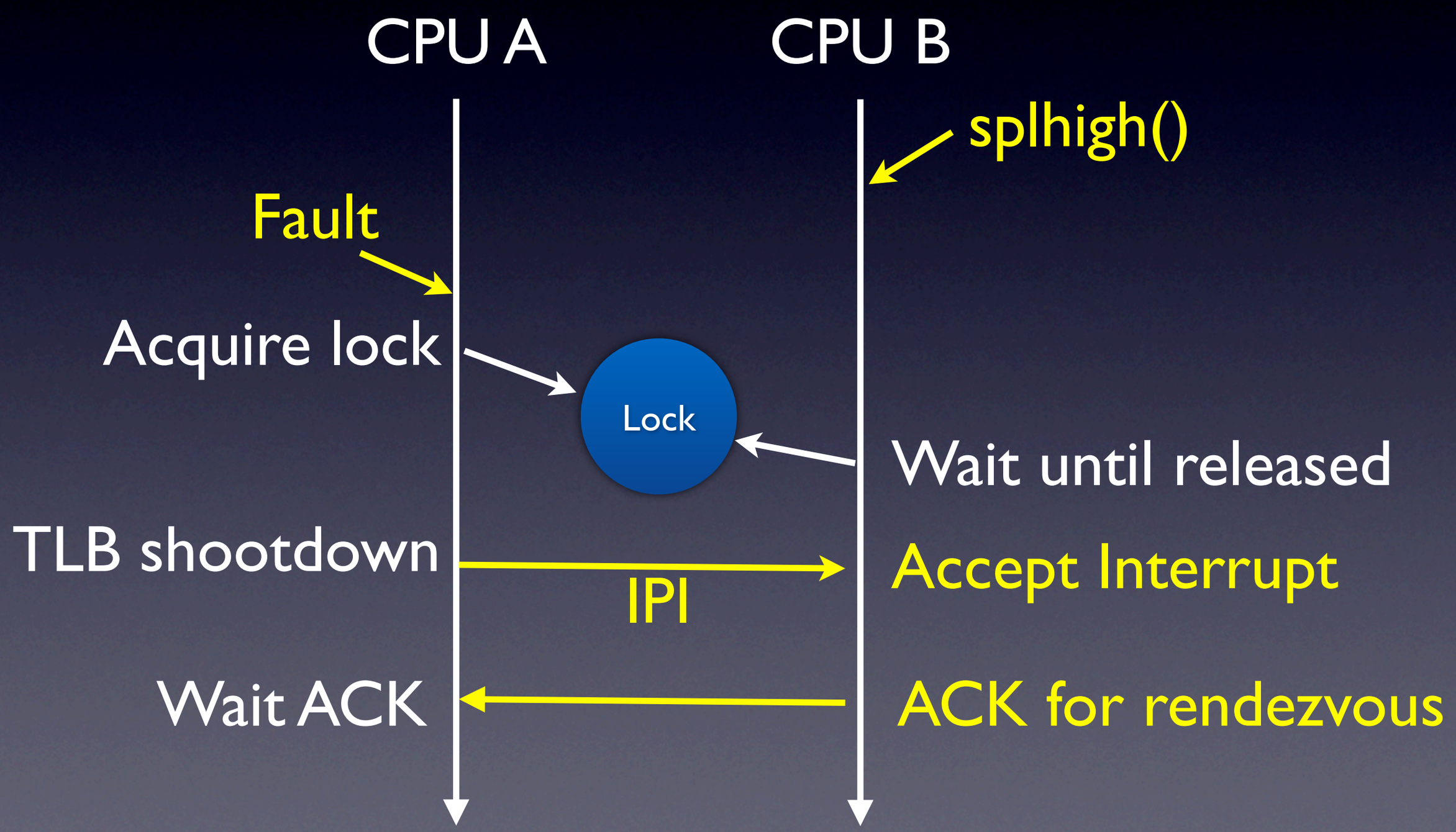
splhigh() blocks IPI



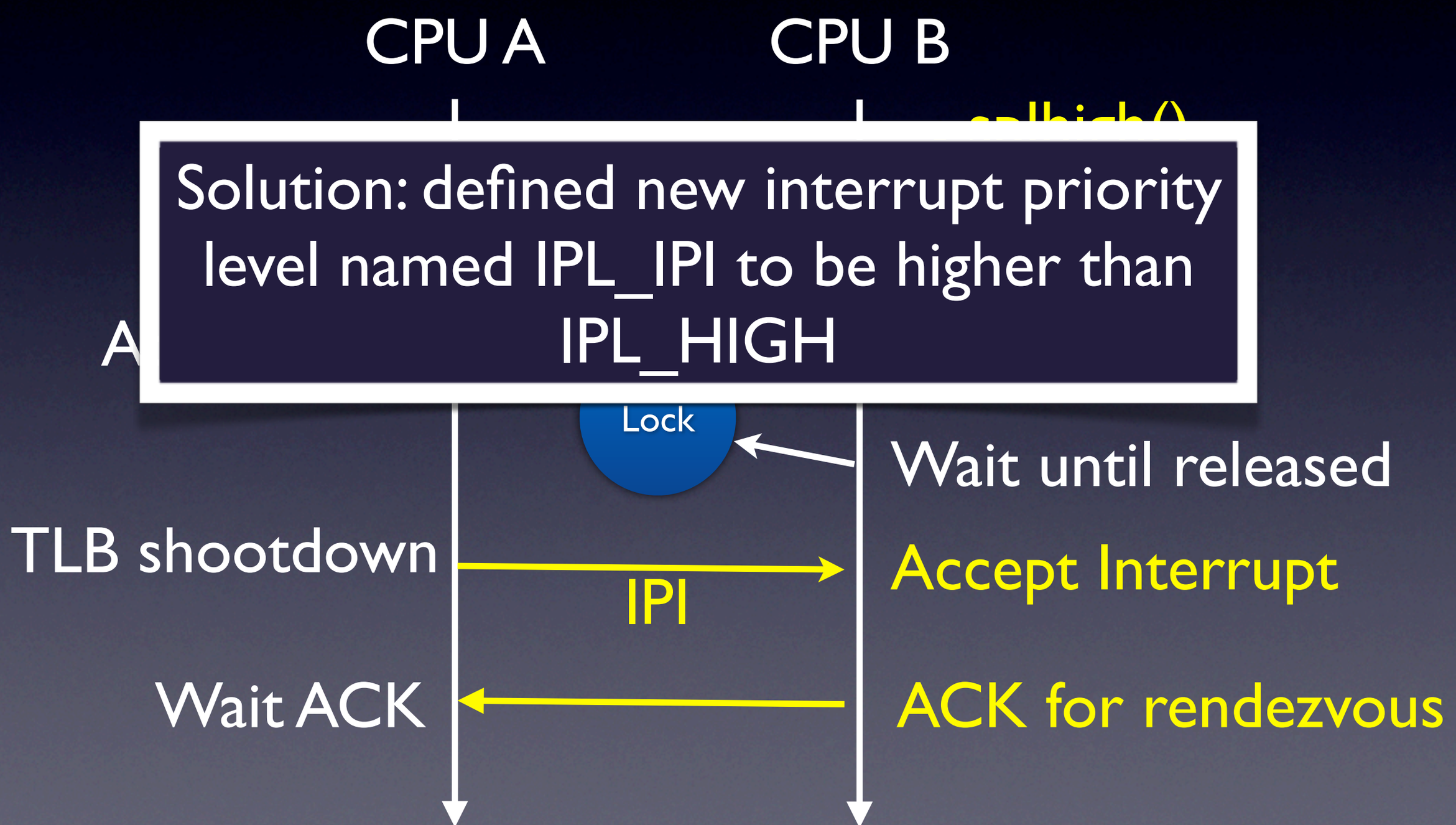
splhigh() blocks IPI



splhigh() blocks IPI



splhigh() blocks IPI



Solution: defined new interrupt priority level named IPL_IPI to be higher than IPL_HIGH

Adding new interrupt priority level

```
#define        IPL_TTY        4        /* terminal */
#define        IPL_VM         5        /* memory allocation */
#define        IPL_CLOCK      6        /* clock */
#define        IPL_HIGH       7        /* everything */
#define        NIPLS          8        /* Number of levels */
```

```
#define        IPL_TTY        4        /* terminal */
#define        IPL_VM         5        /* memory allocation */
#define        IPL_CLOCK      6        /* clock */
#define        IPL_HIGH       7        /* everything */
#define        IPL_IPI        8        /* ipi */
#define        NIPLS          9        /* Number of levels */
```

Dynamic memory allocation without using virtual address

- To support N(>2) processors, the cpu_info structure and the bootstrap kernel stack for secondary processors should be allocated dynamically
- But we can't use virtual address for them
  - stack may used before TLB initialization, thus causing the processor fault
  - MIPS has Software TLB, need to maintain TLB by software
    TLB miss handler is the code to handle it
    This handler refers cpu_info, it cause TLB miss loop
- To avoid these problems, we implemented wrapper function to allocate memory dynamically, then get the physical address and return it

The wrapper function

```
vaddr_t smp_malloc(size_t size)
...
        if (size < PAGE_SIZE) {
                va = (vaddr_t)malloc(size, M_DEVBUF, M_NOWAIT);
                if (va == NULL)
                        return NULL;
                error = pmap_extract(pmap_kernel(), va, &pa);
                if (error == FALSE)
                        return NULL;
        } else {
                TAILQ_INIT(&mlist);
                error = uvm_pglistalloc(size, 0, -1L, 0, 0,
                    &mlist, 1, UVM_PLA_NOWAIT);
                if (error)
                        return NULL;
                m = TAILQ_FIRST(&mlist);
                pa = VM_PAGE_TO_PHYS(m);
        }

        return PHYS_TO_XKPHYS(pa, CCA_CACHED);
```

Reduce frequency of TLB shootdown

- There's a condition we can skip TLB shootdown in invalidate/update:

	using the pagetable	not using
kernel mode	need	need
user mode	need	don't need

- In user mode, if shootee processor doesn't using the pagetable, we won't need shootdown; just changing ASID assign is enough
- In reference pmap implementation, a TLB shootdown performed non-conditionally, even in case it isn't really needed
- We added the condition to reduce frequency of it

```
void pmap_invalidate_page(pmap_t pmap, vm_offset_t va)
...
   arg.pmap = pmap;
   arg.va = va;
   smp_rendezvous(0, pmap_invalidate_page_action, 0,
   (void *)&arg);

void pmap_invalidate_page_action(void *arg)
...
   pmap_t pmap = ((struct pmap_invalidate_page_arg *)arg)->pmap;
   vm_offset_t va = ((struct pmap_invalidate_page_arg *)arg)->va;
   if (is_kernel_pmap(pmap)) {
      pmap_TLB_invalidate_kernel(va);
      return;
   }
   if (pmap->pm_asid[PCPU_GET(cpuid)].gen
       != PCPU_GET(asid_generation))
      return;
   else if (!(pmap->pm_active & PCPU_GET(cpumask))) {
      pmap->pm_asid[PCPU_GET(cpuid)].gen = 0;
      return;
   }
   va = pmap_va_asid(pmap, (va & ~PGOFSET));
   mips_TBIS(va);
```

```
CPU_INFO_FOREACH(cii, ci)
        if (cpuset_isset(&cpus_running, ci)) {
                unsigned int i = ci->ci_cpuid;
                unsigned int m = 1 << i;
                if (pmap->pm_asid[i].pma_asidgen !=
                    pmap_asid_info[i].pma_asidgen)
                        continue;
                else if (ci->ci_curpmap != pmap) {
                        pmap->pm_asid[i].pma_asidgen = 0;
                        continue;
                }
                cpumask |= m;
        }

if (cpumask == 1 << cpuid) {
        u_long asid;

        asid = pmap->pm_asid[cpuid].pma_asid << VMTLB_PID_SHIFT;
        tlb_flush_addr(va | asid);
} else if (cpumask) {
        struct pmap_invalidate_page_arg arg;
        arg.pmap = pmap;
        arg.va = va;

        smp_rendezvous_cpus(cpumask, pmap_invalidate_user_page_action,
                &arg);
}
```

Future works

- Commit machine ddbcpu<#>
- New port for Cavium OCTEON,
  if it's acceptable for OpenBSD project
- Maybe SMP support for SGI Origin 350
- Also interested in MI part of SMP