Interrupts

Krste Asanović
UC Berkeley & SiFive Inc.
krste@berkeley.edu

4th RISC-V Workshop
MIT CSAIL, Cambridge, MA
July 12, 2016

Pre-workshop draft – see riscv.org for final version

Interrupt Uses in Different Applications

- High-performance Unix-like systems
  - Interrupt handling small fraction of processing time
    - Fast cores, smart devices
  - Minimal interrupt handler
  - Scheduling in software
- Low/mid embedded systems
  - Interrupt handling significant fraction of processor time
    - Slow cores, dumb devices
  - Significant fraction of code in handlers
  - Interrupt controller acts as task scheduler
- High-performance real-time systems
  - Can't waste time on interrupt overhead
  - Handlers poll I/O devices with regular heartbeat
- And everything inbetween

RISC-V Interrupt Design Goals

- Simplicity
- Support all kinds of platforms from microcontrollers to virtualized servers
- Enable tradeoffs between performance and implementation cost
- Flexibility to support specialized needs

Categorizing Sources of RISC-V Interrupts

- Local Interrupts
  - Directly connected to one hart
  - No arbitration between harts to service
  - Determine source directly through `xcause` CSR
  - Only two standard local interrupts (software, timer)

- Global (External) Interrupts
  - Routed via Platform-Level Interrupt Controller (PLIC)
  - PLIC arbitrates between multiple harts claiming interrupt
  - Read of memory-mapped register returns source

Machine Interrupt Pending CSR (`mip`)

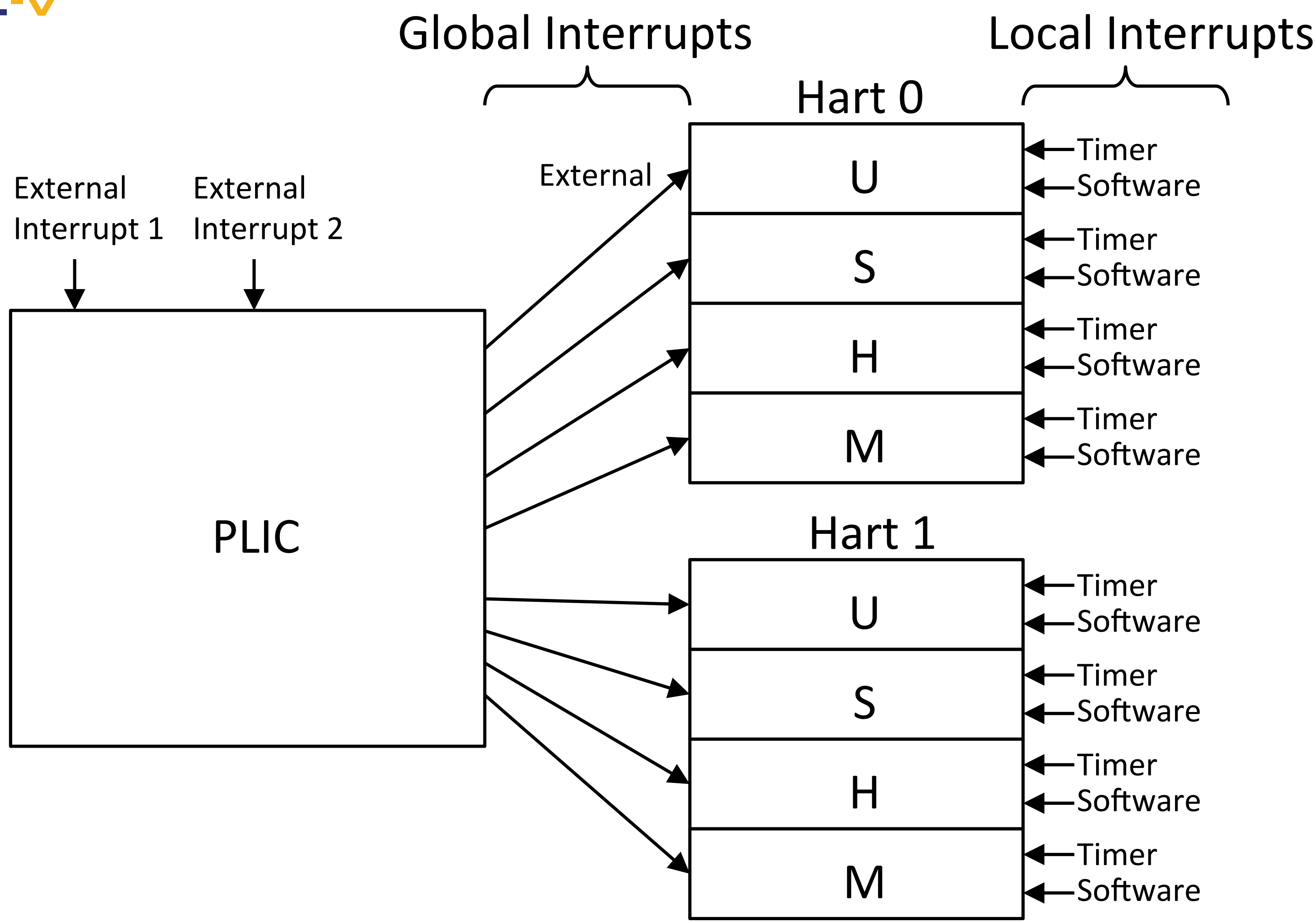
(Add Non-Standard Local Interrupts Here)

XLEN-1 … 12	11	10	9	8	7	6	5	4	3	2	1	0
WIRI	MEIP	HEIP	SEIP	UEIP	MTIP	HTIP	STIP	UTIP	MSIP	HSIP	SSIP	USIP
XLEN-12	1	1	1	1	1	1	1	1	1	1	1	1

External from PLIC (bits 11–8) — Local Timer (bits 7–4) — Local Software (bits 3–0)

- **`mip`** reflects pending status of interrupts for hart
- Separate interrupts for each supported privilege level (M/H/S/U)
- User-level interrupt handling ("N") optional feature when U-mode present (discussed later)

Platform-Level Interrupt Controller (PLIC)



Software Interrupts

- MSIP
  - Only writeable in machine-mode via memory-mapped control register (mapping is platform-specific)
  - One hart can write to different hart's MSIP register
  - Mechanism for inter-hart interrupts
- HSIP, SSIP, USIP
  - Hart can only write bit *x*SIP in own **`mip`** register when running at privilege mode *x* or greater
- App/OS/Hypervisor can only perform inter-hart interrupts via ABI/SBI/HBI calls
  - Destination virtual hart might be descheduled
  - Interrupts virtualized by M-mode software using MSIP

Timer Interrupts

- MTIP
  - Single 64-bit real-time hardware timer and comparator in M-mode
  - MTIP set when **`mtime`** >= **`mtimecmp`**
  - MTIP cleared by writing new **`mtimecmp`** value
- HTIP, STIP, UTIP
  - M-mode multiplexes single hardware timer and comparator for lower-privilege modes on same hart
  - ABI/SBI/HBI calls to set up timer
  - M-mode software writes/clears HTIP/STIP/UTIP

- Most systems will also have other hardware timers attached via PLIC etc.

Machine Interrupt Enable CSR (`mie`)

XLEN-1 12	11	10	9	8	7	6	5	4	3	2	1	0
WPRI	MEIE	HEIE	SEIE	UEIE	MTIE	HTIE	STIE	UTIE	MSIE	HSIE	SSIE	USIE
XLEN-12	1	1	1	1	1	1	1	1	1	1	1	1

External from PLIC (MEIE, HEIE, SEIE, UEIE)

Local Timer (MTIE, HTIE, STIE, UTIE)

Local Software (MSIE, HSIE, SSIE, USIE)

- **`mie`** mirrors layout of **`mip`**
- provides per-interrupt enables

Interrupts in `mstatus`

XLEN-1	XLEN-2 … 29	28 … 24	23 … 19	18	17	16 15	14 13
SD	**WPRI**	VM[4:0] (**WARL**)	**WPRI**	PUM	MPRV	XS[1:0]	FS[1:0]
1	XLEN-30	5	5	1	1	2	2

12 11	10 9	8	7	6	5	4	3	2	1	0
MPP[1:0]	HPP[1:0]	SPP	MPIE	HPIE	SPIE	UPIE	MIE	HIE	SIE	UIE
2	2	1	1	1	1	1	1	1	1	1

Privilege stack

Enable stack

Per-privilege global interrupt enables

- Only take a pending interrupt for privilege mode *x* if *x*IE=1 and running in mode *x* or greater
- Interrupts always disabled for privileges less than current level

All interrupts trap to M-mode by default

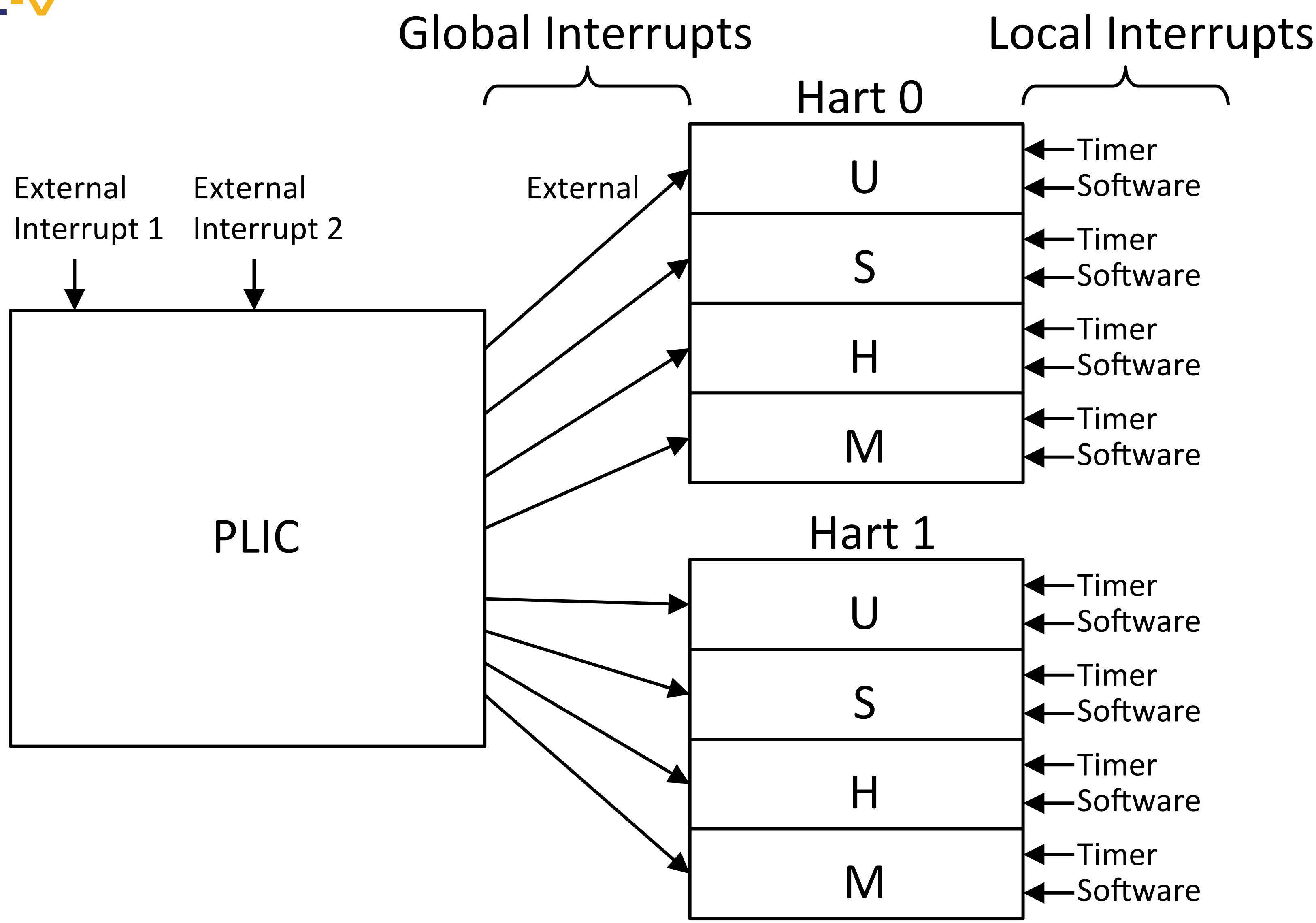
- **`mcause`** CSR indicates which interrupt occurred
- M-mode can redirect to other privilege level by:
  - set up target interrupt and privilege stack
  - copy **`mepc`** to **`hepc/sepc/uepc`** respectively
  - copy **`mcause`** to **`hcause/scause/ucause`**
  - set **`mepc`** to target trap vector
  - set MPP to target privilege level, MPIE to false
  - execute **`mret`**

Interrupt	Exception Code	Description
1	0	User software interrupt
1	1	Supervisor software interrupt
1	2	Hypervisor software interrupt
1	3	Machine software interrupt
1	4	User timer interrupt
1	5	Supervisor timer interrupt
1	6	Hypervisor timer interrupt
1	7	Machine timer interrupt
1	8	User external interrupt
1	9	Supervisor external interrupt
1	10	Hypervisor external interrupt
1	11	Machine external interrupt
1	$\geq$12	*Reserved*

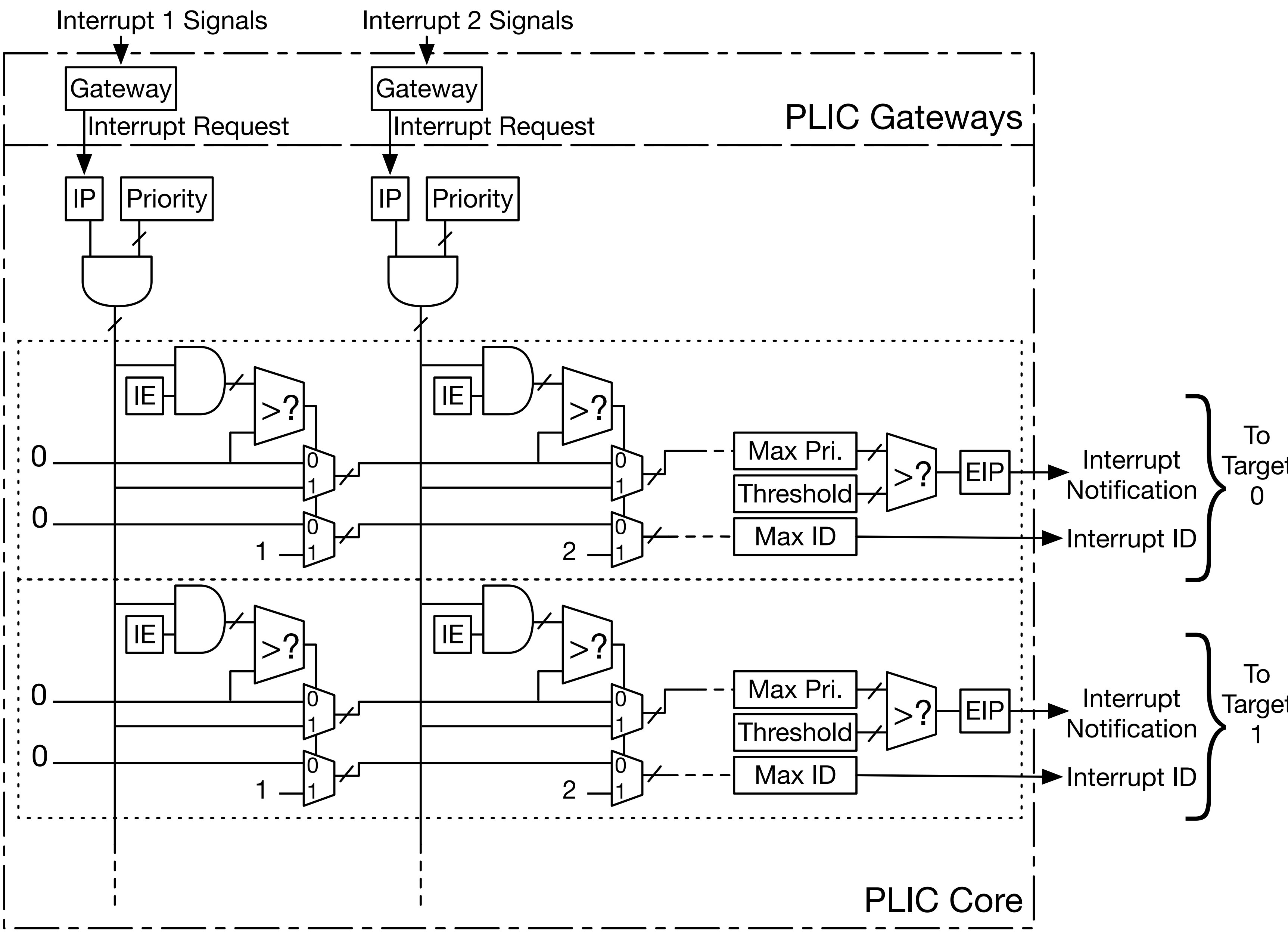
Optional Interrupt Handler Delegation

- Can delegate interrupt (and exception) handling to lower privilege level to reduce overhead
- **`mideleg`** has same layout as **`mip`**
- If a bit is set in **`mideleg`** then corresponding interrupt delegated to next lowest privilege level (H, S, or U)
- Can be delegated again using **`hideleg`**/**`sideleg`**
- Once delegated, the interrupt will not affect current privilege level (MIE setting ignored)

Platform-Level Interrupt Controller (PLIC)



PLIC Conceptual Block Diagram



PLIC Interrupt Gateways

Convert from external interrupt signal/message encoding to internal PLIC interrupt request, e.g.,

- Level-triggered gateways
- Edge-triggered gateways
- Message-signaled gateways
- XXX gateways in future

Will not forward a new request to PLIC core unless previous request's handler has signaled completion

- Level-triggered will issue new PLIC interrupt request if level still asserted after completion signaled
- Edge-triggered/message-signaled could queue requests

PLIC Per-Interrupt ID and Priority

- Each interrupt has ID and priority

- Interrupt IDs are integers from 1…N
- ID of zero means "no interrupt"

- Priorities are integers, larger number is higher priority
- Priority zero means "never interrupt"
- Priorities can be fixed or variable
  - Degenerate case, all are fixed at "1".
- Ties broken by ID (lower ID is higher priority)

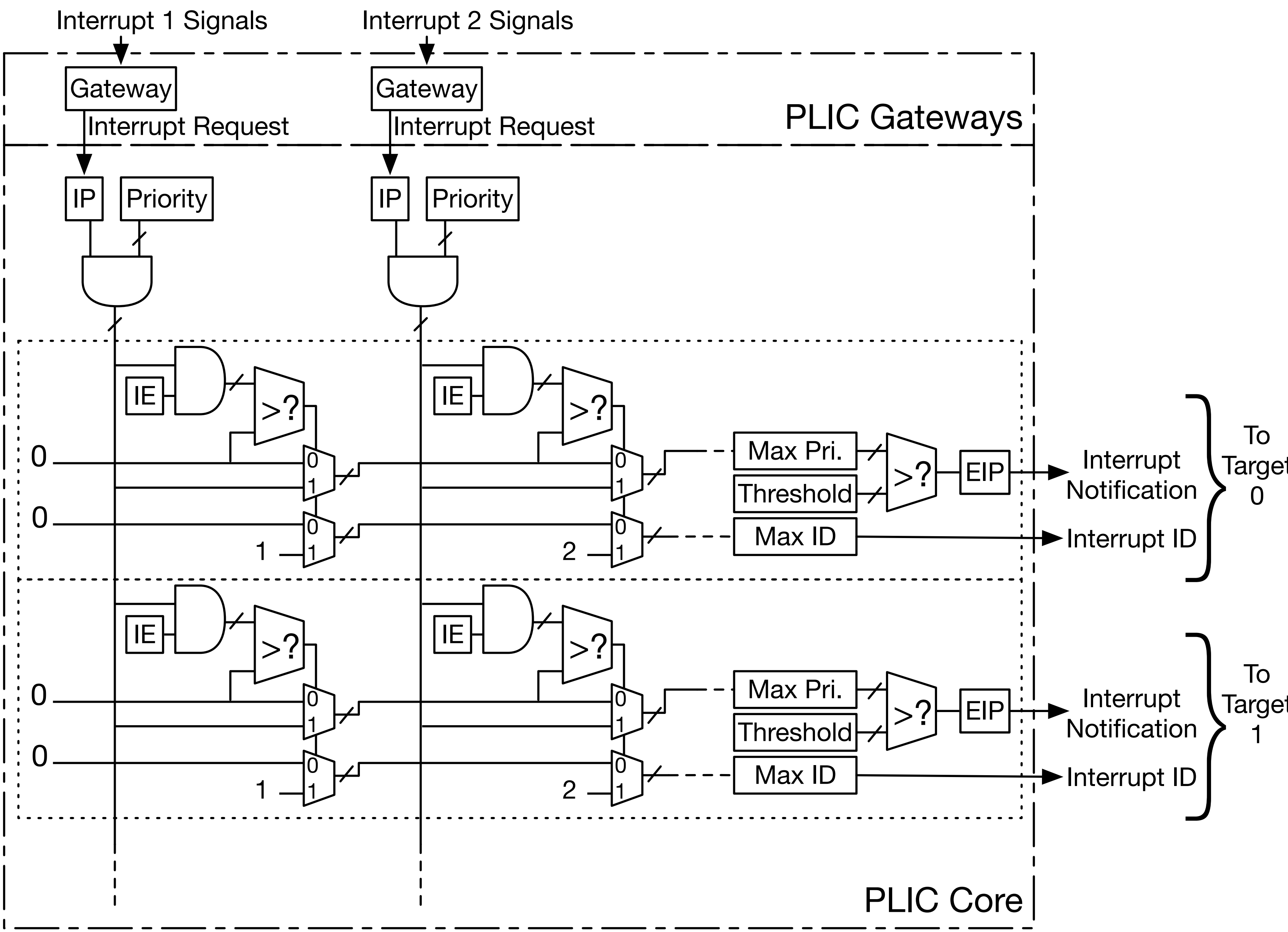
PLIC Per-Target Information

- Each target has vector of interrupt enables

- Each target has priority threshold
- Only interrupts with priority above threshold will cause interrupt
- Set threshold to 0, has no effect
  - Minimal implementation, hardwire threshold to zero
- Set threshold to MAX_PRI, then all interrupts masked

- Interrupt notifications asserted at target if enabled interrupt is above threshold
  - Notifications can take arbitrary time to arrive at target

PLIC Conceptual Block Diagram



PLIC Claim/Completion

- Interrupted hart context *claims* interrupt from PLIC with read of memory-mapped register
- PLIC returns highest priority active interrupt for that hart
- Can return 0 if no active interrupts remain
  - Other hart might have claimed interrupt earlier

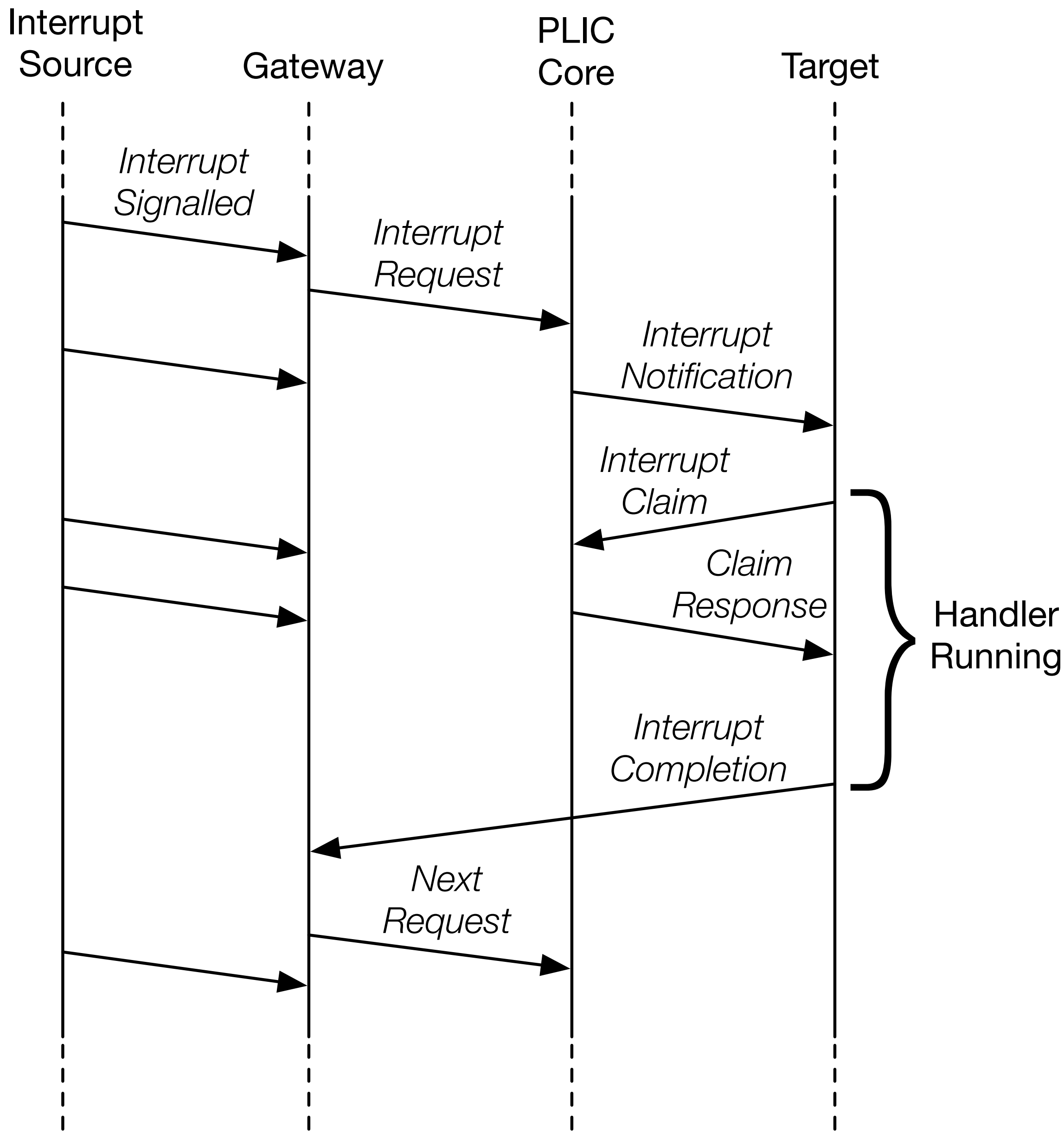
- Hart signals completion to gateway after handler finishes

PLIC Core Atomic Actions

- **Write Register:** A message containing a register write request is dequeued. One of the internal registers is written, where an internal register can be a priority, an interrupt-enable (IE), or a threshold.
- **Accept Request:** If the IP bit corresponding to the interrupt source is clear, a message containing an interrupt request from a gateway is dequeued and the IP bit is set.
- **Process Claim:** An interrupt claim message is dequeued. A claim-response message is enqueued to the requester with the ID of the highest-priority active interrupt for that target, and the IP bit corresponding to this interrupt source is cleared.

- Implementations can perform one action over many cycles, or many actions per cycle, provided behavior agrees with some sequence of these actions

PLIC Interrupt Flow



PLIC Interrupt Preemption/Nesting

- Preemption and nesting are function of the target core, not the PLIC
- Need a different hart context to receive nested interrupt
- Each standard RISC-V privilege level can provide one level of preemption/nesting
  - M-mode interrupt will preempt S-mode handler on hart
- Can add additional hart contexts to core to support nested interrupt handling, with per-cores rules on preemption/priority

PLIC Access Control

- PLIC registers are memory mapped, platform-specific
- M-mode-only access to interrupt enables and priorities
- Lower privilege modes only access claim, completion, and threshold registers
  - can only signal completion for inputs for which they're enabled

SiFive Freedom Platform PLIC Mapping

Address	Description
0x4000_0000	*Reserved*
0x4000_0004	source 1 priority
0x4000_0008	source 2 priority
...	
0x4000_0FFC	source 1023 priority
0x4000_1000	Start of pending array
...	(read-only)
0x4000_107C	End of pending array
0x4000_1800	
...	*Reserved*
0x4000_1FFF	
0x4000_2000	target 0 enables
0x4000_2080	target 1 enables
...	
0x401E_FF80	target 15871 enables
0x401F_0000	
...	*Reserved*
0x401F_FFFC	
0x4020_0000	target 0 priority threshold
0x4020_0004	target 0 claim/complete
0x4020_1000	target 1 priority threshold
0x4020_1004	target 1 claim/complete
...	
0x43FF_F000	target 15871 priority threshold
0x43FF_F004	target 15871 claim/complete

Machine-mode only

Target per page to simplify protection

Interrupt/Trap Vectors

- By default, single entry point per privilege level:
  - **`mtvec/htvec/stvec/utvec`**
- Useful in many systems where common handling code used, with bulk of work scheduled later
  - "Interrupt is data"
- Can optionally add differentiated entry points per trap type for embedded applications
  - "Interrupt is control"

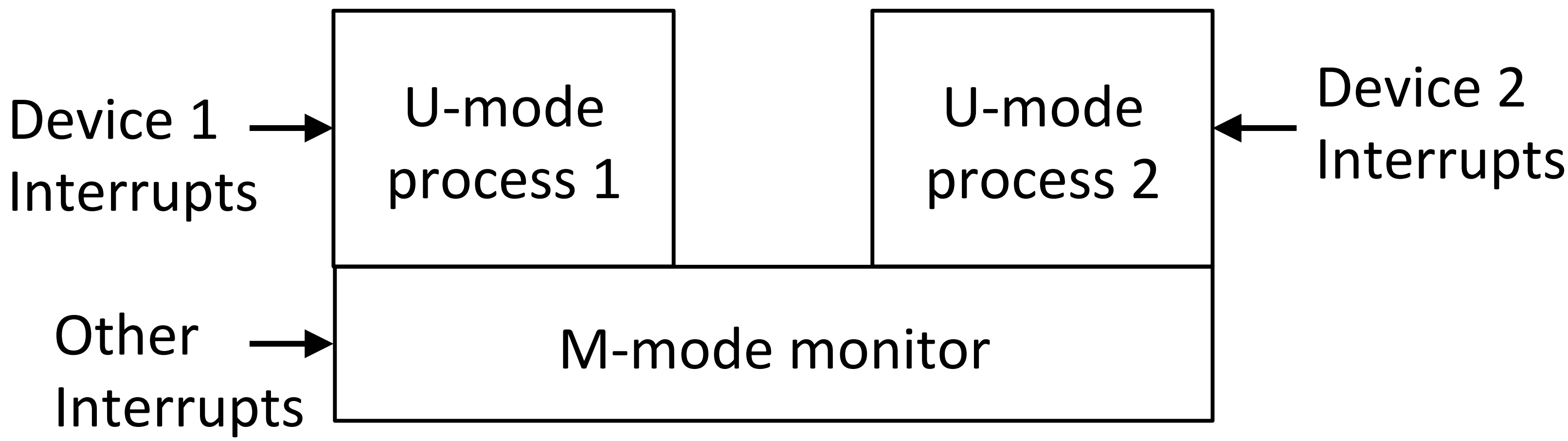
User-Level Interrupts "N"

- Natural extension of interrupt model into user permissions
- Adds user CSRs, and **`uret`** instruction

Number	Privilege	Name	Description
			User Trap Setup
0x000	URW	`ustatus`	User status register.
0x004	URW	`uie`	User interrupt-enable register.
0x005	URW	`utvec`	User trap handler base address.
			User Trap Handling
0x040	URW	`uscratch`	Scratch register for user trap handlers.
0x041	URW	`uepc`	User exception program counter.
0x042	URW	`ucause`	User trap cause.
0x043	URW	`ubadaddr`	User bad address.
0x044	URW	`uip`	User interrupt pending.

Interrupts in Secure Embedded Systems (M, U modes)

- M-mode runs secure boot and runtime monitor
- Embedded code runs in U-mode
- Physical memory protection on U-mode accesses
- Interrupt handling can be delegated to U-mode code
- Provides arbitrary number of isolated subsystems



Questions?