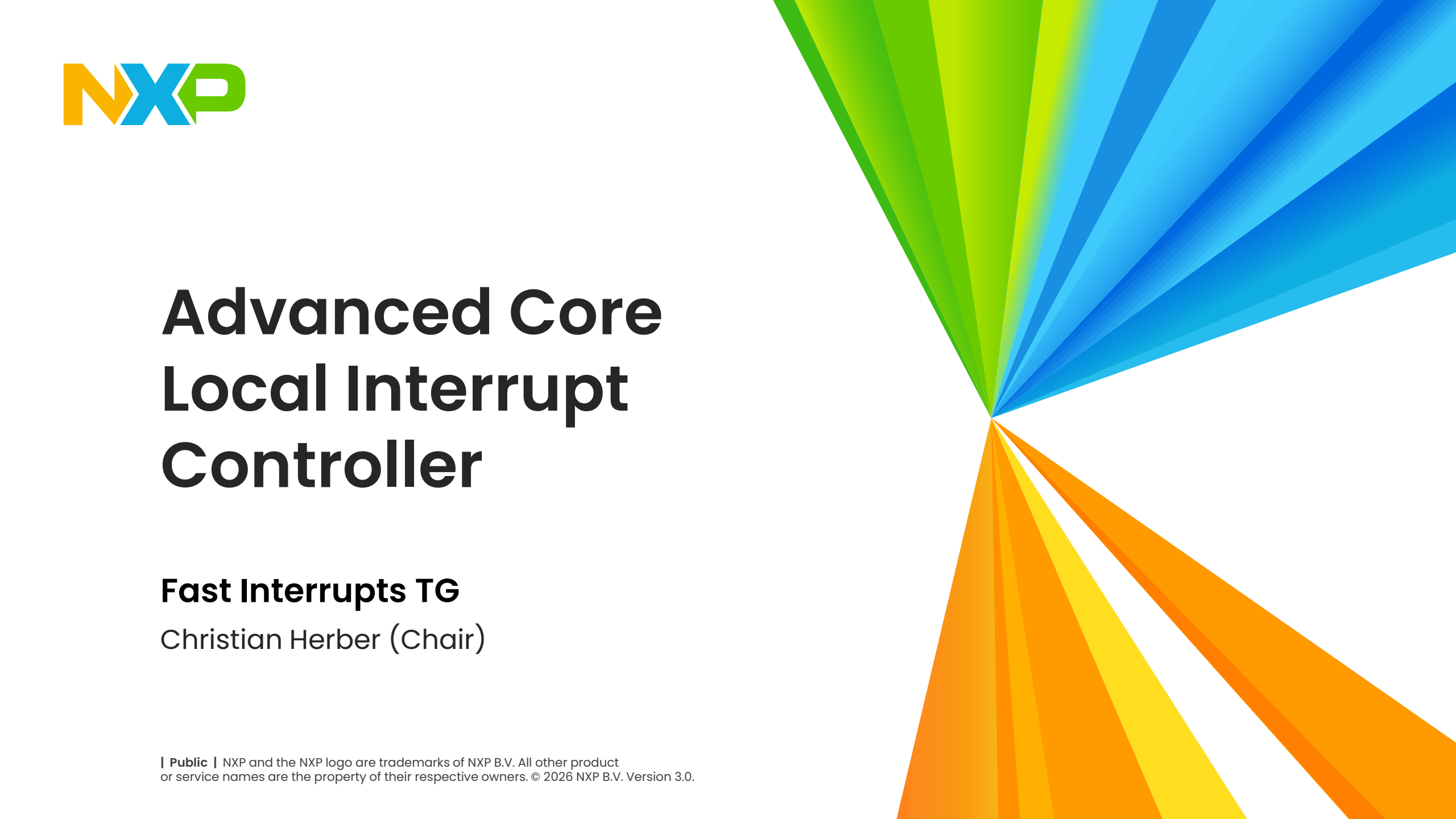


Advanced Core Local Interrupt Controller

Fast Interrupts TG

Christian Herber (Chair)

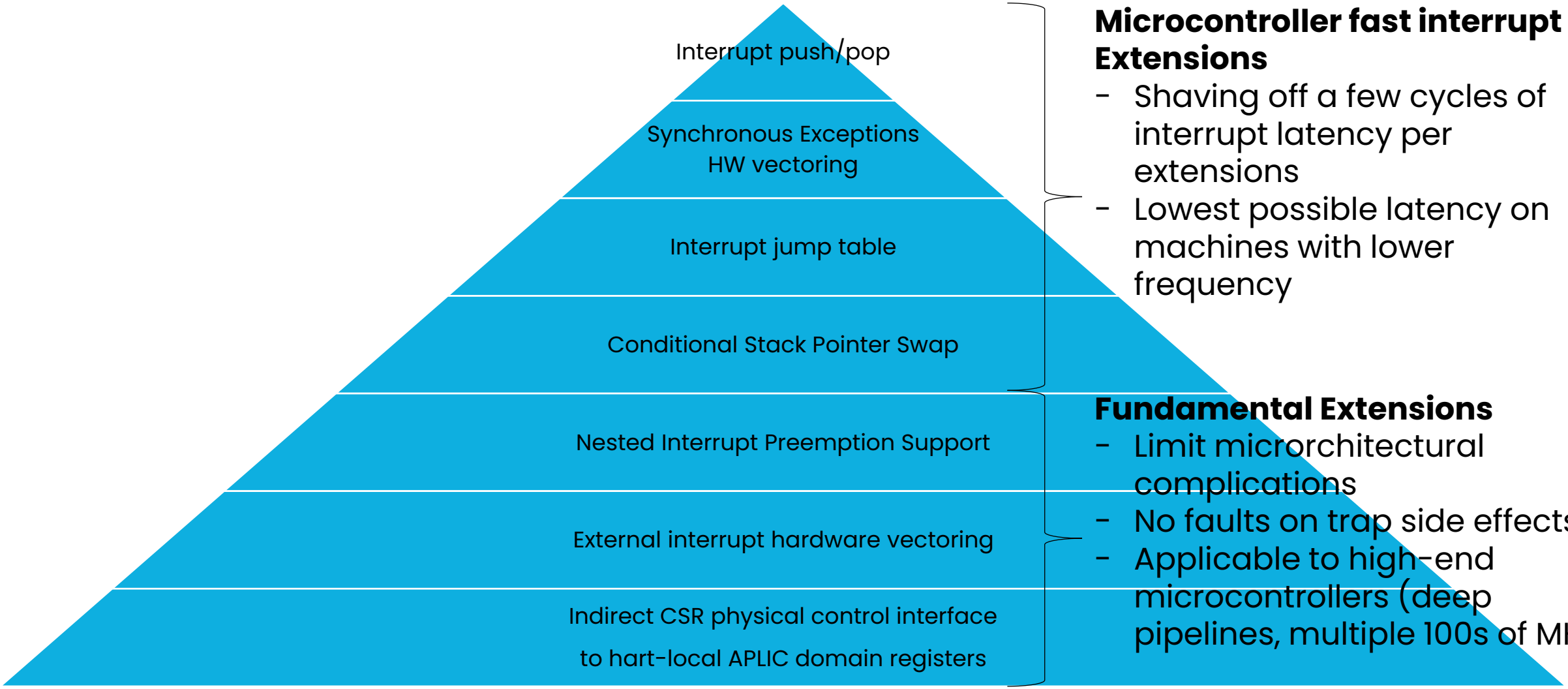
Fast Interrupts TG Charter

*Develop a low-latency, vectored, priority-based, preemptive interrupt scheme for interrupts directed to a **single hart**, compatible with the existing RISC-V standards. Provide both hardware specifications and software ABIs/APIs. Standardize compiler conventions for annotating interrupt handler functions.*

Additionally, since AIA was ratified during the lifetime of Fast Interrupts TG, strive for maximum alignment with AIA.

Hypervisor support is **not** in scope.

ACLIC Extensions Overview



Microcontroller fast interrupt Extensions

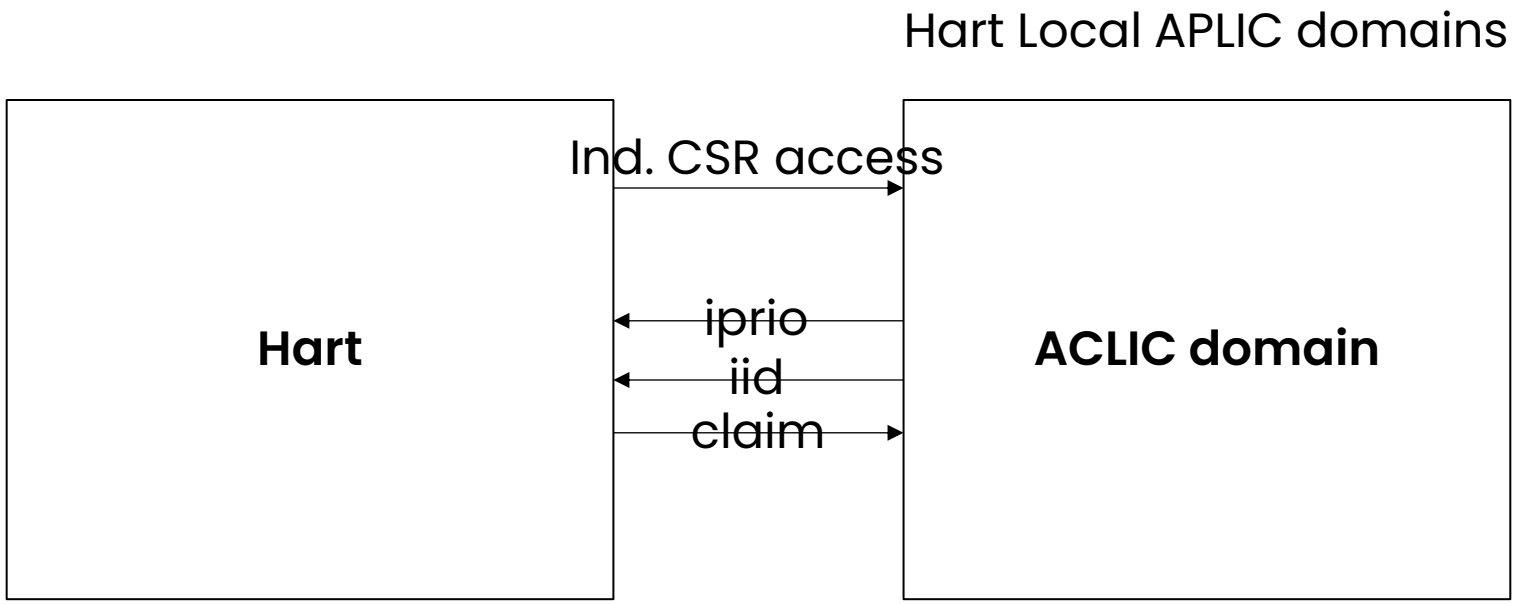
- Shaving off a few cycles of interrupt latency per extensions
- Lowest possible latency on machines with lower frequency

Fundamental Extensions

- Limit microrchitectural complications
- No faults on trap side effects
- Applicable to high-end microcontrollers (deep pipelines, multiple 100s of MHz)

ACLIC Architecture

- Architecture of an M/U system (M/S/U system would have two APLIC domains)



Simplified APLIC domain:
- Fixed to direct delivery mode
- Hart is not configurable
- Threshold not needed (superseded)
- Trivial delegation (fixed child index)
- No IDC structures

Changelog

- Smidctl:
  - Simplified by removing topei
  - Added mtopsi, reporting a signed interrupt identity, merging major and external interrupts
- Smnip:
  - mtopsi is extended to control threshold
- Smihv: New extension for vectoring major and external interrupts
- Smijt (formely Smivt):
  - Use negative index for major interrupt to get rid of one base CSR
  - LSB of table entry controls if we take the table jump, or jump to mtvec.base
  - Programmable shamt field to control entry offset in interrupt jump table
  - Changed table access to read instead of fetch; page faults cannot be recovered (use Smihv)
- Smcsps: Full XLEN sized stack pointer swap register
- Smip:
  - Fixed stack frame size to XLEN for guaranteed psABI compliance
  - Stack pointer swap happens before save/after restore

Preemption Model

- Priority
  - Any interrupt needs to exceed a threshold to be considered enabled
  - The threshold is applied quantized, i.e. several interrupts are in one preemption group
    - This reduces the number of nested preemptions → reduces worst case stack
  - If multiple interrupts exceed the threshold, the full prio is evaluated
- On irq trap
  - Threshold is raised to be higher than interrupt that cause the trap
  - Interrupt is claimed (i.e. pending bit is cleared)
  - Previous threshold is latched into mistatus CSR, so it can be restored later
- On mret
  - Previous threshold is restored from mistatus CSR to threshold CSR
- For SW dispatch:
  - New CSR mtopsi sets threshold to value from source register, claims interrupt, claims interrupt, and sets threshold to claimed interrupt

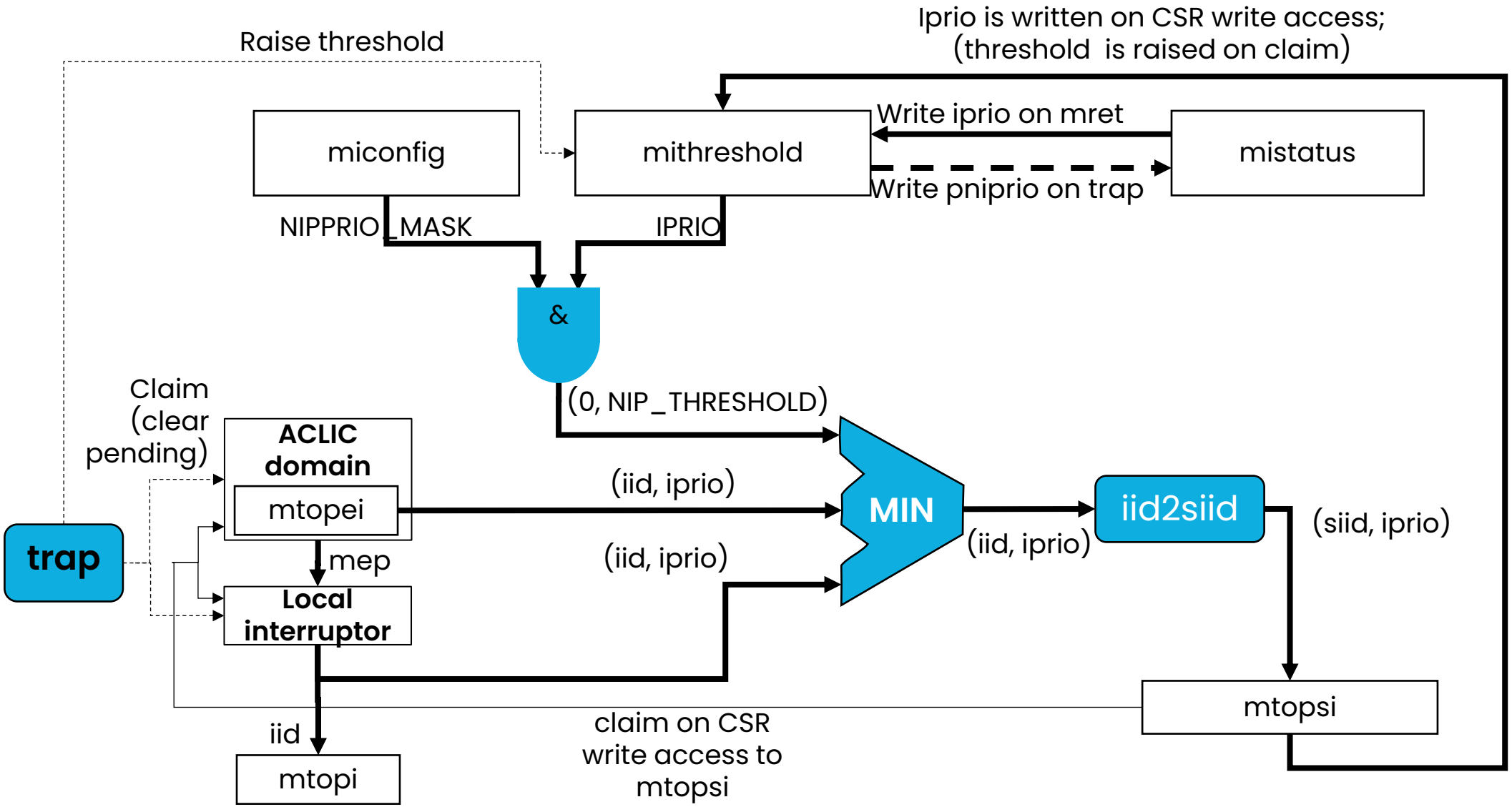
Handler example for SW dispatch

```
interrupt_dispatcher:
    addi    sp, sp, -(4*REGSZ)     # create stack frame
    sr      s0, 0*REGSZ, sp        # save s0 to stack
    csrr    s0, mepc
    sr      s0, 1*REGSZ, sp        # save mepc to stack
    sr      s1, 2*REGSZ, sp        # save s1 to stack
    csrr    s1, mcause
    csrrs   s0, CSR_MISTATUS, 1    # enable interrupts

    call    __riscv_save           # save ra, t0-t6, a0-a7

    slli    a0, s1, P_ALIGN        # create vector table offset
    la      s1, i_handlers + HANDLER_SPLIT

call_handler:
    add     a0, a0, s1
    lr      a0, 0, a0
    jalr    a0

next_handler:
    csrrw   a0, CSR_MTOPSI, s0  # restores threshold from s0[8:0],
  claims interrupt and raises threshold
    srai    a0, a0, (16 - P_ALIGN)

    j       call_handler
```

```
spurious_handler:
dispatch_exit:
    call    __riscv_restore      # restore ra, t0-t6, a0-a7

    lr      s1, 2*REGSZ, sp      # restore s1 from stack

    csrw    CSR_MISTATUS, s0     # disables interrupts
    lr      s0, 1*REGSZ, sp      # restore mepc from stack
    csrw    mepc, s0
    lr      s0, 0*REGSZ, sp      # restore s0 from stack
    addi    sp, sp, (4*REGSZ)    # destroy stack frame


    mret
```

- Minimal overhead during dispatch loop
- Threshold managed by write to mtopsi CSR
- Works with any mtvec.mode setting

Preemption Model (Assuming simplest possible implementation)



Vectoring with External Interrupt Hardware Vectoring Support

Event	Address relative to xtvec[XLEN-1:2]<<2	Table content
Custom IRQ N	-4 * N	j handler_irq_n
…	…	
MTI	-4 * 7	j interrupt_dispatcher
STI	-4 * 5	j interrupt_dispatcher
MSI	-4 * 3	j interrupt_dispatcher
SSI	-4 * 1	j interrupt_dispatcher
Sync. Exception	0	j exception_handler
Ext IRQ 1	4 * 1	j handler_ext_1
Ext IRQ 2	4 * 2	j interrupt_dispatcher
…	…	
Ext IRQ N-1	4 * (N-1)	j handler_ext_n-1

1

```
Interrupt_dispatcher:
  ...
  la s0, vector_base
  ...
  add     a0, siid, s0
  lr      a0, 0, a0
  jalr    a0
  ...
  mret
```

2

3

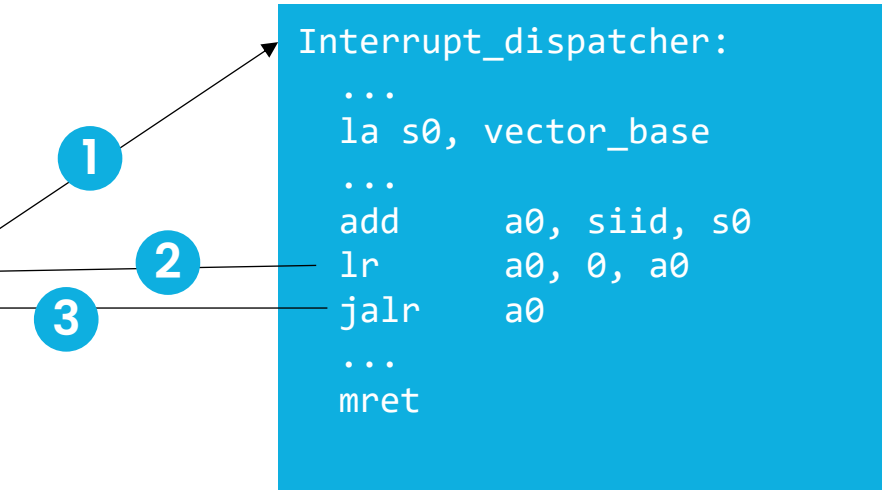
Address relative to vector_base	Table content
-4 * N	-
…	
-4 * 7	handler_mti
-4 * 5	handler_sti
-4 * 3	handler_msi
-4 * 1	handler_ssi
0	no_irq
4 * 1	-
4 * 2	handler_ext_2
…	
4 * (N-1)	handler_ext_n-1

Offset multiplier 8 for pointer width 64

- One table for major and external interrupts, where major interrupts iids are negated for table accesses
- A SW dispatcher can index into a second table to make an ABI-compliant call to handler
- Some RTOSs use two entries per identity. The second contains an argument to the handler

Vectoring with Instruction Jump Table (Smijt/Ssijt)

Event	Address relative to xijt[XLEN-1:6]<<(2+0)	Table content {[:1], 0}	Table content [0] (VEN)
Custom IRQ N	-4 * N	handler_irq_n	1
…	…		
MTI	-4 * 7	handler_mti	0
STI	-4 * 5	handler_sti	0
MSI	-4 * 3	handler_msi	0
SSI	-4 * 1	handler_ssi	0
No irq	0	no_irq	0
Ext IRQ 1	4 * 1	handler_ext_1	1
Ext IRQ 2	4 * 2	interrupt_dispatcher	0
…	…		
Ext IRQ N-1	4 * (N-1)	handler_ext_n-1	1



- Hardware and SW vectoring mechanism share on table → smaller memory footprint
- Configurable shift-amount allows ABI-specific dimensioning of table and/or storing handler arguments in the table (e.g. done in Zephyr)

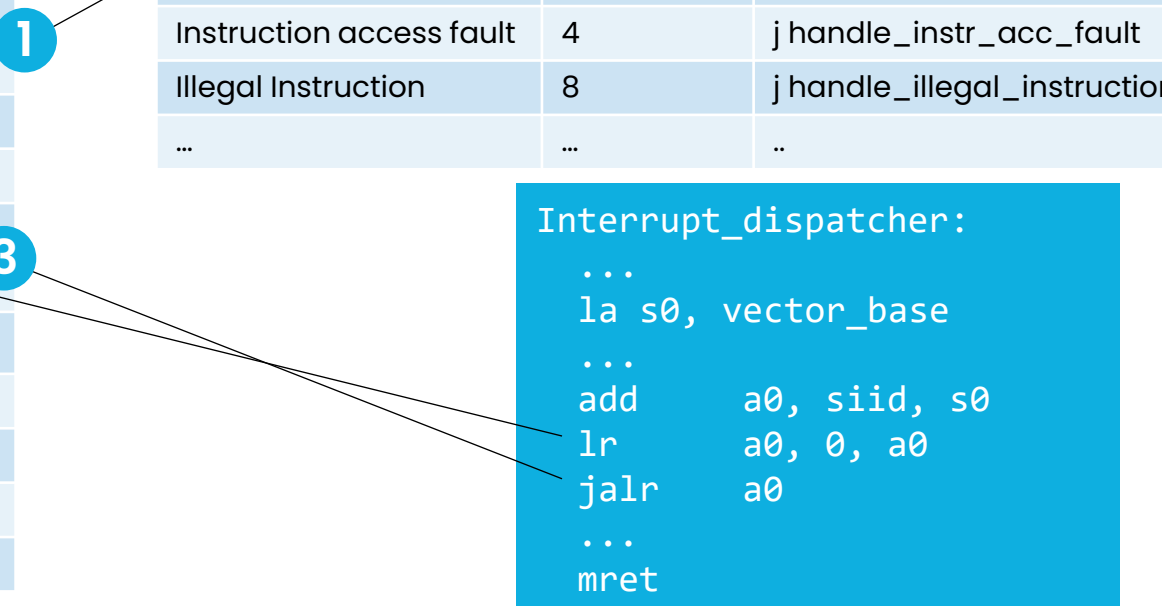
Vectoring with Instruction Jump Table (Smijt/Ssijt) and exception hardware vectoring (Smehv/Ssehv)

Interrupt Jump Table

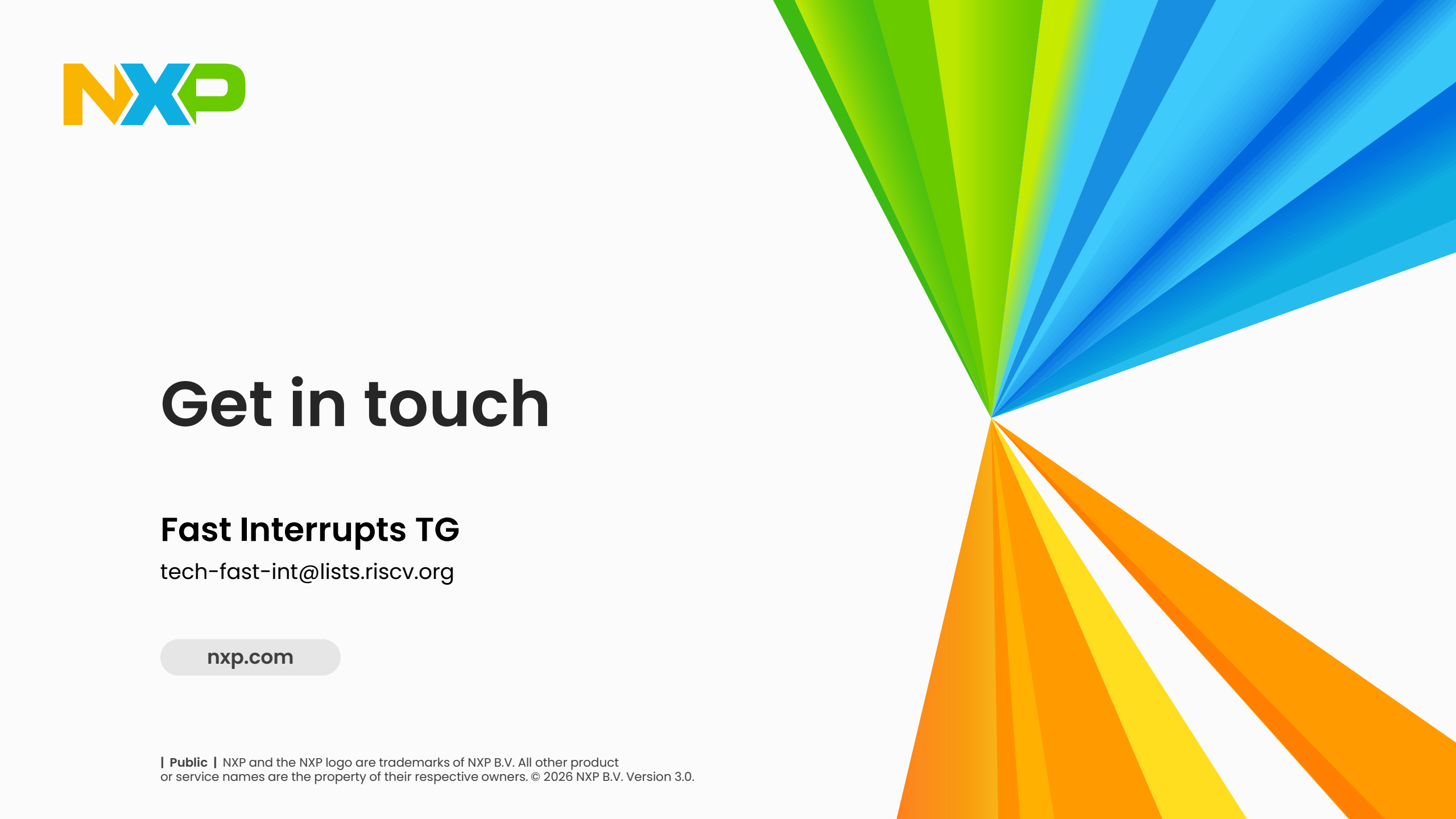
Event	Address relative to xijt[XLEN-1:6]<<(2+0)	Table content {[:1], 0}	Table content [0] (VEN)
Custom IRQ N	-4 * N	handler_irq_n	1
…	…		
MTI	-4 * 7	handler_mti	0
STI	-4 * 5	handler_sti	0
MSI	-4 * 3	handler_msi	0
SSI	-4 * 1	handler_ssi	0
No irq	0	no_irq	0
Ext IRQ 1	4 * 1	handler_ext_1	1
Ext IRQ 2	4 * 2	interrupt_dispatcher	0
…	…		
Ext IRQ N-1	4 * (N-1)	handler_ext_n-1	1

Exception Vector Table

Event	Address relative to xtvec.base	Table content {[:1], 0}
Irq VEN=0	0	j Interrupt_dispatcher
Instruction access fault	4	j handle_instr_acc_fault
Illegal Instruction	8	j handle_illegal_instruction
…	…	..



- Same benefits as Smivt only, plus faster exception handling
- Exception Code 0 is reserved when IALIGN=16 (misaligned fetch), so that addr=xtvec.base can be used for jumping to SW dispatcher



Get in touch

Fast Interrupts TG

tech-fast-int@lists.riscv.org

nxp.com

Brighter
Together

nxp.com