Interrupt-Driven Input/Output

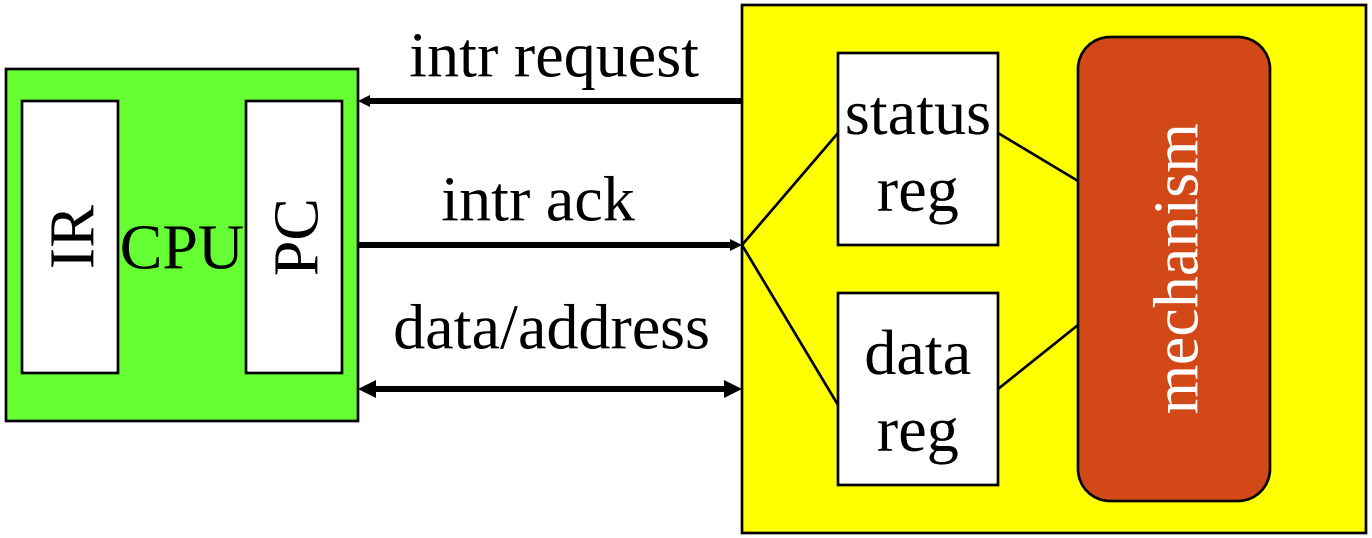
Chapter 3: Section 3.2.4

ARM & Cortex-M4 User Manuals

Interrupt I/O

- Busy/wait is very inefficient.
  - CPU can't do other work while testing device.
  - Hard to do simultaneous I/O.
- Interrupts allow a device to change the flow of control in the CPU.
  - Causes subroutine call to handle device.

Interrupt interface



Interrupt behavior

- Based on subroutine call mechanism.
- Interrupt forces next instruction to be a "subroutine call" to a predetermined location.
  - Return address is saved to resume executing foreground program.
  - "Context" switched to interrupt service routine

Interrupt physical interface

- CPU and device are connected by CPU bus.
- CPU and device handshake:
  - device asserts interrupt request;
  - CPU asserts interrupt acknowledge when it can handle the interrupt.

(See ARM interrupt support)

Example: interrupt-driven main program

```
main() {
  while (TRUE) {
      if (gotchar) { //set by intr routine
        OUT_DATA = achar; //write char
        OUT_STATUS = 1;    //set status
        gotchar = FALSE;   //reset flag
        }
      }
      other processing….
  }
```

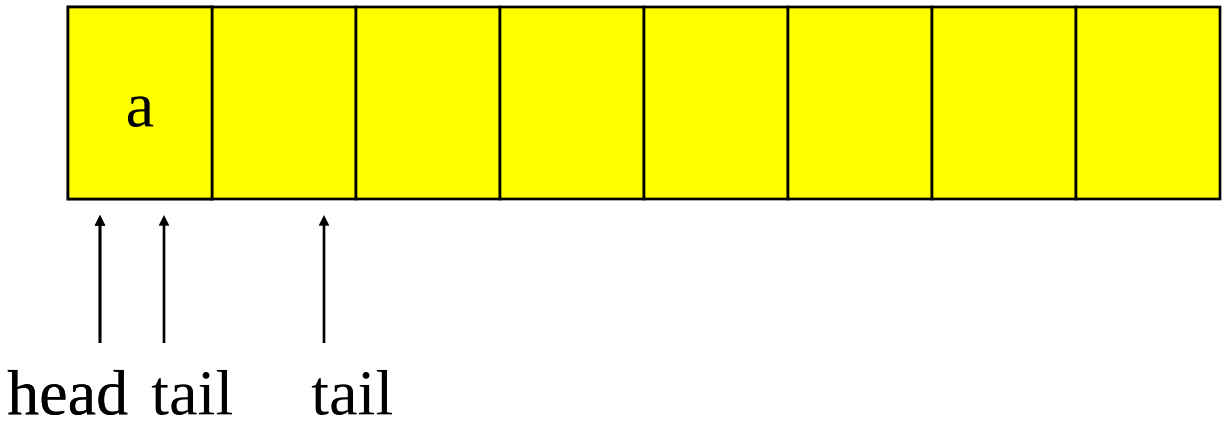
Example: character I/O handlers

```
#define IN_DATA (*((volatile unsigned byte *) 0xE0028018))
#define IN_STATUS (*((volatile unsigned byte *) 0xE002801C))

void input_handler() {
  achar = IN_DATA;          //global variable
  gotchar = TRUE;           //signal main prog
  IN_STATUS = 0;            //reset status
}
void output_handler() {
} //interrupt signals char done
```

Example: interrupt I/O with buffers

- Queue for characters:



leave one empty slot
to allow full buffer to
be detected

Buffer-based input handler

```
void input_handler() {
  char achar;
  if (full_buffer()) error = 1;
  else {
      achar = IN_DATA;          //read char
      add_char(achar); }        //add to queue
  IN_STATUS = 0;                //reset status
  if (nchars >= 1) {            //buffer empty?
       OUT_DATA = remove_char();
       OUT_STATUS = 1; }
}  //above needed to initiate output
```

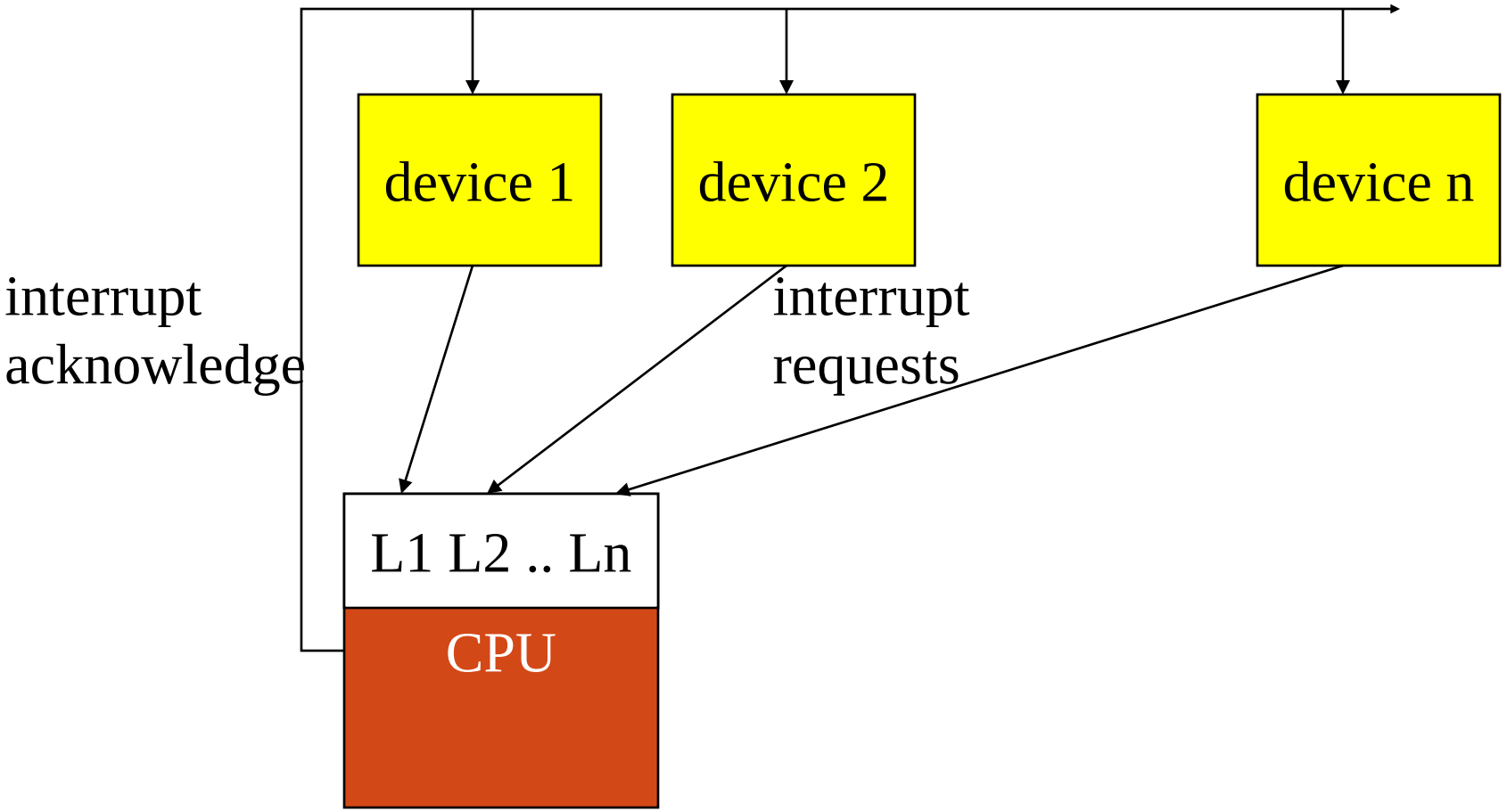
Interrupts vs. Subroutines

- CPU checks interrupt lines between instructions
- Interrupt handler starting address:
  - fixed in some microcontrollers
  - usually provided as a pointer
- CPU saves its "state", to be restored by the interrupt handler
  - Push items on a stack
  - And/Or: Save items in special registers
- Handler should save any other registers that it may use

Priorities and vectors

- Two mechanisms allow us to make interrupts more specific:
  - Priorities determine what interrupt gets CPU first.
  - Vectors determine what code is called for each type of interrupt.
- Mechanisms are orthogonal: most CPUs provide both.

Prioritized interrupts

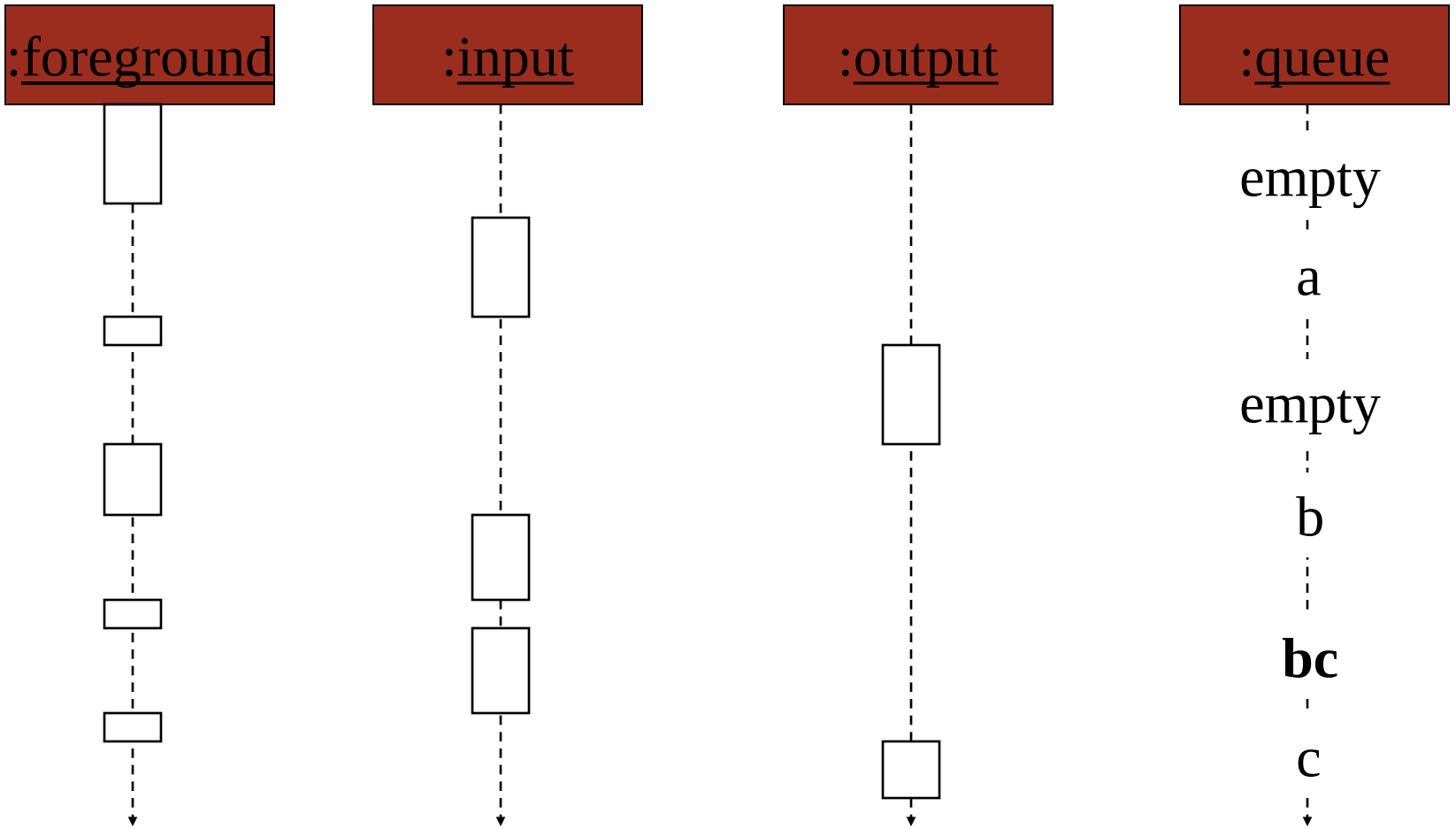


Interrupt prioritization

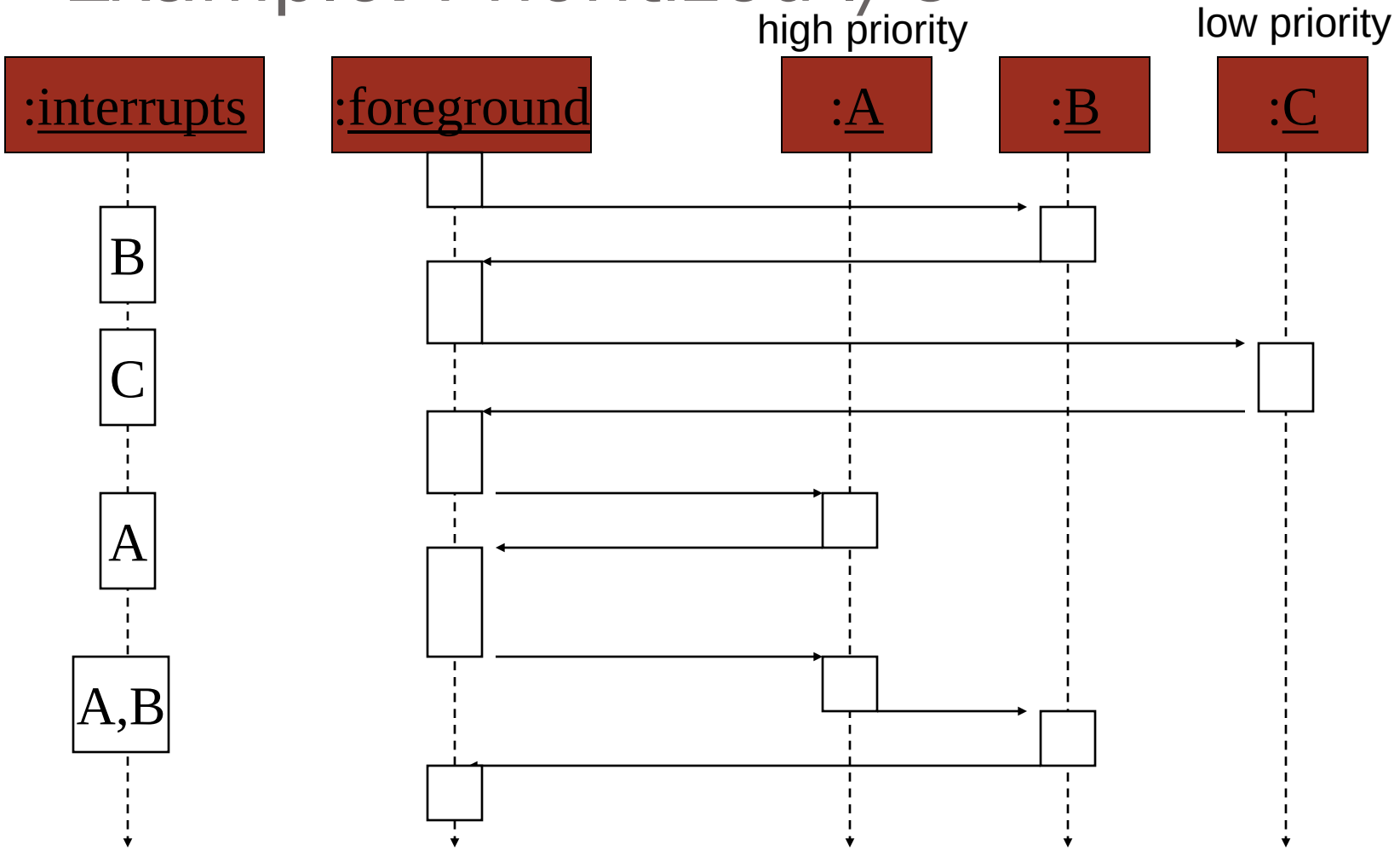
- Masking: interrupt with priority lower than current priority is not recognized until pending interrupt is complete.
- Non-maskable interrupt (NMI): highest-priority, never masked.
  - Often used for power-down.
- Handler may choose to "enable" other interrupts (allows handler to be preempted)

- CPU may also have bit(s) in its status register to enable or mask interrupt requests.

I/O sequence diagram

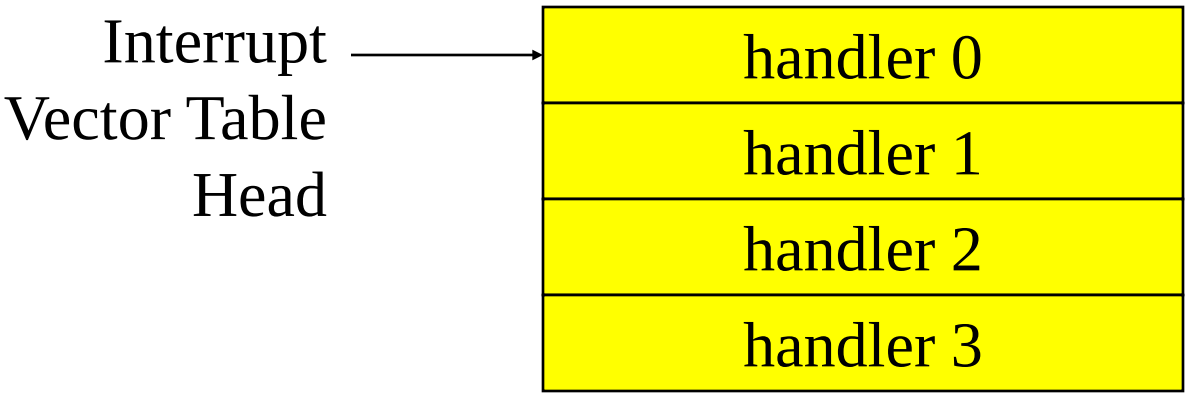


Example: Prioritized I/O

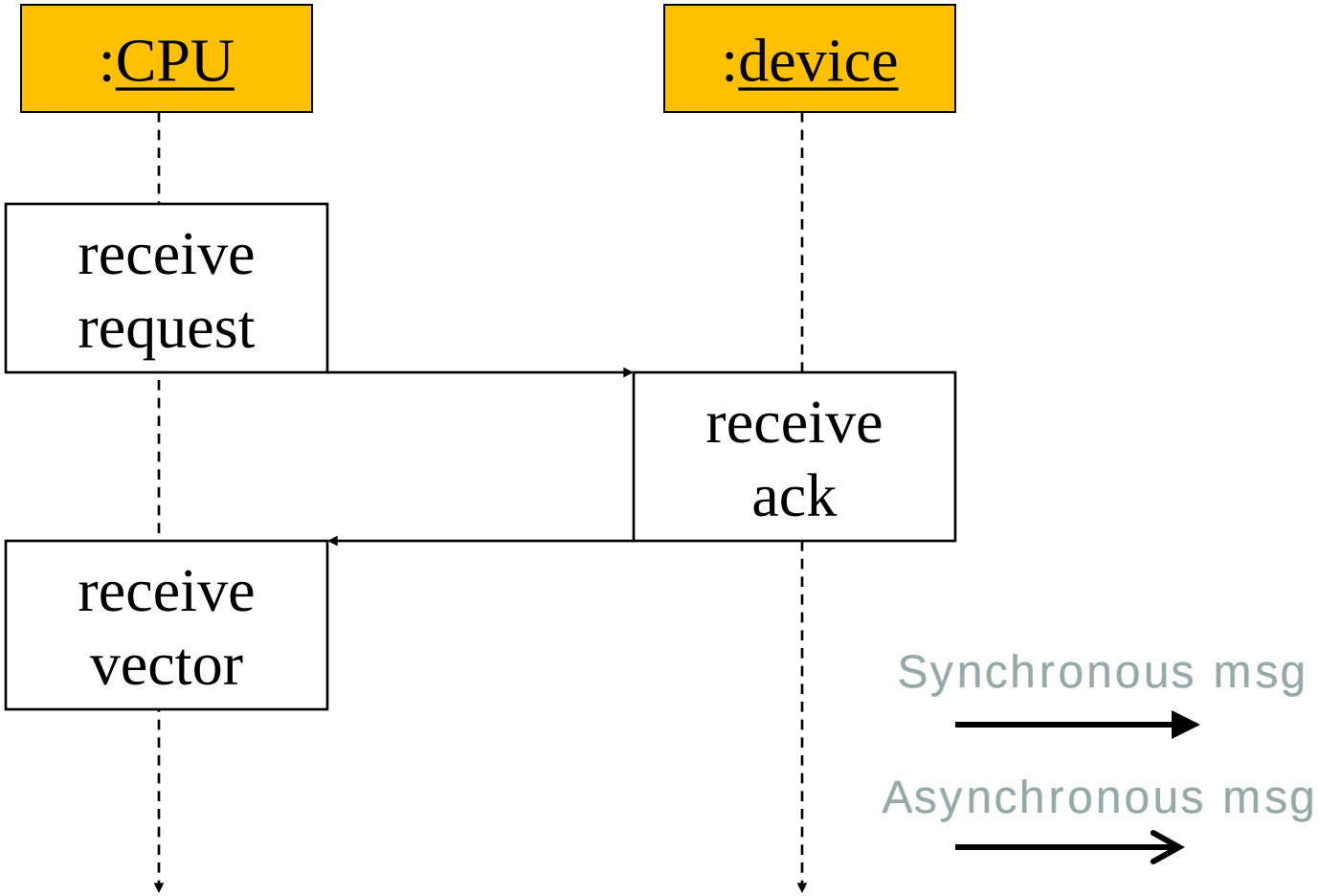


Interrupt vectors

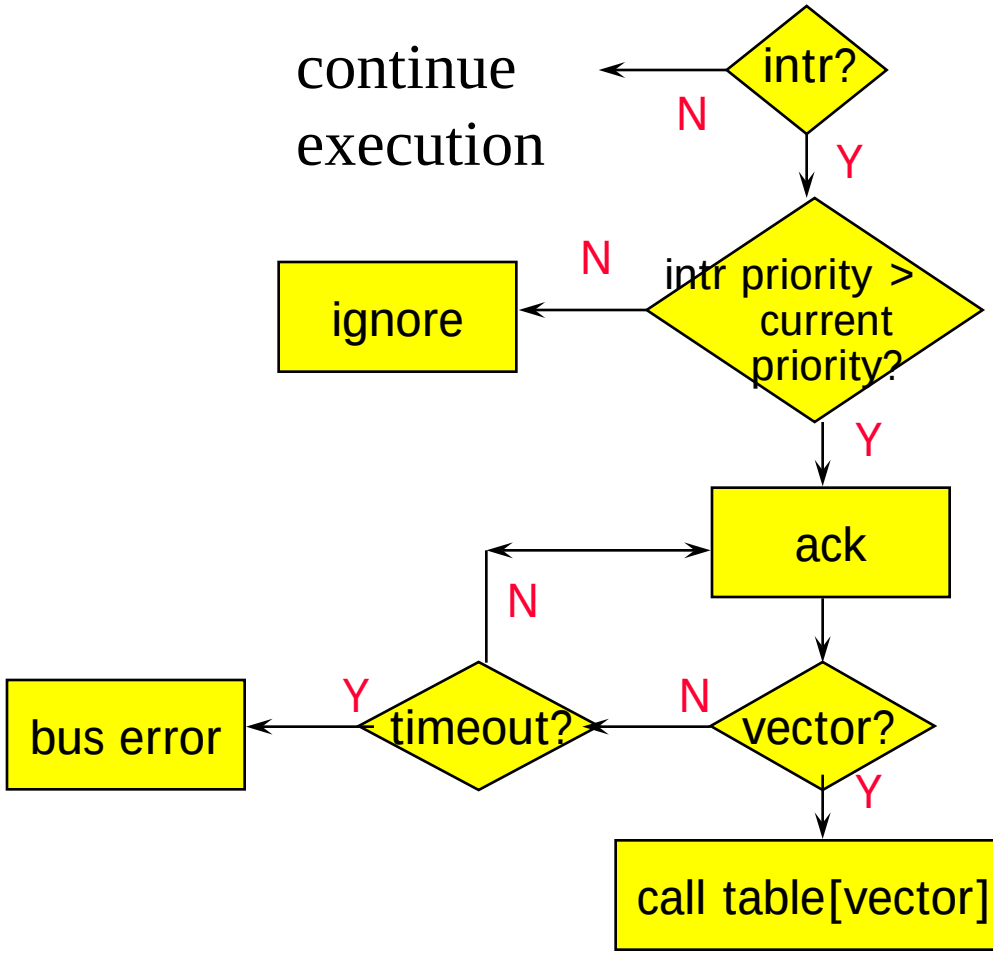
- Allow different devices to be handled by different code.
- Interrupt vector table:
  - Directly supported by CPU architecture and/or
  - Supported by a separate interrupt-support device/function



Interrupt vector acquisition



Generic interrupt mechanism



Assume priority selection is handled before this point.

CPU detects interrupt request

CPU acknowledges the request

Device sends vector

CPU calls handler & Software processes the request

Sources of interrupt overhead

- Handler execution time.
- Interrupt mechanism overhead.
- Register save/restore.
- Pipeline-related penalties.
- Cache-related penalties.
- Interrupt "latency" = time from activation of interrupt signal until event serviced.
- ARM worst-case latency to respond to interrupt is 27 cycles:
  - 2 cycles to synchronize external request.
  - Up to 20 cycles to complete current instruction.
  - 3 cycles for data abort.
  - 2 cycles to enter interrupt handling state.

Exception

- Exception: internally detected error.

  Example: divide by 0

  ARM: undefined opcode, data abort, memory error
- Exceptions are synchronous with instructions but unpredictable.
- Build exception mechanism on top of interrupt mechanism.
- Exceptions are usually prioritized and vectorized.

Trap

- Trap (software interrupt): an exception generated by an instruction.
  - Ex: Enter supervisor mode.

- ARM uses SWI instruction for traps.
- Cortex uses SVC instruction (supervisor call)