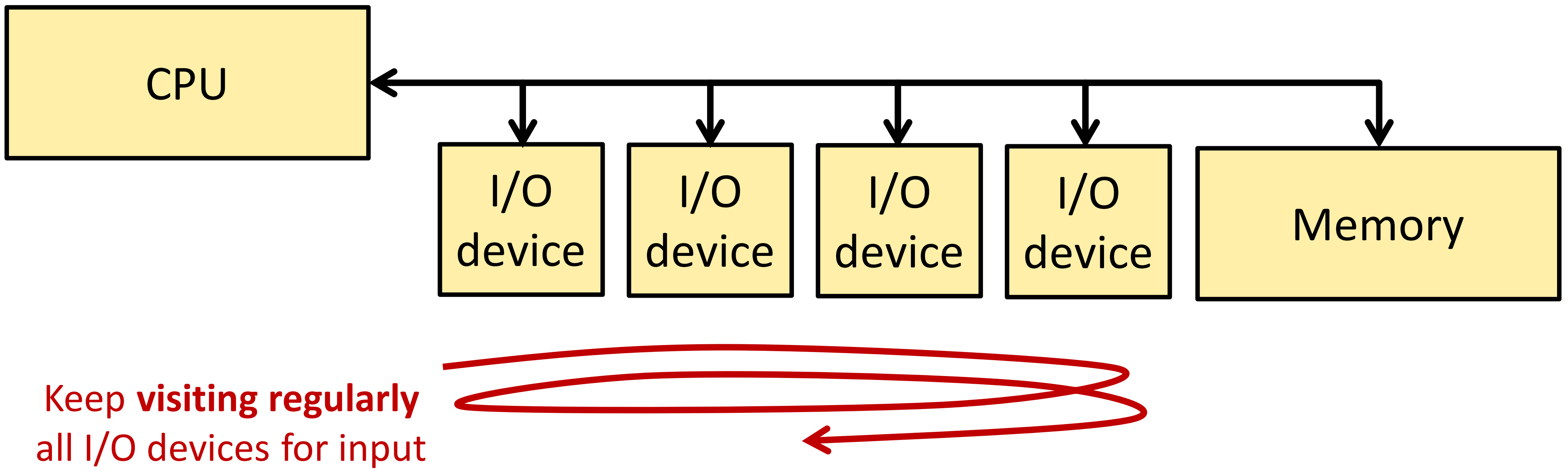
CS-200
Computer Architecture
—
Part 2b. Processor, I/Os, and Exceptions
Interrupts

Paolo Ienne
<paolo.ienne@epfl.ch>

I/O Polling

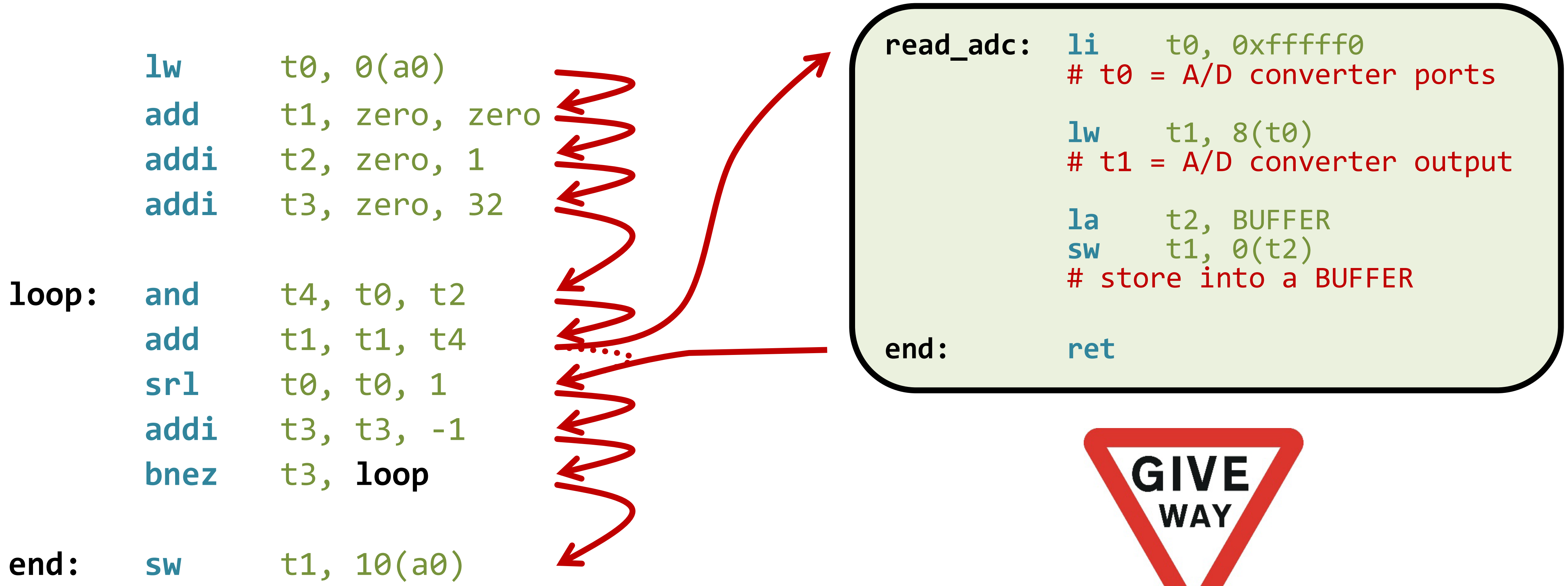
- How do we know if **a peripheral has data** for us (key pressed, packet arrived, etc.)?



- **Very expensive**: if the device is fast and requires immediate action, the processor must spend too much time to check **frequently**

I/O Interrupts

- **Idea**: don’t check peripherals all the time, have them **“ask” for attention**



```
        lw      t0, 0(a0)
        add     t1, zero, zero
        addi    t2, zero, 1
        addi    t3, zero, 32

loop:   and     t4, t0, t2
        add     t1, t1, t4
        srl     t0, t0, 1
        addi    t3, t3, -1
        bnez    t3, loop

end:    sw      t1, 10(a0)
```

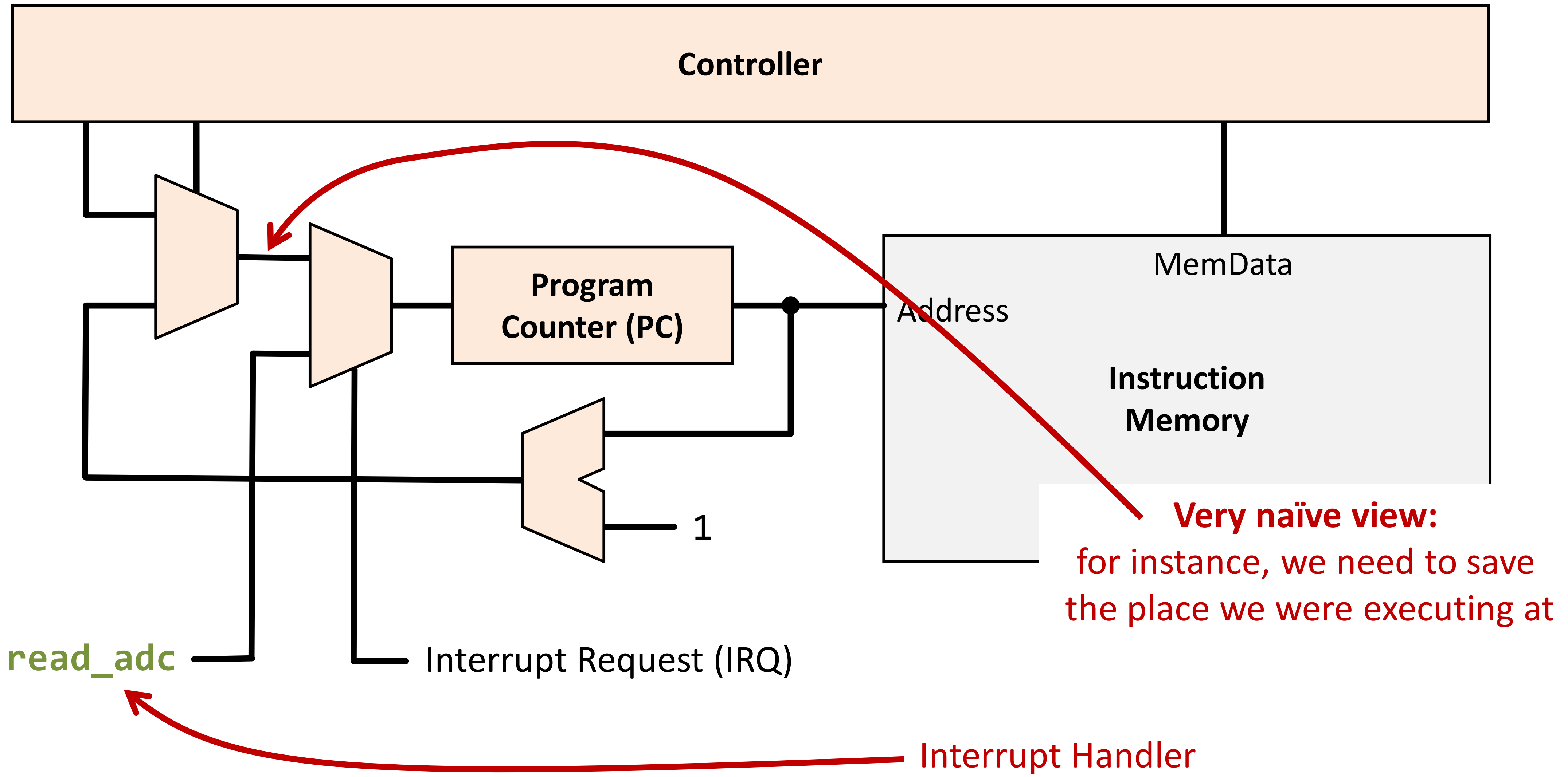
```
read_adc:  li    t0, 0xfffff0
           # t0 = A/D converter ports

           lw    t1, 8(t0)
           # t1 = A/D converter output

           la    t2, BUFFER
           sw    t1, 0(t2)
           # store into a BUFFER

end:       ret
```

Seen This Already in Some Languages?

- Many names for similar mechanisms:
  - **Callbacks**, **Action** or **Event Listeners**, **Signals**, **Promises**, **Futures**, **Hooks**…
- But there are **compilers** and **interpreters** in the picture!
- Not anymore here…

How?!

The Basic Idea of I/O Interrupts



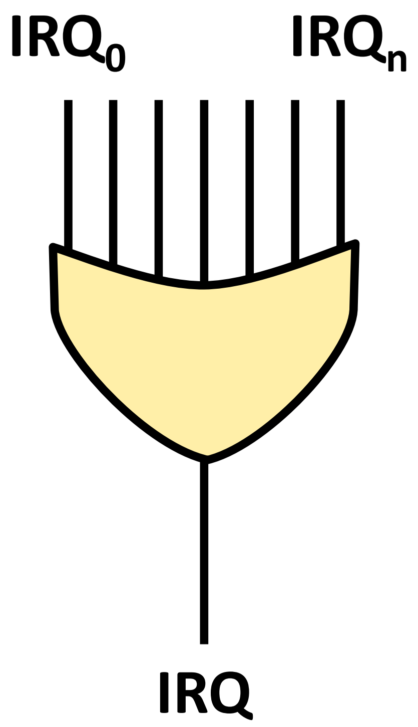
Very naïve view:
for instance, we need to save the place we were executing at

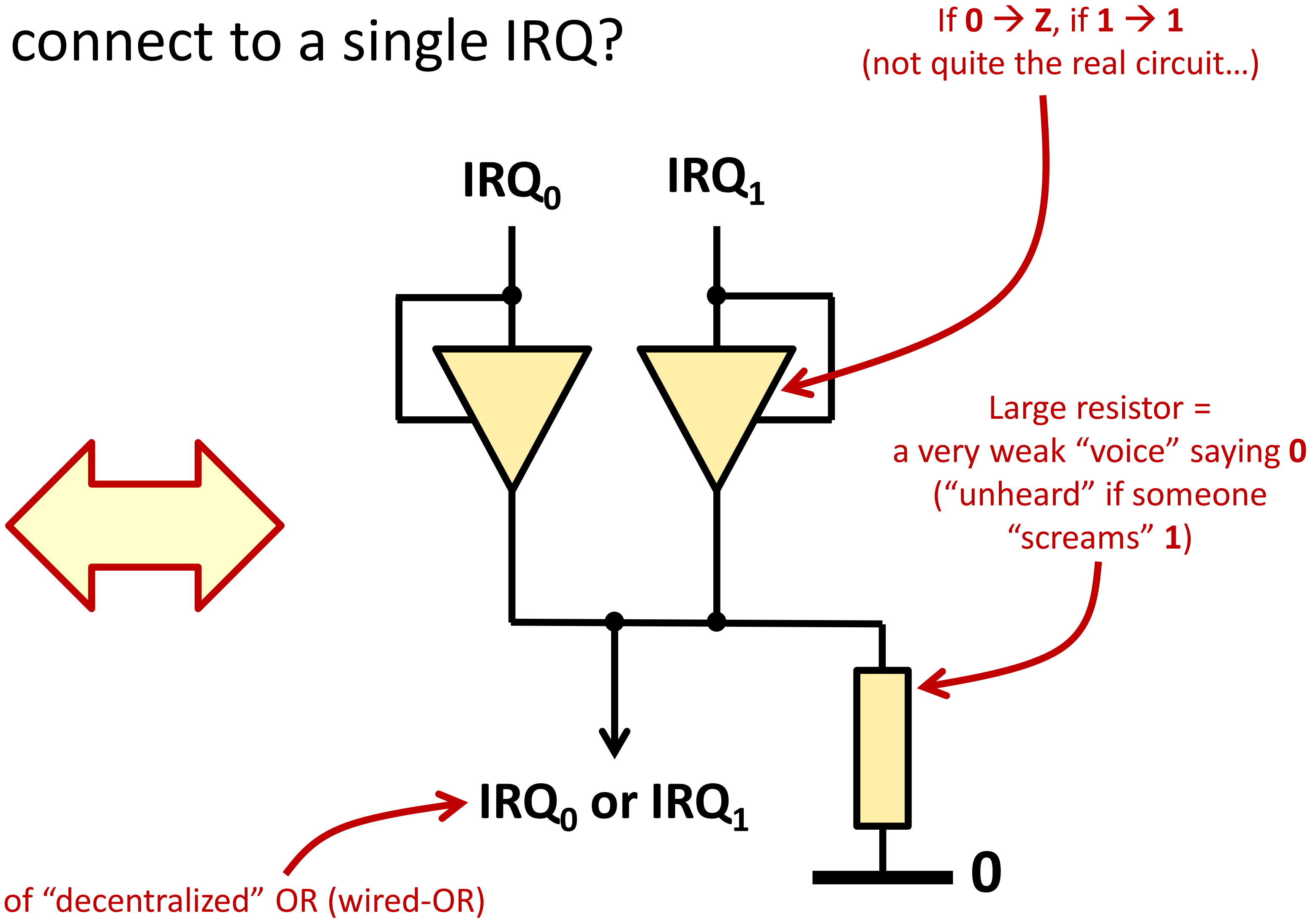
I/O Interrupts

- Several issues to take care of and behaviours to define:
  - Need to know **who needs attention**—we do not have only one peripheral
    - After interrupt**, the software checks all peripherals** in turn (polling), or
    - **I/O peripheral sends identification**
  - **Different priorities** need to be expressed—some peripheral can wait long, some cannot
  - **Impact on current execution**: Current instruction(s) can complete? One? Five? Twenty? What happens of the program that was executing?!

I/O Interrupts

How do many peripherals connect to a single IRQ?





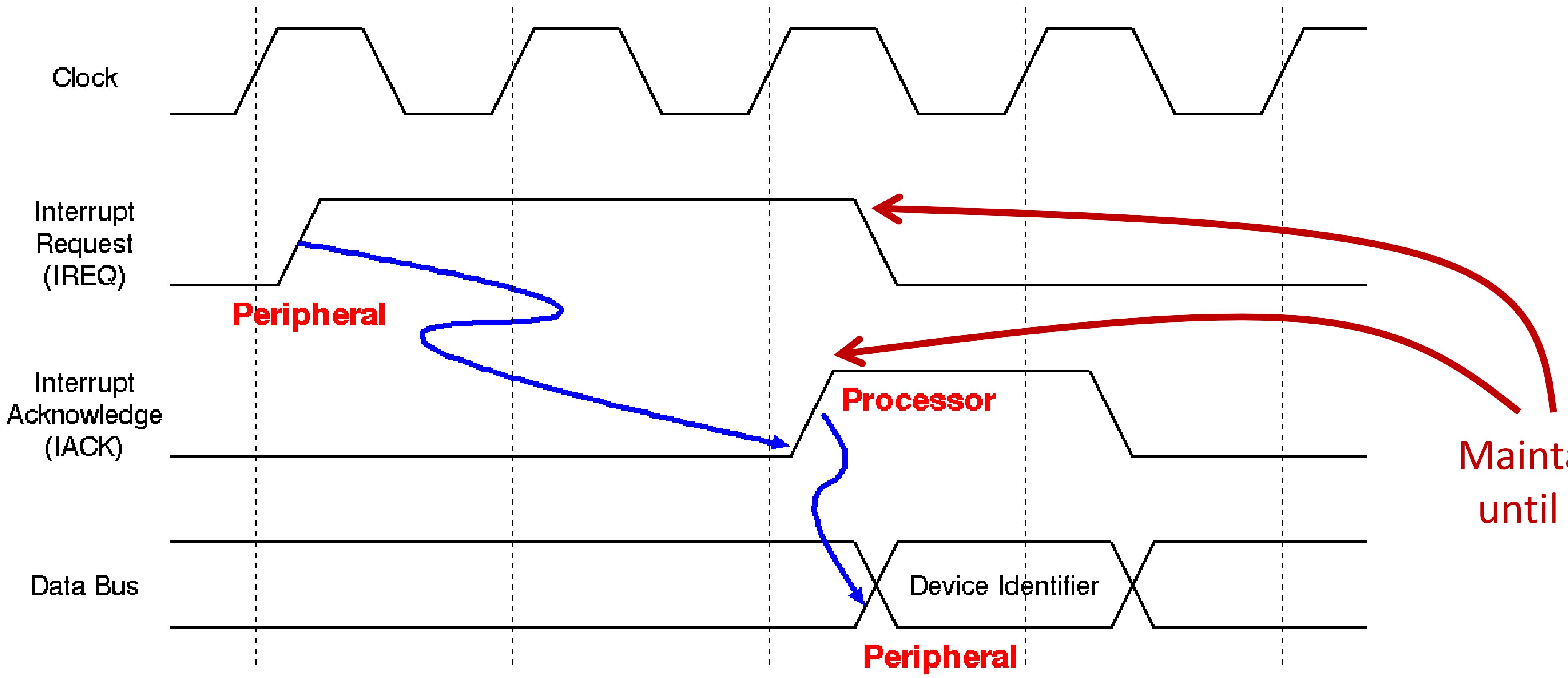
If **0** → **Z**, if **1** → **1**
(not quite the real circuit…)
IRQ_0
IRQ_1
Large resistor =
a very weak "voice" saying **0**
("unheard" if someone
"screams" **1**)
IRQ_0 or IRQ_1
0
A sort of "decentralized" OR (wired-OR)

I/O Interrupts

Example sequence:

1. Peripheral asks for attention through IREQ
2. Processor signals when it is ready to serve the peripheral through IACK (“acknowledges” the interrupt)
3. Peripheral signals its identity
4. Processor takes appropriate action—transfer control to the appropriate Exception Handler
5. Processor reverts to the interrupted task

I/O Interrupts

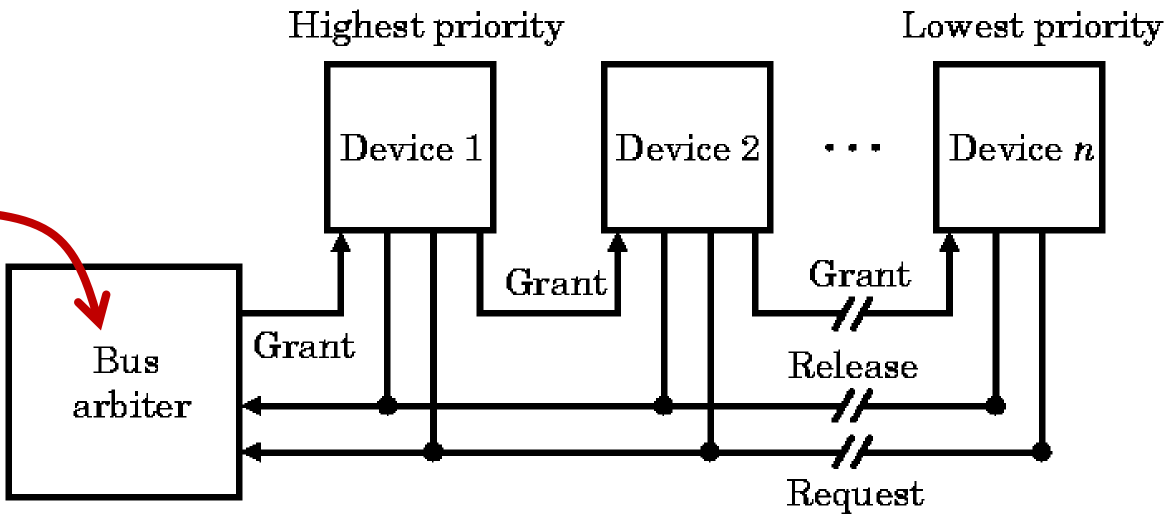


Maintain the request until acknowledged

I/O Interrupt Priorities

- **Daisy Chain Arbitration** is one of the simplest methods
  - Anyone places request (IREQ, Request)
  - Acknowledge line (IACK, Grant) passed from one device to the next
  - Device which wants access, intercepts the signal and hides it from successive devices
  - Simple but (1) slow and (2) hard priorities

The processor or a proxy of the processor

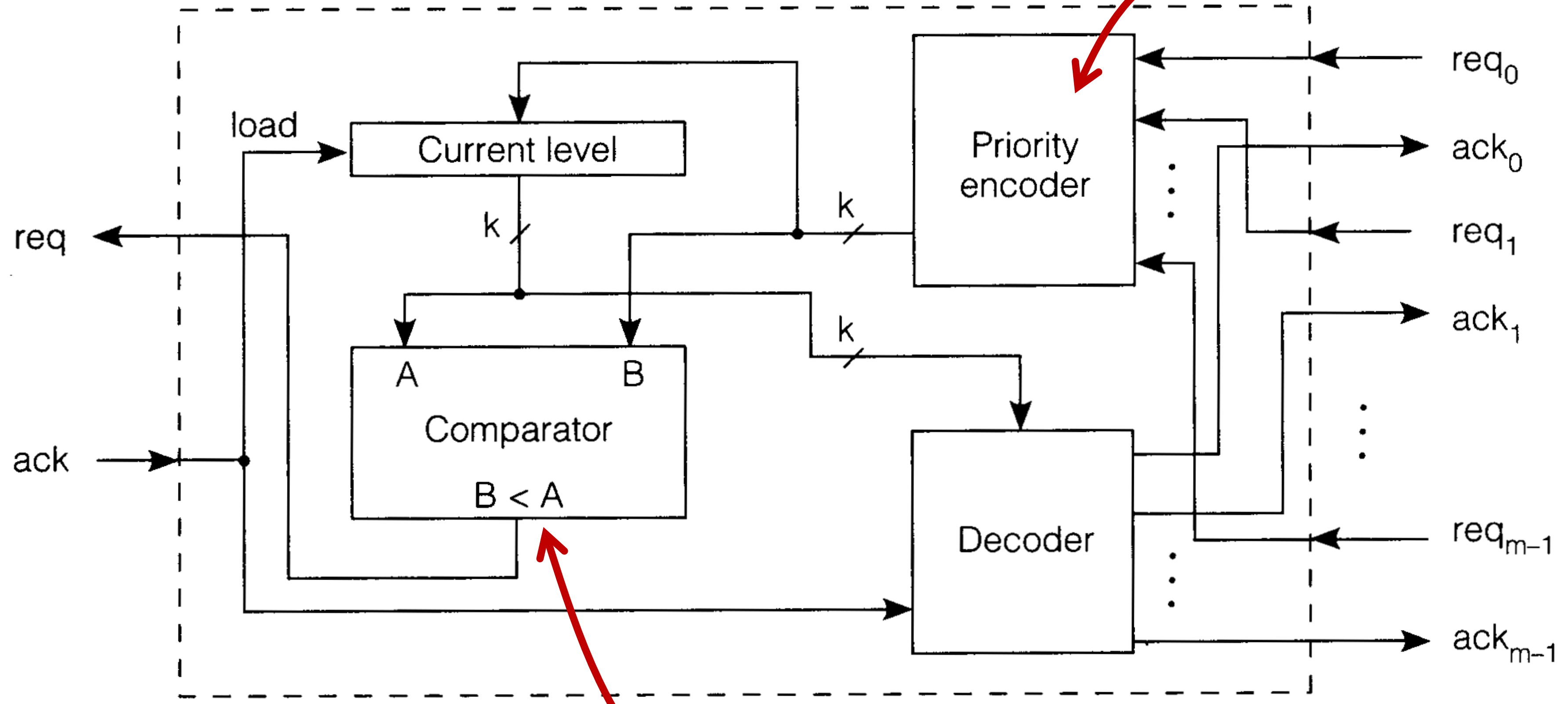


Source: COD, © Morgan Kauffman 1998

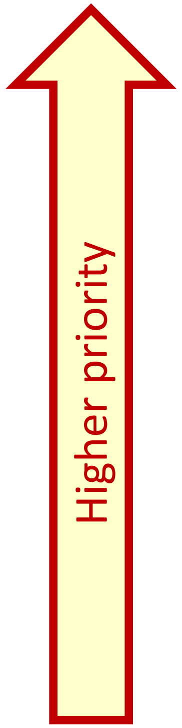
I/O Interrupt Priorities

- More sophisticated methods involve special hardware
- An **interrupt controller** may be expected to
  - Propagate only one IREQ at a time to the processor
    - Select the one with highest fixed priority
    - Select the one with equal priority which has been served last
  - Propagate the returned IACK to the appropriate peripheral
  - Inhibit certain devices from sending IREQs
  - Allow nesting, that is higher priority IREQ to propagate while lower priority interrupts are being served

I/O Interrupt Controller



Output is the **index** of the **lowest** active request



Nested interrupts:
if the new interrupt has higher priority it is propagated to the processor

Source: Heuring & Jordan, © Addison Wesley 1997

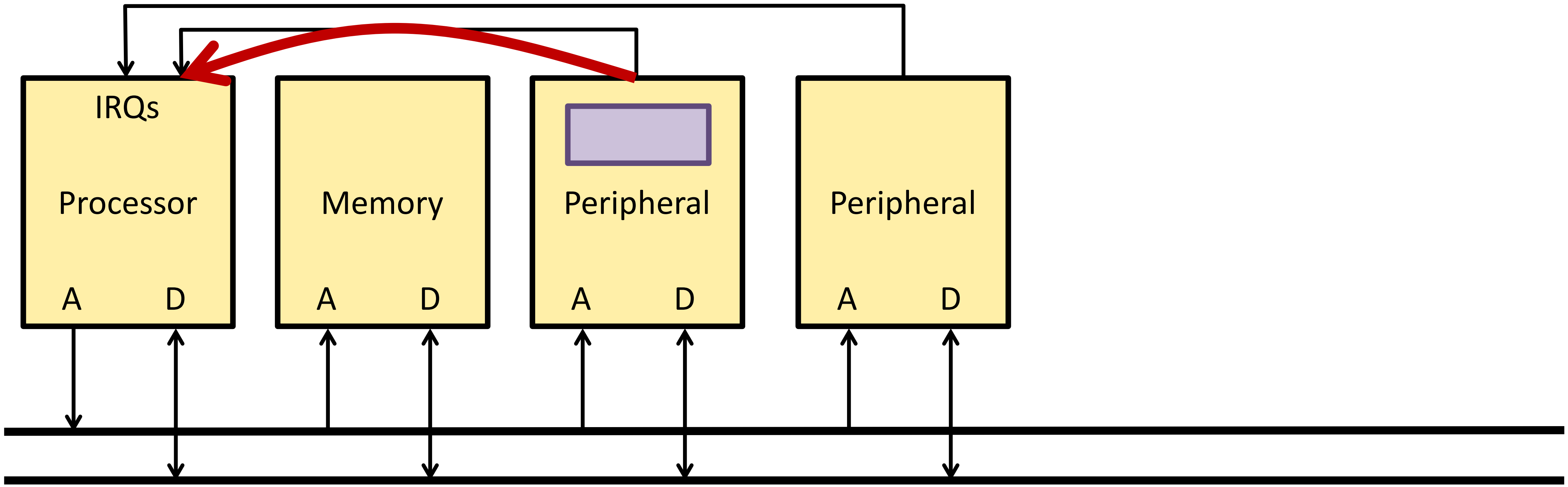
Direct Memory Access (DMA)

- Interrupts save the processor from continuously **checking the I/O devices**
- Yet, the processor may still waste a large share of its time **transferring large chunks of data** to and from high-throughput peripherals (e.g., disks, network)
- **Idea**: let's have a **special peripheral** perform the needed data transfers from and to memory (R/W) and free the processor to continue computation

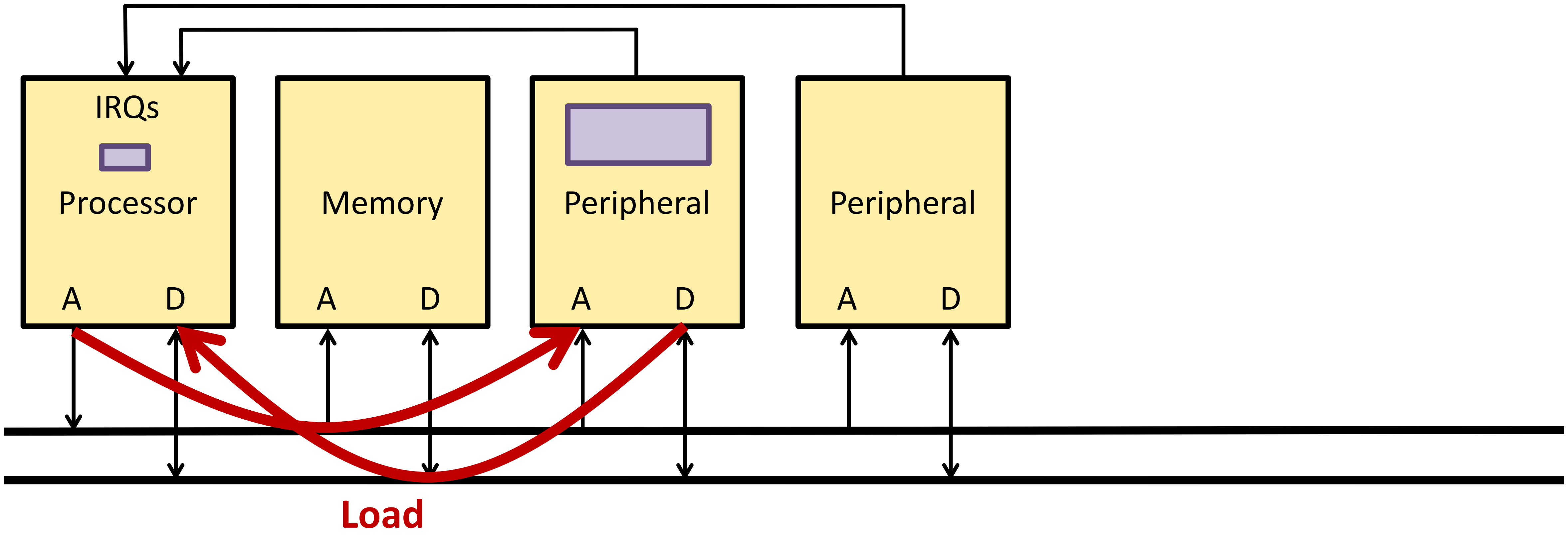
Direct Memory Access

Direct Memory Access (DMA)

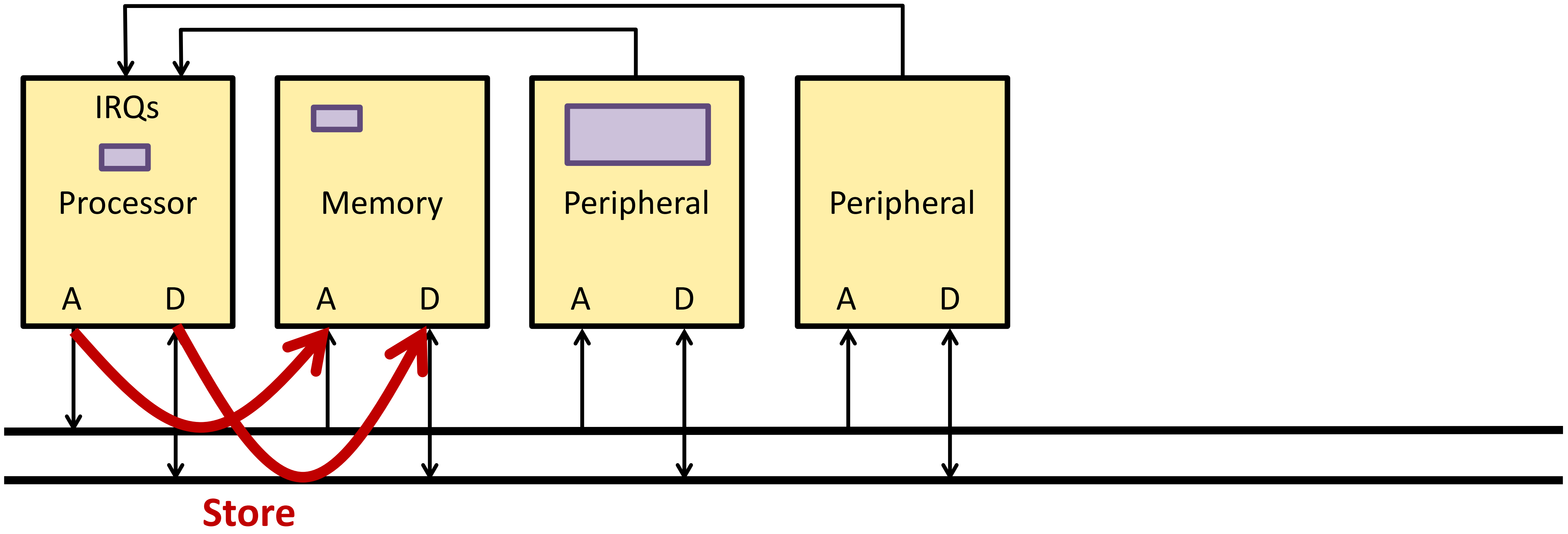
Interrupt request



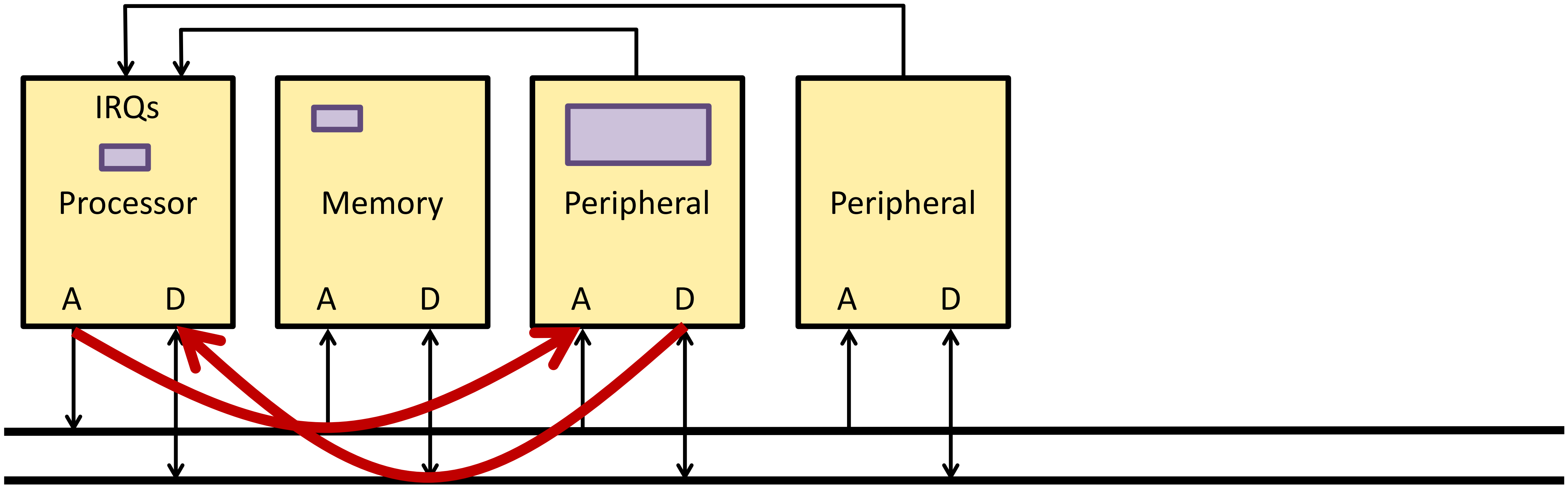
Direct Memory Access (DMA)



Direct Memory Access (DMA)



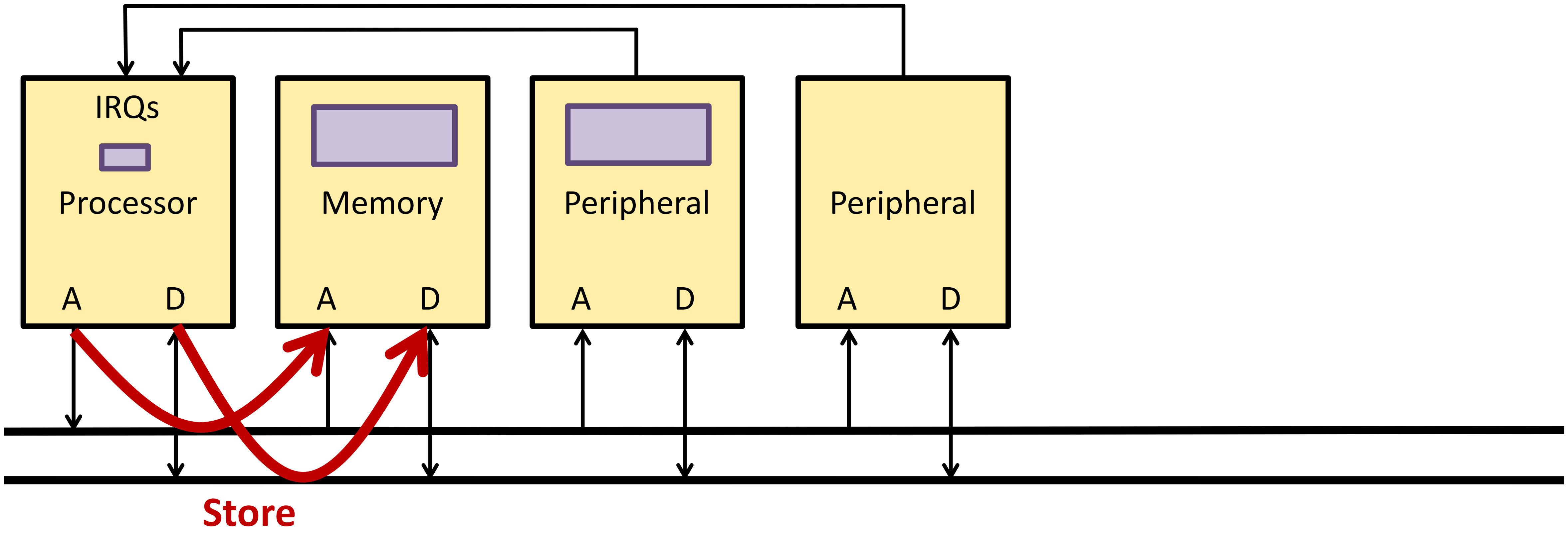
Direct Memory Access (DMA)



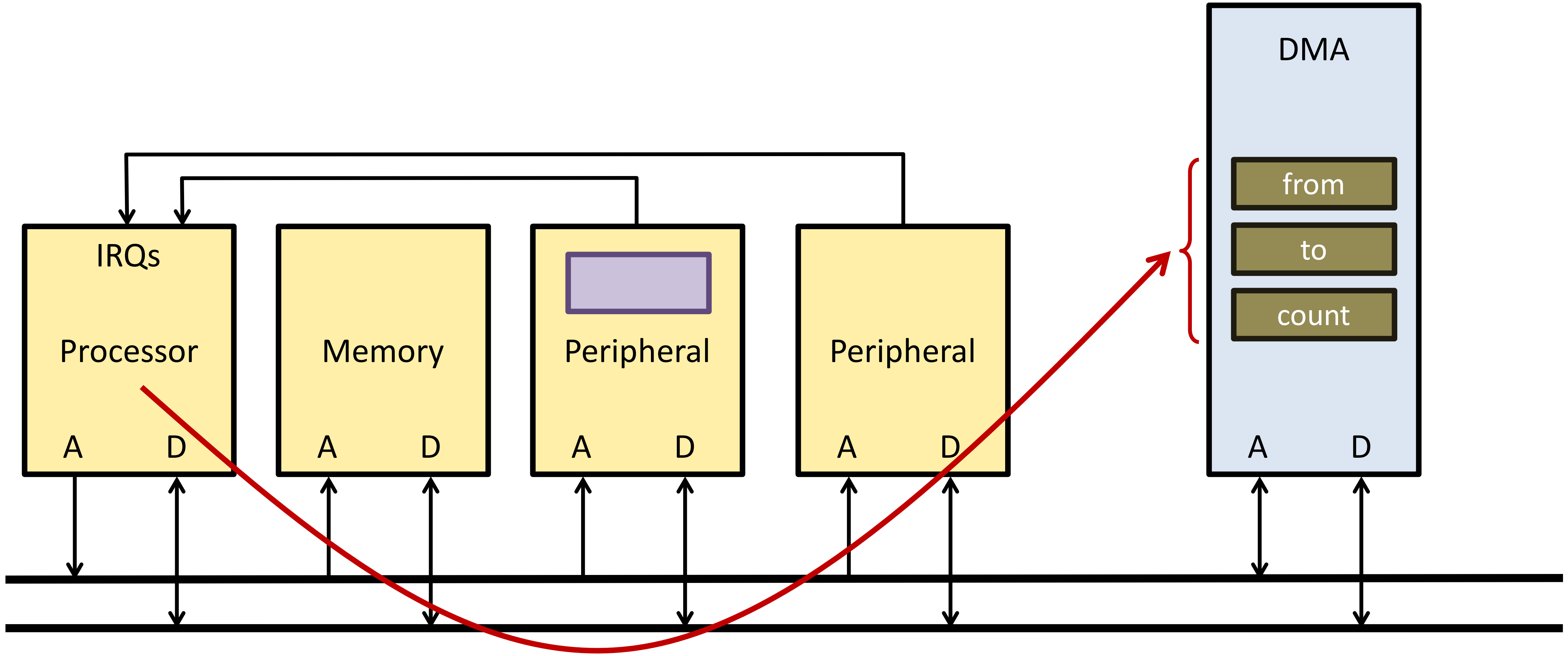
Load

Direct Memory Access (DMA)

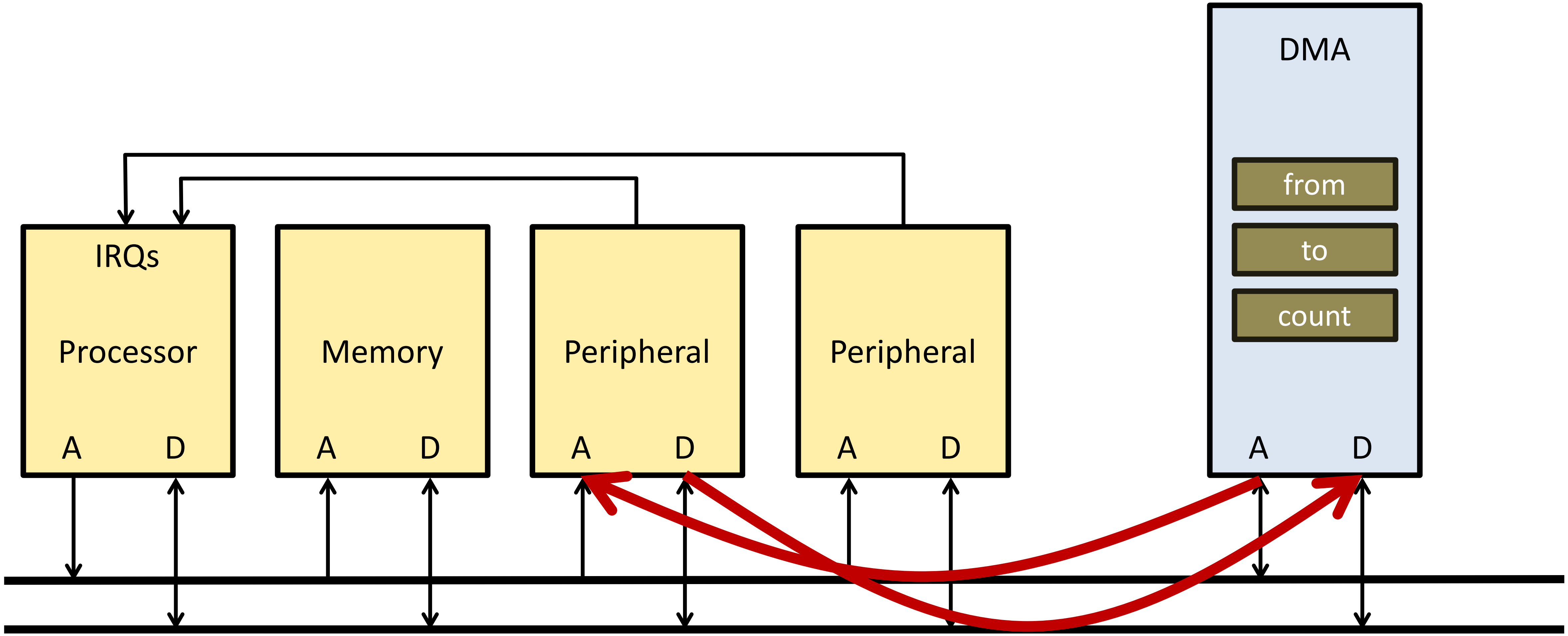
Maybe 1,000,000×…



Direct Memory Access (DMA)



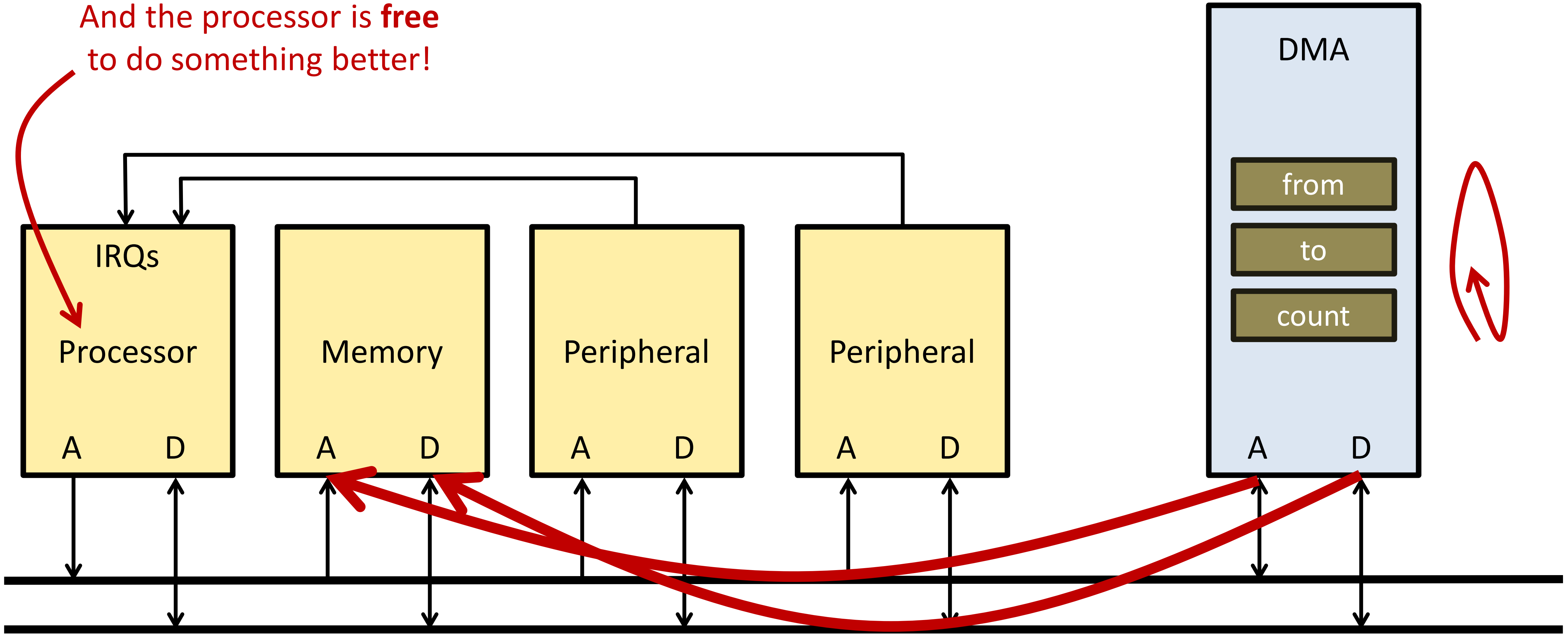
Direct Memory Access (DMA)



"Load" (Read)

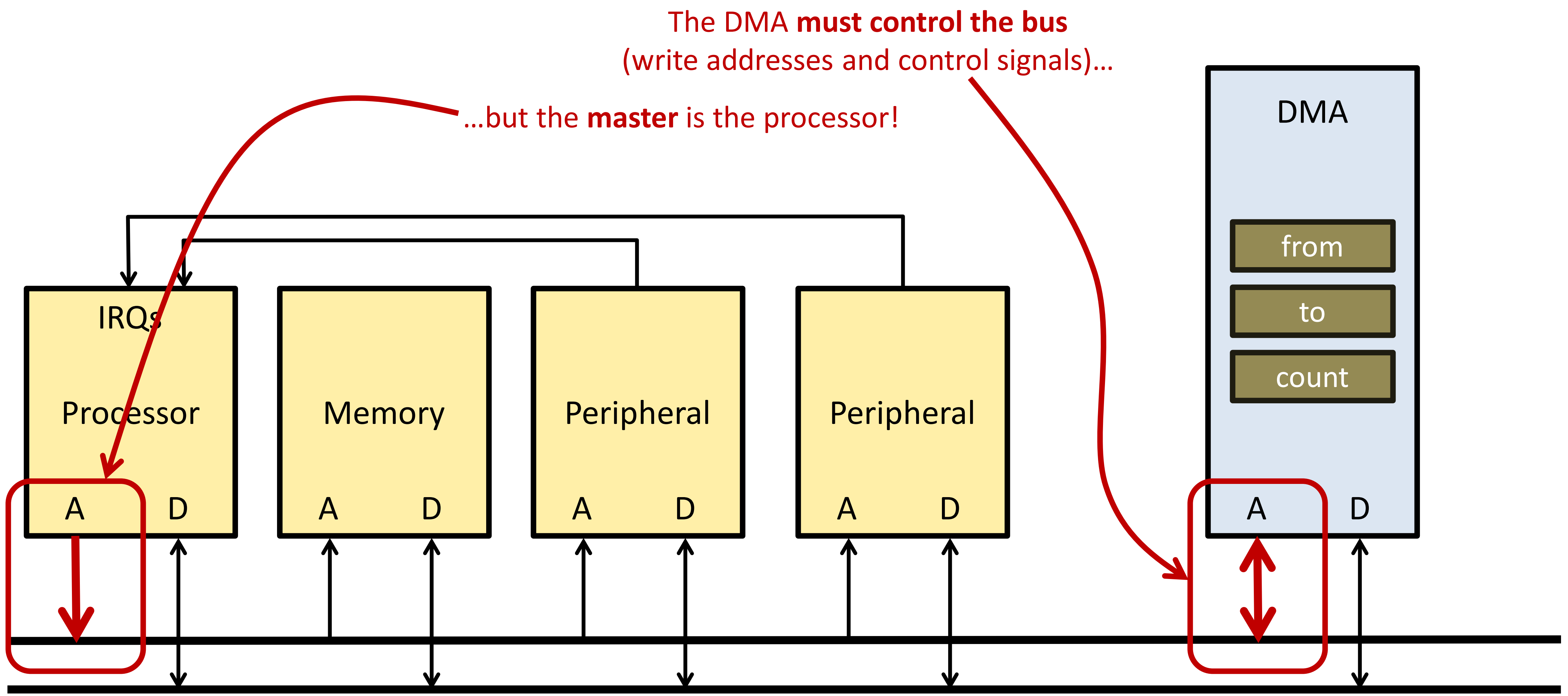
Direct Memory Access (DMA)

And the processor is **free** to do something better!



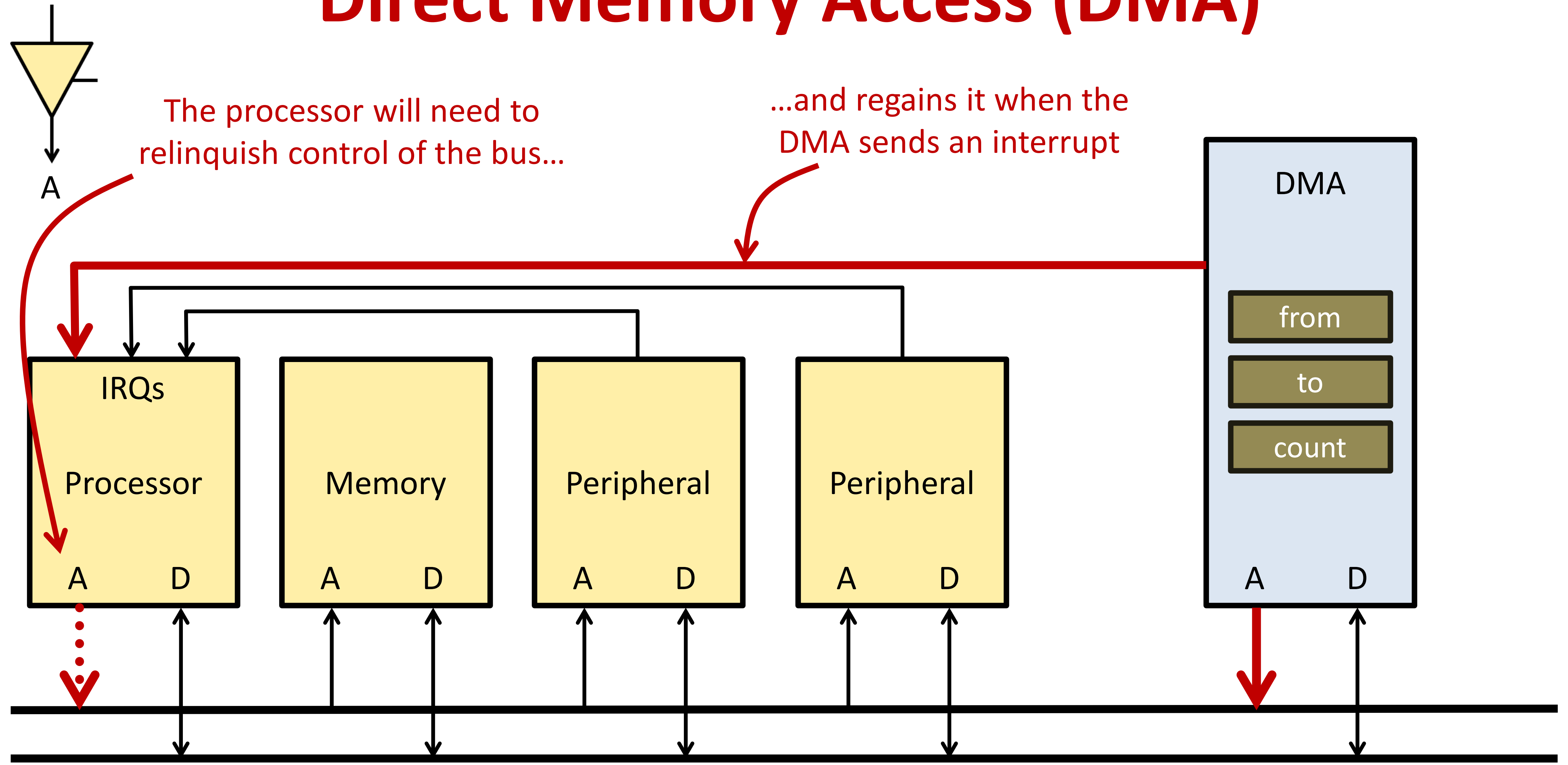
"Store" (Write)

Direct Memory Access (DMA)

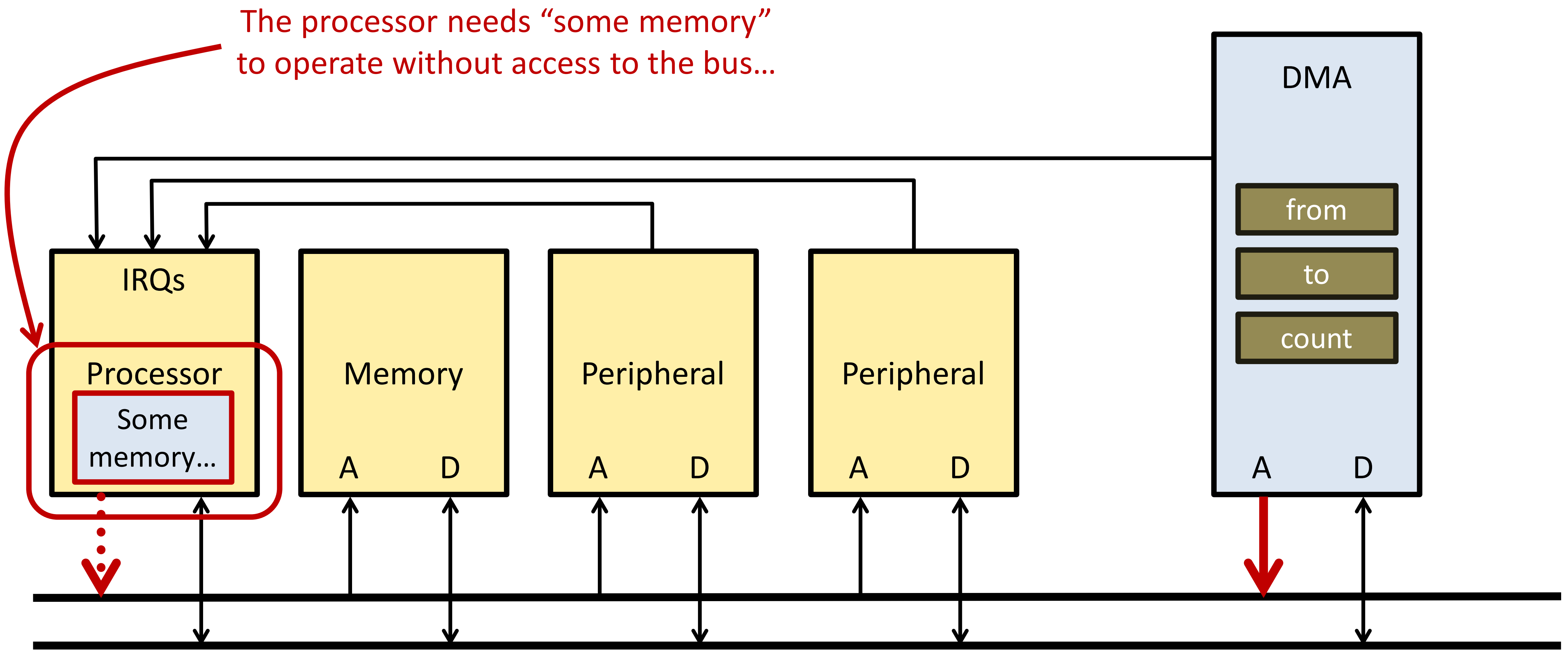


The DMA **must control the bus**
(write addresses and control signals)…
…but the **master** is the processor!
IRQs
Processor
A
D
Memory
A
D
Peripheral
A
D
Peripheral
A
D
DMA
from
to
count
A
D

Direct Memory Access (DMA)


A
The processor will need to relinquish control of the bus…
…and regains it when the DMA sends an interrupt
DMA
from
to
count
IRQs
Processor
Memory
Peripheral
Peripheral
A
D
A
D
A
D
A
D
A
D

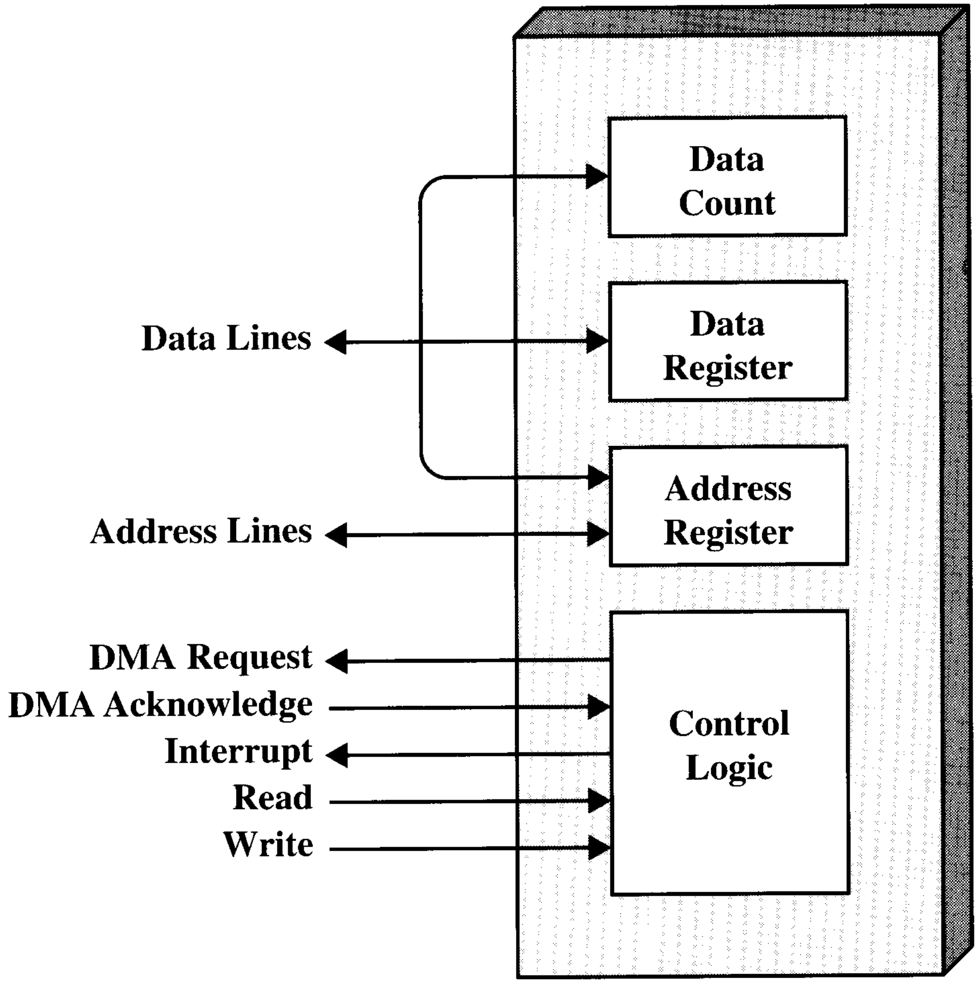
Direct Memory Access (DMA)



Direct Memory Access (DMA)

What is (minimally) in a DMA?

- An **increment register** (how many bytes/words to transfer at a time)
- A couple of **address pointers** (source address pointer and destination address pointer), incremented by the above constant at every transfer
- A **counter** (total number of bytes/words to transfer)



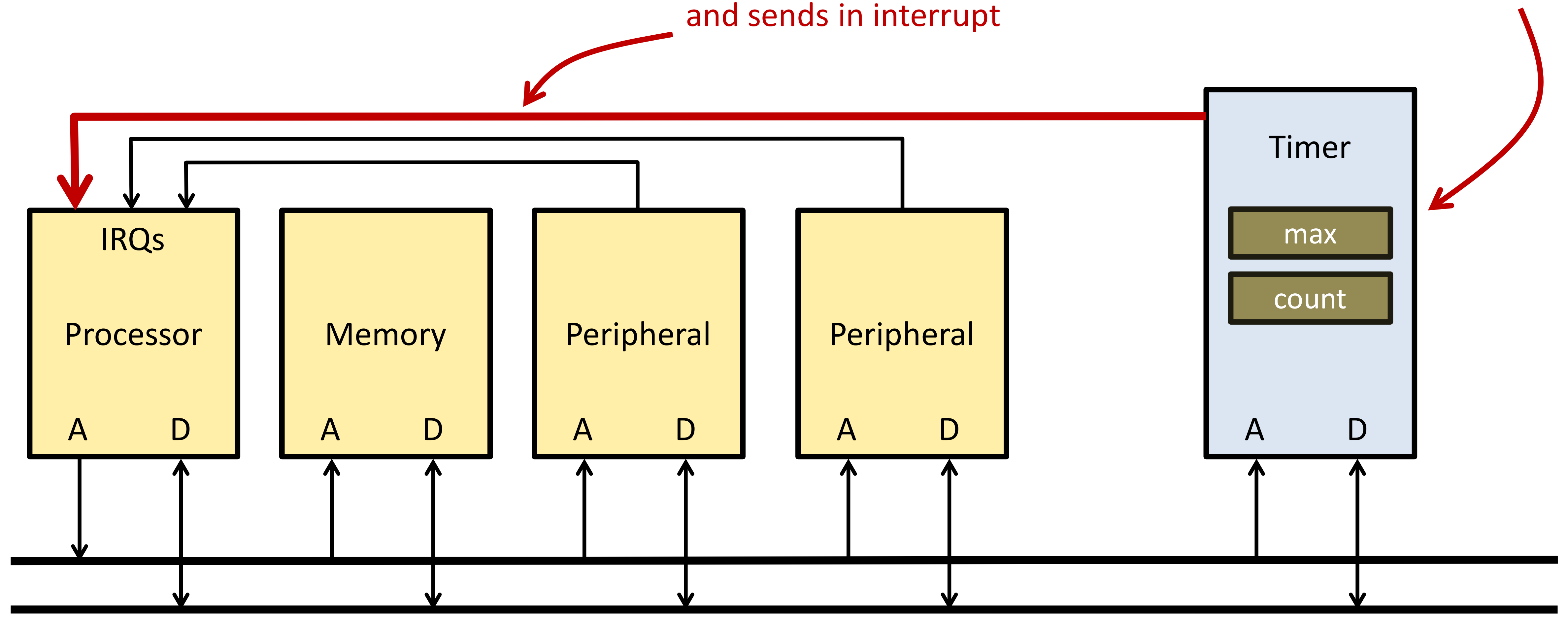
Source: Stalling, © Prentice Hall 2000

Direct Memory Access (DMA)

Example sequence:

1. The processor tells the DMA controller (a) which device to access, (b) where to read or write the memory, and (c) the number of bytes to transfer
2. The DMA controller becomes bus master and performs the required accesses controlling directly the Address and Control busses
3. The DMA controller sends an interrupt to the processor to signal successful completion or errors

Timer


Counts up to **max**
and sends in interrupt
max is programmable,
so that the processor
chooses the frequency
IRQs
Processor
A
D
Memory
A
D
Peripheral
A
D
Peripheral
A
D
Timer
max
count
A
D