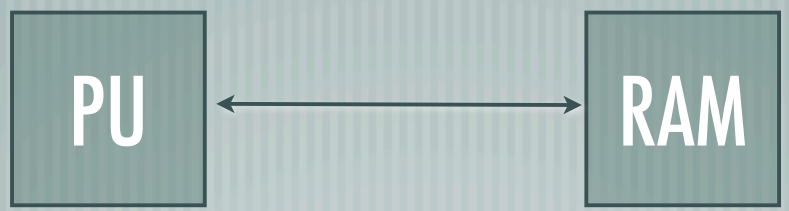
Computer Architecture

R. Poss
Computer Systems Architecture group (UvA)
e-mail: r.c.poss@uva.nl

Input/Output

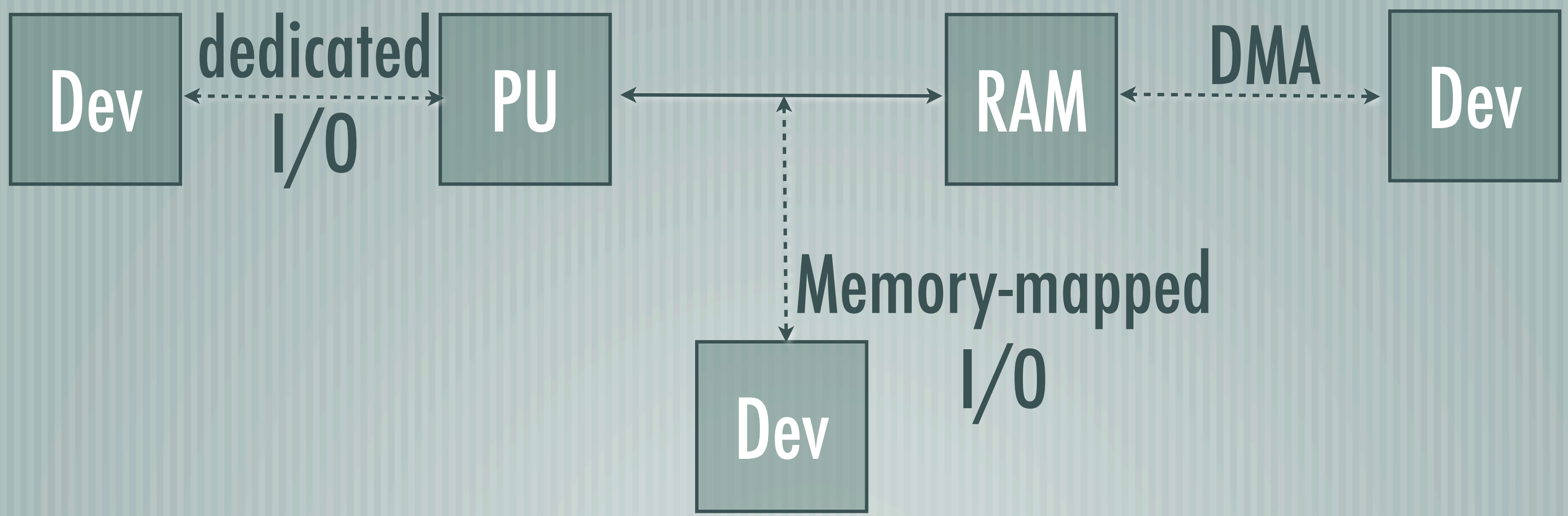
Connection points?

- A computing machine is composed of a processor and a memory – the implementation of Turing's abstract machine



- A useful computer also has input/output to the "outside world"
  - Think: screen, keyboard, network adapter, sound, etc.
- How to interface?

3 Options



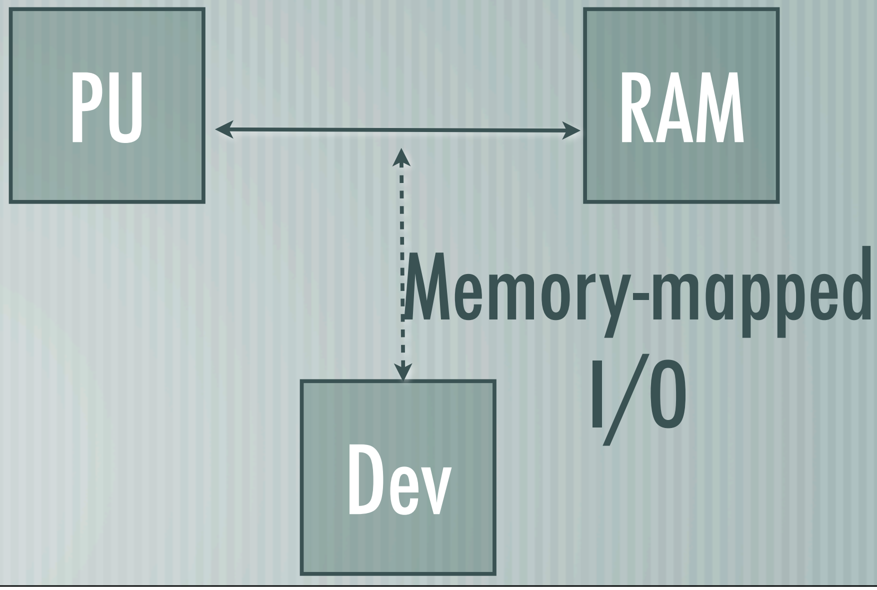
(DMA = Direct Memory Access)

DMA

- Device accesses RAM directly, does not communicate with PU
- Typically used with integrated graphical adapters: the digital-to-video converter gets its data from main RAM, using a separate interface than the processor
  - (Also: sound cards)
  - Advantages: communication fully independent from PU
  - Inconvenients: slower; more difficult to program; difficult to signal availability of input (DMA mostly used for output)

Memory-mapped I/O

- Most common architecture; universally available
  - Read/write operations from the processor to a special range of memory addresses is re-routed to the I/O device instead of RAM
  - Violates the RAM protocol: you may get a different data on reads than what you put with writes; **this confuses caching!**
  - Typically combined with programmable cache behavior on these addresses (eg MTRR on x86)



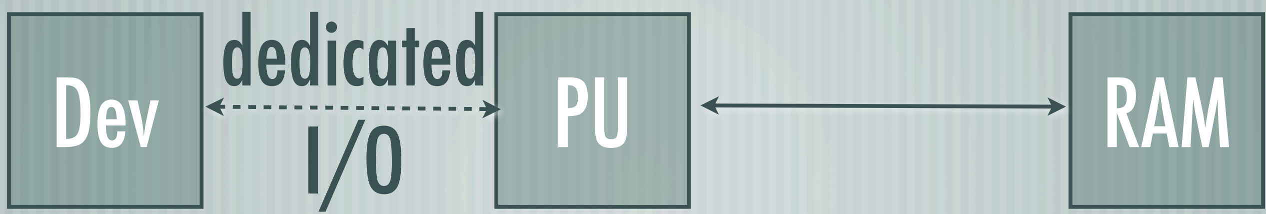
Advantages: cheap to implement, easy to program
Inconvenient: difficult to signal availability of input

Memory-mapped I/O

- Two possible connection points for MMIO:
  - MMIO via System bus or NoC, between processor and memory: the processor only has a memory interface
    - Order: PU - L1 cache - MMIO bridge - RAM
    - Most common
  - MMIO between the pipeline stage and the L1 D-cache: the processor then has two interfaces, one for memory one for I/O
    - Order: PU - MMIO bridge - L1 D-cache - RAM
    - Used eg. by SPARC for cache control; also found in MGSim
- NB: even inside the processor, MMIO only recognizes “read” and “write” operations like RAM

Dedicated I/O

- The processor interface is extended with explicit I/O in addition to the memory load/store interface; it can recognize **more request types**
  - Typically: “I/O read/write”, but also eg. “**wait for event**”
  - Requires new machine instructions to control the new interface
  - eg. x86 has “in” and “out” instructions for dedicated I/O
  - Advantages: most flexibility for designer; inconvenient: every system does it differently!



How to signal input? Polling

- In all 3 options input can be solved by a mixture of **polling** and **buffering**
  - when the processor is not looking, the device accumulates input data into a buffer (or RAM with DMA)
  - every now and then, the processor actively polls (looks into) the buffer or RAM to fetch the available input, if any
  - **poll rate** and **input rate** are interlocked:
    - poll rate lower than input rate causes *data loss*: “buffer overruns”
    - poll rate higher than input rate causes *energy waste*: busy loop
- Dangerous to rely only on polling: what if the program is stuck and does not poll?

How to signal input? Interrupts

- **Interrupts** have been designed to overcome 2 problems of polling:
  - Only do something special if there is input available (avoid busy looping)
  - Re-take control of unresponsive software, time sharing
- Universal mechanism: a device-generated signal forces the program counter to change to a previously agreed value

Interrupts - how do they work?

- A signal (wire) is connected from outside the processor to an interrupt management circuit, controls the pipeline:

1. stop fetching new instructions
2. wait until current work is finished
3. save the current processor context: PC, registers, status codes
4. set status code to “interrupted”, set PC to new value*
5. start fetching instructions at new PC

Interrupts - how do they work?

- The special “new PC” usually comes from a programmable interrupt controller (PIC)
  - This is itself a device outside the processor, usually configurable via dedicated or MMIO. NB: PIC is not the same as interrupt management circuit inside PU.
  - The PIC delivers a different PC to the processor depending on the interrupt’s origin (which I/O device has input available)
- When system starts up, OS sets up interrupt handling routines in PIC: special sub-programs (C functions) to handle the event when it arrives
  - interrupt handler in charge of delivering the input to OS/app, then restore processor state to resume execution of the interrupted program