Systems Programming and Computer Architecture

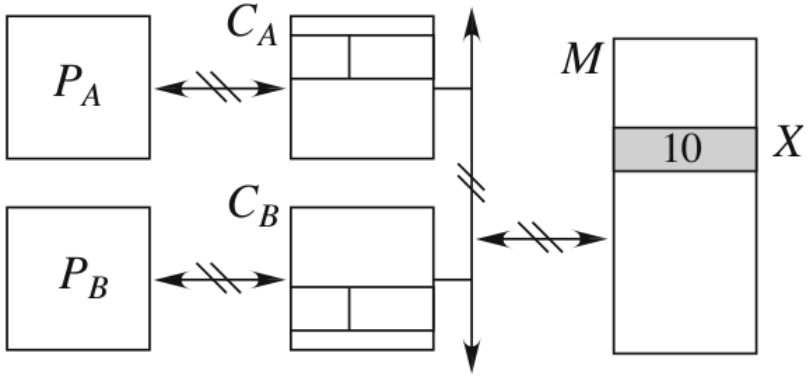
Exercise Session 14

"The last one"

- Assignment 12
- Devices recap
- Assignment 13

Exercise 12

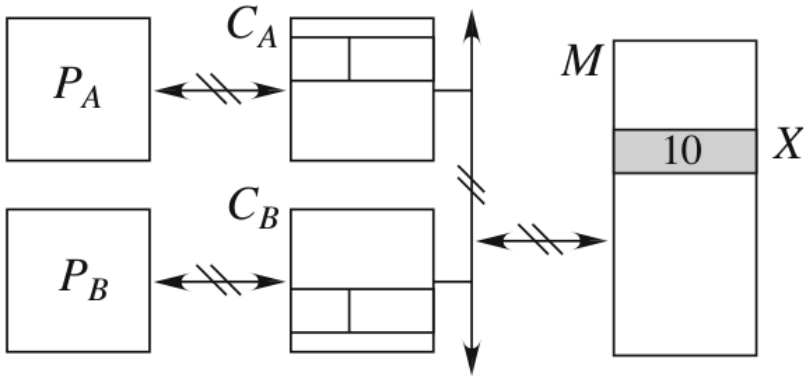
Question 1a: MSI



n	P_A	P_B	Comment
1	mov (X),r1		P_A: r1 := (X)
2		mov (X),r4	P_B: r4 := (X)
3	mov $0,(X)		P_A: (X) := 0
4		mov (X),r5	P_B: r5 := (X)
5		mov $20,(X)	P_B: (X) := 20

n	State Ca	Value Ca	State Cb	Value Cb	Value M
1	S	10	I	–	10
2	S	10	S	10	10
3	M	0	I	–	10
4	S	0	S	0	0
5	I	–	M	20	0

Question 1b: MESI



n	P_A	P_B	Comment
1	mov (X),r1		P_A: r1 := (X)
2		mov (X),r4	P_B: r4 := (X)
3	mov $0,(X)		P_A: (X) := 0
4		mov (X),r5	P_B: r5 := (X)
5		mov $20,(X)	P_B: (X) := 20

Only difference between MESI and MSI

n	State Ca	Value Ca	State Cb	Value Cb	Value M
1	E	10	I	–	10
2	S	10	S	10	10
3	M	0	I	–	10
4	S	0	S	0	0
5	I	–	M	20	0

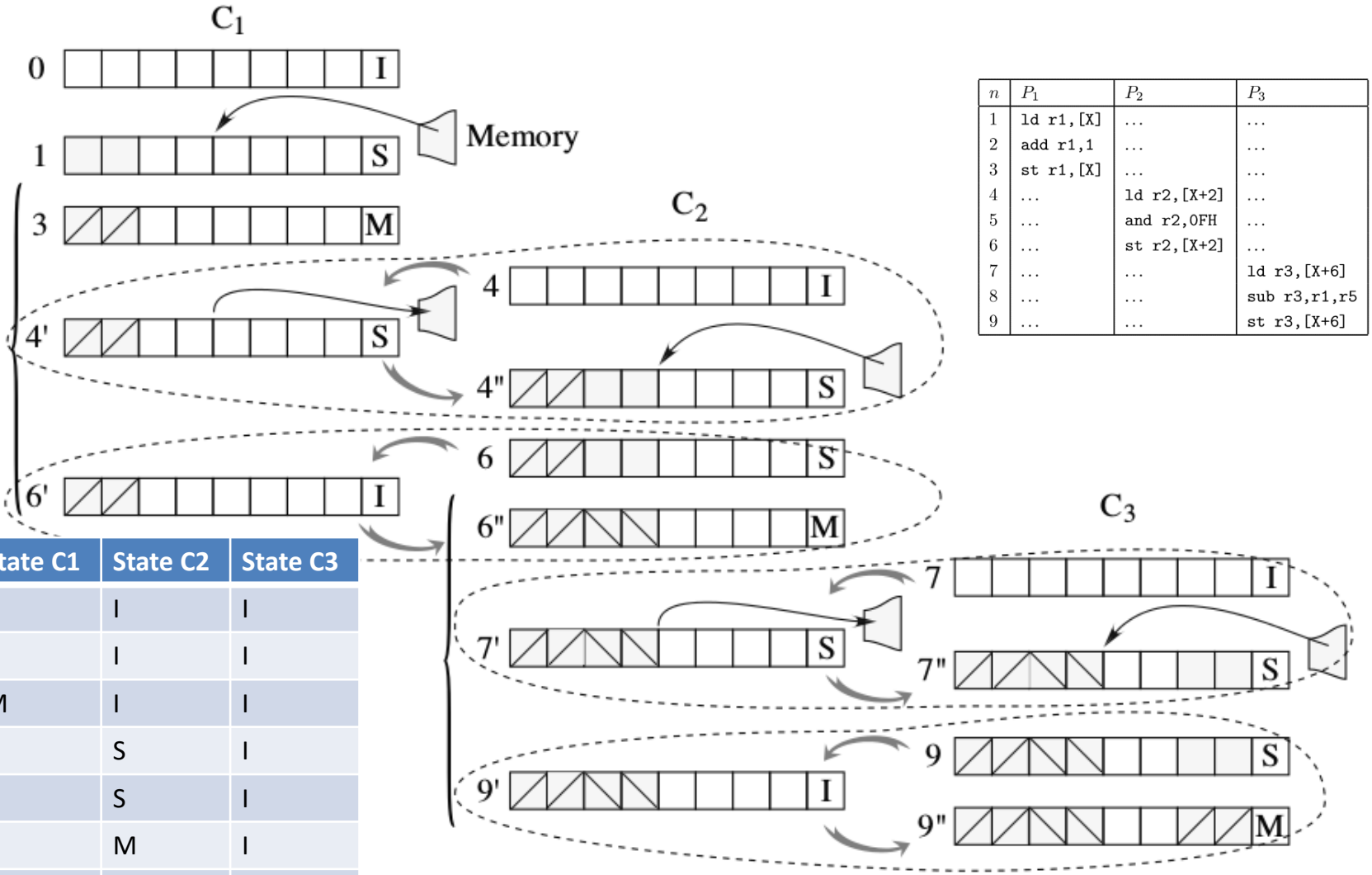
Question 2

- 2a: 0xA0C0 maps to index 8 (bits 6-3).

- 3 cores
- 128b direct-mapped write-back cache, 8b cache line size
- X = 0xA0C0

n	P_1	P_2	P_3
1	`ld r1,[X]`	...	...
2	`add r1,1`	...	...
3	`st r1,[X]`	...	...
4	...	`ld r2,[X+2]`	...
5	...	`and r2,0FH`	...
6	...	`st r2,[X+2]`	...
7	...	...	`ld r3,[X+6]`
8	...	...	`sub r3,r1,r5`
9	...	...	`st r3,[X+6]`

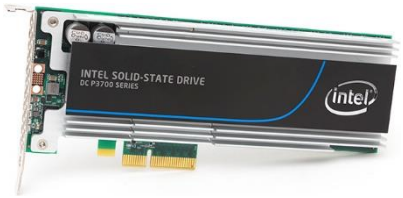
Question 2b: MSI

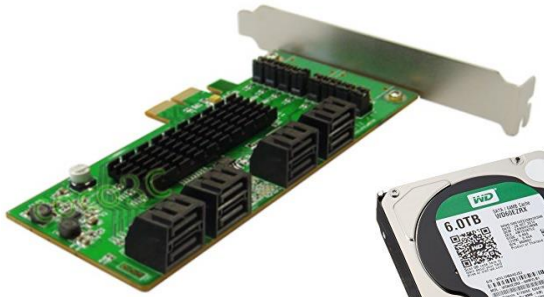


n	P_1	P_2	P_3
1	`ld r1,[X]`	...	...
2	`add r1,1`	...	...
3	`st r1,[X]`	...	...
4	...	`ld r2,[X+2]`	...
5	...	`and r2,0FH`	...
6	...	`st r2,[X+2]`	...
7	...	...	`ld r3,[X+6]`
8	...	...	`sub r3,r1,r5`
9	...	...	`st r3,[X+6]`

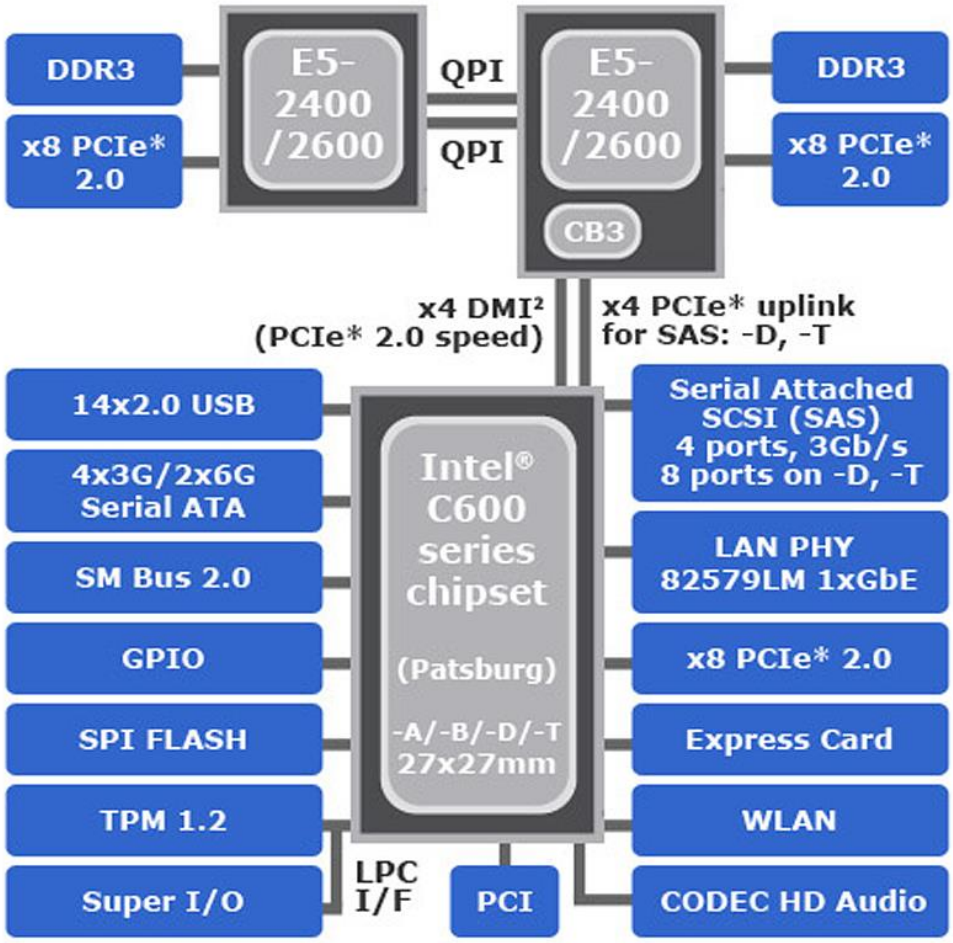
n	State C1	State C2	State C3
1	S	I	I
2	S	I	I
3	M	I	I
4	S	S	I
5	S	S	I
6	I	M	I
7	I	S	S
8	I	S	S
9	I	I	M

Devices Recap



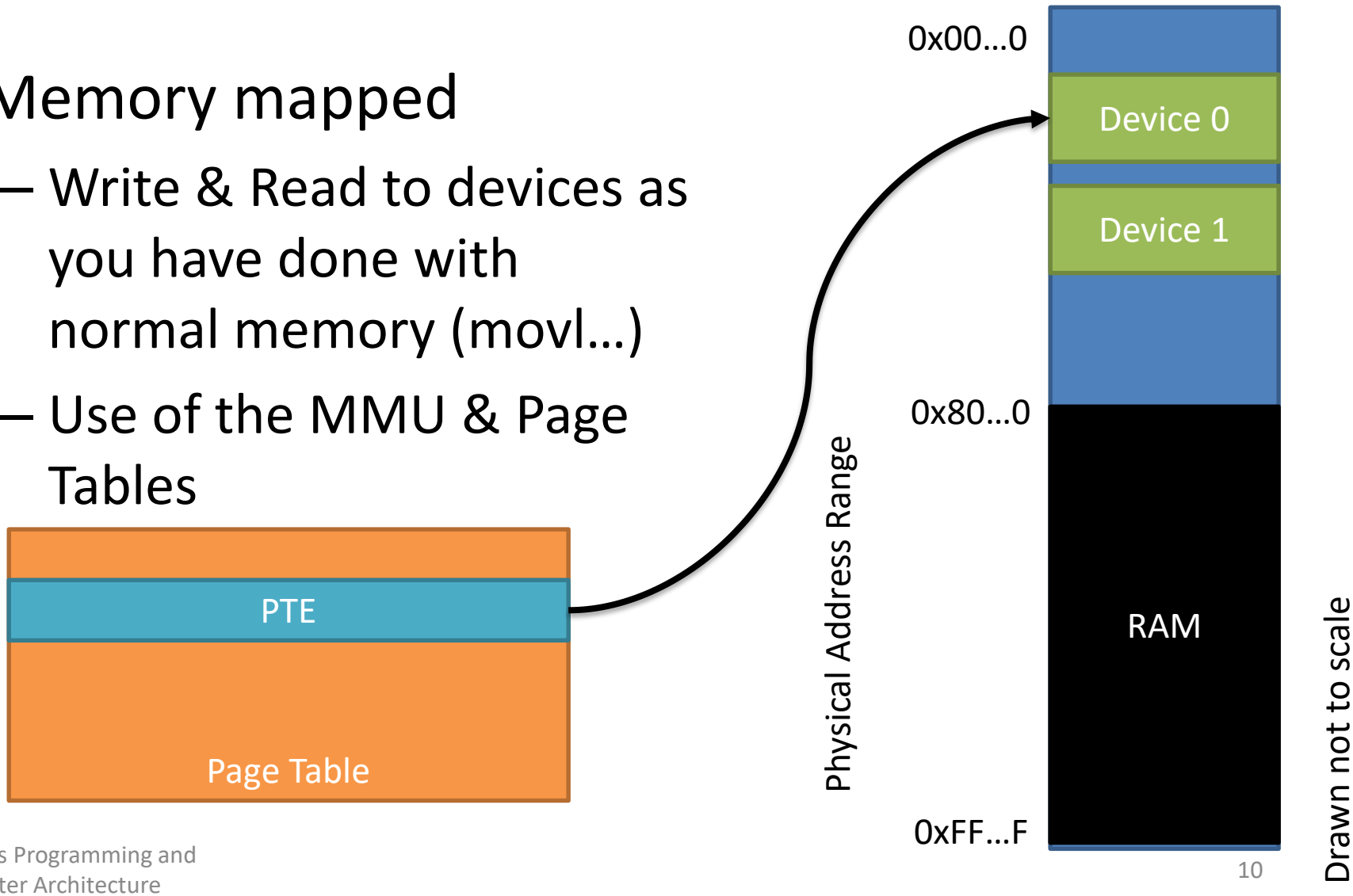


Devices


DDR3
x8 PCIe* 2.0
E5-2400/2600
QPI
QPI
E5-2400/2600
CB3
DDR3
x8 PCIe* 2.0
x4 DMI² (PCIe* 2.0 speed)
x4 PCIe* uplink for SAS: -D, -T
14x2.0 USB
4x3G/2x6G Serial ATA
SM Bus 2.0
GPIO
SPI FLASH
TPM 1.2
Super I/O
LPC I/F
Intel® C600 series chipset
(Patsburg)
-A/-B/-D/-T 27x27mm
PCI
Serial Attached SCSI (SAS) 4 ports, 3Gb/s 8 ports on -D, -T
LAN PHY 82579LM 1xGbE
x8 PCIe* 2.0
Express Card
WLAN
CODEC HD Audio

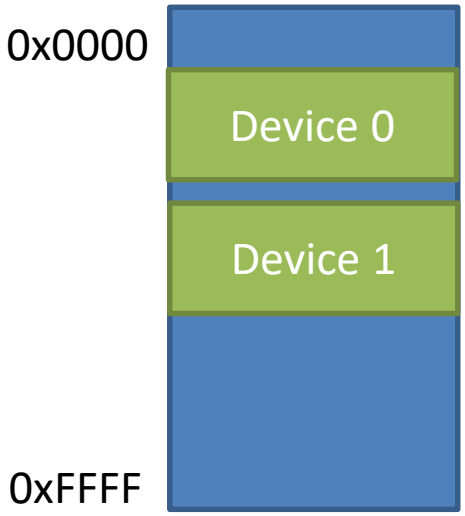
How to access devices?

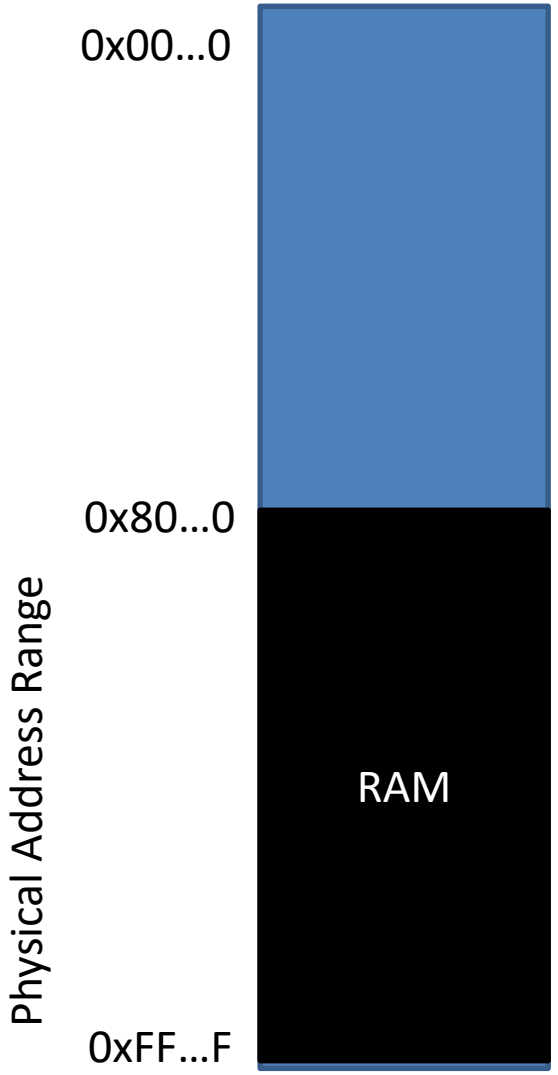
- Memory mapped
  - Write & Read to devices as you have done with normal memory (movl…)
  - Use of the MMU & Page Tables



How to access devices?

- IO Ports
  - Special address space (16 bit)
  - Write to an IO port





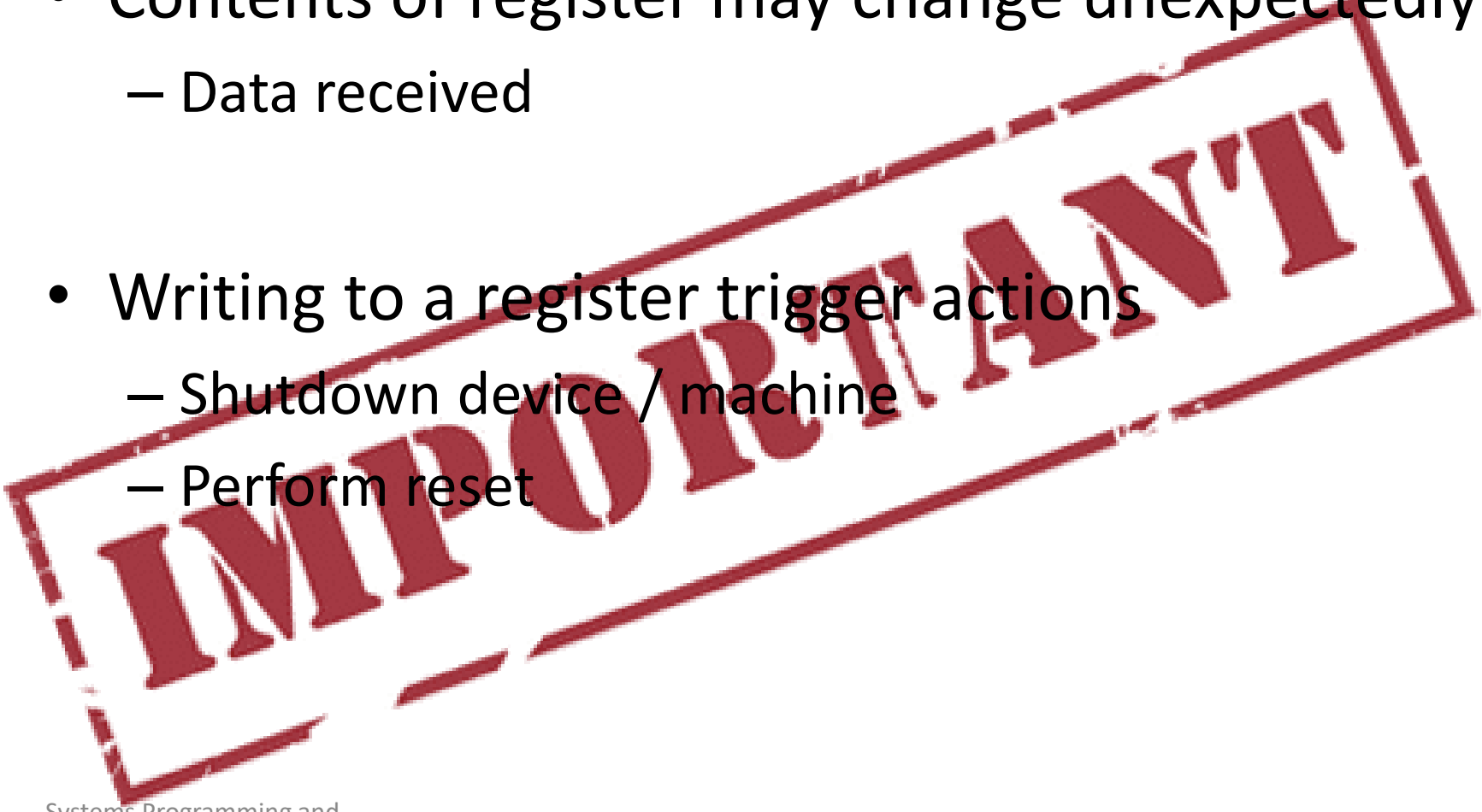
Memory Mapped Devices

Systems@ETH zürich

- Device represented as [Base Address, Length]
  - Base address refers to the physical address where the device starts
    - cf: your beginning of the stack frame
  - Length total occupied memory region by that device (actually used by registers may be lower)
    - cf: the size of your stack frame
  - Set of registers within Base, Base+Length
    - cf: your variables on the stack

Devices are **NOT** memory

- Contents of register may change unexpectedly
  - Data received
- Writing to a register trigger actions
  - Shutdown device / machine
  - Perform reset



Too simple UART driver

```
1.  #define UART_BASE 0x12345000
2.  #define UART_THR (UART_BASE + 0)
3.  #define UART_RBR (UART_BASE + 4)
4.  #define UART_LSR (UART_BASE + 8)

5.  void serial_putc(char c)
6.  {
7.    char *lsr =  (char *) UART_LSR;
8.    char *thr =  (char *) UART_THR;

9.    // Wait until FIFO can hold more chars
10.   while((*lsr & 0x20)== 0);
11.
12.   *thr = c
13. }
```

- What's the problem here?

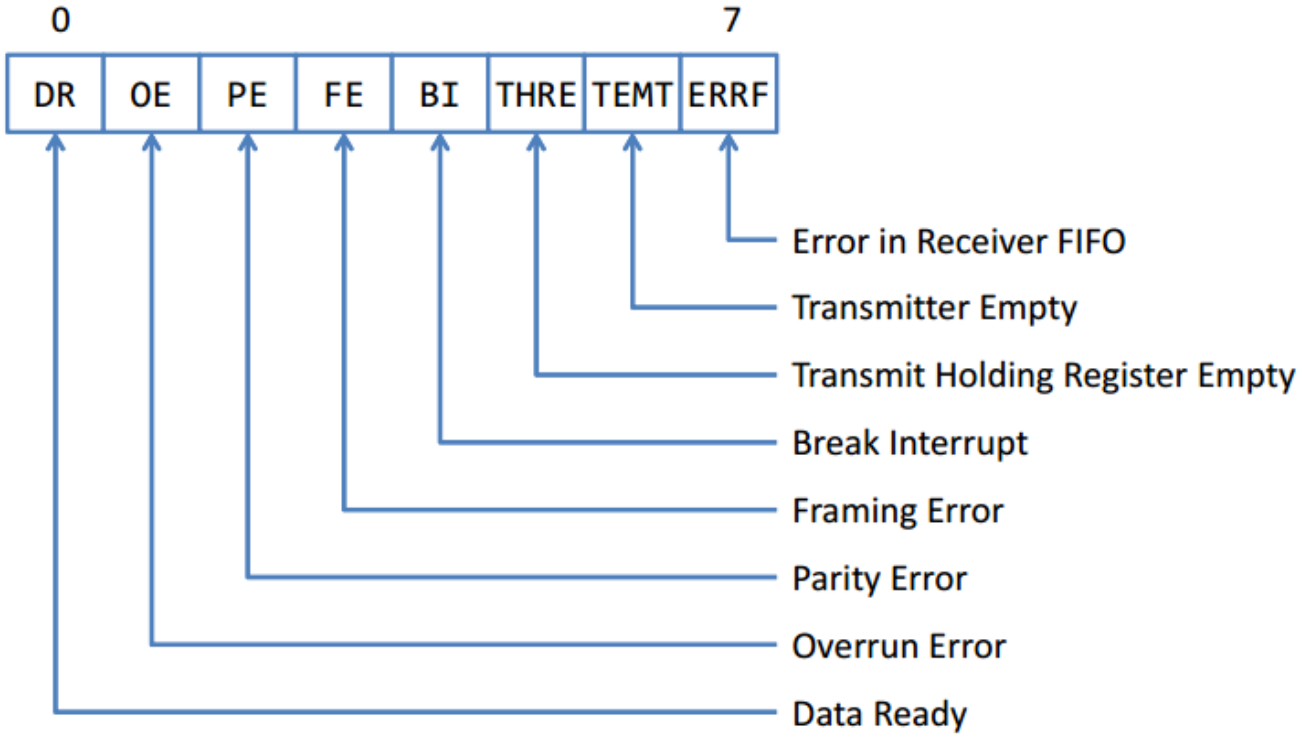
Too simple UART driver

```
1.  #define UART_BASE 0x12345000
2.  #define UART_THR (UART_BASE + 0)
3.  #define UART_RBR (UART_BASE + 4)
4.  #define UART_LSR (UART_BASE + 8)

5.  void serial_putc(char c)
6.  {
7.    volatile char *lsr =  (char *) UART_LSR;
8.    volatile char *thr =  (char *) UART_THR;

9.    // Wait until FIFO can hold more chars
10.   while((*lsr & 0x20)== 0);
11.
12.   *thr = c
13. }
```

- Compiler does not know this is a device register!
- Loop gets optimized away!
- Add: volatile

Register Contents

- Each bit may have a different meaning
- Lots of configurations possible!



Devices

- Writing device drivers is tedious
  - Setting a single bit wrong and the device does not work
  - Debugging is hard: Likely to set a bit wrong due to a wrong shift & mask.

Devices & Caches

Systems@ETH zürich

- Device registers cannot be cached due to inconsistency problem i.e. register content changes w/o CPU write!
- What about cache lines? Would overwrite other register values when writing back
- Set the “no-cache” flag in the page table entry

Interrupts

```
1.  #define UART_BASE 0x12345000
2.  #define UART_THR (UART_BASE + 0)
3.  #define UART_RBR (UART_BASE + 4)
4.  #define UART_LSR (UART_BASE + 8)

5.  void serial_putc(char c)
6.  {
7.    volatile char *lsr =  (char *) UART_LSR;
8.    volatile char *thr =  (char *) UART_THR;

9.    // Wait until FIFO can hold more chars
10.   while((*lsr & 0x20)== 0);
11.
12.   *thr = c
13. }
```

- Any “problems” with that one?

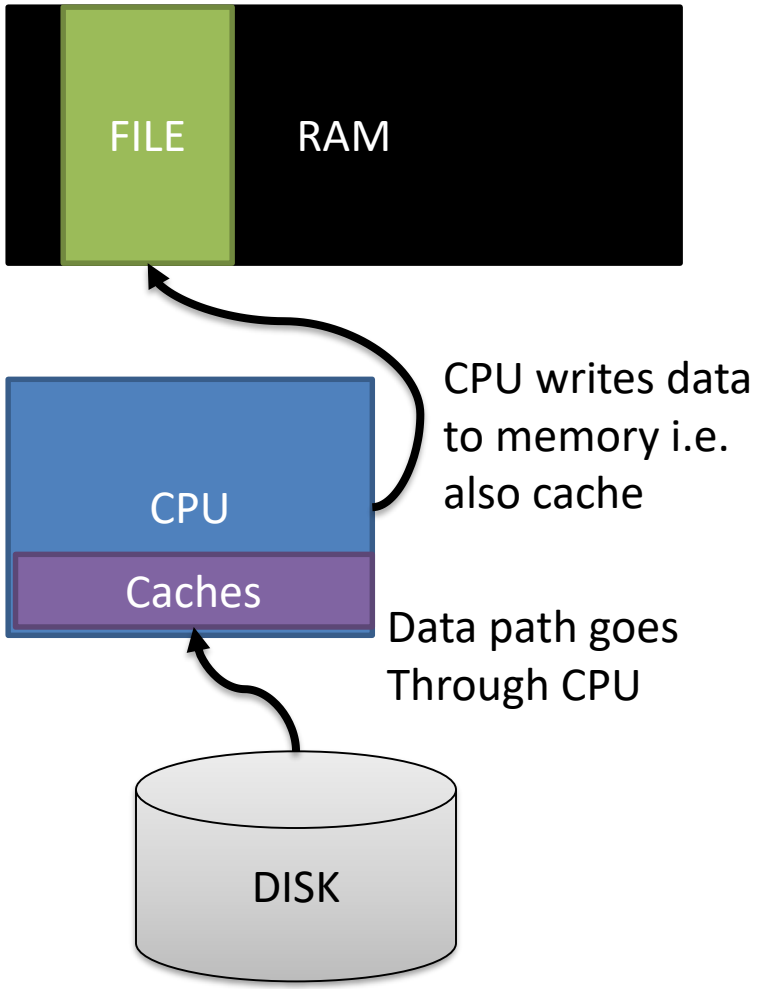
Interrupts

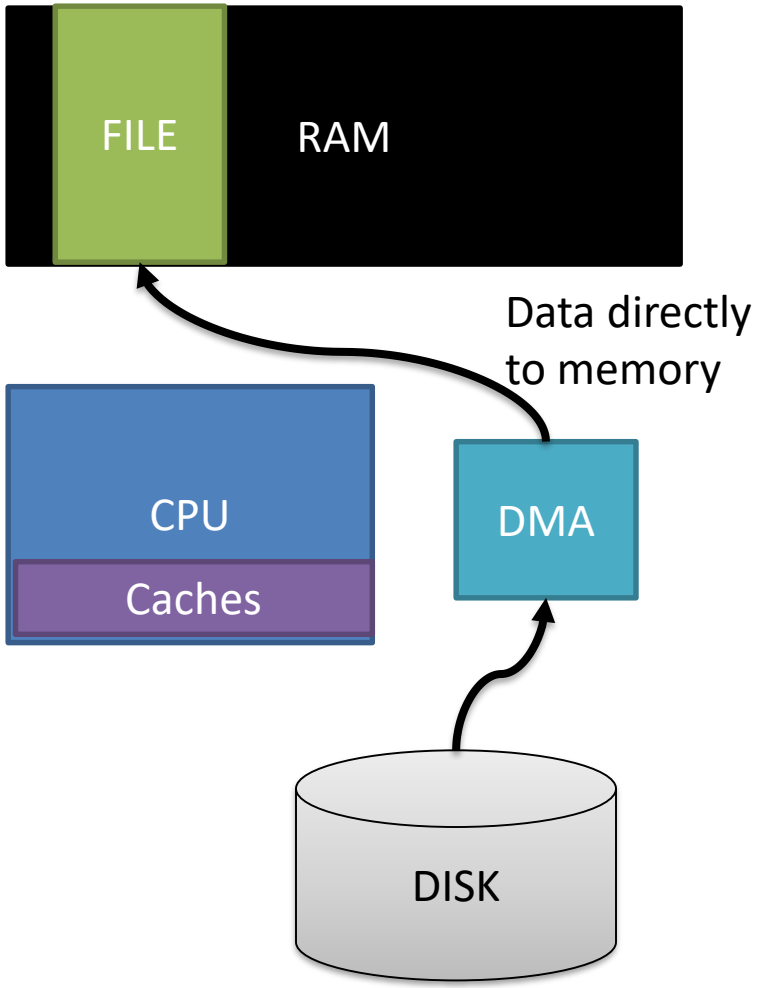
```
1.  #define UART_BASE 0x12345000
2.  #define UART_THR (UART_BASE + 0)
3.  #define UART_RBR (UART_BASE + 4)
4.  #define UART_LSR (UART_BASE + 8)

5.  void serial_putc(char c)
6.  {
7.    volatile char *lsr =  (char *) UART_LSR;
8.    volatile char *thr =  (char *) UART_THR;

9.    // Wait until FIFO can hold more chars
10.   while((*lsr & 0x20)== 0);
11.
12.   *thr = c
13. }
```

- Register callback for an interrupt
- Activate interrupt

Polls register
=
wastes CPU cycles

DMA

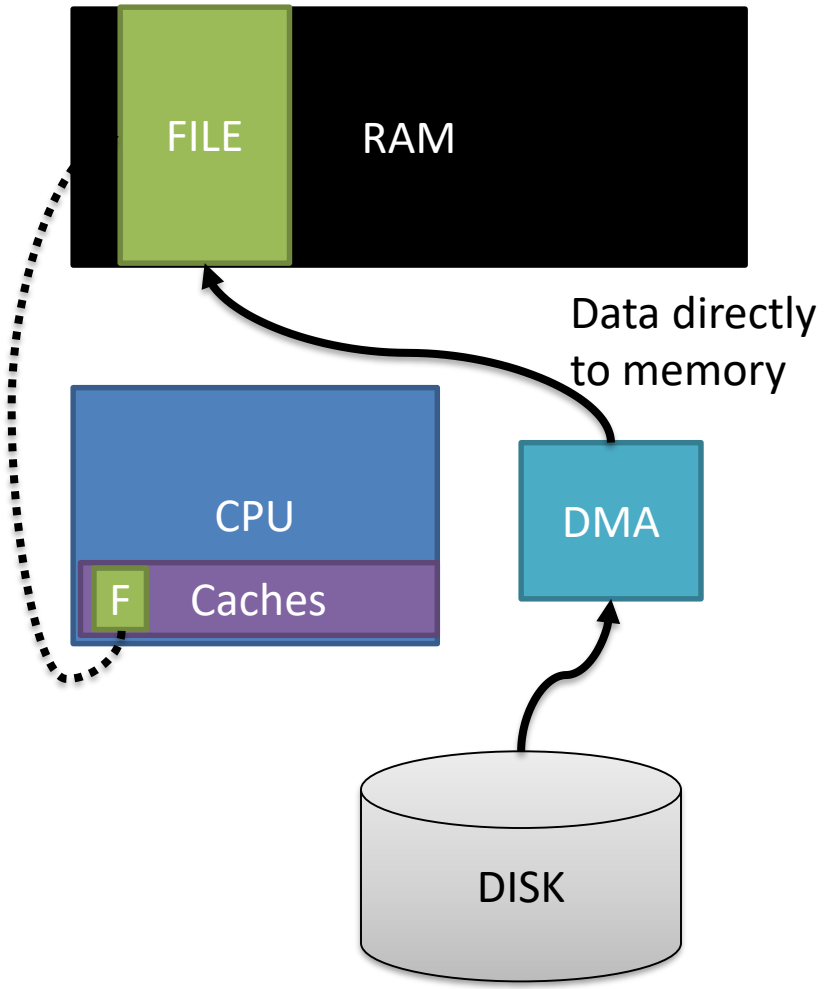




Any Problems with DMA?

DMA

- DMA is like device registers! Changes the contents w/o CPU write.
- Cannot “no-cache”: Bad performance
- Explicitly invalidate cache!



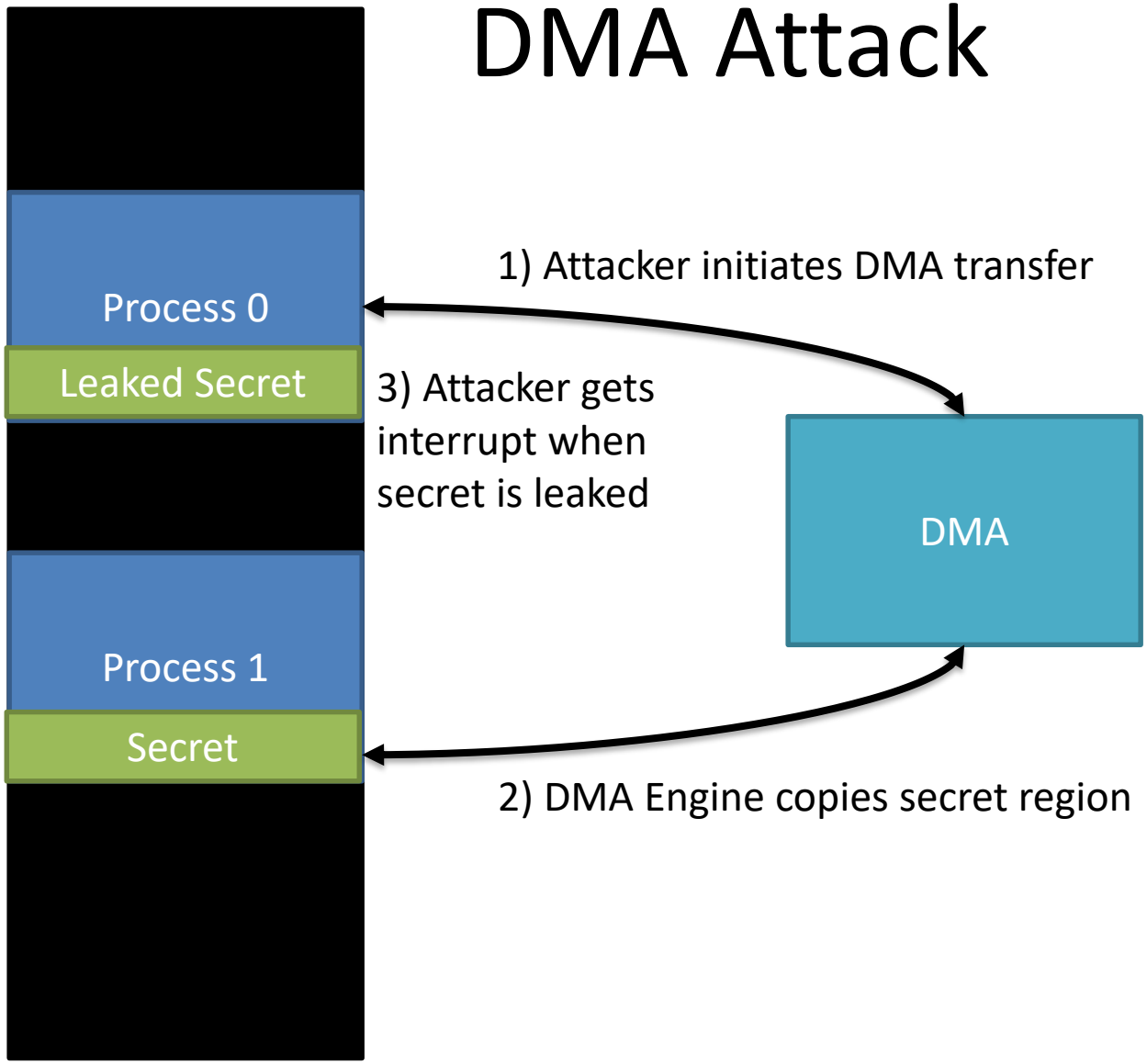
DMA

- Deal with physical addresses only!
  - Any problems with that?

DMA

- Deal with physical addresses only!
  - Any problems with that?
  - Programs deal with virtual addresses: need translation!
  - Physical Range may not be contiguous (gather/scatter)
- How about security?

DMA Attack



Attack done by external devices i.e. firewire / thunderbolt

http://en.wikipedia.org/wiki/DMA_attack

DMA & Cache Problems

Systems@ETH zürich

- On DMA read:
  - Before: Flush the cache to update main memory
  - After: Invalidate the cache to avoid old values seen
- On DMA write:
  - Before: flush or invalidate the cache to update main memory
  - After: invalidate CPU cache

X-Mas Assignment 13

An unknown MPFC (?) device appeared

[ ] Ignore

[x] Write Driver

Task

- You will write a device driver for a MPFC device

- Device Specs
  - Uses single descriptor ring
  - Uses DMA transfers
  - Uses interrupts for “new item” or “no buffer” signals
  - Memory Mapped Device

Steps to do

- Initialize the device (reset) and setting up the in memory data structures
- Start the device
- Start issuing DMA requests to the device
- Activate interrupt and go to sleep
- Do not terminate! Hand back the buffers to the device once used

Hints: Interrupts

Systems@ETH zürich

- You will need to register a handler which is called for received interrupts

- Interrupt received:
  - Wake the sleeping thread
  - Acknowledge the interrupt

Hints: Buffer Ring

Systems@ETH zürich

- Allocate enough memory
- Keep track of who owns the buffer
- Tell the device where to find the buffer rings!
  - (Normally you would have to give the physical address, but we stay virtual this time)

- Whishing you a merry Christmas and all the best in the new year!