



Electronics Systems (938II)

Lecture 3.7

Semiconductor Memories – Computer memory

Summary of semiconductor memories

- So far, we have seen several semiconductor memories, each with advantages and disadvantages over the others
 - Register/DFF
 - ROM/PROM/EPROM/EEPROM/Flash
 - SRAM
 - DRAM

Summary of semiconductor memories

- Let's compare them

Memory technology	Cost* (T)	Speed (Delay)	Other
DFF	10T – 20T	Very high (10 ps)	Volatile
SRAM	4T – 6T	High (1-10 ns)	Volatile
DRAM	≈1T	Medium (100 ns)	Volatile
Flash	1T	Slow (100 μs)	Non-volatile

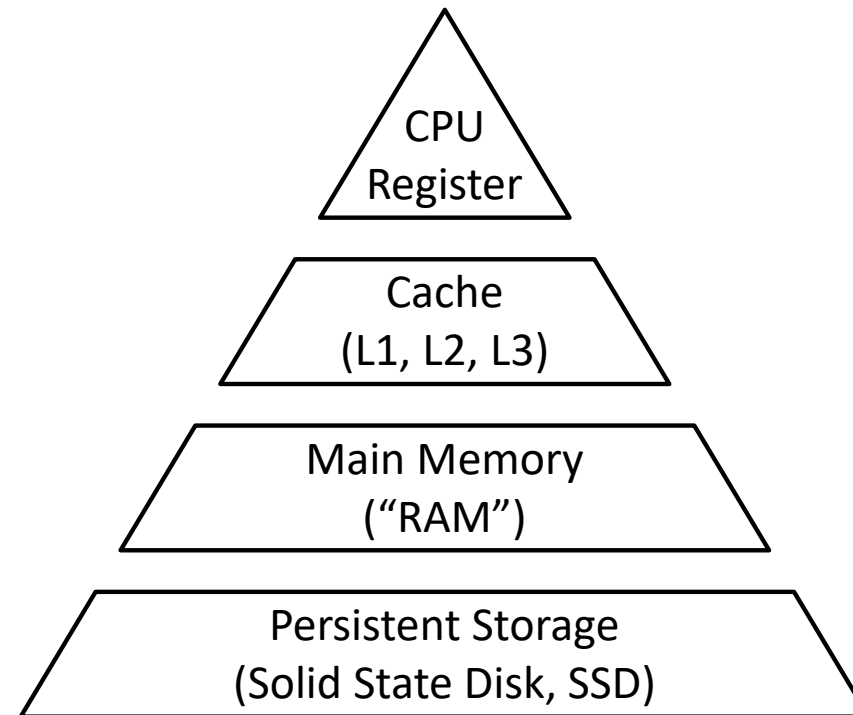
* T = Transistors

Summary of semiconductor memories

- All the analyzed memory technologies are used in modern computers/computing systems for different purposes
- Let's take a look at modern computer memories

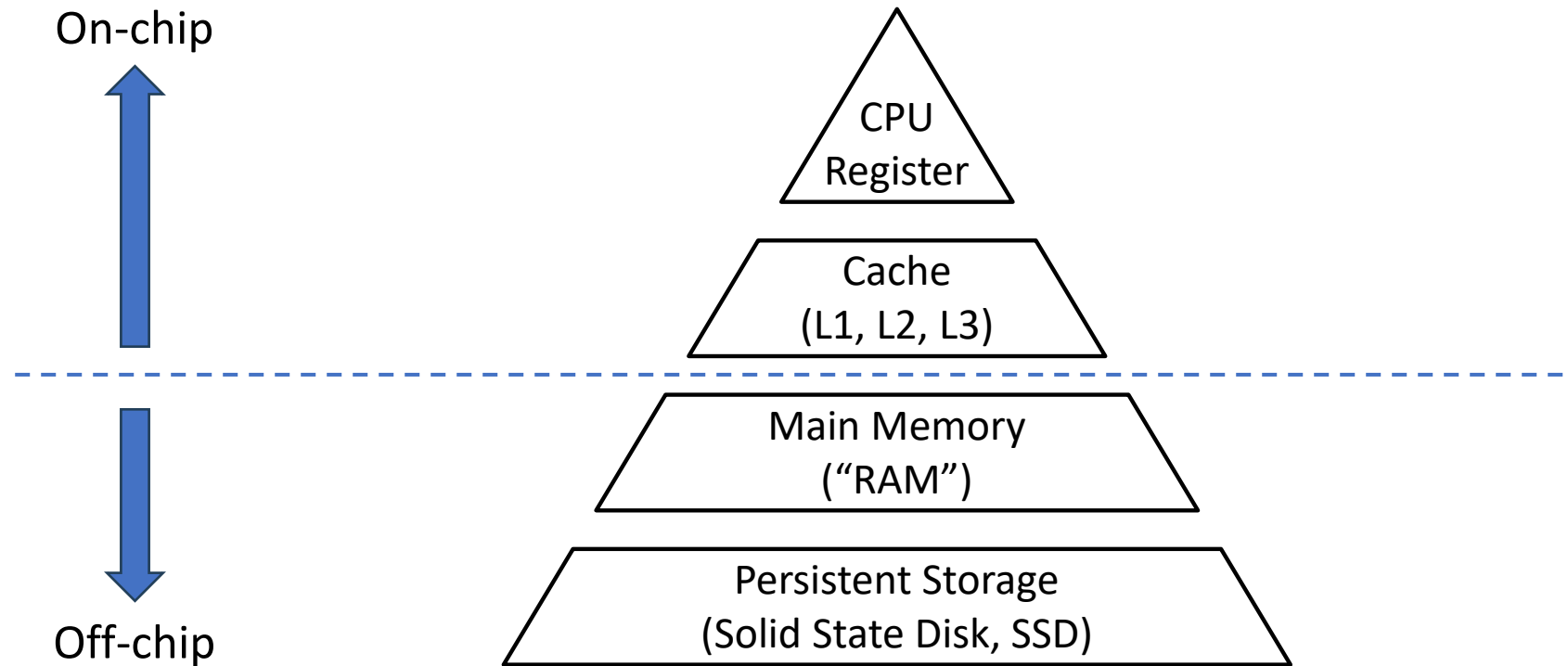
Computer memory

- Hierarchical organization
 - Each level interacts only with the higher one (if any) and the lower one (if any)



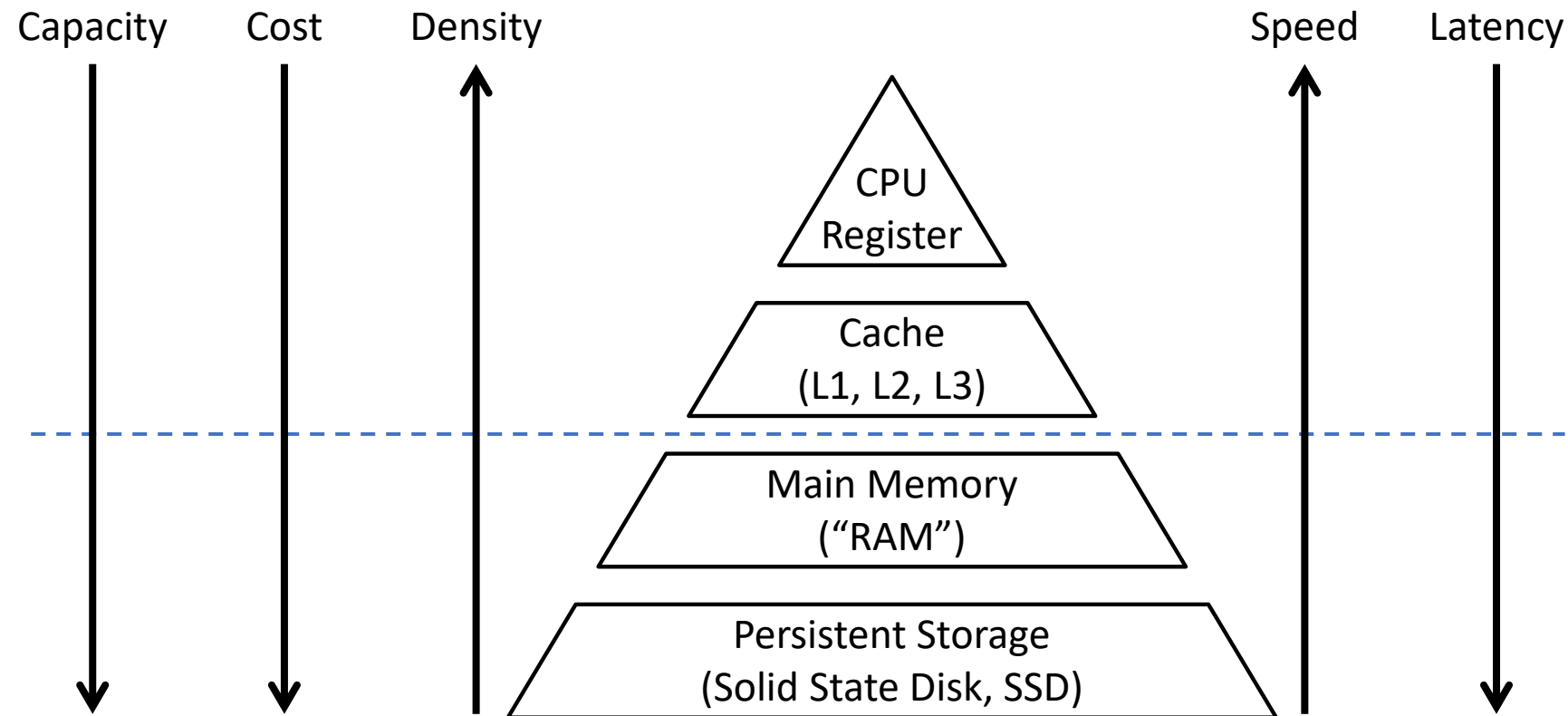
Computer memory

- Hierarchical organization



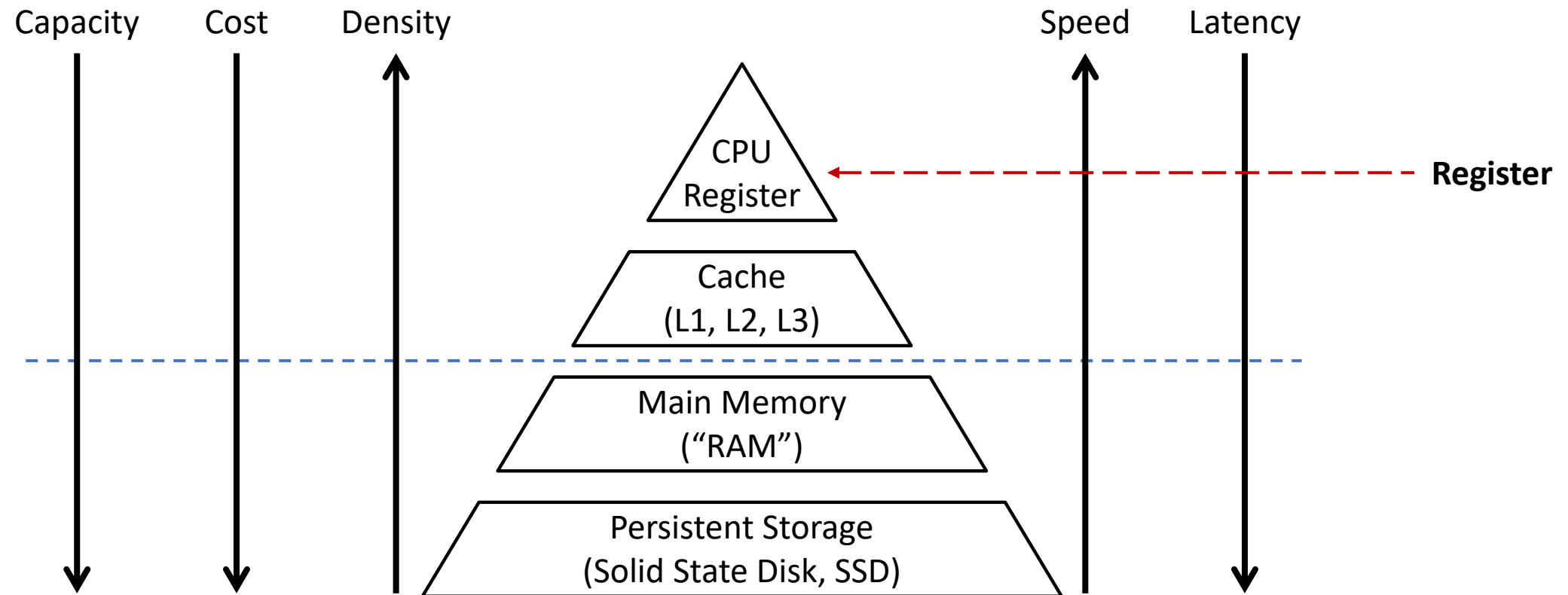
Computer memory

- Hierarchical organization



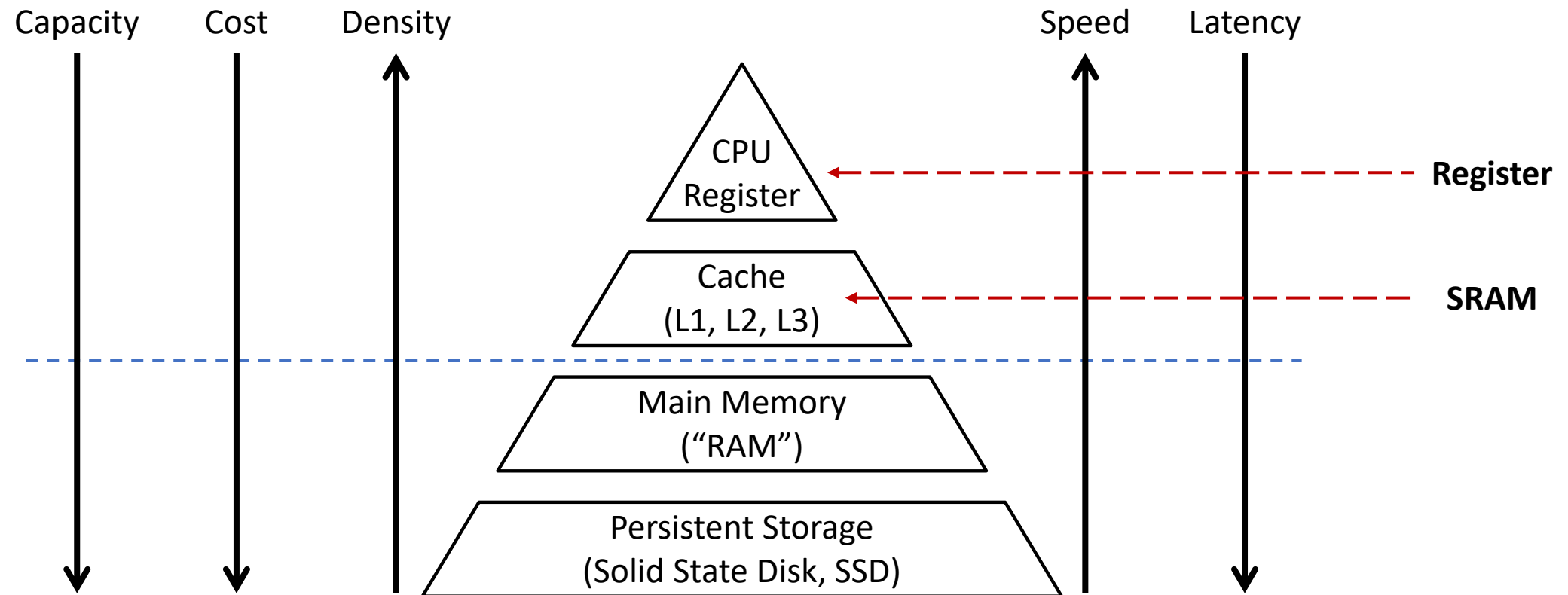
Computer memory

- Hierarchical organization



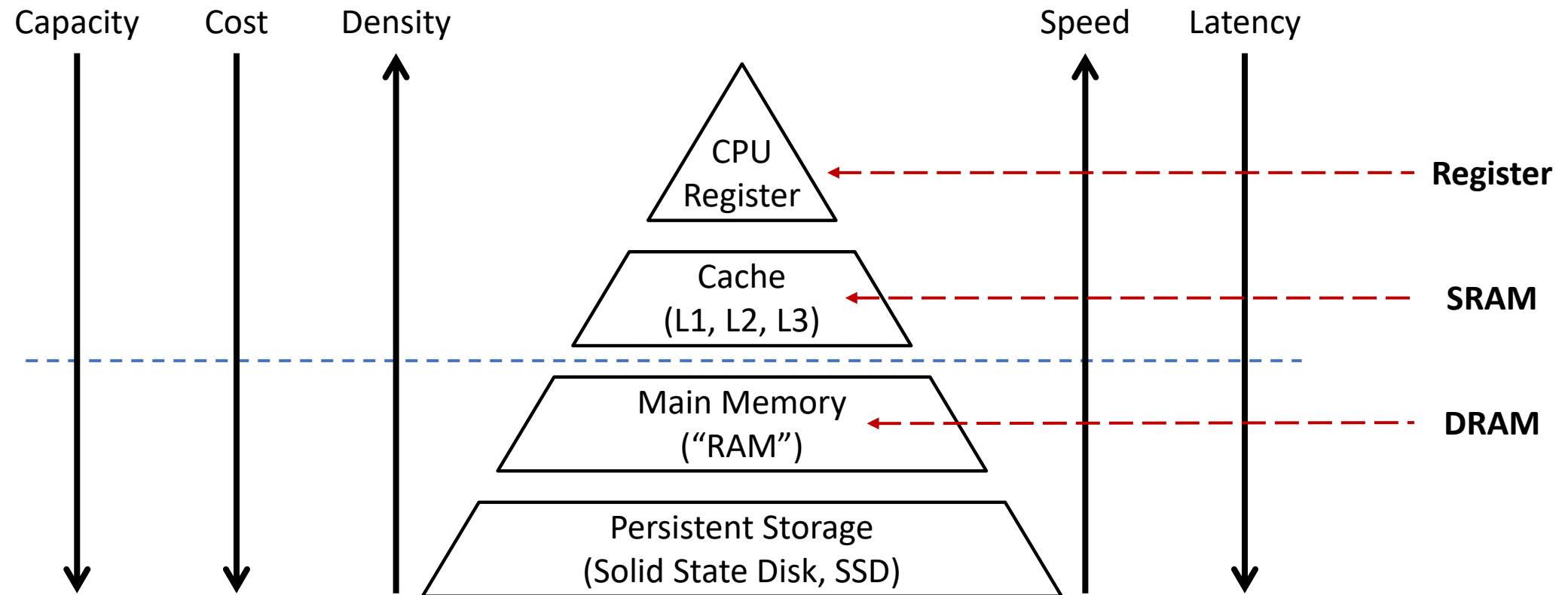
Computer memory

- Hierarchical organization



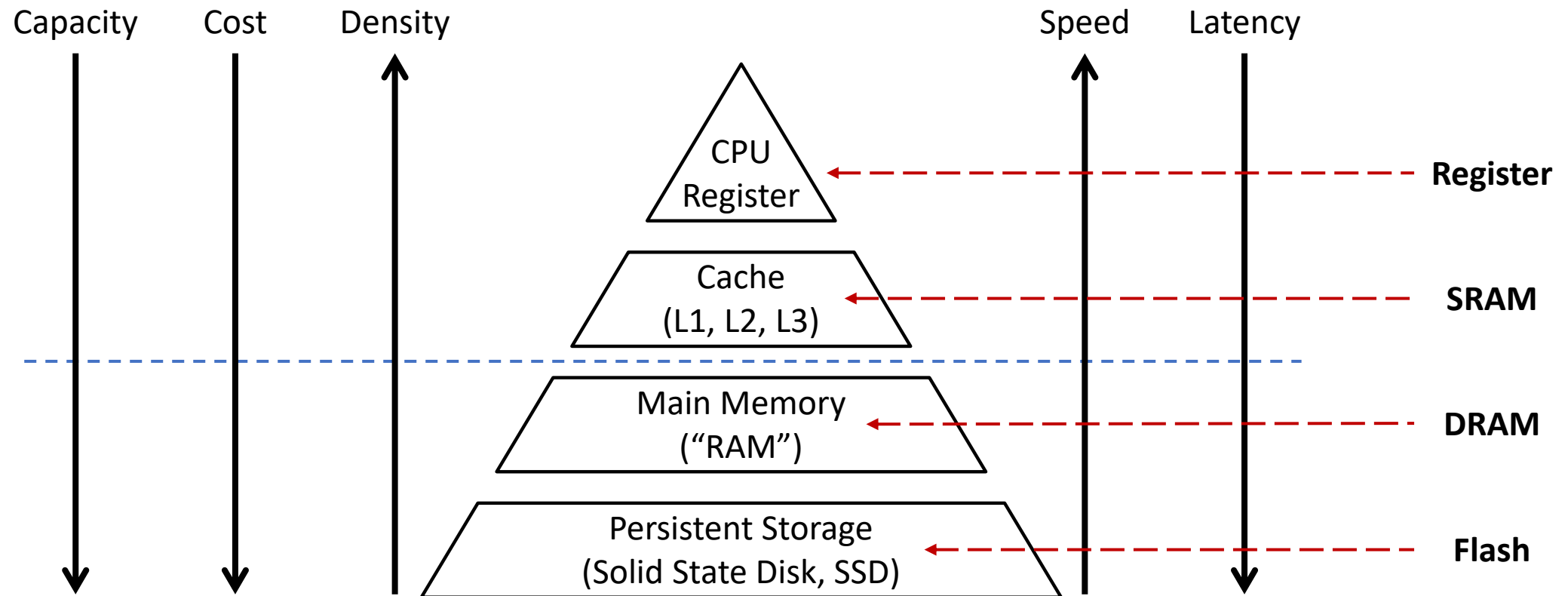
Computer memory

- Hierarchical organization



Computer memory

- Hierarchical organization



Computer memory

- Summary

Computer Memory	CMOS Memory technology	Capacity	Latency	Cost/GB	Position
CPU registers	Register	1 Kb	Very low (10 ps)	\$\$\$\$	On-chip
Cache	SRAM	1 KB – 10 MB	Medium-low (1-10 ns)	100 – 1000 \$	On-chip
Main Memory	DRAM	1 GB – 10 GB	Medium (100 ns)	4 – 5 \$	Off-chip
Storage/SSD	Flash	100 GB – 1 TB	Very High (100 μ s)	0.1 \$	Off-chip

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim
 - Often, RAM (Main Memory) and ROM are used in the HDL design flow as just model for simulation
 - Otherwise, if they are used for memory implementation, they require special and dedicated steps in the design flow
 - E.g., Memory Compiler automation tools

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```
module ram (
```

```
);
```

```
endmodule
```

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```
module ram (
```

```
);
```

```
endmodule
```

- **Ports**

- Address
- Write/Read command
- Input data (D)
- Output data (Q)

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```
module ram (  
    input  [9:0] addr  
    ,input          w  
    ,input  [7:0] d  
    ,output [7:0] q  
);
```

```
endmodule
```

- Ports

- Address
- Write/Read command
- Input data (D)
- Output data (Q)

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```
module ram (  
    input  [9:0] addr  
    ,input          w  
    ,input  [7:0] d  
    ,output [7:0] q  
);
```

```
endmodule
```

- **Ports**

- Address
- Write/Read command
- Input data (D)
- Output data (Q)

- **Memory shape**

- Assume data width = B
- Assume address bit = N $\rightarrow 2^N$ cells
- Shape = $2^N \times B$

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```
module ram (  
    input  [9:0] addr  
    ,input          w  
    ,input  [7:0] d  
    ,output [7:0] q  
);  
  
    reg [7:0] mem [2**10];  
  
endmodule
```

- Ports

- Address
- Write/Read command
- Input data (D)
- Output data (Q)

- Memory shape

- Assume data width = B
- Assume address bit = N $\rightarrow 2^N$ cells
- Shape = $2^N \times B$

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```

module ram (
  input  [9:0] addr  $\rightarrow N=10$ 
  ,input      w
  ,input  [7:0] d  $\rightarrow B=8$ 
  ,output [7:0] q
);

```

```

  reg [7:0] mem [2**10];

```

```

endmodule

```

- Ports

- Address
- Write/Read command
- Input data (D)
- Output data (Q)

- Memory shape

- Assume data width = B
- Assume address bit = N $\rightarrow 2^N$ cells
- Shape = $2^N \times B$
- In this example
 - $B = 8 \leftarrow [7:0] d, [7:0] q$
 - $N = 10 \leftarrow [9:0] addr$

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```

module ram (
  input  [9:0] addr  $\rightarrow N=10$ 
  ,input      w
  ,input  [7:0] d  $\rightarrow B=8$ 
  ,output [7:0] q
);

```

```

  reg [7:0] mem [2**10];

```

Power operator $\Rightarrow 2^{10} = 2^N$

```

endmodule

```

- Ports

- Address
- Write/Read command
- Input data (D)
- Output data (Q)

- Memory shape

- Assume data width = B
- Assume address bit = N $\rightarrow 2^N$ cells
- Shape = $2^N \times B$
- In this example
 - $B = 8 \leftarrow [7:0] d, [7:0] q$
 - $N = 10 \leftarrow [9:0] addr$

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```
module ram (  
    input  [9:0] addr  
    ,input          w  
    ,input  [7:0] d  
    ,output [7:0] q  
);  
  
    reg [7:0] mem [2**10];
```

```
endmodule
```

- **Packed/Unpacked arrays**

- Packed dimension : before signal name (the one seen so far)
- Unpacked dimension : after signal name

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```
module ram (  
    input  [9:0] addr  
    ,input          w  
    ,input  [7:0] d  
    ,output [7:0] q  
);  
  
    reg [7:0] mem [2**10];  
  
endmodule
```

- Packed/Unpacked arrays

- Packed dimension : before signal name (the one seen so far)
- Unpacked dimension : after signal name
- Both optional and both without limitations
 - Multiple dimensions of both kinds can be specified
- If only packed dimension(s) → packed array
- If only unpacked dimension(s) → unpacked array
- If both → mixed packed/unpacked array

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```

module ram (
    input  [9:0] addr
    ,input      w
    ,input  [7:0] d
    ,output [7:0] q
);

    reg [7:0] mem [2**10];

endmodule
  
```

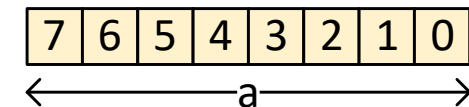
- Packed/Unpacked arrays

- Packed array

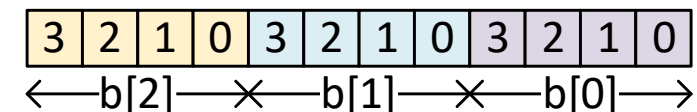
- Contiguous groups of bits
 - Just a smart organization

- Examples:

- reg [7:0] a



- reg [2:0] [3:0] b



Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```

module ram (
    input  [9:0] addr
    ,input      w
    ,input  [7:0] d
    ,output [7:0] q
);

    reg [7:0] mem [2**10];

endmodule
  
```

- Packed/Unpacked arrays

- Unpacked array

- Non-contiguous groups of bits

- Examples:

- wire a [3:0] →

a[3] 0

a[2] 0

a[1] 0

a[0] 0

b[0] 0

b[1] 0

b[2] 0

b[3] 0

- reg b [8] →

b[4] 0

b[5] 0

b[6] 0

b[7] 0

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```

module ram (
    input  [9:0] addr
    ,input      w
    ,input  [7:0] d
    ,output [7:0] q
);

    reg [7:0] mem [2**10];

endmodule
  
```

- Packed/Unpacked arrays

- In this example

mem[0]

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

mem[1]

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

mem[2]

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

⋮

⋮

mem[1022]

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

mem[1023]

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```
module ram (  
    input  [9:0] addr  
    ,input          w  
    ,input  [7:0] d  
    ,output [7:0] q  
);  
  
    reg [7:0] mem [2**10];  
  
endmodule
```

- Packed/Unpacked arrays

- Multidimensional packed/unpacked arrays
 - Access following the order of unpacked dimensions (right to left) then packed dimensions (left to right)
- Example:
 - reg [3:0][7:0] foo [8][24];
 - foo[7][23][3][7];
 - Access in this order: [8] → [24] → [3:0] → [7:0]

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```
module ram (  
    input  [9:0] addr  
    ,input          w  
    ,input  [7:0] d  
    ,output [7:0] q  
);  
  
    reg [7:0] mem [2**10];  
  
    always_comb  
        if(w)  
            mem[addr] = d;  
  
    assign q = mem[addr];  
  
endmodule
```

- Write logic

- Read logic

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```

module ram (
    input  [9:0] addr
    ,input      w
    ,input  [7:0] d
    ,output [7:0] q
);

    reg [7:0] mem [2**10];

    always_comb
        if(w)
            mem[addr] = d;

    assign q = mem[addr];
endmodule

```

- Simulation example



Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```

module ram (
    input  [9:0] addr
    ,input          w
    ,input  [7:0] d
    ,output [7:0] q
);

    reg [7:0] mem [2**10];

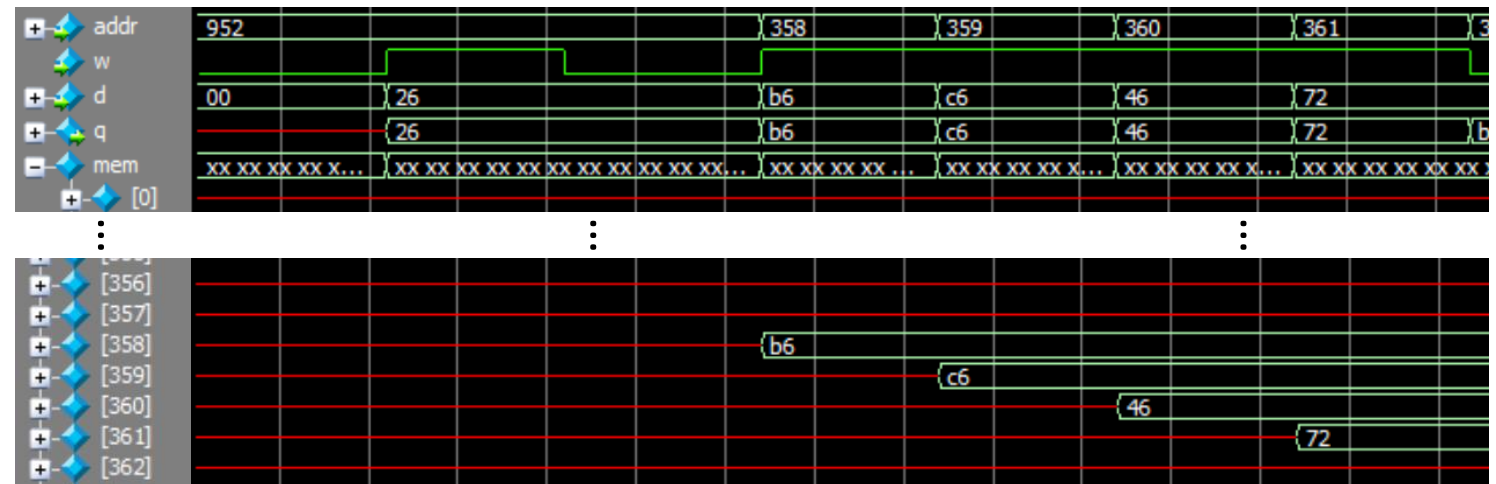
    always_comb
        if(w)
            mem[addr] = d;

    assign q = mem[addr];

endmodule
  
```

- Simulation example

- Write some locations and check by
 - Visual inspection of waveform
 - Reading and displaying written value



Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- RAM

```

module ram (
    input  [9:0] addr
    ,input      w
    ,input  [7:0] d
    ,output [7:0] q
);

    reg [7:0] mem [2**10];

    always_comb
        if(w)
            mem[addr] = d;

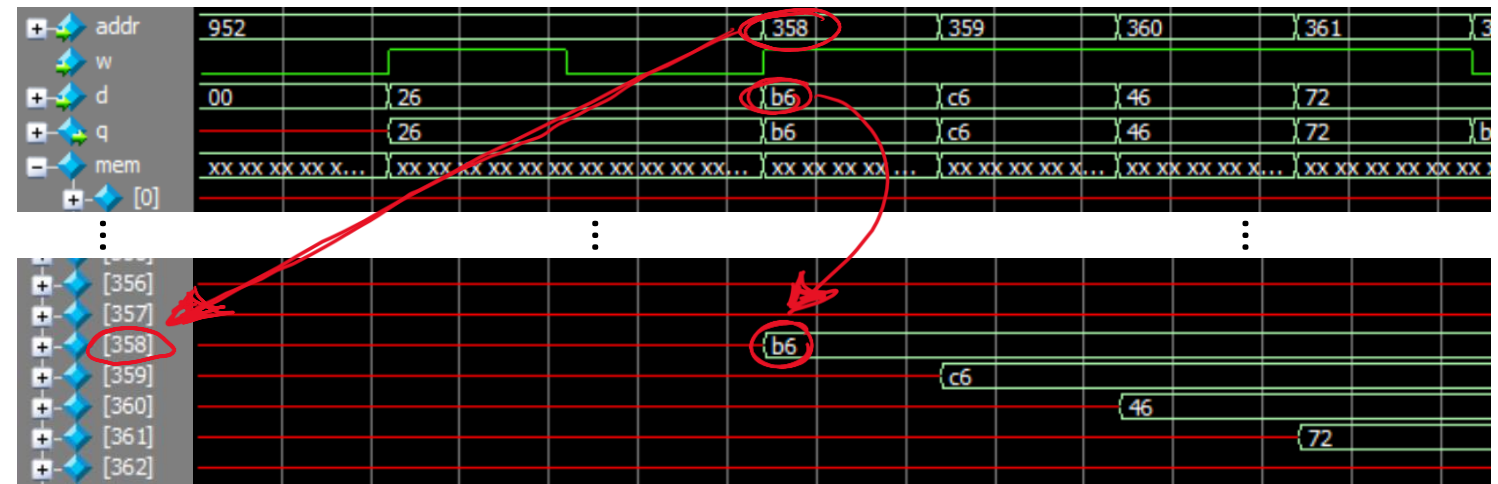
    assign q = mem[addr];

endmodule

```

- Simulation example

- Write some locations and check by
 - Visual inspection of waveform
 - Reading and displaying written value



Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim
 - ROM
 - Similar to RAM but
 - Without input data (D)
 - Without write/Read command
 - Only read logic

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- ROM

```
module rom (  
    input  [3:0] addr  
    ,output [31:0] q  
);  
  
    reg [31:0] mem [2**4];  
  
    assign q = mem[addr];  
  
endmodule
```

- Similar to RAM but
 - Without input data (D)
 - Without write/Read command
 - Only read logic

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- ROM

```
module rom (  
    input  [3:0] addr  
    ,output [31:0] q  
);  
  
    reg [31:0] mem [2**4];  
    assign q = mem[addr];  
  
endmodule
```

- Similar to RAM but
 - Without input data (D)
 - Without write/Read command
 - Only read logic
- But how to “write” content? In other words, how to initialize content of the ROM?

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- ROM

```
module rom (  
    input  [3:0] addr  
    ,output [31:0] q  
);  
  
    reg [31:0] mem [2**4];  
  
    assign q = mem[addr];  
  
    initial  
        $readmemh("ROM.hex", mem);  
  
endmodule
```

- Similar to RAM but
 - Without input data (D)
 - Without write/Read command
 - Only read logic
- But how to “write” content? In other words, how to initialize content of the ROM?
 - **readmem** command (+ initial block)

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- ROM

```
module rom (  
    input  [3:0] addr  
    ,output [31:0] q  
);  
  
    reg [31:0] mem [2**4];  
  
    assign q = mem[addr];  
  
    initial  
        $readmemh("ROM.hex", mem);  
  
endmodule
```

- Similar to RAM but
 - Without input data (D)
 - Without write/Read command
 - Only read logic
- But how to “write” content? In other words, how to initialize content of the ROM?
 - **readmem** command (+ initial block)
 - **\$readmemh/\$readmemb**(“file path”, <memory variable>)

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- ROM

```
module rom (  
    input  [3:0] addr  
    ,output [31:0] q  
);  
  
    reg [31:0] mem [2**4];  
  
    assign q = mem[addr];  
  
    initial  
        $readmemh("ROM.hex", mem);  
  
endmodule
```

- Similar to RAM but
 - Without input data (D)
 - Without write/Read command
 - Only read logic
- But how to “write” content? In other words, how to initialize content of the ROM?
 - **readmem** command (+ initial block)
 - **\$readmemh**/\$readmemb(“file path”, <memory variable>)
 - \$readmemh if digits in file are hexadecimal
 - \$readmemb if digits in file are binary

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- ROM

```

module rom (
    input    [3:0] addr
    ,output [31:0] q
);

    reg [31:0] mem [2**4];

    assign q = mem[addr];

    initial
        $readmemh("ROM.hex", mem);

endmodule
  
```

- File content

- Example ("ROM.hex")

1	922221D1
2	5BBADE69
3	58FFCD1C
4	E49329F4
5	4B03F8EE
6	DACA3E5C
7	7813CFE7
8	7CD9F913
9	95E968CC
10	0EF2B464
11	ED4C63C2
12	B364F698
13	7E0E047C
14	937ABAB8
15	9A104FF4
16	EDFDFB60

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- ROM

```

module rom (
    input  [3:0] addr
    ,output [31:0] q
);

    reg [31:0] mem [2**4];


    assign q = mem[addr];

    initial
        $readmemh("ROM.hex", mem);

endmodule
  
```

- File content

- Example ("ROM.hex")
 - Same bit width per line



1	922221D1
2	5BBADE69
3	58FFCD1C
4	E49329F4
5	4B03F8EE
6	DACA3E5C
7	7813CFE7
8	7CD9F913
9	95E968CC
10	0EF2B464
11	ED4C63C2
12	B364F698
13	7E0E047C
14	937ABAB8
15	9A104FF4
16	EDFDFB60

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- ROM

```

module rom (
    input  [3:0] addr
    ,output [31:0] q
);

    reg [31:0] mem [2**4];

    assign q = mem[addr];

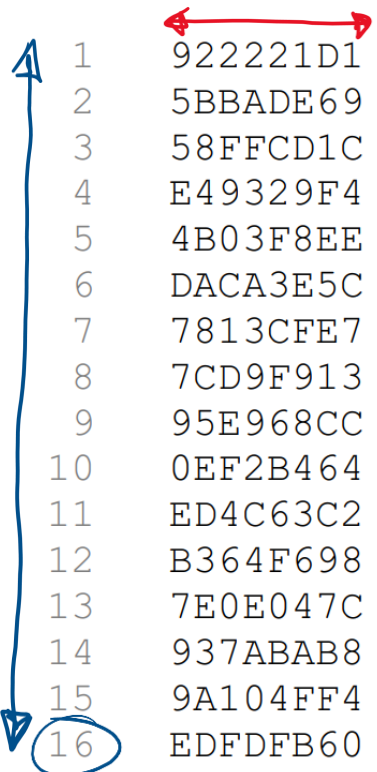
    initial
        $readmemh("ROM.hex", mem);

endmodule
  
```

- File content

- Example ("ROM.hex")

- Same bit width per line
- Each line = one address



1	922221D1
2	5BBADE69
3	58FFCD1C
4	E49329F4
5	4B03F8EE
6	DACA3E5C
7	7813CFE7
8	7CD9F913
9	95E968CC
10	0EF2B464
11	ED4C63C2
12	B364F698
13	7E0E047C
14	937ABAB8
15	9A104FF4
16	EDFDFB60

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- ROM

```
module rom (  
    input  [3:0] addr  
    ,output [31:0] q  
);  
  
    reg [31:0] mem [2**4];  
  
    assign q = mem[addr];  
  
    initial  
        $readmemh("ROM.hex", mem);  
  
endmodule
```

- ROM initialization inside an **initial** block
 - This is not synthesizable
 - This approach can be used only for modelling ROM
 - Synthesizable-code approaches available, but omitted for simplicity

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim

- ROM

```

module rom (
    input  [3:0] addr
    ,output [31:0] q
);

    reg [31:0] mem [2**4];

    assign q = mem[addr];

    initial
        $readmemh("ROM.hex", mem);

endmodule

```

- Simulation example

- Read each address and display value to manually check against the content of initialization file

VSIM 4> run -all	
# Addr. 0x0 >> Q = 922221d1	1 922221D1
# Addr. 0x1 >> Q = 5bbade69	2 5BBADE69
# Addr. 0x2 >> Q = 58ffcd1c	3 58FFCD1C
# Addr. 0x3 >> Q = e49329f4	4 E49329F4
# Addr. 0x4 >> Q = 4b03f8ee	5 4B03F8EE
# Addr. 0x5 >> Q = daca3e5c	6 DACA3E5C
# Addr. 0x6 >> Q = 7813cfe7	7 7813CFE7
# Addr. 0x7 >> Q = 7cd9f913	8 7CD9F913
# Addr. 0x8 >> Q = 95e968cc	9 95E968CC
# Addr. 0x9 >> Q = 0ef2b464	10 0EF2B464
# Addr. 0xa >> Q = ed4c63c2	11 ED4C63C2
# Addr. 0xb >> Q = b364f698	12 B364F698
# Addr. 0xc >> Q = 7e0e047c	13 7E0E047C
# Addr. 0xd >> Q = 937abab8	14 937ABAB8
# Addr. 0xe >> Q = 9a104ff4	15 9A104FF4
# Addr. 0xf >> Q = edfdfb60	16 EDFDFB60

VS.

Exercise with SystemVerilog

- Implementation of RAM/ROM (model) and simulation with Modelsim
 - You can find all the files about this exercise in the dedicated folder on the Team of the course
 - File > Electronics Systems module > Crocetti > Exercises > 3.7
 - ‘ram’ sub-folder, for RAM-related SV files
 - ‘rom’ sub-folder, for ROM-related SV files
 - Try to design and simulate on your own
 - An example of ROM initialization file in ‘rom > ROM.hex’



Thank you for your attention

Luca Crocetti
(luca.crocetti@unipi.it)