COMS12200
Introduction to Computer Architecture

Dr. Cian O'Donnell

cian.odonnell@bristol.ac.uk

Topic 7: Memory paradigms

Topics

1. Data, Control and Instructions
2. Memory
3. Execution cycle
4. Processor control flow
5. State machines and decoding
6. Machine types
7. Memory paradigms

Topics

1. Data, Control and Instructions
2. Memory
3. Execution cycle
4. Processor control flow
5. State machines and decoding
6. Machine types
7. Memory paradigms

7. Memory architecture paradigms

We will compare two different paradigms for the memory subsystem:

- Harvard
- Von Neumann

Why do we need a paradigm?

- In order to provide a clear and consistent mode for both a programmer and a hardware designer, we need to choose an operational paradigm.

- The paradigm determines how memories are laid out and how data is treated.

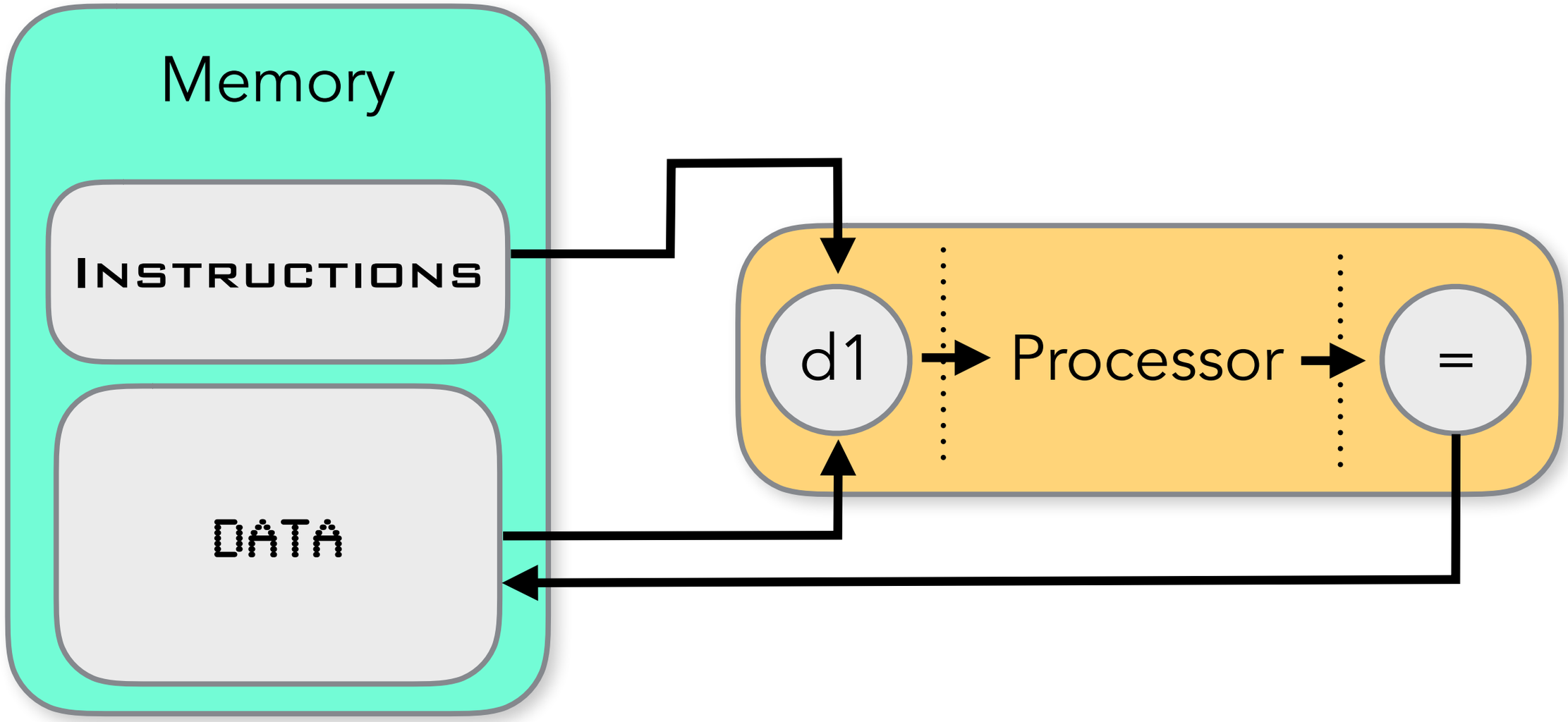
The Harvard architecture

Harvard architecture

- A processing unit with two distinct components
  1. Instruction memory
  2. Data memory

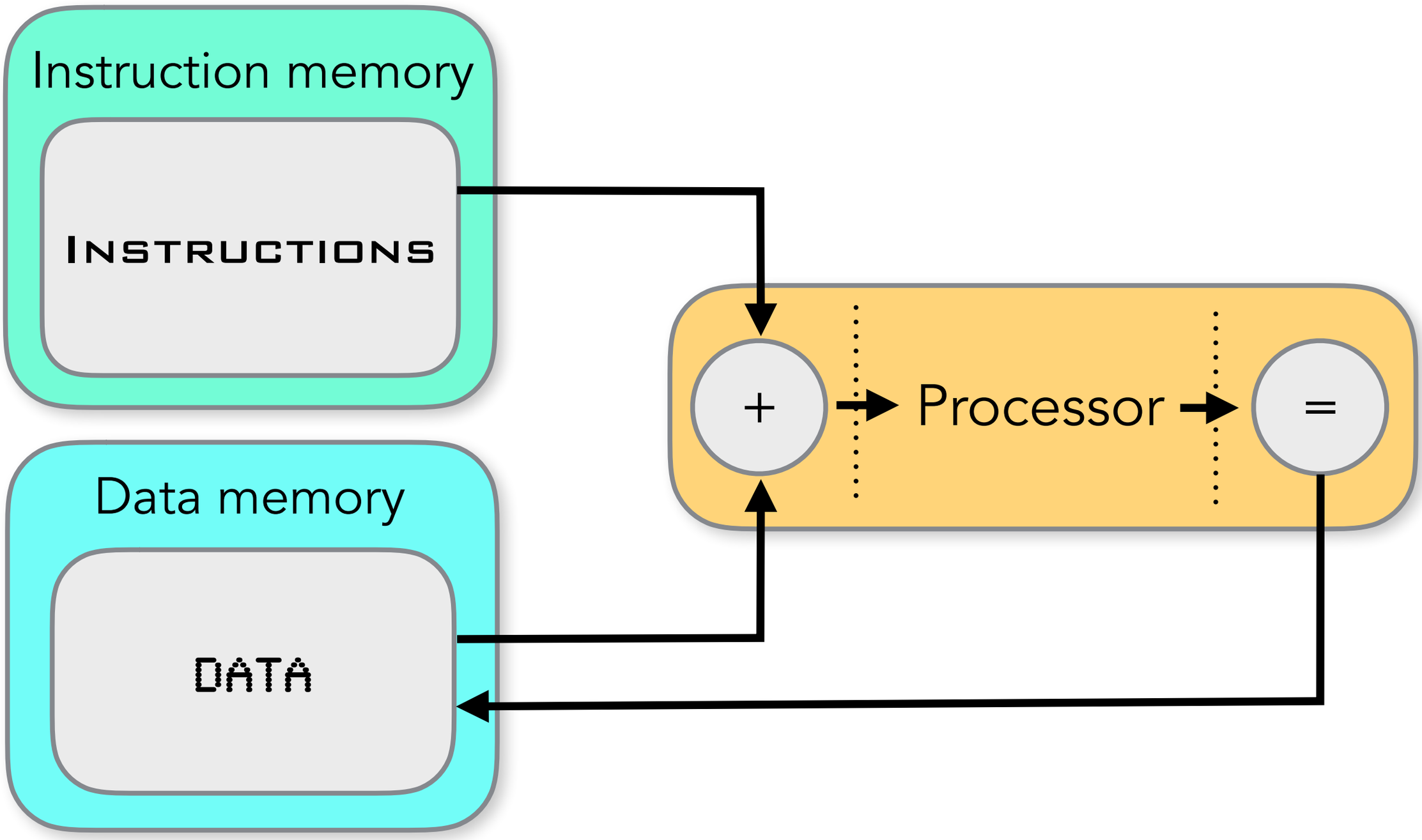
- They are segregated in hardware and have distinct roles.

Recap



Harvard architecture

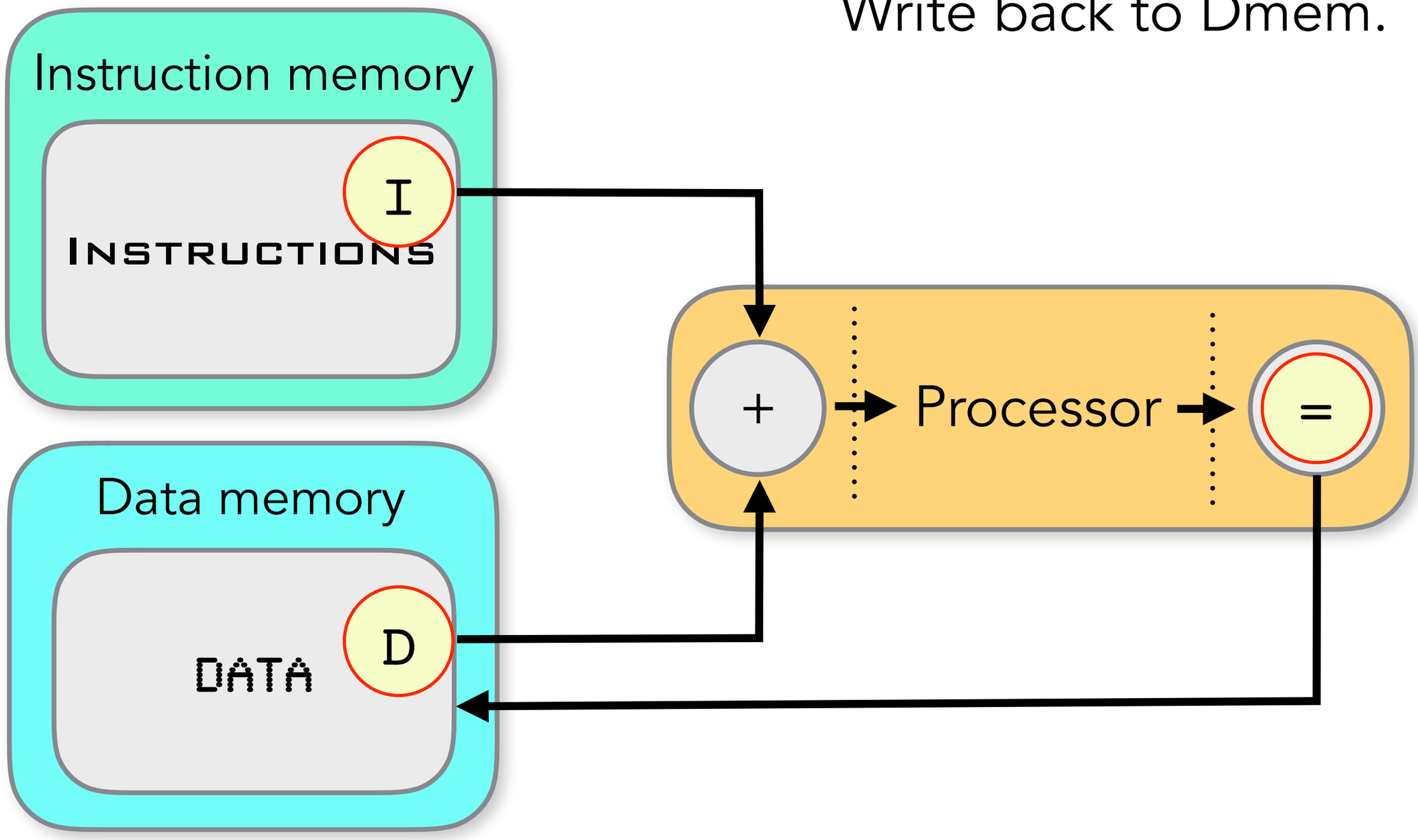
Key concept: separate instructions and data memories



Harvard architecture

Fetch instruction from IMem. Fetch data from Dmem.

Write back to Dmem.



Harvard advantages

- The advantage is segregation.
- Separate memories can be optimised for their relative needs. For example, instruction memory can be *read-only* and *low-power*.
- Independent connections to transfer data more rapidly.
- Instructions are ‘safe’ from accidental overwriting (containment)

Harvard **dis**advantages

- What happens if one memory is full and the other is almost empty? Inefficient.

- Some resources need duplicating:
  - Connections
  - Need a pointer to keep track of each memory

Harvard execution example

- Soon we'll see an example of the Harvard architecture at work.

- First let's define the meaning of certain values as instructions.

A simple instruction set

Instruction code	Argument	Meaning
1	`N`	Move `N` into `ACC`
2	`N`	Store `ACC` in `MEM[N]`
3	`N`	Add `ACC` to `MEM[N]`
4	`N`	Sub `MEM[N]` from `ACC`
5	`-`	Stop processing

Harvard execution example

PC 

Accumulator 

Instruction memory

Address	Content
0	14
1	21
2	12
3	31
4	20

Data memory

Address	Content
0	
1	
2	
3	
4	

Harvard execution example

PC 0

Accumulator 4

Instruction memory

Address	Content
0	14
1	21
2	12
3	31
4	20

Data memory

Address	Content
0	
1	
2	
3	
4	

Harvard execution example

PC 1

Accumulator 4

Instruction memory

Address	Content
0	14
1	21
2	12
3	31
4	20

Data memory

Address	Content
0	
1	4
2	
3	
4	

Harvard execution example

PC 2

Accumulator 2

Instruction memory

Address	Content
0	14
1	21
2	12
3	31
4	20

Data memory

Address	Content
0	
1	4
2	
3	
4	

Harvard execution example

PC 3

Accumulator 2

Instruction memory

Address	Content
0	14
1	21
2	12
3	31
4	20

Data memory

Address	Content
0	
1	4+2
2	
3	
4	

Harvard execution example

PC 3

Accumulator 2

Instruction memory

Address	Content
0	14
1	21
2	12
3	31
4	20

Data memory

Address	Content
0	
1	6
2	
3	
4	

Harvard execution example

PC 4

Accumulator 2

Instruction memory

Address	Content
0	14
1	21
2	12
3	31
4	20

Data memory

Address	Content
0	2
1	6
2	
3	
4	

The von Neumann architecture

John von Neumann
1903-1957

von Neumann architecture

- The von Neumann architecture makes a subtle but very important change to the layout of data and instructions: they are unified.

- Hence this approach is also called the “unified memory architecture”.

Information equality

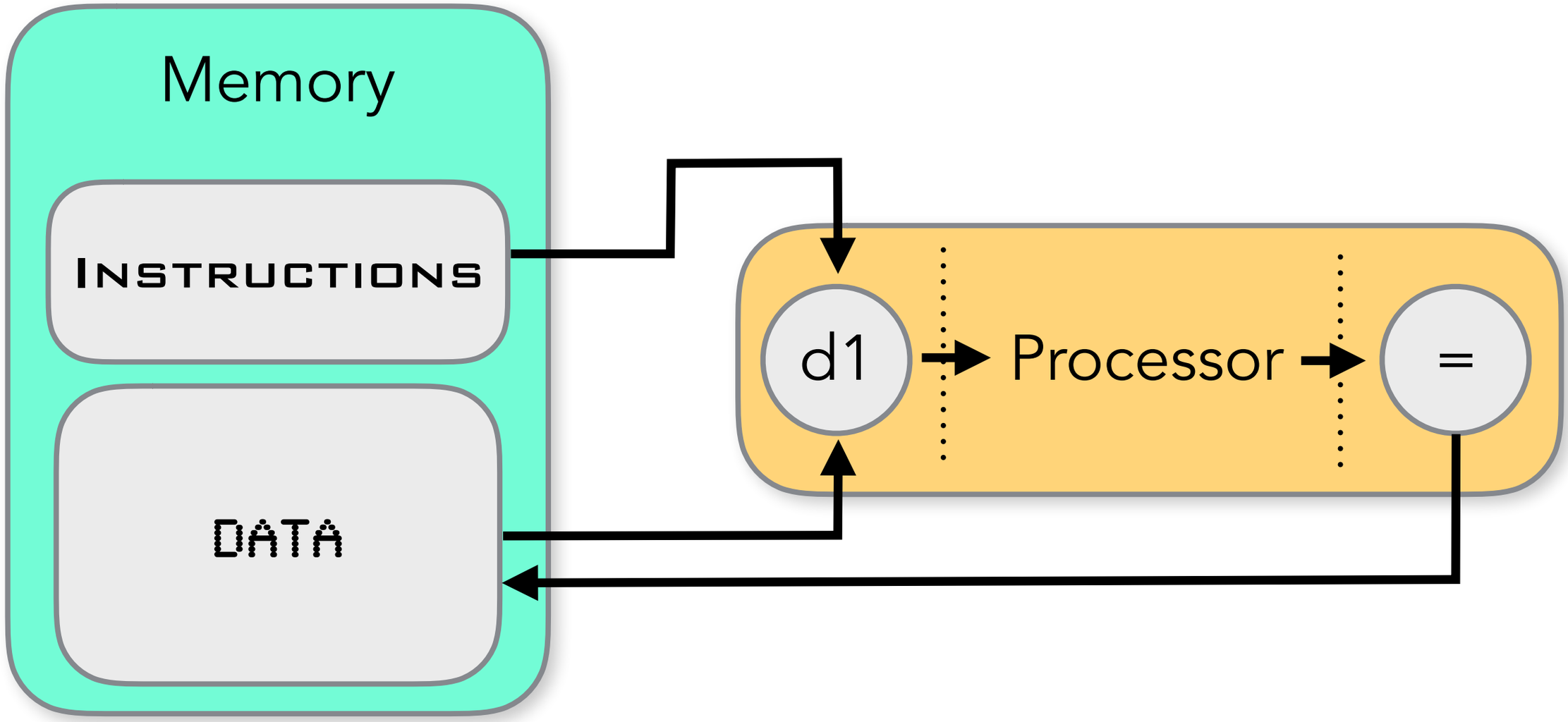
- This architecture leverages a very simple concept:

  Data ≡ Instruction ≡ Information

- There's no difference except for the semantics we give.
- This idea is at the heart of the von Neumann paradigm.

von Neumann architecture

We've seen this before, in our simple fetch-execute machine



von Neumann architecture

- Because of its flexibility, the von Neumann architecture is the most common instantiation of a processor-memory architecture.

- The Harvard architecture does have uses in high-performance systems with limited code-bases.

von Neumann architecture

- The key feature of von Neumann is that the instruction and the data memories are unified.

- This increases efficiency of memory (in terms of space usage)

- BUT, it adds a bottleneck.

- Raises the possibility of flexibility and run-time modifications of code.

von Neumann bottleneck

- This is name given to the slowdown due to having a shared bus for data and instructions from memory to the processor.

- Problem has increased over the last few decades because processors have got faster and memory capacity has grown larger, but data throughput rates haven’t kept up.

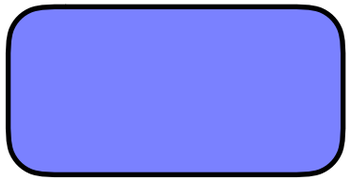
- Workarounds:
  - Caching (more on memory hierarchy in TB2)
  - Harvard architecture in cache
  - Branch prediction

von Neumann architecture

- We'll first re-do the Harvard example using the von Neumann instruction set.

- Then we'll mention how code modification can work and how it can be useful.

von Neumann execution example

PC 

Accumulator 

Memory

Address	Content
0	14
1	21329
2	12
3	31329
4	21328
…	…
1328	
1329	
1330	
1331	

von Neumann execution example

PC 0

Accumulator 4

Memory

Address	Content
0	14
1	21329
2	12
3	31329
4	21328
…	…
1328	
1329	
1330	
1331	

von Neumann execution example

PC 1 Accumulator 4

Memory

Address	Content
0	14
1	21329
2	12
3	31329
4	21328
…	…
1328	
1329	4
1330	
1331	

von Neumann execution example

PC 2

Accumulator 2

Memory

Address	Content
0	14
1	21329
2	12
3	31329
4	21328
…	…
1328	
1329	4
1330	
1331	

von Neumann execution example

PC 3

Accumulator 2

Memory

Address	Content
0	14
1	21329
2	12
3	31329
4	21328
…	…
1328	
1329	6
1330	
1331	

von Neumann execution example

PC **4**

Accumulator **2**

Memory

Address	Content
0	14
1	21329
2	12
3	31329
4	21328
…	…
1328	2
1329	6
1330	
1331	

Self-modifying code

- Since the instructions and data are equal under the von Neumann paradigm, we can do some very interesting things.

- We can manipulate the instruction space.
  - Instructions can be the source and destination for instructions.

- We can execute data.

Self-modifying code

- Advantages: can be used to write compact, fast code. Fun.
- Disadvantages: complicated to analyse and optimise code, since the instructions in the source code are not necessarily the same as those that will get executed. Can lead to crashing.

7. Memory architecture paradigms

- We've seen the two main processor memory paradigms: Harvard and von Neumann.

- The main difference is that instruction and data memory are separate in Harvard but unified in von Neumann (both physically and symbolically).

- von Neumann allows for self-modifying code: run-time interpretation, just-in-time compiling.

Topics

1. Data, Control and Instructions
2. Memory
3. Execution cycle
4. Processor control flow
5. State machines and decoding
6. Machine types
7. Memory paradigms

Topics

1. Data, Control and Instructions
2. Memory
3. Execution cycle
4. Processor control flow
5. State machines and decoding
6. Machine types
7. Memory paradigms