COMS12200
Introduction to Computer Architecture

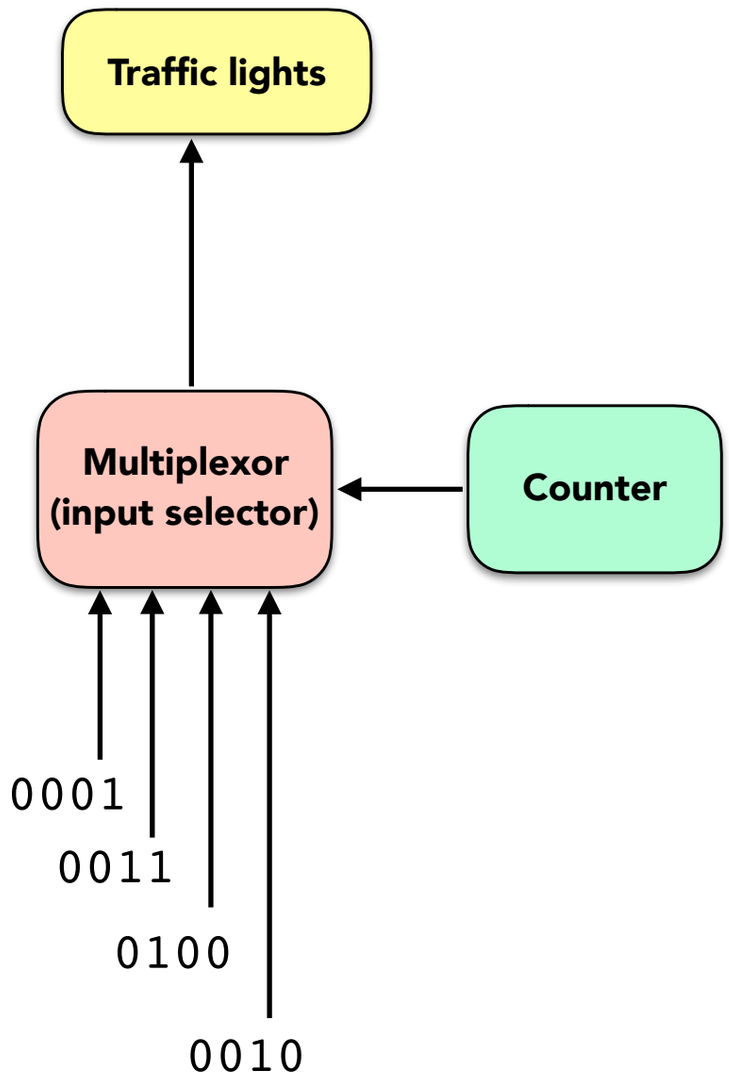
Dr. Cian O'Donnell

cian.odonnell@bristol.ac.uk

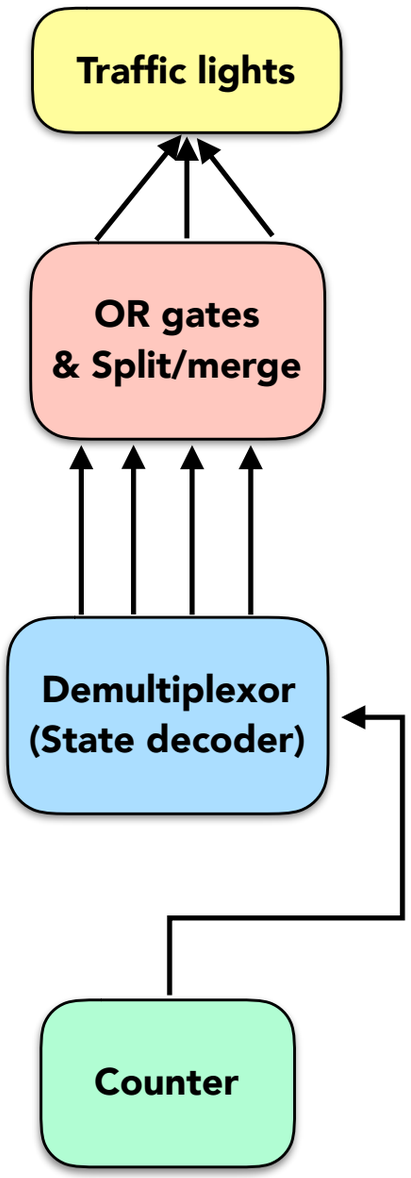
Topic 6: Machine types

Lab 3 recap

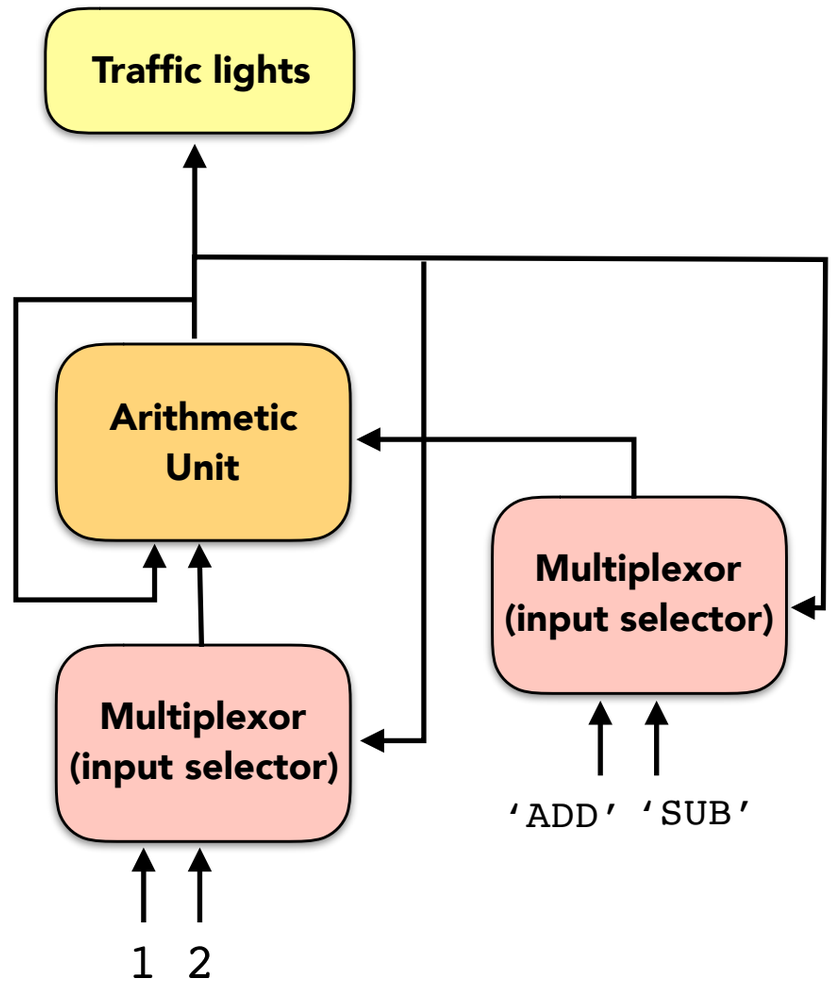
Task 1



Task 2



Task 3



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

5. State machines and decoding

Summary from previous lecture

- State machines are a natural way to model processor flow during program operation.

- Decoding is a key step that transforms instruction information into a set of control signals.

- Showed a method for generating control signals by using Boolean algebra and minterms.

- Program flow can be variable: depending on the output of computations, we can alter data or control signals for future instructions.

5. State machines and decoding

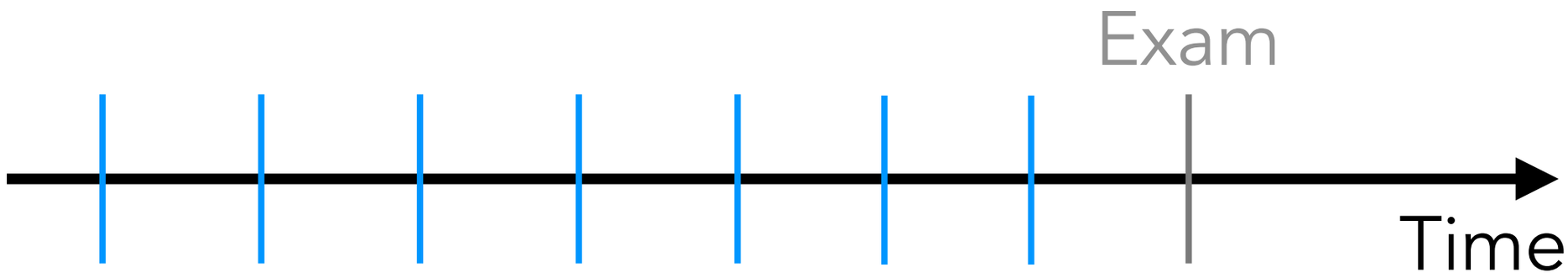
Tips for controlling state machines

1. MUXs are useful for selecting inputs
2. DEMUXs are useful for decoding outputs
3. OR gates are useful for combining states and producing feedback signals.

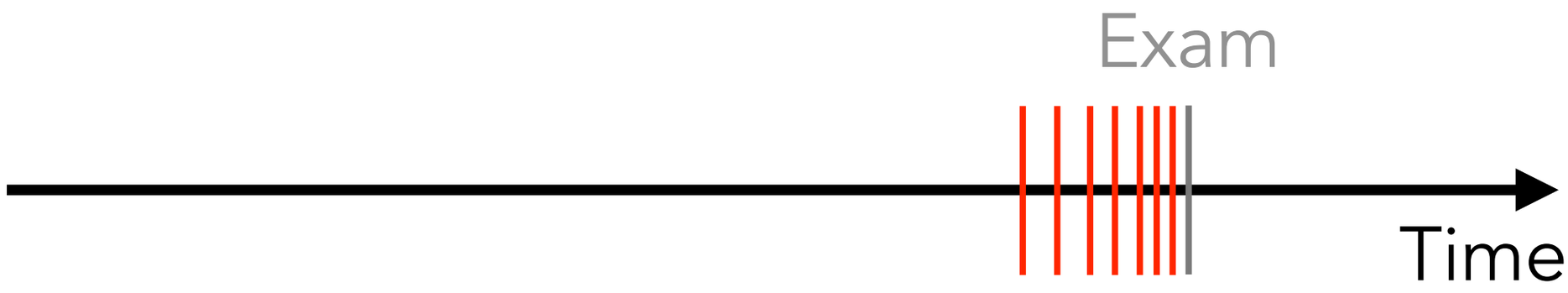
DIVERSION

Spaced vs crammed learning

Spaced learning: BETTER long-term memory

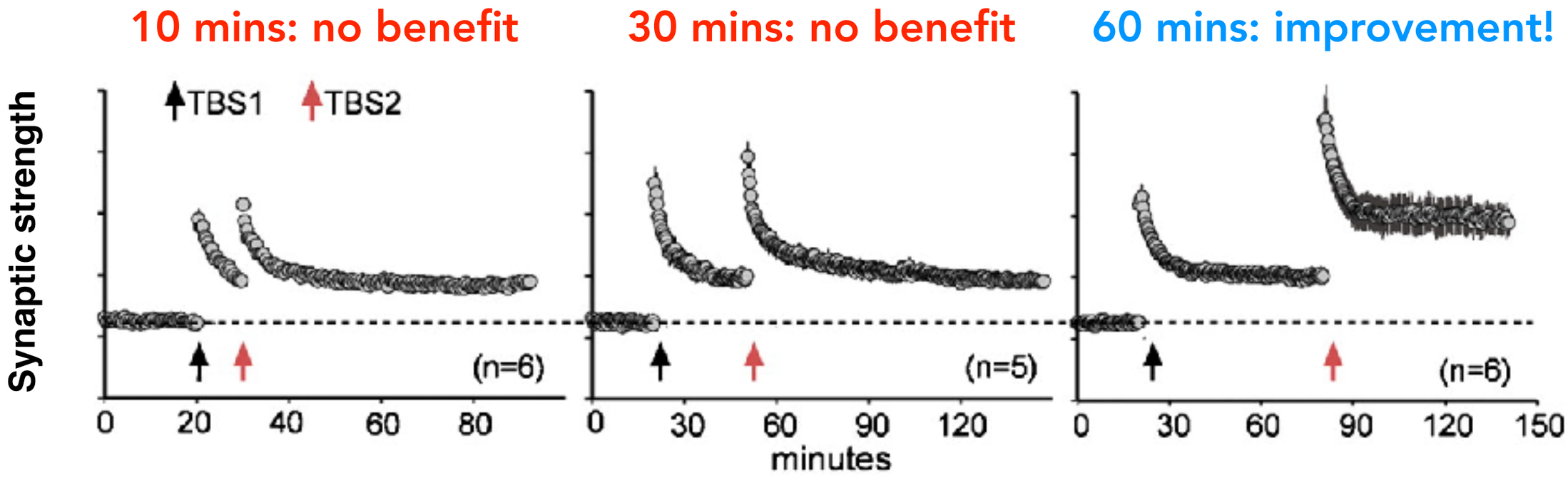


Cramming: WORSE long-term memory



Spaced vs crammed learning

Brain mechanisms



Spaced vs crammed learning

Take-away messages

- Repeat learning at various intervals.
- Test yourself (attempt questions, explain concepts to a friend. Revise in the same way that you will be tested.)
- Pay attention! You can't remember something if you never learned it in the first place.

The Seductive Allure of Neuroscience Explanations

Deena Skolnick Weisberg, Frank C. Keil, Joshua Goodstein, Elizabeth Rawson, and Jeremy R. Gray

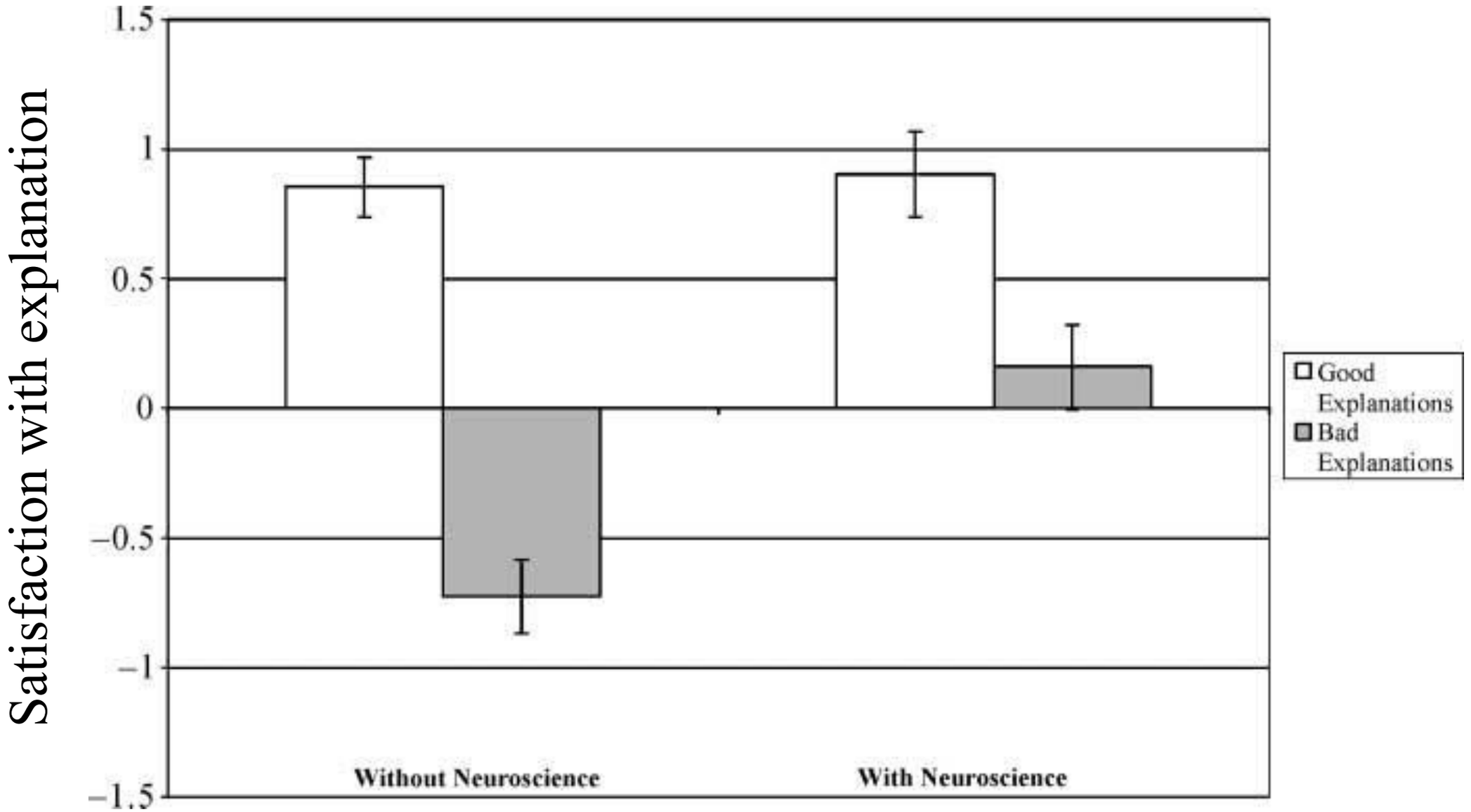
Abstract

■ Explanations of psychological phenomena seem to generate more public interest when they contain neuroscientific information. Even irrelevant neuroscience information in an explanation of a psychological phenomenon may interfere with people's abilities to critically consider the underlying logic of this explanation. We tested this hypothesis by giving naïve adults, students in a neuroscience course, and neuroscience experts brief descriptions of psychological phenomena followed by one of four types of explanation, according to a 2 (good explanation vs. bad explanation) × 2 (without neuroscience vs. with neuroscience) design. Crucially, the neuroscience information was irrelevant to the logic of the explanation, as confirmed by the expert subjects. Subjects in all three groups judged good explanations as more satisfying than bad ones. But subjects in the two nonexpert groups additionally judged that explanations with logically irrelevant neuroscience information were more satisfying than explanations without. The neuroscience information had a particularly striking effect on nonexperts' judgments of bad explanations, masking otherwise salient problems in these explanations. ■

INTRODUCTION

Although it is hardly mysterious that members of the public should find psychological research fascinating,

(Lombrozo & Carey, 2006; Kelemen, 1999). People also tend to rate longer explanations as more similar to experts' explanations (Kikas, 2003), fail to recognize circu-


Satisfaction with explanation
1.5
1
0.5
0
−0.5
−1
−1.5
Without Neuroscience
With Neuroscience
Good Explanations
Bad Explanations


Diversion
ENDS

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

Processor architectures

- What happens to a program internally in the processor when it is executed, and how is computation performed?

- There are three common flavours of implementation paradigm:
  - Accumulator machine
  - Stack machine
  - Register machine

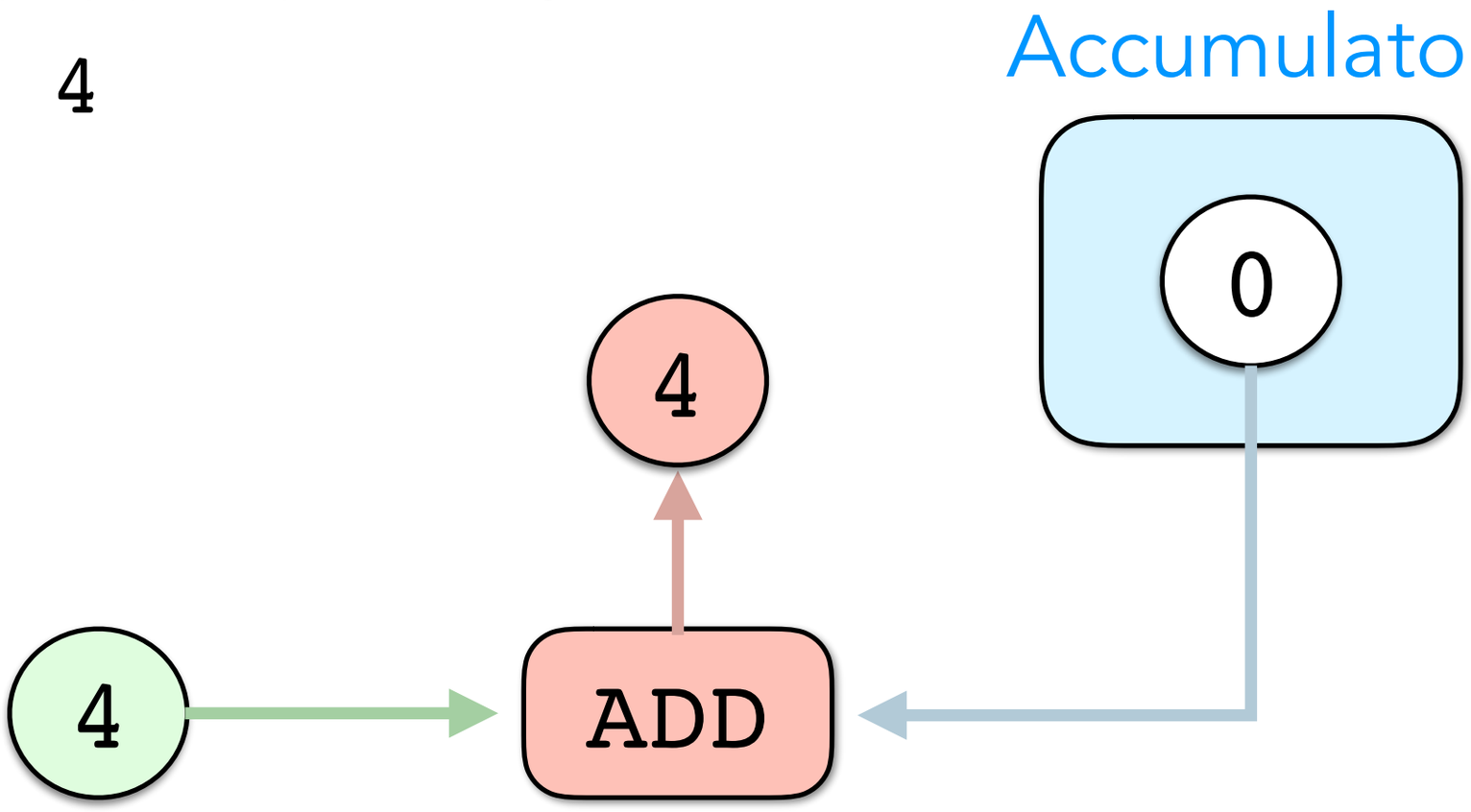
Accumulator machines

- There is a central storage element (register) called "the accumulator"
- All instructions manipulate this store (but may also access memory).
- All data must flow through this store.
- They are simple, fast, but require many instructions to do tasks.

Accumulator machines

e.g. “4 + 2”

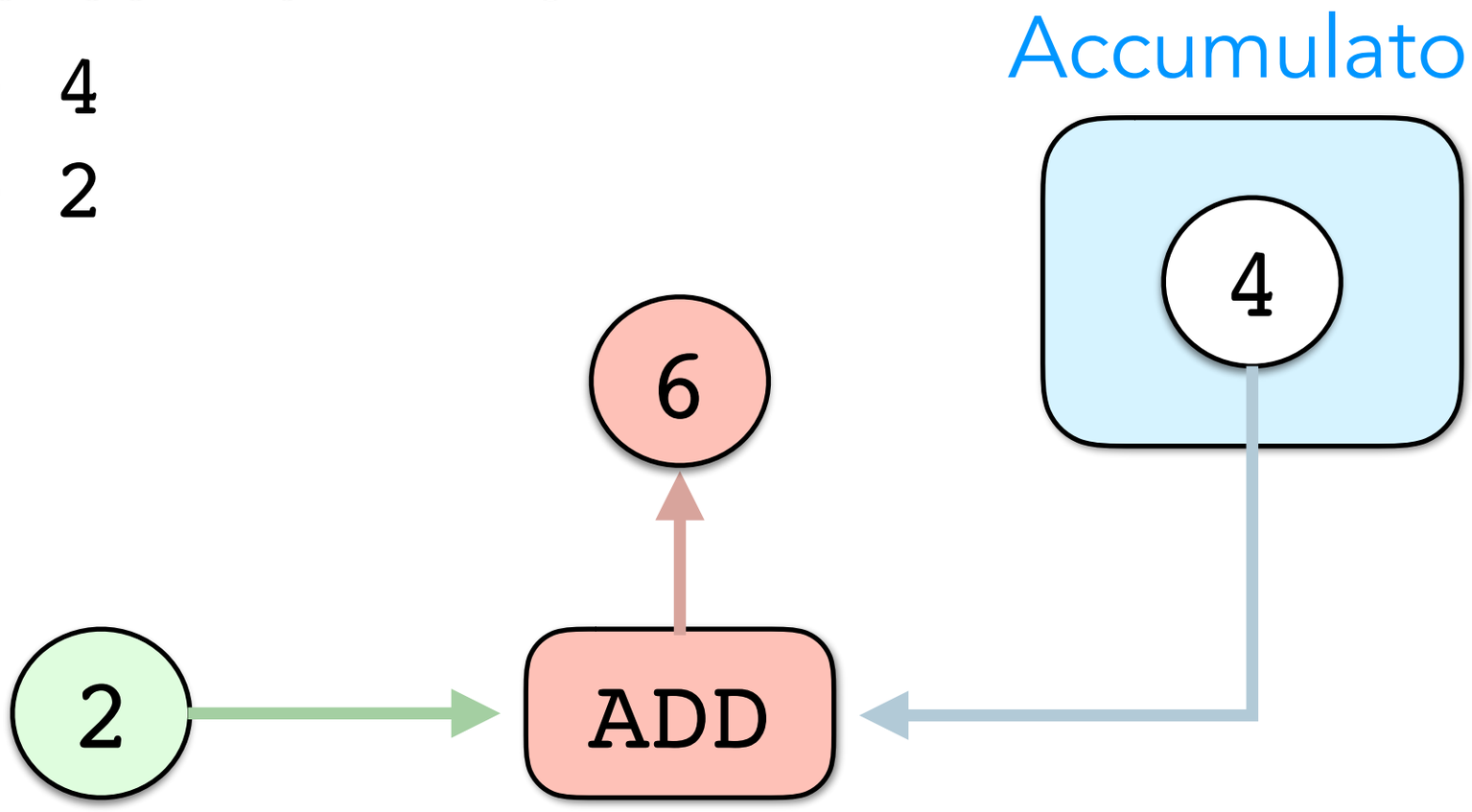
- Initialise with `ZERO`
- `ADD 4`



Accumulator machines

e.g. “4 + 2”

- Initialise with `ZERO`
- `ADD 4`
- `ADD 2`

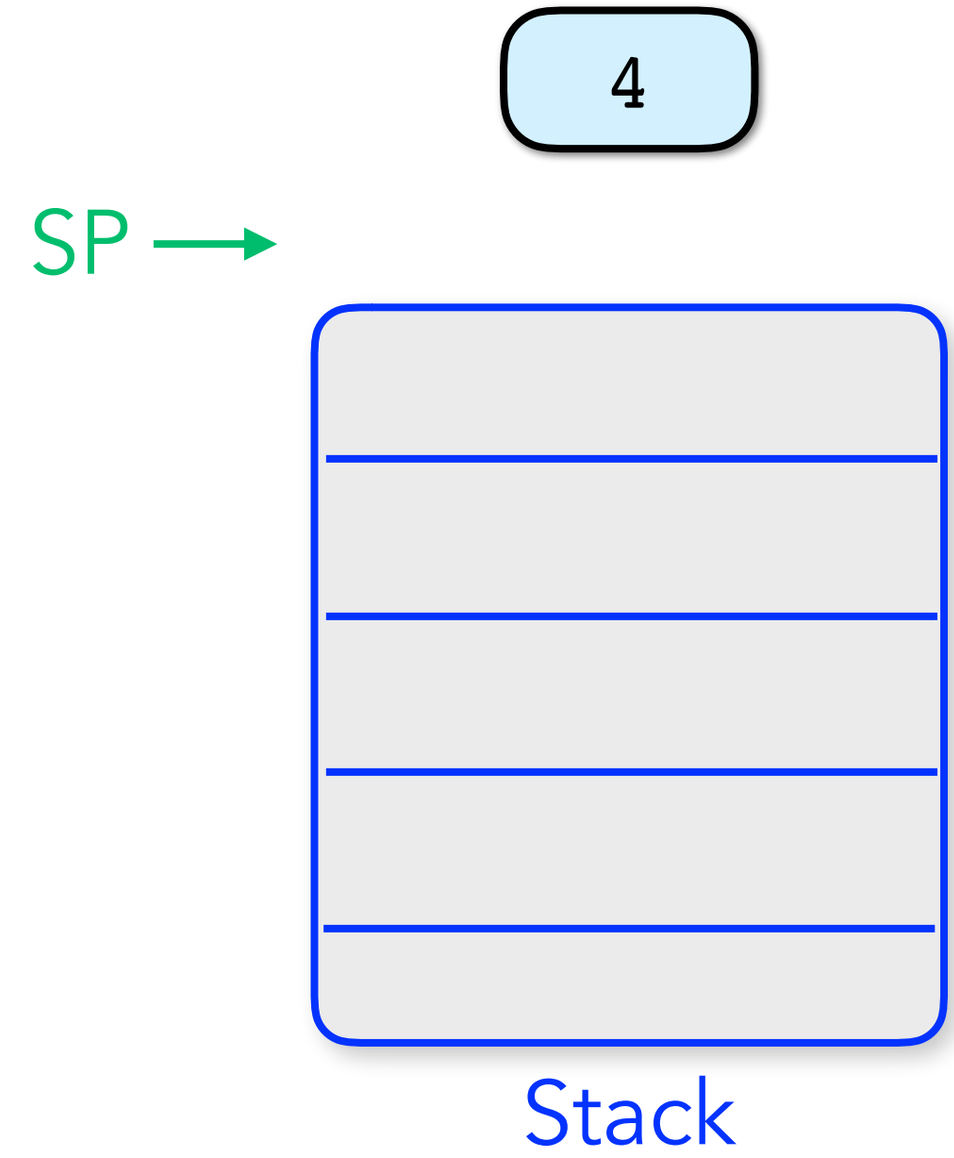


Stack machines

- A stack machine uses a processor stack to store information during execution.
- A stack pointer (a hidden register) keeps track of the current top of the stack.
- All operations modify the stack or stack pointer.
- The stack is a bottleneck

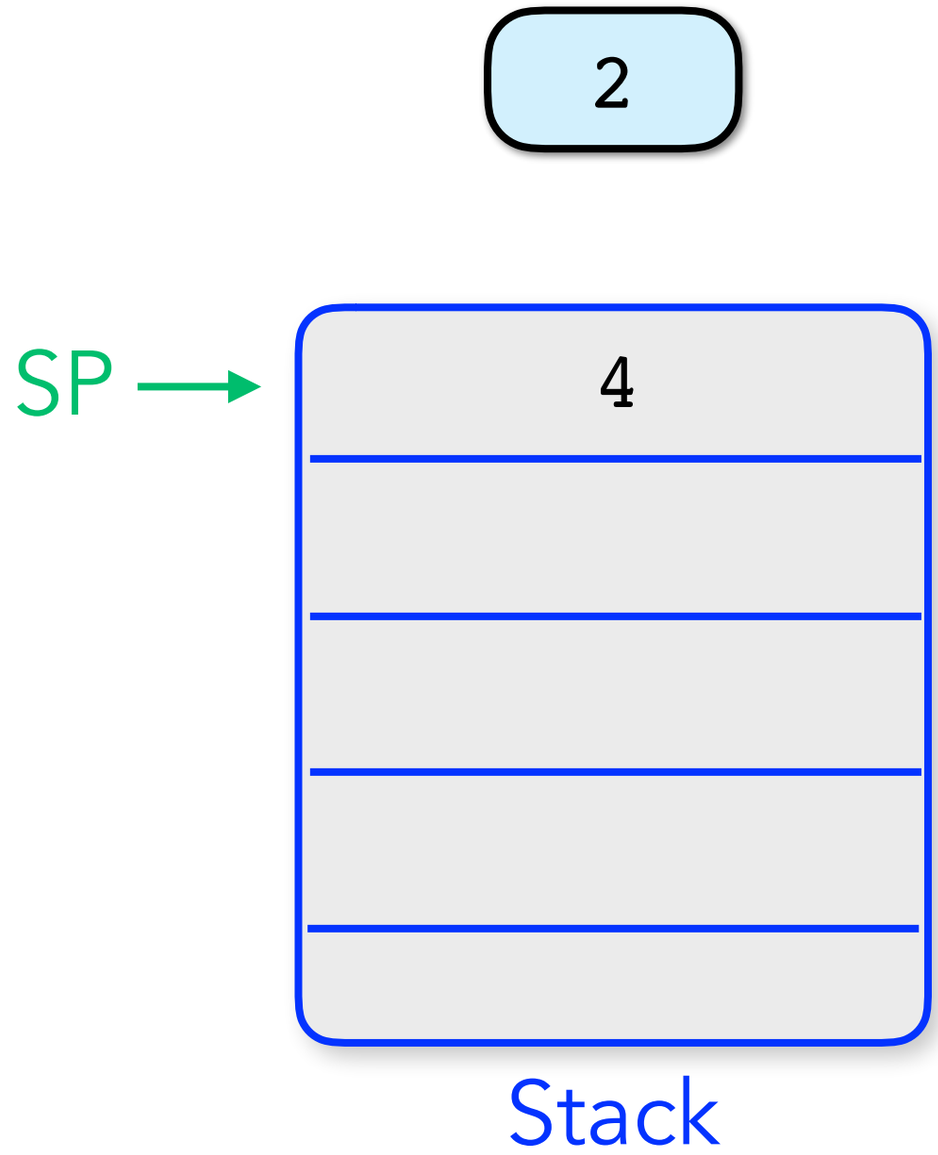
Stack machines

e.g. “4 + 2”



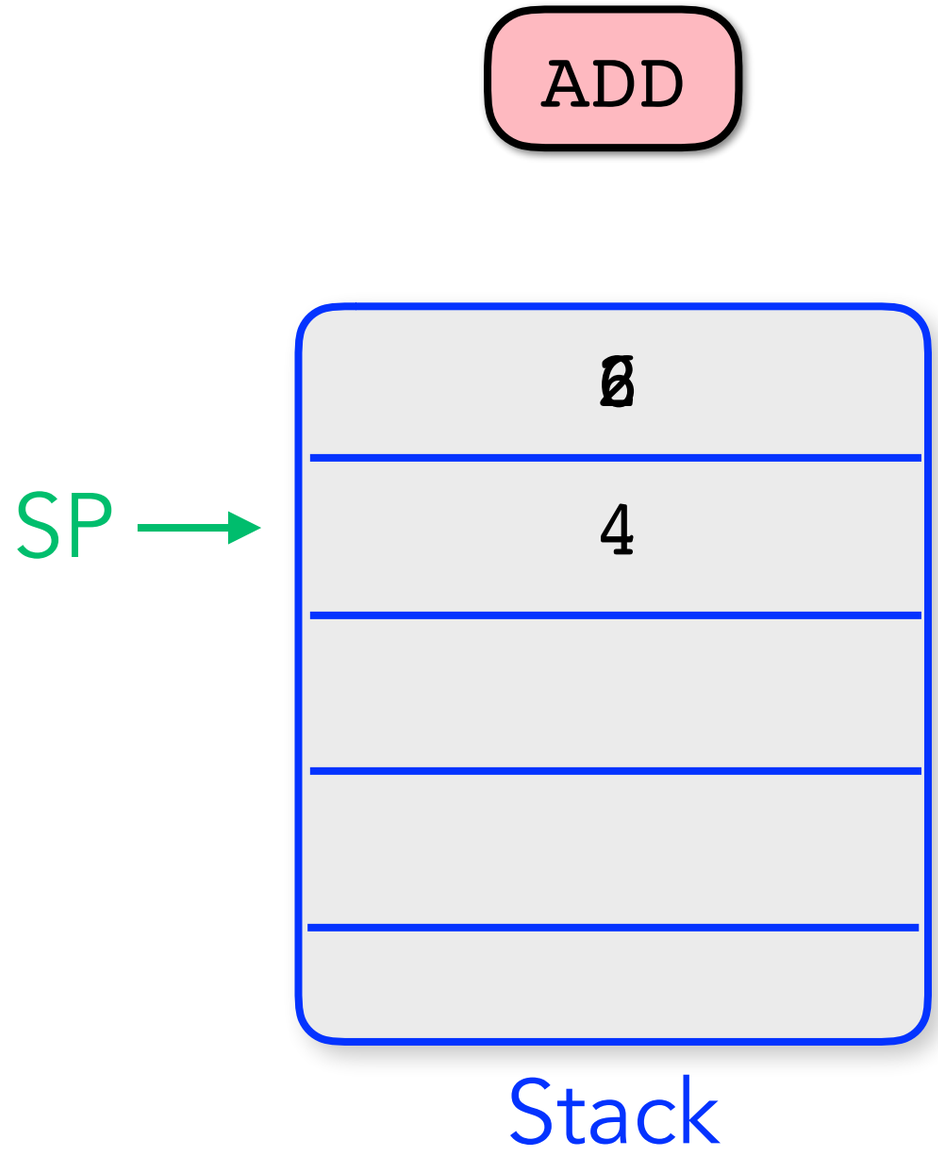
Stack machines

e.g. “4 + 2”



Stack machines

e.g. “4 + 2”



Reverse Polish notation

- We just saw that operations must be re-ordered to work correctly.

- We can write them down in a format that makes sense for a stack machine, called “Reverse Polish” notation.

- It’s a powerful way of mapping the problem and also for reasoning about stack machines.

Reverse Polish notation

- Simple rules of associativity and precedence.
- Operators always appear after their input data.
- Example:

  Desired operation: `4 + 2`

  Reverse Polish: `4, 2, '+'`

- '+' is a binary operator, so consumes the two previous data values.

Register machines

- Register machines use multiple storage registers to store sets of data (=variables) at the same time.

- Instructions can typically access multiple registers at once.

- Results are also committed to registers.

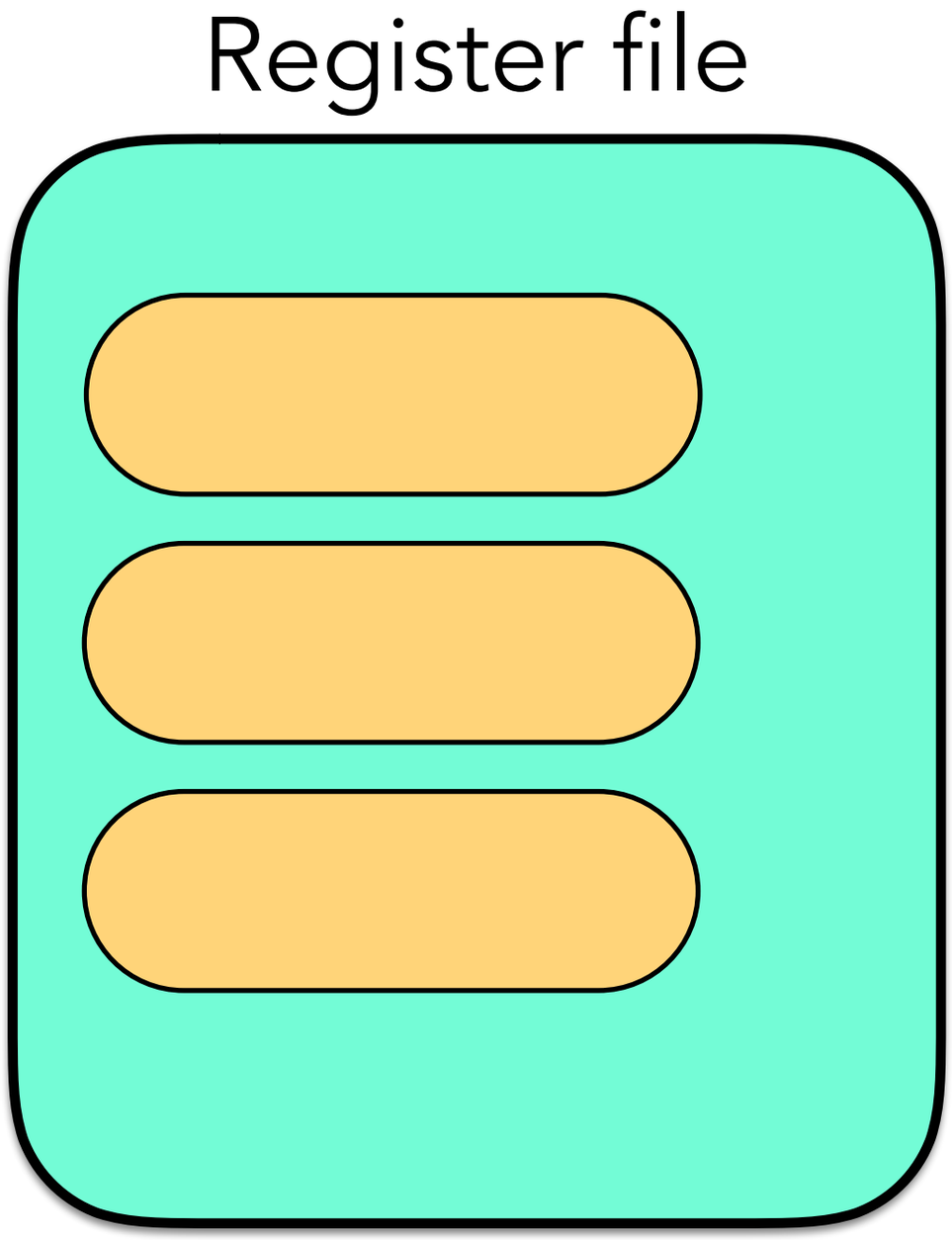
Register machines

If we want to do “4+2”:

```
MOVE (r0, 4)

MOVE (r1, 2)

ADD (r0, r0, r1)
```



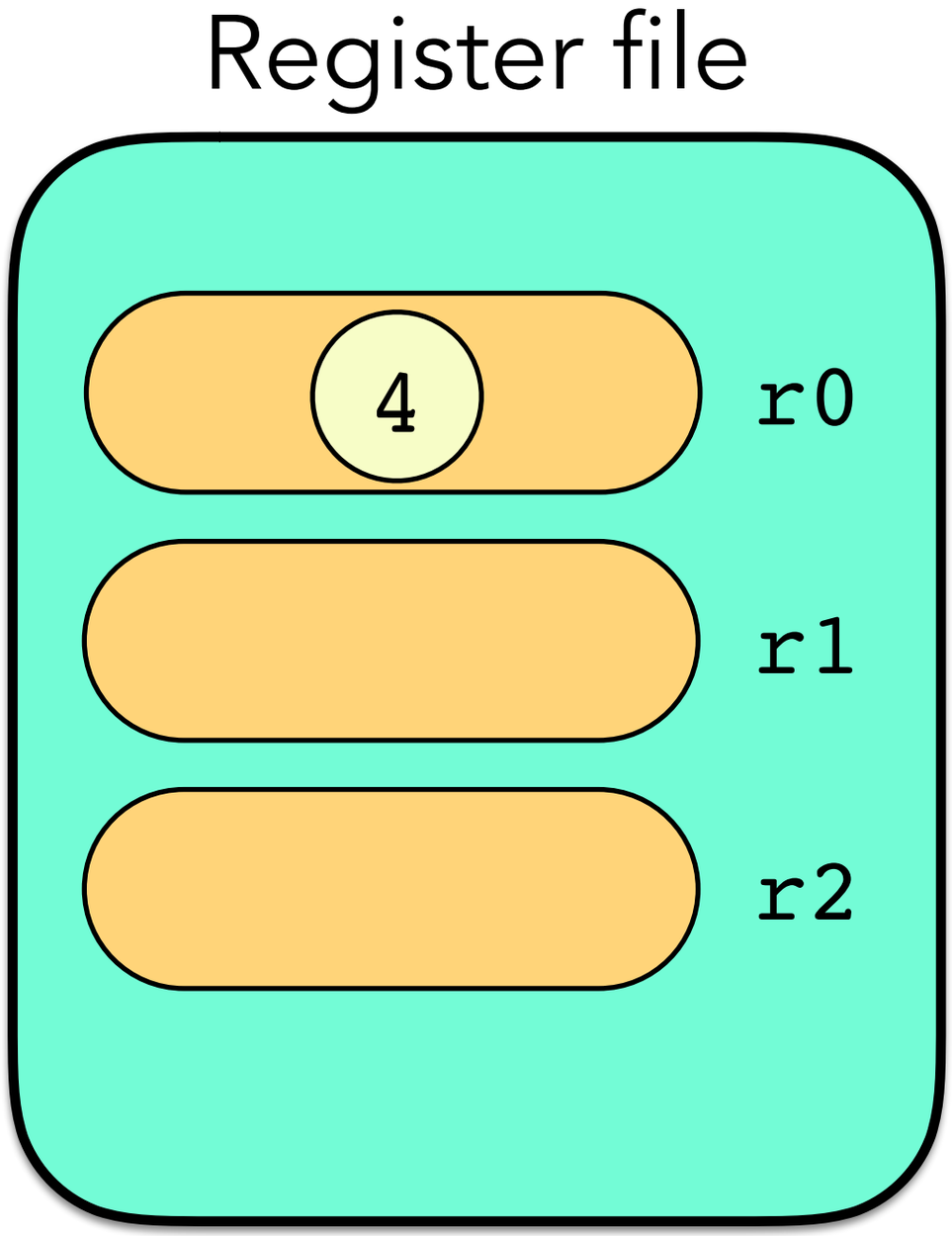
Register machines

If we want to do "`4+2`":

`MOVE (r0, 4)`

`MOVE (r1, 2)`

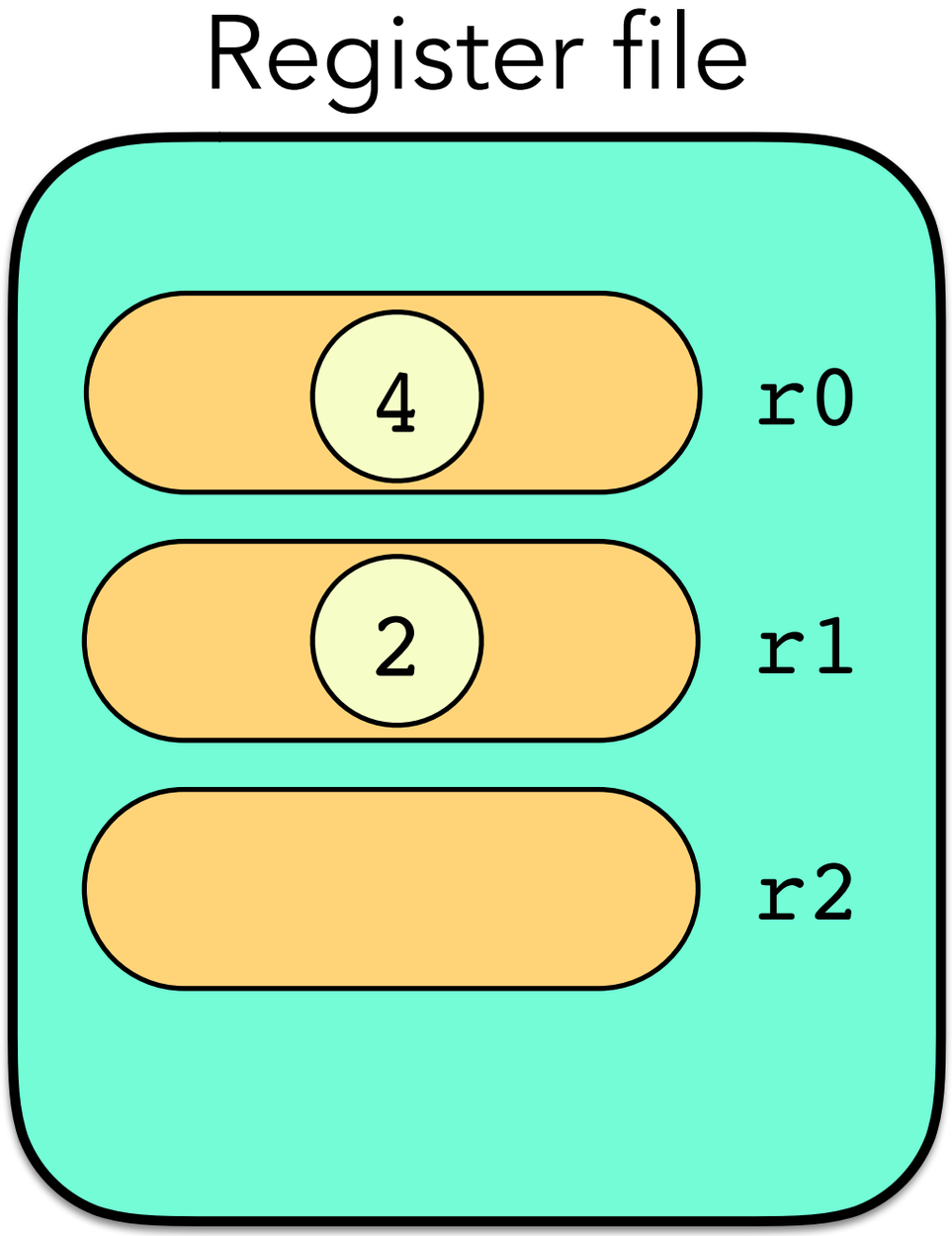
`ADD (r0, r0, r1)`



Register machines

If we want to do "`4+2`":

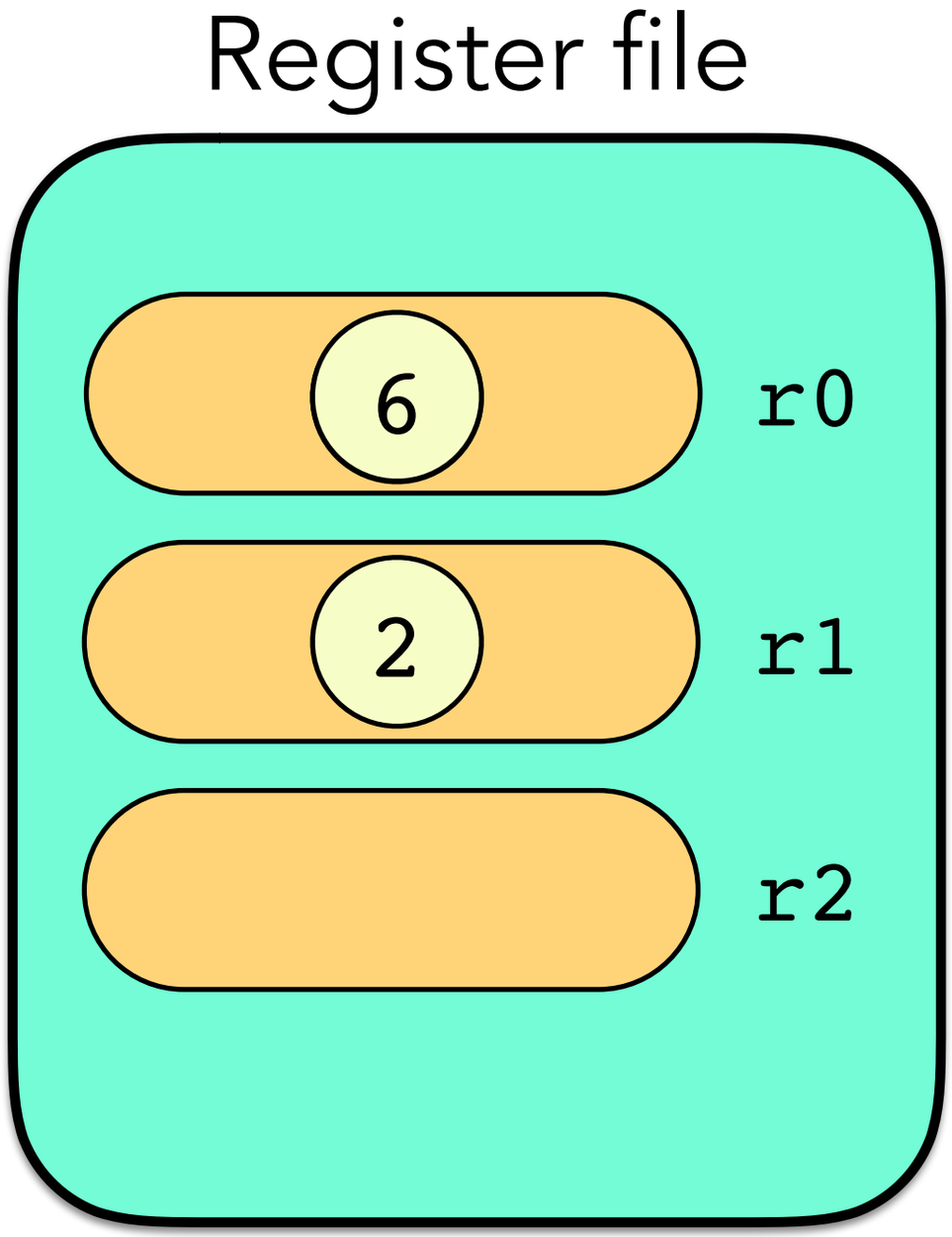
```
MOVE (r0, 4)
MOVE (r1, 2)
ADD (r0, r0, r1)
```



Register machines

If we want to do "`4+2`":

```
MOVE (r0, 4)
MOVE (r1, 2)
ADD (r0, r0, r1)
```



Register machine uses

- The majority of modern architectures use the register machine paradigm, due to its flexibility, high performance.

- Compilers have become very sophisticated to use registers optimally.

- Intel's x86, ARM and MIPS are all register machines.

Register sub-flavours

- Depending on the instruction set, one or multiple registers can be simultaneously accessed.

- Some machine only access one, plus memory: “register-memory” machines (e.g. x86)

- Many give more flexibility and use multiple registers: “register-register” machines. (e.g. ARM, MIPS)

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