High Performance Computing
Lecture 23

Matthew Jacob

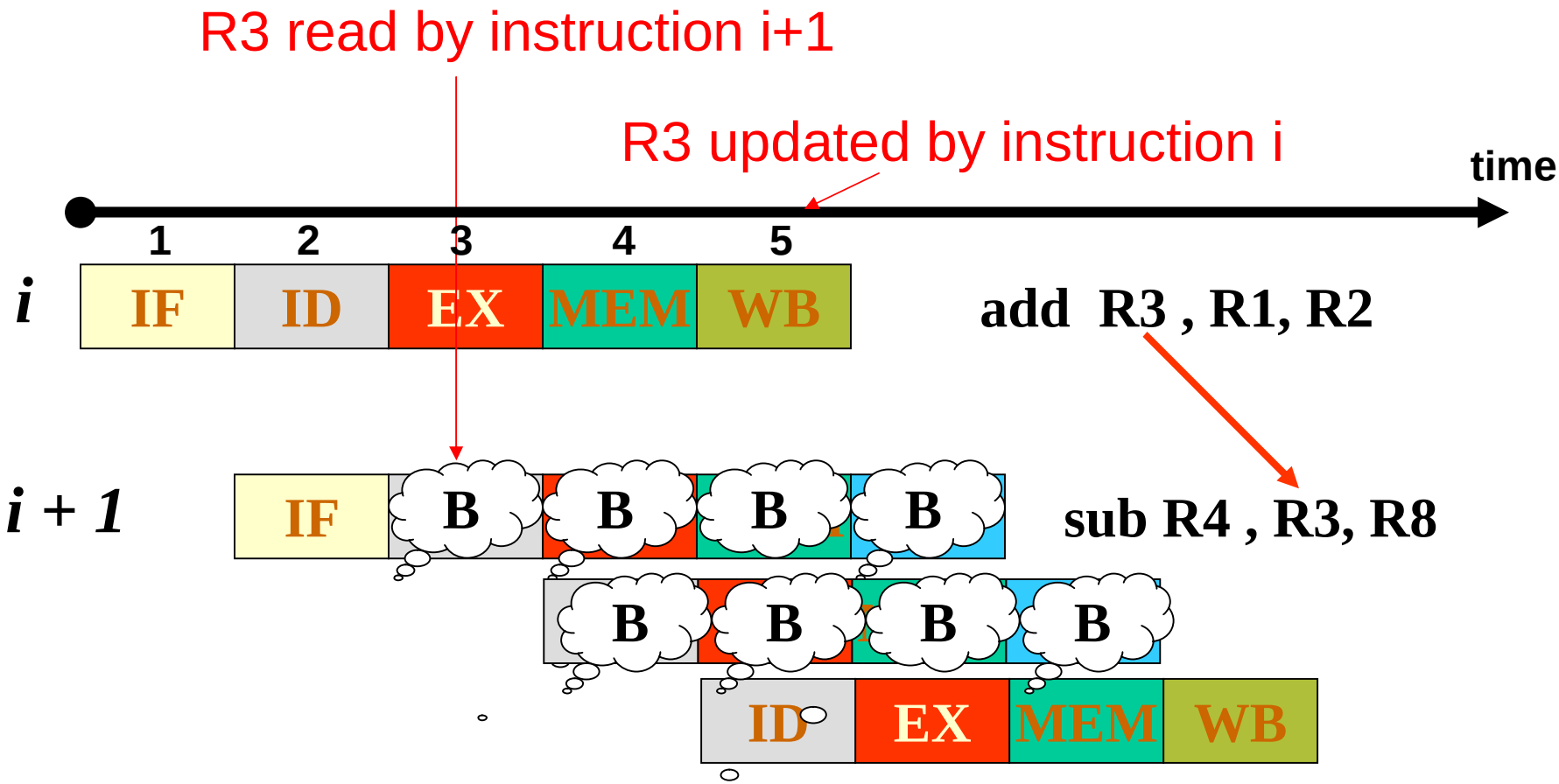
Indian Institute of Science

Problem: Pipeline Hazards

A situation where an instruction cannot proceed through the pipeline as it should

1. Structural hazard: When 2 or more instructions in the pipeline need to use the same resource at the same time
2. Data hazard: When an instruction depends on the data result of a prior instruction that is still in the pipeline
3. Control hazard: A hazard that arises due to control transfer instructions

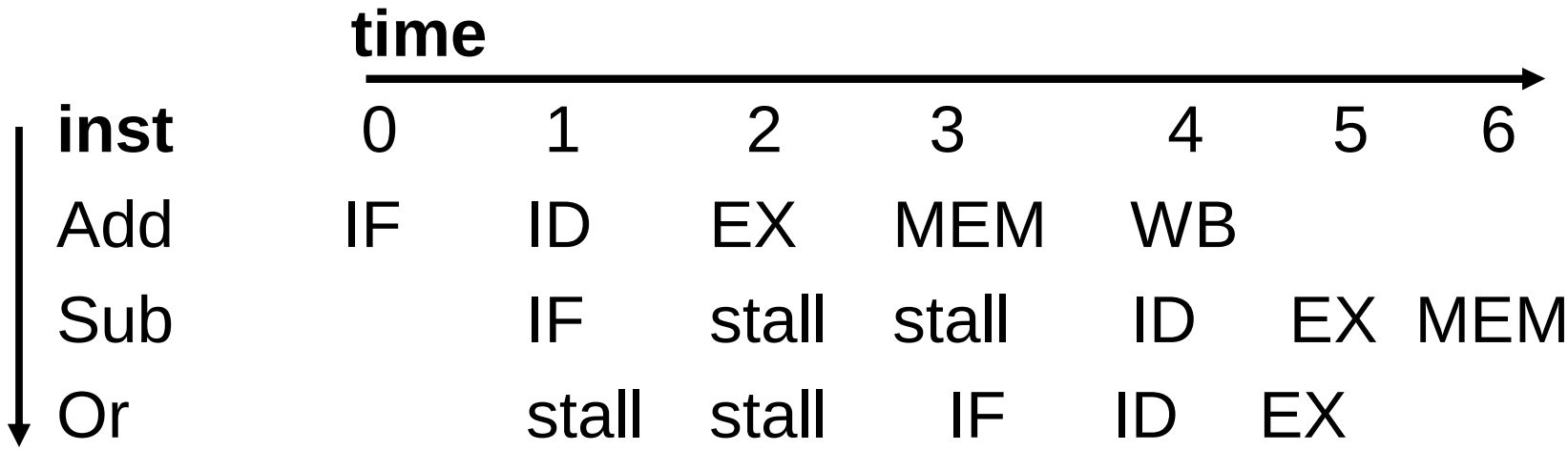
Data Hazard



Idea: Delay (or stall) the progress of instruction i+1 through the pipeline until the data is available in register R3

Solving Data Hazards

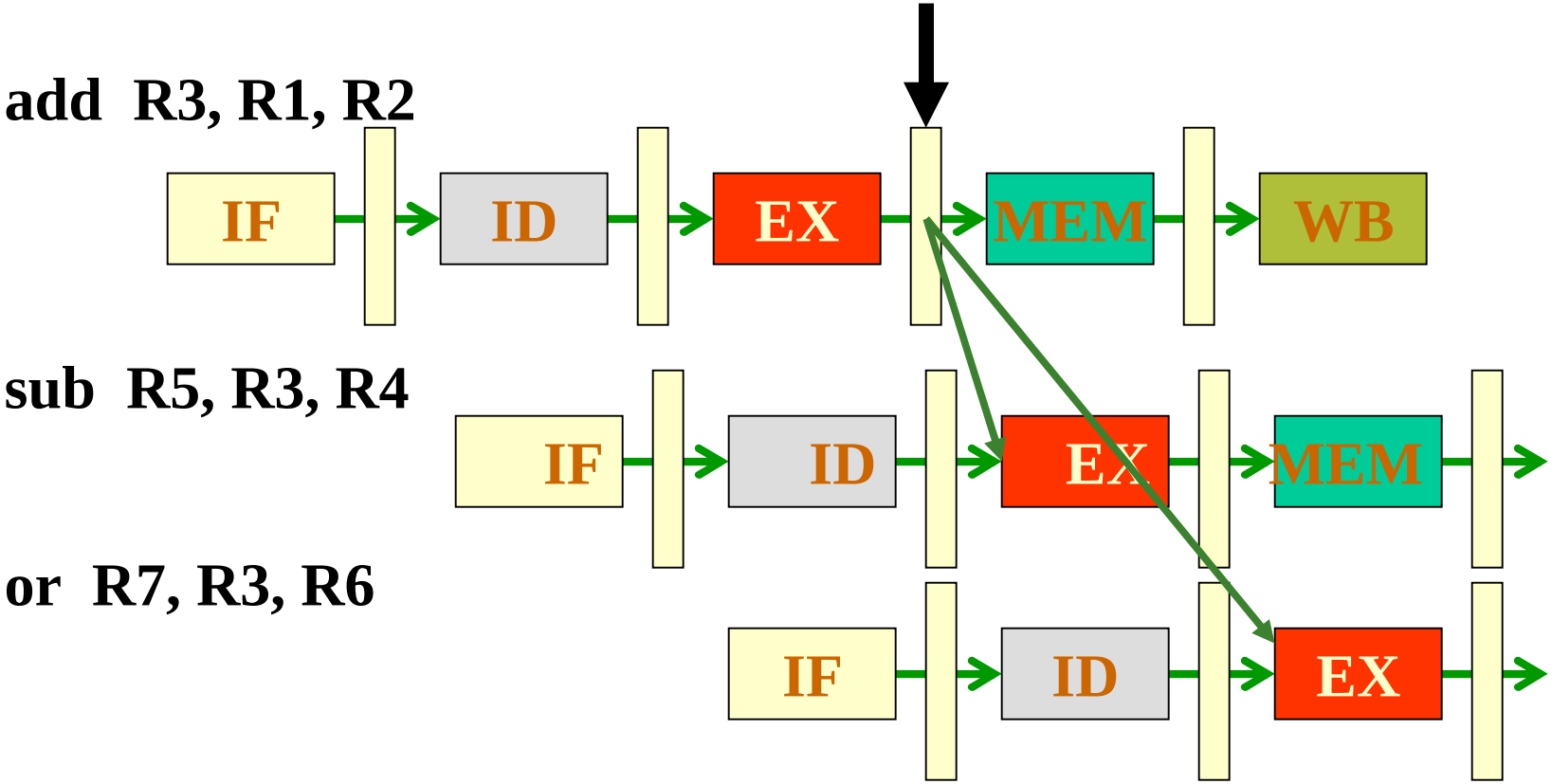
1. Interlock: Hardware that is included in the processor to detect such a data dependency and stall the dependent instruction



Solving Data Hazards

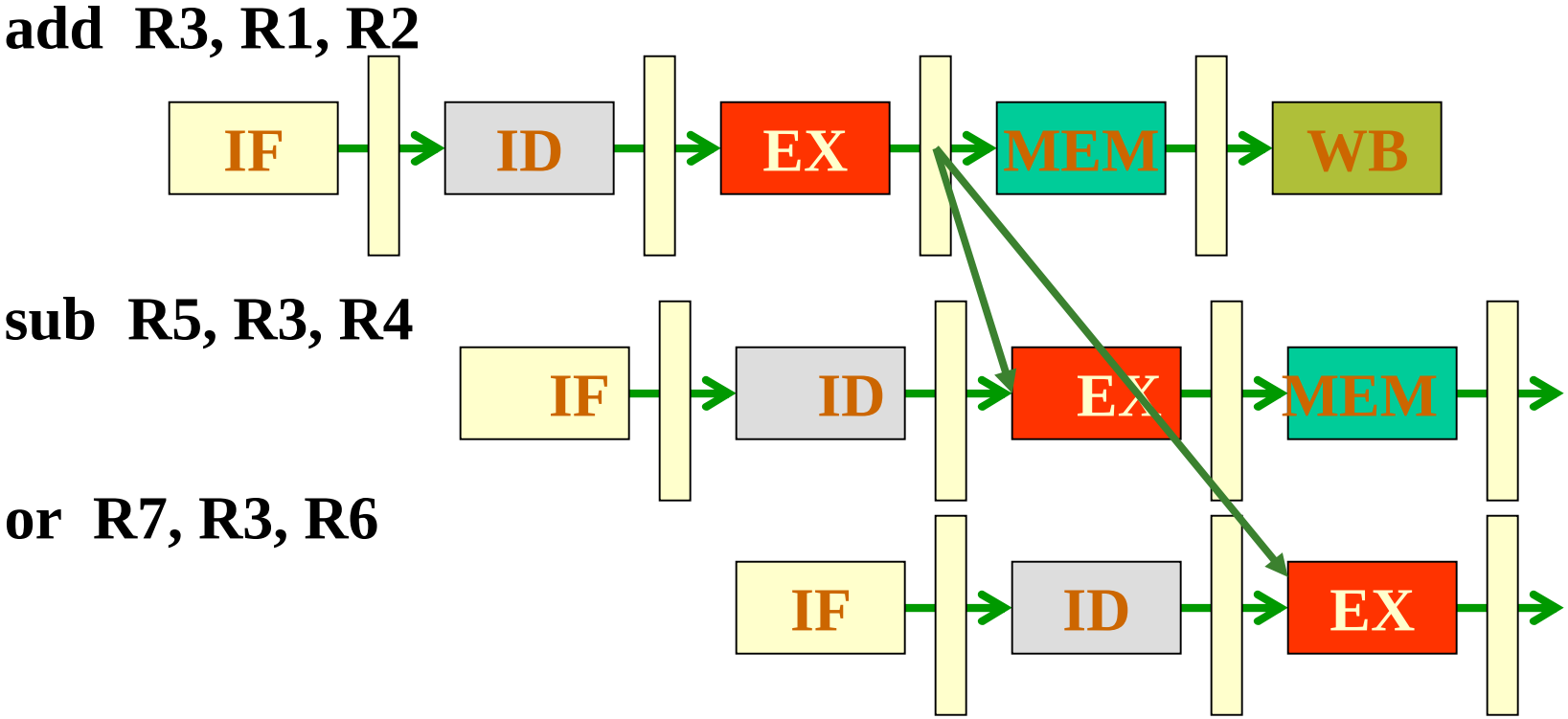
1. Interlocks & stalling dependent instructions

The result is available at the output of the ALU now (in the special purpose register ALUout)

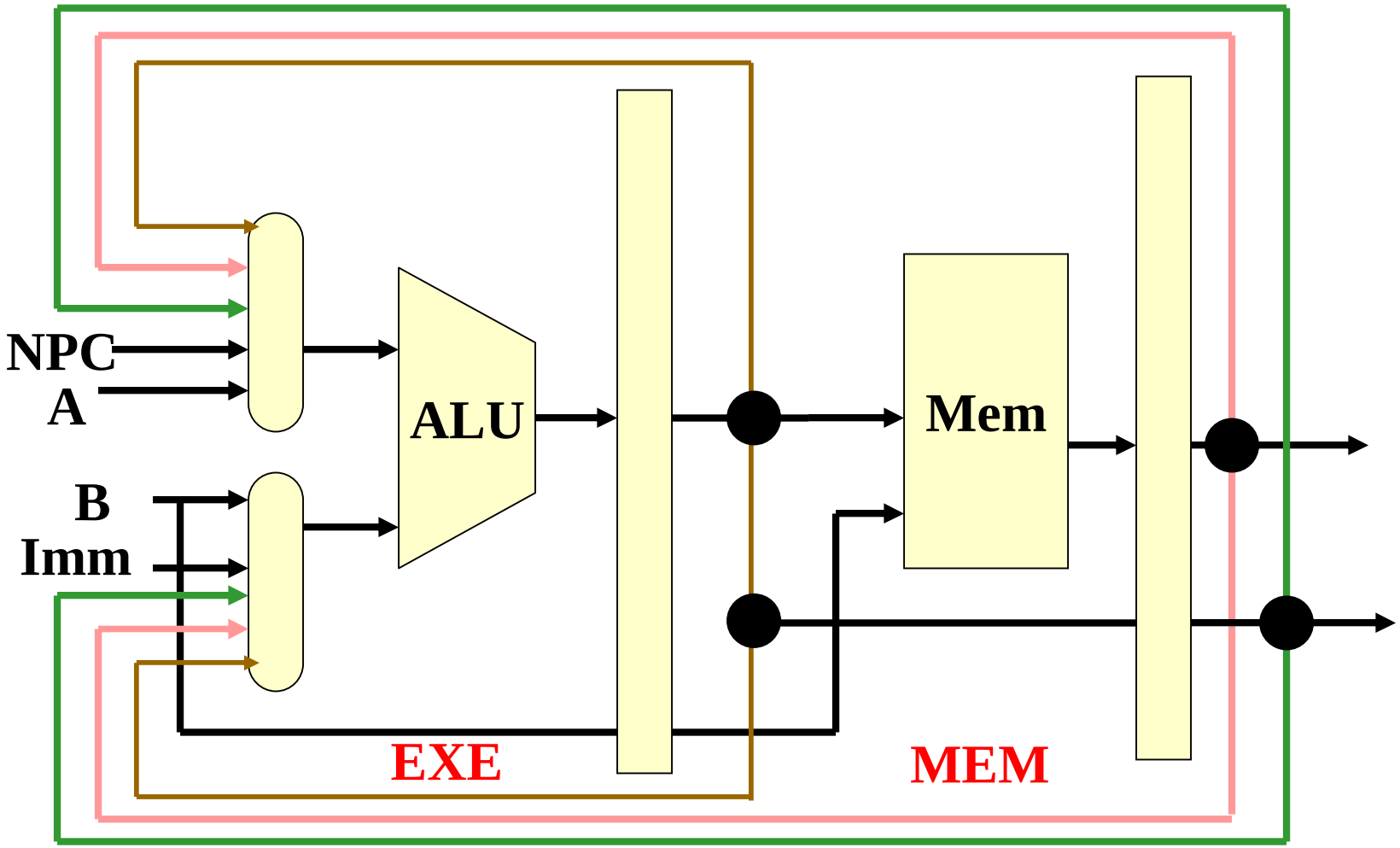


Solving Data Hazards

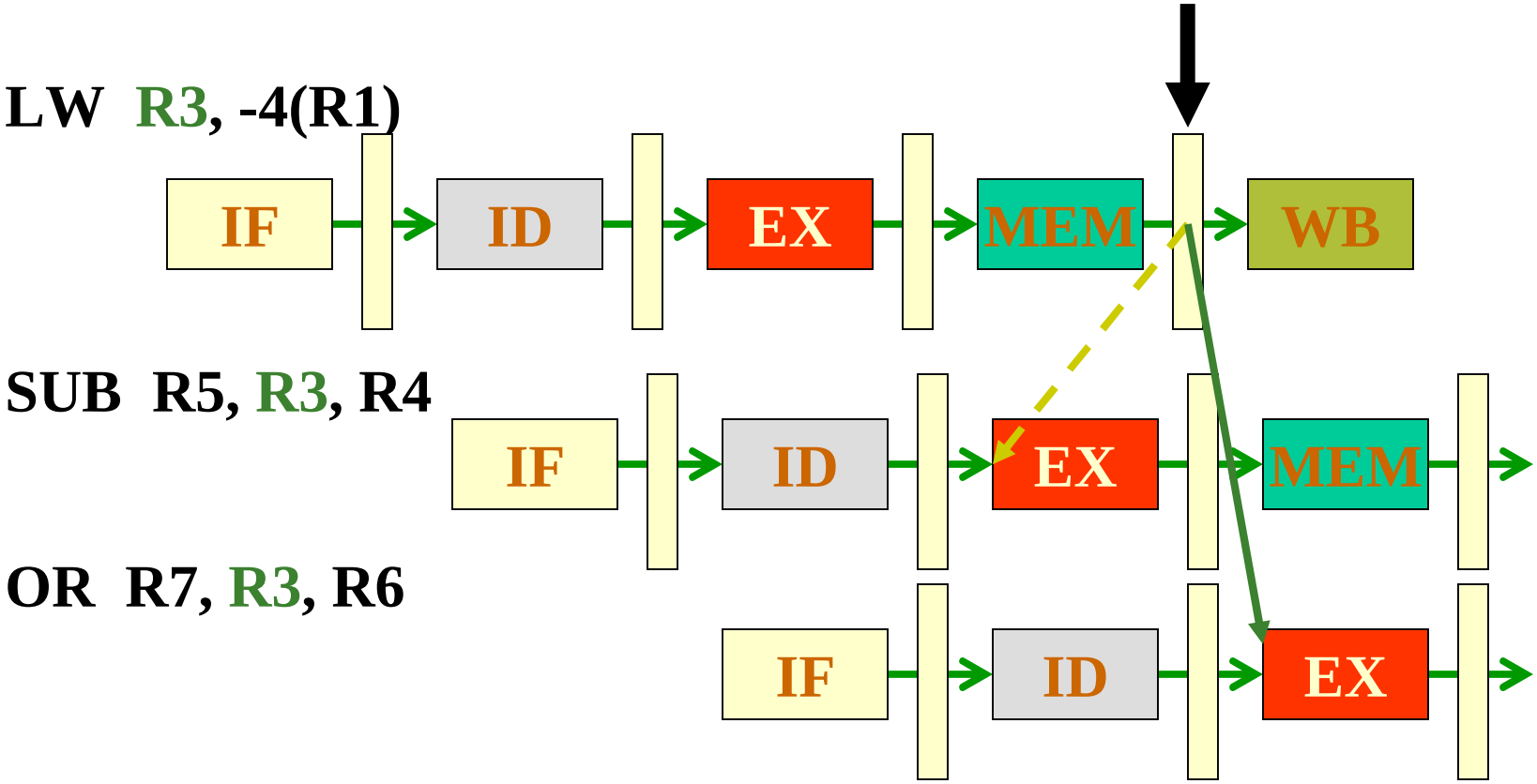
1. Interlocks & stalling dependent instructions
2. Forwarding or Bypassing: forward the result to EX as soon as is is available anywhere in the pipeline



Modified Processor Datapath



But Forwarding is Not Always Possible



Solving Data Hazards

1. Interlocks & stalling dependent instructions
2. Forwarding or Bypassing
3. Load delay slot
   - Build the hardware to assume that an instruction that uses a load value is separated from the load instruction

Recall: Notes from ISA Manual.

- For load instructions: the loaded value might not be available in the destination register for use by the instruction immediately following the load
  - LOAD DELAY SLOT
- For control transfer instructions: the transfer of control takes place only following the instruction immediately after the control transfer instruction
  - BRANCH DELAY SLOT

Solving Data Hazards

1. Interlocks & stalling dependent instructions
2. Forwarding or Bypassing
3. Load delay slot
4. Instruction Scheduling
   - Reorder the instructions of the program so that dependent instructions are far enough apart
   - This could be done either
     - by the compiler, before the program runs: Static Instruction Scheduling
     - by the hardware, when the program is running: Dynamic Instruction Scheduling

Static Instruction Scheduling

- Reorder the instructions of the program to eliminate data hazards …
  - or in general to reduce the execution time of the program
- Reordering must be safe

```
ADD   R1,  R2,  R3        /*  R1 = R2 + R3  */
SUB   R2,  R4,  R5        /*  R2 = R4 – R5  */
```