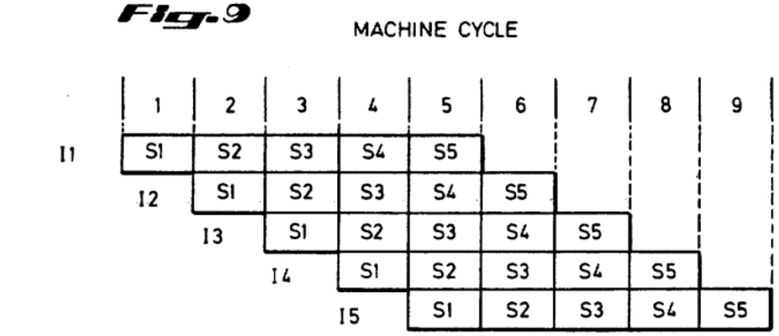
# Pipelining Hazards

Homework 1: now due October 3

Fig.9



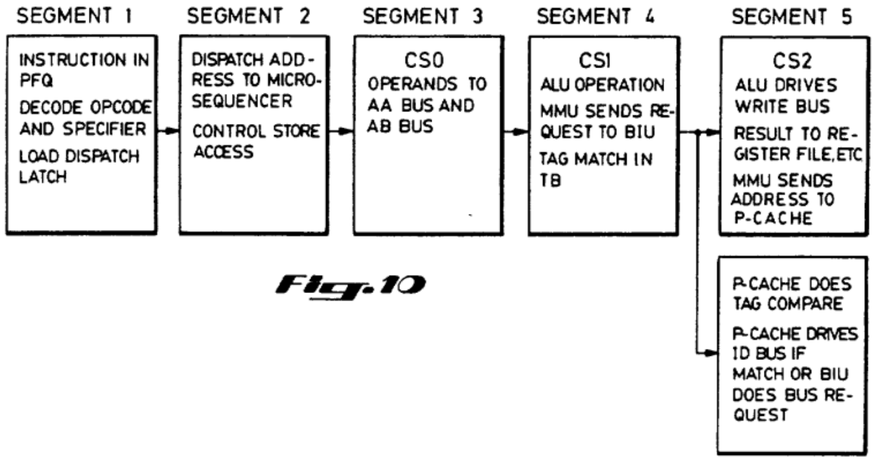




Fig.10

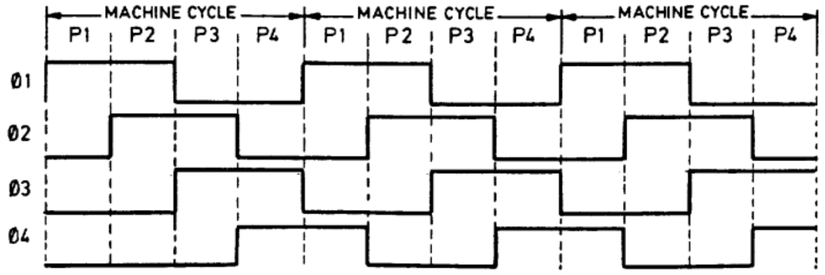



Fig.11

Image credit: https://patentimages.storage.googleapis.com/bb/2a/e8/0470966433594d/US5019967.pdf

# Outline

- Reviewing pipeline pitfalls
- Introducing pipeline hazards
- Hazard mitigation strategies

# Reviewing the Pitfalls of the Pipelined Processor

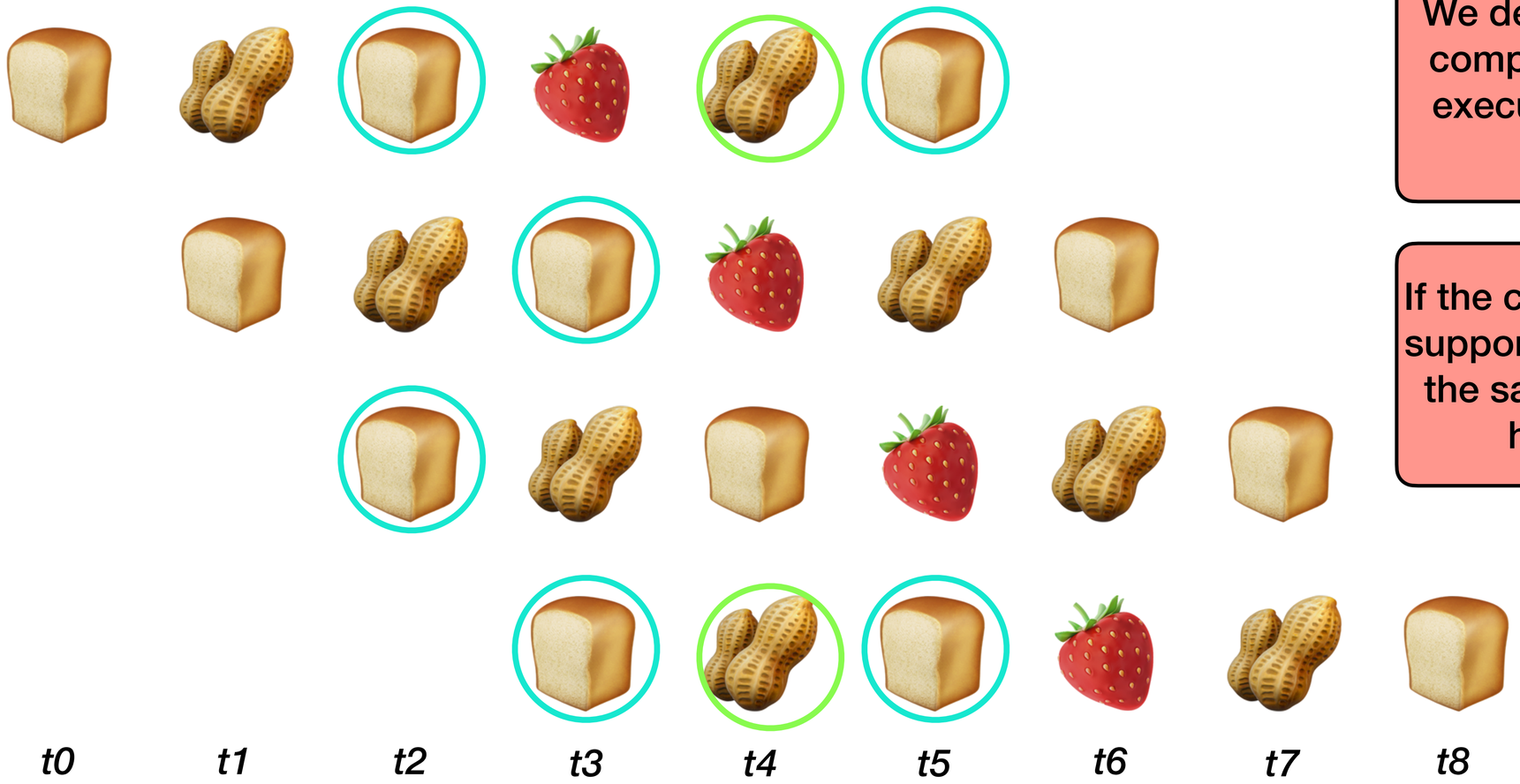
- Ideal: pipeline stages execute with equivalent latencies
- Ideal: indefinite number of instructions to execute to maximize time with a full pipeline
- Ideal: we can start a new instruction every cycle

- In practice: we can do our best, but different components take different latencies
- In practice: programs have a start and end
- In practice: instructions have dependencies which may lead to **hazards**

# Hazards in a Pipelined Processor

- Structural hazards: occurs when a hardware component cannot support the combination of instructions to execute within the same clock cycle
- Data hazards: occurs when there exists a **dependence** between two instructions in the pipeline, where the result from an incomplete instruction is relevant for the execution of another instruction
- Control hazards: occurs when a decision needs to be made about the next instruction to execute before that decision has been computed

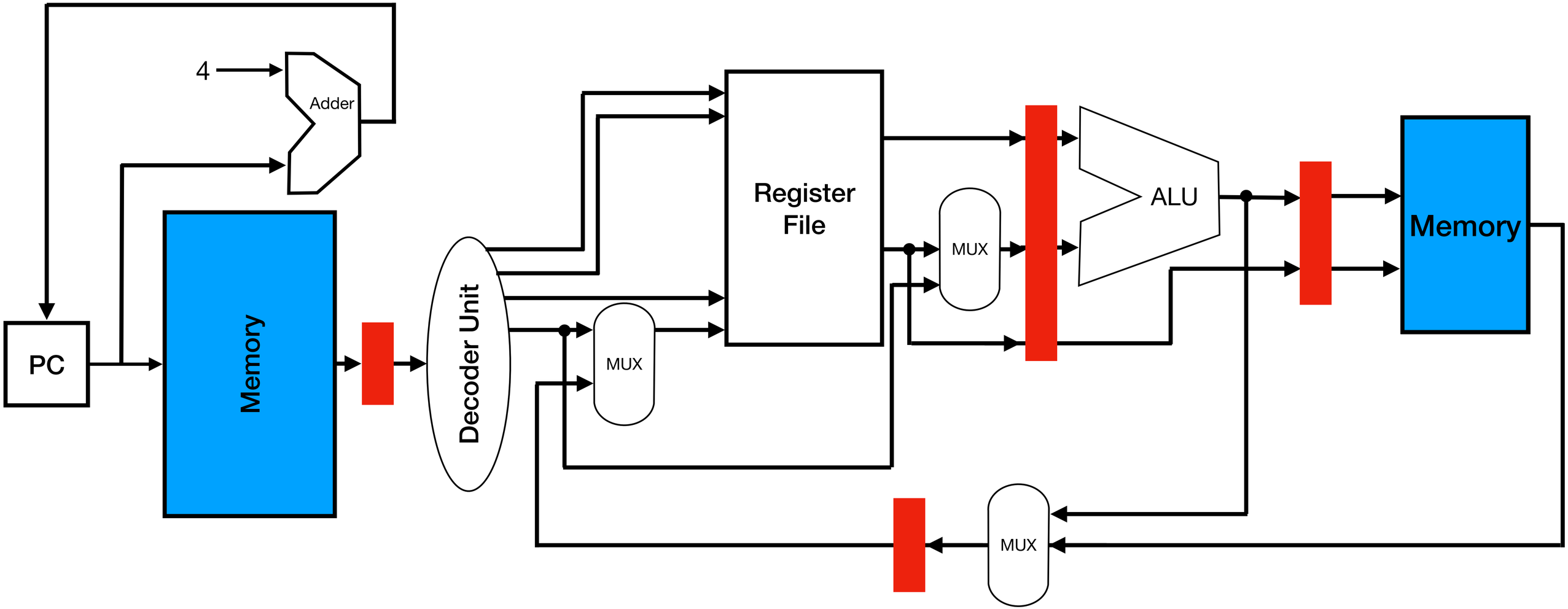
# Examples of a Structural Hazard



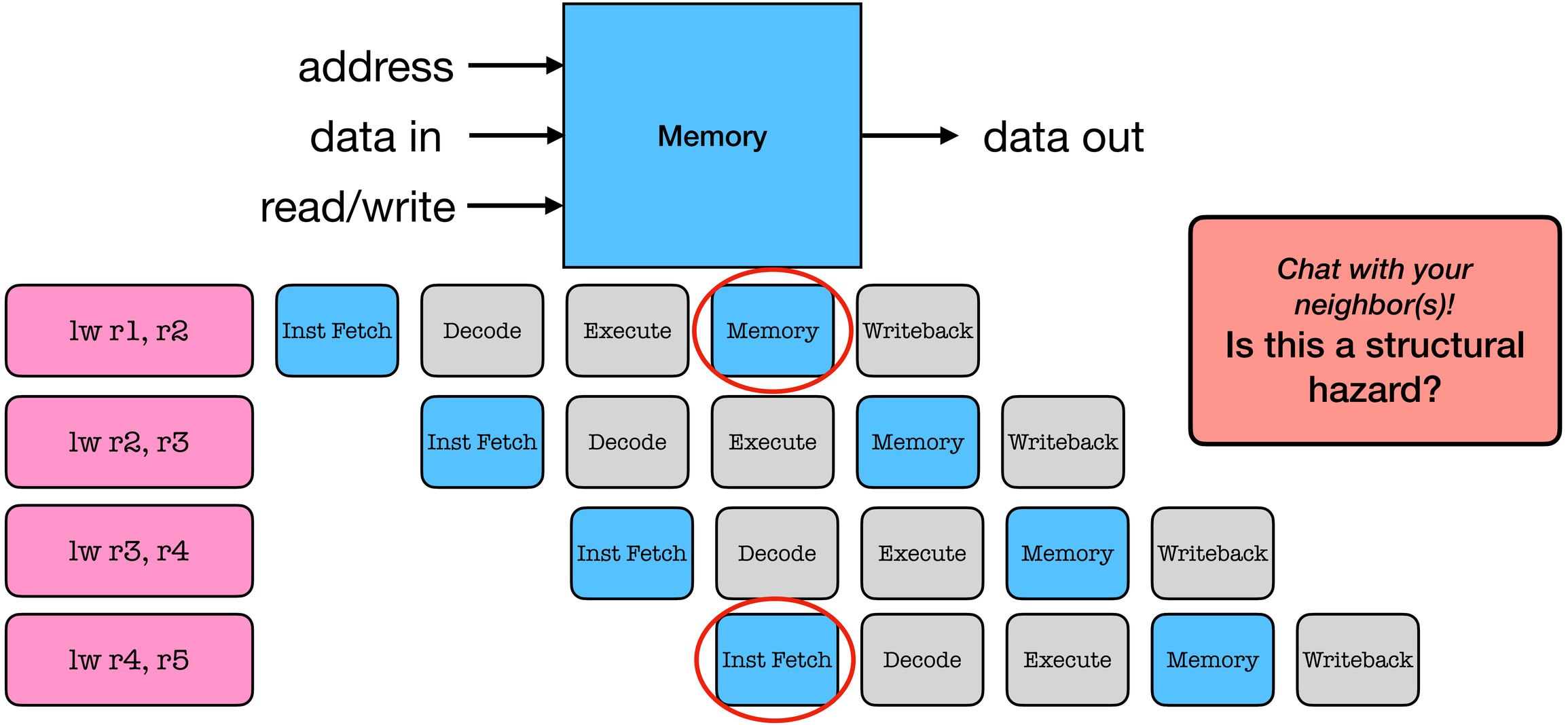

We depend on the same component for different executions at the same cycle!

If the component does not support multiple utilities in the same cycle, then we have a hazard!

# Example of a Structural Hazard



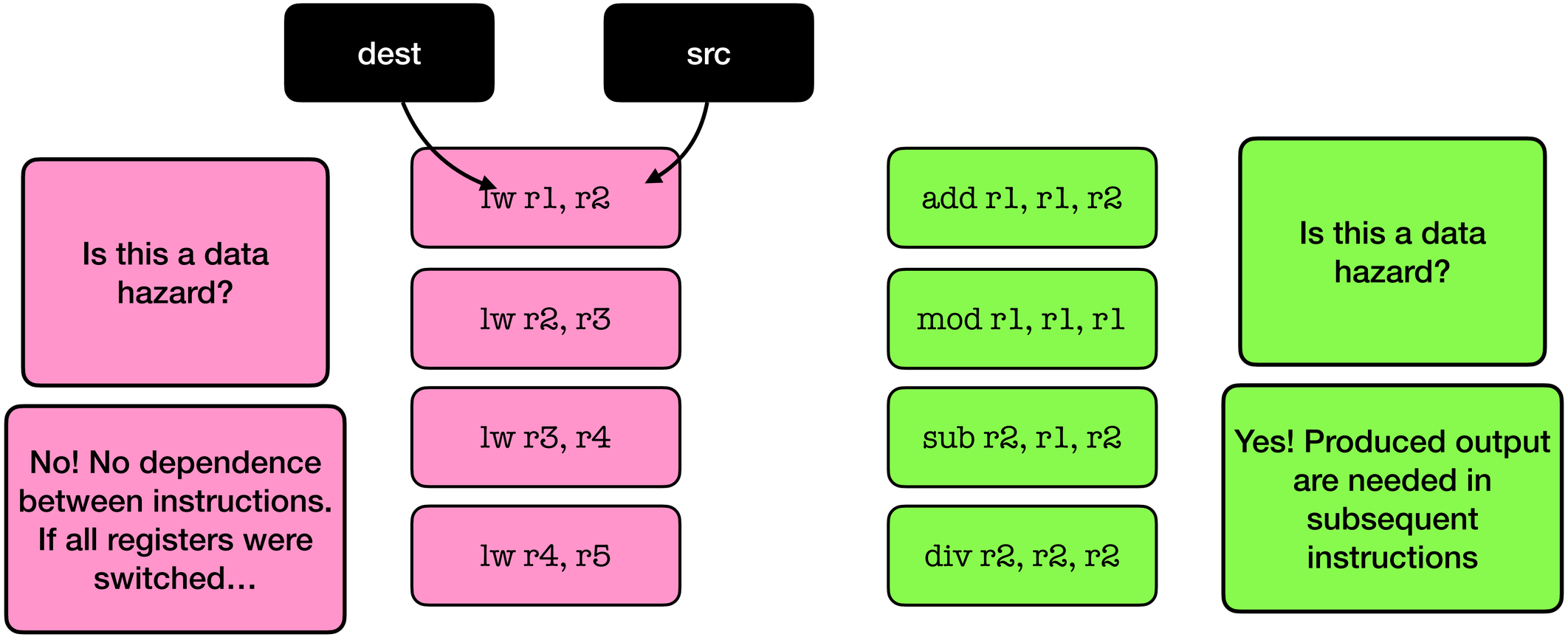
# Example of a Structural Hazard



# Data Hazards

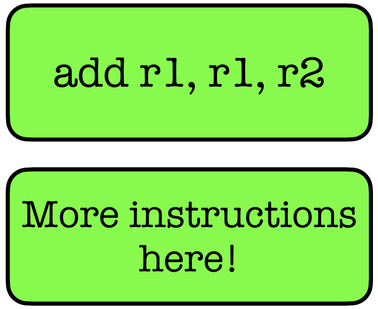
- Planned instructions cannot execute during a data hazard because data that is needed to execute the instruction is not yet available
- This interaction between instructions is sometimes called a *dependence* (plural, *dependences* or *dependencies* — depending on which sounds weirder to you)
- Data hazards are *resolved* once the necessary result is produced

# Data Hazard Example

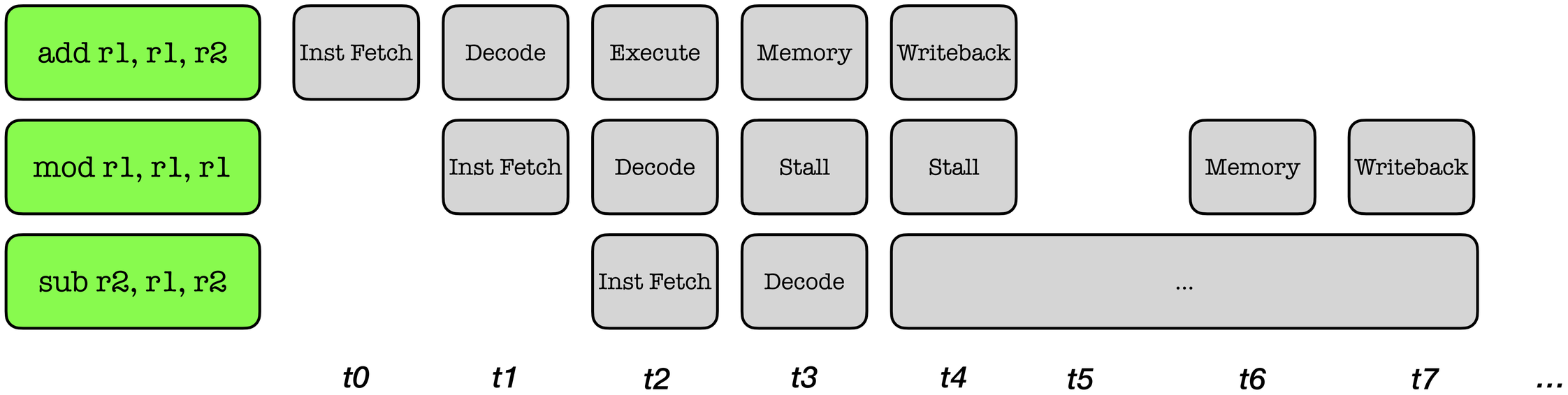


# Resolving Data Hazards

- Option 1: Use the compiler to re-organize instructions! This will allow many hazards to resolve themselves before execution
- Option 2: “Bubble” or stall the pipeline when a hazard is detected to wait for the
- Option 3: Forwarding data from one location in the pipeline to another stage that needs the data



# Resolving Data Hazards (Bubble)



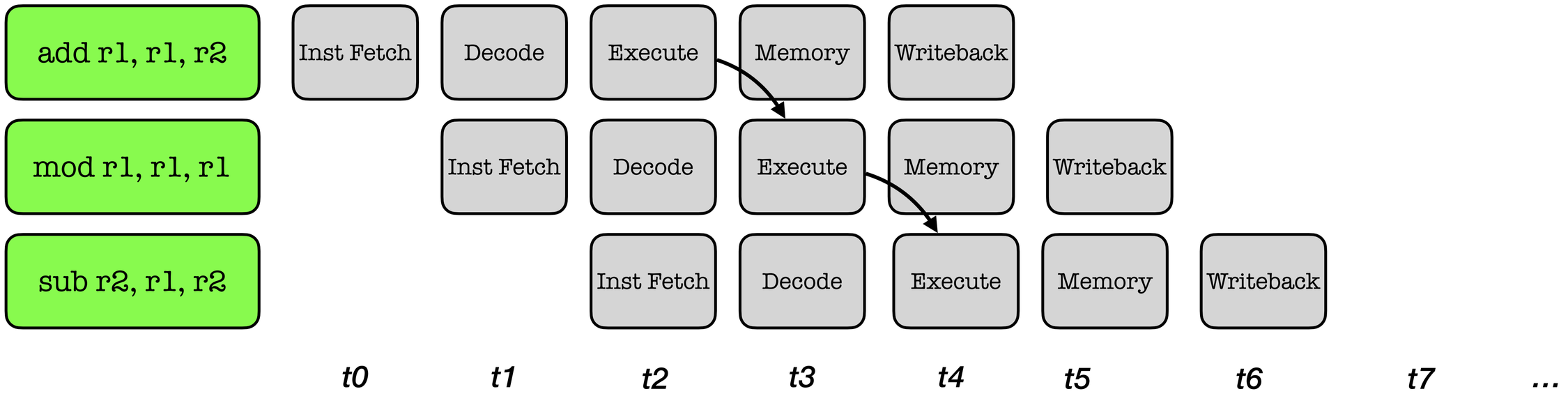

*Chat with your neighbor(s)!*

**What cycle will the “sub” hazard resolve? What cycle will the instruction complete?**

# Hazard Checking Unit

- To detect if there is a data hazard, the processor needs to deploy a *hazard checking unit*
- The hazard checking unit examines the state of the *pipeline registers* to look for inter-instruction dependencies!
- If such a dependence exists, then the processor needs to signal the appropriate components to “do nothing” until the hazard is resolved — only at this point should the signals be reset with the next steps of execution
- This means components may be highly under-utilized!

# Resolving Data Hazards (Forwarding)



# Takeaways

- Hazards are a natural byproduct of pipelining a task
- Processor hazards can be classified as structural, data or control
- Structural hazards must be resolved in the design of the data path
- Data hazards can be addressed in software, but hardware must account any potential hazard