



DEPARTMENT OF  
NETWORKED SYSTEMS  
AND SERVICES

---

# COMPUTER ARCHITECTURES

Information processing models

Budapest,  
02/28/2023

**Gábor Horváth**, [ghorvath@hit.bme.hu](mailto:ghorvath@hit.bme.hu)

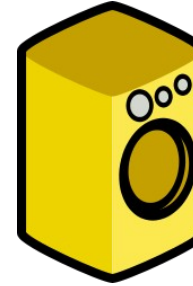


- Homepage: <http://www.hit.bme.hu/~ghorvath/comparch>
- Requirements:
  - Classroom practices:
    - Attendance: at least 4
  - Mid-term test
    - Scores must be >40%
    - Can be re-taken twice
    - Date: May 2 or 3, 18:00-20:00  
1<sup>st</sup> re-take: May 8 or 9, 18:00-20:00  
2<sup>nd</sup> re-take: during the re-take week, date is to be announced.
  - Exam

# WHY DO WE NEED THIS CLASS?

- We are going to understand
  - ...how a modern computer works
  - ...how a modern processor works
- „Professionals”
  - do not consider the computer to be a black box
  - understand how computers work
    - they can write **more efficient software**
  - know the compromises

## WHAT IS A COMPUTER?

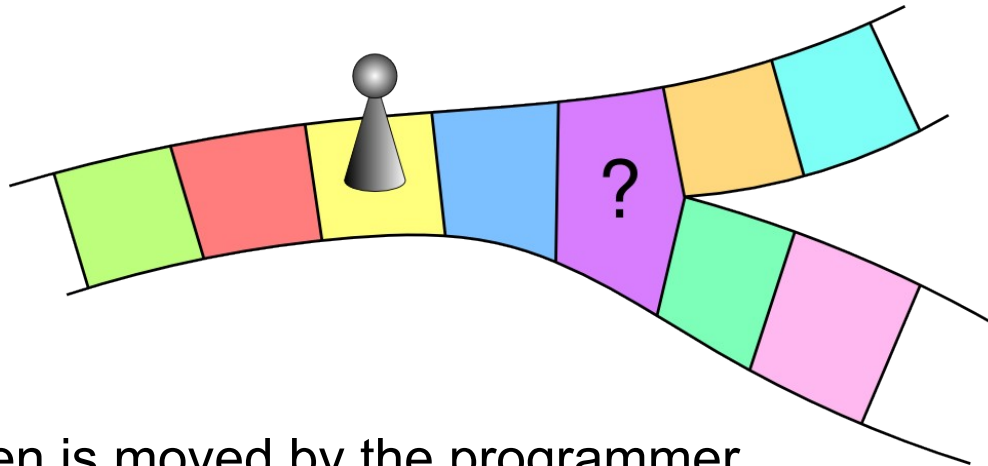




# ***Information Processing Models***

- How to describe problems for a computer?
  - Control flow
  - Data flow
  - Demand driven

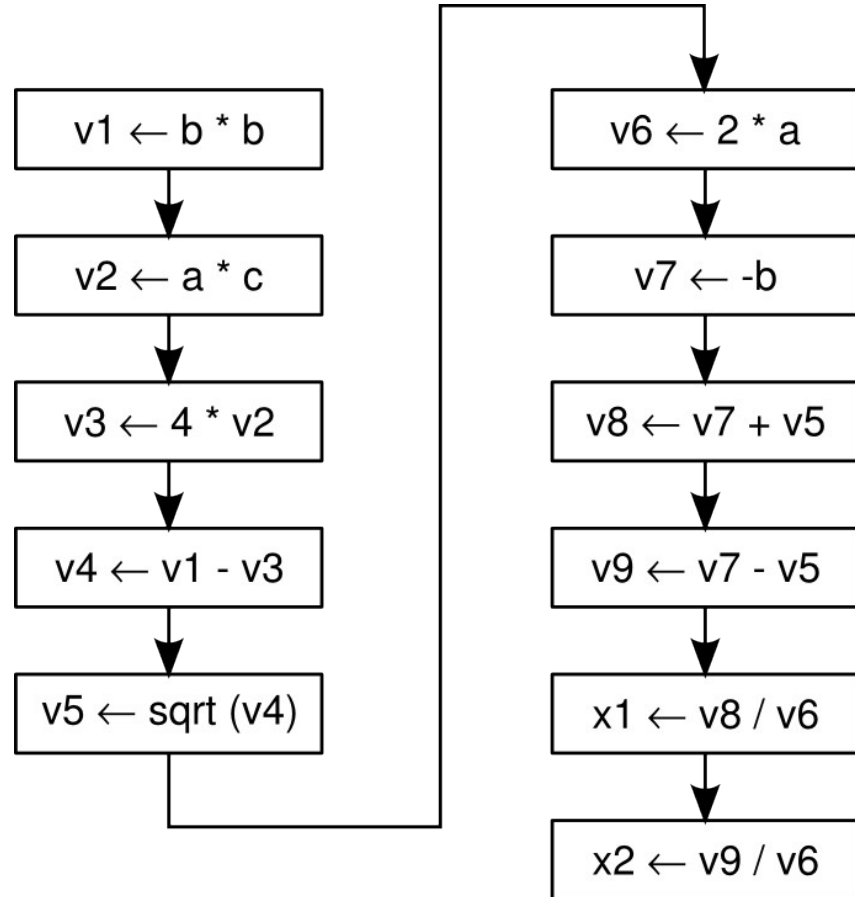
- Order of instructions: given by the control token



- Control token is moved by the programmer
  - Implicitly (to next instruction)
  - Explicitly (branch, goto, function call, return, etc.)
- Operands and results are stored in a shared memory
- Description of the program: *flow-chart*

- Example: solutions of quadratic equations

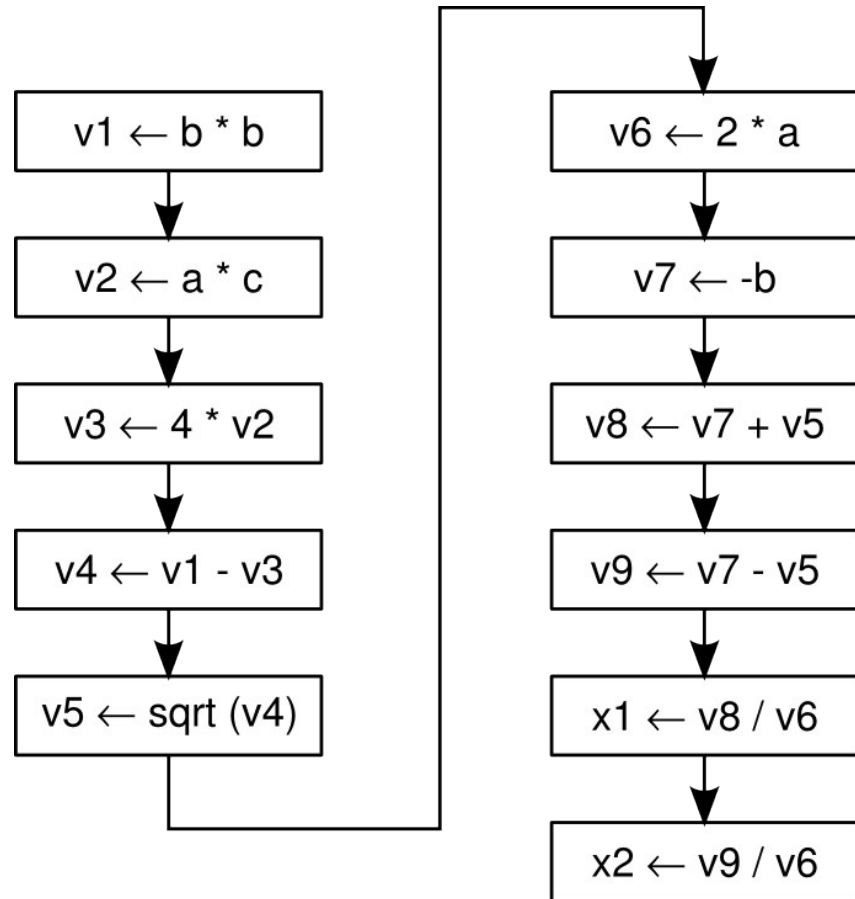
$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$





- Parallelism? Where?

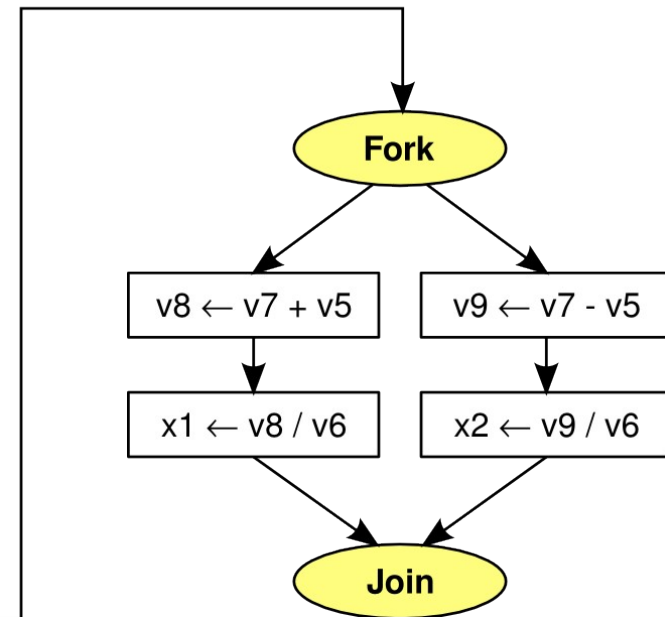
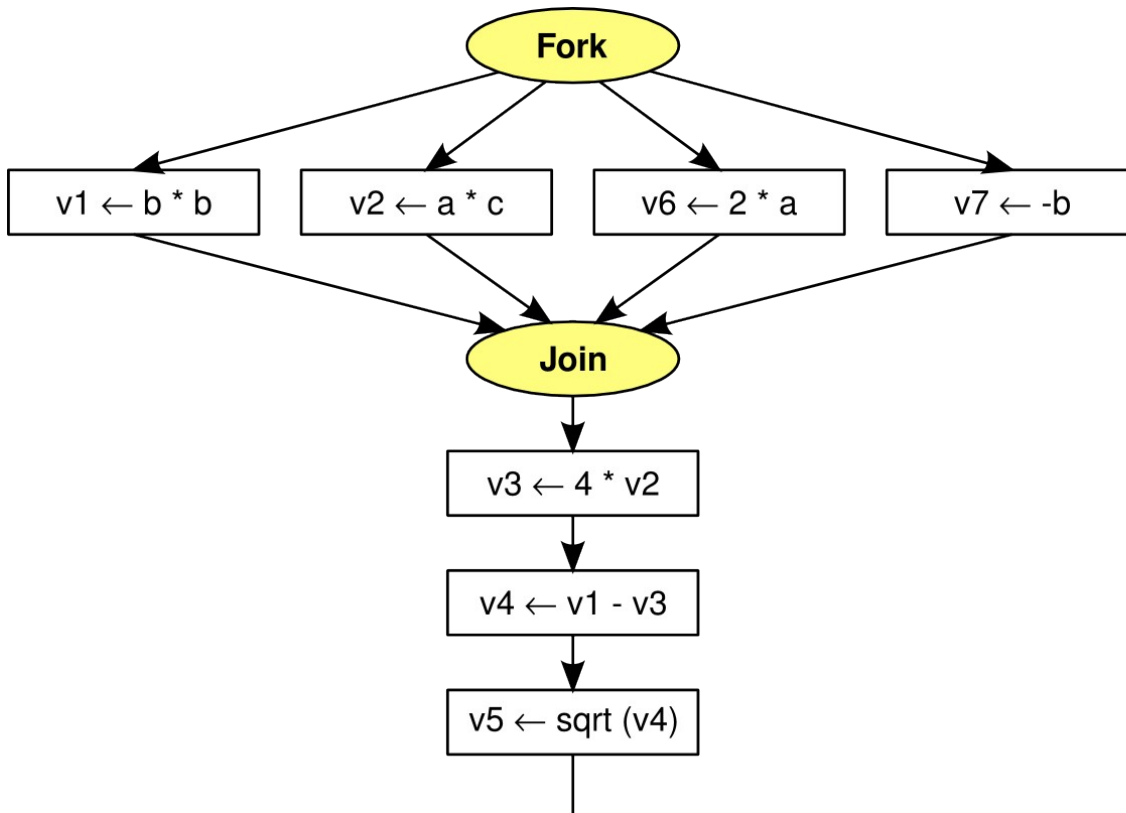
- $v1, v2, v6, v7$
- $v8, v9$
- $x1, x2$



## CONTROL-FLOW BASED MODEL

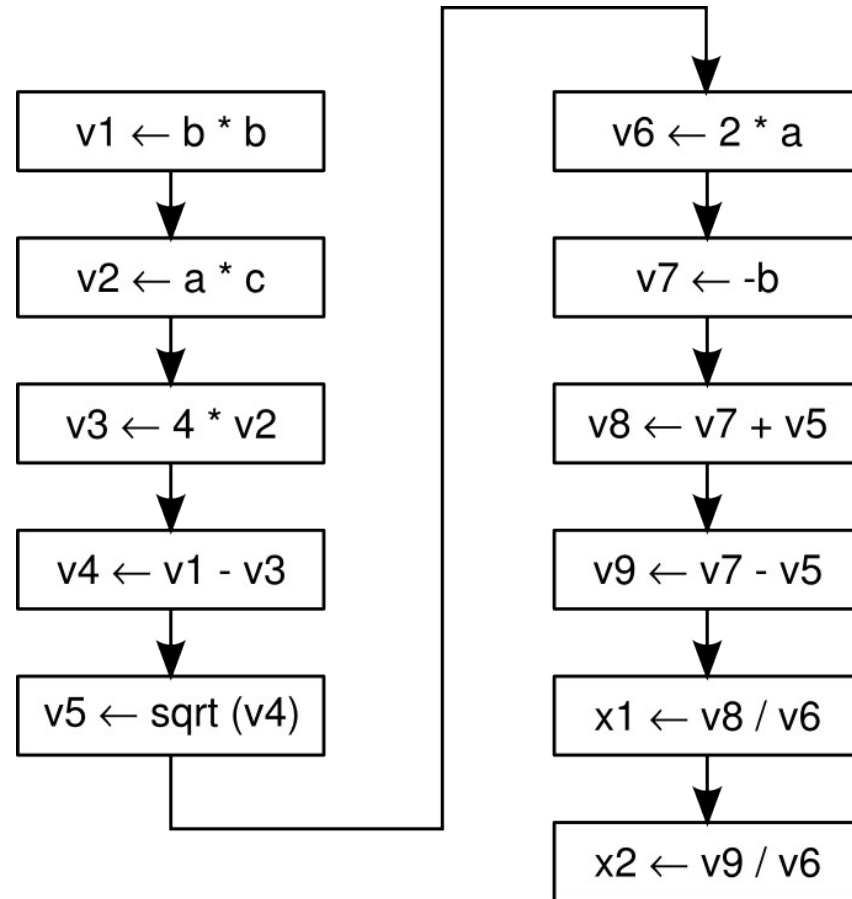
- Parallelism can be described by
- Fork/Join primitives only
- **Fork**: creates multiple control tokens
- **Join**: joins control tokens

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

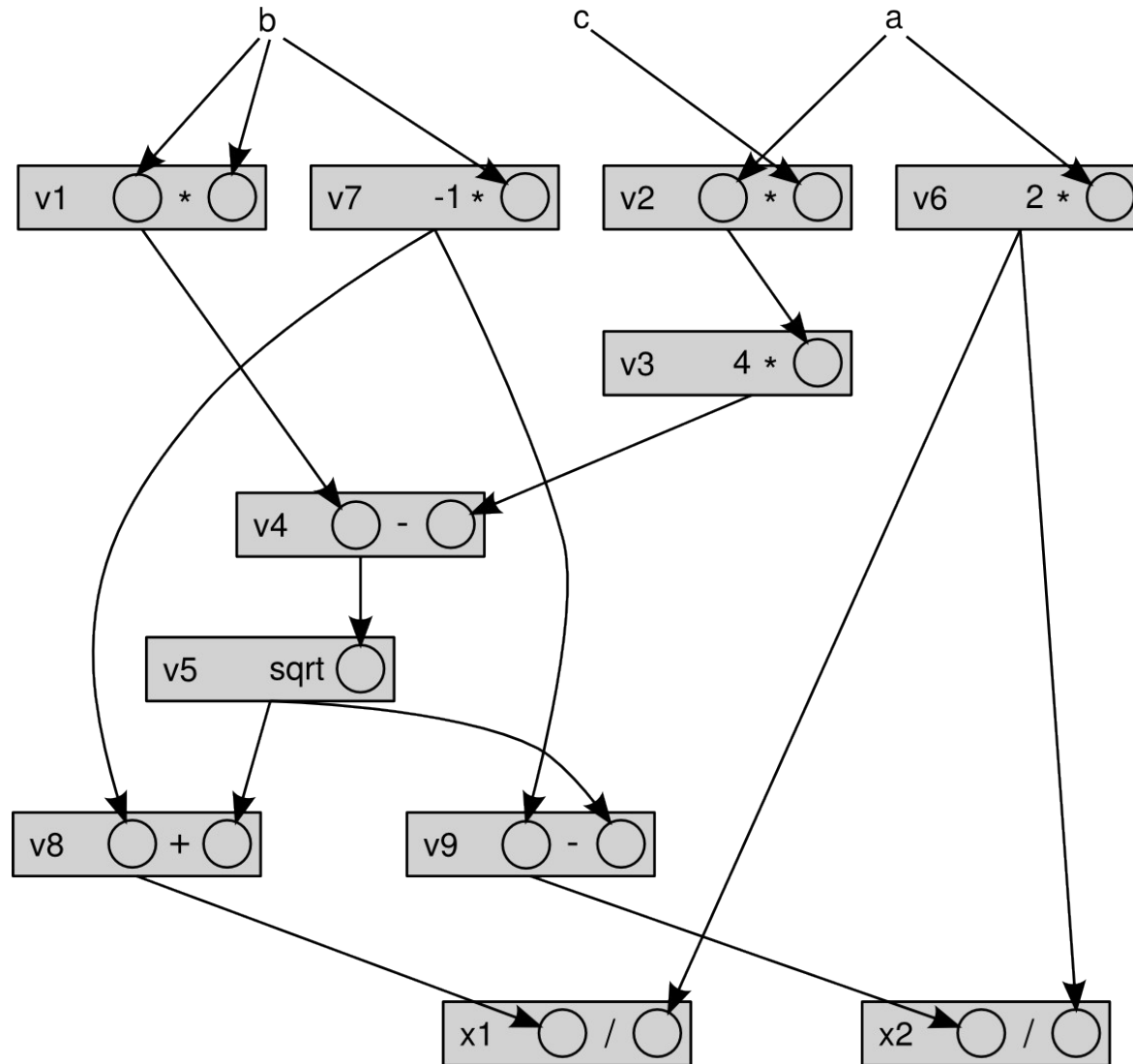


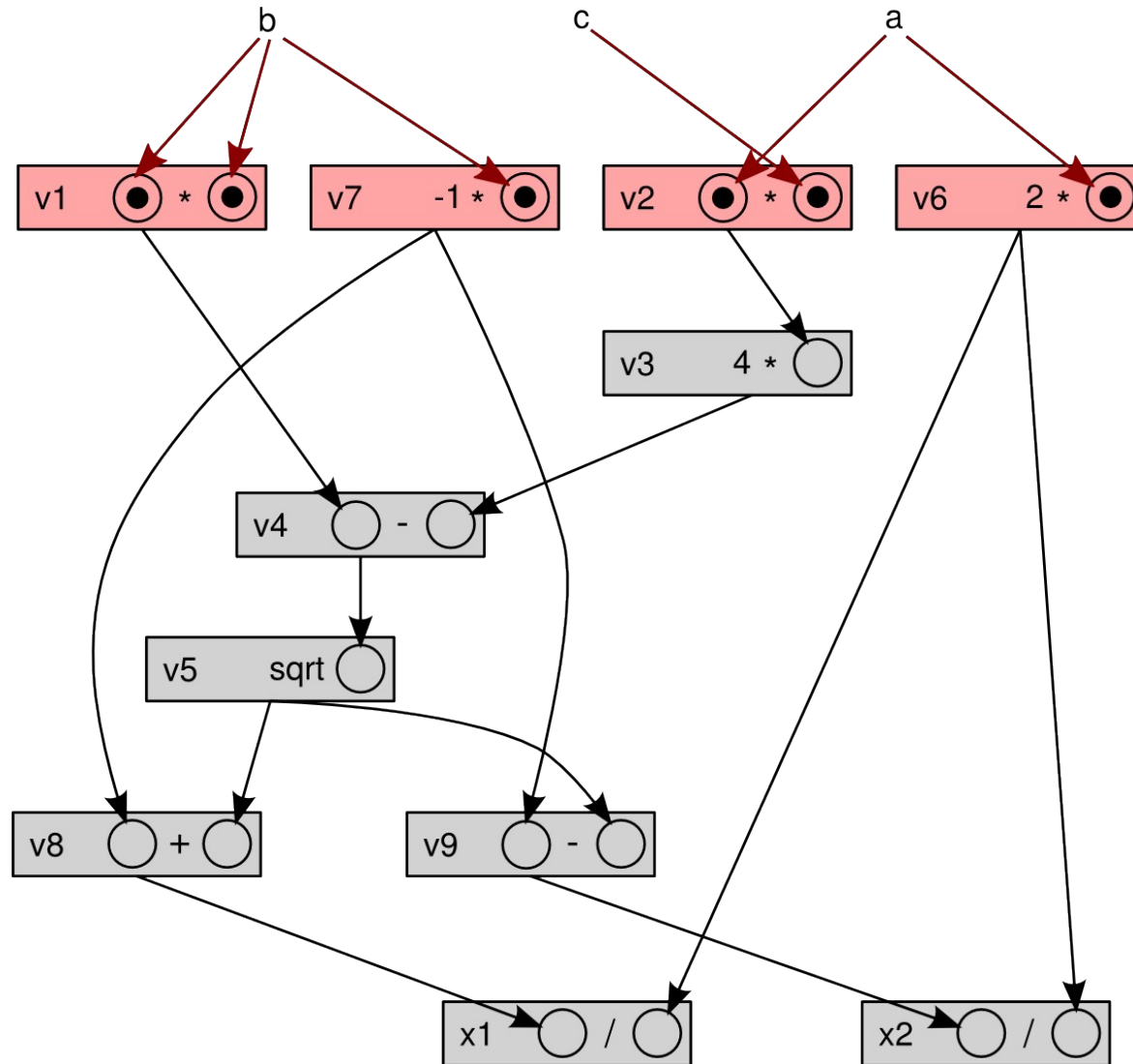
- Features
  - It can not detect parallelism
  - It can not exploit parallelism
  - Very wide-spread
  - Example: Neumann architecture
    - Control token: program counter
    - Shared memory: registers + system memory

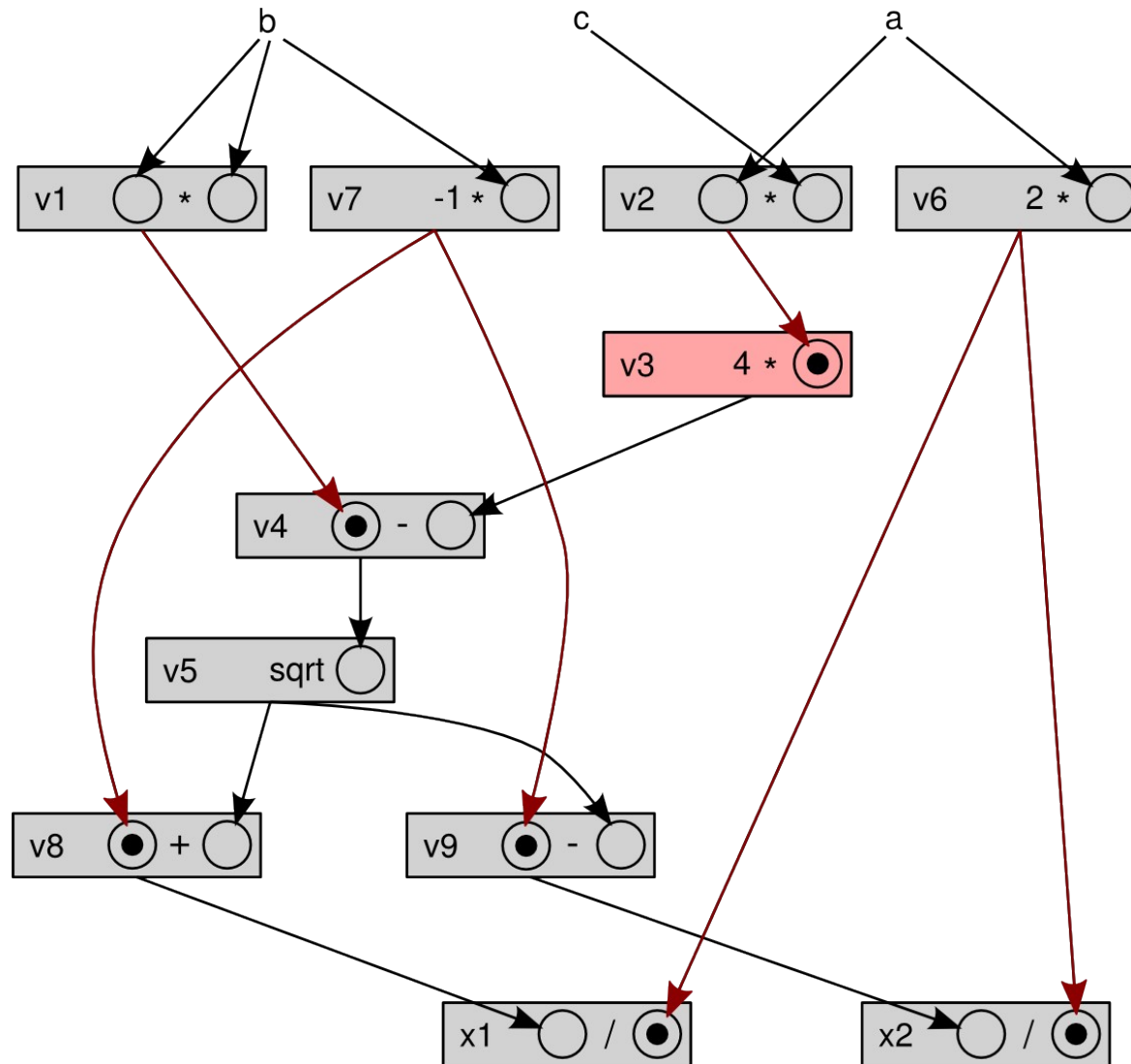
- How do we know that these can be executed in parallel?
  - v1, v2, v6, v7
  - v8, v9
  - x1, x2
- Dependence analysis!



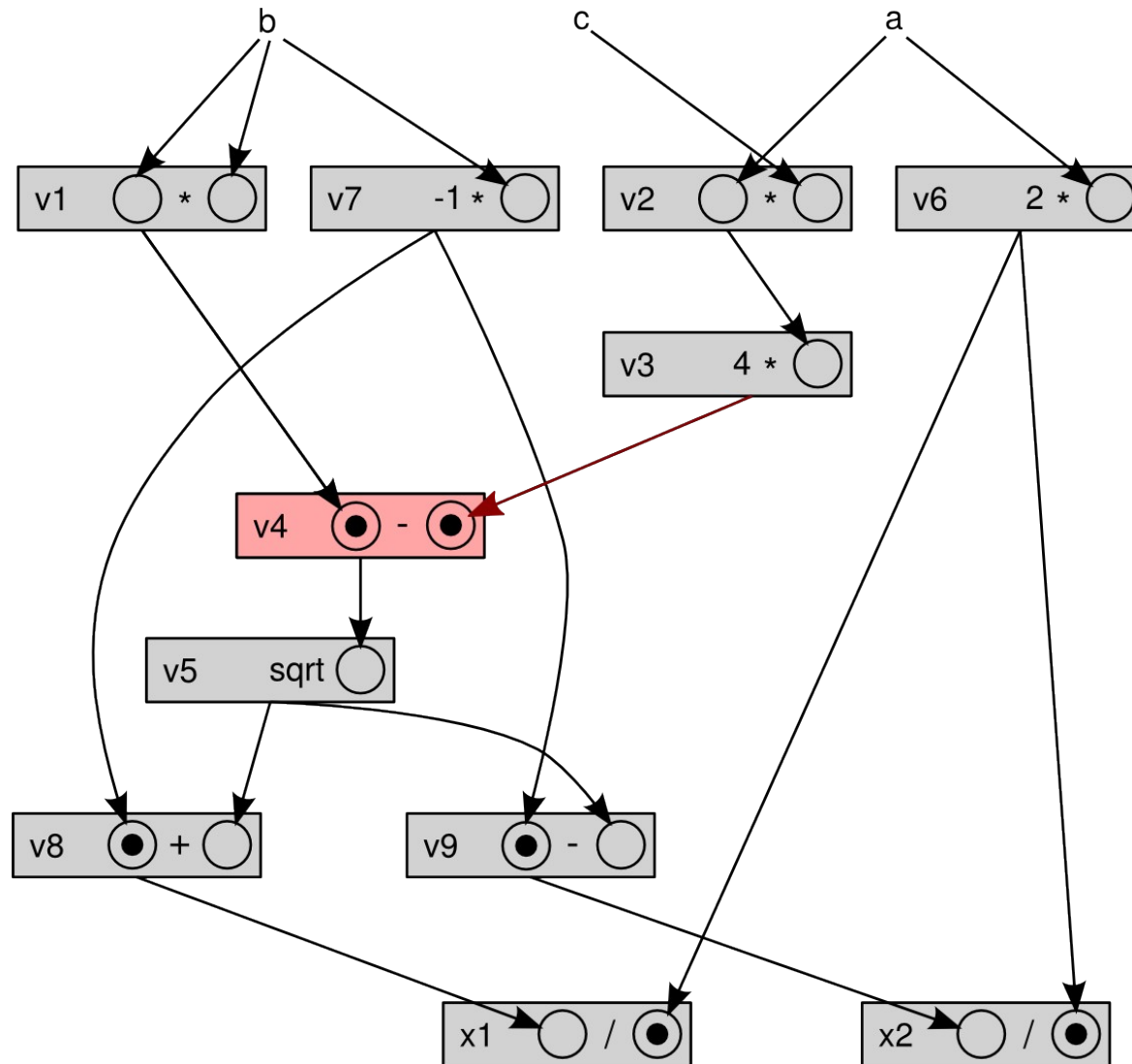
- No control token
- No shared memory
- Instead of control token:
  - Execution of the instructions is automatically given:
    - **Every instruction is executed as soon as all its operands are available**
- Instead of shared memory:
  - Sub-results and operands of the instructions are passed directly among them
- Description of the program: *precedence graph*
- Example: solutions of quadratic equations

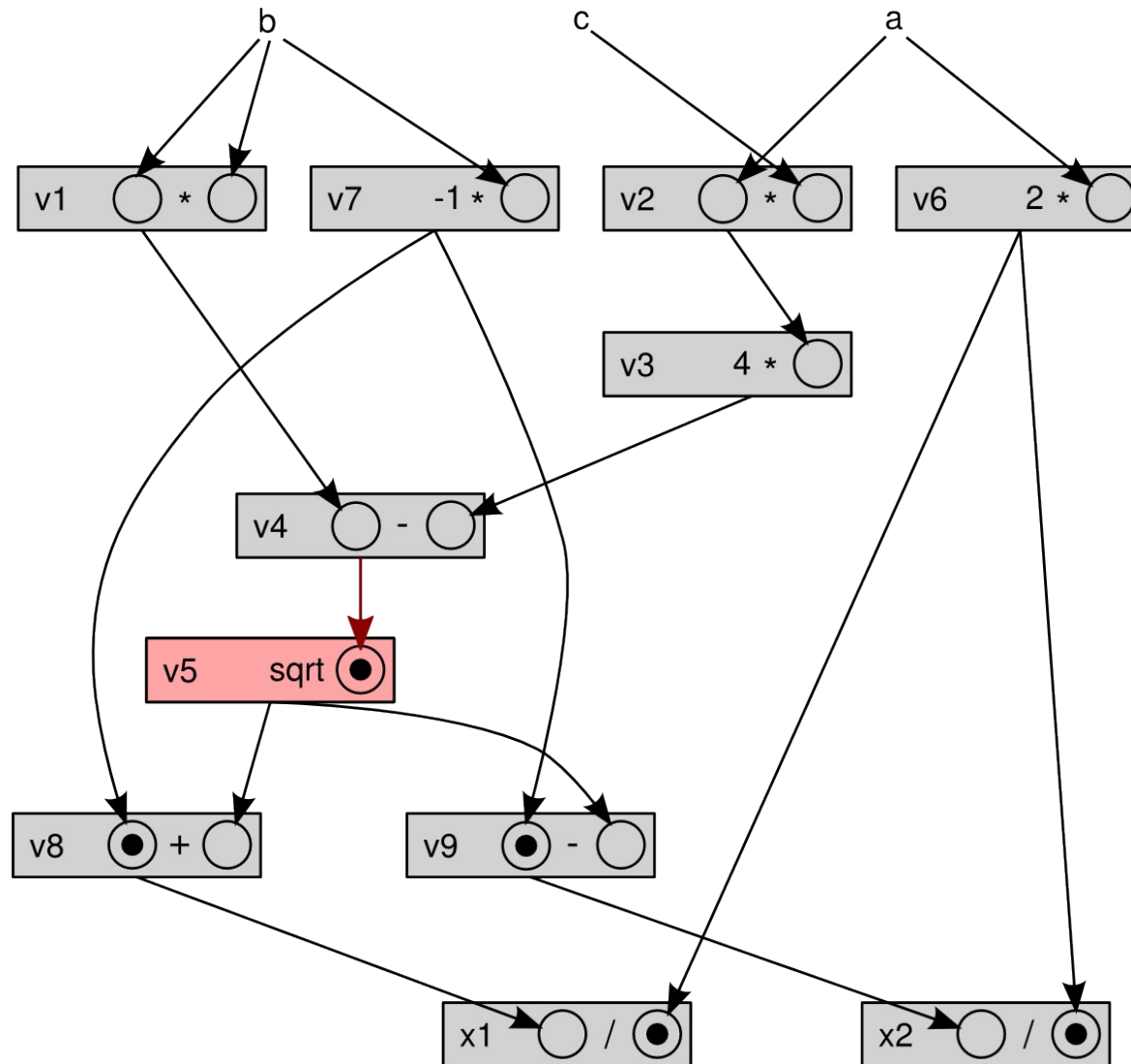


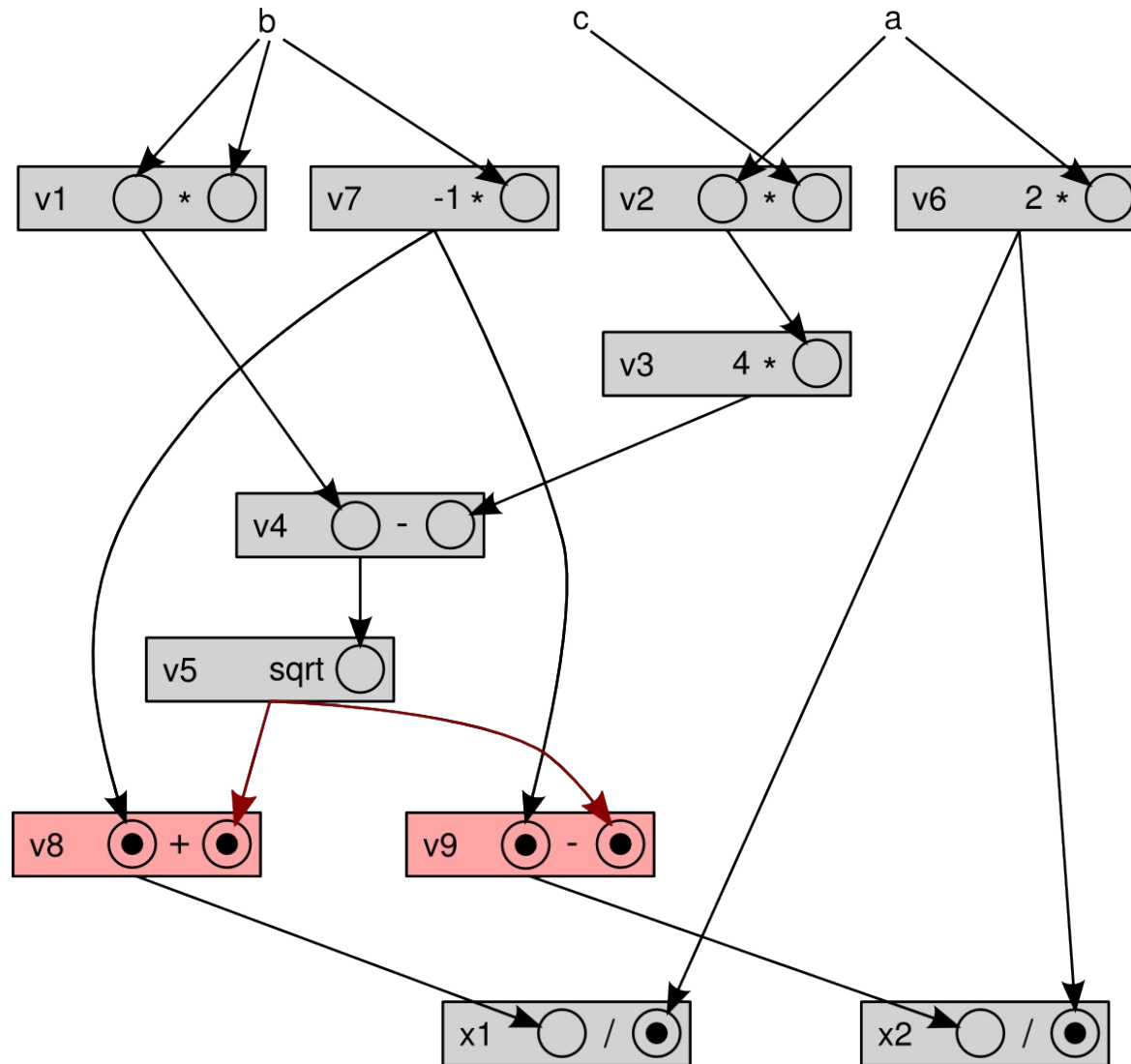


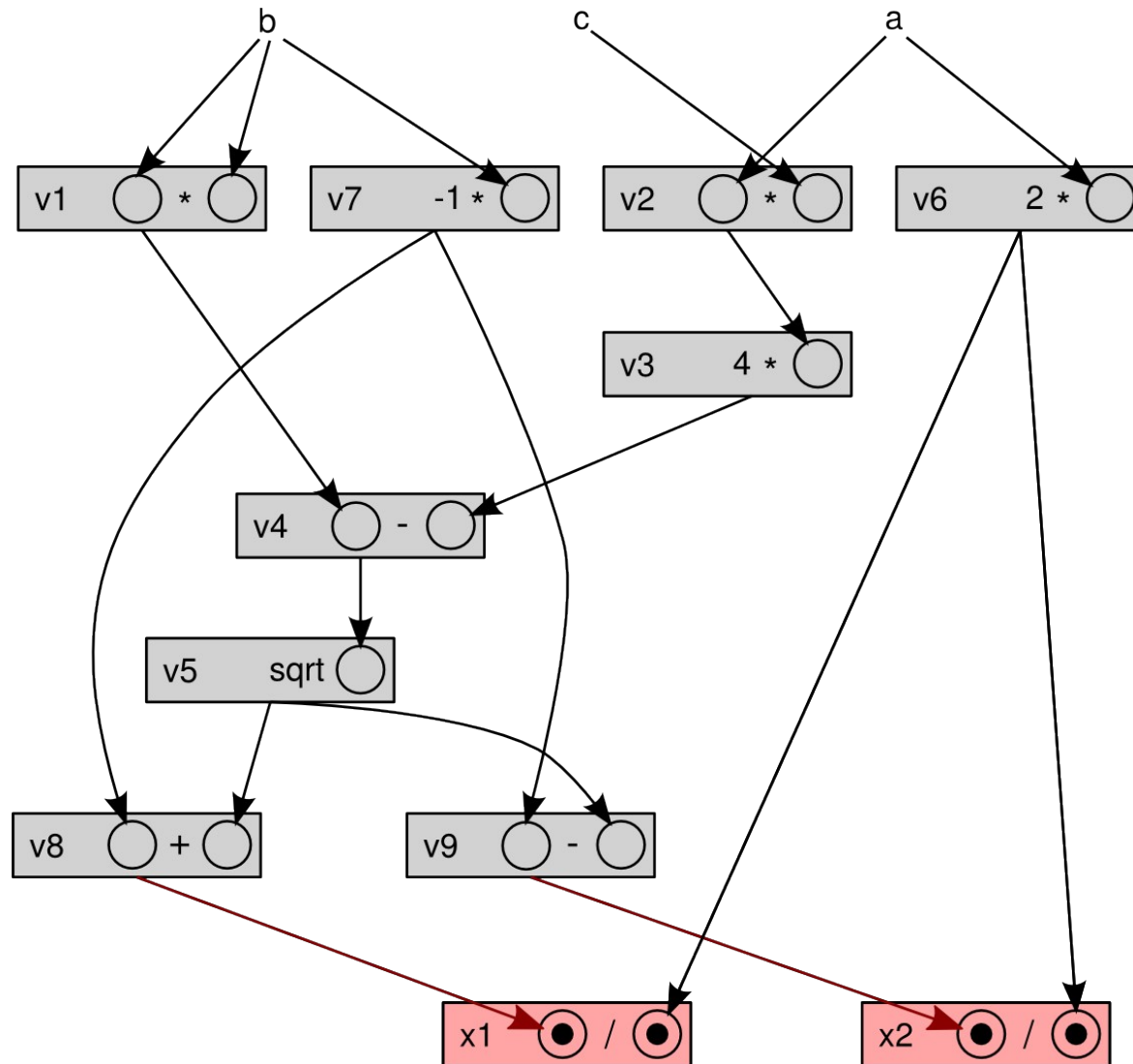








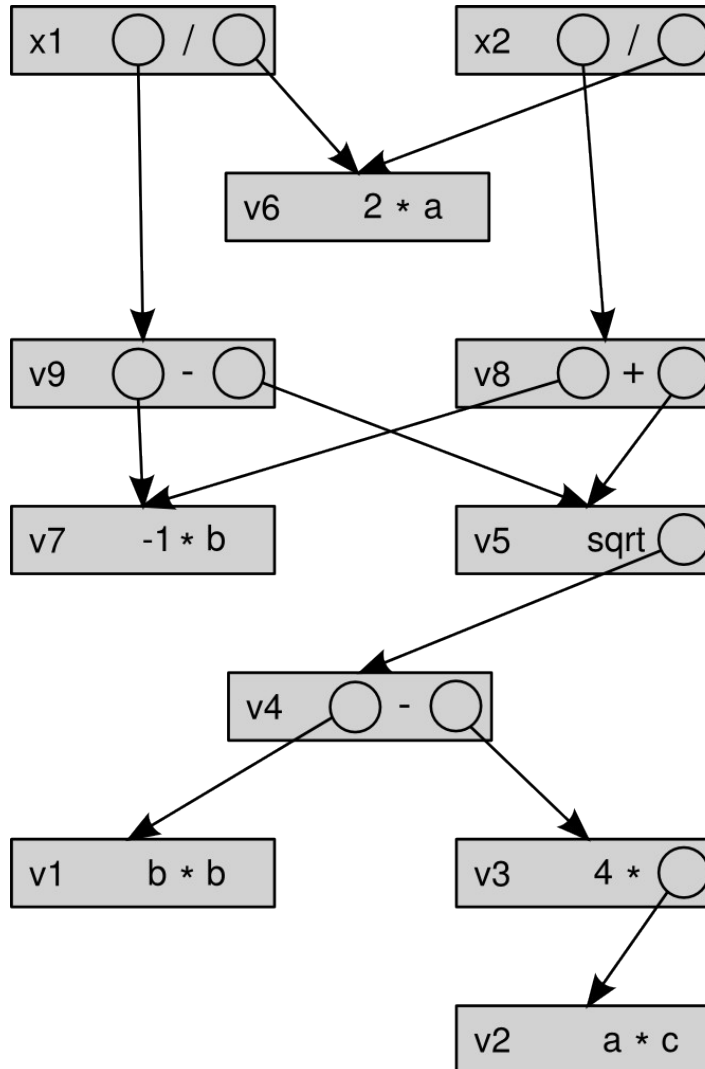




- Features

- It is able to detect and utilize parallelism
- Pure data-flow computers did not become popular
- But the principle did become popular:
  - Spreadsheets
  - Out-of-order execution of instructions in modern CPUs
  - Data-processing systems (Apache PIG)

- Execution of the instructions is automatic:
  - **Every instruction is executed as soon as another instruction needs its result**
- **Comparison:** when instructions are executed
  - Control-flow: when the control token selects it
  - Data-flow: when all operands are available
  - Demand driven: when another instruction needs its result
- Example: solutions of quadratic equations



## DEMAND DRIVEN MODEL

$$x1 \quad \bigcirc / \bigcirc$$

$$x2 \quad \bigcirc / \bigcirc$$

$$v6 \quad 2 * a$$

$$v9 \quad \bigcirc - \bigcirc$$

$$v8 \quad \bigcirc + \bigcirc$$

$$v7 \quad -1 * b$$

$$v5 \quad \text{sqrt} \bigcirc$$

$$v4 \quad \bigcirc - \bigcirc$$

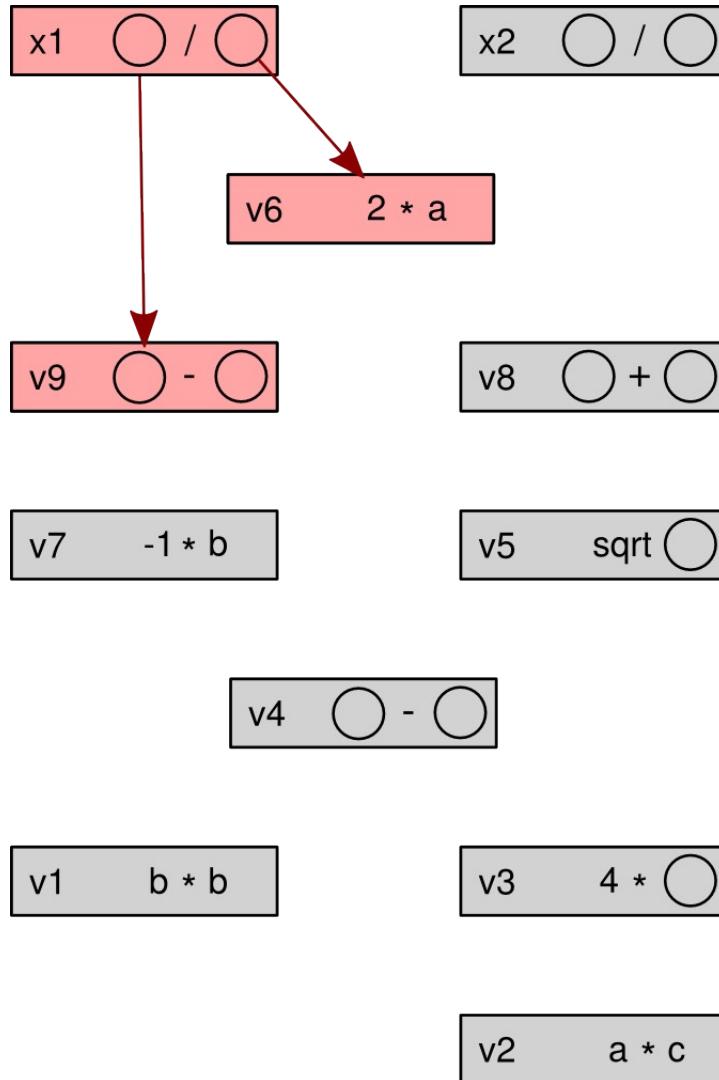
$$v1 \quad b * b$$

$$v3 \quad 4 * \bigcirc$$

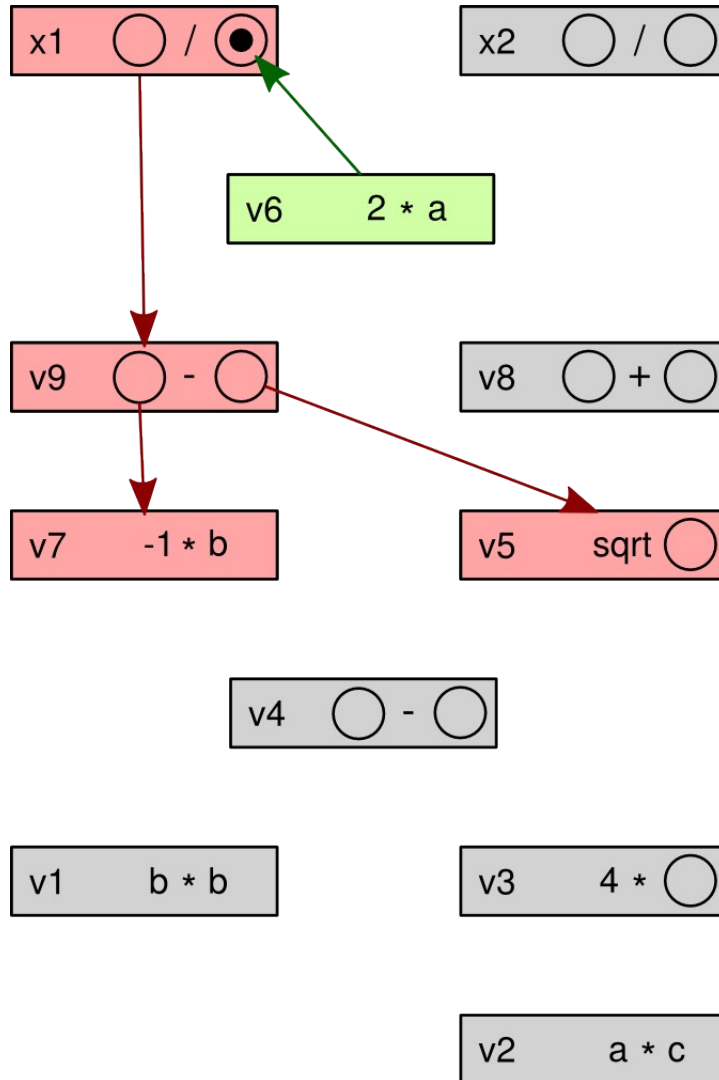
$$v2 \quad a * c$$



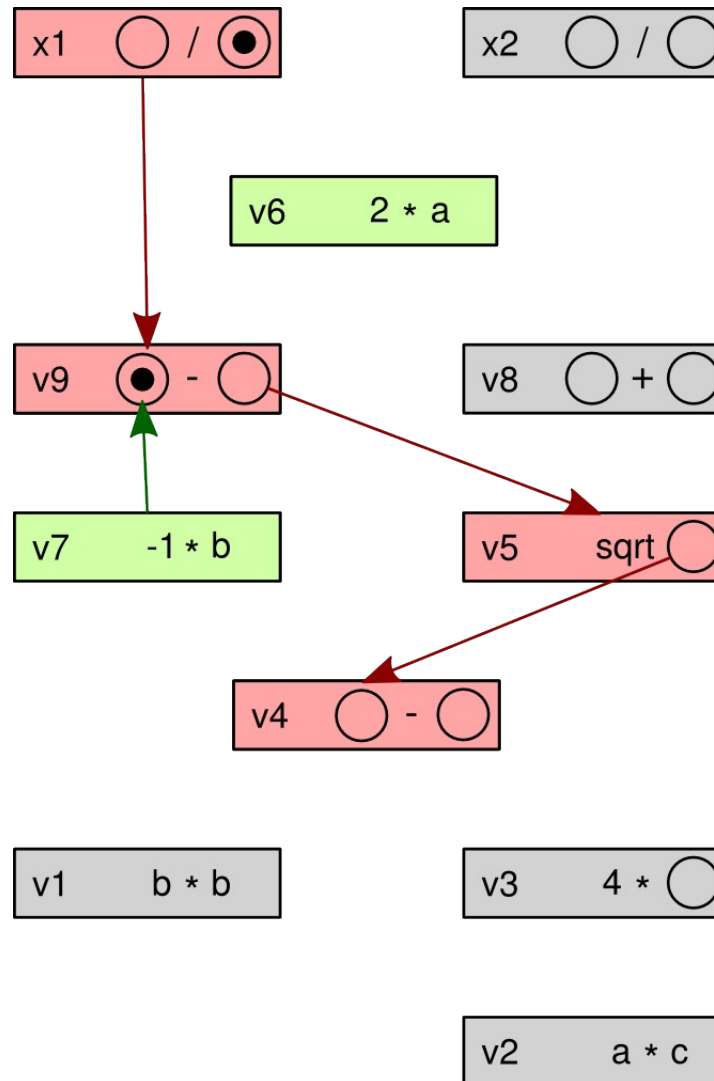
## DEMAND DRIVEN MODEL



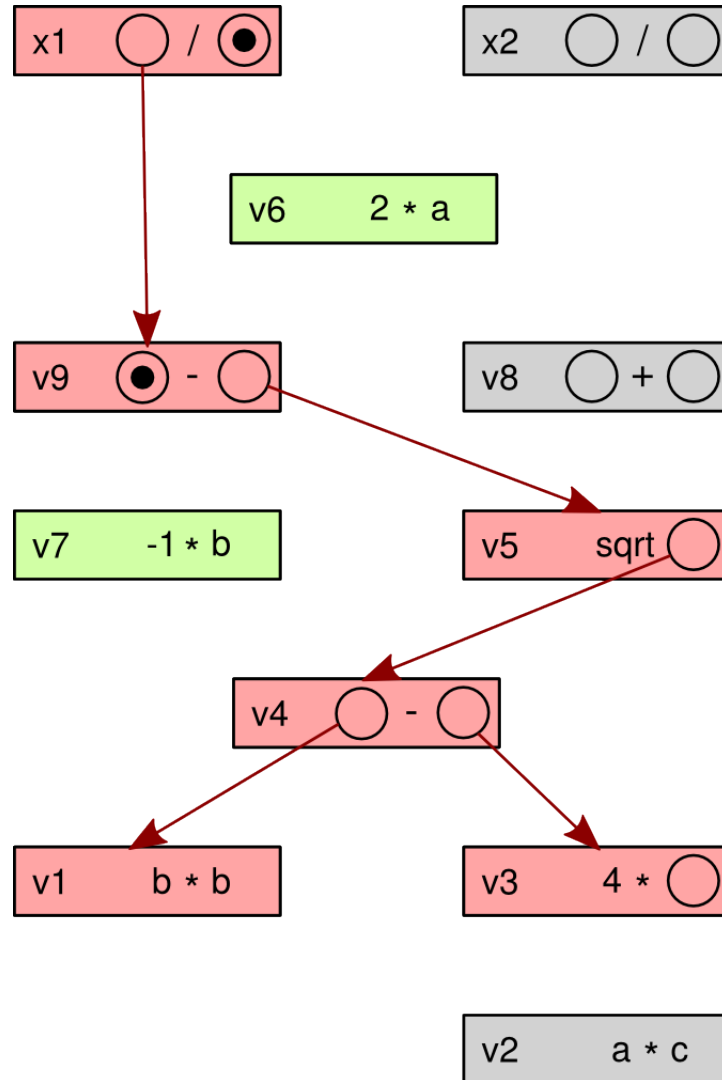
## DEMAND DRIVEN MODEL



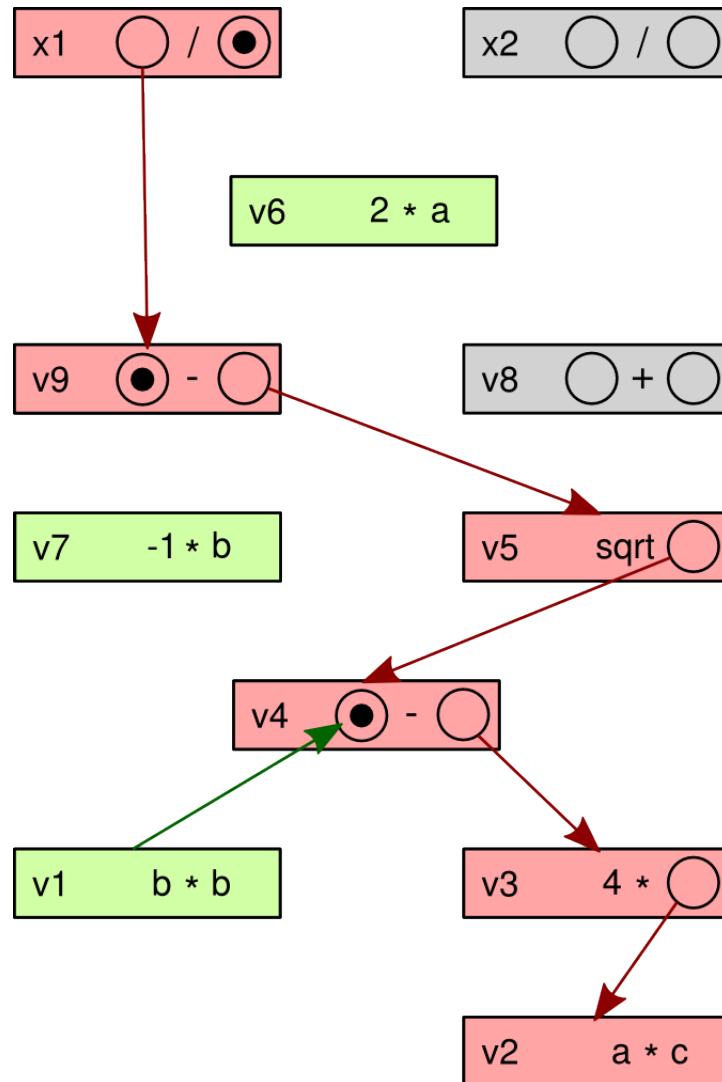
## DEMAND DRIVEN MODEL



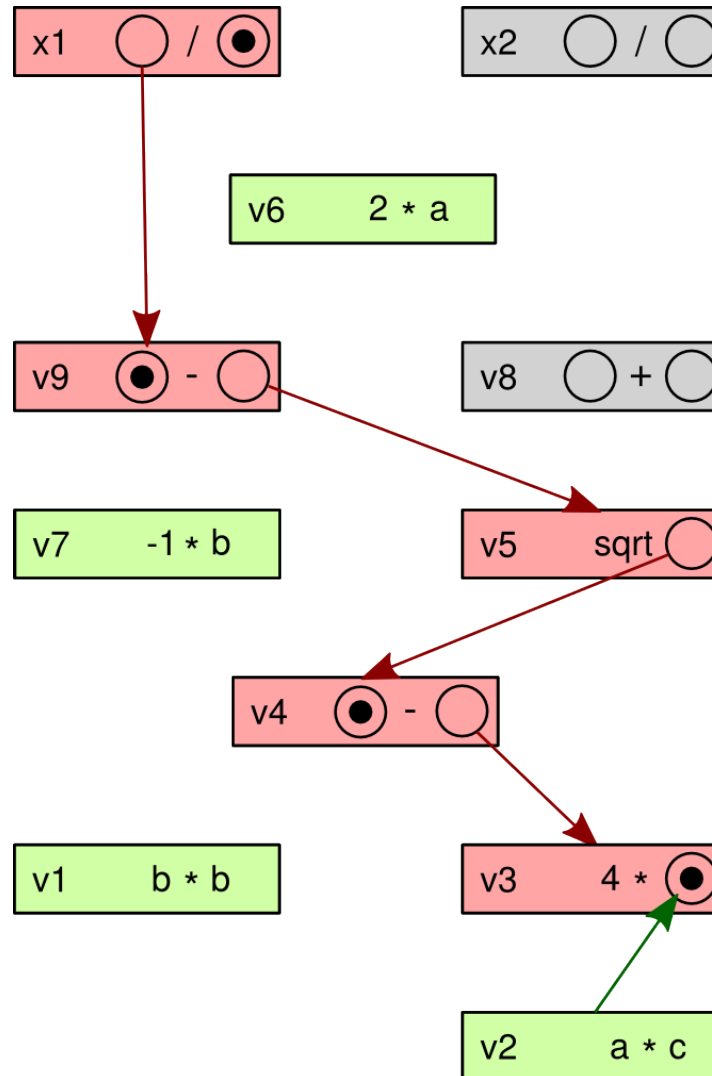
## DEMAND DRIVEN MODEL



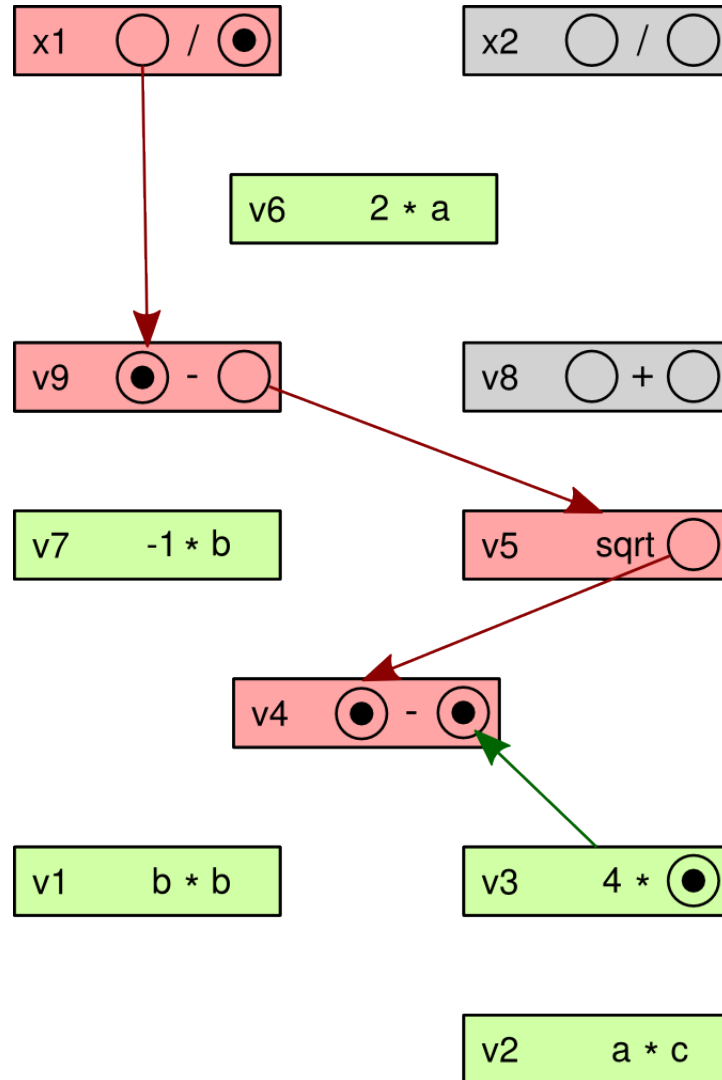
## DEMAND DRIVEN MODEL



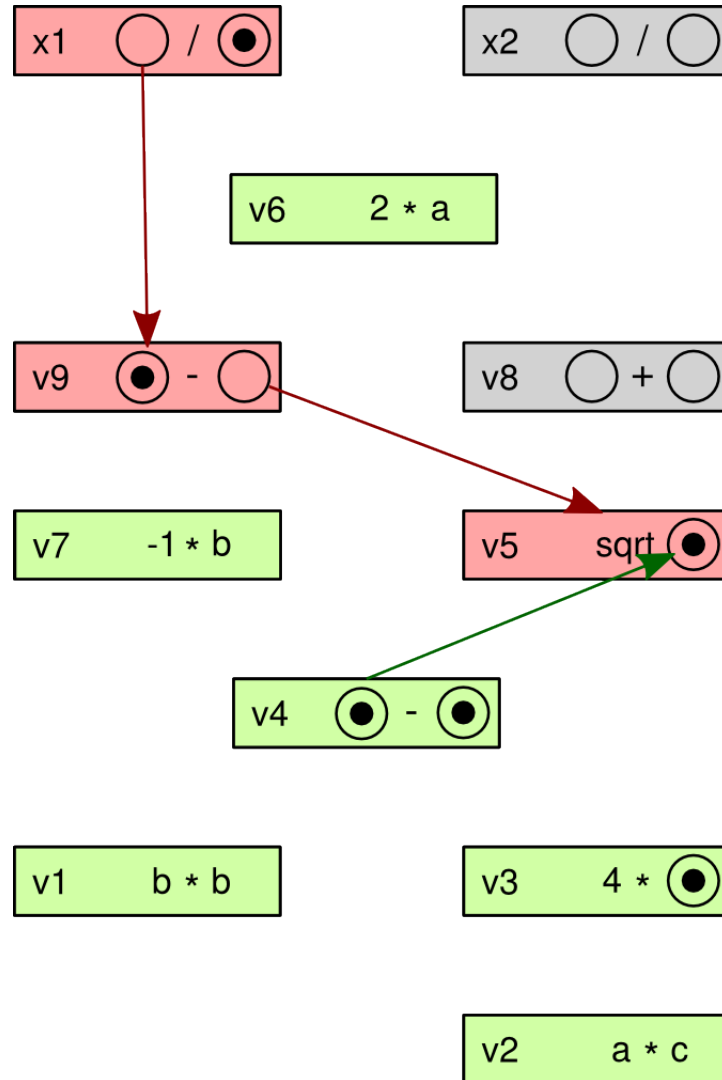
## DEMAND DRIVEN MODEL



## DEMAND DRIVEN MODEL

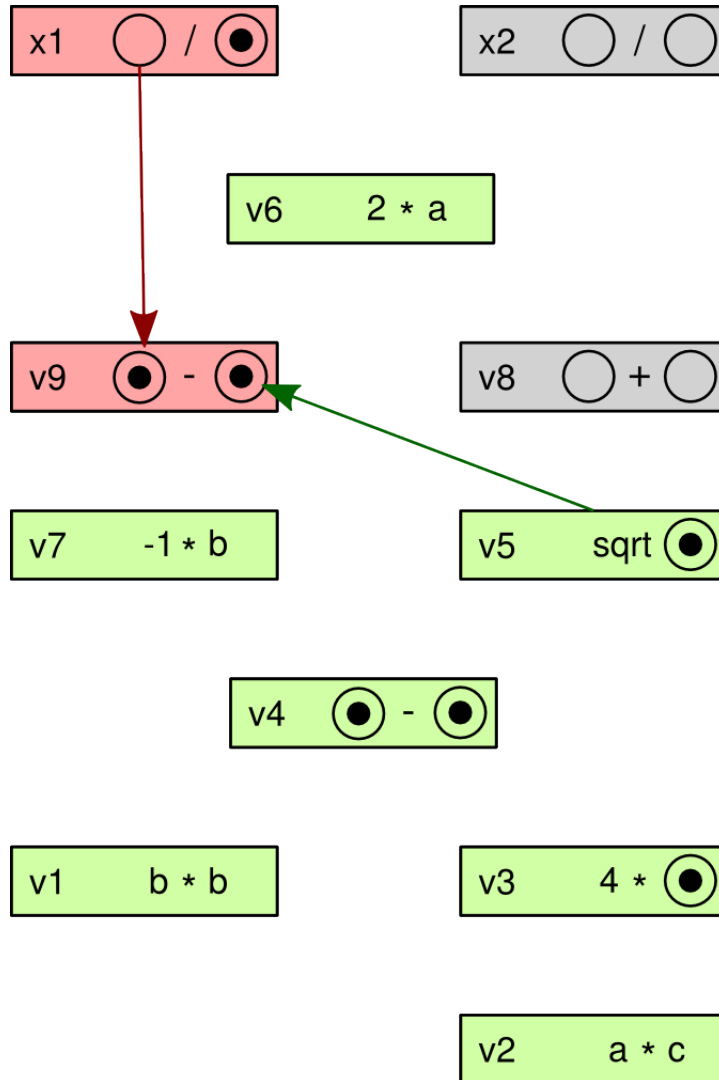


## DEMAND DRIVEN MODEL

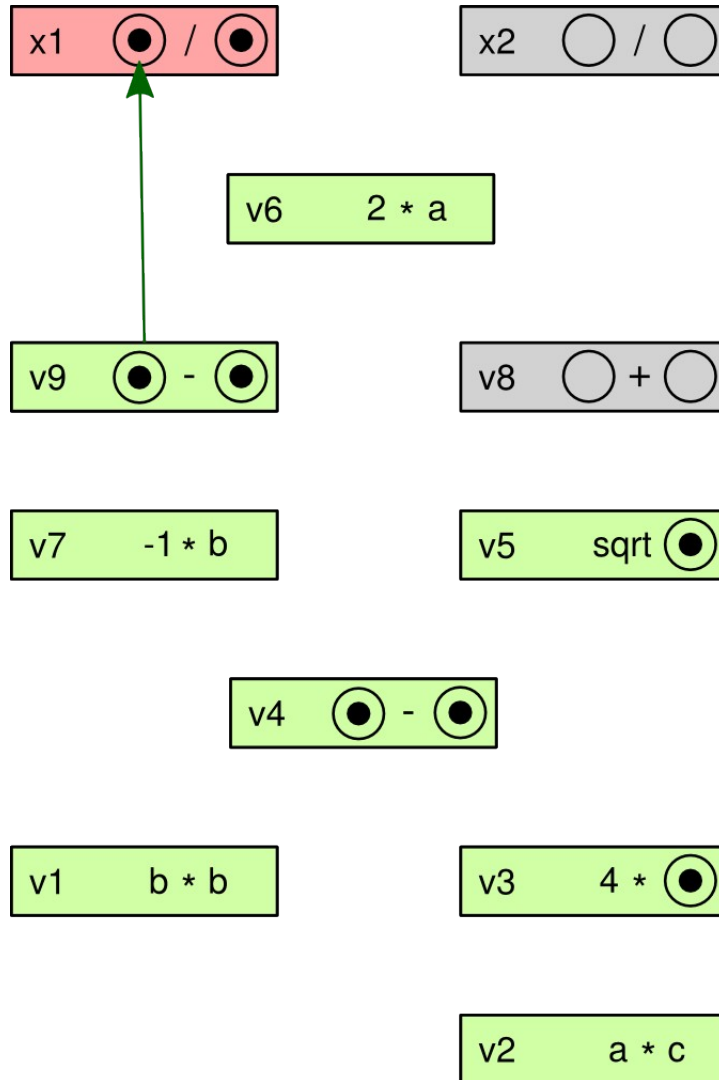




## DEMAND DRIVEN MODEL



## DEMAND DRIVEN MODEL



## DEMAND DRIVEN MODEL

x1  $\odot / \odot$

x2  $\bigcirc / \bigcirc$

v6  $2 * a$

v9  $\odot - \odot$

v8  $\bigcirc + \bigcirc$

v7  $-1 * b$

v5  $\text{sqrt}(\odot)$

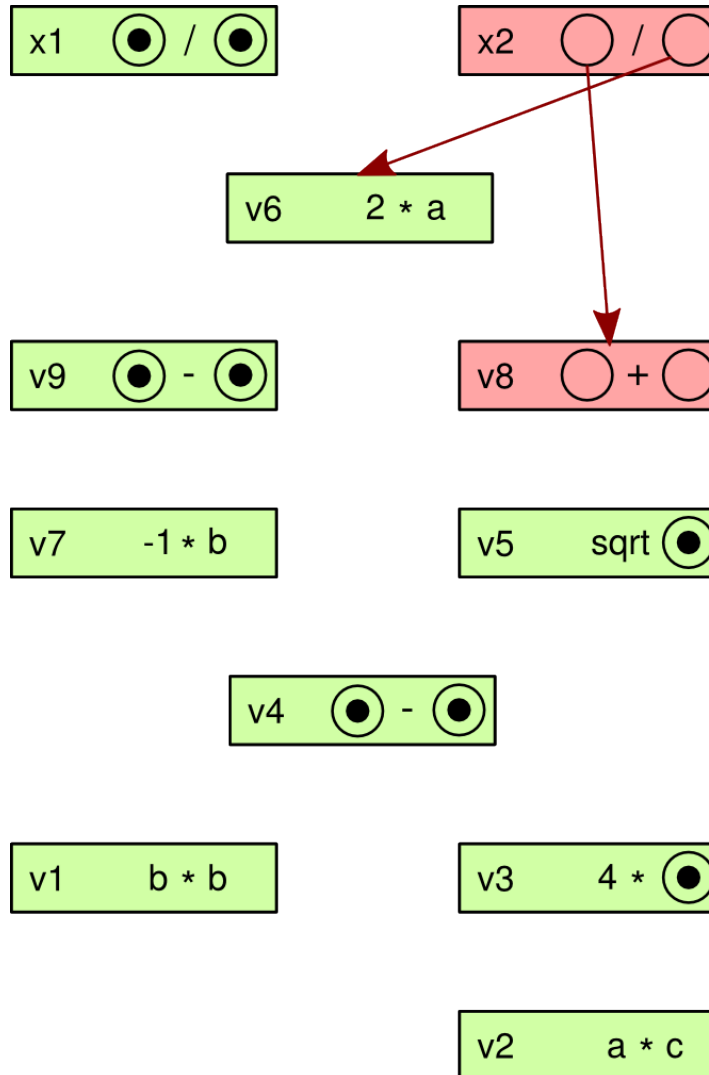
v4  $\odot - \odot$

v1  $b * b$

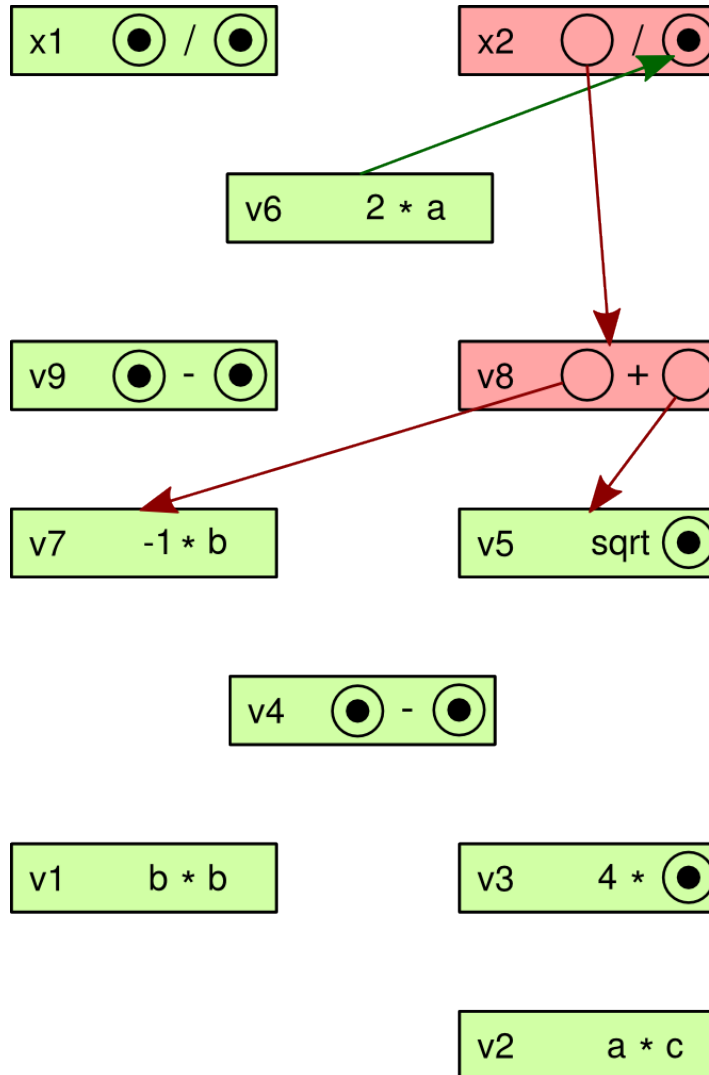
v3  $4 * \odot$

v2  $a * c$

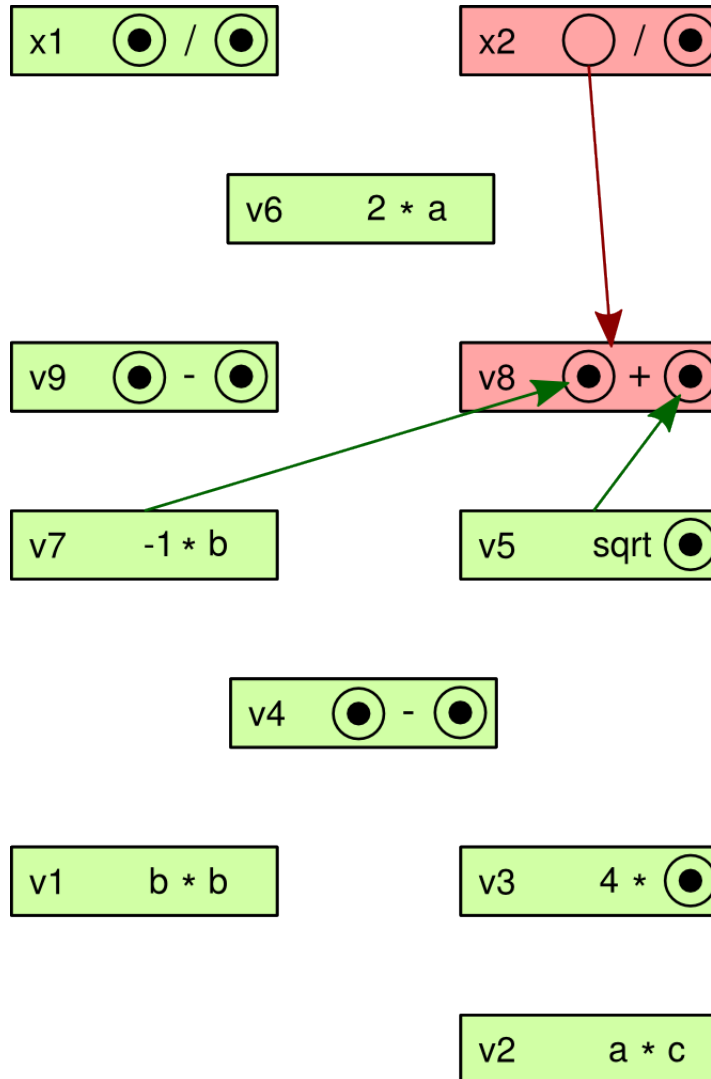
## DEMAND DRIVEN MODEL



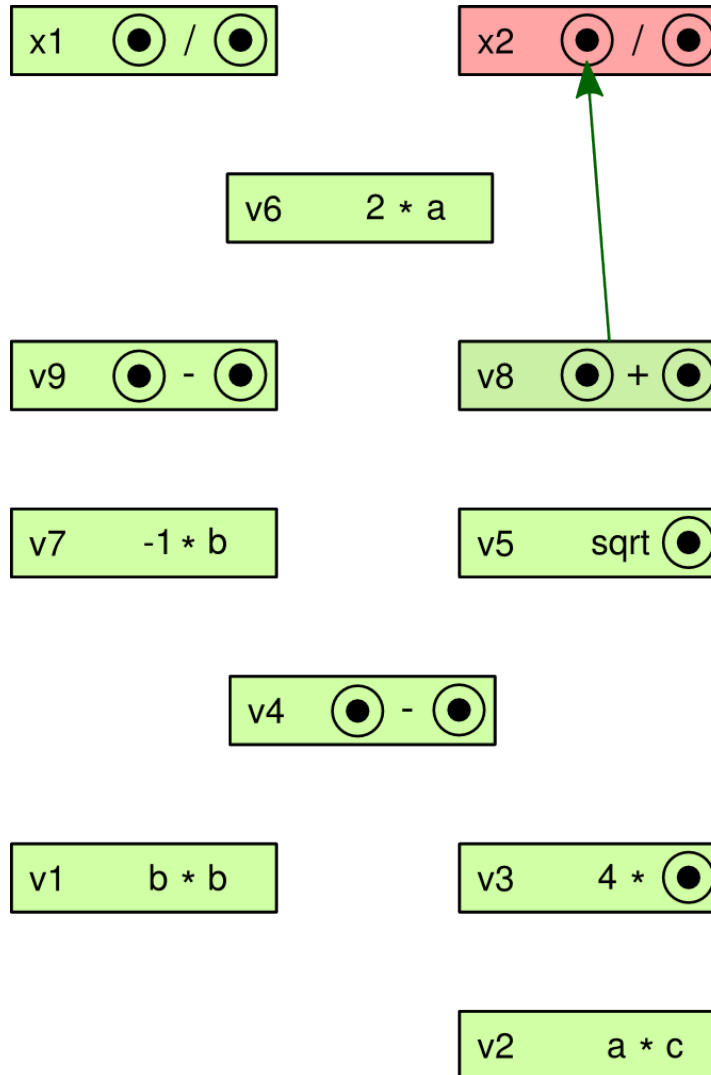
## DEMAND DRIVEN MODEL



## DEMAND DRIVEN MODEL



## DEMAND DRIVEN MODEL



## DEMAND DRIVEN MODEL

$$x1 \quad \odot / \odot$$

$$x2 \quad \odot / \odot$$

$$v6 \quad 2 * a$$

$$v9 \quad \odot - \odot$$

$$v8 \quad \odot + \odot$$

$$v7 \quad -1 * b$$

$$v5 \quad \text{sqrt}(\odot)$$

$$v4 \quad \odot - \odot$$

$$v1 \quad b * b$$

$$v3 \quad 4 * \odot$$

$$v2 \quad a * c$$



- Features
  - It is able to detect and utilize parallelism
  - Pure demand driven computers did not become popular
  - But the principle did become popular:
    - Functional programming languages
    - Google MAP-REDUCE paradigm

- Summary:
  - The programming interface of most CPUs follows the control-flow model
  - Functional languages allow demand-driven programming over CPUs following the control-flow model
  - CPUs following the control-flow model may use data-flow model based re-ordering of the instructions internally



## ***Control-driven architectures***

# THE NEUMANN ARCHITECTURE

- „Official story”:
  - 1945, "First Draft of a Report on the EDVAC"
  - First computer following a Neumann architecture: Institute for Advanced Study, Princeton University (1945-1951)
- The truth:
  - Turing – 1936: first paper describing the concept, but computer scientists were not aware of it
  - Eckert & Mauchly – 1943: Neumann wrote a paper based on the ideas of Eckert & Mauchly. His colleague started to circulate it with only Neumann's name on it.

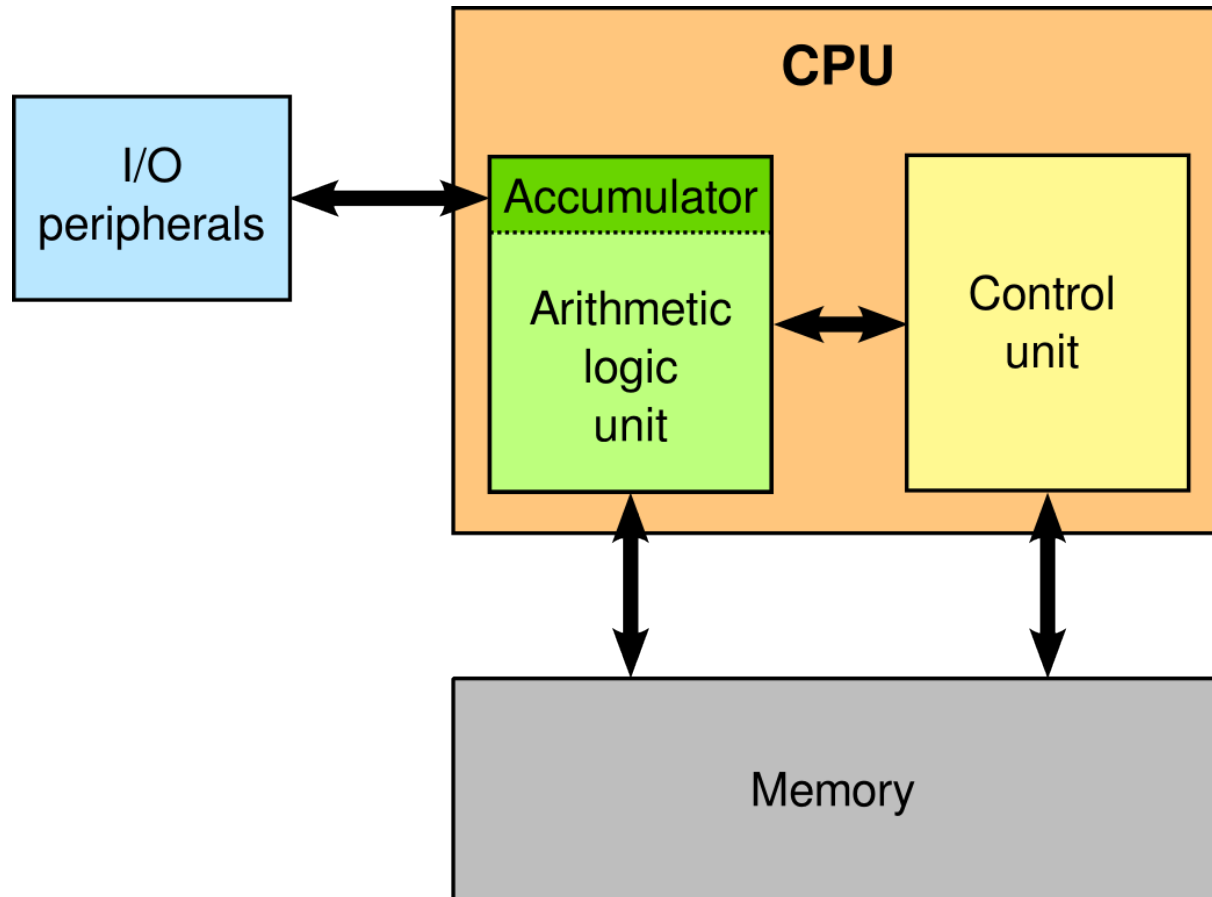


- Novelty:

**The instructions are stored in the memory together with the data**

- Earlier (before Neumann):  
programming = re-building or re-wiring the computer
- Main motivation behind the Neumann architecture:
  - Easier to change the program  
(A new program for a non-Neumann machine like ENIAC took 3 weeks to set up)
  - It is easier to run several programs one after the other
  - It made it possible to write programs generating a program
    - The birth of compilers

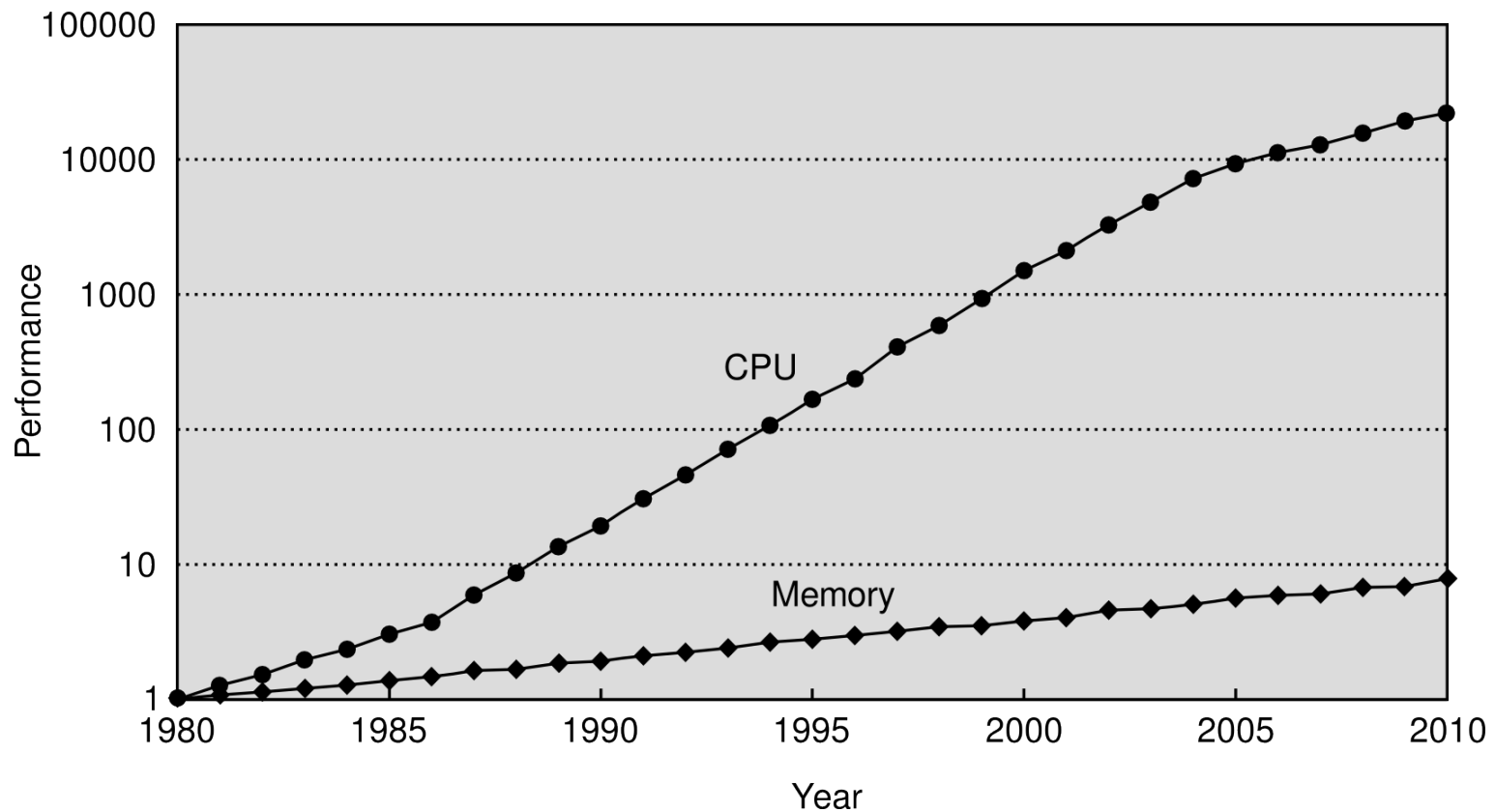
## THE NEUMANN ARCHITECTURE



- Content of the memory:
  - Data units of fixed size
  - Can be:
    - Instruction
    - Integer number
    - Character
    - Floating point number
    - ...
  - How do we know what is stored in the memory at a given address?
  - **We do not know it!**
  - It depends on what kind of instruction is accessing it

# THE NEUMANN ARCHITECTURE

- Bottleneck:
  - **Memory bandwidth!**



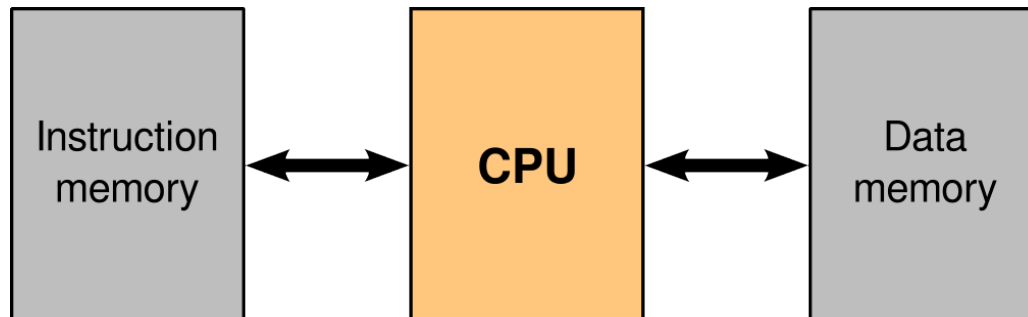


- Self-modifying programs
  - Replace their own instructions by memory operations
  - Some algorithms can be shorter this way
  - Self modifying programs are impossible to maintain
  - Never do it!

## THE HARVARD ARCHITECTURE

- Harvard Mark I:
  - Reads the instructions from a punched card
  - Data is stored by a memory consisting of relays
- Principally different from the Neumann architecture :

**The instructions and the data are stored separately**



- The classical Harvard architecture
  - Instructions: exclusively in the instruction memory
  - Data: exclusively in the data memory
- The two kinds on memory accesses can be overlapped:
  - While fetching the operands of the  $i$ -th instruction from the data memory...
  - ...the processor can fetch the  $(i+1)$ -th instruction from the instruction memory at the same time
- Modified Harvard architecture:
  - The instruction memory can be read by the program as well
  - It can store constant data
- Typical applications:
  - Micro-controllers, digital signal processors (PIC, Atmel, ...)



# ***Topics***

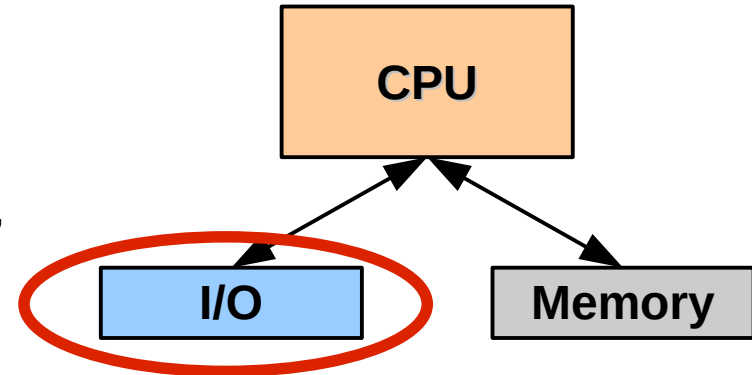
- **I/O peripherals**

- Lectures

- Flow control, decreasing the load of CPU, busses, south bridge – north bridge, etc.
- Mass storage drives: HDD/SSD
- Interfaces: PCI, PCI Express, USB

- Classroom practices

- Project for reviewing digital design:
  - A CPU based door lock project
- Numerical examples related to I/O peripherals
  - Computing load induced by I/O peripherals
  - HDD delay and throughput calculations
  - SSD block management and garbage collection algorithms



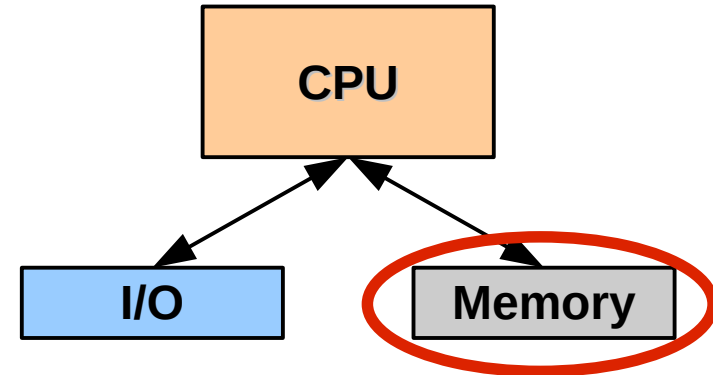
- **Memory**

- Lectures

- Memory technologies: SRAM, DRAM, SDRAM, DDR, DDR2, DDR3, DDR4, GDDR3, GDDR5, etc. memory timings
- Virtual memory system
- Cache memory
- Locality aware programming techniques

- Classroom practice

- Numerical examples
  - Scheduling DRAM commands
  - Page table management
  - Cache management



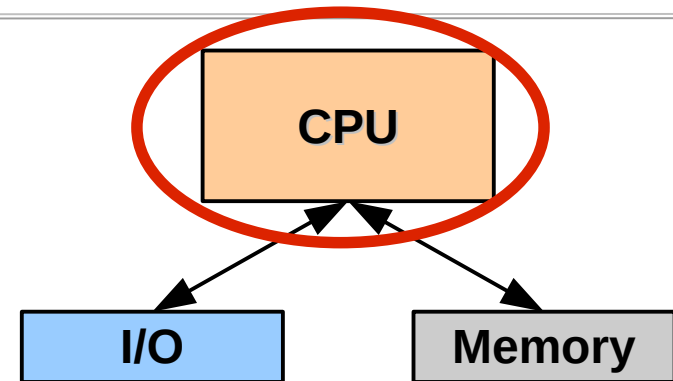
- **CPU**

- Lecture

- Pipeline instruction processing
- Out-of-order instruction scheduling
- Superscalar processors
- Wide pipelines
- Branch prediction

- Classroom practice

- Numerical examples:
  - Instruction scheduling, identifying hazards and dependencies
  - Optimizing the order of instructions



- **Parallel systems**
  - SIMD processing
    - Vector processors
    - Vector instruction sets
    - Vectorization
  - Multiprocessor systems
    - Multi threaded and multi-core systems
    - Multiprocessor systems
    - Interconnections
    - Shared memory management





DEPARTMENT OF  
NETWORKED SYSTEMS  
AND SERVICES

