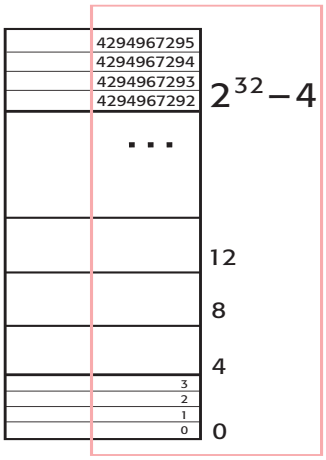
Address Space

Memory addressing model



32-bit address space

Quiz
Define the term **address space.**

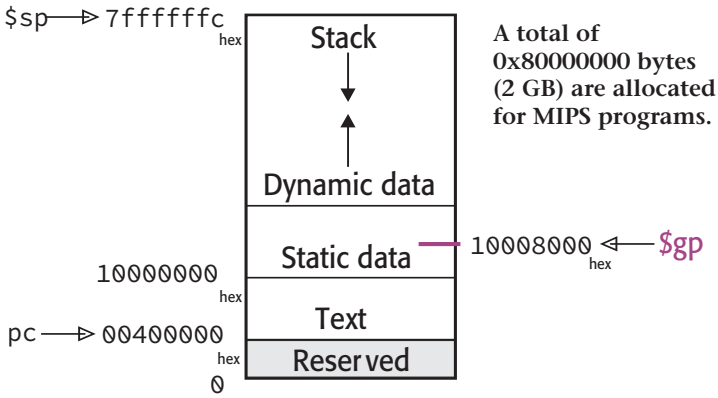
Programmers visualize memory logically as numbered boxes (each store a byte of info).

In reality, where info actually comes from depends on underlying memory hierarchy!

MIPS memory

Program v.addr space (bytes): 0–2147483647

(0x00000000-7fffffff)



A total of 0x80000000 bytes (2 GB) are allocated for MIPS programs.

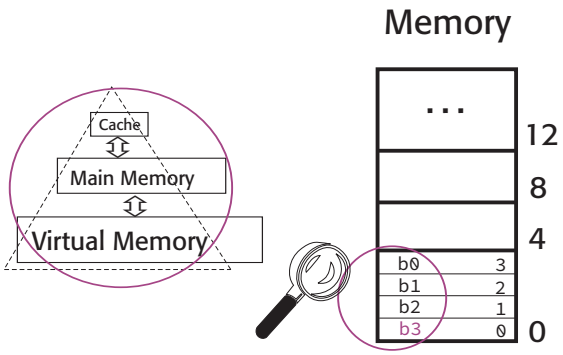
A convention (standard way) to organize how the 2 GB v.addr space is used makes programs easier to read and debug.

Virtual Memory

SRAM

DRAM

Historically magnetic storage but increasingly persistent solid-state tech (flash mostly as of 2019).



VM features close cooperation between hardware (usually integrated with the processor) and software (usually part of the OS). It's invisible to user programs.

Quiz
What *endianess* (big or littel) is depicted in figure? (A low level assembly programming concept).

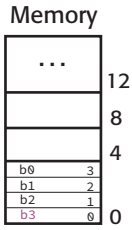
To manage memory efficiently, VM systems provide 2 important services: **protection** (private address space) and **relocation** (independent mapping to physical memory).

Transparently

- Exceed real memory size limitations
- Manage shared memory efficiently

Virtual Memory Concepts

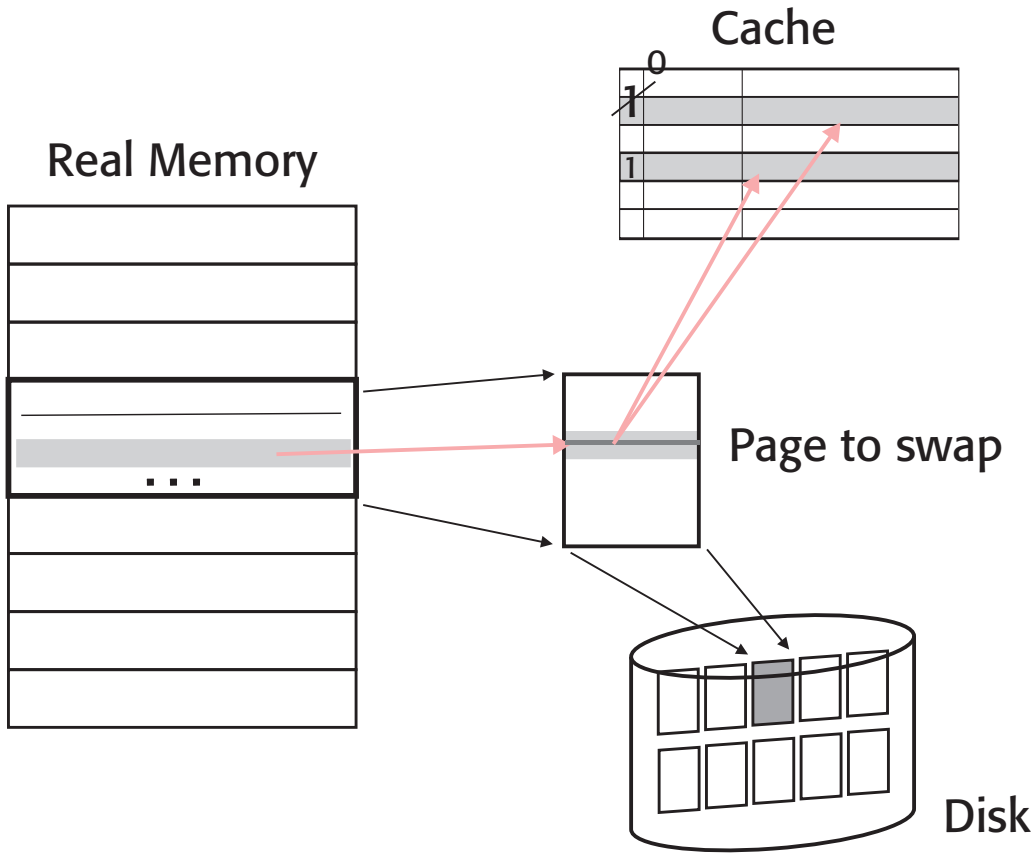
The terms **real memory**, "physical" memory, "primary" storage, or "main" memory are equivalent and usually refer to physical locations used for *active* code, currently implemented in DRAM technology and popularly known as RAM.



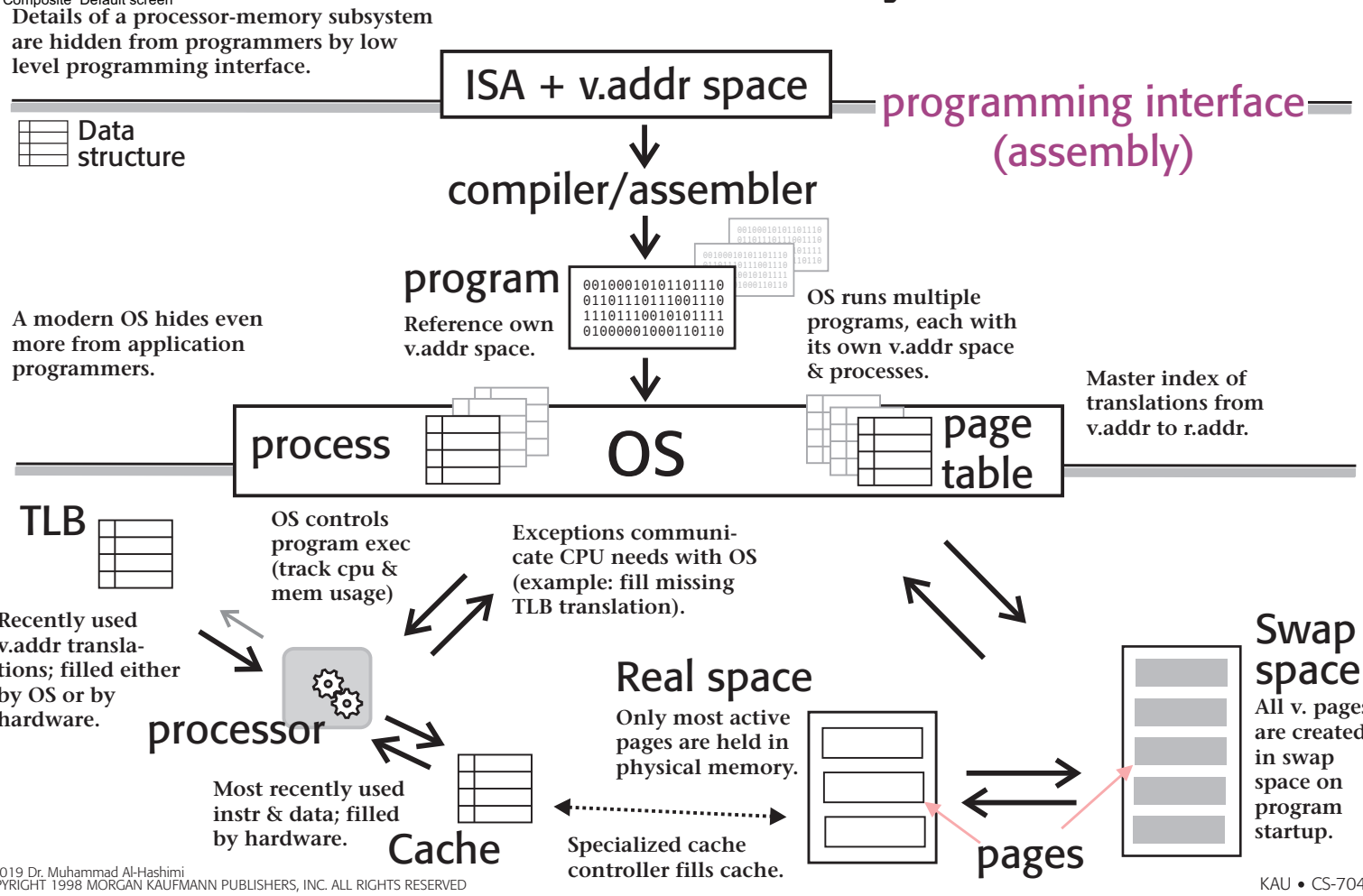
Virtual = not really, but in effect (practically)

Virtual address space corresponds to the concept of memory in the von Neumann model: memory *apparently* available to programs. In practice, addresses in that memory may correspond to "real" locations or to overflow locations in magnetic/flash storage (not real).

- **Virtual vs. real memory**
- **Virtual addresses**
  - CPU generates virtual addresses
- **VM terminology**
  - VM block = page
  - VM miss = page fault
  - Mapping = address translation

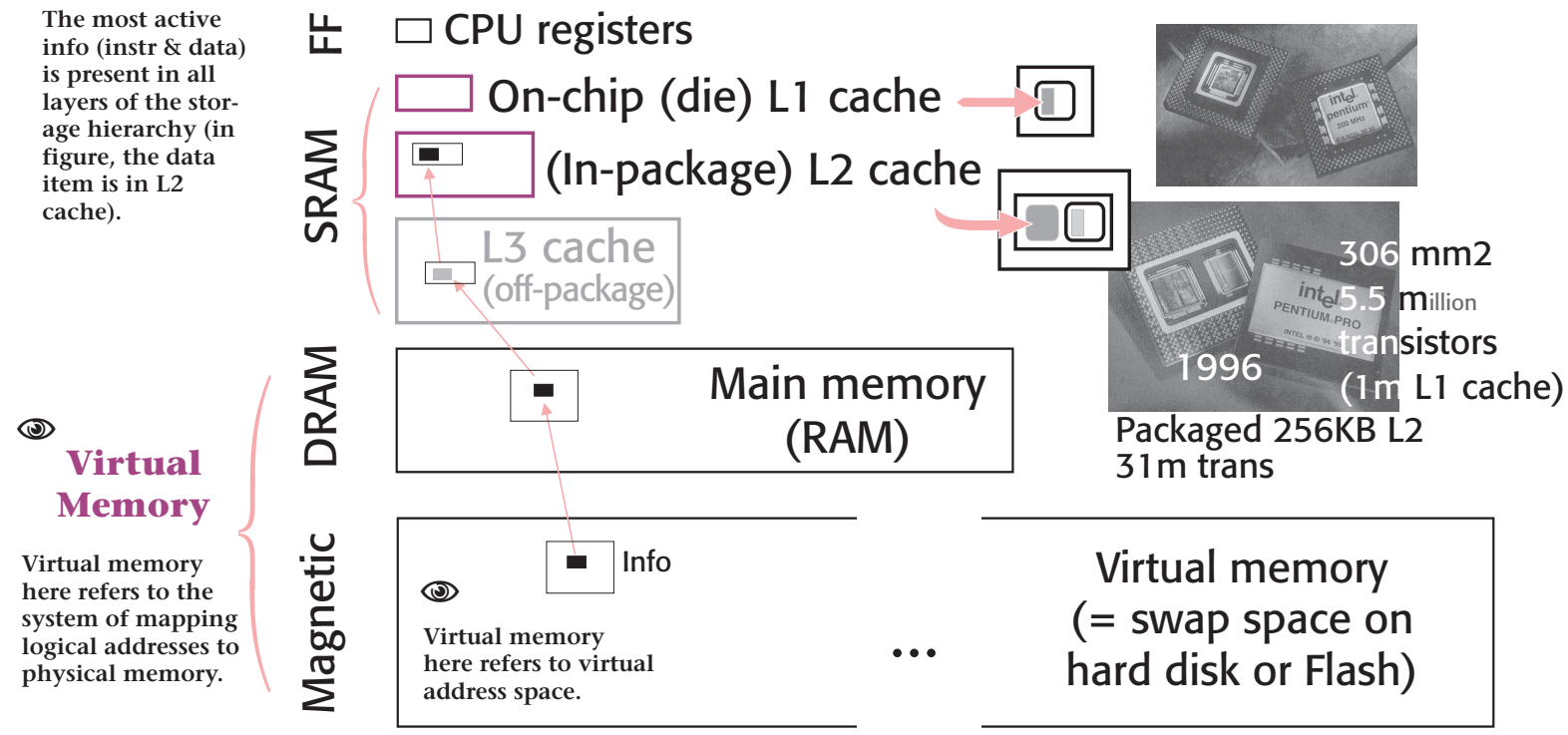


Hierarchy Overview



Concept Review
A Typical Hierarchy

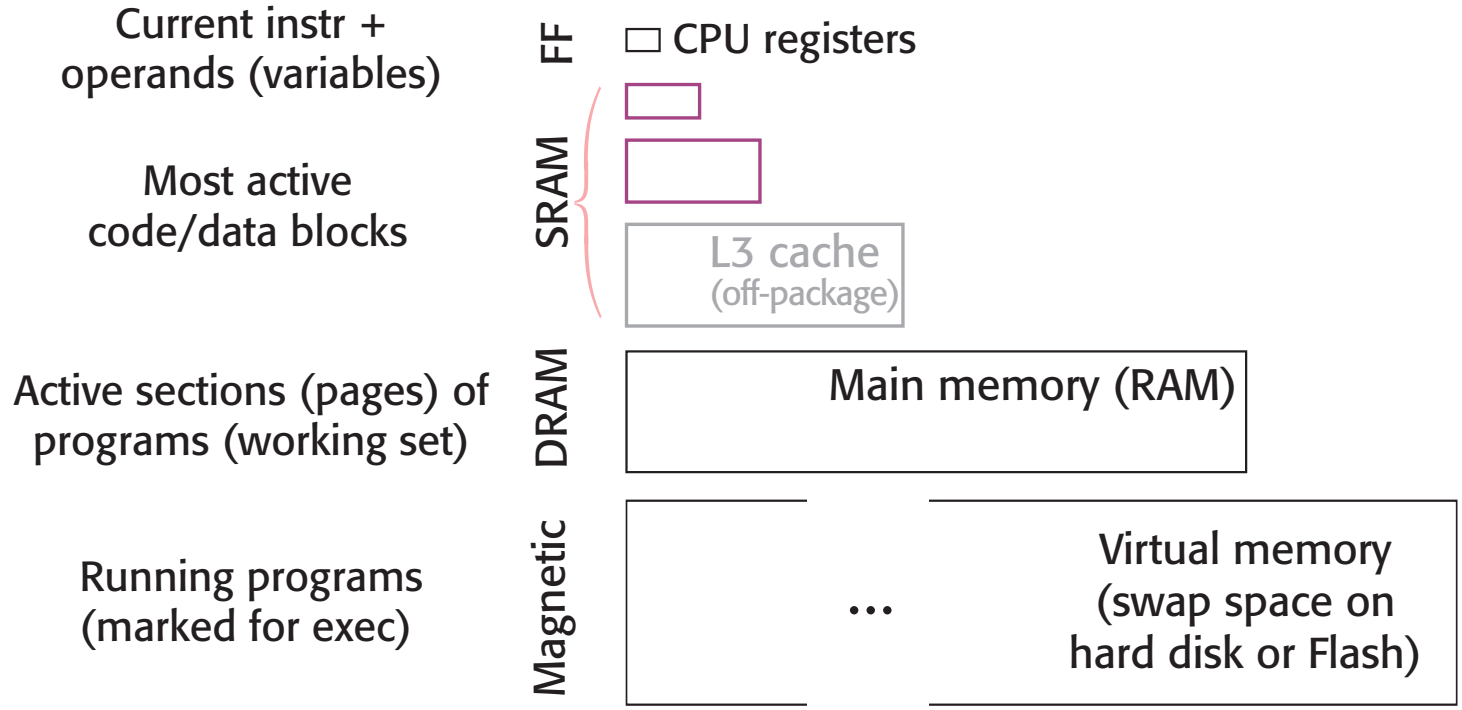
The most active info (instr & data) is present in all layers of the storage hierarchy (in figure, the data item is in L2 cache).



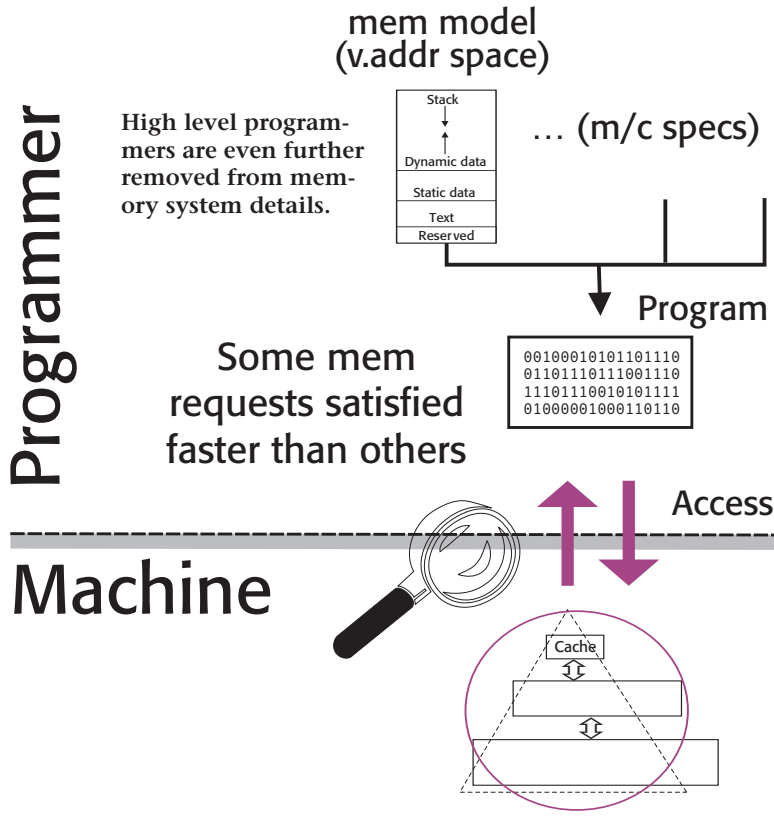
Virtual Memory

Virtual memory here refers to the system of mapping logical addresses to physical memory.

Concept Review
Info Hierarchy



Memory Systems
Program Performance



- **Memory models hide physical hierarchy**
- **Machine code will interact with the hierarchy**
- **Program performance will vary**

Some performance problems can only be resolved by examining hierarchy specifics and adjusting code, sometimes the algorithm

Program Performance

- ☐ **Impact of ISA**
  - ✎ Instruction selection
  - ✎ Instruction design
- ☑ **Program**
  - ✎ Algorithm selection
  - ✎ Programming environment: lang + compiler
- ☐ **Microarchitecture**
  - ✎ Datapath(s) + control streams
  - ✎ Processor and instruction-level parallelism
- ☑ **Memory hierarchy**
- ☐ **Impact of I/O**

Different parts of a computing system interact to affect program performance.

For example, suitably designed ISA can result in cost-effective pipelined microarchitecture which economically delivers power efficient high instruction execution rates. (Such as MIPS and ARM).

Program Performance Example

📖 P&H (4th Revised)
Chapter 2: pp 156
benchmark details

High-level languages try to balance productivity requirements of programmers against performance requirements (e.g., efficient use of machine resources).

Some languages such as Java favor programmer efficiency, while others such as C favor machine efficiency.

Modern bla bla.. Python vs. Rust bla bla

Search set size = 100K words (32-bit)

Language	Exec Method - optimization	Bubble Sort relative performance	Speedup Quicksort vs. Bubble Sort
C	Compiled - none	1.00	2468
	Compiled - best	2.41	
Java	Interpreted	0.12	1050
	Compiled (JIT)	2.13	338

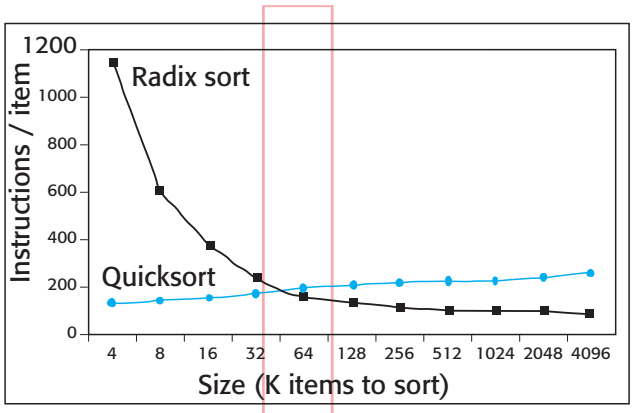
Quicksort algorithmic advantage

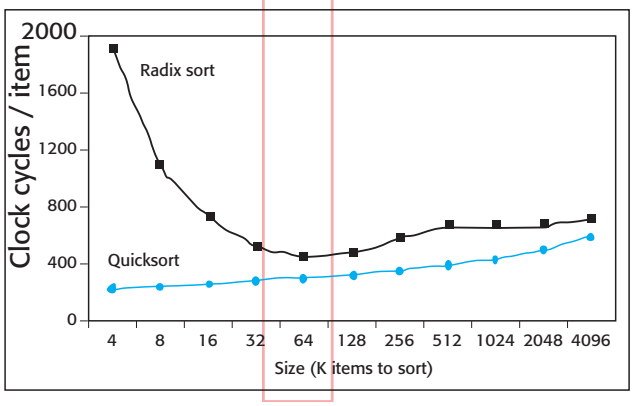
Language/compiler advantage

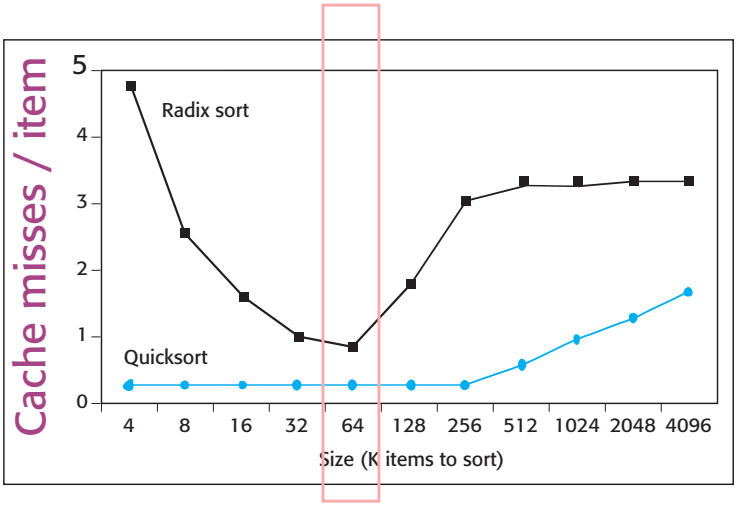
JIT (just-in-time) compilers compile on-demand during execution (compilation overhead at run-time).

Program Performance Example

📖 P&H (4th Revised)
Chapter 2: pp 490
benchmark details







Similarly, a program may interact negatively with the VM system by using too many pages at the same time, or generating too many TLB misses.

📖 Chapter 5: p.517
Understanding Program Performance

Program Performance Improvement

❶ ➪ Algorithmic advantage

Maximize computation efficiency

- ✎ Less operations (reduce work)
- ✎ Overlap operations (parallelize)
- ✎ Better shortcuts (heuristics)

❷ ➪ Language/Compiler advantage

Maximize code efficiency

- ✎ Less code per operation
- ✎ Better use of target machine

Program Performance Improvement

📖 P&H (4th Revised) Chapter 2,5: p.157, 490, 517
Understanding Program Performance

❸ ➪ Architecture advantage: maximize machine utilization

- ✎ Better ILP
- ✎ Use processing elements/cores
- ✎ More block usage before replacement on miss
- ✎ Reduced reliance on bandwidth limited interconnect channels

Although the 3 program performance factors can be controlled independently, they do affect each other.

For example, gaining the architectural advantage may require re-working the algorithm to alter memory locality in order to better reflect the architecture.

Q to Think About
Which performance factor would be easier to control?