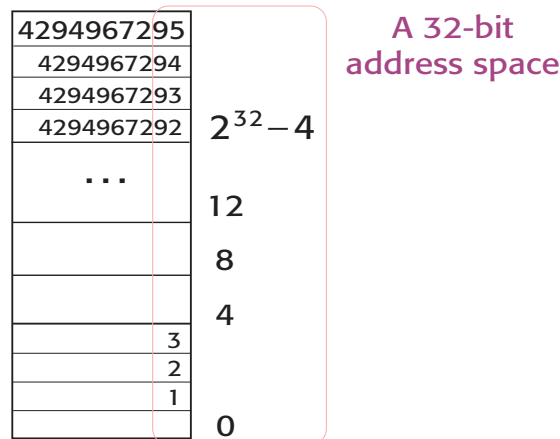


Memory Systems Address Space



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

Where info comes from depends on an underlying hierarchy of physical memories

Address Space Example

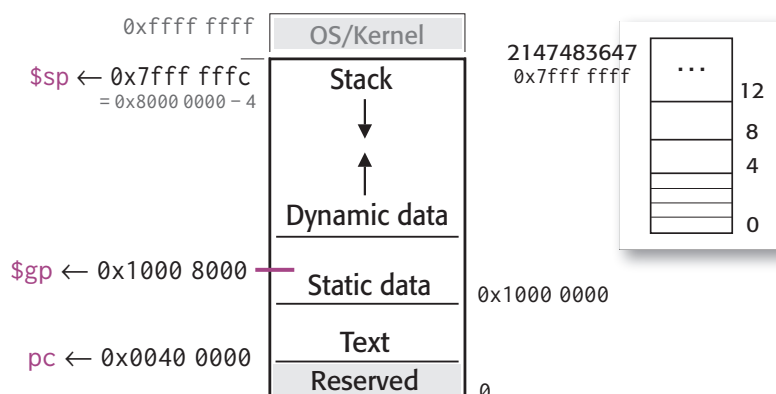
⇒ Object code

Total of 0x80000000 bytes (2 GB) is allocated for a MIPS program (rest of the 32-bit space reserved for the OS).

MIPS 32 program memory

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

- Reserved = OS exceptions code
- Text = program instructions
- Static data = persistent and compiler data (const, literals, global and static vars)
- Dynamic data = runtime heap (*malloc*/objects)
- Stack = procedure arguments, return values, and local vars



Program/object code addr space (bytes): 0 – 2147483647
(0x0000 0000 – 7fff ffff)

Virtual Memory

Virtual = not really, but in effect (practically).

Real memory (popularly known as RAM), also called primary storage or **main memory**, refers to physical memory locations used for **active** code, currently implemented in DRAM technology.

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

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



Virtual address space corresponds to memory *apparently* available to programs from a processor viewpoint (like the logical memory available to programmers).

Virtual vs. real memory

Role: transparently

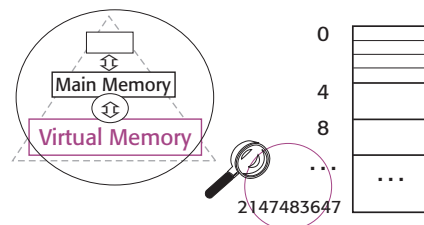


Exceed real memory size limitation



Manage shared memory efficiently

Virtual addresses



Virtual Memory Concepts

Each program can access a private copy of the entire virtual address space.

⇒ Virtual addresses

The addresses, always allocated in overflow locations kept in magnetic/flash storage (a **swap space**), may also map to real memory.

🔑 CPU generates virtual addresses

Different era different terms, same things essentially.

⇒ VM terminology vs cache

VM is an older technology where physical memory acts as a cache for memory kept in slow storage.

📎 VM block = page

Design is dominated by the huge difference in access time/latency between DRAM and magnetic storage.

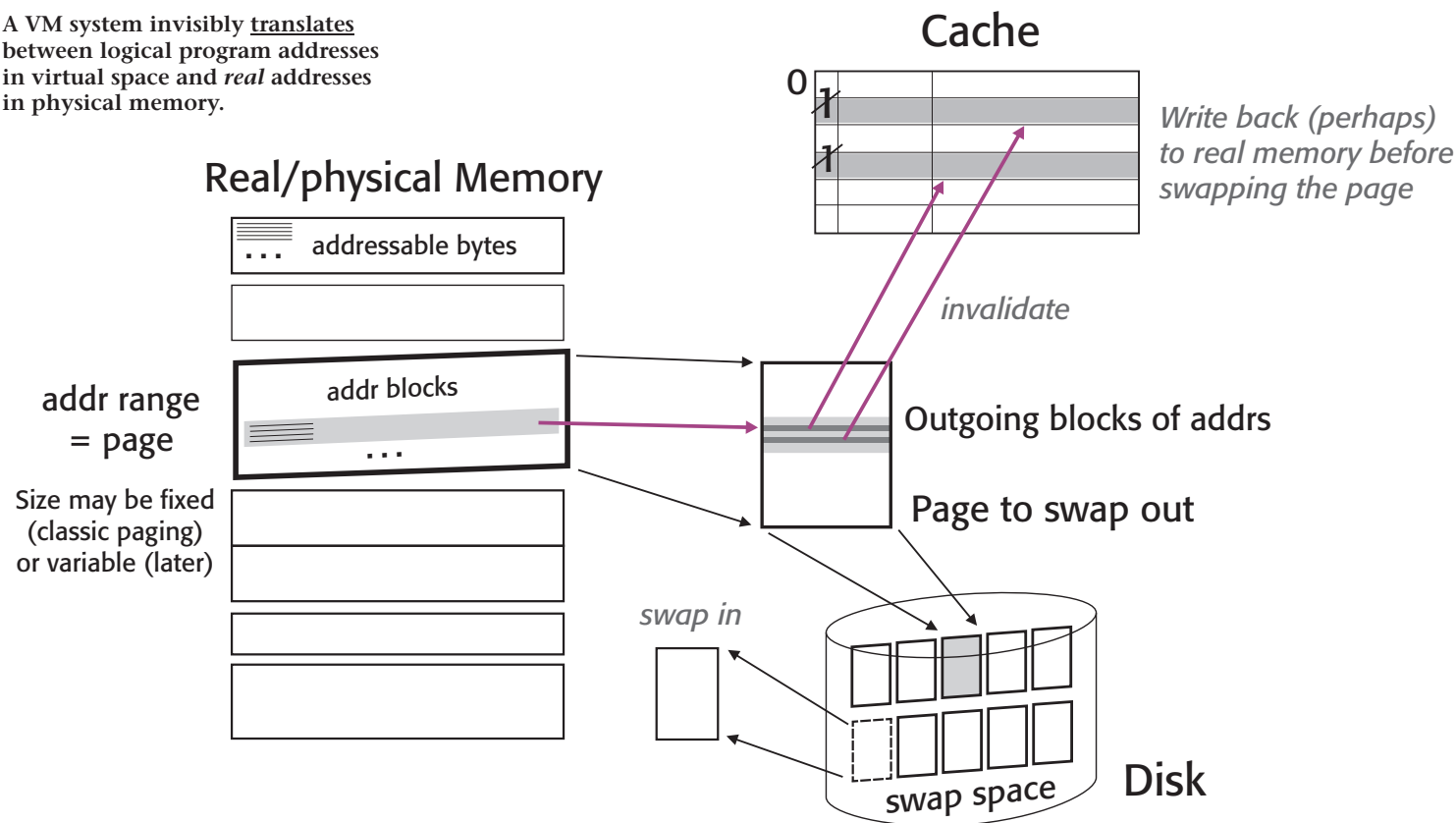
📎 VM miss = page fault

VM historically was about more memory, now mainly to abstract program address space and support robust, secure, and cost-effective multitasking.

📎 Mapping = address translation

Virtual Memory Scheme

A VM system invisibly translates between logical program addresses in virtual space and *real* addresses in physical memory.



© 2022 Dr. Muhammad Al-Hashimi

Memory Systems Info Hierarchy

⇒ Segmented VM

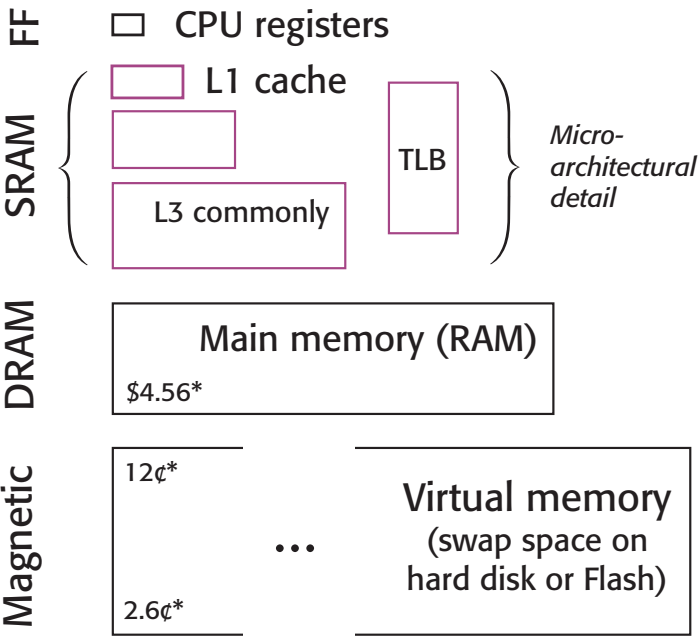
Current instr +
operands (variables)

Most active
code/data blocks

Active sections (pages) of
programs (a working set)

A more flexible VM scheme
uses variable-size address
block **segments** to map
logical-program to physical
memory.

Running programs
(marked for exec)



* Per GB (AVE). Consumer/retail newegg.com@2021/12/14 (best selling) • DRAM DDR3-1333 4/8G DIMM • SSD/NAND 256-1TB • HDD/internal 10-20 TB

Memory Systems

Real World Example

Exercise

Compare with preceding gen i5-750 to find the compromise.



Architectural compromise

Released January 2010, dual core with 2 levels of private cache per core run at 3.33 GHz; off-core die shared L3 cache + other parts of the CPU package, including DRAM controller, at 2.4 GHz.

Quiz

What is the processor clock cycle time in ns? Determine the DRAM latency in processor cycles.

Clearly, L3 cache is designed to minimize expensive miss penalty from DRAM.

Separate die memory controller results in higher access cost than expected from the SDRAM (48.75–52.5 ns*).

* JEDEC, DDR3 SDRAM Specifications, 2010.

Intel Core i5-661, Clarkdale

Clock 3.33/2.4 GHz, associative cache, 64-byte block, DDR3-1066 SDRAM



L1 split 32 KB/4-way, 32 KB/8-way (data)



L2 unified 256 KB 8-way



L3 4 MB 16-way @ 2.4GHz



Latencies:

L1	L2	L3	DRAM
4c	10c	39c	76.4 ns

Compare DRAM latency 1999–2015: 46–60 ns ($t_{RAS}+t_{RP}$), Kevin K. Chang et al. (pp. 323–336, SIGMETRICS '16).

Program Performance

High-level programmers are even further removed from memory system details.

```
#include <stdio.h>
main()
{
    printf("hello, world\n");
}
```

ops/operands
machine specs

mem model
(v.addr space)

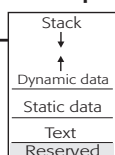
Programmer

*Some mem requests
satisfied faster*



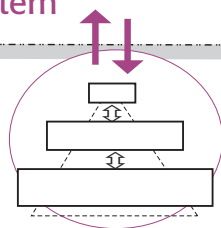
00100010101101110
01101110111001110
11101110010101111
01000001000110110

Object code



Access pattern

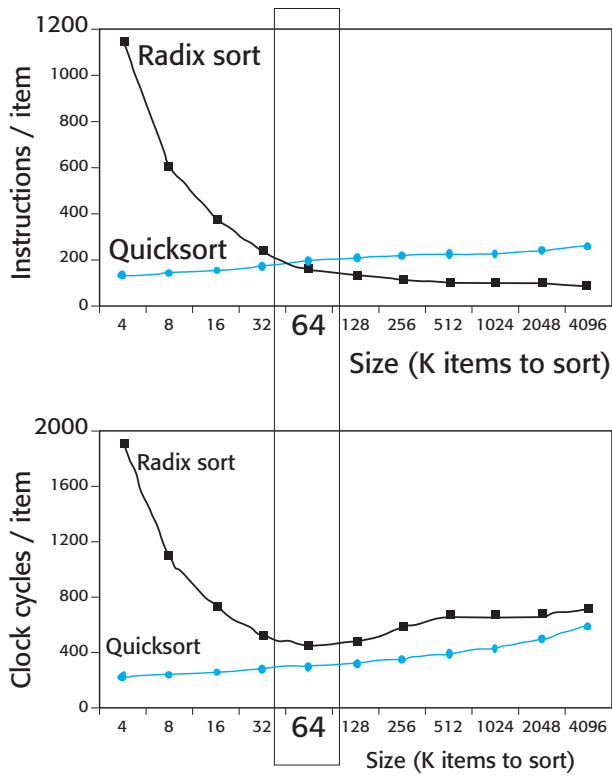
Machine



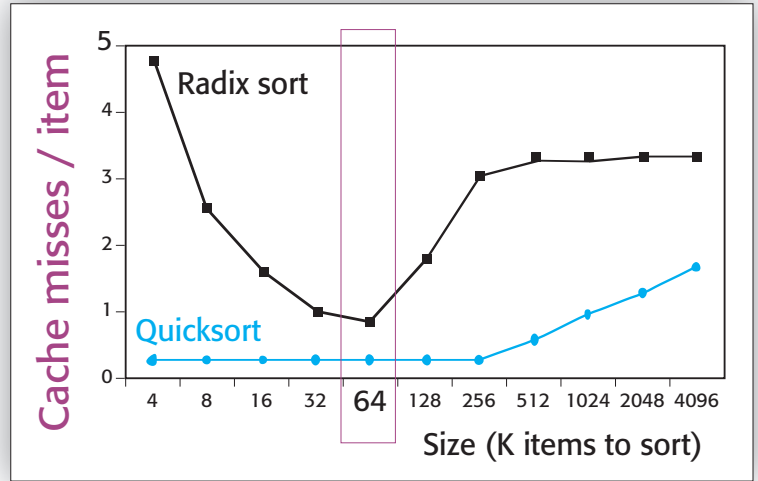
- ⇒ **Memory models hide a physical hierarchy**
- ⇒ **Machine code interacts with the hierarchy**
- ⇒ **Program performance will vary accordingly**

Some performance problems can only be resolved by examining the hierarchy and adjusting code or the algorithm

Program Performance Example



P&H (4th Revised), pp 489-490,
Understanding Program Performance



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.

Program Performance

3 Miss Types

Each of these misses can insert between tens to millions of cycles in latency to individual instructions.

Execution times of different programs increase depending on how much and which type of misses they generate.

In the *Core i5-661*, the slower offcore memory controller adds stall cycles to memory-intensive programs every time the page table is accessed or a cache miss is satisfied from DRAM.

⇒ **Cache miss**

Request not in close reach of CPU

⇒ **TLB miss**

A fast translation not available

⇒ **Page table miss (page fault)**

Address not in physical memory

Program Performance Constituents

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

☒ **Memory hierarchy**

Disk encryption programs run faster on the *i5-661* than previous generation processors due to 6 new instructions which support AES enc/decryption.

A carefully designed ISA can lead to cost-effective pipelined microarchitecture that economically delivers power-efficient high instruction execution rates.

☐ **Impact of ISA**

 Instruction selection

 Instruction design

☐ **Microarchitecture**

 Datapaths/caches + control streams

 Processor and instruction-level parallelism

☐ **Program**

 Algorithm selection

 Programming environment (lang/compiler)

☐ **Impact of I/O**

Program Performance Improvement





① ➡ **Algorithmic advantage**

Maximize computation efficiency

In addition to time, power savings may be achieved.

Maximize parallelizable tasks.

Sometimes it pays to restrict problem instances (focus on those of interest).

-  Less/cheap operations (reduce work)
-  Overlap operations (parallelize)
-  Limit/break scope of work (localize)
-  Clever shortcuts (heuristics)

② ➡ **Language/Compiler advantage**

Maximize code efficiency

-  Less code per operation
-  Better use of machine features





Program Performance Improvement

3

Although the advantages can be exploited independently, they do affect each other.

For example: to accommodate or utilize a specific memory arrangement, gaining the architectural advantage may require reworking the algorithm to alter memory locality.

⇒ Architectural advantage Maximize machine utilization

-  Better ILP
-  Add parallel processing resources
-  More addr block usage before replacement (focus expensive misses)
-  Reduce reliance on bandwidth limited interconnect channels

To Think About

Which performance constituent would be easier to control?