

# Efficiency: memory hierarchy; locality

---

COSC 208, Introduction to Computer Systems, 2021-10-25

## Outline

- Memory hierarchy
- Locality
- Optimizing for locality

No warm-up — Happy Monday!

## Memory hierarchy

- Compares various forms of storage in terms of
  - Access latency
  - Capacity
  - Cost
  - Volatility
- Access latency
  - Let's consider a 1hz CPU, which means 1 cycle = 1 second
  - Registers — 1 cycle = 1 second
  - Caches — ~10 cycles = ~10 seconds
  - Main memory — ~100 cycles = ~2 minutes
  - Solid-state drive — ~1 million cycles = ~11.5 days
  - Hard (i.e., traditional) disk drive — ~10 million cycles = ~115 days
  - Remote (i.e., network) storage — ~20ms = ~2 years
- Storage capacity
  - Let's assume 1 byte = 1mL
  - Registers —  $30 * 8B = \sim 250mL = \sim 1$  cup
  - Caches (Core i7 in MacBook Pro)
    - L1 —  $32KB + 32KB = 64L = \sim 1$  tank of gas
    - L2 —  $512KB * 4 \text{ cores} = 2048L = \sim 7$  bathtubs
  - Main memory = 32GB (in MacBook Pro) = ~13 olympic swimming pools
  - SSD = 1TB (in MacBook Pro) = ~Lake Moraine
- Cost
  - $2 \times 16GB \text{ DRAM} = \sim \$100 = \$3.12 \text{ per GB}$
  - $1TB \text{ SSD} = \$80 = \$0.08 \text{ per GB}$
  - $2TB \text{ HDD} = \$60 = \$0.03 \text{ per GB}$
- Volatility
  - Primary storage (registers, caches, and main memory) — volatile (i.e., data is lost if power is lost)
  - Secondary storage (SSD, HDD, network storage) — non-volatile (i.e., data is preserved if power is lost)

## Data movement

- Recall: *How does data move between the CPU, main memory, and secondary storage in the von Neumann Architecture?* — bus
- *Why does data move to/from secondary storage?* — data stored in primary storage is lost when a machine loses power
- *Why does data move between registers and main memory?* — not enough room in registers to store all values used by a program at runtime
- *How can we move less data?*
  - Make better use of registers — i.e., eliminate unnecessary loads/stores
  - Add additional memory to the CPU — i.e., a cache
- *How do we decide what/when to move data between the registers, cache, and CPU?* — based on locality

## Temporal vs. spatial locality

- *What is temporal locality?*
  - Access the same data repeatedly
  - E.g., for loop variable
- *What is spatial locality?*
  - Access data with a similar scope
  - E.g., next item in array
  - E.g., local variables/parameters, which are stored in the same stack frame
- Analogies for temporal and spatial locality
  - Book storage (Dive Into Systems Section 11.3.2)
    - Temporal locality — store most frequently used books at your desk, less frequently used books on your bookshelf, and least frequently used books at the library
    - Spatial locality — checkout books on the same/nearby subjects when you go to the library
  - Groceries (pre-class questions 3 & 4)
    - Temporal locality — you store food you eat frequently in the front of the refrigerator, while you store food you eat infrequently in the back of the refrigerator
    - Spatial locality — you organize the items on your grocery list based on the aisle in which they are located
  - Breakout groups: *Develop your own analogy for temporal and spatial locality*

## Optimizing assembly code for locality

- Q1: Cross-out redundant loads and stores from the assembly code

```
000000000000088c <interest_due>:
88c:  sub sp, sp, #0x20
890:  str w0, [sp, #12]      XXXXX
894:  str w1, [sp, #8]       XXXXX
898:  ldr w0, [sp, #12]      XXXXX
89c:  ldr w1, [sp, #8]       XXXXX
8a0:  mul w0, w1, w0
8a4:  str w0, [sp, #20]
8a8:  mov w0, #0x4b0
8ac:  str w0, [sp, #24]      XXXXX
8b0:  ldr w1, [sp, #20]
8b4:  ldr w0, [sp, #24]      XXXXX
8b8:  sdiv w0, w1, w0
8bc:  str w0, [sp, #28]      XXXXX
8c0:  ldr w0, [sp, #28]      XXXXX
8c4:  add sp, sp, #0x20
8c8:  ret

00000000000008cc <make_payment>:
8cc:  stp x29, x30, [sp, #-48]!
8d0:  mov x29, sp
8d4:  str w0, [sp, #28]
8d8:  str w1, [sp, #24]
8dc:  str w2, [sp, #20]
8e0:  ldr w1, [sp, #20]
8e4:  ldr w0, [sp, #28]      XXXXX
8e8:  bl 88c <interest_due>
8ec:  str w0, [sp, #40]      XXXXX
8f0:  ldr w1, [sp, #24]
8f4:  ldr w0, [sp, #40]      XXXXX
8f8:  sub w0, w1, w0
8fc:  str w0, [sp, #44]
900:  ldr w1, [sp, #44]
904:  ldr w0, [sp, #28]
908:  cmp w1, w0
90c:  b.le 918 <make_payment+0x4c>
910:  str wzr, [sp, #28]
914:  b 928 <make_payment+0x5c>
918:  ldr w1, [sp, #28]
91c:  ldr w0, [sp, #44]
920:  sub w0, w1, w0
924:  str w0, [sp, #28]
928:  ldr w0, [sp, #28]
92c:  ldp x29, x30, [sp], #48
930:  ret
```