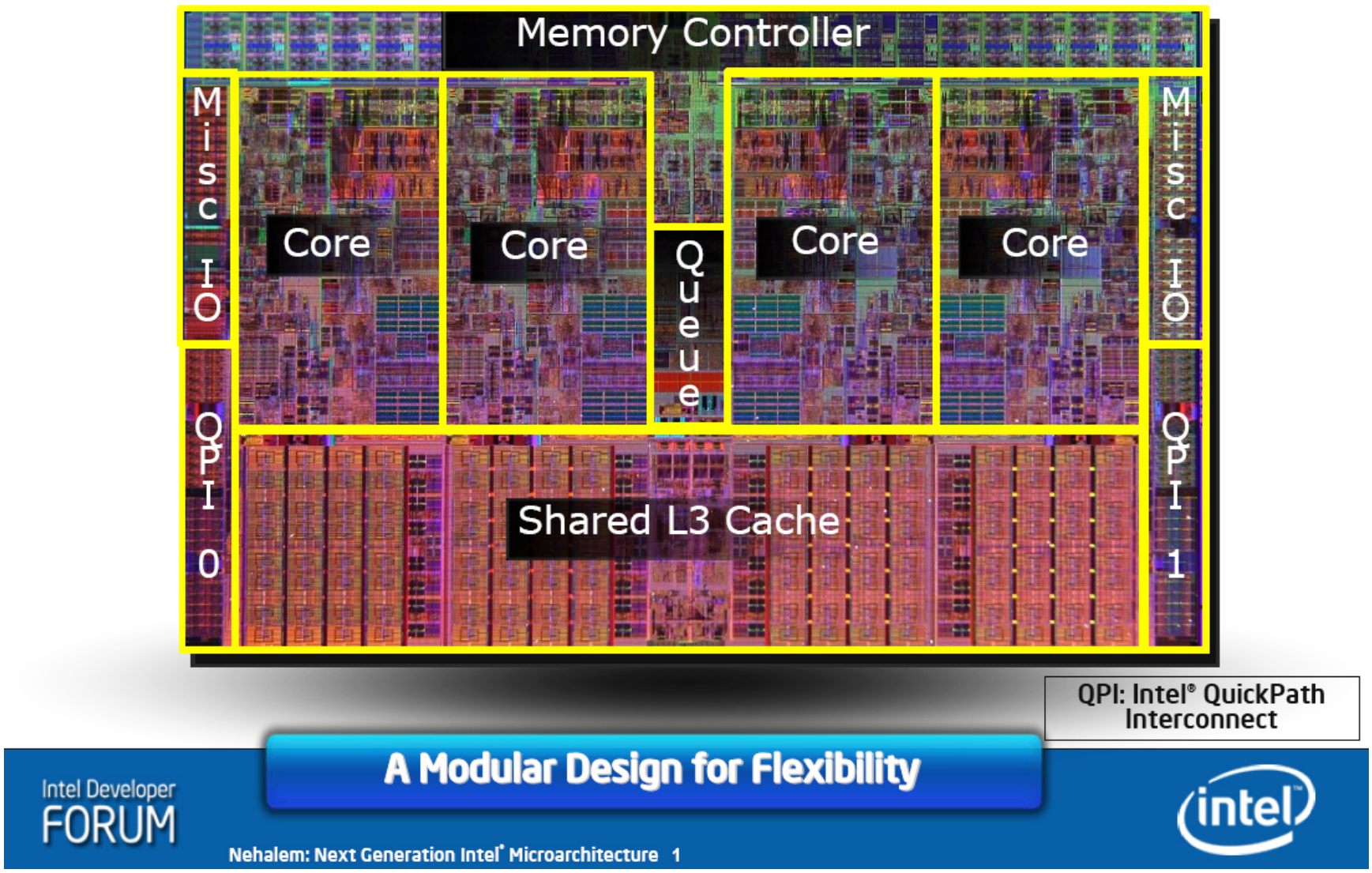
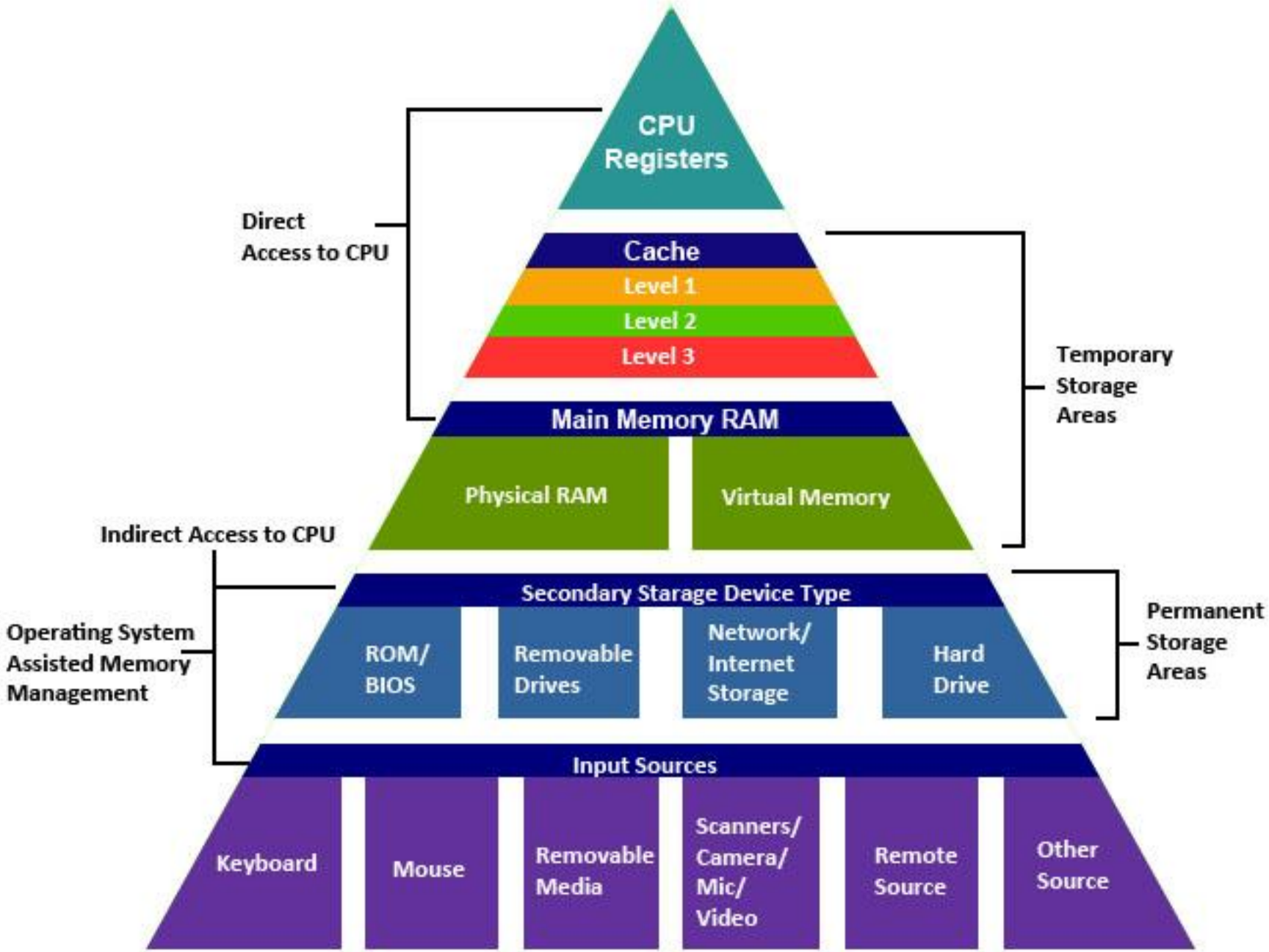
# 06_03_locality_and_memory hierarchy

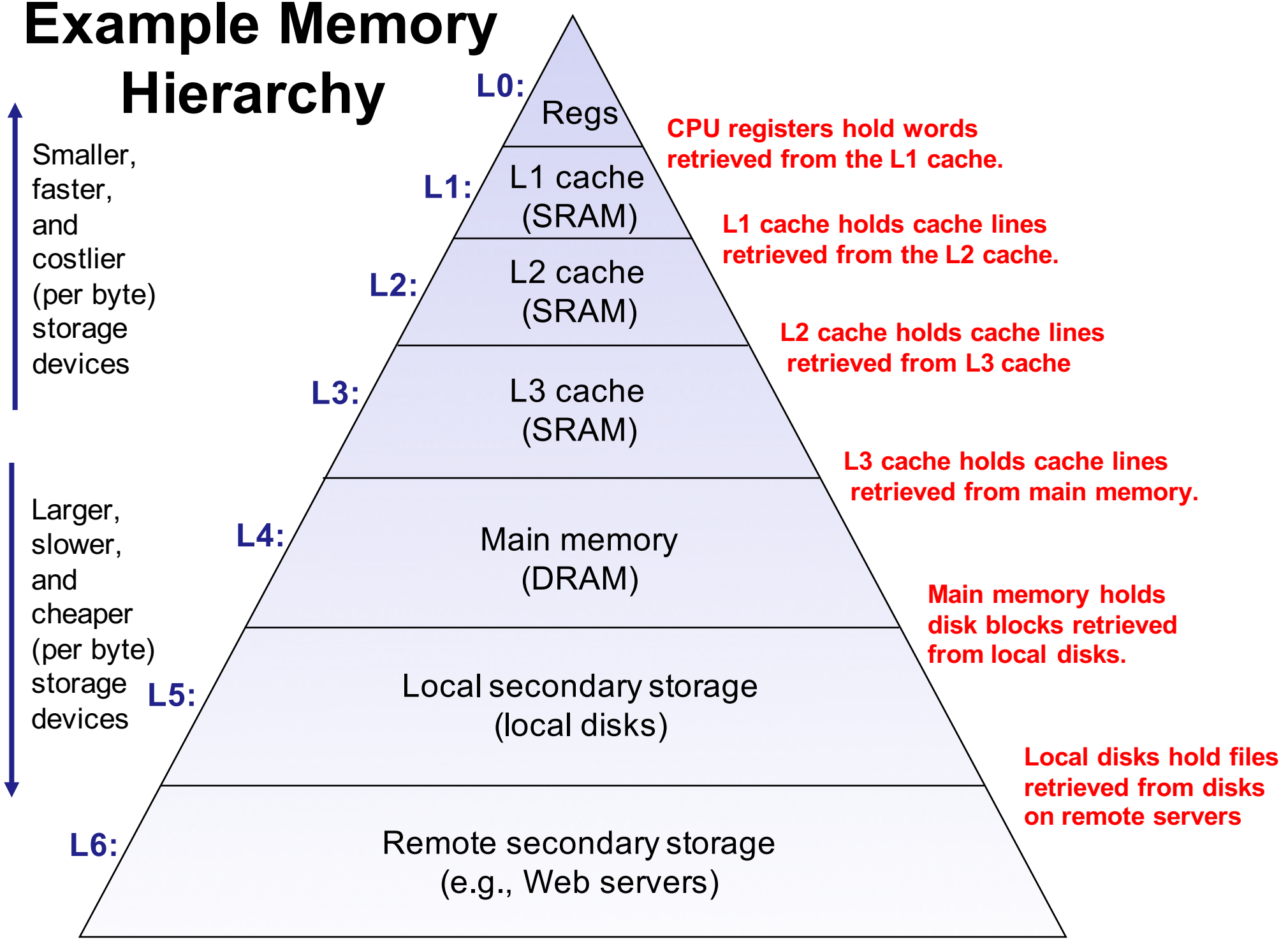
# The First Nehalem Processor



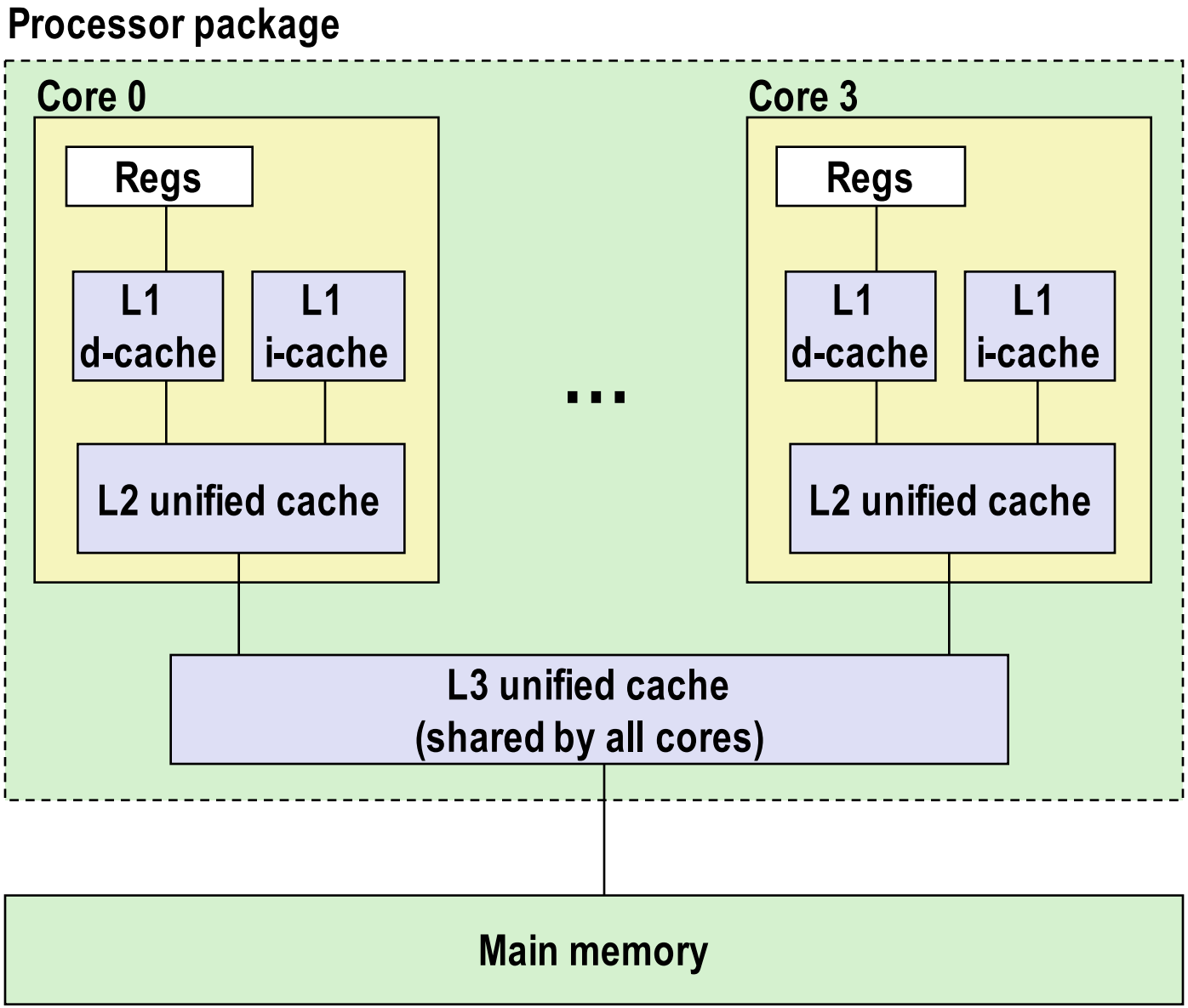

From [Intel (http://download.intel.com/pressroom/kits/corei7/images/Nehalem_Die_callout.jpg)](http://download.intel.com/pressroom/kits/corei7/images/Nehalem_Die_callout.jpg)


CPU Registers
Direct Access to CPU
Cache
Level 1
Level 2
Level 3
Main Memory RAM
Physical RAM
Virtual Memory
Temporary Storage Areas
Indirect Access to CPU
Secondary Starage Device Type
Operating System Assisted Memory Management
ROM/ BIOS
Removable Drives
Network/ Internet Storage
Hard Drive
Permanent Storage Areas
Input Sources
Keyboard
Mouse
Removable Media
Scanners/ Camera/ Mic/ Video
Remote Source
Other Source

# Example Memory Hierarchy



# Intel Core i7 Cache Hierarchy




**L1 i-cache and d-cache:**
32 KB, 8-way,
Access: 4 cycles

**L2 unified cache:**
256 KB, 8-way,
Access: 10 cycles

**L3 unified cache:**
8 MB, 16-way,
Access: 40-75 cycles

**Block size**: 64 bytes for all caches.

# Numbers Everyone Should Know

## Jeff Dean talk at Stanford

| | |
|---|---|
| L1 cache reference | 0.5 ns |
| Branch mispredict | 5 ns |
| L2 cache reference | 7 ns |
| Mutex lock/unlock | 100 ns |
| Main memory reference | 100 ns |
| Compress 1K bytes with Zippy | 10,000 ns |
| Send 2K bytes over 1 Gbps network | 20,000 ns |
| Read 1 MB sequentially from memory | 250,000 ns |
| Round trip within same datacenter | 500,000 ns |
| Disk seek | 10,000,000 ns |
| Read 1 MB sequentially from network | 10,000,000 ns |
| Read 1 MB sequentially from disk | 30,000,000 ns |
| Send packet CA $\rightarrow$ Netherlands$\rightarrow$CA | 150,000,000 ns |

# Fake Problem

| **location** | **access time** |
|---|---|
| L1 cache | 0.5 ns |
| L2 cache | 7 ns |
| RAM | 100 ns |
| hard drive | 10 ms |
| DVD | 140 ms |

# Pretend for a minute

| **location** | **access time** |
|---|---|
| L1 cache | 0.5 ns |
| L2 cache | 7 ns |
| RAM | 100 ns |
| hard drive | 10 ms |
| DVD | 140 ms |

| **location** | ***fake access time*** |
|---|---|
| L1 cache | 1 sec |
| L2 cache | |
| RAM | |
| hard drive | |
| DVD | |

If we keep the ratios the same as on the LHS, what are the remaining numbers on the RHS?

# Pretend for a minute

| **location** | **access time** |
|---|---|
| L1 cache | 0.5 ns |
| L2 cache | 7 ns |
| RAM | 100 ns |
| hard drive | 10 ms |
| DVD | 140 ms |

| **location** | ***fake access time*** |
|---|---|
| L1 cache | 1 sec |
| L2 cache | 14 sec |
| RAM | 200 sec or<br>3 min, 20 sec |
| hard drive | 20,000,000 sec<br>or $\approx$ 231.5 days |
| DVD | 280,000,000 sec or<br>$\approx$ 8.9 years |

# Pretend again.  Let's bake a cake.

if the counter is the CPU and the L1 cache is the cabinet two feet above …

| **location** | ***fake distance*** |
|---|---|
| L1 cache | 2 feet |
| L2 cache | 28 feet |
| RAM | 400 feet |
| hard drive | 40,000,000 feet or about 7,500 miles<br>about twice the distance from Philly<br>to Nome, AK |
| DVD | 560,000,000 feet or about 106,000 miles<br>or about the distance around the earth 4.25 times |

# The moral of the story

- If you're baking a cake and you have to walk to Nome, AK to get the flour:
  1. get what you need for the rest of the recipe
  2. pick up some sugar

# Locality

- Temporal locality
- Spatial locality

# locality

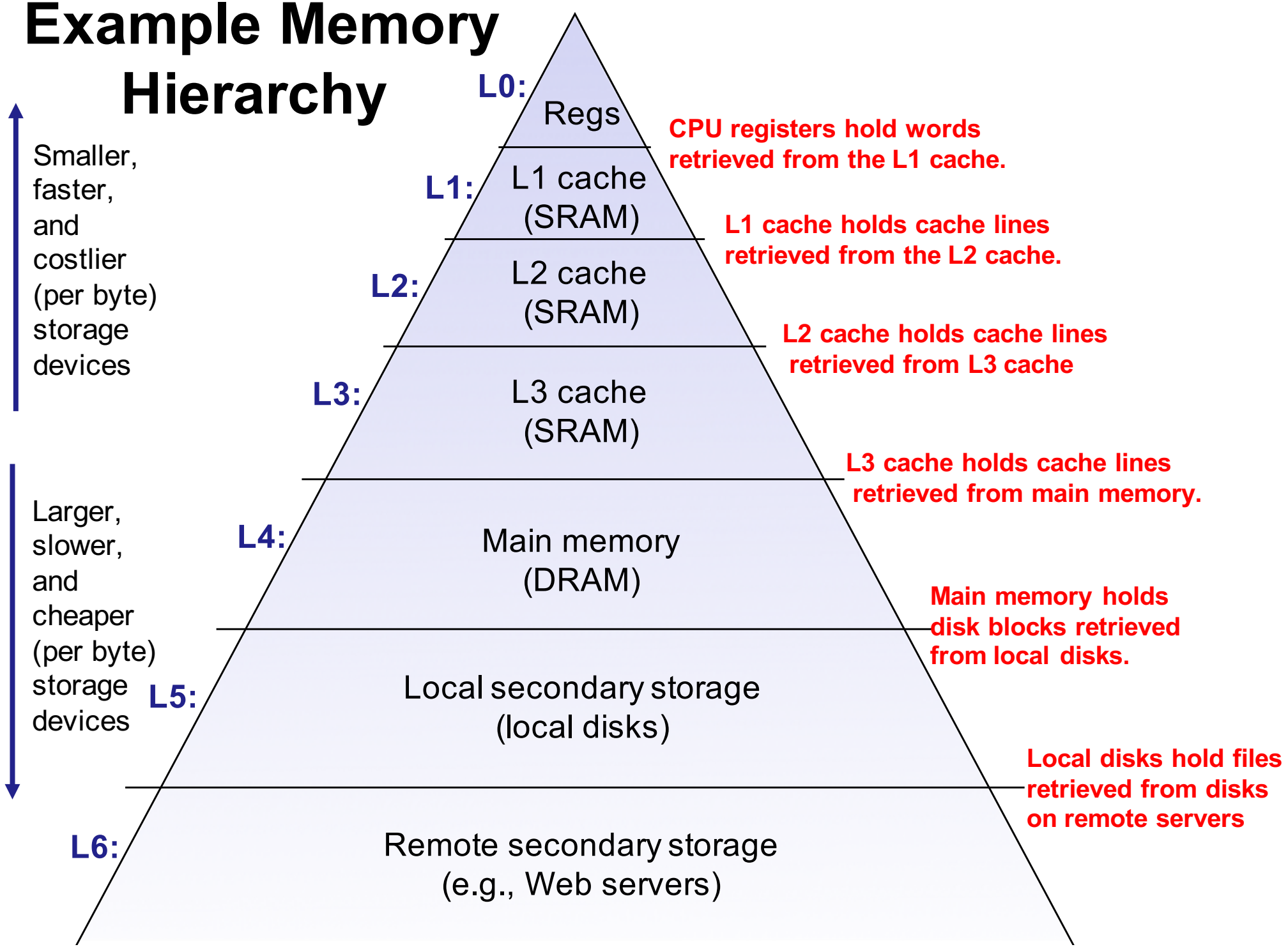
some code

```
for (i=0; i<SIZE; i++)
    sum+=A[i]
```

i, sum good temporal locality

array elements bad temporal, good spatial

# Example Memory Hierarchy



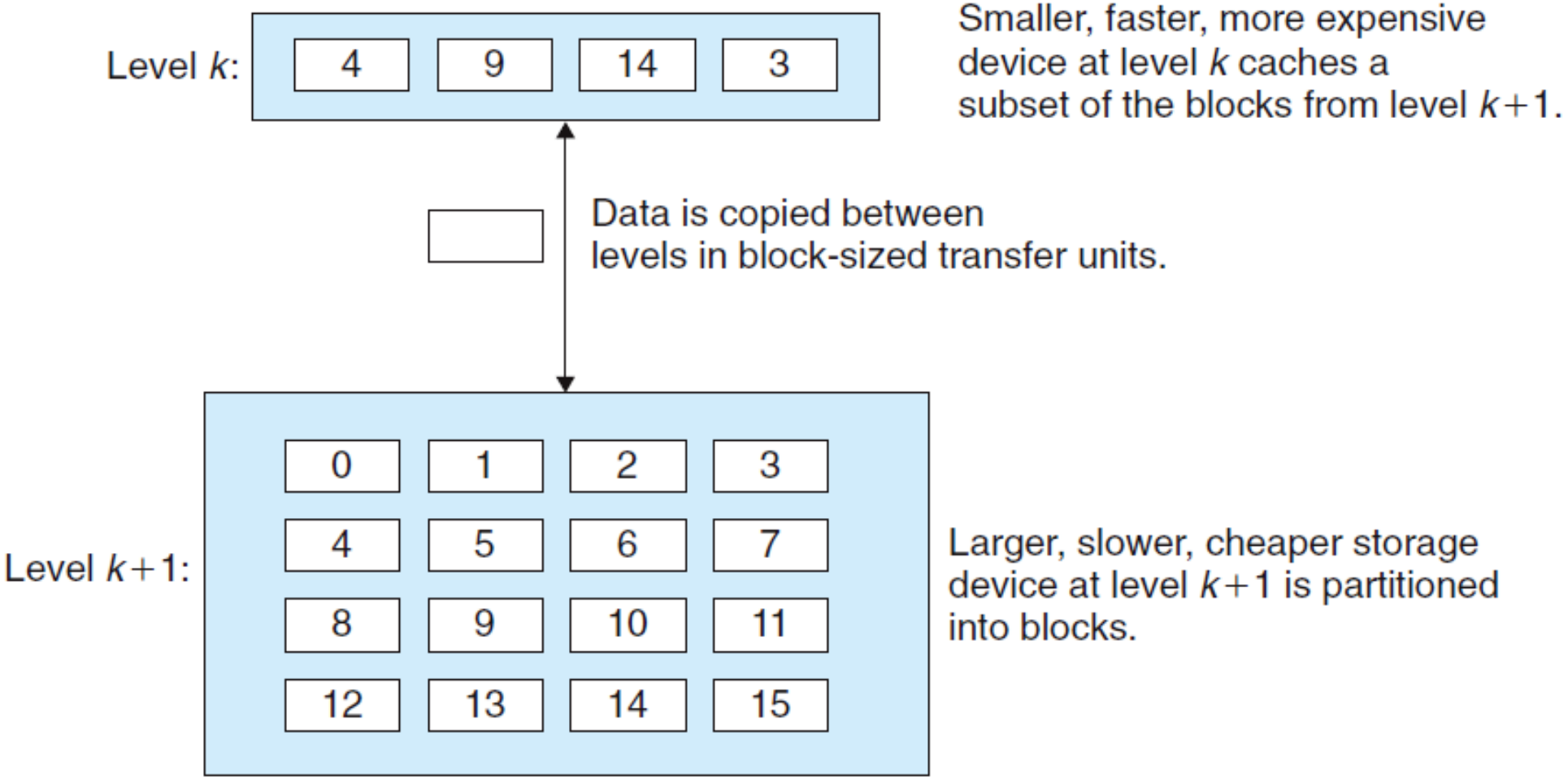
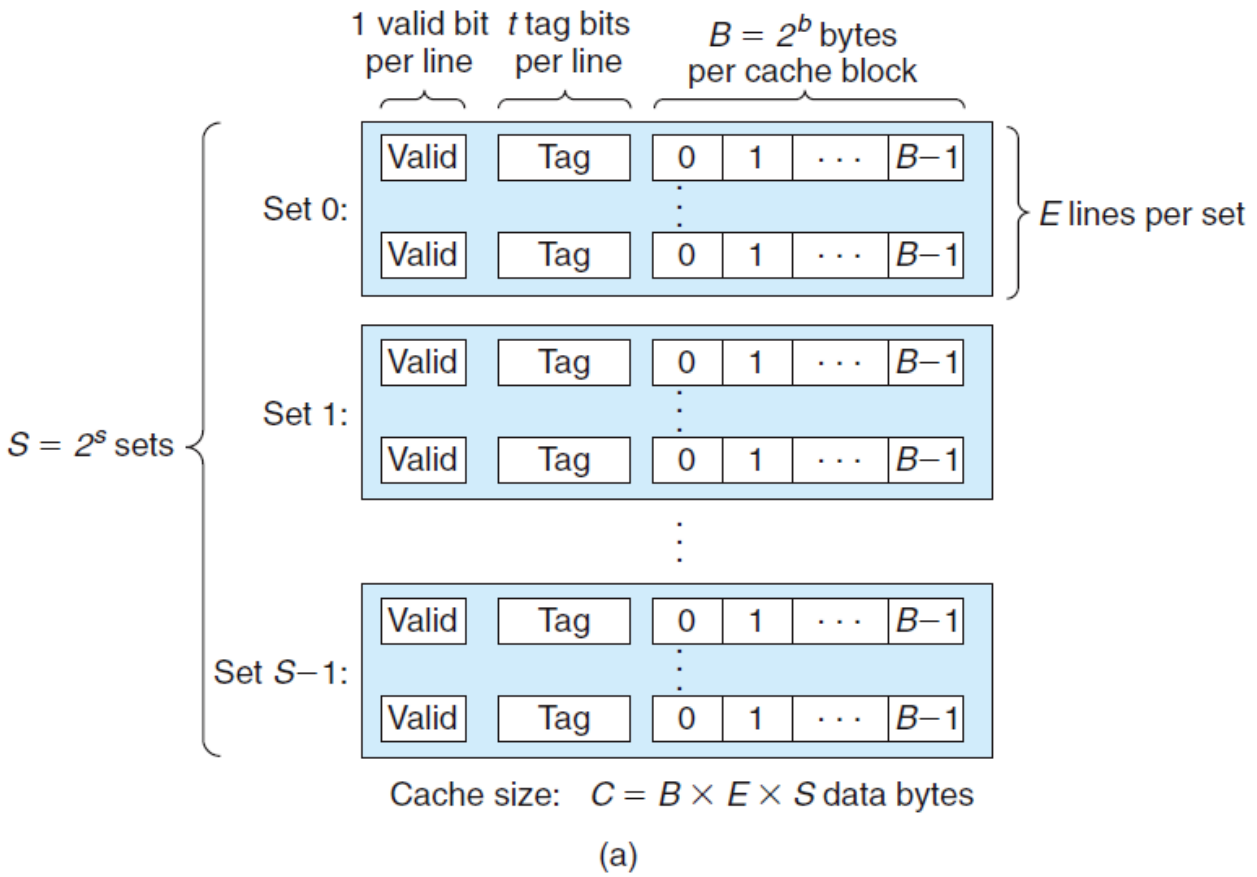



Figure 6.24 **The basic principle of caching in a memory hierarchy.**

| Type | What cached | Where cached | Latency (cycles) | Managed by |
|---|---|---|---|---|
| CPU registers | 4-byte or 8-byte word | On-chip CPU registers | 0 | Compiler |
| TLB | Address translations | On-chip TLB | 0 | Hardware MMU |
| L1 cache | 64-byte block | On-chip L1 cache | 1 | Hardware |
| L2 cache | 64-byte block | On/off-chip L2 cache | 10 | Hardware |
| L3 cache | 64-byte block | On/off-chip L3 cache | 30 | Hardware |
| Virtual memory | 4-KB page | Main memory | 100 | Hardware + OS |
| Buffer cache | Parts of files | Main memory | 100 | OS |
| Disk cache | Disk sectors | Disk controller | 100,000 | Controller firmware |
| Network cache | Parts of files | Local disk | 10,000,000 | AFS/NFS client |
| Browser cache | Web pages | Local disk | 10,000,000 | Web browser |
| Web cache | Web pages | Remote server disks | 1,000,000,000 | Web proxy server |

Figure 6.25 **The ubiquity of caching in modern computer systems.** Acronyms: TLB: translation lookaside buffer, MMU: memory management unit, OS: operating system, AFS: Andrew File System, NFS: Network File System.

Figure 6.27
**General organization of cache** $(S, E, B, m)$. (a) A cache is an array of sets. Each set contains one or more lines. Each line contains a valid bit, some tag bits, and a block of data. (b) The cache organization induces a partition of the $m$ address bits into $t$ tag bits, $s$ set index bits, and $b$ block offset bits.



| Fundamental parameters | |
|---|---|
| Parameter | Description |
| $S = 2^s$ | Number of sets |
| $E$ | Number of lines per set |
| $B = 2^b$ | Block size (bytes) |
| $m = \log_2(M)$ | Number of physical (main memory) address bits |

| Derived quantities | |
|---|---|
| Parameter | Description |
| $M = 2^m$ | Maximum number of unique memory addresses |
| $s = \log_2(S)$ | Number of *set index bits* |
| $b = \log_2(B)$ | Number of *block offset bits* |
| $t = m - (s + b)$ | Number of *tag bits* |
| $C = B \times E \times S$ | Cache size (bytes) not including overhead such as the valid and tag bits |

Figure 6.28 **Summary of cache parameters.**

## Practice Problem 6.10

The following table gives the parameters for a number of different caches. For each cache, determine the number of cache sets ($S$), tag bits ($t$), set index bits ($s$), and block offset bits ($b$).

| Cache | $m$ | $C$ | $B$ | $E$ | $S$ | $t$ | $s$ | $b$ |
|---|---|---|---|---|---|---|---|---|
| 1. | 32 | 1024 | 4 | 1 | | | | |
| 2. | 32 | 1024 | 8 | 4 | | | | |
| 3. | 32 | 1024 | 32 | 32 | | | | |

## Practice Problem 6.10

The following table gives the parameters for a number of different caches. For each cache, determine the number of cache sets ($S$), tag bits ($t$), set index bits ($s$), and block offset bits ($b$).

| Cache | $m$ | $C$ | $B$ | $E$ | $S$ | $t$ | $s$ | $b$ |
|---|---|---|---|---|---|---|---|---|
| 1. | 32 | 1024 | 4 | 1 | | | | |
| 2. | 32 | 1024 | 8 | 4 | | | | |
| 3. | 32 | 1024 | 32 | 32 | | | | |

The solution is a straightforward application of the definitions of the various cache parameters in Figure 6.28. Not very exciting, but you need to understand how the cache organization induces these partitions in the address bits before you can really understand how caches work.

| Cache | $m$ | $C$ | $B$ | $E$ | $S$ | $t$ | $s$ | $b$ |
|---|---|---|---|---|---|---|---|---|
| 1. | 32 | 1024 | 4 | 1 | 256 | 22 | 8 | 2 |
| 2. | 32 | 1024 | 8 | 4 | 32 | 24 | 5 | 3 |
| 3. | 32 | 1024 | 32 | 32 | 1 | 27 | 0 | 5 |