CS429: Computer Organization and Architecture

Cache I

Dr. Bill Young
Department of Computer Sciences
University of Texas at Austin

Last updated: January 13, 2017 at 08:55

Locality

Principle of Locality:

- Programs tend to reuse data and instructions near those used recently, or that were recently referenced.
- *Temporal locality:* Recently referenced items are likely to be referenced in the near future.
- *Spatial locality:* Items with nearby addresses tend to be referenced close together in time.

```
sum = 0;
for ( i = 0; i < n; i++ )
    sum += a[i];
return sum;
```

Data:

- Reference array elements in succession (stride-1): Spatial
- Reference sum each iteration: Temporal

Instructions:

- Reference instructions in sequence: Spatial
- Cycle through loop repeatedly: Temporal

Locality Example

Claim: Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer.

```
int sumarrayrows1( int a[M][N] )
{
    int i, j, sum = 0;

    for ( i = 0; i < M; i++ )
        for ( j = 0; j < N; j++ )
            sum += a[i][j];
    return sum;
}
```

Question: Does this function have good locality?

Locality Example 2

```
int sumarrayrows2( int a[M][N] )
{
    int i, j, sum = 0;

    for ( j = 0; j < N; j++ )
        for ( i = 0; i < M; i++ )
            sum += a[i][j];
    return sum;
}
```

Does this compute the same function as sumarrayrows1?

Question: Does *this* function have good locality? How does it compare to the previous version?

Locality Example 3

A *stride-1 reference pattern* means that successive references are 1 “unit” apart. (Here unit means the size of the data type.)

Can you permute the loops so that this function scans the 3-d array a with a stride-1 reference pattern (and thus has good spatial locality)?

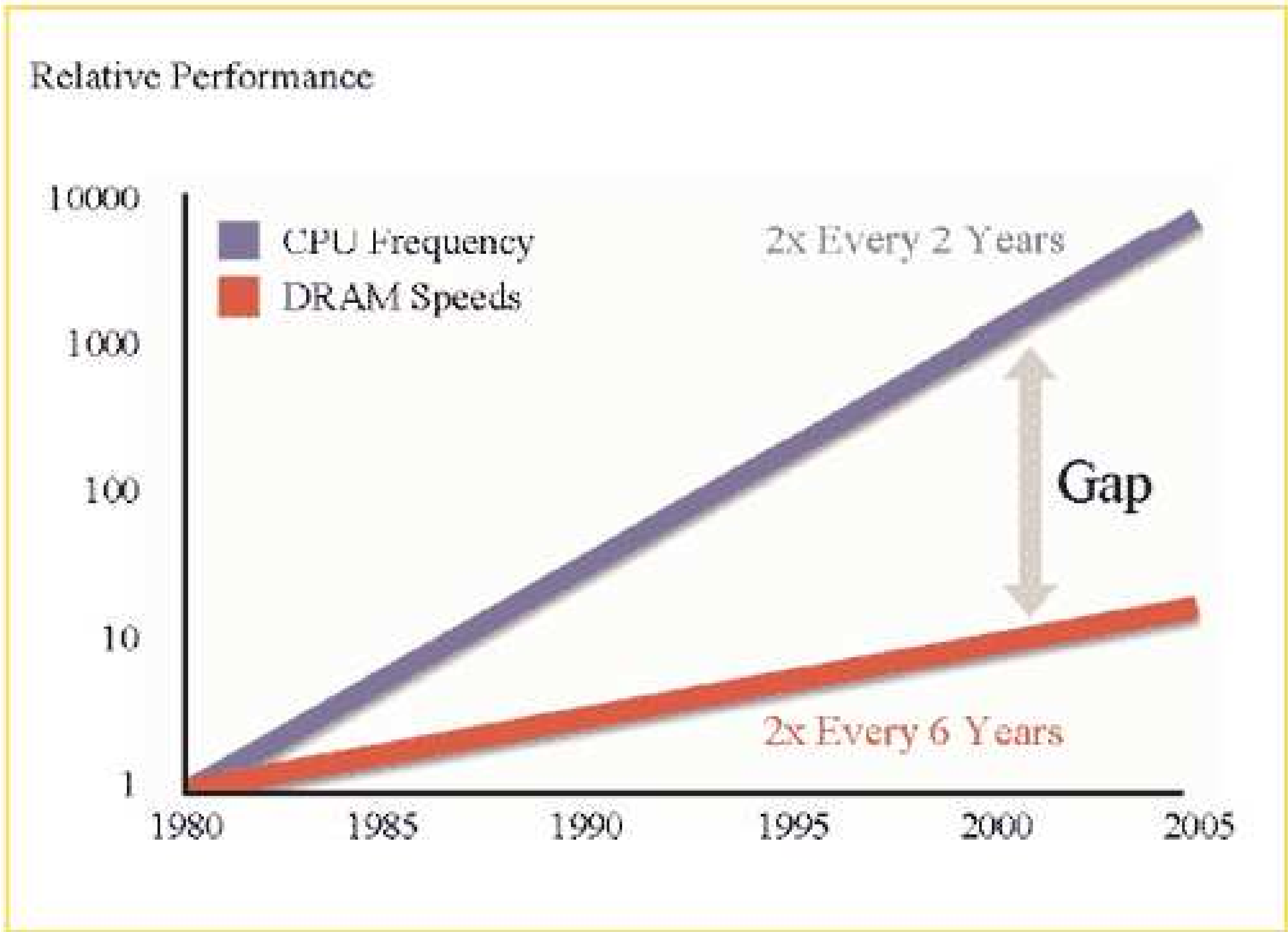
```
int sumarray3d( int a[N][N][N] )
{
    int i, j, k, sum = 0;

    for ( i = 0; i < N; i++ )
        for ( j = 0; j < N; j++ )
            for ( k = 0; k < N; k++ )
                sum += a[k][i][j];
    return sum;
}
```

Remember the CPU-Memory Gap

CPU speed increases *faster* than memory speed, meaning that:

- memory is more and more a limiting factor on performance;
- increased importance for caching and similar techniques.



Memory Hierarchies

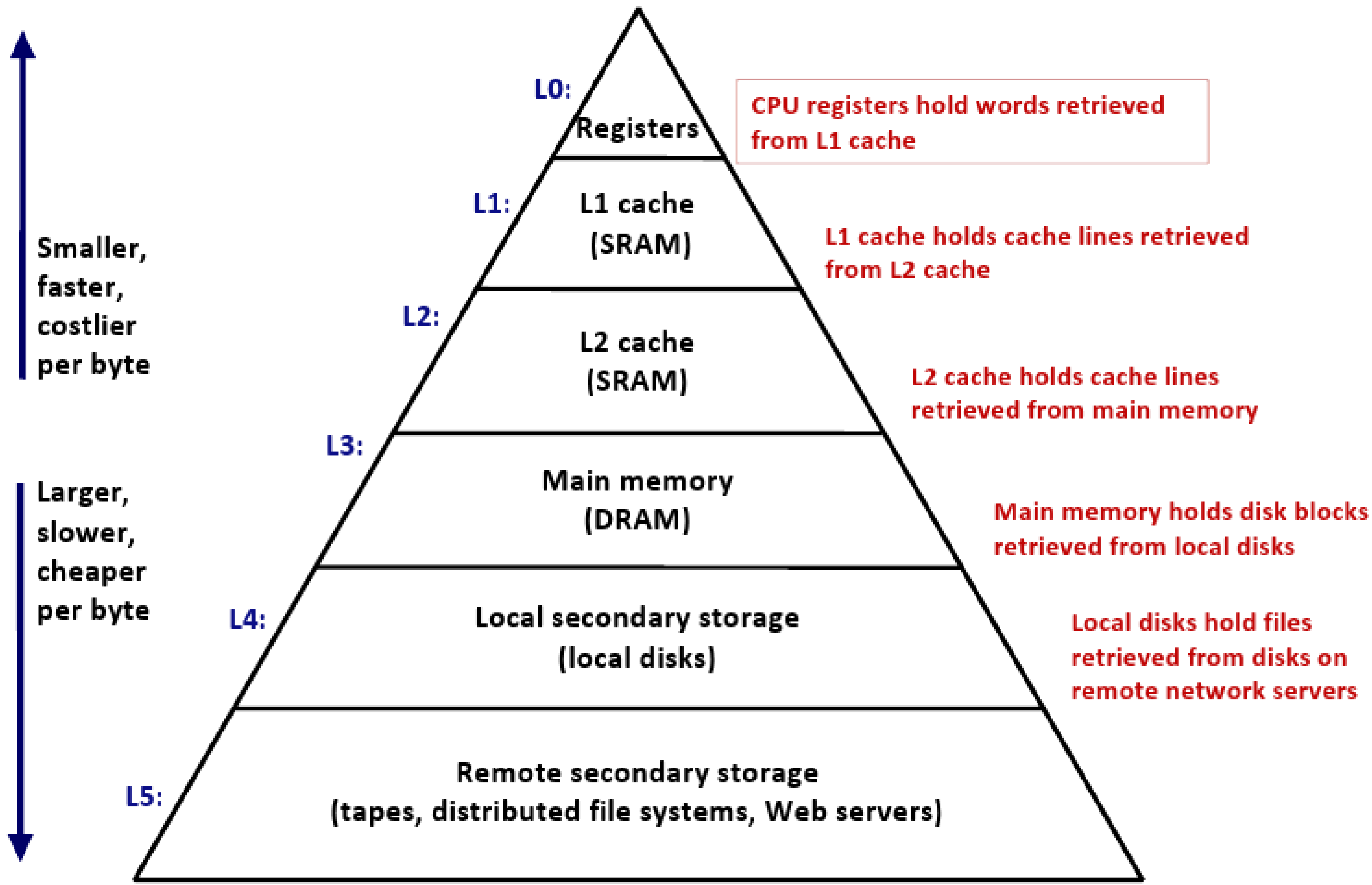
Some fundamental and enduring properties of hardware and software:

- Fast storage technologies typically cost more per byte and have less capacity than slower ones.
- The gap between CPU and main memory speed is widening.
- Well-written programs tend to exhibit good locality.
- Memory systems access "blocks" of data, not individual bytes.

These fundamental properties complement each other beautifully.

They suggest an approach for organizing memory and storage systems known as a *memory hierarchy.*

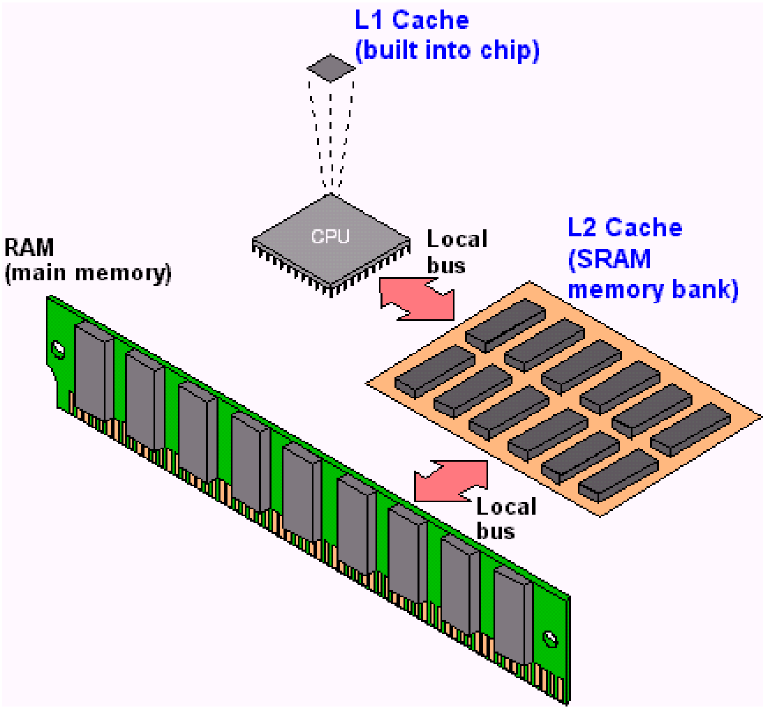
Example Memory Hierarchy



Caches

Cache: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.

The fundamental idea of a memory hierarchy: For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.



Examples of Caching in the Hierarchy

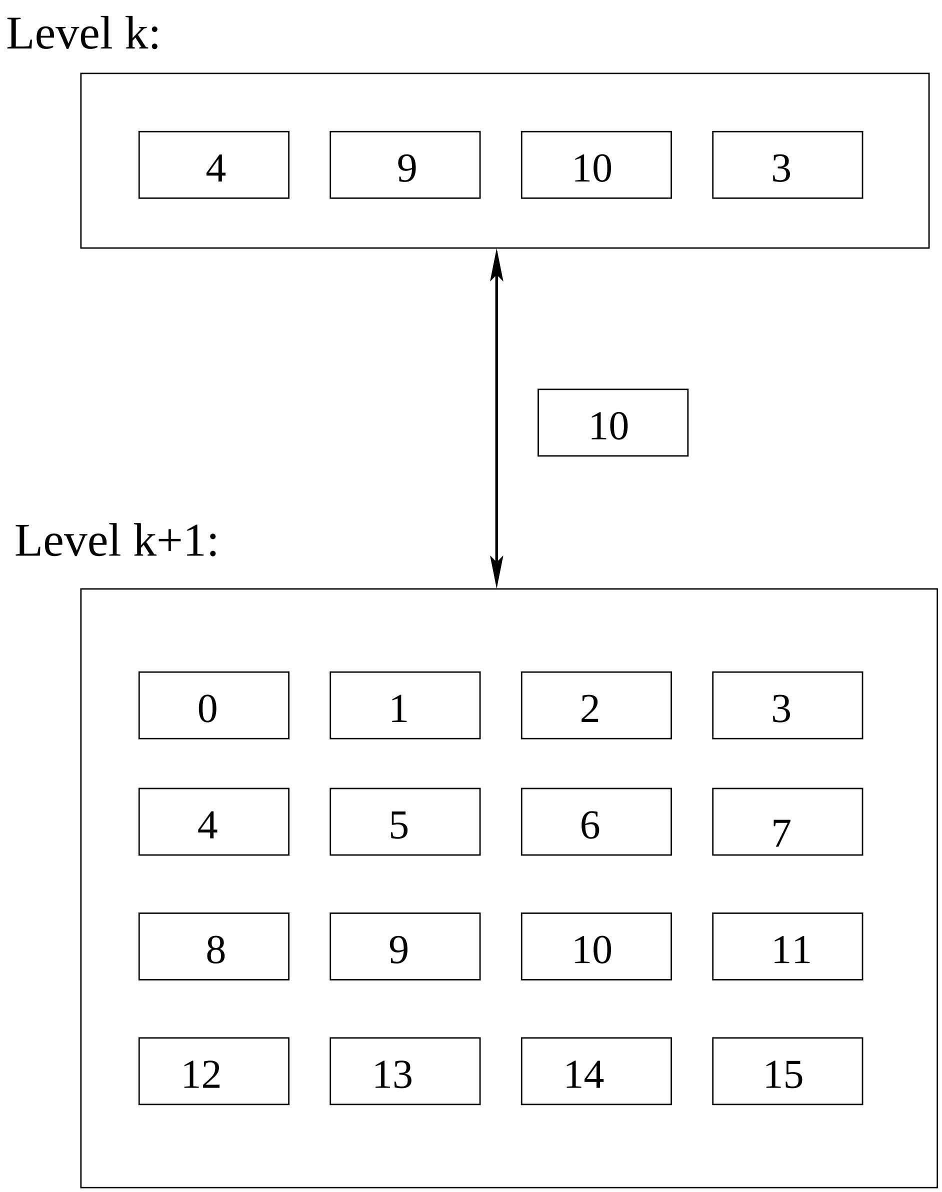
Cache type	What	Where	Latency (cycles)	Managed by
Registers	8-byte word	CPU registers	0	compiler
TLB	address translations	On-chip TLB	0	hardware
L1 cache	32-byte block	On-chip L1	1	hardware
L2 cache	32-byte block	On-chip L2	10	hardware
Virtual Memory	4KB page	main memory	100	hw + OS
Buffer cache	parts of files	main memory	100	OS
Network buffer cache	parts of files	local disk	10M	AFS/NFS client
Browser cache	web pages	local disk	10M	web browser
Web cache	web pages	remote server disks	1000M	web proxy server

Why Memory Hierarchies?

Why do memory hierarchies work?

- Programs tend to access the data at level k more often than they access the data at level k+1.
- Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit.
- *Net effect:* A large pool of memory that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.
- We use a combination of small fast memory and big slow memory to give the illusion of big fast memory.

Caching in a Memory Hierarchy

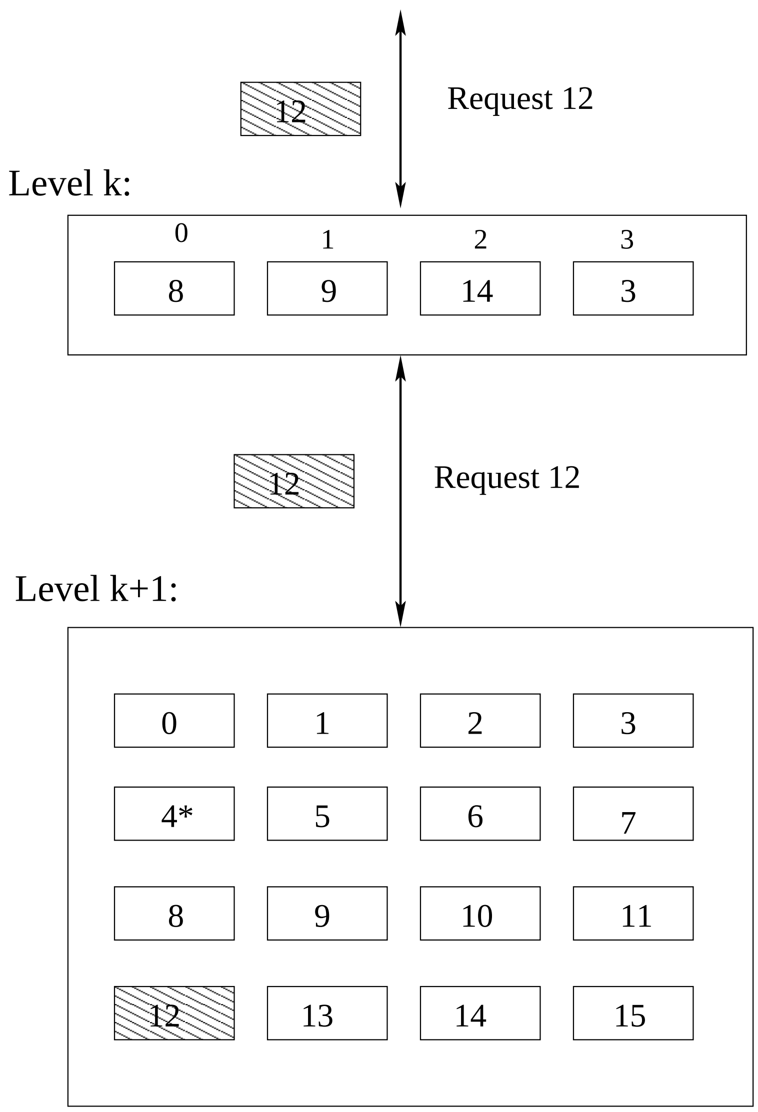


Smaller, faster, more expensive device at level k caches a subset of the blocks from level k+1.

Data is copied between levels in block-sized transfer units.

Larger, slower, cheaper storage device at level k+1 is partitioned into blocks.

General Caching Concepts



Program needs object d, stored in some block b.

Cache hit: program finds b in the level k cache, e.g., block 14.

Cache miss: b is not at level k, so must fetch it from level k+1, e.g., block 12.

- If level k cache is full, then some current block (the victim) must be replaced (evicted).
- *Placement policy:* where can the new block go? E.g., b mod 4.
- *Replacement policy:* Which block should be evicted? E.g., LRU.

General Caching Concepts

Types of cache misses:

Cold (compulsary) miss: the cache is empty.

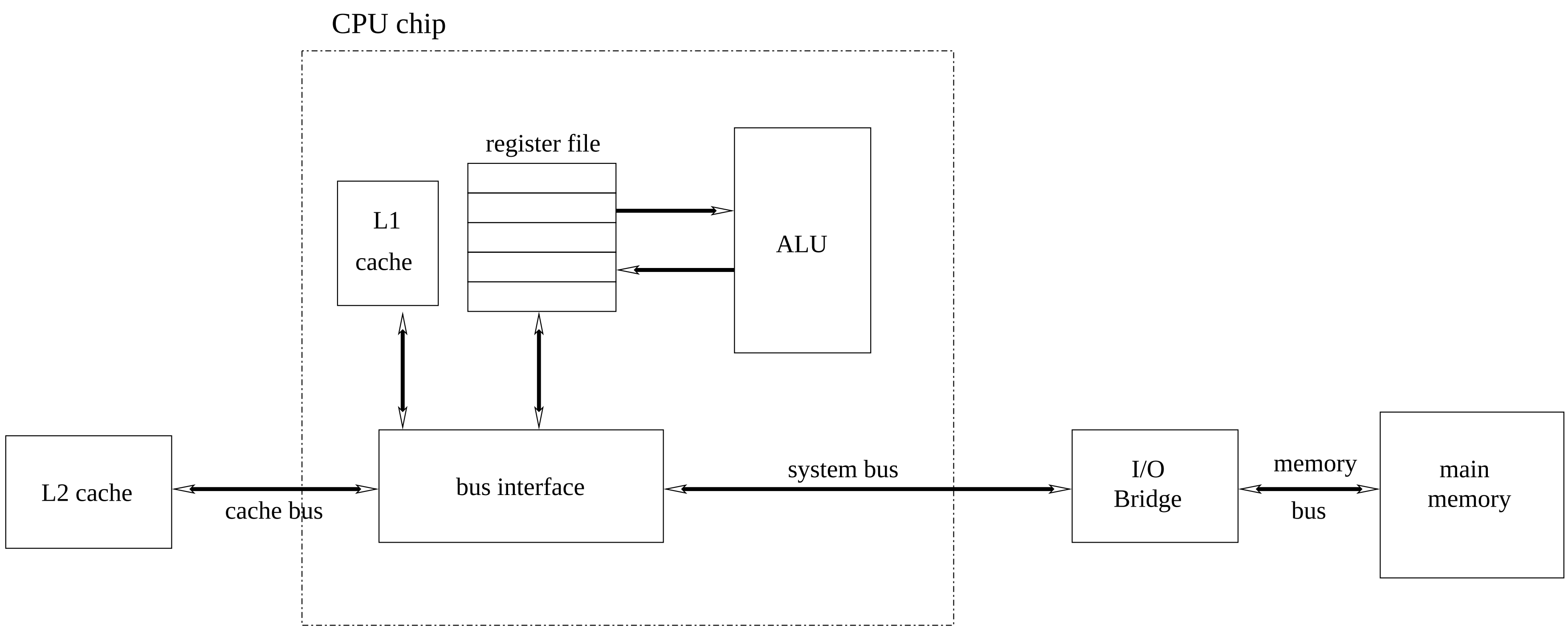
Conflict miss: all available positions at level k are occupied.

- Most caches limit blocks at level k+1 to a small subset (sometimes only one) of the block positions at level k.
- E.g., Block i at level k+1 must be placed in block (i mod 4) at level k.
- Conflict misses occur when multiple data objects all map to the same level k block. Note: there still may be empty slots in the cache.
- E.g., Referencing blocks $0, 8, 0, 8, 0, 8, \ldots$ would miss every time.

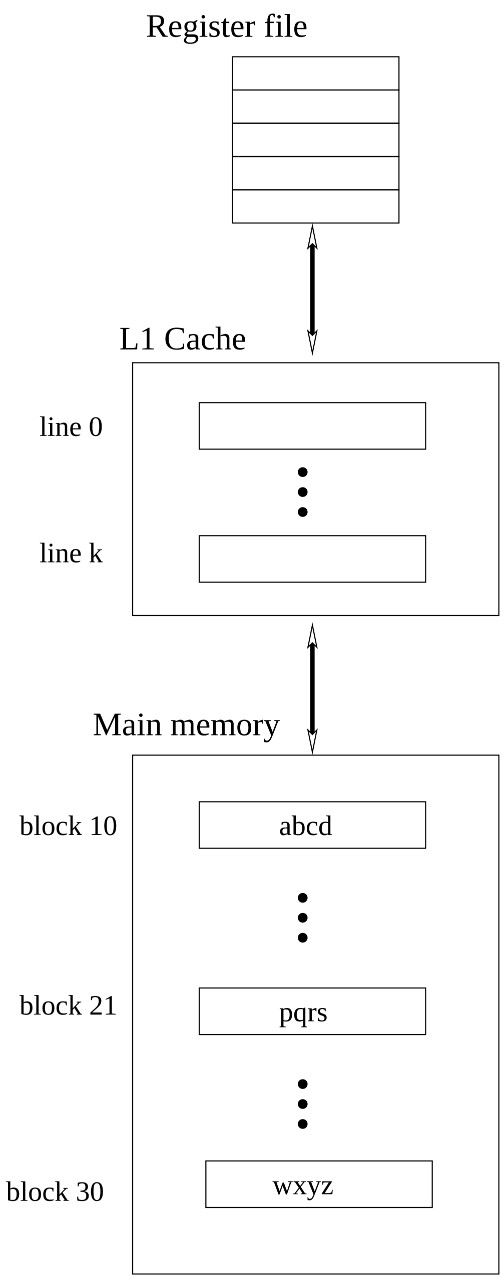
Capacity miss: the set of active cache blocks (working set) is larger than the cache.

Cache Memories

- L1 and L2 cache memories are small, fast SRAM-based memories managed automatically in hardware. They hold frequently accessed blocks of main memory.
- CPU looks first for data in L1, then in L2, then in main memory.
- The typical bus structure is shown below.



Inserting an L1 Cache



The tiny, very fast CPU register file has room for a small number of 8-byte words.

The transfer unit between register file and cache is an 8-byte block.

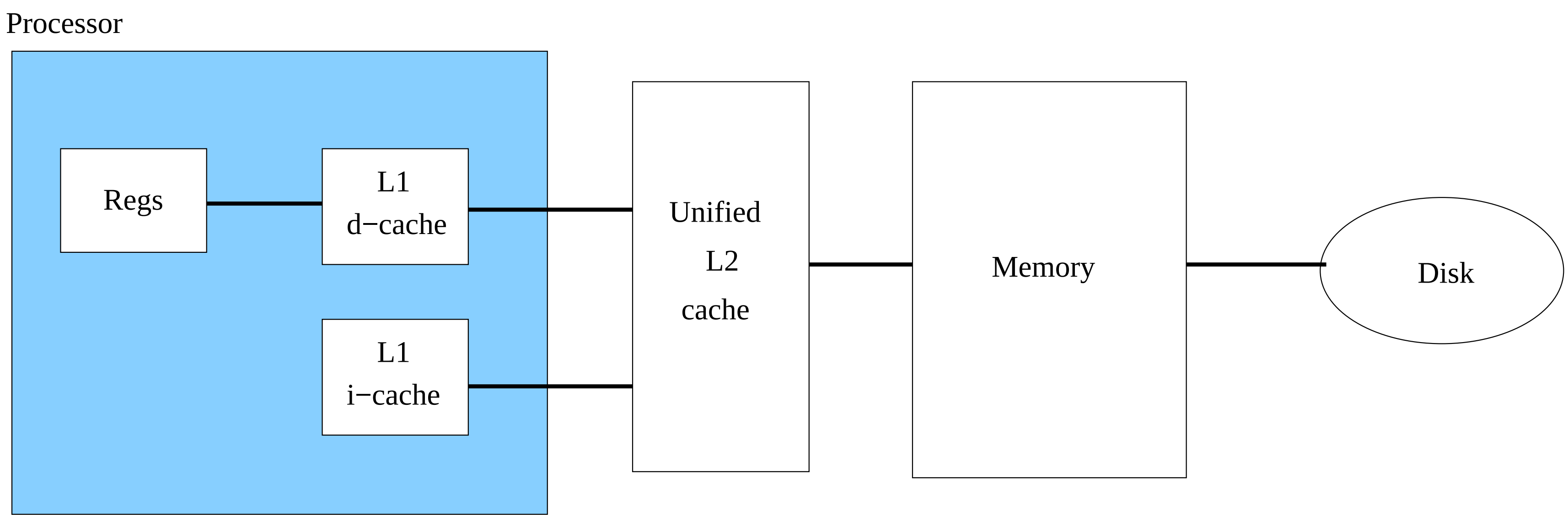
The small, fast L1 cache has room for k lines (each containing several words).

The transfer unit between cache and main memory is a block of bytes.

The big, slow main memory has room for many blocks.

Multi-Level Caches

Options: separate *data* and *instruction caches*, or a *unified cache*.



	registers	L1	L2	memory	disk
size	200B	8-64KB	1-4MB SRAM	128MB DRAM	30GB
speed	3ns	3ns	6ns	60ns	8ms
\$/MB			\$100	\$1.50	\$0.05
line size	8B	32B	32B	8KB	

Summary

This slideset:

- Locality: Spatial and Temporal
- Cache principles
- Multi-level cache hierarchies

Next time:

- Cache organization
- Replacement and writes
- Programming considerations