CS 270: Computer Organization

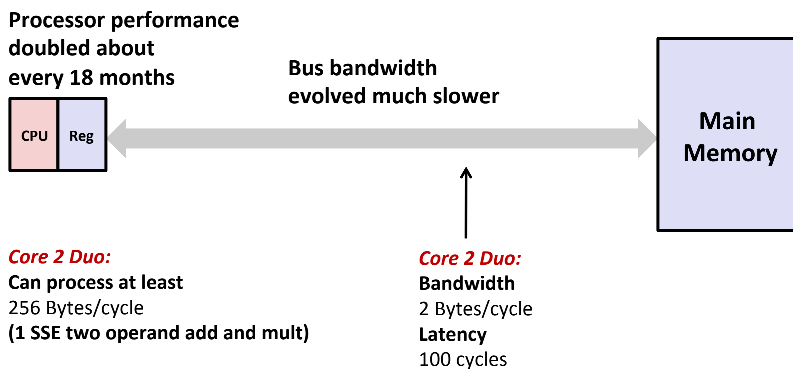
The Memory Hierarchy

Instructor:

Professor Stephen P. Carl

Sewanee: The University of the South

Problem: Processor-Memory Bottleneck

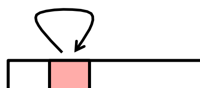


Solution: Caches

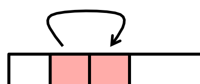
©R.A. Rutenbar, 2001

Why caches work: Locality

- **Principle of Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently**
- **Temporal locality:**
  - Recently referenced items are likely to be referenced again in the near future



- **Spatial locality:**
  - Items with nearby addresses tend to be referenced close together in time



Sewanee: The University of the South

Locality Example

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

- **Data references**
  - Reference array elements in succession (stride-1 reference pattern). — **Spatial locality**
  - Reference variable `sum` each iteration. — **Temporal locality**
- **Instruction references**
  - Reference instructions in sequence. — **Spatial locality**
  - Cycle through loop repeatedly. — **Temporal locality**

Qualitative Estimates of Locality

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

- **Question:** Does this function have good locality with respect to array `a`?

```
int sum_array_rows(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;
}
```

Sewanee: The University of the South

Locality Example

Question: Does this function have good locality with respect to array `a`?

```
int sum_array_cols(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;
}
```

©R.A. Rutenbar, 2001

Locality Example

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

```
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

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

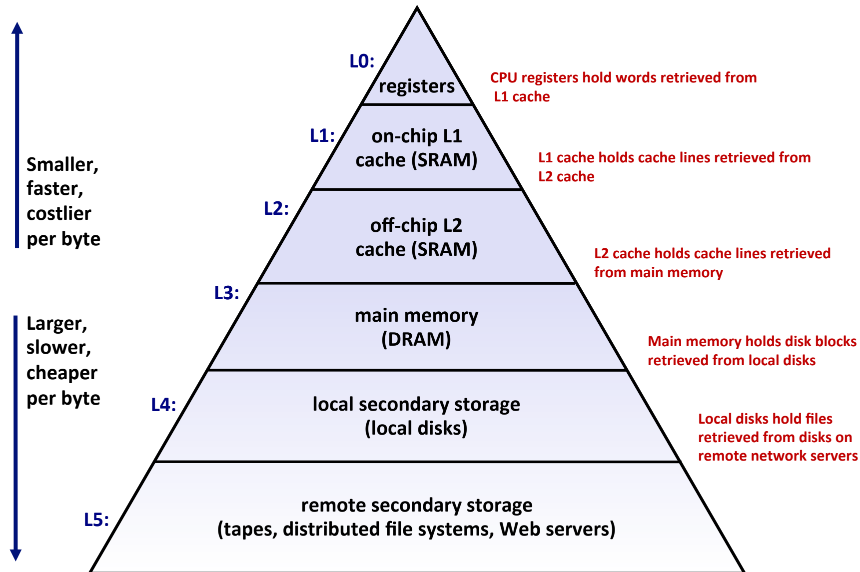
Sewanee: The University of the South

Memory Hierarchies

- **Some fundamental and enduring properties of hardware and software:**
  - Fast storage technologies cost more per byte, have less capacity, and require more power (heat!).
  - The gap between CPU and main memory speed is widening.
  - Well-written programs tend to exhibit good locality.
- **These fundamental properties complement each other beautifully.**
- **They suggest an approach for organizing memory and storage systems known as a memory hierarchy.**

©R.A. Rutenbar, 2001

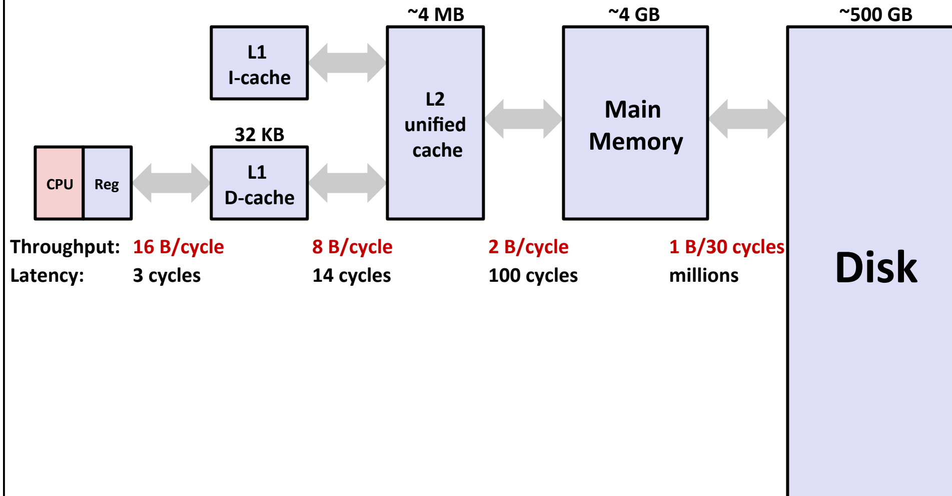
An Example Memory Hierarchy

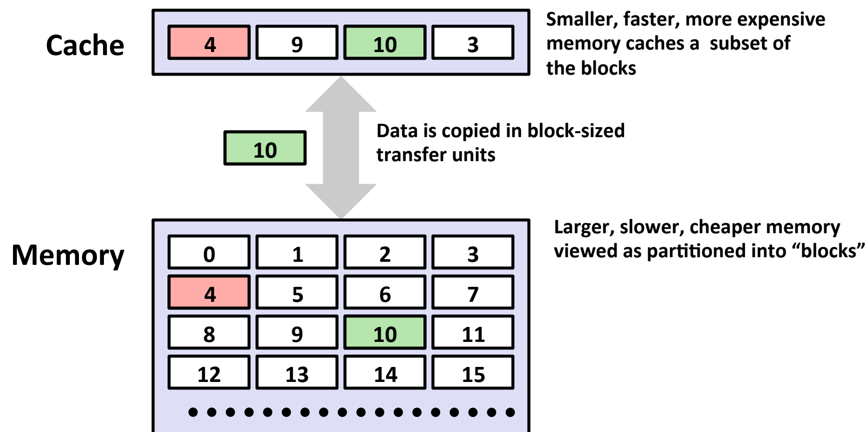


Caches

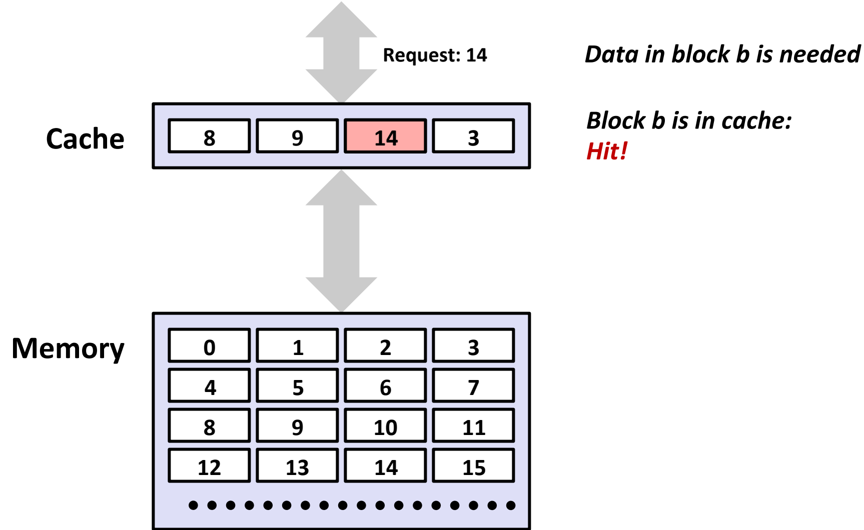
- ***Cache:*** **A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.**
- **Fundamental idea of a memory hierarchy:**
  - For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.
- **Why do memory hierarchies work?**
  - Because of locality, 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.
- ***Big Idea:*** **The memory hierarchy creates a large pool of storage 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.**

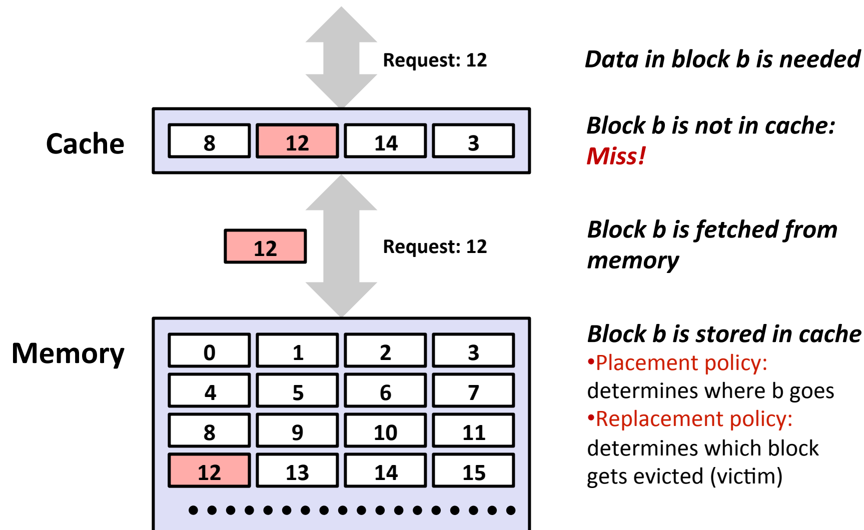
©R.A. Rutenbar, 2001





©R.A. Rutenbar, 2001





©R.A. Rutenbar, 2001

Cache Performance Metrics

- **Miss Rate**
  - Fraction of memory references not found in cache (misses / accesses) = 1 – hit rate
  - Typical numbers (in percentages):
    - 3-10% for L1
    - can be quite small (e.g., < 1%) for L2, depending on size, etc.
- **Hit Time**
  - Time to deliver a line in the cache to the processor
    - includes time to determine whether the line is in the cache
  - Typical numbers:
    - 1-2 clock cycle for L1
    - 5-20 clock cycles for L2
- **Miss Penalty**
  - Additional time required because of a miss
    - typically 50-200 cycles for main memory (Trend: increasing!)

Sewanee: The University of the South

Lets think about those numbers

- **Huge difference between a hit and a miss**
  - Could be 100x, if just L1 and main memory
- **Would you believe 99% hits is twice as good as 97%?**
  - Assume:
    cache hit time of 1 cycle
    miss penalty of 100 cycles
  - Average access time:
    97% hits: 1 cycle + 0.03 * 100 cycles = **4 cycles**
    99% hits: 1 cycle + 0.01 * 100 cycles = **2 cycles**
- **This is why "miss rate" is used instead of "hit rate"**

Types of Cache Misses

- **Cold (compulsory) miss**
  - Occurs on first access to a block
- **Conflict miss**
  - Most hardware caches limit blocks to a small subset (sometimes a singleton) of the available cache slots
    - e.g., block i must be placed in slot (i mod 4)
  - Conflict misses occur when the cache is large enough, but multiple data objects all map to the same slot
    - e.g., referencing blocks 0, 8, 0, 8, ... would miss every time
- **Capacity miss**
  - Occurs when the set of active cache blocks (working set) is larger than the cache

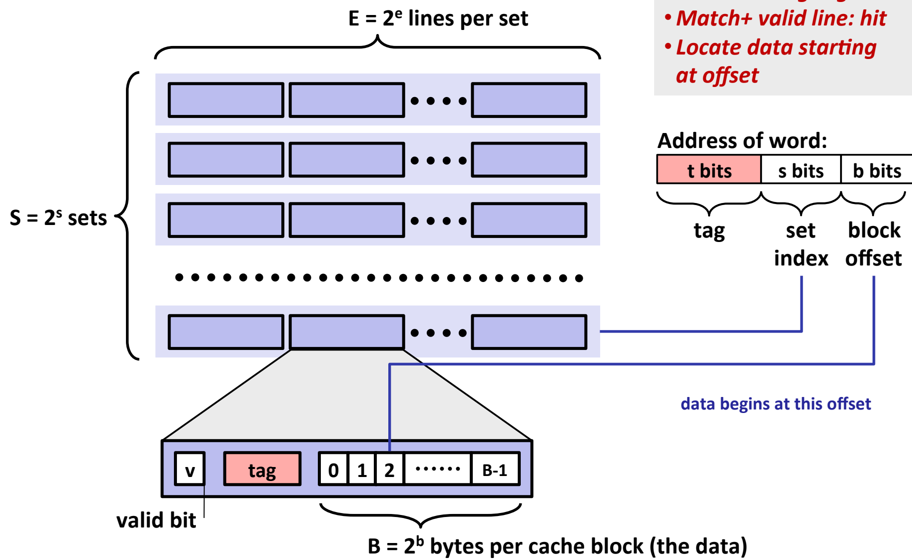
Examples of Caching in the Hierarchy

Cache Type	What is Cached?	Where is it Cached?	Latency (cycles)	Managed By
Registers	4-byte words	CPU core	0	Compiler
TLB	Address translations	On-Chip TLB	0	Hardware
L1 cache	64-bytes block	On-Chip L1	1	Hardware
L2 cache	64-bytes block	Off-Chip L2	10	Hardware
Virtual Memory	4-KB page	Main memory	100	Hardware+OS
Buffer cache	Parts of files	Main memory	100	OS
Network buffer 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

Sewanee: The University of the South

Cache Read

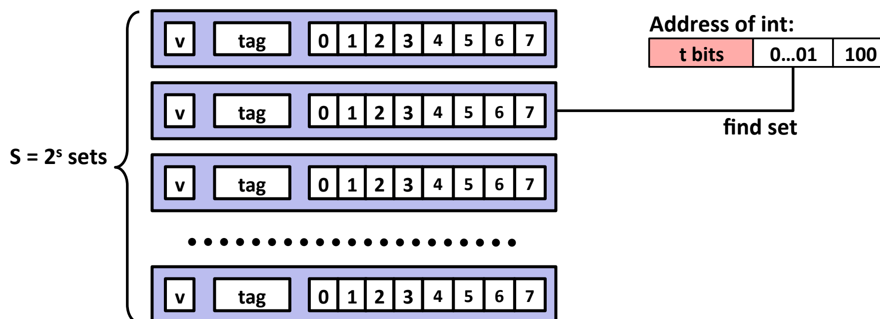
- *Locate set*
- *Check if any line in set has matching tag*
- *Match+ valid line: hit*
- *Locate data starting at offset*



Sewanee: The University of the South

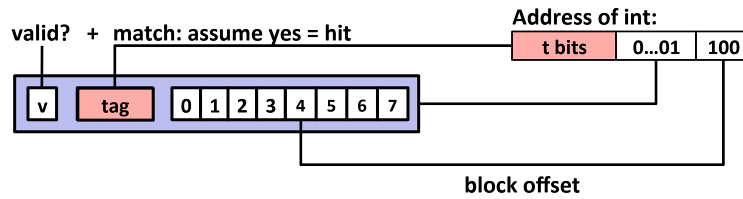
Example: Direct Mapped Cache (E = 1)

Direct mapped: One line per set
Assume: cache block size 8 bytes



Example: Direct Mapped Cache (E = 1)

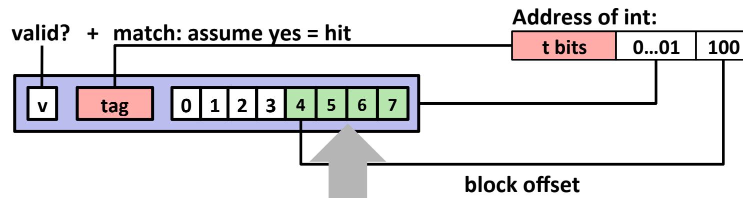
Direct mapped: One line per set
Assume: cache block size 8 bytes



Sewanee: The University of the South

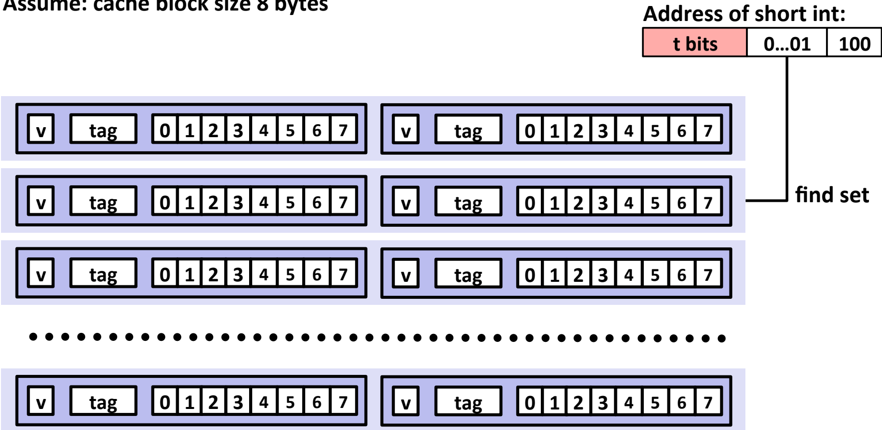
Example: Direct Mapped Cache (E = 1)

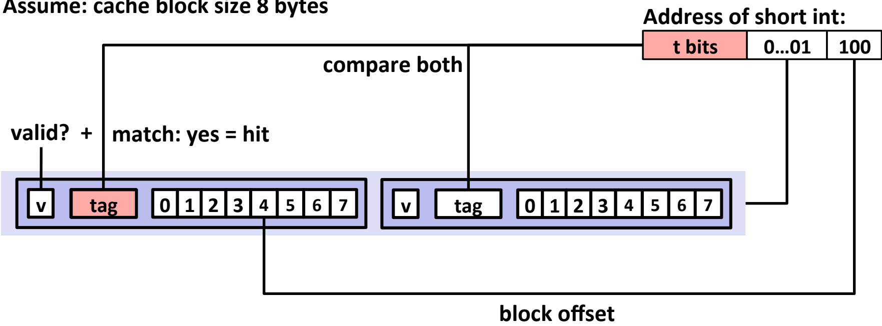
Direct mapped: One line per set
Assume: cache block size 8 bytes



int (4 Bytes) is here

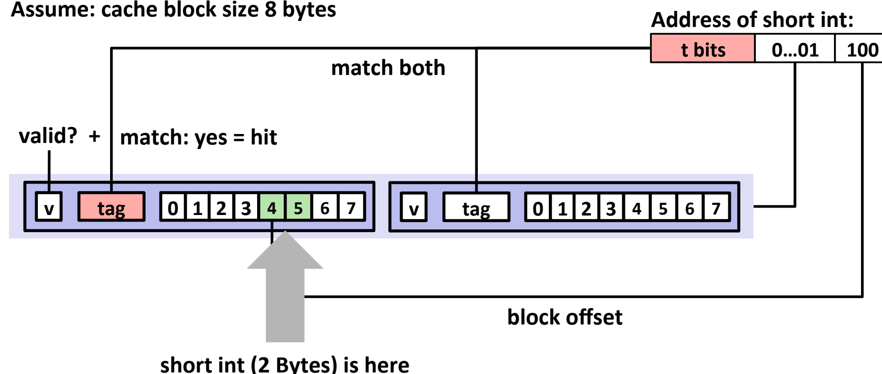
No match: old line is evicted and replaced


Sewanee: The University of the South
E-way Set Associative Cache (Here: E = 2)
E = 2: Two lines per set
Assume: cache block size 8 bytes
Address of short int:
t bits
0…01
100
v
tag
0 1 2 3 4 5 6 7
find set


Sewanee: The University of the South
E-way Set Associative Cache (Here: E = 2)
E = 2: Two lines per set
Assume: cache block size 8 bytes
Address of short int:
t bits
0…01
100
compare both
valid? +
match: yes = hit
v
tag
0 1 2 3 4 5 6 7
block offset

E-way Set Associative Cache (Here: E = 2)

E = 2: Two lines per set
Assume: cache block size 8 bytes



No match:
- One line in set is selected for eviction and replacement
- Replacement policies: random, least recently used (LRU), …

What about writes?

- **Multiple copies of data exist:**
  - L1, L2, Main Memory, Disk
- **What to do one a write-hit?**
  - Write-through (write immediately to memory)
  - Write-back (defer write to memory until replacement of line)
    - Need a dirty bit (line different from memory or not)
- **What to do on a write-miss?**
  - Write-allocate (load into cache, update line in cache)
    - Good if more writes to the location follow
  - No-write-allocate (writes immediately to memory)
- **Typical**
  - Write-through + No-write-allocate
  - **Write-back + Write-allocate**

Software Caches are More Flexible

Examples

- File system buffer caches, web browser caches, etc.

Some design differences

- Almost always fully associative
  - so, no placement restrictions
  - index structures like hash tables are common
- Often use complex replacement policies
  - misses are very expensive when disk or network involved
  - worth thousands of cycles to avoid them
- Not necessarily constrained to single "block" transfers
  - may fetch or write-back in larger units, opportunistically

Sewanee: The University of the South

Optimizations for the Memory Hierarchy

- **Write code that has locality**
  - Spatial: access data contiguously
  - Temporal: make sure access to the same data is not too far apart in time
- **How to achieve?**
  - Proper choice of algorithm
  - Loop transformations
- **Cache versus register level optimization:**
  - In both cases locality desirable
  - Register space much smaller + requires scalar replacement to exploit temporal locality
  - Register level optimizations include exhibiting instruction level parallelism (conflicts with locality)