[Chap.6-2] The Memory Hierarchy

Young Ik Eom (yieom@skku.edu, 031-290-7120)
Distributing Computing Laboratory
Sungkyunkwan University
http://dclab.skku.ac.kr

SKK UNIVERSITY

Contents

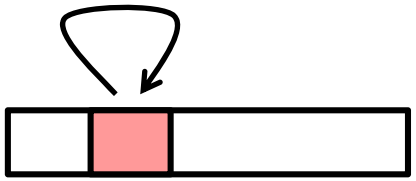
- Storage technologies
- Locality
- Memory hierarchy
- Cache memories
- Writing cache-friendly code
- Impact of caches on program performance

Locality

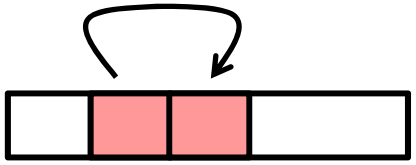
Principle of locality

- 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



Locality

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 the loop repeatedly — **Temporal locality**

Locality

- **Note)**
  - **Stride-k reference pattern**
    - Visiting every k^{th} element of a contiguous vector
  - **Stride-1 reference pattern**
    - Sequential reference pattern
    - Common and important source of spatial locality in programs
  - **As the stride increases, the spatial locality decreases**

Locality

- **Example-1)**
  - Does this function have good locality with respect to array **a**?
    - References the array in the same row-major order that the array is stored
    - Nice stride-1 reference pattern

```
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;
}
```

Locality

- **Example-2)**
  - Does this function have good locality with respect to array **a**?
    - Interchanged the **i** and **j** loops
    - Scans the array column-wise instead of low-wise
    - Stride-N reference pattern (poor spatial locality)

```
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;
}
```

Locality

- **Example-3)**
  - 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[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;
}
```

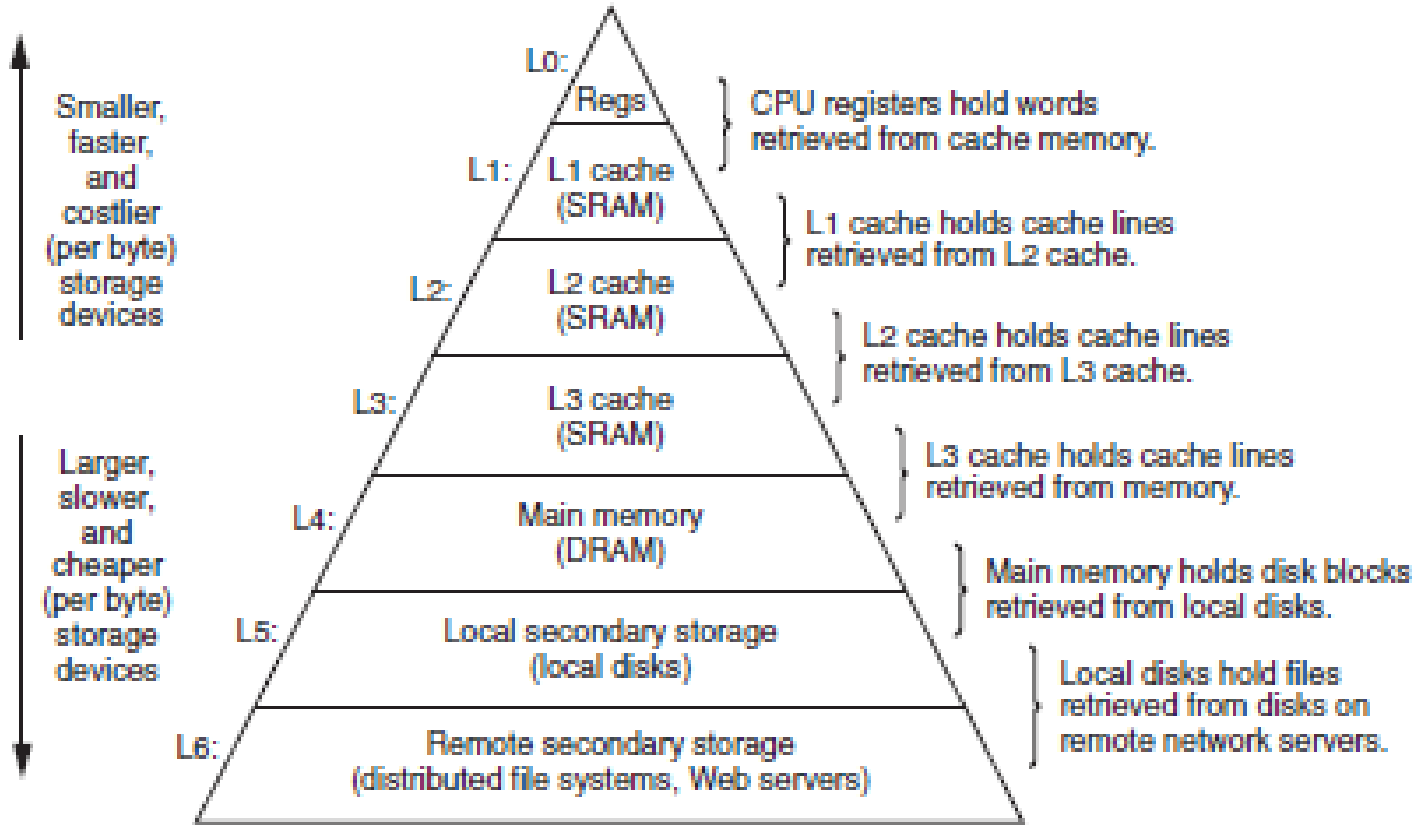
Memory Hierarchy

- **Some fundamental and enduring properties of hardware and software:**
  - Fast storage technologies cost more per byte, have less capacity, and require more power
  - The gap between CPU and 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**

Memory Hierarchy

Memory hierarchy



Memory Hierarchy

Memory hierarchy

- 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

Memory Hierarchy

Memory hierarchy

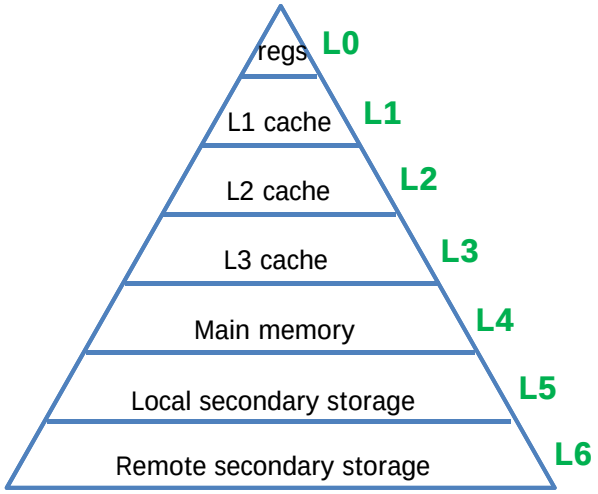
- Big idea
  - The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top

Memory Hierarchy

Memory hierarchy

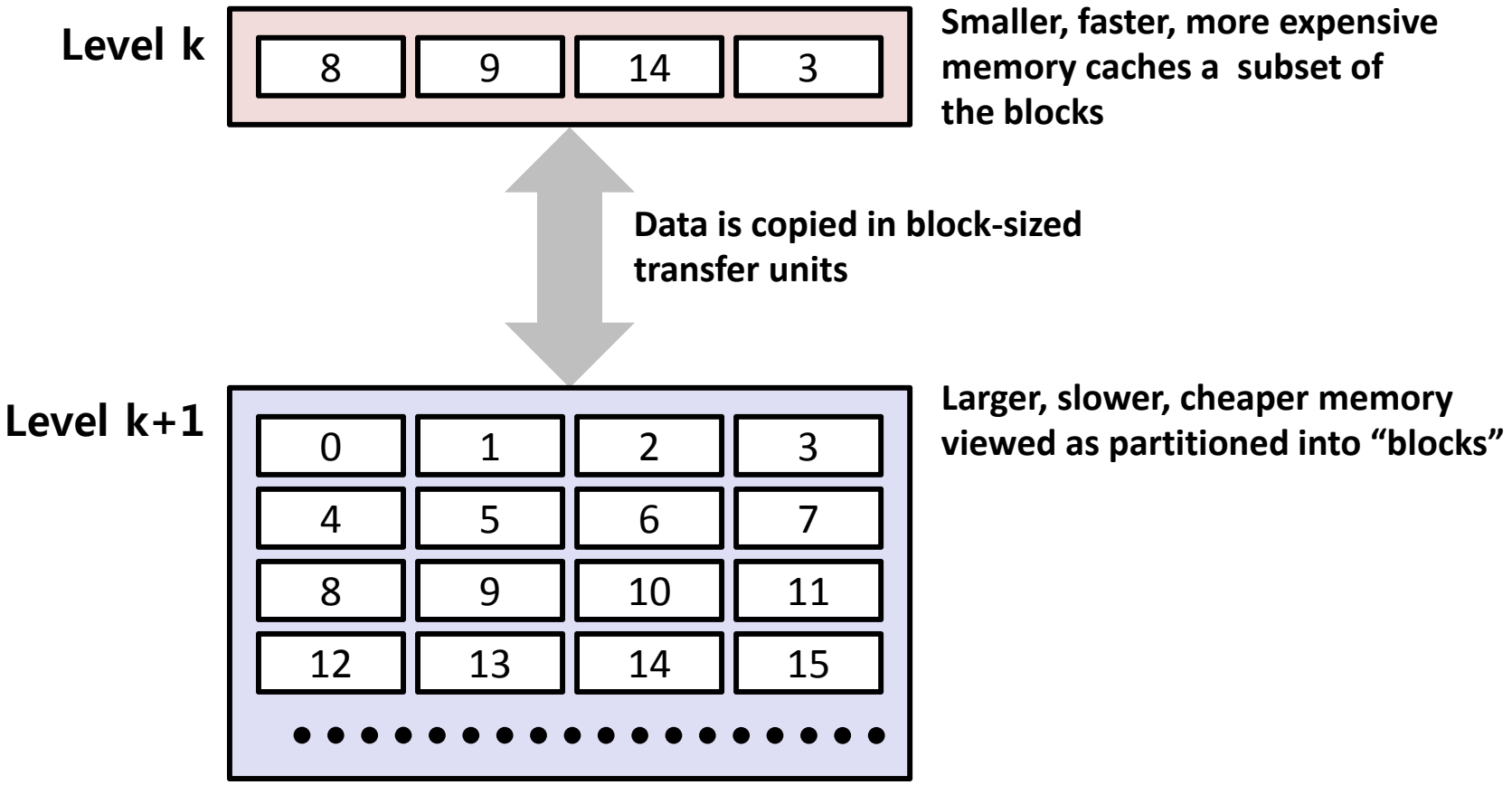
- Typical transfer units
  - 1 word between L1 and L0
  - 8~16 words between L2 and L1 (L3 and L2, L4 and L3)
  - Hundreds or thousands of bytes between L5 and L4

  - Generally, devices lower in the hierarchy have longer access times, and thus tend to use larger block sizes



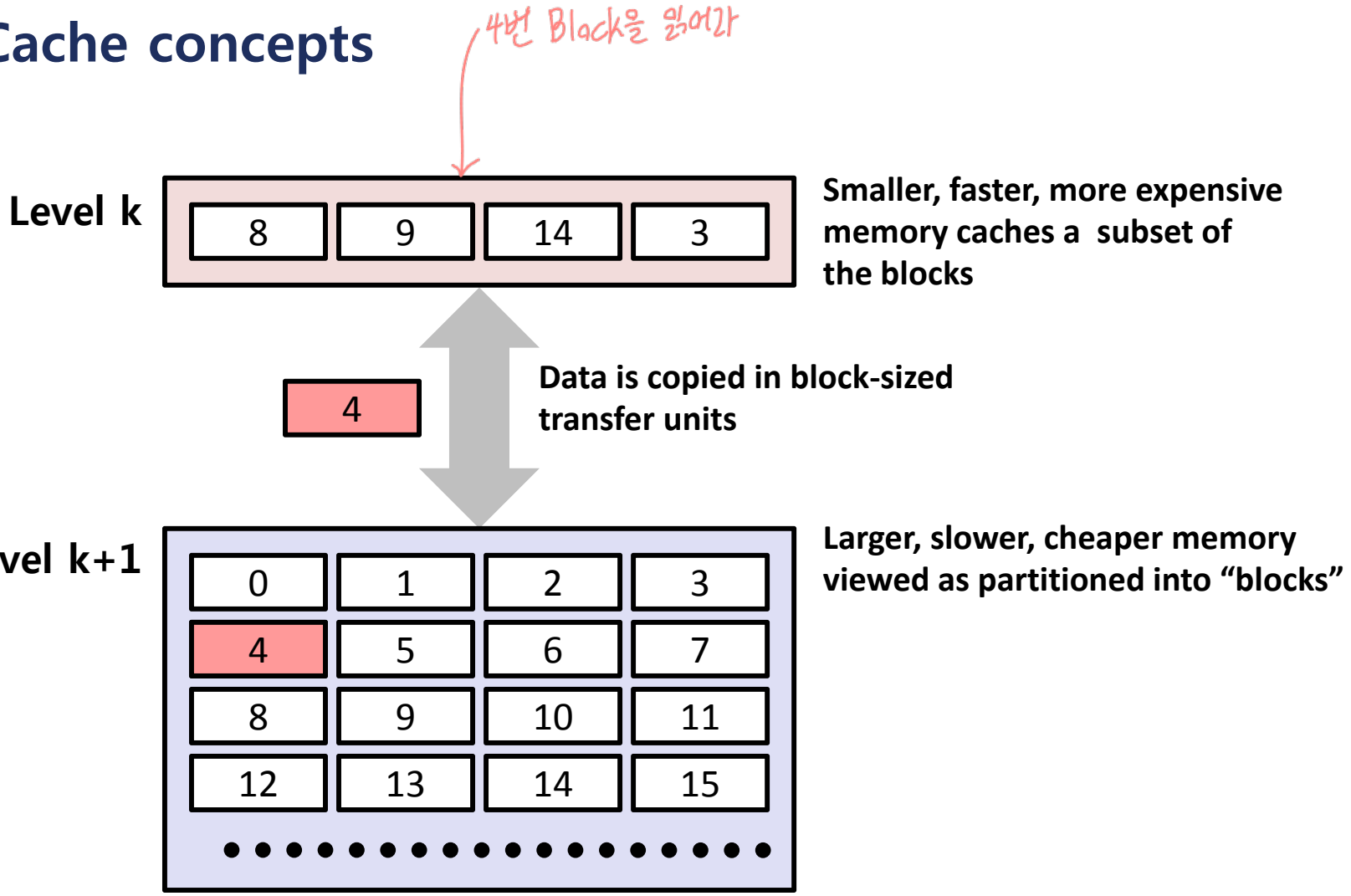
Memory Hierarchy

Cache concepts



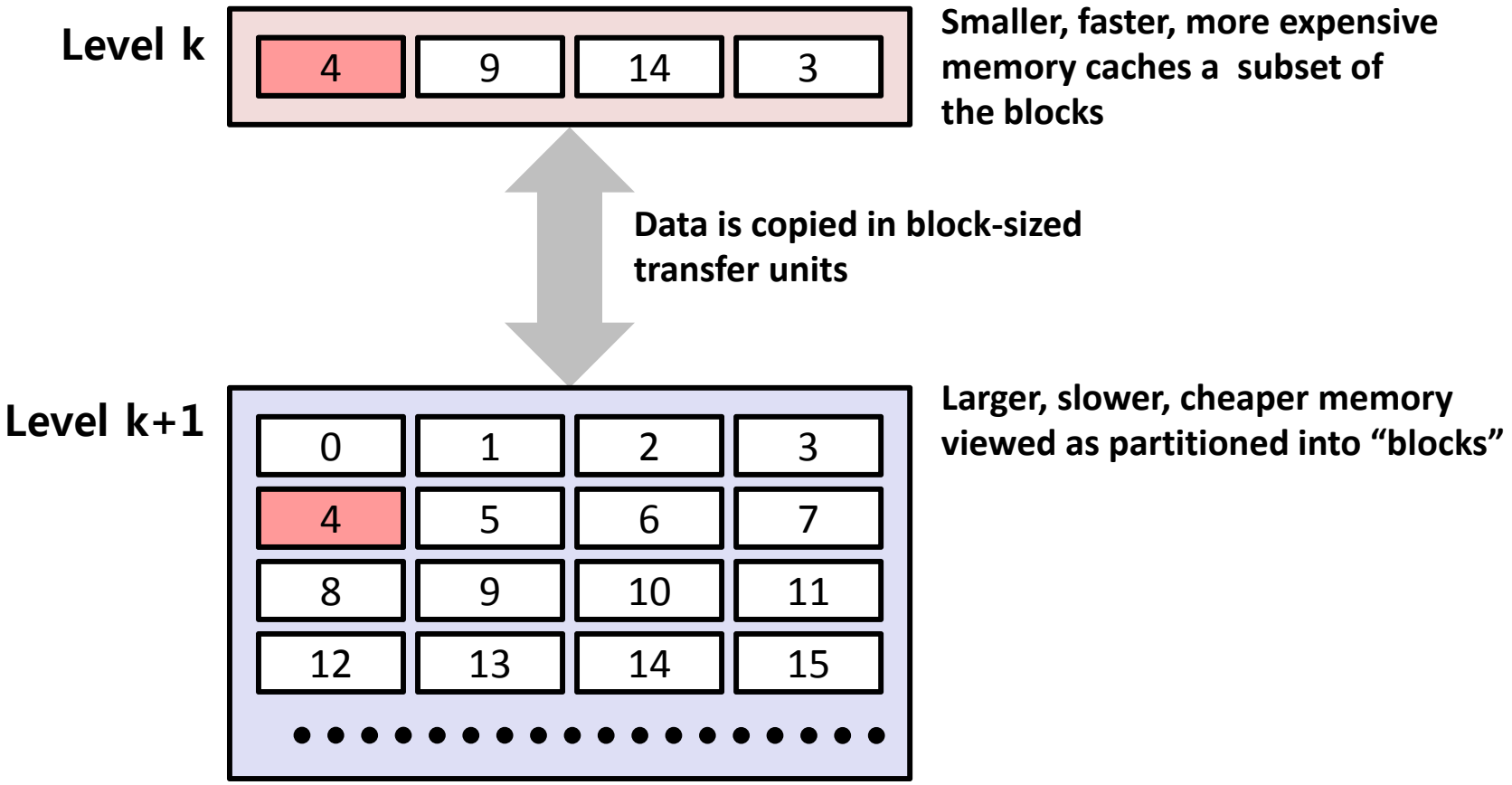
Memory Hierarchy

Cache concepts



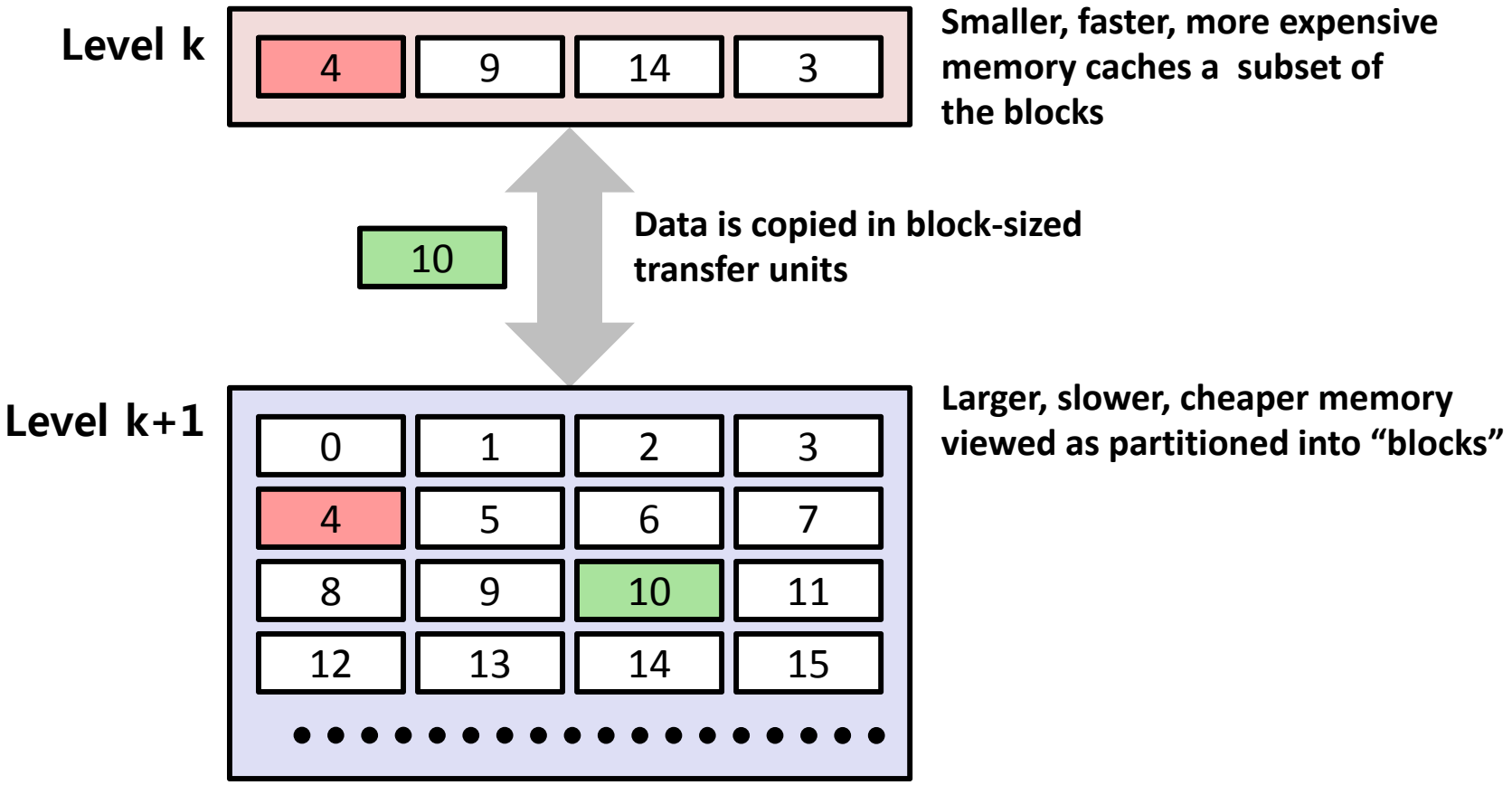
Memory Hierarchy

Cache concepts



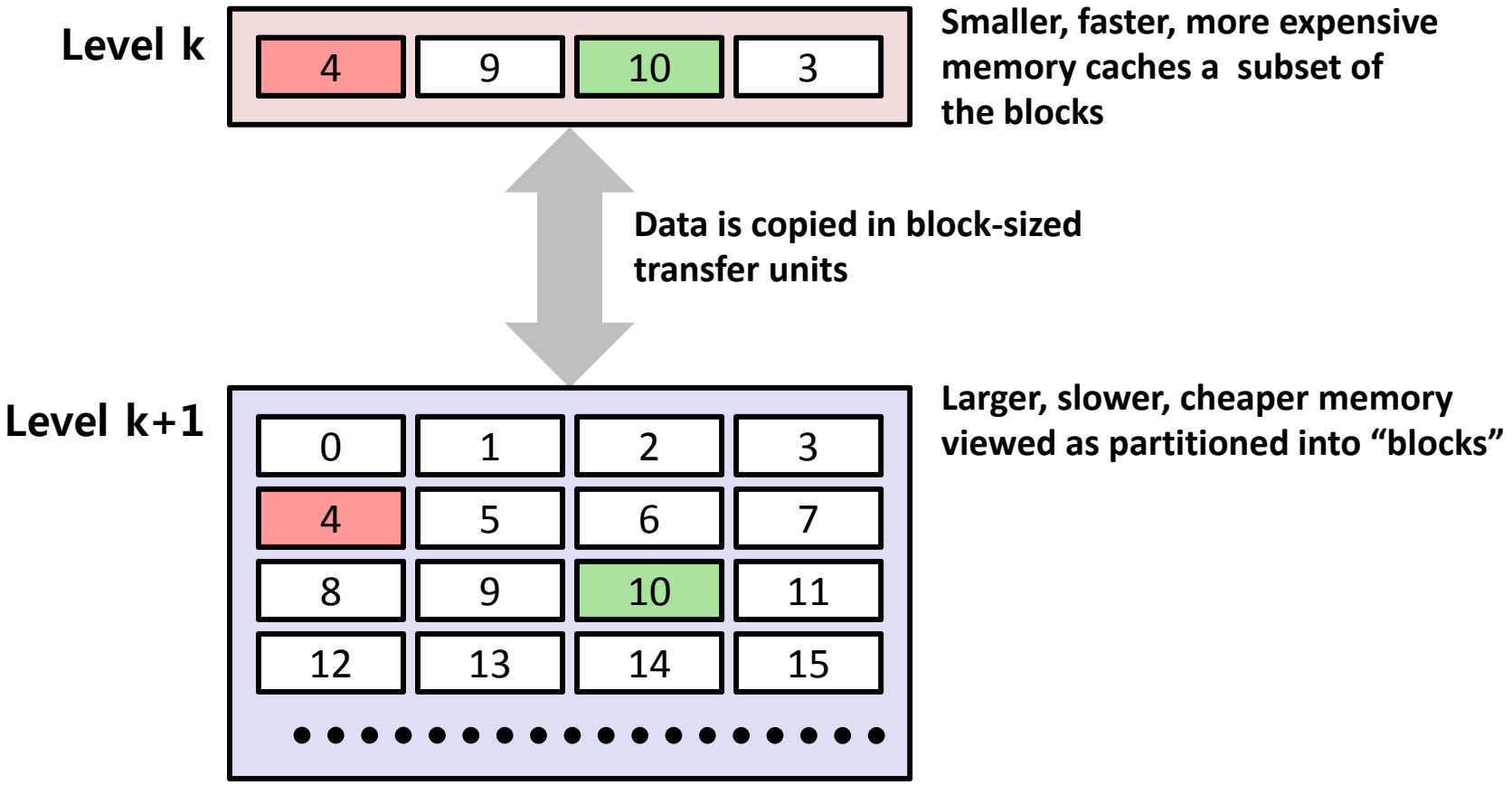
Memory Hierarchy

Cache concepts



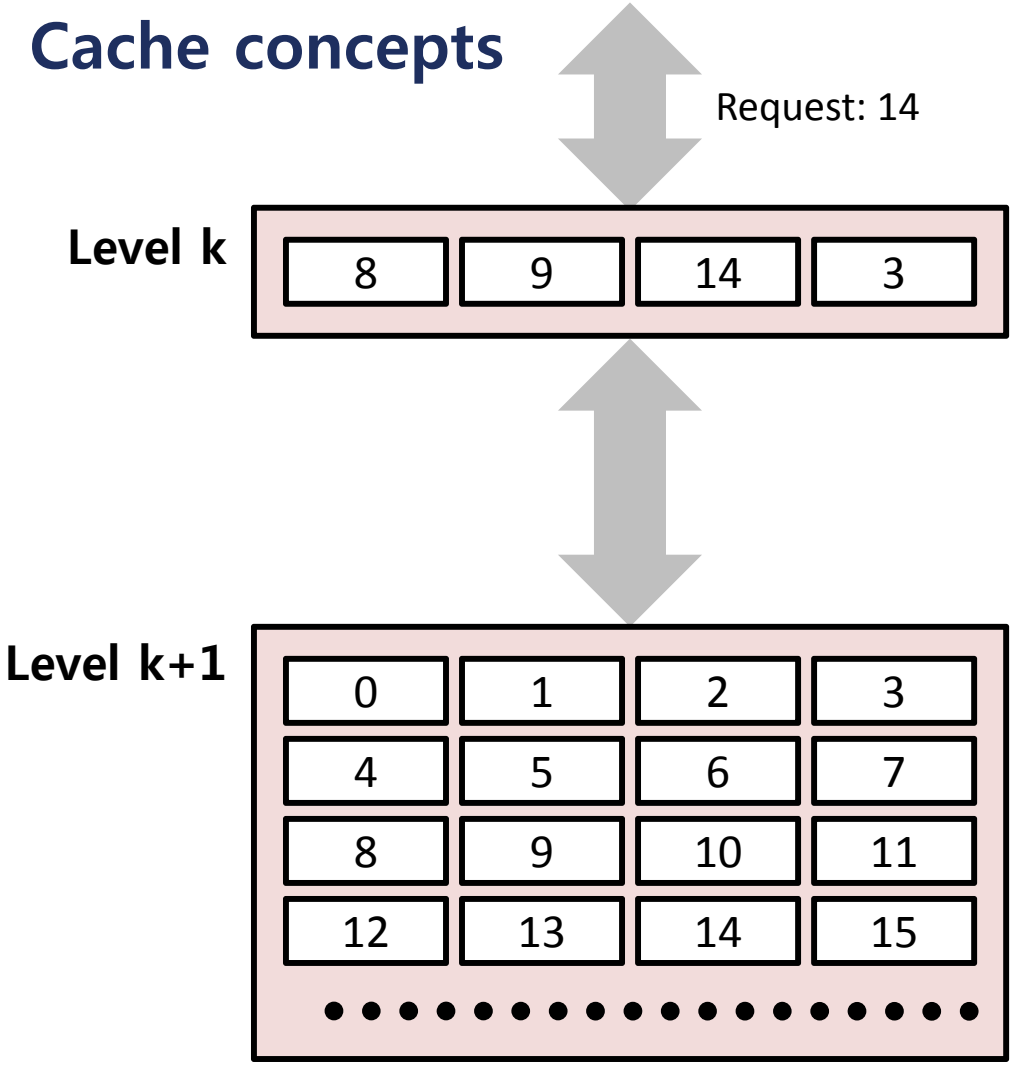
Memory Hierarchy

Cache concepts



Memory Hierarchy

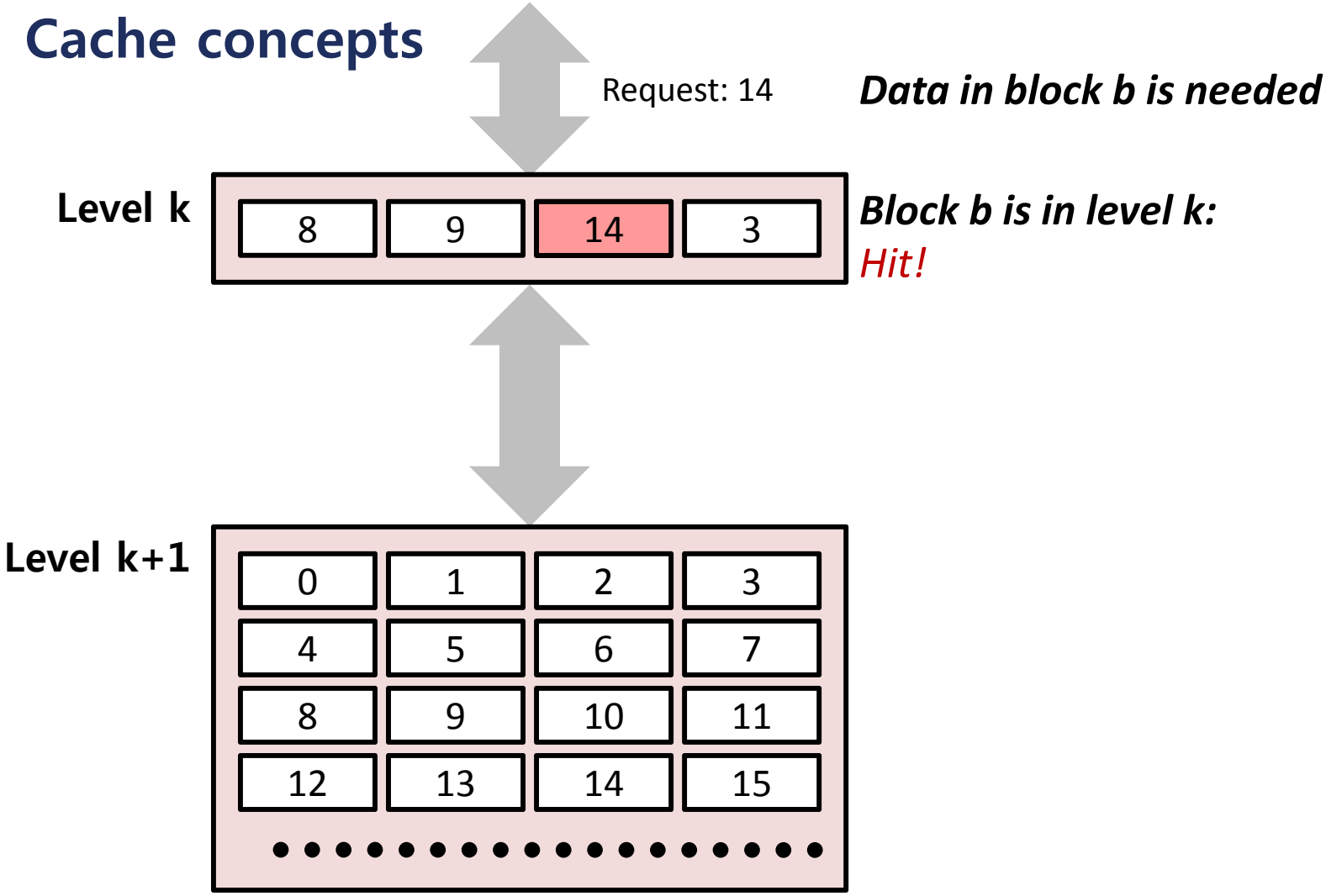
Cache concepts



Data in block b is needed

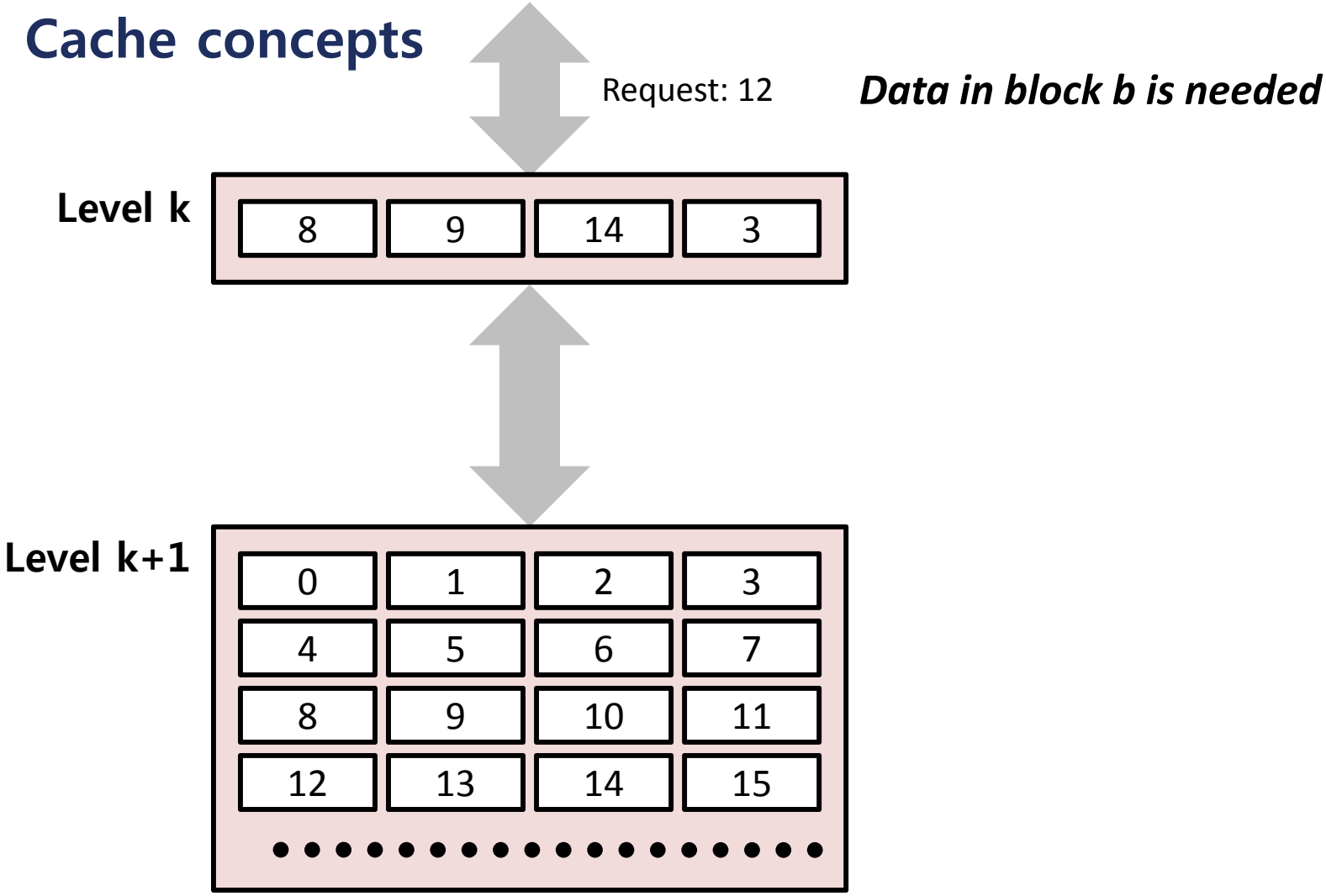
Memory Hierarchy

Cache concepts



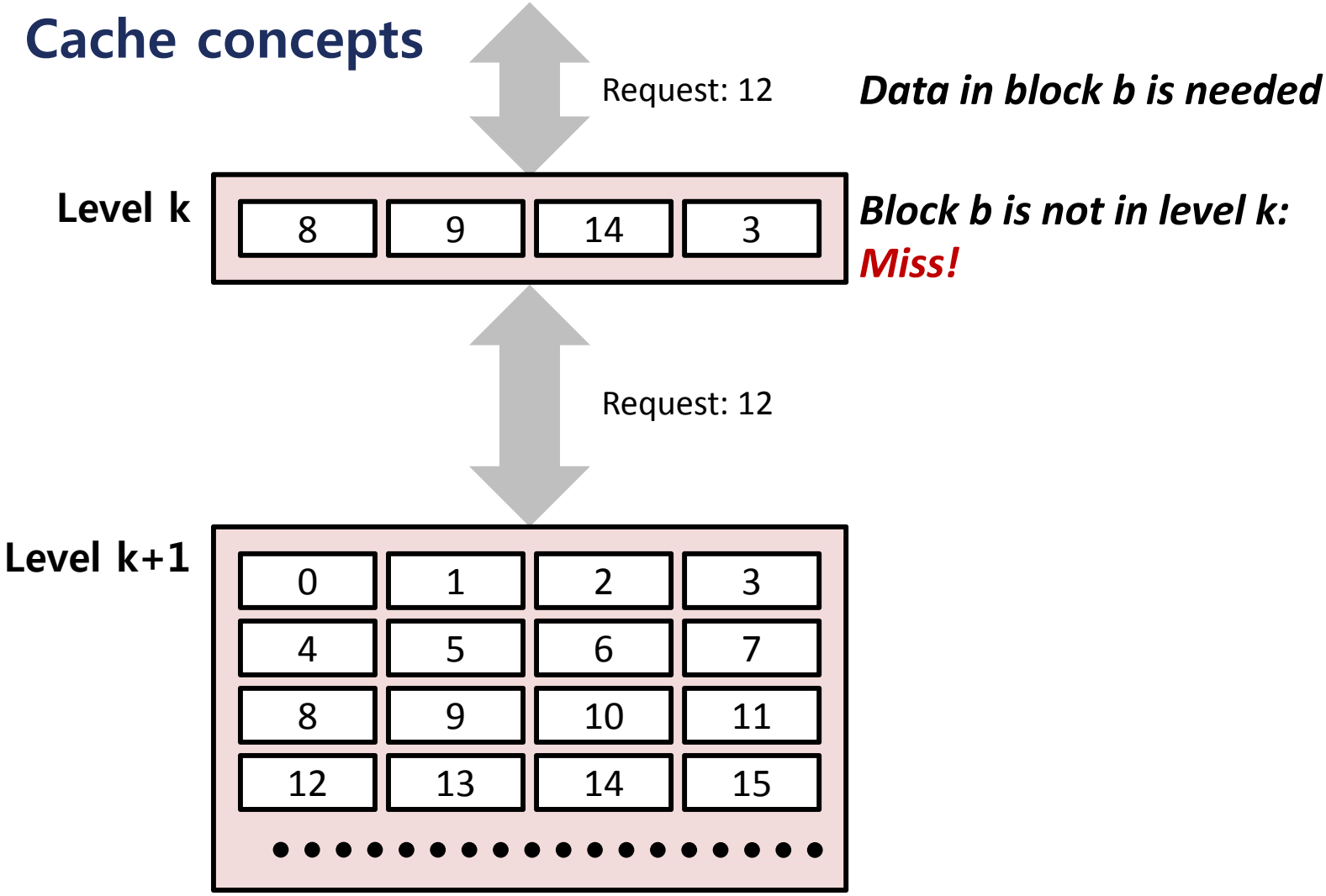
Memory Hierarchy

Cache concepts



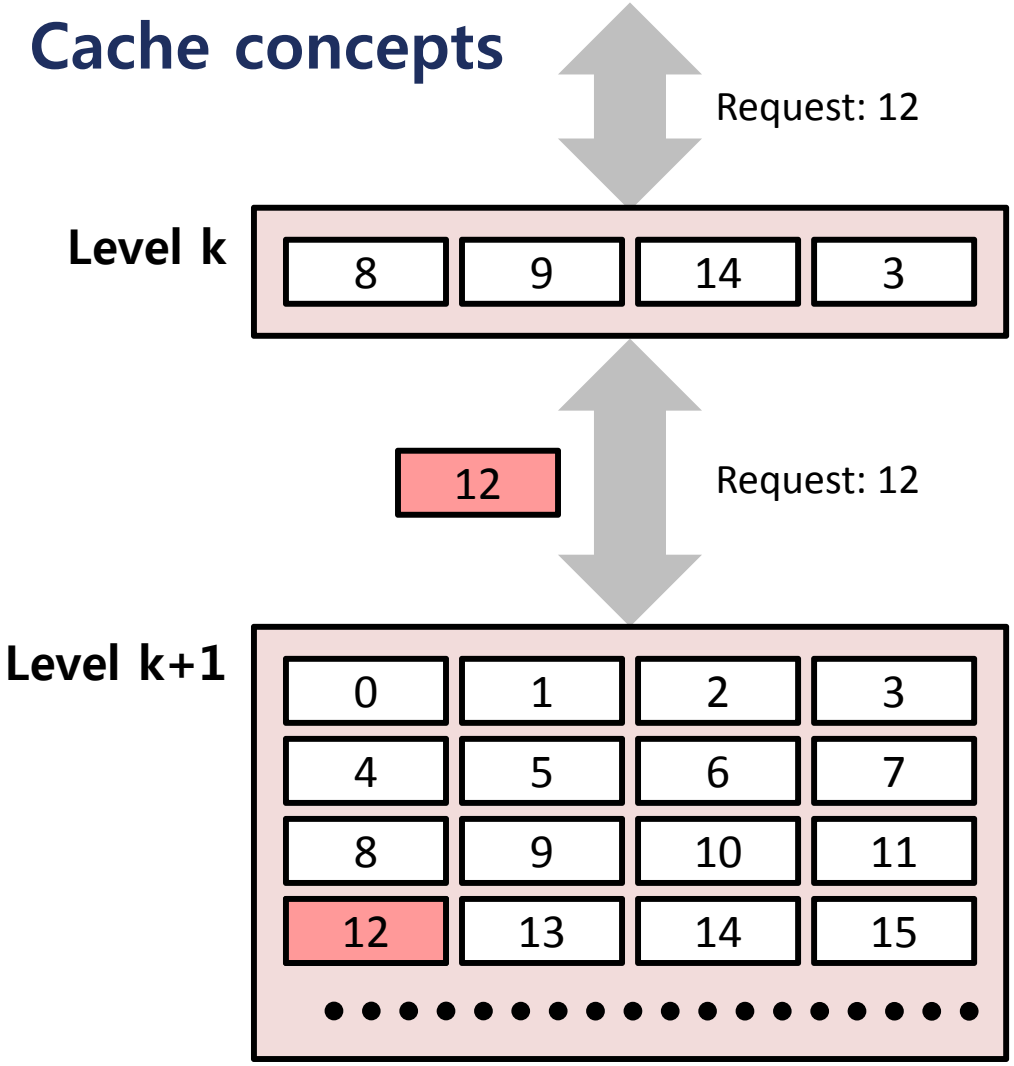
Memory Hierarchy

Cache concepts



Memory Hierarchy

Cache concepts



Data in block b is needed

Block b is not in level k:
Miss!

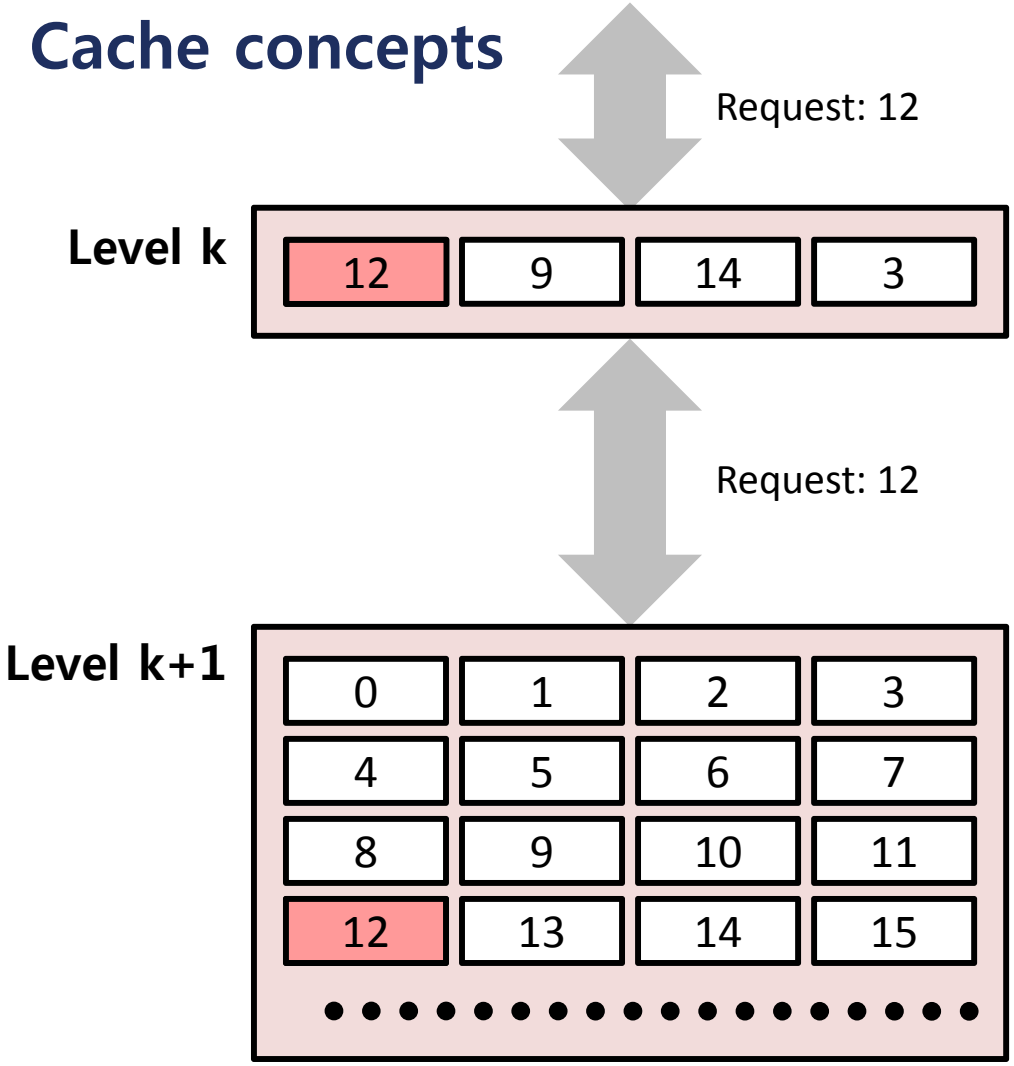
Block b is fetched from Level k+1

Block b is stored in level k

- **Placement policy**:
  Determines where **b** goes
- **Replacement policy**:
  Determines which block gets evicted (victim)

Memory Hierarchy

Cache concepts



Data in block b is needed

Block b is not in level k:
Miss!

Block b is fetched from Level k+1

Block b is stored in level k

- **Placement policy**:
  Determines where **b** goes
- **Replacement policy**:
  Determines which block gets evicted (victim)

Memory Hierarchy

- **Types of cache misses**
  - **Cold (compulsory) miss**
    - Occurs because the cache is empty
  - **Conflict miss**
    - Most caches limit blocks at level **k**+1 to a small subset (sometimes a singleton) of the block positions at level **k**
      - ✓ Eg) Block **i** at level **k**+1 must be placed in block (**i** mod 4) at level **k**
    - Conflict miss occurs when the level **k** cache is large enough, but multiple data objects all map to the same block in level **k**
      - ✓ Eg) Referencing blocks 0, 8, 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

Memory Hierarchy

- **Note)**
  - How to exploit temporal locality?
    - Speed up data accesses by caching data in faster storage
    - Caching in multiple levels, forming a memory hierarchy
      - ✓ The lower levels of the memory hierarchy tend to be slower, but larger and cheaper
  - How to exploit spatial locality?
    - Larger cache line size
      - ✓ Cache nearby data together

Memory Hierarchy

Caching in modern systems

Type	What cached	Where cached	Latency (#cycles)	Managed by
CPU registers	4B or 8B words	On-chip CPU registers	0	Compiler
TLB	Address translations	On-chip TLB	0	Hardware MMU
L1 cache	64B blocks	On-chip L1 cache	4	Hardware
L2 cache	64B blocks	On-chip L2 cache	10	Hardware
L3 cache	64B blocks	On-chip L3 cache	50	Hardware
Virtual memory	4KB pages	Main memory	200	Hardware + OS
Buffer cache	Parts of files	Main memory	200	OS
Disk cache	Disk sectors	Disk controller	100,000	Controller firmware
Network cache	Parts of files	Local disk	10,000,000	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

Summary

- The speed gap between CPU, memory, and storage continues to be widen
- Well-written programs exhibit a property called **locality**
- Memory hierarchies based on caching close the gap by exploiting the **locality**