Caching Concepts and Algorithms

Onur Ascigil

Lancaster University

Outline

- Caching Principle
  - Data Storage Technologies
  - Example: Memory Hierarchy and Locality of Reference
- Basic Cache Concepts
  - Cache Replacement Policies
- Caching in the Internet
  - Server-side caching
  - Caching at Distributed Servers

Part of this material is from Chapter 6 of *Computer Systems: A Programmer's Perspective*, 3/E (CS:APP3e) by Randal E. Bryant and David R. O'Hallaron, Carnegie Mellon University

Data Storage Technologies

- **Disks –** secondary storage systems
  - **Non-volatile memory** – Won't loose information, if turned off
  - Hard Disks (HDD) – magnetic disk (more on this later)
  - Solid state drive (SSD) – Flash-based technology (more on this later)
  - Network File System (NFS), cloud-based storage (e.g., DropBox)
- **Random Access Memory (RAM) –** Known as the main memory
  - Volatile memory – Lose information if turned off
- **Registers**
  - Very small on-chip memory (32 or 64 bits in size)

Random Access Memory (RAM)

- Key features:
  - RAM is traditionally packaged as a chip
  - Basic storage unit is a cell (one bit per cell)
  - Multiple RAM chips for a memory

- RAM comes in two varieties:
  - SRAM (Static RAM)
  - DRAM (Dynamic RAM)

Non-volatile Storage

- Non-volatile memories retain value even if powered off
- Uses and examples of nonvolatile memories:
  - **Firmware** programs stored in a ROM (BIOS, controllers for disk, network cards, etc)
  - **Flash memory** is based on EEPROM and has become an important technology.
  - **Solid state disks (SSD)** are flash-based memory that is faster, sturdier and more energy efficient.

Access Times to Storage Devices

- **milliseconds (10^{-3} s.), microseconds (10^{-6} s.), or nanoseconds (10^{-9} s.)?**
- Nonvolatile Memory:
  - HDD: 5-10 milliseconds
  - SSD: 100 microseconds or less
- RAM:
  - SRAM: ~0.5-2 nanoseconds
  - DRAM: ~50-100 nanoseconds
- On-chip memory:
  - L1, L2, L3 caches - L1 is faster (smaller in size) than L2, and L2 is faster (also smaller in size) than L3
    - L1 cache is essentially SRAM, L2 and L3 are usually larger and slower SRAM designs
  - CPU registers – access time in one CPU cycle

	Trans. per bit	Access time	Needs refresh?	Cost	Applications
SRAM	6	1X	No	100x	Cache memories
DRAM	1	10X	Yes	1X	Main memories, frame buffers

Summary: Storage Technologies

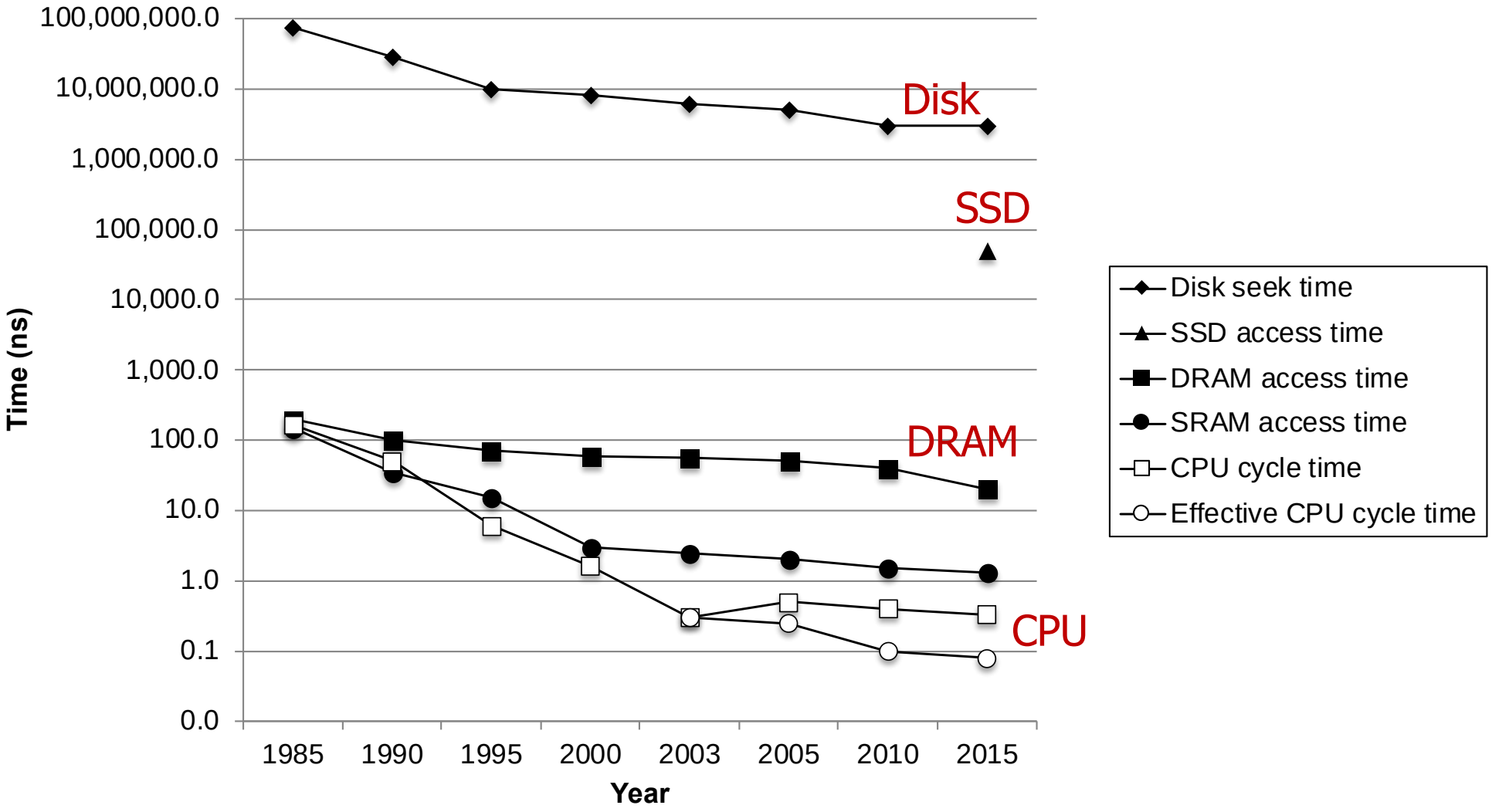
- The data transfer between CPU, memory and disk
  - CPU issues a load/store instruction
  - The data is transferred to an on-chip register
  - Disk controller performs a DMA
- Different access speeds of Disk, RAM, and on-chip memory
  - Disk is the slowest, i.e., more than 10^5 times slower than RAM
  - SSD is significantly faster than HDD

Lancaster University

Outline

- Caching Principle
  - Data Storage Technologies
  - **Example: Memory Hierarchy and Locality of Reference**
- Basic Cache Concepts
  - Cache Replacement Policies
- Caching in the Internet
  - Server-side caching
  - Caching at Distributed Servers

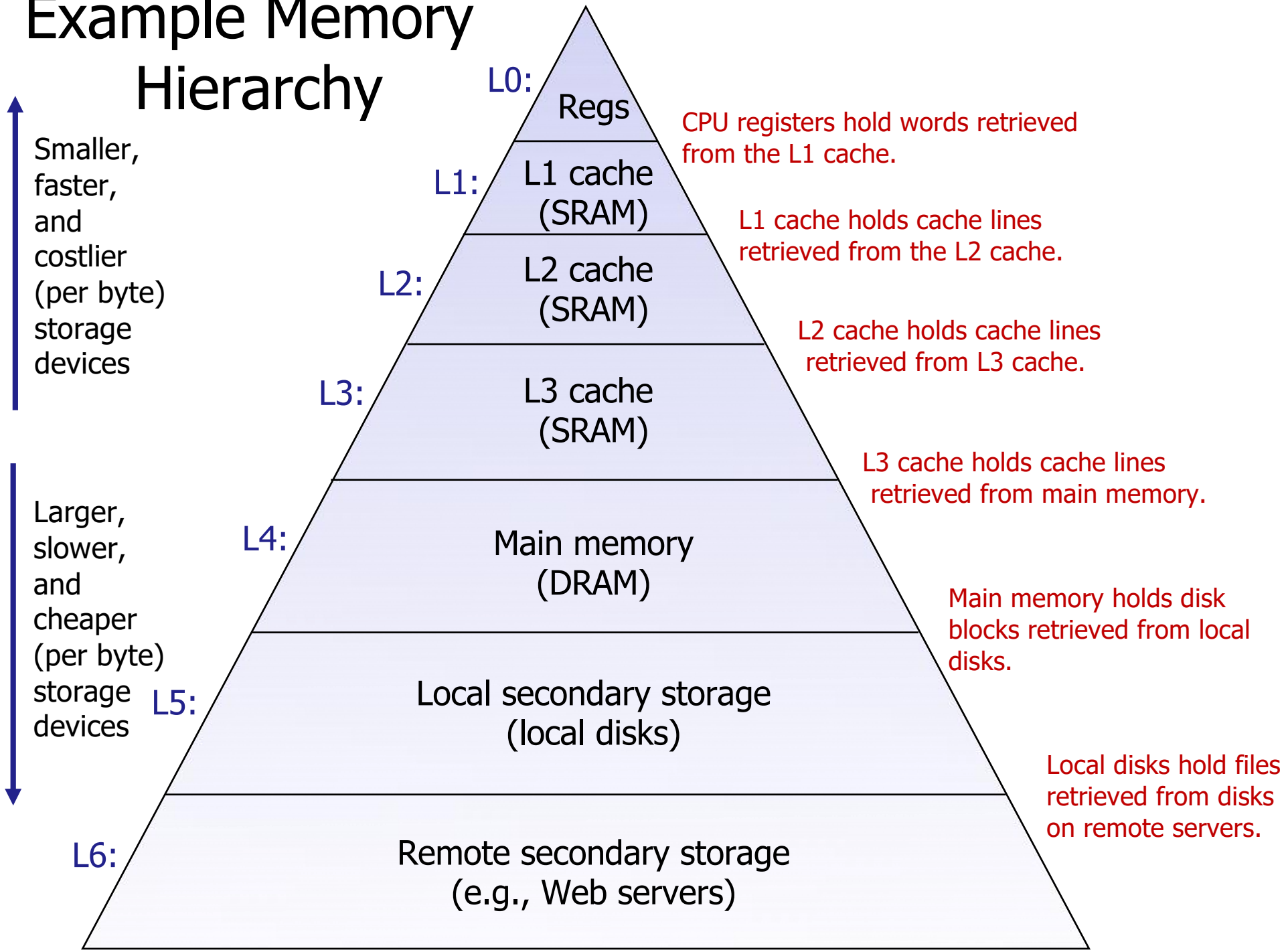
CPU vs Memory Speeds: The gap widens



Why is CPU-Memory gap important?

- CPU is more than ~100x faster than DRAM
  - SRAM is faster than DRAM, but also 100x more expensive than DRAM
- A 2 GHz processor can execute **two million** instructions per millisecond
  - Takes 16 milliseconds to read data from HDD – Equivalent of the time for executing 32 million instructions!
  - Takes 100 microseconds to read data from SSD – Equivalent of 200,000 intructions
- Simply waiting and doing nothing while the transfer is taking place would be enormously wasteful.
  - This is of course not what the CPUs do.
  - A CPU switches to other tasks upon issuing a read instruction and is not involved in the actual reading of data from a disk (that's disk controller's job).
- However, the gap is still important as it slows down processes that involve reads from memory!

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)
  - Each level in the hierarchy k acts as a cache for level k+1 in the hierarchy.
- **Goal:** to achieve the **access speed of the fastest memory**, while paying approximately for **the cost of the cheapest** (i.e., slowest and largest in size) memory in the hierarchy.
- This sounds great but does it work in practice?
  - **Yes, as long as there is locality!**

Principle of locality

- **Important observation:** Programs tend to use data and instructions with addresses near or equal to those they have used recently in the memory.
  - This means there is locality!
- **Temporal locality:**
  - Recently accessed data is likely to be accessed again in the near future.
  - **Strategy to exploit temporal locality**: keep the recently accessed data in a cache (for some time)
- **Spatial locality:**
  - Data near recently accessed addresses is likely to be accessed soon.
  - **Strategy to exploit spatial locality:** Prefetch data (near those that have been used) or instructions into the cache before they are explicitly requested.
- **When there is locality, caching** and **pre-fetching** are effective in achieving a high hit rate (see slide 23)
  - This makes access times closer to that of the cache instead of slower memory levels, significantly improving performance!

Examples of locality

- References to data
  - Reading elements of a[] in succession
    - **Spatial**
  - Repeatedly reading the same variable (sum)
    - **Temporal**
- References to instruction (code)
  - Reference instructions in sequence
    - **Spatial**
  - Cycle through loop repeatedly
    - **Temporal**

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

Estimating Locality in a Program

- **Claim:** Being able to look at code and get a sense of its locality is a key skill for a programmer
- **Question:** Does this function have good locality with respect to a[][]?

- **Hint**: The arrays are stored in the memory in a row-major order:
  - The consecutive elements of a row reside next to each other

YES!

```
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;
}
```

Importance of Access Patterns

- A function visits each element of a vector sequentially is said to have a *stride-1 reference pattern* (with respect to the element size).
- A stride-1 reference pattern is also known as *sequential reference patterns*.
- Visiting every kth element of a contiguous vector is called a *stride*-k *reference pattern*.
- Stride-1 reference patterns are a common and important source of spatial locality in programs. In general, as the stride increases, the spatial locality decreases.

Estimating Locality in a Program

- Seemingly trivial changes to a program can have a big impact on its locality and therefore the performance.

```
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;
}
```

Address	0	4	8	12	16	20
Contents	a_{00}	a_{01}	a_{02}	a_{10}	a_{11}	a_{12}
Access order	1	2	3	4	5	6

Estimating Locality in a Program

- **Claim:** Being able to look at code and get a sense of its locality is a key skill for a programmer
- **Question:** Does this one have a good locality (**notice that the loop is different from the previous one**)?

No, unless M is very small

Address	0	4	8	12	16	20
Contents	a_{00}	a_{01}	a_{02}	a_{10}	a_{11}	a_{12}
Access order	1	3	5	2	4	6

```
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;
}
```

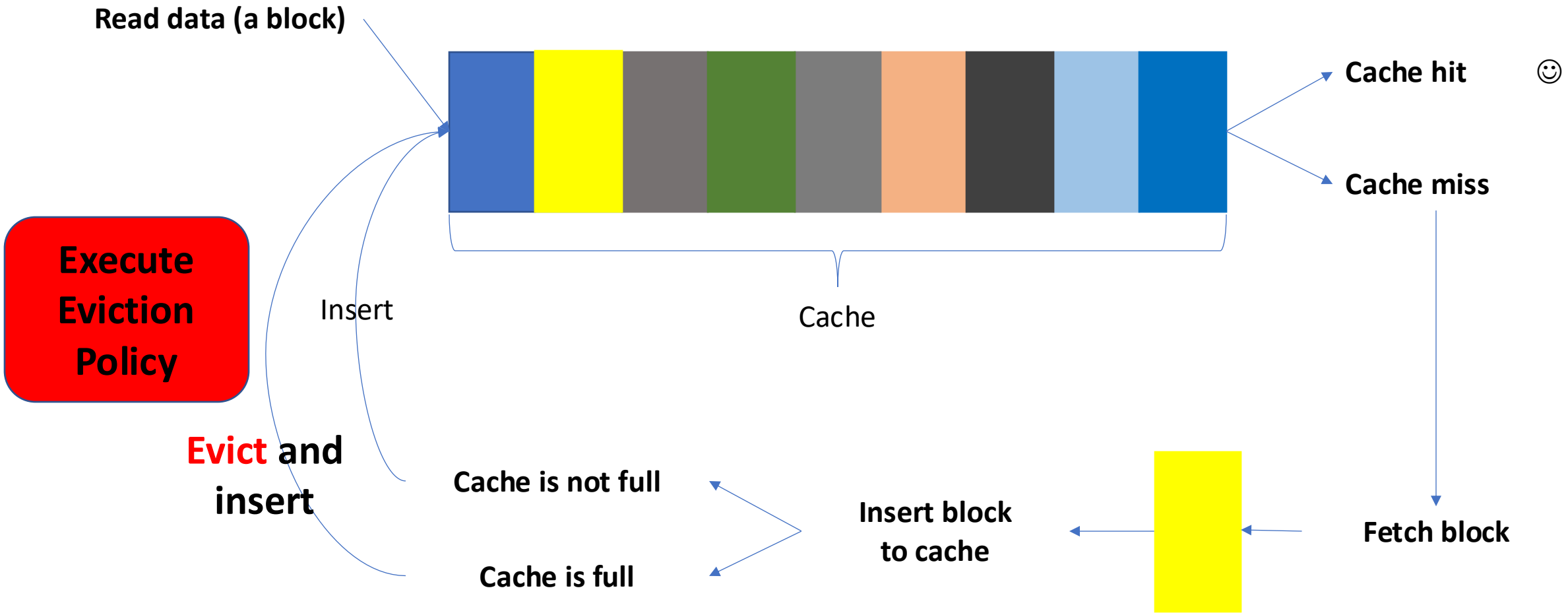
Summary of Locality

- Programs that repeatedly reference the same variables enjoy good temporal locality
- For programs with stride-k reference patterns, the smaller the stride the better the spatial locality.
- Programs with stride-1 reference patterns have good spatial locality.
- Programs that hop around memory with large strides have poor spatial locality.
- Loops have good temporal and spatial locality with respect to instruction fetches. **The smaller the loop body and the greater the number of loop iterations, the better the locality.**

Outline

- Caching Principle
  - Data Storage Technologies
  - Example: Memory Hierarchy and Locality of Reference
- **Basic Cache Concepts**
  - **Cache Replacement Policies**
- Caching in the Internet
  - Server-side Caching
  - Caching at Distributed Servers

Basic Cache Concepts: Operations



Basic Cache Concepts

- Caches are critical for overall performance of a system
  - E.g., Memory access time in hierarchical storage systems as we saw earlier
  - **Cache Hit Rate: Number of Hits / Number of Lookups** (expressed in percentage %)
- Cache Space is scarce
  - Which block to evict when inserting to a full cache?
  - **Cache Replacement (Eviction) Policy**
- Given information of the future lookups (and their order) to a cache
  - **Belady's algorithm** evicts the block that won't be needed for the longest time in future
- In practice, policies must cope with uncertainty, never knowing when cached blocks will be read in the future.

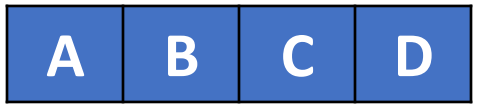
Basic Cache Concepts

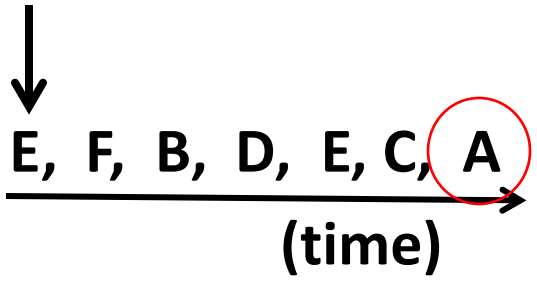
- A cache is initially empty, so in the beginning an access to any block will result in a miss.
  - A missed block is **placed** in the cache as long as there is space
- An empty or non-full cache is sometimes referred to as a ***cold cache***
  - Lookup to a cold cache can cause a miss, also called *compulsory miss* or *cold miss*.
- Cold misses are important because they are often transient events that might not occur in steady state.
  - After the cache has been *warmed up (i.e., full)* by repeated memory accesses.
- Once the cache is full, a missed block **replaces** an existing (stored) block in the cache

Belady's Algorithm Example

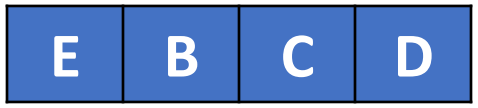
Data Block

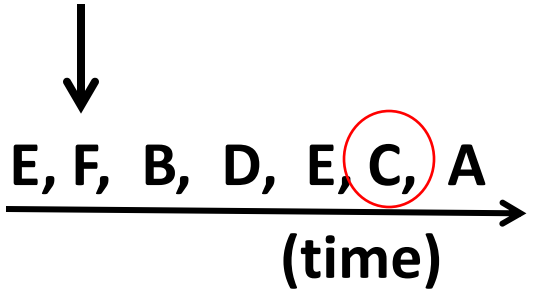
Considering a cache with a size of 4 blocks.



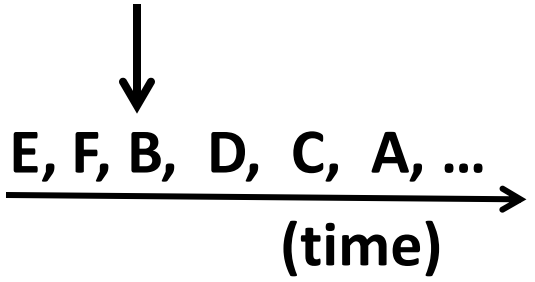


Miss

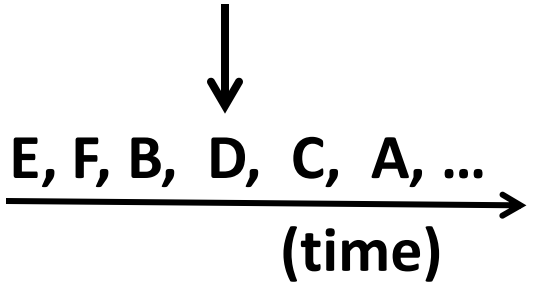




Miss



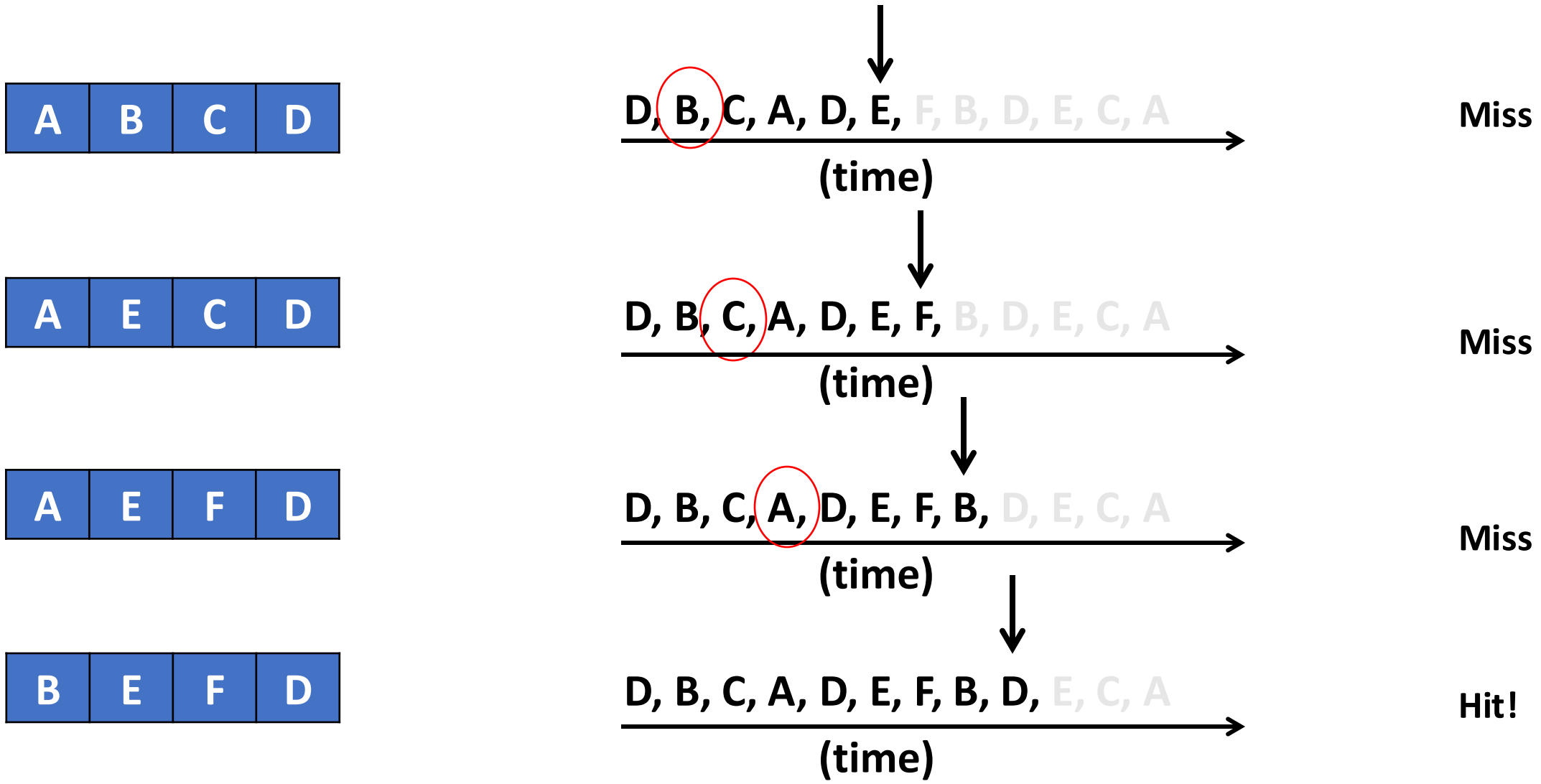
Hit



Hit

50% Hit rate

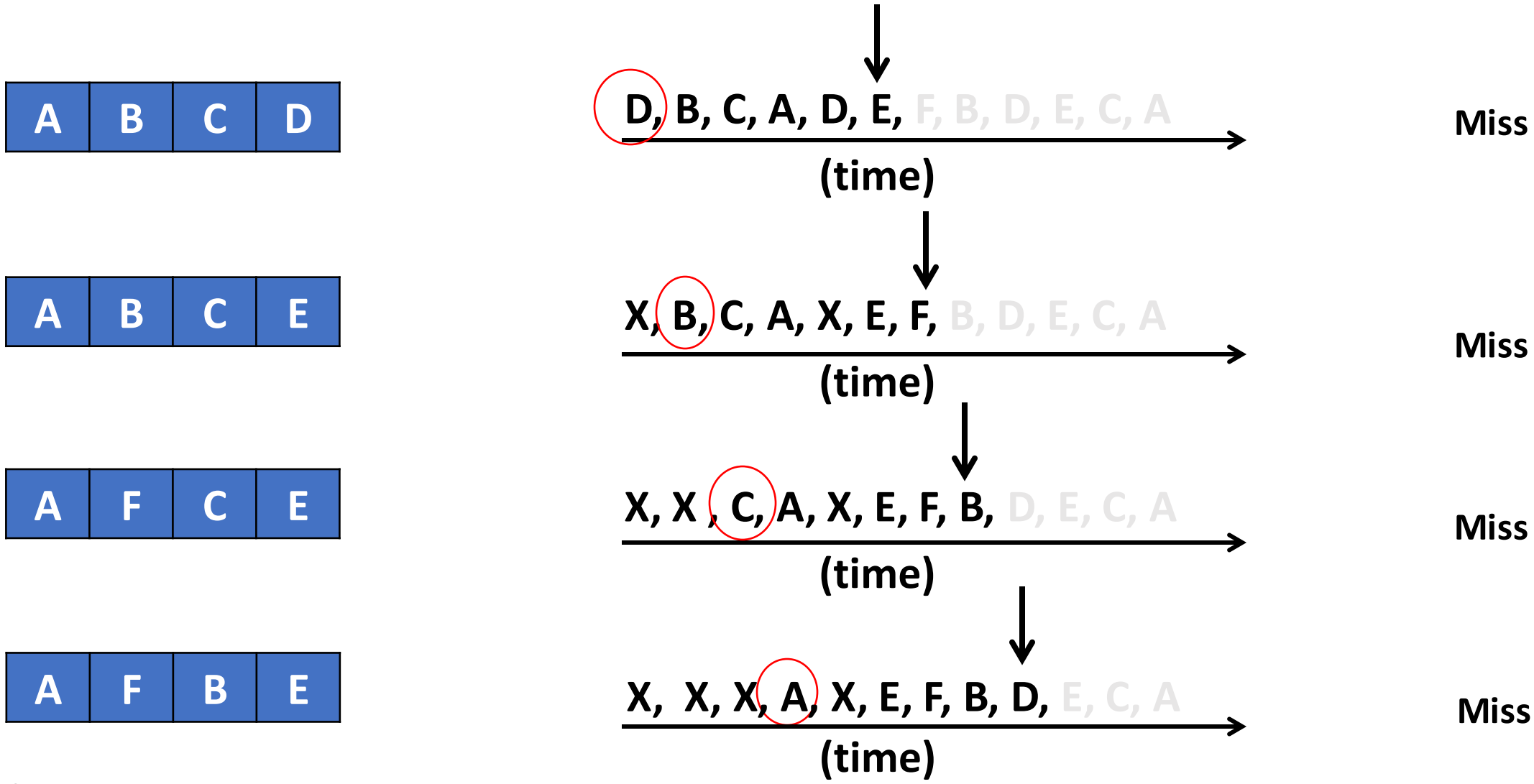
Least Recently Used (LRU) Example



25% Hit rate

Caching decision based on the observation of past (recent) lookups (future lookups are not known)

First-in-first-out Example



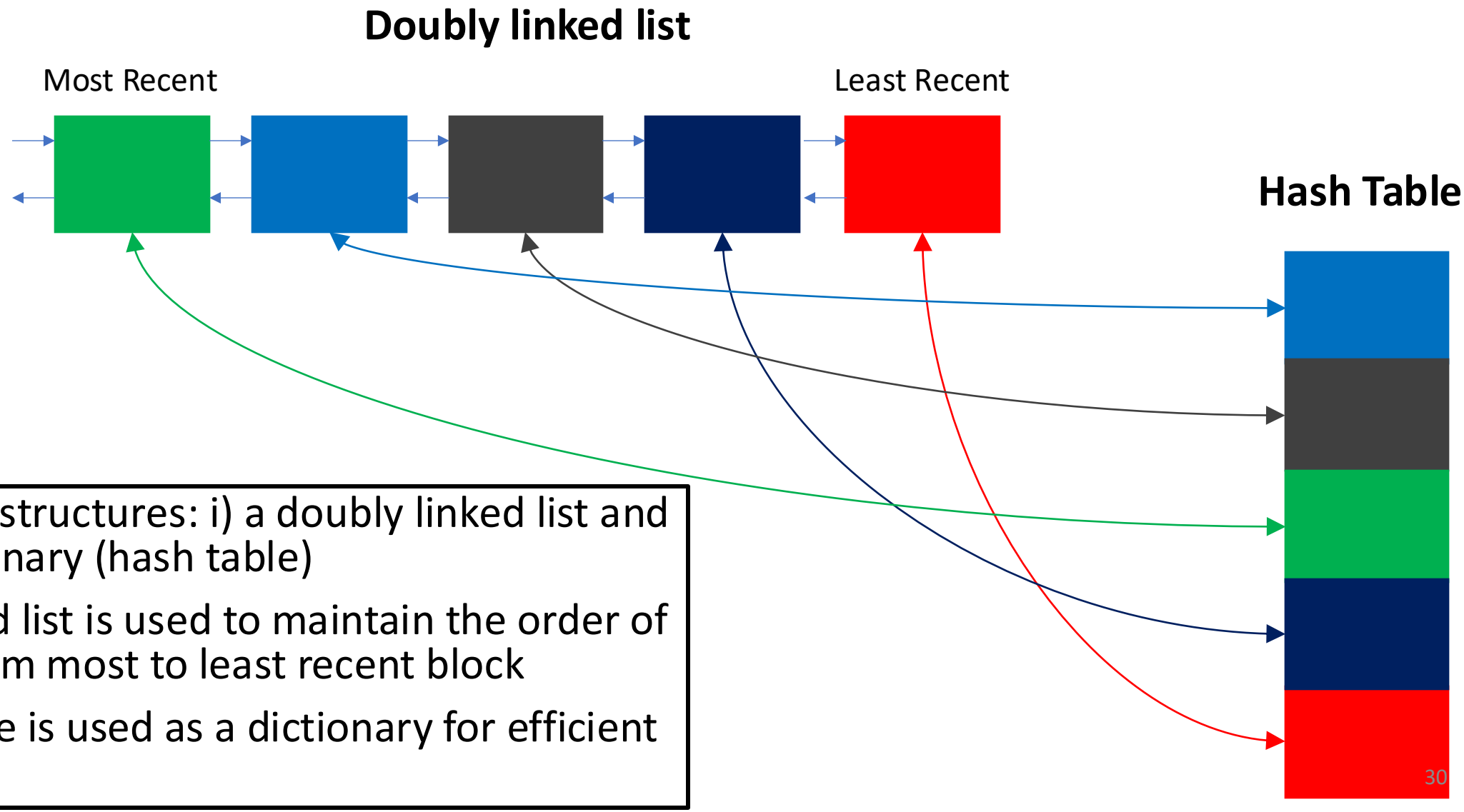
0% Hit rate

Caching decision based on the observation of past (recent) lookups (future lookups are not known)

First-in-first-out (FIFO)

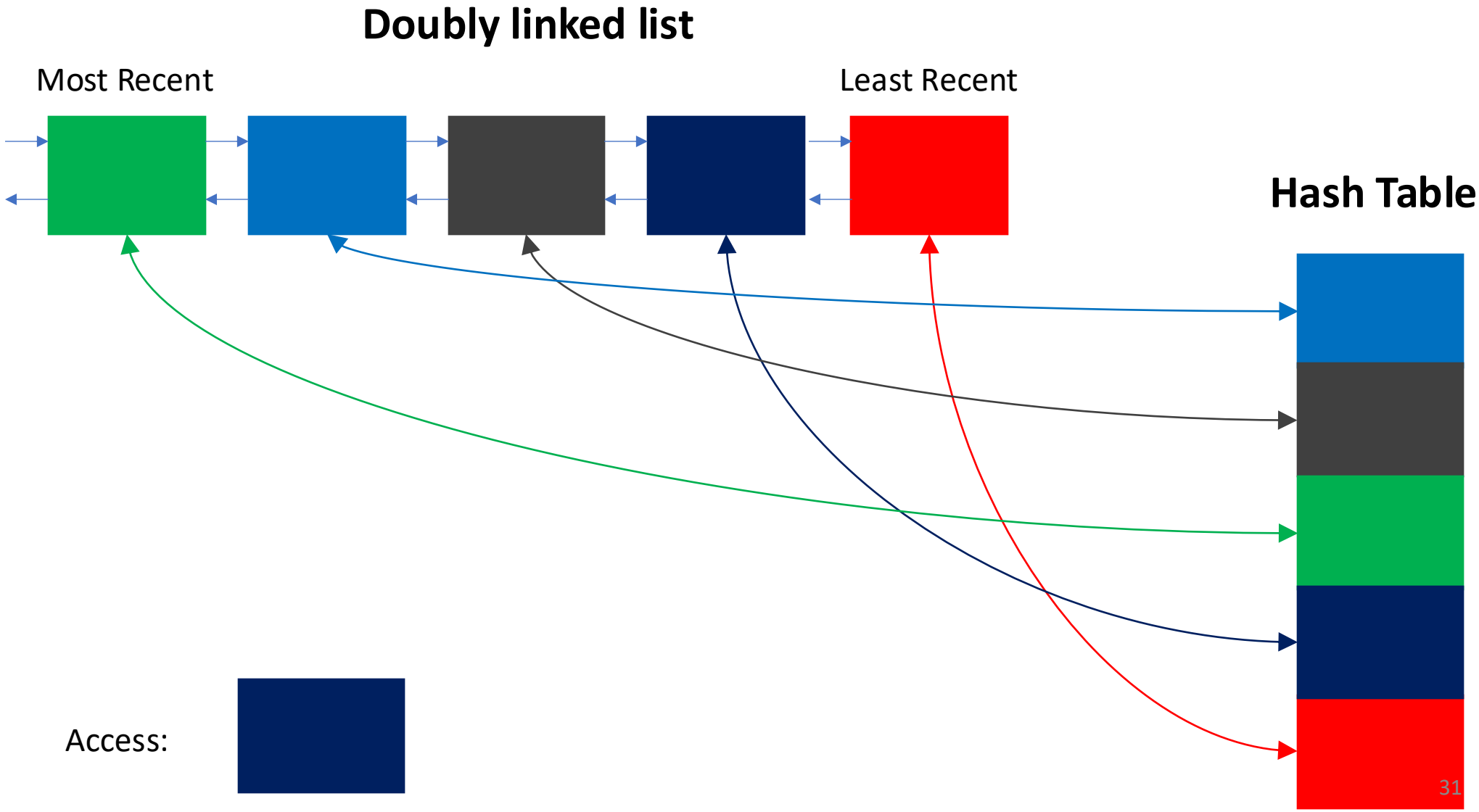
- **Belady's anomaly** occurs when increasing the size of a cache leads to an increase in the number of cache misses for certain access patterns.
- FIFO replacement policy can suffer from Belady's anomaly
  - **You can find an access pattern that results in a higher cache hit rate with a cache with size 3 than a cache with size 4**
- It has been proven that LRU does **not** have this anomaly
- Exercise: Consider the following access pattern using FIFO: A, B, C, D, A, B, E, A, B, C, D, E with cache sizes 3 and 4.

LRU Implementation

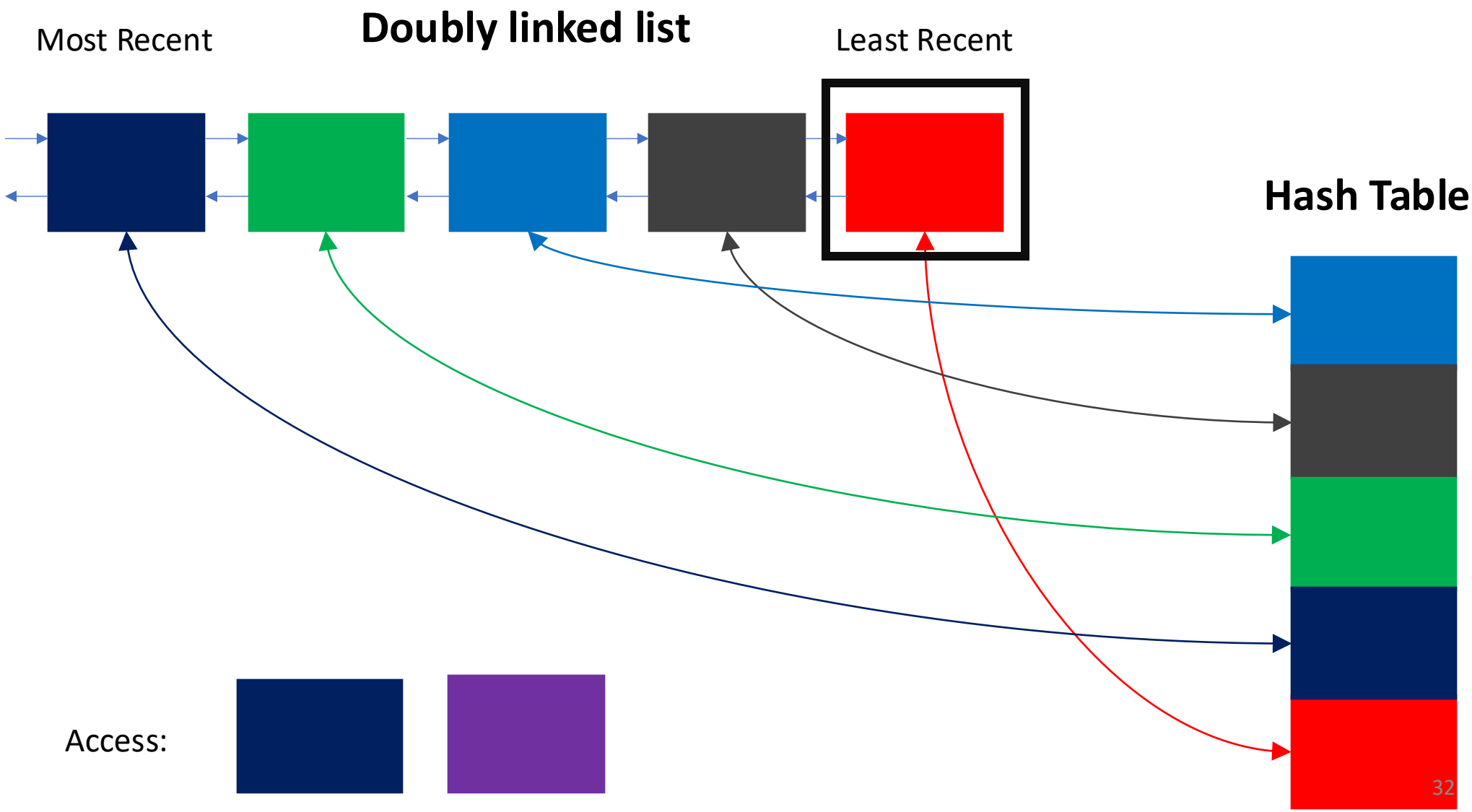


- Two data structures: i) a doubly linked list and ii) a dictionary (hash table)
- The linked list is used to maintain the order of blocks from most to least recent block
- Hash table is used as a dictionary for efficient lookups

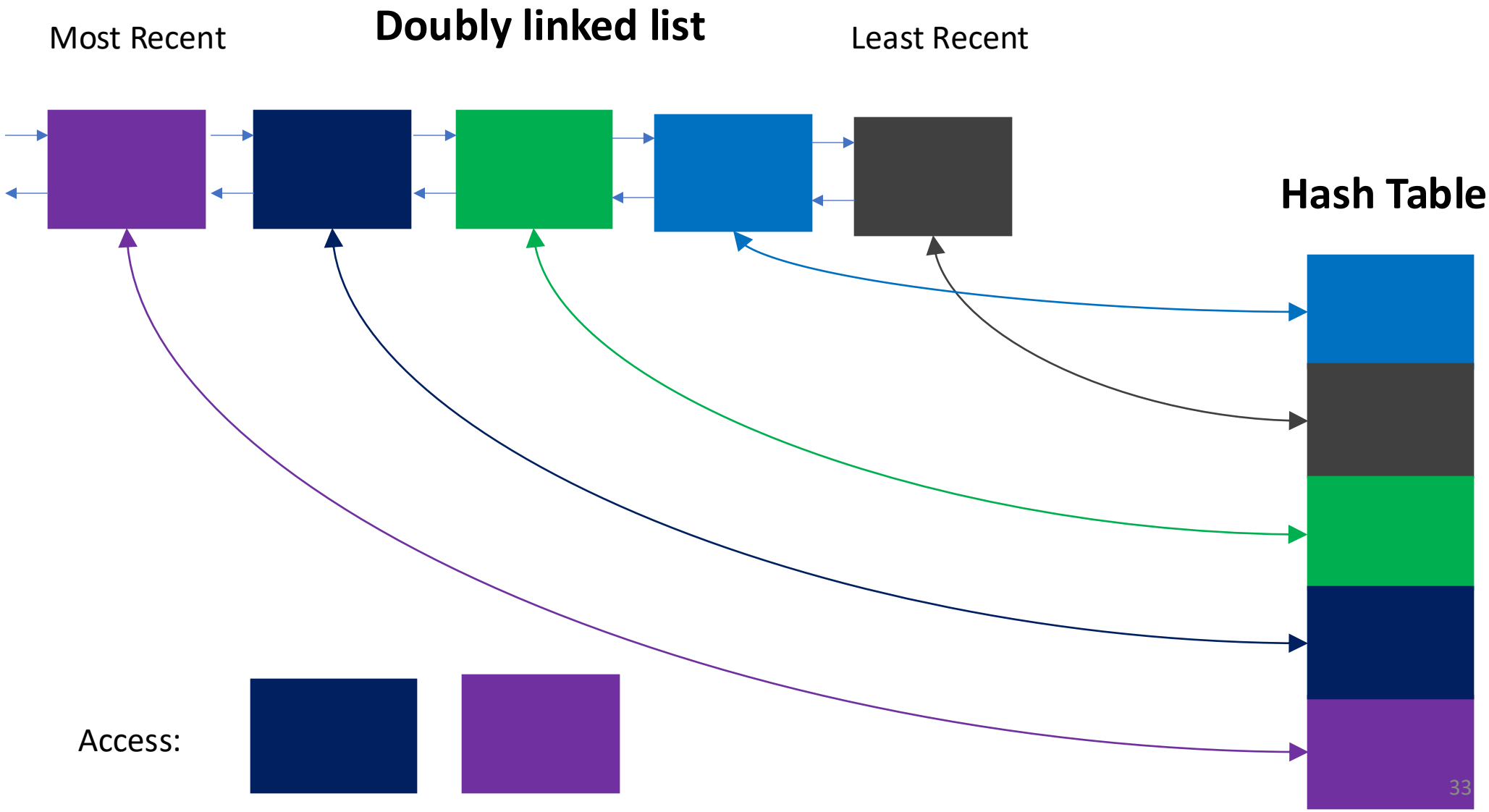
LRU Implementation



LRU Implementation



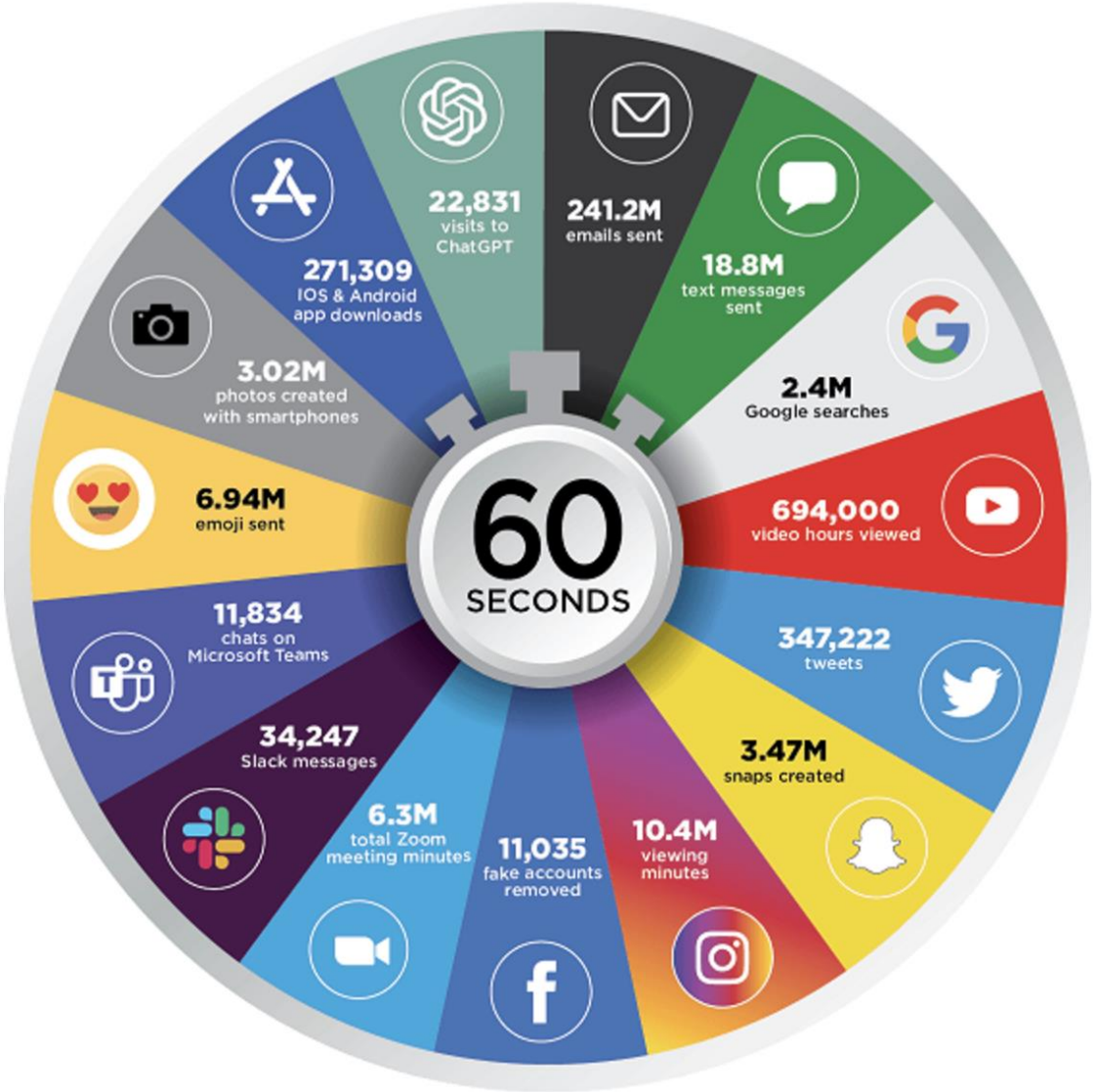
LRU Implementation



Outline

- Caching Principle
  - Data Storage Technologies
  - Memory Hierarchy and Locality of Reference
- Basic Cache Concepts
  - Cache Replacement Policies
- **Caching in the Internet**
  - Server-side caching
  - Caching at Distributed Servers
  - Distributed caching in the network

Very time sensitive



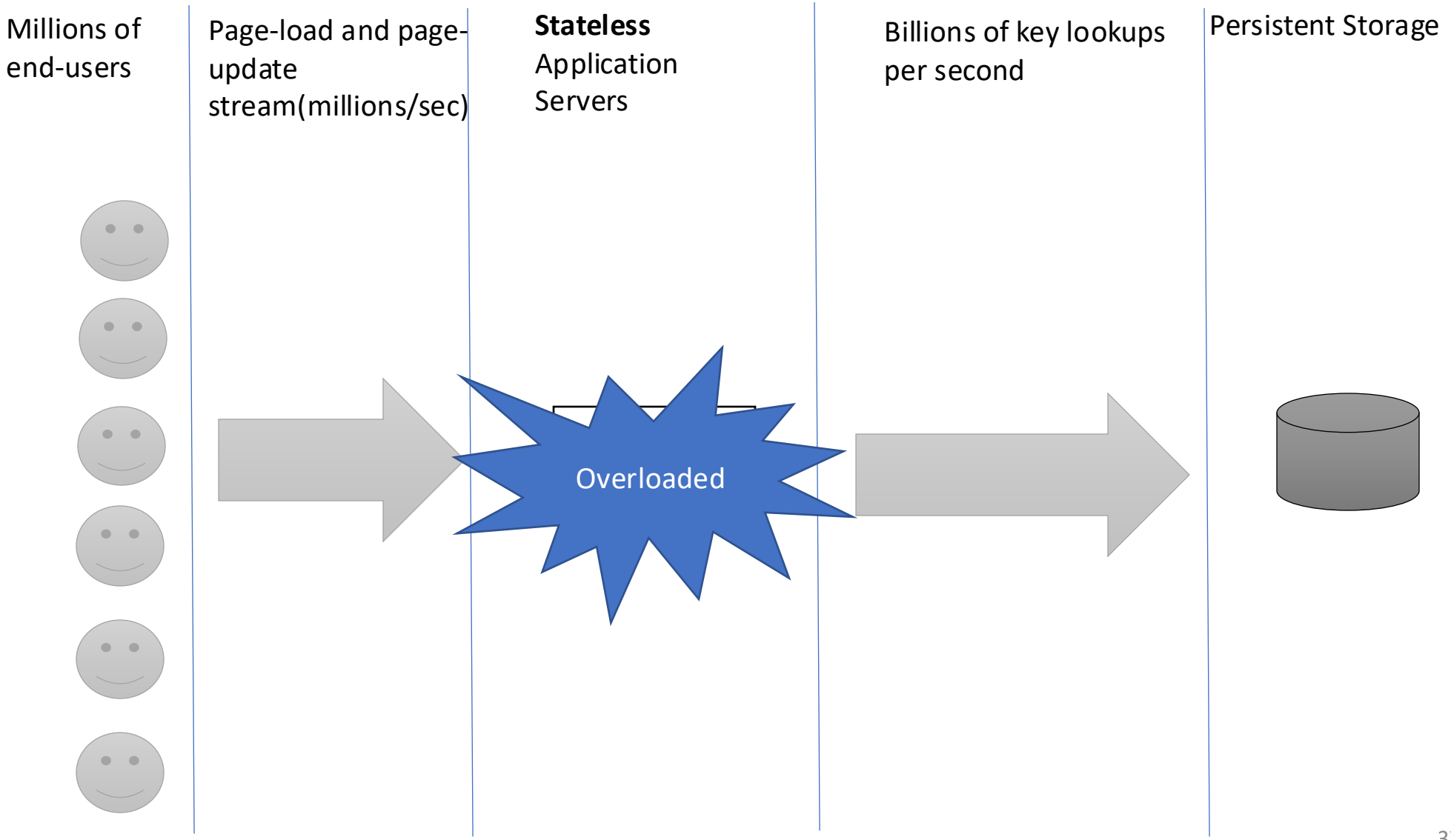
Huge amount of data

- Many systems deployed in the Internet require access to data at scale:
  - **at billions of requests per second rate**
  - **low latency response**
  - **high availability**
- Fast memory is expensive
- How do you achieve at-scale storage with reasonable cost?
  - Caching with storage hierarchy
  - Load-balancing

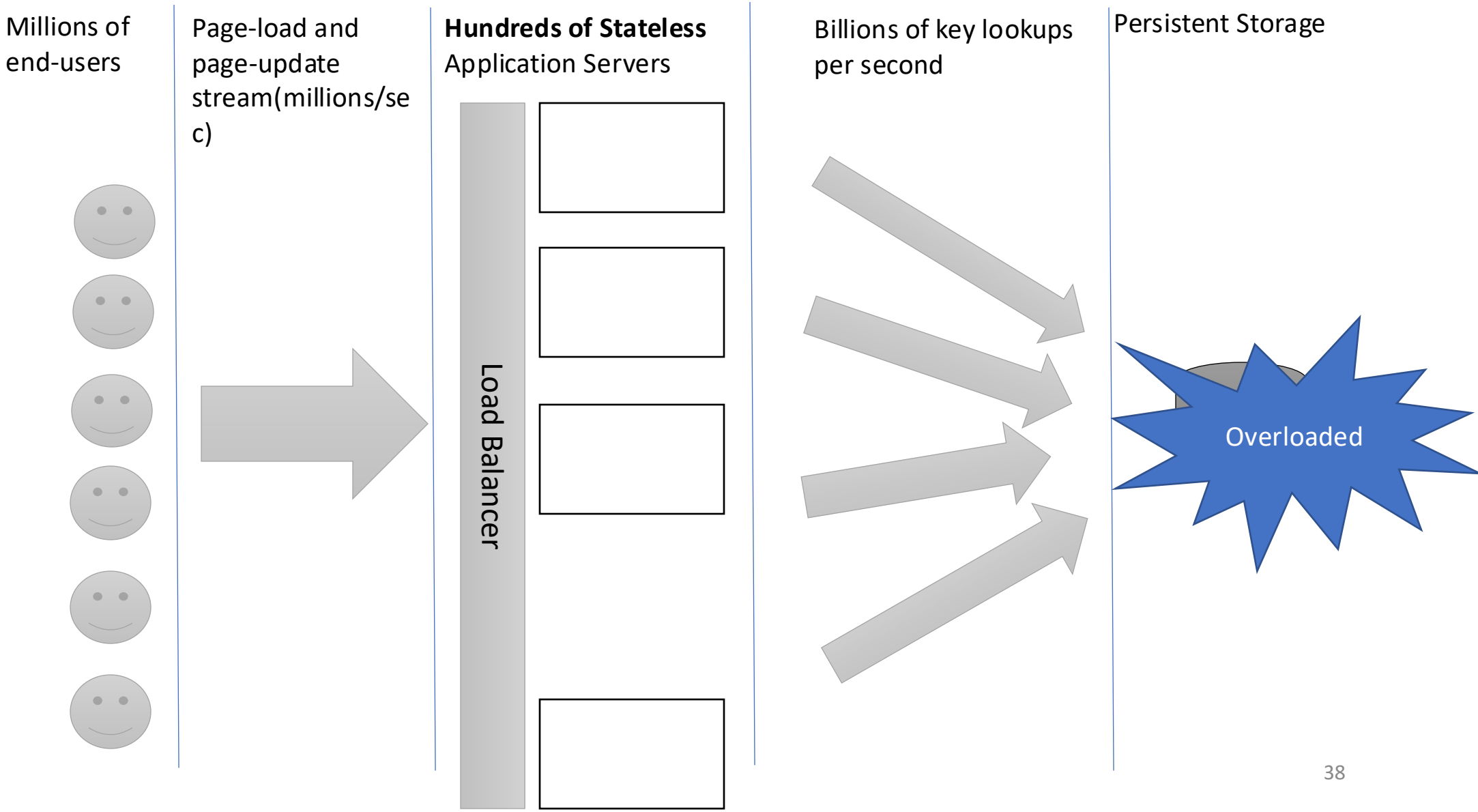
Storage at Scale

- **Globally distributed storage with caching:**
  - Caching is important for **cost-effective** and **scalable** storage.
  - Large-scale systems (e.g., Facebook, Google) require caching to reduce latency and improve performance.
- **Key Requirements:**
  - **Replication and Redundancy**: Ensure data availability and fault tolerance.
  - Handle **failures gracefully** and ensure **load balancing** for high availability.
- **Goals of Server-Side Caching:**
  - **Near real-time communication**: Provide low-latency access to cached data for millions of users worldwide
  - **Manage Popular Shared Content**: Enable rapid access and updates to hot (frequently accessed) content like trending posts or shared resources
  - **Massive Scalability**: Handle millions of requests per second while maintaining low response times

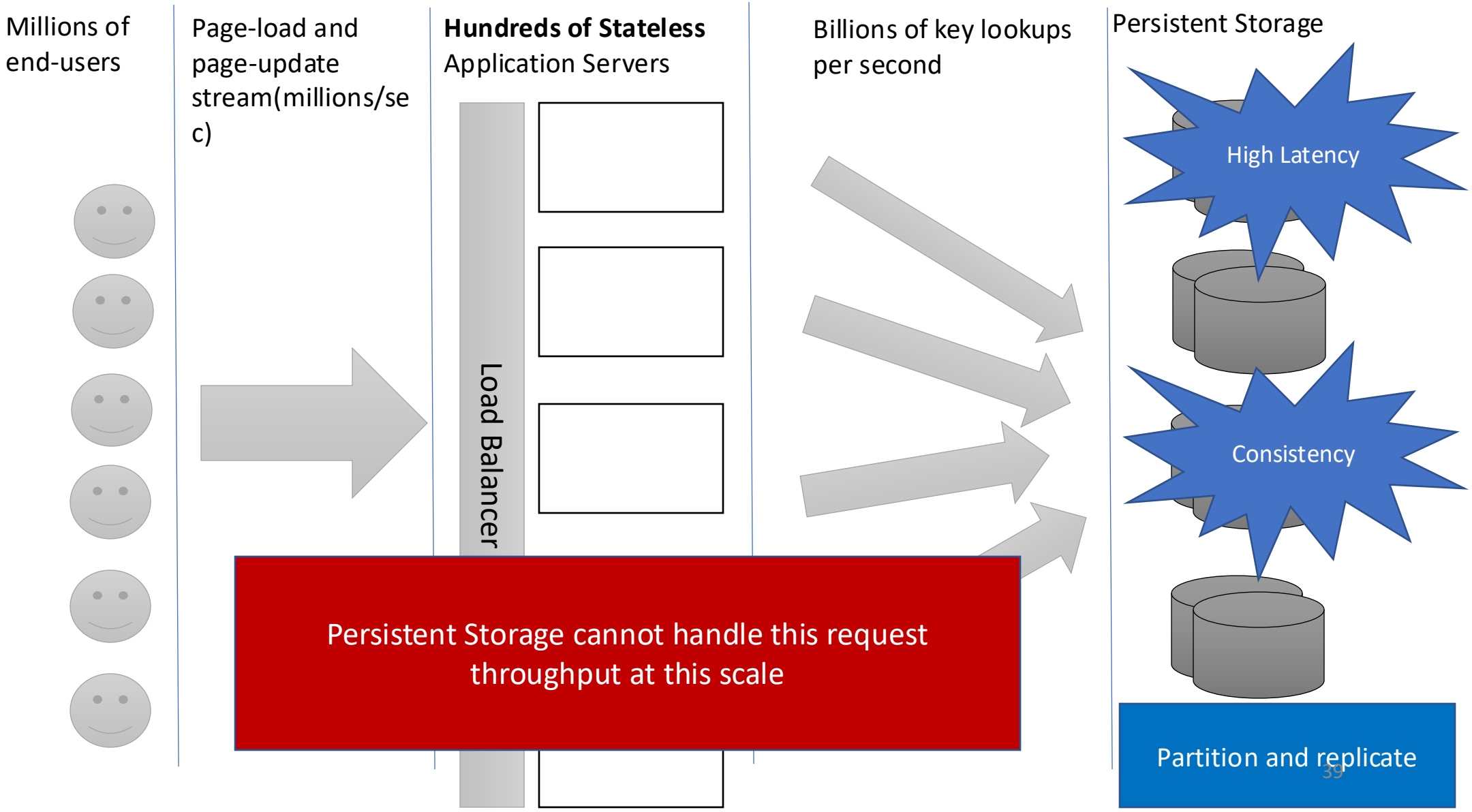
Server-side Caching



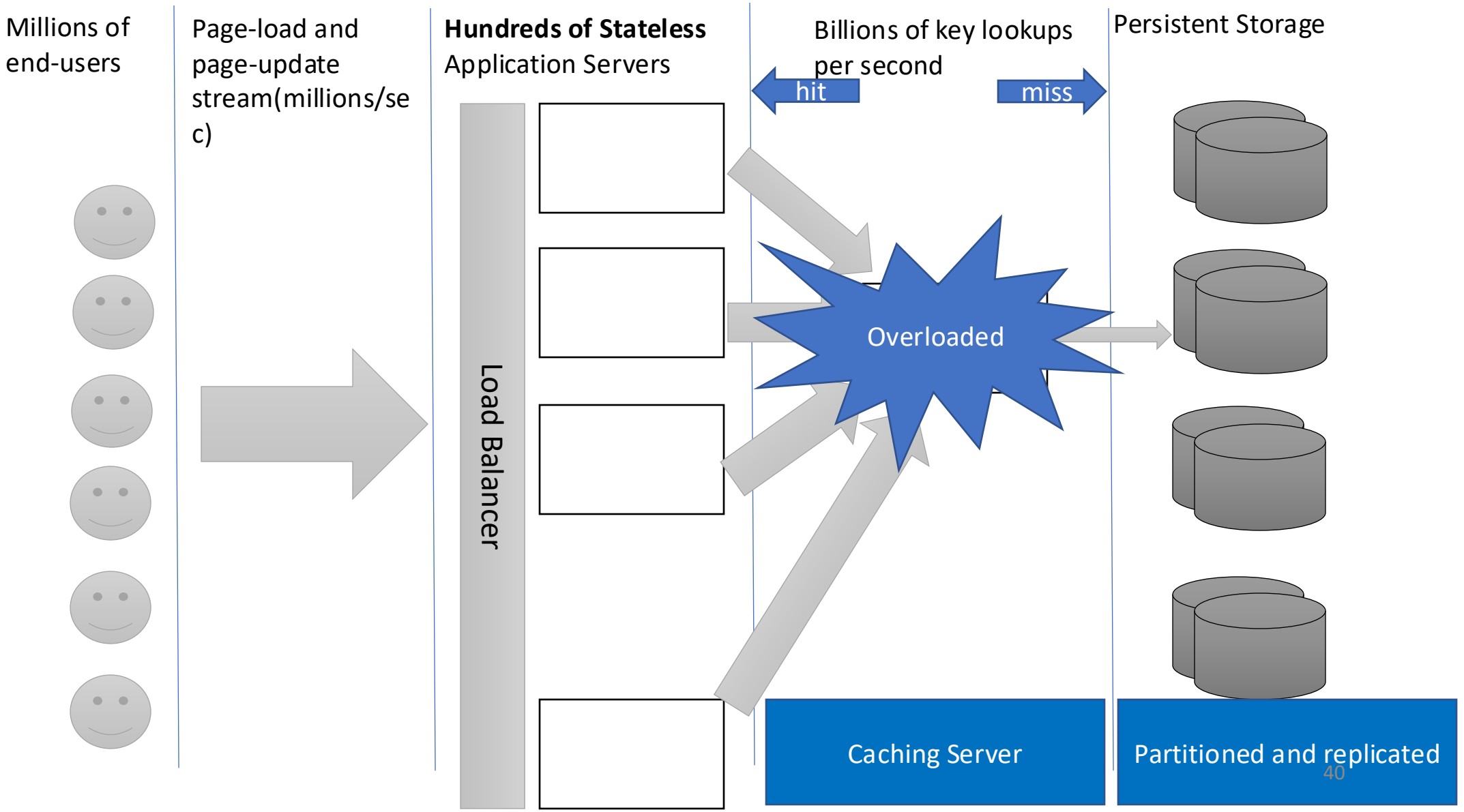
Server-side Caching



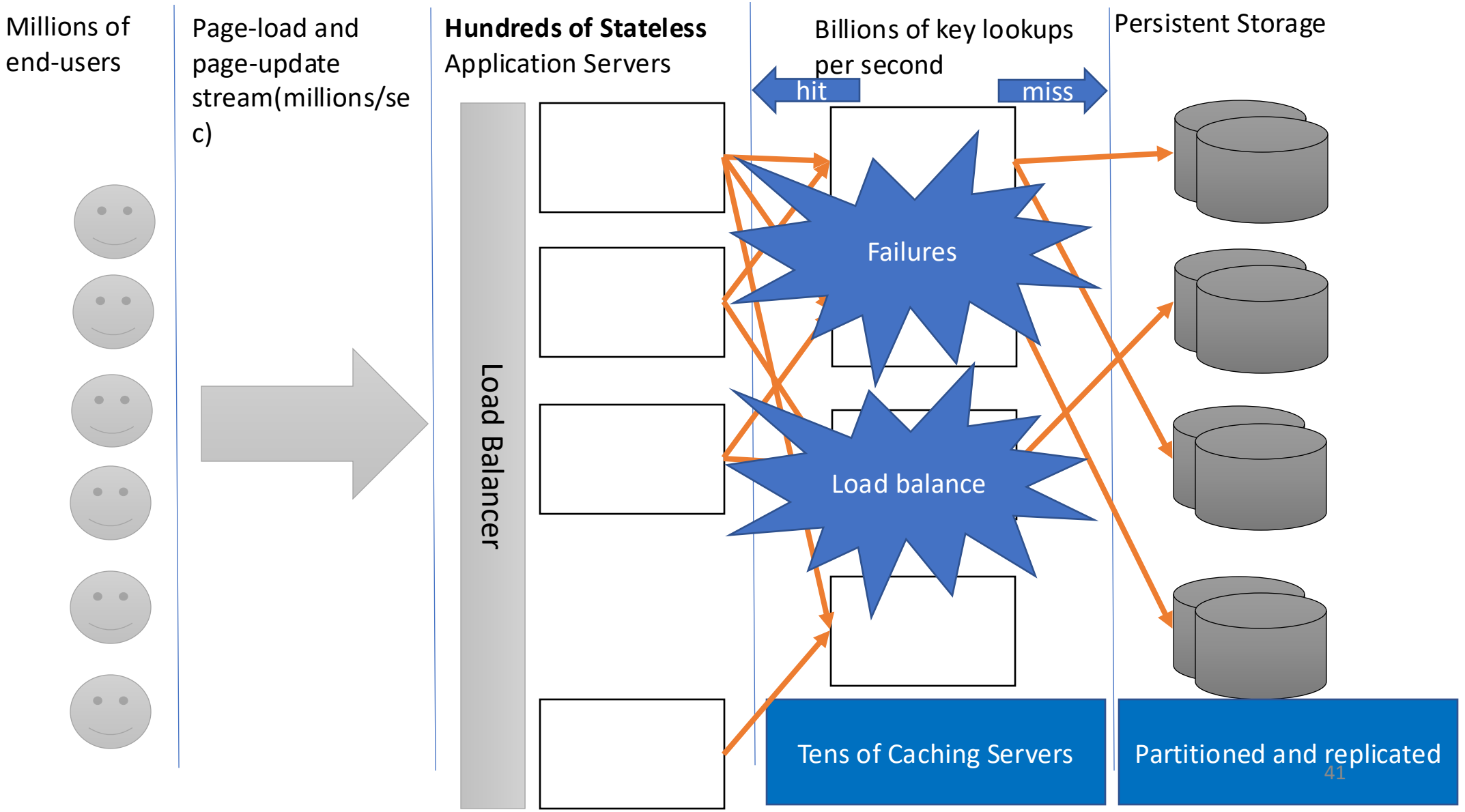
Server-side Caching



Server-side Caching



Server-side Caching



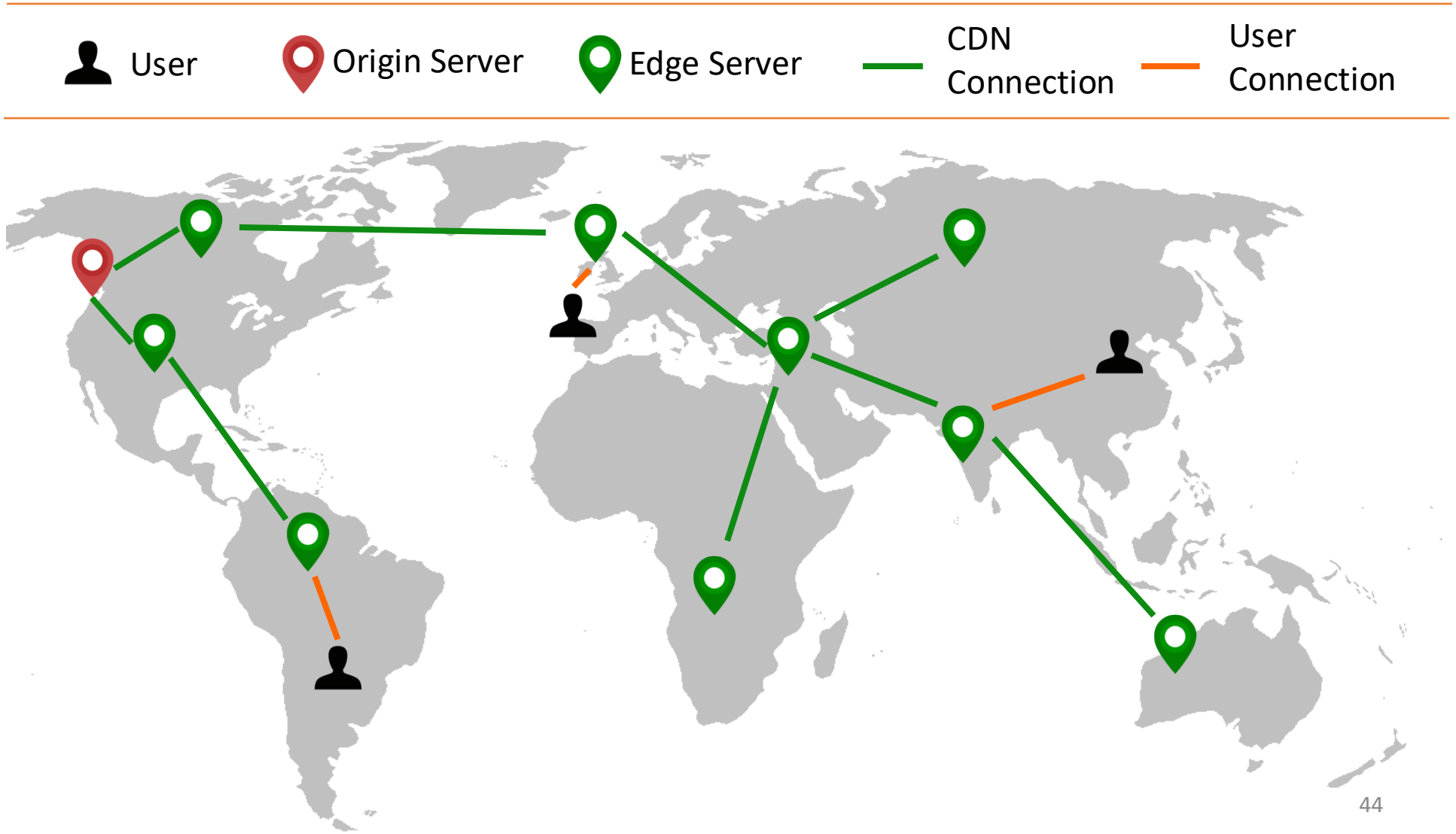
Outline

- Caching Principle
  - Data Storage Technologies
  - Example: Memory Hierarchy and Locality of Reference
- Basic Cache Concepts
  - Cache Replacement Policies
- **Caching in the Internet**
  - Server-side caching
  - Caching at Distributed Servers

Content Distribution Networks (CDN) Basics

- A distribution system on the Internet that accelerates the delivery of web pages, video and other content to users.
- Motivation scenario: stream video to 100,000+ users simultaneously.
- You could use a single large “mega-server”
  - Single point of failure
  - Point of network congestion
  - Long path to distant clients
  - Multiple copies of video sent over outgoing link
- This solution **doesn’t** work in practice
- CDNs solution: replicate the content provider’s data in ”edge servers”

Content Distribution Networks (CDN) Basics



Motivational Scenario

Streaming video to 100,000+ simultaneous users

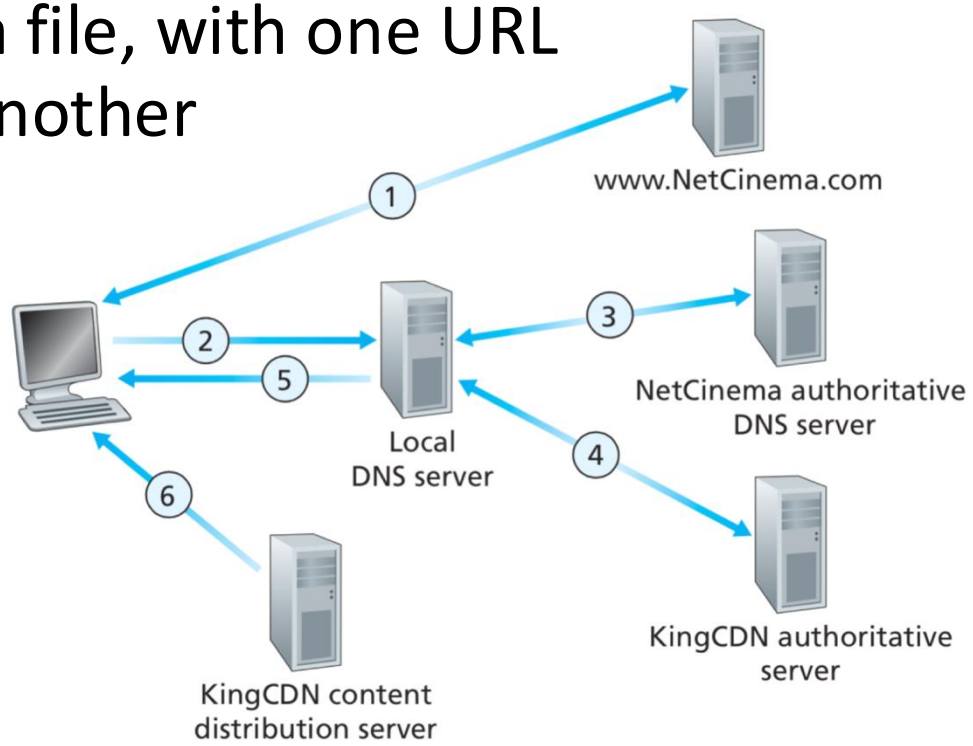
- Working Web solution: store/serve many copies of video at multiple geographically distributed sites (CDN)
- Two strategies:
  - Push CDN servers deep into many access networks
    - Close to users
    - Used by Akamai, 1700 locations
  - Place larger clusters at key points in the network near internet exchanges
    - Used by Limelight

CDN: simple content access scenario

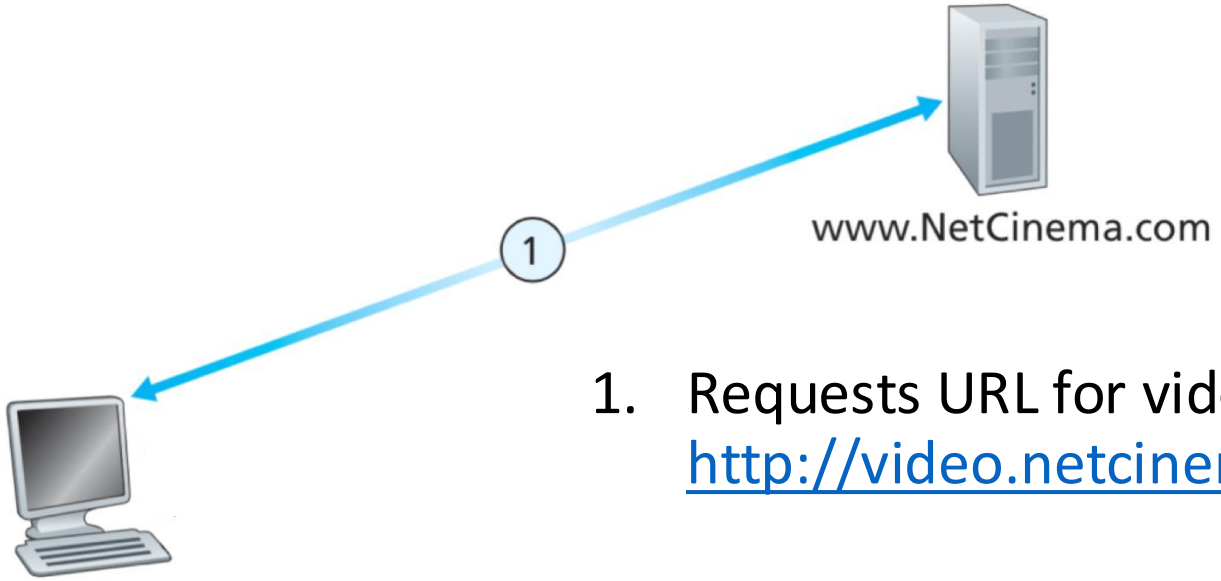
- A CDN has to be able to tell clients where to find resources
- A client will request a file, with one URL but retrieve it from another

http://video.netcinema.com/6Y7B23V

http://KingCDN.com/NetC6Y7B23V

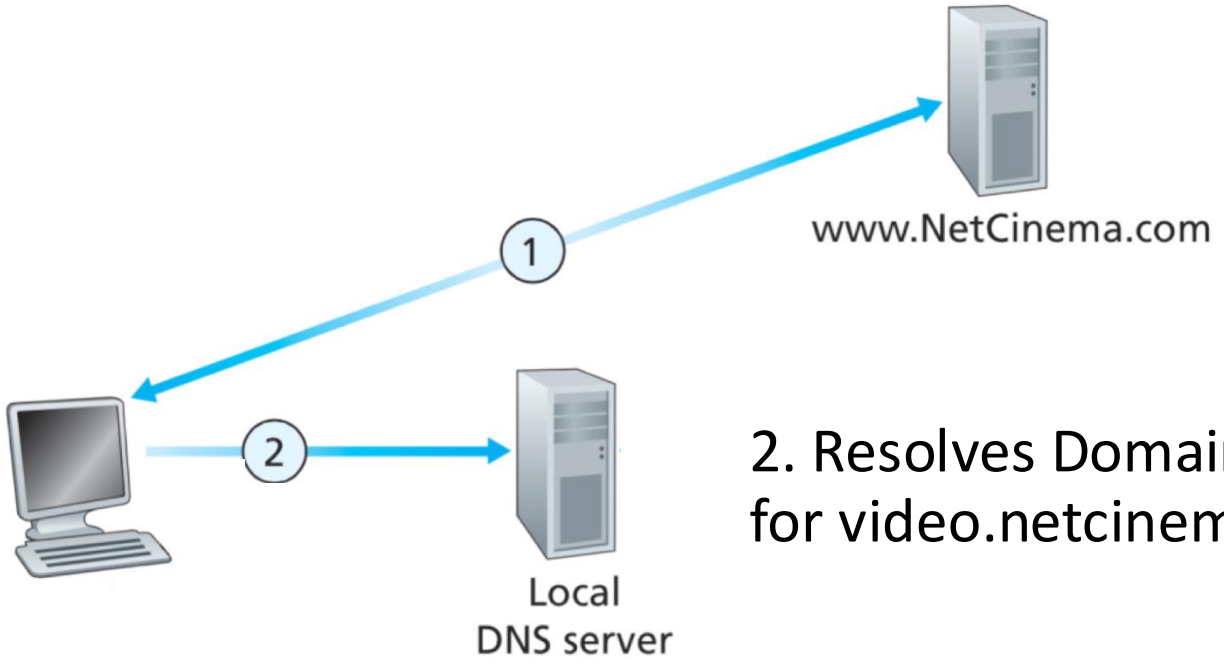


CDN: Simple content access scenario



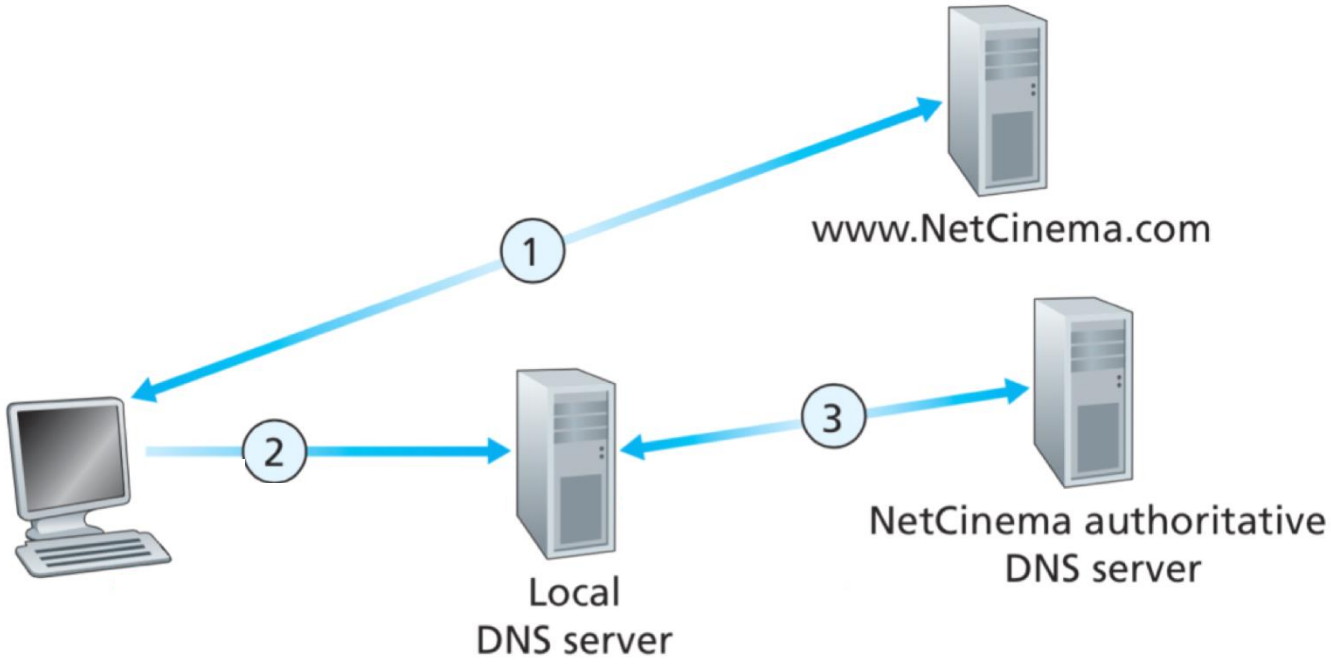
1. Requests URL for video http://video.netcinema.com/6Y7B23V

CDN: Simple content access scenario



2. Resolves Domain name via local DNS server for video.netcinema.com

CDN: Simple content access scenario



3. NetCinema DNS server returns the Domain name **KingCDN.com**

Summary (I)

- **Memory Devices and CPU-Memory Access Gap**:
  - Memory devices (DRAM, SRAM, SSD, etc.) vary in **speed, cost, and size**.
  - The **CPU-memory speed gap** is a growing challenge, making effective memory management essential.
- **Memory Hierarchy and Locality**:
  - The memory hierarchy exploits **temporal** and **spatial locality** to bridge the gap between fast but small caches and slow but large memory/storage.
  - **Locality Principle**:
    - **Temporal Locality**: Reuse recently accessed data.
    - **Spatial Locality**: Access data near recently accessed addresses.

Summary (II)

- **Caching Algorithms**:
  - Explored caching strategies, including:
    - **LRU (Least Recently Used)**: Tracks and evicts the least recently accessed data.
    - **FIFO (First-In-First-Out)**: Evicts the oldest data.
    - **Bélády's Anomaly**: Explained how increasing cache size can sometimes worsen performance (with FIFO).
  - Practical example: **Implementing an LRU cache**.
- **Server-Side Caching**:
  - Discussed caching in large-scale distributed systems (e.g., **Content Delivery Networks (CDNs)**):
    - Improves performance and scalability.
    - Reduces latency for millions of users.
- **Caching on the Internet**:

  Distributed Caching: Used by large-scale applications like Facebook, Google, and CDNs to handle massive workloads efficiently.

Lancaster University

Thanks for listening!
Any questions?