DSECL ZG 522: Big Data Systems

Session 1.2 : Locality of Reference

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Janardhanan PS
janardhanan.ps@wilp.bits-pilani.ac.in

Locality of Reference (LoR)

Spatial and temporal locality principles for processing and storage

Context

- Big Data Systems need transportation of large volumes of data
- Same data may be accessed repeatedly during distributed processing
- Data may be accessed sequentially in consecutive accesses
- Is there a way to reduce latency of data transfer using locality of reference in the data and request origin ?

Levels of storage

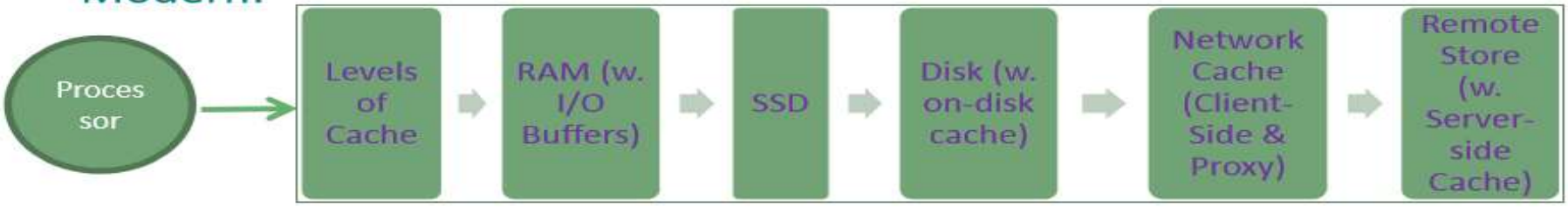
Data Location – Memory vs Storage vs Network

- Computational Data is stored in primary memory aka memory
- Persistent Data is stored in secondary memory aka Storage
- Remote data access from another computer's memory or storage is done over network

Memory hierarchy – motivation

- Memory hierarchy amortizes cost in computer architecture:
  - ✓ fast (and therefore costly) but small-sized memory to
  - ✓ large-sized but slow (and therefore cheap) memory
- Classic:

- Modern:



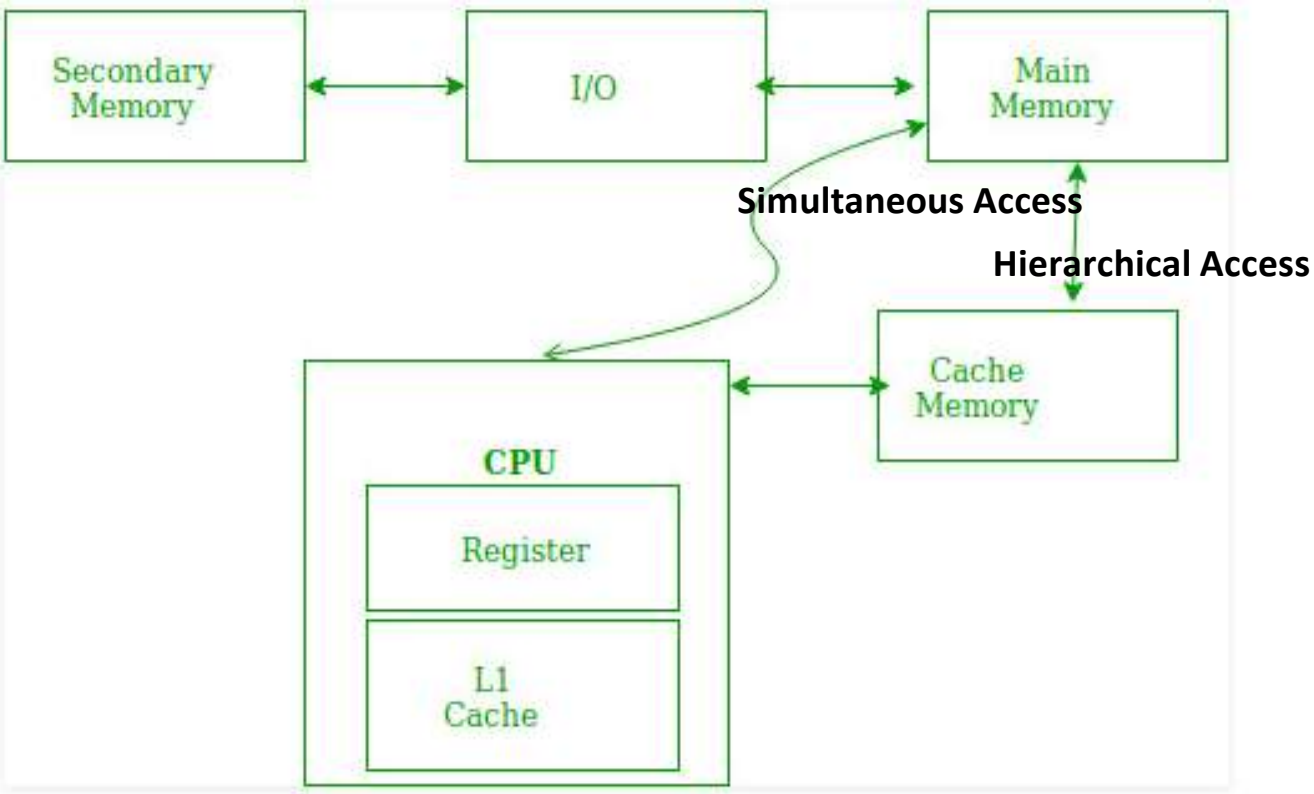
Cost of access: Memory vs. Storage vs. Network

Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 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 disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

Reference: Designs, Lessons and Advice from Building Large Distributed Systems

Memory hierarchy – access



Data reference empirical evidence

- Programs tend to reuse data and instructions they have used recently.
- 90/10 rule comes from empirical observation:
  "A program spends 90% of its time in 10% of its code“
- Implication:
  - ✓ Can predict with reasonable accuracy what instructions and data a program will use in near future based on the recent past

Cache performance – Hit ratio

In Hierarchical access, cache performance is decided by the Hit Ratio

- Cache hit
  - ✓ When CPU refers to memory and find the data or instruction within the Cache Memory
- Cache miss
  - ✓ If the desired data or instruction is not found in the cache memory
    (Cache gets populated with data from memory eventually)

$$\text{Hit} + \text{Miss} = \text{Total CPU Reference}$$

$$\text{Hit Ratio } h = \text{Hit} / (\text{Hit} + \text{Miss})$$

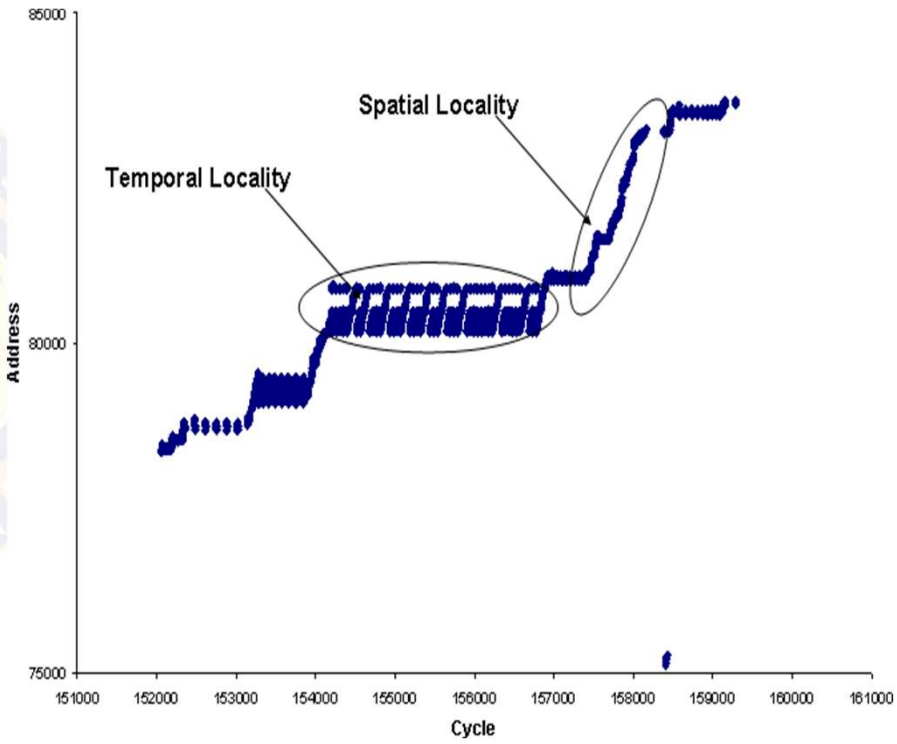
- One can generalize this to any form of cache
  - ✓Database caching using in-memory DBs (a problem follows)
  - ✓Movie request from user to nearest Netflix content cache

Difference between Simultaneous and Hierarchical Access Memory

Simultaneous Access Memory Organisation	Hierarchical Access Memory Organisation
In this organisation, CPU is directly connected to all the levels of Memory.	In this organisation, CPU is always directly connected to L1 i.e. Level-1 Memory only.
CPU accesses the data from all levels of Memory simultaneously.	CPU always accesses the data from Level-1 Memory.
For any "miss" encountered in L1 memory, CPU can directly access data from higher memory levels (i.e. L2, L3, …..Ln).	For any "miss" encountered in L1 memory, CPU cannot directly access data from higher memory levels(i.e. L2, L3, …..Ln). First the desired data will be transferred from lower memory levels to L1 memory. Only then it can be accessed by the CPU.
If H1 and H2 are the Hit Ratios and T1 and T2 are the access time of L1 and L2 memory levels respectively then the Average Memory Access Time can be calculated as: $T=(H1*T1)+((1-H1)*H2*T2$	If H1 is the Hit Ratio and T1 and T2 are the access time of L1 and L2 memory levels respectively then the Average Memory Access Time can be calculated as: **Tavg = (H1*T1) + (1-H1) * (T1 + T2)**

Principle of Locality of Reference (LoR)

1. The locus of data access (and hence that of memory references) is small at any point during program execution
2. It is the tendency of a processor to access the same set of memory locations repetitively over a short period of time
3. Locality is a type of predictable behavior that occurs in computer systems
4. Systems that exhibit strong locality of reference are great candidates for performance optimization through use of techniques such as caching, prefetching for memory and advanced branch predictors at the pipelining stage of a processor core.



Spatial Vs Temporal Locality

S.No.	Spatial Locality	Temporal Locality
1.	In Spatial Locality, nearby instructions to recently executed instruction are likely to be executed soon.	In Temporal Locality, a recently executed instruction is likely to be executed again very soon.
2.	It refers to the tendency of execution which involve consecutive memory locations .	It refers to the tendency of execution where memory location that have been used recently have an access in future.
3.	It is also known as locality in space.	It is also known as locality in time.
4.	It only refers to data item which are close together in memory.	It repeatedly refers to same data in short time span.
6.	Example : Data elements **accessed in array** (where each time different (or just next) element is being accessing).	Example : Data elements **accessed in loops** (where same data elements are accessed multiple times).

Locality of Reference - Temporal locality

- Data that is accessed (at a point in program execution) is likely to be accessed again in the near future:

  i.e. data is likely to be repeatedly accessed in a short span of time during execution

- Examples
  1. Instructions in the body of a loop
  2. Parameters / Local variables of a function / procedure
  3. Data (or a variable) that is computed iteratively
     - e.g. a cumulative sum or product
  4. Another user in Netflix in same region will watch the same episode soon
  5. A recent social media post will be viewed soon by other users

```
void swap(int x,
int y)
{
    t = x;
    x = y;
    y = t;
}
```

Locality of Reference - Spatial locality

- Data accessed (at a point in program execution) is likely located adjacent to data that is to be accessed in near future:
  - ✓ data accessed in a short span during execution is likely to be within a small region (in memory)

```
int sum_array_rows(int marks[8]){
        int i, sum = 0;
        for (i = 0; i < 8; i++)
                sum = sum + marks[i];
        return sum;
}
```

- Examples
  1. A linear sequence of instructions
  2. Elements of an Array (accessed sequentially)
  3. Another user's query will reuse part of the data file brought in for current user's query

Access time of memories (Hierarchical Access)

- Average access time of any memory system consists of two levels:
  - ✓ Cache Memory
  - ✓ Main Memory

- If Tc is time to access cache memory and Tm is the time to access main memory and h is the cache hit ratio, then

$$Tavg = h * Tc + (1-h) * (Tc + Tm)$$

Where

Tavg = Average time to access memory

Typical Calculations

Problem: In a 2-level cache hierarchy, if the top level has an access time of 10ns and the bottom level has an access time of 60ns, what is the hit rate on the top level required to give an average access time of 16ns?

Answer: We have been given Cache Access time (top level) = 10ns and Main memory access time (bottom level)= 60ns and Average memory access time = 16ns. We will apply the concept of hierarchical access(because in question it was written 2-level hierarchy) for calculating the memory access time. Average memory access time = Hit ratio * Cache memory access time + (1 – Hit ratio) * (Cache Memory access time + Main memory access time)

16ns = 10 * Hit ratio + (1 – Hit ratio) * (10ns + 60ns)
16ns-70ns = -60 * Hit ratio
Hit ratio = 54 /60 = 9/10 = **0.9** (Ans.)

Tavg = 16ns with HitRatio = 0.9
Tavg = 22ns with HitRatio = 0.8

If Time required for Block reading of Memory is also taken into consideration

Average Memory Access Time = Hit ratio * Cache Memory Access Time +
(1 – Hit ratio) * (Time required to access a block of main memory + Cache memory block update time)

where:

Time required to access a main memory block = block size * Time required to access main memory,

Cache memory block update time = cache block size * Time required to access cache memory

https://www.geeksforgeeks.org/simultaneous-and-hierarchical-cache-accesses/

Memory hierarchy and locality of reference (1)

- A memory hierarchy is effective only due to locality exhibited by programs (and the data they access)
- Longer the range of execution time of the program, larger is the locus of data accesses
- Need to have cache replacement strategies because increasing cache size will also increase access time and increase cost
  - ✓ Least Recently Used (LRU) is a typical cache entry replacement strategy
    - This is an example of temporal locality

Memory hierarchy and locality of reference (2)

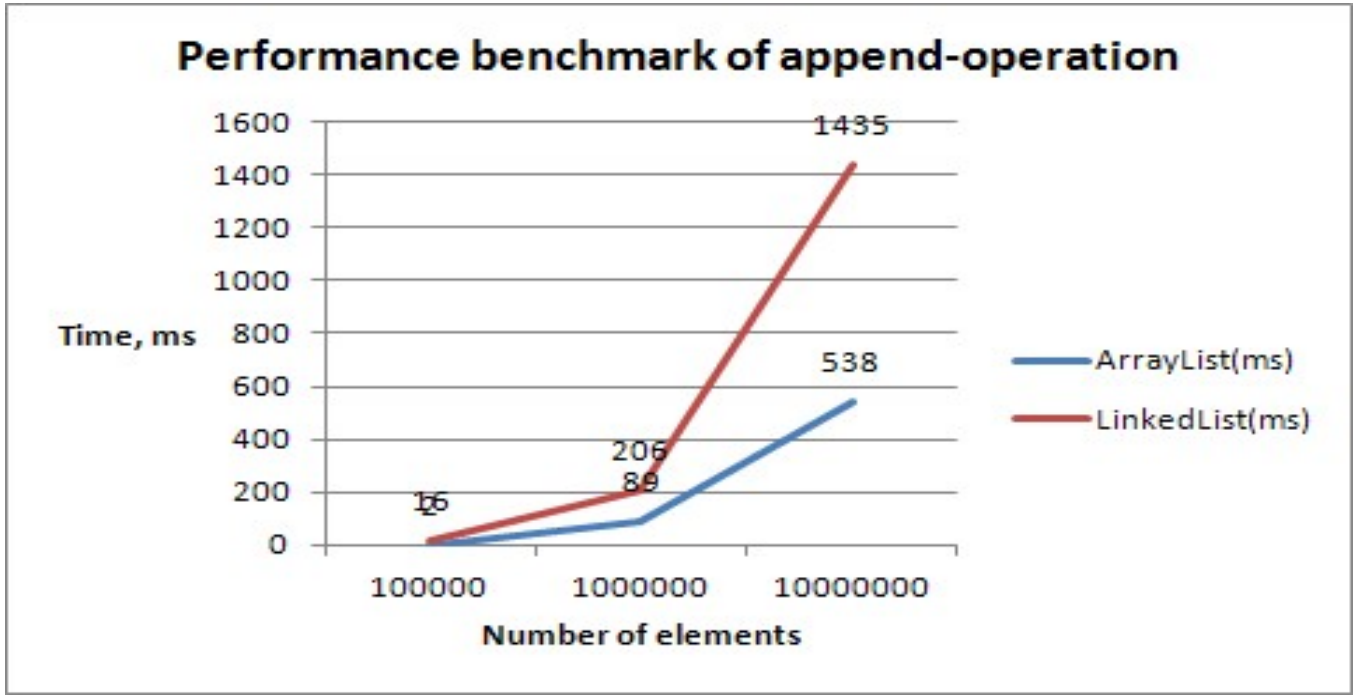
- LOR leads to memory hierarchy at two main interface levels:
  - ✓ Processor - Main memory
    - Introduction of caches
    - Unit of data transfer is blocks (# of cache lines)
  - ✓ Main memory - Secondary memory (storage)
    - Virtual memory (paging systems)
    - Unit of data transfer is pages (page size is 4 or 8 KB)
- Fetching larger chunks of data enables spatial locality

LoR and data structure choices

- In Java, the following 2 classes are available in util package
  - ✓ LinkedList<Integer> QuizMarks – a dynamic list of objects
  - ✓ ArrayList<Integer> QuizMarks - a dynamic array of objects
  - ✓ Which is a better data structure to use for sequential access performance ?
- Test
  - ✓ Append a single element in a loop N times to the end of a LinkedList. Repeat with ArrayList.
  - ✓ Take average for 100 runs.
  - ✓ The time complexity of the operation on both collections is the same.
- Which one works faster ?

Performance of Array vs. Linked-List on Modern Computers

LoR and data structure choices (2)



- Analysis - sequential access in arrays is faster than on linked lists on many machines, because they have a very good spatial locality of reference and thus make good use of data caching.

Locality Example : Matrices addition

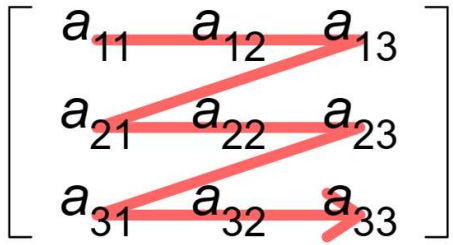
- Consider matrices and an operation such as the addition of two matrices:
- Elements M[i,j] may be accessed either row by row or column by column
- Row-major order and Column-major order are methods for storing multidimensional arrays in linear storage such as random access memory.
- Option 1: Add columns within each row (Row Major)

```
for (i=0; i<N; i++)
    for(j=0; j<N; j++)
        M3[i, j] = M2[i, j] + M1[i, j]
```

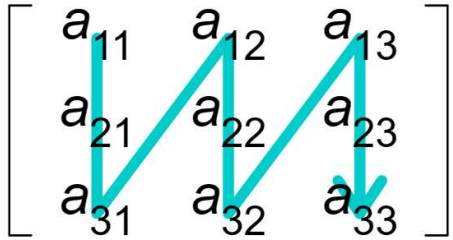
- Option 2: Add rows within each column (Column Major)

```
for (j=0; j<N; j++)
    for(i=0; i<N; i++)
        M3[i, j] = M2[i, j] + M1[i, j]
```

Row-major order



Column-major order

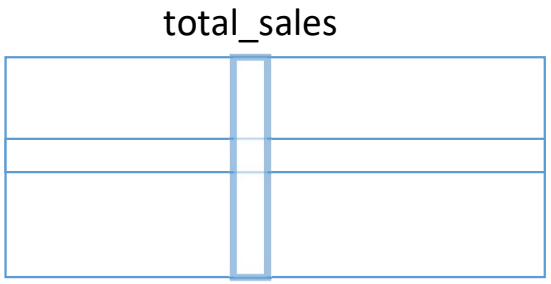


Both options have same time complexity.
Why is one faster ?

Big Data: Storage organisation matters

Example

- We need to build a prediction model of sales for various regions
- There are many attributes for a region and "total sales" is the metric used
- Suppose the database is "columnar" (Column major data organization)
  - ✓ Will exhibit high spatial locality and hit rate because reads will fetch consecutive blocks of total sales column.
  - ✓ Will improve speed of modelling logic
- Columnar storage is common in most Big Data Systems, NoSQL to run analysis and queries that focus on specific attributes at a time for searching, aggregating, modelling etc.



Bigdata: Distributed File System Cache

Advantages

- **Improved performance**: By reducing network traffic and minimizing disk access, file caching can significantly improve the performance of distributed file systems.
- **Reduced latency**: File caching can reduce latency by allowing files to be accessed more quickly without the need for network access or disk access.
- **Better resource utilization**: File caching allows frequently accessed files to be stored in memory or on local disks, reducing the need for network or disk access and improving resource utilization.

Disadvantages

- **Increased complexity**: File caching can add complexity to distributed file systems, requiring additional software and hardware to manage and maintain the cache.
- **Cache consistency issues**: Keeping the cache up-to-date can be a challenge, and inconsistencies between the cache and the actual file system can occur.
- **Increased memory usage**: File caching requires additional memory resources to store frequently accessed files, which can lead to increased memory usage on client machines and servers.

Overall, file caching is an important feature of distributed file systems that can improve performance and reduce latency. However, it also introduces some complexity and requires careful management to ensure cache consistency and efficient resource utilization.

File Cache in Hadoop HDFS (File localization)

- Copy small files from HDFS to local disks of worker nodes
- Saves network bandwidth during run, since these files are locally available
- These files get tagged as localized
- Hadoop NodeManager maintains a count of number of tasks using the localized files
- The file is selected for deletion only when there are no tasks using it
- The file gets deleted when the cache occupancy exceeds a specified limit
- Hadoop deletes localized files to make room for other files getting used
- Files selected for deletion using least recently used algorithm
- One can change the cache size by setting the property

  yarn.nodemanager.localizer.cache.target-size-mb (default 10240)

Typical Database Caching platforms

Caching involves storing frequently used data in a temporary memory to reduce the time required to retrieve it from the source again to

- Boost application performance by providing high-speed local and distributed in-memory data access.
- Optimize database usage by reducing the volume of read and write operations

- Redis
  - ✓ Redis is a popular in-memory key-value store that can be used as a caching solution.
  - ✓ Redis supports a variety of data structures, including strings, hashes, lists, sets, and sorted sets.
  - ✓ Redis is designed to be fast, reliable, and scalable, making it a popular choice for high-traffic applications.
- Memcached
  - ✓ Memcached is a distributed in-memory caching system that allows users to store and retrieve data across multiple servers.
  - ✓ Memcached supports a wide variety of programming languages, including PHP, Python, Ruby, and Java.
  - ✓ Memcached is fast, scalable and can handle a large number of requests per second.
  - ✓ Memcached is highly scalable and can be used in a distributed environment.
- Infinispan
  - ✓ An open-source, in-memory data grid platform primarily used as a distributed cache and key-value NoSQL data store
  - ✓ Can run in local & distributed mode, , its real power lies in distributed mode, where caches cluster together to form a large, shared memory heap
  - ✓ Significantly lower latency for read operations and offer higher throughput for write operations compared to traditional database systems

Problem: Speed up calculation with Database Caching

A healthcare data analytics application utilizes a Cloud-based PostgreSQL cluster. The analytics team conducts patient record analysis on the data stored on the Cloud instance. It takes approximately **45 seconds** per user query on average. You are tasked to implement a Redis based cache in the healthcare center's data center to expedite these queries, aiming for an average query latency of **15 seconds**. The Redis cache has an access latency of 7 seconds. When the cache is established, data can only be transferred from the cache to the application and not directly from the Cloud DB. What should be the projected hit rate in the cache to achieve this ambitious average query latency target? Which locality is used here – Spatial or Temporal ?

With Redis Cache

$Tavg = h * Tc + (1-h) * (Tc + Tm)$

$15 = h*7+(1-h)*(7+45)$

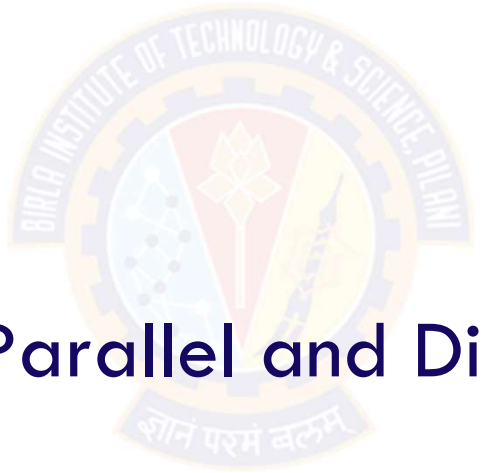
$15 = h*7 + 52 – h*52$

$15 = -h*45 + 52$

$h*45 = 37$

$h = 37/45 = 0.82$

Temporal locality is used in this case.



Next Session:
Parallel and Distributed Processing