Exploiting Sequential Locality for Fast Disk Accesses

Xiaodong Zhang

Ohio State University

In collaboration with

Song Jiang, Wayne State University

Feng Chen and Xiaoning Ding, Ohio State

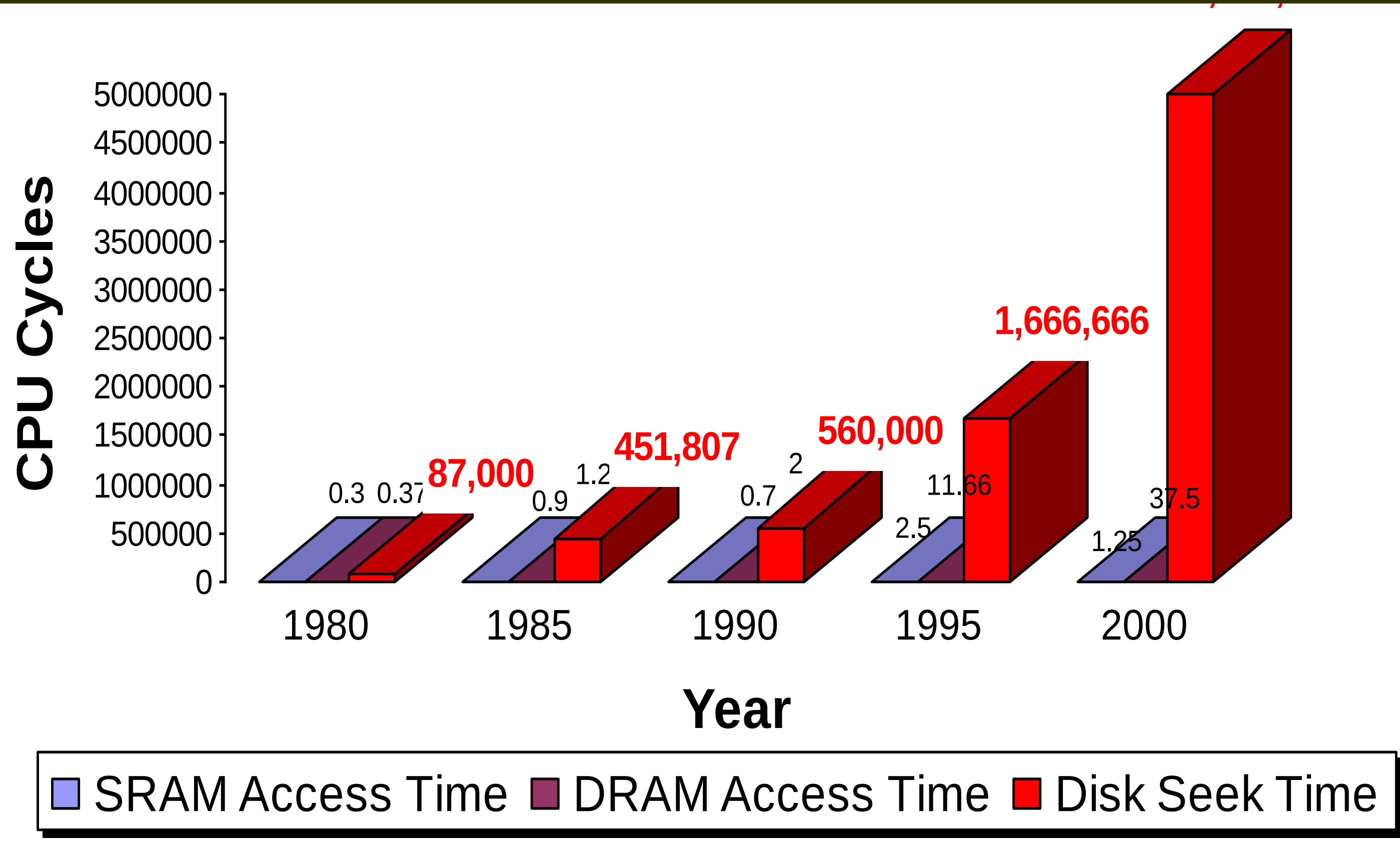
Kei Davis, Los Alamos National Lab

"Disk Wall" is a Critical Issue

- Many data-intensive applications generate huge data sets in disks world wide in very fast speed.
  - LANL Turbulence Simulation: processing 100+ TB.
  - Google searches and accesses over 10 billion web pages and tens of TB data in Internet.
  - Internet traffic is expected to increase from 1 to 16 million TB/month due to multimedia data.
  - We carry very large digital data, films, photos, …
- Data home is the cost-effective & reliable Disks
  - Slow disk data access is the major bottleneck

The disks in 2000 are 57 times "SLOWER" than their ancestors in 1980 --- increasingly widen the Speed Gap

between Peta-Scale computing and Peta-Byte acesses.



Bryant and O'Hallaron, "Computer Systems: A Programmer's Perspective", Prentice Hall, 2003

Data-Intensive Scalable Computing (DISC)

Massively Accessing/Processing Data Sets in Parallel.

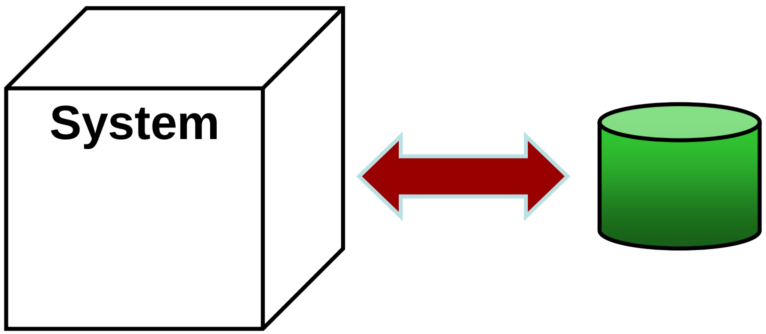
- drafted by R. Bryant at CMU, endorsed by Industries: Intel, Google, Microsoft, Sun, and scientists in many areas.
- Applications in science, industry, and business.

Special requirements for DISC Infrastructure:

- Top 500 DISC ranked by data throughput, as well FLOPS
- Frequent interactions between parallel CPUs and distributed storages. Scalability is challenging.
- DISC is not an extension of SC, but demands new technology advancements.

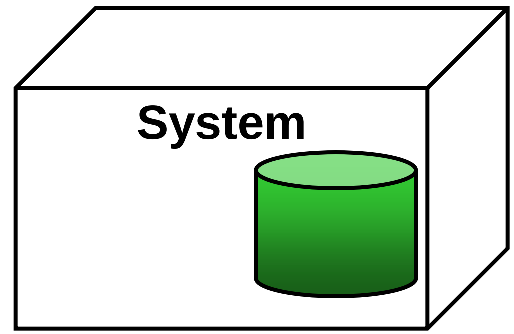
Systems Comparison: (courtesy of Bryant)

Conventional Supercomputers



- Disk data stored separately
  - No support for collection or management
- Brought in for computation
  - Time consuming
  - Limits interactivity

DISC



- System collects and maintains data
  - Shared, active data set
- Computation co-located with disks
  - Faster access

Principles of Locality

During an interval of execution, a set of data/instructions is repeatedly accessed (working set). (Denning, 70)

temporal locality: data will be re-accessed timely.

spatial locality: data stored nearby will be accessed.

- Similar working set observations in many other areas:

  Law of scattering ('34): significant papers hit core journals.

  Zipf's law ('49): frequently used words concentrate on 7%.

  80-20 rule ('41) for wealth distribution: 20% own 80% total.

- Exploiting locality: identify/place working set in caches

  Large caches would never eliminate misses (Kung, 86)

  What can we do after misses?

Sequential Locality is Unique in Disks

- Sequential Locality: disk accesses in sequence fastest
- Disk speed is limited by mechanical constraints.

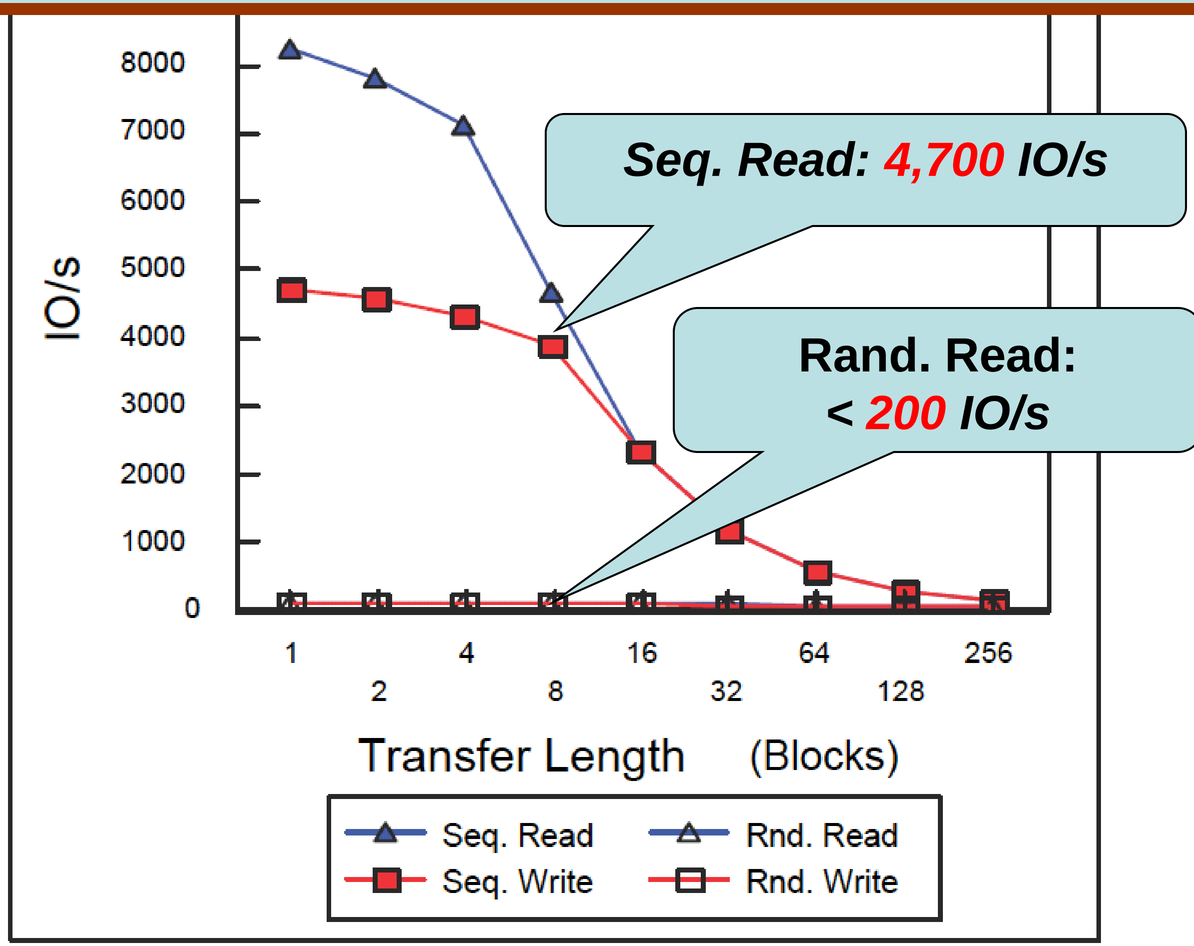
  seek/rotation (high latency and power consumption)
- OS can guess sequential disk-layout, but not always right.

Week OS Ability to Exploit Sequential Locality

- OS is not exactly aware disk layout
  - **Sequential data placement has been implemented**
    - **since Fast File System in BSD (1984)**
    - **put files in one directory in sequence in disks**
    - **follow execution sequence to place data in disks.**
  - Assume temporal sequence = disk layout sequence.
- The assumption is not always right, performance suffers.
  - Data accesses in both sequential and random patterns
  - an application accesses multiple files.
  - Buffer caching/prefetching know little about disk layout.

IBM Ultrastar 18ZX Specification *

Our goal: to maximize opportunities of sequential accesses for high speed and high I/O throughput



. Ultrastar 18ZX with Write Cache on and Read Cache on

* Taken from IBM "ULTRASTAR 9LZX/18ZX Hardware/Functional Specification" Version 2.4

Randomly Scattered Disk Accesses

- Scientific computing
  - Scalable IO (SIO) Report: “in many applications majority of the requests are for small amount of data (less than a few Kbytes)” [Reed 1997]
  - CHARISMA Report: “large, regular data structures are distributed among processes with interleaved accesses of shared files” [Kotz 1996]
- Workloads on popular operating systems
  - UNIX: most accessed files are short in length (80% are smaller than 26 Kbytes) [Ousterhout,1991]
  - Windows NT: 40% I/O operations are to files shorter than 2KBytes [Vogels, 1999]

Random Accesses from Multiple Objects

- Advanced disk arrays:
  - HP FC-60 disk arrays: “Most workloads have a range of small and large jumps in sequential accesses and interferences between concurrent access streams”. [Keeton 2001]
  - Detecting sources of irregular disk access patterns: “…, most data objects are much smaller than the disk request sizes needed to achieve good efficiency.” [Shindler 2002]
- Peta-Byte data analysis relies on random disk accesses:
  - Many Peta-Bytes of active data for BaBar experiments
  - Data analysis: random analysis of small blocks.
  - A researcher has several hundred data streams in batch mode
  - Several hundred concurrent researchers are active.
- PetaCache (CalTech, 2004) is an expensive and temporary solution.

Existing Approaches and Limits

- Programming for Disk Performance
  - Hiding disk latency by overlapping computing
  - Sorting large data sets (SIGMOD'97)
  - Application dependent and programming burden
- Transparent and Informed Prefetching (TIP)
  - Applications issue hints on their future I/O patterns to guide prefetching/caching (SOSP'99)
  - Not a general enough to cover all applications
- Collective I/O: gather multiple I/O requests
  - make contiguous disk accesses for parallel programs

Our Objectives

- Exploiting sequential locality in disks
  - by minimizing random disk accesses
  - making disk-aware caching and prefetching

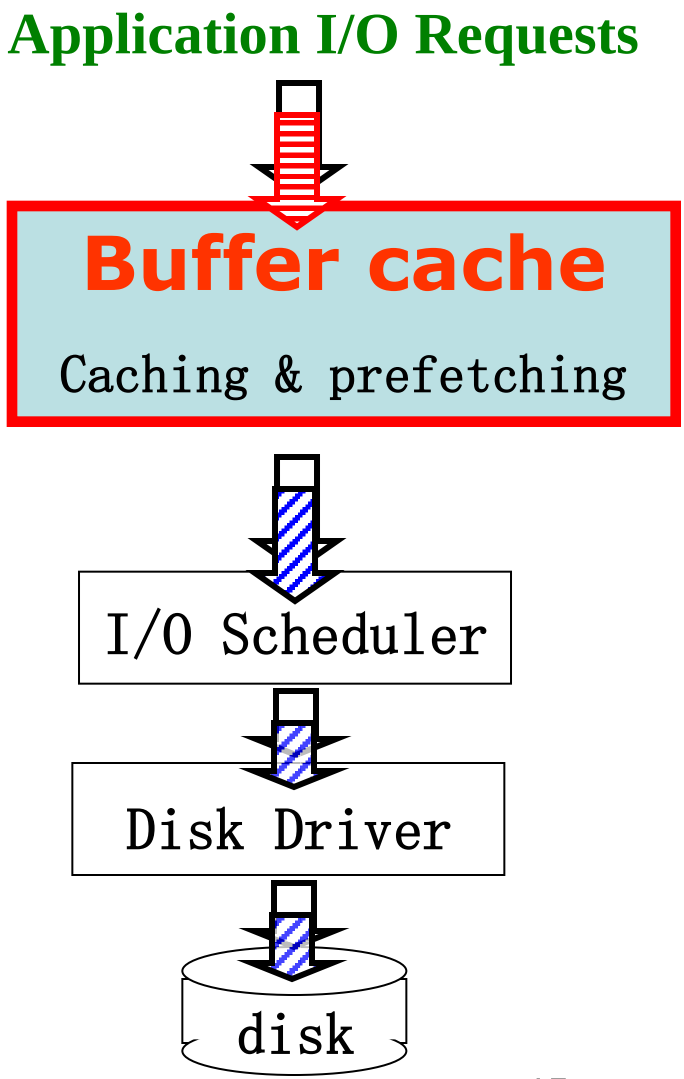
- Application independent approach
  - putting disk access information on OS map
  - Exploiting DUal LOcalities (DULO):
    - Temporal locality of program execution
    - Sequential locality of disk accesses

Outline

- What is missing in buffer cache management?
- Managing disk layout information in OS
- DULO-caching
- DULO-prefetching
- Performance results in Linux kernel
- Summary

What is Buffer Cache Aware and Unaware?

- Buffer is an agent between I/O requests and disks.
  - aware access patterns in time sequence (in a good position to exploit temporal locality)
  - not clear about physical layout (limited ability to exploit sequential locality in disks)
- Existing functions
  - send unsatisfied requests to disks
  - **LRU replacement by temporal locality**
  - make prefetch by sequential access assumption.
- Ineffectiveness of I/O scheduler: sequential locality in is not open to buffer management.

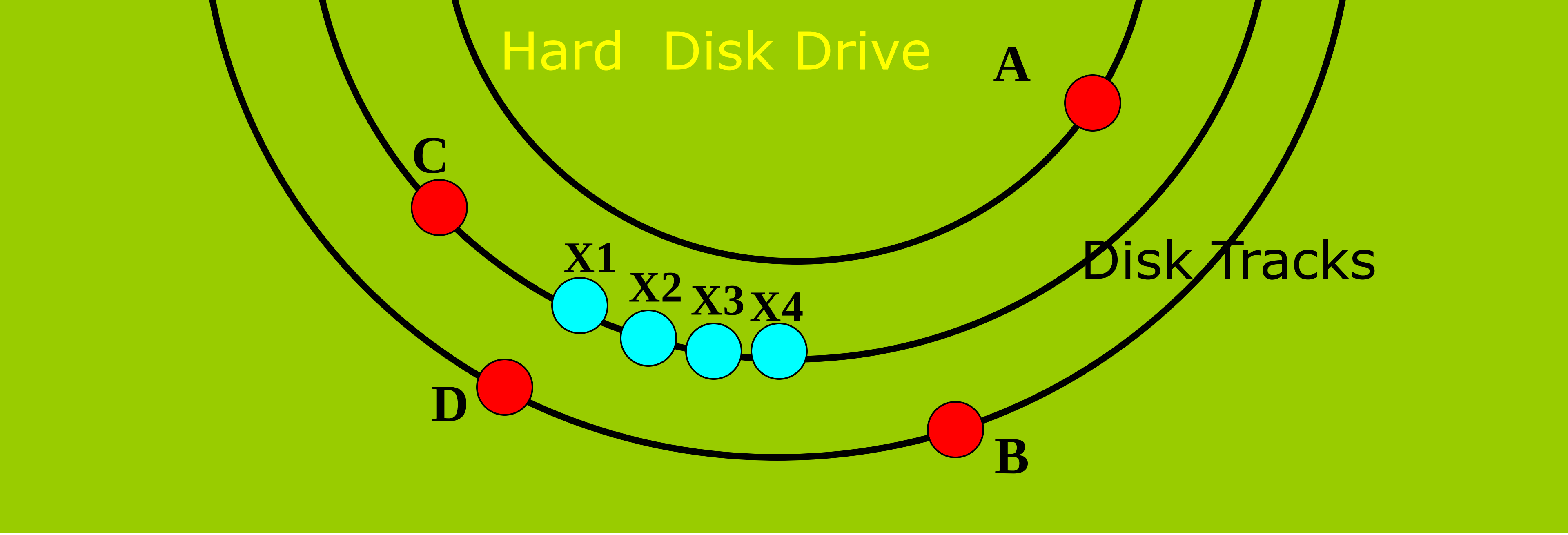


Limits of Hit-ratio based Buffer Cache Management

- Minimizing cache miss ratio by only exploiting temporal locality

$$\begin{aligned} &Average\ access\ time \\ &= \underbrace{Hit\ rate \times Hit\ time}_{\text{Temporal locality}} + \underbrace{Miss\ rate \times \text{Miss penalty}}_{\text{Sequential locality}} \end{aligned}$$

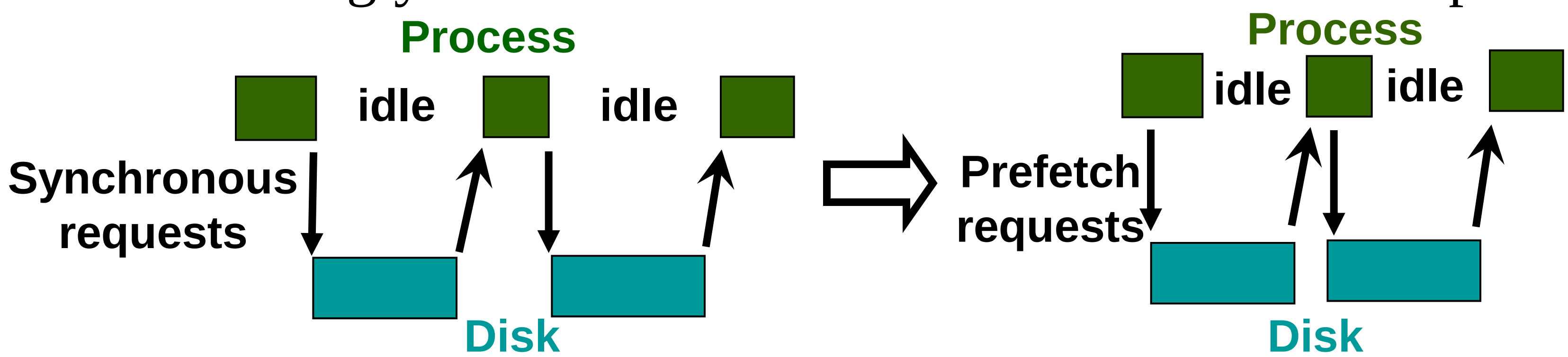
- Sequentially accessed blocks → small miss penalty
- Randomly accessed blocks → large miss penalty



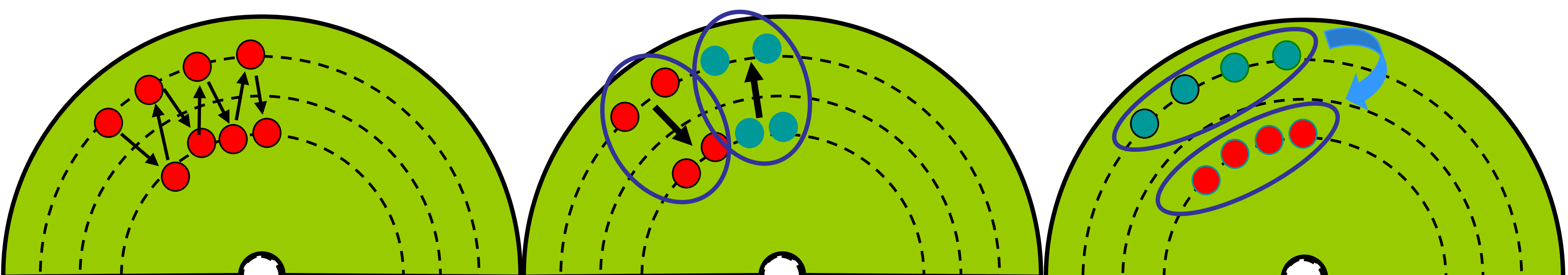
- **Unique and critical roles of buffer cache**
- Buffer cache can influence request stream patterns in disks
- If buffer cache is disk-layout-aware, OS is able to
  - Distinguish sequentially and randomly accessed blocks
  - Give “expensive” random blocks a high caching priority
  - replace long sequential data blocks timely to disks
  - Disk accesses become more sequential.

Prefetching Efficiency is Performance Critical

It is increasingly difficult to hide disk accesses behind computation

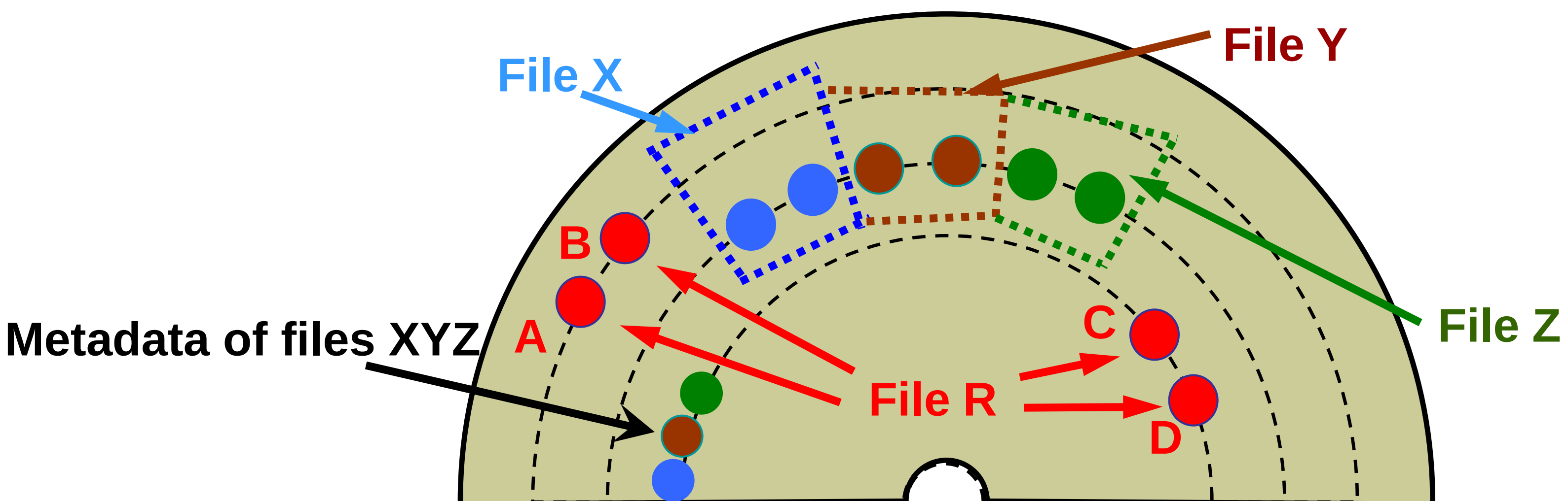


- Prefetching may incur non-sequential disk access
  - Non-sequential accesses are much slower than sequential accesses
  - Disk layout information must be introduced into prefetching policies.



File-level Prefetching is Disk Layout Unaware

- Multiple files sequentially allocated on disks cannot be prefetched at once.
- Metadata are allocated separately on disks, and cannot be prefetched
- Sequentiality at file abstraction may not translate to sequentiality on physical disk.
- Deep access history information is usually not recorded.



Opportunities and Challenges

- **With Disk Spatial Locality (Disk-Seen)**
  - Exploit DULO for fast disk accesses.
  - Make disks as a close part of the system for DISC.
- **Challenges to build Disk-Seen System Infrastructure**
  - Disk layout information is increasingly hidden in disks.
  - analyze and utilize disk-layout Information
  - accurately and timely identify sequential locality
  - trade-offs between buffer cache hit ratio vs miss penalty
  - manage its data structures with low overhead
  - Implement it in OS kernel for practical usage

Disk-Seen Task 1: Make Disk Layout Info. Available

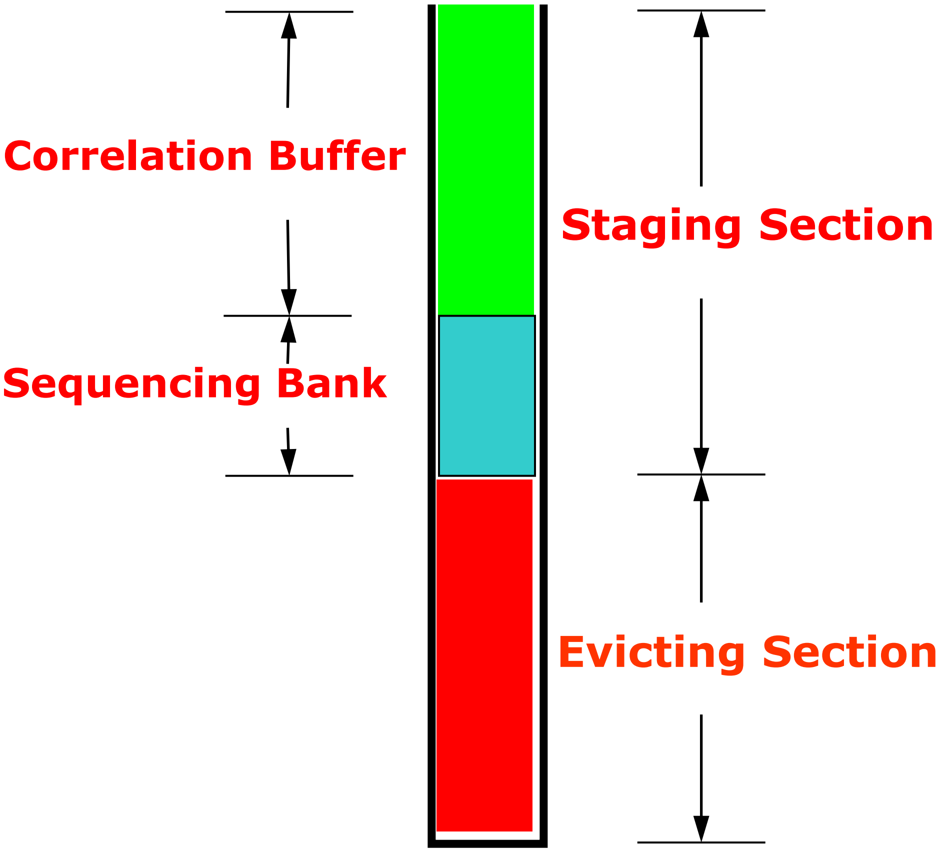
- Which disk layout information to use?
  - Logical block number (LBN): location mapping provided by firmware. (each block is given a sequence number)
  - Accesses of contiguous LBNs have a performance close to accesses of contiguous blocks on disk. (except bad blocks occur)
  - The LBN interface is highly portable across platforms.
- How to efficiently manage the disk layout information?
  - LBN is only used to identify disk locations for read/write;
  - We want to track access times of disk blocks and search for access sequences via LBNs;
  - Disk block table: a data structure for efficient disk blocks tracking.

Disk-Seen TASK 2: Exploiting Dual Localities (DULO)

- ❑ **Sequence Forming**

  Sequence ---- a number of blocks whose disk locations are adjacent and have been accessed during a limited time period.

- ❑ **Sequence Sorting** based on its recency (temporal locality) and size (spatial locality)



LRU Stack

Disk-Seen TASK 3: DULO-Caching

Adapted GreedyDual Algorithm

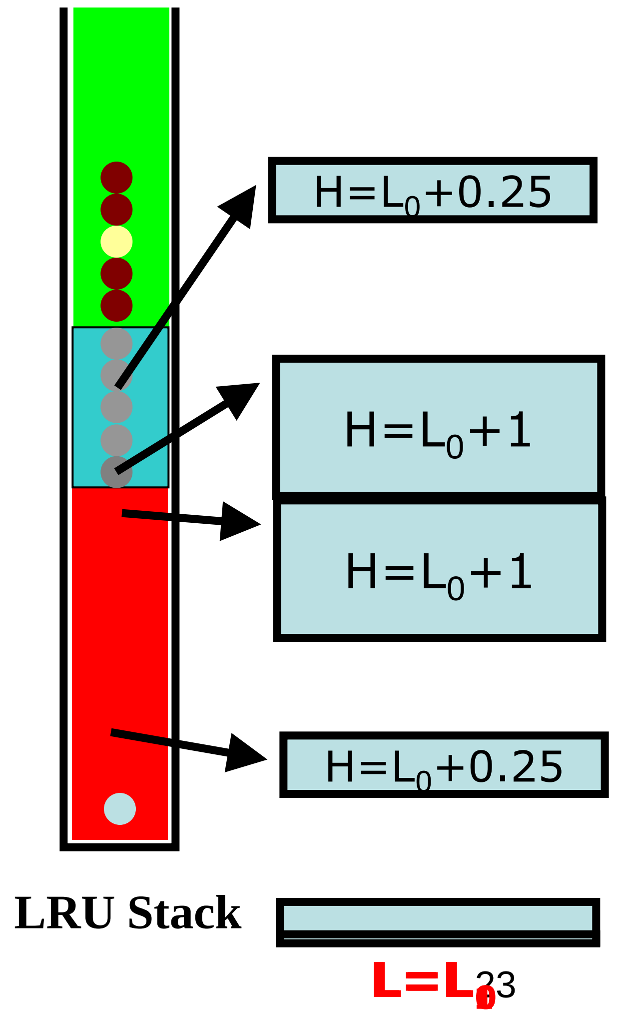
- a global inflation value **L** , and a value **H** for each sequence
- Calculate H values for sequences in sequencing bank:

  H = L + 1 / Length(sequence)

  Random blocks have larger H values
- When a sequence (s) is replaced,

  L = H value of s .

  L increases monotonically and make future sequences have larger H values
- Sequences with smaller H values are placed closer to the bottom of LRU stack



Disk-Seen TASK 3: DULO-Caching

Adapted GreedyDual Algorithm

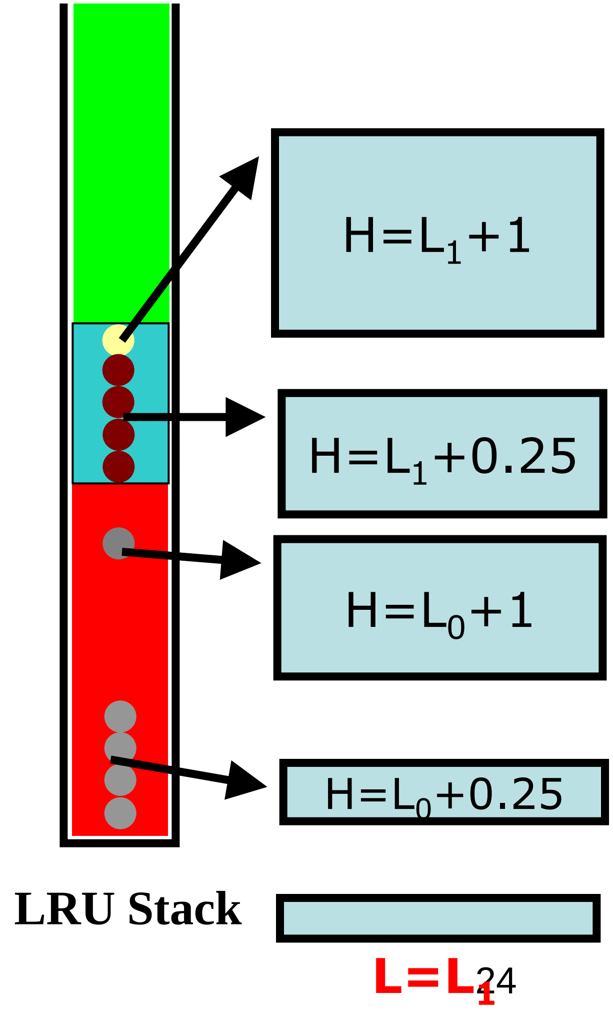
- a global inflation value L , and a value H for each sequence
- Calculate H values for sequences in sequencing bank:

  H = L + 1 / Length(sequence)

  Random blocks have larger H values
- When a sequence (s) is replaced,

  L = H value of s .

  L increases monotonically and make future sequences have larger H values
- Sequences with smaller H values are placed closer to the bottom of LRU stack

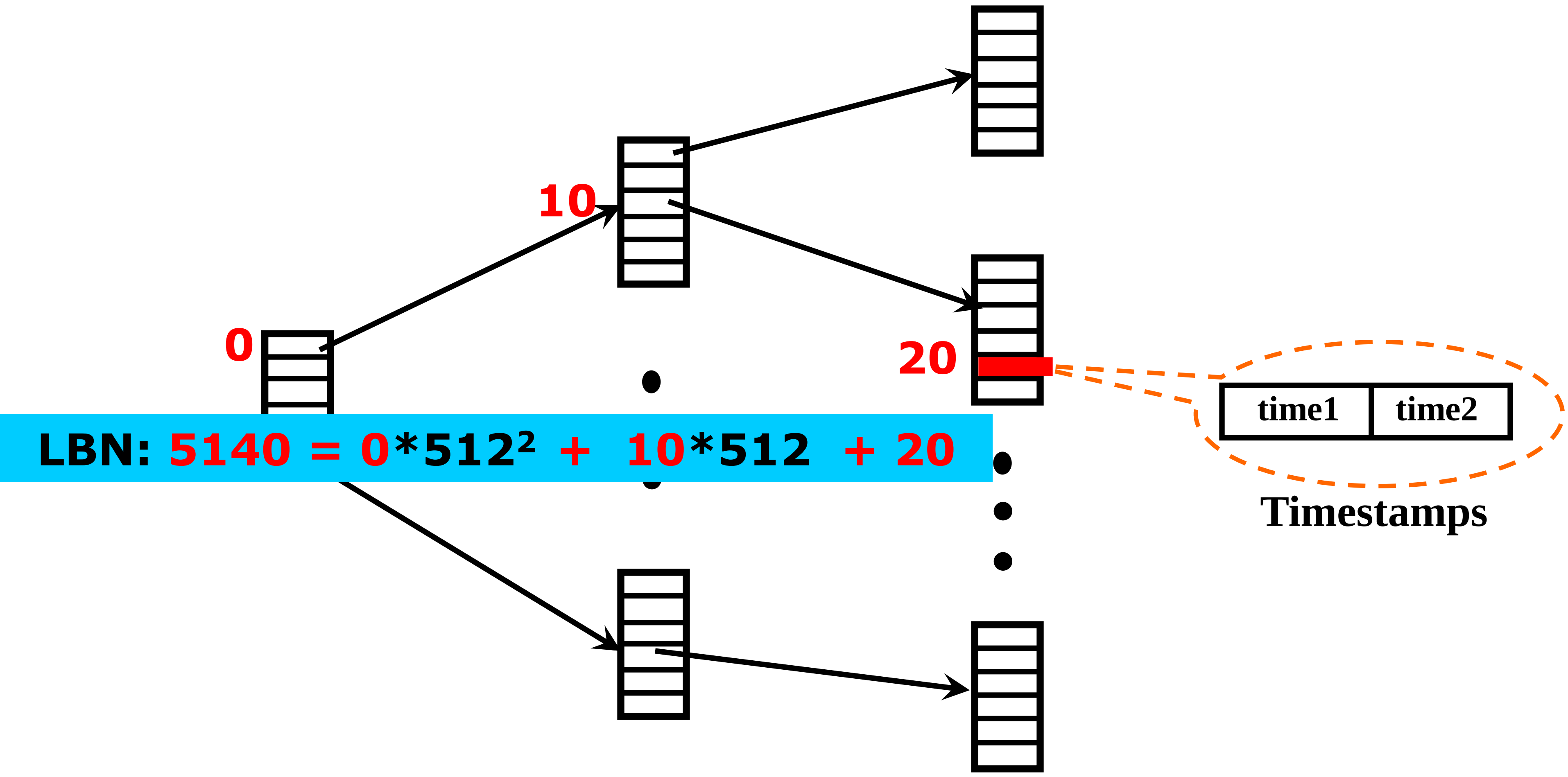


DULO-Caching Principles

- Moving long sequences to the bottom of stack
  - replace them early, get them back fast from disks
- Replacement priority is set by sequence length.
- Moving LRU sequences to the bottom of stack
  - exploiting temporal locality of data accesses
- Keeping random blocks in upper level stack
  - hold them: expensive to get back from disks.

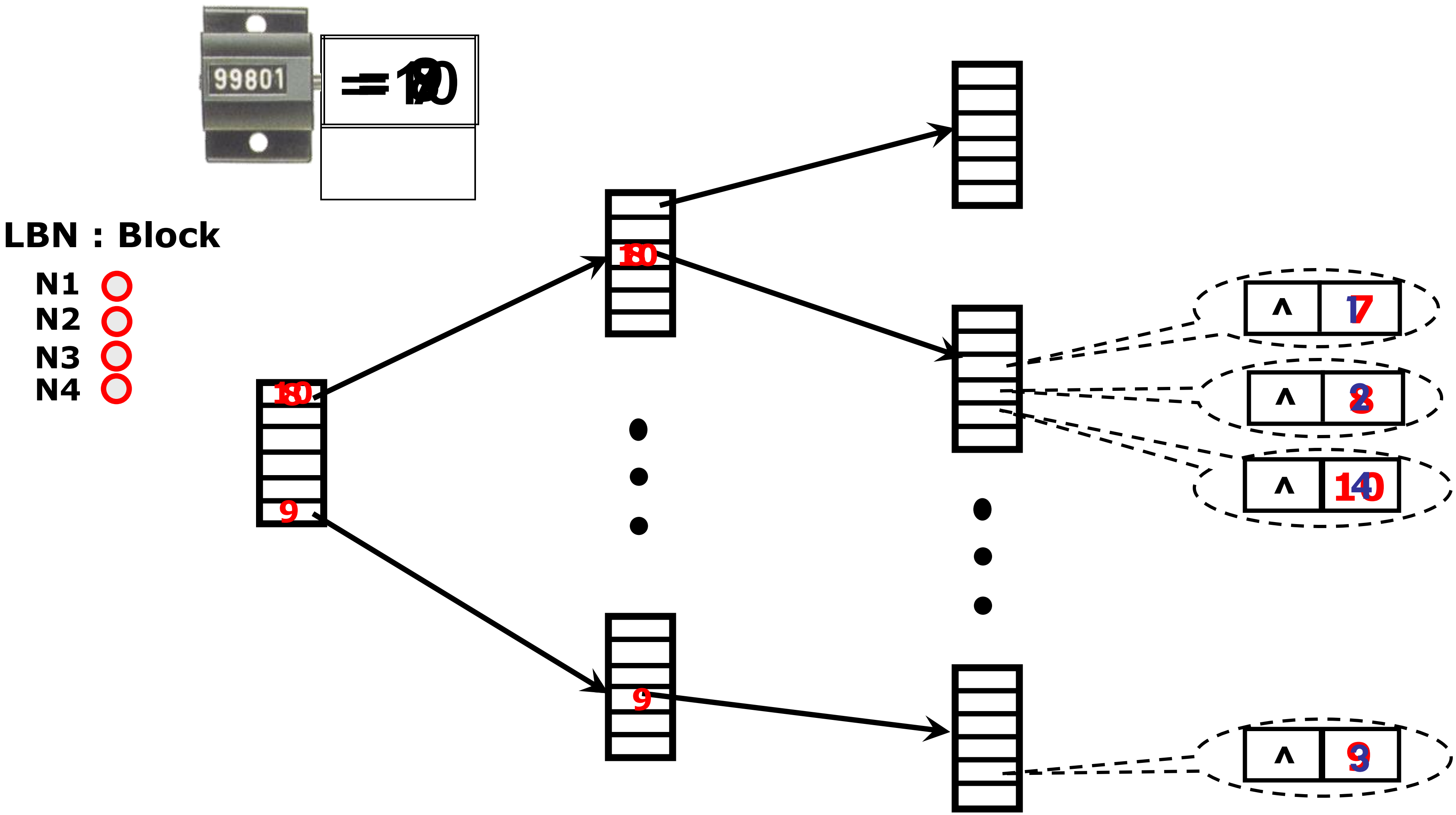
Disk-Seen Task 4: Identifying Long Disk Sequence

a data structure for tracking disk blocks



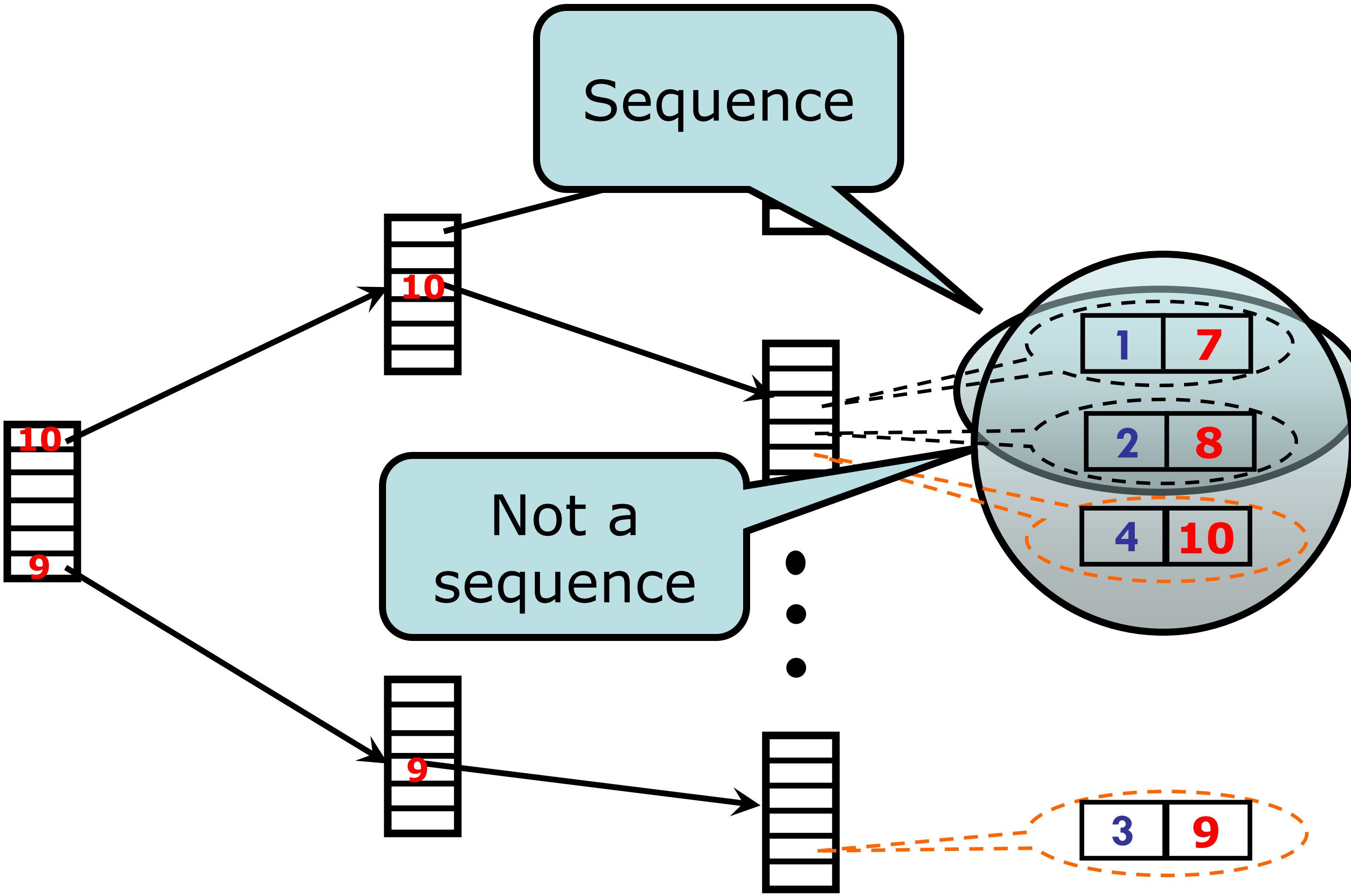
Disk-Seen Task 4: Identifying Long Disk Sequence

a new data structure for tracking disk blocks



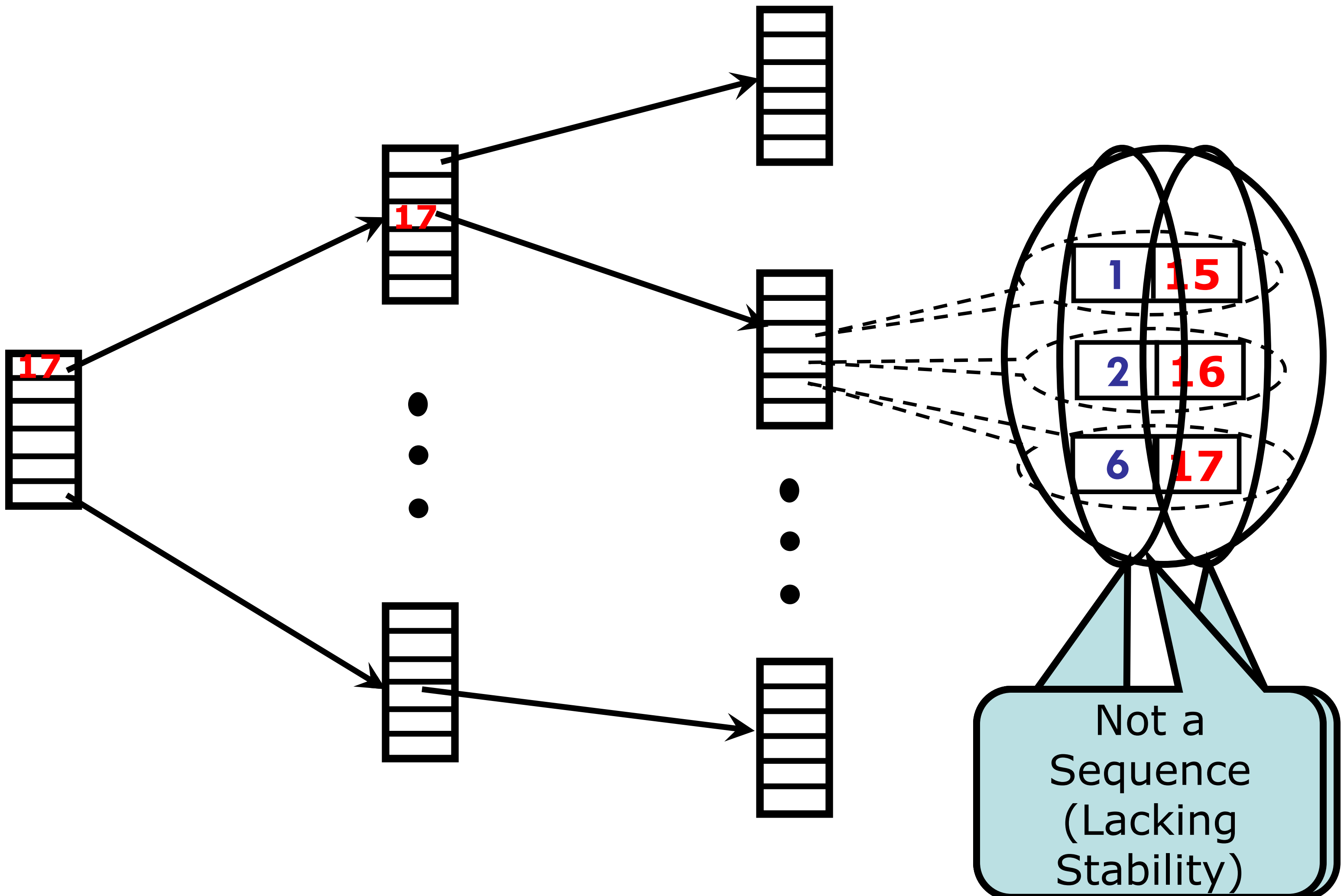
Disk-Seen Task 4: Identifying Long Disk Sequence

a new data structure for tracking disk blocks

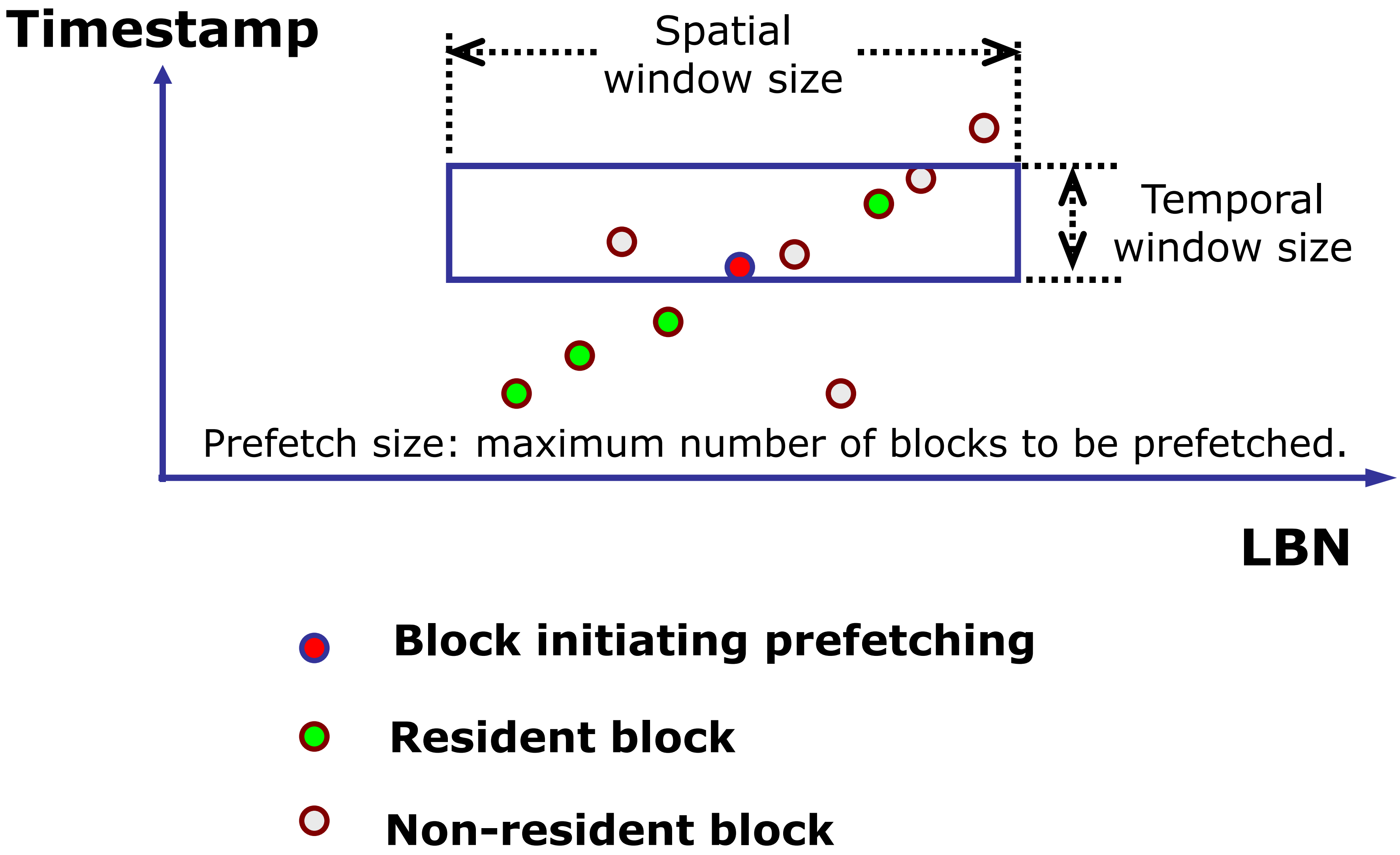


Disk-Seen Task 4: Identifying Long Disk Sequence

a new data structure for tracking disk blocks



Disk-Seen Task 5: DULO-Prefetching

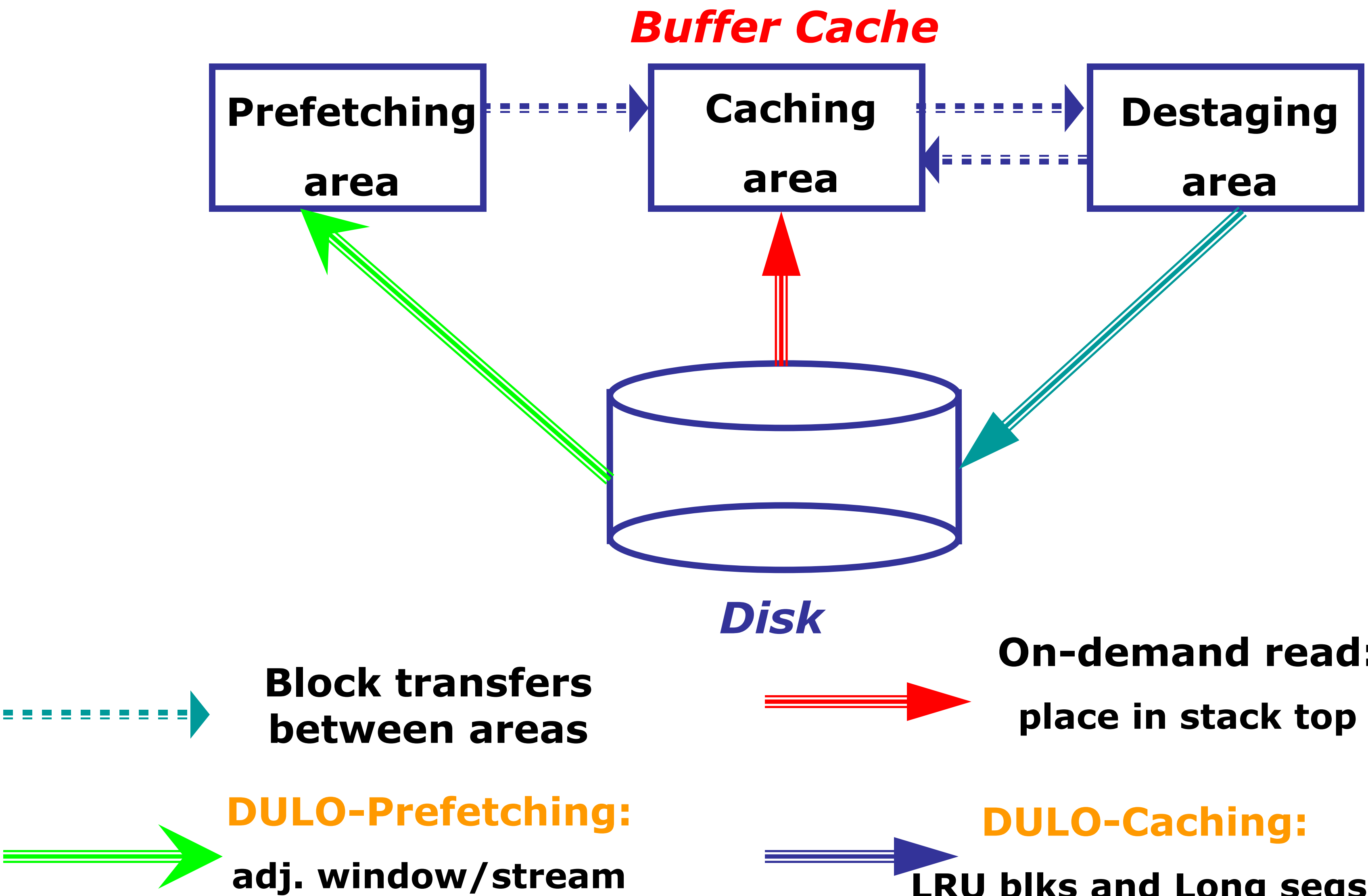


What can DULO-Caching/-Prefetch do and not do?

- Effective to
  - mixed sequential/random accesses. (cache them differently)
  - many small files. (packaging them in prefetch)
  - many one-time sequential accesses (replace them quickly).
  - repeatable complex patterns that cannot be detected without disk info. (remember them)

- Not effective to
  - dominantly random/sequential accesses. (perform equivalently to LRU)
  - a large file sequentially located in disks. (file-level prefetch can do it)
  - non-repeatable accesses. (perform equivalently to file-level prefetch)

DiskSeen: a System Infrastructure to Support DULO-Caching and DULO-Prefetching



The DiskSeen Prototype in Linux 2.6.11

- Use raw device file to prefetch blocks
- Linux file-level prefetching remains enabled
- Blocks without disk mappings are treated as random blocks
- Intel P4 3.0GHz processor, a 512MB memory, and Western Digital hard disk of 7200 RPM and 160GB
- The file system is Ext2.

Benchmarks Programs To Test DULOs (1)

- **BLAST:** a tool searching databases to match nucleotide or protein sequences (mixed patterns)
  - Data file: sequentially accessed
  - Index and header files: randomly accessed
- **PostMark:** a file system benchmark of e-mail servers or news group servers (mixed patterns)
  - Randomly select files and sequentially access each file
  - Small files: random blocks; Large files: long sequences.
- **LXR:** a software serving user queries for searching, browsing, or comparing source code trees through an HTTP server. (mixed patterns, small files)

Benchmark Programs to Test DULOs (2)

- TPC-H: a decision support benchmark
  - 2 of the 22 queries are selected
  - query #4: join two tables and large working sets (random patterns)
  - query #6: table scan a large table (sequential access)

- diff: a tool comparing two files or directories. Compare two Linux kernel trees. (small files, random accesses)

- CVS: a versioning control system (small files, sequential accesses).

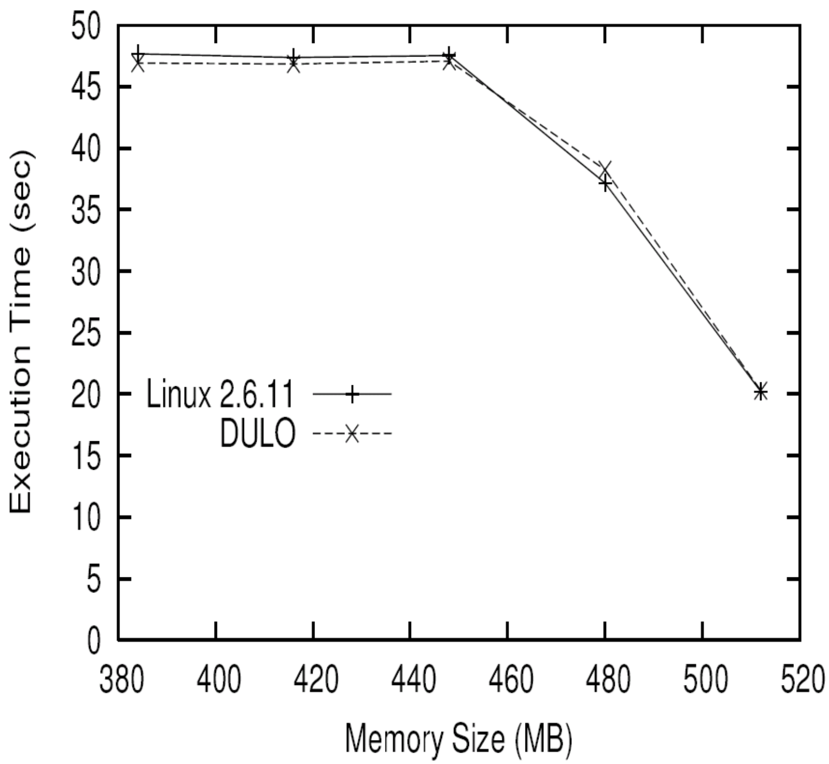
Benchmark Programs to Test DULO (3)

- grep: search a set of files for lines containing a match to a given pattern (small files, sequential accesses).

- Strided: stridedly read a large file (1GB). Skip 4KB then read 8KB in each period. (mixed patterns)

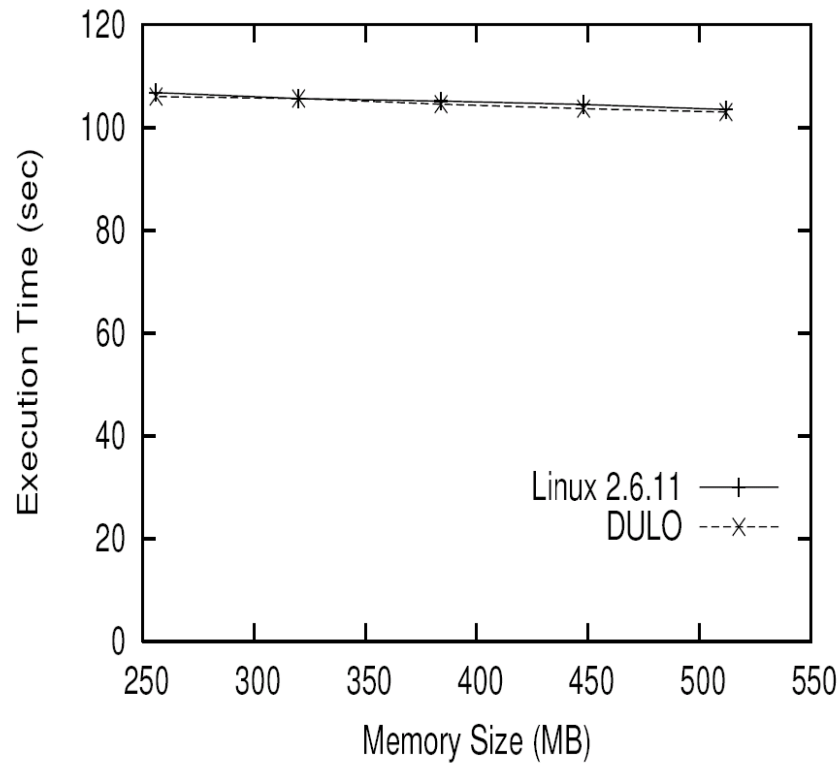
- Reverse: Read a large file (1GB) reversely. (sequential accesses)

DULO Caching does not affect Execution Times of Pure Sequential or Random Workloads



TPC-H Query #6

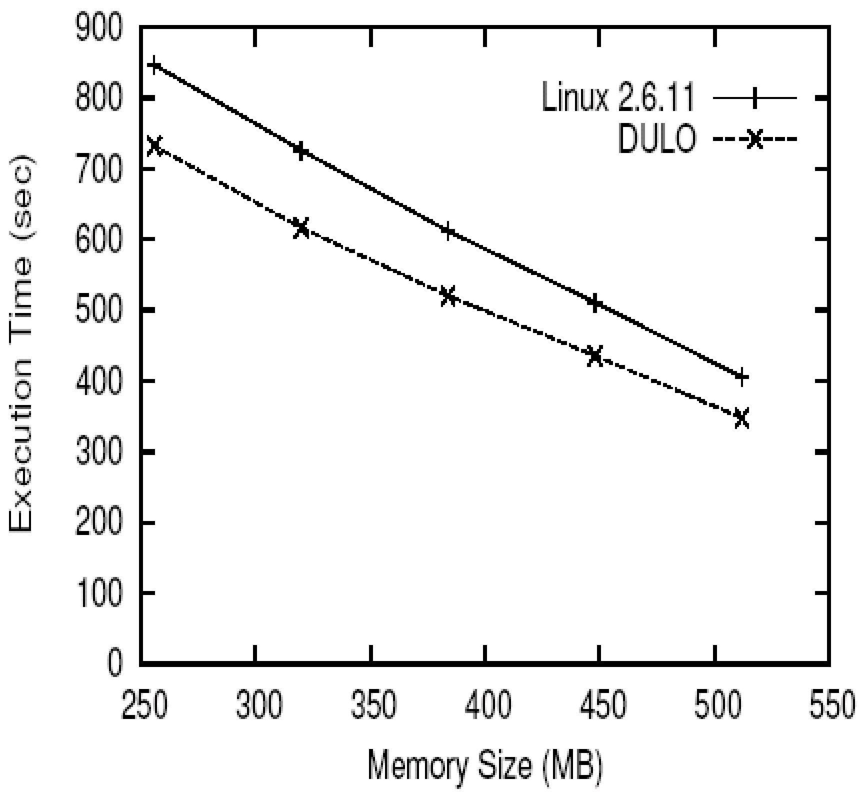
(sequential accesses)

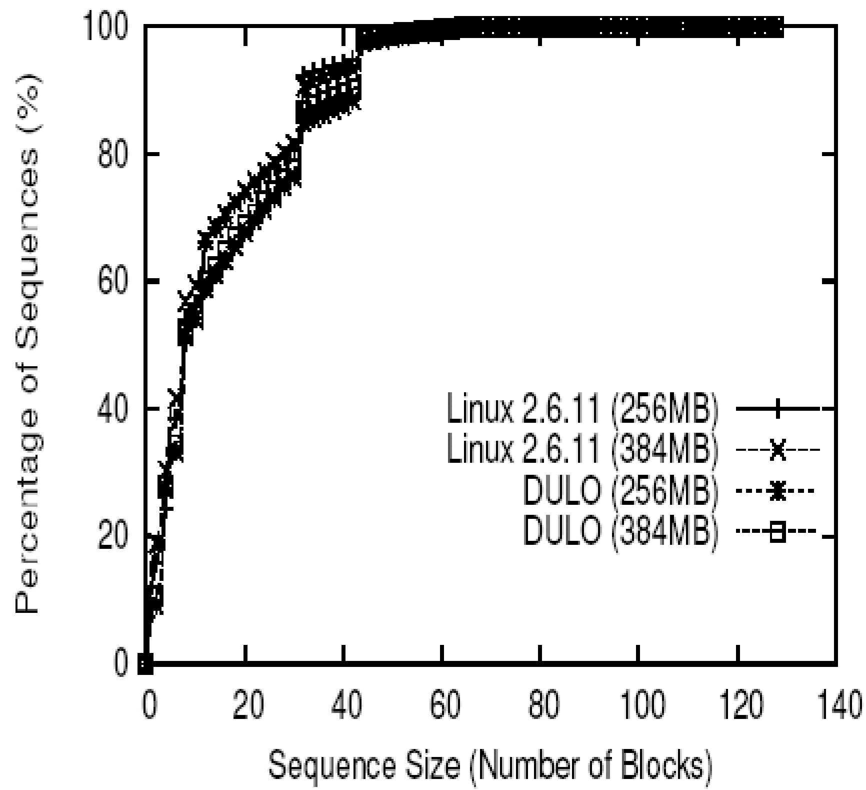


Diff

(random accesses)

DULO Caching Reduces Execution Times for Workloads with Mixed Patterns

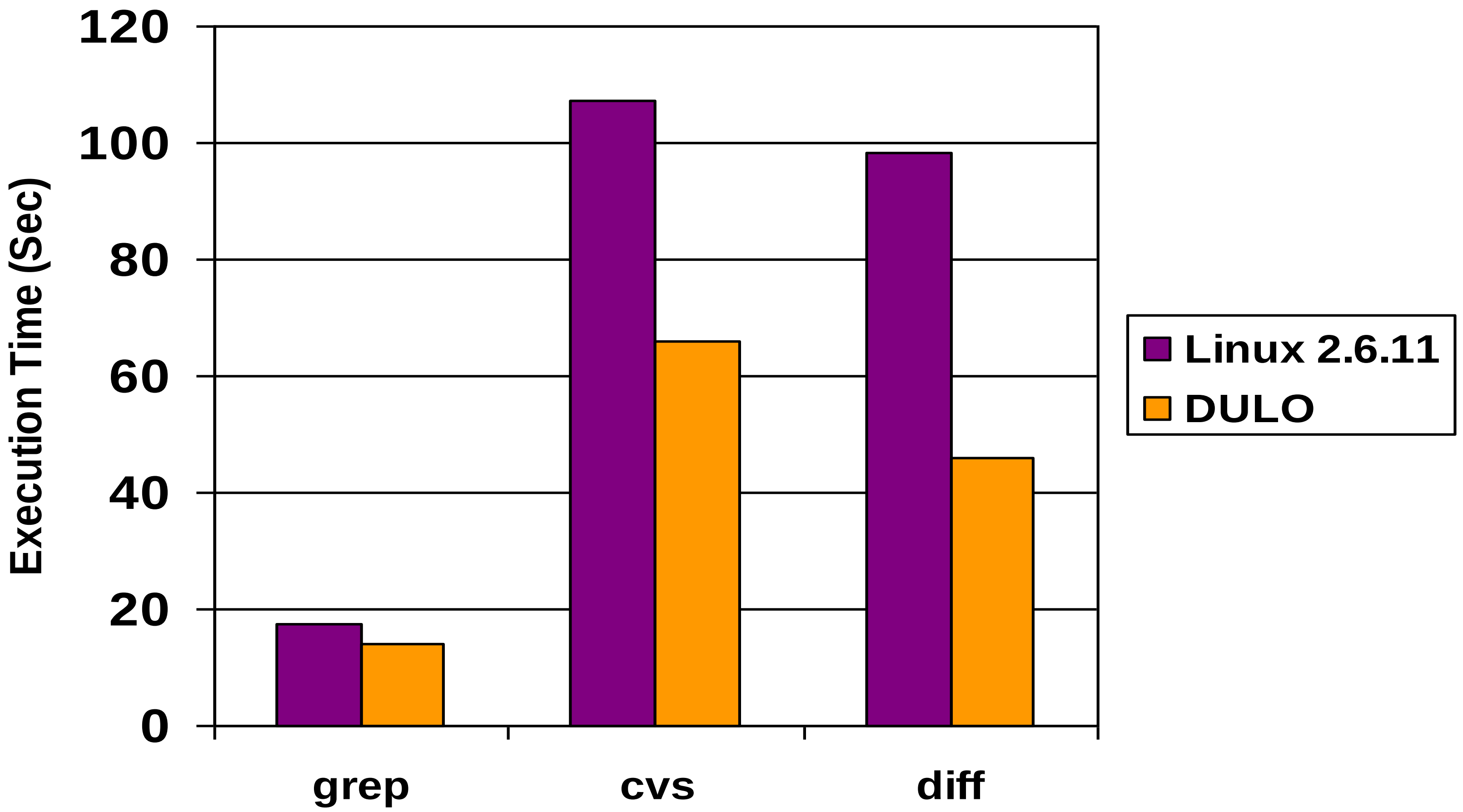




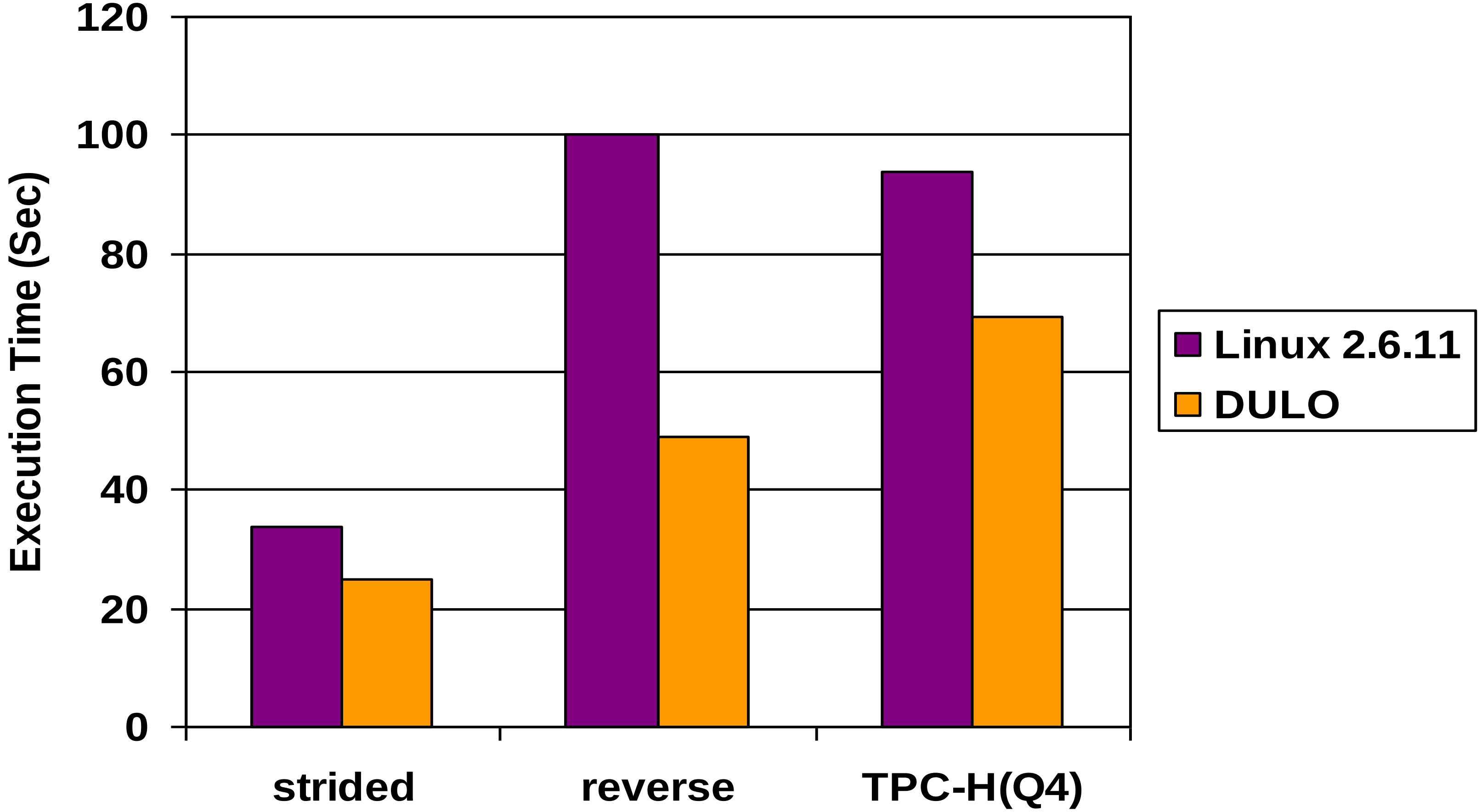
PostMark

(mixed patterns of both sequential and random)

DULO Prefetching Reduces Execution Times for Workloads with Many Small Files



DULO Prefetching Reduces Execution Times for Workloads With Complex Access Patterns



Conclusions

- Disk performance is limited by
  - Non-uniform accesses: fast sequential, slow random
  - OS is unable to effectively exploit sequential locality.
- The buffer cache is a critical component for storage.
  - temporal locality is mainly exploited by existing OS.
- Building a Disk-Seen system infrastructure for
  - DULO-Caching and -prefetching
  - a system component for Data Intensive SuperComputer
- The size of the block table is 0.1% (4 K block) of disk capacity. Its working set can be in buffer cache.