Solid-State Disks: How Do They Change the DBMS Game?

Angelo Brayner
Univ. de Fortaleza, Brazil
Mario A. Nascimento
Univ. of Alberta, Canada

SBBD
28º Simpósio Brasileiro de Banco de Dados 2013

Outline

- Introduction
- Physical Storage
  - HDs and SSDs
- Revisiting Fundamental DBMS Techniques and Algorithms
  - Indexing

  - Join Processing
  - Query optimization
  - Caching
  - Logging

Part 1

Part 2

Tutorial Outline

- Introduction
- Physical Storage
  - HDs and SSDs
- Revisiting Fundamental DBMS Techniques and Algorithms
  - Indexing
  - Join Processing
  - Query optimization
  - Caching
  - Logging

Introduction

- Stable and "durable" storage, e.g., a disk, is non-optional for DBMSs
- While data resides on disk, it needs to be brought up to main memory for processing
- Until recently, hard disks (HDs) were the only option for storage media
  – The difference in access time between main memory and HDs still is in range of a few orders of magnitude (nsecs vs. msecs)

Introduction

- Recently solid state disks (SSDs) became commercially viable for large scale data storage
  - The difference in access time between SSDs and main memory is much smaller (μsecs vs. nsecs)
- How does it affect the DBMS world?

That is what we are going to discuss in the next few hours …

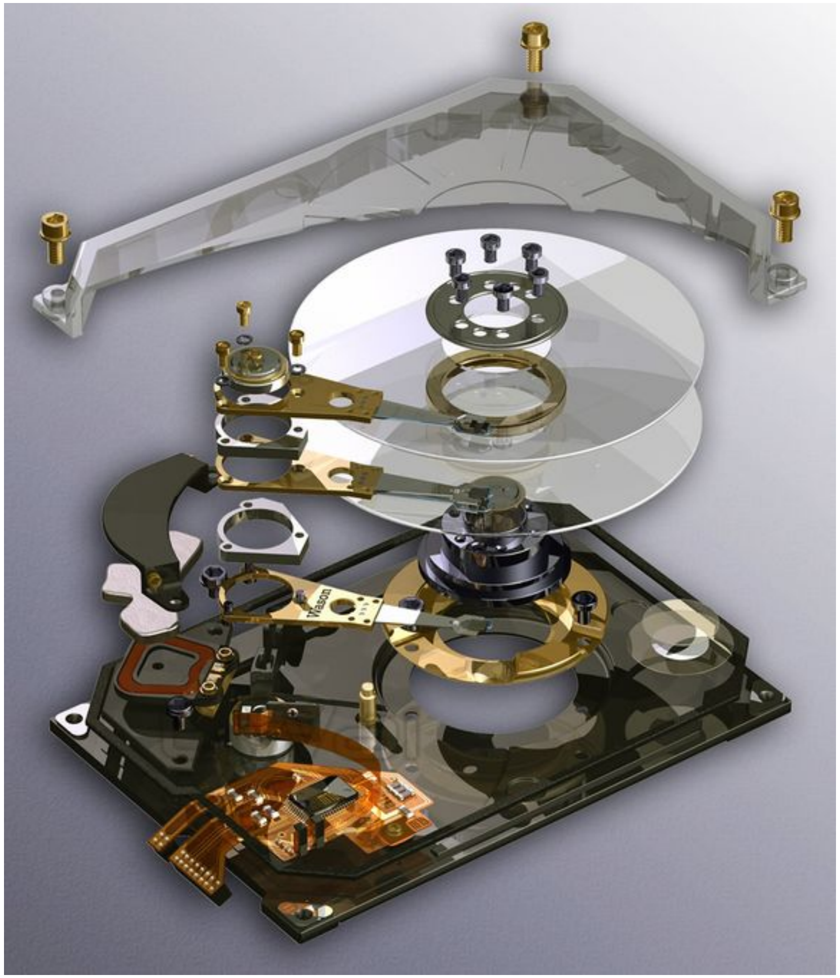
Introduction

- In this tutorial we will discuss:
  - The architecture of HDs and SSDs
    - What makes SSDs fundamentally different from HDs?
    - How these differences affect the way DBMSs work?
  - How important DBMS techniques/algorithms cope (or not) with SSDs:
    - Indexing, join processing, query optimization, caching and logging

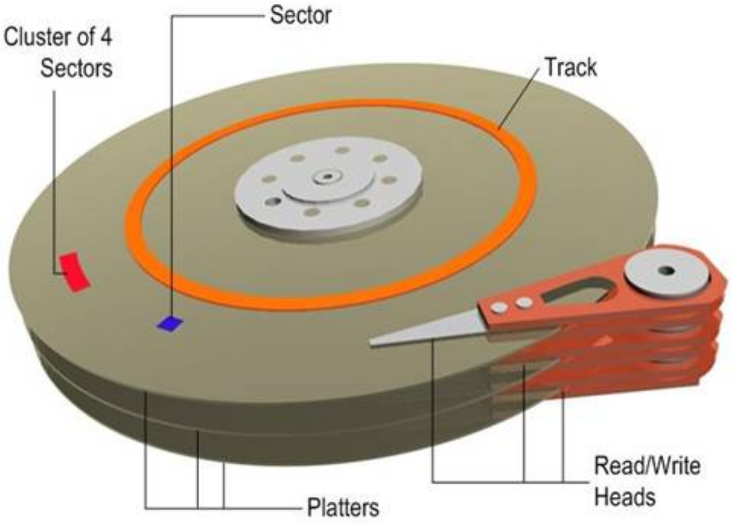
Outline

- Introduction
- Physical Storage
  - HDs and SSDs
- Revisiting Fundamental DBMS Techniques and Algorithms
  - Indexing
  - Join Processing
  - Query optimization
  - Caching
  - Logging

Physical Storage Hard Disks (HDs)



http://www.datarecoverytools.co.uk/



http://technet.microsoft.com/en-us/library/dd758814(v=sql.100).aspx

HDs

- Essentially a mechanical device
- Access data involves:
  - Seek time (finding the right track), rotational delay (finding the right sector, cluster and page) and transfer time (bringing data to main memory
- Time to access a random disk page is in the order of a few msecs and depends heavily on where the data is physically located

HDs

- Physical placement of data on disk is, more often than not, much less than ideal
  - Operating systems (OSs) have different “priorities” when compared to DBMSs, and bypassing an OS is not always feasible
- Virtually every technique and algorithm used within a DBMS today has had the HD’s architecture and inherent overhead as a chief concern

HDs

- In an ideal world we would have the DBMS as well as its data within main memory
- Failing that (which it does) it would help a lot to have faster access time and less dependence on data's physical location
  - Hence, true physical independence in addition to logical independence

Physical Storage
Solid State Disks (SSDs)

http://www.macworld.com/

Physical Storage
Solid State Disks (SSDs)

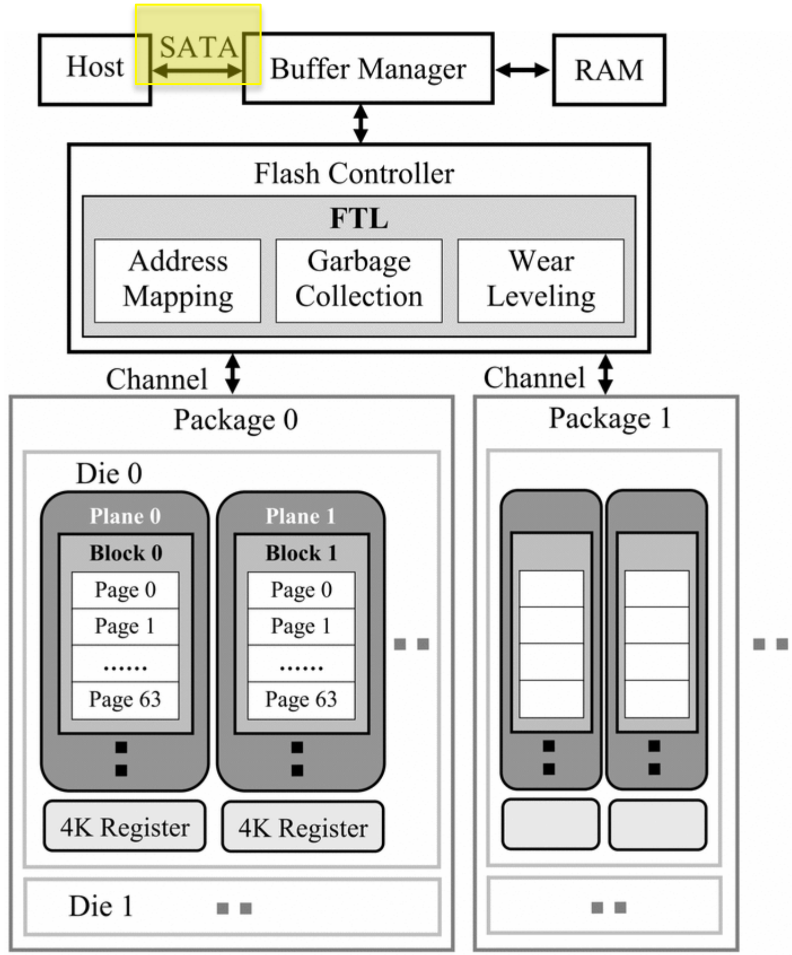
- Despite the naming, SSDs do not have any "disks", in fact, they do not have any mechanical components
- A good comparison between HDs and SSD, across several dimensions, can be found at:
  - http://bit.ly/8IysQk [Wikipedia page]

SSDs

- Yield no “seek time” or “rotational delay”, only transfer time
- Transfer time is orders of magnitude faster than in HDs
- But there is one fundamental difference that will affect DBMS techniques and algorithms:

Read and write operations are (cost-wise) asymmetric

Architecture



Tjioe et al, IEEE NAS 2012

Architecture

- Hierarchy within an SSD:
  - Flash(*) chips
    - Planes
      - Block
        - » Pages

- We are mostly concerned with what happens at the block and page level

(*) Other technologies may be used

R/W Operations

- Read, Program, and Erase
  - Read: reads a page from the disk
  - Program: first-time write on a fresh page
  - Erase: clears up all existing contents within a block
- SSD reads and programs <u>pages</u> but erases <u>blocks</u>.
- SSD pages cannot be overwritten.
  - To update a page within a block, the old page is marked as invalid and then a new <u>fresh</u> page to program the updated value(s) has to be found.

R/W Asymmetry

- On a HDs there is not much difference between the process of reading from or writing onto a page:
  - Bring the right page to memory (subject to seek time, rotational delay and transfer time),
  - Update the page and
  - Flush the page to disk (subject *again* to seek time, rotational delay and transfer time)

R/W Asymmetry

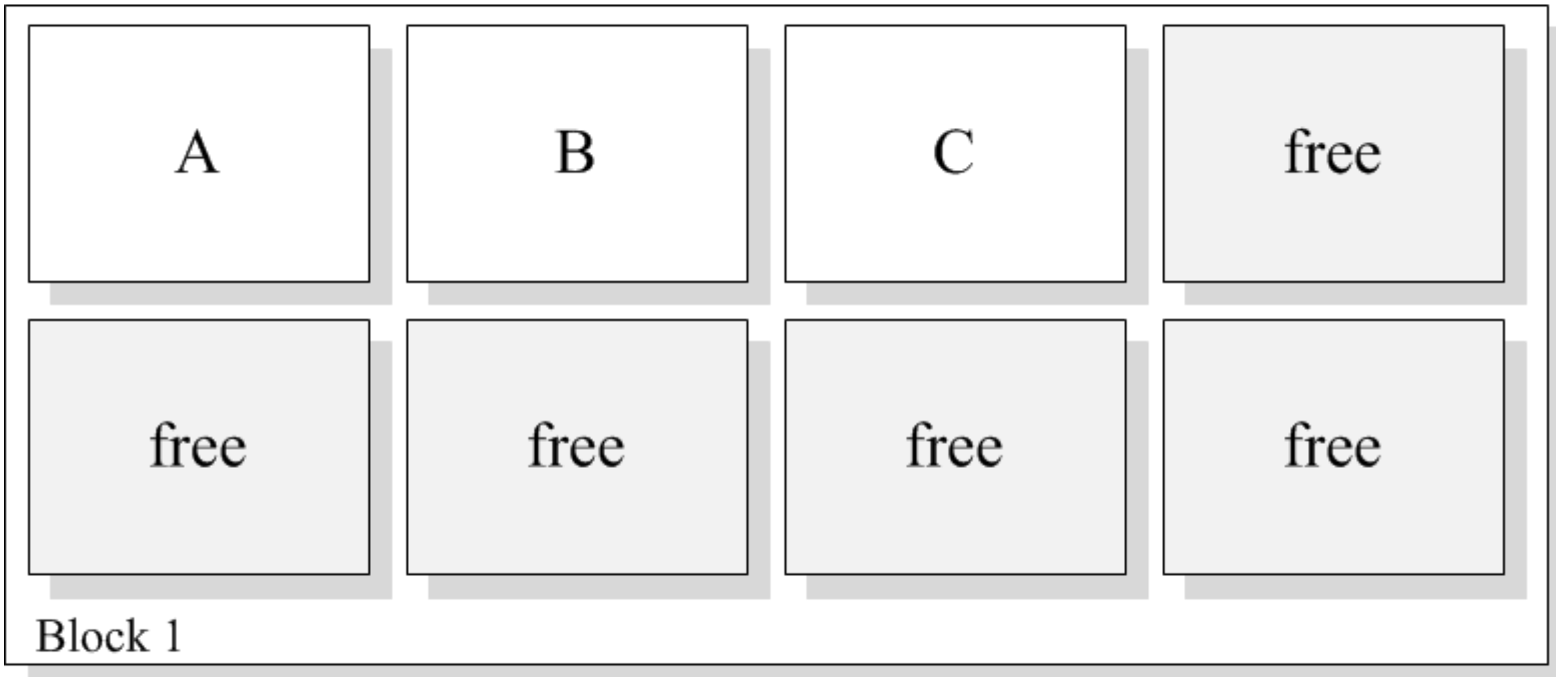
- SSD’s R/Ws are asymmetric due to the need to use a fresh page
- A page read is simply a matter of locating (quickly) the page and transferring it into main memory with no seek time nor rotational delay overhead
- A page write is a completely different story…

The Page Writing Process

(Courtesy of F. Jiang)

A block with 8 free pages initially free (empty)

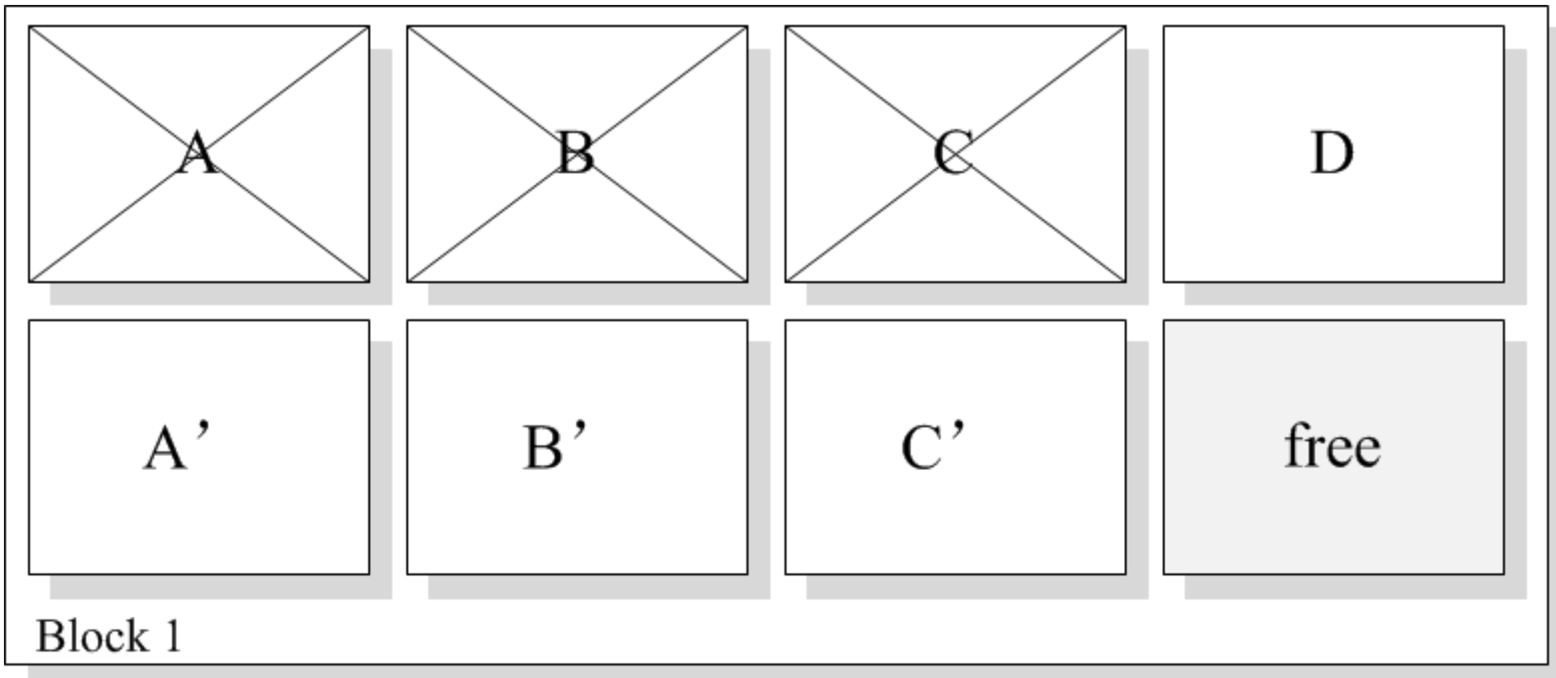
The Page Writing Process



(Courtesy of F. Jiang)

Data items A, B and C can be written to fresh pages

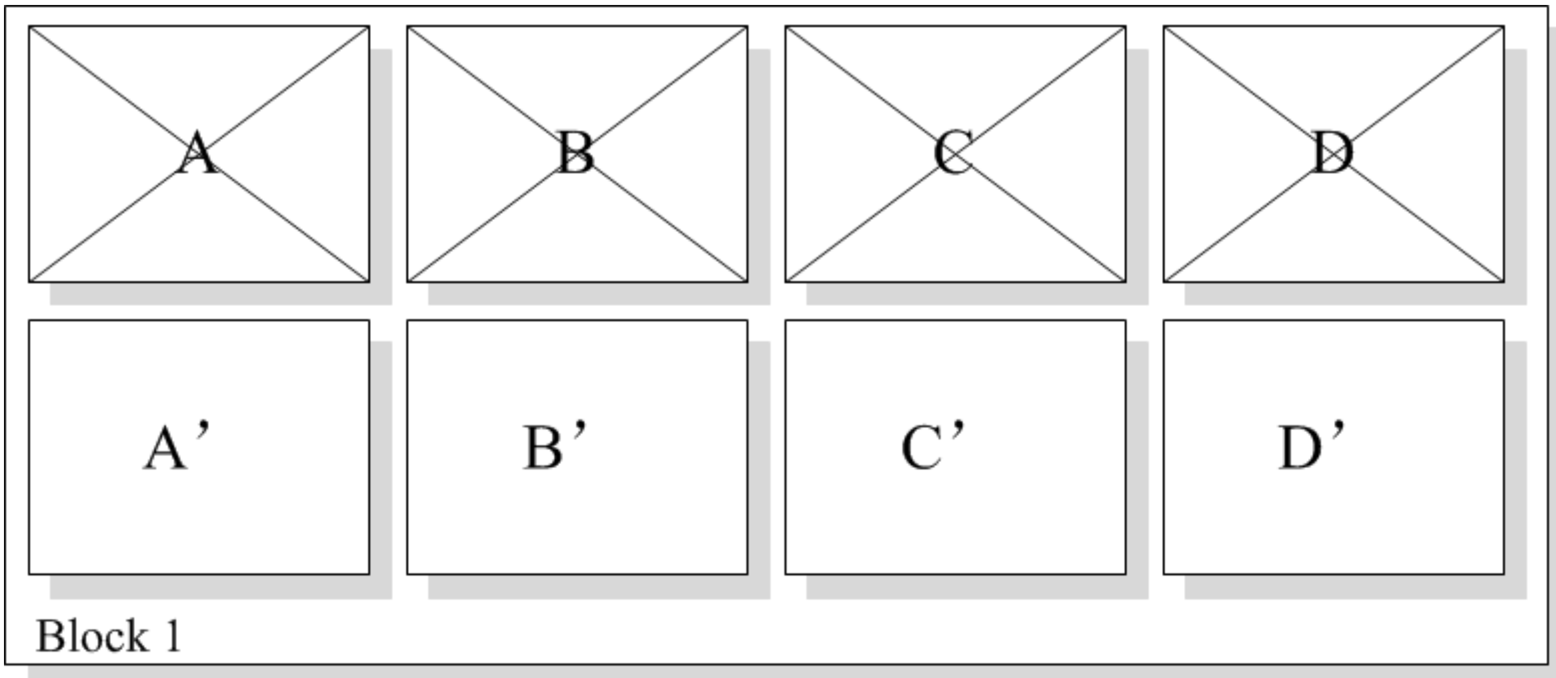
The Page Writing Process



(Courtesy of F. Jiang)

A new page gets the data item D and data items A, B and C are updated
Thus the old pages are invalidated and fresh pages are consumed

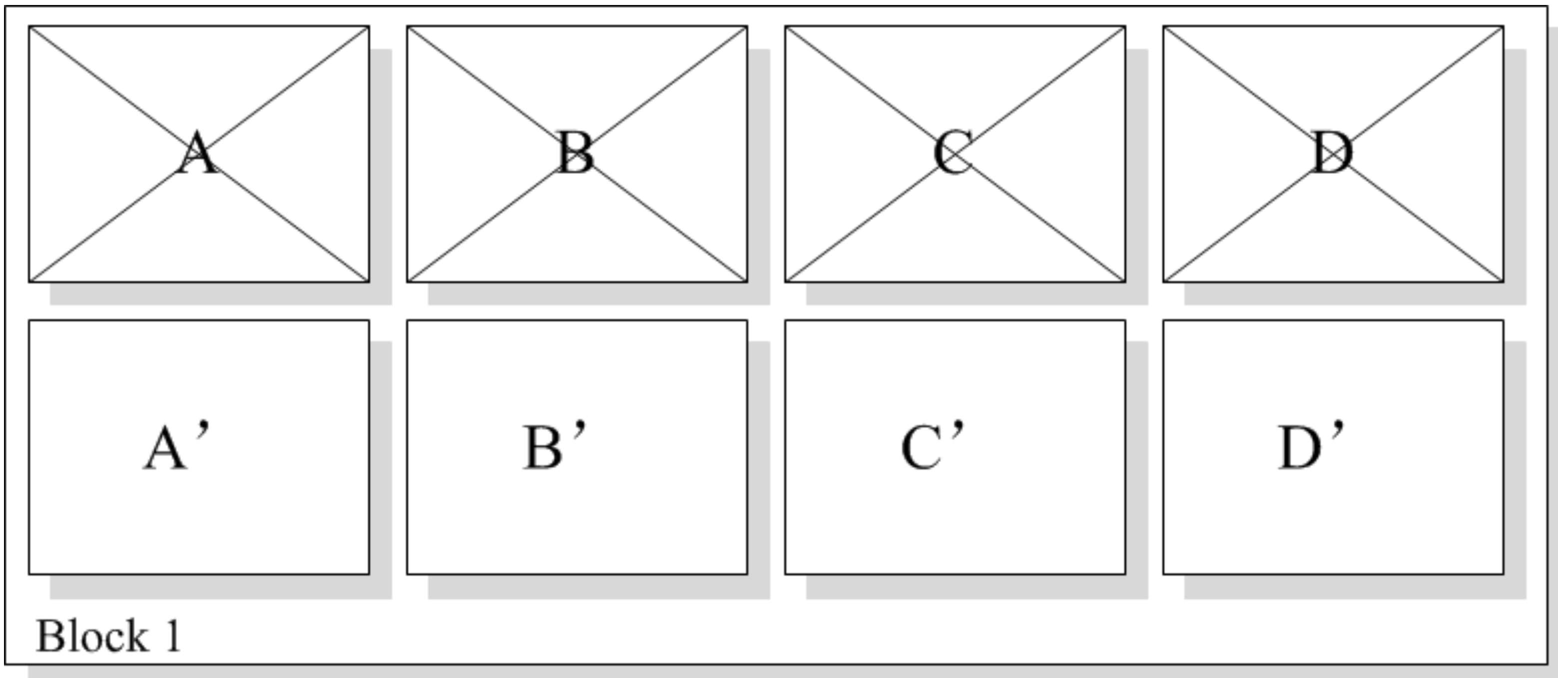
The Page Writing Process



(Courtesy of F. Jiang)

When D is updated, this block will have no more fresh pages, thus no new data item can be programmed into it

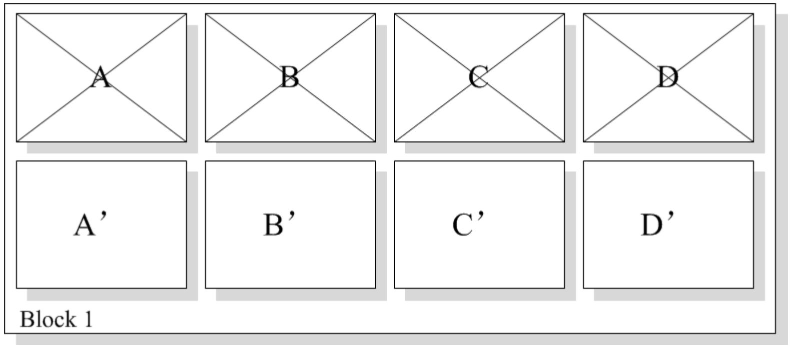
The Page Writing Process



(Courtesy of F. Jiang)

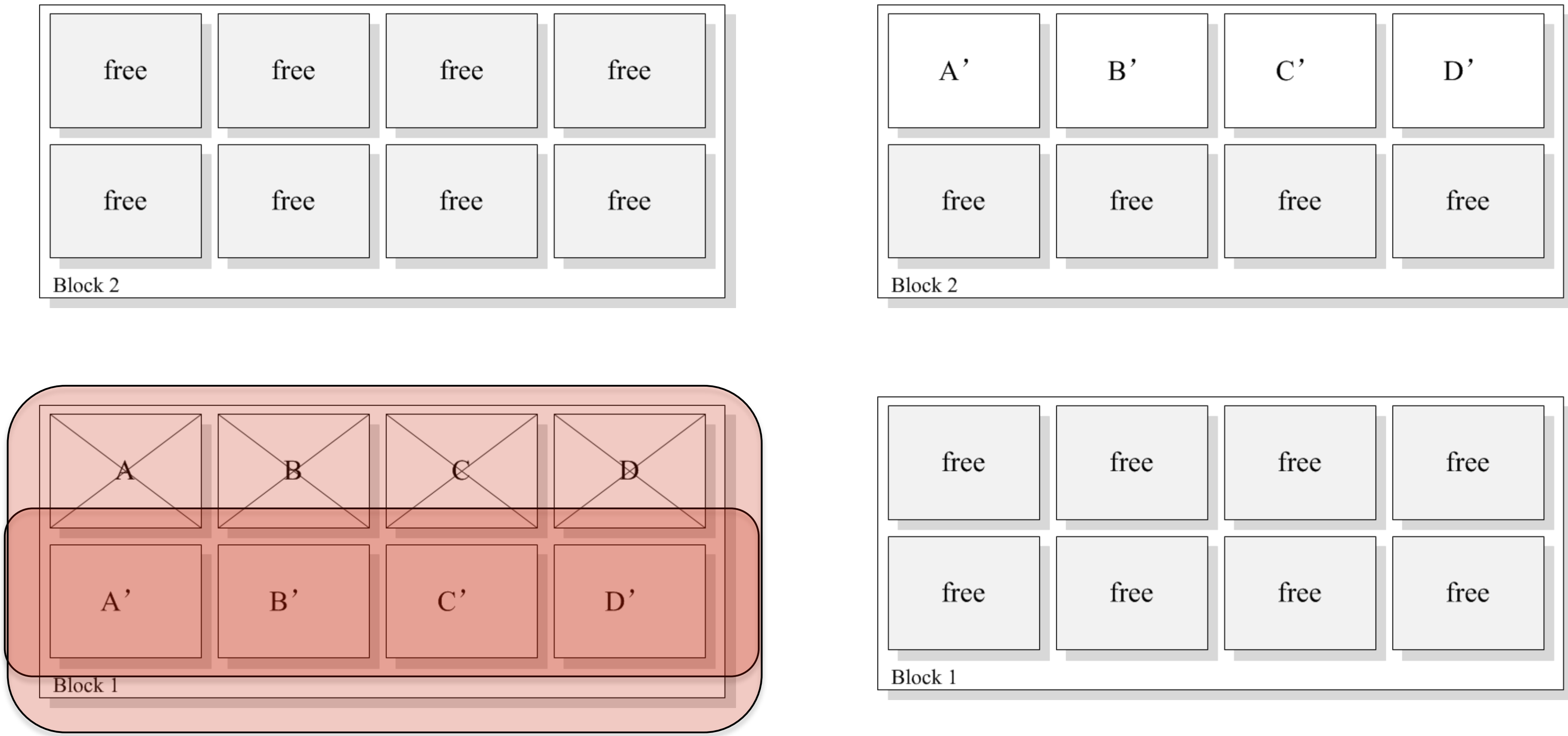
When a sufficient number of pages in a block are invalidated (e.g., 50%) a *garbage collection* process takes place

Garbage Collection



Garbage Collection

New block with fresh pages



Wear Leveling

- Every time a block is written its lifetime is decreased
- *Wear leveling* aims at minimizing this effect by swapping intensely-used blocks with rarely-used ones
- This requires rewriting blocks, which is expensive

Write Amplification

- Both the garbage collection and the wear leveling cause extra writes on disks
- The amount of actual (and relatively slow) *physical* writes on flash disks is thus much larger than the amount of *logical* writes from disk manager

Writing to an SSD may be problematic, but they are faster to read than HDs …

HDs vs (?) SSDs

- Hybrid architectures
  - Concurrent use of HDs and SSDs
  - One can explore the strengths offered by HDs (SSDs) in order to minimize the weaknesses of SSDs (HDs)
    - Different (or not) architectural level

Hybrid architectures

- Concurrent use of HDs and SSDs (1)
  – HDDs and SSDs at the same level in the storage hierarchy
  – Placement of incoming data is determined by the workload on the data
    - Read-intensive data will be placed on the SSD and write-intensive data will be placed on the HDD.
    - If the workload changes, pages might migrate between disks

Hybrid architectures

- Concurrent use of HDs and SSDs (2)
  - HDDs and SSDs at different levels in the storage hierarchy
  - HDs as "write cache", flushed when full onto SSDs
    - Lots of sequential writes are "OK" on SSDs
    - Potential use: writing DBMS log files
  - SSDs as "read cache" (slower than main memory but potentially much larger)

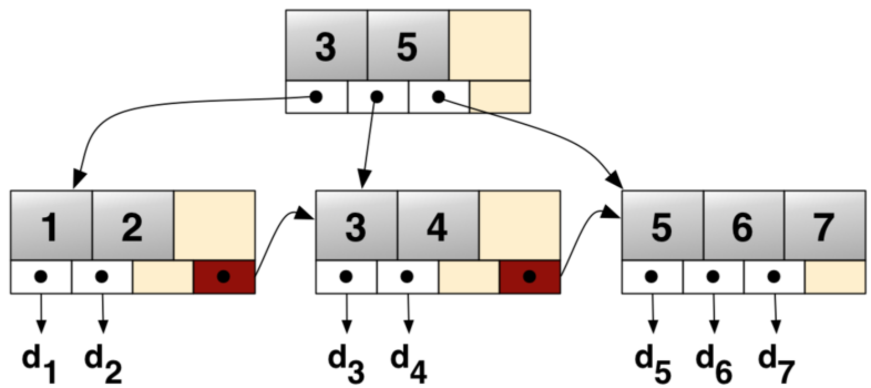
Outline

- Introduction
- Physical Storage
  - HDs and SSDs
- Revisiting Fundamental DBMS Techniques and Algorithms
  - Indexing
  - Join Processing
  - Query optimization
  - Caching
  - Logging

Outline

- Introduction
- Physical Storage
  - HDs and SSDs
- Revisiting Fundamental DBMS Techniques and Algorithms
  - Indexing
  - Join Processing
  - Query optimization
  - Caching
  - Logging

B+-tree



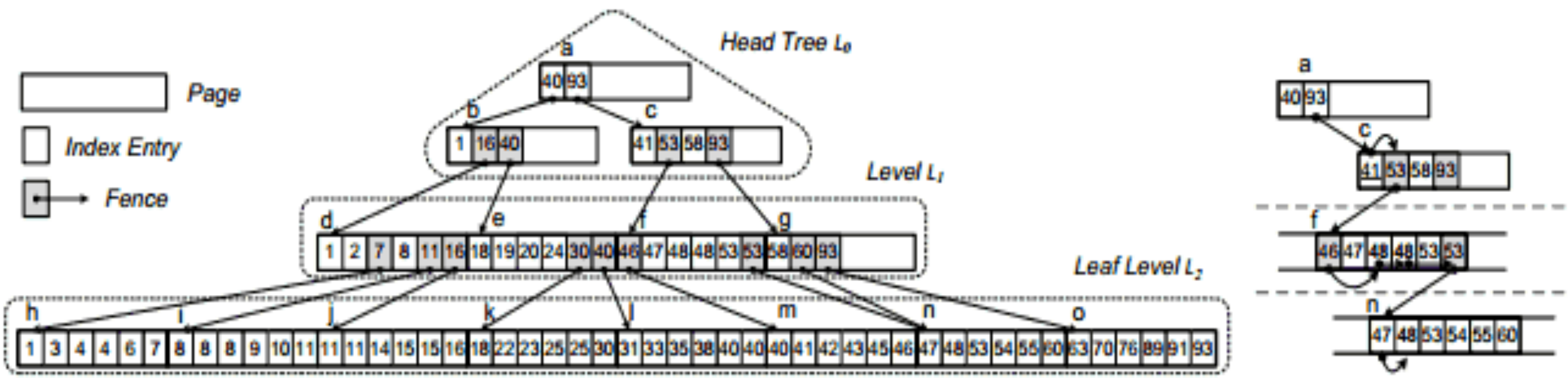
http://en.wikipedia.org/wiki/B+_tree

- Fast random access makes it attractive for indexing trees
- BUT … tree nodes split
  - the expensive writes are a potential problem

FD-Tree (ICDE 2009)

- Due to the B-tree's logarithmic nature, a few upper levels of the tree are enough to hold a lot of information
  - Keep it (the tree's upper levels) in main memory
  - Buffer and arrange all writes so that they can performed sequentially

FD-tree



(a) The overview of the example FD-tree

(b) Searching $key = 48$

Li et al, ICDE 2009

- Insertion
- Search
- Deletion

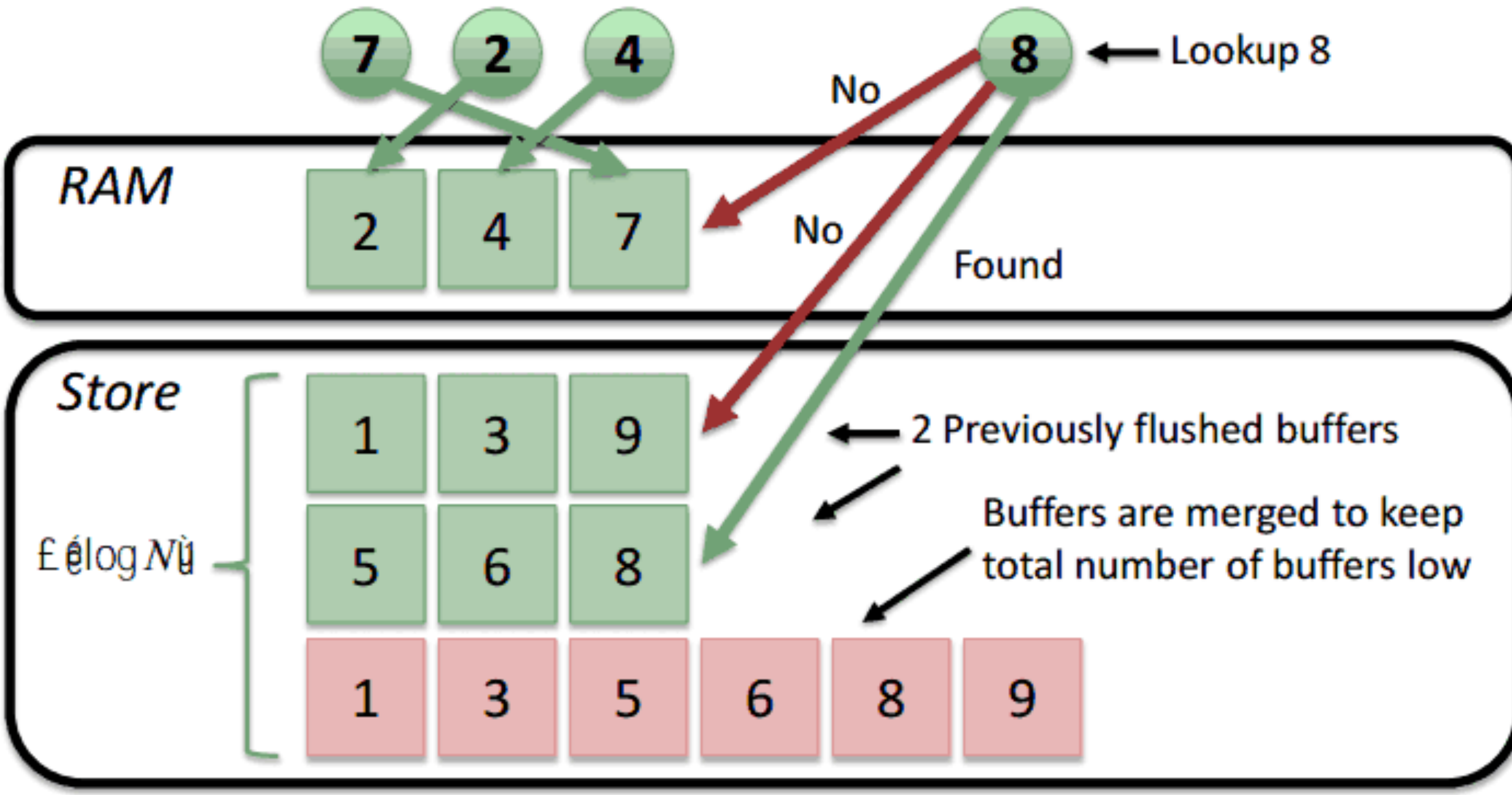
Hashing

- Offers nearly constant access time during searches, which is good
- Makes use of random and uniformly distributed writes on the hash table, which is *not* good
- Relatively speaking, less work has been done on “Hash on Flash”

“Flashing” Bloom Filters

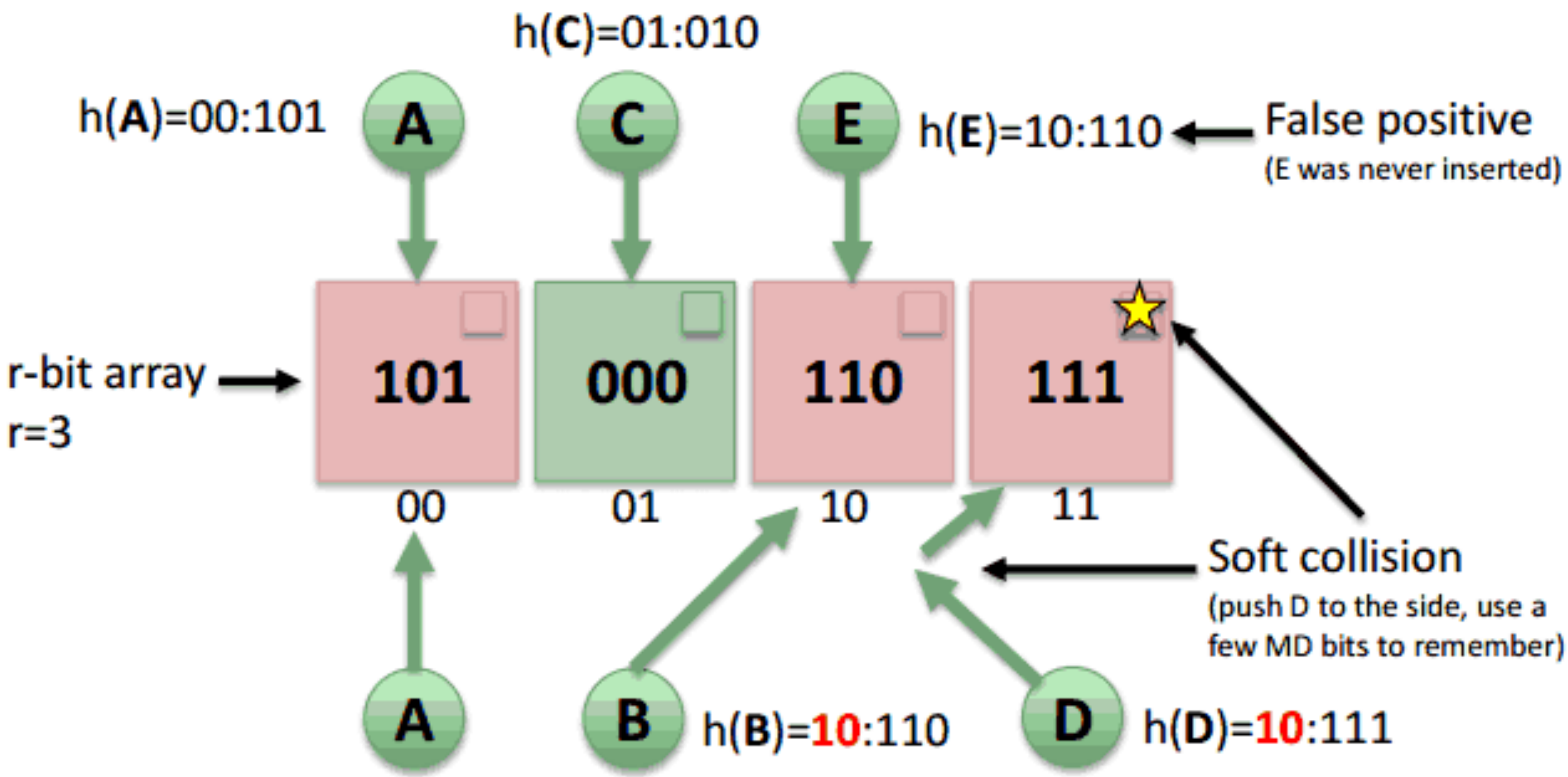
- Recent work [VLDB 2012] proposed to address the random writes issue on hash tables by using cleverly using:
  - Buffered Quotient Filters
  - Cascade Filters

General Idea



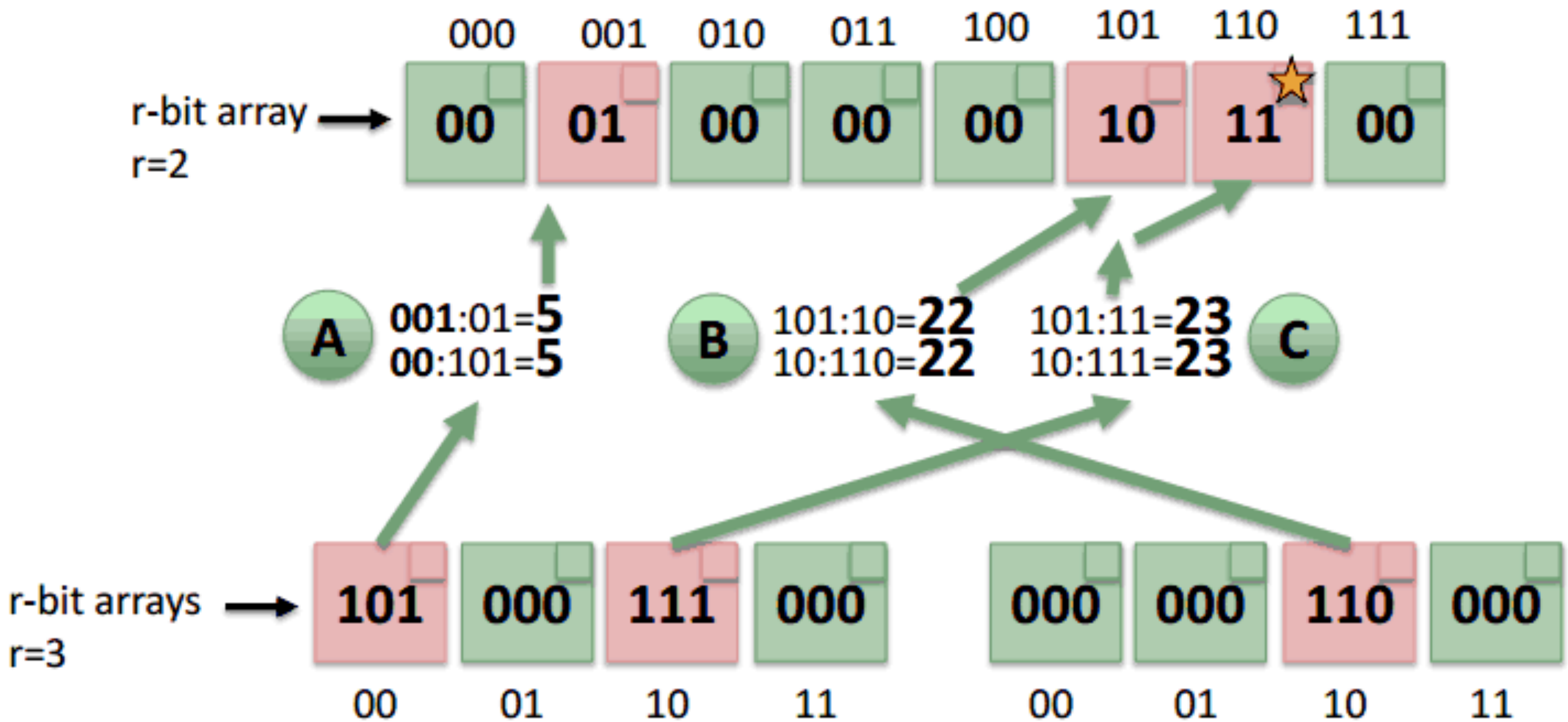
http://www.usenix.org/events/hotstorage11/tech/slides/bender.pdf

Quotient Filter



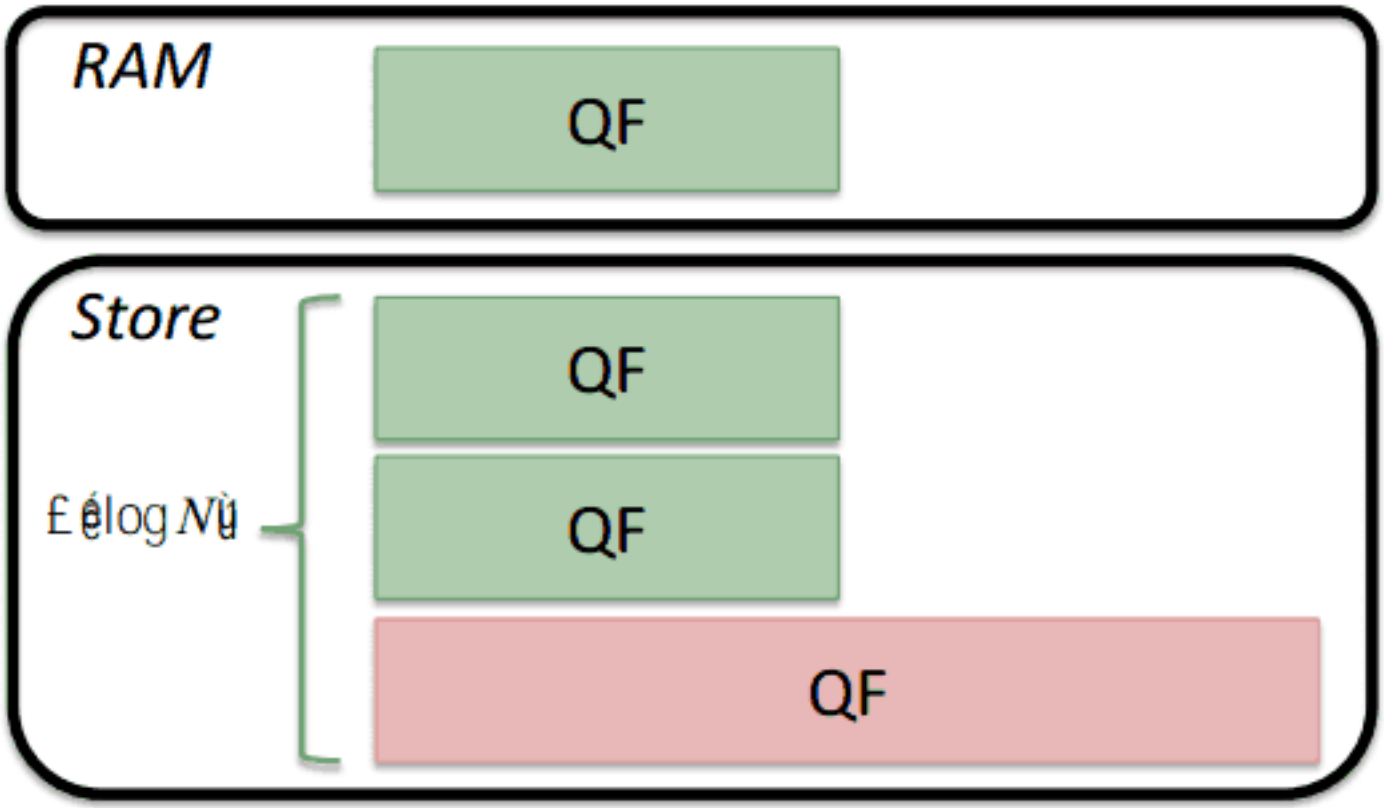
http://www.usenix.org/events/hotstorage11/tech/slides/bender.pdf

Merging Quotient Filters



http://www.usenix.org/events/hotstorage11/tech/slides/bender.pdf

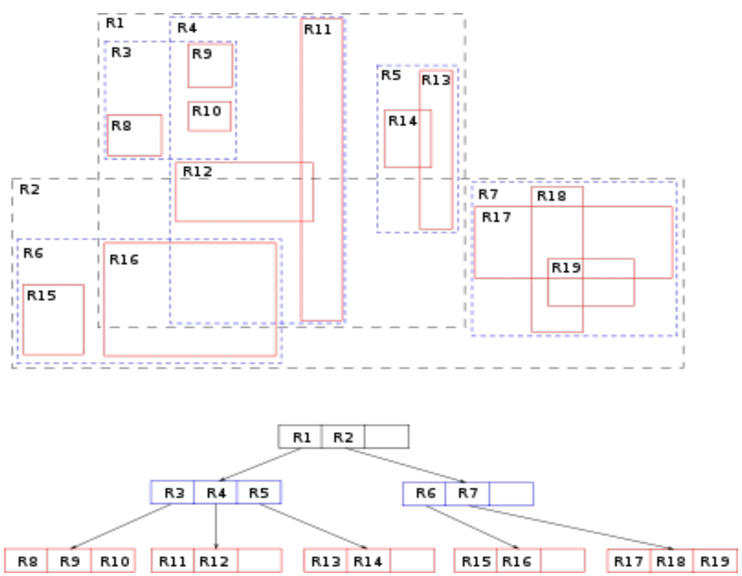
Cascade Filters on Flash

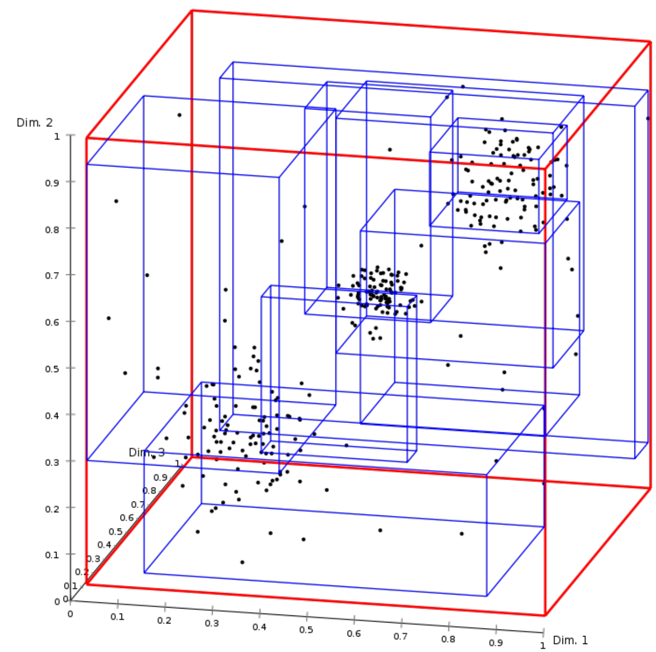


http://www.usenix.org/events/hotstorage11/tech/slides/bender.pdf

R-tree (and its variants)

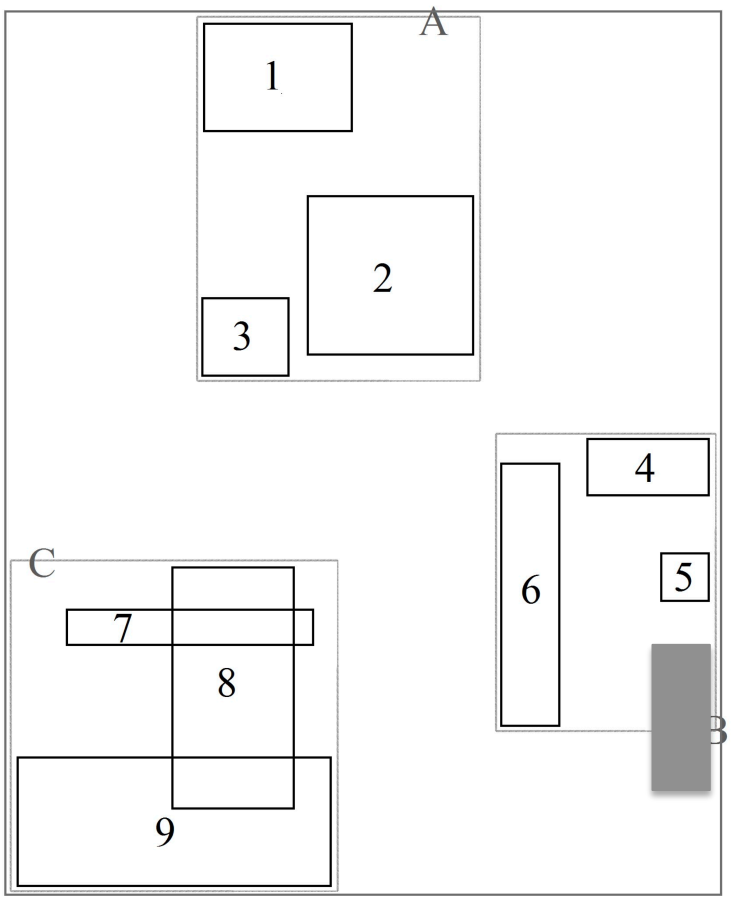
- The *de facto* indexing structure for spatial data

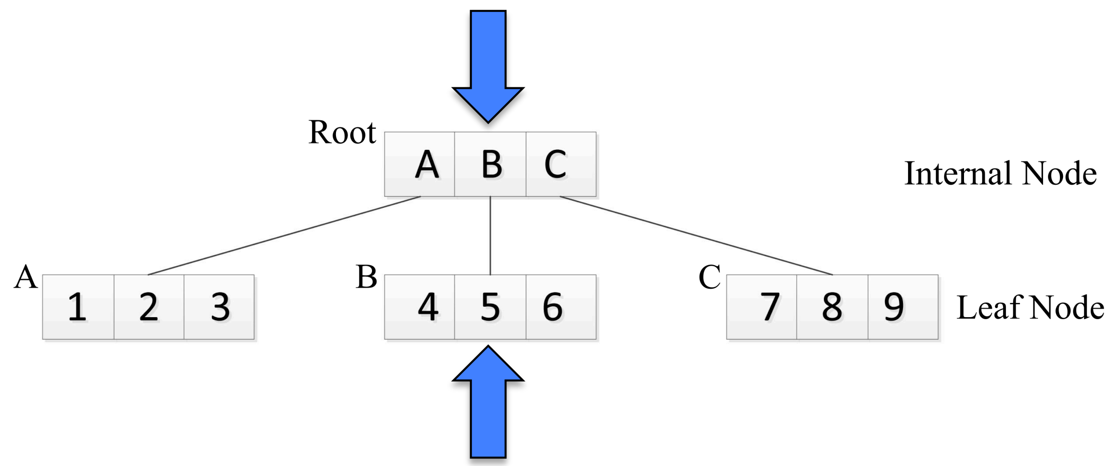




http://en.wikipedia.org/wiki/R-tree

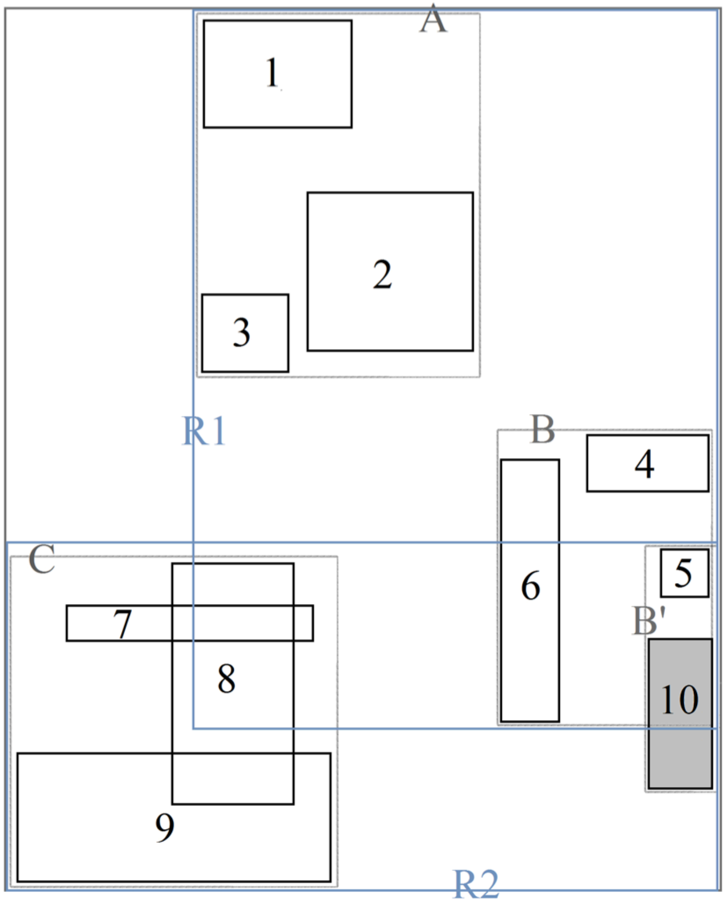
R-tree splits

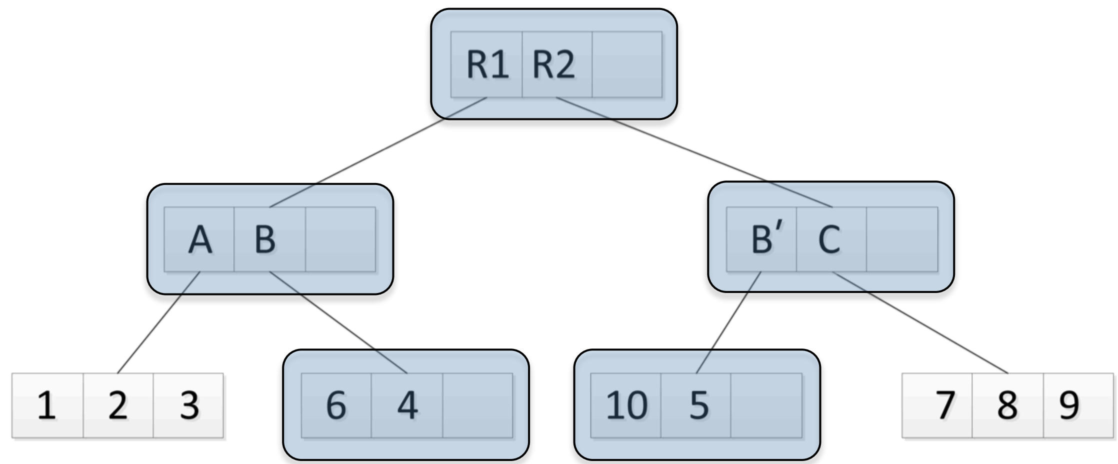




(Courtesy of F. Jiang)

R-tree splits

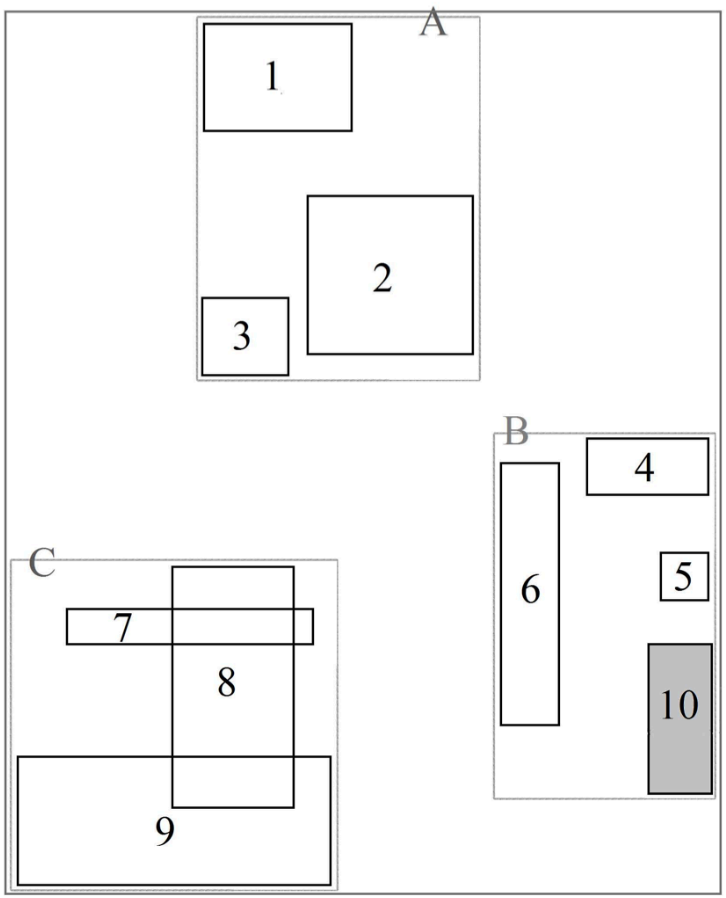


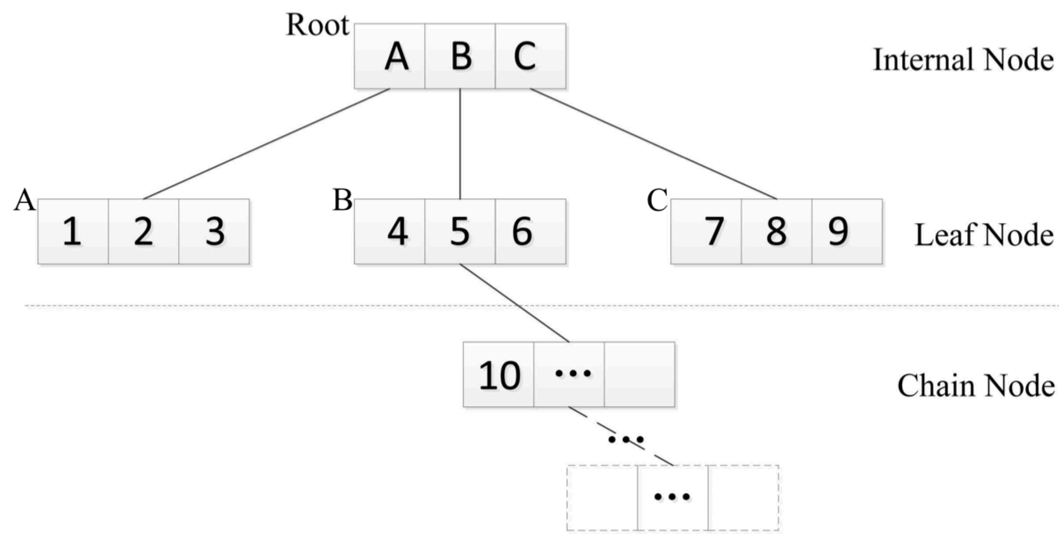


(Courtesy of F. Jiang)

5 writes: 2 for splitting the leaf node, 2 for splitting the parent node, and 1 for the root node.

FAR-tree*

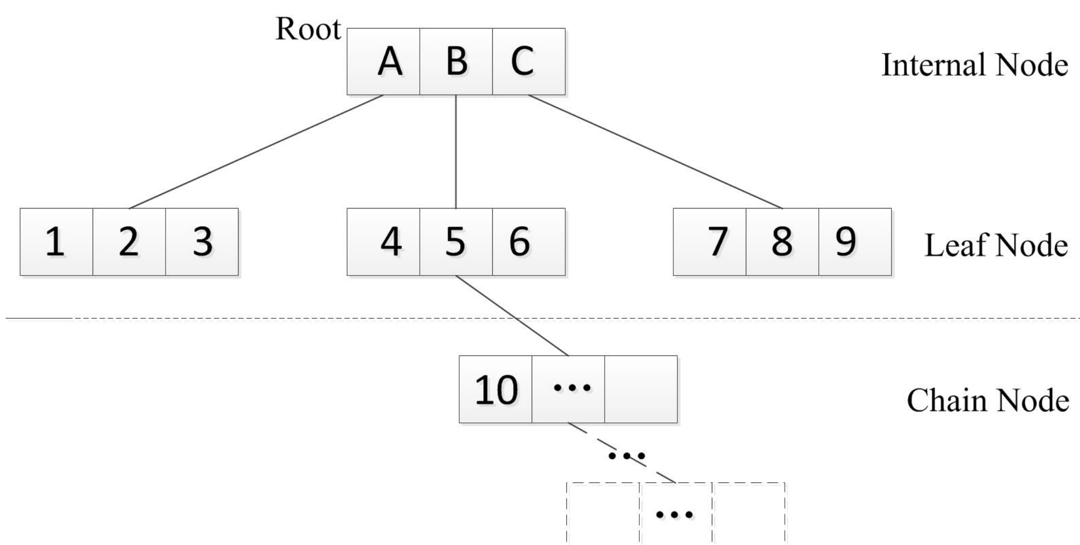




2 writes: 1 for writing the chain node, and 1 for updating the pointer of node B.

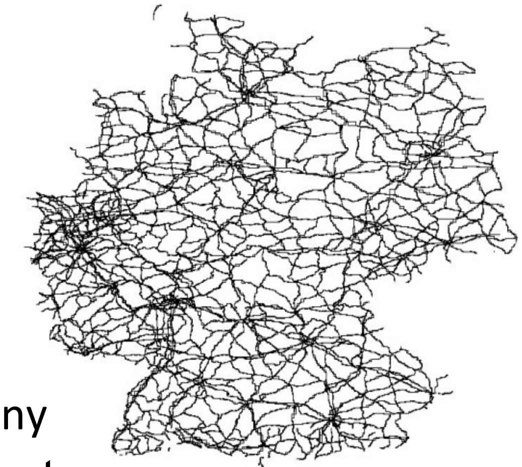
Rebalancing the FAR-tree

- Collect entries in chain nodes
- Re-insert them in the tree
  - Still many writes but likely not as many as were deferred

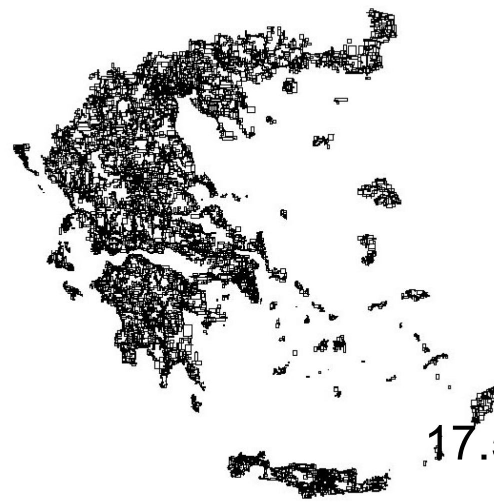


Evaluation ("preview")

- Datasets:



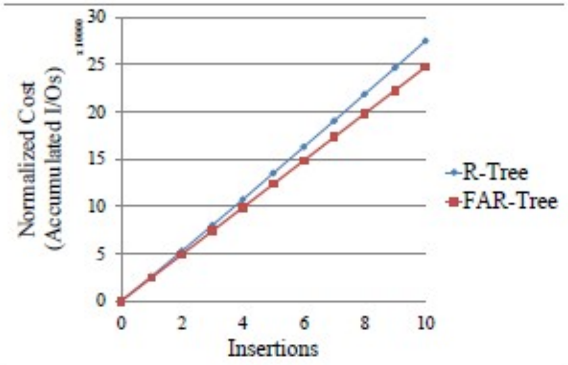
Germany
25k objects



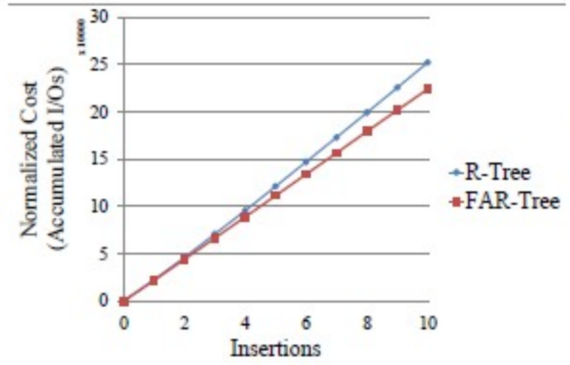
Greece
17.5k objects

- Performance metric: #reads + R x #writes
  - R reflects how slower a write op is wrt a read op

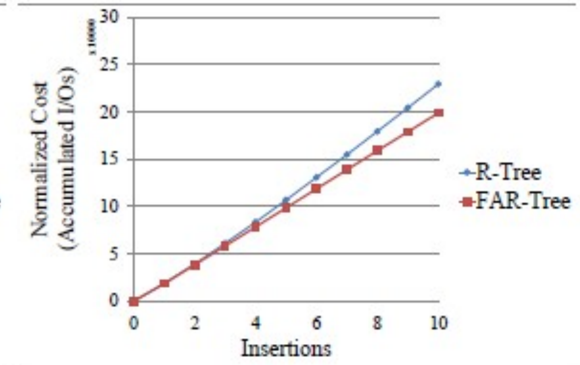
Insertion Cost



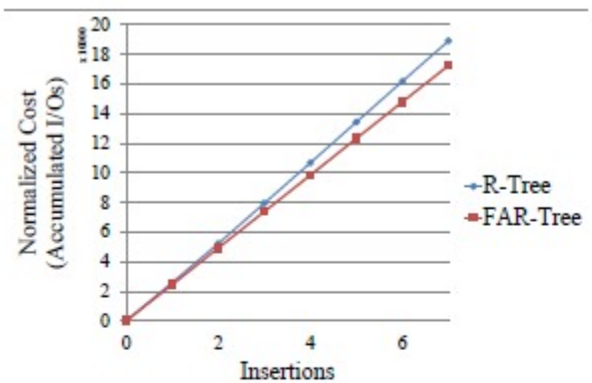
(a) B=10



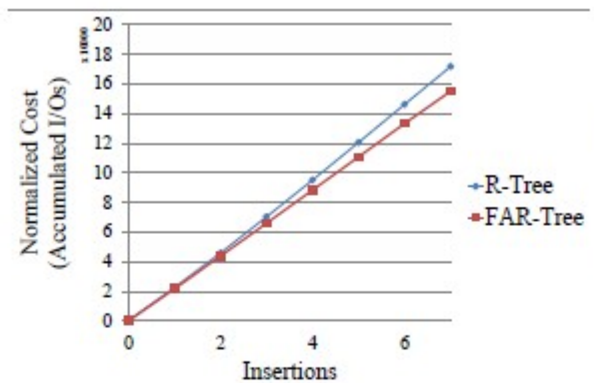
(b) B=25



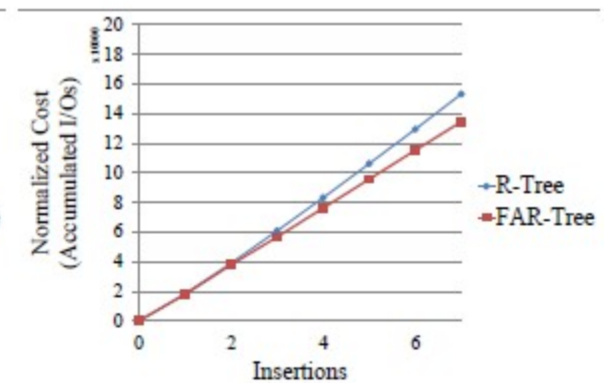
(c) B=50



(a) B=10

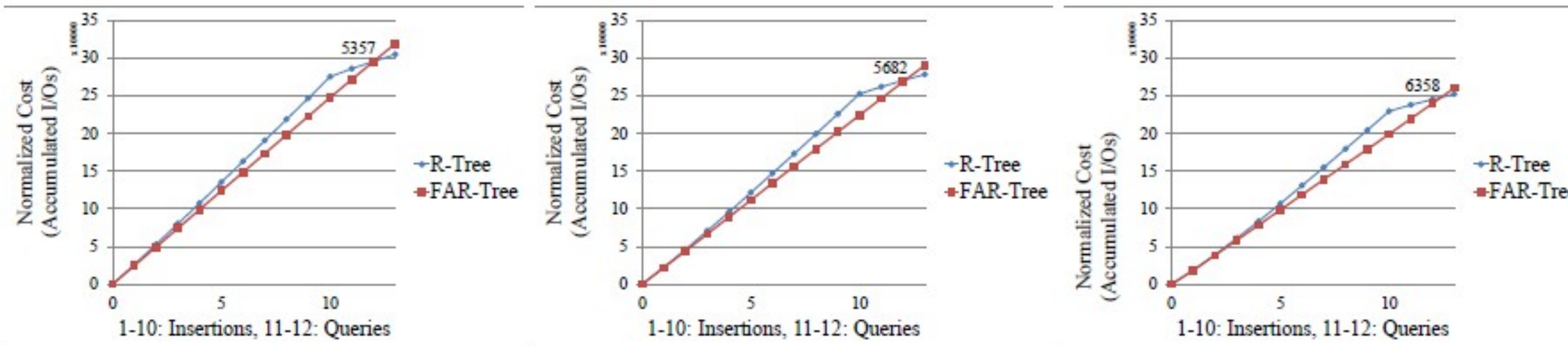


(b) B=25

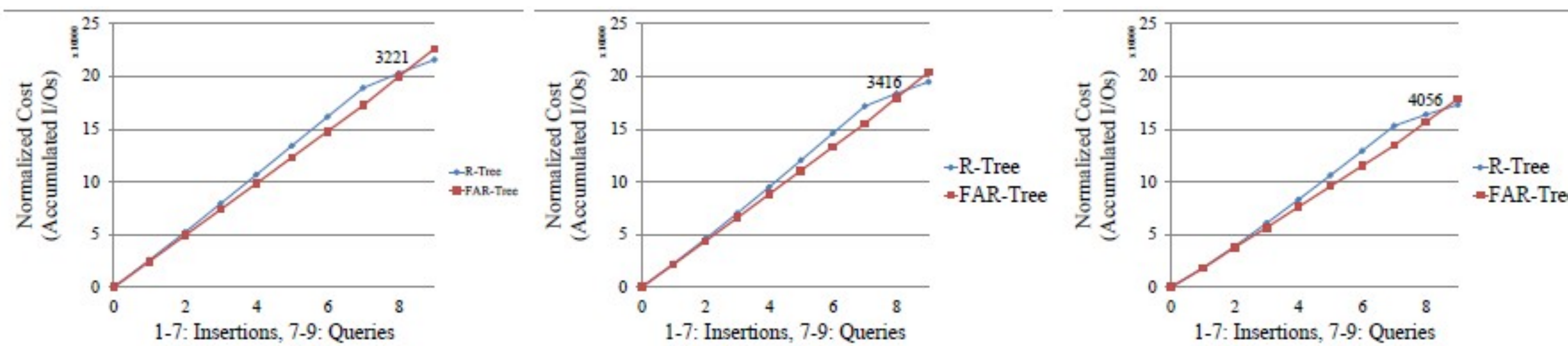


(c) B=50

Query Cost

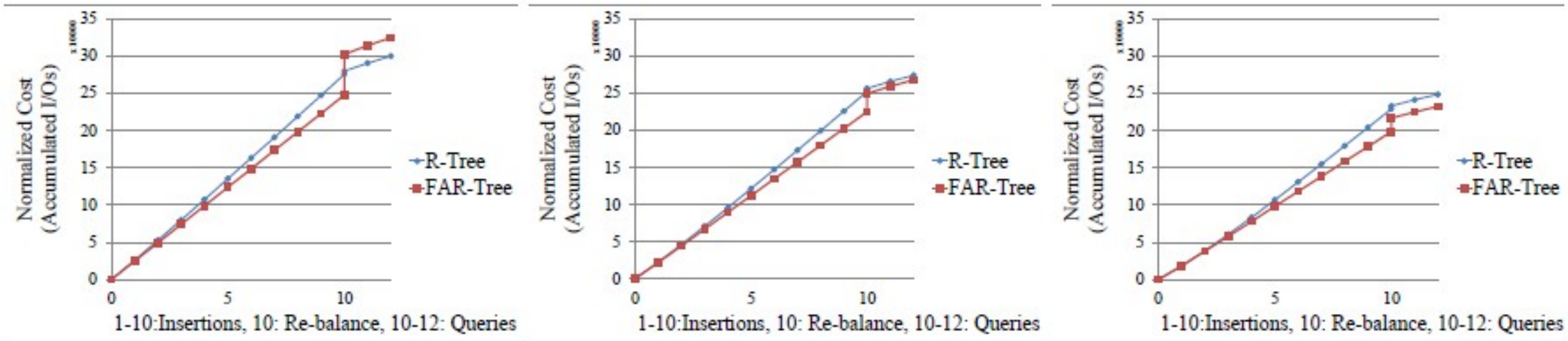


(a) B=10 (b) B=25 (c) B=50

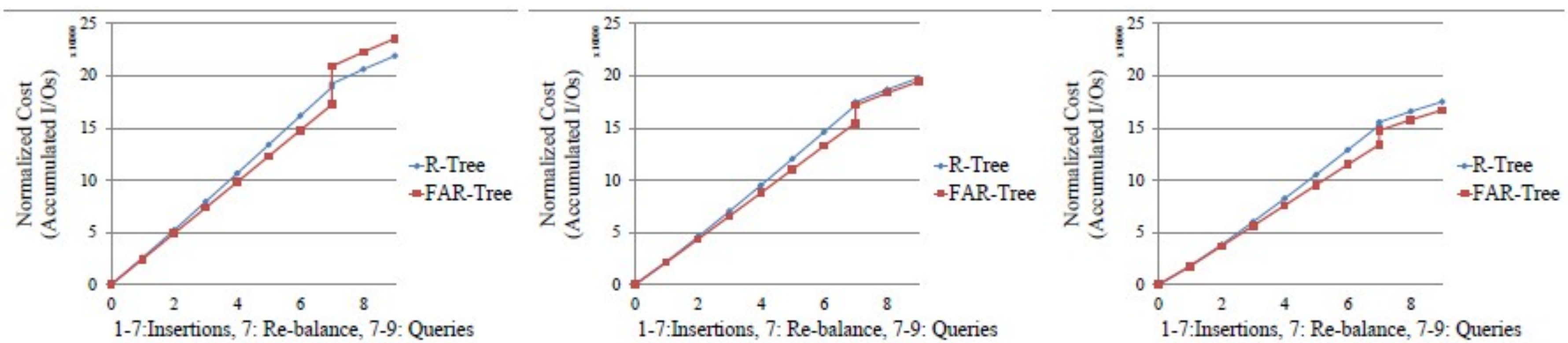


(a) B=10 (b) B=25 (c) B=50

Rebalancing



(a) B=10 (b) B=25 (c) B=50



(a) B=10 (b) B=25 (c) B=50

Some Conclusions

- The FAR-Tree did reduce the number of disk writes during insertions
- The chains may result in more disk reads when searching the index
- The re-balancing overhead was diminished as it utilizes the buffer well
  - The end result is a balanced R-tree
- Query processing time followed the same trend as query processing I/O

End of Part 1

- We have seen:
  - Why SSDs are attractive for replacing HDs within DBMSs
  - SSDs' architecture and the R/W asymmetry (major issue for DBMSs)
  - How indexing can be adapted to be efficiently used with SSDs
- Next:
  - Other DBMS techniques and algorithms on SSDs

Acknowledgements

- NSERC Canada for research support
- SBBD’s Organization for hosting us
- Many colleagues for discussions and Feng Jiang also for many of the figures used here.

References (selected)

[ICDE 2009] Y.Li et al: Tree Indexing on Flash Disks. ICDE 2009: 1303-1306

[SIGMOD 2011] I.Koltsidas and Stratis Viglas: Data management over flash memory (Tutorial abstract). SIGMOD 2011: 1209-1212

[SSTD 2011a] I. Koltsidas and S. Viglas: Spatial Data Management over Flash Memory. SSTD 2011: 449-453

[SSTD 2011b] M.Sarwat et al: FAST: A Generic Framework for Flash-Aware Spatial Trees. SSTD 2011: 149-167

[VLDB 2012] M.A. Bender et al: Don't Thrash: How to Cache Your Hash on Flash. PVLDB 5(11): 1627-1637

Obrigado pela atencao

- Angelo Brayner

  Univ. of Fortaleza, Brazil

  brayner@unifor.br

- Mario A. Nascimento

  Univ. of Alberta, Canada

  mario.nascimento@ualberta.ca

Questions

?