Database Access Methods

Lecture 3

September 26, 2006

Plan for today

- Questions from last time
- Relational Database Management Systems
  - Basic concepts and architecture
- Relations
  - storage
- Data-layout
  - On-disk
  - In-memory
- PAX paper discussion

Tables (Relations)

R1

RID	SSN
1	1237
2	4322
3	1563
4	7658
5	2534
6	8791

Payload for Q1

R2

RID	Name
1	Jane
2	John
3	Jim
4	Suzan
5	Leon
6	Dan

R3

RID	Age
1	30
2	45
3	20
4	52
5	43
6	37

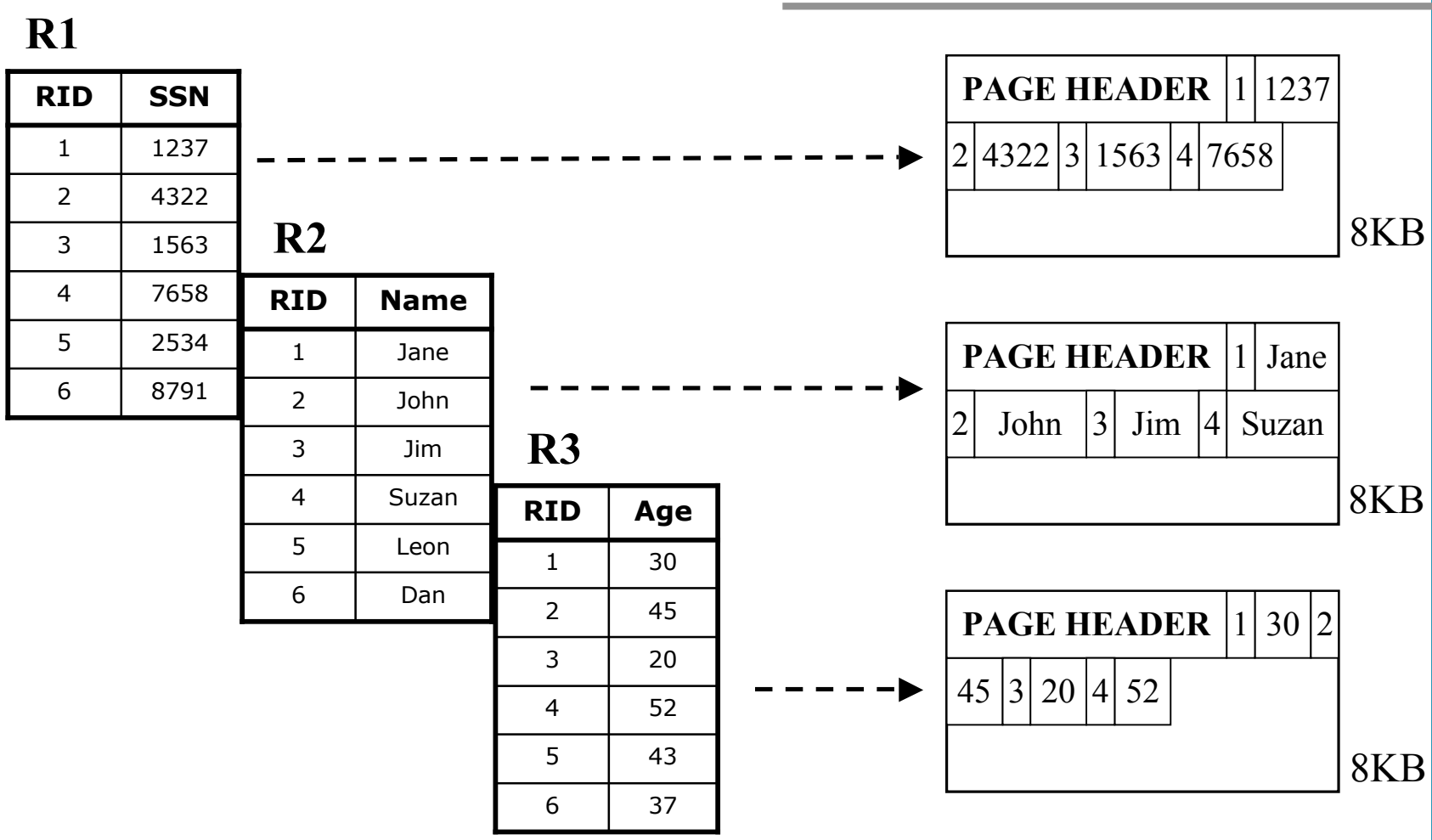
Table Scan: (Q1)

SELECT SSN FROM R1

Join + Table Scan (Q2):

SELECT R3.AGE, R2.NAME WHERE R3.RID = R2.RID

Tables (Relations) and Files



Each table can be a file

page size = file system block size

Overview of a DBMS Architecture

Optimizer

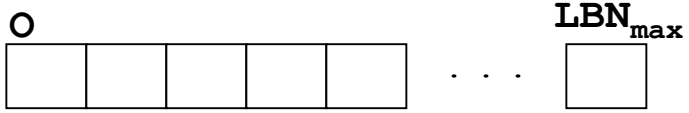
- chooses query plan with the lowest cost
  - sequential vs. random access
- decoupled from info about execution state

Storage Manager

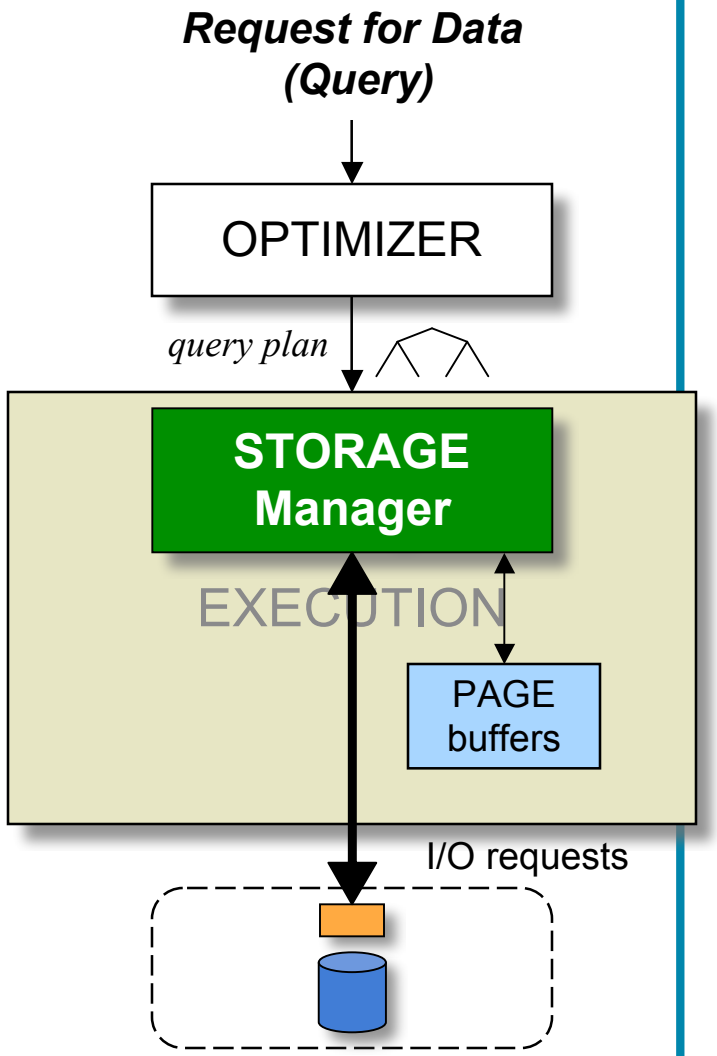
- optimizes I/Os after access patterns are determined by the optimizer
  - prefetching for sequential access
  - caching pages in buffers

Storage Device

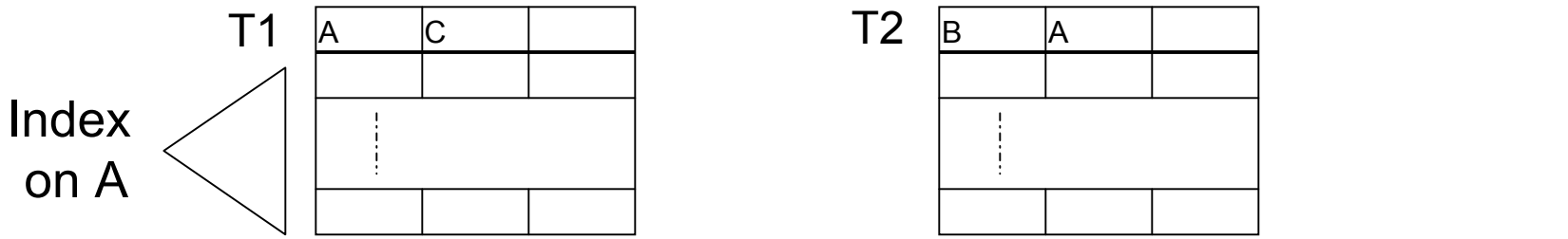
- linear address space of logical blocks



- but access time is non-linear…



Anatomy of a Query



SELECT T1.A,T2.B FROM T1,T2 WHERE T1.A = T2.A AND T1.A<4

relations

join predicate

predicates

SARGABLE predicate

- Operations
  - "filter" out data using SARGABLE predicate
  - Join the two data streams on the join predicate
    - Nested-loop join
    - Sort-merge join
- Index of T1.A
  - Can access a subset of data using index rather than scan T1

"Operation" Costs

- Determinte the *efficiency of* access to data
  - TBLScan
    - sequential access
  - IDXScan
    - non-clustered access via index is inefficient
  - Sort
- Use "efficiency" to annotate query tree
  - consider I/O cost as well as efficiency of access
  - alleviates sensitivity to selectivity factor F

Cost Estimators

- Relational Statistics
  - NCARD(T)
    - cardinality of relation (“number of rows in a table”)
  - TCARD(T)
    - number of pages with T’s data (“datafile layout”)
  - P(T) = TCARD(T) / non-empty pages a segment
    - “sparse”/”dense” allocation
- Index Statistics
  - ICARD(I)
    - number of distinct keys
  - NINDX(I)
    - number of pages in index I

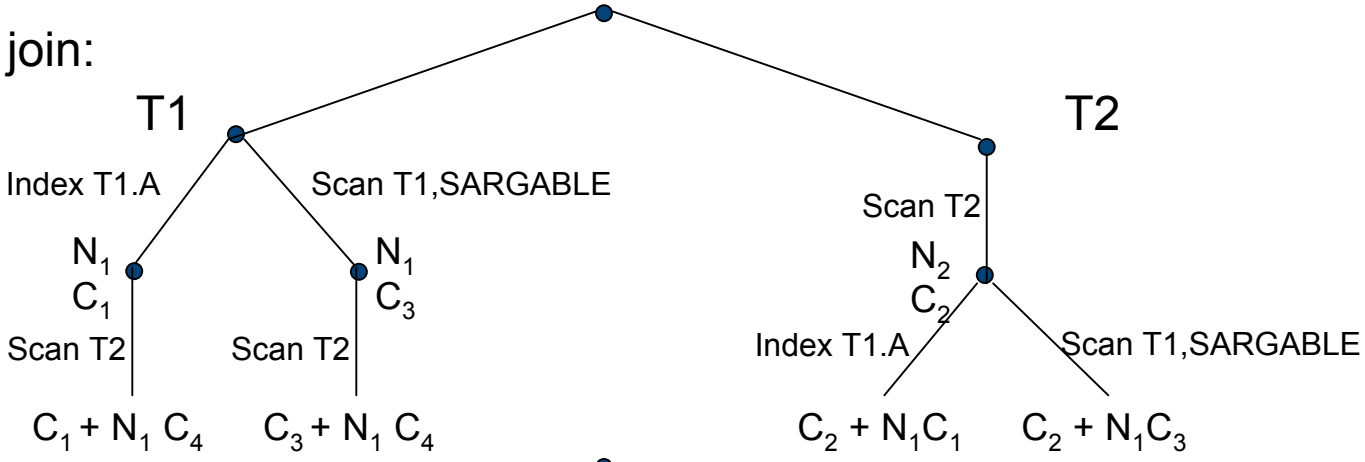
Cost Formulae

- Figure out cost for each relation
  - COST = PAGE FETCHES + w * CALLS
    - determined by the amount of available memory
- Index scan
  - COST = F(preds)*(NINDX(I) + TCARD)+ …
    - selectivity factor F is not easy to get always right
  - clustered vs. non-clustered,
- Relation scan
  - COST = TCARD/P + …
- Combine costs for all relations
  - $C_{outer} = C_{outer} + N * C_{inner}$

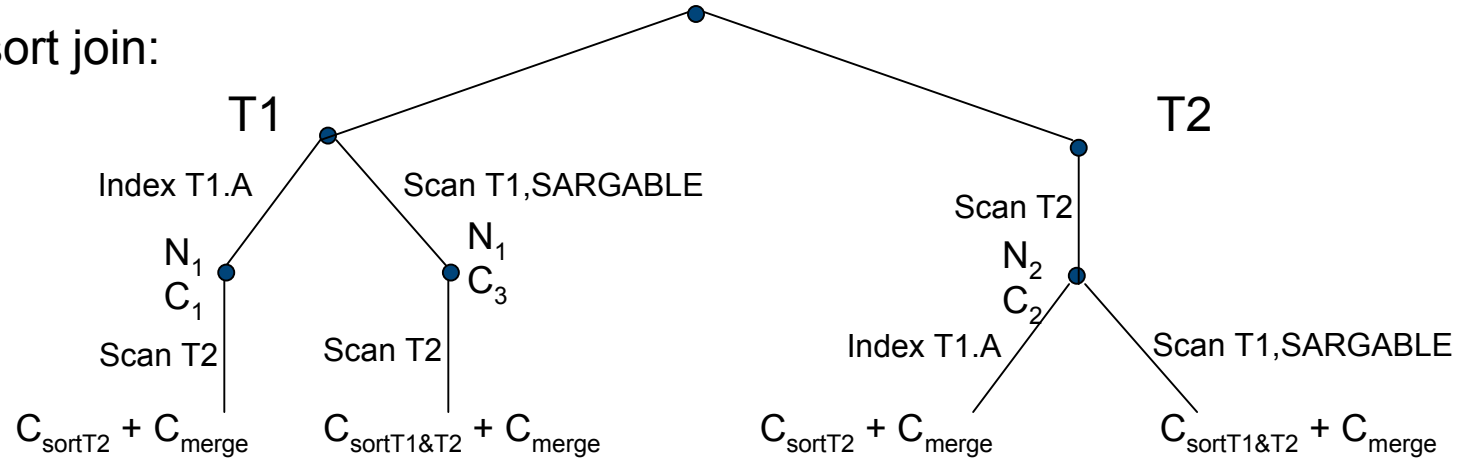
Anatomy of a Query cont'd

SELECT T1.A,T2.B FROM T1,T2 WHERE T1.A = T2.A AND T1.A<4

Nested-loop join:



Merge-sort join:



Current Database Systems

Parser

- translates SQL into intermediate form

Optimizer

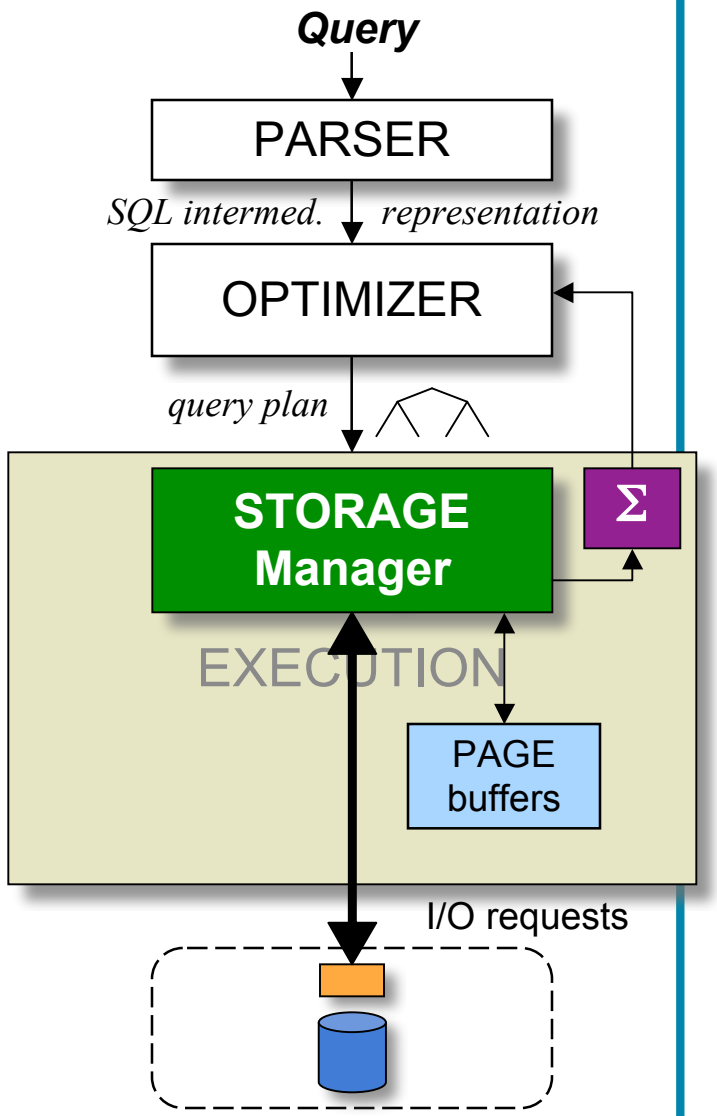
- minimizes total number of I/Os
  - fixed cost for fetching a page
  - cost estimators with statistics
- hands off query plan to execution unit

!

Storage Manager

- optimizes I/O only *after* an access pattern determined by the optimizer
  - prefetching for sequential access
  - caching pages in buffers

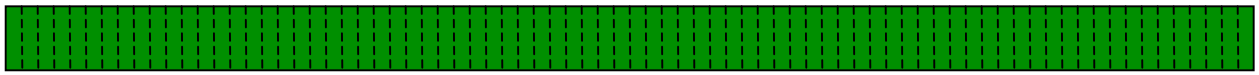
!



PAX Paper Discussion

Scanning a Single Table

- Example – selection query



pages I/Os

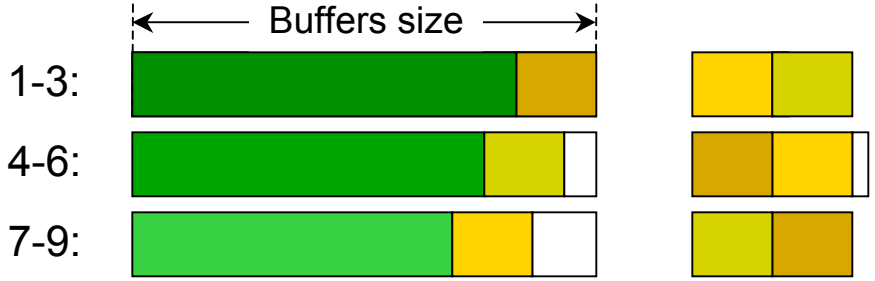
Access in current DB systems

Scanning Two Tables

- Example – join query

Access in current DB systems

- minimize total # of I/Os for a given buffer size



Overview of a DBMS Architecture

Optimizer

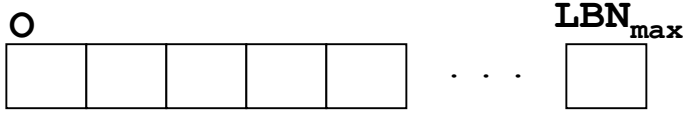
- chooses query plan with the lowest cost
  - sequential vs. random access
- decoupled from info about execution state

Storage Manager

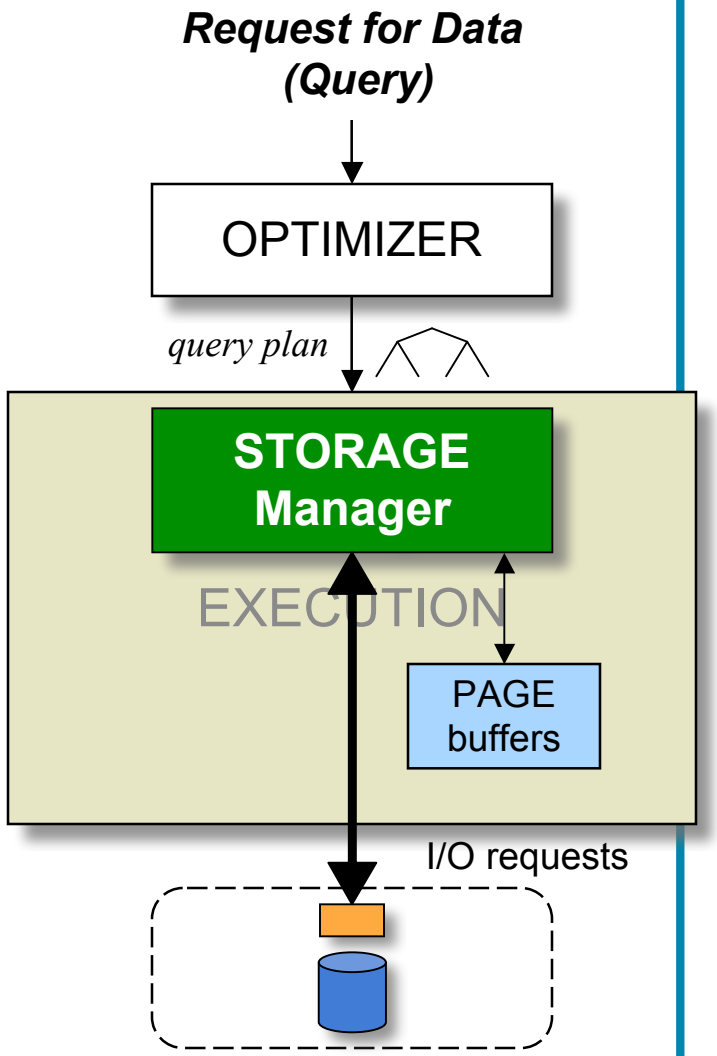
- optimizes I/Os after access patterns are determined by the optimizer
  - prefetching for sequential access
  - caching pages in buffers

Storage Device

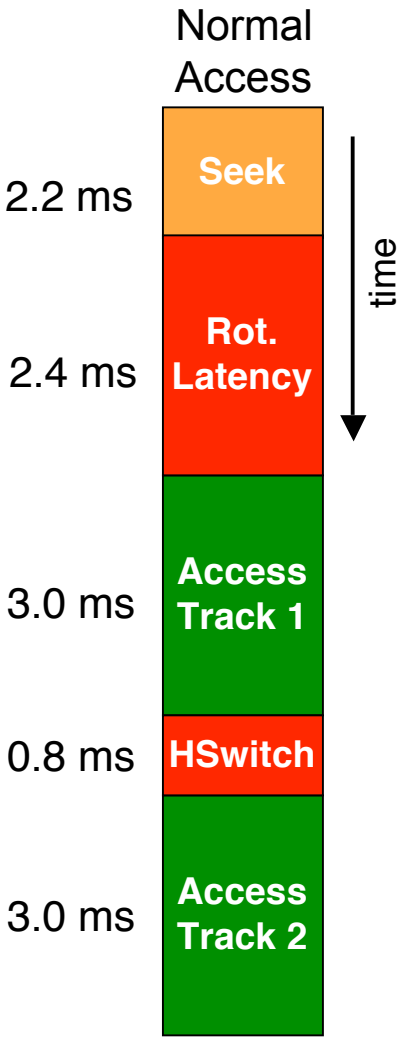
- linear address space of logical blocks

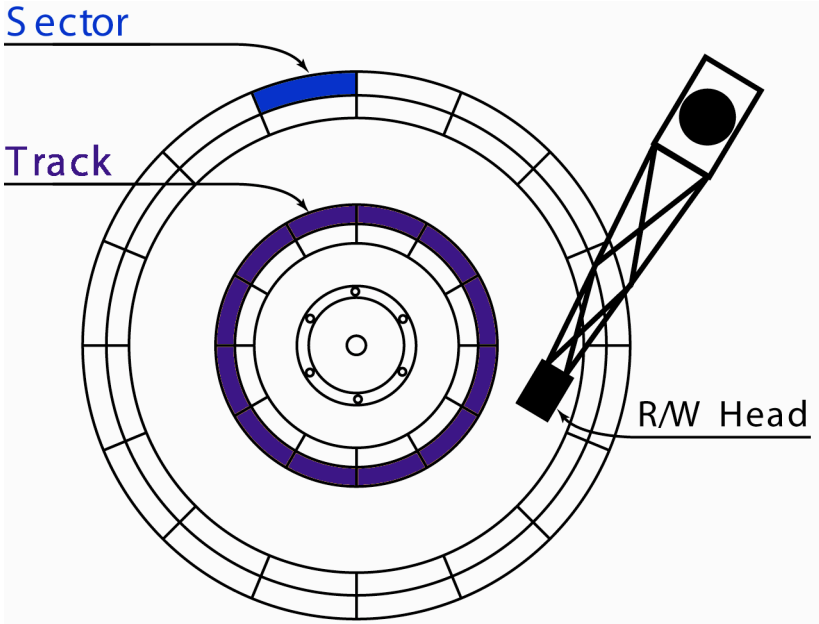


- but access time is non-linear…

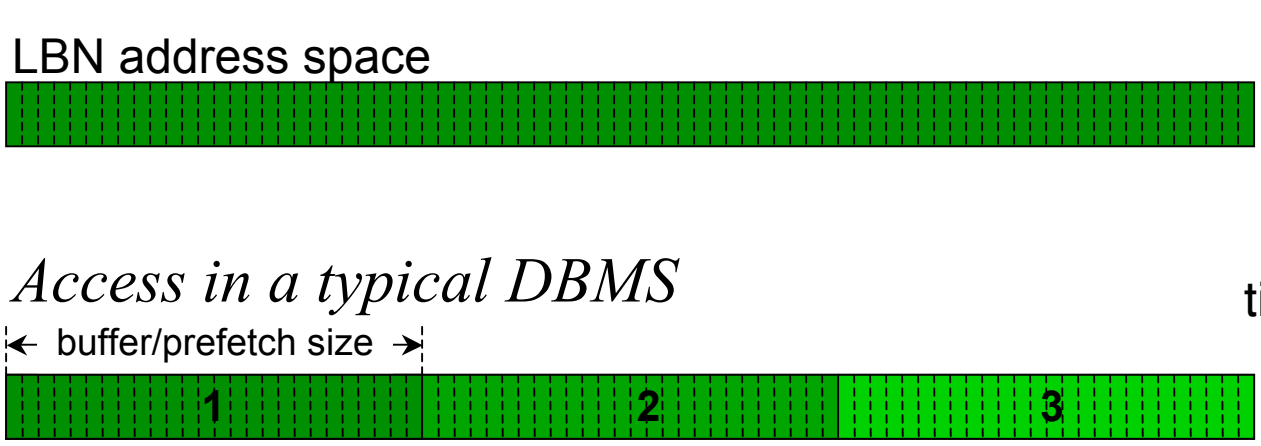


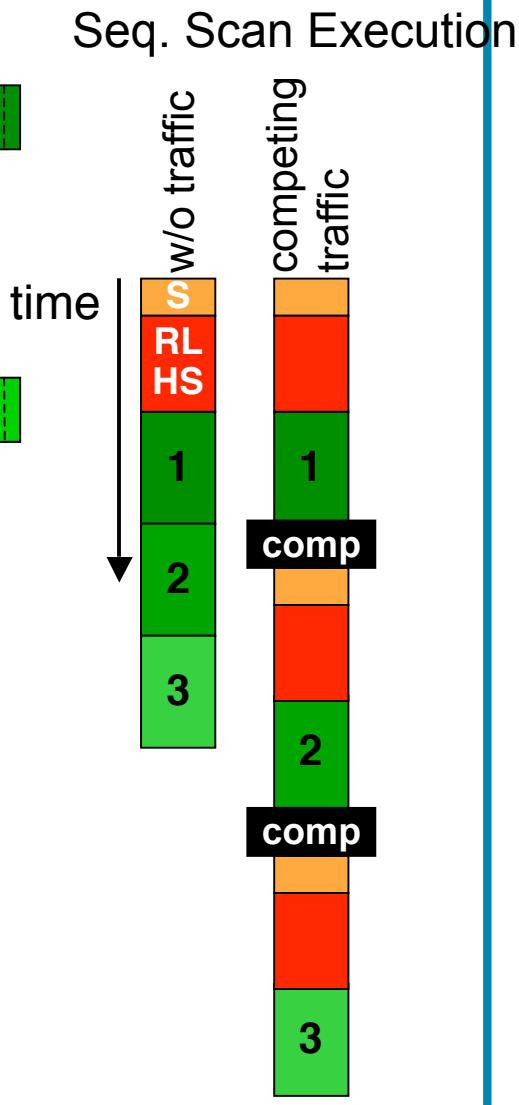
Anatomy of a 264 KB I/O





Execution of a Table Scan Operator





Inefficient with competing traffic

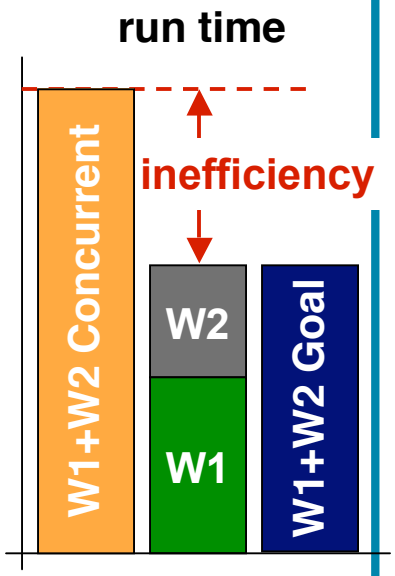
The Fates Project Overview

Efficient query execution at all levels of a database system memory hierarchy

- Exploit unique characteristics at each level
  - L2/main memory
  - storage device
- Request only the data needed by a query
- Clean encapsulation of functionality

Inefficient Use of Storage Resources

- Lots of effort spent in DBMS to improve I/O
  - query optimization techniques
  - manual performance-tuning by DBAs
    - time consuming and expensive ($$$)



- Why is storage used inefficiently?
  - manual configuration is error-prone
  - too much abstracted away from DBMS

Goals

- Make I/O execution efficient for concurrent workloads competing for a storage device
- Eliminate manual I/O performance tuning

(Broken) Assumptions

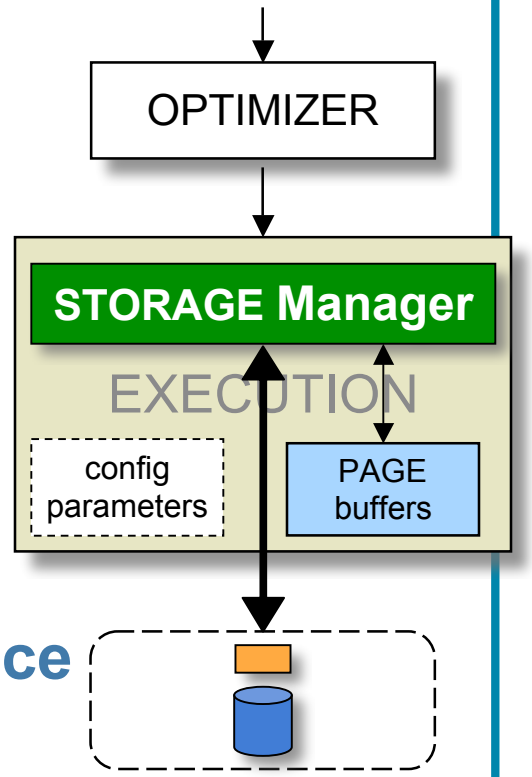
1. I/O costs preserved across abstraction layers
   - sequential access may not result in sequential I/O
     - concurrent execution of different queries

2. Performance will be tuned
   - error-prone manual configuration
     - data layout parameters

       `EXTENTSIZE,`

       `STRIPED_CONTAINERS,` …
     - prefetching parameters

       `PREFETCHSIZE,` …

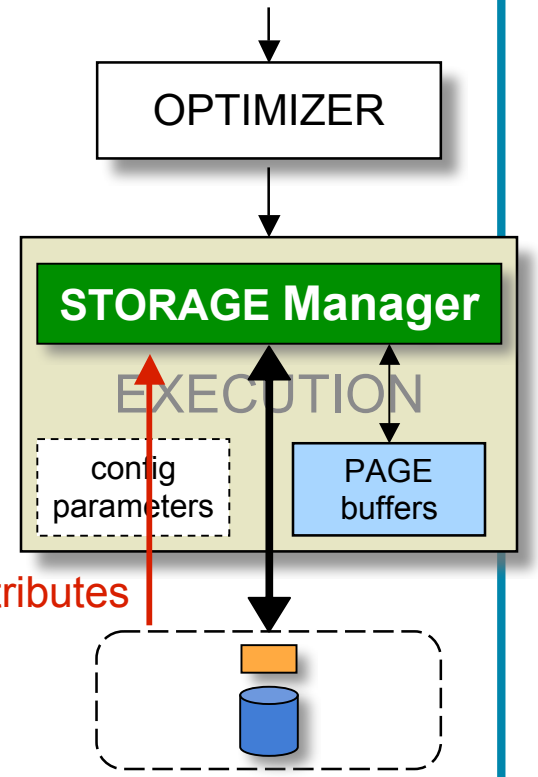


No info about storage device performance

Restoring Assumptions with Lachesis

1. I/O costs preserved across abstraction layers
   - nearly sequential efficiency even with competing I/O

2. Performance is correctly tuned
   - explicit performance attributes allow automatic adjustment
   - simplified management
     - no need for manual tuning



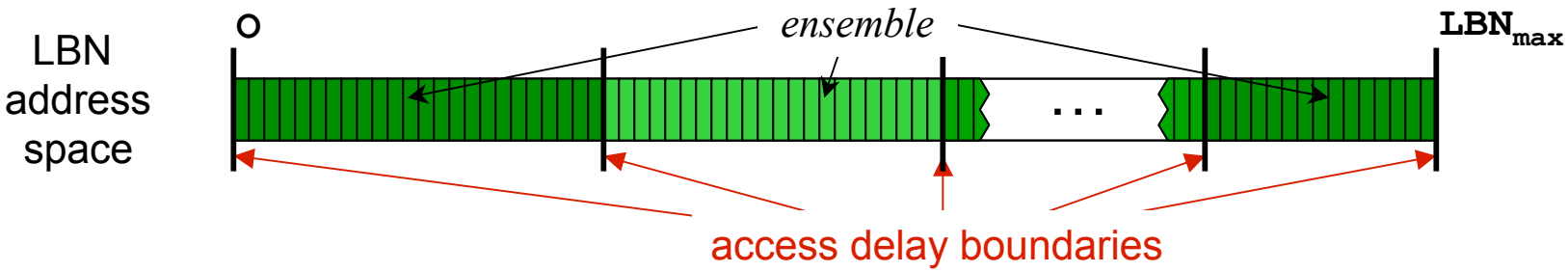
Storage device provides explicit hints

Lachesis: Robust Storage Management

- Storage provides a few hints to DBMS
  - device-neutral expression
  - restores current system abstractions
- Proper division of labor
  - hints from below let DBMS issue efficient I/Os
  - storage device ensures efficient execution
- Eliminates error-prone manual tuning
  - automatic adaptation to device-specifics
  - robust response to dynamic workload changes

Ensemble of Contiguous Blocks

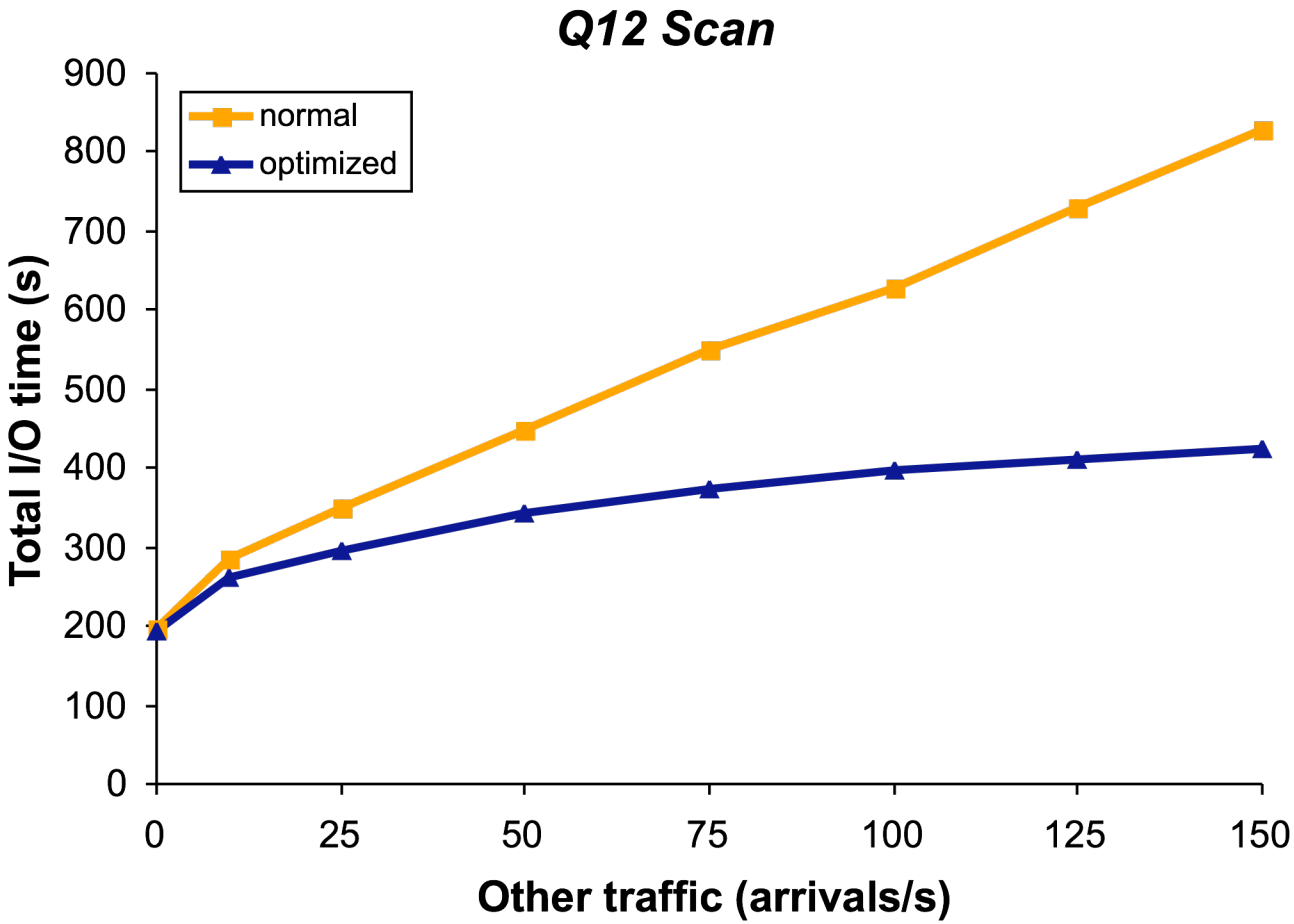
- Encapsulates delays in sequential accesses



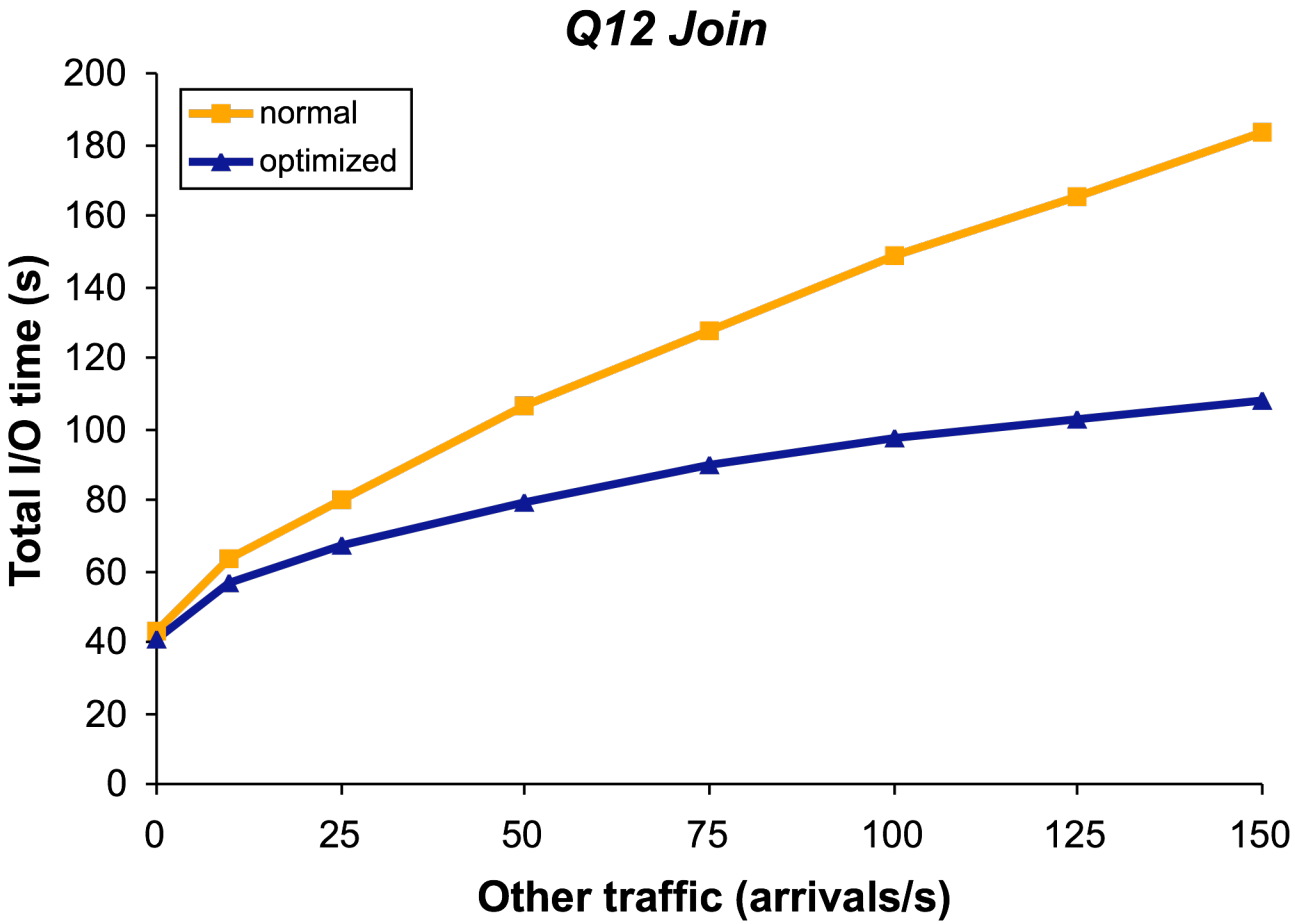
- Explicit contract between devices and DBMS
  - align I/Os to ensemble boundaries
  - match accesses to ensemble units
- Variable size across LBN address space

Device-independent expression of device-specifics

One Table Comparison

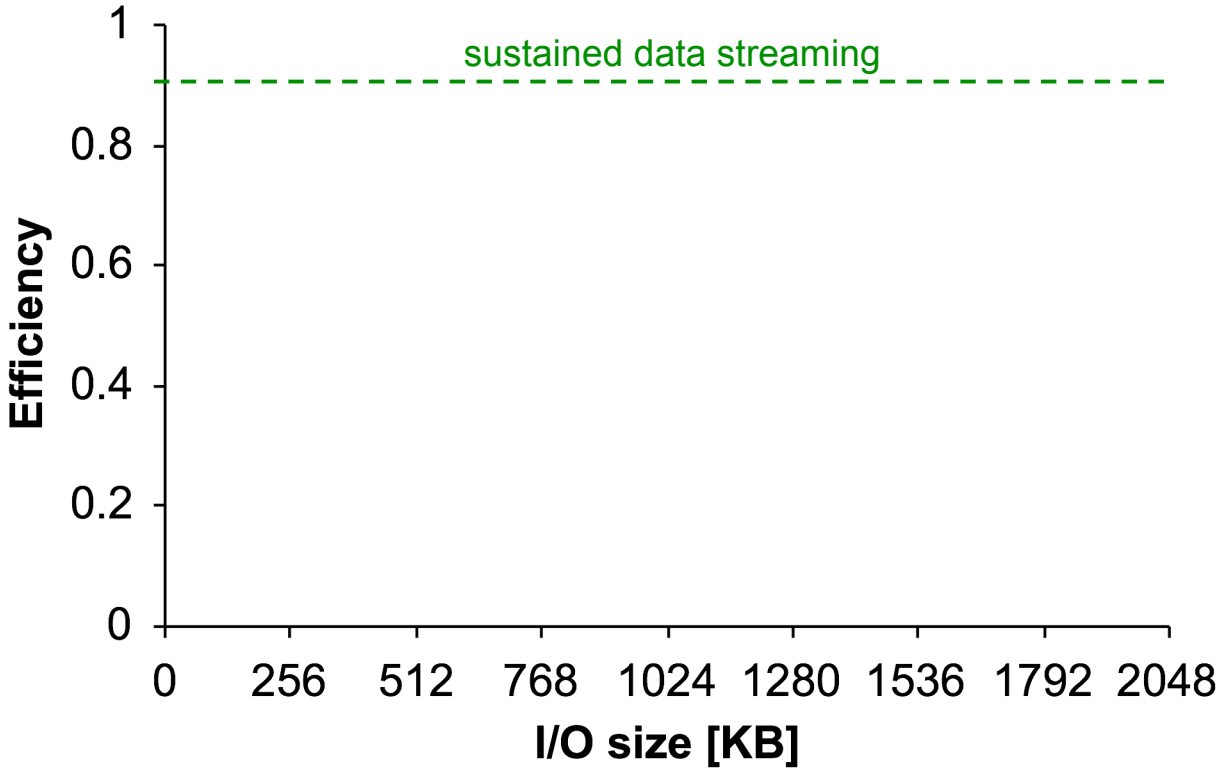


Two Tables Comparison



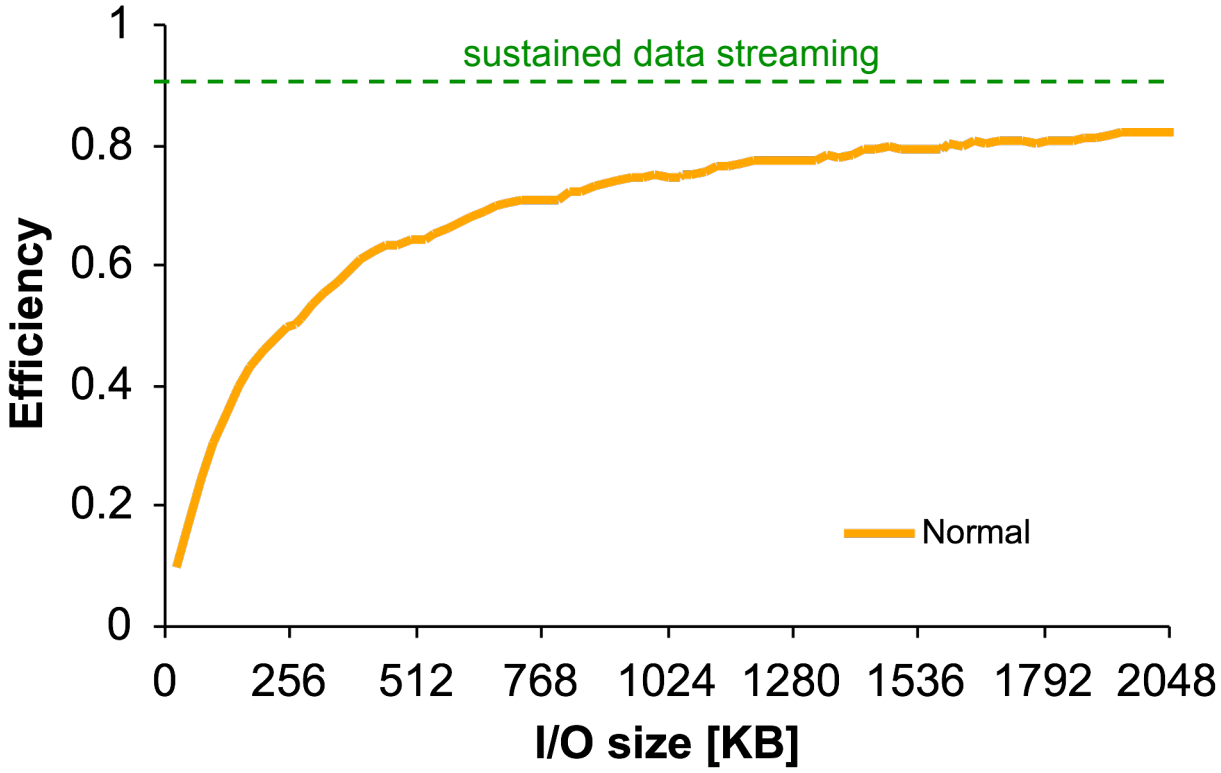
Ideal World Disk Efficiency

$$\text{disk efficiency} = \frac{\text{data transfer time}}{\text{total time}}$$



Measured Disk Efficiency: Real World

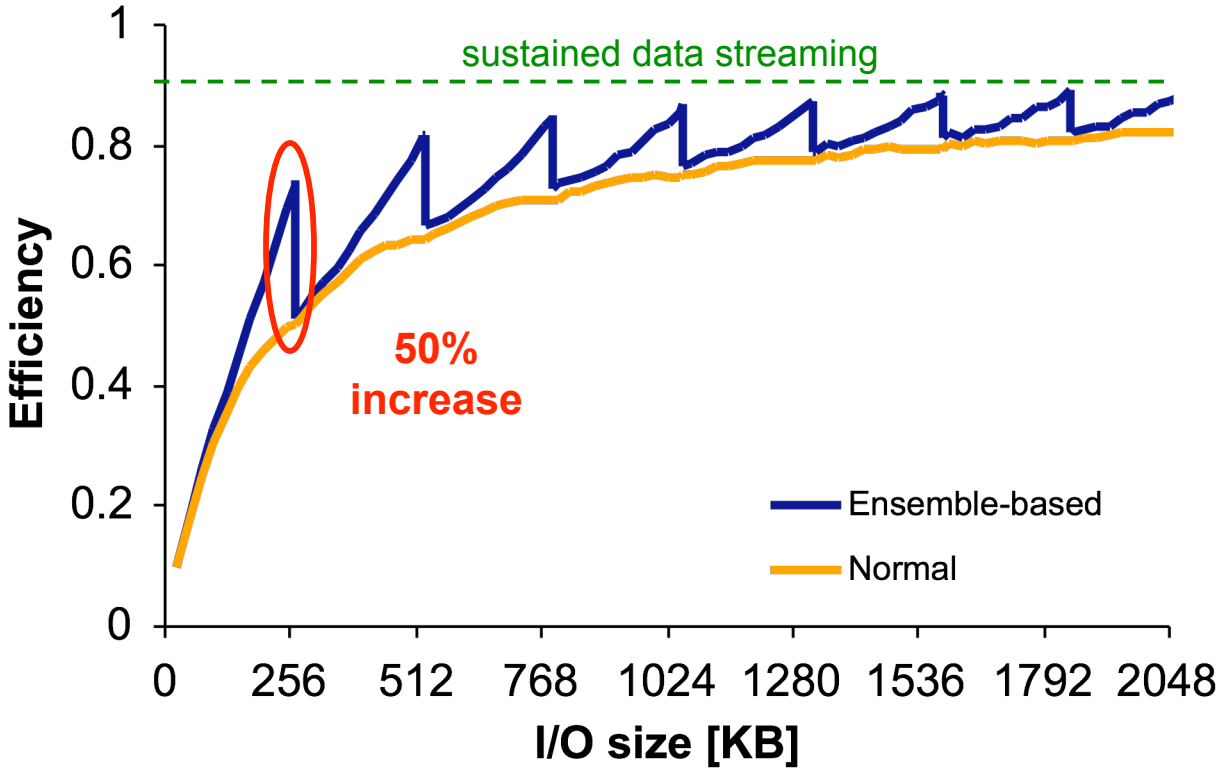
random reads to disk's first zone (264 KB per track)



Inefficient access w/o explicit information

Measured Disk Efficiency: Ensembles

random reads to disk's first zone (264 KB per track)
access aligned on ensemble (track) boundary



Ensemble-based access most efficient when I/O size matches track size

Workload Description & Scenarios

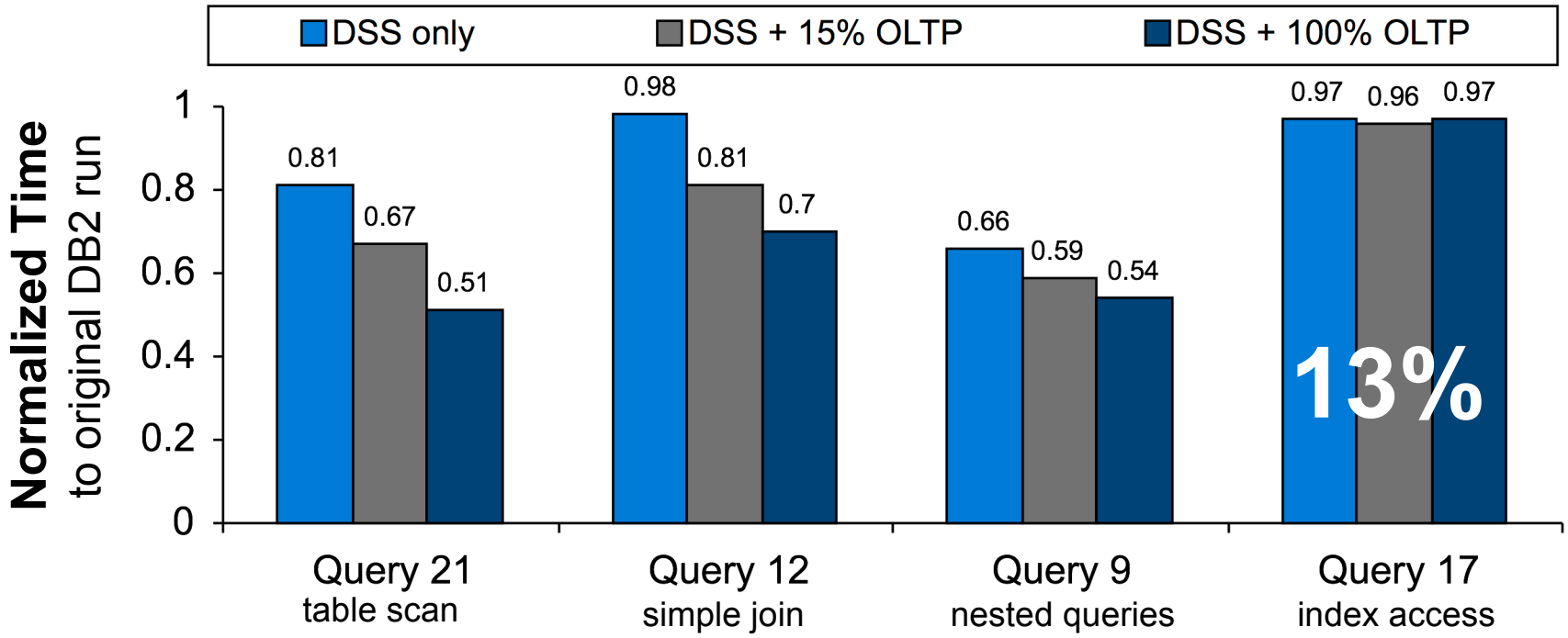
- Compound workloads
  - decision support system queries (DSS)
  - on-line transaction processing (OLTP)
- Three scenarios for DBMS setup
  - dedicated DSS systems
    - doubles the cost of hardware
  - mostly DSS with occasional random traffic
    - 15% I/Os come from OLTP
  - OLTP system with DSS running concurrently
    - live transaction system used for data mining

IBM DB2 Trace Replay

- I/O traces from running TPC-H (DSS)
  - run single query to completion
  - capture block-level traces
- Replay
  - modify traces to simulate ensemble access
    - preserves data access order
  - introduce competing traffic by small random I/O
- TPC-H DB2 configuration
  - 10 GB TPC-H
  - performance manually tuned to given hardware

DB2 Trace Replay simulating Lachesis

- 22 TPC-H benchmark queries
- 10GB dataset on one disk (data&index), another disk (logs)
- 2GHz uniprocessor Pentium 4, 1GB of memory



Improvement grows with increasing amount of traffic

Lachesis Prototype based on Shore

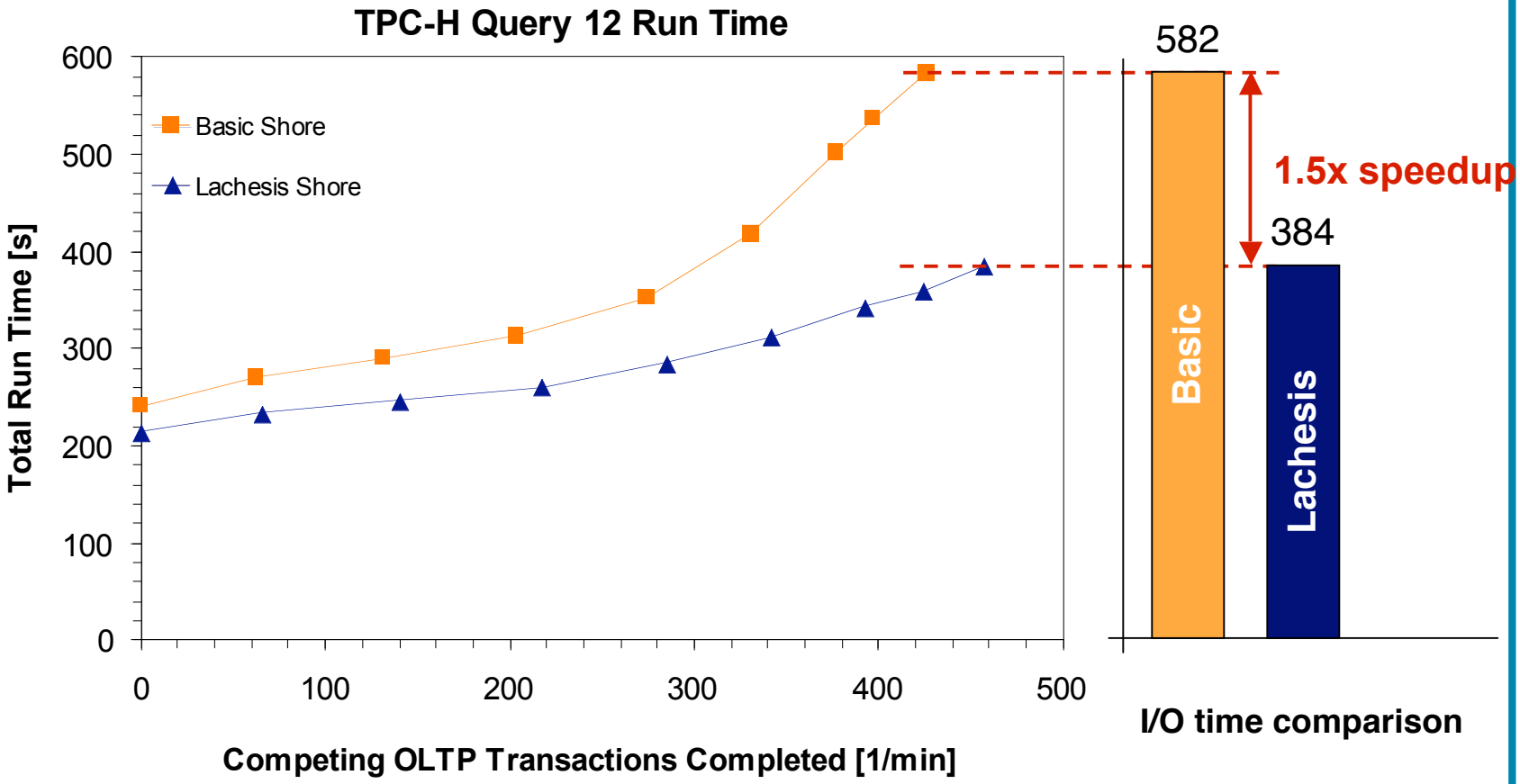
- Few modifications are necessary
  - adjust data allocation
    - match extent sizes to ensembles
  - match data access
    - read and write in ensemble units

- Straightforward integration
  - meshes with existing algorithms and structures
  - changes to 400 lines of code out of 250,000

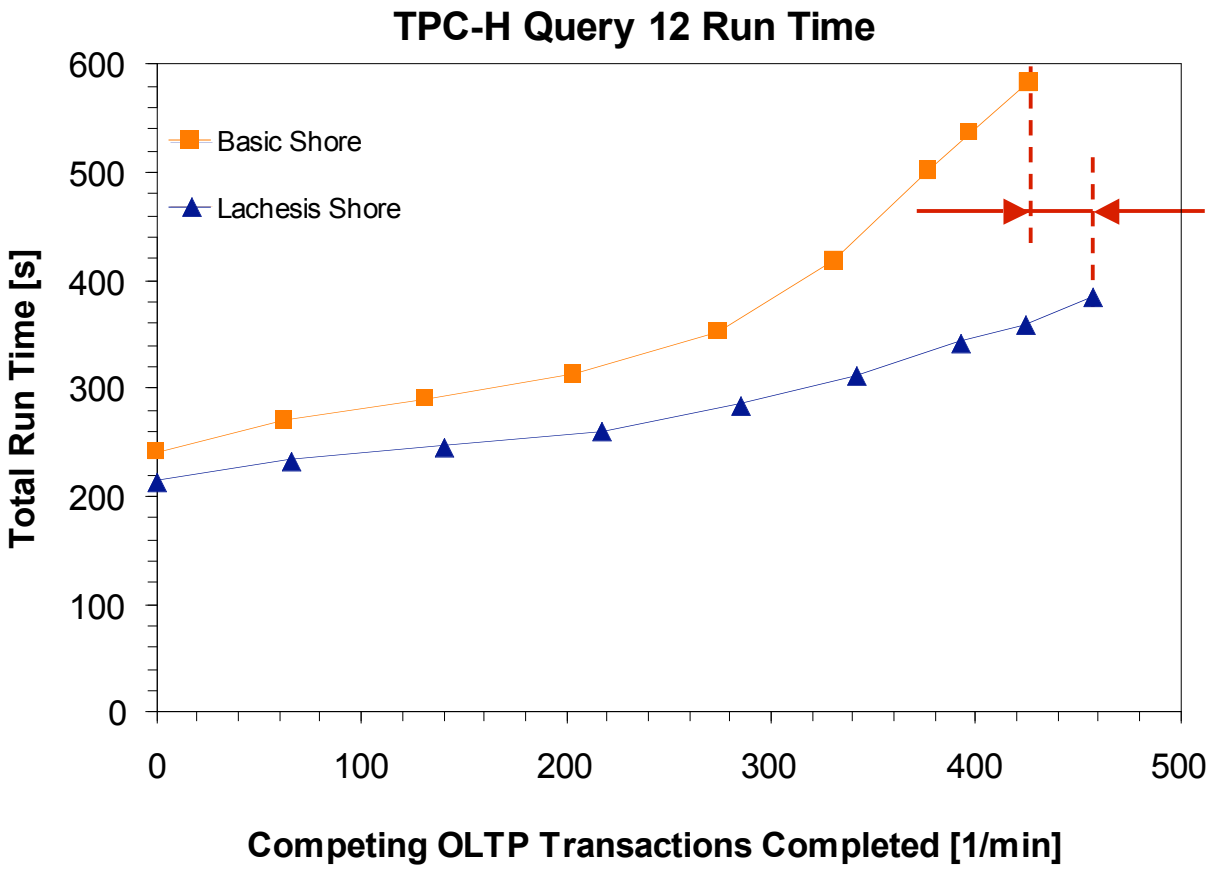
Prototype Implementation Setup

- Lachesis prototype environment
  - 1GB dataset on a single disk (data&index)
  - 2GHz uniprocessor Pentium 4, 1GB memory
- Two workloads running concurrently
  - TPC-H benchmark queries (DSS)
    - run single query to completion
  - TPC-C benchmark transaction mix (OLTP)
    - one transaction at a time
    - vary delay between transactions

Improving DSS Performance

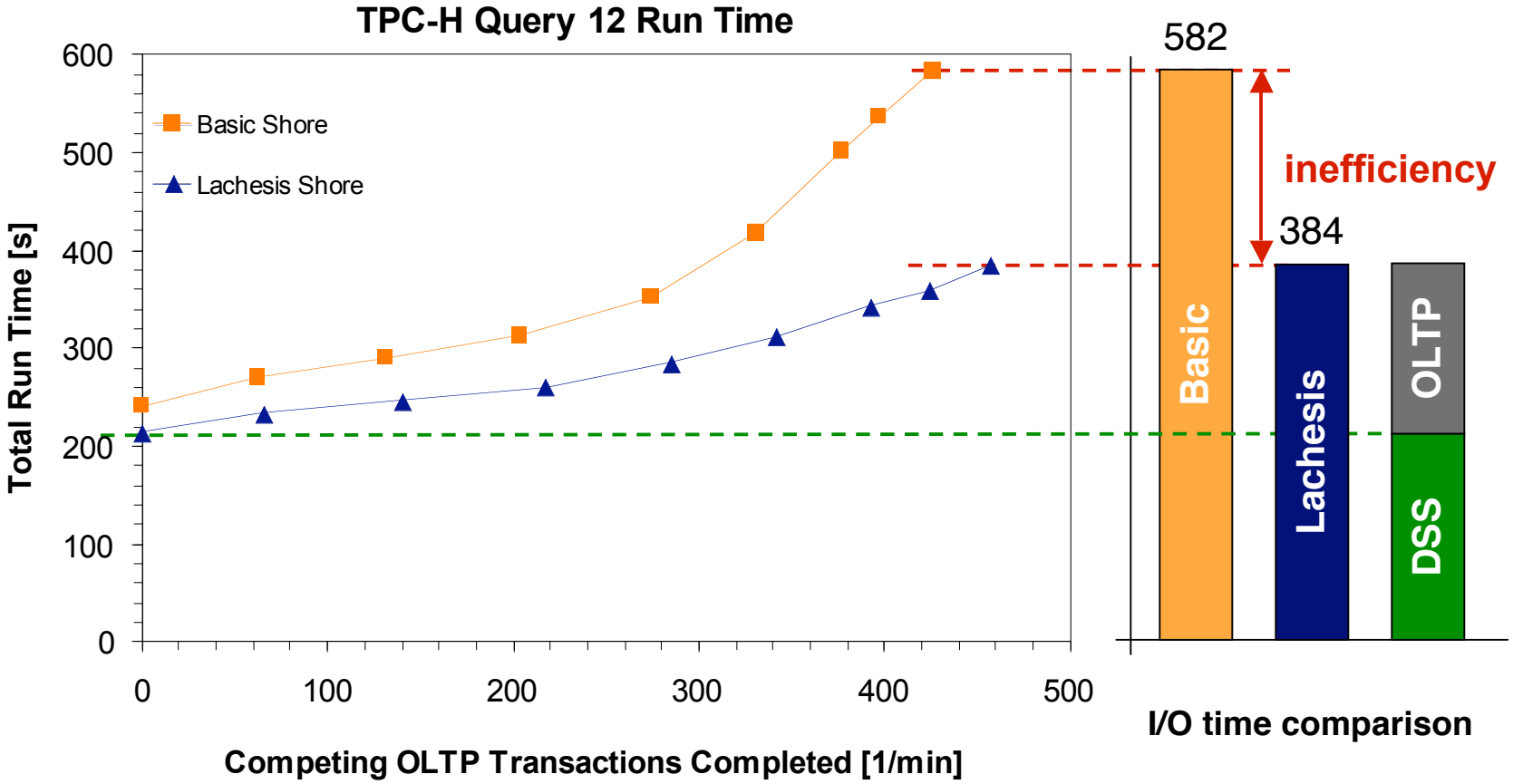


Improving OLTP Performance



7% higher OLTP throughput

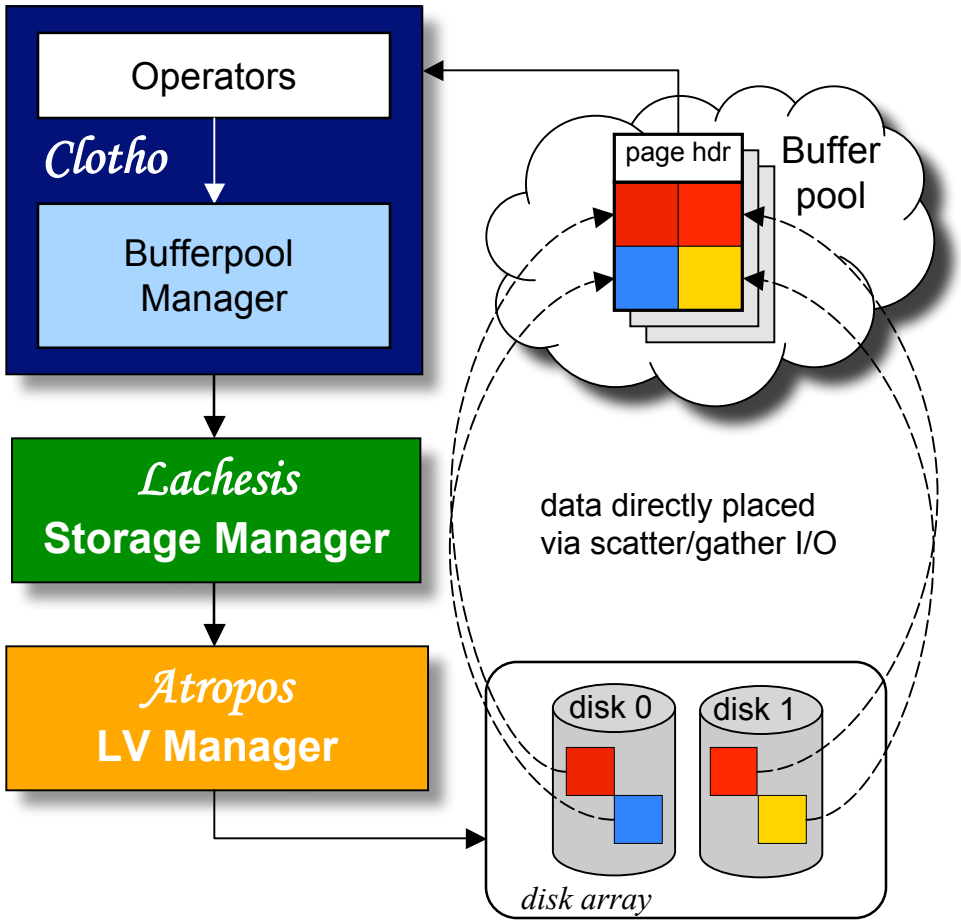
Compound Workload Execution



Same efficiency as when workloads executed separately

The Fates Database Storage Project

Efficient execution at all levels of memory hierarchy



Clotho

new in–memory page layout

- query-specific organization

Lachesis

construction of efficient I/Os

- explicit hints

Atropos

new data organization

- exploits unique disk features

Summary

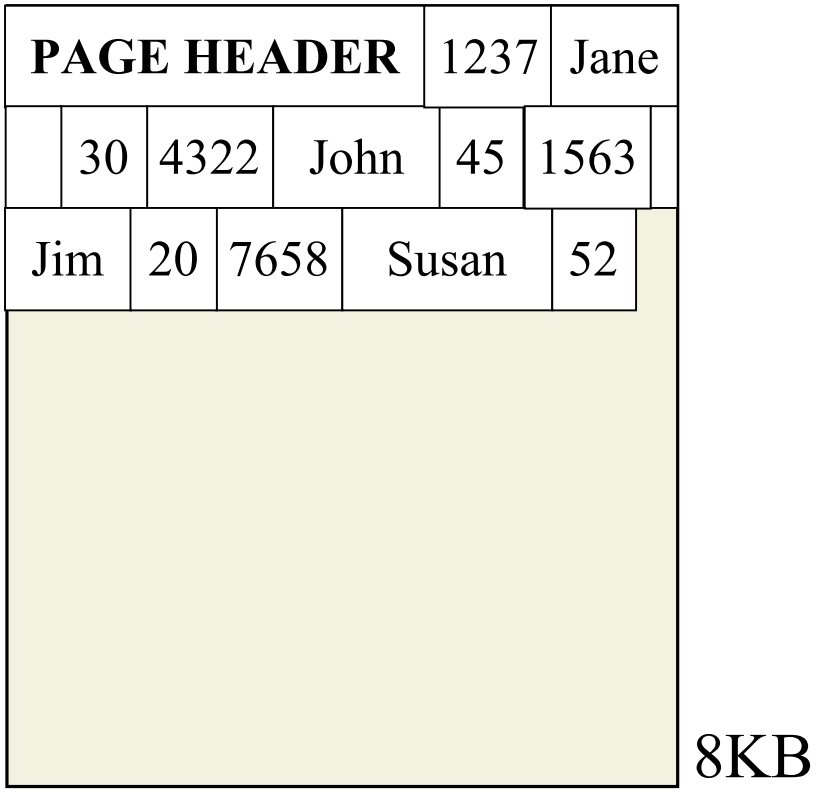
- Device-independent performance hints
  - encapsulate device-specific characteristics
  - restore assumptions about I/O costs
- Proper division of responsibilities
  - DBMS does not have to control everything
  - storage systems execute I/Os efficiently
- Simplified storage management
  - automatic performance tuning
- Robust performance of compound workloads

NSM (N-ary Storage Model)

R

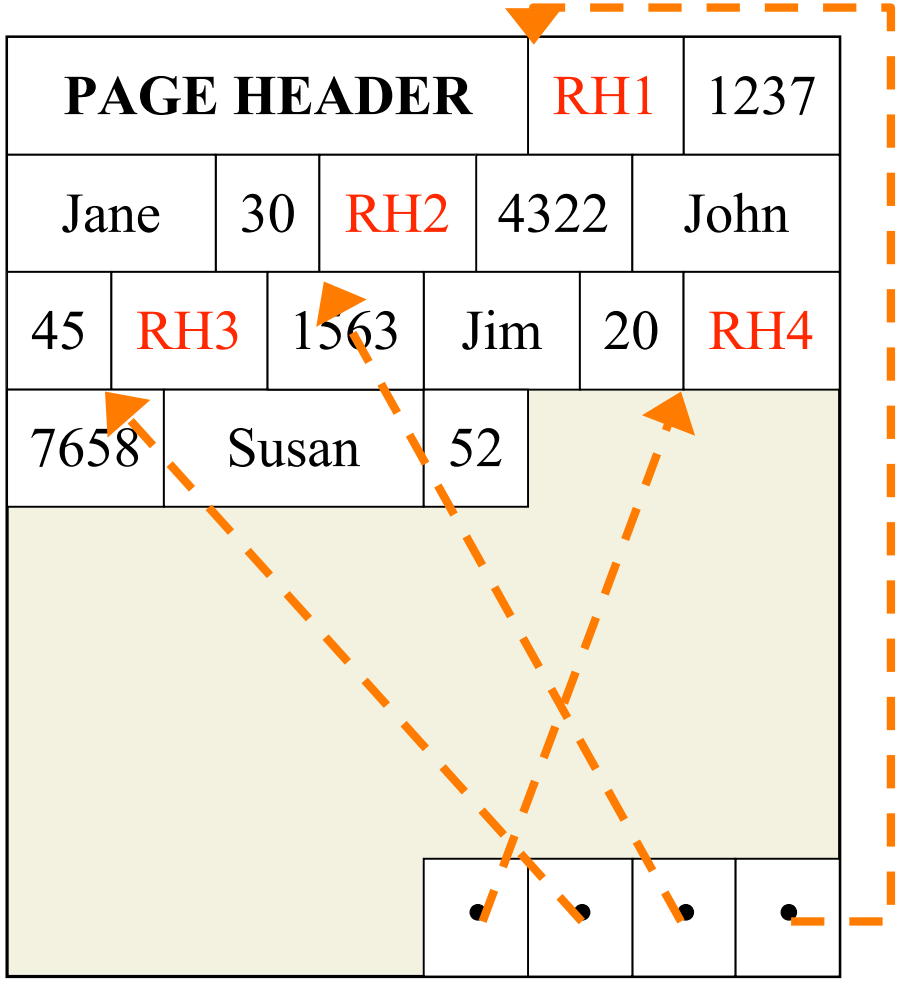
SSN	Name	Age
1237	Jane	30
4322	John	45
1563	Jim	20
7658	Susan	52
2534	Leon	43
8791	Dan	37

A page in Database



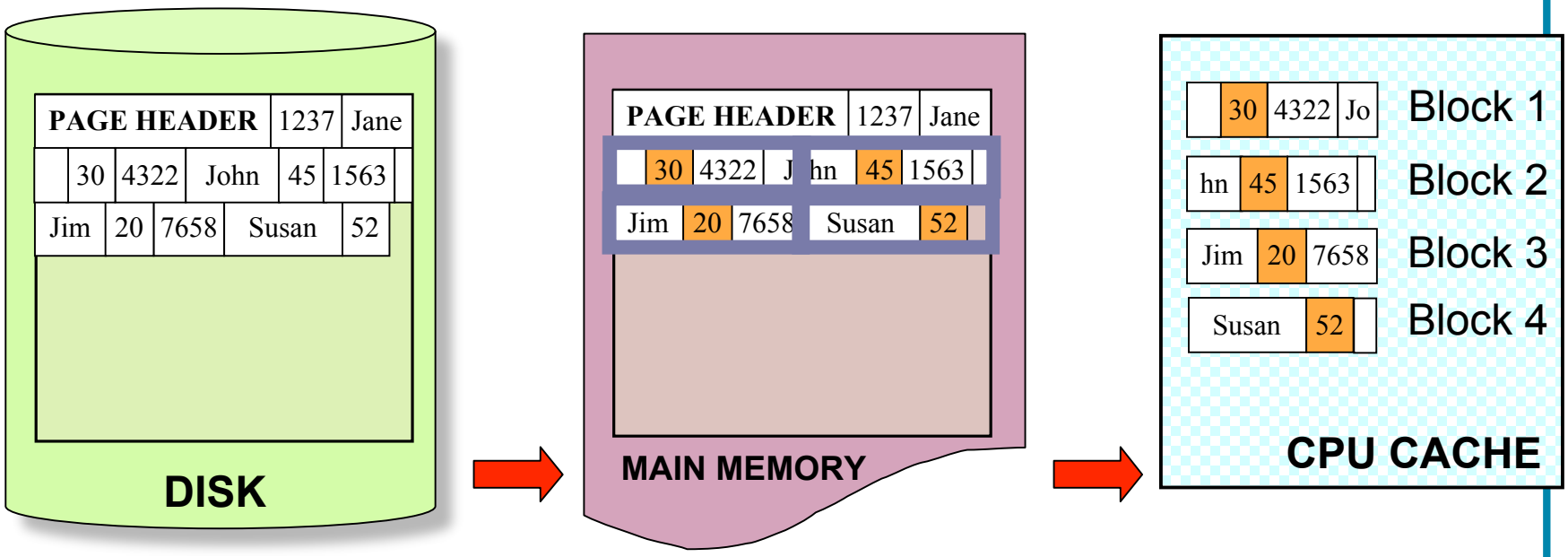
- Records are stored sequentially
- Attributes of a record are stored together

NSM Details


PAGE HEADER
RH1
1237
Jane
30
RH2
4322
John
45
RH3
1563
Jim
20
RH4
7658
Susan
52

NSM Behavior

Query asks for attribute “age” (column-major access)



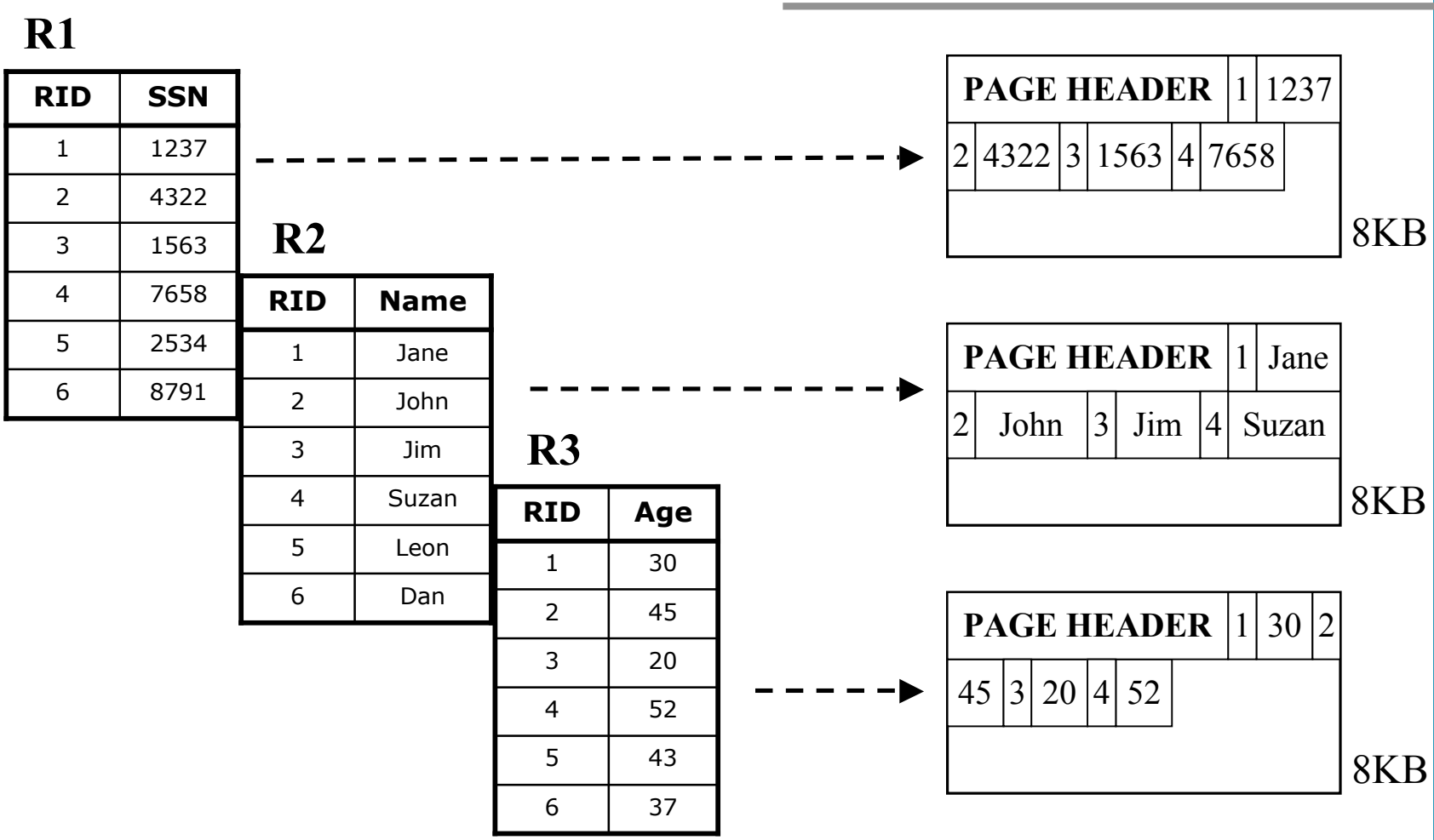
- Optimized for row-major access
- Bad at column-major access
  - Waste I/O bandwidth (fixed page layout)
  - Low spatial locality at CPU cache

DSM (Decomposition Storage Model)

SSN	Name	Age
1237	Jane	30
4322	John	45
1563	Jim	20
7658	Susan	52
2534	Leon	43
8791	Dan	37

Partition original table into *n* 1-attribute sub-tables

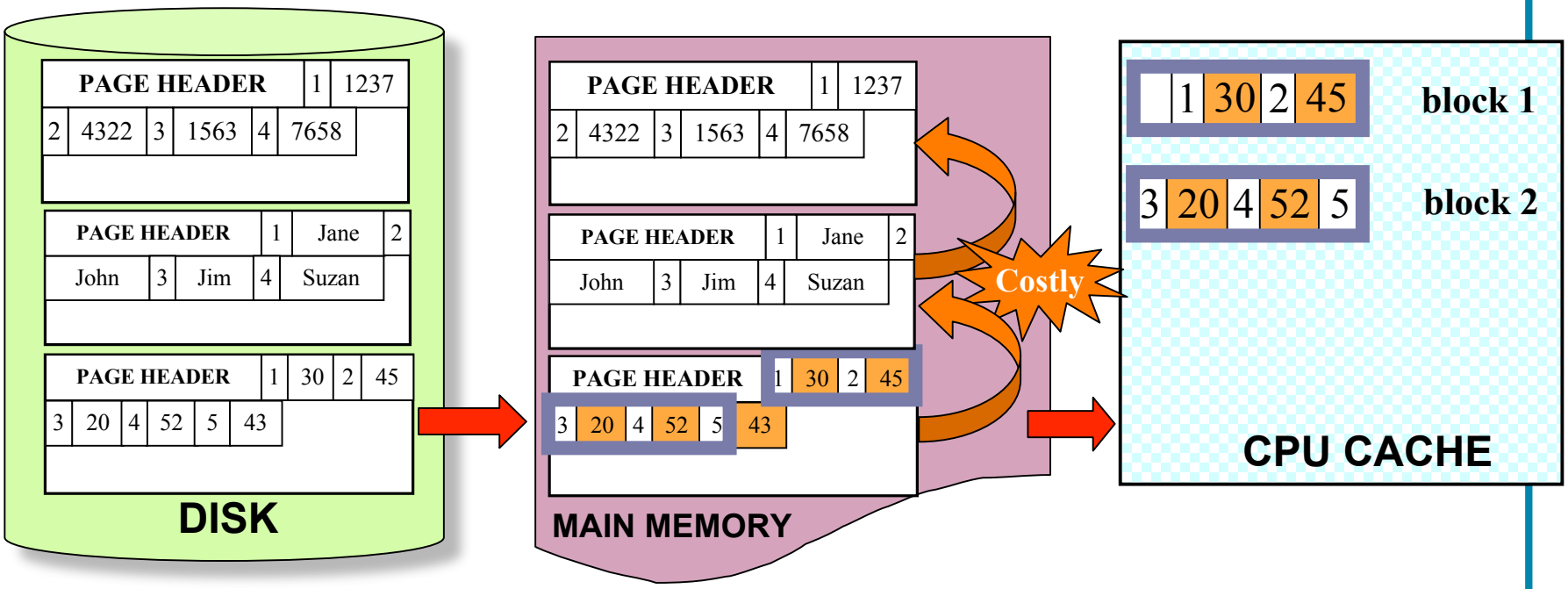
DSM (Decomposition Storage Model)



Each sub-table stored separately in normal NSM pages

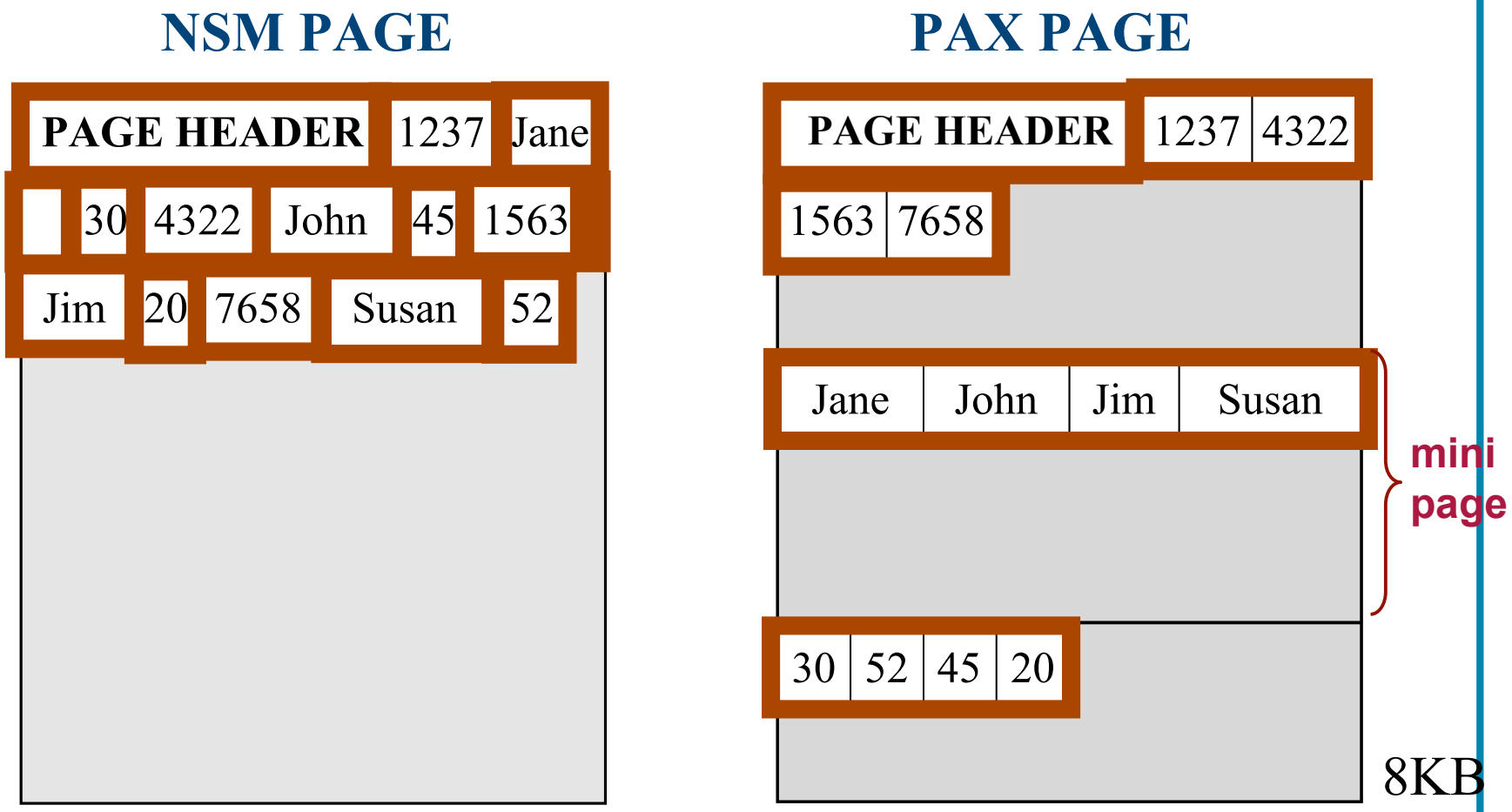
DSM Behavior

- Query asks for attribute “age” (column-major access)
  Query asks for all attributes (row-major access)



- Optimized for column-major access
- Bad at row-major access
  - Reconstructing full record induce random access

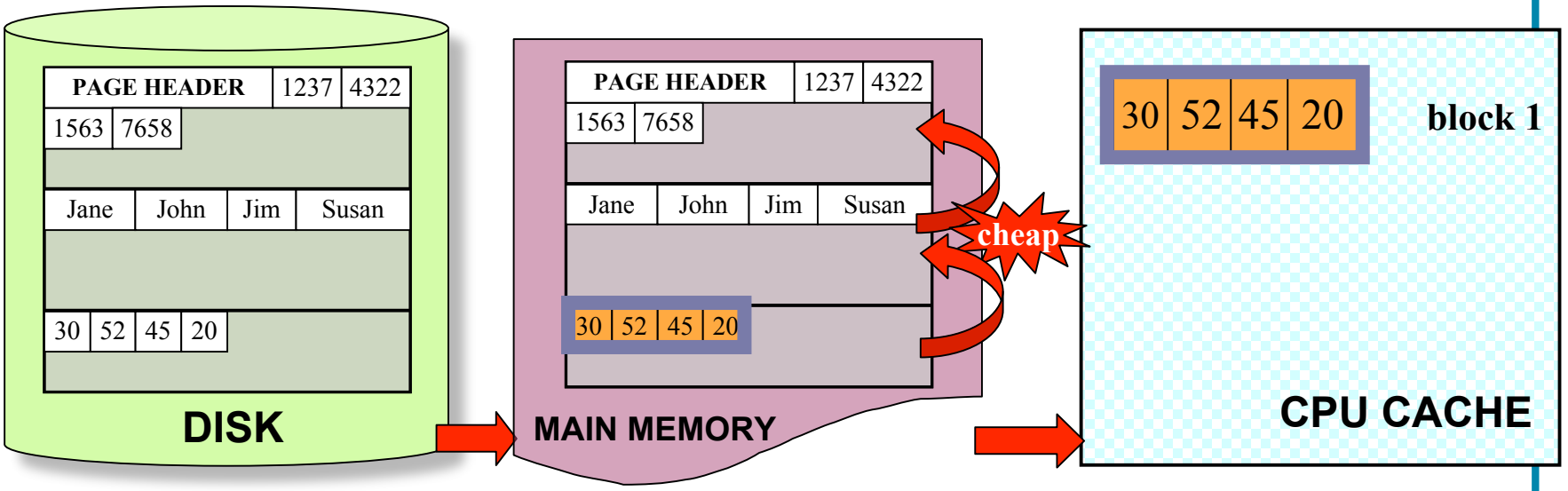
PAX (Partition Attributes Across)



Partition data *within* a page

PAX Behavior

- Query asks for attribute “age” (column-major access)
  Query asks for all attributes (row-major access)



- Optimized at CPU cache-memory level
- Suboptimal I/O performance for column-major access
  - Irrelevant data wastes I/O bandwidth (fixed page layout)

What's next…

- Lecture: 9/26
  - Database structures
  - DB Workloads