# Database storage management with object-based storage devices

Sami Iren
Seagate Research

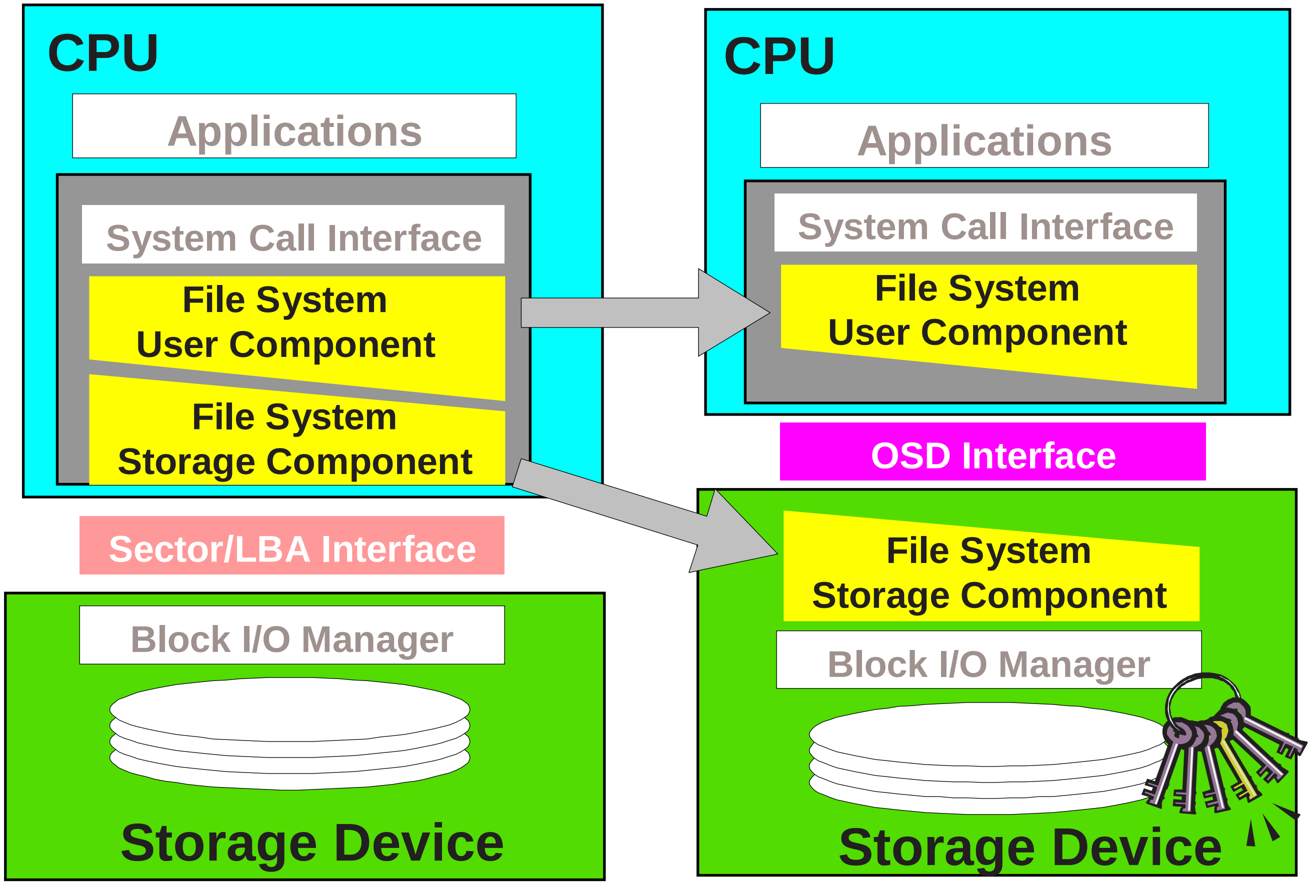
Steve Schlosser
Intel Research Pittsburgh

# Outline

- New hardware: Object-based Storage Devices (OSD)
- Vision: Database-aware storage systems
- Example: Transparent device-specific data placement
- Moving forward: Issues with the current OSD specification

# New hardware: Object-based Storage Devices (OSD)

- Same hardware, new interface …
- Remember Network Attached Secure Disks?



CPU

Applications

System Call Interface

File System User Component

File System Storage Component

Sector/LBA Interface

Block I/O Manager

Storage Device

CPU

Applications

System Call Interface

File System User Component

OSD Interface

File System Storage Component

Block I/O Manager

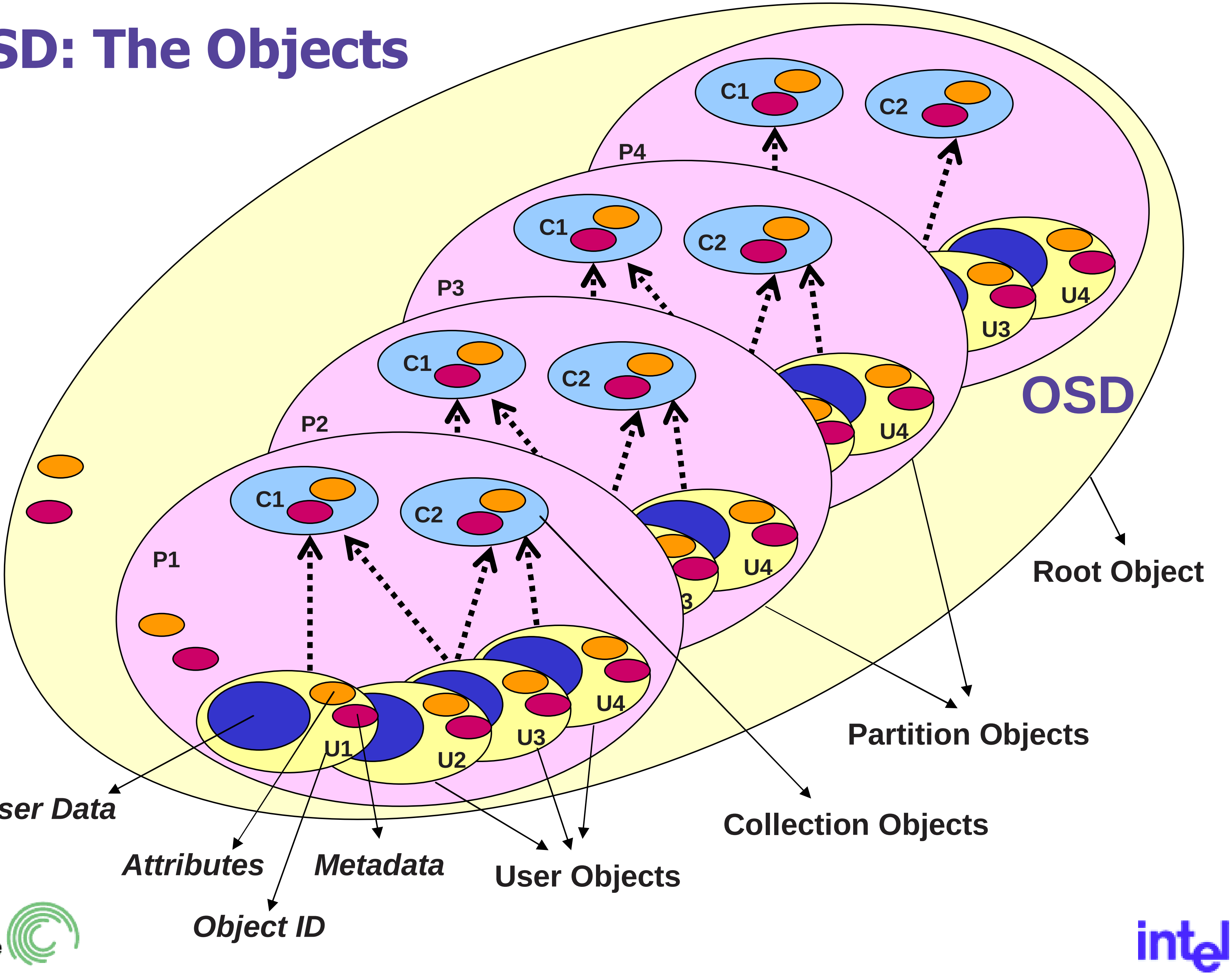
Storage Device

- Disk array/server subsystem
- I.e. LLNL units with Lustre

- “Smart” disk for objects
- I.e. Panasas storage blade

- Highly integrated, single disk
- I.e. prototype Seagate OSD

# OSD: The Objects



# OSD: The Command Set

- **Basic Protocol**
  - READ
  - WRITE
  - CREATE
  - REMOVE
  - GET ATTR
  - SET ATTR

**Very Basic** (READ, WRITE)

**Space Mgmt** (CREATE, REMOVE)

**Attributes** (GET ATTR, SET ATTR)
- opaque
- internal
- shared

- **Specialized**
  - APPEND – write w/o offset
  - CREATE & WRITE – save msg
  - FLUSH OBJ – force to media
  - LIST – recovery of objects

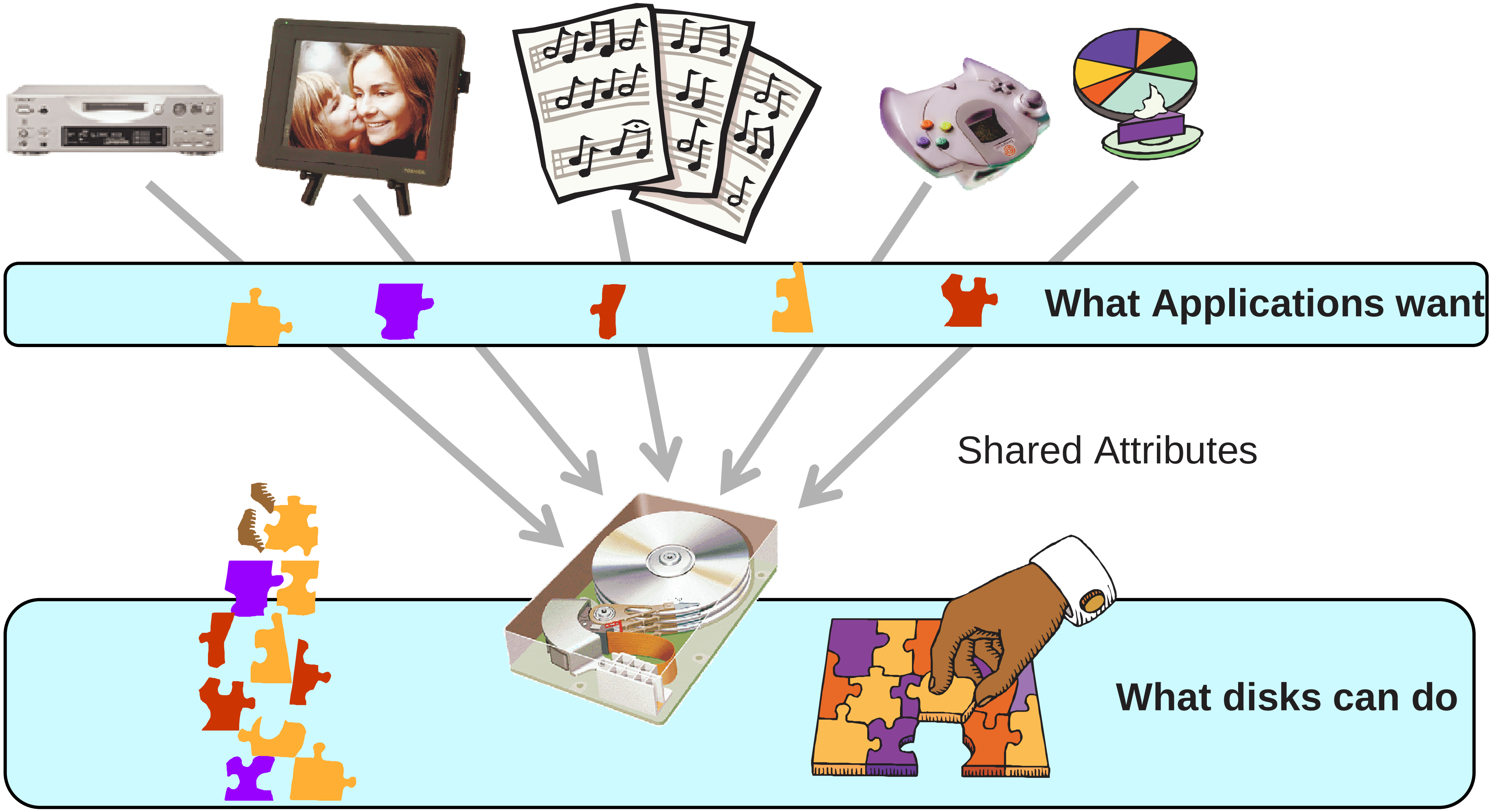
- **Security**
  - Authorization – on each request
  - Integrity – for args & data
  - SET KEY
  - SET MASTER KEY

- **Groups**
  - CREATE COLLECTION
  - REMOVE COLLECTION
  - LIST COLLECTION

- **Management**
  - FORMAT OSD
  - CREATE PARTITION
  - REMOVE PARTITION

# OSD: Shared attributes

- Mechanism to push application information into storage



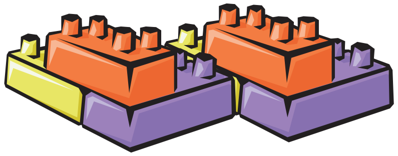
Traditional

OSD

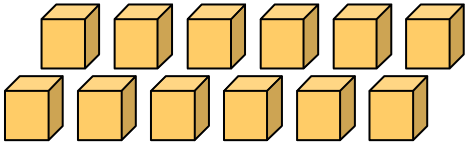
# OSD: Benefits

- Improved performance
  - Hints, QoS, Differentiated Services
  - Data can be differentiated at the device
- Improved device and data sharing
  - Platform-dependent metadata moved to device
  - Systems need only agree on naming
- Improved scalability & security
  - Devices directly handle client requests
  - Object security w/ application-level granularity
  - Finer granularity than LUN-based security
- Improved storage management
  - Self-managed, policy-driven storage
  - Storage devices become more autonomous

Volumes



Objects



Blocks

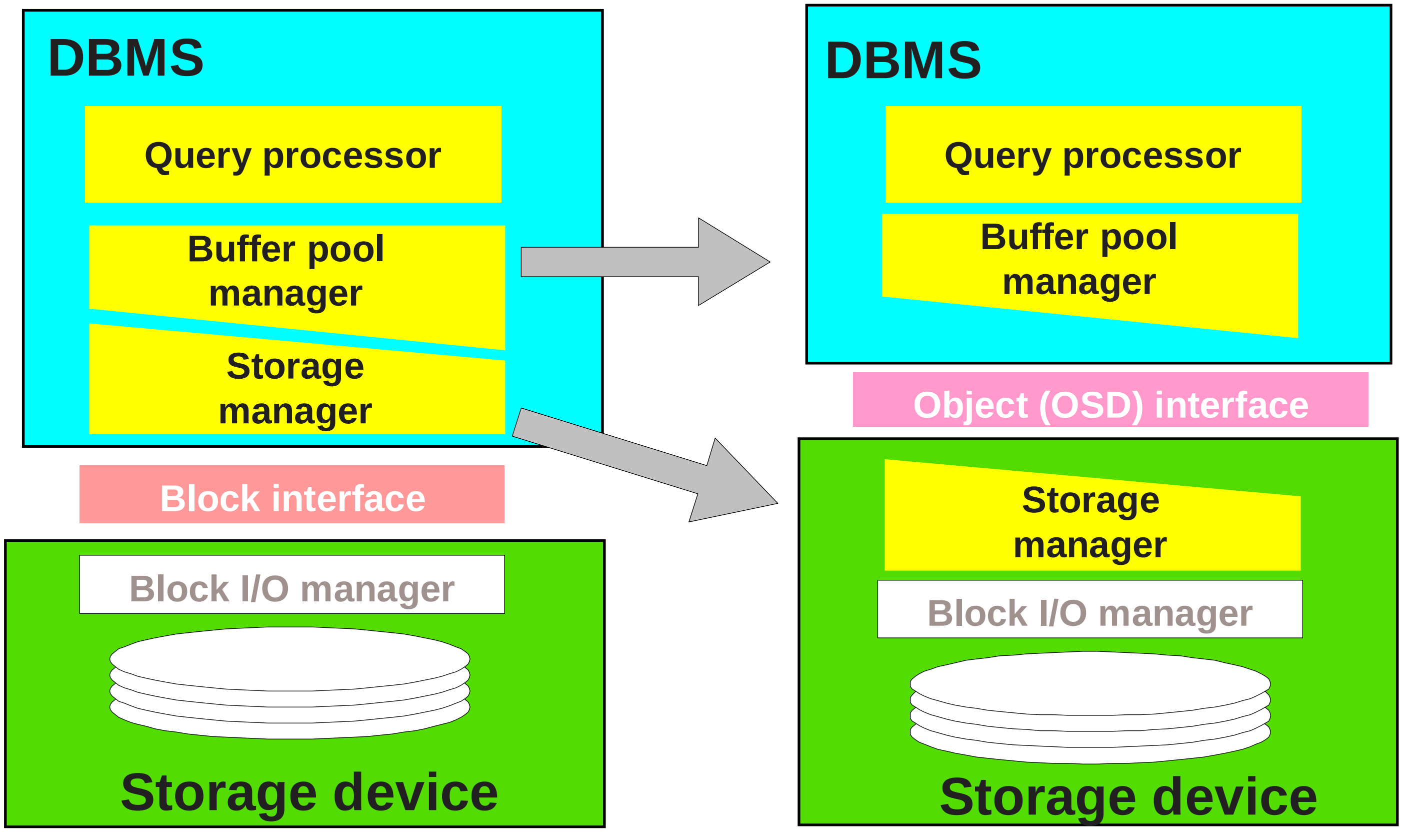
# Outline

- New hardware: Object-based Storage Devices (OSD)
- Vision: Database-aware storage systems
- Example: Transparent device-specific data placement
- Moving forward: Issues with the current OSD specification

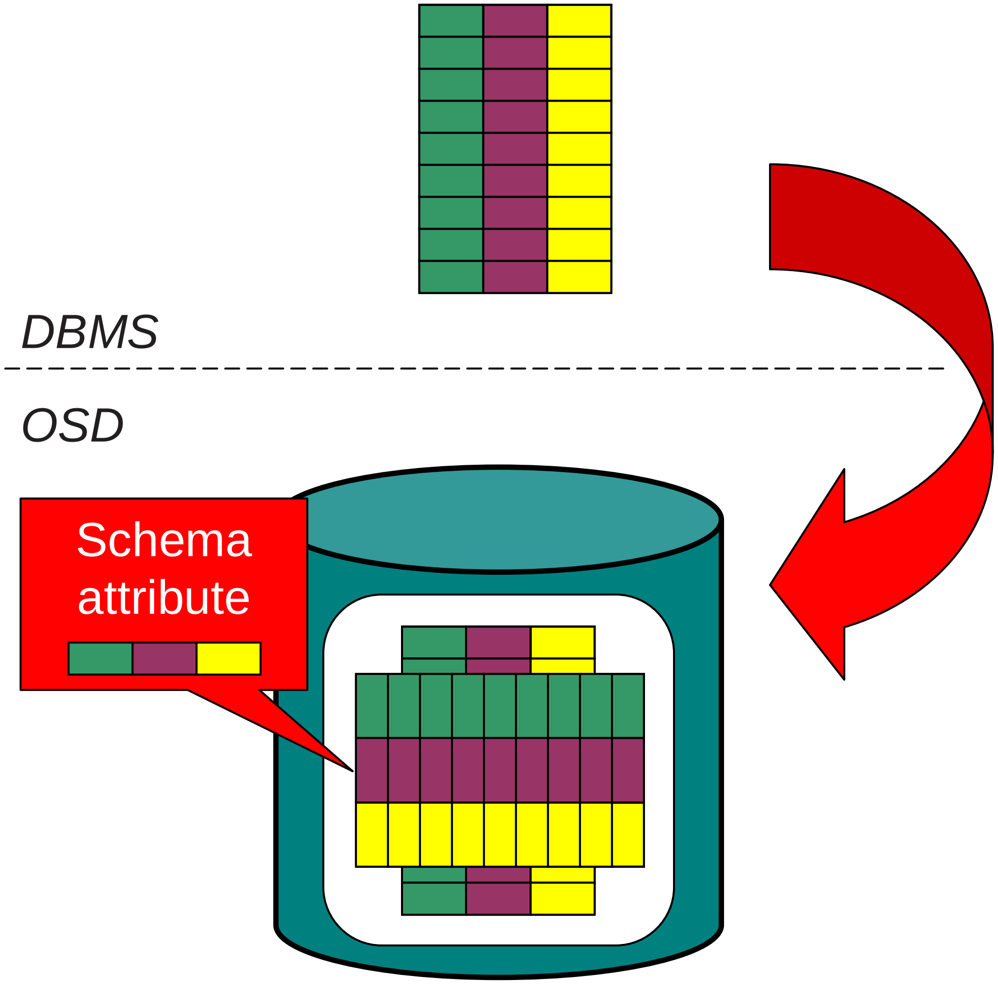
# Vision: Storage that cooperates with databases

- DBMS storage managers do a lot to optimize storage access, with insufficient information
- Storage subsystem front-ends do a lot to optimize under the covers, without enough information about the application
- Wouldn't it be better if we could just get along?
- OSD can provide this mechanism
- **We have a unique opportunity to make DB software and storage more cooperative**
  - If you remember nothing else from this talk…

# Solution: OSD for databases



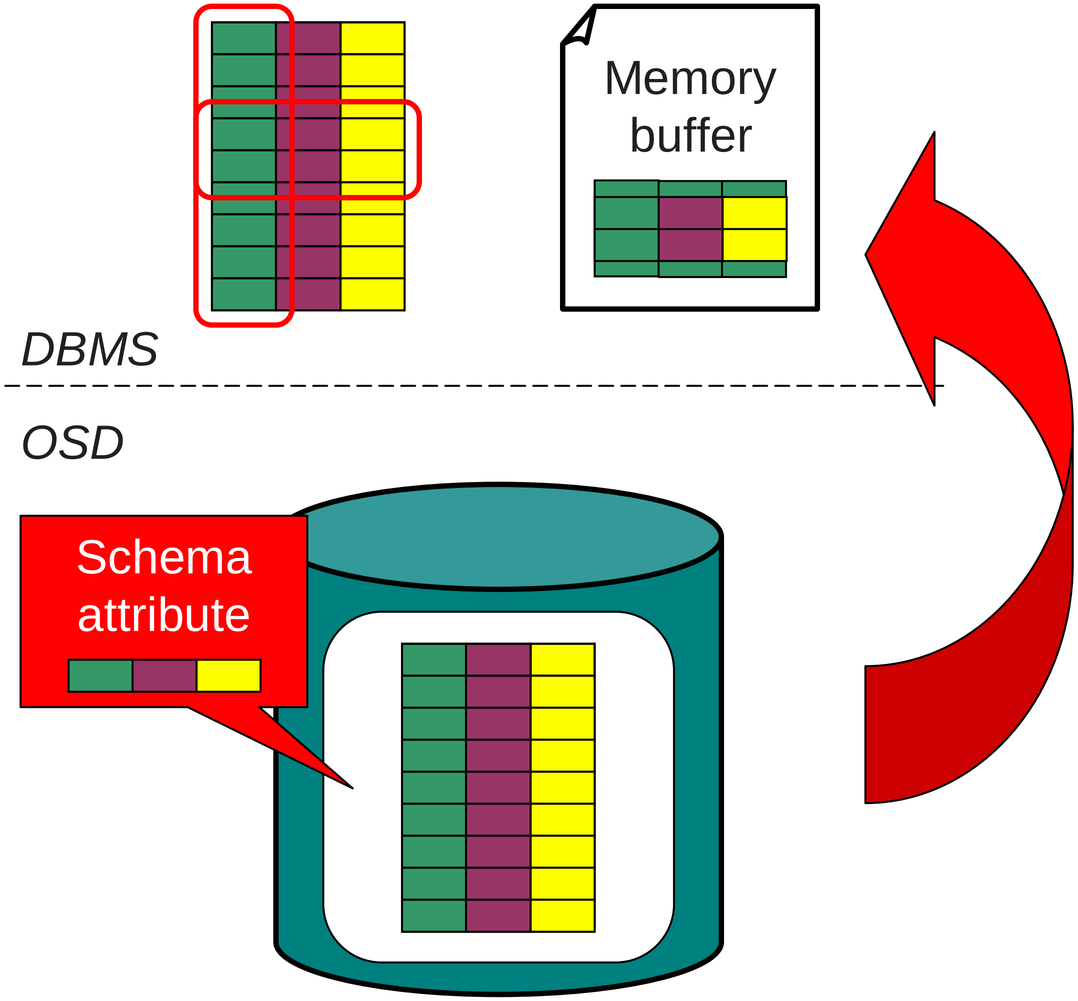
# Database-aware storage systems



1. Create object for relation
2. Attach schema attribute
3. Populate relation

- Goal 1: OSD uses its parameters to optimize layout under the covers
- Goal 2: Access method should be independent of storage optimizations

# Database-aware storage systems



- Goal 2: Access method should be independent of storage optimizations
- DBMS should just specify the data it wants, and the destination address
- DBMS shouldn't worry about
  - Data placement
  - Addressing blocks/bytes

# Outline

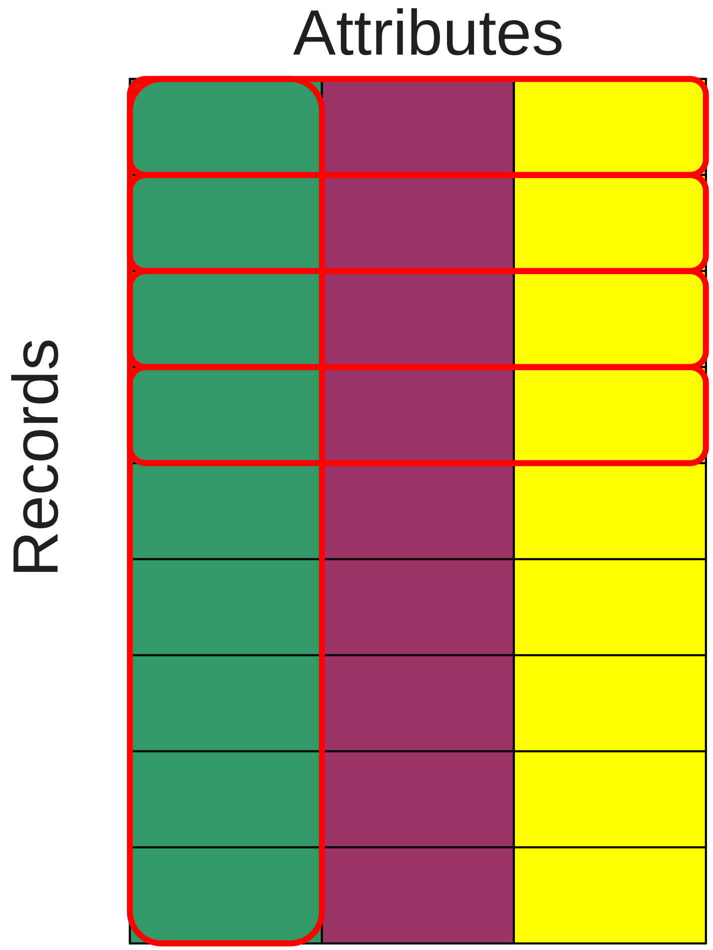
- New hardware: Object-based Storage Devices (OSD)
- Vision: Database-aware storage systems
- Example: Transparent device-specific data placement
- Moving forward: Issues with the current OSD specification

# The Fates project at CMU

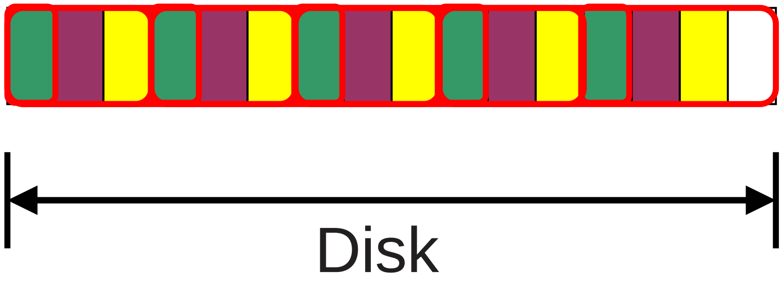
- Collaboration between Database Group and Parallel Data Lab
  - Ongoing involvement of researchers at Intel Research and EMC
  - Lachesis: Improved communication between storage subsystem and DB storage manager (VLDB 2003)
  - Clotho: Retool database software to fetch query-specific data (VLDB 2004)
  - Atropos: Leverage detailed disk information for improved 2D data placement (FAST 2004)

- Key idea: Storage informs database of its characteristics
  - Combination of schema and disk parameters yields 2D placement

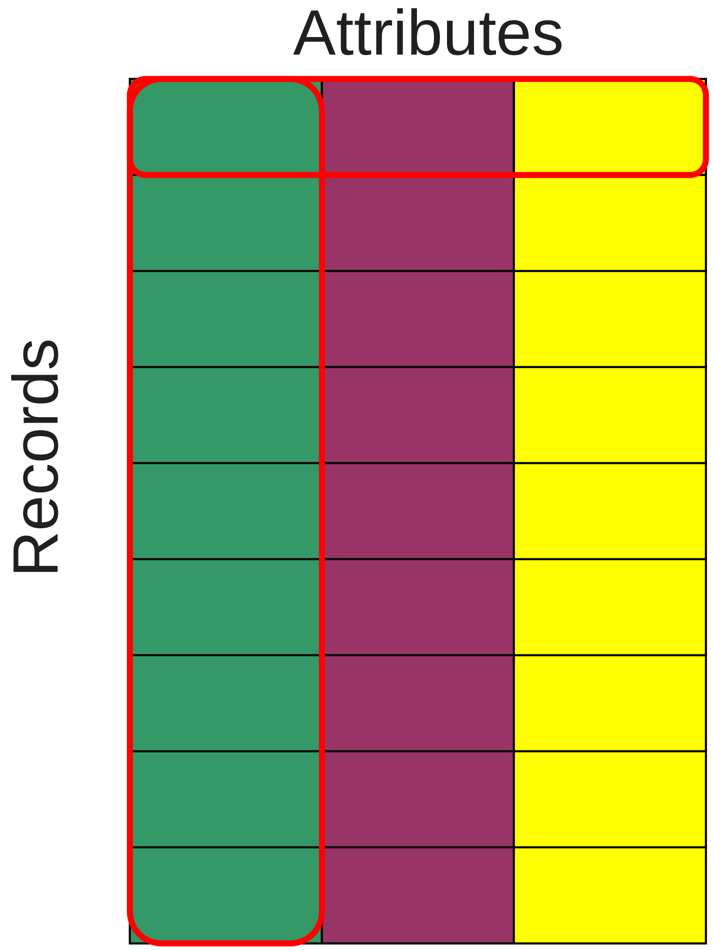
# 2D data structure access



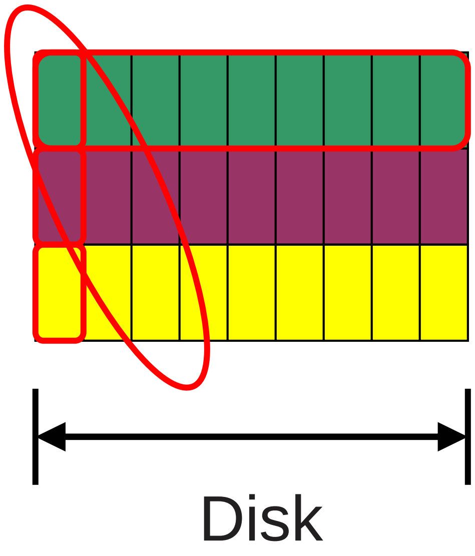
- On-disk storage requires serialization
- Access along one dimension is efficient
  - i.e., sequential
- Access along the other is inefficient
  - i.e., random I/O
  - Or, read entire relation and discard unwanted attributes



# Intelligent data placement in Fates



- One column per disk track
- Column-major access is efficient
- Row-major access is inefficient
    - One block per rotation
- Semi-sequential layout enables efficient row-major access

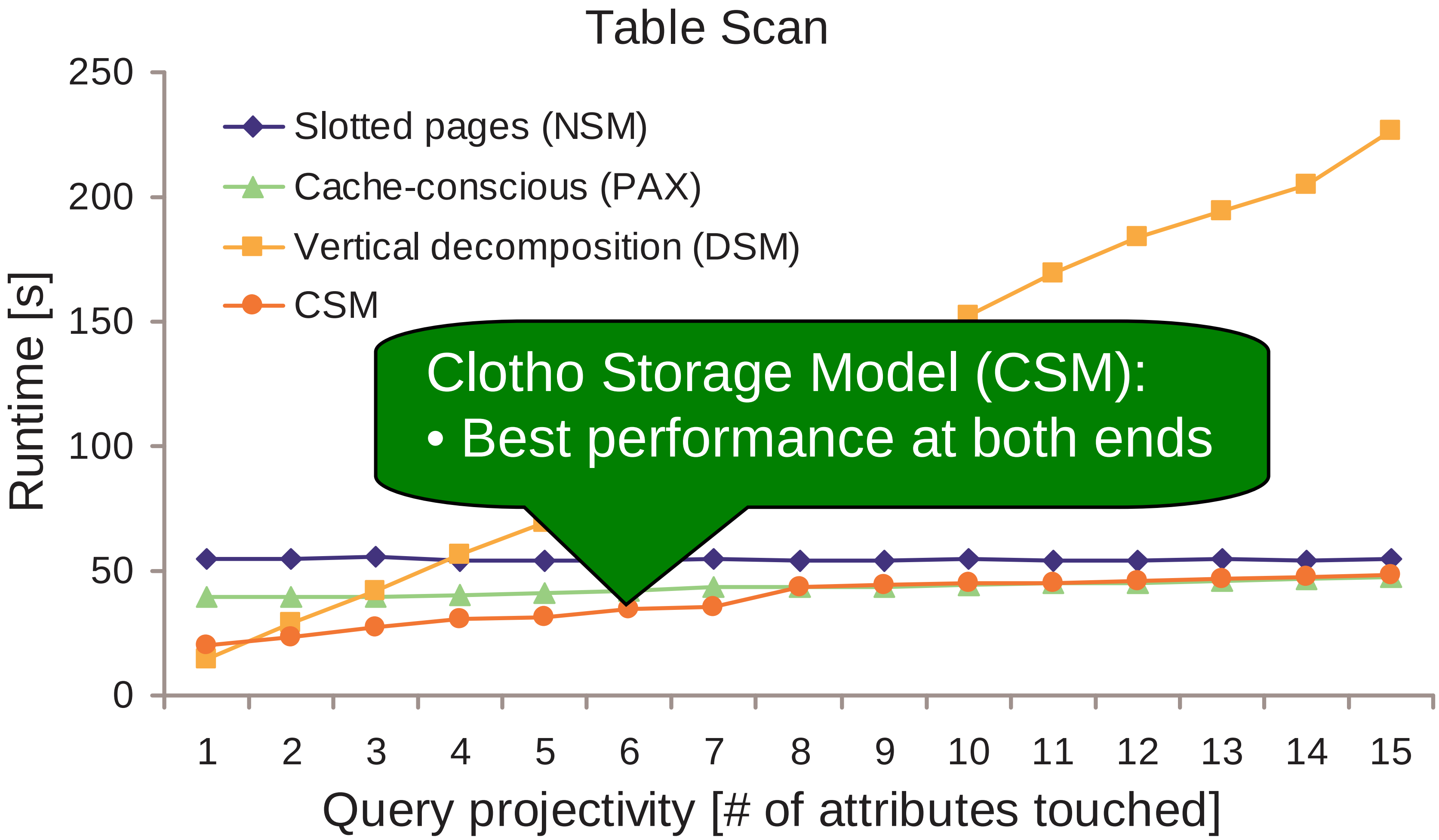


Courtesy of Minglong Shao, CMU

# High-level Fates result

Table: CREATE TABLE R (FLOAT a1, …, FLOAT a15) (1GB)

Query: SELECT a1, a2, …, FROM R WHERE a1 < Hi



Clotho Storage Model (CSM):

- Best performance at both ends

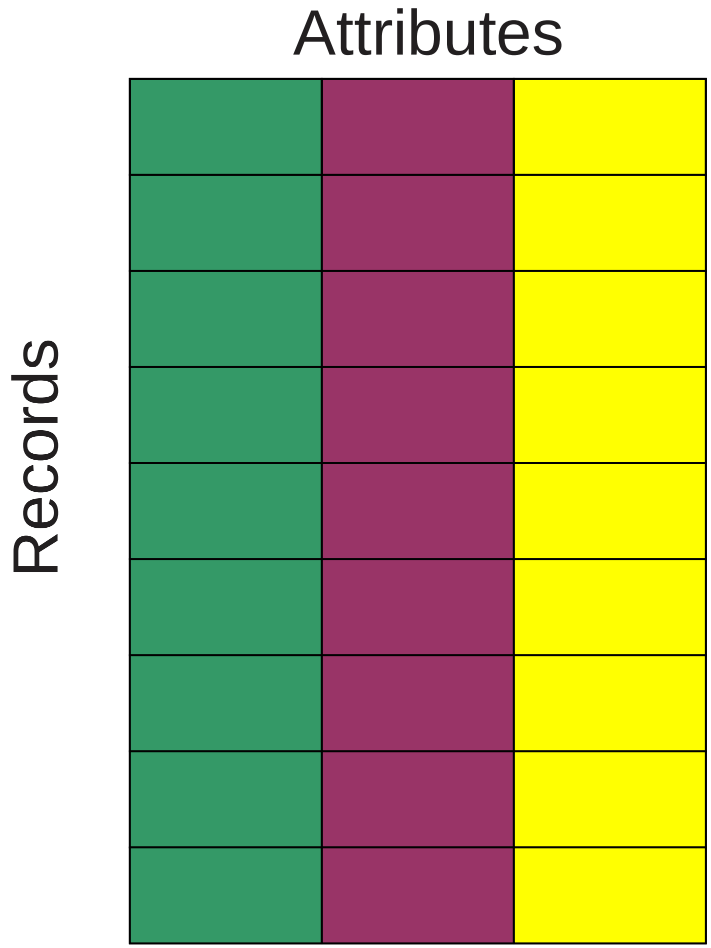
# Solving the "parameter problem"

- Relying on storage vendors to expose parameters is problematic
  - Measuring parameters is difficult and fragile (but not impossible)
- Key idea: OSD can solve this problem
  - Expose schema to storage subsystem via shared attributes
  - Storage subsystem can do data placement under the covers

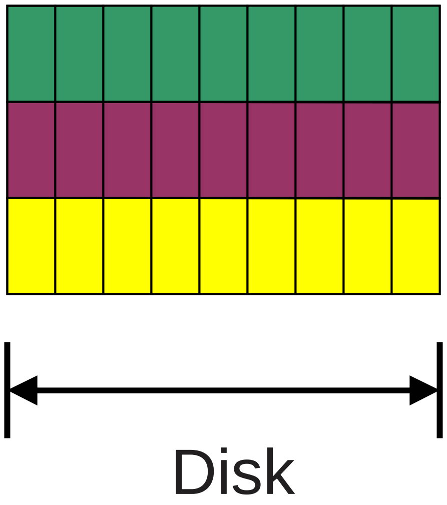
# Outline

- New hardware: Object-based Storage Devices (OSD)
- Vision: Database-aware storage systems
- Example: Transparent device-specific data placement
- Moving forward: Issues with the current OSD specification

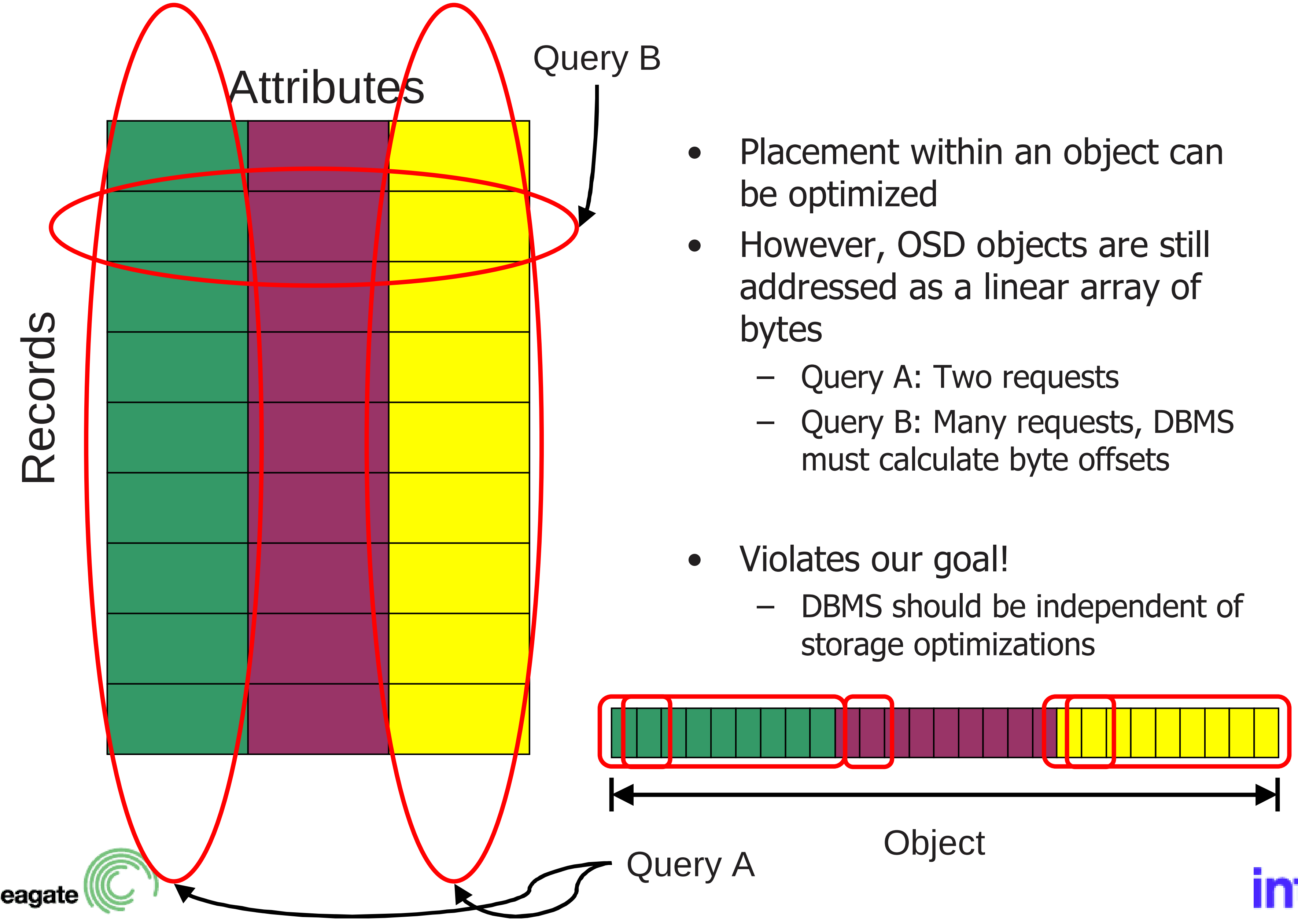
# Linearization problem remains



- Placement within an object can be optimized



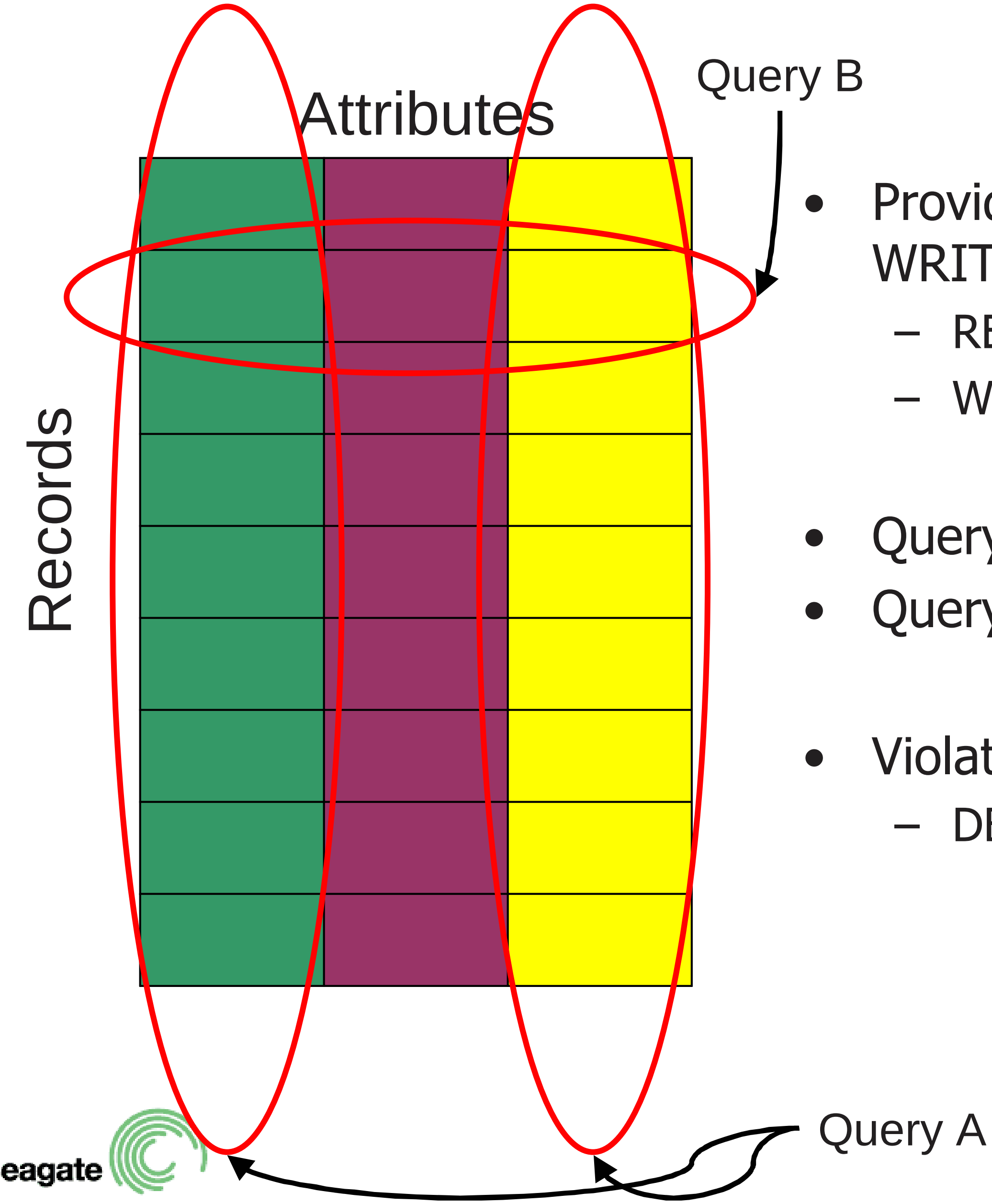
# Linearization problem remains



- Placement within an object can be optimized
- However, OSD objects are still addressed as a linear array of bytes
  - Query A: Two requests
  - Query B: Many requests, DBMS must calculate byte offsets

- Violates our goal!
  - DBMS should be independent of storage optimizations

# Solution #1: Two-dimensional object interface

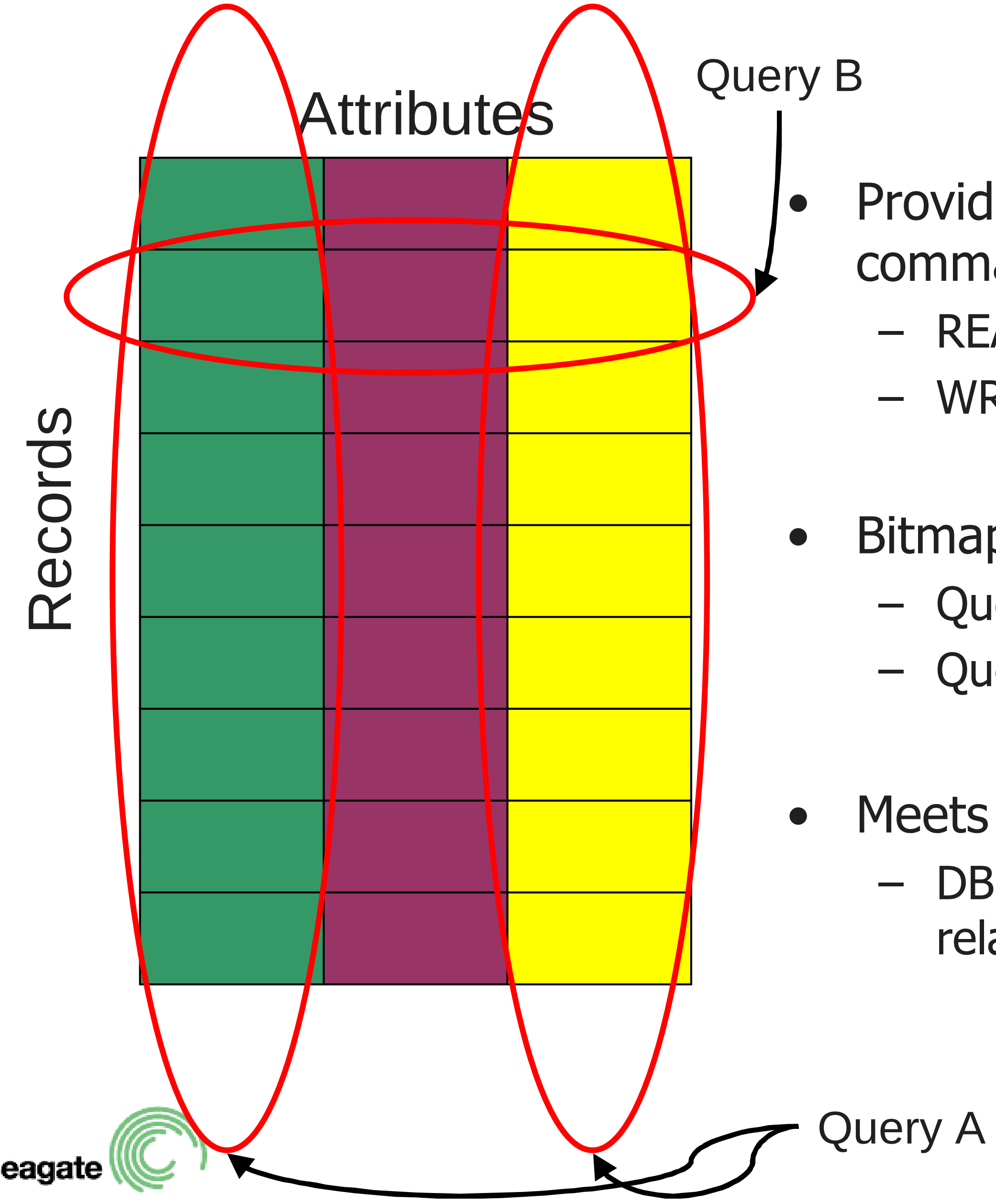


- Provide two-dimensional READ and WRITE commands
  - READ (offsetX, lenX, offsetY, lenY)
  - WRITE (offsetX, lenX, offsetY, lenY)

- Query A: Two requests
- Query B: One request

- Violates our goal!
  - DBMS must still translate byte offsets

# Solution #2: Relational object interface



- Provide relational READ and WRITE commands
  - READ (record offset, len, field bitmap)
  - WRITE (record offset, len, field bitmap)
- Bitmap specifies fields to be accessed
  - Query A: READ(0, 9, 101)
  - Query B: READ(1, 1, 111)
- Meets our goal
  - DBMS addresses object in terms of the relation’s schema

# Conclusion

- Object-based storage devices allow applications to provide storage subsystems with high-level knowledge about data

- Demonstrated how database systems can take advantage of modern placement techniques using OSD interfaces

- Placement is just the first step

- In the future, how can database systems cooperate better with storage systems?