CubeStore

On the Design and Implementation of the Multidimensional CubeStore Storage Manager

Wolfgang Sporer
Wolfgang Lehner
wgsporer@cip.informatik.uni-erlangen.de
Wolfgang.Lehner@informatik.uni-erlangen.de

Department of Database Systems
Friedrich-Alexander-University of Erlangen-Nuremberg
Germany

Outline

A) Problems

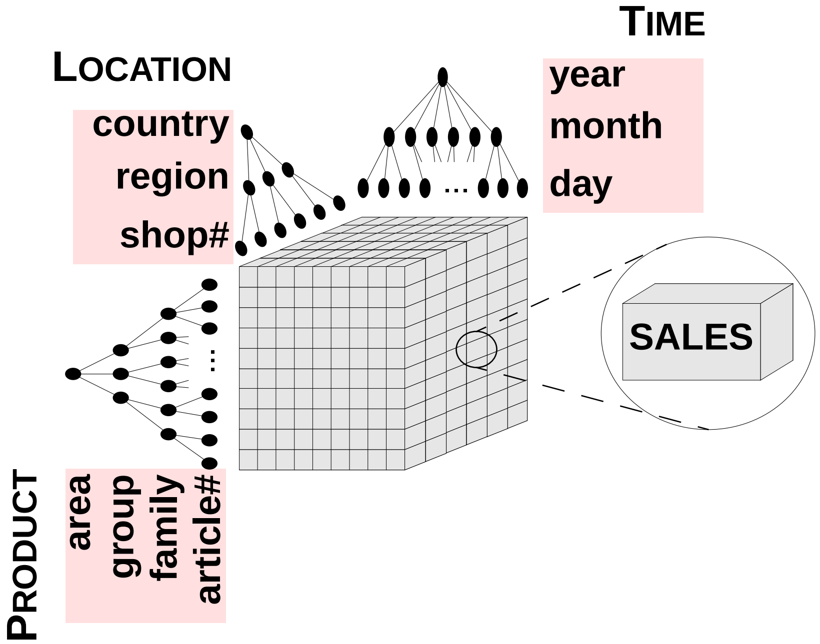
- ❏ Motivation
- ❏ Market Retail Research
- ❏ CubeStar - Project
- ❏ Requirements for CubeStore
- ❏ Redundancy Model

B) Solution

- ❏ Design of CubeStore
  - A Basic Configuration Scheme
  - Two Sample Configurations
- ❏ Implementation of CubeStore
  - Overview
  - Driver Hierarchy
  - Storage Format
- ❏ Conclusion

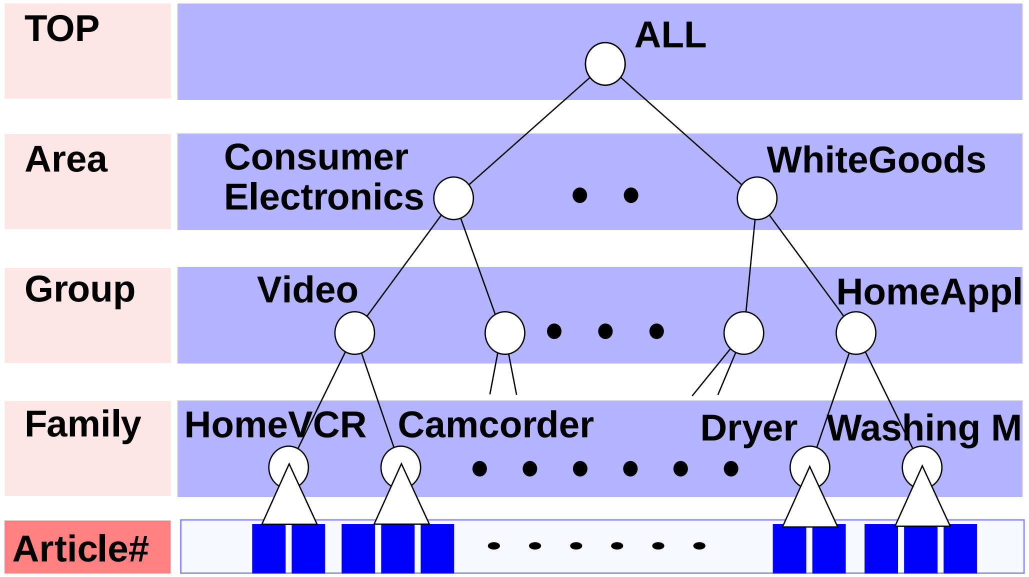
Motivation - SSDB

- SSDB: Statistical & Scientific Databases
- Multidimensionality
  - Quantifying data
  - Qualifying data
- Quantifying data:
  - Several *facts* like sales, stock
  - High sparsity
- Qualifying data:
  - *Dimensions*:
    products, shops, time, etc.
  - Hierarchical structure
  - Many 'dimensional attributes'

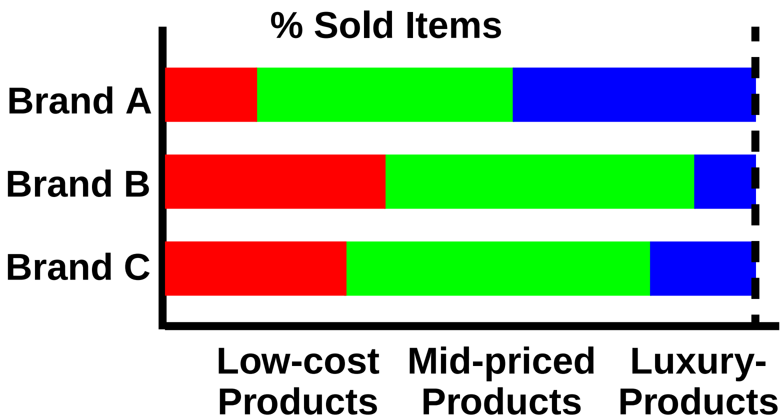


Example: GfK - Market Retail Research

- GfK: large German market research company
- Product dimension
  - 250.000 products in 400 product families
  - 5 global features and about 15 additional features in each family



- Standard analysis queries
  - Raw data grouped and aggregated by different categorizations
  - Examples: product groups, countries, brands and shop types
  - Calculation of sums/averages of sold items, turnover, stock, etc.



Data Volume - Think Big

Today

- Typical panel:
  consumer electronics panel
  in Germany
  - every other month
  - #items sold, price, etc.
  - 250.000 products,
    400 product groups, 5.000 shops

Product	Shop	Period	Sales	Price	...
1012	0001	3/98	150	80	...
5444	0001	3/98	5	1000	...
1012	0114	3/98	100	84	...
...	...	...	...	...	...

- Data volume:

Tomorrow

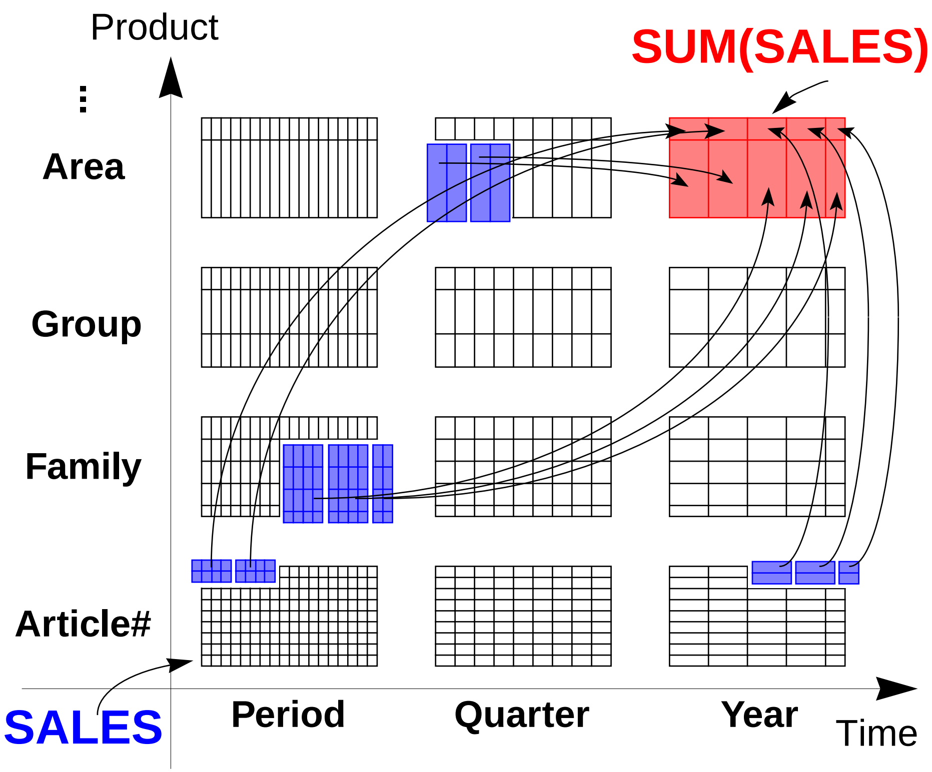
		Duration of Storage			
		1 Month	**2 Months**	**1 Year**	**2 Years**
Frequency	**Bi-monthly**	--	10 GByte	60 GByte	120 GByte
	Monthly	10 GByte	20 GByte	120 GByte	240 GByte
	Weekly	40 GByte	80 GByte	480 GByte	960 GByte

International Analysis

Performance

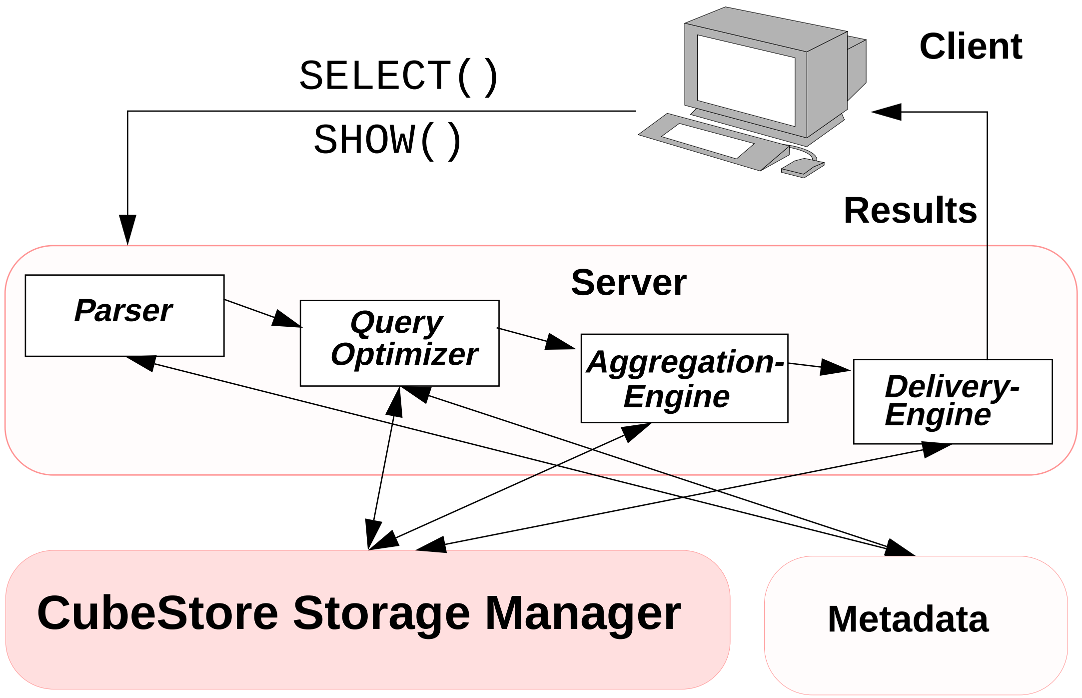
- ❑ Online access
  - Batch jobs may wait - online users are impatient!

- ❑ Solution:
  - Users request aggregated figures (sums, averages, etc.)
  - ➥ Pre-*aggregate* data, keep aggregates and avoid using raw data
  - Dynamic aggregation management necessary



The CubeStar-Project

- ❑ Middleware implementing an INFORMATION-EVERYWHERE approach
- ❑ Client-Server architecture
- ❑ Separation of
  - query execution: `SELECT`
  - delivery of results: `SHOW`
- ❑ Features:
  - Application-oriented data modeling using CROSS-DB
  - Query execution: Redundant hierarchy of materializations



- ❑ CubeStore:
  - Storage Management within CubeStar

Requirements for CubeStore

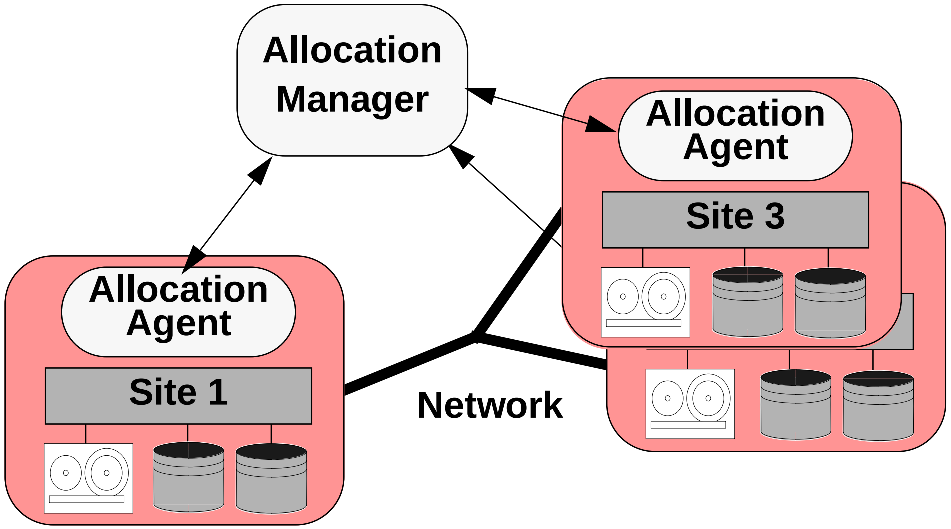
- *Performance*
  - Online data access
  - Fast “range queries”
- Support for different types of storage media
  - Adequate handling of large data volumes
  - Hierarchy of storage media: price vs. performance
  - Open concept: media-independent interface
  - Seamless integration of different storage media
- External media classification according to
  - Access costs
  - Transfer costs
- Optimized physical storage formats

Requirements (2) - Flexible Configuration

- Adaptability to
  - Users changing needs
  - Different data characteristics

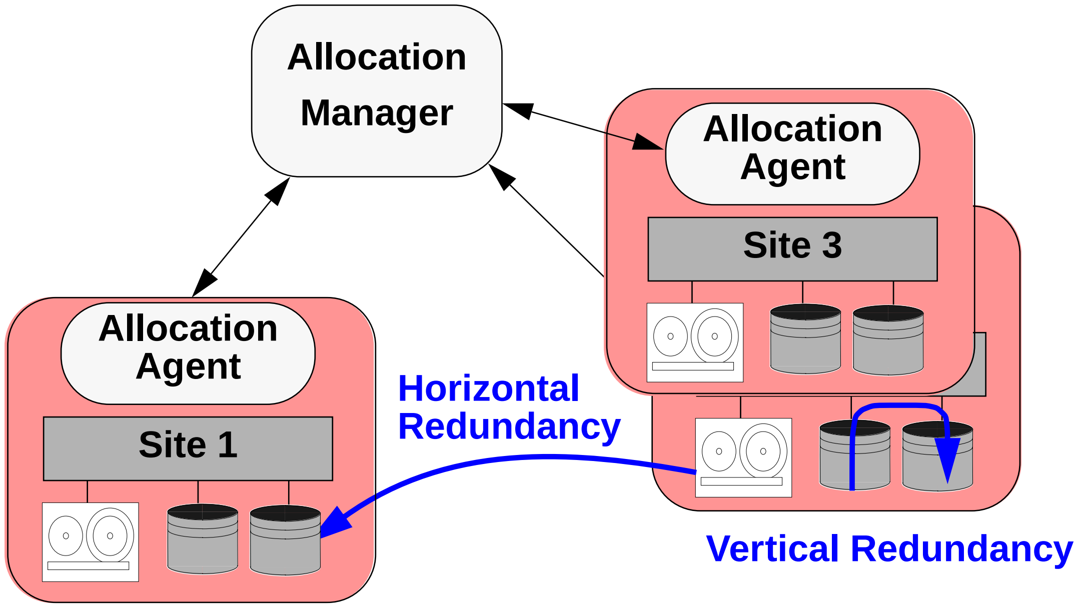
- *What* data, *where* and *how* to store data? - Replication strategies

- Distribution / allocation policy (higher levels of CubeStar)
  - Controlled by a global *allocation manager*
  - Supported by local *allocation agents*
  - Basis for distributed query execution



Redundancy Model

- Theory:
  - One logical object x ↔ several physical objects $x_i = f(x)$
  - *Replication*:
    horizontal redundancy
    - $x_i = id(x)$
  - *Aggregation*:
    vertical redundancy
    - e.g. $x_i = sum(x)$

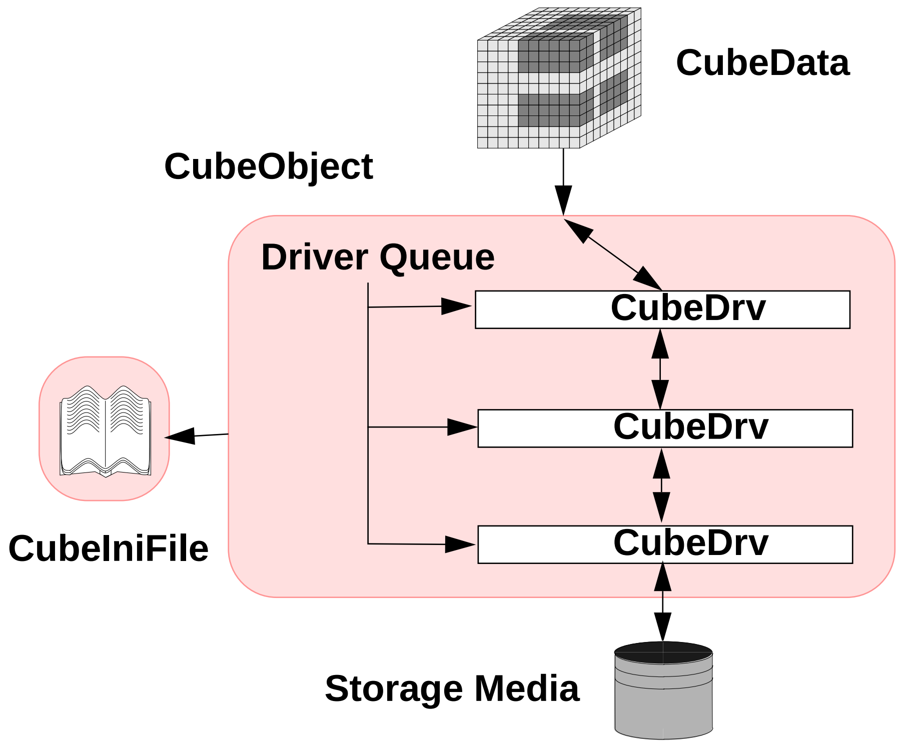


- Enables *efficient online* access to (logically) very large data volumes

- Applicable because of special data access pattern:
  - Production period with bulk updates
  - Afterwards mostly read-only

The Design of CubeStore

- CubeObject
  - Logical object
  - Provides an abstract interface for accessing multidimensional data
  - Hides details of different storage media
- Main components
  - CubeDrv: accessed via driver queue
  - CubeData
  - CubeIniFiles



A Basic Configuration Scheme

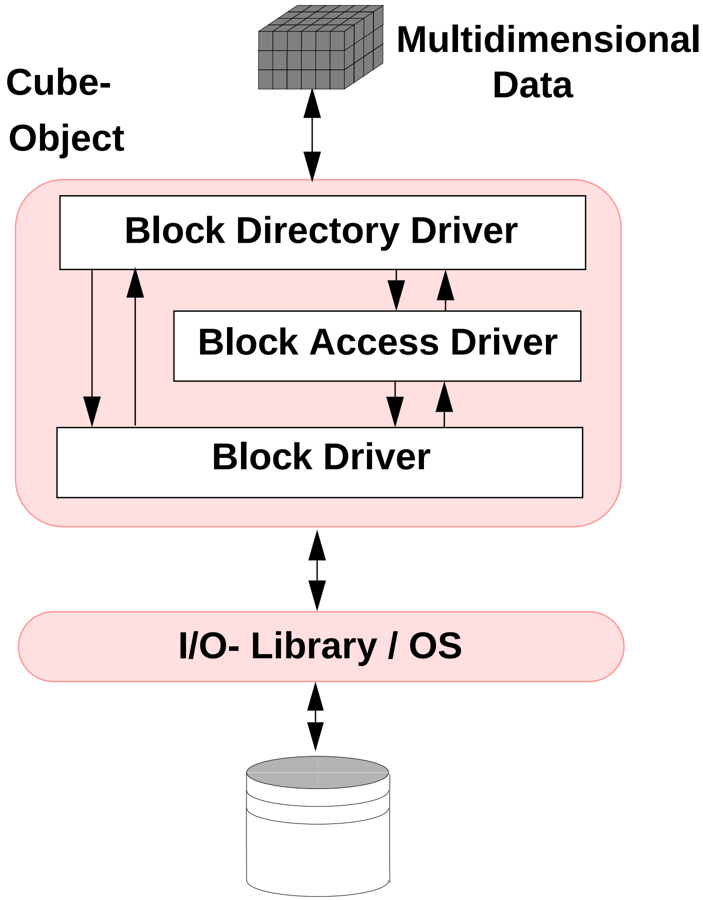
❑ Multidimensional data partitions

divided by

CubeObjects

into

Single data blocks



❑ Two sample configurations:

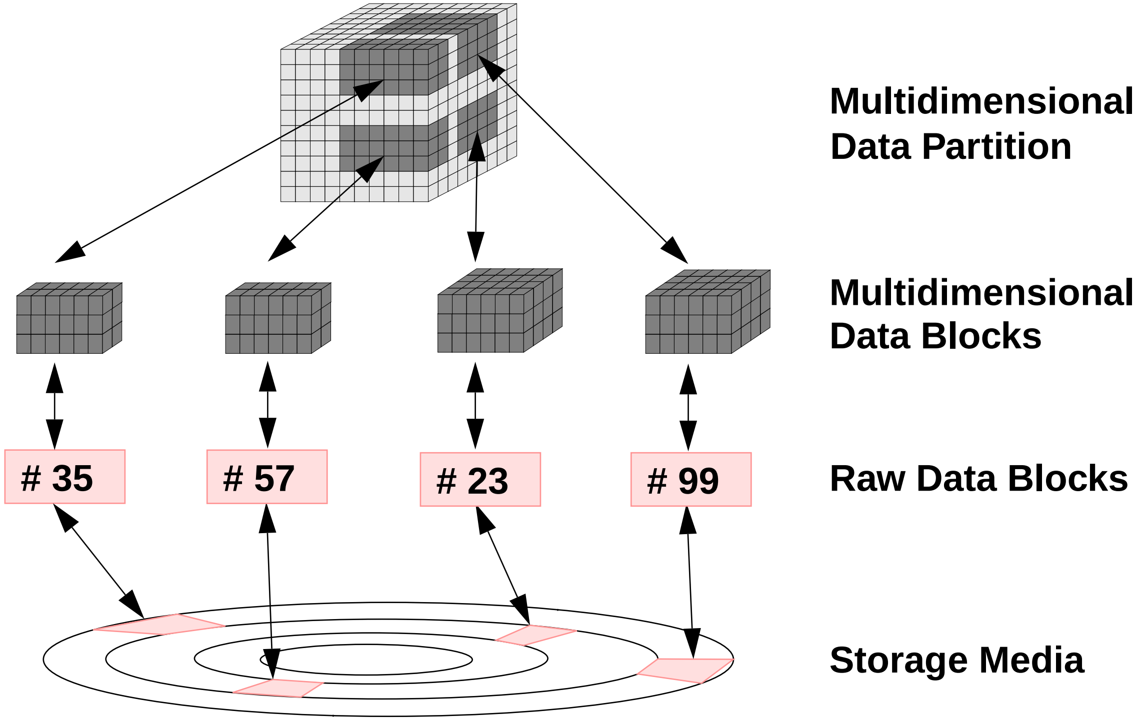
- Storage media with fixed block size
- Storage media with sequential access

❑ Other solutions: e.g. driver for a relational database system

Fixed Block Size

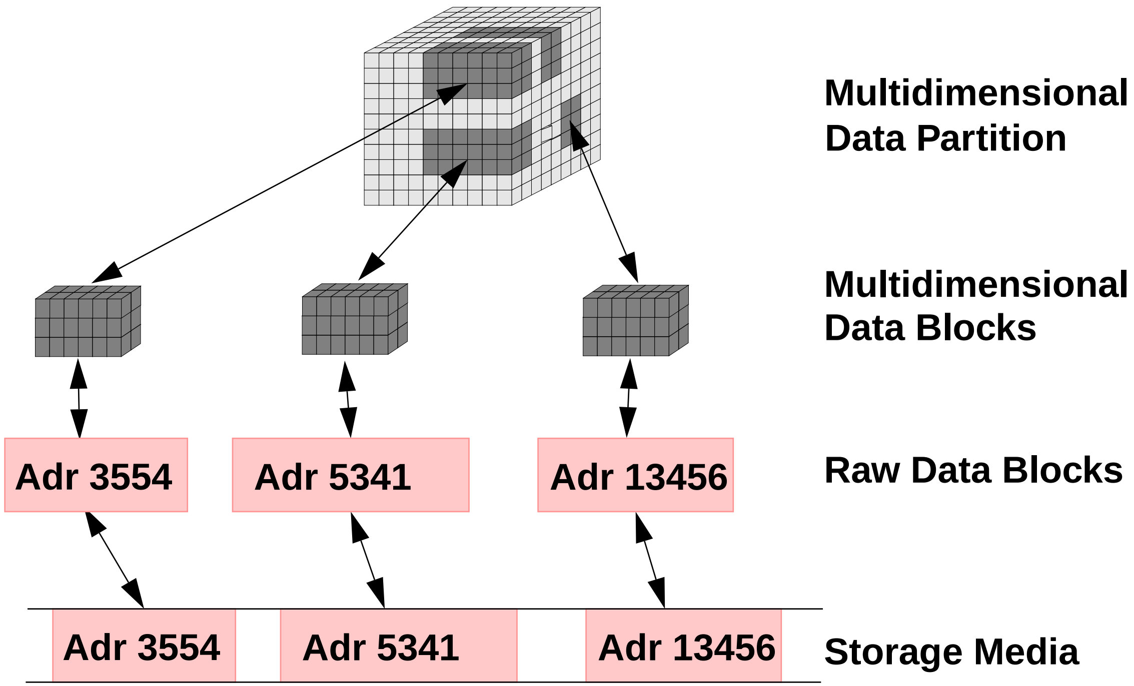
- Storage media with fixed block size

  - Intelligent block directory driver
  - Block access driver compresses to fixed block size
  - Block driver straightforward



Sequential Access

- Storage media with sequential access
  - Block directory driver uses standard block mapping
  - Block access driver compresses as good as possible
  - Block driver accepts variable sized blocks



Implementation of CubeStore

- Prototypical implementation in C++:
  - Optimized for read-access
  - Storage media with fixed block size
- Configuration:
  - CubeIniFile
- Drivers: CubeDriverFactory
  - Global instance for driver instantiation
  - Makes use of new driver classes easier

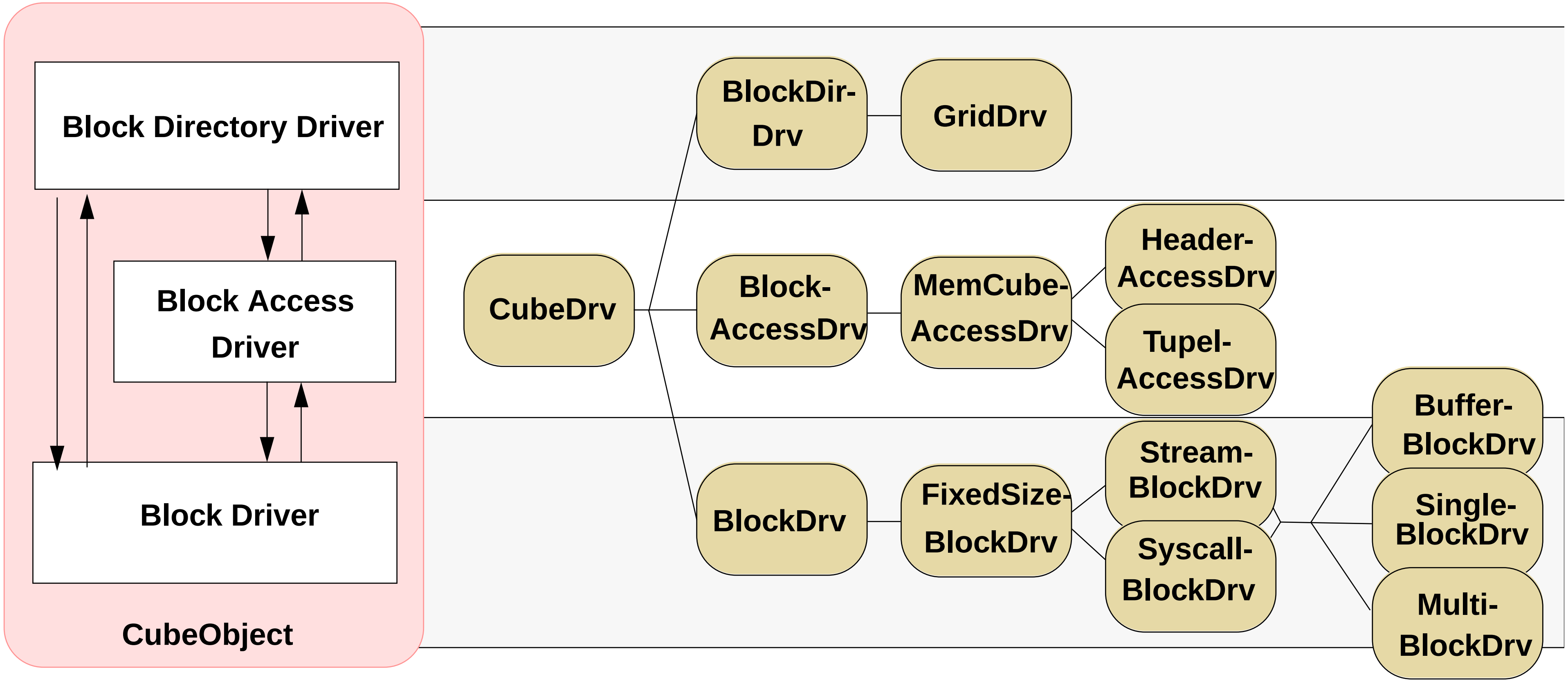
```
SECTION-DRIVER
        DRIVER-MultiBlockDrv
                blocksize: 8192
                filename: BigFile
        DRIVER-END
        DRIVER-TupelAccessDrv
        DRIVER-END
        DRIVER-GridDrv
                splitstrategy: roundrobin
        DRIVER-END
SECTION-END
```

A simple CubeIniFile

- Data access

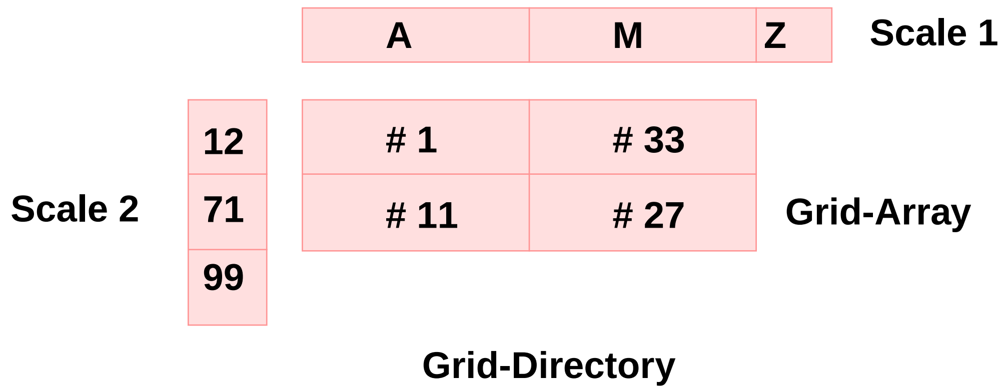
```
Bool Read(MemCubeData<T>& cube_data) {
        return (drv_queue.First())->Read(cube_data);
}
```

Hierarchy of Driver Classes



Data Directory: Grid File Format

- By Nievergelt, Hinterberger et. al 1984
- N-dimensional directory accessed by N separate range scales
  - Directory: contains data block numbers
  - Scales: describe ranges of stored data values



- Benefit:
  - Inherently multidimensional: all dimensions treated equal
  - Self-organizing growth
  - Updates mostly require only local changes
  - Good integration with different access techniques
  - Suitable for “range queries”

Data Compression: SingleCountHeader

- By Eggers, Olken, Shoshani 1981
- Compression by removing one value from an array
  - Here: eliminating Null-values, i.e. non-existing values
  - Used within a data block
  - Why? - Sparsity of the data cube
- Benefit:
  - Coordinate values can be omitted
  - Physical disk space and logical size nearly independent
  - Logarithmic read access without full decompression

Null	**Null**	**33**	**21**	**447**	**Null**	**Null**	**Null**	**99**	**87**	**Null**	**Null**	**Null**	**Null**	**1**	**96**

Logical representation

$\Downarrow$

Header

2	**3**	**5**	**5**	**9**	**7**

+

33	**21**	**447**	**99**	**87**	**1**	**96**

Physical representation

Conclusion

- Architecture
  - Flexible and adaptable
- Implementation
  - Prototypical
  - Fixed block size
- To Do:
  - SQL-Driver
  - Efficiency

CubeStar

- http://www6.informatik.uni-erlangen.de/research/cubestar/index.html