



# *Conditions Database Evaluations*

---

James Ohnemus

*James\_Ohnemus@LBL.Gov*

David R. Quarrie

*DRQuarrie@LBL.Gov*

# *Overview*

---

---

- Common features of OO databases
- Description of databases under evaluation
  - Objectivity
  - ObjectStore
- Benchmark Goals
- Data Model & Data Organization
- Benchmark Results
- Derived Results
- Recommendation
- Status of prototype

# *OO Database Features*

---

---

- Persistent Storage
  - Lifetime of objects longer than creating process
  - Most OO databases store only data members
- Transactions
  - Manage concurrent access. Ensure integrity of database
- Queries
  - Mechanism to select items from a collection
- Schema
  - Set of class definitions stored in database to ensure consistency
- Schema Evolution
  - Modifying existing objects if the schema is modified

# *OO DB Features (cont.)*

---

---

- Collections
  - Bags, Sets, Lists etc. of objects
- Associations
  - Mechanisms to allow objects to reference each other
    - ▶ One-to-one; one-to-many; many-to-one; many-to-many
    - ▶ Uni-directional, bi-directional
- Indexing
  - Technique allowing rapid access to members of a collection based on some characteristics
- Heterogeneity
  - Ability to store and retrieve information between different hardware architectures

# *Objectivity*

---

- Persistence enabled by inheritance from persistent base class
  - *ooObj* (ODMG-93 *d\_Object* in V4)
- Overloaded *new* operator
  - Transient or persistent
  - Clustering information (database ID, object ID etc.)
- Data Definition Language (DDL) defines data model.
  - Compiled into C++ header file & associated .cc files
- Smart object references
  - *ooRef*, *ooHandle* (ODMG-93 *d\_Ref<T>* in V4)
  - Based on 64-bit Object ID (OID)
  - Reference accesses virtual memory when object in client cache
  - Based on storage hierarchy (next slide)

David R. Quarrie: Conditions Database Evaluations

# *Objectivity (cont.)*

---

---

- Storage Hierarchy
  - Federated Database
    - ▶ One per system - database catalog, schema information, lock-server
  - Databases
    - ▶ Unix files - up to 32k per federated database
  - Containers
    - ▶ Regions within database; basis of indexing & iterators
  - Objects
    - ▶ C++ objects inheriting from *ooObj*
  - Pages
    - ▶ Access is page based - all objects on a page accessed together
  - Client cache
    - ▶ *ooRef* points to object in client cache if resident

# *ObjectStore*

---

---

- Anything can be persistent rather than just C++ objects
  - Even single integers or floats etc.
  - No inheritance from persistent class
- Overloaded *new* operator
  - Identical in concept to Objectivity
- Schema Generation File
  - Similar to Objectivity DDL file
- Conventional C++ pointers
  - Swizzled to point to virtual memory when object resident

# *ObjectStore (cont.)*

---

---

- Storage Hierarchy
  - Databases
    - ▶ Correspond to Unix files
  - Segments
    - ▶ Set of pages within database
  - Clusters
    - ▶ Contiguous (1-15) pages
  - Objects
  - Virtual memory, not client cache
- Collections
  - More complete than Objectivity
    - ▶ But Objectivity provides Tools.h++ (not used in these tests)

# *Benchmark Goals*

---

---

- Targeted towards accessing objects by time interval (or run number range)
  - These tests used run number range but will use time intervals for BaBar
  - Goal is to rapidly locate conditions information corresponding to an event, based on the run or time stamp
  - Different conditions information has different validity ranges
- Measure Performance & Overheads
  - Both loading and accessing the database
  - How performance scales with size & number of objects etc.
  - Effect of transactions

# *Data Model*

---

---

- Two Classes used
  - *BdbCalibration*
    - ‡ Objects to be located.
    - ‡ Contains varying amounts of data
      - 1, 10, 100, 1000, 10000 values
  - *BddInterval*
    - ‡ Denote the range of validity of the corresponding *BdbCalibration* object
    - ‡ Start date, time & run number
    - ‡ End date, time and run number
      - Run number uses for these tests
    - ‡ Reference to corresponding *BdbCalibration* object

# *Data Organization*

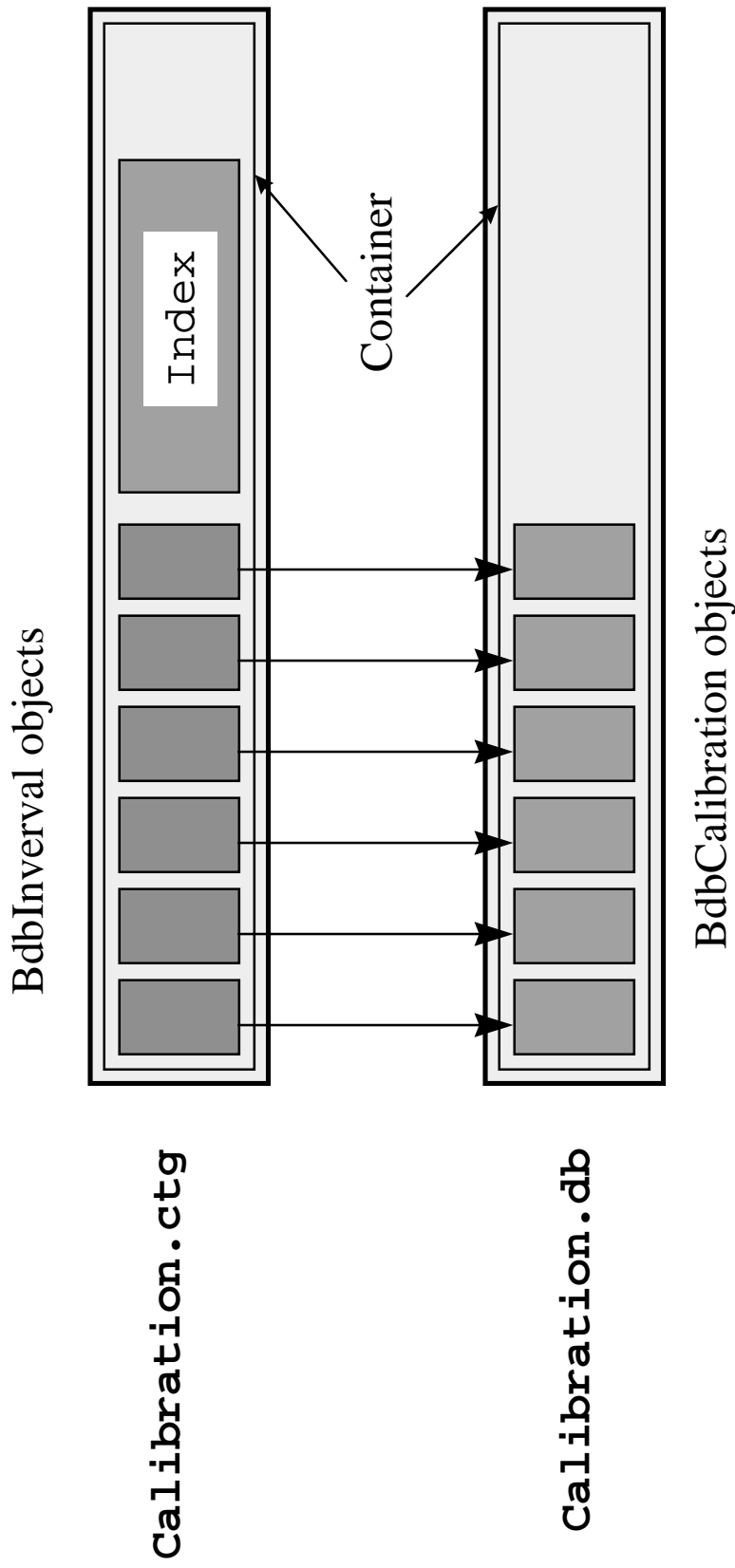
---

- Two different databases
- *BdbInterval* objects are small
  - Keep in single file for duration of experiment
  - Use database indexing to access based on interval
- *BdbCalibration* objects can be large and complex
  - Split into fixed size databases
  - Current tests based on single database
- Best organization for whole experiment?
  - Different database for every conditions class?
  - Different database for every detector subsystem?
    - ◆ All conditions information for subsystem in same database

# *Data Organization*

---

---



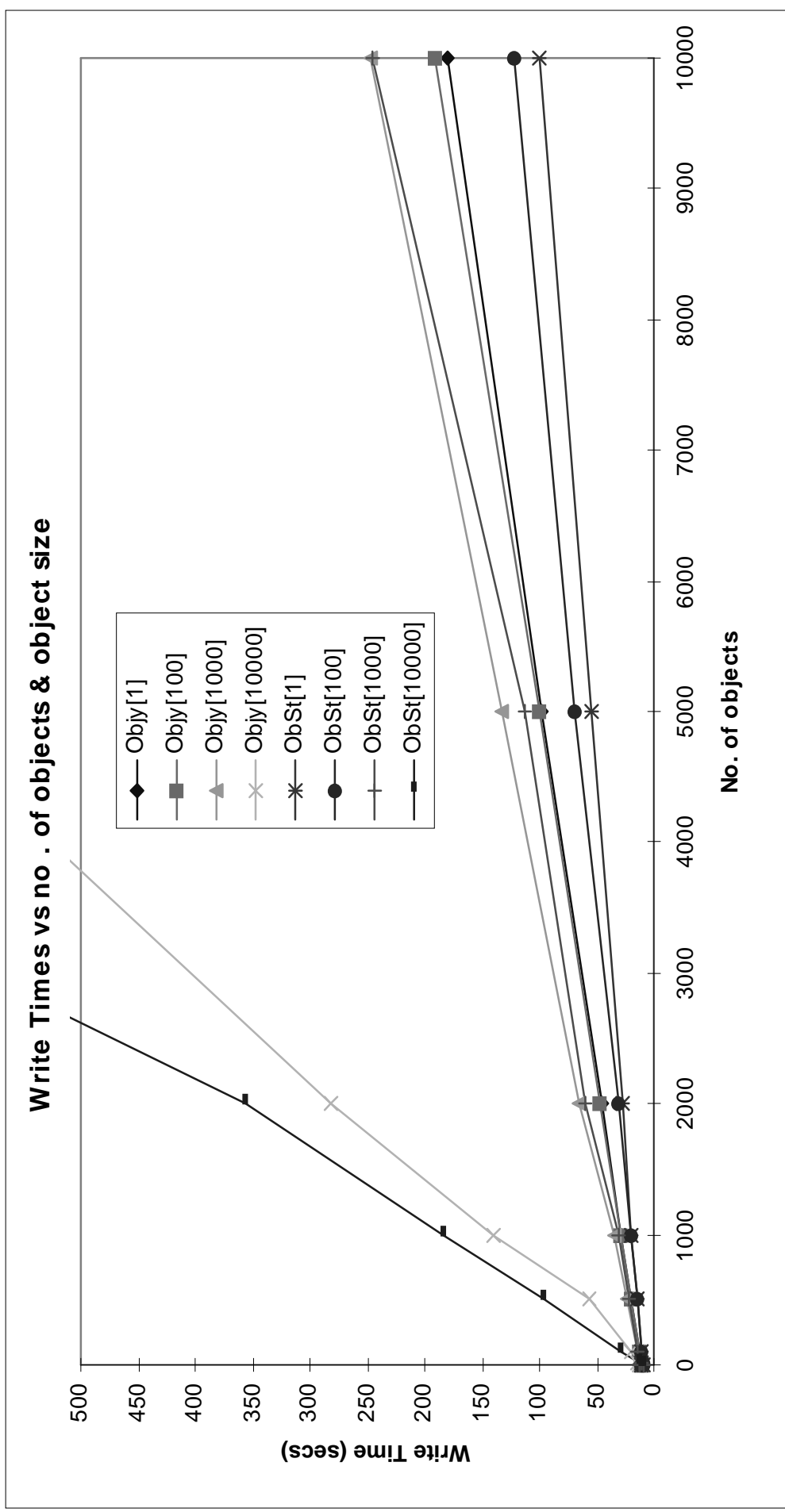
# *Benchmark Results*

---

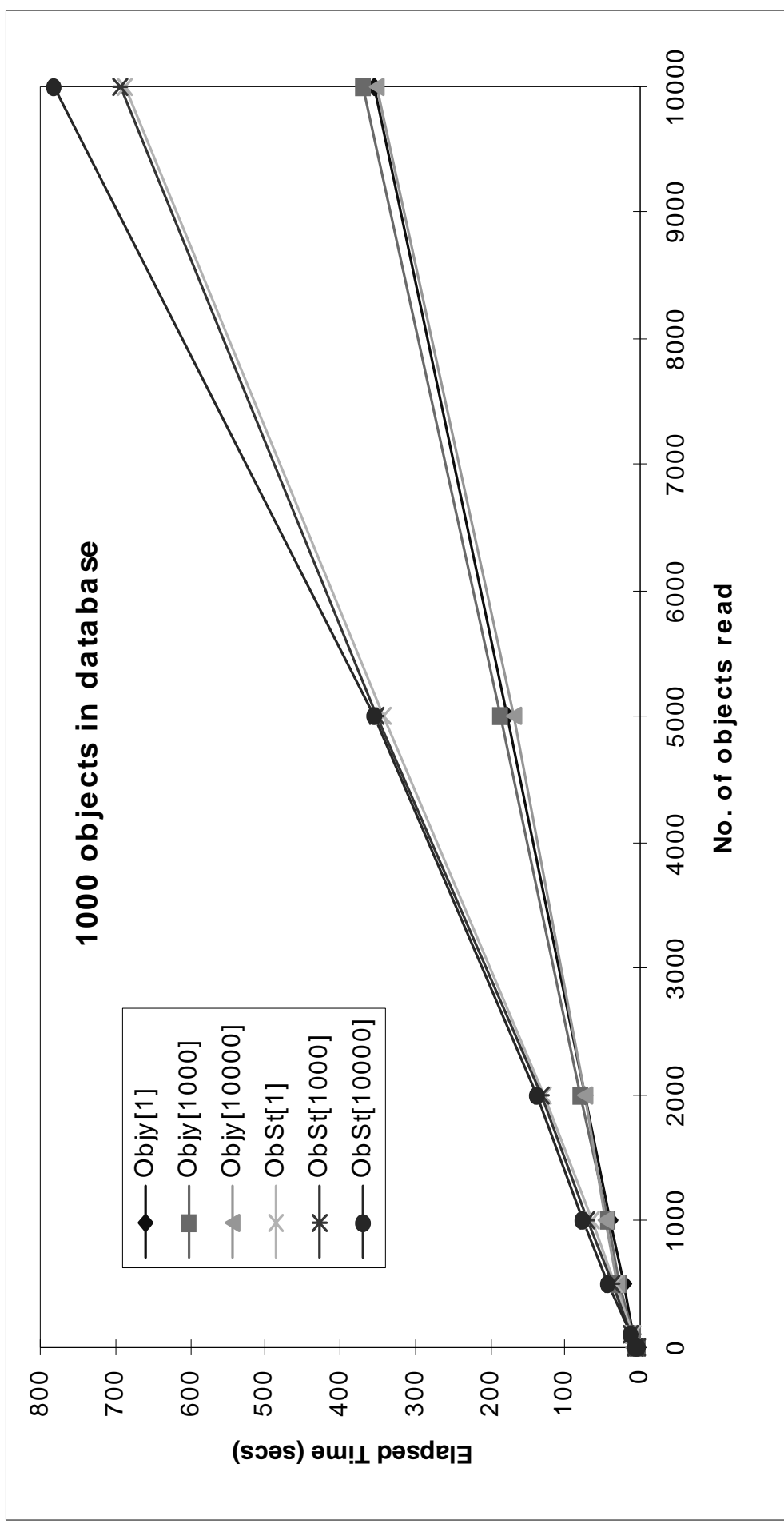
---

- Write Performance
  - As function of number of objects
  - As function of object size
- Read Performance
  - As function of number of objects
  - As function of object size
  - As function of object location in database
  - As function of transaction boundaries
  - As function of numbers of databases opened
- Database Size
  - As function of number of objects & object size

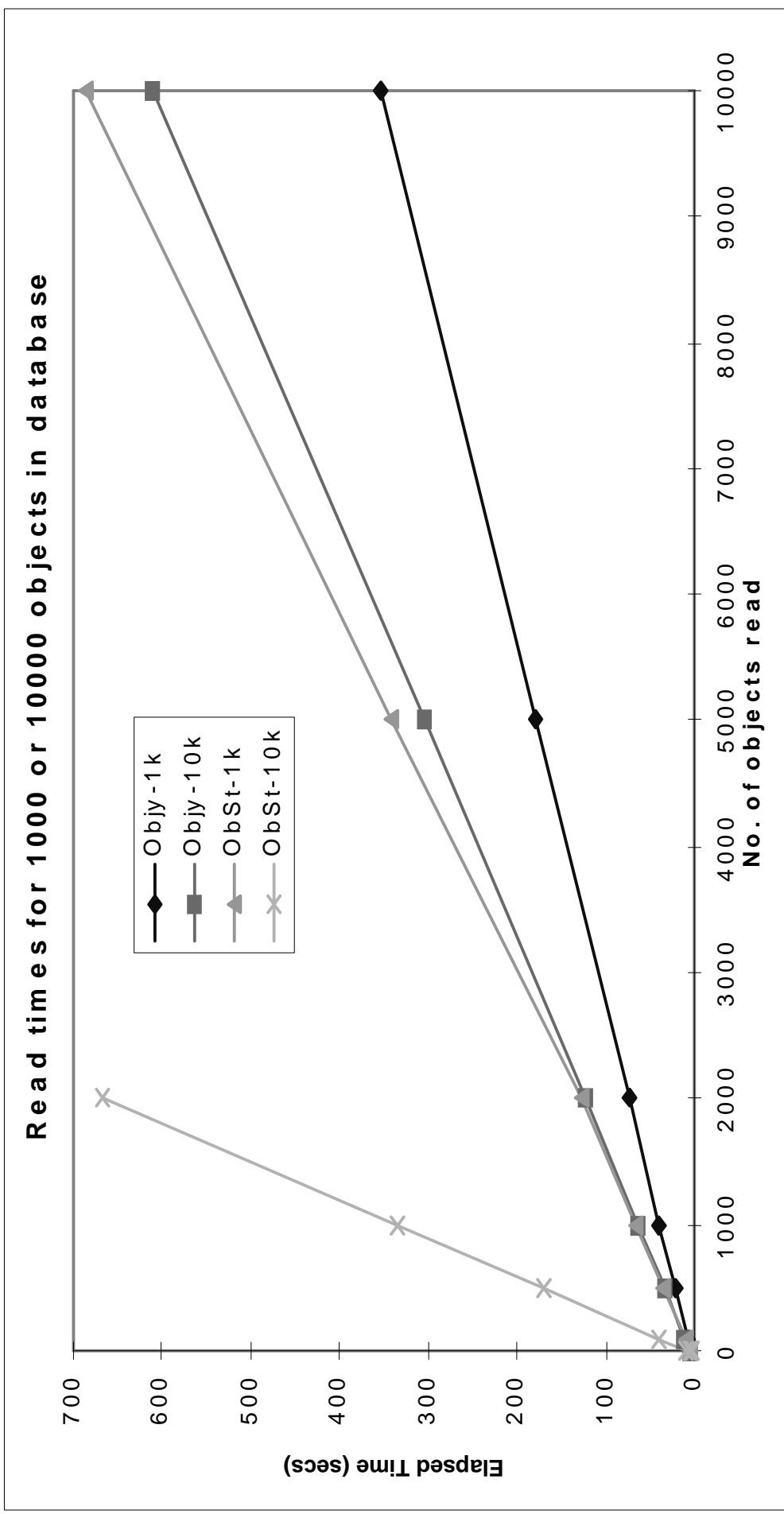
# Write Performance



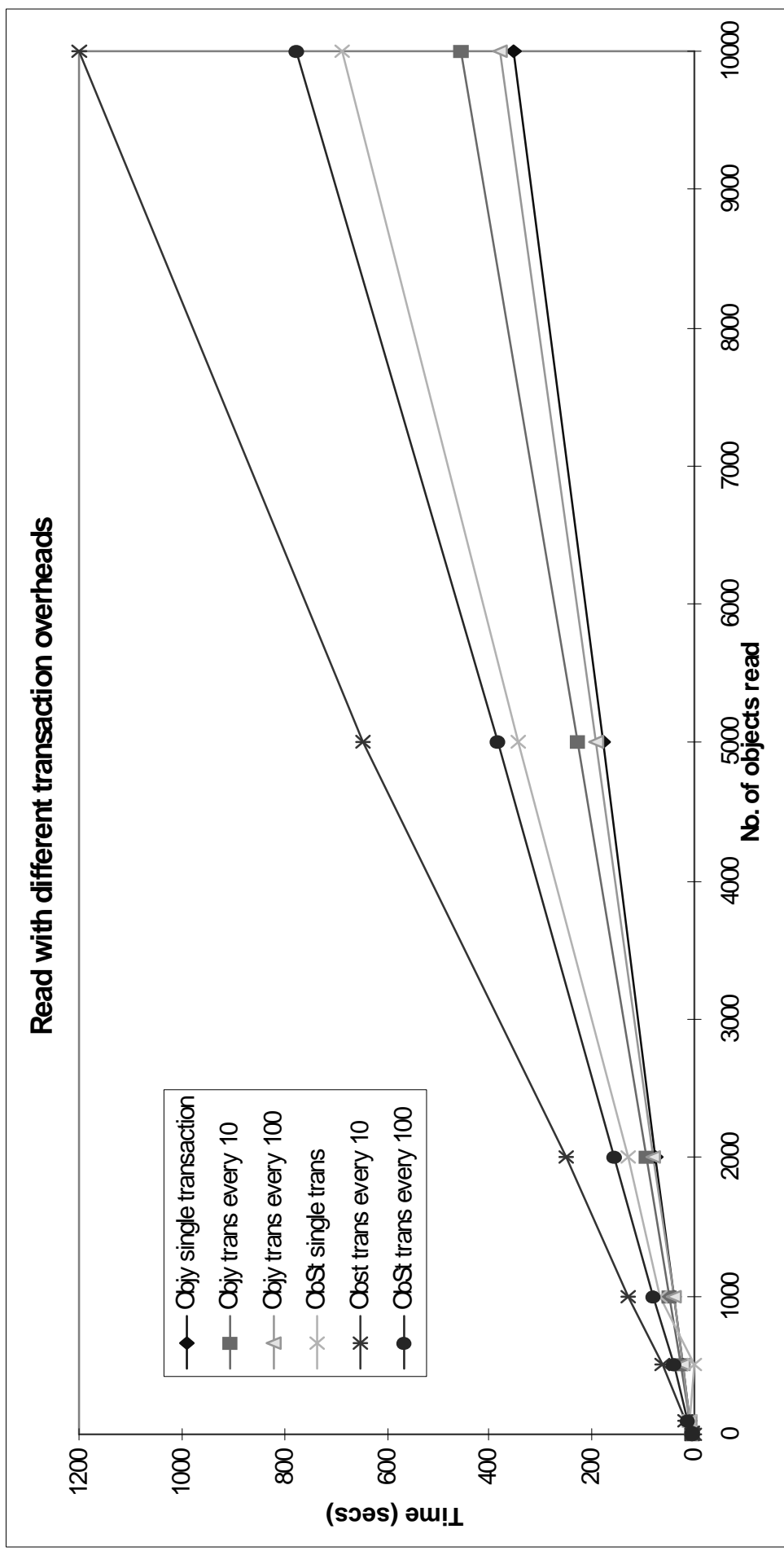
# *Read as function of Object Size*



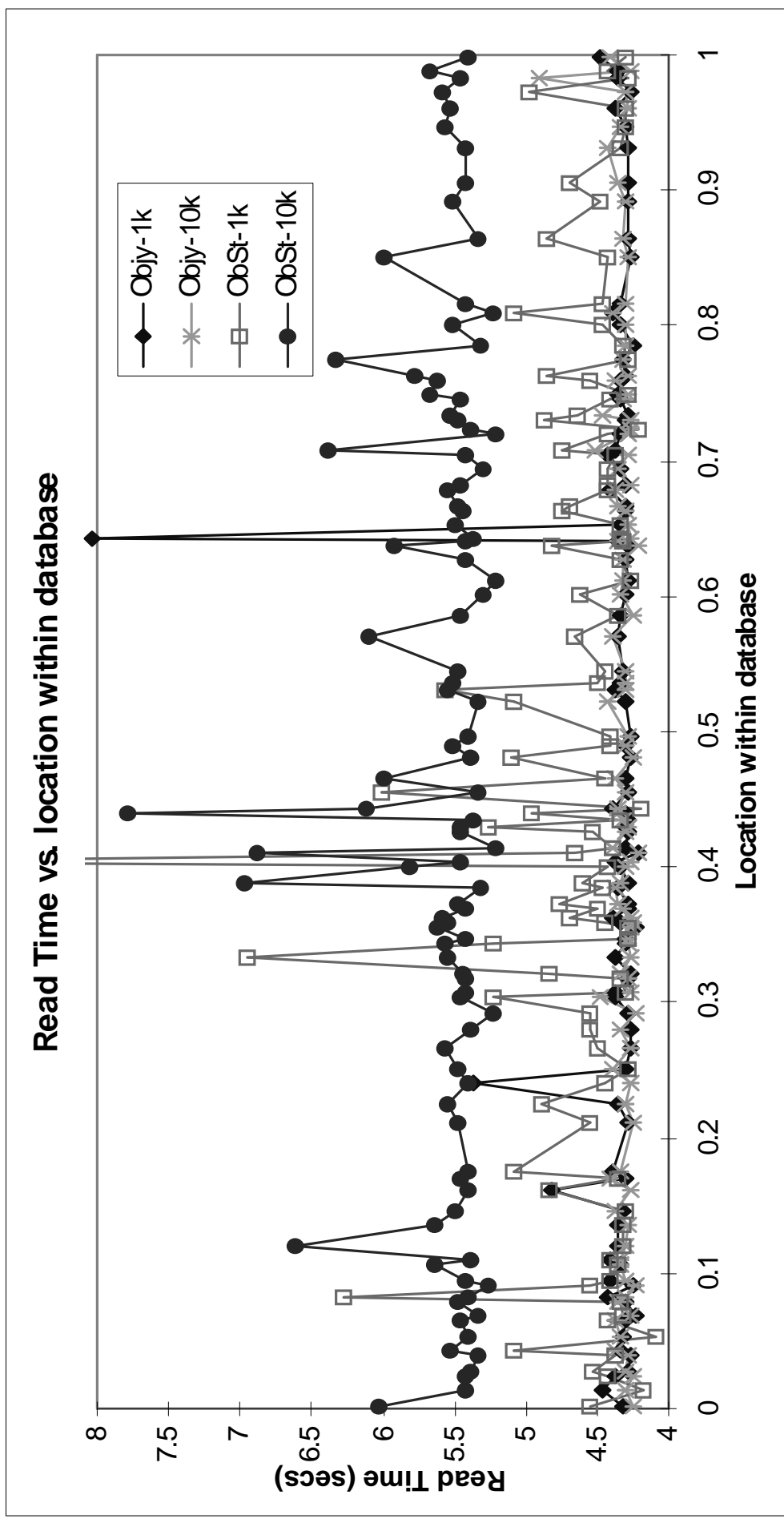
# *Read as function of no. of Objects*



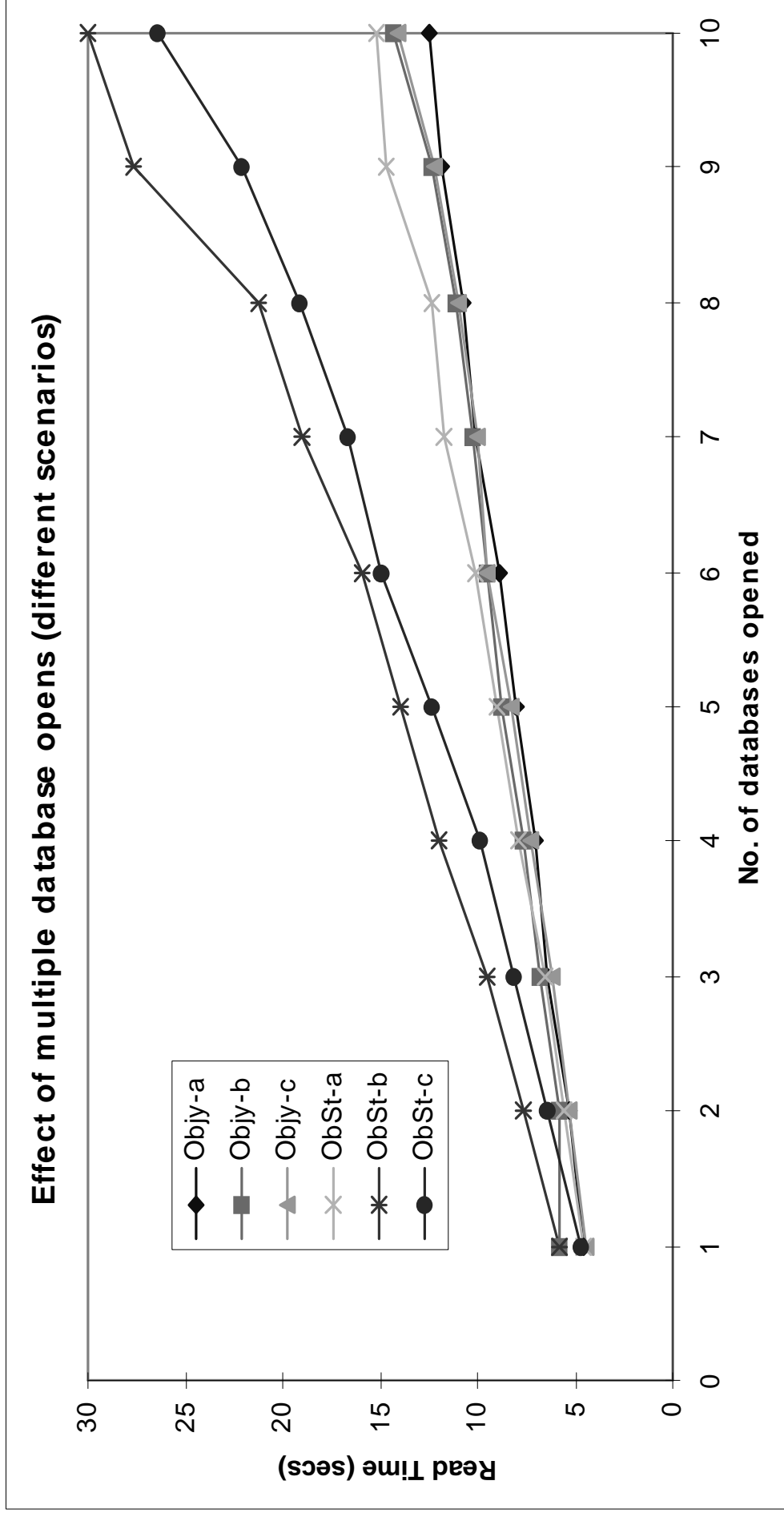
# Read & Transactions



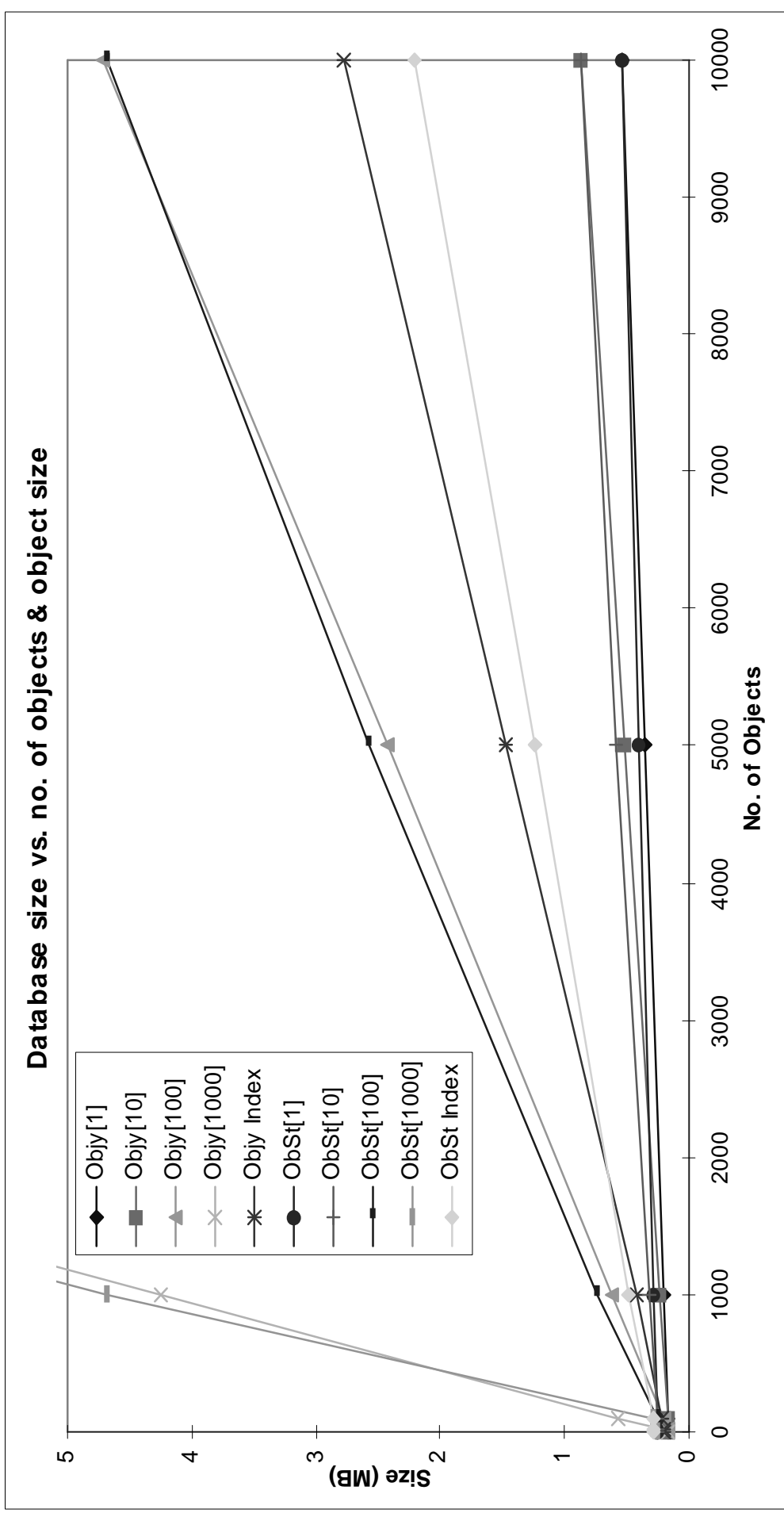
# *Read vs. object location*



# Multiple databases



# Database Size



# *Evaluation Summary*

---

Quantity	Objectivity	ObjectStore
Application overhead (1 database)	~4 secs	~4 secs
Cost per database open/close	0.89 secs	1.17 secs
Cost per transaction	0.28 secs	0.87 secs
Write performance (includes Index update)	28% of Unix	21% of Unix
Search time scaling	$\sim 8 \log_2(N)$	$\sim 80 \log_2(N)$
Fixed overhead per database	$\sim 160$ kbytes	$\sim 250$ kbytes
Overhead per object	14-80 bytes	8-200 bytes

# *Evaluation Notes*

---

- Tests on SparcStation 5
  - Locally mounted (slow) disk
- Write performance depends on object size.
  - Quoted numbers are for large objects
  - Probably dominated by updating Index
    - ▶ Vendors quote better
- Space overheads not fully understood
  - Measured overhead per object varies widely
    - ▶ Presumably an issue of page size
- Search scaling for Objectivity much better than ObjectStore
  - Relative performance improves as no. of objects increases

# *Recommendations*

---

---

- Recommend selection of Objectivity
  - Most test results are better or equal to ObjectStore
  - Better scaling across transactions boundaries than ObjectStore
    - ▶ Because of use of *ooRef* rather than conventional C++ pointer
    - ▶ Not included in these tests
  - Use by RD45 for their event store project
    - ▶ BaBar now active collaborators with RD45
    - ▶ Common solution for both database uses - conditions & event DB
  - DB Partitions might allow automatic Regional Center management
  - SQL interface provides mechanism to input construction information
- Still intend to use ODMG-93 API where possible

# *Prototype Design*

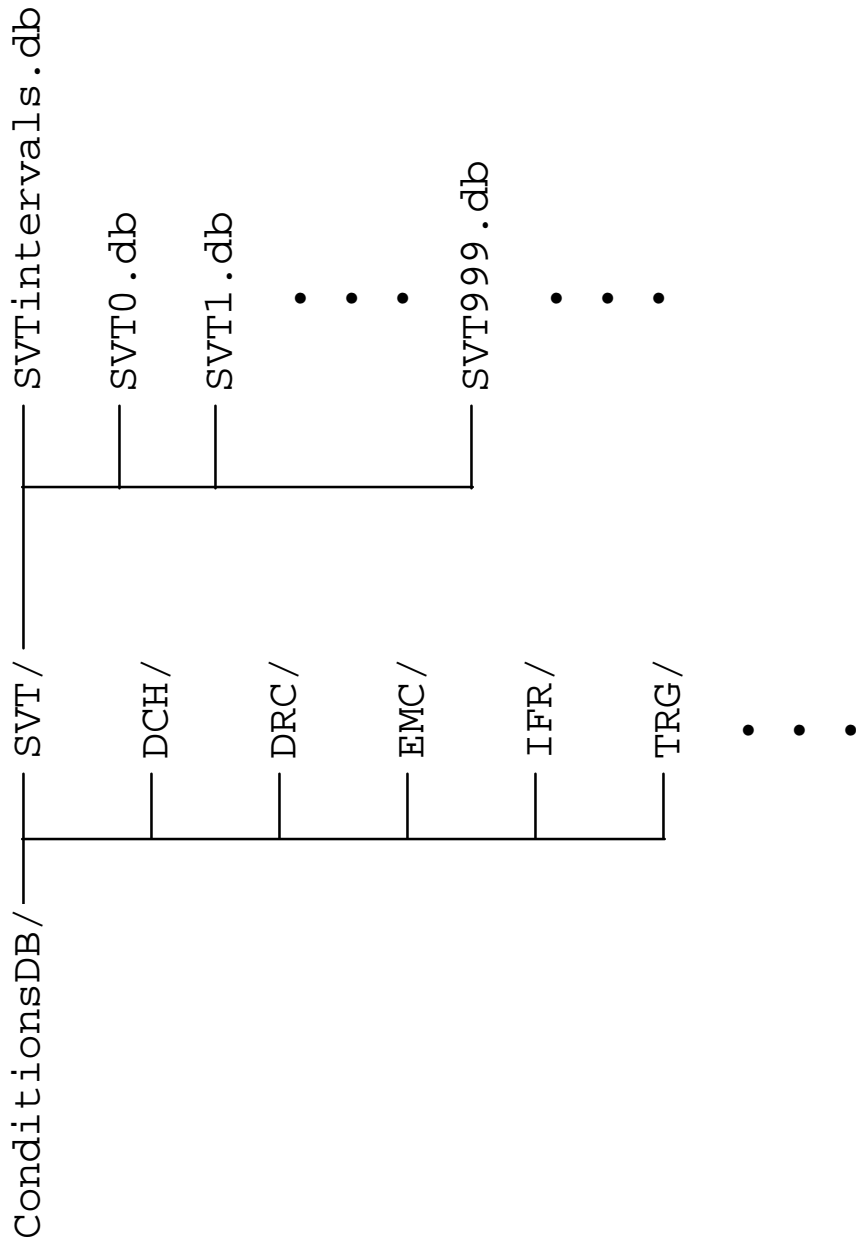
---

- Based on Objectivity
- Design underway
- *BdbInterval* class uses *BbrTime* objects for start & end time
  - Separate mapping between run number and time intervals
- All conditions classes inherit from *BdbObject*
- Database split into subsystems
  - SVT, DRC, EMC, IFR, TRG, etc.
- Databases for each subsystem in separate directory tree
- Different conditions classes kept in different containers
- Database files fixed in size (adjustable for now)

# Directory Tree

---

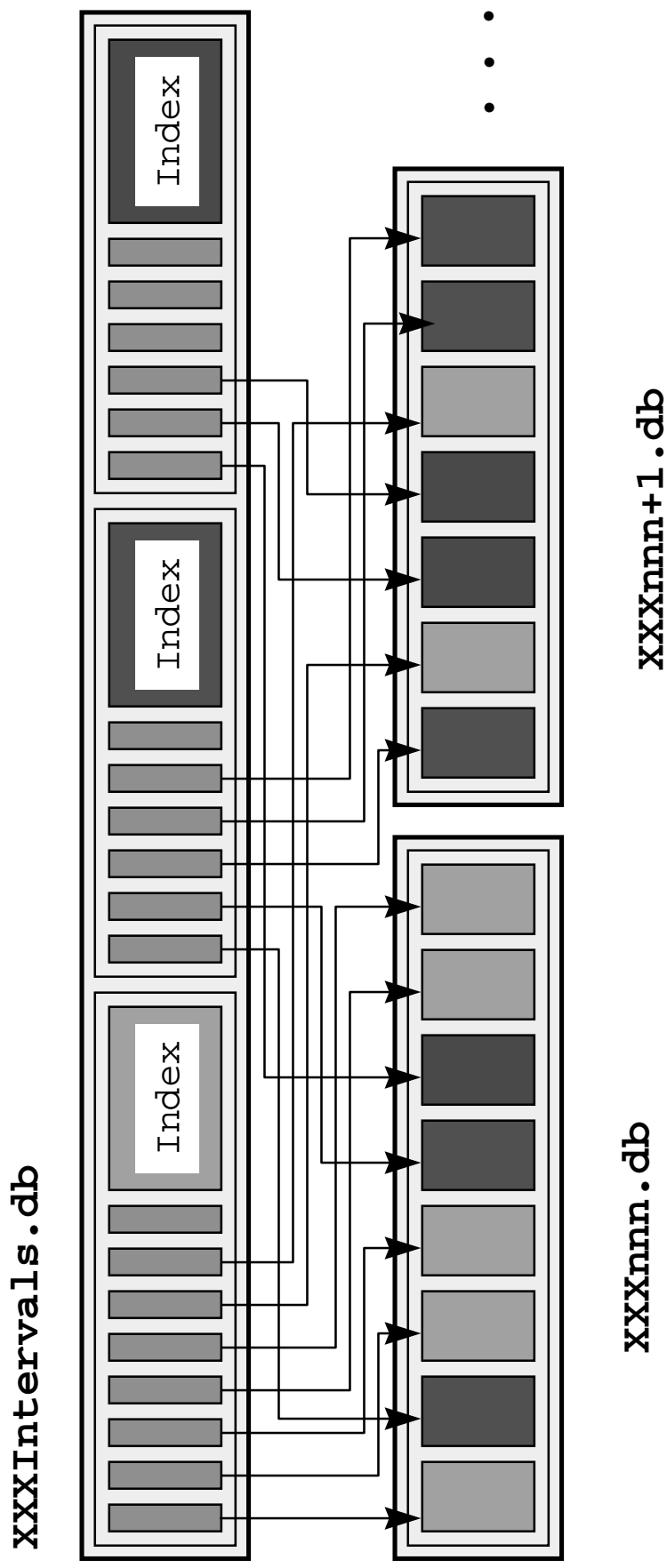
---



# *Data Organization*

---

---



# *Prototype Status*

---

---

- Design underway
- Areas still needing more understanding
  - Trade-off between database & container
    - ▶ Overhead of multiple containers per database not yet known
      - Just measured - small compared to overhead per database
  - Optimum database size
  - Is the detector subsystem the correct mapping to directories
    - ▶ Need another layer?
  - How to deal with repeated calibrations
    - ▶ Can't use data versioning because of different time intervals
- Prototype ready by end of Sept 1996