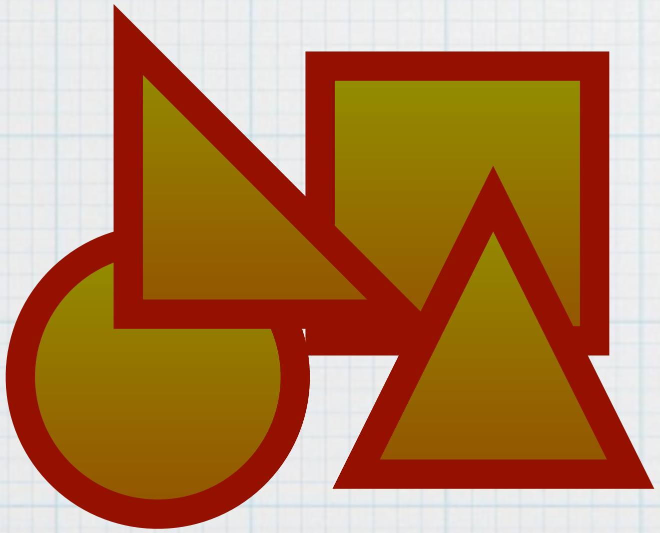
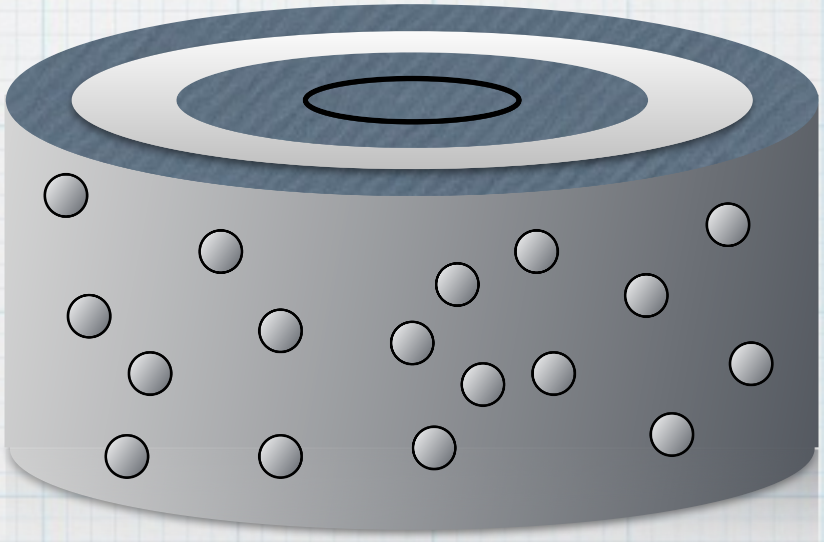
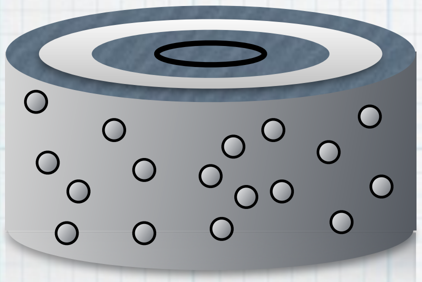
CS630
Object-Oriented DBMS Fundamentals

Les Waguespack, Ph.D.

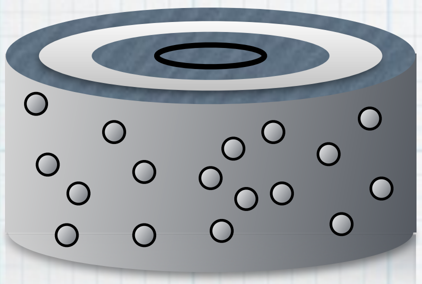






Data Base Models

- 1960's - Hierarchical
  - extended the access functionality of file management - compartmentalized security, recovery and backup
  - centralized file description and documentation
- 1970's - Network
  - standardized data model definition and access (CODASYL) - introduced query language/ report writer capabilities
- 1980's - Relational
  - formalized semantic behavior of queries (normalization) - facilitated "end user" access via standard (SQL)
  - achieved cross platform and distributed consistency
- 1990's - Object Oriented
  - seeks to recapture high-performance in complex models - seeks to bind application and data management models - seeks to exploit reusable data models
  - promises to make multi-media ubiquitous



"Model i Model"

→ Despite some lingering concerns for performance the relational model is considered the model of choice for clarity, consistency, and integrity for designing databases.

← The OODB approach promises "blazing" speed for "wired" data relationships via the "oid" (object id) pointer interconnections.

→ For the most part the relational implementations lack strong connection between the "data model" and the "transaction model"s that are applied to them (data and procedures are separate).

← The OODB depends on "wired" relationships that may impede evolution/ maintenance of data models and violates basic semantic principles of entity/relationship modeling.

Model / Problem Fit

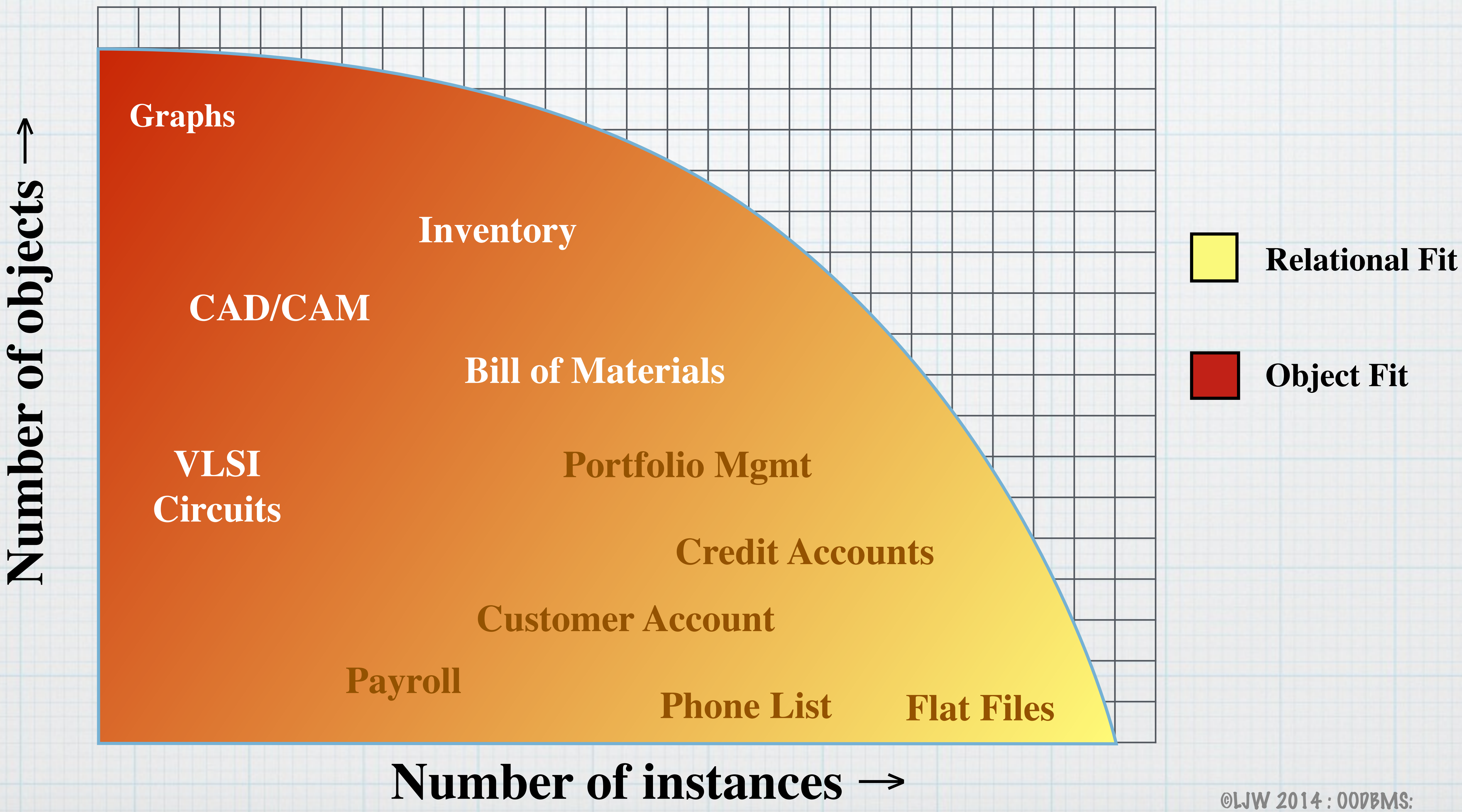
→ Relational fits data problems that are regular and homogeneous.

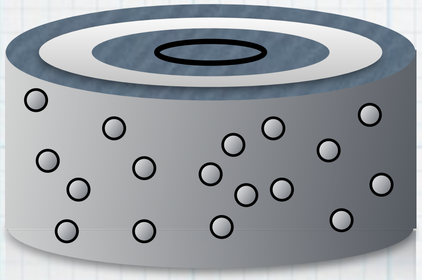
← OODB fits data problems that are irregular and sparse.

Model / Problem Fit

→ Relational fits data problems that are regular and homogeneous.

← OODB fits data problems that are irregular and sparse.





Object "Relations"

- Objects have identity as objects independent from their state (regardless of the content of their instance variables).

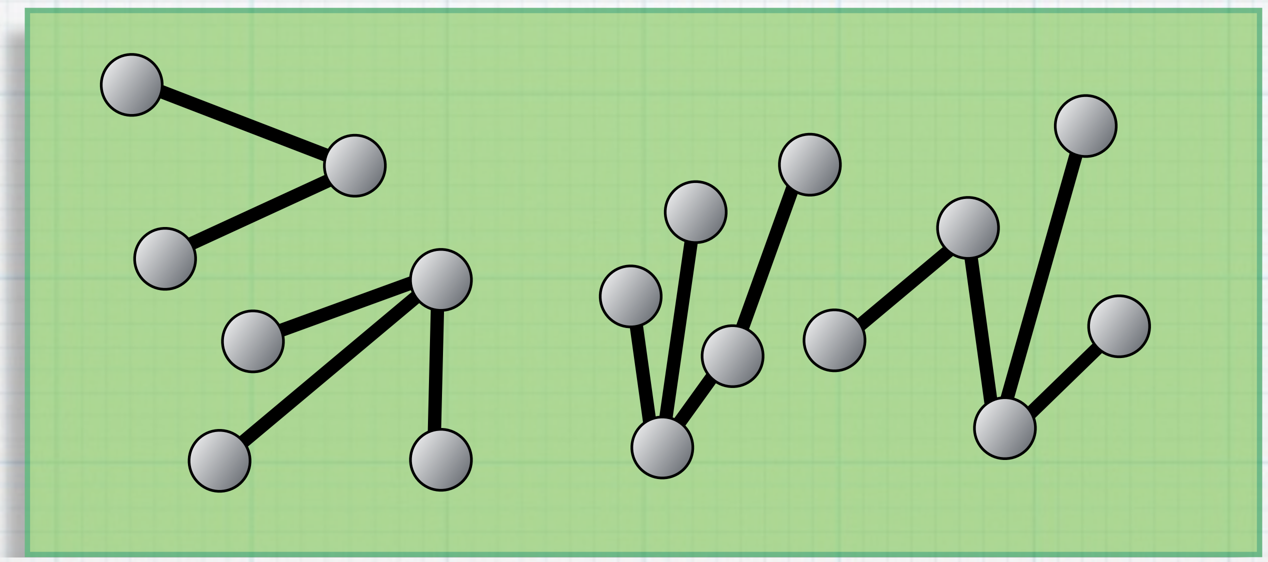
- Objects are referenced in an object system via these identities referred to as OID's or Object ID's.

- Object access is always by reference, (i.e. by following the OID to the object itself.

- Object assignment (X := Y) is accomplish by reproducing the OID rather than by reproducing the object. Therefore X.printon and Y.printon produce the same result even after the methods (X.value ← 9 and Y.value ← 13).

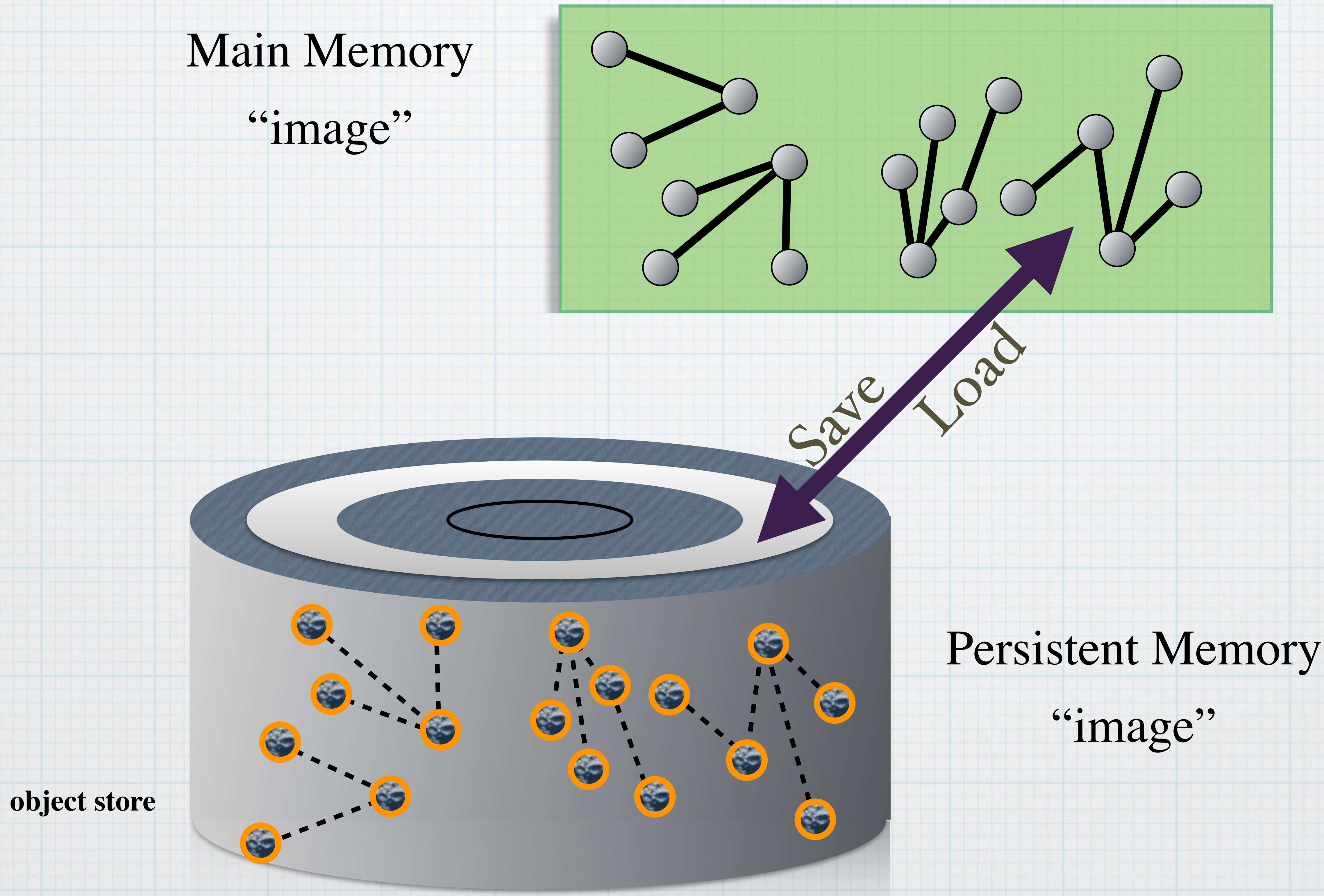
  The both printout "13".

Object "Relations"



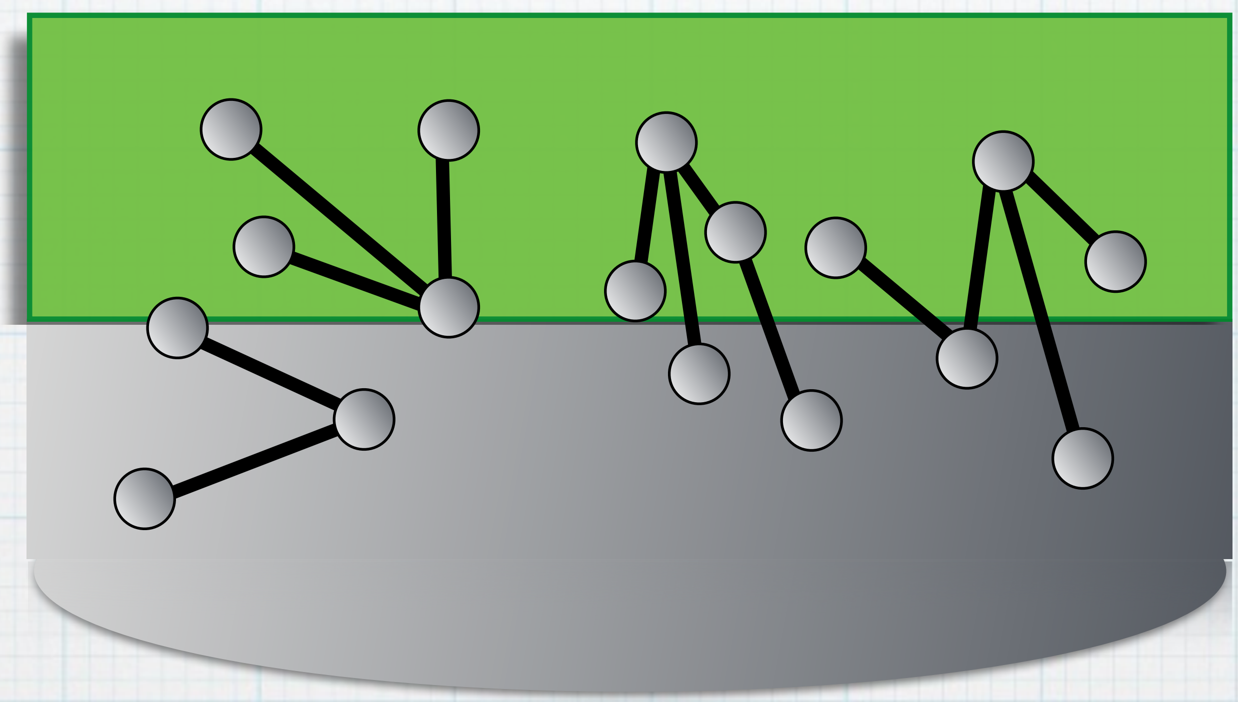
Executable Main Memory

Object "Persistence"



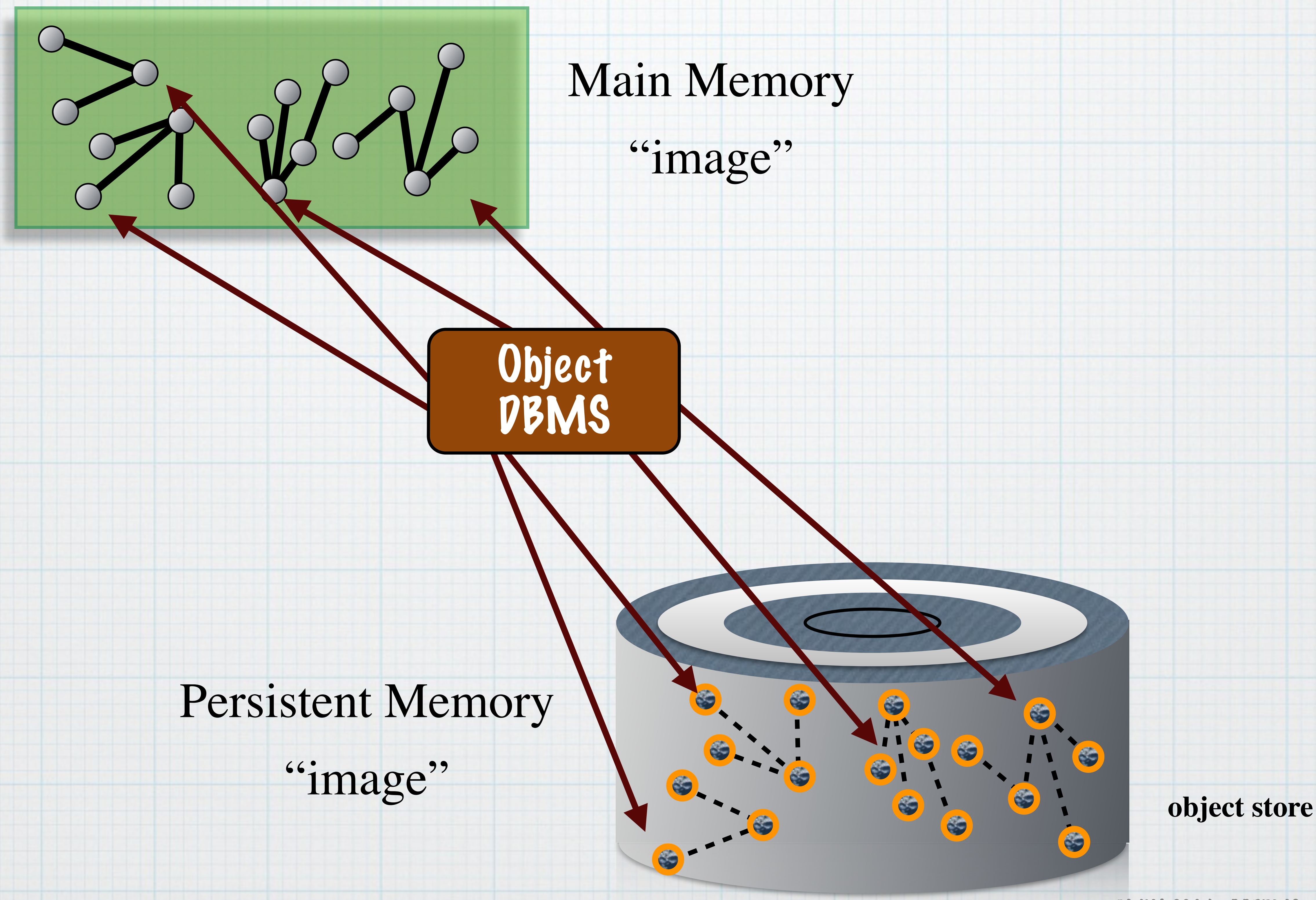
Unitary "Address Space"

Main Memory
“image”



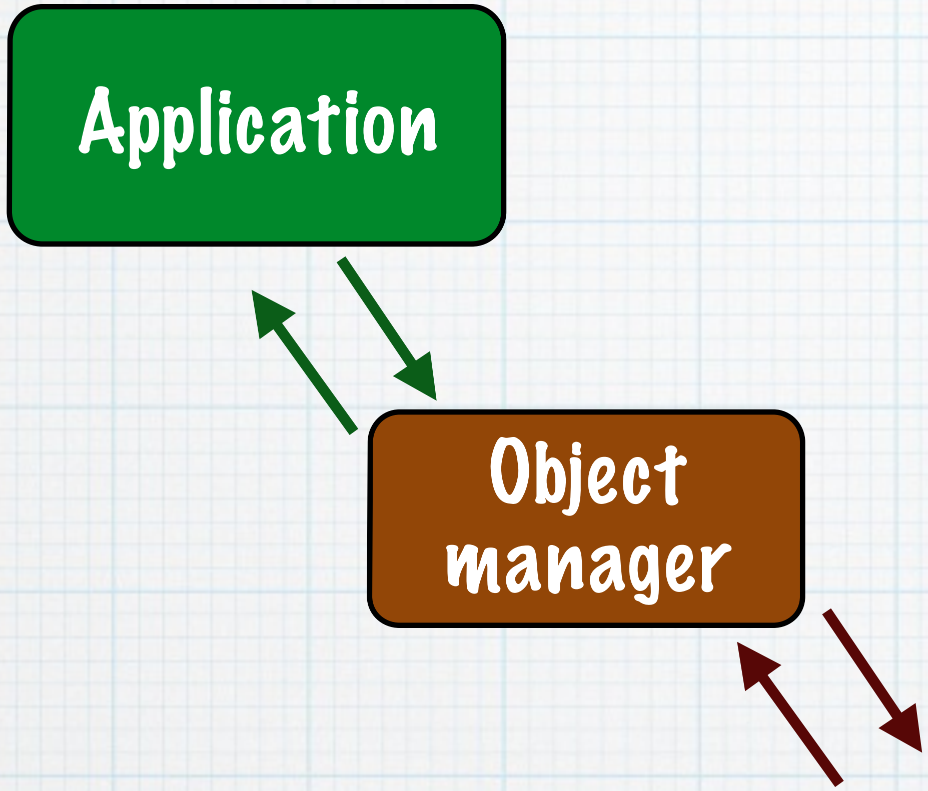
Persistent Memory
“image”

"Object DBMS"

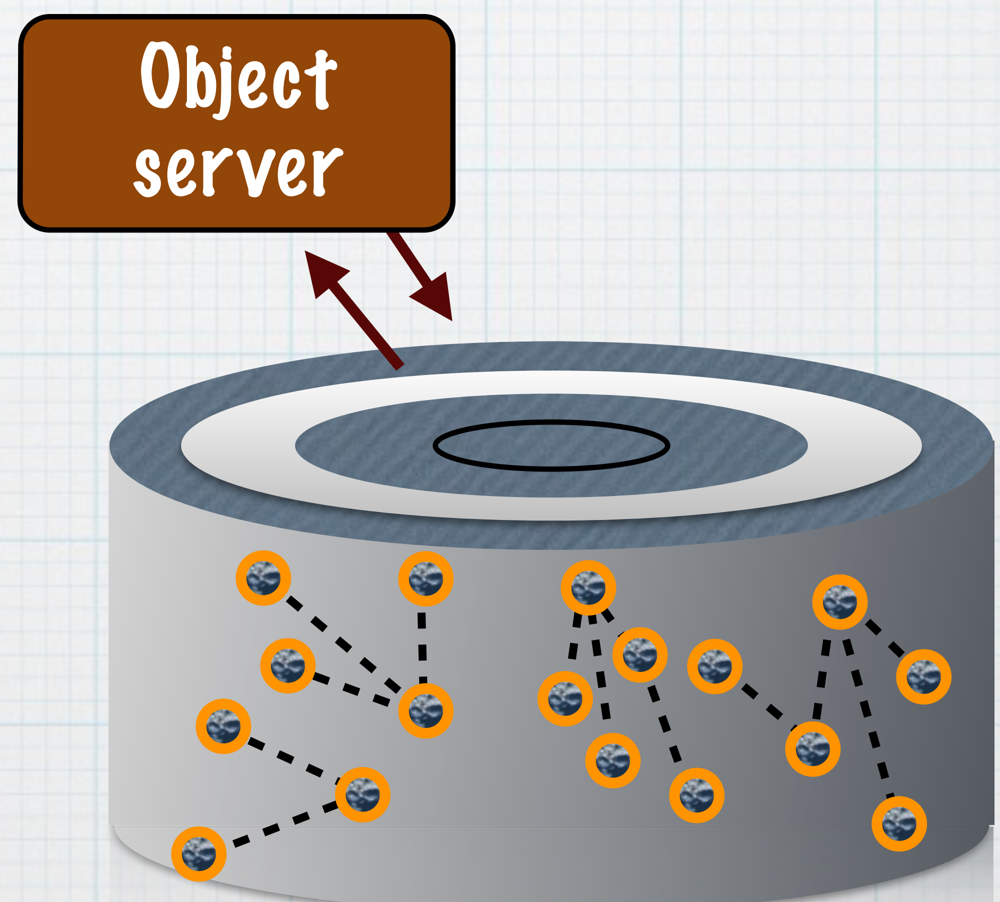


Main Memory
"image"
Object DBMS
Persistent Memory
"image"
object store

"Object DBMS"



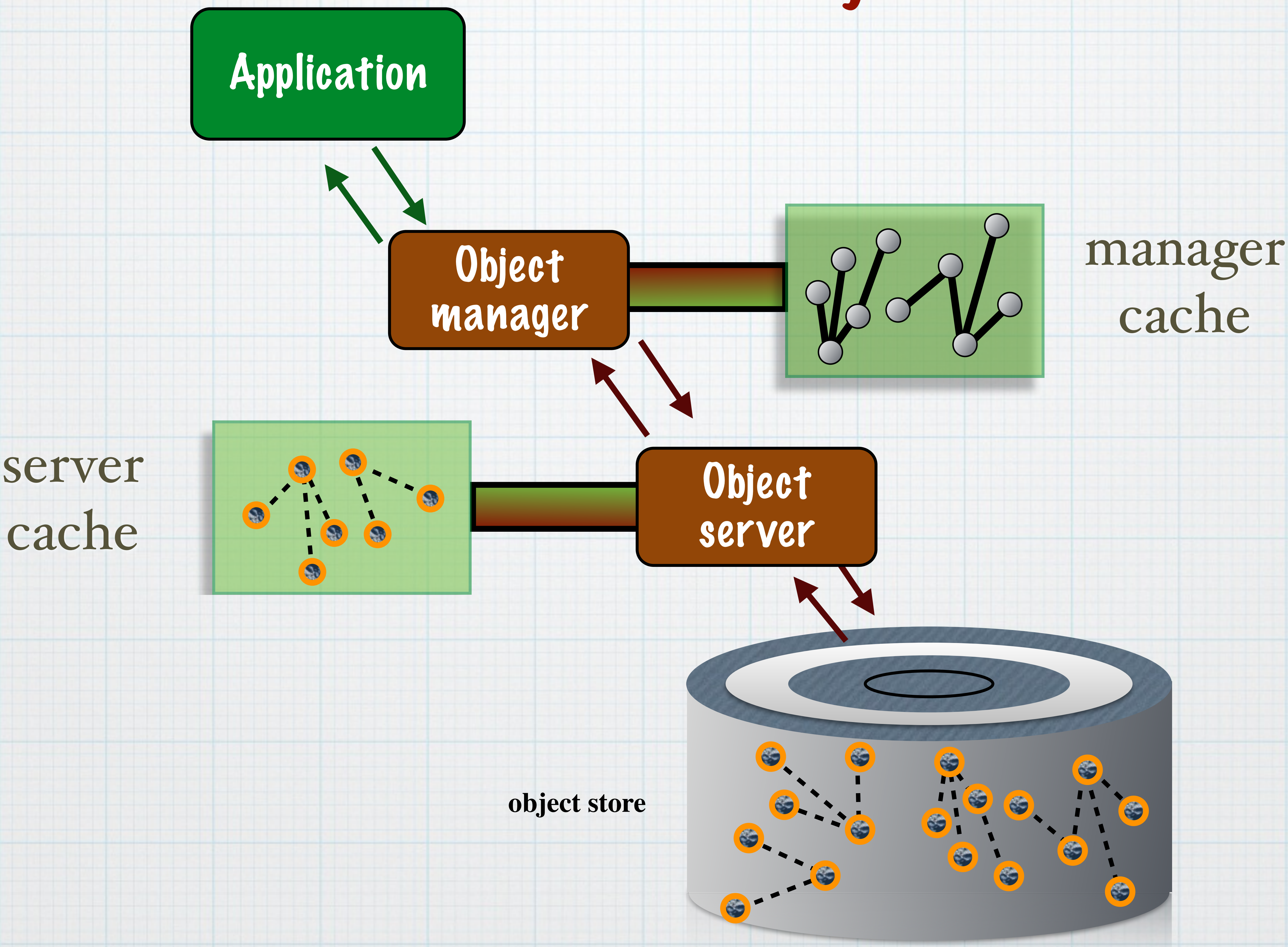
Object manager provides a local cache where transient, application bound, "images" of objects "live" during access and modification until they are committed or reverted.



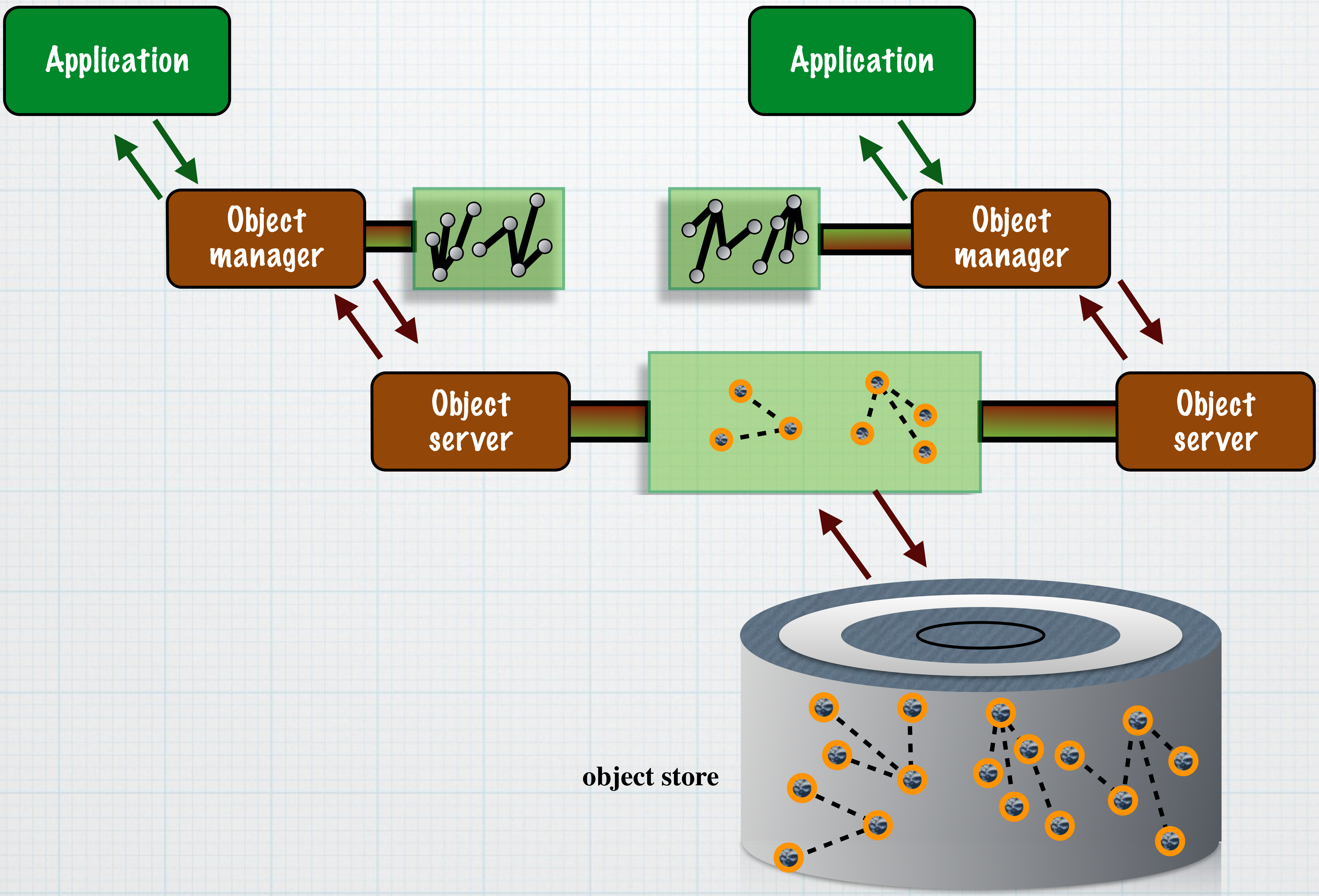
Object server is the steward of the disk image of objects and provides shared access to multiple applications, arbitrating the access through a cache of its own.

object store

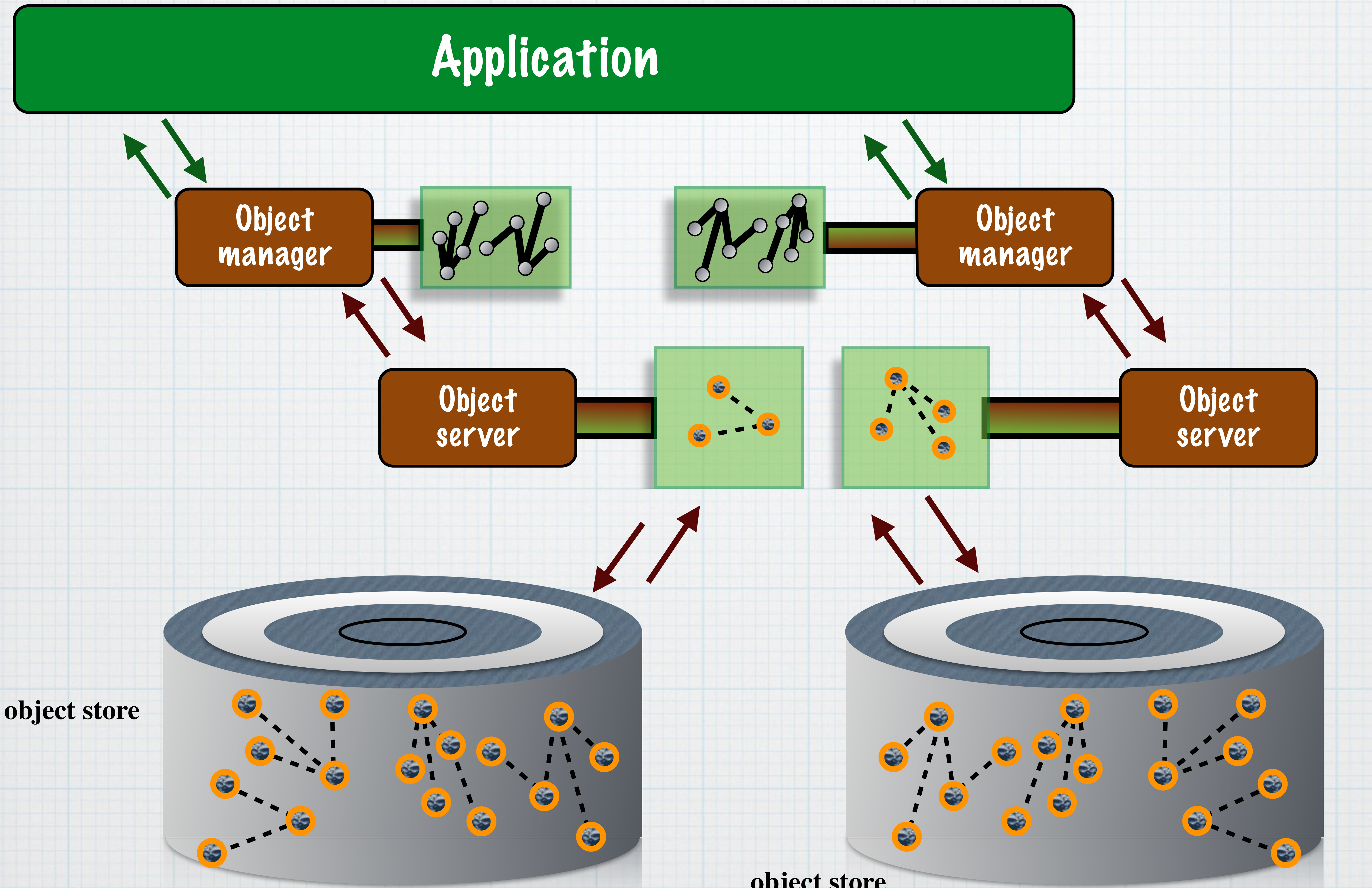
"Object DBMS"

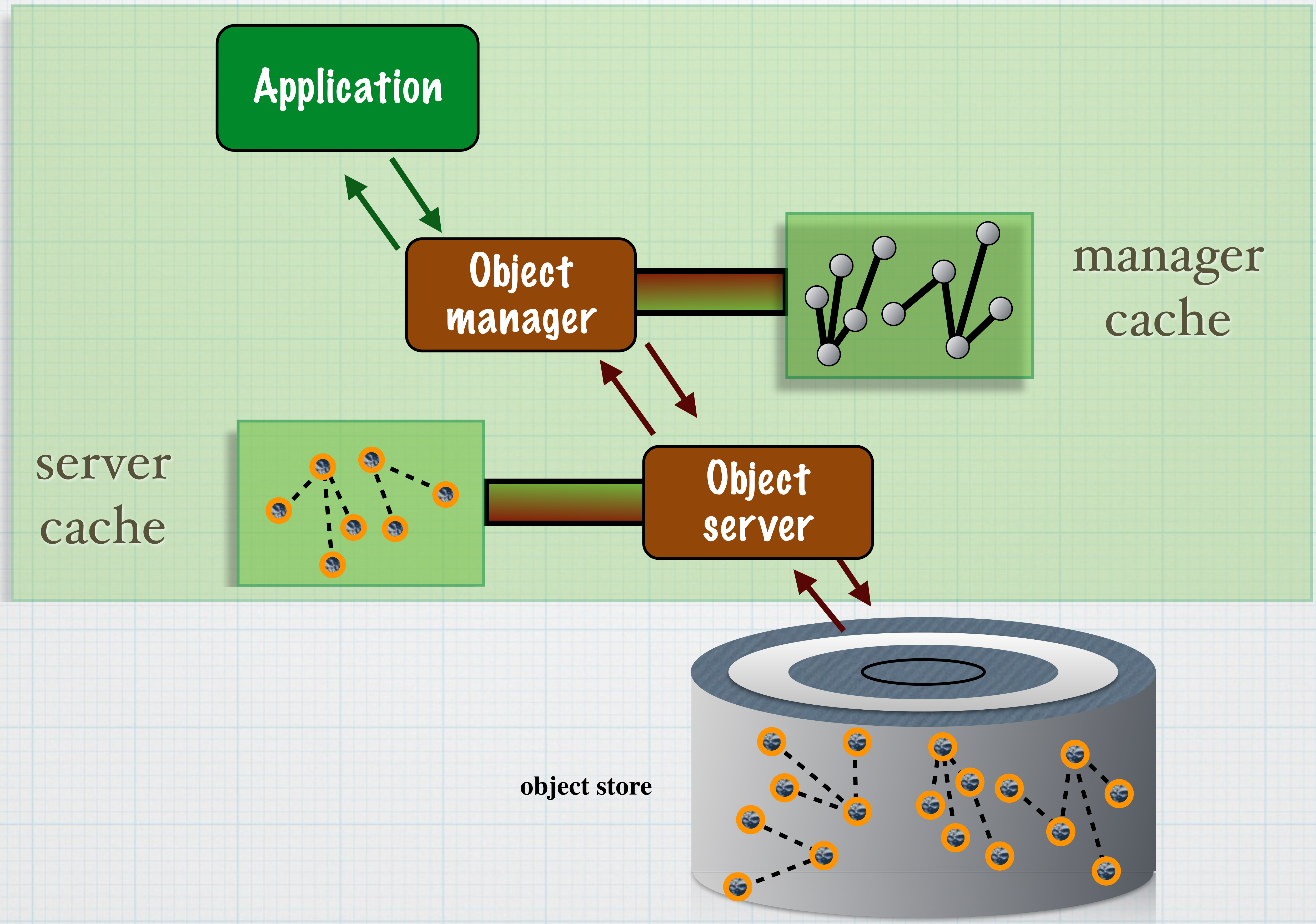

Application
Object manager
manager cache
server cache
Object server
object store

Multi-access

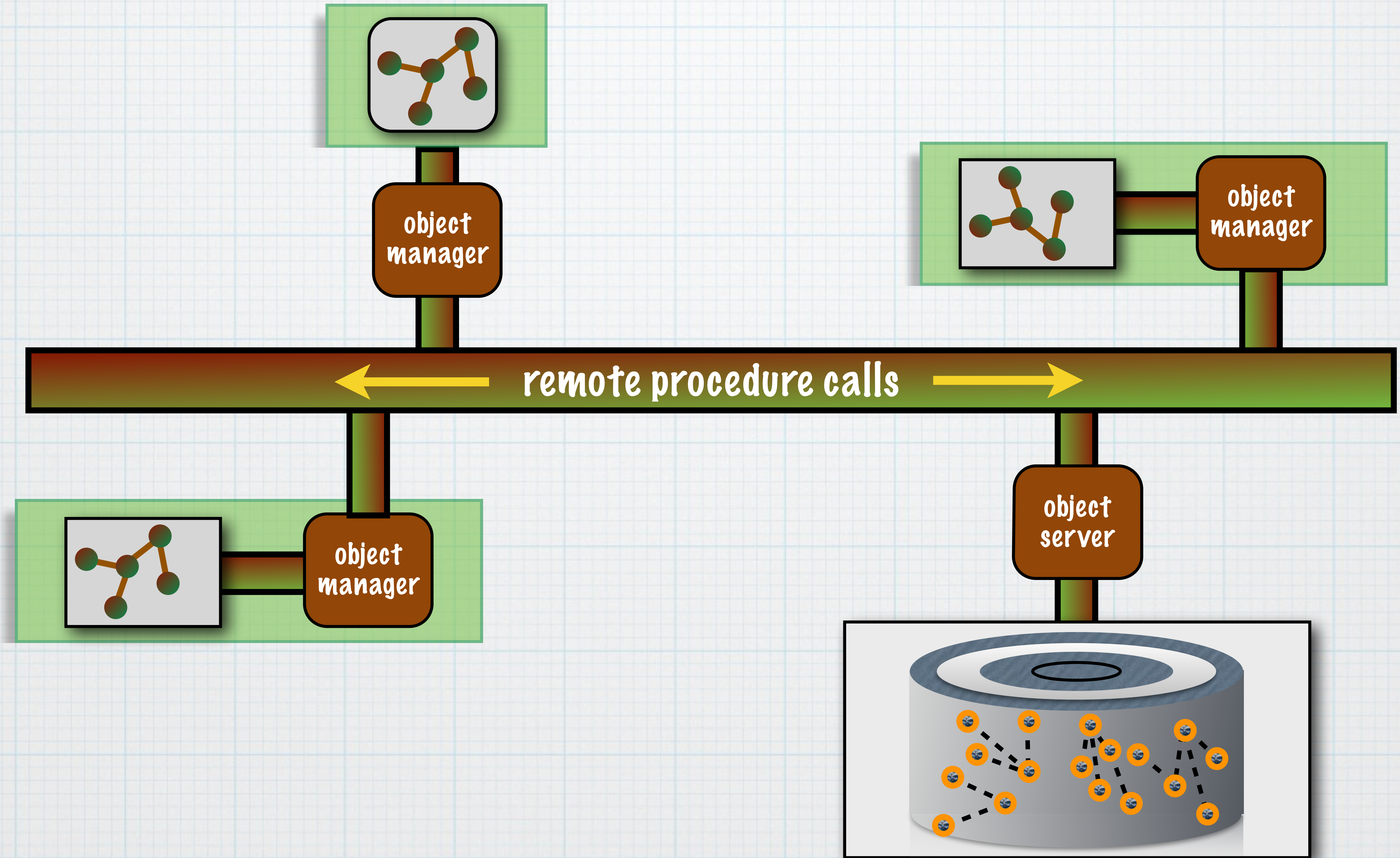


Multi-database




Application
Object manager
manager cache
server cache
Object server
object store

Distributed Components



Implementation

- Methods:
  - bound at compile time the methods may be stored external to the object store, but cannot be modified during a session
  - bound dynamically, changes in methods are immediately in "effect" for subsequent messages
  - stored external to the database allows "de- synchronization" and integrity breaches
  - stored in the object store configuration security is greater and more likely to survive a rollback and recovery cycle

- Transactions:
  - object "locking" evokes all the standard concurrency control problems found in any shared access data management circumstance.
  - object "clustering" may be used to control "lock proliferation" and improve locality of reference, access patterns
  - classic "object application" transactions are very long requiring large amounts of "cache"
  - locks at checkout vs. locks at check in!

Implementation

- Versioning:
  - CAD/CAM-like applications require the ability to maintain versions in linear and tree-structured ancestry - this is common in work group situations where several technicians may be addressing the same "system" in "part" increments.

- Query Support:
  - there is Object SQL on the market with some effort to address a standard
  - in most instances query support is really "report" support since access paths must be established at DDL- time in order for appropriate OID paths to exist to allow collections or families of objects to be scanned in a query processing mode. This evokes memories of hierarchical and inverted data model issues with "design-time" rather than "decision-time" query definition. Many different access acceleration aids are available (hashed, B-tree, "object rape" approaches).

Implementation

- Persistent Object Definition:
  - dynamic binding almost requires that persistent objects and "application" objects reside in the same address space (at least part of the time); this leads to a unified class library approach where persistent objects are simply object derived from a "persistent object ancestor."

- To "connect" persistent and dynamic object behavior requires some form of "interpretive" execution which is not native to compile-time based object systems (C++). This conflict leads to less than "pure" object oriented implementations of the object database interface.