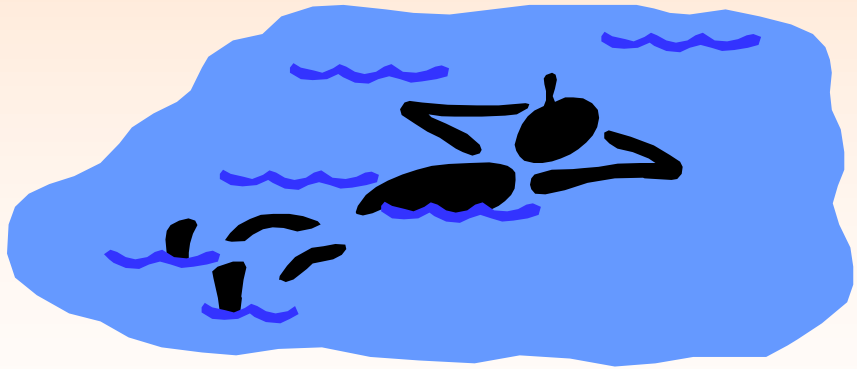
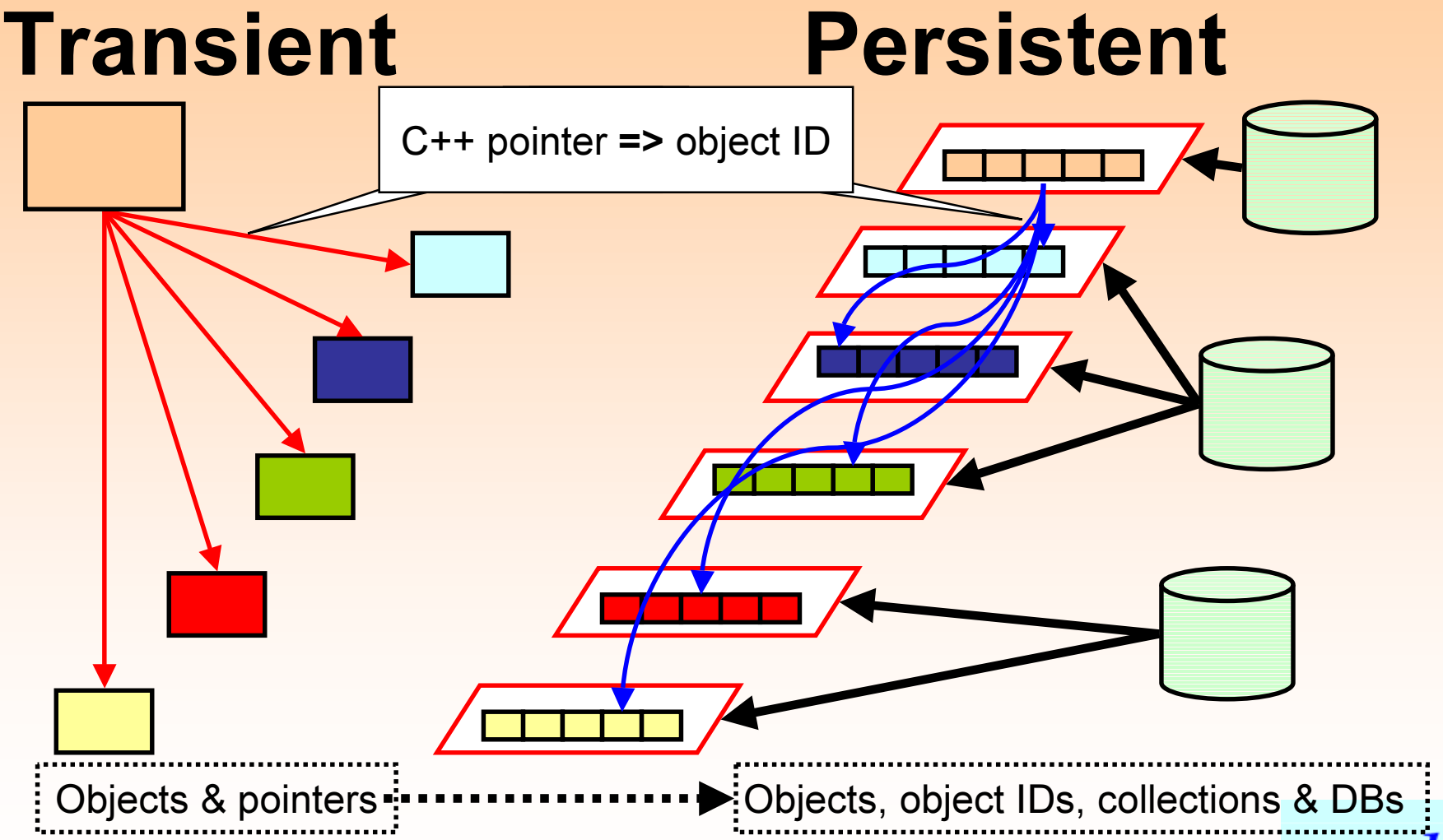
POOL persistency: Status

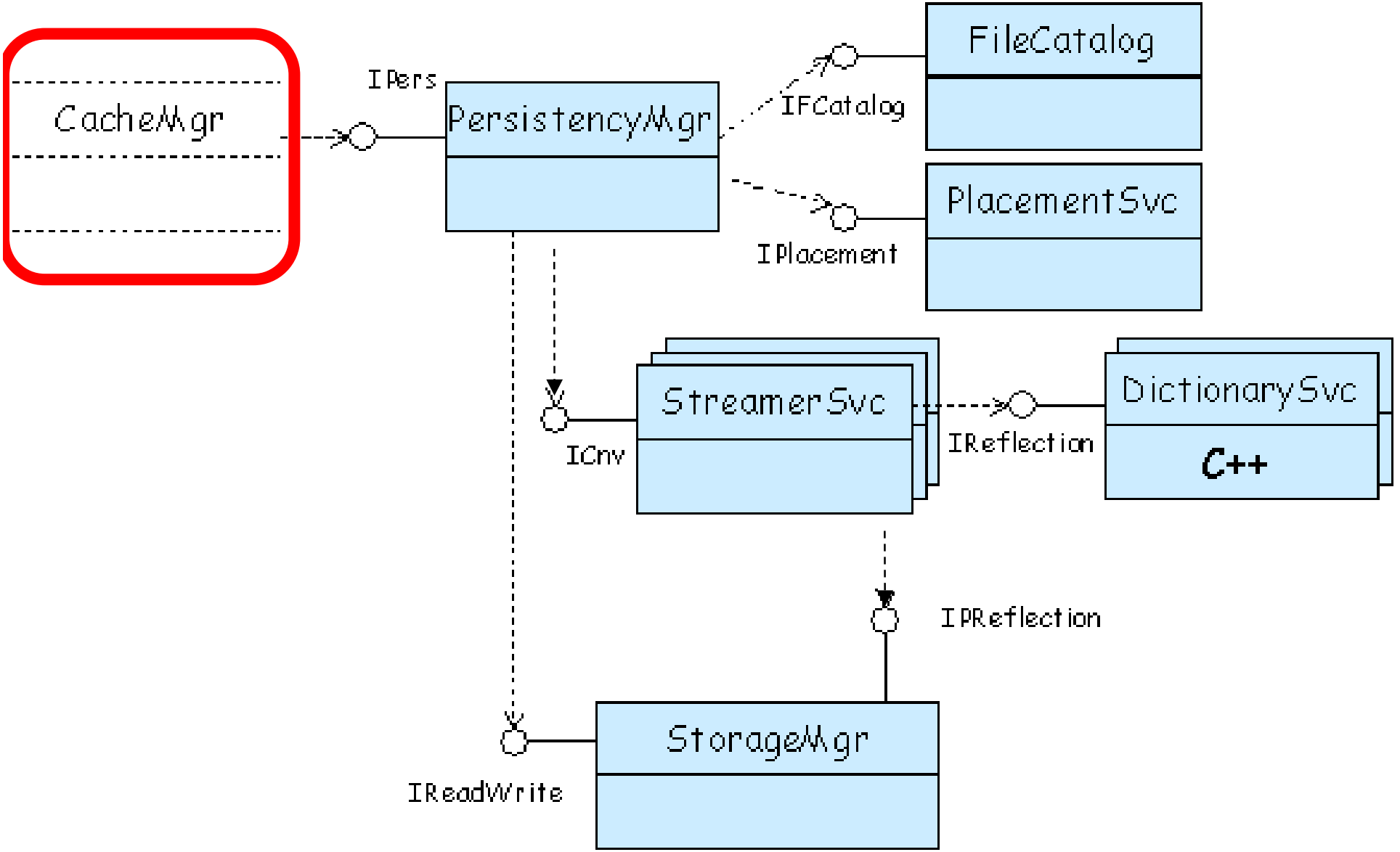
- Current understanding
- Today's model
- Conclusions



Persistency – What is it about?

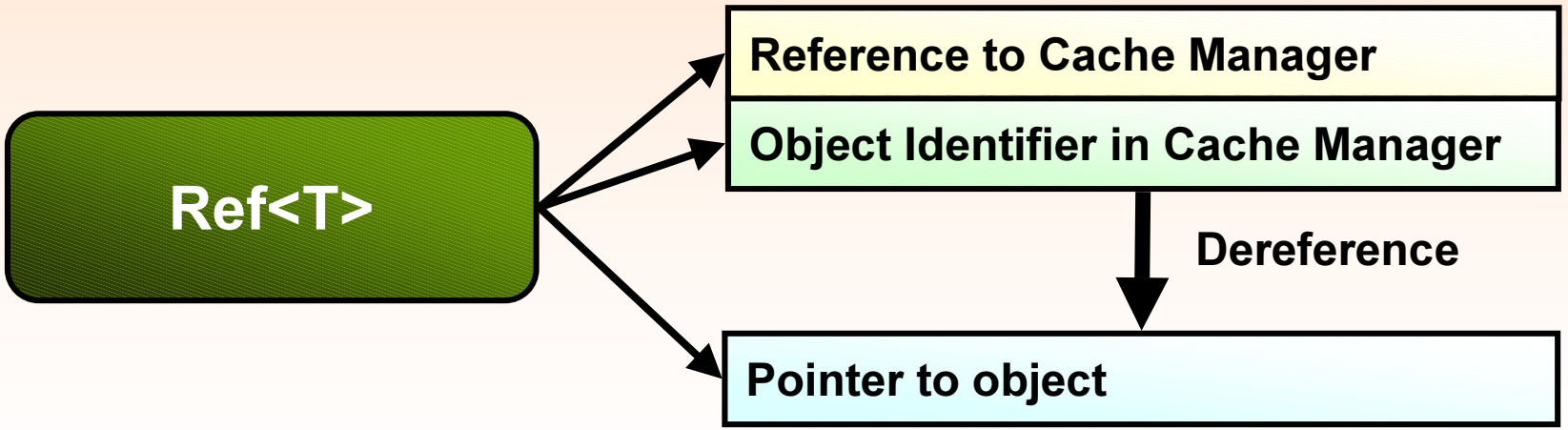


The Object Model (June)

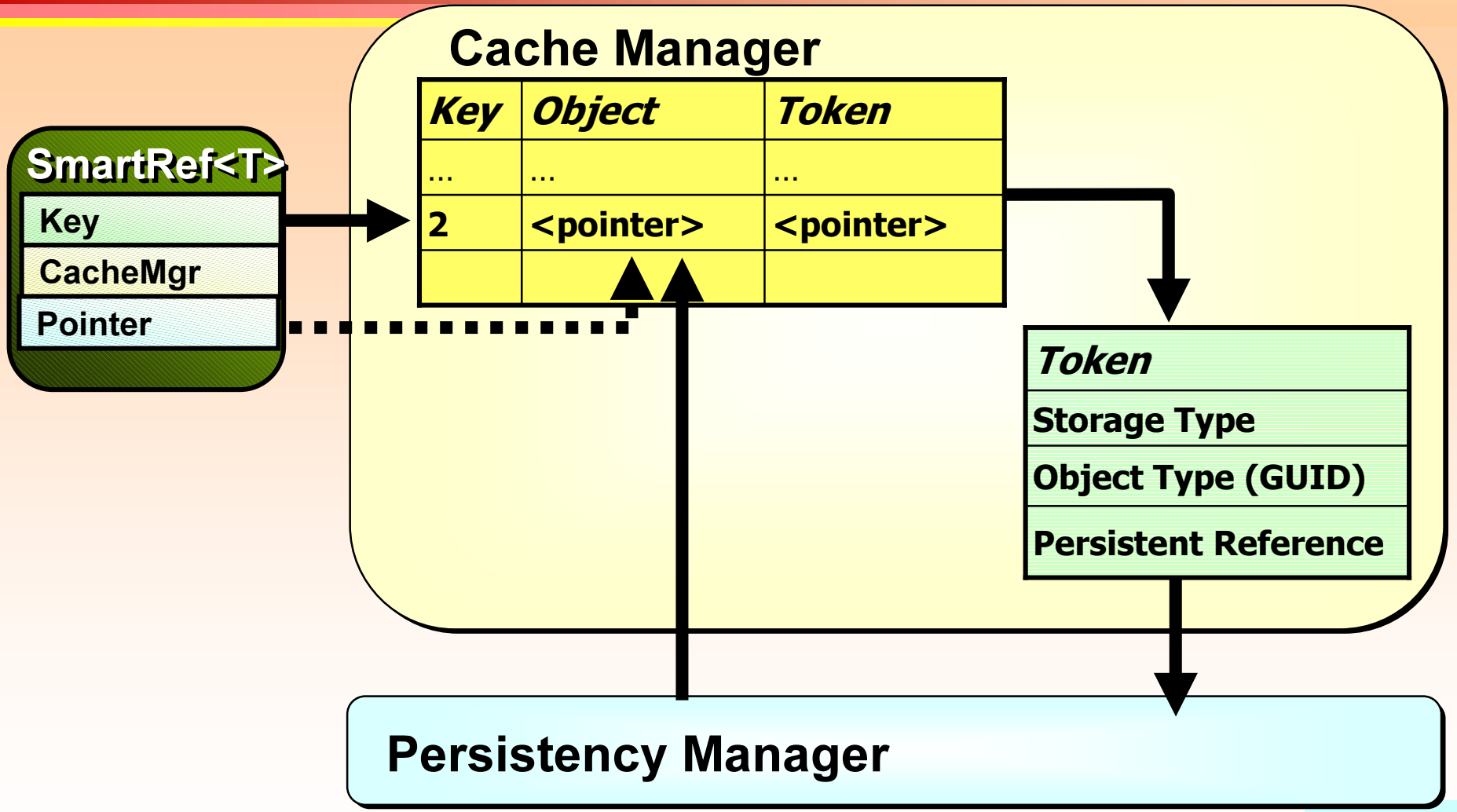


Cache Access Through References

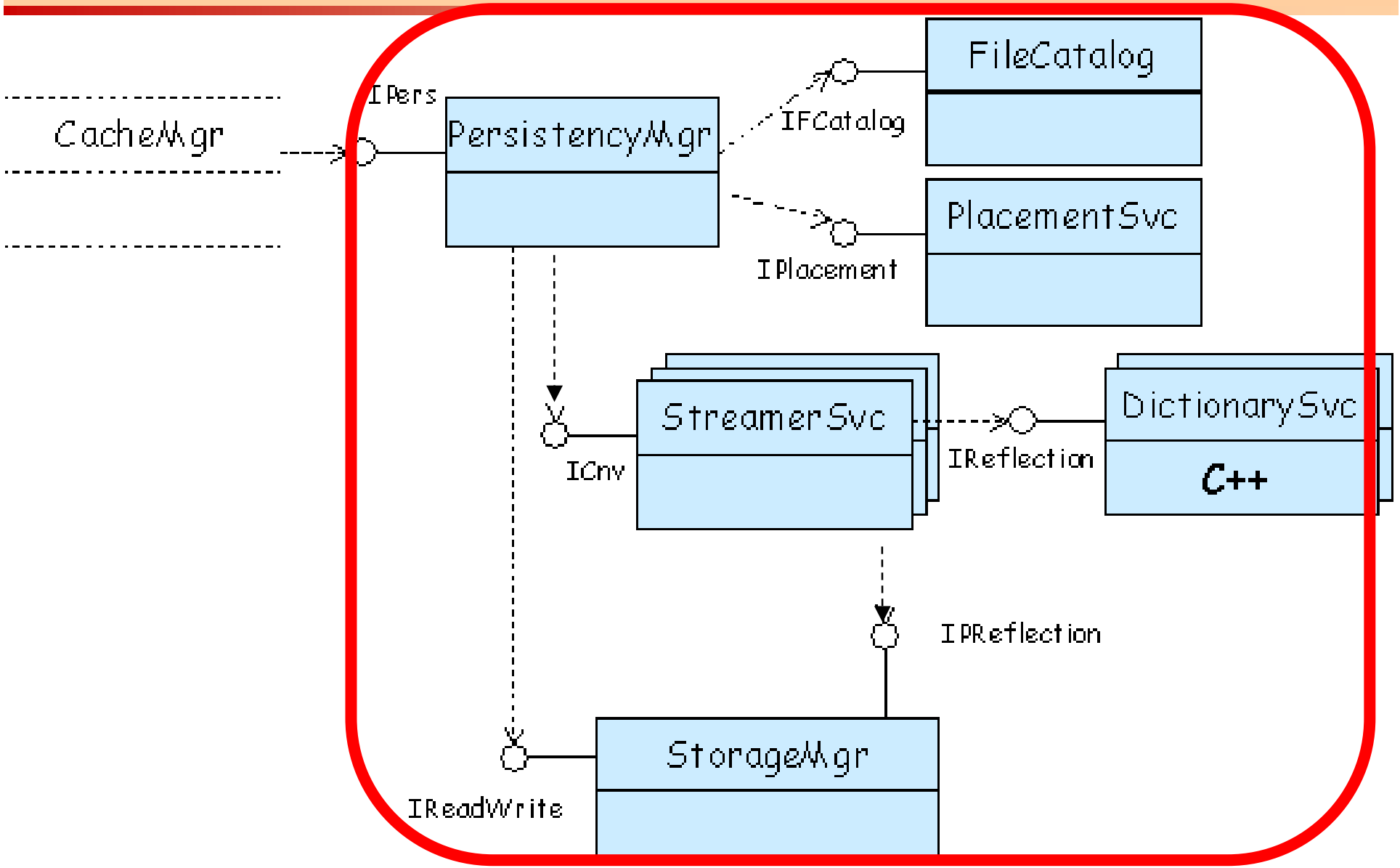
- **References know about the Cache Manager**
- **References are implemented as smart pointers**
  - Use cache manager for “load-on-demand”
  - Use the object key of the cache manager



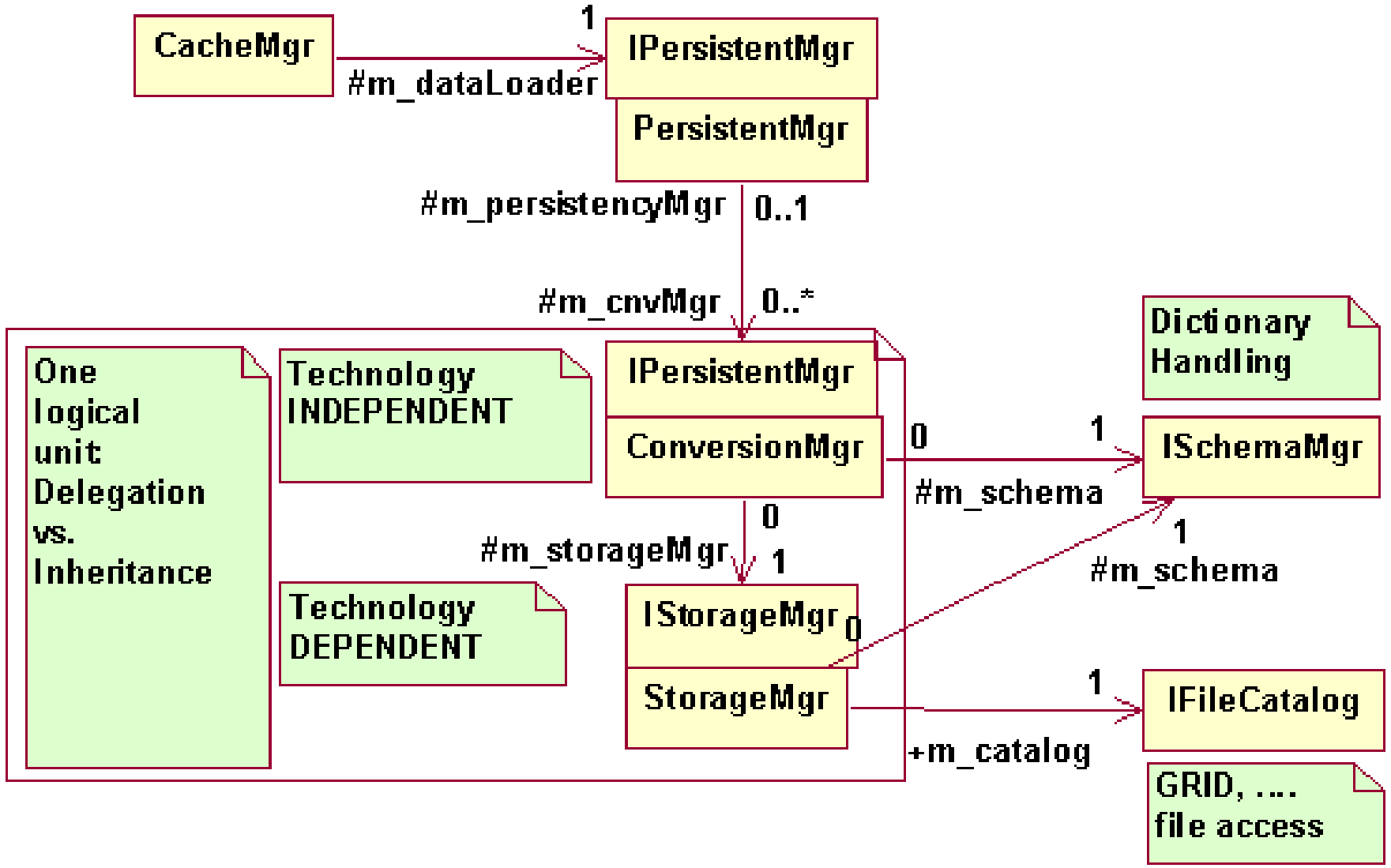
Cache Access



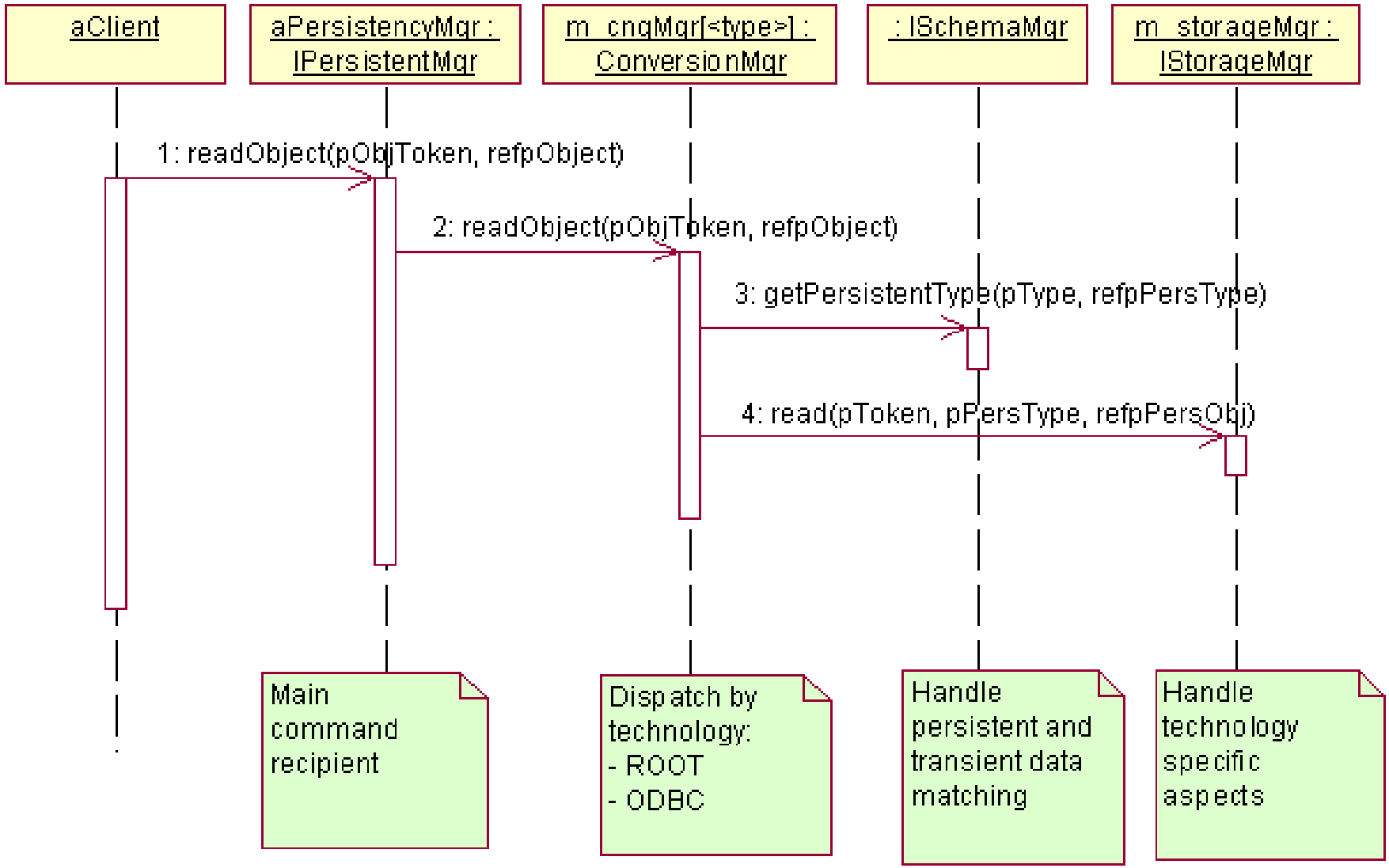
The Object Model (June)



Class Diagram of Today



Typical Calling Sequence



Persistency Interface

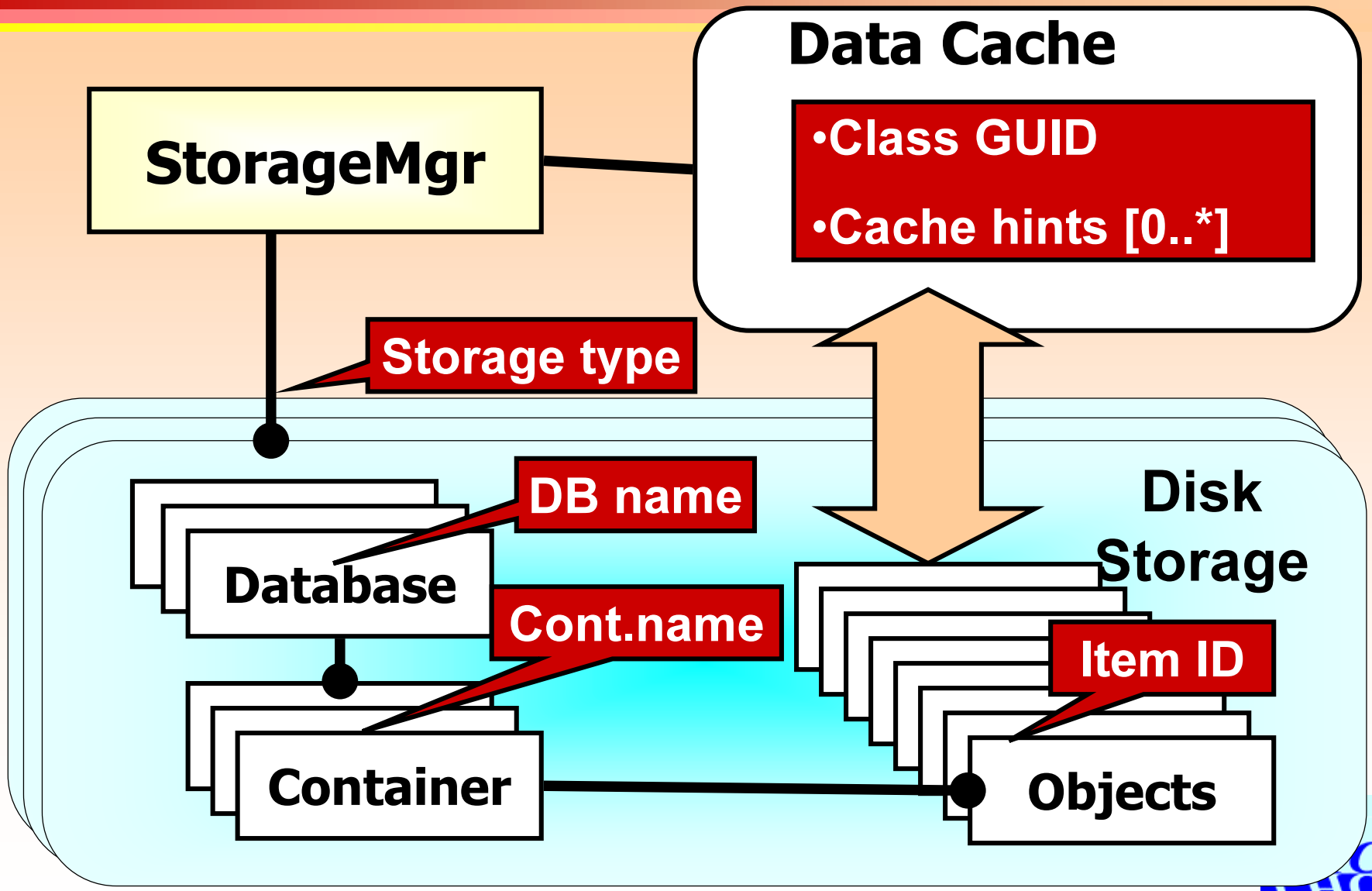
- Read Object access
  - readObject
- Transaction handling
  - Start, commit, rollback
- Register objects for writing

Writing Objects

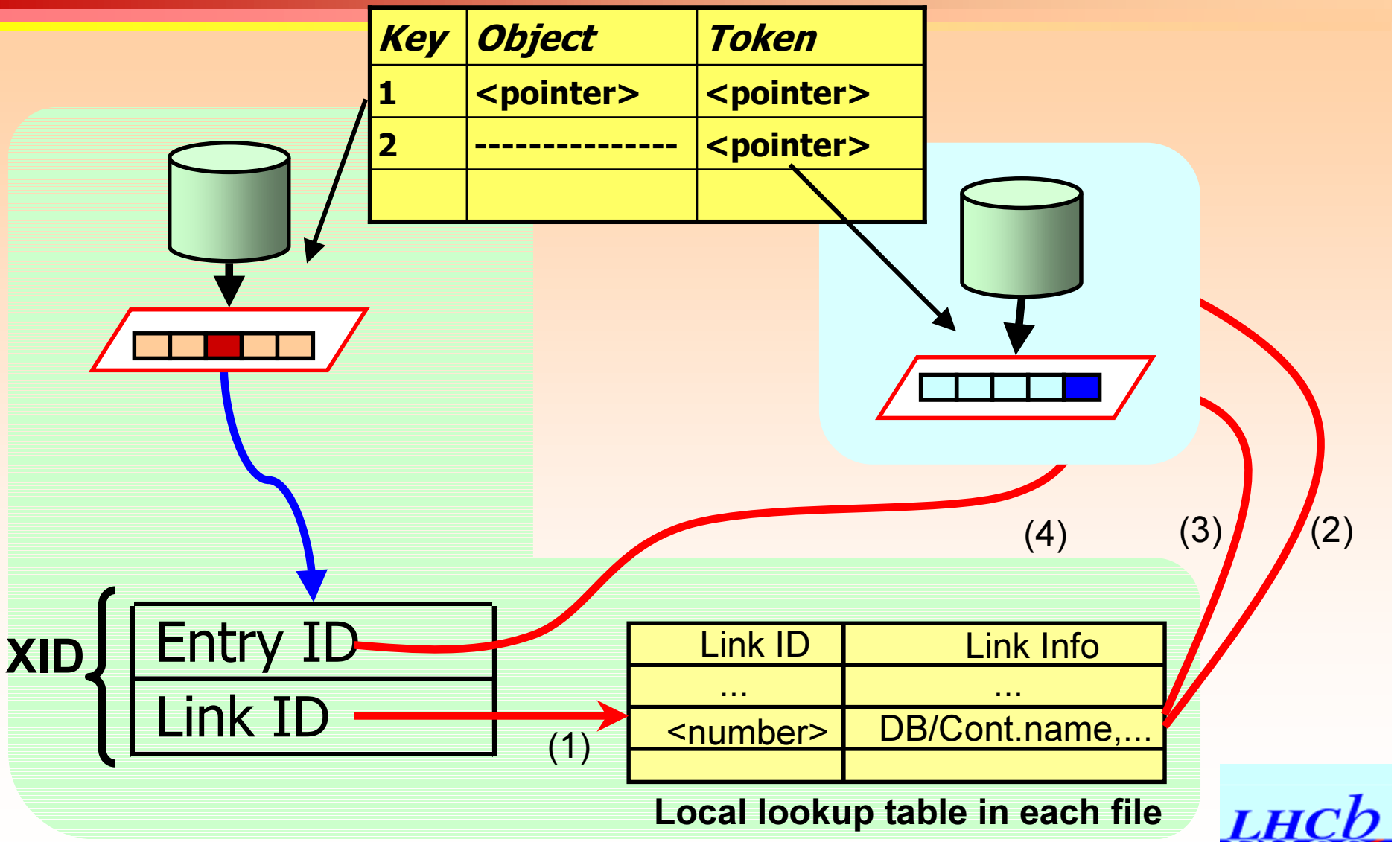
- **Supported by IPersistencyMgr**

- Start transaction
- Loop 1 … n
  - Mark objects for write: requires placement hint
- End Transaction
  - Commit
  - Rollback (if supported by database)

Model Assumptions



Follow Persistent References



The Link Table

- Contains all information to resurrect an object
  - Storage identifier — INT
  - Database “name” — string
  - Container name — string
  - Class identifier — GUID
  - Cache hints — string []
    - E.g. other possible transient conversions

Make sure the object model does not go wild!

Conclusions

- I think this model could work very nicely for any persistency technology based on database files, collections and objects within collections
- The main thinking is done
- Think about some optimizations, which cannot be introduced later
- Looks like it's about time to start prototyping

Persistent Objects

What is the default?

- Objectivity, OOCI:
  - Everything is persistent unless deleted before commit
- ROOT, RDBMS:
  - Nothing is persistent unless explicitly saved
  - ROOT: Tobject::Write(), Ttree::Fill() etc.
  - RDBMS: INSERT INTO <table> VALUES (…)
- …has consequences on allocation mechanism

Persistent Object Allocation

First possibility:

- During "Mark objects for write"
  - In registerObjectForWrite(...)
    - Client can know immediately if object can be written
    - Better handle for corrective actions
    - Create persistent token
  - In endTransaction(...)
    - Fill data, resolve references and update row

"unregisterObjectFromWrite" difficult

Transaction may be open for a long time

Persistent Object Allocation

Second possibility:

- During "End Transaction"
  - In markObjectForWrite(...)
    - Trivial. Simply collects object pointers
    - "unregisterObjectFromWrite" trivial
  - In endTransaction(...)
    - Create persistent token
    - Fill data, resolve references and update row

Only bulk transaction/conversion status

On failure drop all