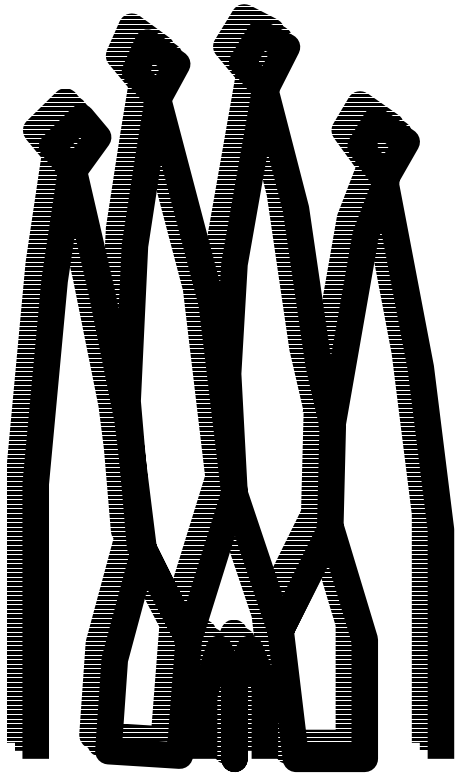
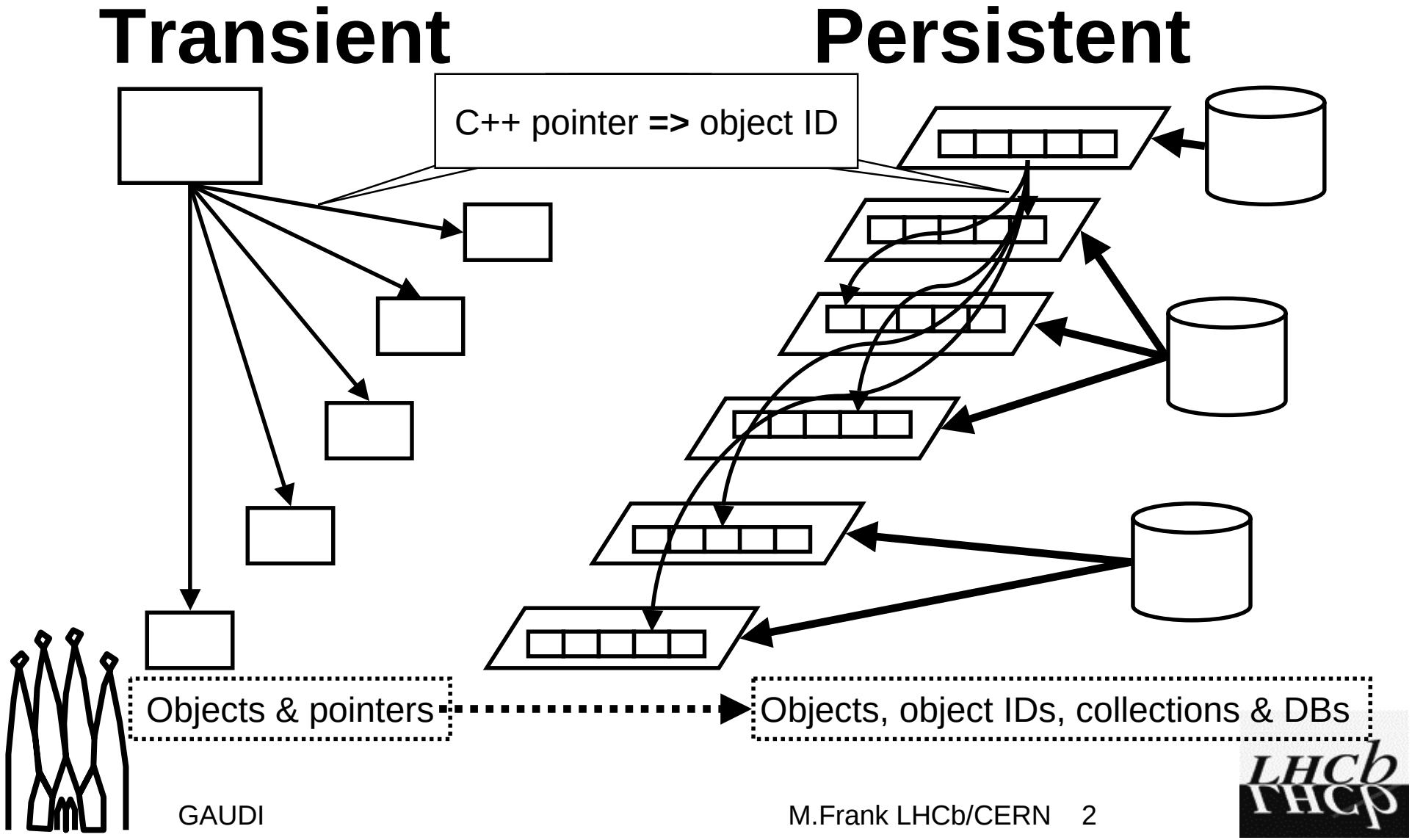
Handling Object References

- Scope
- Generic Model
- Conclusions



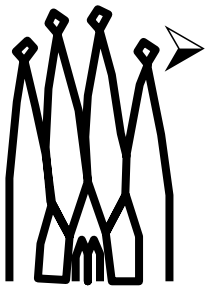
M.Frank LHCb/CERN

Persistency – What is it about?



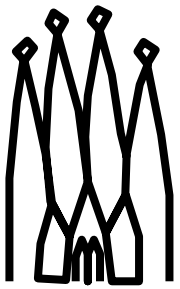
Persistency – What is it about?

- **Data**
  - Stream to file, from file, to Blob, from Blob, …
  - … trivial
- **Relationships**
  - Tricky bit
  - Persistent object “pointers” (OID)
- **Let’s see how references can be handled…**
  - Not blank theory
  - Concepts based on Gaudi implementation

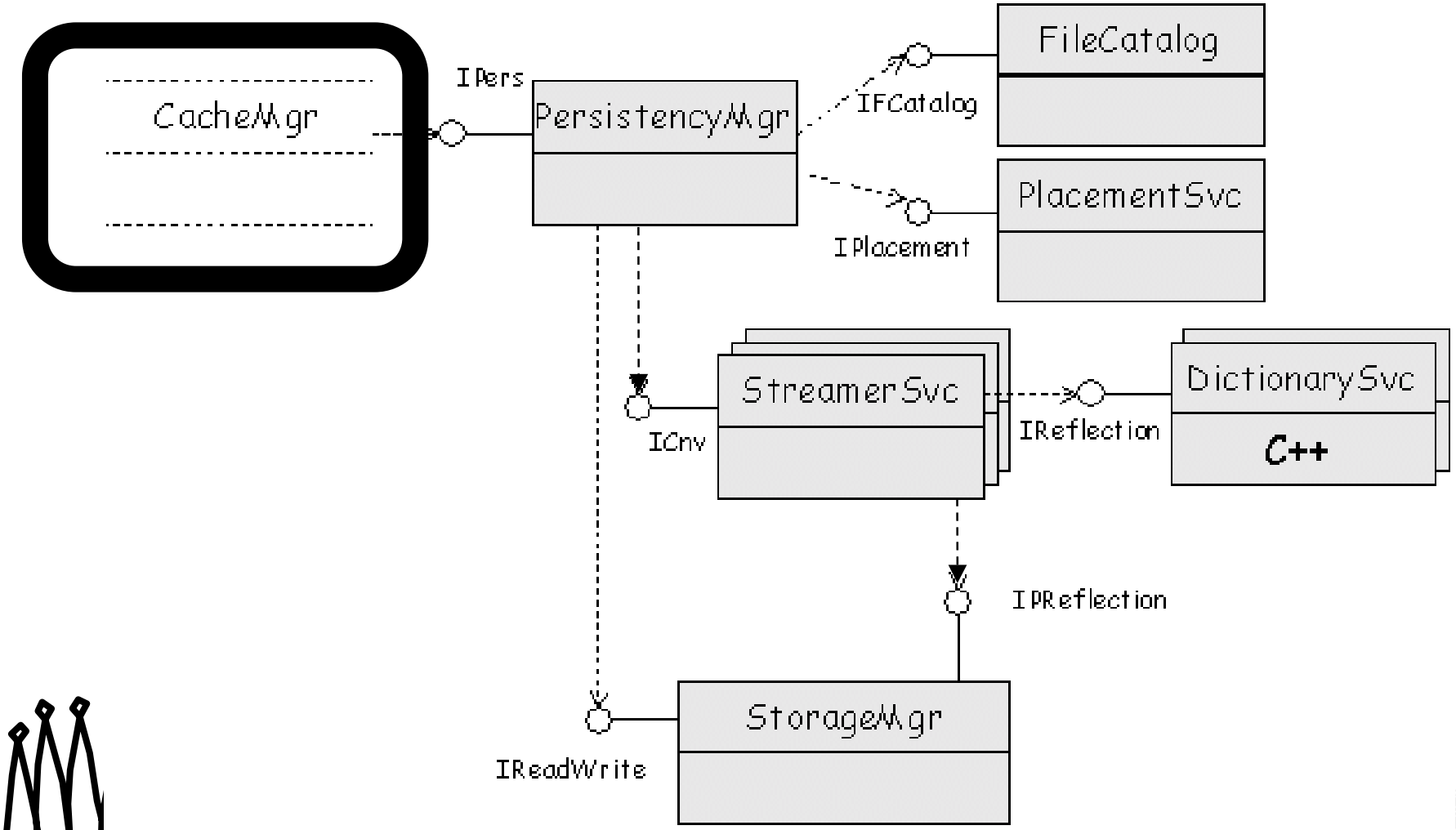


Persistent vs. Transient

- **Persistency should not influence the transient world**
  - No exposed ooObj, TObject & Co.
- **Transient references are independent of the persistent technology**
  - No access to persistent technology dependent calls
  - Use solely “cache manager” to bring objects “online”

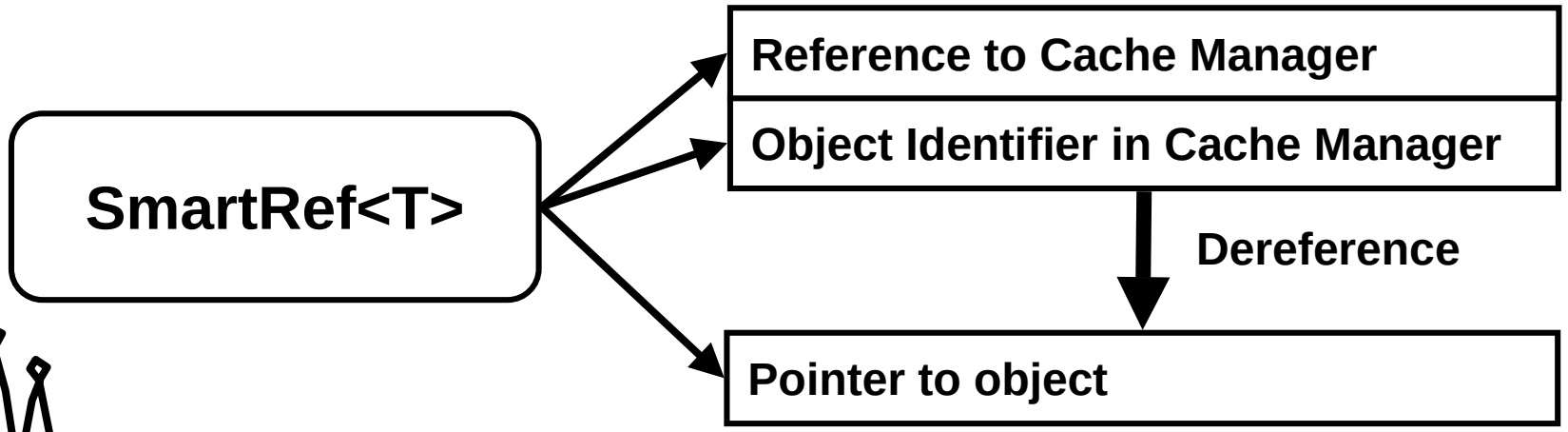


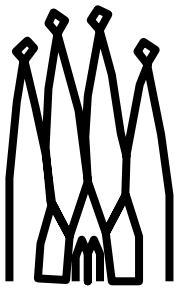
CacheMgr: Door to Persistency



Transient Object References

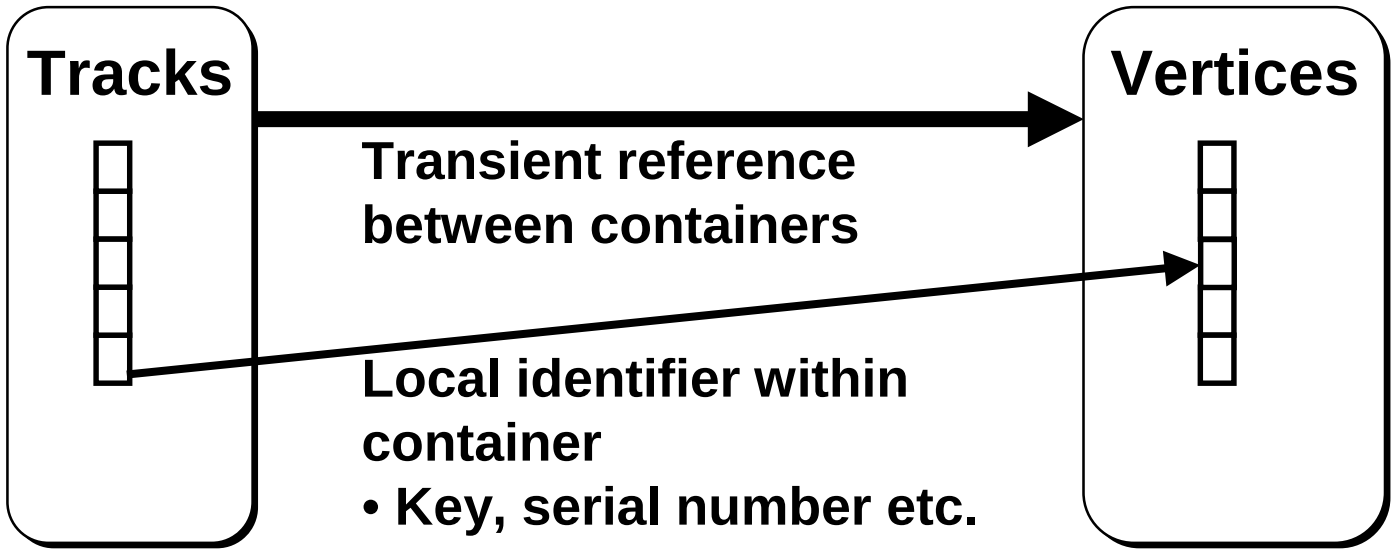
- **Objects know about the Cache Manager**
- **References are implemented as smart pointers**
  - Use cache manager for “load-on-demand”
  - Use the object identifier(s) of the cache manager
  - Can be serialized and de-serialized



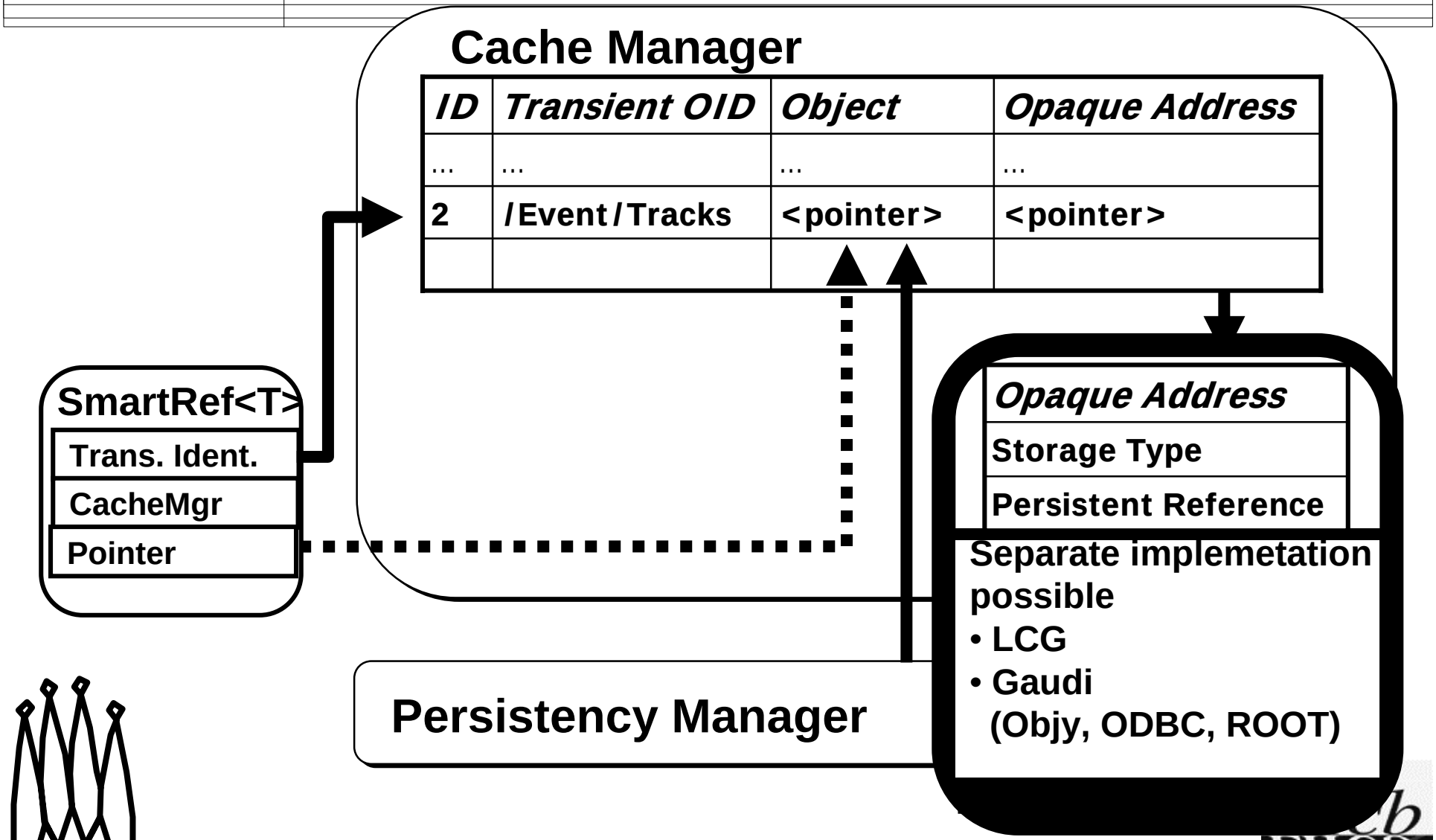


Transient Container References

- **Most objects are aggregated in containers**
  - Identify containers using object identifier of the cache manager
    - Once per container
  - Internal links between objects

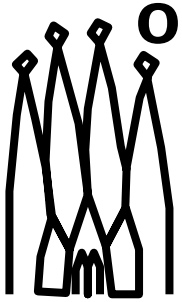


Access Transient References

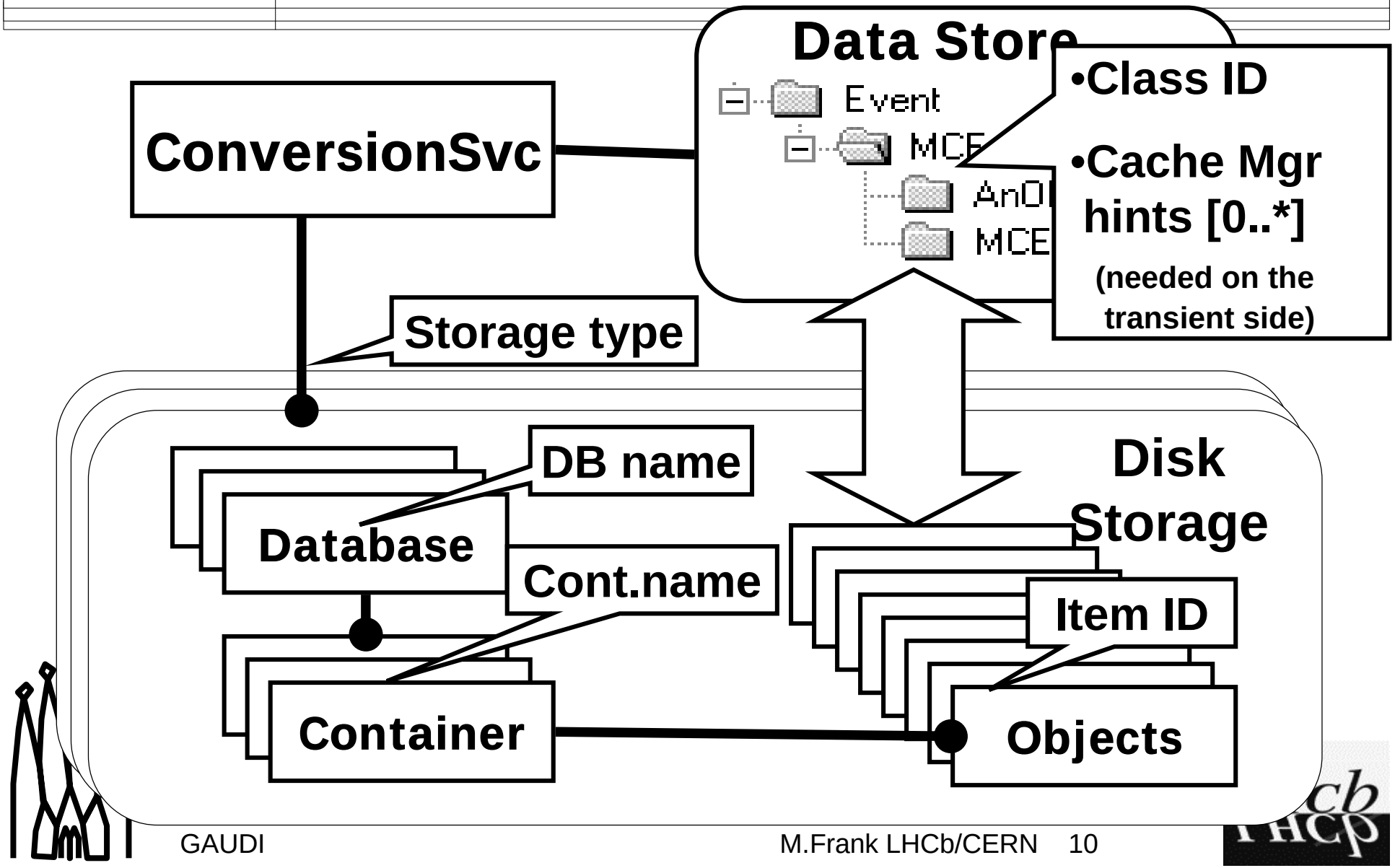


Persistent References

- **…must be *build from transient references***
- **…must allow to *create transient references***
- **…must allow to *create transient objects***
  - Geographically locate persistent objects
  - Find the recipe to create the transient representation
- **…should have a common persistent format**
  - With technology dependent interpretation
- **Roughly One to One correspondence to transient object references**



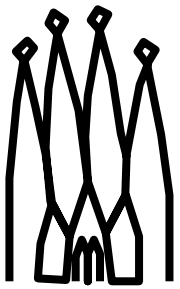
Model Assumptions



Mapping to Database Technologies

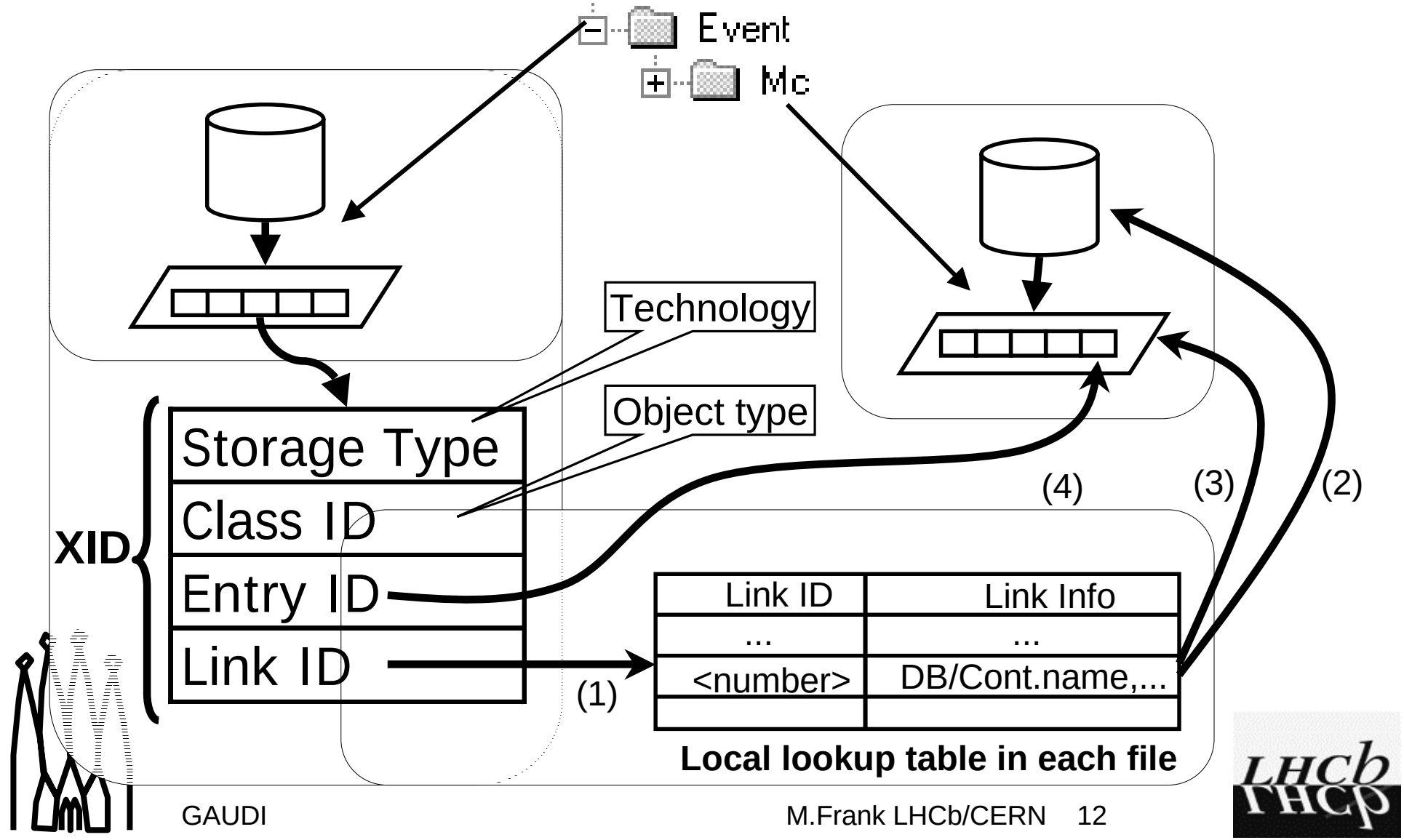
- Identify commonalties and differences
  Necessary knowledge when reading/writing

	Generic	ZEBRA	ROOT	RDBMS	Objy
Write	Database	File	File	Database	Database
	Collection	Bank	Tree/Branch	Table	Container
	Item ID	Record #	Event #	Prim.Key	
Read	Database	As for writing			OID
	Collection				
	Item ID				

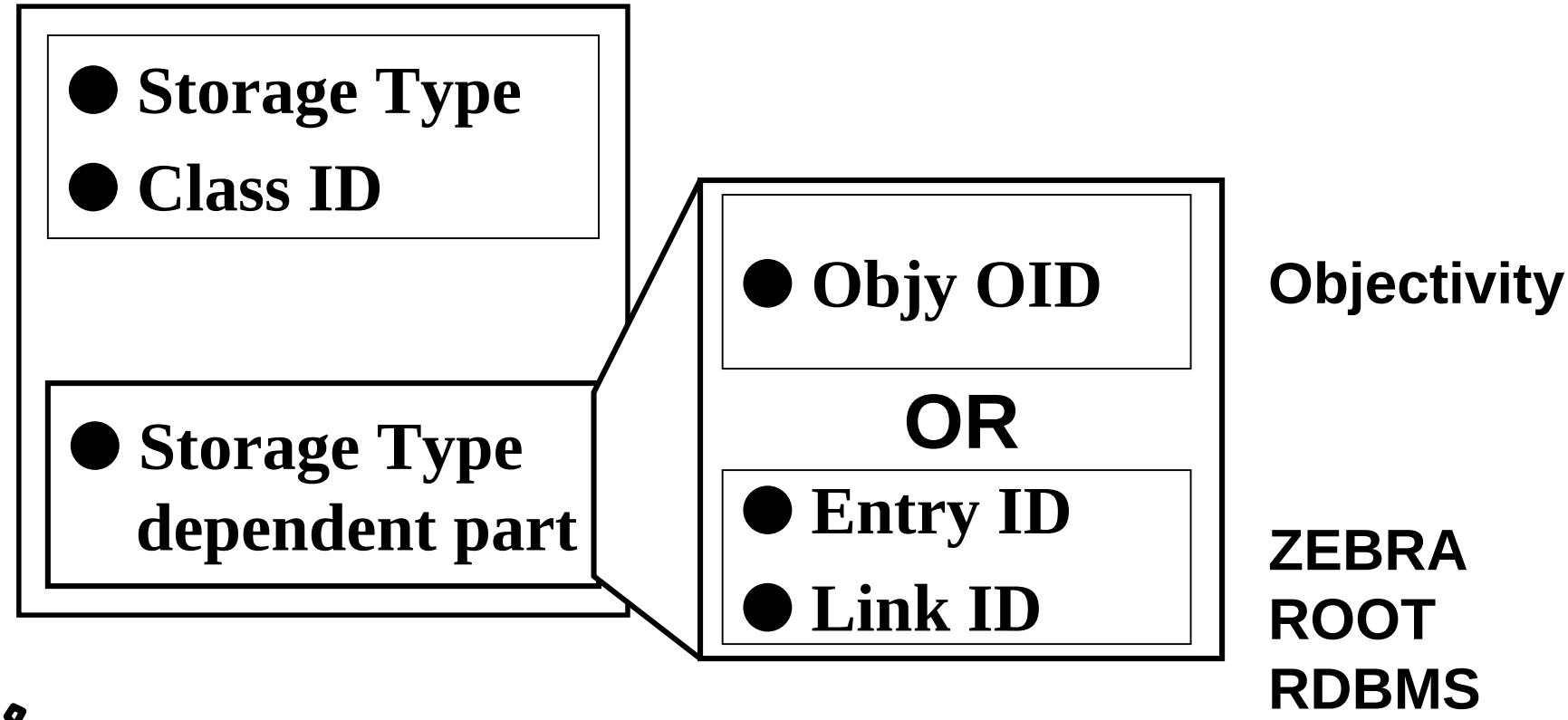


- RDBMS: More or less traditional
- Objy is different

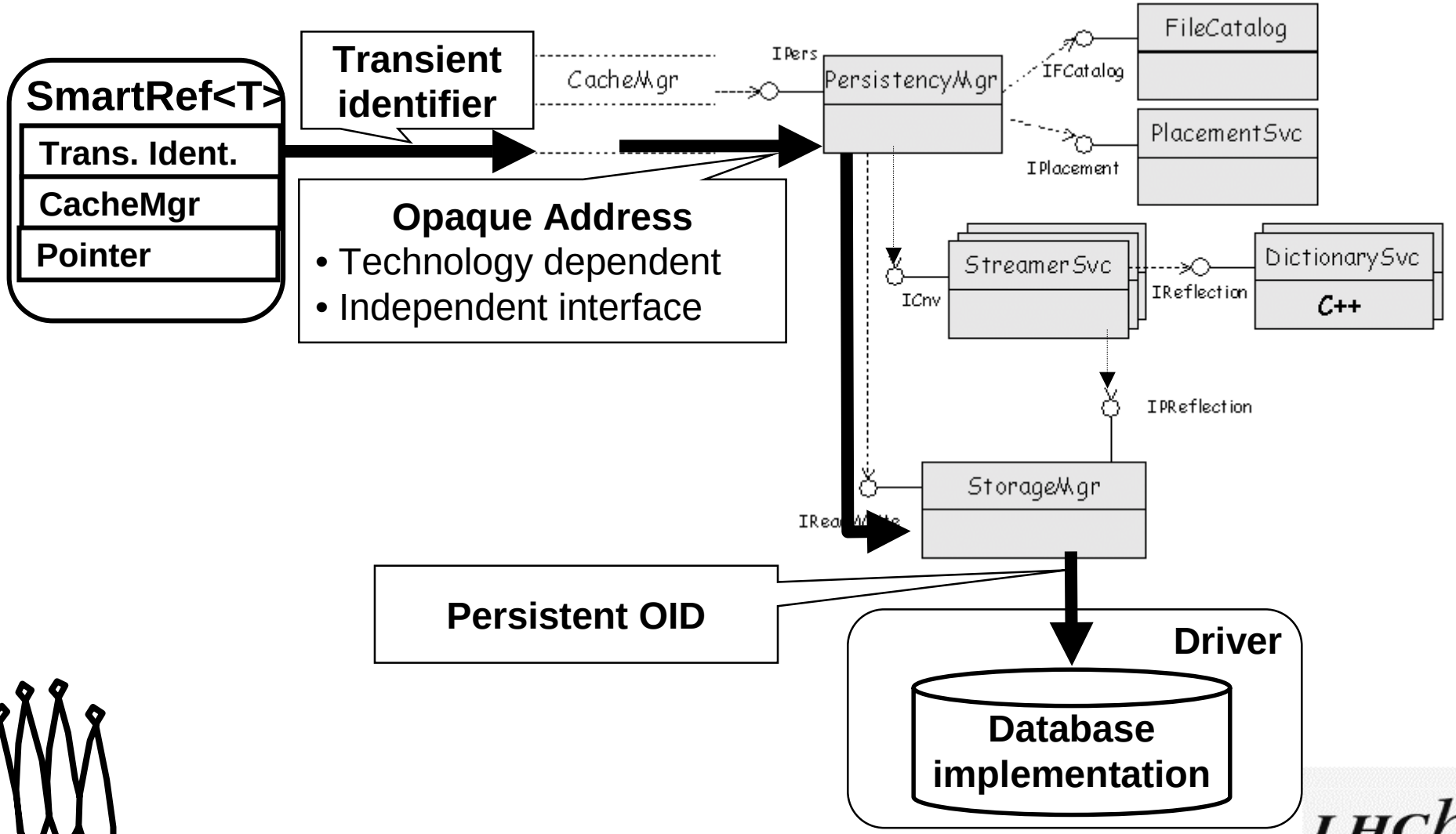
Follow Persistent References



Extended Object ID (XID)

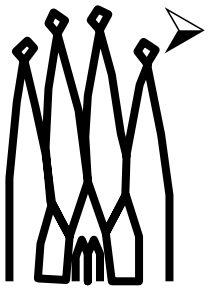


From Transient OID to the Object



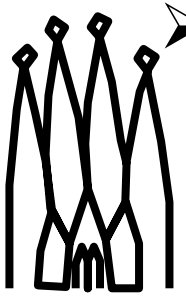
Caveat: Writing Objects

- **Two step process: Bi-directional links**
  - Acquire persistent references for all related objects
  - Convert objects using the persistent refs and write them
- **Easy said, but…**
  - Most generic identifier is container size
    (but also the worst)
  - Assume atomic: container.write(obj); oid=container.size()
  - Typically the identifier can only be received when
    really writing the object
  - Dark side of the story



Caveat: Writing Objects

- **Objectivity:** in place allocation returns OID
- **RDBMS:** in place allocation of “empty” object
  - 2 round trips
- **ROOT:** Internally assigns unique object number
  - Cannot be used (=> Other hacks)
- **Other hacks:** use container size
  - Either: 1 Object/Event/Container
  - Or: Remember internally object allocations and writes
  - Nightmare for multiple writers, multi threading



Conclusions

- It is possible to write physics data without knowledge of the underlying store technology
- This approach can adopt any technology based on database files, collections and objects within collections
  - ZEBRA, ROOT, Objy and RDBMS
  - Allows to choose technologies according to needs
  - Allows to mix technologies