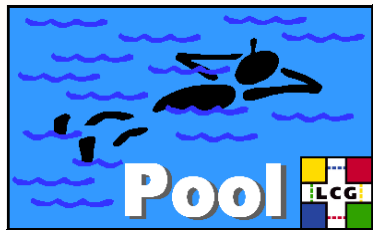
POOL Data Storage, Cache and Conversion Mechanism

- What we want
- Main components and contributors
- Main design issues and component walkthrough
- Documentation
- Performance

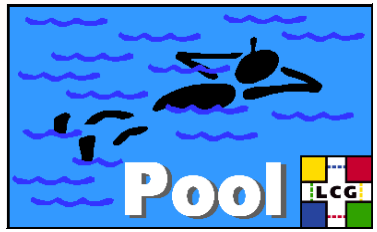
CAT team, M. Frank, G. Govi, I. Papadopoulos

Applications Area Review 2003
October 21, 2003

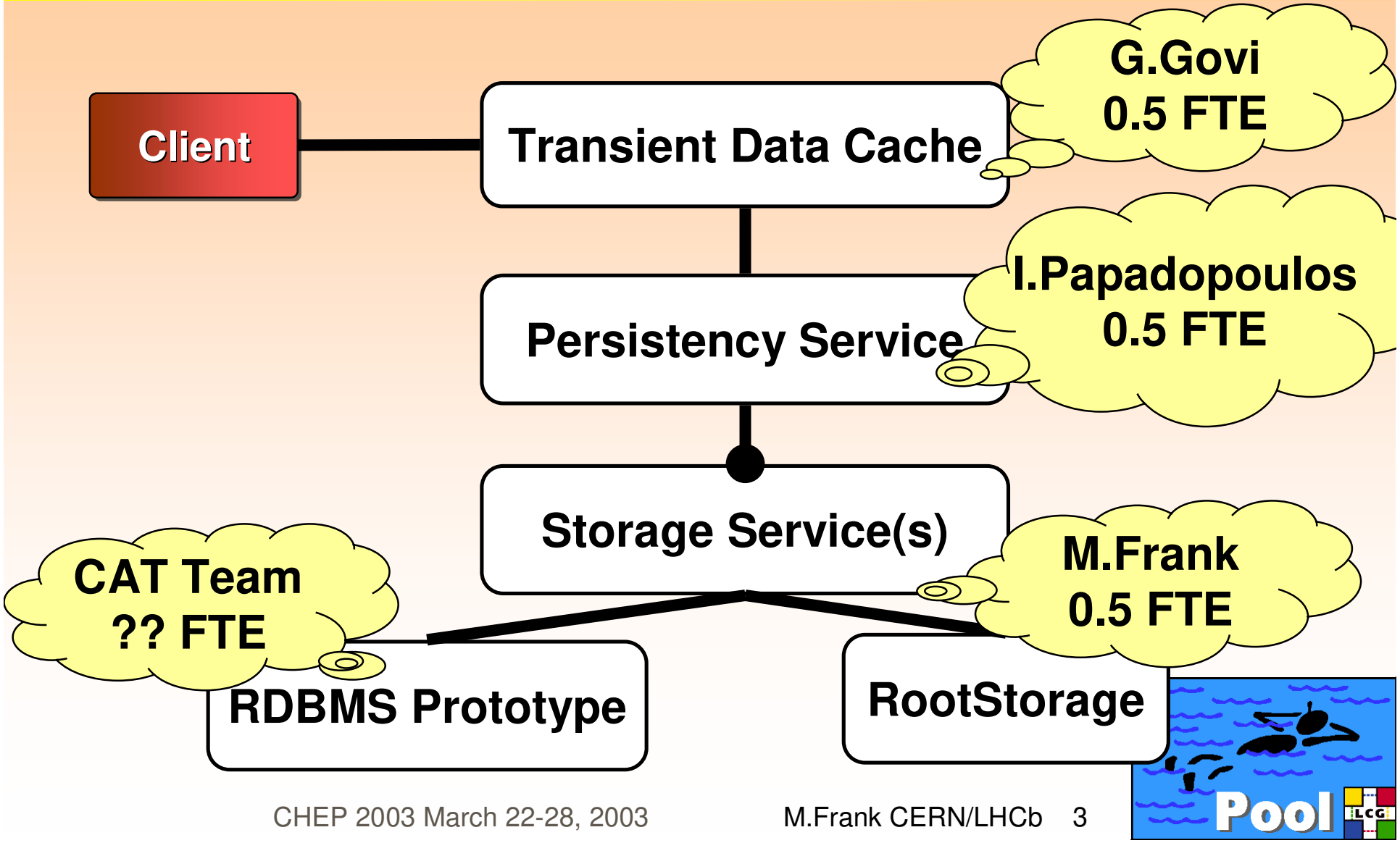


Motivation

- Save and restore physics data independent of the underlying data storage technology
  - Keep full connectivity between objects
- Address data sources of different nature
  - Event data, detector data, statistical data, …
  - The data sizes: O(10^6) to O(10^{13}) Bytes/experiment/year
  - The access patterns differ
- Hide any technology details from the clients
  - Hide all cache/persistency specific details

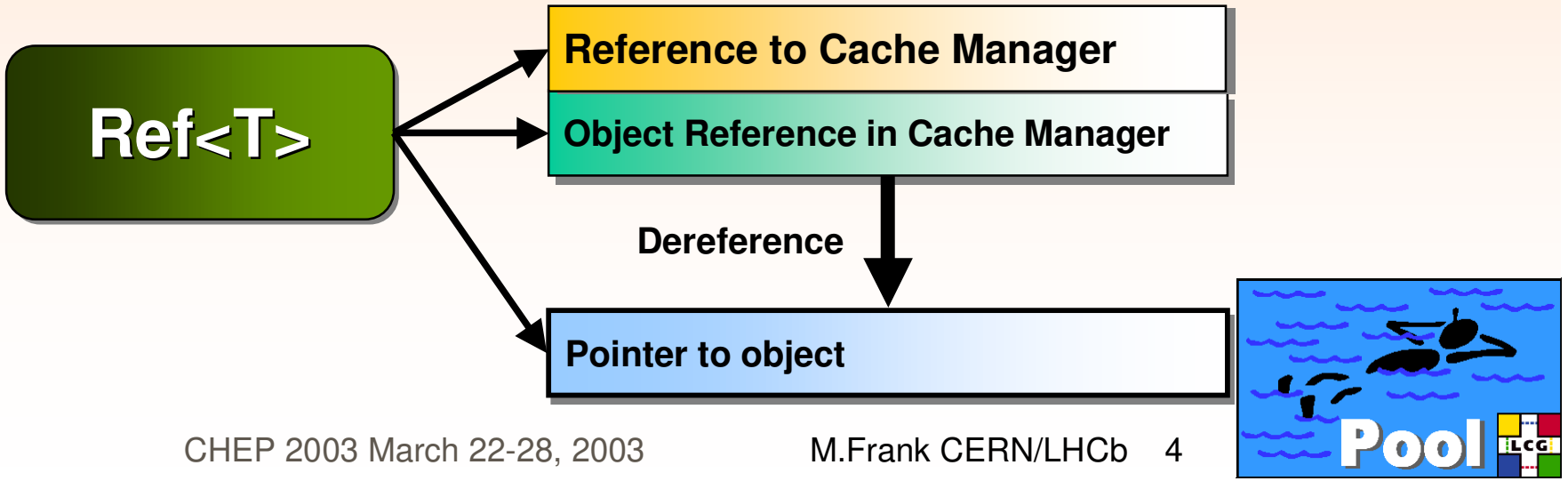


Breakdown of Major Components

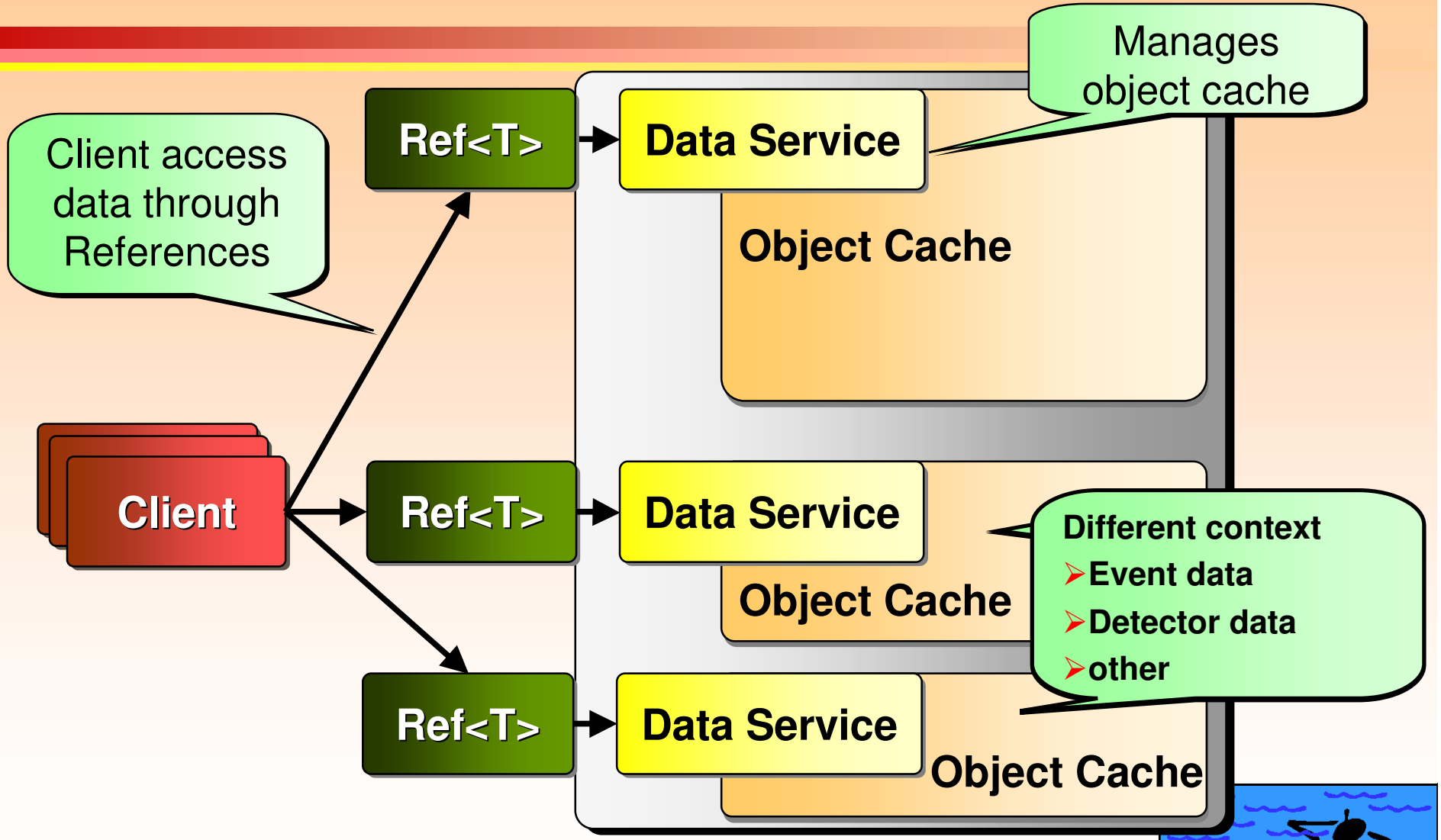


Cache Access Through References

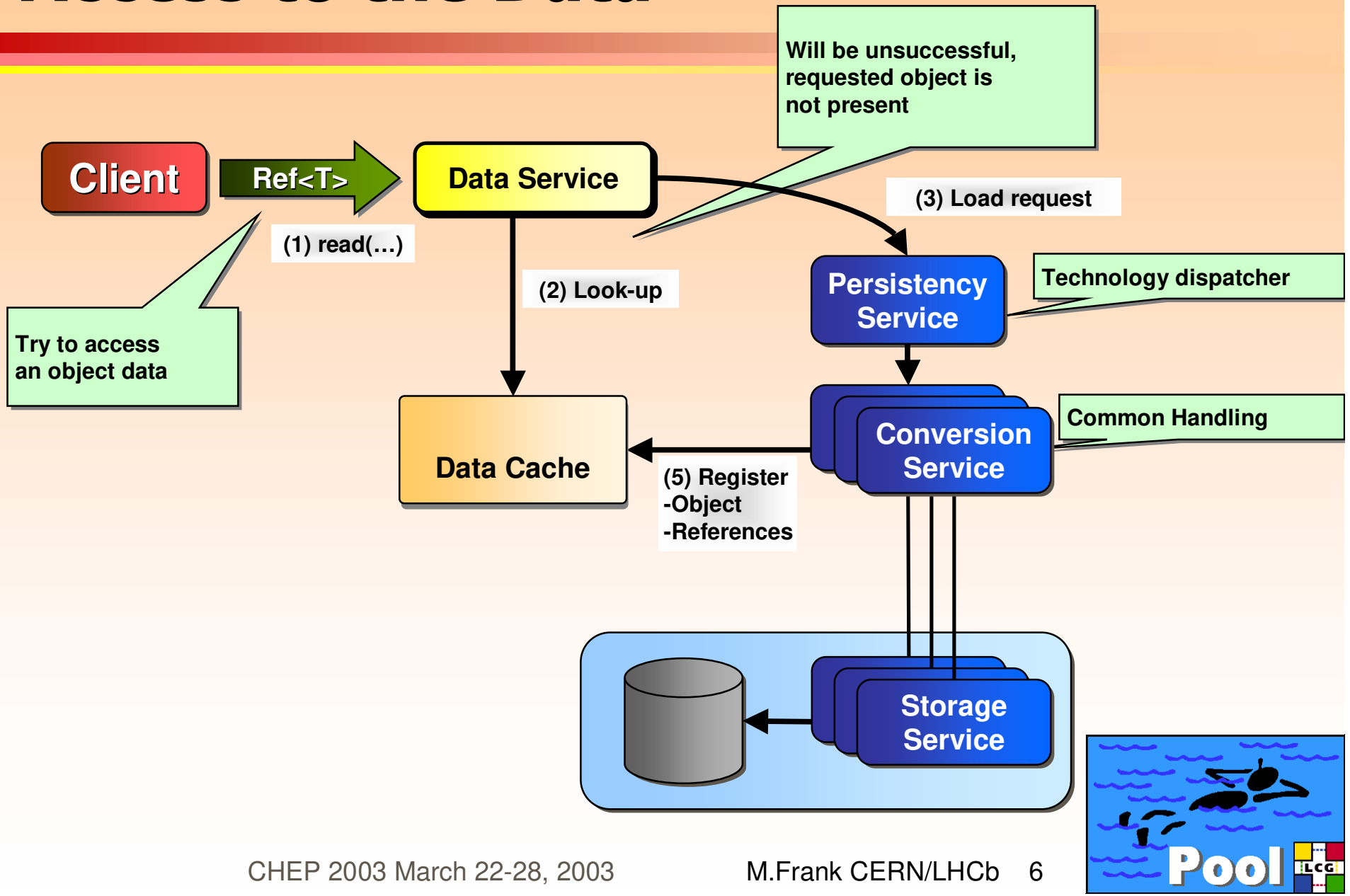
- References know about the Data Cache
  - 2 operation modes: - Clear at checkpoint
    - Auto-clear with reference count
    - With/without object deletion
- References are implemented as smart pointers
  - Use cache manager for "load-on-demand"
  - Use the object key of the cache manager



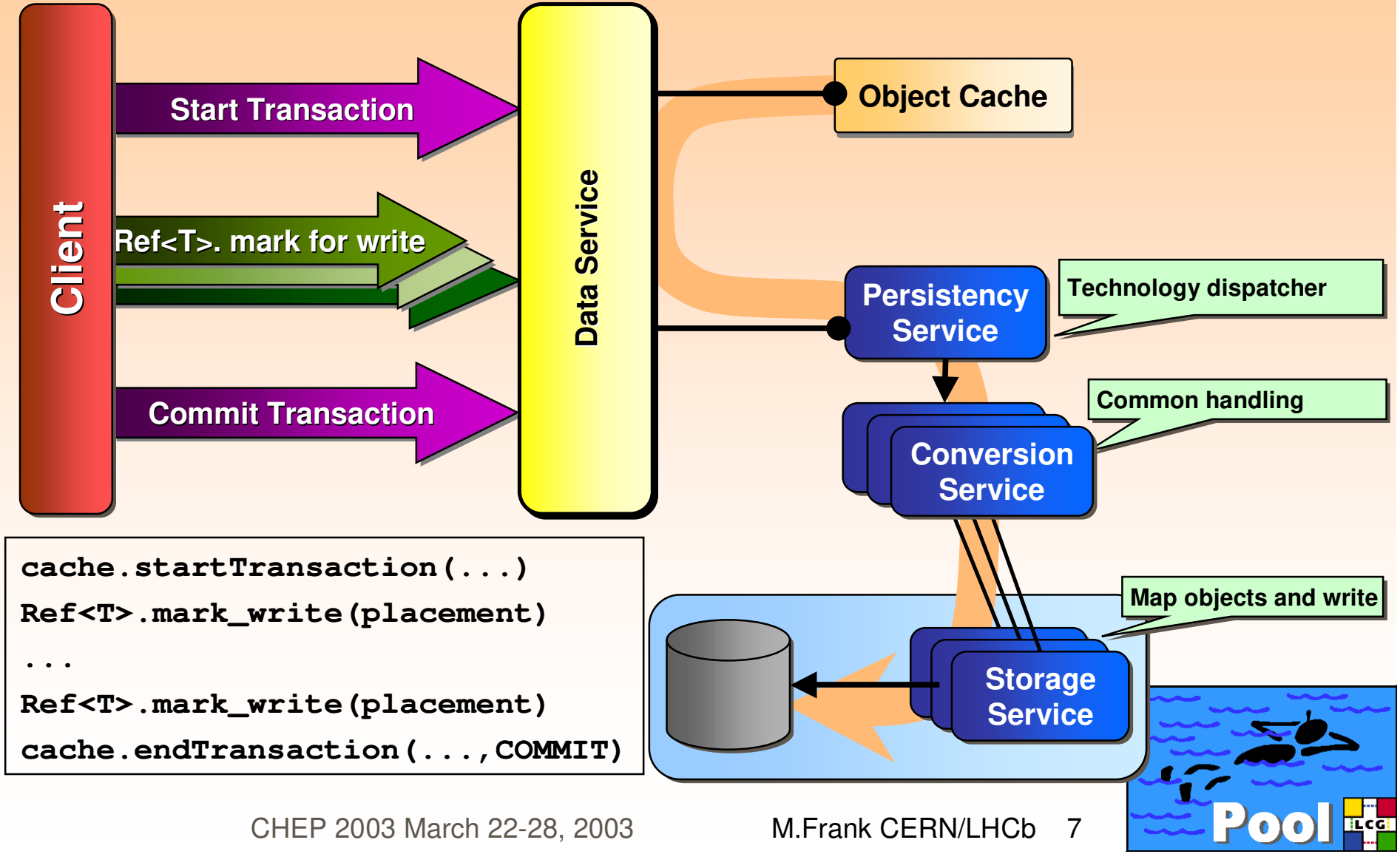
Client Data Access



Access to the Data



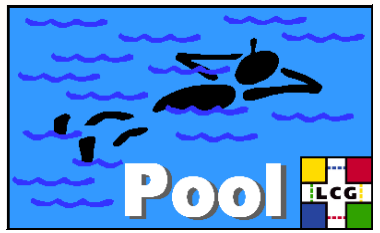
Storing objects




```
cache.startTransaction(...)
Ref<T>.mark_write(placement)
...
Ref<T>.mark_write(placement)
cache.endTransaction(...,COMMIT)
```

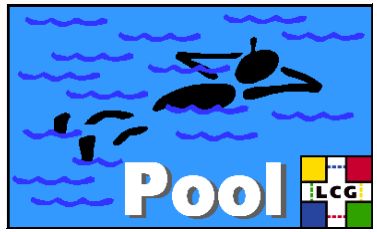
Common Transaction Model

- Implemented by persistency service package
- Main client is the data cache
- Acts on all open databases
  - Transaction handling: start, commit, rollback
  - Pros and cons
  - Can't "forget" an open file
  - Not for free (CPU, I/O etc.)

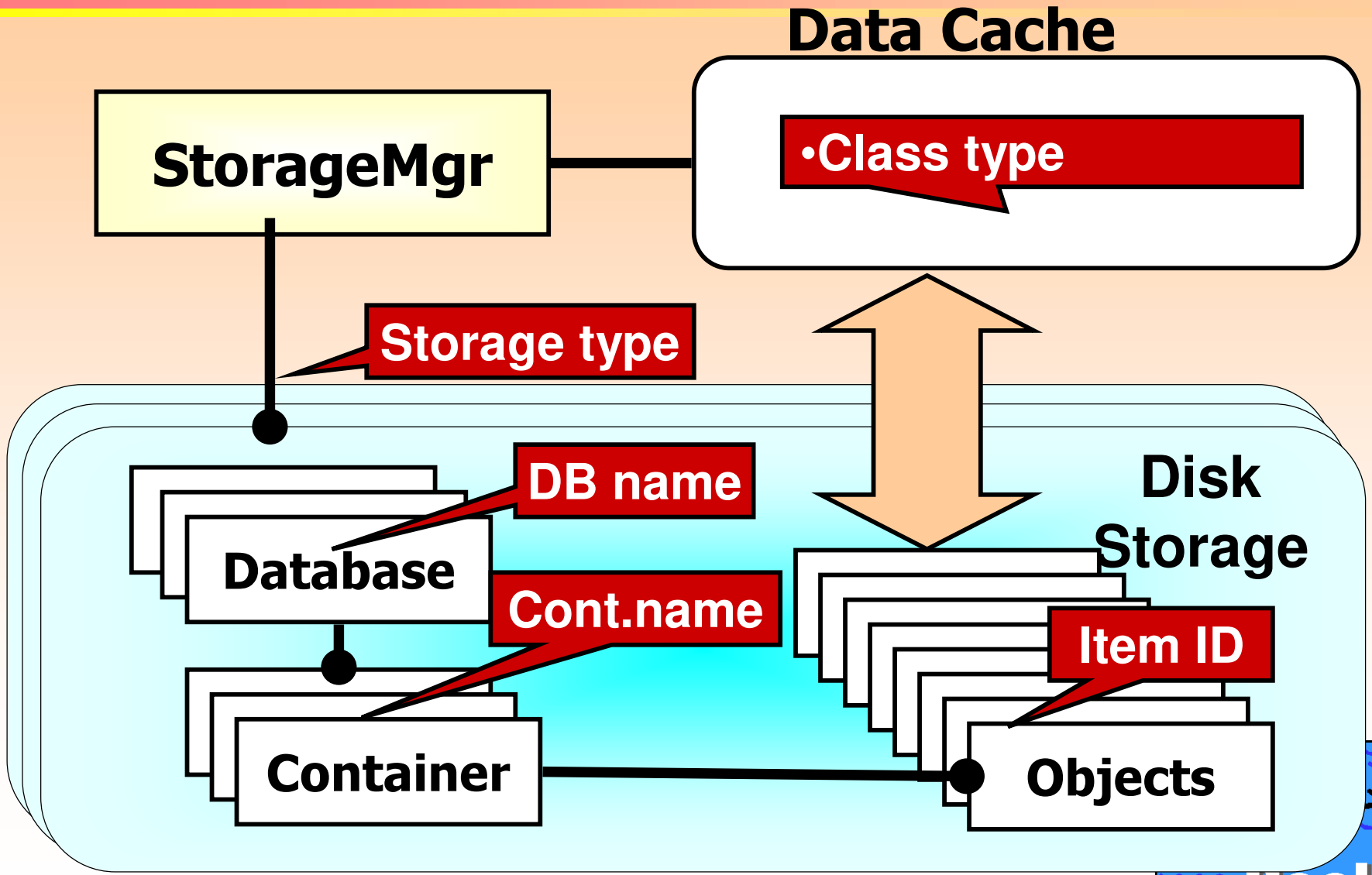


The Storage Mechanism

- The underlying model assumptions
- Migrating objects to/from the persistent medium
  - Object mapping
- Reference handling
  - References are objects, not primitives
  - Need setup: Reference to data cache
  - ROOT: Callback for base class (Streamer)

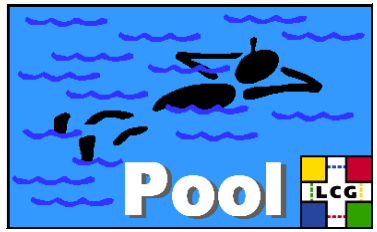


Model Assumptions

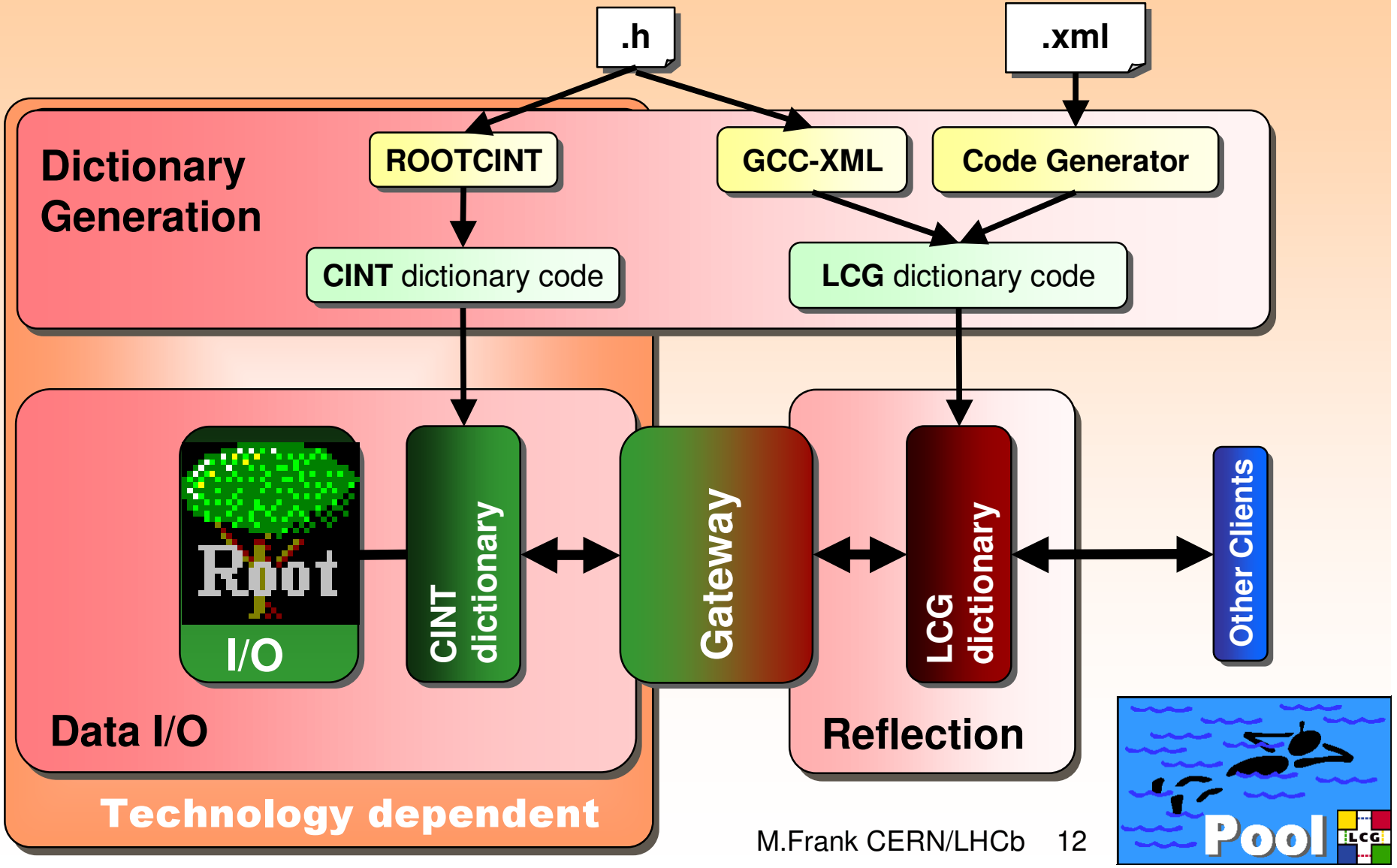


Object Rendering

- Objects must maintain personality when persistent
  - Allow for queries, selections and independent element access
- If technology supports objects…
  - Want to make use of such features
  - These technologies must be instructed how to do it
    - Need object dictionary
- If technologies support only primitives
  - Split objects into primitives [until reasonable level]
  - Need full access to object member data [member offset, type]
  - Constructor and Destructor with defined signature
    - Need object dictionary



SEAL Dictionary: Reminder

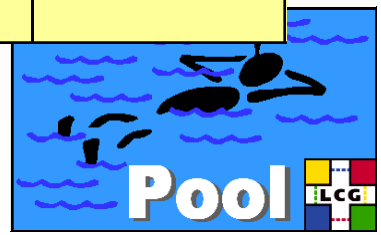


Column Wise Object Layout

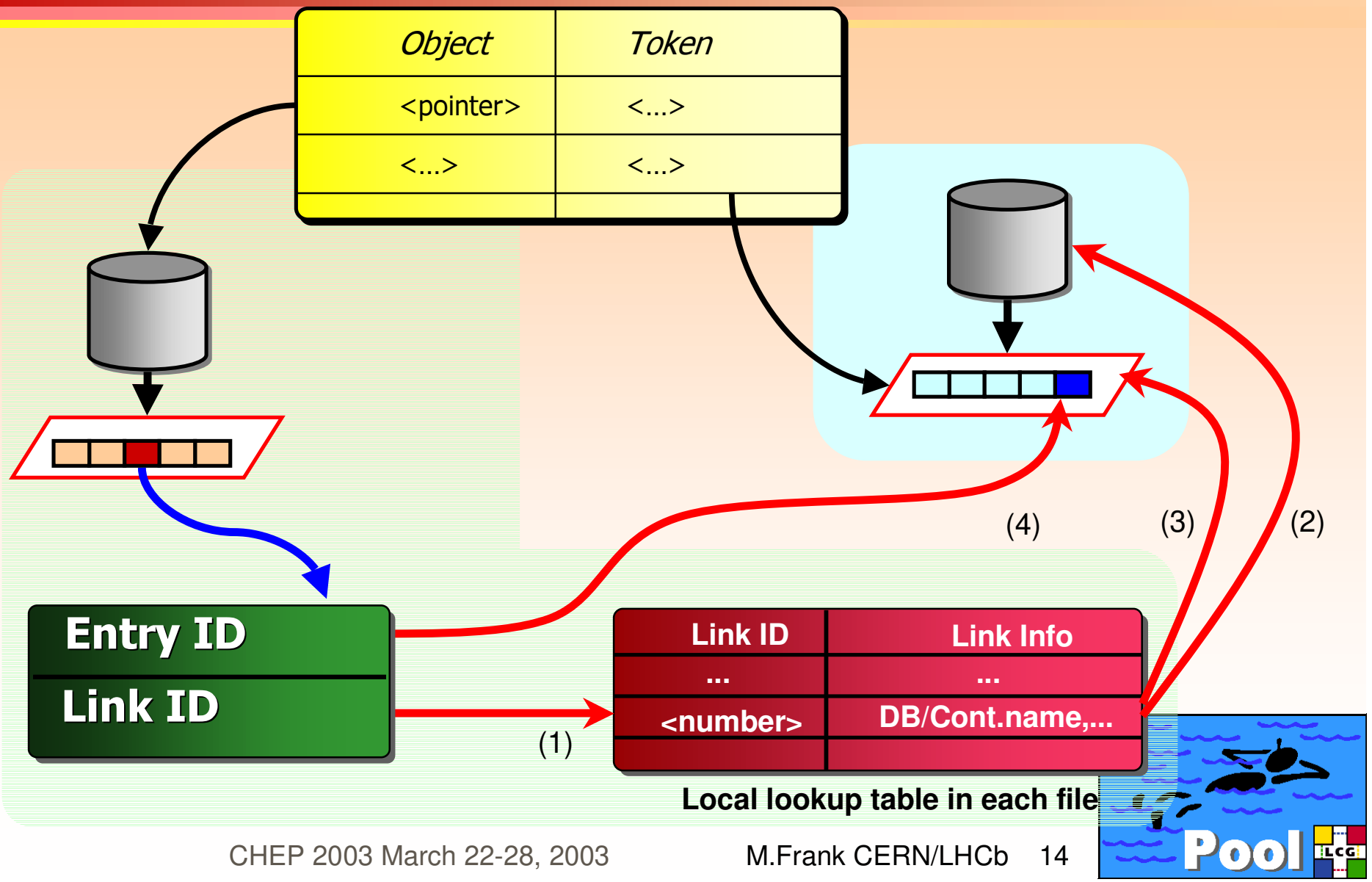
- Object must be split into primitives
  - Done by ROOT
- Columns wise data support
- For POOL (using ROOT)
  - Objects are primitives
- But also POOL can split
  - Ease support for "stupid" back-ends (RDBMS)
  - Store more complex object models

px	py	pz	E	...	m_?
...	...	...	...	...	...

Object1	Object2	...	Object n
...	...	...	...

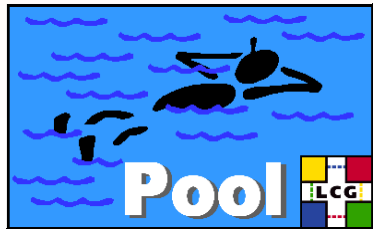


Follow Object Associations



The “Link” Table

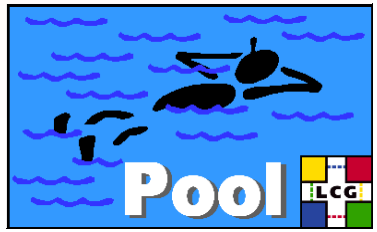
- Optimize size of persistent data
  - Keep full functionality
- Contains all information to resurrect an object
  - Storage type
  - Database name
  - Container name
  - Object type
- Local to every database
  - Limited size, scalable



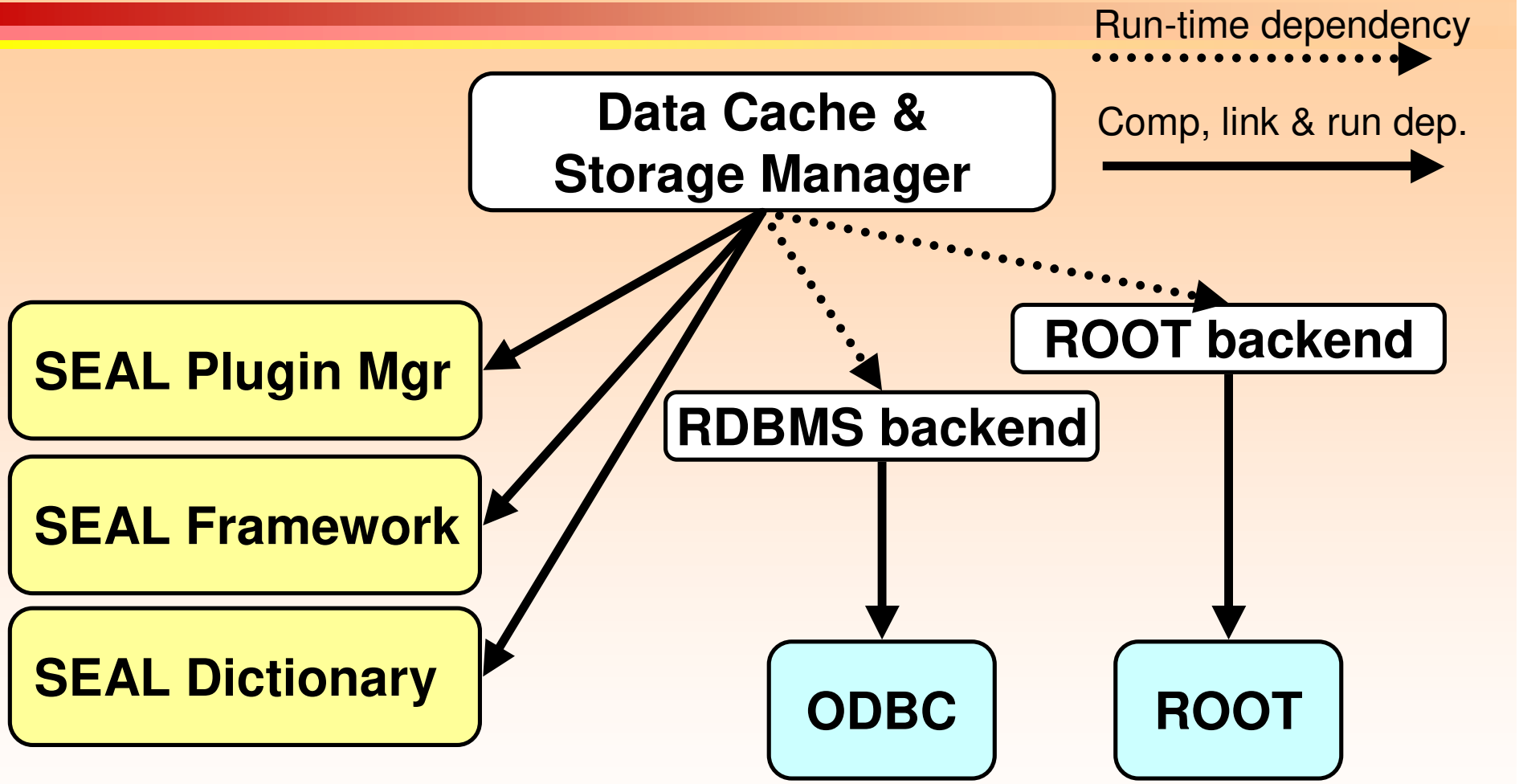
Documentation

- Interface design documents
- Published paper
- Some doxygen generated documentation
- Many examples (too many?)
- Code is still quite rapidly changing
  - It's a bit early for a stable reference manual
  - Good reference manual will have to be addressed
  - May be doxygen based

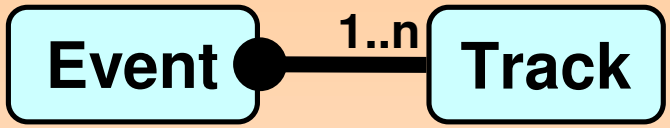
Could be worse, but certainly is not the most glorious point on the agenda



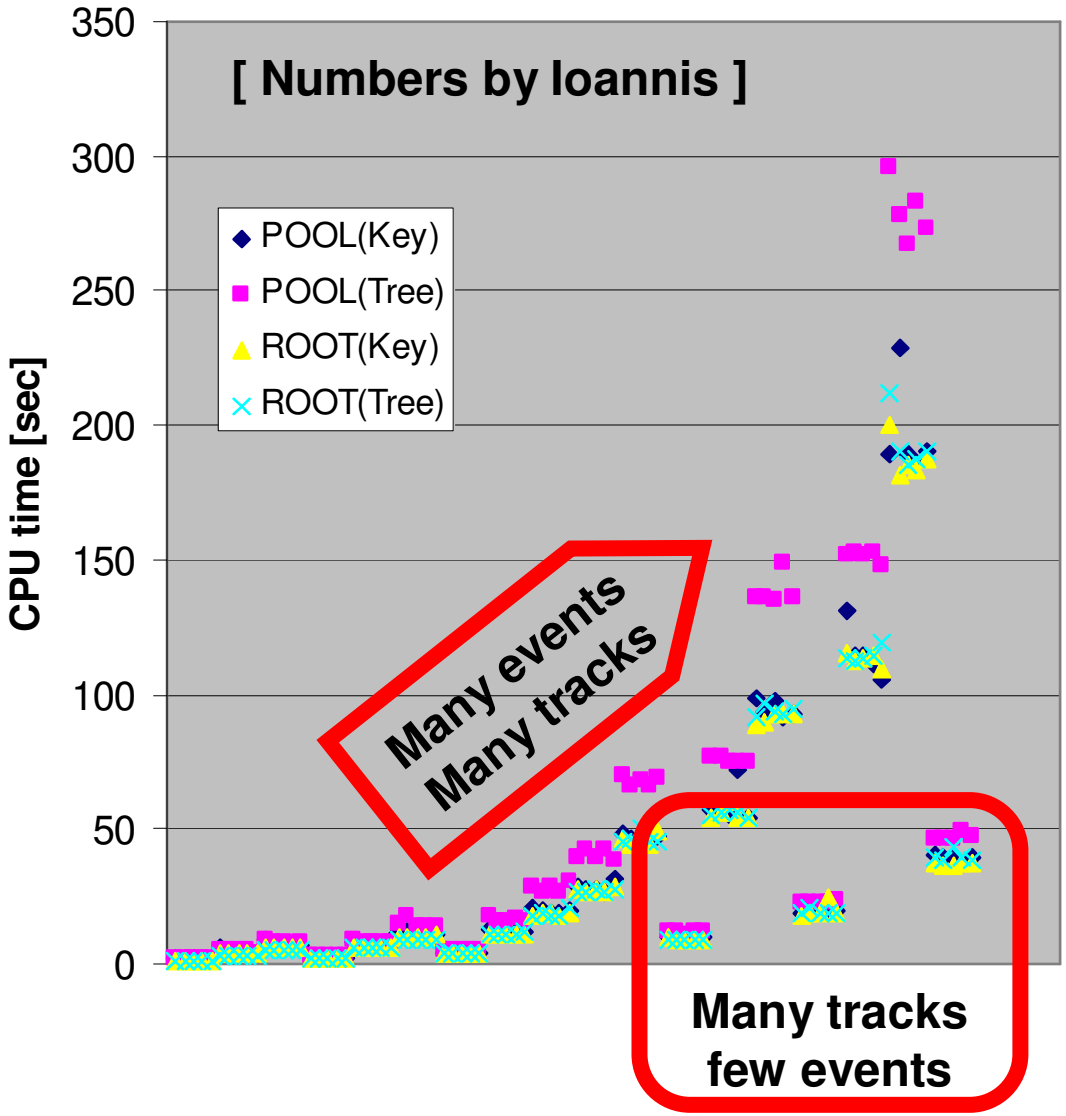
Software dependencies



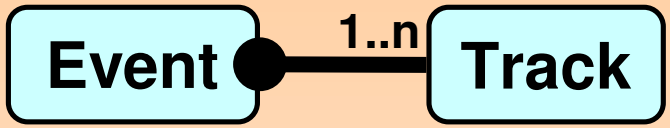
CPU Performance (write)



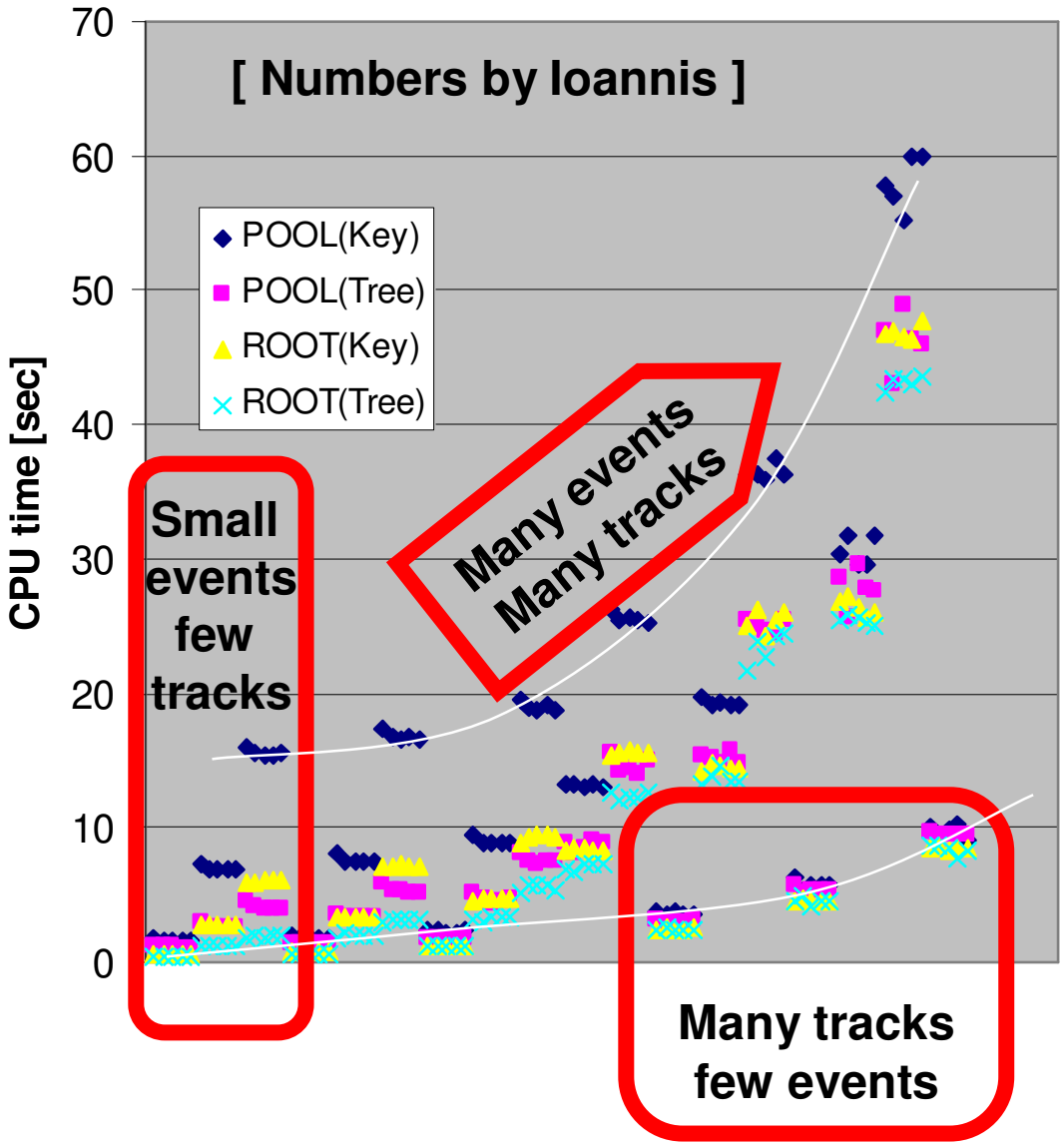
- CPU Usage as function of
  - #events
  - #tracks/event (Event size)
  - #events/transaction
- Be careful
  - Don't compare apples and oranges
- Small events & many transactions => POOL gets bad Framework overhead



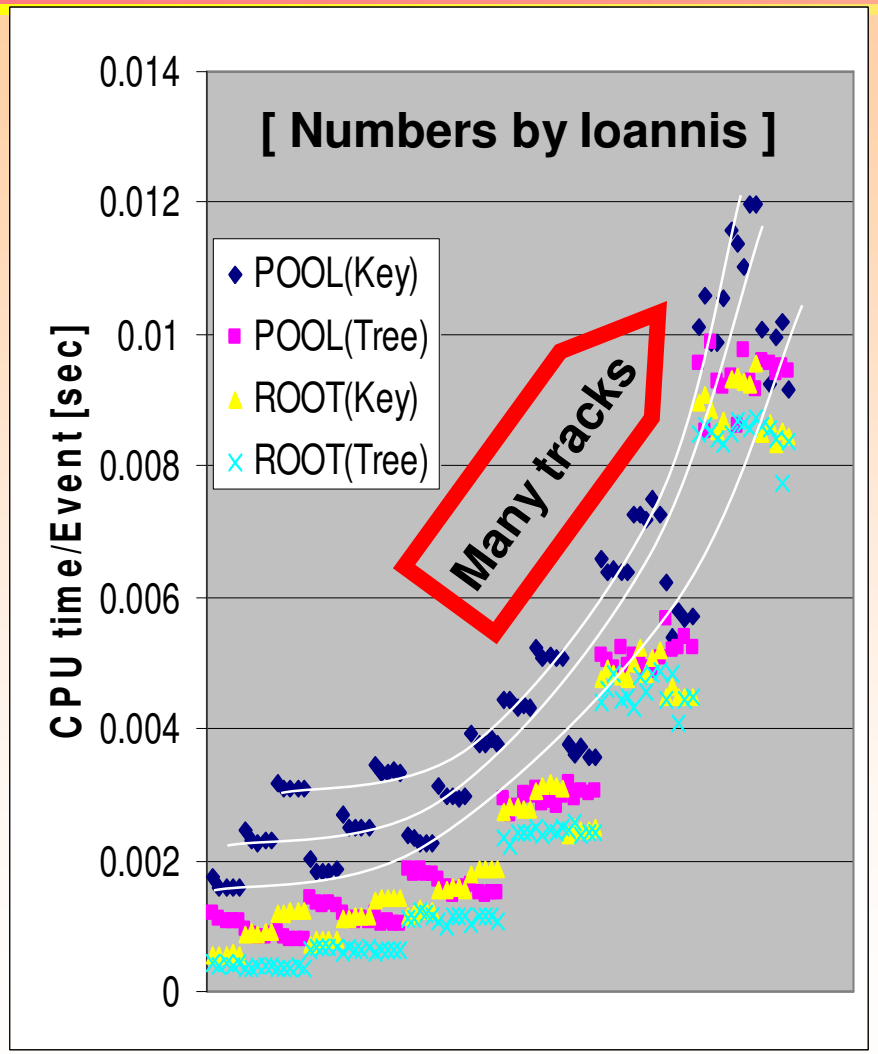
CPU Performance (read)

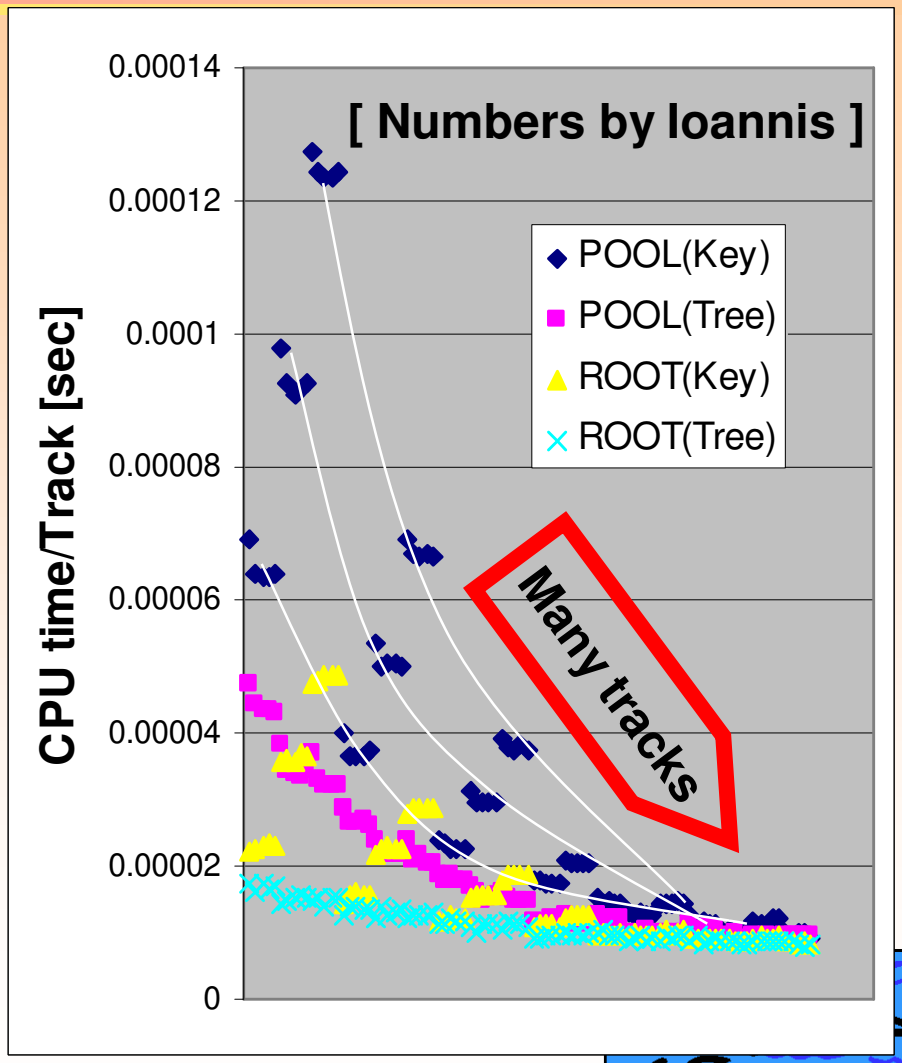


- CPU Usage as function of
  - #events
  - #tracks/event (Event size)
  - #events/transaction

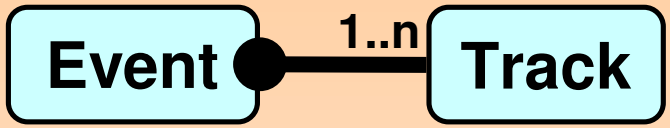


CPU per Event and Track (read)

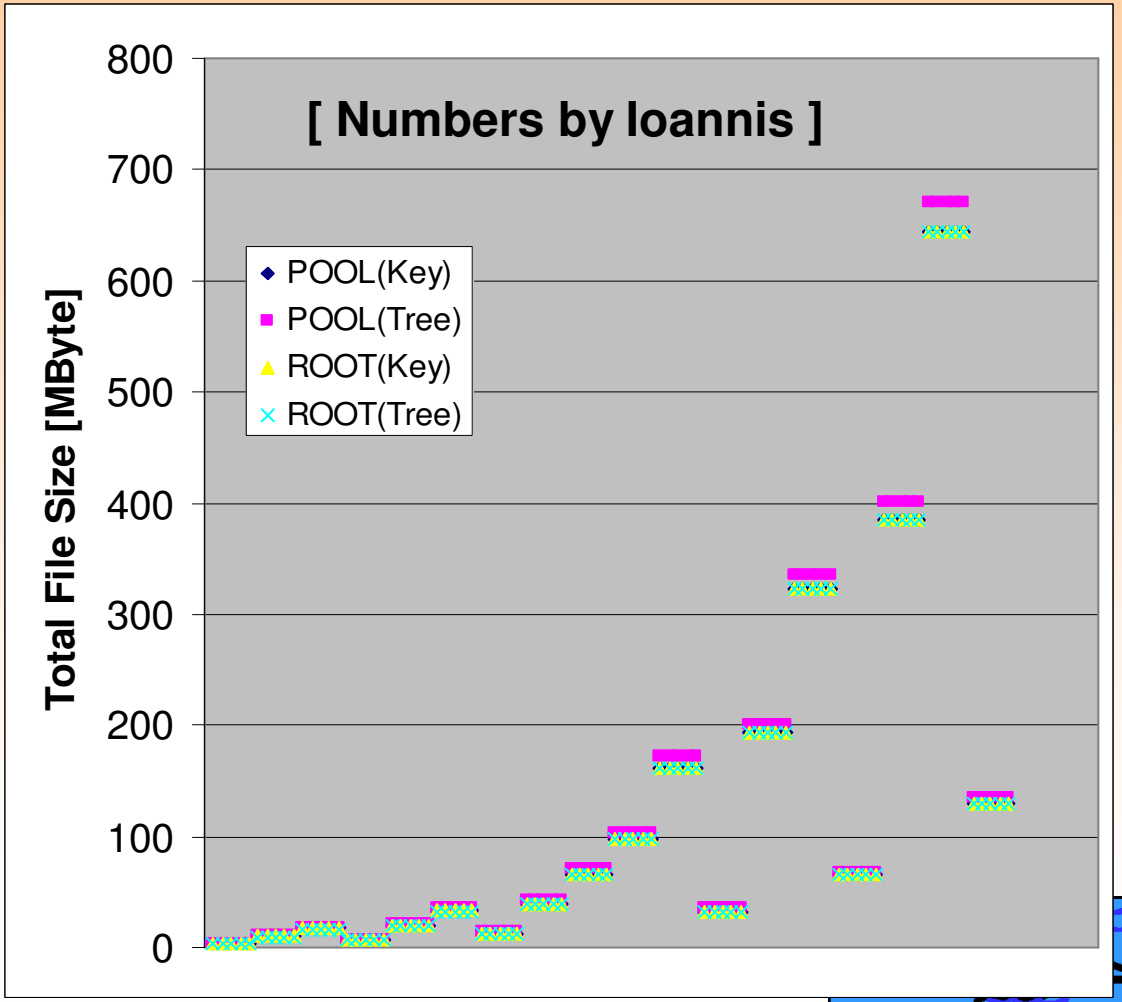




File size

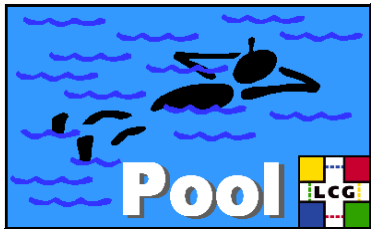


- Disk Usage as function of
  - #events
  - #tracks/event (Event size)
  - #event/transaction
- No big difference to ROOT (no surprises)

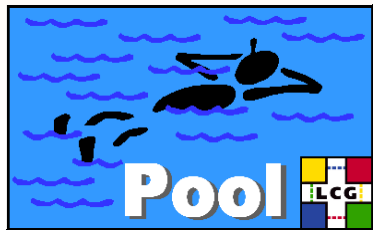


Further Development

- No open milestones
- Further developments
  - Depends if someone has time
  - Plenty to do
    - RDBMS integration
    - ROOT backend works, but is not perfect
    - Optimizations: Speed, CINT usage etc.
  - My part will go down (POOL integration in Gaudi for next ½ year)
  - i.e. left with 2 x 0.5 FTE
- Maintenance will go up
  - Bugs start flowing in



Conclusions

- Cache and storage manager work well
- …at least we survived the first release(s)
- Solution got picked up by 3 out of four expts.
- We have not seen dramatic penalties & overheads

- No plans for many new developments
- Focus on consolidation and optimization