Lightweight Parallel Object Management for Scientific Computing

John Ambrosiano

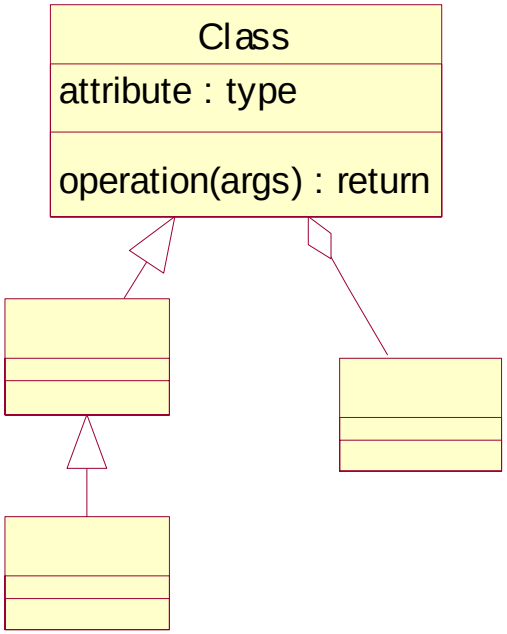
Los Alamos National Laboratory

Workshop on Object-Oriented and Component Technology, July 23-25, 2001

Outline

- Why object-based data management?
- What do “lightweight” and “high-performance” mean?
- Special issues for high-performance computing
- Exploring patterns for a simple open architecture
- Ongoing development

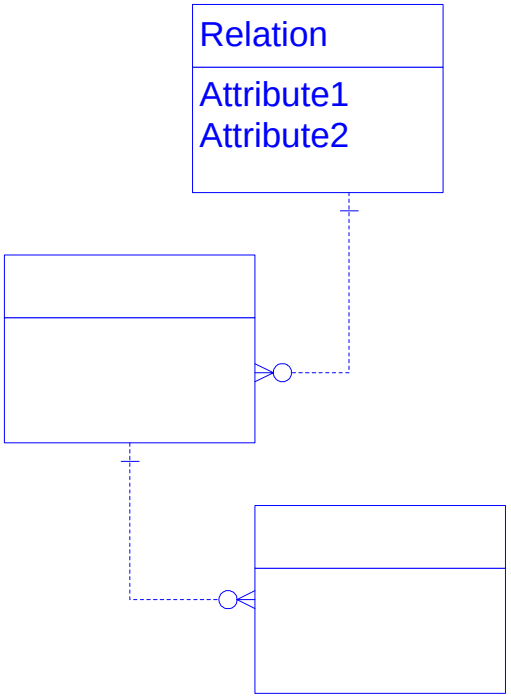
Objects vs. Relations



Pragmatic model (Math: Category theory?)

Relation	
Attribute1	Attribute 2

Solid math model (Set theory, Codd, 1970)



Relational Data Problems

- Relations are too general
- Relations are highly schematic, generic approaches are difficult
- Functions and operations are poorly bound to data
- Naturally useful cognitive relationships such as “is-a” “has-a” “affects” “affected-by” are not directly supported
- Less relevant to human information processing and complex process modeling than OO models

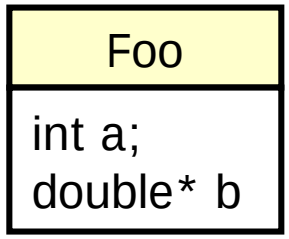
Issues for Science and HPC

- Databases (OODB) are optimized for
  - Millions of small-to-medium objects
  - Object images that fit in memory
  - Large numbers of ad hoc queries
  - Absolute data integrity and fault tolerance
- Scientific applications
  - Have hundreds to thousands of large, distributed objects that don't fit in memory
  - Use more directed queries
  - Trade off integrity and fault tolerance for I/O throughput

POOMA Object Serialization

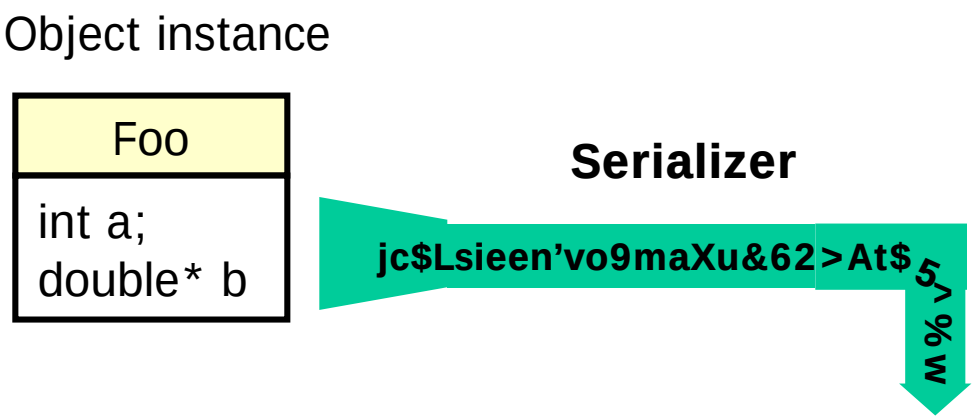
- POOMA Object I/O used template-based "serializers" to move POOMA objects to disk.

Object instance



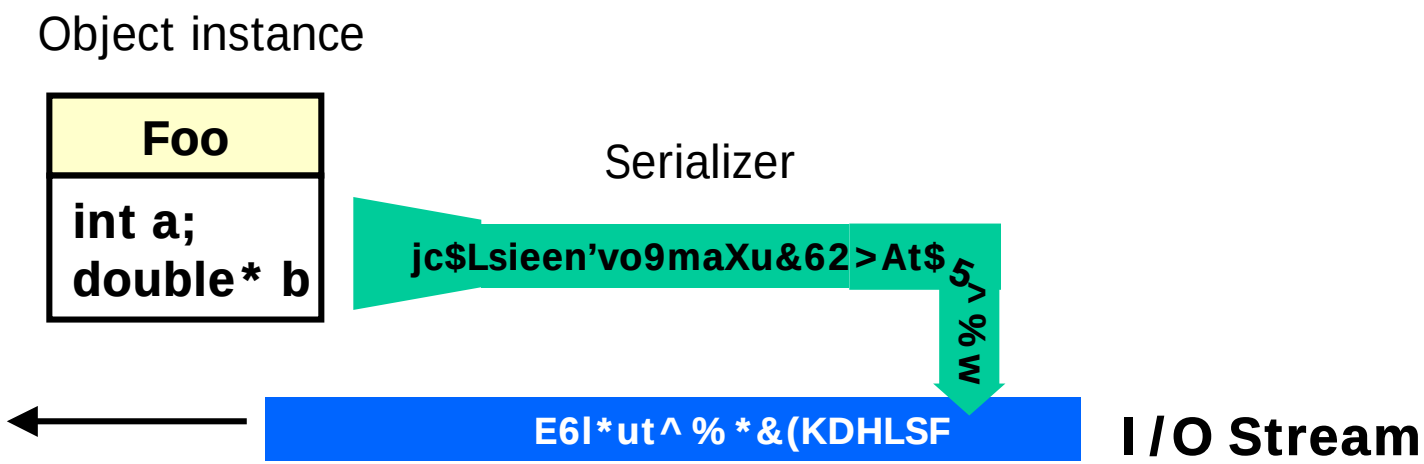
POOMA Object Serialization

- POOMA Object I/O used template-based "serializers" to move POOMA objects to disk.



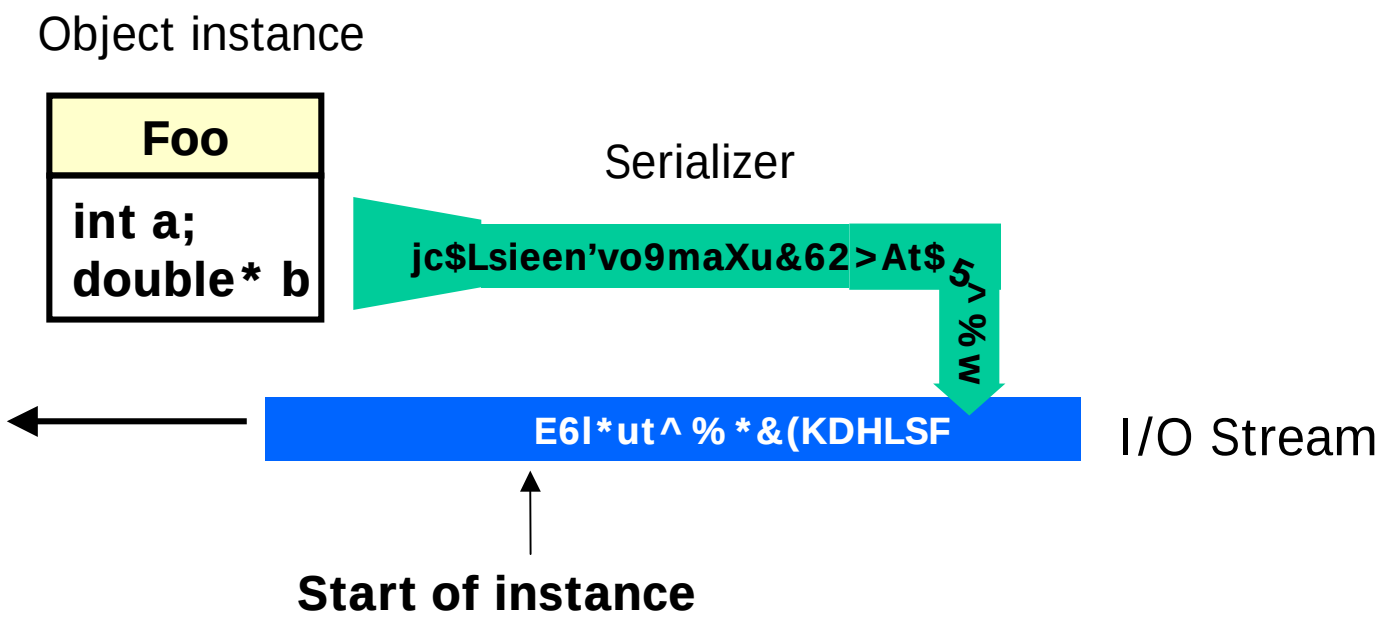
POOMA Object Serialization

- POOMA Object I/O used template-based "serializers" to move POOMA objects to disk.



POOMA Object Serialization

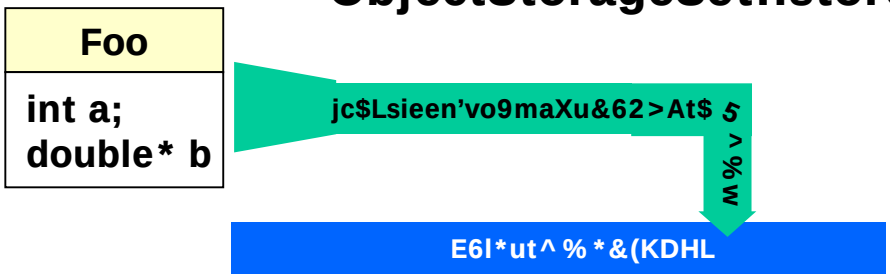
- POOMA Object I/O used template-based "serializers" to move POOMA objects to disk.



POOMA Object Serialization

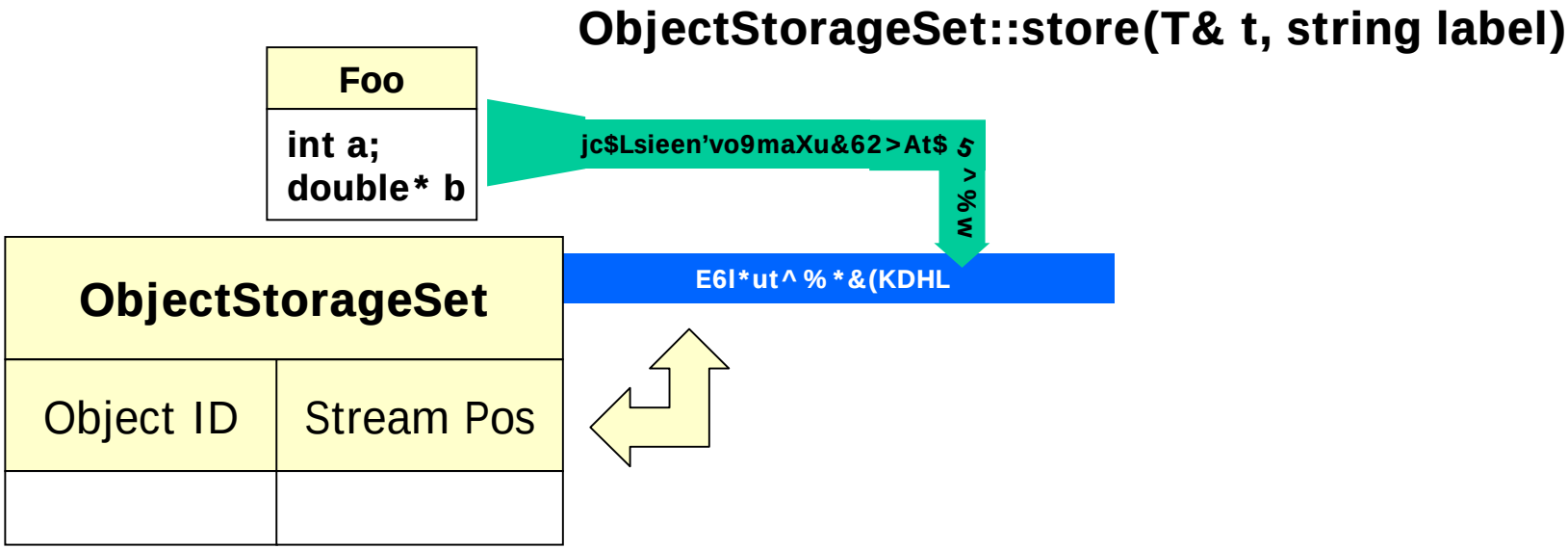
- Dictionary and metadata stored in the stream

ObjectStorageSet::store(T& t, string label)



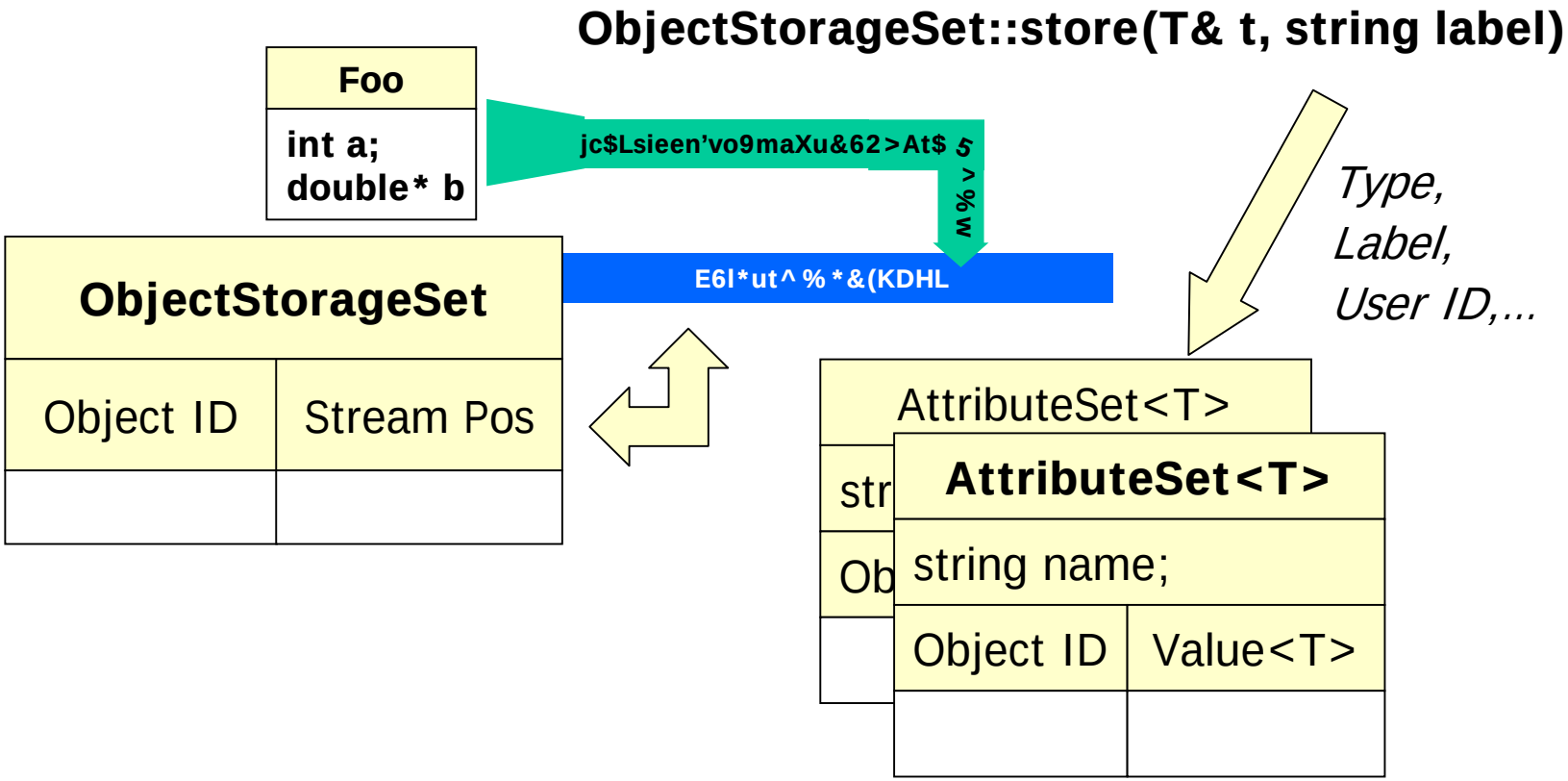
POOMA Object Serialization

- Dictionary and metadata stored in the stream



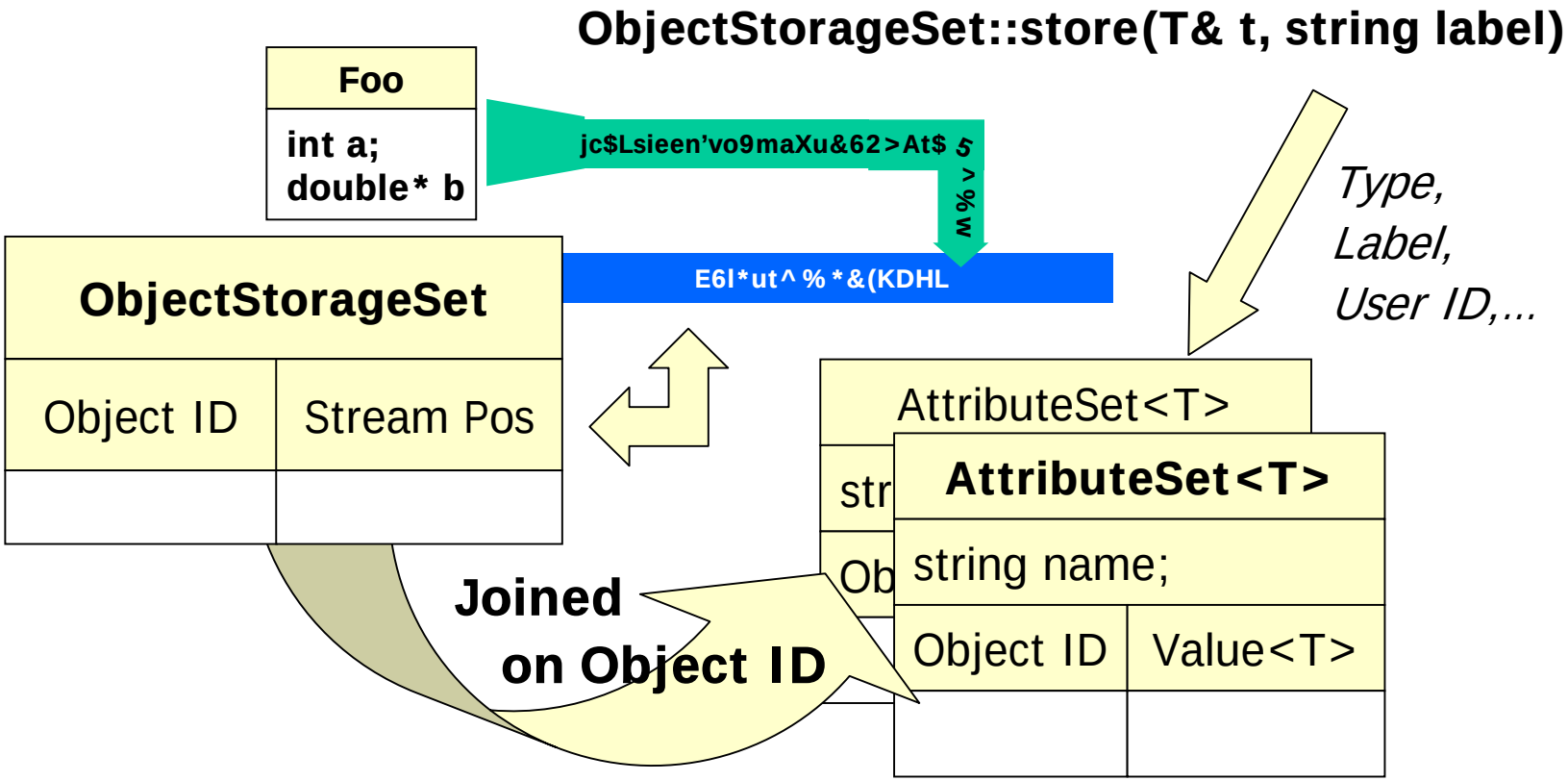
POOMA Object Serialization

- Dictionary and metadata stored in the stream



POOMA Object Serialization

- Dictionary and metadata stored in the stream



POOMA Object Serialization

- Implementation was simple† and performed well with straightforward optimizations

Table 1. I/O Rates in Storing Array Objects of Varying Size

Array Dimensions	Num. of Bytes	Raw I/O Rate	Object I/O Rate*	Percent Efficiency*
10 x 10	400	0.36Mb/s	0.14 (0.2)	39% (56%)
100 x 100	40k	9.7Mb/s	5.8 (2.1)	69% (22%)
1000 x 1000	4M	6.8Mb/s	7.1 (2.3)	~100% (33%)
10,000 x 10,000	400M	4.5Mb/s	4.5 (2.2)	~100% (49%)

** Unoptimized results are shown in parentheses.*

† (too simple)

Shortcomings

- Dependent on the POOMA library and C++ language
- Dependent on a particular data-parallel use case
- Not integrated with existing commercial solutions
- Not a real data management solution

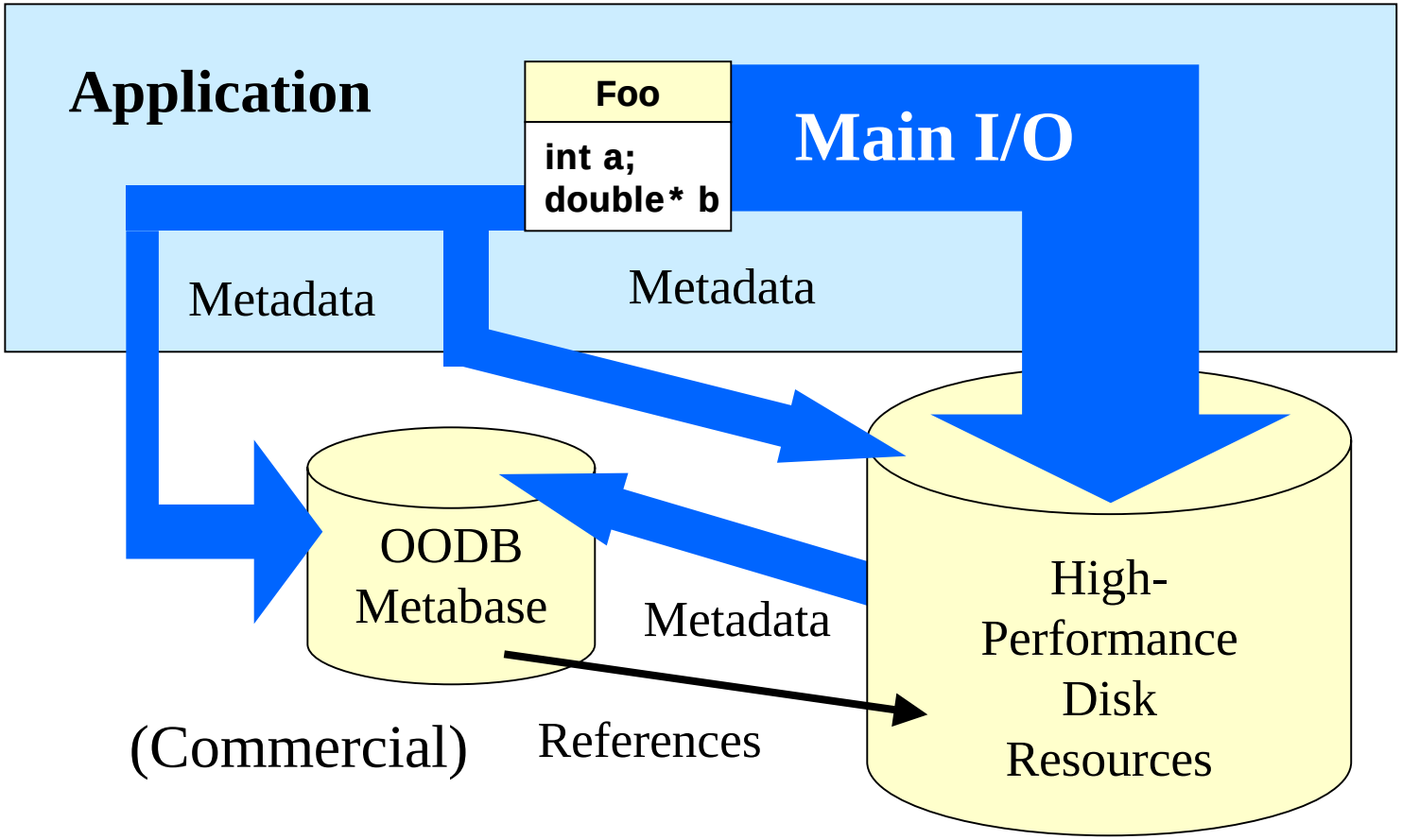
Current Project

- Explore useful patterns for HPC object management based on serialization and parallel I/O
- Look for patterns that couple well to HPC use cases
- Look for opportunities to integrate with commercial distributed computing and component models
- Enable real object database capabilities

Idea

- Databases are optimized for ad hoc queries and fault tolerant transactions. HPC applications mainly need **high throughput parallel I/O**. Can we take the POOMA I/O **serialization** idea and move closer to a database model?
- Solutions should be based on **"scientific" parallel computing patterns** and on **flexible transaction models** that allow tradeoff between fault tolerance and throughput

Overall Concept



Relevance to Components

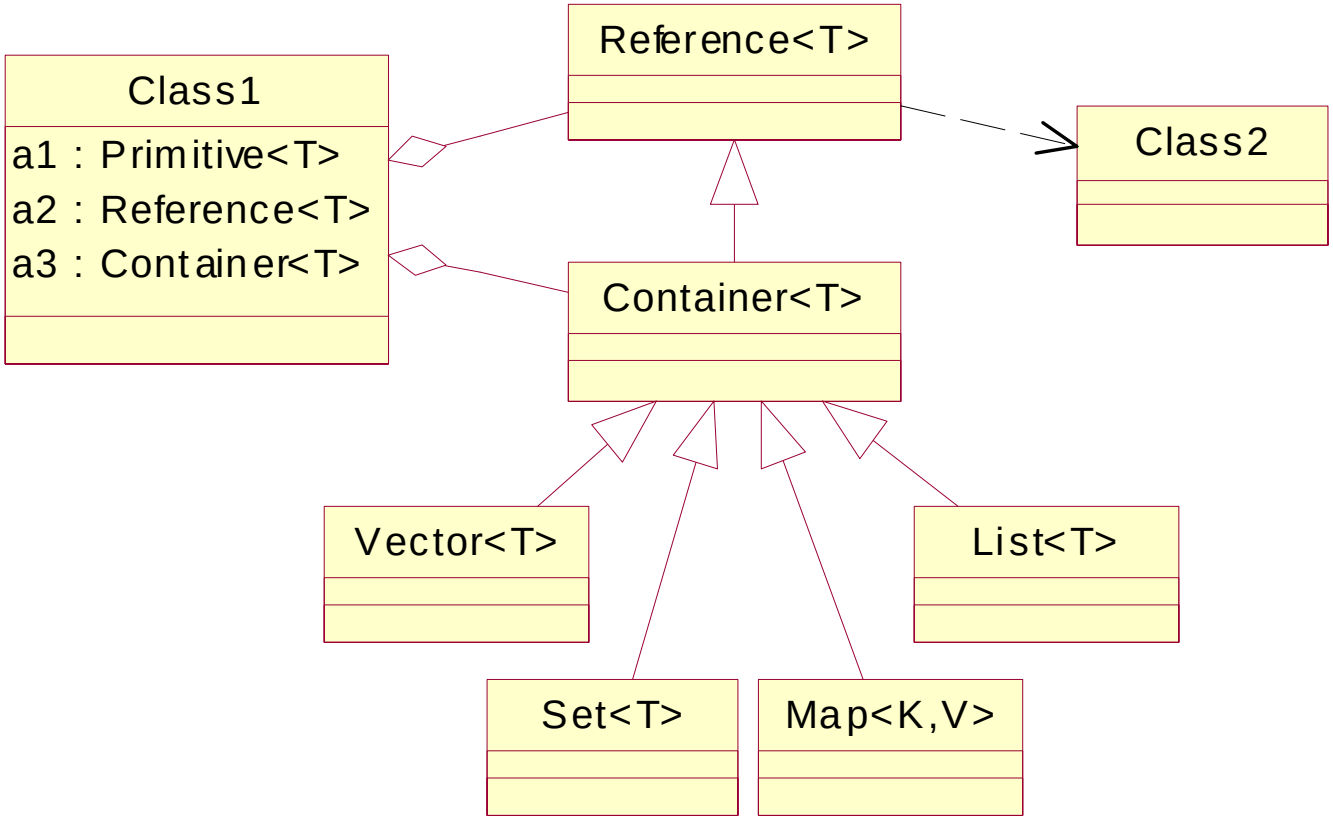
- For our purposes **components**
  - Are **runtime** entities
  - Are based on reusable ***design patterns***
  - Support **reflection**
  - Can be **persistent**
- These are all attributes of the current prototypes

Patterns

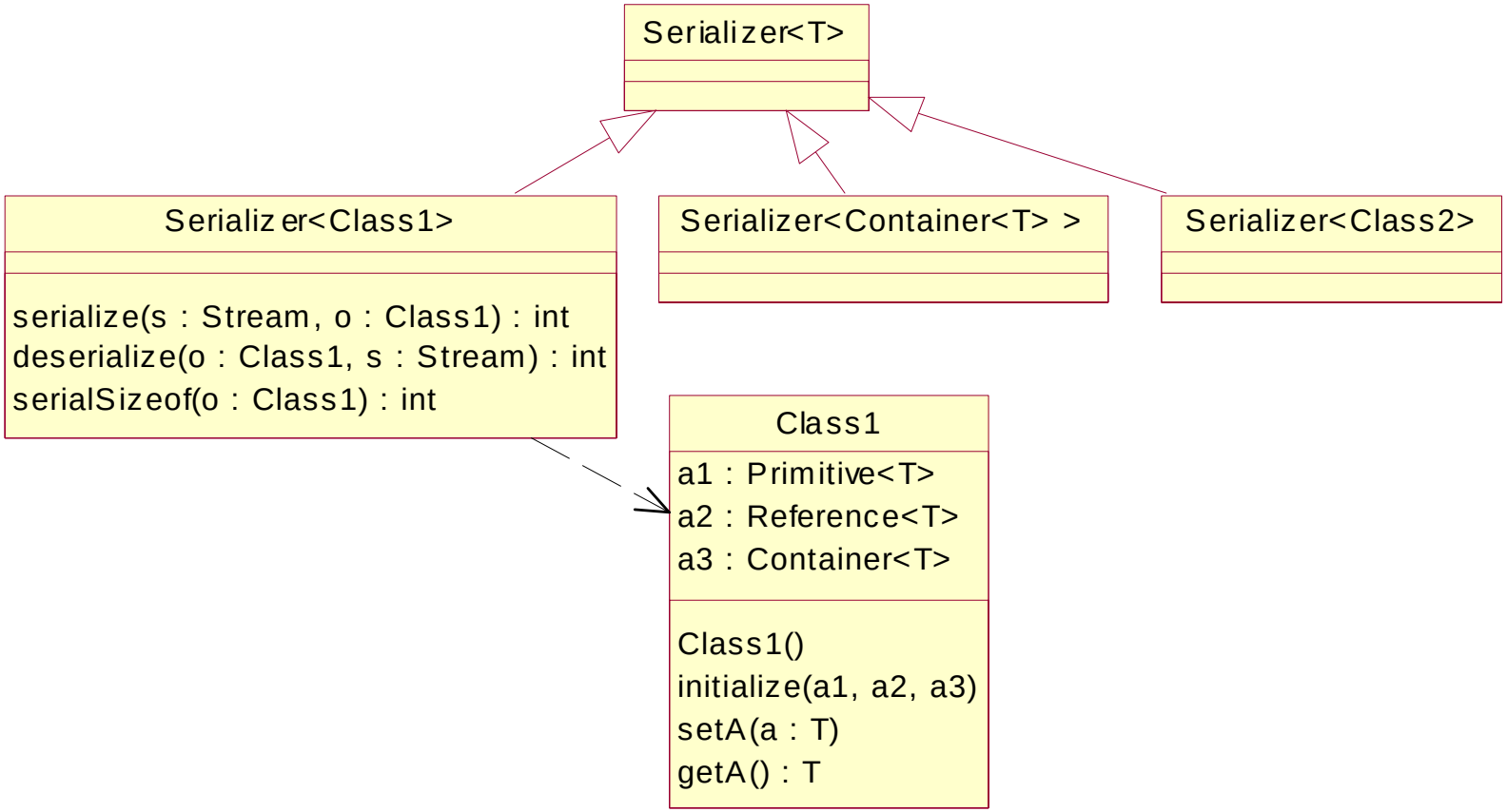
- **Serialization**
- *Storage*
- *Collective Objects*
- *Distributed Object*
- **Metadata**
- **Transactions**

Object Data Management
Parallel Computing

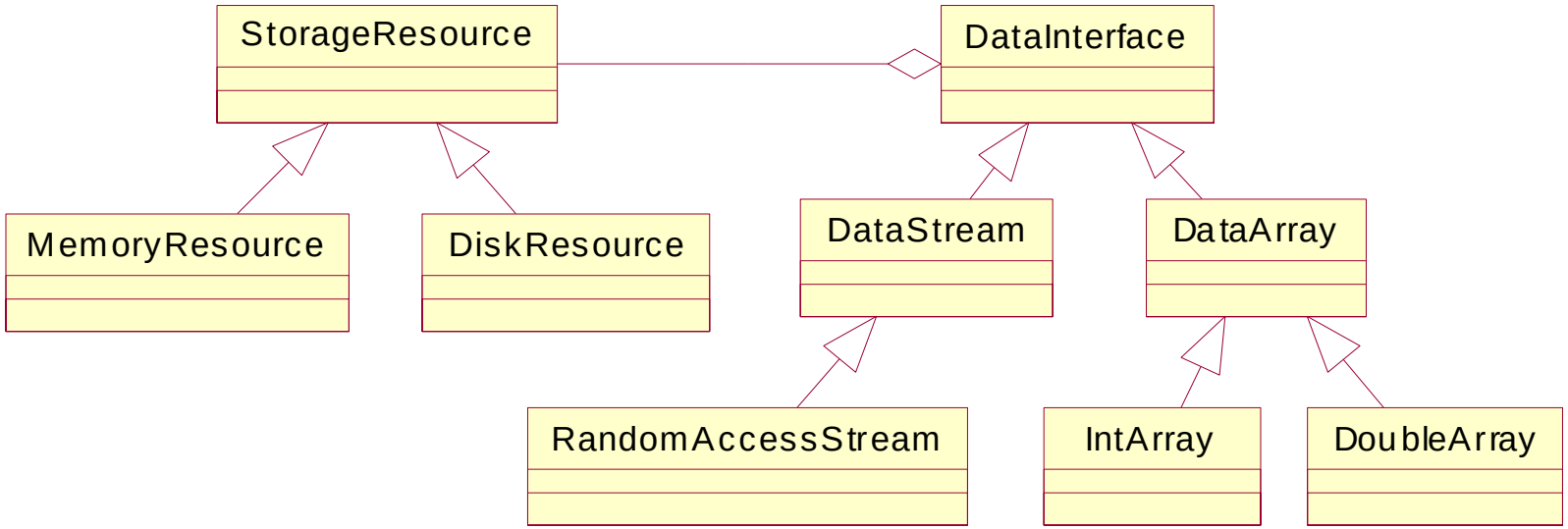
Object Data Model



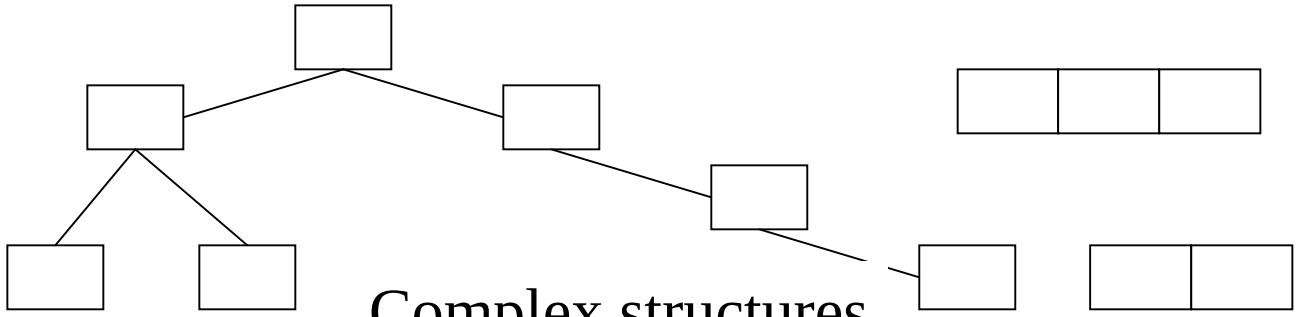
Serialization Contract



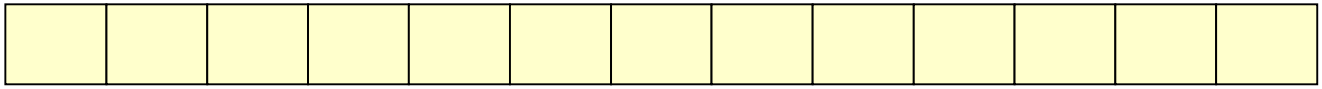
Storage Abstractions



Levels of Container Description



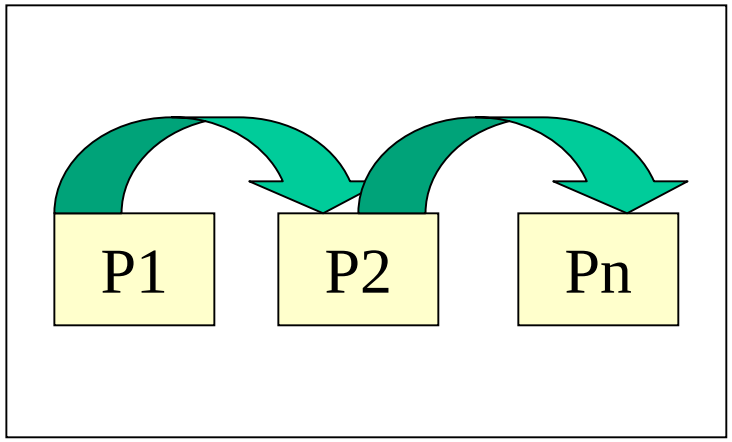
Complex structures



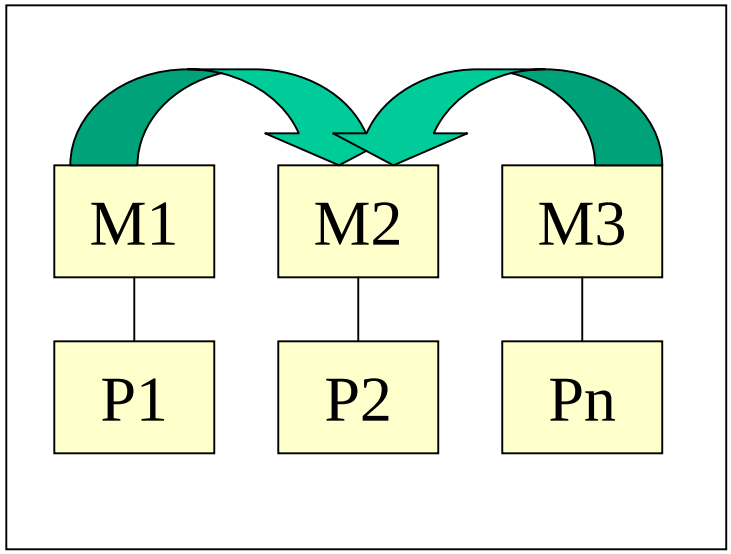
Basic structures: generic typed containers

Storage resources: bytes

Parallel Programming Patterns

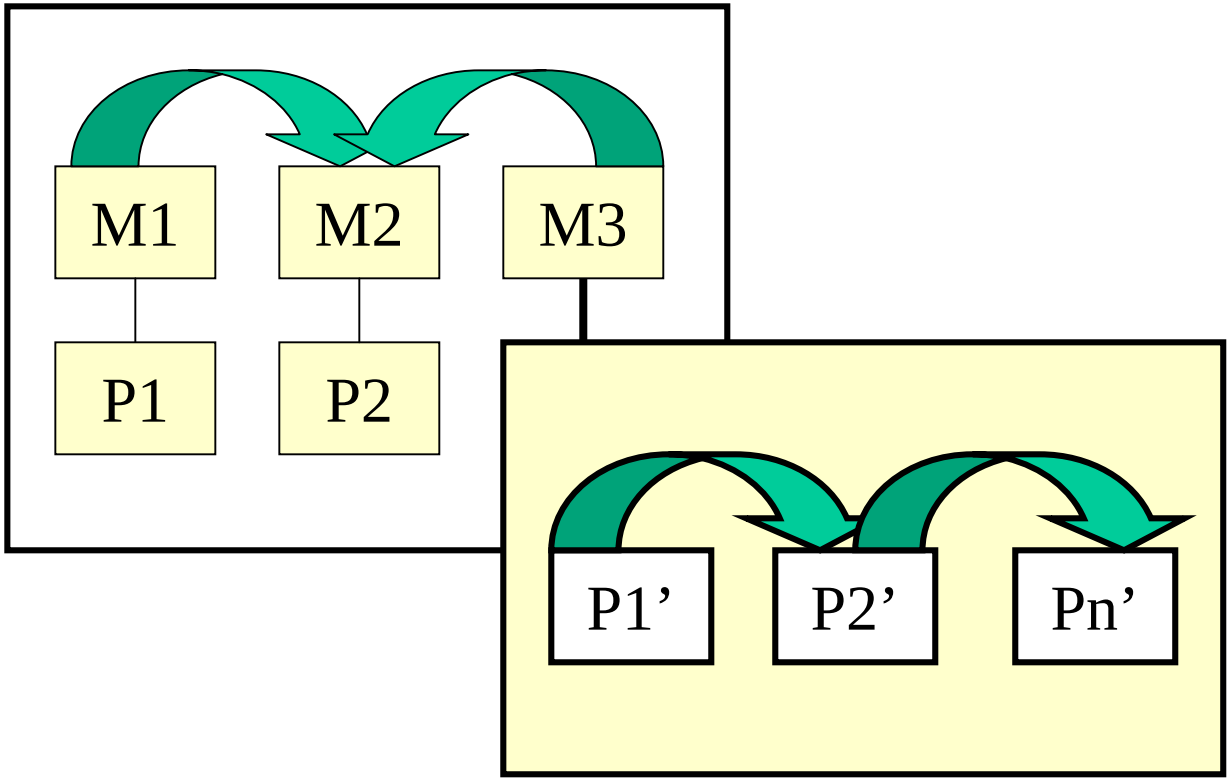


Collective
Synchronous

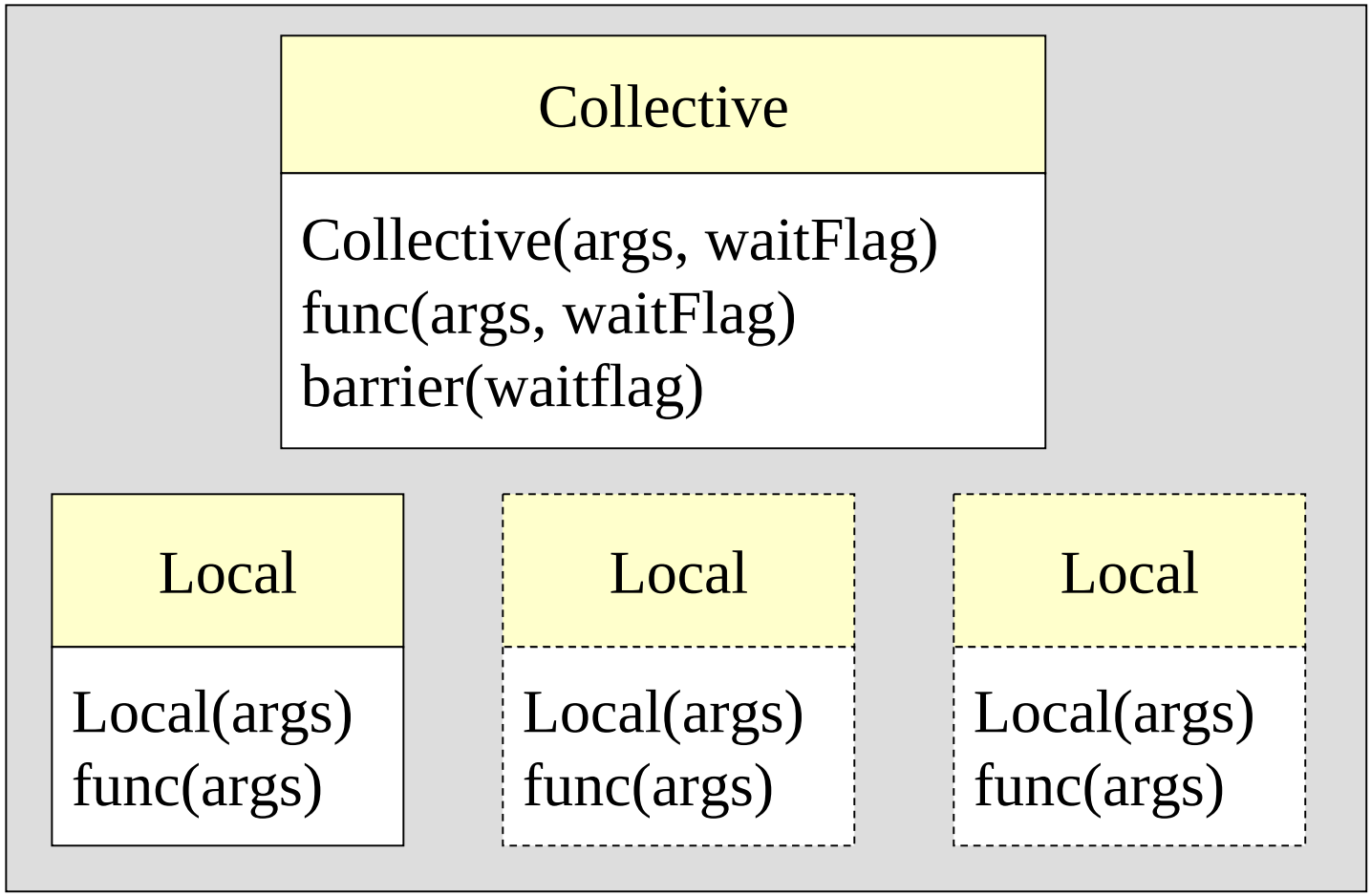


Concurrent
Asynchronous

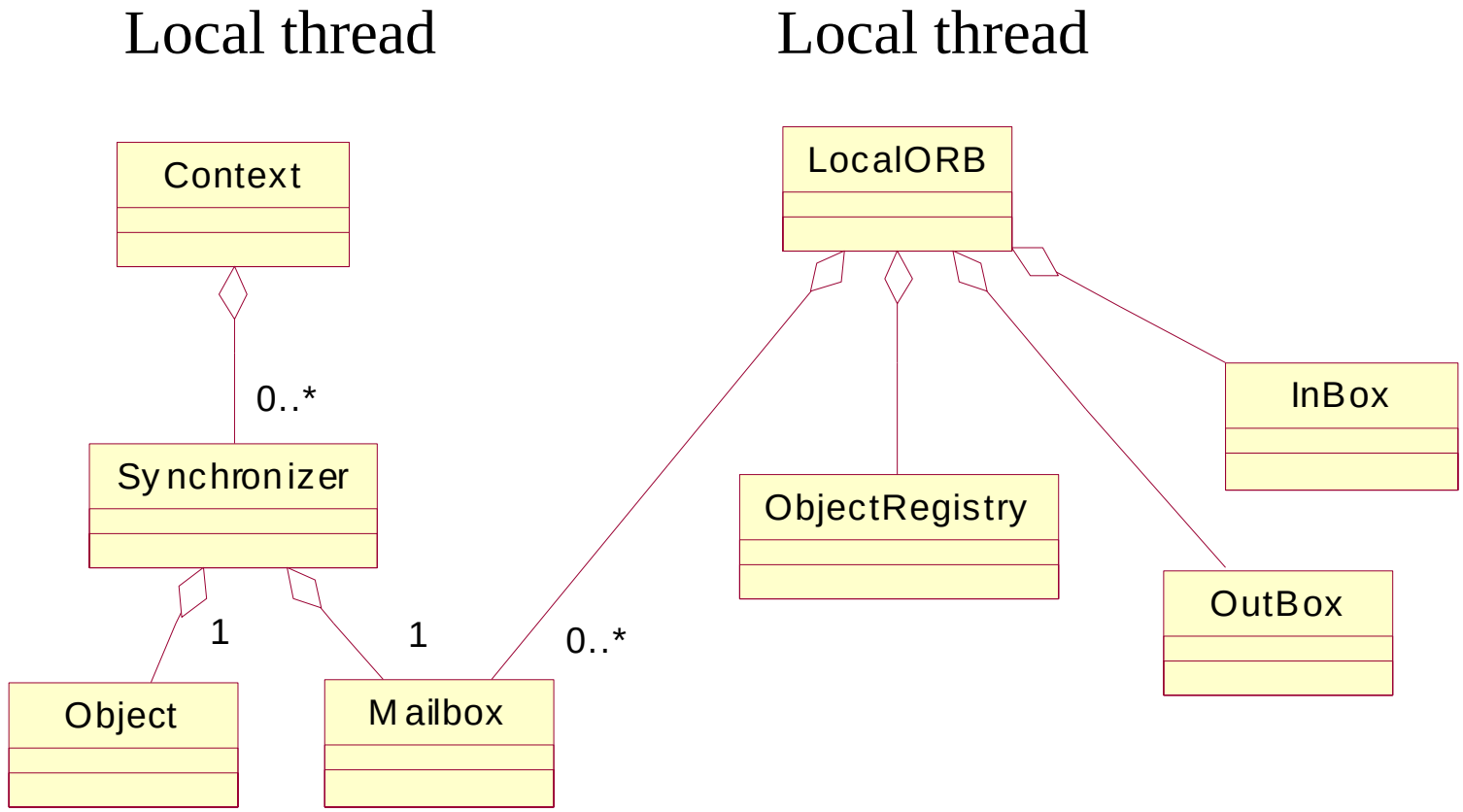
Recursive Parallel Patterns



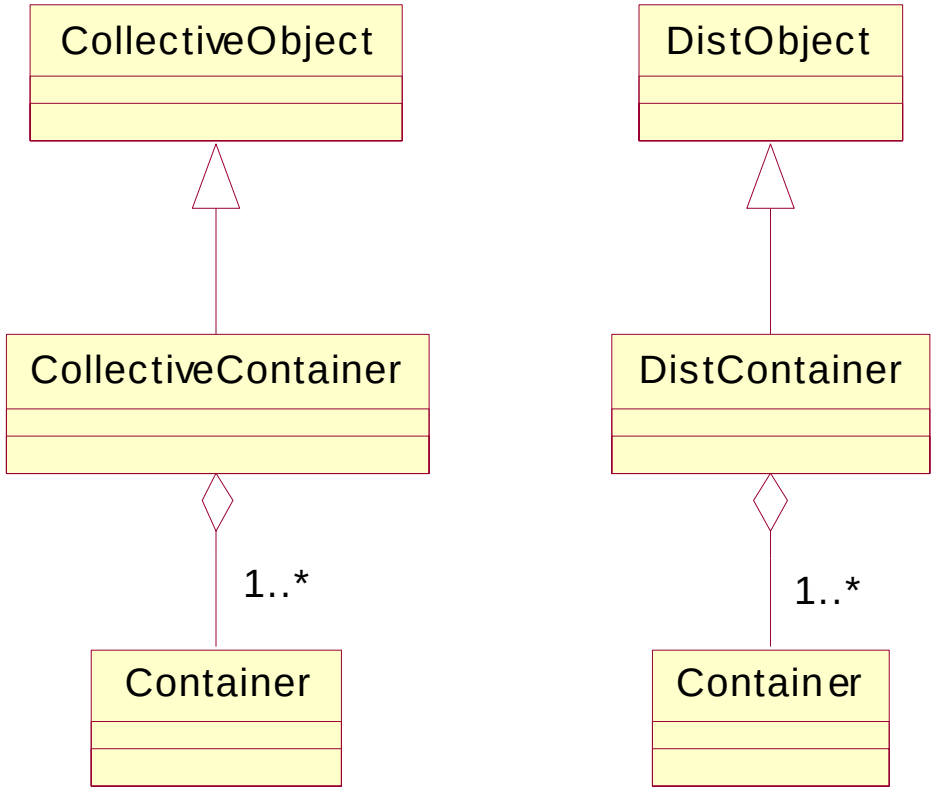
Collective Objects



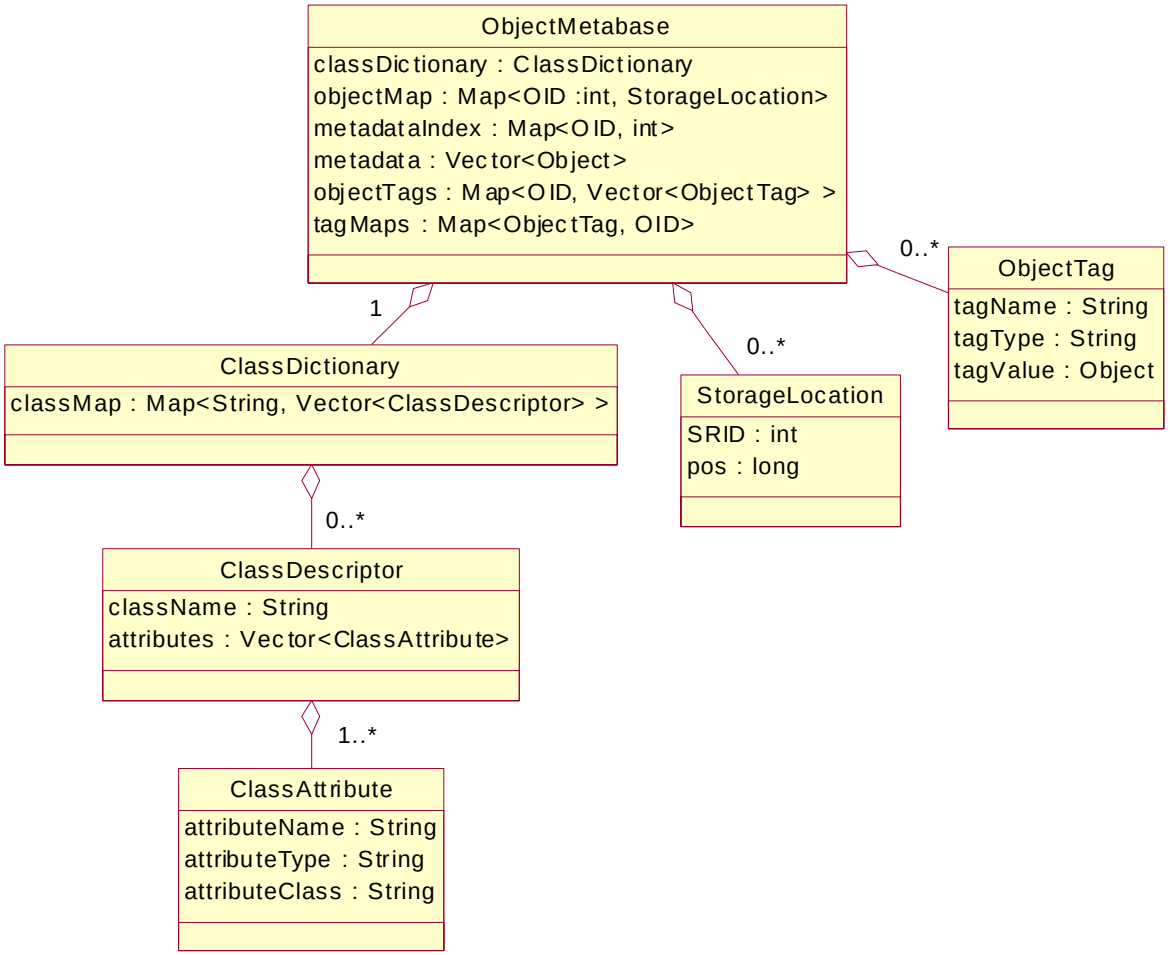
Distributed Objects



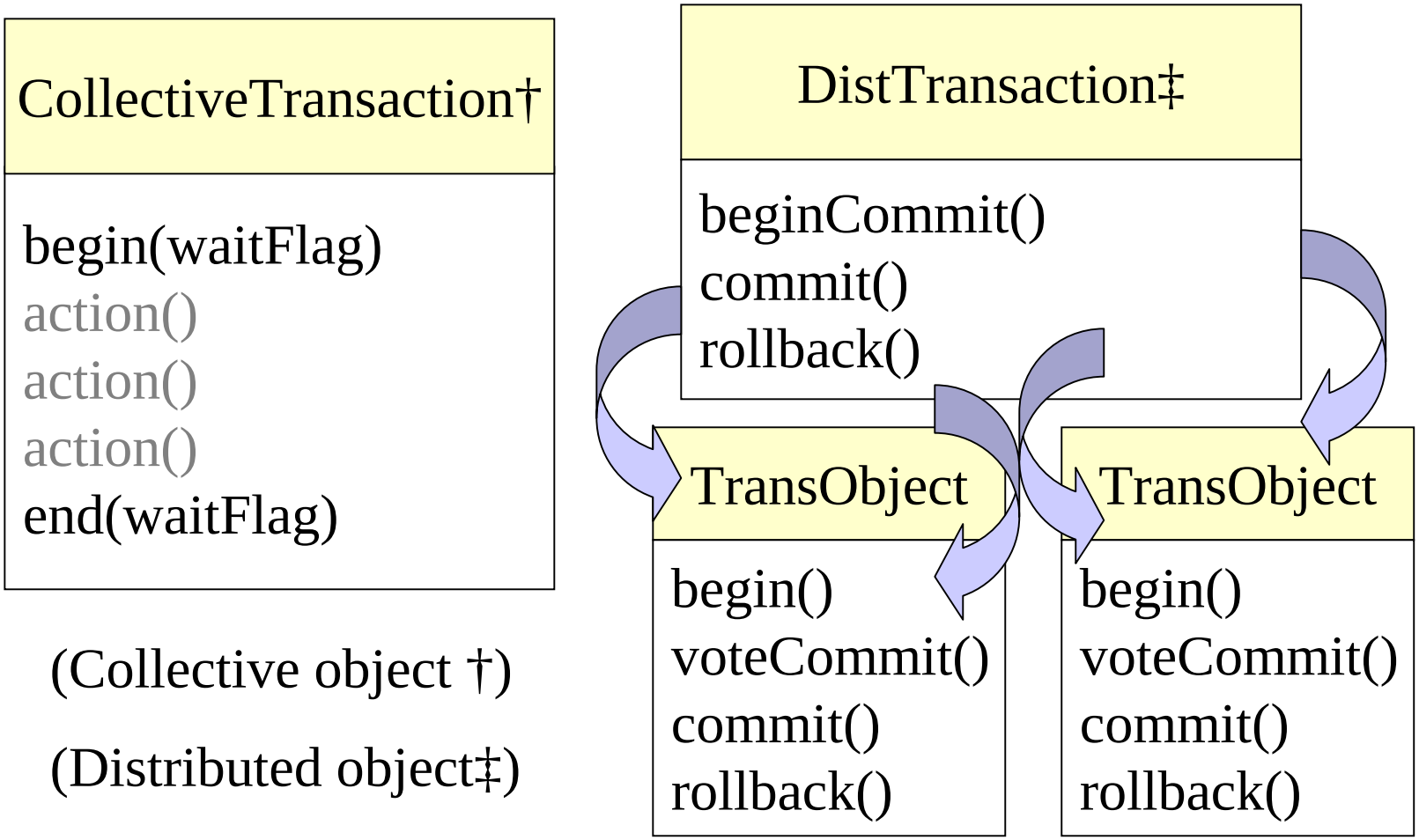
Collective and Distributed Containers



Metadata Abstractions



Transactions



(Collective object †)

(Distributed object‡)

Development Path

- ☑ Exploration of HPC and distributed computing patterns
- ☑ Java prototypes for most pattern classes written (using mpiJava)
- ☑ Beginning integration of parallel pattern classes with storage abstractions
- ☐ Development of transaction model and transaction objects
- ☐ Evaluation and reporting (project exits)