



# Applying Components / Frameworks to the ObjectWeb Persistence Support

Middleware Conference - 01/12/2005

P. Déchamboux (France Télécom R&D)

The present document contains information that remains the property of France Telecom. The recipient's acceptance of this document implies his or her acknowledgement of the confidential nature of its contents and his or her obligation not to reproduce, transmit to a third party, disclose or use for commercial purposes any of its contents whatsoever without France Telecom's prior written agreement.



# Outline



- ▶ **Rationale for investigating components and frameworks**
- ▶ **Open Source for mutualizing efforts & for confronting real use**
- ▶ **Proposal for middleware organisation**
- ▶ **Application of the approach to persistent objects**
  - ▶ Building persistent objects
  - ▶ Managing persistent objects
- ▶ **Evaluation of the approach**
- ▶ **Lessons learned**
- ▶ **Conclusion**

# Rationale for investigating components and frameworks

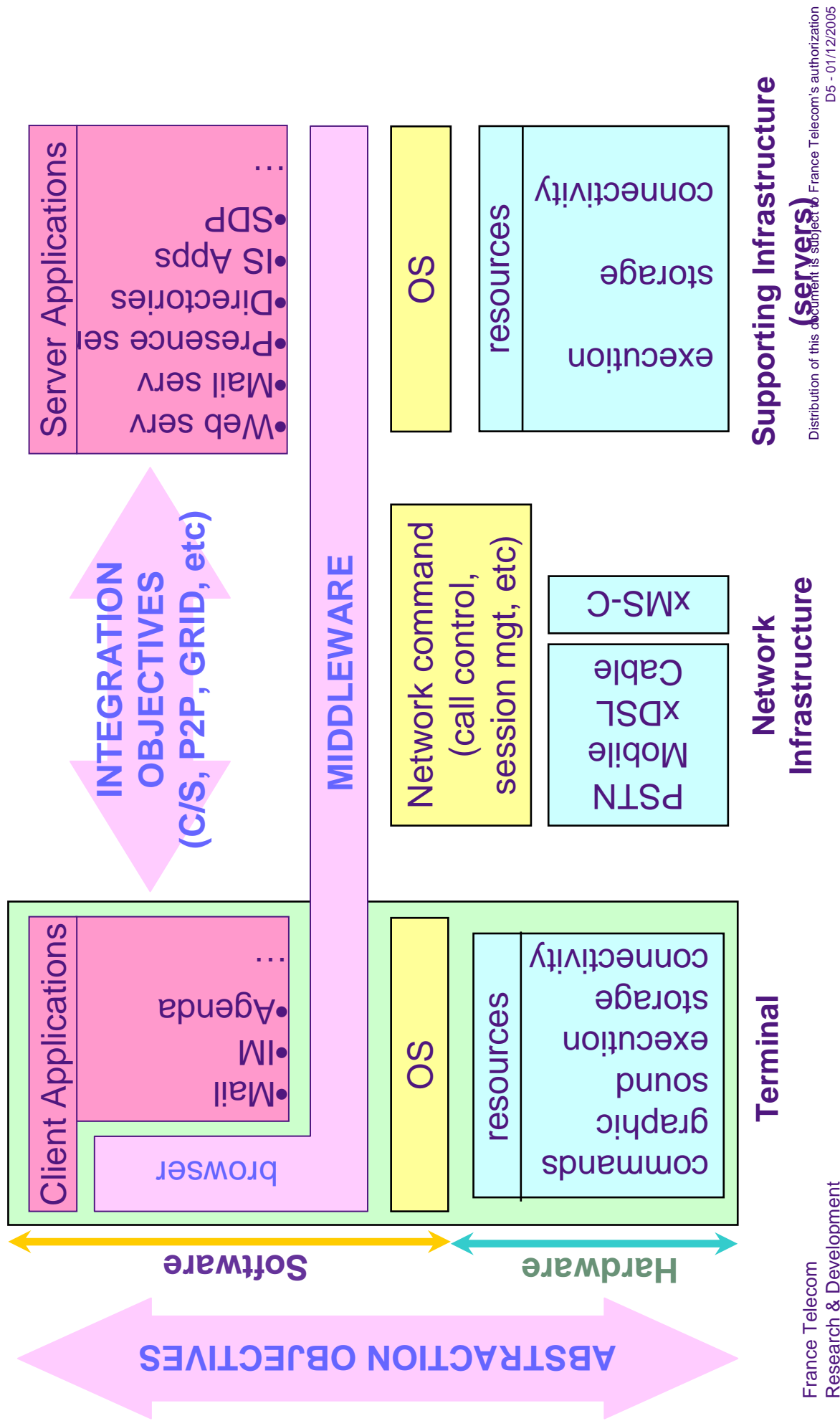


# Application requirements



- ▶ **Complexity increases**
  - A large variety of devices to interconnect (HW & SW heterogeneity)
  - Number of such devices gets bigger (billions of objects to interconnect)
  - A mix of many technical constraints to support (security, scalability, safety, adaptability, timing constraints, etc)
- ▶ **Constraints are stringent on economical issues (investment and operational expenses)**
- ▶ **Requirement for optimizing resources consumption**
  - They may be scarce, they are often expensive
- ▶ **Architecture issues become crucial**
  - No software actor has the capability to master the whole response on its own (integration issues are key)
  - Time to market gets more acute( need to reuse software/operational functions)

# Roles of middleware (the "big picture") &



# Open Source for mutualizing efforts and for confronting real users



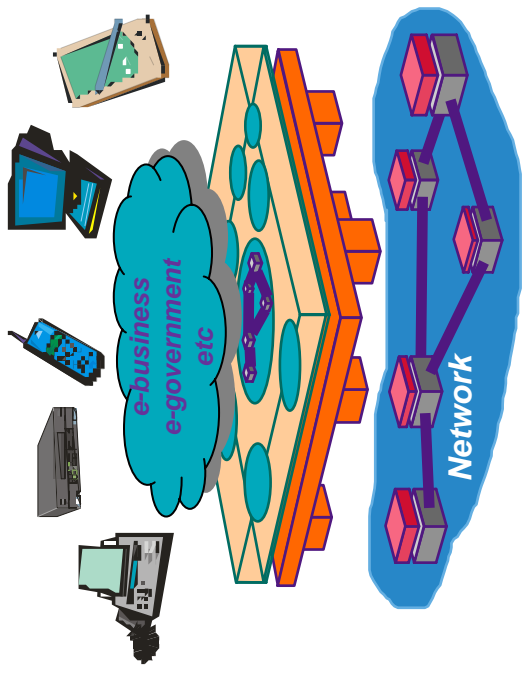
# ObjectWeb: gathering energies to tackle with the whole complexity



▶ ObjectWeb is a community that aims at producing industrial-class open source middleware

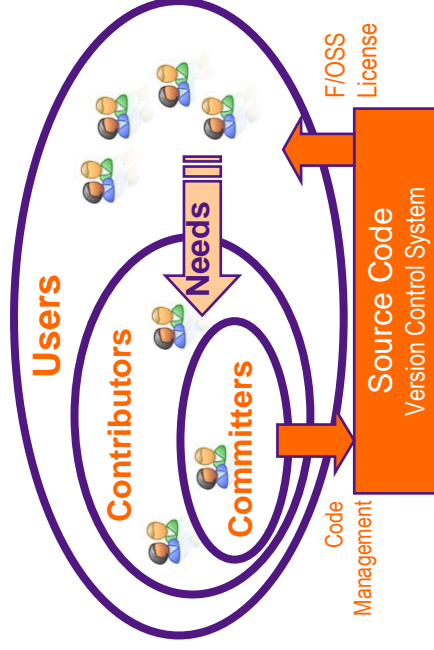
▶ Members are ...

- ▶ Individuals
- ▶ Companies (from SMEs upto groups)
- ▶ Universities & public labs



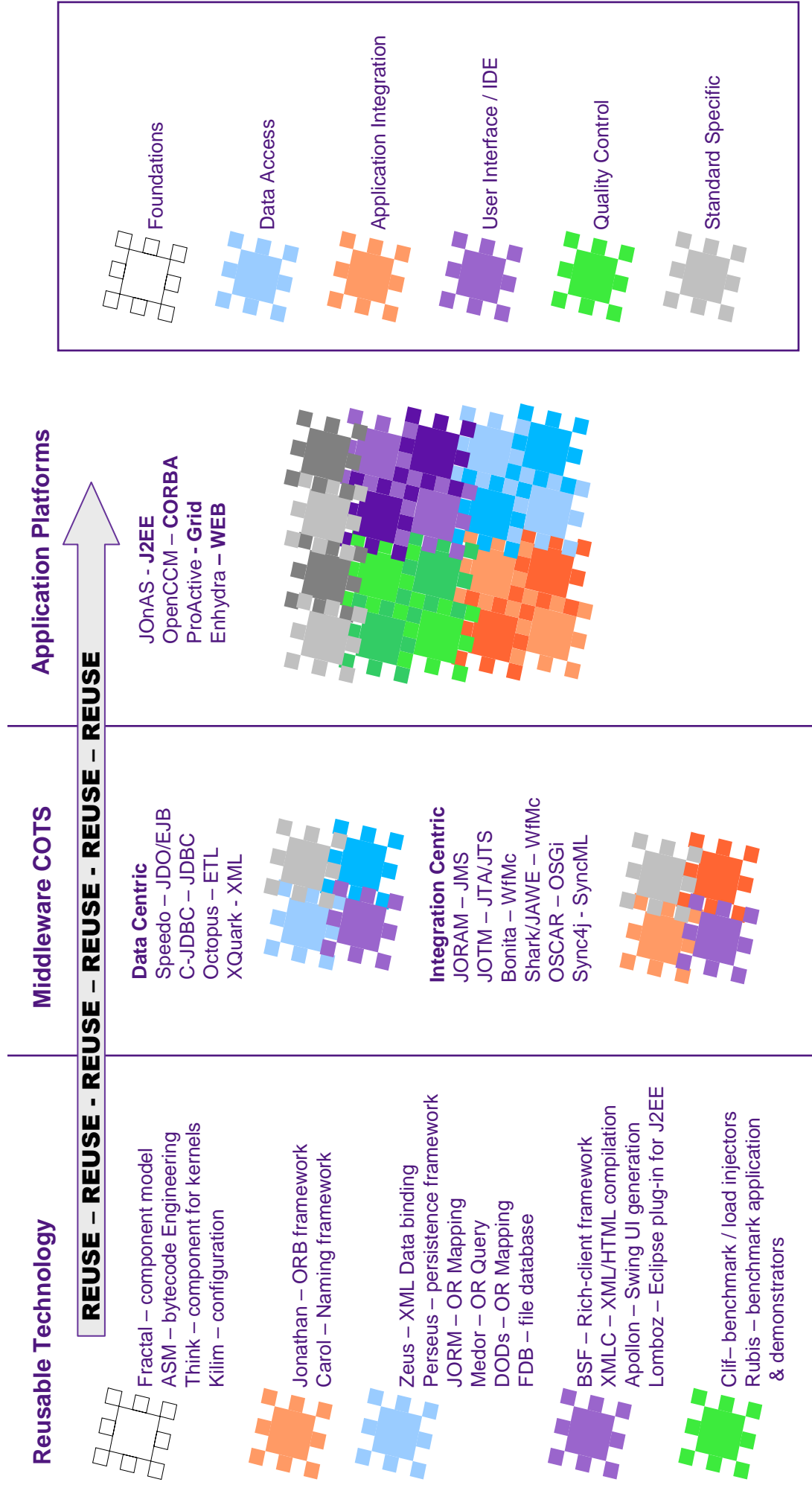
▶ Open-Source as a Process

- ▶ Gathering energies to produce the largest basis for research purpose
- ▶ Vehicle for dissemination
- ▶ Confronting real users to OS products
- ▶ Organising integration





# Component-Based Offer



# Proposal for middleware organisation



# Middleware organisation challenge



- ▶ **Exo-kernel middleware: a view of middleware architecture**
- ▶ **Exo-kernel philosophy**
  - ▶ Open set of infrastructure services (adaptability / extensibility as the main focus)
  - ▶ Applications should be allowed to choose their supporting abstractions
- ▶ **4 strata towards a middleware exo-kernel**
  - ▶ Not software layers
  - ▶ Design principles (e.g., patterns, complex patterns)
- ▶ **Middleware as extensible, configurable component libraries**
- ▶ **Underlies ObjectWeb vision of middleware**

# Exo-kernel middleware strata (1)



## ▶ Stratum #1: Components

- Lightweight, reflective component model
- Middle/coarse-grain component: support for component and assembly description (extensible ADL)
- Fine-grain component (lots of instances): tools for code engineering (static and dynamic code generation and adaptation, e.g. AOP and the like)
- Basis for static and on-line system adaptation and evolution

## ▶ Stratum #2 : Architectural Frameworks

- Pattern-based architectural frameworks for hard recurring issues
- Naming, types and meta-data
- Communications
- Monitoring and failure detection
- Resource management
- Distributed configuration management
- etc

# Exo-kernel middleware strata (2)



## Stratum #3: System services

- P2P indexing and routing
- Asynchronous communication services
- Transactions & Orchestration
- Configuration and Resource Management
- High-availability support
- Persistence support
- Set-oriented query support (a la SQL)
- etc



## Stratum #4: Personalities

- J2EE
- Web Services
- OGIS
- CORBA
- etc

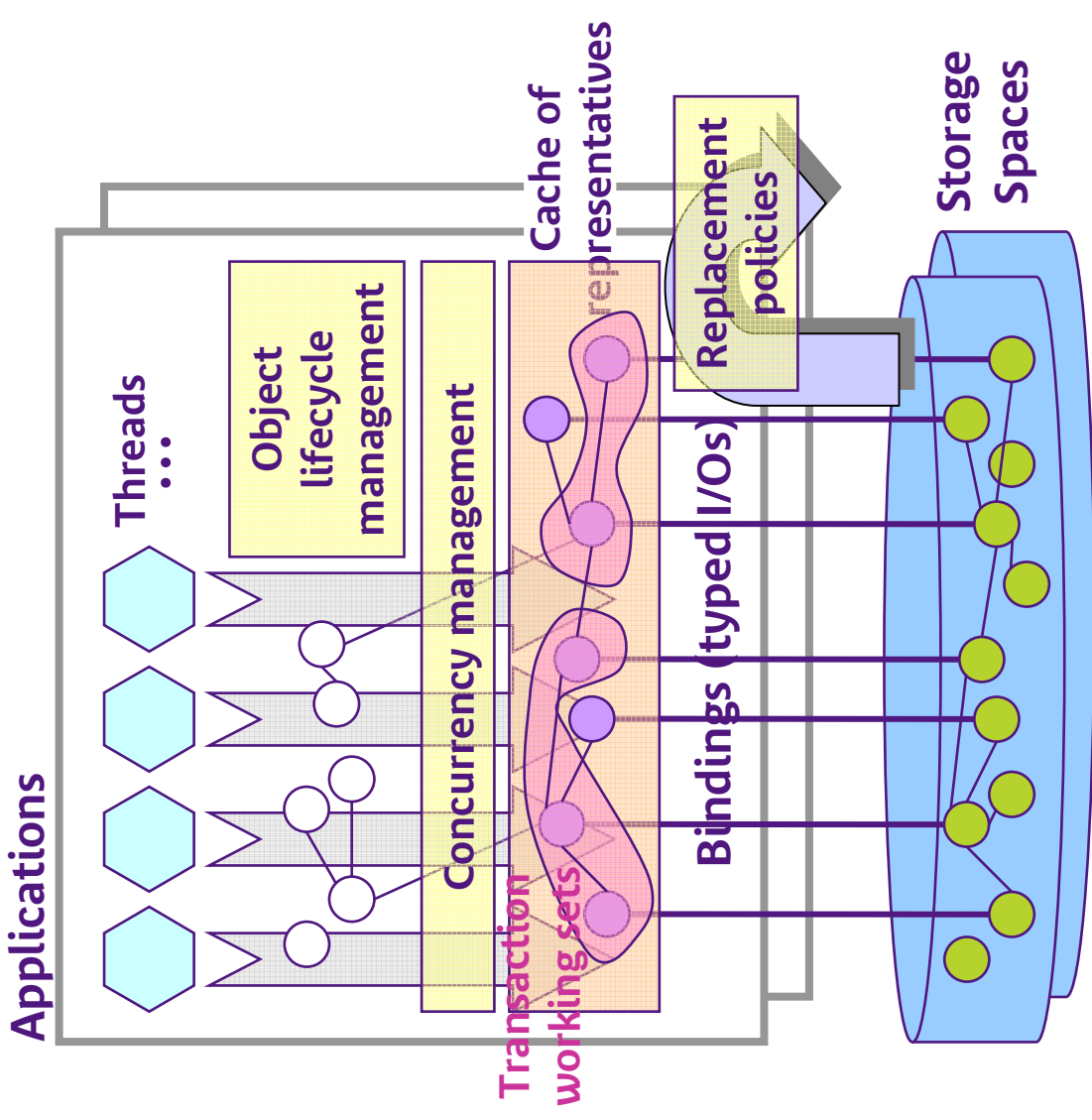
# Application of the approach to persistent objects



# Positioning the problem

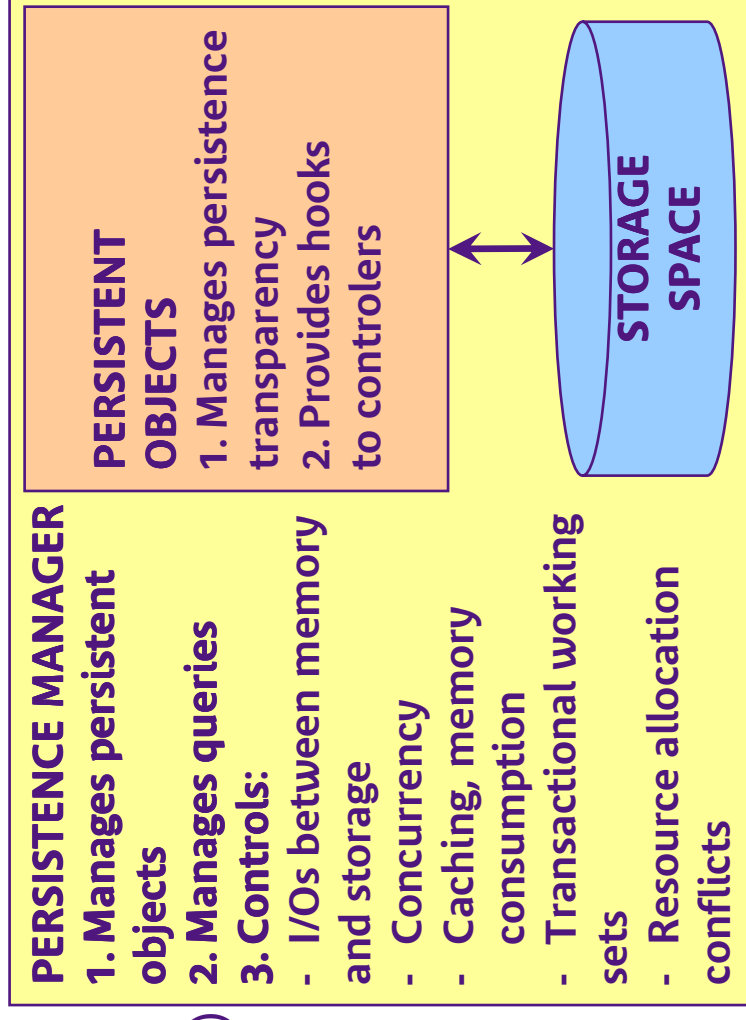


- ▶ Provide transparent storage for business entities
- ▶ Support of mission critical constraints (e.g., tx behaviour)
- ▶ Manage resource consumption
- ▶ A complex support in itself
  - Many functions: typed I/Os, queries, caching, concurrency, transactions, etc
  - Complex interactions
  - Policy related issues
  - An intensive standardisation activity (e.g., EJB, JDO, SDO, etc)



# Applying the approach to Java persistence: the Speedo & case

- ▶ **Apply component technologies**
  - Transparency support (code injection/modification through **ASM**)
  - Persistence manager organisation (**Fractal** architecture definition, assembly)
- ▶ **Define frameworks**
  - **JORM**: a Mapping framework (storage management supporting typed I/Os)
  - **MEDOR**: Query framework
  - **Perseus**: Persistence framework (relationships between managers of I/Os, concurrency, tx, cache, etc)
- ▶ **System service = persistence**
- ▶ **Personality support (EJB2, JDO2, EJB3)**



# Building persistent objects

France Telecom  
Research & Development

Distribution of this document is subject to France Telecom's authorization  
D16 - 01/12/2005



# Which approach for building persistence capable objects?



- ▶ **Need fine-grain code engineering**
  - Sometimes recommended (e.g., JDO promote byte code enhancement)
  - Code manipulation done using **ASM** engine (visitor / decorator patterns)
- ▶ **Need to intercept**
  - Accesses to class variables
    - Differentiating read/write accesses
    - Dating latest accesses (time stamping)
  - ⇒ 1. Use of programmed access to variables (JavaBean pattern, e.g., EJB2, Hibernate): alter transparency & may force encapsulation policies
  - ⇒ 2. Use of code manipulation engine (byte code engineering, e.g., ASM)
- ▶ **AOP was a candidate but...**
  - Usually too expensive to intercept at this granularity
  - Just support addition of code
  - Cannot reorganize code, cannot support "synthesized point cut"

# Transparently manage persistent objects

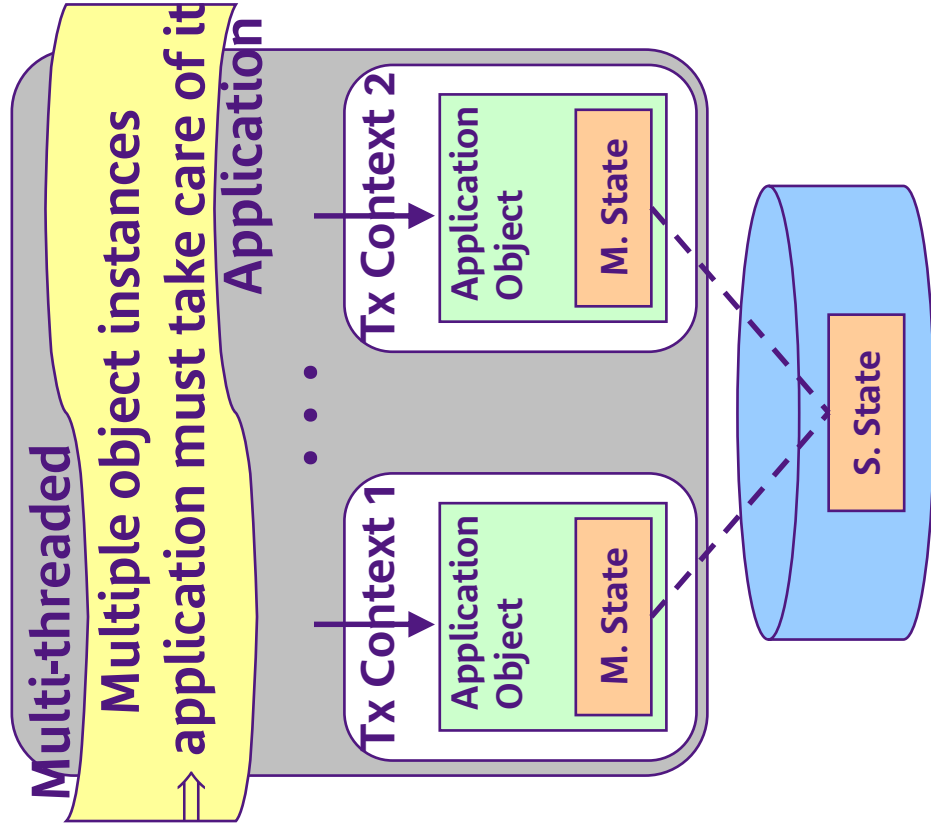


- ▶ **Orthogonal persistence: Using persistent objects like any other object at language level**
- ▶ **Adding persistence**
  - ▶ Some of the variables represent persistent information
  - ▶ Need to manage synchronisation/transformation between two different representations
    - Storage object state (further named S. State)
    - Memory object state (further named M. State)
- ▶ **Conflicting constraints on memory model**
  - ▶ Concurrent accesses require isolation between transactional contexts (different states for the same application object)
  - ▶ Full transparency requires a unique application object representing persistent information as long as it stay in memory

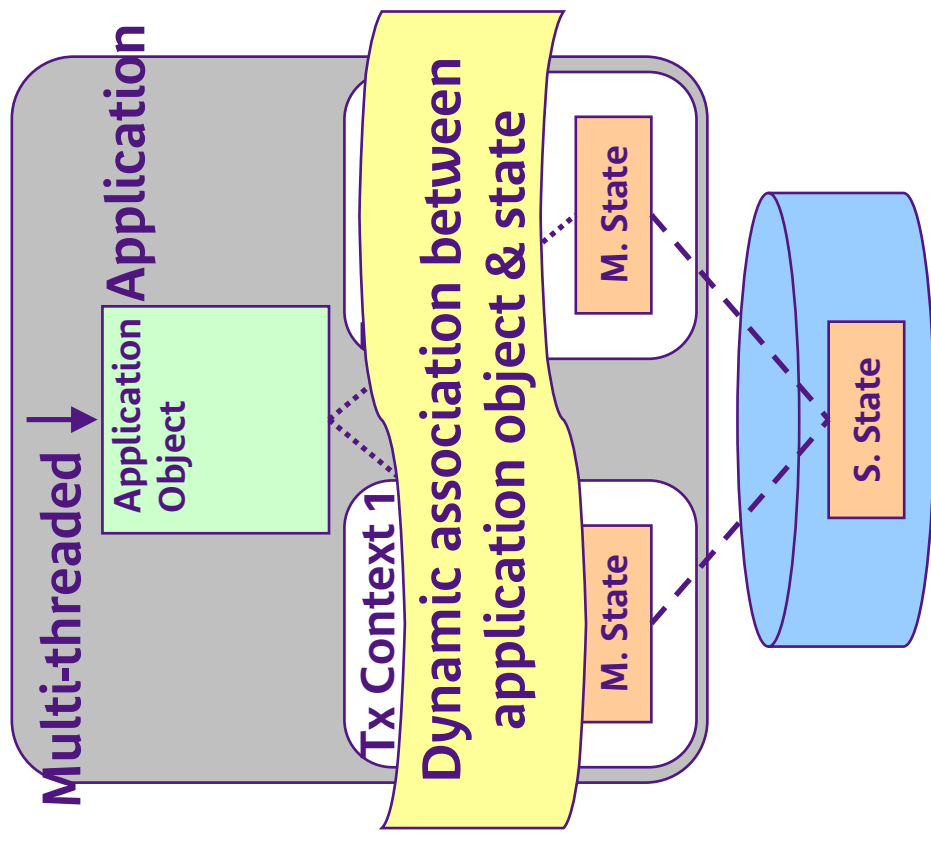
# Two main solutions to the conflict



## Multiple Application Object Solution (MOA solution)



## Single Application Object Solution (SOA solution)



# Dealing with the tradeoff



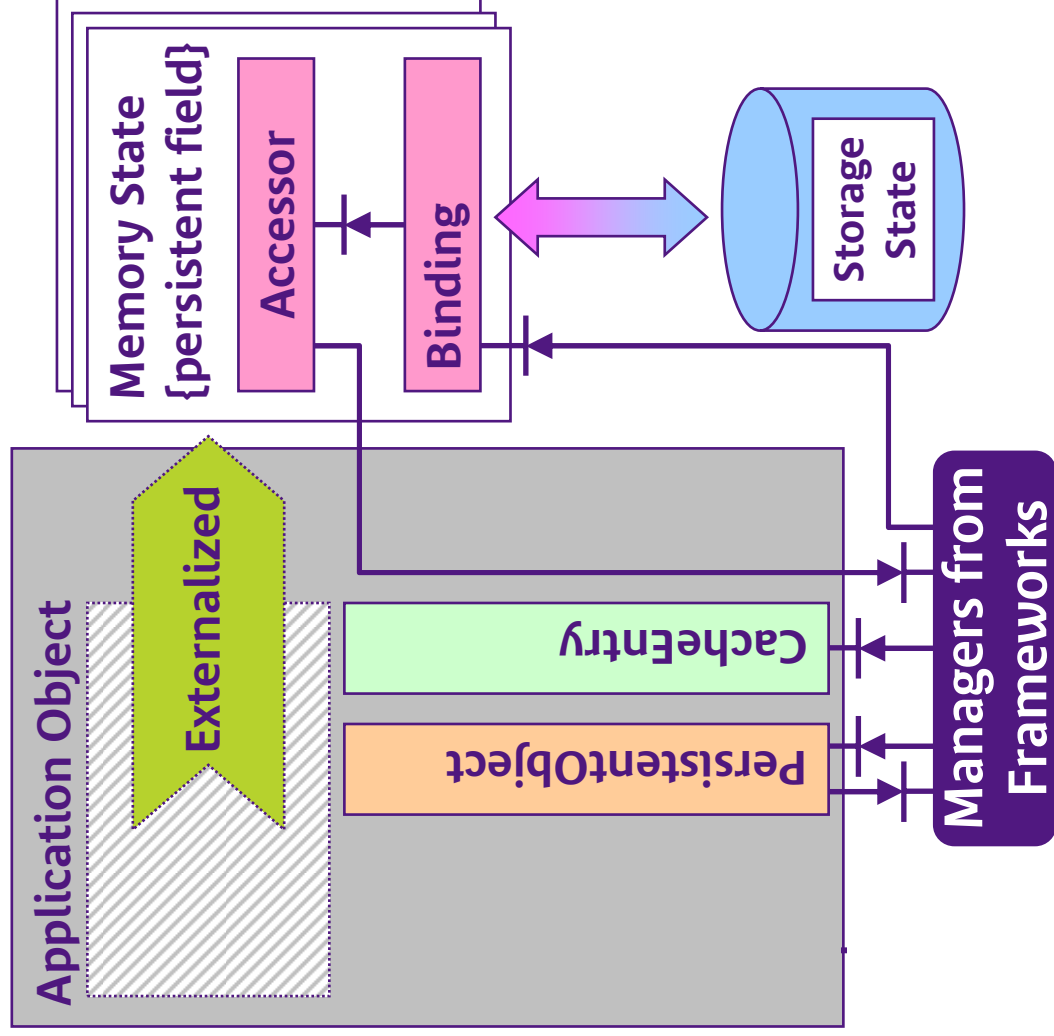
## ▶ MAO Solution

- ▶ Reduce transparency: most of current standards and solutions only mandate this level (i.e., JDO2, EJB3, Hibernate, etc)
- ▶ Easy to implement
  - Application object can be mostly reused as is
  - No re-entrance to be managed on memory object instance

## ▶ SAO Solution (Speedo choice)

- ▶ Total transparency wrt the Java programming model
- ▶ Requires heavy reorganisation of code
- ▶ Give access to advanced memory management (e.g., keep consistent object working set into memory while removing memory states)

# Architecturing persistent objects



Using the state pattern for managing **M. States**



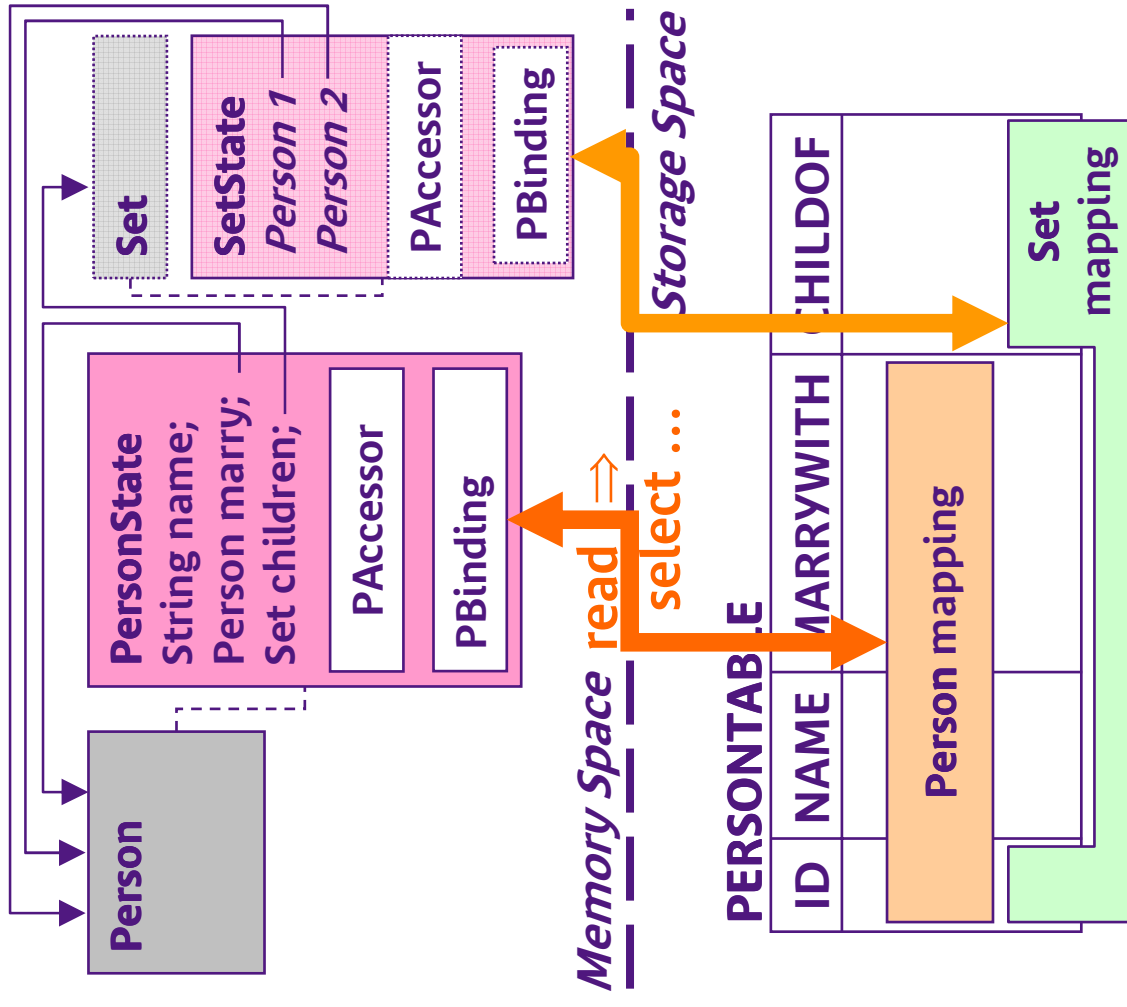
Injecting instance-related code pieces as hooks to other functions

- **Binding** for managing I/Os between **M. State** and **S. State**
- **CacheEntry** for managing in memory object cache and information for replacement policies
- **PersistentObject** for activating "synthesized point cuts" (intention to read or write attached to methods)
- **Personality-specific code** as required by personalities (EJB, JDO, etc)

# JORM mapping FW: setting up bindings



- ▶ "export/bind" pattern (same as for distributed object communication)
- ▶ Names used for identifiers and references
- ▶ Bindings as "stubs"
  - Bind m.state to its storage representative
  - A **name** identifies the storage object bound to the memory object
  - Support APIs to control binding & I/O
    - bind, unbind, export, unexport
    - read, write, exist
  - Manage translations between memory and storage representations



# Managing persistent objects



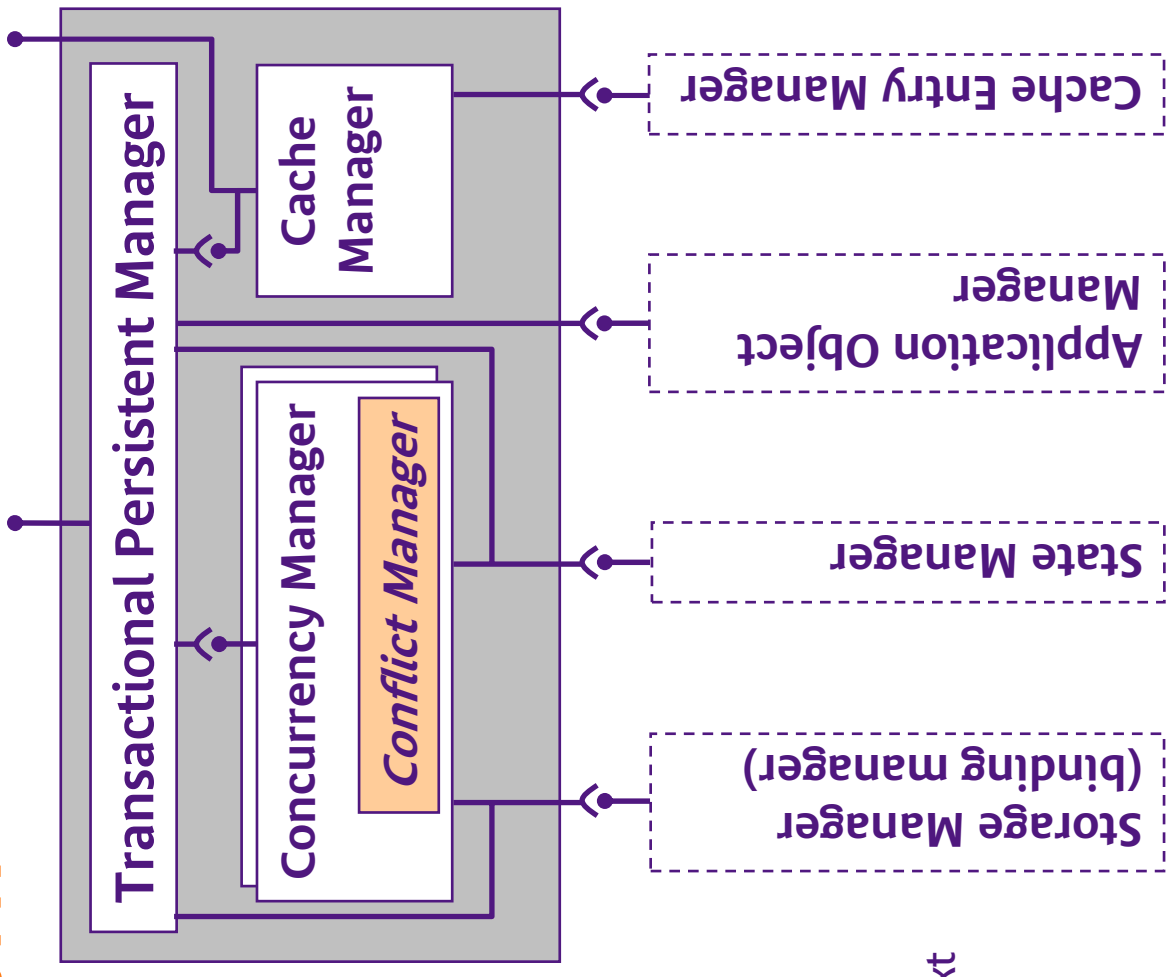
# The Perseus Framework



- ▶ **Provision of functions to engineer persistent systems**
  - Storage / memory I/Os control
  - Caching management
  - Concurrency management
  - State management
- ▶ **Framework applied to dedicated persistent systems**
  - Define a particular organization of functions and their interactions
  - Transactional Persistent System (TPM)
    - Components independent of usage context

⇒ pre-assembled into the TPM

  - Dependencies to component that should adapt to usage context



# Cache Manager

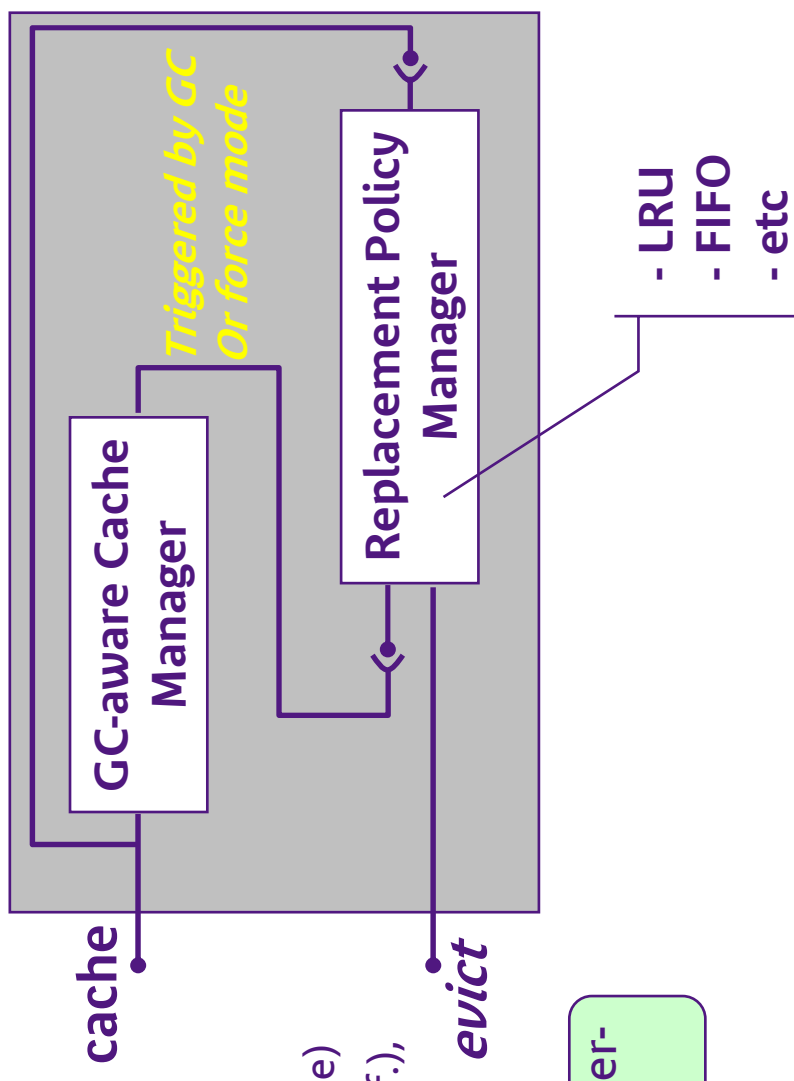


- ▶ Allow control over memory consumption
- ▶ Pattern involving two sub-components
  - Cache (manage association table)
  - Replacement Policy
    - Decide which cache entry to remove)
    - Eviction based on GC (e.g., weak ref.), timer –based, explicit requests, etc

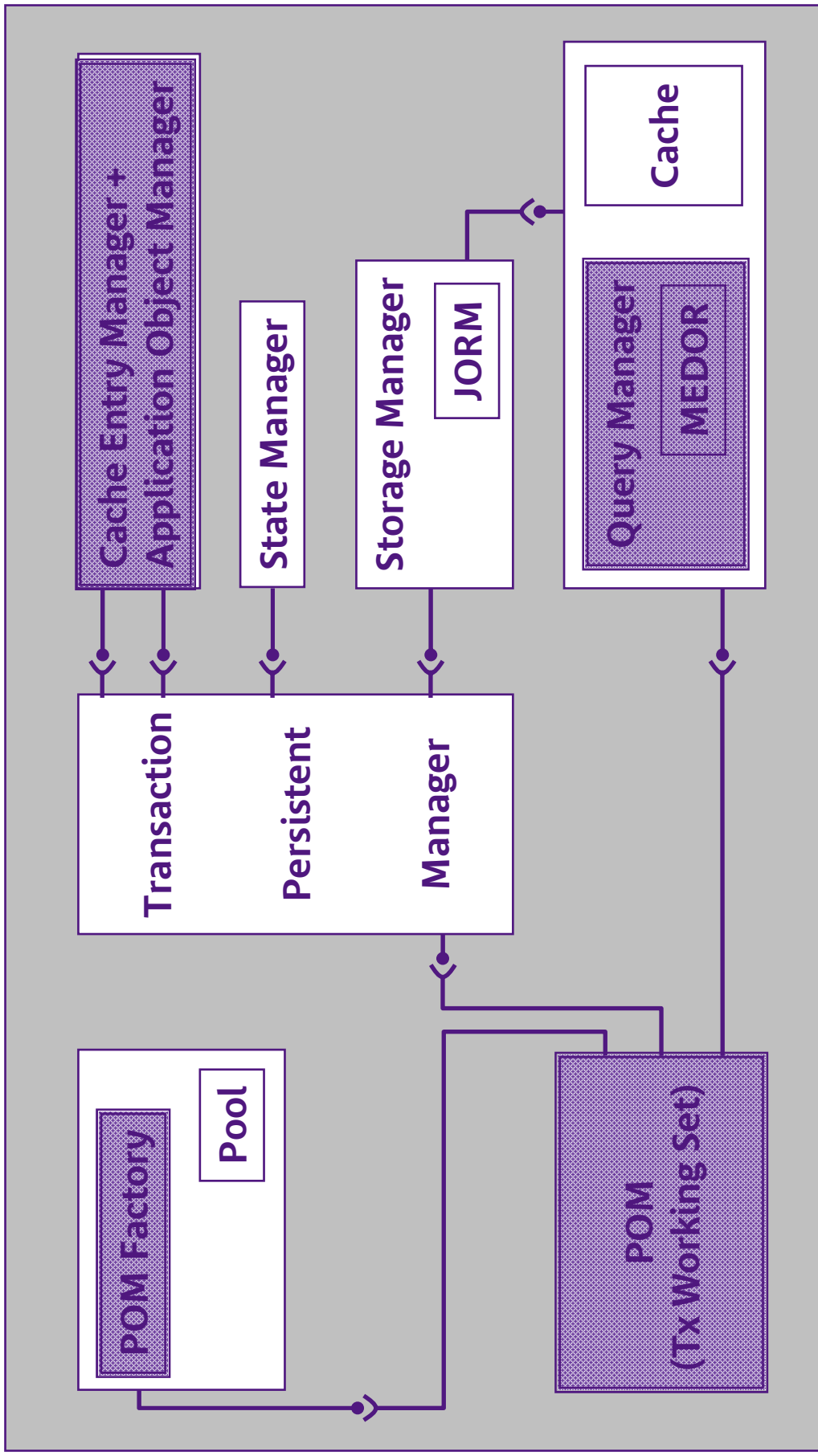
## ▶ Two declinations into Speedo

- Application Objects Cache (for inter-transaction caching)
- Compiled Queries Cache

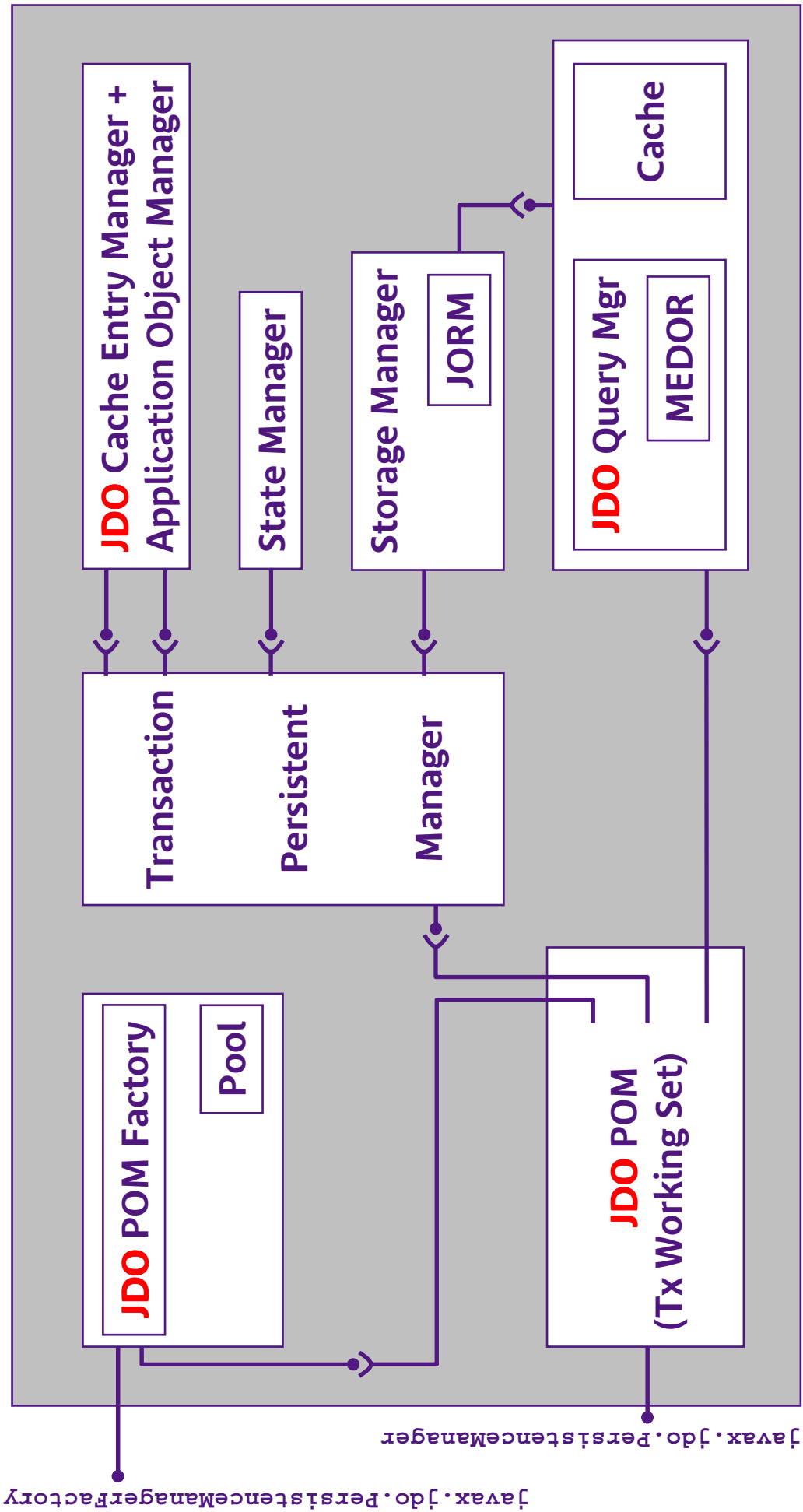
## Example of the Application Object Cache



# The Speedo Framework



# The SpeedoJDO Personality



# Evaluation of the approach & Conclusion



# Evaluation results



## ▶ Code Quantity (lines of code)

- ▶ JORM 62.000
- ▶ MEDOR 31.000
- ▶ Perseus 16.400
- ▶ Speedo 45.000
- ▶ Personalities
  - Speedo JDO 18.000
  - JOnAS EJB2 CMP 25.000

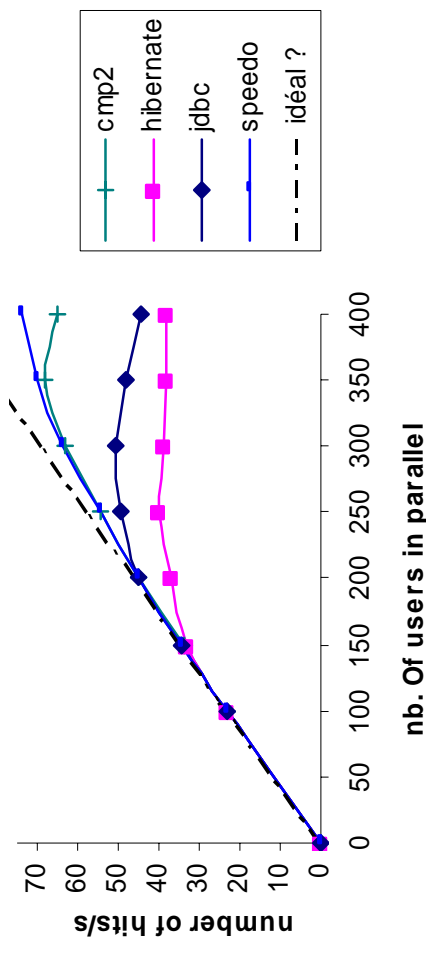
## ▶ Level of Reusing

- ▶ JOnAS EJB2 CMP
- JORM, MEDOR 22 %
- New code
- ▶ Speedo Personalities
  - JORM, MEDOR, Perseus, Speedo
  - New code JDO 10 %
  - JDB3 *under dev.*

## ▶ A few Performance Insights

- ▶ EJBOO app: a typical IS application managing product catalogs and orders
- ▶ Test case: browsing products and look at details (mainly read access)

Performance Comparison for READ access



# Lessons learned



- ▶ **Very helpful to have a support for dealing with architecture issues**
- ▶ **Designing is a complex task especially with complex software infrastructure**
- ▶ **Architecture issues are moving (trade-off is changing from case to case)**
  - ▶ Because of constraints that differ from one usage context to another
  - ▶ Because it is difficult for architects to find the good trade-off at first time

# Conclusion



- ▶ **Propose an approach based on components & frameworks**
- ▶ **Confront the approach to real problems**
  - Persistence (no distribution yet, except for distributed concurrency)
  - Load injection framework / CLIF (involves distribution)
- ▶ **Use Fractal for mastering architecture**
  - Define assembly step by step
  - Identify adaptable sub-components or dependencies
  - Rely on reflexion to exhibit management capabilities (i.e., automated Mbean provisioning)
- ▶ **Improve reuse**
  - Clearly identify adherence
  - Remove adherence due to configuration code
- ▶ **Performance can be "as good as" non component-based code**

# Thank you!!!



To "persistence" folks...

**S. Chassande, A. Lefebvre, Y. Bersihand**

To Fractal folks...

**E. Bruneton, T. Coupaye, B. Dillenseger, N. Rivière, J.B. Stéfani**

**and many others from Bull, INRIA, IMAG/LSR, Redhat and  
France Télécom**