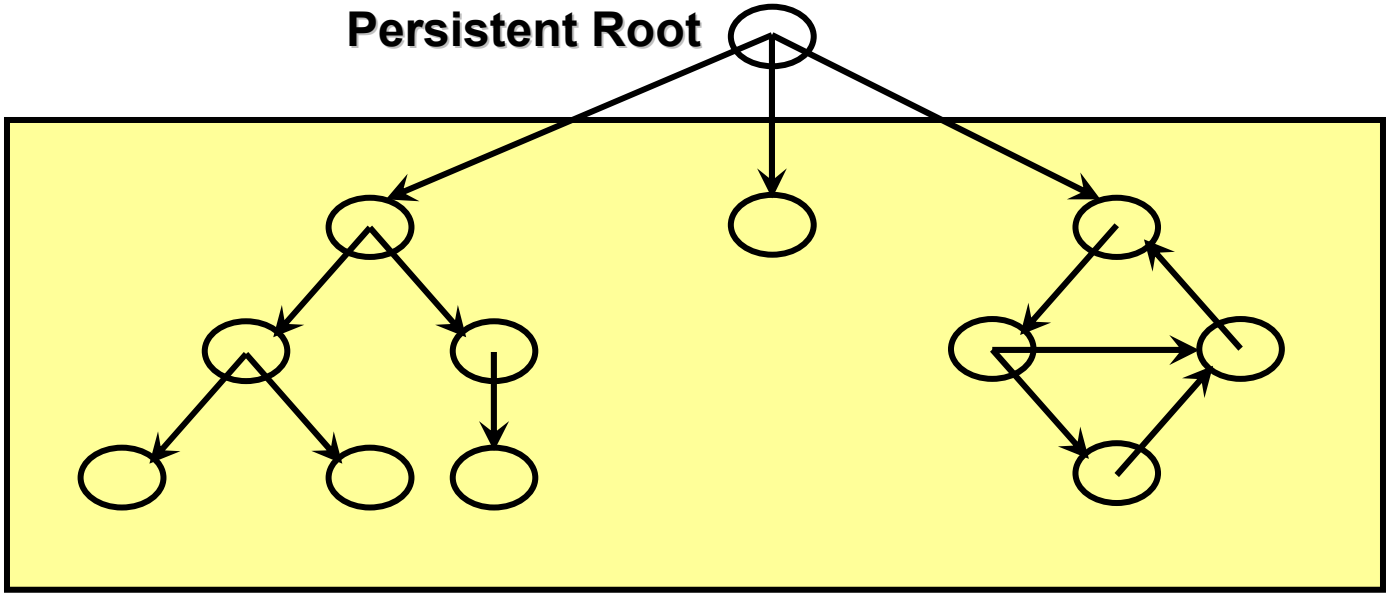
Lazy Modular Upgrades in Persistent Object Stores

Barbara Liskov
MIT CSAIL
October 2003

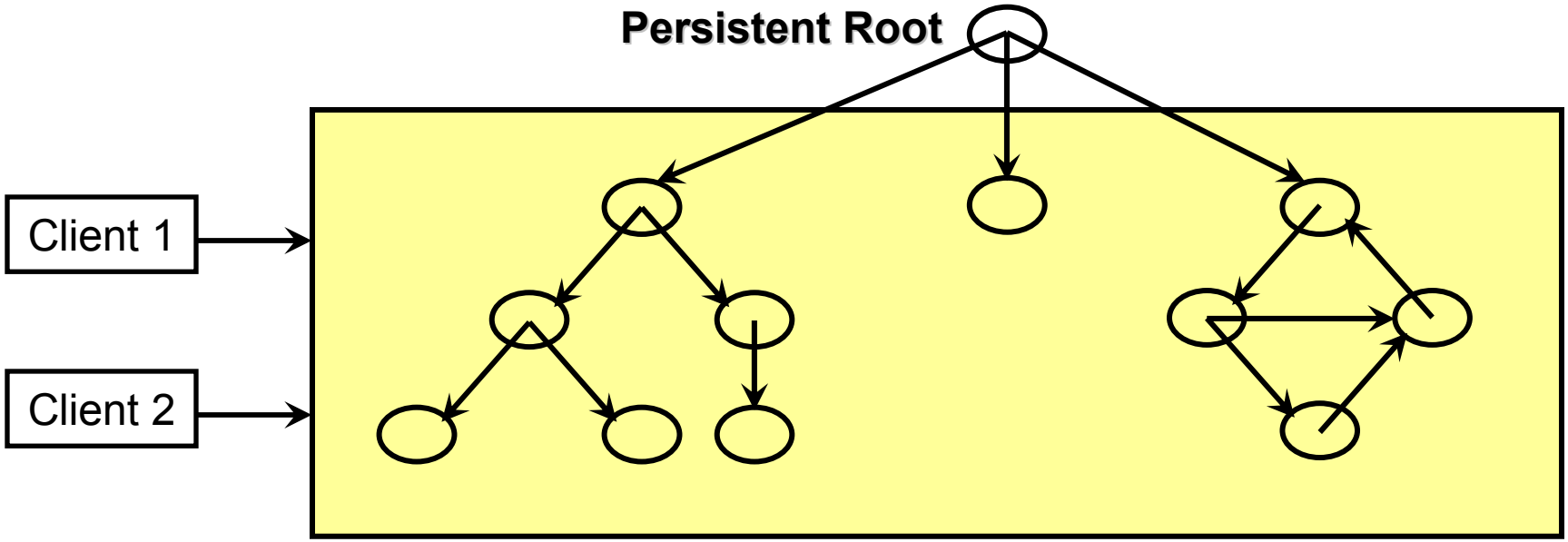
Persistent Object Store

- Stores objects with methods
  - Objects belong to classes
  - Classes implement types



Transactions

- Objects are accessed within transactions
  - Transactions mask concurrency and failures



Upgrades

- Upgrades are needed to
  - Correct errors
  - Improve performance
  - Meet changing requirements

Outline

- Defining upgrades
- Upgrade execution
- Upgrade modularity conditions
- Performance

Defining Upgrades

- Upgrade must preserve persistent state
  - E.g., set implementation changes from vector to hash table

- A class-upgrade is

  <old-class, new-class, TF>
- TF: old-class → new-class
  - TF changes representation of objects
  - System preserves identity

Completeness

- Upgrades can be
  - Compatible
  - Incompatible
- An upgrade is a set of class-upgrades
  - must contain all class-upgrades needed to maintain type correctness

System executes Upgrades

- Requires transforming all old-class objects

- Goal: don’t interfere with applications
  - Don’t stop the world
- Goal: be efficient in space and time
  - Don’t copy the database or use versions
- Goal: be expressive
  - Don’t limit expressive power of TFs

Solution: Lazy, Just in Time

- Applications continue to run
  - Objects are transformed just before first use
- Later upgrades run in parallel with earlier ones
  - If x has two pending transforms, they run in upgrade order

How System Works

- When application accesses x
  - Interrupt the application
  - Run earliest pending transform for x
  - Each transform runs in its own transaction
- Application continues after transform commits

- Transforms can be interrupted too

Example

…; U1; … TF(x); A1; … TF(y); A2; …

U1 is installed

A1 starts to run, accesses x

TF(x) runs and commits

A1 continues and commits

A2 starts to run, accesses y

TF(y) runs and commits

A2 continues and commits

Example

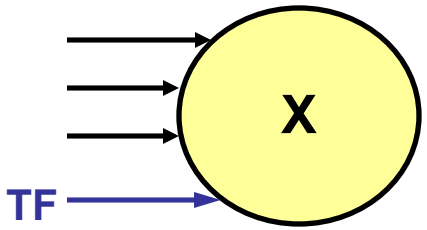
…; U1; … TF(x); A1; … TF(y); A2; …

U1 is installed
A1 starts to run, accesses x
TF(x) runs and commits
A1 continues and commits
A2 starts to run, accesses y
TF(y) runs and commits
A2 continues and commits

Problem: suppose TF(y) accesses x

Modular Reasoning

- Want to support modular reasoning:
  - Programmer can reason about TF as if it were an extra method of the old-class
  - Programmer can assume same old-class interfaces and invariants

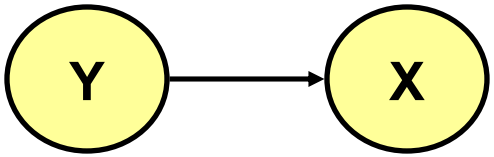


Desired Semantics

- Upgrades appear to run when installed
  - Serialized before all later application transactions
- Upgrades appear to run in upgrade order
- Within an upgrade, transforms run as if each was the first to run

Order within an Upgrade

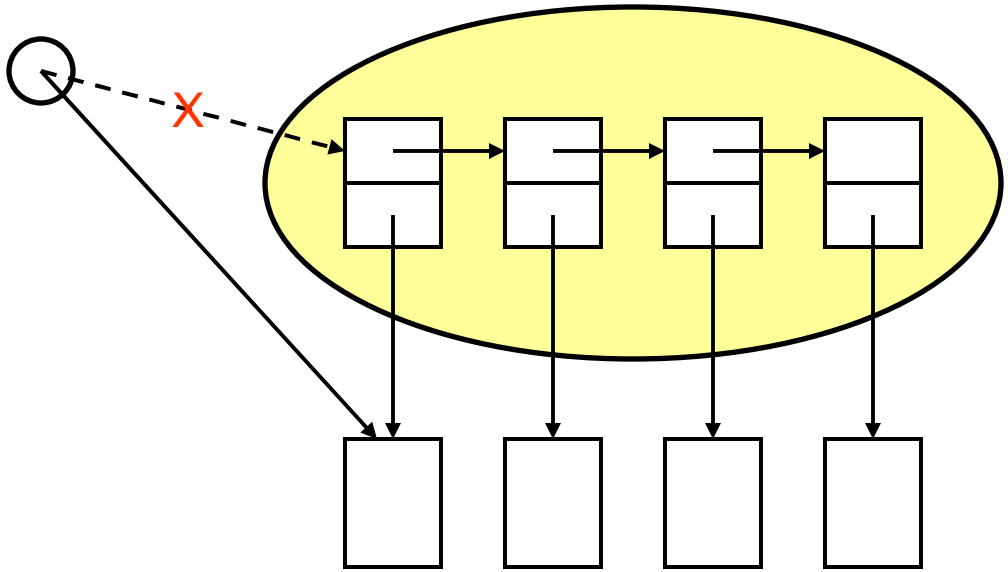
- Consider x and y due to be upgraded



- If TF(y) uses x (transitively) then if [TF(x); TF(y)], this must have same effect as [TF(y); TF(x)]

Ensuring Correct Behavior

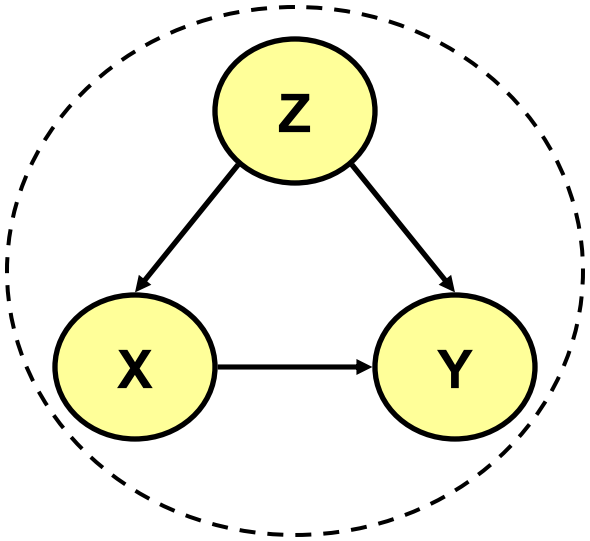
- Based on encapsulation: An object must encapsulate every object it depends on



- A TF is well-behaved if it uses only encapsulated sub-objects

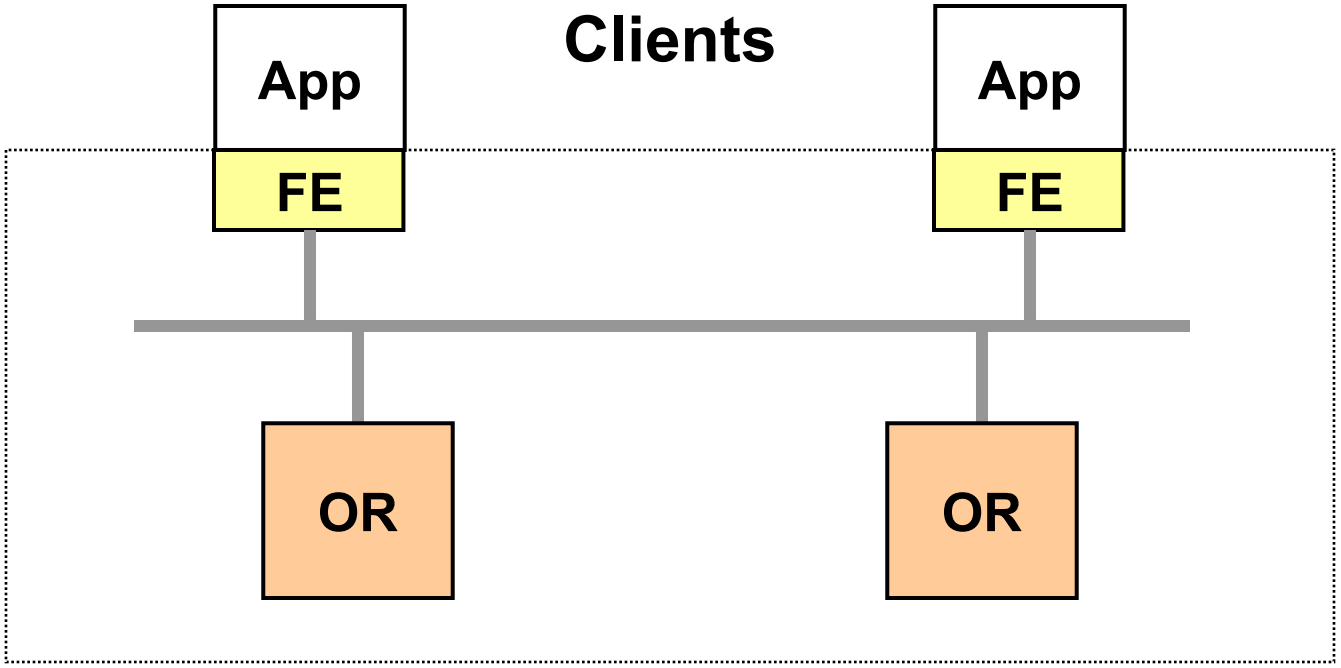
Approach

- Analyze TFs
- Usually they will be well-behaved
- Otherwise notify user
  - User can use triggers to control order
  - Or, we maintain versions

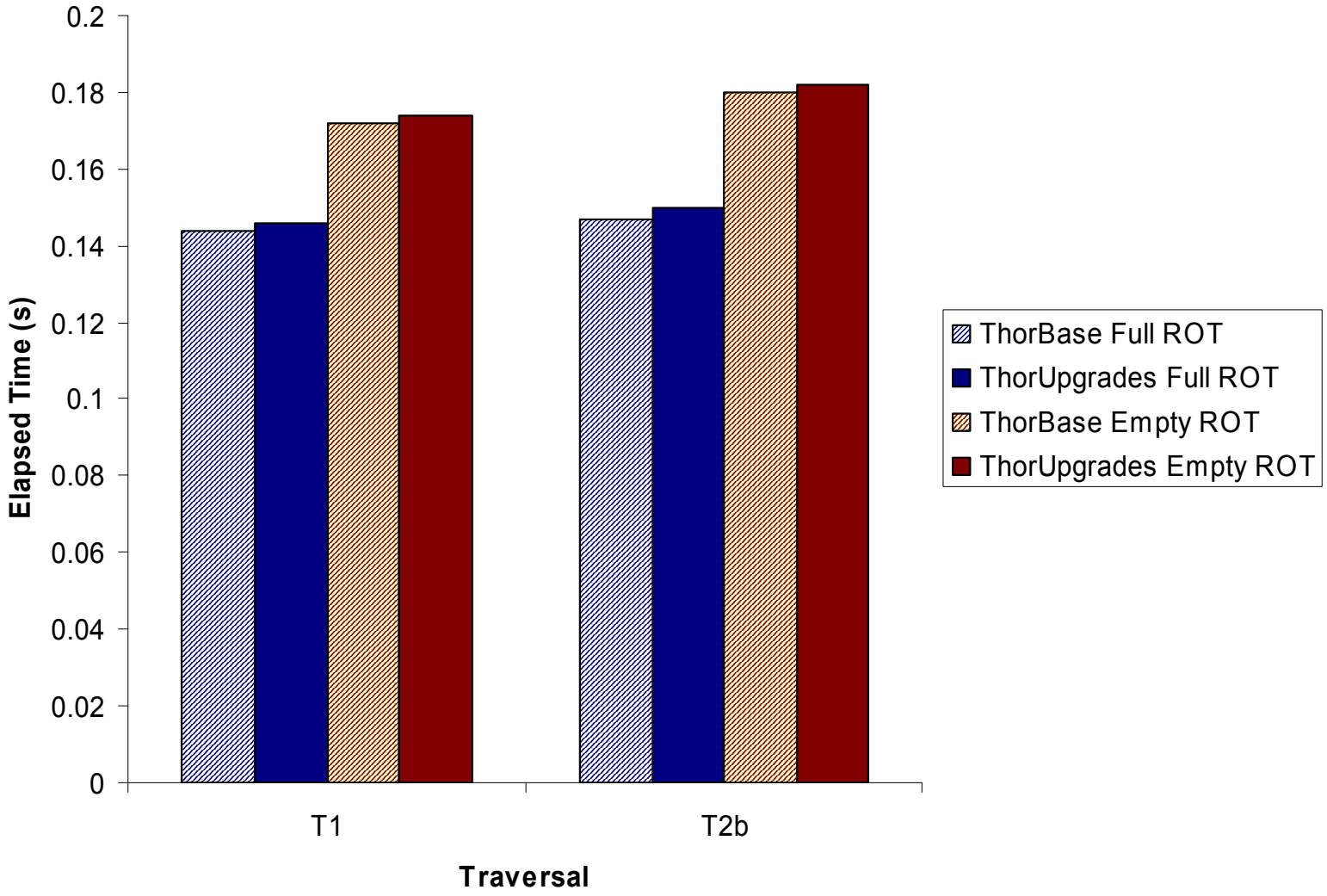


Performance

- Implemented in Thor
- Analyzed overhead



Baseline Overhead



Transform Overhead

Time per object (µs)	T1	T2b
Transform	11.3	11.5
Commit	19.9	1.0

Conclusions

- Correctness conditions for any upgrade system
  - Support modular reasoning
- Our lazy implementation approach
  - Correct and efficient
- Future work: upgrades in distributed systems

Lazy Modular Upgrades in Persistent Object Stores

- Joint work with
  - C. Boyapati
  - L. Shrira
  - C-H. Moh
  - S. Richman

- `http://pmg.csail.mit.edu/`

Execution Goals

- Goal: don’t interfere with applications
  - Don’t stop the world
- Goal: be efficient in space and time
  - Don’t copy the database or use versions
- Goal: be expressive
  - Don’t limit expressive power of TFs

The Right Time

- Upgrades are transactions
  - Serialized at moment of “installation”
- Upgrades must be serialized w.r.t
  - Application transactions
  - Other upgrades