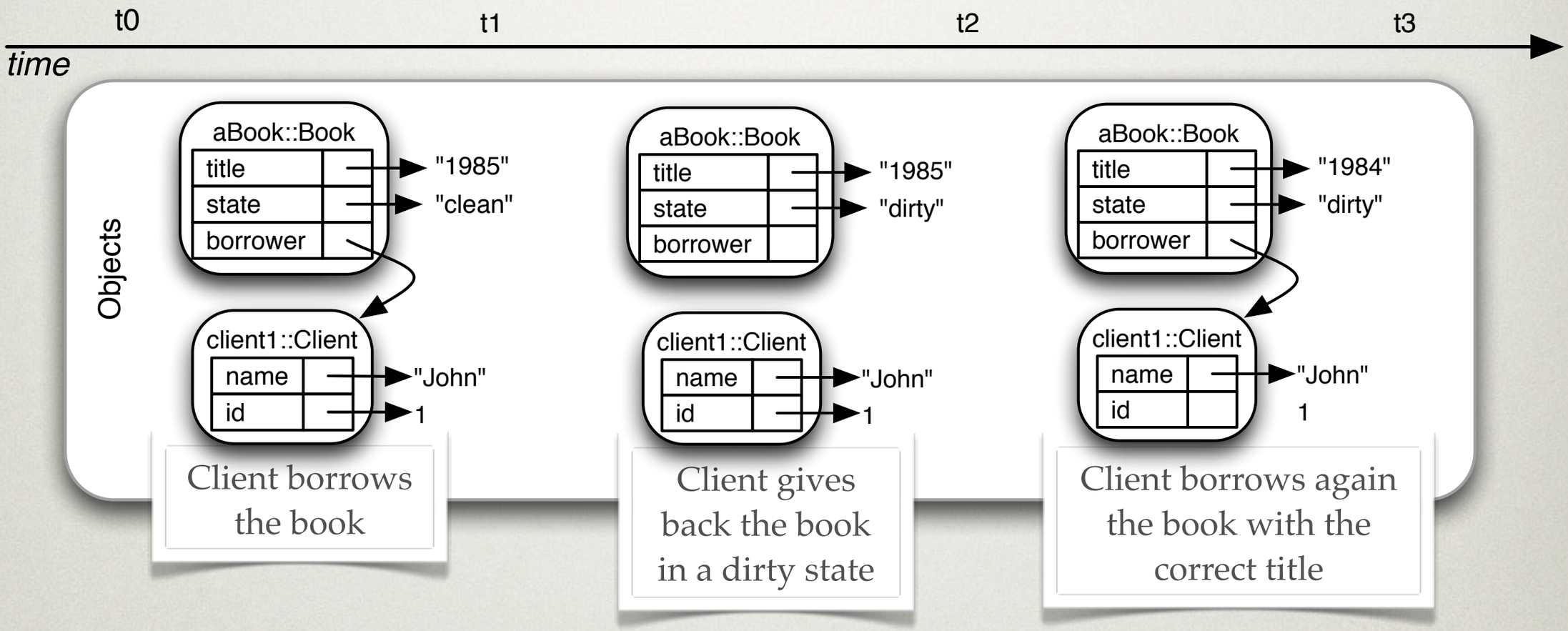
Executing Code in the Past: Efficient In-Memory Object Graph Versioning

Pluquet Frédéric
Stefan Langerman
Roel Wuyts

ULB

GRASCOMP, ULB, November 2011

In-Memory Object Versioning



«If I want revisit these 3 versions, how can I do it ?»

⇨ An ad hoc solution must be built

But...

- It is a recurring problem !
  - Eiffel-like Checked Post-Conditions
  - Stateful Tracer, Debugger, Google Wave
  - More than 20 applications in the theoretical algorithmic domain (computational geometry,...)
  - ...

Our Solution

In-Memory Object Versioning for any Object-Oriented Language

- What are the challenges ?
  - Be **generally applicable** !
  - Be **expressive** !
  - Be **efficient** in time and in space !

Intuitive approaches

- Deep copy of the system after each change

  ➡ The memory fills up very quickly

- Deep copy of the system at each given interval of time

  ➡ Some useful values can be not saved

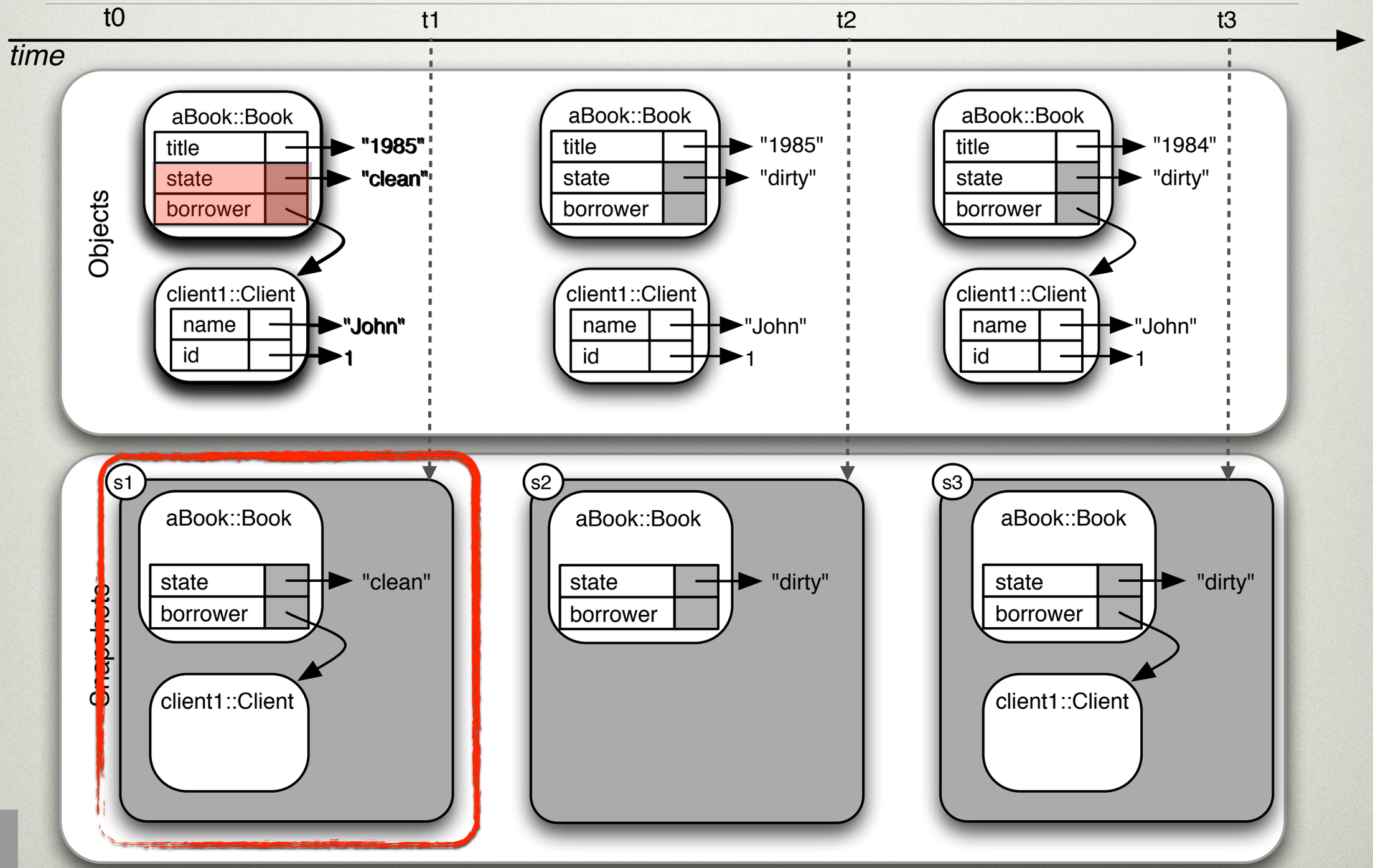
- Use database

  ➡ Not the same goal

What we really want

- Only keep
  - **interesting** old values ⇨ *Snapshots*
  - of **interesting** fields
  - of **interesting** objects ⇨ *Field Selection*

Example


t0
t1
t2
t3
time
Objects
aBook::Book
title
state
borrower
"1985"
"clean"
client1::Client
name
id
"John"
1
aBook::Book
title
state
borrower
"1985"
"dirty"
client1::Client
name
id
"John"
1
aBook::Book
title
state
borrower
"1984"
"dirty"
client1::Client
name
id
"John"
1
Snapshots
s1
aBook::Book
state
borrower
"clean"
client1::Client
s2
aBook::Book
state
borrower
"dirty"
s3
aBook::Book
state
borrower
"dirty"
client1::Client

Implementation

HistOOry

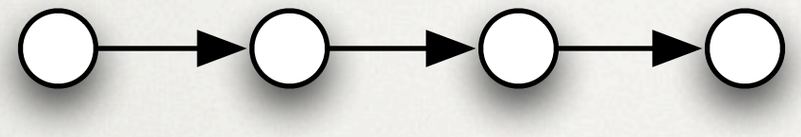
- Early version in Java
- Stable version in Smalltalk
  - A standard library
    - Can be loaded in any application
    - Existing code **does not need modification** (bytecode instrumentation)

Smalltalk integration

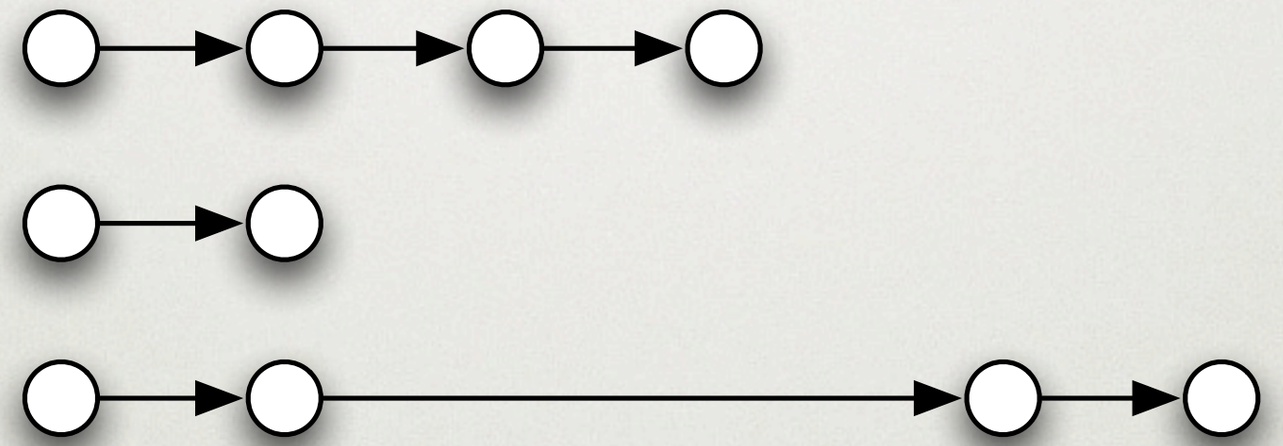
Demo

Kinds Of Versioning

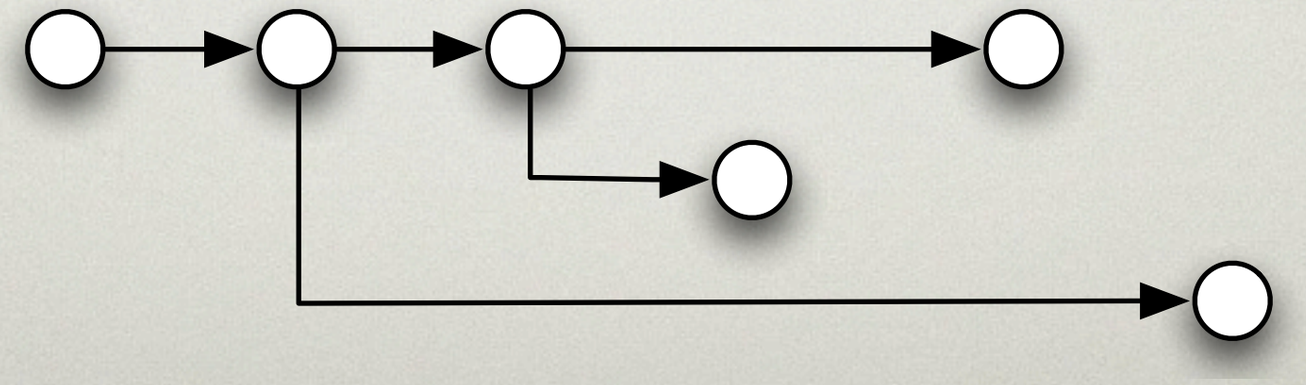
Linear



Backtracking



Branching



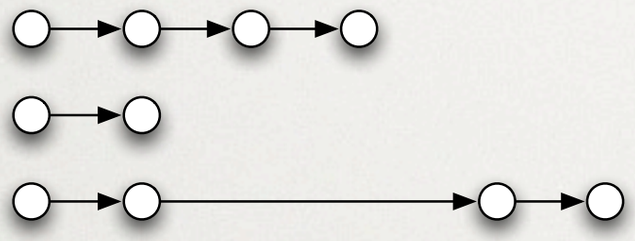
Complexities

Linear

	Complexities
Take a snapshot	O(1)
Access a non selected field	**O(1)**
Store in a selected field	O(1)
Read **last** value of a selected field	O(1)
Read **old** value of a selected field	O(log #values)

Complexities

Backtracking

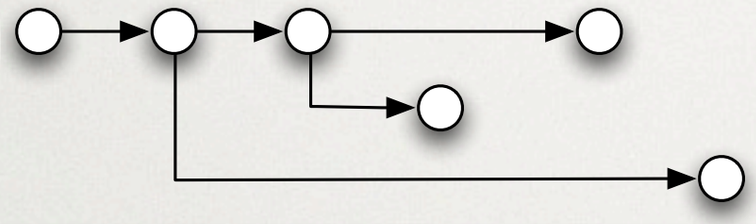


	Complexities
Take a snapshot	O(1)
Backtrack	O(1)
Access a non selected field	**O(1)**
Store in a selected field	O(1)*
Read **last** value of a selected field	O(1)*
Read **old** value of a selected field	O(log #values)

* amortized time

Complexities

Branching



	Complexities
Take a snapshot	O(1)*
Create a branch	O(1)*
Access a non selected field	**O(1)**
Store in a selected field	O(log #values)
Read **last** value of a selected field	O(log #values)
Read **old** value of a selected field	O(log #values)

* amortized time

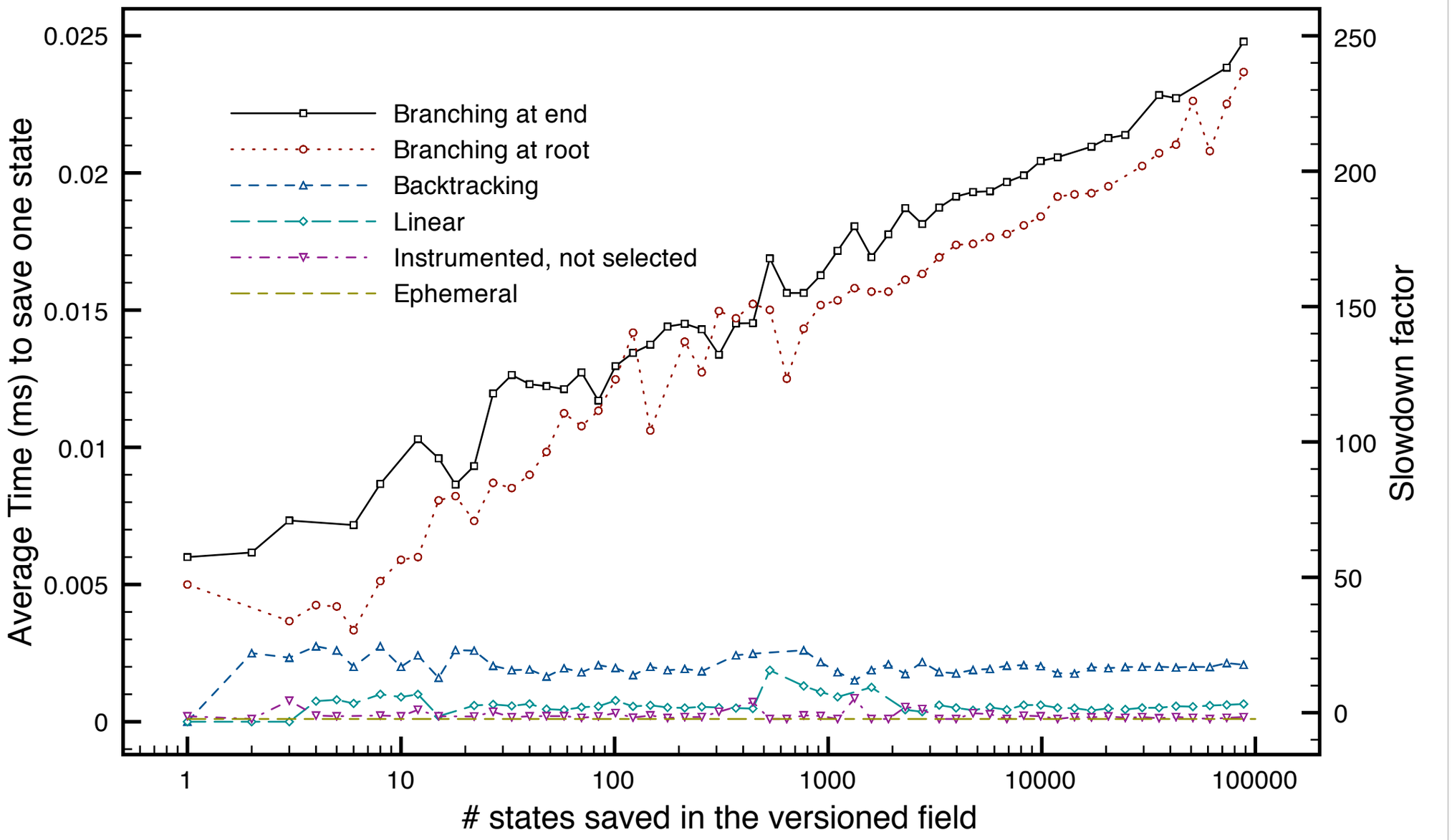
For all kinds of versioning

Access a non selected field	O(1)

If we do not select any field,
your system keeps same performance.

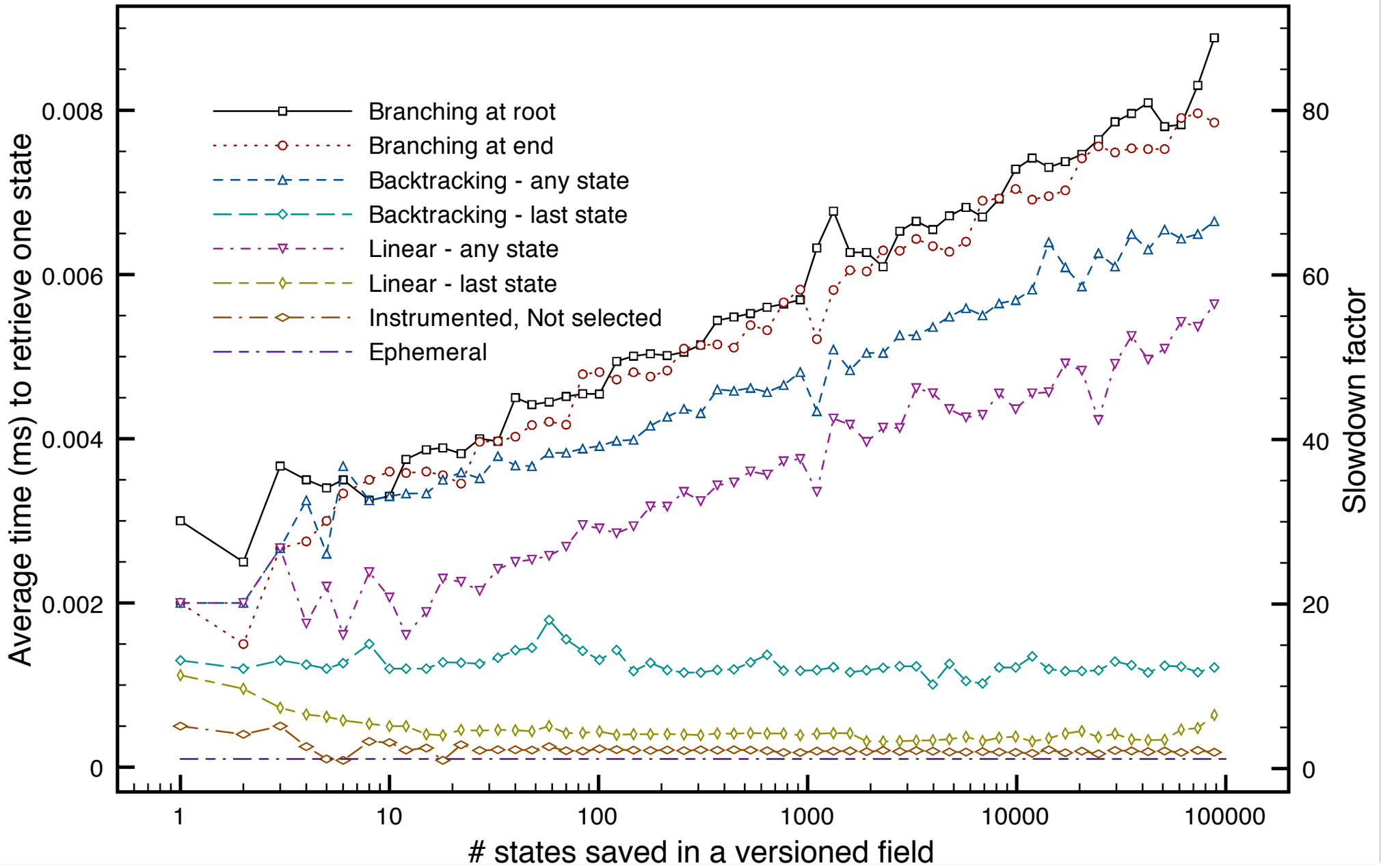
HistOOry Performance

Store



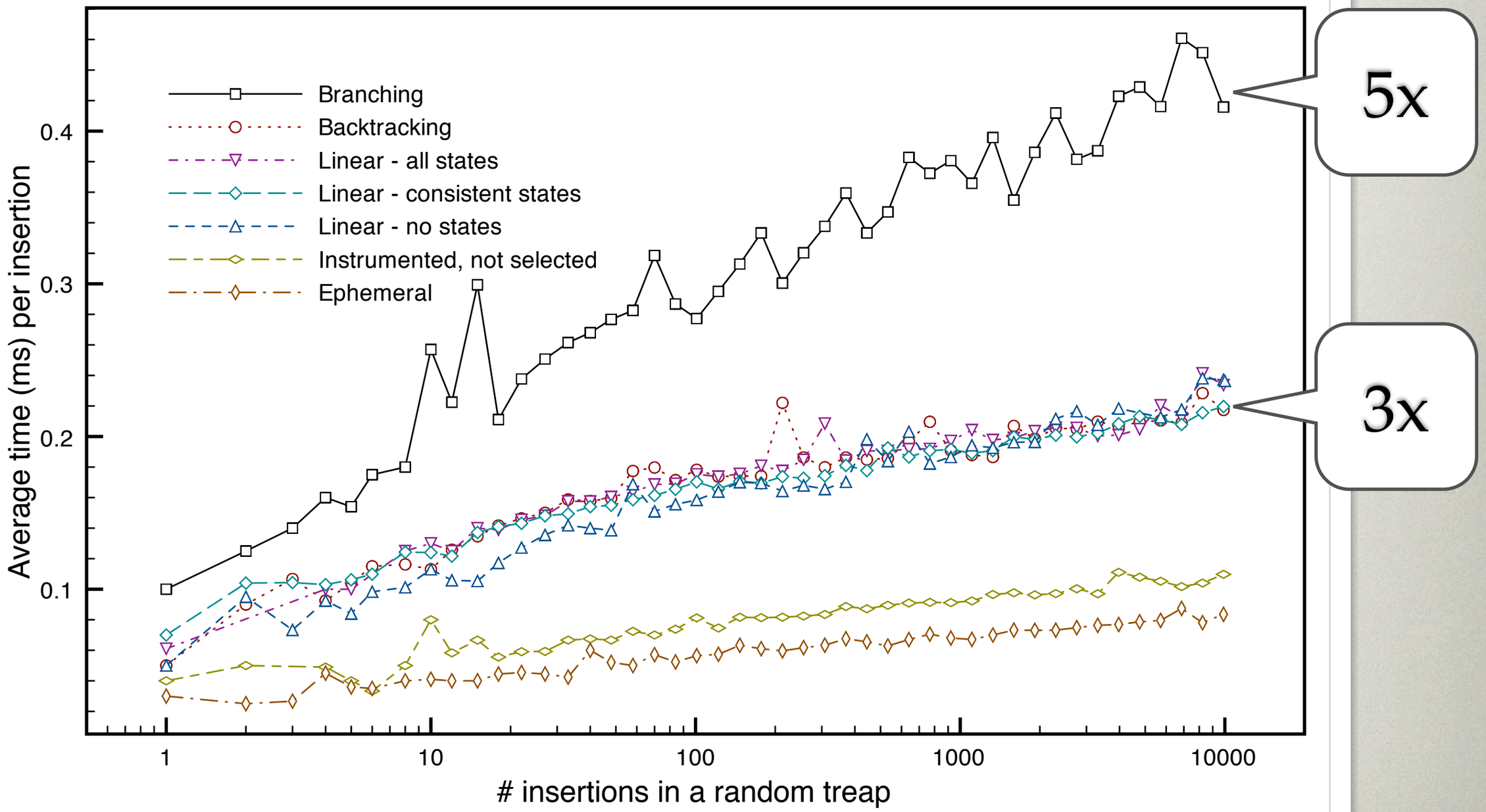
HistOOry Performance

Read



HistOOry Performance

Random Treap store



Conclusion

- A general and efficient model for in-memory object versioning for object-oriented languages
- Theoretical algorithms with implementation challenges
- We implemented it in Smalltalk
  - 3 kinds of versioning
  - Non-intrusive library
  - Efficient

Thank you !

- histoory@fpluquet.be
- HISTORY on Google
- *Executing code in the past: Efficient in-memory object graph versioning*, F. Pluquet, S. Langerman, and R. Wuyts, In Proceedings of the 2009 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'09).
- *Implementing Partial Persistence In Object-Oriented Languages*, F. Pluquet. S. Langerman, A. Marot and R. Wuyts, In Proceedings of ALENEX'08, 2008.