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

Pluquet Frédéric
Stefan Langerman
Roel Wuyts

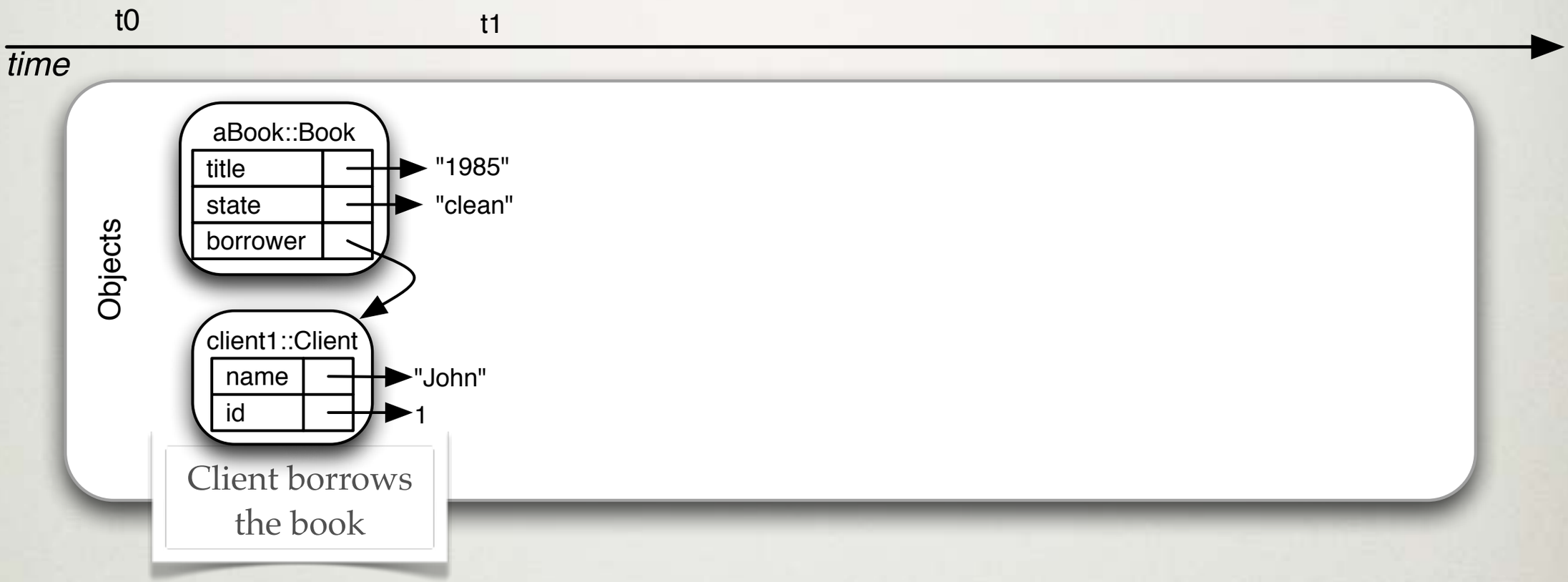
ULB

OOPSLA, October 2009

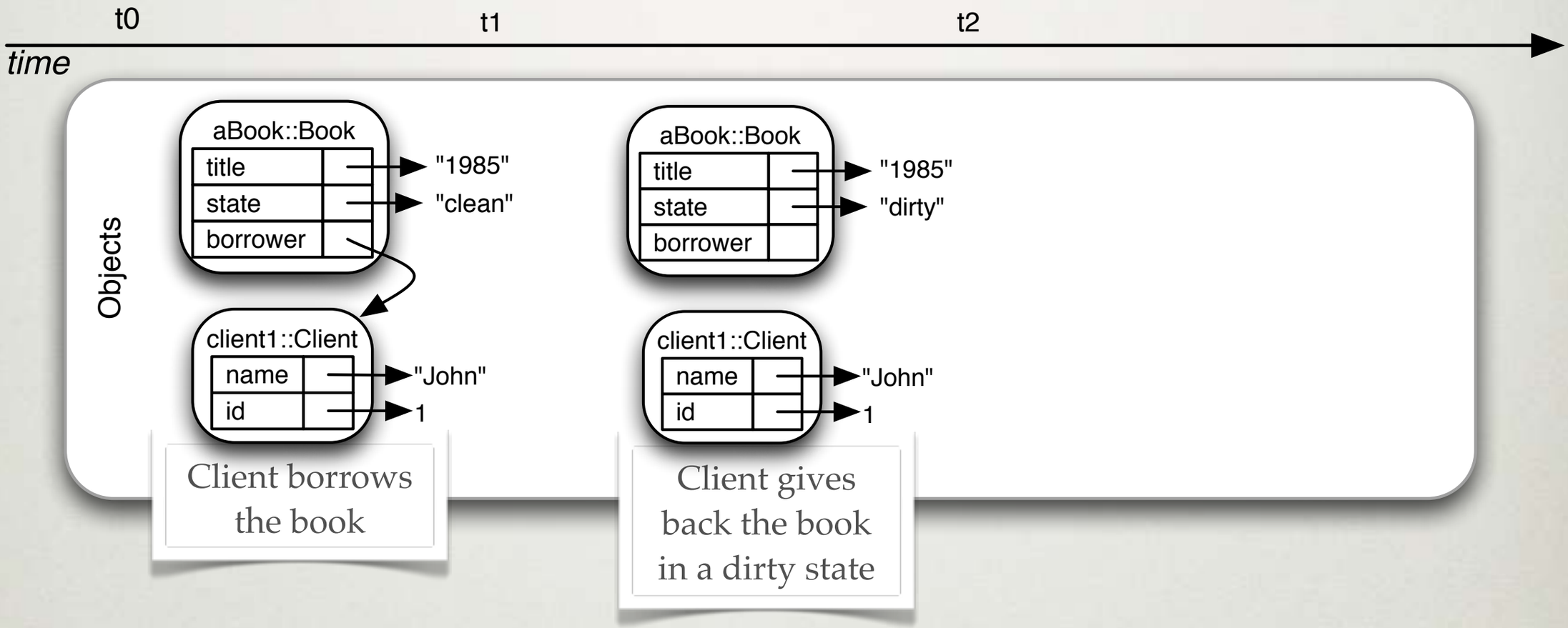
In-Memory Object Versioning

In-Memory Object Versioning

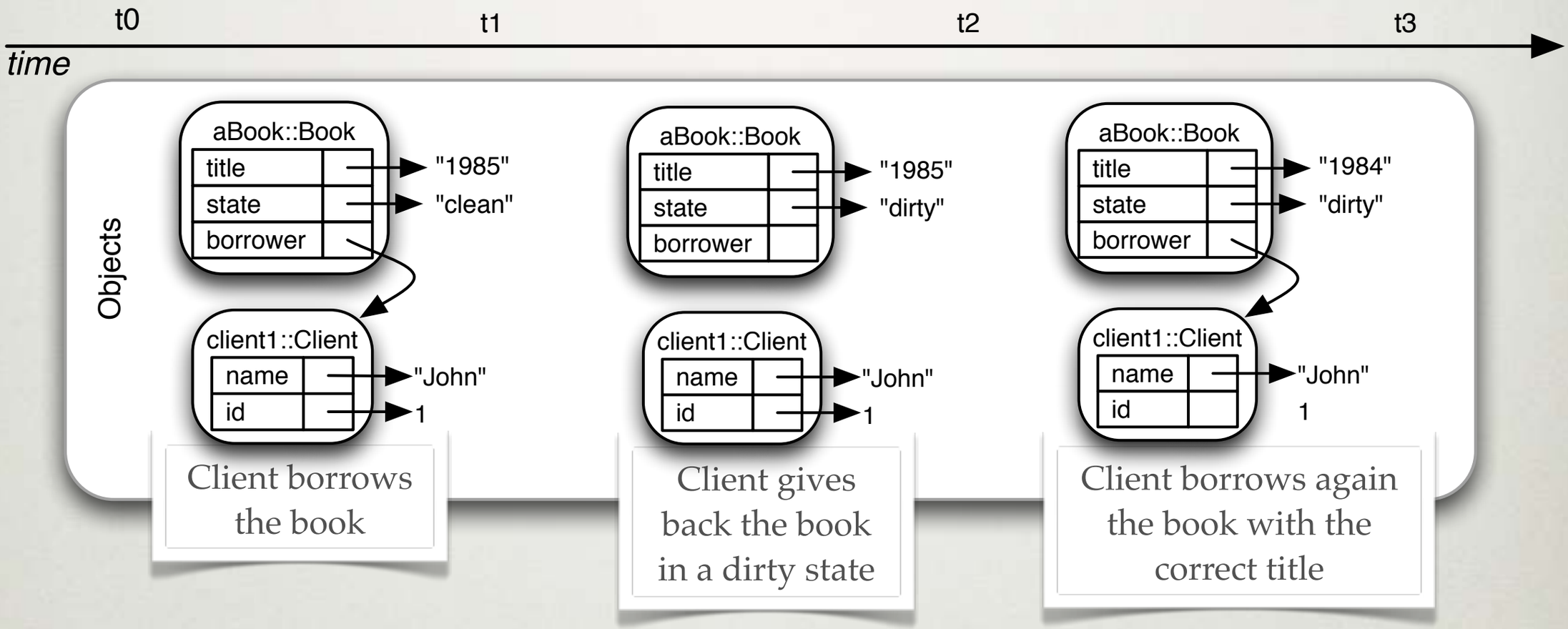
In-Memory Object Versioning



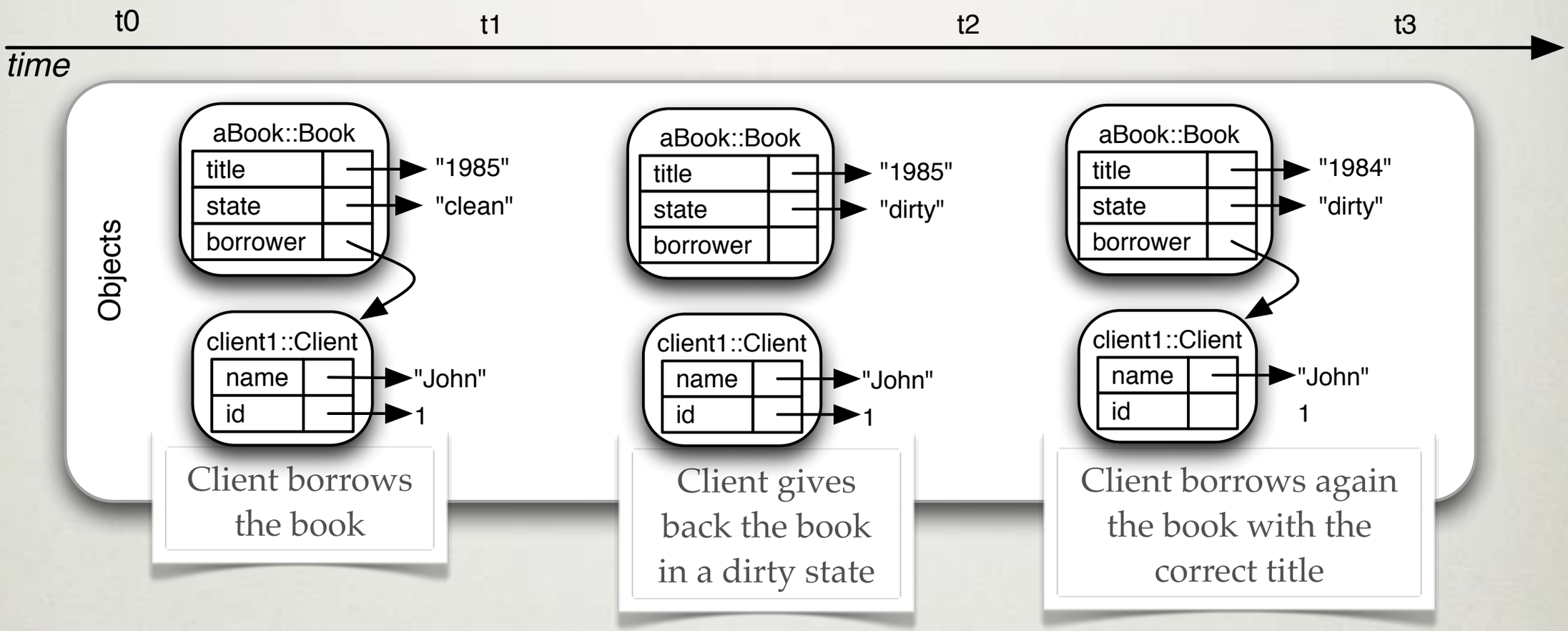
In-Memory Object Versioning



In-Memory Object Versioning

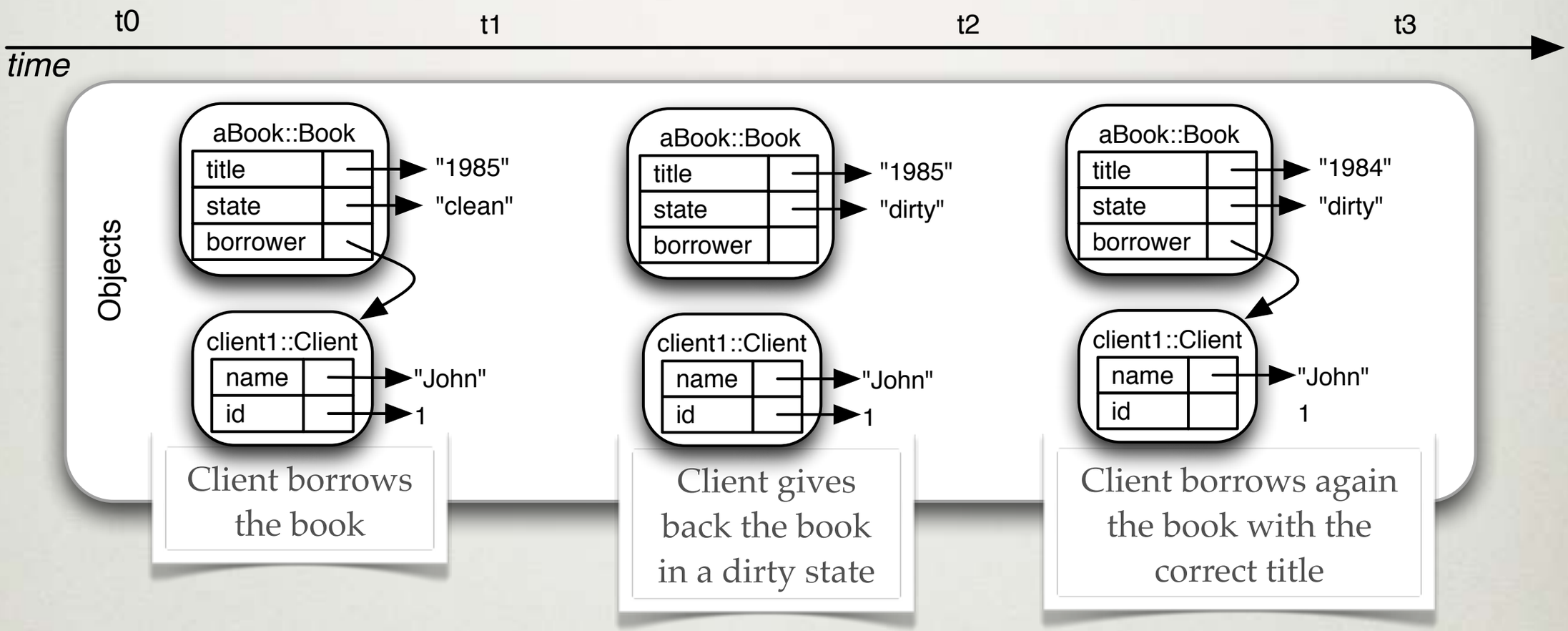


In-Memory Object Versioning



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

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

Introduce In-Memory Object Versioning in any Object-Oriented Language

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

Intuitive approaches

Intuitive approaches

- Deep copy of the system after each change

Intuitive approaches

- Deep copy of the system after each change

  ➡ The memory fills up very quickly

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

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

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

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

What we really want

- Only keep

What we really want

- Only keep
  - **interesting** old values

What we really want

- Only keep
  - **interesting** old values
    - of **interesting** fields

What we really want

- Only keep
  - **interesting** old values
  - of **interesting** fields
  - of **interesting** objects

What we really want

- Only keep
  - **interesting** old values
  - of **interesting** fields
  - of **interesting** objects

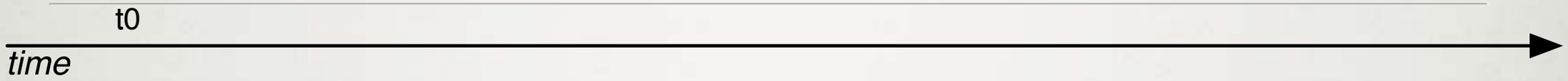
Field selection

What we really want

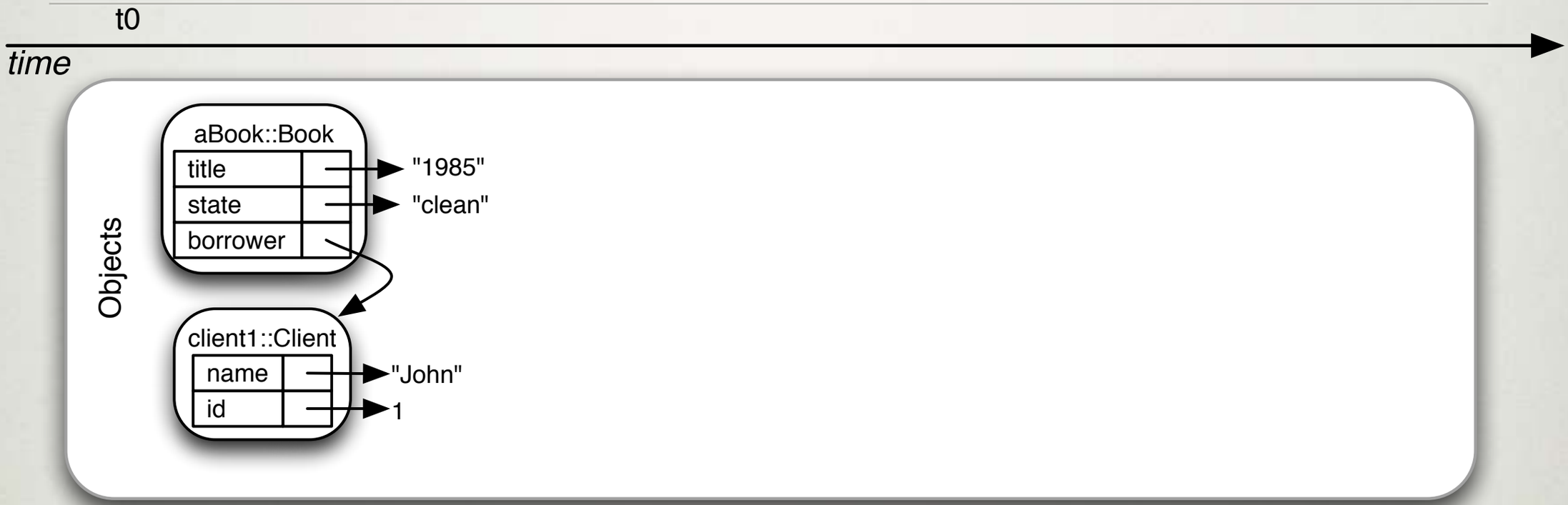
- Only keep
  - **interesting** old values ⇨ *Snapshots*
  - of **interesting** fields
  - of **interesting** objects ⇨ *Field selection*

Example

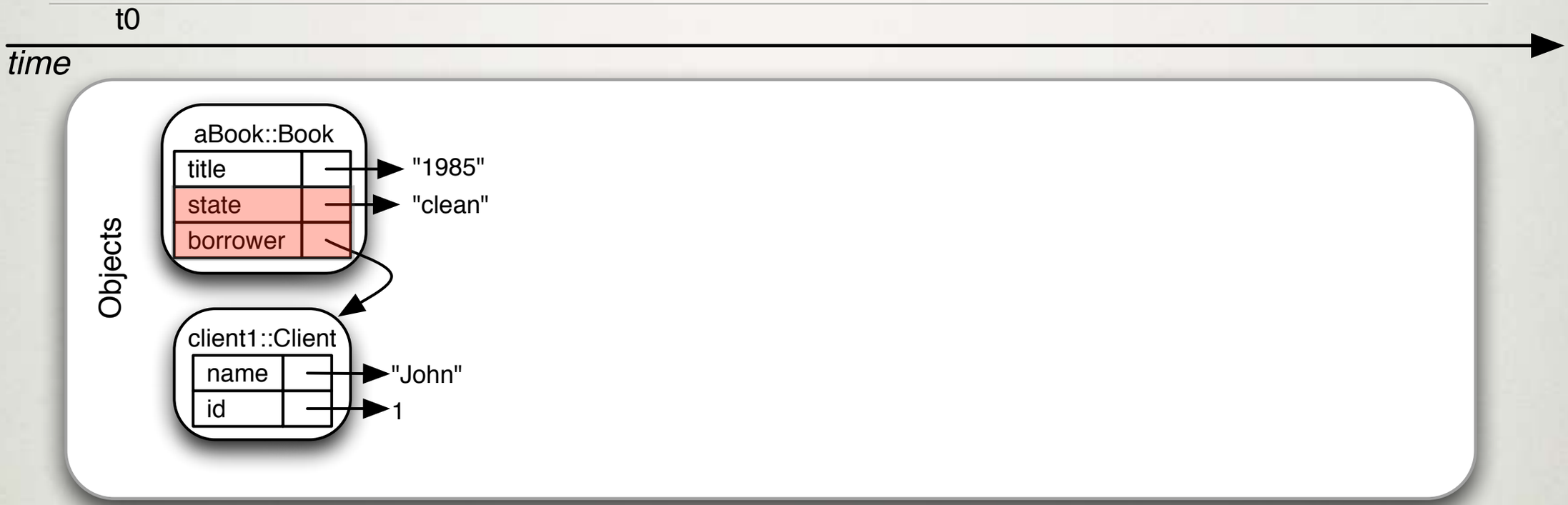
Example



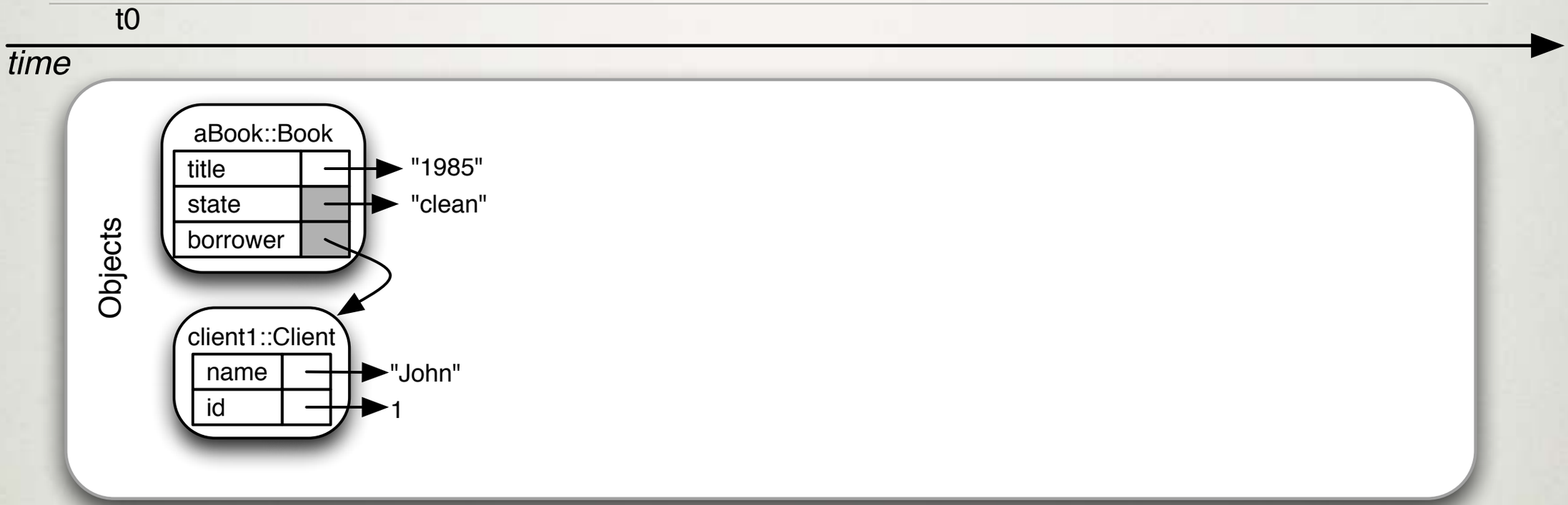
Example


t0
time
Objects
aBook::Book
title
state
borrower
"1985"
"clean"
client1::Client
name
id
"John"
1

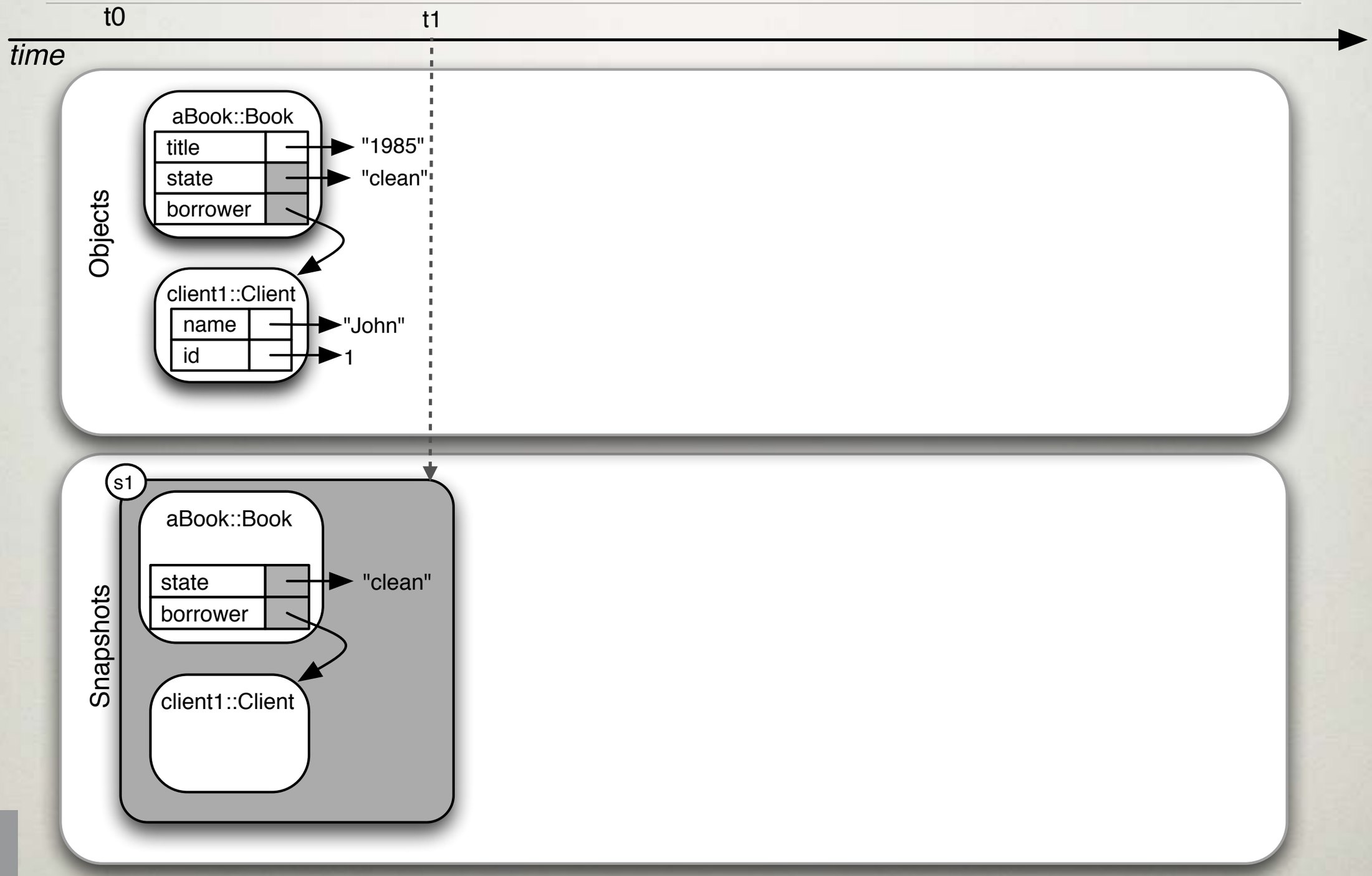
Example


t0
time
Objects
aBook::Book
title
state
borrower
"1985"
"clean"
client1::Client
name
id
"John"
1

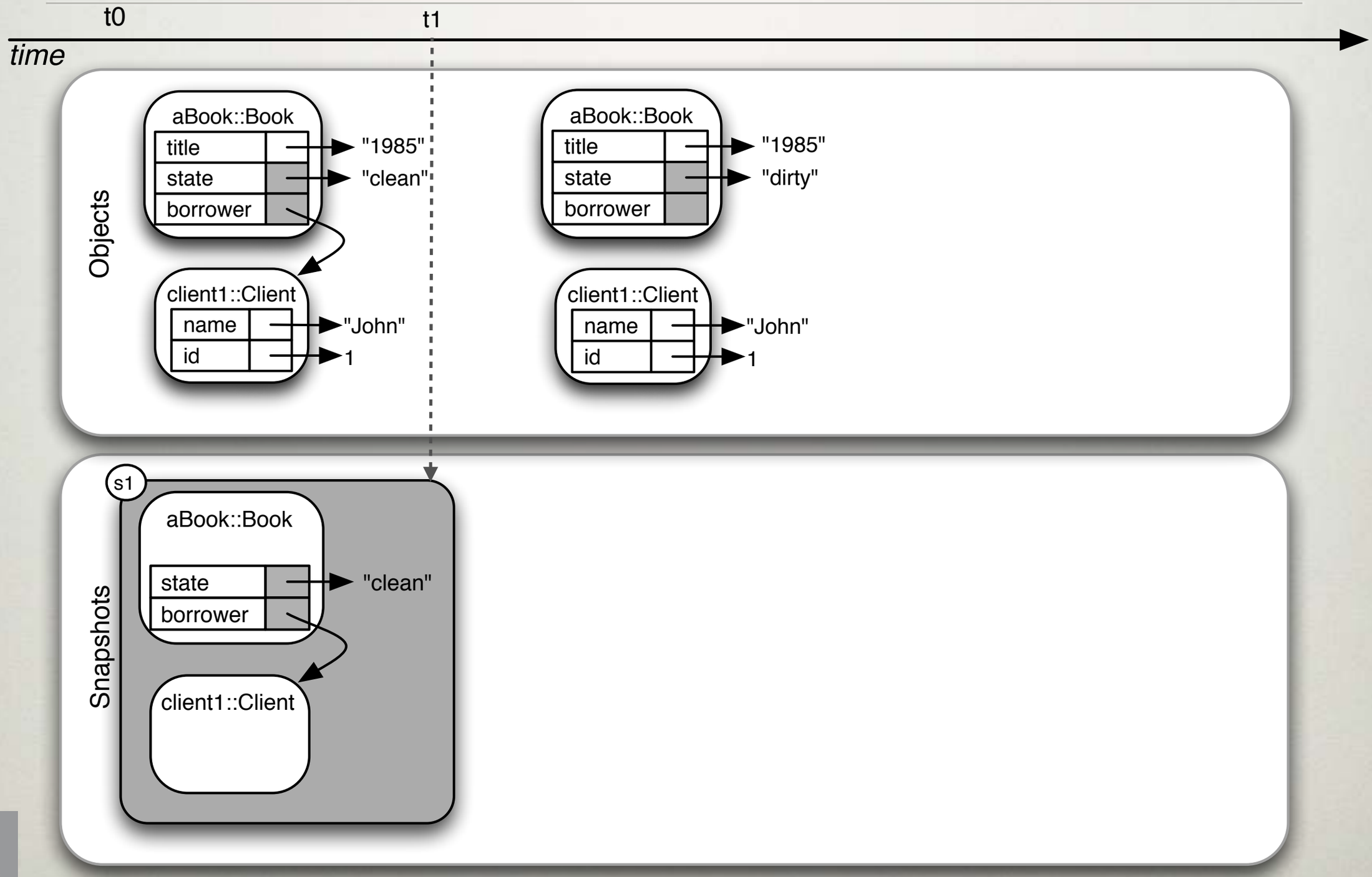
Example


t0
time
Objects
aBook::Book
title
state
borrower
"1985"
"clean"
client1::Client
name
id
"John"
1

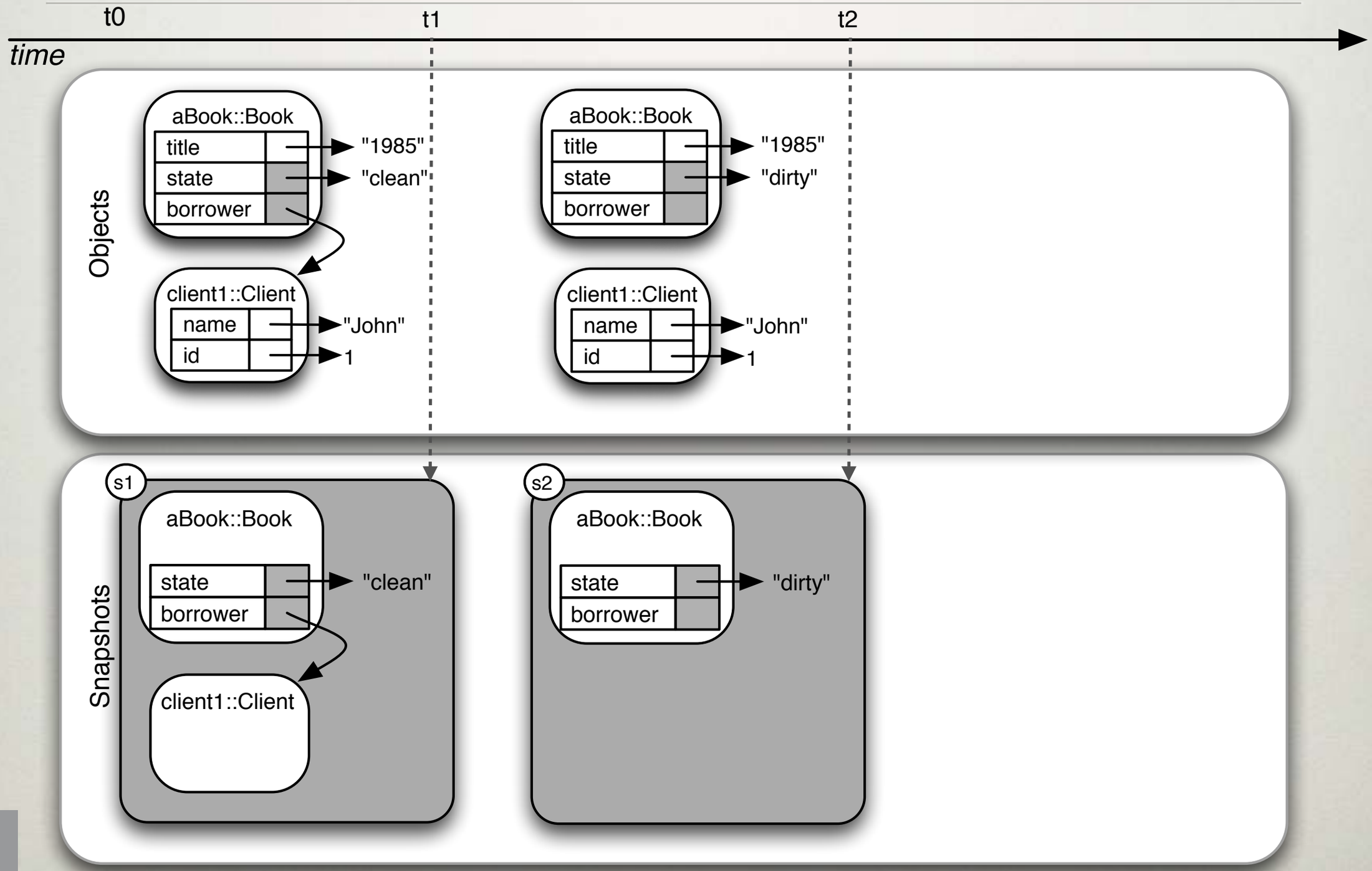
Example


t0
t1
time
Objects
aBook::Book
title
state
borrower
"1985"
"clean"
client1::Client
name
id
"John"
1
Snapshots
s1
aBook::Book
state
borrower
"clean"
client1::Client

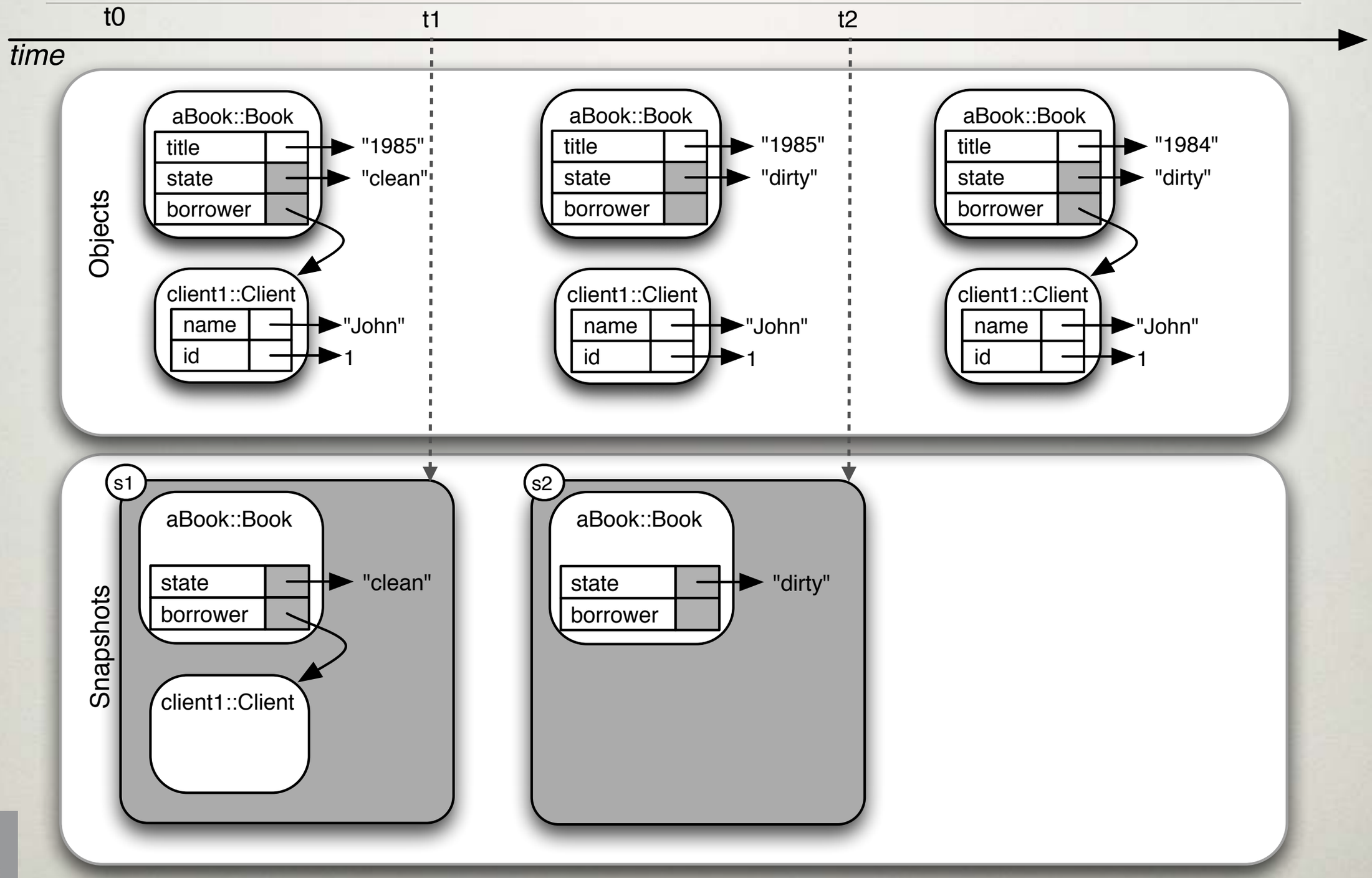
Example


t0
t1
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
Snapshots
s1
aBook::Book
state
borrower
"clean"
client1::Client

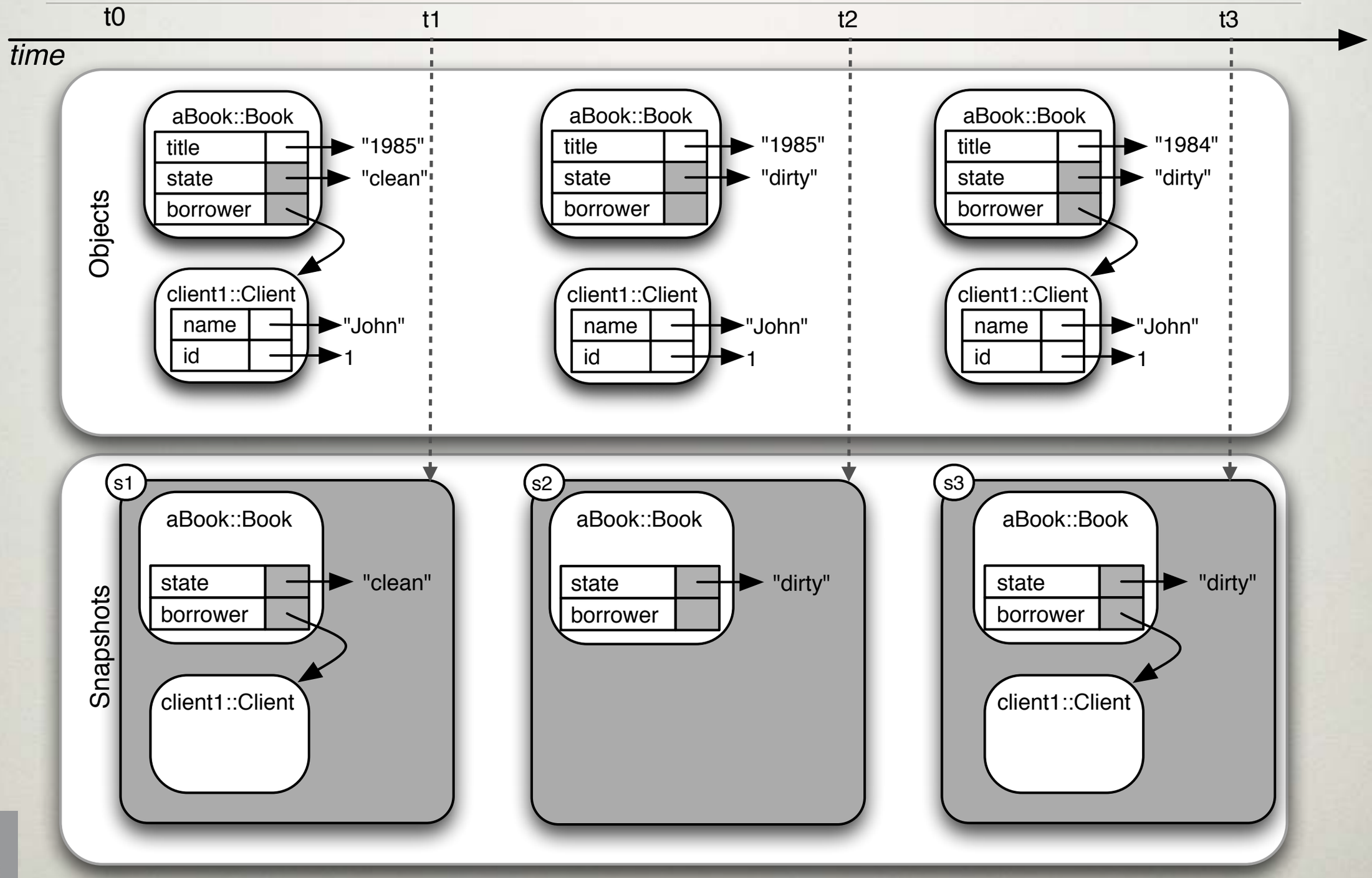
Example


t0
t1
t2
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
Snapshots
s1
aBook::Book
state
borrower
"clean"
client1::Client
s2
aBook::Book
state
borrower
"dirty"

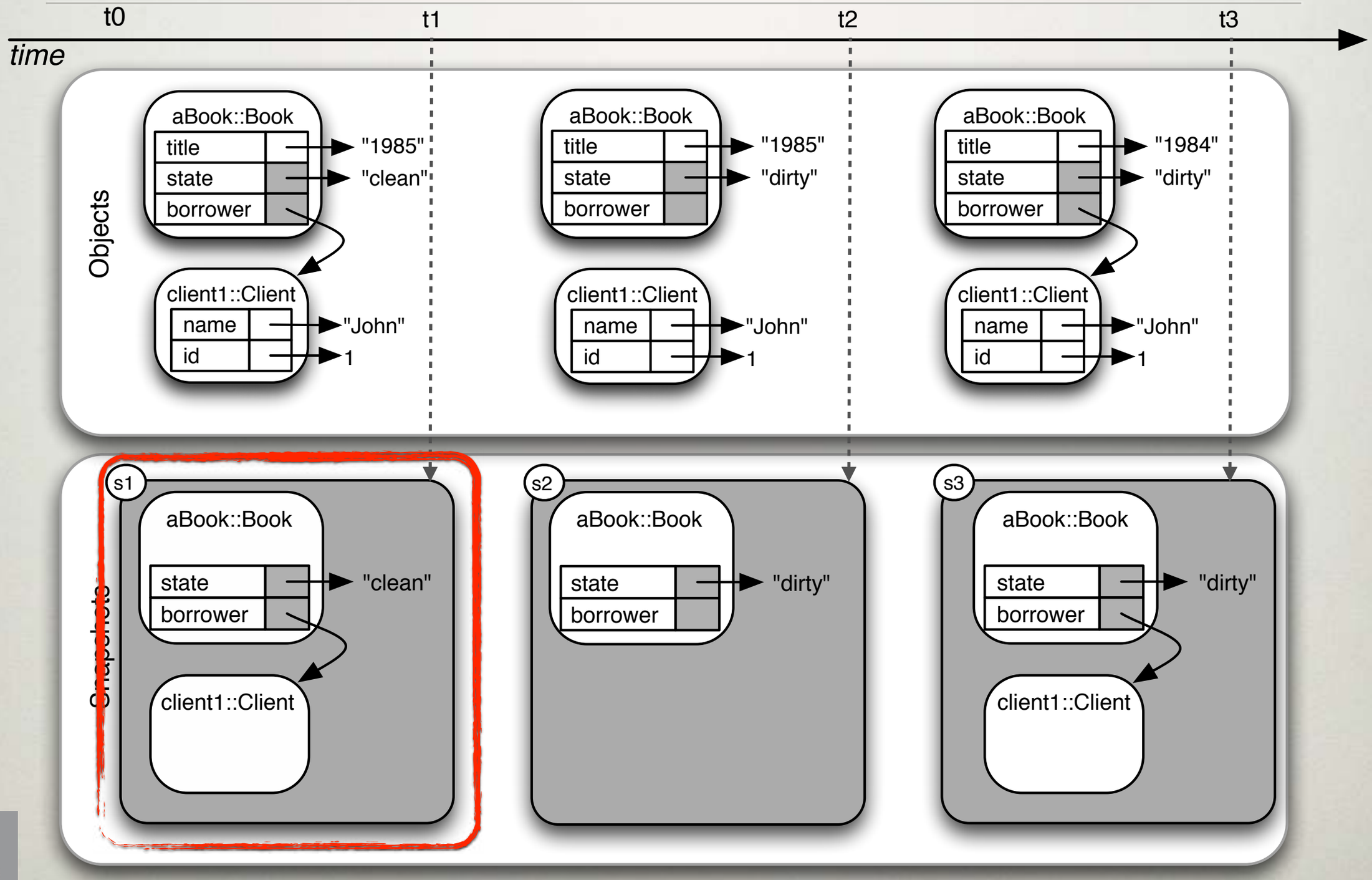
Example


t0
t1
t2
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"

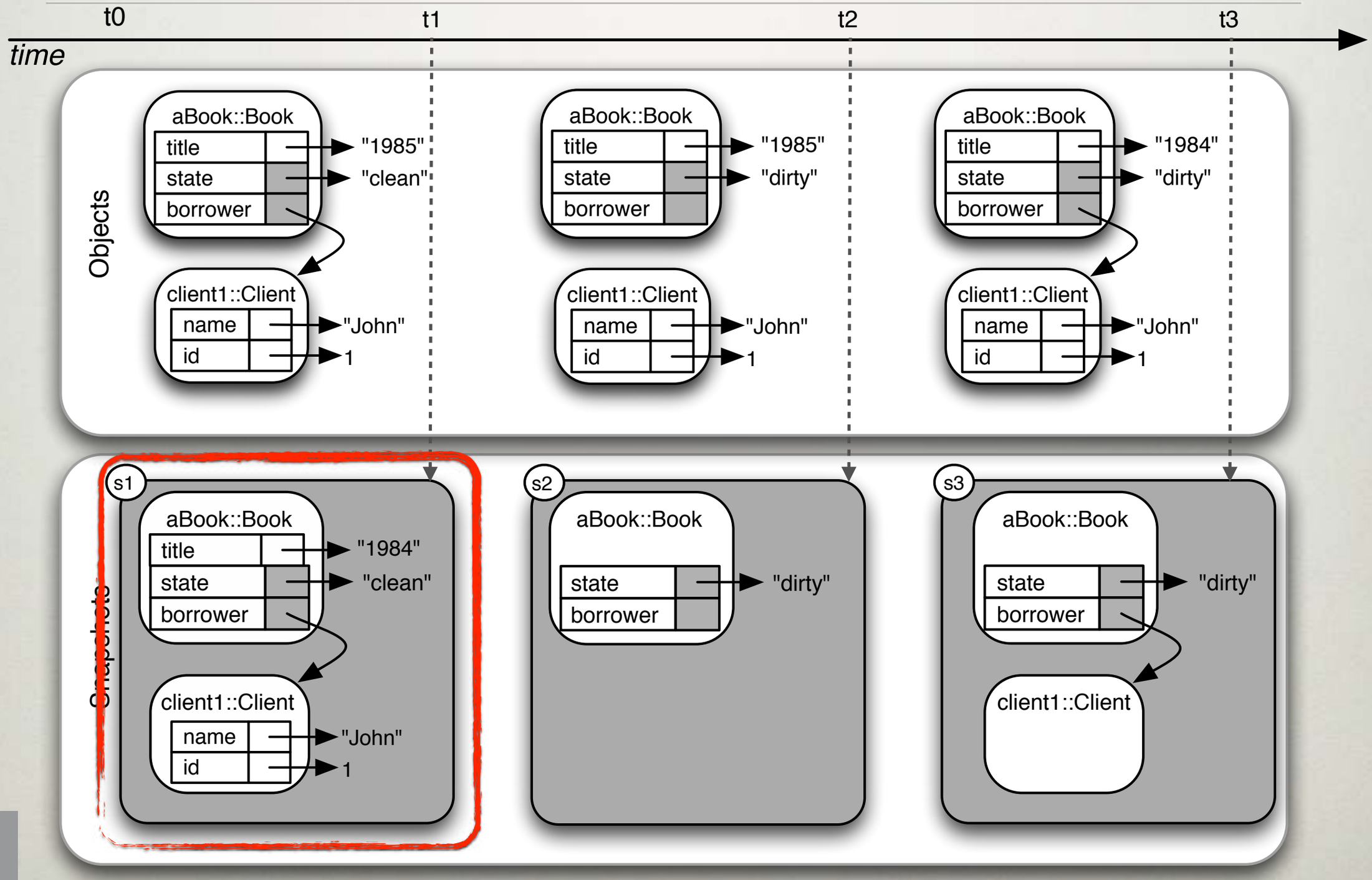
Example


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

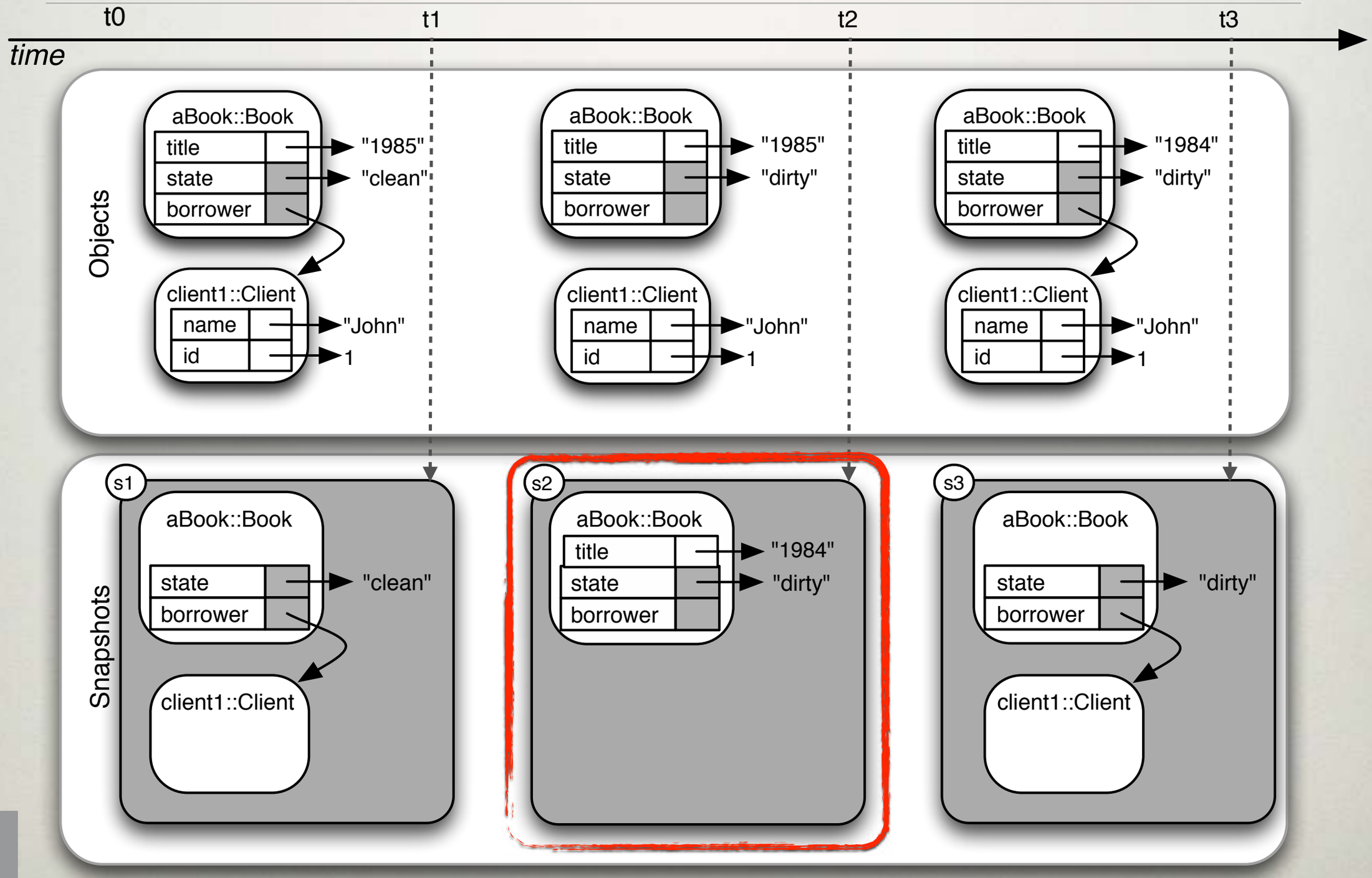
Example


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

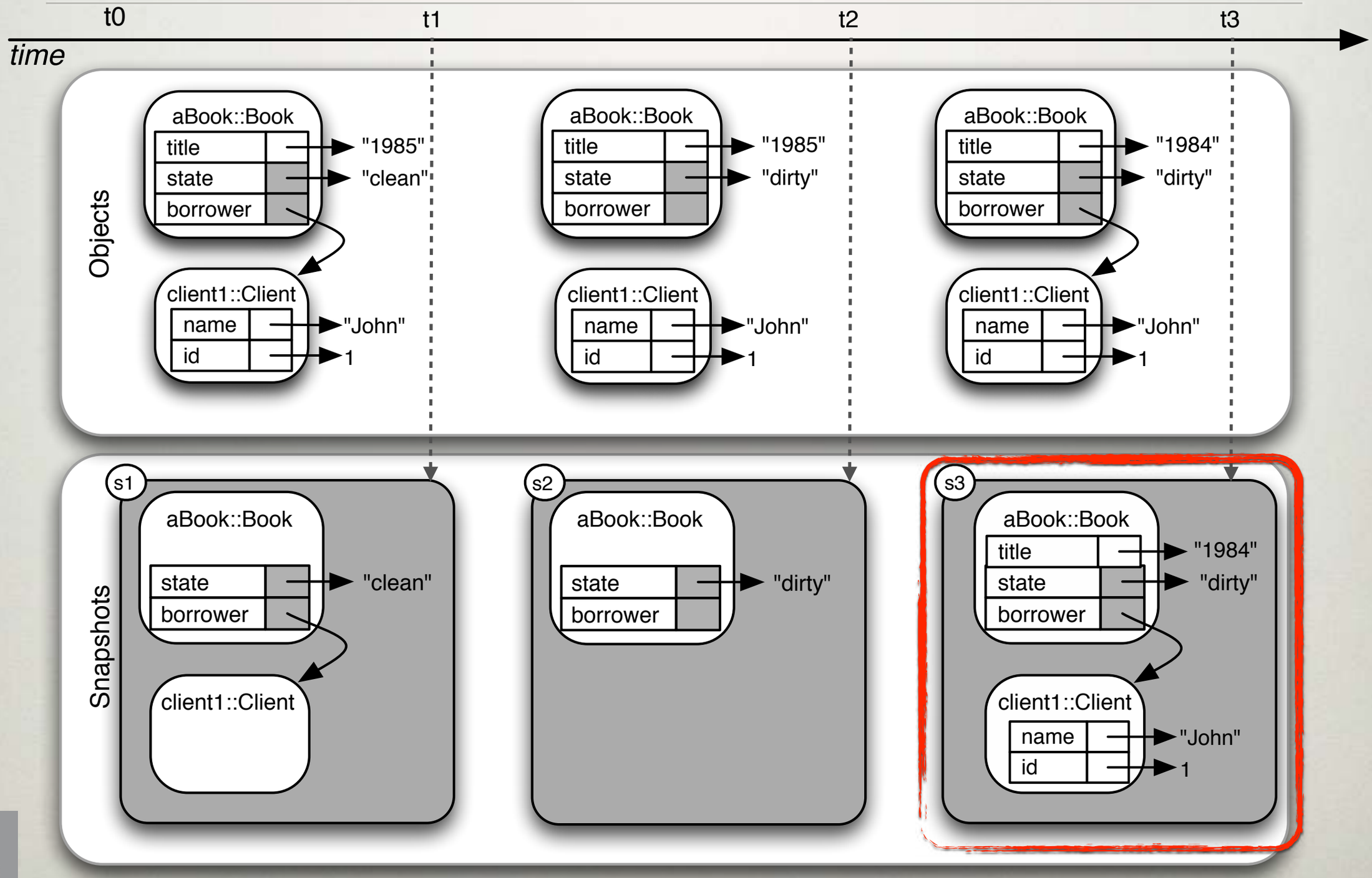
Example


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
title
state
borrower
"1984"
"clean"
client1::Client
name
id
"John"
1
s2
aBook::Book
state
borrower
"dirty"
s3
aBook::Book
state
borrower
"dirty"
client1::Client

Example

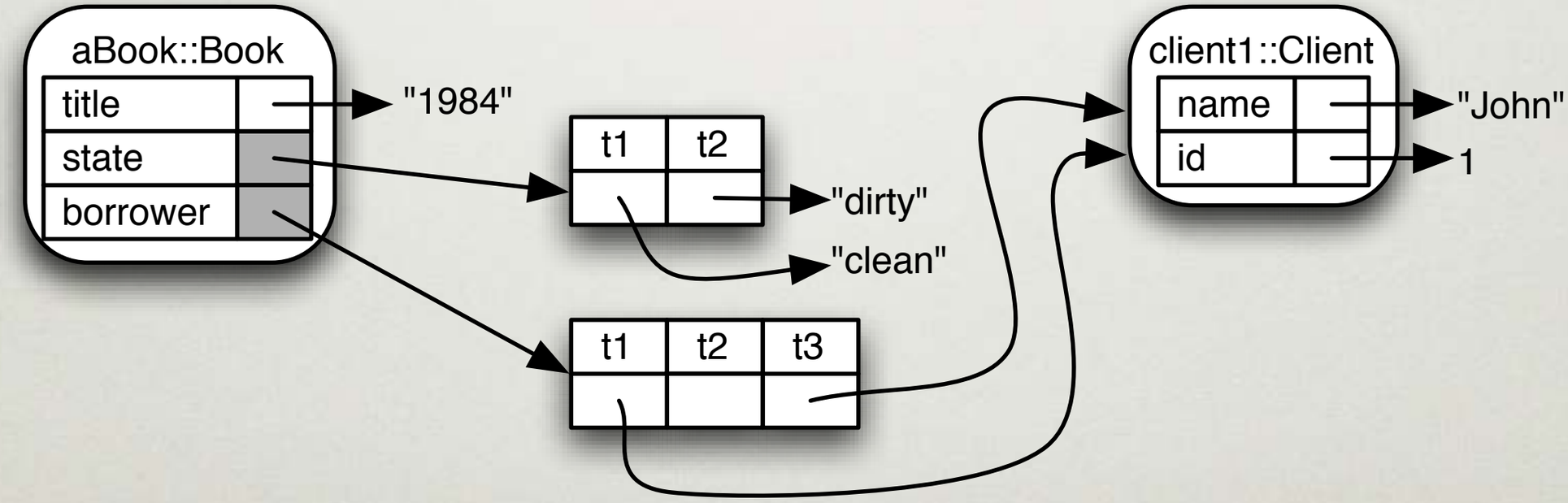

t0
t1
t2
t3
time
Objects
aBook::Book
title
state
borrower
"1985"
"clean"
client1::Client
name
id
"John"
1
aBook::Book
title
state
borrower
"1985"
"dirty"
client1::Client
name
id
"John"
1
aBook::Book
title
state
borrower
"1984"
"dirty"
client1::Client
name
id
"John"
1
Snapshots
s1
aBook::Book
state
borrower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

Example

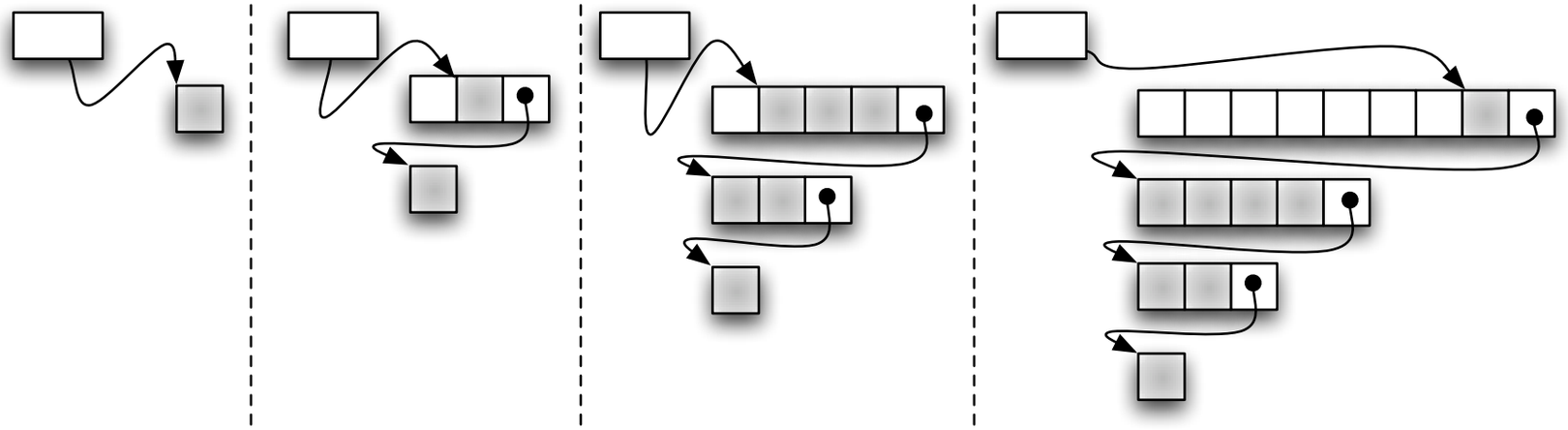

t0
t1
t2
t3
time
Objects
aBook::Book
title
state
borrower
"1985"
"clean"
client1::Client
name
id
"John"
1
aBook::Book
title
state
borrower
"1985"
"dirty"
client1::Client
name
id
"John"
1
aBook::Book
title
state
borrower
"1984"
"dirty"
client1::Client
name
id
"John"
1
Snapshots
s1
aBook::Book
state
borrower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

How to efficiently store old values ?

- *Fat Node Method* [Driscoll et al,1986].
  - Store the old values of a field in the field itself



How to efficiently store old values ?



(a) 1 element | (b) 2 elements | (b) 6 elements | (b) 8 elements

Insert a new value	**O(1)**
Access last value	**O(1)**
Look up for a value	O(log(*#values*))

Complexities

Complexities

	Worst case
Take a snapshot	O(1)

Complexities

	Worst case
Take a snapshot	O(1)
Perform an ephemeral update	O(1)

Complexities

	Worst case
Take a snapshot	O(1)
Perform an ephemeral update	O(1)
Perform a versioned update	O(1)

Complexities

	Worst case
Take a snapshot	O(1)
Perform an ephemeral update	O(1)
Perform a versioned update	O(1)
Read last value of a field	O(1)

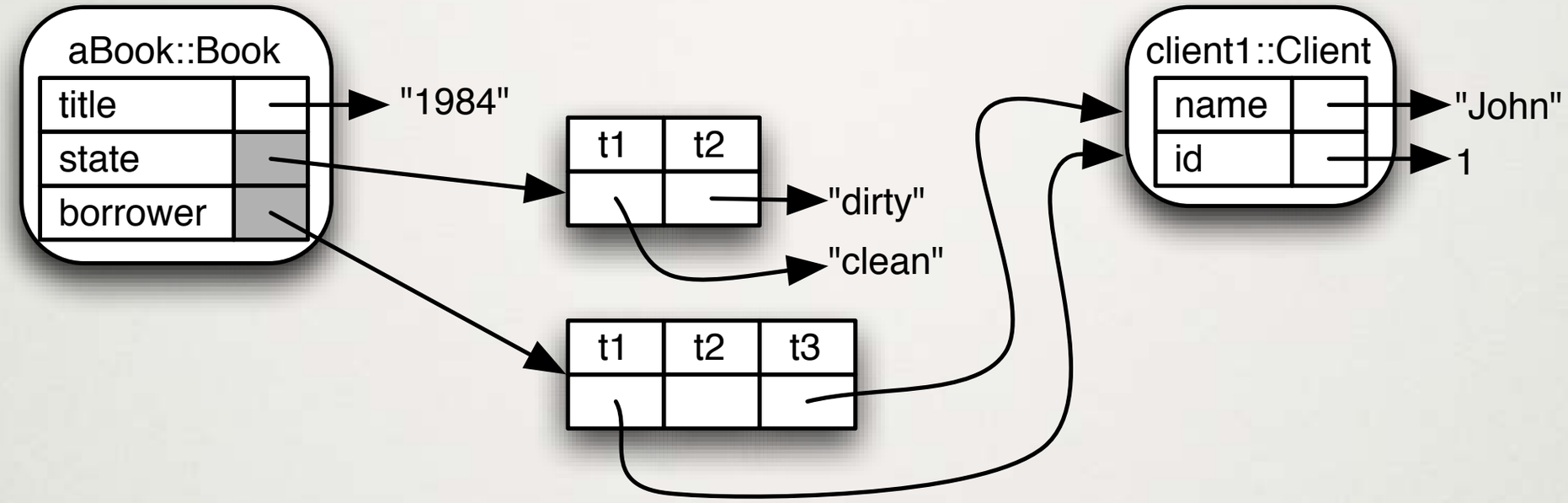
Complexities

	Worst case
Take a snapshot	O(1)
Perform an ephemeral update	O(1)
Perform a versioned update	O(1)
Read last value of a field	O(1)
Read old value of a field	O(log #values)

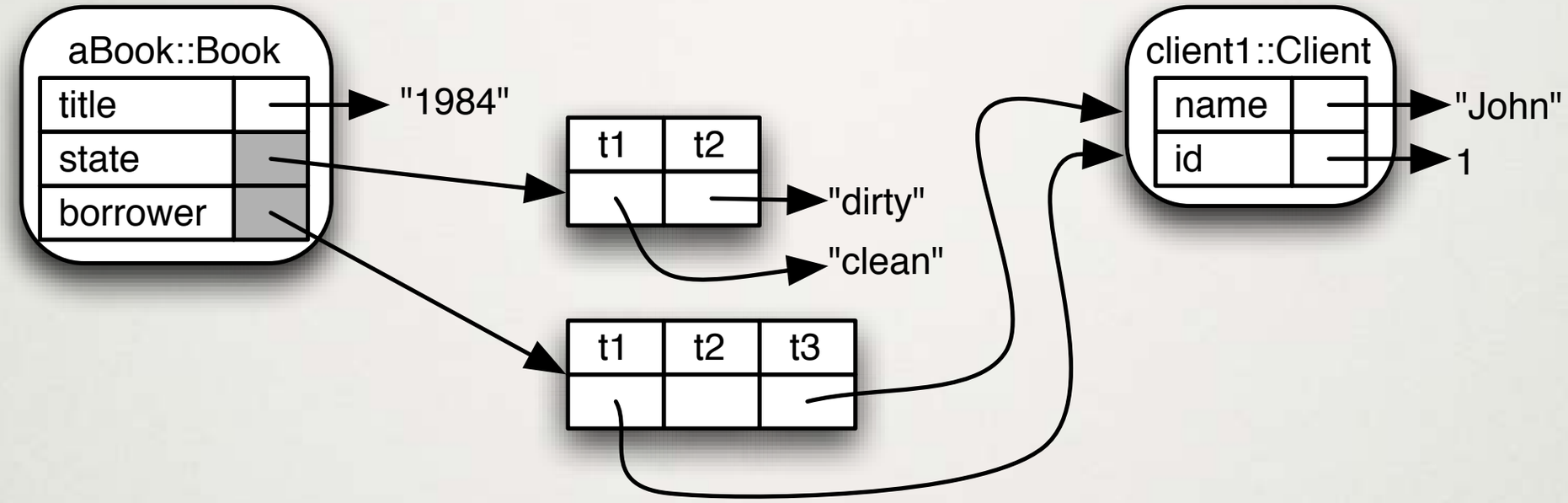
And of course

- Don’t lose time to save useless data
- Developer selects !

Efficiency in Size

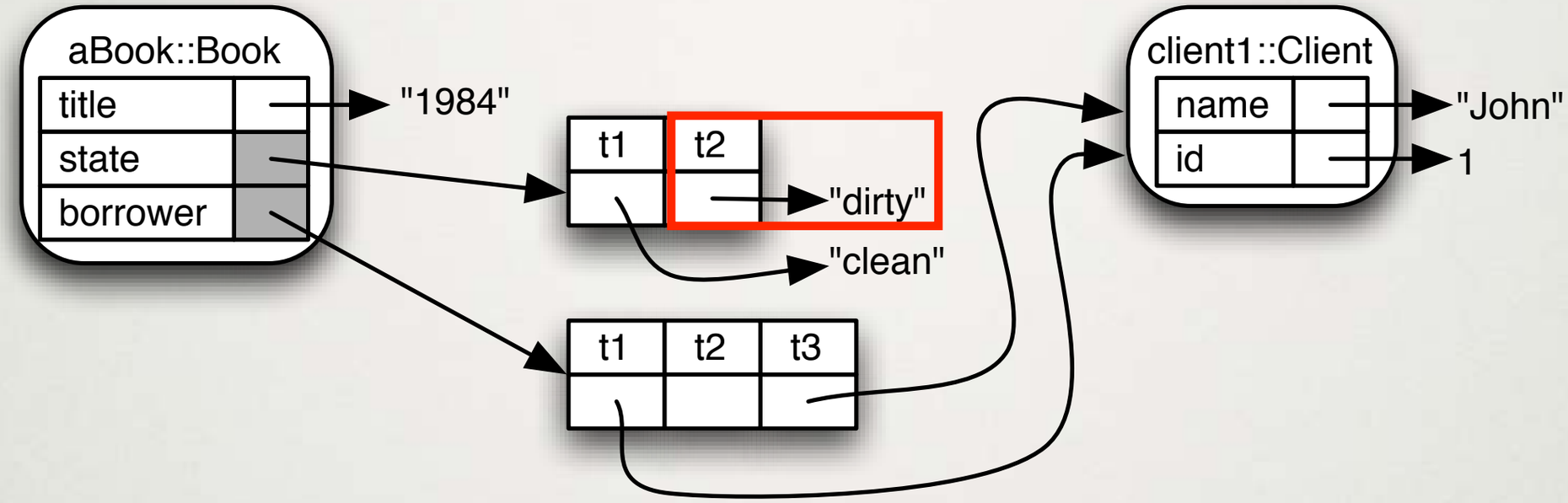


Efficiency in Size



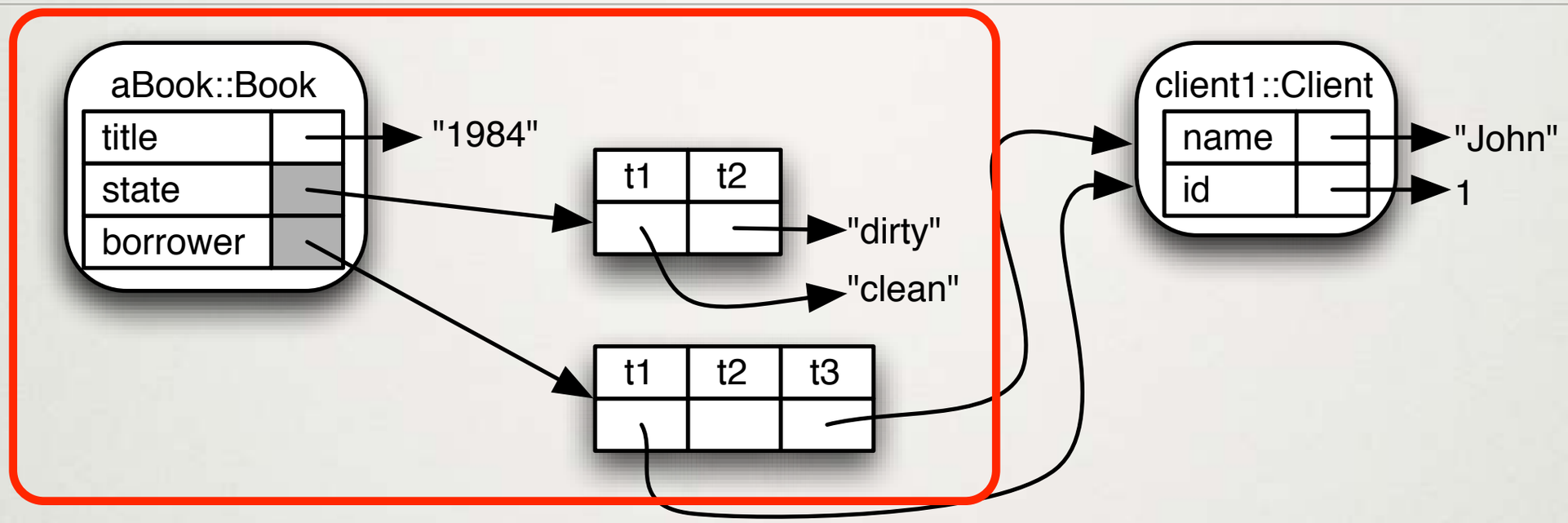
- States are shared between snapshots

Efficiency in Size



- States are shared between snapshots

Efficiency in Size



- States are shared between snapshots
- The states are in the object itself
  - If an object is no longer reachable, it is garbage collected
  - Including all of its fields history

Implementation

History

- Early version in Java
- Stable version in Smalltalk

Smalltalk integration

- A standard library
  - Can be loaded in any application
  - Existing code does not need modification
- Three primitives
  - `anObject` **`selectFields`**
  - `s :=` **`HSnapshot atNow`**
  - `s` **`execute:`** `aBlock`
- Easy to learn and use

Smalltalk integration

borrowing

```
aClient := Client named: ‘John’.
aBook := Book titled: ‘1985’.
aBook state: ‘clean’.
aBook borrower: aClient.
```

returning

```
aBook state: ‘dirty’.
aBook borrower: nil.
```

Smalltalk integration

borrowing

```
aClient := Client named: ‘John’.
aBook := Book titled: ‘1985’.
aBook state: ‘clean’.
aBook borrower: aClient.
aBook selectFields: {#state. #borrower}.
```

returning

```
aBook state: ‘dirty’.
aBook borrower: nil.
```

Smalltalk integration

borrowing

```
aClient := Client named: ‘John’.
aBook := Book titled: ‘1985’.
aBook state: ‘clean’.
aBook borrower: aClient.
aBook selectFields: {#state. #borrower}.
s1 := HSnapshot atNow.
```

returning

```
aBook state: ‘dirty’.
aBook borrower: nil.
```

Smalltalk integration

borrowing

```
aClient := Client named: ‘John’.
aBook := Book titled: ‘1985’.
aBook state: ‘clean’.
aBook borrower: aClient.
aBook selectFields: {#state. #borrower}.
s1 := HSnapshot atNow.
```

returning

```
aBook state: ‘dirty’.
aBook borrower: nil.

aBook state.
```

Smalltalk integration

borrowing

```
aClient := Client named: ‘John’.
aBook := Book titled: ‘1985’.
aBook state: ‘clean’.
aBook borrower: aClient.
aBook selectFields: {#state. #borrower}.
s1 := HSnapshot atNow.
aBook state: ‘dirty’.
aBook borrower: nil.
```

returning

```
aBook state. ⟶ ‘dirty’
```

Smalltalk integration

borrowing

```
aClient := Client named: ‘John’.
aBook := Book titled: ‘1985’.
aBook state: ‘clean’.
aBook borrower: aClient.
aBook selectFields: {#state. #borrower}.
s1 := HSnapshot atNow.
```

returning

```
aBook state: ‘dirty’.
aBook borrower: nil.

aBook state.                    → ‘dirty’
s1 execute: [aBook state].
```

Smalltalk integration

borrowing

```
aClient := Client named: ‘John’.
aBook := Book titled: ‘1985’.
aBook state: ‘clean’.
aBook borrower: aClient.
aBook selectFields: {#state. #borrower}.
s1 := HSnapshot atNow.
aBook state: ‘dirty’.
aBook borrower: nil.

aBook state.  ⟶ ‘dirty’
s1 execute: [aBook state].  ⟶ ‘clean’
```

returning

Performance

Performance

	Slowdown factor
Synthetic worst case update	7

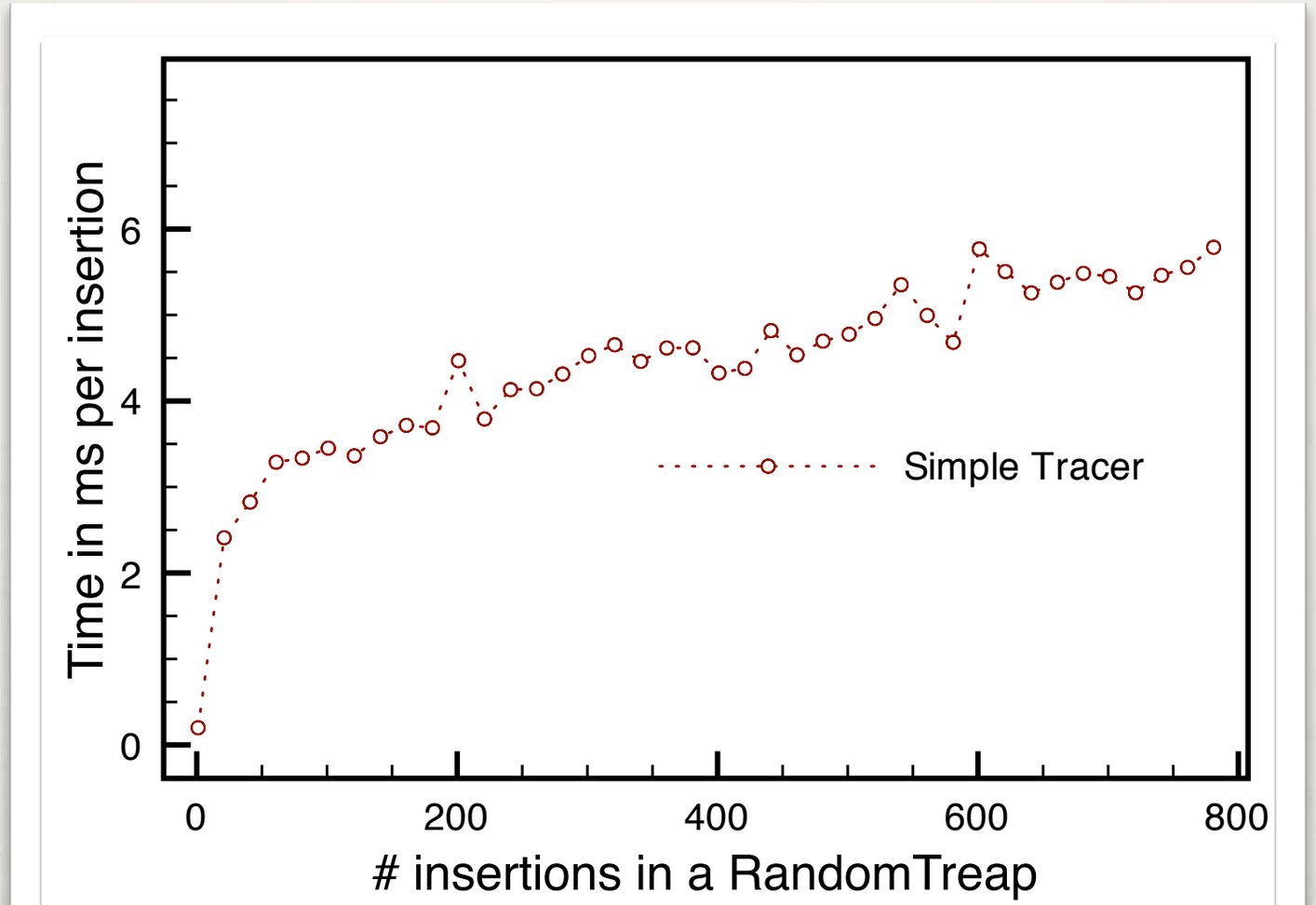
Performance

	Slowdown factor
Synthetic worst case update	**7**
Real cases update	**1.3 to 2**

Applications

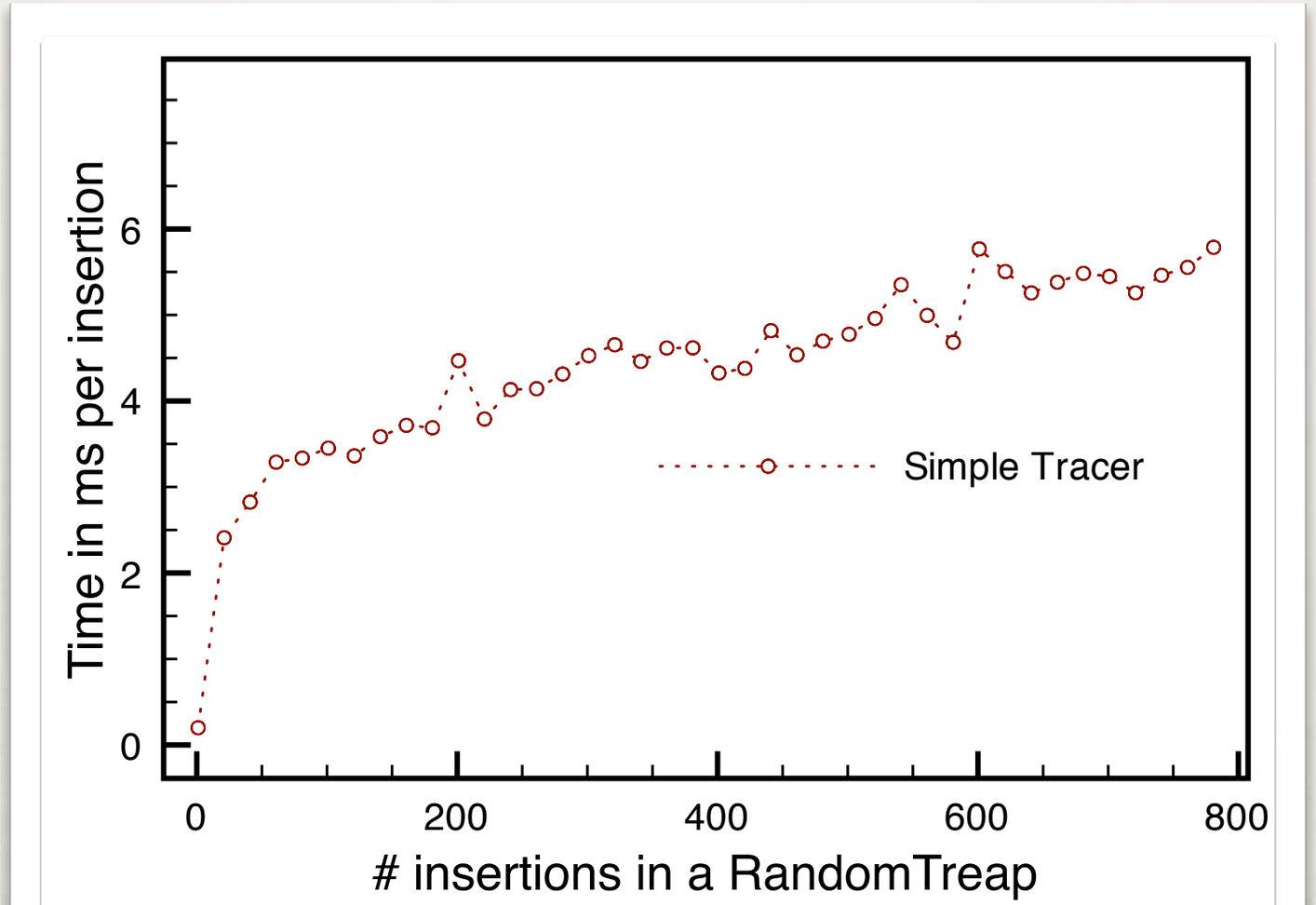
- We implemented
  - Eiffel-like Checked Post-Conditions for Smalltalk
  - Execution tracer with states
  - Planar Point Location

Benchmarks



Stores messages sent in a collection

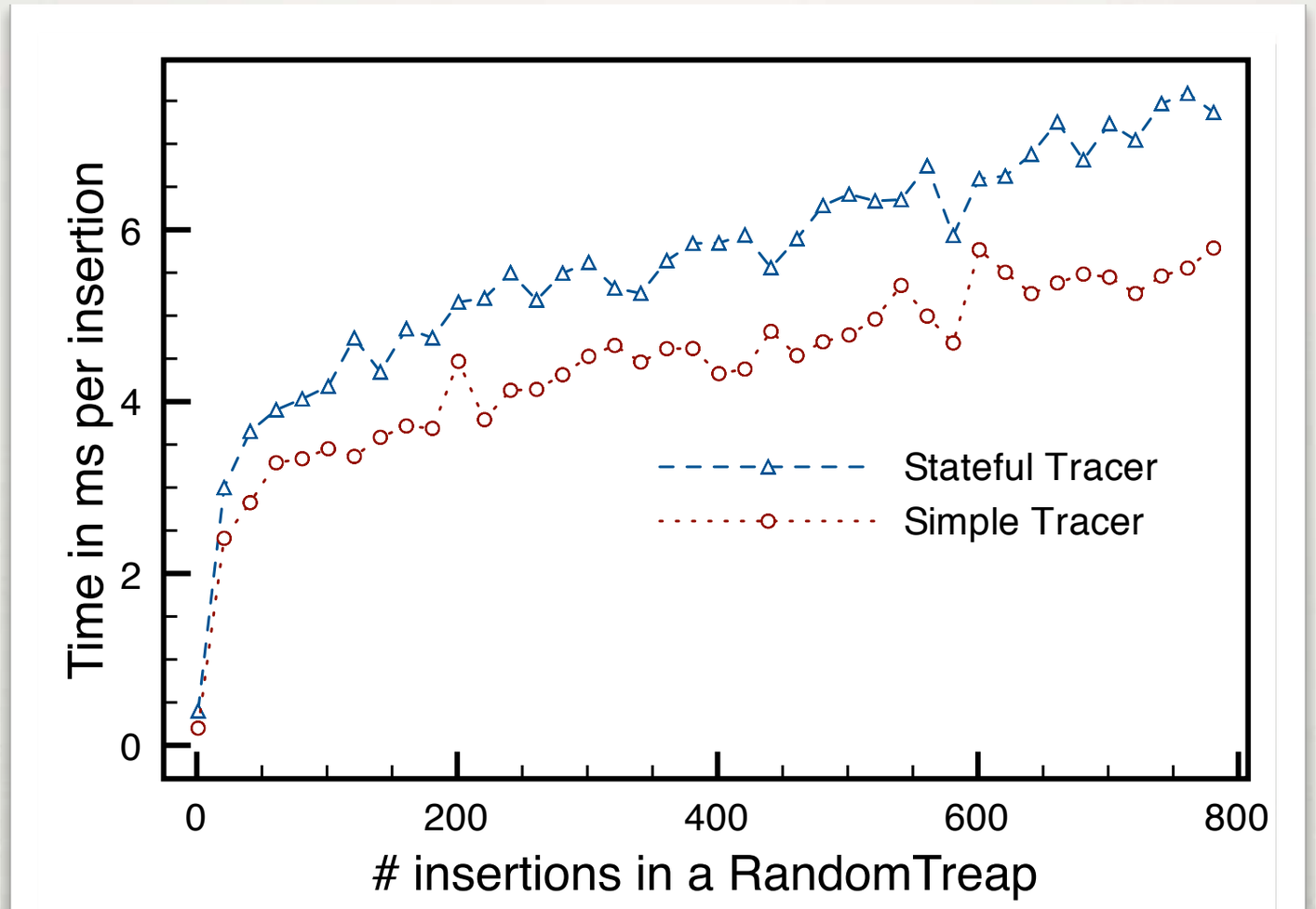
Benchmarks



Stores messages sent in a collection

Also stores states of receiver and arguments

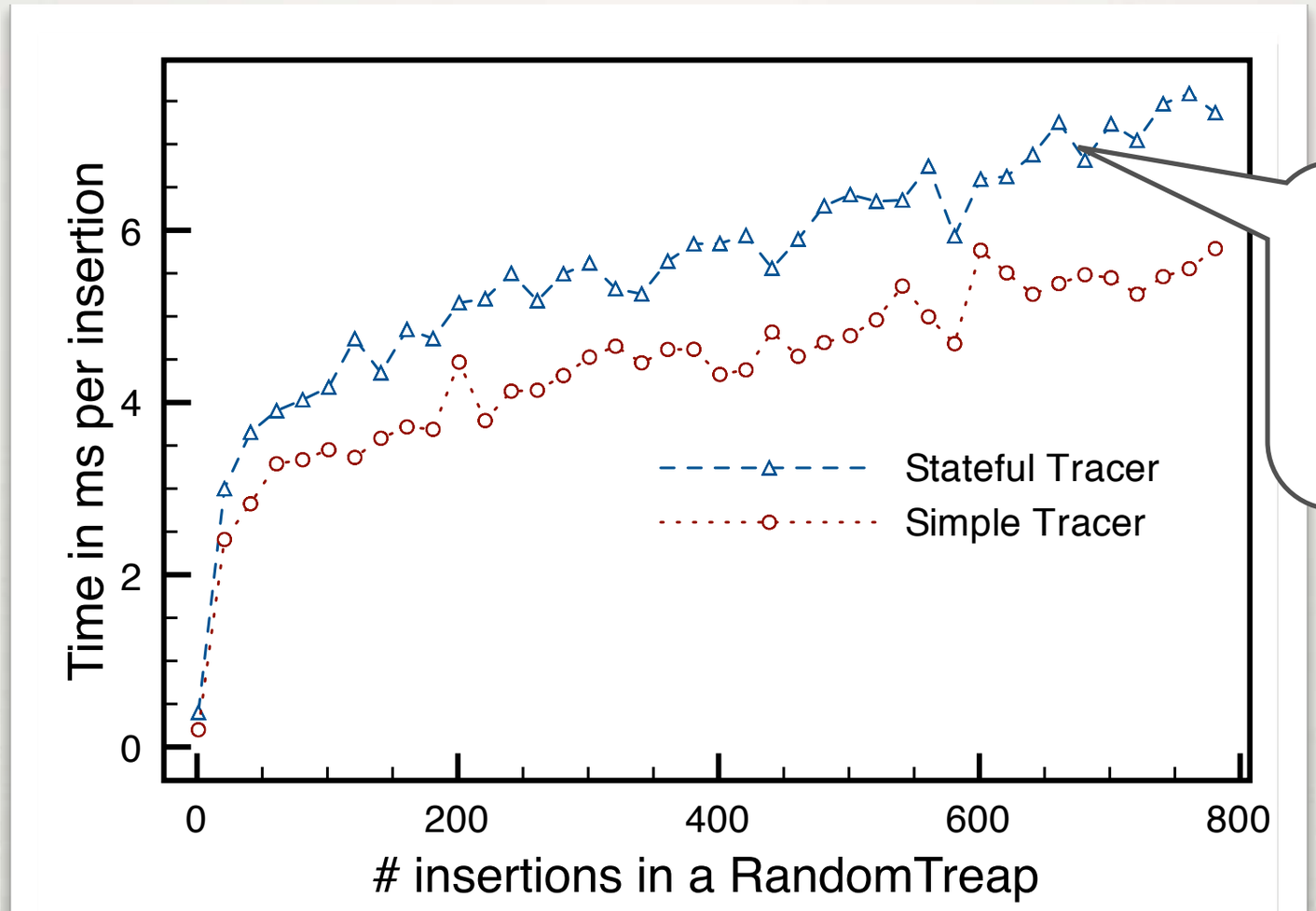
Benchmarks



Stores messages sent in a collection

Also stores states of receiver and arguments

Benchmarks



Only 30% slower !

Stores messages sent in a collection

Also stores states of receiver and arguments

Conclusion

- A general and efficient model for in-memory object versioning for object-oriented languages
- We implemented it in Smalltalk
  - 3 primitives
  - Non-intrusive library
  - Efficient

Thank you !

- fpluquet@ulb.ac.be
- HistOOry 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.