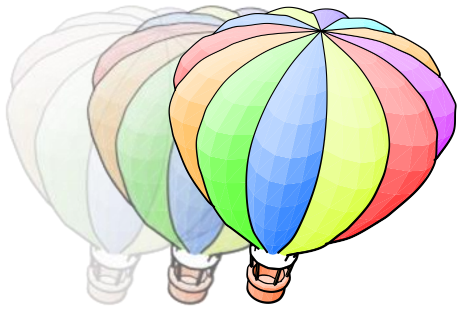


Object-centric Time-Travel Debugging

Exploring Traces of Objects

Christoph Thiede, Marcel Taeumel, and Robert Hirschfeld

Software Architecture Group

Hasso Plattner Institute, Potsdam, Germany

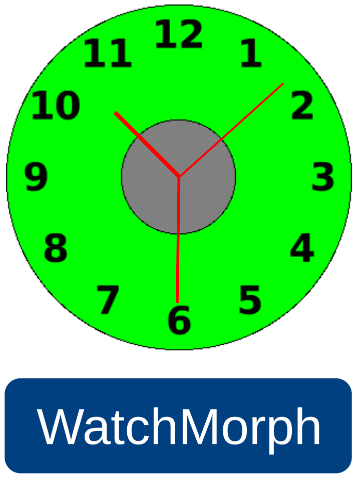
https://hpi.de/swa

Motivation: Object-oriented Programming

- Object-oriented programming (OOP): model systems as objects that exchange messages

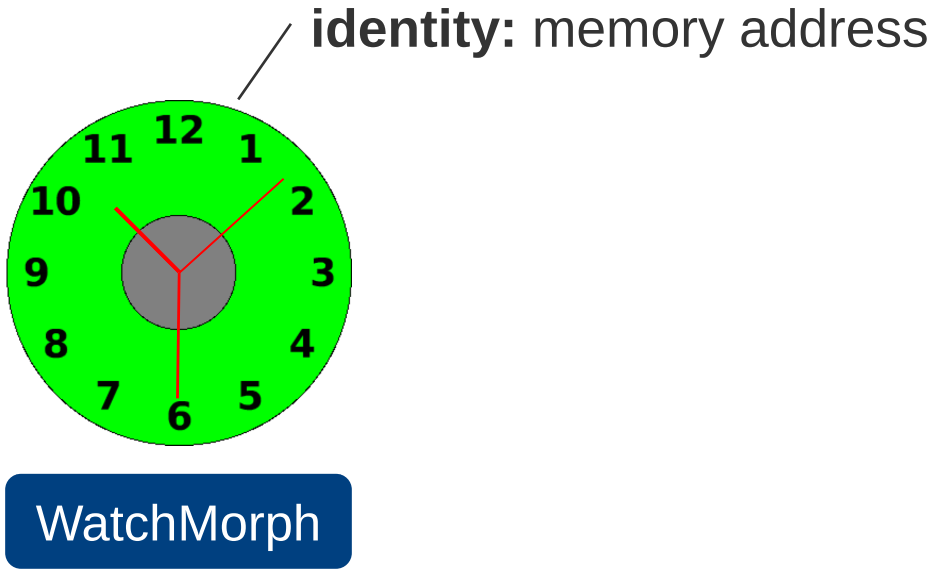
Motivation: Object-oriented Programming

- Object-oriented programming (OOP): model systems as objects that exchange messages



Motivation: Object-oriented Programming

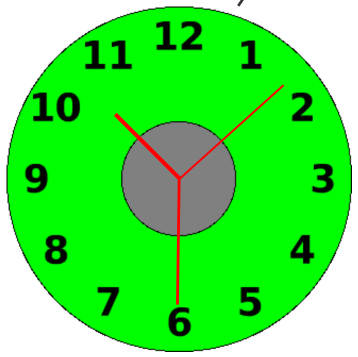
- Object-oriented programming (OOP): model systems as objects that exchange messages



Motivation: Object-oriented Programming

- Object-oriented programming (OOP): model systems as objects that exchange messages

identity: memory address



state: variables

color

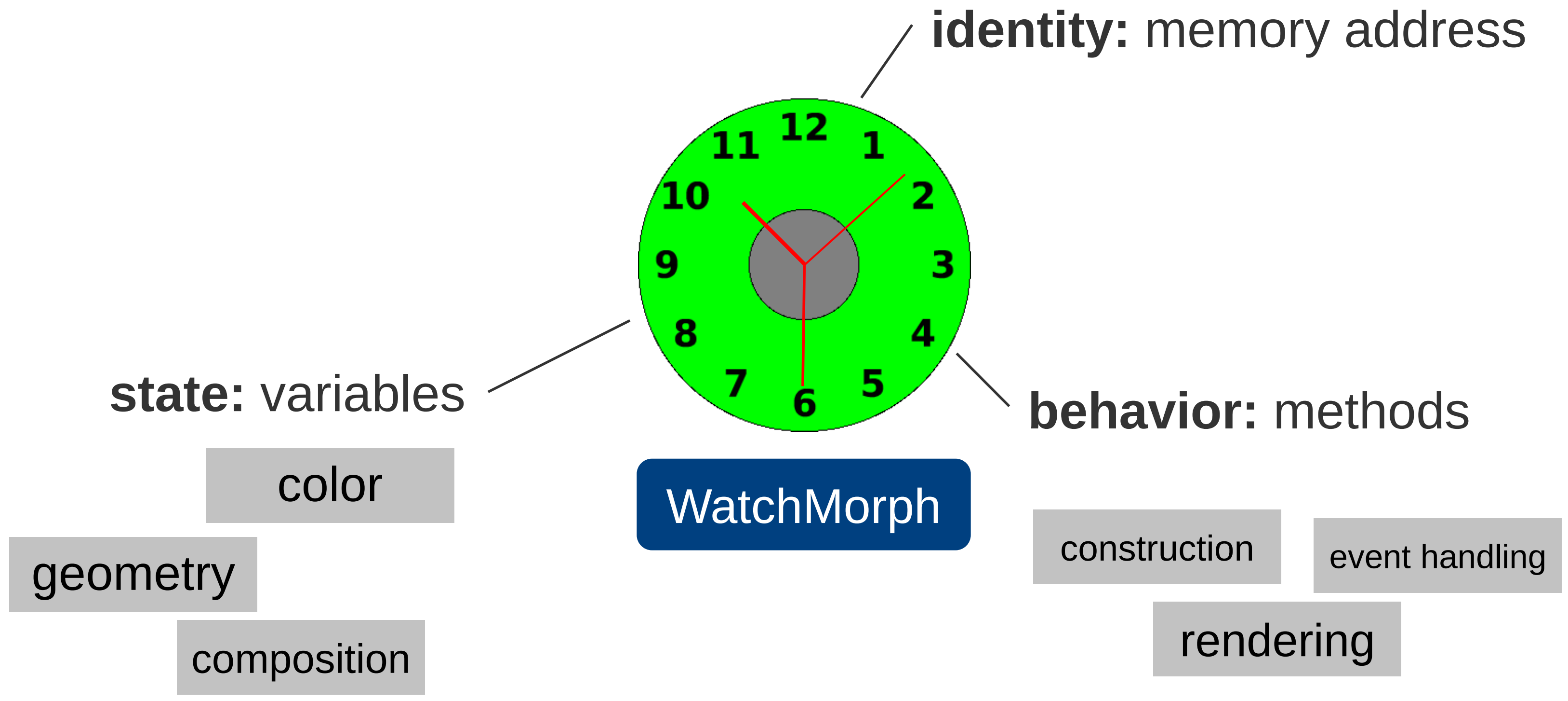
geometry

composition

WatchMorph

Motivation: Object-oriented Programming

- Object-oriented programming (OOP): model systems as objects that exchange messages

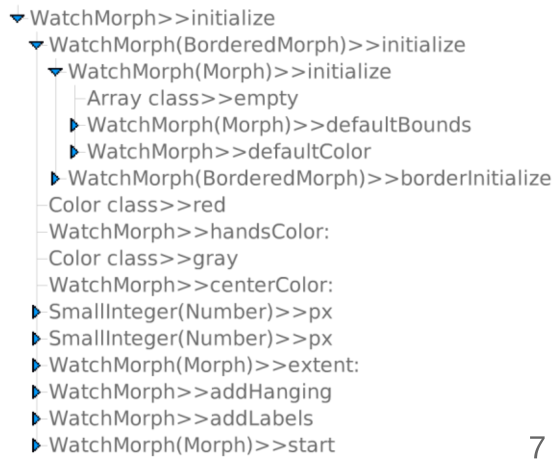


Motivation: Debugging

- Debugging: understand the behavior of objects
  - identify the cause of a bug, discover potential extensions points, …
  - directly interact with objects to explore their current state and behavior
- Time-travel debugging: record and replay program execution
  - time-independent exploration of program trace
  - program trace: prior states and method activations
- Still, context trees can become very large …

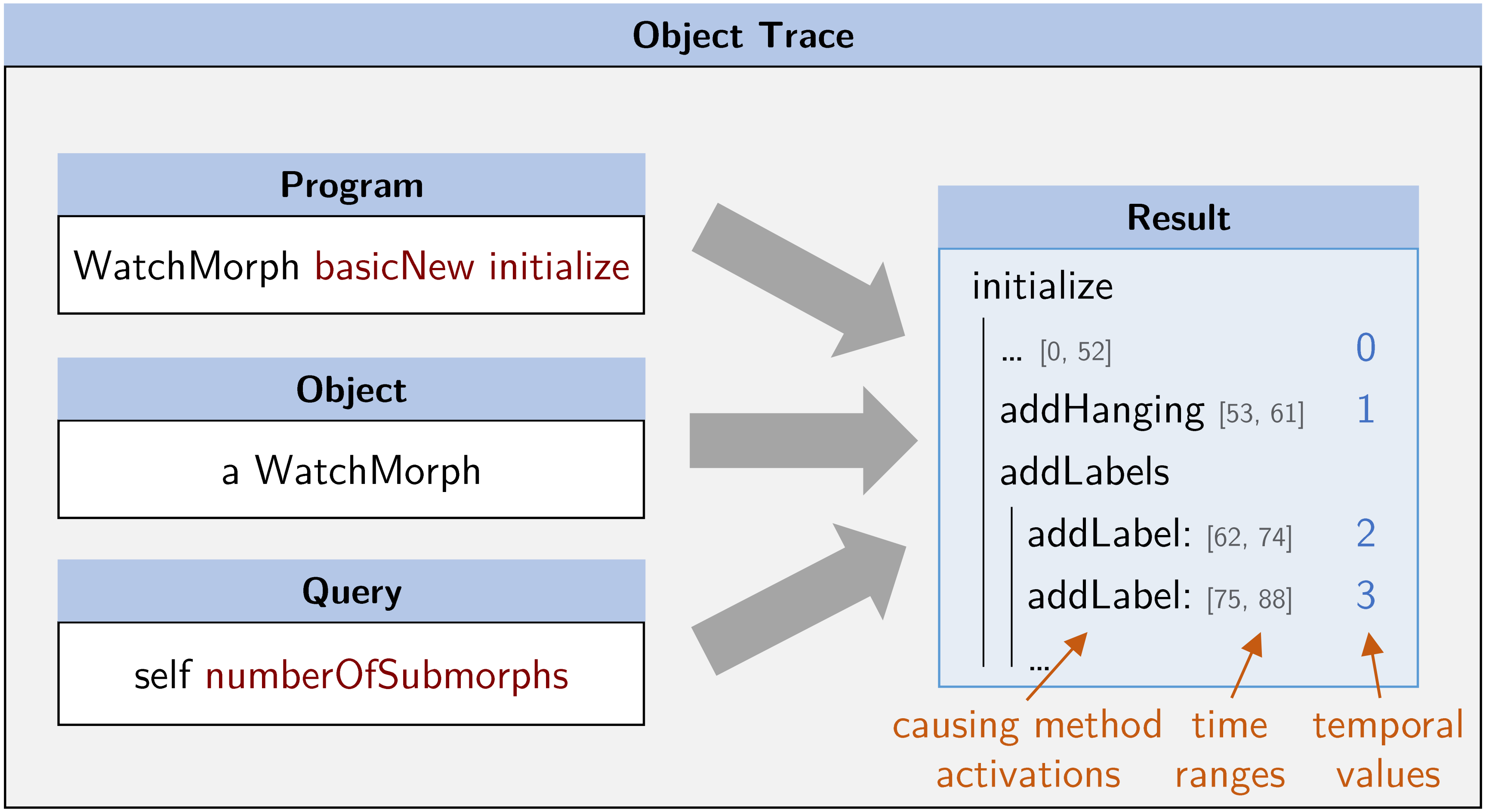
Program

```
WatchMorph basicNew initialize
```



```
▾ WatchMorph>>initialize
  ▾ WatchMorph(BorderedMorph)>>initialize
    ▾ WatchMorph(Morph)>>initialize
        Array class>>empty
      ▸ WatchMorph(Morph)>>defaultBounds
      ▸ WatchMorph>>defaultColor
    ▸ WatchMorph(BorderedMorph)>>borderInitialize
    Color class>>red
    WatchMorph>>handsColor:
    Color class>>gray
    WatchMorph>>centerColor:
  ▸ SmallInteger(Number)>>px
  ▸ SmallInteger(Number)>>px
  ▸ WatchMorph(Morph)>>extent:
  ▸ WatchMorph>>addHanging
  ▸ WatchMorph>>addLabels
  ▸ WatchMorph(Morph)>>start
```

Solution

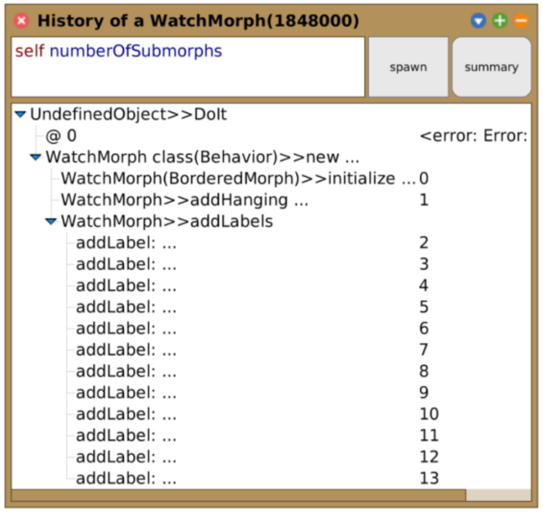

Object Trace
Program
WatchMorph basicNew initialize
Object
a WatchMorph
Query
self numberOfSubmorphs
Result
initialize
... [0, 52] 0
addHanging [53, 61] 1
addLabels
addLabel: [62, 74] 2
addLabel: [75, 88] 3
...
causing method activations
time ranges
temporal values

Solution: Object Trace

- Interactive, tangible object
- Dynamic structure and granularity defined by the query
- Query:
  - select interesting portions of state
  - single variables or complex state
  - domain-specific accessors and representations (moldable tool)

Demo: TraceDebugger

(Squeak/Smalltalk)

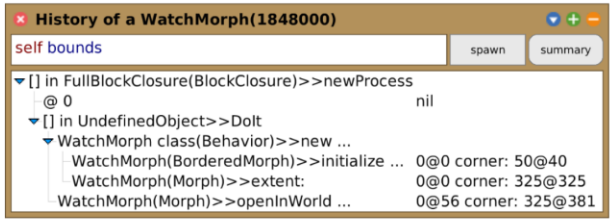


History of a WatchMorph(1848000)

self numberOfSubmorphs

```
▾ UndefinedObject>>DoIt
    @ 0                                          <error: Error:
  ▾ WatchMorph class(Behavior)>>new ...
      WatchMorph(BorderedMorph)>>initialize ...  0
      WatchMorph>>addHanging ...                 1
    ▾ WatchMorph>>addLabels
        addLabel: ...                            2
        addLabel: ...                            3
        addLabel: ...                            4
        addLabel: ...                            5
        addLabel: ...                            6
        addLabel: ...                            7
        addLabel: ...                            8
        addLabel: ...                            9
        addLabel: ...                            10
        addLabel: ...                            11
        addLabel: ...                            12
        addLabel: ...                            13
```

(a) Viewing the entire history for the number of submorphs.

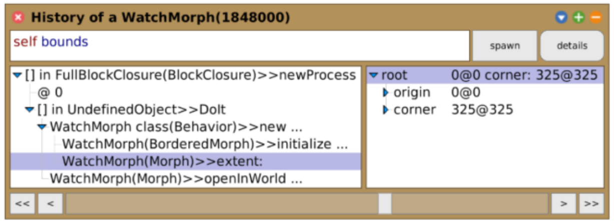


History of a WatchMorph(1848000)

self bounds

```
▾ [] in FullBlockClosure(BlockClosure)>>newProcess
    @ 0                                          nil
  ▾ [] in UndefinedObject>>DoIt
    ▾ WatchMorph class(Behavior)>>new ...
        WatchMorph(BorderedMorph)>>initialize ...  0@0 corner: 50@40
        WatchMorph(Morph)>>extent:                 0@0 corner: 325@325
      WatchMorph(Morph)>>openInWorld ...           0@56 corner: 325@381
```

(b) Viewing the morph's geometry after opening it.

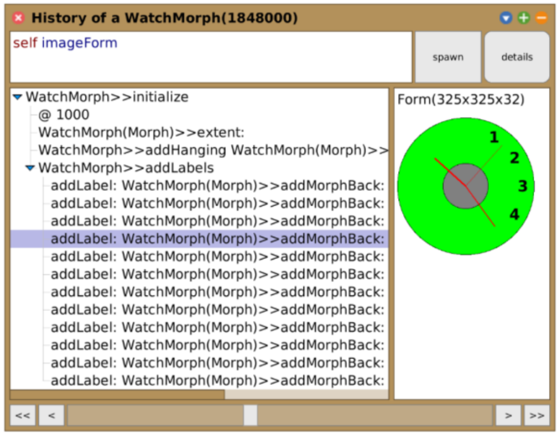


History of a WatchMorph(1848000)

self bounds

```
▾ [] in FullBlockClosure(BlockClosure)>>newProcess      ▾ root     0@0 corner: 325@325
    @ 0                                                   ▸ origin  0@0
  ▾ [] in UndefinedObject>>DoIt                           ▸ corner  325@325
    ▾ WatchMorph class(Behavior)>>new ...
        WatchMorph(BorderedMorph)>>initialize ...
        WatchMorph(Morph)>>extent:
      WatchMorph(Morph)>>openInWorld ...
```

(c) Inspecting the morph's geometry after opening it, evolved after the third change.



History of a WatchMorph(1848000)

self imageForm

```
▾ WatchMorph>>initialize                                  Form(325x325x32)
    @ 1000
    WatchMorph(Morph)>>extent:
    WatchMorph>>addHanging WatchMorph(Morph)>>
  ▾ WatchMorph>>addLabels
      addLabel: WatchMorph(Morph)>>addMorphBack:
      addLabel: WatchMorph(Morph)>>addMorphBack:
      addLabel: WatchMorph(Morph)>>addMorphBack:
      addLabel: WatchMorph(Morph)>>addMorphBack:
      addLabel: WatchMorph(Morph)>>addMorphBack:
      addLabel: WatchMorph(Morph)>>addMorphBack:
      addLabel: WatchMorph(Morph)>>addMorphBack:
      addLabel: WatchMorph(Morph)>>addMorphBack:
      addLabel: WatchMorph(Morph)>>addMorphBack:
      addLabel: WatchMorph(Morph)>>addMorphBack:
      addLabel: WatchMorph(Morph)>>addMorphBack:
      addLabel: WatchMorph(Morph)>>addMorphBack:
```

(d) Inspecting a rendered screenshot of the morph after constructing the fourth label. All render errors (due to uninitialized variables) have been excluded through a filter.

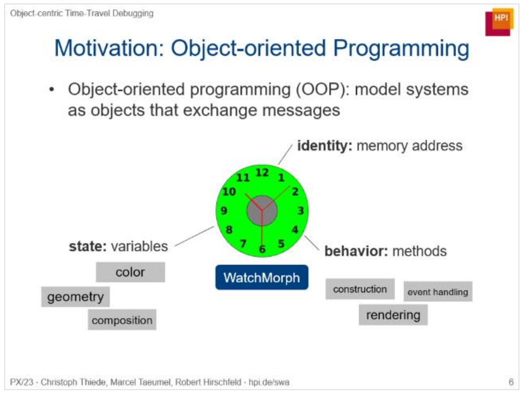
Discussion

+	–
• efficiently find sources of side effects • explore evolution of object graphs	• cognitive investment for writing queries • inconvenient for convoluted state models

- for the best experience, combine behavior-centric and object-centric views

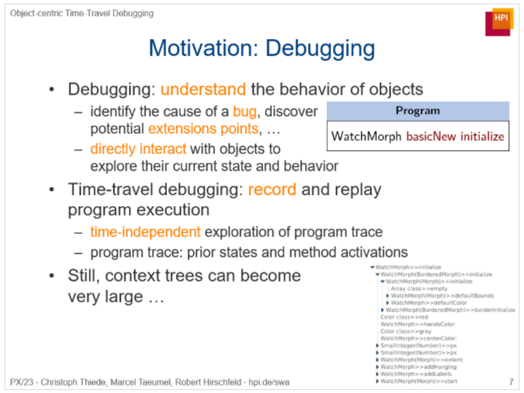
- performance of prototype:
  - tracing overhead: ≤1,000,000%
  - query overhead: ≤100,000%
  - still, practical response times (≤5 secs) for medium workloads

Object-centric Time-Travel Debugging



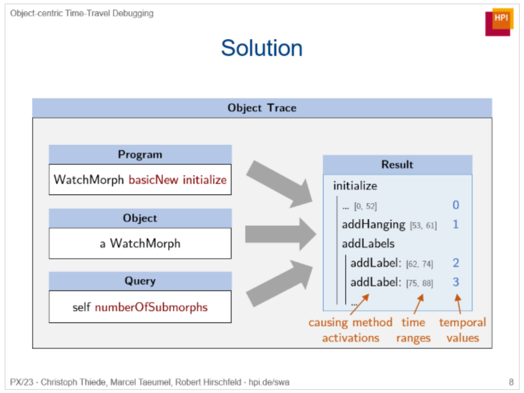
Motivation: Object-oriented Programming

- Object-oriented programming (OOP): model systems as objects that exchange messages

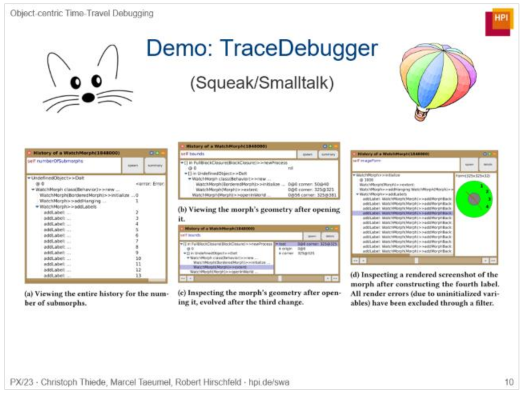


Motivation: Debugging

- Debugging: understand the behavior of objects
  - identify the cause of a bug, discover potential extensions points, …
  - directly interact with objects to explore their current state and behavior
- Time-travel debugging: record and replay program execution
  - time-independent exploration of program trace
  - program trace: prior states and method activations
- Still, context trees can become very large …



Solution



Demo: TraceDebugger

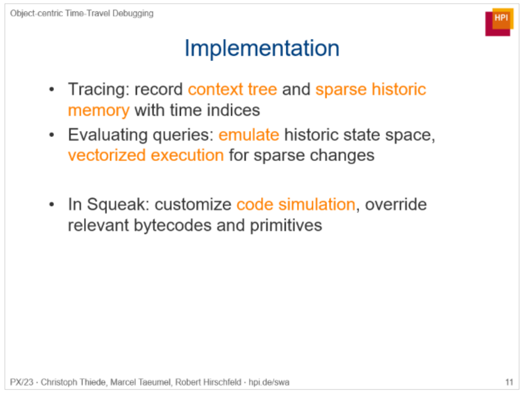
(Squeak/Smalltalk)

(a) Viewing the entire history for the number of submorphs.

(b) Viewing the morph's geometry after opening it.

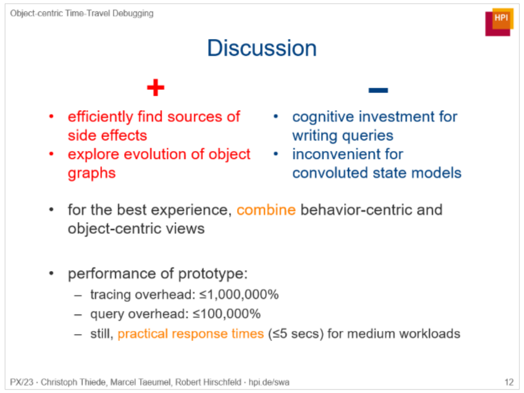
(c) Inspecting the morph's geometry after opening it, evolved after the third change.

(d) Inspecting a rendered screenshot of the morph after constructing the fourth label. All render errors (due to uninitialized variables) have been excluded through a filter.



Implementation

- Tracing: record context tree and sparse historic memory with time indices
- Evaluating queries: emulate historic state space, vectorized execution for sparse changes
- In Squeak: customize code simulation, override relevant bytecodes and primitives



Discussion

+
- efficiently find sources of side effects
- explore evolution of object graphs

–
- cognitive investment for writing queries
- inconvenient for convoluted state models

- for the best experience, combine behavior-centric and object-centric views
- performance of prototype:
  - tracing overhead: ≤1,000,000%
  - query overhead: ≤100,000%
  - still, practical response times (≤5 secs) for medium workloads

https://github.com/hpi-swa-lab/squeak-tracedebugger

Thank you!