Object Versioning for the Lively Kernel

Preserving Access to Previous System States in an Object-oriented Programming System

Master's Thesis

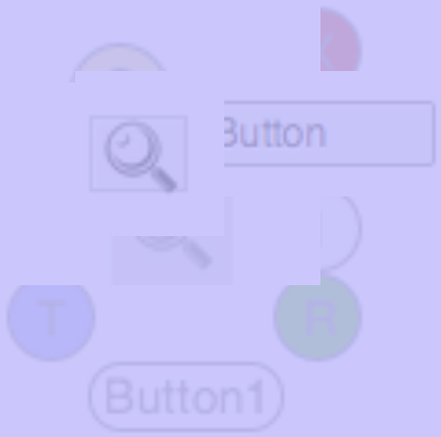
Lauritz Thamsen

Prof. Dr. Robert Hirschfeld
Bastian Steinert

June 04, 2014

blank.html

localhost:9001/blank.html



Workspace

Menu

```
$morph('MagnifierButton').addScript(function onMouseMove(evt) {
    var target = this.currentTarget();
    if (target && target.world() && !this.isHighlighting()) {
        this.getHighlightRectangle().update(target);
        this.world().addMorph(this.getHighlightRectangle());
        this.getHighlightRectangle().bringToFront();
        this.targetHighlight = this.getHighlightRectangle();
    }
}).tag(□);
```

ObjectEditor

Menu

Tag: all | <lively.morphic.World#1> | Tests | save | run

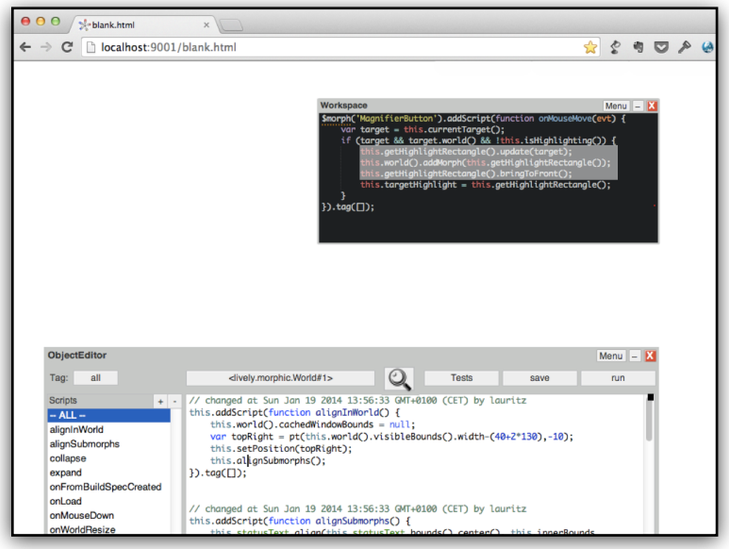
Scripts + -

- ALL -
alignInWorld
alignSubmorphs
collapse
expand
onFromBuildSpecCreated
onLoad
onMouseDown
onWorldResize

```
// changed at Sun Jan 19 2014 13:56:33 GMT+0100 (CET) by lauritz
this.addScript(function alignInWorld() {
    this.world().cachedWindowBounds = null;
    var topRight = pt(this.world().visibleBounds().width-(40+2*130),-10);
    this.setPosition(topRight);
    this.alignSubmorphs();
}).tag(□);


// changed at Sun Jan 19 2014 13:56:33 GMT+0100 (CET) by lauritz
this.addScript(function alignSubmorphs() {
    this.statusText.align(this.statusText.bounds().center(), this.innerBounds
```

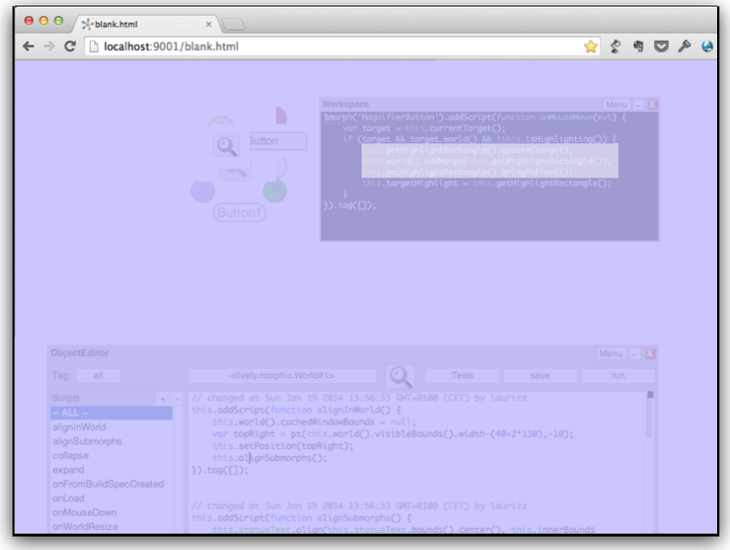
Changed States

State 1



: World
extent = *aPoint* (x: 800, y: 600) submorphs = [… …]

State 2



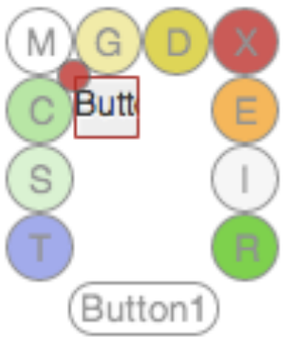
: World
extent = *aPoint* (x: 800, y: 600) submorphs = [… ***aRectangle*** *…*]

How to undo changes to objects?

Command Pattern

> Encapsulate a request as an object, thereby letting you [..] queue or log requests, and support undoable operations
>
> [Gamma, et al.]

but: each action needs to be accompanied by an undo action



Workspace — Menu

```
$morph('MagnifierButton').addScript(function onMouseMove(evt) {
    var target = this.currentTarget();
    if (target && target.world() && !this.isHighlighting()) {
        this.getHighlightRectangle().update(target);
        this.world().addMorph(this.getHighlightRectangle());
        this.getHighlightRectangle().bringToFront();
        this.targetHighlight = this.getHighlightRectangle();
    }
}).tag(□);
```

What about Worlds? *[Warth, et al.]*

a language construct to explicitly control the scope of side-effects to support experimentation

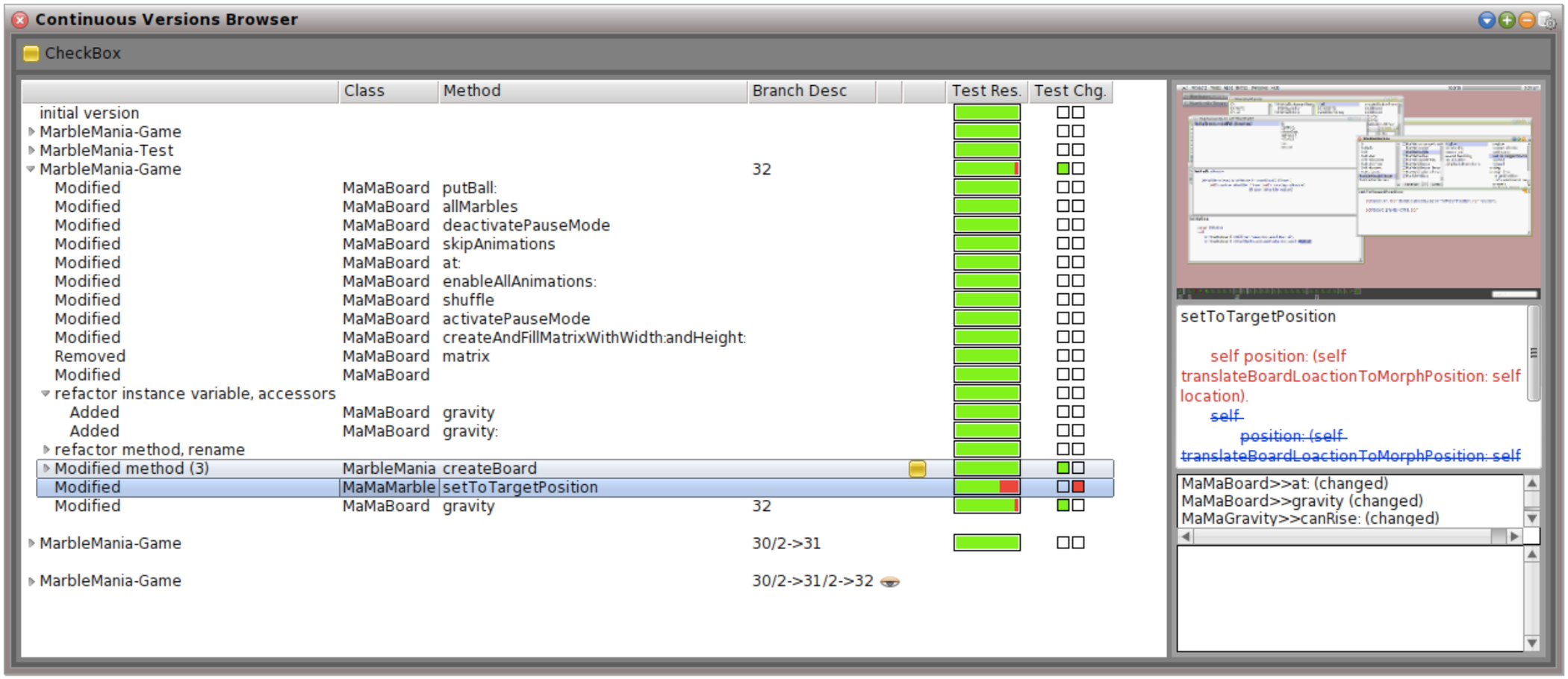
```
A = thisWorld;
p = new Point(1, 2);
```

in A:

```
p = {  x = 1,
       y = 23 }
```

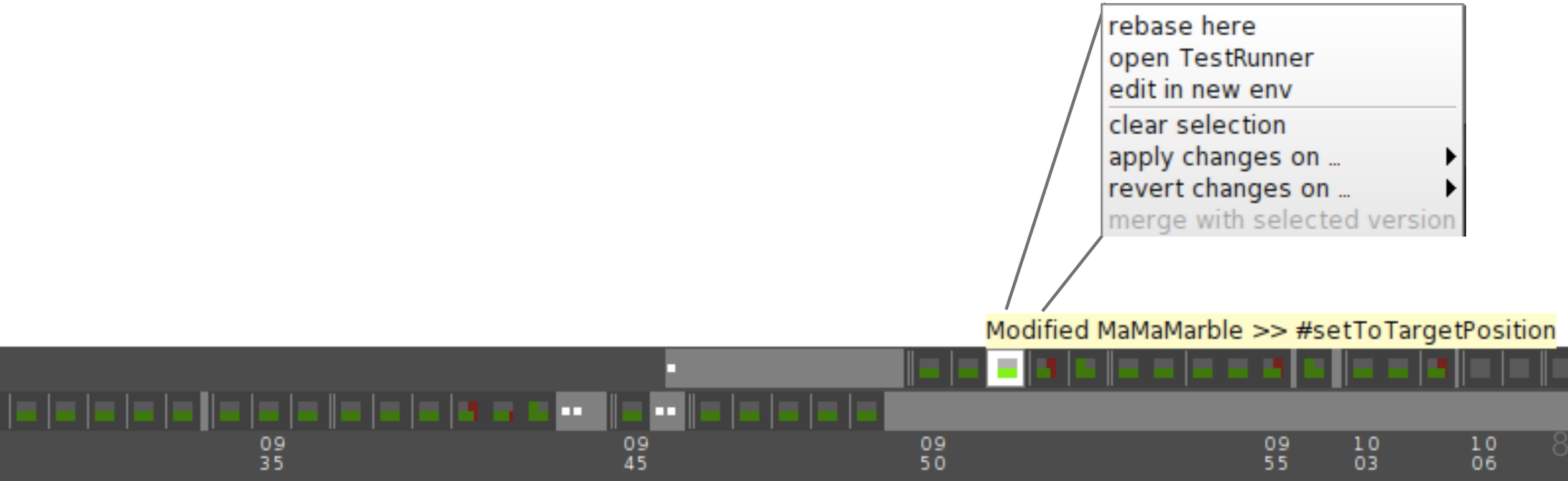
```
B = A.sprout();

in B { p.y = 3; }

B.commit();

delete B;
```

CoExist: Continuous Versioning *[Steinert, et al.]*



CoExist: Continuous Versioning *[Steinert, et al.]*

- recover previous development states whenever changes turn out to be inappropriate
- concentrate on implementing ideas instead of also having to protect intermediate development states



But Continuous Versioning of **code** is not enough for systems like the Lively Kernel

Project Questions

Goal: fine-grained versioning of all objects in Lively

- How can we implement fine-grained object versioning?
- Is that really practical?

Object Versioning

Versions of Objects

v1

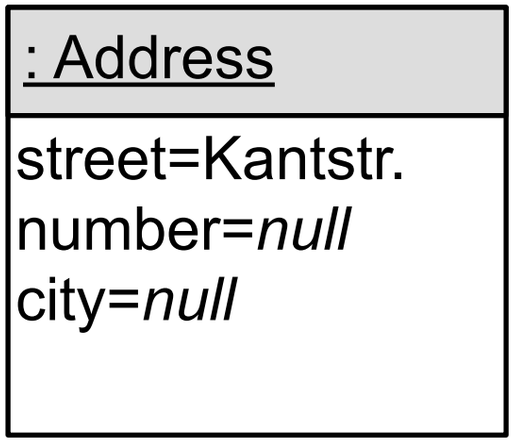
: Address
street=Kantstr. number=*null* city=*null*

v2

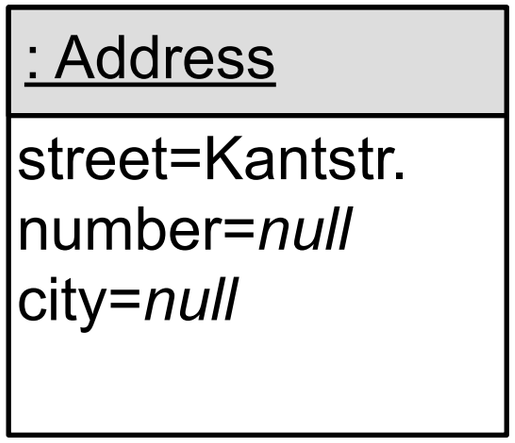
: Address
street=Kantstr. number=**148** city=**Berlin**

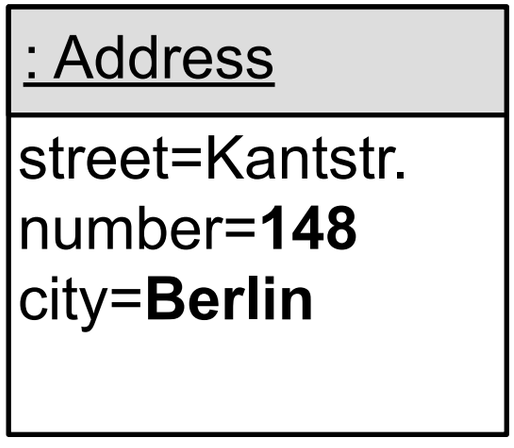
Preserving Previous Versions

v1



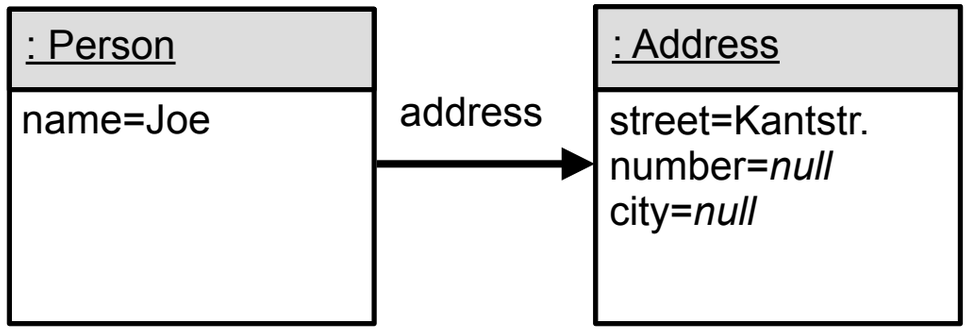
v2



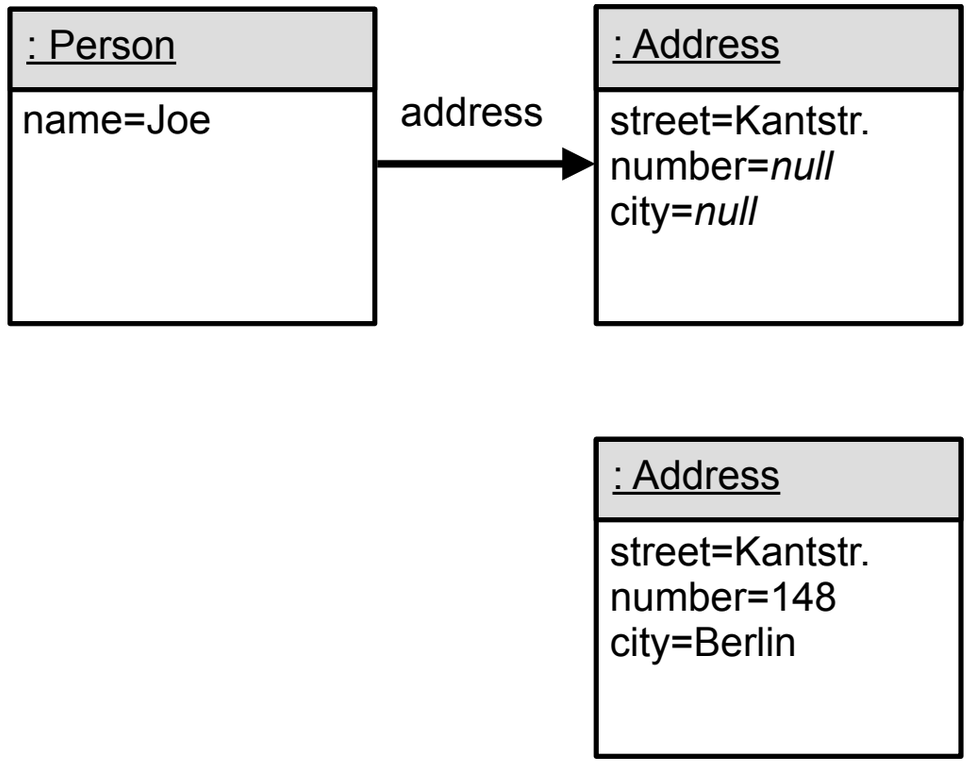


References to Objects

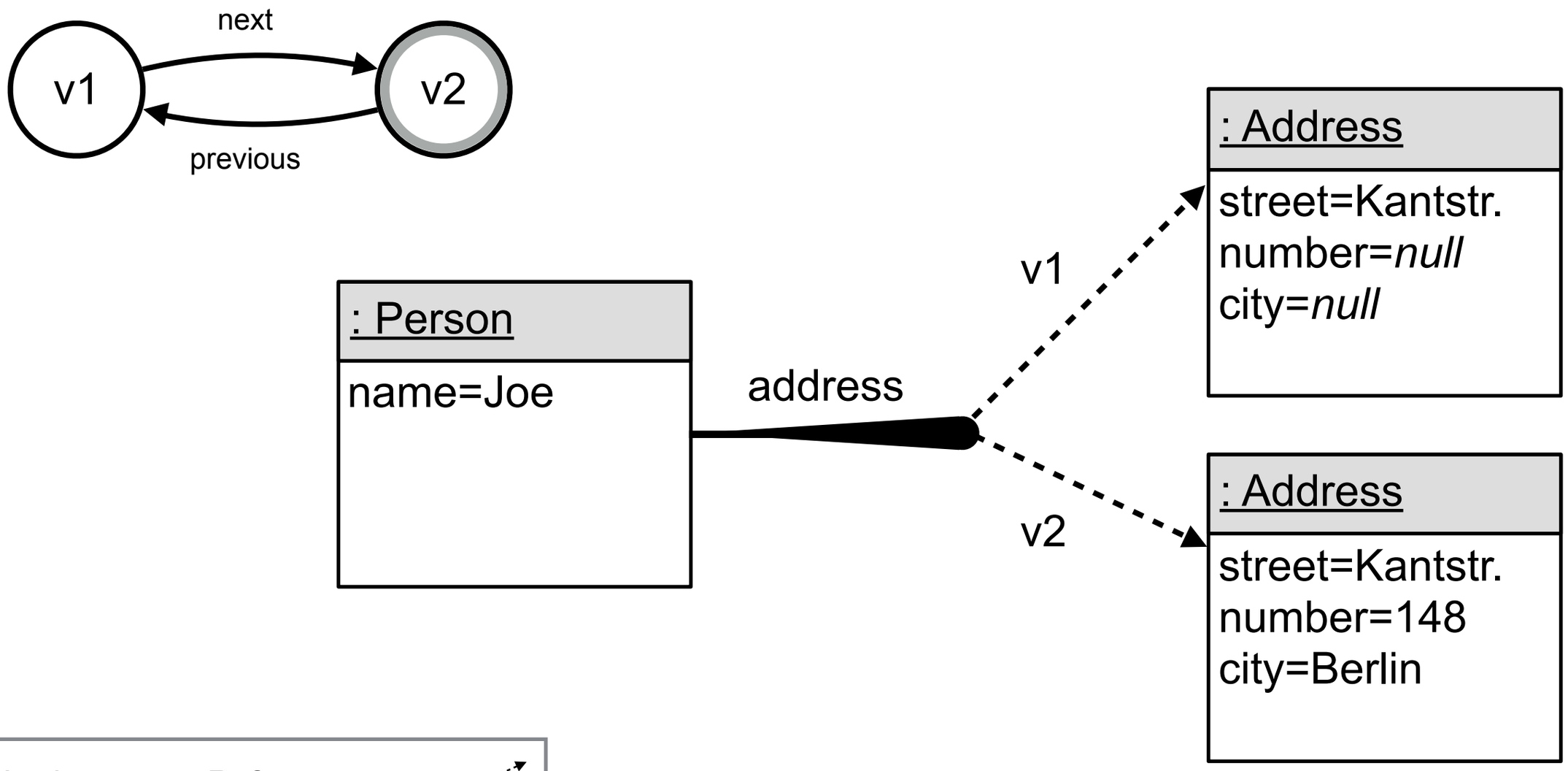
v1



v2



Version-aware References



Version-aware Reference

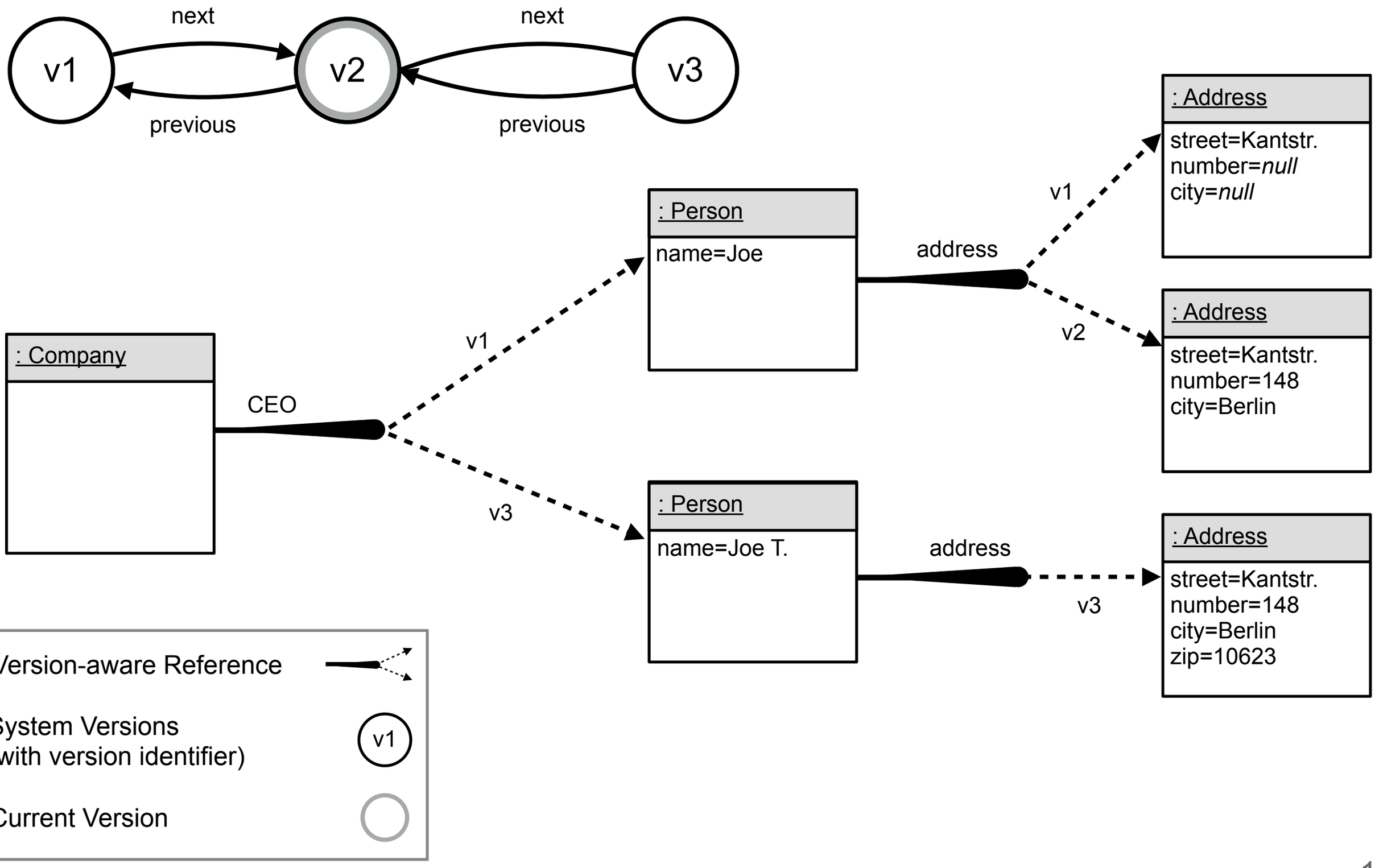
System Versions
(with version identifier)

v1

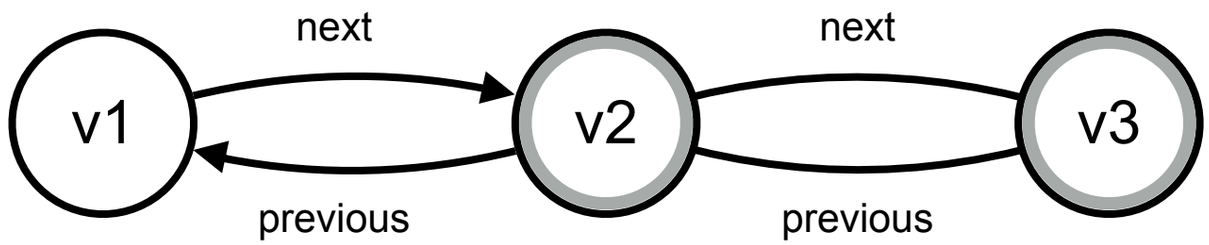
Current Version

```
aPerson.address.city === 'Berlin'
```

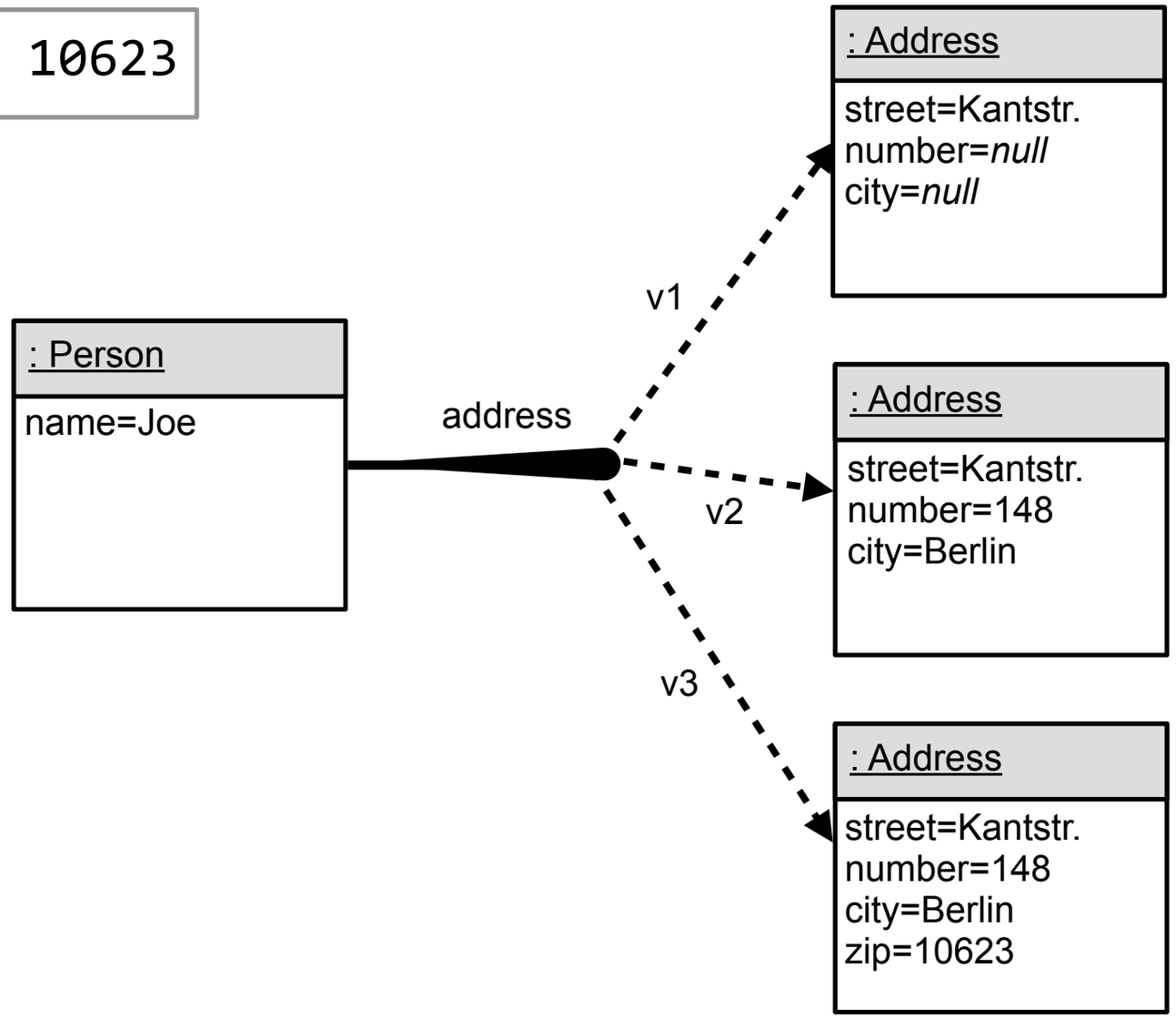
Version-aware References (cont.)

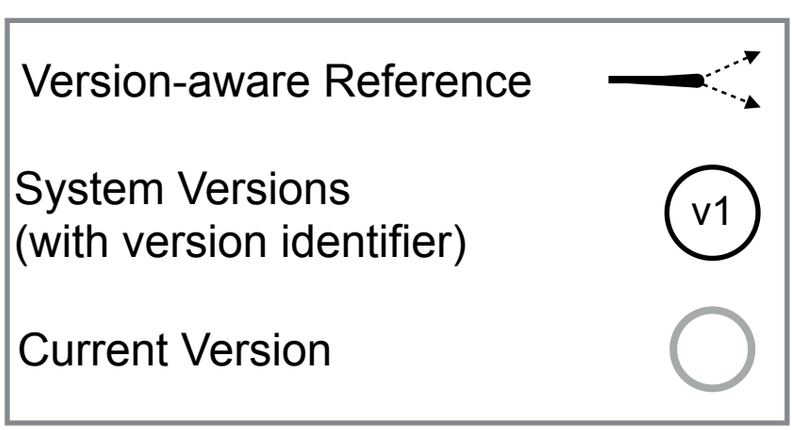


Explicit Creation of Discrete System Versions



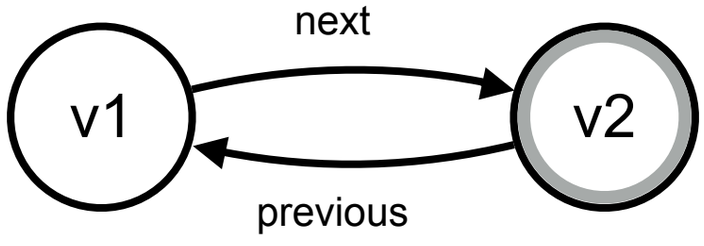

```
aPerson.address.zip = 10623
```



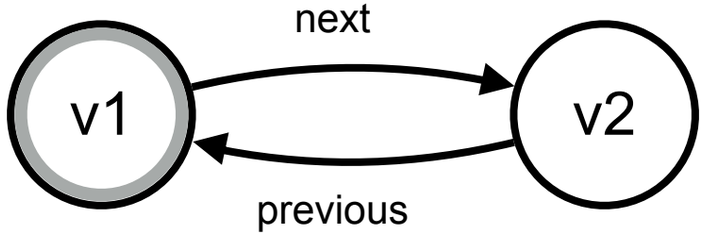


Changing Versions

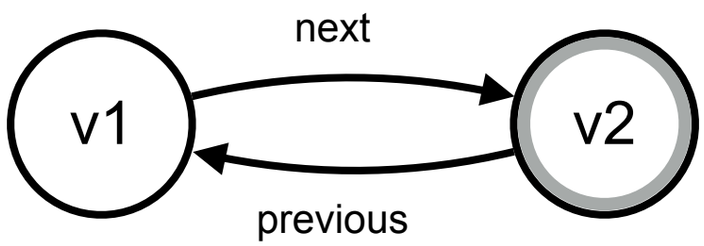
Given:



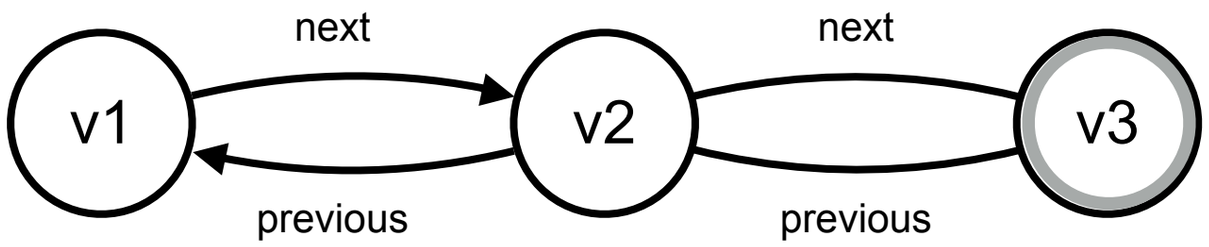
Undo:



Redo:

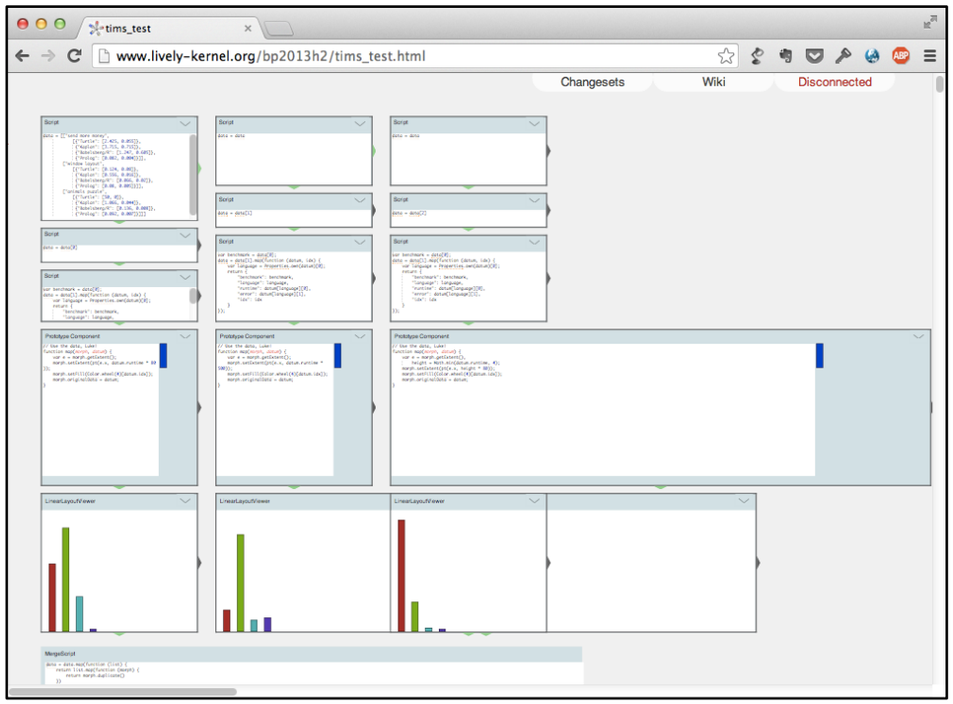


Commit:

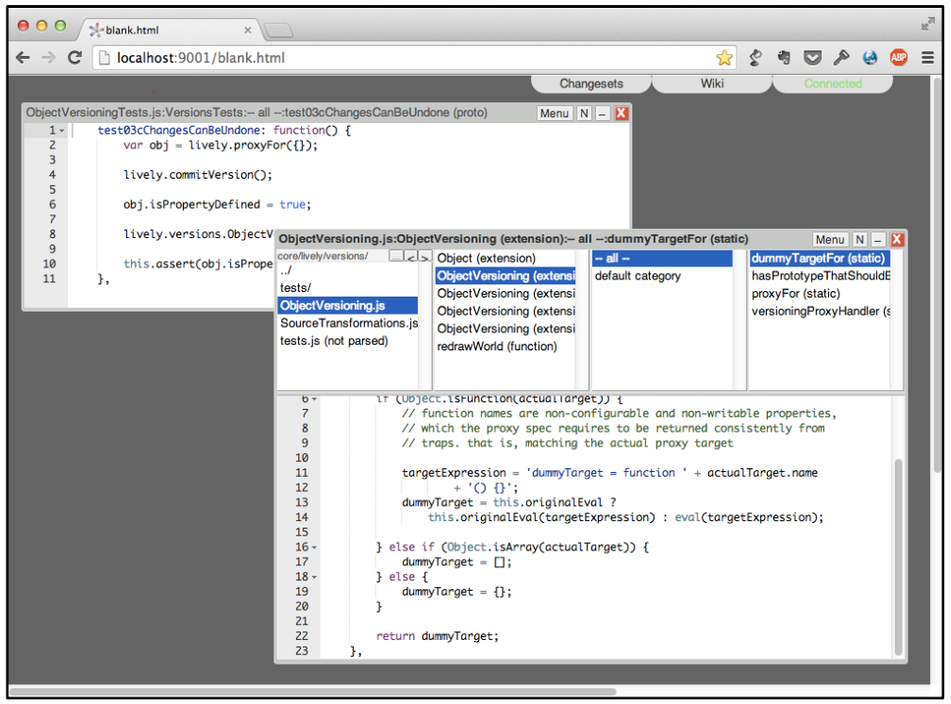


Generic Object Versioning provides versioning for all kinds of applications

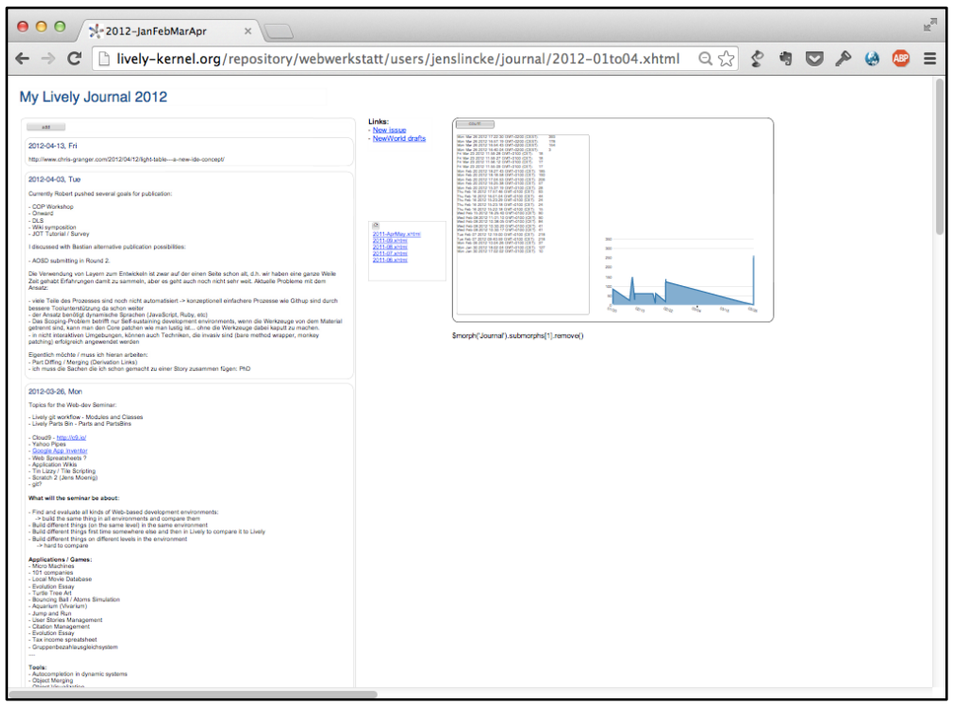
Scripts / Parts



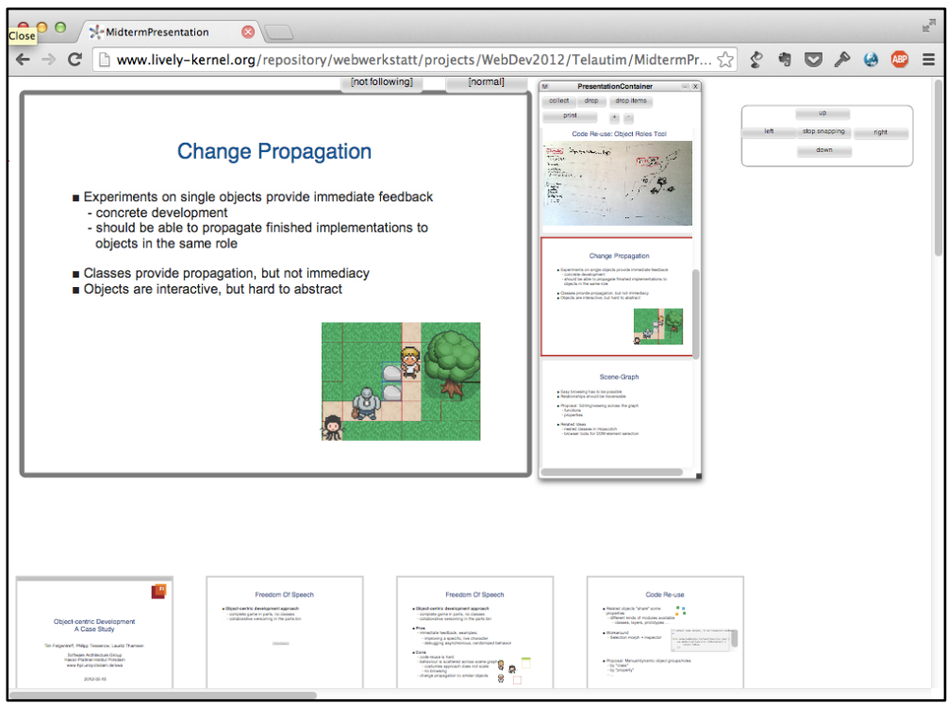
IDE / Tools



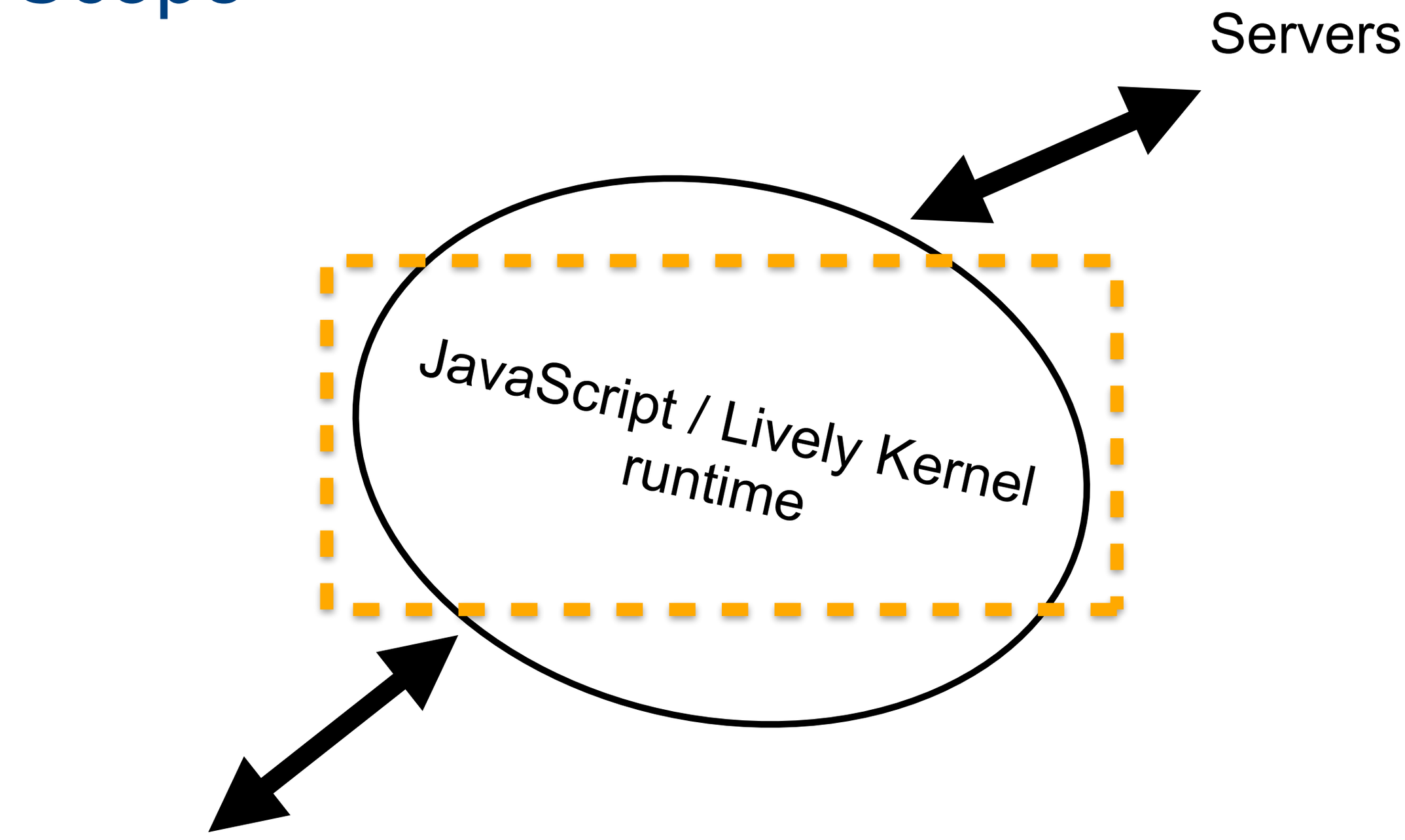
Wiki / Text



Applications / Games

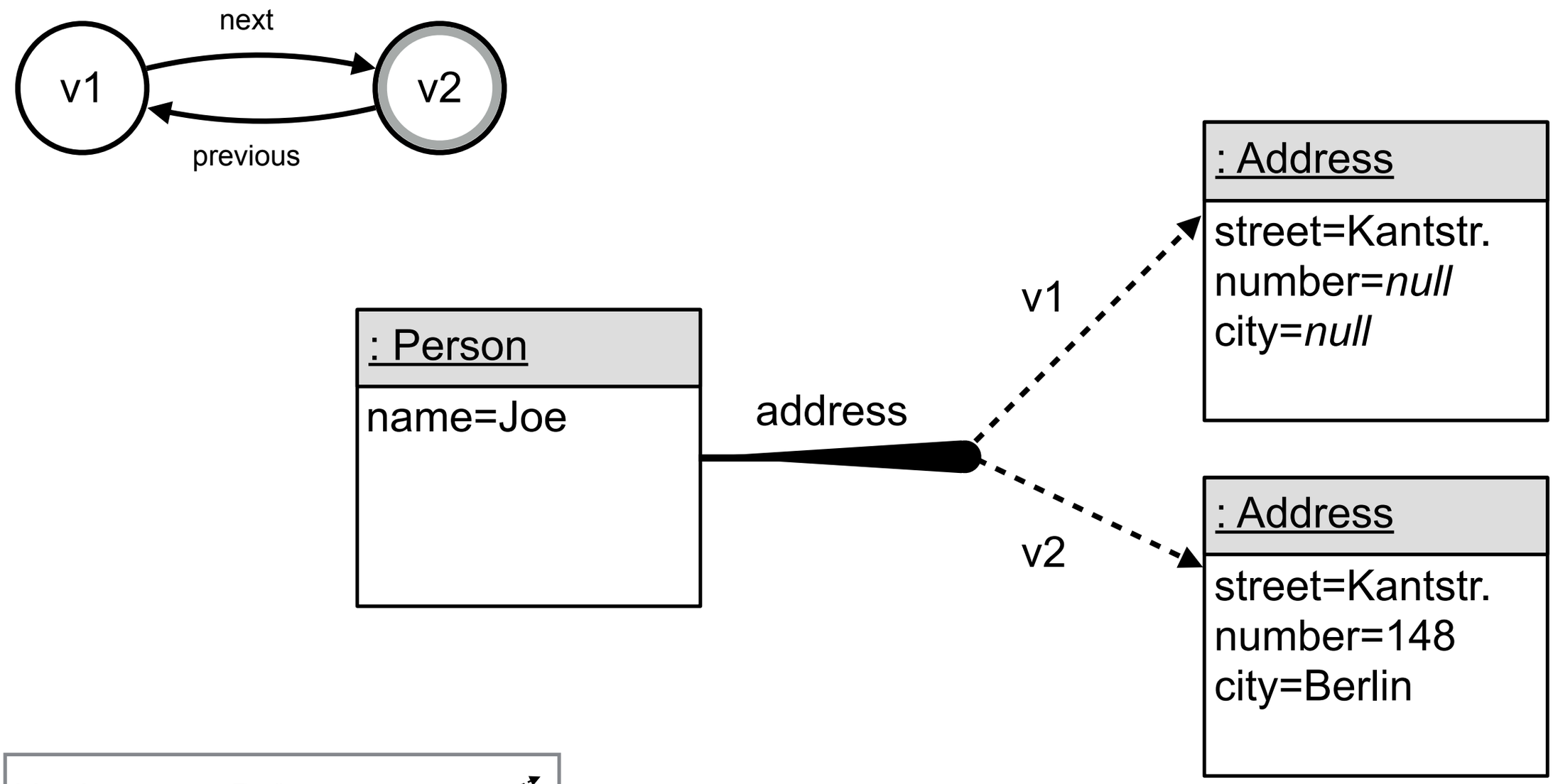


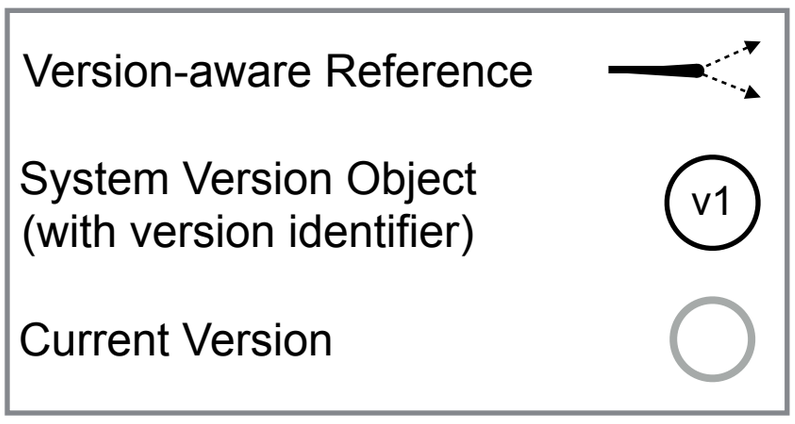
Scope


Servers
JavaScript / Lively Kernel runtime
Document / HTML page

Proxy-based Design & Implementation

Version-aware References (revisited)

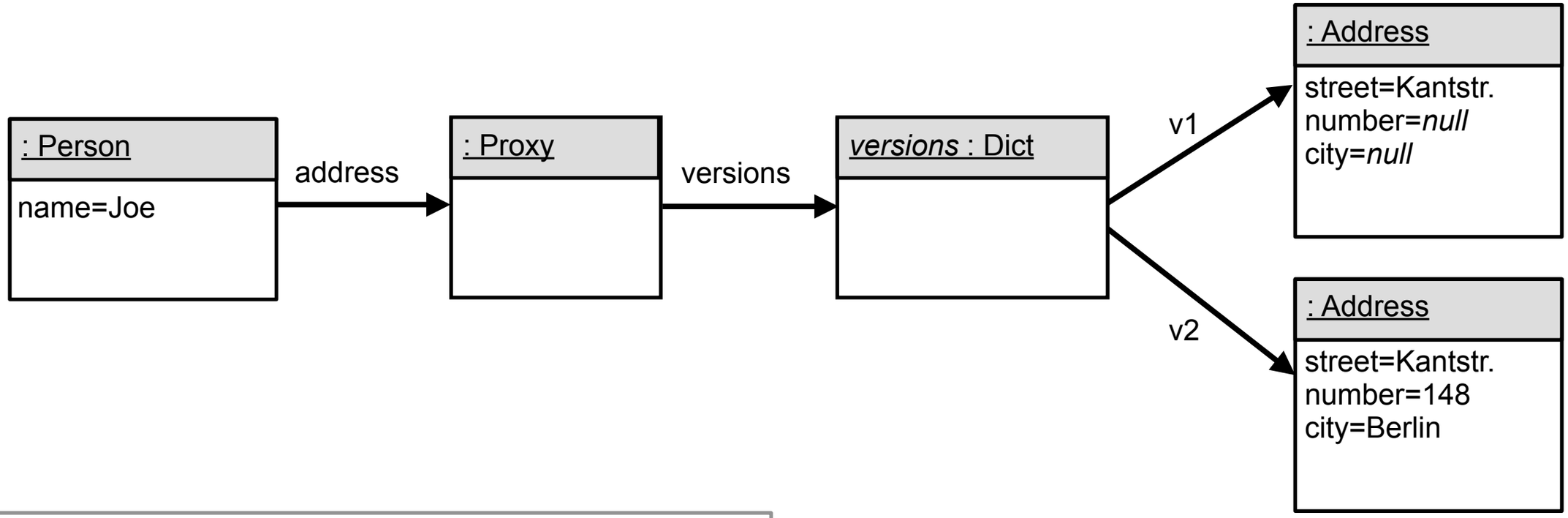




Proxies as Version-aware References

proxies delegate interactions transparently to the correct versions

```
lively.CurrentVersion = { ID: 'v2',
                          predecessor: …,
                          sucessor: … }
```

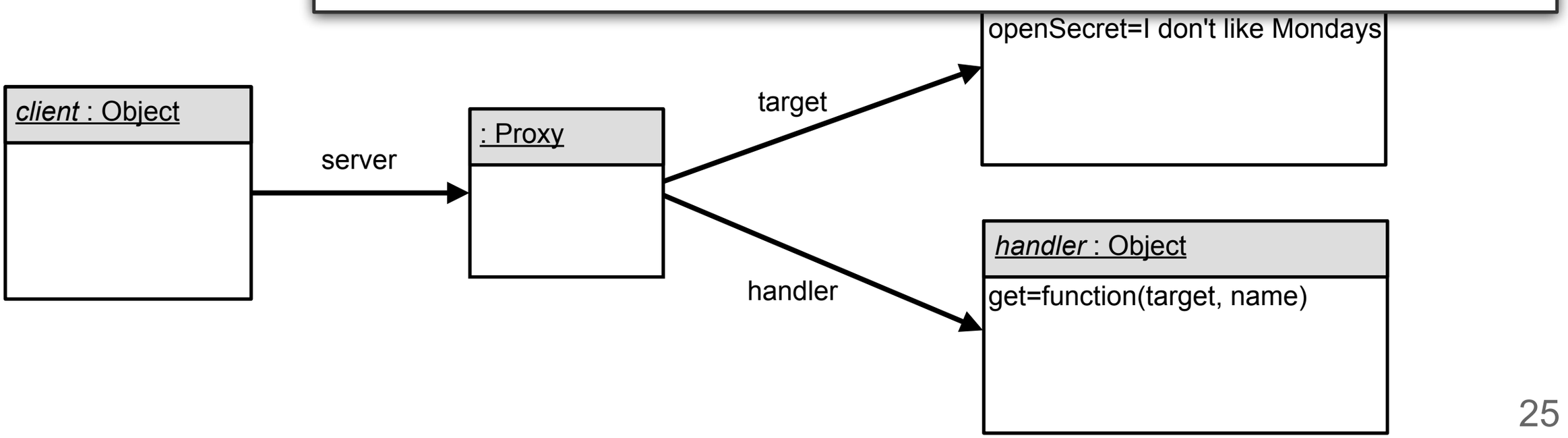



```
aPerson.address.city === 'Berlin'
```

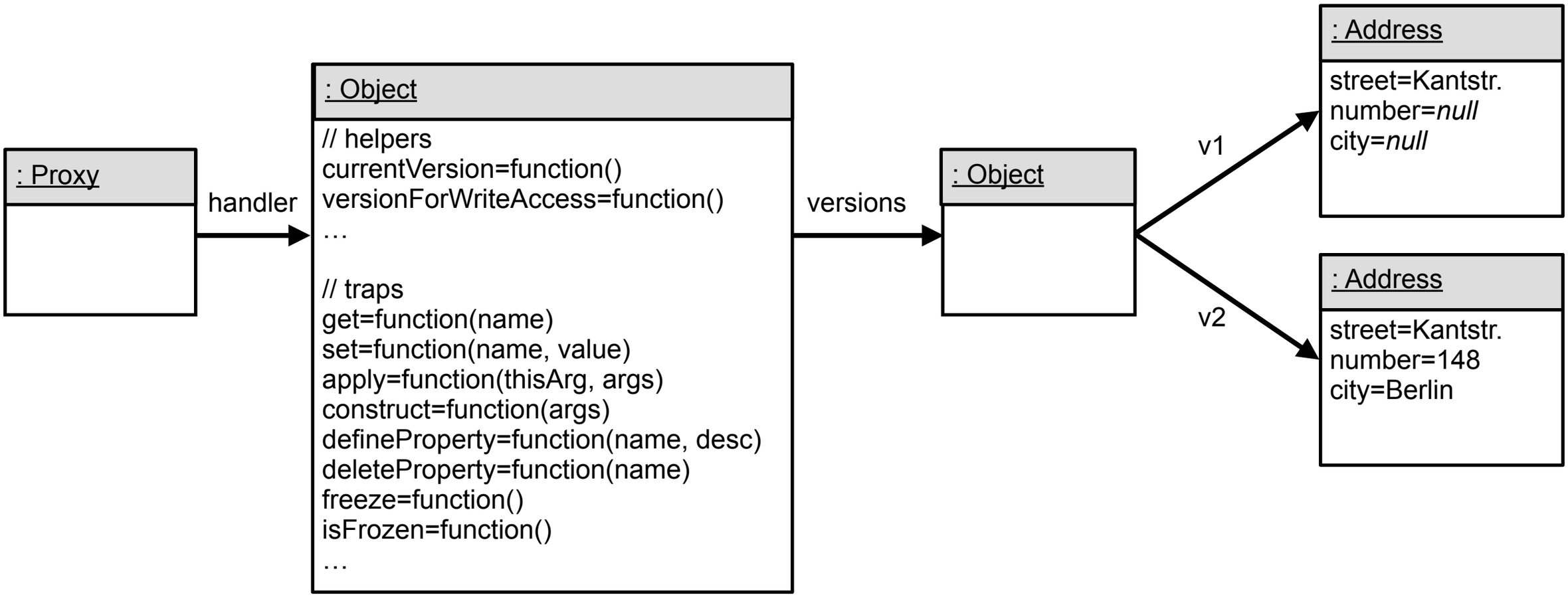
ECMAScript 6 Proxies

```
var client = {},
    server = {openSecret: "I don't like Mondays"},
    handler = {
        get: function(target, name) {
            console.log(name + ' was read at ' + Date());

            return target[name];
        }
    }

client.server = ne
```

```
> client.server.openSecret
openSecret was read at Sun Jun 01 2014 17:58:38 GMT+0200 (CEST)
< "I don't like Mondays"
```



ECMAScript 6 Proxies as Version-aware References



Handler: Read-only Traps

```
get: function(dummyTarget, name) {
    var version = this.currentVersion();

    return version[name];

}
```

```
currentVersion: function() {

    var objectVersion,
        systemVersion = lively.CurrentVersion;

    while(!objectVersion && systemVersion) {
        objectVersion = this.versions[systemVersion.ID];
        systemVersion =  systemVersion.predecessor;
    }

    return objectVersion;
}
```

Handler: Traps That Intercept Changes

```
set: function(target, name, value, receiver)
defineProperty: function(
deleteProperty: function(
freeze: function(target)
seal: function(target)
preventExtensions: functi
apply: function(target, thisArg, args)
```

```
set: function(dummyTarget, name, value) {
    var version = this.versionForWriteAccess();

    version[name] = value;

    return true;
}
```

```
versionForWriteAccess: function() {
    var newVersion;

    if (!this.versions[lively.CurrentVersion.ID]) {
        newVersion = this.copyObject(this.currentVersion());
        this.versions[lively.CurrentVersion.ID] = newVersion;
    }

    return this.currentVersion();
}
```

Use Version-aware References

return proxies for all new objects

Literals

```
{}, [], function () {}
```

Specific built-in functions

```
Array(), eval(),
Object.create(proto)
```

Constructors

```
new Morph()
```

Use Version-aware References: Transform Literal Objects

`{..}`	->	`proxyFor({..})`
`[..]`	->	`proxyFor([..])`
`function (..) {..}`	->	`proxyFor(function(..) {..})`
	...	

Use Version-aware References: Transform Specific Built-in Functions

`new Array()` —> `new proxyFor(Array)()`

…Array, Boolean, Date, Function, Number, Object, RegExp, String, Math, JSON, document, Worker, XMLHttpRequest, window…

`eval(..)` —> `eval(transformSource(..))`

Use Version-aware References: Return Value of Constructors

```
construct: function(dummyTarget, args) {

    // construct new instance

    return proxyFor(newInstance);
}
```

```
Morph = proxyFor(function(..) {..})
```

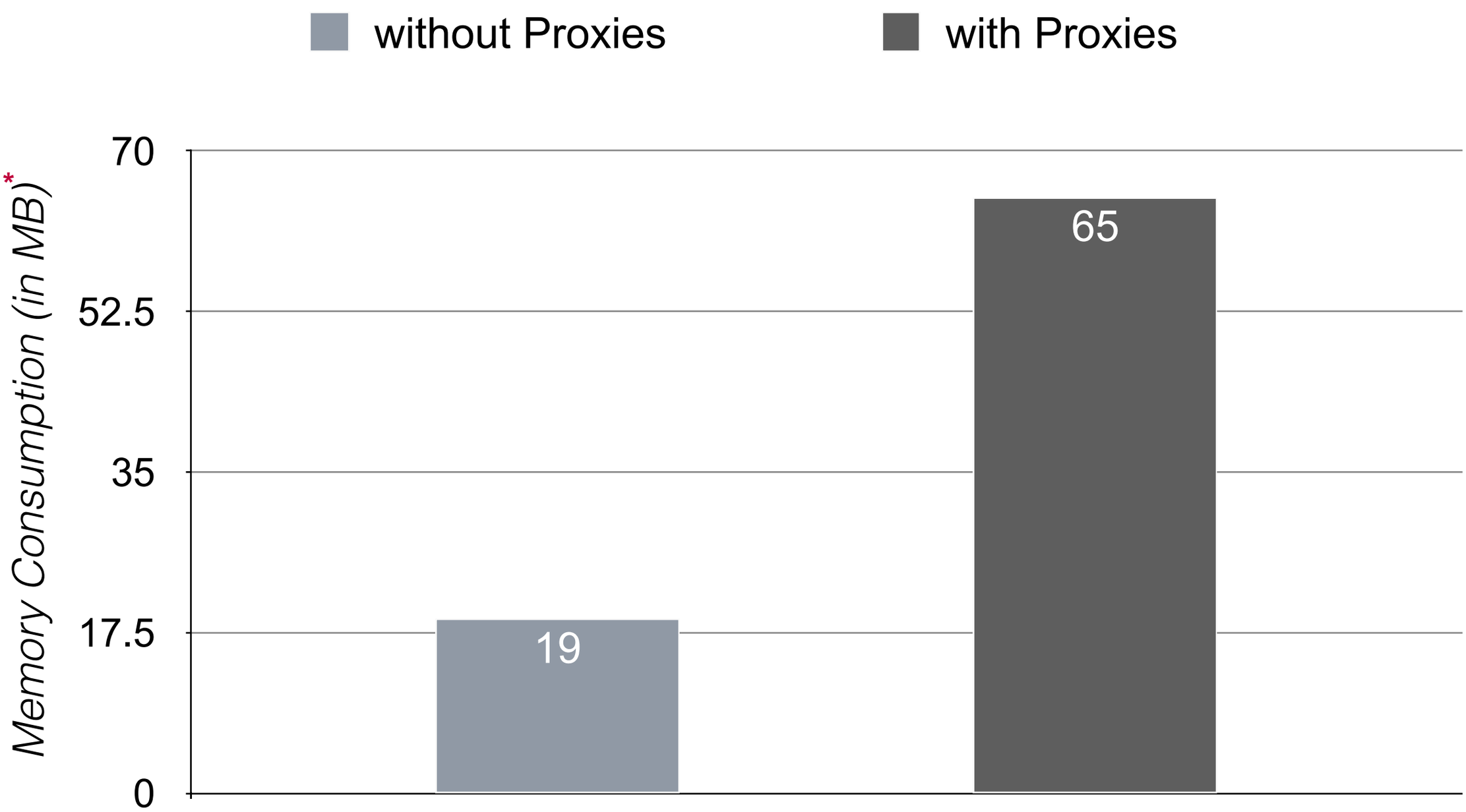
```
new Morph()
```

Evaluation

Functionality

DEMO /
SCREENCAST

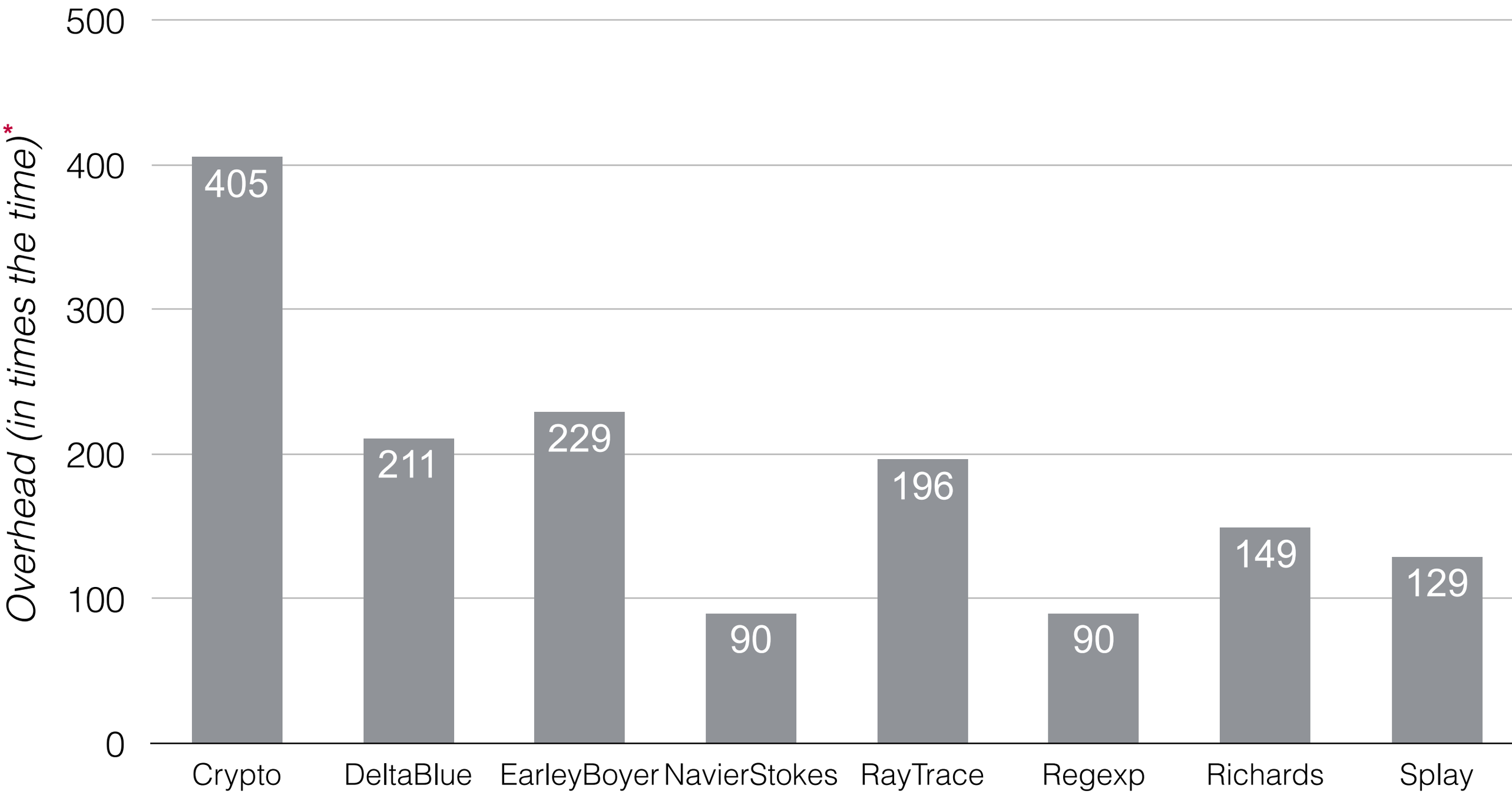
Memory Overhead for Proxies



Size of heap snapshots after loading the Blank world

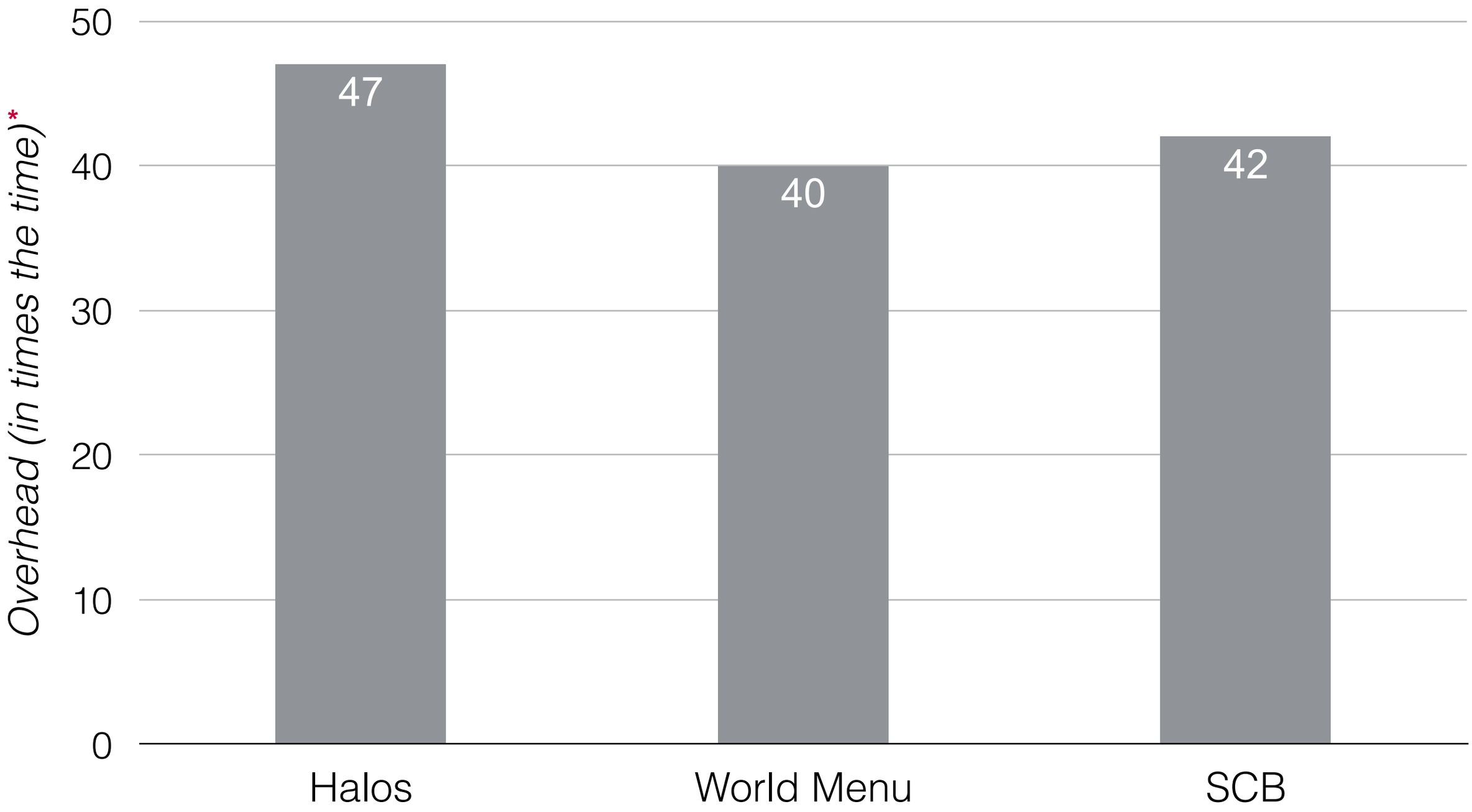
* averaged over five runs, on 5/9/2014, Chrome 34.0.1847.131, Macbook Air, 2 GHz i7, 8 GB RAM.

Execution Overhead: Octane Benchmarks



* averaged over five runs, on 5/9/2014, Chrome 34.0.1847.131, Macbook Air, 2 GHz i7, 8 GB RAM.

Execution Overhead: Lively Interactions



* averaged over five runs, on 5/9/2014, Chrome 34.0.1847.131, Macbook Air, 2 GHz i7, 8 GB RAM.

Future Work

Improving the Performance: Wait for ECMAScript 6

our implementation
(versioning proxy handler)

harmony-reflect library
(direct proxies)

Experimental Harmony Features
(catch-all proxies)

- release target for ECMAScript 6: 12/2014

Improving the Performance: Alternative Implementation

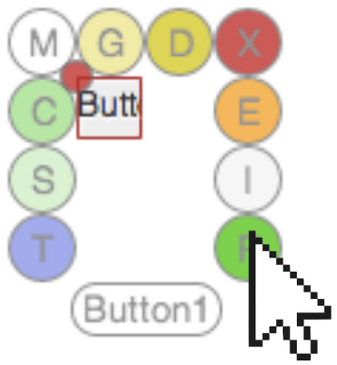
use source transformations and ordinary functions

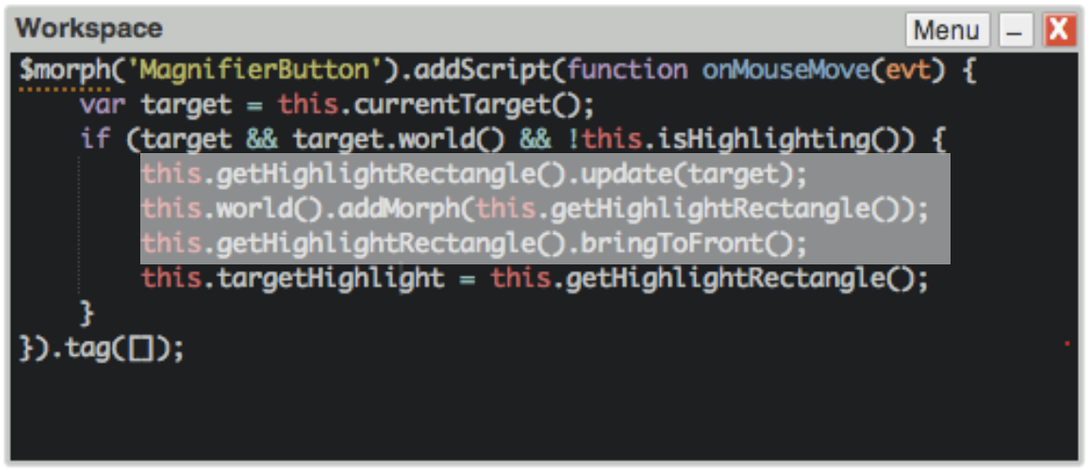
```
person.name          ->   get(person, 'name')
person.dance()       ->   apply(person, get(person, 'dance'))
                     …
```

```
function get(standIn , propertyName) {
   var version = lively.getCurrentVersionOf(standIn);

   return version[propertyName];

}
```

Tool Support: Continuous Versioning

continuous creation of versions corresponding to developer actions: direct manipulation, scripting, do-its, user-triggered events

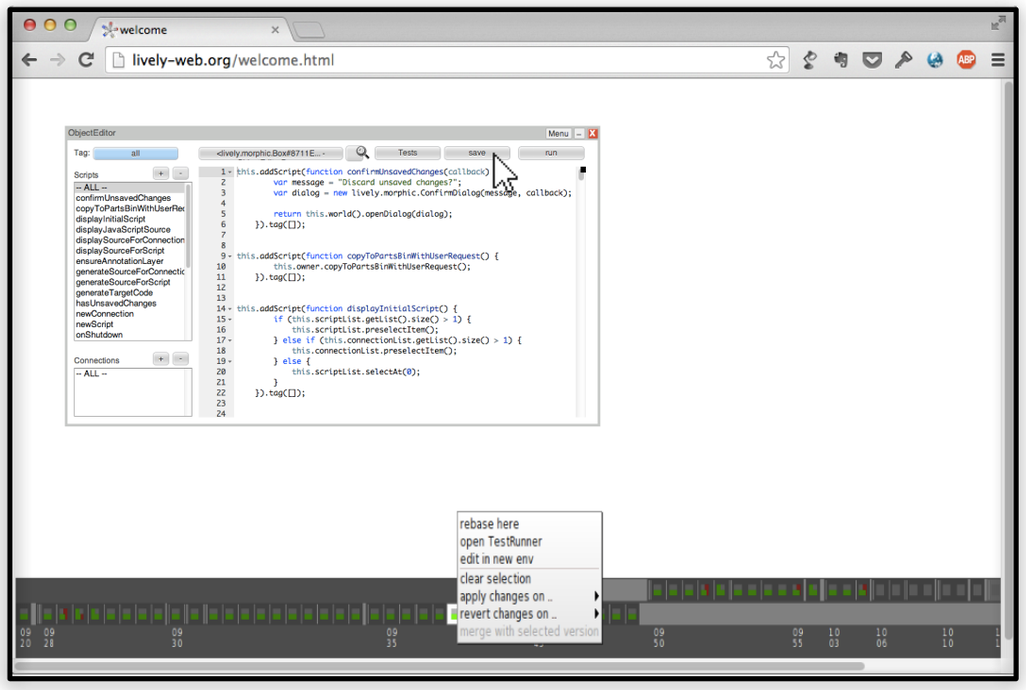


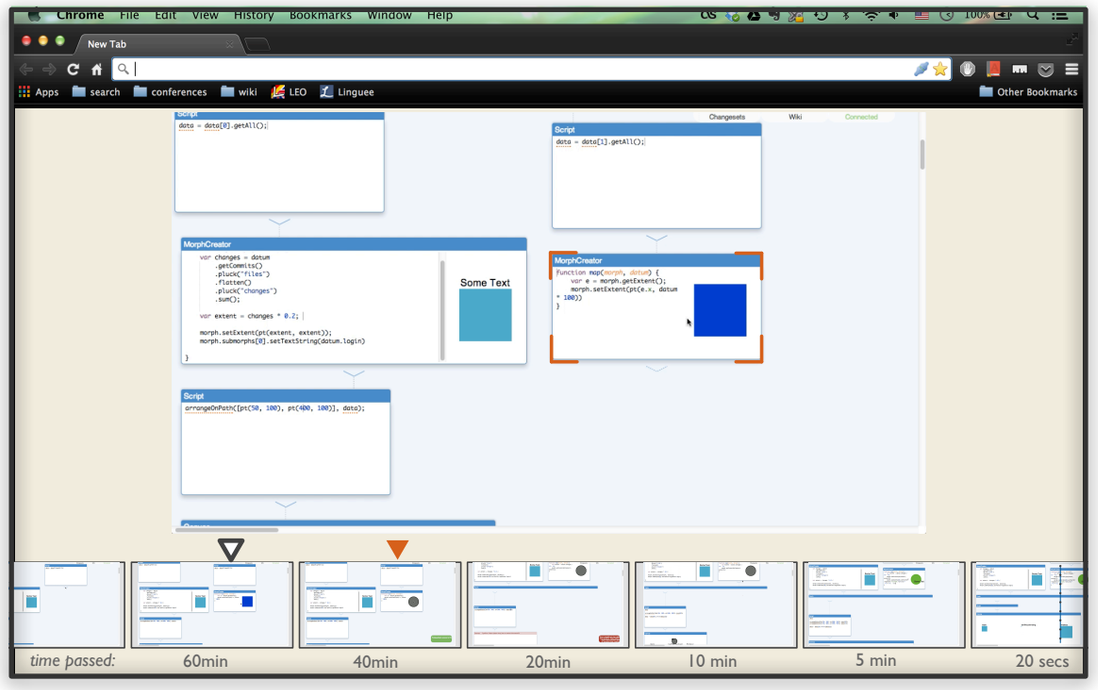


Tool Support:
Version Management Tools

- helpful information to find relevant versions of objects

- fine-grained version control to recover previous states





Related Work

Recovering Previous States

CoExist
[Steinert, et al.]

Lively Offline Worlds
[Czuchra]

Back-in-Time Debugging
[Lewis, et al.]

Software Transactional Memory
[Shavit, et al.]

Scoping Changes Dynamically

Worlds
[Warth, et al.]

Changeboxes
[Denker, et al.]

Object Graph Versioning
[Pluquet, et al.]

Practical Object-Oriented Back-in-Time Debugging
[Lienhard, et al.]

Context-oriented Programming
[Hirschfeld, et al.], [Lincke, et al.]

Conclusions

- the version-aware references approach seems to be sufficient for fine-grained versioning of objects
- implementation with ECMAScript 6 proxies is not (yet) practical
- developing the implementation for Lively helped a lot
- interesting open questions and future work

References

[Steinert, et al.] Bastian Steinert, Damien Cassou, and Robert Hirschfeld. “CoExist: Overcoming Aversion to Change”. In: Proceedings of the 8th Symposium on Dynamic Languages. DLS ’12. ACM, Jan. 2012, pp. 107–118.

[Warth, et al.] Alessandro Warth, Yoshiki Ohshima, Ted Kaehler, and Alan Kay. “Worlds: Controlling the Scope of Side Effects”. In: Proceedings of the 25th European Conference on Object-oriented Programming. ECOOP’11. Springer, July 2011, pp. 179–203.

[Lienhard, et al.] Adrian Lienhard, Tudor Gîrba, and Oscar Nierstrasz. “Practical Object-Oriented Back-in-Time Debugging”. In: Proceedings of the 22Nd European Conference on Object-Oriented Programming. ECOOP ’08. Springer, July 2008, pp. 592–615.

[Hirschfeld, et al.] Hirschfeld, Robert and Costanza, Pascal and Nierstrasz, Oscar. “Context-oriented Programming”. In: Journal of Object Technology 7.3 (Mar. 2008), pp. 125–151.

[Lincke, et al.] Jens Lincke and Robert Hirschfeld. “Scoping Changes in Self-supporting Development Environments Using Context-oriented Programming”. In: Proceedings of the International Workshop on Context-Oriented Programming. COP ’12. ACM, June 2012, 2:1–2:6.

[Shavit, et al.] Nir Shavit and Dan Touitou. “Software Transactional Memory”. In: Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing. PODC ’95. ACM, June 1995, pp. 204–213.

[Pluquet, et al.] Frédéric Pluquet, Stefan Langerman, and Roel Wuyts. “Executing Code in the Past: Efficient In-memory Object Graph Versioning”. In: Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications. OOPSLA ’09. ACM, Oct. 2009, pp. 391–408.

[Denker, et al.] Marcus Denker, Tudor Gîrba, Adrian Lienhard, Oscar Nierstrasz, Lukas Renggli, and Pascal Zumkehr. “Encapsulating and Exploiting Change with Changeboxes”. In: Proceedings of the 2007 International Conference on Dynamic Languages. ICDL’07. ACM, Aug. 2007, pp. 25–49.

[Czuchra, et al.] Martin A. Czuchra. “Offline Worlds: Automated Client–Side Persistence in Lively Kernel”. Master Thesis. Hasso-Plattner-Institute, University of Potsdam, Jan. 2012.

[Lewis, et al.] Bill Lewis. “Debugging Backwards in Time”. In: Proceedings of the Fifth International Workshop on Automated Debugging. AADEBUG’03. Springer, Sept. 2003, pp. 225–235.

[Gamma, et al.] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Jan. 1995.