The case for the Three R's of Systems Research:

Repeatability Reproducibility & Rigor

Jan Vitek

Kalibera, Vitek. Repeatability, Reproducibility, and Rigor in Systems Research. *EMSOFT11*

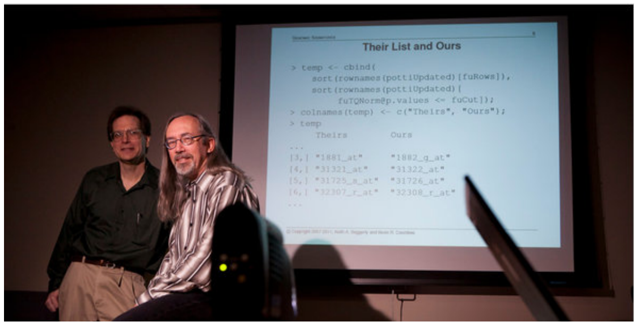
Science Done Bad

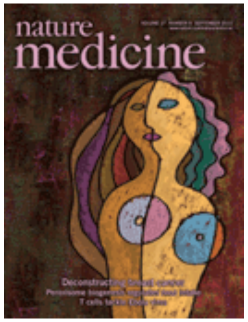
In 2006, Potti&Nevins claim they can predict lung cancer

In 2010, papers retracted, bancruptcy, resignations & investigation

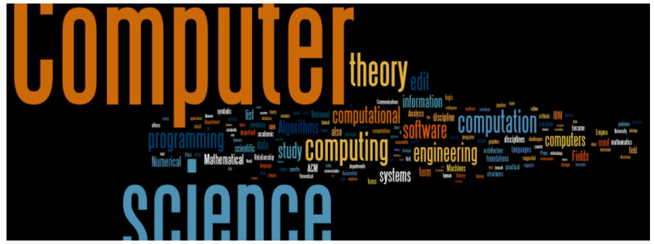
Bad science ranging from fraud, unsound methods, to off-by-one errors in Excel

Uncovered by a repetition study conducted by Baggerly&Coombes with access to raw data and 2,000 hours of effort





NEJM

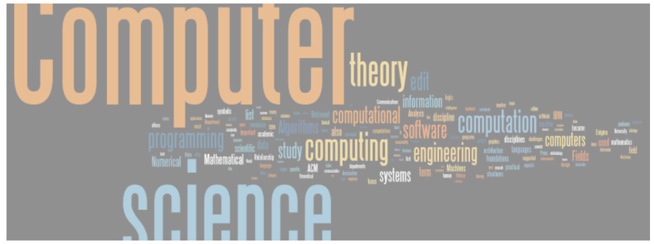


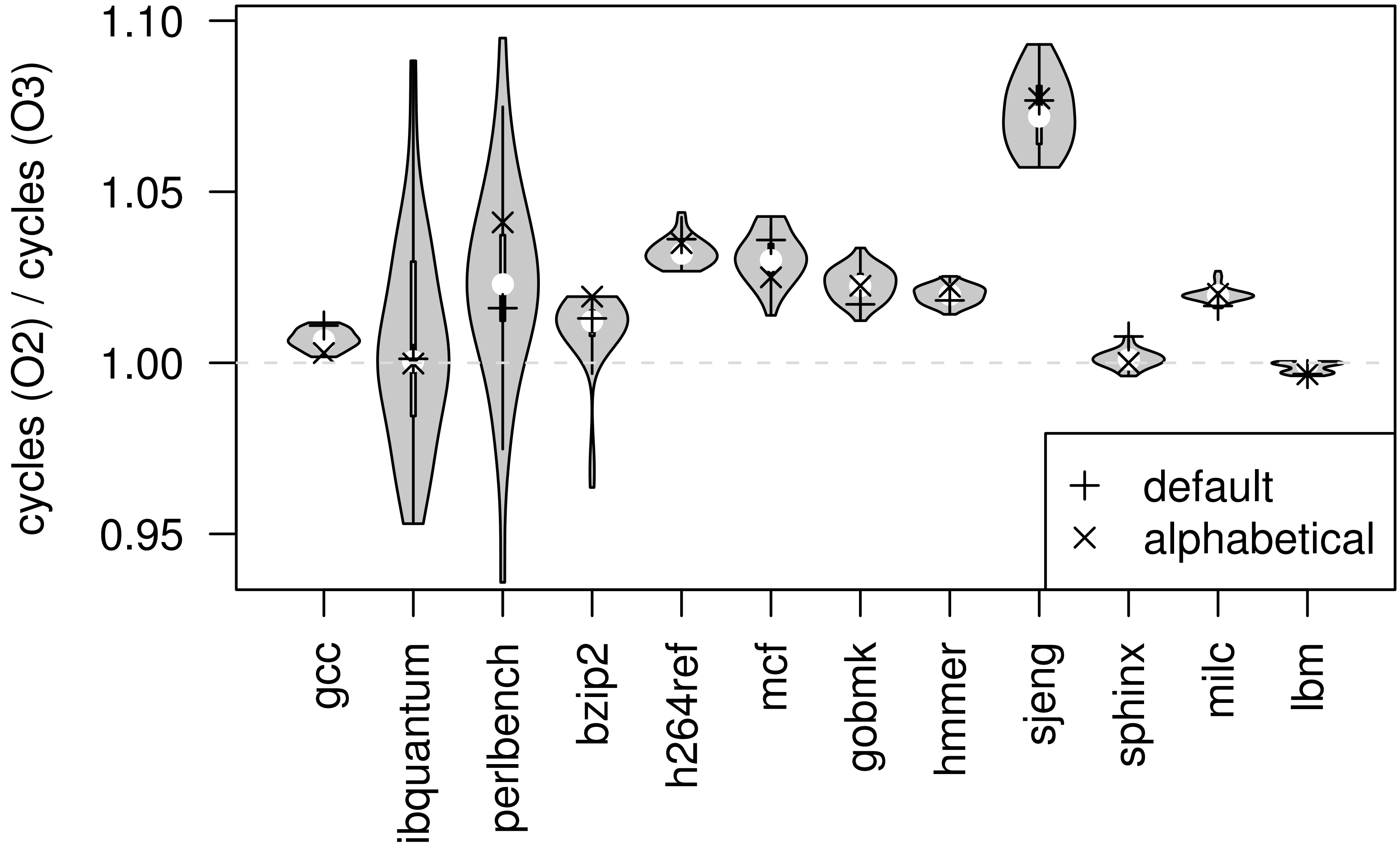
Out of 122 papers in ASPLOS, ISMM, PLDI, TACO, TOPLAS

90 evaluated execution time based on experiments

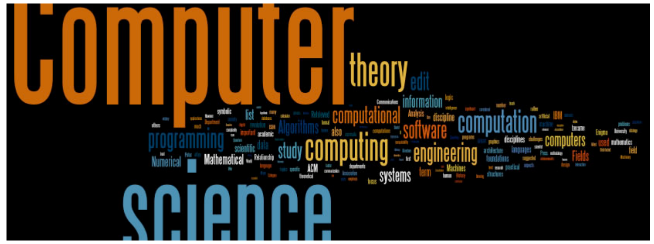
71 of these 90 papers ignored uncertainty

Aparté





Mytkowicz, Diwan, Hauswirth, Sweeney. Producing Wrong Data Without Doing Anything Obviously Wrong! *ASPLOS'09*



Out of 122 papers in ASPLOS, ISMM, PLDI, TACO, TOPLAS

90 evaluated execution time based on experiments

71 of these 90 papers ignored uncertainty

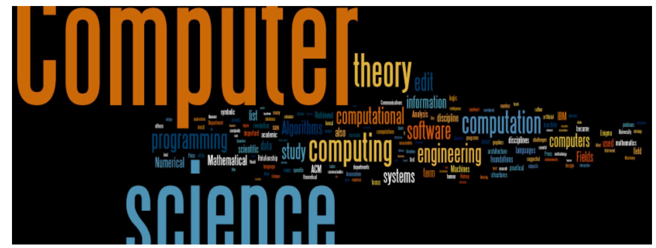
This lack of rigor undermines the results

Yet, no equivalent to the Duke Scandal.

Are we better?
Is our research not worth reproducing?
Is our research too hard to reproduce?

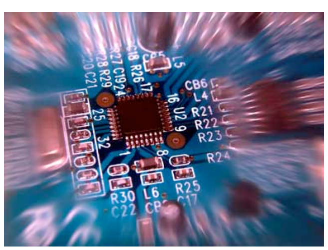
Reproduction

…independent researcher implements/realizes the published solution from scratch, under new conditions



Repetition

… re-doing the same experiments on the same system and using the same evaluation method



Is our research hard to repeat ?
Is our research hard to reproduce ?

TRUE STORIES

Goal

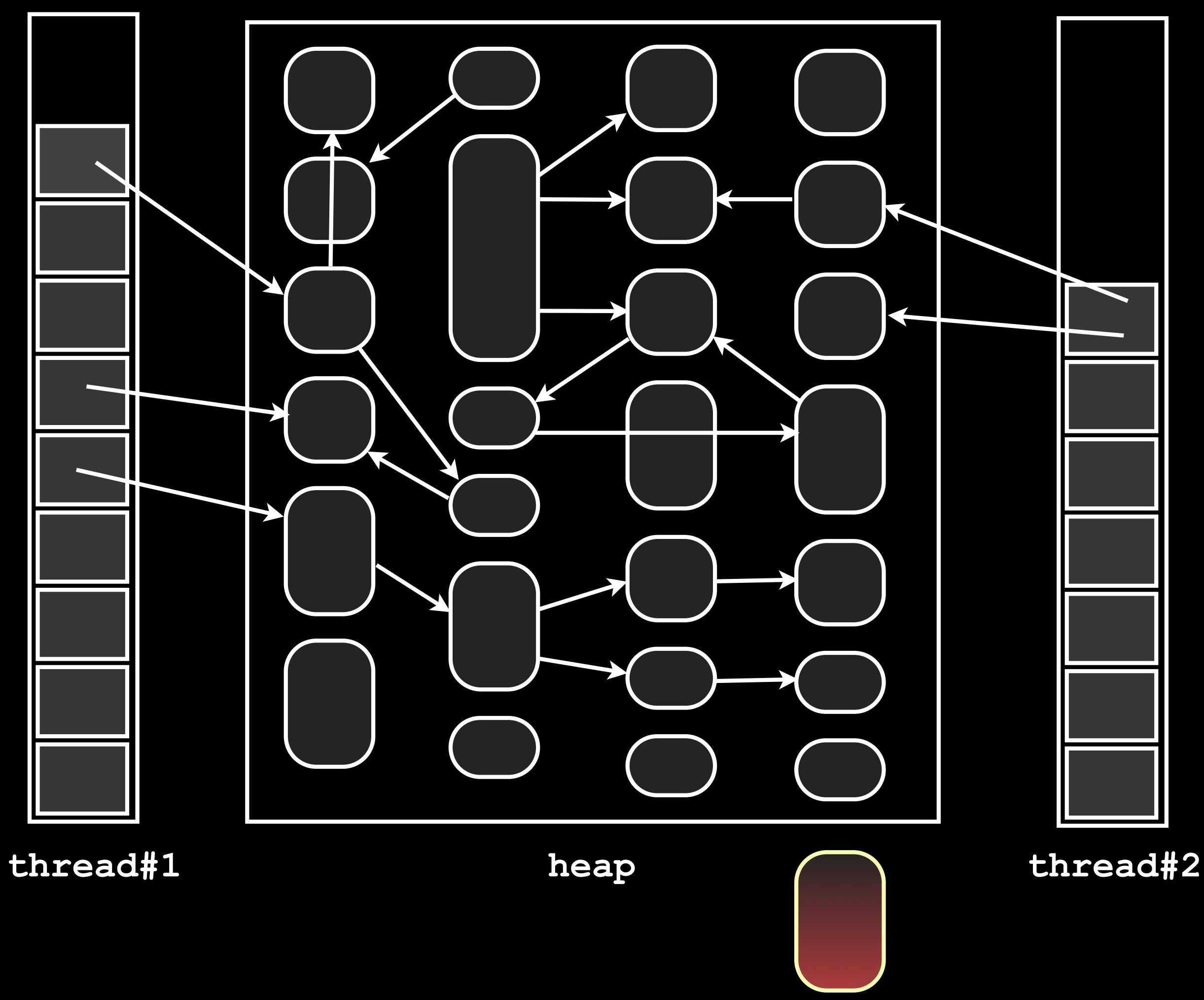
Break new ground in

hard real-time concurrent garbage collection

Aparté

GC in 3 minutes

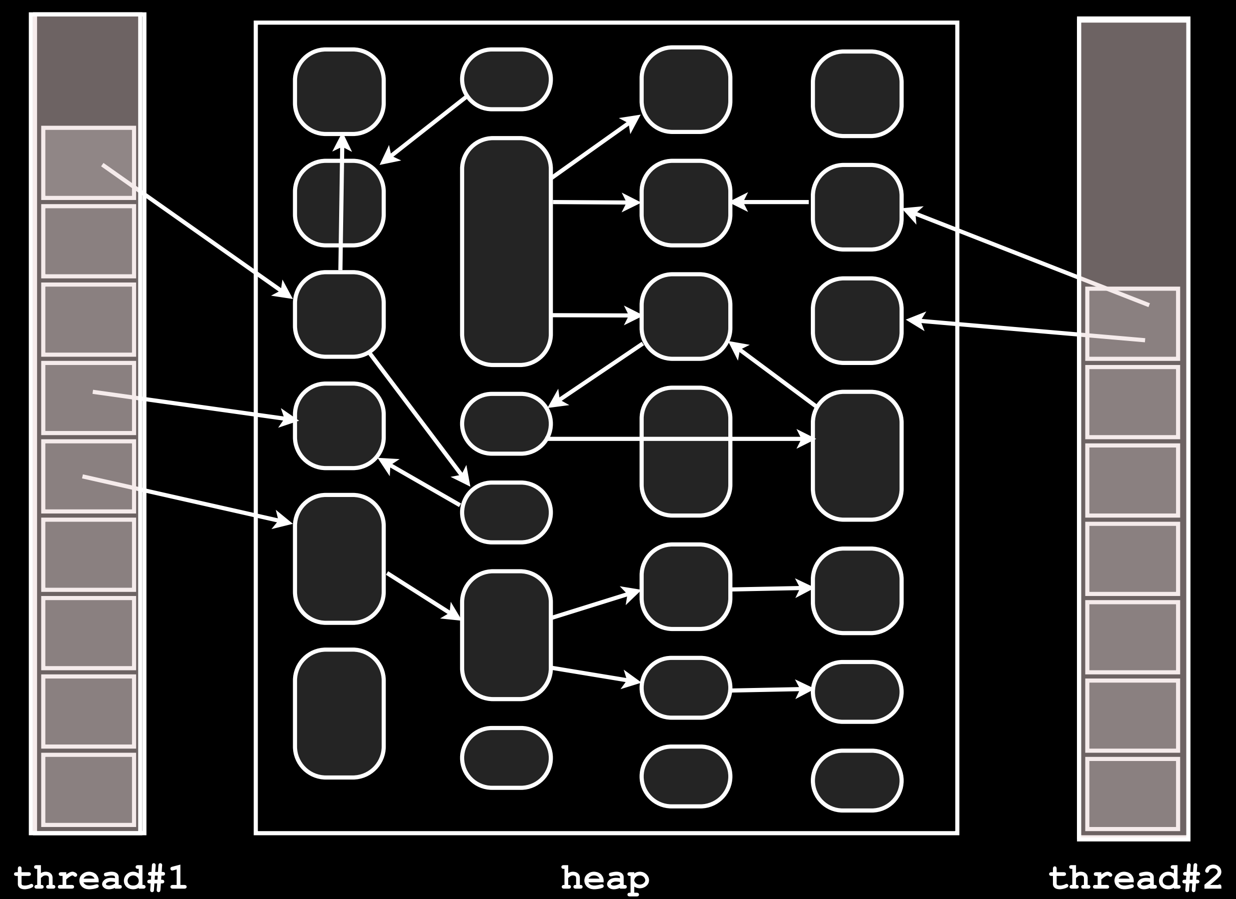
Garbage Collection



Phases

- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction

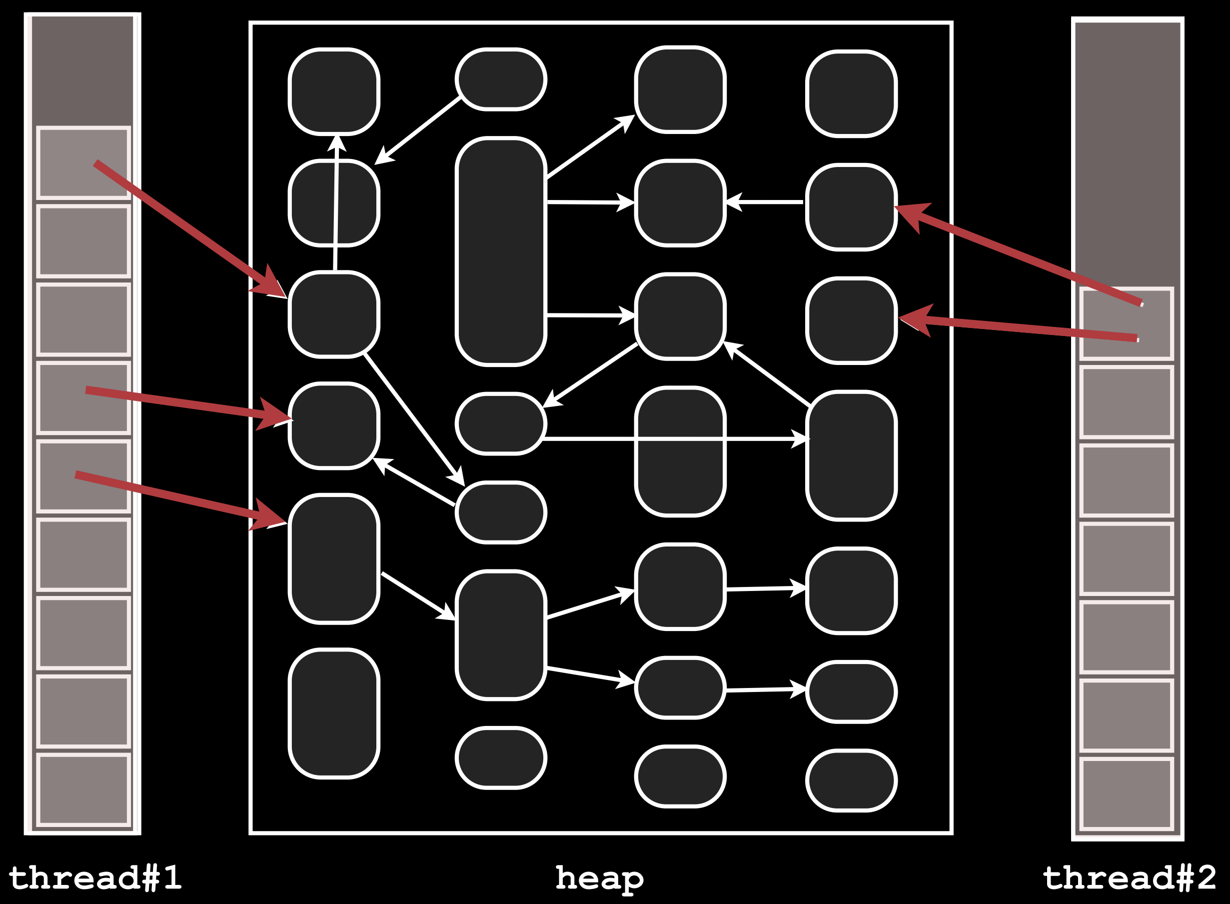
Garbage Collection



Phases

- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction

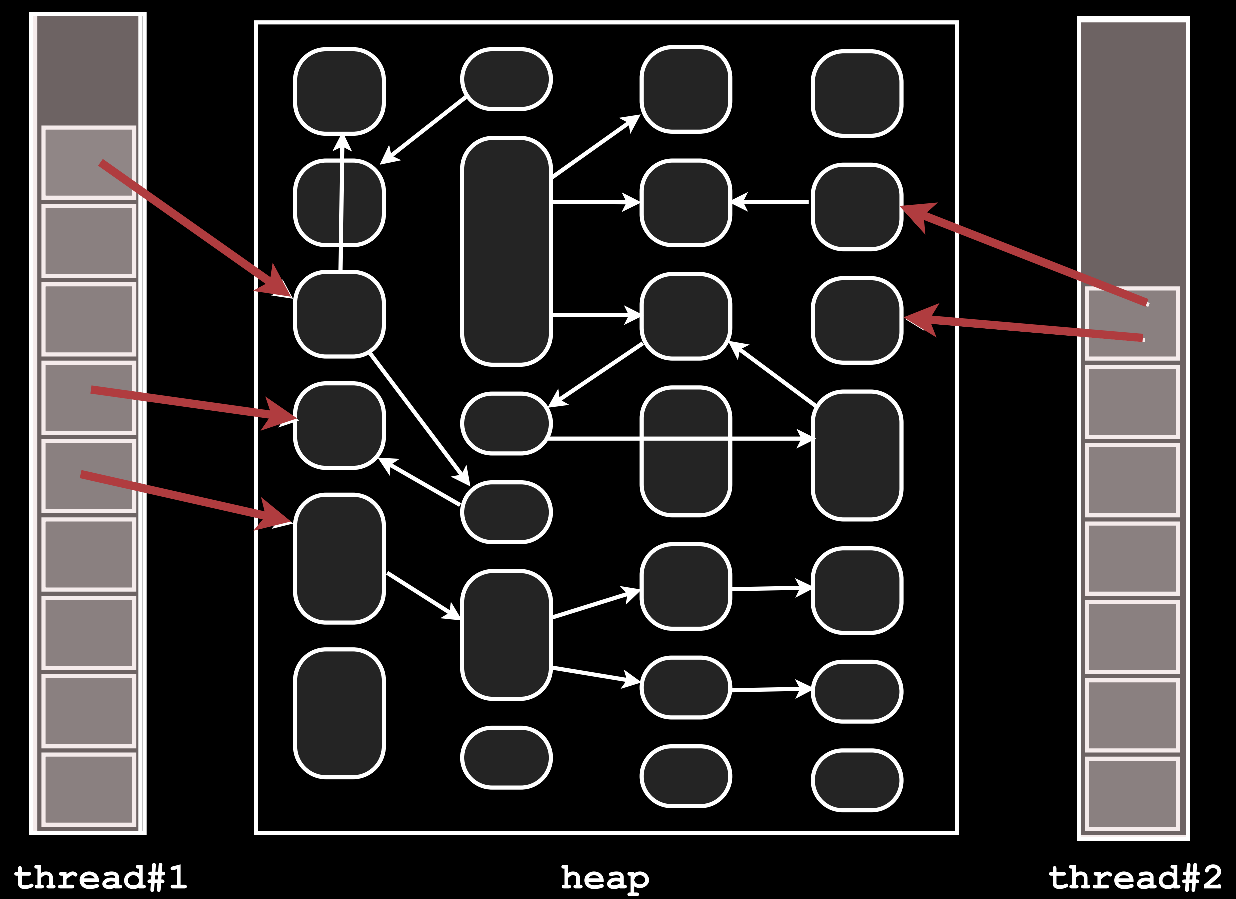
Garbage Collection



Phases

- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction

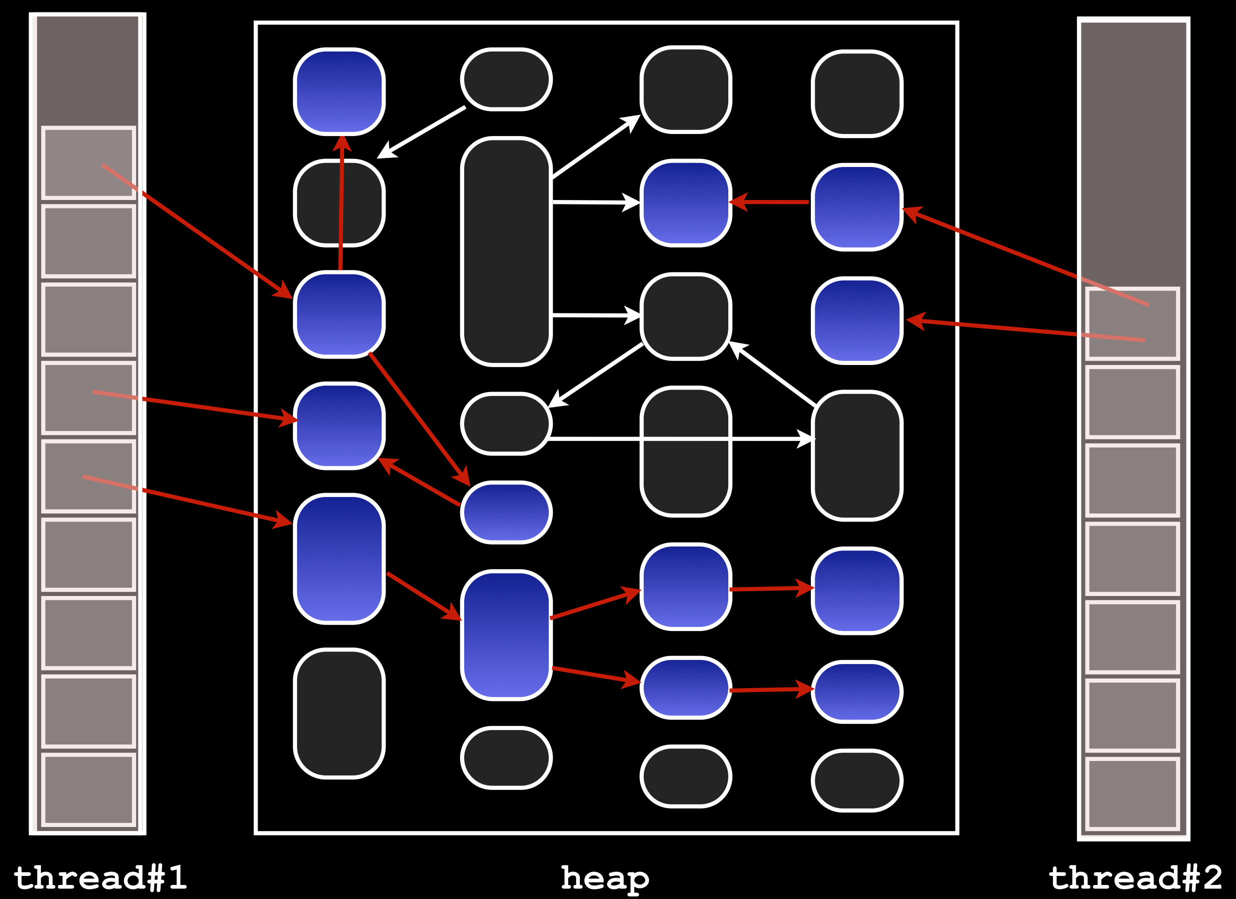
Garbage Collection



Phases

- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction

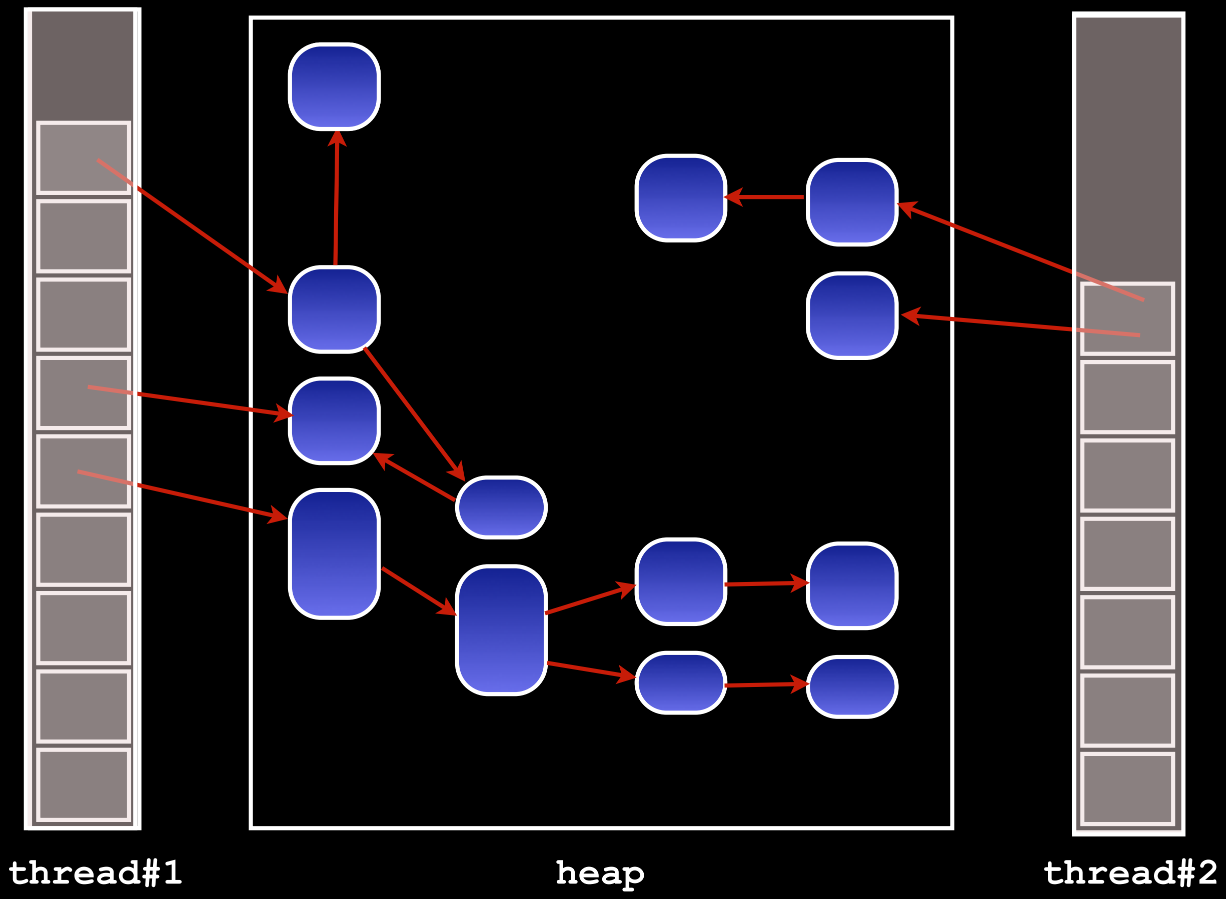
Garbage Collection



Phases

- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction

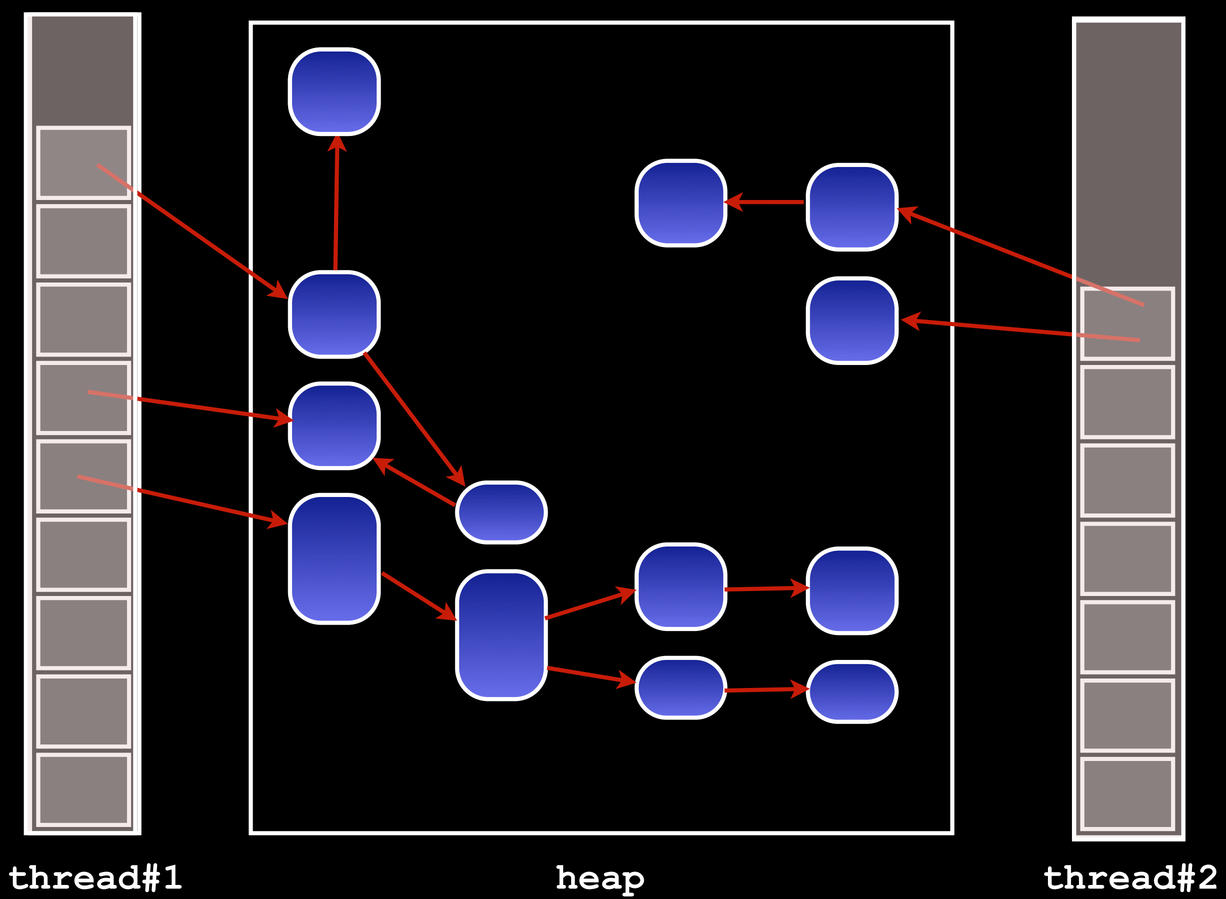
Garbage Collection



Phases

- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction

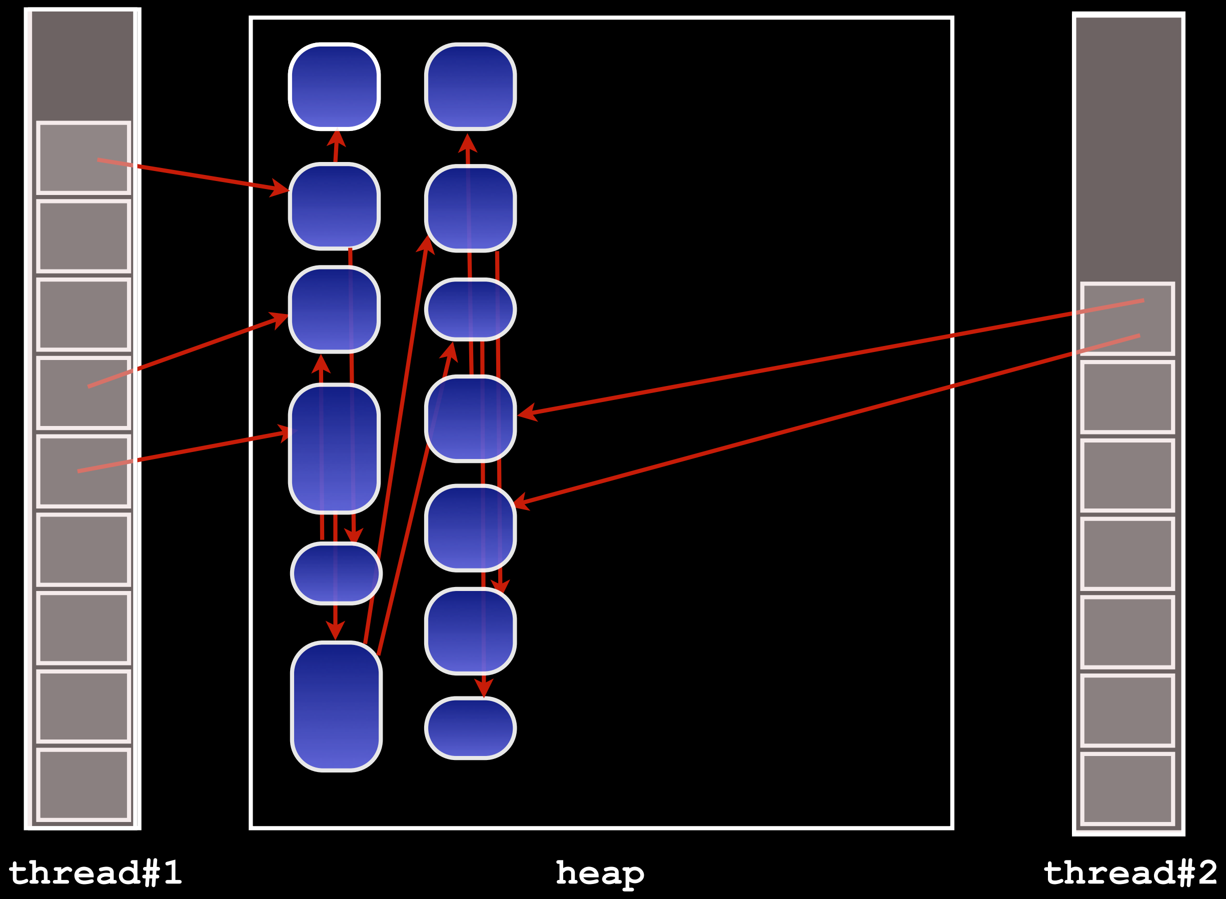
Garbage Collection



Phases

- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction

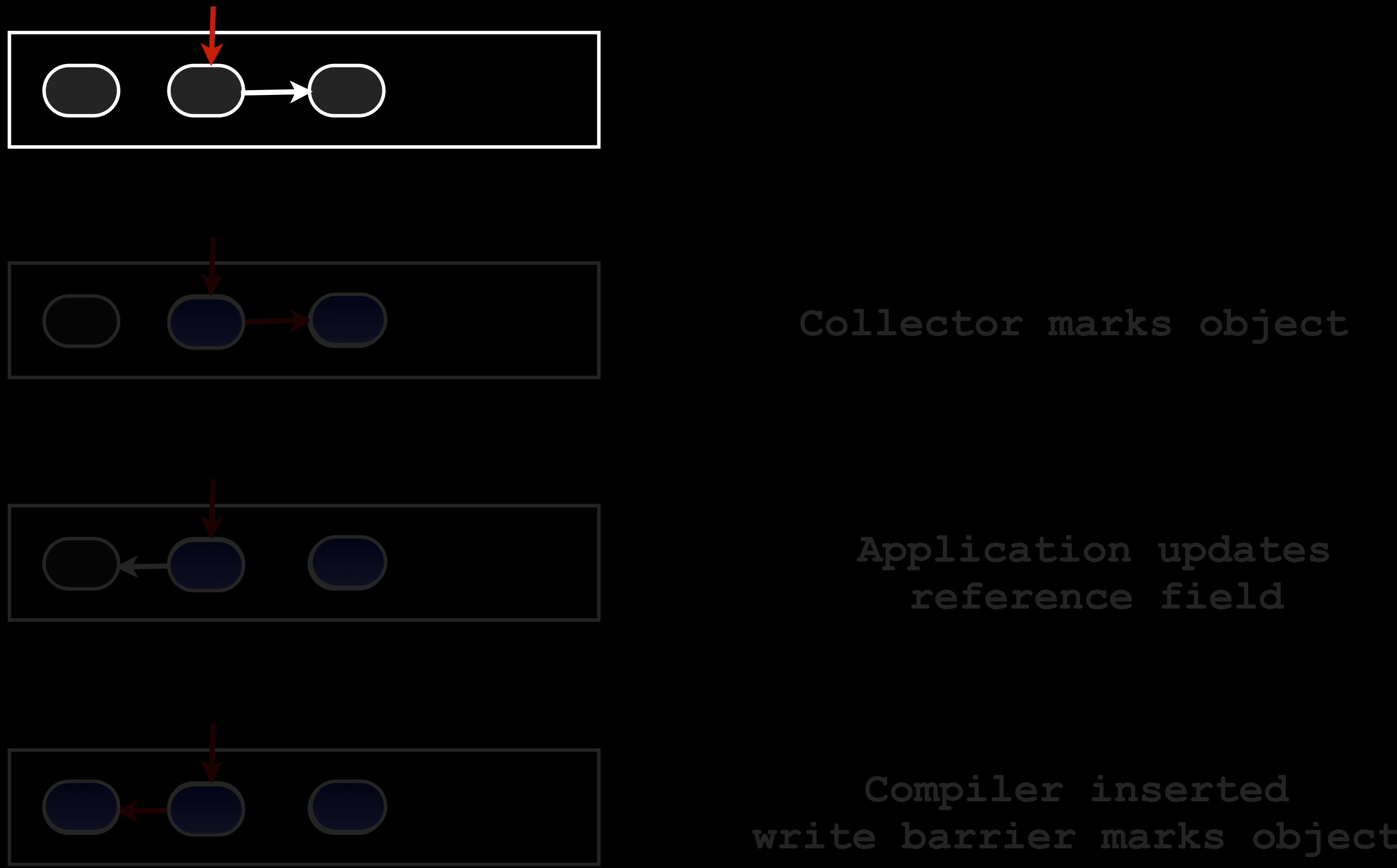
Garbage Collection



Phases

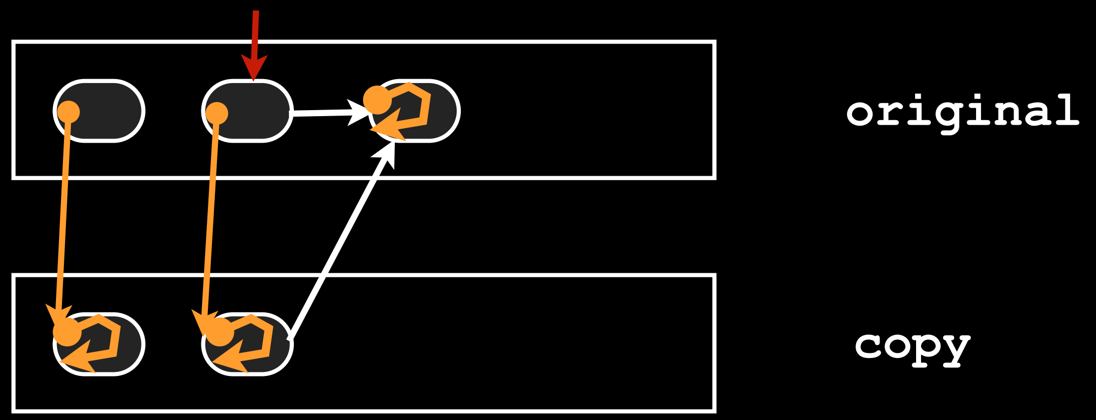
- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction

Incrementalizing marking



Incrementalizing compaction

- Forwarding pointers refer to the current version of objects
- Every access must start with a dereference



TRUE STORIES

Obstacles

No real-time benchmarks for GCed languages

No clear competition, two GC algorithms claim to be best

No accepted measurement methodology

No open source experimental platform for comparison

Step 1

Develop an open source experimental platform

Picked the Real-time Specification for Java

First generation system, about 15 man/years

Flew on a Boeing ScanEagle

Second generation system, about 6 man/years

Competitive with commercial JVMs

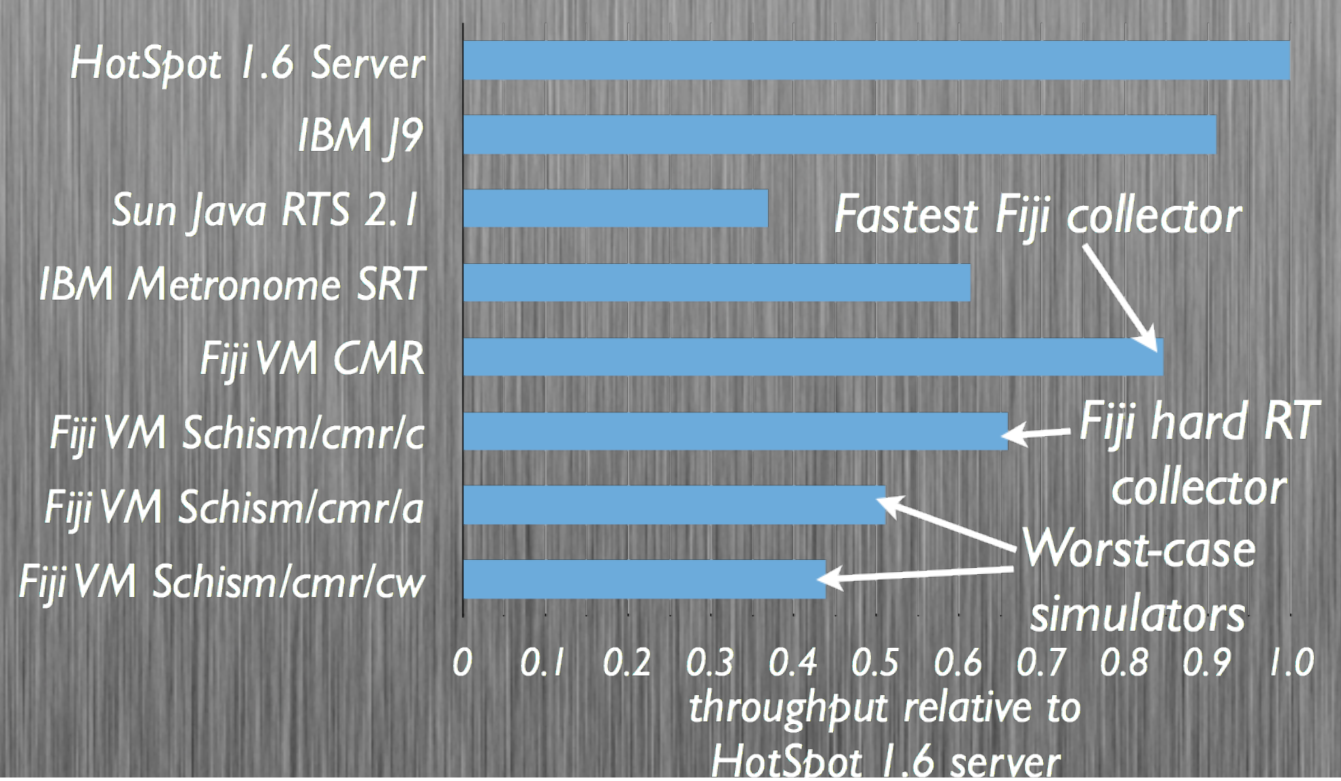
A Real-time Java Virtual Machine for Avionics. TECS, 2006

Observations

Results on noncompetitive systems not relevant

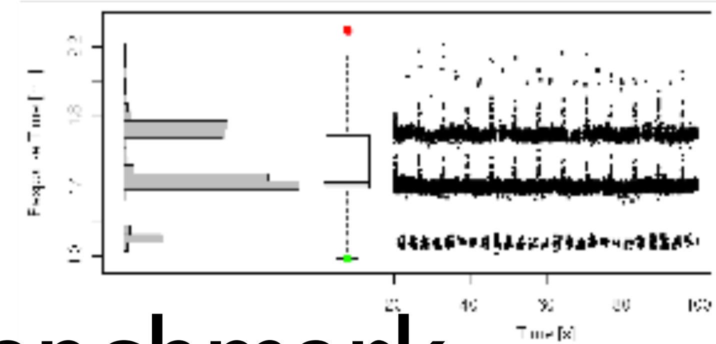
Much of work went into a credible research platform

SpecJVM98



TRUE STORIES

Step 2



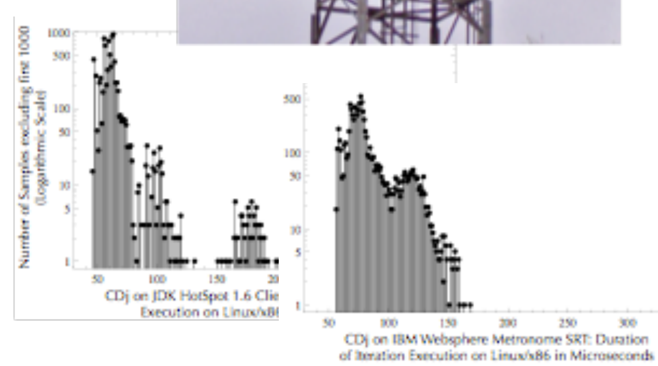
Develop an open source benchmark

- *Collision Detector Benchmark*
- *In Java, Real-time Java, and C (Linux/RTEMS)*

Measure response time, release time jitter

- *Simulate air traffic control*
- *Hard RT collision detector thread*
- *Scalable stress on garbage collector*

About 1.5 man/years



A family of Real-time Java benchmarks. CC:PE 2011

Observation

Understanding what you measure is critical

Running on a real embedded platform and real-time OS, difference between Java & C small…

Good news?

No. The LEON3 lacks a FP unit, & the benchmark is FP intensive...

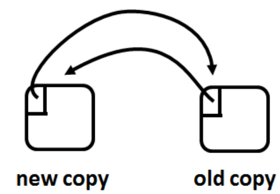
TRUE STORIES

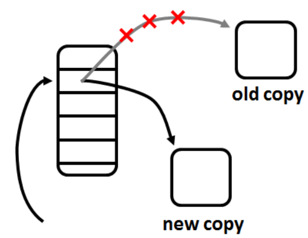
Step 3

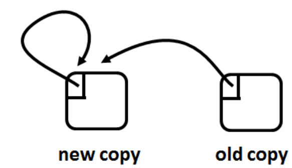
Gain experience with the state of the art

Experiment with different GC techniques

GC in uncooperative environment
Brooks forwarding
Object replication
Object handles







About 2 man/years

Accurate Garbage Collection in Uncooperative Environments. CC:P&E, 2009
Hierarchical Real-time Garbage Collection. LCTES, 2007
Replicating Real-time Garbage Collector. CC:P&E, 2011
Handles Revisited: Optimising Performance and Memory... ISMM, 2011

Observation

Trust but verify, twice.

From workshop to journal, speed 30% better

Good news?

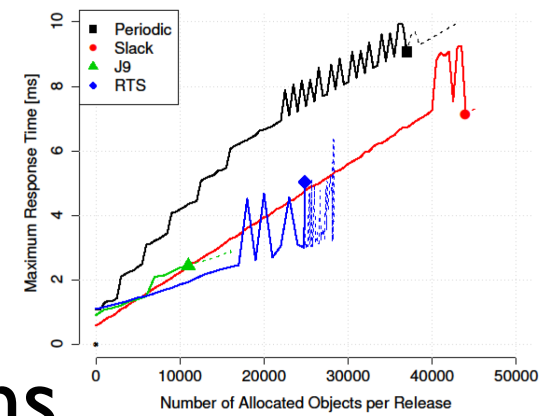
Later realized switching to GCC 4.4 slowed baseline (GCC didn't inline a critical function)

Once accounted for this our speed up was 4%…

A correction was issued...

TRUE STORIES

Step 4



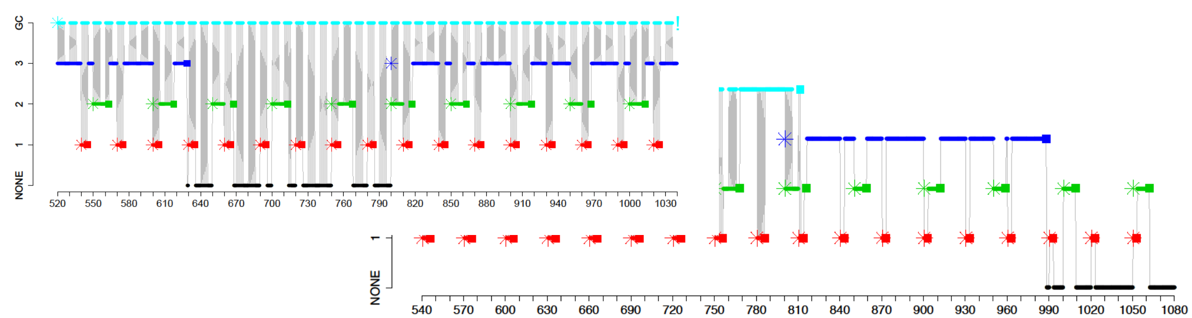
Reproduce state of the art algorithms from IBM and Oracle

Metronome, Sun Java RTS

Choose measurement methodology

Existing metric (MMU) inadequate

About 3 man/years



Scheduling Real-Time Garbage Collection on Uniprocessors. TOCS 2011
Scheduling Hard Real-time Garbage Collection. RTSS 2009

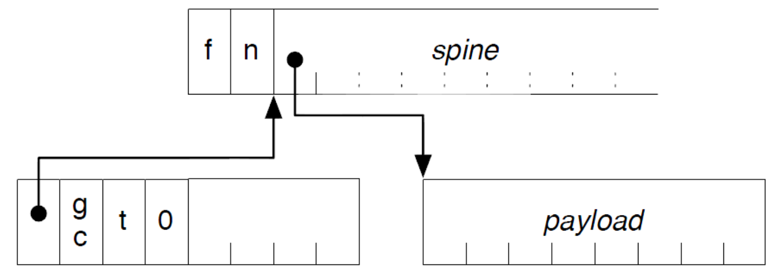
Observation

Reproduction was difficult because of closed-source implementations & partial description of algorithms

Repetition was impossible because no common platform

TRUE STORIES

Step 5

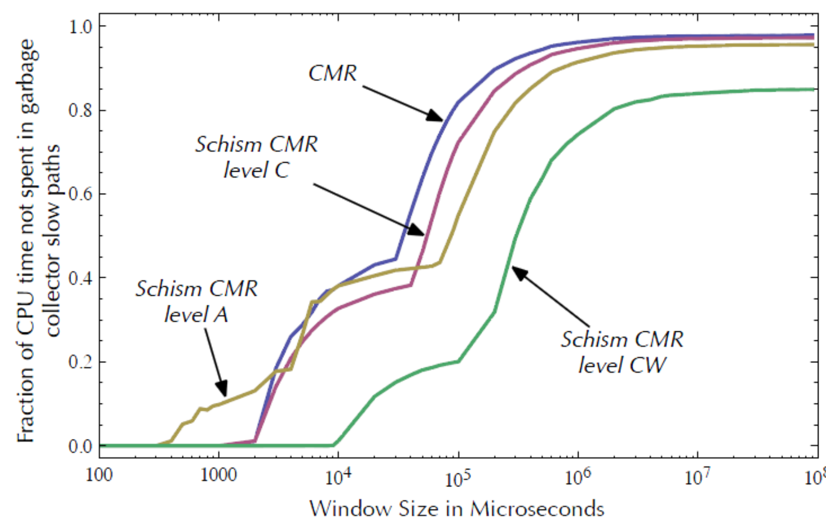


Develop a novel algorithm

Fragmentation tolerant

Constant-time heap access

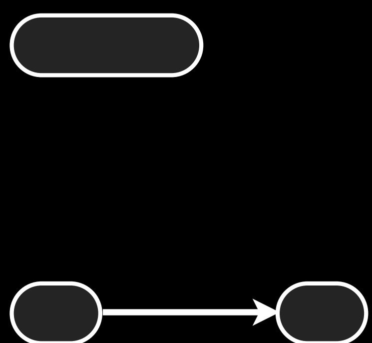
About 0.6 man/years



Schism: Fragmentation-Tolerant Real-Time Garbage Collection. *PLDI 2011*

Schism: objects

- Avoid external fragmentation by splitting objects in 32-byte chunks

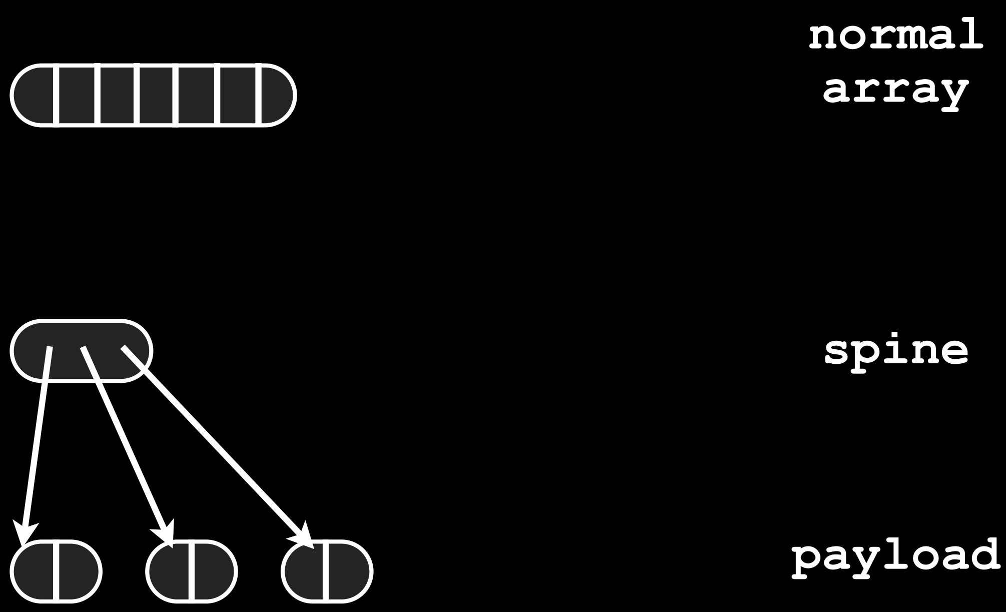


normal object

split object

Schism: arrays

- For faster array access, array = variable sized spine
  + 32-byte chunk payload

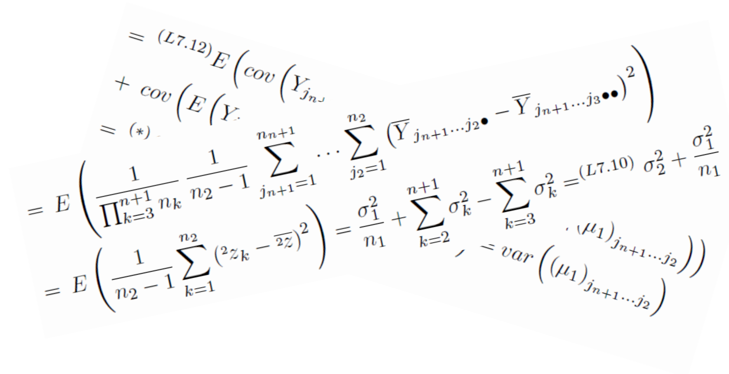


TRUE STORIES

In summary,
28 m/y reproduction
.6 m/y novel work

Experimental platform	21	man/years
Benchmark	2	man/years
Implementing basic techniques	2	man/years
Reproduction of state-of-the art +measurement methodology	3	man/years
Implementing novel algorithm	0.6	man/years

Rigor



Cater for random effects, non-determinism

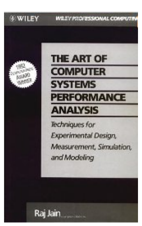
Repeat experiment runs, summarize results

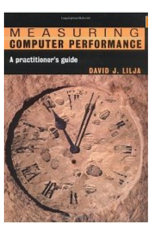
Threat to validity detectable by failure to repeat

Guard against bias

Use multiple configurations, hardware platforms

Threat to validity detectable by failure to reproduce

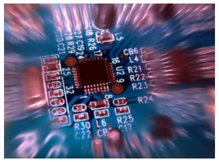




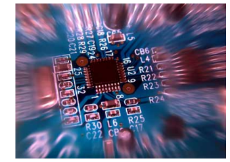
Jain: The Art of Computer Systems Performance Analysis

Lilja: Measuring Computer Performance, A Practitioner's Guide

Evaluate Collaboratory, http://evaluate.inf.usi.ch/



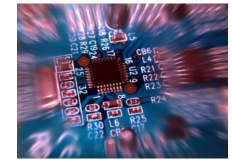
Repeatability



Enable repetition studies

- Archival
  - *Automate and archive*
- Disclosure
  - *Share experimental details*

Reproducibility



Community support for focused reproductions

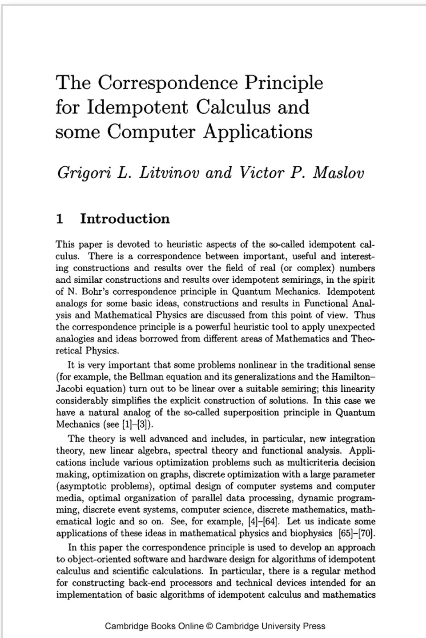
Open benchmarks and platforms

Reward system for reproductions

Publish reproduction studies

Regard them as 1st class publications

TRUE STORIES


The Correspondence Principle
for Idempotent Calculus and
some Computer Applications

Grigori L. Litvinov and Victor P. Maslov

1 Introduction

This paper is devoted to heuristic aspects of the so-called idempotent calculus. There is a correspondence between important, useful and interesting constructions and results over the field of real (or complex) numbers and similar constructions and results over idempotent semirings, in the spirit of N. Bohr's correspondence principle in Quantum Mechanics. Idempotent analogs for some basic ideas, constructions and results in Functional Analysis and Mathematical Physics are discussed from this point of view. Thus the correspondence principle is a powerful heuristic tool to apply unexpected analogies and ideas borrowed from different areas of Mathematics and Theoretical Physics.

It is very important that some problems nonlinear in the traditional sense (for example, the Bellman equation and its generalizations and the Hamilton–Jacobi equation) turn out to be linear over a suitable semiring; this linearity considerably simplifies the explicit construction of solutions. In this case we have a natural analog of the so-called superposition principle in Quantum Mechanics (see [1]–[3]).

The theory is well advanced and includes, in particular, new integration theory, new linear algebra, spectral theory and functional analysis. Applications include various optimization problems such as multicriteria decision making, optimization on graphs, discrete optimization with a large parameter (asymptotic problems), optimal design of computer systems and computer media, optimal organization of parallel data processing, dynamic programming, discrete event systems, computer science, discrete mathematics, mathematical logic and so on. See, for example, [4]–[64]. Let us indicate some applications of these ideas in mathematical physics and biophysics [65]–[70].

In this paper the correspondence principle is used to develop an approach to object-oriented software and hardware design for algorithms of idempotent calculus and scientific calculations. In particular, there is a regular method for constructing back-end processors and technical devices intended for an implementation of basic algorithms of idempotent calculus and mathematics

Paper

uses

backs claims

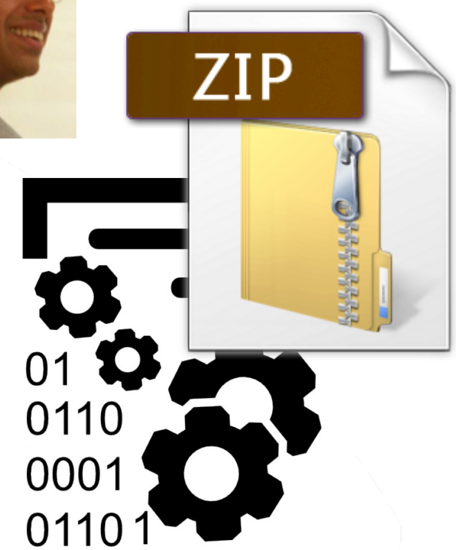
Artifact (code, data, etc.)

(c) Camil Demetrescu

TRUE STORIES

Key ideas

Program Committee



Artifact Evaluation Committee

(c) Camil Demetrescu

TRUE STORIES

Key ideas

Artifact Evaluation Committee
PhD students
postdocs

+

Senior co-chairs

TRUE STORIES

Authoritative site: http://www.artifact-eval.org/

(c) Camil Demetrescu

TRUE STORIES

Criteria

(c) Camil Demetrescu

Consistent with the Paper

Paper

Artifact

Complete

Easy to Reuse

vs.

Well Documented

(c) Camil Demetrescu

(c) Camil Demetrescu

TRUE STORIES

Statistics from OOPSLA'13

2 AEC co-chairs
24 AEC members
3 reviews per AEC member
3 reviews per artifact

18 accepted

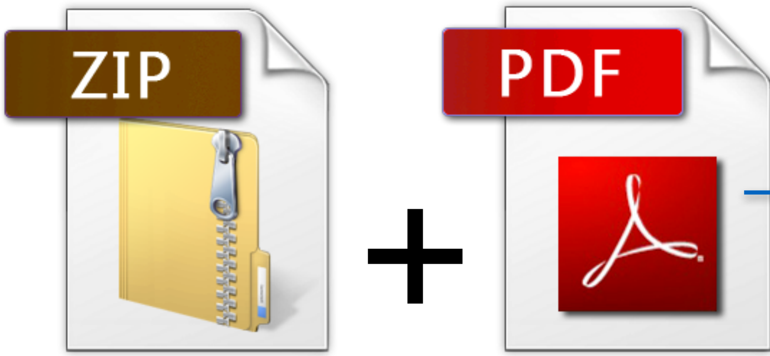
21 artifacts submitted

50 papers accepted

TRUE STORIES

ECOOP Prague 2015

Artifact publication



Software, data, etc.

Artifact key info

TreatJS: Higher-Order Contracts for JavaScript (Artifact)

Matthias Keil and Peter Thiemann

Institute for Computer Science
University of Freiburg
Freiburg, Germany
{keilr,thiemann}@informatik.uni-freiburg.de

Abstract

TreatJS is a language embedded, higher-order contract system for JavaScript which enforces contracts by run-time monitoring. Beyond providing the standard abstractions for building higher-order contracts (base, function, and object contracts), *TreatJS*'s novel contributions are its guarantee of non-interfering contract execution, its systematic approach to blame assignment, its support for contracts in the style of union and intersection types, and its notion of a parameterized contract scope, which is the building block for composable run-time generated contracts that generalize dependent function contracts.

TreatJS is implemented as a library so that all aspects of a contract can be specified using the full JavaScript language. The library relies on JavaScript proxies to guarantee full interposition for contracts. It further exploits JavaScript's reflective features to run contracts in a sandbox environment, which guarantees that the execution of contract code does not modify the application state. No source code transformation or change in the JavaScript run-time system is required.

1998 ACM Subject Classification D.2.4 Software/Program Verification
Keywords and phrases Higher-Order Contracts, JavaScript, Proxies
Digital Object Identifier 10.4230/DARTS.1.1.1
Related Article Matthias Keil and Peter Thiemann, "*TreatJS*: Higher-Order Contracts for JavaScript", in Proceedings of the 29th European Conference on Object-Oriented Programming (ECOOP 2015), LIPIcs, Vol. 37, pp. 28–51, 2015.
http://dx.doi.org/10.4230/LIPIcs.ECOOP.2015.28
Related Conference 29th European Conference on Object-Oriented Programming (ECOOP 2015), July 5–10, 2015, Prague, Czech Republic

1 Scope

The artifact is designed to support repeatability of all the experiments of the companion paper, allowing users to test the contract system on a variety of benchmarks. In particular, it allows to include *TreatJS* in existing JavaScript code, to specify contracts by plain JavaScript functions, to construct contracts by an unrestricted combination of other contracts, and to enforce contracts in all contexts of use.

2 Content

The artifact package includes:
- the main source of *TreatJS*;
- a set of test cases to examine the feature of the contract system;
- modified version of the Google Octaine 2.0 benchmark suite;
- detailed instructions for using the artifact, provided as an `index.html` file.

Dagstuhl Artifacts Series, Vol. 1, Issue 1, Artifact No. 1, pp. 01:1–01:2
DAGSTUHL ARTIFACTS SERIES Dagstuhl Artifacts Series
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

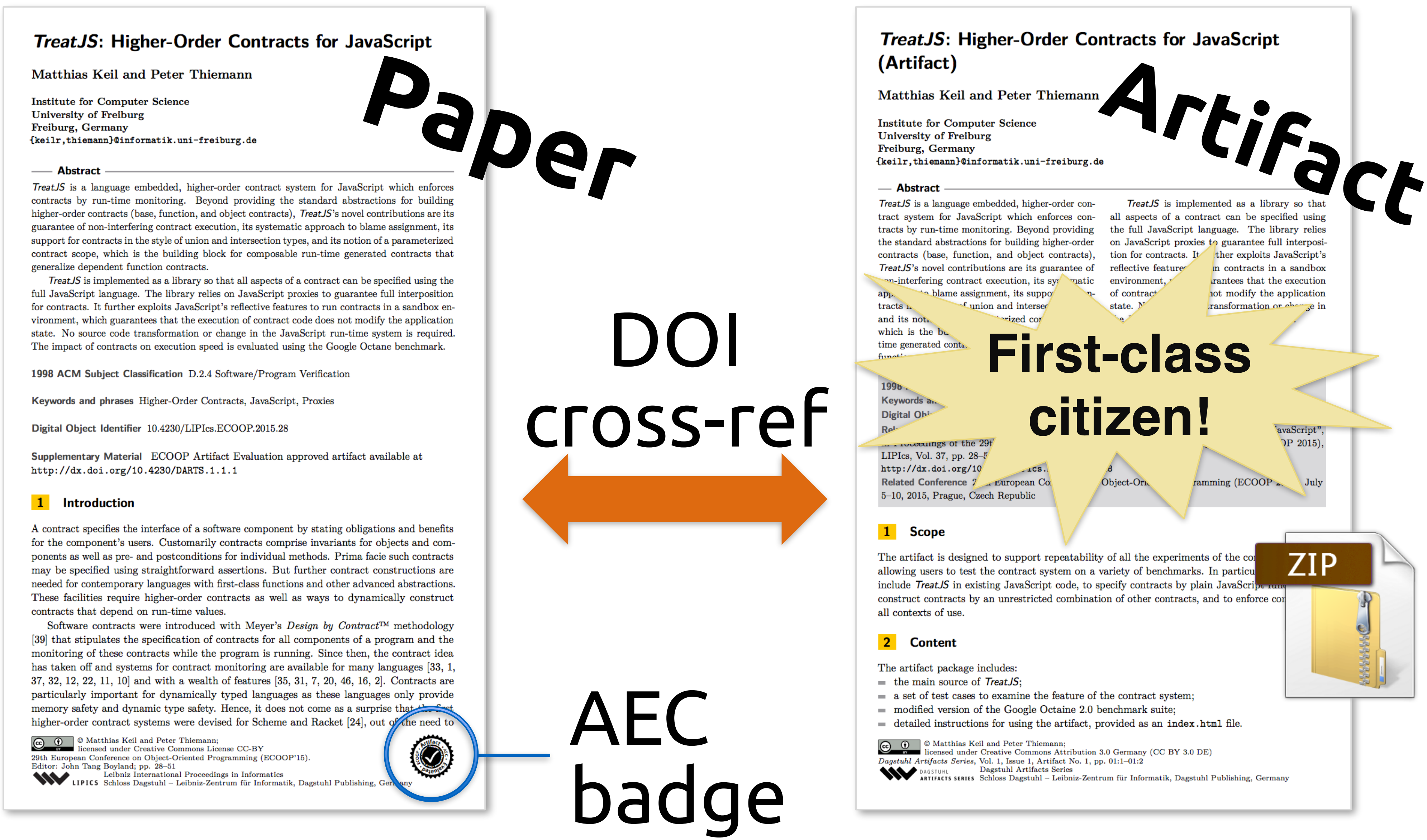
Title

Authors

Abstract

Metadata (DOI, etc.)

Scope, content, license, etc.



(c) Camil Demetrescu

Artifacts as first-class citizens

dblp
computer science bibliography

[+] [–] Jonathan Aldrich

> Home > Persons

[–] Person information

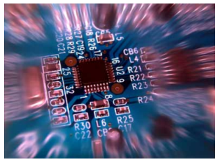
- *affiliation:* Carnegie Mellon University, Pittsburgh, USA

[–] 2010 – today

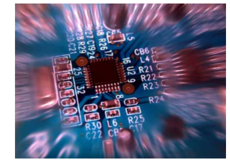
2015

[j10] Joseph Lee, Jonathan Aldrich, Troy Shaw, Alex Potanin, Benjamin Chung: **A Theory of Tagged Objects (Artifact).** DARTS 1(1): 03:1-03-3 (2015) — **artifact**

[c84] Joseph Lee, Jonathan Aldrich, Troy Shaw, Alex Potanin: **A Theory of Tagged Objects.** ECOOP 2015: 174-197 — **paper**



Conclusions



Develop open source benchmarks

Codify documentation, methodologies & reporting standards

Require executable artifacts

Publish reproduction studies