Run-time Evaluation of Opportunities for Object Inlining in Java

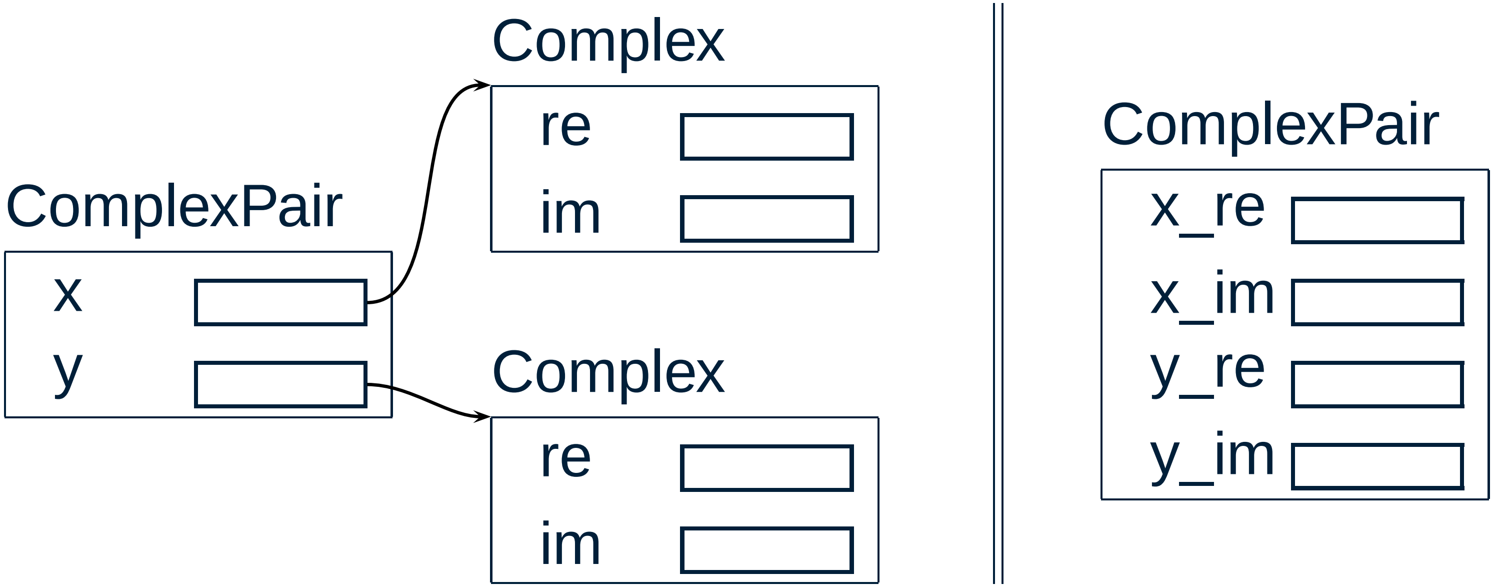
Ondřej Lhoták and Laurie Hendren

Sable Research Group
McGill University

November 5th, 2002

Motivation

- Java allows only references to objects as fields, not the objects themselves.
- Object Inlining has been studied as a method to implement languages with this restriction efficiently.



Motivation

- Java allows only references to objects as fields, not the objects themselves.
- Object Inlining has been studied as a method to implement languages with this restriction efficiently.

How would Object Inlining affect typical Java programs?

Contributions

- Classification scheme for fields according to how they can be inlined.
- Empirical limit study of the potential effects of object inlining (upper bound on improvements achievable by object inlining optimization).
- Technique for determining which inlinable fields are important to optimize — could be useful to programmers.
- Observation of complex interactions between object inlining and other optimizations: effect of “pointer chasing” is minor in comparison.

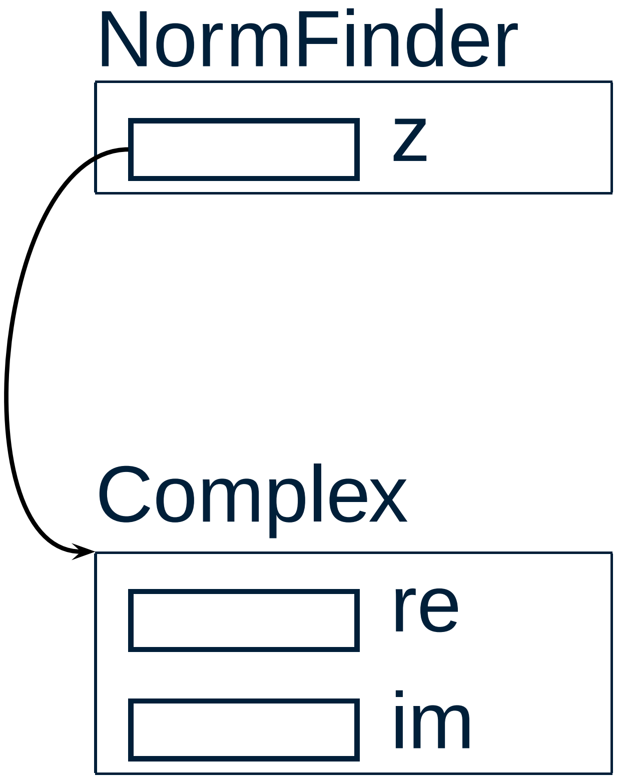
Outline

- Object Inlining and Related Work
- Definitions
- Experiments and Results
- Conclusion and Future Work

Object Inlining

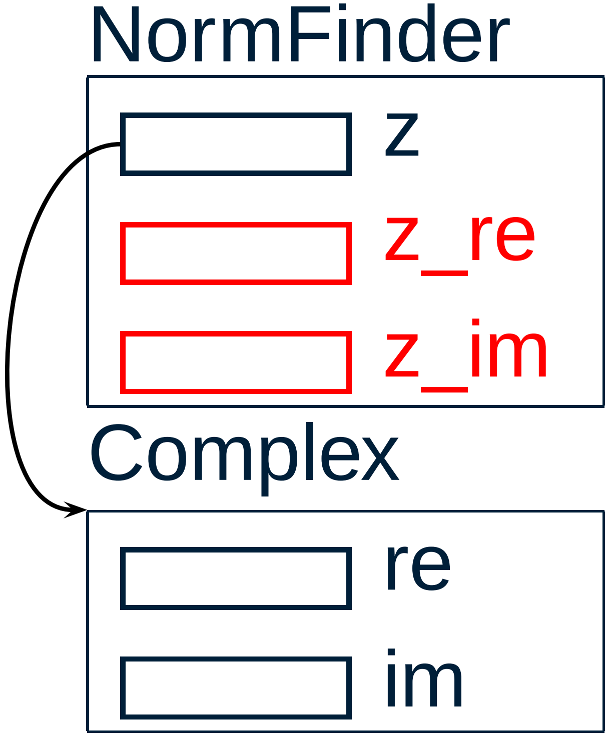
```
class Complex {
   double re, im;
}
class NormFinder {
   Complex z;

   double normSq() {
      return
         z.re*z.re +
         z.im*z.im;
   }
}
```



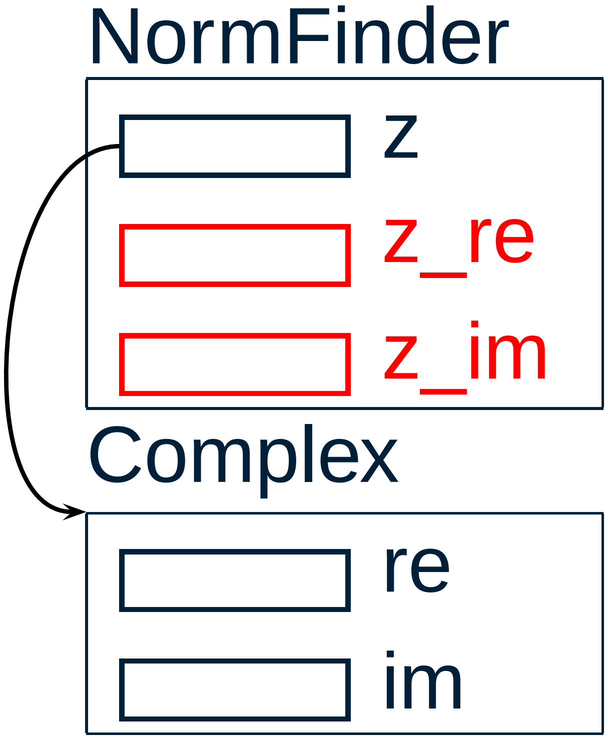
Object Inlining

```
class Complex {
   double re, im;
}
class NormFinder {
   Complex z;
   double z_re, z_im;
   double normSq() {
      return
         z.re*z.re +
         z.im*z.im;
   }
}
```



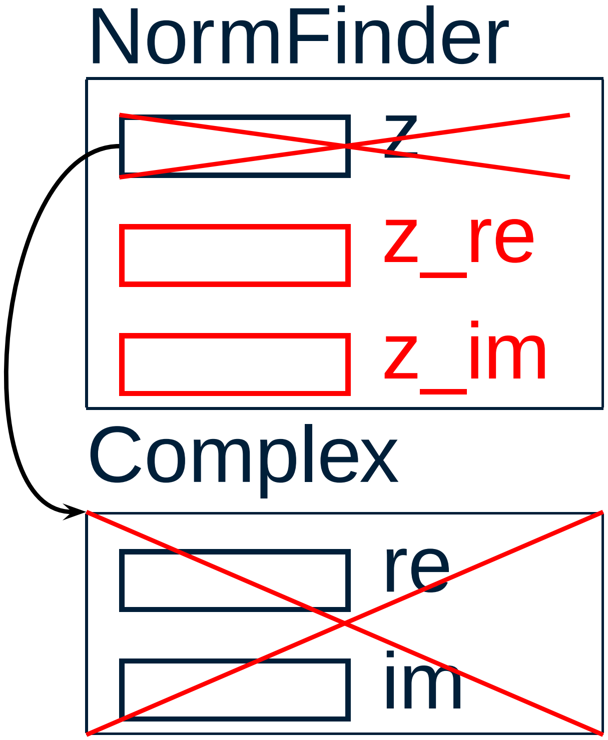
Object Inlining

```
class Complex {
   double re, im;
}
class NormFinder {
   Complex z;
   double z_re, z_im;
   double normSq() {
      return
         z_re*z_re +
         z_im*z_im;
   }
}
```



Object Inlining

```
class Complex {
    double re, im;
}
class NormFinder {
    ~~Complex z;~~
    double z_re, z_im;
    double normSq() {
        return
            z_re*z_re +
            z_im*z_im;
    }
}
```



Related Work

- Dolby, Chien.
    - PLDI ’97. Automatic Inline Allocation of Objects.
    - OOPSLA ’98. An Evaluation of Automatic Object Inline Allocation Techniques.
    - PLDI ’00. An Automatic Object Inlining Optimization and its Evaluation.
- Laud.
    - JOSES ’01 (ETAPS). Analysis for Object Inlining in Java.

Related Work

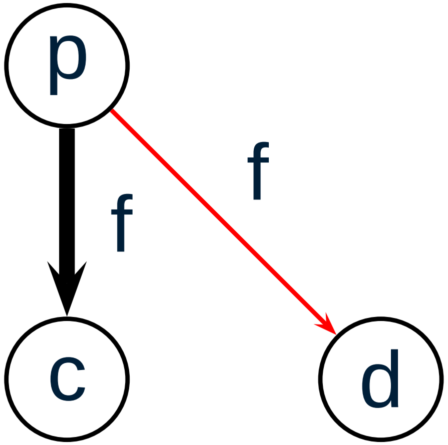
- Ghemawat, Randall, Scales.
  - PLDI ’00. Field Analysis: Getting Useful and Low-Cost Interprocedural Information.
- Budimlić.
  - Ph.D. thesis, 2001. Compiling Java for High Performance and the Internet.

Predicates

- [contains-unique] Every container having **f** refers to only one contained object through **f**.

p.f = c;

~~**p.f = d;**~~

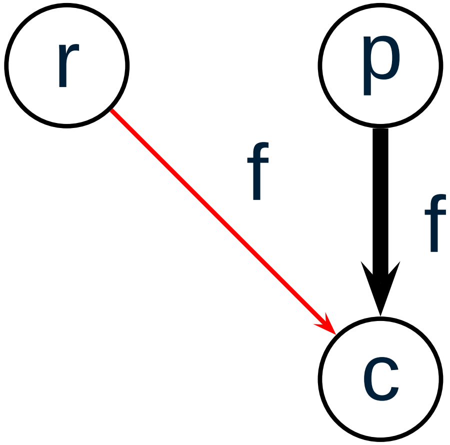


Predicates

- [unique-container-same-field] No object stored into field **f** is stored into field **f** of any other container.

p.f = c;

~~**r.f = c;**~~

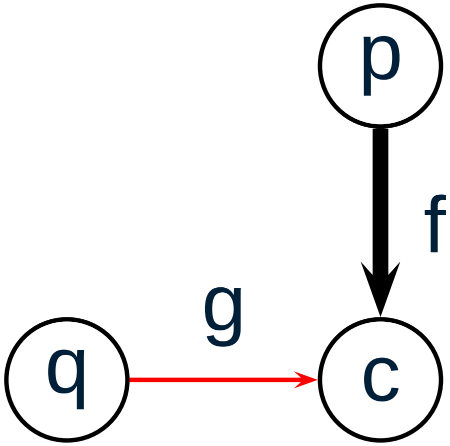


Predicates

- [unique-container-different-field] No object stored into field **f** is stored into any field **g** of any other container.

p.f = c;

~~**q.g = c;**~~



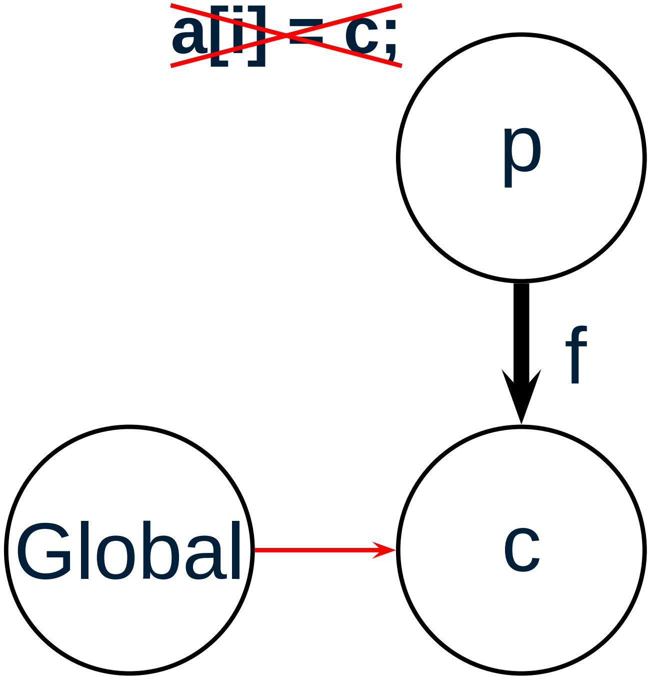
Predicates

- [not-globally-reachable] No object contained in **f** is ever stored into an array or static field.

p.f = c;

~~**Class.g = c;**~~

~~**a[i] = c;**~~



Field Classification

[contains-unique]

[unique-container-same-field]

[unique-container-different-field]

[not-globally-reachable]

Field Classification

→ [contains-unique]

→ [unique-container-same-field]

[unique-container-different-field]

[not-globally-reachable]

Field-sensitive

one-to-one

[Dolby, Chien]

Move fields to container

Remove contained

object

Field Classification

→ [contains-unique]

→ [unique-container-same-field] ←

[unique-container-different-field] ←

[not-globally-reachable] ←

Field-sensitive one-to-one	Unique-store
[Dolby, Chien]	[Laud]
Move fields to container Remove contained object	Copy fields to container Keep contained object

Field Classification

→ [contains-unique]

→ [unique-container-same-field] ←

[unique-container-different-field] ←

[not-globally-reachable] ←

Field-sensitive	All 4 predicates	Unique-store
one-to-one	Simply	
[Dolby, Chien]	one-to-one	[Laud]

Move fields to container	Copy fields to container
Remove contained	Keep contained
object	object

Experiments

- Instrument benchmarks using Soot to record **getfield**, **putfield**, **putstatic** and **aastore**.
- For each field, look for violations of each predicate in the traces.

eg.

```
   ⋮
p.f = c;
   ⋮
p.f = d;
   ⋮
```

$\Longrightarrow$ ~~[contains unique](**f**)~~

Experiments

- Instrument benchmarks using Soot to record **getfield**, **putfield**, **putstatic** and **aastore**.
- For each field, look for violations of each predicate in the traces.

eg.

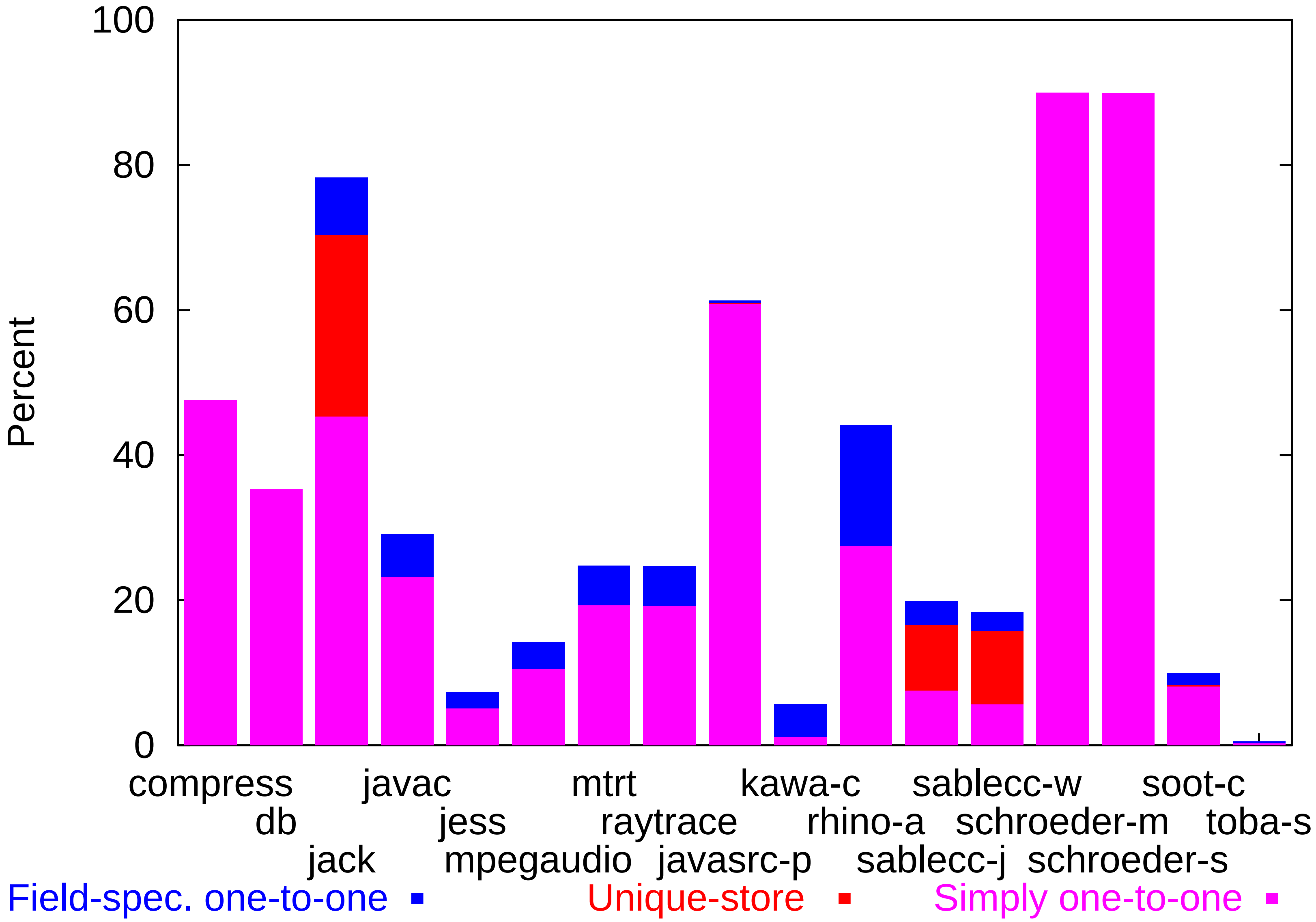
```
⋮
p.f = c;
⋮
q.f = c;
⋮
```

$\Longrightarrow$ ~~[unique-container-same-field](**f**)~~

Benchmarks

compress	javasrc-p	(Java to HTML)
db	kawa-c	(Scheme compiler)
jack	rhino-a	(Javascript interp.)
javac	sablecc-j	(Parser generator)
jess	sablecc-w	
mpegaudio	schroeder-m	(Audio editor)
mtrt	schroeder-s	
raytrace	soot-c	(Bytecode optimizer)
	toba-s	(Java native compiler)

Fraction of Field Reads Inlinable

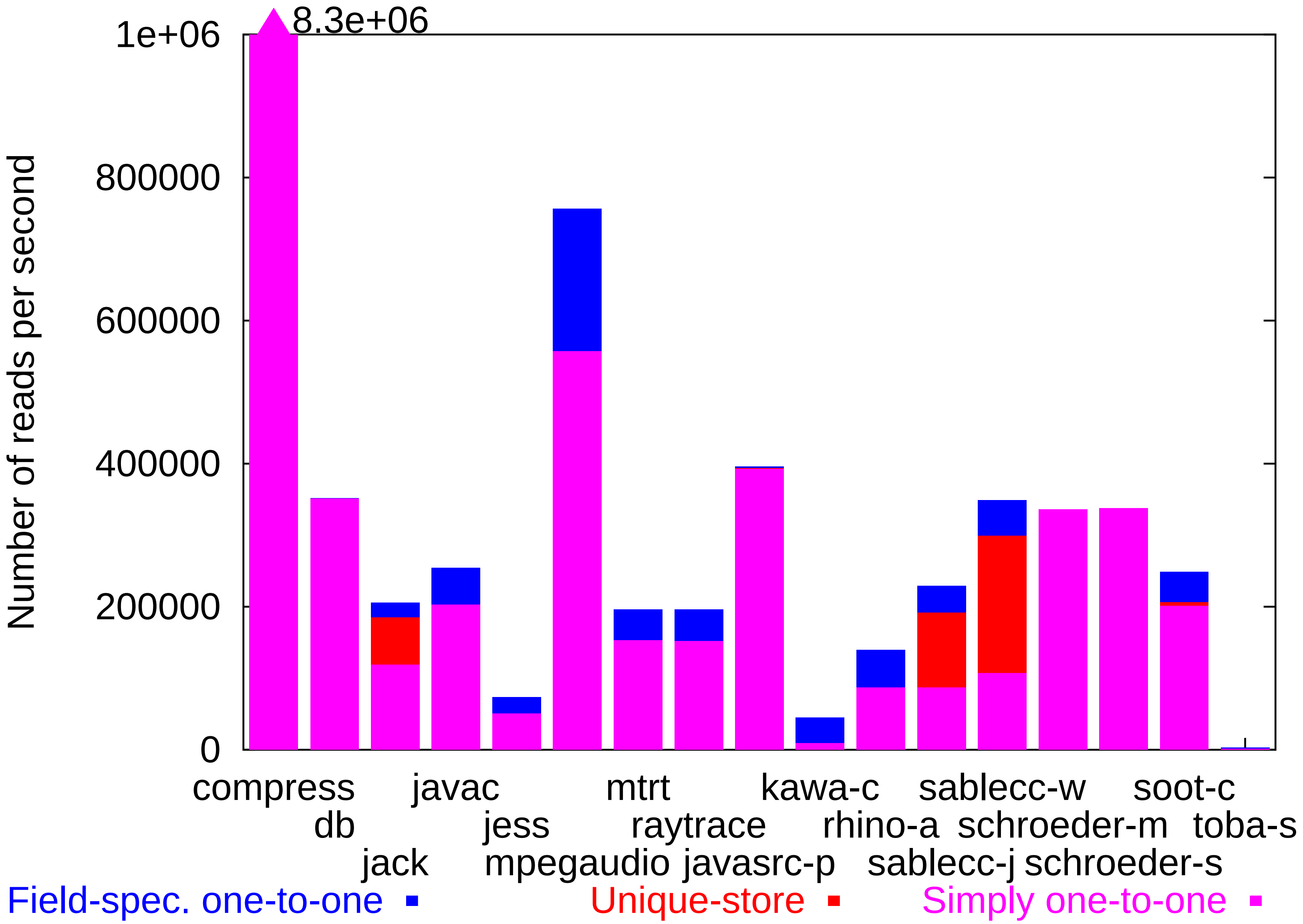


How Many Inlinable Fields are Important?

Fields accounting for 90% of inlinable field reads

compress	6	javasrc-p	6
db	1	kawa-c	20
jack	7	rhino-a	3
javac	8	sablecc-j	12
jess	5	sablecc-w	8
mpegaudio	4	schroeder-m	4
mtrt	5	schroeder-s	4
raytrace	5	soot-c	20
		toba-s	6

Inlinable Field Reads per Second



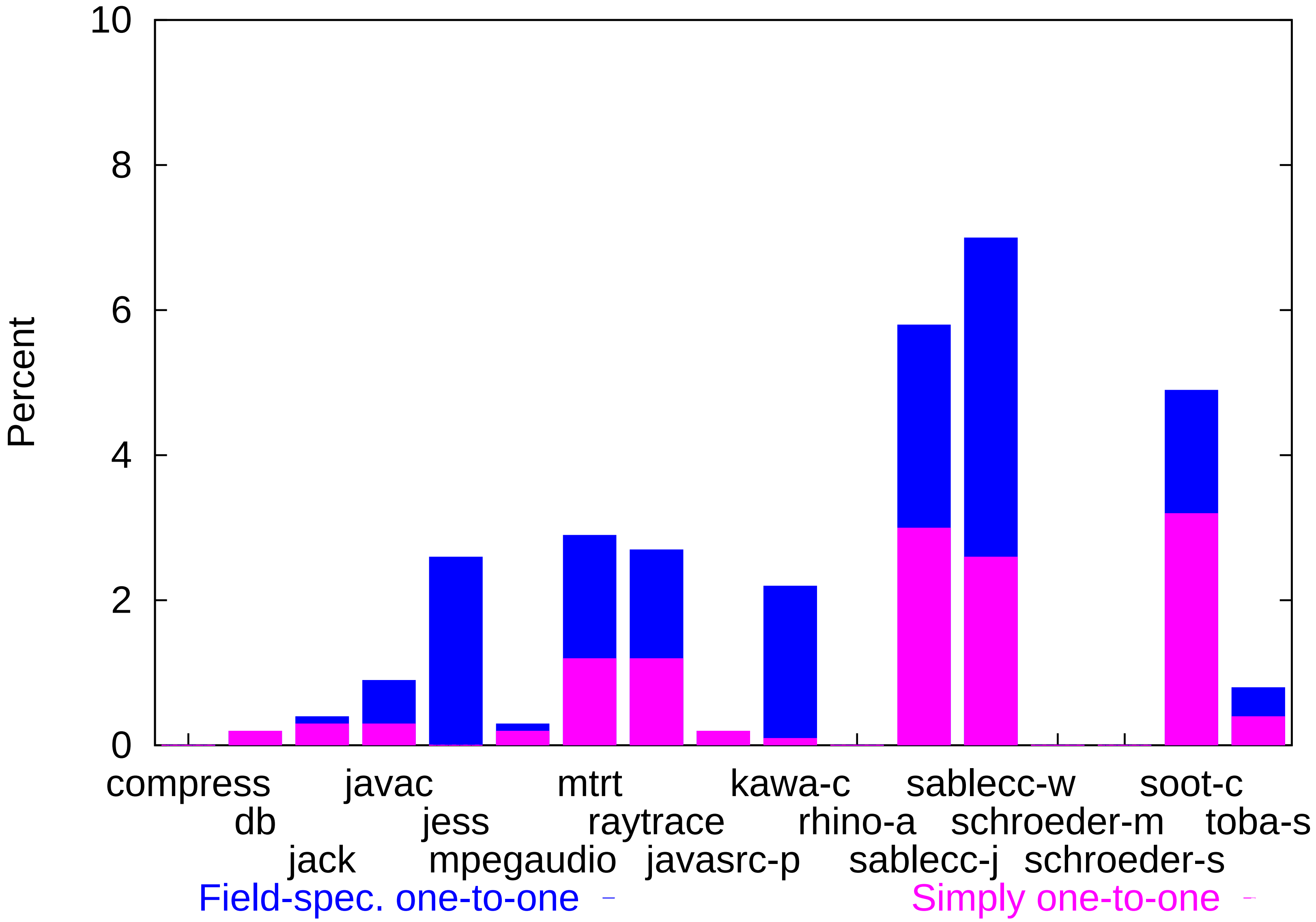
Speedup from Hand-Inlining

- compress
  - Speedup 7.8% to 10.8%
- db
  - Speedup up to 10.6% from one field
- javasrc-p
  - No significant change

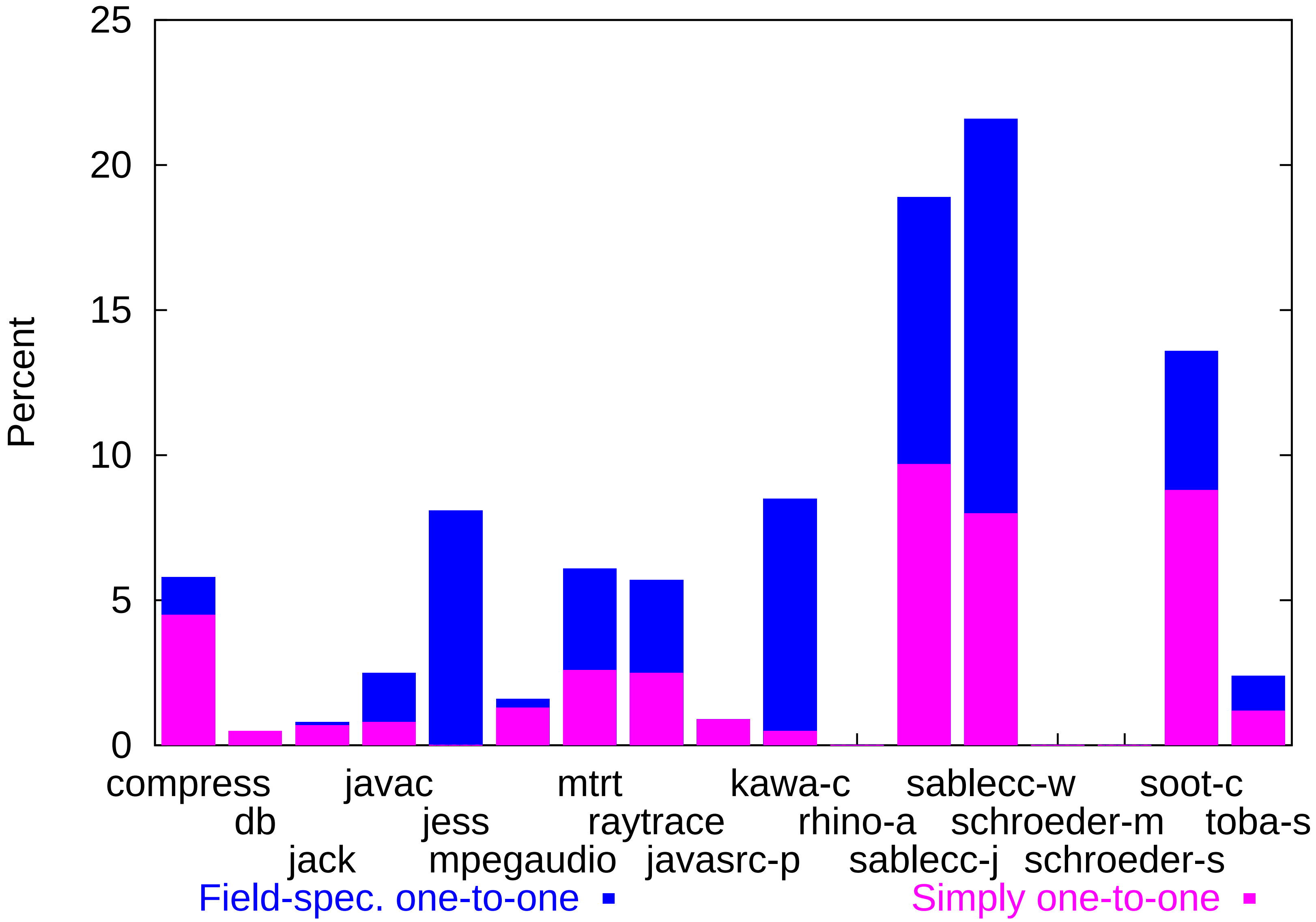
Loop Invariant Hoisting

- Fields satisfying [contains-unique] predicate are loop invariant.
- Hoisting loop invariants should give similar benefit.
  - In compress, benefit from loop invariant hoisting is about half the benefit of object inlining.

Bytes of Allocations Saved



Object Allocations Saved



Conclusions

- Object inlining can produce speedups of up to 10%, but highly dependent on individual benchmark.
- Complex interactions with other optimizations; cost of “pointer chasing” insignificant in comparison.
- Inlining field-specific one-to-one fields can yield savings of up to 7% of bytes, 21.6% of objects allocated.
- Small number of fields are important
  $\Rightarrow$ could be hand-optimized.