Type-Intertwined Heap Analysis

University of Colorado Boulder

Aarhus University
May 18, 2015
Lab: **Program analysis in the whole bug mitigation process**
Lab: **Program analysis** in the **whole bug mitigation process**
Lab: Program analysis in the whole bug mitigation process
Lab: Program analysis in the whole bug mitigation process
Lab: **Program analysis in the whole bug mitigation process**

Verifier

**Thresher: Goal-Directed Refutation Analysis**

[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]
Lab: Program analysis in the whole bug mitigation process

Verifier ✔ proof of no bug
Alarm ☒
Report

Program Manual

Triaging

Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]

Manual Triaging

Given a program configuration goal, derive a contradiction w.r.t. its reachability

\[(x \mapsto \hat{x} \ast \hat{x} \cdot f \mapsto \hat{a} \ast \text{true}) \land \hat{a} \neq \text{null}\]

backwards abstract interpretation of separation logic
Lab: **Program analysis in the whole bug mitigation process**
Lab: Program analysis in the whole bug mitigation process

Verifier

Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]
Lab: Program analysis in the whole bug mitigation process

Verifier

Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]
Program analysis in the whole bug mitigation process

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, in prep]

Verifier

Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]
Program analysis in the whole bug mitigation process

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, in prep]

Verifier

Test Input → Specification → Program

Tester

Test Output

proof of no bug

Alarm Report

Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]
Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, in prep]

Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]

Divva: Synthesizing Short-Circuiting Data Structure Checks
[under review]

Proof of no bug

Alarm Report
Lab: Program analysis in the whole bug mitigation process

**Fissile Types:** Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, in prep]

**Divva:** Synthesizing Short-Circuiting Data Structure Checks
[under review]

Use **static shape analysis** to synthesize short-circuiting dynamic validation of data structure invariants

**Thresher:** Goal-Directed Refutation Analysis
[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]

Manual Triaging
Program analysis in the whole bug mitigation process

**Fissile Types:** Checking Almost Everywhere Invariants

**Divva:** Synthesizing Short-Circuiting Data Structure Checks

**Thresher:** Goal-Directed Refutation Analysis

**Manual Triaging**

**Verifier**

**Runner**

**Test Input**

**Test Output**

**Verdict:** proof of no bug

**Alarm Report**
Program analysis in the whole bug mitigation process

**Fissile Types:** Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, in prep]

**Divva:** Synthesizing Short-Circuiting Data Structure Checks
[under review]

**Program Triaging**

**Github**

**Fixr:** Mining Bug Fixes from Commits
[in progress]

**Verifier**

**Test Input**

**Specification**

**Test Output**

**Test Runner**

**Thresher:** Goal-Directed Refutation Analysis
[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]
Program analysis in the whole bug mitigation process.

- **Lab:** Program analysis
- **Verifier**
  - **Divva:** Synthesizing Short-Circuiting Data Structure Checks
    [under review]
- **Jsana:** Abstract Domain Combinators for Dynamic Languages
  [Cox+ ECOOP’13, Cox+ SAS’14, Cox+ ESOP’15]
- **Thresher:** Goal-Directed Refutation Analysis
  [Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]

**Fissile Types:** Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, in prep]

**Program Triaging**
- **Github**
- **Fixr:** Mining Bug Fixes from Commits
  [in progress]
Program analysis in the whole bug mitigation process

- **Fissile Types:** Checking Almost Everywhere Invariants
  - [Coughlin+ POPL’14, in prep]

- **Divva:** Synthesizing Short-Circuiting Data Structure Checks
  - [under review]

- **Jsana:** Abstract Domain Combinators for Dynamic Languages
  - [Cox+ ECOOP’13, Cox+ SAS’14, Cox+ ESOP’15]

- **Divva:** Synthesizing Short-Circuiting Data Structure Checks
  - [under review]

- **Fixr:** Mining Bug Fixes from Commits
  - [in progress]

- **Heap with set symbols partitioning “open objects”**

- separation logic with open object predicates and desynchronized separation
Program analysis in the whole bug mitigation process:

**Fissile Types:** Checking Almost Everywhere Invariants
- [Coughlin+ POPL’14, in prep]

**Program:**
- Test Input
- Specification
- Program

**Verifier**
- Jsana: Abstract Domain Combinators for Dynamic Languages
  - [Cox+ ECOOP’13, Cox+ SAS’14, Cox+ ESOP’15]
- Thresher: Goal-Directed Refutation Analysis
  - [Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]
- Divva: Synthesizing Short-Circuiting Data Structure Checks
  - [under review]

**Manual Triaging**
- Github
- Fixr: Mining Bug Fixes from Commits
  - [in progress]

**Alarm Report**
This Talk

Fissile Types: Checking Almost Everywhere Invariants
[Coughlin+ POPL’14, in prep]

Thresher: Goal-Directed Refutation Analysis
[Blackshear+ SAS’11, Blackshear+ PLDI’13, under review]

Jsana: Abstract Domain Combinators for Dynamic Languages
[Cox+ ECOOP’13, Cox+ SAS’14, Cox+ ESOP’15]

Divva: Synthesizing Short-Circuiting Data Structure Checks
[under review]

Fixr: Mining Bug Fixes from Commits
[in progress]
Type-Intertwined Heap Analysis

Devin Coughlin
Ross Holland
Bor-Yuh Evan Chang

University of Colorado Boulder

Aarhus University
May 18, 2015
How to type check a program that is almost well-typed?
How to type check a program that is almost well-typed?
A motivating example

Property of interest: reflective method call safety

Specification system: dependent-refinement types
Property of interest:

**reflective method call** safety

**type** safety

Specification system:

**dependent-refinement** types

**simple** types
Property of interest: **reflective method call** safety

Specification system: **dependent-refinement** types
class Callback
  var sel: Str
  var obj: Obj

  def call()
    this.obj.[this.sel]()
Reflective method call is dispatch based on a run-time value

class Callback
  var sel: Str
  var obj: Obj

def call()
  this.obj.[this.sel]()
Reflective method call is dispatch based on a run-time value

class Callback
  var sel: Str
  var obj: Obj

def call()
  this.obj.[this.sel]()

Calls method with name (selector) stored in on sel object stored in obj

If sel held string "notifyClick" would call notifyClick() on obj.
Reflective method call is dispatch based on a run-time value

class Callback
    var sel: Str
    var obj: Obj

    def call()
        this.obj.[this.sel]()
class Callback
    var sel: Str
    var obj: Obj

def call()
    this.obj.[this.sel]()
A dependent-refinement type expresses the required relationship

```python
class Callback:
    var sel: Str
    var obj: Obj

    def call():
        this.obj.[this.sel]()
```

obj must "respond to" sel
A dependent-refinement type expresses the required relationship

class Callback
    var sel: Str
    var obj: Obj | r2 sel

def call()
    this.obj.[this.sel]()
class Callback

    var sel: Str
    var obj: Obj i r2 sel

    def call()
        this.obj.[this.sel]()
class Callback
    var sel: Str
    var obj: Obj

def call()
    this.obj.[this.sel]()
class Callback
  var sel: Str
  var obj: Obj | r2 sel

  def call()
    this.obj.[this.sel]()
Array indexing safety is a similar relationship

class Iterator
    var idx: Int
    var buf: Obj[] | indexedBy idx

    def get(): Obj
        return this.buf[this.idx]
Array indexing safety is a similar relationship

class Iterator
    var idx: Int
    var buf: Obj[]

    def get(): Obj
        return this.buf[this.idx] | indexedBy idx

    idx must be a valid index into buf
Array indexing safety is a similar relationship

class Iterator
    var idx: Int
    var buf: Obj[]

    def get(): Obj
        return this.buf[this.idx]

    | indexedBy idx
        idx must be a valid index into buf

Guarantees no ArrayOutOfBoundsException error here
Array indexing safety is a similar relationship

class Iterator
var idx: Int
var buf: Obj[]

def get(): Obj
    return this.buf[this.idx]

Recurring theme: **Relationships** are important to **many safety properties**
class Callback
  var sel: Str
  var obj: Obj | r2 sel

  def call()
    this.obj.[this.sel]()
class Callback
    var sel: Str
    var obj: Obj | r2 sel

    def call()
        this.obj.[this.sel]()

    def update(s: Str, o: Obj | r2 s)
        this.sel = s
        this.obj = o
Relationships can be broken

class Callback
    var sel: Str
    var obj: Obj | r2 sel

    def call()
        this.obj.[this.sel]()

    def update(s: Str, o: Obj | r2 s)
        this.sel = s
        this.obj = o
class Callback

    var sel: Str
    var obj: Obj | r2 sel

    def call()
    this.obj.[this.sel]()

    def update(s: Str, o: Obj | r2 s)
    this.sel = s
    this.obj = o
class Callback
  var sel: Str
  var obj: Obj | r2 sel

  def call()
    this.obj.[this.sel]()

  def update(s: Str, o: Obj | r2 s)
    this.sel = s
    this.obj = o

obj must **always** respond to sel

○ guaranteed to respond to s

_Type error: old obj may not respond to new sel_
Relationships can be broken

class Callback
    var sel: Str
    var obj: Obj | r2 sel

def call():
    this.obj.[this.sel]()

def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
class Callback
    var sel: Str
    var obj: Obj | r2 sel

    def call()
        this.obj.[this.sel]()

    def update(s: Str, o: Obj | r2 s)
        this.sel = s
        this.obj = o
Why is this a false alarm?

class Callback
  var sel: Str
  var obj: Obj | r2 sel

def call()
  this.obj.[this.sel]()

def update(s: Str, o: Obj | r2 s)
  this.sel = s
  this.obj = o

Reasoning by global invariant: call safe because relationship holds
class Callback
    var sel: Str
    var obj: Obj | r2 sel

def call()
    this.obj.[this.sel]()  

def update(s: Str, o: Obj | r2 s)
    this.sel = s
    this.obj = o

Reasoning by **global invariant**: call safe because relationship holds

Reasoning about **local effects** of imperative updates
Why is this a false alarm?

class Callback
  var sel: Str
  var obj: Obj | r2 sel

def call()
  this.obj.[this.sel]()  

def update(s: Str, o: Obj | r2 s)
  this.sel = s
  this.obj = o

Reasoning by global invariant: call safe because relationship holds

Reasoning about local effects of imperative updates
class Callback
var sel: Str
var obj: Obj | r2 sel

def call()
    this.obj.[this.sel]()

def update(s: Str, o: Obj | r2 s)
    this.sel = s
    this.obj = o
class Callback
var sel: Str
var obj: Obj | r2 sel

def call()
this.obj.[this.sel]()
Why is this a false alarm?

class Callback:
    var sel: Str
    var obj: Obj | r2 sel

    def call():
        this.obj.[this.sel]()

    def update(s: Str, o: Obj | r2 s):
        this.sel = s
        this.obj = o

Reasoning by **global invariant**: call safe because relationship holds

Relationship violated

Relationship restored

**Reasoning about local effects** of imperative updates
two reasoning styles
Idea: Selectively alternate between and intertwine these two reasoning styles in verification.
Idea: Selectively alternate between and intertwine these two reasoning styles in verification.

\[ x : \tau, \ldots \]
flow-insensitive typing

\[ x \mapsto \hat{a} \ast \hat{a} \cdot f \mapsto \hat{v} \ast \ldots \]
flow-sensitive abstract interpretation
Verification of almost-everywhere invariants with intertwined type- and separation logic-based analysis
Verification of **almost-everywhere invariants** with intertwined type- and separation logic-based analysis

Switch to separation logic on **type error**: when **global type invariant violated**

- types
- analysis time
- X types violated
Verification of almost-everywhere invariants with intertwined type- and separation logic-based analysis.
Verification of **almost-everywhere invariants** with intertwined
type- and separation logic-based analysis

- **Types**
  - **Types violated**
  - **Types restored**

Switch to separation logic on **type error**: when **global type invariant violated**

Back to types when **invariant restored**
Verification of almost-everywhere invariants with intertwined type- and separation logic-based analysis

Switch to separation logic on type error: when global type invariant violated

Back to types when invariant restored

- types violated
- types restored
- types violated
- types restored

Analysis time
Verification of almost-everywhere invariants with intertwined type- and separation logic-based analysis
Key Ideas

ok

Tolerating temporary violations with almost type-consistent heaps

Coughlin and Chang. POPL 2014.
Key Ideas

ok  Tolerating temporary violations with almost type-consistent heaps
    Coughlin and Chang. POPL 2014.

⚠️ Type-intertwined framing with gated separation
    Under preparation.
Key Ideas

ok Tolerating temporary violations with almost type-consistent heaps
   Coughlin and Chang. POPL 2014.

⚠️ Type-intertwined framing with gated separation
   Under preparation.
Key Ideas

Tolerating temporary violations with almost type-consistent heaps
Coughlin and Chang. POPL 2014.

Type-intertwined framing with gated separation
Under preparation.
Extend separation logic with a summary for an almost type-consistent heap
Extend separation logic with a summary for an almost type-consistent heap.
Extend separation logic with a summary for an almost type-consistent heap

Summarize cells that are **not immediately type-inconsistent** into single formula literal.

- Immediately type-inconsistent
- Local
Extend separation logic with a summary for an almost type-consistent heap

Summarize cells that are not immediately type-inconsistent into single formula literal
Extend separation logic with a summary for an almost type-consistent heap

Summarize cells that are not immediately type-inconsistent into single formula literal

Describes storage without explicitly enumerating it

local
Extend separation logic with a **summary** for an **almost type-consistent heap**

- Summarize cells that are **not immediately type-inconsistent** into single formula literal
- Describes storage **without explicitly enumerating it**
- Value stored in location **differs from location’s declared type**
Extend separation logic with a summary for an almost type-consistent heap

Summarize cells that are not immediately type-inconsistent into single formula literal

Not immediately type-inconsistent but still transitively type-inconsistent

Value stored in location differs from location’s declared type

Describes storage without explicitly enumerating it
Extend separation logic with a summary for an almost type-consistent heap

Summarize cells that are **not immediately type-inconsistent** into single formula literal

Describes storage **without explicitly enumerating** it

Not immediately type-inconsistent but still transitively type-inconsistent

Value stored in location **differs from location’s declared type**

\[ \text{ok} \ast \hat{a} \cdot f \leftrightarrow \hat{v} \]
Extend separation logic with a summary for an almost type-consistent heap

Summarize cells that are not immediately type-inconsistent into single formula literal

Describes storage without explicitly enumerating it

Not immediately type-inconsistent but still transitively type-inconsistent

Value stored in location differs from location’s declared type

Split heap into two regions: almost type-consistent and (potentially) immediately type-inconsistent
Separation logic can **materialize** from the almost type-consistent summary

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

![Diagram showing the update function and its effects on `this`, `s`, and `o`]
Separation logic can **materialize** from the almost type-consistent summary

**Handoff** to separation logic on type error, heap is **consistent** with declared types

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

![Diagram showing separation logic]
Separation logic can **materialize** from the almost type-consistent summary

**Handoff** to separation logic on type error, **heap is consistent** with declared types

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

Entire heap consists of the almost type-consistent summary
Separation logic can **materialize** from the almost type-consistent summary

```python
def update(s: Str, o: Obj | r2 s)
  this.sel = s
  this.obj = o
```

**Handoff** to separation logic on type error, **heap is consistent** with declared types

```
Entire heap consists of the almost type-consistent summary
```
Separation logic can **materialize** from the almost type-consistent summary

**Handoff** to separation logic on type error, heap is **consistent** with declared types

```python
def update(s: Str, o: Obj | r2 s)
    this.sel = s
    this.obj = o
```

Entire heap consists of the almost type-consistent summary

Analysis reasons explicitly about contents of materialized cells
Values in `materialized` storage allowed to differ from declared types

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```
Values in **materialized** storage allowed to differ from declared types

Analysis performs **strong updates** on materialized cells

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```
Values in **materialized** storage allowed to differ from declared types

Analysis performs **strong updates** on materialized cells

```python
def update(s: Str, o: Obj | r2 s)
    this.sel = s
    this.obj = o
```
Values in **materialized** storage allowed to differ from declared types

Analysis performs **strong updates** on materialized cells

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

After first update, storage is **immediately type-inconsistent**
Values in **materialized** storage allowed to differ from declared types

**Analysis performs strong updates** on materialized cells

```python
def update(s: Str, o: Obj | r2 s)
    this.sel = s
    this.obj = o
```

After first update, storage is **immediately type-inconsistent**

this.obj does not respond to this.sel
Values in **materialized** storage allowed to differ from declared types

Analysis performs **strong updates** on materialized cells

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

After first update, storage is **immediately type-inconsistent**

```
this.obj does not respond to this.sel
```

```
ok
```

```
``
Values in **materialized** storage allowed to differ from declared types

Analysis performs **strong updates** on materialized cells

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

After second update, storage is no longer immediately type-inconsistent again
Can **summarize** not immediately type-inconsistent locations back into the almost type-consistent heap

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

```
                     ok
                o ->
             s ->
        this.sel ->
        this.obj ->
```

This diagram shows the update process, where `this.sel` and `this.obj` are updated with `s` and `o` respectively.
Can **summarize** not immediately type-inconsistent locations back into the almost type-consistent heap

```python
def update(s: Str, o: Obj | r2 s)
    this.sel = s
    this.obj = o
```
Can summarize not immediately type-inconsistent locations back into the almost type-consistent heap.

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

No explicit reasoning about summarized storage.
Can **summarize** not immediately type-inconsistent locations back into the almost type-consistent heap

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

If **entire heap** does **not** have immediately type-inconsistent locations then the heap is type-consistent
Can **summarize** not immediately type-inconsistent locations back into the almost type-consistent heap

```python
def update(s: Str, o: Obj | r2 s):
    this.sel = s
    this.obj = o
```

If **entire heap** does **not** have immediately type-inconsistent locations then the heap is type-consistent
Soundness

**Theorem (Soundness of Handoff).**

The entire state is **type-consistent** iff all locations are not immediately **type-inconsistent**.
**Theorem** (*Soundness of Handoff*).

The entire state is **type-consistent** iff all locations are **not immediately type-inconsistent**.

**Theorem** (*Soundness of Materialization/Summarization*).

Locations that are **not immediately type-inconsistent** can be safely *materialized* and *summarized* into the almost type-consistent heap **ok**.
Case Study: Reflection in Objective-C

Prototype analysis implementation
Plugin for clang static analyzer in C++

9 Objective-C benchmarks
6 libraries and 3 applications
1,000 to 176,000 lines of code

Manual type annotations
76 r2 annotations on system libraries
136 annotations on benchmark code
Case Study: Reflection in Objective-C

Prototype analysis implementation
  Plugin for clang static analyzer in C++

9 Objective-C benchmarks
  6 libraries and 3 applications
  1,000 to 176,000 lines of code

Manual type annotations
  76 r2 annotations on system libraries
  136 annotations on benchmark code

Including Skim, Adium, and OmniGraffle
Case Study: Reflection in Objective-C

Why Objective-C?
Statically typed plus reflective method call

Prototype analysis implementation
Plugin for clang static analyzer in C++

9 Objective-C benchmarks
6 libraries and 3 applications
1,000 to 176,000 lines of code

Manual type annotations
76 r2 annotations on system libraries
136 annotations on benchmark code

Including Skim, Adium, and OmniGraffle
Empirical Evaluation Questions

**Precision:** What is improvement over flow-insensitive checking alone?

**Cost:** What is the cost of analysis in running time?
<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>reflective call sites</th>
<th>flow-insensitive</th>
<th>almost-everywhere</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>7</td>
<td>7</td>
<td>2 (-71%)</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>12</td>
<td>2</td>
<td>0 (-100%)</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0</td>
<td>0 (-)</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>40</td>
<td>4</td>
<td>1 (-75%)</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>68</td>
<td>50</td>
<td>10 (-80%)</td>
</tr>
<tr>
<td>OmniFramework</td>
<td>160769</td>
<td>192</td>
<td>82</td>
<td>74 (-10%)</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>186</td>
<td>59</td>
<td>38 (-36%)</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>207</td>
<td>43</td>
<td>43 (-0%)</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>587</td>
<td>87</td>
<td>70 (-20%)</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>1327</strong></td>
<td><strong>334</strong></td>
<td><strong>238 (-29%)</strong></td>
</tr>
<tr>
<td>benchmark</td>
<td>size (loc)</td>
<td>reflective cases</td>
<td>flow-insensitive</td>
<td>almost-everywhere</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
<td>------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>7</td>
<td>7</td>
<td>2 (-71%)</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>12</td>
<td>2</td>
<td>0 (-100%)</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0</td>
<td>0 (-)</td>
</tr>
<tr>
<td>SPARKLE</td>
<td>5289</td>
<td>40</td>
<td>4</td>
<td>1 (-75%)</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>68</td>
<td>50</td>
<td>10 (-80%)</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>192</td>
<td>82</td>
<td>74 (-10%)</td>
</tr>
<tr>
<td>VIENNA</td>
<td>37327</td>
<td>186</td>
<td>59</td>
<td>38 (-36%)</td>
</tr>
<tr>
<td>SKIM</td>
<td>60211</td>
<td>207</td>
<td>43</td>
<td>43 (-0%)</td>
</tr>
<tr>
<td>ADIUM</td>
<td>176629</td>
<td>587</td>
<td>87</td>
<td>70 (-20%)</td>
</tr>
<tr>
<td>combined</td>
<td>461080</td>
<td>1327</td>
<td>334</td>
<td>238 (-29%)</td>
</tr>
</tbody>
</table>

**Baseline**: standard, flow-insensitive type analysis – no switching
### Precision

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>reflective call sites</th>
<th>flow-insensitive</th>
<th>false alarms (almost-everywhere)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>7</td>
<td>7</td>
<td>2 (-71%)</td>
</tr>
<tr>
<td>SCRECORDEr</td>
<td>2716</td>
<td>12</td>
<td>2</td>
<td>0 (-100%)</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0</td>
<td>0 (-)</td>
</tr>
<tr>
<td>SPARKLE</td>
<td>5289</td>
<td>40</td>
<td>4</td>
<td>1 (-75%)</td>
</tr>
<tr>
<td>ASIHTTPREQUEST</td>
<td>14620</td>
<td>68</td>
<td>50</td>
<td>10 (-80%)</td>
</tr>
<tr>
<td>OmniFrameWORKS</td>
<td>160769</td>
<td>192</td>
<td>82</td>
<td>74 (-10%)</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>186</td>
<td>59</td>
<td>38 (-36%)</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>207</td>
<td>43</td>
<td>43 (-0%)</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>587</td>
<td>87</td>
<td>70 (-20%)</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>1327</strong></td>
<td><strong>334</strong></td>
<td><strong>238 (-29%)</strong></td>
</tr>
</tbody>
</table>

**Baseline:** standard, **flow-insensitive** type analysis – no switching
<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>reflective call sites</th>
<th>flow-insensitive</th>
<th>almost-everywhere</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>7</td>
<td>7</td>
<td>2 (-71%)</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>12</td>
<td>2</td>
<td>0 (-100%)</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>28</td>
<td>0</td>
<td>0 (-)</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>40</td>
<td>4</td>
<td>1 (-75%)</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>68</td>
<td>50</td>
<td>10 (-80%)</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>192</td>
<td>82</td>
<td>74 (-10%)</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>186</td>
<td>59</td>
<td>38 (-36%)</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>207</td>
<td>43</td>
<td>43 (-0%)</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>587</td>
<td>87</td>
<td>70 (-20%)</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>1327</strong></td>
<td><strong>334</strong></td>
<td><strong>238 (-29%)</strong></td>
</tr>
</tbody>
</table>

**Baseline:** standard, **flow-insensitive** type analysis – no switching.

Almost everywhere techniques show 29% improvement in false alarms.
<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>reflective call sites</th>
<th>flow-insensitive</th>
<th>almost-everywhere</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>7</td>
<td>2</td>
<td>2 (-71%)</td>
</tr>
<tr>
<td></td>
<td>2716</td>
<td>12</td>
<td>2</td>
<td>0 (-100%)</td>
</tr>
<tr>
<td></td>
<td>3301</td>
<td>28</td>
<td>0</td>
<td>0 (-)</td>
</tr>
<tr>
<td></td>
<td>5289</td>
<td>40</td>
<td>4</td>
<td>1 (-75%)</td>
</tr>
<tr>
<td></td>
<td>14620</td>
<td>68</td>
<td>50</td>
<td>10 (-80%)</td>
</tr>
<tr>
<td></td>
<td>160769</td>
<td>192</td>
<td>82</td>
<td>74 (-10%)</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>186</td>
<td>59</td>
<td>38 (-36%)</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>207</td>
<td>43</td>
<td>43 (-0%)</td>
</tr>
<tr>
<td>Aidiom</td>
<td>176629</td>
<td>587</td>
<td>87</td>
<td>70 (-20%)</td>
</tr>
<tr>
<td>combined</td>
<td>461080</td>
<td>1327</td>
<td>334</td>
<td>238 (-29%)</td>
</tr>
</tbody>
</table>

**Baseline:** standard, **flow-insensitive** type analysis – no switching.

Almost everywhere techniques show 29% improvement in false alarms.

Also found a **real reflection bug** in Vienna, which we reported and which was fixed.
## Cost: Analysis Time

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>analysis time (Time)</th>
<th>Rate (kloc/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>0.24s</td>
<td>5.3</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>0.28s</td>
<td>10.8</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>0.10s</td>
<td>33</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>0.67s</td>
<td>7.9</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>0.50s</td>
<td>27.2</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>4.25s</td>
<td>37.8</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>2.79s</td>
<td>13.4</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>2.49s</td>
<td>24.1</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>8.79s</td>
<td>20.1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>20.09s</strong></td>
<td><strong>23</strong></td>
</tr>
</tbody>
</table>
## Cost: Analysis Time

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>analysis time</th>
<th>Rate (kloc/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAUTH</td>
<td>1248</td>
<td>0.24s</td>
<td>5.3</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>0.28s</td>
<td>10.8</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>0.10s</td>
<td>33</td>
</tr>
<tr>
<td>SKIM</td>
<td>37327</td>
<td>2.79s</td>
<td>13.4</td>
</tr>
<tr>
<td>SKIM</td>
<td>60211</td>
<td>2.49s</td>
<td>24.1</td>
</tr>
<tr>
<td>AIDUM</td>
<td>176629</td>
<td>8.79s</td>
<td>20.1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>20.09s</strong></td>
<td><strong>23</strong></td>
</tr>
</tbody>
</table>

Includes analysis time but **not parsing, base type checking**.
## Cost: Analysis Time

<table>
<thead>
<tr>
<th>Name</th>
<th>Size (loc)</th>
<th>Analysis Time (s)</th>
<th>Rate (kloc/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OA</td>
<td>1248</td>
<td>0.24</td>
<td>5.3</td>
</tr>
<tr>
<td>SCR</td>
<td>2735</td>
<td>0.28</td>
<td>10.8</td>
</tr>
<tr>
<td>Z</td>
<td>3301</td>
<td>0.10</td>
<td>33</td>
</tr>
<tr>
<td>S</td>
<td>5289</td>
<td>0.67</td>
<td>7.9</td>
</tr>
<tr>
<td>ASIHTTPR</td>
<td>14620</td>
<td>0.50</td>
<td>27.2</td>
</tr>
<tr>
<td>OMNI</td>
<td>160769</td>
<td>4.25</td>
<td>37.8</td>
</tr>
<tr>
<td>VIENNA</td>
<td>37327</td>
<td>2.79</td>
<td>13.4</td>
</tr>
<tr>
<td>SKIM</td>
<td>60211</td>
<td>2.49</td>
<td>24.1</td>
</tr>
<tr>
<td>ADIUM</td>
<td>176629</td>
<td>8.79</td>
<td>20.1</td>
</tr>
<tr>
<td><strong>Combined</strong></td>
<td>461080</td>
<td>20.09</td>
<td>23</td>
</tr>
</tbody>
</table>

*Includes analysis time but not parsing, base type checking. Does not include system headers.*
## Cost: Analysis Time

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>analysis time (s)</th>
<th>Rate (kloc/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>0.24s</td>
<td>5.3</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>0.28s</td>
<td>10.8</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>0.10s</td>
<td>33</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>0.67s</td>
<td>7.9</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>0.50s</td>
<td>27.2</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>4.25s</td>
<td>37.8</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>2.79s</td>
<td>13.4</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>2.49s</td>
<td>24.1</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>8.79s</td>
<td>20.1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>20.09s</strong></td>
<td><strong>23</strong></td>
</tr>
</tbody>
</table>

**Fast:** 5 to 38 kloc/s with most time spent analyzing system headers
### Cost: Analysis Time

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Size (loc)</th>
<th>Analysis Time (s)</th>
<th>Rate (kloc/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>0.24s</td>
<td>5.3</td>
</tr>
<tr>
<td>SCRRecorder</td>
<td>2716</td>
<td>0.28s</td>
<td>10.8</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>0.10s</td>
<td>33</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>0.67s</td>
<td>7.9</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>0.50s</td>
<td>27.2</td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>4.25s</td>
<td>37.8</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>2.79s</td>
<td>13.4</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>2.49s</td>
<td>24.1</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>8.79s</td>
<td>20.1</td>
</tr>
<tr>
<td><strong>Combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>20.09s</strong></td>
<td><strong>23</strong></td>
</tr>
</tbody>
</table>

**Fast**: 5 to 38 kloc/s with most time spent analyzing system headers

**Interactive speeds**
### Cost: Analysis Time

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>analysis time (Time)</th>
<th>Rate (kloc/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>0.24s</td>
<td>5.3</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>0.28s</td>
<td>10.8</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>0.10s</td>
<td>33</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>0.67s</td>
<td>7.9</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>0.50s</td>
<td><strong>27.2</strong></td>
</tr>
<tr>
<td>OmniFrameworks</td>
<td>160769</td>
<td>4.25s</td>
<td><strong>37.8</strong></td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>2.79s</td>
<td>13.4</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>2.49s</td>
<td>24.1</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>8.79s</td>
<td>20.1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>20.09s</strong></td>
<td><strong>23</strong></td>
</tr>
</tbody>
</table>

**Fast:** 5 to 38 kloc/s with most time spent analyzing system headers

**Higher rate** for projects with larger translation units
## Cost: Analysis Time

<table>
<thead>
<tr>
<th>benchmark</th>
<th>size (loc)</th>
<th>analysis time (Time)</th>
<th>Rate (kloc/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OAuth</td>
<td>1248</td>
<td>0.24s</td>
<td>5.3</td>
</tr>
<tr>
<td>SCRecorder</td>
<td>2716</td>
<td>0.28s</td>
<td>10.8</td>
</tr>
<tr>
<td>ZipKit</td>
<td>3301</td>
<td>0.10s</td>
<td>33</td>
</tr>
<tr>
<td>Sparkle</td>
<td>5289</td>
<td>0.67s</td>
<td>7.9</td>
</tr>
<tr>
<td>ASIHTTPRequest</td>
<td>14620</td>
<td>0.50s</td>
<td>27.2</td>
</tr>
<tr>
<td>OmniFramework</td>
<td>160769</td>
<td>4.25s</td>
<td>37.8</td>
</tr>
<tr>
<td>Vienna</td>
<td>37327</td>
<td>2.79s</td>
<td>13.4</td>
</tr>
<tr>
<td>Skim</td>
<td>60211</td>
<td>2.49s</td>
<td>24.1</td>
</tr>
<tr>
<td>Adium</td>
<td>176629</td>
<td>8.79s</td>
<td>20.1</td>
</tr>
<tr>
<td><strong>combined</strong></td>
<td><strong>461080</strong></td>
<td><strong>20.09s</strong></td>
<td><strong>23</strong></td>
</tr>
</tbody>
</table>

**Fast:** 5 to 38 kloc/s with most time spent analyzing system headers. Maintains key benefit of flow-insensitive analyses: speed.
Key Ideas

ok

Tolerating temporary violations with almost type-consistent heaps
Coughlin and Chang. POPL 2014.

⚠️

Type-intertwined framing with gated separation
Under preparation.
Key Ideas

Tolerating temporary violations with almost type-consistent heaps

Coughlin and Chang. POPL 2014.

Type-intertwined framing with gated separation

Under preparation.
Key Ideas

Tolerating temporary violations with almost type-consistent heaps
Coughlin and Chang. POPL 2014.

Type-intertwined framing with gated separation

Under preparation.
Recall: Soundness

**Theorem (Soundness of Handoff).**

The entire state is **type-consistent** iff all locations are **not immediately type-inconsistent**.

![Diagram of Theorem (Soundness of Handoff).]

**Theorem (Soundness of Materialization/Summarization).**

Locations that are **not immediately type-inconsistent** can be safely **materialized and summarized** into the almost type-consistent heap **ok**.

![Diagram of Theorem (Soundness of Materialization/Summarization).]
Recall: Soundness

**Theorem (Soundness of Handoff).**
The entire state is *type-consistent* iff all locations are *not immediately type-inconsistent*.

![Diagram of the theorem](image)

**Theorem (Soundness of Materialization/Summarization).**
Locations that are *not immediately type-inconsistent* can be safely materialized and summarized into the almost type-consistent heap *ok*.

![Diagram of the theorem](image)
Theorem (Soundness of Handoff).

The entire state is type-consistent iff all locations are not immediately type-inconsistent.

Theorem (Soundness of Materialization/Summarization).

Locations that are not immediately type-inconsistent can be safely materialized and summarized into the almost type-consistent heap ok.
Recall: Soundness

**Theorem (Soundness of Handoff).**

The entire state is **type-consistent** iff all locations are **not immediately** type-inconsistent.

---

[Fissile Types, POPL 2014]

No way to **frame** and then handoff to types
Type-intertwined frame rule with standard separating conjunction is **unsound**

```python
def badUpdate(s: Str, o: Obj | r2 s):
    this.sel = s
    this.call()
    this.obj = o
```
Type-intertwined frame rule with standard separating conjunction is **unsound**

Immediately type-inconsistent portion of heap is **disjoint** from almost type-consistent summary

```python
def badUpdate(s: Str, o: Obj | r2 s):
    this.sel = s
    this.call()
    this.obj = o
```
Type-intertwined frame rule with standard separating conjunction is **unsound**

```python
def badUpdate(s: Str, o: Obj | r2 s):
    this.sel = s
    this.call()
    this.obj = o
```

Immediately type-inconsistent portion of heap is **disjoint** from almost type-consistent summary.

After framing out, entire heap is almost type-consistent.
Type-intertwined frame rule with standard separating conjunction is **unsound**

Immediately type-inconsistent portion of heap is **disjoint** from almost type-consistent summary

```python
def badUpdate(s: Str, o: Obj | r2 s)
    this.sel = s
    this.call()
    this.obj = o
```

After framing out, entire heap is almost type-consistent

Analysis might **unsoundly** switch to back to type checking
Type-intertwined frame rule with standard separating conjunction is **unsound**

Immediately type-inconsistent portion of heap is **disjoint** from almost type-consistent summary

```
def badUpdate(s: Str, o: Obj | r2 s)
    this.sel = s
    this.call() ✗
    this.obj = o
```

After framing out, entire heap is almost type-consistent

Points to type-inconsistent memory

Analysis might **unsoundly** switch to back to type checking
So what if there’s no type-intertwined framing?
So what if there’s no type-intertwined framing?

class Callback
    var sel: Str
    var obj: Obj | r2 sel

def call()
    this.obj.[this.sel]()
So what if there’s no type-intertwined framing?

class Callback
    var sel: Str
    var obj: Obj | r2 sel

    def call()
        this.obj.[this.sel]()

var o = ...object with a method m...
var cb = new Callback("m", o)
cb.call()
var Callback = Class({
    __init__: function(s,o){...},
    call: function(){
        return this.obj[this.sel].apply(this.obj)
    },
})

var o = ... object with a method m ...
var cb = New(Callback,“m”,o)
cb.call()
var Callback = Class({
  __init__: function(s,o){...
,
  call: function(){
    return this.obj[this.sel].apply(this.obj)
  },
})

var o = ... object with a method m ...
var cb = New(Callback,"m",o)
cb.call()
var Callback = Class({
    __init__: function(s,o){...
    ,
    call: function(){
        return this.obj[this.sel].apply(this.obj)
    },
})

var o = ... object with a method m ...
var cb = New(Callback,“m”,o)
    cb.call()
```javascript
var Callback = Class({
  __init__: function(s,o){…},
  call: function(){
    return this.obj[this.sel].apply(this.obj)
  },
})

var o = ... object with a method m ...
var cb = New(Callback,"m",o)
cb.call()
```

Want to type check this call "like before" — i.e., using the same type system
var Callback = Class({
    __init__: function(s,o){...},
    call: function(){
        return this.obj[this.sel].apply(this.obj)
    },
})

var o = ... object
var cb = New cb.call()
Gated separation expresses a dis-pointing relation between foregate and aftgate.
Gated separation expresses a dis-pointing relation between foregate and aftgate.

\[ M_{\text{fore}} \prec M_{\text{aft}} \]
Gated separation expresses a dis-pointing relation between foregate and aftgate

\[ M_{\text{fore}} \not\preceq M_{\text{aft}} \]

Foregate cannot directly point into aftgate

Figure 1: Memory Formulas and Concretization

Figure 2: Gate Partitioning

Figure 3: Aftgate Shrinking

Figure 4: Neutral and Absorbing Elements

Figure 5: Gate Partitioning

Figure 6: Aftgate Shrinking

Figure 7: Neutral and Absorbing Elements
Gated separation expresses a dis-pointing relation between foregate and aftgate.
**Gated separation** expresses a **dis-pointing** relation between foregate and aftgate.

![Diagram showing the relationship between foregate and aftgate]

- **Foregate cannot directly point into aftgate.**
- **Aftgate may be indirectly reachable from foregate.**
- **But aftgate can point into foregate.**

\[ M_{\text{fore}} \preceq M_{\text{aft}} \]
Gated separation expresses a dis-pointing relation between foregate and aftgate.

Aftgate may be indirectly reachable from foregate.

Foregate cannot directly point into aftgate.

But aftgate can point into foregate.

Gated separation is a non-commutative strengthening of standard separating conjunction restricting the contents of the foregate.
Callback in JavaScript

```javascript
var Callback = Class({
  _init__: function(s,o){...},
  call: function(){
    return this.obj[this.sel].apply(this.obj)
  },
});

var o = ... object with a method m ...
var cb = New(Callback,"m",o)
  cb.call()
```
Callback in JavaScript

```javascript
var o = ... object with a method m ...
var cb = New(Callback,"m",o)
cb.call()
```
var o = ... object with a method m ...

var cb = New(Callback, "m", o)

cb.call()
var o = … object with a method m …
var cb = **New**(Callback,"m",o)
**cb.call**()
var o = ... object with a method m ...

var cb = `New(Callback, "m", o)`

`cb.call()`
Callback in JavaScript

```javascript
var o = ...; // object with method
var cb = New Callback("m", o);

"Non-typeable heap" is gate-separated

cb.call()
```
Strong enough to ensure type-intertwined frame rule is sound

```javascript
var o = ... object with a method m ...
var cb = New(Callback,"m",o)
cb.call()
```
Strong enough to ensure type-intertwined frame rule is sound

```javascript
var o = ... object with a method m ...
var cb = New(Callback, "m", o)
    cb.call()
```
Strong enough to ensure type-intertwined frame rule is sound

```javascript
var o = ... object with a method m ...
var cb = New(Callback,"m",o)
cb.call()
```

Heap is now type-consistent, so switch to type checking
Strong enough to ensure type-intertwined frame rule is sound.

```javascript
var o = ... object with a method m ...
var cb = New(Callback,"m",o)
```

Heap is now type-consistent, so switch to type checking.

Framed-out memory cannot be touched during type checking because of gating.
Challenge: Static analysis with gated separation
Challenge: **Static analysis** with gated separation

\[ x.f = y \]
Challenge: Static analysis with gated separation

\[ x.f = y \]
Challenge: **Static analysis** with gated separation

\[ x.f = y \]
Challenge: **Static analysis** with gated separation

\[ \hat{x} \cdot f \longrightarrow \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \-quarters
Challenge: **Static analysis** with gated separation

\[ \hat{x} \cdot f \rightarrow \hat{y} \]

\[ y \leftarrow \hat{y} \]

Cell to write in a foregate

\[ x.f = y \]

Value to be written
Challenge: **Static analysis** with gated separation

\[
\hat{x} \cdot f \rightarrow \hat{y}
\]

Cell to write in a foregate

\[
x \cdot f = y
\]

Value to be written

Sound? Not if \( \hat{y} \) may be the address of a cell in the concretization of the aftgate
Challenge: Static analysis with gated separation

\[ \hat{x} \cdot f \leftrightarrow \hat{y} \]

Sound? Not if \( \hat{y} \) may be the address of a cell in the concretization of the aftgate.

Transfer functions for \textit{writes} require rearrangement and weakening of gated separation.
How to type check a program that is almost well-typed?
How to type check a program that is almost well-typed almost?
Type-Intertwined Separation Logic
Tolerating temporary violations with almost type-consistent heaps

Coughlin and Chang. POPL 2014.
Type-Intertwined Separation Logic

ok  Tolerating temporary violations with almost type-consistent heaps

Coughlin and Chang. POPL 2014.

When the type invariant is temporarily broken
Type-Intertwined Separation Logic

ok

Tolerating temporary violations with almost type-consistent heaps

Coughlin and Chang. POPL 2014.

When the type invariant is temporarily broken

⚠️

Type-intertwined framing with gated separation

Under preparation.
Type-Intertwined Separation Logic

ok  Tolerating temporary violations with almost type-consistent heaps

Coughlin and Chang. POPL 2014.

When the type invariant is temporarily broken

Type-intertwined framing with gated separation

Under preparation.

When the type invariant applies to only part of the heap