A case for:

Sound Gradual Typing

Ben Greenman @ Northeastern University
Goal: make gradual typing fast
Goal: make gradual typing fast
(or at least usable)
"The end-product appears to be a 50% performance hybrid due to boundary contracts"

"So far Typed Quad is running about 10x slower than regular"

"From 1 ms to 12 seconds ... I feel like I got a bit burned here"
If sound then slow
If sound then slow

Why bother?
Unsound Types are Dangerous!

# untyped

def addVotes(n):
    assert!(n >= 0)
    this.numVotes += n
Unsound Types are Dangerous!

# untyped

```python
def addVotes(n):
    assert!(n >= 0)
    this.numVotes += n
```

# typed, but unsound

```python
def addVotes(n : Natural):
    assert!(n >= 0)
    this.numVotes += n
```
Unsound Types are Dangerous!

# untyped

def addVotes(n):
    assert!(n >= 0)
    this.numVotes += n

# typed, but unsound

def addVotes(n : Natural):
    assert!(n >= 0)
    this.numVotes += n
Unsound Types are Dangerous!

# untyped

def addVotes(n):
    assert!(n >= 0)
    this.numVotes += n

# typed, but unsound

def addVotes(n : Natural):
    assert!(n >= 0)
    this.numVotes += n
Unsound Types are Dangerous!

# untyped

def addVotes(n):
    assert!(n >= 0)
    this.numVotes += n

# typed, but unsound

def addVotes(n : Natural):
    assert!(n >= 0)
    this.numVotes += n

addVotes(-1)
Unsound Types are Dangerous!
Unsound Types are Dangerous!

```python
def getWords(str : String):
    return str.split()
```
Unsound Types are Dangerous!

def getWords(str : String):
    return str.split()

class Atom:
    def split():
        ....
Unsound Types are Dangerous!

def getWords(str : String):  
    return str.split()

class Atom:  
    def split():  
        ....

uranium = Atom()  
getWords(uranium)
Soundness Strategy

τ → assert proxy
Soundness Strategy

τ \rightarrow \text{proxy}

List(Int) \rightarrow \text{for } n \text{ in list: assert(int? n)}
<table>
<thead>
<tr>
<th>Project name</th>
<th>Modules</th>
<th>Untyped LOC</th>
<th>Module structure</th>
<th>Mean Overhead</th>
<th>Max Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>snake</td>
<td>8</td>
<td>161</td>
<td><img src="image1.png" alt="Module Structure" /></td>
<td>32x</td>
<td>121x</td>
</tr>
<tr>
<td>tetris</td>
<td>9</td>
<td>305</td>
<td><img src="image2.png" alt="Module Structure" /></td>
<td>33x</td>
<td>117x</td>
</tr>
<tr>
<td>synth</td>
<td>10</td>
<td>837</td>
<td><img src="image3.png" alt="Module Structure" /></td>
<td>40x</td>
<td>86x</td>
</tr>
<tr>
<td>gregor</td>
<td>13</td>
<td>996</td>
<td><img src="image4.png" alt="Module Structure" /></td>
<td>3x</td>
<td>5x</td>
</tr>
<tr>
<td>quad</td>
<td>16</td>
<td>6722</td>
<td><img src="image5.png" alt="Module Structure" /></td>
<td>31x</td>
<td>56x</td>
</tr>
</tbody>
</table>
snake (8 modules)
typed/untyped ratio 0.92x
max. overhead 121.51x
mean overhead 32.30x
3-deliverable 4 (2%)
3/10-usable 28 (11%)
snake (8 modules)

typed/untyped ratio 0.92x
max. overhead 121.51x
mean overhead 32.30x
3-deliverable 4 (2%)
3/10-usable 28 (11%)

\[ L = 1 \]  

\[ L = 2 \]
<table>
<thead>
<tr>
<th>Module</th>
<th>Typed/Untyped Ratio</th>
<th>Max. Overhead</th>
<th>Mean Overhead</th>
<th>3-Deliverable</th>
<th>3/10-Usable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tetris</strong></td>
<td>(9 modules)</td>
<td>0.97x</td>
<td>117.28x</td>
<td>33.34x</td>
<td>128 (25%)</td>
</tr>
<tr>
<td><strong>synth</strong></td>
<td>(10 modules)</td>
<td>1.03x</td>
<td>85.90x</td>
<td>39.69x</td>
<td>15 (1%)</td>
</tr>
<tr>
<td><strong>gregor</strong></td>
<td>(13 modules)</td>
<td>1.22x</td>
<td>4.72x</td>
<td>2.72x</td>
<td>5644 (69%)</td>
</tr>
<tr>
<td><strong>quad</strong></td>
<td>(16 modules)</td>
<td>13.34x</td>
<td>56.43x</td>
<td>31.50x</td>
<td>2046 (3%)</td>
</tr>
</tbody>
</table>