Software Transactional Memory on PyPy

Pseudo-Goal

- “Kill the GIL”
- GIL = Global Interpreter Lock

Real Goals

- Multi-core programming
- But *reasonable* multi-core programming
- Using the recent model of Transactional Memory

PyPy-STM

- An executable `pypy-stm` which uses internally Software Transactional Memory
- Optimistically run multiple threads in parallel
- The only new feature is `atomic`:

```
with atomic:
    piece of code...
```

Example of higher-level API

```
def work(...):
    ...
    several more calls to:
        transaction.add(work, ...)
    ...
```

- Starts N threads, scheduling *work()* calls to them
- Each *work()* is done in an `atomic` block
- Multi-core, but as if all the *work()* are done sequentially

Status

- Kind of working without the JIT
- Roughly three times slower (you need four cores to see benefits)
- Working on the JIT support

Q&A

- Thank you!
- Budget of $10k left, likely more needed too