# STM in Python

Leo Franchi

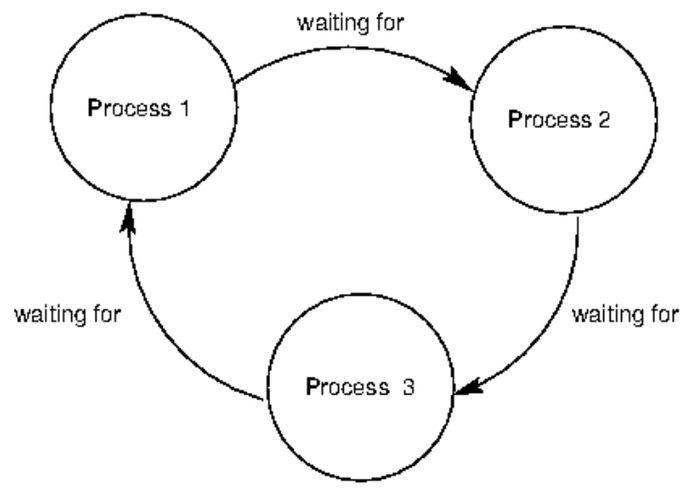
Dec. 17th 2012

# Outline

- Difficulties of multithreaded programming
- What is STM?
- How does STM work?
- Why can't I have a free lunch (except Eatsy)?

# Why is shared state hard?

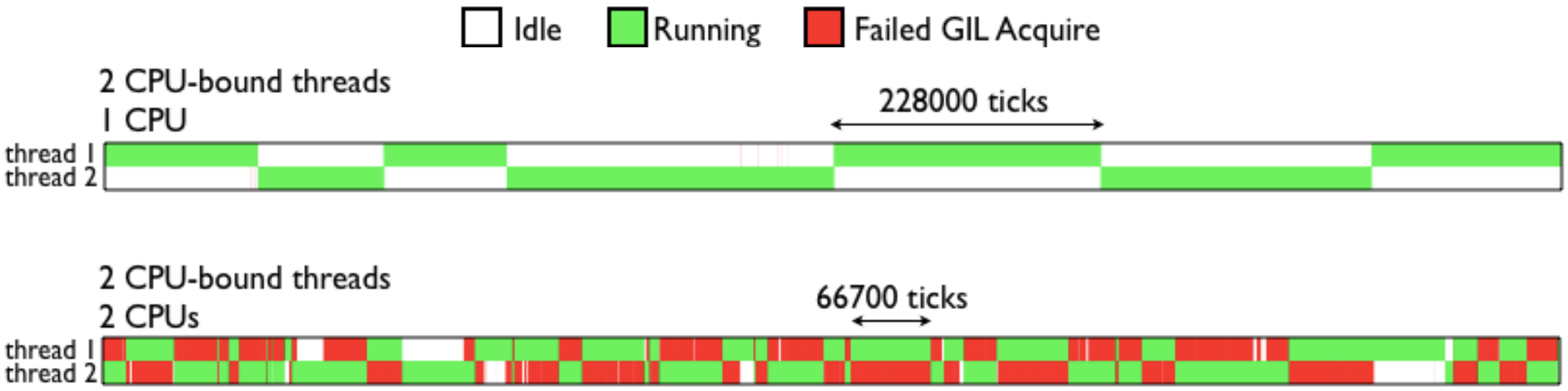
Locking == Mutexes
Deadlocks
Livelocks
Data races




Tuesday, December 18, 12

Composability of locks
Acquiring order of mutexes
Two threads, each waiting on each other to finish something, so neither are able to make any progress either by waiting or spinning.
The worst problem you can run into---two threads concurrently running (without order guarantees) that manipulate or read the same data. Indeterminate result (in fact, in c++11 standard this means your program execution is undefined---anything could happen).

# Global Interpreter Lock




http://dabeaz.blogspot.com/2010/01/python-gil-visualized.html



Data races are less prevalent in python due to the GIL. CPython requires a global lock to execute python code, so all your python code (threads included) is happening in a linear order. By default something like 100 bytecode instructions between context switches---so you might still have data races!. Makes it harder to run into multithreading issues in Python code---but still possible, and still valid for other languages with true multithreading (also jython).

# Software Transactional Memory

- Removes locks from user code
- Removes need to think about serializing access to shared state
- Generic software technique, open research topic



* transactions
Software Transactional Memory is an approach to allow the user to ignore race conditions and threaded conflicts. It pushes into the language level a way for user code to be multithreaded-safe.

# STM in Clojure

```
(def r1 (ref 0))      ;; r1, r2 are refs that point to integers
(def r2 (ref 100))
(alter r1 inc)        ;; error! not in a transaction!
(dosync
    (alter r1 inc)    ;; inc is a function, increments argument
    (alter r2 dec))   ;; dec does the opposite


                      ;; @r1 is now 1, r2 is now 99
```

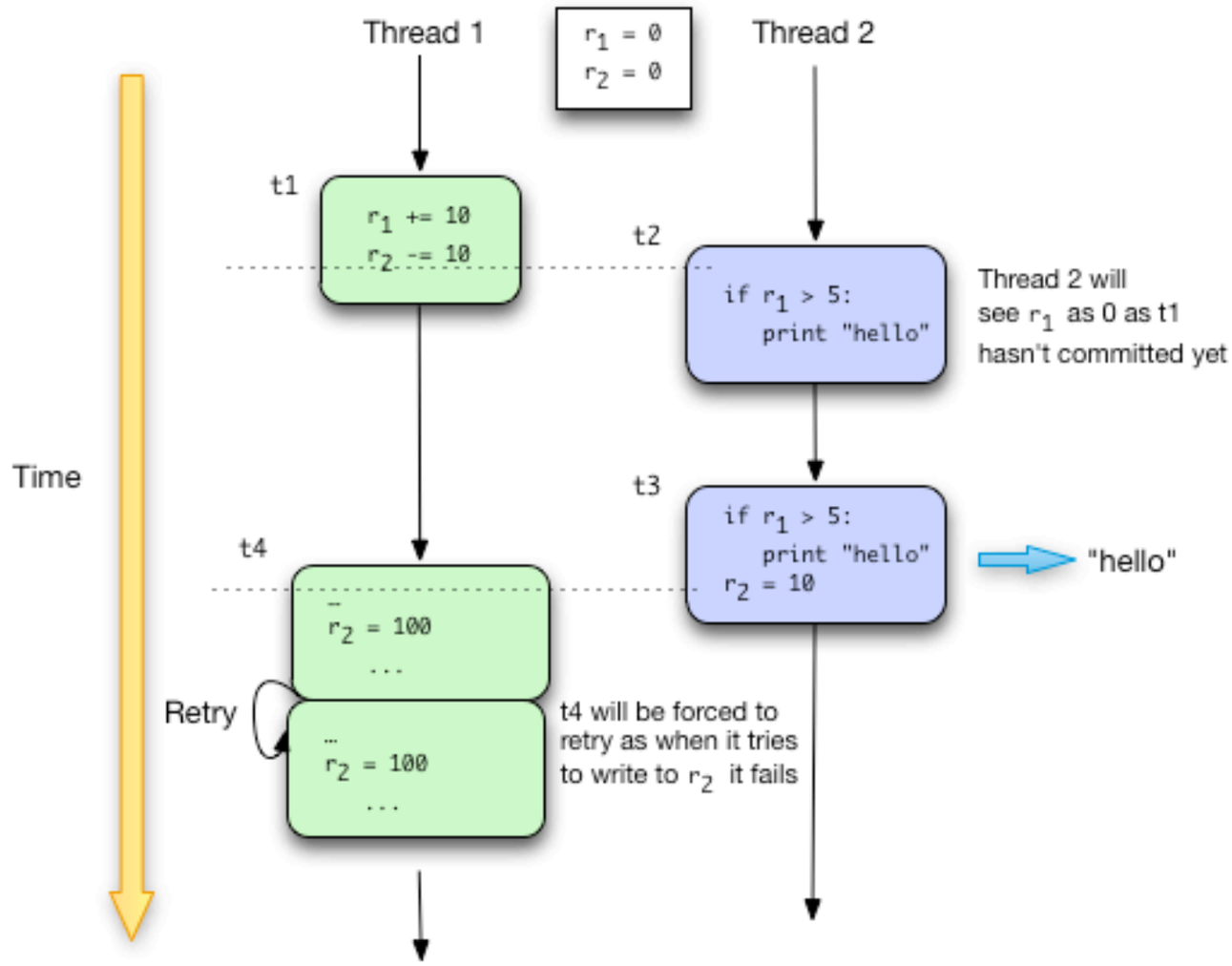
data is immutable

# Transactions

- atomic
- consistent world view
- automatic retries

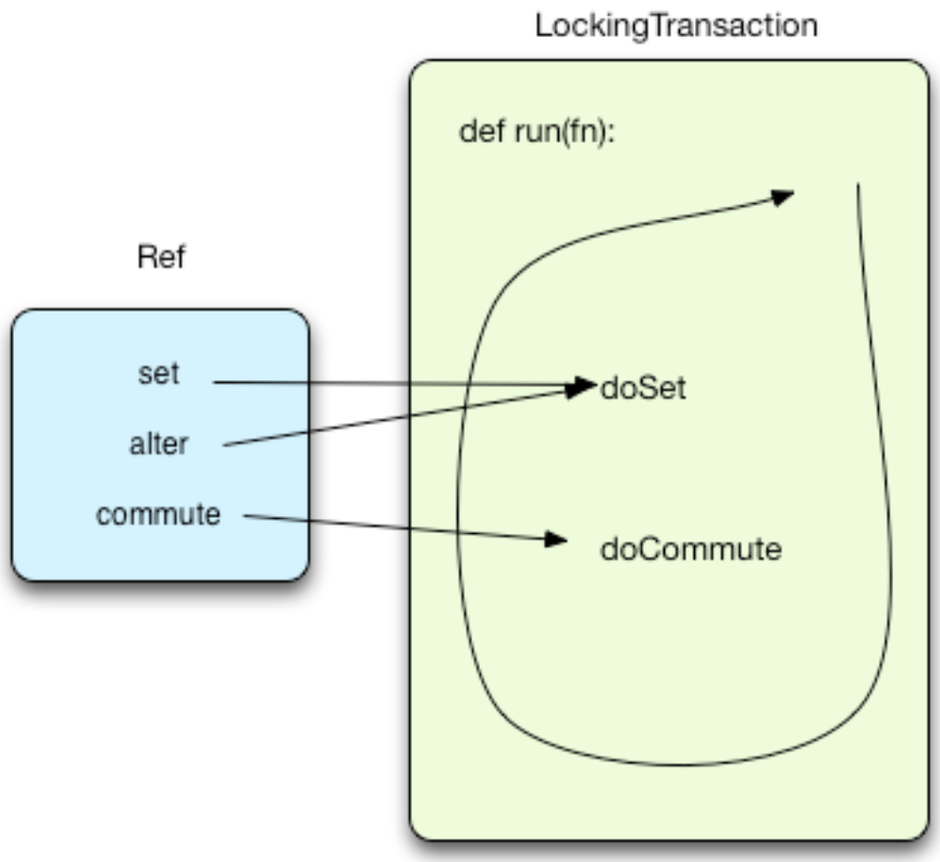


question: can anyone tell me if they have an immediate gut feeling about potential problems with this?




Tuesday, December 18, 12

Don't have to think about locking. Just write multithreaded code that modifies refs in a transaction -> just works.
Talk about the ants.clj example.


LockingTransaction
def run(fn):
Ref
set
alter
commute
doSet
doCommute

# (almost) code time!

- reads are cheap and do not block
- writes go through a transaction



MVCC
make the easy cast fast: reads (optimized for few writes)

code

Question: Should the unit tests be in clojure or in python?

# Problems

- STM is not a magic bullet
- Tradeoffs are considerable
- Tuning might be needed, leaky abstraction!

- history chain length
- ensure
- commute
- faults

Why is immutable data is required for this STM system to work?