PERCONA

Software Transactional Memories

Fernando Ipar

RubyConfUY 2013

$> whoami

job:

Consultant @ Percona

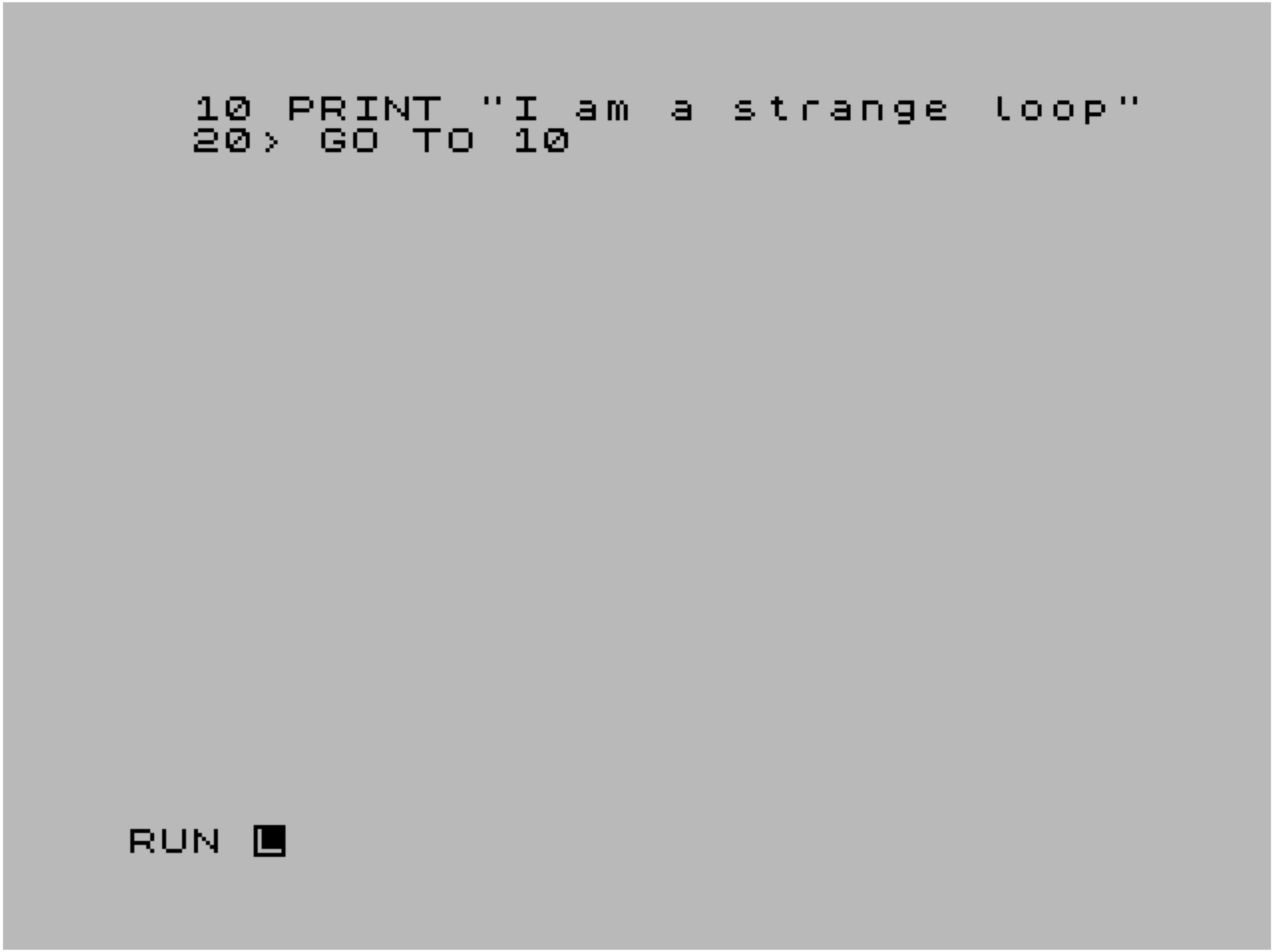
Contact me:

fernando.ipar@percona.com

fipar on irc.freenode.net

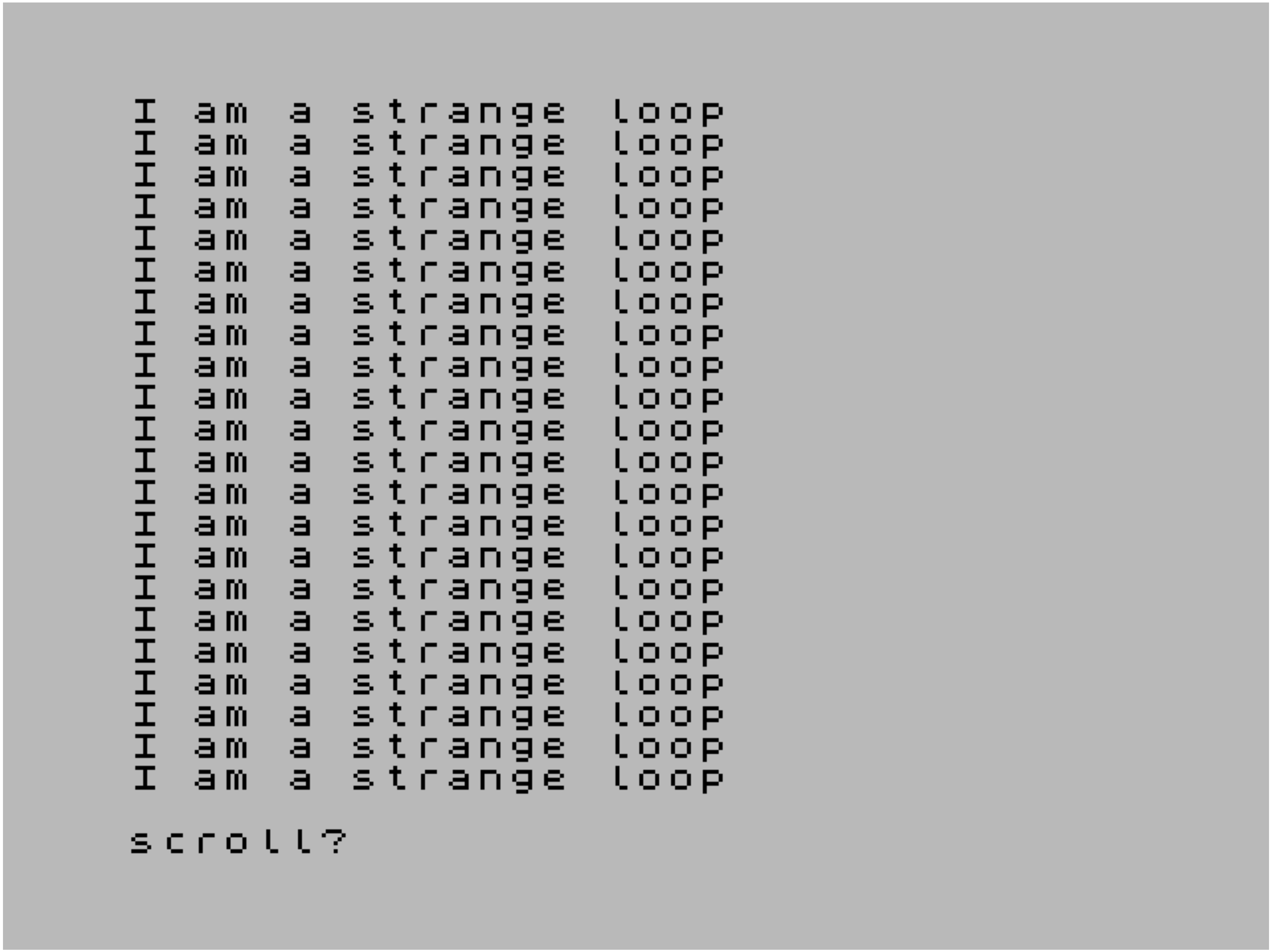
This talk:

A little history



```
  10 PRINT "I am a strange loop"
  20> GO TO 10
```

```
RUN
```



```
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop
I am a strange loop

scroll?
```

Z80A @ 3.58 MHZ
16K RAM

Z80A @ 3.58 MHZ
16K RAM

Single Processor.

Brief history of scaling

```
10 Wait for the next processor upgrade
20 GO TO 10
```

```
ScalingError: 2005 arrived
    from (irb):3:in `/'
    from (irb):3
    from /Users/fernandoipar/.rvm/rubies/ruby-1.9.3-
    p362/bin/irb:16:in `<main>'
```

http://bit.ly/FreeLunchOver

Crux of the matter

Single threaded:

- one variable? one value at one moment in time

Multi threaded:

- one variable? who knows?!

http://en.wikipedia.org/wiki/File:Master_Padlock.jpg

It all boils down to locks

- It’s test-and-set all the way

But we have different abstractions

- synchronized functions/methods
- message passing
- software transaction memories

Transactions

Atomicity
Consistency
Isolation
Durability

Normally only in the context of transactional databases

JRuby options for this:

- Deuce STM
- Multiverse
- Clojure

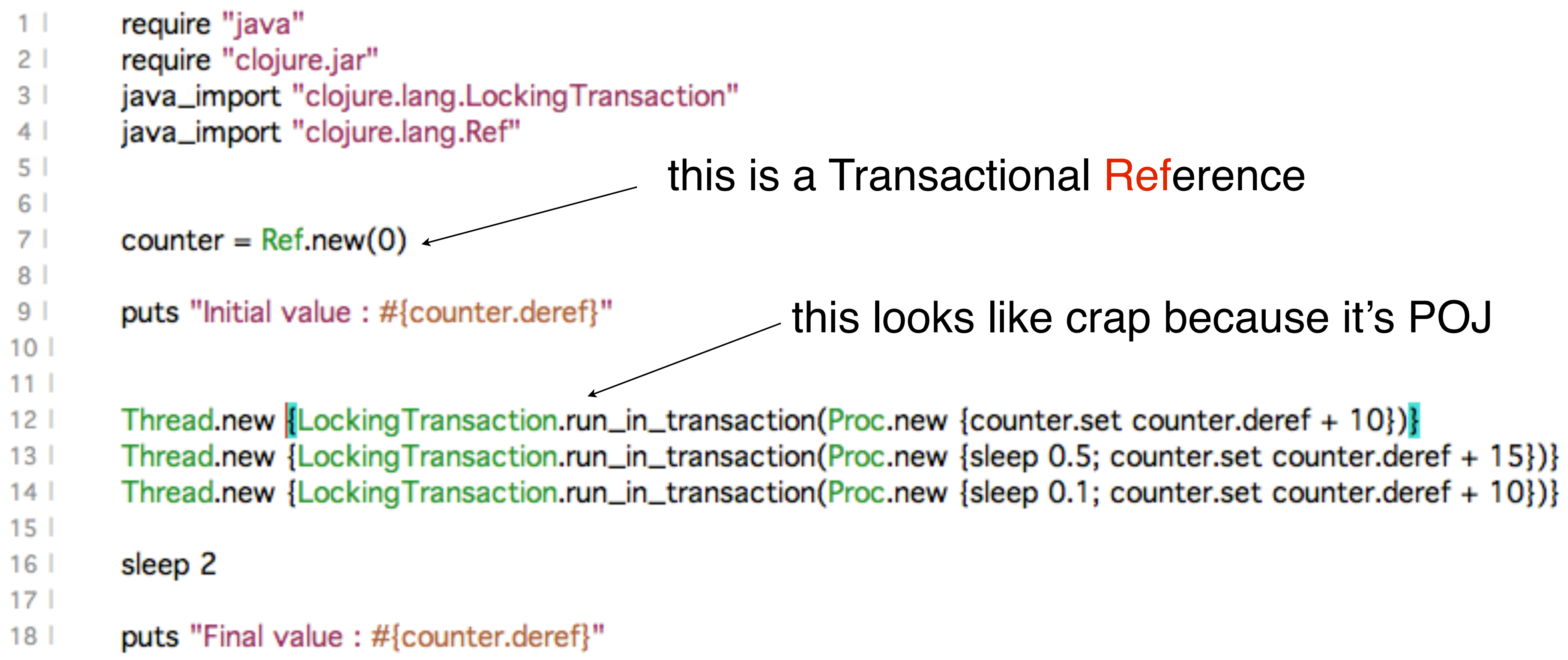
Clojure’s STM

Core part of the language

Optimistic concurrency control

Implements MVCC

https://gist.github.com/fipar/5206269



```
require "java"
require "clojure.jar"
java_import "clojure.lang.LockingTransaction"
java_import "clojure.lang.Ref"


counter = Ref.new(0)

puts "Initial value : #{counter.deref}"


Thread.new {LockingTransaction.run_in_transaction(Proc.new {counter.set counter.deref + 10})}
Thread.new {LockingTransaction.run_in_transaction(Proc.new {sleep 0.5; counter.set counter.deref + 15})}
Thread.new {LockingTransaction.run_in_transaction(Proc.new {sleep 0.1; counter.set counter.deref + 10})}

sleep 2

puts "Final value : #{counter.deref}"
```

this is a Transactional Reference

this looks like crap because it’s POJ

A quick look under the hood

clojure.Agent

- Atomic reads
- Async writes
- In a txn, writes are deferred until commit

clojure.Ref

- Atomic reads
- Writes only within a txn
- Multiple versions
  - Uncommited (maintained by txn)
  - Commited (maintained by Ref)

The take home message

Concurrency is ubiquitous now

- and not going anywhere

It doesn’t have to be a pain

- abstractions are not for free

JRuby ==

- your fav. language +
- ∞ libs

(println “thank you”)