

Concurrencia y Python

lucio.torre@gmail.com

<http://www.python.org/ar>

disclaimer

concurrancia

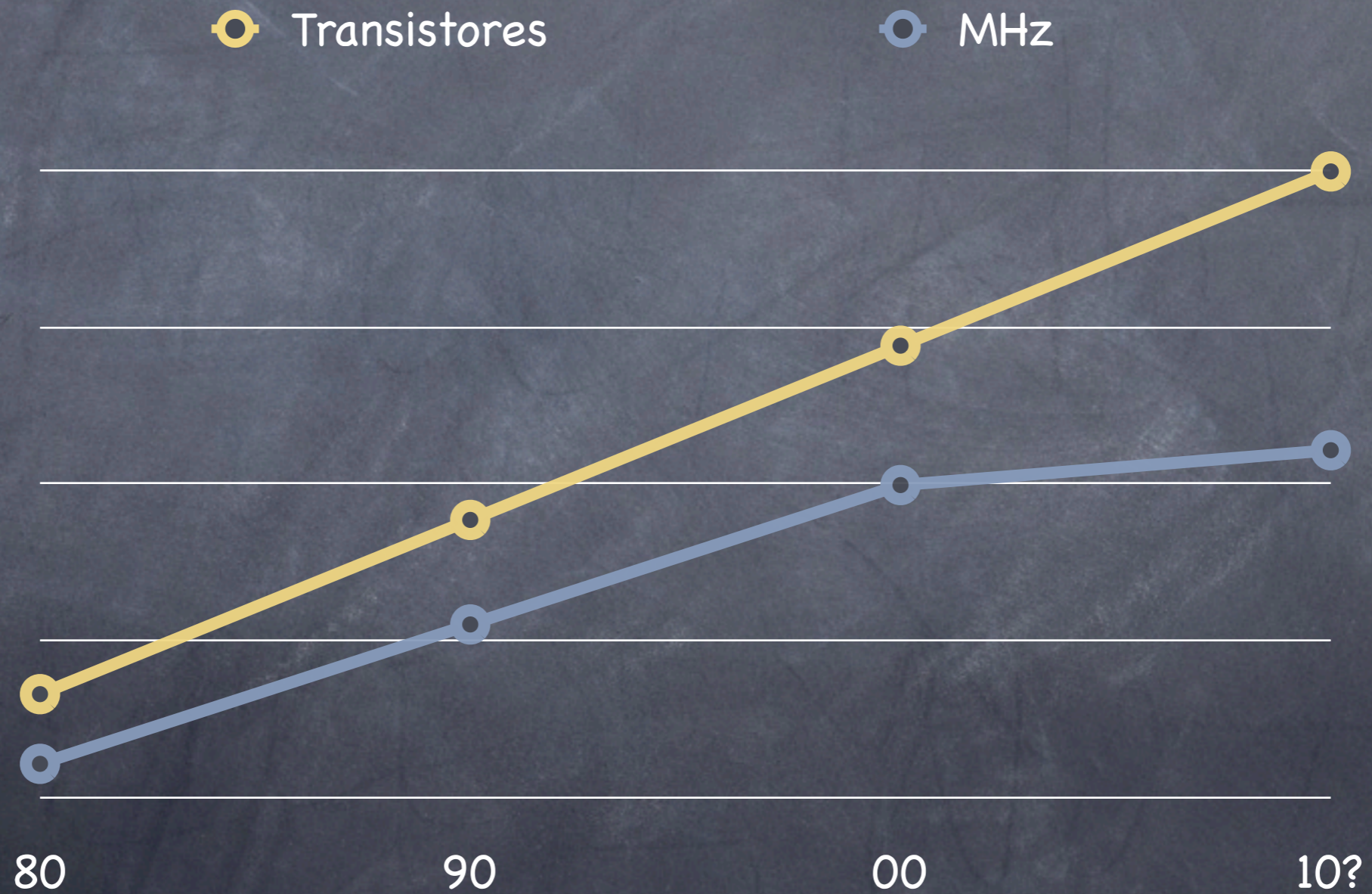
Coincidencia, concurso simultáneo de varias circunstancias.

porque concurrencia

el mundo es concurrente

```
def familyLunch():  
    husband.eat(lunch)  
    wife.eat(lunch)  
    son.eat(lunch)  
    daughter.eat(lunch)
```

se termino el almuerzo gratis



la concurrencia es difícil

caso de estudio

```
def send_mail(rcpt, body):  
    c = connect()  
    send_data(c, rcpt, body)  
    c.close()
```

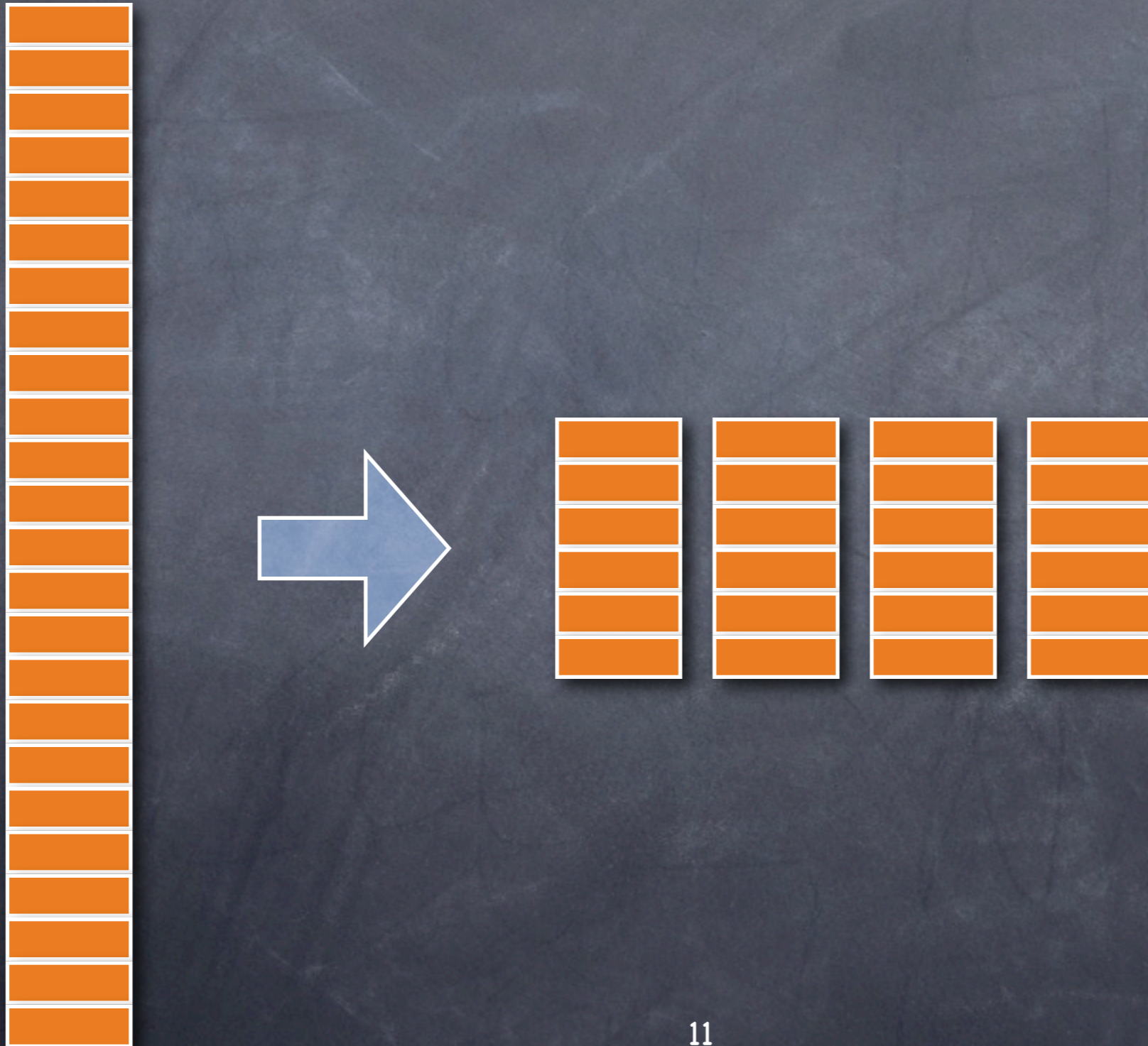
enviar 100 mails

```
enviados = 0
for i in range(100):
    send_mail(rcpt, body%i)
    enviados += 1
```

problemas

| |
|----------------|
| secuencialidad |
| |
| |
| |
| |
| |
| |
| |

partición/descomposición



solucion 1: particion por datos con threads

```
enviados = 0
```

```
def enviar(rcpt, body):
```

```
    global enviados
```

```
    send_mail(rcpt, body)
```

```
    enviados += 1
```

```
for i in range(num_mails):
```

```
    start_new_thread(enviar, (rcpt, body%i))
```

problemas

threads

~~secuencialidad~~

enviados+=1

| Thread 1 | Stack 1 | enviados | Stack 2 | Thread 2 |
|----------------|---------|----------|---------|----------------|
| LOAD_GLOBAL 0 | [1] | 1 | [] | |
| LOAD_CONST 1 | [1,1] | 1 | [] | |
| INPLACE_ADD | [2] | 1 | [] | |
| | | | [1] | LOAD_GLOBAL 0 |
| | | | [1,1] | LOAD_CONST 1 |
| STORE_GLOBAL 0 | [] | 2 | [1,1] | |
| | [] | 2 | [2] | INPLACE_ADD |
| | [] | 2 | [] | STORE_GLOBAL 0 |

problema: estado compartido

... we did not (and still do not) believe in the standard multithreading model, which is preemptive concurrency with shared memory: we still think that no one can write correct programs in a language where 'a=a+1' is not deterministic.

problemas

threads

~~secuencialidad~~

estado compartido

locks

```
enviados = 0
```

```
enviados_lock = threading.Lock()
```

```
def enviar(rcpt, body):  
    global enviados  
    send_mail(rcpt, body)  
    enviados_lock.acquire()  
    enviados += 1  
    enviados_lock.release()
```

problemas

threads

~~secuencialidad~~

locks

~~estado compartido~~

problemas con locks

- deadlock
- composicionalidad
- baja concurrencia vs complejidad
- difícil de debuggear

“Just Say No to the combined evils of locking, deadlocks, lock granularity, livelocks, nondeterminism and race conditions.” - GvR

“A computer is a state machine. Threads are for people who can’t program state machines.” — Alan Cox

problemas

threads

~~secuencialidad~~

~~locks~~

estado compartido

problema: escalabilidad

enviar 10000 mails

Traceback (most recent call last):

File "<stdin>", line 3, in ?

thread.error: can't start new thread

linux 2.6.15,python2.4.3, 1gb mem: 382 threads

darwin8.9.0, python2.4.4, 1gb mem: 7027 threads

winxp, python2.5,512mb: 1030 threads

problemas

threads

~~secuencialidad~~

estado compartido

cantidad de threads

Python GIL

ceval.c:

```
/* Interpreter main loop */
PyObject *PyEval_EvalFrameEx(PyFrameObject *f, int throwflag) {
    for (;;) {
        [...]

        /* Do periodic things. Doing this every time through the loop
        would add too much overhead, so we do it only every Nth
        instruction. [...] */
        if (--_Py_Ticker < 0) {
            _Py_Ticker = _Py_CheckInterval;
            if (interpreter_lock) {
                /* Give another thread a chance */
                PyThread_release_lock(interpreter_lock);
                /* Other threads may run now */
                PyThread_acquire_lock(interpreter_lock, 1);
            }
        }
    }
}
```

problemas

~~threads~~

secuencialidad

estado compartido

cantidad de threads

nota: partición por tareas

- cpu bound vs io bound:
 - la red es io bound
- cantidad de estado compartido
- particion anidada

solución 2: multithreading cooperativo (twisted!)

el principio de Hollywood:
no nos llames, nosotros te llamamos.

ejemplo

```
class Mailer:
    def start(self):
        d = connect(host)
        d.addCallback(self.send_data)
    def send_data(self, c):
        d = c.write(data)
        d.addCallback(self.close)
    def close(self, c):
        c.close()
        enviados += 1
```

problemas

secuencialidad

~~estado compartido~~

~~cantidad de threads~~

eventos

problemas: inversión de control / stack ripping

class Mailer:

```
def __init__(self, rcpt, body):
```

```
    self.rcpt = rcpt
```

```
    self.body = body
```

```
    d = connect(host)
```

```
    d.addCallback(self.send_data)
```

```
def send_data(self, c):
```

```
    d = send_data(c, self.rcpt, self.body)
```

```
    d.addCallback(self.close)
```

```
def close(self, c):
```

```
    c.close()
```

```
    enviados += 1
```

```
def send_mail(rcpt, body):
```

```
    c = connect()
```

```
    send_data(c, rcpt, body)
```

```
    c.close()
```

problemas

secuencialidad

estado ~~compartido~~

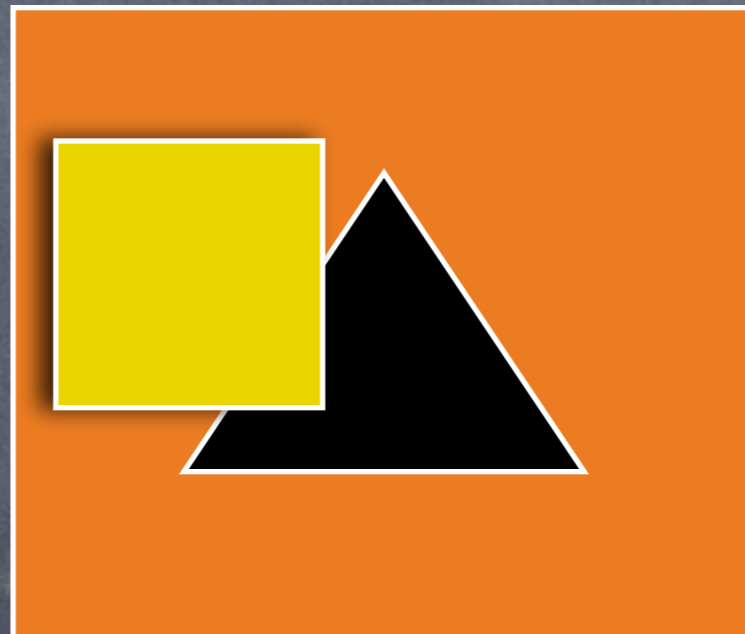
cantidad ~~de~~ threads

inversión de control



eventos

problemas: integración



problemas

secuencialidad

~~estado compartido~~

~~cantidad de threads~~

inversión de control

integración

eventos

solución 3: usando corutinas

```
class Mailer:
    def __init__(self, rcpt, body):
        self.rcpt = rcpt
        self.body = body
        d = connect(host)
        d.addCallback(self.send_data)
    def send_data(self, c):
        d = send_data(c, self.rcpt, self.body)
        d.addCallback(self.close)
    def close(self, c):
        c.close()
        enviados += 1

@defer.deferredGenerator
def send_mail(rcpt, body):
    c = yield connect()
    yield send_data(c, rcpt, body)
    yield c.close()
```

problemas

secuencialidad

~~estado compartido~~

~~cantidad de threads~~

~~inversión de control~~

integración

composicionalidad

eventos

generadores

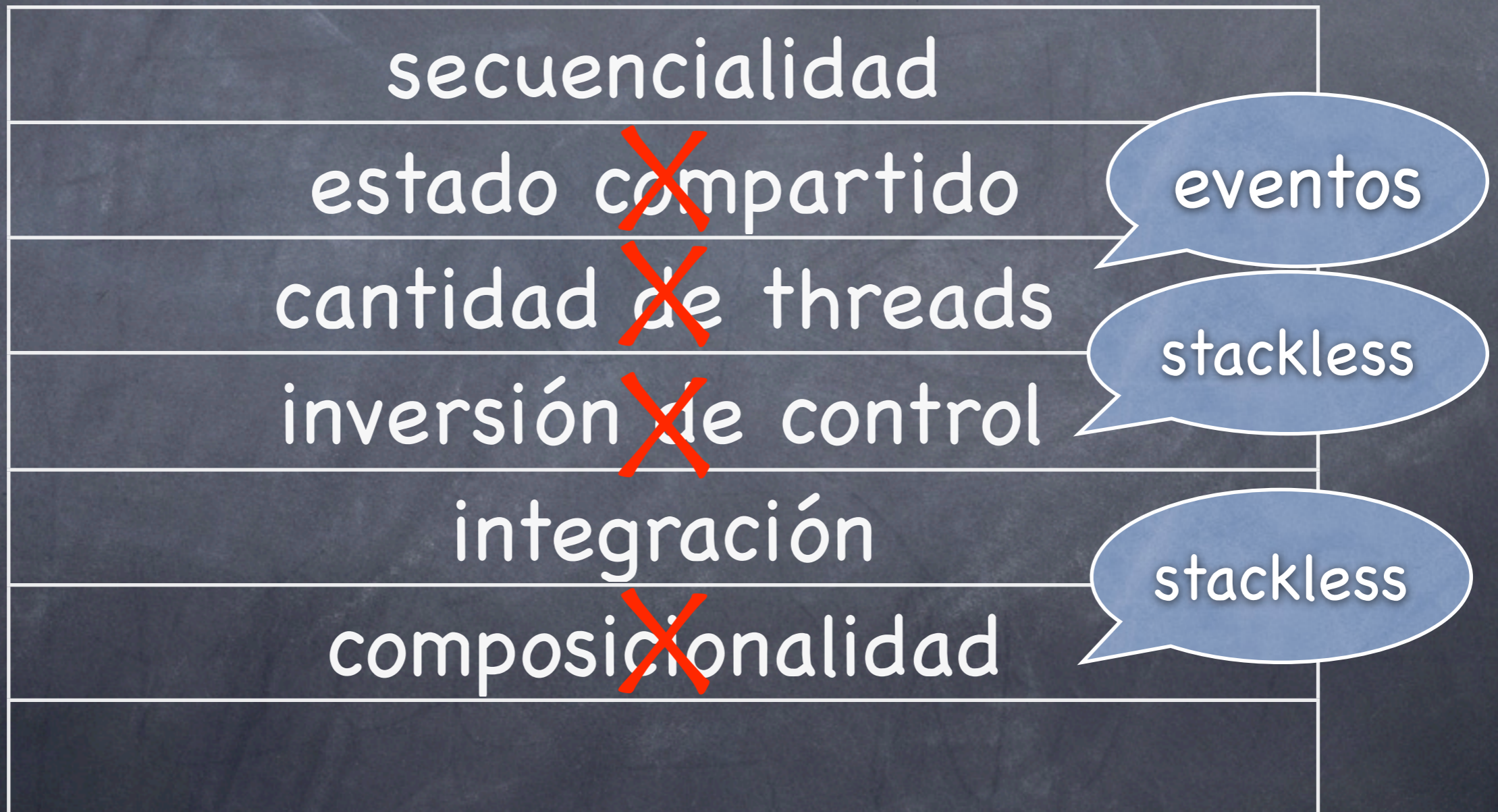
solución 4: stackless

```
import stackless
import stacklesssocket as socket
enviados = 0

def enviar(rcpt, body):
    global enviados
    send_mail(rcpt, body)
    enviados += 1

for i in range(num_mails):
    stackless.tasklet(send_mail)(rcpt, body%i)
stackless.run()
```

problemas



escalabilidad: procesos

“Threads are not the only way to do that. Processes do the job nearly as well with a drop of the complexity. And that's exactly how Rails is scaling to use all the cores you can throw at it.” -- DHH

from processing import Process

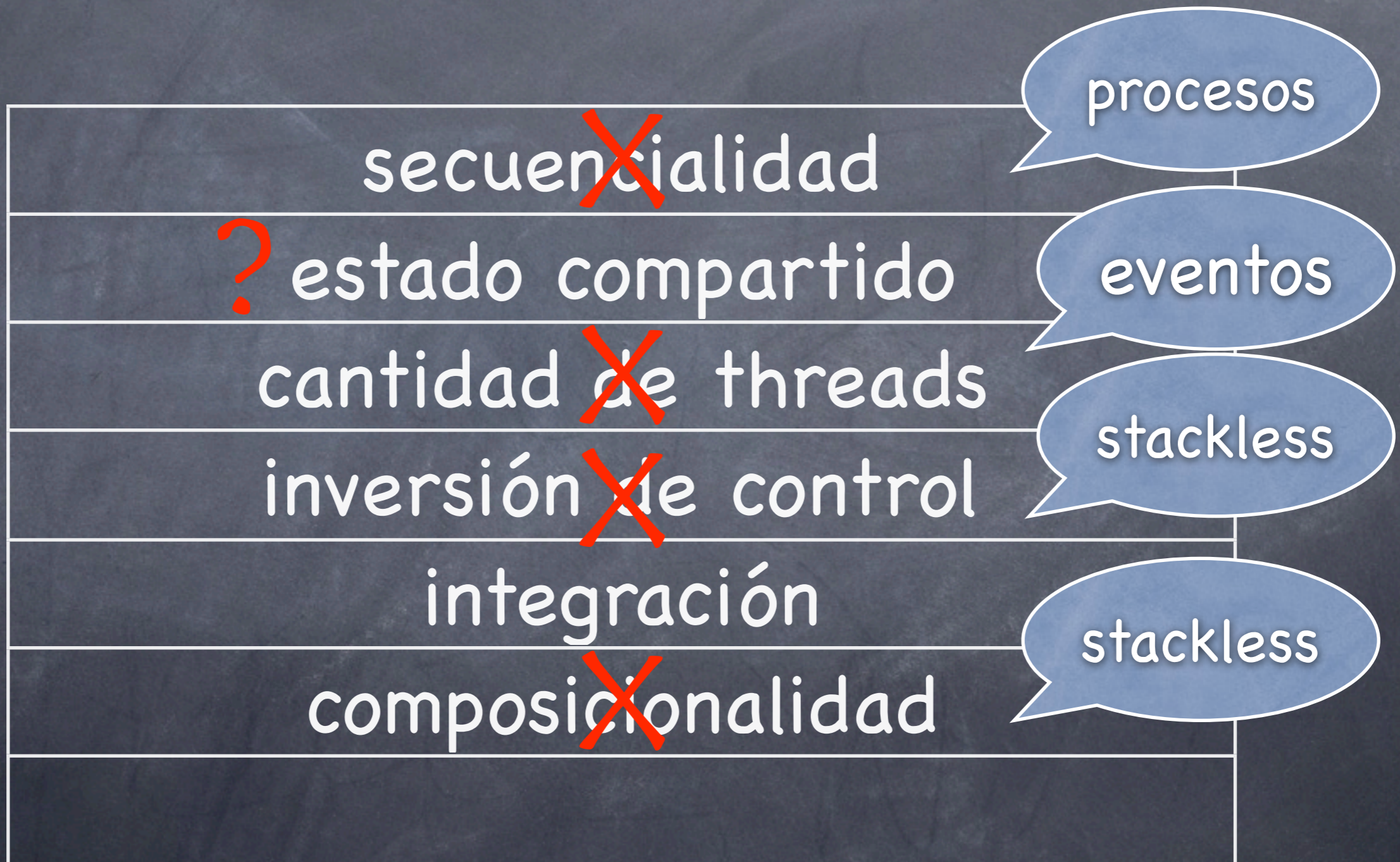
```
processes = [Process(target=send_mail, args=num_mails/  
tasks)
```

```
    for i in range(TASKS)]
```

```
for p in processes:  
    p.start()
```

```
for p in processes:  
    p.join()
```

problemas

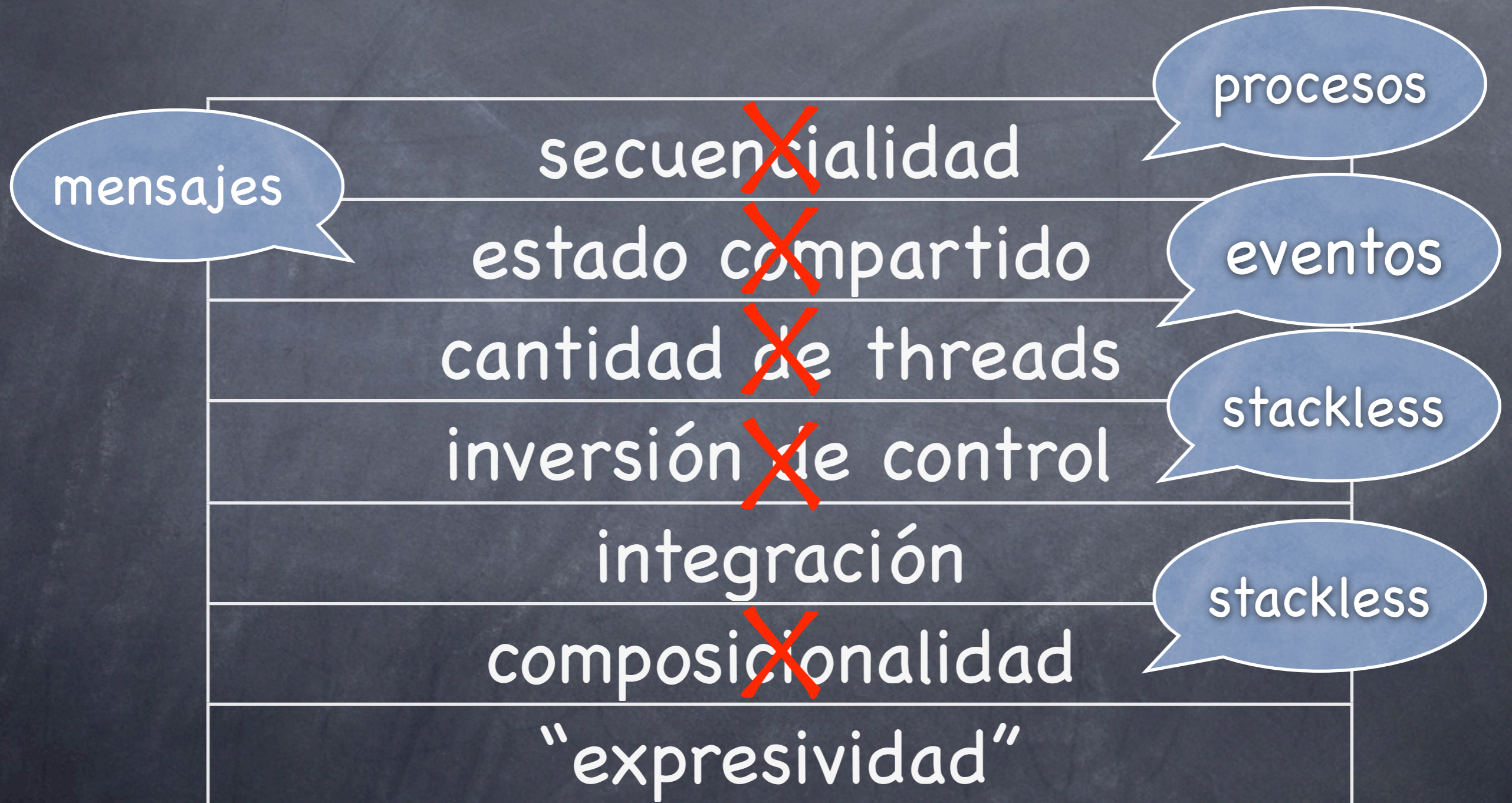


estado compartido: message passing

*Erlang['s message passing] is to
locks what Java is to pointers.*

- Colas
 - `threading.Queue`
 - `processing.manager.Queue`

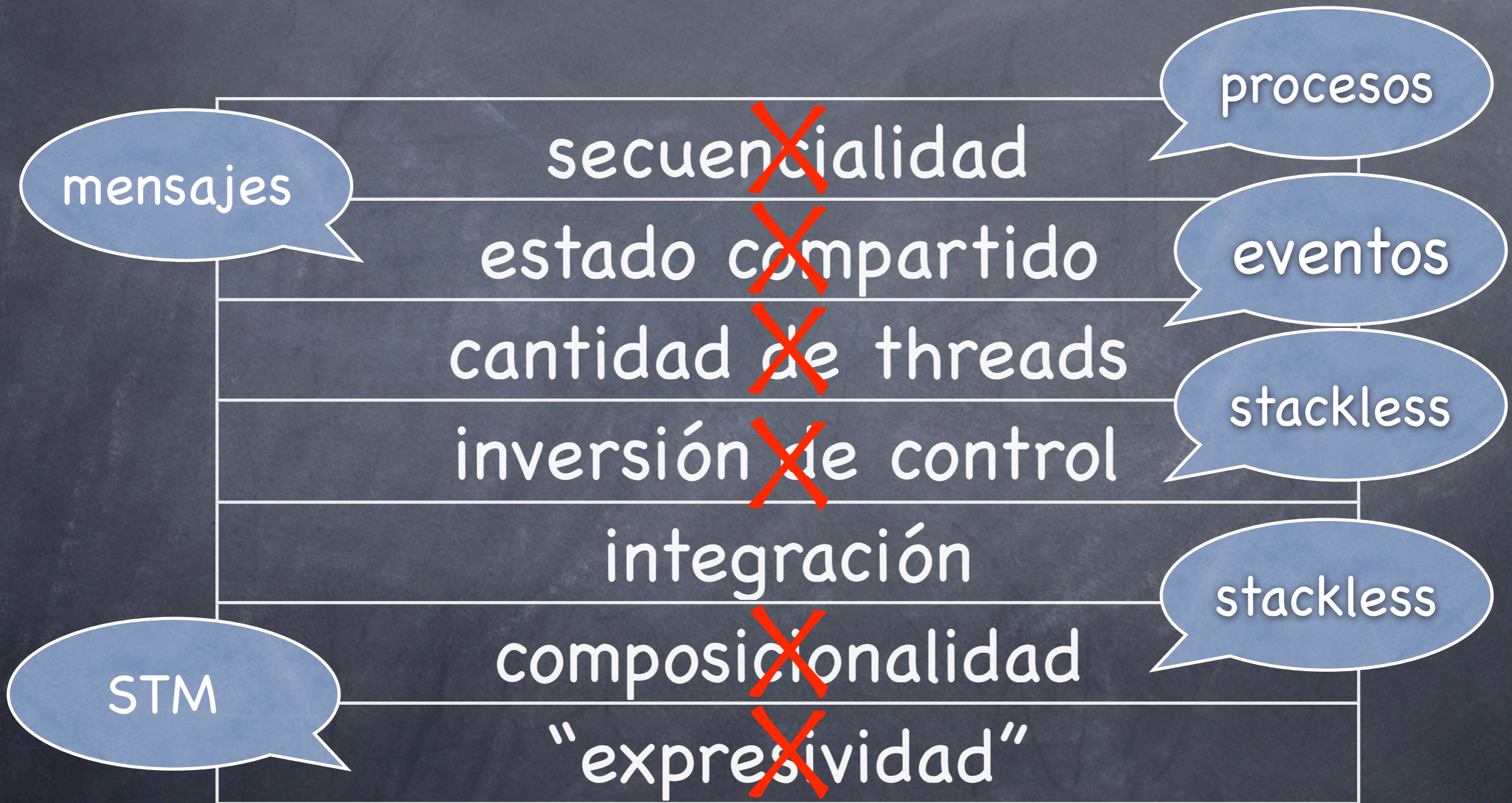
problemas



locks: Software Transactional Memory

- se puede componer
- muy similar a bases de datos
- zodb + zeo
- durus

problemas



conclusiones

- threads y el estado compartido mutable son el default equivocado
- para que sea escalable y paralelizable el código debe haber sido realizado con eso en mente

*“Threads are Evil. Processes are Ugly. State
Machines send you Mad. Samba4: Choose
your own combination of Evil, Ugly and
Mad” -- Andrew Tridgell*

“If my hunch is right, I expect that instead of writing massively parallel applications, we will continue to write single-threaded applications that are tied together at the process level rather than at the thread level.... I expect that most problems (even most problems that we will be programming 10-20 years from now) get little benefit out of MP.”

-- GvR

“I'm thoroughly unconvinced that Python is doomed unless it adds more thread support.” -- GvR

“In the general case they are layered, with a strict functional inner layer, a deterministic concurrency layer, a message-passing concurrency layer, and a shared-state concurrency layer, in that order. I postulate that this common structure will be part of one possible definitive programming language”
-- P & R

“If you want to utilize all of that unused performance, it’s going to become more of a risk to you and bring pain and suffering to the programming side.”

— John Carmack

fuentes

- <http://www.tecgraf.puc-rio.br/~lhf/ftp/doc/hopl.pdf> : The Evolution of Lua
- <http://www.rae.es> Diccionario de la Real Academia Espaniola
- http://members.verizon.net/olsongt/stackless/why_stackless.html : Introduction to Concurrent Programming with Stackless Python
- <http://www.gotw.ca/publications/concurrency-ddj.htm> : The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software
- <http://jazz.external.hp.com/training/sqltables/c5s14.html> : IMAGE/SQL: Issues and answers concerning SQL tables, 5.14 How does granularity affect locking?
- <http://www.stackless.com/wiki/Tasklets> : Tasklets / Microthread en stackless python
- <http://www.softpanorama.org/People/Ousterhout/Threads/sld001.htm> : Whats wrong with threads.
- http://www.usenix.org/events/hotos03/tech/full_papers/vonbehren/vonbehren_html/index.html : Why Events Are A Bad Idea (for high-concurrency servers)
- <http://wiki.zope.org/ZODB/FrontPage> : ZODB
- <http://www.mems-exchange.org/software/durus/> : durus
- <http://mail.python.org/pipermail/python-3000/2007-May/007414.html> : [Python-3000] the future of the GIL
- <http://cheeseshop.python.org/pypi/processing/0.21> : processing package
- <http://www.info.ucl.ac.be/~pvr/bookcc.html> : Convergence in language design: a case of lightning striking four times in the same place
- <http://www.loudthinking.com/posts/7-multi-core-hysteria-and-the-thread-confusion>: Multi-core hysteria and the thread confusion
- <http://www.algorithm.com.au/talks/concurrency-erlang/>: Concurrency and erlang

preguntas?

lucio.torre@gmail.com

http://www.python.org/ar

pyar-subscribe@decode.com.ar