How Do You Solve a Problem Like Santa Claus ?

Prototyping Join Patterns with stackless.py for Stackless Python

Andrew Francis
af.stackless@gmail.com

Montreal Python 26

December 17th, 2011

Exploring

Feedback

Play

Fearless Programming

Trial-and-error

Learning

Serendipity

Syncretism

Baling wire and Chewing Gum

Innovation

Friendship

Challenges

The Santa Claus Problem

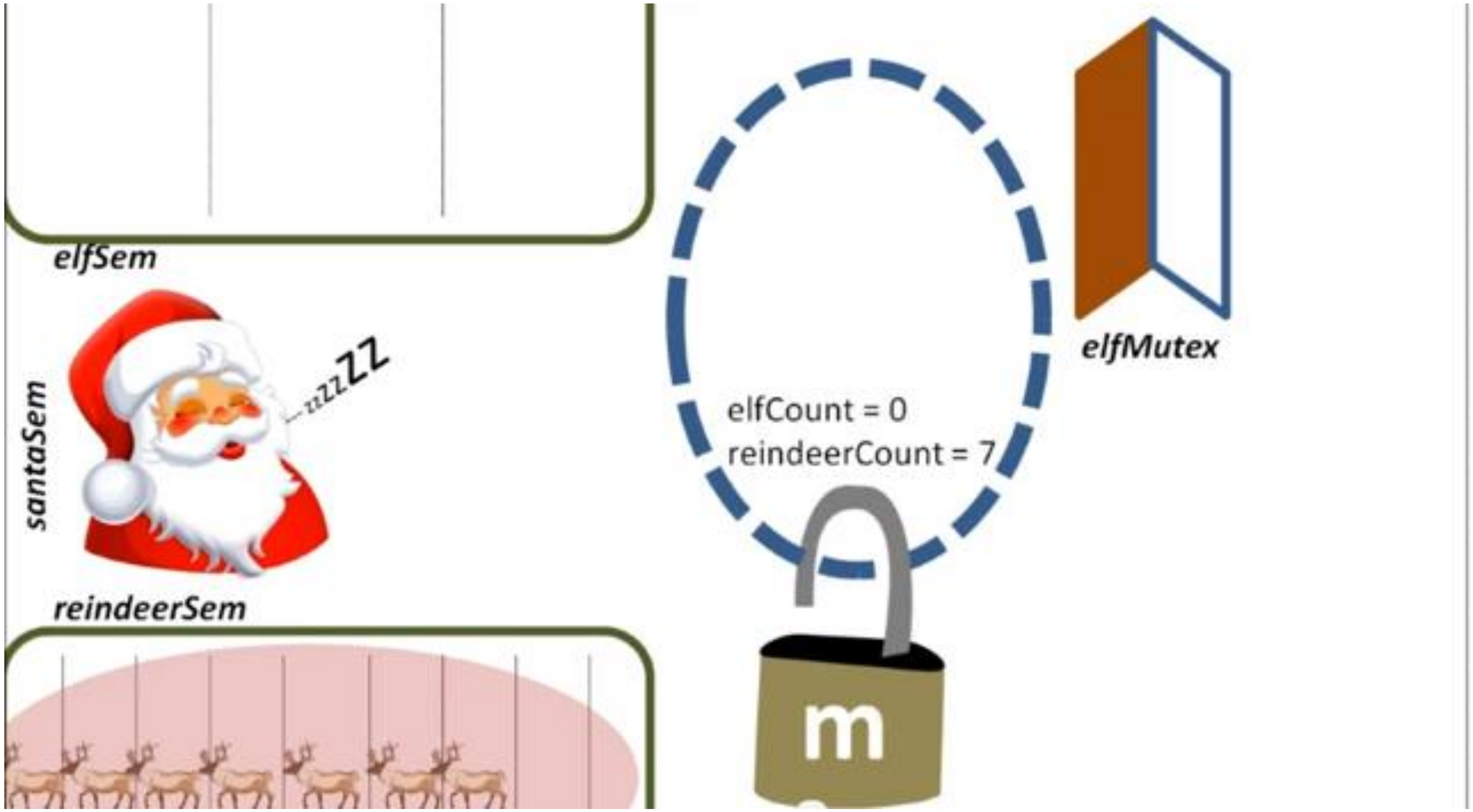
Santa repeatedly sleeps until wakened by either all of his nine rein-deer, back from their holidays, or by a group of three of his ten elves. If awakened by the reindeer, he harnesses each of them to his sleigh, delivers toys with them and finally unharnesses them (allowing them to go off on holiday). If awakened by a group of elves, he shows each of the group into his study, consults with them on toy R&D and finally shows them each out (allowing them to go back to work).

Santa should give priority to the reindeer in the case that there is both a group of elves and a group of reindeer waiting.

Perception

Taken from The Santa Claus Problem: Thread Synchronization

Reality



http://www.youtube.com/watch?v=pqO6tKN2lc4

Solutions Through The Years

Year	Language	Mechanism
1994	C	Semaphores
1997	Ada 95	Protected objects, Rendezvous (select)
2003	Polyphonic C#	Join Patterns
2007	Haskell	Software Transactional Memory

Join Patterns a la Polyphonic C#

The Join Pattern (Chord)

Message Body

```
class Join2 {
    void wait(out int i, out int j)
    & async first(int r1)
    & async second(int r2) {
          i = r1; j = r2; return;
    }
}

//client code
int i,j;
Join2 x = new Join2();
...
//synchronous method will block
x.wait(i,j);
// do something with i and j
```

Chord Rules

- When pending method calls match a pattern, its body runs.
- If there is no match, the invocations are queued up.
- If there are several matches, an unspecified pattern is selected.
- If receiver is synchronous, body executes in receiver’s thread.
- If only asynchronous methods, the body runs in a new thread.

Stackless Python

- Superset of CPython.
  - Inspired by Bell Labs Limbo language
- Reputation for user space threads too cheap to meter.
- Cooperative and pre-emptive multi-tasking.

Example

```
import stackless

def producer(aChannel):
    aChannel.send("hello")

def consumer(aChannel):
    print aChannel.receive()

aChannel = stackless.channel()
stackless.tasklet(producer)(aChannel)
stackless.tasklet(consumer)(aChannel)
stackless.schedule()
```

Synchronous Channels

Before: [producer, …]$_{scheduler}$
(producer, send,“hello”) -> [ ]$_{receive}$

After : [~~producer~~]$_{scheduler}$
[(producer, send, “hello”)]$_{send}$

Before: (consumer, receive, null) -> [(Producer)]$_{send}$
After: (1) consumer.val = “hello”
(2) [~~(producer, send, “hello”)~~~~,…~~]$_{send}$
(3) […, Producer]$_{scheduler}$

The Select Algorithm

```
def select(operations):
    choice = None
    source = getcurrent()
    numberReady = 0

    for operation in operations:
        if operation.ready():
            numberReady += 1
            if nrand(numberReady) == 0:
                choice = operation

    if choice:
       choice.action()
    else:
       for operation in operations:
           operation.add()

       schedule_remove()
       schedule()

       choice = source._operation
       source._operation = None

    return choice
```

Question

Question: Why is knowing about select() important?

Answers

- Stackless Python did not originally implement select().
  - This is proof-of-concept that we can modify stackless.py to create new features
- Select() is used under the hood to implement join patterns
- We are going to extend the select algorithm developed in “Prototyping Go’s Select with stackless.py” to include join patterns.

Christian Tismer’s Sleight of Hand

- Stackless Python does not support select()
- Eliminated a queue
- Created a channel property: balance
  - > 0 : senders on the queue
  - < 0 : receivers on the queue
  - 0: empty

stackless.py

- A PyPy module that implements the Stackless Python API
- Written in Python!
- Low level concurrency provided by different packages
  - greenlets
  - Continulets

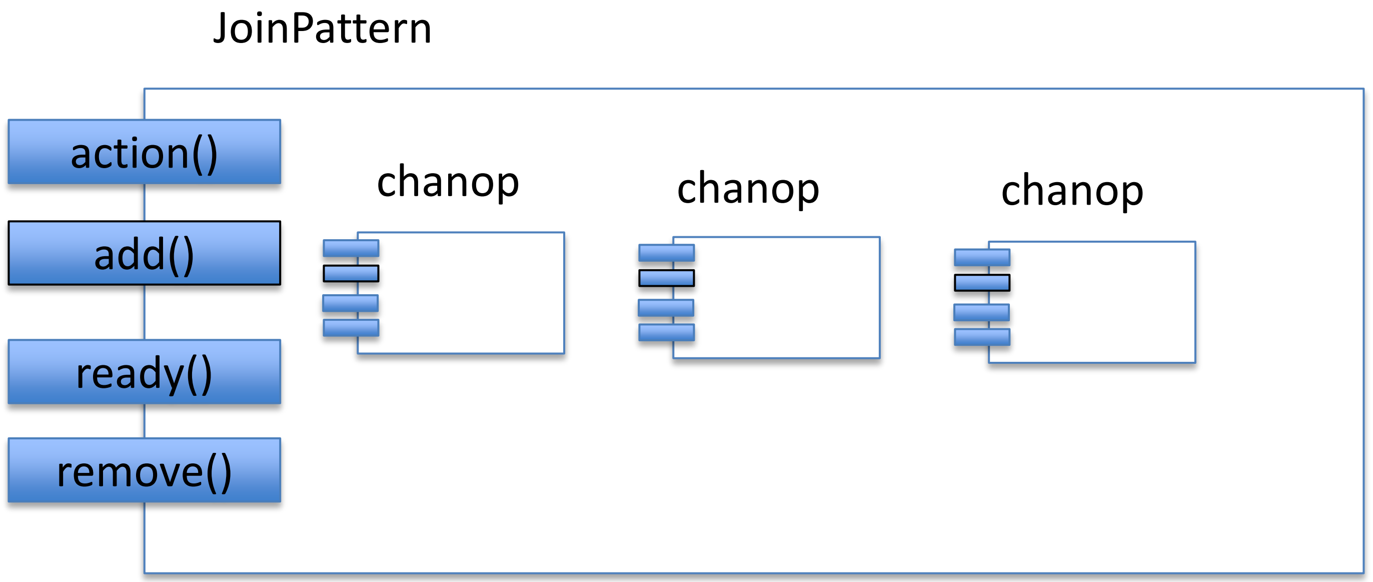
Strategy

- Implemented sub-set of join pattern’s behaviour
  - Synchronous receiver
  - No built-in asynchrony
- object.method = channel.receive()
  - i.e., join2.wait() *equivalent to* join2.receive()
  - Simplifies API

Strategy

- Why implement a method body (and class) for synchronization patterns?
  - we already have channels.
  - Message bodies seem to exist to store internal messages and compute a return value ?
- Rather
  - return all internally queued values associated with pattern.
  - let receiver decide what to do.

Another Sleight of Hand

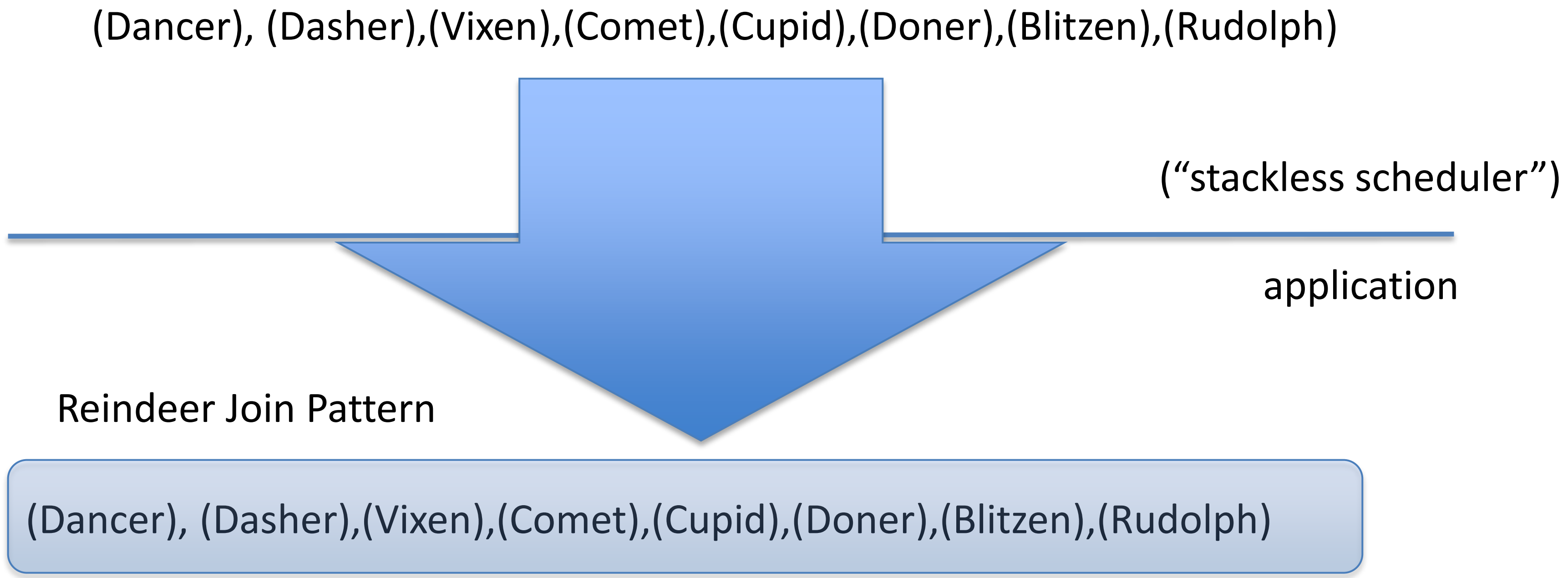


- A join pattern is a composite channel operation
- A join pattern uses the same interface as a channel operation
- Now we can express disjunctions of conjunctions!

Famous Last Words (Le Mot de Cambronne)

States? Lock free bags? Atomicity? OS threads aren’t involved. How difficult can this be to implement?

“All or Nothing” (“Atomicity”)


(Dancer), (Dasher),(Vixen),(Comet),(Cupid),(Doner),(Blitzen),(Rudolph)
(“stackless scheduler”)
application
Reindeer Join Pattern
(Dancer), (Dasher),(Vixen),(Comet),(Cupid),(Doner),(Blitzen),(Rudolph)

Transfer all the data if and only if all nine reindeer channels are ready

New Rules

Postpone Rendezvous (lazy acquire)

Before: (rudolph, S, msg) -> $[(\text{Santa,JR})]_{\text{rudolph-receive}}$
After : $[(\text{rudolph,join-send, msg})]_{\text{rudolph-send}}$

Steal

Before: (Mrs Claus, R, Null) -> $[(\text{rudolph,JS,msg})_{\text{send}}$
After : Mrs Claus.val = msg
$[(\text{Santa,JR})]_{\text{rudolph-receive}}$

Words of Wisdom

Hence *plan to throw one away; you will, anyhow.*

--Fred Brooks (on pilot projects)

```
def santa(reindeer, elves):
    joinObject = stackless.join(). \
       addPattern([ch for _, ch, _ in reindeer]).\
       addPattern([ch for _, ch, _ in elves],3)

    reindeerPattern, elfPattern = joinObject.patterns

    while True:
        pattern = joinObject.join()
        if reindeerPattern.ready():
            reindeerPattern.join()
            pattern = reindeerPattern
        if pattern is reindeerPattern:
            harness(reindeerPattern)
            deliveryToys(reindeerPattern)
            unharness(reindeerPattern)
        elif pattern is elfPattern:
            consultWithSanta(elfPattern)
```

Lessons Learnt

- Asynchrony matters
  - Prototype not powerful enough to handle Dining Philosophers
  - Synchronous channels with buffers.
- Atomicity a powerful feature.
  - In case of Dining Philosophers, just does the right thing
- “Transactional” logic and expressiveness come to the forefront quickly.

Status

- Prototype is still held together by baling wire and chewing gum.
- A lot of work needs to be done before it is prime time

A New Language: Conversation with Scalable Join Pattern's Authors

Concurrent ML

Atomic transactions

Transactional events

Eager and lazy acquire

Composibility

Lock-free data structures

Optimistic locking protocols

Inevitability

The Ghosts of Software Present, Past, and Future

The Present: Concurrent ML:
guard(event, f)

The Past: Chandler Notification Manager 2.0 (R.I.P)*:*
eval("city == MTL and band == Interpol")

The Future: C.E.P & S.0.A with Stackless Python:
Guard(function(chanop.val), concertFeedChannel)
return eval("city=MTL and band==Hannah Georgas")

References

- The Santa Claus Problem: Thread Synchronization, http://www.youtube.com/watch?v=pqO6tKN2lc4
- Scalable Join Patterns, Claudo Russo & Aaron Turon, http://www.ccs.neu.edu/home/turon/scalable-joins.pdf
- Jingle Bells, Solving the Santa Claus Problem in Polyphonic #C, http://research.microsoft.com/en-us/um/people/nick/polyphony/
- The Polyphonic #C Bedroom Poster, http://research.microsoft.com/en-us/um/people/nick/polyphony/polyphoniccsharpposter2002.ppt
- http://swtch.com/usr/local/plan9/src/libthread/channel.c, Russ Cox
- Concurrent Programming in ML.
  John H. Reppy. Cambridge University Press, Cambridge, England, 1999.
- The Notification Manager, http://www.osafoundation.org/archives/2003_11.html
- stackless.py (with select), http://stacklessexamples.googlecode.com/svn/trunk/sandbox/select/stackless.py
- Prototyping Go's Select with stackless.py for Stackless Python, http://andrewfr.files.wordpress.com/2010/07/july31revised.pdf

For More information

http://andrewfr.wordpress.com

Joyeux Noël