Prototyping Go's Select with Stackless.py for Stackless Python

Andrew Francis
af.stackless@gmail.com
http://andrewfr.wordpress.com
July 19th, 2010
EuroPython 2010
Birmingham, UK
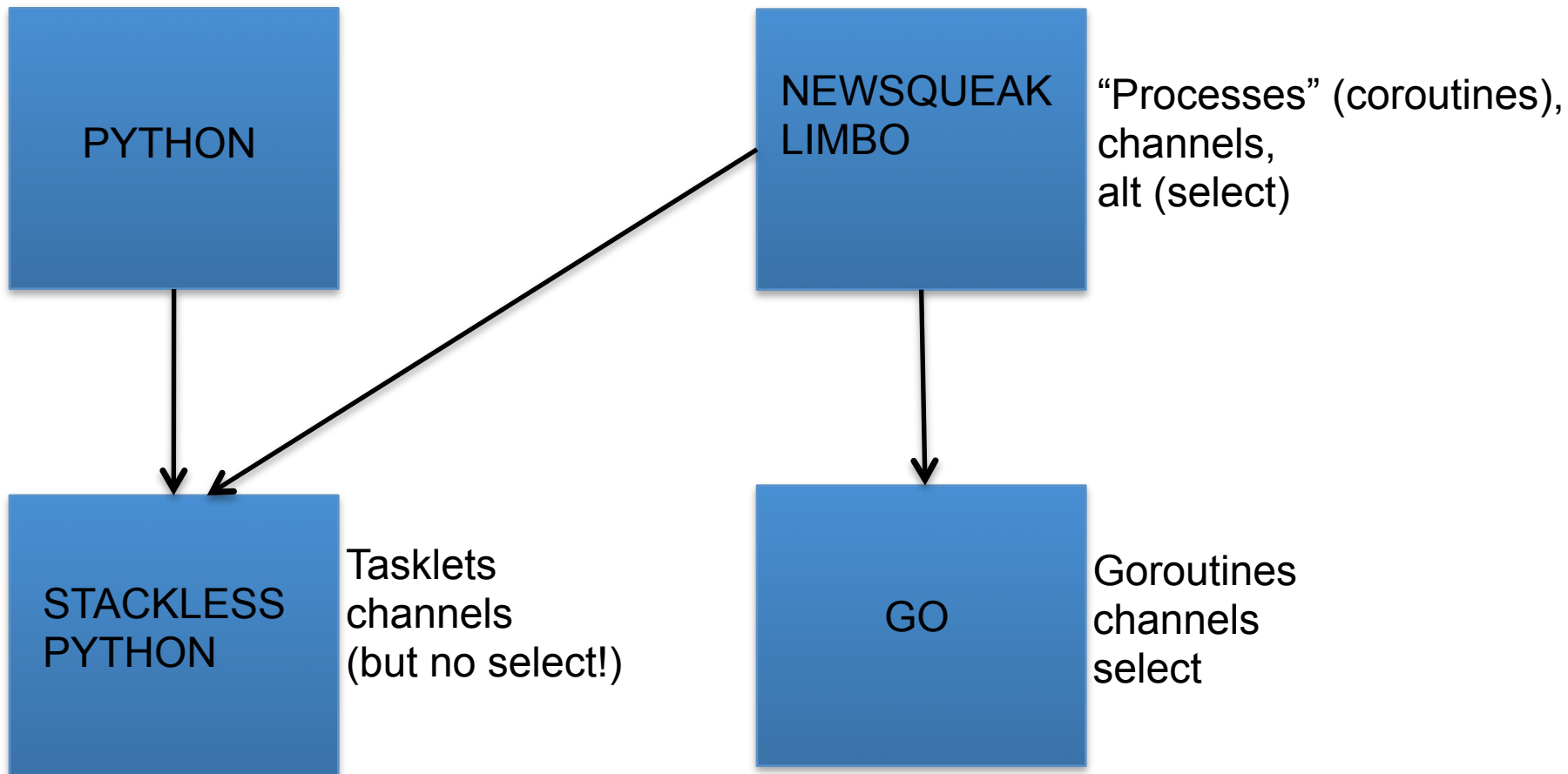
Purpose

- To show how PyPy's stackless.py module can be used to prototype new concurrency features for Stackless Python

  – Stackless Python is a superset of Python renowned for microthreads too cheap to meter.

  – Stackless Python also great for writing new concurrency constructs!

Why Prototype with stackless.py?

- Occasionally there are concurrency constructs that are difficult to correctly implement solely with Stackless Python's classes
  - need finer control over scheduling
  - Need to supplant underlying C data structures

- Prototyping with Stackless Python's C code base a costly way to experiment

Why Go's Select : A Family Tree

PYTHON

NEWSQUEAK
LIMBO

"Processes" (coroutines),
channels,
alt (select)

STACKLESS
PYTHON

Tasklets
channels
(but no select!)

GO

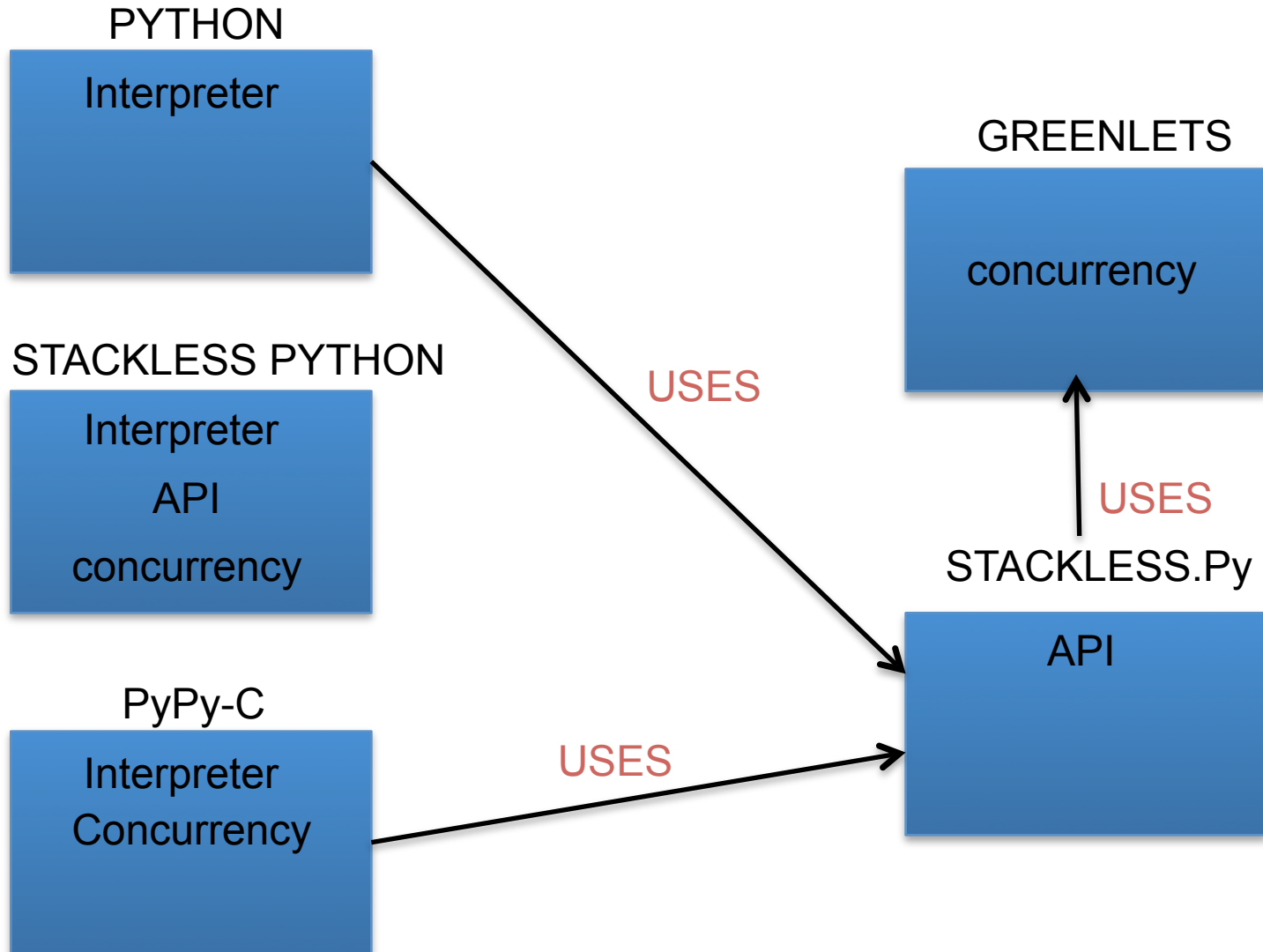Goroutines
channels
select

By The Way

Select allows a coroutine to wait on multiple channels for an action to occur without resorting to polling

(conceptually similar but not the same as UNIX select)
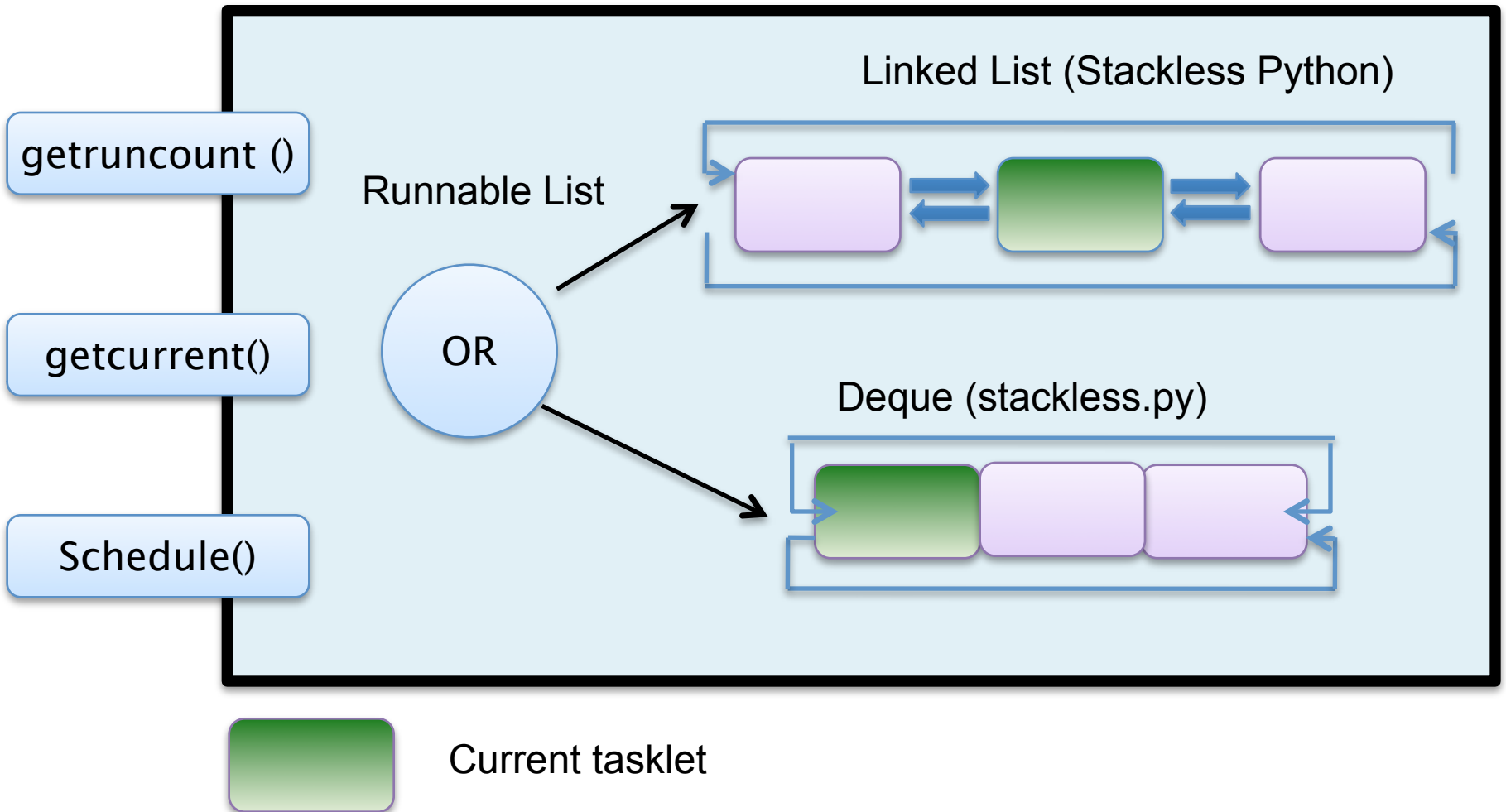
Questions Explored through Prototyping

- What is a suitable interface for a Stackless Python select?

- What would internally change?

- How would pre-existing Stackless Python applications break?

The Cast of Characters

PYTHON

Interpreter

STACKLESS PYTHON

Interpreter

API

concurrency

PyPy-C

Interpreter
Concurrency

GREENLETS

concurrency

STACKLESS.Py

API

USES

USES

USES

Implementation Details

Scheduler

getruncount ()

getcurrent()

Schedule()

Runnable List

OR

Linked List (Stackless Python)

Deque (stackless.py)

Current tasklet

The Approach

- Read Rob Pike's paper "The Implementation of Newsqueak"
  - includes great description of the channel based message passing algorithm
  - An important theme is the opacity of the underlying system's state to the application
  - First prototype was based solely on Pike's description

Approach Continued: Quick

chanelobject.c

stackless.py

Int
PyChannel_Send()

static pyObject
*generic_channel_action()

def send(self, msg)

def channel_action
(self, arg, dir)

Mimics Stackless Python's logical structure. Other variants of
stackless.py don't

Approach Continued
Ask Questions

- Asked questions in Go Lang Nuts and Stackless mailing lists
  - GoLang Nuts: Rob Pike, Russ Cox, Ian Taylor
    - Read libthread.c
  - Stackless: Christian Tismer and Richard Tew
    - Select cannot be done in Stackless without additional tasklets

Prototyping

- Late April to June Sunday sessions
  - partner Kevin Bulušek (Thanks!)
- Two prototypes done by end of April
  - implementation of an eventHandler
    - Got familiar with issues and stackless.py
  - Stackless Python mock-up based on Plan 9's libthread (Kevin)
    - An API for select
    - data structures

An Overview of Stackless Python and Go

Introduction

```
andrew@parker:~/lab/stacklessSelect/comparisons$ python
Python 2.6.5 Stackless 3.1b3 060516 (python-2.65:82030M, Jun 26
2010, 15:49:57)
[GCC 4.3.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import stackless
[32376 refs]
>>> dir(stackless)
['__doc__', '__name__', '__reduce__', '__reduce_ex__', '_gc_track',
'_gc_untrack', '_get_all_objects', '_get_refinfo', '_pickle_moduledict',
'_wrap', 'bomb', 'cframe', 'channel', 'cstack', 'enable_softswitch',
'get_thread_info', 'getcurrent', 'getmain', 'getruncount', 'run',
'schedule', 'schedule_remove', 'select', 'set_channel_callback',
'set_schedule_callback', 'slpmodule', 'stackless', 'tasklet',
'test_cframe', 'test_cframe_nr', 'test_cstate', 'test_outside']
', 'set_ignore_nesting', 'setup', 'tempval', 'thread_id']
```
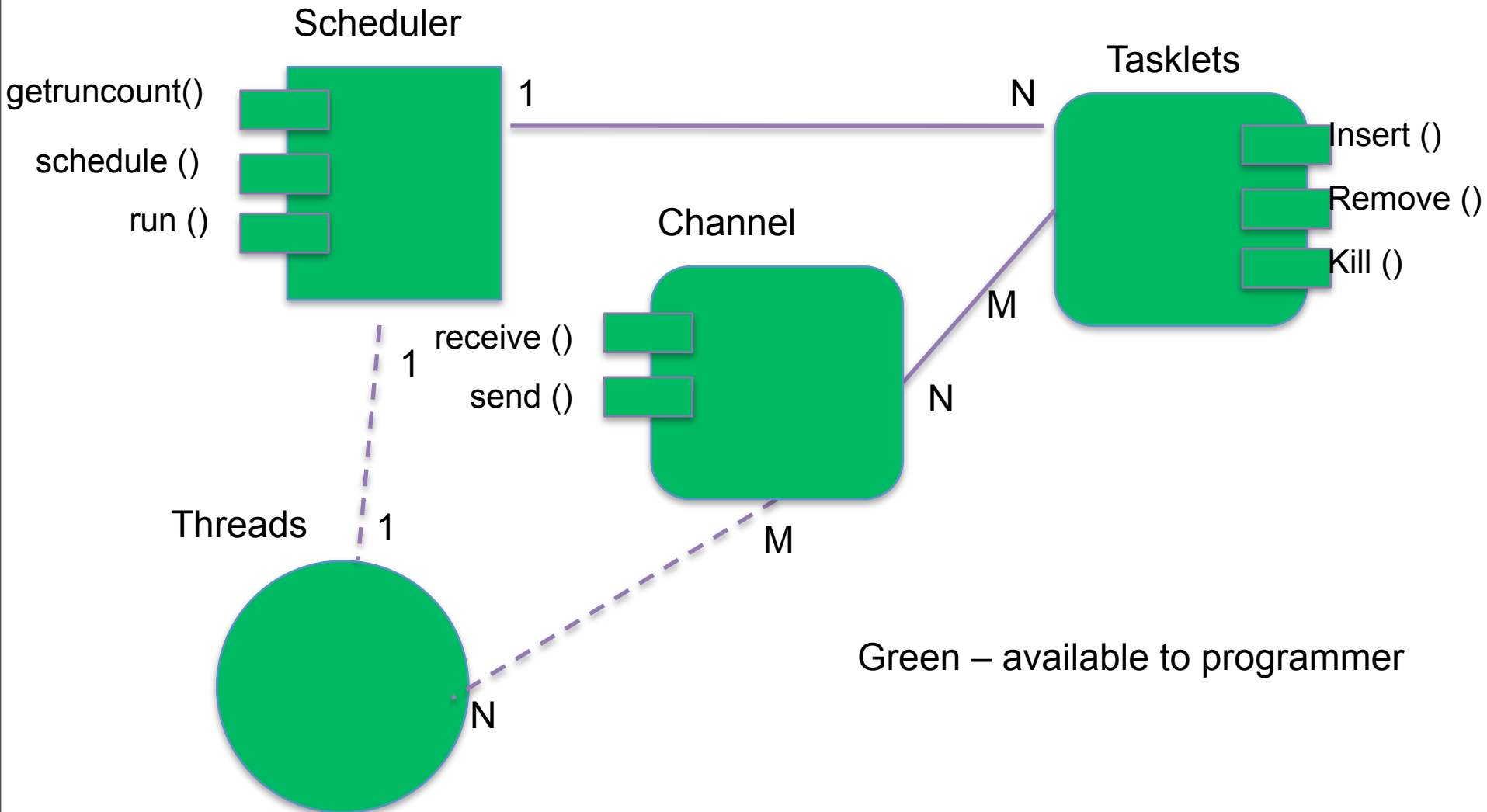
Stackless Python Elements

- Tasklets
  - User space light weight threads
  - Executes actual work
- Scheduler
  - Performances context switching between tasklets
  - Uses a round-robin scheduling
  - Two modes: pre-emptive and cooperative

Channels

- Used for communications and synchronization
- Bi-directional
- Can support iteration
- An object including channels and exceptions can be passed
- Can be subclassed

Stackless Python's World

Scheduler

getruncount()

schedule ()

run ()

1   N

Tasklets

Insert ()

Remove ()

Kill ()

Channel

receive ()

send ()

M   N

Threads

1   1

N   M

Green – available to programmer

Simple Stackless Programme

```python
import stackless

def reader(channel):
    print "entering reader"
    print channel.receive()
    print "exiting reader"


def writer(channel):
    print "entering writer"
    channel.send("hello world")
    print "exiting writer"

if __name__ == "__main__":
    ch = stackless.channel()
    stackless.tasklet(reader)(ch)
    stackless.tasklet(writer)(ch)
    stackless.run()
```

Output

entering reader
entering writer
hello world
exiting reader
exiting writer

A Bad Channel Equivalent

```python
import stackless

tempVal = None

class namedTasklet(stackless.tasklet):
    name = None

    def __repr__(self):
        return self.name

def reader():
    global tempVal
    print "entering reader"
    stackless.schedule()
    print tempVal
    print "exiting reader"

def printQueue():
    h = stackless.getcurrent()
    p = h
    while True:
        print "->", p
        p = p.next
        if p == h:
            break

def writer():
    global tempVal
    print "entering writer"
    tempVal = "hello world"
    t.remove()
    stackless.schedule()
    print "exiting writer"

def publisher(t):
    print "entering publisher"
    printQueue()
    t.insert()
    print
    printQueue()
    stackless.schedule()
    print "exiting publisher"

if __name__ == "__main__":
    t = namedTasklet(reader)()
    t.name = "reader"
    x = namedTasklet(writer)()
    x.name = "writer"
    x = namedTasklet(publisher)(t)
    x.name = "publisher"
    stackless.run()
```

Output

```
entering reader
entering writer
entering publisher
-> publisher
-> writer

-> publisher
-> writer
-> reader
exiting writer
hello world
exiting reader
exiting publisher
```

Why is Bad Important?

- Demonstrates that channels can be constructed from lower level methods available to the programmer
  - schedule()
  - schedule_remove()
- Application programmer has complete control over the scheduler
- Almost all the building blocks are there to build select using the Stackless API!

Channels: Rendezvous Semantics

- For a successful communication to occur there must be a sending tasklet and a receiving tasklet
  - active tasklet is the source
  - inactive tasklet the target
- If there is not a target, the active tasklet will block until another tasklet performs a complimentary operation

How Channels Work
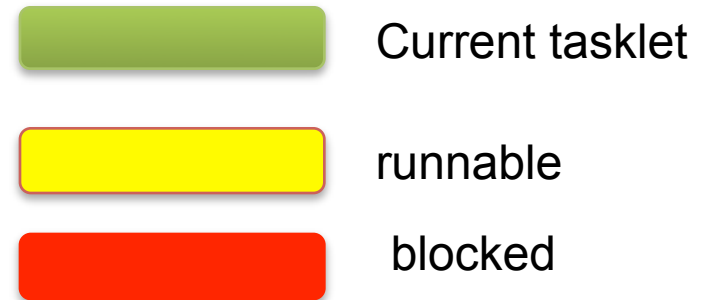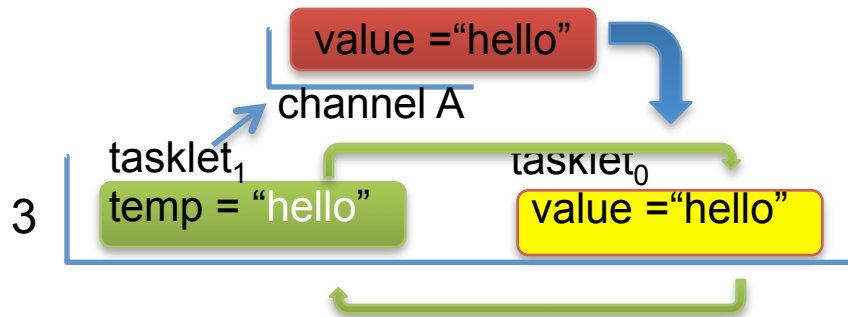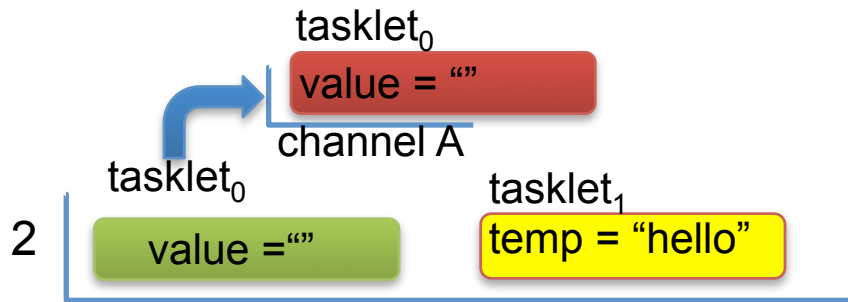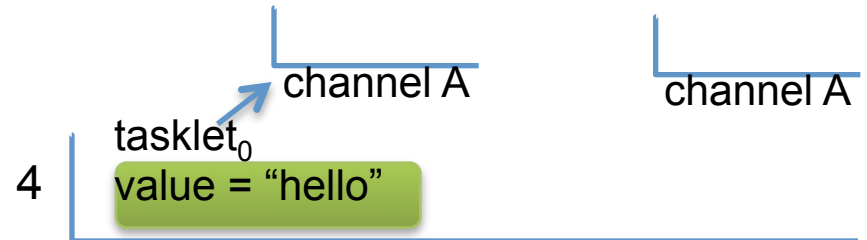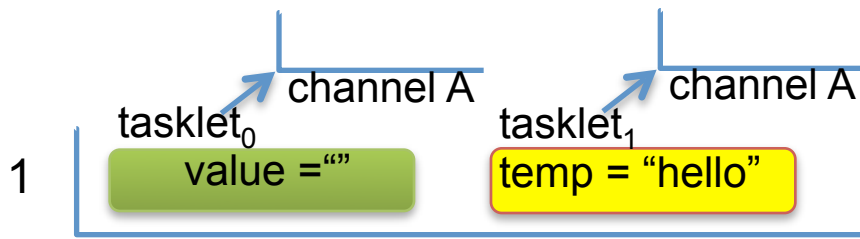
Channel Implementation Details:

```
>>> dir(stackless.channel)
['__class__', '__delattr__', '__doc__', '__format__', '__getattribute__',
'__hash__', '__init__', '__iter__', '__module__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__setstate__', '__sizeof__',
'__slots__', '__str__', '__subclasshook__', 'balance', 'close', 'closed',
'closing', 'next', 'open', 'preference', 'queue', 'receive', 'schedule_all',
'send', 'send_exception', 'send_sequence']
```

-Channel.balance determines whether a tasklet will block
-A balance of zero causes the tasklet to block
- blocked tasklets are placed on a FIFO queue
-send() increments balance by 1
-receive() decrements balance by 1

A limitation of the rendezvous semantics model

```
def eventHandler(channels):
    while True:
        for ch in channels:
            message = ch.receive()
            stackless.tasklet
(doSomething)(message)
```

Ch[0] ready after T + 10
Ch[1] ready after T + 15
Ch[2] ready after T + 1
Ch[3] ready after T + 2

Clearly this is a throughput problem....

Enter Go

- Very different from Python
  - Statically typed
  - Compiled
  - limited OO features
- Similar concurrency constructs
  - Implemented as language features
  - Support for multiple CPUs

Simple Go Programme

```go
package main

import fmt "fmt"

func reader(in chan int) {
    fmt.Printf("entering reader\n");
    x := <- in;
    fmt.Printf("->%d \n", x);
    fmt.Printf("exiting reader\n");
    in <- 1
}


func writer(out chan int) {
    fmt.Printf("entering writer\n");
    out <- 1;
    fmt.Printf("exiting writer\n");
}
```
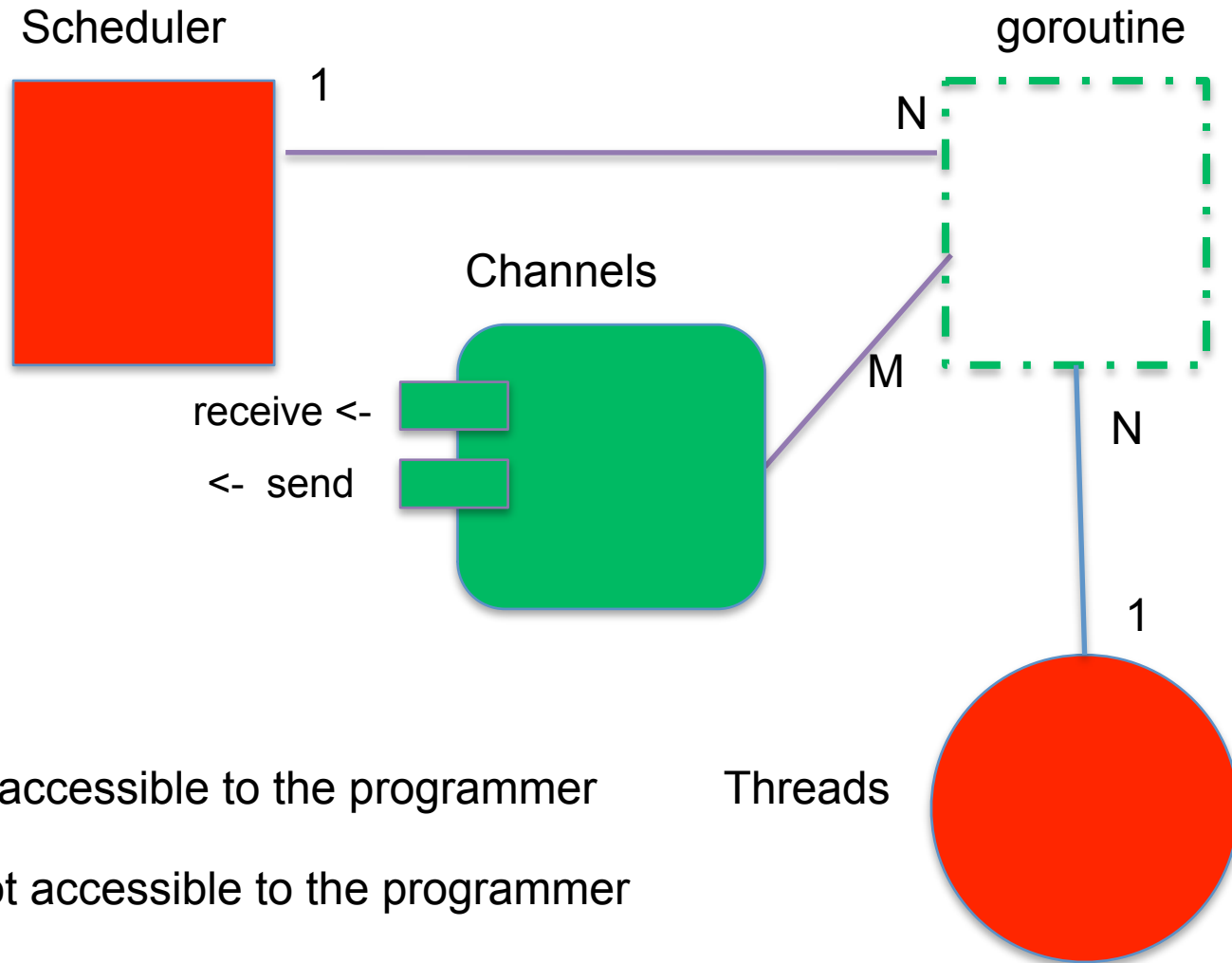
```go
func main() {
    var ch = make(chan int);
    fmt.Printf("I got here \n");
    go reader(ch);
    go writer(ch);
    fmt.Printf("Main Ending \n");
    <-ch;
}
```

Go Constructs

Go	Stackless Python
variable <- channel	variable = channel.receive()
Channel <- variable	channel.send(variable)
data, ok = ch <- variable	if channel.balance != 0

Go's World

Scheduler

goroutine

1

N

Channels

M

receive <-

<- send

N

1

Green – accessible to the programmer

Threads

Red – not accessible to the programmer

So?

- 'Bad' example could not be implemented in Go
  - goroutines are not objects
  - Scheduler almost totally opaque to the programmer
- Under the hood
  - Support for multiple CPUs requires extensive locking
  - Is fine grained control over the scheduler in such an environment desirable?

The Select Statement

```
select {
case a := <- ch[0]:
    go doSomething(a);
case b := <- ch[1]
    go doSomething(b);
case c := <- ch[2]
    go doSomething(c);
case d := <- ch[3]:
    go doSomething(d);
}
```

Could We Implement Select with Only Stackless Python?

- Yes
  - Use an additional tasklet per case
    - And an extra join channel
  - Hard to mimic behaviour
    - Problem dealing with tasklets that unblock after the select has finished
    - The 20% that requires 80% of the effort?
  - Bad performance?

Fragment of a Pseudo Solution

```
def select(cases):
    selector = stackless.channel()

    def case(ch,operation, value):
        if operation == RECEIVE:
            value = ch.receive()
        else:
            ch.send(value)

        selector.send(ch, operation, value)

    for ch, op, value in cases:
        if op == RECEIVE:
            stackless.tasklet(reader)(ch, value)
        else:
            stackless.tasklet(writer)(ch, value)

    # block until a case is ready
    retChannel, retOperation, retVal = selector.receive()
```

stackless.py

Section four

Description

- A Python based implementation of the Stackless Python module
  - scheduler, channels, tasklets
  - Single file, roughly 650 lines

- Part of the PyPy Framework
  - Currently PyPy implements Python 2.5
  - Although Stackless is implemented, it is not integrated with the JIT.

```
andrew@parker:~$ pypy-c
Python 2.5.2 (75825, Jul 05 2010, 02:48:27)
[PyPy 1.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
And now for something completely different: ``it's not a hack, it's a
workaround''
>>>> import stackless
>>>> dir(stackless)
['DEBUG',  'TaskletExit', '__all__', '__builtins__', '__doc__', '__file__',
'__name__', '__nrand_next', '_channel_callback', ,'_global_task_id', '_init',
'_last_task', '_main_coroutine', '_main_tasklet', '_run_calls',
'_schedule_callback', '_scheduler_append', '_scheduler_contains',
'_scheduler_remove', '_scheduler_switch', '_squeue',
'_stackless_primitive_registry', 'bomb', 'channel', 'coroutine', 'debug', 'deque',
'dprint', 'getcurrent', 'getmain', 'getruncount', 'greenlet', 'nrand',
'operator', 'register_stackless_primitive', 'rewrite_stackless_primitive', 'run',
'schedule', 'schedule_remove', 'set_channel_callback',
'set_schedule_callback', 'sys', 'tasklet', 'traceback']
>>>>
```
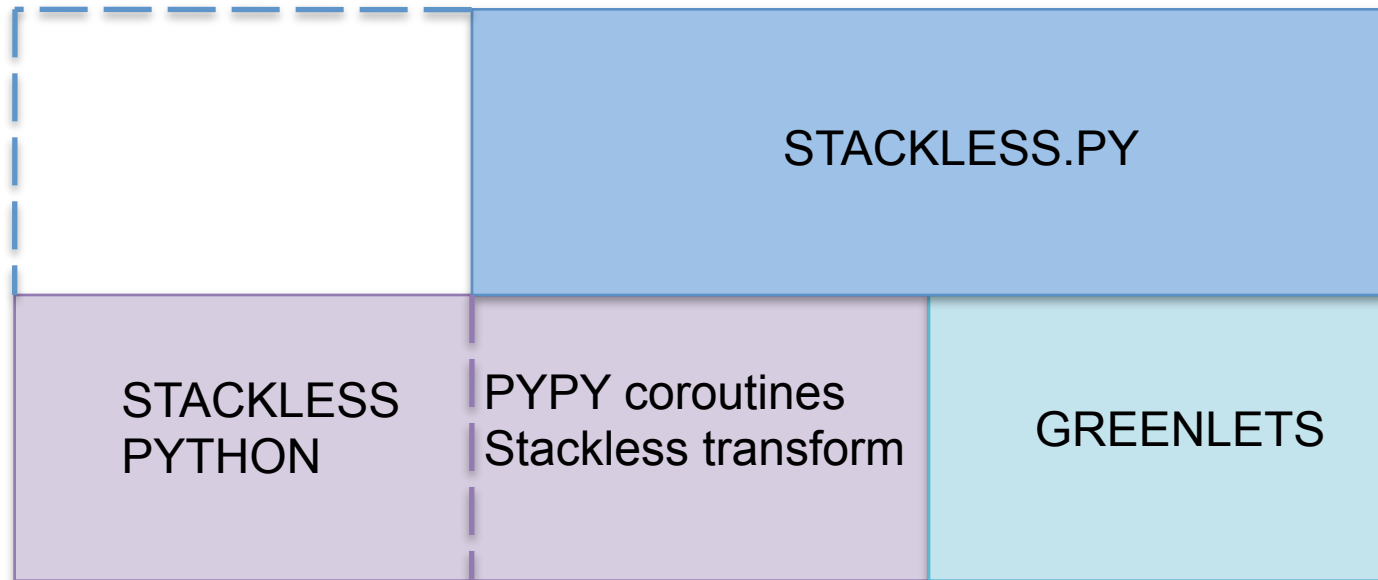
stackless.py's Abstraction Layer

Example of Abstraction Layer in code

```
try:
    from _stackless import coroutine, greenlet
except ImportError: # we are running from CPython
    from greenlet import greenlet
    try:
        from functools import partial
    except ImportError: # we are not running python 2.5
```

Usage

- Stackless.py resides in pypy/lib
- PyPy interpreter on top of Standard Python
  - Too slow
  - Defeats purpose of rapid prototyping
- pypy-c
  - Like Stackless Python requires a separate binary
  - Avoid lengthy build by getting precompiled version

A Trick of the Trade

- Use the Greenlets package with standard Python
  - Low level microthreading package
  - Many Python packages use greenlets
    - Eventlets
    - gEvent
  - stackless.py already included!
- This is the approach the PyPy team used to develop stackless.py module

Limitations

- Does not support threads
- Does not support pre-emptive mode
- Does not implement all of the class attributes

The Select Algorithm

Section Five

```python
def select(self, 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 = self._operation
        self._operation = None


     return choice.result()
```
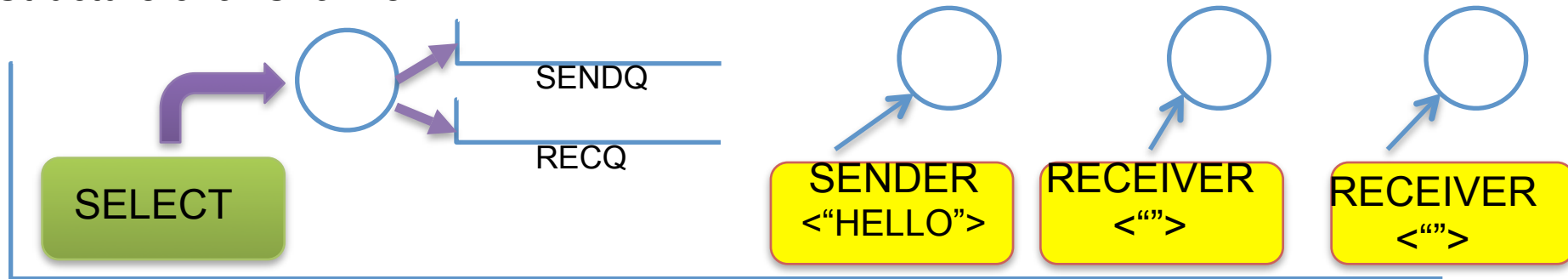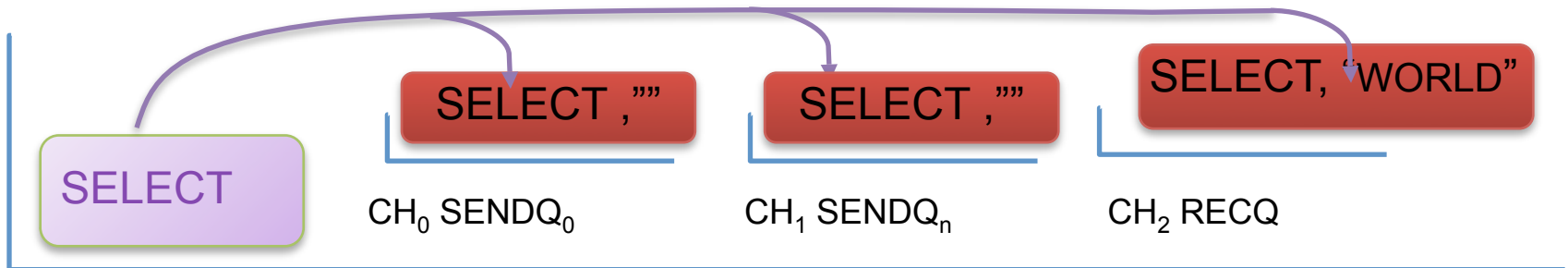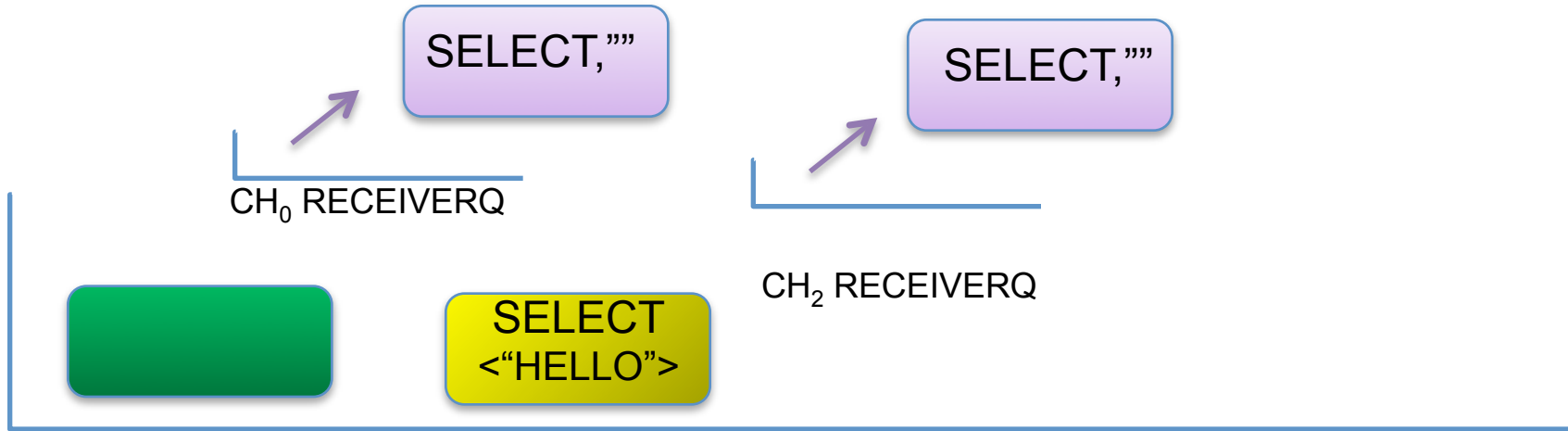
Select Under the Hood

Structure of a Channel

SENDQ

RECQ

SELECT

SENDER
<"HELLO">

RECEIVER
<"">

RECEIVER
<"">

Runnable Queue

BLOCKING

SELECT

SELECT ,""

SELECT ,""

SELECT, "WORLD"

CH_0 $SENDQ_0$

CH_1 $SENDQ_n$

CH_2 RECQ

Select Under the Hood

Tear Down

SELECT,""

SELECT,""

CH_0 RECEIVERQ

CH_2 RECEIVERQ

SELECT
<"HELLO">

- the source coroutine
- Transfers data to target (with select)
- Takes select coroutine off the participating channel queues
- Places target on the runnable list

def _channel_action()

```
def _channel_action(self, operation):

    if _channel_callback is not None:
        _channel_callback(self, getcurrent(), operation)

        target = self.queue

        operation.copyOperation(target)
        target.tasklet._operation = target

        #clear operation from remaining channels
        target.removeall()

        target.tasklet.blocked = 0
```

Source channel responsible for moving data and tearing down channels

Implementing select with stackless.py

Section Five

New Stackless Methods

- stackless.select(list of chanops)
  - returns (channel, operation, value)
- channel.sendCase()
  - returns _chanop
- channel.receiveCase()
  - returns _chanop

New Class: _chanop

_chanop(RECEIVE)

_chanop(SEND, value)

```
select {
case a := <- ch[0]:
      go doSomething(a);
case b := <- ch[1]
      go doSomething(b);
case c := <- ch[2]
      go doSomething(c);
case ch[3]<- d:
      go doSomethingElse();
}
```

Example

```
def selector(a,b,c):
    while flag:
        ch, operation, value = stackless.select(\
[a.sendCase("A"), b.receiveCase(), c.sendCase("C")])

        if ch == a:
            print "sender A completed"
        elif ch == b:
            print "received ", value, "from receiver B"
        elif ch == c:
            print "sender C completed"
            flag = False
        else:
            print "should not get here"
```
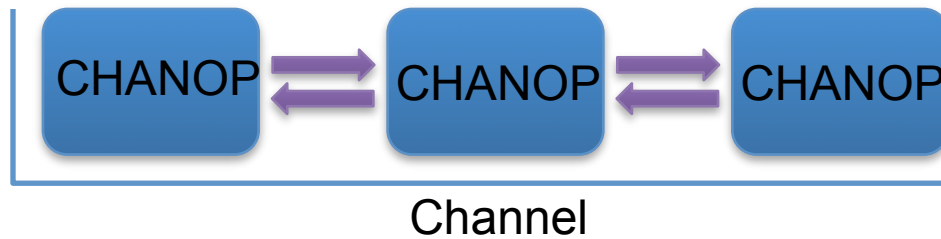
Changes to Channels

BEFORE

$$\text{TASKLET}_0 \quad \text{TASKLET}_1 \quad \text{TASKLET}_2$$
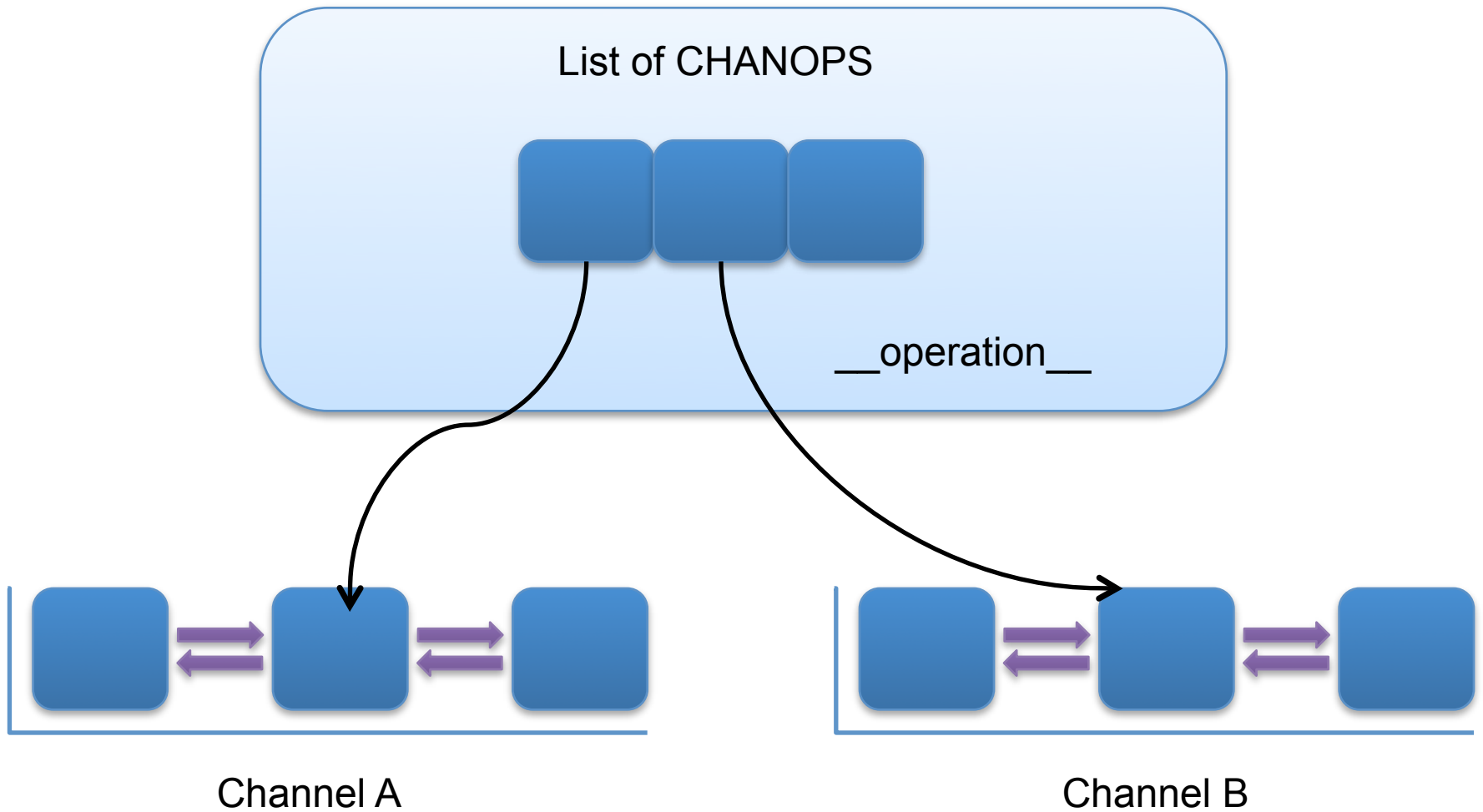
Channel

AFTER

CHANOP ⇄ CHANOP ⇄ CHANOP

Channel

New Channel Methods

- channel.addOperation(chanop)
  - need way to add operations to channels without blocking

Changes to Tasklets

Short Cuts

- For now, separate send and receive queues are not implemented.
- channel.balance remains
  - its so handy…
- Limitation
  - A receive chanop and send chanop cannot be on the same channel queue

What Breaks?

- Only applications that depends on internal state

  - A channel queue now consists of chanops not tasklets

  - What does __channel__ now mean?

  - channel.balance could disappear.

Moving to Stackless Python

Chapter Five

The Stackless Python C Version

- Programmer now responsible for
  - Setting up house keeping structures (especially for C Extensions)
  - Memory allocation
  - Reference counting
  - Manipulating complex data structures
- Stackless Python C code much more verbose
  - ~1000 lines versus ~150

Example: Channel_action

```
PyAltObject *
slp_alt_action(PyAltObject *self)
{
        PyThreadState *ts = PyThreadState_GET();
        PyTaskletObject *t = ts->st.current;
        PyAltObject *target;

        assert(PyAlt_Check(self));
        assert(self->tasklet == t);

        target = slp_channel_remove(self->channel);
        if (target == NULL)
                return NULL;
```

Channel_action Continued

```
        assert(PyAlt_Check(target));
        assert(target->tasklet != t);

        Py_INCREF(target);

        alt_copy(self, target);
        alt_remove_all(target);

        target->tasklet->flags.blocked = 0;

        return target;
}
```

Summary of C Code

- Most of the changes are isolated in:
  - channel's generic_channel_action
  - The alt object
- Stackless Python C code is just a C equivalent of the stackless.py code
- Esoteric issues concerning stack frames and locks are avoided
  - The GIL is our friend

Conclusions

Lessons Learnt

- Select relative easy to implement but requires substantial changes
- Clean room descriptions good
- Concentrate on proper API
- Initially copy to learn.
- Get working prototypes up and running quickly

Status

- Not quite prime time
  - Slight signature differences between C and stackless.py
  - C version fails a few unit tests
    - Mostly pertaining to pickling
  - Problems with complex select tests
    - Both C and stackless.py
  - Performance problem with stackless.py
    - A bug with channel preferences?

However this was meant to be a prototype not production code!

Whither C Stackless Python?

- Psyco JIT/Stackless Python integration will make
  - Stackless binaries a thing of the past
  - Minimize the need for writing C-extensions
- PyPy-C
  - with JIT, faster than Cpython
  - What will happen when stackless support is integrated with the JIT?

Future Directions

- Experiment with supporting select as a language feature
- Optimize select
  - Can we avoid costly teardowns?
- Prototype other concurrency features
  - Join patterns a la Polyphonic #C and Jocaml
  - Is this a gateway to Complex Event Processing?

Have Prototype Will Travel!

References

- "The Implementation of Newsqueak" by Rob Pike
- http://www.stackless.com
- http://codespeak.net/pypy/dist/pypy/doc/stackless.html
- The GoLang-Nuts mailing list
- http://swtch.com/usr/local/plan9/src/libthread/channel.c
- "Stuff What I Posted", Richard Tew's Blog

Acknowledgements

- Special thanks to
  - Kevin Bulušek !!!!
  - Annette Hollman (for helping me with the slides)
  - The McGill Continuing Education Student Building staff (MACES)
  - And ….

For allowing me to do a dry run of this talk!

Questions?

Thank You