A Technical Anatomy of How OpenMPI Applications Can Inherit Fault Tolerance Using SPM.Python

Minesh B. Amin
mamin@mbasciences.com
http://www.mbasciences.com

PyHPC Workshop
Supercomputing Conference 2011
Seattle, Washington
Nov 18, 2011

Prologue

GNU/Linux [] mpirun ... **./hello_world** -prefix "api"

Prologue

GNU/Linux [] mpirun ... **./hello_world** -prefix "api"

Typical OpenMPI application ...

Prologue

GNU/Linux [] mpirun ... **./hello_world** -prefix "api"

Typical OpenMPI application ... that lacks support for:

Prologue

GNU/Linux [] mpirun ... **./hello_world** -prefix "api"

Typical OpenMPI application ... that lacks support for:

- fault tolerance

Prologue

GNU/Linux [] mpirun ... **./hello_world** -prefix "api"

Typical OpenMPI application ... that lacks support for:

- fault tolerance
- timeout

Prologue

GNU/Linux [] mpirun ... **./hello_world** -prefix "api"

Typical OpenMPI application ... that lacks support for:

- fault tolerance
- timeout
- detection of deadlocks

Prologue

GNU/Linux [] mpirun ... **./hello_world** -prefix "api"

Typical OpenMPI application ... that lacks support for:

- fault tolerance
- timeout
- detection of deadlocks

$\Rightarrow$ Prototyping is (deeply)$^{\infty}$ frustrating

Problem Statement

Prototyping should be frictionless

Problem Statement

Prototyping should be frictionless

↓

Must use original OpenMPI application

- original source code
- original binary

Problem Statement

Prototyping should be frictionless

↓

Must use original OpenMPI application

- original source code
- original binary

↓

Original OpenMPI application must inherit support for:

- fault tolerance
- timeout
- detecting deadlocks

Problem Statement

Prototyping should be frictionless

↓

Must use original OpenMPI application

- original source code
- original binary

↓

Original OpenMPI application must inherit support for:

- fault tolerance
- timeout
- detecting deadlocks

↓

GNU/Linux [] **spm.python ...** ~~mpirun~~ ... **./hello_world** -prefix "api"

Problem Statement (Cont'd)

GNU/Linux [] **spm.python ...** ~~mpirun ...~~ **./hello_world** -prefix "api"

Problem Statement (Cont'd)

GNU/Linux []

spm.python ...
~~mpirun~~ ...

./hello_world -prefix "api"

A

Problem Statement (Cont'd)

GNU/Linux []

spm.python ...
~~mpirun~~ ...

B

./hello_world -prefix "api"

A

Problem Statement (Cont'd)

GNU/Linux []

spm.python ...
~~mpirun~~ ...

B

./hello_world -prefix "api"

A

Exploiting two very different forms of parallelism:

- Using same resources
- At the same time

Problem Statement (Cont'd)

GNU/Linux []

spm.python ...
~~mpirun~~ ...

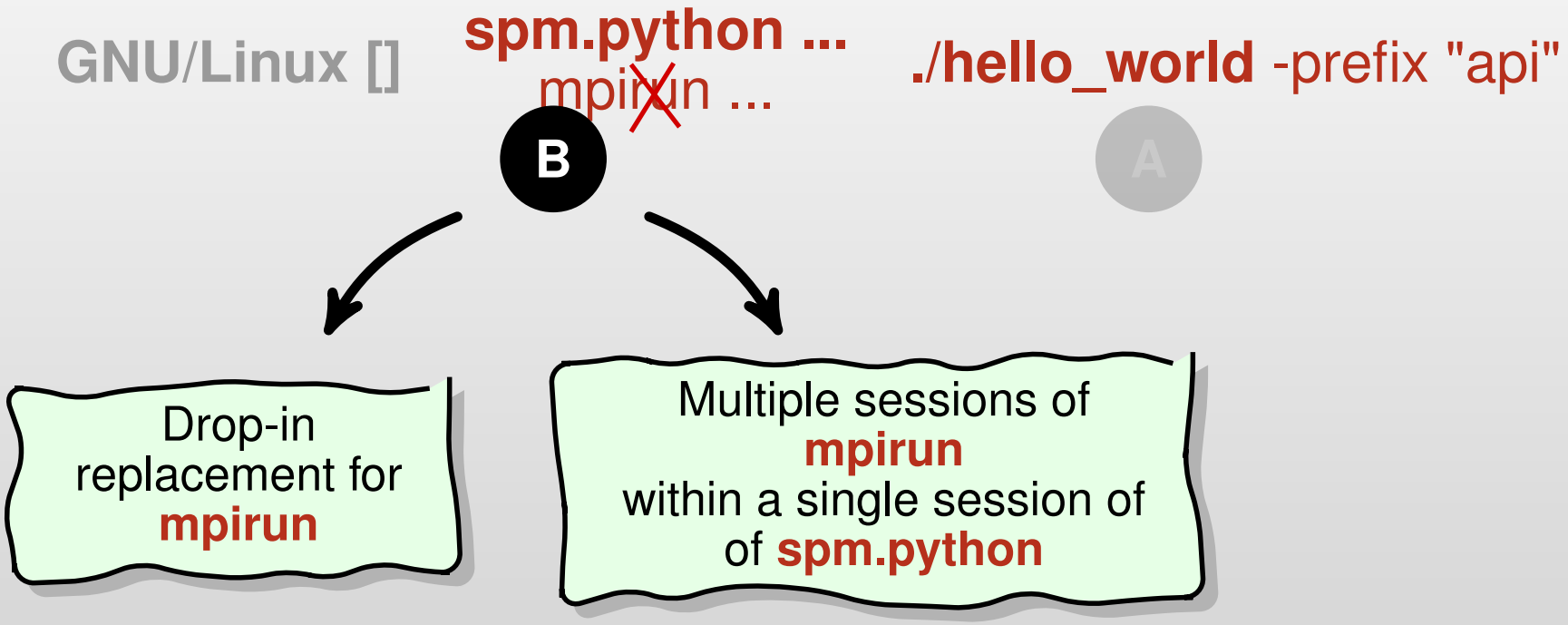
B

./hello_world -prefix "api"

A

Drop-in replacement for **mpirun**

Problem Statement (Cont'd)



Problem Statement (Cont'd)

GNU/Linux []

spm.python ...
~~mpirun~~ ...

B

./hello_world -prefix "api"

A

Drop-in replacement for **mpirun**

Multiple sessions of **mpirun** within a single session of of **spm.python**

Can use same resources for:

- Checkpoint based parallelism
- What-if analysis
- Stress testing

Terminology: "Exploiting Parallelism"

Exploiting parallelism entails the management of a collection of serial tasks which may communicate using only compatible communication primitives

Terminology: "Exploiting Parallelism"

Exploiting parallelism entails the management of a collection of serial tasks which may communicate using only compatible communication primitives

management refers to policies by which:

Terminology: "Exploiting Parallelism"

Exploiting parallelism entails the management of a collection of serial tasks which may communicate using only compatible communication primitives

management refers to policies by which:

- tasks are scheduled,

Terminology: "Exploiting Parallelism"

Exploiting parallelism entails the management of a collection of serial tasks which may communicate using only compatible communication primitives

management refers to policies by which:

- tasks are scheduled,
- premature terminations are handled,

Terminology: "Exploiting Parallelism"

Exploiting parallelism entails the management of a collection of serial tasks which may communicate using only compatible communication primitives

management refers to policies by which:

- tasks are scheduled,
- premature terminations are handled,
- preemptive support is provided,

Terminology: "Exploiting Parallelism"

Exploiting parallelism entails the management of a collection of serial tasks which may communicate using only compatible communication primitives

management refers to policies by which:

- tasks are scheduled,
- premature terminations are handled,
- preemptive support is provided,
- communication primitives are enabled/disabled, and

Terminology: "Exploiting Parallelism"

Exploiting parallelism entails the management of a collection of serial tasks which may communicate using only compatible communication primitives

management refers to policies by which:

- tasks are scheduled,
- premature terminations are handled,
- preemptive support is provided,
- communication primitives are enabled/disabled, and
- the manner in which resources are obtained and released

Terminology: "Exploiting Parallelism"

Exploiting parallelism entails the management of a collection of serial tasks which may communicate using only compatible communication primitives

management refers to policies by which:

- tasks are scheduled,
- premature terminations are handled,
- preemptive support is provided,
- communication primitives are enabled/disabled, and
- the manner in which resources are obtained and released

serial tasks are classified in terms of either:

Terminology: "Exploiting Parallelism"

Exploiting parallelism entails the management of a collection of serial tasks which may communicate using only compatible communication primitives

management refers to policies by which:

- tasks are scheduled,
- premature terminations are handled,
- preemptive support is provided,
- communication primitives are enabled/disabled, and
- the manner in which resources are obtained and released

serial tasks are classified in terms of either:

- Coarse grain ...

  where tasks may not communicate prior to conclusion, or

Terminology: "Exploiting Parallelism"

Exploiting parallelism entails the management of a collection of serial tasks which may communicate using only compatible communication primitives

management refers to policies by which:

- tasks are scheduled,
- premature terminations are handled,
- preemptive support is provided,
- communication primitives are enabled/disabled, and
- the manner in which resources are obtained and released

serial tasks are classified in terms of either:

- Coarse grain ...
  where tasks may not communicate prior to conclusion, or
- Fine grain ...
  where tasks may communicate prior to conclusion.

Terminology: "Exploiting Parallelism"

Exploiting parallelism entails the management of a collection of serial tasks which may communicate using only compatible communication primitives

management refers to policies by which:

- tasks are scheduled,
- premature terminations are handled,
- preemptive support is provided,
- communication primitives are enabled/disabled, and
- the manner in which resources are obtained and released

serial tasks are classified in terms of either:

- Coarse grain ...

n, or

Management policies codify how serial tasks are to be managed ... independent of what they may be

Terminology: "Parallel Enabling Technologies"

Means to the end

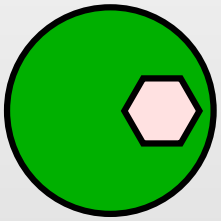
Terminology: "Parallel Enabling Technologies"

Means to the end

- Bottom-up

OpenMPI OpenMP
CUDA OpenGL

- Maximum flexibility
- Maximum headaches
- Must implement fault tolerance



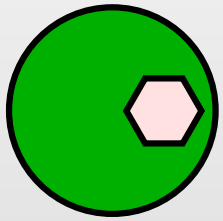
Terminology: "Parallel Enabling Technologies"

Means to the end

- Bottom-up

  OpenMPI OpenMP
  CUDA OpenGL

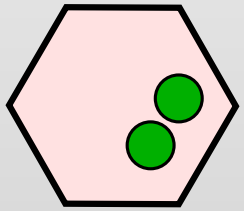
  - Maximum flexibility
  - Maximum headaches
  - Must implement fault tolerance



- Top-down

  Hadoop Goldenorb
  GraphLab

  - Limited flexibility
  - Fewer headaches
  - Fault tolerance is inherited



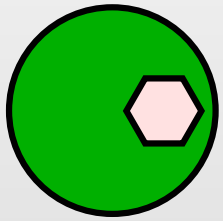
Terminology: "Parallel Enabling Technologies"

Means to the end

- Bottom-up

  OpenMPI OpenMP
  CUDA OpenGL

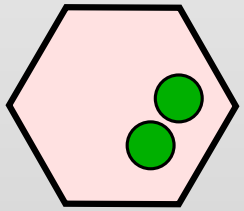
  - Maximum flexibility
  - Maximum headaches
  - Must implement fault tolerance



- Top-down

  Hadoop Goldenorb
  GraphLab

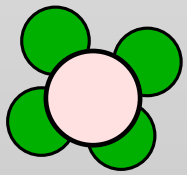
  - Limited flexibility
  - Fewer headaches
  - Fault tolerance is inherited



- Self-contained environment

  SPM.Python

  - Maximum flexibility
  - Fewest headaches
  - Fault tolerance is inherited



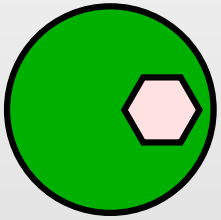
Terminology: "Parallel Enabling Technologies"

Means to the end

- Bottom-up

  OpenMPI OpenMP
  CUDA OpenGL

  - Maximum flexibility
  - Maximum headaches
  - Must implement fault tolerance



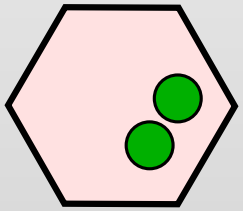
- Top-down

  Hadoop Goldenorb

  - Limited flex

  N environments/installations for *N* Frameworks

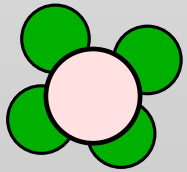
  ...lerance is inherited



- Self-contained environment

  SPM.Python

  - Maximum flexibility
  - Fewest headaches
  - Fault tolerance is inherited



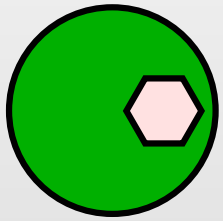
Terminology: "Parallel Enabling Technologies"

Means to the end

- Bottom-up

  OpenMPI OpenMP
  CUDA OpenGL

  - Maximum flexibility
  - Maximum headaches
  - Must implement fault tolerance



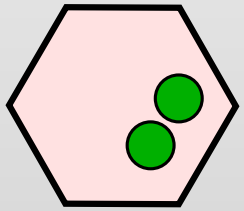
- Top-down

  Hadoop Goldenorb

  - [illegible]

  N environments/installations for *N* Frameworks

  [illegible] tolerance is inherited

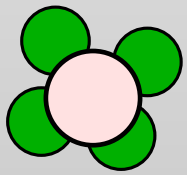


- Self-contained environment

  SPM Pyth[illegible]

  One environment/installation, *N* suites of Pclosures

  [illegible] tolerance is inherited



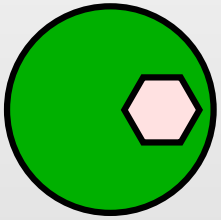
Terminology: "Parallel Enabling Technologies"

Means to the end

- Bottom-up

  OpenMPI OpenMP
  CUDA OpenGL

  - Maximum flexibility
  - Maximum headaches
  - Must implement fault tolerance

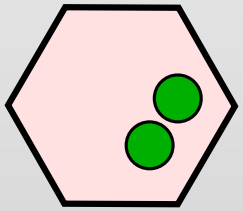


- Top-down

  Hadoop Goldenorb

  N environments/installations for *N* Frameworks

  tolerance is inherited



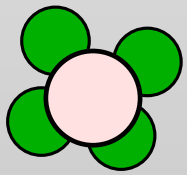
- Self-contained environment

  SPM

  One

```
>>> cmdA
>>> cmdB
>>> cmdC
>>> cmdD
```

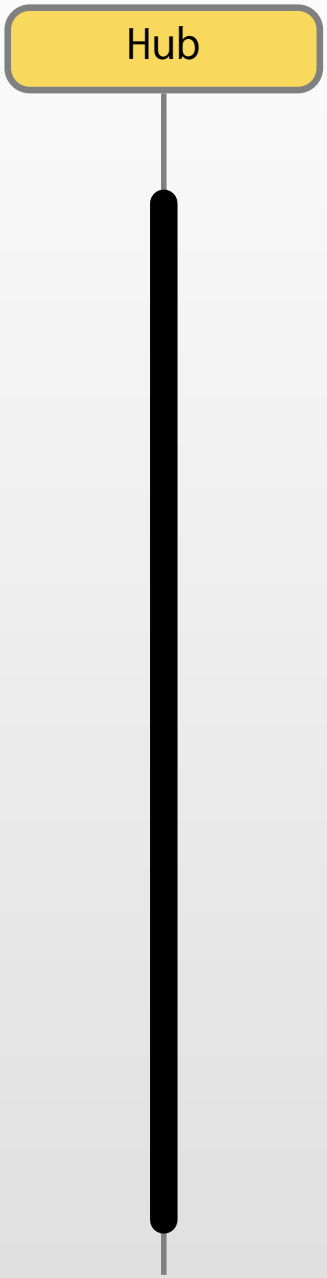
```
>>> createVirtualCloud -async
>>> cmdA -parallel
>>> cmdB -parallel
>>> cmdC -parallel
>>> cmdD -parallel
```



Anatomy: Timeline

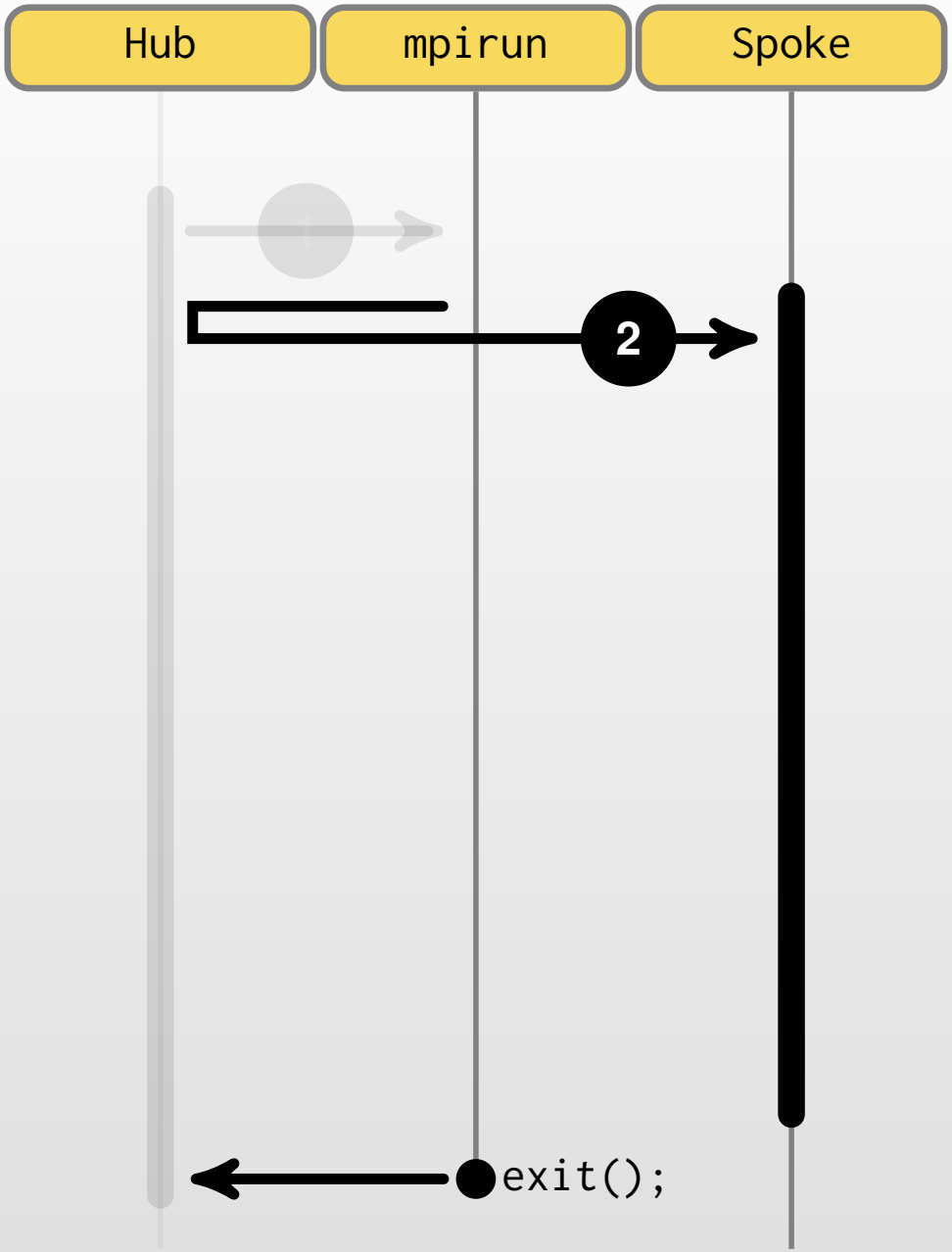
GNU/Linux [] **spm.python ...** ~~mpirun~~ **./hello_world** -prefix "api"

Anatomy: Timeline (Cont'd)

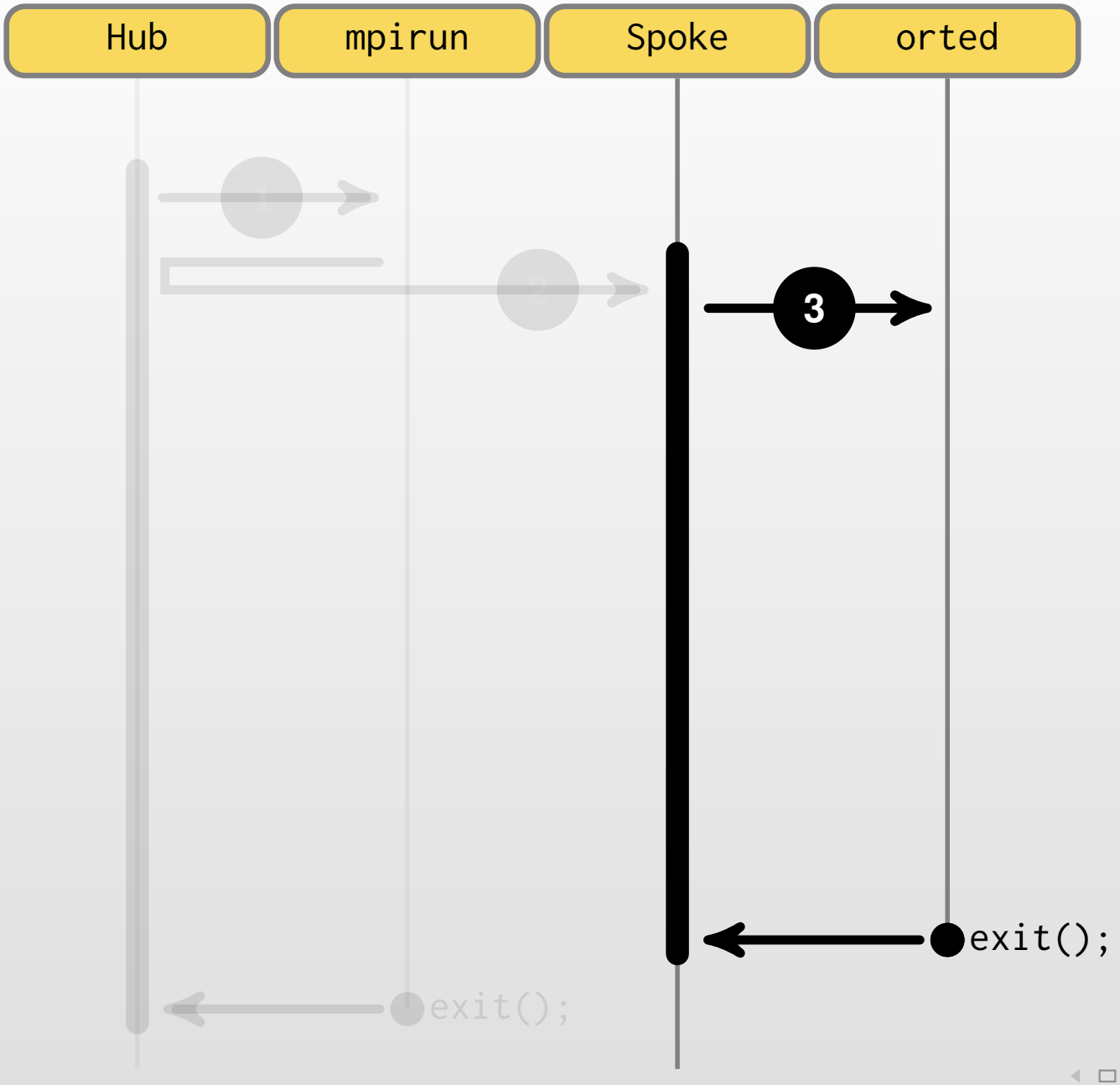


Anatomy: Timeline (Cont'd)

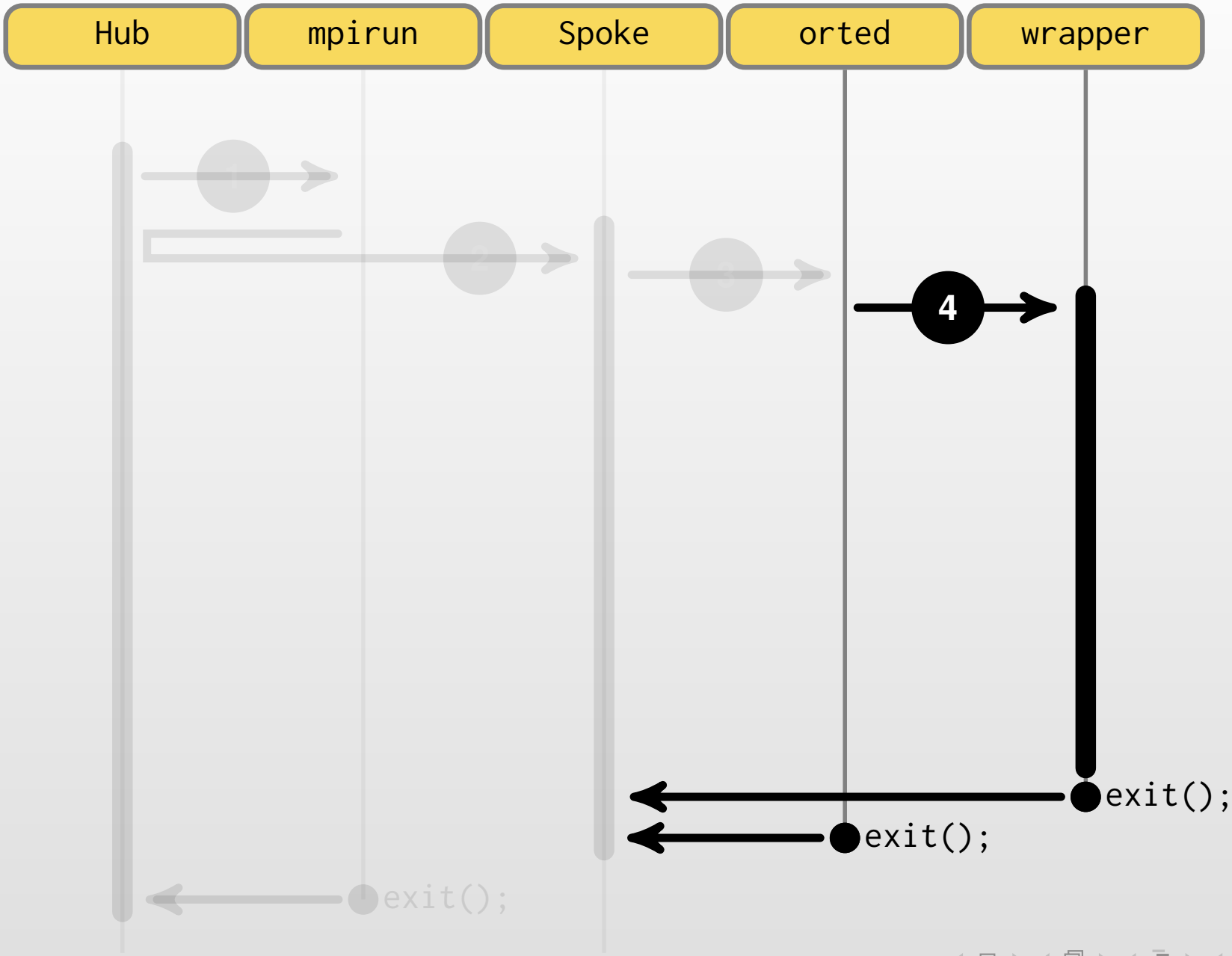
Anatomy: Timeline (Cont'd)



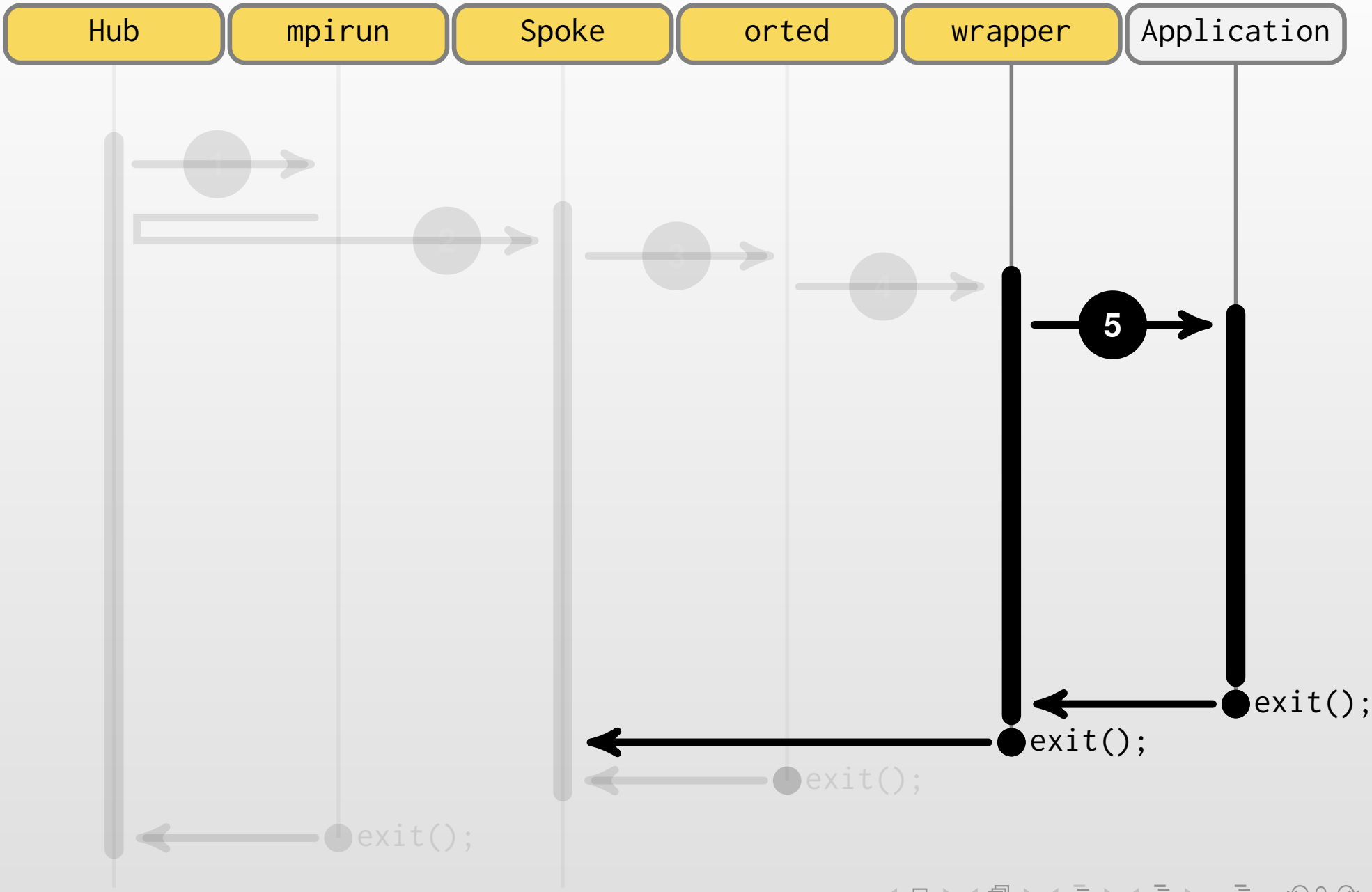
Anatomy: Timeline (Cont'd)



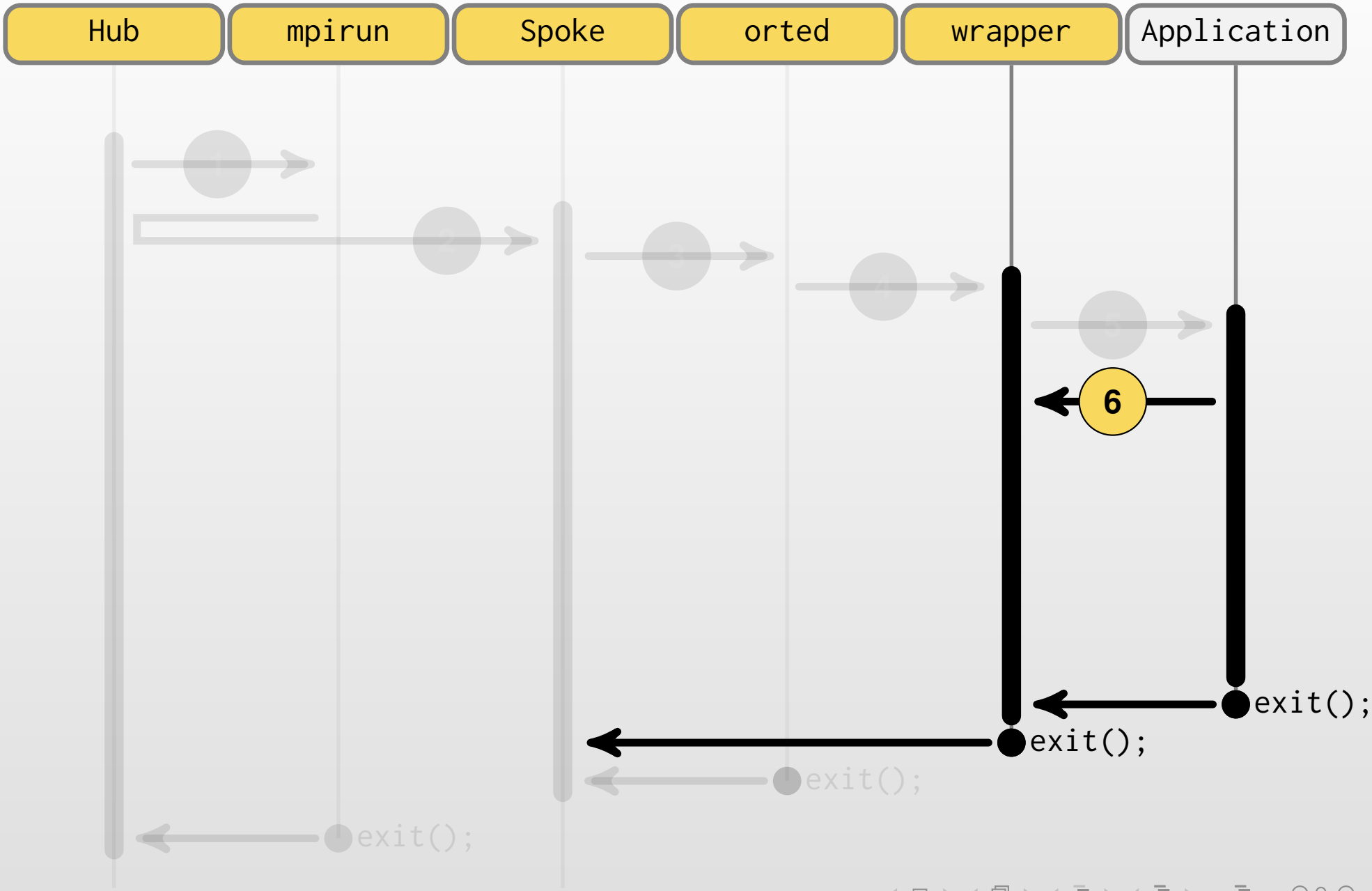
Anatomy: Timeline (Cont'd)



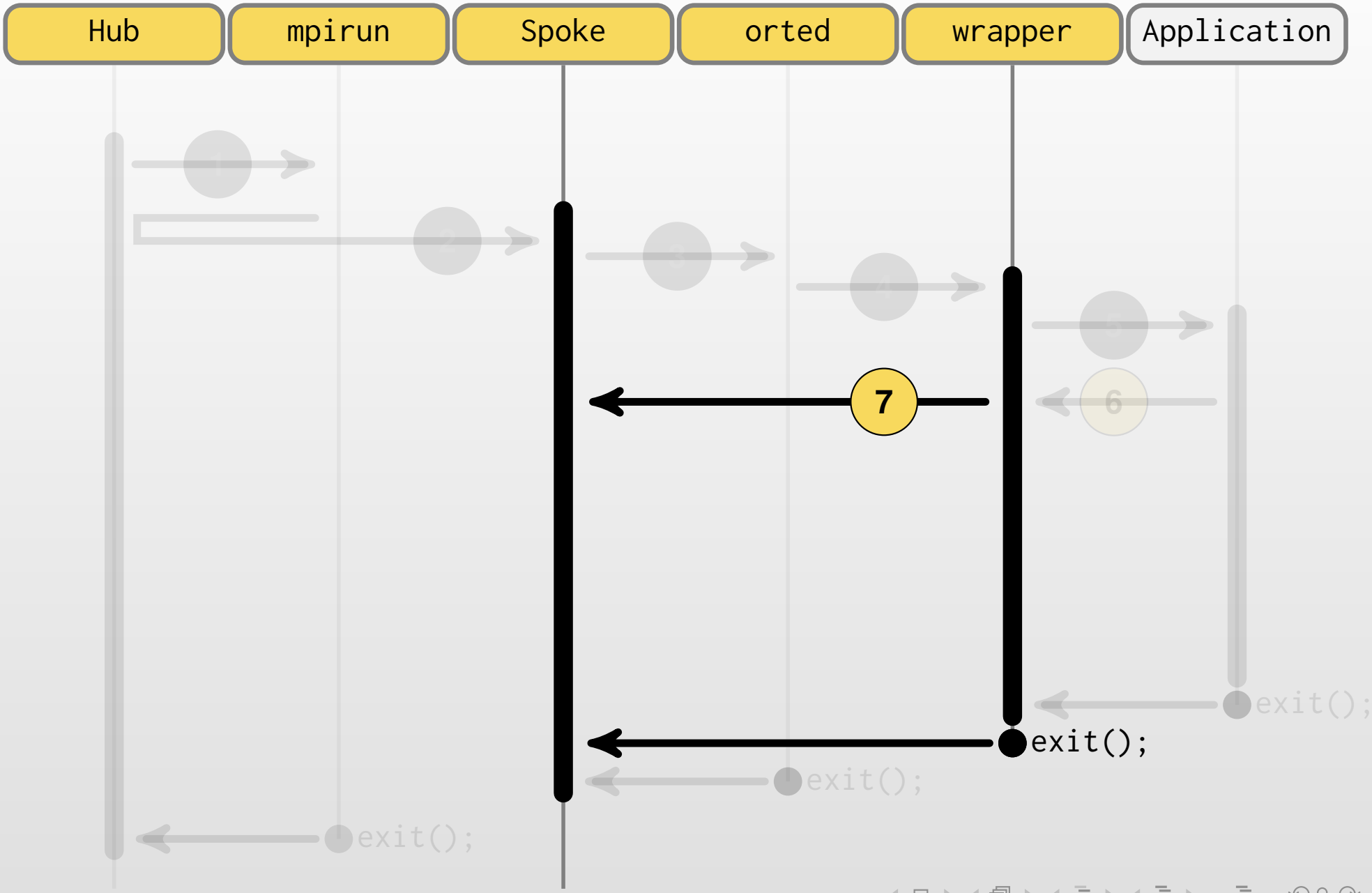
Anatomy: Timeline (Cont'd)



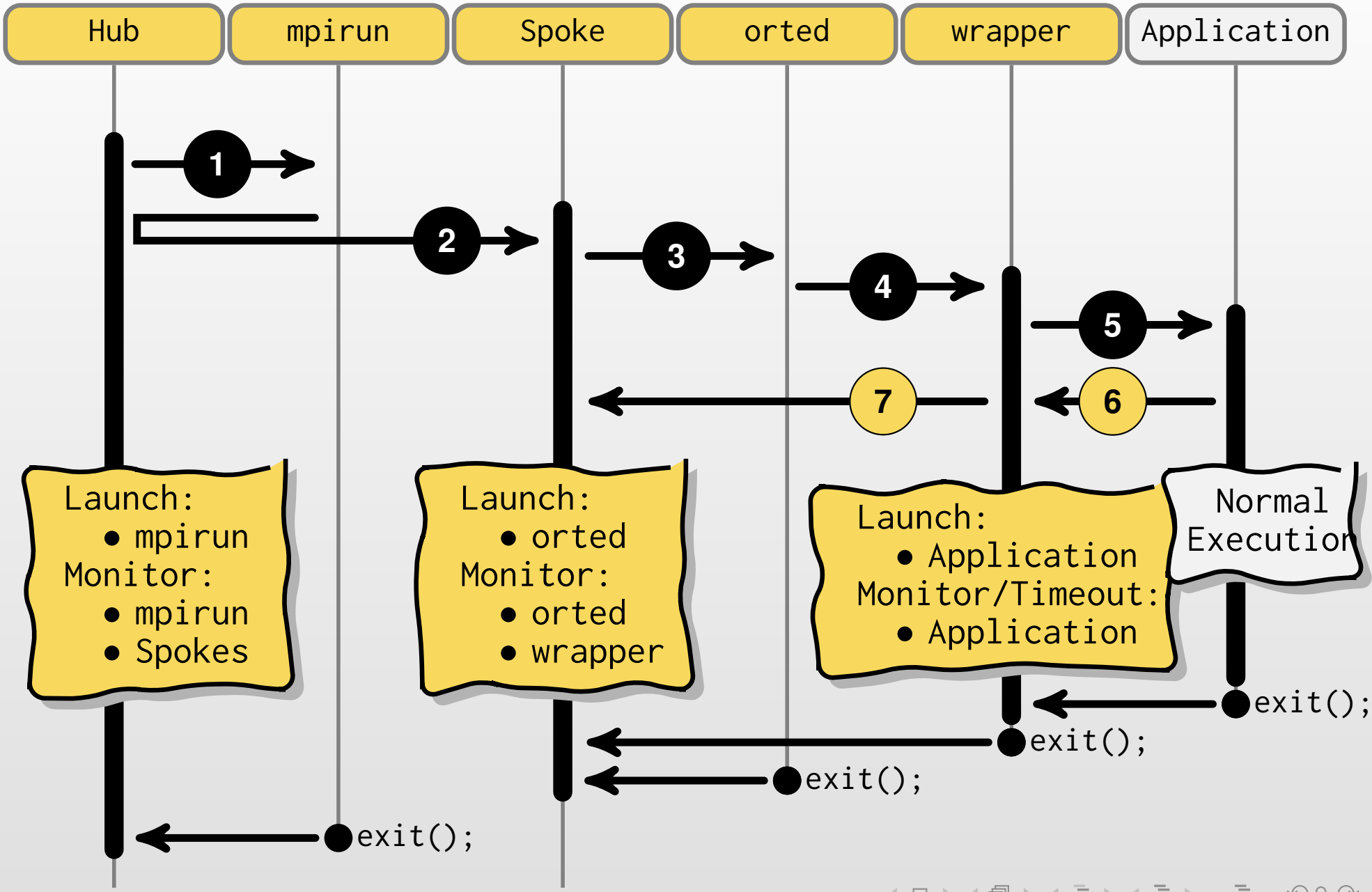
Anatomy: Timeline (Cont'd)



Anatomy: Timeline (Cont'd)



Anatomy: Timeline (Cont'd)

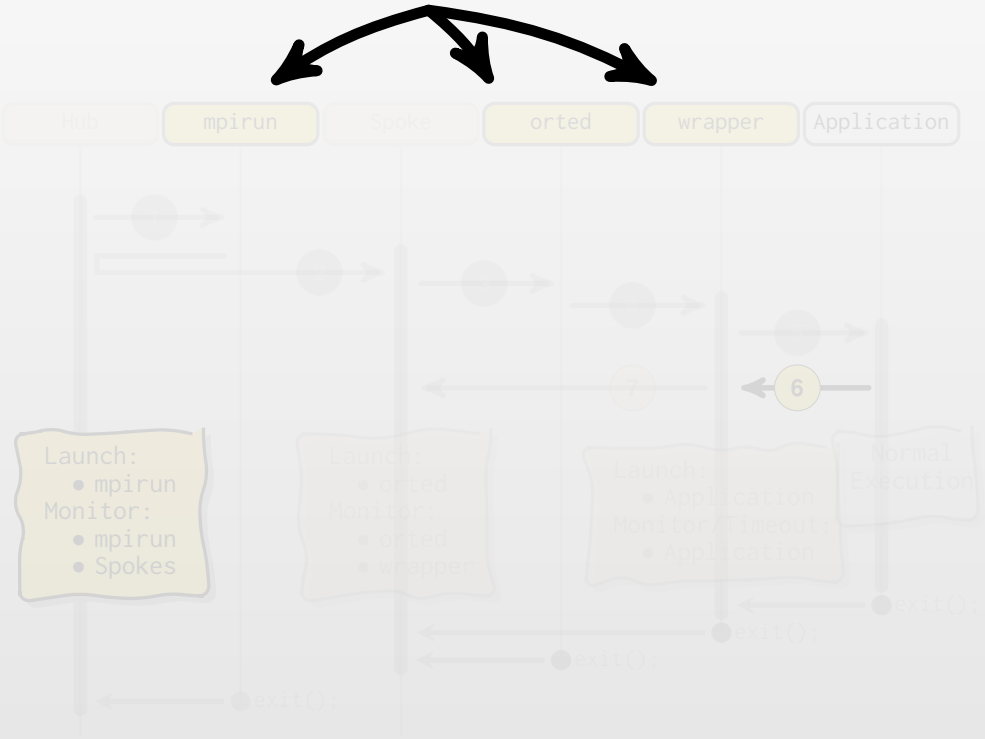


Anatomy: Breakdown

Anatomy: Breakdown

Built-in Package Management System

- Selectively change default OpenMPI env



Built-in Package Management System

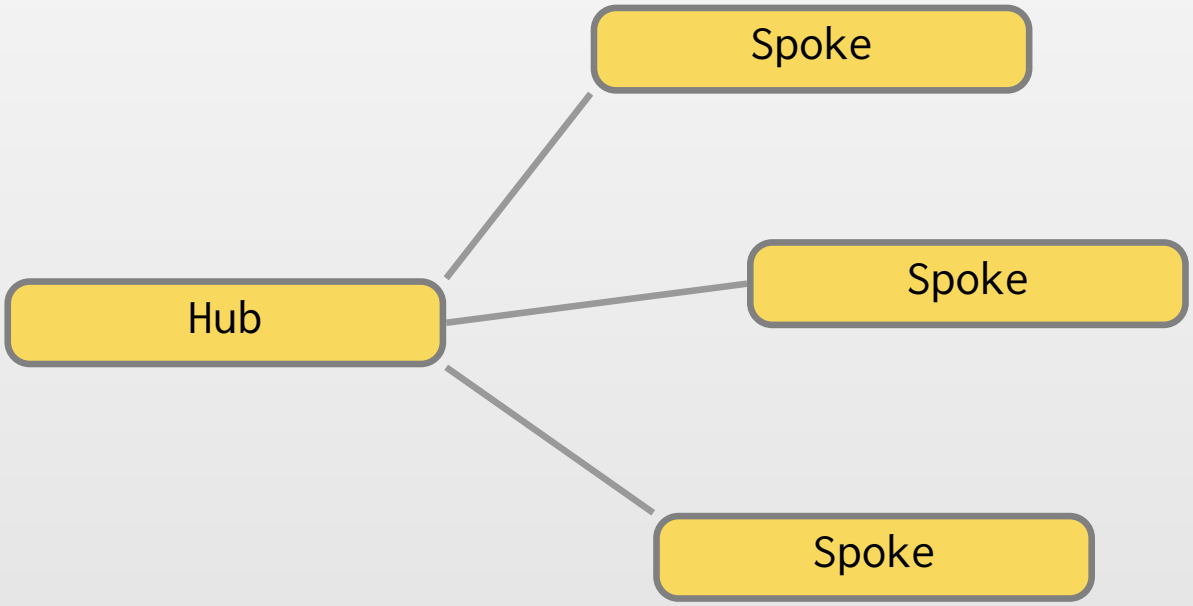
```
>>>  sys.path = [ ".", "/-@-/pkg.builtin", "/opt/default" ]
>>>  import pycuda
```

Built-in Package Management System

```
>>>   sys.path = [ ".", "/-@-/pkg.builtin", "/opt/default" ]
>>>   import pycuda
```

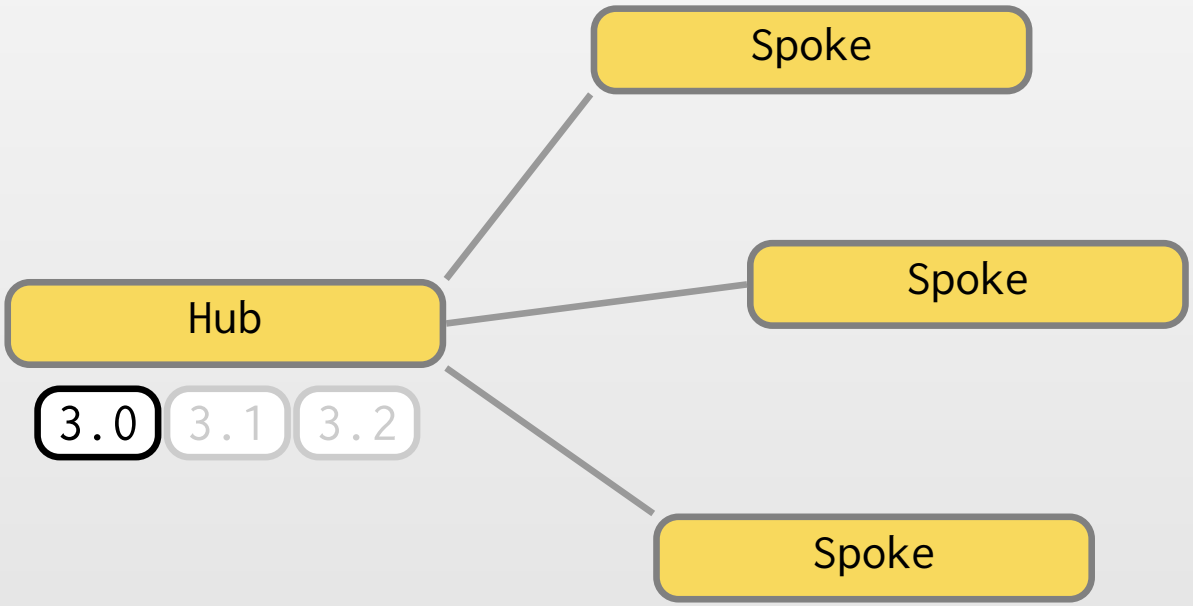
Built-in Package Management System

```
>>>   sys.path = [ ".", "/-@-/pkg.builtin", "/opt/default" ]
>>>   import pycuda
```



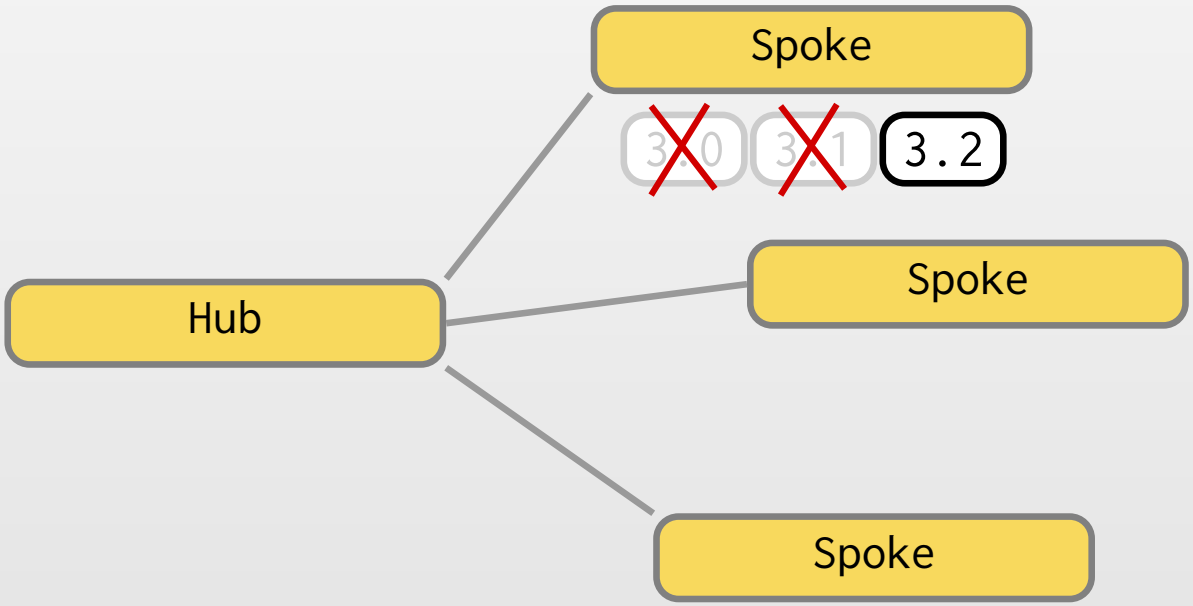
Built-in Package Management System

```
>>>   sys.path = [ ".", "/-@-/pkg.builtin", "/opt/default" ]
>>>   import pycuda
```



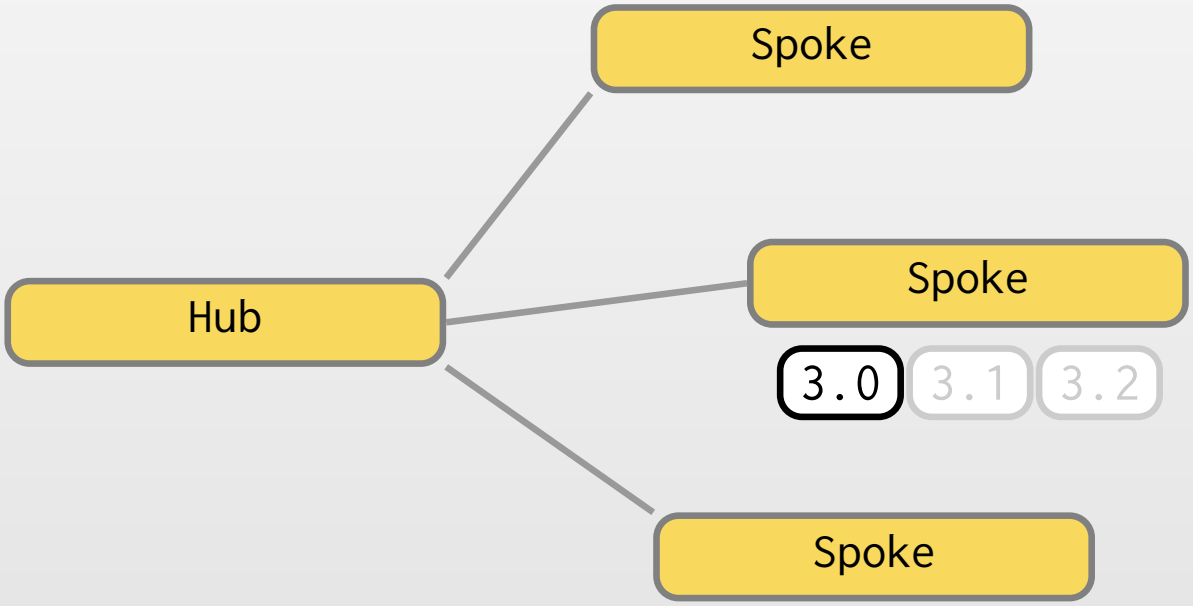
Built-in Package Management System

```
>>>   sys.path = [ ".", "/-@-/pkg.builtin", "/opt/default" ]
>>>   import pycuda
```



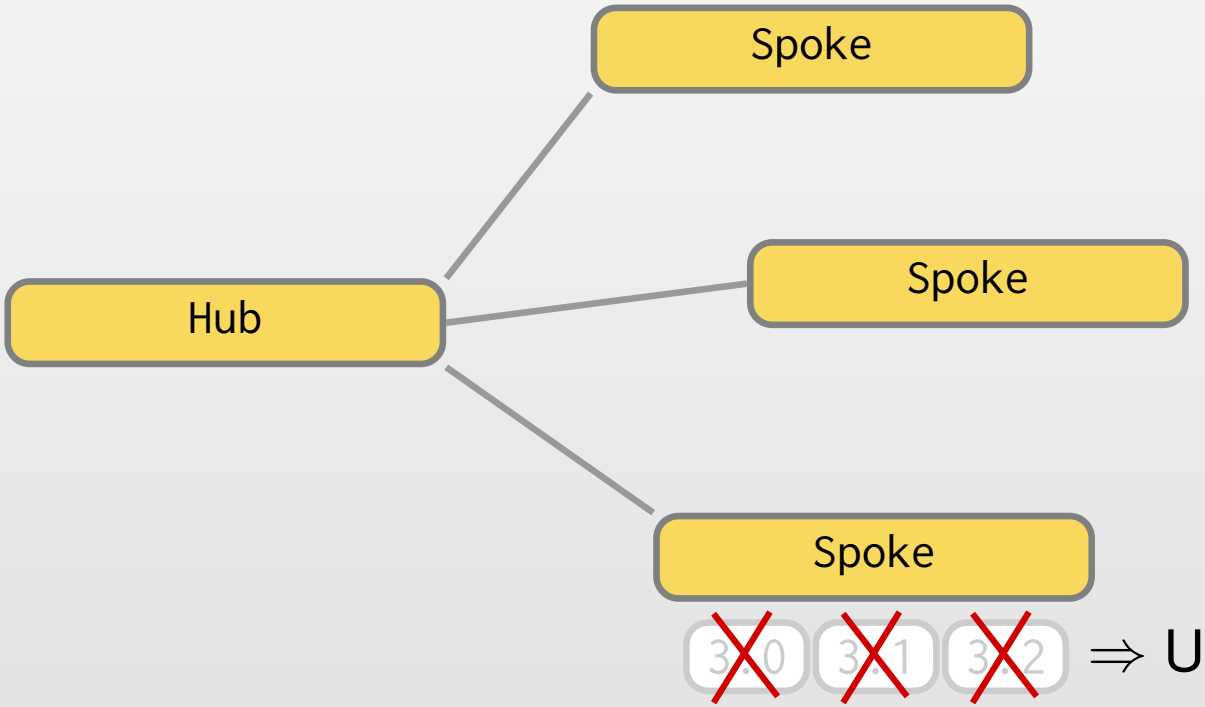
Built-in Package Management System

```
>>>   sys.path = [ ".", "/-@-/pkg.builtin", "/opt/default" ]
>>>   import pycuda
```



Built-in Package Management System

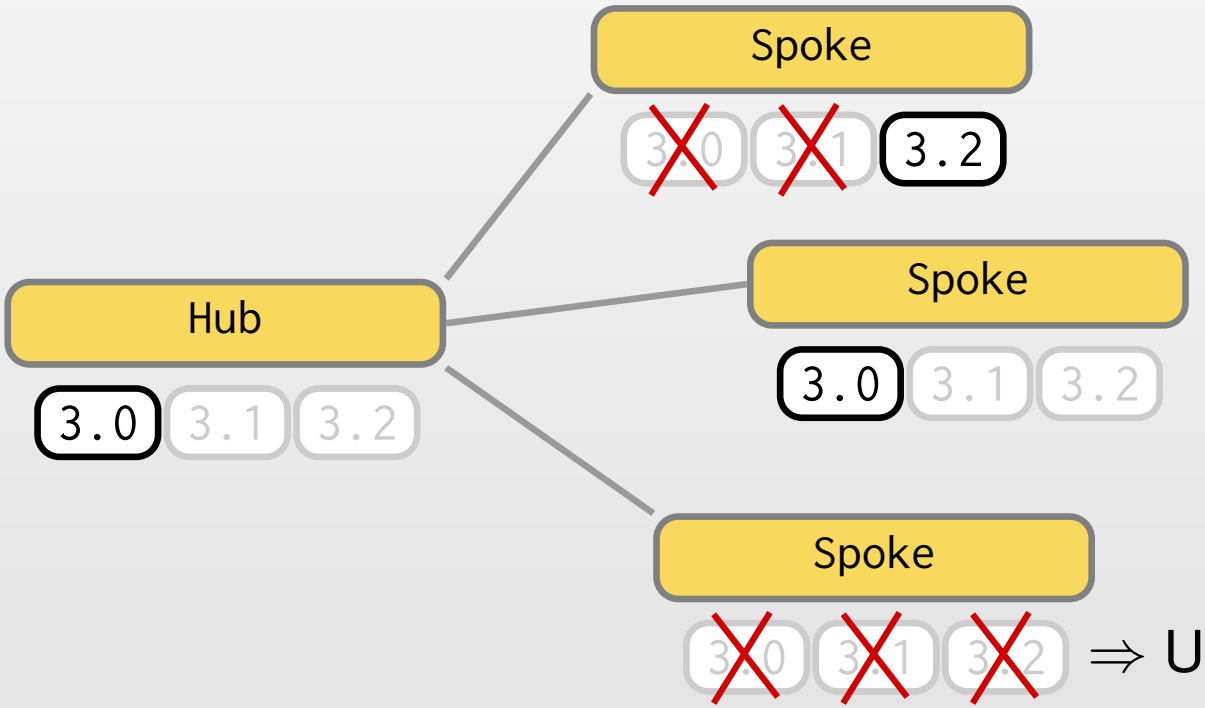
```
>>>  sys.path = [ ".", "/-@-/pkg.builtin", "/opt/default" ]
>>>  import pycuda
```



$\Rightarrow$ Uncatchable exception

Built-in Package Management System

```
>>>   sys.path = [ ".", "/-@-/pkg.builtin", "/opt/default" ]
>>>   import pycuda
```

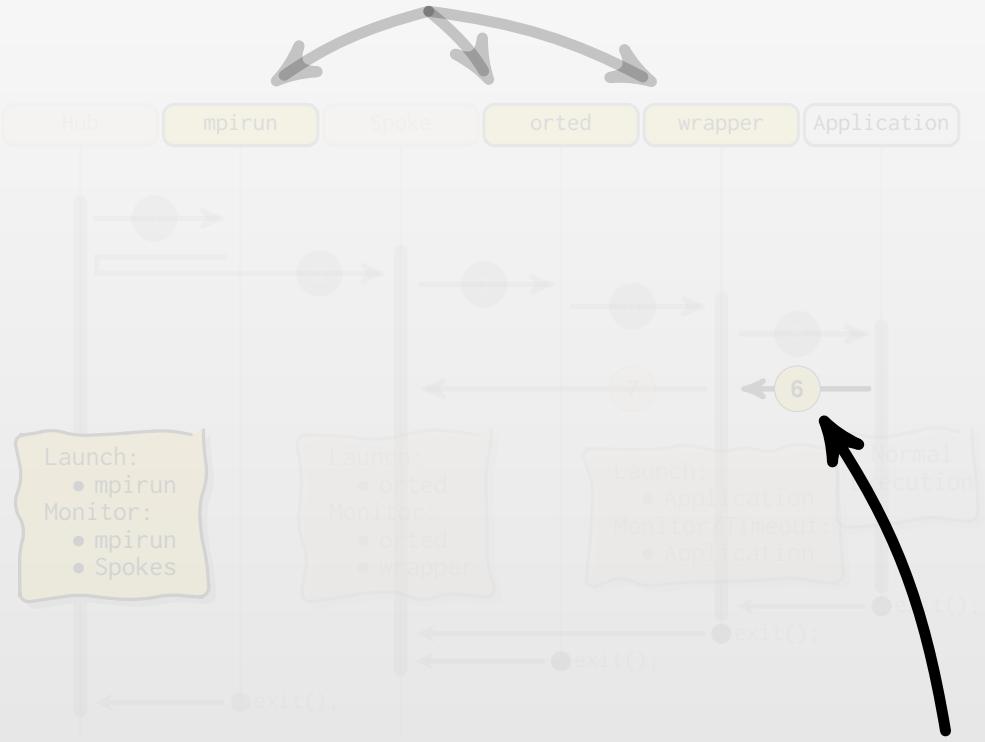


⇒ Uncatchable exception

Anatomy: Breakdown

Built-in Package Management System

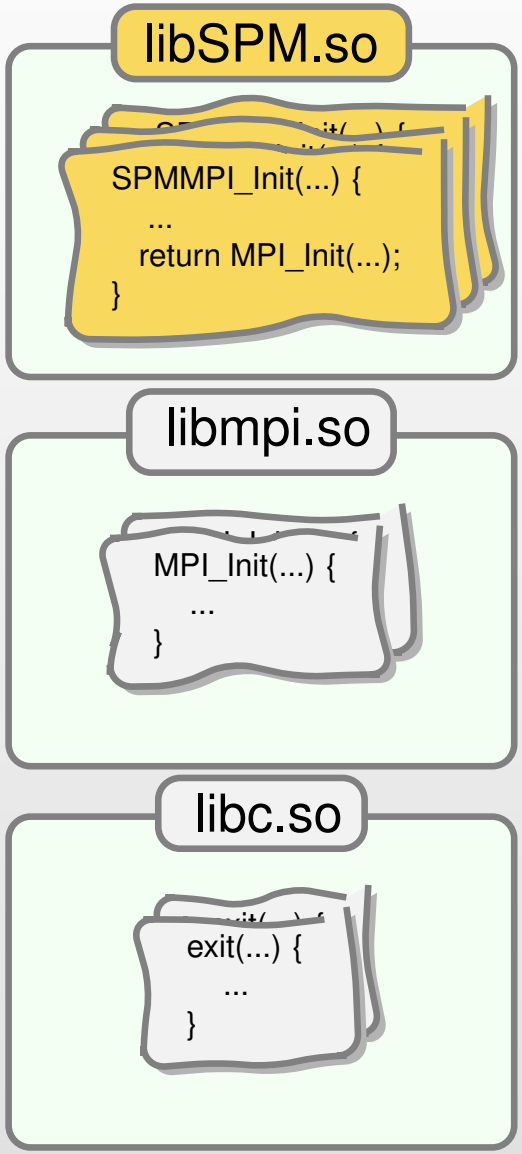
- Selectively change default OpenMPI env



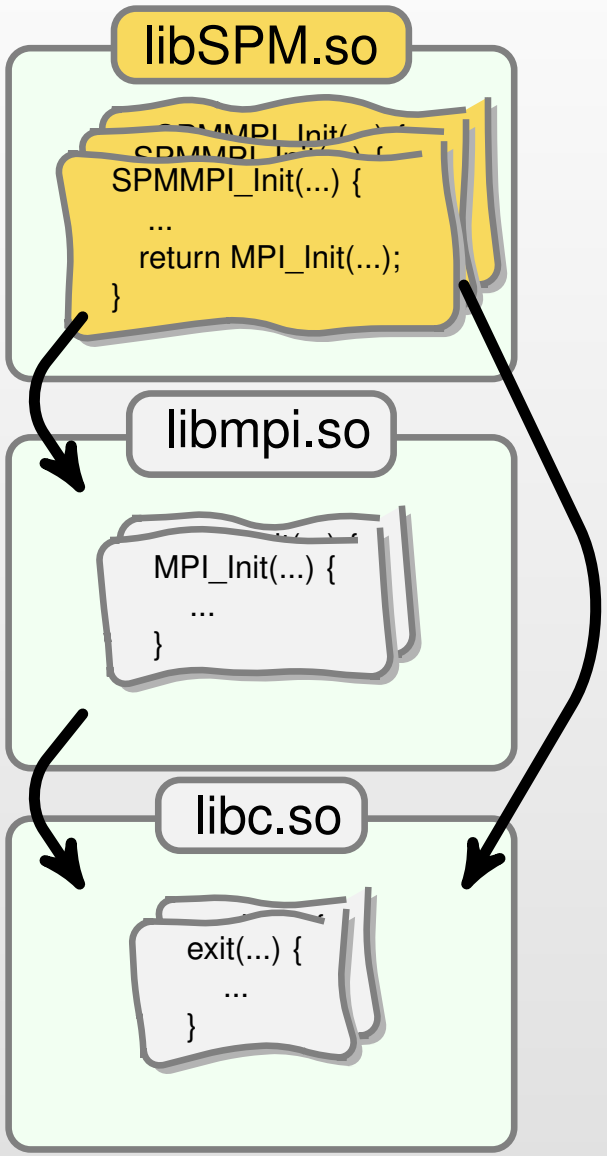
Redirection of library calls

- Augment libmpi.so, libc.so ... with libSPM.so

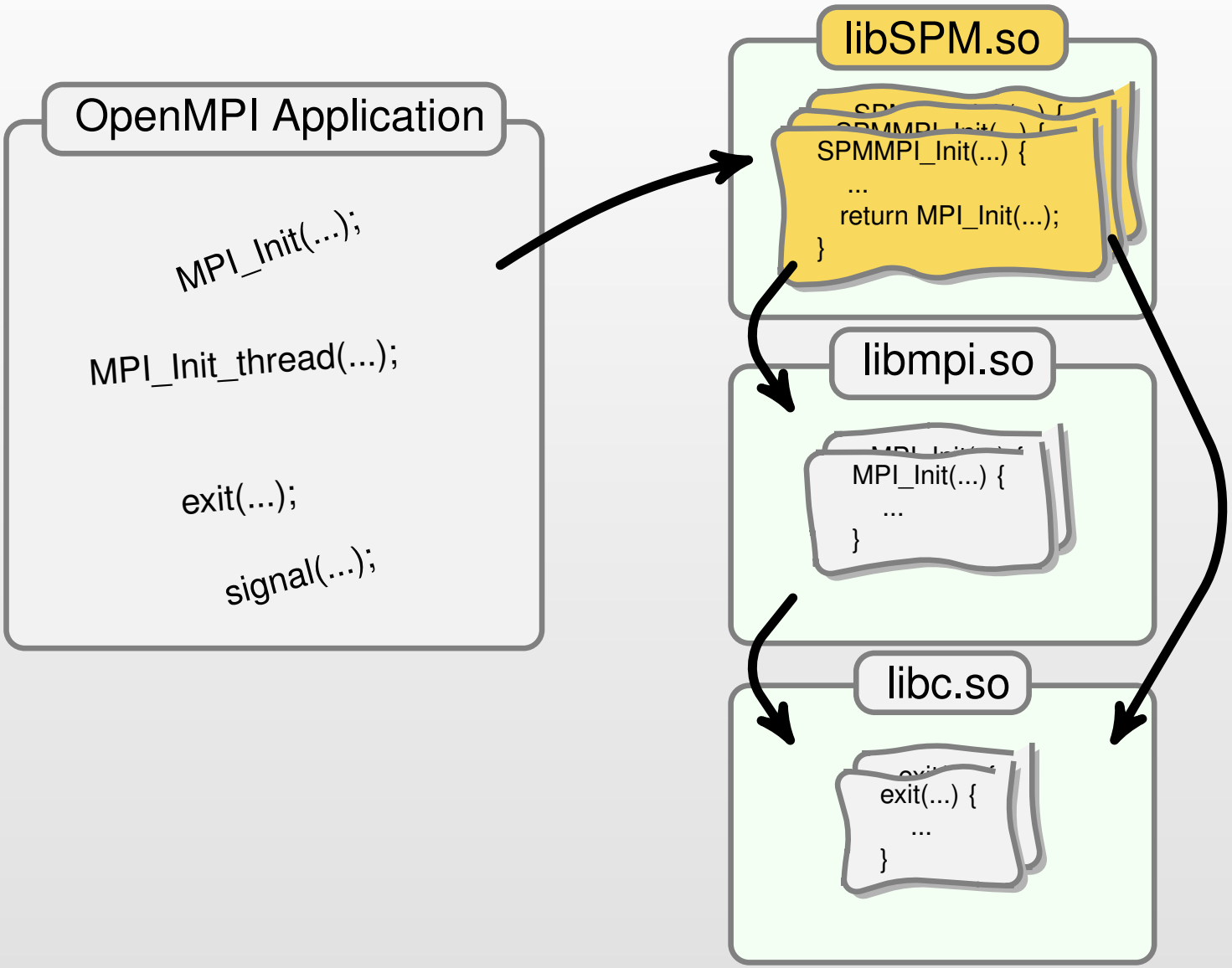
Redirecting Shared Library Calls



Redirecting Shared Library Calls



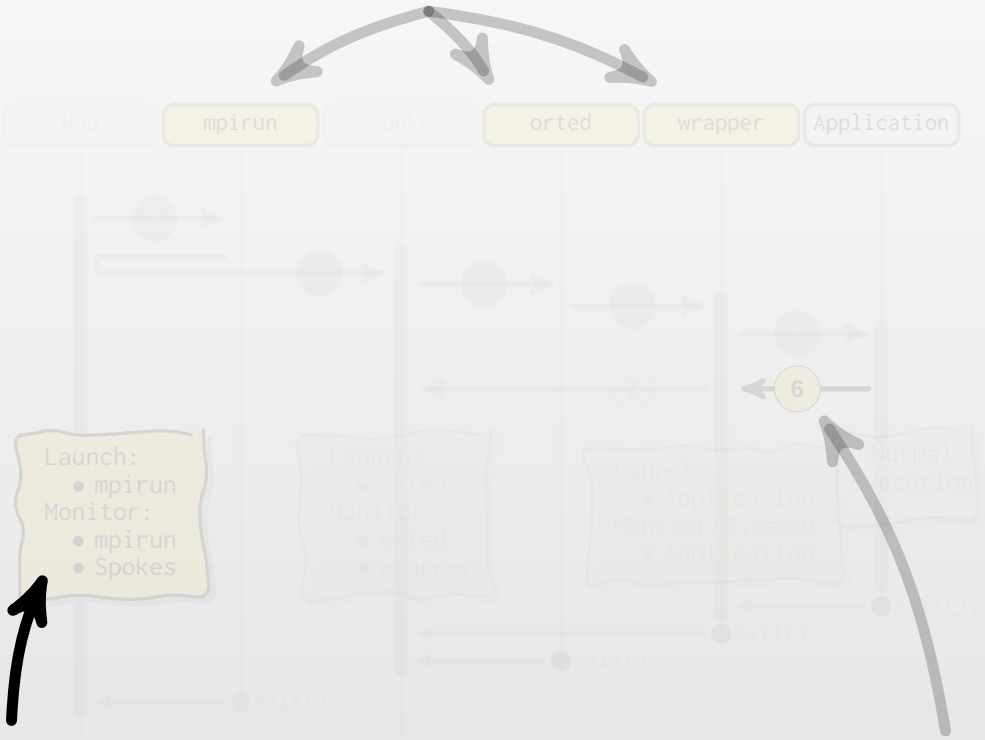
Redirecting Shared Library Calls



Anatomy: Breakdown

Built-in Package Management System

- Selectively change default OpenMPI env



Second Parallel Capability

- $\sim$ 60-line python script
- Authored by developer

Redirection of library calls

- Augment libmpi.so, libc.so ... with libSPM.so

Second Parallel Capability

Declaration + Definition of Pclosure

```
@spm.util.dassert(predicateCb = spm.sys.sstat.amOffline)
@spm.util.dassert(predicateCb = spm.sys.pstat.amHub)
def __init():
  return spm.pclosure.macro.papply.template.openMPI.\
         policyA.defun(signature = 'signature::Hub',
                       stage1Cb  = __taskStat,
                      );

__pc = __init();
```

Second Parallel Capability

Population + Invocation of Pclosure

```
@spm.util.dassert(predicateCb = spm.sys.sstat.amOffline)
@spm.util.dassert(predicateCb = spm.sys.pstat.amHub)
def main(pool,
         taskApiArgs,
         taskTimeout):
  # Initialize 'stage0'.
  __pc.stage0.init.main(typedef = ...);
  hdl = __pc.stage0.payload.tie();
  # Populate the template task
  hdl.spm.meta.label   = '***'; # Not interested.
  hdl.spm.meta.apiArgs = taskApiArgs;
  hdl.spm.meta.timeout = taskTimeout;
  # Invoke the pmanager
  __pc.stage0.event.manage(pool                    = pool,
                           nSpokesMin              = ...
                           nSpokesMax              = ...
                           timeoutWaitForSpokes    = ...
                           timeoutExecution        = ...
                           );
  return;
```

Second Parallel Capability

Typedef for Template Task

```
r"""
 task<template>      ::struct {
   # SPM component ...
   spm                ::struct {
     meta             ::struct {
       label          ::scalar<stringSnippet> = deferred;
       apiArgs        ::dict<string,mixed>    = deferred;
       timeout        ::scalar<timeout>       = deferred;
     };

     core             ::struct {
       relaunchPre    ::scalar<bool>          = None;
       relaunchPost   ::scalar<bool>          = None;
       nameHost       ::scalar<auto>          = None;
       whoAmI         ::scalar<auto>          = None;
     };

     stat             ::struct {
       exception      ::scalar<auto>          = None;
       returnValue    ::scalar<record>        = None;
     };
   };
   # non-SPM component ...
 };
 """
```

Second Parallel Capability

Callback for Status Reports

```
@spm.util.dassert(predicateCb = spm.sys.sstat.amOnline)
@spm.util.dassert(predicateCb = spm.sys.pstat.amHub)
def __taskStat(pc):
  try:
    hdl         = pc.stage1.payload.tie();
    returnValue = hdl.spm.stat.returnValue;
    if (returnValue.Has(attr = 'stdOut')):
      print("\tstdOut    : %s", returnValue.stdOut);
    if (returnValue.Has(attr = 'stdErr')):
      print("\tstdErr    : %s", returnValue.stdErr);
    if (returnValue.Has(attr = 'stdOutErr')):
      print("\tstdOutErr: %s", returnValue.stdOutErr);
  except (SPMTaskDropped,
          SPMTaskLoad,
          SPMTaskEval,
          ), (hdl,):
    pass;

  return (pc.stage1.event.done(),
          None,
          )[-1];
```

SPM.Python Session

```
GNU/Linux [] spm.3.111116.trial.A.python
```

```
(Trial Edition)
Spm.Python 3.111116 / Python 2.4.4
[GCC 4.1.2 (64 bit) on linux2]

NOTE
>>>      Trial period ends at          <<<
>>> 24:00 hrs (Pacific Standard Time)  <<<
>>>      December 29, 2011             <<<

Type "help", "copyright", "credits", "license" or "spm.Api()" for more informati
Type "spm.DemoExtract(dirname = '...')" to extract demo scripts.

Please visit www.mbsciences.com for the latest and growing
collection of scripts and technical briefs classified in terms of
parallel management patterns.
```

```
>>> import pool
>>> import demo
>>> import os;
>>> taskApiArgs = \
      dict(app         = os.getcwd() + '/hello_world',
           appOptions = "-prefix='app'",
          );
>>> taskTimeout = spm.util.timeout.after(seconds = 10);
>>> demo.main(pool         = pool.intraAll(),
              taskApiArgs = taskApiArgs,
              taskTimeout = taskTimeout)
  #: MetaStatus (hub): Waiting - ForSpokes ...
  #: MetaStatus (hub): Tasks   - Eval
        app => 0
        app => 1
  #: MetaStatus (hub): Tasks   - EvalDone
>>> demo.main(pool         = pool.intraOnePerServer(),
              taskApiArgs = taskApiArgs,
              taskTimeout = taskTimeout)
  #: MetaStatus (hub): Waiting - ForSpokes ...
  #: MetaStatus (hub): Tasks   - Eval
  #: MetaStatus (hub): Tasks   - EvalDone
>>> exit()
GNU/Linux []
```

Conclusion

Prototyping should be frictionless

↓

Must use original OpenMPI application

- original source code
- original binary

↓

Original OpenMPI application must inherit support for:

- fault tolerance
- timeout
- detecting deadlocks

↓

GNU/Linux []

spm.python ...
~~mpirun~~ ...

./hello_world -prefix "api"

Conclusion (Cont'd)

http://www.mbasciences.com

{ SPM.Python distribution
Technical Briefs
Parallel Management Patterns }

Elementary Parallel Primitives

- Clone
  - Once
  - Repeat
- Partition
  - DAG
  - List
- PartitionAggregate
  - Centralized
  - Decentralized

HPC Parallel Primitives

- Partition
  - Grid/OpenMPI

Limited Beta
Nov 30

;

Data / Graph Parallel Primitives

- Partition
  - Data Flow
  - Graph

Stanford U
Dec 6

;