Composable Asynchrony

Suresh Jagannathan
LaME-O-in-Waiting

PURDUE
UNIVERSITY

Context

- Synchronous communication

safety

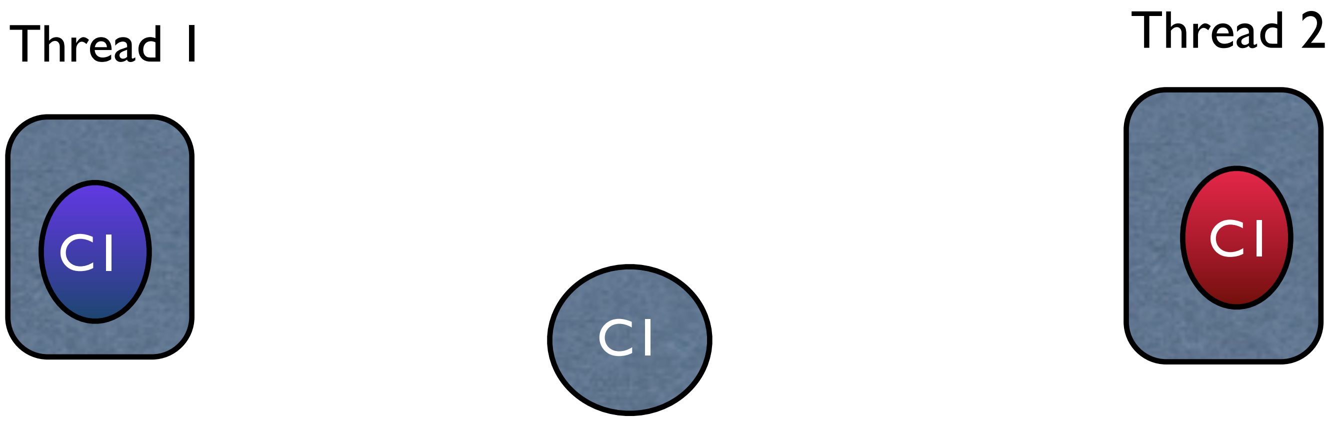
  - ★ Easy to reason about
  - ★ Convenient to build composable communication protocols
- Asynchronous communication
  - ★ Harder to reason about
    - ✦ stack ripping to express callbacks

performance

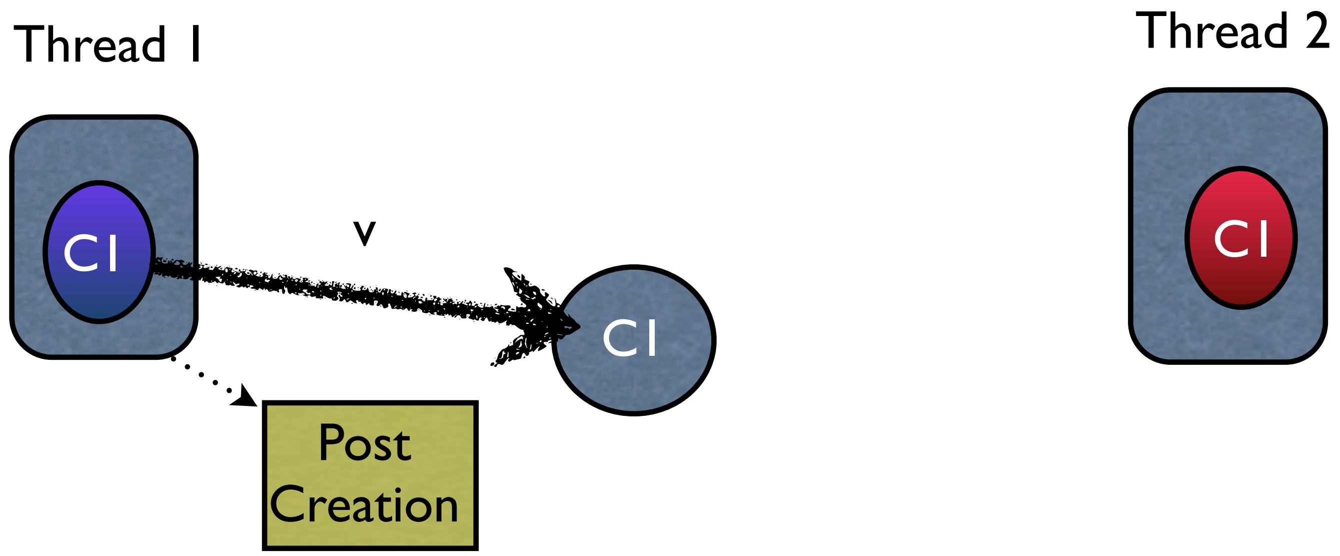
  - ★ Added expressivity
    - ✦ callbacks executed only when communication action completed
  - ★ Not straightforward to see how we might compose different asynchronous actions
- Challenge:
  - ★ *Adding and reasoning about asynchrony shouldn't compromise ability to build composable protocols*

Anatomy of an Asynchronous Action

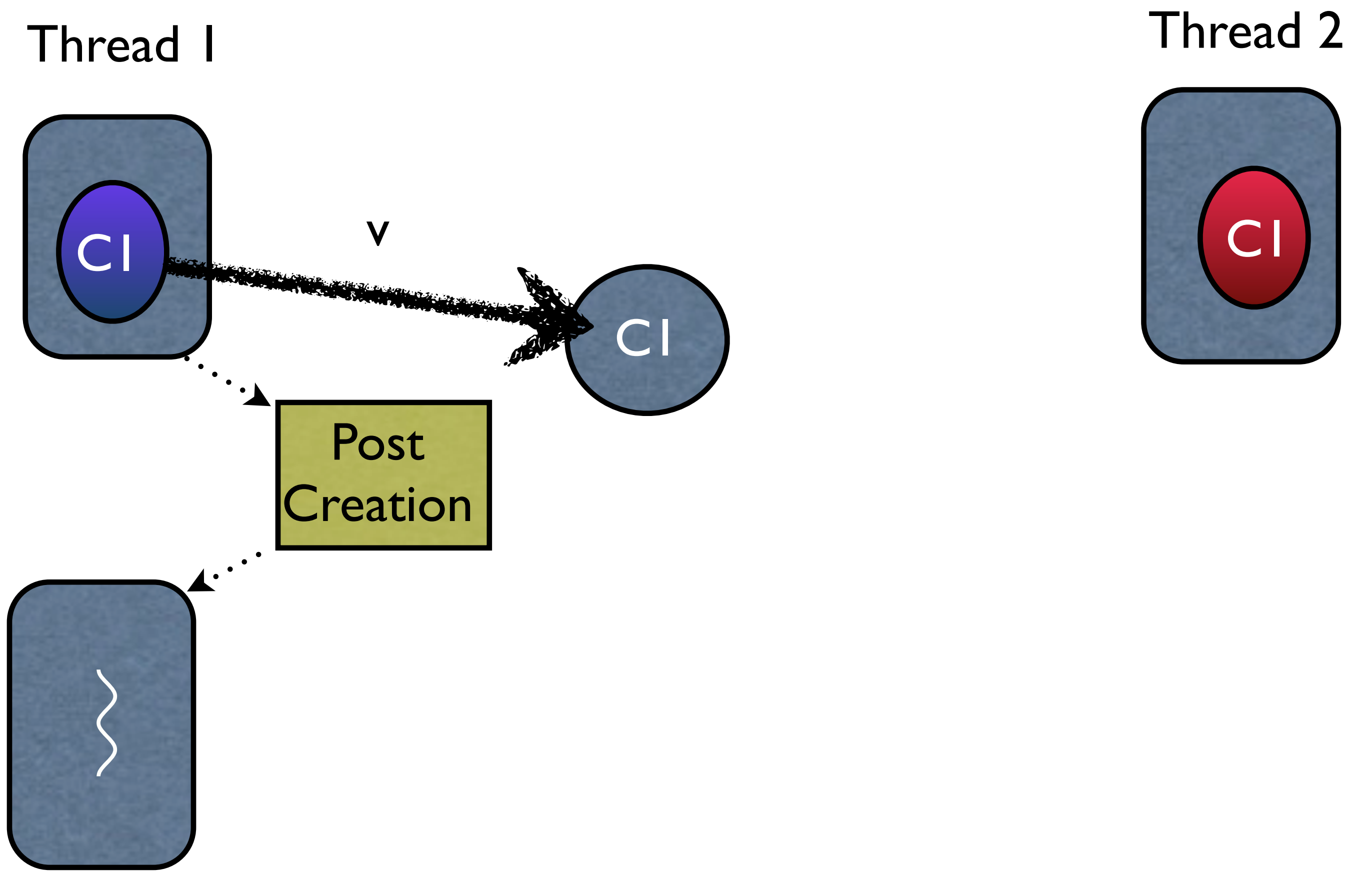
Anatomy of an Asynchronous Action



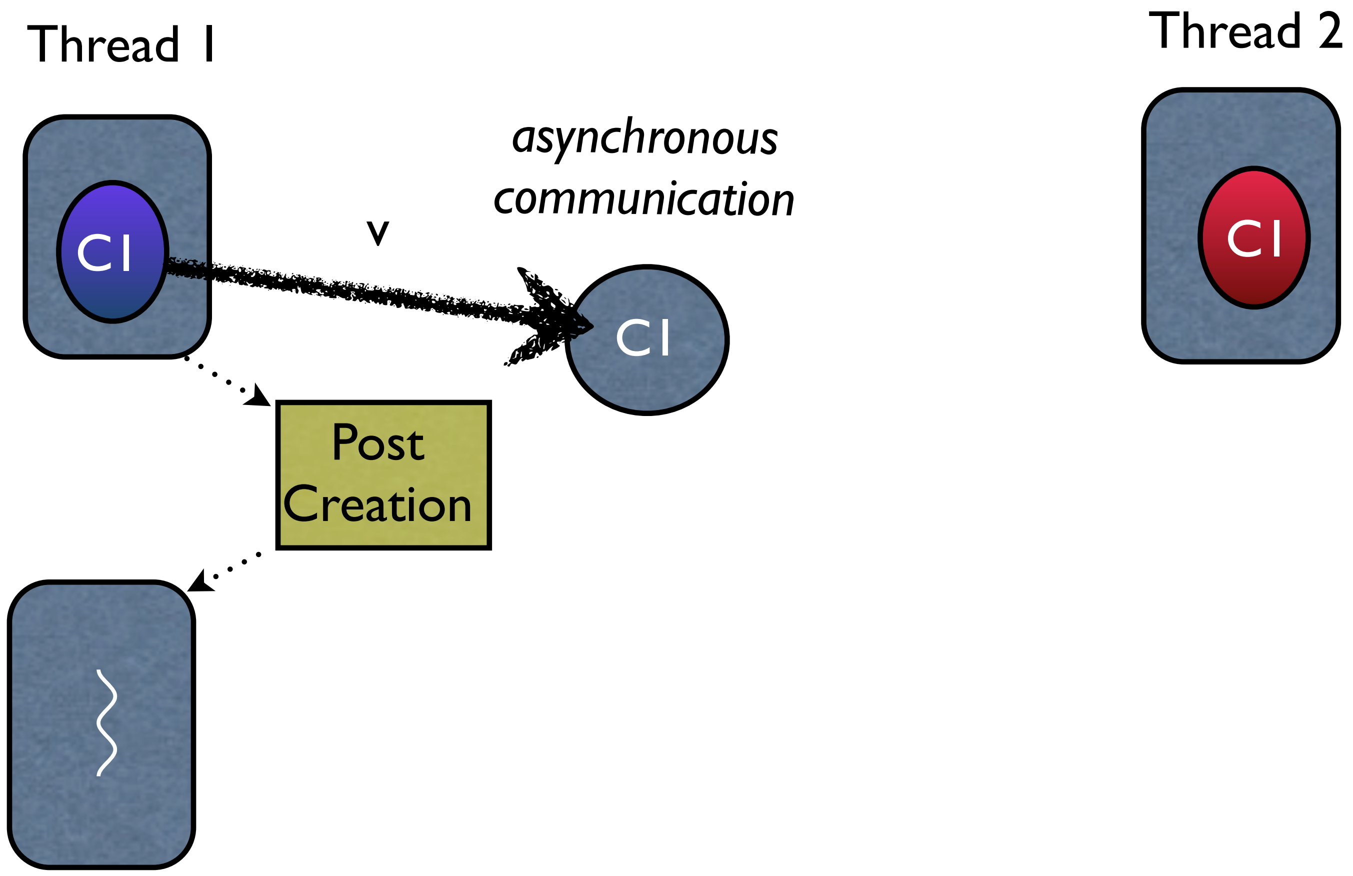
Anatomy of an Asynchronous Action



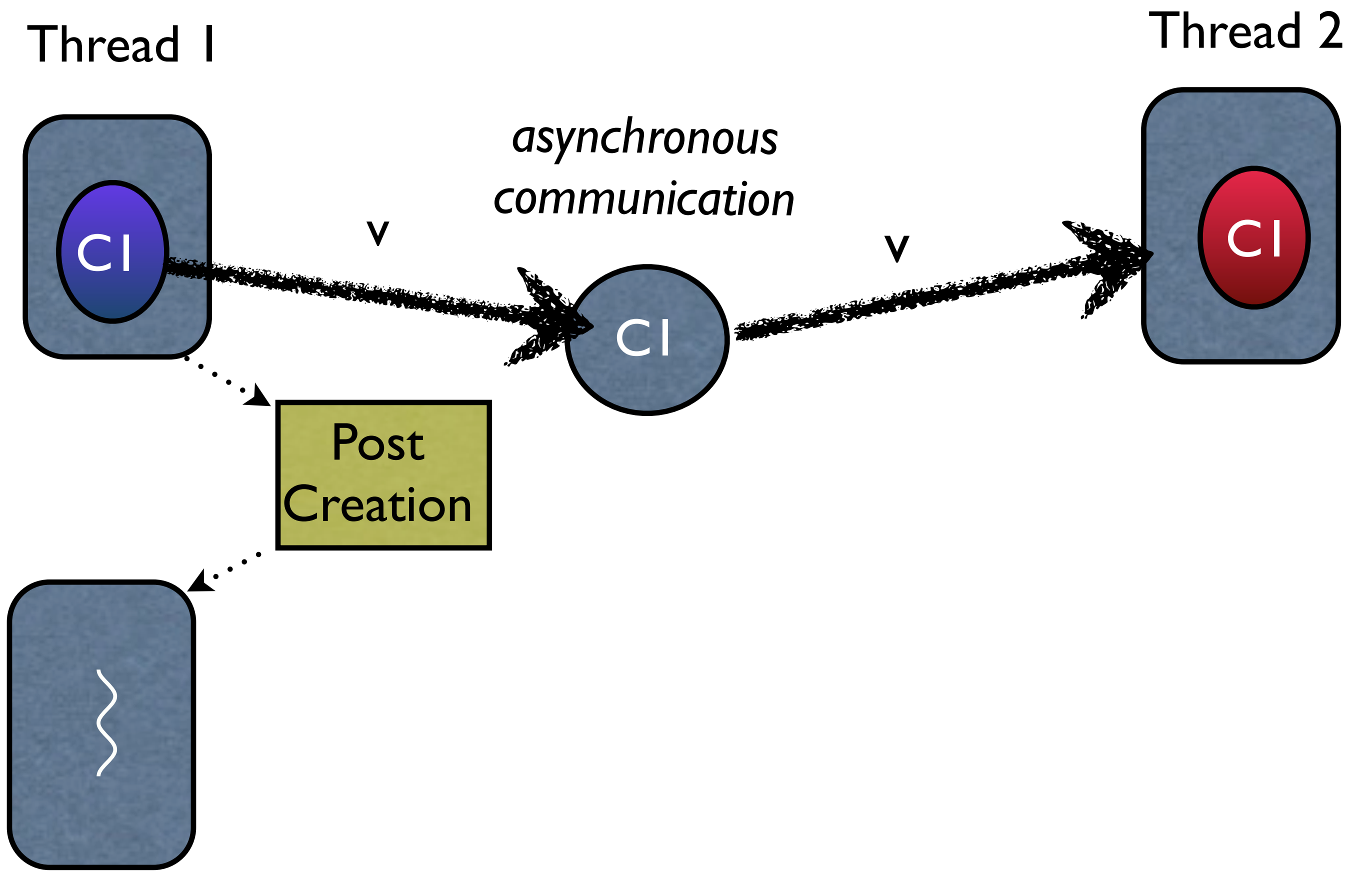
Anatomy of an Asynchronous Action



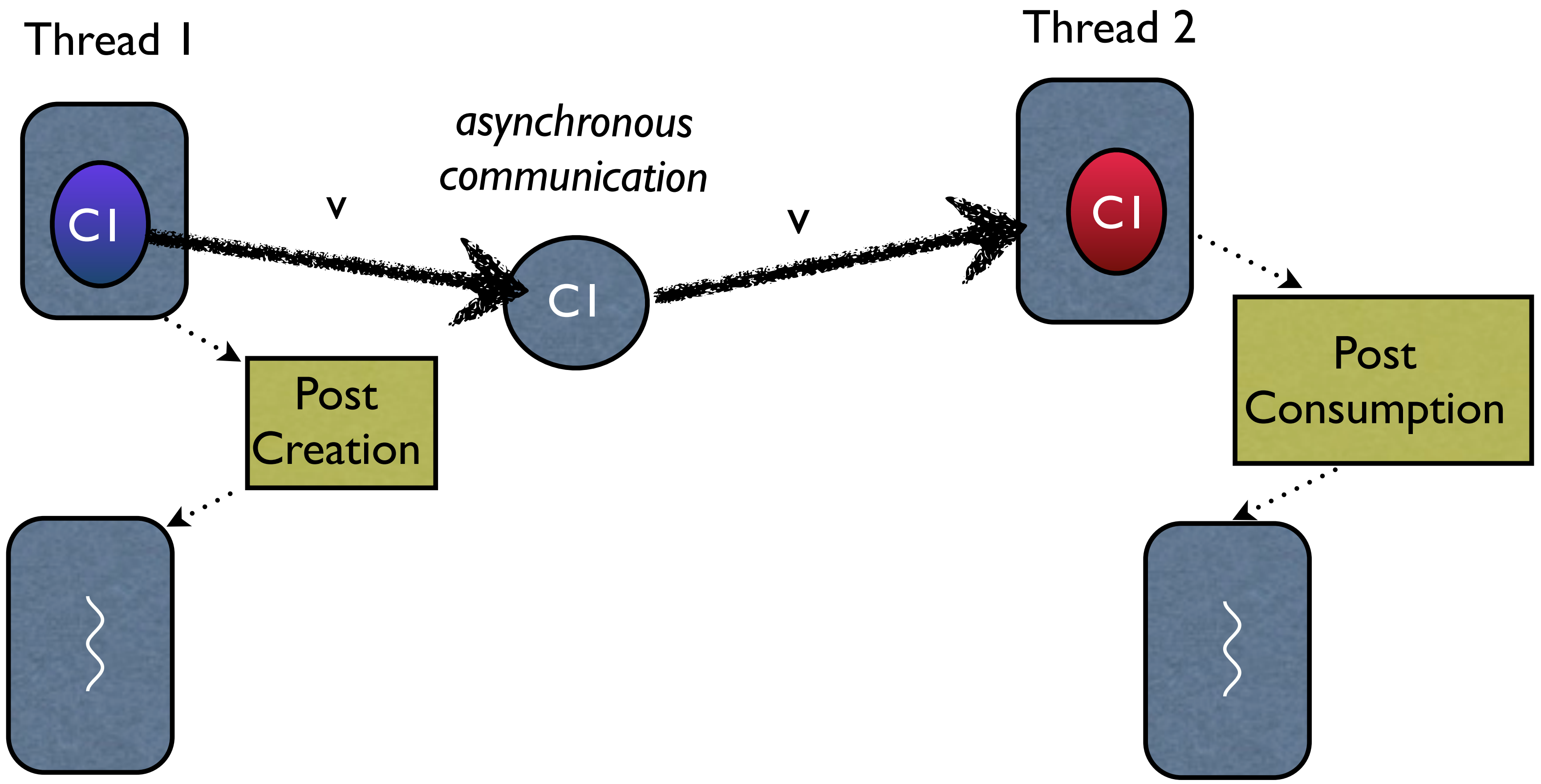
Anatomy of an Asynchronous Action



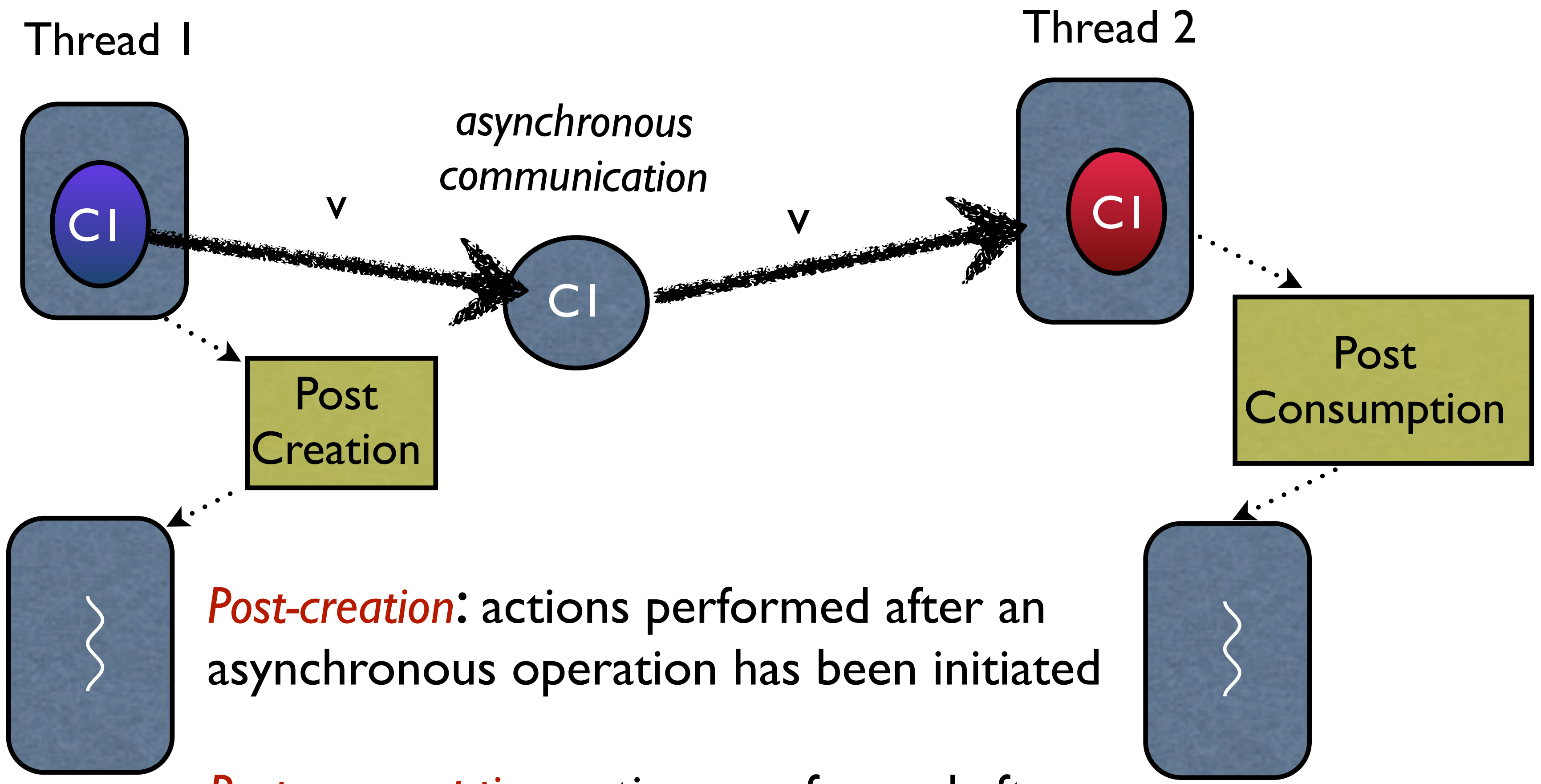
Anatomy of an Asynchronous Action



Anatomy of an Asynchronous Action



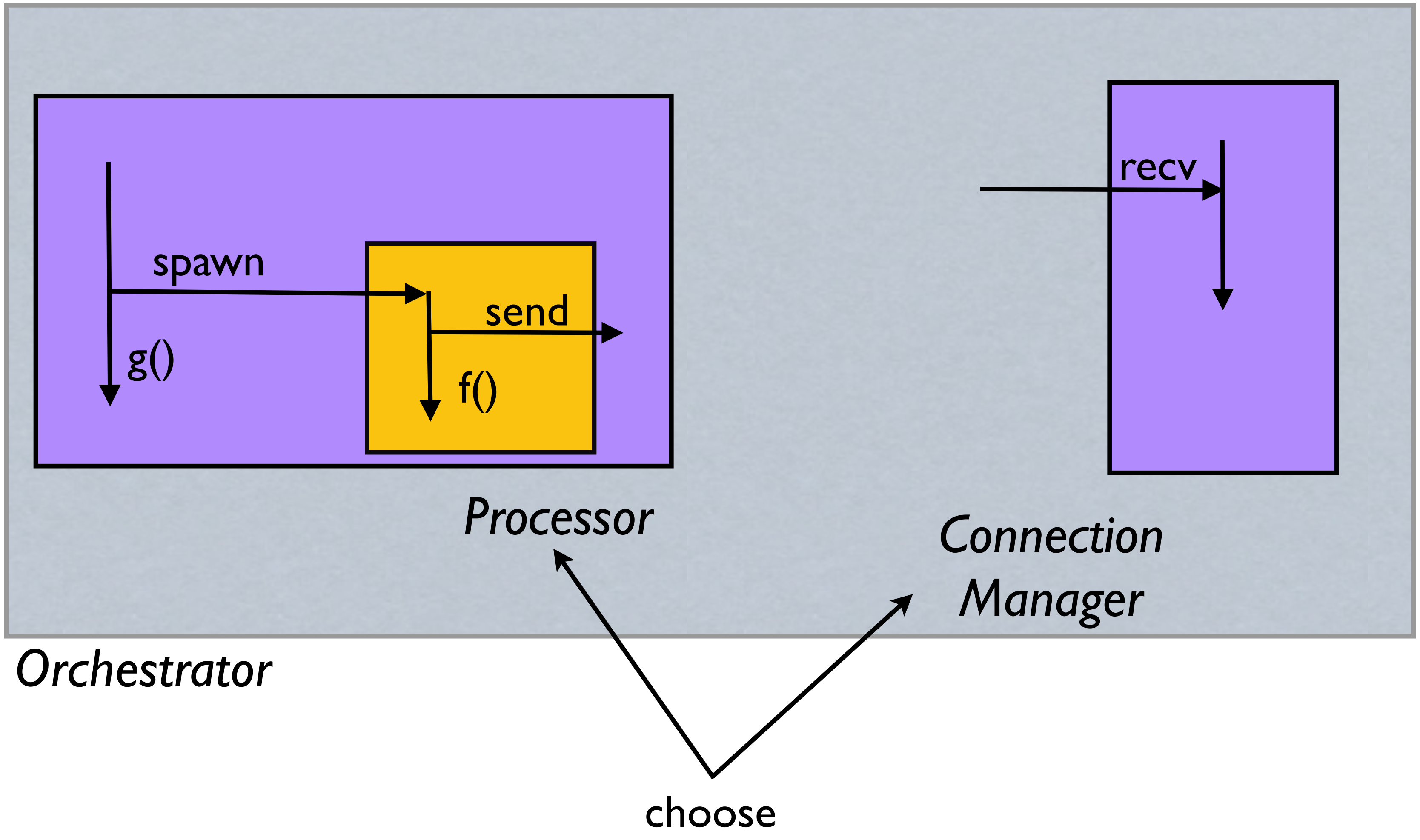
Anatomy of an Asynchronous Action



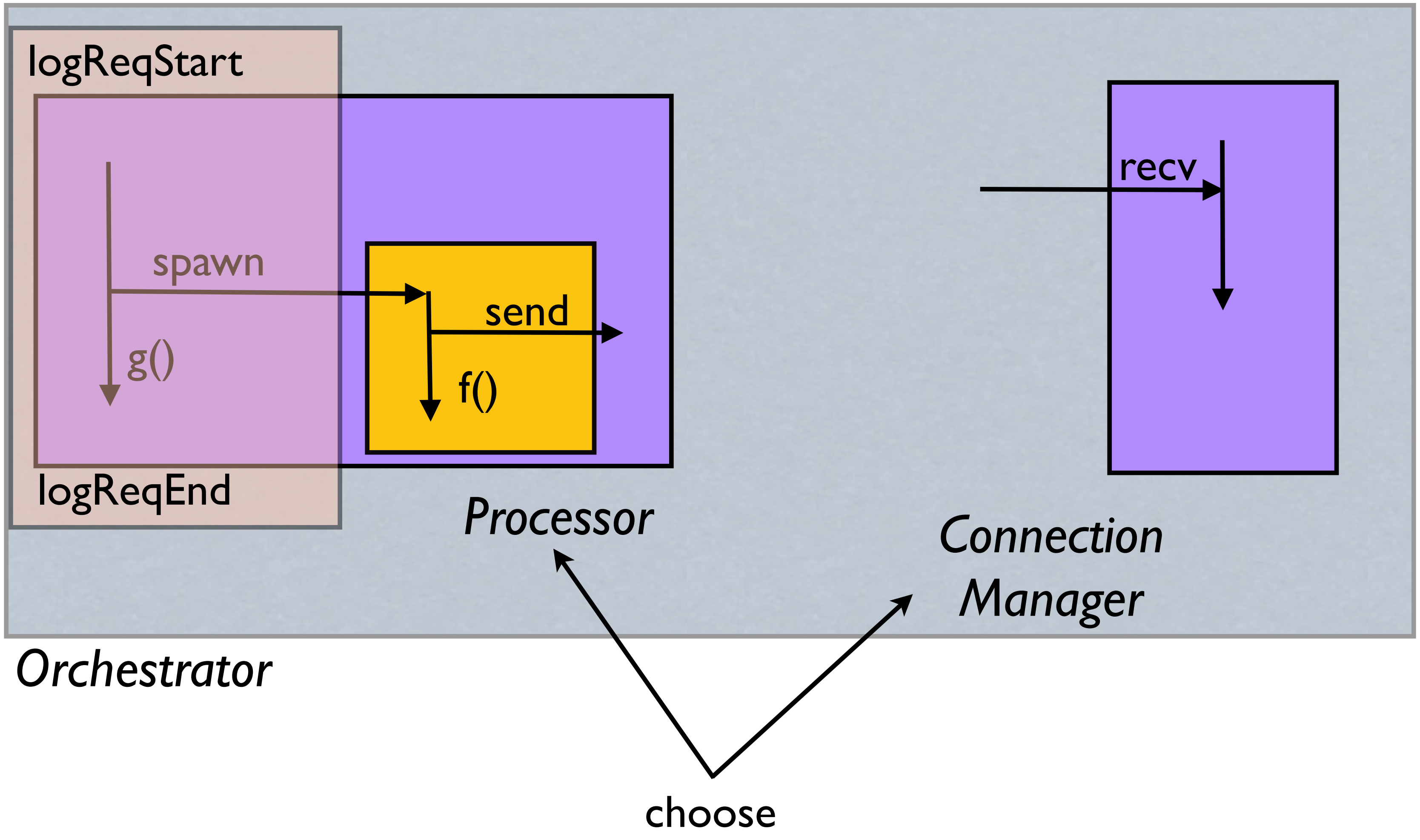
Post-creation: actions performed after an asynchronous operation has been initiated

Post-consumption: actions performed after an asynchronous operation has completed

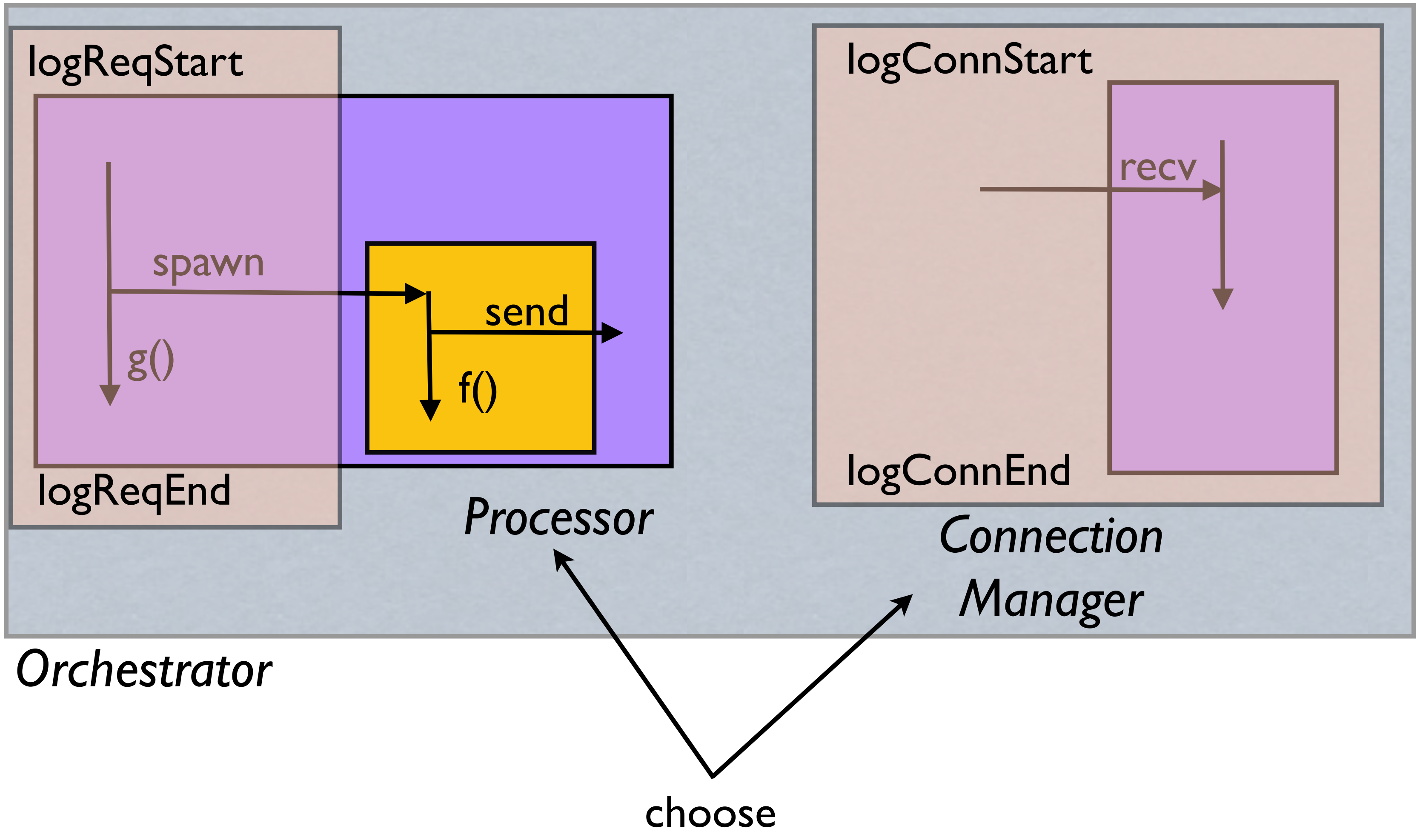
Example


spawn
send
g()
f()
recv
Processor
Connection Manager
Orchestrator
choose

Example


logReqStart
spawn
g()
send
f()
logReqEnd
Processor
recv
Connection Manager
Orchestrator
choose

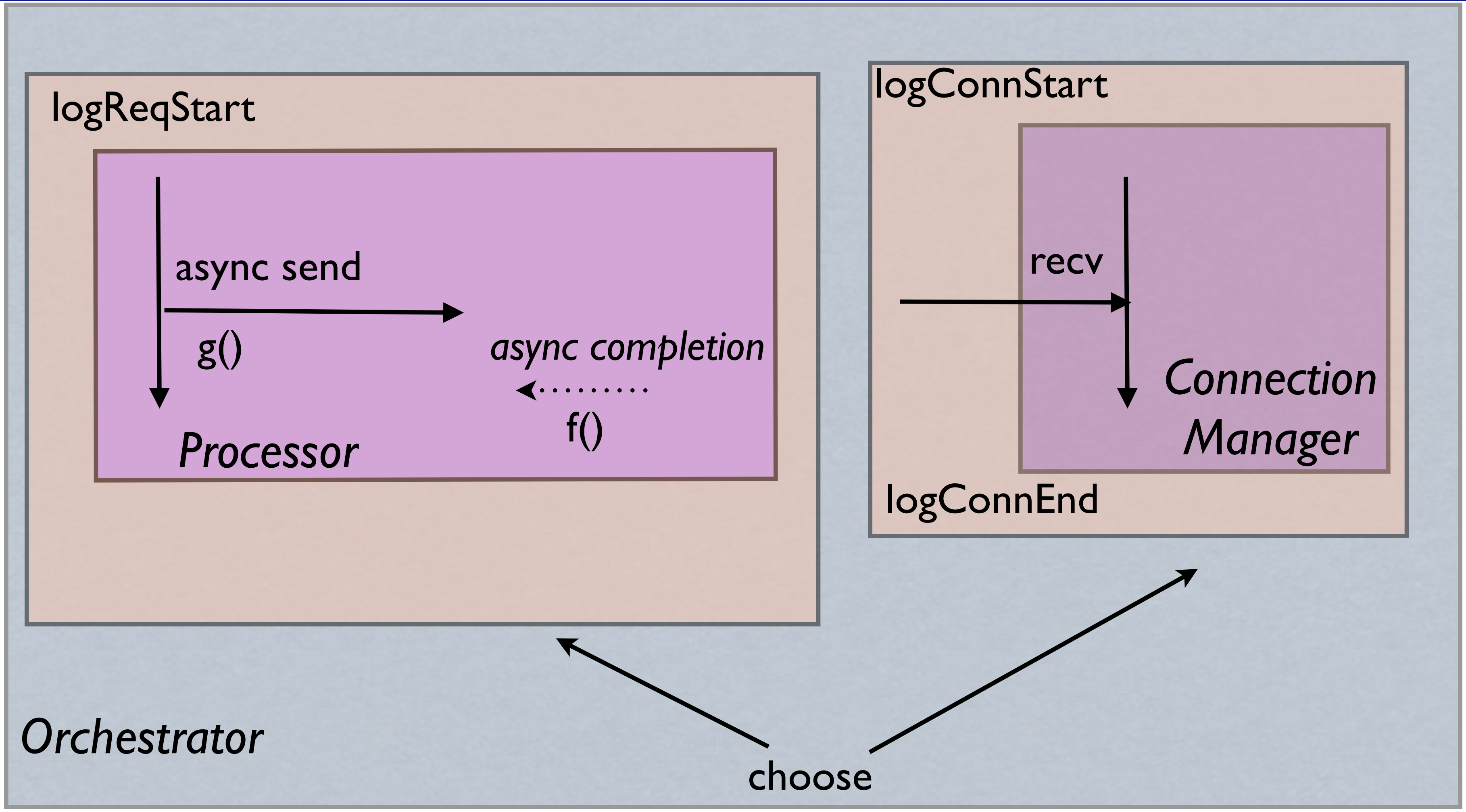
Example


logReqStart
spawn
g()
send
f()
logReqEnd
Processor
Orchestrator
logConnStart
recv
logConnEnd
Connection Manager
choose

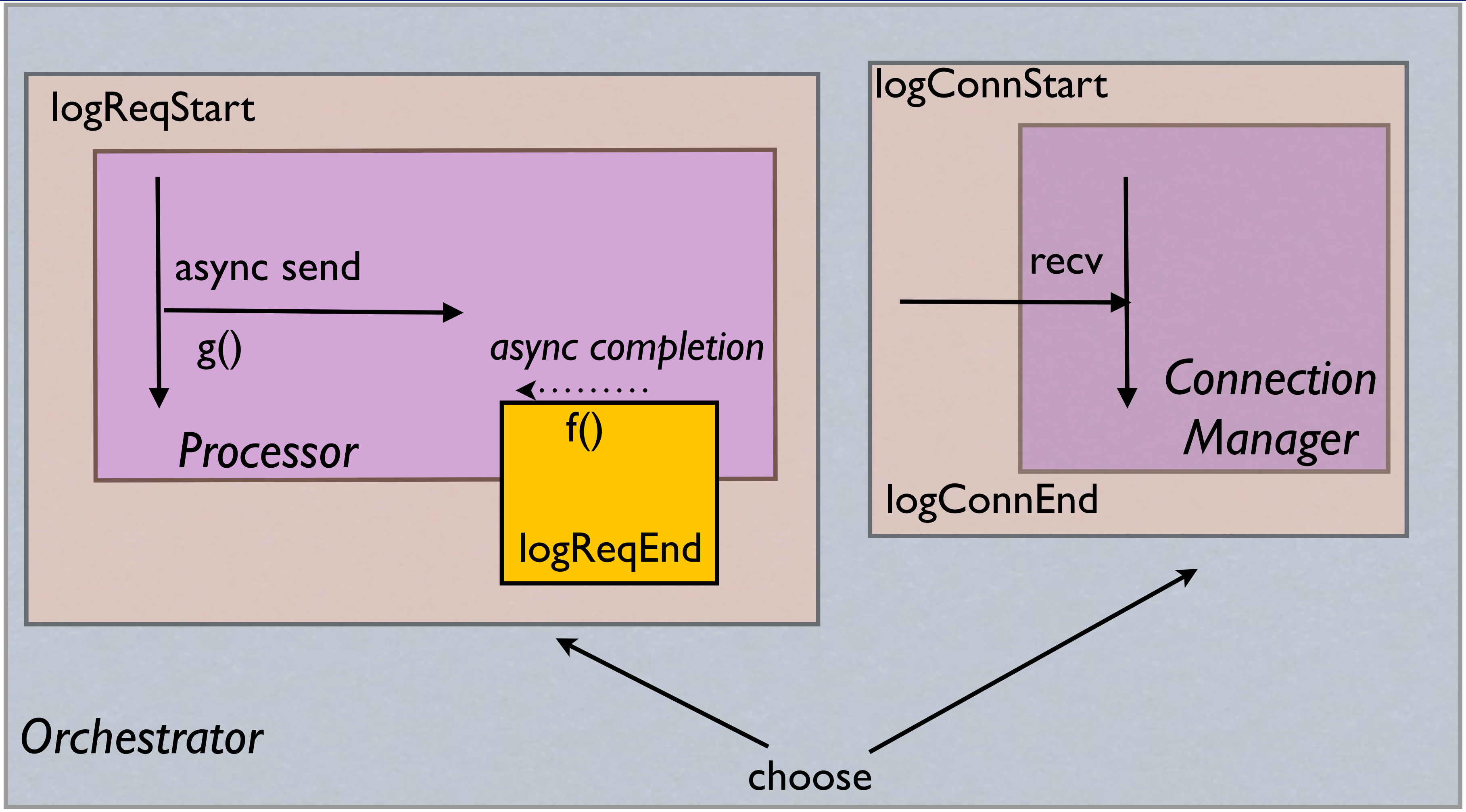
The Problem

- *Dichotomy in language abstractions*
  - ★ Asynchrony fundamentally expressed using distinct units of control
    - ✦ either continuations (tasks) or threads
  - ★ But, composability achieved through abstractions that should be thread and continuation unaware

Example Revisited



Example Revisited



Composable Callbacks (ACML)

Synchronous first-class events $\longrightarrow$ *Aynchronous first-class events*

```
callbackEvt : (‘a, ‘c) AEvent * (‘c -> ‘b) ->
                (‘b Event, ‘c) AEvent
```

```
fun callbackEvt (ev, f) =
  let clocal = channel()
  in sWrap(aWrap(ev,
                 fn x => (aSync(aSendEvt(clocal,x)); x)),
           fn _ => wrap(recvEvt (clocal), f))
  end
```

Composable Callbacks (ACML)

Synchronous first-class events $\longrightarrow$ *Aynchronous first-class events*

```
callbackEvt : ('a, 'c) AEvent * ('c -> 'b) ->
              ('b Event, 'c) AEvent
```

```
fun callbackEvt (ev, f) =
  let c_local = channel()
  in sWrap(aWrap(ev,
                 fn x => (aSync(aSendEvt(c_local,x)); x)),
           fn _ => wrap(recvEvt (c_local), f))
  end
```

Defines a post-creation action

Composable Callbacks (ACML)

Synchronous first-class events $\longrightarrow$ *Aynchronous first-class events*

```
callbackEvt : ('a, 'c) AEvent * ('c -> 'b) ->
              ('b Event, 'c) AEvent
```

```
fun callbackEvt (ev, f) =
  let clocal = channel()
  in sWrap(aWrap(ev,
                fn x => (aSync(aSendEvt(clocal,x)); x)),
           fn _ => wrap(recvEvt (clocal), f))
  end
```

Defines a post-creation action

This action creates a new event that synchronously waits for a value on c_{local}, and invokes f (the callback) on that value

Composable Callbacks (ACML)

Synchronous first-class events $\longrightarrow$ *Aynchronous first-class events*

```
callbackEvt : ('a, 'c) AEvent * ('c -> 'b) ->
              ('b Event, 'c) AEvent
```

Defines a post-consumption action

```
fun callbackEvt (ev, f) =
  let clocal = channel()
  in sWrap(aWrap(ev,
                fn x => (aSync(aSendEvt(clocal,x)); x)),
          fn _ => wrap(recvEvt (clocal), f))
  end
```

Defines a post-creation action

This action creates a new event that synchronously waits for a value on c_{local}, and invokes f (the callback) on that value

Composable Callbacks (ACML)

Synchronous first-class events $\longrightarrow$ *Aynchronous first-class events*

```
callbackEvt : (‘a, ‘c) AEvent * (‘c -> ‘b) ->
                  (‘b Event, ‘c) AEvent
```

Defines a post-consumption action

```
fun callbackEvt (ev, f) =
  let clocal = channel()
  in sWrap(aWrap(ev,
                  fn x => (aSync(aSendEvt(clocal,x)); x)),
            fn _ => wrap(recvEvt (clocal), f))
  end
```

This action asynchronously sends the result of synchronizing on ev to c_{local}

Defines a post-creation action

This action creates a new event that synchronously waits for a value on c_{local}, and invokes f (the callback) on that value

Composable Callbacks (ACML)

Synchronous first-class events → *Aynchronous first-class events*

```
callbackEvt : (‘a, ‘c) AEvent * (‘c -> ‘b) ->
              (‘b Event, ‘c) AEvent
```

Defines a post-consumption action

Can compose result of ev with other event combinators

```
fun callbackEvt (ev, f) =
  let clocal = channel()
  in sWrap(aWrap(ev,
                  fn x => (aSync(aSendEvt(clocal,x)); x)),
           fn _ => wrap(recvEvt (clocal), f))
  end
```

This action asynchronously sends the result of synchronizing on ev to c_{local}

Defines a post-creation action

This action creates a new event that synchronously waits for a value on c_{local}, and invokes f (the callback) on that value

Instances

- ACML
- AC: Composable Asynchronous IO for Native Languages
  - ★ composable post-creation actions via async and do ... finish constructs
- Reagents
  - ★ combinators for extensible and composable concurrency abstractions
  - ★ post-commit actions
- Asynchronous workflows in F# and C#
  - ★ callbacks represented as continuations
- Monadic concurrency
  - ★ reactive programming
  - ★ interaction between applications and IO actions delivered asynchronously
- Asynchronous exceptions and kill-safe abstractions
- Asynchrony without stack-ripping
  - ★ lightweight event handlers
  - ★ Scala Actors, Kilim, Protothreads, Tame, Clarity, ...

Open Issues

- Composability and libraries
- Lightweight or heavyweight support for composability
- Interaction with legacy code
- Simplicity, modularity, orthogonality, ...
- Performance rationalization
- Interplay between synchrony and asynchrony
- Transformers (automatic)
- Typing
- Verification
- Memory model
- ...

reasoning