Solving problems with actors

Robey Pointer
<robeypointer@gmail.com>
http://github.com/robey

How I learned to love actors

How I learned to love actors

How I learned to love actors

How I learned to love actors

Writing a chat proxy server for phones

Chat proxy: the problem

- long-lived connections

Chat proxy: the problem

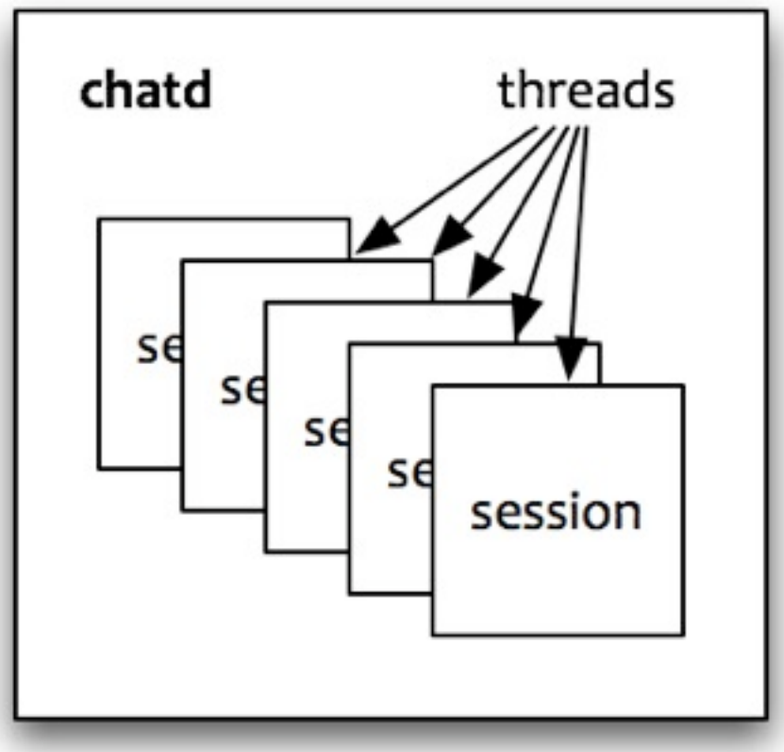
- long-lived connections
- many, many of them

Chat proxy: the problem

- long-lived connections
- many, many of them
- usually idle, with short bursts of traffic

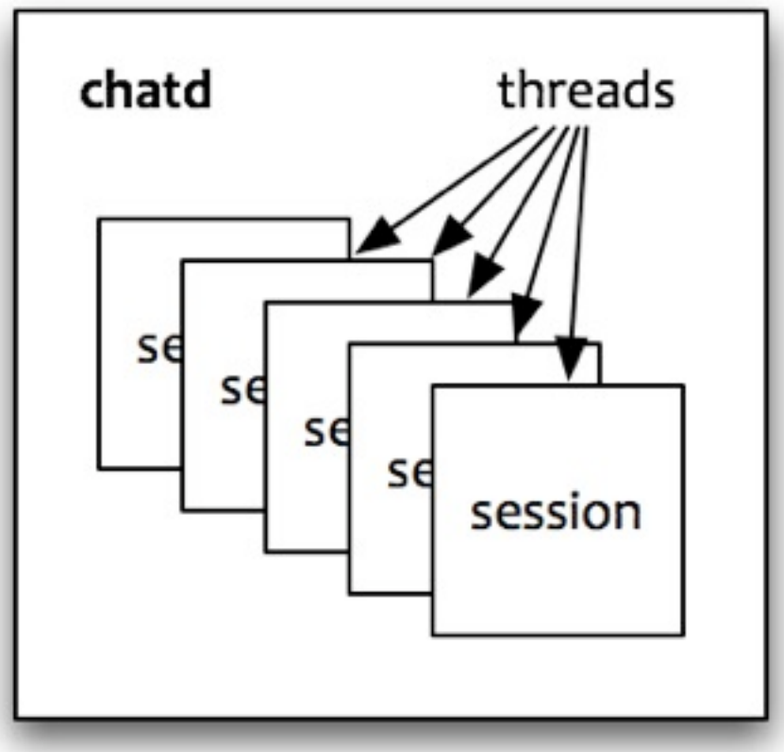
Chat proxy, take 1

- each session is a thread



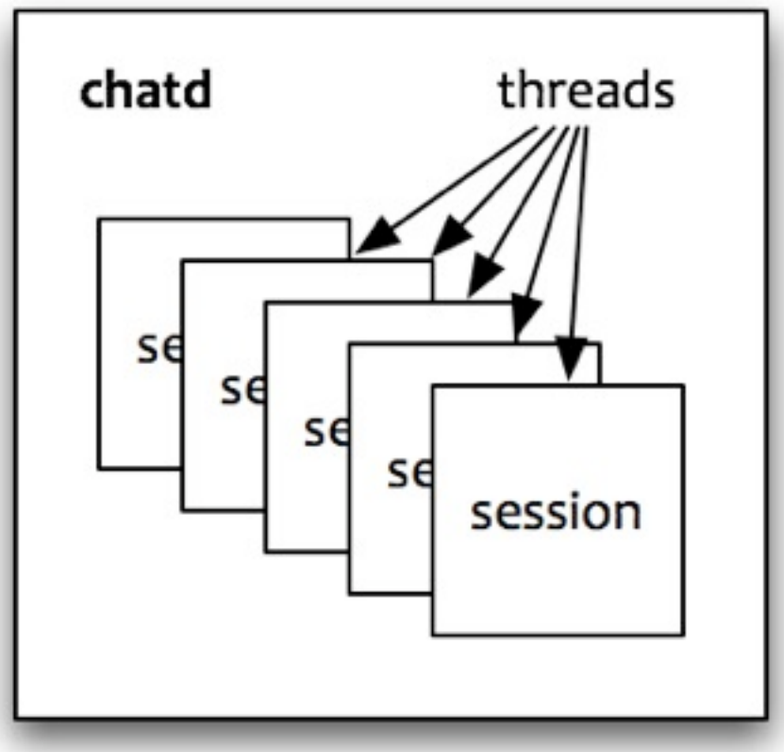
Chat proxy, take 1

- each session is a thread
- usually blocked on I/O (read)



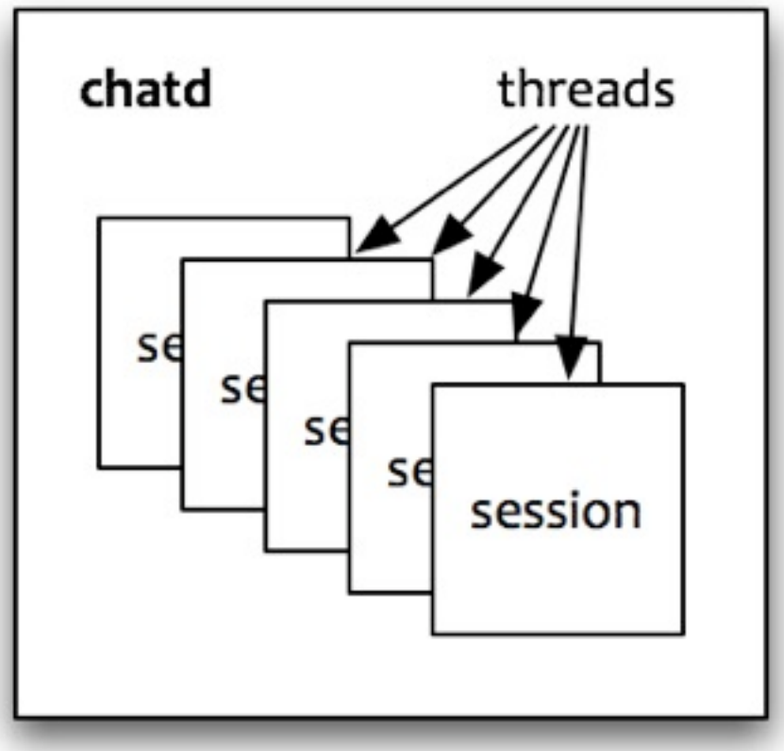
Chat proxy, take 1

- each session is a thread
- usually blocked on I/O (read)
- 5000 threads / sessions max in practice



Chat proxy, take 1

- each session is a thread
- usually blocked on I/O (read)
- 5000 threads / sessions max in practice
- easy to understand; scales poorly



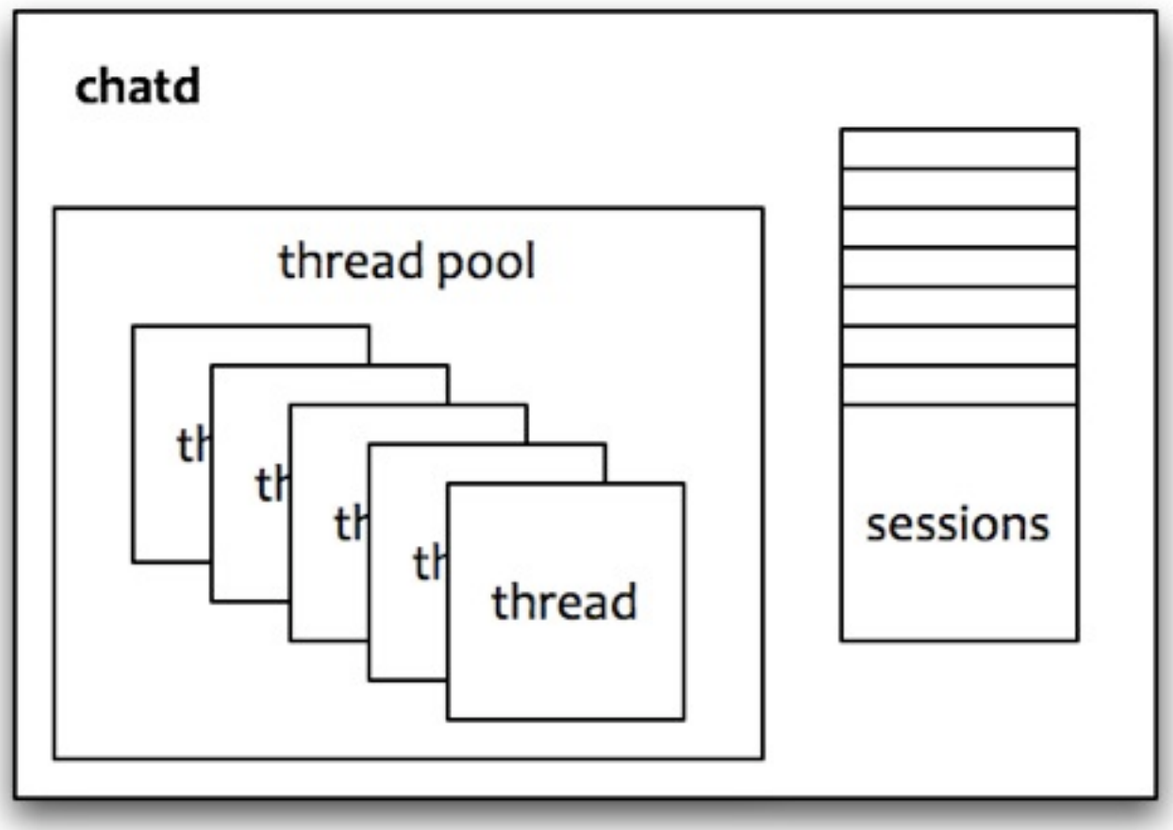
Chat proxy, take 1

- each session is a thread
- usually blocked on I/O
- 5000 threads / sessions max in practice
- easy to understand; scales poorly

```
class ChatSession {
  public void start(Socket s) {
    // sequential code...
  }
}
```

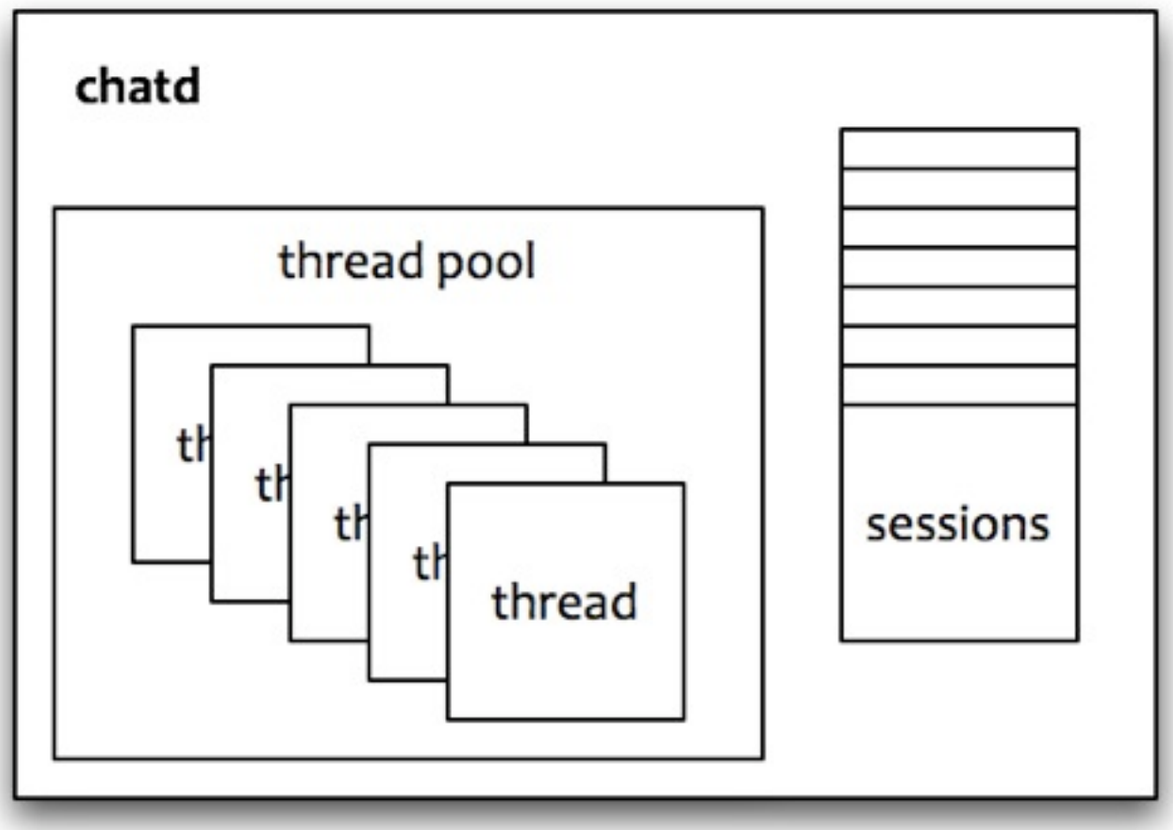
Chat proxy, take 2

- thread pool and async I/O



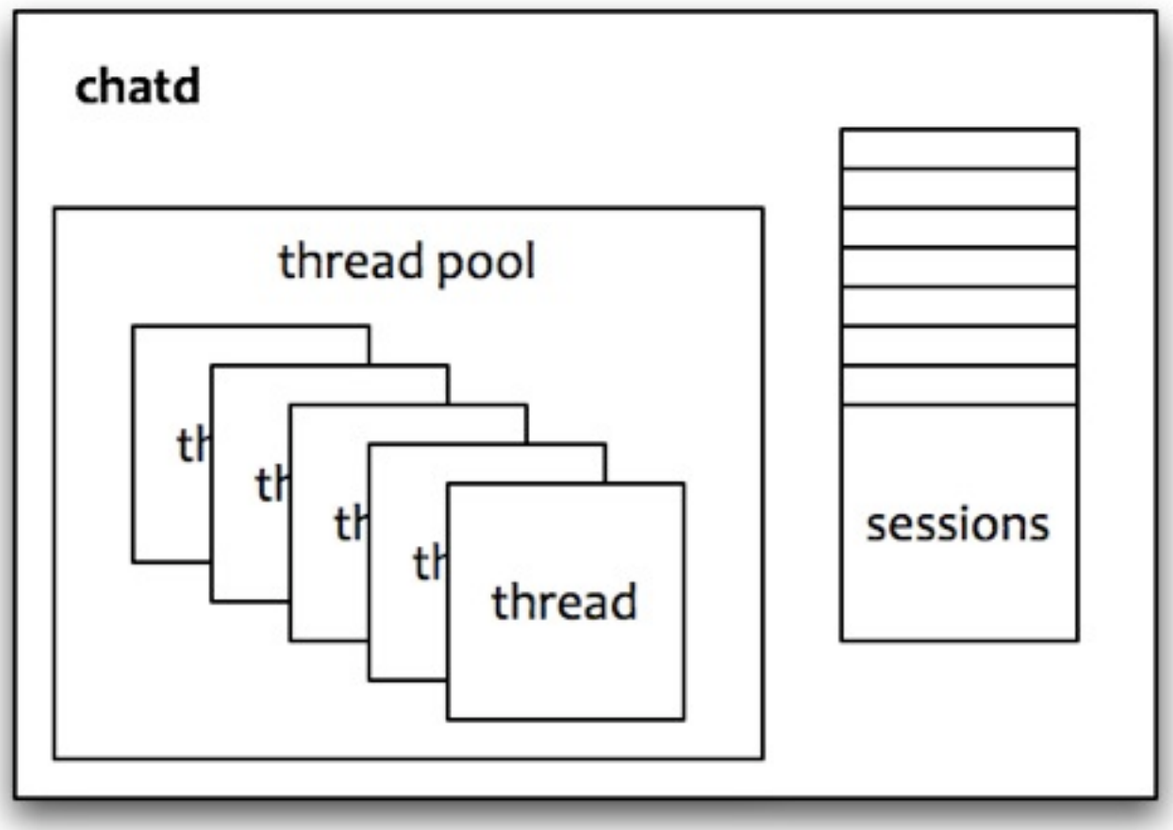
Chat proxy, take 2

- thread pool and async I/O
- sessions are “state objects”



Chat proxy, take 2

- thread pool and async I/O
- sessions are “state objects”
- harder to read; more scalable



Chat proxy, take 2

- thread pool and async I/O
- sessions are “state objects”
- harder to read; more scalable

```
class ChatSession {
  public void gotData(byte[] data) {
    // buffer, decode
    // check state...
  }
}
```

Chat proxy, take 2

- fatal flaw: blocking on other services

```
// download profile image:
byte[] data = HTTPClient.get(url);
// oh noes! now it may be 2 seconds later!
```

Chat proxy, take 2

- fatal flaw: blocking on other services
- locks up a precious thread

```
// download profile image:
byte[] data = HTTPClient.get(url);
// oh noes! now it may be 2 seconds later!
```

Chat proxy, take 2

- fatal flaw: blocking on other services
- locks up a precious thread
- sync I/O inside async callbacks

```
// download profile image:
byte[] data = HTTPClient.get(url);
// oh noes! now it may be 2 seconds later!
```

Chat proxy, take 3

- fix all APIs to be async

Chat proxy, take 3

- fix all APIs to be async

```
HTTP.get(url, new HTTPCallback() {
  public void success(byte[] data) {
    sendImageToPhone(transcode(data));
    // continue processing client request
  }
  public void failure(Exception x) { ... }
});
```

Chat proxy, take 3

- fix all APIs to be async

```
HTTP.get(url, new HTTPCallback() {
  public void success(byte[] data) {
    sendImageToPhone(transcode(data));
    // continue processing client request
  }
  public void failure(Exception x) { ... }
});
```

- if it doesn't fit on a slide, it ain't good code

Actors for I/O

- each session is an actor

Actors for I/O

- each session is an actor
- I/O events are just messages

Actors for I/O

- each session is an actor
- I/O events are just messages
- can “seek ahead” for specific events

Actors for I/O

- each session is an actor
- I/O events are just messages
- can “seek ahead” for specific events

```
react {
  case DataReceived(data) => ...
  case SessionClosed => ...
}
```

Actors for I/O

- works really well with `java.nio`

Actors for I/O

- works really well with `java.nio`
- actor-based wrapper for apache mina:

  `naggati` (on my github page)

Actors for I/O

- works really well with `java.nio`
- actor-based wrapper for apache mina:

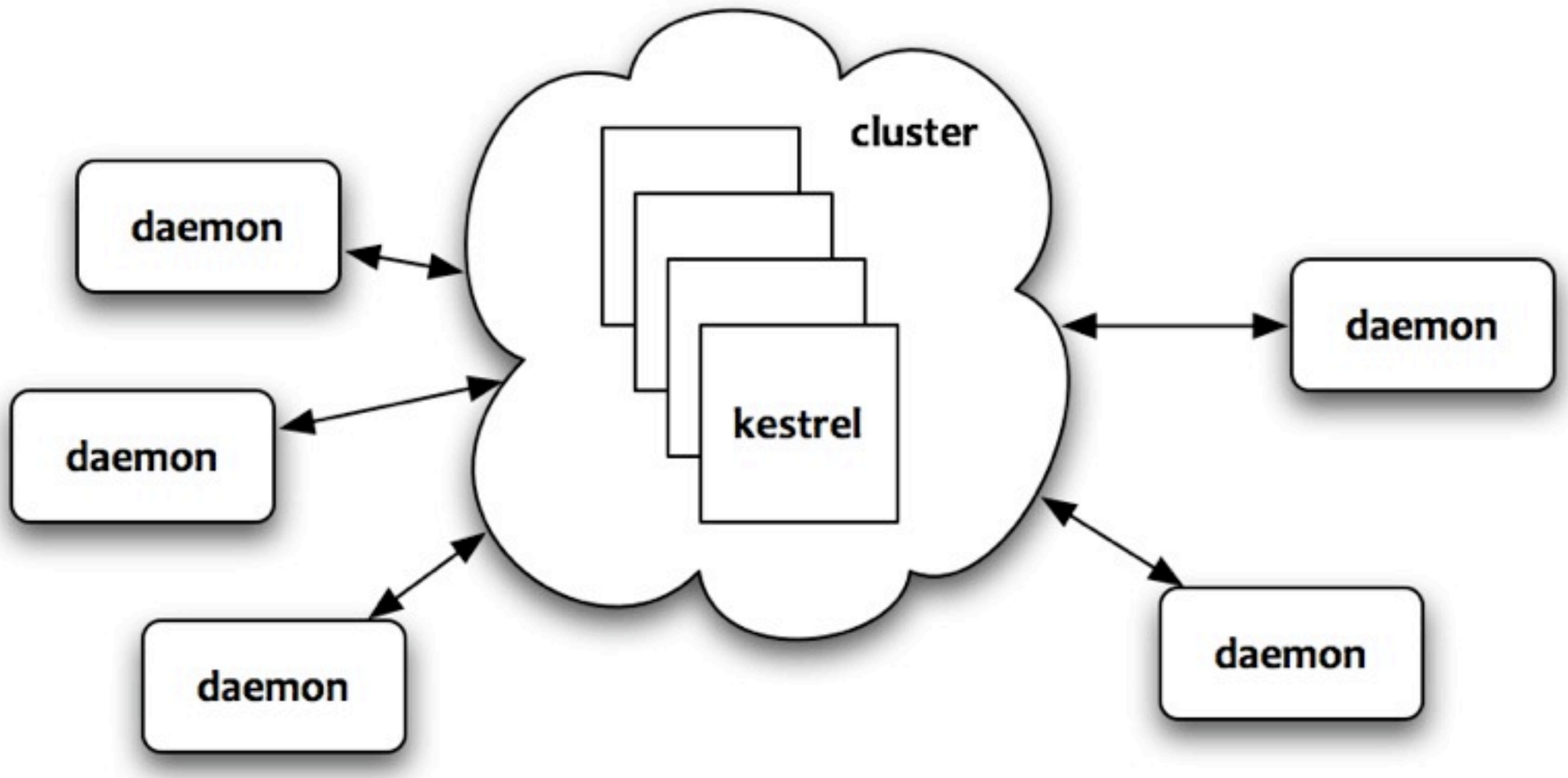
  `naggati` (on my github page)

```
HTTP.get(url)
react {
  case HTTP.Success(data) =>
    sendImageToPhone(transcode(data));
    // continue...
  case HTTP.Failure(reason) => ...
}
```

Kestrel

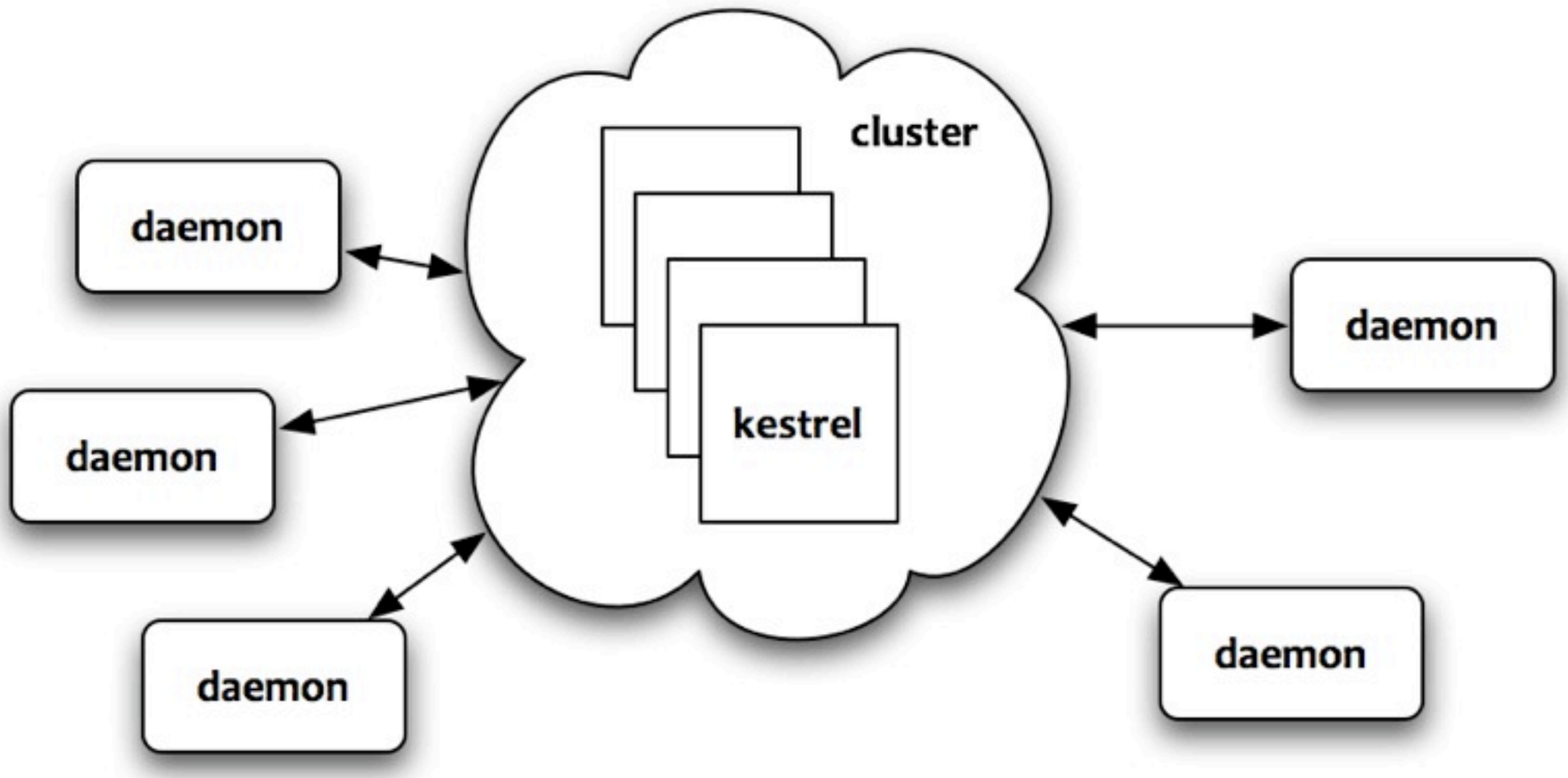
Kestrel

- very simple message queue



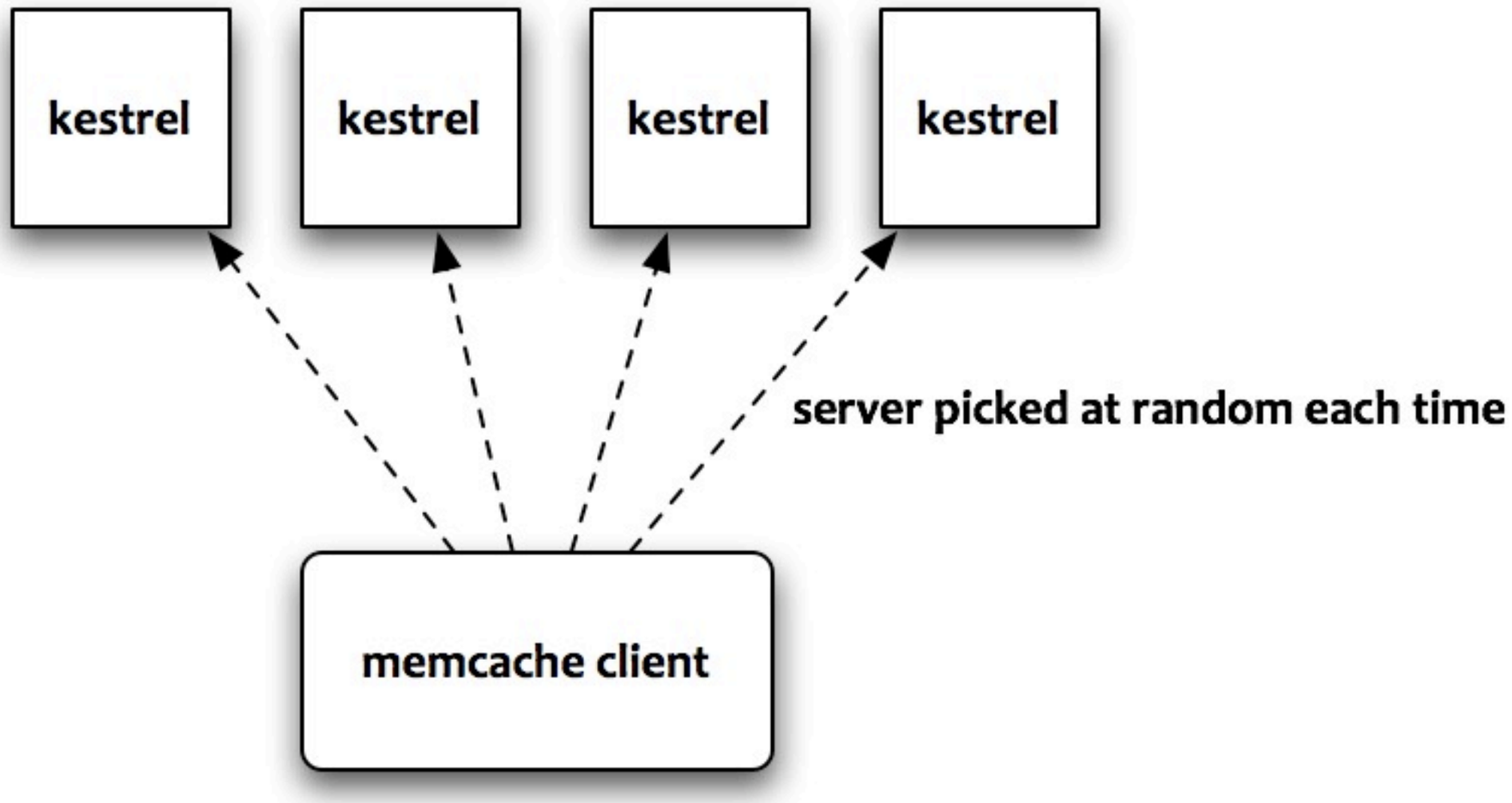
Kestrel

- very simple message queue
- each server stands alone



Kestrel

- very simple message queue
- each server stands alone



Kestrel

- each kestrel instance is strictly ordered

Kestrel

- each kestrel instance is strictly ordered
- ...making the whole cluster loosely ordered

Kestrel

- many long-lived connections

Kestrel

- many long-lived connections
- usually idle, with bursts of activity

Kestrel

- many long-lived connections
- usually idle, with bursts of activity
- (sound familiar?)

Kestrel

- using naggati, one actor per client

Kestrel

- using naggati, one actor per client
- memcache protocol interface as a mina plugin

Kestrel

- using naggati, one actor per client
- memcache protocol interface as a mina plugin
- 1.5 kloc

Kestrel

- using naggati, one actor per client
- memcache protocol interface as a mina plugin
- 1.5 kloc
- 7 class files (+ 8 test files)

Kestrel

wins

scala: about half the lines of code as java

actors: avoided concurrency puzzles

mina: complete async I/O library

Kestrel

- success

Kestrel

- success

Kestrel

- success often means “good enough”

Kestrel

- success often means “good enough”
- horizontally scales by adding machines

Kestrel

- success often means “good enough”
- horizontally scales by adding machines
- simple to understand & operate

Kestrel

- success often means “good enough”
- horizontally scales by adding machines
- simple to understand & operate
- minimal locking / thinking about concurrency

Kestrel

- success often means “good enough”
- horizontally scales by adding machines
- simple to understand & operate
- minimal locking / thinking about concurrency

Kestrel

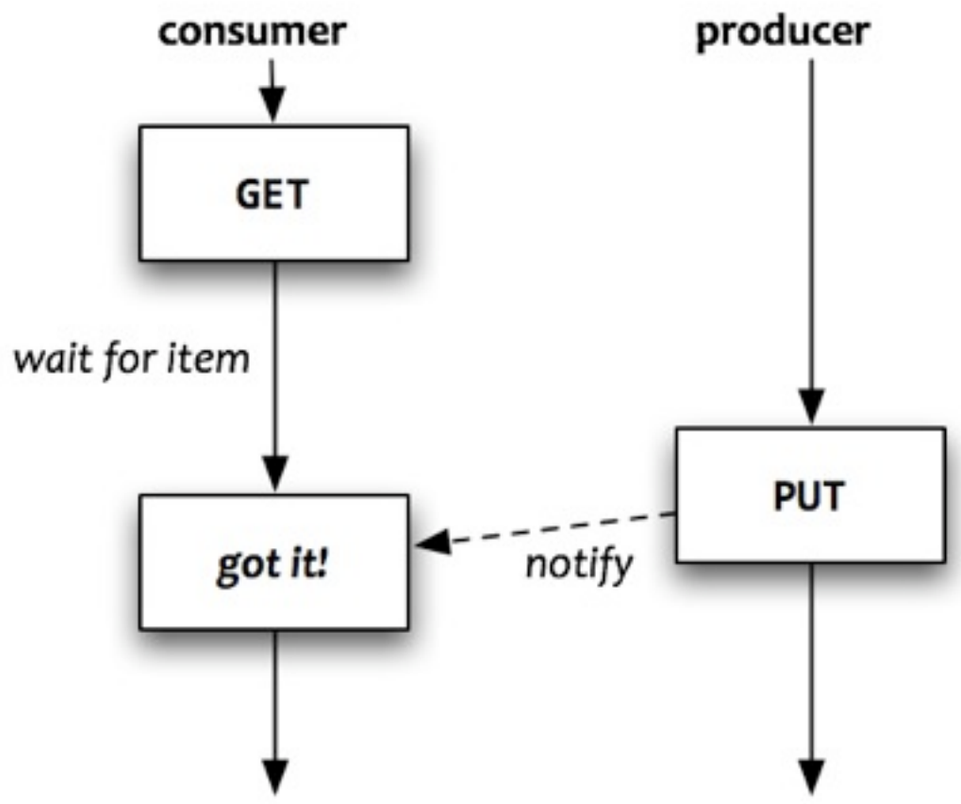
- success often means “good enough”
- horizontally scales by adding machines
- simple to understand & operate
- minimal locking / thinking about concurrency

```
>stats
STAT uptime 3138307
STAT cmd_get 4090226689
STAT cmd_set 1631380861
STAT bytes_written 237156246614
```

Serendipity

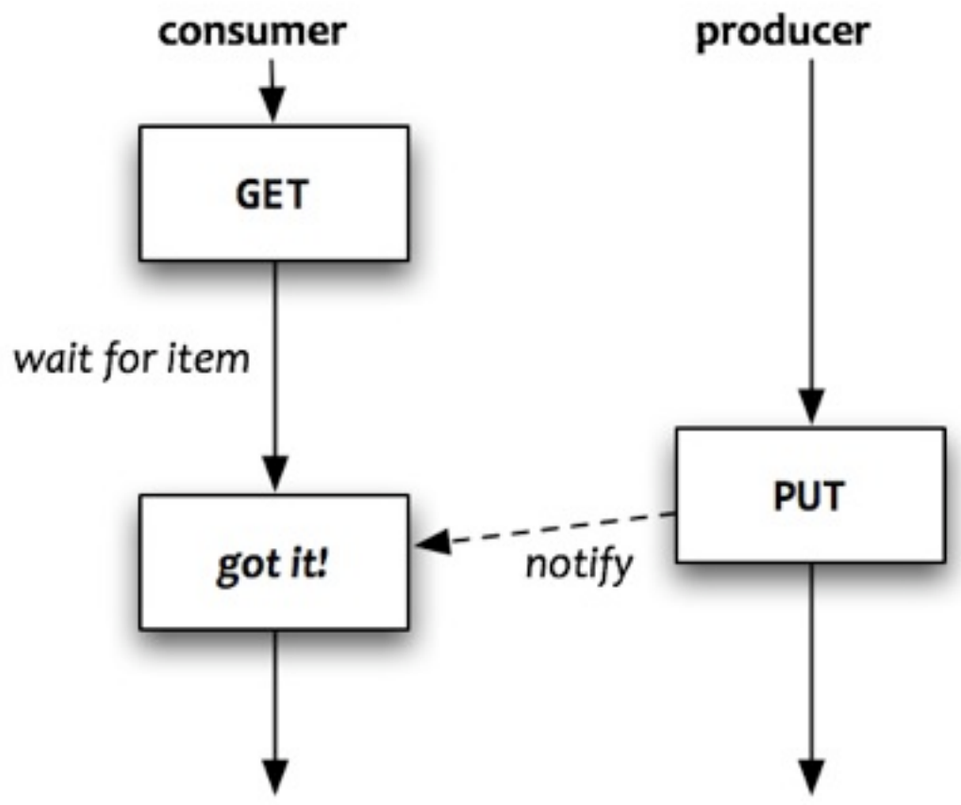
Serendipity

- actors are just one of many tools



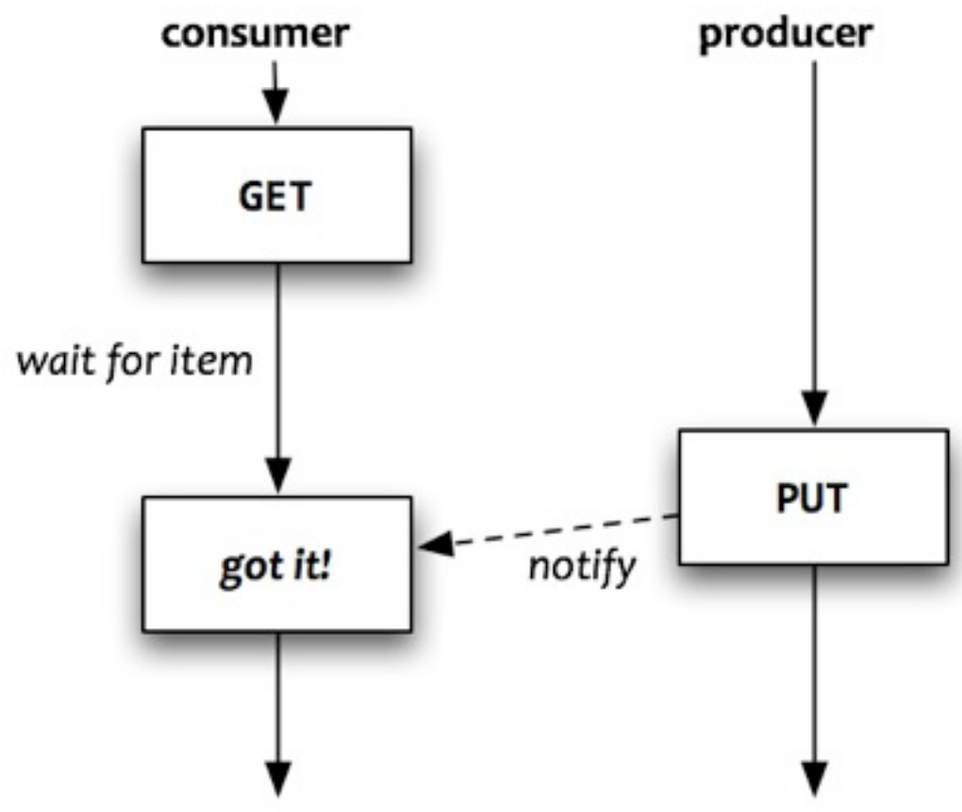
Serendipity

- actors are just one of many tools
- can be combined with locks/etc



Serendipity

- actors are just one of many tools
- can be combined with locks/etc
- example: want to "hand off" a queue item from a producer to a consumer



Serendipity

```
val waiters = new ArrayBuffer[Actor]

def put(item: QItem) {
  ...
  synchronized {
    if (waiters.size > 0) {
      waiters.remove(0) ! ItemArrived
    }
  }
}
```

Serendipity

```
def removeWithin(timeout: Long) {
  synchronized {
    if (queue.isEmpty) {
      waiters += self
      receiveWithin(timeout) {
        case ItemArrived => remove()
        case TIMEOUT => None
      }
    }
  }
}
```

Serendipity

- yes, I used `synchronized`

Serendipity

- yes, I used `synchronized`
- probably subject to excommunication now

Serendipity

- yes, I used `synchronized`
- probably subject to excommunication now
- but the results were worth it

Where actors didn't work

- first draft: each queue is an actor!

Where actors didn't work

- first draft: each queue is an actor!
- `queue ! PUT(item)`

Where actors didn't work

- first draft: each queue is an actor!
- `queue ! PUT(item)`
- message delivery overhead was too high

Where actors didn't work

- first draft: each queue is an actor!
- `queue ! PUT(item)`
- message delivery overhead was too high
- the put operation was just too small:

```
memoryQueue.add(item)
journal.write(Put(item))
```

Where actors didn't work

- first draft: each queue is an actor!
- `queue ! PUT(item)`
- message delivery overhead was too high
- the put operation was just too small:

```
memoryQueue.add(item)
journal.write(Put(item))
```

- find this out with profiling -- don't guess!

Where actors are shaky (in scala)

- lifetime issues still being shaken out
  (easy to workaround; fixed in next release)

Where actors are shaky (in scala)

- lifetime issues still being shaken out
  (easy to workaround; fixed in next release)
- mixing threads with actors
  (raw threads get proxy actors which are hard to GC correctly)

Where actors are shaky (in scala)

- lifetime issues still being shaken out
  (easy to workaround; fixed in next release)
- mixing threads with actors
  (raw threads get proxy actors which are hard to GC correctly)
- but! TOP MINDS are working on it

Where actors are shaky (in scala)

- lifetime issues still being shaken out
  (easy to workaround; fixed in next release)
- mixing threads with actors
  (raw threads get proxy actors which are hard to GC correctly)
- but! TOP MINDS are working on it
- scala 2.8 should have significant improvements / simplifications

FIN