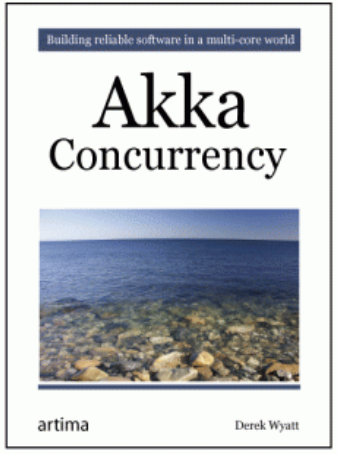
www.movio.co

Work Pulling Pattern

- www.michaelpollmeier.com
- @pollmeier

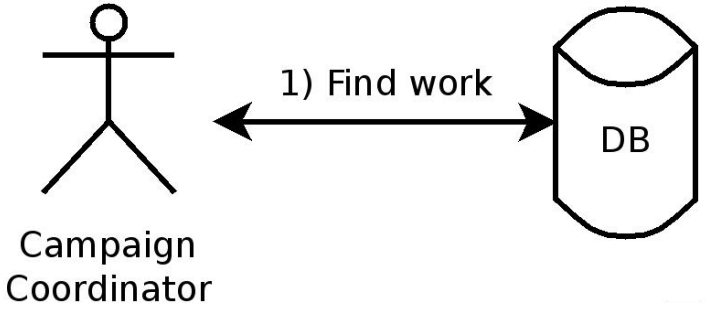
- Based on Derek Wyatt's work



Motivation

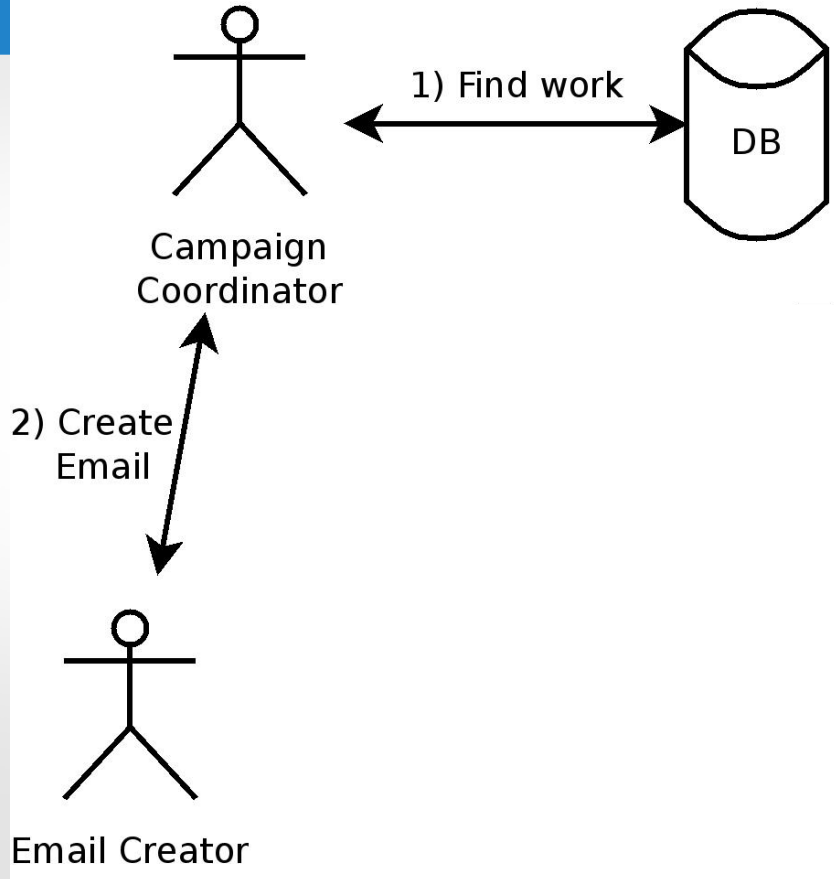
Scenario

```
val work = dao.findWork()
```



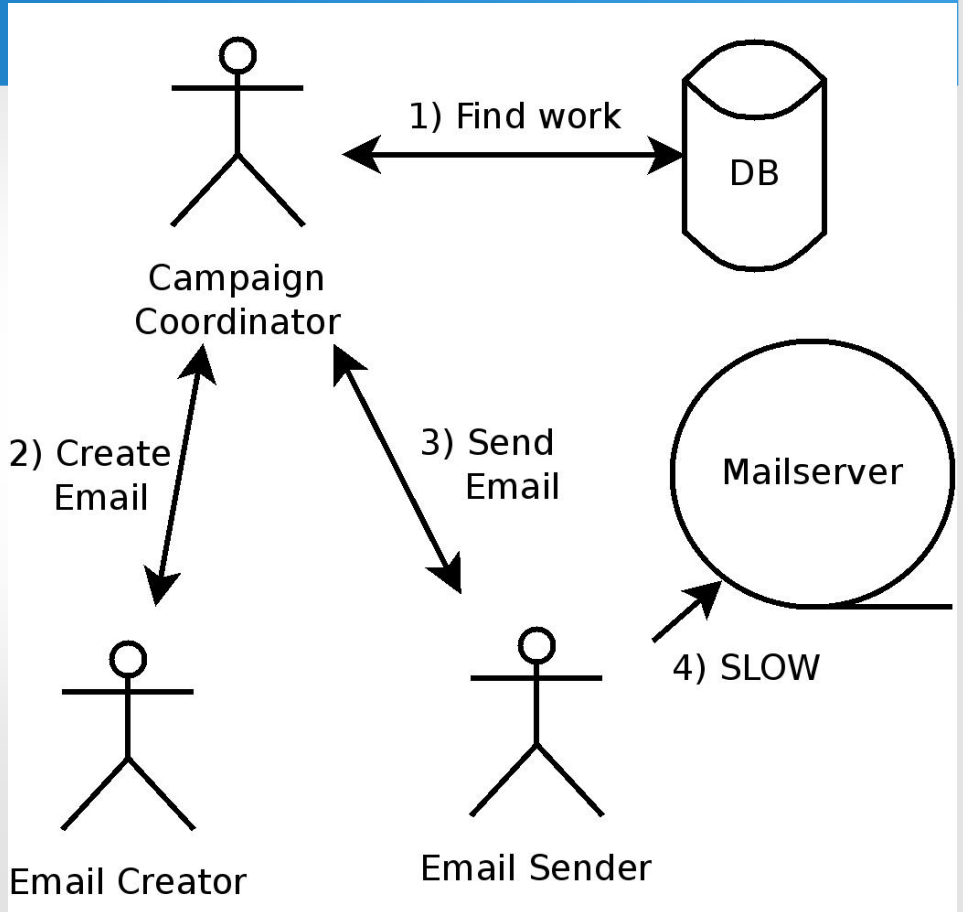
Scenario

```
val work =  dao.findWork()

val email = creator
      .createEmail(work)
```



Scenario

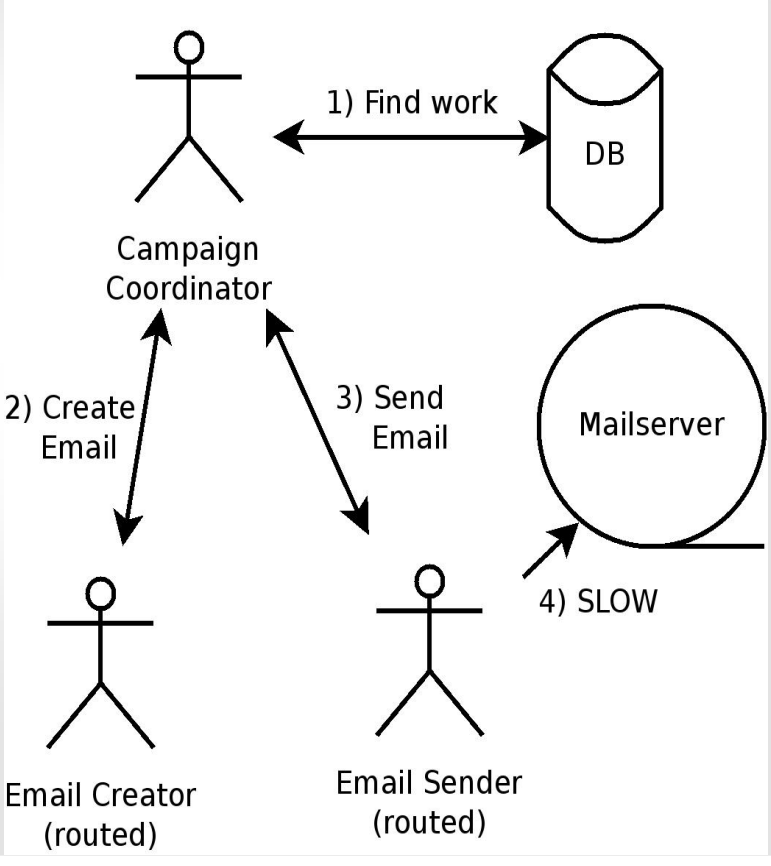
```
val work =
      dao.findWork()

val email = creator
      .createEmail(work)

emailSender
      .send(email)
```



First try with Actors

```
case Start ⇒ findWorkBatch()
      foreach { emailCreator ! _ }
      self ! Start
case email: Email ⇒
      emailSender ! email
```

Problem?



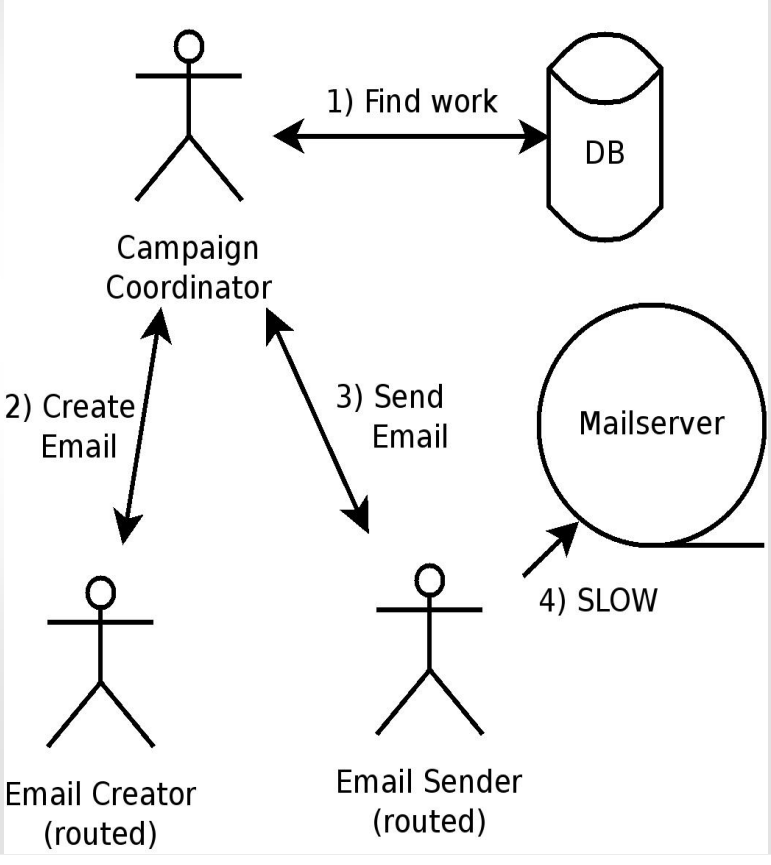
First try with Actors

```
case Start ⇒ findWorkBatch()
      foreach { emailCreator ! _ }
      self ! Start
case email: Email ⇒
      emailSender ! email
```

Problem: producing work faster than working it off

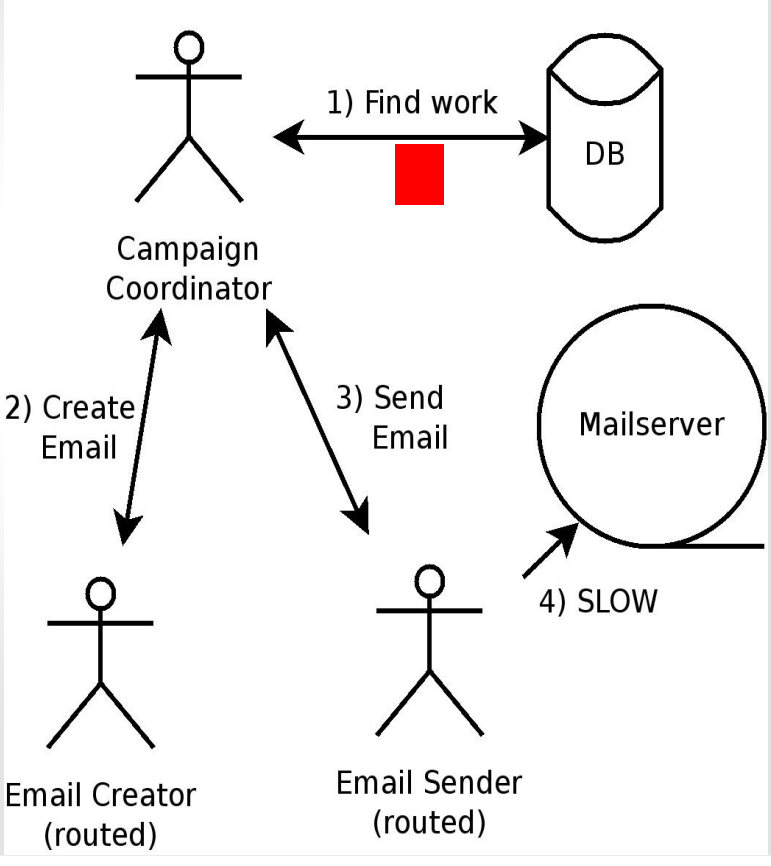
Risk: message queue overflow

Solution?



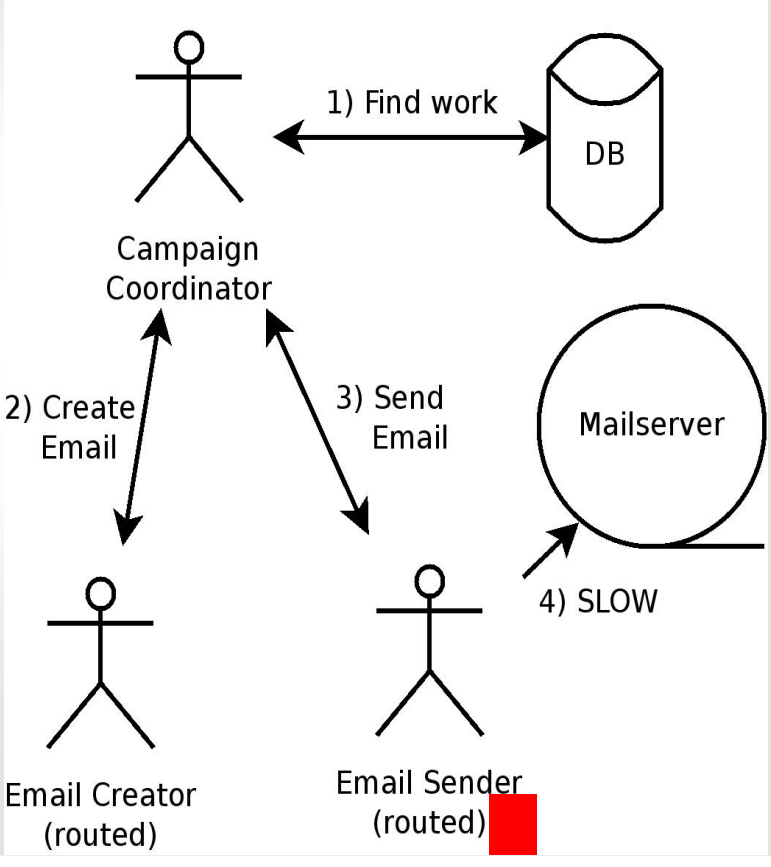
Solutions with Caveats

Small Work



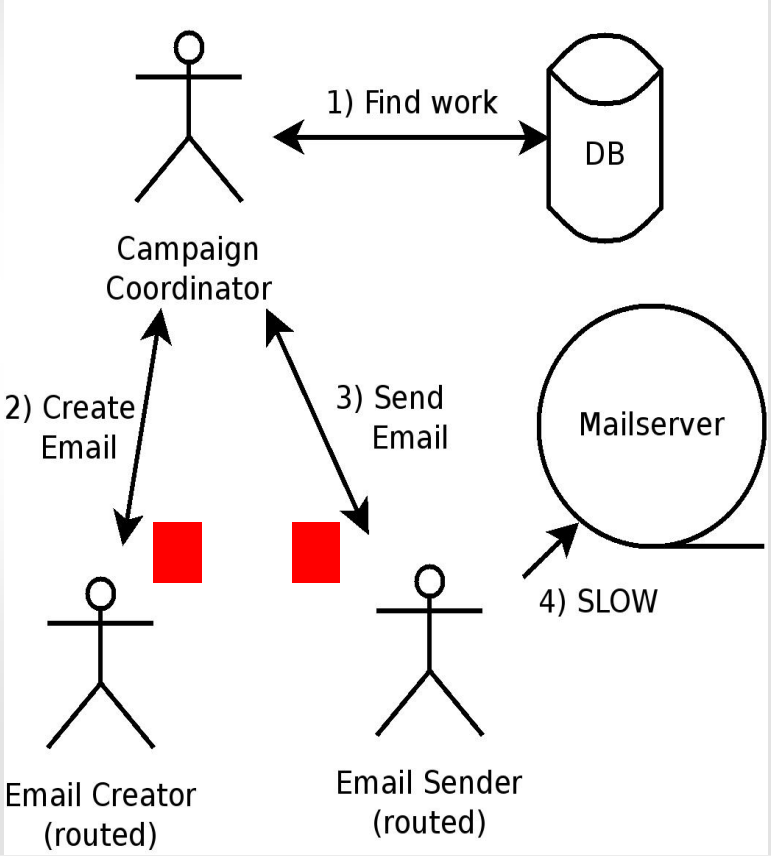
Solutions with Caveats

Use <insert high number> workers



Solutions with Caveats

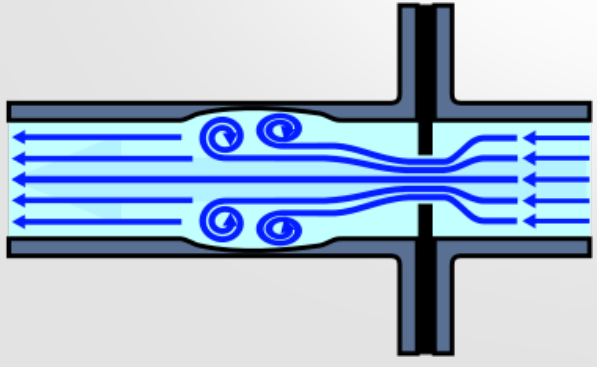
Limit worker's mailbox size

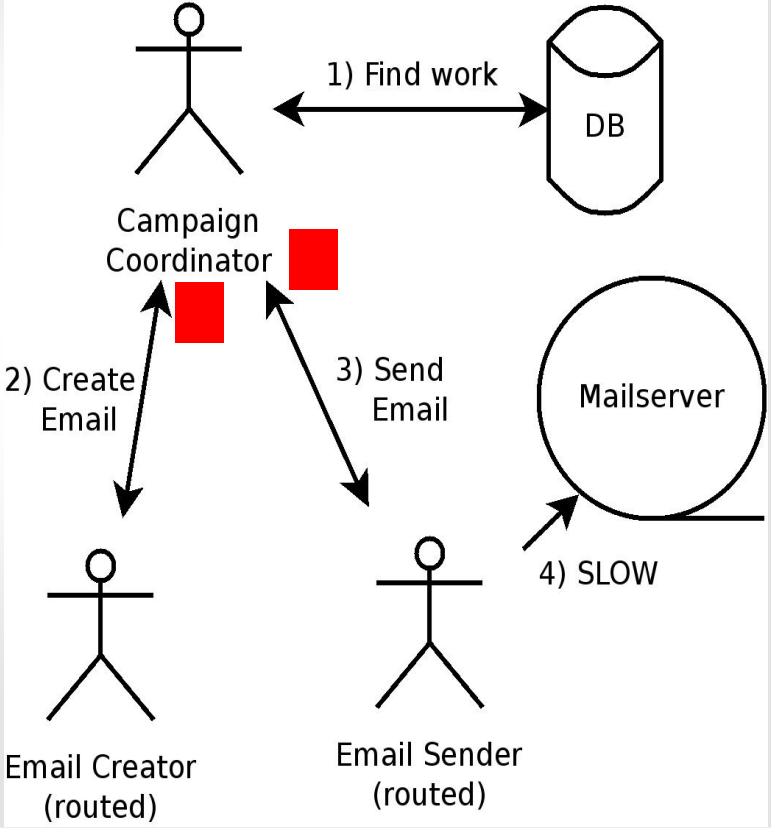


Solutions with Caveats

Derek Wyatt's PressureQueue

https://github.com/derekwyatt/PressureQueue-Concept/

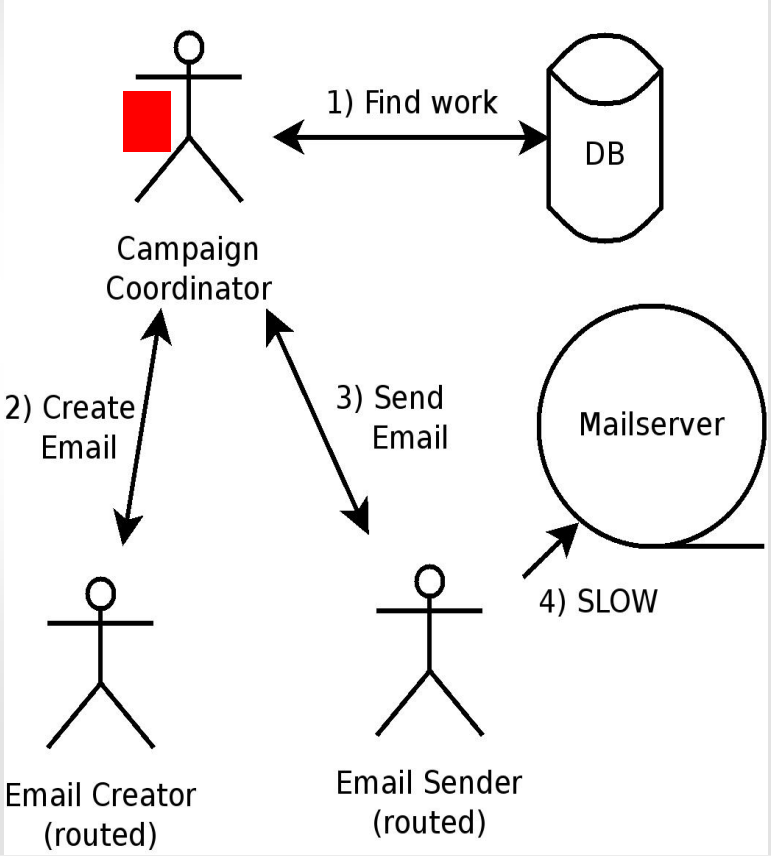




Solutions with Caveats

TimerBasedThrottler

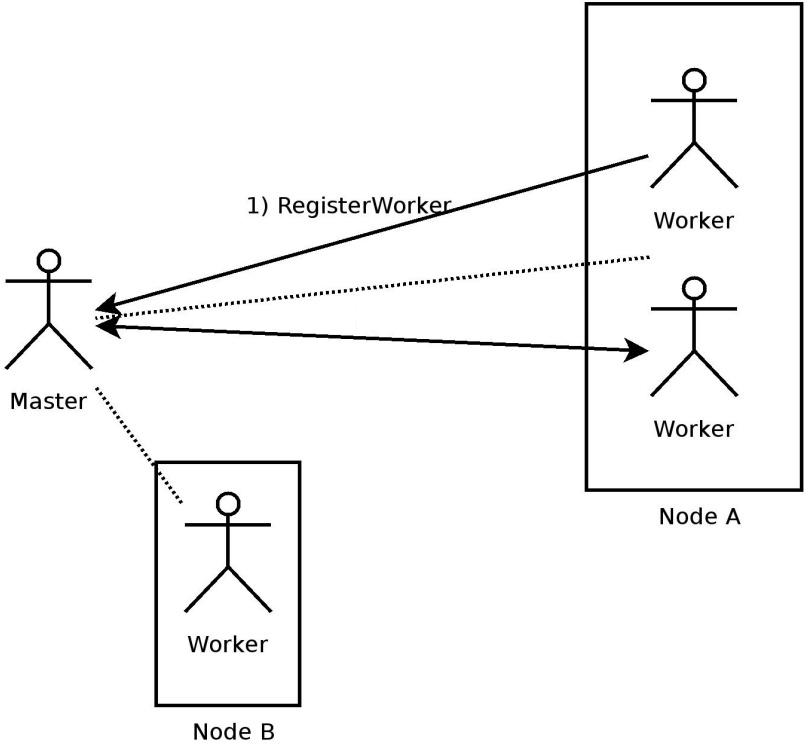
http://letitcrash.com
/post/28901663062/



Work Pulling Pattern

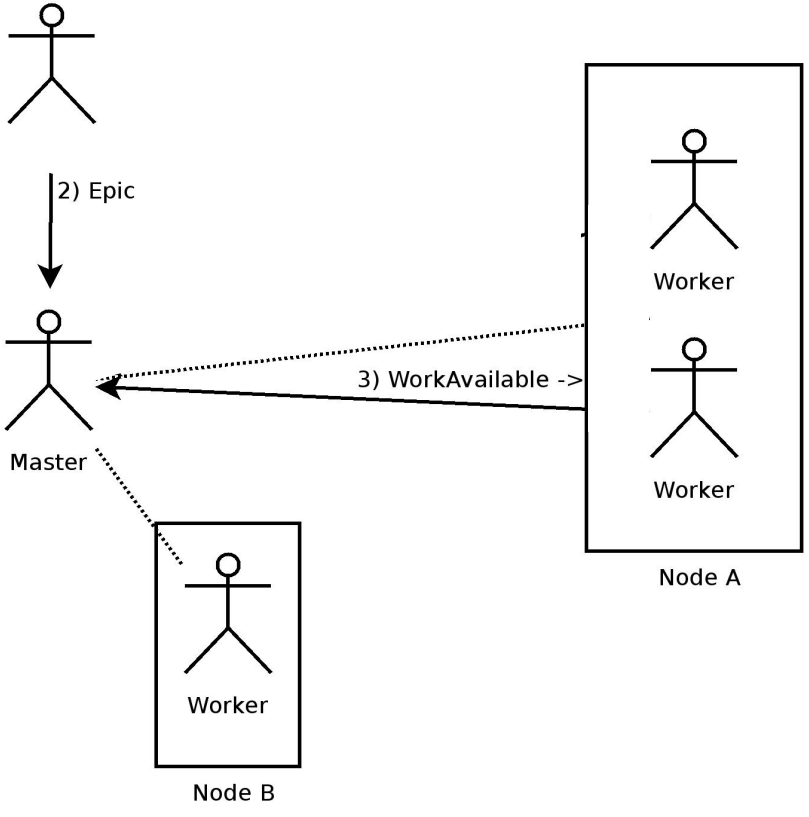
Workers can join any time
from anywhere

```
Worker:
preStart {
  master ! RegisterWorker(self)
}

Master:
case Register(worker) ⇒
  workers += worker
```



Work Pulling Pattern

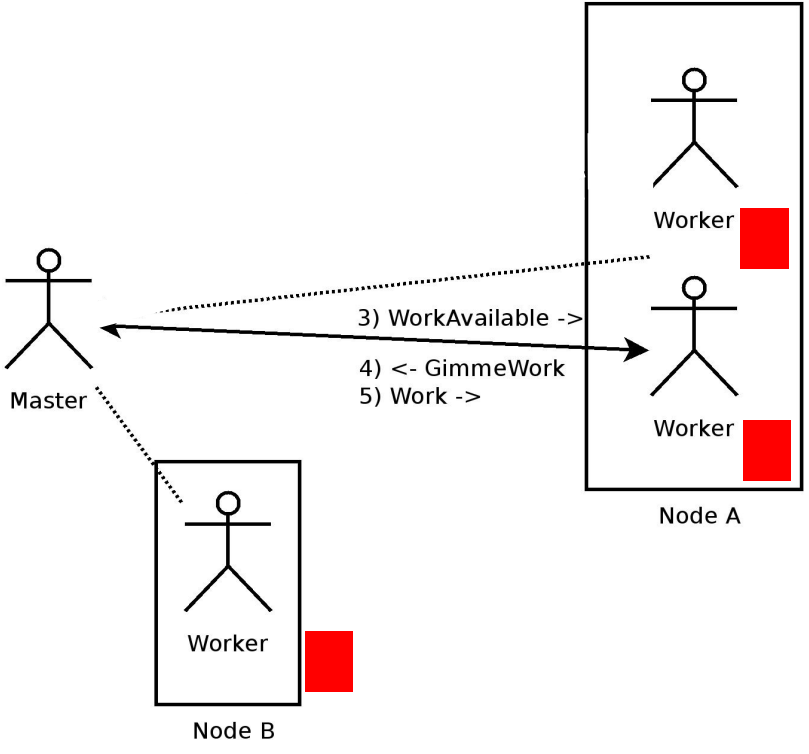
```
type Epic = Stream[T]

case epic: Epic ⇒
      currentEpic = epic
      workers foreach
      { _ ! WorkAvailable }
```



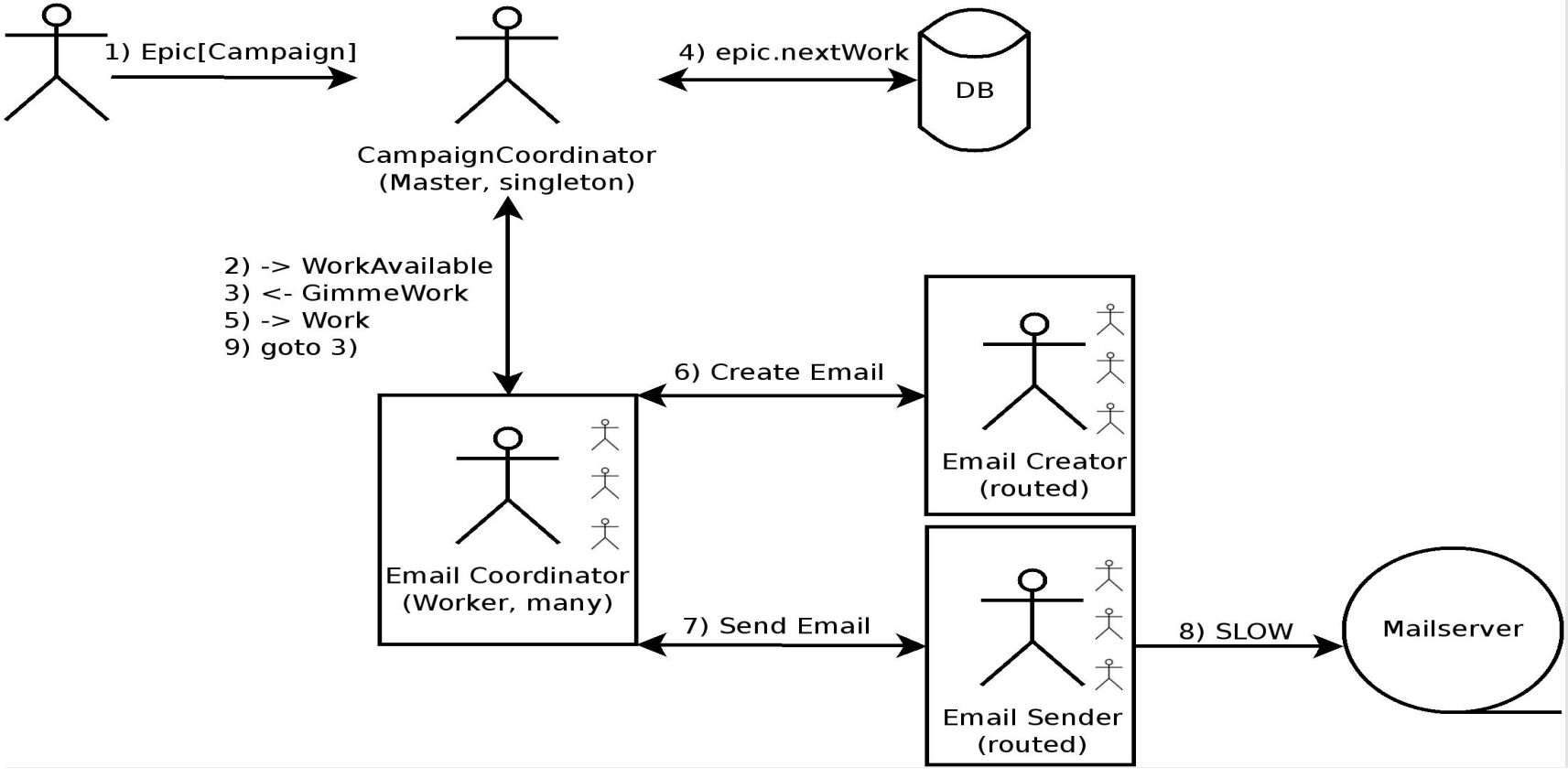
Work Pulling Pattern

```
Worker:
case WorkAvailable ⇒
      master ! GimmeWork

case Work(work: T) ⇒
     doWork(work) onComplete
     master ! GimmeWork

def doWork(work: T): Future[T]
```

Not polling!



The Complete Picture

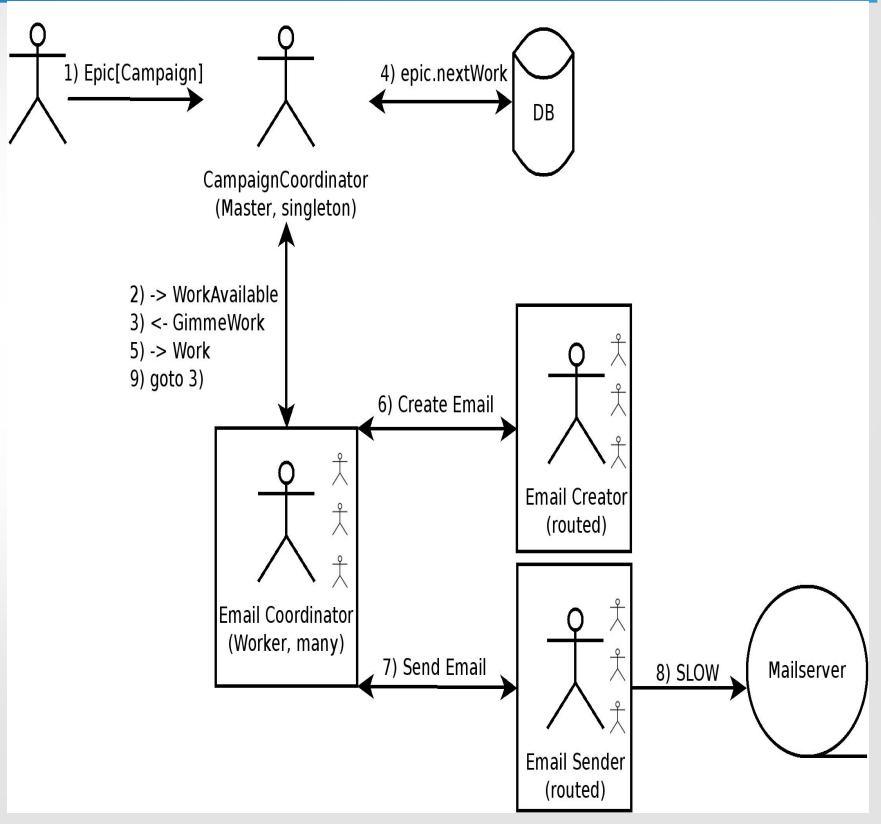


EmailCoordinator (Worker)

```
def doWork(context: T): Future[_] =
  for {
    email  ← creator ? context
    result  ← sender ? email
  } yield result
```

Alternatively:

```
def doWork(context: T): Future[_] =
creator ? context flatMap {sender ? _}
```



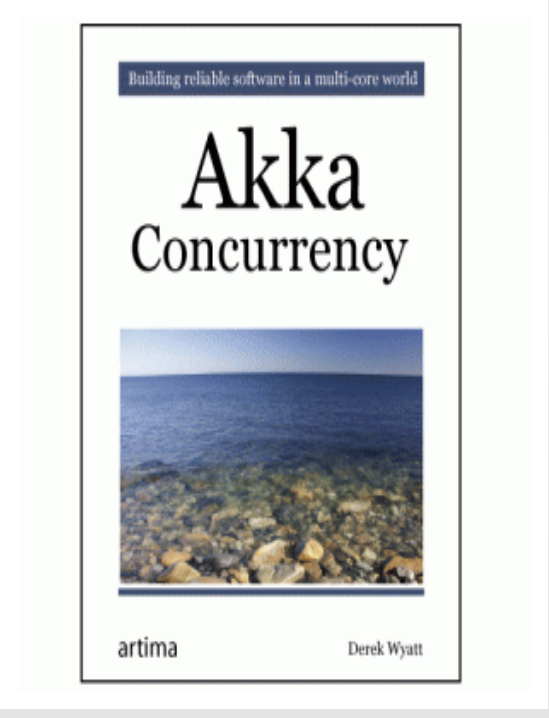
The Alternative: Futures

Reuse of existing Actors?
Supervision, deathwatch etc?
Distribute work across cluster?

Thank you!

www.michaelpollmeier.com
@pollmeier
www.movio.co → we're hiring!

Derek Wyatt:
http://letitcrash.com/post/29044669086/




MOVIO