



# GPars

Groovy Parallel Systems

Václav Pech

# About me

- Passionate programmer
- Concurrency enthusiast
- GParS @ Codehaus lead
- Technology evangelist @ JetBrains
- JetBrains Academy member



<http://www.jroller.com/vaclav>  
[http://twitter.com/vaclav\\_pech](http://twitter.com/vaclav_pech)



# GParS

Apache 2 license

Hosted @ Codehaus

5 committers

Tight relation with the Groovy team

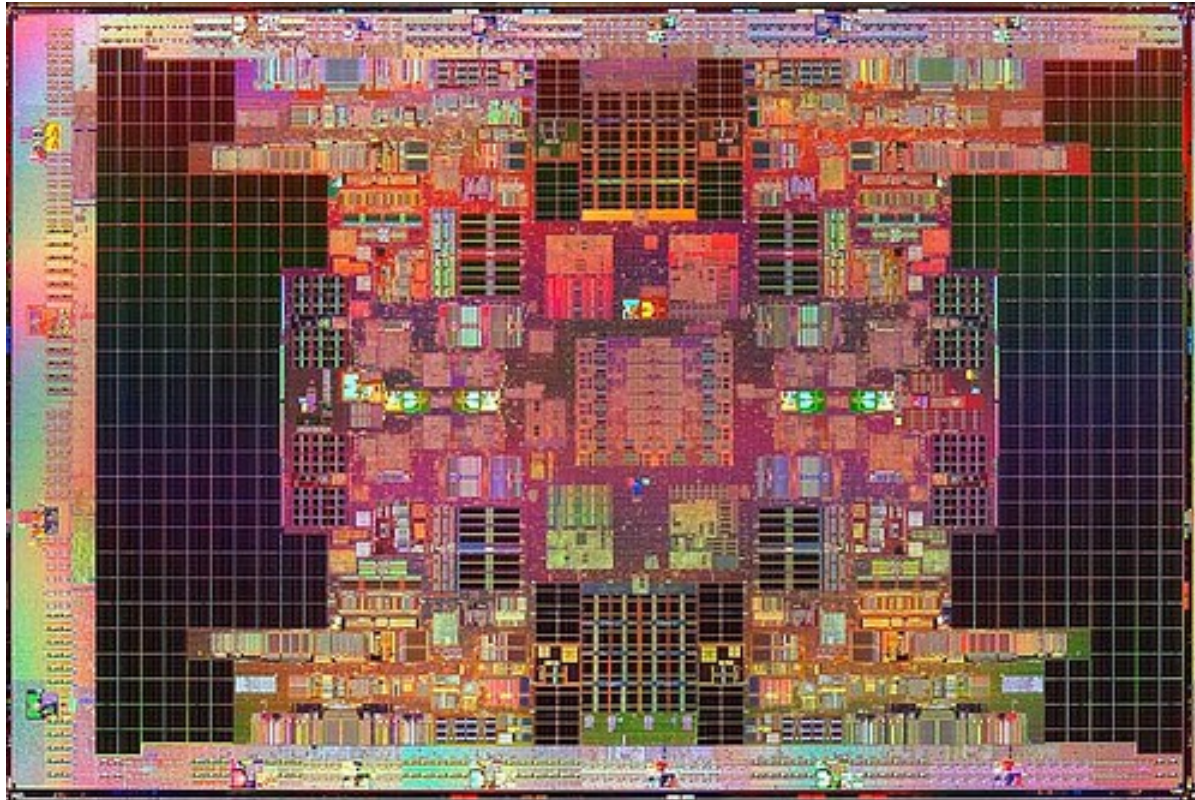
Originally named GParallelizer

# Pantha rei



Entering a new era!

# We're quad core already



Beware: More cores to come shortly!

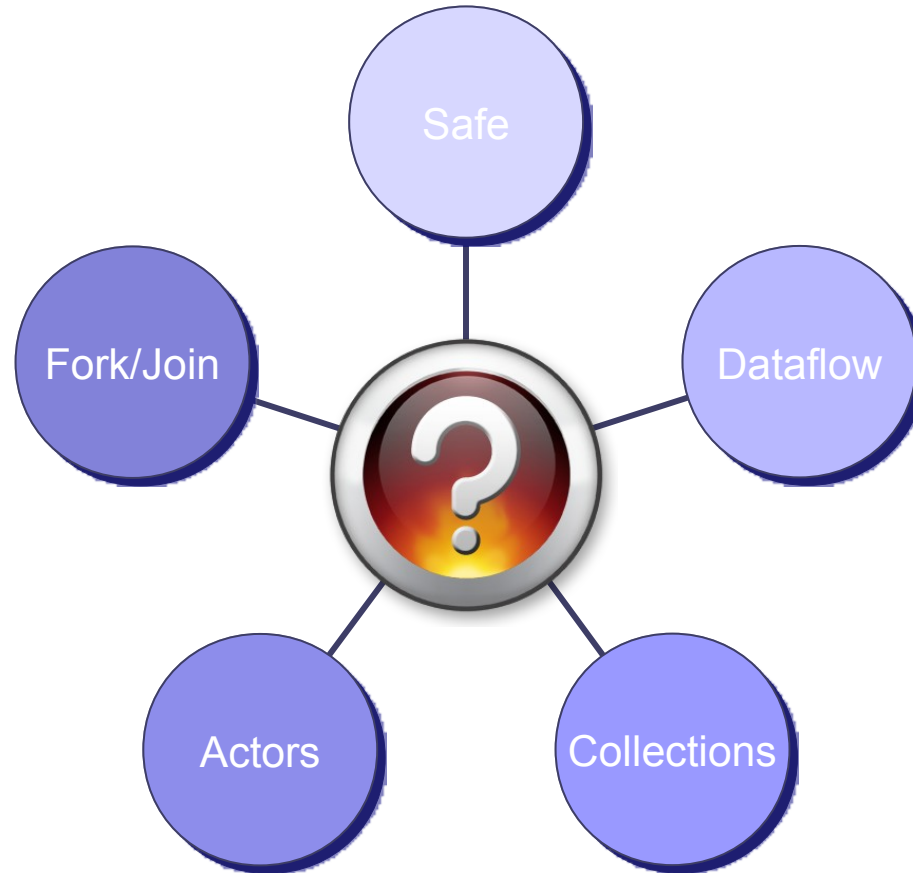


# Locks and threads

Multithreaded programs today work mostly by accident!



# Toolset

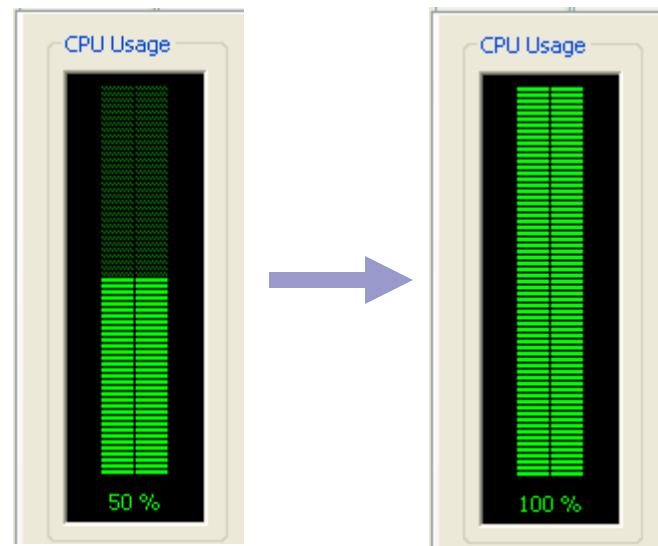




# Parallelizer

```
doParallel {  
  images.findAllParallel{it.contains me}  
  .collectParallel{convert it}  
  .groupByParallel{it.size()}  
}
```

- **Important:**  
Side-effect-free functions only!



# Transparently parallel

```
doParallel(4) {  
  images.makeTransparent()  
    .findAll {it.contains me}  
    .collect {convert it}  
    .groupBy {it.size()}  
}
```

# Parallelize existing code

```
buyBargain stocks.makeTransparent()
```

```
def buyBargain(stocks) {  
  buy stocks.findAll {  
    it.download().price < 50  
  }  
}
```

Use with CAUTION!

# Functional flavor

## Map / Reduce

map, reduce, filter, min, max, sum, ...

```
(1..n).parallel.filter {it%2==0}
```

```
.map {it ** 2}
```

```
.reduce {a, b -> a + b}
```

# Call closures asynchronously

```
def isSelfPortrait = {image -> image.contains me}  
def flag = isSelfPortrait.call(img1)
```

```
def future = isSelfPortrait.callAsync(img1)
```

...

```
def flag = future.get()
```

# Asynchronous closures

```
def resize = {img -> img.resize(64, 64)}  
def fastResize = resize.async()
```

```
def resized = images.collect fastResize
```

```
...
```

```
createAlbum resized*.get()
```

# Fork/Join Orchestration

```
protected void compute() {
    long count = 0;
    file.eachFile {
        if (it.isDirectory()) {
            println "Forking a thread for $it"
            forkOffChild(new FileCounter(it))
        } else {
            count++
        }
    }
    setResult(count + (childrenResults?.sum() ?: 0))
}
```





Waits for children  
without blocking the  
thread!



# GroovyDSL – IntelliJ IDEA CE



```
Parallelizer.doParallel {  
    images.collectParallel {it.toGrayscale()}  
    if (images.anyParallel {it.size() > MAX_IMG_SIZE}) rejectImages()  
    def familyImages = images.findAll  
}
```

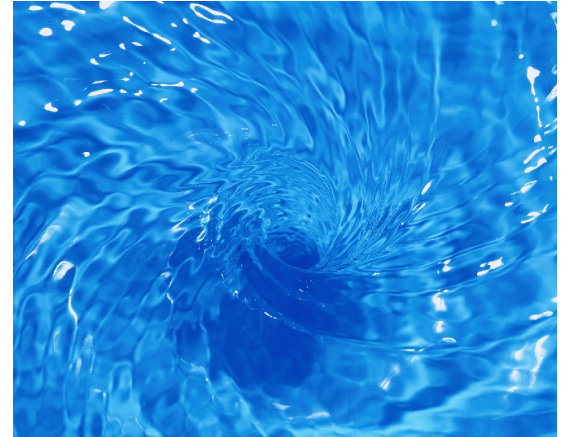
 	<code>findAll(Closure closure)</code>	<code>Collection&lt;T&gt;</code>
 	<code>findAllParallel(Closure closure)</code>	<code>Object</code>



# Dataflow Concurrency

- No race-conditions
- No live-locks
- Deterministic deadlocks

Completely deterministic programs



## BEAUTIFUL code

(Jonas Bonér)

# Dataflow Variables

```
task { z << x.val + y.val }
```

```
task { x << 40 }
```

```
task { y << 2 }
```

```
assert 42 == z.val
```

- Single-assignment variables  
with blocking read

# DataFlows

```
def df = new DataFlows()
```

```
task { df.z = df.x + df.y }
```

```
task { df.x = 10 }
```

```
task { df.y = 5 }
```

```
assert 15 == df.z
```

# DataFlows Making Money

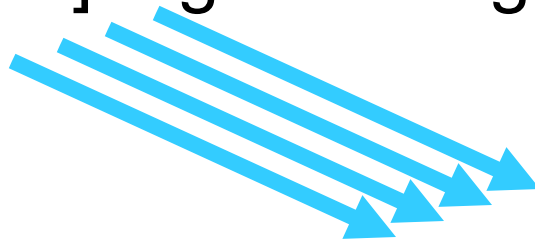
```
def stocks = ['AAPL', 'GOOG', 'IBM', 'JAVA']
```

```
def price = new DataFlows()
```

```
stocks.each( {stock ->
```

```
    price[stock] = getClosing(stock, 2009)
```

```
}.async())
```



```
def topStock = stocks.max { price[it] }
```



# Dataflow Operators

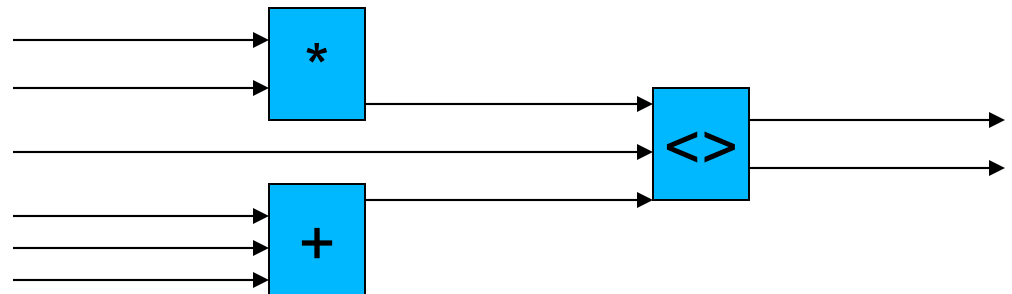
```
operator(inputs: [stocksStream],  
         outputs: [pricedStocks])
```

```
{stock ->
```

```
  def price = getClosing(stock, 2008)
```

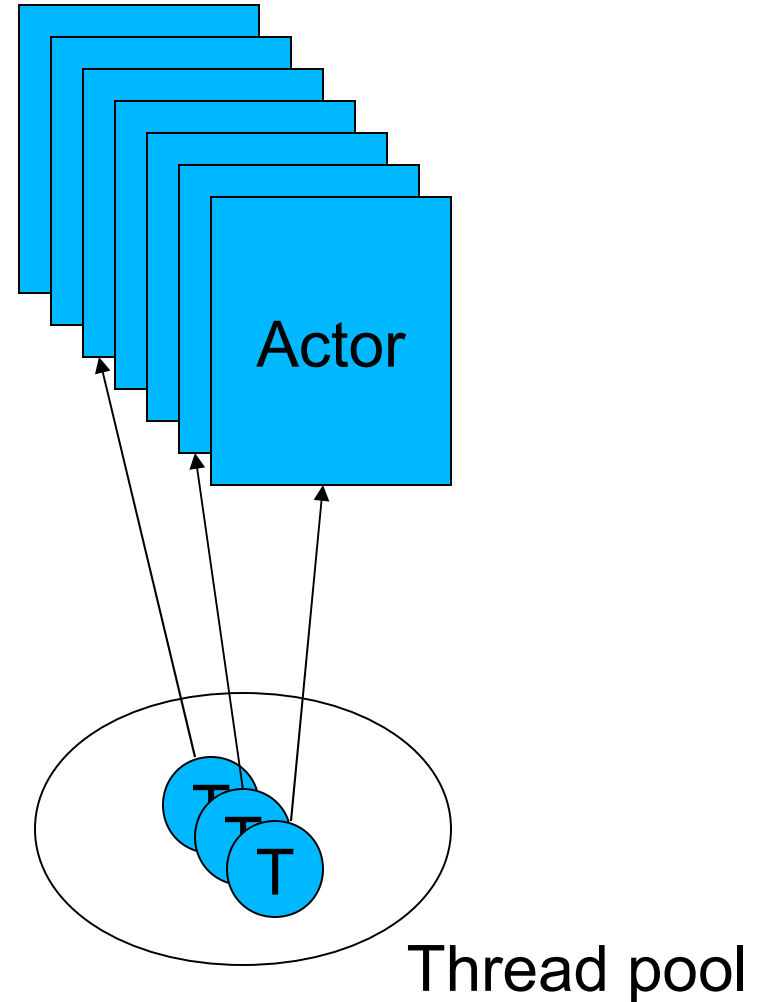
```
  bindOutput(0, [stock: stock, price: price])
```

```
}
```

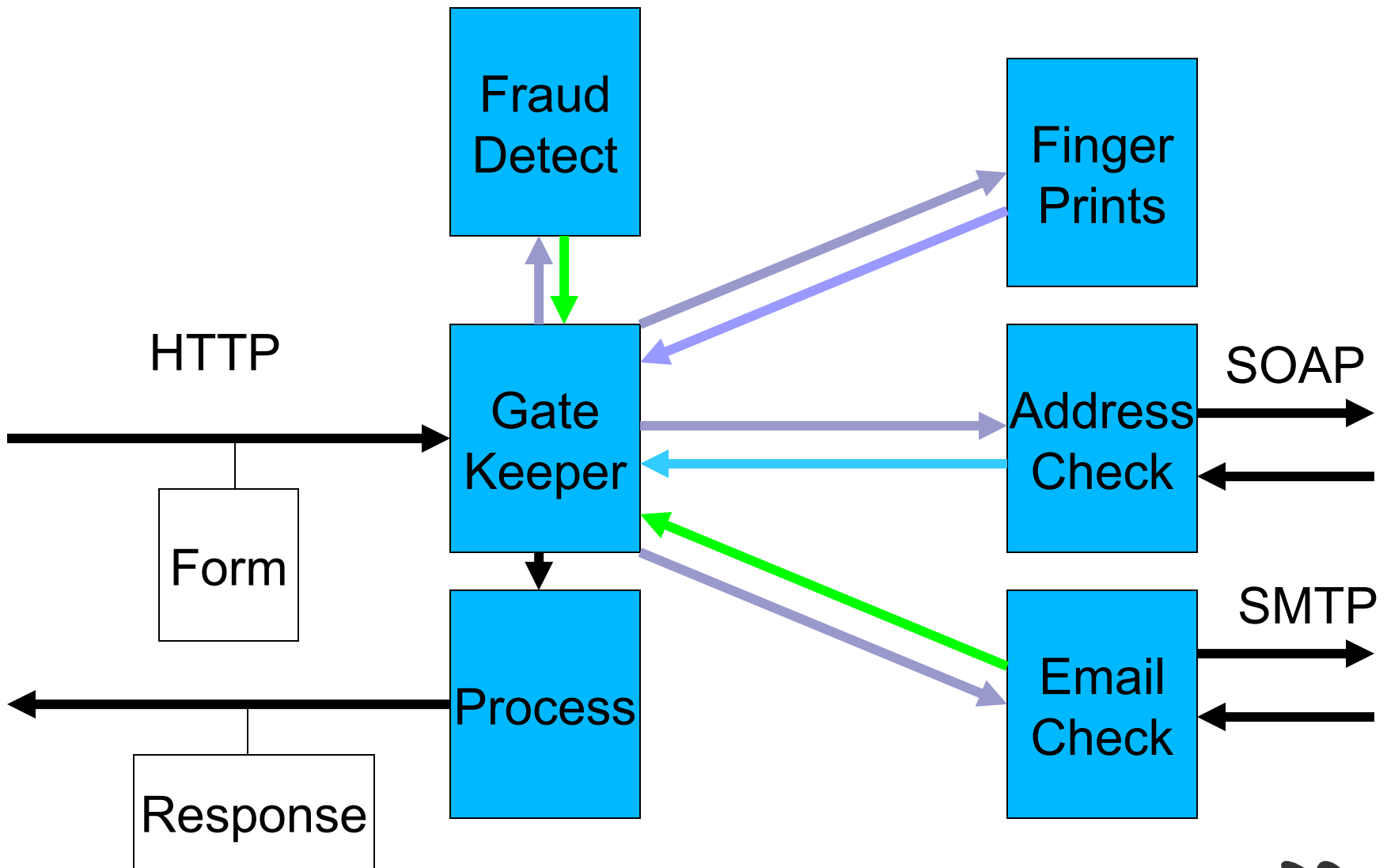


# Actors

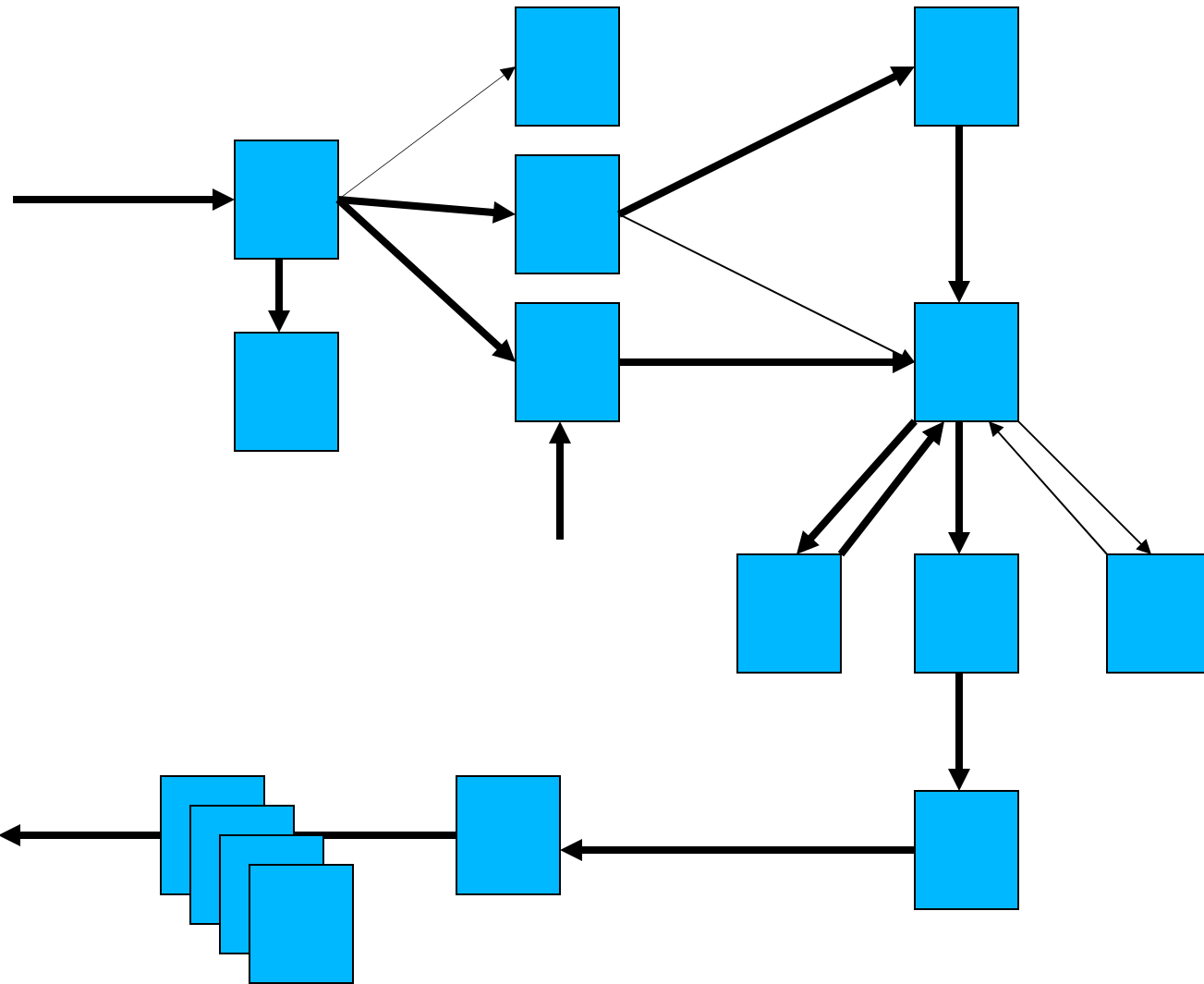
- Isolated
- Communicating
  - Immutable messages
- Active
  - Pooled shared threads
- Activities
  - Create a new actor
  - Send a message
  - Receive a message



# Actors use



# Actors patterns



Enricher

Router

Translator

Endpoint

Splitter

Agregator

Filter

Resequencer

Checker



# Creating Actors

```
class MyActor extends AbstractPooledActor {  
  void act() {  
    def buddy = new YourActor()  
    buddy << 'Hi man, how\'re things?'  
    def response = receive()  
  }  
}
```

# Creating Actors

```
def decryptor = actor {  
  loop {  
    react {msg ->  
      reply msg.reverse()  
    }  
  }  
}
```

```
decryptor << 'noitcA nI yvoorG'
```

# Sending messages

```
buddy.send 10.eur
```

```
buddy << new Book(title:'Groovy Recipes',  
                    author:'Scott Davis')
```

```
def canChat = buddy.sendAndWait 'Got time?'
```

```
buddy.sendAndContinue 'Need money!', {cash->  
    pocket.add cash  
}
```

# Reacting to messages

```
react {gift1 ->
  reply "Thank you for $gift1"
  react {gift2 ->
    reply "Wow, one more $gift2"

    [gift1, gift2].max{it.price}.reply
      'Will you marry me?'
  }
  //Never reached
}
```

# Choosing the Reaction

```
react / receive {gift ->
  switch (gift) {
    case Money:reply 'Thanks';pocket gift; break
    case [iPhone, iPod]:child << gift; break
    case (BigFlat..SmallHouse):moveIn(gift); break
    case Clothes:
      putOn(gift)
      fits(gift)?reply 'Thanks':reply gift
      break
    case EXIT:stop()
  }
}
```

# Continuation Style

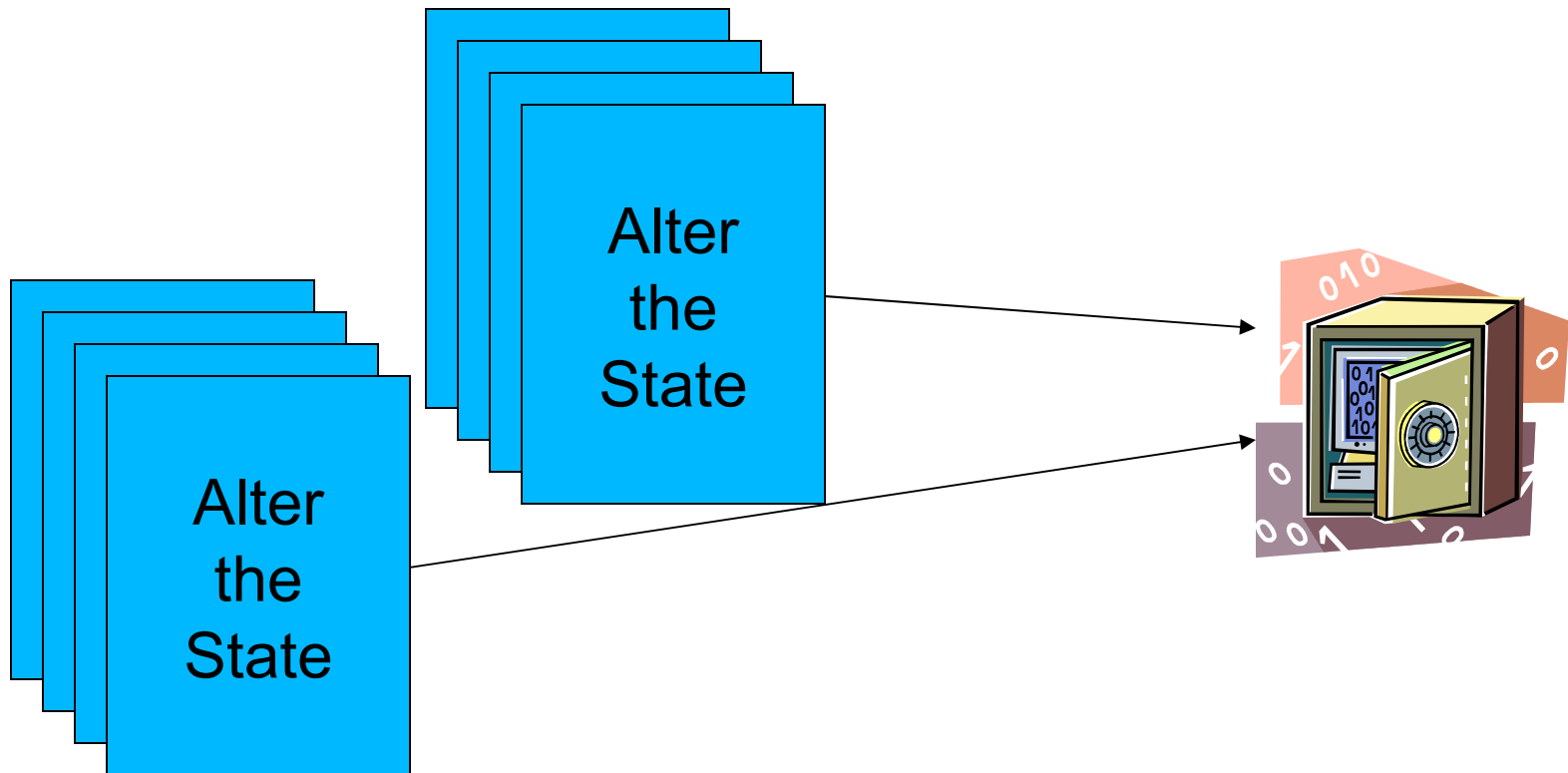
```
loop {  
  ...  
  react {  
    ...  
    react { /*schedule the block; throw CONTINUE*/  
      ...  
    }  
    //Never reached  
  }  
  //Never reached  
}  
//Never reached
```

# Dynamic Dispatch Actor

```
def actor = new DynamicDispatchActor({  
  when {BigDecimal num ->  
    println 'Received BigDecimal'}  
  when {String code ->  
    compileAndRun code }  
  when {Book book ->  
    read book }  
})
```

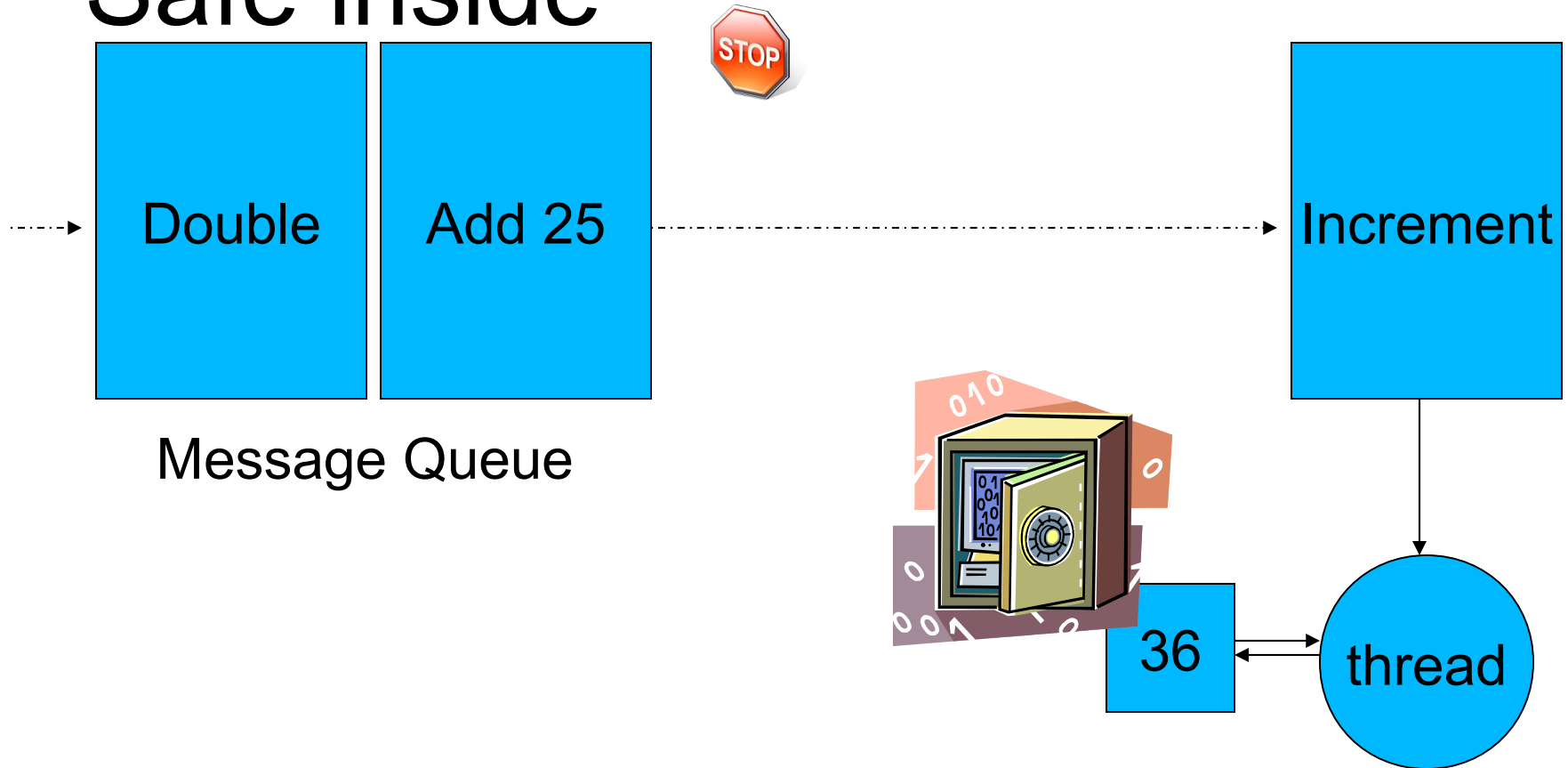
# Safe (Agent)

- Lock **Shared Mutable State** in a **Safe**





# Safe inside



# Safe List

```
def jugMembers = new Safe(['Me']) //add Me
```

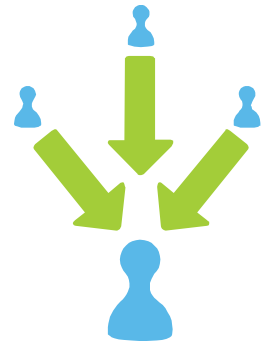
```
task {  
    jugMembers.send {it.add 'Joe'} //add Joe  
}
```

```
task {  
    jugMembers << {it.add 'Dave'} //add Dave  
    jugMembers << {it.add 'Alice'} //add Alice  
}
```

```
...  
println jugMembers.val
```

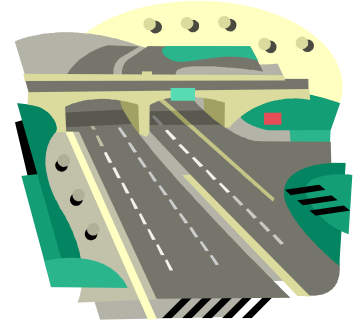
# Integration

- Bundled with Groovy dists
- Maven
- Gradle
- Grape (@Grab)
- Griffon plugin
- Grails plugin



# Roadmap

- Map/Reduce, Fork/Join
- Dataflow enhancements
- API evolution
- Actor and DataFlow remoting
- CSP?, STM?, ActiveObjects?



# Summary

Tasty concurrency menu

Actors, Collections, Dataflow, Safe, ...

Enjoyable parallelism

<http://gpars.codehaus.org>

Questions?