goto conference aarhus

APPROACHABLE CONCURRENCY FOR THE JVM WITH GPARS

Dierk König
Canoo

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

gotocon.com

Welcome!

Dierk König

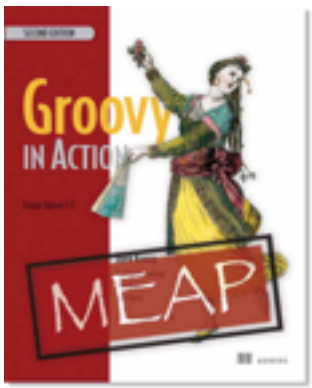
Fellow @ Canoo Engineering AG, Basel (CH)

Rich Internet Applications

Products, Projects, Consulting

www.canoo.com

Open-source committer Groovy, Grails, GPars



canoo

Groovy & GPars mission

1. Built for Java developers
2. Mend with Java
3. Make concurrency **simpler**

canoo

The Java state of affairs

Starting new threads is easy.
Some real goodies in java.util.concurrent.* & Java 7

Manual thread-coordination is difficult.
Access to shared state is error-prone.

Scheduling issues for many threads with bad concurrency characteristics.
Good use of pooling is not obvious.
Concepts are rather „low level“.

It‘s all about coordination

Fork/Join	Working on collections with fixed coordination
Map/Reduce	
Actor	Explicit coordination
Safe	Delegated coordination
Dataflow	Implicit coordination

1.8.2

canoo

It's all about coordination

Fork/Join	Working on collections with fixed coordination
Map/Reduce	
Actor	Explicit coordination
Safe	Delegated coordination
Dataflow	Implicit coordination

more

Asynchronizer
STM

1.8.2

canoo

Fork/Join on collections

```
    import static groovyx.gpars.GParsPool.withPool

def numbers = [1, 2, 3,  4,  5,  6]
def squares = [1, 4, 9, 16, 25, 36]

withPool {
 assert squares == numbers.collectParallel  { it * it }
}
```


canoo

Fork/Join on collections

```
    import static groovyx.gpars.GParsPool.withPool

def numbers = [1, 2, 3,  4,  5,  6]
def squares = [1, 4, 9, 16, 25, 36]

withPool {
 assert squares == numbers.collectParallel  { it * it }
}
```

Variation

makeTransparent()

canoo

More such methods

any { ... } collect { ... } count(filter)
each { ... } eachWithIndex{ ... }
every { ... }
find { ... } findAll { ... } findAny { ... }
fold { ... } fold(seed) { ... }
grep(filter)
groupBy { ... }
max { ... } max()
min { ... } min()
split { ... } sum()

canoo

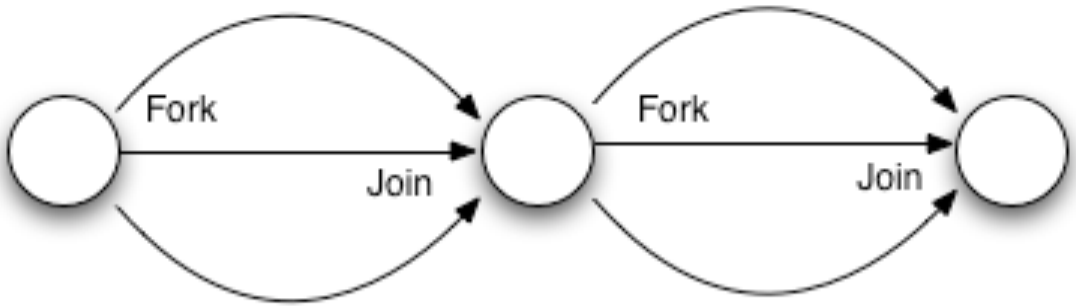
Map/Reduce on collections

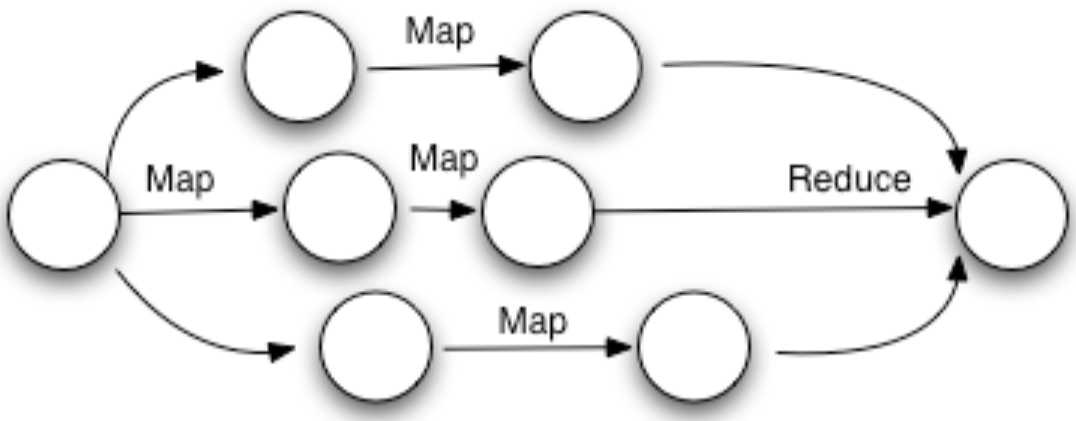
```
    import static groovyx.gpars.GParsPool.withPool

withPool {
    assert 55 == [0, 1, 2, 3, 4].parallel
        .map    { it + 1  }
        .map    { it ** 2 }
        .reduce { a, b -> a + b }
}
```


canoo

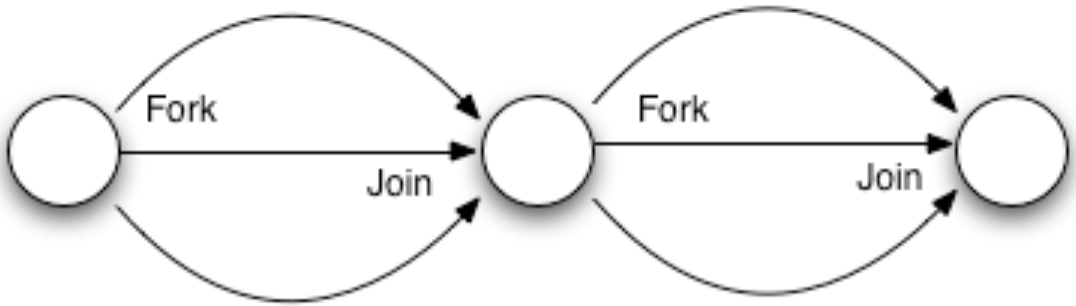
Fork/Join vs Map/Reduce



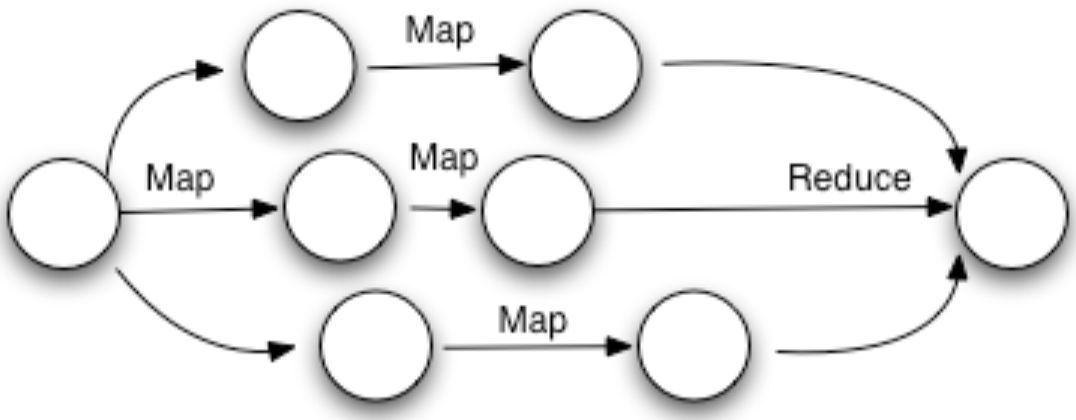


canoo

Fork/Join vs Map/Reduce



fixed coordination



canoo

Explicit coordination with Actors

```
import static groovyx.gpars.actor.Actors.*

def printer   = reactor { println it }
def decryptor = reactor { message ->
    if (message in String) reply message.reverse()
    else stop()
}
actor {
    decryptor.send    'lellarap si yvoorG'
    react {
        printer.send  'Decrypted message: ' + it
        decryptor.send false
    }
}.join()
```


canoo

Actors

Process one message at a time.

Dispatch on the message type,
which fits nicely with dynamic languages.

Are often used in composition,
which can lead to further problems down the road.

Personal note:
Actors are overrated

canoo

Delegate to an Agent

```
    import groovyx.gpars.agent.Agent

def name = new Agent<List>( ['GPars'] )

name.send { it.add 'is safe!'  }
name.send { updateValue it * 2 }

println name.val
```


canoo

Safe

Analogous to Clojure agents (atoms, refs, ...)

Implementations differ much in efficiency.

canoo

DataFlow for implicit coordination

```
import groovyx.gpars.dataflow.DataFlows
import static groovyx.gpars.dataflow.DataFlow.task

final flow = new DataFlows()
task { flow.result = flow.x + flow.y }
task { flow.x = 10 }
task { flow.y =  5 }

assert 15 == flow.result
```


canoo

DataFlow

Flavors: variables, streams, operators, tasks, flows

Write-Once, Read-Many (non-blocking)

Feel free to use millions of them

Fast, efficient, safe, and testable!

Model the flow of data,
not the control flow!

canoo

KanbanFlow in code

```
import static ProcessingNode.node
import groovyx.gpars.kanban.KanbanFlow

def producer = node { below -> below << 1 }
def consumer = node { above -> println above.take() }

new KanbanFlow().with {
    link producer to consumer
    start()
    links*.addTray()
    // run for a while
    stop()
}
```


canoo

Efficient Producer-Consumer

KanbanFlow pattern by /me

http://people.canoo.com/mittie/kanbanflow.html

Simple idea, amazing results

Resource efficient, composable, testable

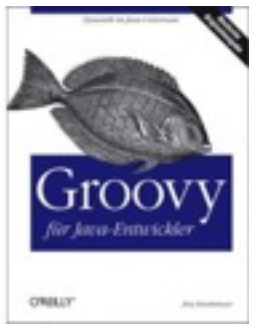
Non-blocking writes,
Deadlock-free by design

Takeaways

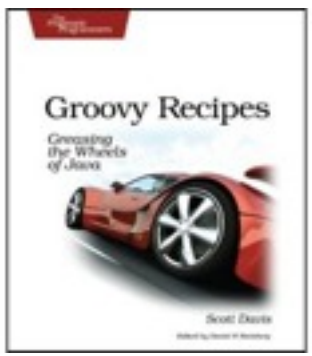
1. Experiment with GPars!
2. Great for learning concepts!
3. Get involved!

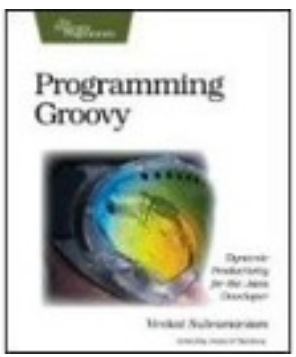
canoo

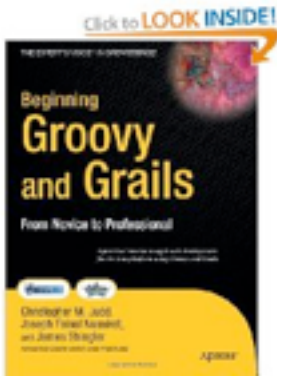
Further reading

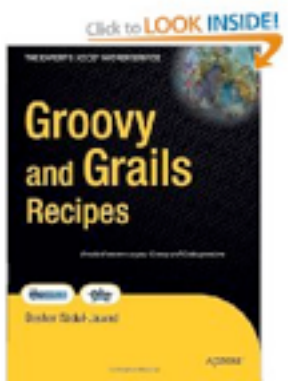


- **Groovy in Action** groovy.canoo.com/gina
  Manning, 2007, Foreword by James Gosling
  König with Glover, Laforge, King, Skeet
- groovy.codehaus.org
  gpars.codehaus.org









canoo

canoo

canoo

Discussion

credits:
Paul King

Discussion

dierk.koenig@canoo.com
@mittie

credits:
Paul King

canoo