Reactive Programming with Algebra

André van Delft
Anatoliy Kmetyuk

LambdaDays Krakow
27 February 2-15

Overview

- Introduction
  - Programming is Still Hard
  - Some History
  - Algebra of Communicating Processes
- SubScript
  - Example applications
  - . Debugger demo
- Dataflow
  - Twitter Client
  - SubScript Actors
- Conclusion

Programming is Still Hard

Mainstream programming languages: imperative

- good in batch processing
- not good in parsing, concurrency, event handling
- Callback Hell

Neglected idioms

- Non-imperative choice: BNF, YACC
- Data flow: Unix pipes

Math!

Algebra can be easy and fun

Area	Objects	Operations	Rules
Numbers	0, 1, ..., x, y, ...	$+ \;\cdot\; - \;/$	$x+y = y+x$
Logic	F, T, x, y, ...	$\vee \;\wedge\; \neg$	$x \vee y = y \vee x$
Processes	0, 1, a, b,..., x, y, ...	$+ \;\cdot\; \& \;\mid\; \&\& \;\parallel\; /$	$x+y = y+x$

Some History

1955	Stephen Kleene	~~> regular expressions, *
	Noam Chomsky	~~> language grammars
1960	John Backus & Peter Naur	~~> BNF
	Tony Brooker	~~> Compiler Compiler
1971	Hans Bekič	~~> Algebra of Processes
1973	Stephen Johnson	~~> YACC
1974	Nico Habermann & Roy Campbell	~~> Path Expressions
1978	Tony Hoare	~~> Communicating Sequential Processes (CSP)
1980	Robin Milner	~~> Calculus of Communicating Systems (CCS)
1982	Jan Bergstra & Jan Willem Klop	~~> Algebra of Communicating Processes (ACP)
1989	Robin Milner	~~> Pi-Calculus
	Henk Goeman	~~> Self-applicative Processes

Algebra of Communicating Processes - 1

Bergstra & Klop, Amsterdam, 1982 - ...

ACP ~ Boolean Algebra

$+$	choice
$\cdot$	sequence
0	deadlock
1	empty process

atomic actions a,b,…
parallelism
communication
disruption, interruption
time, space, probabilities
money
...

Algebra of Communicating Processes - 2

Less known than CSP, CCS

Specification & Verification

- Communication Protocols
- Production Plants
- Railways
- Coins and Coffee Machines
- Money and Economy

Strengths

- Familiar syntax
- Precise semantics
- Reasoning by term rewriting
- Events as actions

Algebra of Communicating Processes - 3

$$x+y = y+x$$

$$(x+y)+z = x+(y+z)$$

$$x+x = x$$

$$\boxed{(x+y)\cdot z = x\cdot z+y\cdot z}$$

$$(x\cdot y)\cdot z = x\cdot(y\cdot z)$$

$$0+x = x$$

$$0\cdot x = 0$$

$$\boxed{1\cdot x = x}$$

$$x\cdot 1 = x$$

$$\begin{aligned}(x+1)\cdot y &= x\cdot y + 1\cdot y\\ &= x\cdot y + y\end{aligned}$$

Algebra of Communicating Processes - 4

$$x \parallel y = x \mathbin{|\!\lfloor} y + y \mathbin{|\!\lfloor} x + x \mid y$$

$$\begin{aligned}
(x+y) \mathbin{|\!\lfloor} z &= \ldots \\
a \cdot x \mathbin{|\!\lfloor} y &= \ldots \\
1 \mathbin{|\!\lfloor} x &= \ldots \\
0 \mathbin{|\!\lfloor} x &= \ldots
\end{aligned}$$

$$\begin{aligned}
(x+y) \mid z &= \ldots \\
\ldots &= \ldots
\end{aligned}$$

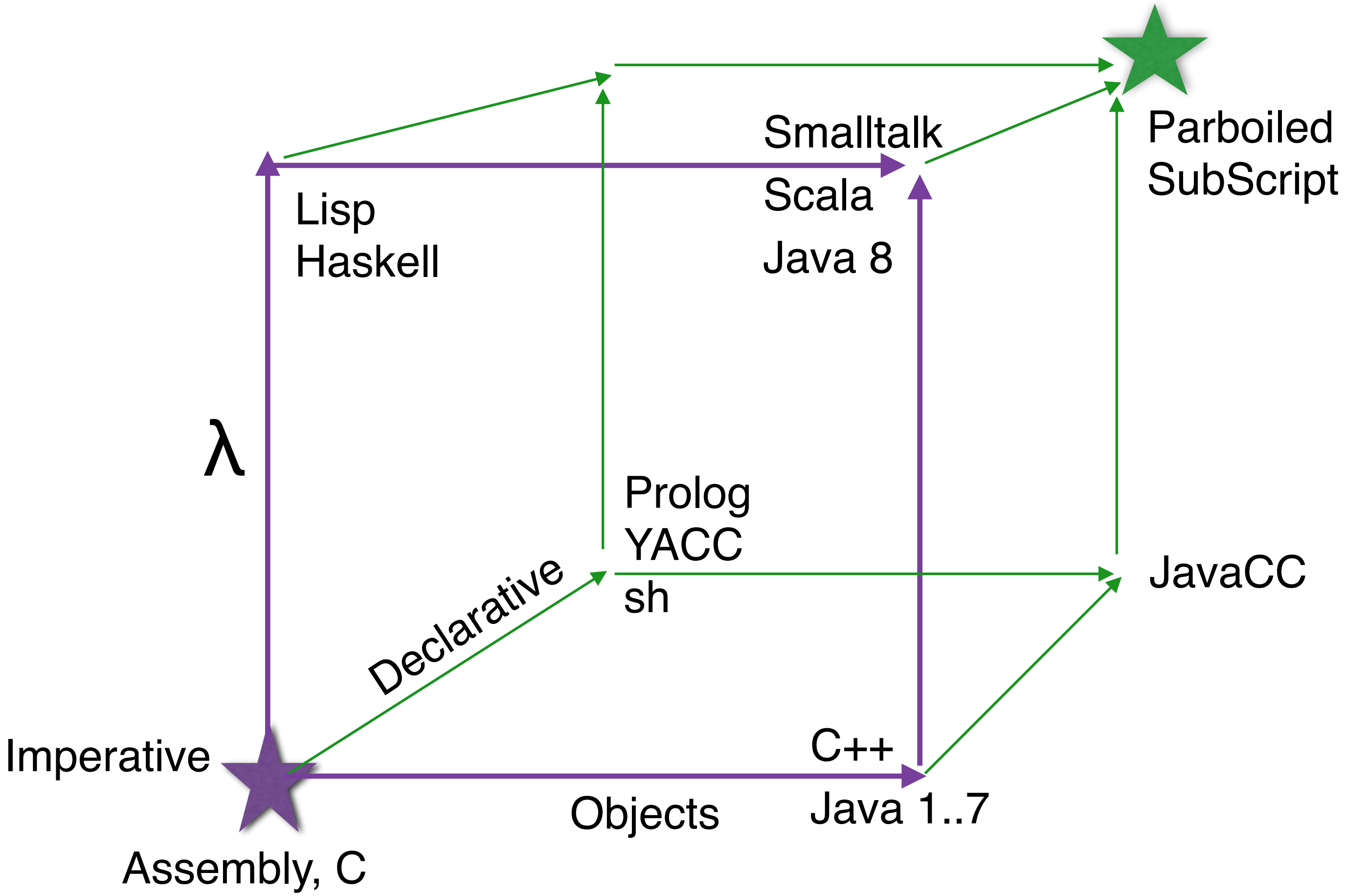
ACP Language Extensions

- 1980: Jan van den Bos - Input Tool Model [Pascal, Modula-2]
- 1988-2011: André van Delft - Scriptic [Pascal, Modula-2, C, C++, Java]
- 1994: Jan Bergstra & Paul Klint - Toolbus
- 2011-...: André van Delft - SubScript [Scala, JavaScript (?)]

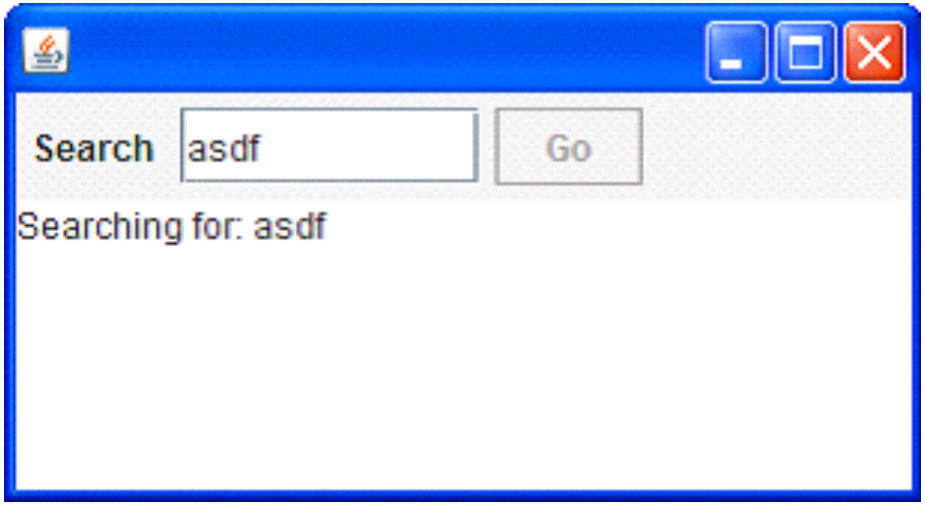
Application Areas:

- GUI Controllers
- Text Parsers
- Discrete Event Simulation
- Reactive, Actors, Dataflow

Programming Paradigms

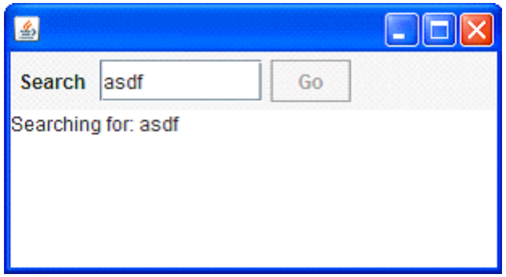


GUI application - 1



Searching for: asdf

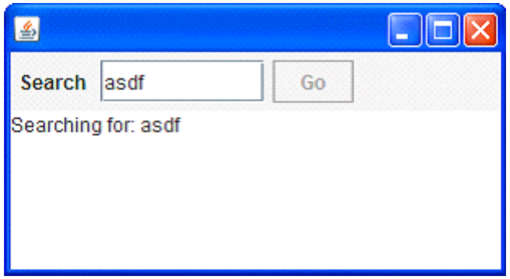
- Input Field
- Search Button
- Searching for…
- Results



Searching for: asdf

GUI application - 2

```
val searchButton = new Button("Go”) {
  reactions.+= {
    case ButtonClicked(b) =>
      enabled = false
      outputTA.text = "Starting search...”
      new Thread(new Runnable {
       def run() {
        Thread.sleep(3000)
        SwingUtilities.invokeLater(new Runnable{
          def run() {outputTA.text="Search ready”
                     enabled = true
        }})
      }}).start
  }
}
```



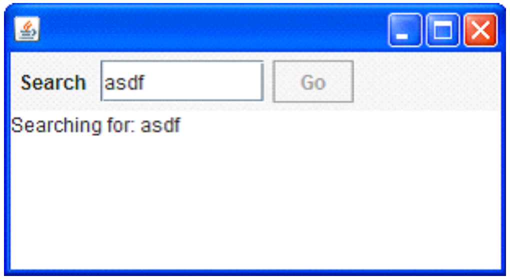
asdf

Searching for: asdf

GUI application - 3

```
live =       searchButton
        @gui: {outputTA.text="Starting search.."}
              {* Thread.sleep(3000) *}
        @gui: {outputTA.text="Search ready"}
              ...
```

- Sequence operator: white space and ;
- gui: code executor for
  - SwingUtilities.InvokeLater+InvokeAndWait
- {* … *}: by executor for new Thread

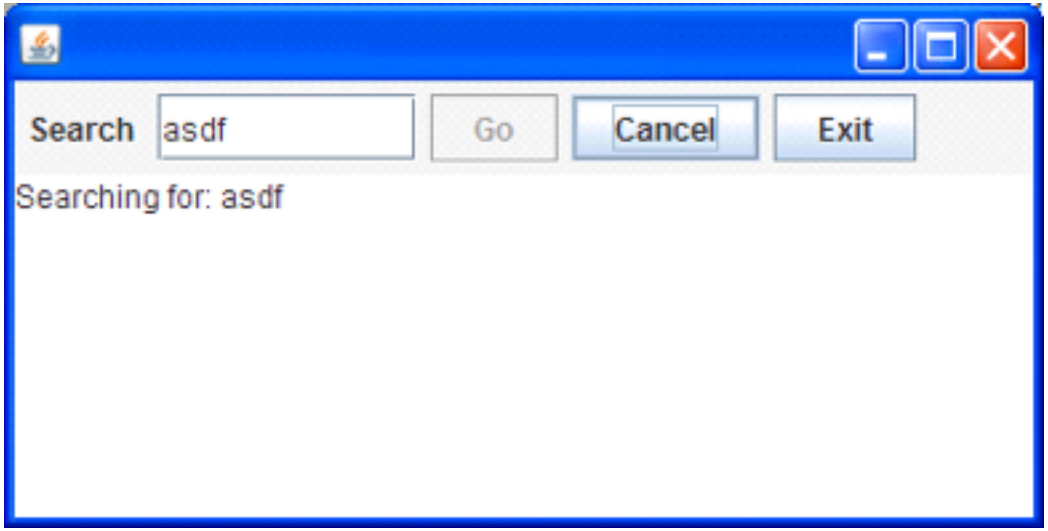


Searching for: asdf

GUI application - 4

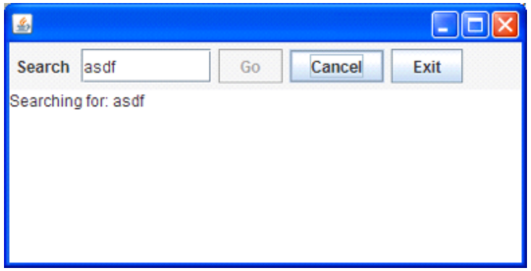
```
live               = searchSequence...

searchSequence     = searchCommand
                     showSearchingText
                     searchInDatabase
                     showSearchResults


searchCommand      = searchButton
showSearchingText  = @gui: {outputTA.text = "…"}
showSearchResults  = @gui: {outputTA.text = "…"}
searchInDatabase   = {* Thread.sleep(3000) *}
```

GUI application - 5



Searching for: asdf

- Search: button or Enter key
- Cancel: button or Escape key
- Exit: button or ☒ ; ; "Are you sure?"…
- Search only allowed when input field **not** empty
- Progress indication



Search asdf

Searching for: asdf

GUI application - 6

```
live              = searchSequence... || exit

searchCommand     = searchButton + Key.Enter
cancelCommand     = cancelButton + Key.Escape
exitCommand       =   exitButton + windowClosing☒
exit              =   exitCommand @gui:{confirmExit} ~~(b:Boolean)~~> while(!b)

cancelSearch      = cancelCommand @gui: showCanceledText

searchSequence    = searchGuard searchCommand
                    showSearchingText searchInDatabase showSearchResults
                    / cancelSearch

searchGuard       = if(!searchTF.text.isEmpty) . anyEvent(searchTF) ...

searchInDatabase  = {*Thread.sleep(3000)*} || progressMonitor
progressMonitor   = {*Thread.sleep( 250)*}
                    @gui:{searchTF.text+=here.pass} ...
```

SubScript Features

"Scripts" – process refinements as class members

```
script a = b; {c}
```

- Much like methods: override, implicit, named args, varargs, ...
- Invoked from Scala: `_execute(a, aScriptExecutor)`

  Default executor: `_execute(a)`
- Body: process expression

  Operators: `+  ;  &  |  &&  ||  /` ...

  Operands: script call, code fragment, if, while, ...
- Output parameters: ?, ...
- Shared scripts:

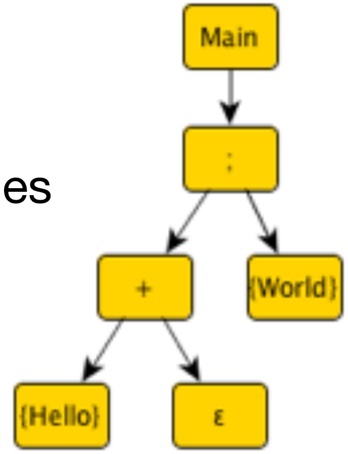
  ```
  script send,receive = {}
  ```

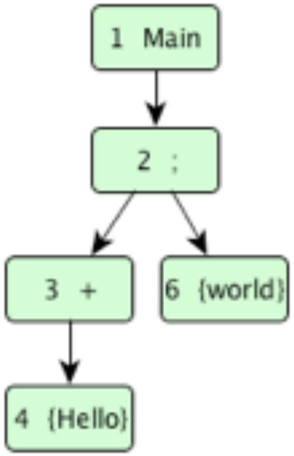
Implementation - 1

- Branch of Scalac: 1300 lines (scanner + parser + typer)

```
script Main = ({Hello} + ε); {World}

import subscript.DSL._
def Main = _script('Main) {
             _seq(_alt(_normal{here=>Hello}, _empty),
                       _normal{here=>World}          )
           }
```

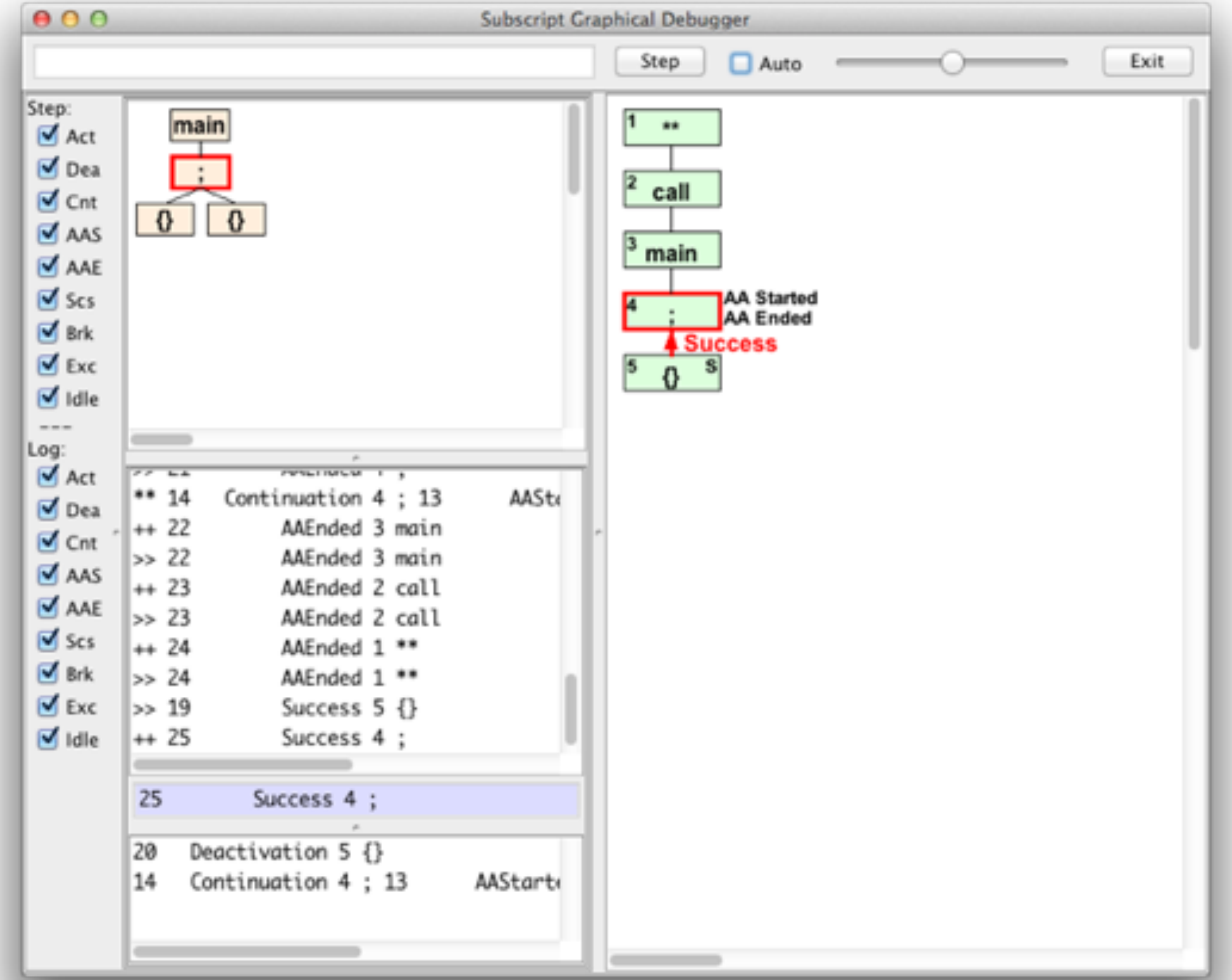
- Virtual Machine: 2000 lines
  – static script trees
  – dynamic Call Graph





- Swing event handling scripts: 260 lines
- Graphical Debugger: 550 lines (10 in SubScript)

Debugger - 1



☐ Auto

Step:
☑ Act
☑ Dea
☑ Cnt
☑ AAS
☑ AAE
☑ Scs
☑ Brk
☑ Exc
☑ Idle

Log:
☑ Act
☑ Dea
☑ Cnt
☑ AAS
☑ AAE
☑ Scs
☑ Brk
☑ Exc
☑ Idle

```
** 14   Continuation 4 ; 13        AASt
++ 22        AAEnded 3 main
>> 22        AAEnded 3 main
++ 23        AAEnded 2 call
>> 23        AAEnded 2 call
++ 24        AAEnded 1 **
>> 24        AAEnded 1 **
>> 19        Success 5 {}
++ 25        Success 4 ;
```

```
25          Success 4 ;
```

```
20   Deactivation 5 {}
14   Continuation 4 ; 13        AAStart
```

Debugger - 2

built using SubScript

```
live        = stepping || exit

stepping    = {* awaitMessageBeingHandled(true) *}
              if shouldStep then (
                 @gui: {! updateDisplay !}
                 stepCommand || if autoCheckBox.selected then sleepStepTimeout
              )
              { messageBeingHandled(false) }
              ...

    exit        = exitCommand
                  var     isSure = false
                  @gui: { isSure = confirmExit }
                  while (!isSure)

exitCommand = exitButton + windowClosing
```

One-time Dataflow - 1

```
exit = exitCommand
       var     isSure = false
       @gui: { isSure = confirmExit }
       while (!isSure)
```

Arrows + λ's

```
exit = exitCommand  @gui:{confirmExit} ~~> r:Boolean => [while(!r)]

exit = exitCommand  @gui:{confirmExit} ~~> r:Boolean ==> while(!r)

exit = exitCommand  @gui:{confirmExit} ~~> while(!_)

exit = exitCommand  @gui:{confirmExit} ~~(r:Boolean)~~> while(!r)
```

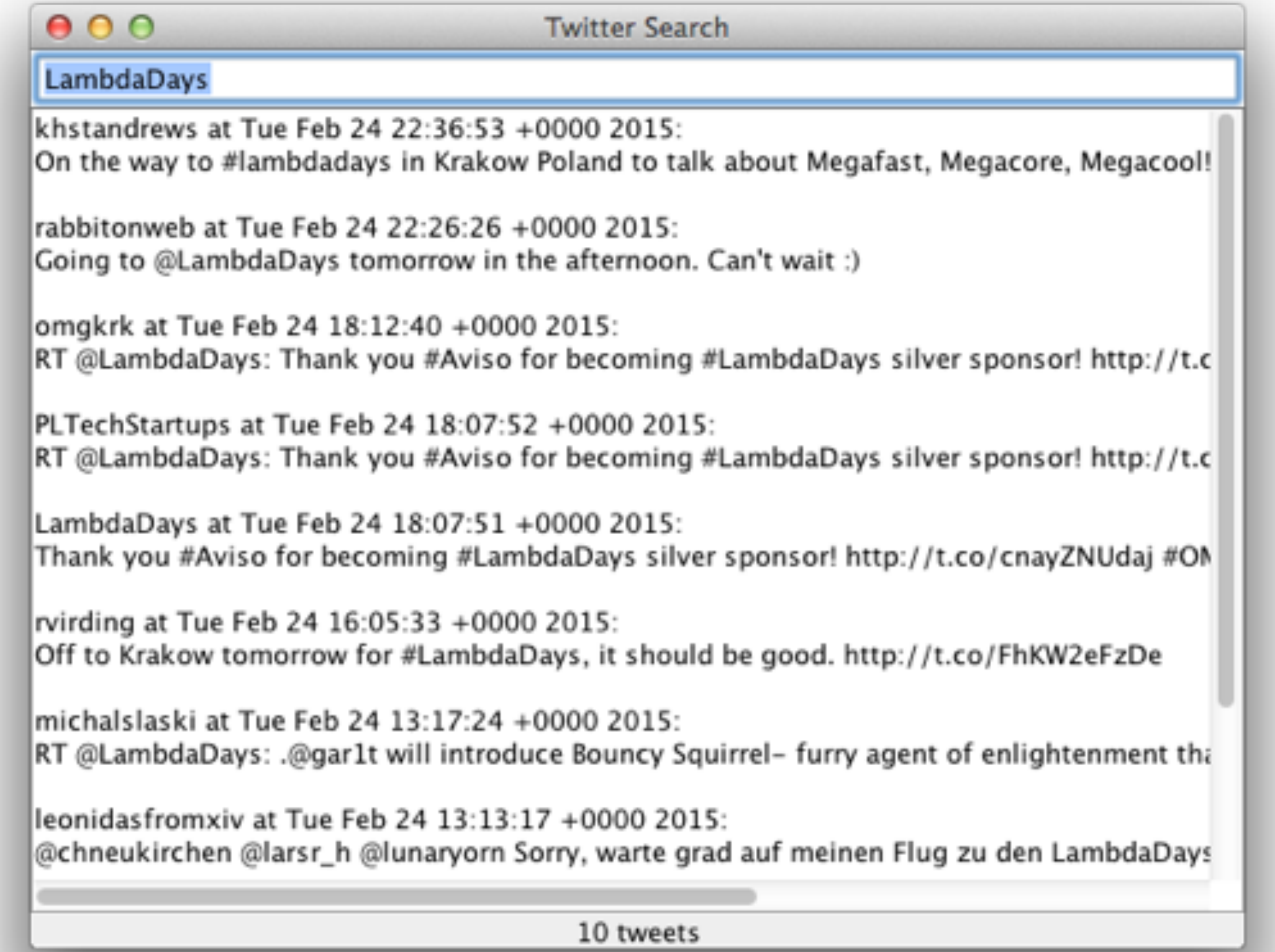
One-time Dataflow - 2

- Script result type `script confirmExit:Boolean = ...`
- Result values `$: Try[T]`
- Result propagation `call^     {result}^`
- Data Flow `x ~~> y`
- Exception Flow `x~/~> y`
- Ternary `x ~~> y +~/~> z`

- Matching flow:

```
x ~~(b:Boolean      )~~> y1
 +~~(i:Int if i<10)~~> y2
 +~~( _             )~~> y3
+~/~(e:IOException)~~> z1
+~/~(e:  Exception)~~> z2
+~/~(e:  Throwable)~~> z3
```

Example: Twitter Search Client - 1



```
khstandrews at Tue Feb 24 22:36:53 +0000 2015:
On the way to #lambdadays in Krakow Poland to talk about Megafast, Megacore, Megacool!

rabbitonweb at Tue Feb 24 22:26:26 +0000 2015:
Going to @LambdaDays tomorrow in the afternoon. Can't wait :)

omgkrk at Tue Feb 24 18:12:40 +0000 2015:
RT @LambdaDays: Thank you #Aviso for becoming #LambdaDays silver sponsor! http://t.c

PLTechStartups at Tue Feb 24 18:07:52 +0000 2015:
RT @LambdaDays: Thank you #Aviso for becoming #LambdaDays silver sponsor! http://t.c

LambdaDays at Tue Feb 24 18:07:51 +0000 2015:
Thank you #Aviso for becoming #LambdaDays silver sponsor! http://t.co/cnayZNUdaj #OM

rvirding at Tue Feb 24 16:05:33 +0000 2015:
Off to Krakow tomorrow for #LambdaDays, it should be good. http://t.co/FhKW2eFzDe

michalslaski at Tue Feb 24 13:17:24 +0000 2015:
RT @LambdaDays: .@gar1t will introduce Bouncy Squirrel– furry agent of enlightenment tha

leonidasfromxiv at Tue Feb 24 13:13:17 +0000 2015:
@chneukirchen @larsr_h @lunaryorn Sorry, warte grad auf meinen Flug zu den LambdaDays
```

Example: Twitter Search Client - 2

```
class PureController(val view: View) extends Controller with Reactor {

  def start() = {initialize; bindInputCallback}

  def bindInputCallback = {
    listenTo(view.searchField.keys)

    val fWait   = InterruptableFuture {Thread sleep keyTypeDelay}
    val fSearch = InterruptableFuture {searchTweets}

   reactions += {case _                       => fWait  .execute()
     .flatMap   {case _                       => fSearch.execute()}
     .onComplete{case Success(tweets)         => Swing.onEDT{view. ...()}
                 case Failure(e:CancelException) => Swing.onEDT{view. ...()}
                 case Failure( e               ) => Swing.onEDT{view. ...()}
} } } }
```

Example: Twitter Search Client - 3

```
class SubScriptController(val view: View) extends Controller {
  def start() = _execute(_live())



  script..
    live         = initialize; (mainSequence/..)...

    mainSequence = anyEvent(view.searchField)
                   waitForDelay
                   searchInBG ~~(ts:Seq[Tweet])~~> updateTweetsView(ts)
                            +~/~(t: Throwable )~~> setErrorMsg(t)

    waitForDelay = {* Thread sleep keyTypeDelay *}
    searchInBG   = {* searchTweets *}

    updateTweetsView(ts: Seq[Tweet]) = @gui: {view.set...}
    setErrorMsg     (t : Throwable ) = @gui: {view.set...}

}
```

Example: Twitter Search Client - 4

```
class SubScriptController(val view: View) extends Controller {
  def start() = _execute(_live())
  val fWait   = InterruptableFuture {Thread sleep keyTypeDelay}
  val fSearch = InterruptableFuture {searchTweets}

  script..
    live         = initialize; (mainSequence/..)...

    mainSequence = anyEvent(view.searchField)
                   fWait
                   fSearch    ~~(ts:Seq[Tweet])~~> updateTweetsView(ts)
                            +~/~(t: Throwable )~~> setErrorMsg(t)



    updateTweetsView(ts: Seq[Tweet]) = @gui: {view.set...}
    setErrorMsg     (t : Throwable ) = @gui: {view.set...}

}
```

Example: Twitter Search Client - 4

```
implicit script future2script[T](f:InterruptableFuture[T]): T
= @{f.execute()
     .onComplete {case aTry => there.executeForTry(aTry)}}: {.  .}
```

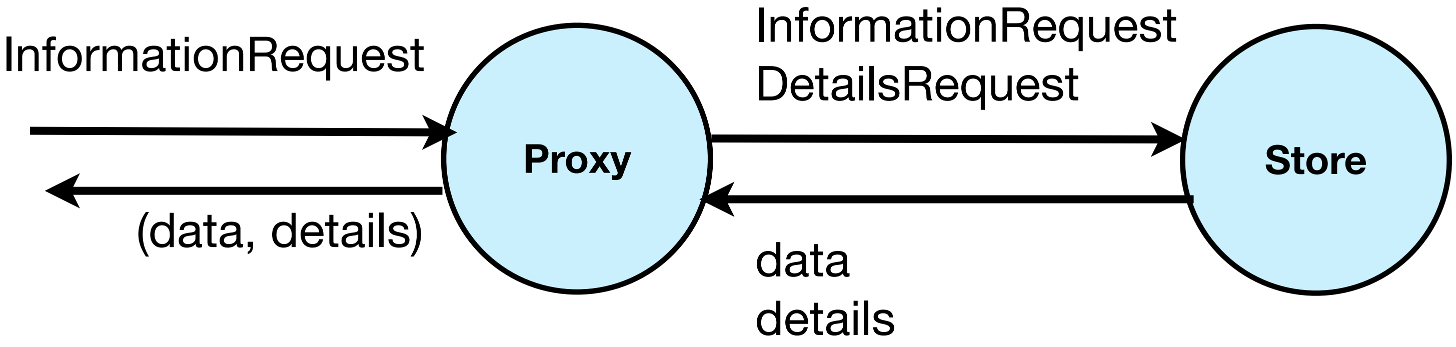
```
implicit def script2future[T](s:Script[T]): InterruptableFuture[T]
= { ... }
```

SubScript Actors: Ping Pong

```
class Ping(another: ActorRef) extends Actor {

  override def receive: PartialFunction[Any,Unit] = {case _ =>}

     another ! "Hello"
     another ! "Hello"
     another ! "Terminal"
}
```

```
class Pong extends SubScriptActor {

  implicit script str2rec(s:String) = << s >>

  script ..
    live = "Hello" ... || "Terminal" ; {println("Over")}
}
```

SubScript Actors: DataStore - 1



InformationRequest

(data, details)

Proxy

InformationRequest
DetailsRequest

data
details

Store

```
class DataStore extends Actor {

  def receive = {
    case InformationRequest(name) => sender ! getData   (name)
    case DetailsRequest    (data) => sender ! getDetails(data)
  }

}
```

SubScript Actors: DataStore - 2

```
class DataProxy(dataStore: ActorRef) extends Actor {

  def waitingForRequest = {
    case req: InformationRequest =>
      dataStore ! req
      context become waitingForData(sender)
  }

  def waitingForData(requester: ActorRef) = {
    case data: Data =>
      dataStore ! DetailsRequest(data)
      context become waitingForDetails(requester, data)
  }

  def waitingForDetails(requester: ActorRef, data: Data) = {
    case details: Details =>
      requester ! (data, details)
      context become waitingForRequest
  }
}
```

SubScript Actors: DataStore - 3

```
class DataProxy(dataStore: ActorRef) extends SubScriptActor {

  script live = << req: InformationRequest
                => dataStore ! req
               ==>
                   var response: (Data, Details) = null
                   << data: Data
                   => dataStore ! DetailsRequest(data)
                  ==>
                      << details:Details ==> response = (data,details) >>
                   >>
                   {sender ! response}
                >>
                ...
}
```

SubScript Actors: DataStore - 4

```
class DataProxy(dataStore: ActorRef) extends SubScriptActor {

  script live =
    << req: InformationRequest ==> {dataStore ? req}
                  ~~(data:Data)~~> {dataStore ? DetailsRequest(data)}
            ~~(details:Details)~~> {  sender  ! (data, details)}
    >>
    ...
}
```

SubScript Actors: Shorthand Notations

```
<< case a1: T1 => b1 ==> s1
   case a2: T2 => b2 ==> s2
   ...
   case an: Tn => bn ==> sn >>
```

```
<< case a1: T1 => b1
   case a2: T2 => b2
   ...
   case an: Tn => bn >>
```

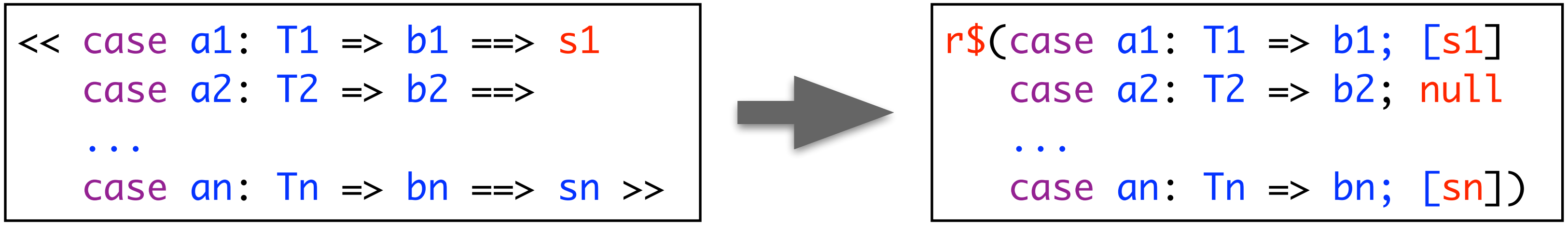
```
<< case a1: T1
   case a2: T2
   ...
   case an: Tn >>
```

```
<< case a: T => b ==> s >>

     << a: T => b ==> s >>

     << a: T => b >>

     << a: T   >>

     << 10 >>
```

SubScript Actors: Implementation - 1

```
trait SubScriptActor extends Actor {
  private val callHandlers = ListBuffer[PartialFunction[Any, Unit]]()

  def _live(): ScriptNode[Any]
  private def script terminate = Terminator.block
  private def script die       = {if (context ne null) context stop self}

  override def aroundPreStart() {
   runner.launch( [ live || terminate ; die ] )
   super.aroundPreStart()
  }

  override def aroundReceive(receive: Actor.Receive, msg: Any) {
    ...
    callHandlers.collectFirst {
      case handler if handler isDefinedAt msg => handler(msg) } match {
        case None    => super.aroundReceive( receive        , msg)
        case Some(_) => super.aroundReceive({case _: Any =>}, msg)
  } }
 ...
}
```

SubScript Actors: Implementation - 2



```
<< case a1: T1 => b1 ==> s1
   case a2: T2 => b2 ==>
   ...
   case an: Tn => bn ==> sn >>
```

```
r$(case a1: T1 => b1; [s1]
   case a2: T2 => b2; null
   ...
   case an: Tn => bn; [sn])
```

```
trait SubScriptActor extends Actor {
  ...
  script r$(handler: PartialFunction[Any, ScriptNode[Any]]) =

    var s:ScriptNode[Any]=null
    @{val handlerWithAA = handler andThen {hr => {s = hr; there.eventHappened}}
                          synchronized {callHandlers += handlerWithAA}
      there.onDeactivate {synchronized {callHandlers -= handlerWithAA}}
    }:
    {. .}
    if s != null then s
}
```

Conclusion

- Easy and efficient programming
- 10^4...10^5 actions per second
- Simple implementation: 6000 lines, 50%
  - Scalac branch ~~> Parboiled (like `ScalaTex`) + Macro's
  - VM
  - scripts for actors, swing, ..., NodeJS(?)
- Open Source:
  subscript-lang.org
  github.com/AndreVanDelft/scala
- Still much to do and to discover
- To join the project: andre.vandelft@gmail.com