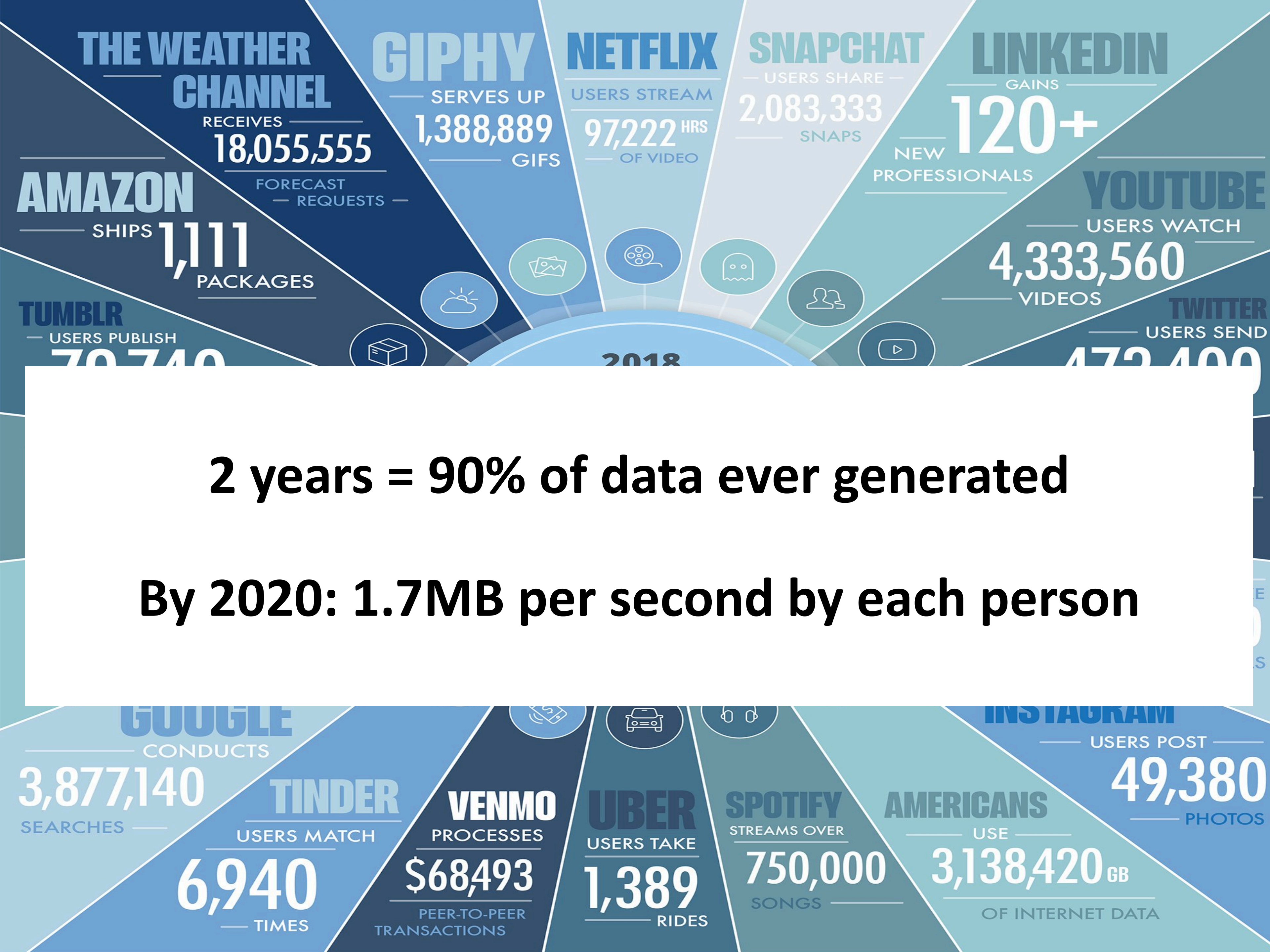
Why Languages for Distributed Systems are Inevitable*

Prof. Guido Salvaneschi

***With a tip of the hat to Jonathan Aldrich**

[Jonathan Aldrich. The power of interoperability: why objects are inevitable, Onward! 2013]



2 years = 90% of data ever generated

By 2020: 1.7MB per second by each person

Real Time Processing

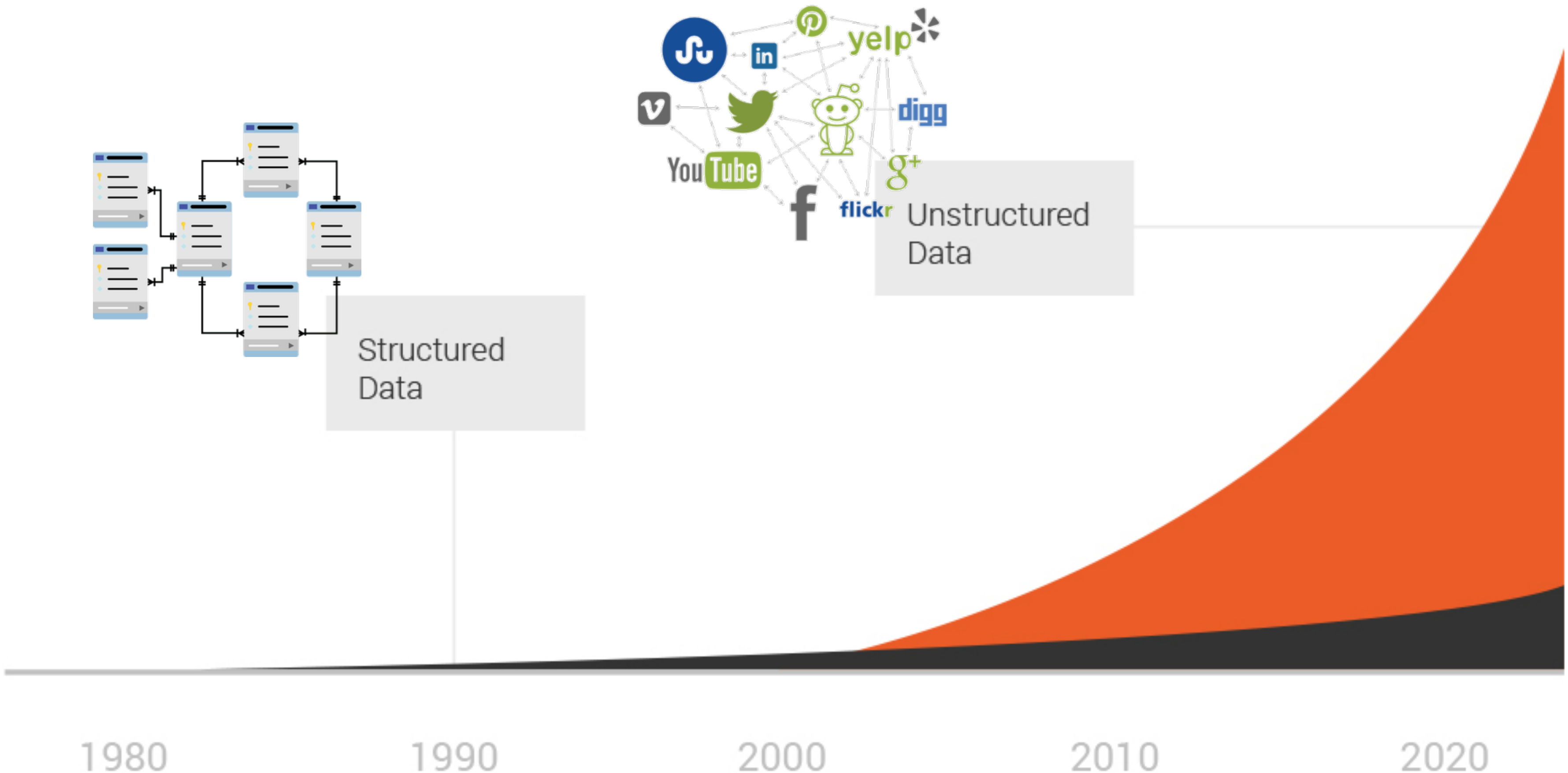
Fraud Detection

Real Time Business Intelligence

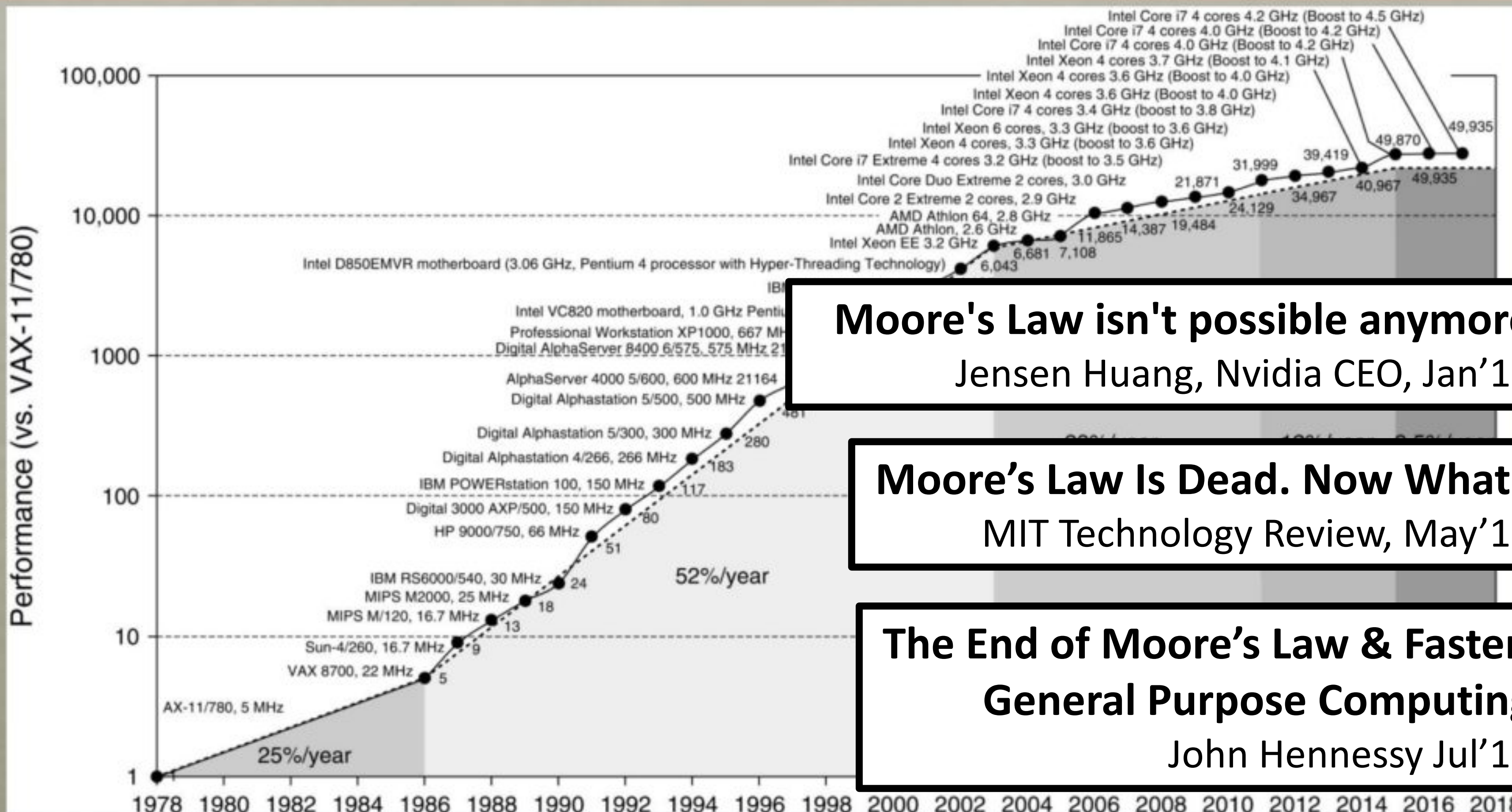
Cloud Monitoring

Structured and Unstructured Data

IDC and EMC project that data will grow to 40 ZB by 2020



UNIPROCESSOR PERFORMANCE (SINGLE CORE)



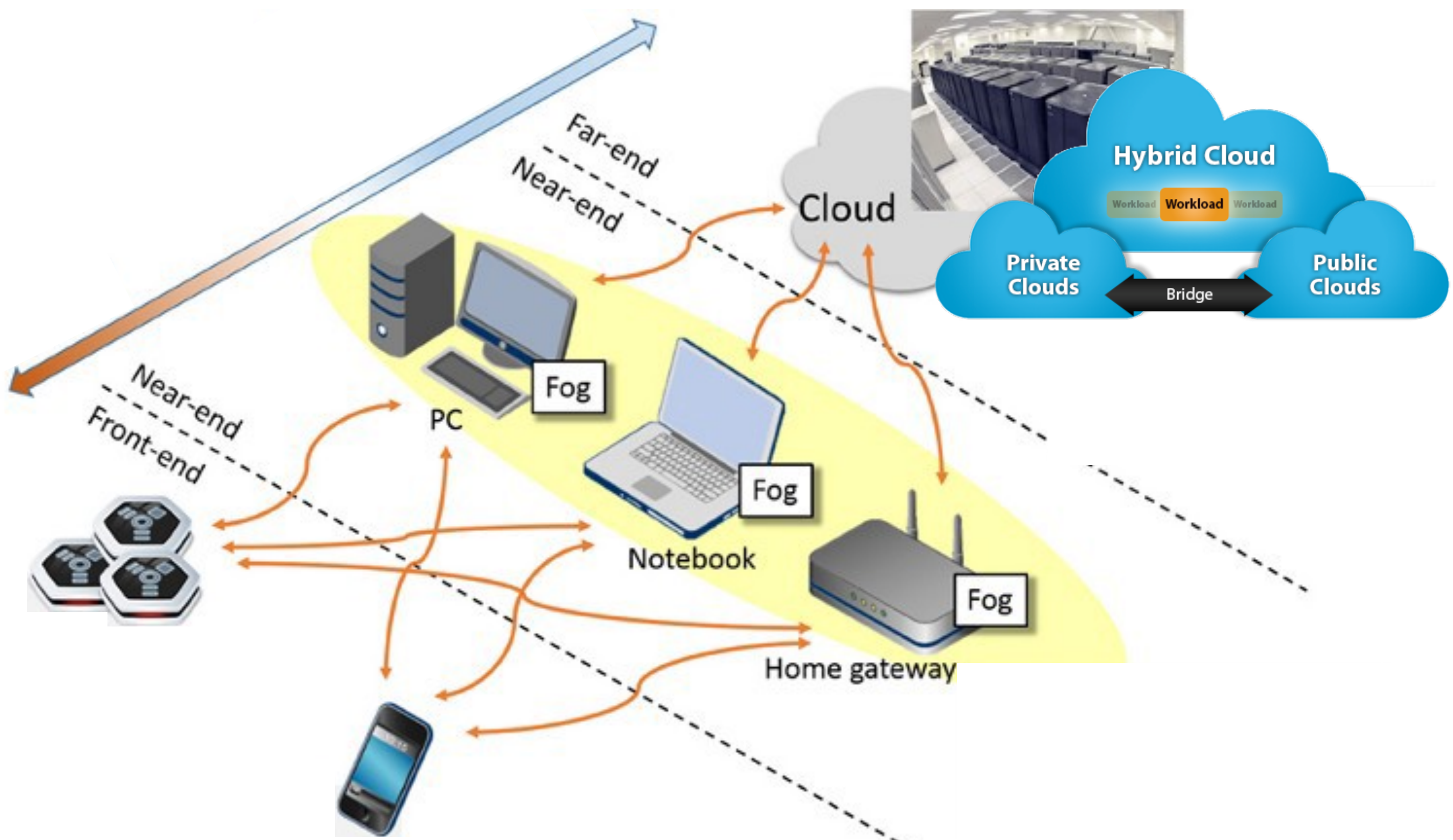
Moore's Law isn't possible anymore
Jensen Huang, Nvidia CEO, Jan'19

Moore's Law Is Dead. Now What?
MIT Technology Review, May'16

The End of Moore's Law & Faster General Purpose Computing
John Hennessy Jul'18

https://www.nextbigfuture.com/2019/02/the-end-of-moores-law-in-detail-and-starting-a-new-golden-age.html

Processing at the Edge



DOI:10.1145/2461256.2461274

Article development led by acmqueue
queue.acm.org

Building a distributed system requires a methodical approach to requirements.

BY MARK CAVAGE

There Is No Getting Around It: You Are Building a Distributed System

DISTRIBUTED SYSTEMS ARE difficult to understand, design, build, and operate. They introduce exponentially more variables into a design than a single machine does, making the root cause of an application problem

Distributed systems are difficult to understand, design, build, and operate.

They introduce exponentially more variables into a design than a single machine does, [...]

Architecting for Failure
Why are distributed systems so hard?

Markus Eisele

Lightbend

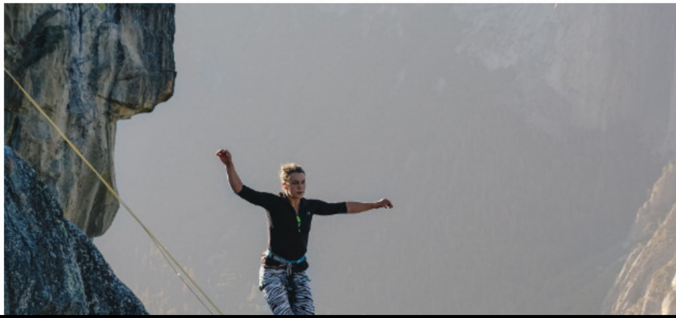
THENEWSTACK Ebooks Podcasts Events Newsletter

Architecture Development Operations

CLOUD NATIVE / CONTAINERS / MICROSERVICES / CONTRIBUTED

Distributed Systems Are Hard

25 Aug 2017 6:00am, by Anne Currie



Why Distributed Systems Are Hard to Develop — and H

Jon Edvald in Garden Follow

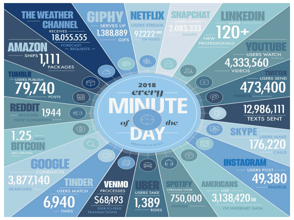
Mar 28 · 5 min read

Mot

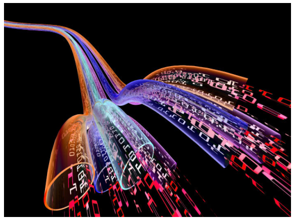
Distributed Systems: Ugly, Hard, and Here to Stay

- Developing, testing and tuning distributed applications is hard

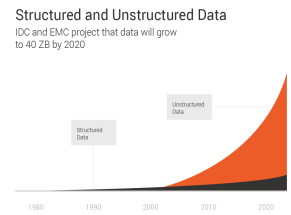
LANGUAGES FOR DISTRIBUTED APPLICATIONS



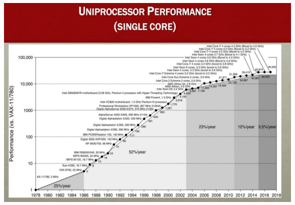
Big Data



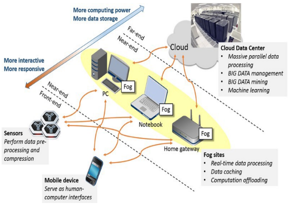
Real Time Requirements



Unstructured Data



No Moore's Law



Edge

SCALABLE

LOW LATENCY

EVENT BASED

DISTRIBUTED

HETEROGENEOUS

DATA-INTENSIVE
DISTRIBUTED APPLICATIONS

SCALABLE

LOW LATENCY

EVENT BASED

DISTRIBUTED

HETEROGENEOUS

Languages for Distributed Applications

Reactivity

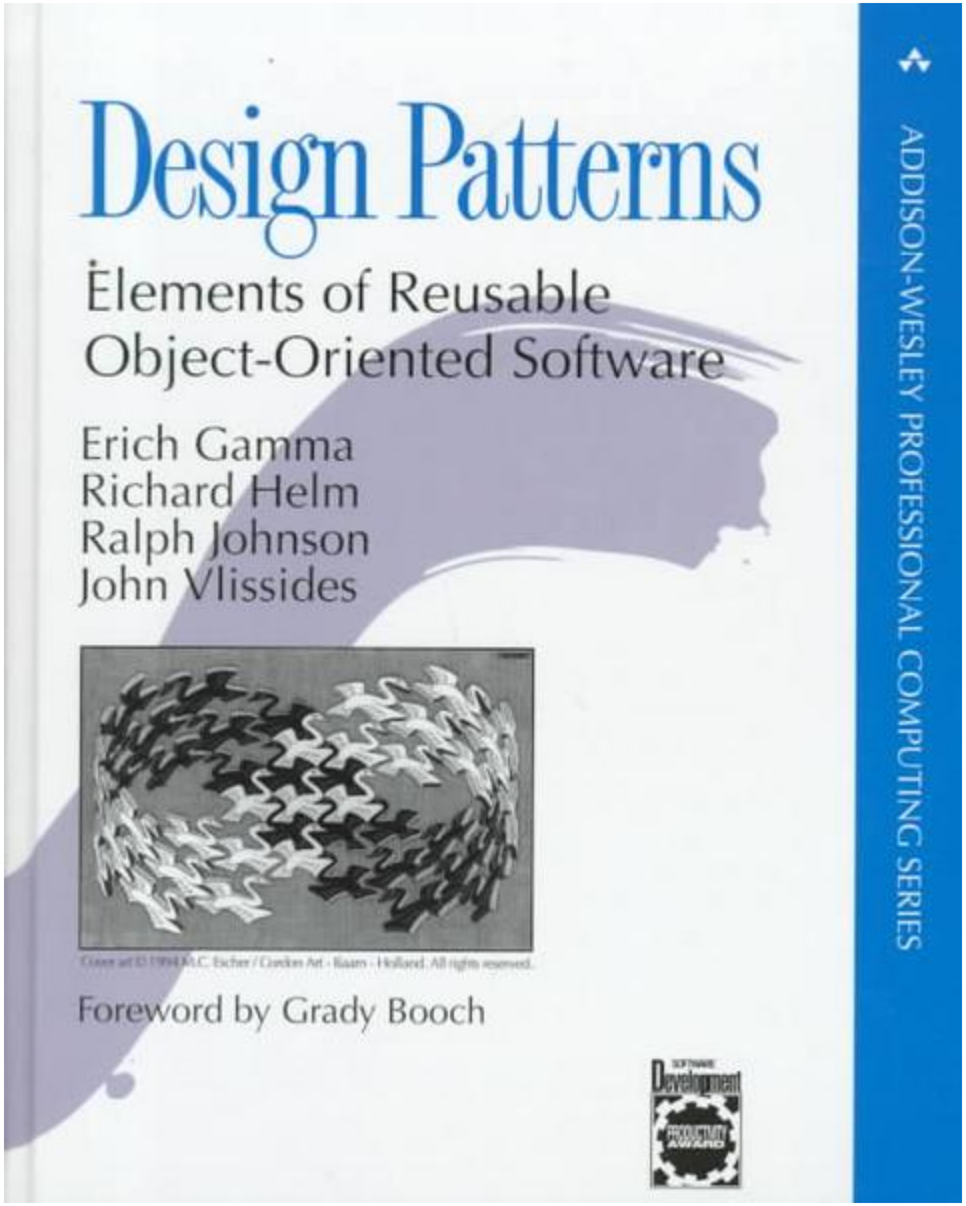
Software Design

Privacy

Fault tolerance

Consistency

REACTIVITY



Observer Pattern

*"Define a one-to-many dependency between objects so that when one object changes state, **all its dependents are notified** and updated automatically."*

Is current technology enough?

```
imperative evt tick[Unit]
var hour: Int = 0
var day: Int = 0
var week: Int = 0

tick += nextHour
def nextHour() {
  hour = (hour + 1) % 24
}
evt newDay [Unit] = tick && (() => hour == 0)
newDay += nextDay
def nextDay () {
  day = (day + 1) % 7
}
evt newWeek [Unit] = ...
newWeek += nextWeek
def nextWeek() {
  ...
}
```

EVENTS

01:12:04
ww:dd:hh

This is all what we want to express!

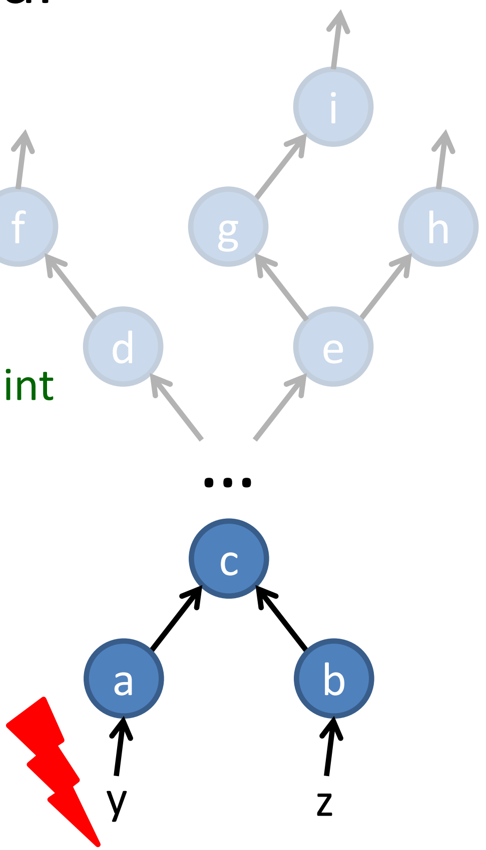
```
val tick = 0 // Increase
val hour <== tick % 24
val day <== (tick/24)%7 + 1
val week <== ...
```

REScala: Combining Signals and Events

- Signals: What about expressing functional dependencies as constraints ?

```
val a = 3
val b = 7
val c = a + b // Statement
 ...
println(c)
> 10
a= 4
println(c)
> 10
```

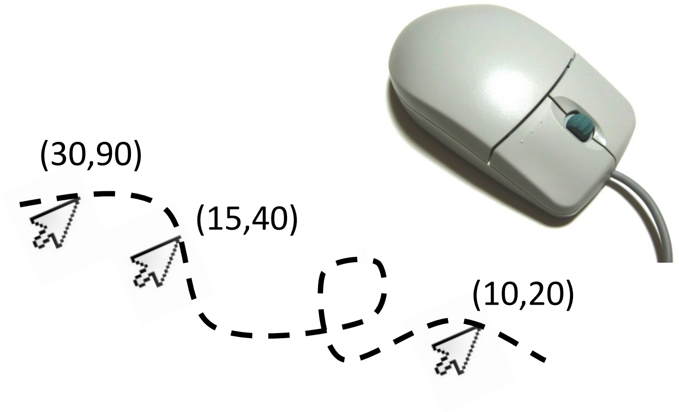
```
val a = Var(3)
val b = Var(7)
val c = Signal{ a + b } // Constraint
 ...
println(c)
> 10
a= 4
println(c)
> 11
```



www.rescala-lang.com

Reactive Programming: Example

- Mixing signals and events
- Reactive code is simple!



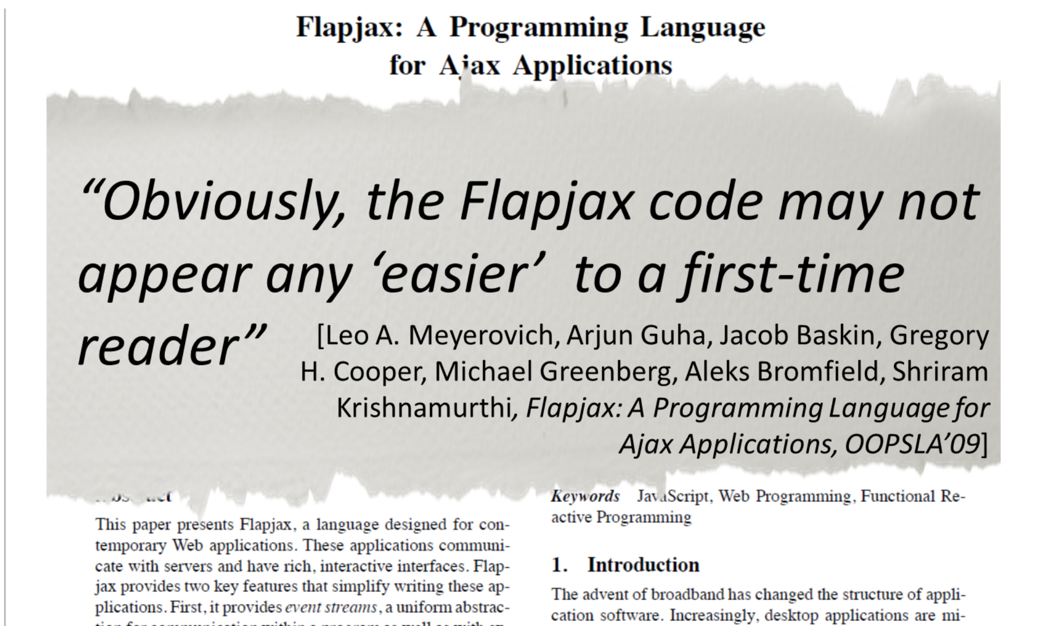

```
val position: Signal[(Int,Int)] = mouse.position
val shiftedPosition: Signal[(Int,Int)] = Signal{ mouse.position + (10, 10) }

evt clicked: Event[Unit] = mouse.clicked
val lastClick: Signal[(Int,Int)] = position snapshot clicked
```

OO integration: Both signals and events are subject to inheritance and runtime polymorphism!

[Guido Salvaneschi, Mira Mezini, **Reactive Behavior in Object-oriented Applications: An Analysis and a Research Roadmap**, MODULARITY'13]

Claim: RP beats OO (Observer)

- Easier to compose
- Declarative style
- **Easier program comprehension**
- State management not explicit
- Automatic memory management



Flapjax: A Programming Language for Ajax Applications

"Obviously, the Flapjax code may not appear any 'easier' to a first-time reader" [Leo A. Meyerovich, Arjun Guha, Jacob Baskin, Gregory H. Cooper, Michael Greenberg, Aleks Bromfield, Shriram Krishnamurthi, *Flapjax: A Programming Language for Ajax Applications, OOPSLA'09*]

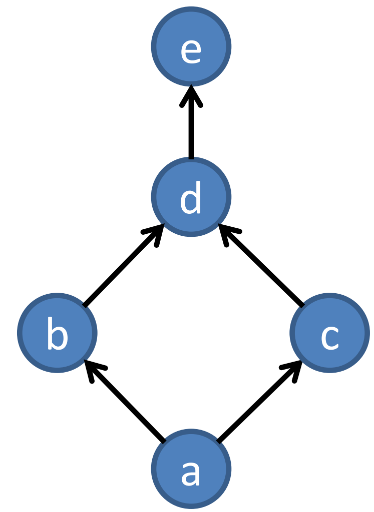
This paper presents Flapjax, a language designed for contemporary Web applications. These applications communicate with servers and have rich, interactive interfaces. Flapjax provides two key features that simplify writing these applications. First, it provides *event streams*, a uniform abstraction for communication within a program as well as with ex-

Keywords JavaScript, Web Programming, Functional Reactive Programming

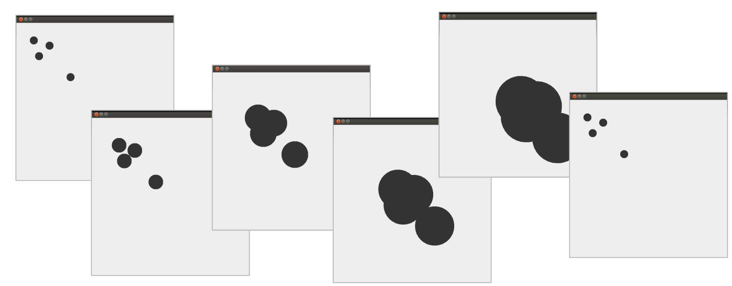
1. Introduction

The advent of broadband has changed the structure of application software. Increasingly, desktop applications are migrating to the Web. Programs that once made brief forays

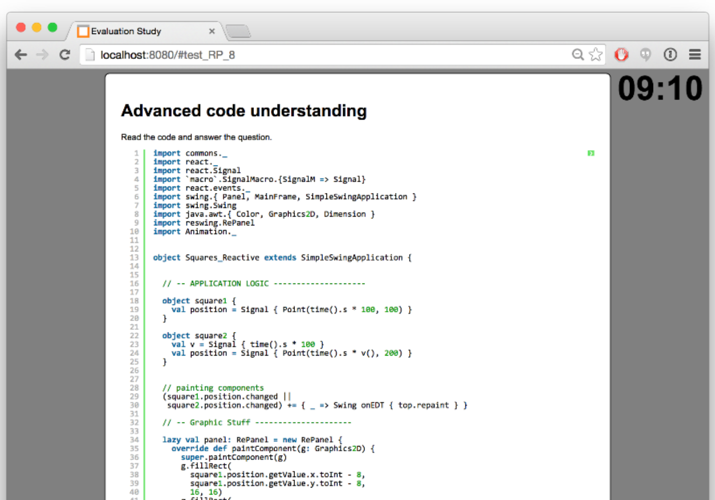
The study



- 10 applications, ~130 subjects
  - RP and OO group (**between** subj.)
  - Questions for comprehension



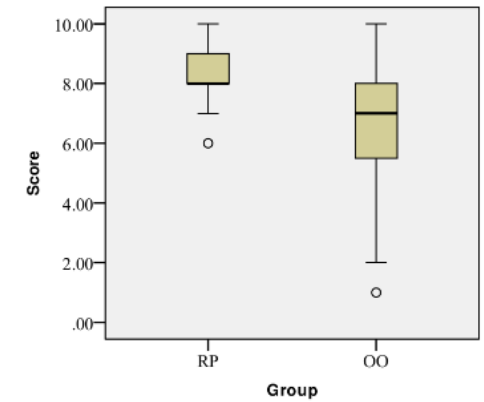
- What to measure?
  - **Time** to answer a question
  - Amount of **correct answers**

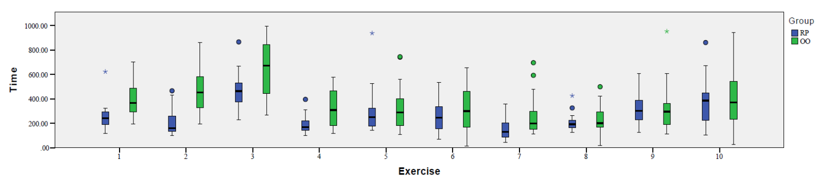


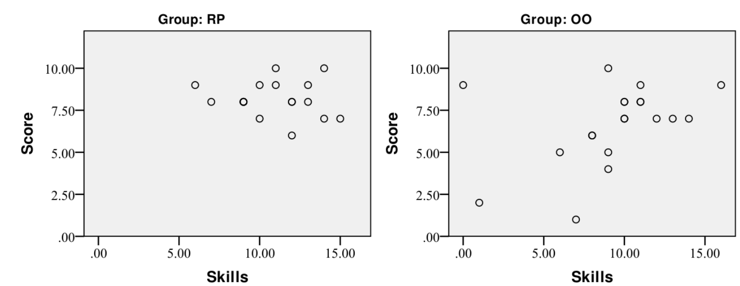
Results

RESscala **increases correctness** of program comprehension

In REScala, comprehension is **no more time-consuming**







[Guido Salvaneschi, Sven Amann, Sebastian Proksch, Mira Mezini, **An Empirical Study on Program Comprehension with Reactive Programming,** FSE'14.]

[G. Salvaneschi, S. Proksch, S. Amann, S. Nadi, M. Mezini, ***On the Positive Effect of Reactive Programming on Software Comprehension: An Empirical Study,*** TSE'17]

Teaching Reactive Programming

Master course (9CP)

Software Engineering: Design & Constr ction

- Design patterns
- Domain specific l ges
- Software architec

Reactive P

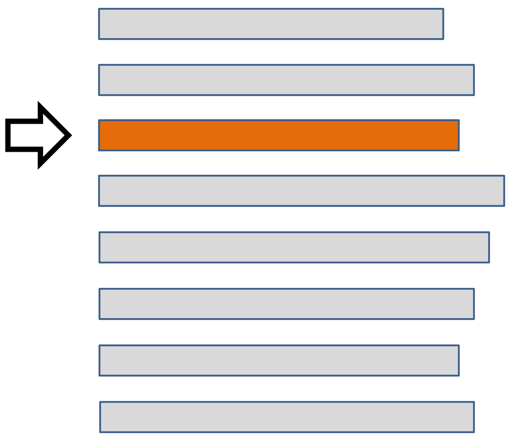
…

HOW TO DEBUG
REACTIVE PROGRAMS !?

Debugging for Reactive Programing

Traditional debugging

(Imperative)



Step over statements

```
0x051DE590  a0 e5 1d 05 4c
0x051DE5A0  20 e6 1d 05 b1
0x051DE5B0  40 e6 1d 05 00
0x051DE5C0  bc c3 94 70 b4
```

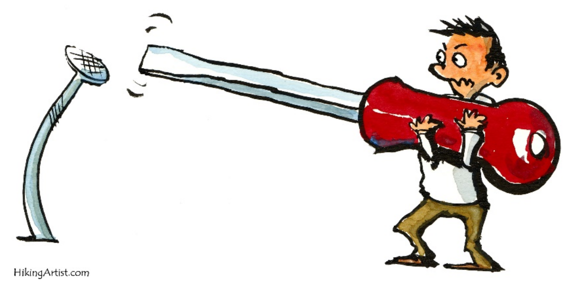
Inspect state

Program Stack

Reactive Programming

(Declarative)

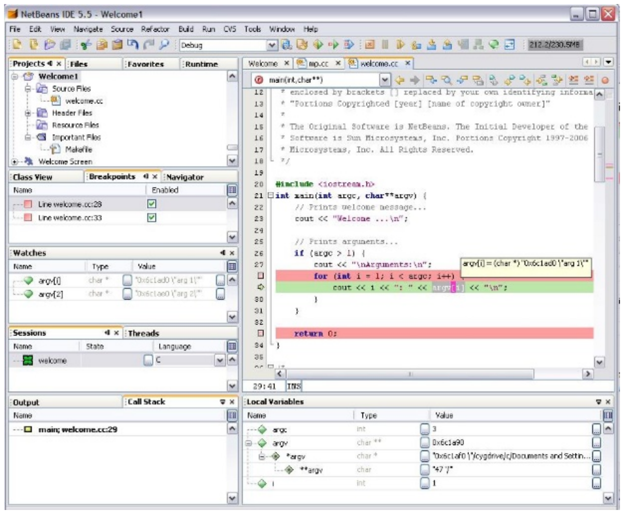
Signals

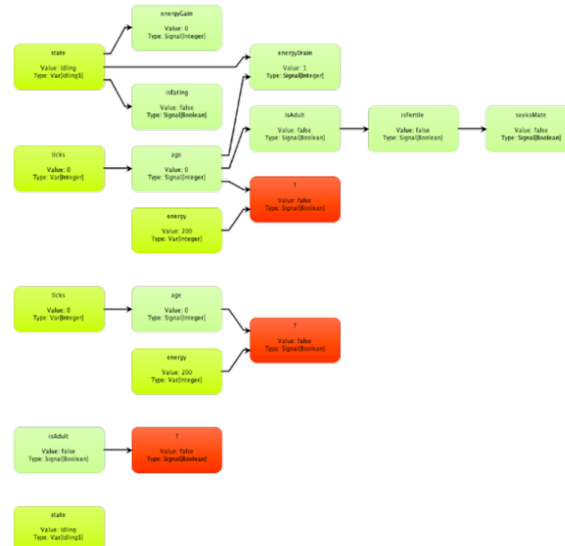


Abstract over state

!?

A Paradigm Shift

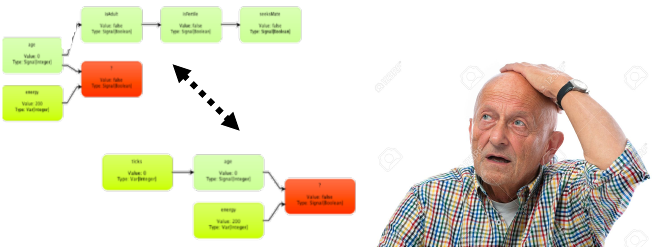




Traditional Debugging	RP Debugging
Stepping over statements	Stepping over the dependency graph
Breakpoint on line X	Breakpoint on node X
Inspect memory	Inspect values in the dependency graph
Navigate object references	Navigate signals in the graph
Per-function absolute performance	Per-node relative performance

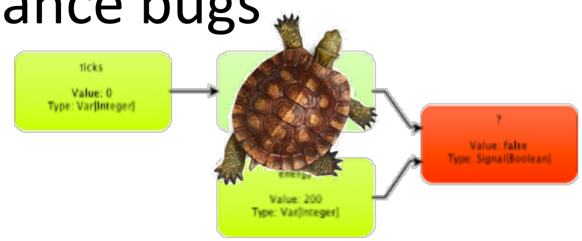
Bug Hunting with Reactive Debugging

Missing dependencies

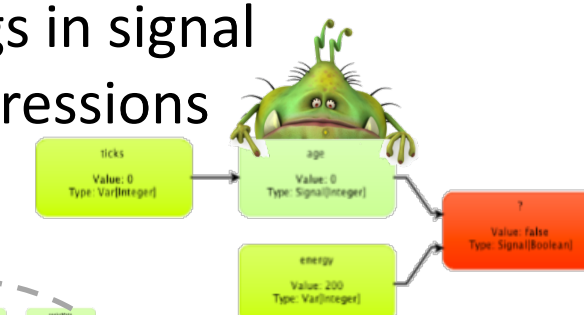


Understanding RP programs

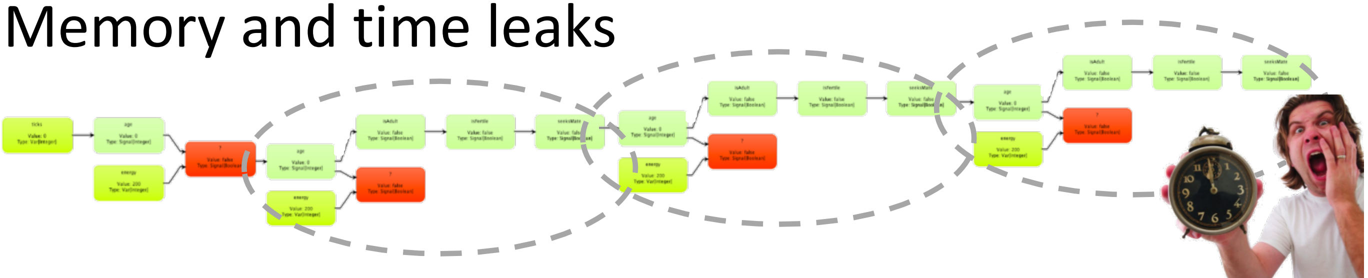
Performance bugs



Bugs in signal expressions

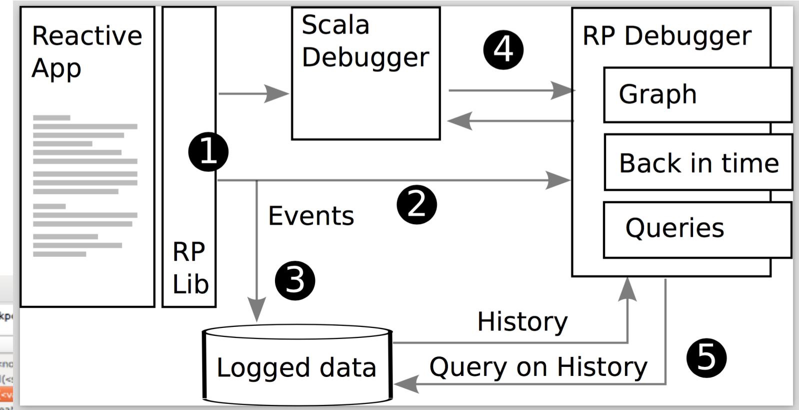


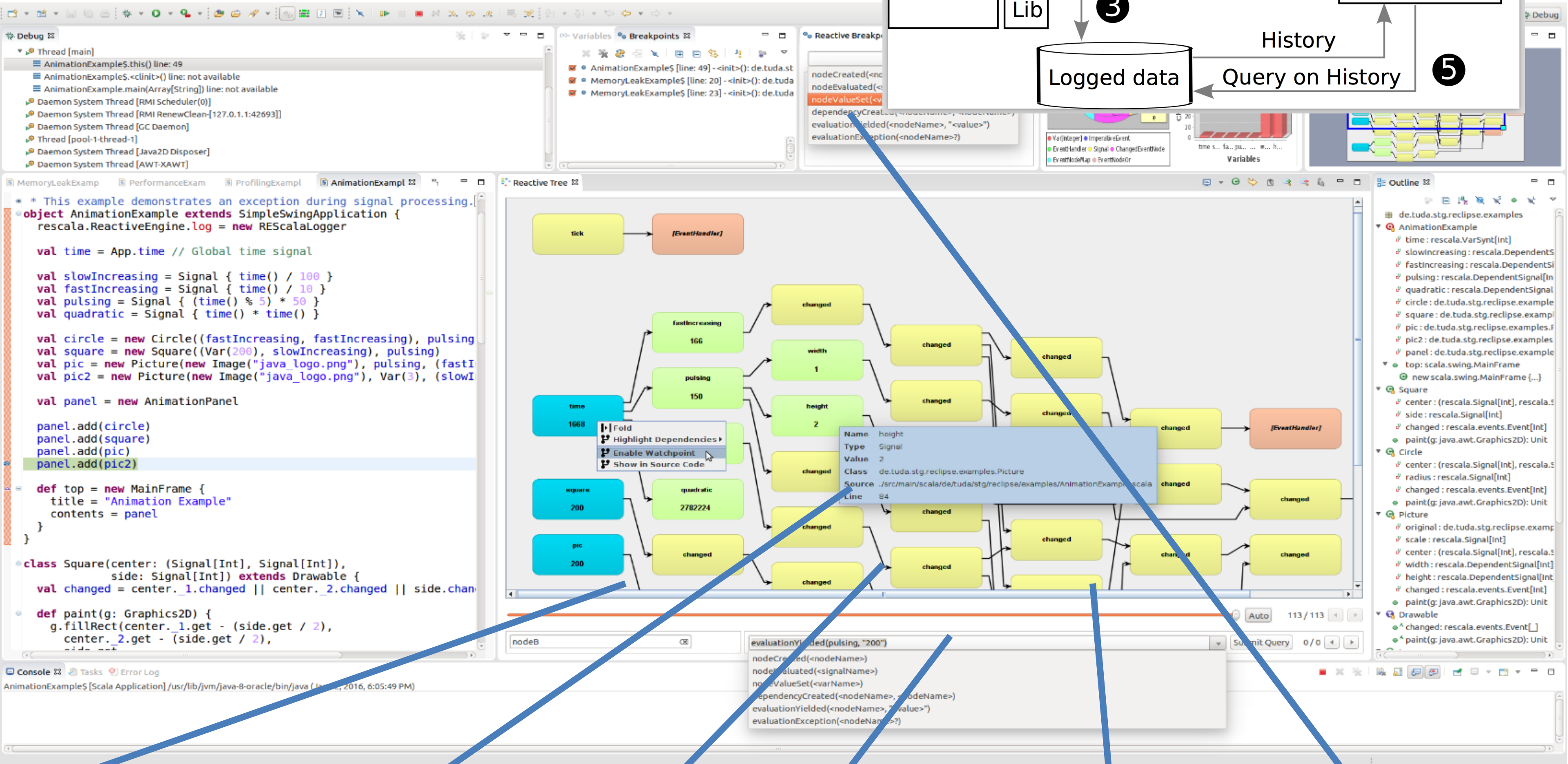
Memory and time leaks



Reactive Inspector

(Eclipse plugin - Scala IDE)





NODE SEARCH

NODE BREAKPOINTS

TREE INSPECTION

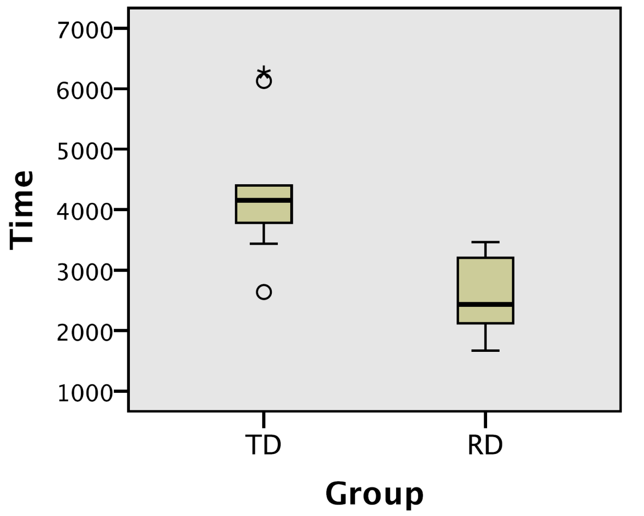
NODE QUERIES

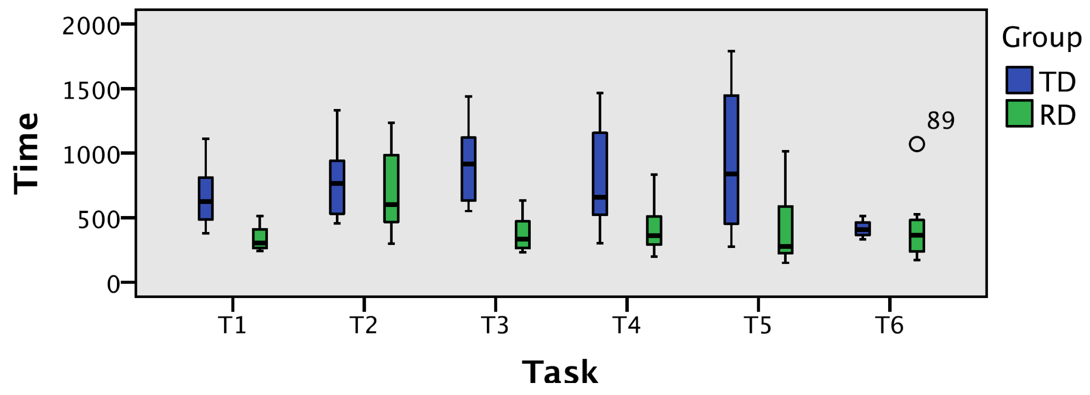
BACK-IN-TIME DEBUGGING

REACTIVE BREAKPOINTS

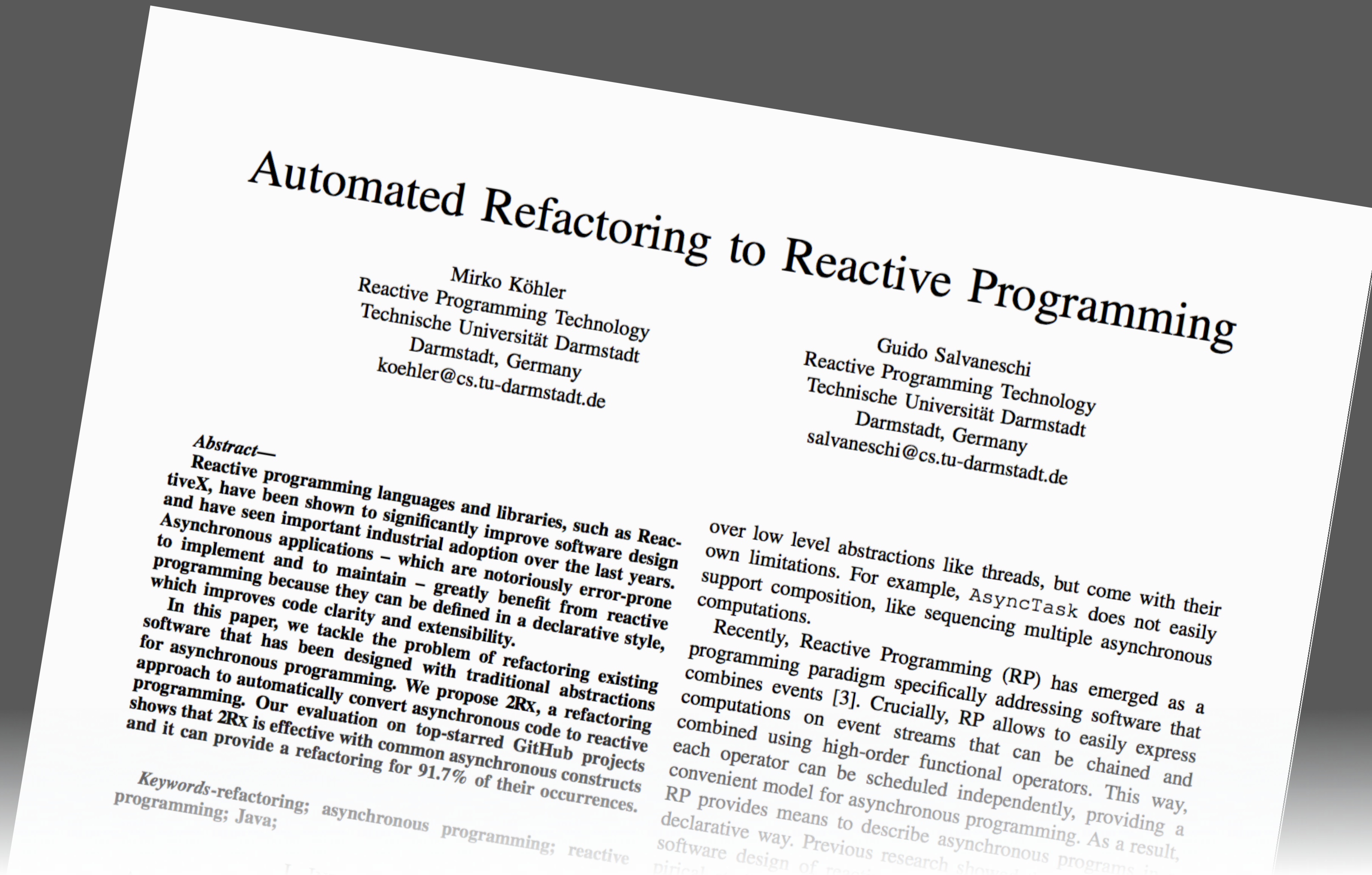
Evaluation

- 18 subjects, 2 groups
- 6 applications,
  - 2D simulation, fisheye animation, reactive network, arcade Pong, RSS Feed reader, shapes animation





[Guido Salvaneschi, Mira Mezini, **Debugging for Reactive Programming,** ICSE'16]



Automated Refactoring to Reactive Programming

Mirko Köhler
Reactive Programming Technology
Technische Universität Darmstadt
Darmstadt, Germany
koehler@cs.tu-darmstadt.de

Guido Salvaneschi
Reactive Programming Technology
Technische Universität Darmstadt
Darmstadt, Germany
salvaneschi@cs.tu-darmstadt.de

***Abstract*—Reactive programming languages and libraries, such as ReactiveX, have been shown to significantly improve software design and have seen important industrial adoption over the last years. Asynchronous applications – which are notoriously error-prone to implement and to maintain – greatly benefit from reactive programming because they can be defined in a declarative style, which improves code clarity and extensibility.**

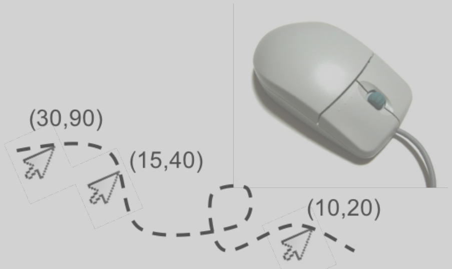
In this paper, we tackle the problem of refactoring existing software that has been designed with traditional abstractions for asynchronous programming. We propose 2Rx, a refactoring approach to automatically convert asynchronous code to reactive programming. Our evaluation on top-starred GitHub projects shows that 2Rx is effective with common asynchronous constructs and it can provide a refactoring for 91.7% of their occurrences.

Keywords-refactoring; asynchronous programming; reactive programming; Java;

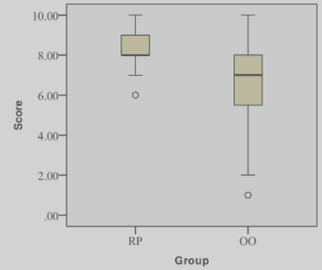
over low level abstractions like threads, but come with their own limitations. For example, `AsyncTask` does not easily support composition, like sequencing multiple asynchronous computations.

Recently, Reactive Programming (RP) has emerged as a programming paradigm specifically addressing software that combines events [3]. Crucially, RP allows to easily express computations on event streams that can be chained and combined using high-order functional operators. This way, each operator can be scheduled independently, providing a convenient model for asynchronous programming. As a result, RP provides means to describe asynchronous programs in a declarative way. Previous research showed ...

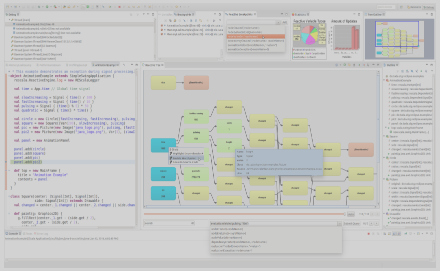
[ASE'19]



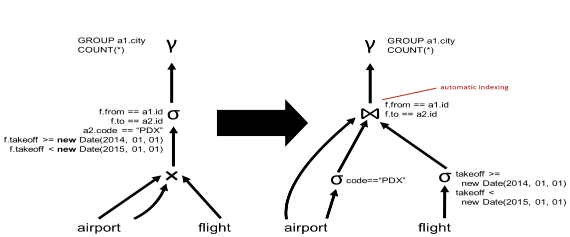
Language **abstractions** for OO reactive programming



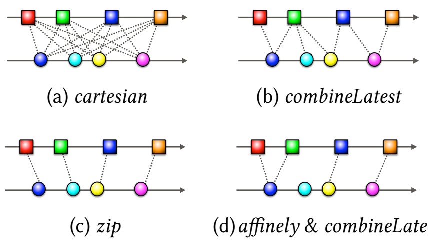
Controlled **experiments**



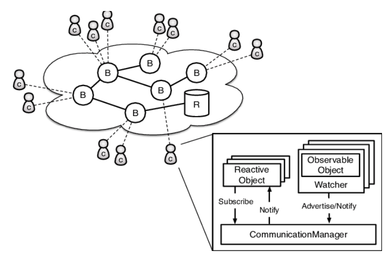
Tools supporting the development process



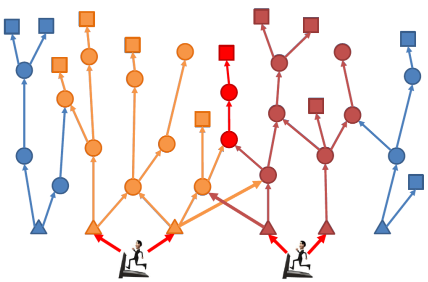
Incremental changes



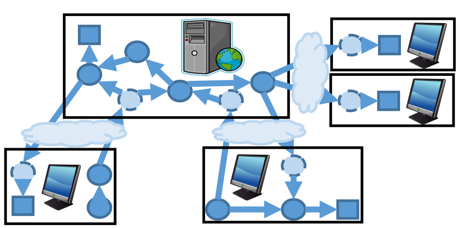
Semantics of Event Correlation



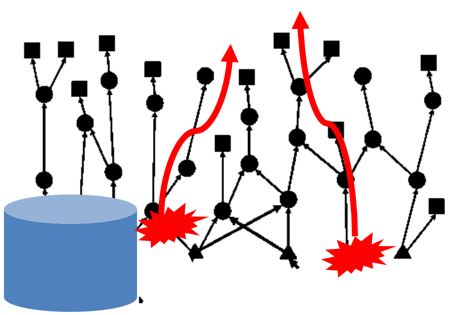
Configurable Consistency



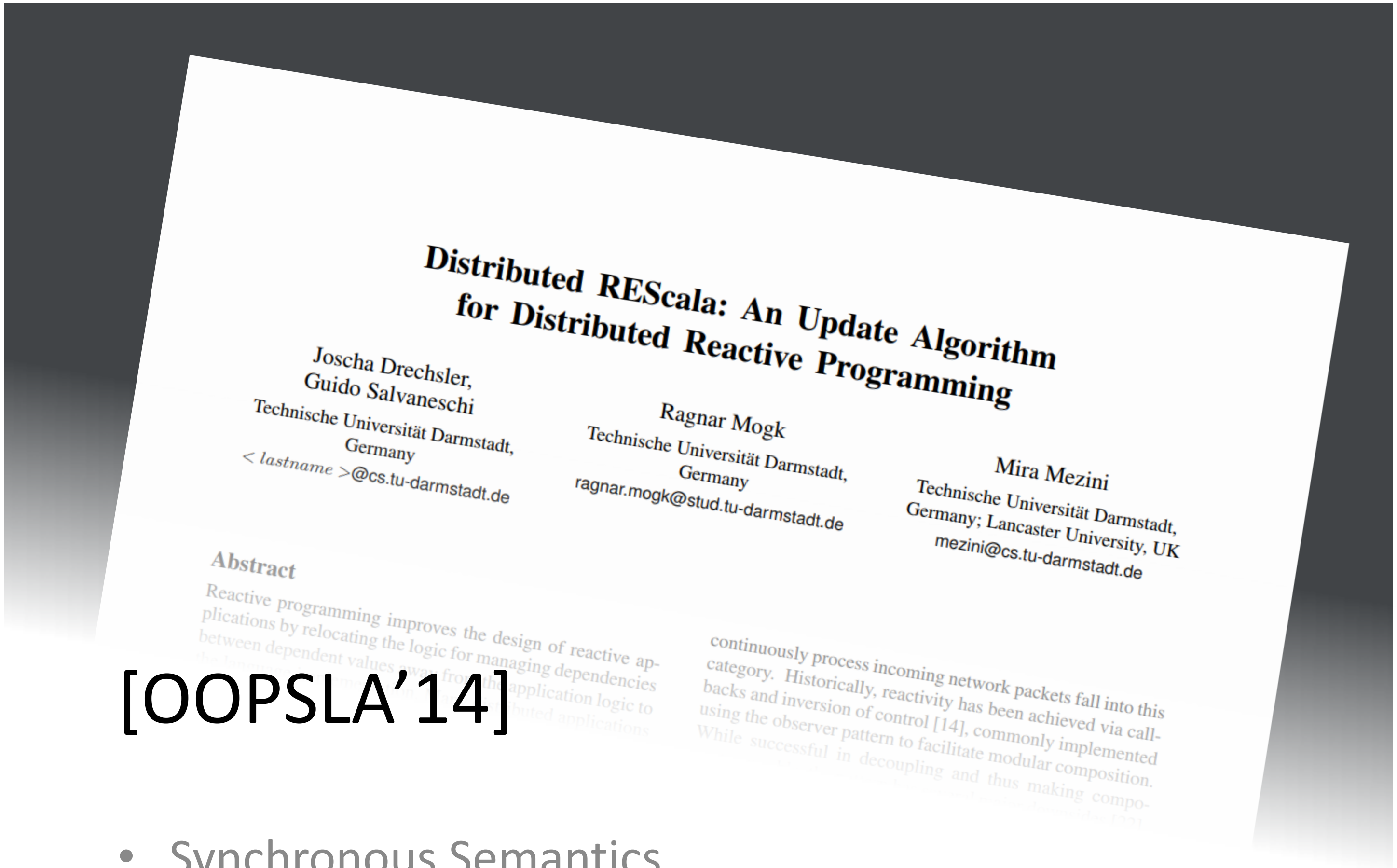
Concurrency



Distribution



Fault Tolerance



Distributed REScala: An Update Algorithm for Distributed Reactive Programming

Joscha Drechsler,
Guido Salvaneschi
Technische Universität Darmstadt,
Germany
< *lastname* >@cs.tu-darmstadt.de

Ragnar Mogk
Technische Universität Darmstadt,
Germany
ragnar.mogk@stud.tu-darmstadt.de

Mira Mezini
Technische Universität Darmstadt,
Germany; Lancaster University, UK
mezini@cs.tu-darmstadt.de

Abstract

Reactive programming improves the design of reactive applications by relocating the logic for managing dependencies between dependent values away from the application logic to

continuously process incoming network packets fall into this category. Historically, reactivity has been achieved via callbacks and inversion of control [14], commonly implemented using the observer pattern to facilitate modular composition. While successful in decoupling and thus making compo-

[OOPSLA'14]

- Synchronous Semantics
- Decentralized
- Dynamic Edges


Artifacts Available
Artifacts Evaluated Reusable
Artifacts Evaluated Functional

Thread-Safe Reactive Programming

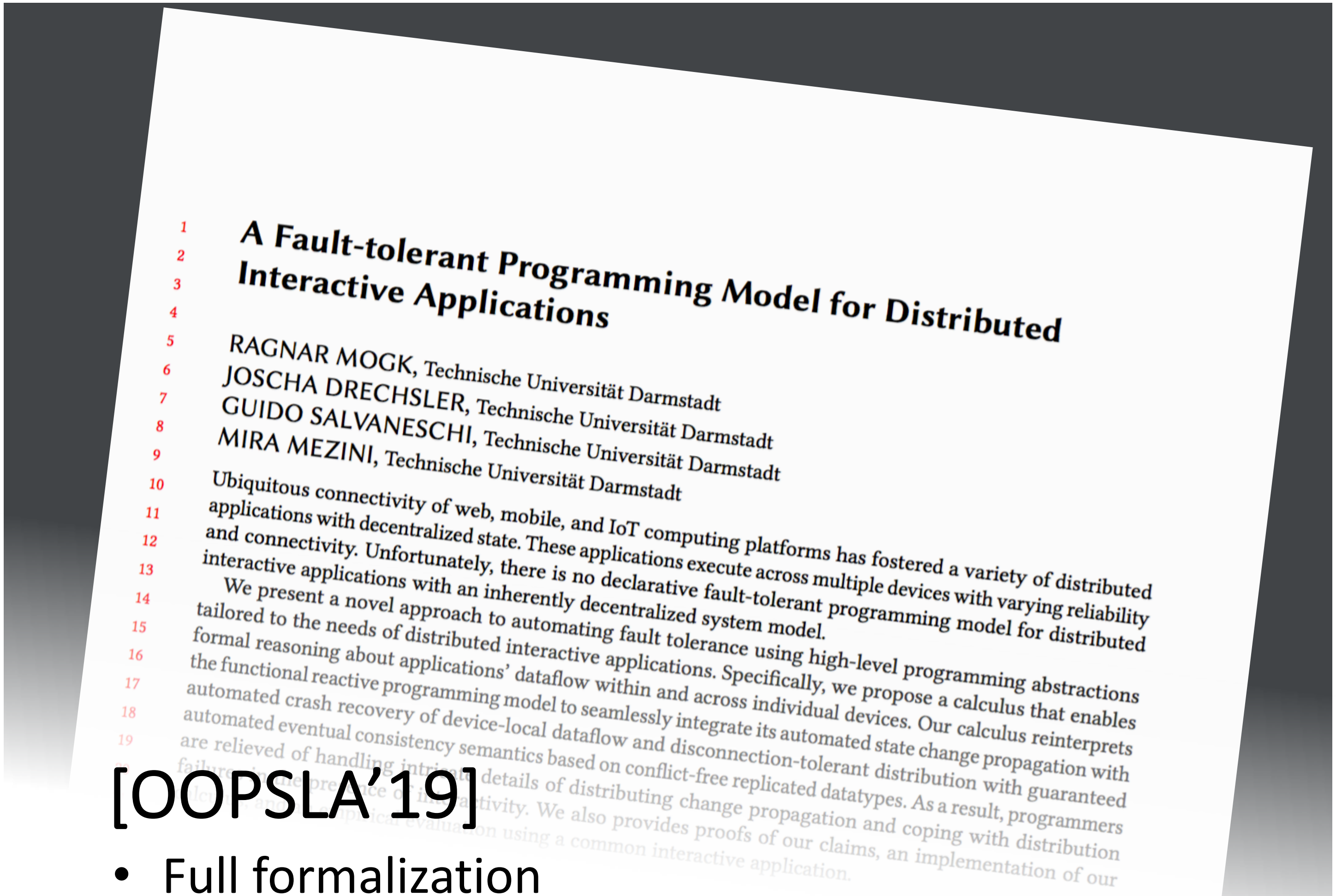
JOSCHA DRECHSLER, Technische Universität Darmstadt, Germany
RAGNAR MOGK, Technische Universität Darmstadt, Germany
GUIDO SALVANESCHI, Technische Universität Darmstadt, Germany
MIRA MEZINI, Technische Universität Darmstadt, Germany

The execution of an application written in a reactive language involves transfer of data and control flow between imperative and reactive abstractions at well-defined points. In a multi-threaded environment, multiple such interactions may execute concurrently, potentially causing data races and event ordering ambiguities. Existing RP languages either disable multi-threading or handle it at the cost of reducing expressiveness or weakening consistency. This paper proposes a model for thread-safe reactive programming (RP) that ensures abort-free strict serializability under concurrency while sacrificing neither expressiveness nor consistency. We also propose an architecture for integrating a corresponding scheduler into the RP language runtime, such

107

[OOPSLA'18]

- Synchronous Semantics
- Fine-Grained Parallelism
- Dynamic Edges



A Fault-tolerant Programming Model for Distributed Interactive Applications

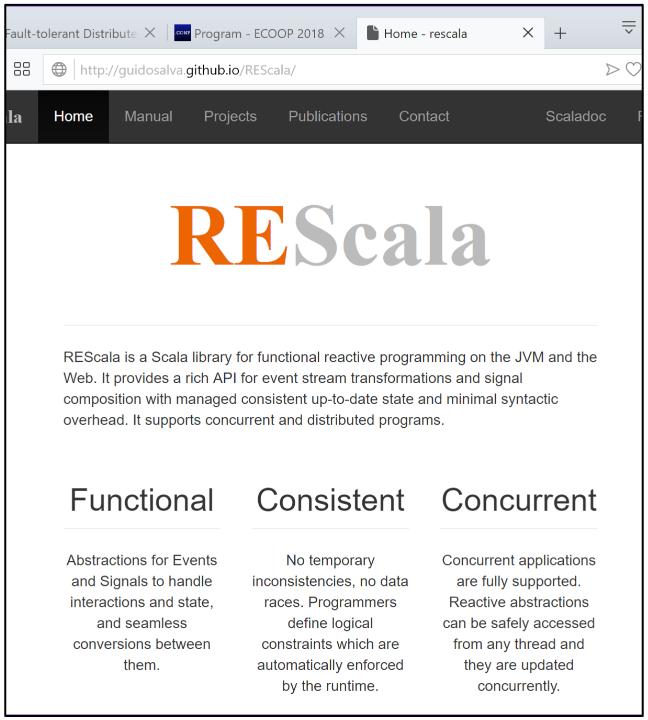
RAGNAR MOGK, Technische Universität Darmstadt
JOSCHA DRECHSLER, Technische Universität Darmstadt
GUIDO SALVANESCHI, Technische Universität Darmstadt
MIRA MEZINI, Technische Universität Darmstadt

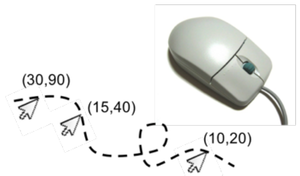
Ubiquitous connectivity of web, mobile, and IoT computing platforms has fostered a variety of distributed applications with decentralized state. These applications execute across multiple devices with varying reliability and connectivity. Unfortunately, there is no declarative fault-tolerant programming model for distributed interactive applications with an inherently decentralized system model.

We present a novel approach to automating fault tolerance using high-level programming abstractions tailored to the needs of distributed interactive applications. Specifically, we propose a calculus that enables formal reasoning about applications' dataflow within and across individual devices. Our calculus reinterprets the functional reactive programming model to seamlessly integrate its automated state change propagation with automated crash recovery of device-local dataflow and disconnection-tolerant distribution with guaranteed automated eventual consistency semantics based on conflict-free replicated datatypes. As a result, programmers are relieved of handling intricate details of distributing change propagation and coping with distribution failures in the presence of interactivity. We also provides proofs of our claims, an implementation of our calculus, and an empirical evaluation using a common interactive application.

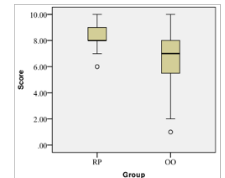
[OOPSLA'19]

- Full formalization
- CRDTs between graphs
- Recovery after disconnection

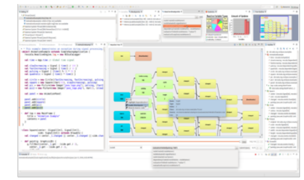




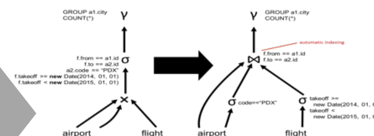
Abstractions for OO reactive programming



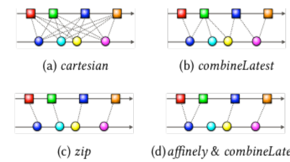
Controlled **experiments**



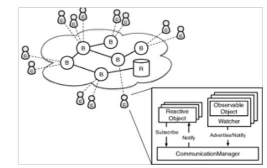
Tools supporting the development process



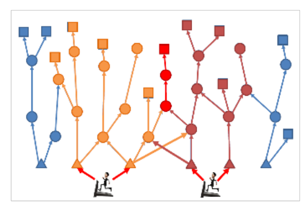
Incremental changes



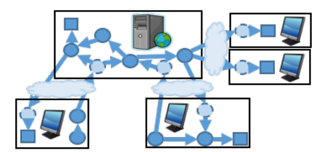
Semantics of Event Correlation



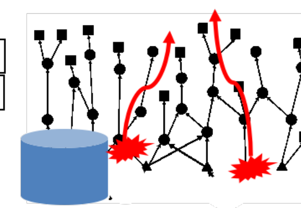
Configurable Consistency



Concurrency



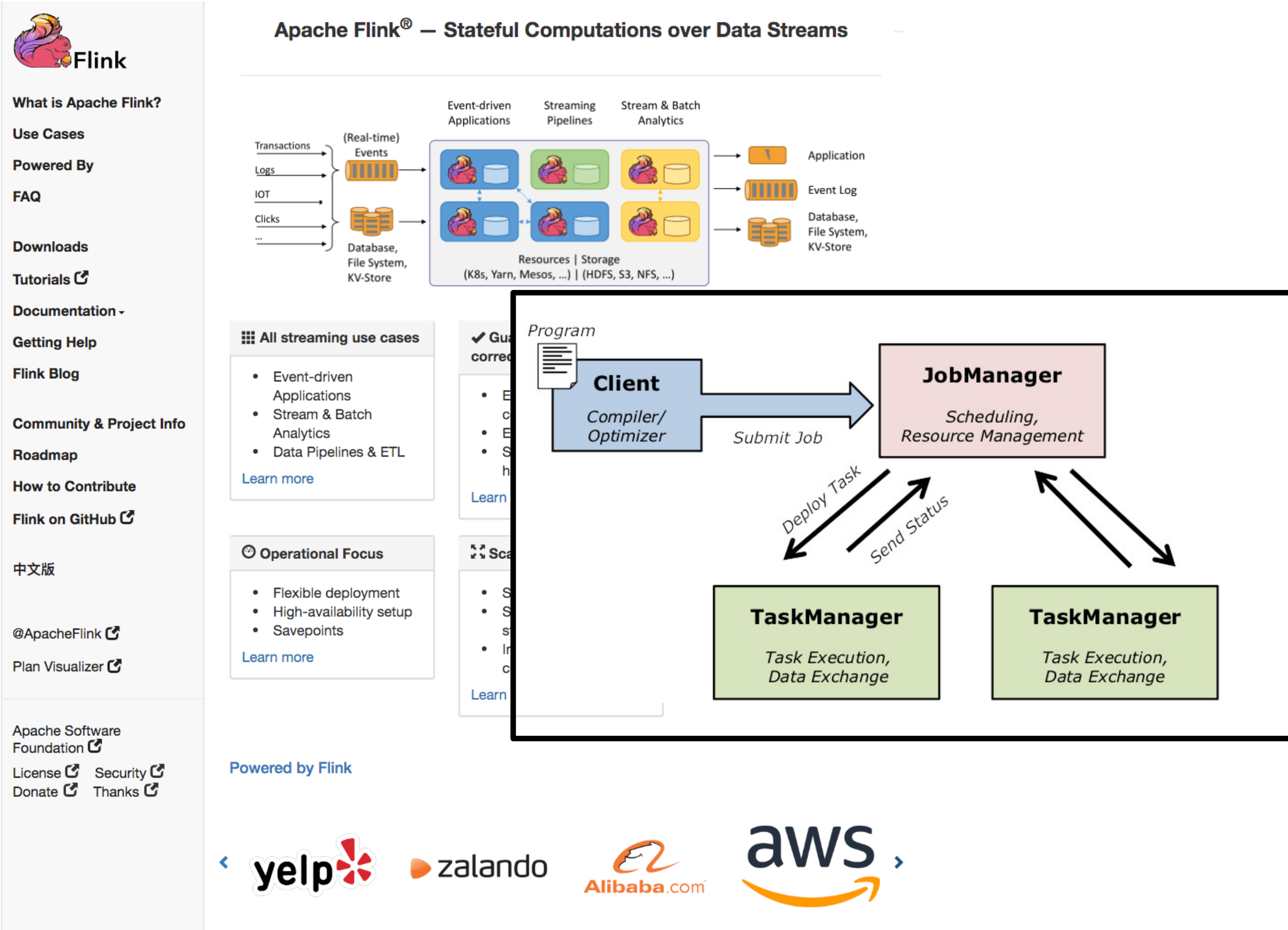
Distribution



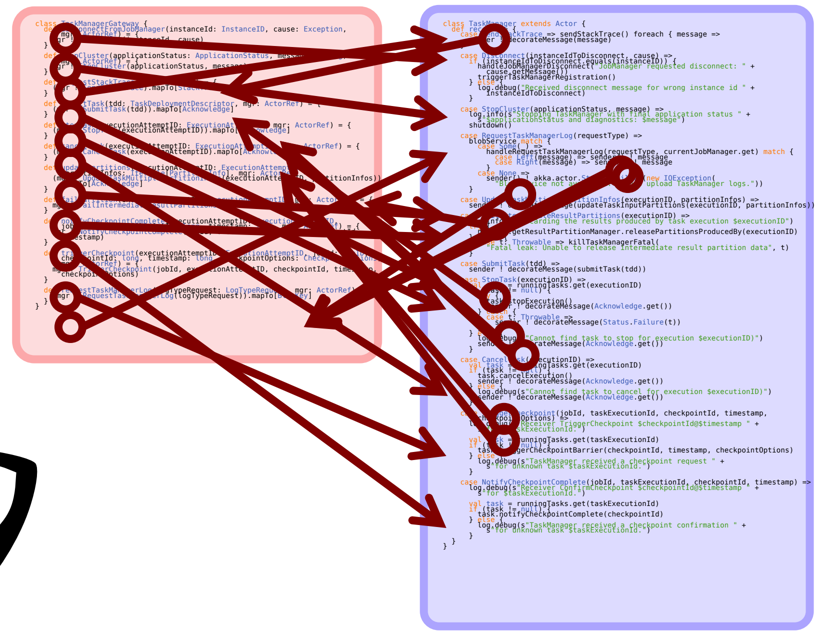
Fault Tolerance

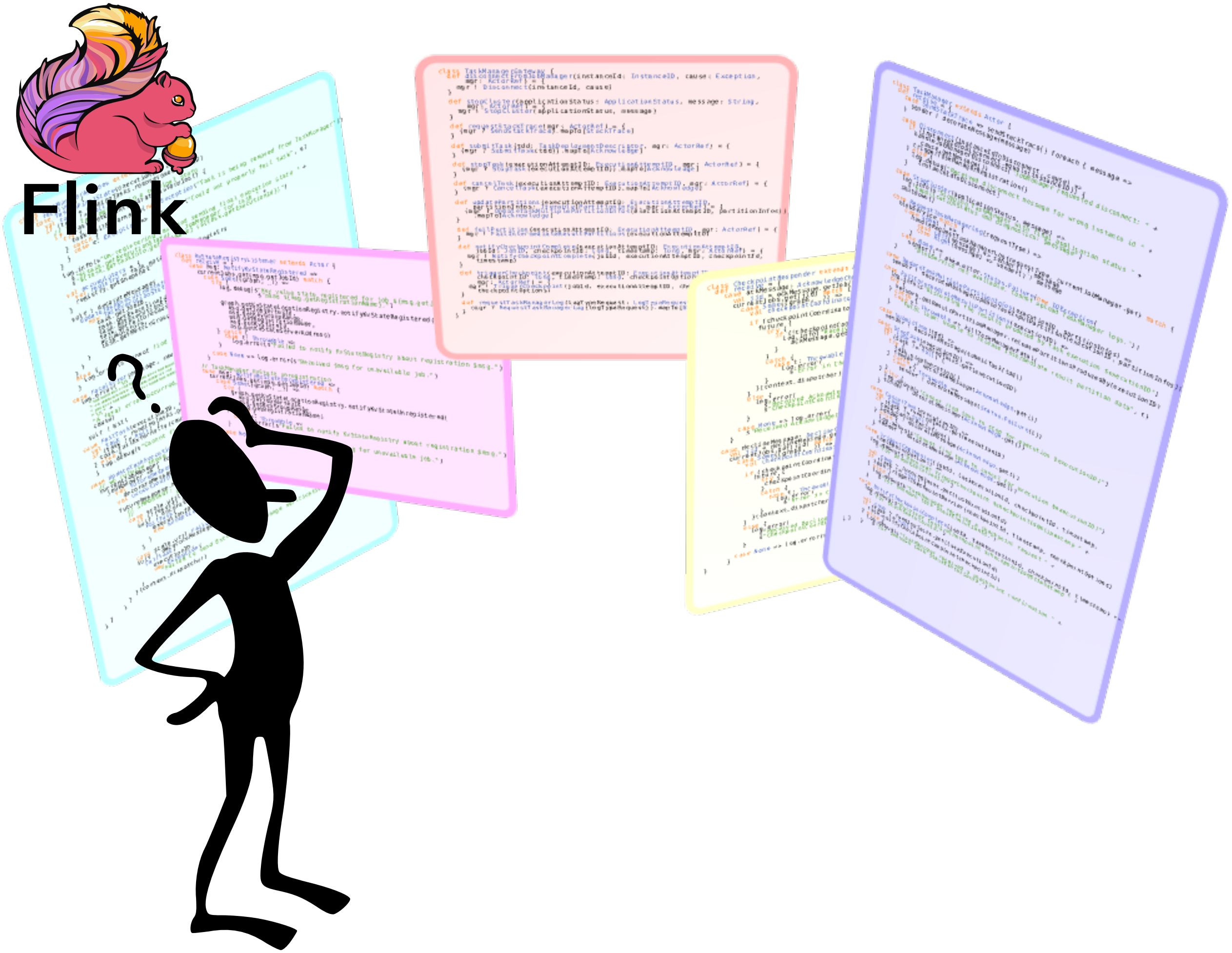
www.rescala-lang.com

SOFTWARE DESIGN

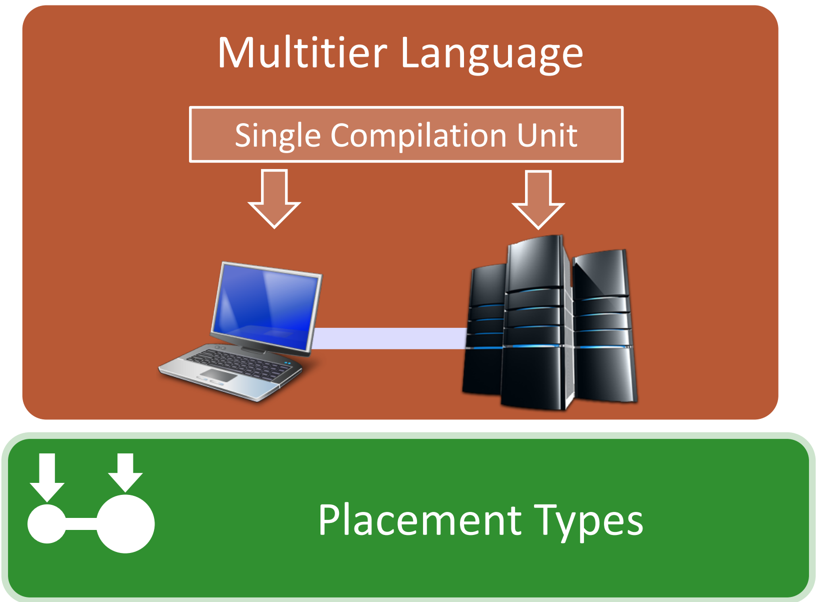

Flink
Apache Flink® — Stateful Computations over Data Streams
What is Apache Flink?
Use Cases
Powered By
FAQ
Downloads
Tutorials
Documentation
Getting Help
Flink Blog
Community & Project Info
Roadmap
How to Contribute
Flink on GitHub
中文版
@ApacheFlink
Plan Visualizer
Apache Software Foundation
License
Security
Donate
Thanks
Event-driven Applications
Streaming Pipelines
Stream & Batch Analytics
Transactions
Logs
IOT
Clicks
...
(Real-time) Events
Database, File System, KV-Store
Resources | Storage
(K8s, Yarn, Mesos, ...) | (HDFS, S3, NFS, ...)
Application
Event Log
Database, File System, KV-Store
All streaming use cases
Event-driven Applications
Stream & Batch Analytics
Data Pipelines & ETL
Learn more
Operational Focus
Flexible deployment
High-availability setup
Savepoints
Learn more
Learn
Learn
Powered by Flink
yelp
zalando
Alibaba.com
aws
Program
Client
Compiler/ Optimizer
Submit Job
JobManager
Scheduling, Resource Management
Deploy Task
Send Status
TaskManager
Task Execution, Data Exchange
TaskManager
Task Execution, Data Exchange

Flink

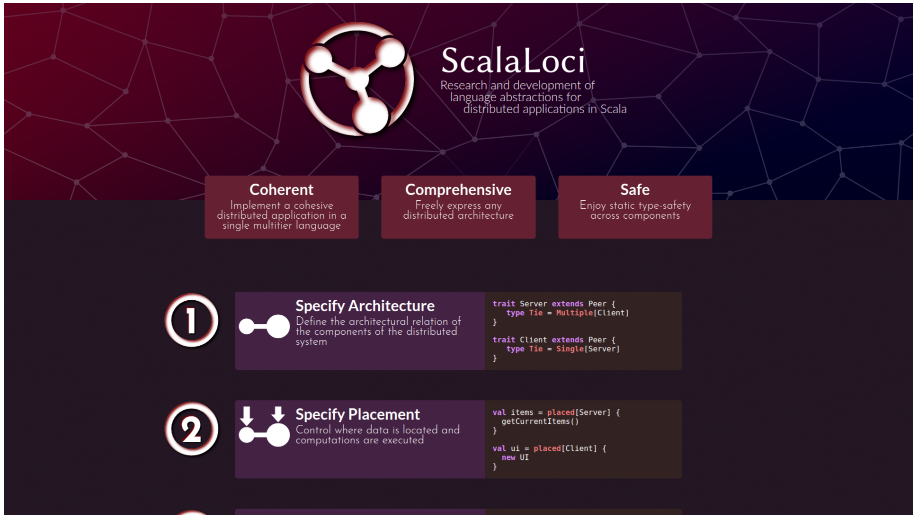



Flink

ScalaLoci Programming Framework



www.scala-loci.github.io



[P.Weisenbureger, M.Koehler, G.Salvaneschi, **Distributed System Development with ScalaLoci,** OOPSLA'18]

[P.Weisenbureger, G.Salvaneschi, **Multitier Modules,** ECOOP'19]

Placement Types

```
trait Registry extends Peer
trait Node extends Peer

val message: Event[String] on Registry
  = placed { getMessageStream() }
```

Peers

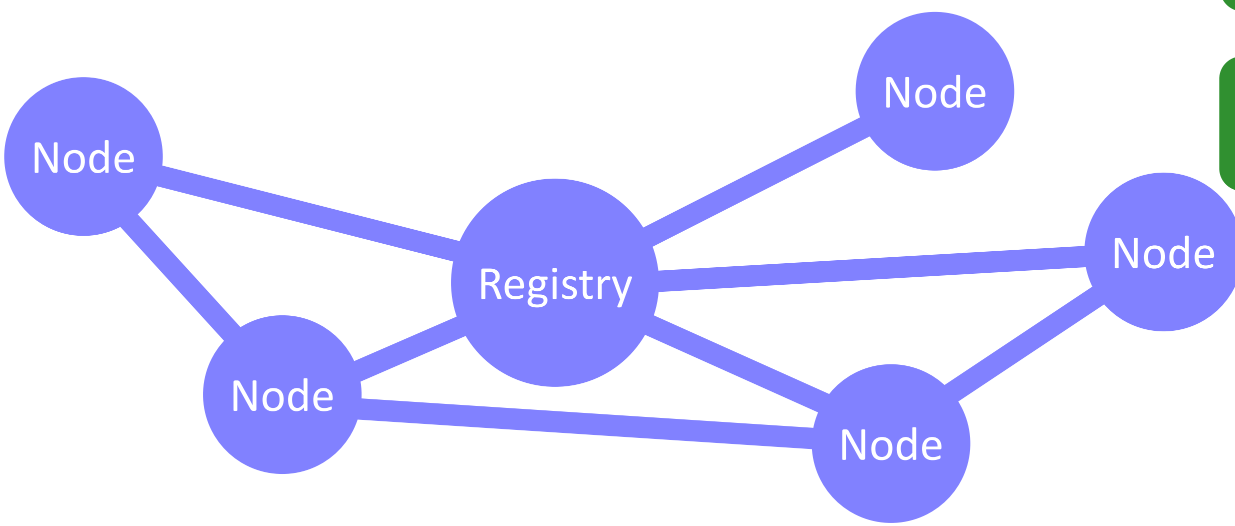
Placement Types

Architecture

```
trait Registry extends Peer { type Tie = Multiple[Node] }
trait Node extends Peer { type Tie = Single[Registry] with Multiple[Node]
```

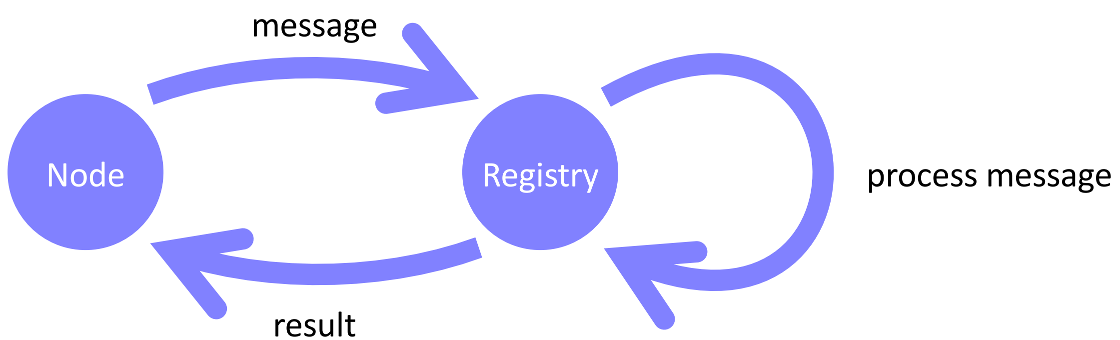
Architecture Specification through Peer Types

Architecture-Based Remote Access



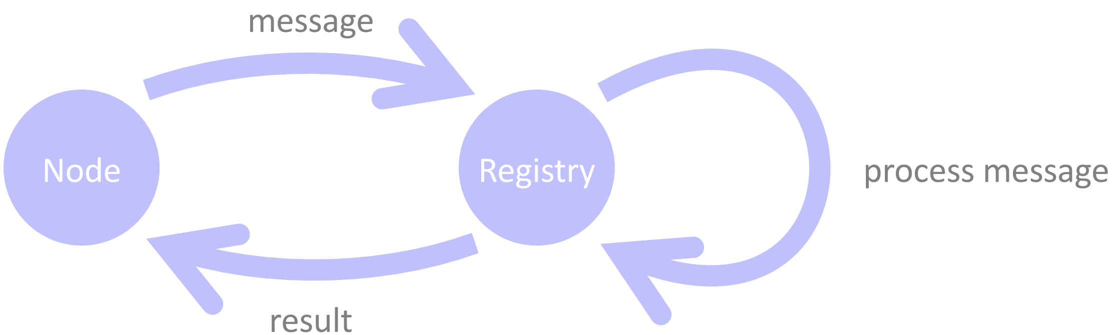
Data Flow

```
val message = Event[String]()
val result = message map processMessage
val ui = new UI(result)
```



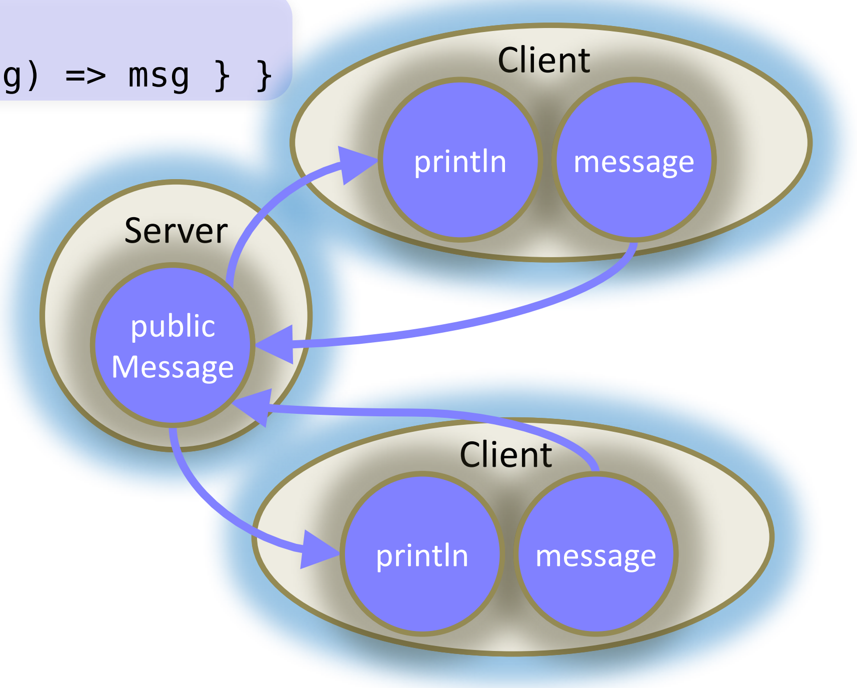
Distributed Data Flow

```
val message: Event[String] on Node = placed[Node] { Event[String]() }
val result = placed[Registry] { message.asLocal map processMessage }
val ui = placed[Node] { new UI(result.asLocal) }
```

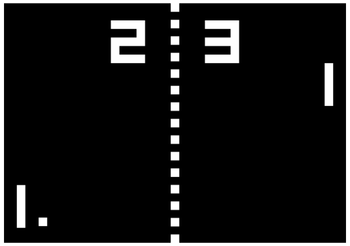


Complete Distributed Chat

```
@multitier object Chat {
  trait Server extends Peer { type Tie = Multiple[Client] }
  trait Client extends Peer { type Tie = Single[Server] }

  val message = placed[Client] { Evt[String] }

  val publicMessage = placed[Server] {
    message.asLocalFromAllSeq map { case (_, msg) => msg } }

  placed[Client].main {
    publicMessage.asLocal observe println
    for (line <- io.Source.stdin.getLines)
      message fire line } }
```



Porting to Distribution



Local

```scala
val ballSize = 20
val maxX = 800
val maxY = 400
val leftPos = 30
val rightPos = 770
val initPosition = Point(400, 200)
val initSpeed = Point(10, 8)

val ball: Signal[Point] = tick.fold(initPosition) {
  (ball, _) => ball + speed.get }

val areas = {
  val racketY = Seq(
    Signal { UI.mousePosition().y },
    Signal { ball().y })
  val leftRacket = Racket(leftRacketPos, racketY(0))
  val rightRacket = Racket(rightRacketPos, racketY(1))
  val rackets = List(leftRacket, rightRacket)
  Signal { rackets map { _.area() } } }

val leftWall = ball.changed && { _.x < 0 }
val rightWall = ball.changed && { _.x > maxX }

val xBounce = {
  val ballInRacket = Signal { areas() exists { _ contains ball() } }
  val collisionRacket = ballInRacket changedTo true
  leftWall || rightWall || collisionRacket }
val yBounce = ball.changed &&
  { ball => ball.y < 0 || ball.y > maxY }

val speed = {
  val x = xBounce toggle (initSpeed.x, -initSpeed.x)
  val y = yBounce toggle (initSpeed.y, -initSpeed.y)
  Signal { Point(x(), y()) } }

val score = {
  val leftPoints = rightWall.iterate(0) { _ + 1 }
  val rightPoints = leftWall.iterate(0) { _ + 1 }
  Signal { leftPoints() + " : " + rightPoints() } }

val ui = new UI(areas, ball, score)
```

ScalaLoci

```scala
trait Server extends ServerPeer[Client]
trait Client extends ClientPeer[Server]

val ballSize = 20
val maxX = 800
val maxY = 400
val leftPos = 30
val rightPos = 770
val initPosition = Point(400, 200)
val initSpeed = Point(10, 8)

val clientMouseY = placed[Client] {
  Signal { UI.mousePosition().y } }

val isPlaying = placed[Server].local {
  Signal { remote[Client].connected().size > 2 } }

val ball: Signal[Point] on Server = placed {
  tick.fold(initPosition) { (ball, _) =>
    if (isPlaying.get) ball + speed.get else pos } }

val players = placed[Server].local { Signal {
  remote[Client].connected() match {
    case left :: right :: _ => Seq(Some(left), Some(right))
    case _ => Seq(None, None) } } }

val areas = placed[Server] {
  val racketY = Signal { players() map { _ map {
    client => (clientMouseY from client).asLocal() } getOrElse
    initPosition.y } }
  val leftRacket = Racket(leftPos, Signal { racketY()(0) })
  val rightRacket = Racket(rightPos, Signal { racketY()(1) })
  val rackets = List(leftRacket, rightRacket)
  Signal { rackets map { _.area() } } }

val leftWall = placed[Server].local { ball.changed && { _.x < 0 } }
val rightWall = placed[Server].local { ball.changed && { _.x > maxX } }

val xBounce = placed[Server].local {
  val ballInRacket = Signal { areas() exists { _ contains ball() } }
  val collisionRacket = ballInRacket changedTo true
  leftWall || rightWall || collisionRacket }
val yBounce = placed[Server].local { ball.changed &&
  { ball => ball.y < 0 || ball.y > maxY } }

val speed = placed[Server].local {
  val x = xBounce toggle (initSpeed.x, - initSpeed.x)
  val y = yBounce toggle (initSpeed.y, - initSpeed.y)
  Signal { Point(x(), y()) } }

val score = placed[Server] {
  val leftPoints = rightWall.iterate(0) { _ + 1 }
  val rightPoints = leftWall.iterate(0) { _ + 1 }
  Signal { leftPoints() + " : " + rightPoints() } }

val ui = placed[Client] {
  new UI(areas.asLocal, ball.asLocal, score.asLocal) }
```

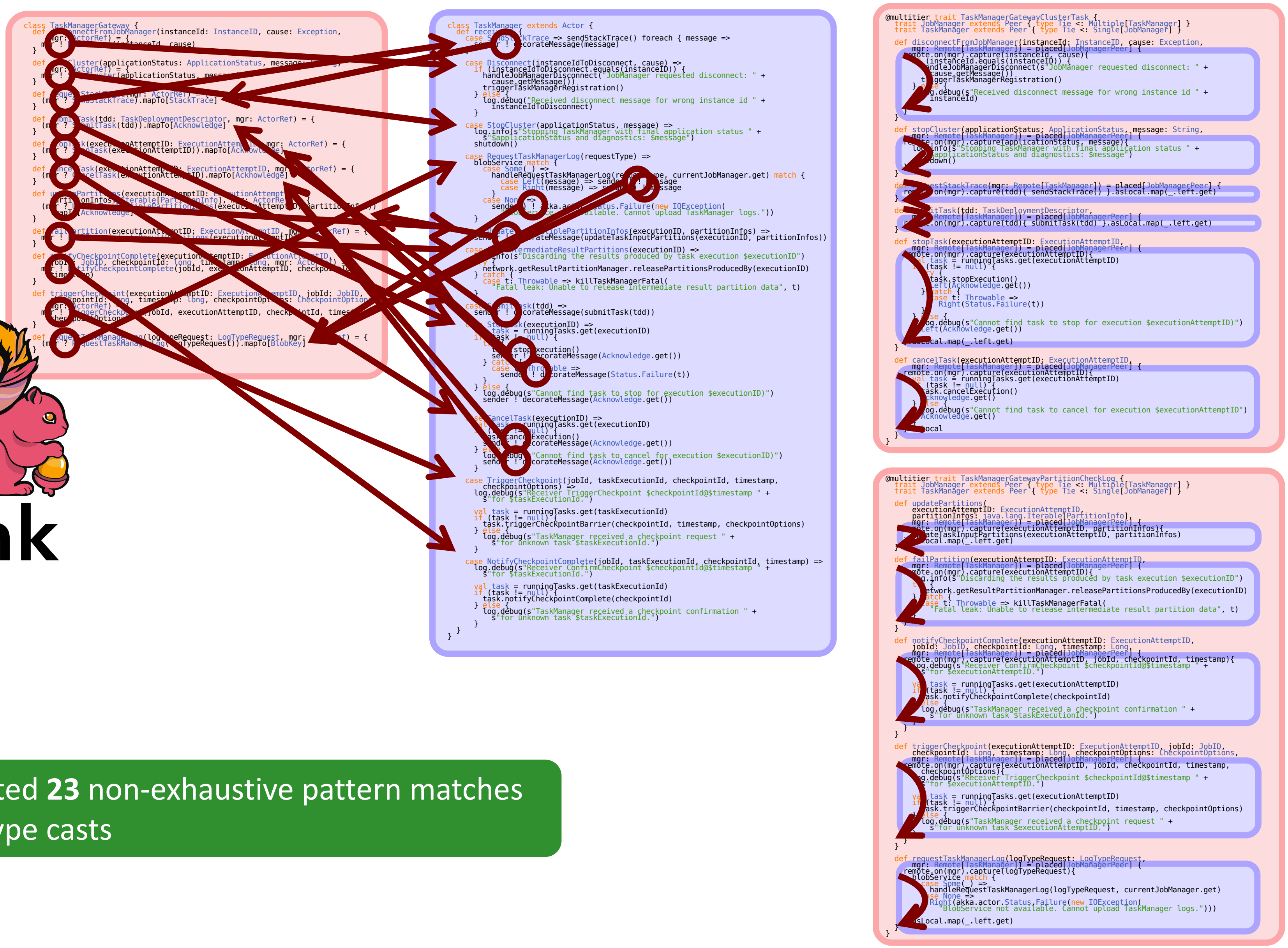
Akka

```scala
val ballSize = 20
val maxX = 800
val maxY = 400
val leftPos = 30
val rightPos = 770
val initPosition = Point(400, 200)
val initSpeed = Point(10, 8)

class Server extends Actor {
  def receive = addPlayer orElse mouseYChanged

  val clients = Var(Seq.empty[ActorRef])

  val mousePositions = Var(Map.empty[ActorRef, Int])

  def mouseYChanged: Receive = { case MouseYChanged(y) =>
    mousePositions transform { _ + (sender -> y) } }

  val isPlaying = Signal { clients().size >= 2 }

  val ball: Signal[Point] =
    tick.fold(initPosition) { (ball, _) =>
      if (isPlaying.get) ball + speed.get else ball }

  def addPlayer: Receive = { case AddPlayer =>
    clients transform { _ :+ sender } }

  val players = Signal {
    clients() match {
      case left :: right ::   => Seq(Some(left), Some(right))
      case _ => Seq(None, None) } }

  val areas = {
    val racketY = Signal {
      players() map {
        _ flatMap { mousePositions() get _ } getOrElse initPosition.y } }
    val leftRacket = new Racket(leftRacketPos, Signal { racketY()(0) })
    val rightRacket = new Racket(rightRacketPos, Signal { racketY()(1) })
    val rackets = List(leftRacket, rightRacket)
    Signal { rackets map { _.area() } } }

  val leftWall = ball.changed && { _.x < 0 }
  val rightWall = ball.changed && { _.x > maxX }

  val xBounce = {
    val ballInRacket = Signal { areas() exists { _ contains ball() } }
    val collisionRacket = ballInRacket changedTo true
    leftWall || rightWall || collisionRacket }
  val yBounce = ball.changed &&
    { ball => ball.y < 0 || ball.y > maxY }

  val speed = {
    val x = xBounce toggle (Signal { initSpeed.x }, Signal { -initSpeed.x })
    val y = yBounce toggle (Signal { initSpeed.y }, Signal { -initSpeed.y })
    Signal { Point(x(), y()) } }

  val score = {
    val leftPlayerPoints = rightWall.iterate(0) { _ + 1 }
    val rightPlayerPoints = leftWall.iterate(0) { _ + 1 }
    Signal { leftPlayerPoints() + " : " + rightPlayerPoints() } }

  areas observe { areas => clients.now foreach {   ! UpdateAreas(areas) } }
  ball observe { ball => clients.now foreach { _ ! UpdateBall(ball) } }
  score observe { score => clients.now foreach { _ ! UpdateScore(score) } }

  clients observe {   foreach { client =>
    client ! UpdateAreas(areas.now)
    client ! UpdateBall(ball.now)
    client ! UpdateScore(score.now) } }
}

abstract class Client(server: ActorSelection) extends Actor {
  val areas = Var(List.empty[Area])~
  val ball = Var(Point(0, 0))~
  val score = Var("0 : 0")~

  mousePosition observe { pos =>
    ~server ! MouseYChanged(pos.y)~ }

  val ui = new UI(areas, ball, score)

  def receive = {~
    case UpdateAreas(areas) => this.areas set areas
    case UpdateBall(ball) => this.ball set ball
    case UpdateScore(score) => this.score set score }

  server ! AddPlayer
}
```

RMI

```scala
val ballSize = 20
val maxX = 800
val maxY = 400
val leftPos = 30
val rightPos = 770
val initPosition = Point(400, 200)
val initSpeed = Point(10, 8)

@remote trait Server {
  def addPlayer(client: Client): Unit
  def mouseYChanged(client: Client, y: Int): Unit }

class ServerImpl extends Server {
  val clients = Var(Seq.empty[Client])

  val mousePositions = Var(Map.empty[Client, Int])

  def mouseYChanged(client: Client, y: Int) = synchronized {
  mousePositions() = mousePositions.get + (client -> y) }

  val isPlaying = Signal { clients().size >= 2 }

  val ball: Signal[Point] =
    tick.fold(initPosition) { (ball, _) =>
      if (isPlaying.get) ball + speed.get else ball }

  def addPlayer(client: Client) = synchronized {
    clients transform { _ :+ client } }

  val players = Signal {
    clients() match {
      case left :: right :: _ => Seq(Some(left), Some(right))
      case _ => Seq(None, None) } }

  val areas = {
    val racketY = Signal {
      players() map {
        _ flatMap { mousePositions() get _ } getOrElse initPosition.y } }
    val leftRacket = new Racket(leftRacketPos, Signal { racketY()(0) })
    val rightRacket = new Racket(rightRacketPos, Signal { racketY()(1) })
    val rackets = List(leftRacket, rightRacket)
    Signal { rackets map { _.area() } } }

  val leftWall = ball.changed && { _.x < 0 }
  val rightWall = ball.changed && { _.x > maxX }

  val xBounce = {
    val ballInRacket = Signal { areas() exists { _ contains ball() } }
    val collisionRacket = ballInRacket changedTo true
    leftWall || rightWall || collisionRacket }
  val yBounce = ball.changed &&
    { ball => ball.y < 0 || ball.y > maxY }

  val speed = {
    val x = xBounce toggle (initSpeed.x, -initSpeed.x)
    val y = yBounce toggle (initSpeed.y, -initSpeed.y)
    Signal { Point(x(), y()) } }

  val score = {
    val leftPoints = rightWall.iterate(0) { _ + 1 }
    val righrPoints = leftWall.iterate(0) { _ + 1 }
    Signal { leftPoints() + " : " + righrPoints() } }

  areas observe { updateAreasClients(clients.get, _) }
  ball observe { updateBallClients(clients.get, _) }
  score observe { updateScoreClients(clients.get, _) }

  clients observe { clients =>
    updateAreasClients(clients, areas.get)
    updateBallClients(clients, ball.get)
    updateScoreClients(clients, score.get) }

  def updateAreasClients(clients: Seq[Client], areas: List[Area]) =
    clients foreach { _ updateAreas areas }
  def updateBallClients(clients: Seq[Client], ball: Point) =
    clients foreach { _ updateBall ball }
  def updateScoreClients(clients: Seq[Client], score: String) =
    clients foreach { _ updateScore score }
}

@remote trait Client {
  def updateAreas(areas: List[Area]): Unit
  def updateBall(ball: Point): Unit
  def updateScore(score: String): Unit }

class ClientImpl(server: Server) extends Client {
  val self = makeStub[Client](this)

  val areas = Var(List.empty[Area])
  val ball = Var(Point(0, 0))
  val score = Var("0 : 0")

  UI.mousePosition observe { pos =>
    server mouseYChanged (self, pos.y) }

  val ui = new UI(areas, ball, score)

  def updateAreas(areas: List[Area]) = synchronized { this.areas() = areas }
  def updateBall(ball: Point) = synchronized { this.ball() = ball }
  def updateScore(score: String) = synchronized { this.score() = score }

  server addPlayer self
}
```

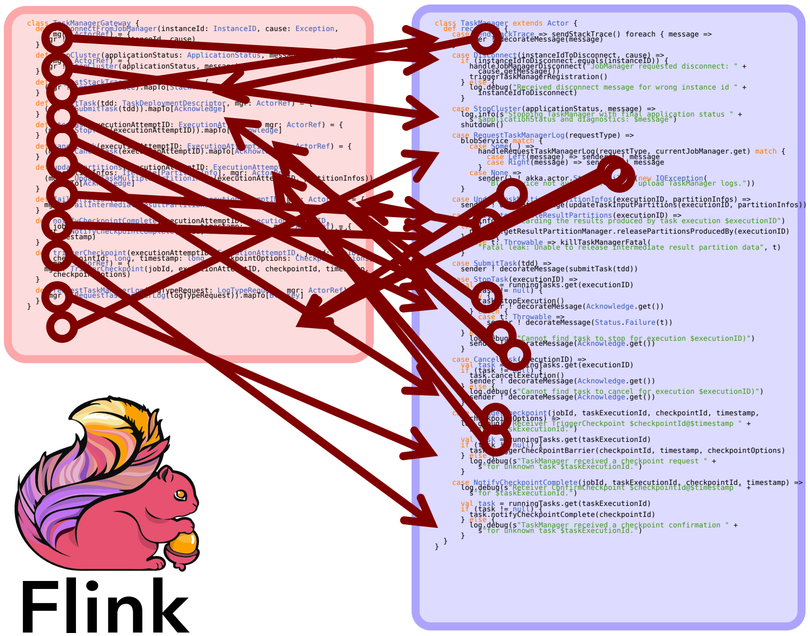
- (light yellow) multi-user support
- (orange) distribution



Flink

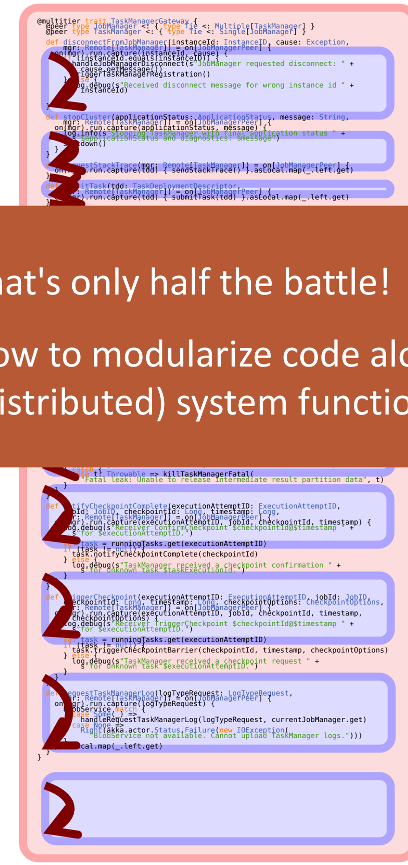
Eliminated **23** non-exhaustive pattern matches and **8** type casts

Crosscutting functionality separated among compilation units



Flink

Developers are **not** forced to modularize **along network boundaries**



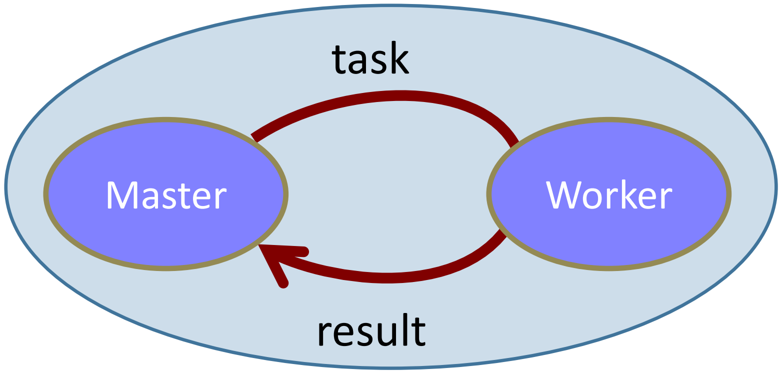
That's only half the battle!

How to modularize code along (distributed) system functionalities?

Multitier Modules

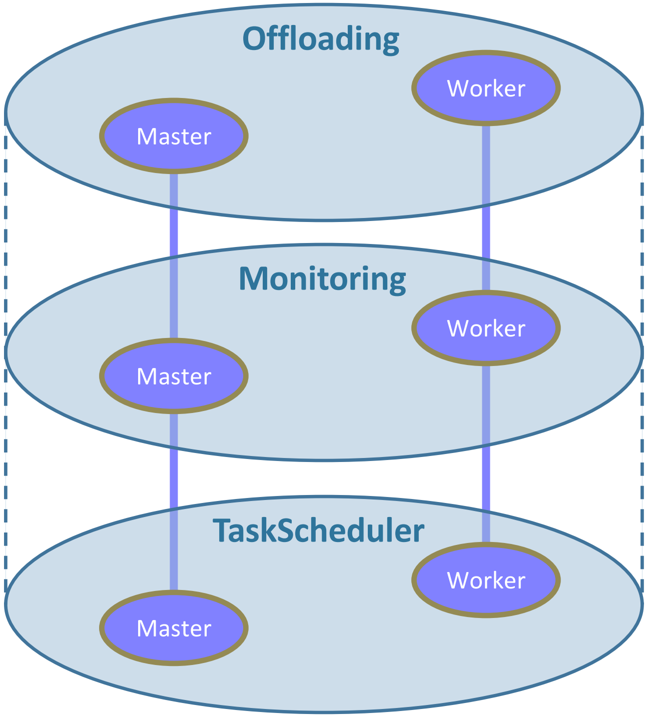
Distributed functionality = Module

Composing modules = Composing subsystems



Handle large code bases

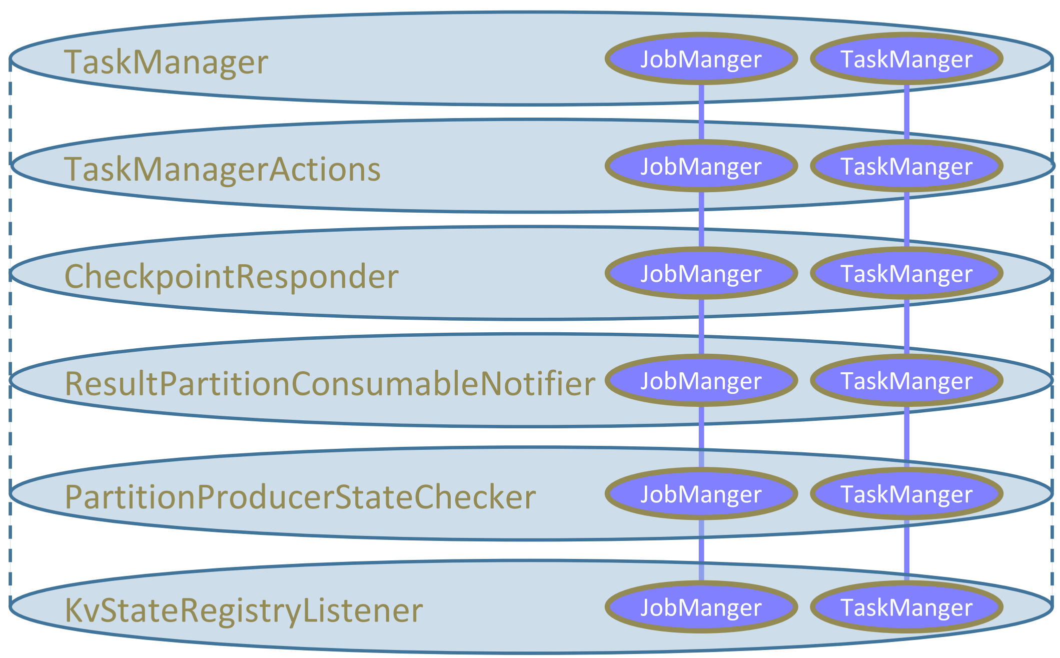
Stacking Multitier Modules




```scala
@multitier trait Offloading[T] {
  @peer type Master <: { type Tie <: Multiple[Worker] }
  @peer type Worker <: { type Tie <: Single[Master] }
  def run(task: Task[T]): Future[T] on Master =
    placed { (remote(selectWorker()) call execute(task)).asLocal }
  private def execute(task: Task[T]): T on Worker =
    placed { task.process() }
}

@multitier trait Monitoring {
  @peer type Master <: { type Tie <: Multiple[Worker] }
  @peer type Worker <: { type Tie <: Single[Master] }
  def monitoredTimedOut(monitored: Remote[Worker]): Unit on Master
}

@multitier trait TaskScheduler[T] extends
  Offloading[T] with
  Monitoring
```

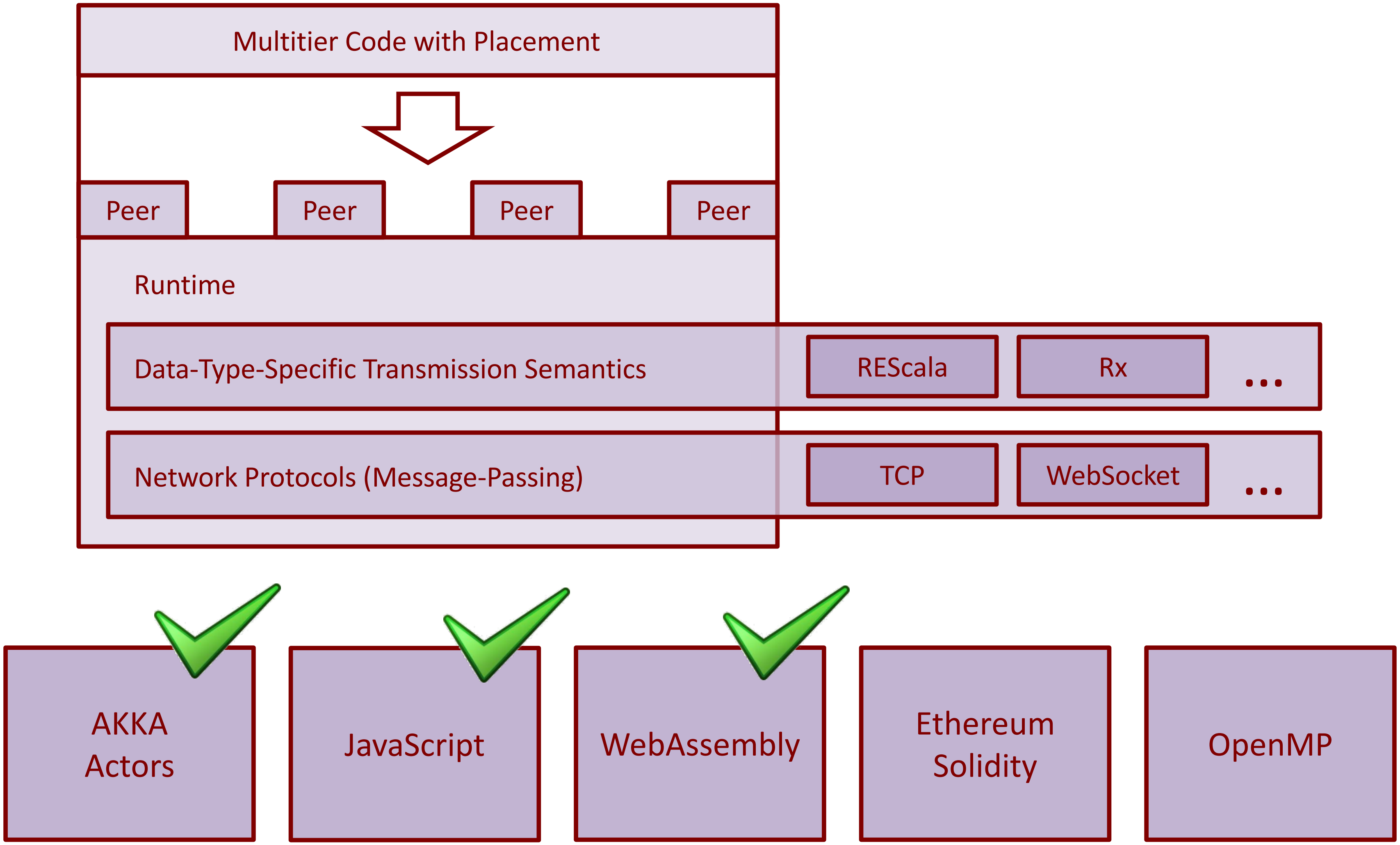
Flink Case Study

```
@multitier object TaskDistributionSystem extends
  TaskManager with
  TaskManagerActions with
  CheckpointResponder with
  ResultPartitionConsumableNotifier with
  PartitionProducerSta
  KvStateRegistryList
```




@multitier trait TaskDistributionSystem extends CheckpointResponder with KvStateRegistryListener with PartitionProducerStateChecker with ResultPartitionConsumableNotifier with TaskManagerGateway with TaskManagerActions
@multitier trait TaskManagerActions {
@multitier trait TaskManagerGateway {
@multitier trait KvStateRegistryListener {
@multitier trait ResultPartitionConsumableNotifier {
@multitier trait CheckpointResponder {
@multitier trait PartitionProducerStateChecker {
Flink

ScalaLoci: Backends



www.scala-loci.github.io

Distributed System Development with ScalaLoci

Fault Tolerance

Dynamic Topologies

Design Metrics

Microbenchmarks

Multiple Backends

Formalization

PASCAL WEISENBURGER, Technische Universität Darmstadt, Germany
MIRKO KÖHLER, Technische Universität Darmstadt, Germany
GUIDO SALVANESCHI, Technische Universität Darmstadt, Germany

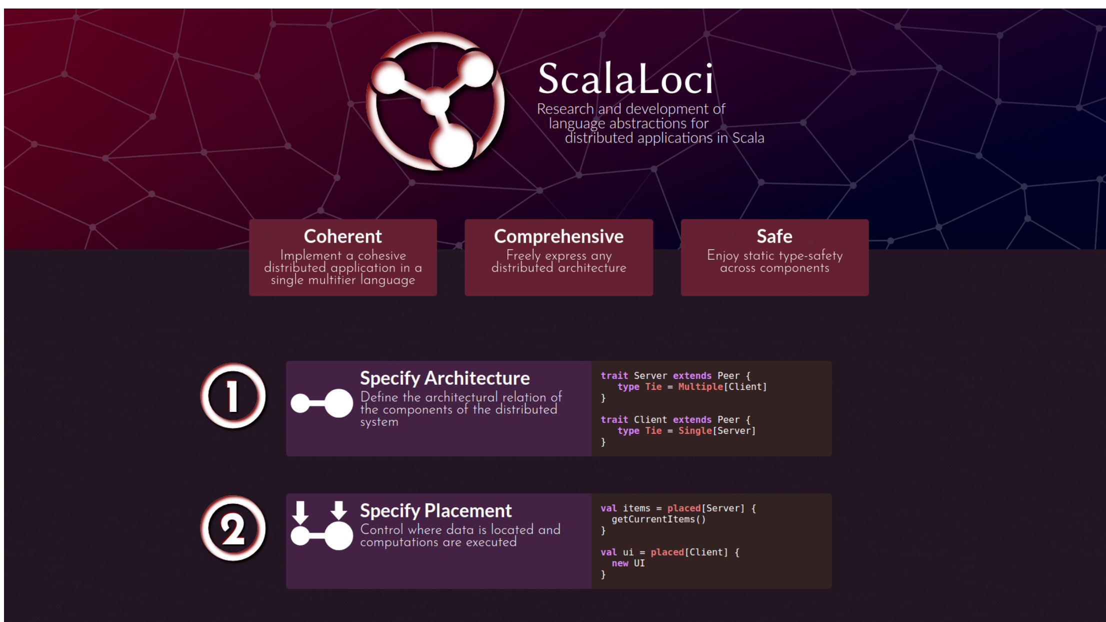
Distributed applications are traditionally developed as separate modules, often in different l... react to events, like user input, and in turn produce new events for the other modules. Separa... nents requires time-consuming integration. Manual implementation of communication forc... to deal with low-level details. The combination of the two results in obscure distributed dat... among multiple modules, hindering reasoning about the system as a whole.

The ScalaLoci distributed programming language addresses these issues with a coherent ... placement types that enables reasoning about distributed data flows, supporting multiple softw... via dedicated language features and abstracting over low-level communication details and ... As we show, ScalaLoci simplifies developing distributed systems, reduces error-prone com... and favors early detection of bugs.

CCS Concepts: • **Software and its engineering** → **Distributed programming languages**... languages; • **Theory of computation** → *Distributed computing models*;

...Programming, Reacti...

...stributed System Development with ...er 2018), 30 pages. https://doi.org/10.



www.scala-loci.github.io

Thank you

QUESTIONS?

@guidosalva
www.guidosalvaneschi.com