PL4DS Meeting

Feb 23rd 2018

Guido Salvaneschi

TU Darmstadt - Germany

PhD - Politecnico di Milano, Italy

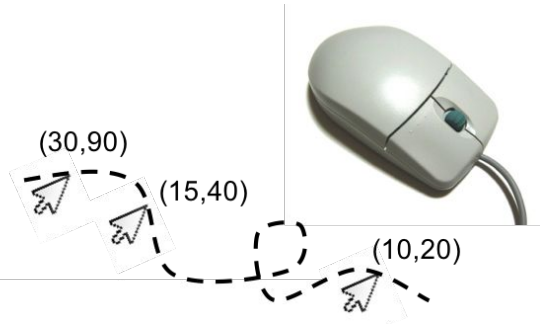
PL for (distributed) adaptive systems

Erlang: Hot swap of code modules

(Distributed) Reactive Programming

REScala programming language

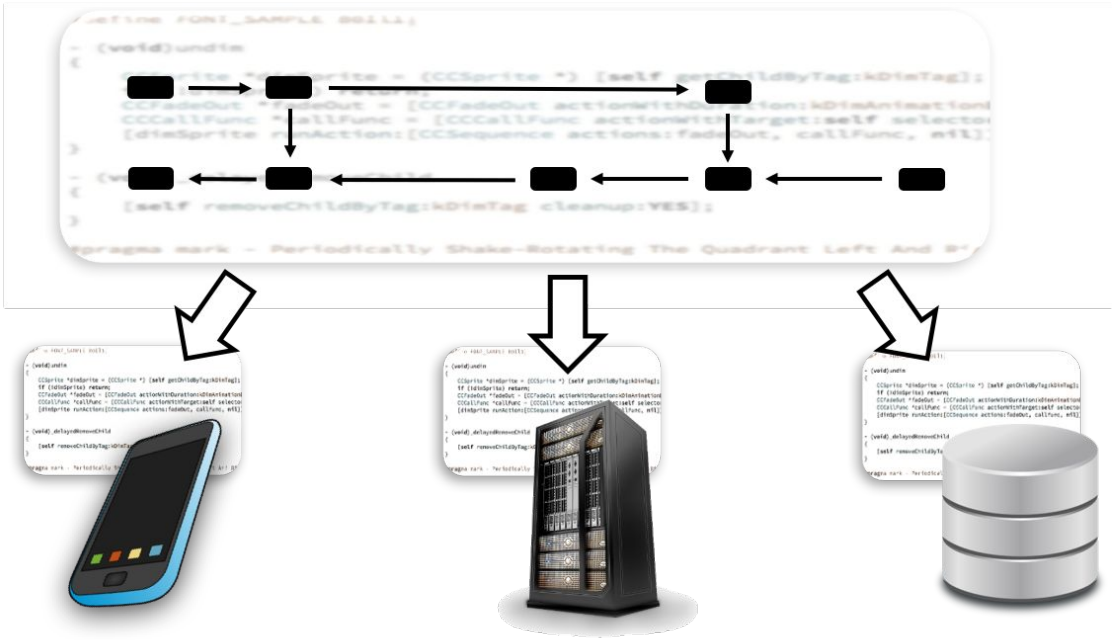
- Distribution
- Concurrency
- Debugging
- Controlled experiments




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

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

www.rescala-lang.com

Tierless Programming: ScalaLoci



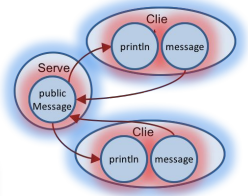
Data Streams cross tier boundaries

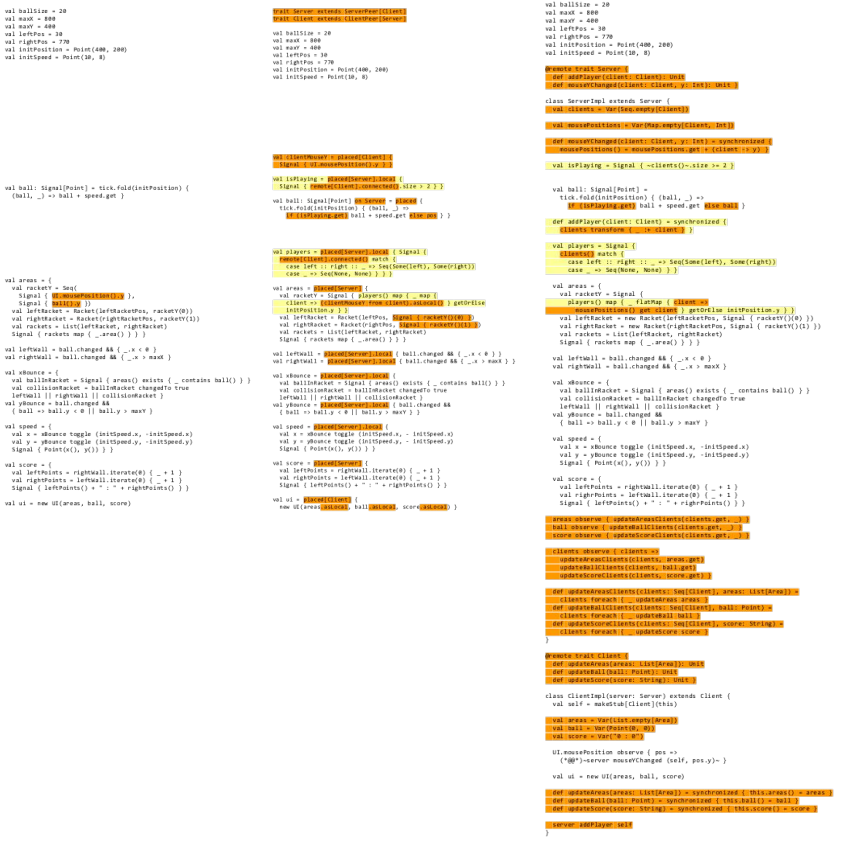
Compiler splits the code

ScalaLoci

Placement types: Server[Int], Client[Bool]

```
import architecture.MultiClientServer._
import serializable.upickle._
import rescalaTransmitter._

@multitier
object Chat {
  class Server extends ServerPeer[Client] {
    def connect = TCP(port = 43053) }
  class Client extends ClientPeer[Server] {
    def connect = load (TCP) ("client.config") }

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

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

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





Carla Ferreira

Universidade NOVA de Lisboa

PhD - University of Southampton, UK

- Formal methods
- Concurrency and distribution
  - Calculi for (failure recovery of) long running transactions
  - Principled methods and verification techniques for cloud applications

Challenges in globally distributed data

- To ensure availability, cloud applications tend to trade strong consistency for weaker models
- These weaker models do not ensure global ADT invariants
  - Solution: Strengthen consistency selectively
- Large cloud providers started to provide multiple consistency choices
- Hard to figure out the **minimum consistency** necessary to **maintain ADT invariants**

CISE consistency model

- A generic model that expresses most existing consistency models
  - Each operation acquires a set of tokens
  - Operations with conflicting tokens cannot run concurrently
- **First proof rule** to check correctness of weakly consistent applications
  - Based on rely-guarantee reasoning
  - Polynomial time analysis
  - Assumes causal consistency
- Static analysis tool (Boogie)

CISE analysis

- Allows developers to build applications assuming initially a sequential setting
  - Then, the analysis detects what are the problematic operations if executed concurrently
- Specification effort is high: input is a Boogie spec of the application
- Cloud apps moved from monoliths to microservices
  - Extend the proof rule to support compositional reasoning

Carlos Baquero

Universidade do Minho - Portugal

HASLab - INESC TEC

(Large Scale) Distributed Systems

Eventual Consistency - CRDTs

Distributed Data Aggregation

Eventual Consistency ADT Design

- Available under Partitions designs (CAP)
  - Preservation of Sequential Semantics
  - Design choices under concurrency
    - Add vs Remove Wins
  - Outcomes not explained by sequential executions
    - Multi-value Registers
  - Causal Consistency

Eventual Consistency Support

- Reliable FIFO Channels
  - Preserve source FIFO
  - Trees provide Causal Consistency
- Reliable Causal Delivery
  - Causal Consistency for arbitrary graphs
    - Delivery pre-conditions vs batch propagation
- FIFO not across TCP connection incarnations

Crossing the CAP Barrier

“Observable” Causal Consistency strongest under AP

Towards global invariants:

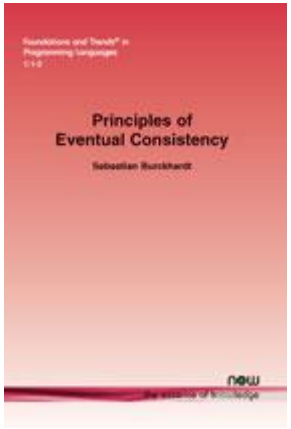
A) Escrow, Tokens, Reliable transfers
B) Consensus Services

Sebastian Burckhardt

Microsoft Research, Redmond

I like to make complicated things look simple.

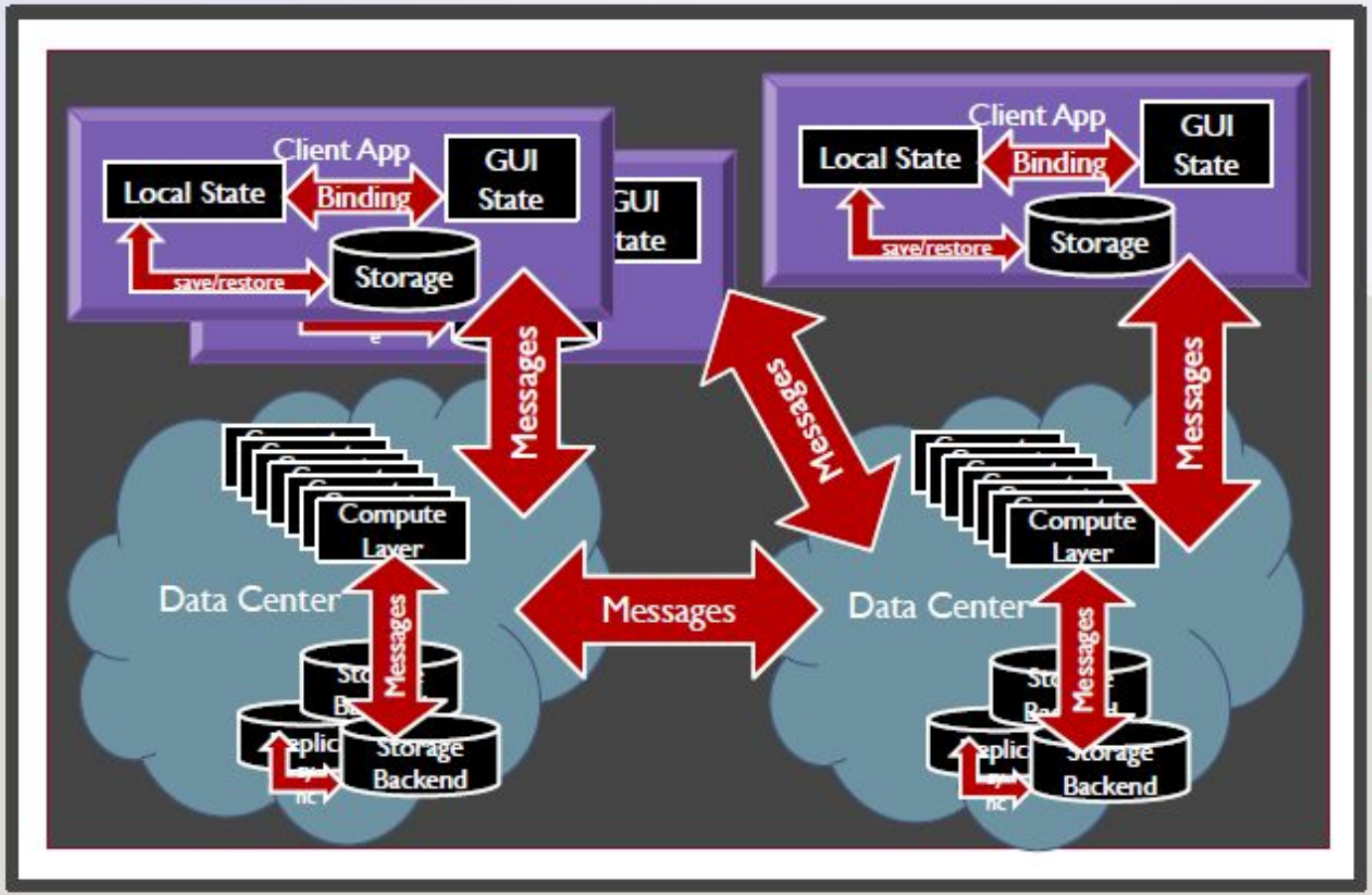
Theoretical Interest: Specification/Verification/Optimality of ~~Concurrent~~Distributed ~~Memory~~~~Objects~~Services



Practical Interest: figuring out the new abstraction stack

(how to build applications in the age of devices+services)

Reactivity, streaming, cloud storage, browsers...

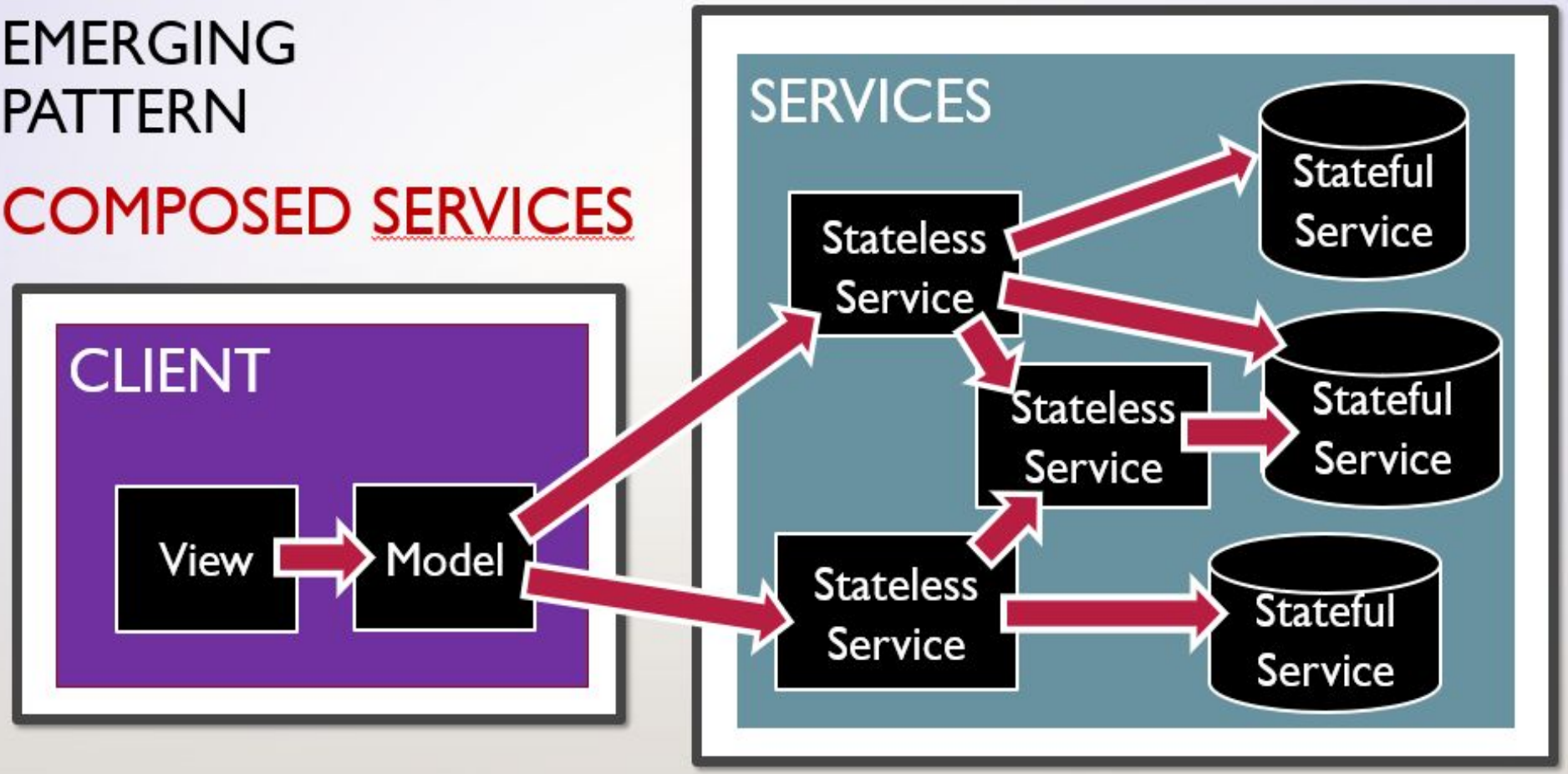


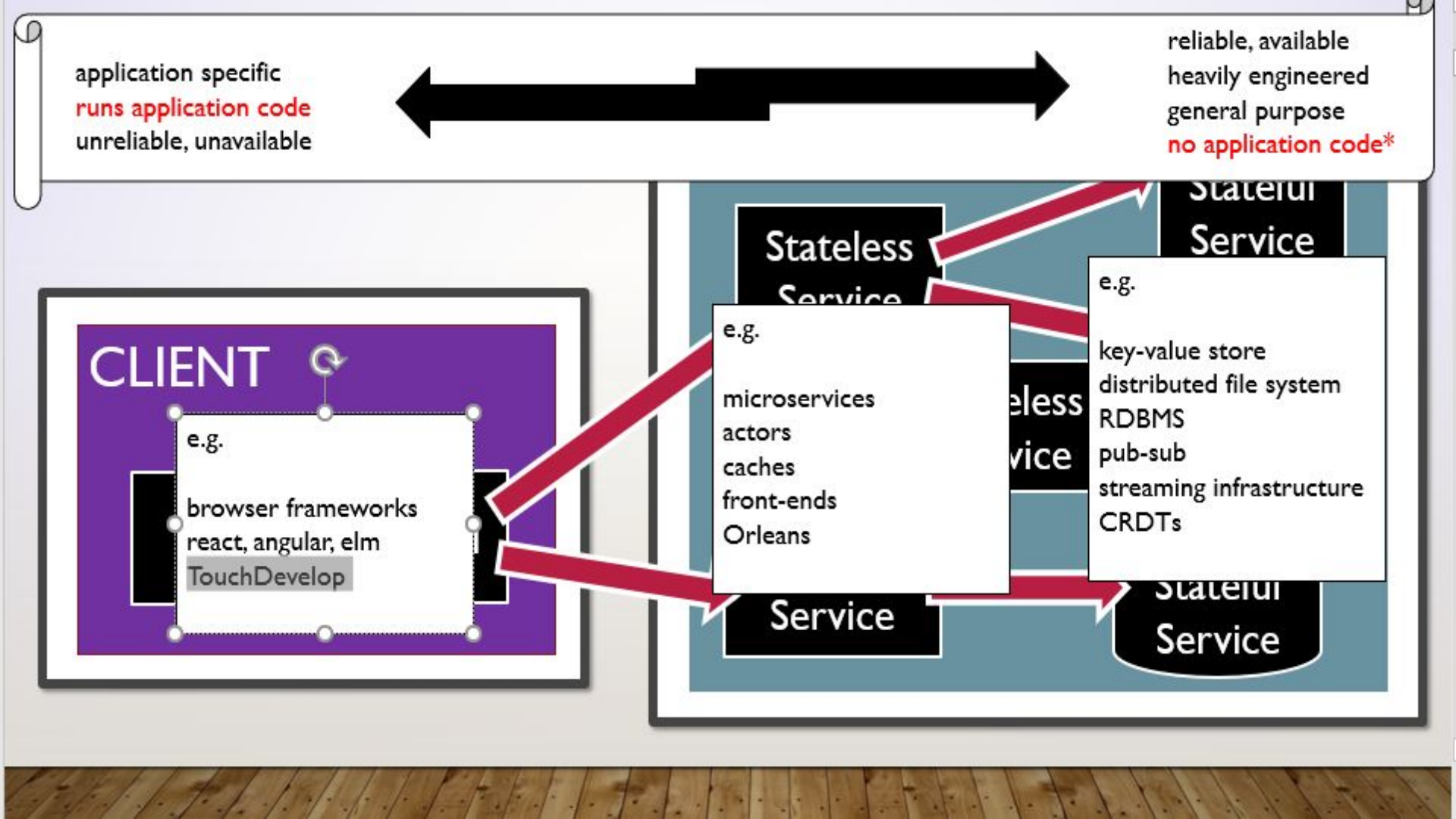
PERFECT STORM OF COMPLEXITY

- security,
  concurrency,
  parallelism,
  distribution,
  consistency,
  failures,
  dynamic topology.

EMERGING PATTERN

COMPOSED SERVICES




application specific
runs application code
unreliable, unavailable
reliable, available
heavily engineered
general purpose
no application code*
CLIENT
e.g.
browser frameworks
react, angular, elm
TouchDevelop
Stateless
Service
e.g.
microservices
actors
caches
front-ends
Orleans
Stateful
Service
e.g.
key-value store
distributed file system
RDBMS
pub-sub
streaming infrastructure
CRDTs
Service
Stateful
Service

Damien Zufferey

- PhD: IST Austria (2013)
- Postdoc: MIT CSAIL (2013-2016)
- Research group leader: MPI-SWS (2016-...)

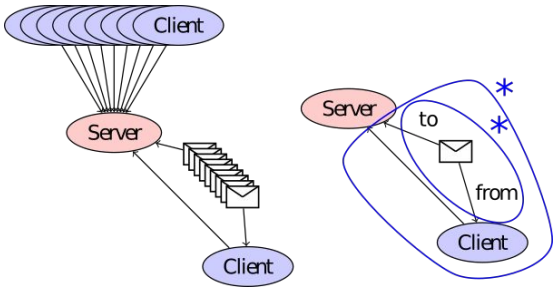
Research related to distributed systems:

- Verification of mobile systems (pi-calculus)
- Programming abstractions for messages and faults
- Coordination for CPS

More info at **http://dzufferey.github.io/**

Damien Zufferey: Research in Pictures

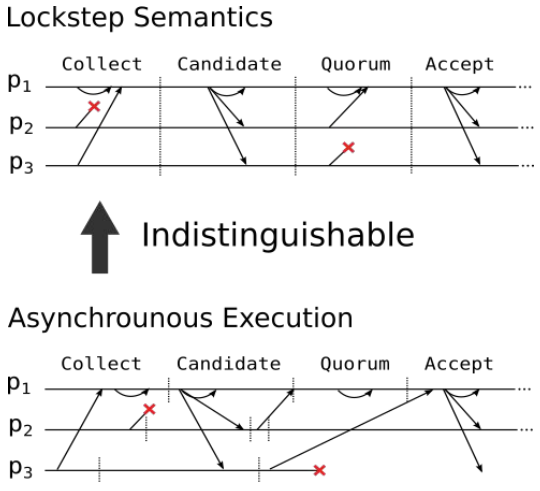
Verification of mobile processes



- Process creation
- Changing communication topology
- ...

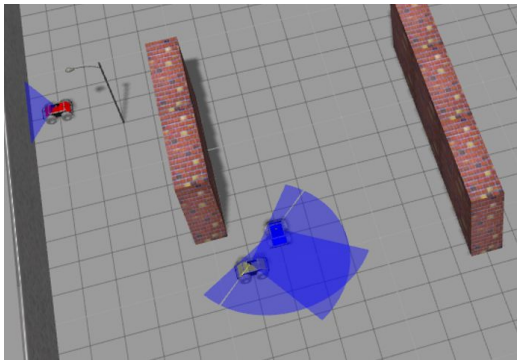
Fault and messages:

Using communication-closure to simplify programming and verification



Coordination for CPS

When messages carries informations about the “real” world



Aleksandar Prokopec

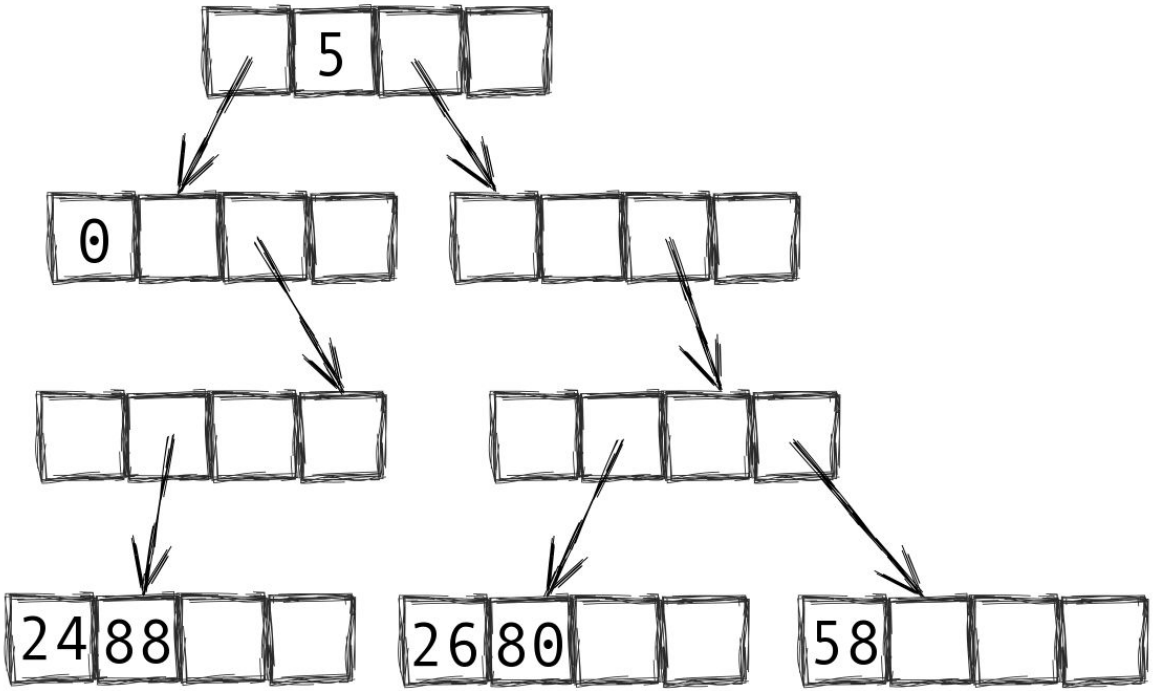
- 2009-2014 PhD at EPFL
  - Scala Team
- 2014-2016 Google
  - Google Maps
- 2016-2018 Principal Researcher at Oracle Labs
  - GraalVM

Research:

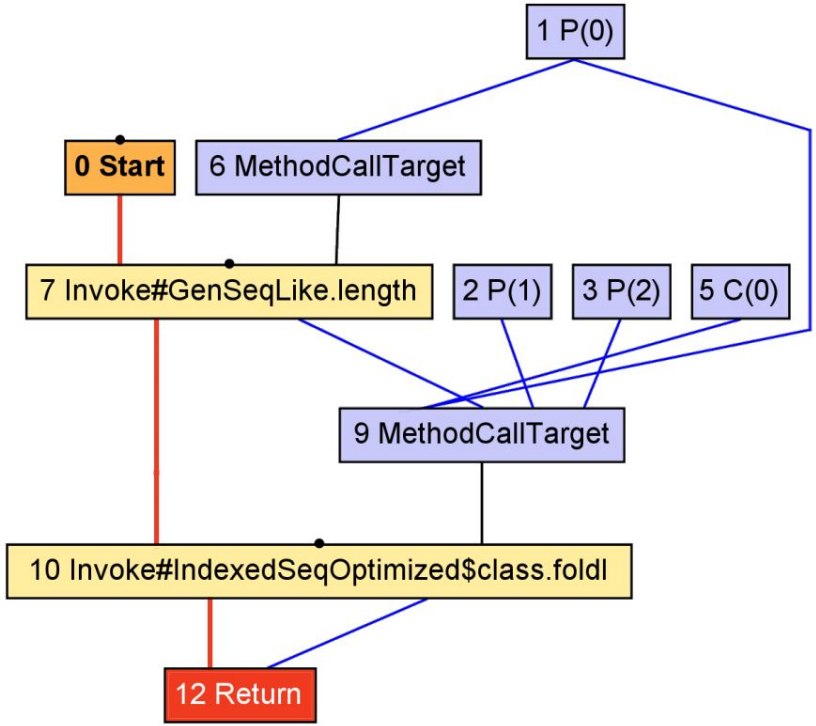
- concurrency, non-blocking data structures
- programming models for distributed systems
- compilers, optimizations, type systems

Papers, CV, other details: **http://aleksandar-prokopec.com**

Aleksandar Prokopec: Concurrent Data Structures

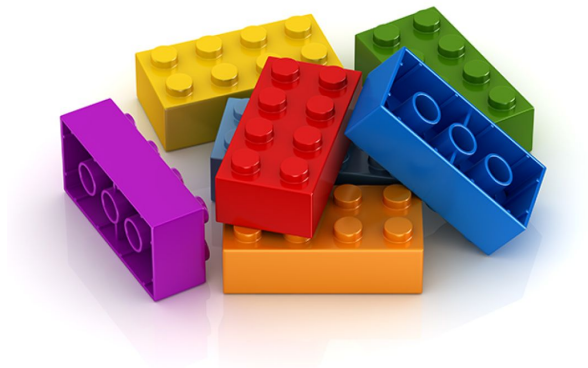


Aleksandar Prokopec: Compilers



Aleksandar Prokopec: Distributed Computing

```
process {
  val work = broadcast(workers)
  val decisions = paxos(leaders)
  while (true) {
    work ! receive(decisions)
  }
}
```



Wolfgang De Meuter

Prof @ Vrije Universiteit Brussel (Software Languages Lab)

Topics of my team:

- Distributed & concurrent programming languages
  Reactive Programming languages (procedural, logic)
- Reactive Big Data processing languages

Current Application **Themes** (project portfolio):

- Participatory Sensing (citizen science, crowd sensing)
- Cyber Physical Systems
- Internet of Things
- Security by CEP
- Smart Grids

Technologies:

- Scheme, Clojure, Lisp
- Smalltalk, AmbientTalk
- Spark, Flink, ….
- JavaScript, TypeScript
- Scala, Haskell, ...

Topic #1: Concurrent & Distributed Languages

- AmbientTalk, AmbientTalk/M, AmbientTalk/R

  “Failure is the rule rather than the exception”

- Clojure extension with STM + Futures + actors

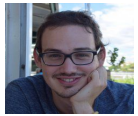
  “One paradigm doesn’t fit all”

Janwillem Swalens

2016

- Static Analysis of Actor-based languages

  “Prevent your actor from exploding”



Quentin Stievenart

2017

- Reactive Programming for Actor Coordination

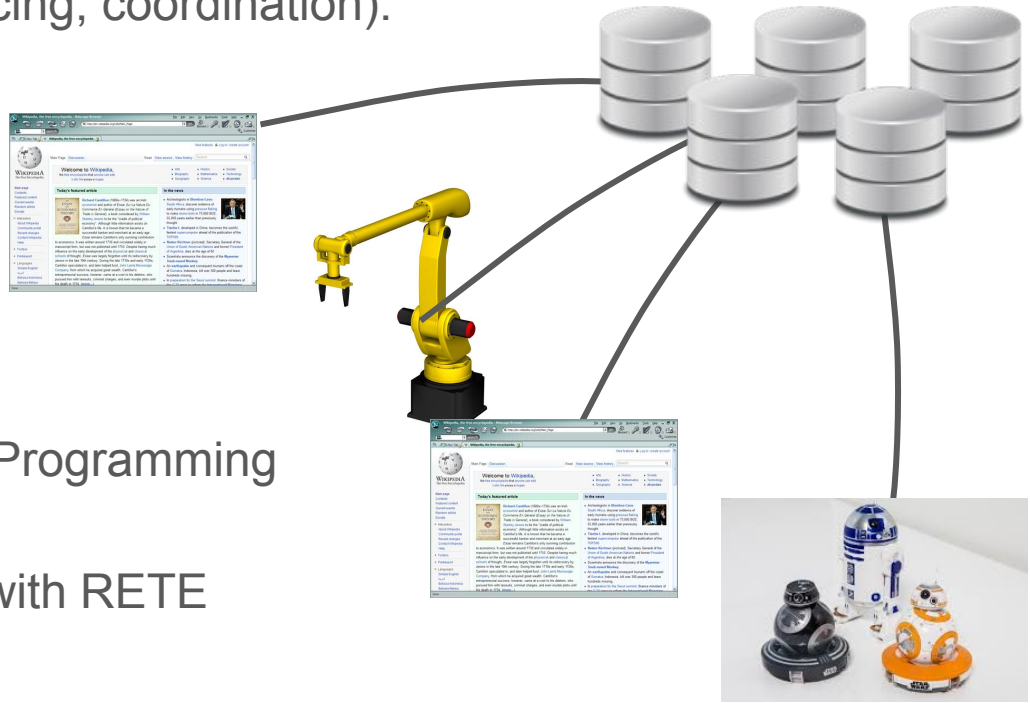
  “Coordination logic = reacting to complex message patterns”

Humberto Avila Rodriguez

Topic #2: Reactive Programming Languages

Combining reactive programming with distribution, multiple external signals, imperativeness/loops (e.g. sequencing, coordination):

The Actor-Reactor model



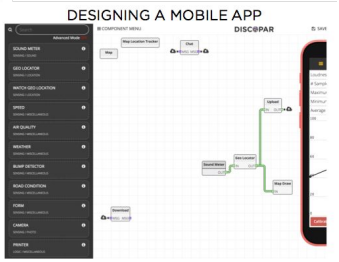
Fundamentals of RP:

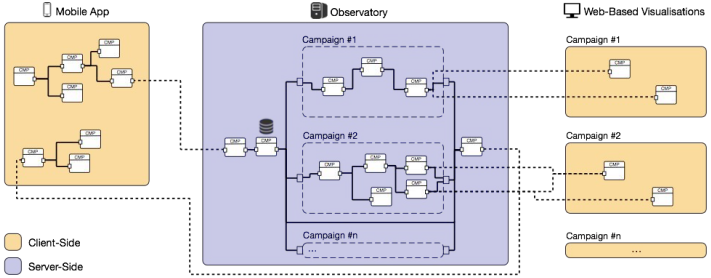
- Reactive VM & instruction set
- Higher Order Reactors 4 Live Programming
- Reactive Meta-programming
- Logic Reactive Programming with RETE

Topic #3: Reactive Big Data Processing Languages

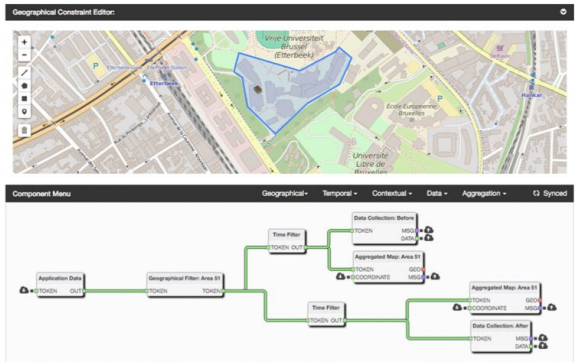
Jesse Zaman

DISCOPAR





SERVER-SIDE DATA PROCESSING



TZU-CHUN (Gina) CHEN

TU Darmstadt - Germany (2014- 2018)
Postdoc, working with Prof. Patrick Eugster
ERC *LiveSoft project*, *DSP group*

University of Torino - Italy (2013- 2014)
Postdoc, working with Prof. Mariangiola Dezani
and Prof. Luca Padovani

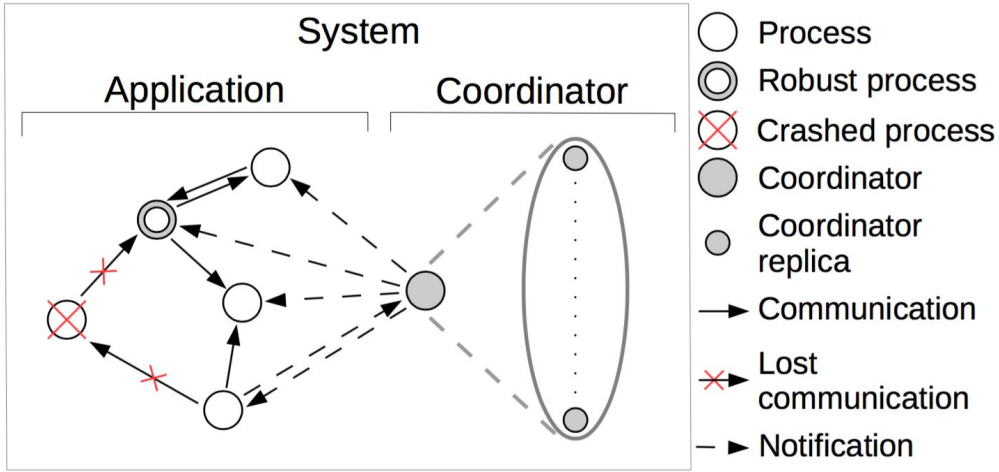
PhD -- Queen Mary, University of London, UK (2013)

Research interests

Type systems for distributed programming
PL for safe&efficient failure handling in DS/IoT
Safe and secure communication for DS/IoT

Challenges in DS Failure Handling @ N Processes

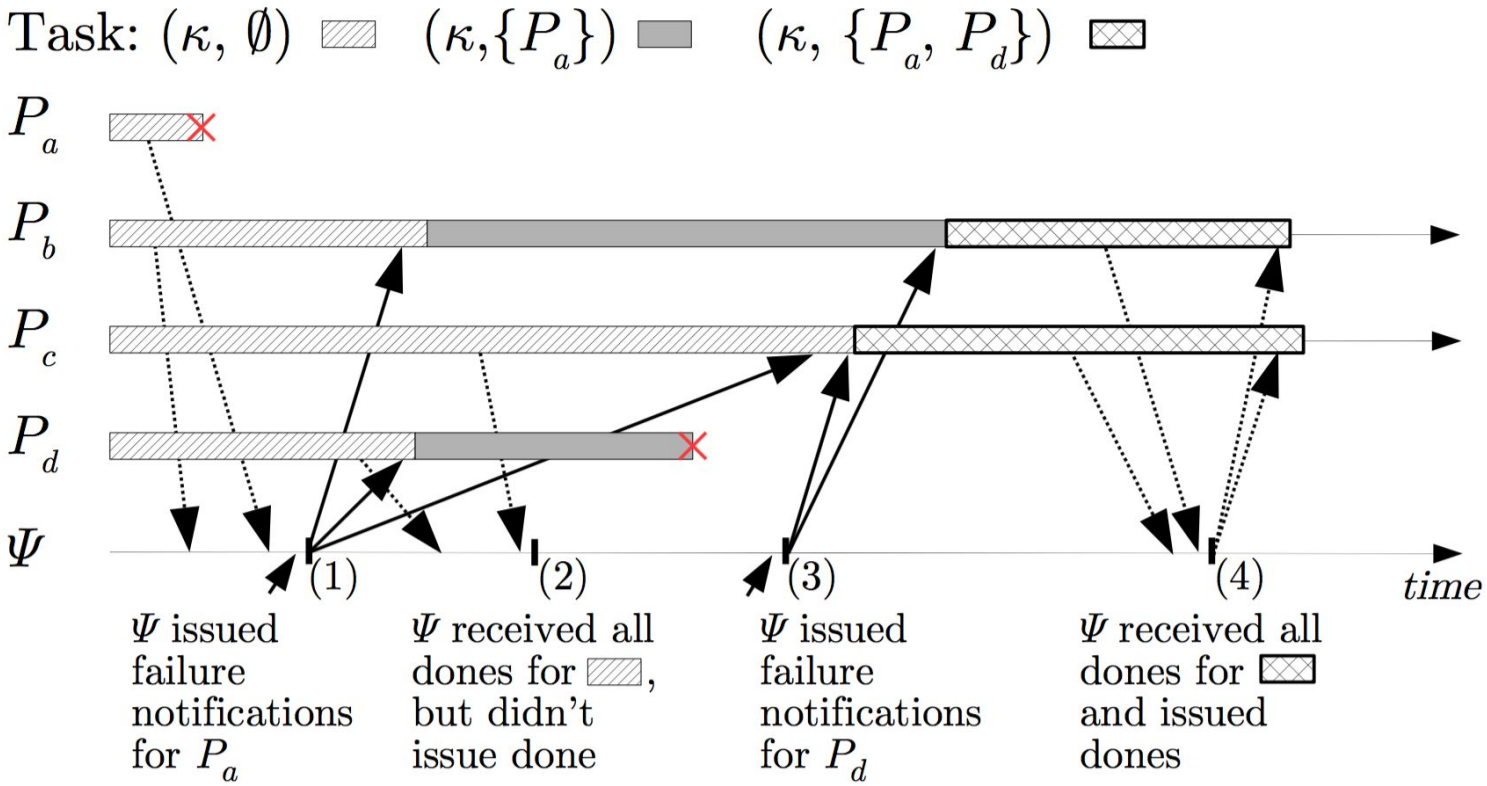
- **Asynchrony, concurrency, failure (process crash)**
  - *It is **impossible** for processes to reach an agreement*
    -- by Fischer, Lynch, Paterson JACM' 85
- **We introduce a coordinator Ψ to the system for failure handling**



Ensuring communication safety and system progress:

- From PL perspective
- Session-based process calculus with failure handling capability
- Session-type-based type disciplines

Non-trivial DS Failure Handling



Non-trivial DS Failure Handling

Related Publications

A Type Theory for Robust Failure Handling in Distributed Systems. FORTE 2016 --- no crash, no coordinator model

A Typing Disciplines for Statically Verified Crash Failure Handling in Distributed Systems. ESOP 2018 (accepted) -- with coordinator model

Ongoing works

Formalising Zookeeper for Spark Failure Handling--PhD Malte Viering

Implementing Our Coordinator Model for Spark--PhDMalte Viering

Failure Handling/Safe Communication for IoT Applications-PhD Seema Kumar

Peter Van Roy

Université catholique de Louvain – Belgium

Programming Languages and Distributed Computing

LightKone H2020 project: Lightweight Computation for Networks at the Edge (2017-2019)

Concepts, Techniques, and Models of Computer Programming (MIT Press, 2004)

Taking advantage of weak synchronization: convergent computation and synchronization-free services

Convergent computation

- Computation is always converging to a result
  - Generalizes eventual consistency to simplify programming
  - Automatically maintains correctness of program data with respect to the external world
  - Naturally tolerates node and communication problems, e.g., on edge networks
- Lasp (“Lattice Programming”) **`lasp-lang.org`**
  - Programming language for writing convergent computations
  - Basis for an edge computing platform being implemented in the LightKone project

Lasp example

- Sets connected with a map (Erlang syntax):

```
S1=declare(set),
bind(S1, {add, [1,2,3]}),
S2=declare(set),
map(S1, fun(X)->X*2 end, S2).
```

- Deterministic dataflow functional semantics
- This is a convergent computation
  - Tolerates node and communication failures
  - Keeps consistency using only weak synchronization
- It has an efficient implementation
  - Graph of CRDTs connected with operations
  - Communication layer based on hybrid gossip

unbalancedparentheses Follow

Federico Carrone. A happy member of The Erlang, Rust/ML and Lisp Evangelism Strikef...

May 9 · 8 min read

Lasp: a little further down the Erlang rabbithole

A few years ago I found Lasp: *"a suite of libraries aimed at providing a comprehensive programming system for planetary scale Elixir and Erlang applications"*. At this point it should come as no surprise for you to learn that here at Not a Monad Tutorial we are are interested in distributed systems and Erlang. After playing a little bit with Lasp I watched a few talks by its creator: Christopher Meiklejohn. After watching his talk "Distributed, Eventually Consistent Computations" I decided it was time to interview Christopher.

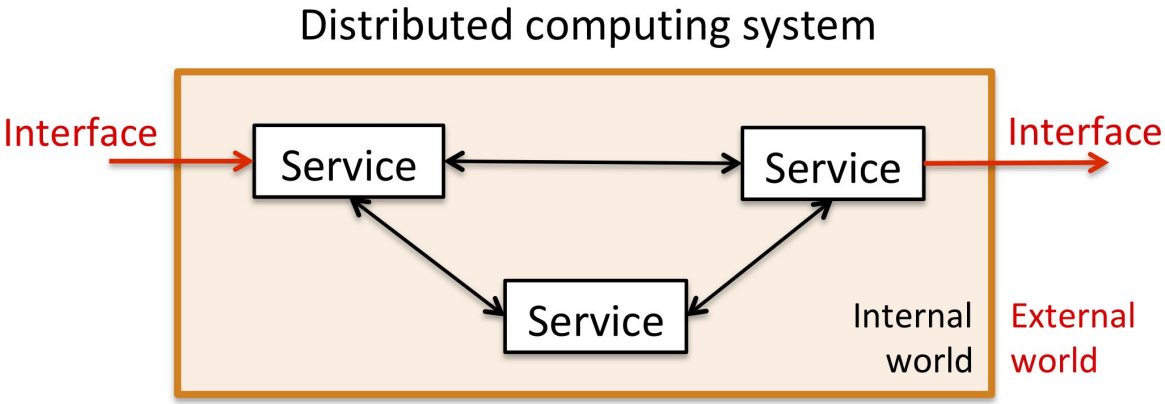
Reach me via twitter at @unbalancedparen if you have any comments or interview request for This is not a Monad tutorial. **Stay tuned!**

Discuss and vote at lobsters, reddit and hn.

What is Lasp?

Originally, Lasp was a programming model designed for deterministic distributed computing with weak synchronization. Lasp's

Synchronization-free services



- Most existing services synchronize unnecessarily at their APIs
- Instead, give the *overall system* a synchronization boundary
  - Inside the boundary, *all services use weak synchronization* to interact
  - Strong synchronization is only needed at the boundary, to interface with the external world

Programming Language Frontends

- Abstractions
- Notations
- Analyses
- Editors

Sebastian Erdweg

TU Delft

Language Design — Domain-Specific Languages

CorrI: programmable event correlation in CEP

```
with slidingWindow(TVReviews, 1 Month)
  correlate {
    release from TVReleases
    reviews(5) from TVReviews
    where
      distinct(reviews)
      forall(reviews)(_.rating >= 3.5)
      forall(reviews)(equal(_.model, release.model))
    yield release
  }
```

i3QL: privacy-aware, distributed, incremental LINQ

```
1 val students: Table[Student] = new Table[Student]()
2
3 val sallies: View[Student] =
4  (SELECT (*) FROM students
5   WHERE (s => s.firstName == "Sally")).asMaterialized
6
7 students.add(new Student("Sally", "Fields"))
8 students.add(new Student("George", "Tailor"))
9
10 sallies.foreach(s => println(s.lastName))
11 // prints: "Fields"
12
13 students.add(new Student("Sally", "Joel"))
14 // incremental update of sallies
```

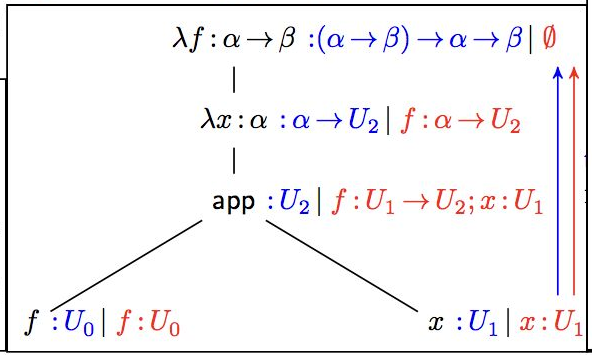
Incremental Program Analysis

65x speedup [OOPSLA'14, ASE'16]

FindBugs: Java

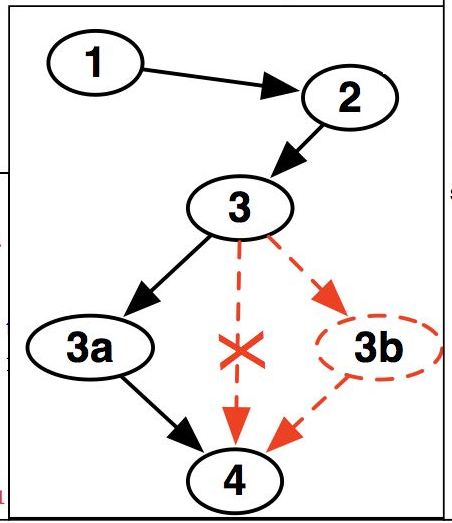
10x speedup [OOPSLA'15, ECOOP'17]

Type checking: Java, Rust



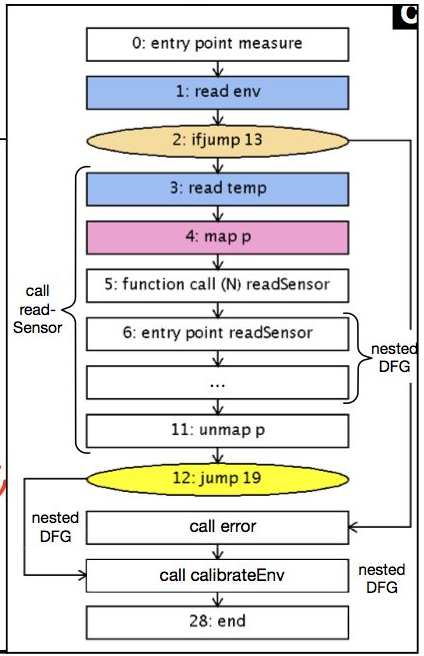
250x speedup [ASE'16]

Control flow: C, Java



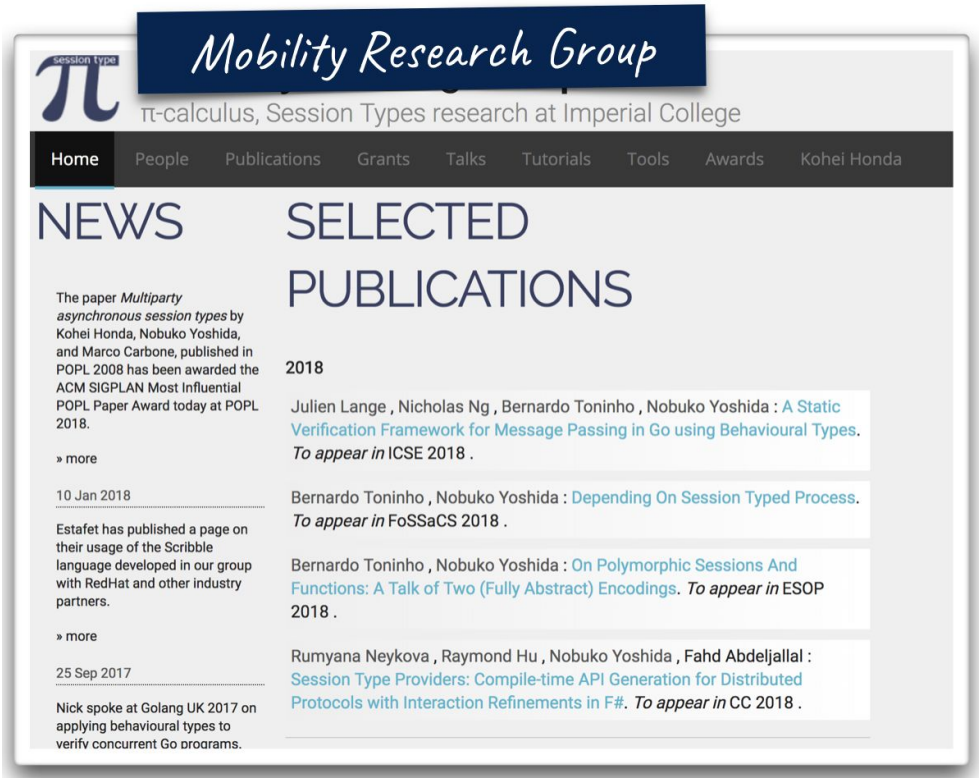
6000x speedup

Data flow: Java



Nobuko Yoshida

http://mrg.doc.ic.ac.uk



Post-docs:

Simon CASTELLAN

David CASTRO

Francisco FERREIRA

Raymond HU

Rumyana NEYKOVA

Nicholas NG

Alceste SCALAS

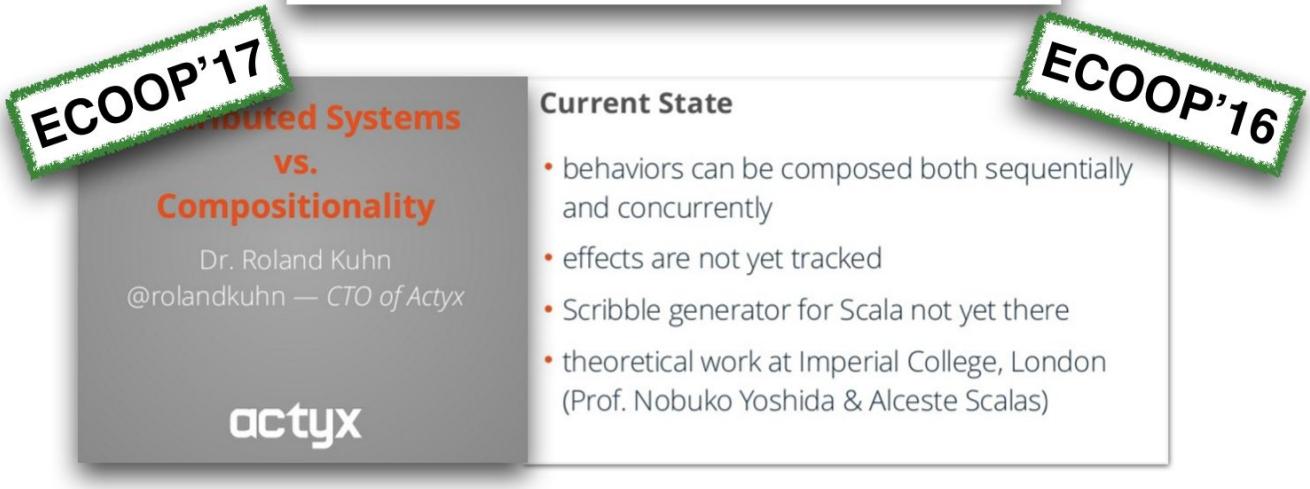
PhD Students:

Assel ALTAYEVA

Juliana FRANCO

Eva GRAVERSEN

Interactions with Industries



the morning paper

ICSE'18

an interesting/influential/important paper from the world of CS every weekday morning, as selected by Adrian Colyer

Home About InfoQ QR Editions Subscribe

A static verification framework for message passing in Go using behavioural types

JANUARY 25, 2018

tags: Concurrency, Programming Languages

A static verification framework for message passing in Go using behavioural types Lange et al., *ICSE 18*

With thanks to Alexis Richardson who first forwarded this paper to me.

We're jumping ahead to ICSE 18 now, and a paper that has been accepted for publication there later this year. It fits with the theme we've been exploring this week though, so I thought I'd cover it now. We've seen verification techniques applied in the context of **Rust** and **JavaScript**, looked at the integration of **linear types in Haskell**, and today it is the turn of Go!

SUBSCRIBE

never miss an issue! The Morning Paper delivered straight to your inbox.

SEARCH

type and press enter

ARCHIVES

Select Month

MOST READ IN THE LAST FEW DAYS

Selected Publications 2017/2018

- **[CC'18]** Rumyana Neykova , Raymond Hu, NY, Fahd Abdeljallal: Session Type Providers: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#.
- **[FoSSaCS'18]** Bernardo Toninho, NY: Depending On Session Typed Process.
- **[ESOP'18]** Bernardo Toninho, NY: On Polymorphic Sessions And Functions: A Talk of Two (Fully Abstract) Encodings.
- **[ESOP'18]** Malte Viering, Tzu-Chun Chen, Patrick Eugster, Raymond Hu , Lukasz Ziarek: A Typing Discipline for Statically Verified Crash Failure Handling in Distributed Systems.
- **[ICSE'18]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY : A Static Verification Framework for Message Passing in Go using Behavioural Types.
- **[ECOOP'17]** Alceste Scala, Raymond Hu, Ornela Darda, NY: A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming.
- **[COORDINATION'17]** Keigo Imai, NY, Shoji Yuen: Session-ocaml: a session-based library with polarities and lenses.
- **[FoSSaCS'17]** Julien Lange, NY: On the Undecidability of Asynchronous Session Subtyping.
- **[FASE'17]** Raymond Hu, NY: Explicit Connection Actions in Multiparty Session Types.
- **[CC'17]** Rumyana Neykova, NY: Let It Recover: Multiparty Protocol-Induced Recovery.
- **[POPL'17]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY: Fencing off Go: Liveness and Safety for Channel-based Programming.

Gustavo Petri

IRIF - Paris Diderot - Paris 7

- Verification for weak consistency
- Language-based techniques for distributed applications

INSTITUT
DE RECHERCHE
EN INFORMATIQUE
FONDAMENTALE

PhD. INRIA - Sophia Antipolis

- True concurrency for weak memory models

PostDoc(s)

DePaul University (Chicago): Semantic models for weak memory

Purdue University (Indiana): Verification of concurrent algorithms

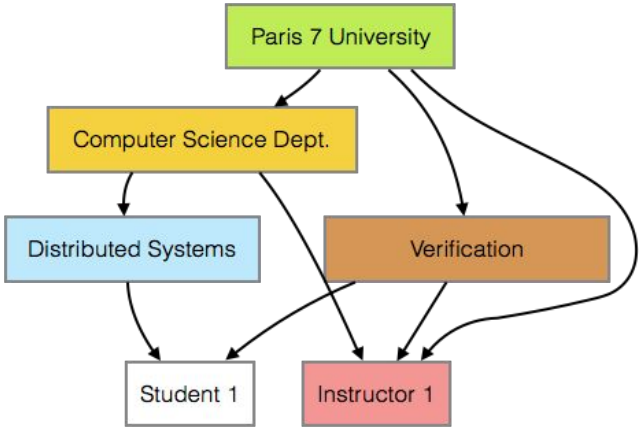
Programming Models for the Cloud

Strong semantics for actor-based transactions

- Dynamic ownership model for transactions
- Provably Linearizable
- Do we really need linearizability?

Programmable Elasticity

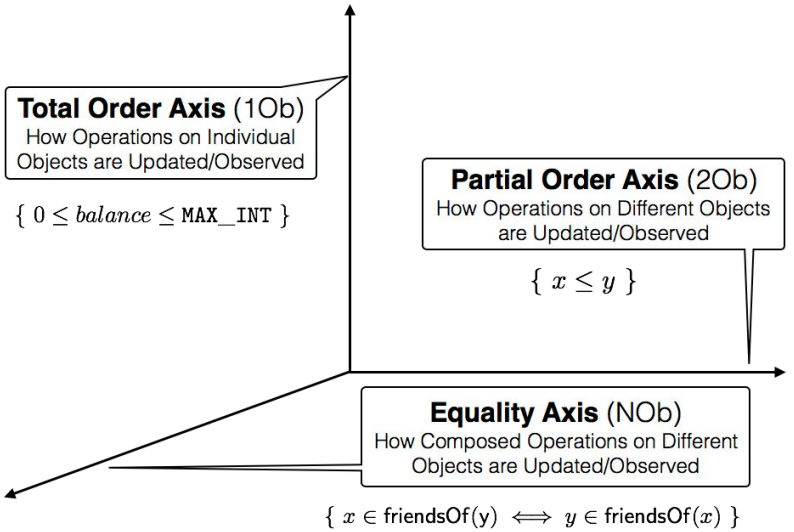
- Let the programmer help decide when and what to migrate across the DC



Verification of Applications under Weak Consistency

Dimensions of Consistency (often times invariants indicate the necessary consistency)

- Strong Consistency =>
  General Invariants
- Causal Consistency =>
  Referential Integrity, Inequalities
- Transactions =>
  Equivalence Relations



Invariants for applications using CRDTs

- What can we prove of applications using many CRDTs and/or transactions
- Verification-aided development

PL4DS – Raymond Hu

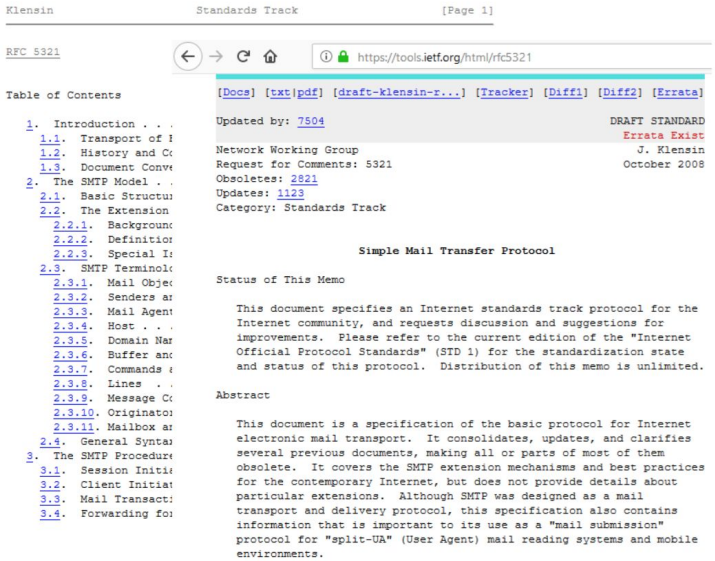
- RA at Imperial College London (in the group of Nobuko Yoshida)
  - PhD – Imperial College London 2011

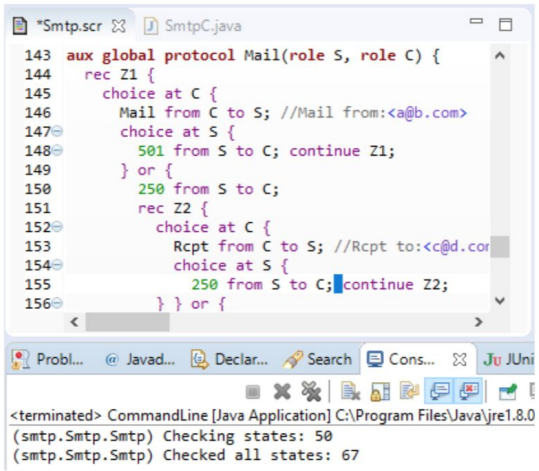
- Implementations and applications of multiparty *session types* (MPST)
  - Formal languages for message passing between concurrent processes
  - Static type system for "communication safety"
    - E.g., freedom from reception errors, deadlocks and orphan messages

- Practical session types for "mainstream" engineering languages
  - Language extensions, DSLs, runtime monitoring, code generation
  - Static typing, dynamic verification, "hybrid" approaches

Scribble: protocol-specific API generation from session types

- MPST-based toolchain for Java (also .NET, Go, ...)
  - Protocol validation: MPST syntax + model checking (+ SAT solving)
  - Endpoint programming using generated distributed APIs

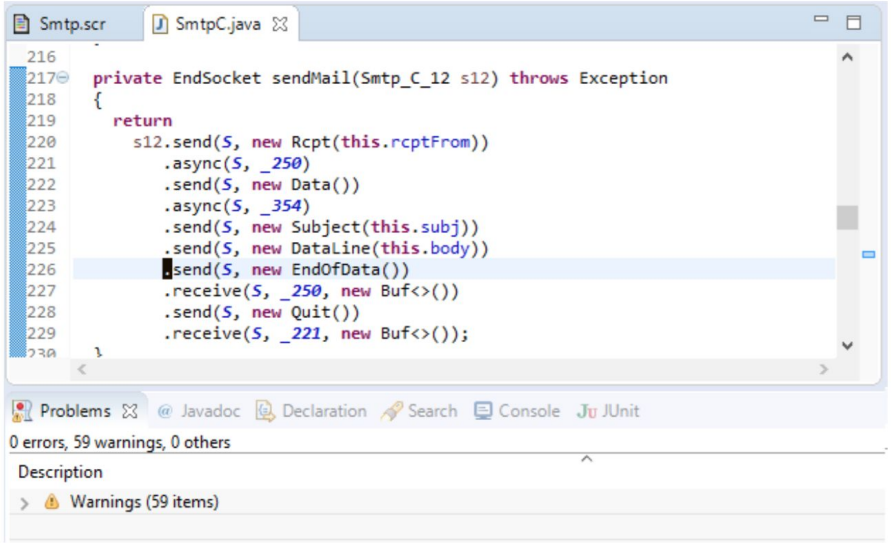




- Homepage and tutorials `http://www.scribble.org/`
- Scribble-Java source `https://github.com/scribble/scribble-java`

Scribble: protocol-specific API generation from session types

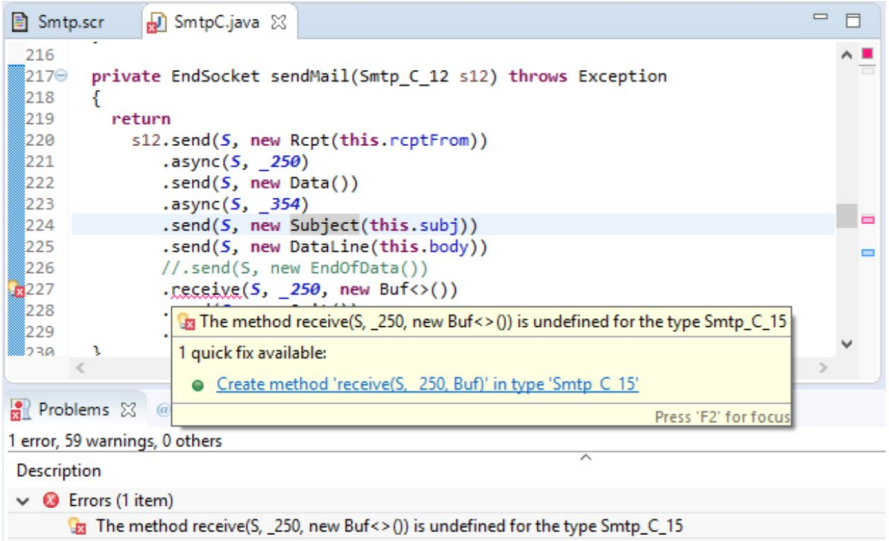
- MPST-based toolchain for Java (also .NET, Go, ...)
  - Protocol validation: MPST syntax + model checking (+ SAT solving)
  - Endpoint programming using generated distributed APIs



- Homepage and tutorials `http://www.scribble.org/`
- Scribble-Java source `https://github.com/scribble/scribble-java`

Scribble: protocol-specific API generation from session types

- MPST-based toolchain for Java (also .NET, Go, ...)
  - Protocol validation: MPST syntax + model checking (+ SAT solving)
  - Endpoint programming using generated distributed APIs



- Homepage and tutorials `http://www.scribble.org/`
- Scribble-Java source `https://github.com/scribble/scribble-java`

PL4DS challenges (from a session types view)

- Context: "high-level" (TCP-like) distributed applications
  - (Choreographic) *specifications* of real-world protocols
    - Engineering, verification, expressiveness, ...
  - *Implementations* of endpoint programs in "mainstream" languages
    - Native, distributed (heterogeneous), interoperability, ...
  - Practical safety guarantees
    - I/O primitives vs. libraries, channel linearity/aliasing/..., multithreaded/event-driven/actors/...

- Cross-discipline approaches to safe distributed programming
  - Centering on language-based protocol enforcement
  - Type checking/inference, model checking, SAT solving, code generation, meta programming (e.g., type providers), ...

Mira Mezini

TU Darmstadt

Programming languages:

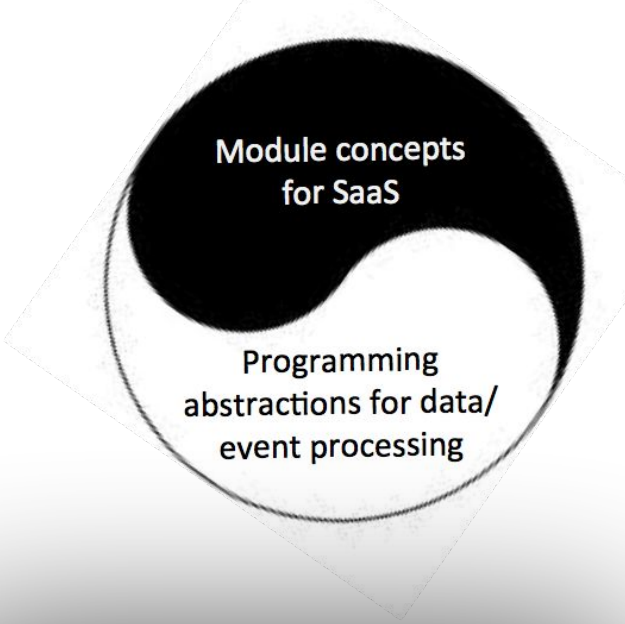
- interactive data-intensive distributed applications
- software defined networking
- privacy-preserving computations

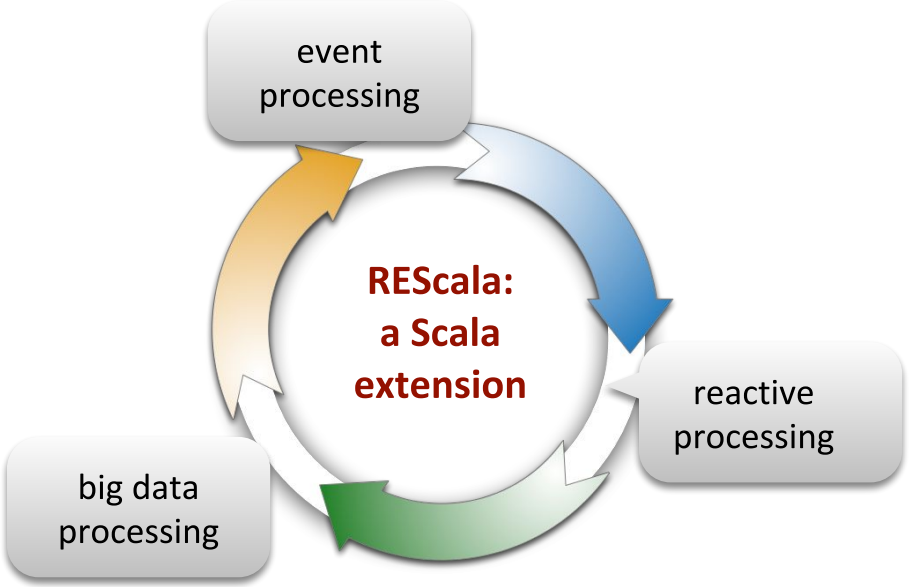
Program analysis:

- vulnerability/malware detection
- automated software engineering (“big code”)

European Research Council
Established by the European Commission
Supporting top researchers from anywhere in the world

PACE





WORKING HYPOTHESIS

Application design quality would benefit from **integration and unification of abstractions** for reactive, event and big data processing, making them **composable.**

ReScala: Reactive programming for distributed applications (REScala)

Ubiquitous callbacks!

Hypothesis: Distributed reactives reduce the complexity due to ubiquitous pub-sub systems.

- Distributed reactive values (signals/events)
- Decentralized, concurrent, fault-tolerant reactive updates
- Different levels of consistency (strong consistency via glitch-freedom; eventual consistency via CRDTs … consistency types)

BUT: Fixed join semantics (“most-recent”)

DATA/EVENT AGGREGATION …

big data programming

Spark
Sawzall
DryadLINQ
PIG … LiveLINQ HIVE
Impala
Incoop

stream processing

S4
SummingBird
Flink … Storm

concurrent programming languages

Actor … systems Join Languages

complex-event processing

StreamBase
… SQLstream
Esper StreamInsight

reactive programming

Rx FRP
Cloud … Dataflow

- zoo specialised data/event processing APIs/languages…
- huge space of feature/semantic variability
- unclear how features across domains compose/interact

Highly needed: Unification/generalization/composability based on common language foundations with systematic support for customisability.

CorrL: Unified language framework for data/even correlations (CorrL)

- Conceptual core: Joins = cartesian product of asynchronous streams
- Custom join semantics expressed via algebraic effects /handlers
- Streamlined, extensible language - correlation features as libraries.
- Correlation semantics can co-exist and even be dynamically selected.

an analogy by Ion Stoica

First cellular phones

Specialized devices

Unified device (smartphone)

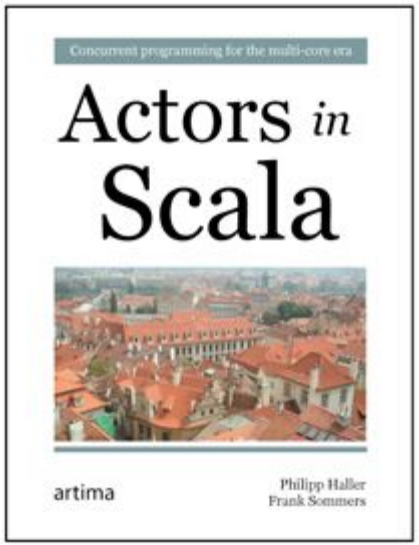
Philipp Haller

- 2014-present: KTH Royal Institute of Technology
- 2012-2014: Typesafe Inc.
- 2011-2012: Stanford and EPFL
- PhD EPFL, 2010

Research interests:

- Reactive and distributed programming
- Type systems for concurrency and distribution
- Deterministic concurrency
- Weak consistency

Reactive programming and actors



Philipp Haller, Frank Sommers
Artima Press, 2011
ISBN 978-0-9815316-5-6

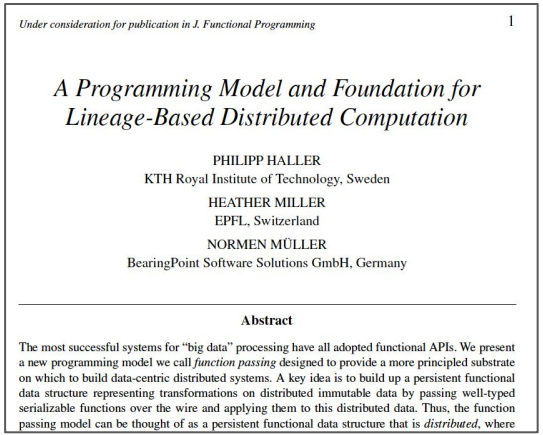
- Programming languages for reactive systems
- Actors
- Foundations
- Experimental evaluation
- Geo distribution

Linear and affine types

- Linear and affine types "for the masses":
  - scale to fully-fledged languages
  - applicable to real code
- Typestate
- "Types for concurrency and distribution"

Haller and Loiko.
LaCasa: lightweight affinity and object capabilities in Scala. OOPSLA 2016

Deterministic concurrency



Under consideration for publication in J. Functional Programming 1

A Programming Model and Foundation for Lineage-Based Distributed Computation

PHILIPP HALLER
KTH Royal Institute of Technology, Sweden
HEATHER MILLER
EPFL, Switzerland
NORMEN MÜLLER
BearingPoint Software Solutions GmbH, Germany

Abstract

The most successful systems for "big data" processing have all adopted functional APIs. We present a new programming model we call *function passing* designed to provide a more principled substrate on which to build data-centric distributed systems. A key idea is to build up a persistent functional data structure representing transformations on distributed immutable data by passing well-typed serializable functions over the wire and applying them to this distributed data. Thus, the function passing model can be thought of as a persistent functional data structure that is *distributed*, where

- Programming models
- Foundations
- Distribution
- Fault tolerance

Haller, Miller, and Müller.
A Programming Model and Foundation for
Lineage-Based Distributed Computation.
Journal of Functional Programming, 2018. To appear.

Annette Bieniusa

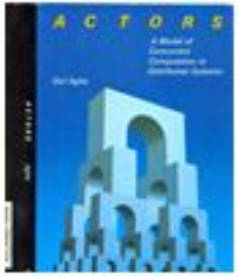
- Since 2012 Senior researcher / lecturer at TU Kaiserslautern in AG Softech
- 2011-12 Postdoc at Inria Paris / LIP6
- 2007-11 PhD at University of Freiburg (with Peter Thiemann)

Research interests

- Shared state on highly-scalable distributed systems
  - CRDTs
  - (Distributed) Software Transactional memory
- Developing correct applications on weakly-consistent data stores
- Framework for offline functionality in mobile apps

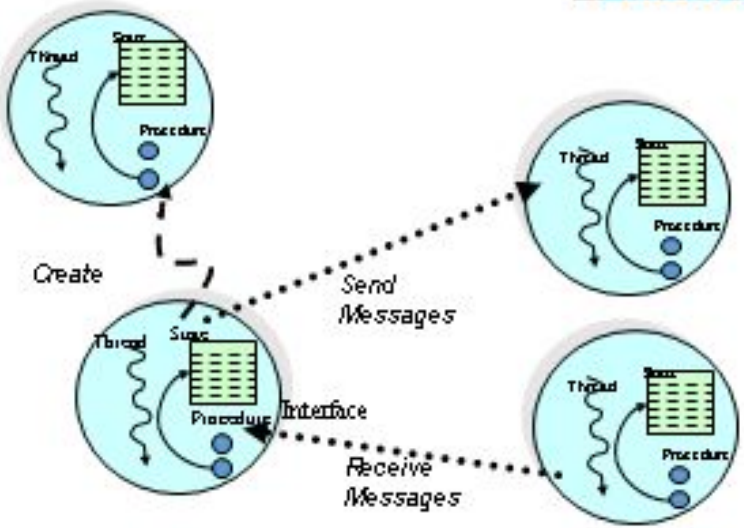
- Geo-replicated data store with low latency and high availability
- Data model: CRDTs
- Transactional Causal+ Consistency
- Transactions group updates and reads from snapshot
- Specification, verification and testing with Repliss
    - Data invariants + Invariants over history of invocations
- Event-based Parallel Temporal Logic (EPTL)

```
val chat: CrdtMap_dw[ChatId, CrdtSet_rw[MessageId]]
val message: CrdtMap_dw[MessageId, {
  author: CrdtMvReg[UserId],
  content: CrdtMvReg[String],
  chat: CrdtMvReg[ChatId],
}]

def sendMessage(from: UserId,
      content: String, to: ChatId) {
  atomic {
    val m = new MessageId
    message(m).author.assign(from)
    message(m).content.assign(content)
    message(m).chat.assign(to)
    chat(to).add(m)
  }
}

def editMessage(id: MessageId, newContent: String) {
  atomic {
    if (messages(id).exists) {
      message(id).content.assign(newContent)
    }
  }
}

def deleteMessage(id: MessageId) {
  atomic {
    val c = first(message(id).chat)
    chat(c).remove(id)
    message.delete(id)
  }
}
```



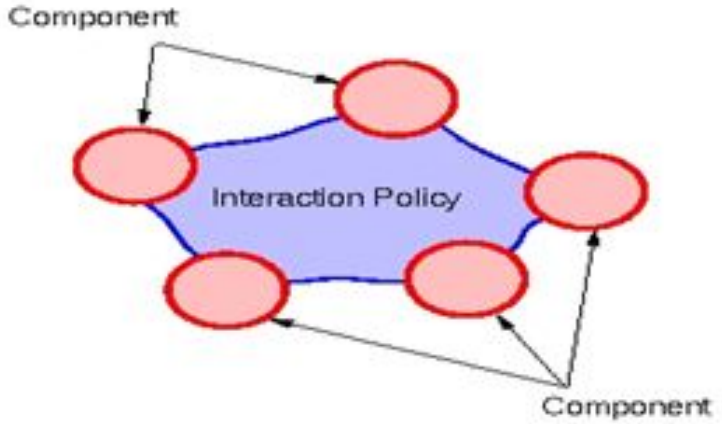
The Actor Model

- The world is naturally decomposed into autonomous objects that act concurrently
- There is no action at a distance: asynchronous communication
- Scalable, reactive and messaging applications (Twitter, LinkedIn, Facebook Chat, ..)
- Database and Cloud applications (Orleans)



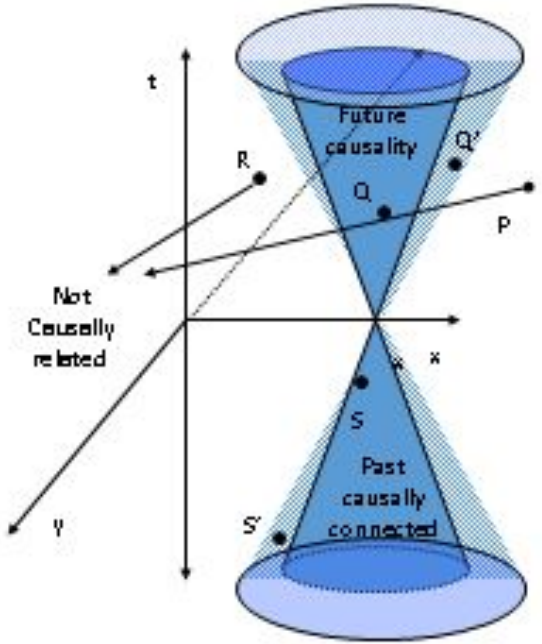
Abstracting Interaction

- How to express *coordination (e.g. synchronization intention)* modularly?
- Separate *interaction policy* from *protocols* used to implement policy
  - Multiparty Session types
  - Synchronizers
  - Streams, joins, ..
  - Workflows



Actors Sample points in Space-Time

- **Events separated in space are separated in time:**
  - Scheduling delays
  - Latency and communication delays
- **Such delays are probabilistic in nature**
- **Probabilistic cone**
  - Probabilistic Actor Programming
  - Statistical Model Checking



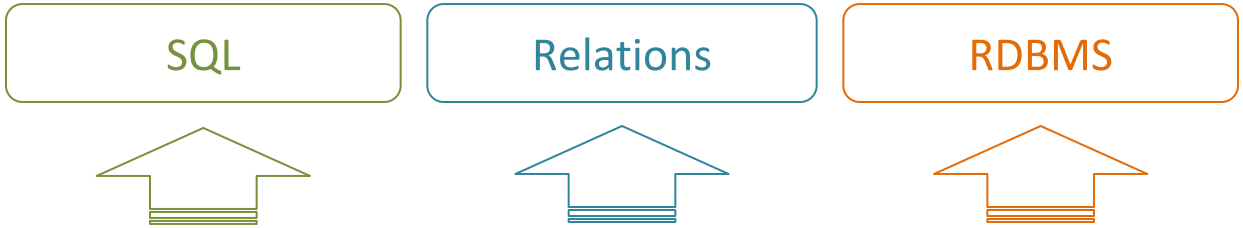
Cyberphysical Space

- **Continuous as well as discrete variables as first-class objects**
  - Location, time, motion, strain, vibration, etc. are types of physical attributes best represented by continuous variables
- **Stochastic variables as first-class objects**
  - Real-world systems (IoT, sensor networks, web search, robots) involve probabilistic analysis and statistical decisions
- **Interoperation of suitable programming abstractions:**
  - Actors
  - Constraints

Declarative Data Processing and Mosaics

A DBMS and Big Data Processing Perspective

A Billion $$$ Mantra...



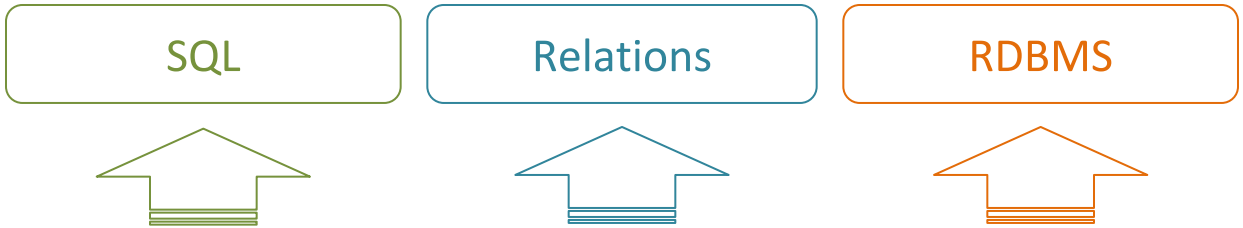
Declarative Data Processing

An effective, formal foundation based on relational algebra and calculus (Codd ’71).

A simple, high-level language for querying data (Chamberlin ’74).

An efficient, low-level execution environment tailored towards the data (Selinger ’79).

With 40+ years of success...



Declarative Data Processing

Is Being Revised

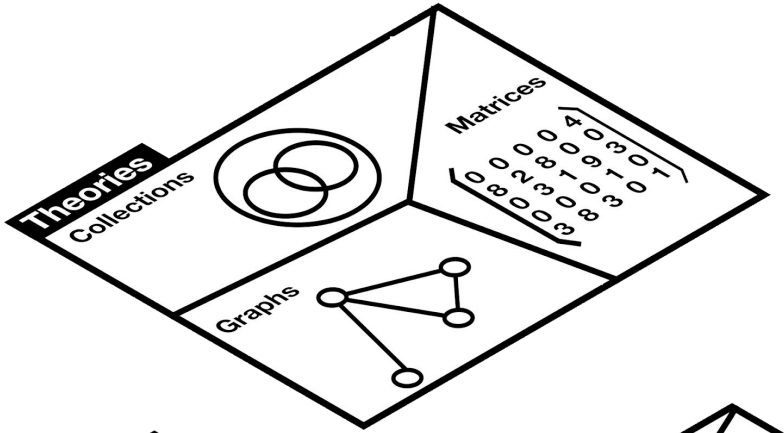
SQL | Relations | RDBMS

Declarative Data Processing

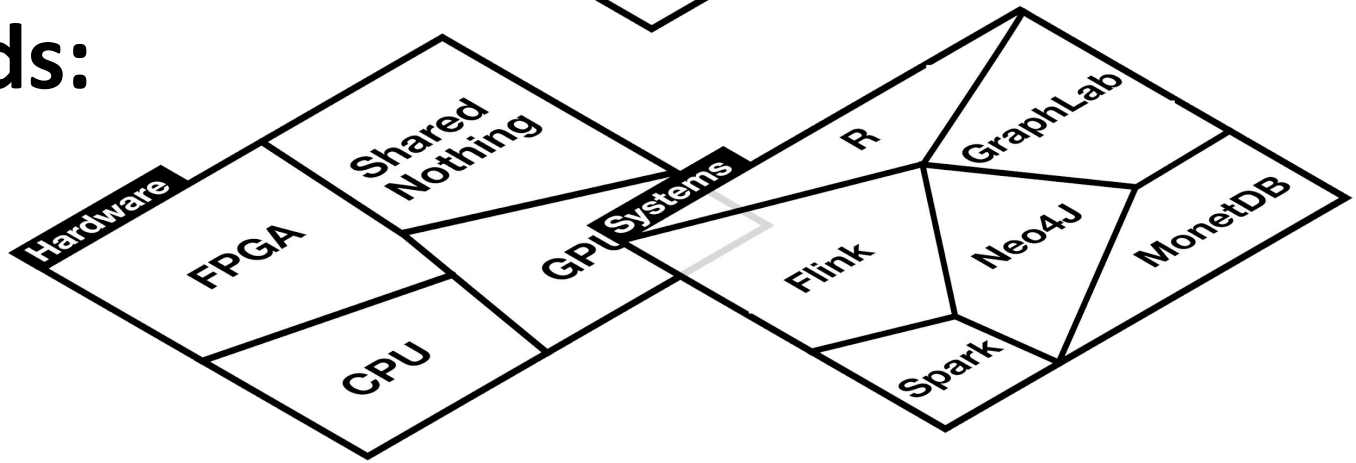
Second-Order Functions | Distributed Collections | Parallel Dataflow Engines

Mosaics

Frontend:



Backends:



Research

1. Unifying Modelling Across Theories
2. Cross Theory Optimization

3. Optimizing Across Engines
4. Predicting and Learning Program Runtimes

5. Optimizing Across Hardware
6. Generating Hardware-Targeted Code