Philipp Haller

- Associate Professor at KTH Royal Institute of Technology in Stockholm, Sweden

- Previous positions at Typesafe Inc., Stanford University, and EPFL

- PhD 2010 EPFL, Switzerland

- Research interests:
  Programming languages, concurrent and distributed programming, type systems, static analysis

Goals

- ***Programming languages for distributed systems*** that provide high scalability, reliability, and availability

- Prevent hazards in distributed systems > concurrent systems

Static and dynamic approaches

Programming Models

- ***Concurrent and distributed programming***
  - Scala Actors (Haller & Odersky 2009)
  - Scala Joins (Haller & Van Cutsem 2008)
  - Scala futures (2012), Scala Async (Haller & Zaugg 2013)
  - Asynchronous observables (Haller & Miller 2019)
- ***Deterministic concurrency***
  - Reactive Async (Haller et al. 2016)

Production use at BBC, The Guardian, Twitter, ..

Wide production use

Ensuring Language-based Fault-Tolerance Properties

- Specific fault-tolerance mechanism:
  Lineage-based fault recovery

  - Lineage records dataset identifier plus transformations
    - Dataset resulting from application of transformations can be reconstructed using that lineage

    - Maintaining lineage information in available, replicated storage enables recovering from replica failures

- ***A widely-used fault-recovery mechanism***

Lineage-based Distributed Computation

- ***How to statically ensure fault-tolerance properties for languages based on lineage-based fault recovery?***

- Need foundations for languages based on lineages

- Example program:

```
val persons: SiloRef[List[Person]] = …

val adults: SiloRef[List[Person]] =
  persons.apply(spore { ps =>
    ps.filter(p => p.age >= 18)
  })
```

Some Results

- P. Haller, H. Miller, N. Müller. **A programming model and foundation for lineage-based distributed computation.** J. Funct. Program. 28: e7 (2018)

- proof establishing the preservation of lineage mobility

- proof of finite materialization of remote, lineage-based data

Ongoing & Future Work

- Interaction
  - When do we ***not*** have to specify “when”?
    - ***Properties*** like commutativity, monotonicity
    - ***Language constructs, abstractions***
    - ***Types***
      - Lightweight formal methods
- Latency and time

Understandable!

For all of these:
Modularity, scalability and availability as ***essential*** aspects

Challenges

- Static guarantees about fault-tolerance, availability, and consistency

- Preventing hazards of distributed systems aiming to provide high scalability, reliability, and availability