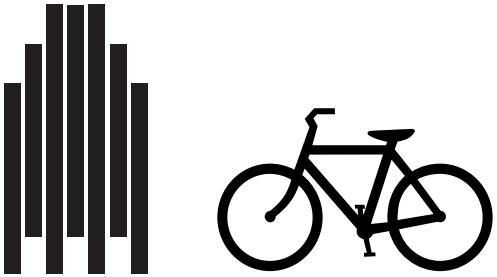
Actors

Jonas Kaiser

19. Oktober 2018

Wer bin ich, . . .

Gebürtiger Münsteraner

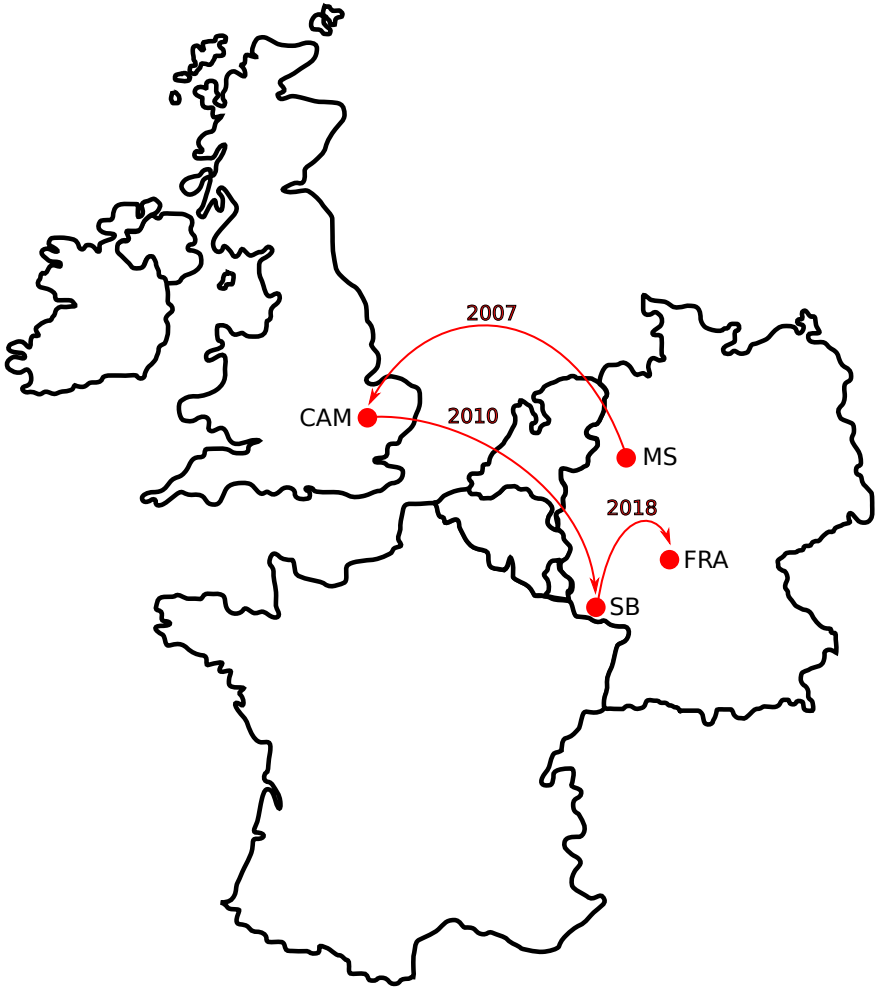


Informatik Studium / Promotion

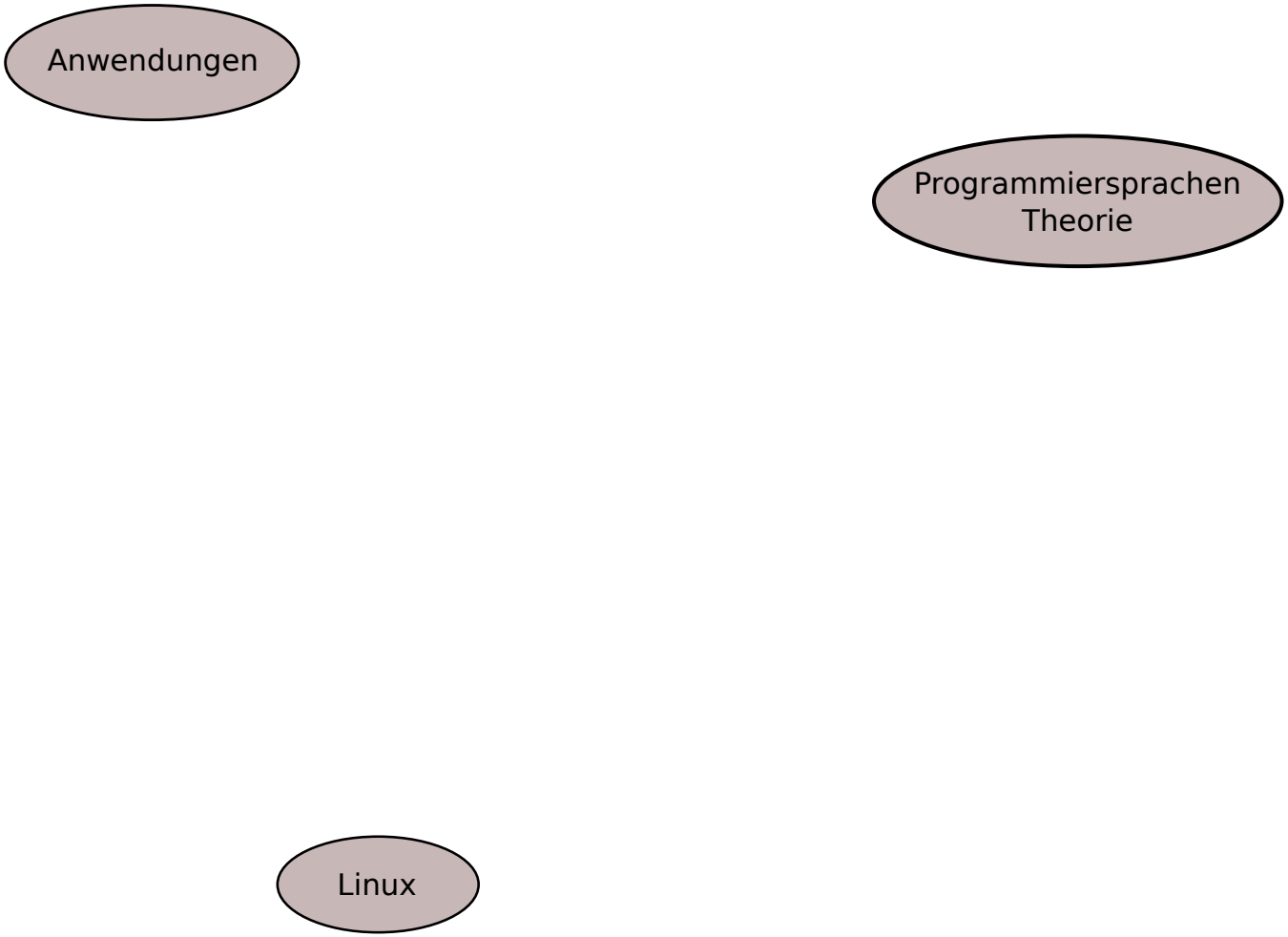
- 2010 – BA, Cambridge
- 2013 – MSc, Saarbrücken
- *bald* – PhD, Saarbrücken

Wechsel in die Wirtschaft

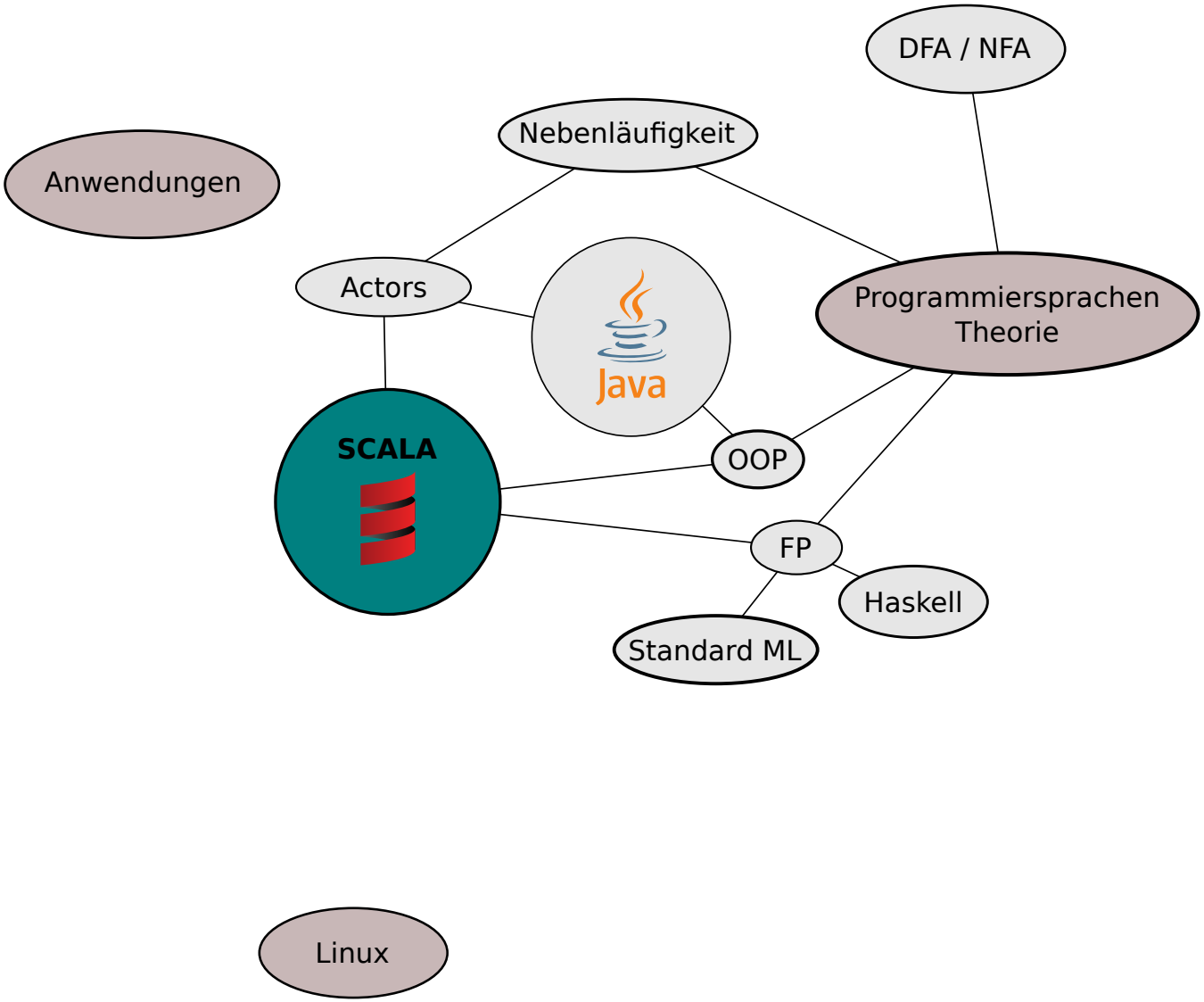
- Juni 2018



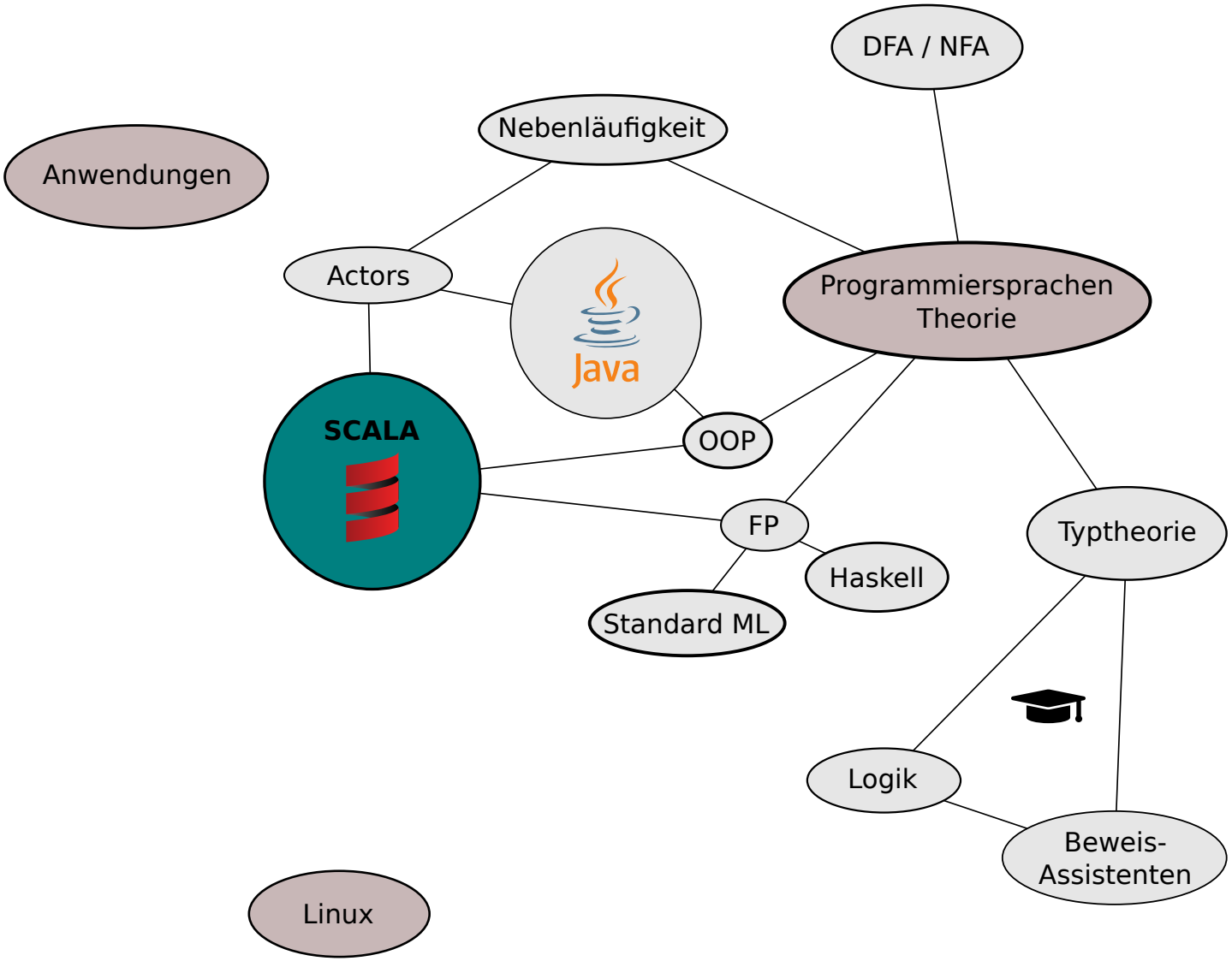
. . . und wofür schlägt mein Herz?



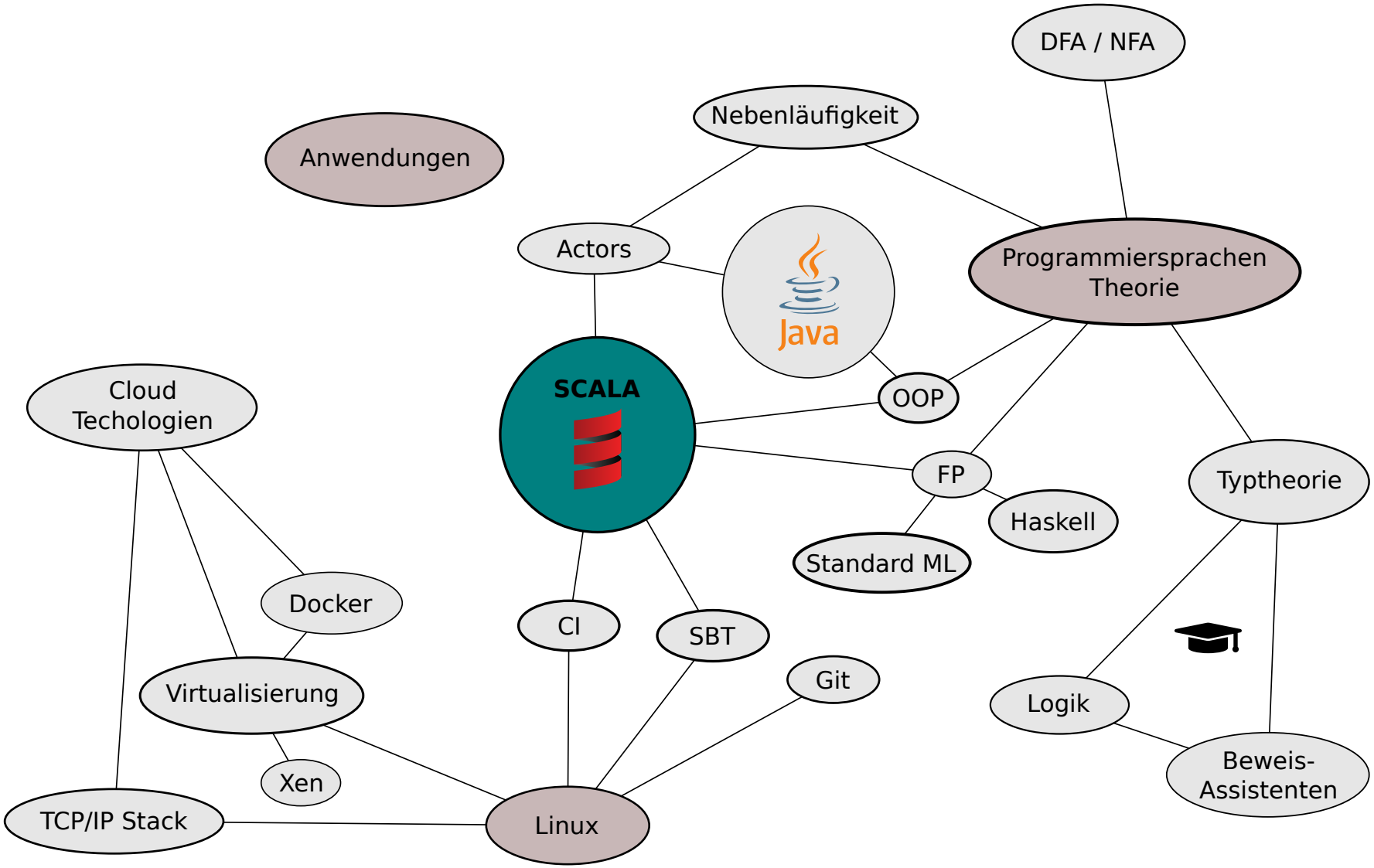
. . . und wofür schlägt mein Herz?



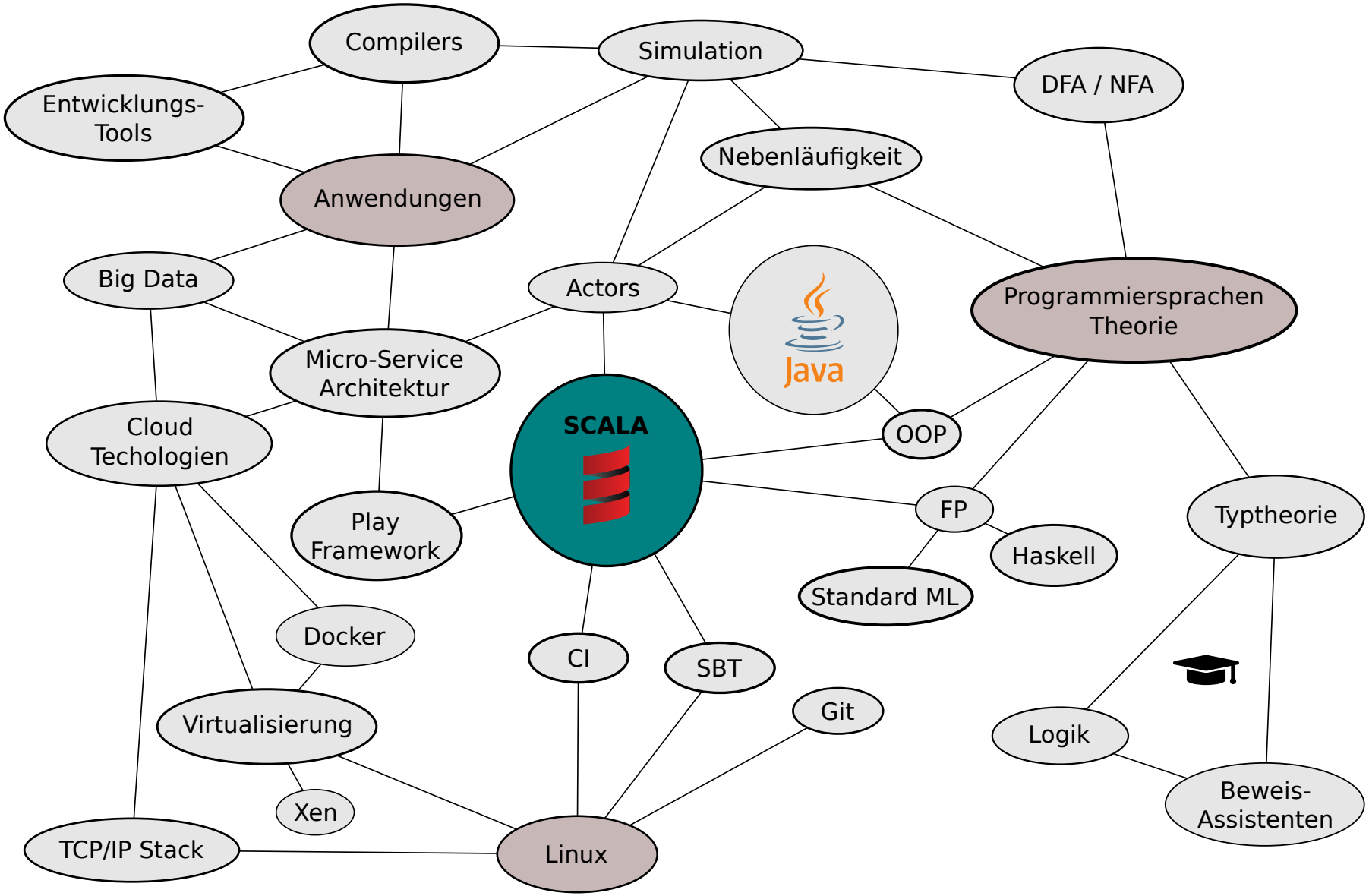
. . . und wofür schlägt mein Herz?



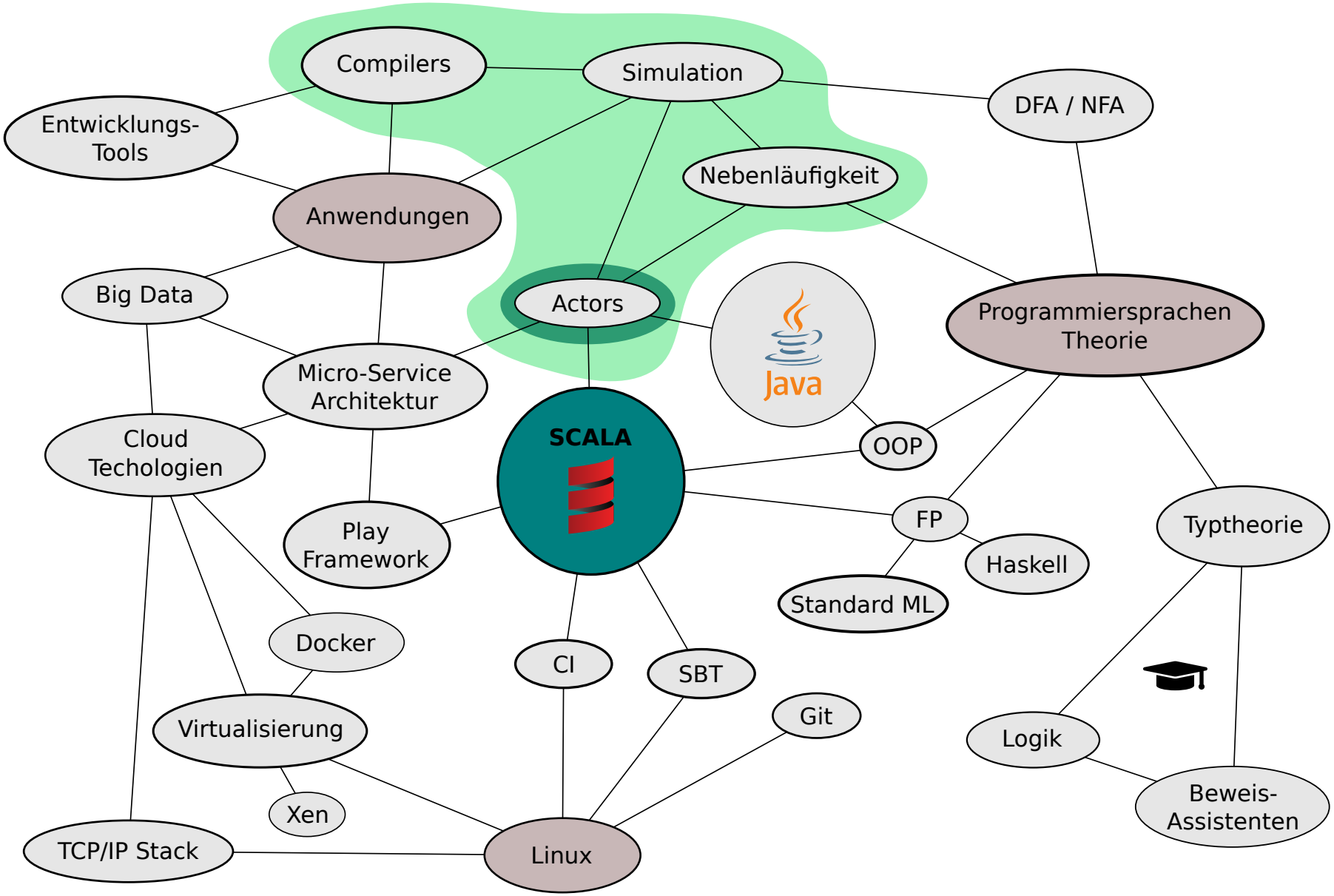
. . . und wofür schlägt mein Herz?



. . . und wofür schlägt mein Herz?



. . . und wofür schlägt mein Herz?



– Actormodell –

Ziel: Optimale Auslastung von Ressourcen

- **Früher:** Multi-Core CPUs, heterogene Prozessor Architektur, Hyperthreading
- **Heute:** Vernetzte Systeme, Cloud
  - Physisch: Server, Workstation, Cluster
  - Virtuell: Vom schlanken Container bis zum voll virtualisierten OS

Ziel: Optimale Auslastung von Ressourcen

- **Früher:** Multi-Core CPUs, heterogene Prozessor Architektur, Hyperthreading
- **Heute:** Vernetzte Systeme, Cloud
  - Physisch: Server, Workstation, Cluster
  - Virtuell: Vom schlanken Container bis zum voll virtualisierten OS

$\Rightarrow$ Wir brauchen Modelle für Nebenläufigkeit (= logische Parallelität)

Ziel: Optimale Auslastung von Ressourcen

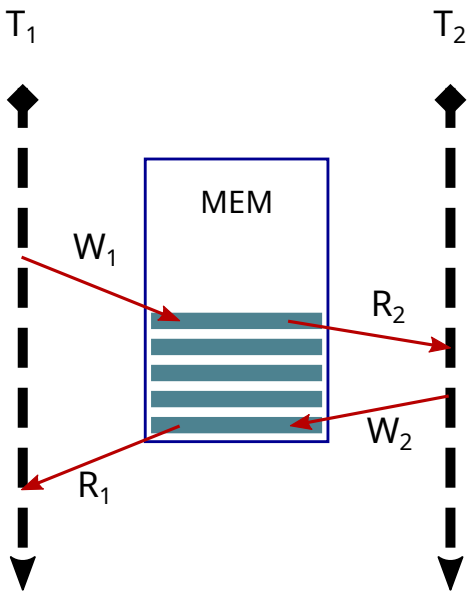
- **Früher:** Multi-Core CPUs, heterogene Prozessor Architektur, Hyperthreading
- **Heute:** Vernetzte Systeme, Cloud
  - Physisch: Server, Workstation, Cluster
  - Virtuell: Vom schlanken Container bis zum voll virtualisierten OS

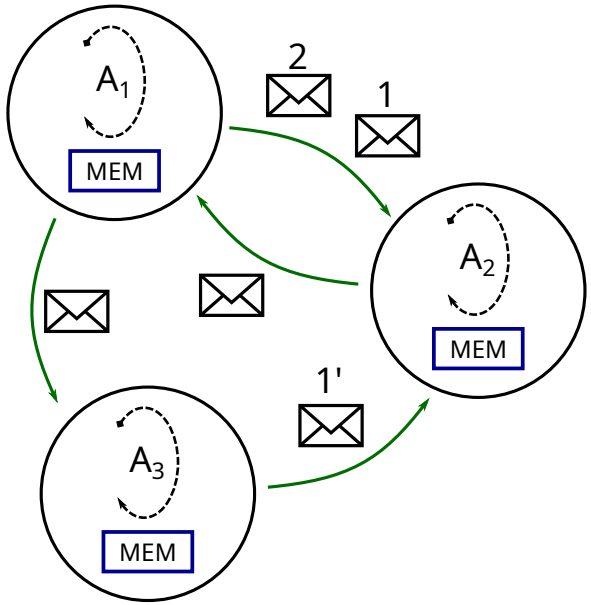
$\Rightarrow$ Wir brauchen Modelle für Nebenläufigkeit (= logische Parallelität)

Relevante Metriken

- Verständlichkeit / Benutzerfreundlichkeit
- Effizienz
- Skalierbarkeit / Elastizität
- Fehleranfälligkeit

Zwei Modelle: Threads vs. Actors[1]





- Geteilter Speicher
- Erfordert Synchronisation
- Näher an der Maschine

- Lokaler, privater Speicher
- Nachrichtenaustausch (async)
- Näher am Menschen

[1] [Hewitt, Bishop und Steiger 1973 / Agha 1985]

Actormodell: Pro und Contra

- Moderne CPUs haben lokale Caches an jedem Kern
  - Einheitlicher, konsistenter Speicher ist unrealitisch
- Verteiltes Speichermodell skaliert zu Cloud-Anwendungen
- Nachrichtenaustausch ist Grundbaustein (TCP, REST, Mail, . . .)
- Asynchroner Nachrichtenaustausch ist intuitiv
- Actors sind logisch geschlossene Einheiten
  - erleichtert dynamische Topologien
  - erleichtert Programmierung (weniger Bugs)
- Einziger Nachteil: Weniger performant aufgrund der zusätzlichen Kommunikationsebene

Actormodell: Pro und Contra

- Moderne CPUs haben lokale Caches an jedem Kern
  - Einheitlicher, konsistenter Speicher ist unrealitisch
- Verteiltes Speichermodell skaliert zu Cloud-Anwendungen
- Nachrichtenaustausch ist Grundbaustein (TCP, REST, Mail, . . .)
- Asynchroner Nachrichtenaustausch ist intuitiv
- Actors sind logisch geschlossene Einheiten
  - erleichtert dynamische Topologien
  - erleichtert Programmierung (weniger Bugs)
- Einziger Nachteil: Weniger performant aufgrund der zusätzlichen Kommunikationsebene

Fazit

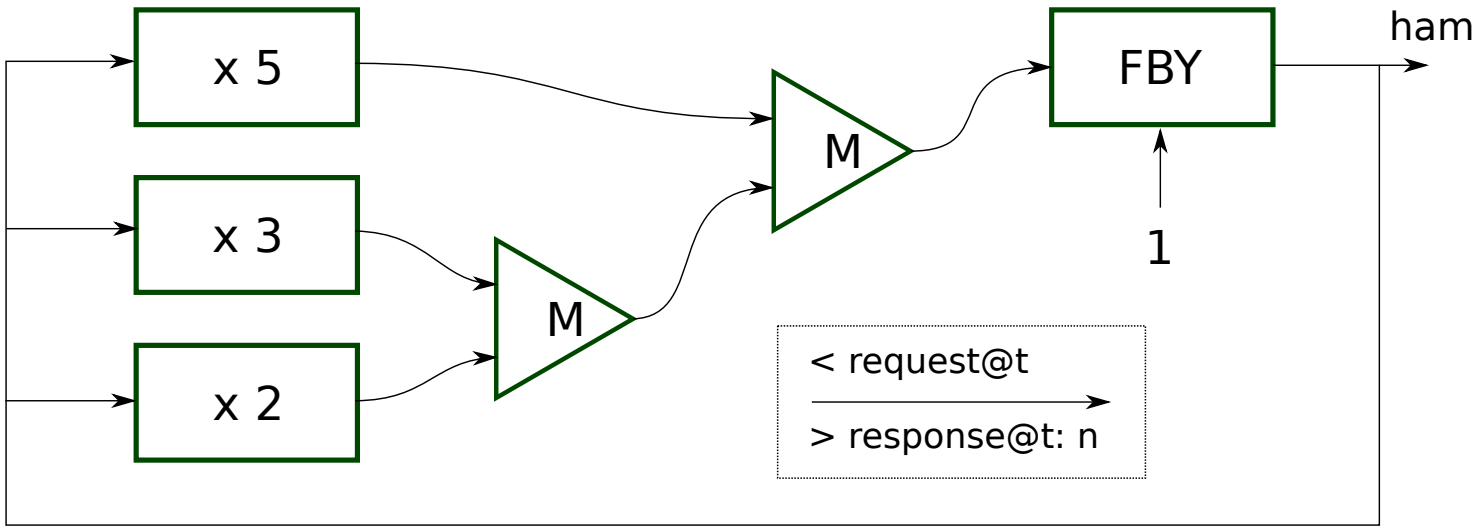
Das Actormodell ist fast immer die bessere Wahl

– Fallstudie: simLucid –

Projektübersicht, BA Thesis 2010

- Entwicklung eines Compilers inkl. Laufzeitumgebung
- Objektsprache: *Lucid* (Datenflusssprache) [Wadge und Ashcroft 1985]
  - Primärer Datentyp: Streams (Stream-Algebra)
- Implementierungsprache: *Scala* (JVM, objektorientiert und funktional)
  - Frontend: Parser-Kombinatoren
  - Backend / Laufzeitumgebung: nebenläufiger Datenfluss-Simulator
- Ziel: Evaluierung der Skalierbarkeit
- Motivation: Datenfluss-Semantik $\approx$ Actormodell
  - Semantik: Programm = Datenfluss-Netzwerk
  - Vorwärts: Data-Driven, Push
  - Rückwerts: Demand-Driven, Pull

Beispiel: Hamming Sequenz als Datenfluss-Diagramm



$$\text{ham} = 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, \ldots = [2^i 3^j 5^k \mid \forall ijk]$$

$$\text{ham} \, x \, 2 = 2, 4, 6, 8, 10, 12, 16, \ldots$$

$$\text{ham} \, x \, 3 = 3, 6, 9, 12, 15, 18, 24, \ldots$$

$$\text{ham} \, x \, 5 = 5, 10, 15, 20, 25, 30, 40, \ldots$$

Implementierung der simLucid Laufzeitumgebung

- Ein Actor pro Operator ($+, -, \times$, *fby*, ...)
- Jeder Actor kennt seine Kind- und Elternknoten
  - Kindknoten = Operand
- Eingabe: AST des Datenfluss-(Teil-)Programms
- Start: ein IO-Actor der mit dem User interagiert
  - Anfrage von Parametern
  - Ausgabe von Resultaten
- Evaluierung: Demand-Driven
  - IO-Knoten generiert Sequenz von Anfragen an den obersten Ausdrucksknoten für $t_0, t_1, t_2, \ldots$
- Actornetzwerk wird bei Bedarf dynamisch aufgebaut
- Verwaltung eines *Telefonbuchs* für mehrfach benötigte Operanden
  - *Diese Buchhaltung war etwas unbefriedigend*

– Das Actormodell heute / Cloud Deployment –

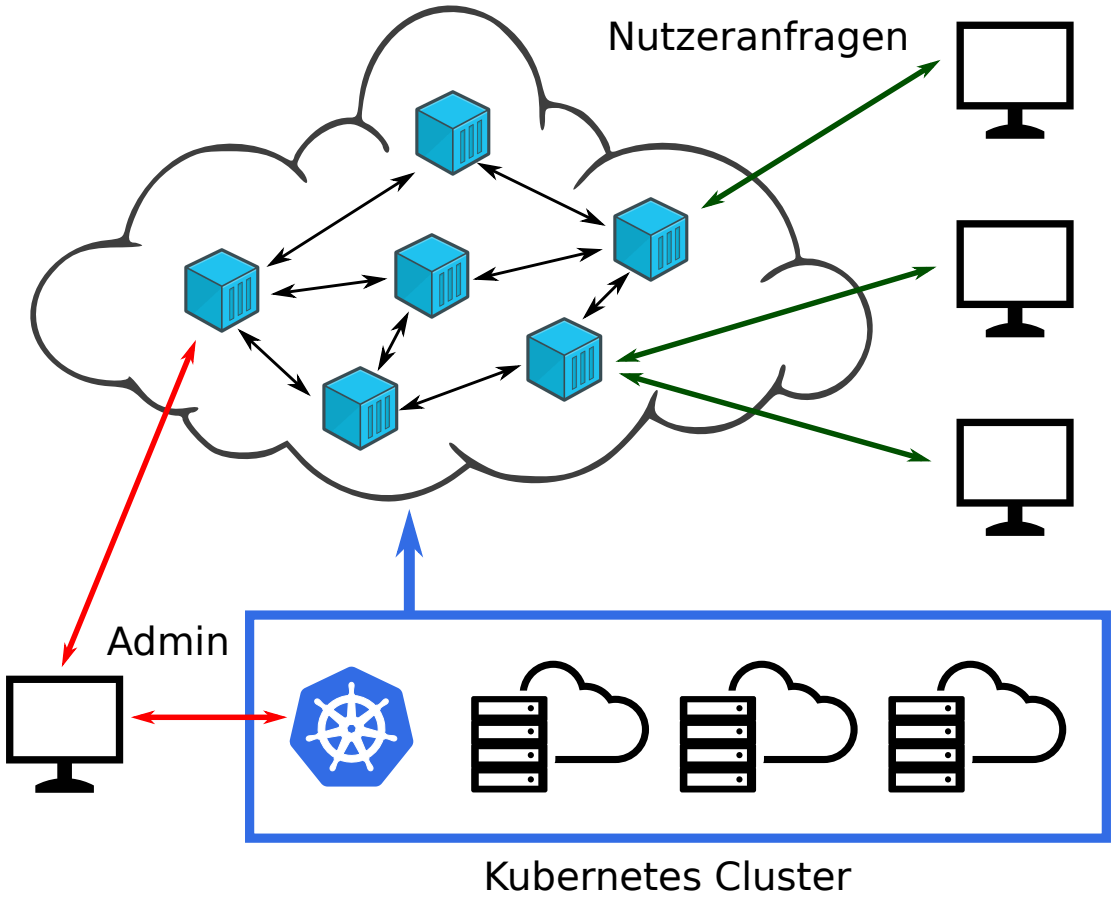
Wo stehen wir heute?

- Umfangreiche Actor-Programmbibliotheken
  - z.B. Akka und Akka Streams[2] von Lightbend für Java und Scala
- Eingebaute Verwaltung von *Actorsystemen*
  - Kein manuelles Telefonbuch mehr nötig
  - Koordinierung von großen Mengen von Actors
  - Kontrollierter Spin-Up und Shutdown der Systeme
- Reibungsloser Übergang von Intra- zu Inter-System-Kommunikation
- Kompatibel mit gängigen Micro-Service Frameworks

[2] `https://akka.io/`

Wo stehen wir heute?

- Big Data: Map-Reduce, Hadoop, Kafka, Spark
- Cloud Deployment: Docker, Kubernetes, Helm, REST



Zusammenfassung

- Actors liefern ein ideales Modell für nebenläufige Prozesse
  - lokale Anwendungen
  - verteilte Cloud-Anwendungen

Zusammenfassung

- Actors liefern ein ideales Modell für nebenläufige Prozesse
  - lokale Anwendungen
  - verteilte Cloud-Anwendungen
- Simulatoren sind rechenintensiv und auf Nebenläufigkeit angewiesen, ...

Zusammenfassung

- Actors liefern ein ideales Modell für nebenläufige Prozesse
  - lokale Anwendungen
  - verteilte Cloud-Anwendungen
- Simulatoren sind rechenintensiv und auf Nebenläufigkeit angewiesen, . . . allerdings nur während sie auch wirklich laufen

Zusammenfassung

- Actors liefern ein ideales Modell für nebenläufige Prozesse
  - lokale Anwendungen
  - verteilte Cloud-Anwendungen
- Simulatoren sind rechenintensiv und auf Nebenläufigkeit angewiesen, . . . allerdings nur während sie auch wirklich laufen
  - Cloud-Deployments liefern die nötige Elastizität

Vielen Dank