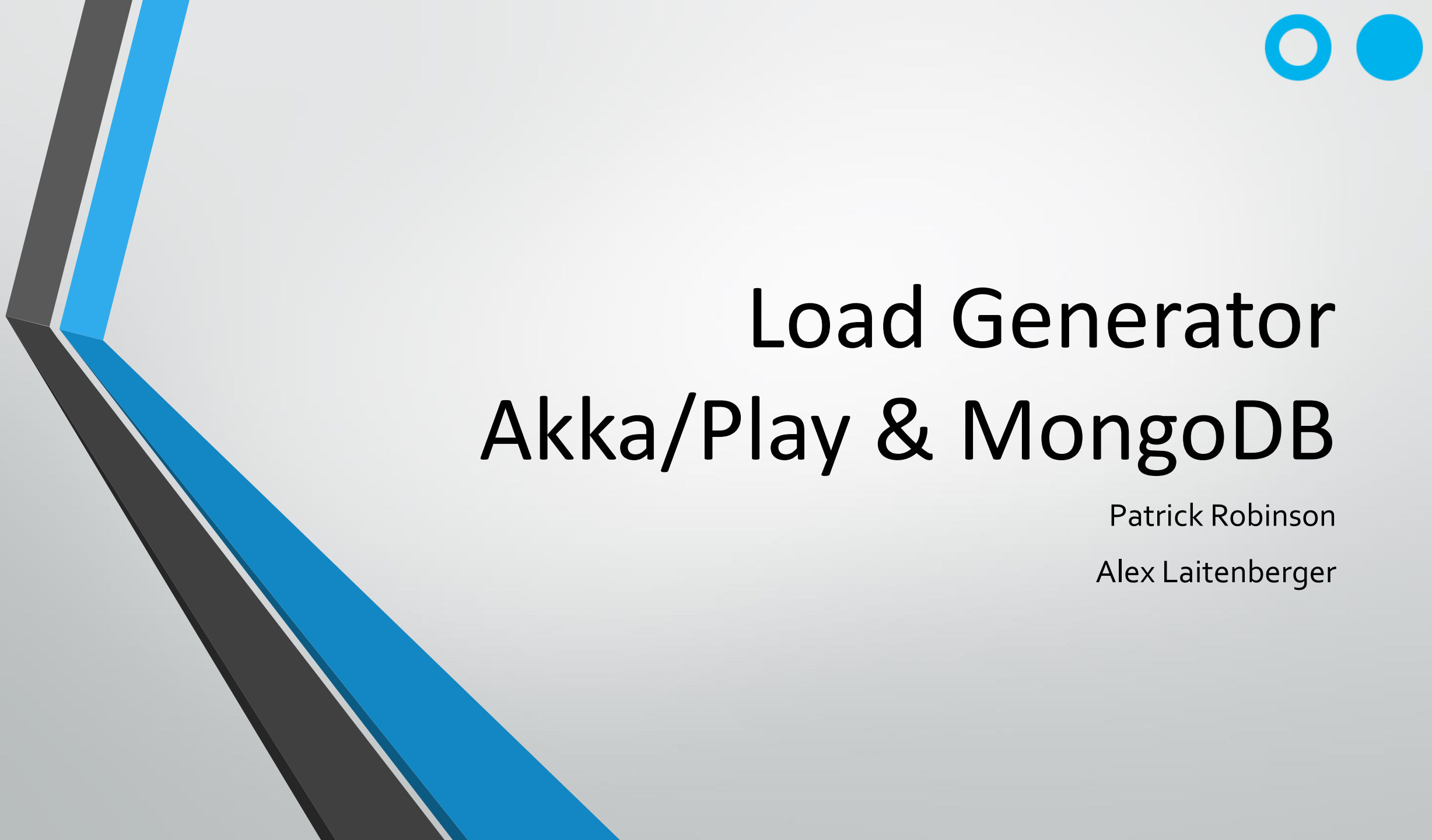


Load Generator Akka/Play & MongoDB

Patrick Robinson

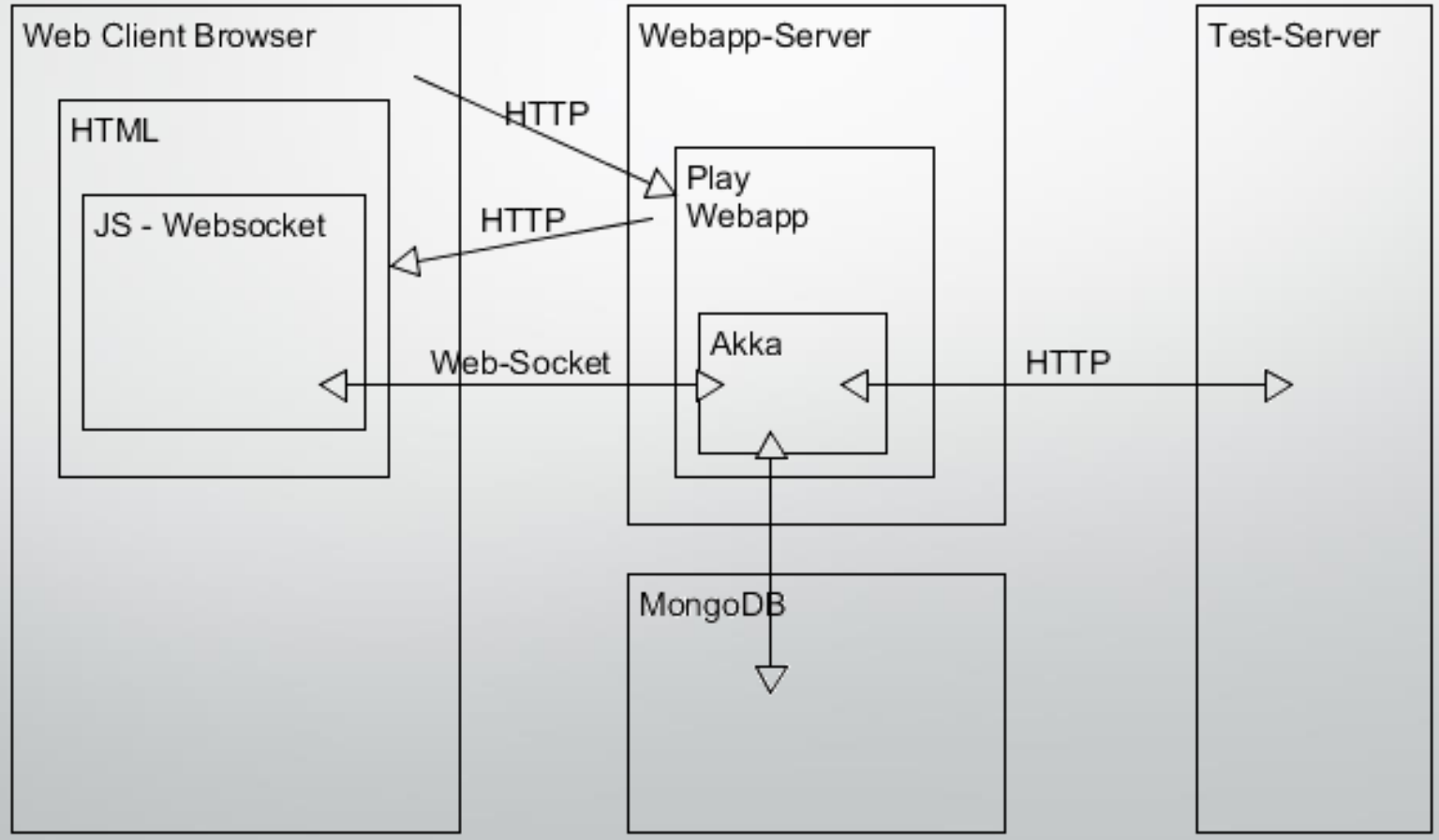
Alex Laitenberger

Inhaltsverzeichnis

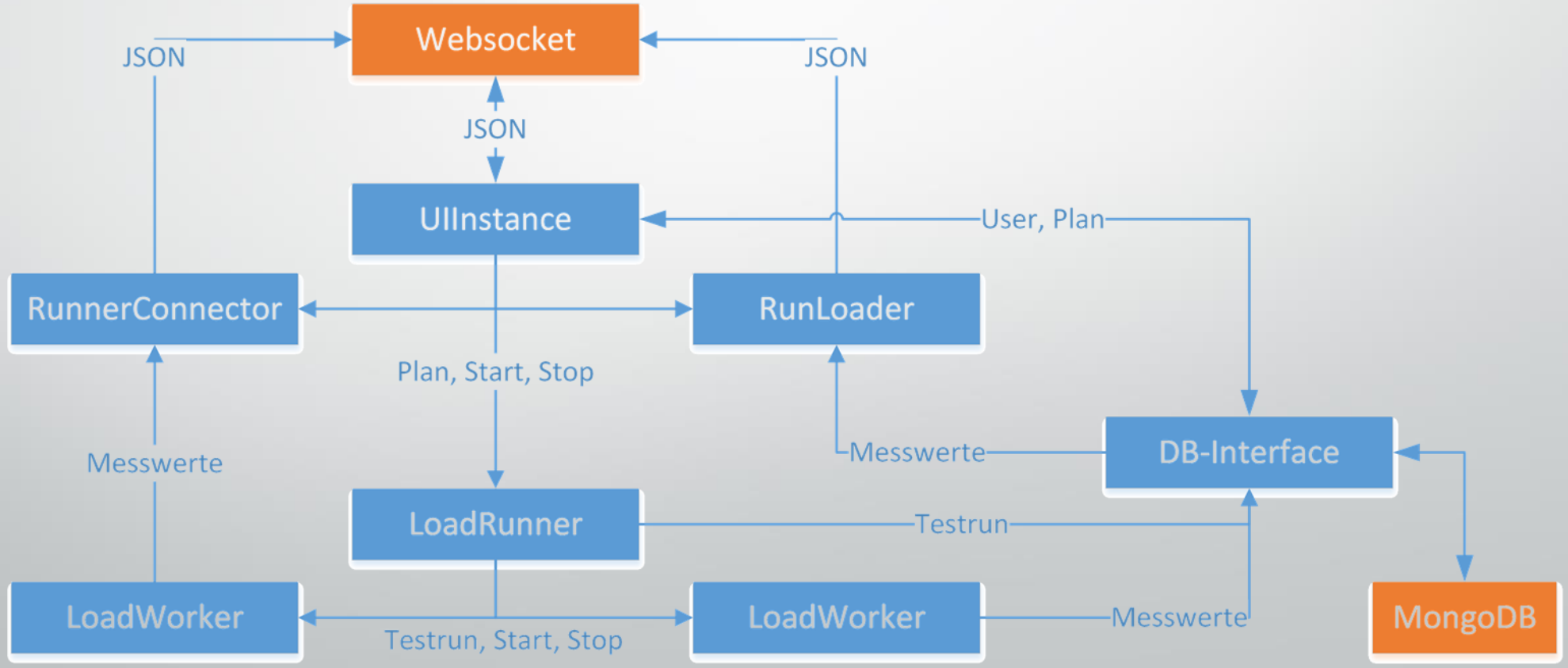
Entwicklungsumgebung

- Github, Issues
- IntelliJ / Eclipse
- Activator, SBT
- Frameworks
  - Play (Routen, WebSocket & JSON)
  - Akka
  - Data Driven Documents D3JS (Graph)
  - MongoDB & Scala Driver

Architektur - Überblick



Aktorsystem



DB

- WriteConcern
  - Unacknowledged: DB hat Request empfangen
  - Acknowledged(Default): DB konnte Request anwenden (im Arbeitsspeicher)
  - Journaled: Änderungen sind auch im Journal (nach Ausfall wiederherstellbar)
- User: {_id, name, password}, Journaled
- Testplan: {_id, user, path, connectionType, wait, parallelity, numRuns}, Journaled
- Testrun: {_id, testplan, [raws]}, Acknowledged
- raw: testrun.raws.push({start, end, iter}), Unacknowledged

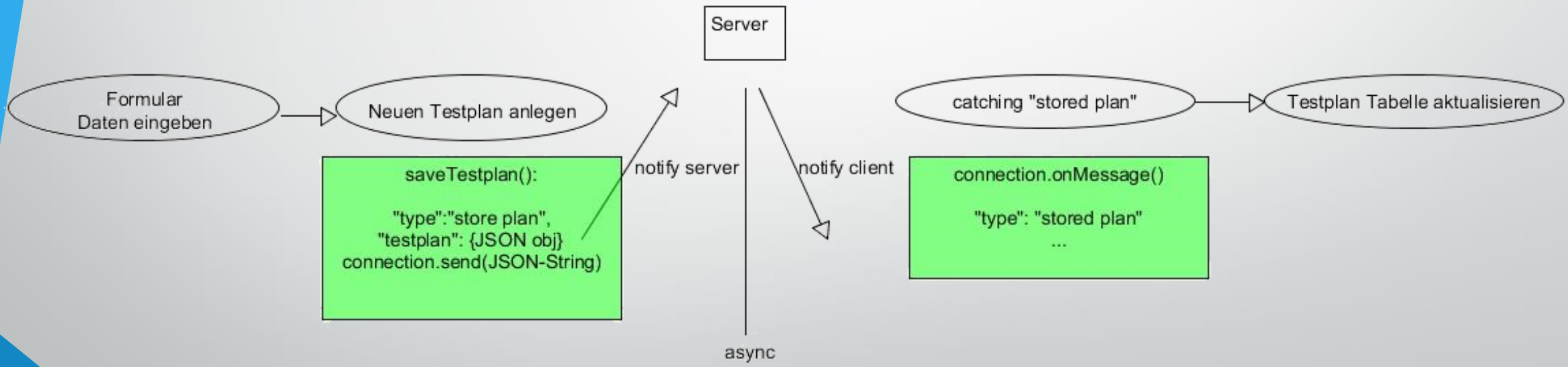
Futures & Promises

- Scalas eigenes Framework für einfache asynchrone Ausführung
  - Future {*code*}: *code* wird asynchron ausgeführt, Rückgabe: Promise
  - Promise: „Versprechen“, dass es zu einem späteren Zeitpunkt Daten haben wird
  - Await
- Nutzung:
  - z.B. Testrun: Future {*get Testplan*}
    - Langwieriges Laden des Testplans zu einem Testrun im Hintergrund
    - Testrun kann schon weiterverarbeitet und –versendet werden
    - Zugriff mit await, erst wenn tatsächlich auf den Testplan zugegriffen werden muss

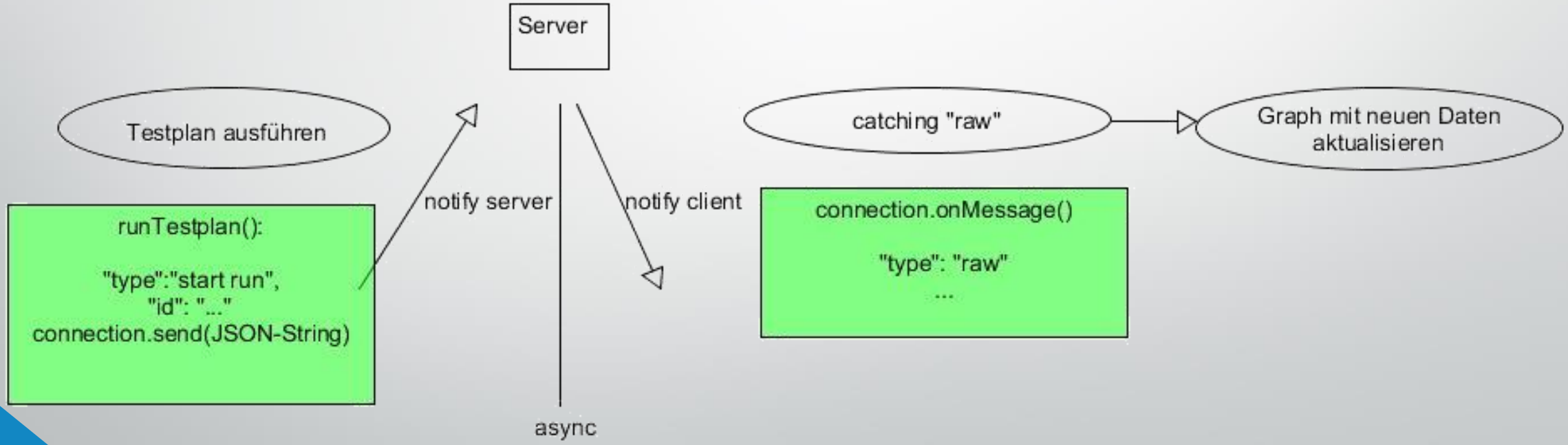
User Interface

- One-Pager
- Websocket Verbindung zum Aktor „UI-Instance“ über eingebettetes JavaScript
- Aktualisiert Inhalte „sofort“ auf Server-push über den Websocket ohne Request-Response (Formular Daten, Testplan speichern, Testrun → Graph)
- D3Js Graph – fügt sofort neue Testergebnisse vom Server als neues DOM-Element in den Graphen ein und passt dynamisch Skalar und Größe der Datenpunkte an

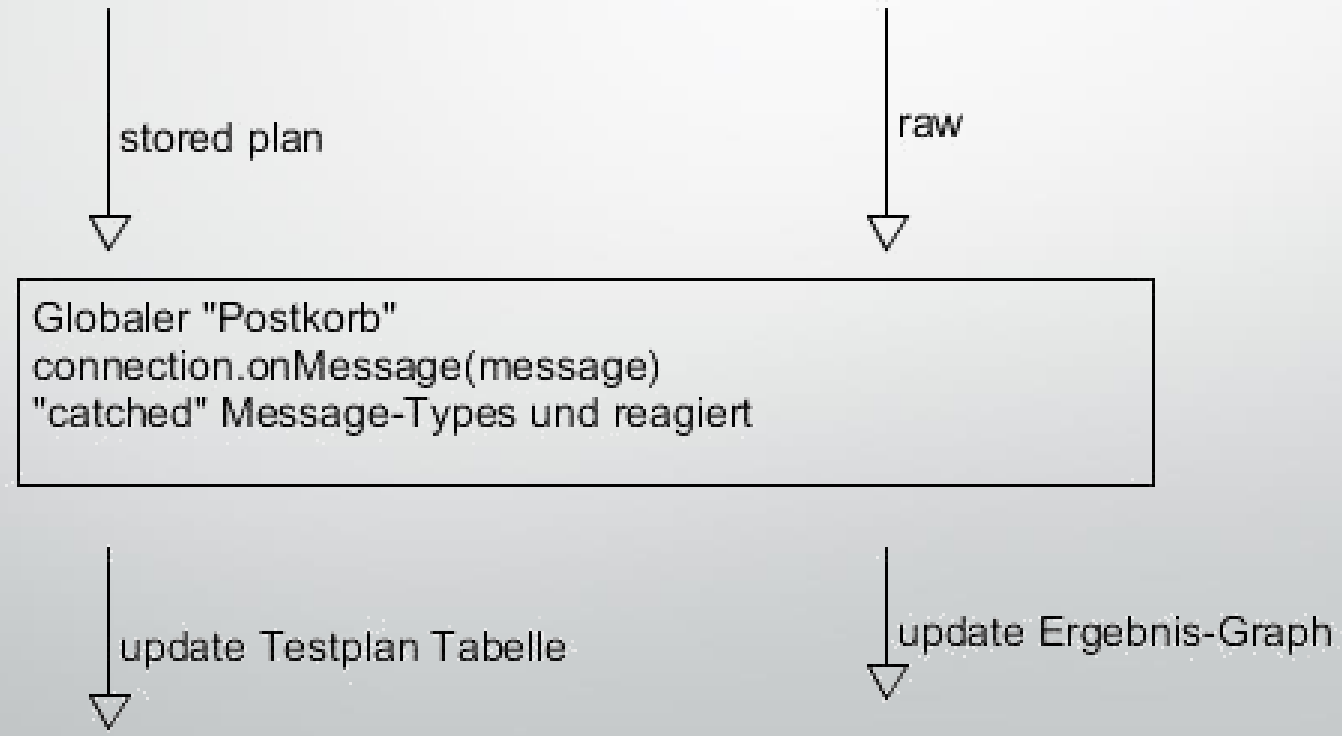
Client: Prozess Testplan anlegen



Client: Prozess Testplan ausführen



Client: Konzept Reaktivität



Graph – D3JS: Data Driven Documents

- JavaScript library: manipuliert automatisch Dokumente aus (neuen) Daten per „Data Join“
- Basiert auf HTML, SVG and CSS
- Vereinfachte Form der DOM Manipulation über Selektoren (d3.selectAll(“p“))
- „Chaining Methods“ auf Selektionen vereinfacht Code
- „Scaling to Fit“ lässt Graphen automatisch abhängig von Daten skalieren
- Bietet Interaktion und Glatte Übergänge v. Chart-Updates
- Daten-Updates
  - Datenbasis: Array (z.B. data = [1, 5, 9];)
  - Data Join mit Selektion: bar.data(data);
  - Neue graph. Elemente: datapoints.enter().append('circle'); //iteriert automatisch

Tests

- Allgemein
  - Separate Test-DB
  - Ursprünglich Activator JUnitklassen, ersetzt durch separaten Testsocket
- LoadWorker: kurzer Test
- DB: Simulation des Aktorsystems und Überprüfung der Antworten & Queries
- UIInstance & Aktorsystem: Simulation der Interaktion durch den WebSocket
- Graph: Generierung von Daten in JS, statt aus Aktorsystem

Reflexion - Designentscheidungen

- Vollständiger Trace
  - Sehr aufwendig
  - Testlauf lässt sich später exakt laden & abspielen
  - Alternativen: Stichproben, teilweise aggregierte Daten
- Data Driven Documents – Dynamisch aktualisierender Graph
  - Zusammen mit vollständigem Trace sehr aufwendig
- Tests durch Simulation
  - Tests selbst sehr fehleranfällig
  - Activator-JUnit-Tests instabil

Reflexion - Frameworks

- D3JS
  - Sehr interessant & flexibel
  - Performancehungrig
- Play (insbesondere JSON)
  - Prinzipiell gut, aber teilweise nur mit Scala gut benutzbar
- Scala
  - Hätten wir von Anfang an nutzen sollen
  - Futures/Promises: Sehr intuitiv & nützlich

Reflexion - Frameworks

- Akka
  - Eher Multiagent/RMI/async als reaktiv
    - Explizites Ziel von Nachrichten
  - Kontextverlust, keine spezifische Antwort
  - Bei sehr hoher Anzahl von Nachrichten: nicht intelligent
- MongoDB
  - ID-Generierung an beliebiger Stelle
  - Nicht normalisierte Datenstruktur

Demo