JAVACREAM

Training
Consulting
Projectmanagement

Neo4J

Vorstellungsrunde

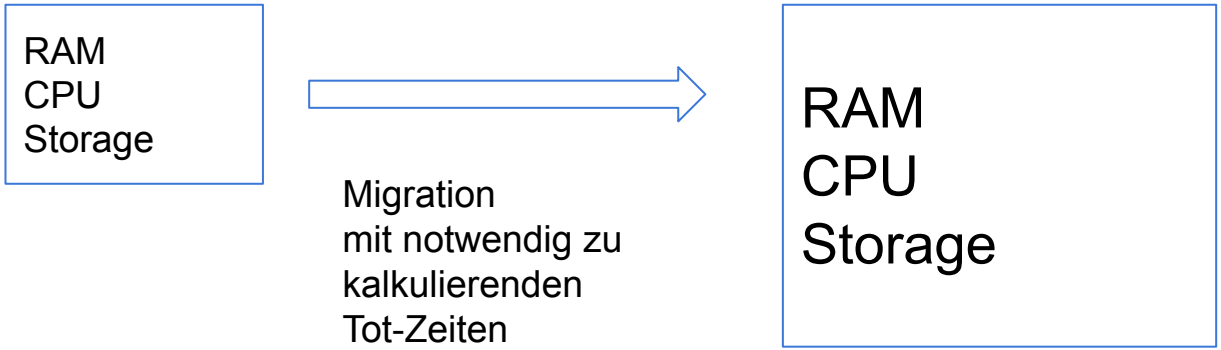
JAVACREAM
Training
Consulting
Projectmanagement

- Name, Rolle im Unternehmen
- Konkrete Problemstellung
- Themenbezogene Vorkenntnisse
- Konkrete individuelle Zielsetzung

Ausgangssituation

Ursprüngliche Problemstellung

- Big&Fast Data
  - 2006: Die Grenze der Ablage von Daten in relationalen Datenbanksystemen war erreicht
    - Storage kann nicht beliebig erhöht werden, weil relationale DBMS vertikal skalieren



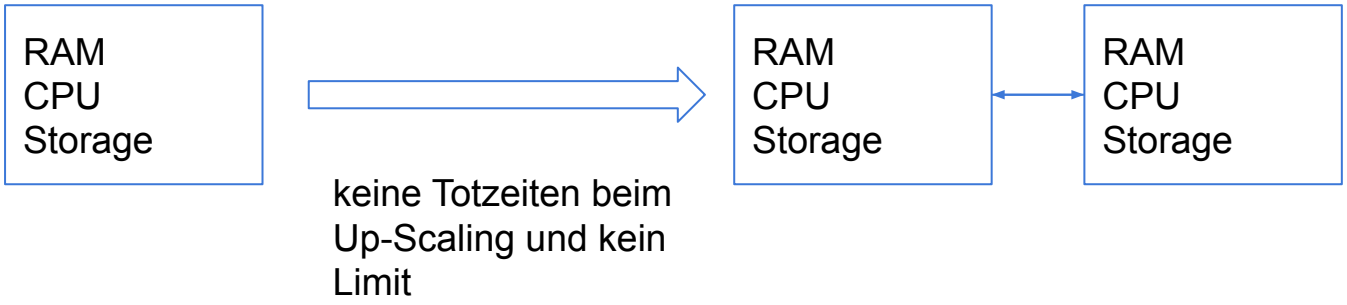
    - Daten-Analyse übersteigt durch die Komplexität der Daten-Zusammenhänge CPU- und RAM-Limits

NoSQL-Bewegung

- NoSQL als Begriff ist eher unglücklich gewählt
  - Besser: NoRelational
  - “No” ist nicht eine Ablehnung sondern eine Abkürzung für “not only”
- Bei der Umsetzung eines Entity-Modells sollen auch alternative Modelle gleichberechtigt in Erwägung gezogen werden
- https://java.integrata-cegos.de/nosql-eine-einfuehrung/

Konkrete Ideen

- “Wie modelliere ich meine Datenhaltung, um diese in einem horizontal skalierenden System halten zu können?”



- **Key-Value-Store**
  - select value from store where key = aKey
- **Column-oriented Databases**
  - Das sind die eigentlichen Big Data-Datenbanken

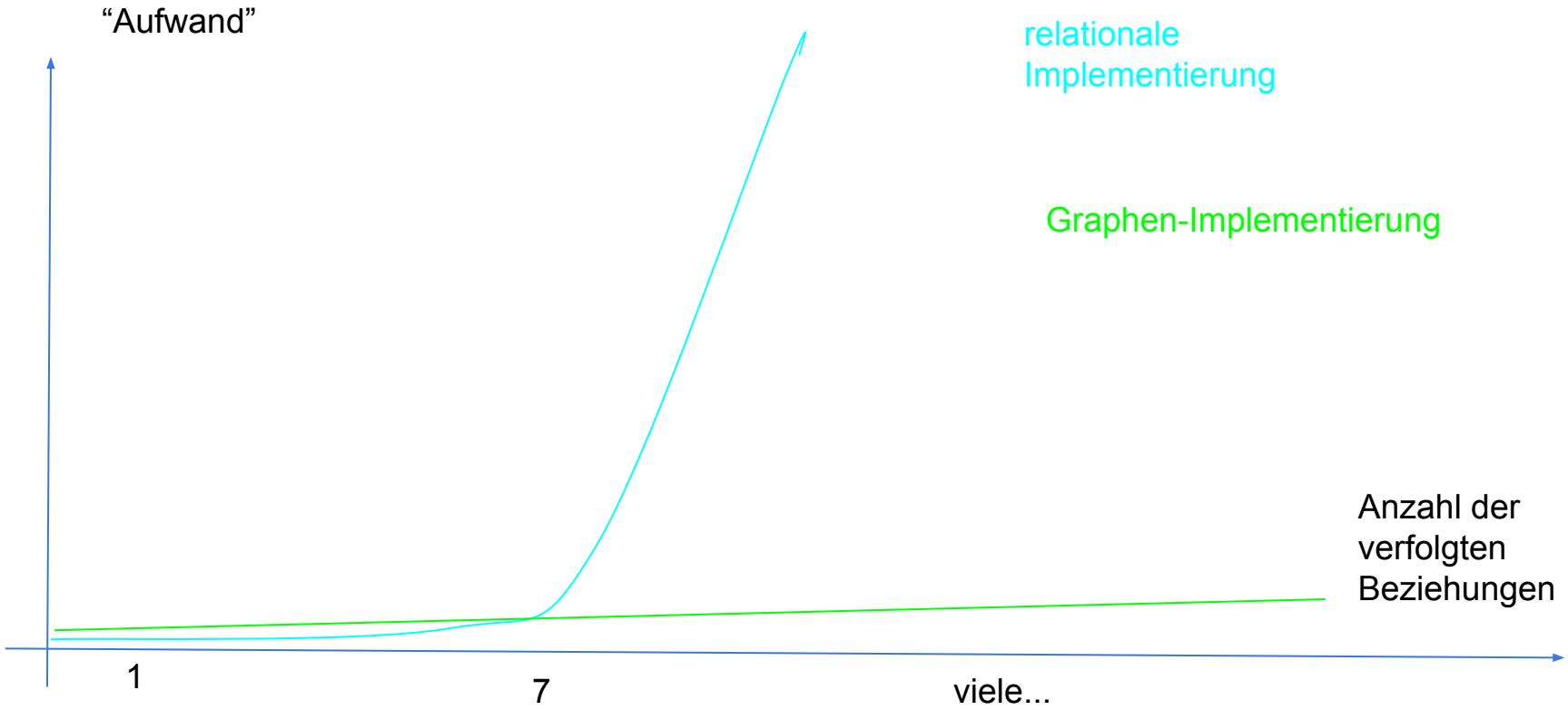
Konkrete Ideen

- “Wie können komplexe Analysen formuliert und effizient durchgeführt werden?”
  - Problematik des “Join”: Relationale Datenbank-Systeme können nur eine beschränkte Anzahl von Joins durchgeführen (etwa 7)
- Lösung
  - **Dokumenten-orientierten Datenbanken**
    - Dokumente enthalten alle zugehörigen Daten “en block” und Beziehungen zwischen Dokumenten sind Links
      - Die Rolle der Client-Anwendung ist hier deutlich höher als bei einem relationalen Client, der seine Daten komplett aufbereitet von der Datenbank bekommt
    - Neue Beziehungen können problemlos eingeführt werden
      -

Konkrete Idee

- Beziehungen und deren Analyse sollen von einem Datenbank-System durchgeführt werden
  - Vorsicht: Very Big Data ist hier definitiv nicht der Fokus
- Ein Knoten, “eine Node” enthält alle Daten, die für ihn relevant sind
  - Damit entspricht ein Knoten einem einzelnen Dokument
- Beziehungen, “eine Relation” sind vollständige Dokumente, die jedoch eine Quelle und ein Target besitzen
  - Eine Relation verbindet zwei Knoten miteinander
    - Relationen können nicht auf andere Relationen verbinden
- Mit Nodes und Relations kann die interne Datenhaltung vollkommen anders implementiert und optimiert werden
  - Graphen-orientierte Datenbank

Vergleich: Auflösen von Beziehungen



Warum sind die relationalen “so schlecht”

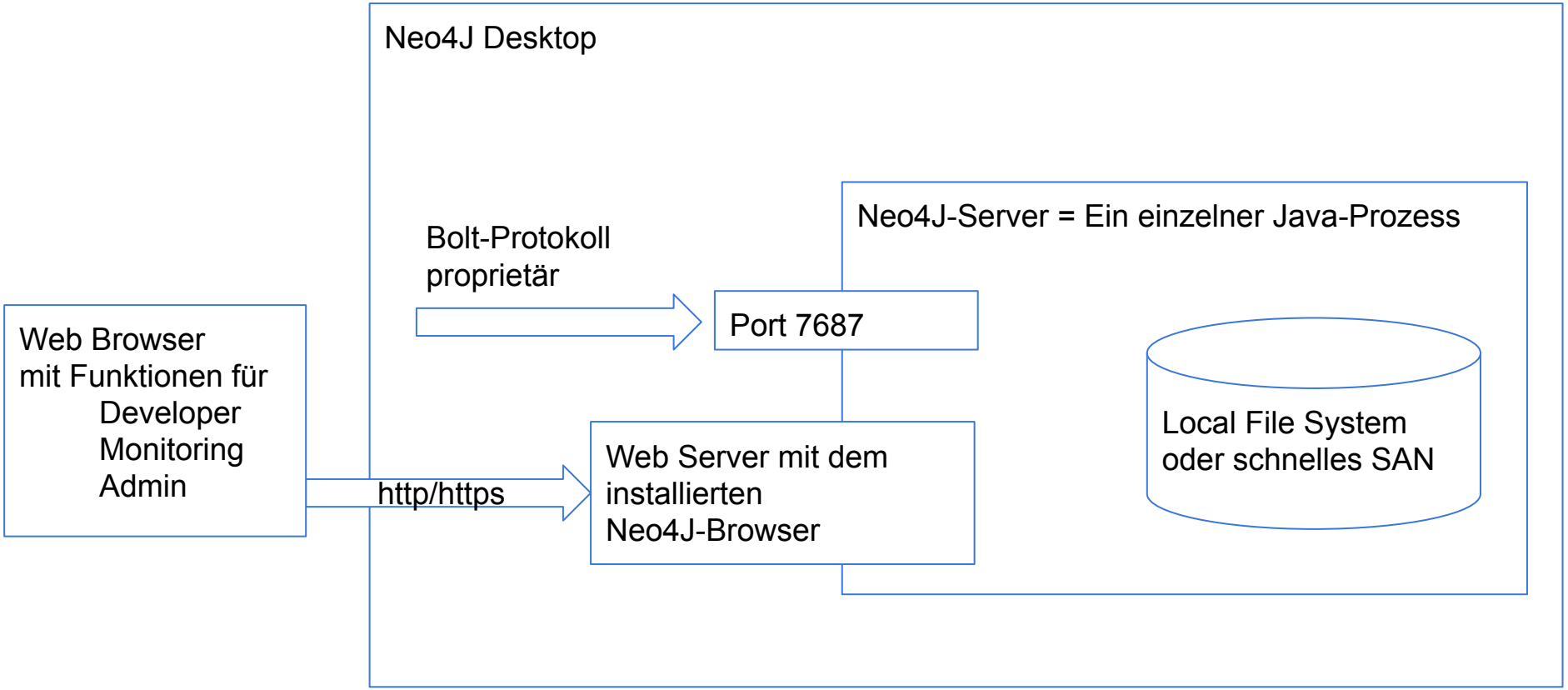
- Sie sind nicht schlecht, sondern bieten ganz andere Features
  - Konsistenz der Datenhaltung
    - Daten-Normalisierung
  - Statisches Schema inklusive Constraints
    - Inklusive Beziehungen zwischen Datensätzen
- Aktuelle Entwicklungen und Trends führen aktuell zu sehr interessanten Umsetzungen von Graphen-orientierten Ideen in relationalen Datenbank-Systemen

Unsere Trainings-Umgebung

- Benutzung der Integrata-Cegos-Rechner
  - Zugriff via RdWeb auf einen fertig eingerichteten Rechner
    - Installiert und eingerichtet ist eine Neo4J Desktop
- Eine lokale Installation des Neo4J Desktops
- Installtion eines Neo4J-Servers auf http://h2908727.stratoserver.net:7474/browser/
  - Die Präsentation erfolgt auf dieser Umgebung
    - neo4j
    - javacream

Neo4J: First Contact

Systemlandschaft



Exkurs: Neo4J-Skalierung

- Laut Katalog
  - “Ein horizontal skalierendes System”
  - VORSICHT: Eine Datenhaltung in mehreren Neo4Js paralle hat deutlichen Einfluss auf die Performance der Abfragen
- Effizientes Neo4J ist ein Server-Prozess
  - Ausfallsicherheit eher klassisch mit Active/Passive
  - Daten-Skalierung eher durch eine Partitionierung der Daten

Neo4J-“Feeling” ist sehr an relationale Datenbanken angelehnt

- Organisation der Datenbanken in einer Datenbank-Server-Instanz
  - User und Rollen regeln den Zugriff auf die Datenbanken
    - Zu den Rollen
      - PUBLIC mit lesende Zugriff auf die Default-Datenbank
      - reader -> editor -> publisher -> architect -> admin
- Abfragesprache “Cypher” ist SQL-orientiert
  - Absolute Ausnahme im Vergleich zu anderen Graphen-orientierten Systemen
    - Deren Abfragesprache sind Script-Programme, die auf der OOP Collection-Verarbeitung gründen
- Tooling
  - Daten-Export, -Import, describe/explain für die Beurteilung der Abfragen, Indizes, über Constraints Schema-Definitionen

Zur Konsole

- Zentraler Bestandteil des Neo4J-Browsers
  - Syntax-Highlighting
  - Autovervollständigung
- Ausgaben erfolgen im JSON-Format
  - Visualisierung in Neo4J-Browser
    - Tabelle
    - Plain Tabelle
    - “Code” = Raw JSON-Format

Erste Konsolen-Befehle

- Datenbanken
  - SHOW DATABASES
  - SHOW DATABASE <db-name>
  - CREATE DATABASE <db-name>
  - DROP DATABASE <db-name>
- User und Roles
  - create user <my_user> set password <pwd> change not required
  - create role...
  - grant ROLE publishers to <my_user>

Anlegen von Daten in Neo4J

- Primär sind Daten in Neo4J “Dokumente”
  - Ein Dokument besteht aus
    - Attribute-Value-Paaren
    - im JSON-Format
  - Typisierung
    - Value kann sein
      - Zeichenkette, “”, ‘’
      - Numerische Werte: 3, 42, 3.42
      - Zustände: true, false
      - eine Liste [“A”, “B”, “C”]
      - {“key”:value}
- Node-Dokumente haben noch zusätzlich ein “Label”
  - Best Practice: Verwenden Sie Label als eine Typisierung ihrer Nodes
- Relations-Dokumente haben einen verpflichtendes Type
  - Source-Node werden über die Relation mit einem Target-Node gerichtet verbunden

Cypher-Befehl zum Anlegen eines Dokuments

- CREATE
  - Node
    - ()
  - Relation
    - <from_direction> [] <to_direction>

Cypher für eine erste Abfrage

- WICHTIG: Elemente eines Graphen werden nicht selektiert, sondern an Hand eines Muster-Ausdruckes “gematched”
- MATCH
  - Node
    - ()
  - Relation [] + Directions
-

Exkurs: Cypher-Sprache

- Angelehnt an SQL
- Aber eigentlich eine Skript-Sprache mit
  - Variablen
    - Scope, eine Lebensdauer
  - Return-Anweisung
- Als erstes Beispiel ein “matche einen beliebigen Knoten”
  - MATCH (result) //result = MATCH ()
  - return result

Detail: Selektieren eines Knoten

- (x)
  - Selektiere ohne Kriterium, Ergebnis steht im Script unter dem Namen ‘x’ zur Verfügung
- (x :Label)
  - Selektion nach Label
- (x {Candidate})
  - Selektiert nach den Attributen des Candidate-Objekts
- Kombinationen möglich

Details zur Ausgabe in einer Kachel

- Umschalter
  - Graph
  - Table
  - Plain Table
  - Code
- Iteration über die Ergebnismenge erfolgt automatisch
- Genaue Darstellung (was ist die farbe der Knoten? Was wird als Info im Knoten dargestellt) wird von der Kachel automatisch bestimmt

ToDo

- Legen Sie sich eine eigene Test-Datenbank an
  - z.B. training
- In dieser Test-Datenbank anlegen von Knoten
- Labels: Person (lastname, firstname, height), Address (city, postalCode, street)
- Lernziel:
  - Umgang mit der Konsole
  - Anlegen, Listen von Knoten
  - Umgang mit der Ergebnis-Kachel
- Hinweis:
  - Es gibt auch eine where-Klausel -> später
  - Ebenso: Relationen