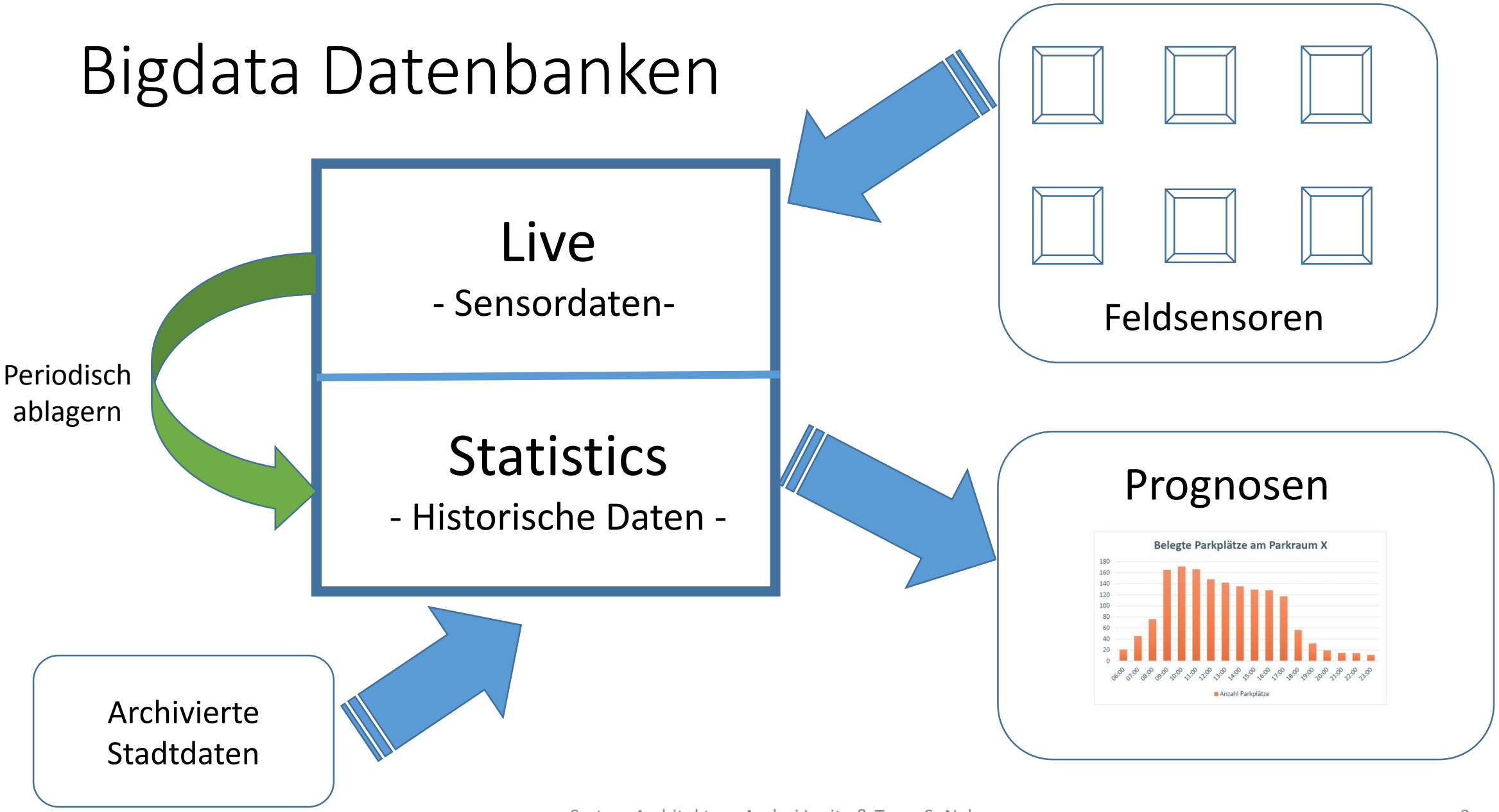
System Architektur v0.0.2

06.10.16 – Andrei Ionita & Team SoNah

Übersicht

1. Bigdata Datenbanken
   - Live Datenbank
   - Statistics Datenbank
   - Sensordaten simulieren
2. statische Datenbank
   - Open Street Map Daten importieren
3. Parken Abfragen

Bigdata Datenbanken

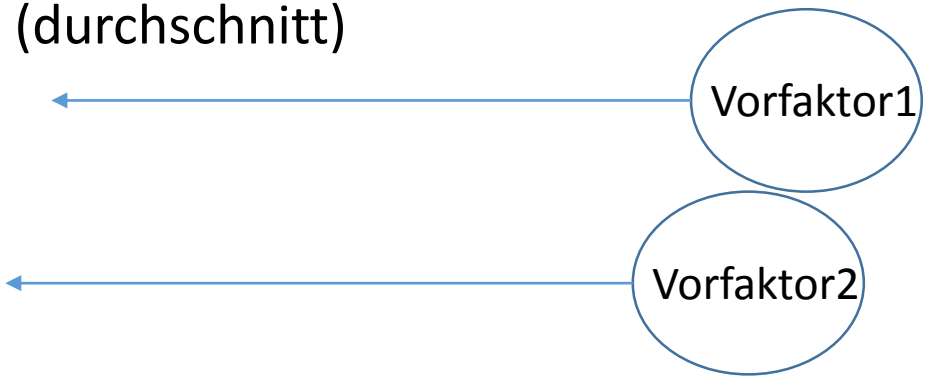


Bigdata Datenbank (2) – Live

- Sensormeldungen bestehen aus:
  - Sensor id
  - Timestamp
  - Parkarea
  - isFree – ist jetzt der Platz frei oder belegt? (TRUE oder FALSE)

Bigdata Datenbank (3) – Statistics

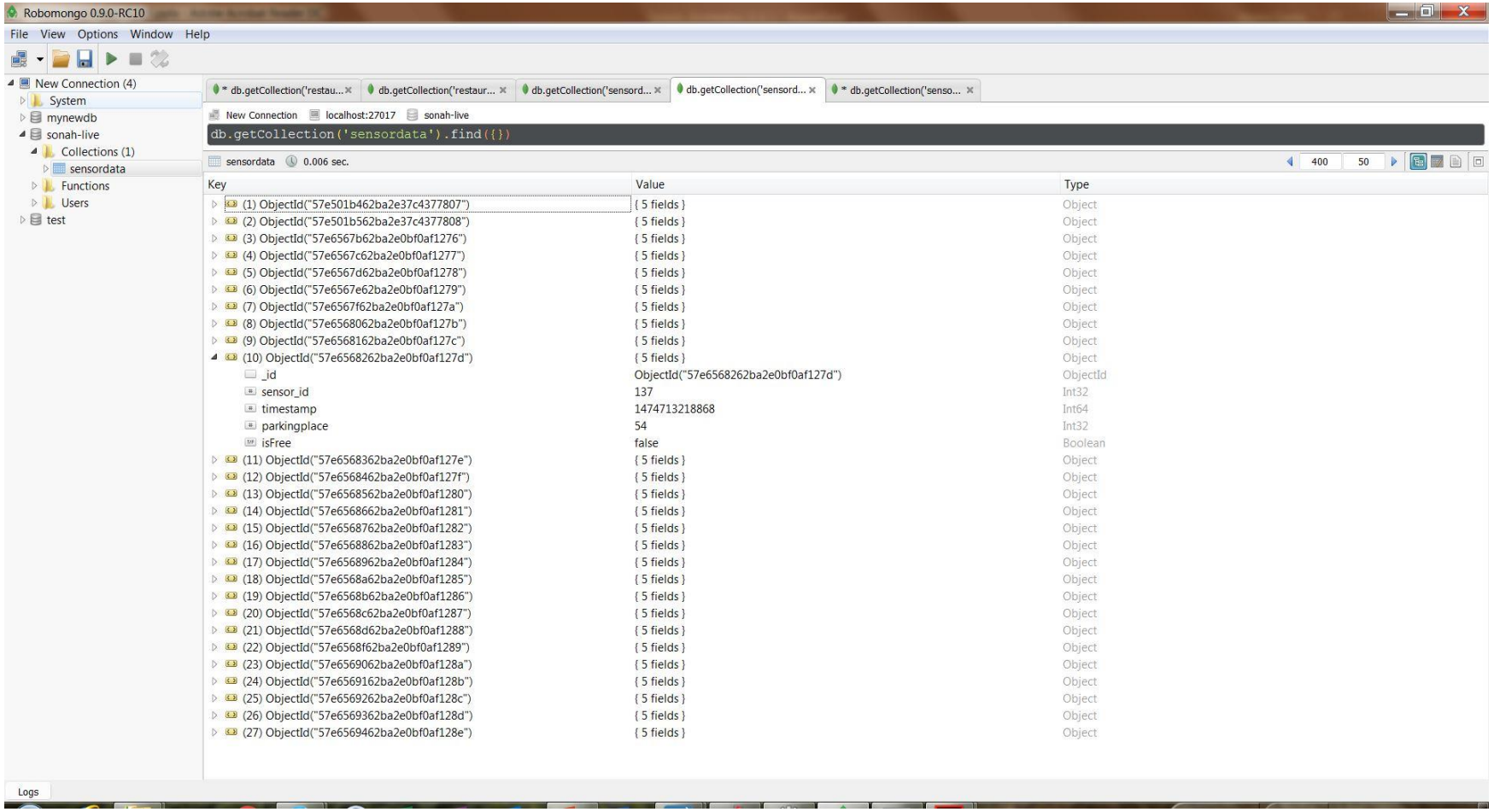
- Einträge bestehen aus
  - Parkarea ID
  - Tag der Woche
  - isHoliday
  - Uhrzeit
  - Anzahl der freien Plätze/AfP (durchschnitt)
  - Anzahl der belegten Plätze/AbP (durchschnitt)
  - Anzahl Meldungen Sensortyp1
  - AfP und AbP für Sensortyp1
  - Anzahl Meldungen Sensortyp2
  - AfP und AbP für Sensortyp2
  - …



Sensordaten simulieren

- (Bis genug echte Daten verfügbar sind)
- Je nach Speicherart unterscheiden sich folgende Strategien:
  - 1 Eintrag / Event (aktuelle Wahl)
  - Pro Sekunde
    - Falls mehrfache Events pro Sekunde
    - Ein Eintrag hat Zehntelsekunde Untereinträge (0..9)
    - Vorteil: weniger Inserts, spart Schreibenzeit
  - Pro Minute
    - Sekunde-Events 0..59 sind zusammen gespeichert

Mongodb - Screenshot



Push Aktualisierungen

- Die freien Parkplätze per Parkareas aggregieren und periodisch aktualisieren

- Map/Reduce
  - Filter: Einträge von zB letzter Minute
  - Map: Parkarea Id + isFree Wert (+1 / - 1)
  - Reduce: Delta berechnen = summiert die (+1 / -1) Werte pro Parkarea

- Die Daten in eine separate Tabelle abspeichern

Statische Datenbank – Schema (1)

- Informationen über die Sensoren

```
CREATE TABLE `sensors` (
        `SensorID` bigint(20) NOT NULL AUTO_INCREMENT,
        `Latitude` double NOT NULL,
        `Longitude` double NOT NULL,
        `Altitude` double NOT NULL DEFAULT '0',
        `ParkingAreaID` bigint(20) COMMENT 'Parking area to which it belongs',
        `Added` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
        `Type` varchar(30) COMMENT 'Type of sensor (optical, etc.)',
        PRIMARY KEY (`SensorID`)
);
```

Statische Datenbank – Schema (2)

- Informationen über die Parkräume

```
CREATE TABLE `parkingareas` (
        `ParkingAreaID` bigint(20) NOT NULL AUTO_INCREMENT,
        `ParkingAreaName` varchar(30),
        `AddressHousenumbers` varchar(80),
        `CenterLatitude` double NOT NULL,
        `CenterLongitude` double NOT NULL,
        `Access` varchar(20) COMMENT 'E.g. public, customers, residents, employees',
        `Type` varchar(20) COMMENT 'E.g. surface, multi-storey, underground, rooftop, sheds,
garage_boxes',
        `FeeID` bigint(20) NOT NULL,
        /* time interval when open – missing, mysql has no such inbuilt type*/
        `ParkareaFeaturesIDS` varchar(80) NOT NULL,
        PRIMARY KEY (`ParkingAreaID`)
);
```

Statische Datenbank – Schema (3)

- Informationen über die relevante Parkräume vor Ort bzw. Typen von Gebäude
- Essentiell, Prognose zu gleichartigen Parkräumen späterhin abzuleiten

```
CREATE TABLE `parkingareafeatures` (
        `ParkingAreaFeatureID` bigint(20) NOT NULL AUTO_INCREMENT,
        `ParkingAreaFeatureType` varchar(80), COMMENT 'E.g. school, supermarket, company, etc.',
        `ParkingAreaFeatureSize` varchar(80), COMMENT ,E.g. 1-100, 101- 300, 300 – 1000, etc. ',
        PRIMARY KEY (` ParkingAreaFeatureID `)
);
```

Open Street Map Daten

- Daten werden durch Abfragen an OSM Overpass API erhalten
  - Parkautomaten (`vending = parking_tickets`)
  - Parkräume (`amentiy = parking`)
  - Parkplätze an den Seiten (`parking:lane`)

- Aktuell
  - `sensors` Tabelle wird mit OSM Parkautomaten-Daten befüllt
  - `parkingareas` Tabelle wird mit OSM Parkräume-Daten befüllt
  - sensor : parkingarea Zuordnung??

Parken Abfragen (1)

- Vereinfacht: Finde die besten Parkmöglichkeiten in der Nähe von Position `(lat, lon)`, jetzt!
- Vorgeschlagener Algorithmus:
  - Suche im Umkreis 500m
    - finde die Parkareas, die den aktuellen Standort geometrisch enthalten
    - Berechne Anteil an freien Plätzen für diese Parkareas (stand: aktuell)
    - Erstelle eine Rangliste; E.g. P1: 35% frei, P2: 24% frei, P3: 10% frei
    - Falls nicht genug Parkareas gefunden/die Auslastungsgrad zu hoch ist, dann…
  - Suche im Umkreis 1000m
    - das Gleiche
  - Suche im Umkreis 1500m

Parken Abfragen (2)

- Schleife endet sobald man **drei** Parkareas gefunden hat

- Aktueller Auslastungsgrad der gewählten Parkareas der aggregierten Datenbank abfragen

- User bekommt eine Liste mit verfügbaren Freiplätzen in den nahliegenden Parkräumen; zum Beispiel:
  1. Adalbertsteinweg Parkhaus / 65%
  2. Jülicher Straße 4 – 14 / 40%
  3. Normaluhr Parken / 38%