Labor für Verteilte Systeme

DOPSY group | Labor für Verteilte Systeme Distributed Systems Lab

Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

Gegründet

Prof. Dr. Reinhold Kröger

Leitung

Prof. Dr. Robert Kaiser

→ Echtzeitbetriebssysteme, Mikrokerne und Virtualisierung, Embedded Systems

Prof. Dr. Steffen Reith

→ Automotive Kryptographie, Open Source Crypto Hardware

Marcus Thoss M.Sc.

→ Energiemanagement und Self-X für Embedded Systems, Sensornetze

Mitarbeiter

Kai Beckmann, Alexander Züpke, Fabian Meyer, Andreas Textor

Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

Ortsbasierte Topic-Filterung für die datenzentrierte Middleware sDDS

03. März 2017

Olga Dedi

Design Informatik Medien
Hochschule **RheinMain**

GLIEDERUNG

1. Hintergrund und Problemstellung
2. Anforderungen
3. Architekturdesign
4. Evaluation
5. Fazit und Ausblick

HINTERGRUND UND PROBLEMSTELLUNG

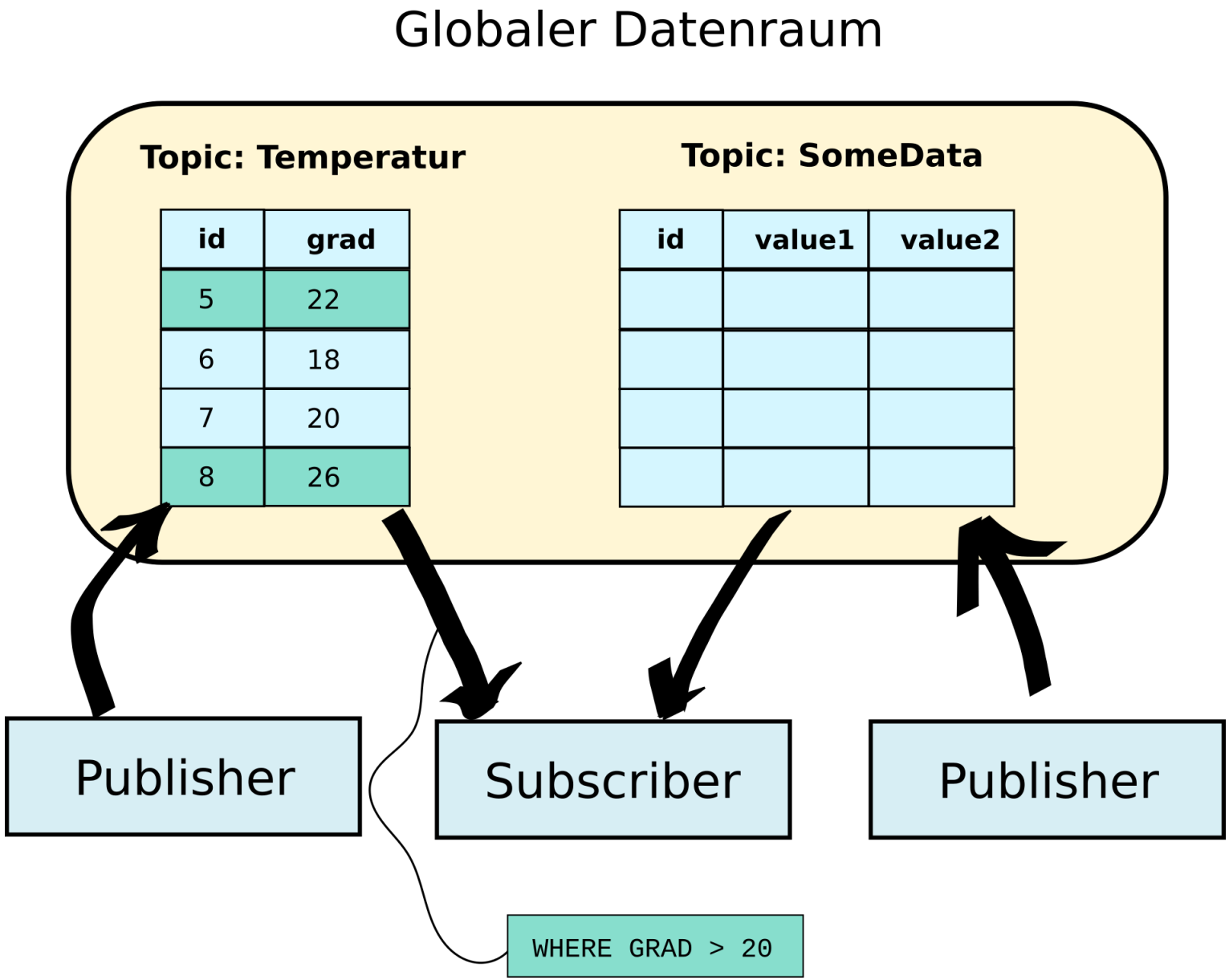
HINTERGRUND UND PROBLEMSTELLUNG

Ortsbasierte Topic-Filterung für die datenzentrierte Middleware sDDS

DATA DISTRUBUTION SERVICE (DDS)

- → Middleware-Standard der Object Management Group (OMG)
- → Publisher-Subscriber-Prinzip
- → Kein Broker
- → Definiert globalen Datenraum
- → Daten als Datenstruktur modelliert und einem *Topic* zugeordent
- → Zuordnung zwischen Publisher und Subscriber über Discovery
- → 22 veschiedene *Quality-of-Service*-Merkmale konfigurieren die Dienstgüte
- → Über QoS-Einstellungen wird Echtzeitfähigkeit ermöglicht
- → Ursprünglicher Einsatz Militär, immer mehr Anwendung in anderen Bereichen

DDS ANWENDUNGSSICHT



DDS MIDDLEWARESICHT

Publisher

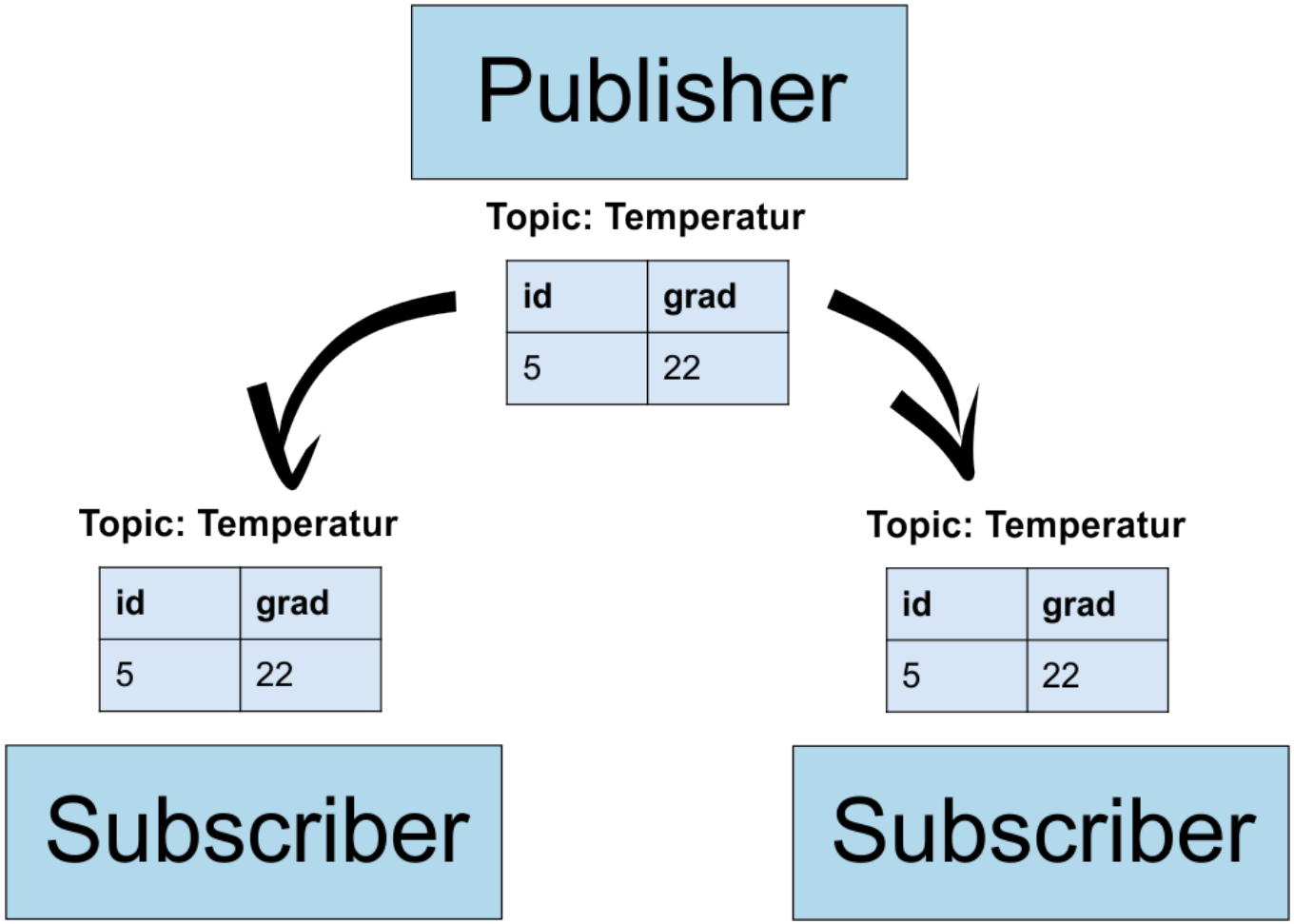
Topic: Temperatur

id	grad
5	22

Subscriber

Subscriber

DDS MIDDLEWARESICHT

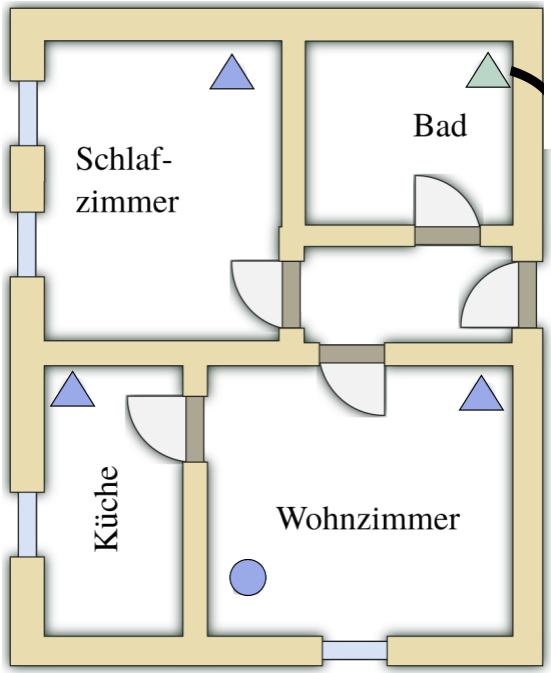


SENSOR DATA DISTRIBUTION SERVICE (SDDS)

- → sDDS wird im Labor für Verteilte Systeme entwickelt
- → Implementiert Teilmenge des DDS-Standards für ressourcenarme Sensorknoten
- → Plattformunabhängig
- → Auf mehrere Betriebssysteme portiert (Linux, RIOT OS, Contiki OS, ...)
- → Modellgetriebener Entwicklungsprozess
- → Funktionen sind in Module aufgebrochen, so dass nur genutzte Funktionen aktiviert bzw. dazugelinkt werden
- → Keine dynamische Speicherverwaltung

SDDS 4 SMART HOME

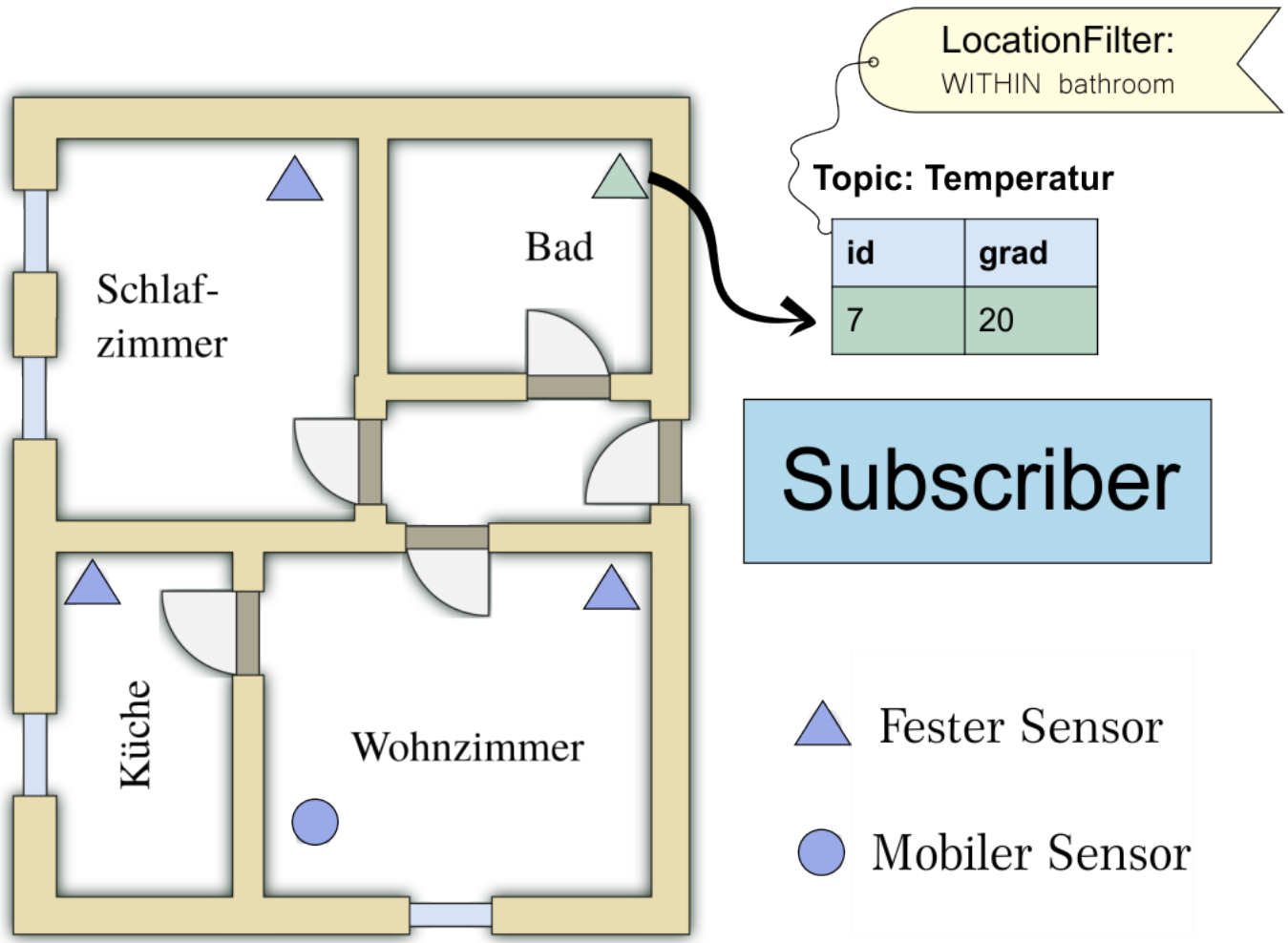
- → Aktuelles Forschungsprojekt in Kooperation zwischen der Hochschule RheinMain, der Hochschule Darmstadt und mehreren Kooperationspartnern.
- → Evaluiert den Einsatz von sDDS im Bereich Smart Home
- → Master-Thesis war Teil des Forschungsprojektes
- → Use Cases legen besonderen Fokus auf Smart Home



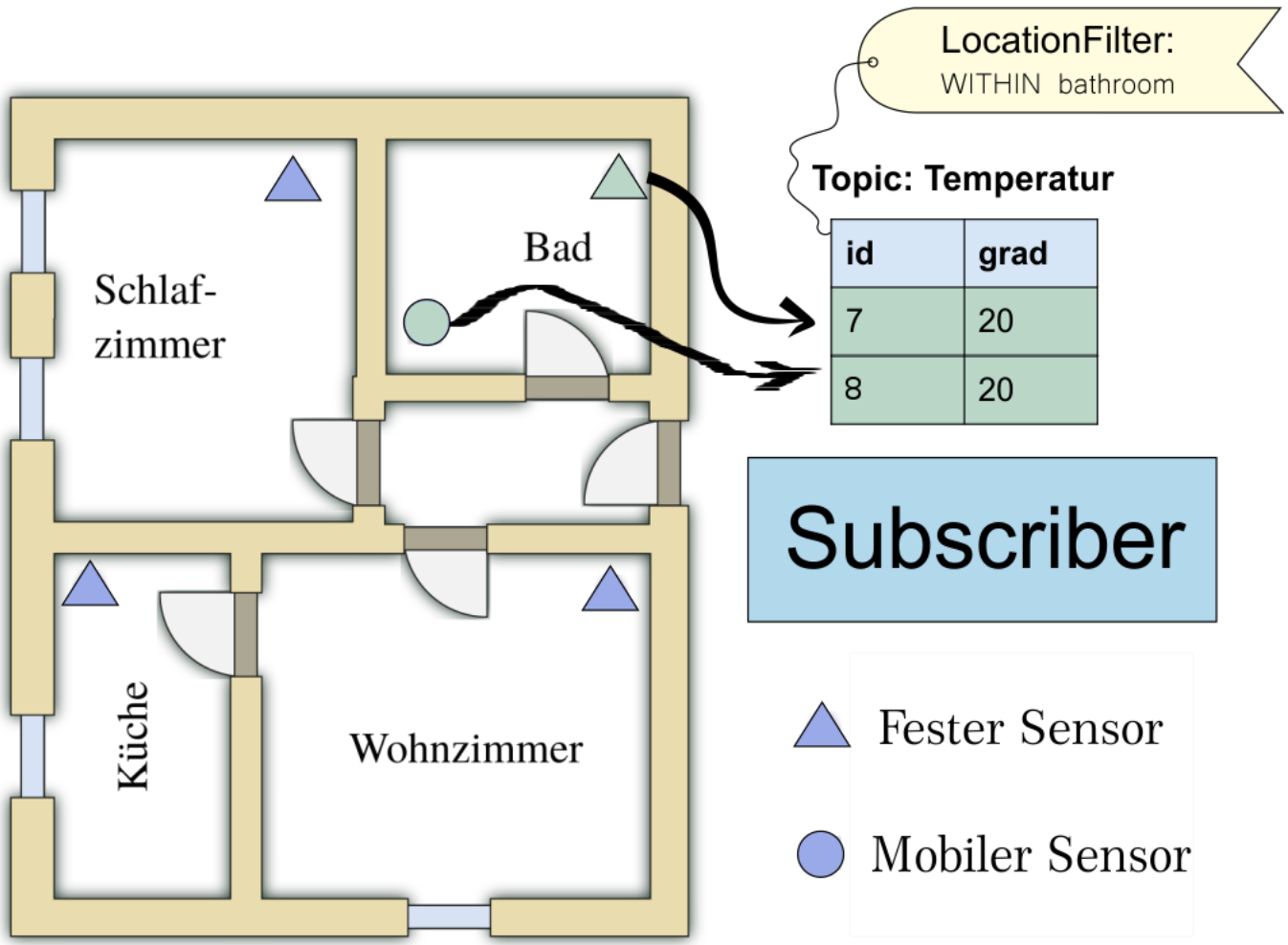
PROBLEMSTELLUNG UND HINTERGRUND

Ortsbasierte Topic-Filterung für die datenzentrierte Middleware sDDS

ORTSBASIERTER FILTER



ORTSBASIERTER FILTER



ANFORDERUNGEN

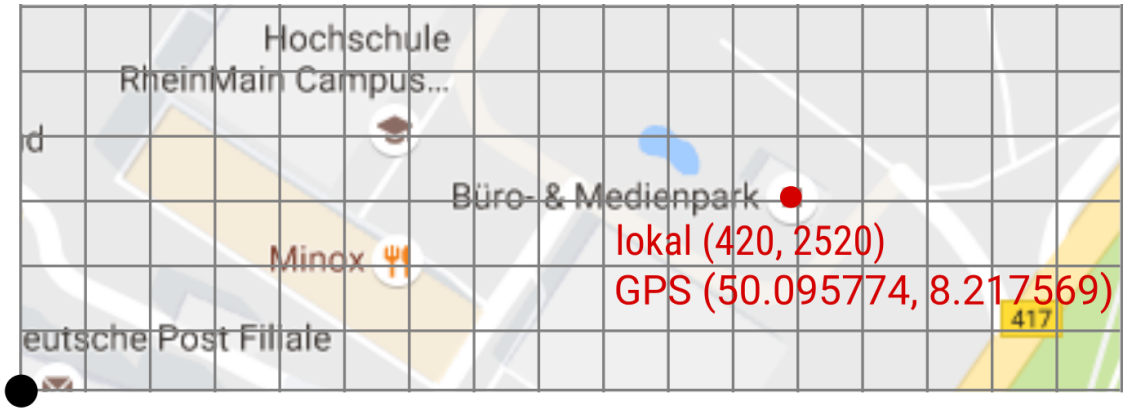
ANFORDERUNGEN

- → Repräsentation der Ortsinformation
- → Verbreitung der Ortsinformation im Netzwerk
- → Spezifikation und Umsetzung eines ortsbasierten Filters
- → API-Kompatibilität zum DDS-Standard
- → Integration in sDDS
- → Ressourcenbeschränkte Plattformen

ARCHITEKTURDESIGN

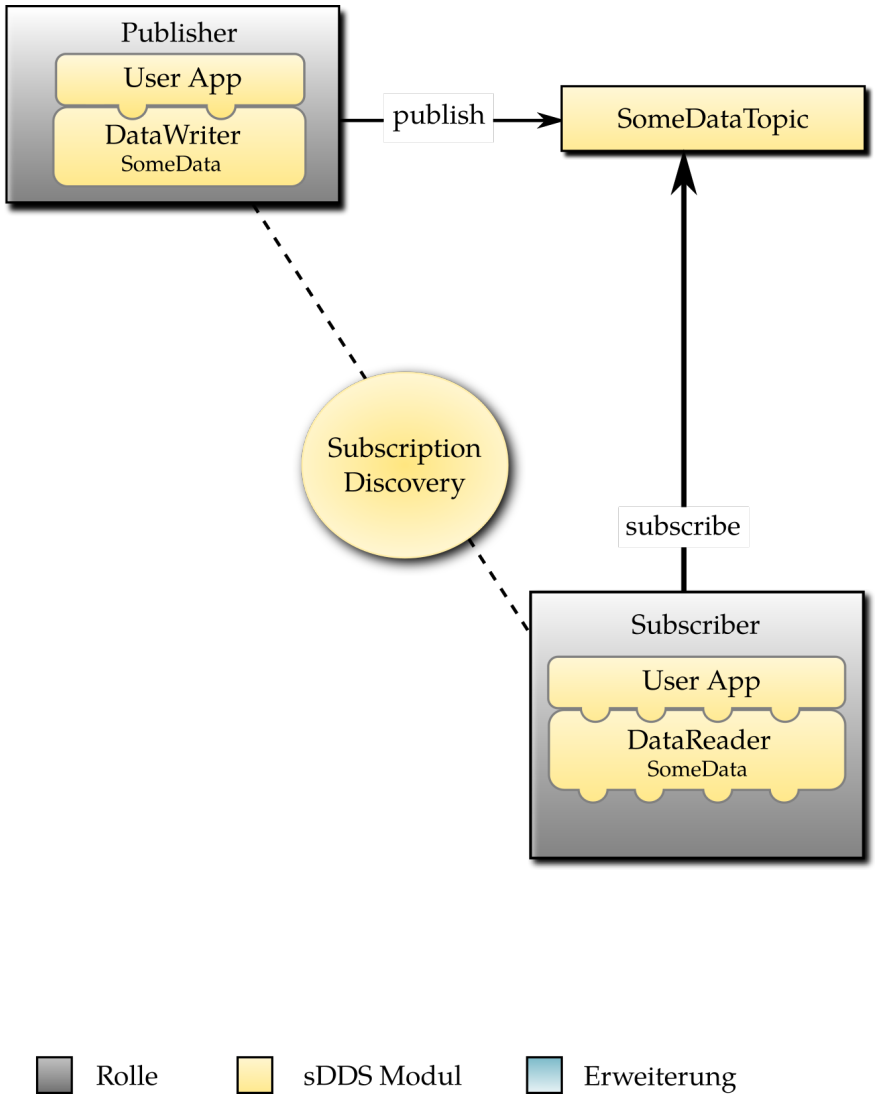
GENERISCHE SDDS-ERWEITERUNGEN

- → Lokales Koordinatensystem mit globalem Referenzpunkt
- → Umfasst nur anwendungsseitig interessierenden Bereich
- → Koordinaten als Integer gespeichert



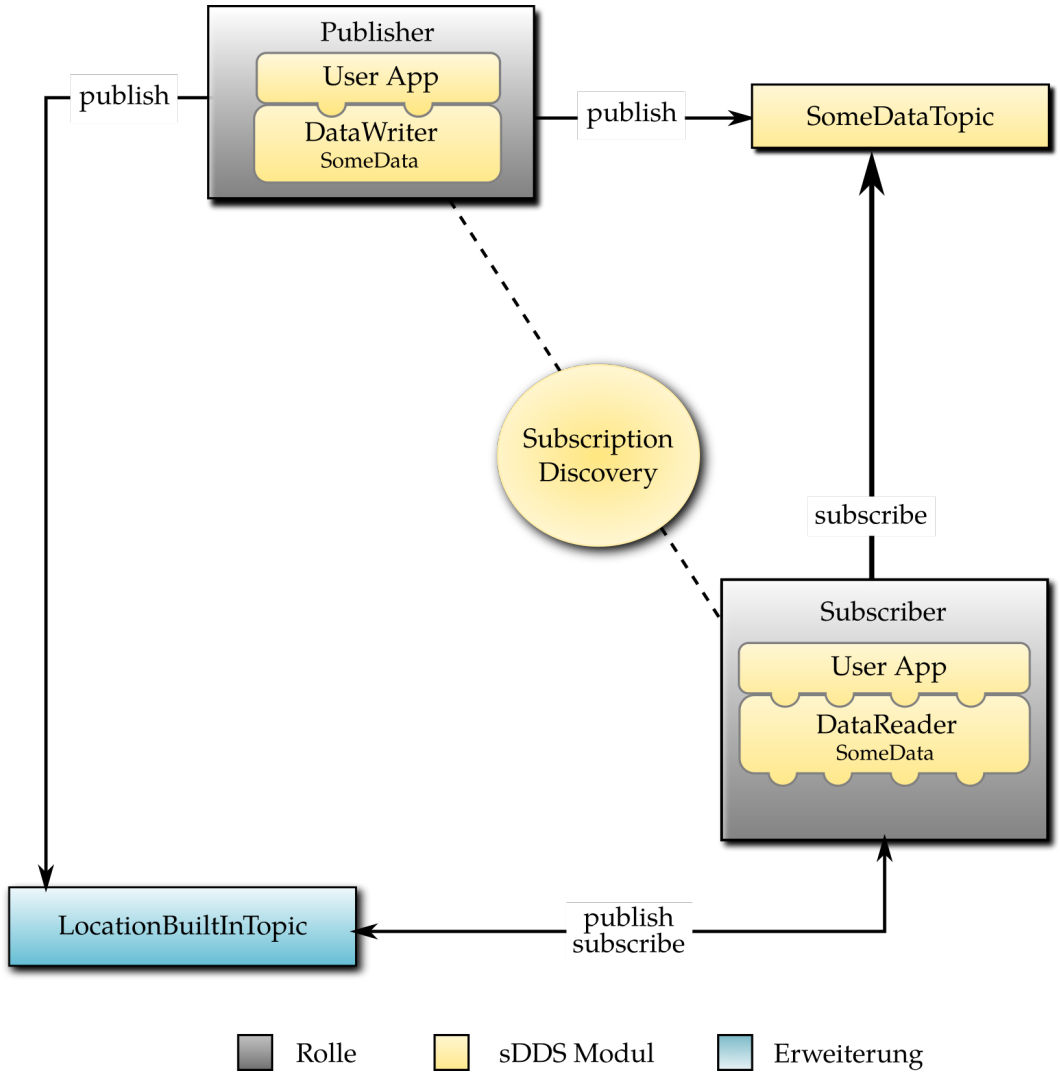
- → Verwendung des Open Geospatial Information Standard (openGIS) für Geometriemodell
- → Problem: Alle Formen als Polygone definiert
- → Lösung: Erweiterung durch einfache Grundformen
- → Lokalisierung wurde simuliert

BISHER


Publisher
User App
DataWriter
SomeData
publish
SomeDataTopic
Subscription
Discovery
subscribe
Subscriber
User App
DataReader
SomeData
Rolle
sDDS Modul
Erweiterung

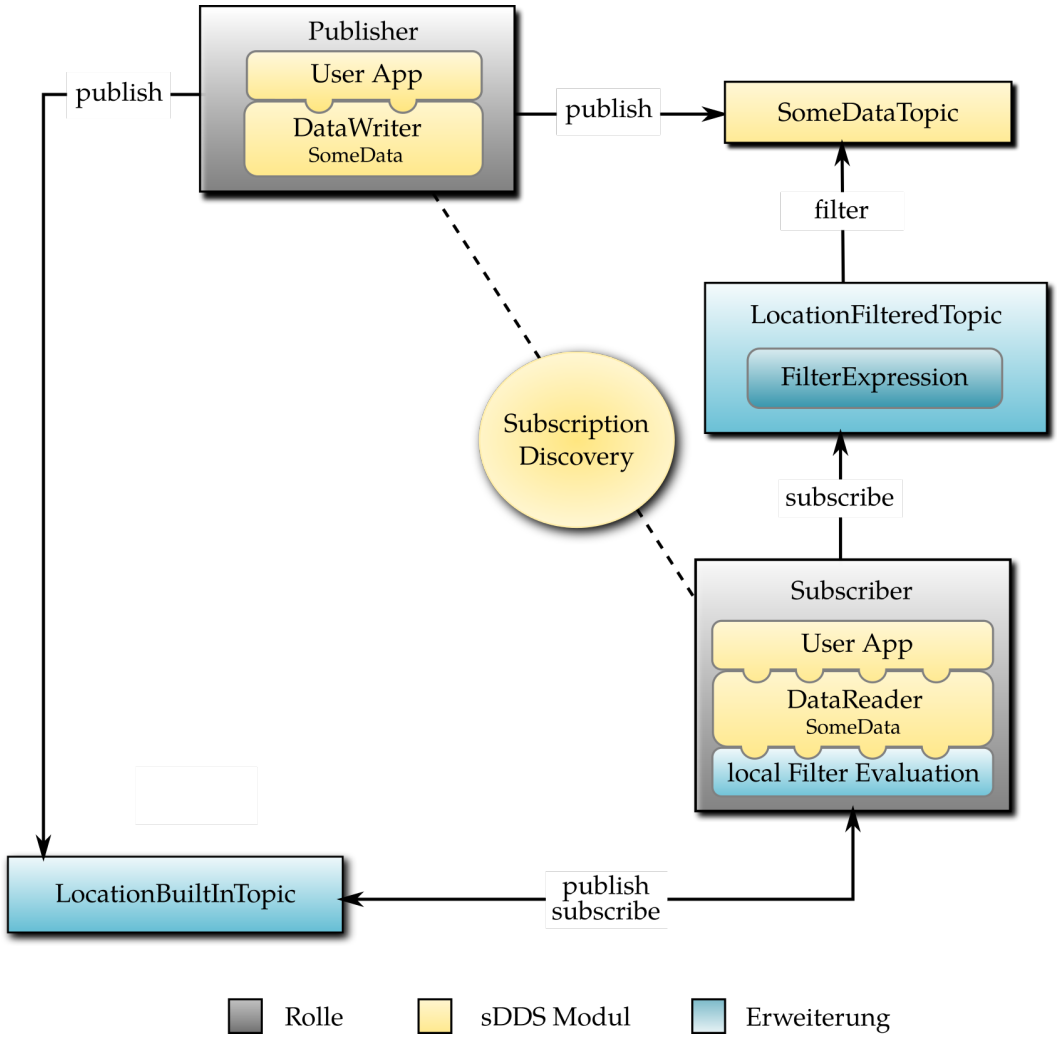
LOCATION BUILT-IN TOPIC

- → Neues Built-In Topic
- → Verbreitet Ortsinformationen zwischen allen Teilnehmern
- → Ortsinformation kann selbständig oder durch andere publiziert werden
- → Ermöglicht dadurch infrastrukturbasierte Lokalisierung



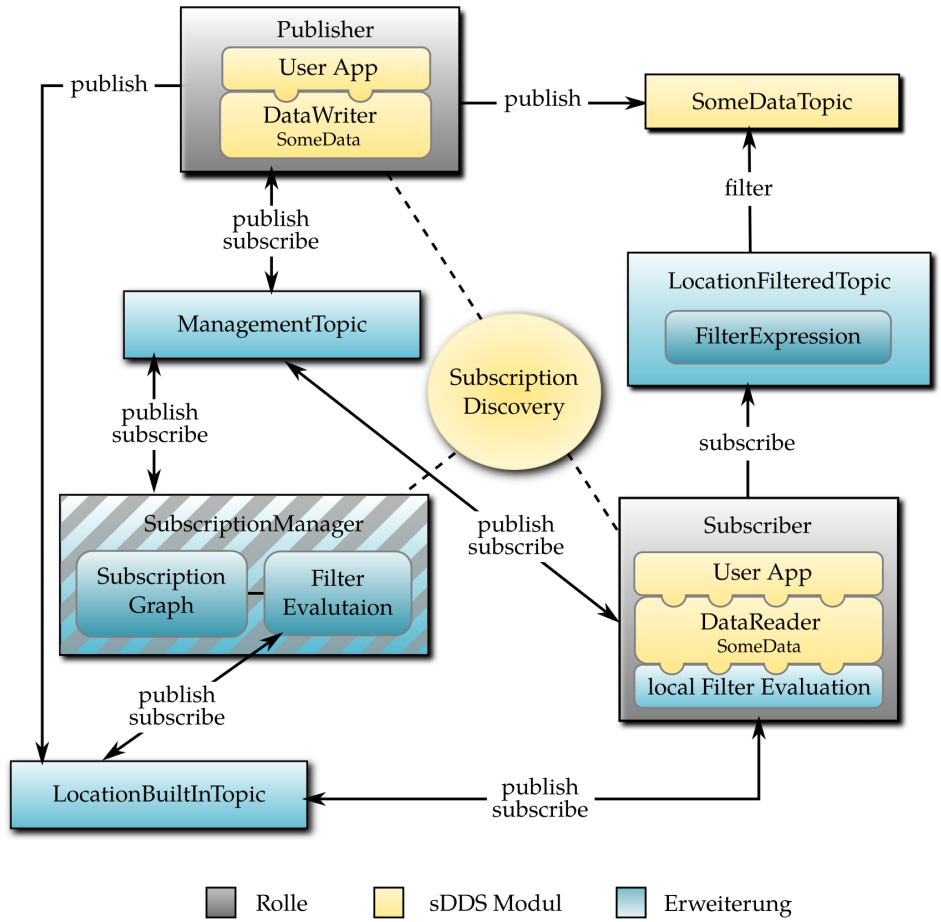
LOCATION FILTERED TOPIC

- → Ermöglicht Definition eines ortsbasierten Topic-Filters
- → Gibt Filtersyntax vor (openGIS SQL-Erweiterung)
- → Stellt Funktionen zur Auswertung des Filters bereit
- → Möglichkeit empfängerseitig zu filtern



SUBSCRIPTION MANAGER

- → Zentrale Instanz
- → Optimiert Datenverkehr
- → Beobachtet die Subscription Discovery
- → Speichert Informationen über alle Subscriptions in einem Graphen
- → Fordert Filter von Subscribern an
- → Wertet alle Filter aus
- → Wenn Daten irrelevant, wird Subscription pausiert
- → Möglichkeit quellseitig zu filtern



EVALUATION

EVALUATION

Zwei verschiedene Testumgebungen

- → Lokales RPi-Testbett
  - → Linux
  - → Raspberry Pi Cluster
  - → 20 Pis 1B+
  - → Gigabit-Switch
- → FIT/IoT-LAB-Testbett
  - → RIOT OS
  - → 382 STM32F1 (11 genutzt)
  - → ARM Cortex-M3, 72 Mhz, 64 kB RAM
  - → Kommunikation über 6LoWPAN
  - → Routing über mehrere Hops von Multicast-Adressen derzeit nicht von RIOT OS unterstützt
  - → Deshalb SubscriptionManager ebenfalls auf M3-Knoten

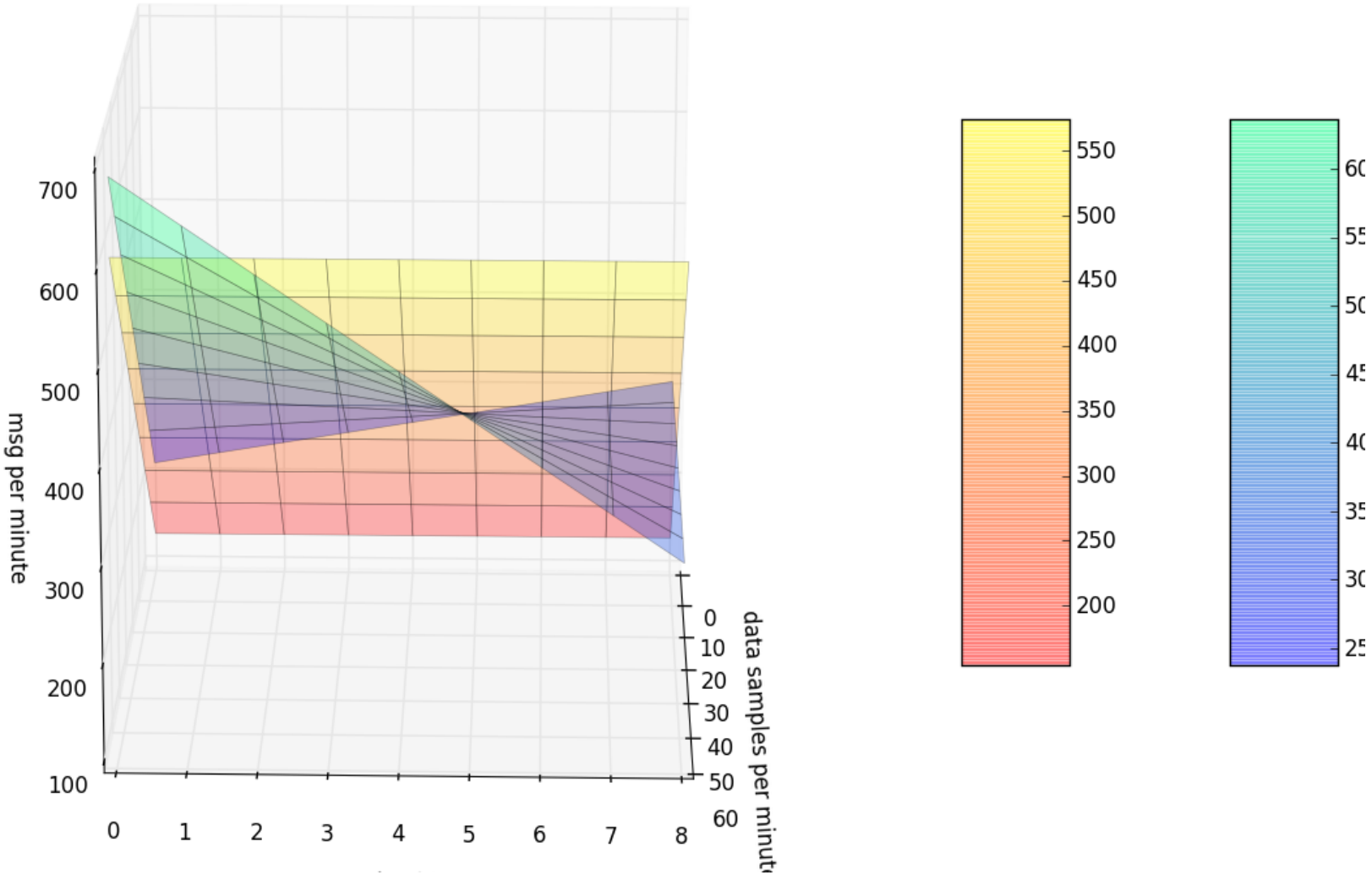
EVALUATION

Performance-Betrachtung

- → Dauer einen Filter zu setzen (RPi: 36,26 μs | M3: 38,00 μs)
- → Dauer der Filterauswertung (RPi: 7,00 μs | M3: 17,00 μs)
- → Dauer zum vollständigen Aufbau des Subscription-Graphen 11 Knoten: (RPi: 17,49 μs | M3: 45,08 μs)
- → Vorteil durch die Optimierung

EVALUATION

Optimierung durch Pausieren der Publisher (Pause 5 s)



EVALUATION

Zusammengefasste Ergebnisse

- → Setzen und Auswerten des Filters auf dem Raspberry Pi wesentlich schneller als auf dem M3 ⇒ bestätigt die Designentscheidung.
- → Optimierung lohnt sich, wenn Datenverkehr reduziert werden muss (viele Publisher, wenige Subscriber)
- → und um den Rechenaufwand der Subscriber zu reduzieren.

EVALUATION

Qualitative Betrachtungen

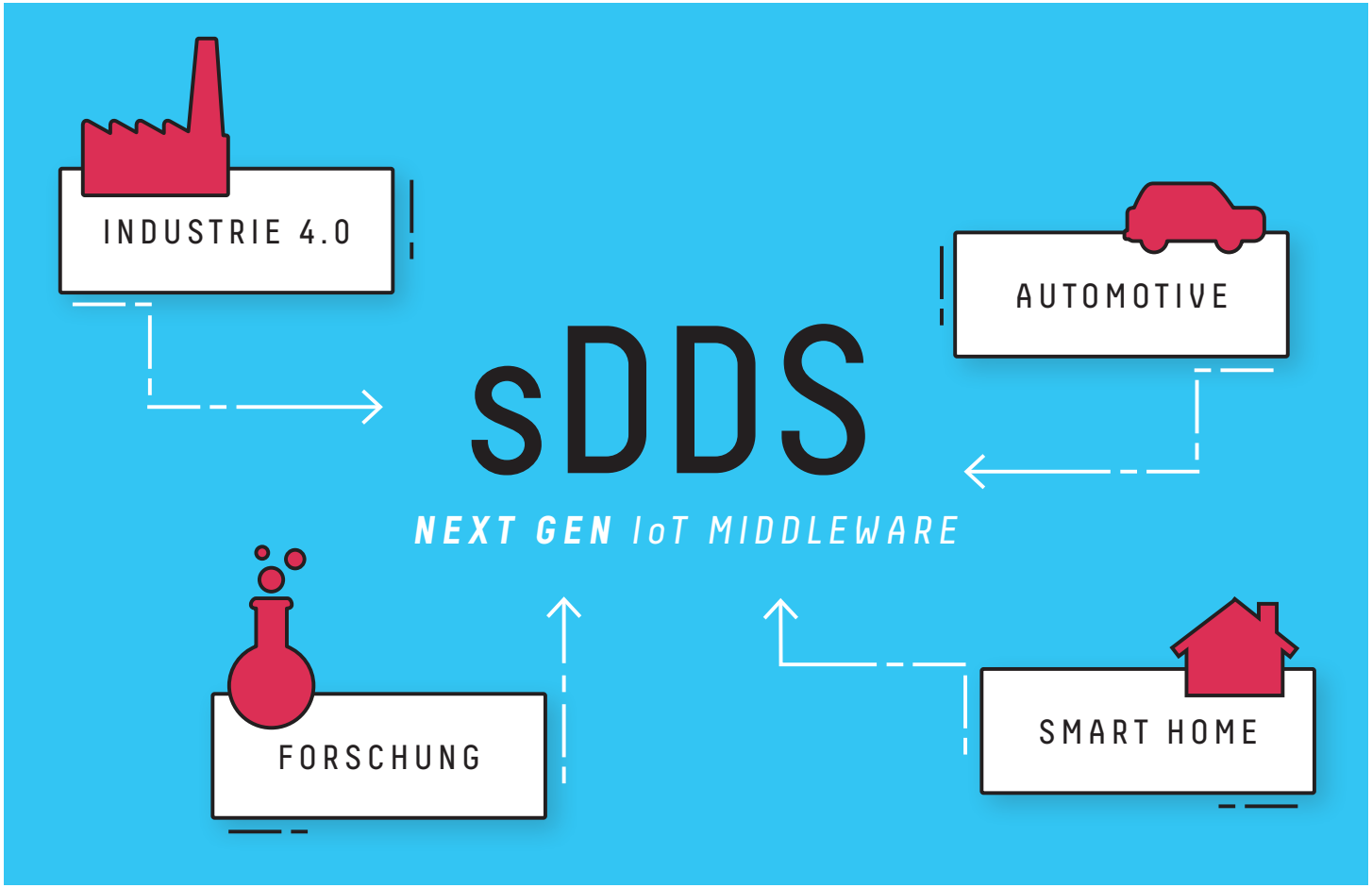
- → Modularität der Architektur
  - → Stand von sDDS zu Beginn der Thesis musste sauberer modularisiert werden.
  - → Danach war eine Modularisierung der Erweiterung problemlos möglich.
- → Portierung auf andere Plattformen
  - → Einfache Portierung durch die Plattformunabhängigkeit der Erweiterung.
  - → Lokalisierungsschnittstelle muss für die jeweilige Plattform implementiert werden.
  - → Durch Modularisierung lassen sich Teilfunktionen auch für kleinere Plattformen verwenden.

FAZIT UND AUSBLICK

FAZIT UND AUSBLICK

- → Ergebnis der Thesis stellt Basislösung dar
- → Möglichkeiten für Erweiterungen, z.B.:
  - → Vollständige Implementierung des Geometriemodells
  - → Optimierung auf Performance
- → Möglicher Einsatz im Szenario des Forschungsprojektes
- → Einsatz nicht nur im Smart Home möglich, auch Industrie 4.0, Logistik, Automotive etc.

SDDS GOES OPEN SOURCE



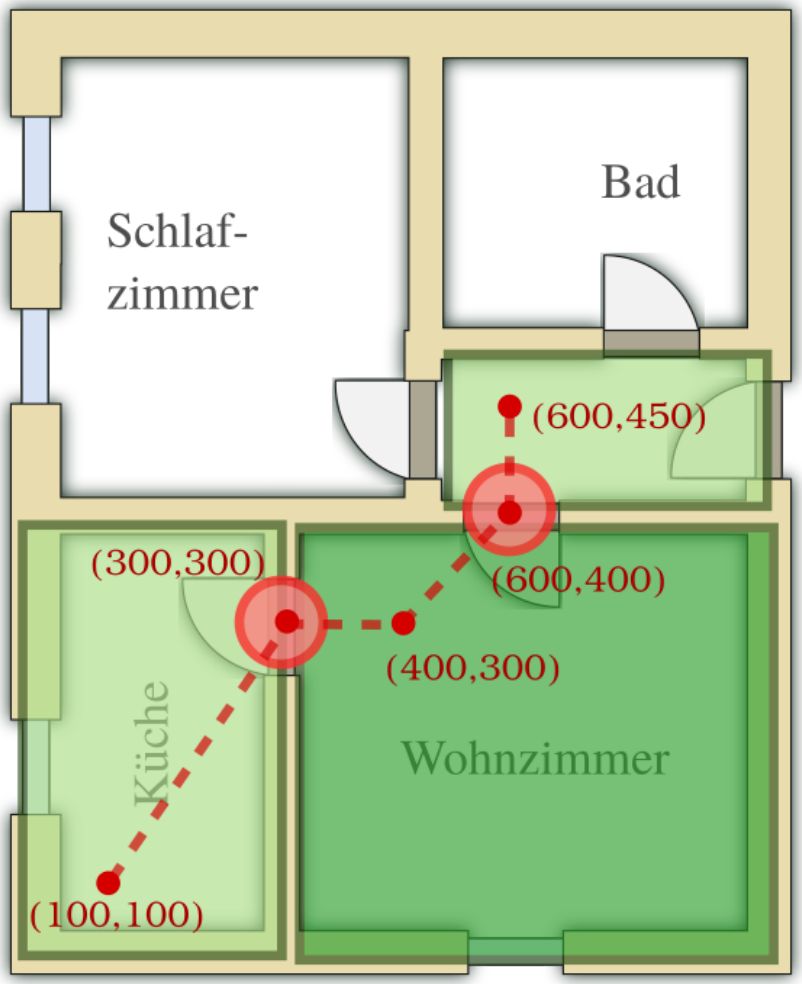
https://sdds.io

BACKUP

BACKUP

BEISPIEL

- → Staubsaugroboter bewegt sich zwischen Flur und Küche
- → Da im Wohnzimmer ein Teppich liegt, kann Staubsaugroboter hängen bleiben
- → Wenn Staubsaugroboter zwischen Küche und Wohnzimmer oder zwischen Flur und Wohnzimmer, dann Status empfangen
- → OVERLAPS kitchen OR OVERLAPS hallway AND OVERLAPS livingroom



ALLGEMEINE SDDS-ERWEITERUNGEN

Primär- und Sekundärschlüssel

- → Bis dahin noch nicht unterstützt
- → Freie Festlegung der Attribute bei der Modellierung
- → Codegenerierung für den Zugriff auf die Schlüssel und den Vergleich von Schlüsseln
- → Anpassen der Implementierung, so dass ein Aktualisieren der Daten möglich ist

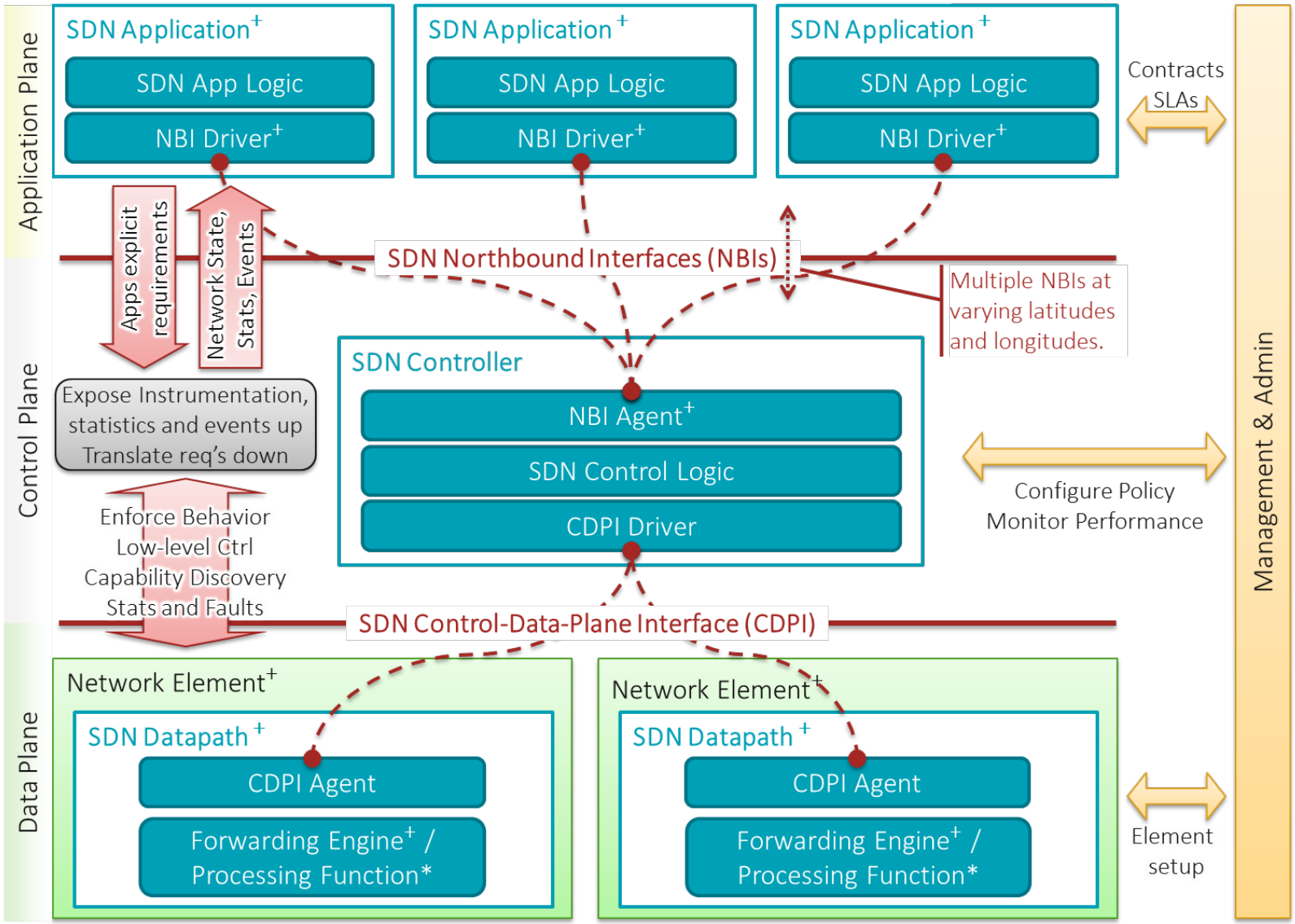
Discovery

- → Semantische Umstrukturierung des Moduls
- → Separates Modul zum Publizieren und Empfangen/Verarbeiten der Information

FILTERSYNTAX

```
1 <EXPR>       ::= <FUNC> <GEOMETRY> | NOT <FUNC> <GEOMETRY> | <EXPR> <CONNECTOR> <EXPR>
  <CONNECTOR>  ::= AND | OR
3 <FUNC>       ::= EQUALS | DISJOINT | INTERSECTS | TOUCHES | CROSSES | WITHIN | CONTAINS |
  OVERLAPS
5 <GEOMETRY>   ::= Point | LineString | LineRing | Line | Ellipse | Square | Polygon |
      PolyheadralSurface | EllipseExtrusion | SquareExtrusion | PolygonExtrusion |
      PolyhedralSurfaceExtrusion
```

SDN ARCHITEKTUR



+ indicates one or more instances | * indicates zero or more instances