Ubiquitous Computing in Business Processes Part III

Prof. Dr. Lutz Heuser
Urban Software Institute

Darmstadt
WS 2023/ 2024

Outline

1. Open Urban Data Platforms

Open Urban Data Platform

- facilitate the exploitation of city data, and intelligent monitoring and control of infrastructure and assets in cities, and with that enable new and improved services (both for public and private ends and organizations).
- They can be seen as system of systems, consisting of existing and new elements and evolve over time.
- An Open Urban Platform integrates (previously) siloed city infrastructures, data platforms, and services, from different domains.
- To put it more simple, urban platforms integrate IoT enabled infrastructure and assets in cities, make it possible to operate and collect and (re)distribute/use data from these and other sources, and make these available to applications.

Open (Urban) Data

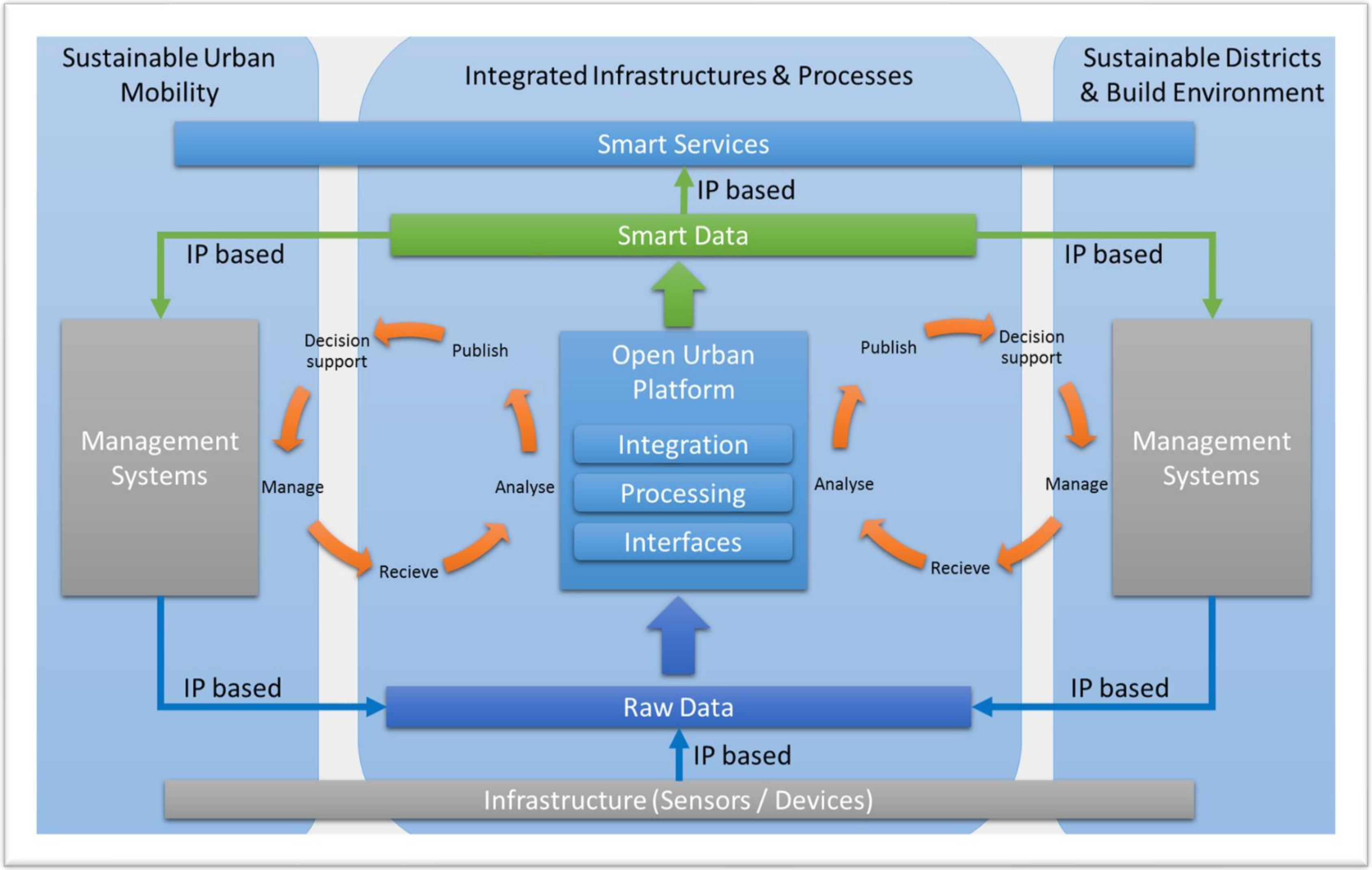
- Open data refers to data sets provided by public authorities and public enterprises to the general public.
- Often, this term is also used to define the usage of the data sets as "free of charge" using an appropriate open data license agreement.
- Open Data is open if the provided data can be processed and used by anyone and for any purpose. In general, Open Data shall adhere to criteria such as "timeliness", "as close as possible to the data source", "publication under an open license" such as published by the Creative Commons[1].
- Cities provide Open Data Portals to share their public data, e.g.
  - https://daten.berlin.de/ (Berlin Open Data)
  - https://www.offenedaten-koeln.de/ (Offene Daten Köln)
- **Issue**: Most open data sets do not provide real-time data, i.e. no real-time data from smart items

[1] https://creativecommons.org/about/program-areas/open-data/

Open Urban Data Platform (OUP)

1. Open Urban Data Platforms are data-driven architectures
2. Data flows from external (management systems) to the OUP by
   a. providing APIs
   b. or pushing data, e.g. via eMail
3. Smart Items could send their data
   a. Either directly to the OUP
   b. or via a given management system
4. Incoming data is called “**raw data**”, as the data has been provided in an unchanged often proprietary format
5. The OUP harmonize the data and processed the data such that it generates consumable data called “**smart data**”.
6. The output will be consumed by smart services such as APPs or by (other) management systems for further processing.

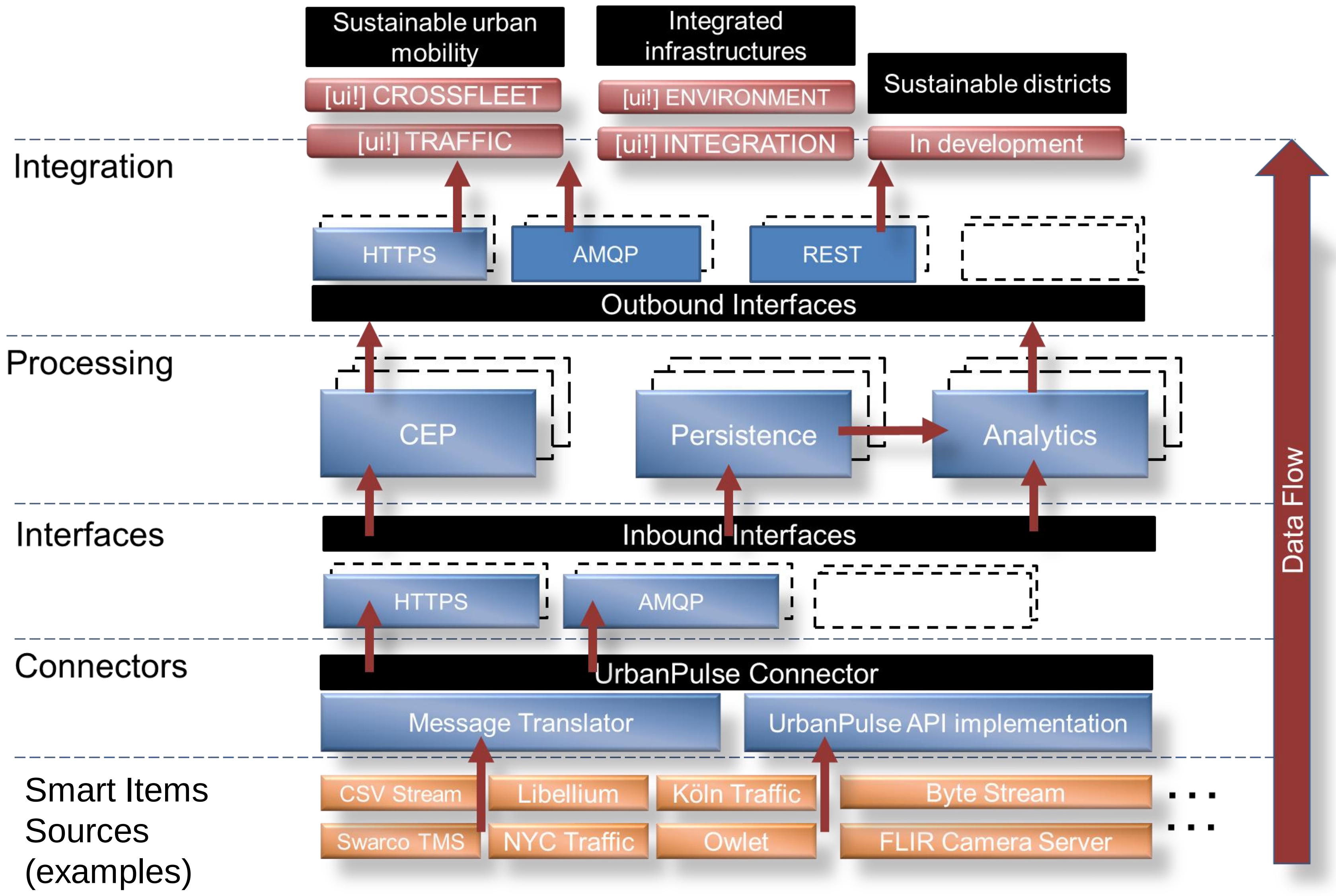
Open urban Platform based on DIN SPEC 91357



The Data Lifecycle of OUPs

1. Receive
    a. The OUP receives the “raw data” from both management systems and smart items
2. Analyse
    a. Harmonize the input data to allow for integrated analysis
    b. Analyse using AI-tools such as Machine Learning algorithms
3. Publish
    a. via OpenAPIs “smart data” is published for consumption by smart services and management systems
    b. or via a given management system
4. Decision Support
    a. Based on the “smart data” better insights lead to educated decisions (rf. High resolution enterprise management)
5. Manage
    a. Based on the educated decisions processes may be adapted

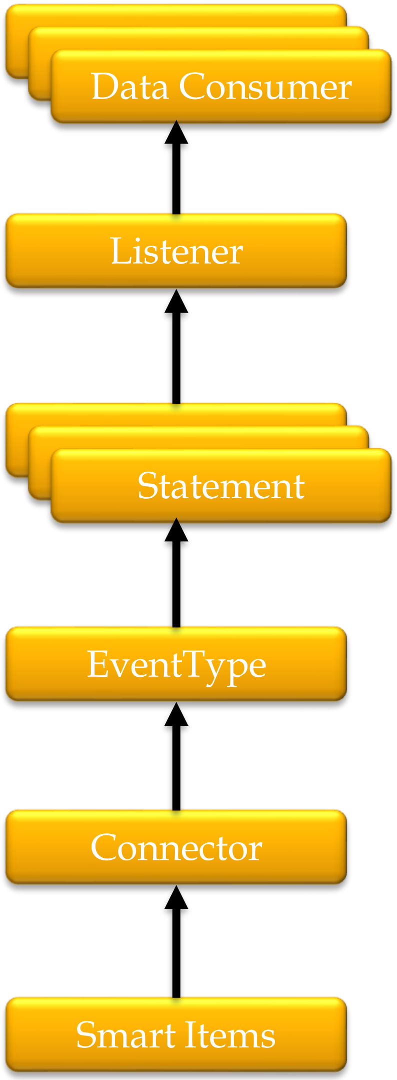
[ui!] UrbanPulse – An Example of DIN SPEC 91357 OUP



The Data Flow of [ui] UrbanPulse

1. Smart Items Sources
   a. Data coming from Smart Items or their related management systems
2. Connectors
   a. Takes data and harmonize the input data to allow for integrated analysis
3. Interfaces
   a. As connectors may reside at the site of the Smart Items Sources the data needs to be send to the OUP using different messaging protocols
4. Processing
   a. Event processing taking the data of the Smart Items as input ("raw data") to create output ("smart data")
5. Integration
   a. Provide the smart data via OpenAPIs and different messaging protocols

[ui] UrbanPulse – Principle Data Modell



e.g. [ui!] Cockpit

Listener receives events, which are defined by statements.

Statements select and combines events to „business events“

Each Smart Item detects events of a given Type (EventType)

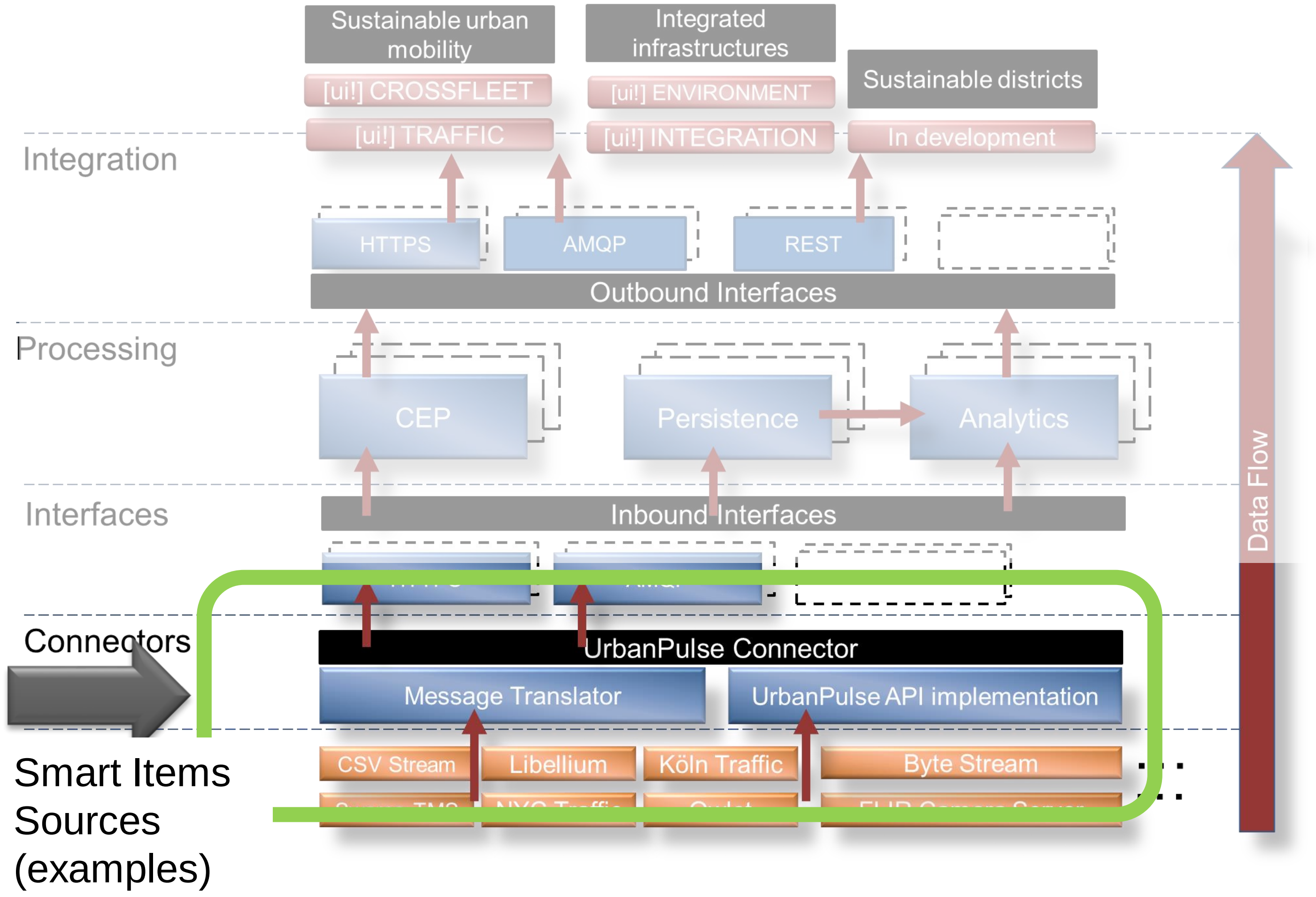
A connector wraps the API to the sensor data provider

Each Smart Item has a definition and a universal unique ID (aka auto-ID).

UrbanPulse - Module

UrbanPulse
Modules

UrbanPulse - Connectors



UrbanPulse - Connectors

Connectors allow to connect systems and Smart items of different vendors, types, and domains:

- Energy
- Traffic
- Buildings
- Environment

Tasks

- Communicate with [ui!] UrbanPulse API
- Authentication
- Transformation of raw data in Smart Items agnostic format

[ui] UrbanPulse - Connectors

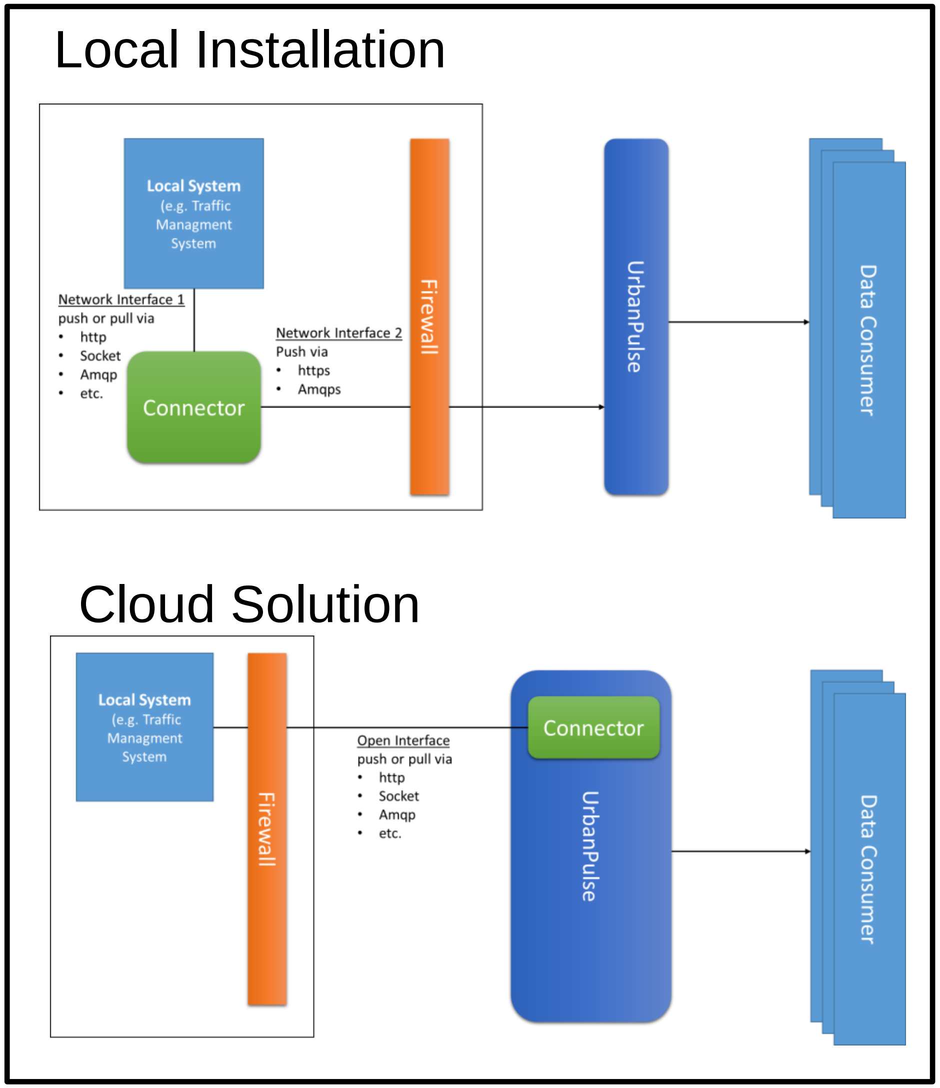
Connectors can be operated locally or in the cloud.

Supported communication protocols

- TCP/UDP
- HTTP(S)
- Web socket
- MQTT
- And if needed, more…

Benefits

- Easy
- Simple
- Scalable



UrbanPulse – Connectors (Example)

- SM!GHT is an integrated street light lamppost consisting of
  - Environmental Sensors such as
    - Particle Matters (PM 10) – referred to as "Dust detector"
    - Ambient Light detector - referred to as "Ambient Light V2"
    - Barometer sensor – referred to as "Barometer"
    - Humidity sensor – referred to as "Humidity"
    - Temperature sensor – referred to as "Temperature"
  - Sound sensor – referred to as "Sound Intensity"
- Each SM!GHT sends its data to the SM!GHT Backend (aka management system) which provides an API to the OUP
- A SM!GHT is a complex Smart Items as it stores multiple data sets of different event types
- Therefore, the SM!GHT Connector has to harmonize the data sets such that the different events could be processed according to their semantics

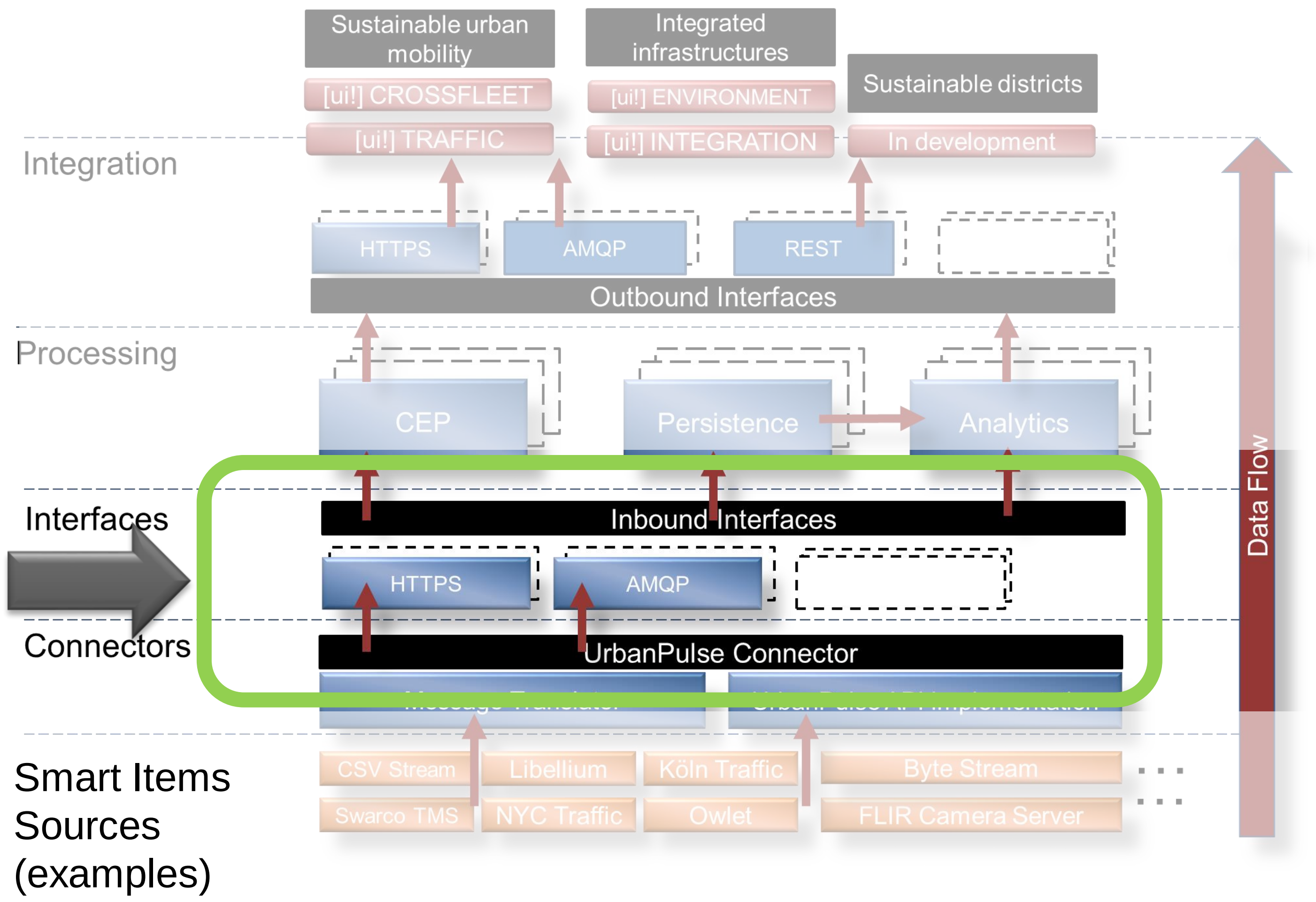
UrbanPulse – Connectors (Example)

SM!GHT
smart.city.light

```
public class ProcessingVerticle extends DefaultProcessingVerticle {
   /**
    * @param event
    * @return EnBW event type name mapped from 'sensorTyp' field, null in case of null or
invalid field
    */
   @Override
   public String getEventTypeName(JsonObject event) {
      String sensorTyp = event.getString("sensorTyp");
      if (null == sensorTyp) {
         container.logger().warn("null 'sensorTyp' field!");
         return null;
      }

      switch (sensorTyp) {
         case "Dust detector":
            return "DustEventType";
         case "Ambient Light V2":
            return "AmbientLightEventType";
         case "Barometer":
            return "BarometerEventType";
         case "Humidity":
            return "HumidityEventType";
         case "Sound Intensity":
            return "SoundIntensityEventType";
         case "Temperature":
            return "TemperatureEventType";
         default:
            container.logger().warn("unsupported 'sensorTyp' value: " + sensorTyp);
            return null;
      }
   }
```

UrbanPulse – InboundInterfaces



[ui] UrbanPulse Inbound Interfaces

The Inbound Interface is a highly scalable publish-subscribe service that can receive millions of events per second and stream them into the OUP.

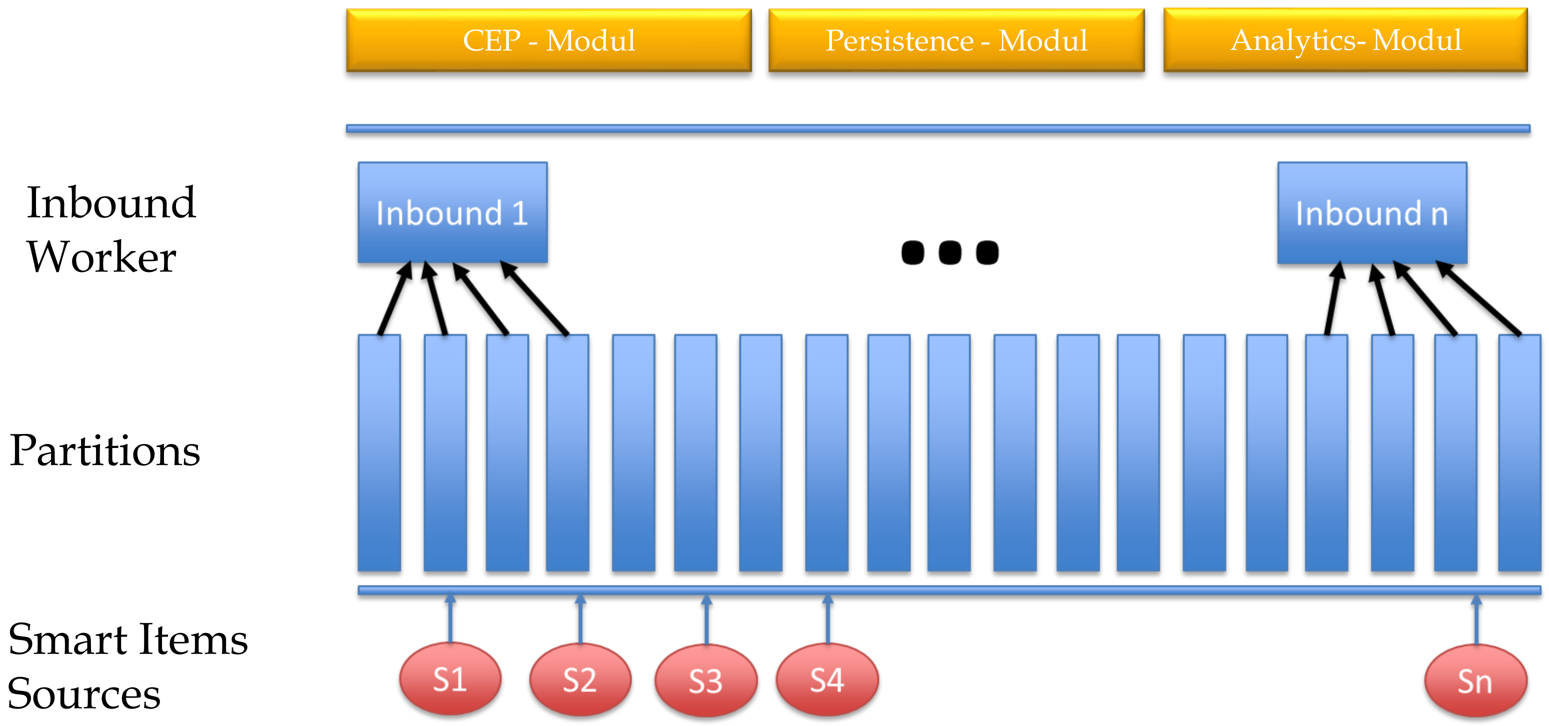
1. Smart Items Sources
   a. Data coming from Smart Items or their related management systems via CONNECTORS **at high volume and need to be consumed in parallel**

2. Partitions
   a. Smart Items event messages will be streamed and ingested into the OUP by storing them into a Blob Storage, which is optimized for storing massive amounts of unstructured data.

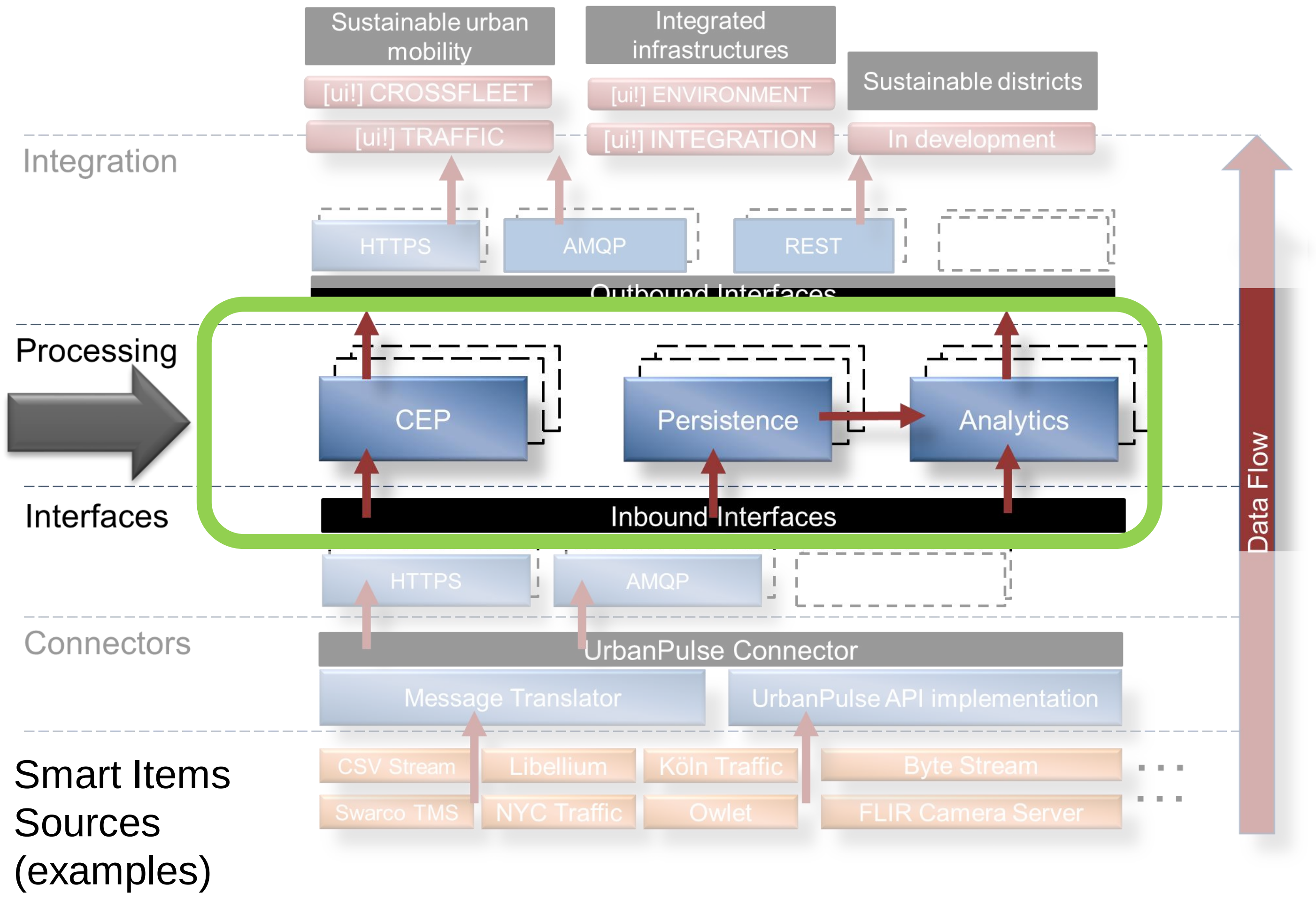
3. Inbound
   a. The Inbound Interface receives the Smart Items event messages from the Connectors and transmits them to the other UrbanPulse modules (in the processing layer).

[ui] UrbanPulse – Inbound Interfaces

High performance and scalable interface for Smart Items event messages.



UrbanPulse - Processing



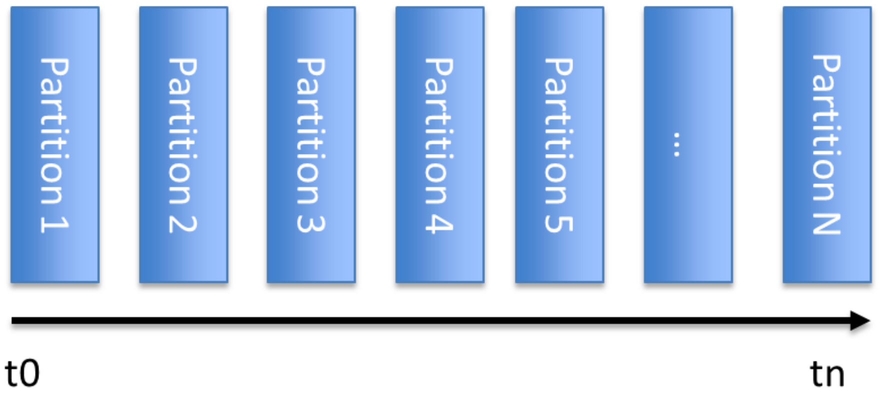
UrbanPulse Persistence

- As the OUP [ui!] UrbanPulse is designed to store massive streaming data from Smart Items, the persistence layer is optimized such that other (analytics) modules could find and access data sets, fast.
- The Persistence module also has the capability to configure a hierarchical storage implementation, which allows combining the advantages of different storage system.
- The first level storage can be configured to make use of an In-Memory storage system in order to provide fast responses while the second level can be a long-term storage system.
- By doing so the module can collect huge amounts of data on the one hand, on the other hand it ensures a fast response time if historical data is requested.
- Time Stamp Schema
  a. Allows to store those events nearby, which have happened within the “same” period
  b. Allows fast reading, e.g. “search for all events which happened last week, same day, same time”

UrbanPulse - Persistence

High performance and scalable storage structure for events

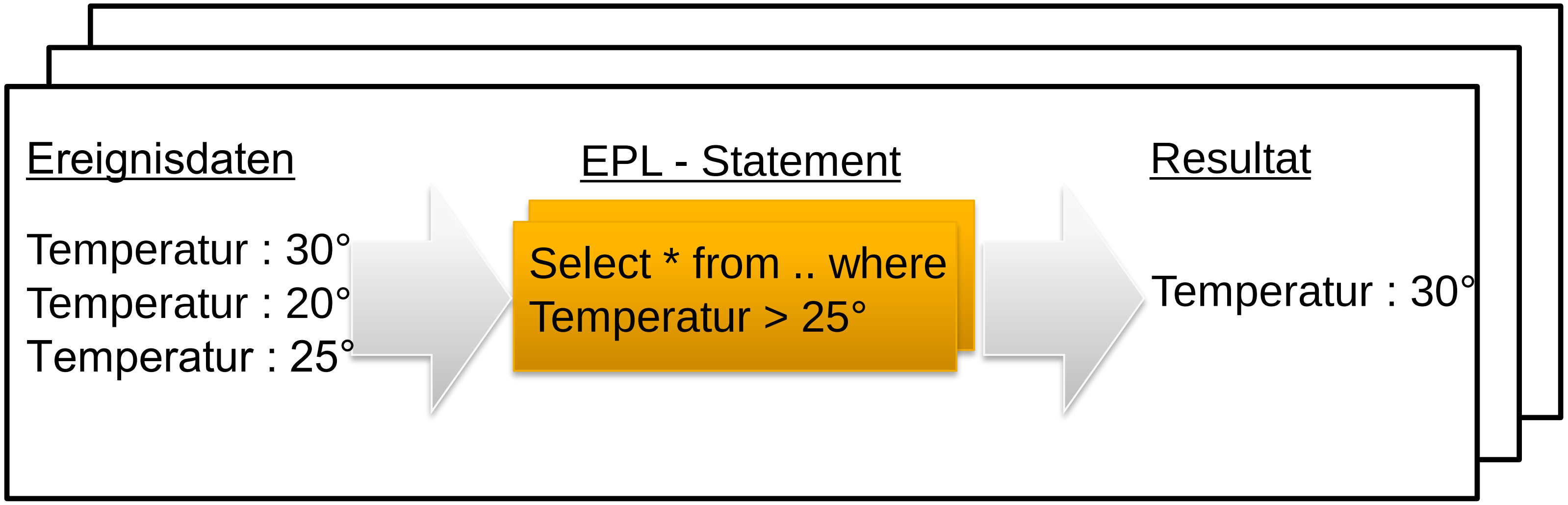
Partitions will be stored on different storage servers



20151009101010999_X |Daten
20151009101011999_X |Daten
20151009101012999_X |Daten
...
20151009101030999_X |Daten
20151009101059999_X |Daten

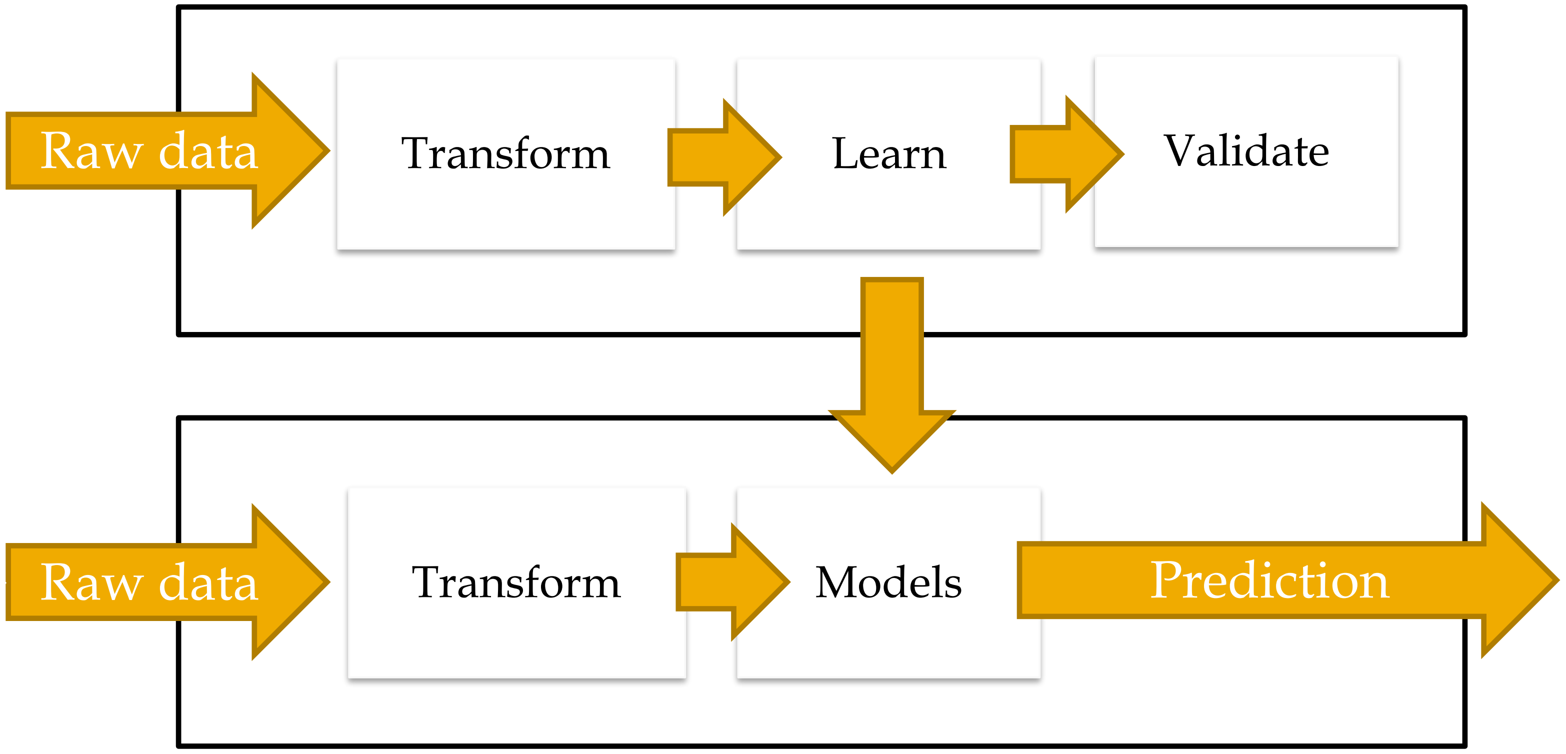
UrbanPulse – Complex Event Processing

Complex Event Processing Module to select and aggregate events

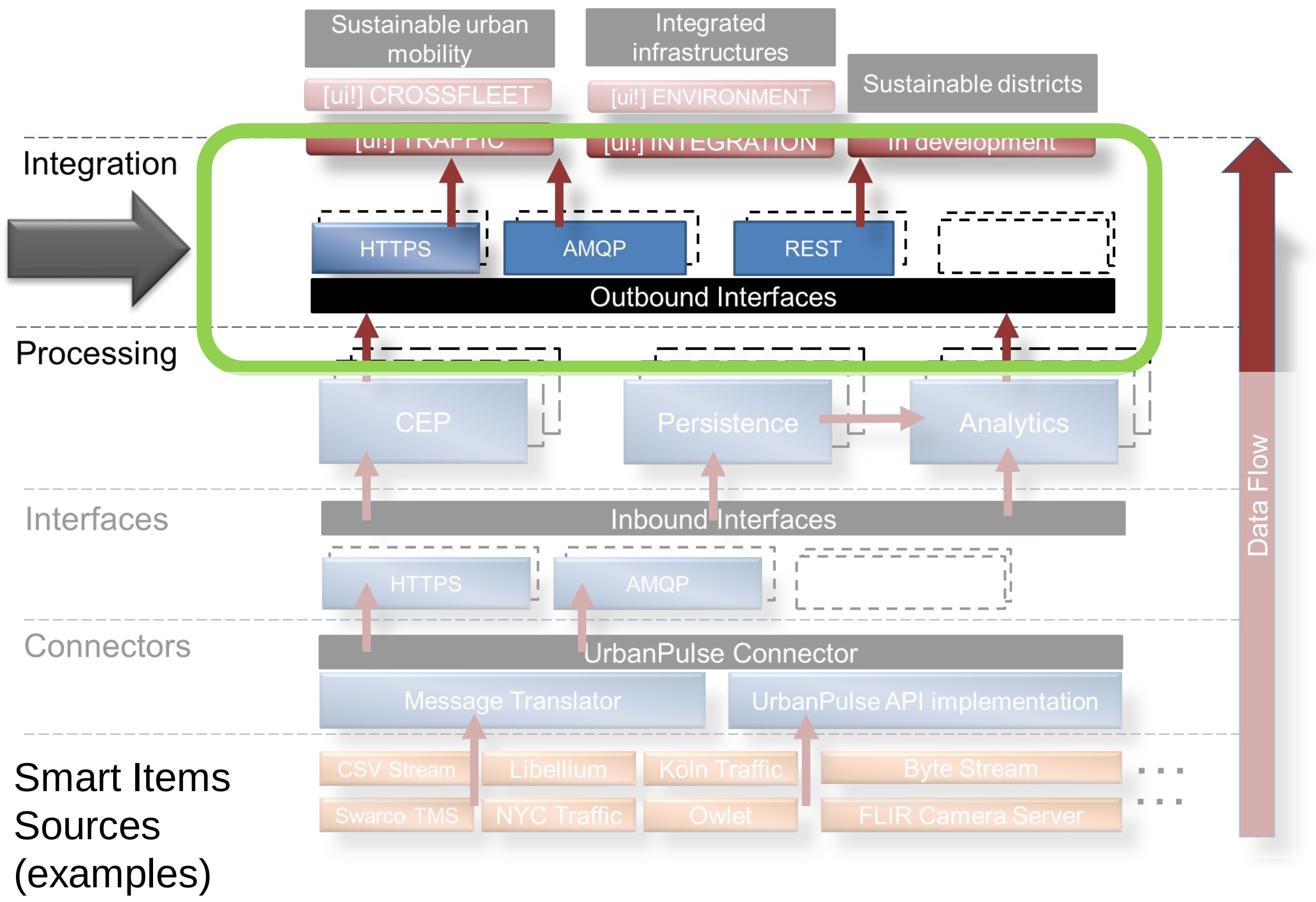


UrbanPulse - Analytics

Stream Processing und Machine Learning

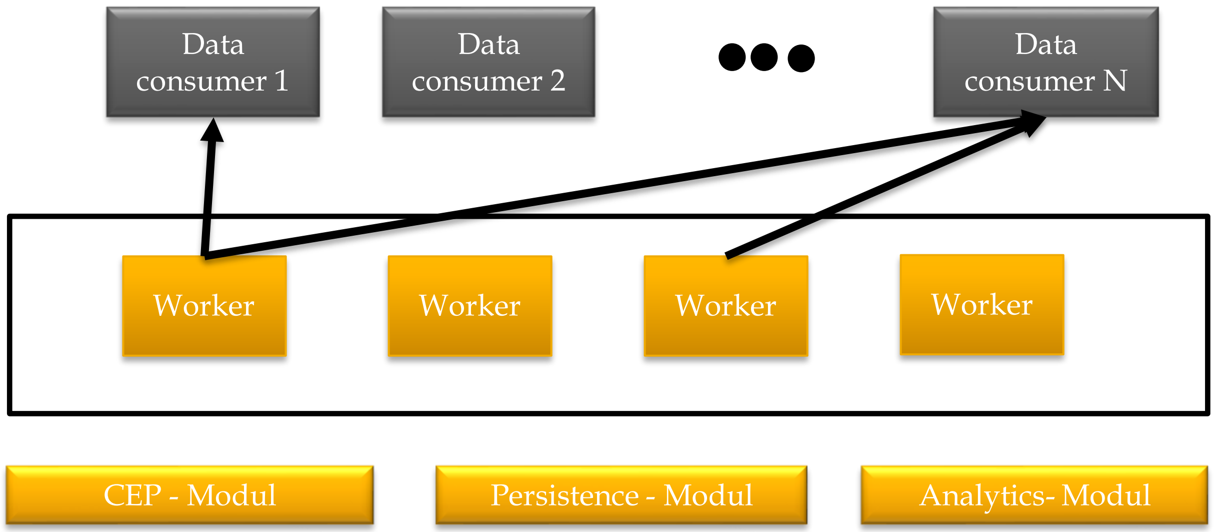


UrbanPulse - Integration



[ui!] UrbanPulse – Outbound Interfaces

Worker of Outbound Interfaces distribute the Data to different consumers



[ui!] UrbanPulse – Outbound Interfaces

- The modules of the Outbound Interface layer are responsible for data distribution and provisioning.
- They provide the interfaces between the platform process layer services and the data consumers.
- The [ui!] UrbanPulse Outbound Interfaces can be seen as a facade for the urban data sources and analytic services.
- The communication between the backend services and the client applications is realized by a combination of a pub/sub system for event data and REST (Representational State Transfer) Interfaces for persisted data.