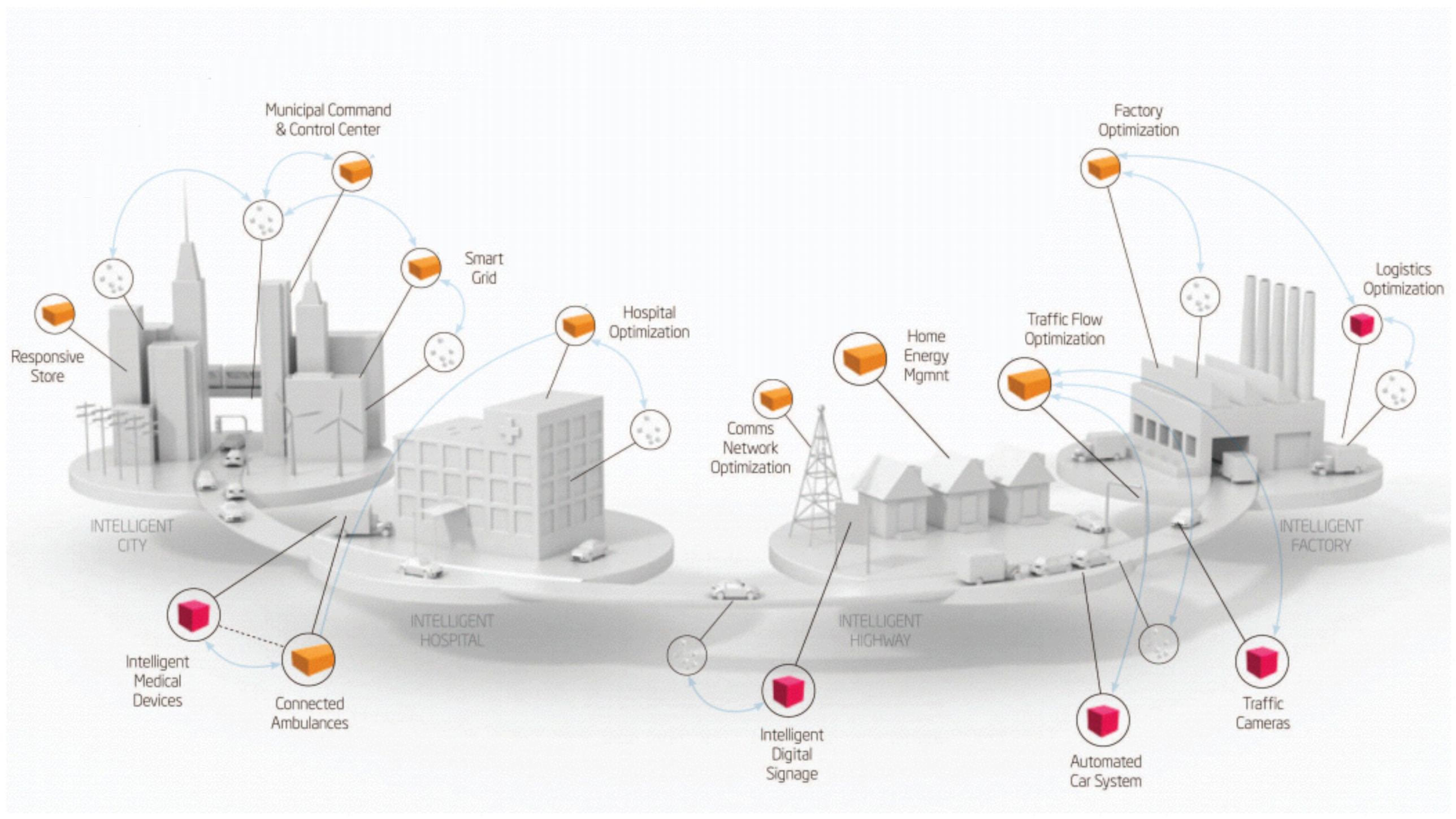
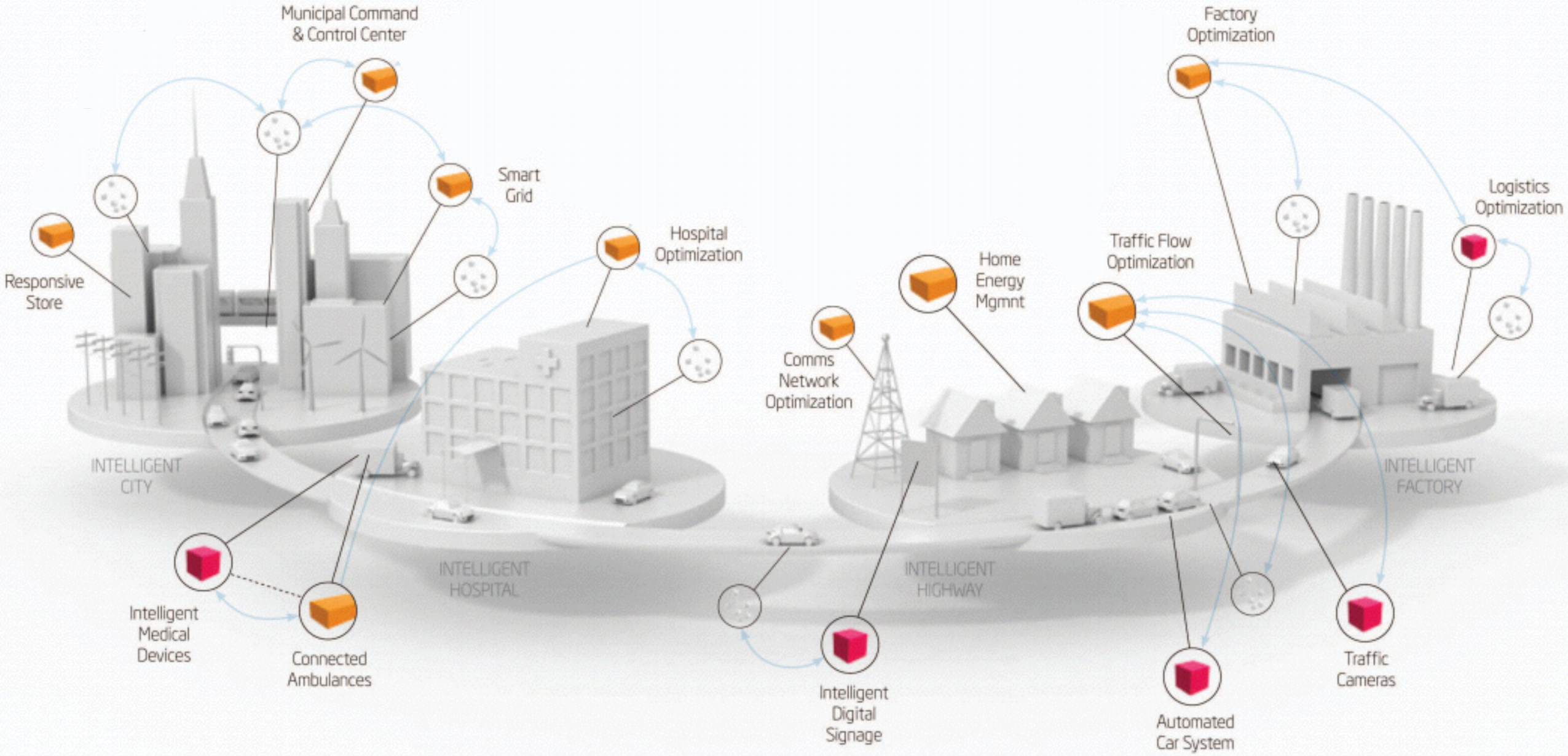
PRISMTECH

Reactive Data Centric Architectures with DDS

Angelo Corsaro, PhD
Chief Technology Officer
angelo.corsaro@prismtech.com

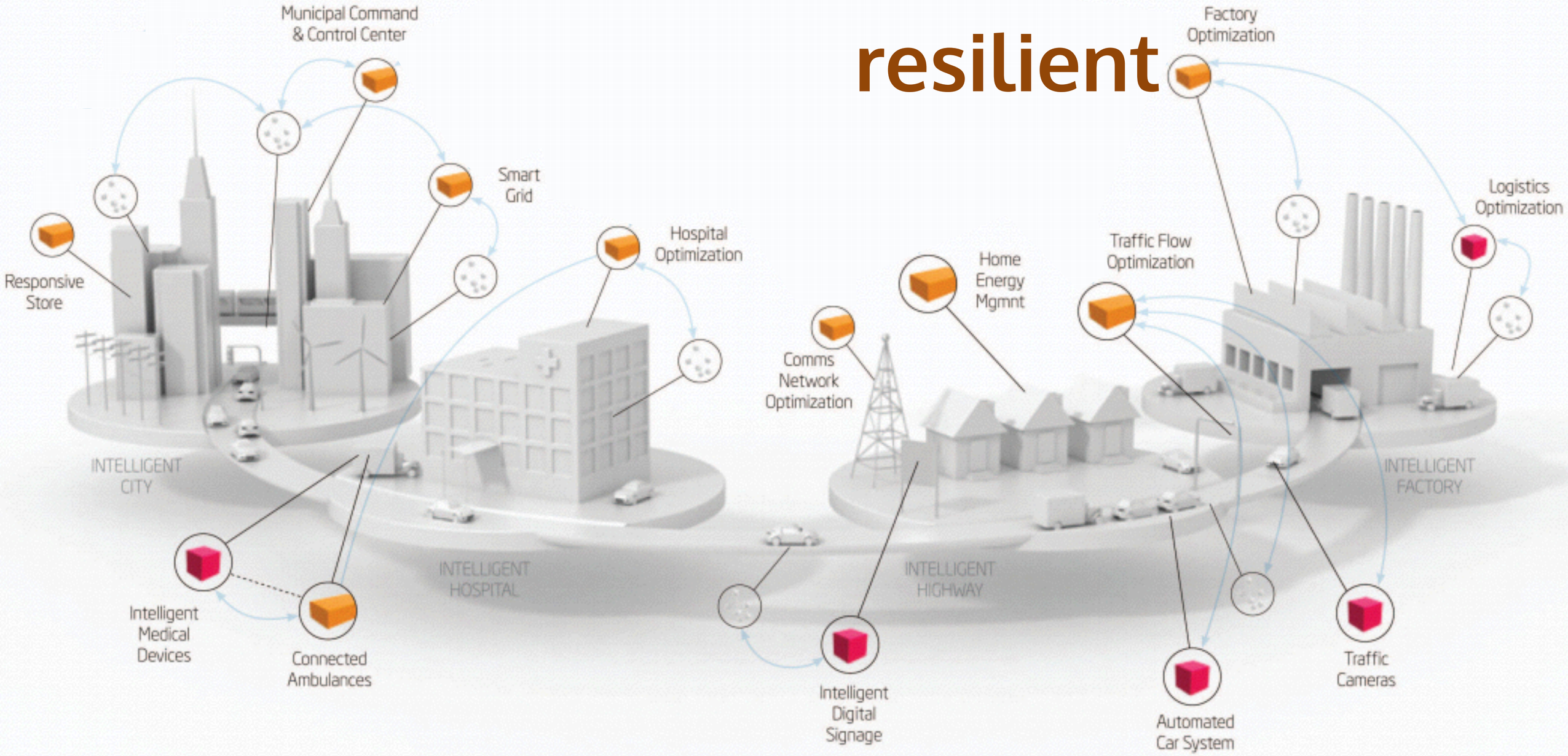

Municipal Command & Control Center
Smart Grid
Responsive Store
Hospital Optimization
INTELLIGENT CITY
INTELLIGENT HOSPITAL
Intelligent Medical Devices
Connected Ambulances
Home Energy Mgmnt
Comms Network Optimization
INTELLIGENT HIGHWAY
Intelligent Digital Signage
Factory Optimization
Traffic Flow Optimization
Logistics Optimization
INTELLIGENT FACTORY
Automated Car System
Traffic Cameras

responsive


Municipal Command & Control Center
Smart Grid
Responsive Store
Hospital Optimization
INTELLIGENT CITY
INTELLIGENT HOSPITAL
Intelligent Medical Devices
Connected Ambulances
Factory Optimization
Logistics Optimization
Traffic Flow Optimization
Home Energy Mgmnt
Comms Network Optimization
INTELLIGENT FACTORY
INTELLIGENT HIGHWAY
Intelligent Digital Signage
Automated Car System
Traffic Cameras

responsive

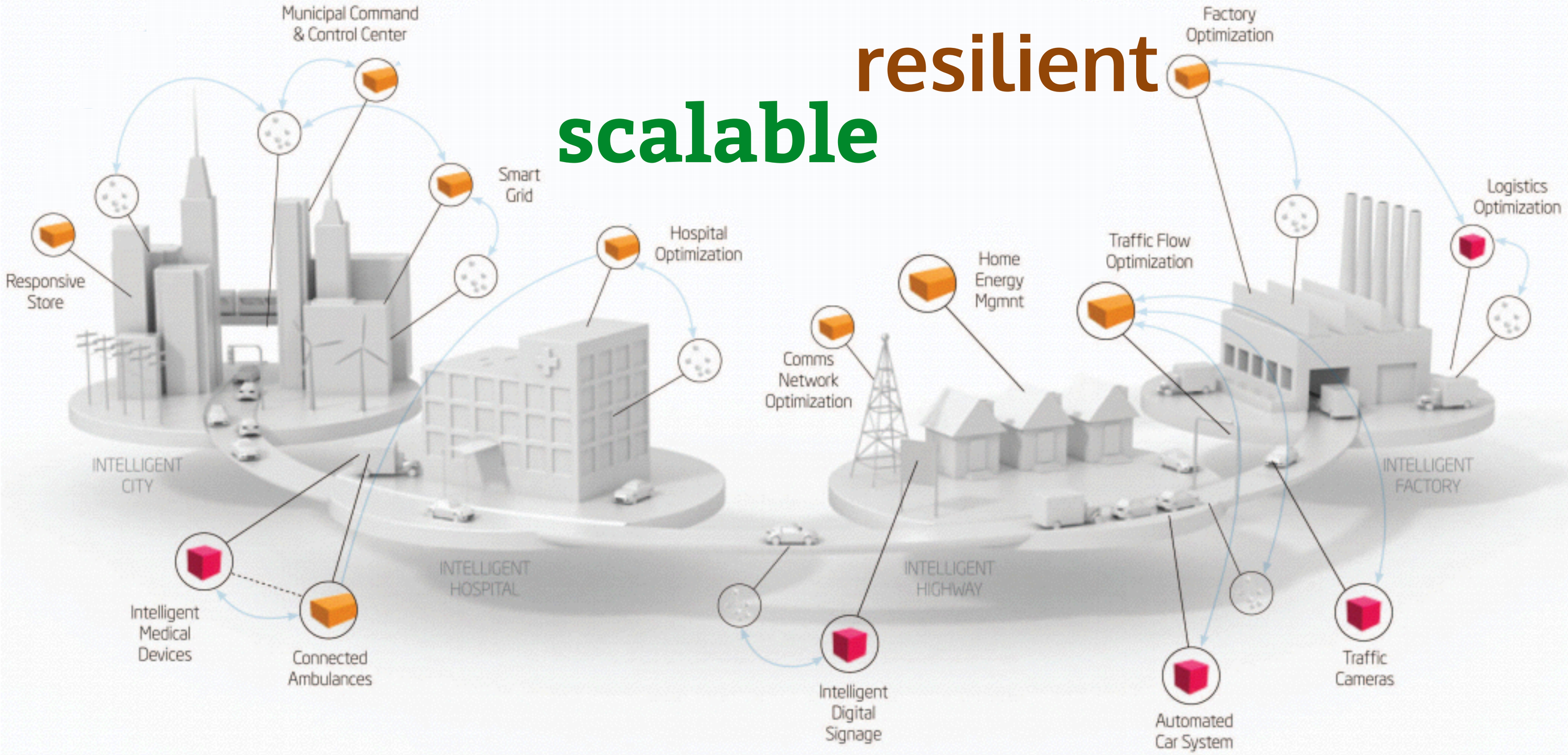
resilient



responsive

resilient

scalable

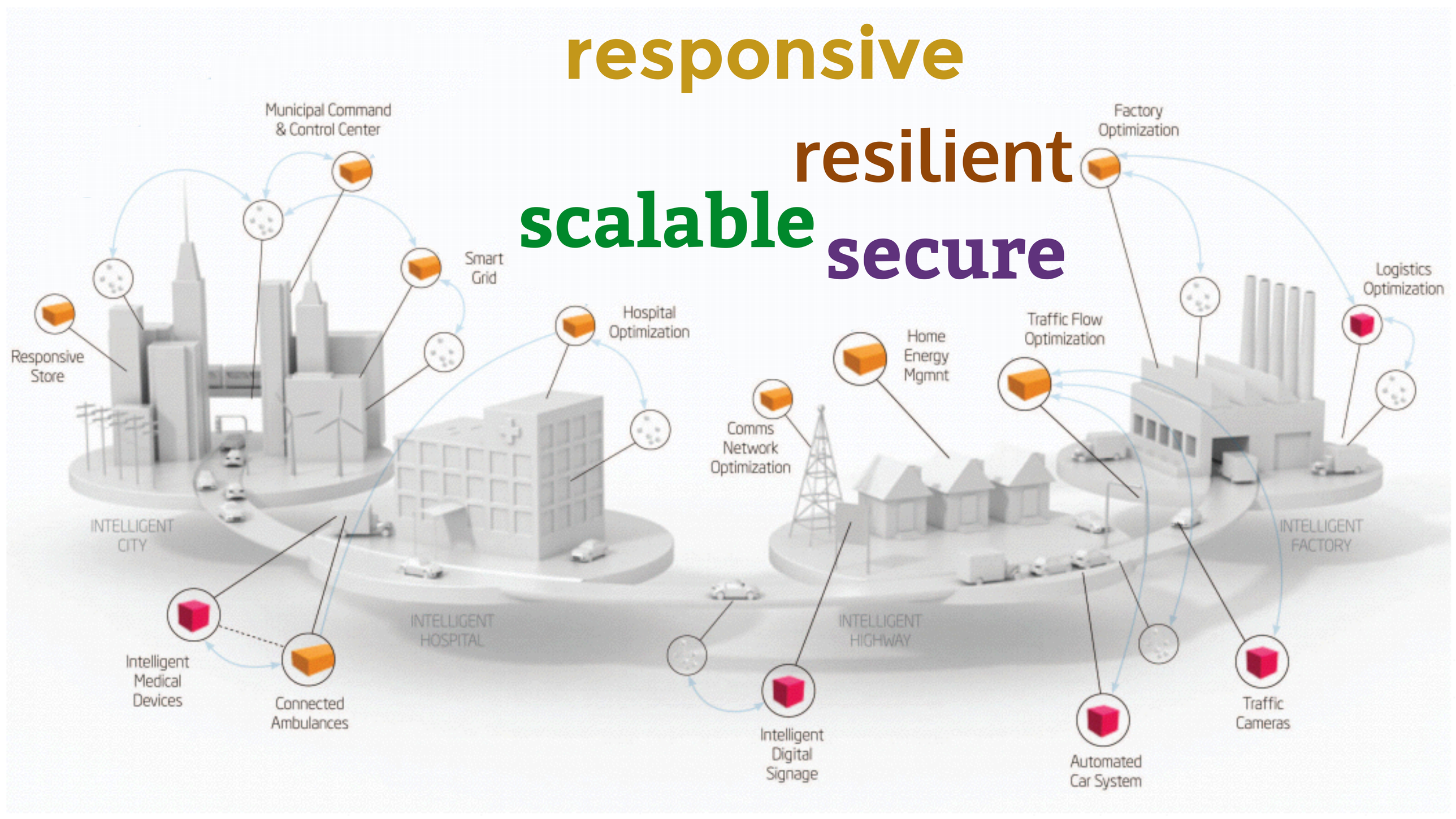


responsive

resilient

scalable

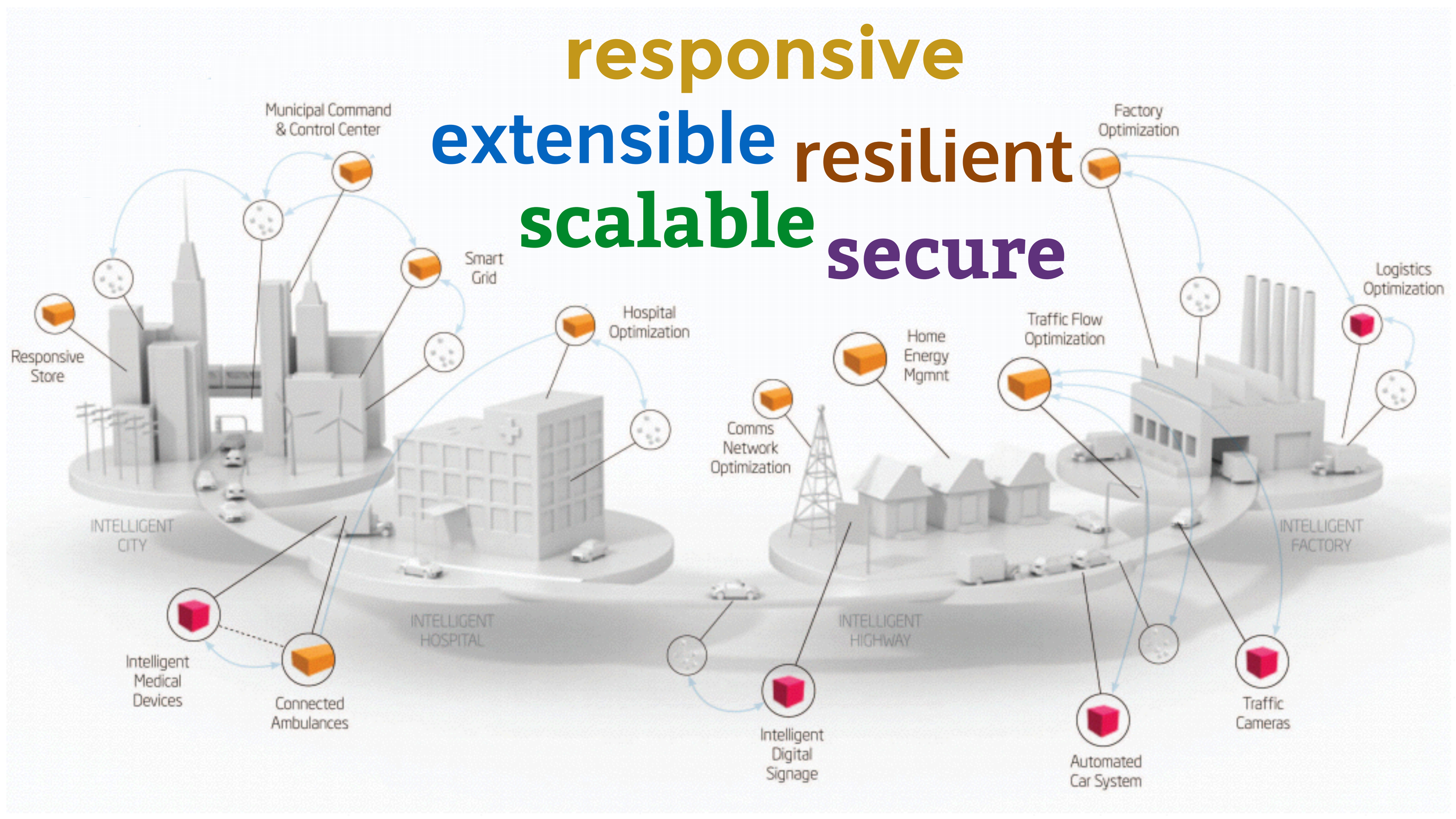
secure



responsive

extensible resilient

scalable secure



Architectural Trends

The Importance Of

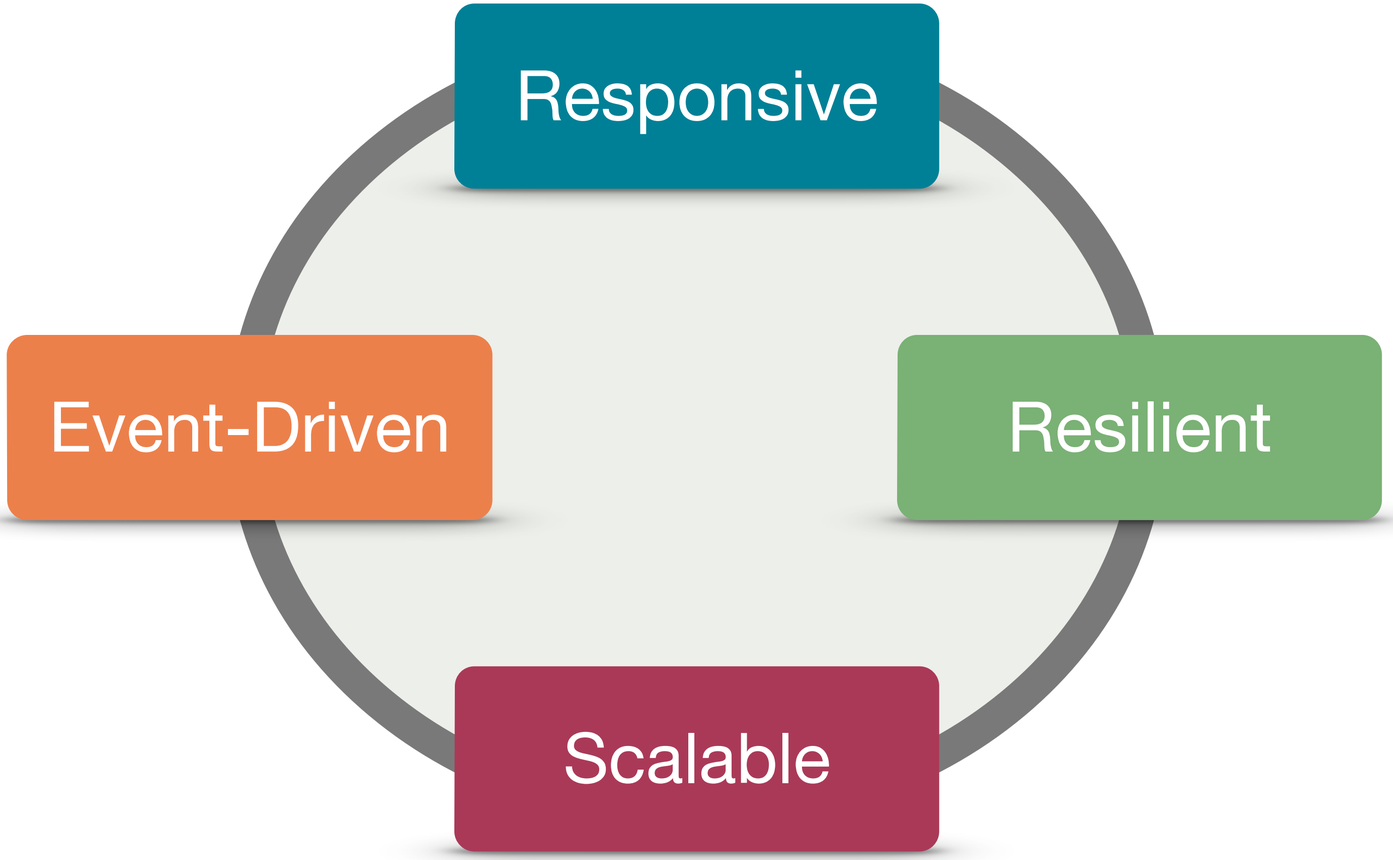
BEING REACTIVE

Up to recently Reactive Systems were mainstream only domains such as Industrial, Military, Telco, Finance and Aerospace

Challenges imposed by the **Internet of Things** and **Big Data Applications** an increasing number of architects and developers is recognising the importance of the techniques and technologies used for building Reactive Systems

REACTIVE MANIFESTO

http://datacentricmanifesto.org



The Importance Of

BEING DATA-CENTRIC

The prevailing application-/service-centric mindset by focusing on the **control-flow** as opposed than the data-flow has lead to design systems that are **hard to extend, scale and make fault-tolerant**

An increasing number of architects has realised that the remedy is to change the main point of attention: **Data is the centre of the universe** — applications are ephemeral

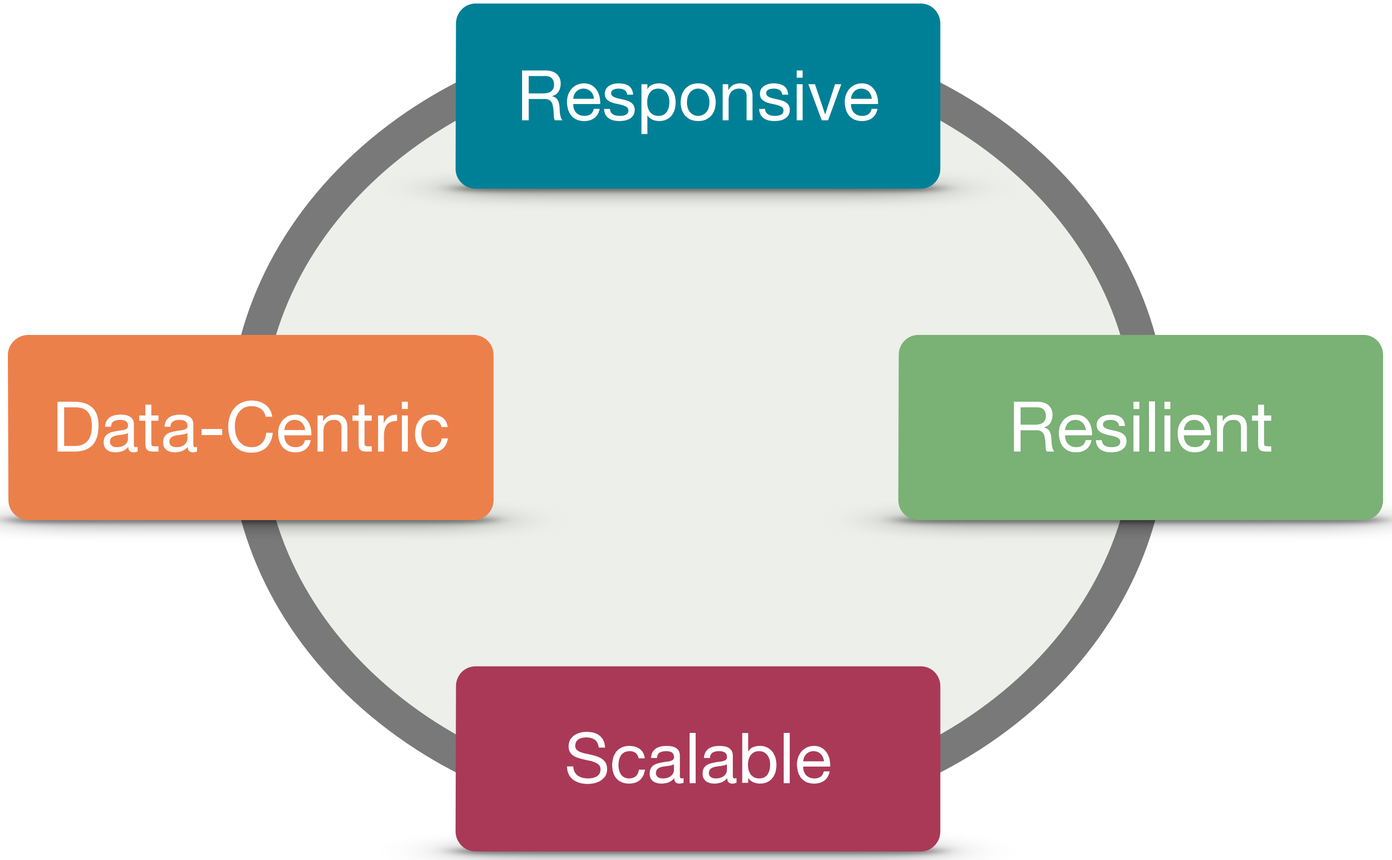
DATA-CENTRIC MANIFESTO

http://datacentricmanifesto.org

The data-centric revolution puts data at the centre of the system

Applications are optional visitors to the data

DATA-CENTRIC REACTIVE MANIFESTO



DATA-CENTRIC REACTIVE MANIFESTO

Data-Centric: data is what matters. Application autonomously and asynchronously transform data

Responsive: Timely react to stimuli

Resilient: Tolerate failures functionally and ideally temporally

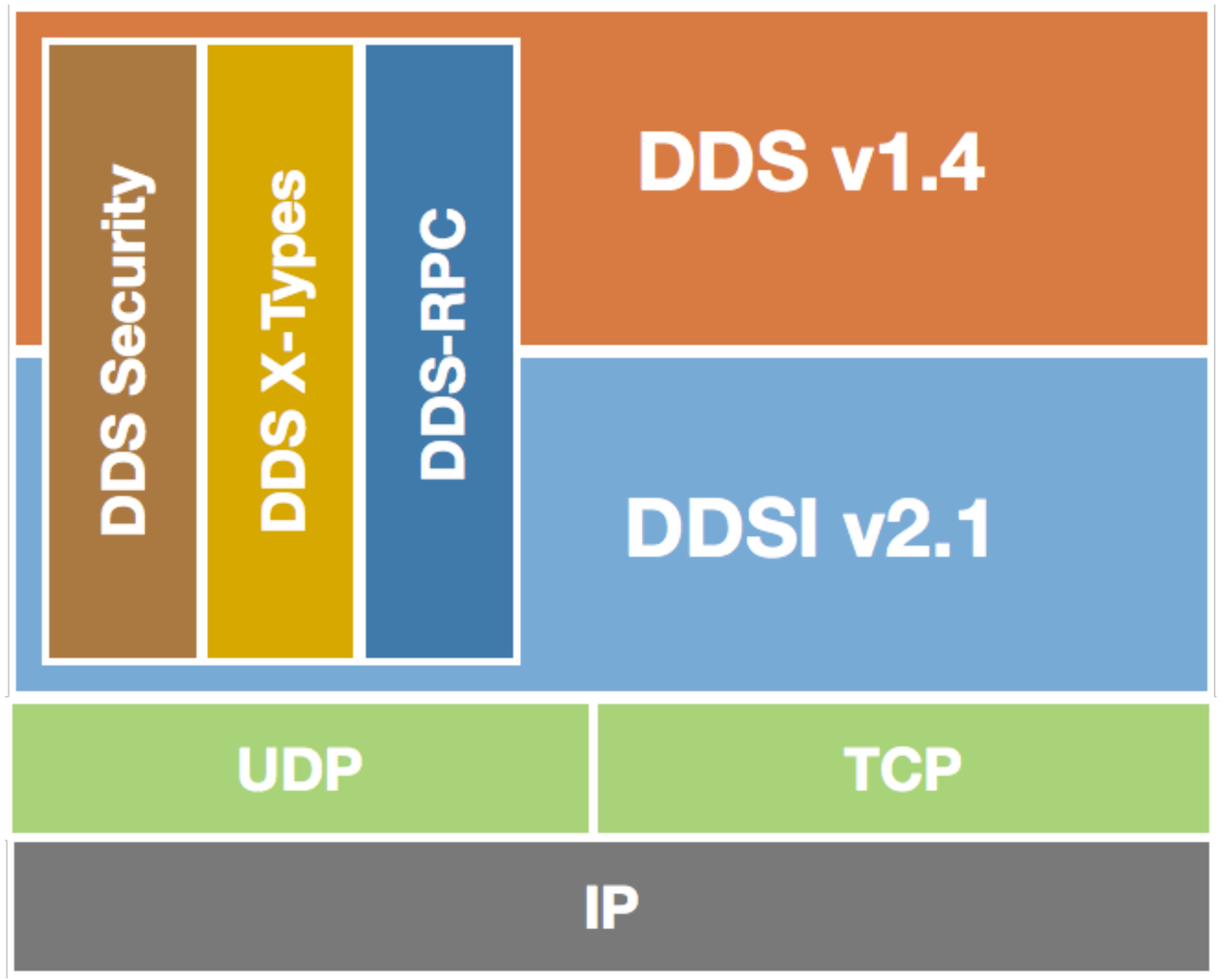
Scalable: Gracefully scale up and down depending on load demands

The DDS Standard

industrial internet® CONSORTIUM

DDS
The IoT Data Sharing Standard

STANDARD



industrial internet® CONSORTIUM

DDS
The IoT Data Sharing Standard

ADOPTION

DISA

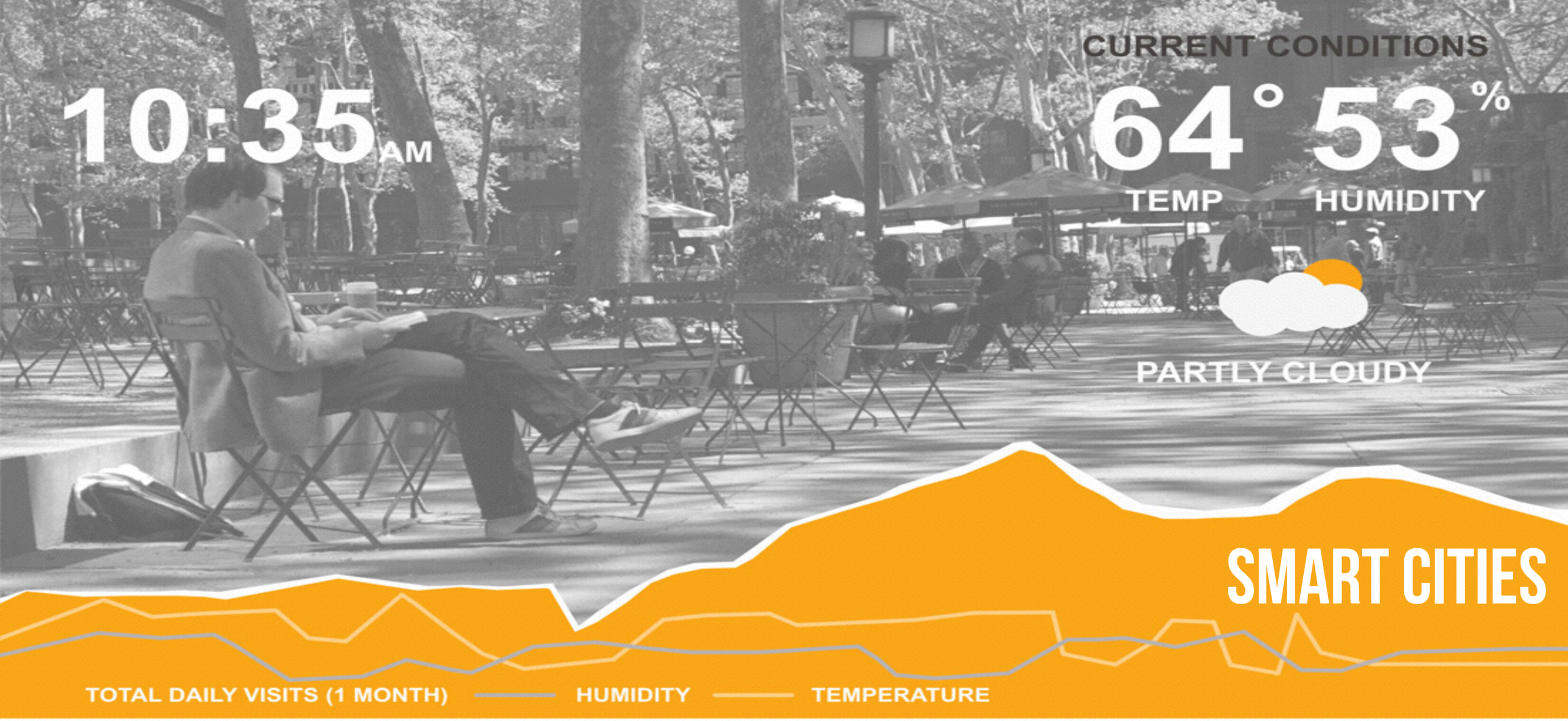
MINISTRY OF DEFENCE

DGA

Federal Ministry
of Defence

QinetiQ


SMART FARMING



TOTAL VISITS TODAY

622 ↑

CURRENT CHAIR OCCUPANCY

65% ↑

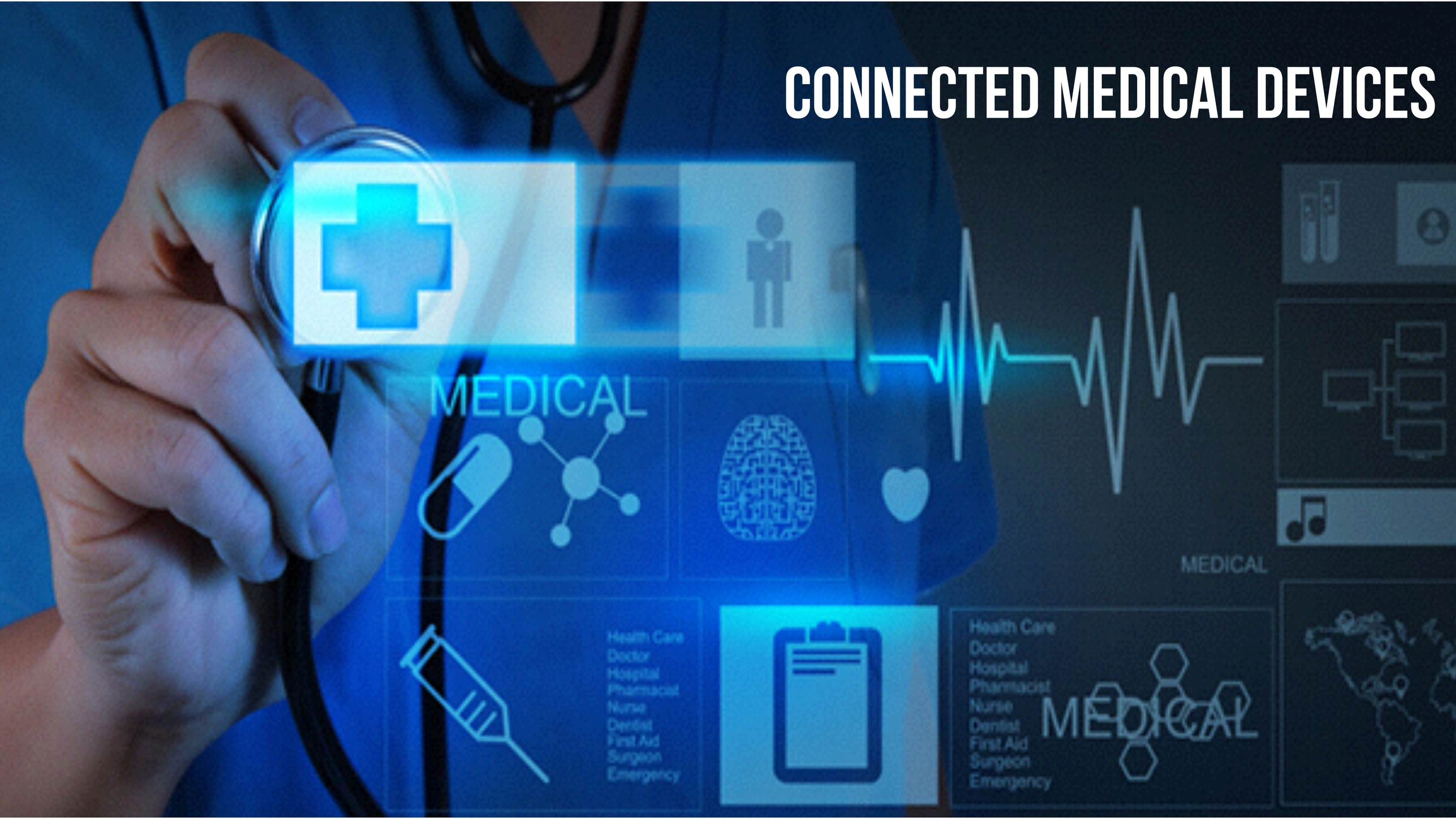
OCCUPANCY @ TABLES

43% ↓

OCCUPANCY @ BENCHES

23% ↑

CONNECTED MEDICAL DEVICES



SMART GRIDS AND POWER GENERATION

EUROPEAN AIR TRAFFIC CONTROL

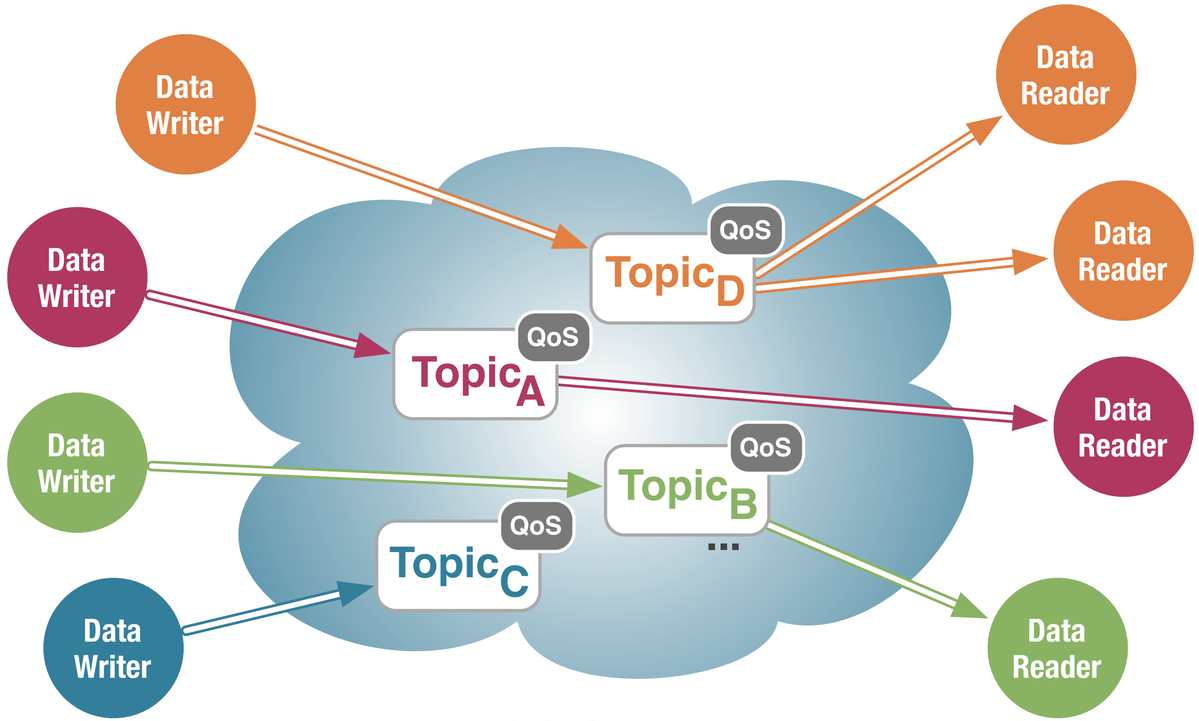
DDS is the standard recommended by Eurocontrol/Eurocae for Pan-European flight data sharing

Reactive Data-Centric Systems with DDS

Step 1:
Data Centricity in DDS

High Level Abstraction

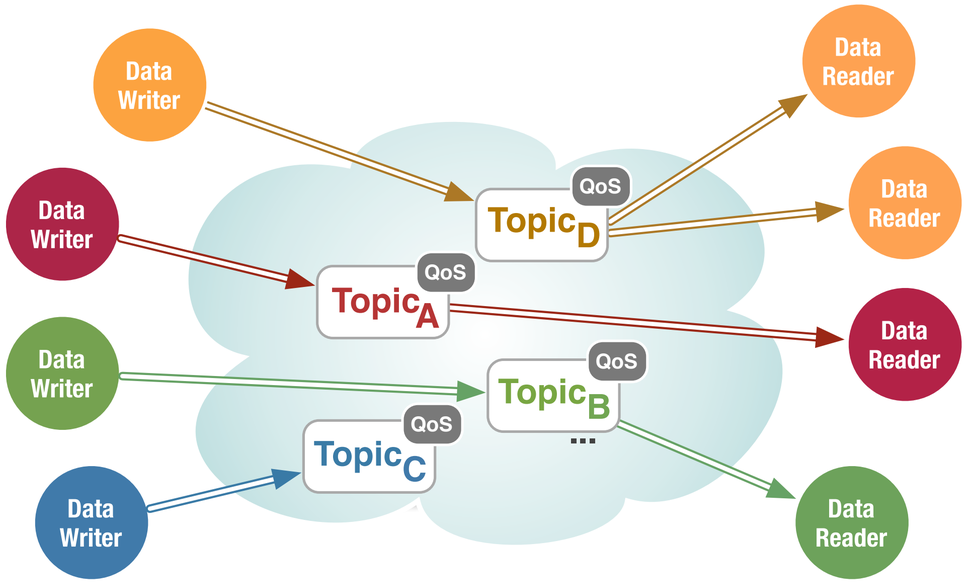
- DDS provides a **Distributed Data Space** abstraction where **applications** can **autonomously** and **asynchronously read** and **write data** enjoying **spatial** and **temporal decoupling**
- Its built-in **dynamic discovery isolates applications** from **network topology** and **connectivity details**
- DDS' Data Space is **completely decentralised**



DDS Global Data Space

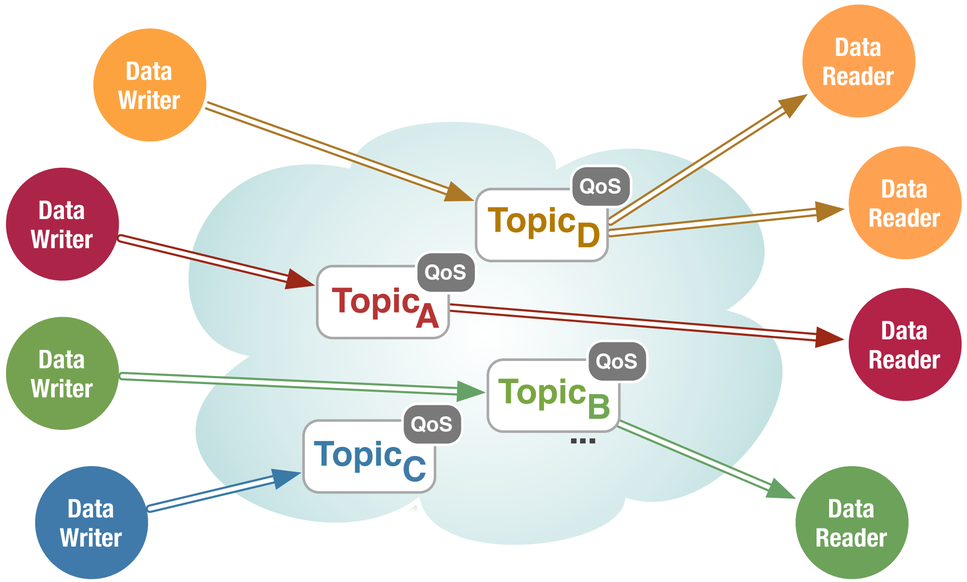
PRISMTECH

Conceptual Model



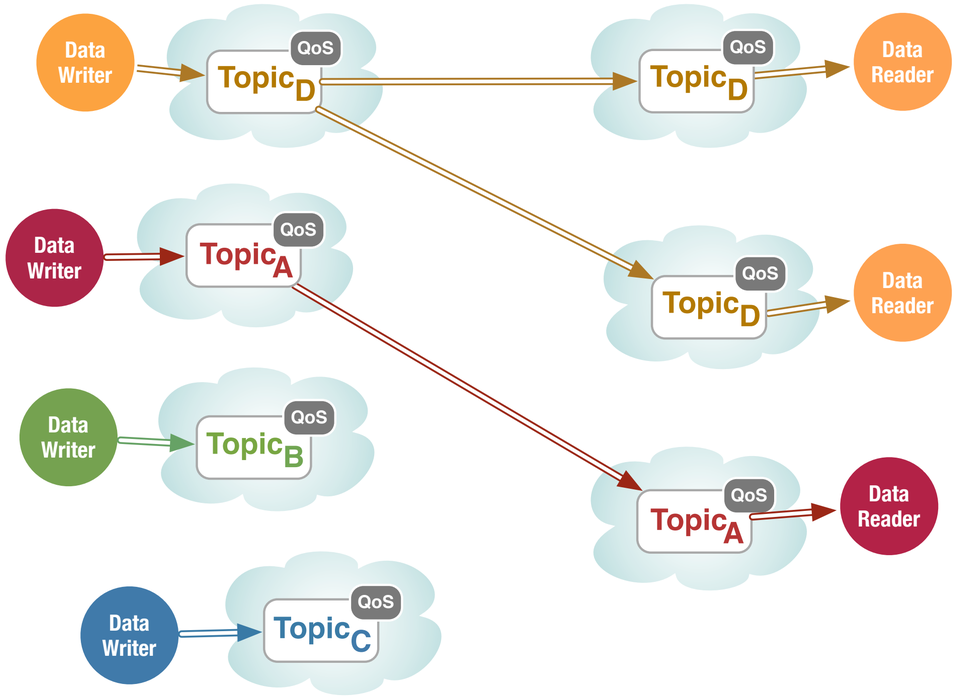
DDS Global Data Space

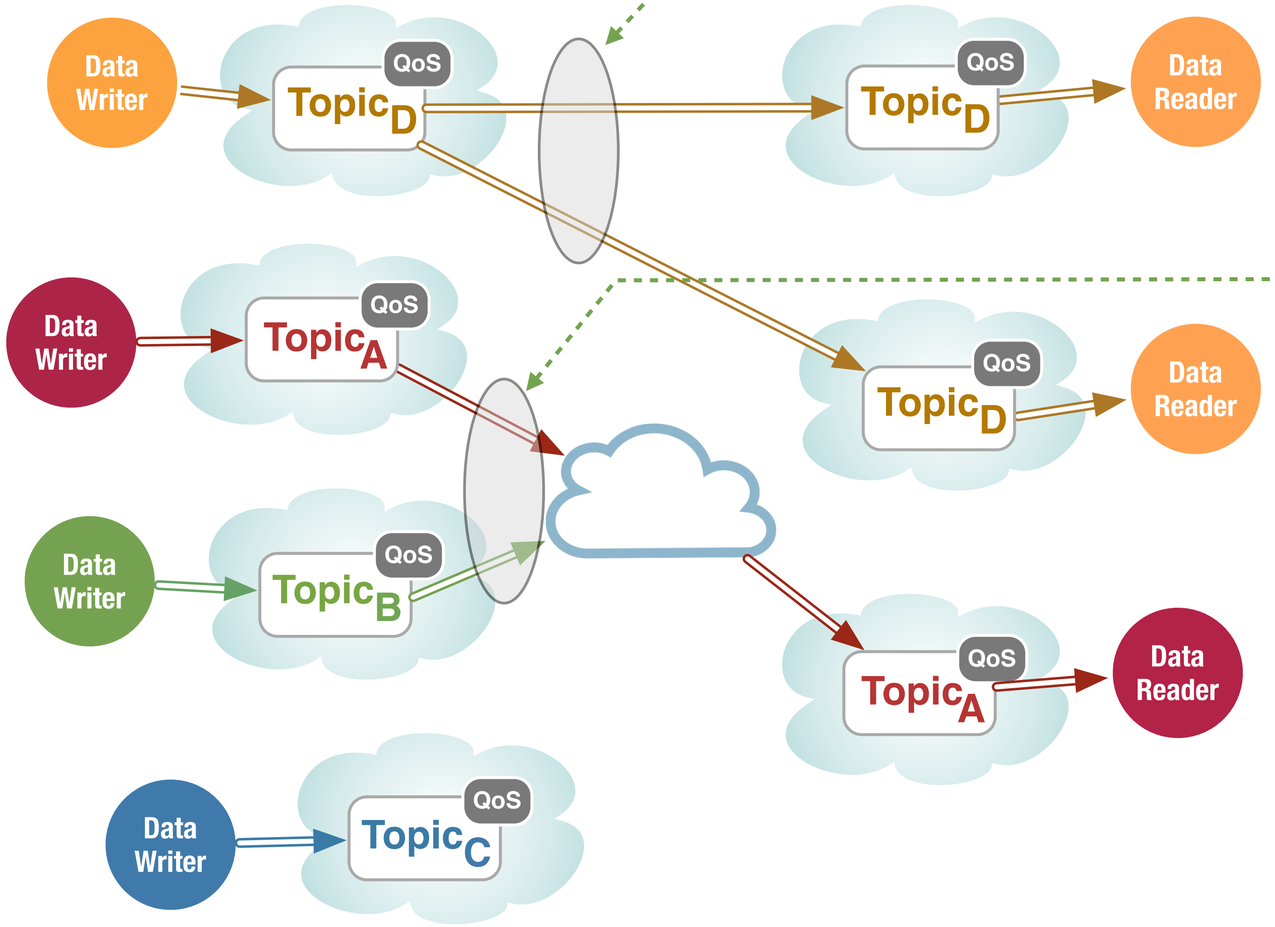
Conceptual Model



DDS Global Data Space

Actual Implementation



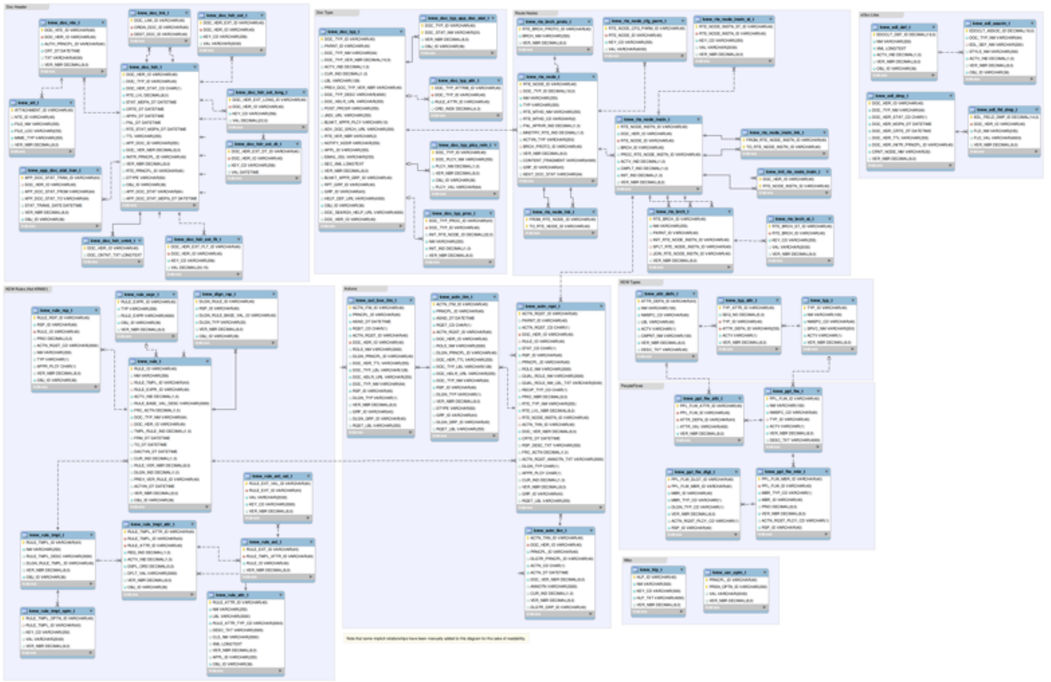


The communication between the DataWriter and matching DataReaders can be peer-to-peer exploiting UDP/IP (Unicast and Multicast)or TCP/IP

The communication between the DataWriter and matching DataReaders can be “brokered” but still exploiting UDP/IP (Unicast and Multicast)or TCP/IP

Data Centricity

- DDS supports the definition of **Data Models**.
- These data models allow to **naturally represent physical and virtual entities characterising the application domain**
- DDS types are extensible and evolvable, thus allowing incremental updates and upgrades

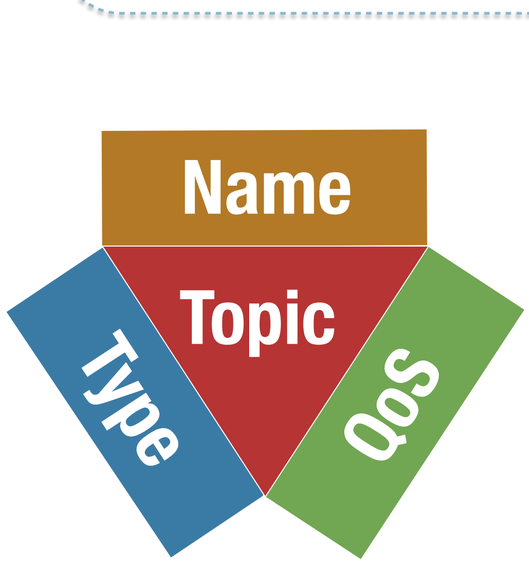


PRISMTECH

Topic

- A Topic defines a **domain-wide** information's class
- A **Topic** is defined by means of a (name, type, qos) tuple, where
  - **name:** identifies the topic within the domain
  - **type:** is the programming language type associated with the topic. Types are extensible and evolvable
  - **qos**: is a collection of policies that express the non-functional properties of this topic, e.g. reliability, persistence, etc.

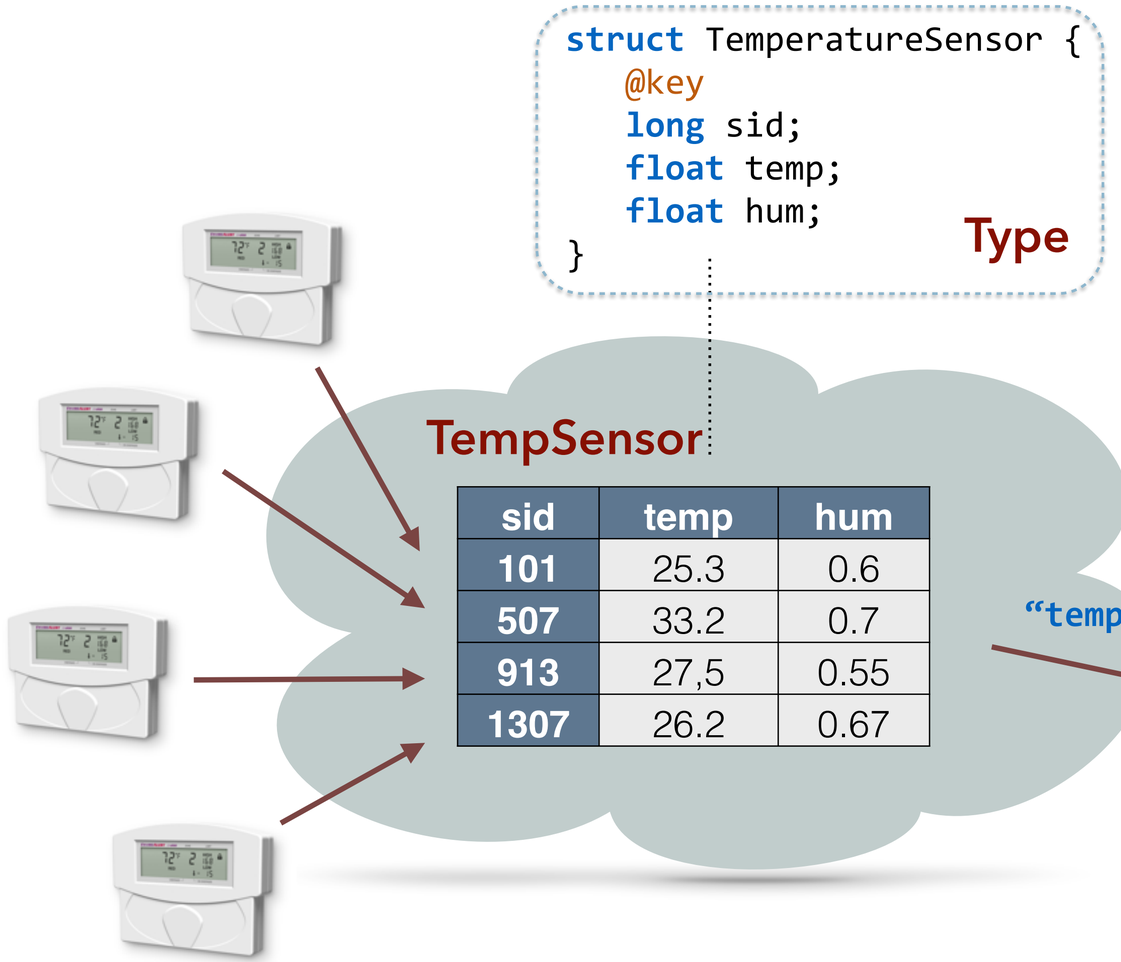
```
struct TemperatureSensor {
    @key
    long sid;
    float temp;
    float hum;
}
```



PRISMTECH

Content Awareness

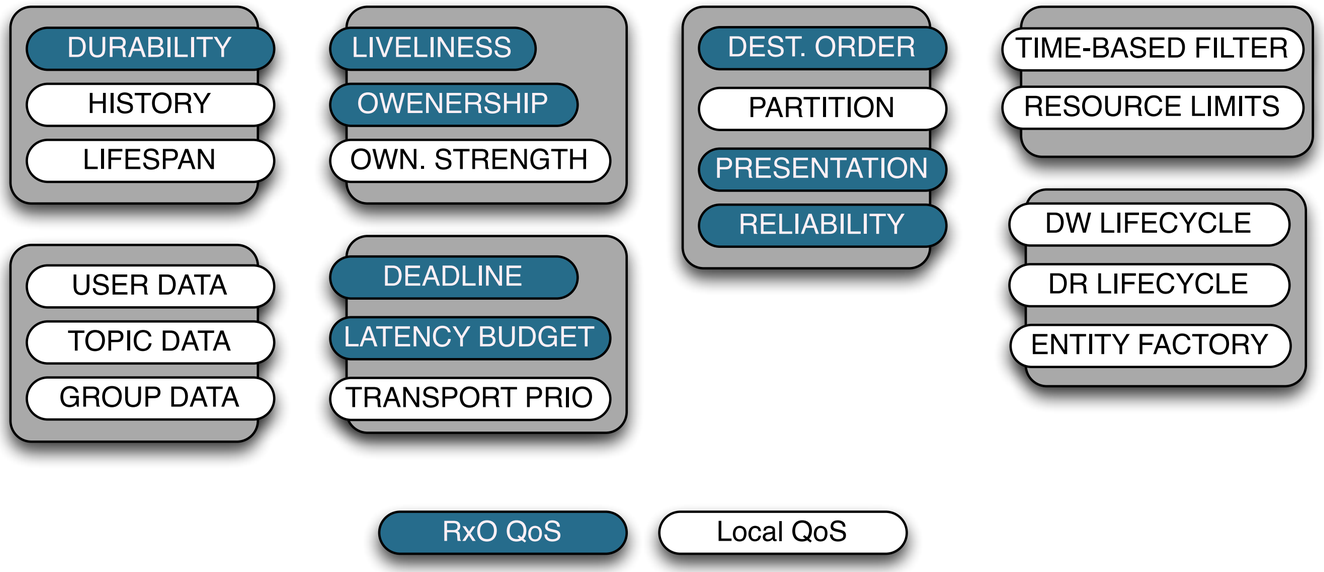
- DDS “**knows**” about application data types and uses this information provide type-safety and content-based routing



PRISMTECH

QoS Policies

- DDS provides a rich set of QoS-Policies to control local as well as end-to-end properties of data sharing
- Some QoS-Policies are matched based on a Request vs. Offered (RxO) Model

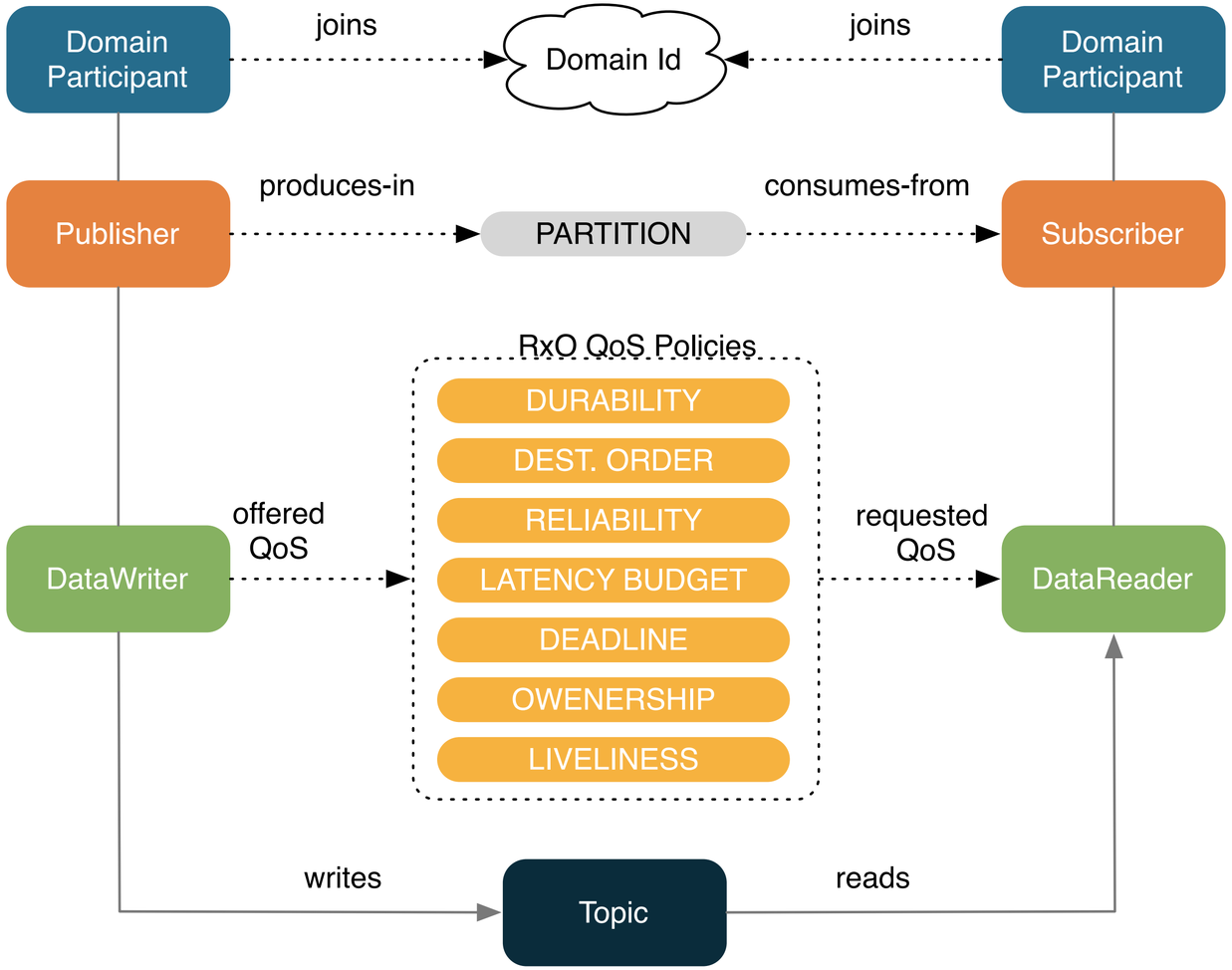


PRISMTECH

Quality of Service

For data to flow from a DataWriter (DW) to one or many DataReader (DR) a few conditions have to apply:

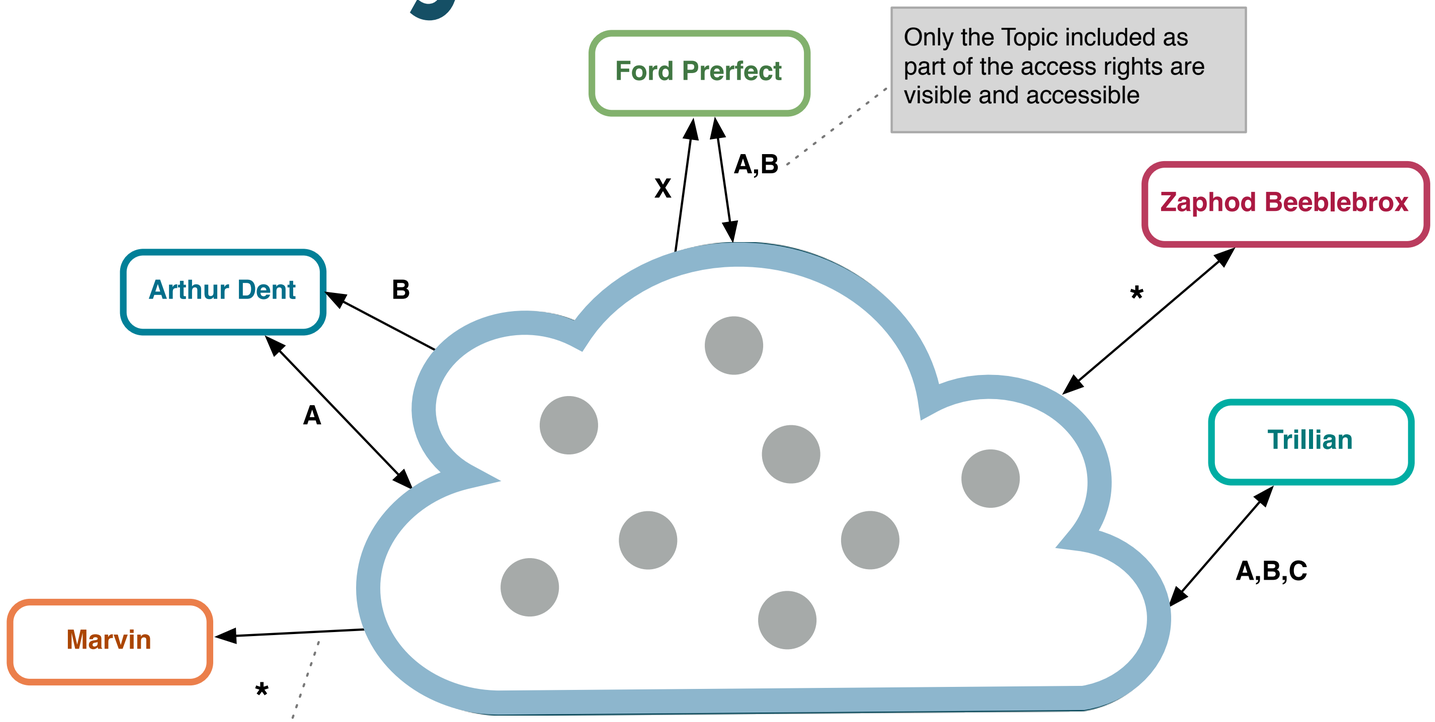
- The DR and DW domain participants have to be in the same domain
- The partition expression of the DR's Subscriber and the DW's Publisher should match (in terms of regular expression match)
- The QoS Policies offered by the DW should exceed or match those requested by the DR



PRISMTECH

Security

- Support for fine grained access control
- Support for Symmetric and Asymmetric Authentication
- Standard Authentication, Access Control, Crypto, and Logging plug-in API



Identity	Access Rights
Arthur Dent	A(r,w), B(r)
Ford Prerfect	A(r,w), B(r,w), X(r)
Zaphod Beeblebrox	*(r,w)
Trillian	A(r,w), B(r,w), C(r,w)
Marvin	*(r)

PRISMTECH

Platform Independent

- DDS is independent from the
  - Programming language,
  - Operating System
  - HW architecture







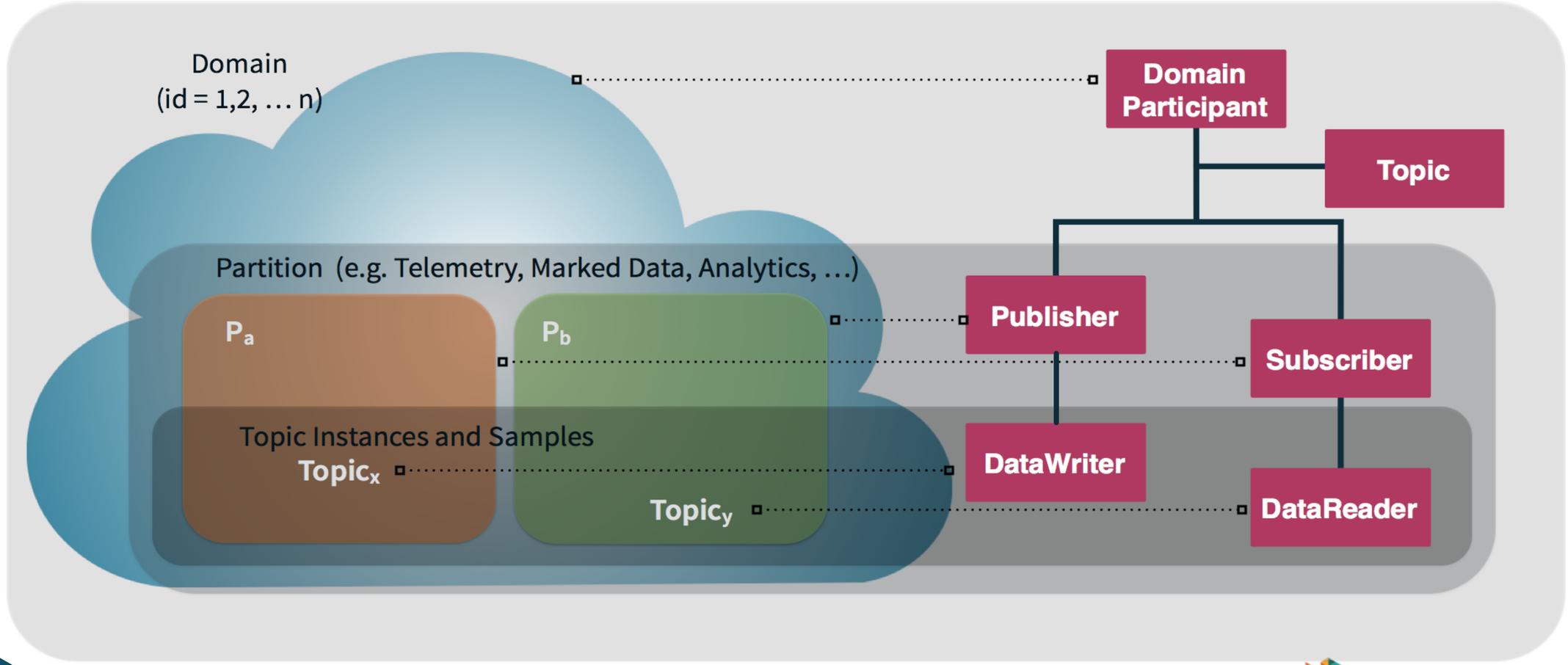







Your First DDS App!

Anatomy of a DDS Application



PRISMTECH

Writing Data in C++

```
#include <dds.hpp>

int main(int, char**) {

  DomainParticipant dp(0);
  Topic<Meter> topic("SmartMeter");
  Publisher pub(dp);
  DataWriter<Meter> dw(pub, topic);

  while (!done) {
     auto value = readMeter()
     dw.write(value);
     std::this_thread::sleep_for(SAMPLING_PERIOD);
  }

  return 0;
}
```

```
enum UtilityKind {
    ELECTRICITY,
    GAS,
    WATER
};

struct Meter {
   string sn;
   UtilityKind utility;
   float reading;
   float error;
};
#pragma keylist Meter sn
```


PRISMTECH

Reading Data in C++

```
#include <dds.hpp>

int main(int, char**) {

  DomainParticipant dp(0);
  Topic<Meter> topic("SmartMeter");
  Subscriber sub(dp);
  DataReader<Meter> dr(dp, topic);

  LambdaDataReaderListener<DataReader<Meter>> lst;
  lst.data_available = [](DataReader<Meter>& dr) {
       auto samples = data.read();
       std::for_each(samples.begin(), samples.end(), [](Sample<Meter>& sample) {
         std::cout << sample.data() << std::endl;
       }
   }
   dr.listener(lst);
   // Print incoming data up to when the user does a Ctrl-C
   std::this_thread::join();
   return 0;
}
```

```
enum UtilityKind {
    ELECTRICITY,
    GAS,
    WATER
};

struct Meter {
    string sn;
    UtilityKind utility;
    float reading;
    float error;
};
#pragma keylist Meter sn
```


PRISMTECH

Writing Data in Python

```python
import dds
import time

if __name__ == '__main__':
    topic = dds.Topic("SmartMeter", "Meter")
    dw = dds.Writer(topic)

    while True:
        m = readMeter()
        dw.write(m)
        time.sleep(0.1)
```

```
enum UtilityKind {
    ELECTRICITY,
    GAS,
    WATER
};

struct Meter {
    string sn;
    UtilityKind utility;
    float reading;
    float error;
};
#pragma keylist Meter sn
```

Reading Data in Python

```python
import dds
import sys

def readData(dr):
    samples = dds.range(dr.read())
    for s in samples:
        sys.stdout.write(str(s.getData()))

if __name__ == '__main__':
    t = dds.Topic("SmartMeter", "Meter")
    dr = dds.Reader(t)
    dr.onDataAvailable = readData
```

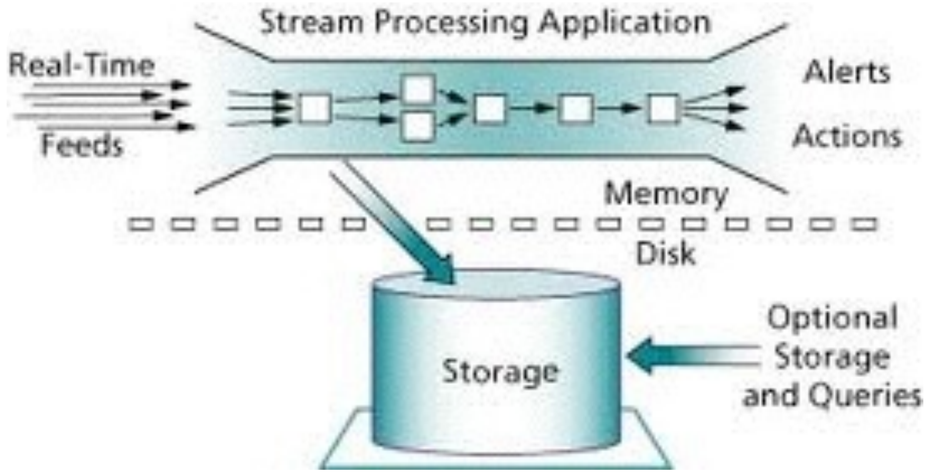
```
enum UtilityKind {
    ELECTRICITY,
    GAS,
    WATER
};

struct Meter {
    string sn;
    UtilityKind utility;
    float reading;
    float error;
};
#pragma keylist Meter sn
```

Step 2:
Reactive Architectures with DDS

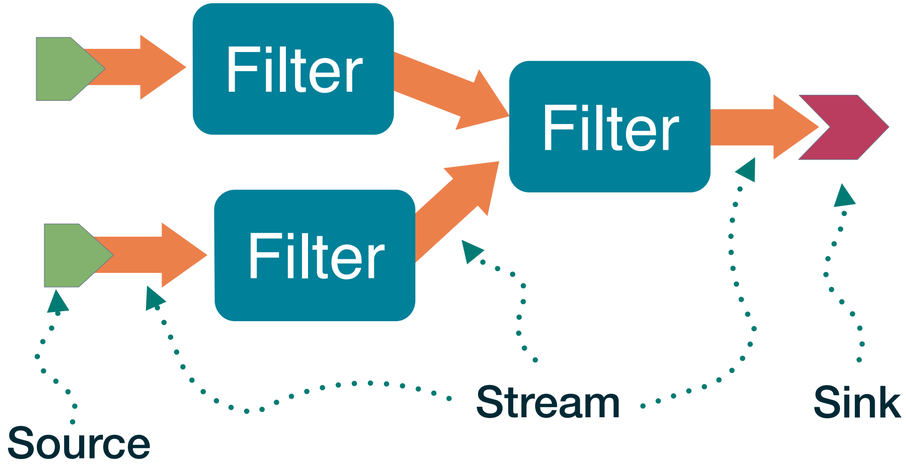
STREAM PROCESSING

Stream Processing is the commonly adopted architectural pattern for building reactive systems



Stream Processing

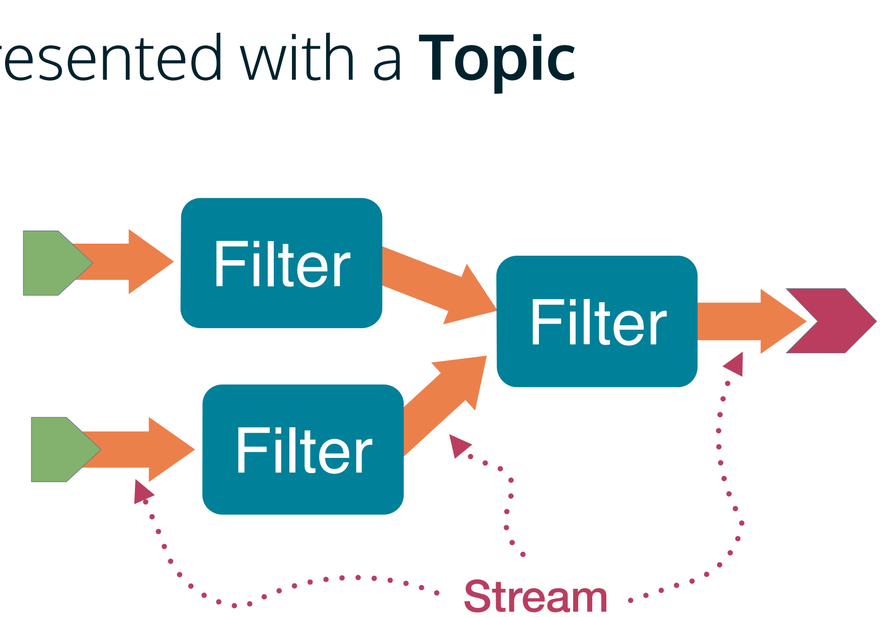
- **Stream Processing Systems** are typically modelled as **collection of concurrent modules communicating via typed data channels**
- Modules usually play one of the following roles:
  - **Sources**: Injecting data into the System
  - **Filters/Actors**: Performing some computation over sources
  - **Sinks**: Consuming the data produced by the system



Stream Processing with DDS

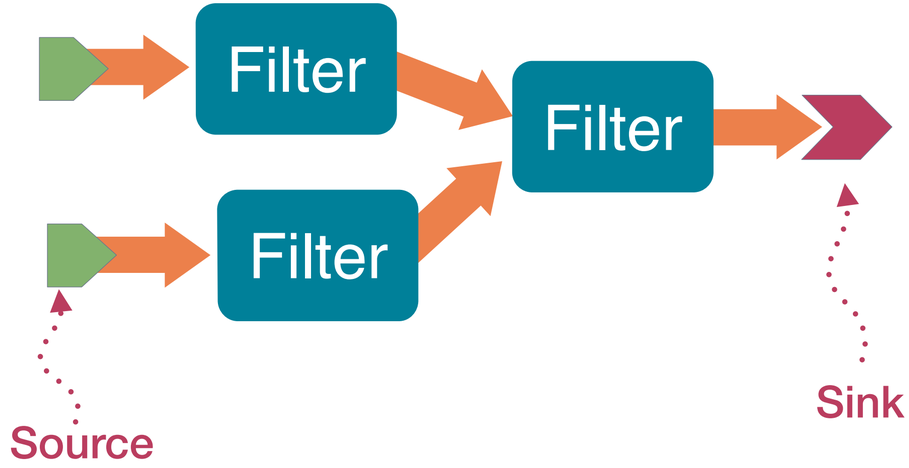
Streams in DDS

- A **stream** represents a potentially infinite sequence of data samples of a given type T
- As such, in DDS, a stream can be naturally represented with a **Topic**
- Example:
  - Topic<PressureType>
  - Topic<MarketData>



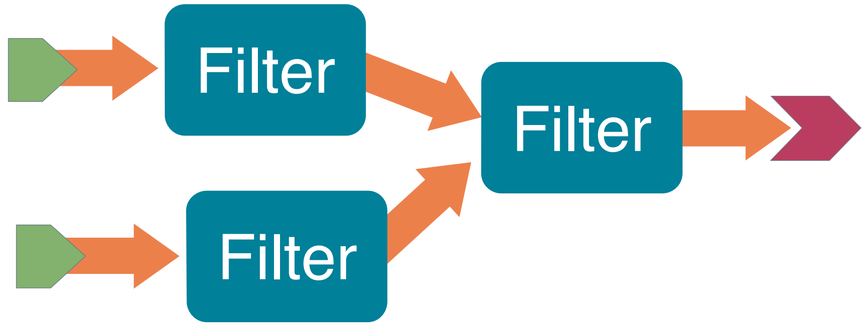
Source and Sinks in DDS

- **Sources** in DDS are mapped to **DataWriters** for a given Topic
- **Sinks** in DDS are mapped to **DataReaders** for a given Topic



Filters in DDS

- Filters implement bit and pieces of the application business logic
- Each Filter has one or more input stream and one or more output streams
- Obviously, a Filter consumes data from an input stream through a DDS data reader and pushed data into an output stream through a DDS data writer



DDS Architectural Benefits

Loose Coupling

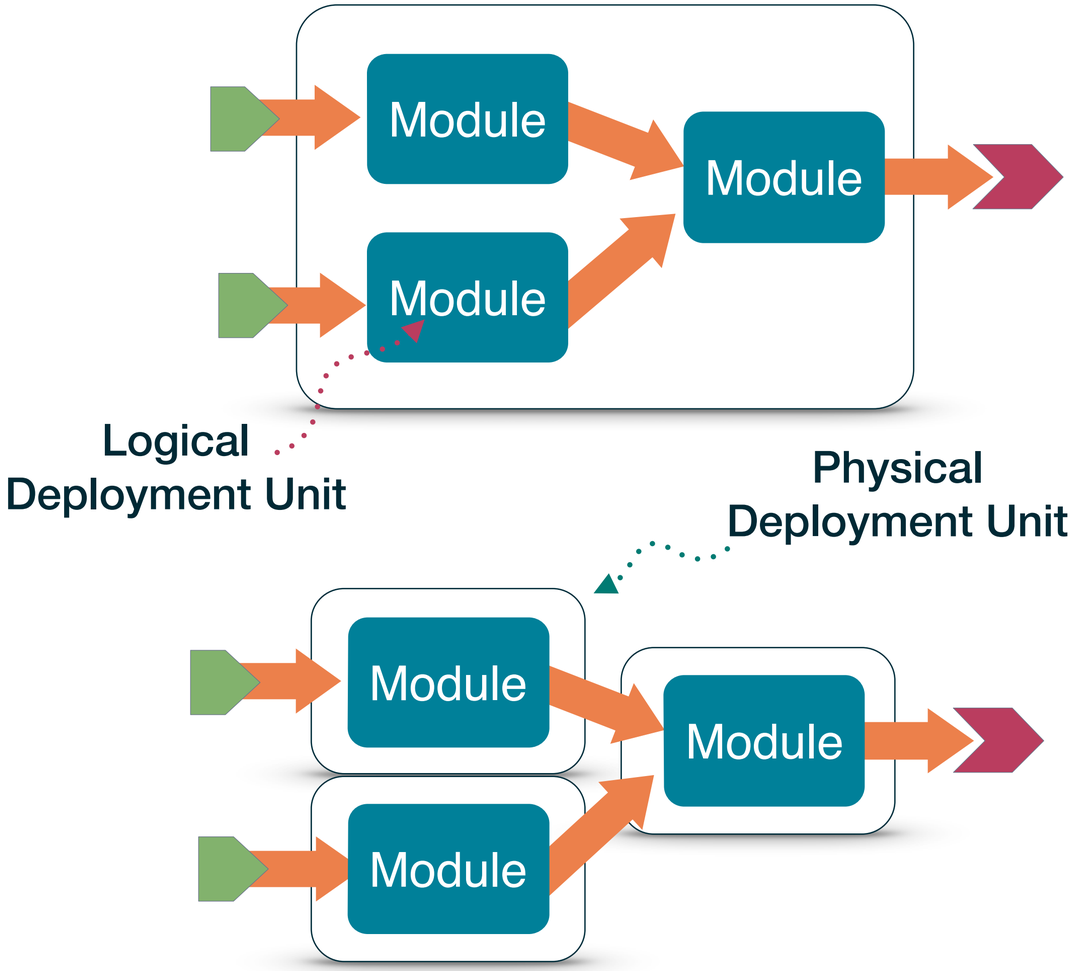
Stream Processing Architectures in general and DDS-based architecture in particular promote loose coupling

- **Anonymous Communication =>** No Spatial Coupling
- **Non-Blocking and Asynchronous Interaction** => No control or temporal coupling

In DDS the only thing that matters is the kind of information a module is interested in consuming or producing — who is producing the data or when the data has been produced is not relevant

Location Transparency

- **Location Transparency** is an important architectural property as it makes it possible to completely decouple the logical decomposition of a system from its actual physical deployment
- DDS **Automatic Discovery** brings location transparency to the next level by enabling systems that are zero-conf, self-forming and self-healing

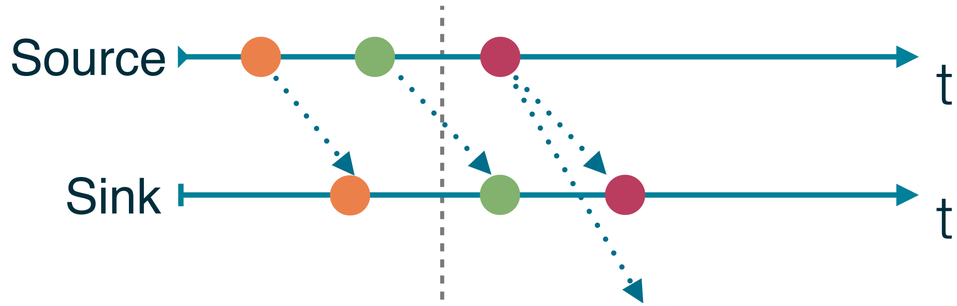


PRISMTECH

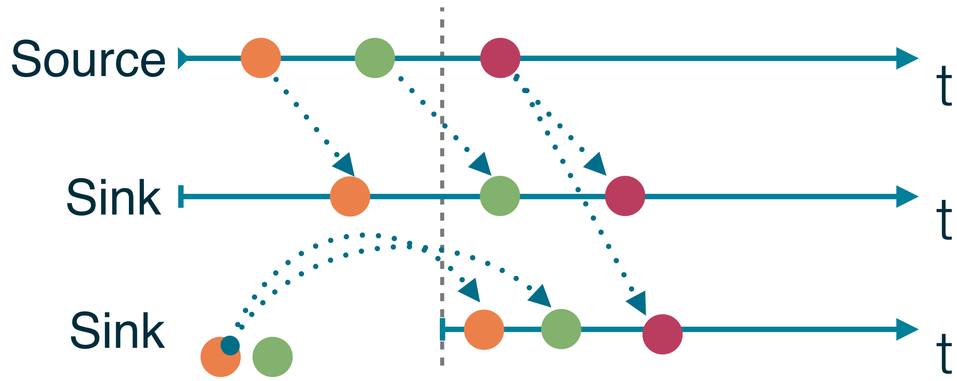
Temporal Decoupling

- DDS provides full control over **temporal decoupling** by means of its **Durability** Policy
- As such the data produced by a **Source** can be:
  - **Volatile**: available for those that are around at the time at production
  - **Transient:** available as far as the system/source is running
  - **Durable:** available forever

Volatile Durability



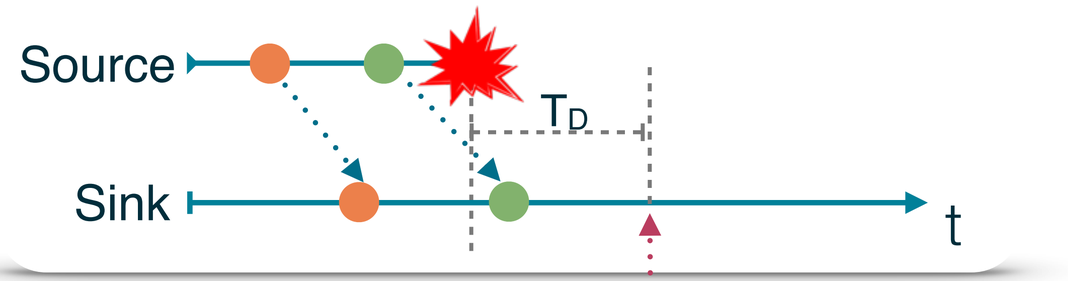
Transient Durability

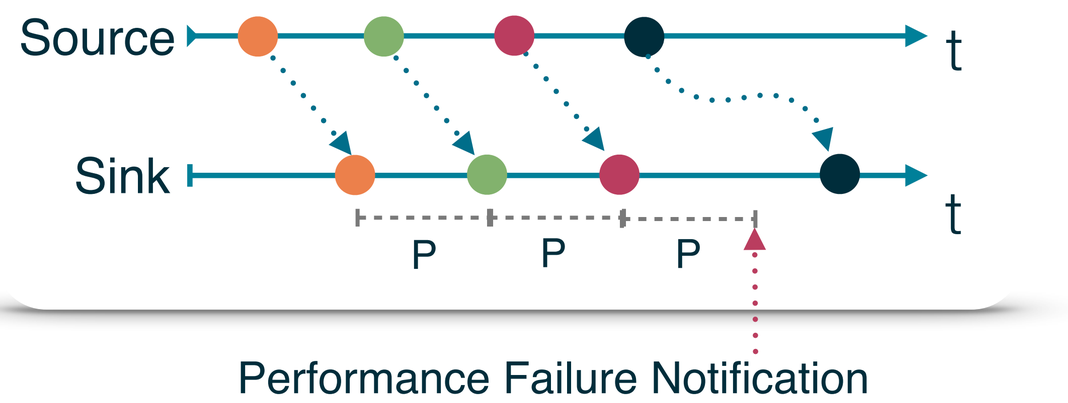


PRISMTECH

Failure Detection

- DDS provides mechanism for detecting traditional faults as well as performance failures
- The Fault-Detection mechanism is controlled by means of the DDS Liveliness policy
- Performance Failures can be detected using the Deadline Policy which allows to receive notification when data is not received within the expected delays

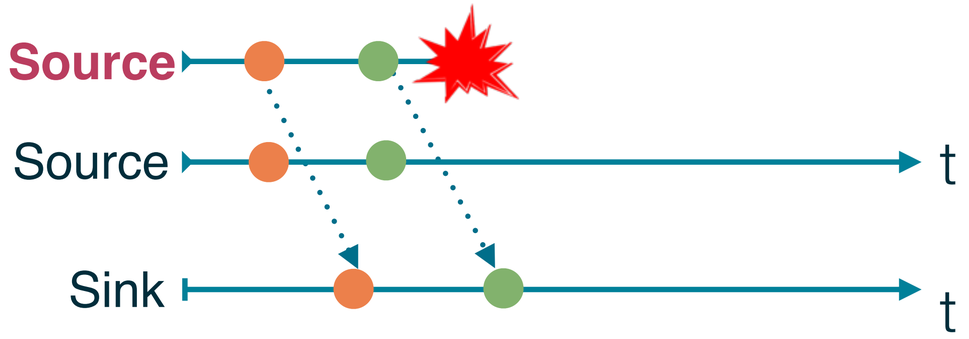




PRISMTECH

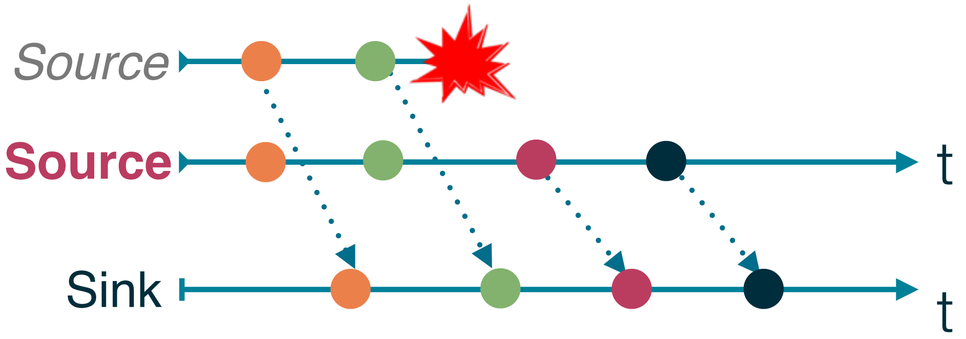
Fault-Masking

- DDS provides a built-in fault-masking mechanism that allow to replicate **Sources** and transparently switch over when a failure occurs
- At any point in time the “active” source is the one with the highest strength. Where the strength is an integer parameter controller by the user

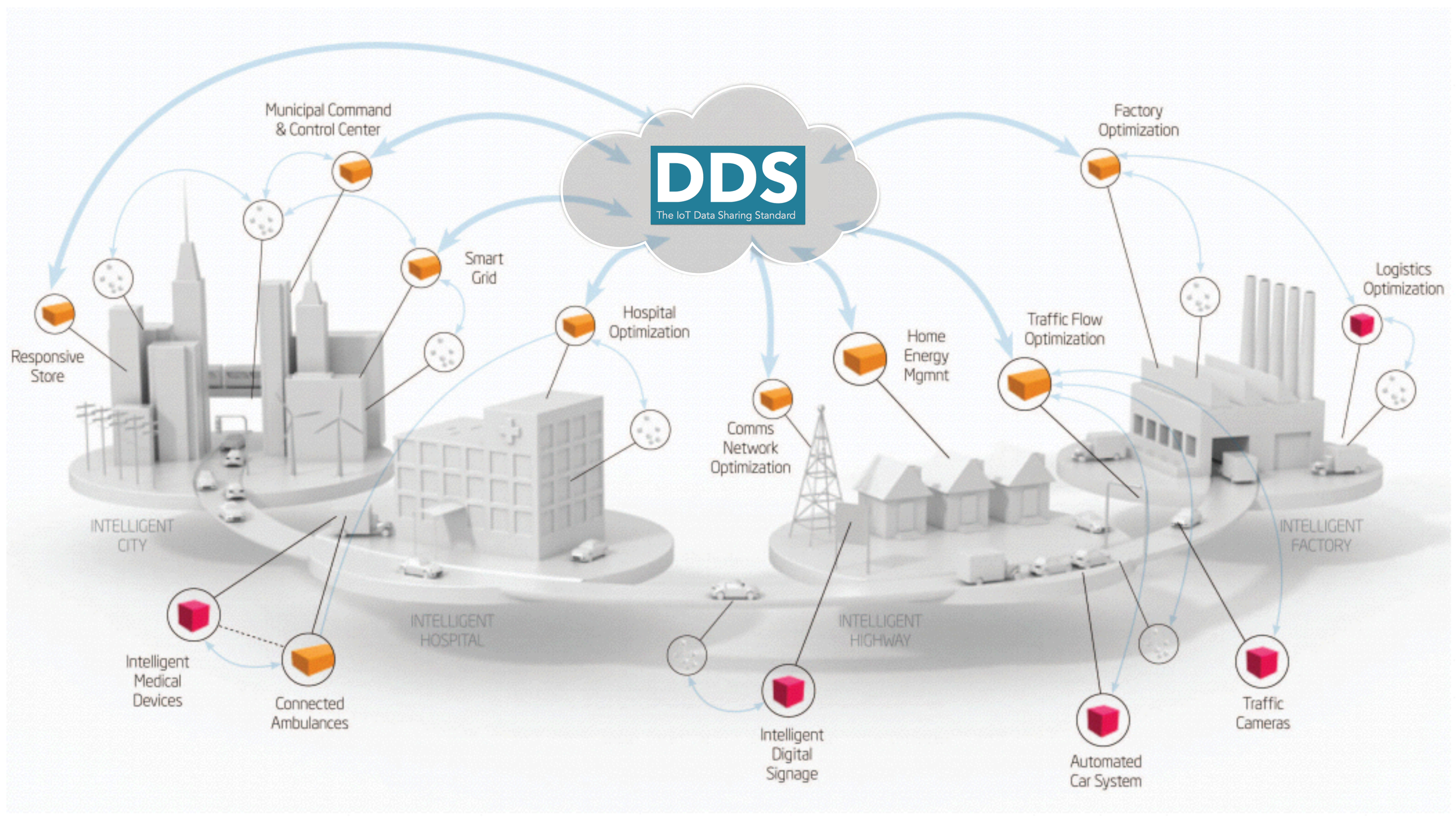


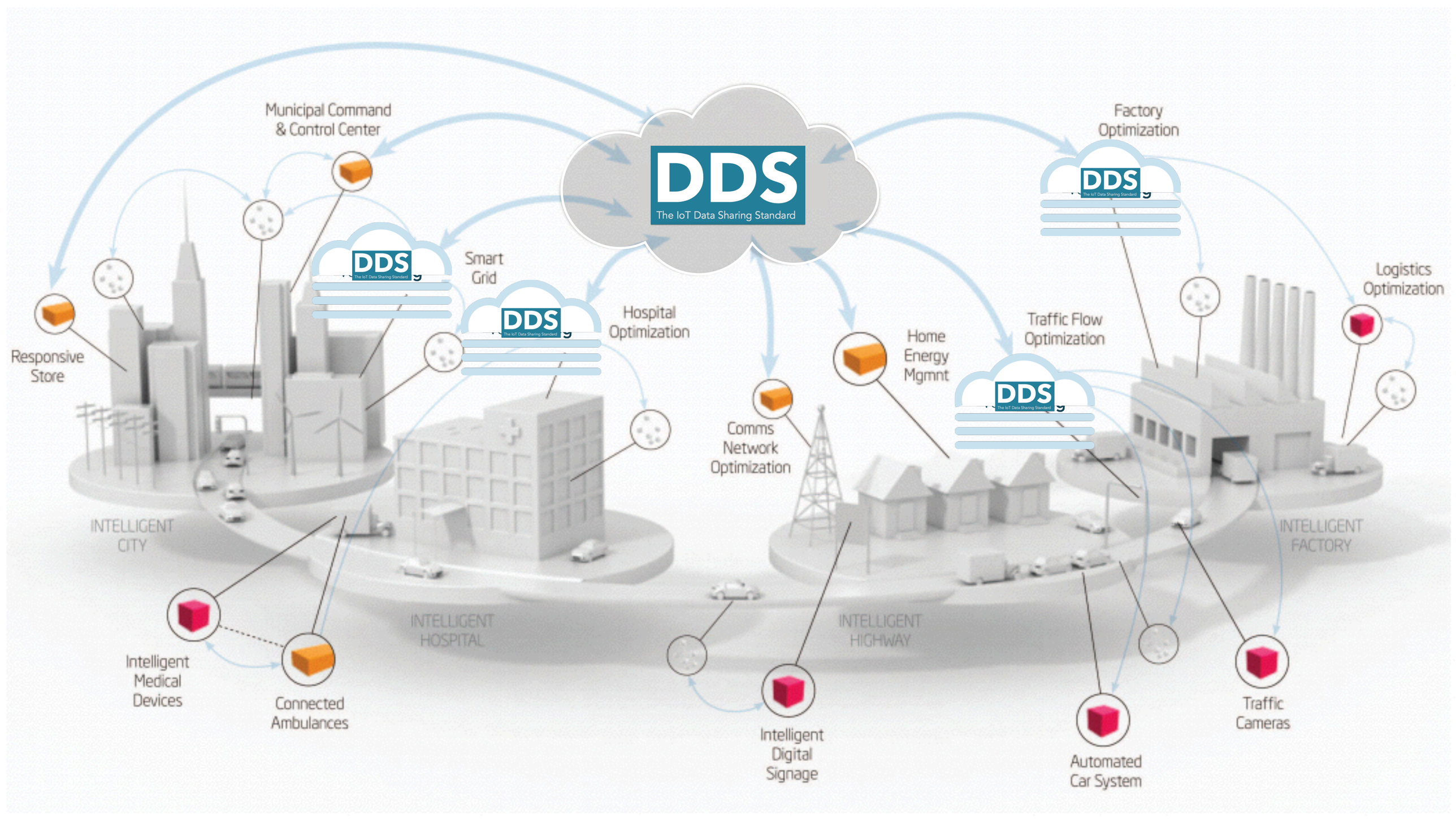
Fault-Masking

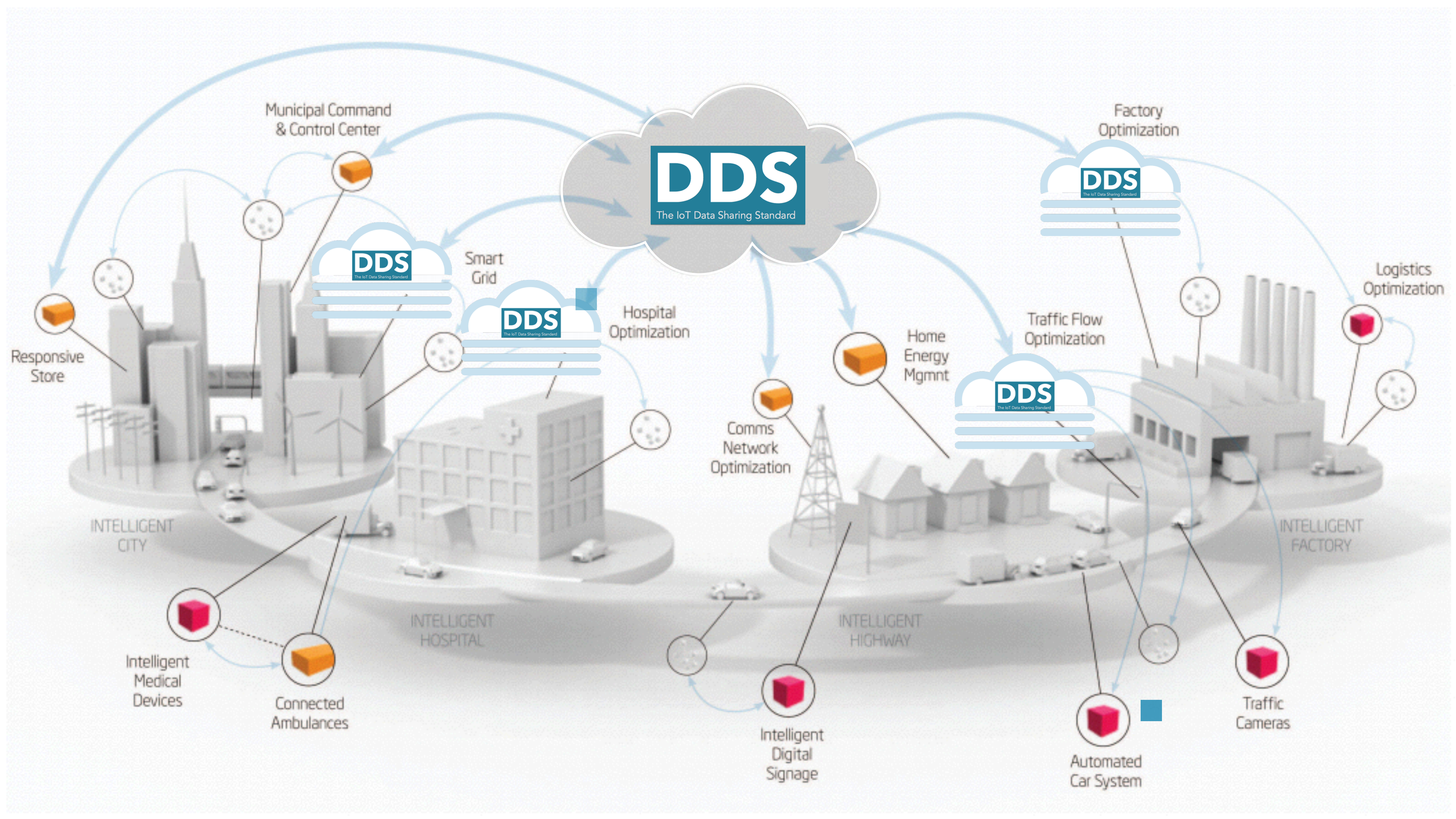
- DDS provides a built-in fault-masking mechanism that allow to replicate **Sources** and transparently switch over when a failure occurs
- At any point in time the “active” source is the one with the highest strength. Where the strength is an integer parameter controller by the user



Putting All Together


DDS
The IoT Data Sharing Standard
Municipal Command & Control Center
Smart Grid
Responsive Store
Hospital Optimization
Comms Network Optimization
Home Energy Mgmnt
Factory Optimization
Traffic Flow Optimization
Logistics Optimization
INTELLIGENT CITY
INTELLIGENT HOSPITAL
INTELLIGENT HIGHWAY
INTELLIGENT FACTORY
Intelligent Medical Devices
Connected Ambulances
Intelligent Digital Signage
Automated Car System
Traffic Cameras


DDS
The IoT Data Sharing Standard
Municipal Command & Control Center
Factory Optimization
Smart Grid
Hospital Optimization
Logistics Optimization
Responsive Store
Traffic Flow Optimization
Home Energy Mgmnt
Comms Network Optimization
INTELLIGENT CITY
INTELLIGENT HOSPITAL
INTELLIGENT HIGHWAY
INTELLIGENT FACTORY
Intelligent Medical Devices
Connected Ambulances
Intelligent Digital Signage
Automated Car System
Traffic Cameras


DDS
The IoT Data Sharing Standard
Municipal Command & Control Center
Factory Optimization
Smart Grid
Hospital Optimization
Logistics Optimization
Responsive Store
Home Energy Mgmnt
Traffic Flow Optimization
Comms Network Optimization
INTELLIGENT CITY
INTELLIGENT HOSPITAL
INTELLIGENT HIGHWAY
INTELLIGENT FACTORY
Intelligent Medical Devices
Connected Ambulances
Intelligent Digital Signage
Automated Car System
Traffic Cameras

DDS

DDS is a standard technology for ubiquitous, interoperable, secure, platform independent, and real-time data sharing across network connected devices

DDS provides a the ideal platform for architecting and building Data-Centric Reactive Systems