

# Middleware & Applications

IEEE RFID 2026 Tutorial presented by **Jeffrey Dungen**



Why can't the [RFID HARDWARE] just talk to the [SOFTWARE]?

Well—well look. I already told you: I deal with the [SOFTWARE] so the [HARDWARE] engineers don't have to. *I HAVE PEOPLE SKILLS*; I am good at dealing with people. Can't you understand that? What the hell is wrong with you people?

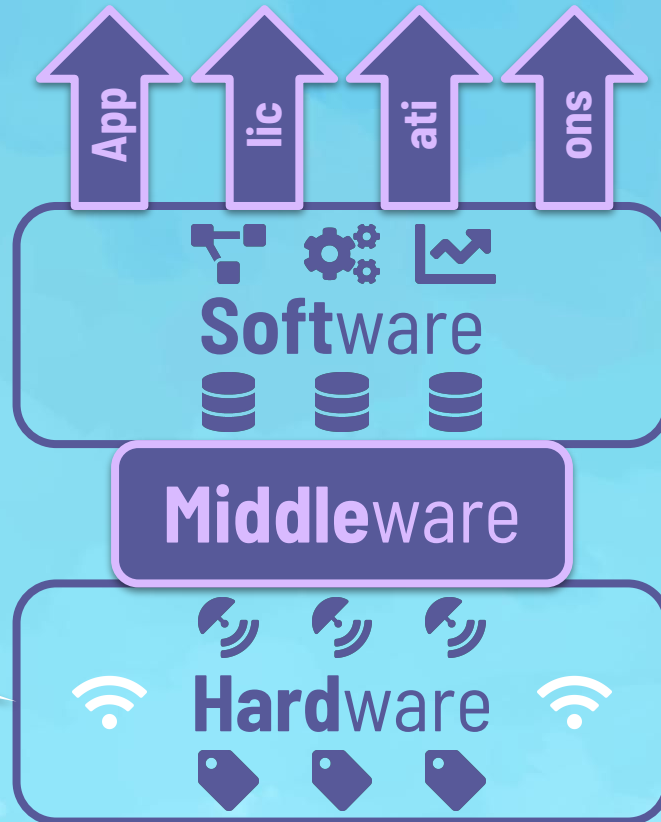


Source: Office Space (1999)

Middleware



# Hard-Middle-Soft, then Applied...



People skills?



Remember everything that **Dobkin, Durgin** and **Degnan** talked about?

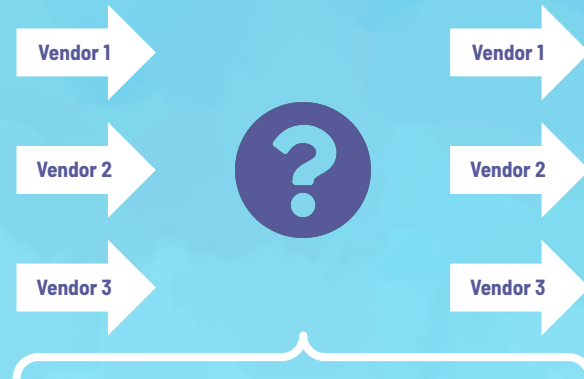
# Why do we need Middleware?

Middleware is like a universal *translator* between hardware & software.



Standards observed over-the-air:

- RAIN RFID
- Bluetooth Low Energy
- etc.



Vendor-specific interfaces...



**Frustration** for integrators and users developing applications *without* middleware.



# Appreciate the app developer

The people developing the **application** layer:

- **Don't care** about how the hardware works.\*
- **Don't care** about how the protocols work.\*
- **Do care** about having **usable data** in a format they understand.

\* sorry Dan, Greg and Brian!

Let's tidy up the messy middle and make them happy!



# Part 1:

## The Hardware Side of Things



# Hardware and Protocols

Here's my identifier  
and perhaps some sensor data!



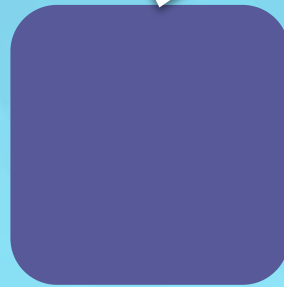
Tag/Beacon/Device



Protocol



I just decoded a tag  
and here's some metadata!



Reader/Gateway/AP



# Protocols: RAIN RFID

There's a standard for the wireless interface:

- **EPC UHF Gen2 Air Interface Protocol** (ISO/IEC 18000-63)\*
- Enjoy reading: <https://ref.gs1.org/standards/gen2/>

\* presented in Dan's tutorial

There's a standard for identifiers and data:

- **EPC Tag Data Standard** (TDS)\*
- Enjoy reading: <https://ref.gs1.org/standards/tds/>

\* we'll come back to this later



# Protocols: Bluetooth Low Energy

There's a standard for the wireless interface and more:

→ **Bluetooth Core Specification** (currently v6.2)\*

→ Enjoy reading: <https://www.bluetooth.com/specifications/specs/core-specification/>

\* it's only 3897 pages long!

There are complementary specifications for identifiers and data:

→ Read them all: <https://www.bluetooth.com/specifications/specs/>

# Protocols: Bluetooth Low Energy

Rather read an IEEE RFID tutorial than a standard specification!?!



<https://2017.ieeerfid.org/files/2017/01/IEEE-RFID-2017-BLE-as-Active-RFID.pdf>

# Protocols: but wait, there's more...

Other wireless protocols used around the world include:

- **UWB** (Ultra WideBand): lots of different flavours 🤔
- **NFC** (Near-Field Communication): short range, broad adoption in mobile
- **EnOcean** (ISO/IEC 14543-3-10): wireless sensor networks
- and many others...

Do they each have their own unique specification? *Of course!*

# Readers decode RAIN RFID

I *could* observe standards, but chances are, I'll relay data in a **vendor-specific** format.



RAIN Tag



RAIN Reader



(UHF RFID)  
RAIN RFID  
has  
actual  
standards!



# LLRP and RCI: RAIN has standards!

## Low-Level Reader Protocol (LLRP)

- Widely supported by readers.
- Legacy protocol (2007).\*
- Enjoy reading:

<https://www.gs1.org/standards/epc-rfid/llrp/1-1-0>

\* remember XML!?!

## RAIN Communication Interface (RCI)

- Low adoption.
- Modern protocol (~2018).
- Latest version may not even be publicly available? 🙋

Our generation speaks **LLRP!**



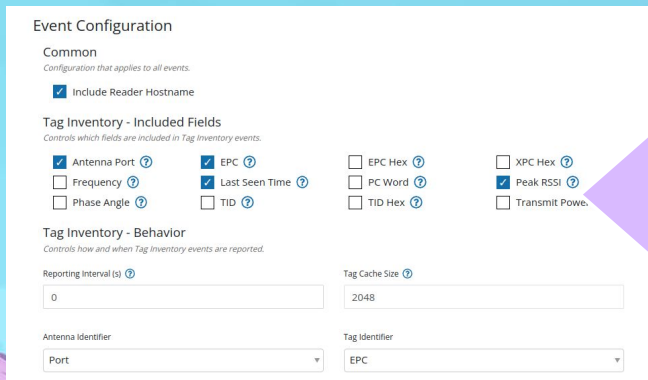
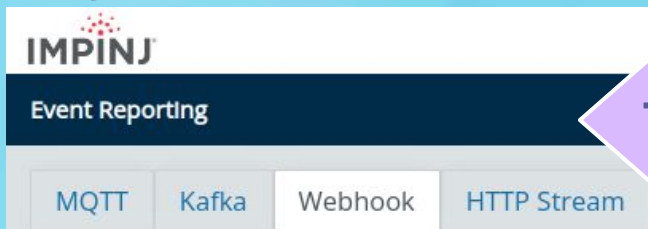
I (alone?) speak **RCI!**



Clairvoyant Technology S2 Reader

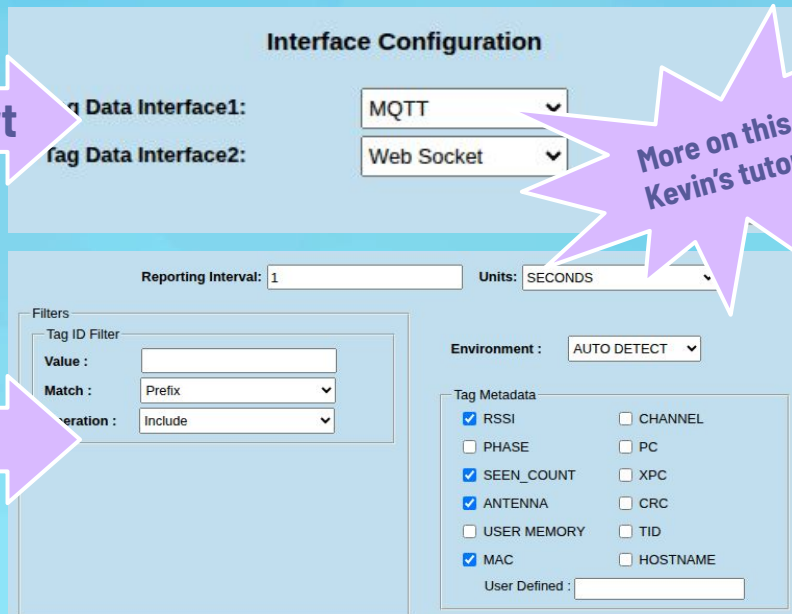
# Reader configuration examples

Impinj R700 web interface:

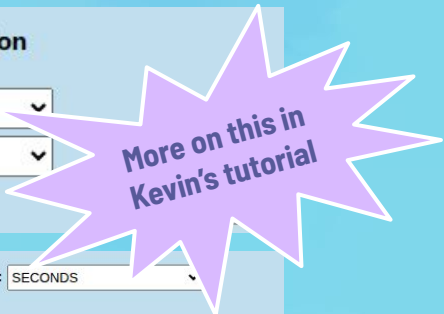


<https://reelyactive.github.io/diy/impinj-r700-config/>

Zebra FX9600 web interface:



<https://reelyactive.github.io/diy/zebra-fx9600-config/>



# Gateways decode BLE

No standards? No worries!  
I'll relay data in a  
**vendor-specific** format.



BLE Beacon



BLE Gateway

Vendor-specific

Vendor-specific

Vendor-specific

The  
Bluetooth SIG  
never created  
a standard  
equivalent to  
LLRP or RCI.



# Gateway configuration example

Minew MG7 mobile app interface:



### Transport

MAC e4b3230ac5e8

Server Settings

MQTT Setting  HTTP Setting

\*Url  
192.168.1.218:3001/minew/

KeepAlive

Authentication Type  
 Basic  None

Network Settings

Proto  
 dhcp  static

Bluetooth Settings

\*Rssi  
-100dBm

\*Scan Interval  
100 milliseconds

Last Confirm

### Data

Bluetooth Settings

\*Rssi  
-100dBm

\*Scan Interval  
100 milliseconds

\*Scan Window  
Scan window parameters should not be greater than scan interval parameters  
100 milliseconds

MAC Filtering

Mac RegEx  
Please enter a regular expression

Raw Data  
Please enter filter

Other Settings

\*Upload Interval  
1 seconds

Last Confirm

\* Fun fact: many BLE Gateways can be configured using BLE!

<https://reelyactive.github.io/diy/minew-mg7-config/>



# WiFi Access Points decode BLE

Many of us include a BLE radio which allows us to act as **ubiquitous** BLE gateways.\*



BLE Beacon



WiFi AP

Vendor-specific

Vendor-specific

Vendor-specific

Again, there is no single standard to structure and transport the data.

\* 2024 Tutorial: Towards Ubiquitous RFID Infrastructure

<https://www.reelyactive.com/science/ieeerfid/reelyActive-TowardsUbiquitousRFIDInfrastructure-240604.pdf>



# AP infrastructure configuration

Example: HPE Aruba Networking Central cloud platform

The screenshot displays the HPE Aruba Networking Central interface. The 'Classify' section on the left shows a grid of application categories such as 'HPE AIOV Devices', 'Edgecore Devices', 'Normakuba', 'Beacons', 'EndOcean BLE', 'Minow Devices', 'Bleec Devices', 'Hibouair', 'MOLIM', 'ASSA AIRLOY', 'TrainProtect Sensors', 'Amberbox Gunshot Det...', 'Aruba Meridian', 'HPE EndOcean', 'HYPROS Location and IoT...', 'Securitas Healthcare BLE...', 'OpenLocate BLE Beacons', and 'SonderONE AP-LINK'. The 'Observe' section on the right shows a table of devices with columns for Status, Address, Classes, App Dev..., Loca..., Company Ide..., and Last Seen. A 'Transport' dialog box is overlaid in the center, showing configuration options for a device.

Status	Address	Classes	App Dev...	Loca...	Company Ide...	Last Seen
Online	D5:78:EB:57:AC:23	hibouair	220342	HibouAIR		03/04/2025, 06:14:34
Online	F0:A3:65:33:1B:D1	hibouair	2202f7	HibouAIR	Smart Sensor Devices AB	03/04/2025, 06:14:32
Online	F3:FD:48:1F:69:AC	InteroperableIdentifier	0002764			03/04/2025, 06:14:26
					EnOcean GmbH	03/04/2025, 06:14:25
					Pur3 Ltd	03/04/2025, 06:13:58
						03/04/2025, 06:13:33
		InteroperableIdentifier	ac23:3fab:7e:d7			03/04/2025, 06:13:33
						03/04/2025, 06:13:32
					Apple	03/04/2025, 06:13:31

<https://relyactive.github.io/diy/aruba-iotops-config/>

# So how do we receive data now?



RAIN Tag



RAIN Reader



BLE Beacon



BLE Gateway



BLE Beacon



WiFi AP



**Standard**  
Over-the-Air



JSON via Webhook

RAIN Communication Interface (RCI)

Low-Level Reader Protocol (LLRP)

STOMP over WebSocket

JSON via MQTT

Binary via Webhook

Binary over TCP

Protobuf via MQTT

Binary over UDP

Binary serial stream

Protobuf over WebSocket



# How'd we prefer to receive data?



RAIN Tag



RAIN Reader



BLE Beacon



BLE Gateway



BLE Beacon



WiFi AP



**Standard**  
Over-the-Air



**Middleware**



Single,  
common  
interface for  
software &  
applications.



# That's why middleware matters

**Standard** over-the-air. **Non-standard** post reader/gateway/AP.

**Middleware** bridges the gap between what **hardware** *provides* and what **software** applications *expect*.

# Part 2:

## Dealing with Data



# A common read? Indeed!

Is it nonetheless possible to define a **standard** way for readers, gateways and APs to represent RFID decoding data??? **YES!** 🧐

IEEE RFID 2019:



<https://www.reelyactive.com/science/ieeerfid/reelyActive-CoLocatedRFIDSystemsUnite-190402.pdf>

IEEE WF-IoT 2024:

### raddec: Elevating IoT Interoperability Through a Common Radio Decoding Data Format

Jeffrey Dungen | reelyActive | Montréal, Canada

**Abstract**  
Whereas the interoperability of radio-frequency devices at the hardware level of the chip is often the result of the manufacturer's requirements, the interoperability at the higher level of application of the data which is produced by the different radio-identification technologies can be presented as a common representation of a radio decoding for hardware IoT technologies, such as RFID, NFC and Bluetooth Low Energy, favoring interoperability between the hardware and software layers of the IoT and beyond. Interoperability is an operational force that defines what is possible and what is not possible in software development for IoT. All application code activity in both these areas, the radio, conforms to a common representation of a radio decoding, protocols and hardware. The reason for this is the radio-agnostic data format created for the representation of radio decoding would greatly benefit the IoT ecosystem. It is a shared task to create a common representation that is related to every use of the radio: in their own work and to contribute to the IoT ecosystem.

**Motivation**  
Interoperability through a common radio decoding data format across hardware IoT technologies and their vendor-specific implementations.

**Efficient Transport & Processing**  
Gateway IoT Middleware Software  
Pareto  
Class methods: → appearance → Event drivers: → deployment → security → transport → driver service  
raddec: abstractCoding() → appearance → deployment → security → transport → driver service  
raddec: abstractURL() → deployment → security → transport → driver service  
raddec: merge() → security → transport → driver service  
raddec: toString() → transport → driver service  
etc.

**Continuous Evolution**  
Since 2019, radio properties have been added to accommodate new technologies and features from the IoT ecosystem. These include:  
- json: For real-time location system (RTLS)  
- json: For real-time asset tracking  
- json: For RTLS based on angle of arrival  
The radio is the core data structure of Pareto. It opens source IoT middleware, and is used in conjunction with hardware IoT gateways and APs. It is a shared task to create a common representation of a radio decoding for hardware IoT technologies and their vendor-specific implementations. It is a shared task to create a common representation that is related to every use of the radio: in their own work and to contribute to the IoT ecosystem.

**Binary representation**  
38865804d451b0ac0781c3820823454789d0f083e \\ 974157f4012462d877a3be43440820241f64980 \\ 32632324565677890abcdeef1234567890

GitHub/reelyactive/raddec

IEEE 10th World Forum on Internet of Things | Ottawa, Canada | 2024

<https://www.reelyactive.com/science/reelyActive-IoT2024.pdf>

IEEE  
RFID  
2026

Jeffrey Dungen

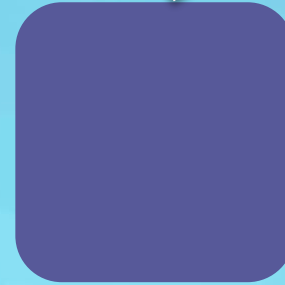
Middleware & Applications

# Radio decoding: common stuff

I have a unique **identifier**.



I too have a unique **identifier**.



I'm received at a given **signal strength**,  
at a given **time**.



# Common properties: “raddec”

A **radio decoding (raddec)** includes:

- Transmitter identifier
- Receiver identifier
- RSSI (received signal strength indication)
- Timestamp
- An *optional* payload and other metadata

regardless of the underlying technologies, protocols and vendor(s).

Open-source implementation:

<https://github.com/reelyactive/raddec>



# What's in payloads and memory

Sometimes there's more to RFID than just ID.

**RAIN RFID** tags may have **memory**:

- memory can be *read*
- can be interpreted as sensor data
- memory may be written to

But wait, there's more!



**BLE** devices may expose **services and characteristics**:


- these too can be *read*
- can include sensor data
- may be written to

**BLE** devices can also spontaneously transmit packets with user-defined payloads.



# A tale of temperature (RAIN)

There's a standard for temperature data, specified in **Section 8** of the **EPC® Radio-Frequency Identity Gen-2 UHF RFID Standard**:



EPC® Radio-Frequency Identity Generation-2 UHF RFID Standard

**Table 8-1: Snapshot Sensor data formats**

Sensor type	Sensor	Units	Sensor data
0001 <sub>2</sub>	Temperature	°C	12-bit signed integer

**Table 8-2: Snapshot Sensor data values**

Sensor type	Sensor	Scale factor	Minimum value		Maximum value		Error value
			binary	scaled	binary	scaled	
0001 <sub>2</sub>	Temperature	0,0625	100000000001 <sub>2</sub>	-127,9375	011111111111 <sub>2</sub>	127,9375	100000000000 <sub>2</sub>

$21^{\circ}\text{C} = 336 \times 0.0625$   
**000101010000**

<https://ref.gs1.org/standards/gen2/>



# A tale of temperature (RCI)

There's a standard representation of temperature data, specified in **Annex H** of the **RAIN Reader Communication Interface Guideline (v5)**:

Sensor	Sensor name	Units	Sensor value
Temperature	TempC	°C	JSON number

Enable snapshot sensor data interpretation in the spot profile:

```
{
  "Cmd": "SetProf",
  "InterpretData": [ { "SNAPSHOTSENSOR": null } ]
}
```

Receive snapshot sensor data as friendly **JSON**:

```
"SNAPSHOTSENSOR": {
  "ResponseCode": { "Code": 0, "Desc": "OK" },
  "Temperature": 21.0
}
```

\* JavaScript Object Notation

You had me at **JSON**\*.



# A tale of temperature (BLE)

There's a standard for temperature data, specified in **Section 3.8** of the **Assigned Numbers** document and **Section 3** of the **GATT Specification Supplement**:

Assigned Numbers / Document

## 3.8 Characteristics

Referenced from the following:

- Bluetooth Core Specification [Vol 3] Part C, Section 12 [4].
- Bluetooth Core Specification [Vol 3] Part F, Section 2 [4].

See Bluetooth Core Specification [Vol 3] Part G, Section 3.3 [4].

### 3.8.1 Characteristics by Name

Characteristic Name	UUID
Temperature	0x2A6E

<https://www.bluetooth.com/specifications/assigned-numbers/>

2026-02-05

GATT Specification Supplement / Document

## 3.234 Temperature

The Temperature characteristic is used to represent a temperature.

The structure of this characteristic is defined below.

Field	Data Type	Size (in octets)	Description
Temperature	sint16	2	Base Unit: org.bluetooth.unit.thermodynamic_temperature.degree_celsius Represented values: M = 1, d = -2, b = 0 Unit is degrees Celsius with a resolution of 0.01 degrees Celsius. Allowed range is: -273.15 to 327.67. A value of 0x8000 represents "value is not known". All other values are prohibited.

Table 3.360: Structure of the Temperature characteristic

<https://www.bluetooth.com/specifications/gss/>

$$21^{\circ}\text{C} = 2100 \times 0.01$$

0x0834 -> 0x3408

Big Endian -> Little Endian

# The reality of temperature (BLE)

BLE affords much freedom... ..to reinvent the wheel, over and over.

Byte offset	Field	Description
0	Frame Type	Value = 0x20
1	Version	TLM version, value = 0x00
2	VBATT[0]	Battery voltage, 1 mV/bit
3	VBATT[1]	
4	TEMP[0]	Beacon temperature
5	TEMP[1]	
6	ADV_CNT[0]	Advertising PDU count
7	ADV_CNT[1]	
8	ADV_CNT[2]	
9	ADV_CNT[3]	
10	SEC_CNT[0]	Time since power-on or reboot
11	SEC_CNT[1]	
12	SEC_CNT[2]	
13	SEC_CNT[3]	

Eddystone-TLM  
Service: 0xfeaa

Offset	Length	Type	Data	Details
0	1	Data Length	0x02	AD Structure Data Length
1	1	Flag Data Type	0x01	AD Type is Flags
2	1	Flag Data	0x06	LE General Discoverable Mode, BR/EDR not supported
3	1	Data Length	/	AD Structure Data Length
4	1	AD Type	0xFF	AD Type is Manufacturer Specific Data
5	2	Company ID	0x3906	LE, Manufacturer ID, 0x0639 = Minew
7	1	Frame Type	0xCA	Minew Connect V3
8	1	Frame Version	0x1B	Sensor Frame
9	1bit	Encrypto Flag	-	0: Disable, 1: Enable
9	7bit	Sub-version	0x01	Sub-version
10	2	Product ID	/	Product ID, 0x0007 = MTB02
12	1bit	CTE Flag	-	0: Disable, 1: Enable
12	5bit	CTE Length <sup>1</sup>	-	CTE Length Value, range 2-20, unit: 8us
12	2bit	Encrypto Range	-	Encryption Range, refer to Encrypto Range
13	6	MAC Address	/	LE, Product Serial Number ID, equal to MAC address. 0x0165332211AC = AC:11:22:33:44:55
19	2	Battery	/	BE, Battery Voltage, Units: mV
21	2	SOC Temperature	/	8.8 Fixed Point, 0x1973 => 25.15
23	2	Reserved	/	Reserved
24	1	Measured Power	/	Calibrated RSSI value at 1m
25	2	Nonce Salt	/	-
27	4	Nonce Counter	/	-

Object id	Property	Data type	Factor	Example	Result	Unit
0x57	temperature	sint8 (1 byte)	1	57EA	-22	°C
0x58	temperature	sint8 (1 byte)	0.35	58EA	-7.7	°C
0x45	temperature	sint16 (2 bytes)	0.1	451101	27.3	°C
0x02	temperature	sint16 (2 bytes)	0.01	02CA09	25.06	°C

BHome.io  
Service: 0xfcd2



Minew Connect  
Company: 0x0639

Middleware & Applications

Jeffrey Dungen

IEEE  
RFID  
2026



# Best practices for BLE

For those who *don't* enjoy reading protocol specification documents.



# advlib: advertising data library



advlib-epc-tds

advlib-esp

advlib-ble

...0110101

advlib-interoperable



advlib

<https://github.com/reelyactive/advlib>



```
{
  acceleration: [ 0.2, 0.9, 0.3 ],
  batteryPercentage: 50,
  deviceIds: [ '...', '...' ],
  heartRate: 60,
  illuminance: 333,
  isButtonPressed: [ false ],
  isContactDetected: [ true ],
  isMotionDetected: [ false ],
  magneticField: [ 0.7, 0, 0.1 ],
  name: "advlib by reelyActive",
  position: [ -73.5, 45.5, 88 ],
  relativeHumidity: 69,
  temperature: 21.0,
  txCount: 123456789,
  unicodeCodePoints:[ 0x1f989 ],
  uptime: 60000,
  uri: "https://...",
  uuids: [ '...', '...' ],
  version: "1.2.0"
}
```

😬 Raw binary

😎 Friendly JSON

# advlibs , not Mad Libs\*

Eddystone-TLM data (BLE): 20000bb815000000004500000258



[github.com/google/eddystone/](https://github.com/google/eddystone/)

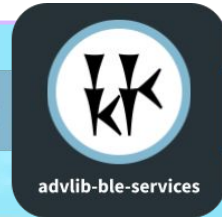
Once upon a time, a weary beacon whose battery was only **3V** measured a temperature of **21C**. It shouted for the **69th** time, exclaiming "did you know I've been up for **60** seconds?!?"

\* Fun fact:  
Mad Libs still exist!  
[madlibs.com](https://www.madlibs.com)

IEEE  
RFID  
2026

Jeffrey Dungen

UUID + Service Data



advlib-ble-services



Middleware & Applications



# Dynamic ambient data

Common, vendor-and-technology-agnostic properties\* include:

acceleration	elevation	isOccupancyDetected	pm1.0 / pm2.5 / pm10
ammoniaConcentration	energy	isSmokeDetected	position
amperage	heading	isTamperDetected	power
angleOfRotation	heartRate	languages	<b>pressure</b>
angularVelocity	illuminance	levelPercentage	relativeHumidity
batteryPercentage	isButtonPressed	luminousFlux	soundPressure
batteryVoltage	isCarbonMonoxideDetected	magneticField	speed
carbonDioxideConcentration	isContactDetected	methaneConcentration	temperature
count	isGasDetected	nearest	text
dissolvedOxygen	isHealthy	nitrogenDioxideConcentration	txCount
distance	isInputDetected	nitrogenOxidesIndex	txCycle
duration	isLightDetected	numberOfOccupants	uptime
	isLiquidDetected	passageCounts	velocityOverall
	isMotionDetected	pH	voltage

\* defined & observed by reelyActive's middleware

<https://reelyactive.github.io/diy/cheatsheet/#dynam>

For the **TPMS reports**  
in Kevin's tutorial 😊

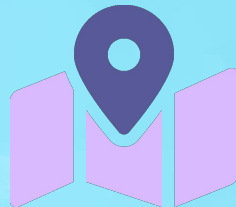
# Decoding metadata

A radio decoding may result in more than just **RSSI**...

- **AoA** (angle of arrival)
- **TDOA** (time difference of arrival)
- **Range** (ex: phase-based ranging, round-trip time, ToF, ...)
- **IQ samples** (ex: to calculate AoA) [[https://wikipedia.org/wiki/In-phase\\_and\\_quadrature\\_components](https://wikipedia.org/wiki/In-phase_and_quadrature_components)]
- **Frequency**
- **Phase**

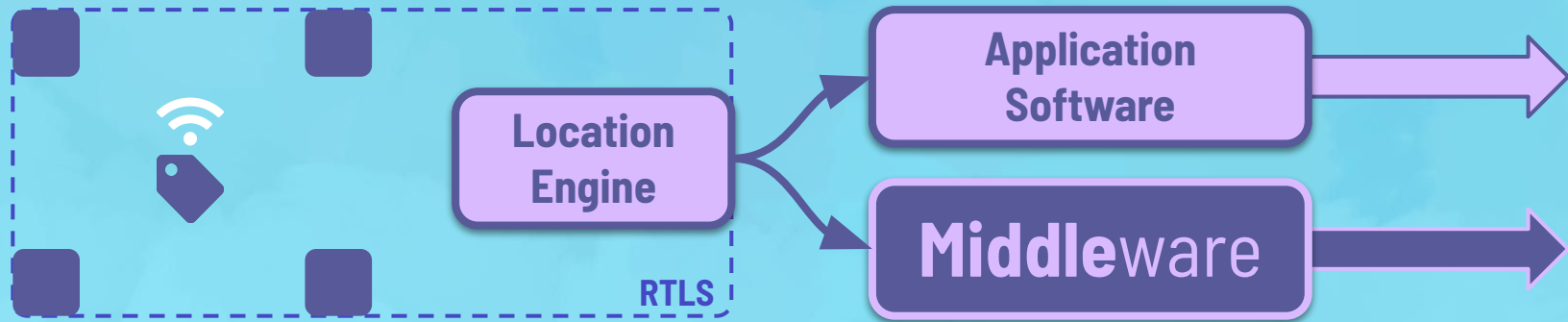
Recall Dan's tutorial!

Often, these are used for absolute or relative **positioning** estimation.



# Real time location systems (RTLs)

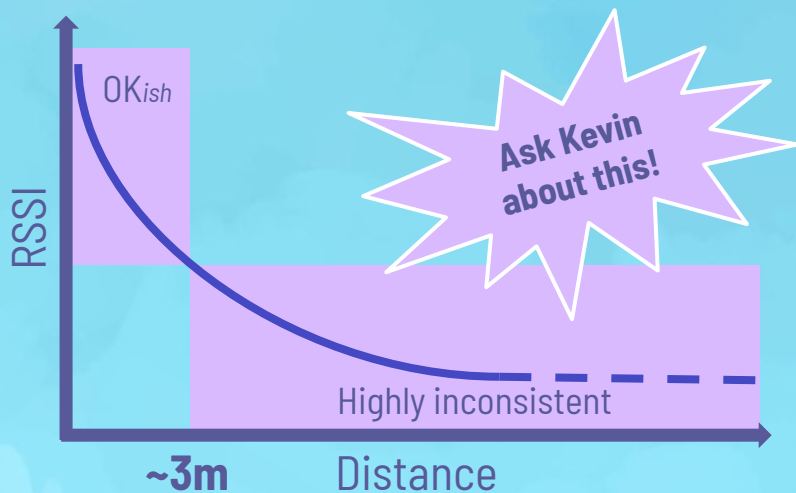
A RTLs combines hardware and a **location engine** to estimate the position of tags/beacons based on decoding metadata.



Some RTLs vendors bundle application software to create an end-to-end solution. Middleware may also be used.

# At least we always have RSSI

Simple, ubiquitous **RSSI-based** position estimation may nonetheless be “good enough” provided sufficient infrastructure density\*.



\* Does it make more sense to:

- install a gateway/antenna every 3m (10ft)?
- invest in a complex RTLS?

Write your own positioning/location engine:  
<https://github.com/reelyactive/chimps>



# Part 3:

## Digital Twin for the Win



# A digital twin of what?

Are we “attached” to a purist definition of a digital twin? 🤔

I'm a **tag!**

But you represent me,  
the **parcel**.

I'm a  
**MTB11!**

But you represent me,  
the **attendee**.

# Advertising a URL (BLE)

Bluetooth Low Energy devices can *spontaneously* broadcast a (short\*) **URL** (in a GAP-standard way!) to any observers in range.



<https://linkedin.com/in/dungen/>

I'm gonna look up this guy's **digital twin**.

\* The URL above is just too long to fit in a *legacy* BLE advertising packet.

# Product lookup (RAIN & GS1)

Show an Example Clear

**GS1 Key or other identifier** — as used in bar codes

GTIN + serial (AI 01 + AI 21) (01) 80614141123458  
6789

**Serial** ←

GS1 Company Prefix Length 7 digits

**EPC Pure Identity URI (urn:epc:id:...) — as used in EPCIS**

urn:epc:id:sgtin:0614141.812345.6789

**RFID Control Information**

Tag Size 96 bits Filter Value 3 - reserved

**EPC Tag URI (urn:epc:tag:...) — as used in RFID middleware**

urn:epc:tag:sgtin-96:3.0614141.812345.6789

**RFID Tag EPC Memory Bank Contents (Hexadecimal) — starting at bit 20h**

3074257bf7194e4000001a85

**EPC** ←

 This number is registered to **GS1 US, INC.**

**Company information**

Information about the company that licenced this GTIN

Company Name	GS1 US, INC.
Address	7887 WASHINGTON VILLAGE DR STE 300 DAYTON, OH 45459-3988 United States of America
Website	Unknown
License Key	0614141
License Type	GS1 Company Prefix
Global Location Number (GLN)	0614141000005
Licensing GS1 Member Organisation	GS1 US

<https://www.gs1.org/services/epc-encoderdecoder>  
<https://www.gs1.org/services/verified-by-gs1/results?gtin=80614141123458>

**SGTIN Example: EPC = SGTIN-96**  
Serialised Global Trade Item Number

3074257bf7194e4000001a85

# EU Digital Product Passport

The European Union DPP, which is currently in a phased roll-out, will be *mandatory* for certain products sold in the EU.

- A DPP is linked to a product by an **identifier**
- The identifier can be in the form of a QR code, **RFID tag**, etc.
- DPP **information** may be publicly available, or user-privileged
- A DPP may allow **updates** as it follows the product life cycle

The emerging standard(s) should serve as a blueprint for broader **digital twin initiatives** worldwide.

<https://untp.unece.org/>

<https://therainalliance.org/gs1-and-rain-alliance-joint-dpp-statement/>

# Digital twin applications

RFID, and more generally *Auto-ID\**, enable machine-readable links between **physical things** and their **digital representations**.

We've also seen that a real-time location system (RTLS) provides a digital representation of their **physical location**. And we've seen that ambient **sensor data** can be digitally represented too.

🤔 *Can middleware mix all these together into something useful?*



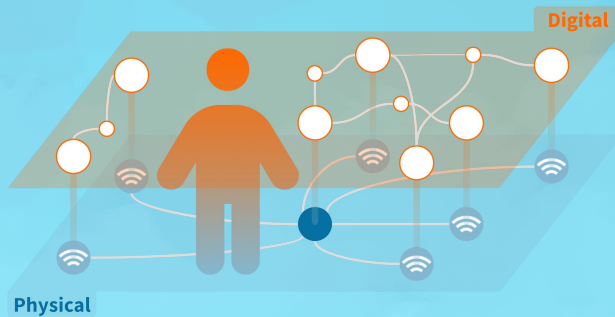
\* 50+ years of barcodes!

# Hyperlocal Context

“A machine-readable, real-time contextual representation of a physical space and its occupants.”

Hyperlocal context is a **digital snapshot** of *who/what is where/how*, achieved by combining:

- Real-time location
- Digital twins
- Live sensor data



<https://www.reelyactive.com/context/>

# Hyperlocal Context tutorial(s)

Once again, there's an IEEE RFID tutorial for that!\*



Tutorial: **RFID as Ambient Data**

Jeffrey Dungen  
reelyActive

May 17, 2022



Tutorial: **Context-aware physical spaces**

Jeffrey Dungen  
reelyActive

June 13<sup>th</sup>, 2023

<https://www.reelyactive.com/science/ieeerfid/>

\* Actually, two tutorials for that!



# Part 4:

## Application of *everything*



# Real-time data

#	EPC	Read Count	Last Read From	First Seen	Last Seen	Time Since Last Seen	Last RSSI	RSSI Avg	RSSI Max	RSSI Min	Antenna
1	CCC08B7700023300ABC00024	82	My R700 IoT	0.378	8.906	0.071	-63.500	-64.010	-62.000	-65.500	1
2	CCC08B7700023300ABC00022	82	My R700 IoT	0.378	8.906	0.071	-48.500	-51.322	-48.500	-54.000	1
3	CCC08B7700023300ABC00036	82	My R700 IoT	0.378	8.906	0.071	-62.500	-65.271	-61.500	-69.000	1
4	CCC08B7700023300ABC00020	82	My R700 IoT	0.378	8.906	0.071	-57.000	-60.362	-56.500	-64.500	1
5	CCC08B7700023300ABC00021	82	My R700 IoT	0.378	8.906	0.071	-53.000	-54.382	-53.000	-56.000	1
6	CCC08B7700023300ABC0002E	82	My R700 IoT	0.378	8.906	0.071	-61.500	-60.647	-59.000	-62.000	1
7	CCC08B7700023300ABC00034	60	My R420	0.378	8.906	0.071	-64.000	-63.367	-60.500	-68.000	1
8	CCC08B7700023300ABC00026	81	My R700 IoT	0.378	8.906	0.071	-67.000	-66.538	-65.000	-72.000	1
9	CCC08B7700023300ABC00025	69	My R700 IoT	0.427	8.906	0.071	-62.000	-61.396	-59.000	-64.500	1
10	CCC08B7700023300ABC0002F	2	My R700 IoT	3.500	7.748	1.238	-66.000	-67.226	-66.000	-68.000	1

## Impinj ItemTest for Windows (Inventory Showcase)

Source: ItemTest User Manual

device (signature)	events (latest)	rssi (strongest)	receiver	rec/dec/pac (# since event)	timestamp (latest event)
ac233fd35ae2/2	i	-48	001a7dda7112/2	3 / 2 / 5	09:47:28
e5001000158c/3	i	-50	f01aa0458c21/2	3 / 3 / 1	09:47:13
cd2f82cc1029/2	i	-52	001a7dda7112/2	1 / 3 / 1	09:47:26
ac233fd35aea/2	i	-51	001a7dda7112/2	2 / 3 / 7	09:47:28
e121dfd9bfef/2	i	-55	001a7dda7112/2	1 / 2 / 1	09:47:25
ac233fae300b/2	i	-57	001a7dda7112/2	3 / 5 / 19	09:47:28
ac233fd398cb/2	i	-57	001a7dda7112/2	2 / 3 / 8	09:47:27
e50010001597/3	i	-57	001a7dda7112/2	3 / 1 / 1	09:47:59
ac233fab7ed7/2	i	-57	f01aa0458c21/2	2 / 3 / 1	09:47:26
ac233fac7c71/2	i	-57	001a7dda7112/2	1 / 2 / 1	09:47:25
da132d2a0089/3	i	-58	f01aa0458c21/2	2 / 3 / 1	09:47:26
c300021be77/3	i	-58	f01aa0458c21/2	2 / 3 / 1	09:47:26
ac233fd35a5b/2	i	-59	001a7dda7112/2	3 / 7	09:47:27

## Devices Observer web app

Demo: <https://reelyactive.github.io/pareto-anywhere-apps/devices-observer/?demo=default>

# A quick note on transport

There are many ways to transport data between the “wares”:

- **MQTT**: publish/subscribe messaging protocol
- **WebSocket**: bidirectional real-time comms over TCP
- **Webhook**: HTTP POST to endpoint
- **TCP / UDP**: low-level connection-oriented/less protocols

Industrial standards that combine transport & structured data:

- **Sparkplug**: MQTT with Protobufs, payload/state definitions, ...
- **OPC-UA**: transport agnostic communication/info standard\*

\* standardises telemetry and, now, “unified locating” for spatial intelligence too!

# Event-driven applications

Flow-based, low-code, event-driven programming.

The screenshot shows the Node-RED web interface. The main workspace contains a flow with the following components:

- A **pareto-anywhere-socketio** node (blue) connected to a **Pareto Anywhere** node (blue).
- The **Pareto Anywhere** node is connected to three processing nodes: **raddec**, **dynamb**, and **spatem** (all green).
- A status indicator below the **Pareto Anywhere** node shows a green dot and the text "connected".

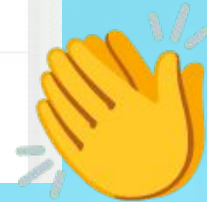
Annotations in the interface include:

- A speech bubble at the top: "Connects via Socket.IO to Pareto Anywhere on localhost:3001".
- A speech bubble at the bottom: "Observe JSON data in the debug messages tab".

The right-hand side shows the **debug** console with the following JSON data:

```
25/09/2024, 12:17:45 node: raddec  
msg.payload : Object  
  ▶ { timestamp: 1  
    deviceId: "ac233  
    deviceIdType: 2,  
    batteryPercentage: 69,  
    acceleration: array[3] }  
  
25/09/2024, 12:17:45 node: raddec  
msg.payload : Object  
  ▶ { transmitterId:  
    "7ababae2e7c4",
```

Whenever Jeff's badge enters or leaves the stage, light up the "Applause" sign.



**Node-RED** open source software  
<https://nodered.org/> See also: <https://flowfuse.com/>

# Databases

Store and query historical data. Flavours include:

- **SQL**: relational databases that support SQL queries
- **NoSQL**: non-relational databases that don't require data to conform to a schema
- **Time Series**: databases optimised for compressing and querying high volumes of timestamped data

Middleware generally supports a variety of common databases, ideally the database you already use!

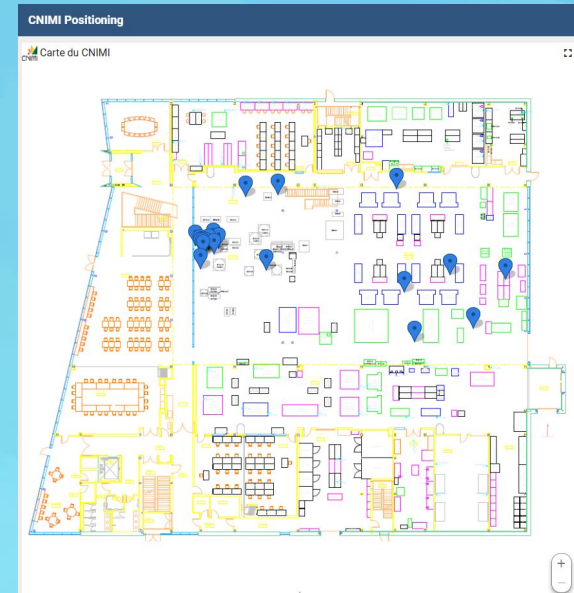
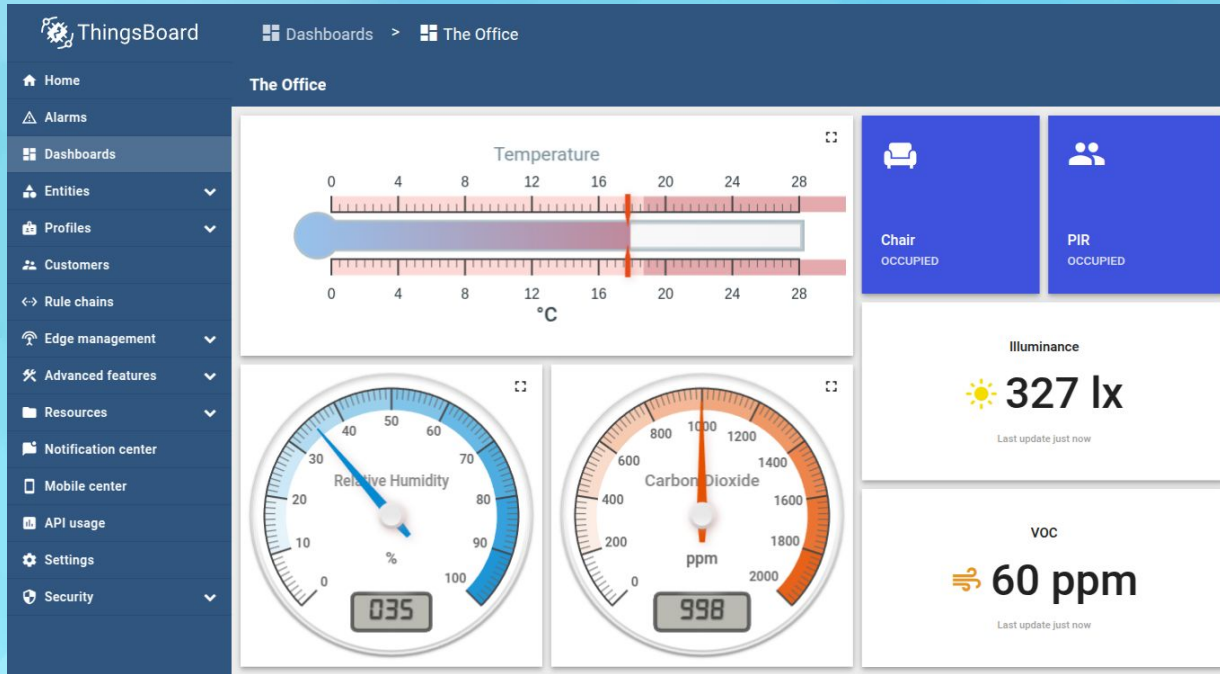
That's me!



# Dashboards (ex: Grafana)



# Suites (ex: ThingsBoard)

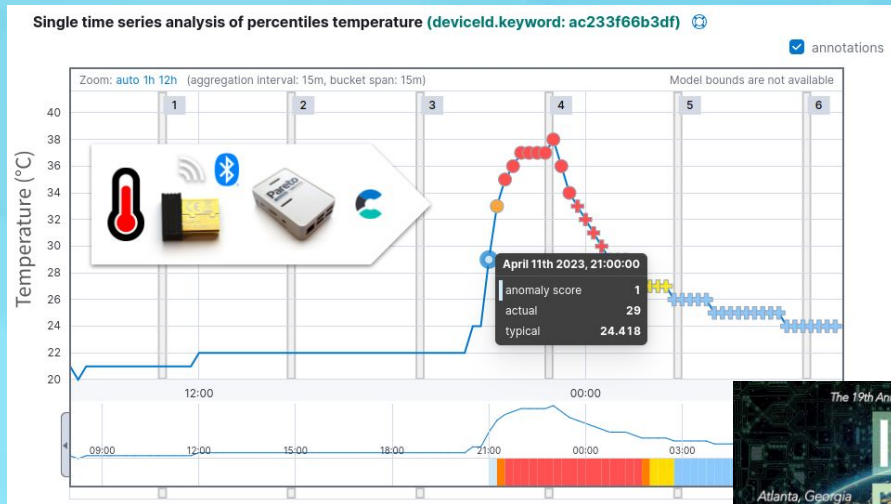


Real-time location of RAIN RFID tags at CNIMI, in Drummondville, Québec.

Suite combines database, dashboard, flows, etc.

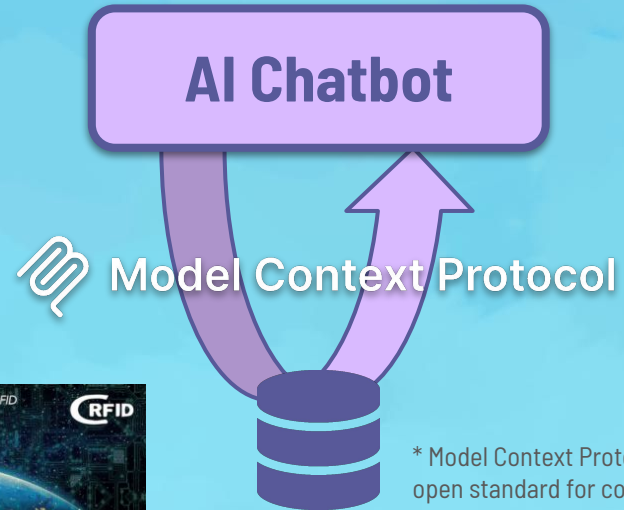
# Artificial Intelligence

## Anomaly Detection of time series data



Real example using Elastic Stack

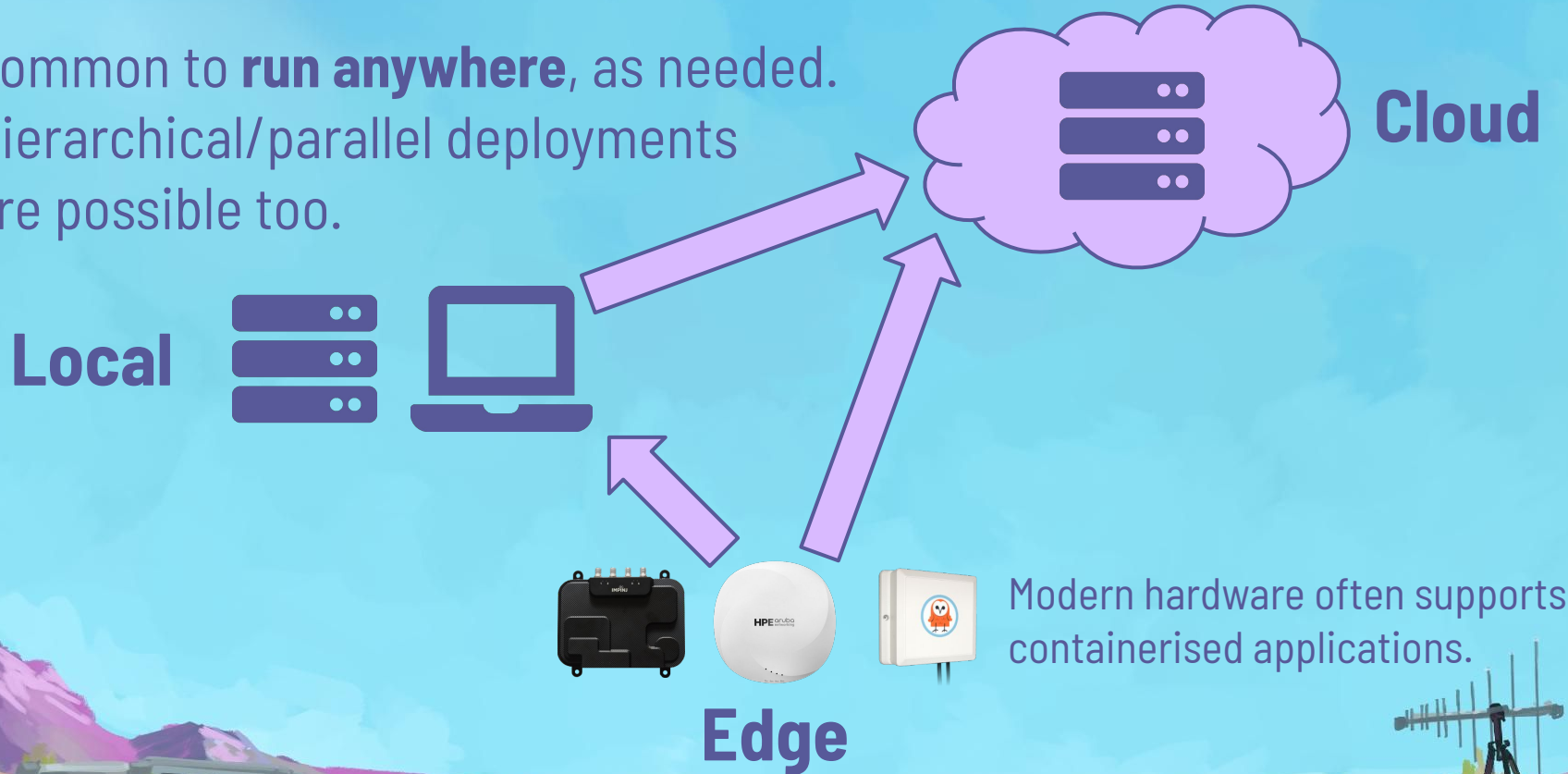
## MCP\* interface to databases



<https://www.reelyactive.com/science/ieeerfid/reelyActive-RFIDinAIoT-250422.pdf>

# Where do middleware & apps run?

Common to **run anywhere**, as needed.  
Hierarchical/parallel deployments  
are possible too.



# Summary

**Hardware** does the hard part, but leaves a *mess* of data/interfaces.

**Middleware** tidies the mess, making data *usable* for software.

**Software** enables *applications*, Kevin\* will present examples next.

\* not Kevin



Source: Office Space (1999)

*I believe you have my stapler.  
I put a beacon on it, and am  
tracking it with open source  
middleware, using the office  
WiFi access points....*

# Middleware & Applications

IEEE RFID 2026 Tutorial

presented by

**Jeffrey Dungen**



Up next!

