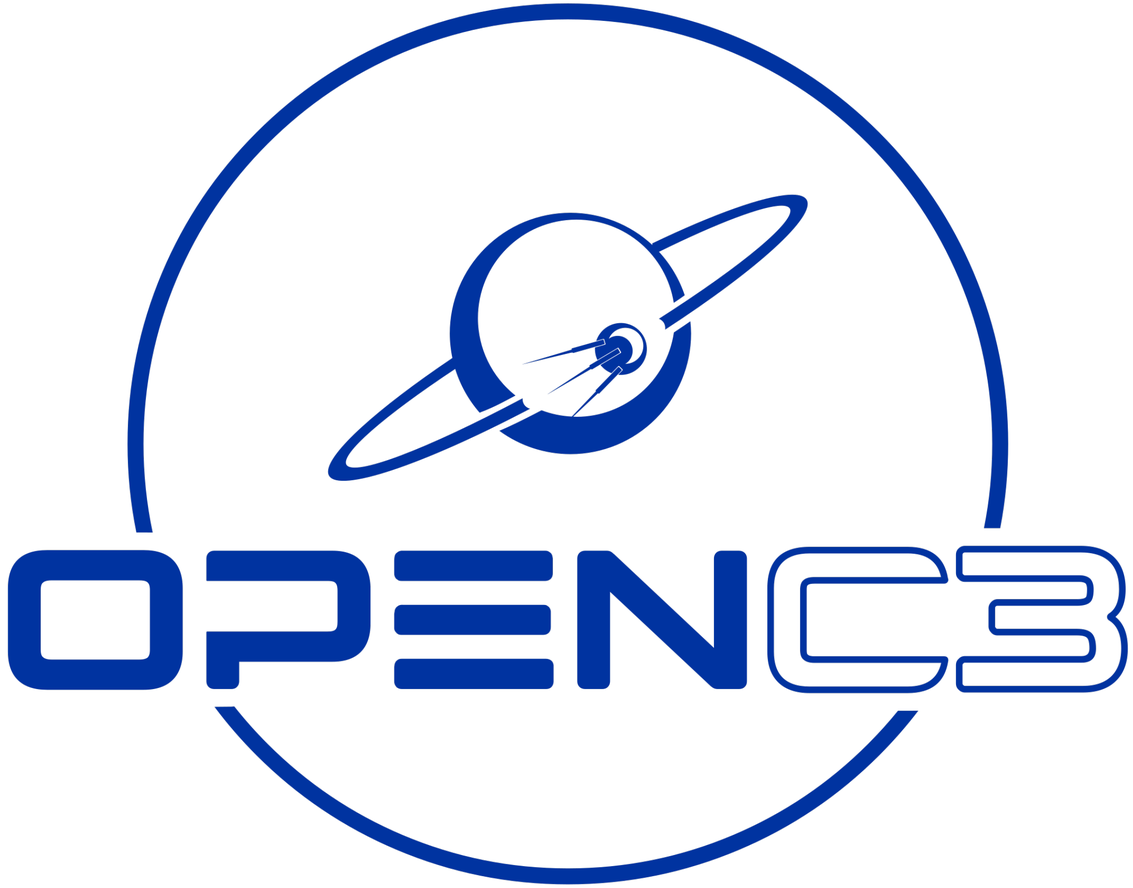
Interoperability Without Standards

An Architecture For Handling Data in Any Format

Ryan Melton
Jason Thomas



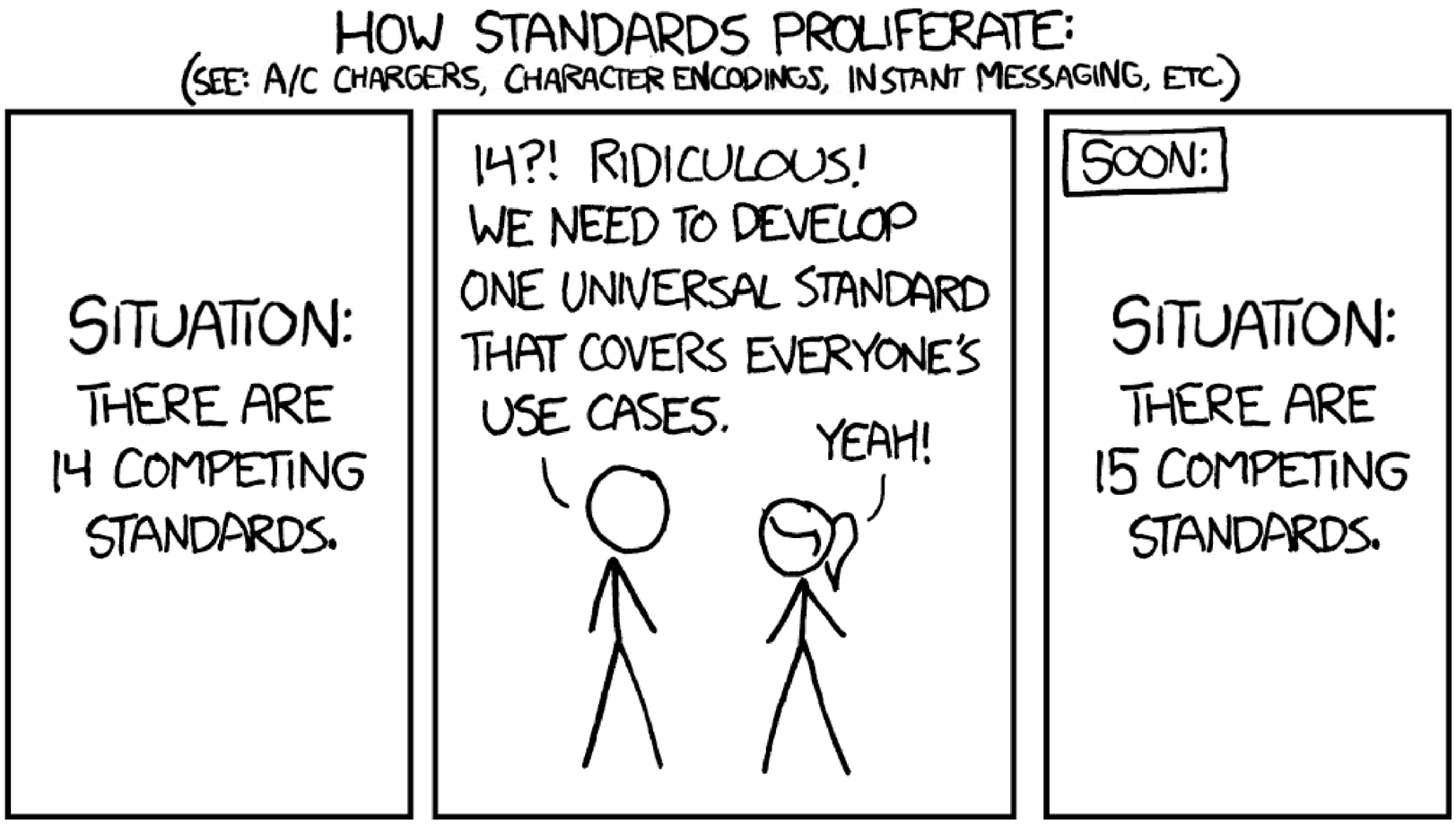
The Fundamental Problem

How can we make it **easy** to communicate with any piece of hardware?

The Traditional Answer

Make everything conform to the same standard!

Relevant XKCD



https://xkcd.com/927/

Reality

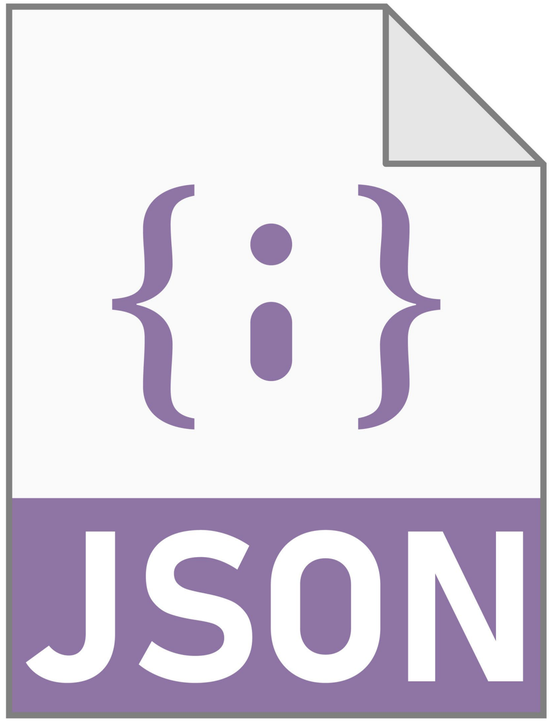
There are hundreds of standards and engineers are creating new ones every day

But Why?

- Humans are driven to make things better… and they can always be better
- We learn from the standards that came before us and make improvements from there
- An optimized standard is inherently non-universal
- Standards that try to be universal are generally overly complex

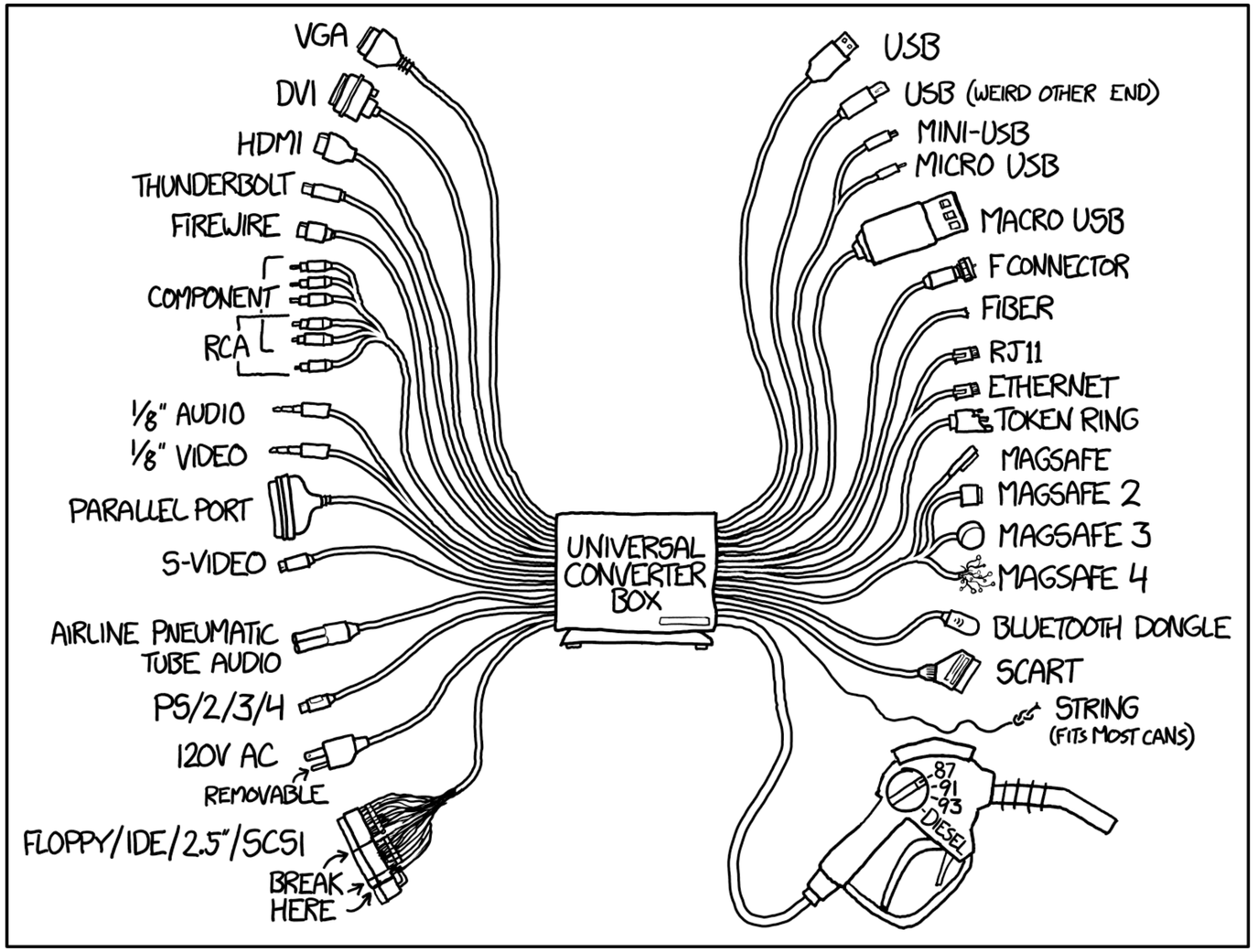
Tradeoffs

- All standards have design tradeoffs… ie:
  - JSON is easy for humans to read/edit, is self-describing, and cross platform
  - But… it is not bandwidth efficient (utf-8 text), has no built-in way to transfer binary data, and has no comments



What We End Up Needing

- TCP/IP
- UDP
- Serial
- HTTP
- MQTT
- HTML
- XML
- JSON
- CBOR



https://xkcd.com/1406/

- CCSDS
- CFDP
- CSP
- Websockets
- GEMS
- GMSEC
- ActiveMQ
- ZeroMQ
- Etc.

First Things First

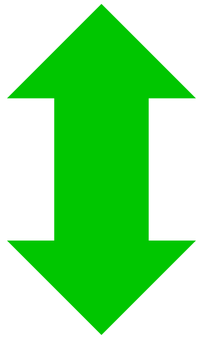
With all these different kinds of data, how can we provide a common pattern for interfacing with anything?

Key Value Pairs To/From Binary

TARGET: MYSAT

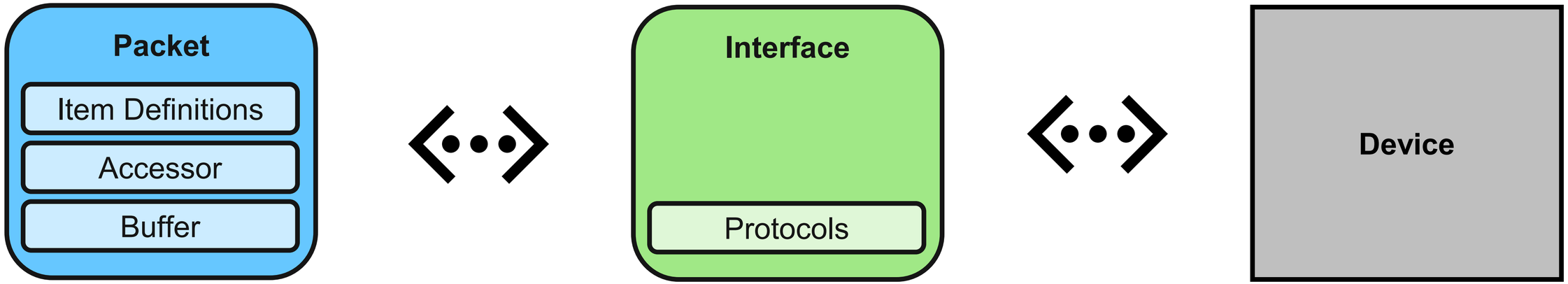
COMMAND: HEATER

SETPOINT: 30.0



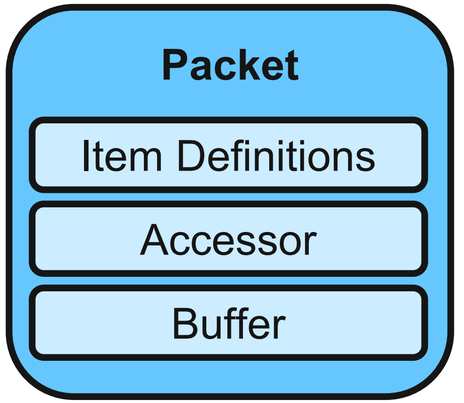
1A CF FC 1D 03 02 58 93 02 19 E1 87 FF

Our Solution



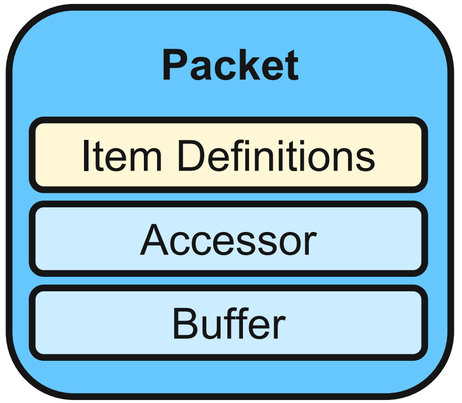
Packets

- Packets form the set of information read from / written to Interfaces.
- Item Definitions provide all the metadata necessary for an Accessor to read/write data from the Buffer
- Accessors handle different Buffer serialization formats
- The Buffer holds the actual message in hardware format



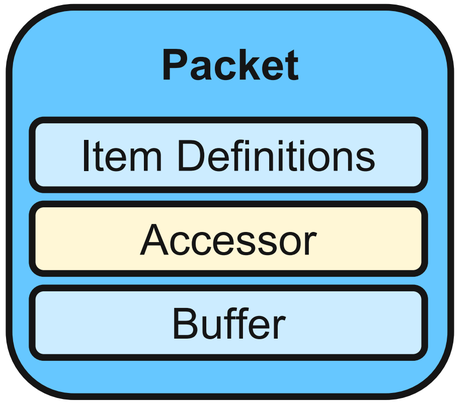
Item Definitions

- Contain the information needed to read and write every item in the buffer
- Key Information Includes:
  - Item Name
  - Bit Offset
  - Bit Size
  - Data Type
  - XPath/JsonPath style path for complex structures



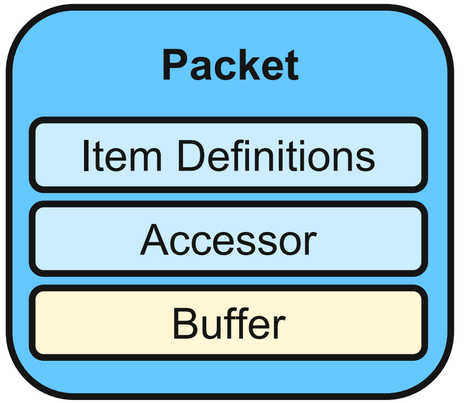
Accessors

- Accessors are what read and write values from the Buffer
- Modular class structure allows creating Accessors for any serialization format
- Standard accessors include:
  - Binary Structures
  - JSON
  - CBOR
  - XML
  - HTML
  - Web Forms
  - Protocol Buffers



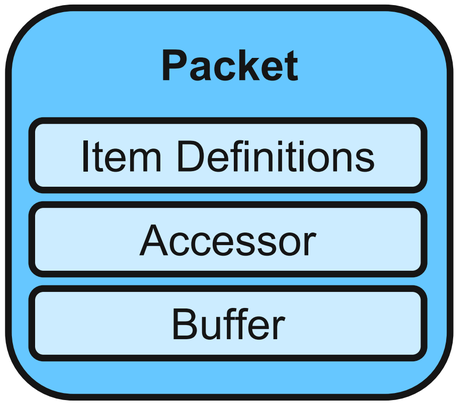
Buffer

- The buffer holds the actual bytes that make up the telemetry that was received from hardware, or will be sent to hardware as a command
- Can be any serialization format that a corresponding Accessor exists for



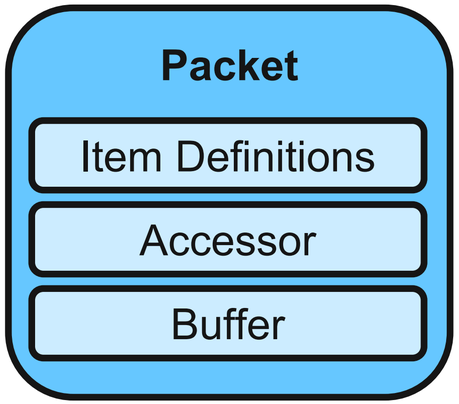
Configuring Packets: JSON

```
COMMAND MYSAT CMD BIG_ENDIAN "JSON Accessor Command"
  ACCESSOR JsonAccessor
  TEMPLATE '{"id_item":1, "item1":101, "more": { "item2":3.14, "item3":"Example” } }'
  APPEND_ID_PARAMETER ID_ITEM 32 INT 1 1 1
    KEY $.id_item
  APPEND_PARAMETER ITEM1 16 UINT MIN MAX 10
    KEY $.item1
    UNITS CELCIUS C
  APPEND_PARAMETER ITEM2 64 FLOAT MIN MAX 3.14
    KEY $.more.item2
  APPEND_PARAMETER ITEM3 128 STRING "Example"
    KEY $.more.item3
```



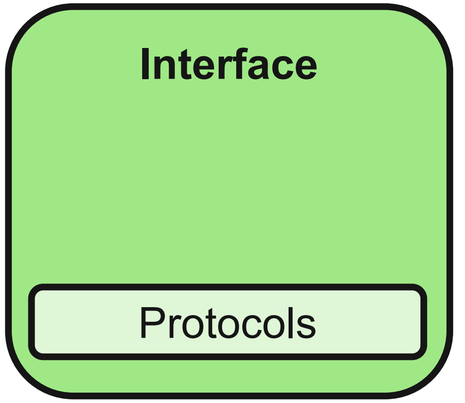
Configuring Packets: Binary

```
COMMAND MYSAT CMD BIG_ENDIAN "JSON Accessor Command"

  APPEND_ID_PARAMETER ID_ITEM 32 INT 1 1 1

  APPEND_PARAMETER ITEM1 16 UINT MIN MAX 10

    UNITS CELCIUS C
  APPEND_PARAMETER ITEM2 64 FLOAT MIN MAX 3.14

  APPEND_PARAMETER ITEM3 128 STRING "Example"
```



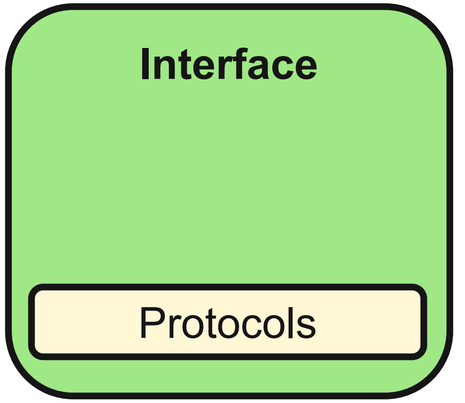
Interfaces

- Packets are sent/received from hardware using **Interfaces**
- Interfaces generally should not care about the specific data sent over them
- Some common interfaces include:
  - TCP/IP Client or Server
  - UDP
  - Serial
  - MQTT
  - HTTP
  - SNMP
  - Various message buses (ActiveMQ, RabbitMQ, etc.)



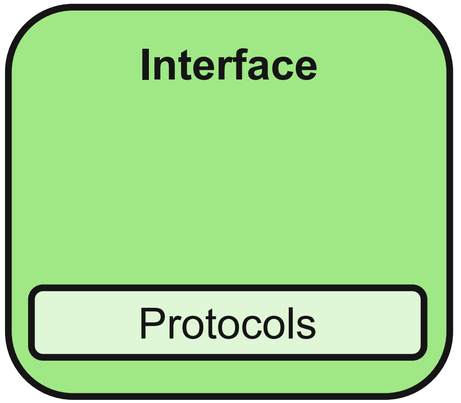
Protocols

- Often what the hardware needs to receive isn't exactly the same as the packet buffer
- Sometimes it has been encrypted, encoded, or broken into smaller frames
- Protocols take packets and convert them into exactly what is needed to send out over an Interface
- They can also calculate and fill in information like length fields and CRCs
- Protocols are also responsible for delineating packets in a stream



Configuring Interfaces

```
# Declare our target named EXAMPLE

TARGET EXAMPLE EXAMPLE

# Create the Interface to connect to the target

INTERFACE EXAMPLE_INT tcpip_client_interface.rb localhost 3000 3000 10.0 nil LENGTH 32 16 7

    PROTOCOL READ_WRITE CrcProtocol CRC

    MAP_TARGET EXAMPLE
```



Target Name

Folder Name

Interface File

Write Port

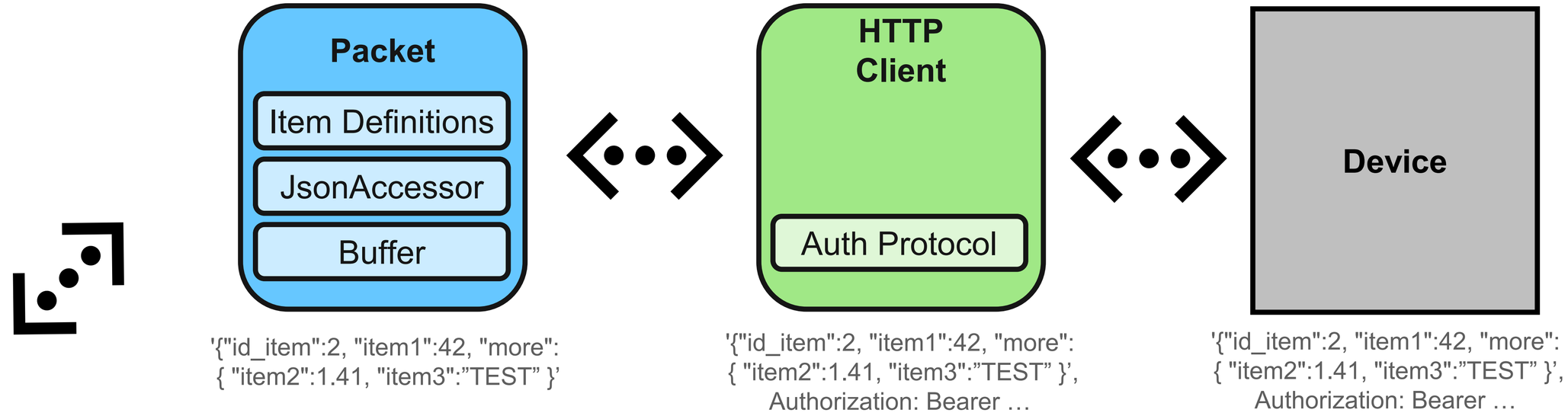
Write Timeout

Read Timeout

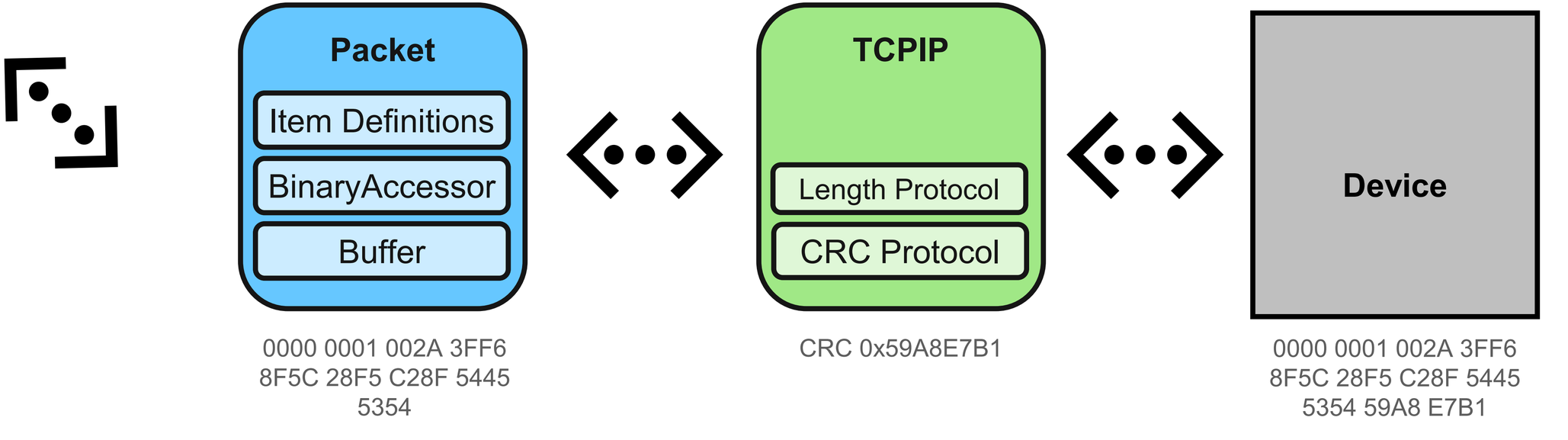
Read Port

Hostname

Data Commutation

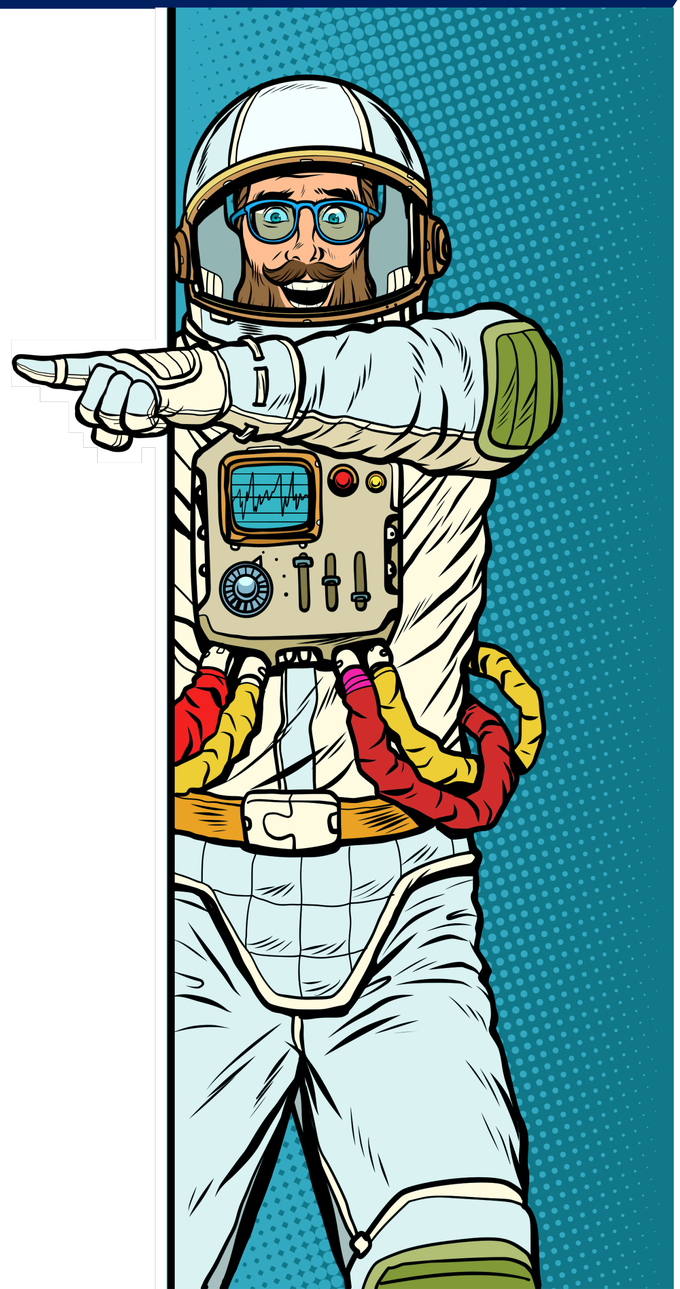


cmd("MYSAT TEST with ITEM1 42, ITEM2 1.41, ITEM3 'TEST'")



Summary and Conclusions

- Connecting to hardware can be broken down into six distinct components:
  - Packets
  - Item Definitions
  - Accessors
  - Buffer
  - Interfaces
  - Protocols
- Using this framework, we can create a system that can connect to anything regardless of whatever standard (or non-standard) it speaks



QUESTIONS ?

Have Something Standard (or Non-Standard) to Connect To?

OPENC3

Check us out at openc3.com