Microsoft

Azure

Azure Messaging

Standards Matter!

Clemens Vasters
Principal Architect – Azure Messaging, Microsoft
@clemensv

OASIS AMQP Technical Committee Chair
OASIS MQTT TC Member
CNCF CloudEvents Architect
OPC UA PubSub Architect

Service Bus
Event Hubs
Event Grid
Relay
IoT

Agenda

What are the Azure Messaging services?

Messaging Patterns and Protocol Characteristics

Push/Pull, Queues, Ingestors/Streams, Routers/Distributors

Messaging Protocol Standards

MQTT, AMQP, HTTP, Kafka, gRPC

Encoding Standards

XML, CSV, JSON, CBOR, Avro, Thrift, Protobuf

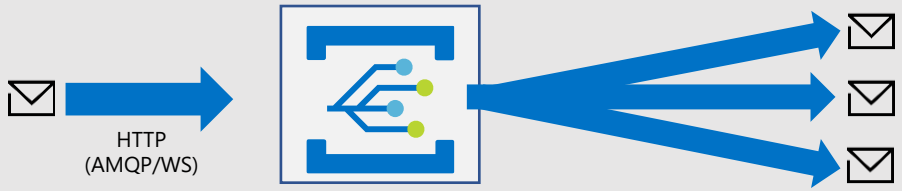
Abstraction Standards

OPC UA, CNCF CloudEvents, JMS 2.0

Azure Eventing and Messaging Core Services

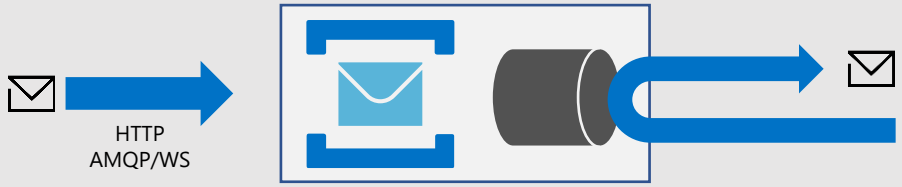
Message Oriented Services

Event Grid



Push-style distribution of discrete events to serverless workloads

Service Bus



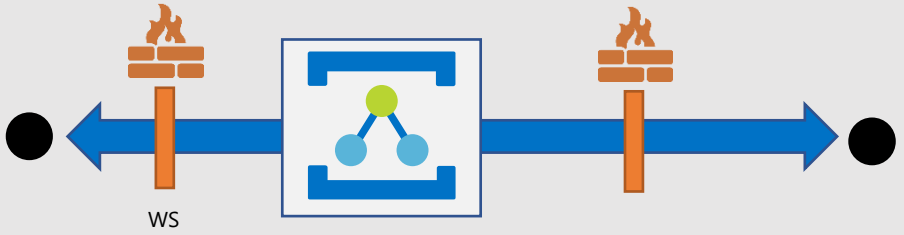
Pull-style, queue-based transfer of jobs and control via message queues and topics

Event Hubs

Partitioned, high-volume, tape-drive-style sequential recording and unlimited, pull-style re-reads of event streams.

Connectivity Services

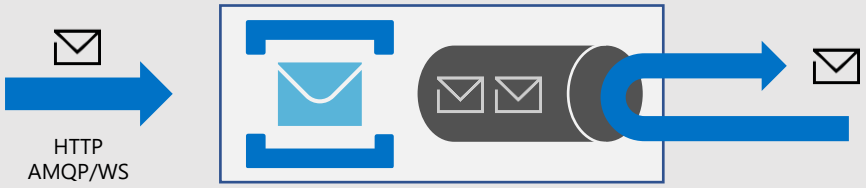
Relay



Discovery and connectivity service for securely bridging streams across network boundaries in hybrid edge/cloud scenarios.

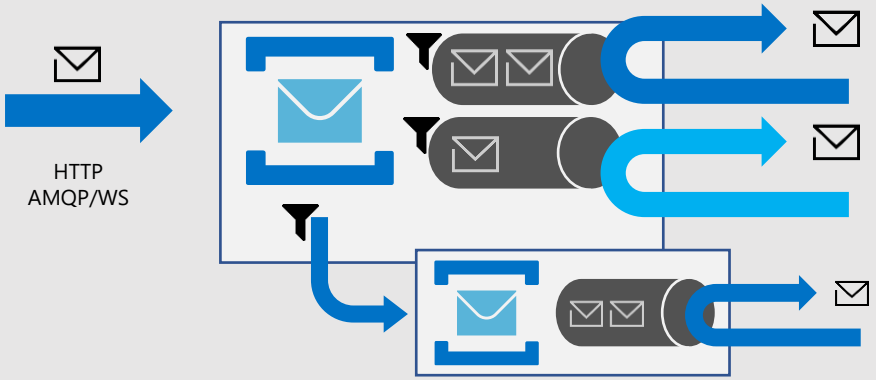
Service Bus Architectural Patterns

Queues



- Assignment of work with load-aware balancing
- Load-leveling for “spiky” workload traffic shapes
- Transactional, once-and-only-once processing
- Multiplex handling of in-order message sequences
- Deduplication, deferral, and “poison” handling

Topics



- All of the above, plus:
- Copies to 100s of concurrent subscribers
- Filter rules and message markup
- Message routing

Service Bus is a “swiss army knife” for messaging-driven workloads.

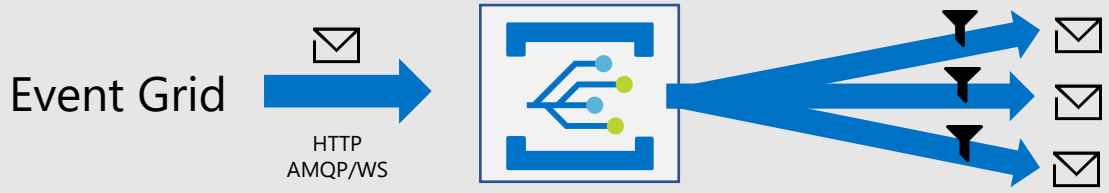
Event Hubs Architectural Patterns

Event Hub
(IoT Hub)

- Ingestion and storage of large event streams
  - > 2 Gigabyte per second if required
- Separation of event streams into partitions
- Client-chosen offsets into event stream allow arbitrary reads and re-reads during retention
- Retention of raw event data from 1 up to 90 days
- Automated archival into Avro containers for subsequent batch-style processing
- Publisher policies for data origin attestation and access control

Event Hubs is a high-scale, high-availability, multi-protocol event stream engine used for collecting and consolidating events for real-time analytics and other high-throughput computations

Event Grid Architectural Patterns



- Ingestion and push-style distribution of discrete events (events not correlated into streams) to interested subscribers.
- Per-subscriber application of simple and complex filters to select particular events of interest
- Abuse protection for event publishers
- Event schema mapping and support for CNCF CloudEvents 1.0 standard and bindings
- Multitenancy support for SaaS applications.
- Simple integration with a catalog of available event sources and sinks.

Event Grid is the Azure-wide eventing backplane for distributing and handling discrete events raised at the platform level, by custom applications, and by partner platforms.

Messaging Patterns

Jobs and Commands

Events

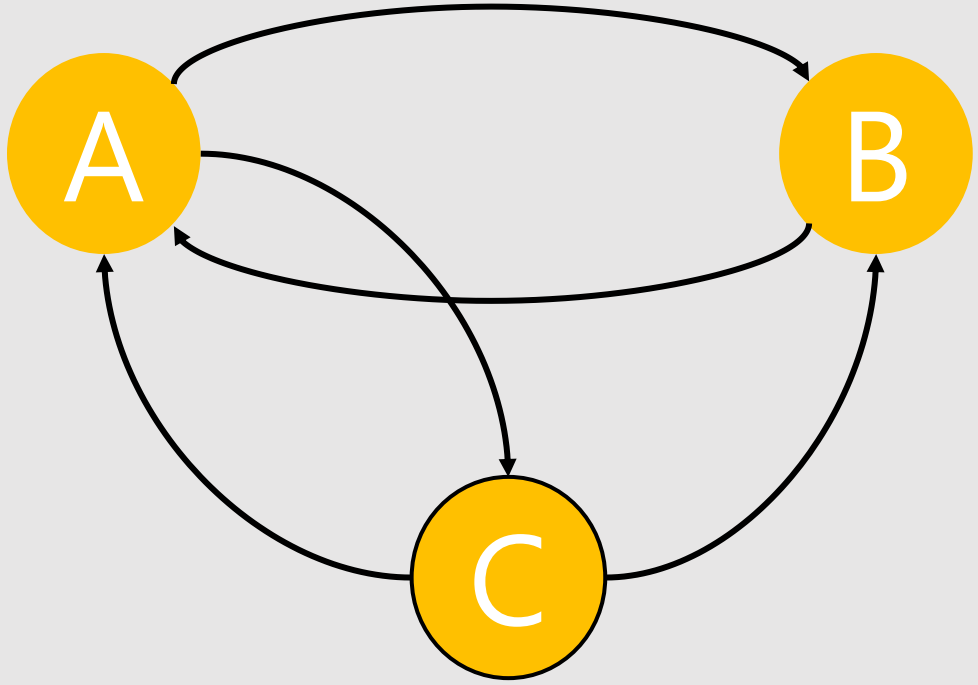
Intents

Facts

Expectations
Conversations
Contracts
Control Transfer
Value Transfer

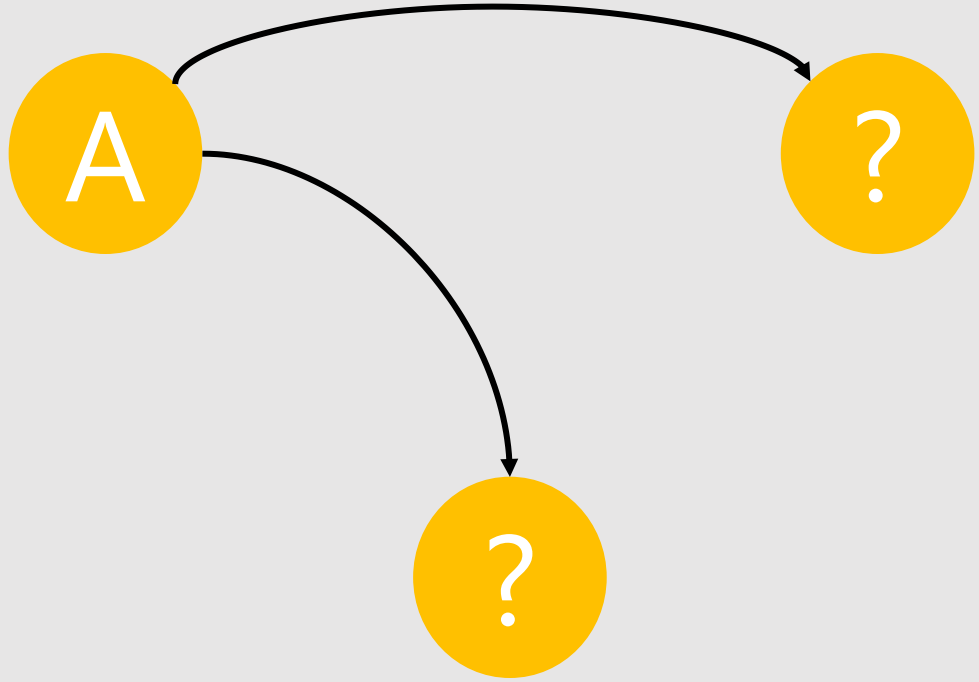
History
Context
Order

Jobs and Commands



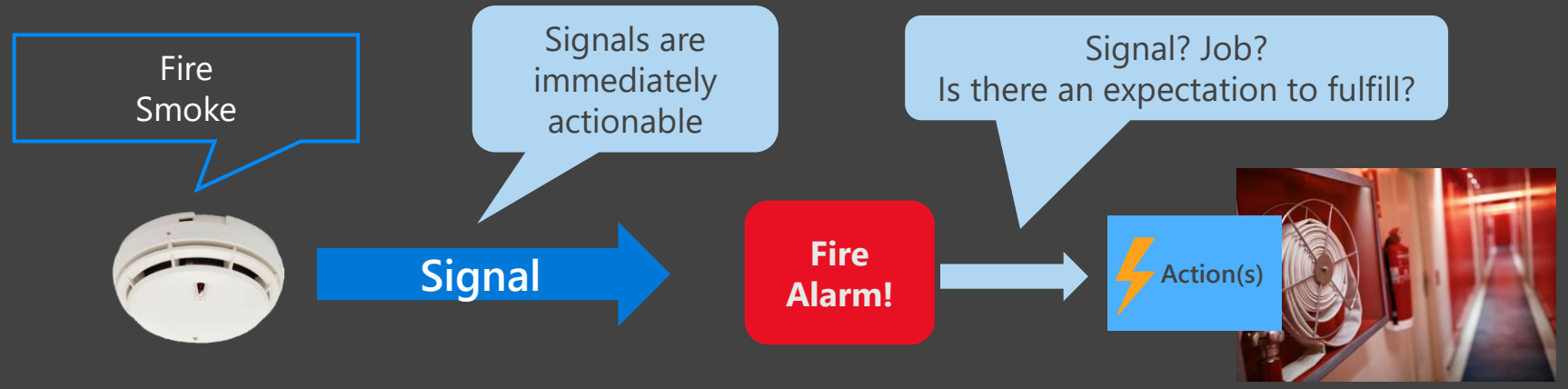
One message, one receiver

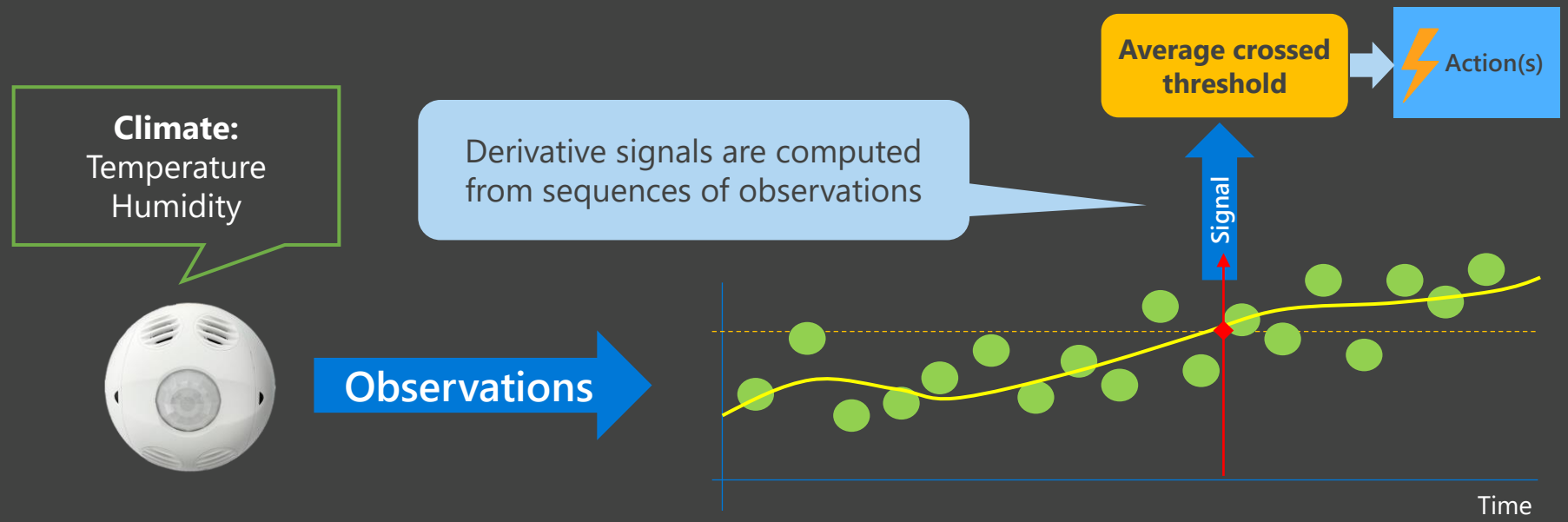
Events



One event, 0 to many receivers

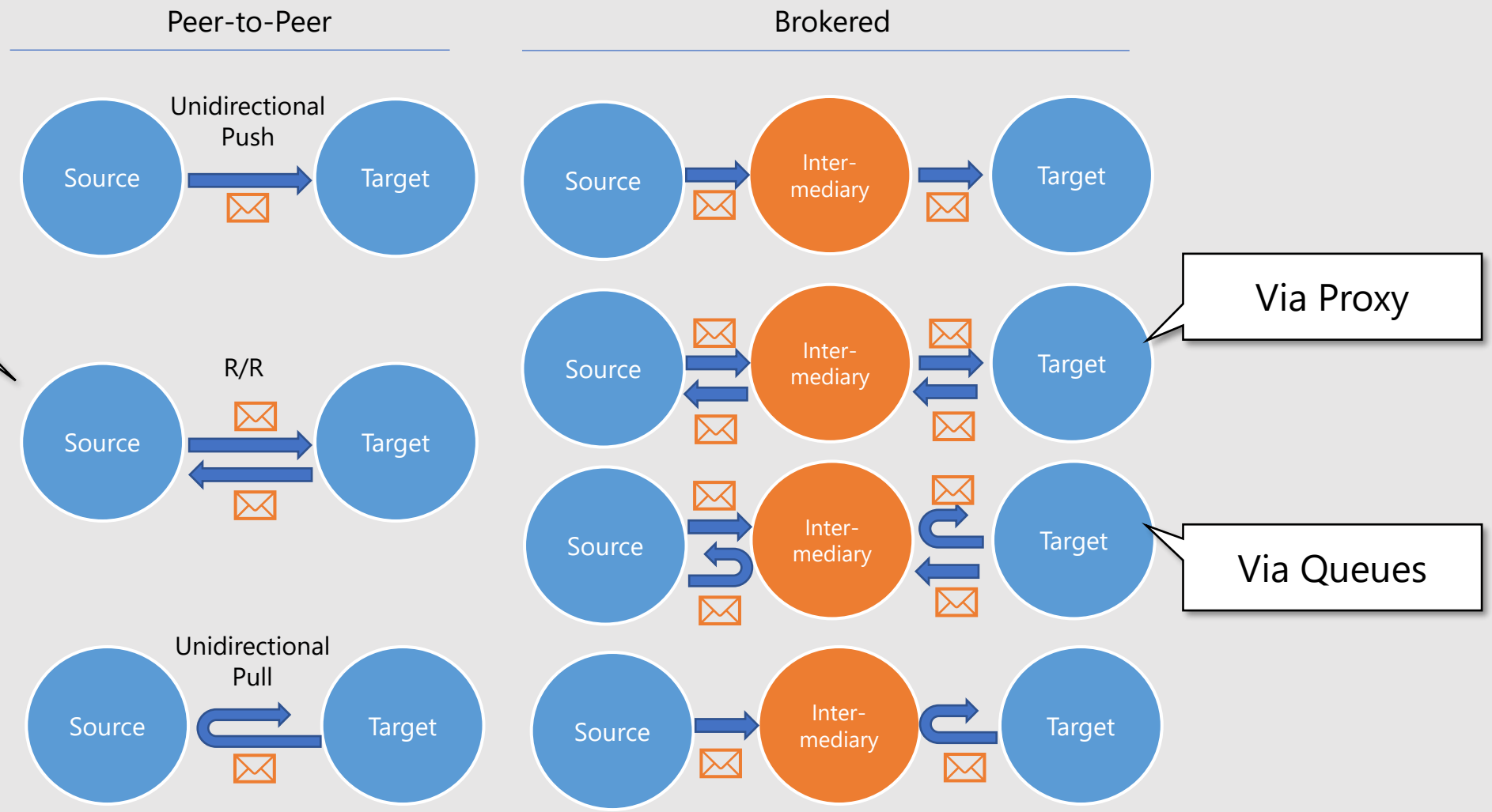
Events: Observations, Signals, Jobs

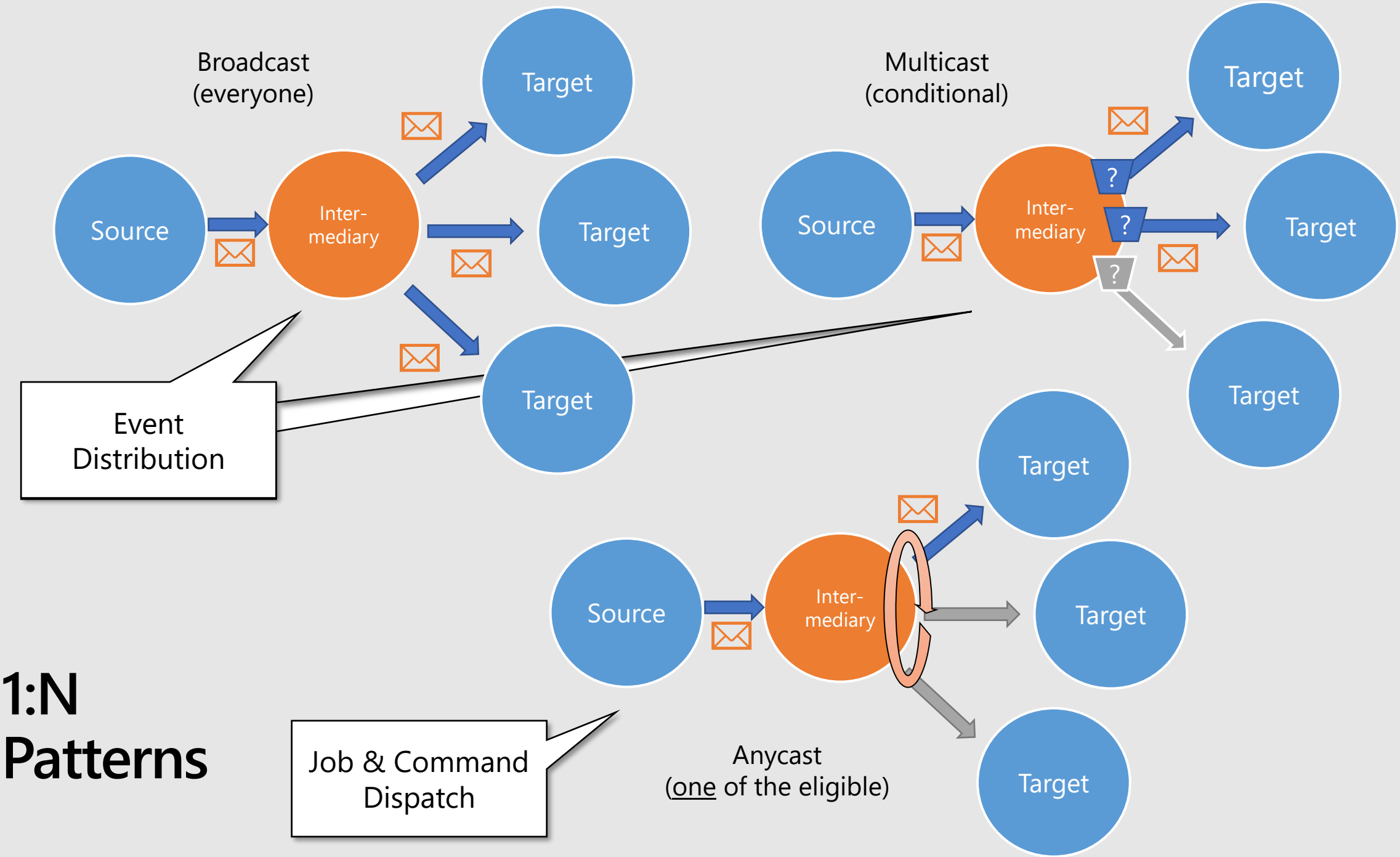




HTTP & RPC

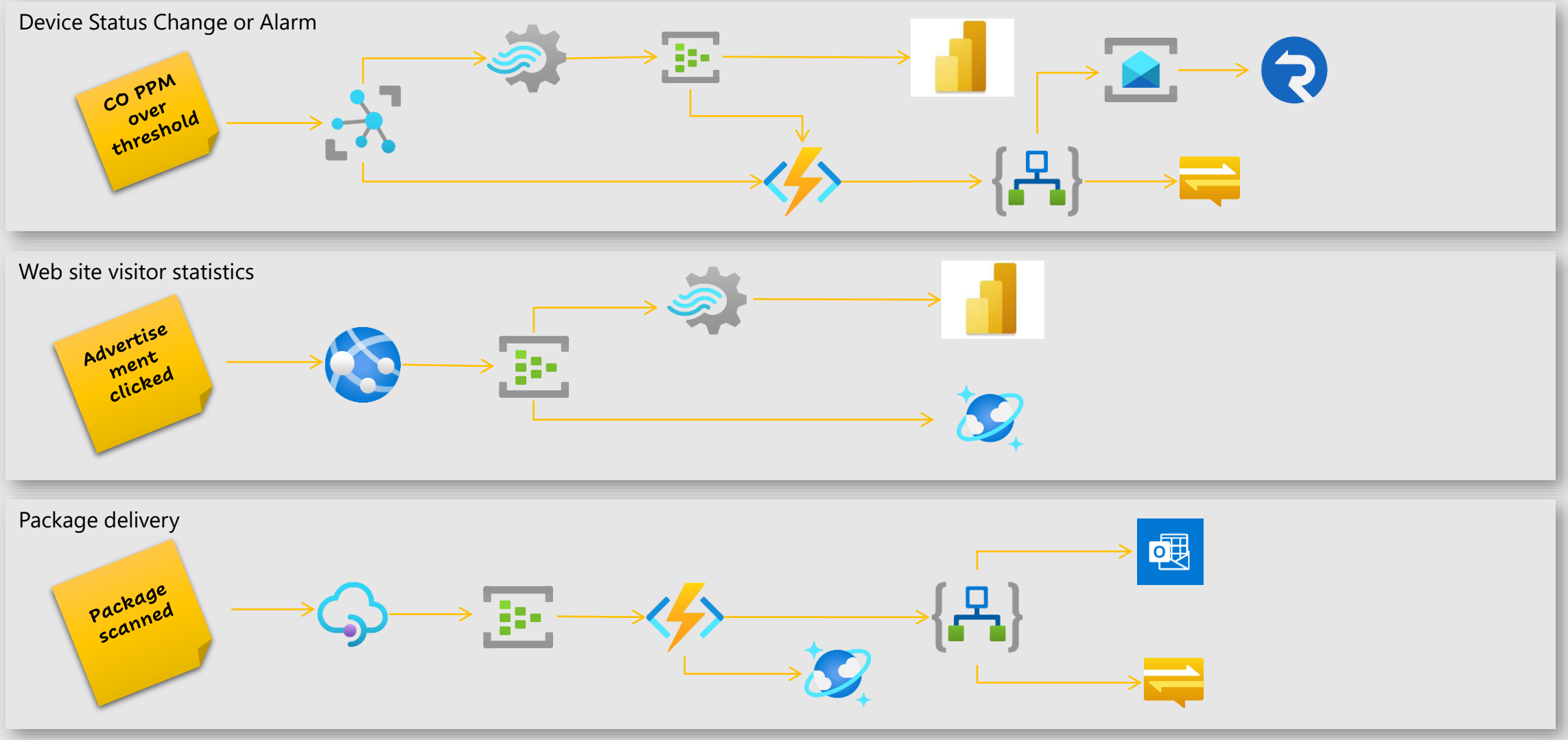
1:1 Patterns




Broadcast
(everyone)
Source
Inter-
mediary
Target
Target
Target
Multicast
(conditional)
Source
Inter-
mediary
?
?
?
Target
Target
Target
Event
Distribution
Source
Inter-
mediary
Target
Target
Target
Anycast
(one of the eligible)
Job & Command
Dispatch

1:N Patterns

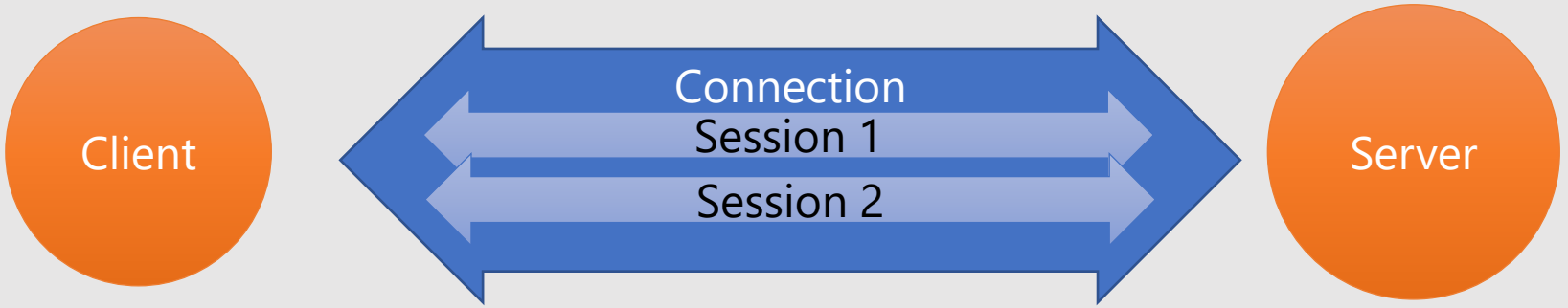
Beyond a single hop: Event Journeys



Symmetry

- A protocol is **symmetric** when is allows all of its supported gestures (except for connection establishment) independent of who initiated the connection.

Multiplexing



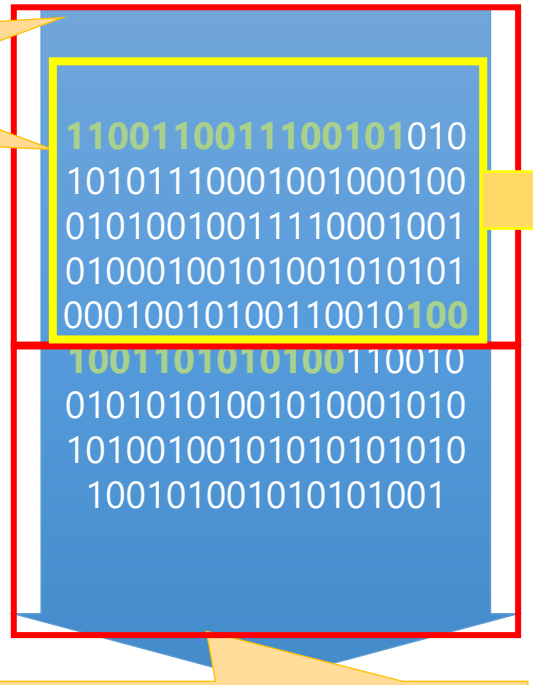
- **Multiplexing** allows a singular network connection to be used for multiple concurrent communication sessions (or links)
- Establishing connections can be enormously costly, multiplexing saves the effort for further connections between parties

Framing, Encoding, Data Layout

Framing

Frame "header"

Frame "body"



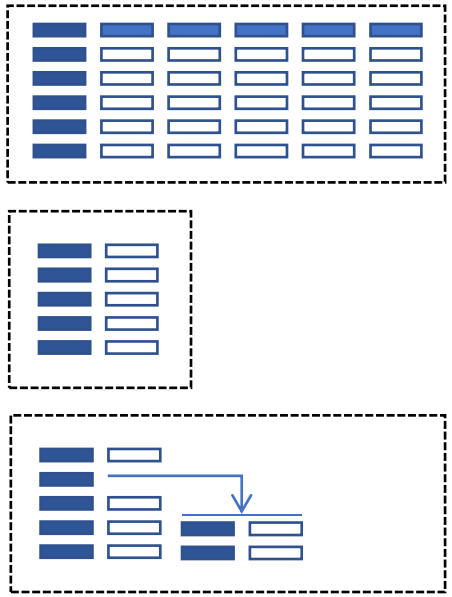
A message or framing protocol splits the data stream into distinct chunks that can be processed in sequence or separately.
A frame/message header contains information about the size, content, destination, and often also an expression of the the sender's intent

Encoding

JSON	Avro	Msg Pack
XML	AMQP	CSV
Proto	Text	Raw

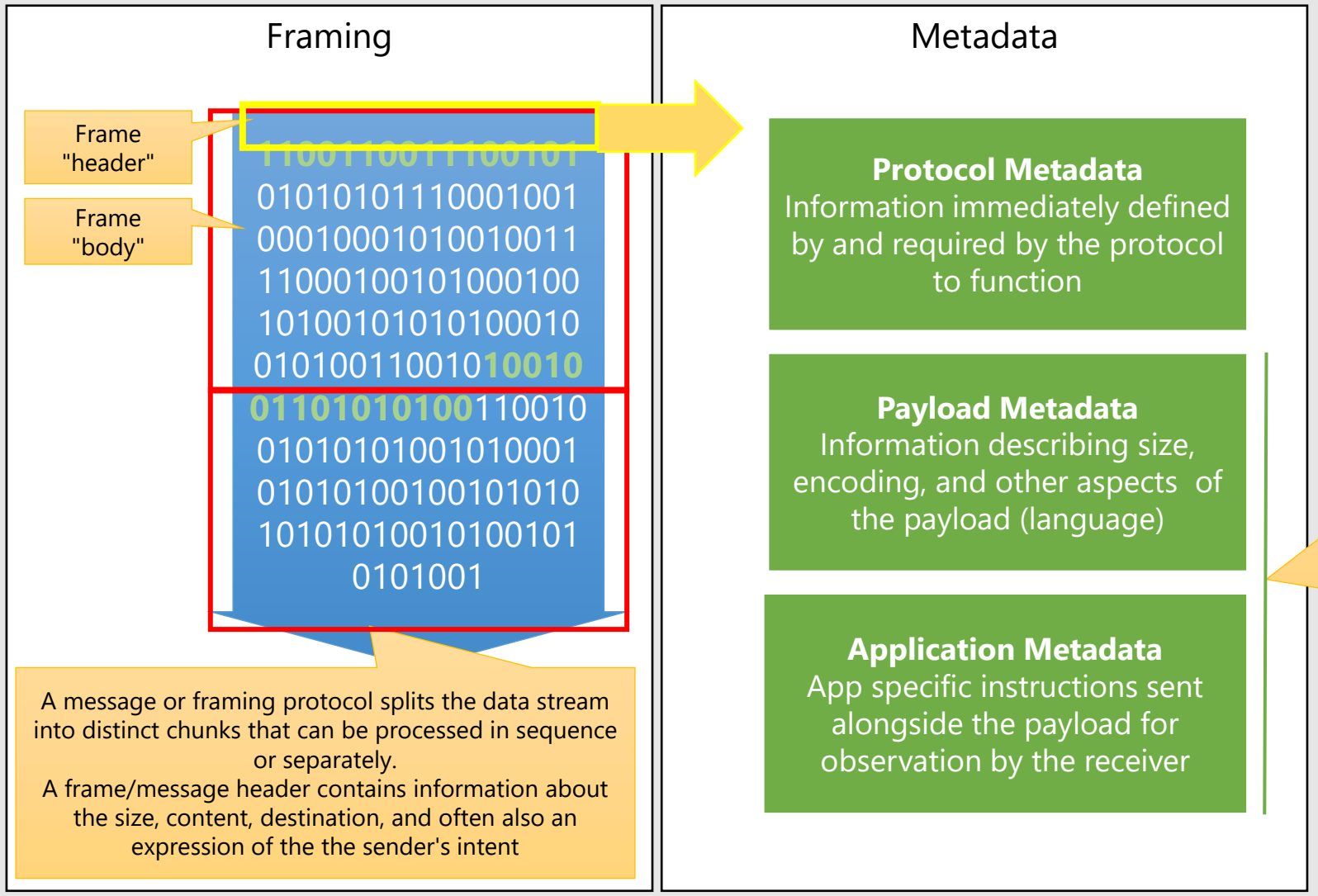
The encoding (or media-type) tells a message processor how to interpret the payload data of the message.

Layout



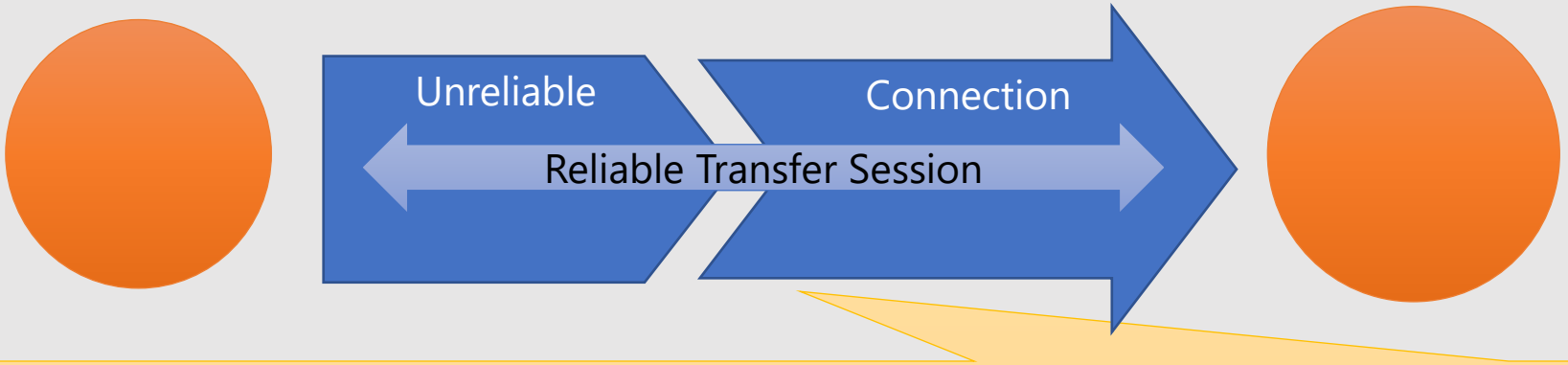
A data layout convention can tell a processor how to deal with structured data in a dynamic fashion to distinguish objects or rows/columns

Metadata



- Not all protocols allow for payload and application metadata, requiring externally agreed conventions establishing mutual understanding of message content

Transfer Assurances



- Reliable protocols allow transfer of frames more reliably than underlying protocol layers
  - Compensating for data loss, preventing duplication, ensuring order
- Various strategies to compensate for data loss
  - Resend on negative acknowledgment („data didn't get here")
  - Resend on absence of acknowledgment
  - Send duplicates of frames
- **Common Transfer Assurances**
  - "Best Effort" or "At Most Once" – no resend, not reliable
  - "At Least Once" – frame is resent until it is understood that is has been delivered at least once
  - "Exactly Once" – frame is delivered exactly once [see next]

Standards

Application Protocols and APIs

Our Standardization Engagement Principles

Open door development whenever possible (ex. W3C, CNCF)

Develop in public

Share with public

Work product royalty free

Consortium standards orgs case-by-case (ex. OASIS, OPC UA)

Develop in private

Share with public when finalized

Work product royalty free or RAND at minimum

Standards are not useful unless they're broadly accessible to anyone interested in implementing or otherwise working with them.

Why should YOU care?

Betting on standards.

Using a common, standardized message model increases the interoperability between parts of your own applications.

The CloudEvents project already did the work of figuring out what the essential attributes of an event are and how those best map to common transports and protocols

Using standard APIs makes it easier to switch between broker products, e.g. migrate from on-premises to the cloud

Using standard protocols allows tapping into a broad ecosystem of client software for most runtimes and accommodates many frameworks.

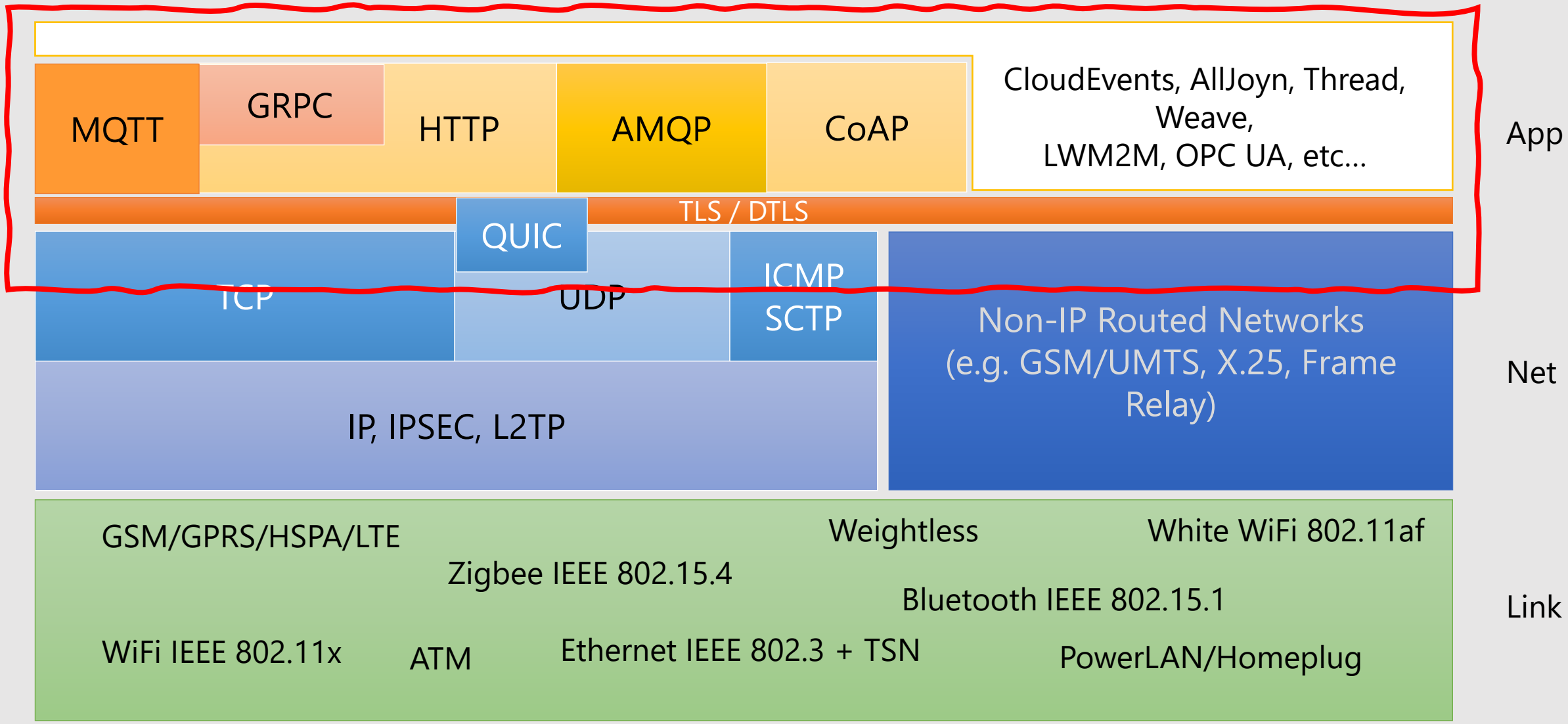
What is a Protocol?

A (network-) protocol defines rules and conventions to allow flow of information from some party to another party (and back, sometimes)

Making information flow between parties can get very complicated, so the job of making that happen is split up into layers that each focus on specific aspects, such as representing bits as radio waves.

The layers usually compose such that higher, more abstract concepts provide a choice of which lower layer to leverage, like Internet protocol routing over wired connections or radio.

Application Protocols

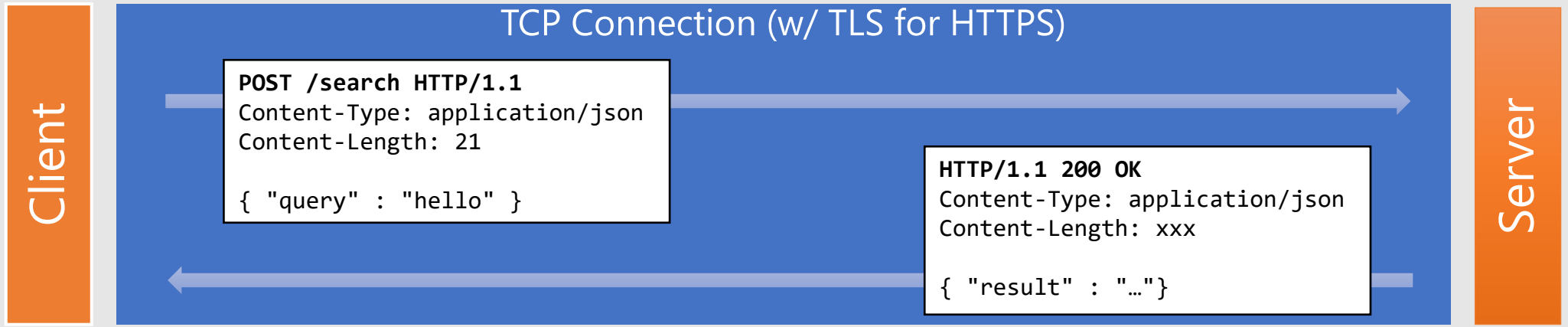


HTTP 1.1

Patterns	ReqResp
Symmetric	No
Multiplexing	No
Encodings	Variable
Metadata	Yes
Assurances	-

HTTP 1.1 is the Application Protocol for the web

- Simple structure, text based, ubiquitious
- Client-initiated (asymmetric) request/response flow
- No multiplexing
- HTTP embodies the principles of "Representational State Transfer".
- REST is not a protocol, it's the architectural foundation for the WWW.

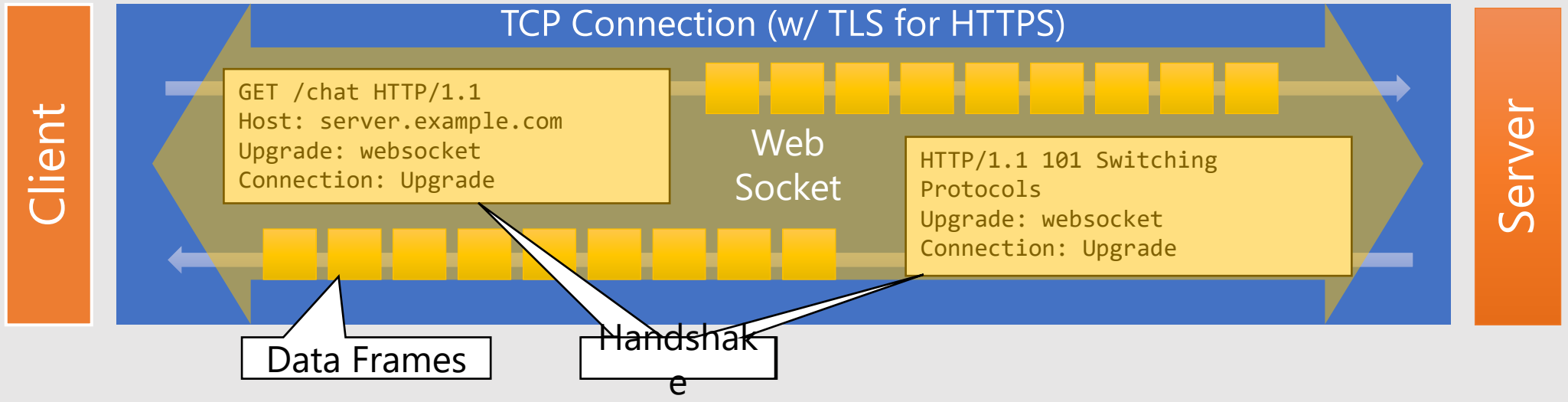


Web Sockets

Patterns	Duplex
Symmetric	No
Multiplexing	No
Encodings	Fixed
Metadata	No
Assurances	-

Web Sockets is a Stream Tunneling Protocol

- Allows using the HTTP 1.1 port (practically only HTTPS) for bi-directional, non-HTTP stream transfer
- Web Sockets by itself is neither a Messaging or an Application Protocol, as it defines no encoding or semantics for the stream.
- Web Sockets can tunnel AMQP, MQTT, CoAP/TCP, etc.

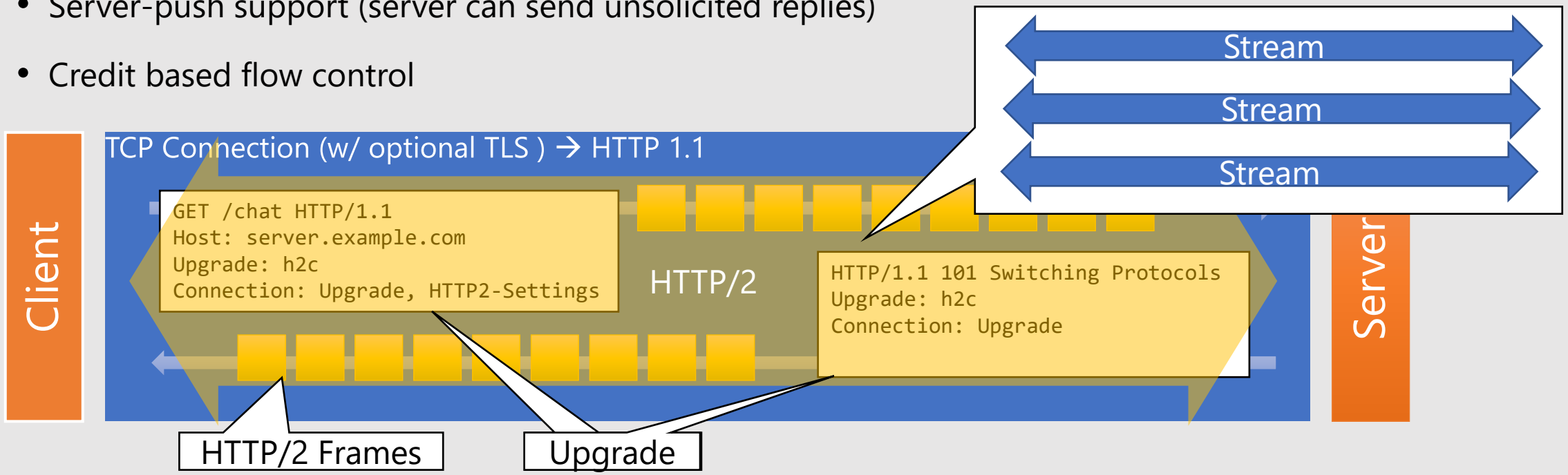


HTTP/2 (+ GRPC)

Patterns	RR, OW/SC
Symmetric	No
Multiplexing	Yes
Encodings	Variable
Metadata	Yes
Assurances	-

HTTP/2 is an **Application Protocol**; successor of HTTP 1.1

- Same semantics and message model, different implementation
- Multiplexing support, binary standard headers, header compression.
- Uses Web Socket like upgrade for backward compatible integration with HTTP 1.1, no WS support
- Server-push support (server can send unsolicited replies)
- Credit based flow control

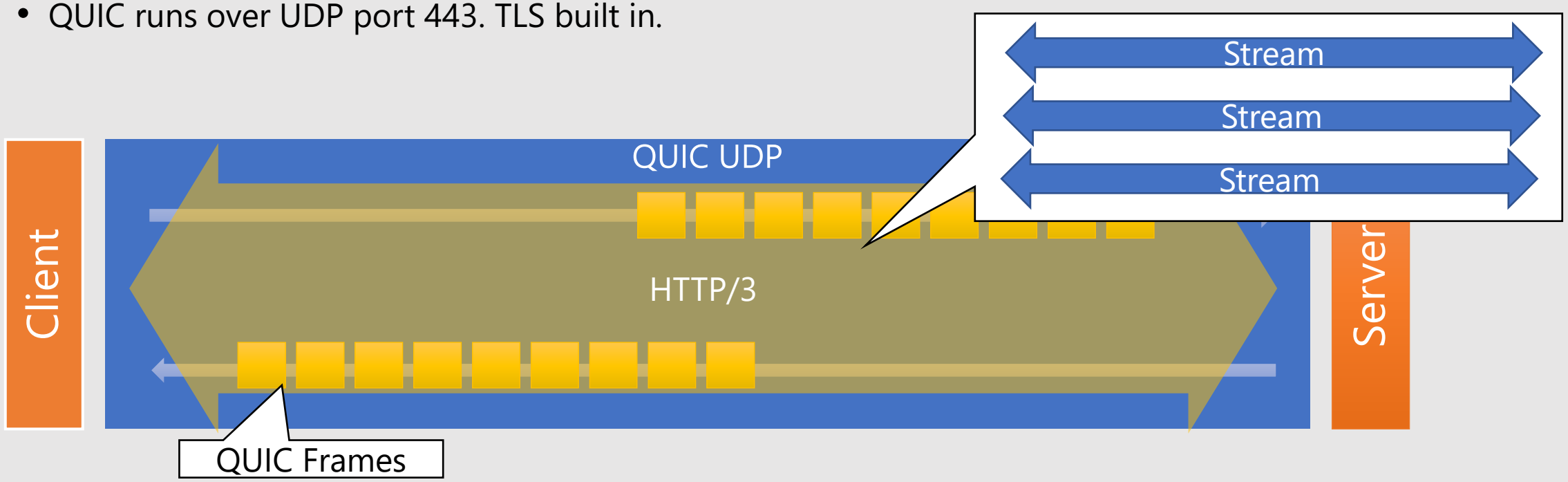


HTTP/3

Patterns	RR, OW/SC
Symmetric	No
Multiplexing	Yes
Encodings	Variable
Metadata	Yes
Assurances	-

HTTP/3 is an **Application Protocol**; coexists with HTTP/2

- Same semantics and message model, different implementation
- Largely a redo of HTTP/2, moving to QUIC
- Multiplexing support via QUIC (UDP Streams), binary standard headers, header compression.
- QUIC runs over UDP port 443. TLS built in.



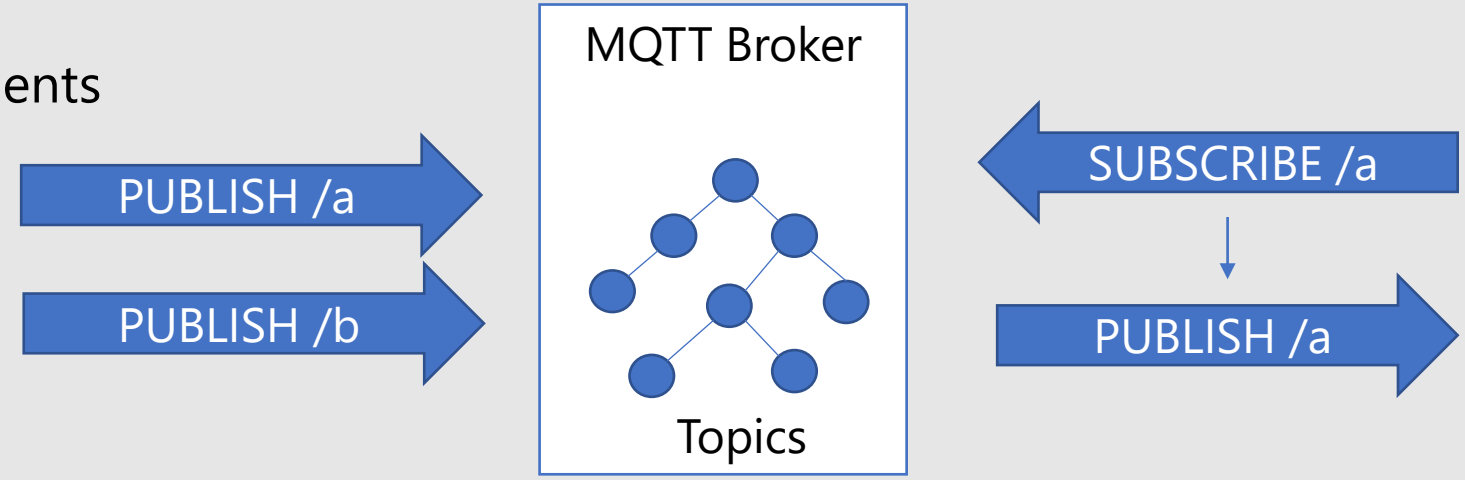
OASIS MQTT

MQTT is a lightweight **Publish and Subscribe Protocol**

- Easy to implement for publishers and subscribers, assumes broker
- Rigorously optimized for minimizing wire overhead
- Publish/Subscribe gestures are explicit elements of the protocol; “subscribe” = “receive”
- Often used for submitting and subscribing to telemetry and sharing state changes amongst peers, can model request/reply routes on top

MQTT 3.1.1 most used

MQTT 5.0 new with many improvements



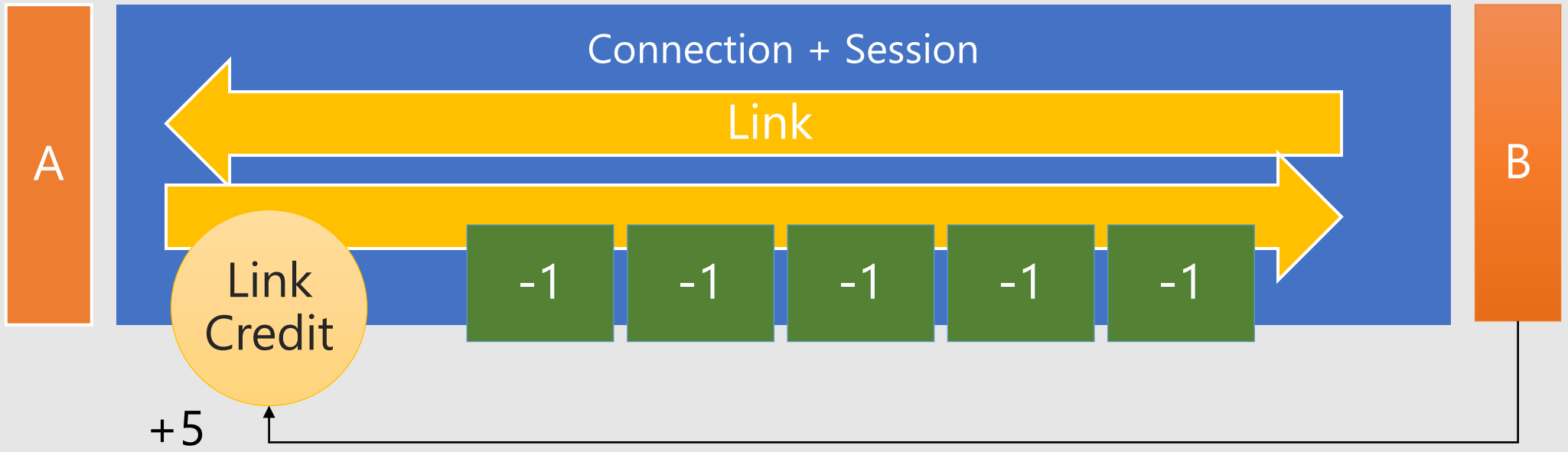
OASIS AMQP

Advanced Message Queuing Protocol

AMQP 1.0 is a symmetric, reliable **Message Transfer Protocol**
with support for multiplexing and flow control

- Supports queuing, pub/sub, filters, one-way, request-response, streams
- No topology assumptions, multi-hop routing facilities

AMQP 0.9 (what RabbitMQ uses) is an expired draft with a fixed topology model



Azure Compatible, Generic AMQP 1.0 Clients

Client Stack	URL
Apache Qpid Proton-C (Go, Python, Ruby, C++)	https://qpid.apache.org/proton/
Apache Qpid Messaging API	https://qpid.apache.org/components/messaging-api
Apache Qpid Proton-J	https://qpid.apache.org/proton/
Apache NMS AMQP	https://activemq.apache.org/components/nms/providers/amqp/
AMQP .NET Lite (all variants of .NET, incl. Nano & Micro)	https://github.com/Azure/amqpnetlite
Azure AMQP (our own server stack)	https://github.com/Azure/azure-amqp
Azure uAMQP C (Python, PHP)	https://github.com/Azure/azure-uamqp-c https://github.com/Azure/azure-uamqp-python
Rhea (NodeJS)	https://github.com/amqp/rhea
Go AMQP	https://github.com/Azure/go-amqp
Vert.X AMQP Client	https://vertx.io/docs/vertx-amqp-client/java/

Apache Kafka

Patterns	Kafka
Symmetric	No
Multiplexing	No
Encodings	Variable
Metadata	Yes
Assurances	ALO

The Apache Kafka protocol is a project-specific request/response protocol for the Apache Kafka broker

- SASL preamble for authentication
- API keys identify operations, specific message types per key
- Effectively an RPC protocol tailored to Kafka's features
- Not a standard and fully controlled by the Kafka project, but quite popular

Apache Kafka® Clients

https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=30736606

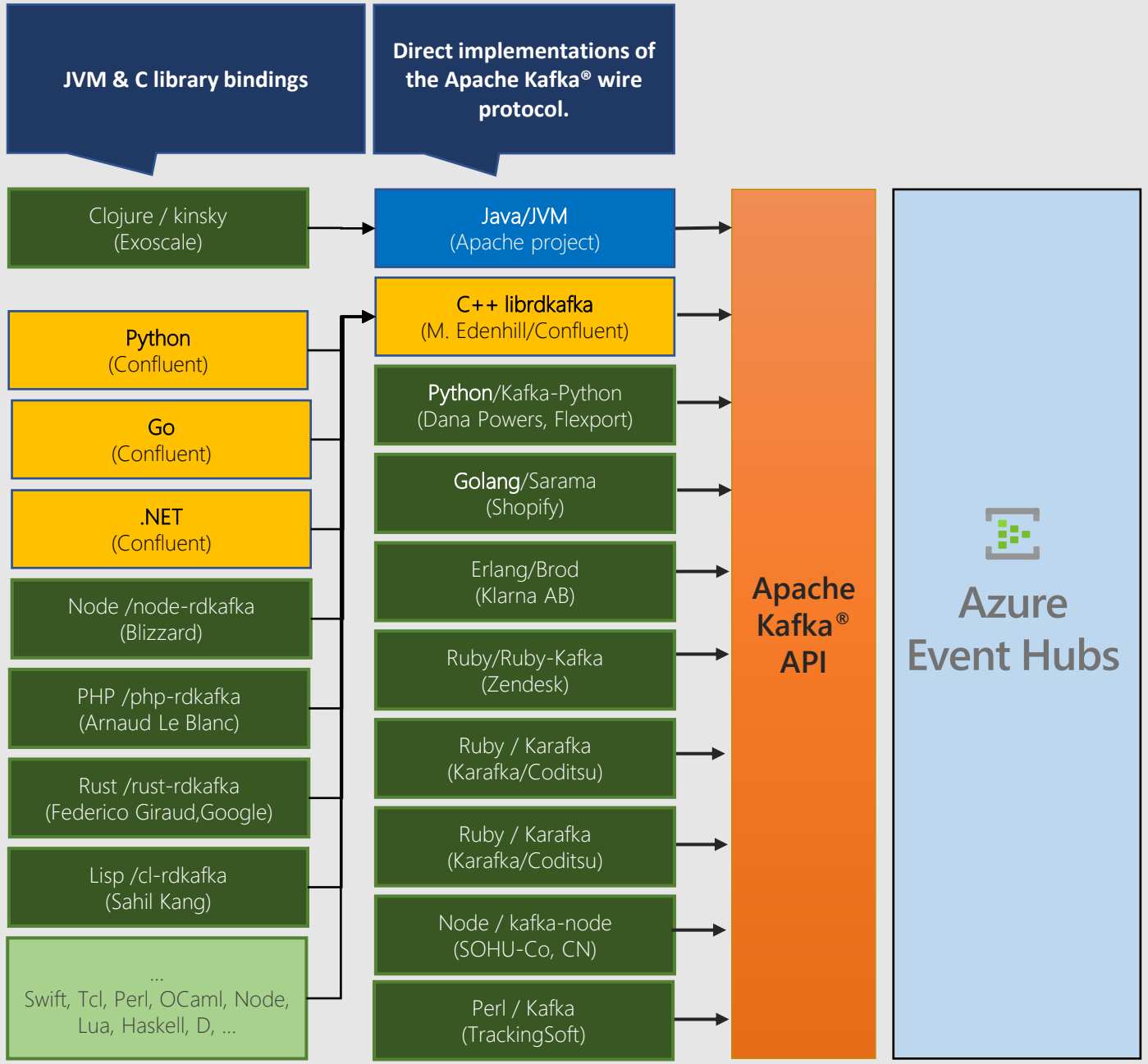
Azure Event Hubs is compatible with the producer and consumer APIs of all current Apache Kafka clients.

Only the Java/JVM client is part of the Apache Kafka project. All other clients are under various non-foundation ownerships and licenses. Azure does not provide QFE support for any of these clients but does provide support for service-side compatibility.

Apache Software Foundation

Non-Foundation, Community / Best Effort Support

Non-Foundation, Commercial Support Option



Unifying Abstractions

Java JMS 2.0 (Jakarta Messaging)

An exceptionally successful Java/JVM API standard that works with most mature, queue-based pubsub message brokers on the market today.

```
Topic topic = session.createTopic(dest.toString());

MessageProducer sender = session.createProducer(dest);

TopicSubscriber topicSubscriber1 =
    session.createDurableSubscriber(topic, sub1Name);

TopicSubscriber topicSubscriber2 =
    session.createDurableSubscriber(topic, sub2Name, "JMSCorrelationID='5'", false);

MessageConsumer topicSubscriber3 =
    session.createSharedDurableConsumer(topic, sub3Name, "JMSCorrelationID='5'");
```

```
TextMessage message = session.createTextMessage("Text!");

message.setJMSCorrelationID(Integer.toString(i));

sender.send(message, DELIVERY_MODE, Message.DEFAULT_PRIORITY,
Message.DEFAULT_TIME_TO_LIVE);
```

- Queues
- Topics
- JMS Message Selectors
- Shared Durable Subscriptions
- Unshared Durable Subscriptions
- Shared Volatile Subscriptions
- Unshared Volatile Subscriptions
- Queue Browser
- Queue Message Selectors
- Temporary Queues
- Temporary Topics
- JMS Transactions

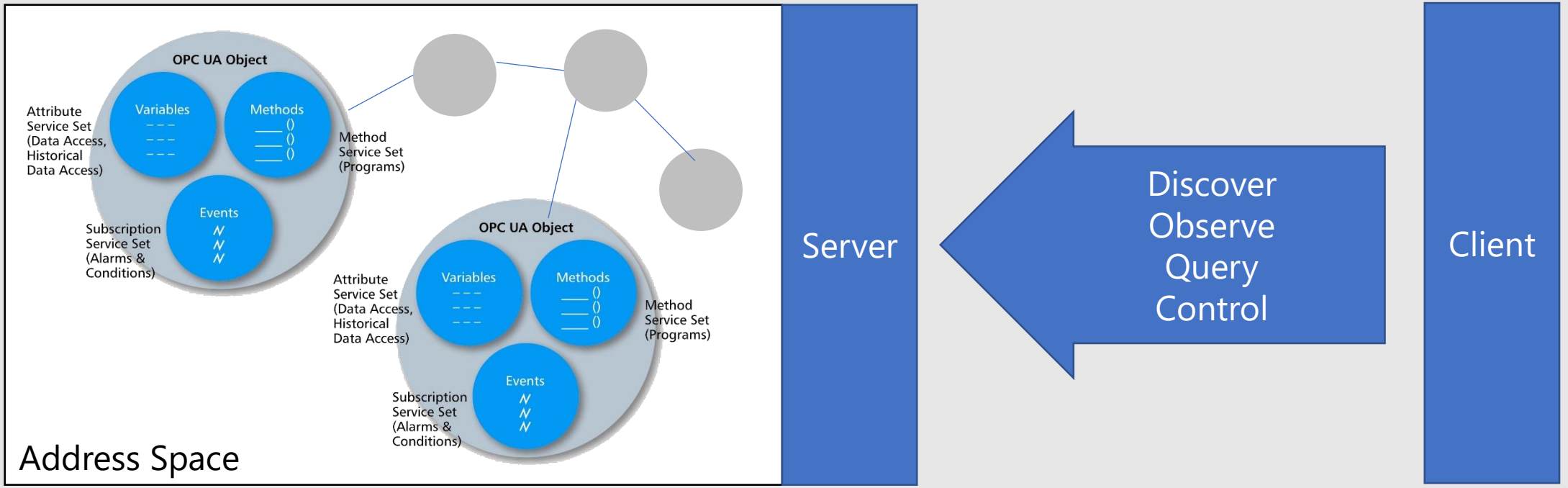
OPC UA

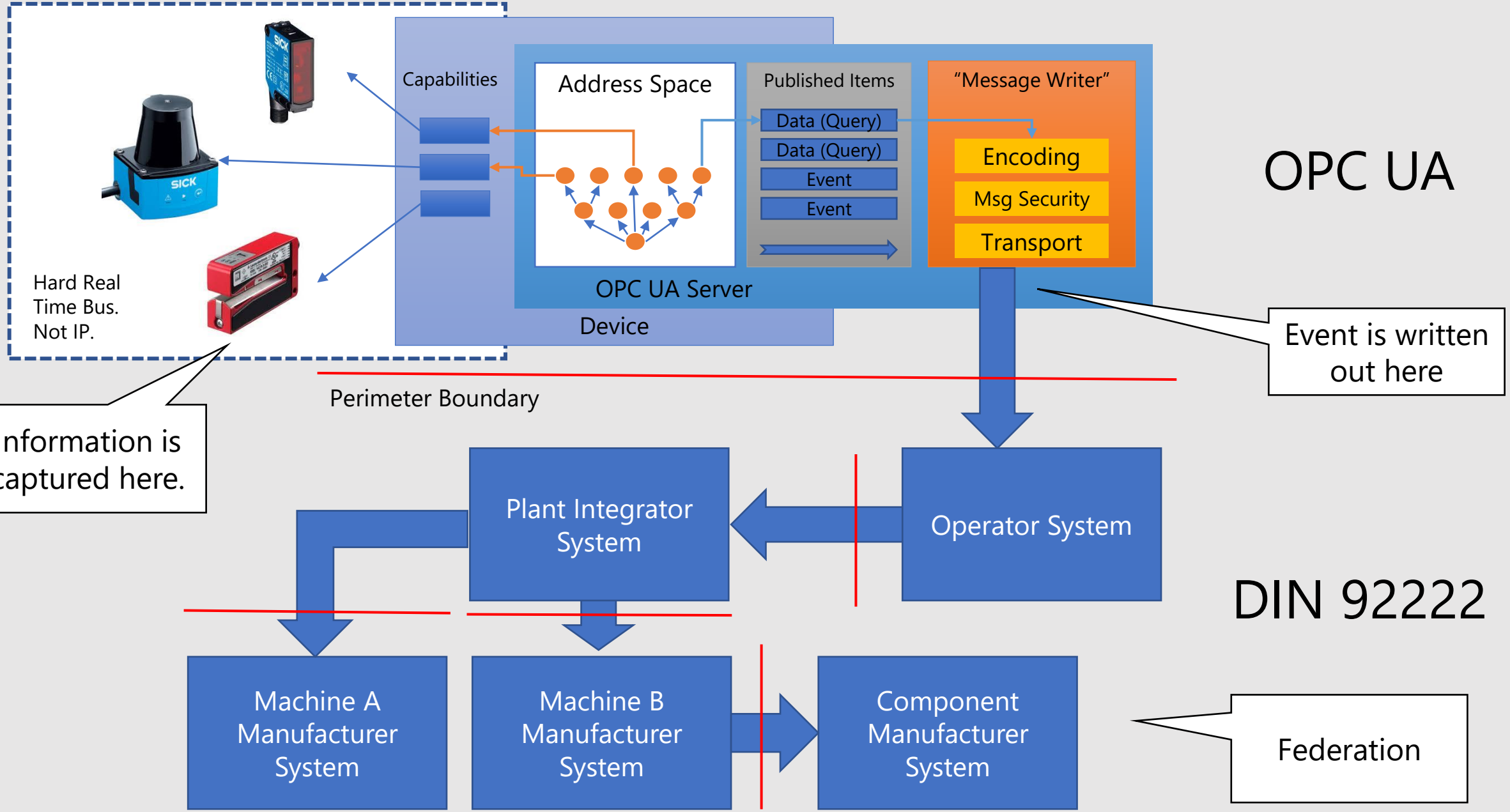
OPC Foundation Unified Architecture

OPC Foundation standard; IEC standard

Foundational architecture model for (industrial) device management and information flow

Cloud integration via Pub/Sub




Capabilities
Address Space
Published Items
"Message Writer"
Data (Query)
Data (Query)
Event
Event
Encoding
Msg Security
Transport
OPC UA Server
Device
Hard Real
Time Bus.
Not IP.
OPC UA
Event is written
out here
Perimeter Boundary
Information is
captured here.
Plant Integrator
System
Operator System
DIN 92222
Machine A
Manufacturer
System
Machine B
Manufacturer
System
Component
Manufacturer
System
Federation

CNCF CloudEvents

Event Protocol Suite developed in CNCF Serverless WG

Common metadata attributes for events

Flexibility to innovate on event semantics

Simple abstract type system mappable to different encodings

Transport options

HTTP(S) 1.1 Webhooks, also HTTP/2

MQTT 3.1.1 and 5.0

AMQP 1.0

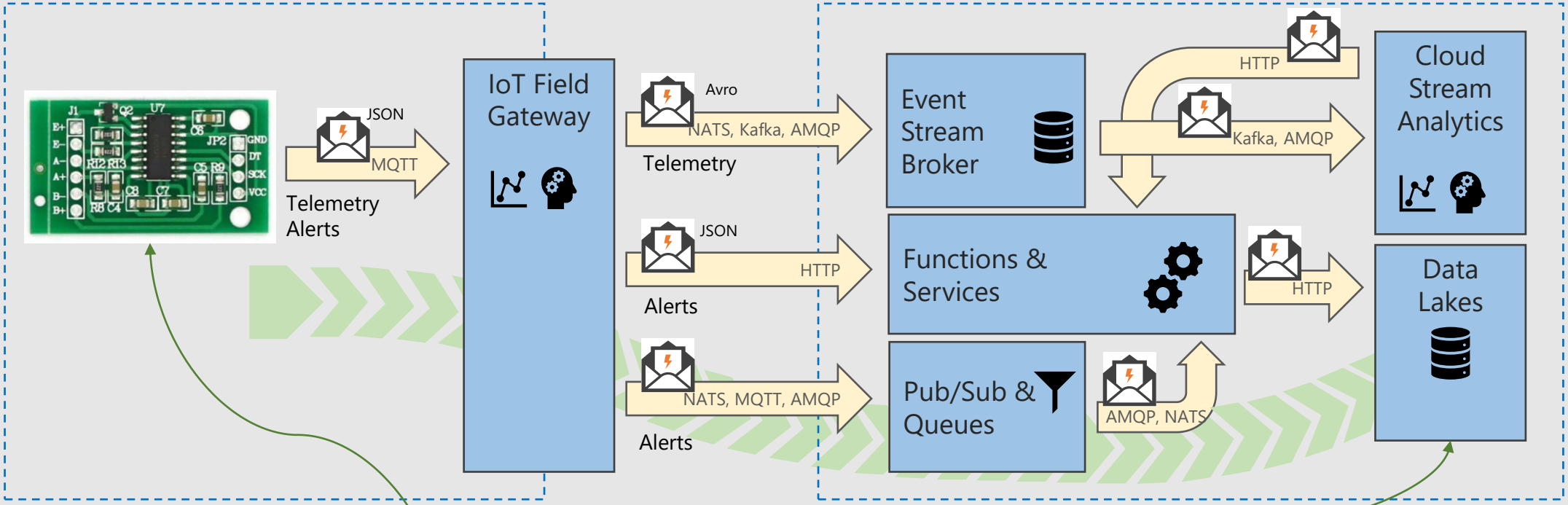
Kafka, NATS

Encoding options

JSON (required for all implementations)

Extensible for binary encodings: Avro, AMQP, etc.

Why CloudEvents?



- Event data is often routed via multiple hops and often using different protocols
- How is what gets sent here easily routed to and stored here in hybrid edge/cloud and multi-cloud systems?

Why CloudEvents?

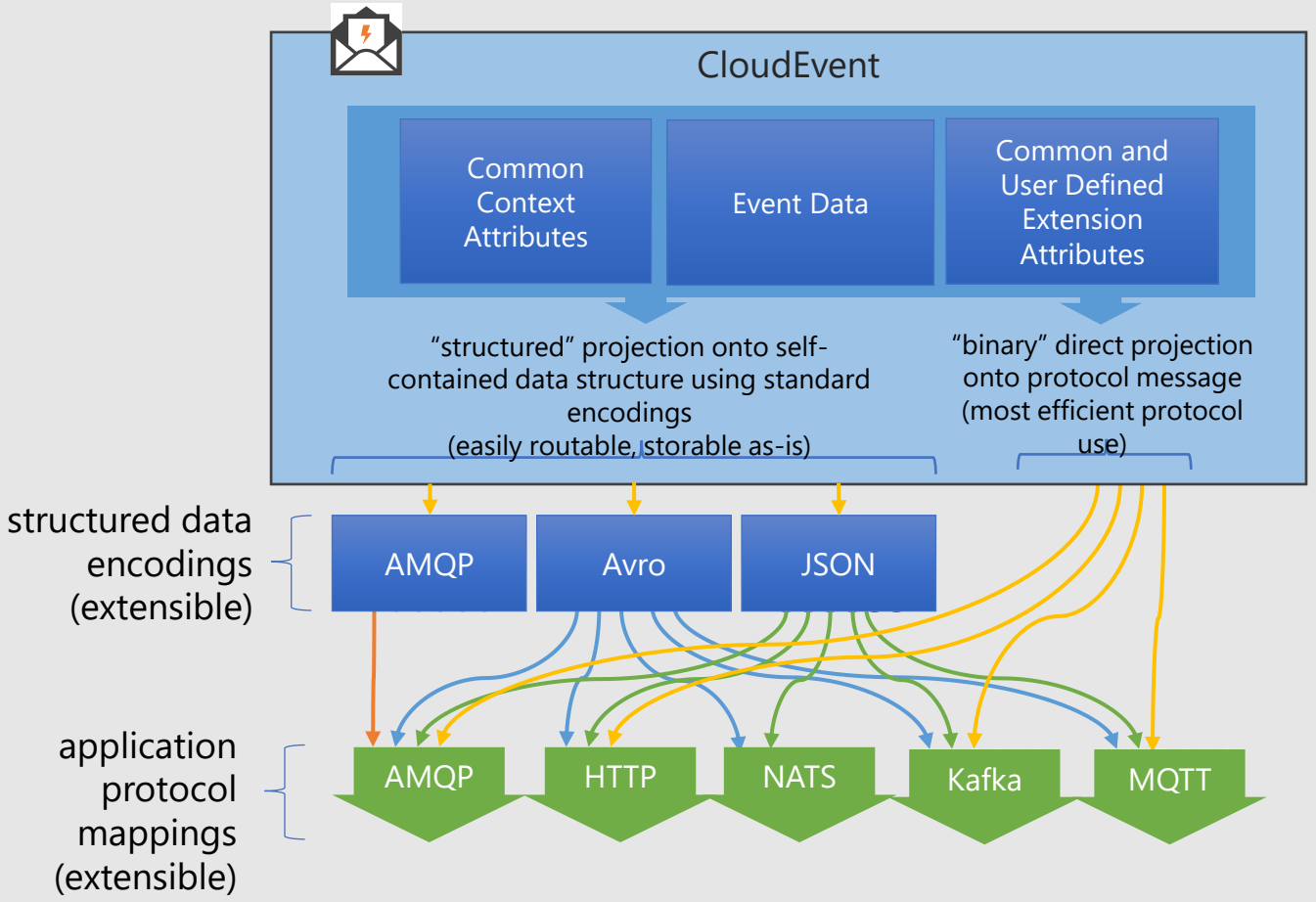
Binds to existing standard application protocols

Does not try to abstract away protocols but leverages each for its strengths

Integrates with existing messaging and eventing stacks

Leverages existing data encodings and is easy to adapt to new ones (Protobuf, CBOR, MsgPack, etc.)

Allows for protocol switching and transcoding on multi-hop routes



CloudEvents - Base Specification

CloudEvents is a lightweight common convention for events.

It's *intentionally* not a messaging model* to keep complexity low.

No reply-path indicators, no message-to-message correlation, no target address indicators, no command verbs/methods.

Metadata for handling of events by generic middleware and/or dispatchers

What kind of event is it? **type, specversion**

When was it sent? **time**

What context was it sent out of? **source, subject**

What is this event's unique identifier? **id**

What's the shape of the carried event data? **datacontenttype**, **schema**

Event data may be text-based (esp. JSON) or using some binary encoding

Event Formats

Event formats bind the abstract CloudEvents information model to specific wire encodings.

All implementation must support JSON. Avro is a supported binary format.

Further compact binary event format candidates might be CBOR, or Protobuf.

```
{
  "specversion" : "1.0",
  "type" : "myevent",
  "source" : "uri:example-com:mydevice",
  "id" : "A234-1234-1234",
  "time" : "2018-04-05T17:31:00Z",
  "datacontenttype" : "text/plain",
  "data" : "Hello"
}
```

JSON Representation

Example

HTTP - Binary

```
POST /event HTTP/1.0
Host: example.com
Content-Type: application/json
ce-specversion: 1.0
ce-type: com.bigco.newItem
ce-source: http://bigco.com/repo
ce-id: 610b6dd4-c85d-417b-b58f-3771e532

{
  "action": "newItem",
  "itemID": "93"
}
```

HTTP - Structured

```
POST /event HTTP/1.0
Host: example.com
Content-Type: application/cloudevents+json

{
  "specversion": "1.0",
  "type": "com.bigco.newItem",
  "source": "http://bigco.com/repo",
  "id": "610b6dd4-c85d-417b-b58f-3771e532",
  "datacontenttype": "application/json",
  "data": {
    "action": "newItem",
    "itemID": "93"
  }
}
```

Transport Bindings

HTTP 1.1, HTTP/2, HTTP/3:

Binds to the HTTP message

Binary and structured modes

AMQP:

Binds event to the AMQP message

Binary and structured modes

MQTT:

Binds event to MQTT PUBLISH frame.

Binary and Structured for MQTT v5

Structured mode only for MQTT v3.1.1

NATS:

Binds event to the NATS message.

Structured mode only

Apache Kafka:

Binds to the Kafka message

Structured and binary mode

Protocol bindings directly map onto the protocol's message structure and using protocol semantics. Accepts that protocols are different.

Binary mode: Event metadata projected onto the protocol message metadata, event data onto the protocol message payload

Structured mode: Event is self-contained as an encoded byte stream, metadata may be promoted (duplicated) into protocol message metadata.

Discovery, Catalog, Subscriptions

Cloud Events Futures

Service Bus & Azure Queues

Cloud messaging

- AMQP
- HTTP
- Cloud-Events
- JMS 2.0

Event Hubs

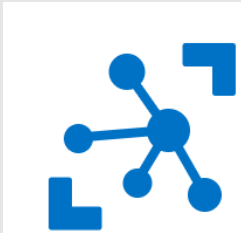
Telemetry stream ingestion

- AMQP
- HTTP
- Kafka
- Cloud-Events

Event Grid

Event distribution

- AMQP
- HTTP
- Cloud-Events



IoT Hub

IoT messaging and manage-

- AMQP
- MQTT
- OPCUA
- Cloud-Events

Relay

Discovery, Firewall/NAT Traversal

- HTTP
- Web-Sockets
- Cloud-Events