Microsoft

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

Microsoft Azure Messaging

Agenda

- Objective: Understand the messaging standards landscape and why you have choices

- Messaging Patterns and Protocol Characteristics
  - Push/Pull, Queues, Ingestors/Streams, Routers/Distributors
- Messaging Protocol Standards
  - MQTT, AMQP, HTTP, Kafka, COAP, gRPC, OPC/TCP, OPC/UADP
- Encoding Standards
  - XML, CSV, JSON, CBOR, Avro, Thrift, Protobuf
- Abstraction Standards
  - OPC UA, CNCF CloudEvents

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)*

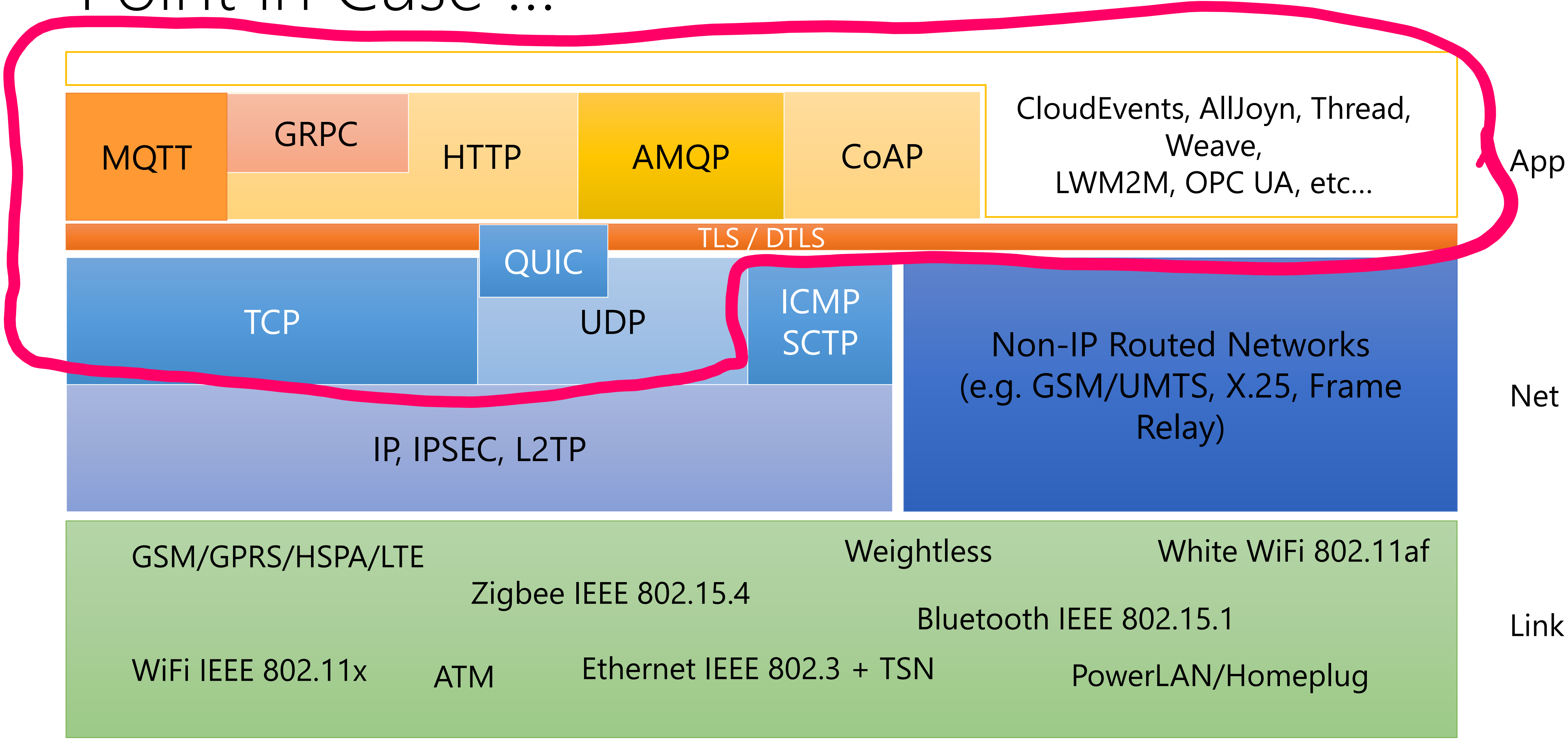
- Making information flow between parties can get very complicated, so the job of making that happen is split up into layers that each focus on specific aspects, such as representing bits as radio waves.

- 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.

“The nice thing about standards is that you have so many to choose from”

Andrew S. Tanenbaum, *Computer Networks*, 2nd ed., p. 254.

Point in Case …



MQTT

GRPC

HTTP

AMQP

CoAP

CloudEvents, AllJoyn, Thread, Weave, LWM2M, OPC UA, etc…

App

TLS / DTLS

QUIC

TCP

UDP

ICMP
SCTP

IP, IPSEC, L2TP

Non-IP Routed Networks (e.g. GSM/UMTS, X.25, Frame Relay)

Net

GSM/GPRS/HSPA/LTE

Weightless

White WiFi 802.11af

Zigbee IEEE 802.15.4

Bluetooth IEEE 802.15.1

WiFi IEEE 802.11x

ATM

Ethernet IEEE 802.3 + TSN

PowerLAN/Homeplug

Link

Patterns

Messaging

Intents

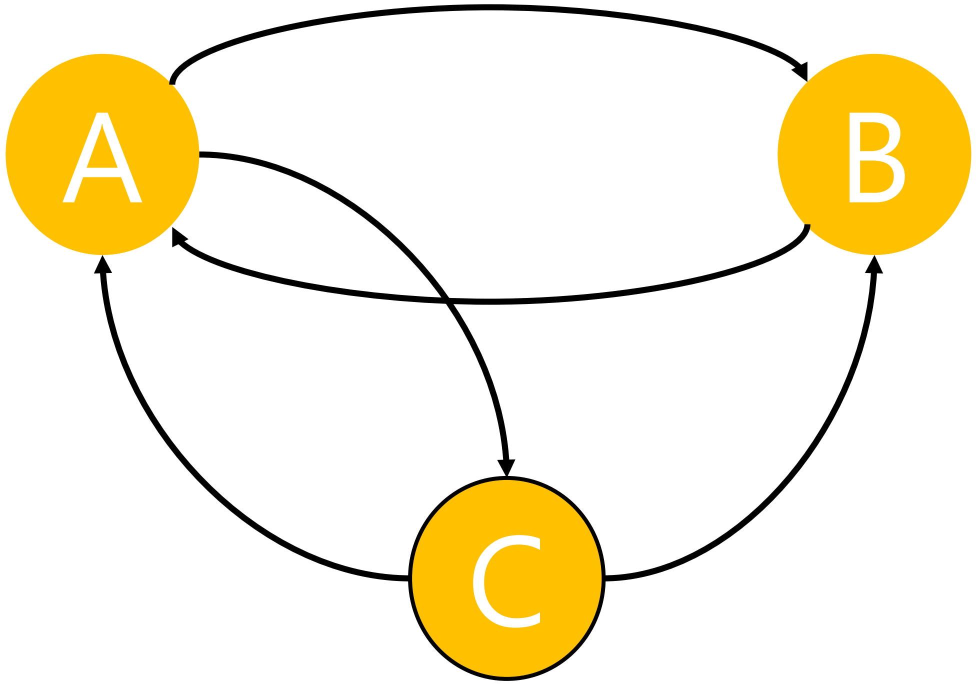
Expectations
Conversations
Contracts
Control Transfer
Value Transfer

Eventing

Facts

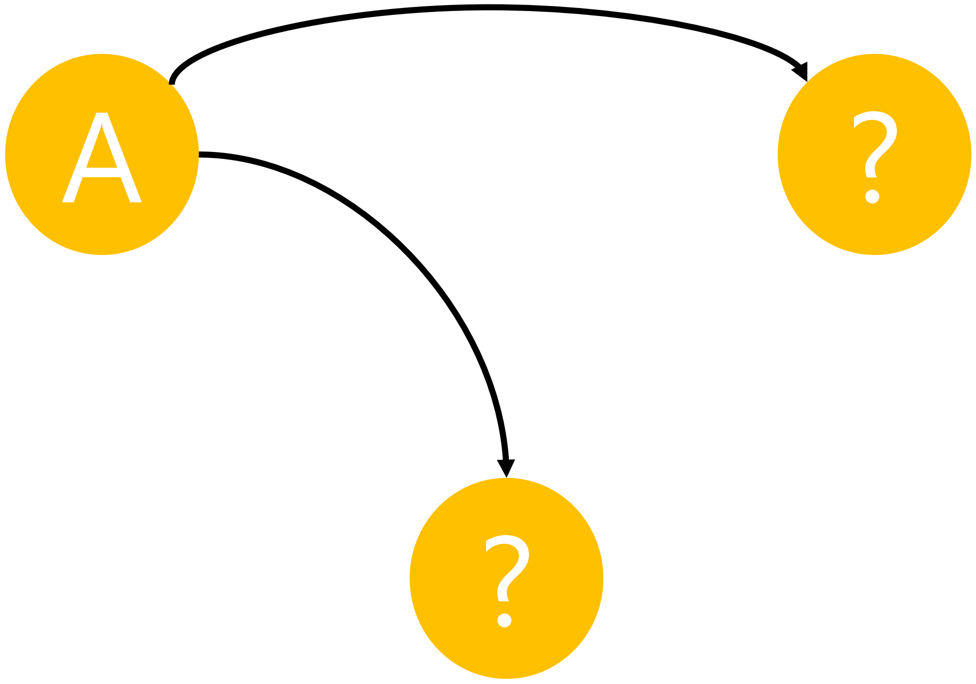
History
Context
Order

Messaging



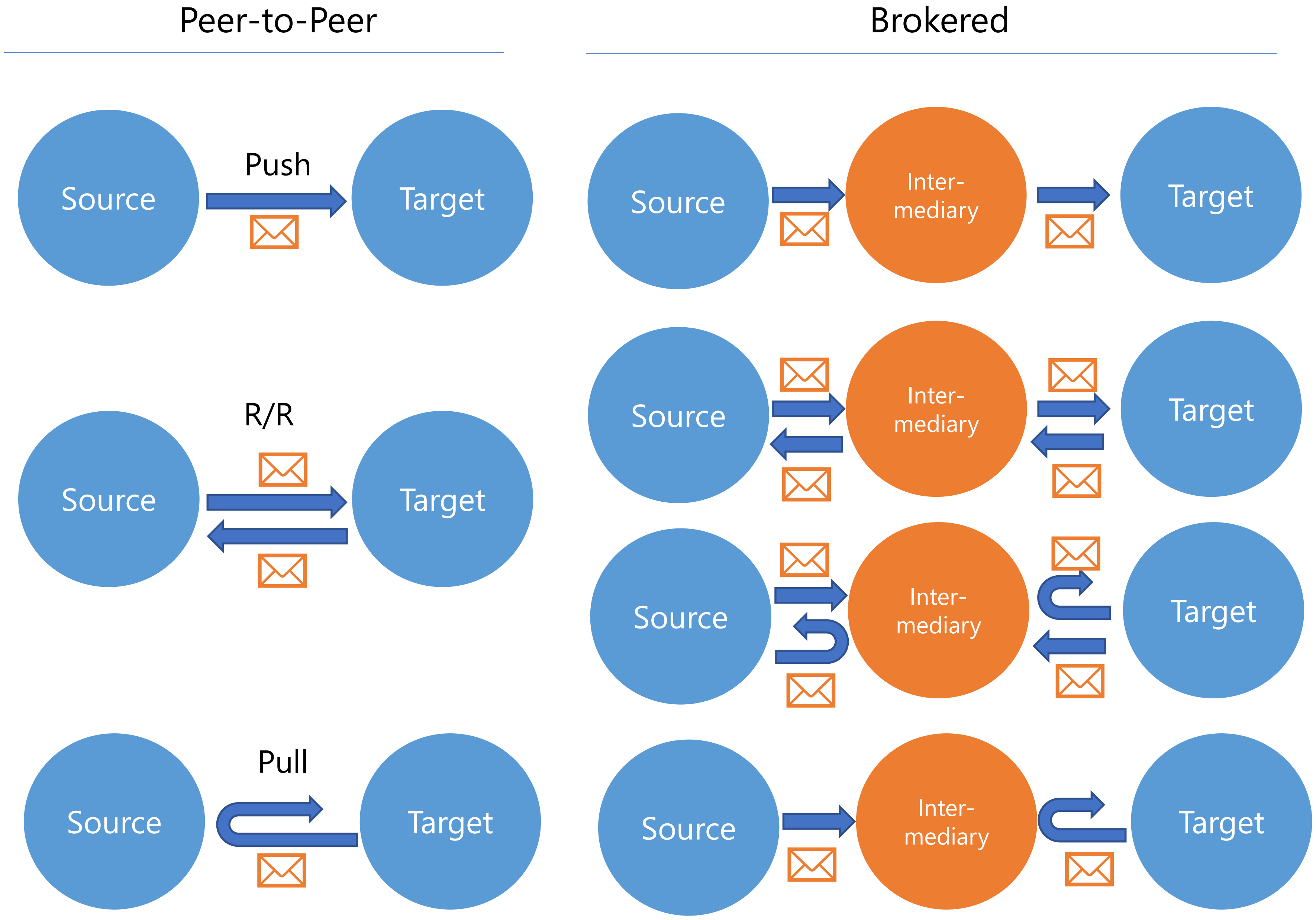
Often: One message, one receiver

Eventing

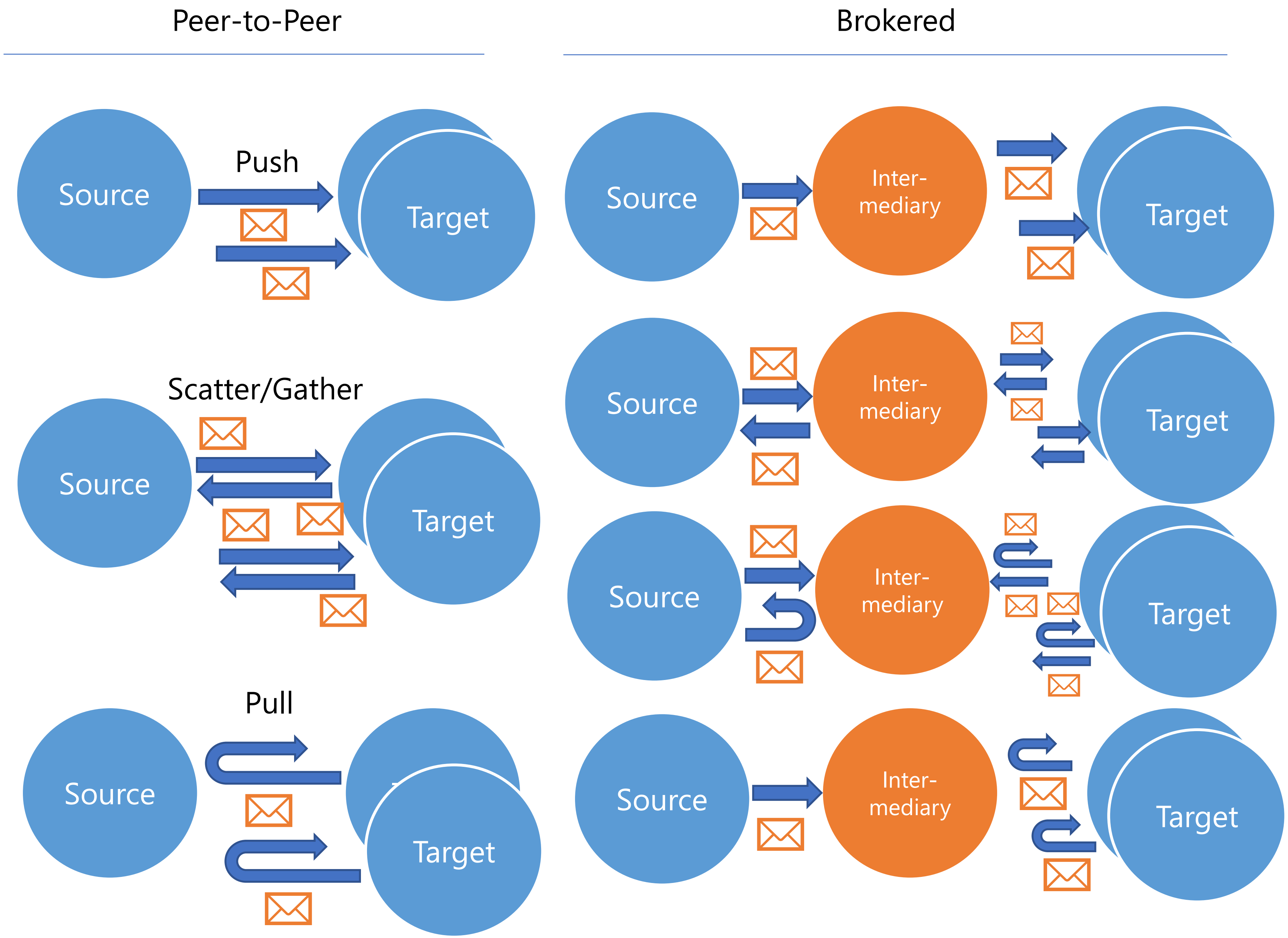


Often: One event, 0 to many receivers

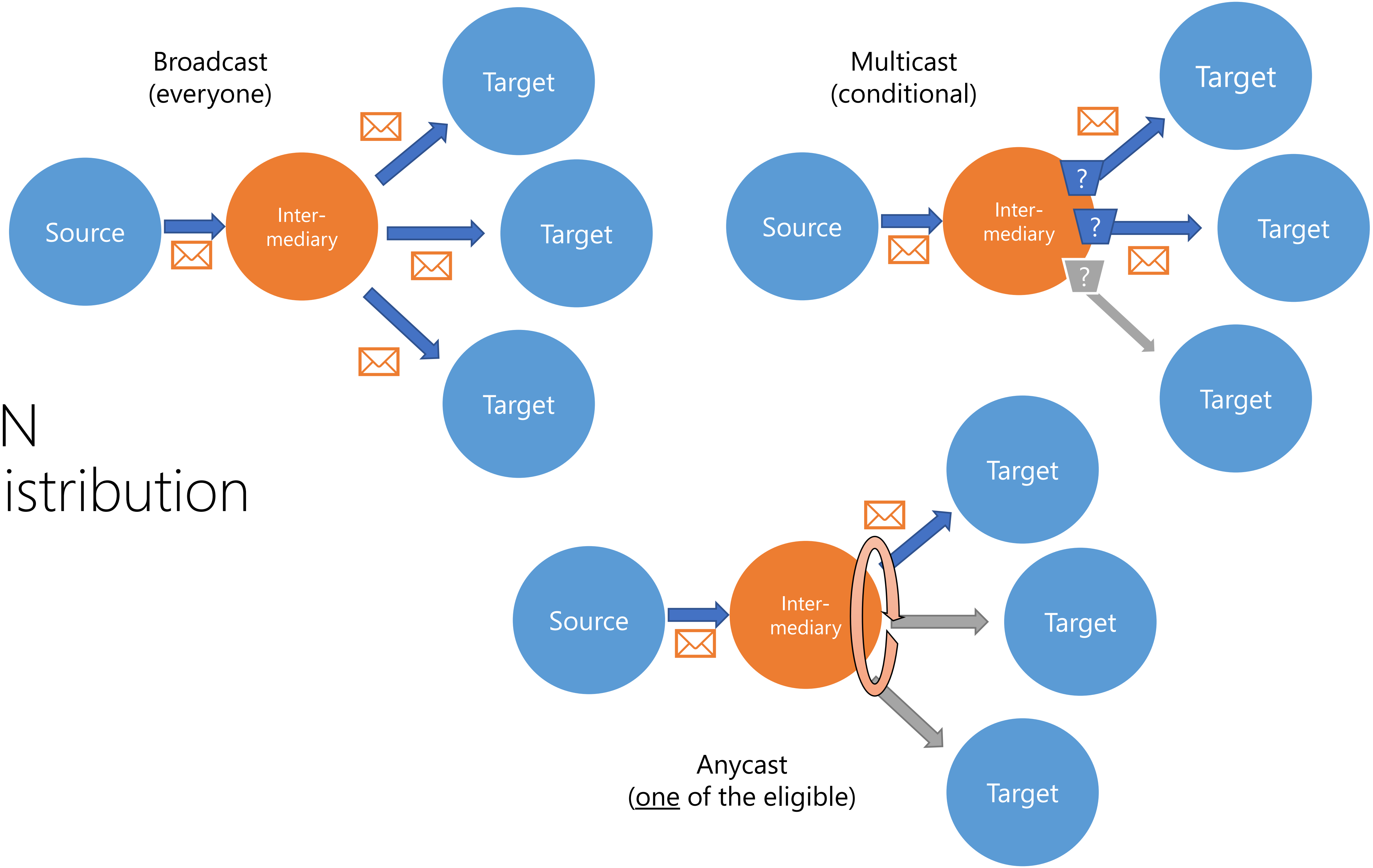
1:1 Patterns


Peer-to-Peer
Brokered
Source
Push
Target
Source
Inter-
mediary
Target
Source
R/R
Target
Source
Inter-
mediary
Target
Source
Inter-
mediary
Target
Source
Pull
Target
Source
Inter-
mediary
Target

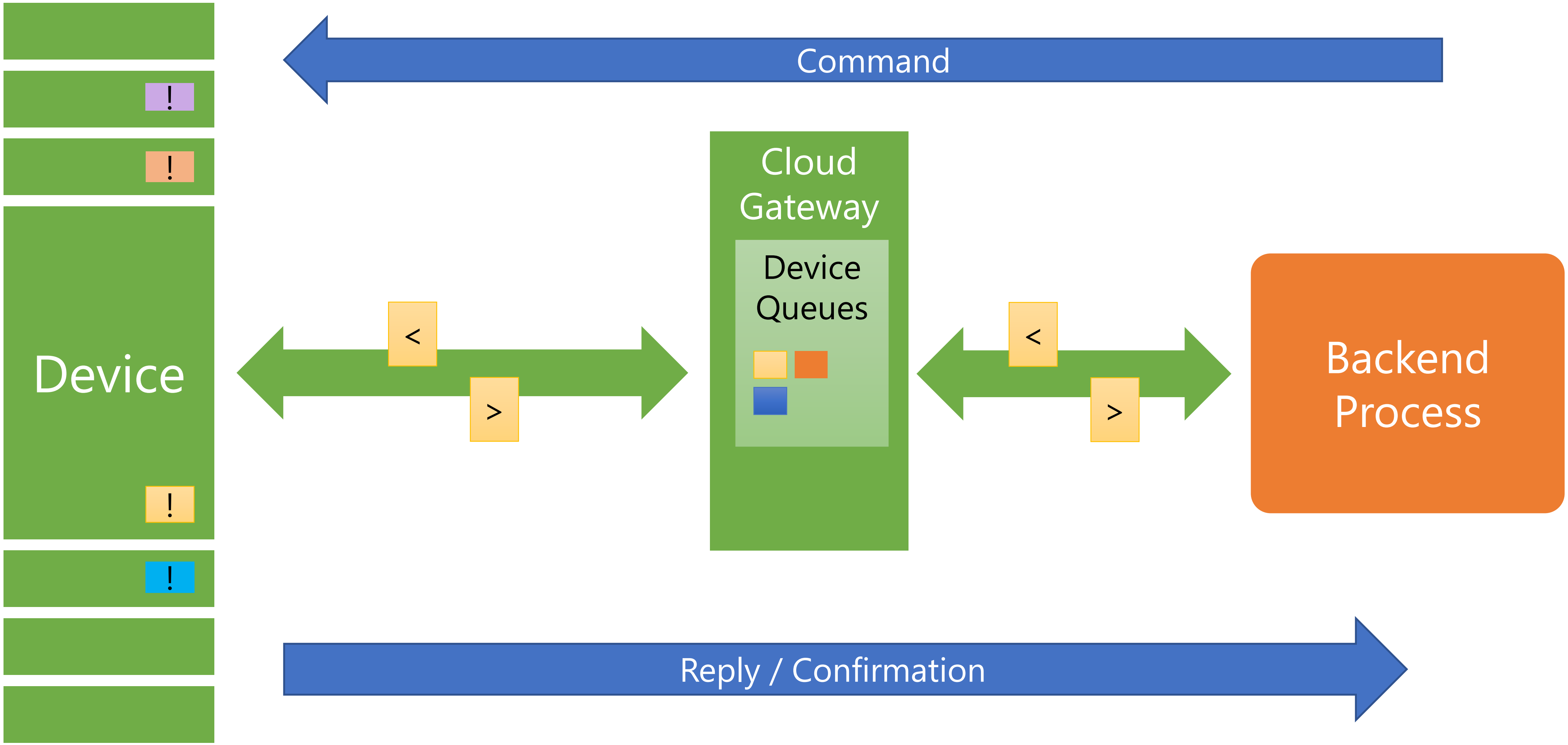
1:N Patterns


Peer-to-Peer
Brokered
Push
Source
Target
Source
Inter-mediary
Target
Scatter/Gather
Source
Target
Source
Inter-mediary
Target
Source
Inter-mediary
Target
Pull
Source
Target
Source
Inter-mediary
Target

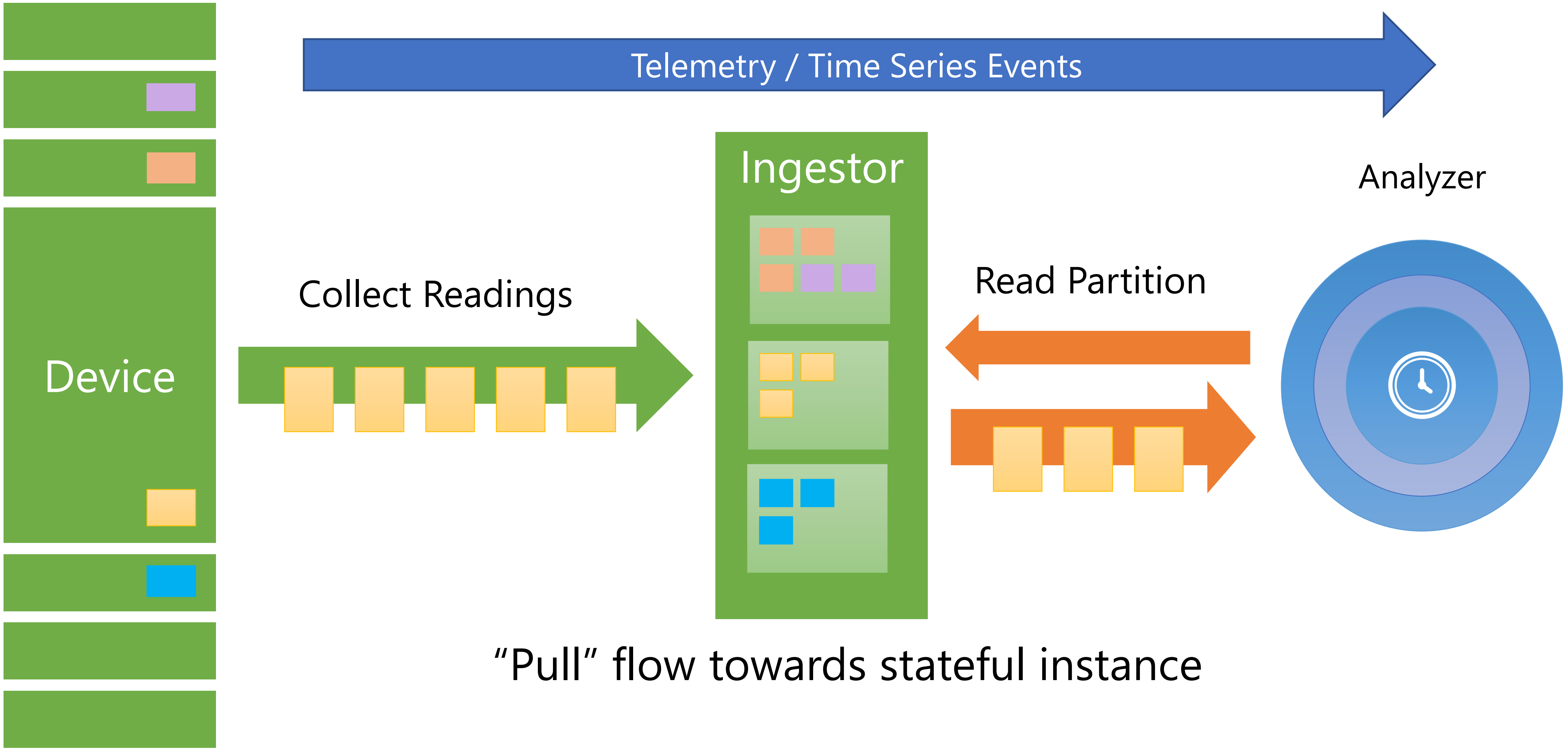
1:N Distribution



Example: Sending Commands

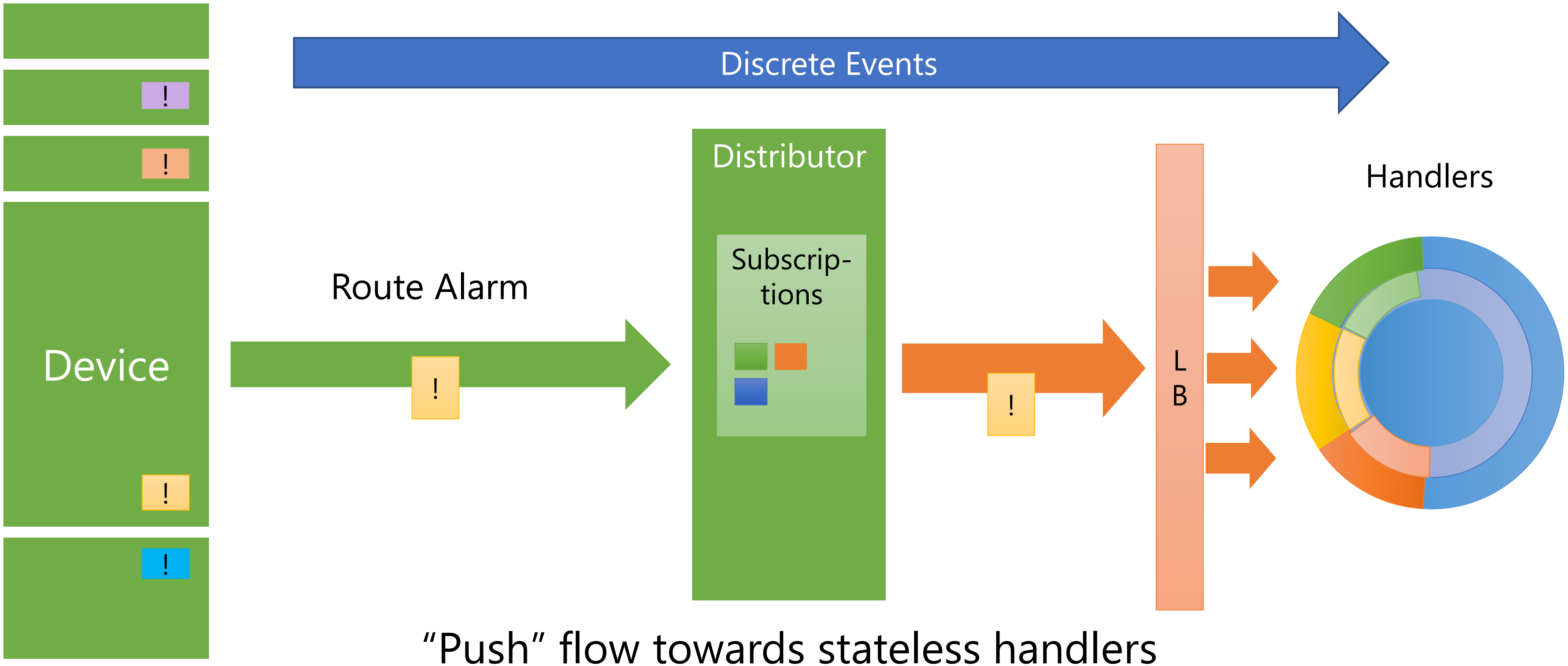


Example: Time Series Processing



"Pull" flow towards stateful instance

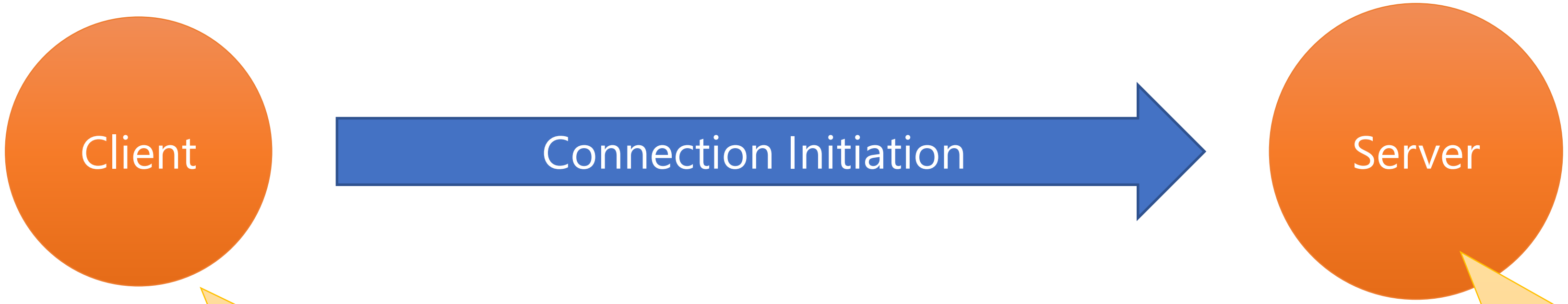
Example: Discrete Event Handling



"Push" flow towards stateless handlers

Azure Event Grid

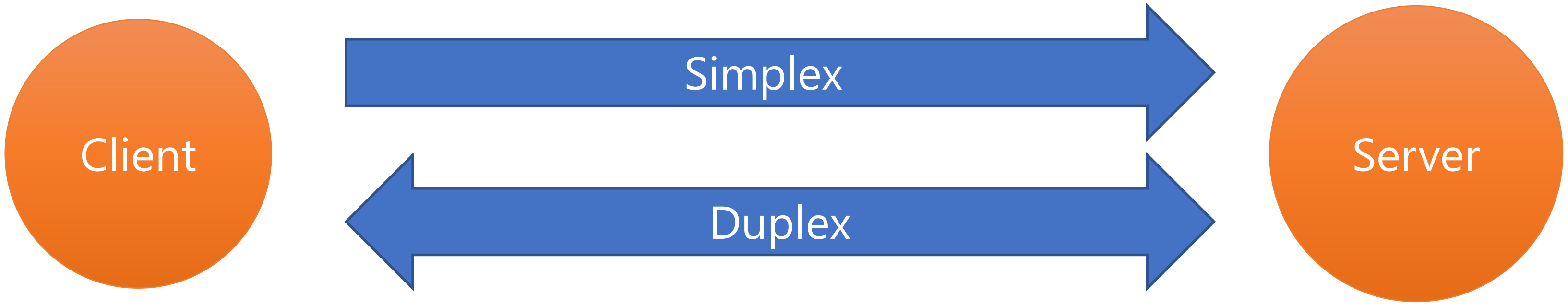
Client vs. Server



- A "client" commonly decides which "server" it wants to talk to and when.
- The client needs to locate the server, choose a protocol the server provides, and initiate a connection.
- The client will then typically provide some form of authentication proof as part of the connection handshake

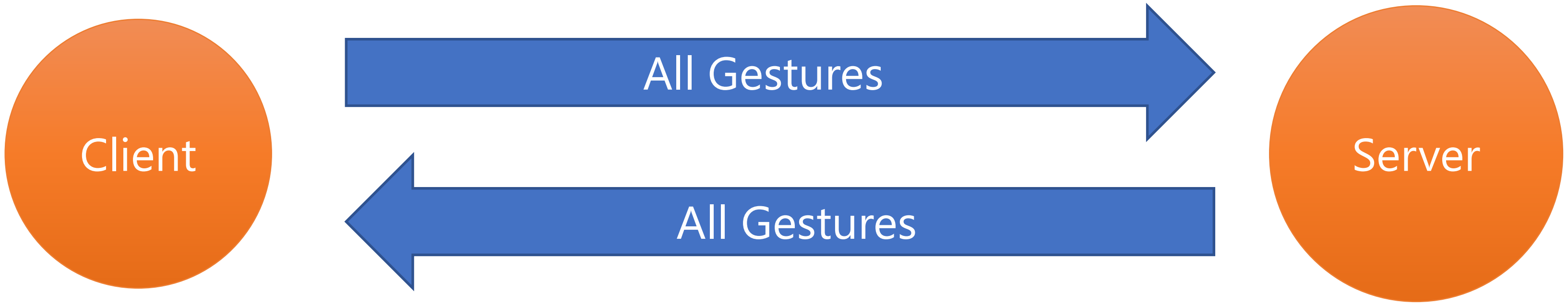
- A "server" commonly listens for client-initiated connections, on one or multiple network protocol endpoints.
- Once a client attempts to connect, the server will typically request some authentication proof that is then validated for access authorization.
- The server needs to deal with any malformed or malicious requests

Directionality



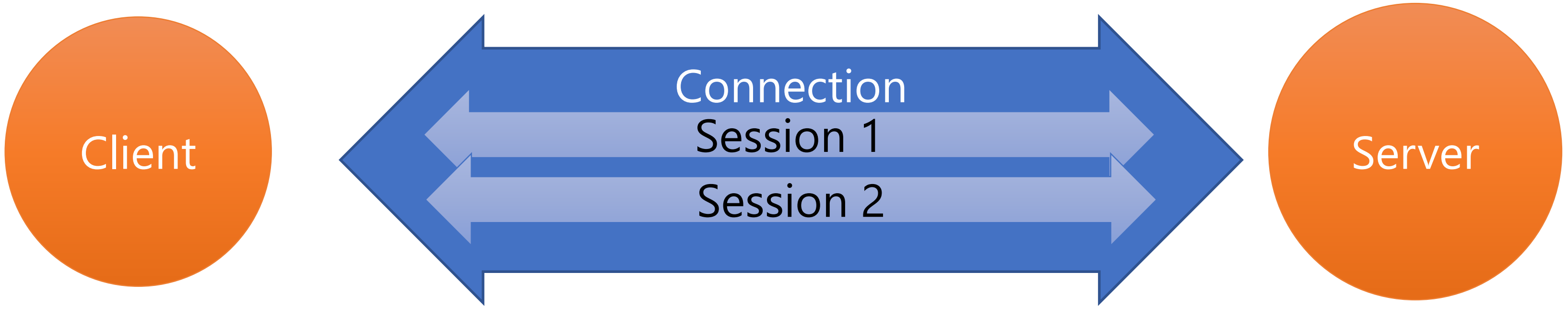
- A **simplex** (or uni-directional) protocol allows flow of data in just one direction.
- A **duplex** (or bi-directional) protocol allows independent flow of data in both directions.
  - Half-duplex only allows one of the parties to communicate at a time
  - Full-duplex allows both parties to communicate concurrently

Symmetry



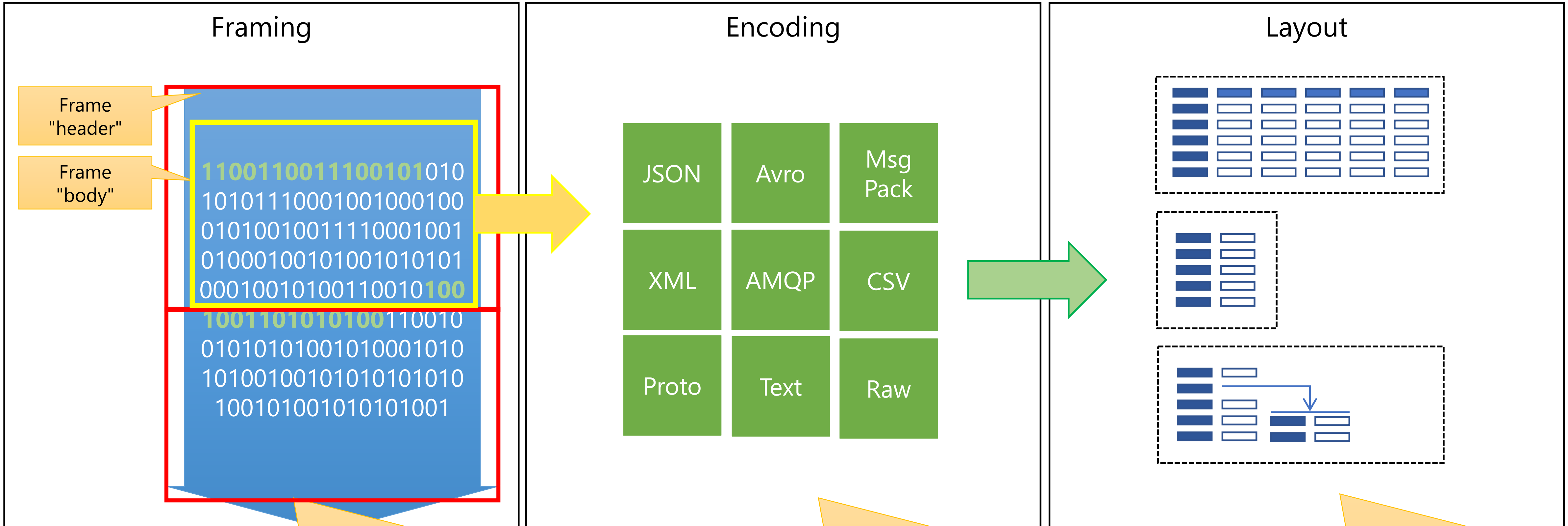
- A protocol is **symmetric** when is allows all of its supported gestures (except for connection establishment) independent of who initiated the connection.

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

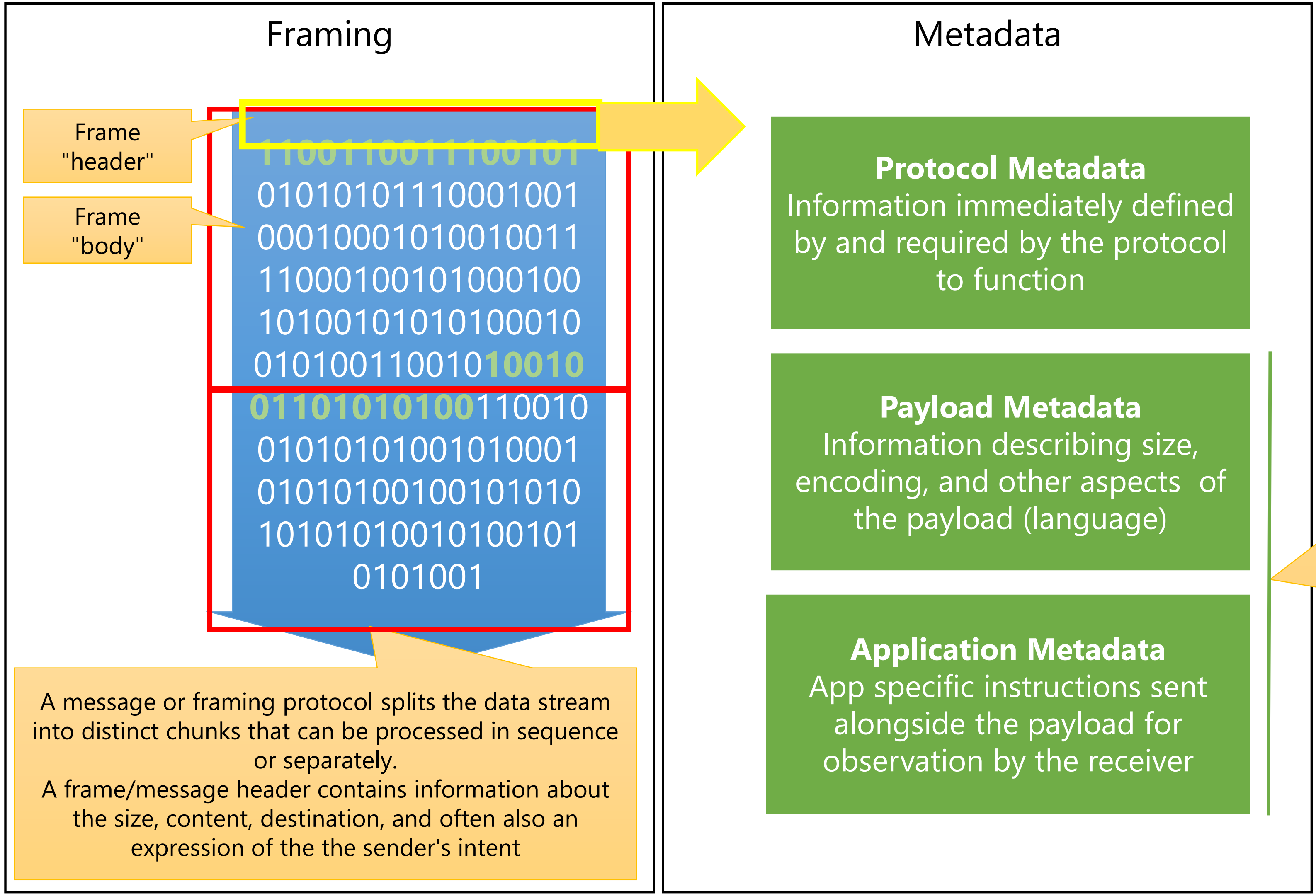


A message or framing protocol splits the data stream into distinct chunks that can be processed in sequence or separately.
A frame/message header contains information about the size, content, destination, and often also an expression of the the sender's intent

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

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



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

Metadata

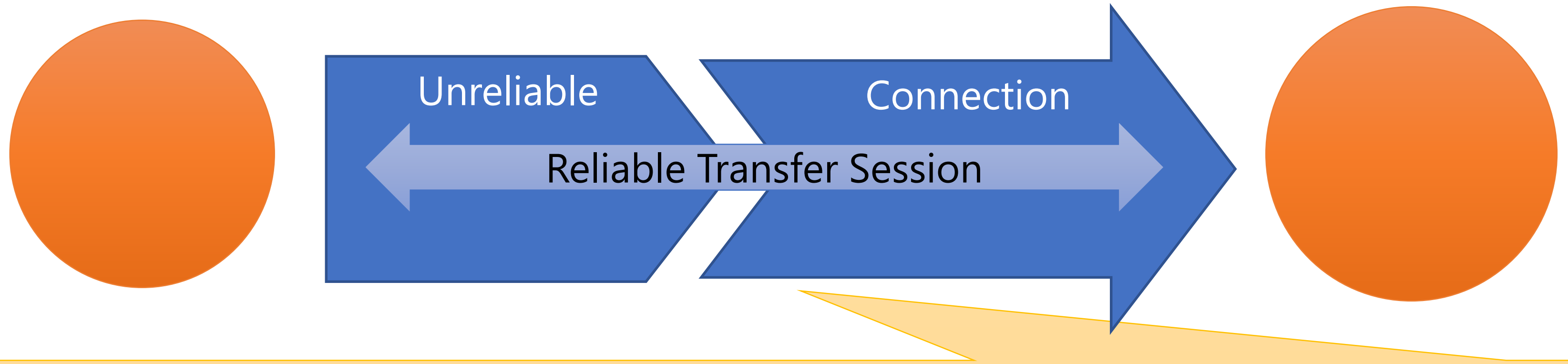
Protocol Metadata
Information immediately defined by and required by the protocol to function

Payload Metadata
Information describing size, encoding, and other aspects of the payload (language)

Application Metadata
App specific instructions sent alongside the payload for observation by the receiver

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

Transfer Assurances



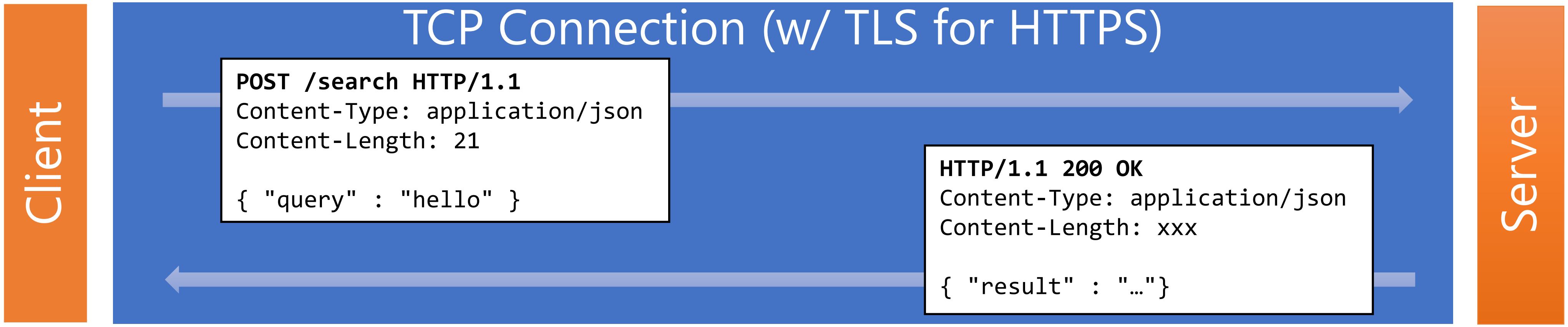
- Reliable protocols allow transfer of frames more reliably than underlying protocol layers
  - Compensating for data loss, preventing duplication, ensuring order
- Various strategies to compensate for data loss
  - Resend on negative acknowledgment („data didn‘t get here“)
  - Resend on absence of acknowledgment
  - 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]

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.



```
POST /search HTTP/1.1
Content-Type: application/json
Content-Length: 21

{ "query" : "hello" }
```

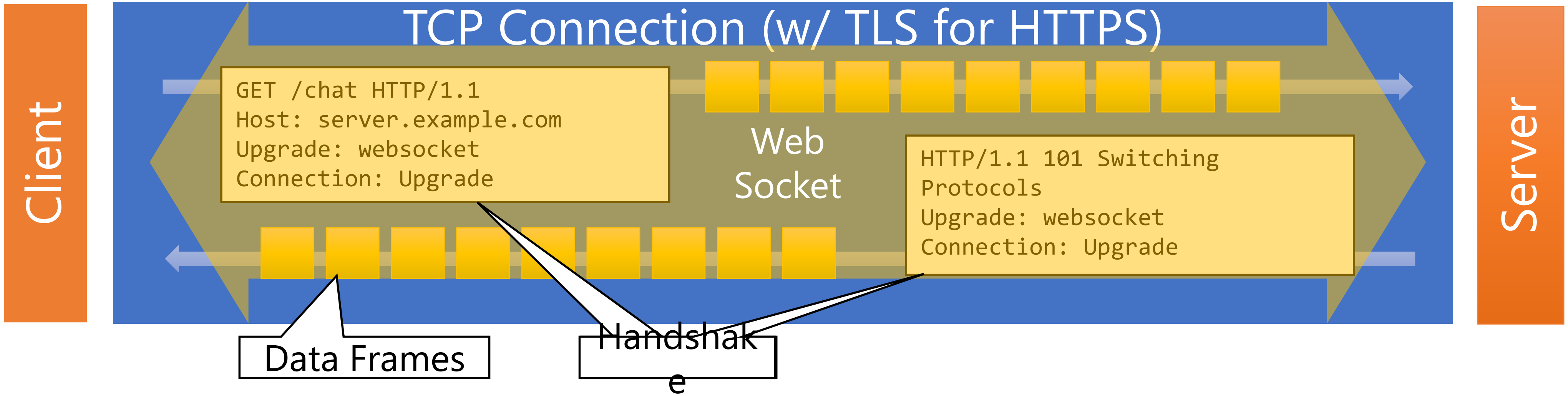
```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: xxx

{ "result" : "…"}
```

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.



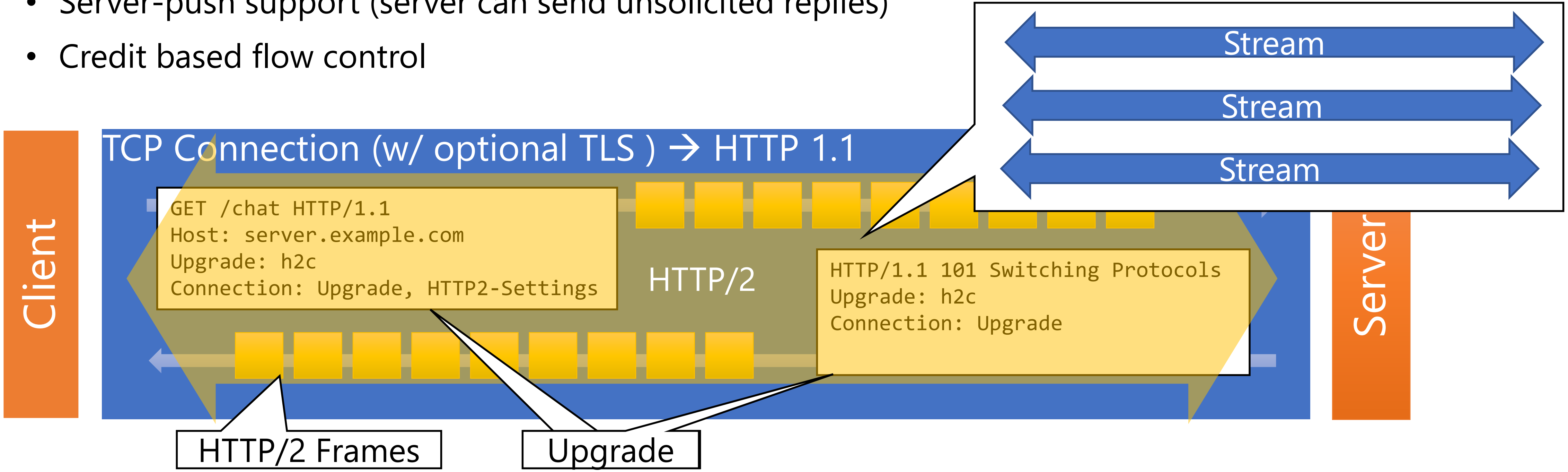
```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
```

```
HTTP/1.1 101 Switching
Protocols
Upgrade: websocket
Connection: Upgrade
```

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



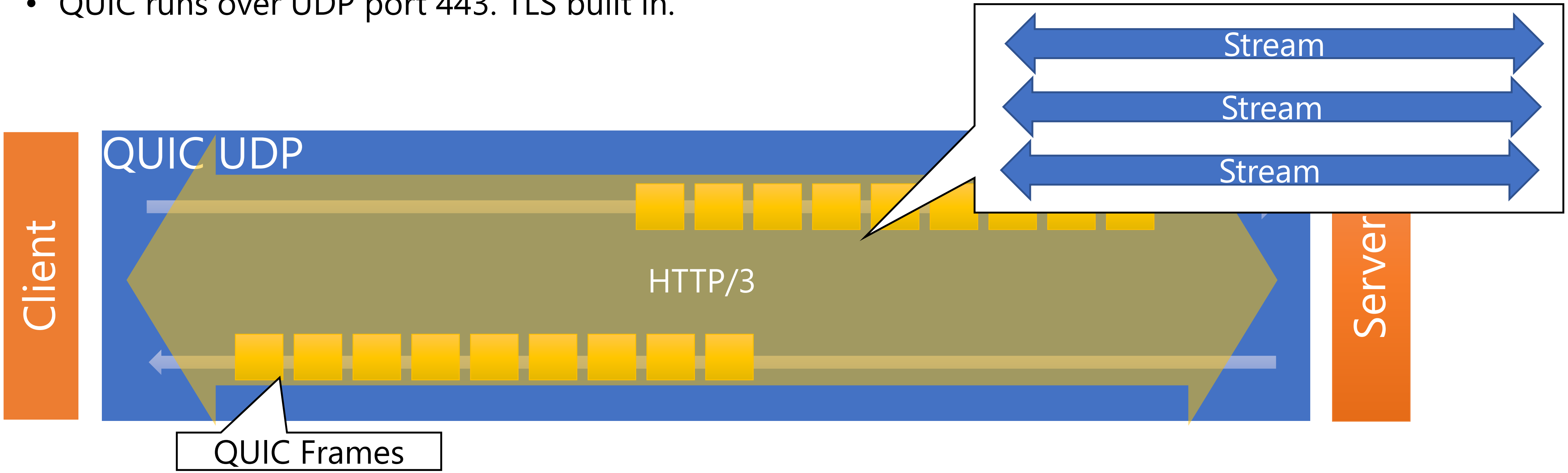
```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: h2c
Connection: Upgrade, HTTP2-Settings
```

```
HTTP/1.1 101 Switching Protocols
Upgrade: h2c
Connection: Upgrade
```

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.

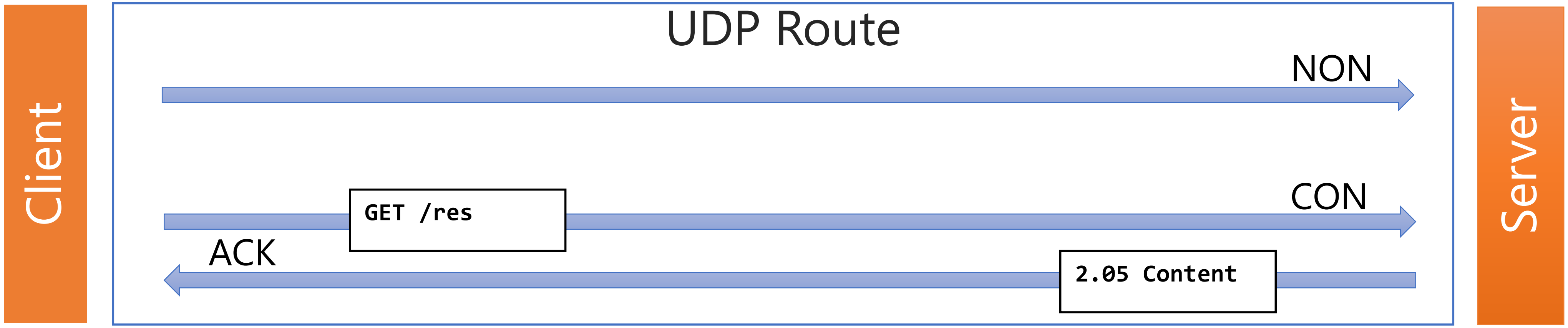


CoAP

Constrained Application Protocol

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

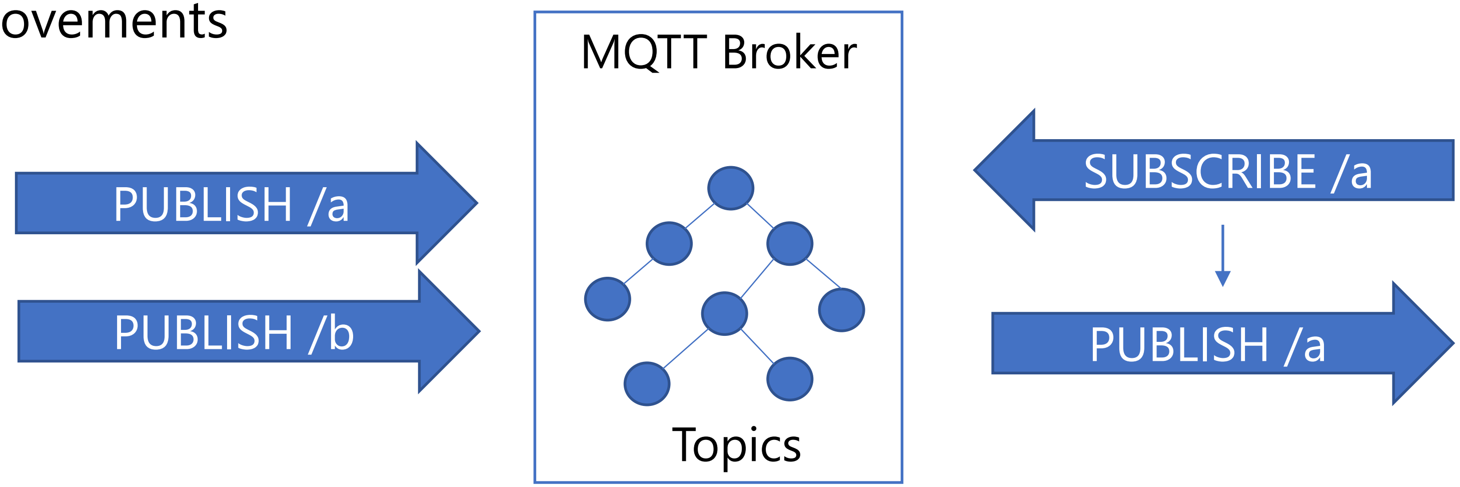
- CoAP is a lightweight **Application Protocol**
  - Adapts principles of HTTP to very constrained devices
  - CoAP is originally based on UDP, definition of CoAP for TCP exists
  - Supports multicast on UDP
  - Creates a simple reliability layer over UDP using ACKs



OASIS MQTT

Patterns	Oneway
Symmetric	No
Multiplexing	No
Encodings	Variable (5.0)
Metadata	Yes (5.0)
Assurances	AMO, ALO, EO

- 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

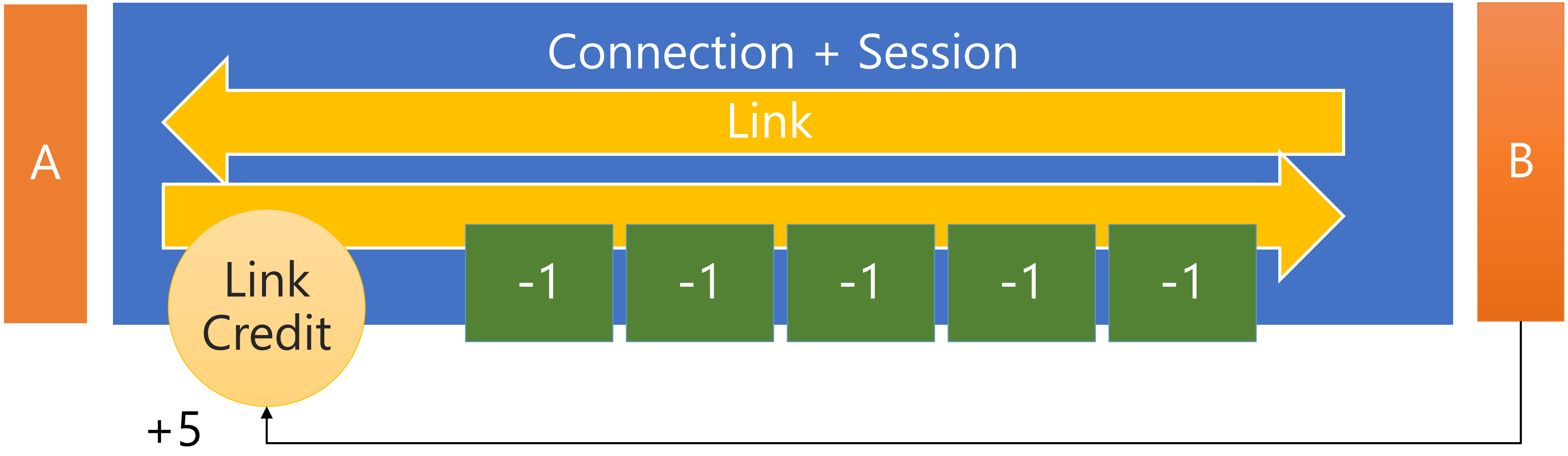


OASIS AMQP

Advanced Message Queuing Protocol

Patterns	Any
Symmetric	Yes
Multiplexing	Yes
Encodings	Variable
Metadata	Yes
Assurances	AMO, ALO, EO

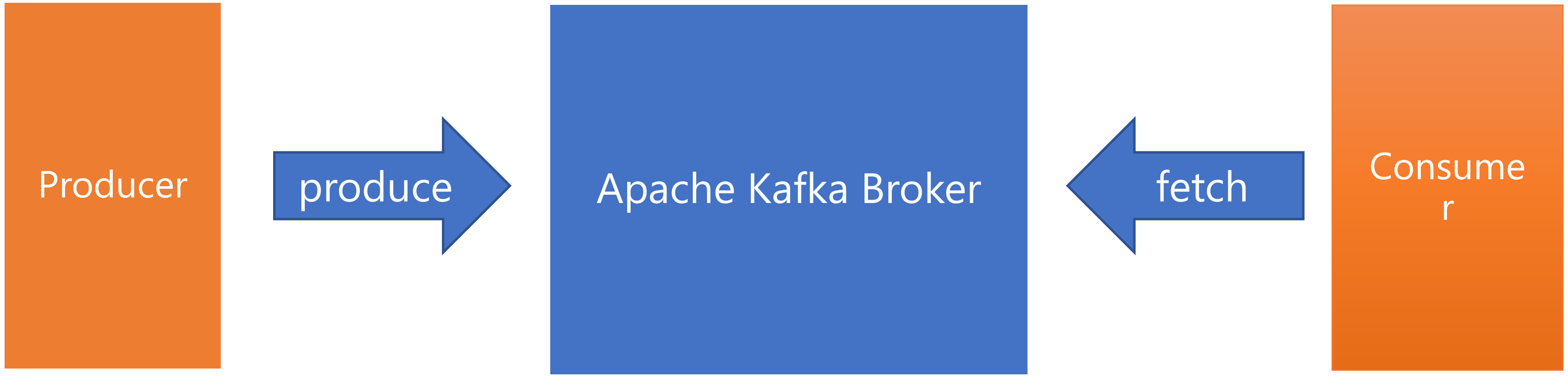
- AMQP 1.0 is a symmetric, reliable **Message Transfer Protocol** with support for multiplexing and flow control
  - Supports queuing, pub/sub, filters, one-way, request-response, streams
  - No topology assumptions, multi-hop routing facilities
- AMQP 0.9 is an expired draft with a fixed topology model (



Apache Kafka

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

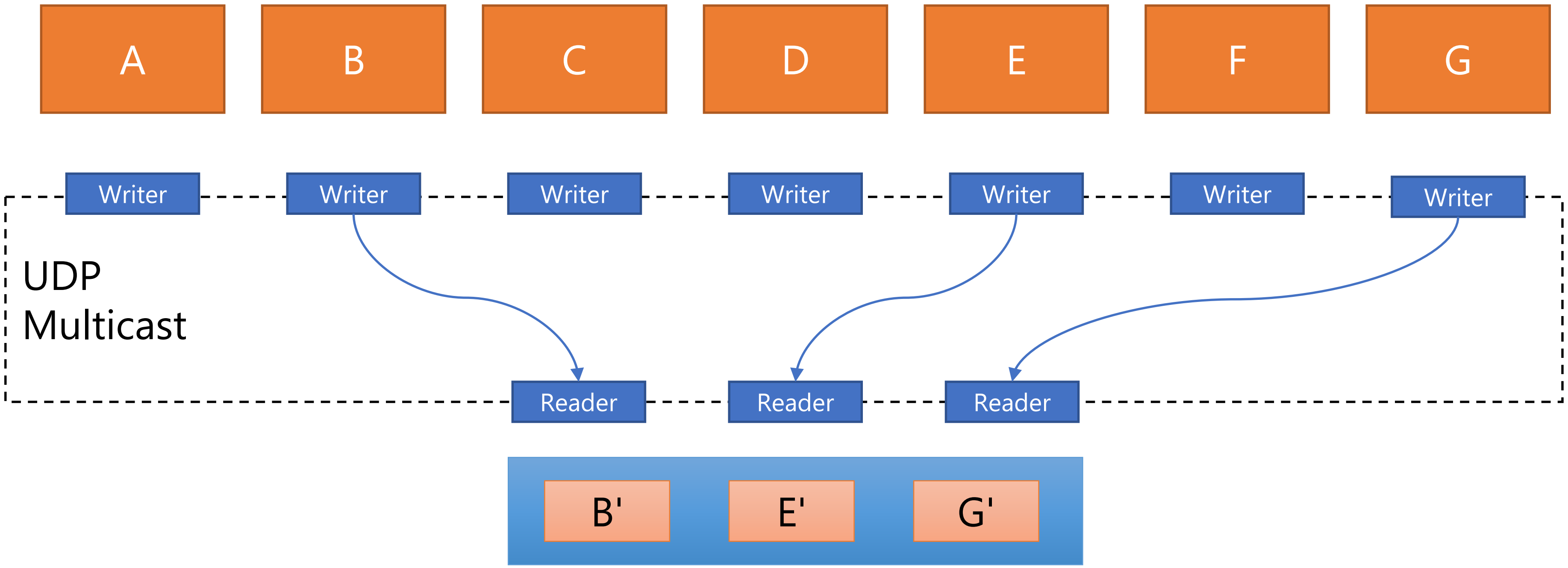
- The Apache Kafka protocol is a project-specific request/response protocol for the Apache Kafka broker
  - SASL preamble for authentication
  - API keys identify operations, specific message types per key
  - Effectively an RPC protocol tailored to Kafka’s features



OMG DDS

Data Distribution System

- OMG (Object Management Group) Standard describing a distributed, multi-master cache replication infrastructure
- Built on top of RTPS which builds on UDP multicast



OPC/TCP

Patterns	Any
Symmetric	Yes
Multiplexing	No
Encodings	Variable
Metadata	Yes
Assurances	AMO, ALO, EO

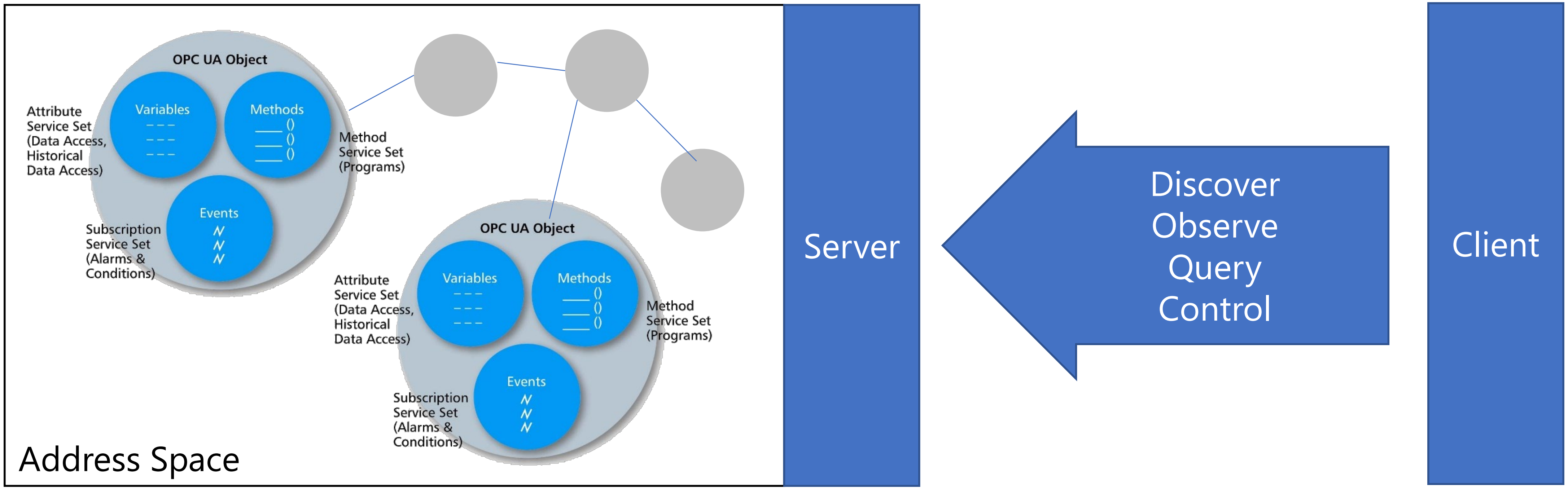
- OPC/TCP is a symmetric **message transfer protocol**
  - Compact data encoding w/ external schema
  - Created specifically for OPC UA (more later)
  - Inline, non-TLS security model w/ optional message-level encryption

Unifying Abstractions

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

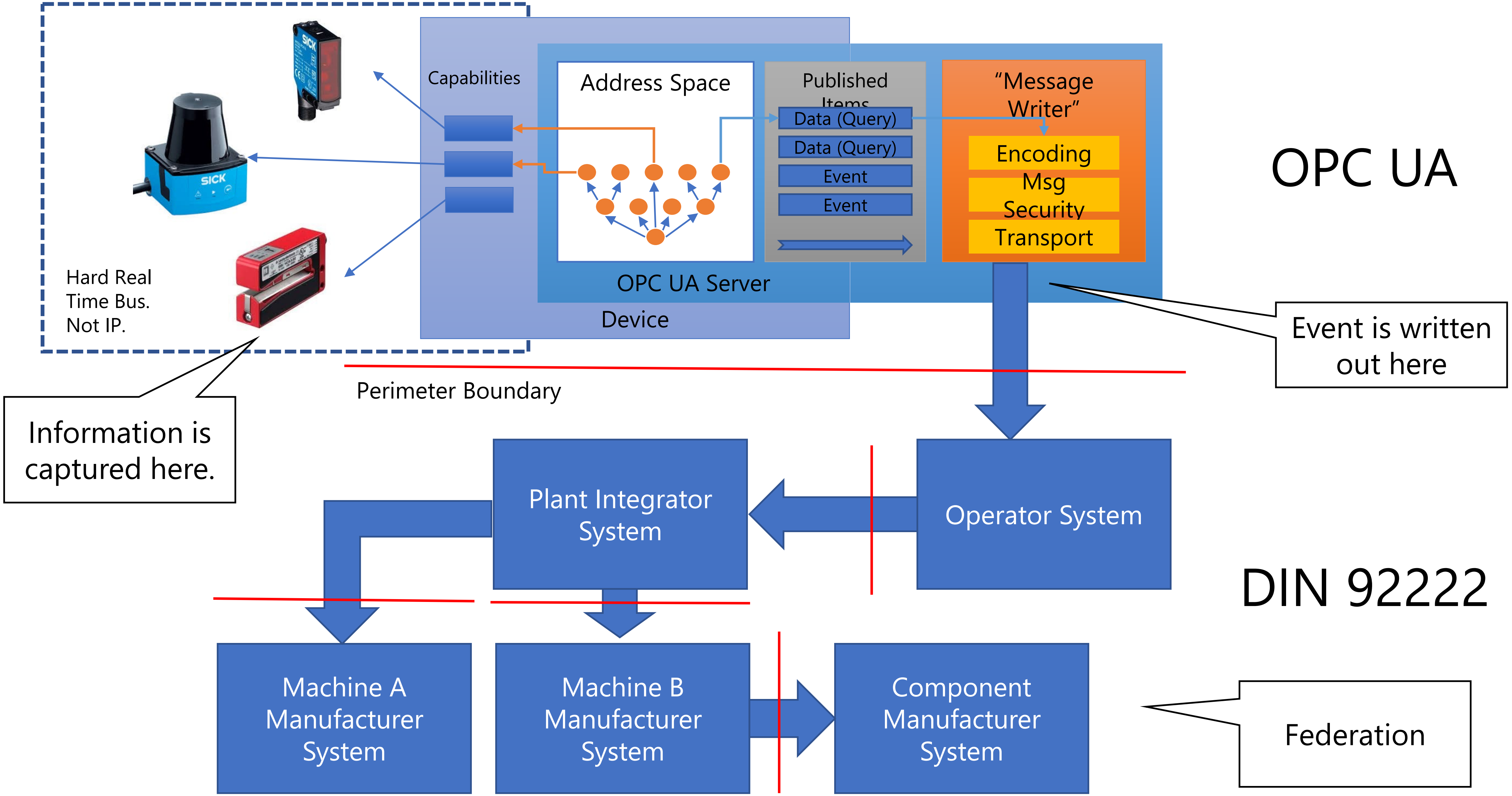

OPC UA Object
Variables
Methods
Events
Attribute Service Set (Data Access, Historical Data Access)
Method Service Set (Programs)
Subscription Service Set (Alarms & Conditions)
OPC UA Object
Variables
Methods
Events
Attribute Service Set (Data Access, Historical Data Access)
Method Service Set (Programs)
Subscription Service Set (Alarms & Conditions)

Address Space

Server

Discover
Observe
Query
Control

Client



Capabilities

Address Space

Published Items

Data (Query)

Data (Query)

Event

Event

“Message Writer”

Encoding

Msg

Security

Transport

OPC UA Server

Device

Hard Real Time Bus. Not IP.

Information is captured here.

Event is written out here

Perimeter Boundary

Plant Integrator System

Operator System

Machine A Manufacturer System

Machine B Manufacturer System

Component Manufacturer System

Federation

OPC UA

DIN 92222

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
- Encoding options
  - JSON (required for all implementations)
  - Extensible for binary encodings: Avro, AMQP, etc.

cloudevents

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**
  - When was it sent? **time**
  - What context was it sent out of? **source**
  - 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 binary

CloudEvents – 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.
- AMQP type system encoding defined for metadata mapping to AMQP properties and annotations
- Further compact binary event format candidates might be CBOR, or Protobuf.

```
{
  "specversion" : "0.1",
  "type" : "myevent",
  "source" : "uri:example-com:mydevice",
  "id" : "A234-1234-1234",
  "time" : "2018-04-05T17:31:00Z",
  "datacontenttype" : "text/plain",
  "data" : "Hello"
}
```

JSON Representation

CloudEvents – 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.

Service Bus & Azure Queues

Cloud messaging

AMQP

HTTP

Cloud-Events

Event Hubs

Telemetry stream ingestion

AMQP

HTTP

Kafka

Cloud-Events

Event Grid

Event distribution

AMQP

HTTP

Cloud-Events

IoT Hub

IoT messaging and management

AMQP

MQTT

OPCUA

Cloud-Events

Relay

Discovery, Firewall/NAT Traversal

HTTP

Web-Sockets

Cloud-Events

Microsoft Azure

Microsoft