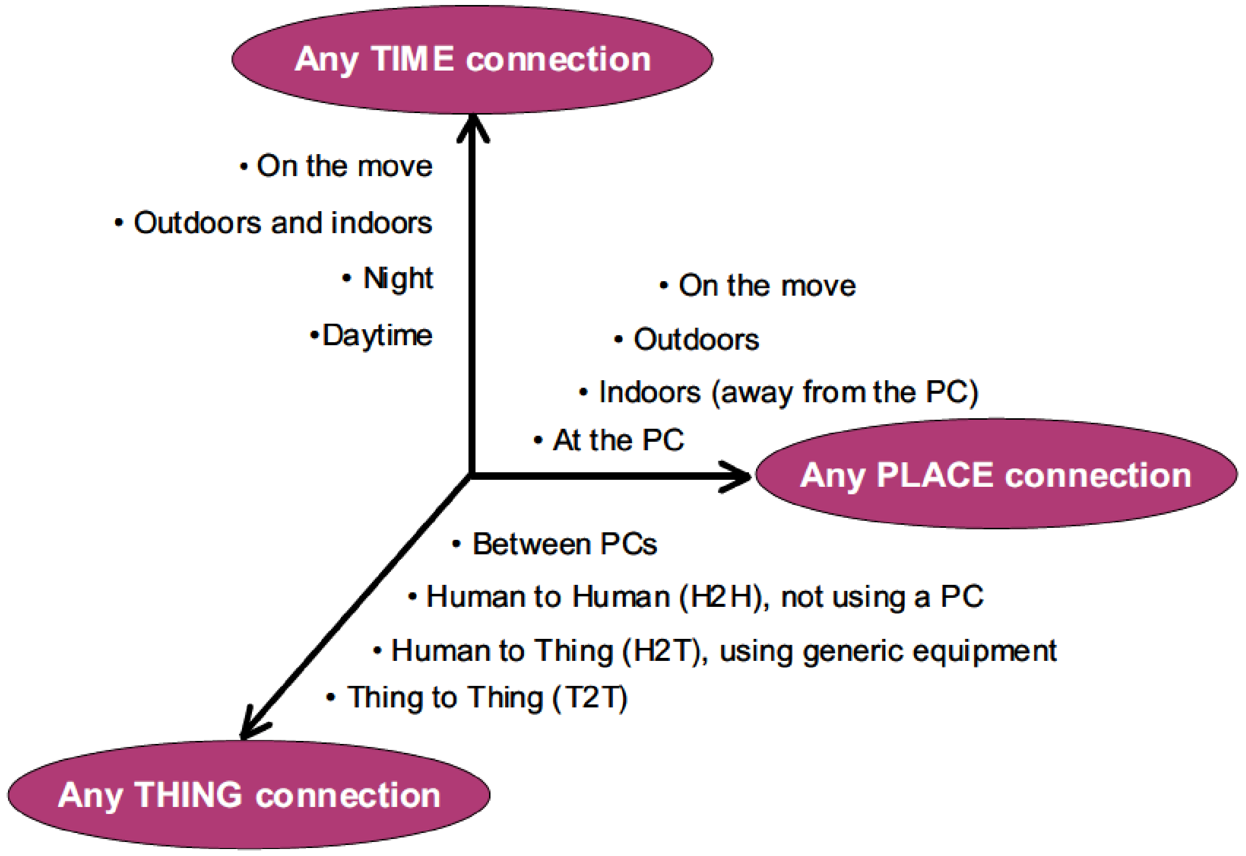
Logical Design of IoT

IoT – New Dimesion



Source: ITU adapted from Nomura Research Institute

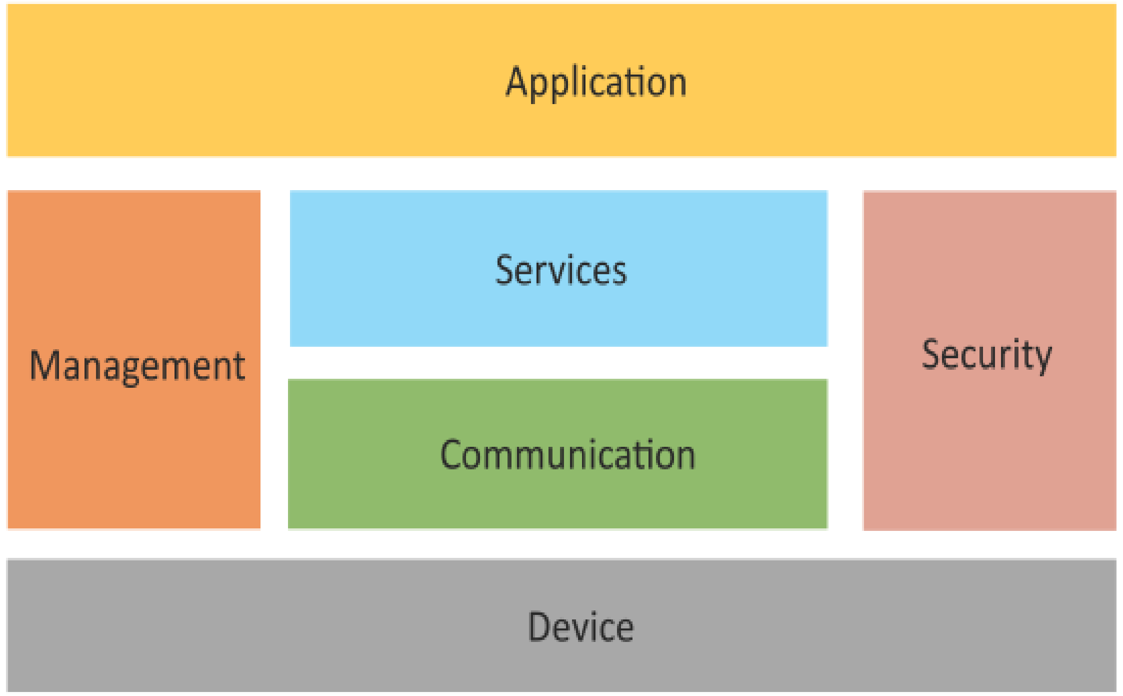
Logical Design of IoT

- Logical design of an IoT system refers to an abstract representation of the entities and processes without going into the low-level specifics of the implementation.
- An IoT system comprises a number of functional blocks that provide the system the capabilities for identification, sensing, actuation, communication and management.
- The process of logical design involves arranging data into a series of logical relationships called entities and attributes.
  - An *entity* represents a chunk of information. In relational databases, an entity often maps to a table.
  - An *attribute* is a component of an entity and helps define the uniqueness of the entity. In relational databases, an attribute maps to a column.

IoT Functional Blocks
IoT Communication Models
IoT Communication APIs

IoT Functional Blocks

- Device : Devices such as sensing, actuation, monitoring and control functions.
- Communication : Handles communication systems with IoT Protocols
- Services: like device monitoring, device control services, data publishing services and device discovery
- Management : Functions to govern the system
- Security : Functions as authentication, authorization, message and content integrity, and data security
- Applications: Interface that user can use to control & monitor the IoT system

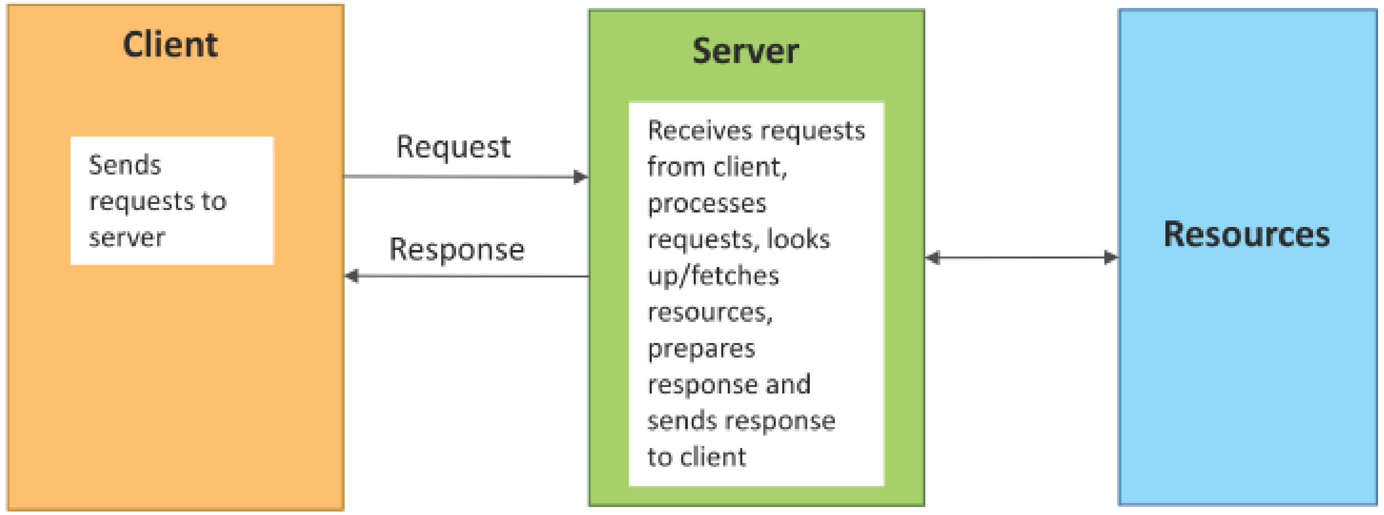


IoT Communication Models

- Request – Response
- Publish – Subscribe
- Push-Pull
- Exclusive Pair

Request–Response Communication Model

- Request–Response is a communication model in which the client sends requests to the server and the server responds to the requests.
- When the server receives a request, it decides how to respond, fetches the data, retrieves resource representations, prepares the response and then sends the response to the client.
- Stateless communication model- receiver not to retain the state of previous request-response session, every request is understood in isolation

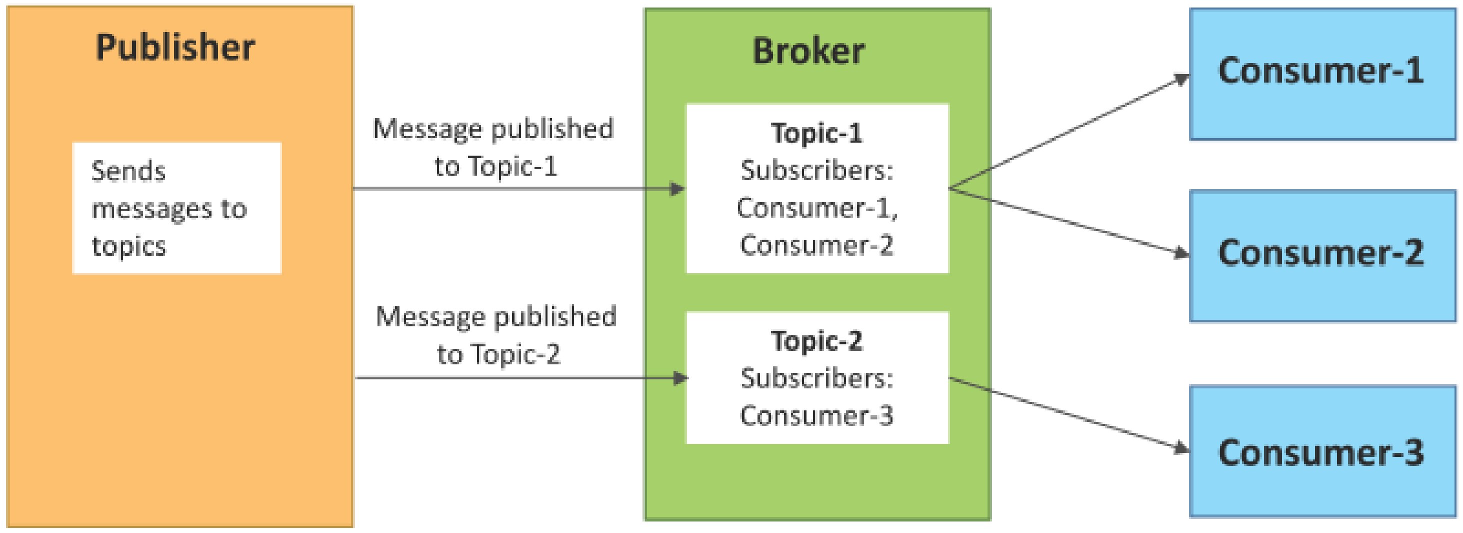

Client
Sends requests to server
Request
Response
Server
Receives requests from client, processes requests, looks up/fetches resources, prepares response and sends response to client
Resources

Request–Response Communication Model

- Spread sheet (client) sent to a network-printer (server)
- Watching a You Tube video on a Smart Phone (YouTube browser/App-client and Youtube Server)
- In automation, PC is the client and the PLC is the server. The HMI application on your PC requests data from the PLC in order to display it on the monitor.
-

Publish–Subscribe Communication Model

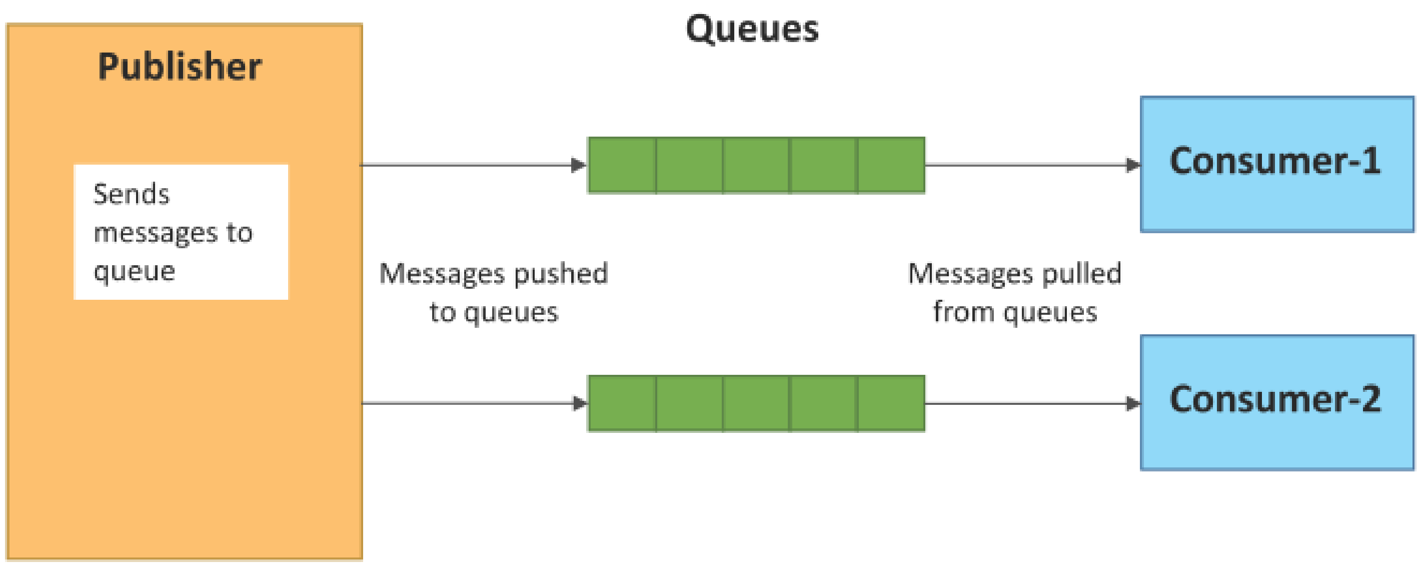
- Publish–Subscribe is a communication model that involves publishers, brokers and consumers.
- Publishers are the source of data. Publishers send the data to the topics which are managed by the broker. Publishers are not aware of the consumers.
- Consumers subscribe to the topics which are managed by the broker.
- When the broker receives data for a topic from the publisher, it sends the data to all the subscribed consumers.



- Broker does not store the data (no unloading), sends it to subscibers (only routing)

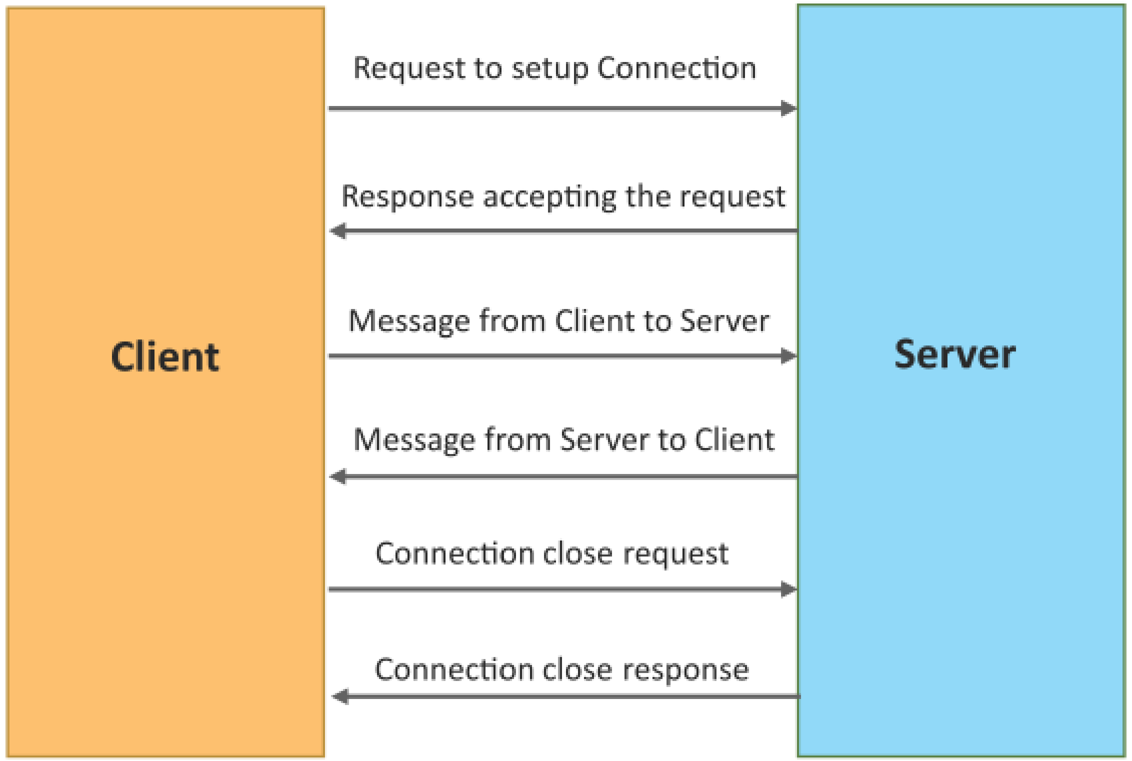
Push–Pull Communication Model

- Push–Pull is a communication model in which the data producers push the data to queues and the consumers pull the data from the queues. Producers do not need to be aware of the consumers.
- Queues help in decoupling the messaging between the producers and consumers.
- Queues also act as a buffer which helps in situations when there is a mismatch between the rate at which the producers push data and the rate at which the consumers pull data.



Exclusive Pair Communication Model

- Exclusive Pair is a bidirectional, fully duplex communication model that uses a persistent connection between the client and the server.
- Once the connection is set up it, remains open until the client sends a request to close the connection.
- Client and server can send messages to each other after connection setup.

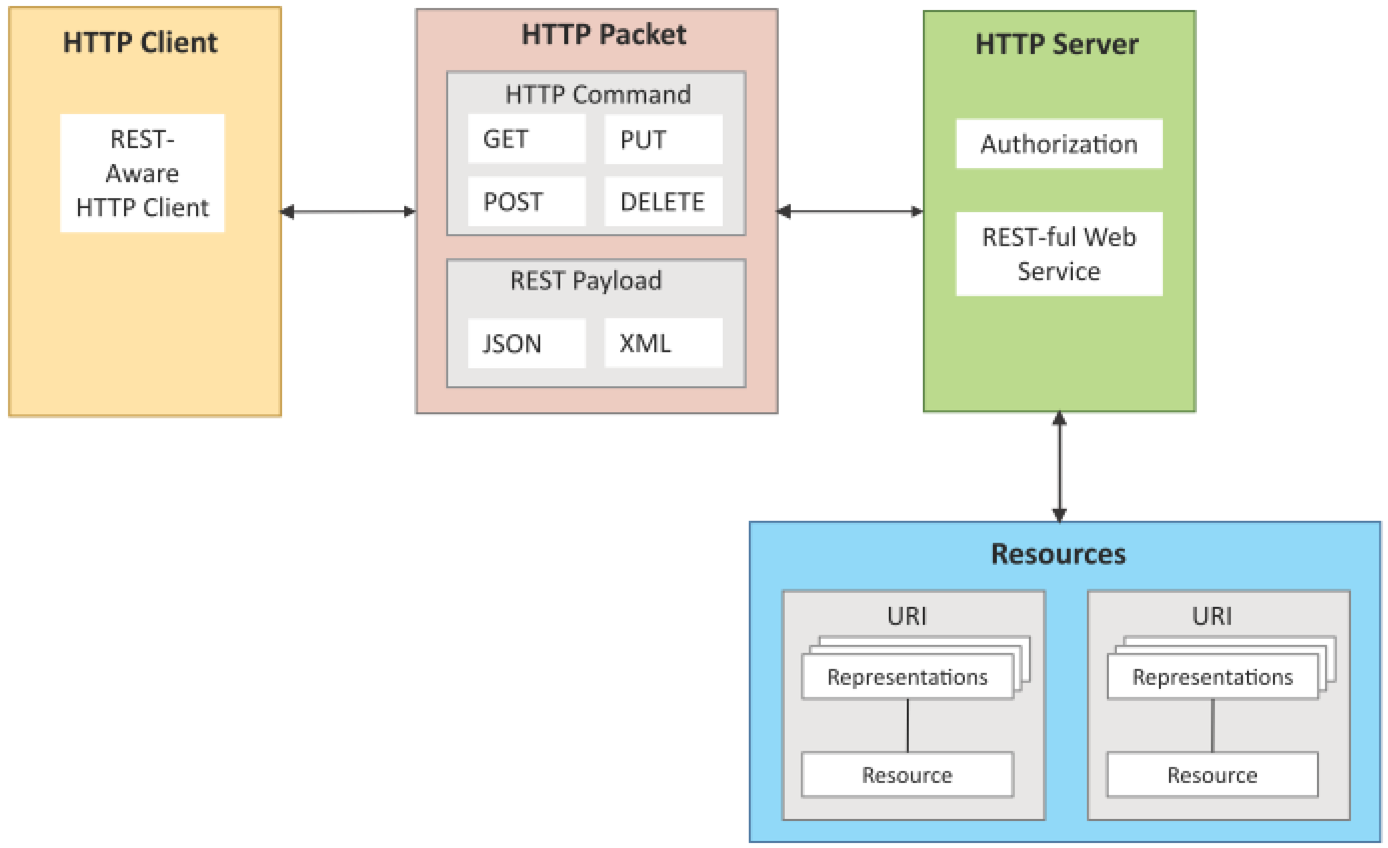

Client
Server
Request to setup Connection
Response accepting the request
Message from Client to Server
Message from Server to Client
Connection close request
Connection close response

IoT Communication APIs

- REST – based Communication APIs
- WebSocket – based Communication APIs

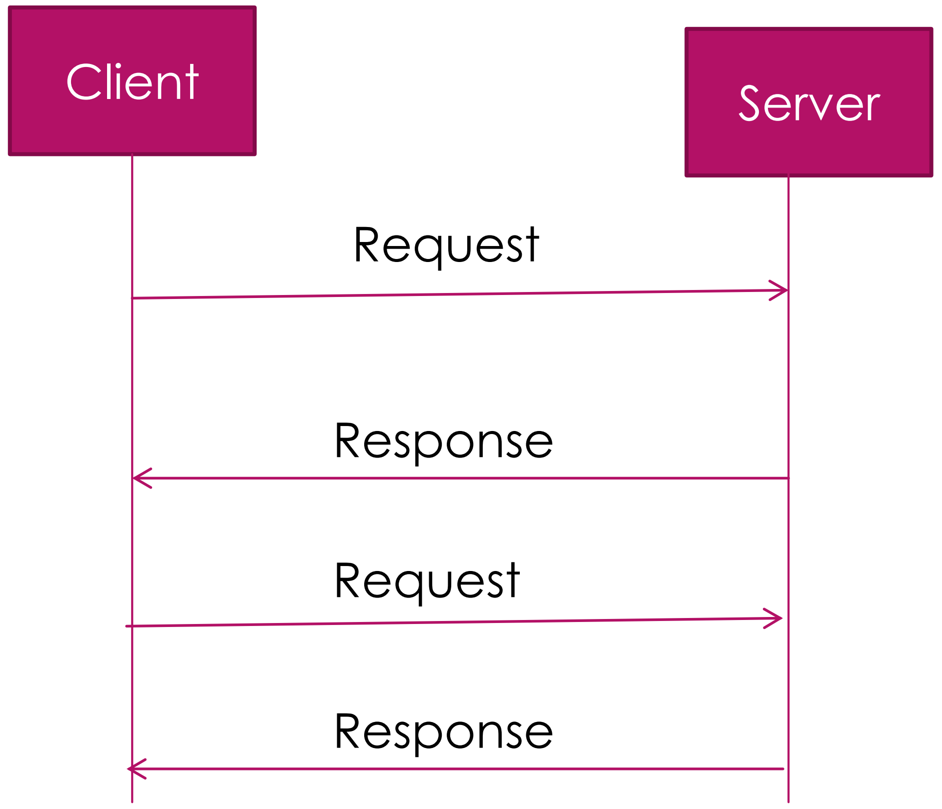
REST-based Communication APIs

- Representational State Transfer (REST) is a set of architectural principles by which you can **design web services and web APIs** that focus on a system's resources and how resource states are addressed and transferred.
- REST APIs **follow the request–response communication model.**
- REST architectural constraints apply to the components, connectors and data elements within a distributed hypermedia system.



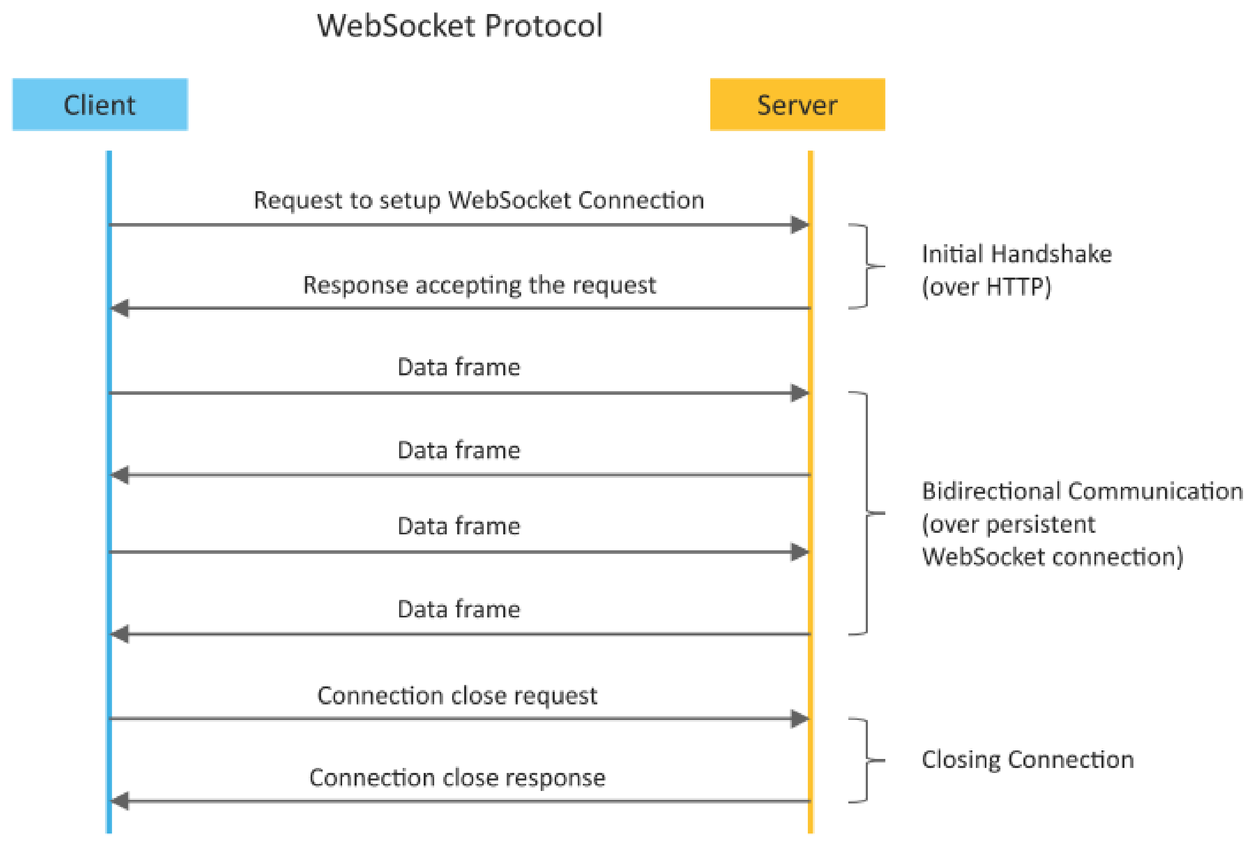
REST-based Communication APIs Constraints

- Client – Server
- Stateless
- Cacheable
- Layered System
- Uniform Interface
- Code on demand



WebSocket-based Communication APIs

- WebSocket APIs **allow bi-directional, full duplex communication** between clients and servers.
- WebSocket APIs follow the **exclusive pair communication model**.


WebSocket Protocol
Client
Server
Request to setup WebSocket Connection
Response accepting the request
Initial Handshake (over HTTP)
Data frame
Data frame
Data frame
Data frame
Bidirectional Communication (over persistent WebSocket connection)
Connection close request
Connection close response
Closing Connection

Difference between REST and WebSocket-based Communication APIs

Comparison Based on	REST	Websocket
State	Stateless	Stateful
Directional	Unidirectional	Bidirectional
Req-Res/Full Duplex	Follow Request Response Model	Exclusive Pair Model
TCP Connections	Each HTTP request involves setting up a new TCP Connection	Involves a single TCP Connection for all requests
Header Overhead	Each request carries HTTP Headers, hence not suitable for real-time	Does not involve overhead of headers.
Scalability	Both horizontal and vertical are easier	Only Vertical is easier

Thank you!