Tortoise Evolved

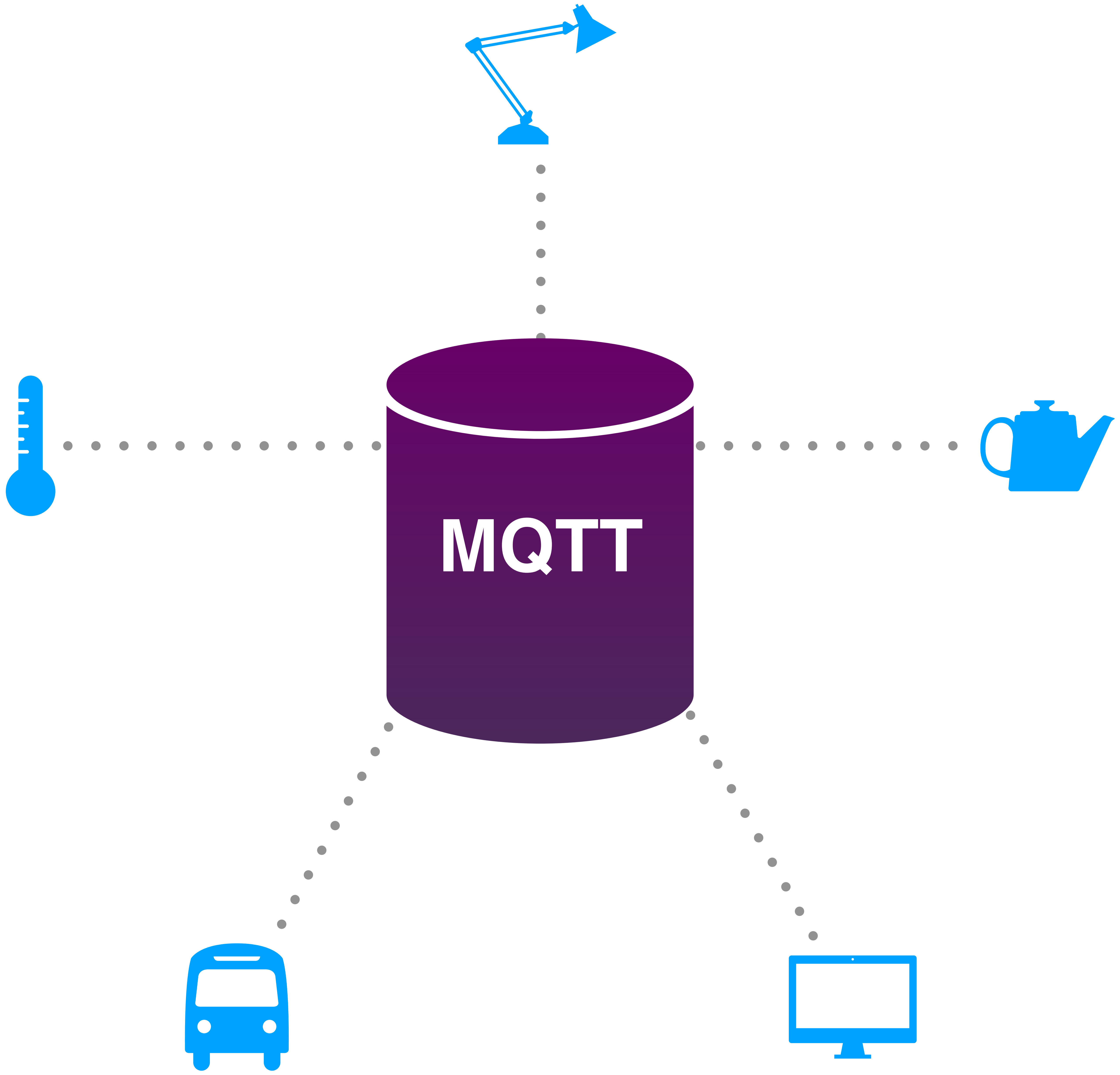
The road to MQTT 5 support in the Tortoise MQTT Client

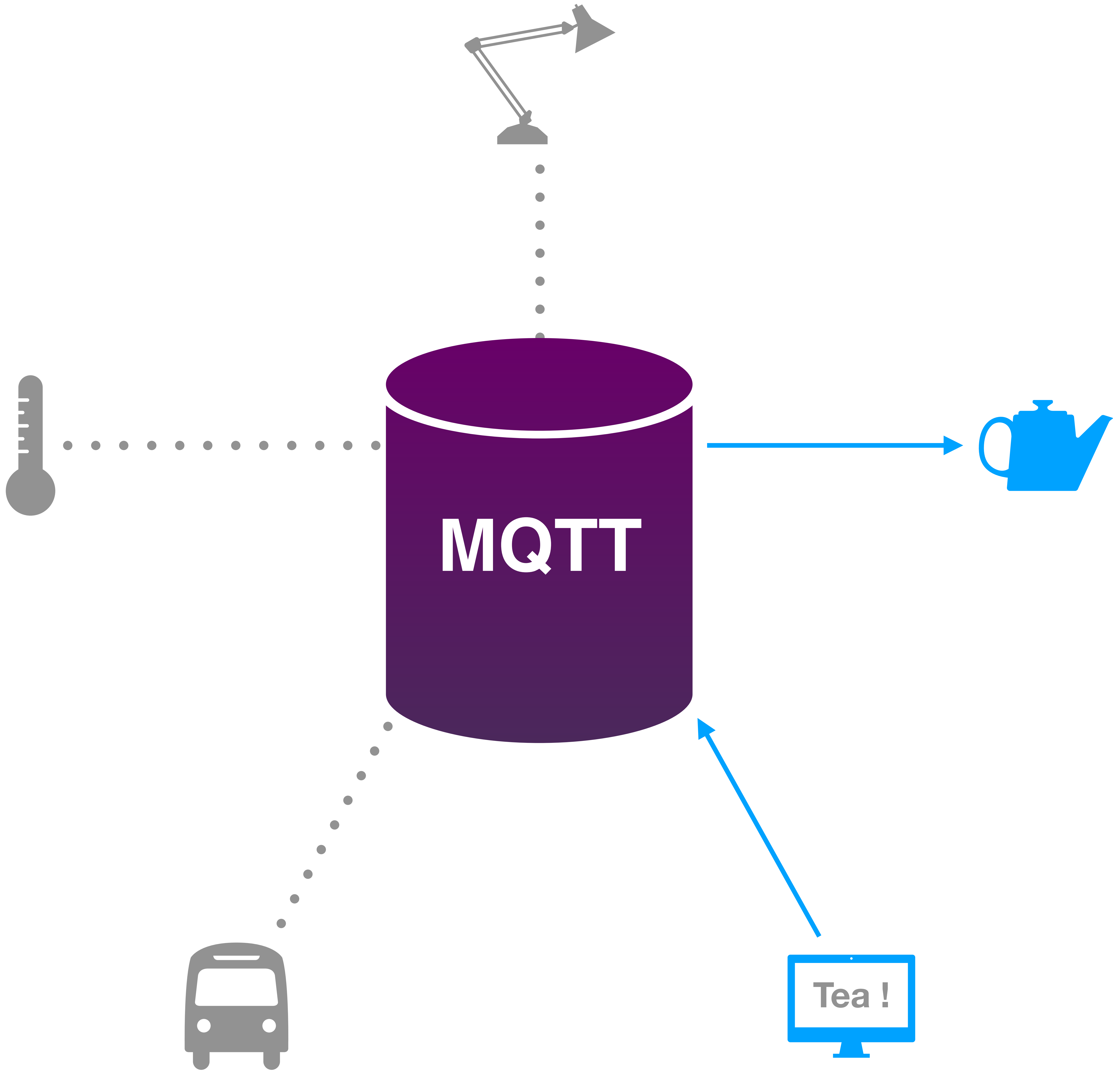
Martin Gausby
Senior Elixir Developer at Erlang Solutions

Agenda

- Introduction to the MQTT Protocol
- Introducing the Tortoise MQTT Client
- How some things change with MQTT 5

Introduction to MQTT


MQTT


MQTT
Tea !

«Topics» & «Topic Filters»

OCP Tower

ocp-tower/3/temperature

ocp-tower/2/temperature

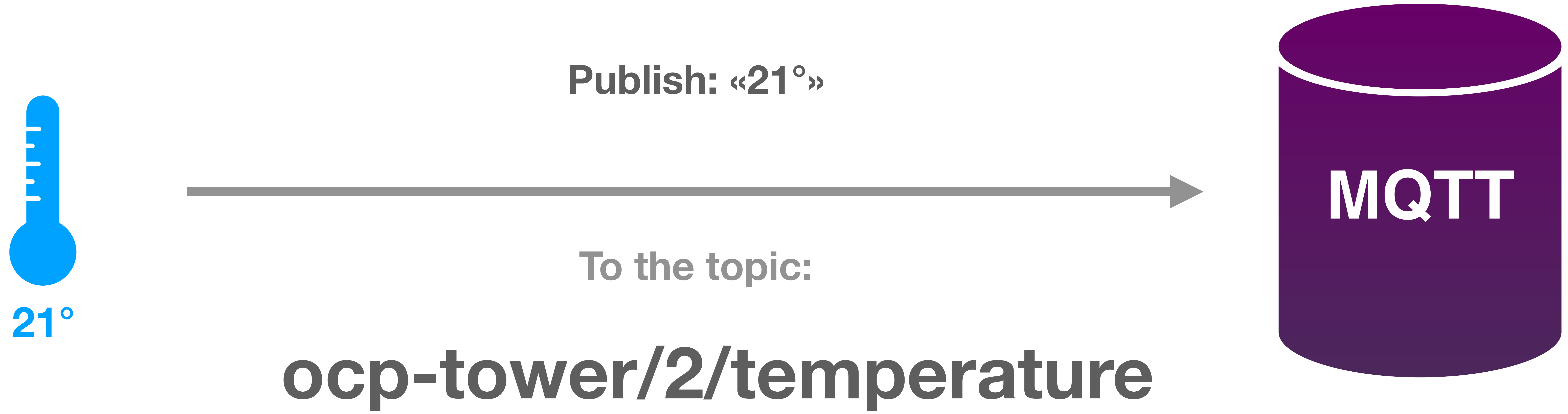
ocp-tower/1/temperature

Nakatomi Plaza

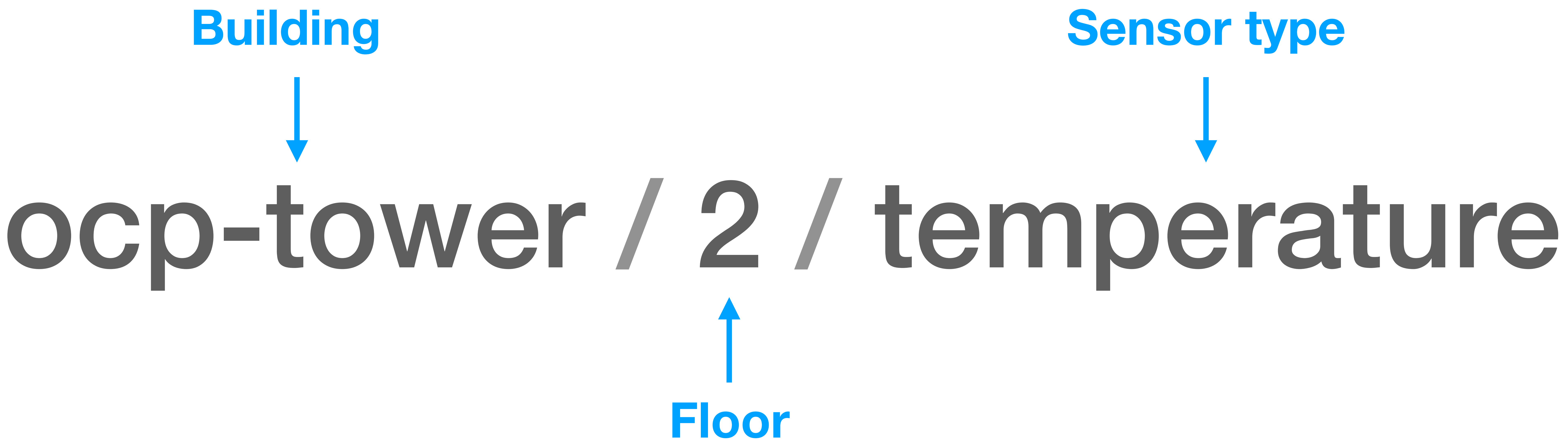
nakatomi-plaza/3/temperature

nakatomi-plaza/2/temperature

nakatomi-plaza/1/temperature


21°
Publish: «21°»
To the topic:
ocp-tower/2/temperature
MQTT

ocp-tower/2/temperature


Building
Sensor type
ocp-tower / 2 / temperature
Floor

OCP Tower

ocp-tower/3/temperature

ocp-tower/2/temperature

ocp-tower/1/temperature

Nakatomi Plaza

nakatomi-plaza/3/temperature

nakatomi-plaza/2/temperature

nakatomi-plaza/1/temperature

topic filter:

ocp-tower/2/temperature

OCP Tower

ocp-tower/3/temperature

ocp-tower/2/temperature

ocp-tower/1/temperature

Nakatomi Plaza

nakatomi-plaza/3/temperature

nakatomi-plaza/2/temperature

nakatomi-plaza/1/temperature

Single level wildcard

building / + / temperature

topic filter:

ocp-tower/+/temperature

OCP Tower

ocp-tower/3/temperature

ocp-tower/2/temperature

ocp-tower/1/temperature

Nakatomi Plaza

nakatomi-plaza/3/temperature

nakatomi-plaza/2/temperature

nakatomi-plaza/1/temperature

topic filter:

+/3/+

OCP Tower

ocp-tower/3/temperature

ocp-tower/2/temperature

ocp-tower/1/temperature

Nakatomi Plaza

nakatomi-plaza/3/temperature

nakatomi-plaza/2/temperature

nakatomi-plaza/1/temperature

Multi level wildcard

building / #

topic filter:

nakatomi-plaza/#

OCP Tower

ocp-tower/3/temperature

ocp-tower/2/temperature

ocp-tower/1/temperature

Nakatomi Plaza

nakatomi-plaza/3/temperature

nakatomi-plaza/2/temperature

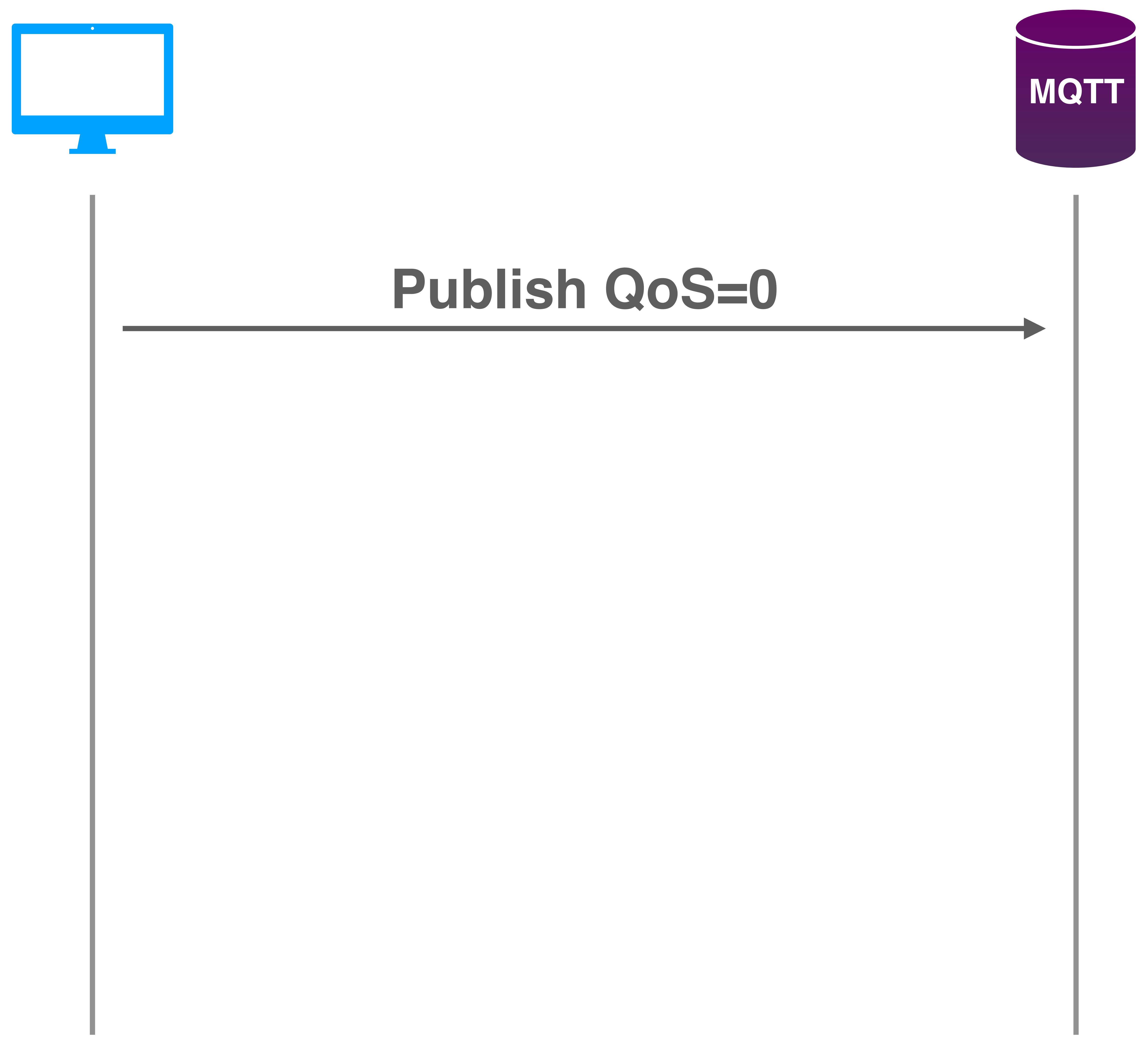
nakatomi-plaza/1/temperature

- A «topic» is used in a publish
- A «topic filter» is used in a subscription

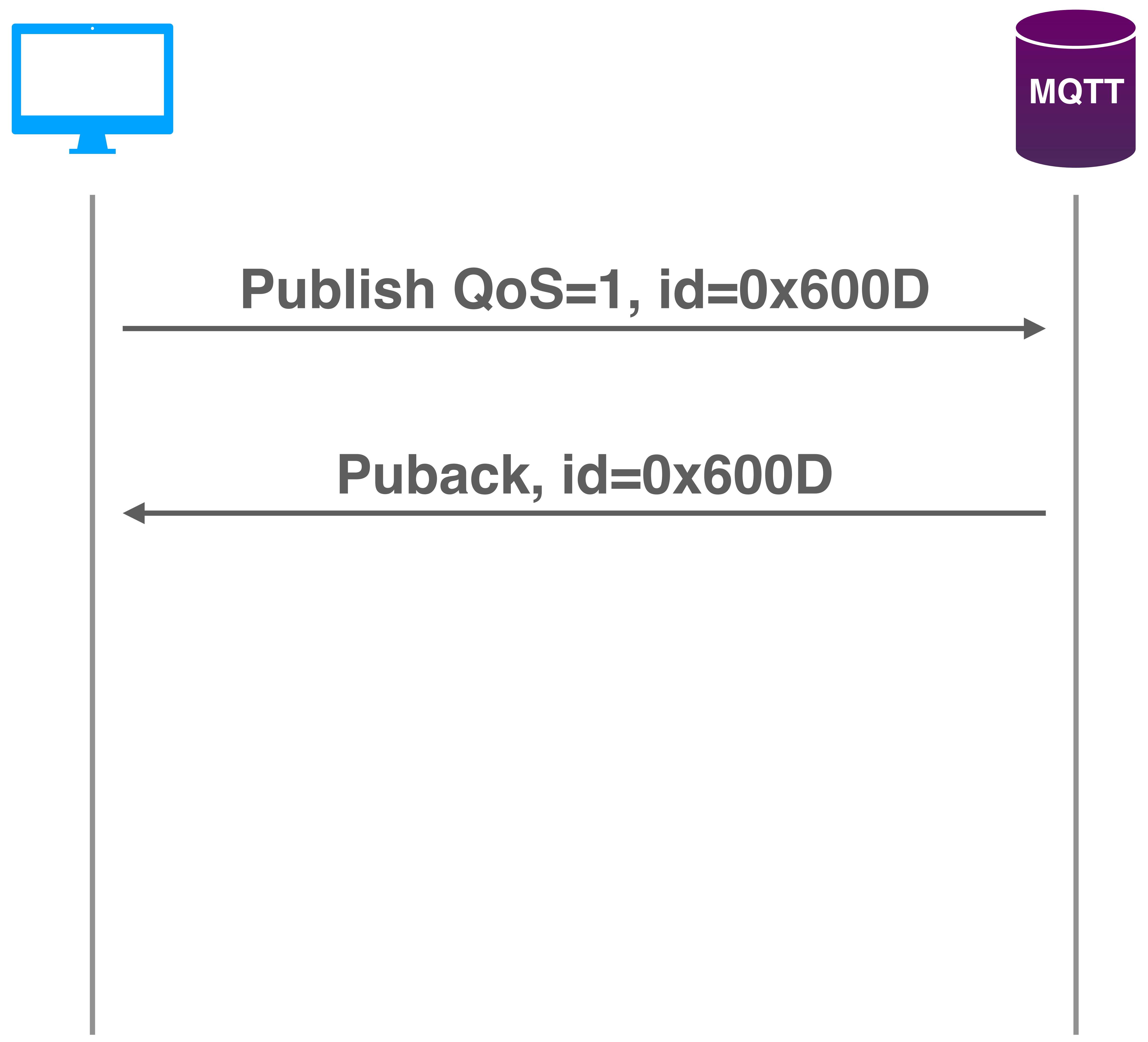
Quality of Service

- QoS=0 at most once
- QoS=1 at least once
- QoS=2 only once

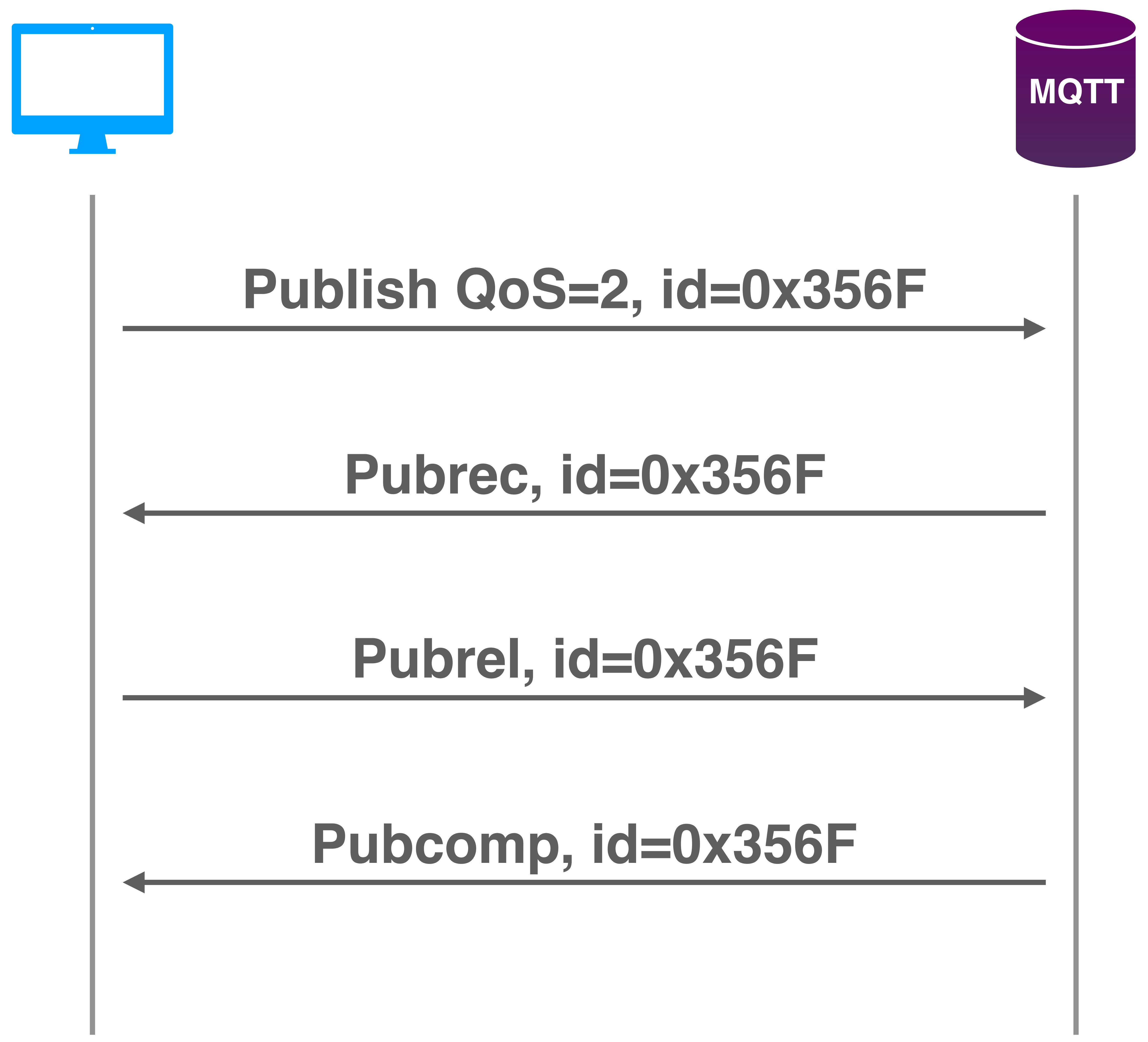
QoS=0; at most once delivery



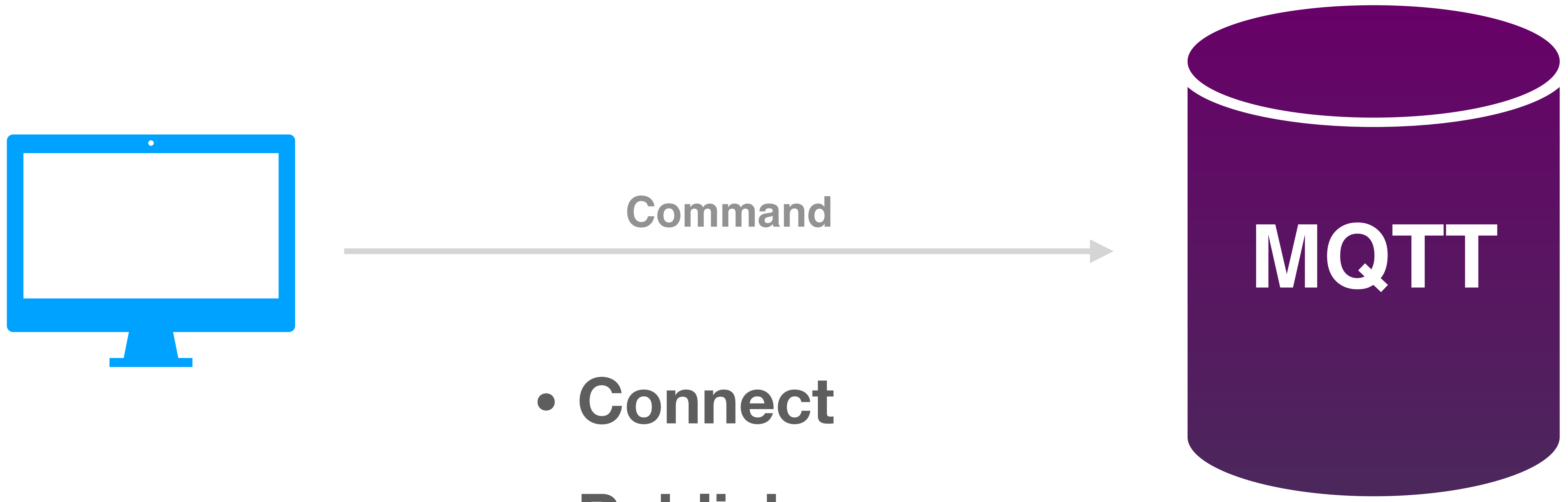
QoS=1; at least once delivery



QoS=2; only once delivery



The higher the QoS the more messages will get on the wire. *Always pick the lowest you can get away with*.



- Connect
- Publish
- Subscribe
- Unsubscribe
- Ping
- Disconnect

Tortoise MQTT

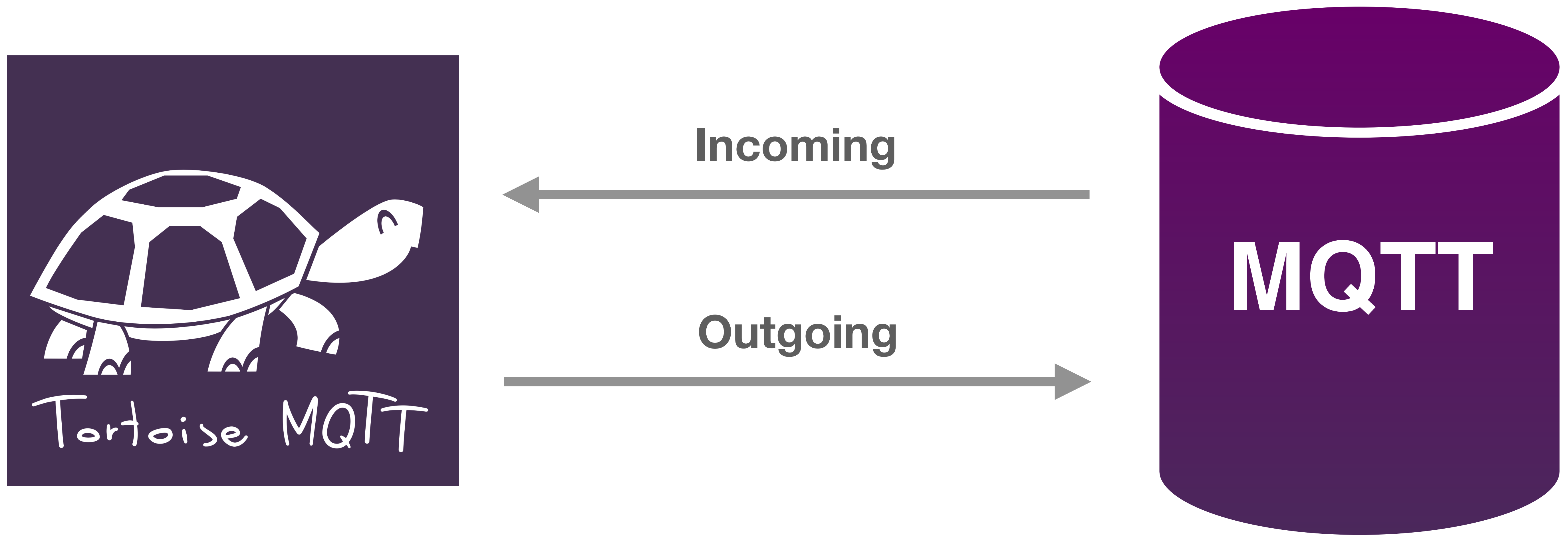
Logo by @LRTVRI
Follow him on Twitter!

- The client should allow everything the protocol allows
- The author should keep their opinions out of it (but may provide defaults)
- Hide things that can be automated

A client should *hide the protocol details that can be hidden* from the user,
without limiting the user in what they want to do

Map Elixir semantics to MQTT Semantics & figure out the stuff that can happen behind the scenes

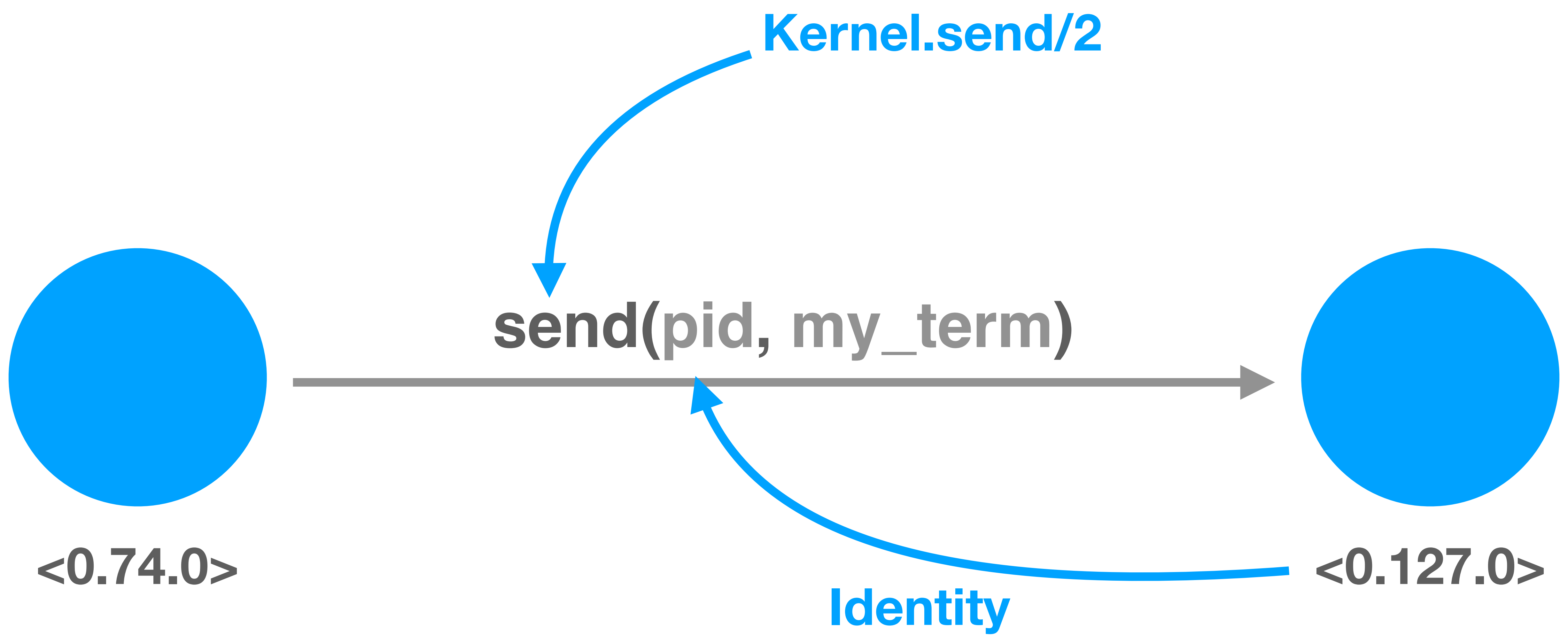
Semantics

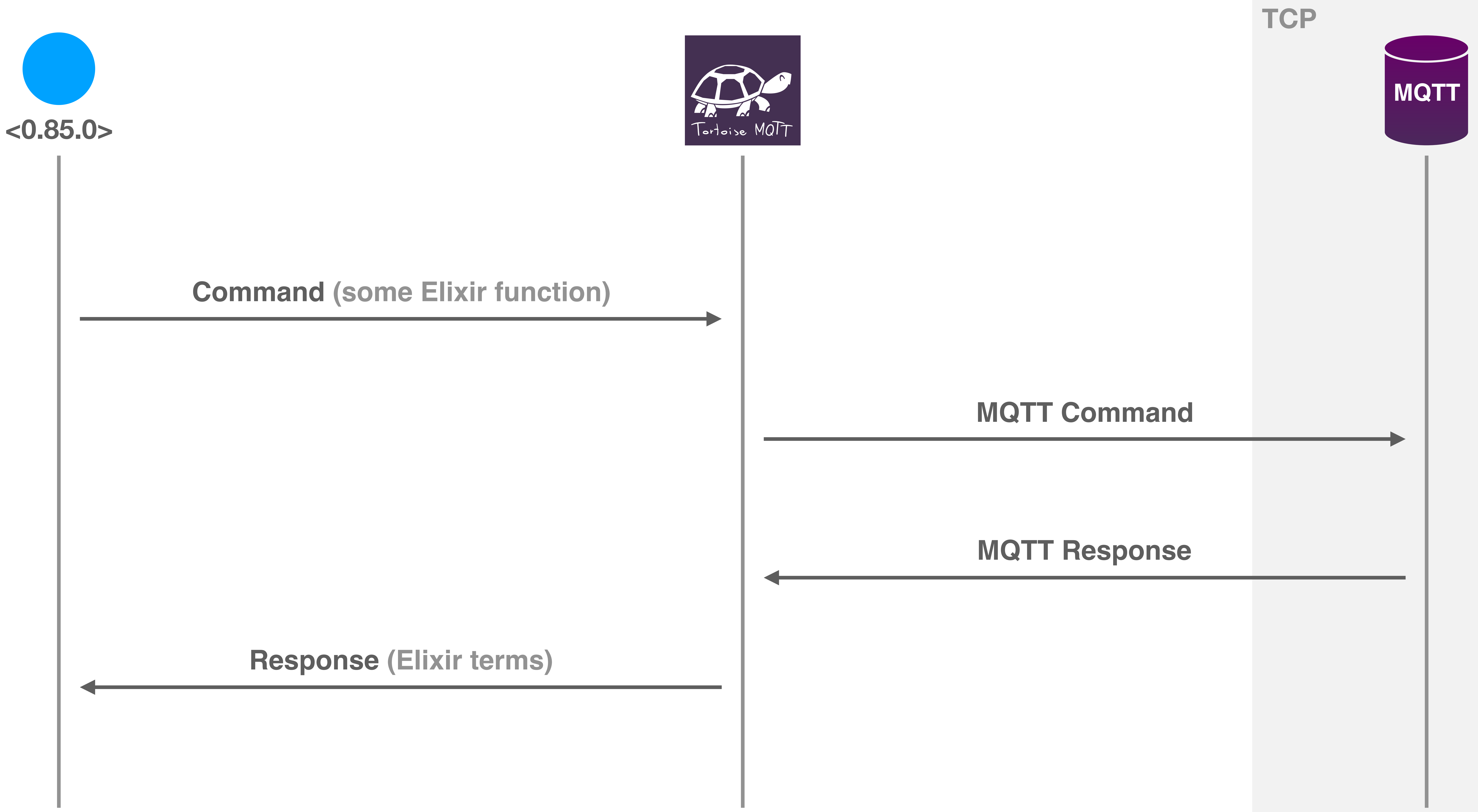


Outgoing Messages

Life-cycle

Elixir Semantics




TCP
<0.85.0>
Tortoise MQTT
MQTT
Command (some Elixir function)
MQTT Command
MQTT Response
Response (Elixir terms)

“Identity”

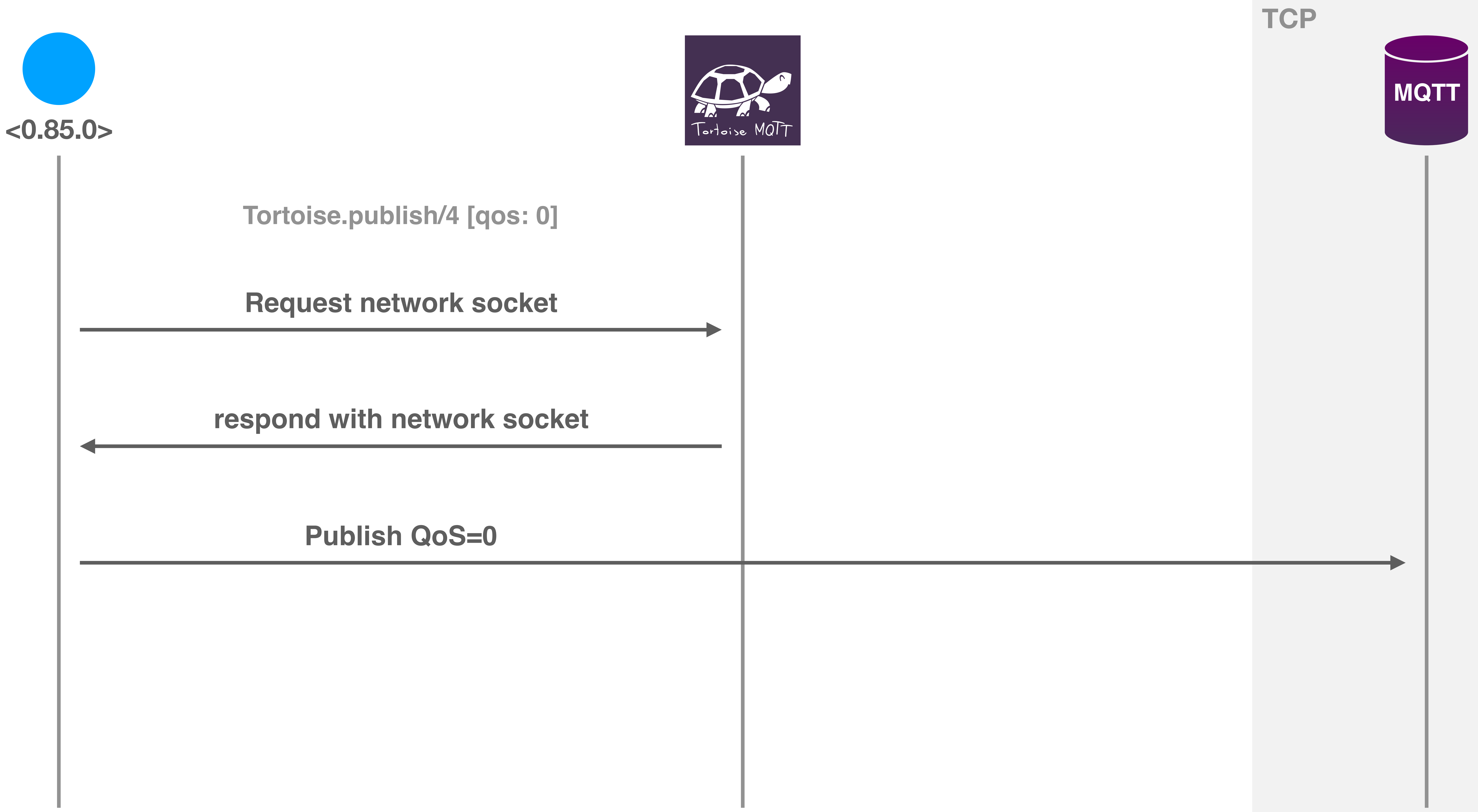
```
client_id = "toes"
```

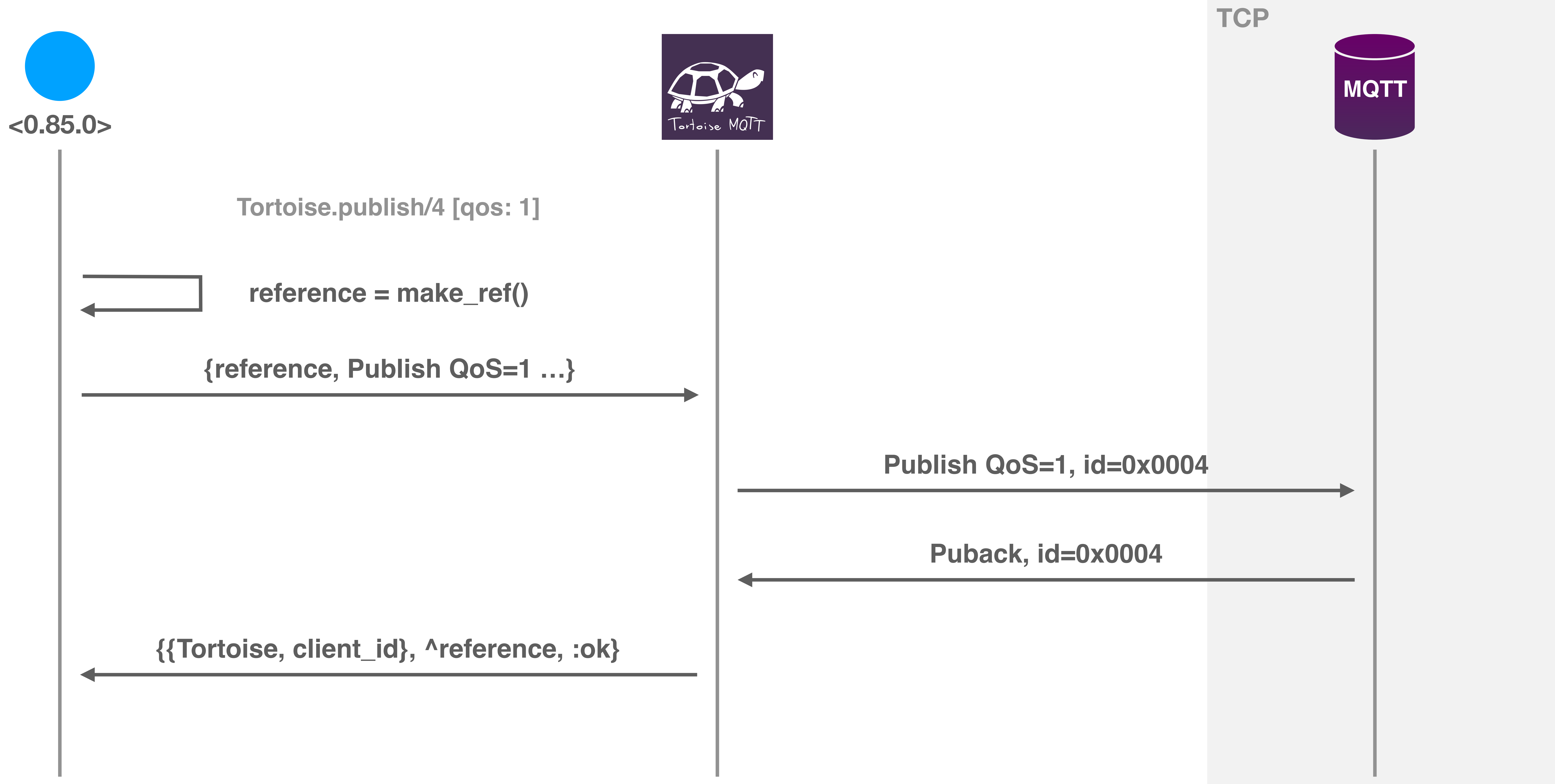
```elixir
client_id = "toes"

{:ok, pid} = Tortoise.Connection.start_link(
  client_id: client_id,
  server: {Tortoise.Transport.Tcp, host: 'localhost', port: 1883},
  handler: {Tortoise.Handler.Default, []}
)
```

```
client_id = "toes"

{:ok, pid} = Tortoise.Connection.start_link(
  client_id: client_id,
  server: {Tortoise.Transport.Tcp, host: 'localhost', port: 1883},
  handler: {Tortoise.Handler.Default, []}
)
```

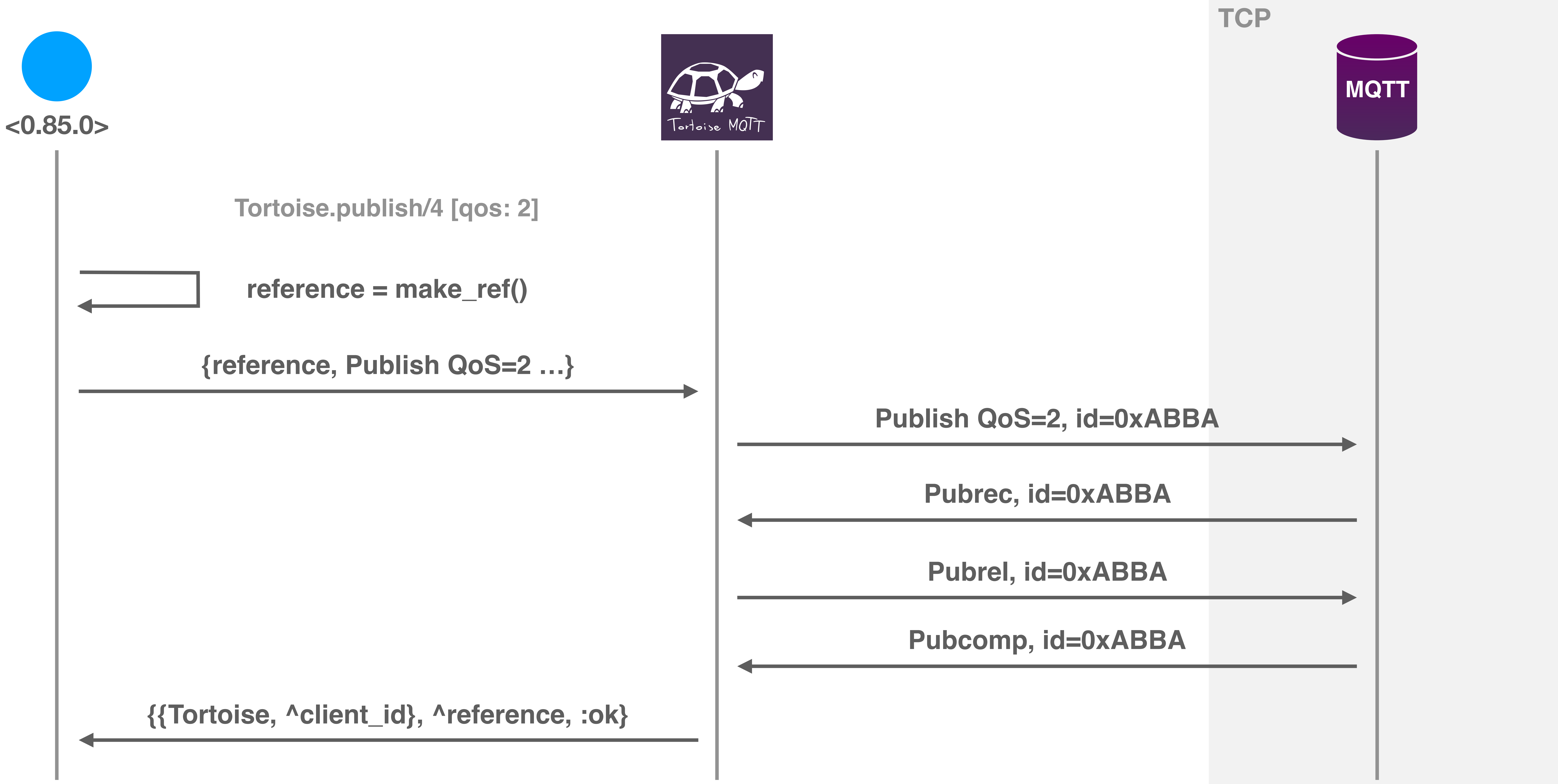
```elixir
topic = "ocp-tower/3/temperature"

payload = <<21::float-32>>

Tortoise.publish(client_id, topic, payload, [qos: 0])

# returns :ok
```

```
topic = "ocp-tower/3/temperature"
payload = <<21::float-32>>
Tortoise.publish(client_id, topic, payload, [qos: 0])
# returns :ok
```


TCP
<0.85.0>
Tortoise MQTT
MQTT
Tortoise.publish/4 [qos: 0]
Request network socket
respond with network socket
Publish QoS=0



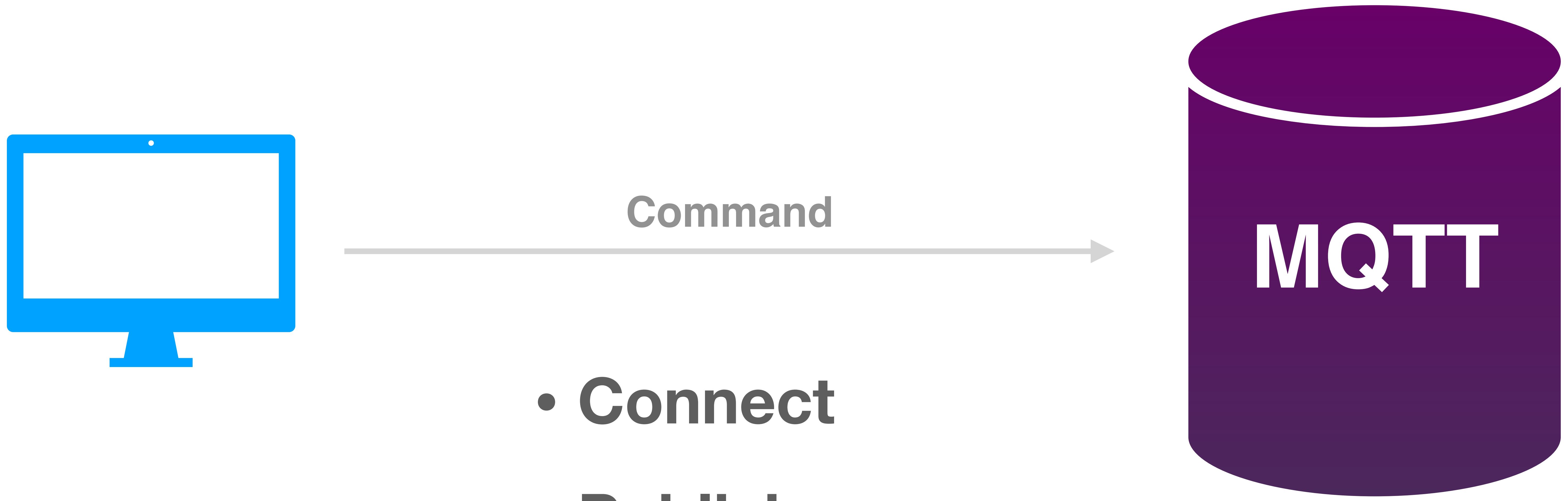
TCP
<0.85.0>
Tortoise MQTT
MQTT
Tortoise.publish/4 [qos: 1]
reference = make_ref()
{reference, Publish QoS=1 …}
Publish QoS=1, id=0x0004
Puback, id=0x0004
{{Tortoise, client_id}, ^reference, :ok}


TCP
<0.85.0>
Tortoise MQTT
MQTT
Tortoise.publish/4 [qos: 2]
reference = make_ref()
{reference, Publish QoS=2 …}
Publish QoS=2, id=0xABBA
Pubrec, id=0xABBA
Pubrel, id=0xABBA
Pubcomp, id=0xABBA
{{Tortoise, ^client_id}, ^reference, :ok}

```
{{Tortoise, ^client_id}, ^reference, :ok}
```

```elixir
receive do
  {{Tortoise, ^client_id}, ^reference, result} ->
    {:ok, result}
after 200 ->
  {:error, :timeout}
end
```

- Tortoise.Connection.start_link/3
- Tortoise.publish/4
- Tortoise.Connection.subscribe/3
- Tortoise.Connection.unsubscribe/3
- Tortoise.Connection.ping/1
- Tortoise.Connection.disconnect/1



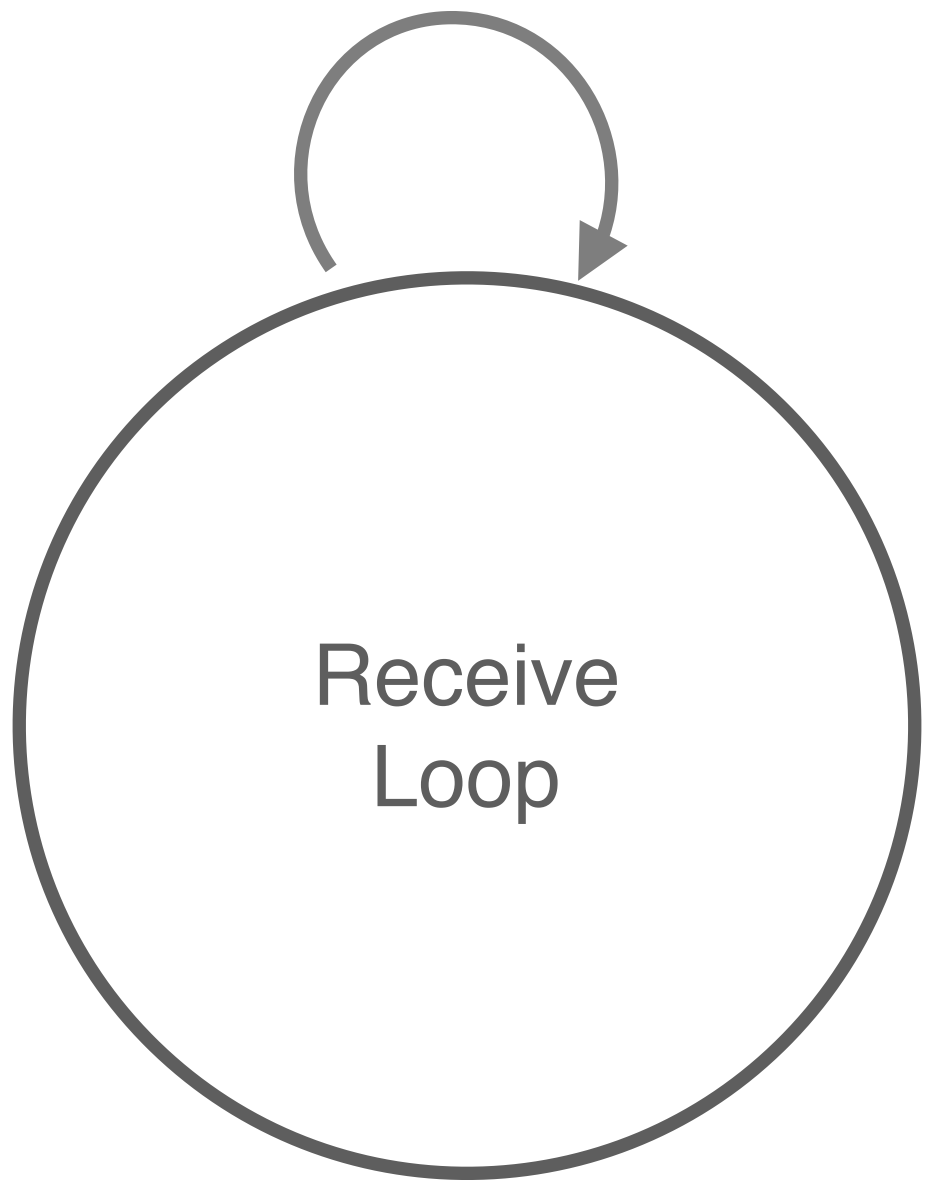
- Connect
- Publish
- Subscribe
- Unsubscribe
- Ping
- Disconnect

Incoming Messages

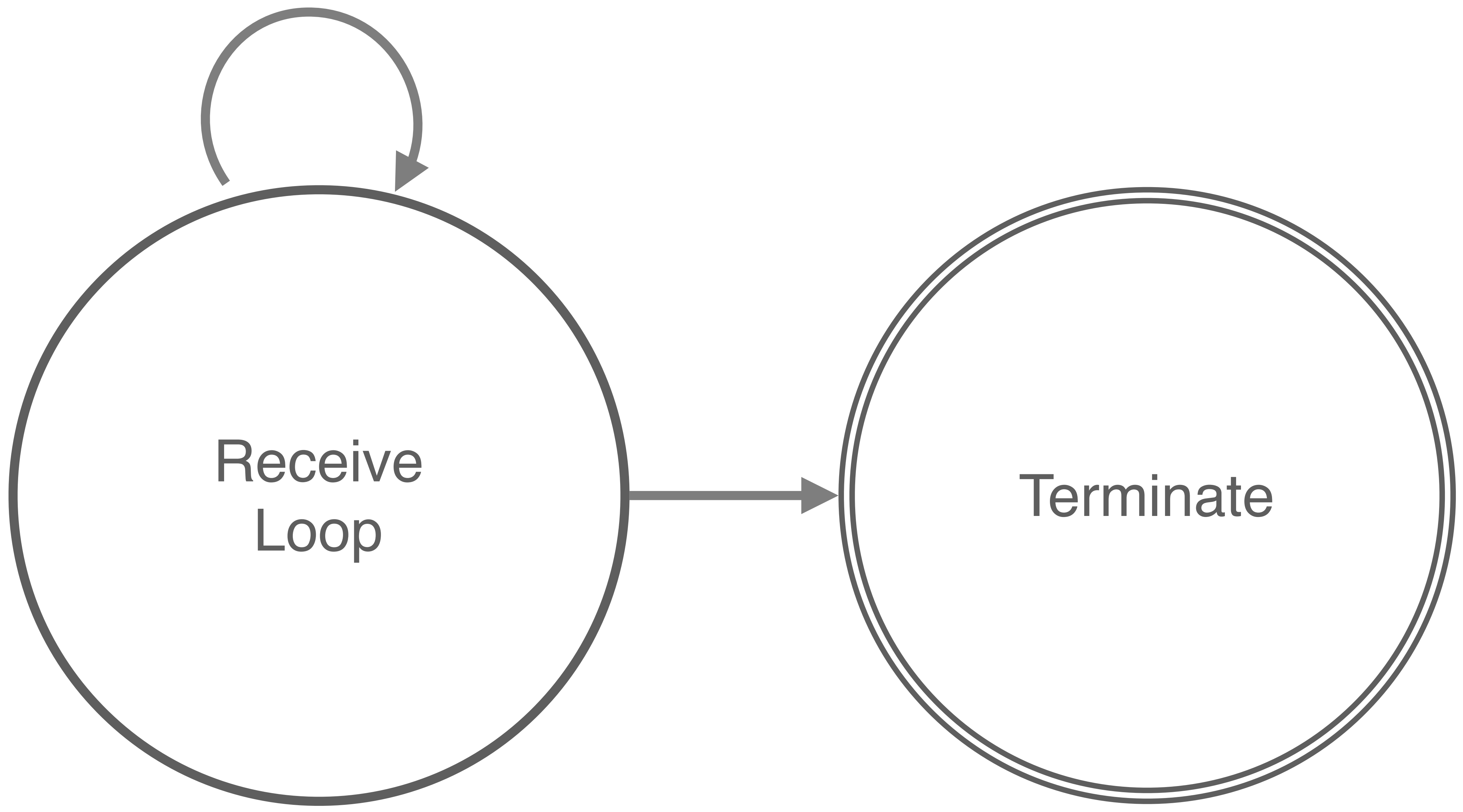
Life-cycle

Connection Life-cycle

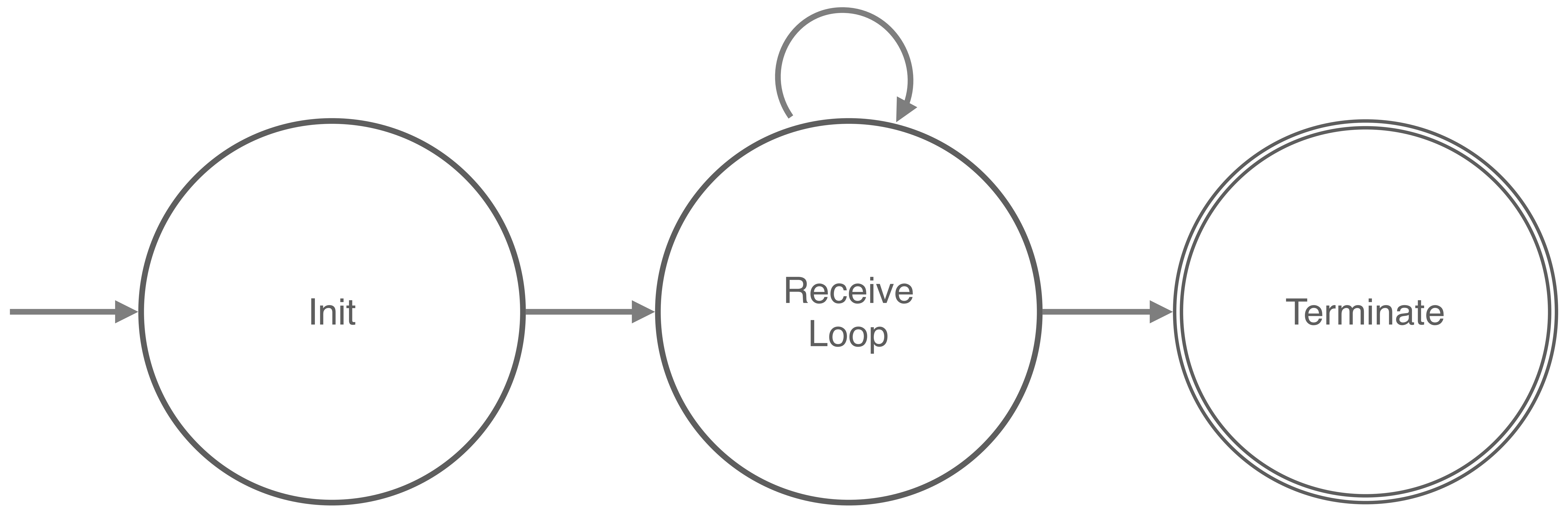
GenServer Life-cycle



GenServer Life-cycle

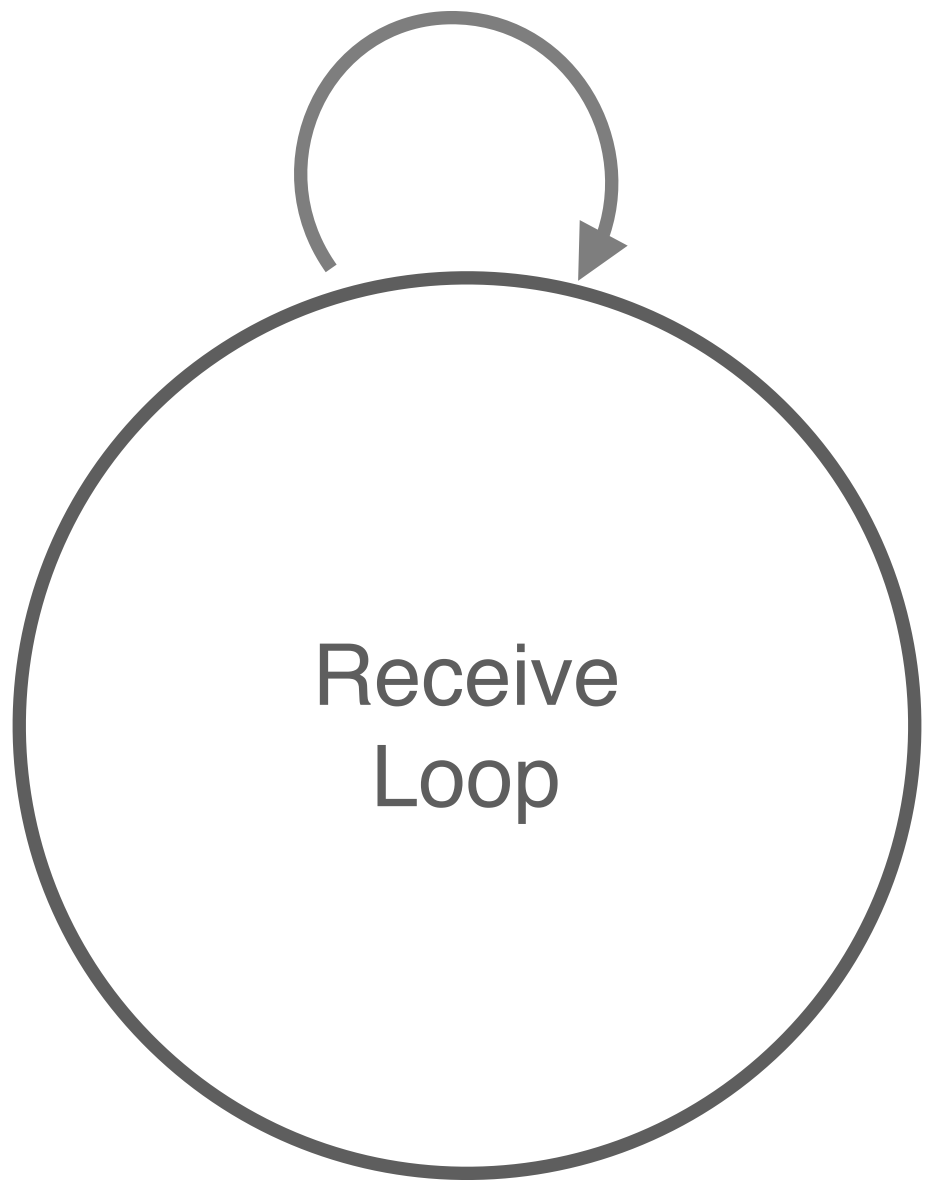


GenServer Life-cycle

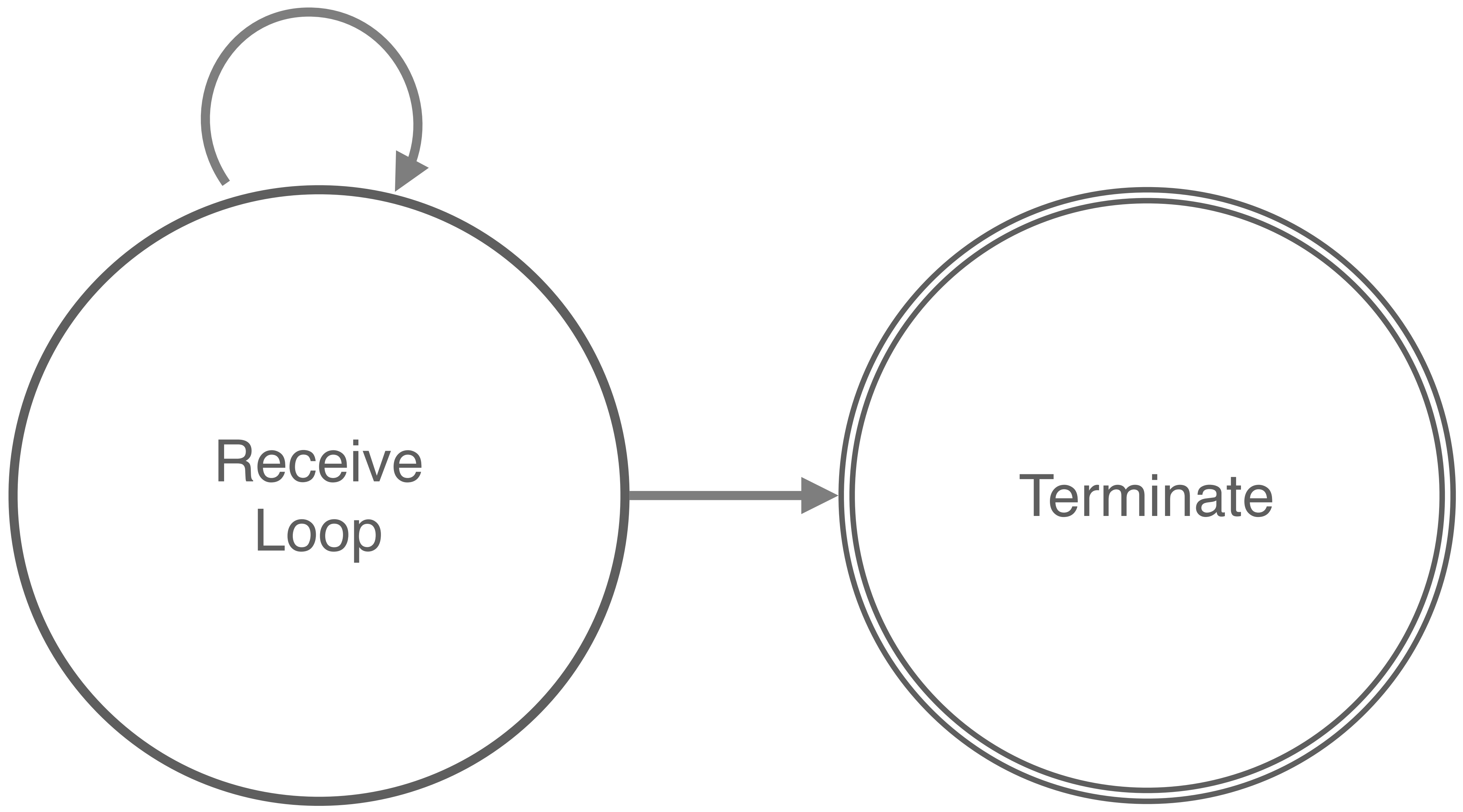


```
@behaviour Tortoise.Handler
```

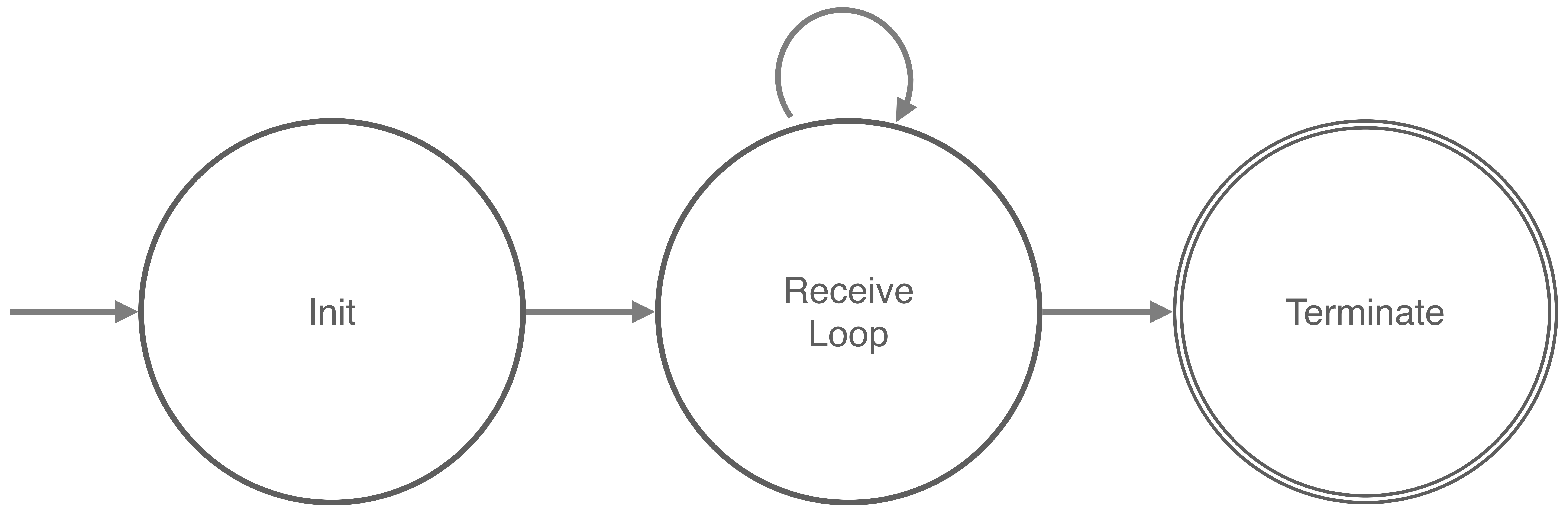
Tortoise.Handler Life-cycle



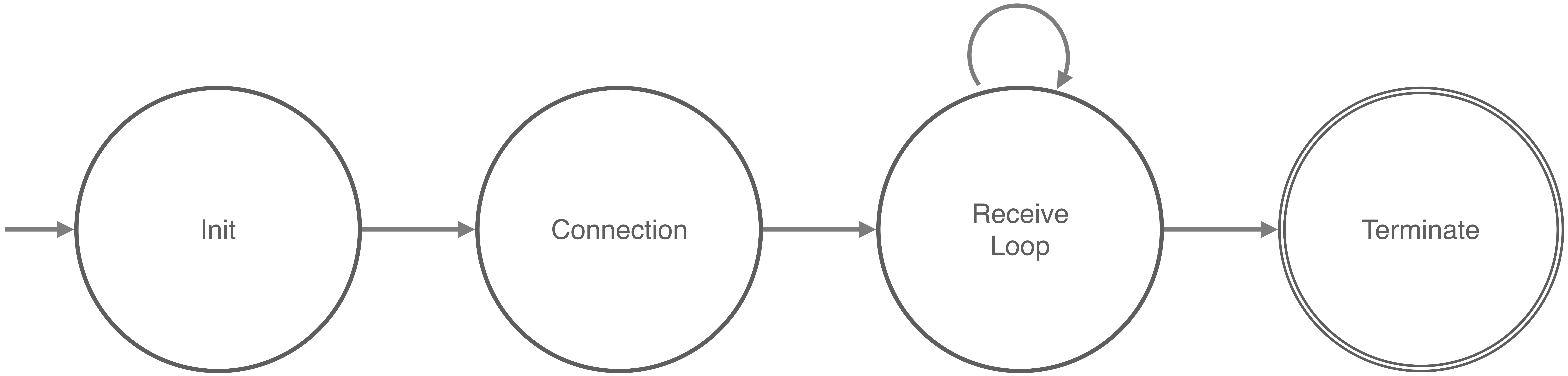
Tortoise.Handler Life-cycle



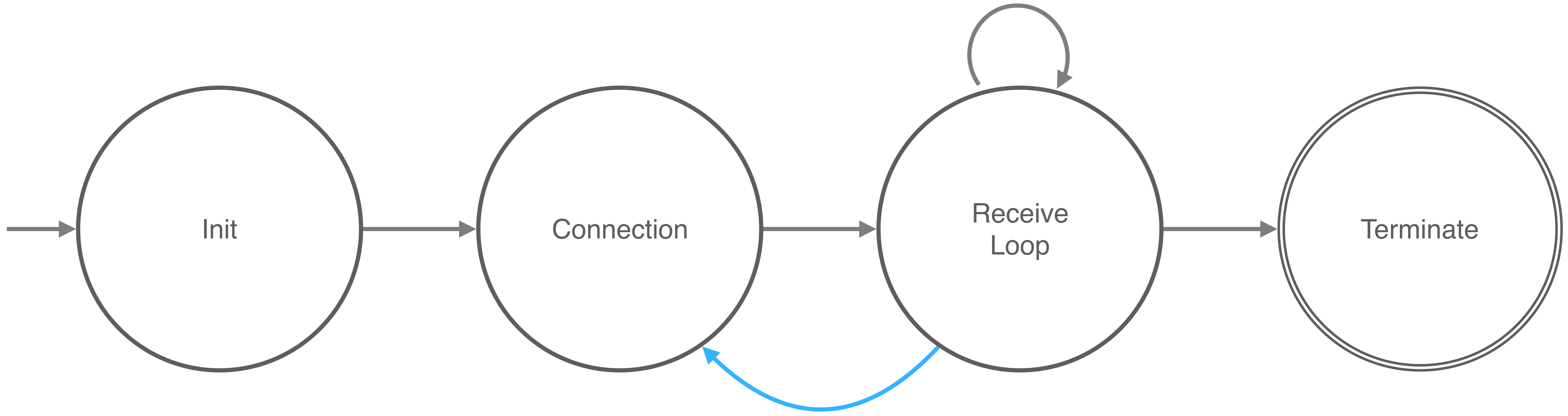
Tortoise.Handler Life-cycle



Tortoise.Handler Life-cycle



Tortoise.Handler Life-cycle



Tortoise.Handler Callbacks

```
init(argument)

connection(status, state) # status: :up | :down

subscription(status, topic_filter, state) # status: :up | :down

handle_message(topic_list, payload, state)

terminate(reason, state)
```

```
handle_message(topic_list, payload, state)
```

topic_list

ocp-tower / 3 / temperature

["ocp-tower", "3", "temperature"]

topic_list

ocp-tower / + / temperature

["ocp-tower", _floor, "temperature"]

topic_list

```
ocp-tower / #

["ocp-tower" | _topic_levels]
```

```
def MyHandler do
  use Tortoise.Handler
  # …
  def handle_message([building, floor, "temperature"], payload, state) do
    # do stuff with data
    {:ok, state}
  end
end
```

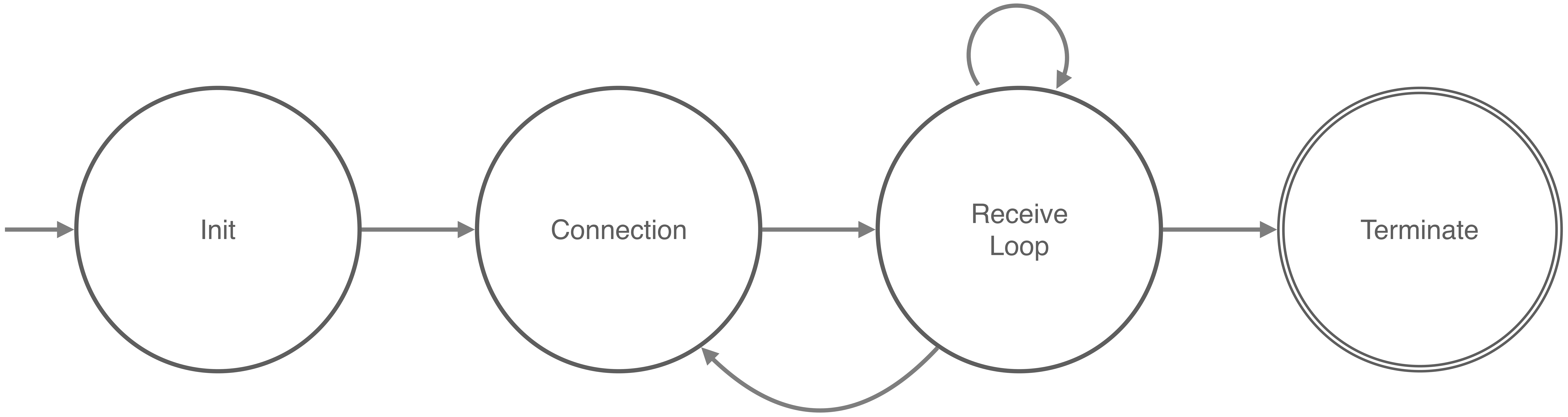
MQTT 5

New in MQTT 5

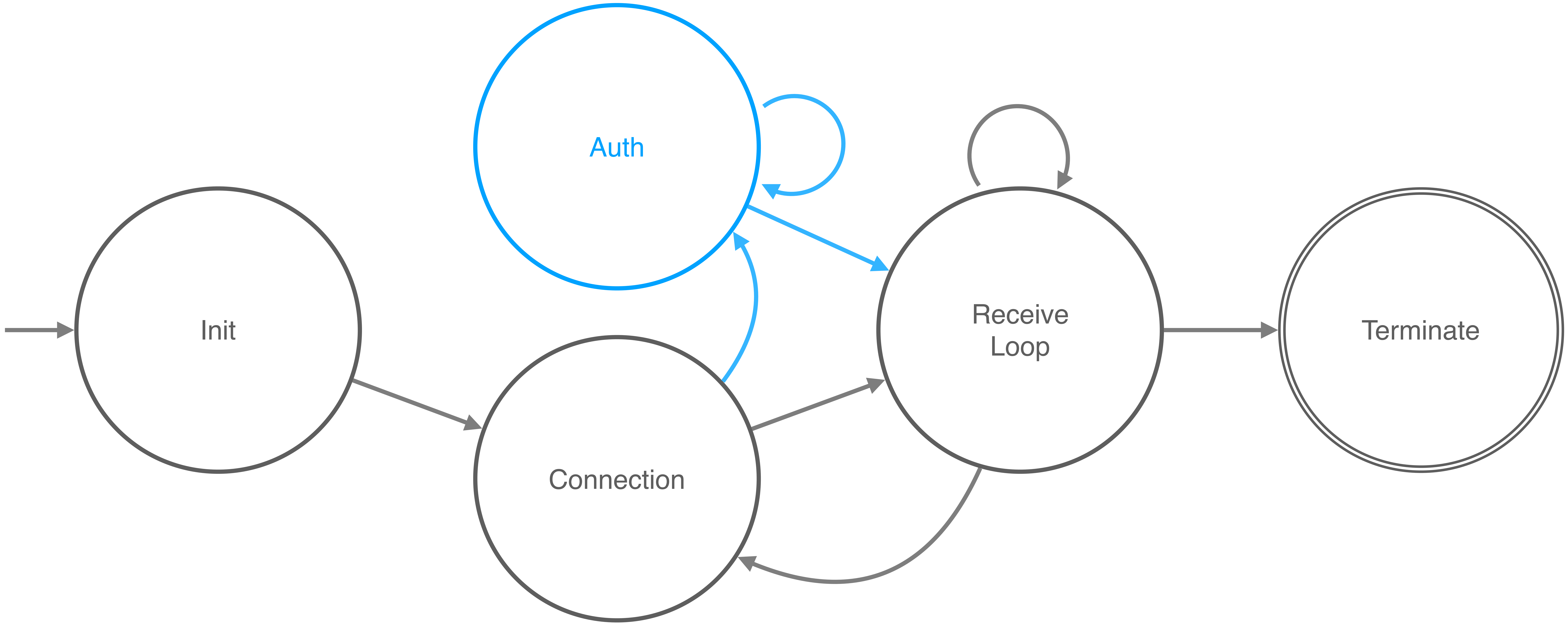
- Session expiry (and message expiry)
- Verbose error messages (reason codes on acks, disconnect, etc)
- Capability negotiation on connect (maximum package size allowed, etc)
- Formalisation of some community patterns (RPC, shared subscriptions, etc)
- Enhanced authorisation
- User defined properties
- …and more

Enhanced Auth

Tortoise.Handler Life-cycle

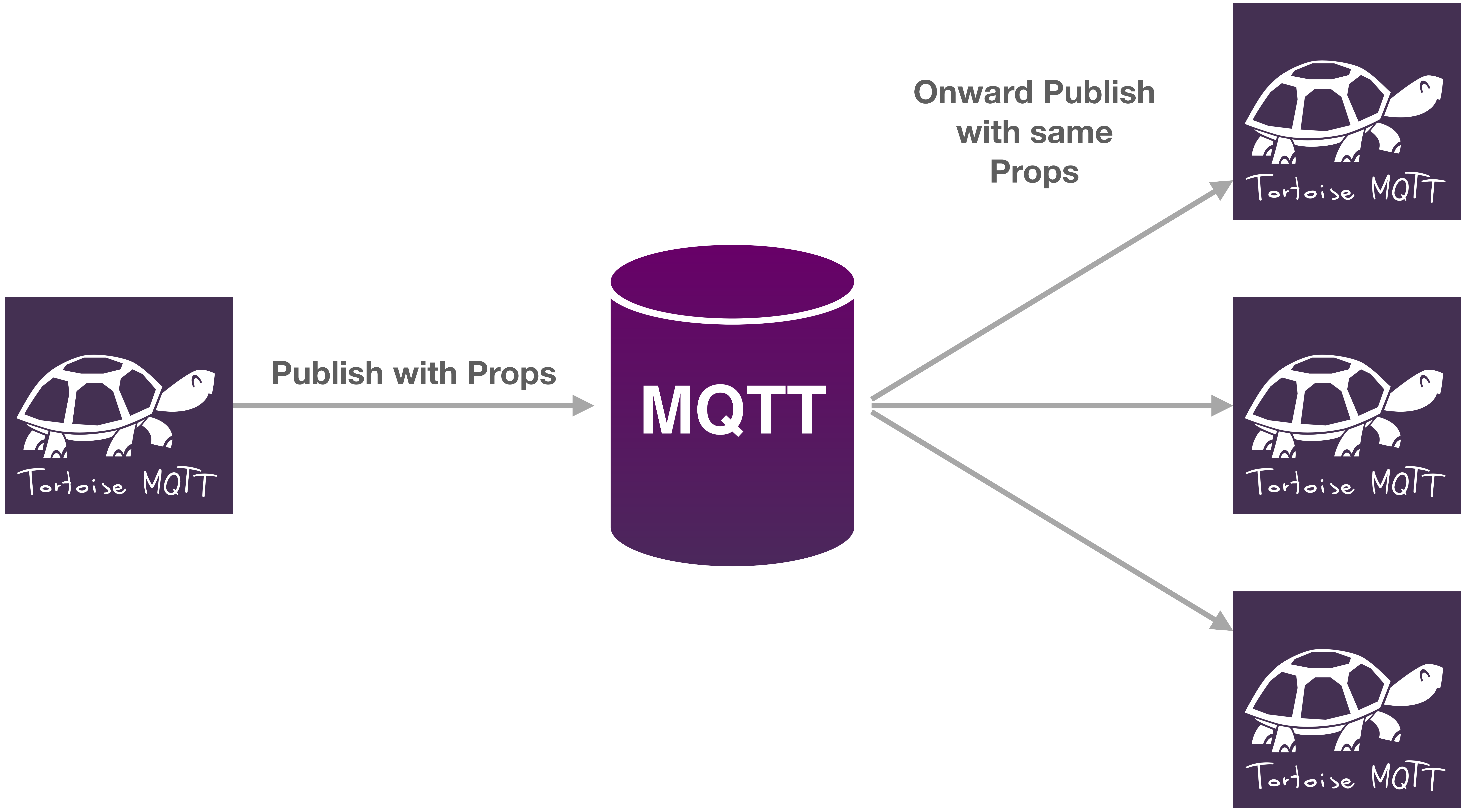


Tortoise.Handler Life-cycle



User Properties

```
Tortoise.publish(client_id, "nakatomi-plaza/3/temperature", <<21::float-32>>,
  user_properties: [ {"scale", "celsius"},
                     {"sensor-id", "a0478625"} ]
)
```

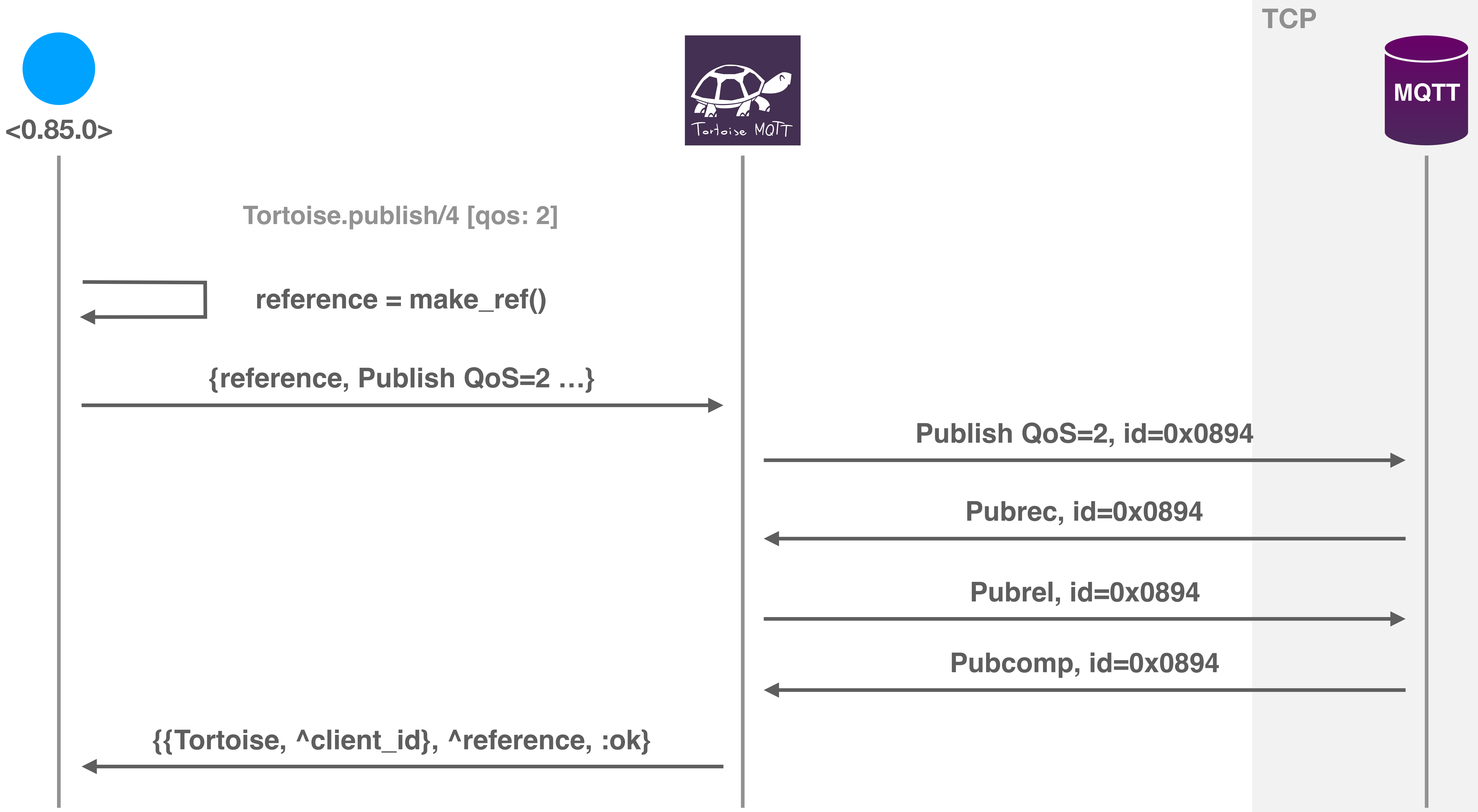

Publish with Props
MQTT
Onward Publish with same Props
Tortoise MQTT

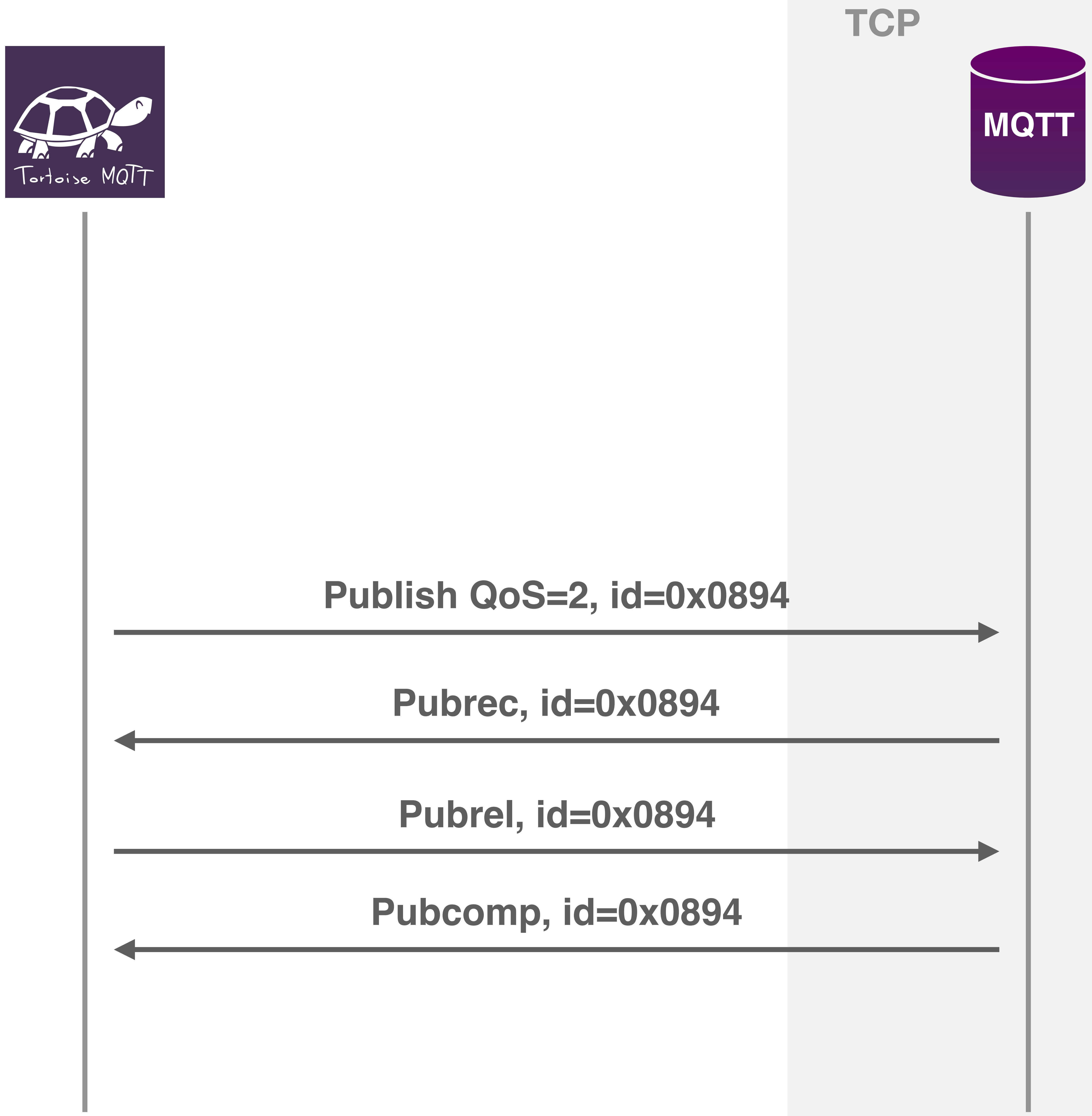
```
def MyHandler do
  use Tortoise.Handler
  # …
  def handle_message(topic_list, payload, state) do
    # do stuff with data
    {:ok, state}
  end
end
```

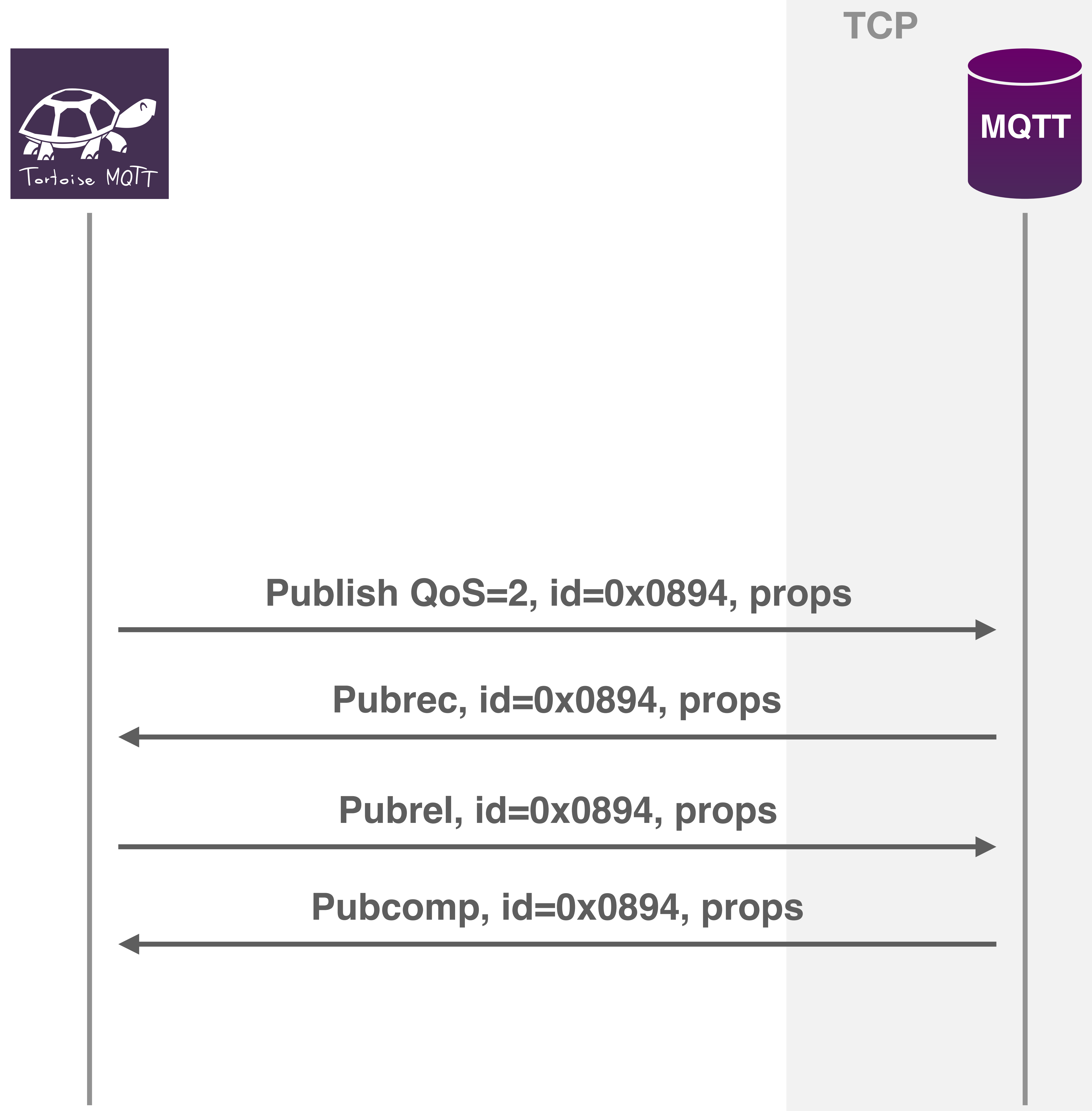
```
def MyHandler do
  use Tortoise.Handler
  # …
  def handle_message(topic_list, %Publish{} = publish, state) do
    # do stuff with data
    #   - publish.payload
    #   - publish.properties, etc.
    {:ok, state}
  end
end
```

No big deal really

…except other protocol packages have user properties as well


TCP
<0.85.0>
Tortoise MQTT
MQTT
Tortoise.publish/4 [qos: 2]
reference = make_ref()
{reference, Publish QoS=2 …}
Publish QoS=2, id=0x0894
Pubrec, id=0x0894
Pubrel, id=0x0894
Pubcomp, id=0x0894
{{Tortoise, ^client_id}, ^reference, :ok}


TCP
Tortoise MQTT
MQTT
Publish QoS=2, id=0x0894
Pubrec, id=0x0894
Pubrel, id=0x0894
Pubcomp, id=0x0894


TCP
Tortoise MQTT
MQTT
Publish QoS=2, id=0x0894, props
Pubrec, id=0x0894, props
Pubrel, id=0x0894, props
Pubcomp, id=0x0894, props

- The client should allow everything the protocol allows
- The author should keep their opinions out of it (but may provide defaults)
- Hide things that can be automated

- ~~The client should allow everything the protocol allows~~
- ~~The author should keep their opinions out of it (but may provide defaults)~~
- Hide things that can be automated

Tortoise.Handler Callbacks

```
init(argument)

connection(status, state) # status: :up | :down

subscription(status, topic_filter, state) # status: :up | :down

handle_message(topic_list, payload, state)

terminate(reason, state)
```

Tortoise.Handler Callbacks

```
init(argument)
handle_auth(%Auth{}, state)
connection(status, state) # status: :up | :down
handle_connack(%Connack{}, state)
handle_publish(topic_levels, %Publish{}, state)
handle_puback(%Puback{}, state)
handle_pubcomp(%Pubcomp{}, state)
handle_pubrec(%Pubrec{}, state)
handle_pubrel(%Pubrel{}, state)
handle_suback(%Subscribe{}, %Suback{}, state)
handle_unsuback(%Unsubscribe{}, %Unsuback{}, state)
handle_disconnect(%Disconnect{}, state)
terminate(reason, state)
```

Conclusion

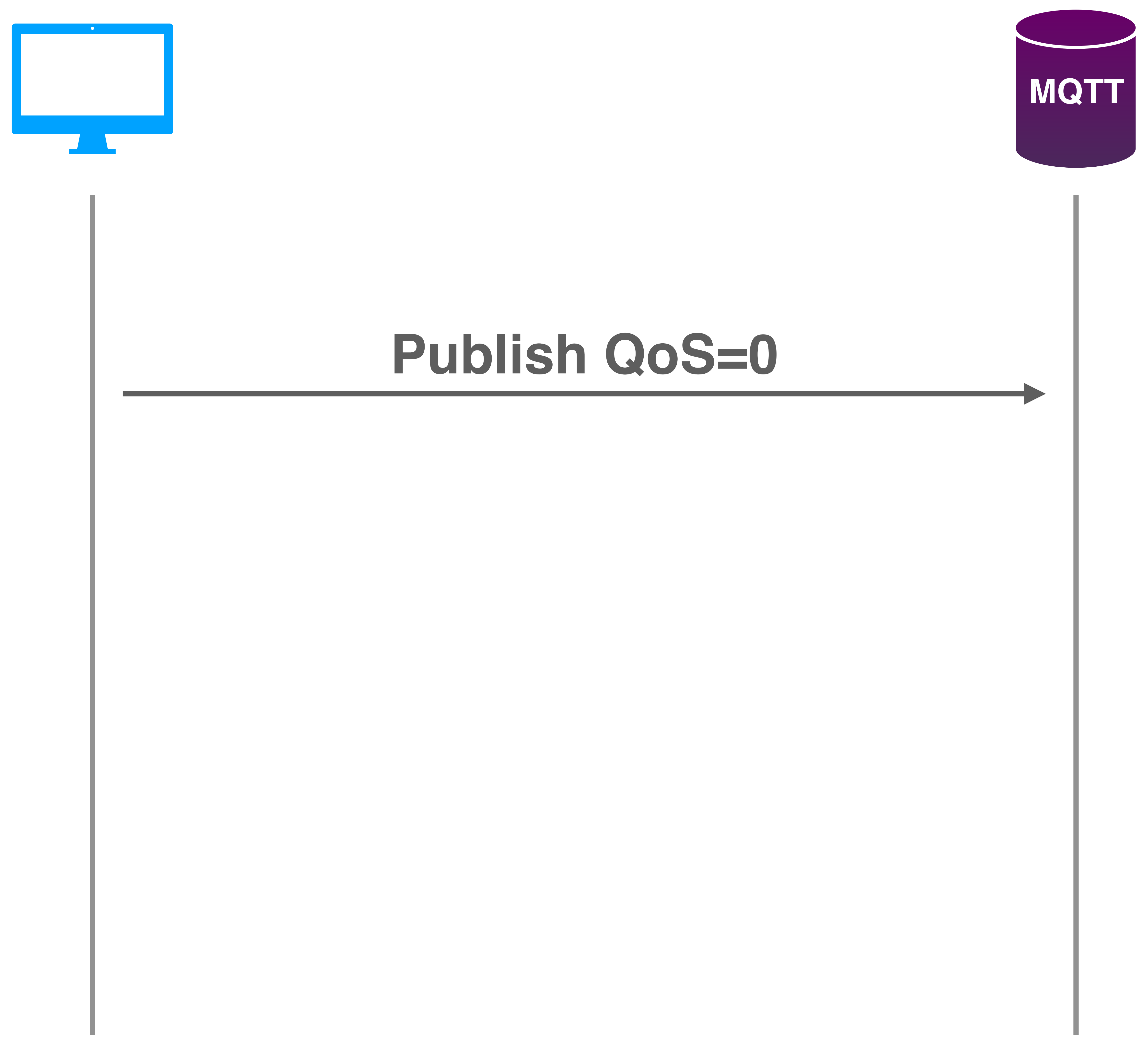
- Tortoise (for MQTT 3.1.1) is ready for production today
- MQTT 5 will cause some changes to the API
- MQTT 5 support in a branch, kinda works, but need some time
- Tortoise will drift towards a low-level API to support everything

Tortoise MQTT

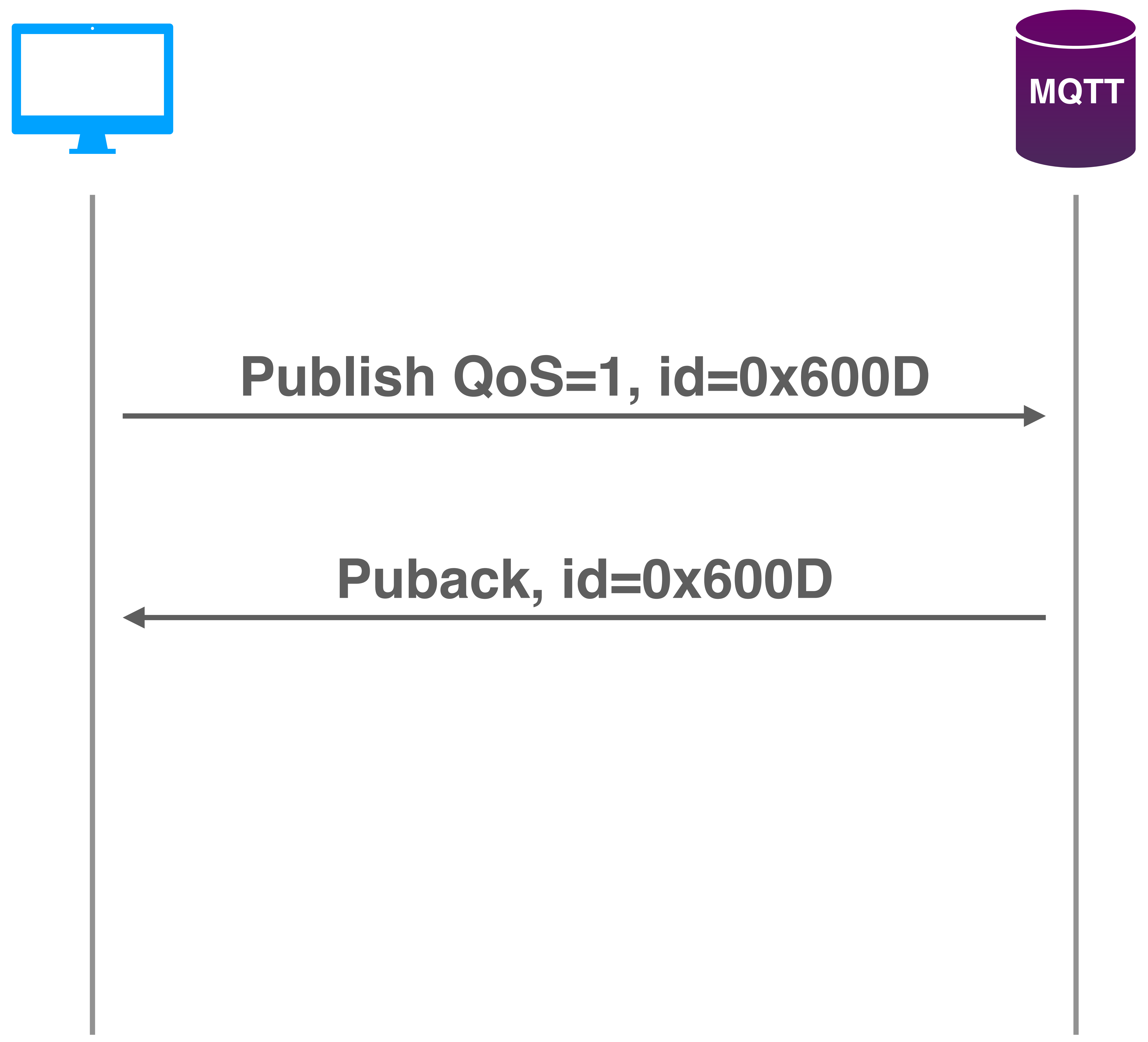
https://github.com/gausby/tortoise

Logo by @LRTVRI
Follow him on Twitter!

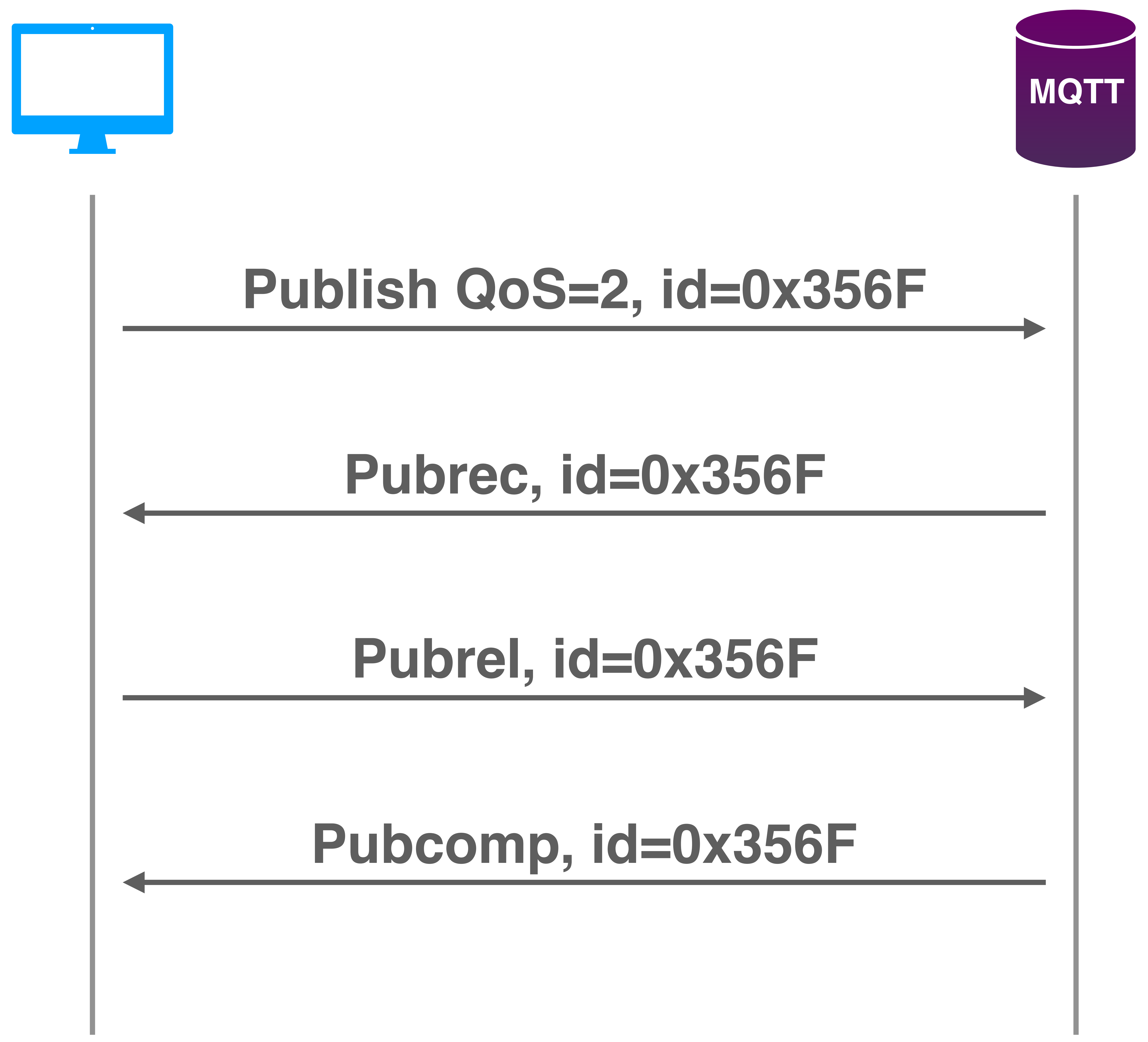
QoS=0; at most once delivery

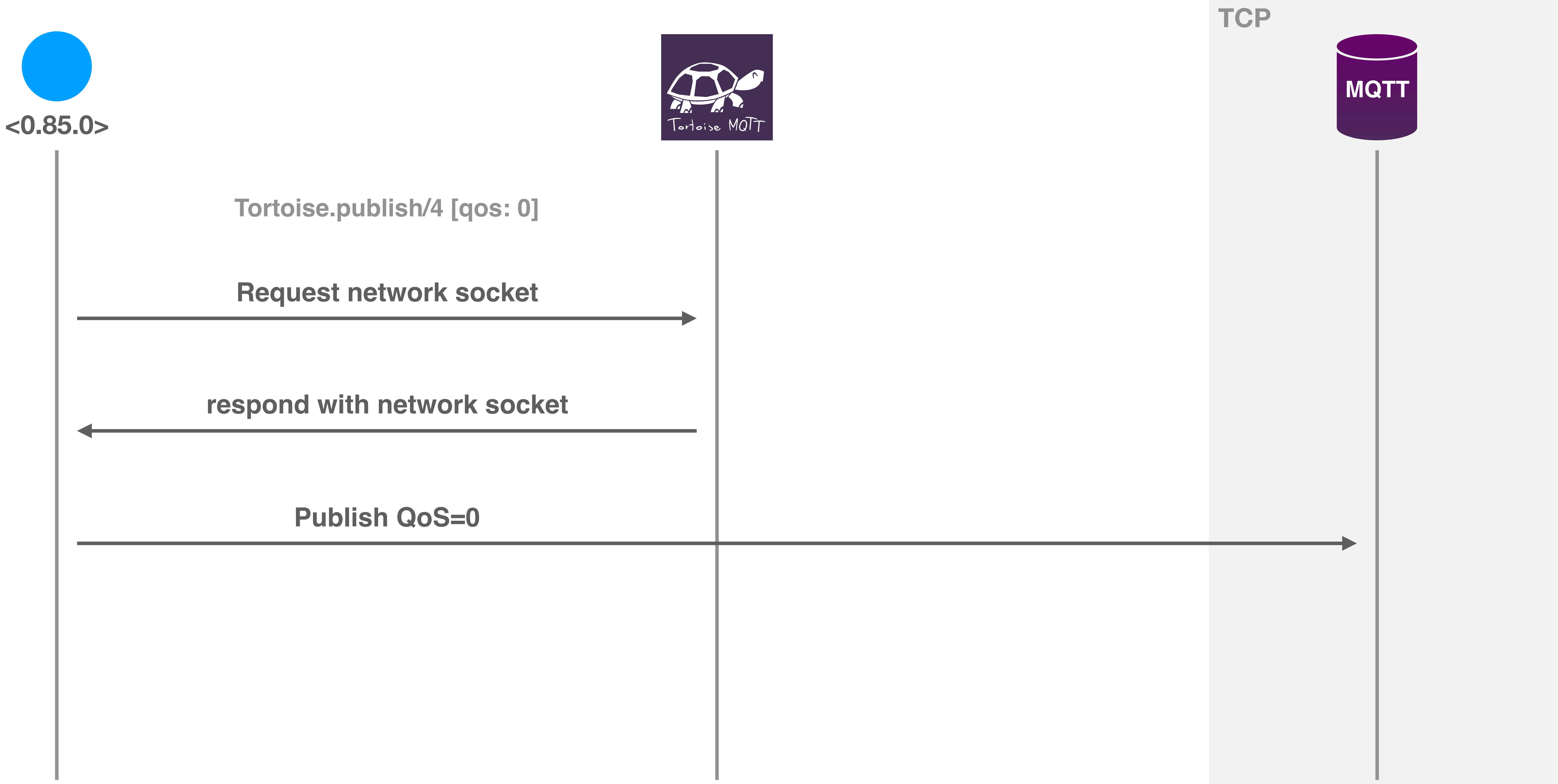


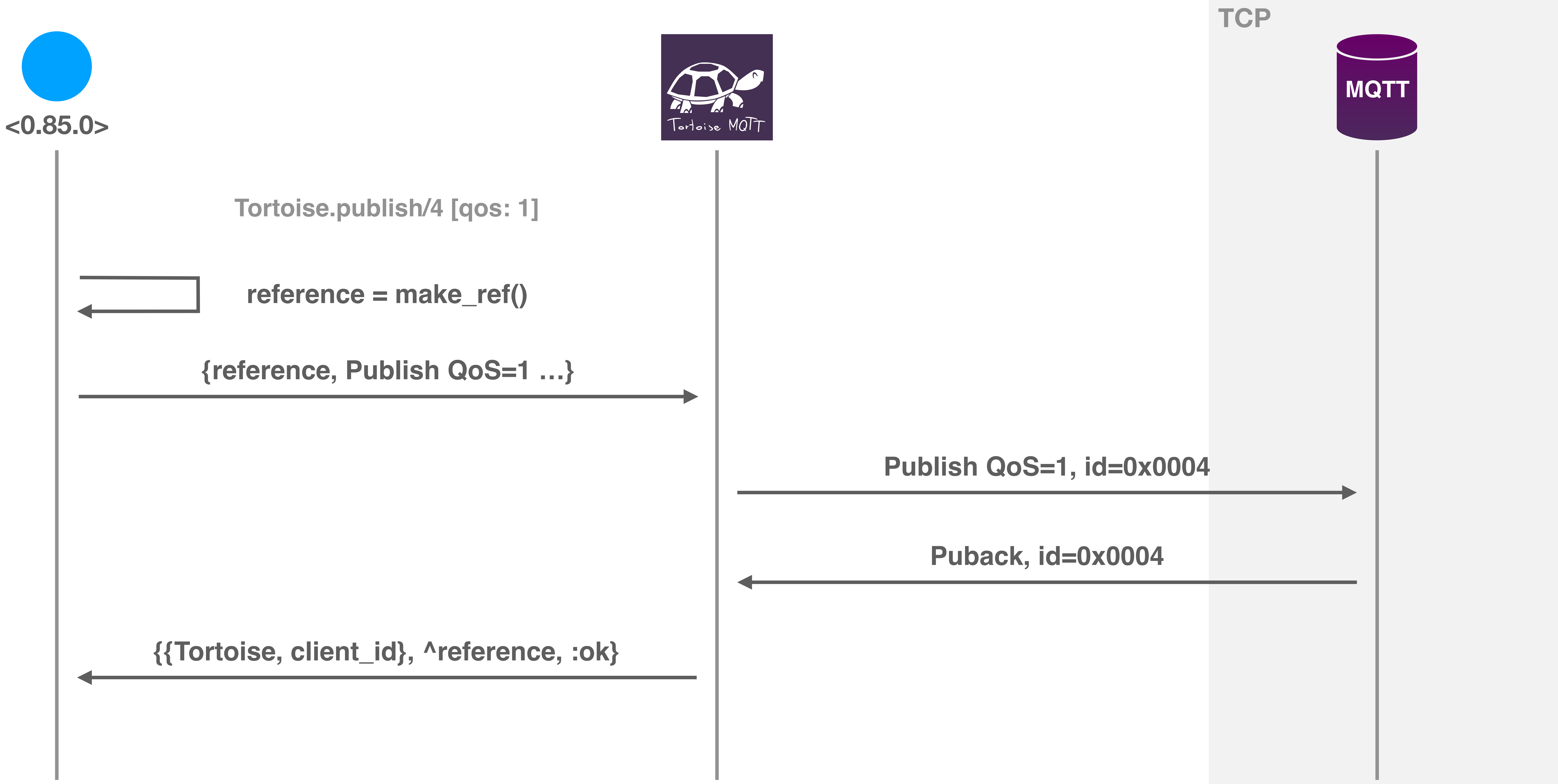
QoS=1; at least once delivery

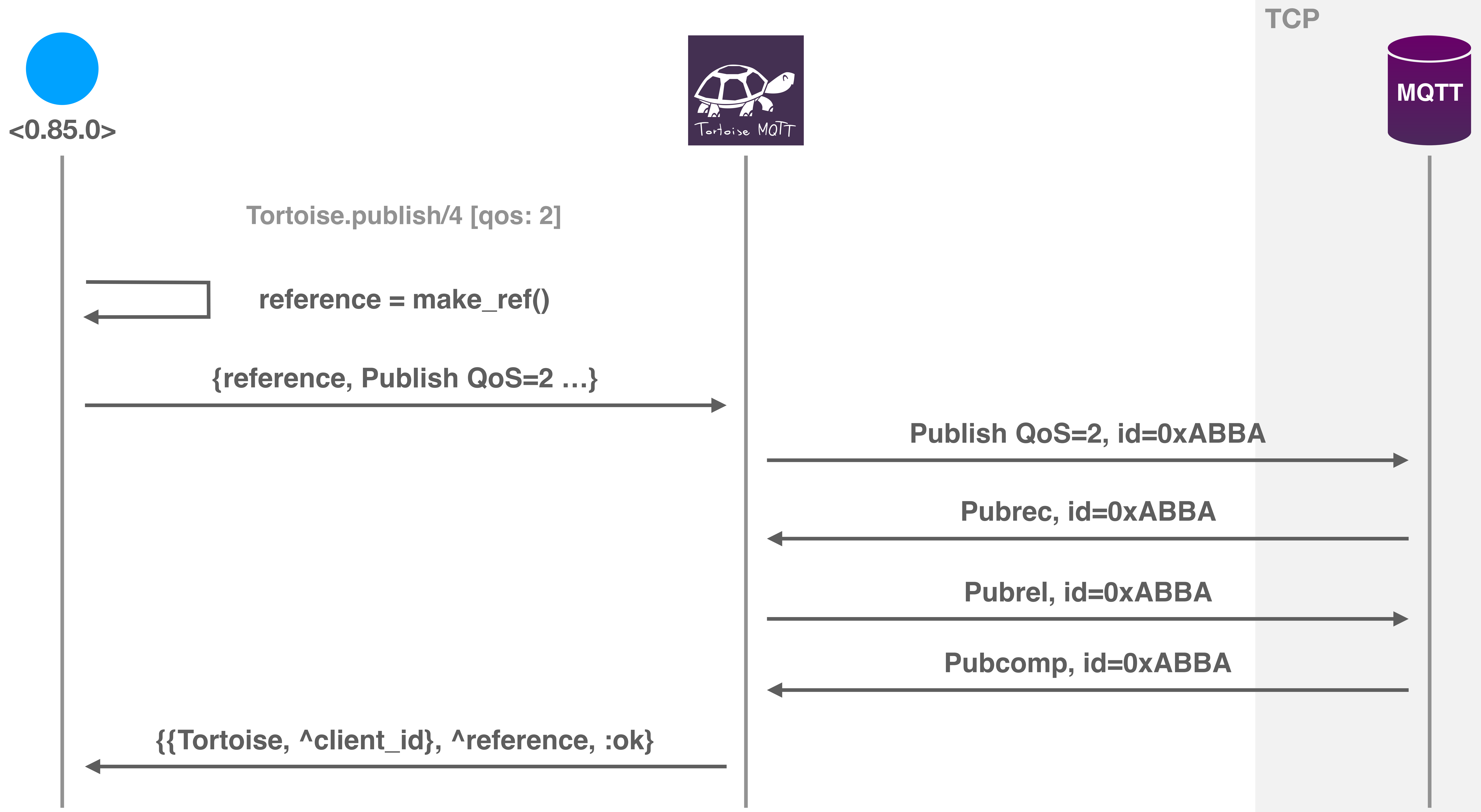


QoS=2; only once delivery

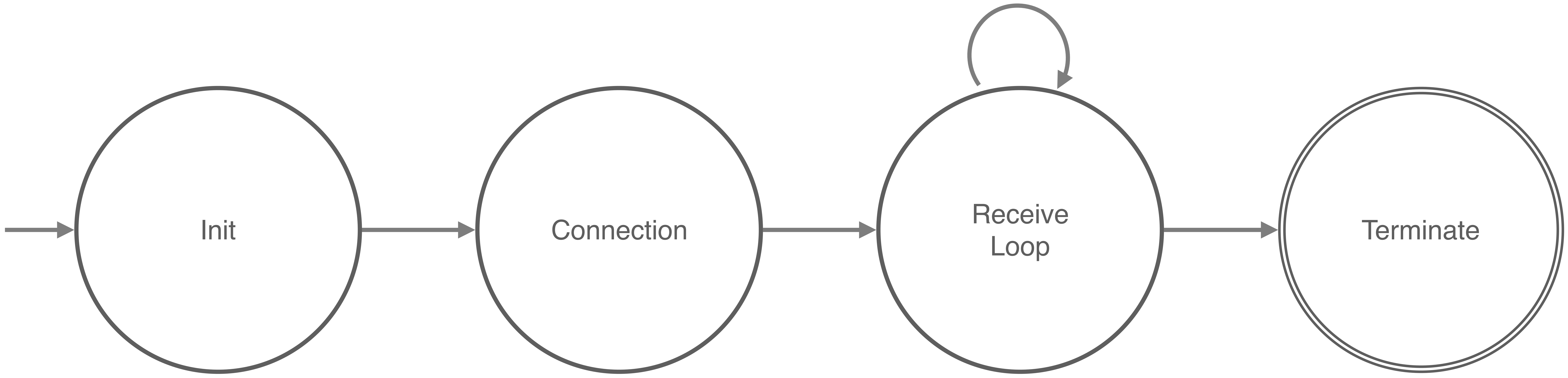



TCP
<0.85.0>
Tortoise MQTT
MQTT
Tortoise.publish/4 [qos: 0]
Request network socket
respond with network socket
Publish QoS=0

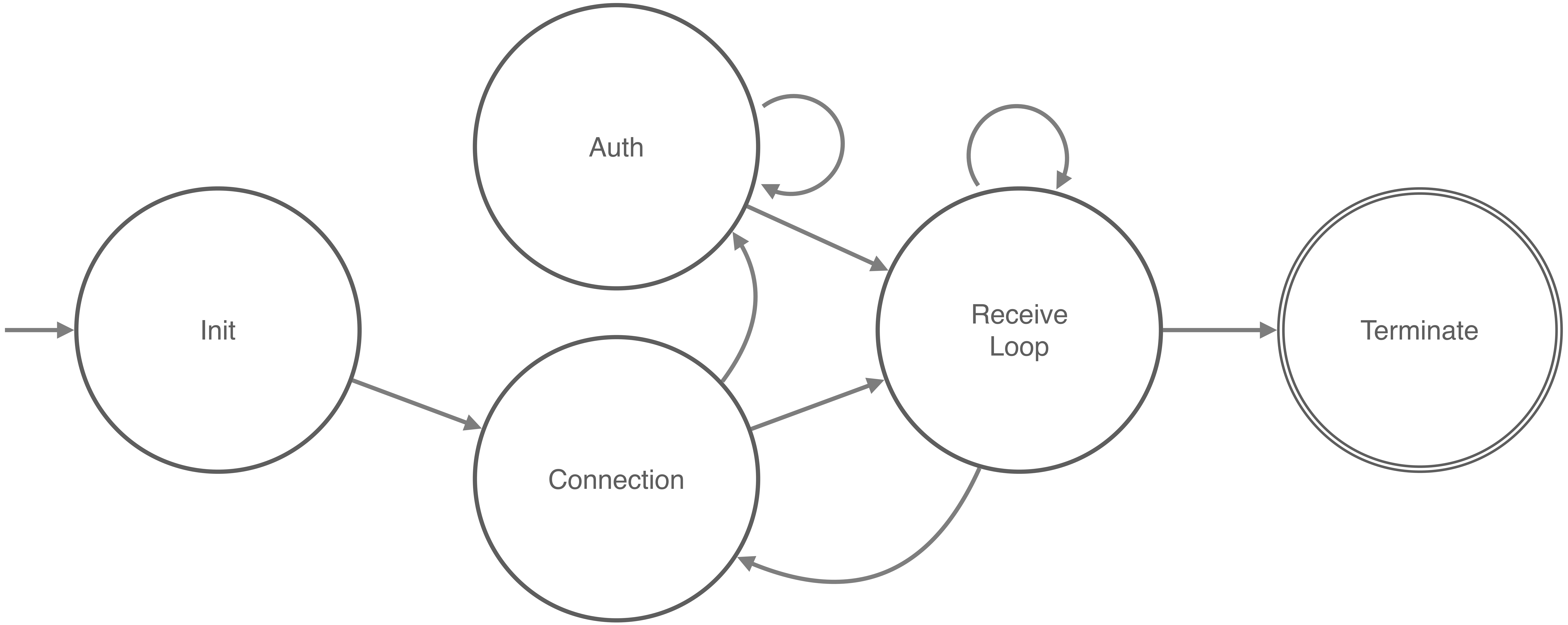

TCP
<0.85.0>
Tortoise MQTT
MQTT
Tortoise.publish/4 [qos: 1]
reference = make_ref()
{reference, Publish QoS=1 …}
Publish QoS=1, id=0x0004
Puback, id=0x0004
{{Tortoise, client_id}, ^reference, :ok}


TCP
<0.85.0>
Tortoise MQTT
MQTT
Tortoise.publish/4 [qos: 2]
reference = make_ref()
{reference, Publish QoS=2 …}
Publish QoS=2, id=0xABBA
Pubrec, id=0xABBA
Pubrel, id=0xABBA
Pubcomp, id=0xABBA
{{Tortoise, ^client_id}, ^reference, :ok}

Tortoise.Handler Life-cycle



Next Tortoise.Handler Life-cycle


Auth
Init
Receive Loop
Terminate
Connection