Using Messaging Protocols to Build Mobile and Web Applications

Jeff Mesnil

Jeff Mesnil

- Software Engineer at Red Hat
- Core developer on WildFly Application Server, lead for its messaging component
- Developed messaging brokers (HornetQ, JBoss Messaging, JORAM)
- http://jmesnil.net/
- jmesnil@gmail.com
- @jmesnil on Twitter

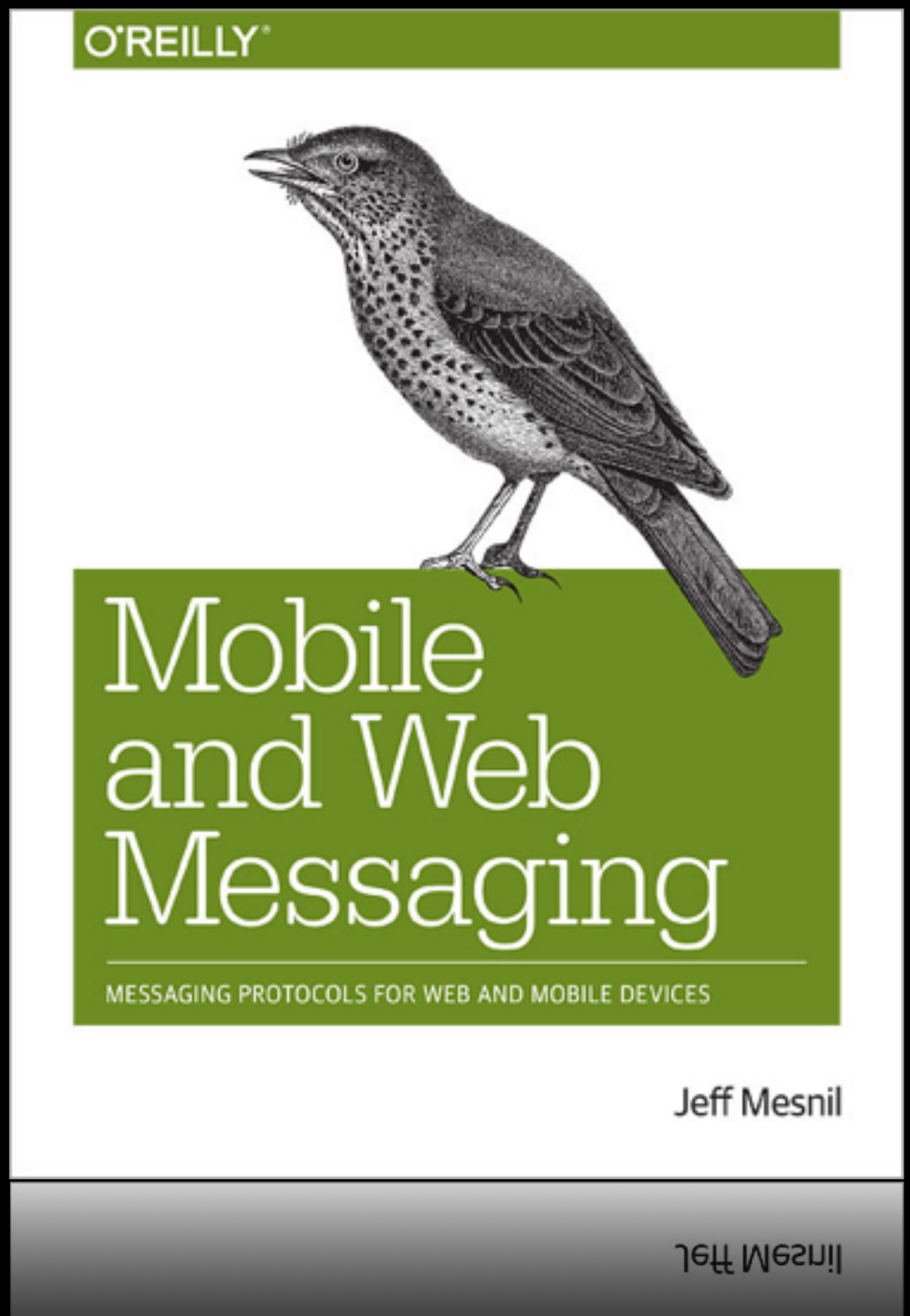
Mobile & Web Messaging

Messaging Protocols for Web and Mobile Devices

http://shop.oreilly.com/product/0636920032366.do

http://mobile-web-messaging.net/

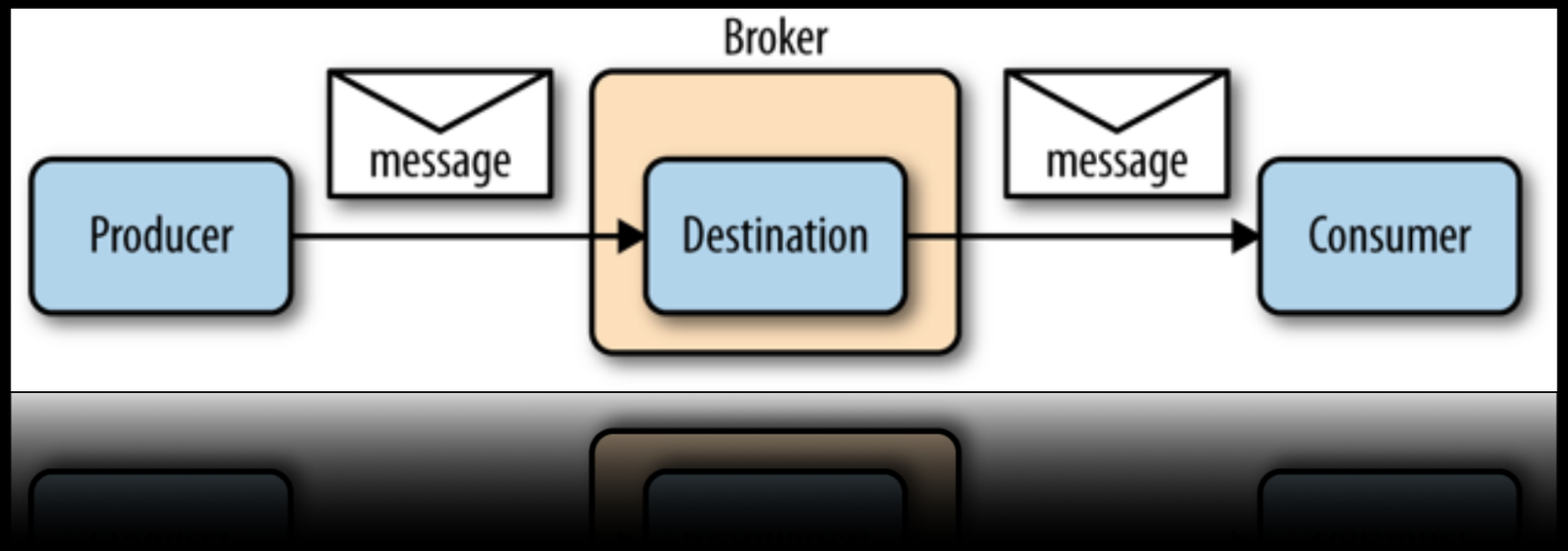
- Published by O'Reilly Media
- Released in August 2014



Summary

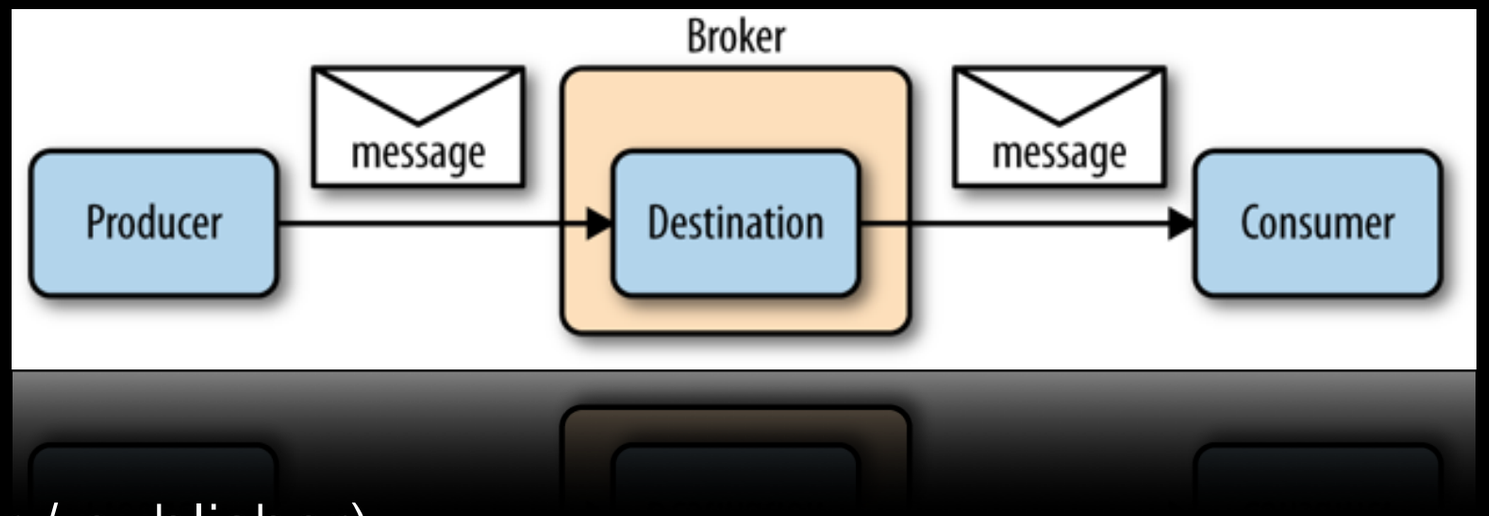
- Messaging Protocols
- Mobile & Web Devices
- STOMP
- MQTT
- Q & A

Messaging Concepts



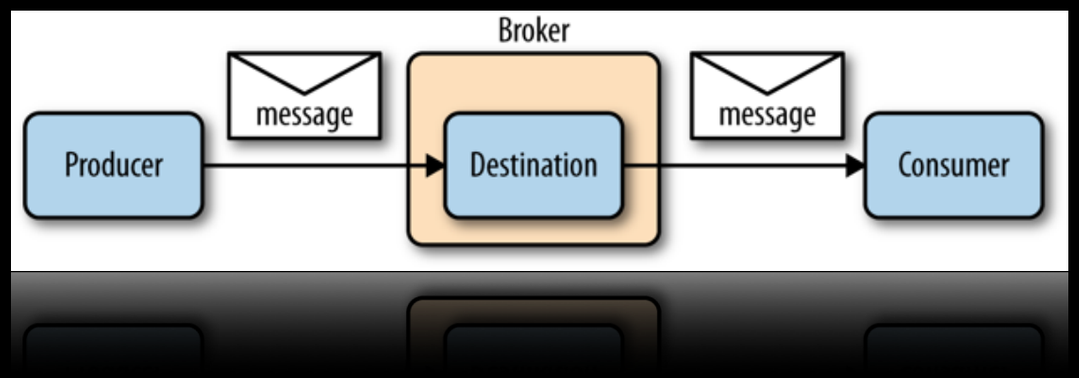
Messaging Concepts

- Message
- Destination
- Producer (sender / publisher)
- Consumer (receiver / subscriber)
- Broker

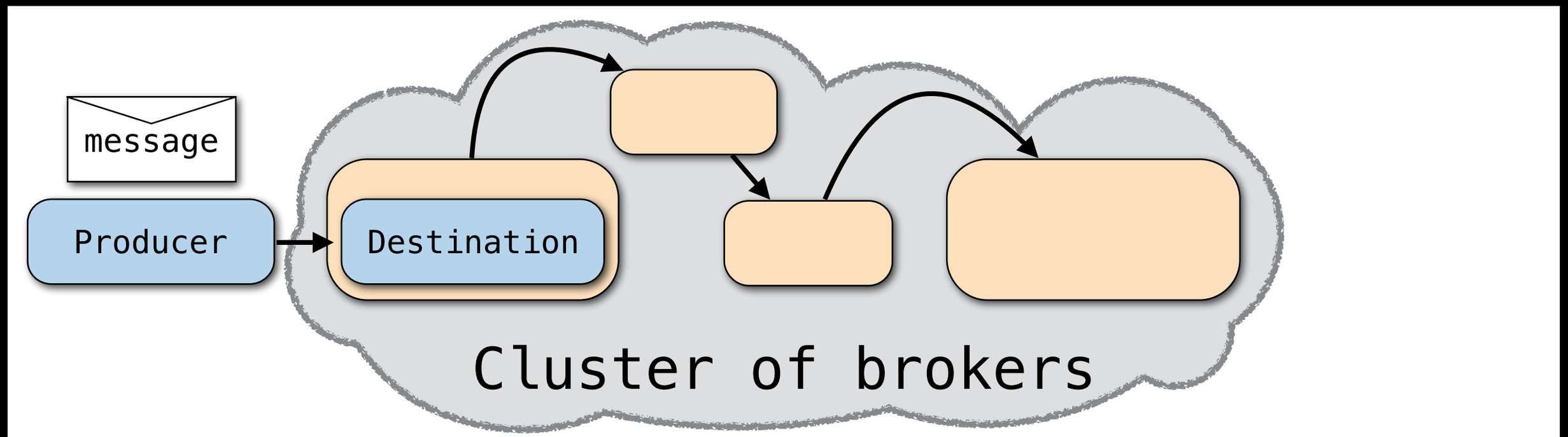


Messaging Concepts

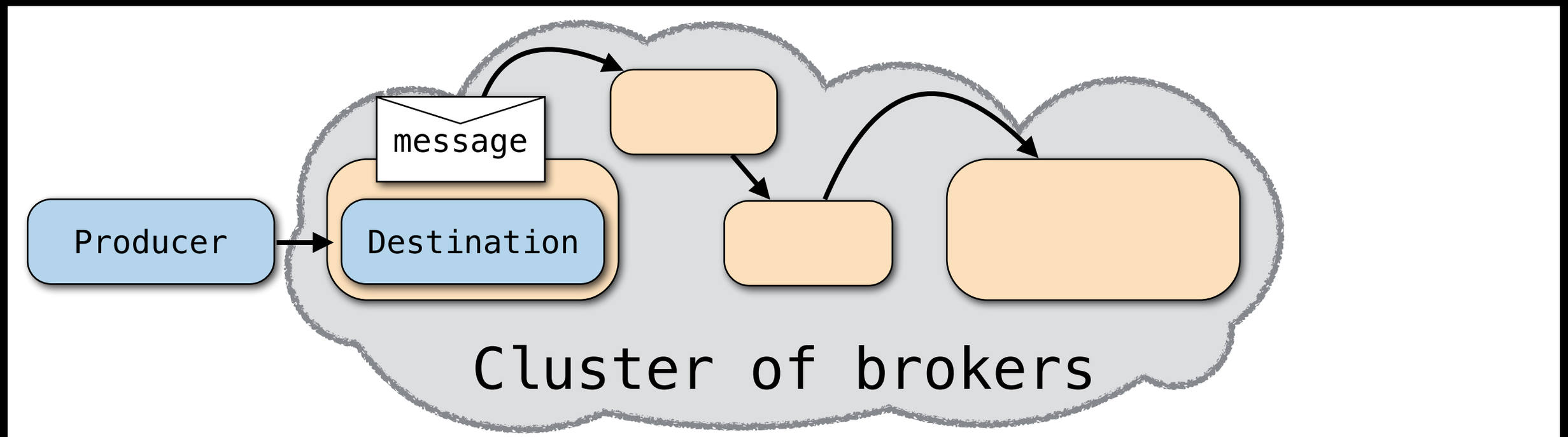
- Loosely coupled
  - Time independence
  - Location independence
- Strong Contract
  - Destination
  - Message



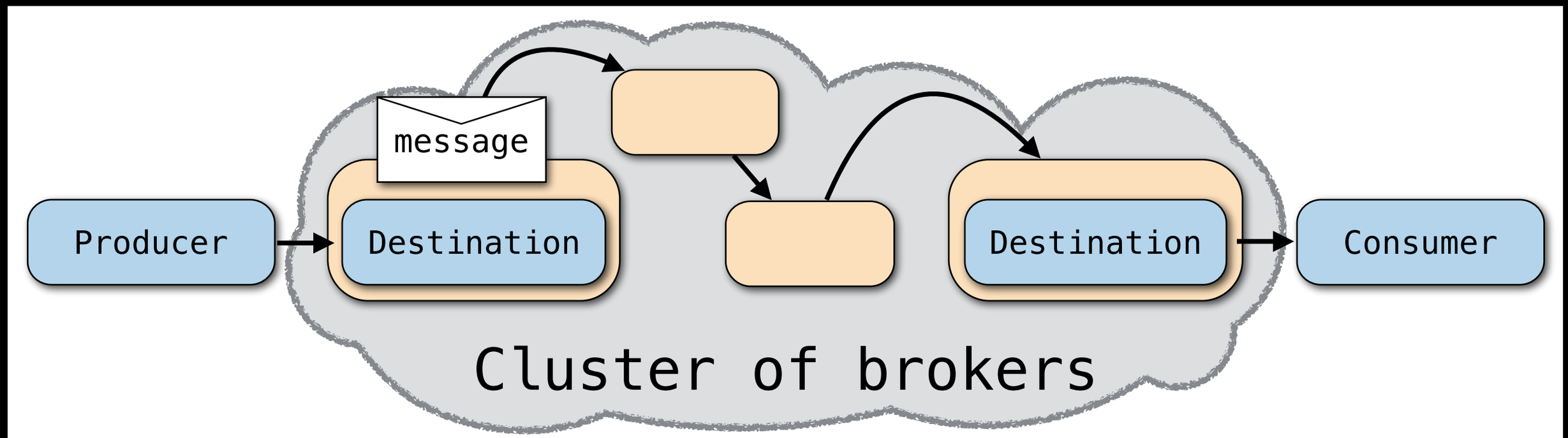
Messaging Concepts



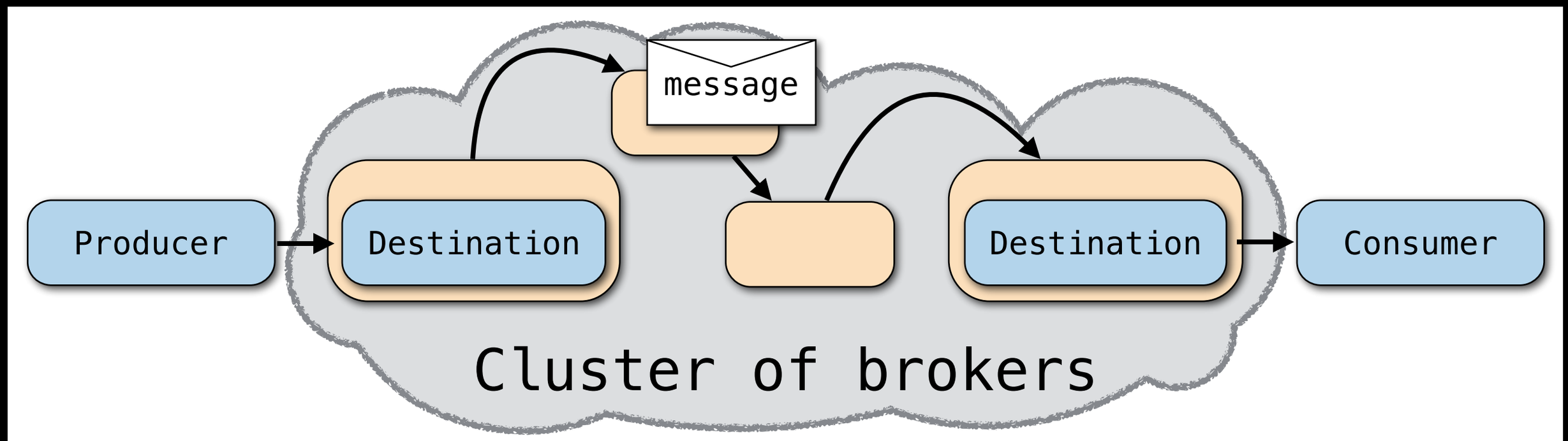
Messaging Concepts



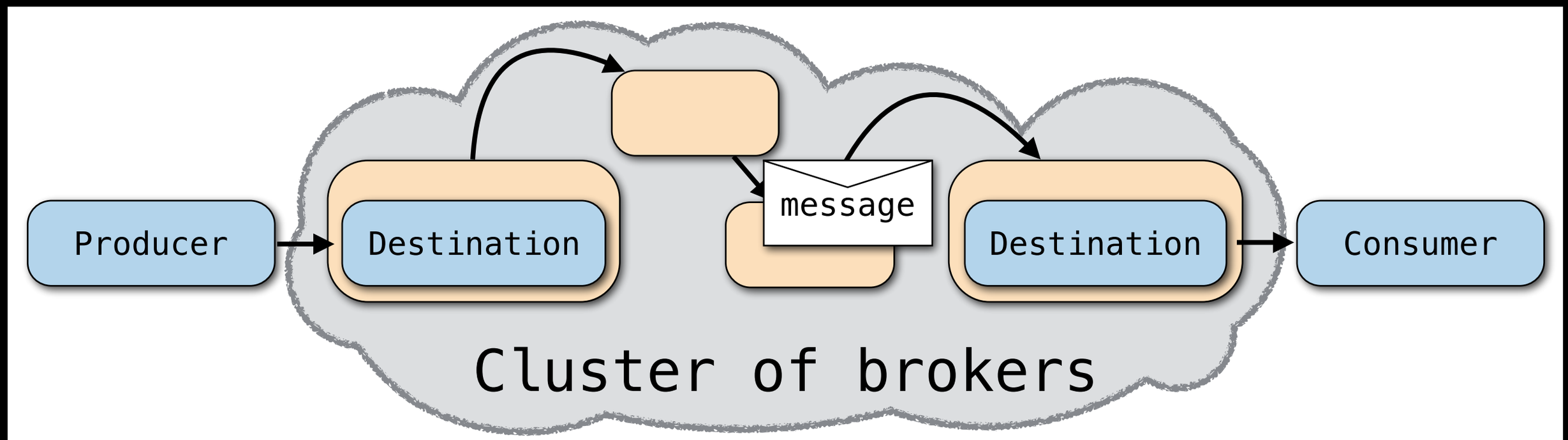
Messaging Concepts



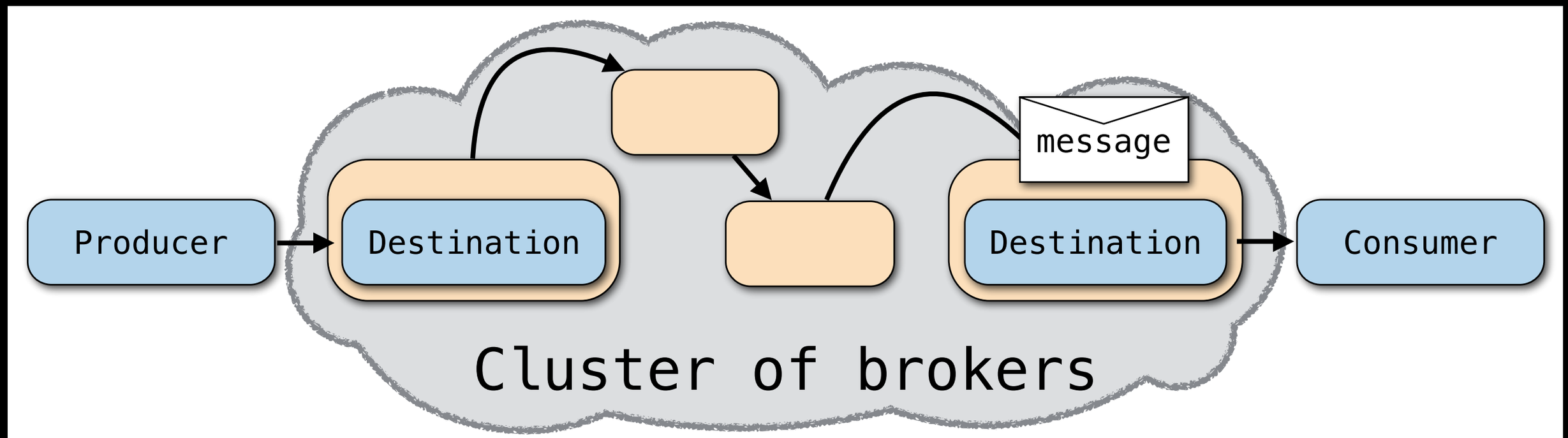
Messaging Concepts



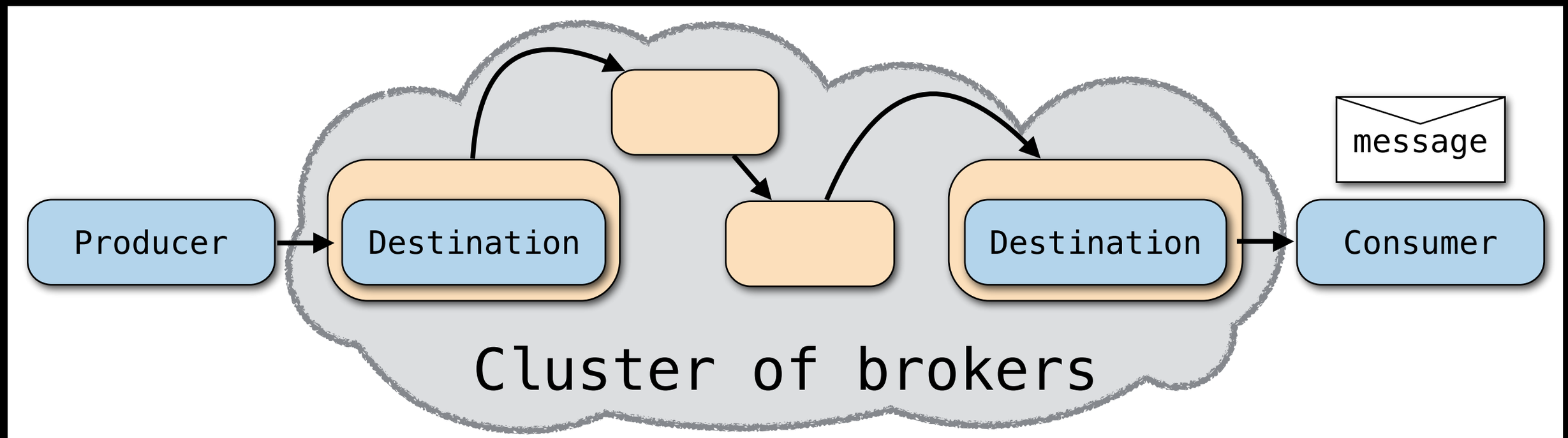
Messaging Concepts



Messaging Concepts



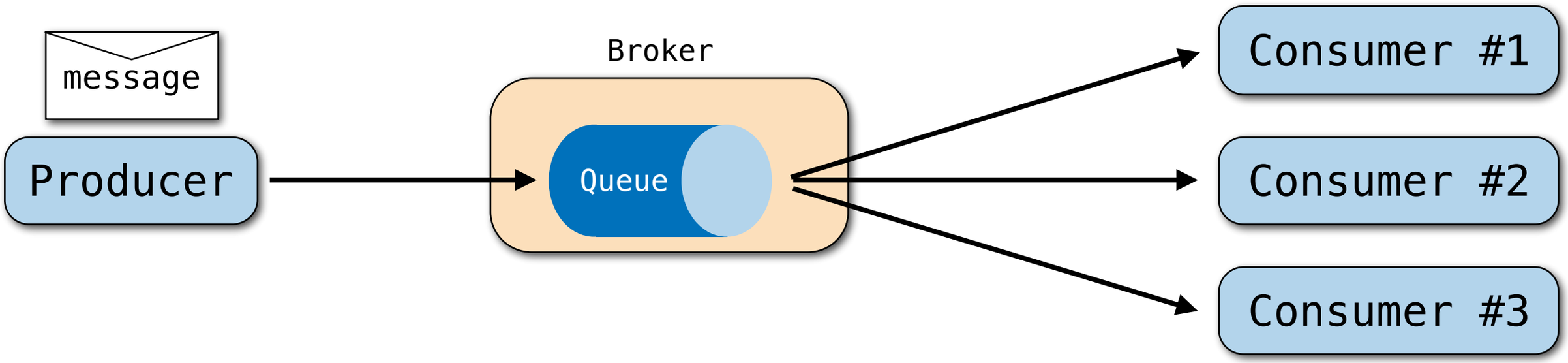
Messaging Concepts



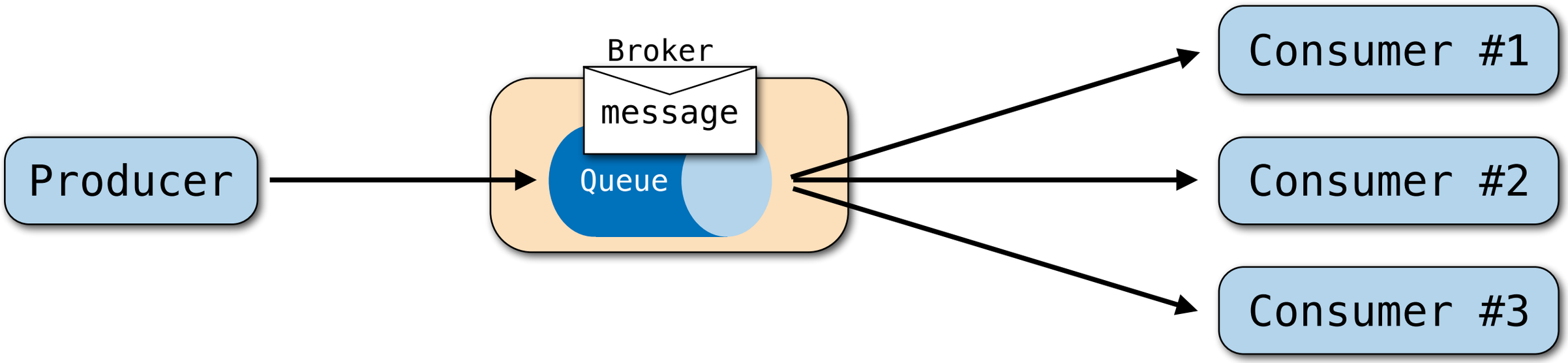
Messaging Models

- How messages will be routed from producers to consumers
- Point-to-point
- Publish/Subscribe
- Others

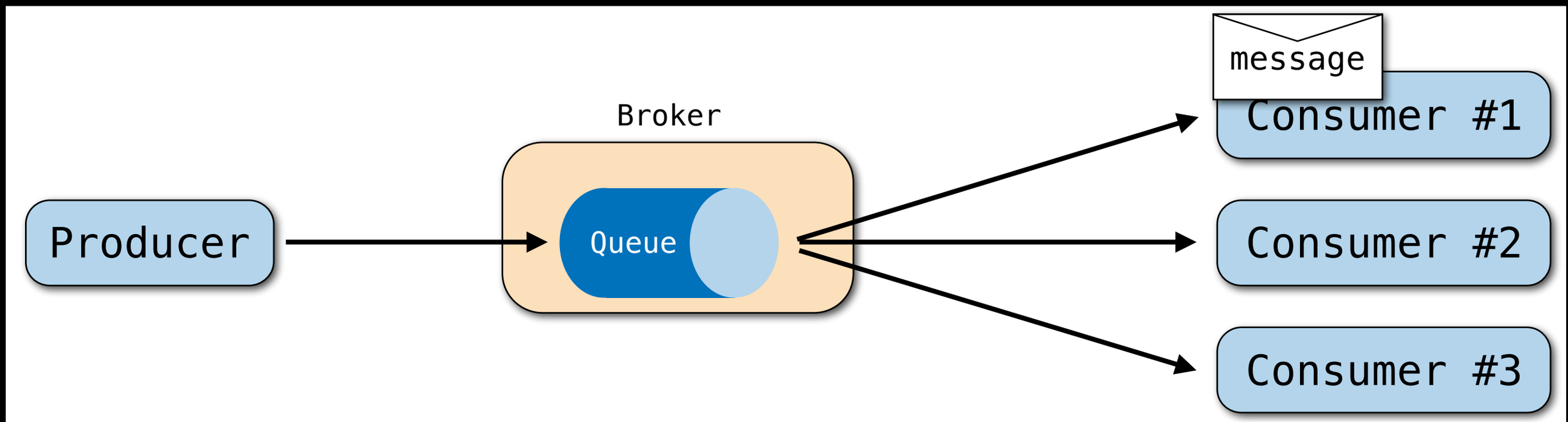
Point-to-Point Model



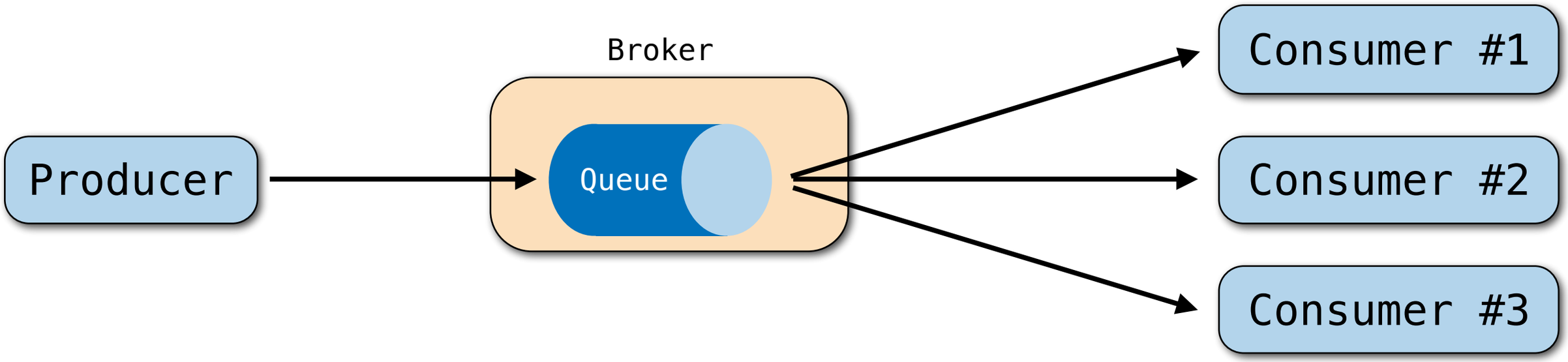
Point-to-Point Model



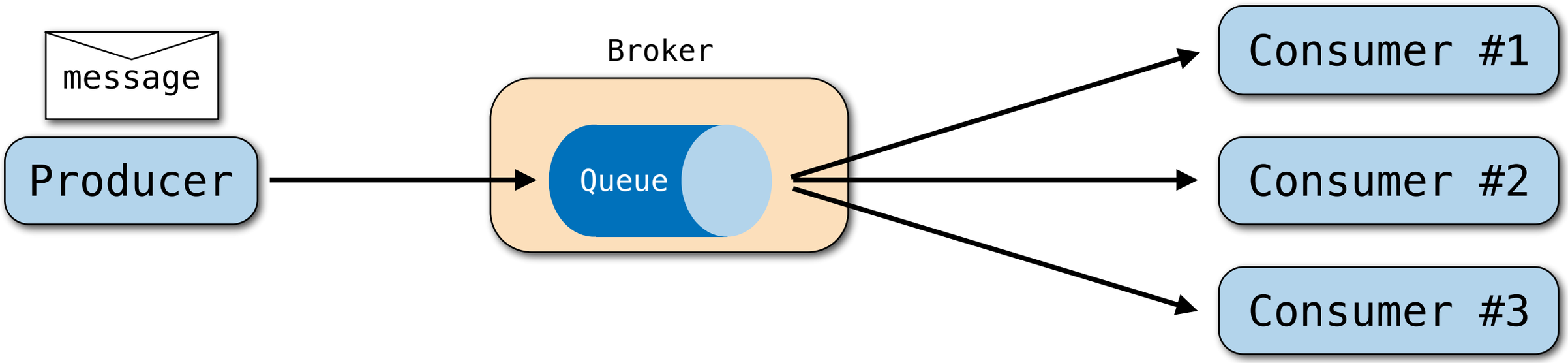
Point-to-Point Model



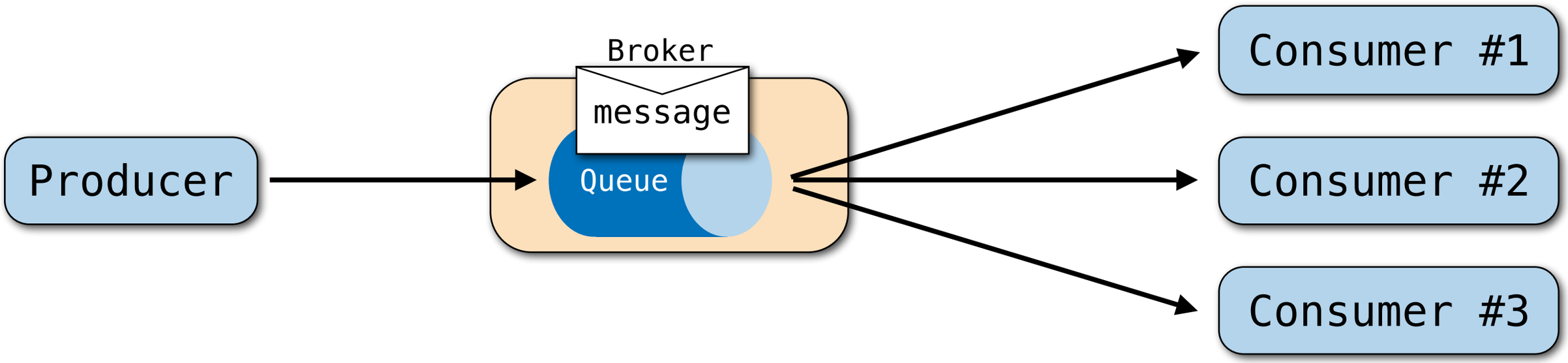
Point-to-Point Model



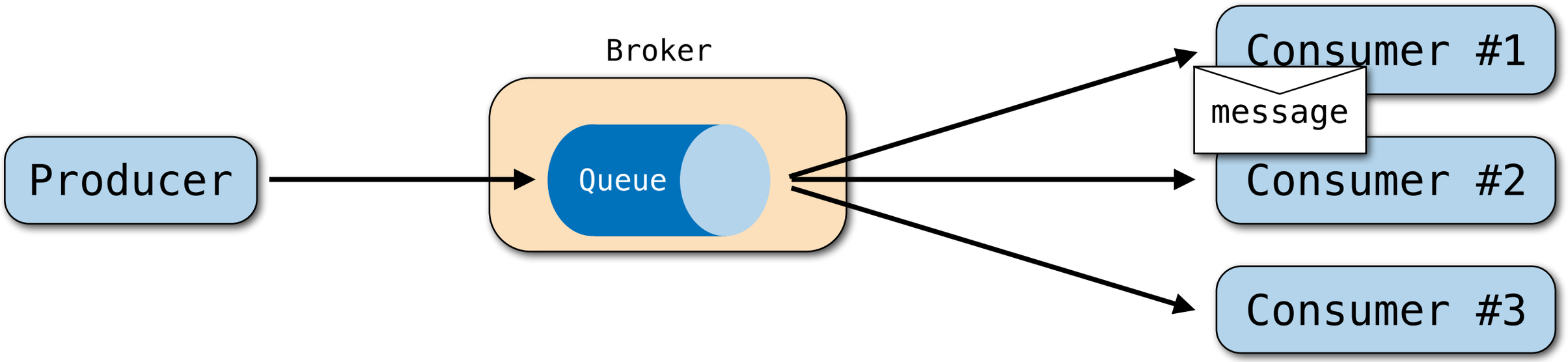
Point-to-Point Model



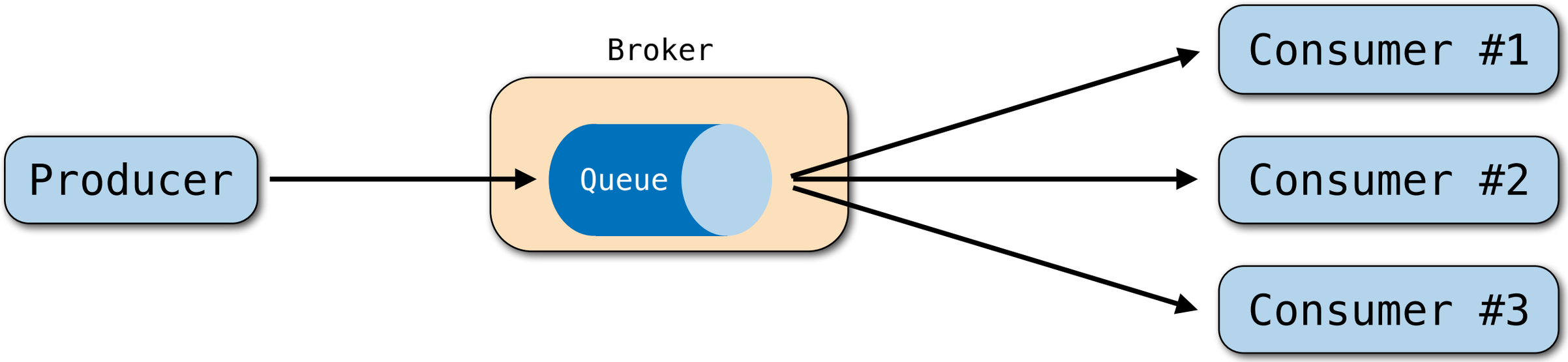
Point-to-Point Model



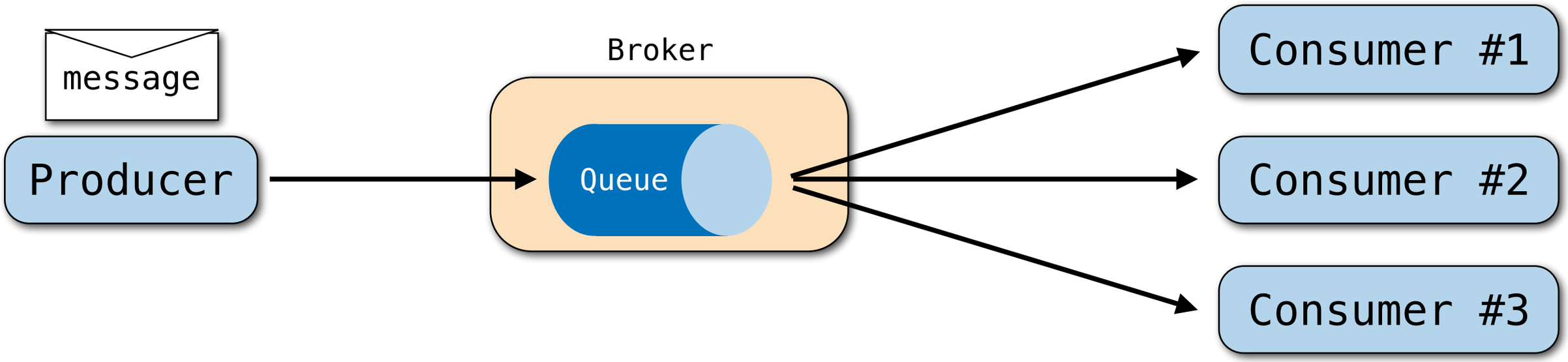
Point-to-Point Model



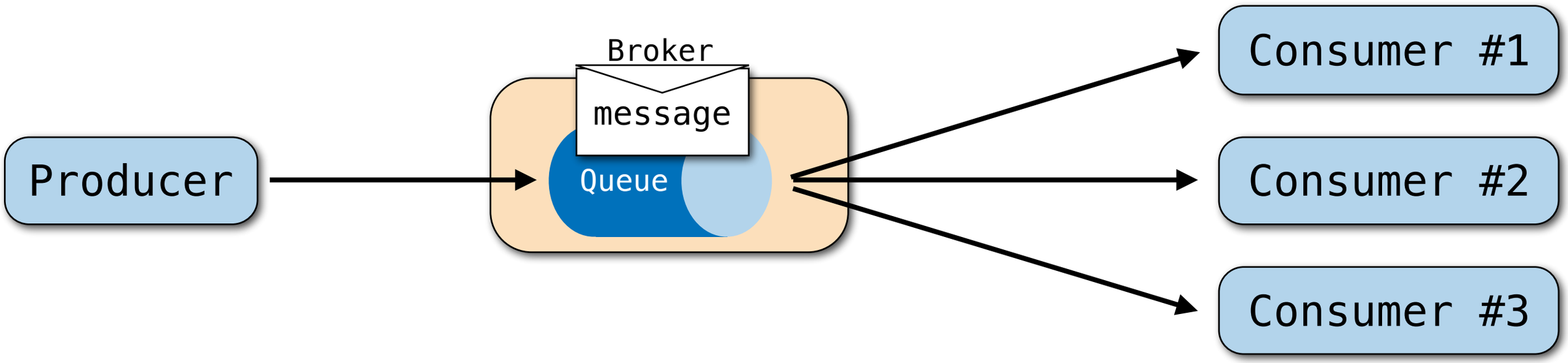
Point-to-Point Model



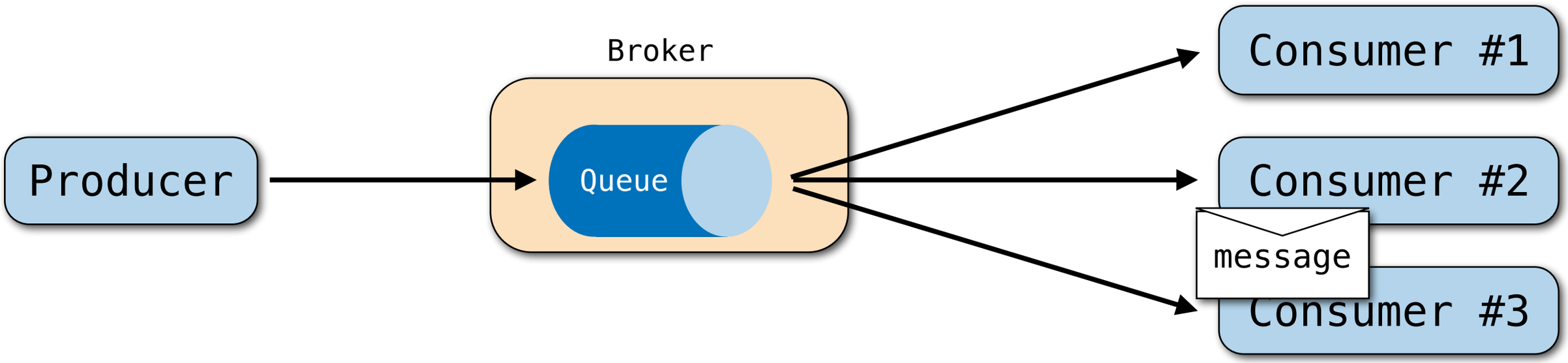
Point-to-Point Model



Point-to-Point Model



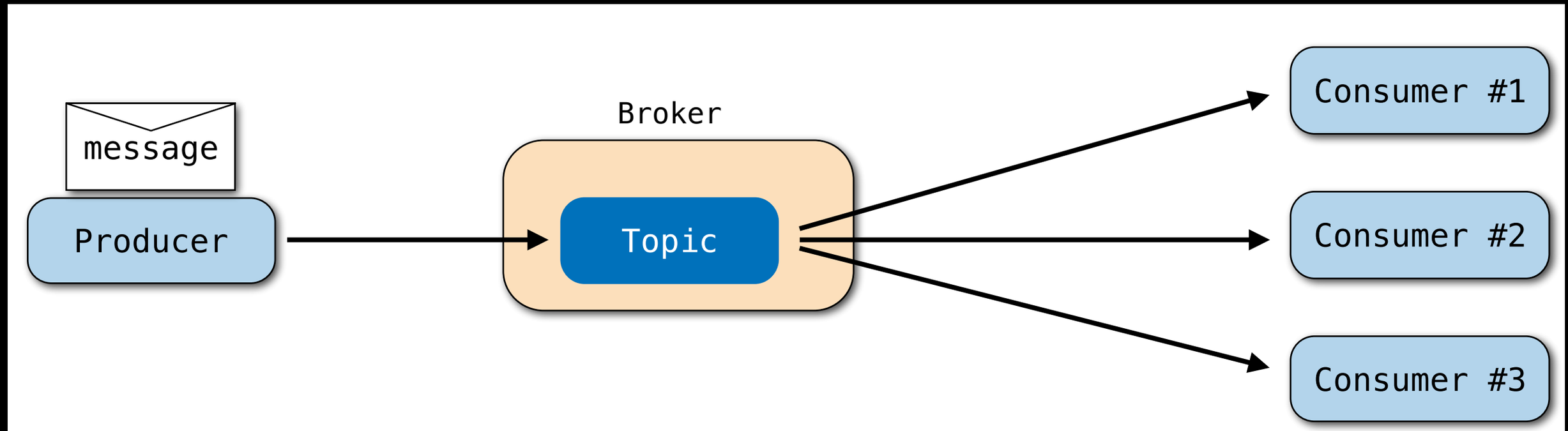
Point-to-Point Model



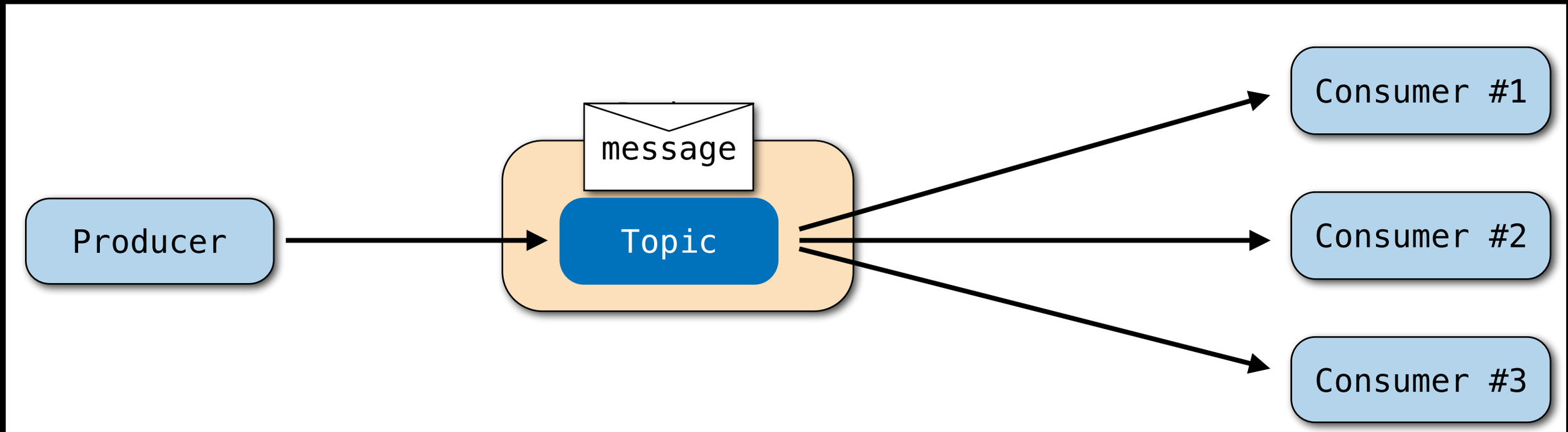
Point-to-Point Model

- Queues
- One-to-one model
  - One message produced
  - One message consumed

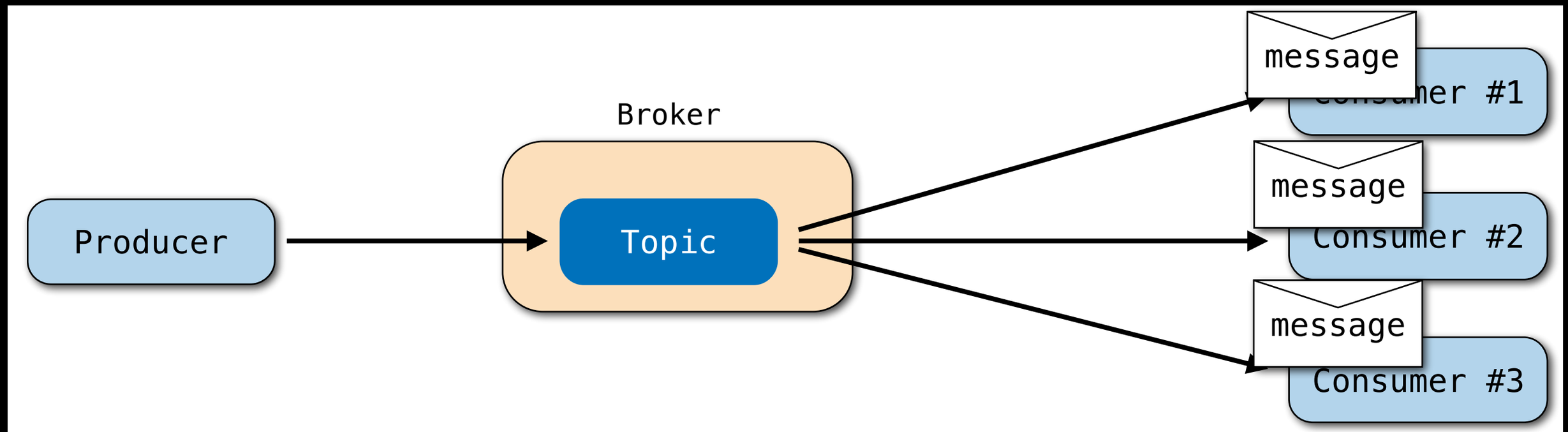
Publish/Subscribe Model



Publish/Subscribe Model



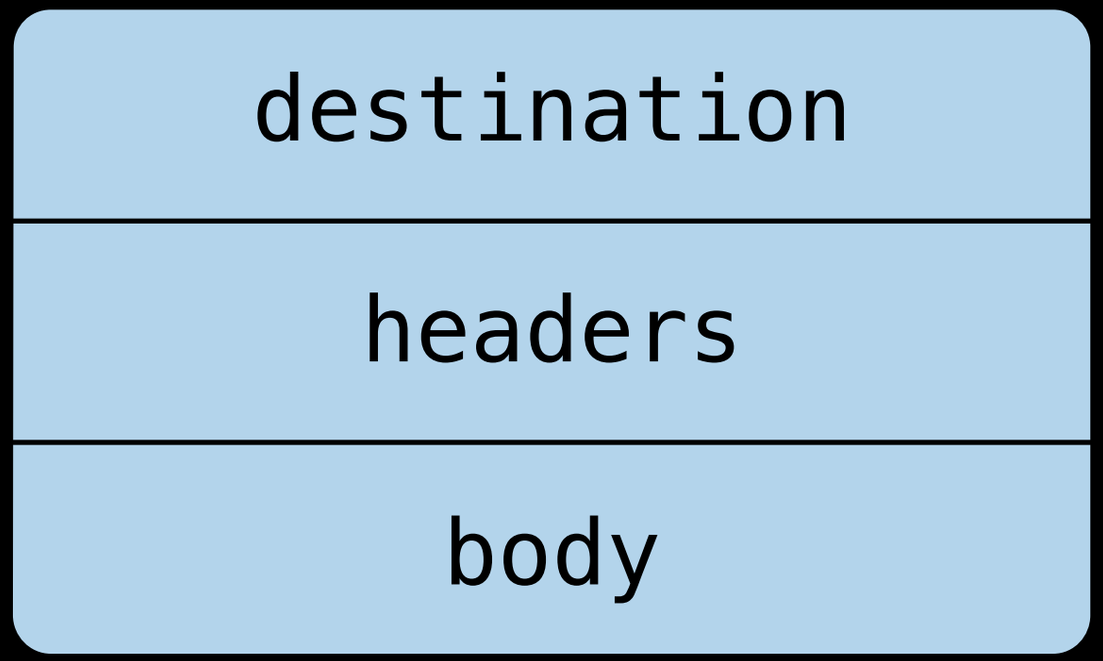
Publish/Subscribe Model



Publish/Subscribe Model

- Topics
- One-to-many model
  - One message produced
  - Many messages consumed
- Durable subscription

Message

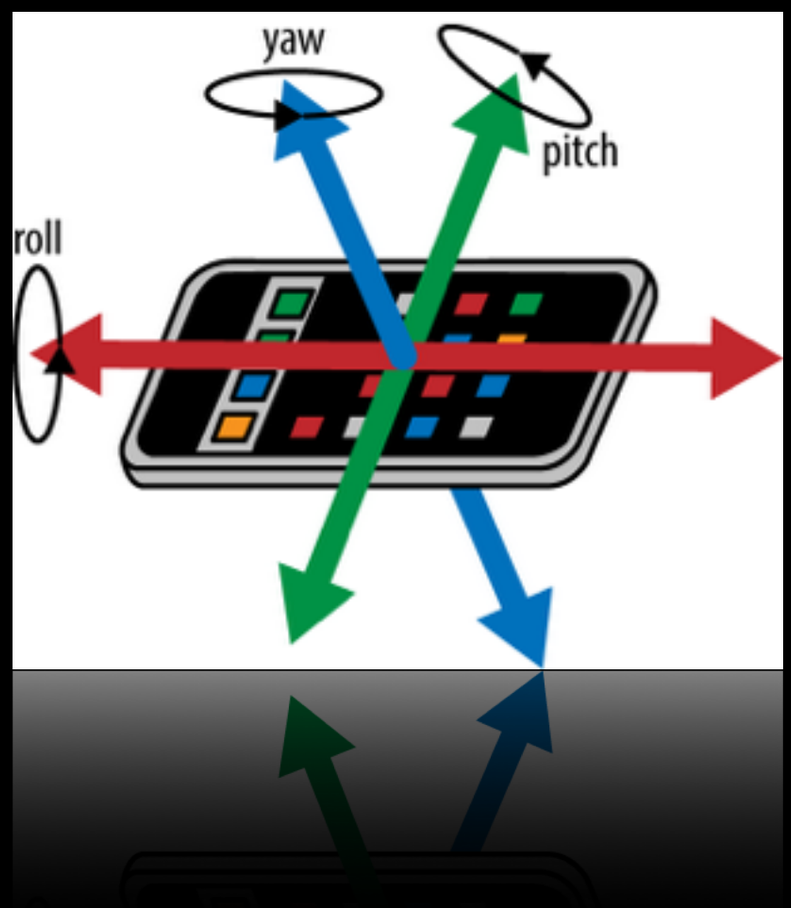


Messaging Protocols

- Entreprise
  - AMQP, JMS (Java), MSMQ
- First-mile
  - STOMP, MQTT

Mobile Devices

- Many sensors (GPS, motion, health)
- Always On
- Always with you



Mobile Devices

- Battery usage
- Intermittent network connection
- Network bandwidth
- Available memory

Messaging for Mobile Devices

- Small network footprint
- Small memory usage
- Deal with network failures

Web Browsers

- “Old” Web browser communication model
  - Browser initiates the request, server replies
    - No “right” way to push data from server to browser
      - HTTP long polling or streaming (RFC 6202)
      - Frequent HTTP requests

Web Browsers

- HTML5 Web Sockets
  - TCP socket for the Web
  - Protocol + JavaScript API
- Enabler for messaging protocols and pushing data from the server to the browser

Examples

- Locations application
    - GPS data, iOS application, Web application
- Motions application
    - Motion data, iOS application, Web application

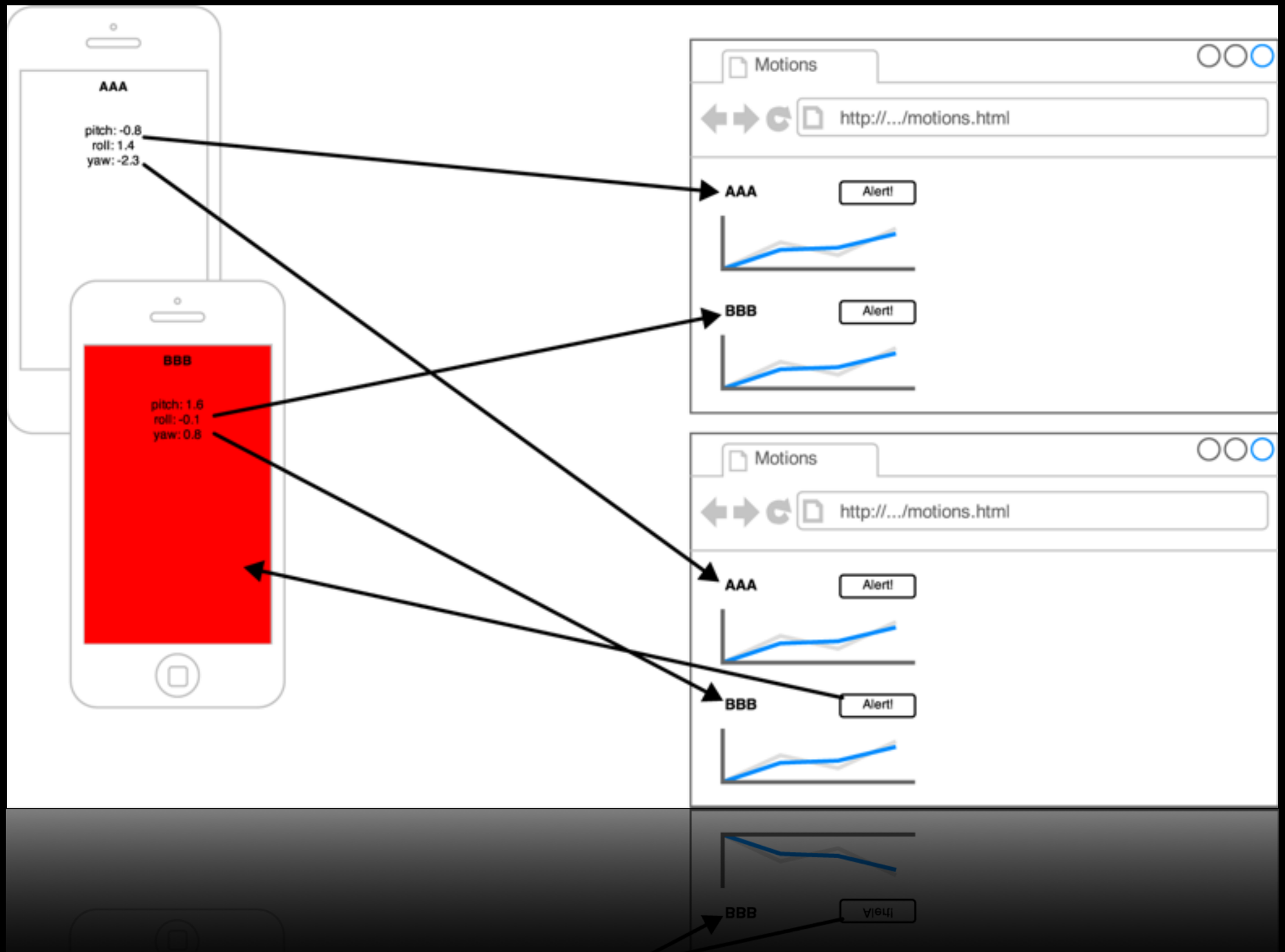
Locations


AAA
tracking position
Texts
text #1
text #2
text #3
Where are you heading to?
BBB
tracking position
Texts
text #1
text #2
Locations
http://.../locations.html
AAA
BBB
AAA
Where are you heading to?
Locations
http://.../locations.html
AAA
BBB
AAA
Where are you heading to?

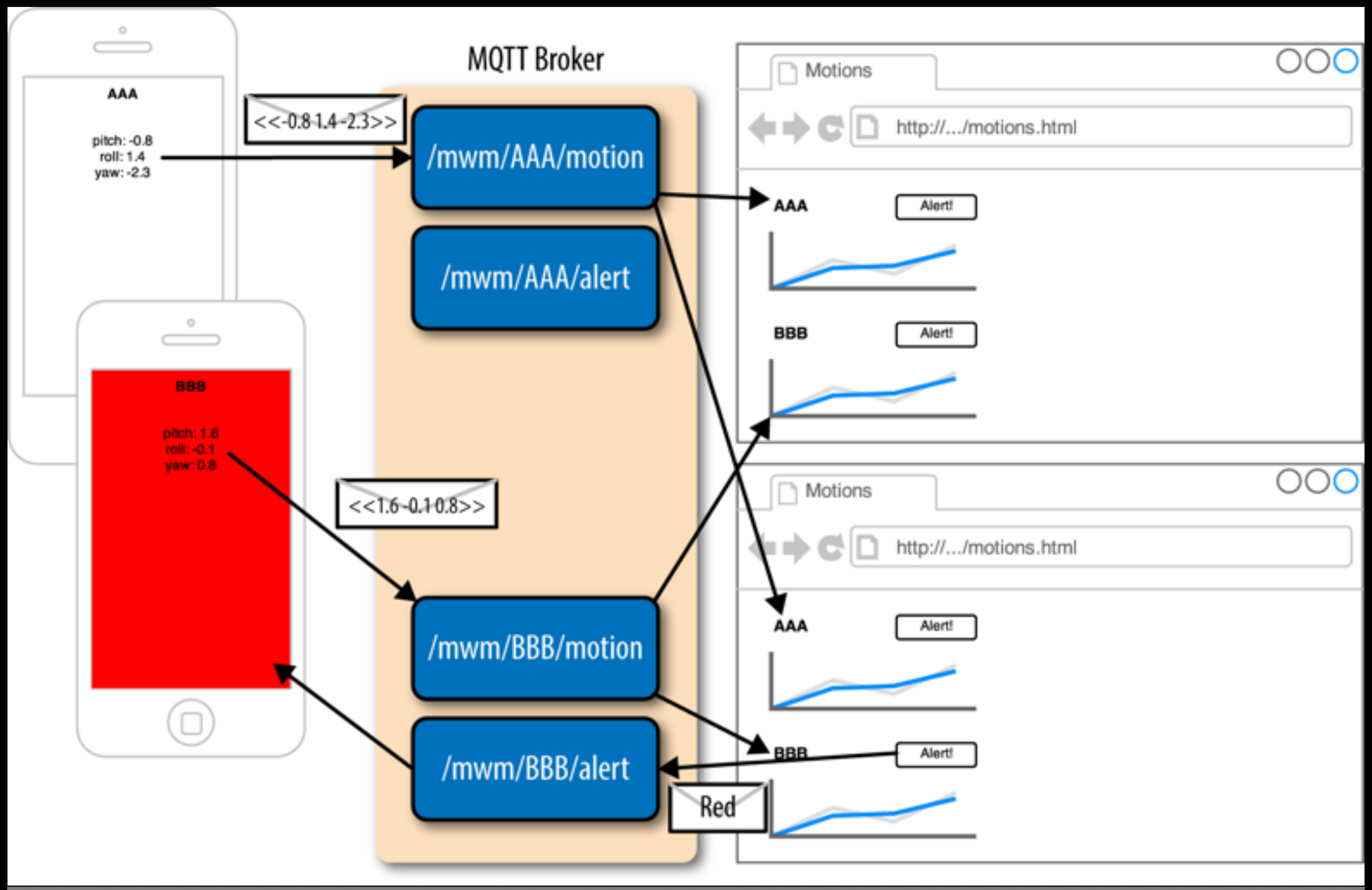
Locations


STOMP Broker
device.AAA.location
device.AAA.text
"Where are you heading to?"
{
"deviceID";"BBB",
"lat"; 46.8581,
"lng"; 2.2946,
"ls";"2012-09-23T08:432"
}
device.BBB.location
device.BBB.text
AAA
tracking position
Texts
text #1
text #2
text #3
Where are you heading to?
BBB
tracking position
Texts
text #1
text #2
Locations
http://.../locations.html
AAA
BBB
AAA
Where are you heading to?
Locations
http://.../locations.html
AAA
BBB
BBB

Motions


AAA
pitch: -0.8
roll: 1.4
yaw: -2.3
BBB
pitch: 1.6
roll: -0.1
yaw: 0.8
Motions
http://.../motions.html
AAA
Alert!
BBB
Alert!
Motions
http://.../motions.html
AAA
Alert!
BBB
Alert!
BBB
Alert!

Motions


AAA
pitch: -0.8
roll: 1.4
yaw: -2.3
BBB
pitch: 1.6
roll: -0.1
yaw: 0.8
MQTT Broker
<<-0.8 1.4 -2.3>>
/mwm/AAA/motion
/mwm/AAA/alert
<<1.6 -0.1 0.8>>
/mwm/BBB/motion
/mwm/BBB/alert
Red
Motions
http://.../motions.html
AAA
Alert!
BBB
Alert!
Motions
http://.../motions.html
AAA
Alert!
BBB
Alert!

STOMP

- Simple Text Oriented Messaging Protocol
- http://stomp.github.io
- Open Source brokers (Apache ActiveMQ, JBoss HornetQ, RabbitMQ,…)
- Many client implementations

STOMP

- Text-based
- Inspired by HTTP
- Messaging model is not specified

STOMP frame

```
COMMAND
header1:value1
header2:value2

body^@
```

STOMP SEND frame

```
SEND
destination:/topic/device.XXX.location
content-type:application/json;charset=utf-8
content-length:83

{“lng”:-122.032,”lat”:
37.335,”ts”:”2014-03-13T17:19:05+01:00”,”de
viceID”:”XXX”}
```

STOMP producer

- `CONNECT` to a broker (+ receives `CONNECTED`)
- `SEND` message to a destination
- `DISCONNECT` from the broker

STOMP consumer

- `CONNECT` to a broker (+ receives `CONNECTED`)
- `SUBSCRIBE` to a destination
- receive `MESSAGE`s
- `UNSUBSCRIBE` from the destination
- `DISCONNECT` from the broker

Authentication

- `login/passcode` in `CONNECT`
- => `CONNECTED` if successful
- => `ERROR` (+ socket closed) else

Heart-Beating

- `heartbeat` negotiation in `CONNECT`/`CONNECTED`
  - client-to-broker
  - broker-to-client
- Connection is considered dead after failing to receive heart-beat during 2x the negotiated value

Message Acknowledgement

- `ack` in `SUBSCRIBE`
  - `auto*, client, client-individual`
- `ACK/NACK` frames for client acknowledgement

Transactions

- `BEGIN` / `COMMIT` / `ABORT` + `transaction` for transaction boundaries
- `transaction` in `SEND` / `ACK` frames to identify the transaction

Receipt

- Confirmation that broker has received frames
- `receipt` in any frame
- => `RECEIPT` + `receipt-id` sent by the broker upon reception

Error Handling

- **`ERROR`** sent by the broker (+ connection closed)
- `message` contains the description of the error
- Body may contain more information
- No standard status code

STOMP extensions

- Persistence: `SEND` with `persistent:true`
- Filtered consumer: `SUBSCRIBE` with `selector: "country in (‘FR’, ‘DE’, ‘IT’)"`
- Priority: `SEND` with `priority:7`
- Expiration: `SEND` with `expires:1415631438`

StompKit

- Objective-C library (iOS, OS X)
- Event-driven (Grand Central Dispatch + Blocks)
- Open Source, Apache License 2.0
- github.com/mobile-web-messaging/StompKit/

StompKit producer

```
// create the client
STOMPClient *client = [[STOMPClient alloc] initWithHost:@“192.168.1.25”
                                                   port:61613];
// connect to the broker
[client connectWithLogin:@"mylogin"
                passcode:@"mypassword"
       completionHandler:^(STOMPFrame *_, NSError *error) {
           if (err) {
               NSLog(@"%@", error);
               return;
           }

           // send a message
           [client sendTo:@"/queue/myqueue" body:@"Hello, iOS!"];
           // and disconnect
           [client disconnect];
       }];
```

StompKit consumer

```
// create the client
STOMPClient *client = [[STOMPClient alloc] initWithHost:@“192.168.1.25”
                                                   port:61613];
// connect to the broker
[client connectWithLogin:@"mylogin"
                passcode:@"mypassword"
       completionHandler:^(STOMPFrame *_, NSError *error) {
            if (err) {
                NSLog(@"%@", error);
                return;
            }

            // subscribe to the destination
            [client subscribeTo:@"/queue/myqueue"
                        headers:@{}
                 messageHandler:^(STOMPMessage *message) {
                    // called back when the client receive a message
                    // for the /queue/myqueue destination
                    NSLog(@"got message %@", message.body);
                    // => "Hello, iOS"
                }];

            }];
```

stomp.js

- JavaScript library for Web browsers & node.js
- Open Source, Apache License 2.0
- github.com/jmesnil/stomp-websocket

stomp.js producer

```
var url = "ws://localhost:61614/stomp";
var client = Stomp.client(url);

client.connect("mylogin", "mypasscode", function(frame) {
  // upon successful connection
  var data = { “deviceID”: “XYZ” } // JSON object

  client.send("/queue/myqueue",
              {}, // no headers
              JSON.stringify(data));

  client.disconnect();
});
```

stomp.js consumer

```
var url = "ws://localhost:61614/stomp";
var client = Stomp.client(url);

client.connect("mylogin", "mypasscode", function(frame) {
  // upon successful connection
  client.subscribe("/queue/myqueue", function(message) {
    // called back when the client receive a message
    // for the /queue/myqueue destination
    var data = JSON.parse(message.body);
    // => { “deviceID”: “XYZ” }
  });

});
```

MQTT

- light-weight publish/subscribe messaging protocol
- http://mqtt.org
- Open Source brokers (Apache ActiveMQ, RabbitMQ, Mosquitto)
- Many client implementations (Eclipse Paho)

MQTT

- Binary protocol
- Publish/subscribe only (no point-to-point)
- Internet of Things
- Public broker at iot.eclipse.org

MQTT Control Packets

- CONNECT / CONNACK / DISCONNECT
- PUBLISH (+ PUBACK / PUBREC / PUBREL / PUBCOMP)
- SUBSCRIBE / SUBACK
- UNSUBSCRIBE / UNSUBACK
- PINGREQ / PINGRESP

MQTT Producer

- **CONNECT** to the MQTT Broker (+ receives CONNACK)
- PUBLISH a message to a topic
- DISCONNECT

MQTT Consumer

- **CONNECT** to the MQTT Broker (+ receives **CONNACK**)
- **SUBSCRIBE** to a topic (+ **SUBACK**)
- **PUBLISH** message from topic
- **UNSUBSCRIBE** from the topic (+ **UNSUBACK**)
- **DISCONNECT**

Topic Wildcards

- `/` (topic level separator)
  - `/mwm/XYZ/alerts`
- `#` (multilevel wildcard)
  - `/mwm/#` (will match `/mwm/XYZ/alerts`)
- `+` (single level wildcard)
  - `/mwm/+/alerts` (will match `/mwm/XYZ/alerts` but not `/mwm/XYZ/data`)

Quality Of Service (QoS)

- Guarantee of message delivery
  - At Most Once (x1)
  - At Least Once (x2 packets)
  - Exactly Once (x4 packets)

Retained Message

- Last message is retained by the topic and delivered to new consumers
- `Last Known Good` value

Last Will

- Broker will publish a `Last Will` message on a `Last Will` topic on behalf of a client in case of unexpected disconnection
- Setup by the client in `CONNECT`
- Monitor liveliness of clients

Clean Session

- Whether a broker stores any client state between its connections
  - Setup by the client in `CONNECT` based on the client identifier
  - Required for reliable messaging
  - Memory and administration overhead

MQTT Features

- Username / password credentials
- No error handling (MQTT broker closes the connection upon error)
- Heart-beating

MQTTKit

- Objective-C library (iOS, OS X)
- Event-driven (Grand Central Dispatch + Blocks)
- Open Source, Apache License 2.0
- github.com/mobile-web-messaging/MQTTKit

MQTTKit producer

```
// create the client with a unique client ID
NSString *clientID = ...
MQTTClient *client = [[MQTTClient alloc] initWithClientId:clientID];

// connect to the MQTT server
[self.client connectToHost:@"iot.eclipse.org"
         completionHandler:^(NSUInteger code) {
    if (code == ConnectionAccepted) {
        // when the client is connected, send a MQTT message
        [self.client publishString:@"Hello, MQTT"
                           toTopic:@"/MQTTKit/example"
                           withQos:AtMostOnce
                            retain:NO
                 completionHandler:^(int mid) {
            NSLog(@"message has been delivered");
        }];
    }
}];
```

MQTTKit consumer

```
// create the client with a unique client ID
NSString *clientID = ...
MQTTClient *client = [[MQTTClient alloc] initWithClientId:clientID];

// define the handler that will be called when MQTT messages are
//received by the client
[client setMessageHandler:^(MQTTMessage *message) {
    NSString *text = [message.payloadString];
    NSLog(@"received message %@", text);
}];

// connect the MQTT client
[client connectToHost:@"iot.eclipse.org"
          completionHandler:^(MQTTConnectionReturnCode code) {
    if (code == ConnectionAccepted) {
        // when the client is connected, subscribe to the topic
        // to receive message.
        [self.client subscribe:@"/MQTTKit/example"
         withCompletionHandler:nil];
    }
}];
```

mqtt.js

- JavaScript library for Web browsers
- Open Source, Eclipse Public License 1.0
- eclipse.org/paho/clients/js/

mqtt.js producer

```
var clientID = Math.random().toString(12);
var client = new Messaging.Client(“iot.eclipse.org", 80, clientID);

client.connect({onSuccess: function(frame) {
    // this function is executed after a successful connection to
    // the MQTT broker
    var message = new Messaging.Message(“Alert from XYZ!!”);
    message.destinationName = “/mwm/XYZ/alerts”;
    client.send(message);
  },
  onFailure: function(failure) {
    alert(failure.errorMessage);
  }
});
```

mqtt.js consumer

```
var clientID = Math.random().toString(12);
var client = new Messaging.Client(“iot.eclipse.org", 80, clientID);

client.connect({onSuccess: function(frame) {
    // once the client is successfully connected,
    // subscribe to all the mwm topics
      client.subscribe("/mwm/+/alerts");
  },
  onFailure: function(failure) {
    alert(failure.errorMessage);
  }
});

// subscription callback
client.onMessageArrived = function(message) {
  console.log(“got message “ + message.payloadString);
  // Alert from XYZ!!
};
```

Conclusion

- Messaging to complement request/reply protocols (HTTP)
- MQTT & STOMP as first-mile protocols (from device/browser to broker)
- Mobile devices and modern Web browsers
- Internet of Things

Q & A

Thank You!

- jmesnil.net
- jmesnil@gmail.com
- @jmesnil