An Introduction to, and Comparison of, the Different MQ APIs

Matthew Whitehead
IBM MQ Development
mwhitehead@uk.ibm.com

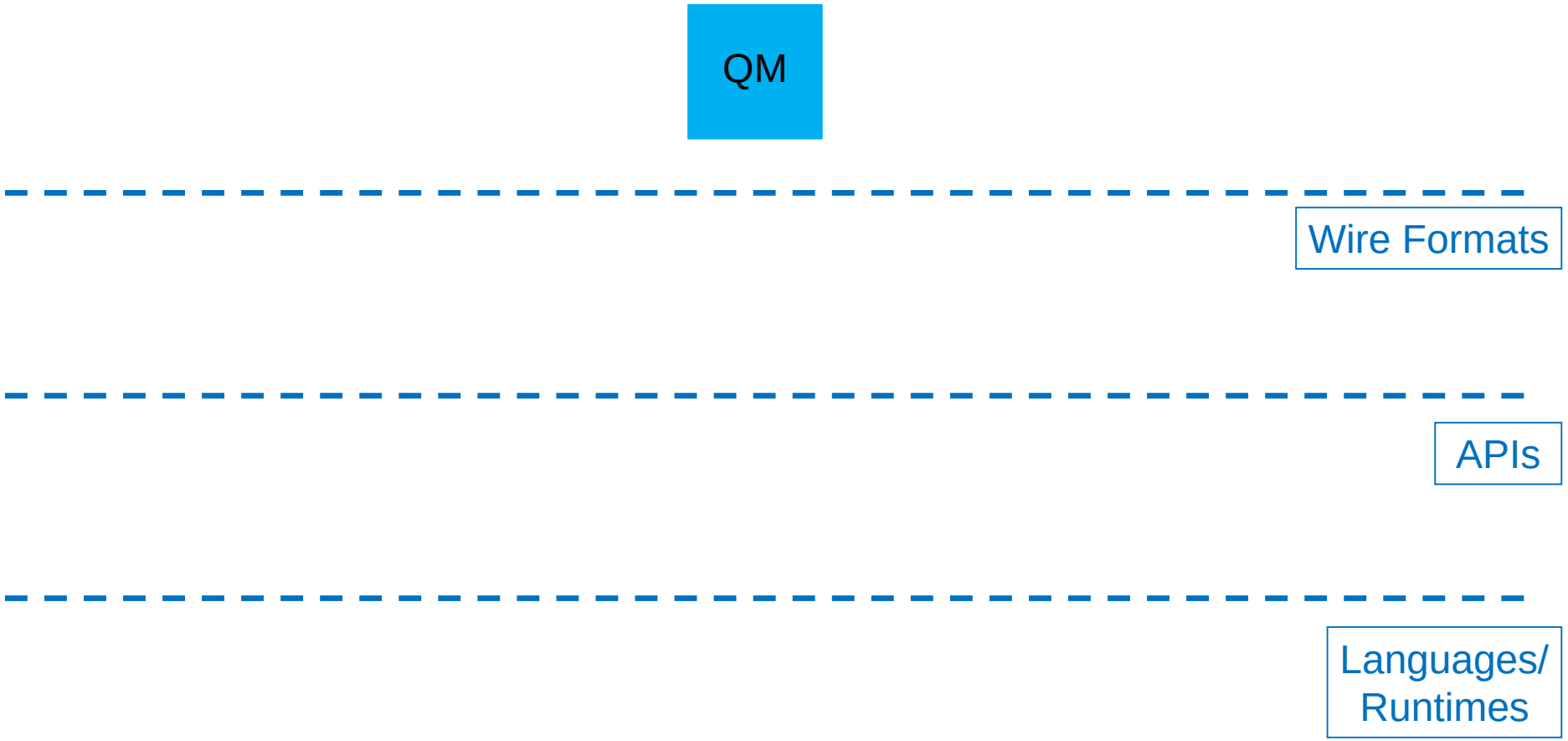
Agenda

- Languages, Wire Formats, and APIs
- An Overview of C MQI, JMS, and MQ Light
- Recap of the Key MQ Features
- Features supported in the different MQ APIs

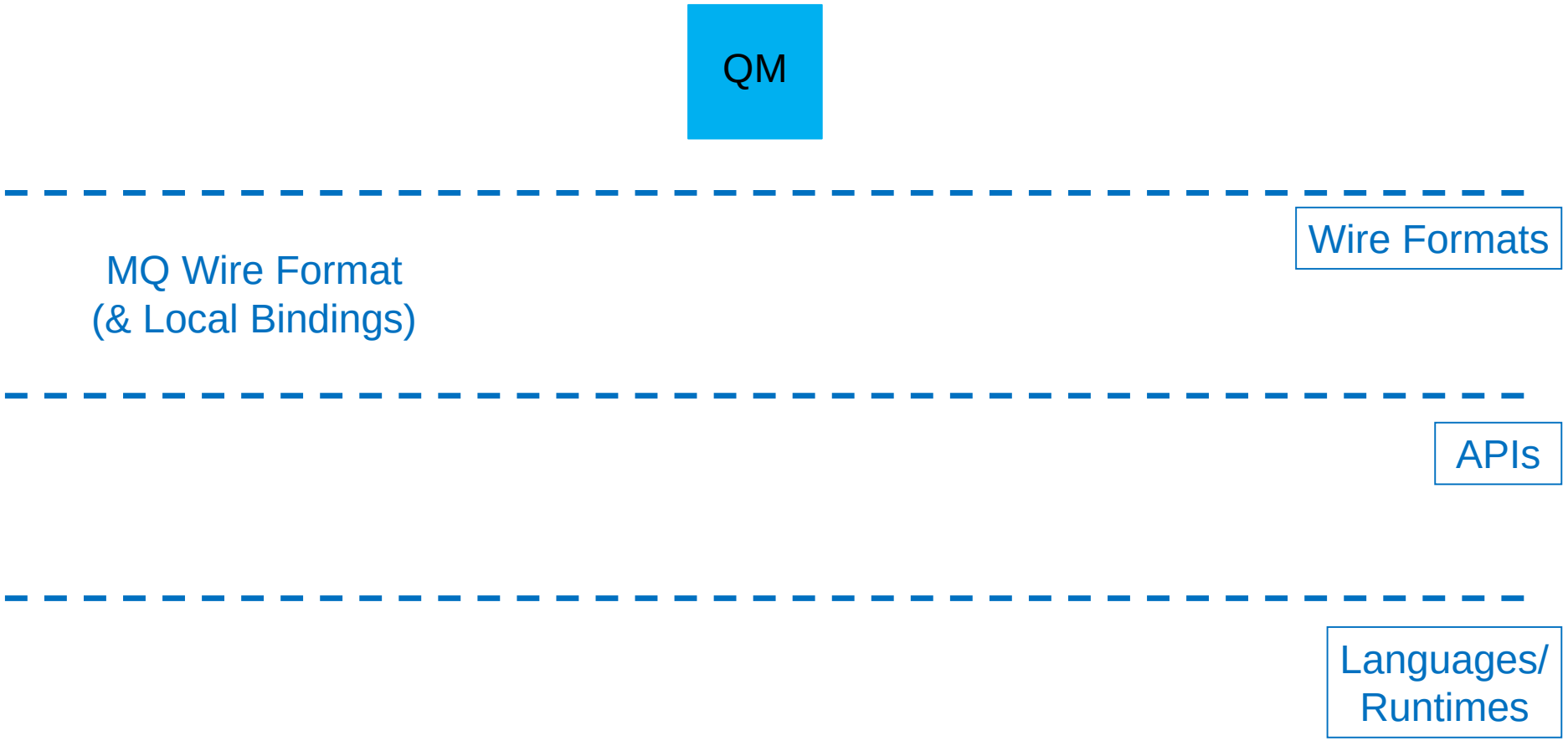
Agenda

- **Languages, Wire Formats, and APIs**
- **An Overview of C MQI, JMS, and MQ Light**
- **Recap of the Key MQ Features**
- **Features supported in the different MQ APIs**

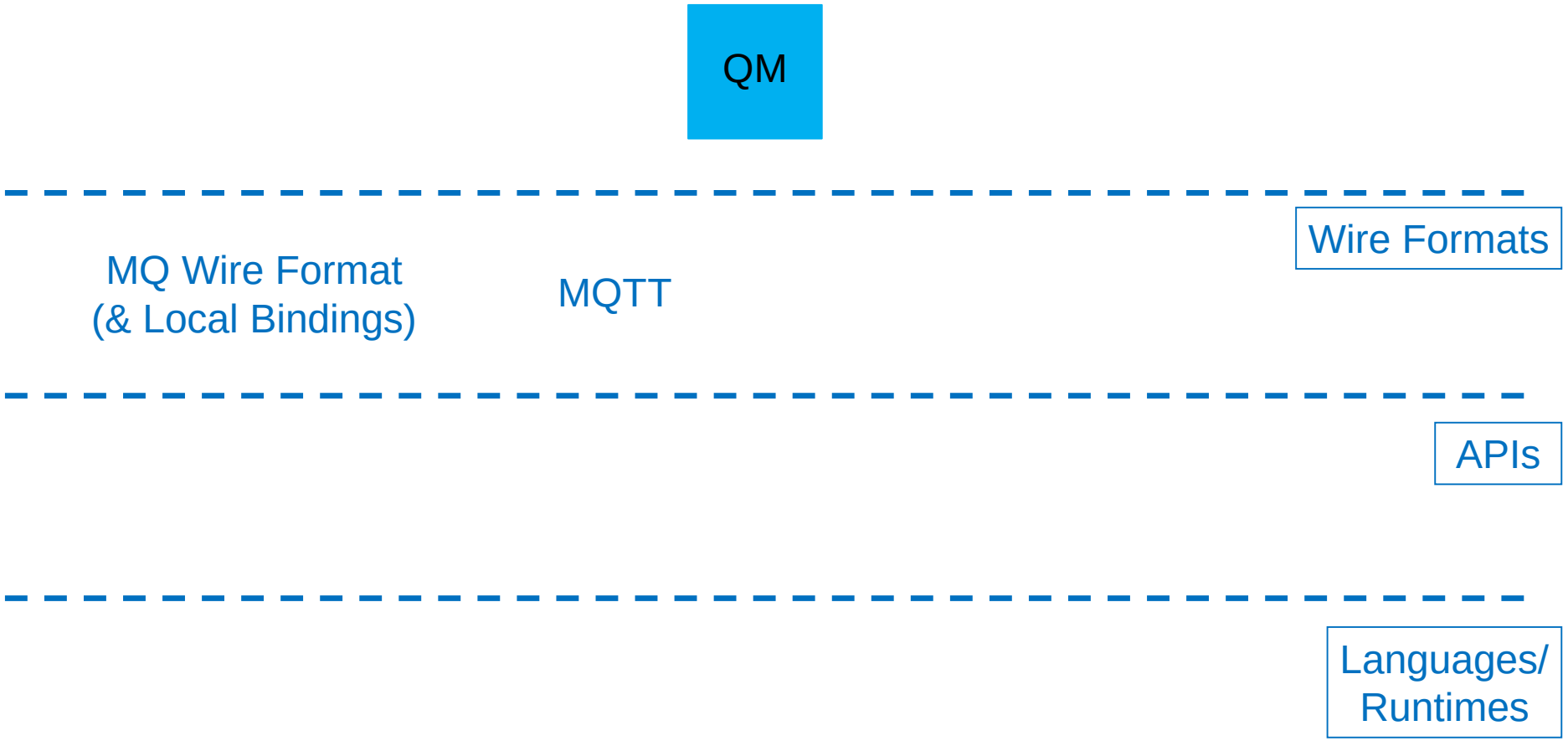
Languages, Wire Formats, and APIs



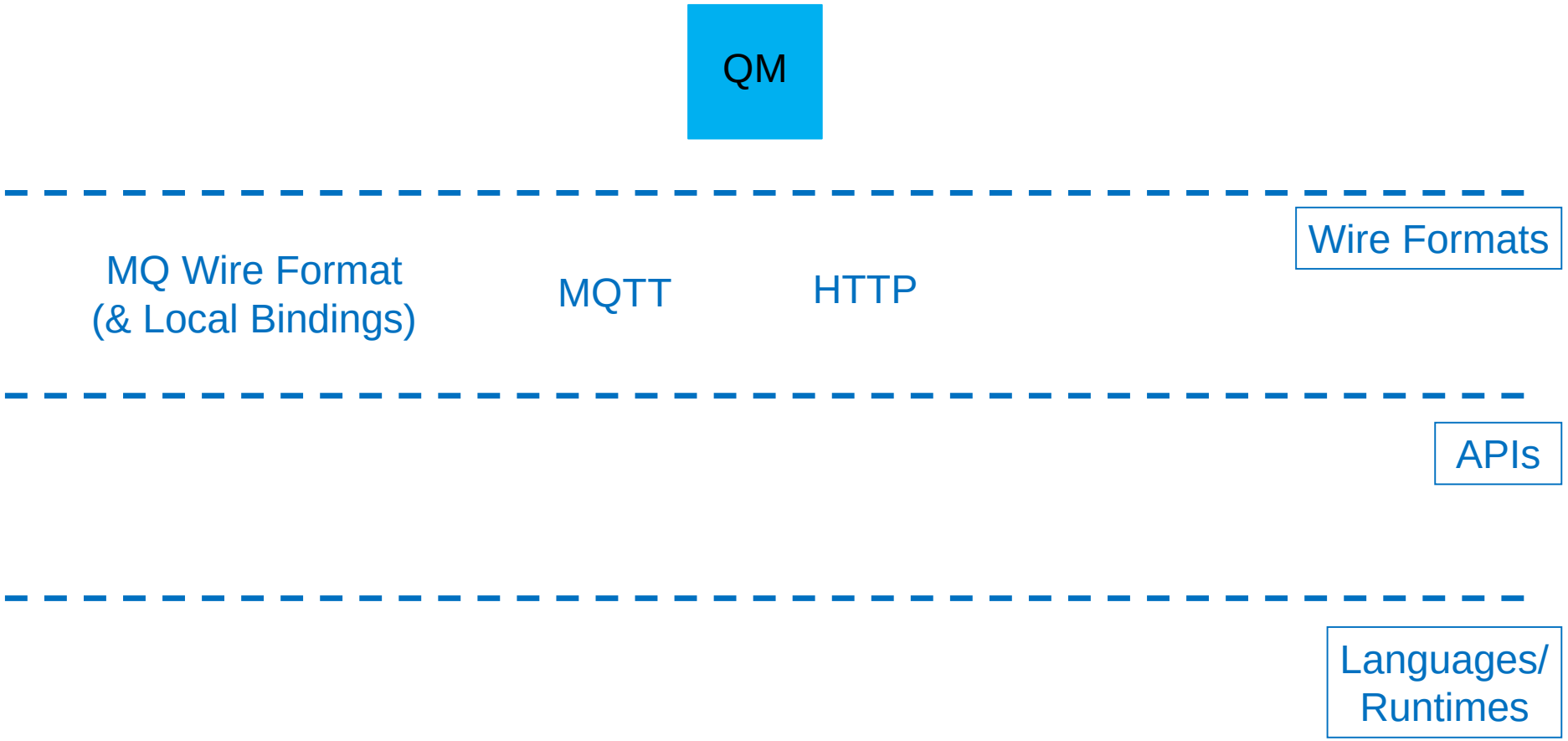
Languages, Wire Formats, and APIs



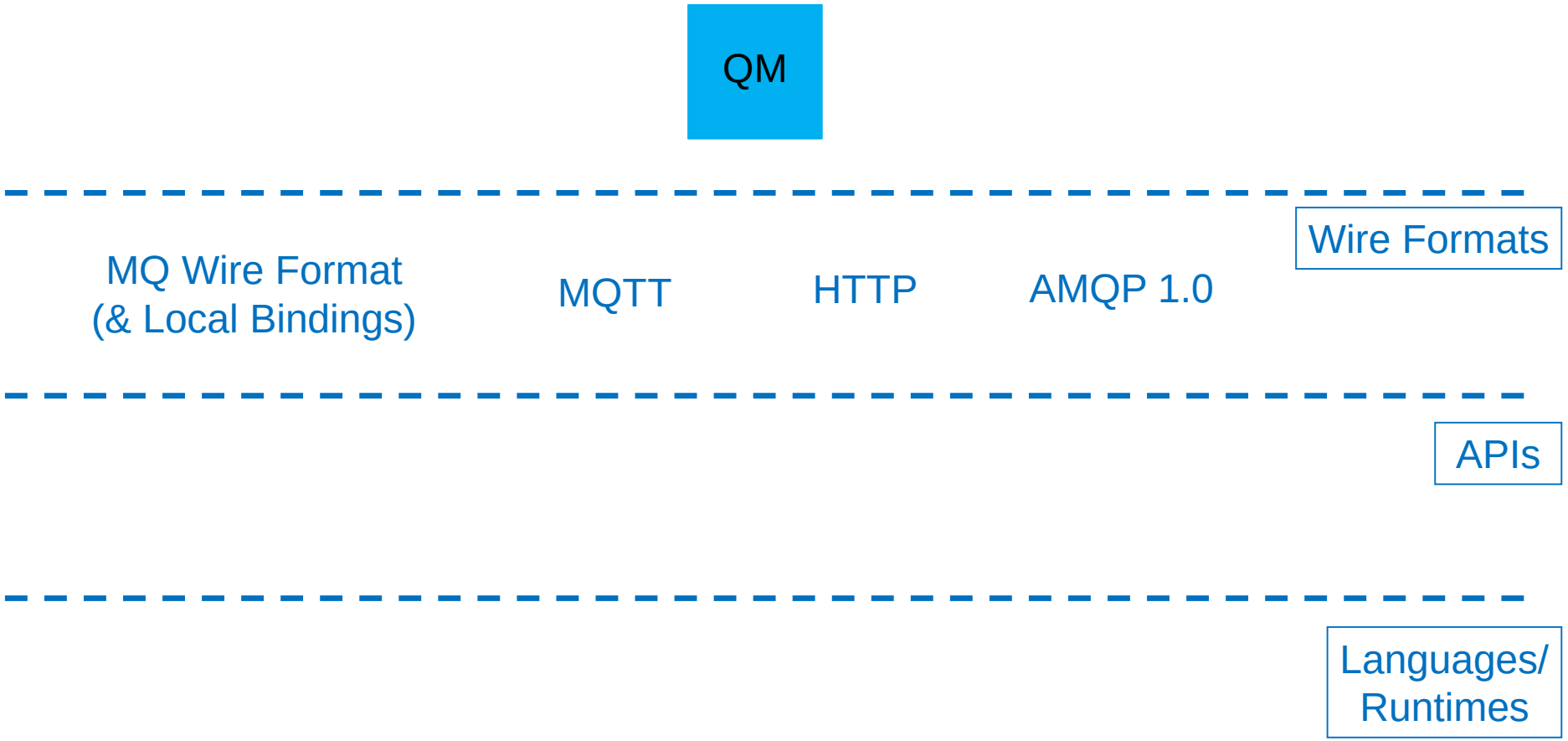
Languages, Wire Formats, and APIs



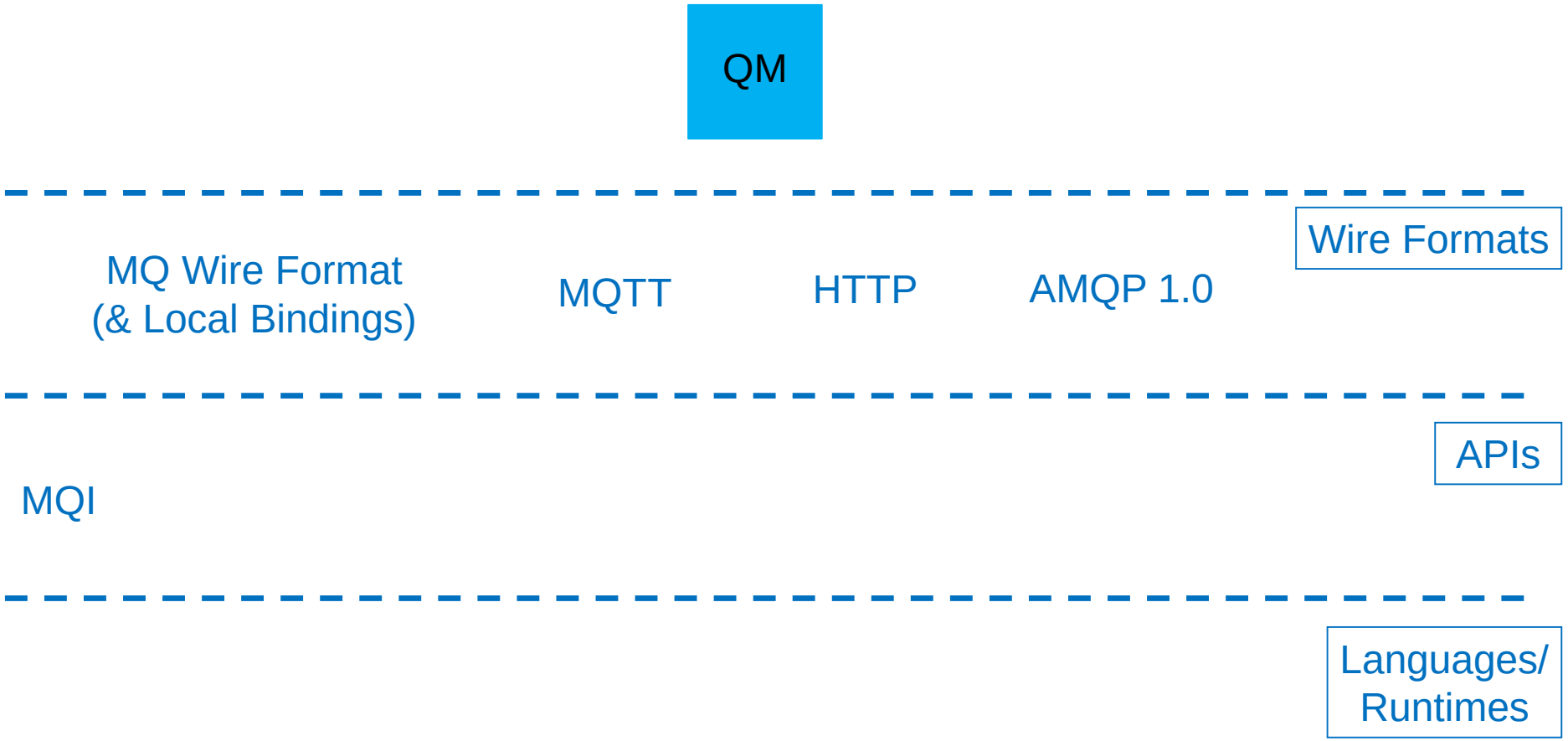
Languages, Wire Formats, and APIs



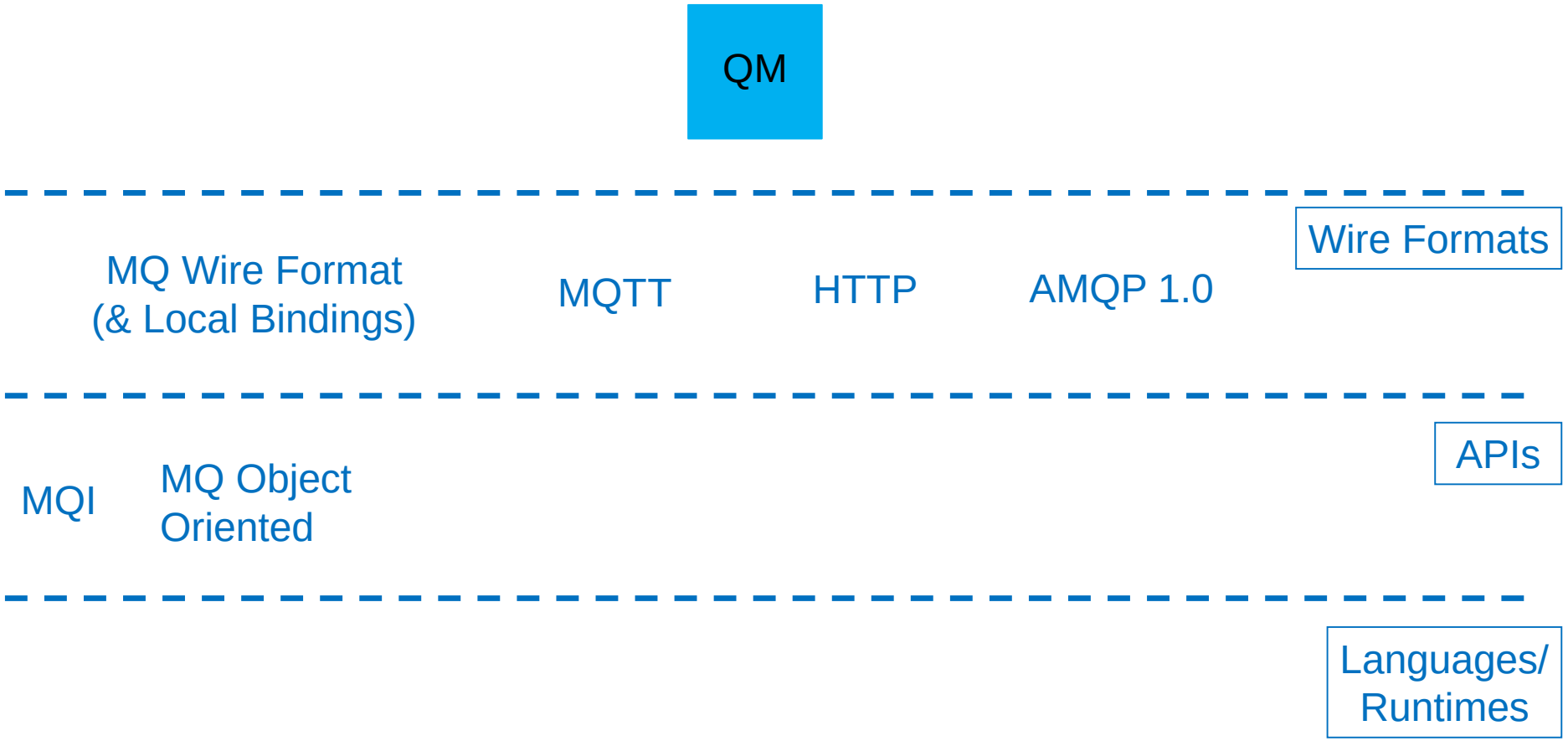
Languages, Wire Formats, and APIs



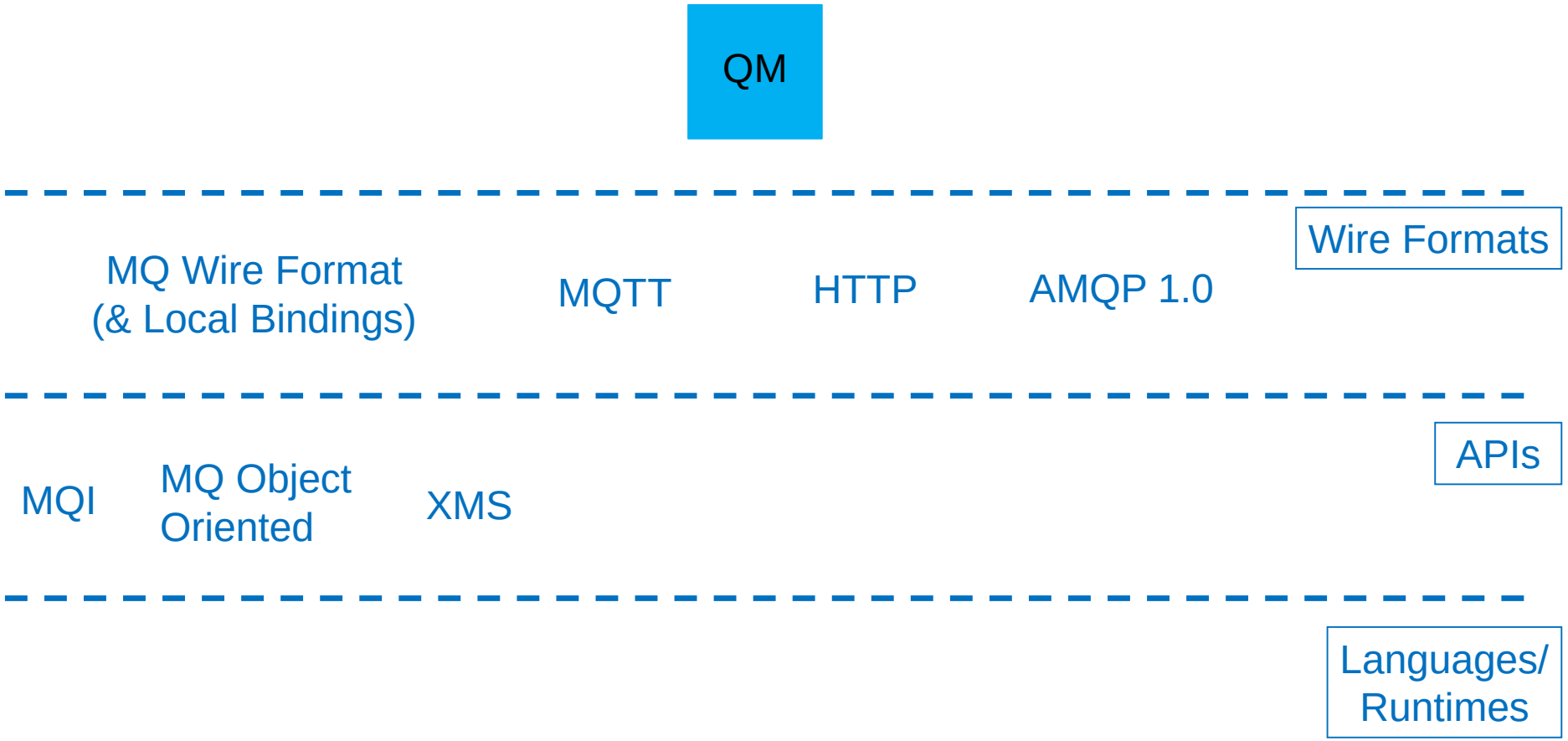
Languages, Wire Formats, and APIs



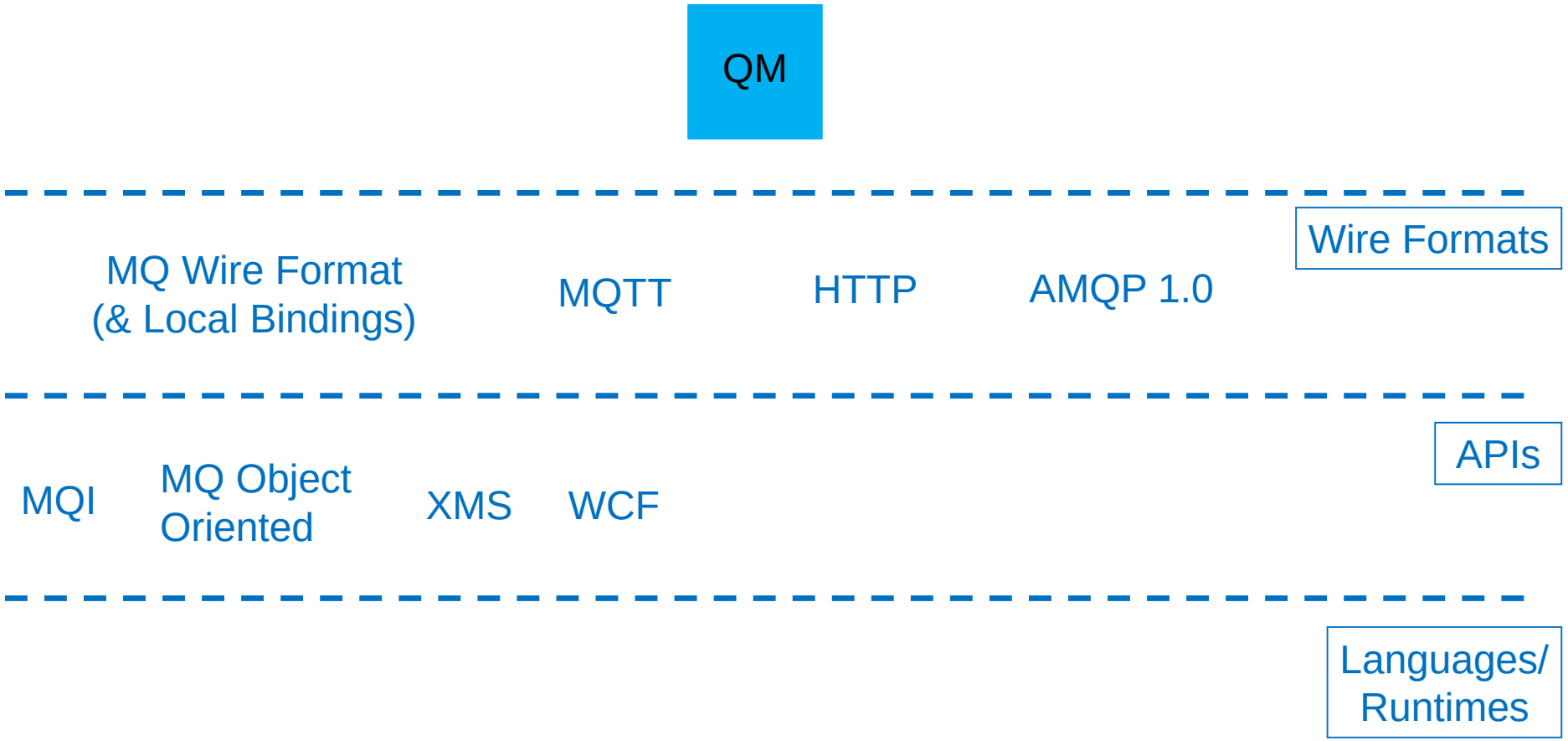
Languages, Wire Formats, and APIs



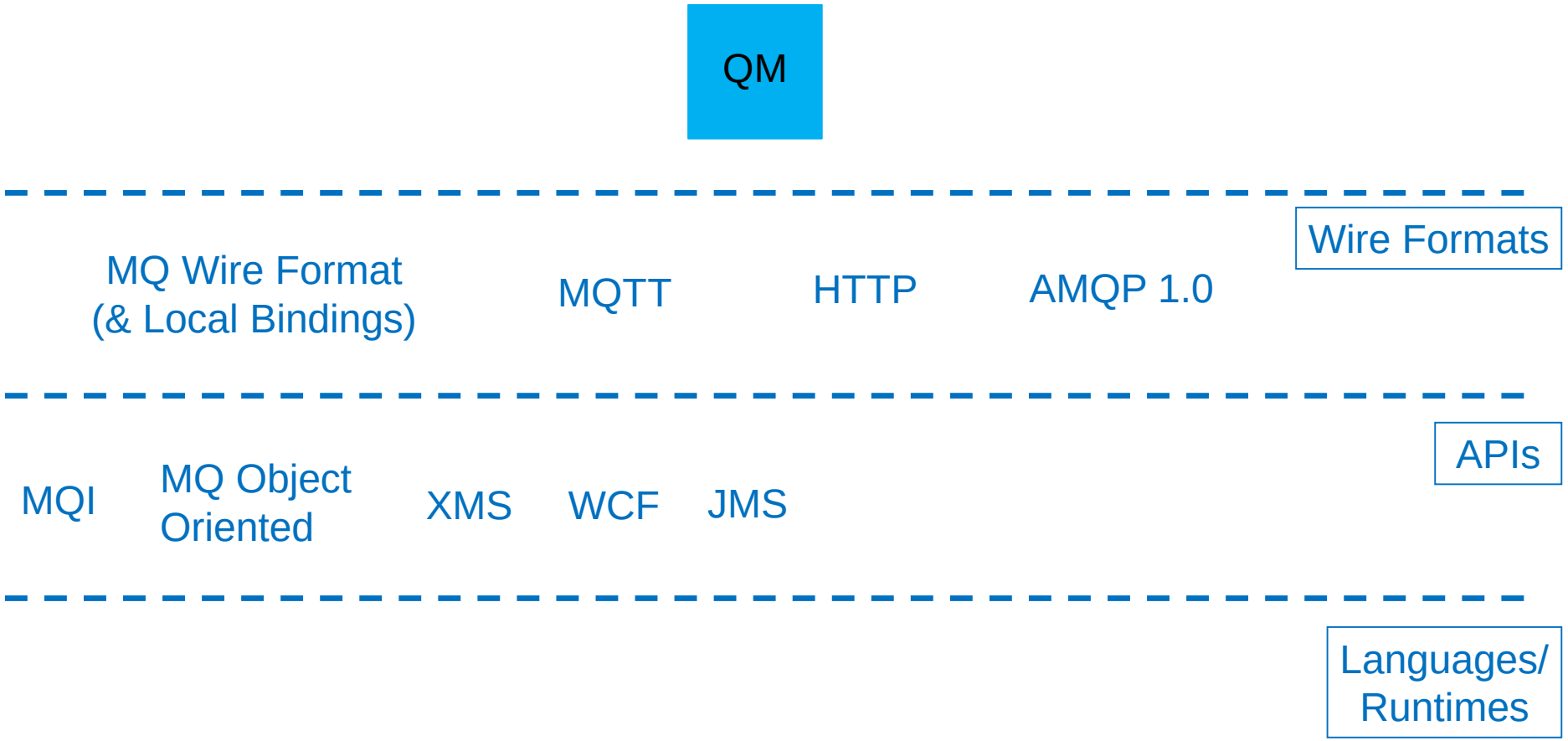
Languages, Wire Formats, and APIs



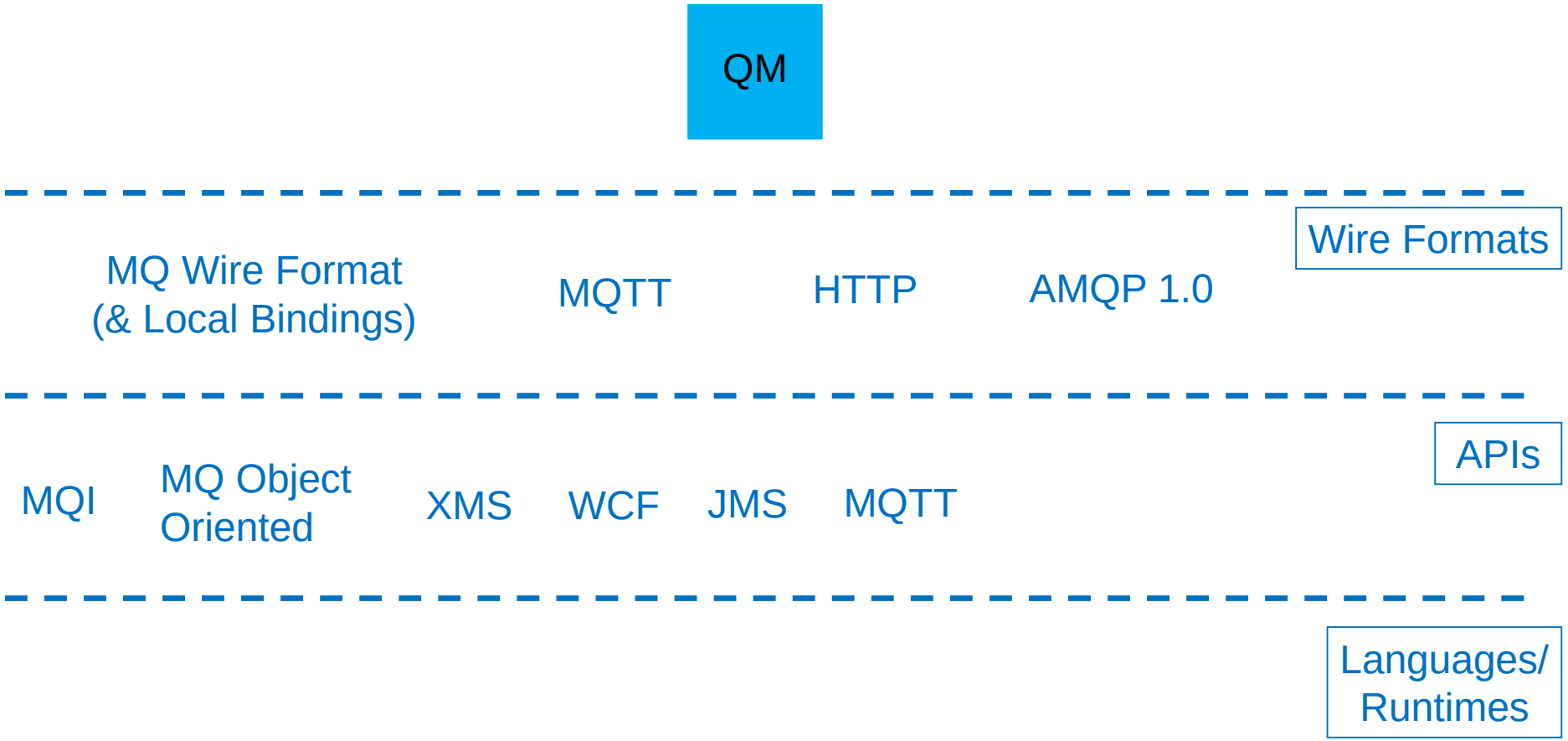
Languages, Wire Formats, and APIs



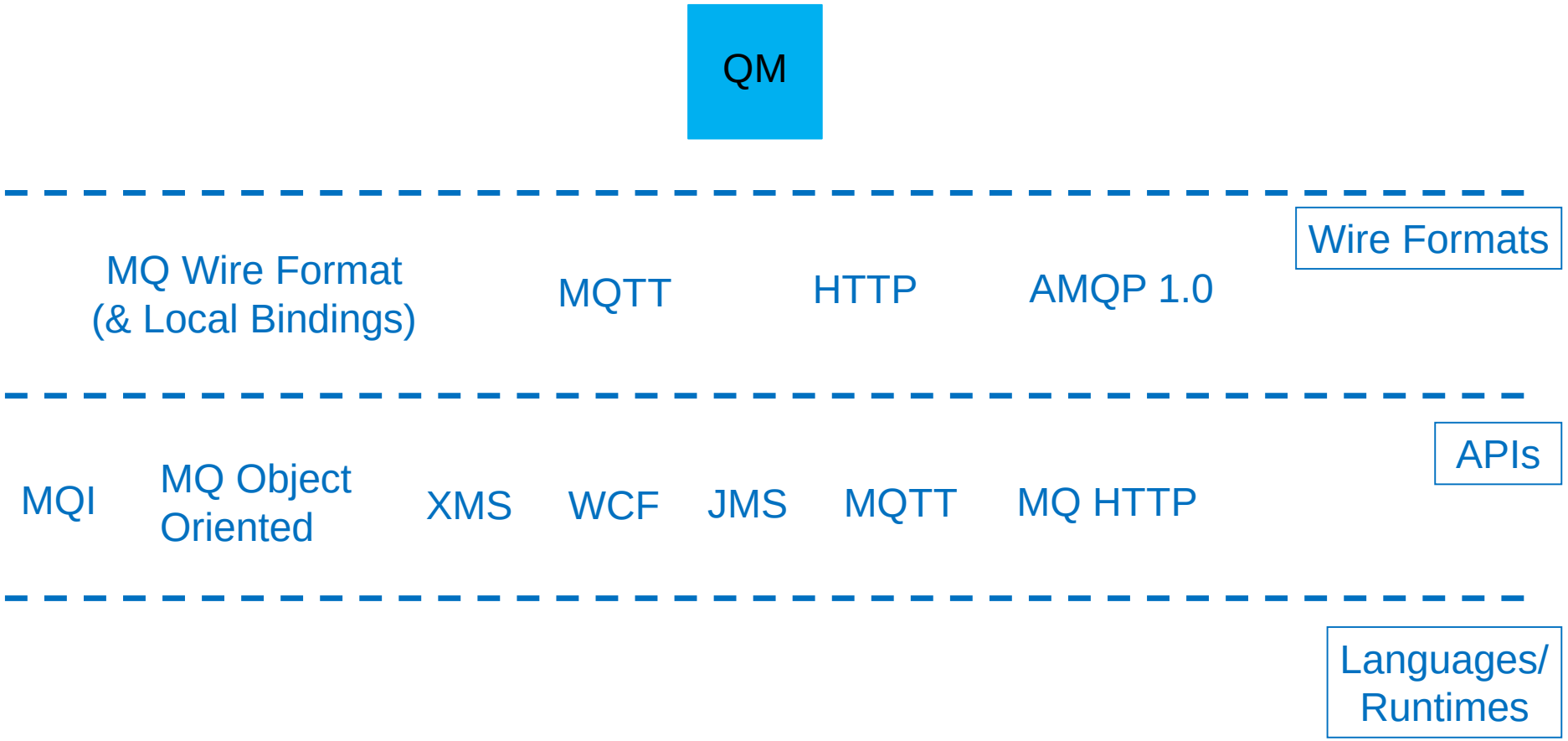
Languages, Wire Formats, and APIs



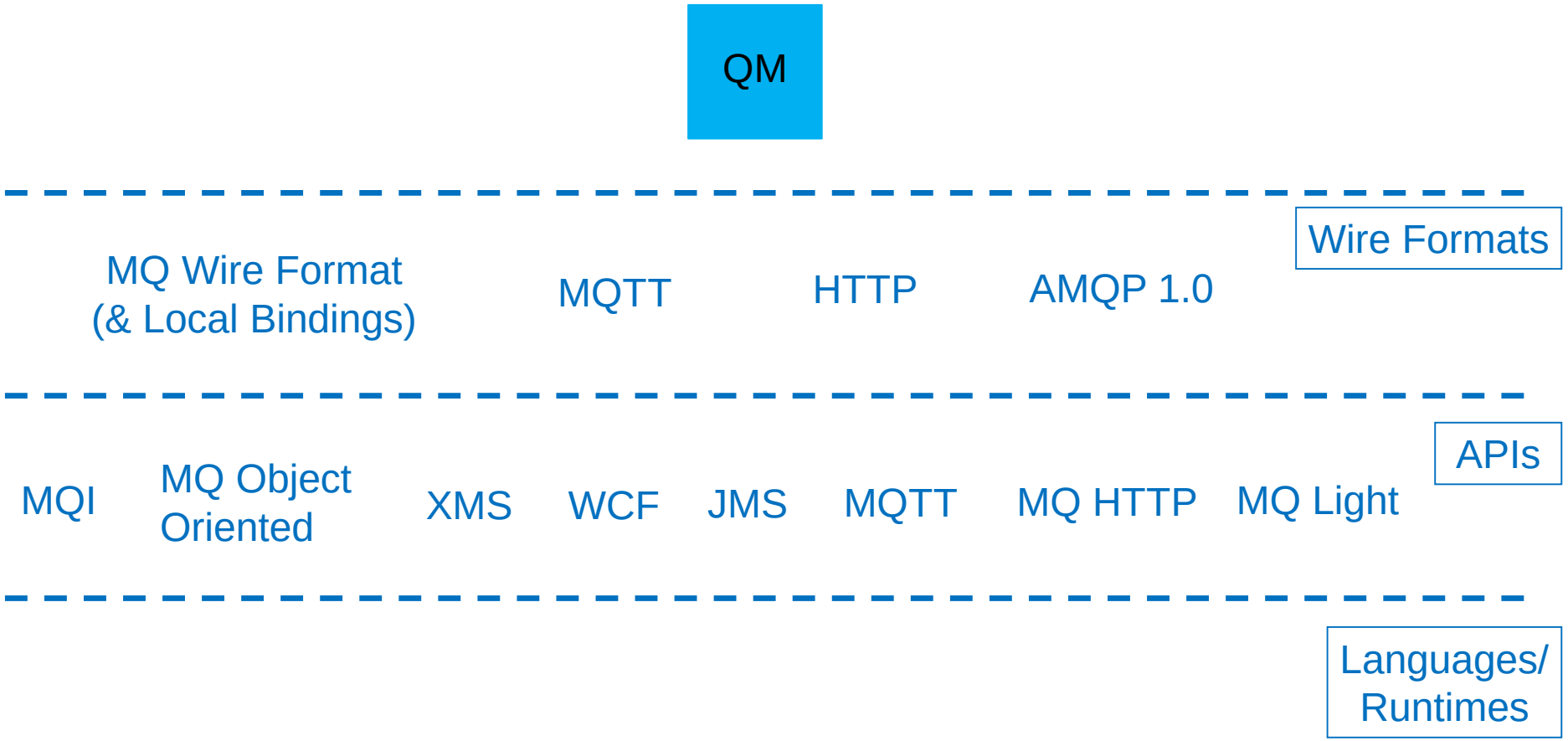
Languages, Wire Formats, and APIs



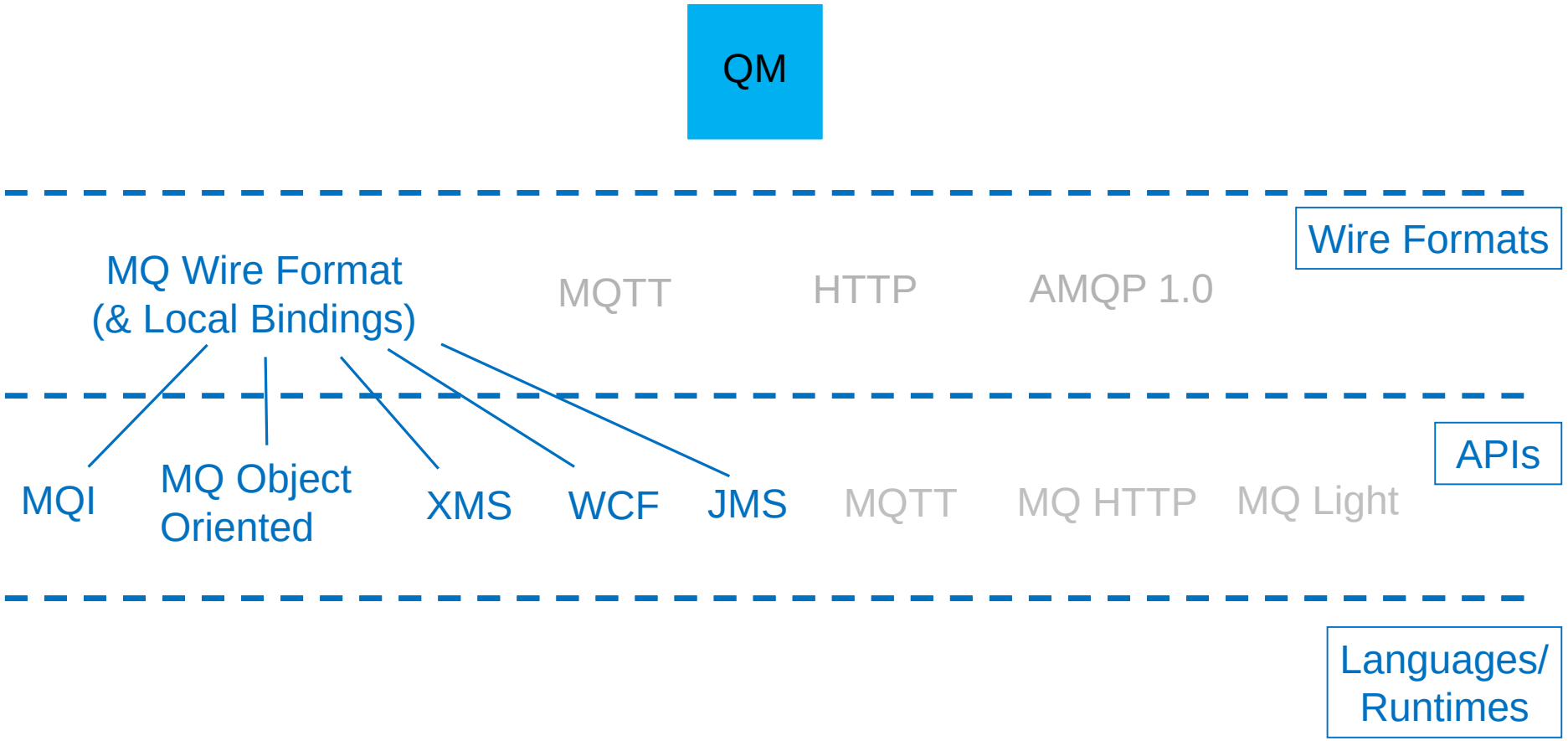
Languages, Wire Formats, and APIs



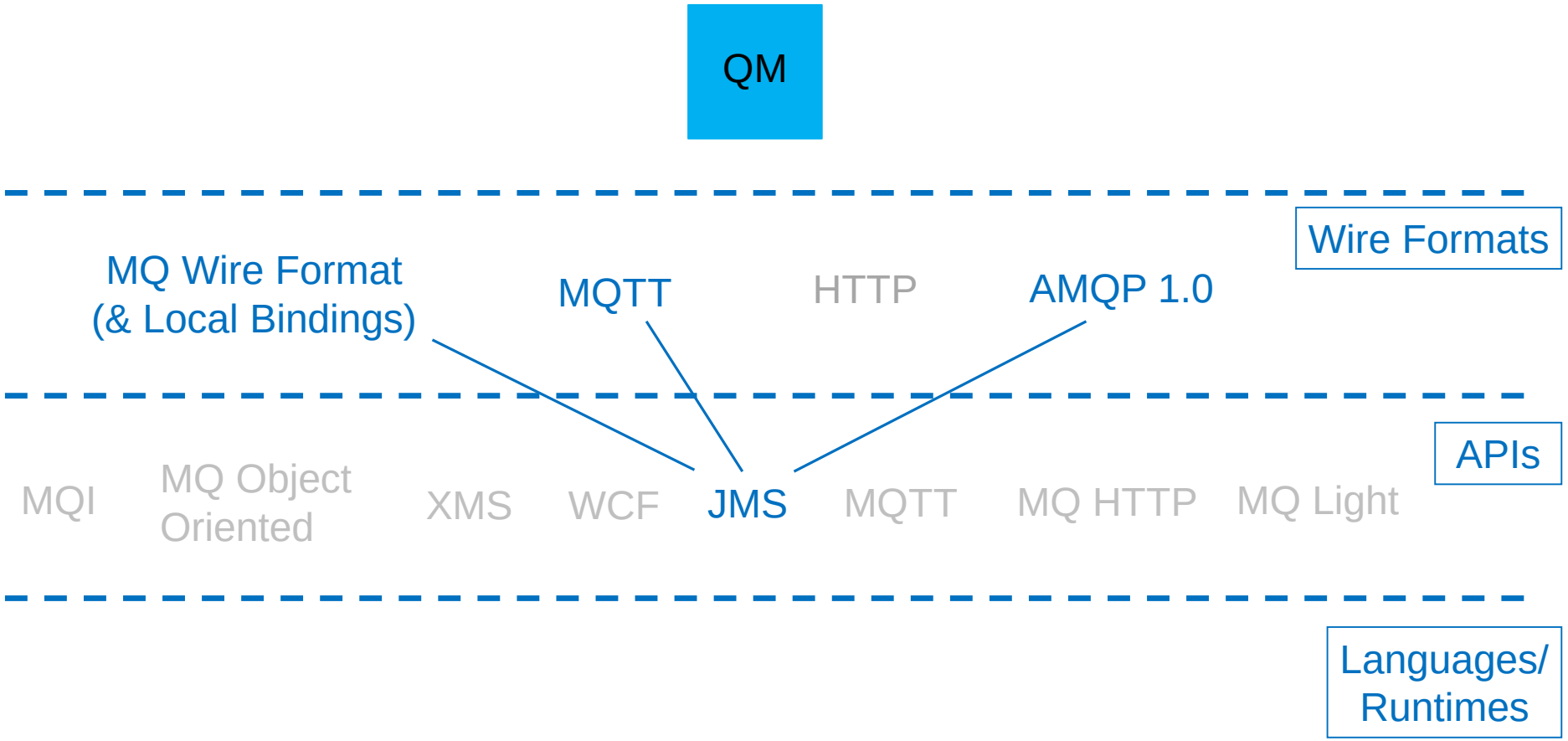
Languages, Wire Formats, and APIs



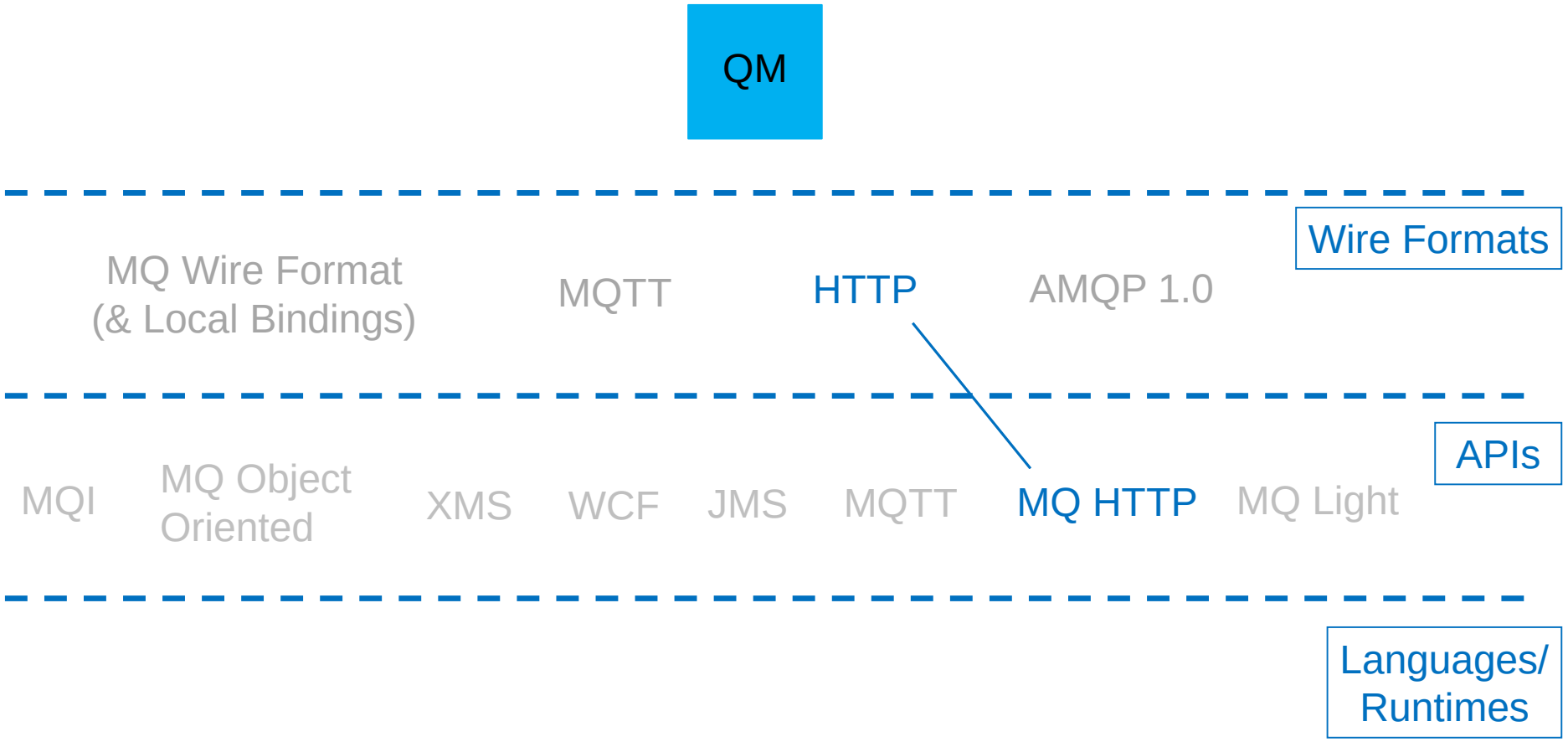
Languages, Wire Formats, and APIs



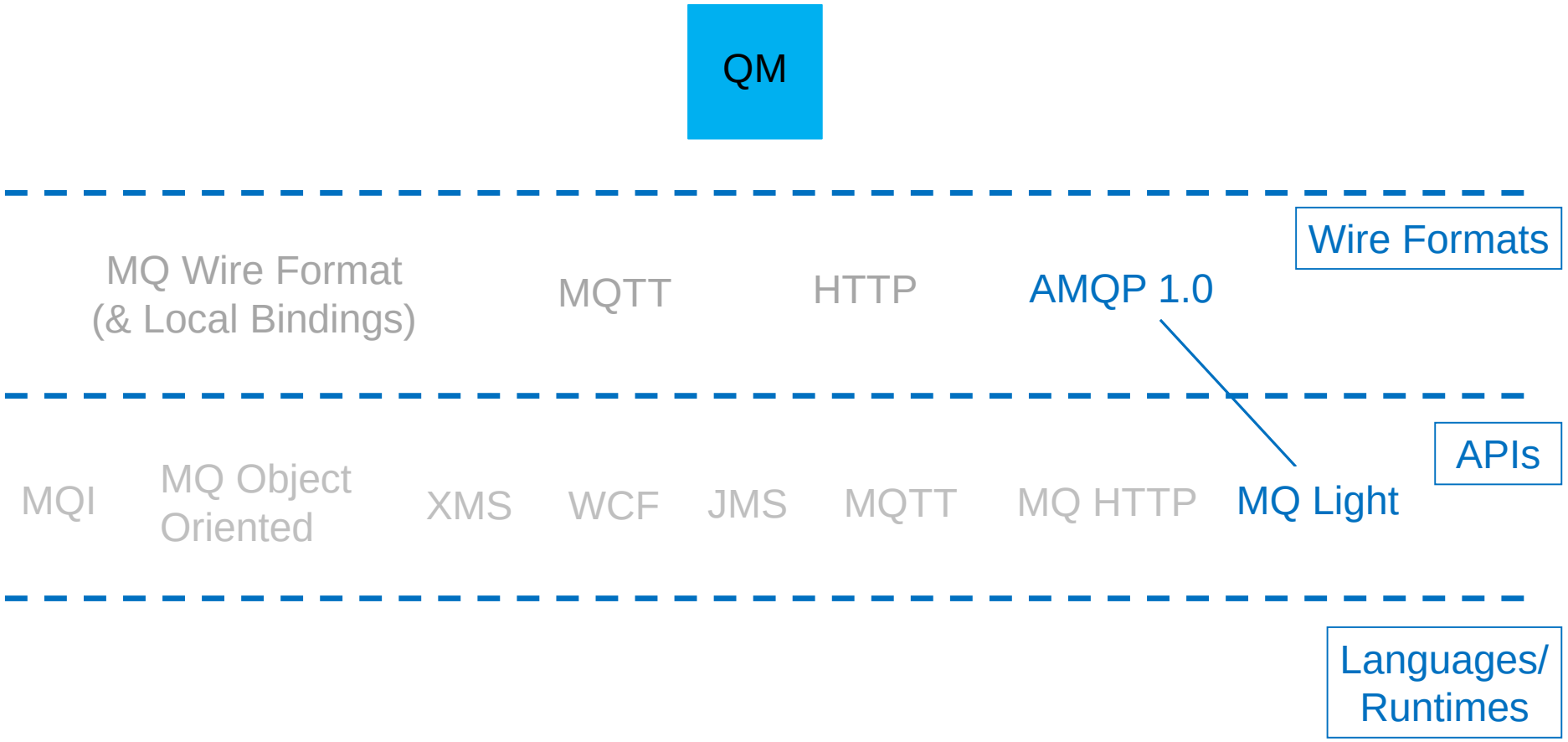
Languages, Wire Formats, and APIs



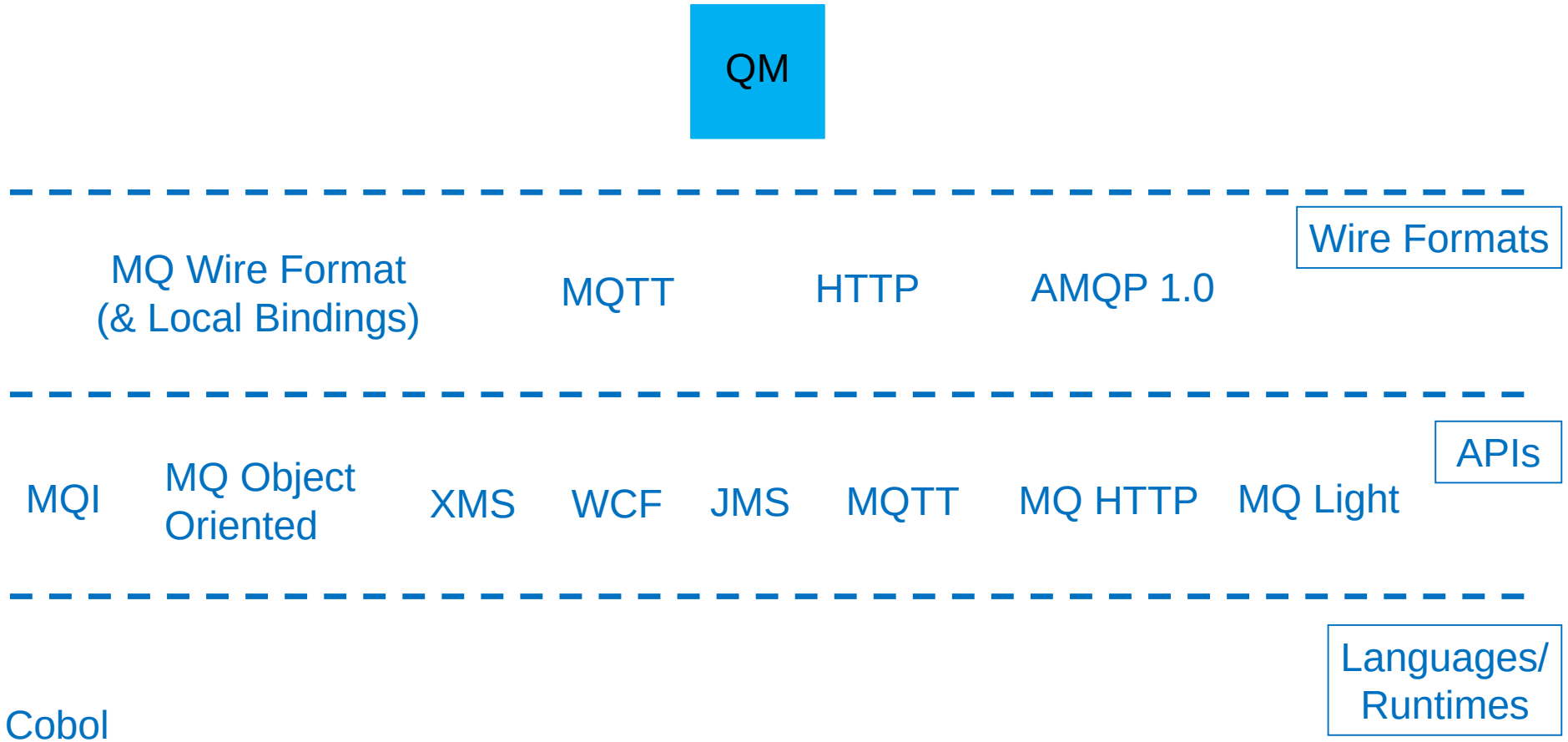
Languages, Wire Formats, and APIs



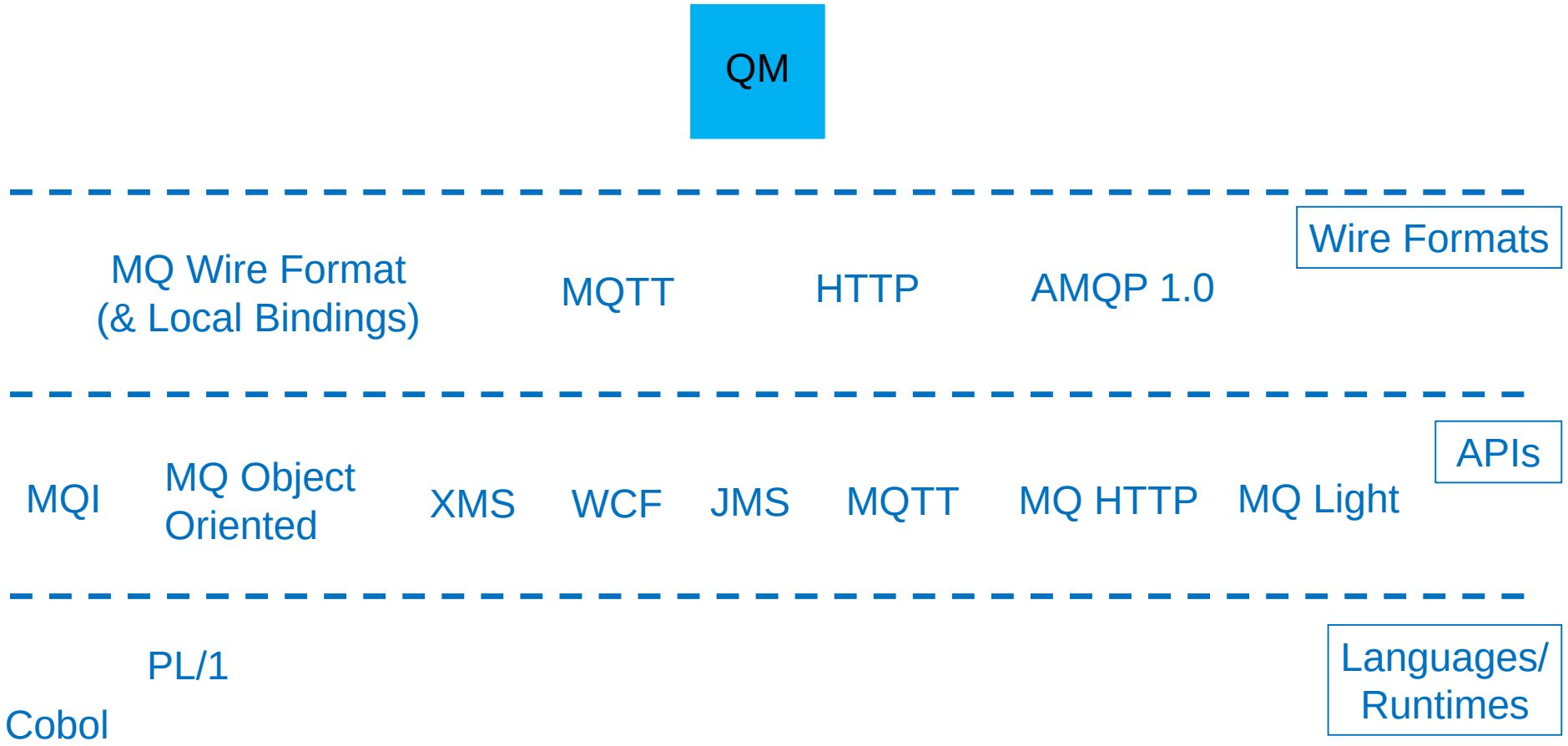
Languages, Wire Formats, and APIs



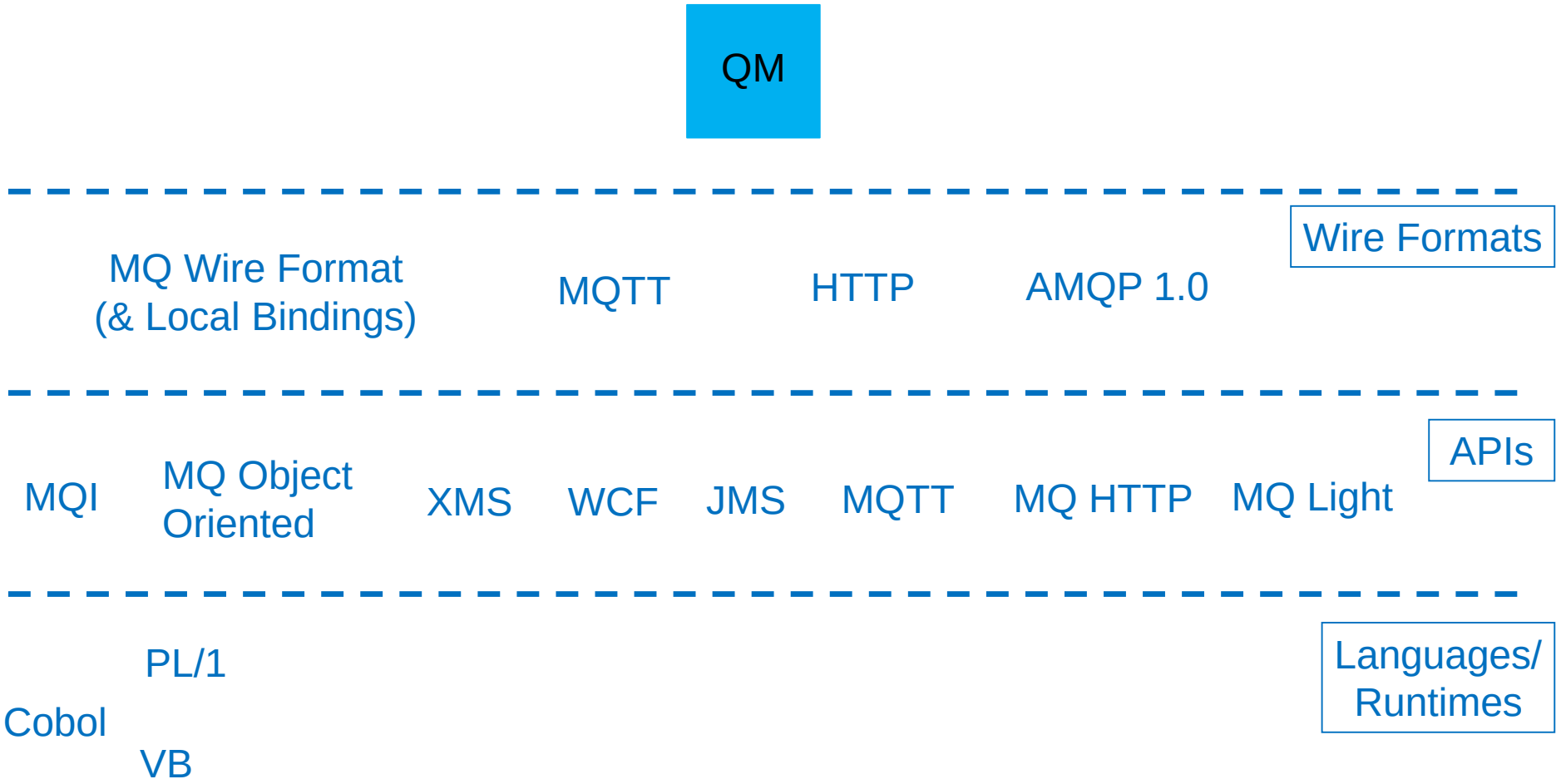
Languages, Wire Formats, and APIs



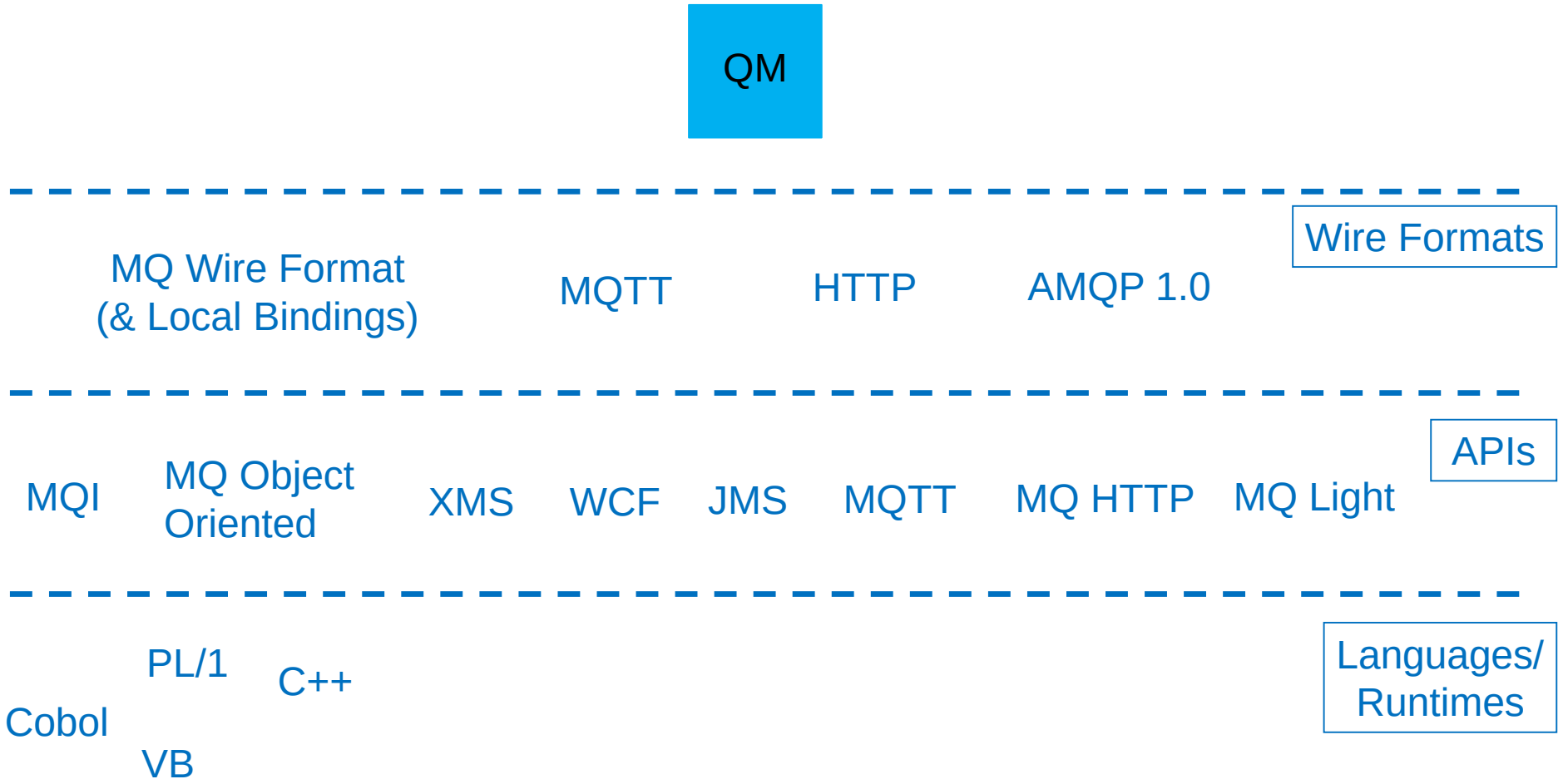
Languages, Wire Formats, and APIs



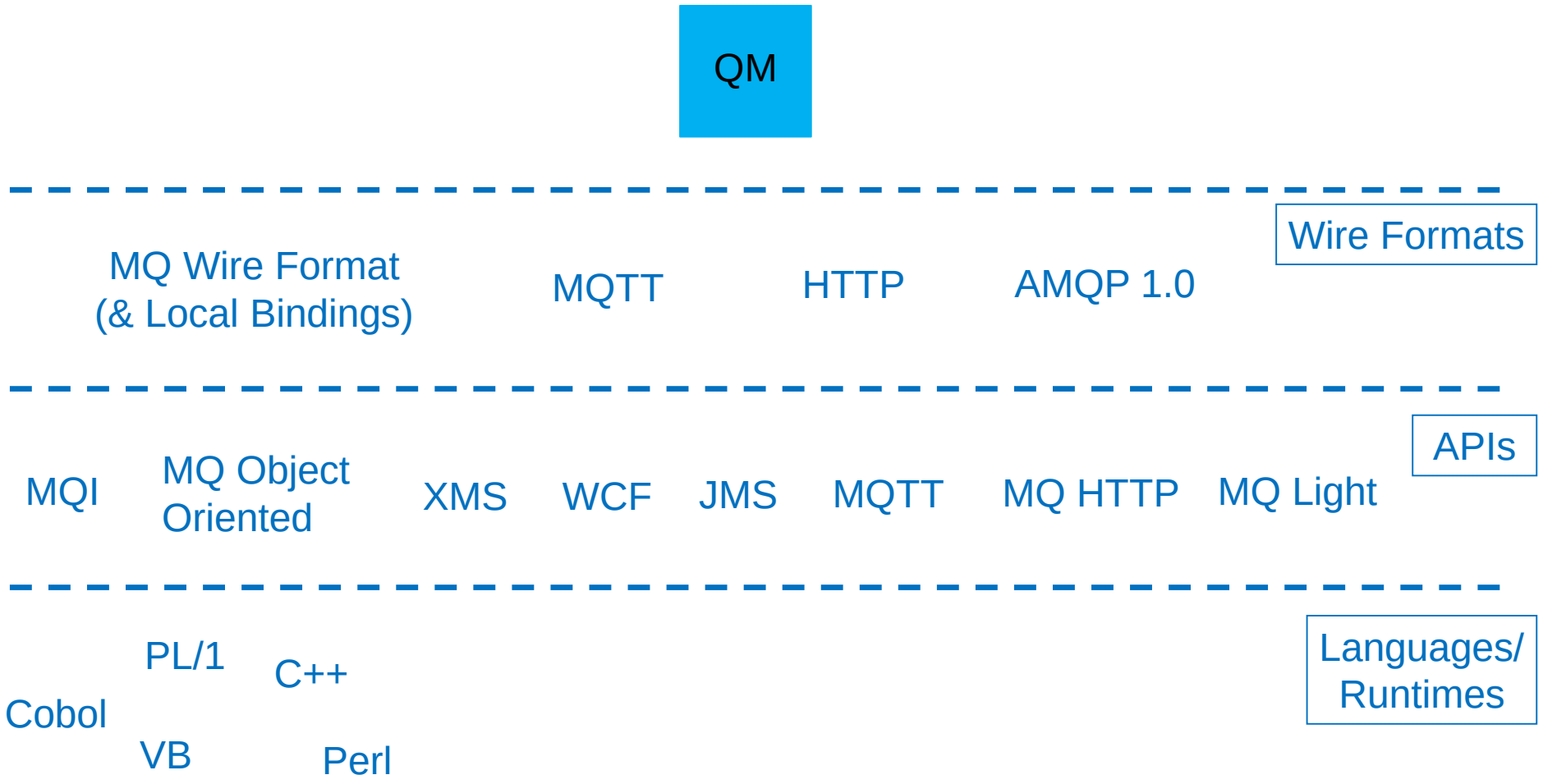
Languages, Wire Formats, and APIs



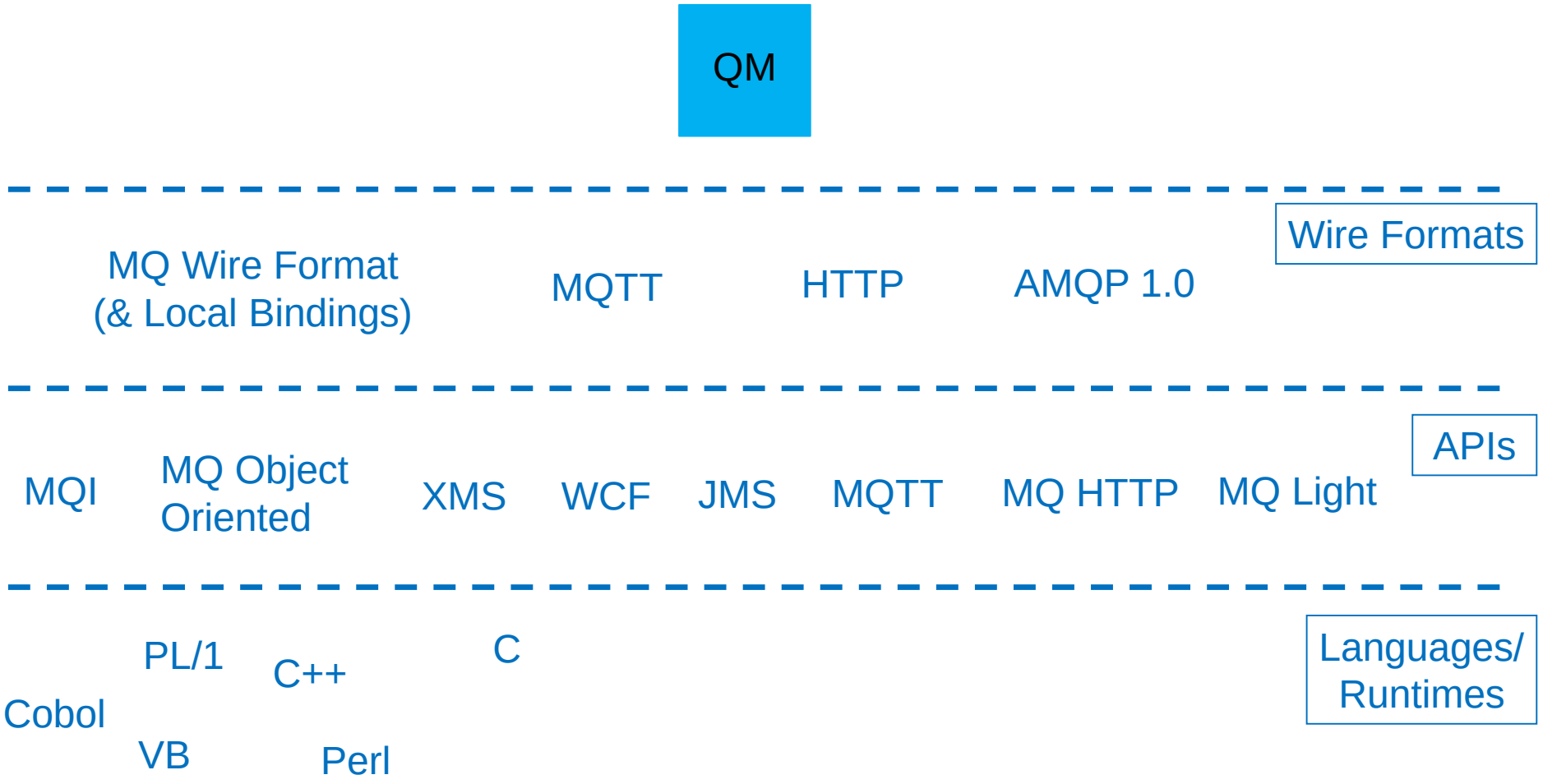
Languages, Wire Formats, and APIs



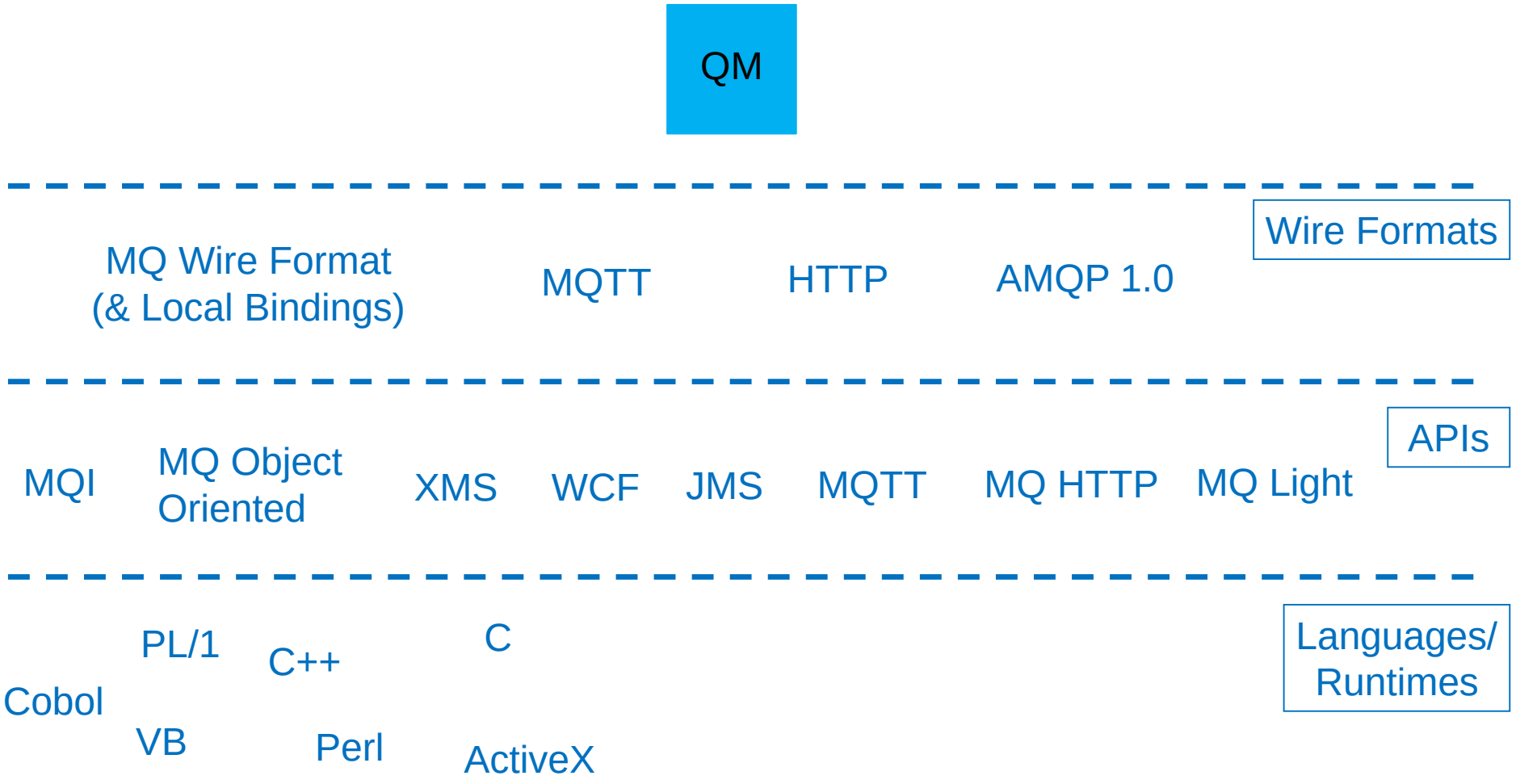
Languages, Wire Formats, and APIs



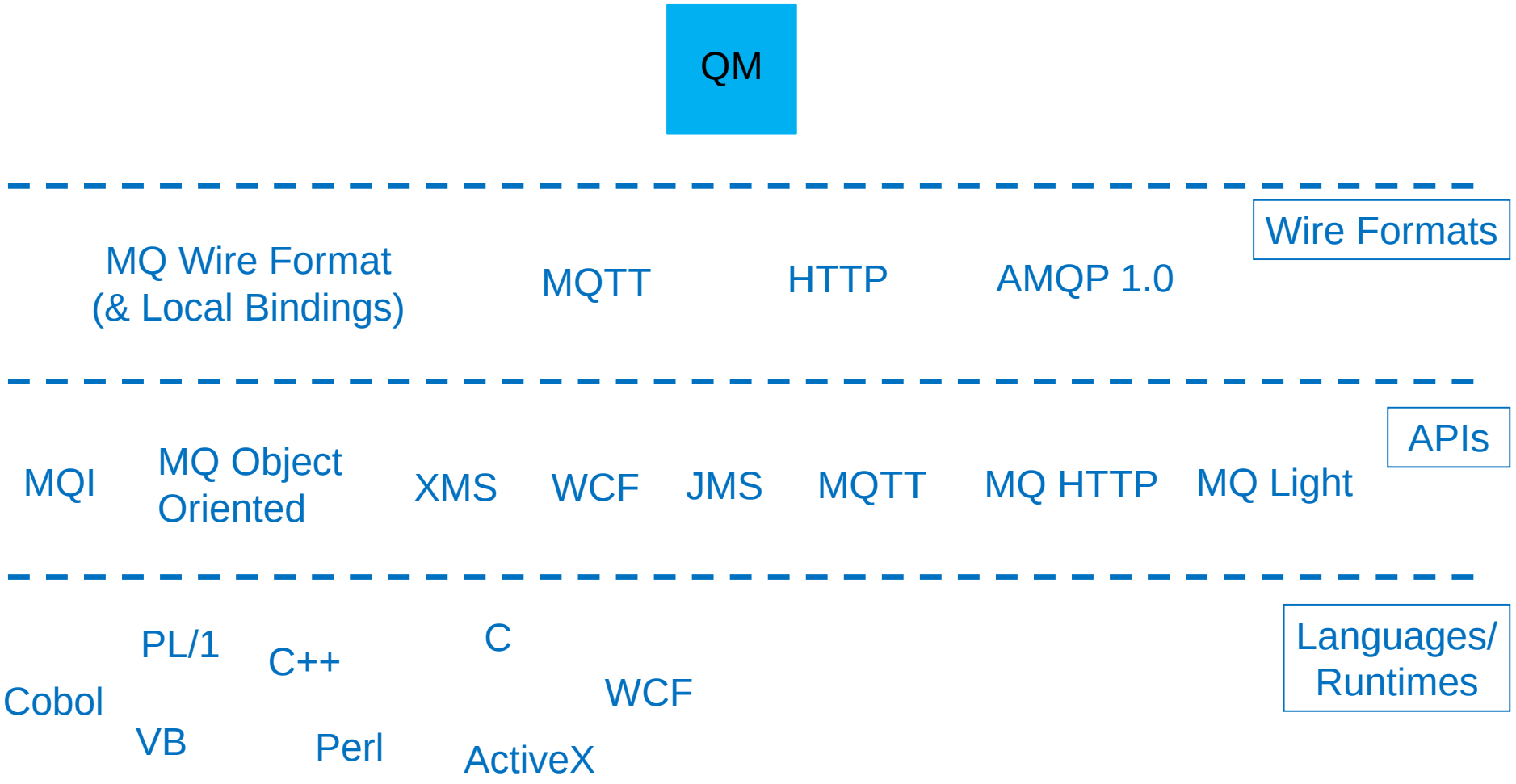
Languages, Wire Formats, and APIs



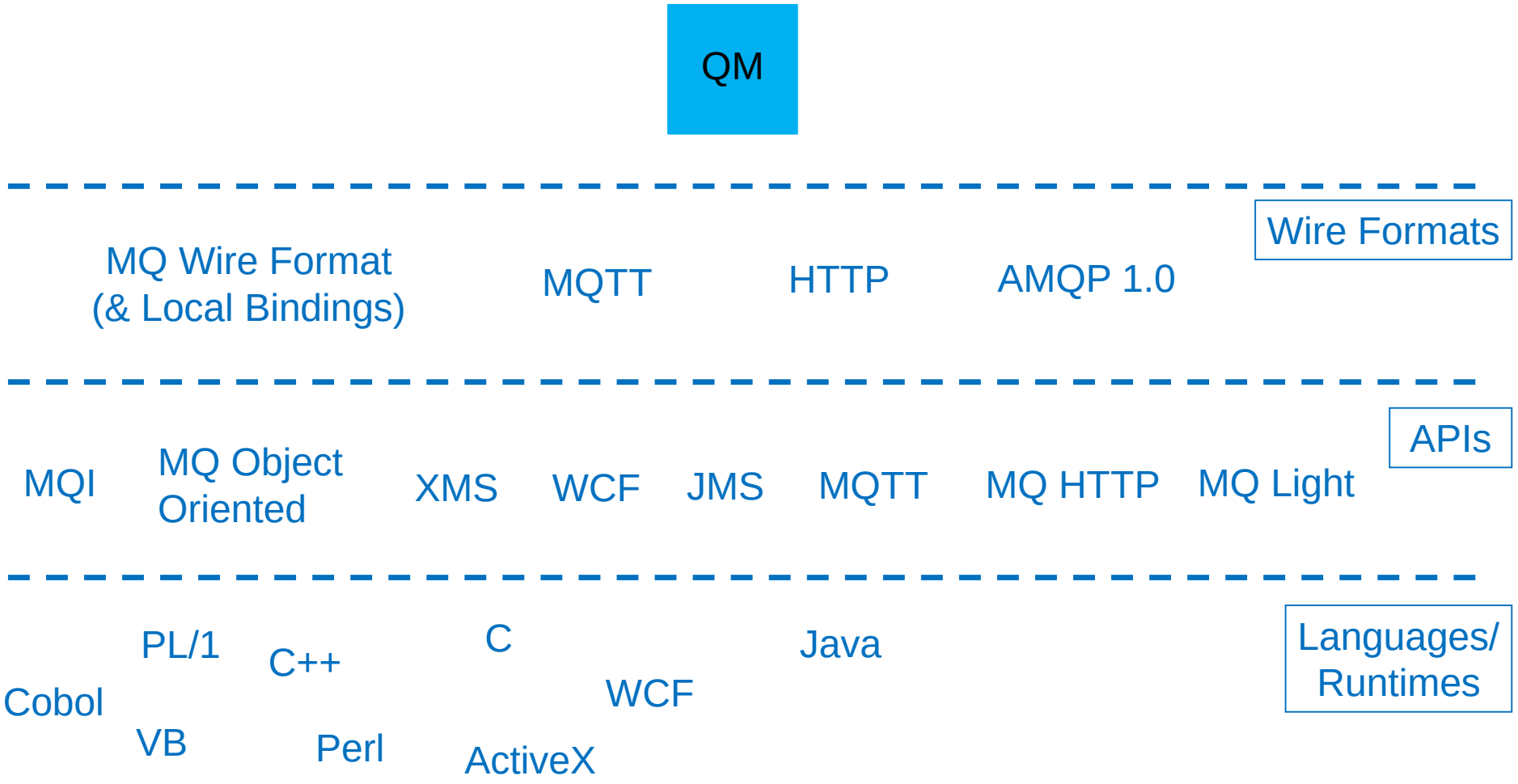
Languages, Wire Formats, and APIs



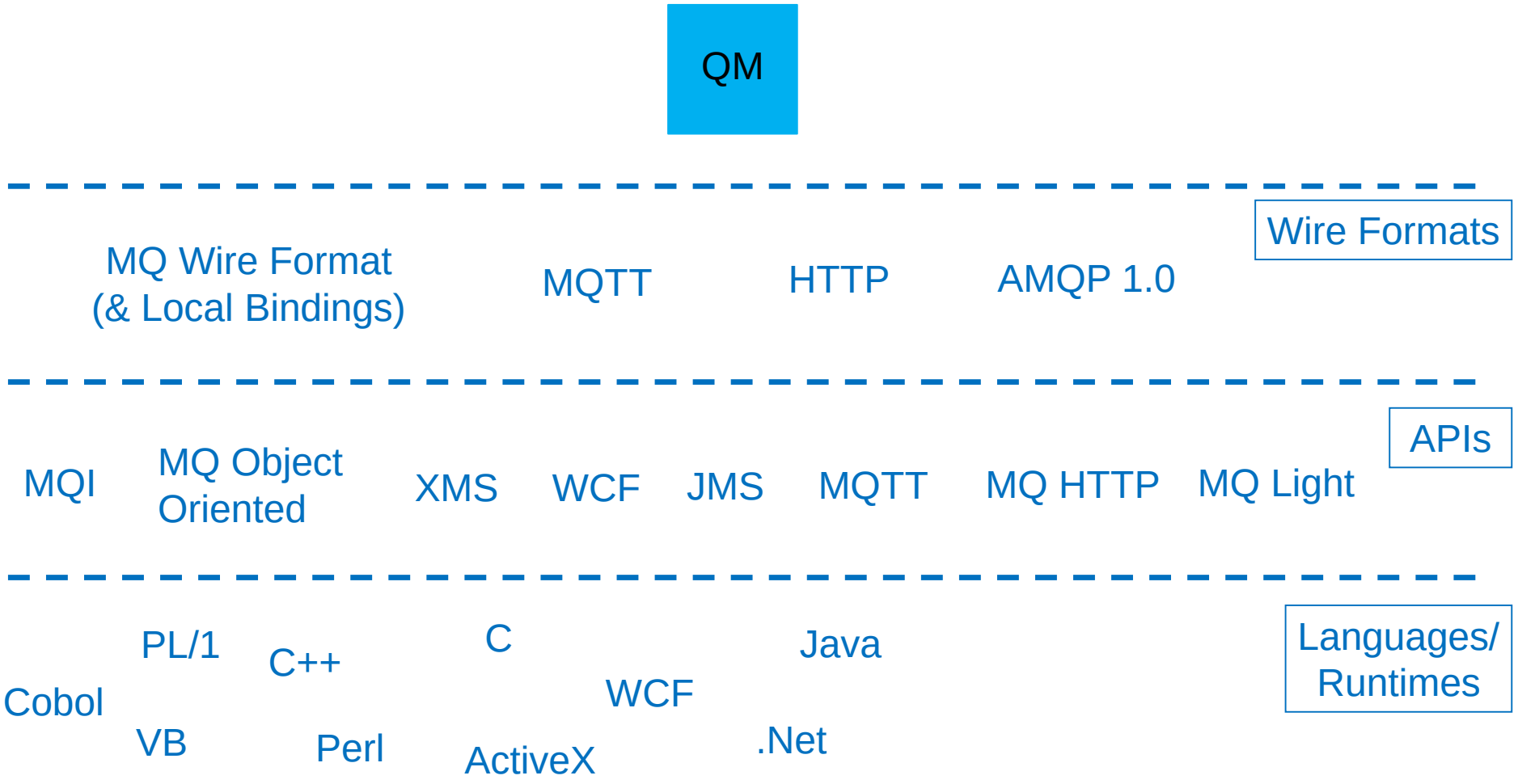
Languages, Wire Formats, and APIs



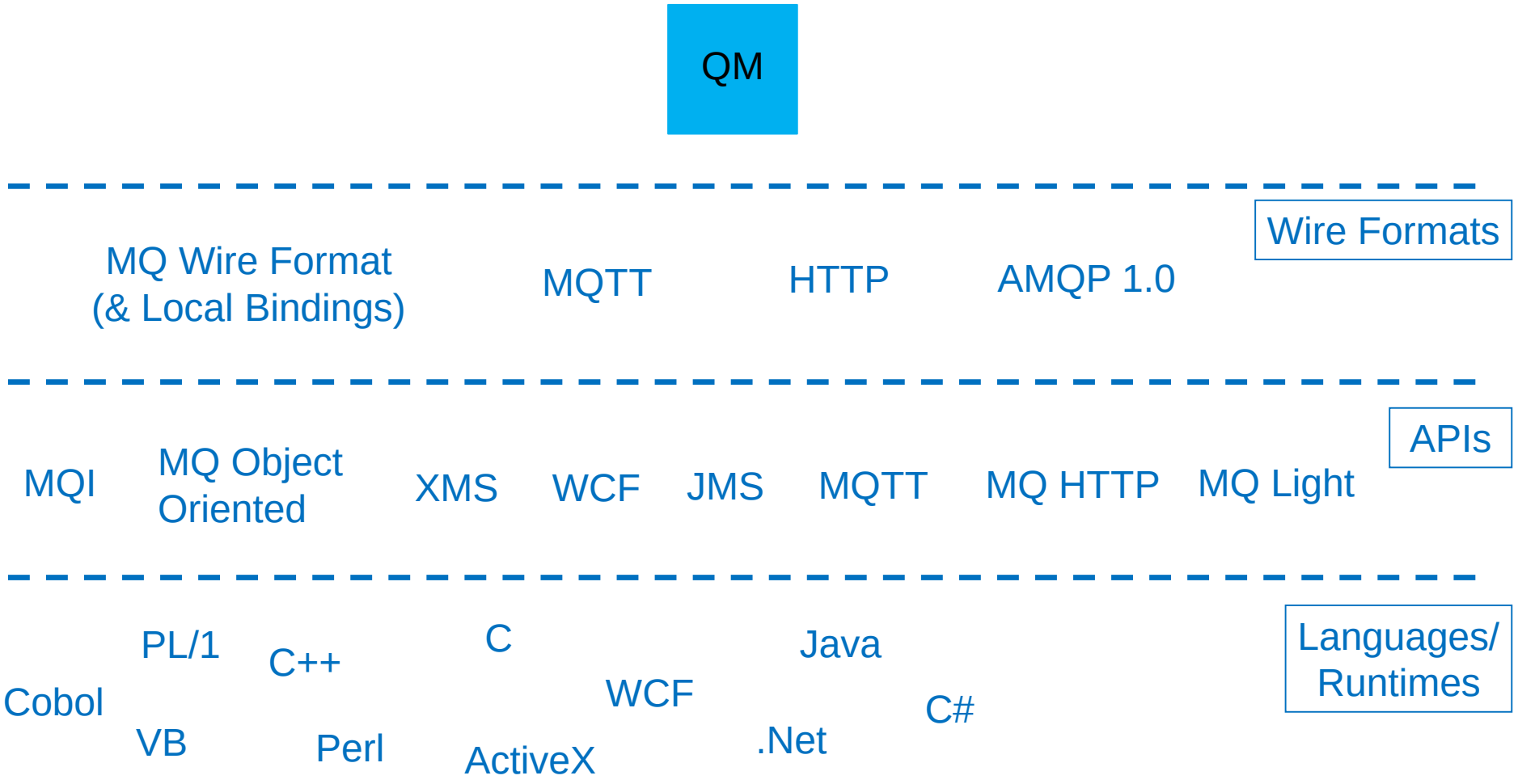
Languages, Wire Formats, and APIs



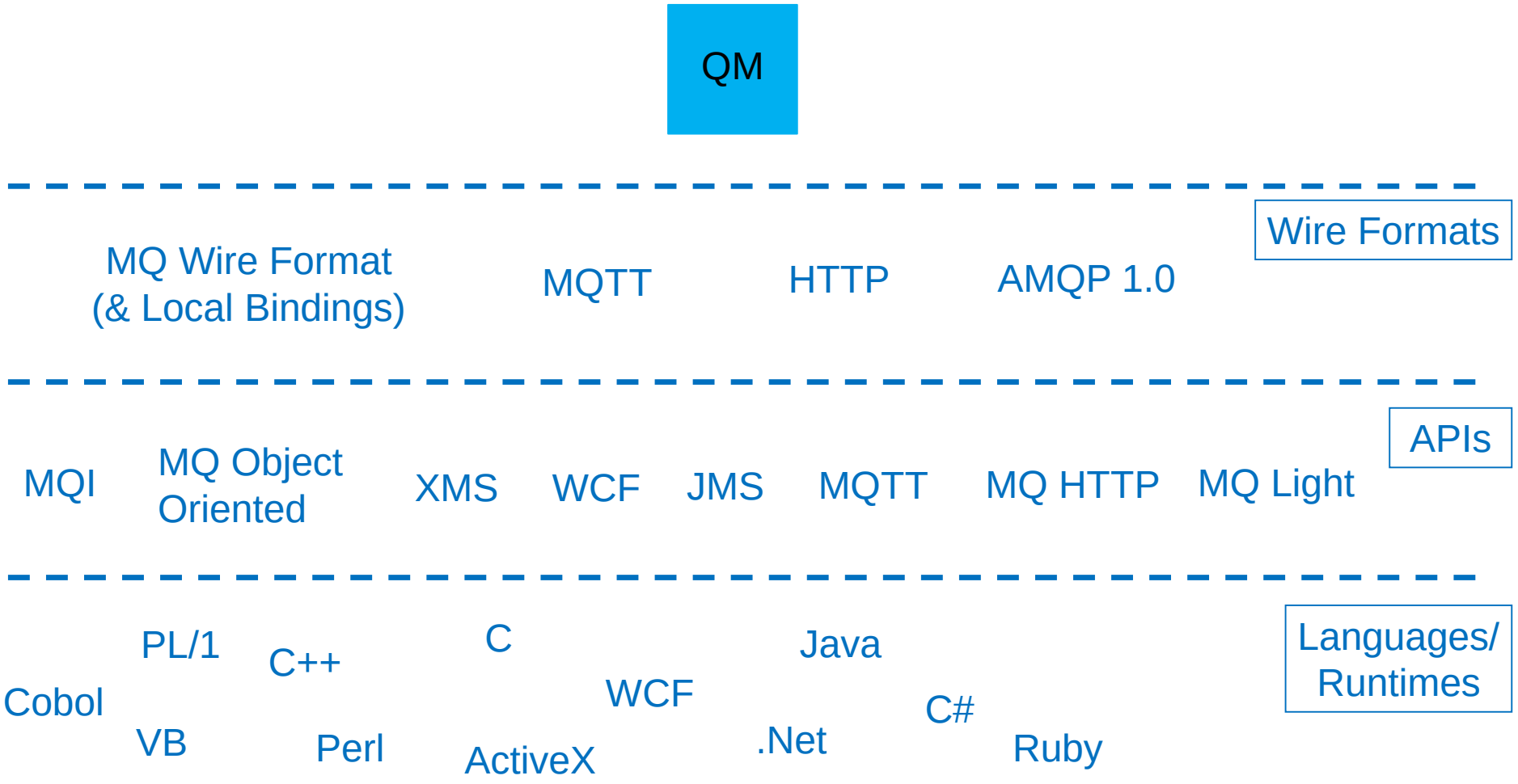
Languages, Wire Formats, and APIs



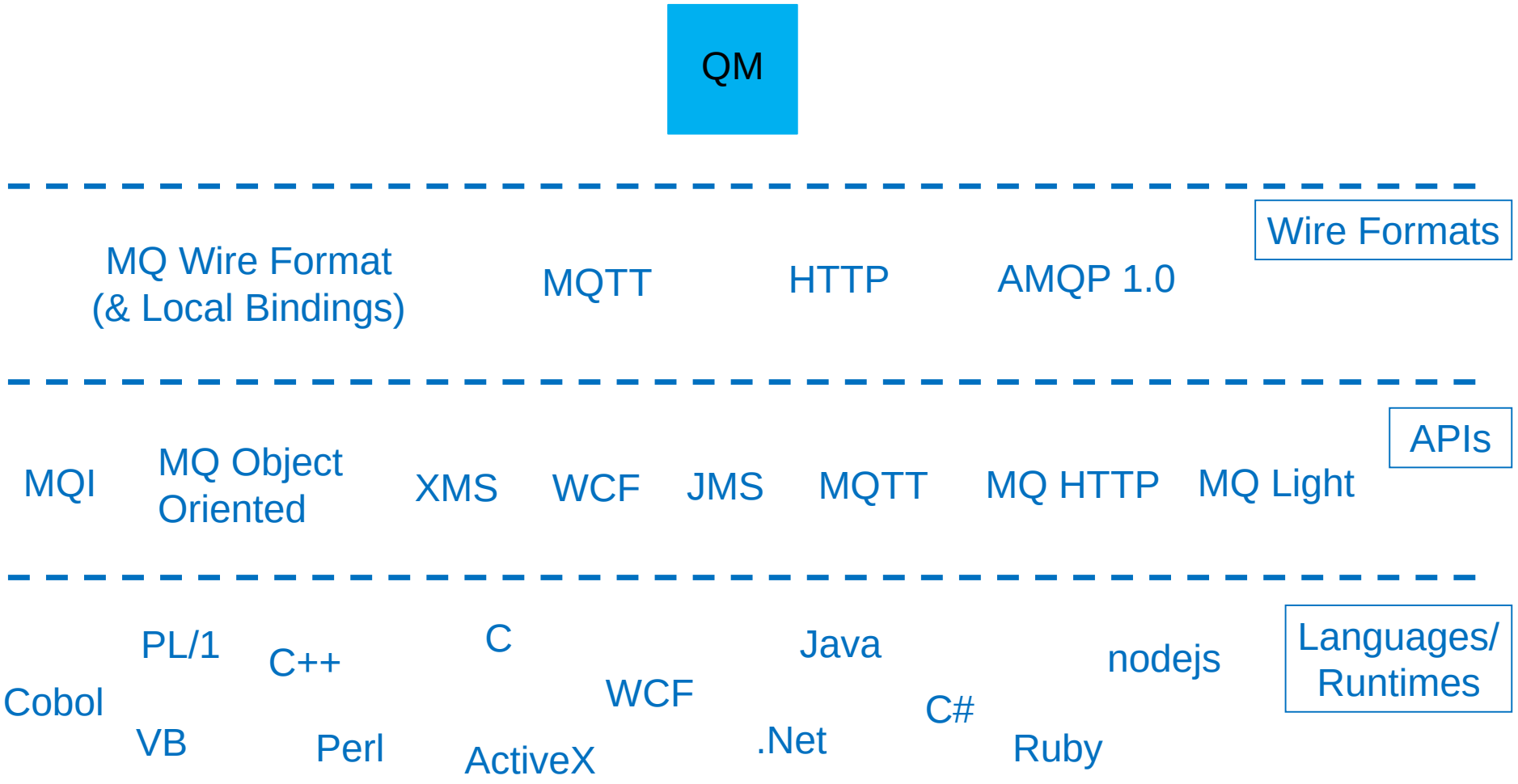
Languages, Wire Formats, and APIs



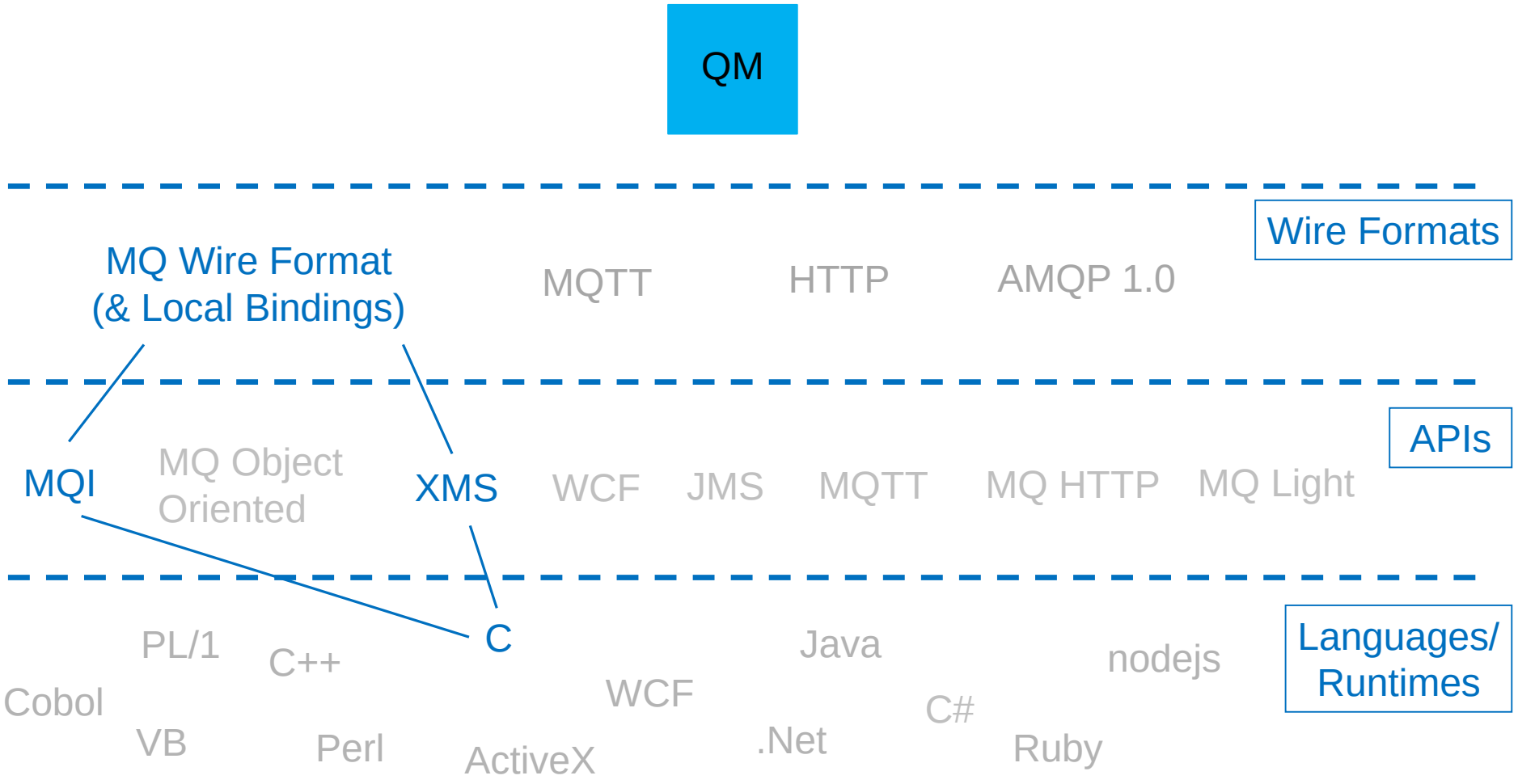
Languages, Wire Formats, and APIs



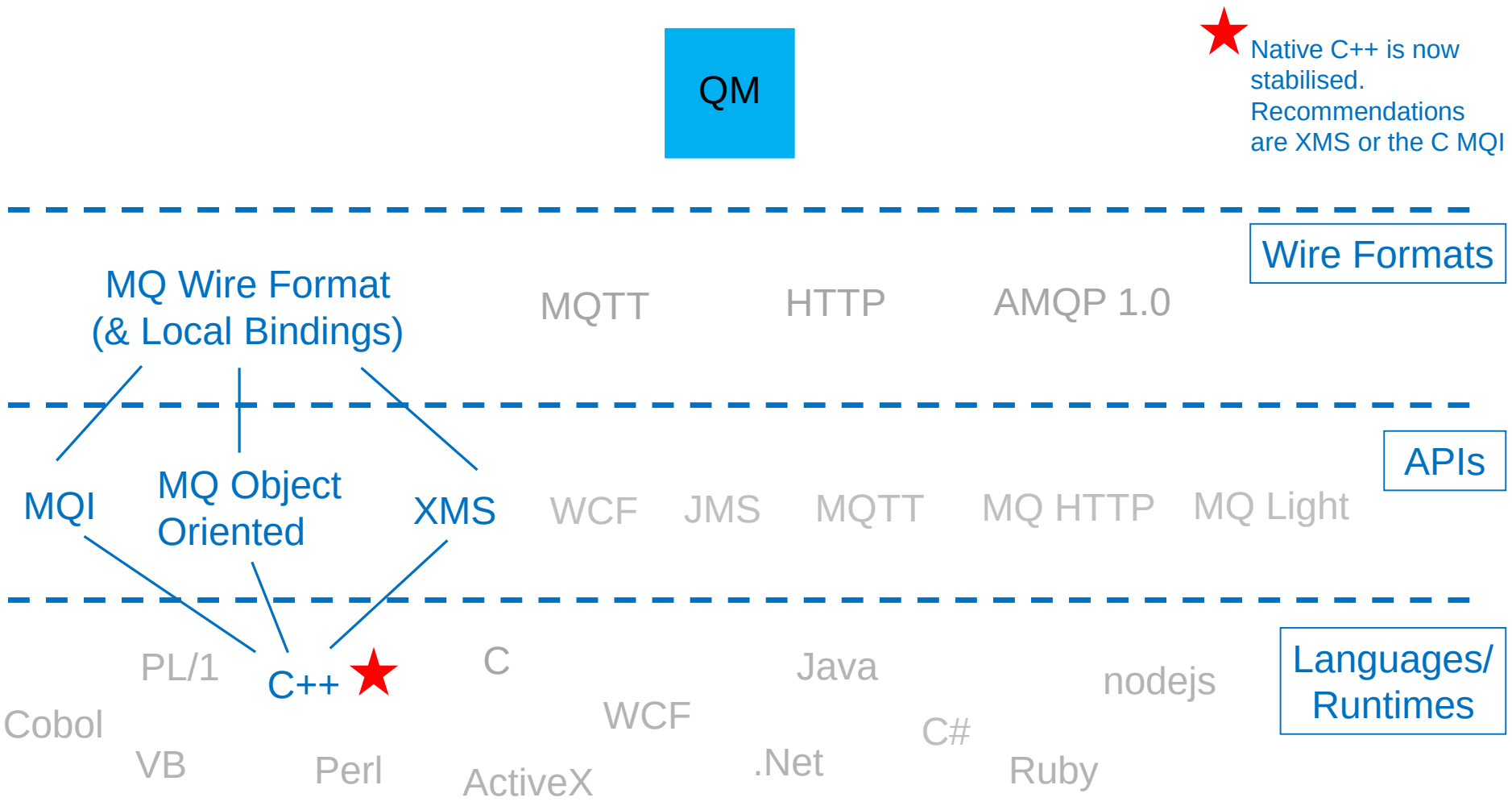
Languages, Wire Formats, and APIs



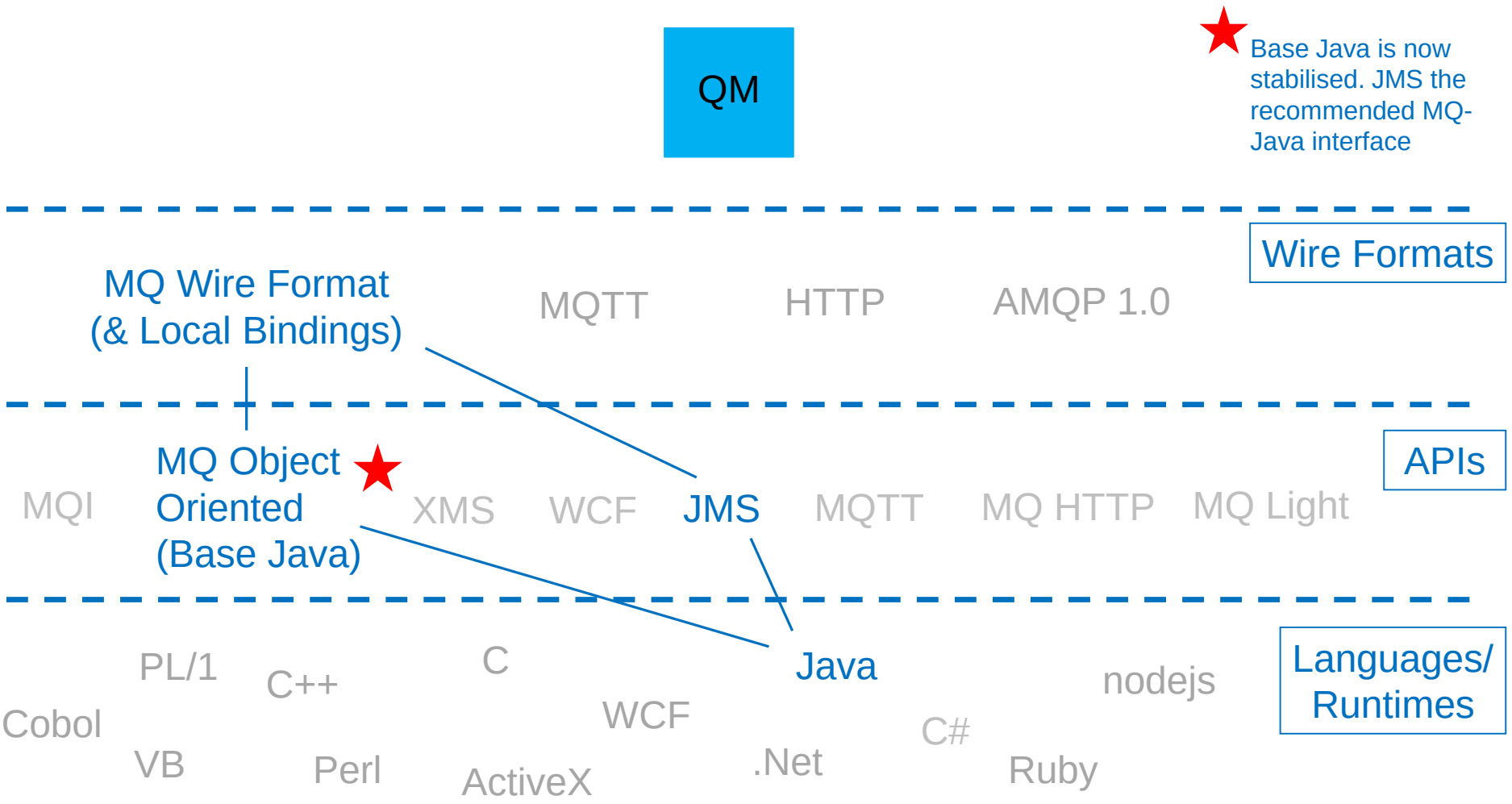
Languages, Wire Formats, and APIs



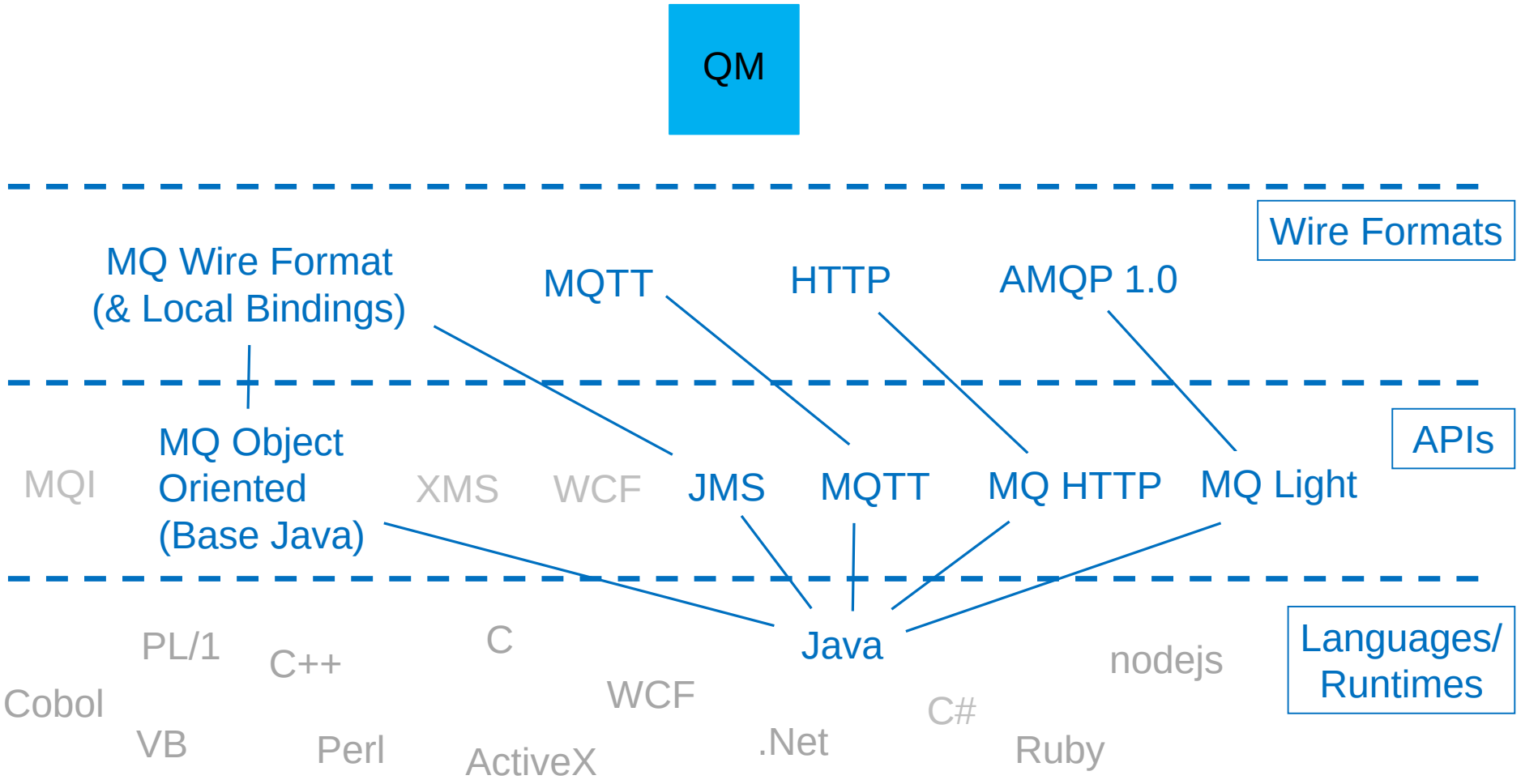
Languages, Wire Formats, and APIs



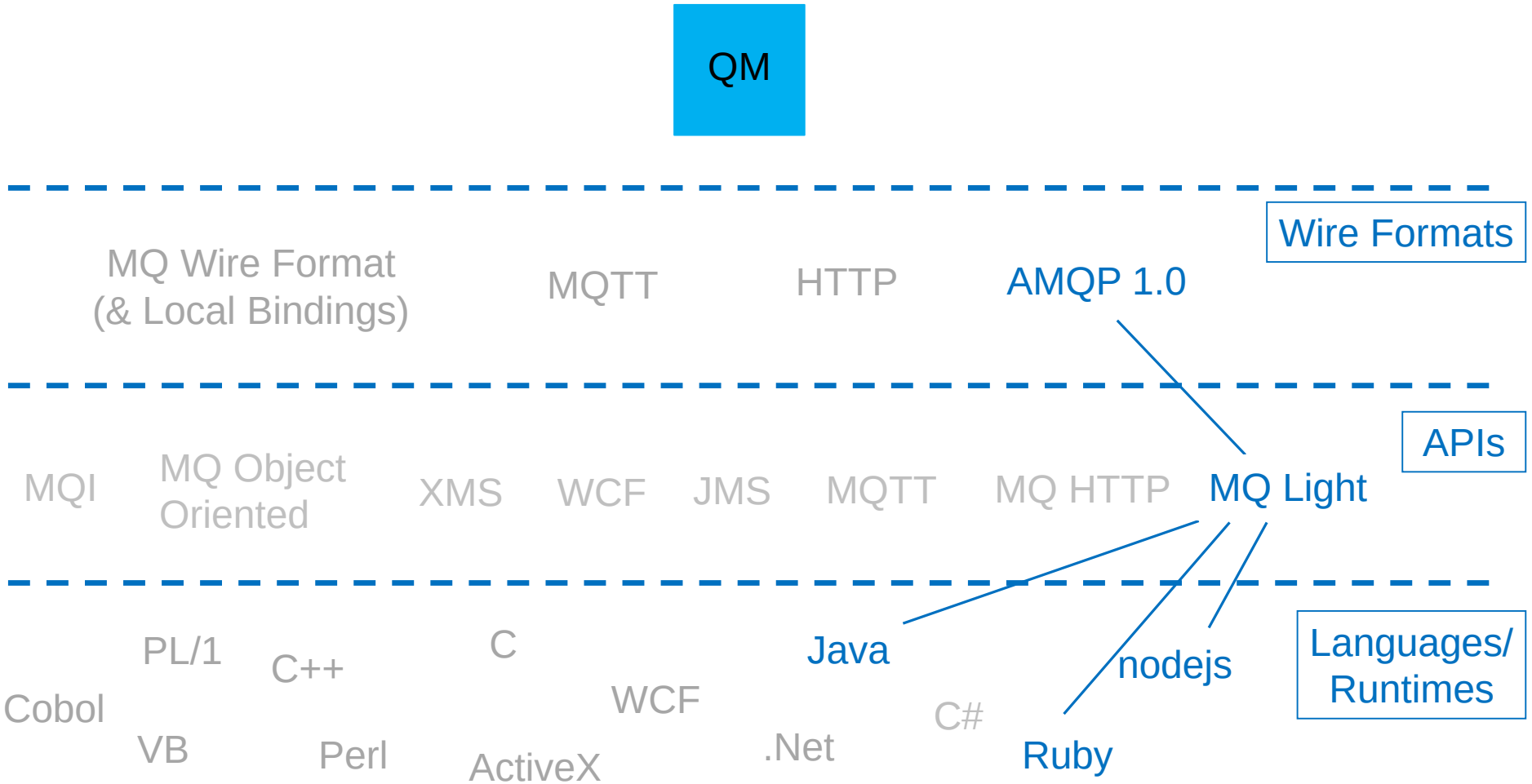
Languages, Wire Formats, and APIs



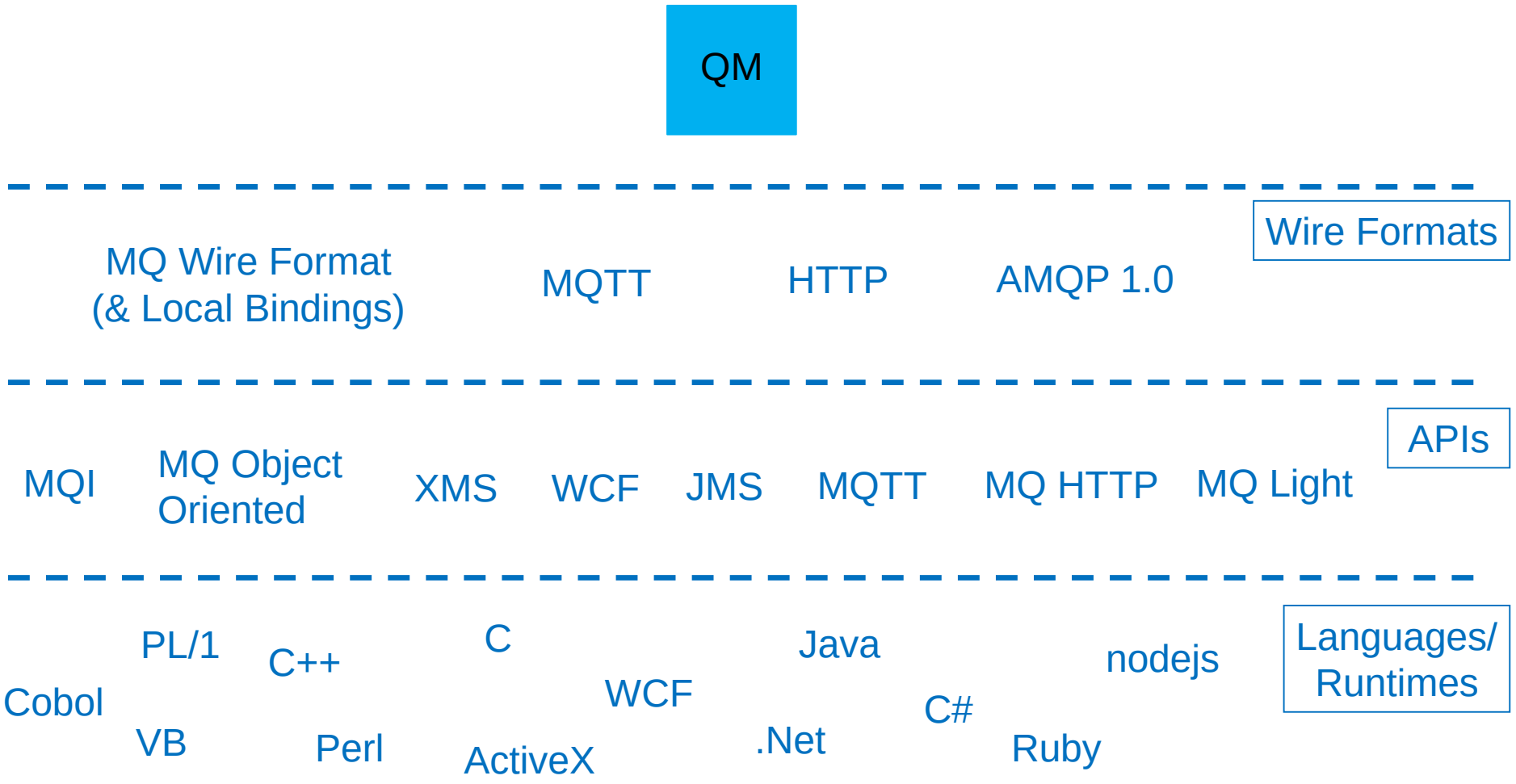
Languages, Wire Formats, and APIs



Languages, Wire Formats, and APIs



Languages, Wire Formats, and APIs



Agenda

- Languages, Wire Formats, and APIs
- **An Overview of C MQI, JMS, and MQ Light**
- Recap of the Key MQ Features
- Features supported in the different MQ APIs

Structures

MQMD – Message descriptor
MQOD – Object descriptor
MQSD – Subscription descriptor
MQPD – Property descriptor
MQCD – Client channel descriptor

MQCNO – Connect options
MQGMO – Get message options
MQPMO – Put message options
MQCSP – Security parameters

Verbs

MQCONN – Connect to QM
MQCONNX – Connect with opts
MQDISC – Disconnect from QM

MQOPEN – Open queue/topic
MQCLOSE – Close queue/topic

MQPUT - Put/publish a msg
MQPUT1 – Open/put/close
MQGET – Get message
MQSUB – Subscribe to topic
MQCTL – Control callbacks

MQBEGIN – Start global transaction
MQCMIT – Commit transaction
MQBACK – Rollback transaction
MQINQ – Inquire about object
MQSET – Set object attribute

Constants - Completion codes

MQCC_OK – Verb completed OK
MQCC_WARNING – Verb completed but with warnings (see MQRC)
MQCC_FAILED – Verb failed

Constants - Return codes (there are many of these), e.g:

MQRC_NONE – Nothing to check
MQRC_Q_MGR_QUIESCING – Can't do the action – QM is stopping
MQRC_NOT_AUTHORIZED – Can't do the action – you're not authorzed
MQRC_TRUNCATED_MSG_ACCEPTED – The message is too big for the buffer
MQRC_MSG_TOO_BIG_FOR_Q – Guess what this means…

Constants - Misc (there are **loads** of these), e.g:

MQIA_CURRENT_Q_DEPTH, MQCACH_CHANNEL_NAME,
MQCA_CLUSTER_NAME, …

C MQI Overview

mwhitehead@mrw-ubuntu-1604: ~

```
MQOD      od = {MQOD_DEFAULT};     /* Object Descriptor              */
MQMD      md = {MQMD_DEFAULT};     /* Message Descriptor             */
MQPMO    pmo = {MQPMO_DEFAULT};    /* put message options            */
MQCNO    cno = {MQCNO_DEFAULT};    /* connection options             */
MQCSP    csp = {MQCSP_DEFAULT};    /* security parameters            */

MQHCONN   Hcon;                    /* connection handle              */
MQHOBJ    Hobj;                    /* object handle                  */
MQLONG    O_options;               /* MQOPEN options                 */
MQLONG    C_options;               /* MQCLOSE options                */
MQLONG    CompCode;                /* completion code                */
MQLONG    OpenCode;                /* MQOPEN completion code         */
MQLONG    Reason;                  /* reason code                    */
MQLONG    CReason;                 /* reason code for MQCONNX        */
MQLONG    messlen;                 /* message length                 */
char      buffer[65535];           /* message buffer                 */
char      QMName[50];              /* queue manager name             */
char     *UserId;                  /* UserId for authentication      */
char      Password[MQ_CSP_PASSWORD_LENGTH + 1] = {0}; /* For auth   */
```

:

mwhitehead@mrw-ubuntu-1604: ~

```
MQCONNX(QMName,                 /* queue manager                */
        &cno,                   /* connection options           */
        &Hcon,                  /* connection handle            */
        &CompCode,              /* completion code              */
        &CReason);              /* reason code                  */

if (CompCode == MQCC_FAILED)
{
  // Something went wrong - print out the error and quit
}

if (CompCode == MQCC_WARNING)
{
  // Handle the warning, decide whether to continue
}
```

:

mwhitehead@mrw-ubuntu-1604: ~

```
O_options = MQOO_OUTPUT                 /* open queue for output     */
          | MQOO_FAIL_IF_QUIESCING;     /* but not if MQM stopping   */


MQOPEN(Hcon,                            /* connection handle              */
       &od,                             /* object descriptor for queue    */
       O_options,                       /* open options                   */
       &Hobj,                           /* object handle                  */
       &OpenCode,                       /* MQOPEN completion code         */
       &Reason);                        /* reason code                    */


if (OpenCode == MQCC_FAILED)
{
  // Something went wrong - print the error and quit
}

if (Reason != MQRC_NONE)
{
  // Something went wrong - print the error and quit
}
:
```

mwhitehead@mrw-ubuntu-1604: ~

```
if (fgets(buffer, sizeof(buffer), stdin) != NULL) /* read a message from keyboard input */
{
  messlen = (MQLONG)strlen(buffer);               /* get the length of the message */

  if (buffer[messlen-1] == '\n')                  /* remove the last newline character */
  {
    buffer[messlen-1]   = '\0';
    --messlen;
  }

  memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId) );

  MQPUT(Hcon,                           /* connection handle                */
        Hobj,                           /* object handle                    */
        &md,                            /* message descriptor               */
        &pmo,                           /* default options (datagram)       */
        messlen,                        /* message length                   */
        buffer,                         /* message buffer                   */
        &CompCode,                      /* completion code                  */
        &Reason);                       /* reason code                      */

  if (Reason != MQRC_NONE)
  {
    // Print out reason code and quit
  }
}
:
```

Objects

`ConnectionFactory` – from which connections are created
`Connection` – from which sessions are created
`Session` – from which producers, consumers, and destinations are created
`Destination` – to which messages are sent (**`Queue`** and **`Topic`** sub-types)

`MessageProducer` – with which messages are sent
`MessageConsumer` – with which messages are received
`QueueBrowser` – with which messages are browsed from a queue
`TextMessage/BytesMessage/ObjectMessage/MapMessage/StreamMessage`

`JMSContext` – JMS 2 – combination of connection and session
`JMSProducer` and **`JMSConsumer`** – JMS 2

Key Actions

`connection.start(…)` **`<-`** (not required in JMS 2)

```
session.createProducer(..)
producer.send(…)
message.acknowledge()
connection.close()
```

```
session.createConsumer(…)
consumer.receiveWithWait(…)
```

mwhitehead@mrw-ubuntu-1604: ~

```
Connection connection = null;
Session session = null;
Destination destination = null;
MessageProducer producer = null;

try {
  // Create a connection factory
  JmsFactoryFactory ff =
                  JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
  JmsConnectionFactory cf = ff.createConnectionFactory();

  // Set up connection properties
  cf.setStringProperty(WMQConstants.WMQ_HOST_NAME, host);
  cf.setIntProperty(WMQConstants.WMQ_PORT, port);
  cf.setStringProperty(WMQConstants.WMQ_CHANNEL, channel);
  cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, queueManagerName);
  cf.setStringProperty(WMQConstants.USERID, user);
  cf.setStringProperty(WMQConstants.PASSWORD, password);
  cf.setBooleanProperty(WMQConstants.USER_AUTHENTICATION_MQCSP, true);
} catch (JMSException e) {  // Handle exceptions here
}
:
```

mwhitehead@mrw-ubuntu-1604: ~

```
// Create JMS objects
connection = cf.createConnection();      // MQCONN is made under the covers

session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

if (isTopic) {
  destination = session.createTopic(destinationName);
}
else {
  destination = session.createQueue(destinationName);
}

// MQOPEN is performed under the covers
producer = session.createProducer(destination);
```

:

mwhitehead@mrw-ubuntu-1604: ~

```
try {
  BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

  do {
    System.out.print("Enter some text to be sent <ENTER to finish>:");
    System.out.flush();

    String line = in.readLine();
    if (line != null && line.trim().length() != 0) {

      TextMessage message = session.createTextMessage(line);
      producer.send(message);          // MQPUT happens under the covers

      System.out.println("Sent message:\n" + message);
    }
  } while (line != null);
} catch (Exception JMSException | IOException e) {
  // Handle exceptions here
}
```

:

Library Import

```
var mqlight = require('mqlight');
```

Client instantiation

```
var opts = {service: "amqp://localhost", id: "matt" };
var client = mqlight.createClient(opts);
```

Client-connected callback

```
client.on('started', function() {<code to run when the client has
connected>});
```

Client-error callback

```
client.on('error', function() {<code to run when the client has an
error>});
```

Message-received callback

```
client.on('message', function(data, delivery) {<code to run when a message is received>});
```

Malformed-message-received callback

```
client.on('malformed', function(data, delivery) {<code to run when a malformed message is received>});
```

Client subscribe, plus client-subscribed callback

```
client.subscribe(pattern, share, options, function(err, pattern) {<code to run when the subscribe returns>});
```

Send message, plus message-sent or message-not-sent callback

```
var buffered = !client.send(topic, body, options, callback);
```

mwhitehead@mrw-ubuntu-1604: ~

```
var mqlight = require('mqlight');      // Import the MQ Light client module
var nopt = require('nopt');
var uuid = require('node-uuid');
var fs = require('fs');

var opts = {                           // Define configuration options
  service: "amqp://localhost:5672",
  id: "client-01"
};

var client = mqlight.createClient(opts);   // Create the connection
```

:

mwhitehead@mrw-ubuntu-1604: ~

```
// Once we have connected we can start sending messages
client.on('started', function() {

  var sendNextMessage = function() { setImmediate(sendMessage); } };

  var sendMessage = function() {

    var options = { qos: mqlight.QOS_AT_LEAST_ONCE };

    var callback = function(err, topic, data, options) {
      if (err) {
        // The send failed - print the error and quit
      } else {
        // Nothing to do - send took place successfully
      }
    }

    client.send(topic, body, options, callback);
  }
}
```

Agenda

- A refresher on C, JMS, and MQ Light
- An Overview of C MQI, JMS, and MQ Light
- **Recap of the Key MQ Features**
- **Features supported in the different MQ APIs**

MQ features

- Point-to-Point messaging
- Publish/Subscribe messaging
- Shared Subscriptions
- Message persistence
- Message expiry
- Message grouping
- Message segmentation
- Message selection
- Local transactions
- XA/Global transactions
- At least once/at most once/exactly once delivery
- HA failover
- Message properties
- Asynchronous Consume
- Message browsing

Point-to-Point Messaging

MQ Wire Format	MQTT	HTTP	AMQP 1.0
✓			~

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
✓	✓		✓	~

Notes:

- ~ can be set indirectly

Examples

C/MQI →

```
strncpy(od.ObjectName, "Q1", (size_t)MQ_Q_NAME_LENGTH);

MQOPEN(Hcon, &od, O_options,&Hobj, &OpenCode, &Reason);
```

JMS →

```
destination = session.createQueue("Q1");

producer = session.createProducer(destination);
```

nodejs/
MQ Light →

```
client.send("Q1", body, options, callback);
```

Publish/Subscribe Messaging

MQ Wire Format	MQTT	HTTP	AMQP 1.0
✓	✓		~

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
✓	✓	✓	✓	✓

Notes:

- ~ can be set indirectly

Examples

C/MQI →

```
od.ObjectString.VSPtr="my/first/messaging/topic";
od.ObjectString.VSLength=(MQLONG)strlen(argv[1]);
od.ObjectType = MQOT_TOPIC;

MQOPEN(Hcon, &od, O_options,&Hobj, &OpenCode, &Reason);
```

JMS →

```
destination = session.createTopic("my/first/topic");

producer = session.createProducer(destination);
```

nodejs/ MQ Light →

```
client.send("my/first/messaging/topic", body, options,
callback);
```

Shared Subscriptions

MQ Wire Format	MQTT	HTTP	AMQP 1.0
✓			~

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
	JMS 2			✓

Notes:

- ~ can be set indirectly
- Not supported by C MQI
- Primarily added to MQ for JMS 2 support
- MQI has other ways of achieving similar pattern

Examples

C/MQI → N/A

JMS 2 →

```
MessageConsumer messageConsumer =
     session.createSharedConsumer(topic,"mySubscription");

MessageConsumer messageConsumer =
session.createSharedDurableConsumer(topic,"myDurableSub");
```

nodejs/ MQ Light →

```
client.subscribe(pattern, "myShare", options,
            function(err, pattern) { // Handle errors });
```

Message Persistence

MQ Wire Format	MQTT	HTTP	AMQP 1.0
✓	~		✓

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
✓	✓	~	✓	~

Notes:

- ~ can be set indirectly

Examples

C/MQI →

```
MQMD     md = {MQMD_DEFAULT};
         md.Persistence = MQPER_PERSISTENT;
…
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
```

JMS →

```
MessageProducer producer = session.createProducer(dest);

producer.setDeliveryMode(DeliveryMode.PERSISTENT);
```

nodejs/
MQ Light →

```
var options = { qos: mqlight.QOS_AT_LEAST_ONCE };

client.send(topic, body, options, callback);
```

Message Expiry

MQ Wire Format	MQTT	HTTP	AMQP 1.0
✓		✓	✓

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
✓	✓		✓	✓

Notes:

- MQ expiry value is ‘tenths of a second’

Examples

C/MQI →

```
MQMD     md = {MQMD_DEFAULT};
         md.Expiry = 3000;    // Expire in 5 minutes
…
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
```

JMS →

```
MessageProducer producer = session.createProducer(dest);

producer.setTimeToLive(300000);  // Expire in 5 minutes
```

nodejs/ MQ Light →

```
var options = { ttl: 300000 };  // Expire in 5 minutes

client.send(topic, body, options, callback);
```

Message Grouping

MQ Wire Format	MQTT	HTTP	AMQP 1.0
✓			✓

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
✓	✓			

Examples

C/MQI →

```
pmo.Options = MQPMO_LOGICAL_ORDER;
…
md.MsgFlags = MQMF_MSG_IN_GROUP;          // For all but last msg
md.MsgFlags = MQMF_LAST_MSG_IN_GROUP;     // For last msg
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
```

JMS →

```
message.setStringProperty("JMSXGroupID", groupId);
message.setIntProperty("JMSXGroupSeq", 1);
message.setBooleanProperty("JMS_IBM_Last_Msg_In_Group", true);
```

nodejs/ MQ Light → N/A

Message Segmentation

MQ Wire Format	MQTT	HTTP	AMQP 1.0
✓			

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
✓	Some langs			

Notes:

- Not supported in most OO APIs
- Not supported on MQ z/OS
- Really a feature of the queue manager, not the API/lang

Examples

C/MQI →

```
md.MsgFlags = MQMF_SEGMENTATION_ALLOWED;
memcpy(md.GroupId, MQGI_NONE, MQ_GROUP_ID_LENGTH);
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
```

```
gmo.Options = MQGMO_COMPLETE_MSG | <other options>;
MQGET(Hcon, Hobj, &md, &gmo, bufflen, buff, &msglen, &CC, &RC);
```

JMS → N/A

nodejs/
MQ Light → N/A

Message Selection (1)

MQ Wire Format	MQTT	HTTP	AMQP 1.0
✓			✓

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
✓	✓			

Notes:

- Use on MQOPEN to filter queued messages
- Applies to message properties

Examples

C/MQI →

```
od.SelectionString.VSPtr="Colour='red'";
od.SelectionString.VSLength=MQVS_NULL_TERMINATED;

MQOPEN(Hcon, &od, O_options, &Hobj, &CC, &RC);
```

JMS →

```
String selector = "NewsType = 'Sports'";

MessageConsumer consumer =
         session.createConsumer(queue, selector);
```

nodejs/
MQ Light → N/A

Message Selection (2)

MQ Wire Format	MQTT	HTTP	AMQP 1.0
✓			✓

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
✓	✓			

Notes:

- Use on MQSUB to filter published messages
- Applies to message properties

Examples

C/MQI →

```
sd.SelectionString.VSPtr="Colour='red'";
sd.SelectionString.VSLength=MQVS_NULL_TERMINATED;

MQSUB(Hcon, &sd, &Hobj, &Hsub, &CompCode, &Reason);
```

JMS →

```
String selector = "NewsType = 'Sports'";

MessageConsumer consumer =
         session.createConsumer(queue, selector);
```

nodejs/ MQ Light → N/A

Local transactions

- Ability to put or get multiple messages to/from a destination, either all of them or none at all
- Messages are hidden from other applications until the transaction is committed.
- If a failure occurs mid-sequence, the transaction is rolled back and none of the messages are delivered. Other applications are never aware of the messages.
- If everything completes successfully, the messages are made available to other applications.

Local transactions

MQ Wire Format	MQTT	HTTP	AMQP 1.0
✓			✓

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
✓	✓			

Notes:

- Only queue manager resources involved
- JEE containers manage JMS transactions for you

Examples

C/MQI →

```
pmo.Options = MQPMO_SYNCPOINT;
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
MQCMIT(Hcon, &CC, &RC);
```

```
gmo.Options = MQGMO_SYNCPOINT;
MQGET(Hcon, Hobj, &md, &gmo, bufflen, buff, &msglen, &CC, &RC);
MQCMIT(Hcon, &CC, &RC);
```

JMS →

```
session = connection.createSession(true,
                                   Session.AUTO_ACKNOWLEDGE);
// Get and put messages
session.commit();
```

nodejs/
MQ Light → N/A

Global/XA transactions

- **Similar to local transactions, but with multiple resources to update, e.g.**
  - Application consumes 5 related messages from a queue and writes them all to a database
  - If everything happens successfully, the messages are guaranteed to be removed from the queue and written to the database
  - If a failure occurs, the messages are rolled back to the queue and none are in the database
    - Potential for a transaction to go in-doubt needing manual resolution

Global/XA transactions

MQ Wire Format	MQTT	HTTP	AMQP 1.0
✓			

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
✓	✓			

Notes:

- Multiple resources coordinated together (e.g. MQ and Database)
- In this example the queue manager is coordinating the transaction
- Can be coordinated by DB/WAS etc.
- Can result in in-doubt transactions if a resource fails mid-transaction
- In JMS/JEE the container does the work
- MQ must be configured with the DB XA libraries

Examples

C/MQI →

```
MQBEGIN(Hcon, &bo, &CC, &RC);
// Read from database
pmo.Options = MQPMO_SYNCPOINT;
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
MQCMIT(Hcon, &CC, &RC);
```

JMS →

```
session = connection.createXASession();
// Read from database, put messages
xaResource = session.getXAResource();
xaResource.prepare(xaId, false);
xaResource.commit(xaId, false);
```

nodejs/ MQ Light → N/A

Quality of service

- **Describes the level of assurance given that a message will be delivered**

  - At most once delivery

  The message might not arrive, but it is guaranteed that you won't receive duplicates

  - At least once delivery

  The message will definitely arrive, but you may receive duplicates

  - Exactly once delivery

  The message is guaranteed to be delivered once and exactly once.

  Often considered to be the 'best' option but brings with it various additional costs and complexities

  Where are you measuring from?

At-most-once delivery

MQ Wire Format	MQTT	HTTP	AMQP 1.0
✓	✓	✓	✓

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
✓	✓	✓	✓	✓

Examples

C/MQI →

```
md.Persistence = MQPER_NOT_PERSISTENT;
…
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
```

JMS →

```
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
session = connection.createSession(transacted,
                                   Session.AUTO_ACKNOWLEDGE);
producer.send(destination, message)
```

nodejs/ MQ Light →

```
var options = { qos: mqlight.QOS_AT_MOST_ONCE };

client.send(topic, body, options, callback);
```

At-least-once delivery

MQ Wire Format	MQTT	HTTP	AMQP 1.0
✓	✓	✓	✓

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
✓	✓	✓	✓	✓

Notes:

- Can't always just turn at-least-once ON with a single parameter
- Relates to several areas:
  - Reliability of network protocol
  - Persistence of messages
  - Durability of subscriptions
  - ACK features in client libraries

Examples

C/MQI →

```
md.Persistence = MQPER_PERSISTENT;
pmo.Options = MQPMO_SYNCPOINT;
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
MQCMIT(Hcon, &CC, &RC);
```

JMS →

```
producer.setDeliveryMode(DeliveryMode.PERSISTENT);
session = connection.createSession(false,
                                   Session.CLIENT_ACKNOWLEDGE);
message.acknowledge();
```

nodejs/ MQ Light →

```
var options = { qos: mqlight.QOS_AT_LEAST_ONCE };

client.send(topic, body, options, callback);
```

Exactly-once delivery

MQ Wire Format	MQTT	HTTP	AMQP 1.0
✓	✓		

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
✓	✓	✓		

Notes:

- Requires XA/Global transactions
- Useful if messages **MUST** arrive and **MUSTN'T** be duplicated, but:
  - Network flow more complicated
  - Might require a human to resolve in-doubt transactions

Examples

C/MQI →

```
MQBEGIN(Hcon, &bo, &CC, &RC);
pmo.Options = MQPMO_SYNCPOINT;
MQPUT(Hcon, Hobj, &md, &pmo, messlen, buffer, &CC, &RC);
MQCMIT(Hcon, &CC, &RC);
```

JMS →

```
session = connection.createXASession();
// Put messages
xaResource = session.getXAResource();
xaResource.prepare(xaId, false);
xaResource.commit(xaId, false);
```

nodejs/ MQ Light → N/A

HA Failover/Auto-Reconnect

MQ Wire Format	MQTT	HTTP	AMQP 1.0
N/A	N/A	N/A	N/A

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
N/A	N/A	N/A	N/A	N/A

Client	Built-in HA reconnect?
C MQI Client	✓
JMS MQ Client	✓
Cobol MQI Client	✓
MQ Base Java Client	✗
C++ MQ Client	✓
XMS C/C++/.Net	✓
VB	✓
.Net WCF	✗
MQ Light Node/Ruby/Java/Python clients	✓

Notes:

- Really a feature of the client library rather than the wire format or API

More options

Message properties

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
✓	Not base java		✓	✓

Asynchronous consume

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
✓	Not base java or .Net/C#	some langs		✓

Message browsing

MQI	MQ OO	MQTT	MQ HTTP	MQ Light
✓	✓		✓	

Thank You - Questions?

Please Note

IBM’s statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM’s sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user’s job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Trademark Statement

- IBM and the IBM logo are trademarks of International Business Machines Corporation, registered in many jurisdictions. Other marks may be trademarks or registered trademarks of their respective owners.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
- Red Hat Enterprise Linux is a registered trademark of Red Hat, Inc. in the United States and other countries.
- Ubuntu and Canonical are registered trademarks of Canonical Ltd.
- SUSE and SLES are registered trademarks of SUSE LLC in the United States and other countries
- Mac and OS X are trademarks of Apple Inc., registered in the U.S. and other countries
- Other company, product and service names may be trademarks, registered marks or service marks of their respective owners.
- References in this publication to IBM products and services do not imply that IBM intends to make them available in all countries in which IBM operates.