IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Les Bases de ZeroMQ

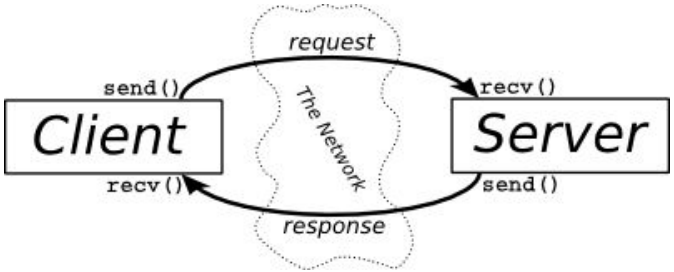
Feyza Hizem & Hamza Achibane

CHAPITRE 1
Les Bases de ZeroMQ

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

- ZeroMQ : Communication **asynchrone**
  Systèmes distribués/concurrents

- Il utilise les sockets réseaux de type : **Socket ZMQ**
  - une communication par message
  - la gestion d'une file d'attente
  - la gestion des reconnexions
  - protocole de transport des messages
  - le support de différents patterns de communication



Fonctions principales

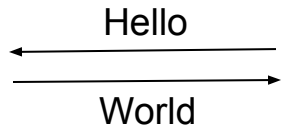
- **Context ( )**
- **socket (*type*) :** ZMQ_REQ, ZMQ_REP, ZMQ_ROUTER...
- **bind (*endpoint*)**
- **connect (*endpoint*)**
- **recv/send**

Types de communication

➔ Request-Reply

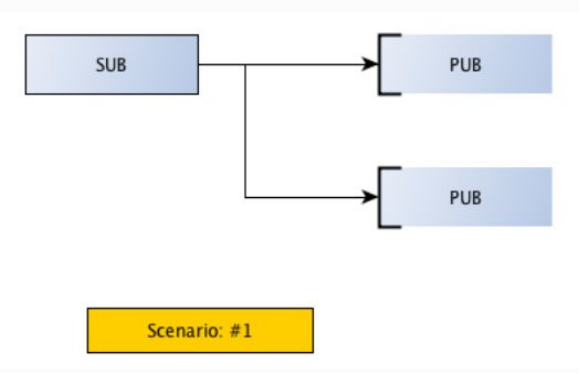
➔ Publish-Subscribe

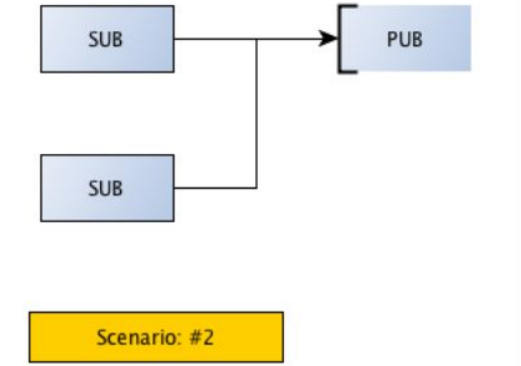
➔ Push-Pull




```
#
#   Hello World server in Python
#   Binds REP socket to tcp://*:5555
#   Expects b"Hello" from client, replies with b"World"
#

import time
import zmq

context = zmq.Context()
socket = context.socket(zmq.REP)
socket.bind("tcp://*:5555")

while True:
    #  Wait for next request from client
    message = socket.recv()
    print("Received request: %s" % message)

    #  Do some 'work'
    time.sleep(1)

    #  Send reply back to client
    socket.send(b"World")
```

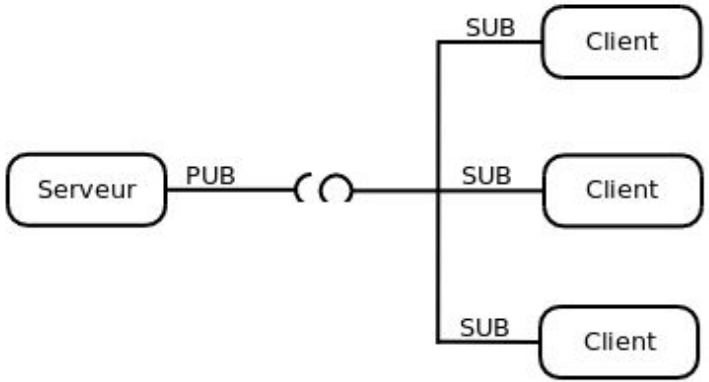
```
#
#   Hello World client in Python
#   Connects REQ socket to tcp://localhost:5555
#   Sends "Hello" to server, expects "World" back
#

import zmq

context = zmq.Context()

#  Socket to talk to server
print("Connecting to hello world server…")
socket = context.socket(zmq.REQ)
socket.connect("tcp://localhost:5555")

#  Do 10 requests, waiting each time for a response
for request in range(10):
    print("Sending request %s …" % request)
    socket.send(b"Hello")

    #  Get the reply.
    message = socket.recv()
    print("Received reply %s [ %s ]" % (request, message))
```





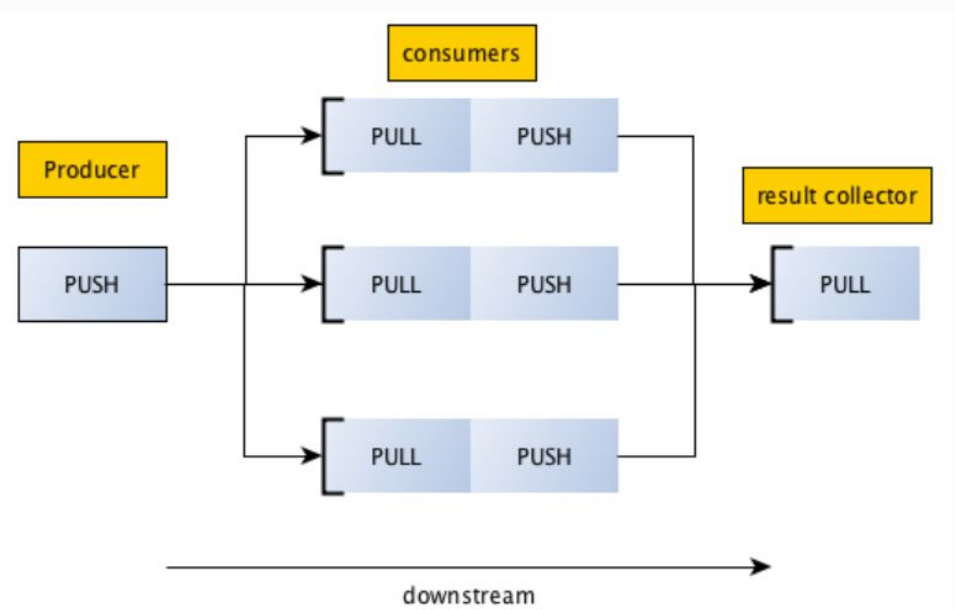
- Publishers ne programment pas l'envoie des messages directement aux Subscribers.
- scénario #2: schéma général plus connu : plusieurs subscribers s'abonnent aux messages/topics publiés par le publisher.

  scénario #1: plus intéressant, ZMQ.SUB peut se connecter à plusieurs ZMQ.PUB (publishers).
- Les messages des deux publishers sont entrelacés.



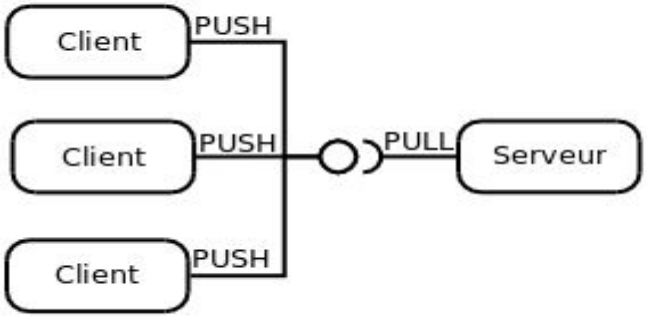

```
context = zmq.Context()
socket = context.socket(zmq.PUB)
socket.bind("tcp://*:%s" % port)
socket.send ("Hello")
```

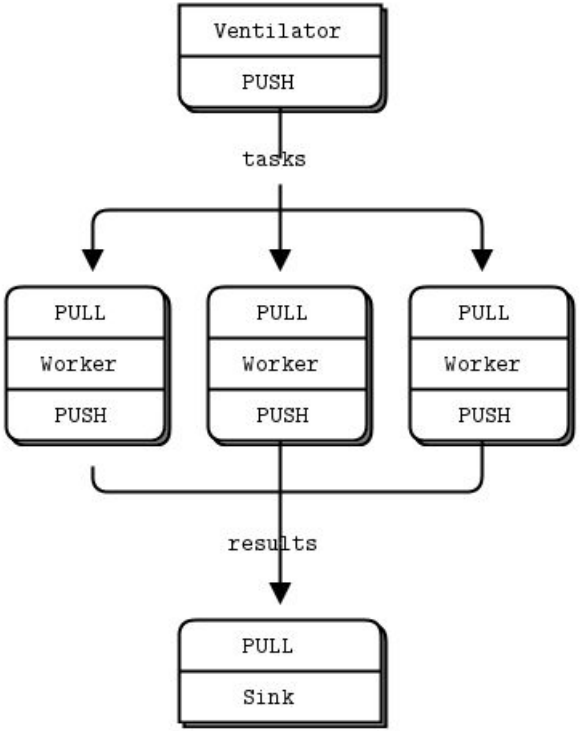
```
# Socket to talk to server
context = zmq.Context()
socket = context.socket(zmq.SUB)
socket.connect ("tcp://localhost:%s" % port)
string = socket.recv()
```



- Les sockets **Push** et **Pull** vous permettent de distribuer des messages à plusieurs workers, disposés en pipeline.
- Une socket **Push** distribue les messages envoyés à ses clients Pull de manière uniforme.
- Équivalent au modèle **producer/ consumer**.
- Les résultats calculés par consumer sont envoyés en aval **(downstream)** vers un autre socket d'extraction / consumer PULL.

3. Push-Pull : Exemples





3. Push-Pull : Le code en Python

PULL

```
import sys
import time
import zmq

context = zmq.Context()

# Socket to receive messages on
receiver = context.socket(zmq.PULL)
receiver.bind("tcp://*:5558")

while True:
    print receiver.recv()
```

PUSH

```
import zmq
import random
import time

context = zmq.Context()

# Socket with direct access to the sink: used to syncronize start of batch
sink = context.socket(zmq.PUSH)
sink.connect("tcp://localhost:5558")

# Initialize random number generator
random.seed()

while True:
    workload = random.randint(1, 100)
    sink.send(str(workload))
```