Messaging Service and Client Software

Lionel Cons – Massimo Paladin

2nd EMI Technical Forum - Garchin, 27th March 2012

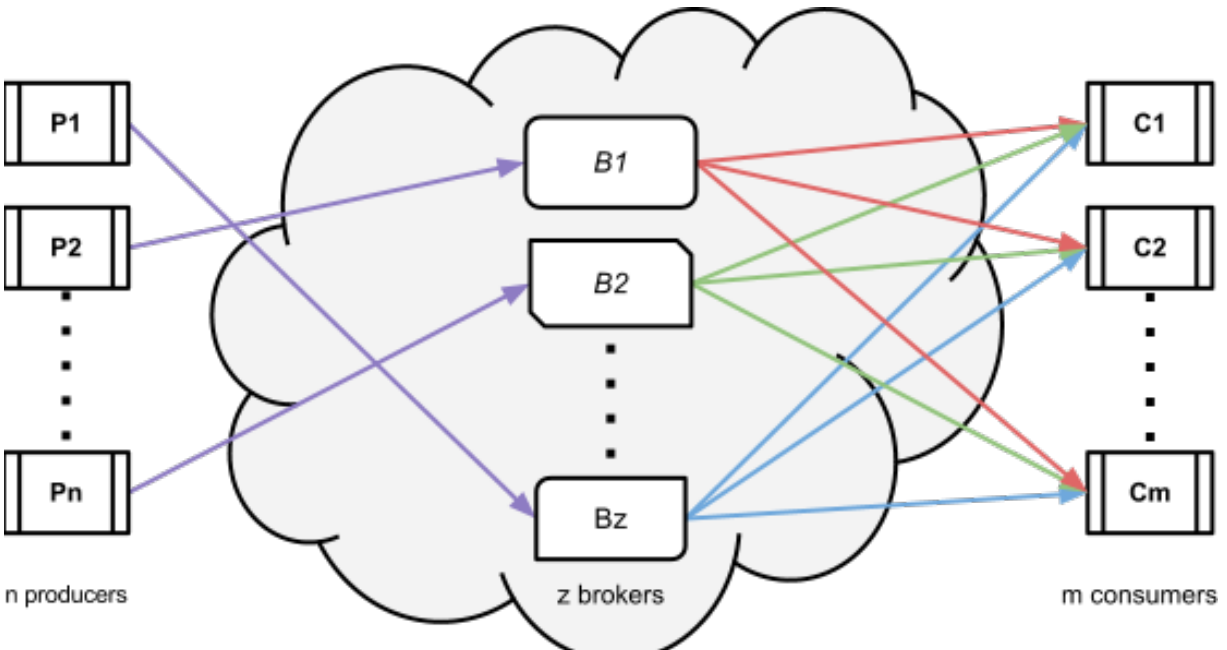
EMI is partially funded by the European Commission under Grant Agreement RI-261611

Outline

- Recommendations for Messaging Services
- From use cases to client software
- Recommended libraries and software

- Application dedicated services
  - Application isolation, misbehaving applications do not affect others
- Independent brokers
  - Heterogeneous products
  - Horizontal scalability
  - Easier management and operations
  - Can be easily used with load-balanced DNS
    - *Produce to any*
    - *Consume from all*

Dedicated Messaging Service of Independent Brokers



EMI INFSO-RI-261611

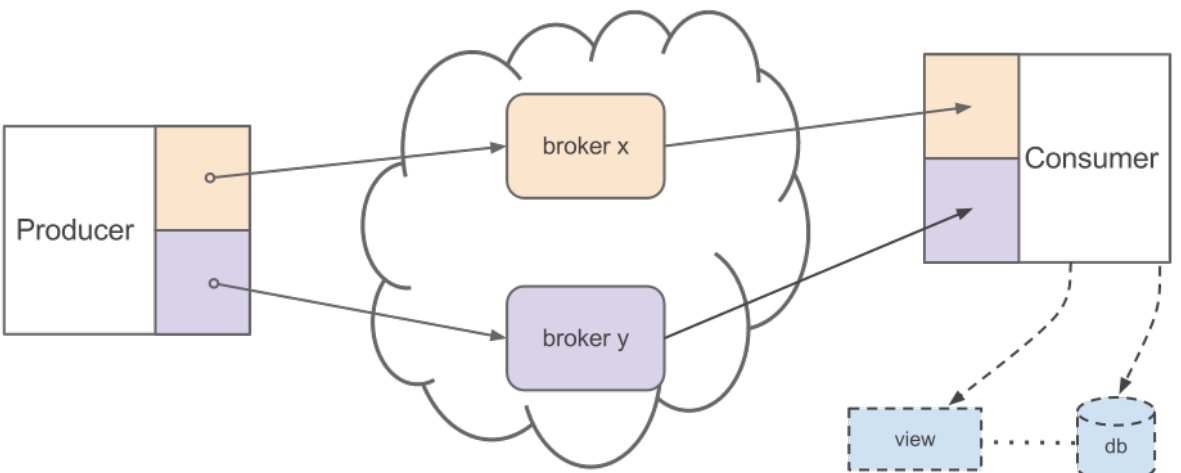
Software

- Messaging brokers are available
  - Recommendations available in the twiki page
- Protocol level client libraries available
  - Many alternatives
  - Many programming languages

	STOMP	AMQP	OpenWire
C / C++	x	x	x
Java	x	x	x
Perl	x	x	
Python	x	x	
Ruby	x	x	
…			

Using messaging

- Tough to make reliable usage of messaging:
  - Support of different protocols and programming languages leads to duplication of code
  - Error handling is not trivial!



EMI INFSO-RI-261611

How to solve this?

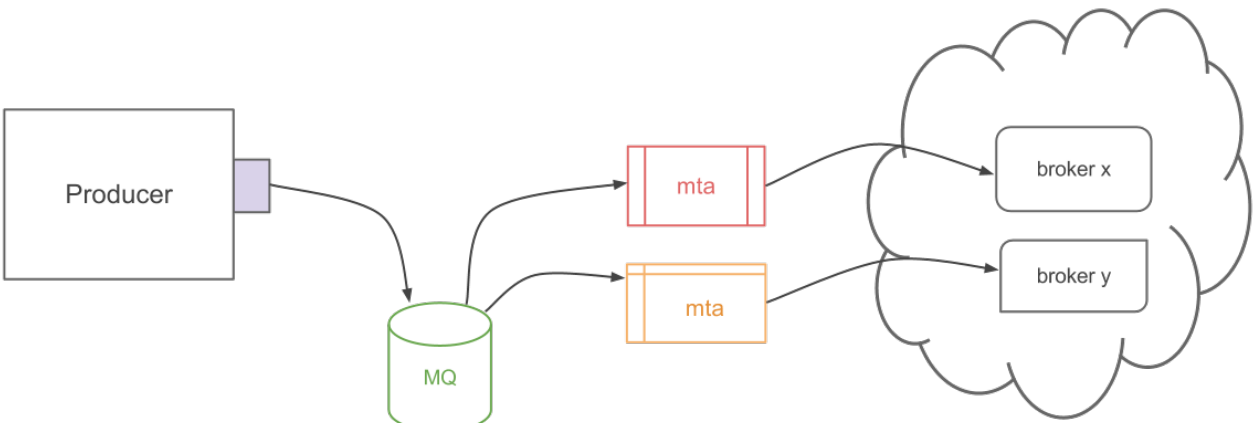
- What about lego bricks?
  - Small reusable components
  - Flexible when combined

- What are our bricks?

- Message Queue
  - File based message queue
  - Simple and robust API

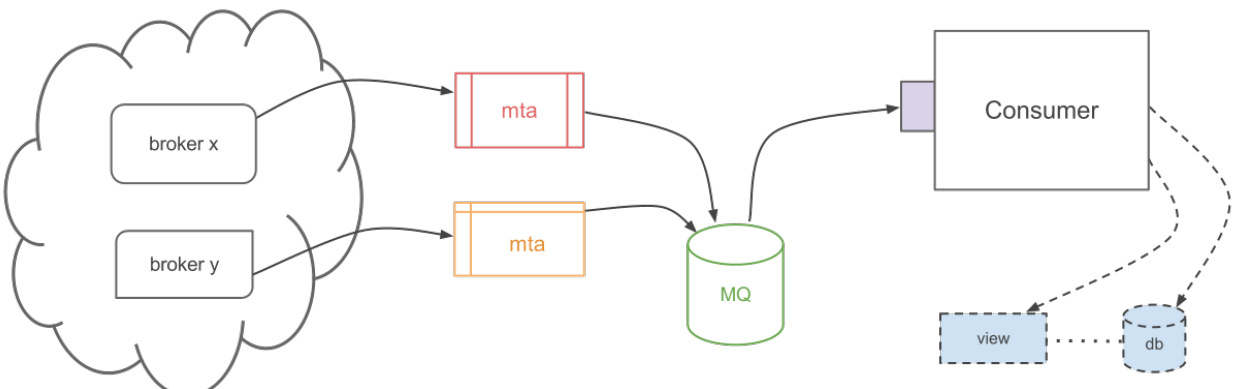
- Messaging transfer agent
  - Transfer messages between a broker and a message queue (all combinations)

Simplifying the producer



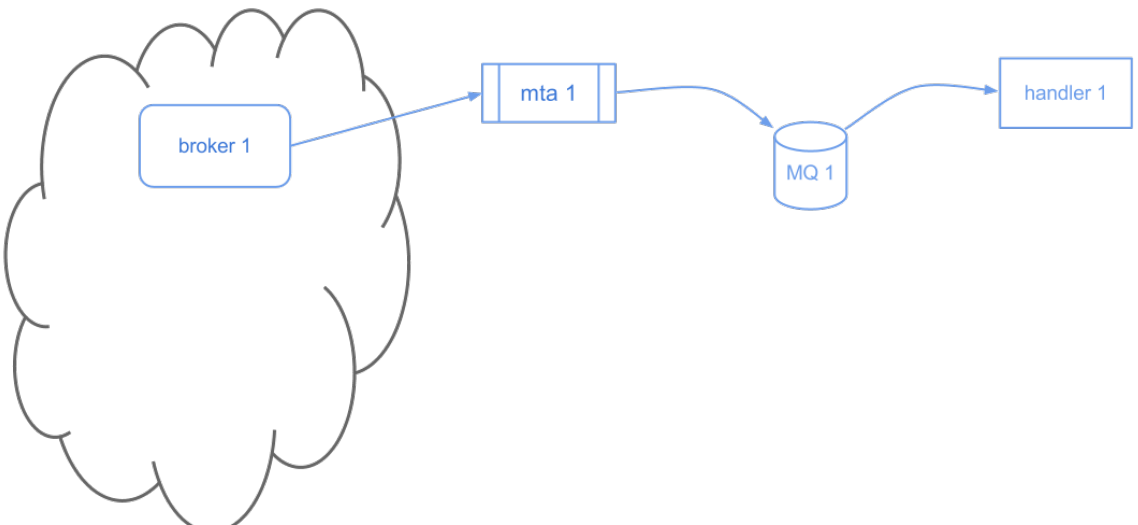
EMI INFSO-RI-261611

Simplifying the consumer



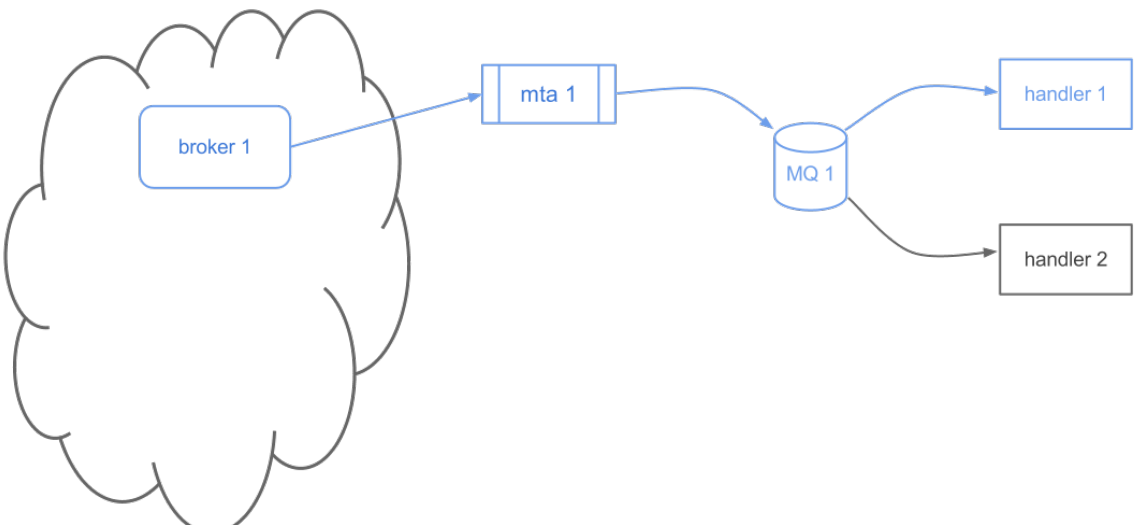
EMI INFSO-RI-261611

How can we scale the consumer side? 1/4



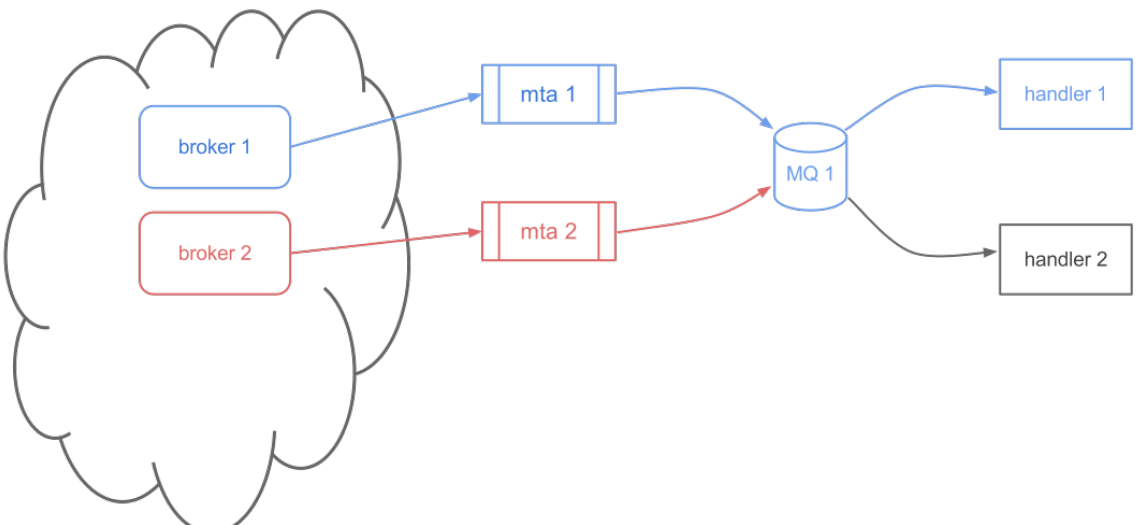
EMI INFSO-RI-261611

How can we scale the consumer side? 2/4



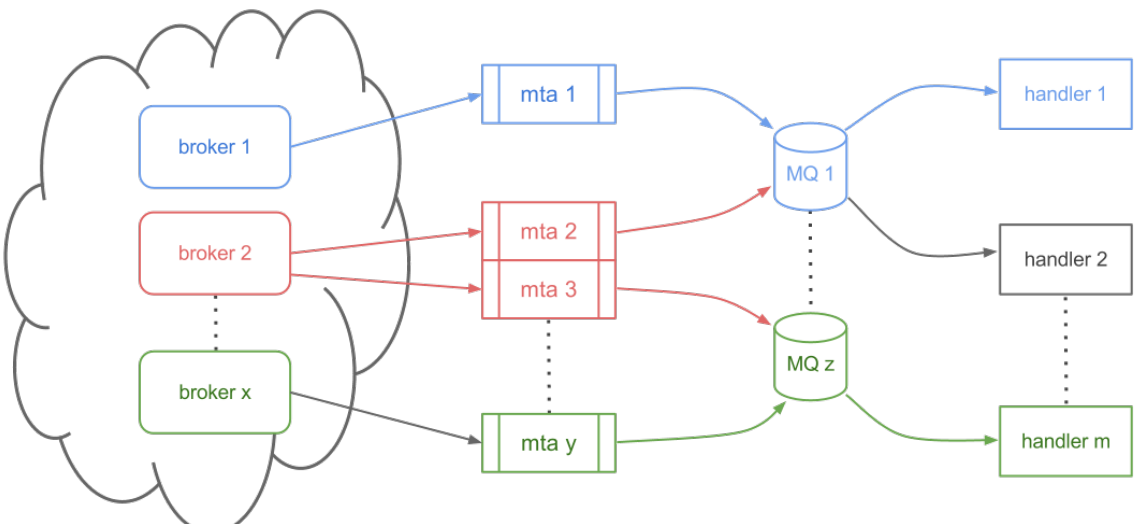
EMI INFSO-RI-261611

How can we scale the consumer side? 3/4

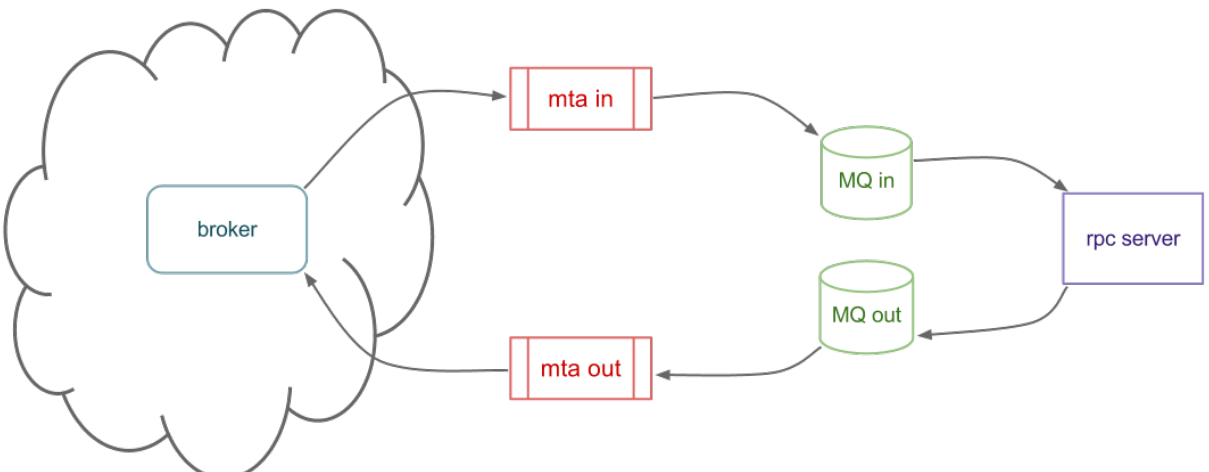


EMI INFSO-RI-261611

How can we scale the consumer side? 4/4



What about RPC pattern?



In practice 1/2

- All the basic blocks are available
- Producers and consumers need to know only about the Message Queue
  - Perl: perl-Messaging-Message + perl-Directory-Queue
  - Python: python-messaging + python-dirq
  - simple algorithm, easy to port to other programming languages

```
from messaging.message import Message
from messaging.queue.dqs import DQS

# create a message queue
mq = DQS(path = "/some/where")

# add a message to the queue
msg = Message(body = "hello world")
print("msg added as %s" % mq.add_message(msg))

# browse the queue
for name in mq:
    if mq.lock(name):
        msg = mq.get_message(name)
    # one could use mq.unlock(name) to only browse the queue...
    mq.remove(name)
```

In practice 2/2

- Messaging transfer agent
  - STOMP protocol: stompclt
  - AMQP protocol: amqpclt (in the future)

stompclt sender example

```
<incoming-queue>
 path = /var/spool/sender
</incoming-queue>

callback-code = <<EOF
$hdr{destination} = "/queue/myapp.data";
$hdr{persistent} = "true";
EOF

<outgoing-broker>
 uri = "stomp://broker.acme.com:6163"
</outgoing-broker>

pidfile = /var/run/sender.pid

loop = true
remove = true
```

stompclt receiver example

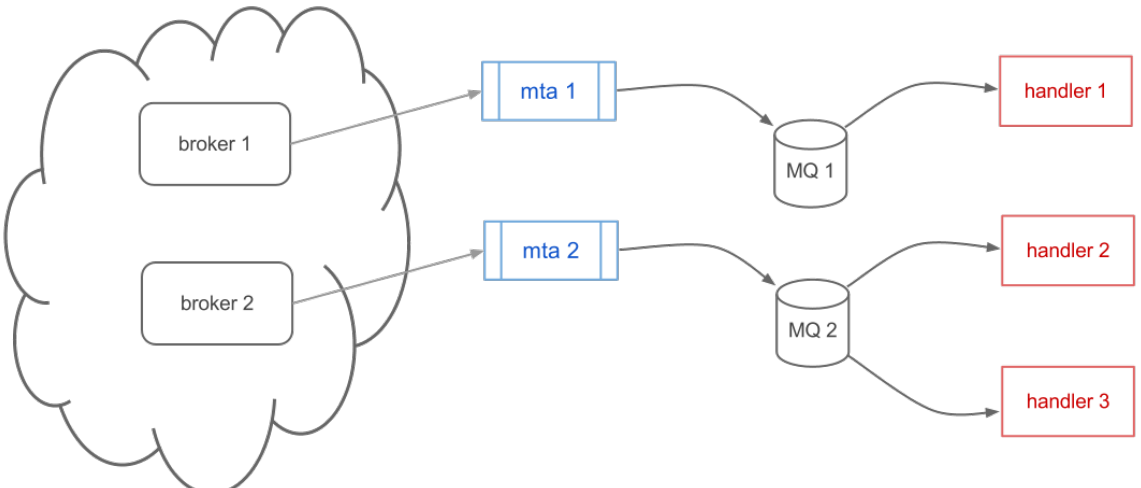
```
<incoming-broker>
 uri = "stomp://broker.acme.com:6163"
 <auth>
   scheme = plain
   name = receiver
   pass = secret
 </auth>
</incoming-broker>

<subscribe>
 destination = /queue/myapp.data
</subscribe>

<outgoing-queue>
 path = /var/spool/receiver
</outgoing-queue>

pidfile = /var/run/receiver.pid
```

How can we handle an elastic service?



- where components grow and shrink at need…

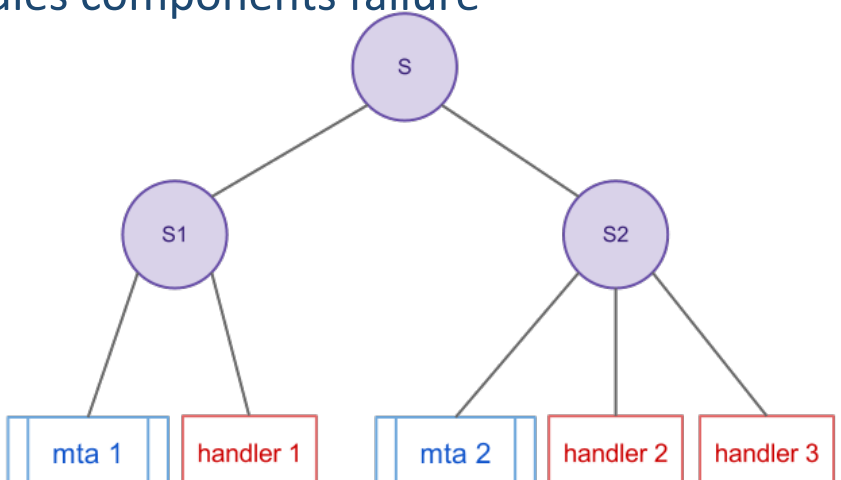
How do we assemble lego bricks?

- Lego bricks need to be combined to create manageable services
- Many components that can fail
  - Error handling is tricky
    - *What would you do if the connection dies? Do you try to reconnect? How many times?*
    - *…to be done for each service!*
- Well established solution
  - Let it fail. Have another process deal with it (Joe Armstrong thesis)

EMI INFSO-RI-261611

Supervisor concept by Erlang OTP

- We developed a daemon supervisor inspired by Erlang OTP
- It is called simplevisor
  - It can supervise hierarchies of services
  - Handles components failure



EMI INFSO-RI-261611

More about software

- The Messaging Product Team
  - Identified the reusable components
  - Improved the existing ones
  - Created the missing ones
- Most of the components are available
  - Production ready
  - Testing
- Already in EPEL or will be part of it

EMI INFSO-RI-261611

Contact / Feedback

- If interested in using messaging or want to provide feedback
- Visit the twiki page
  - https://twiki.cern.ch/twiki/bin/view/EMI/EMIMessaging (short: http://goo.gl/JZ8o5)
- Write to the mailing list

  emi-jra1-messaging@eu-emi.eu

EMI INFSO-RI-261611

EUROPEAN MIDDLEWARE INITIATIVE

Thank you!

EMI is partially funded by the European Commission under Grant Agreement RI-261611