Tala - Distributed Computing

Jeff McGough

Department of Computer Science and Engineering
South Dakota School of Mines and Technology
Rapid City, SD 57701, USA

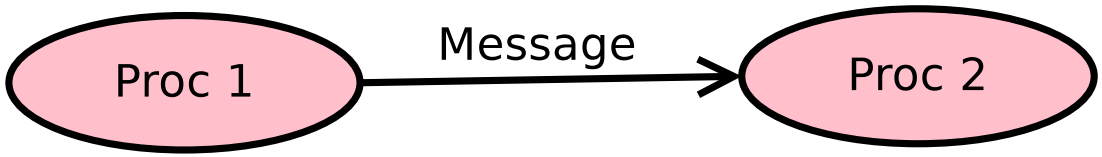
September 8, 2020

Tala

Distributed Communications

Problem

What if you want to send a message between two processes:



There are a lot of approaches to this problem. File system (files and pipes) are easy, but not efficient. For processes on different address spaces, using shared memory does not work. And neither files or SHM will work on when using completely separate systems.

Using sockets is the popular answer.

Sockets

There are several issues using sockets

- You need to know the location of the process. IP address and Port must be known.
- There must be some synchronization, buffering or queuing between the processes.
- Your communication pattern needs to be client server where the sending process is the client and receiving process is the server. It is asymmetrical in nature.
- Socket libraries support low level communications and not the communication patterns we would like such as Publish/Subscribe.
- In all but the most simple applications, it requires a threaded design.

Communication Patterns

There are two patterns we will start with:

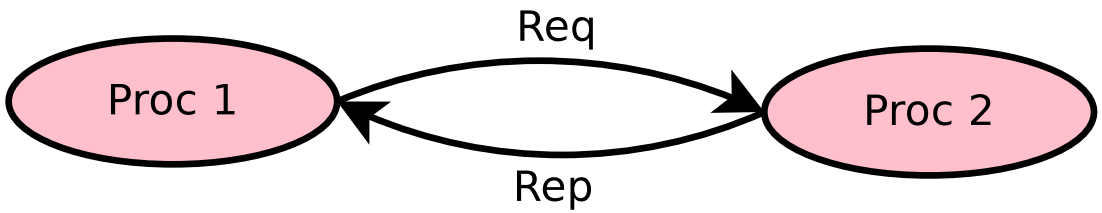
- Request - Reply : ReqRep
- Publish - Subscribe : PubSub

There are other patterns but for getting started on this project these two will suffice.

REQREP

Request-Reply is a more traditional client server communication pattern.

It is one to one direct communication.



A client will send a message to the server requesting some information. That is returned in the reply. It is a client initiated exchange.

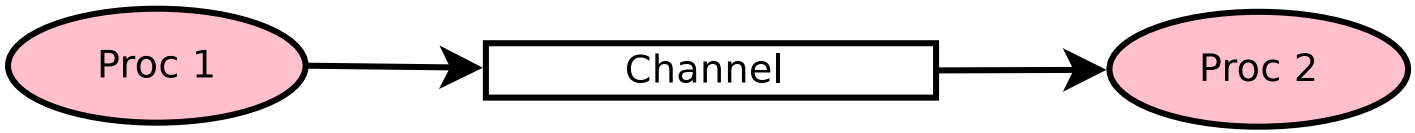
This will be useful for one-off types of communication.

PUBSUB

Publish-Subscribe is our main communication pattern for this project.

Formally, it is a buffered, indirect, many to many communication pattern.

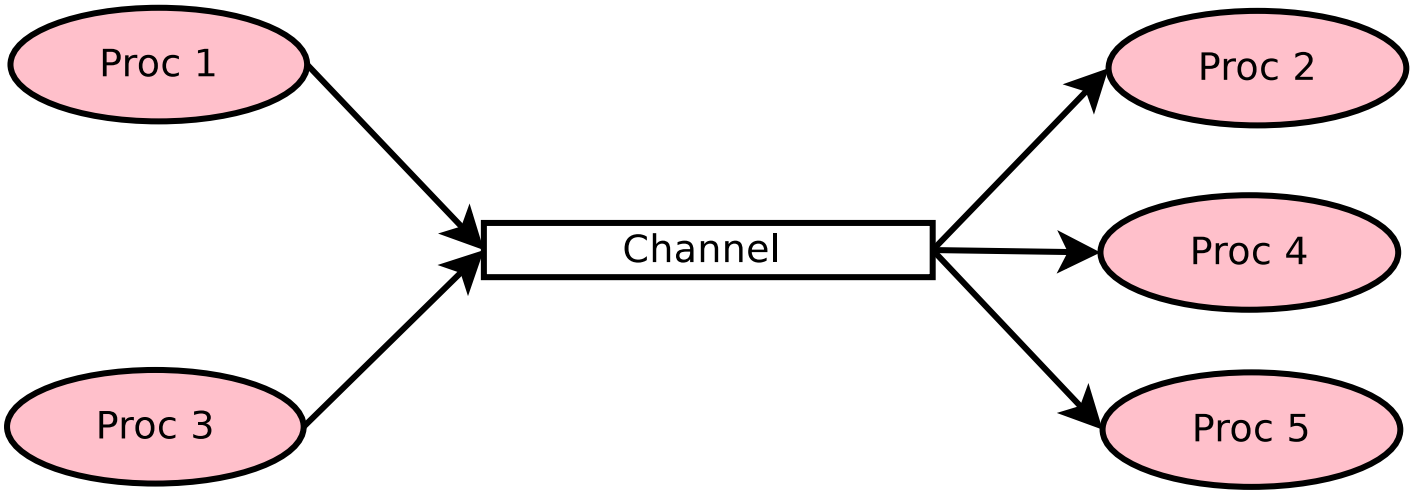
In PubSub, a process will *publish* a message on a named channel or *topic*. This means that the message is placed in a message queue which is normally a FIFO.



A process that is subscribed can receive the message off of the other end of the topic (FIFO). Where this FIFO is actually located is to be determined.

PUBSUB

Any number of processes can publish on the topic. These messages are queued up in the FIFO.



Any number of processes may subscribe to the topic.

More details

I should not need to know where my other processes are running.

I should only need to publish to a topic or subscribe to a topic;.

Topics can carry multiple messages and have a message type associated with them.

Tala

A fast distributed message system for remote interprocess communication.

Usage

The following graphical user story describes a common use case.

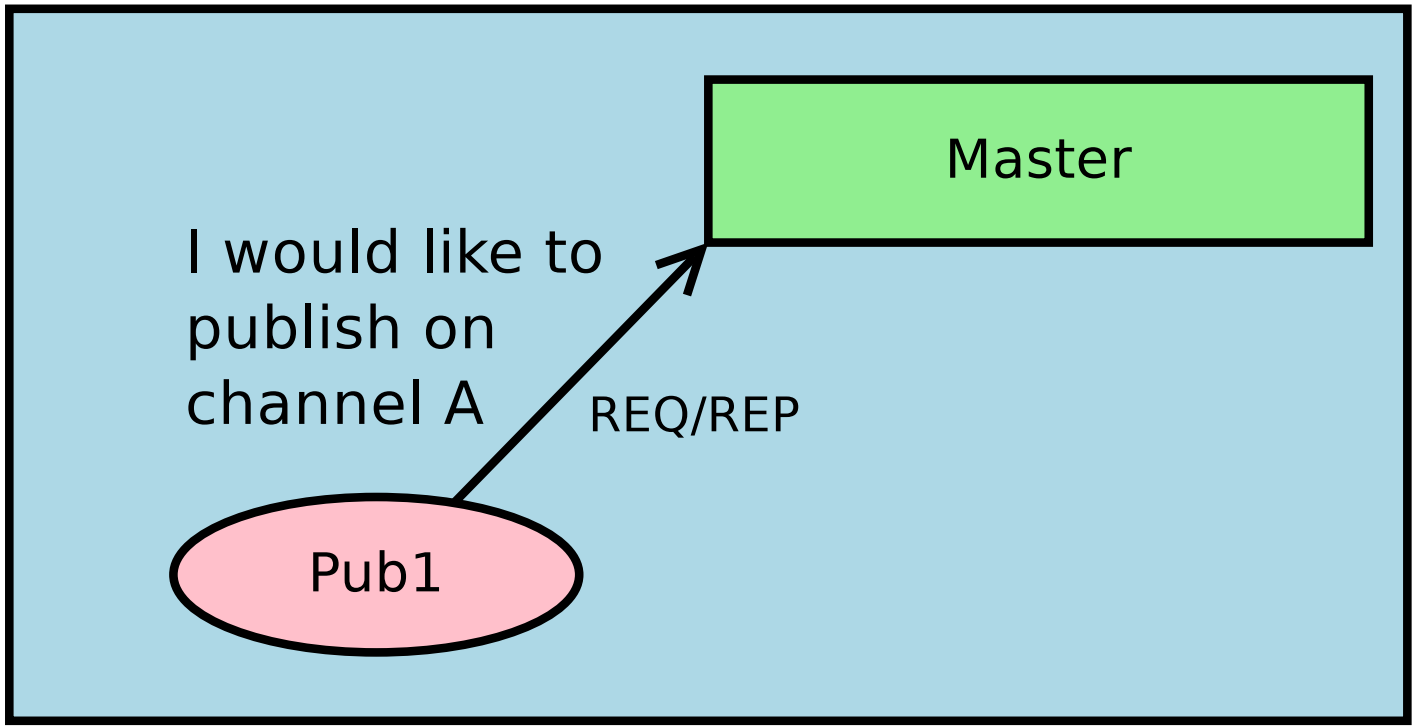
Although not all aspects of Tala are represented here, this use case is essential to understand before moving on.

In this example, we look at a design that has the FIFO or topic as a standalone or separate entity. The reasoning is that if the FIFO is bound to either publisher or subscriber, it might limit the number of publishers or subscribers (this will get us started for now).

This is a design decision. The following "User Stories" are to give a flavor of how the tool is used. How the directory service is constructed or the details on communication topics are not defined.

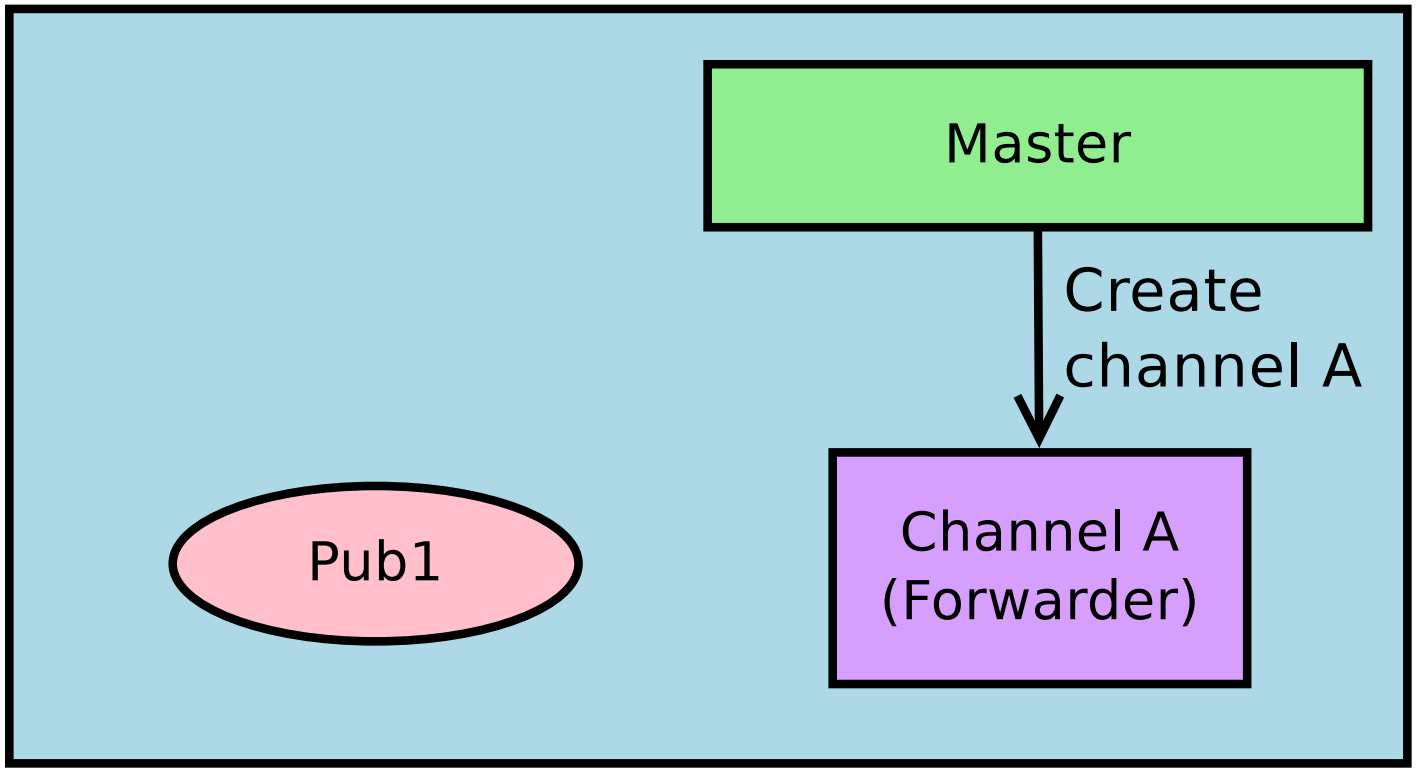
User Story

Request to connect to a specific topic. This can be built into a "topic open" type function call. Master process described later.



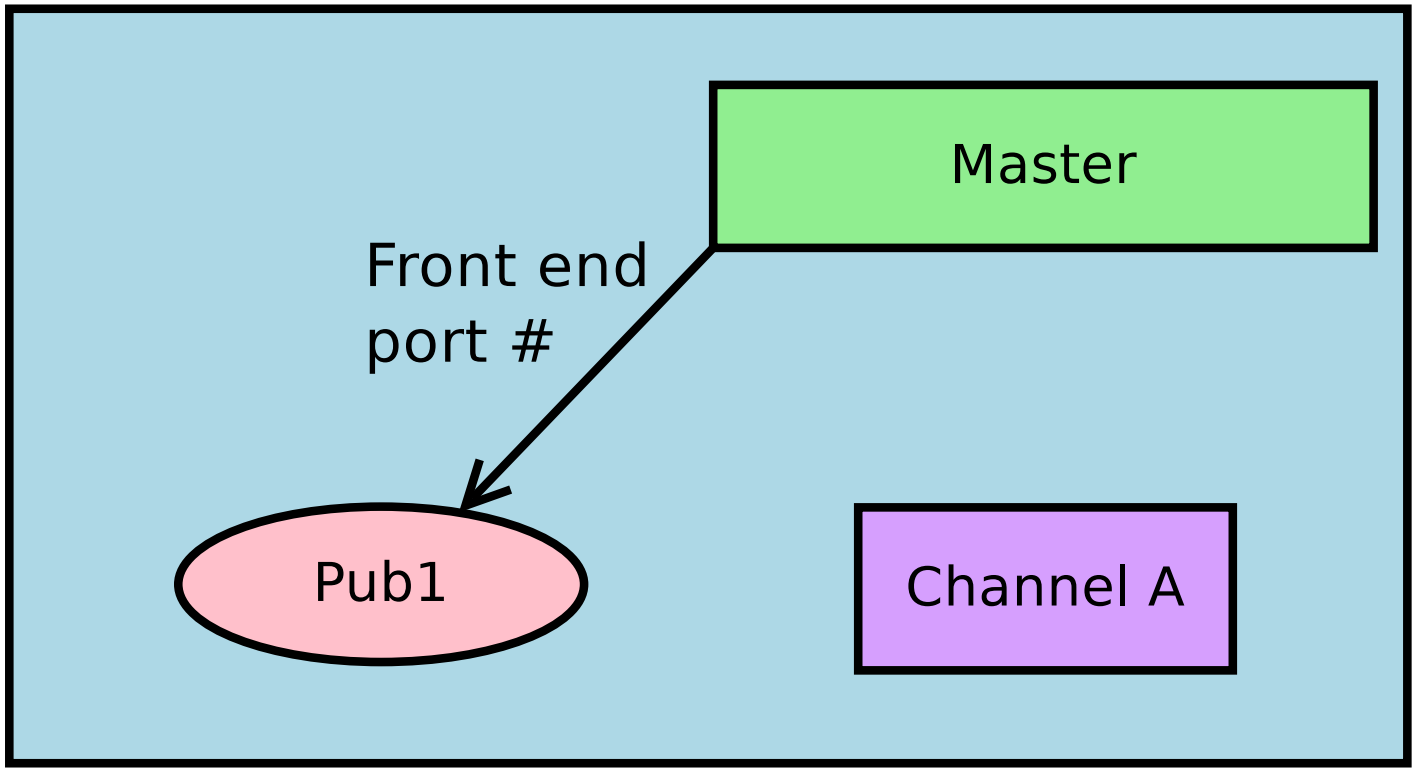
User Story

First time, the topic does not exist and needs to be created.



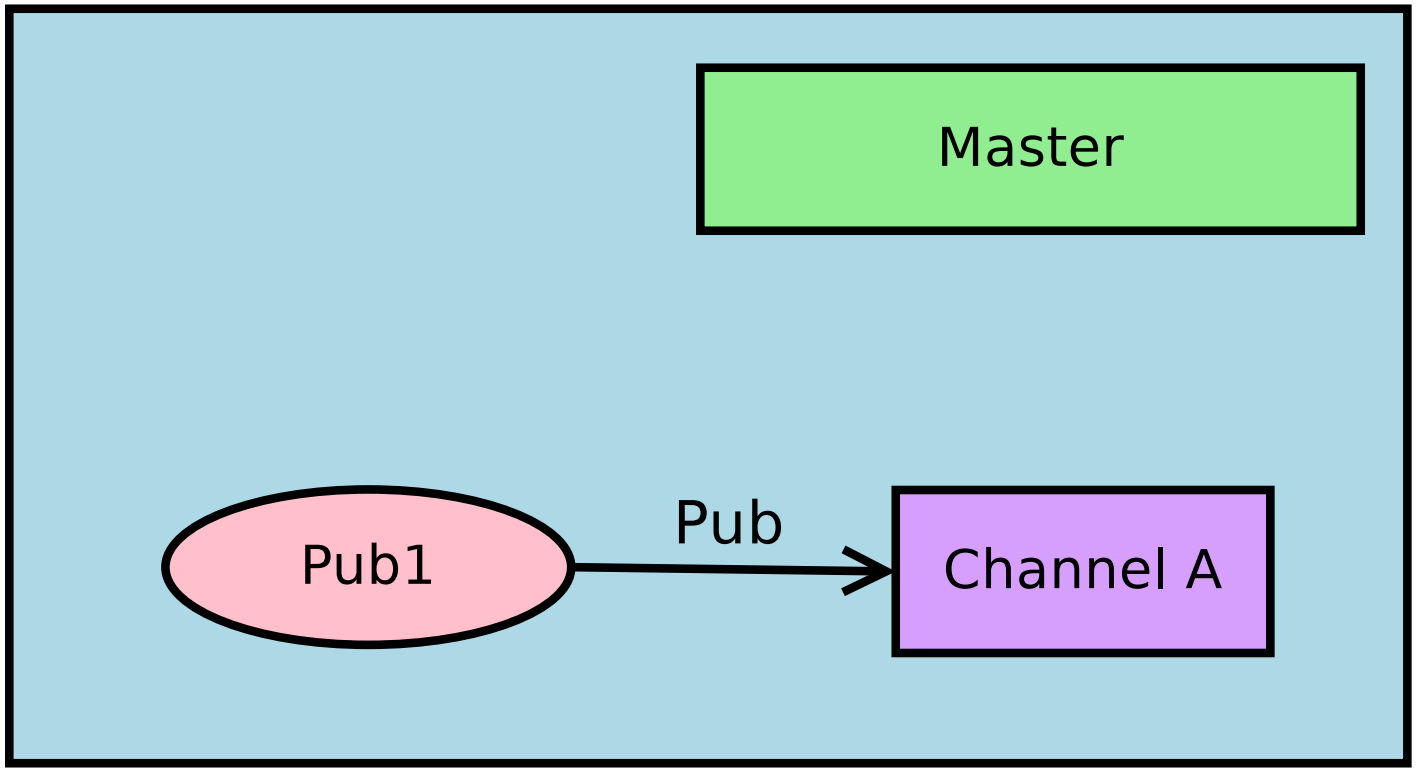
User Story

The relevant connection credentials are provided by the master to the prospective publisher (IP, port, etc for publisher side).



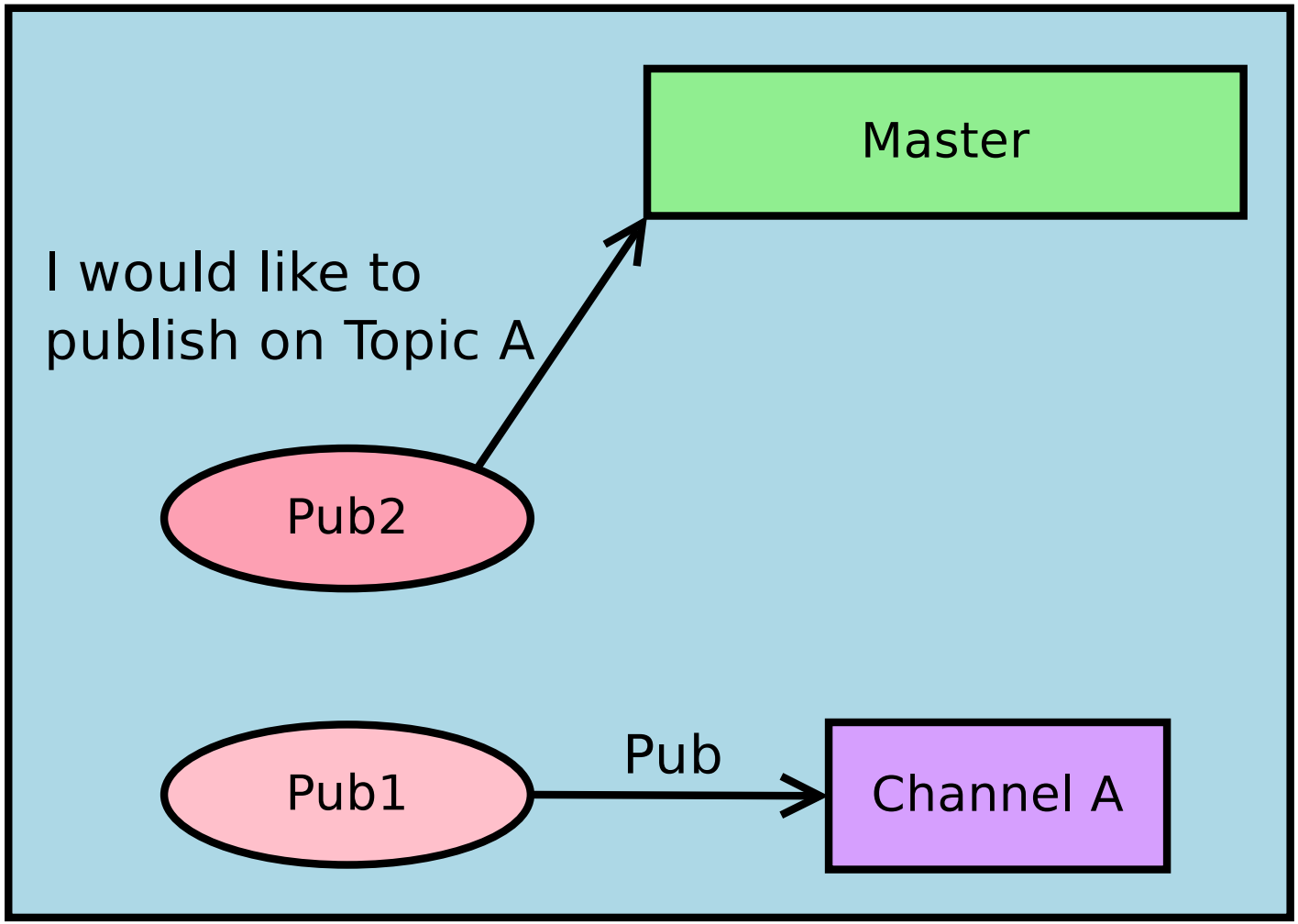
User Story

Using the credentials, the process (publisher) can connect with the topic "publish" messages.



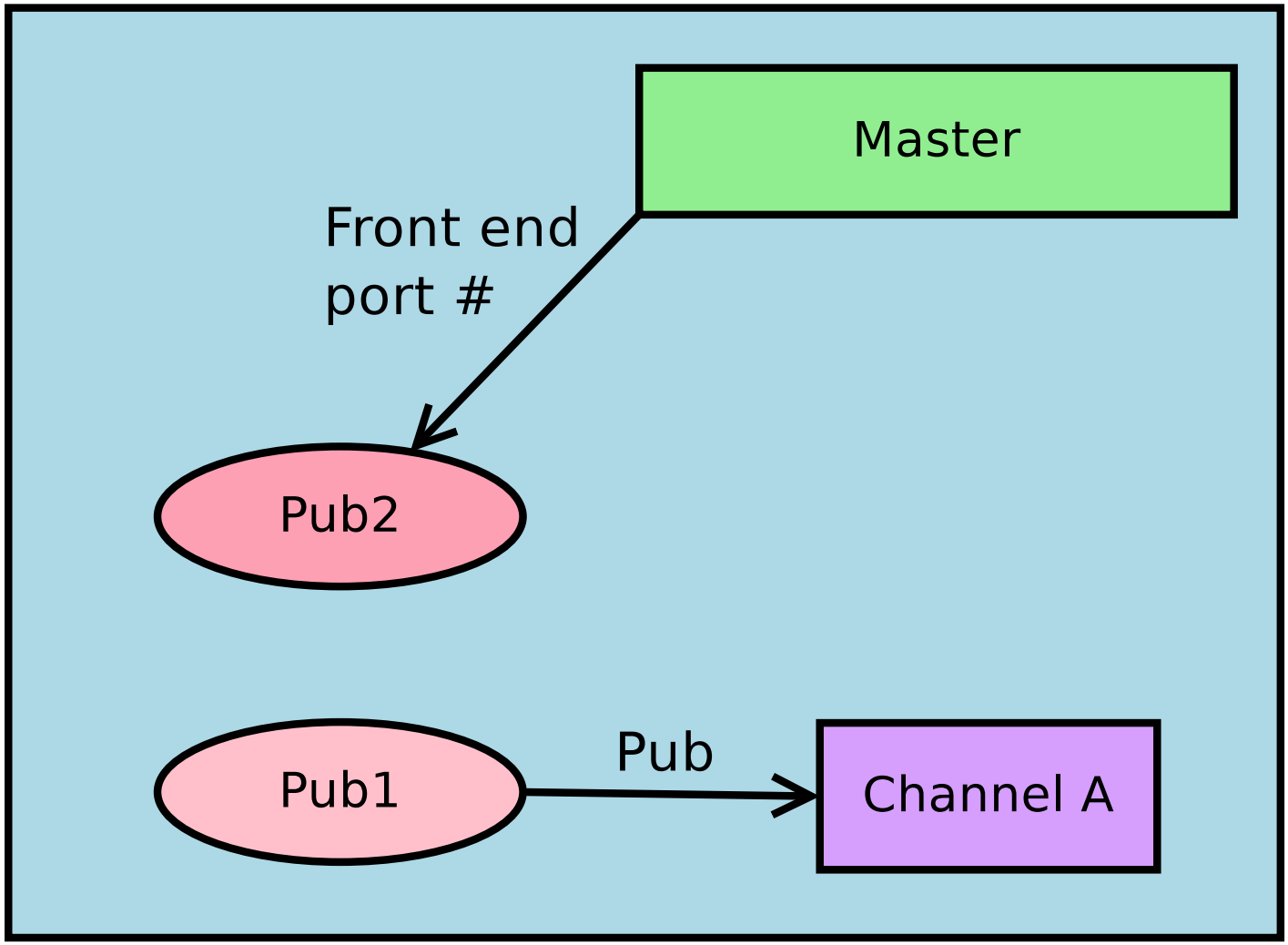
User Story

Another process may enter the process cluster and request to publish on topic A.



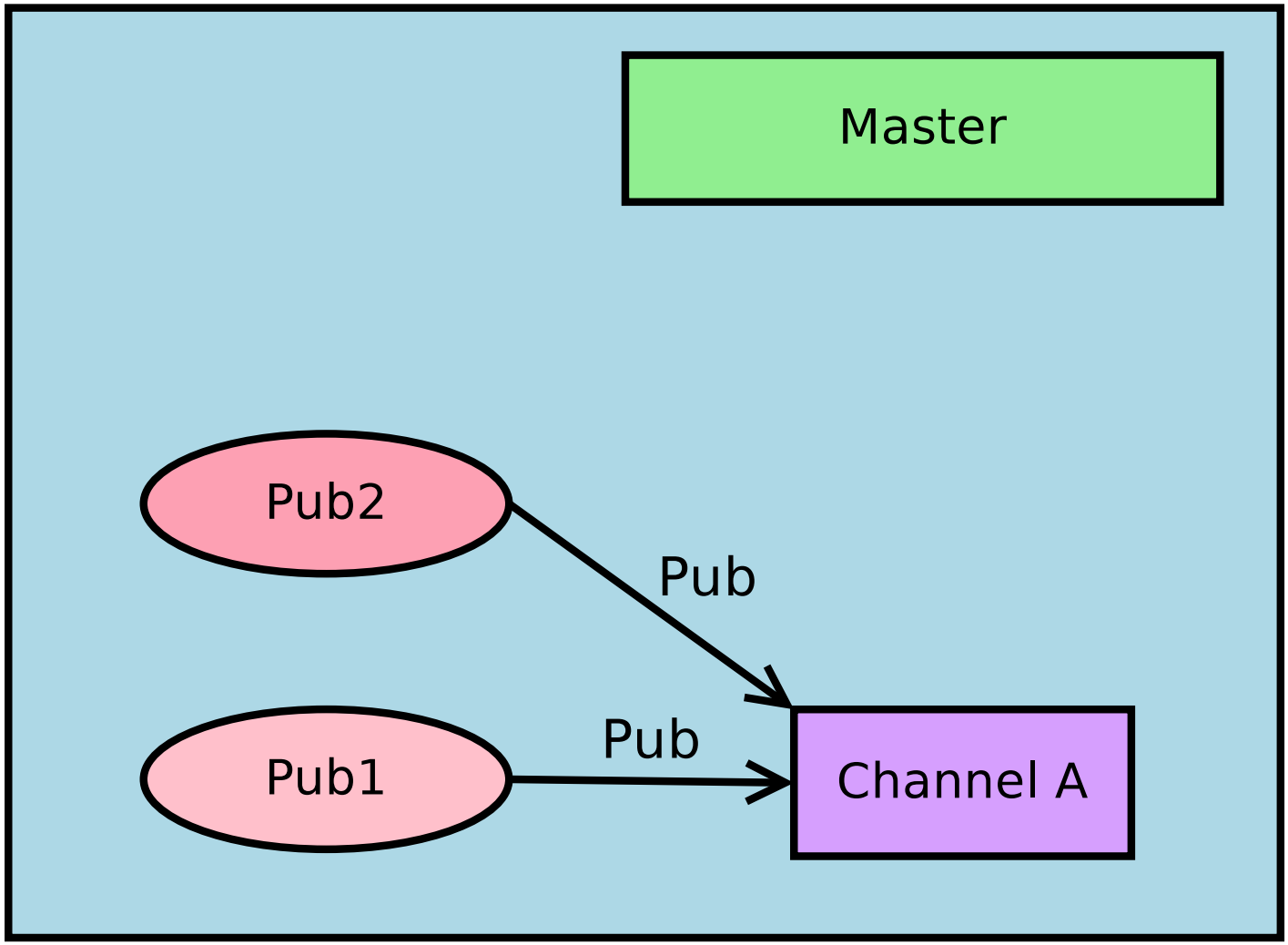
User Story

Topic A exists and so the credentials are sent from the Master's database.



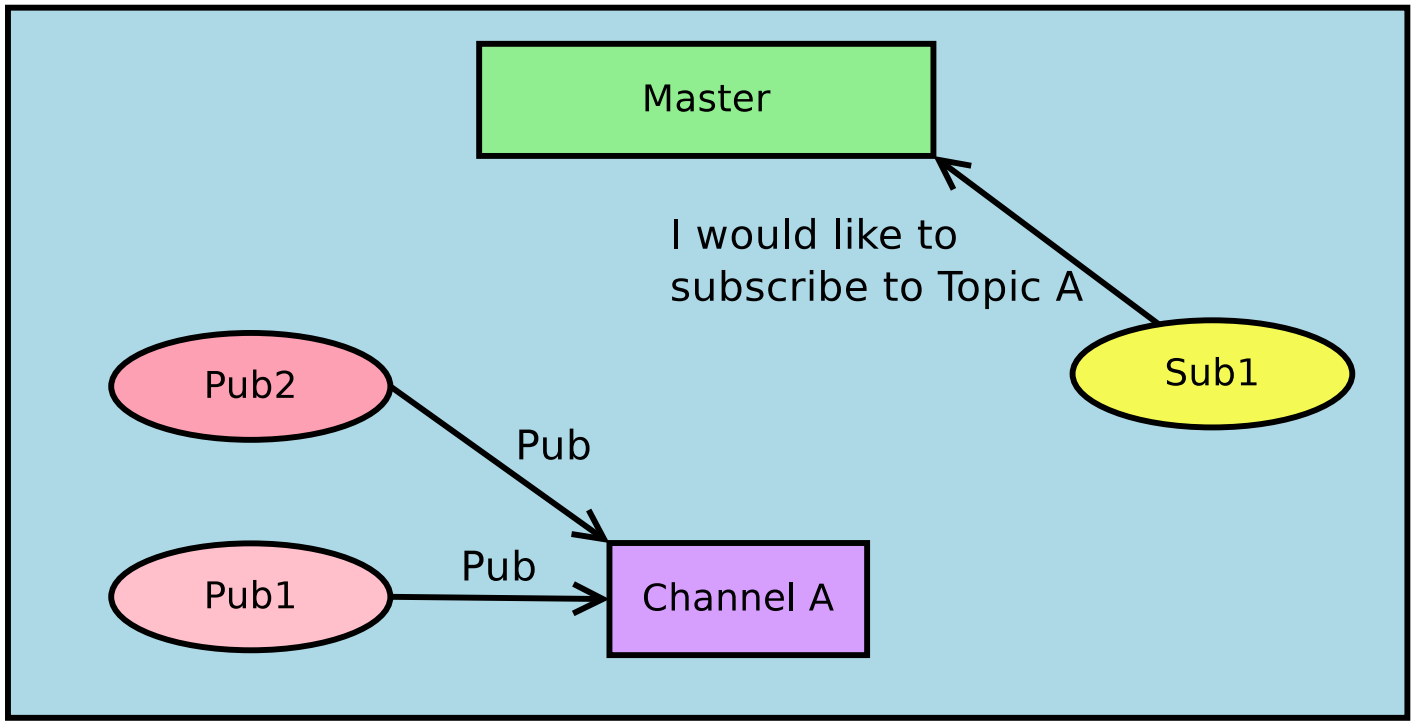
User Story

So both processes can publish on the topic.



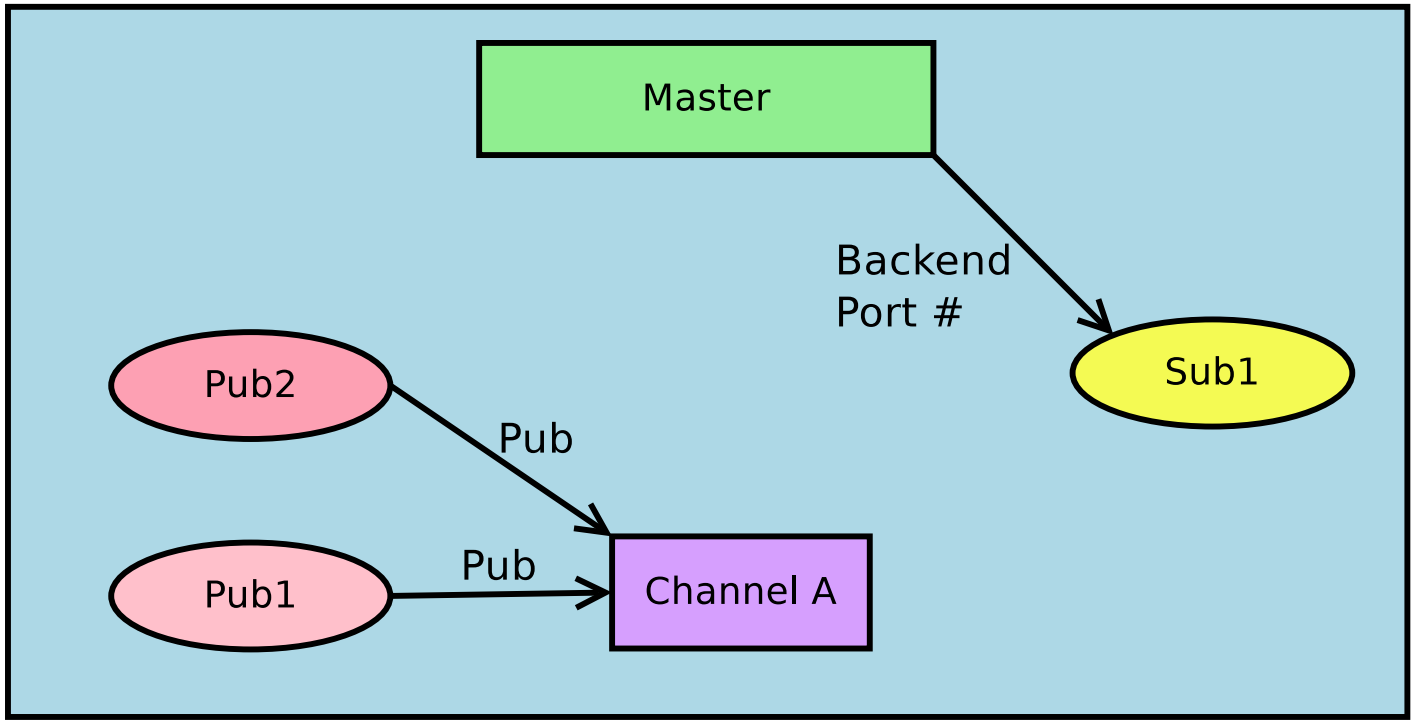
User Story

So far no process is receiving. Now, another process enters the cluster and requests to receive messages from topic A.



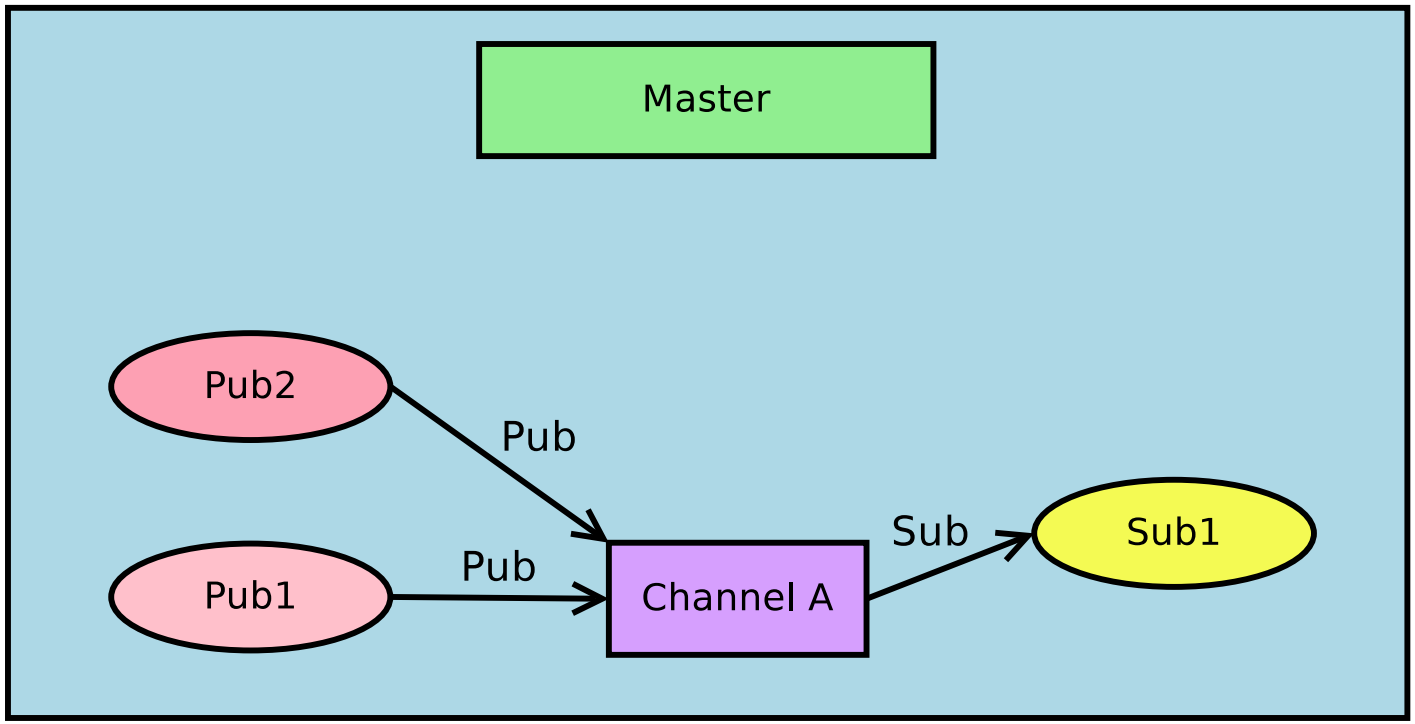
User Story

The master can send the credentials for the receiver side.



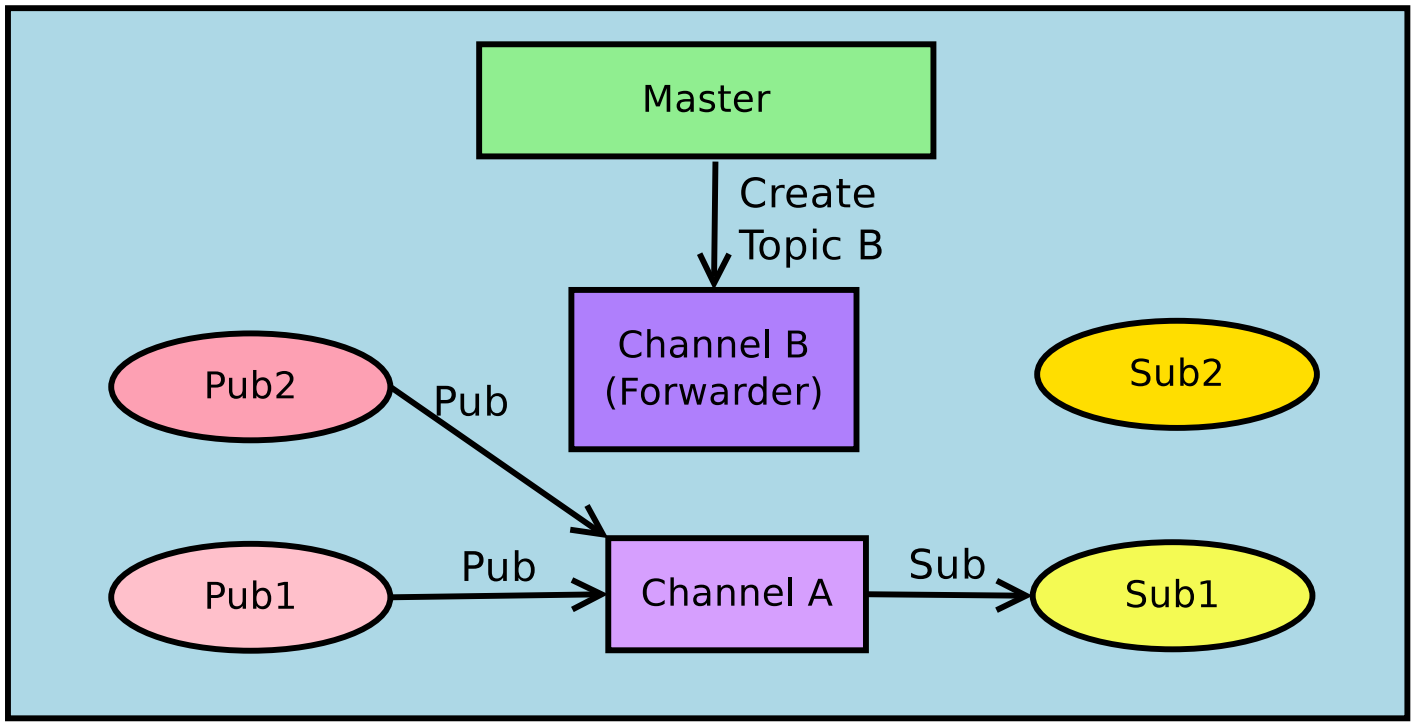
User Story

And this process can connect to the topic and receive the messages.



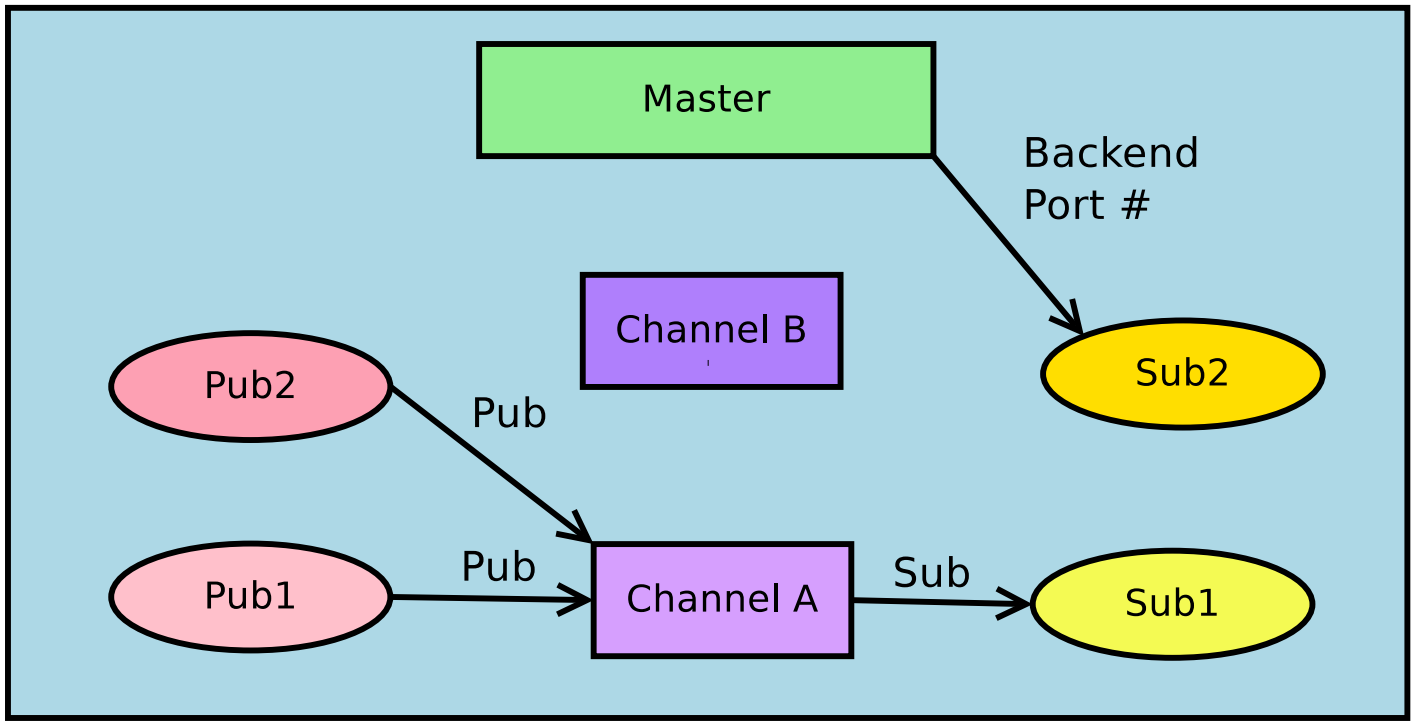
User Story

A fourth process can enter and request to receive messages on topic B. Since this is new, the master will create, store the data in its database.



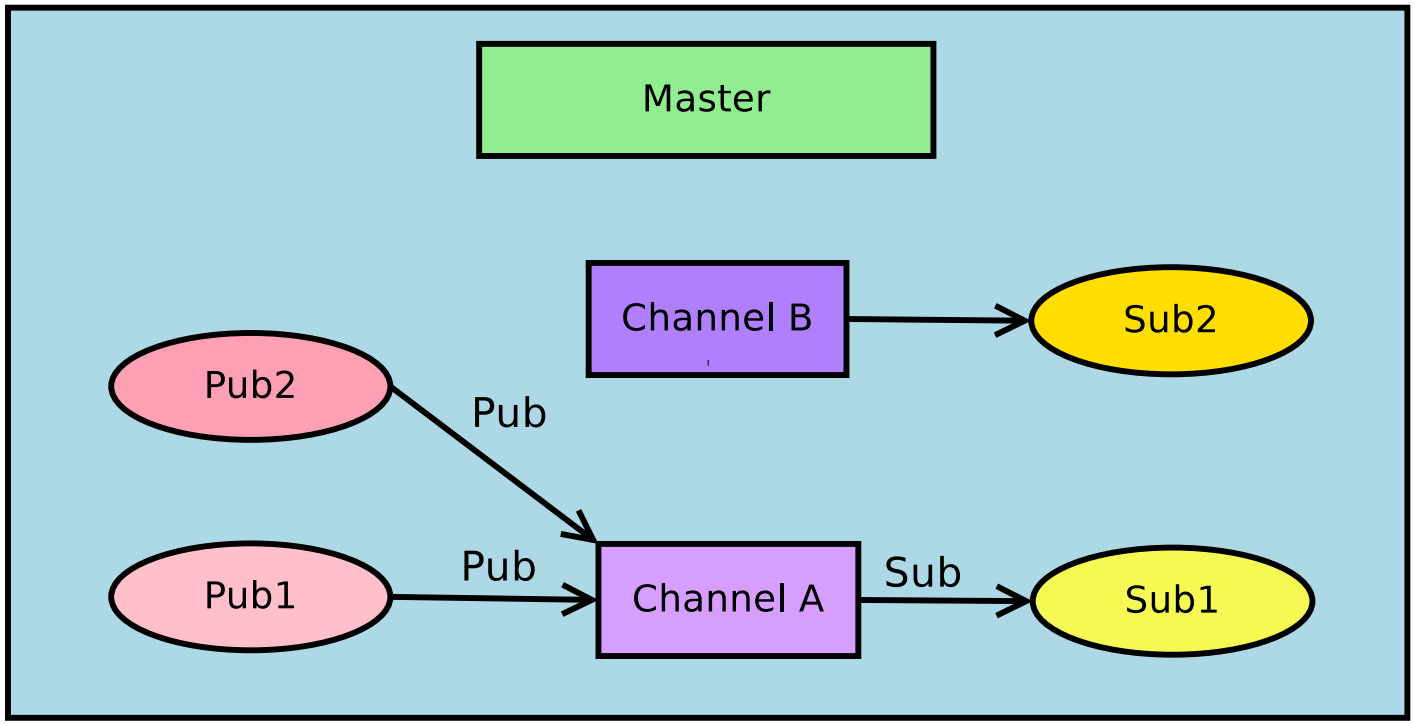
User Story

The master will then provide the relevant credentials to the new subscriber.



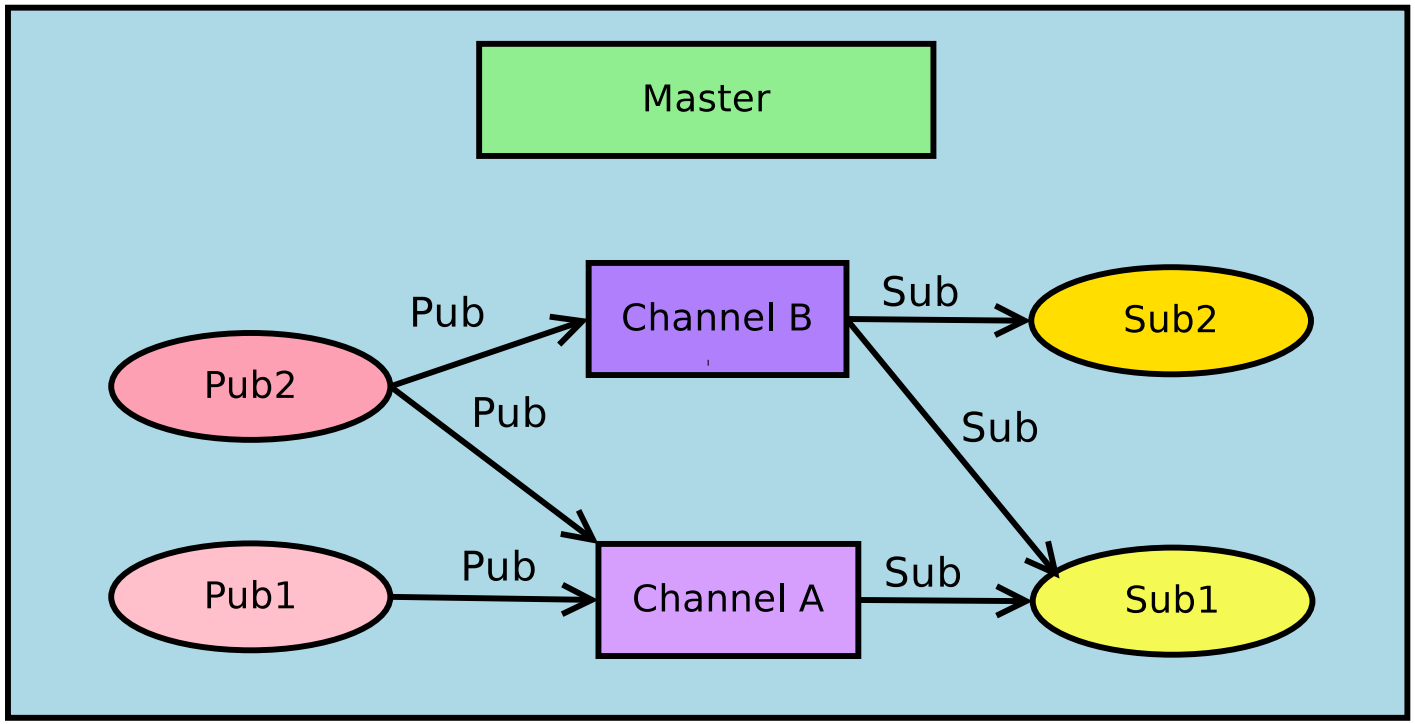
User Story

The second subscriber can connect and begin listening. The point here is that it should not matter the order that pubs and subs connect.



User Story

A process can publish to multiple topics and can subscribe to multiple topics.



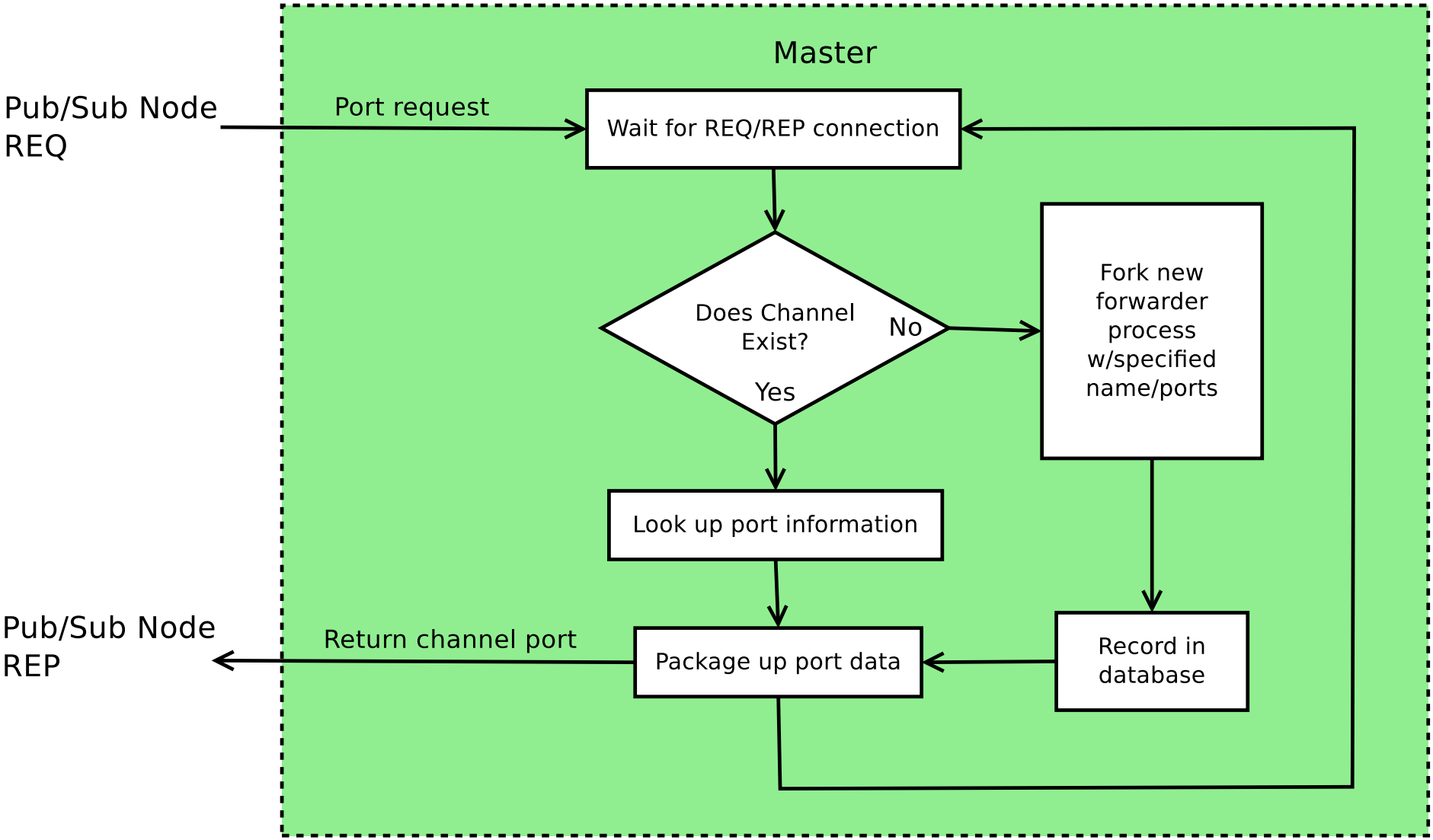
More

The master process is a directory service but not the router or message queue. Messages go directly from process to topic or topic to process.

This avoids bottlenecks in scaling.

Master keeps database of ports and IPs all pubs and sub. Keeps ports and IPs for processes managing the topics.

Master


Master
Pub/Sub Node
REQ
Port request
Wait for REQ/REP connection
Does Channel Exist?
No
Yes
Fork new forwarder process w/specified name/ports
Look up port information
Pub/Sub Node
REP
Return channel port
Package up port data
Record in database

Python Code Concept - Publisher

```
import tala as tl

master = "default"
name = "talkernode"
tl.join(name,master)

topic = "realityTV"

pub = tl.publisher(topic)

key = "0"
message = tl.pack("It is all scripted.")
tl.send(pub, message, key)
```

Python Code Concept - Subscriber

```
import tala as tl

master = "default"
name = "listennode"
tl.join(name,master)

topic = "realityTV"

sub = tl.subscribe(topic)

flag = 0  # timeout codes
message, key = tl.receive(sub, flag)
print(tl.unpack(message))
```

Aspects

There are configuration aspects (in a config file and dynamic).

- Message queue size.
- Message removal behavior and if multiple buffers required.
- Full FIFO behavior.
- Message data types.
- Message data marshalling.

Alternate Design

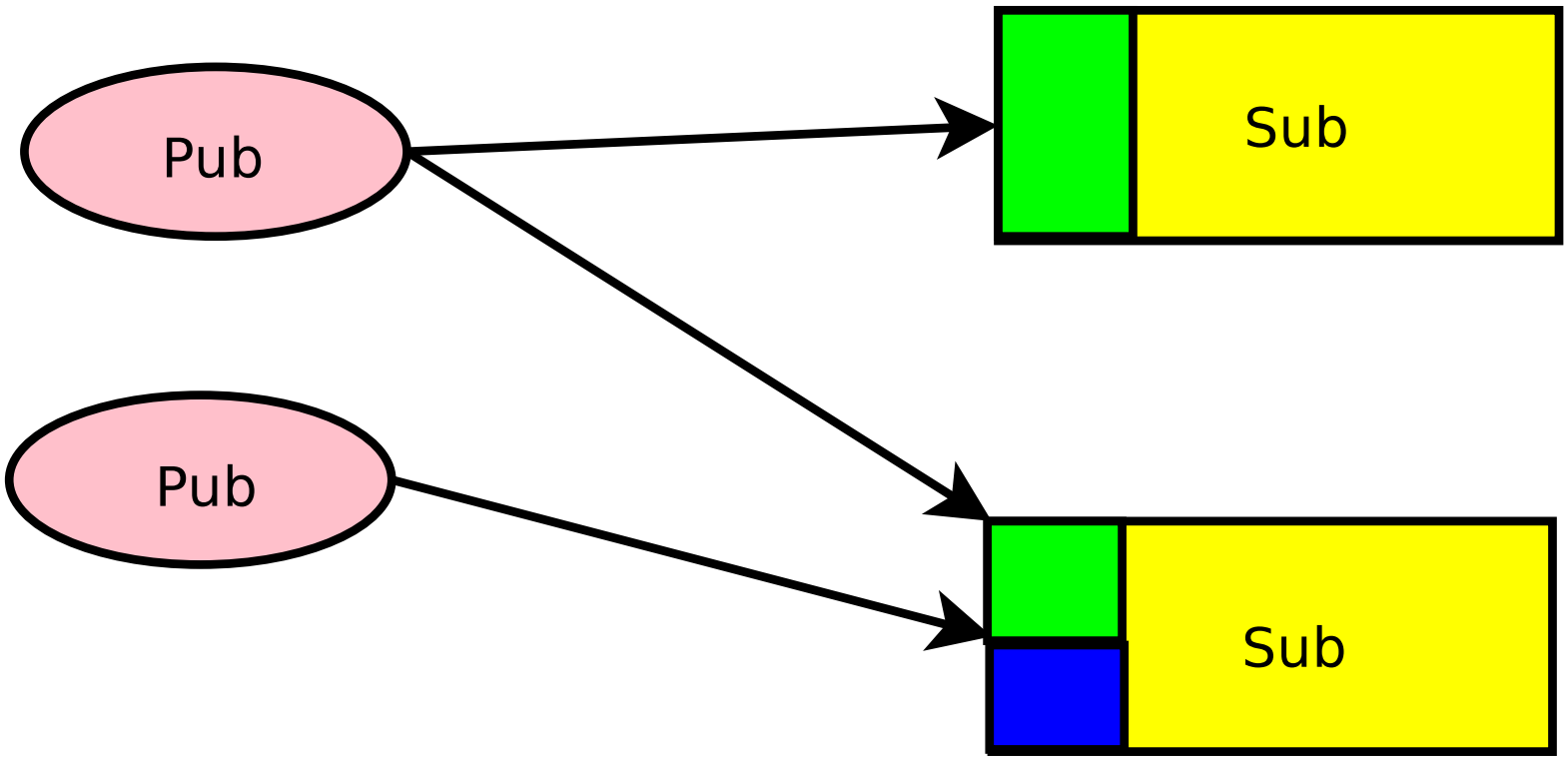
The topic can be bound to the subscriber.

The master will hand the subscriber's IP/port to the publisher and point to point communciation would happen.

For multiple subscribers, the master would need to contact each publisher and add a subscriber to their broadcast list.

Much of the user stories are the same, just the change in the location of the FIFO.

Alt Design



Design Issues

- Binding languages: Julia, Python, Rust
- Language for master (Rust or C++)
- Threading architecture
- Communication lib (0MQ vs Sockets)
- Communication architecture
- FIFO architecture

Tools etc

- GUI management tool
- Node/topic Graph plotting tool
- Listener and Talker examples in various languages.

Open source aspects (host, license, etc)

Requirements

- Directory service
  - Separate process
  - Tracks nodes in the compute cluster
  - Answers requests for node communication data
  - Internal database for network graph
  - API for GUI management
- Python & Julia APIs
  - Node registration
  - Pub/sub/req/rep setup functions
  - Push/pop/send/receive functions
  - Node exit and admin functions
- GUI Management
  - Launch master
  - Runtime config
  - Port ranges
- Graph tool
  - Present the node-topic graph

Software Expectations

- A prototype of the directory service - aka master.
- An initial API module for Python and Julia for calling Tala.
- A prototype of the GUI management tool.
- A prototype of the network graph tool.
- Some sample programs to illustrate Tala.

Project Expectations

- Design - because this is Senior Design.
- Communication - keep a dialog.[1]
- Teamwork - a sense of service to others.
- A great attitude - because every day is a gift.
- Open source project - with software described above.
- Documentation - of the code and the experience.

[1] I can handle bad news - I don't want any surprises