Wednesday

in case of problems yesterday:

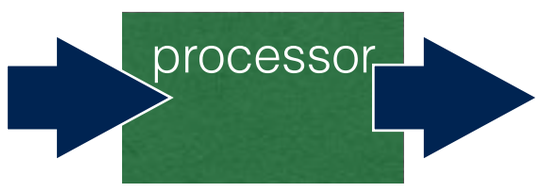
```
panda@panda-workshop:~/workshop/PandaRoot-trunk/$ mv MQ_samp_sink MQ_samp_sink_day1
panda@panda-workshop:~/workshop/PandaRoot-trunk/$ wget http://web-docs.gsi.de/~karabowi/
thailand/MQ_samp_sink.tgz
panda@panda-workshop:~/workshop/PandaRoot-trunk/$ tar xvzf MQ_samp_sink.tgz
```

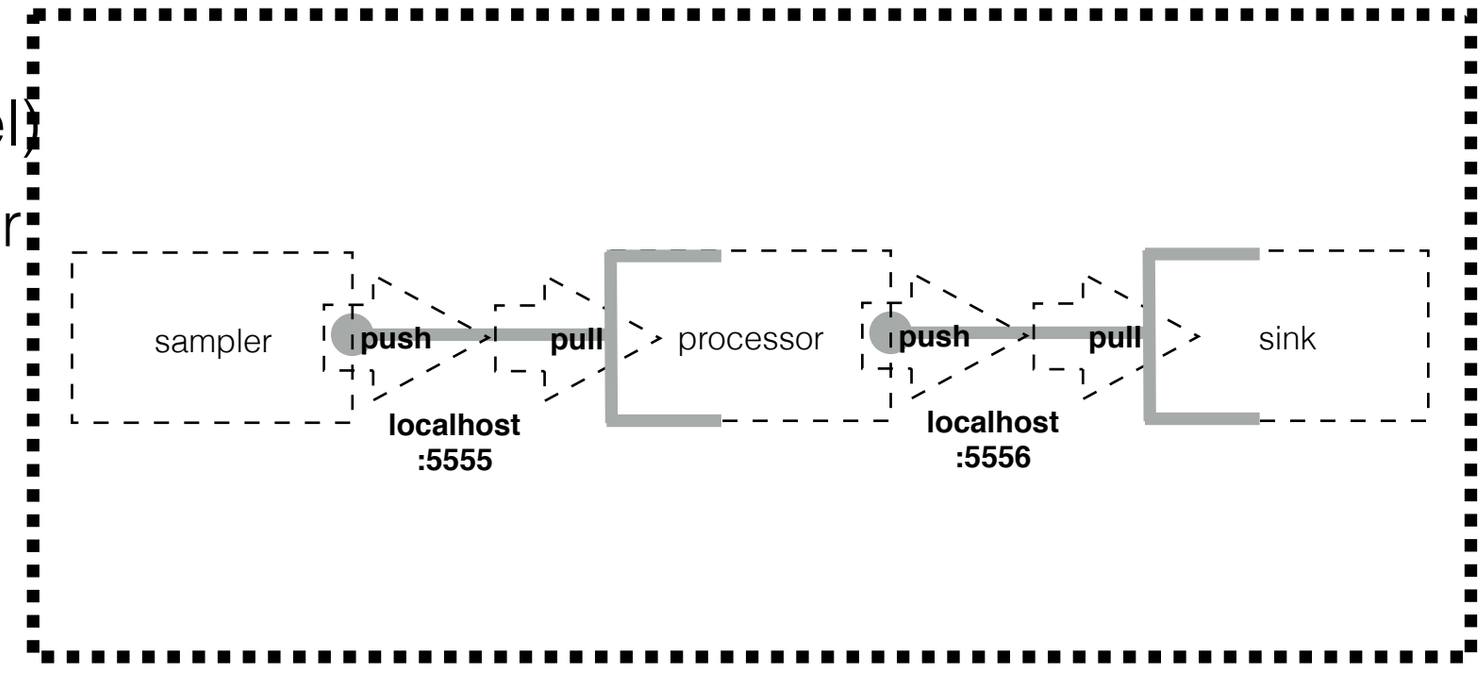
edit the PndMQ1Sampler.cxx file and change “sending” to “Sending”;

do `. ./config.sh` and `make` in the pandaroot build directory;

Day 1. Excercise 2

- create simple processor to process string, together with executable
- change CMakeLists.txt (device, executables)
- create topology consisting of three devices:
  - sampler1 (binding transport channel to push on this channel)
  - processor1 (connect to sampler channel, bind new channel)
  - sink1 (connecting to processor channel to pull from this channel)





Explanation

- Edit the topology description file (`MQ_samp_sink/options/MQ1_samp_sink.json`)

```
{
    "fairMQOptions": {
        "devices": [
            {
                "id": "sampler1",
                "channels": [
                    {
                        "name": "data",
                        "sockets": [
                            {
                                "type": "push",
                                "method": "bind",
                                "address": "tcp://*:5555",
                                "sndBufSize": 1000,
                                "rcvBufSize": 1000,
                                "rateLogging": 0
                            }
                        ]
                    }
                ]
            },
            {
                "id": "sink1",
                "channels": [
                    {
                        "name": "data",
                        "sockets": [
                            {
                                "type": "pull",
                                "method": "connect",
                                "address": "tcp://localhost:5555",
                                "sndBufSize": 1000,
                                "rcvBufSize": 1000,
                                "rateLogging": 0
                            }
                        ]
                    }
                ]
            }
        ]
    }
}
```

defines device with id sampler1

with one channel named “data”

with one socket

the socket will bind on port 5555, and will push data

TCP buffer size

the logging rate is 0 (no logging)

defines device with id sink1

with one channel named “data”

with one socket

the socket will connect to localhost port 5555, and will pull data

TCP buffer size

the logging rate is 0 (no logging)

Implement

- The processor is a device that has one input channel and one output channel.
- Create new topology file `MQ_samp_sink/options/MQ1_samp_proc_sink.json`
- Create new device with name “processor1”, with two channels: “dataIn” (pulls, connects to sampler port), “dataOut” (pushes, bind a port for sink).
- Use two different port numbers, f.e.:
  - 5555 for sampler's data channel and for processor’s dataIn channel
  - 5556 for sink’s data channel and for processor’s dataOut channel

Implement

- The processor has to receive (like sink) and send (like sampler) data;
- copy the PndMQ1Sink to PndMQ1Processor and change the class names;
- the processor creates output message based on input message, fe:

```
bool PndMQ1Proc::HandleData(FairMQMessagePtr& msg, int /*index*/)
{
  fText = string(static_cast<char*>(msg->GetData()), msg->GetSize());
  fText.insert(0,fId);
  fText.insert(0,"processed_");
```

- copy the code that creates and sends message from PndMQ1Sampler into PndMQ1Processor;
- the channel names to receive and send data have to match the ones used in the topology file.

Implement

- create corresponding `runPndMQ1Processor.cxx`;
- edit the `MQ_samp_sink/CMakeLists.txt` and add the processor device and executable;
- compile

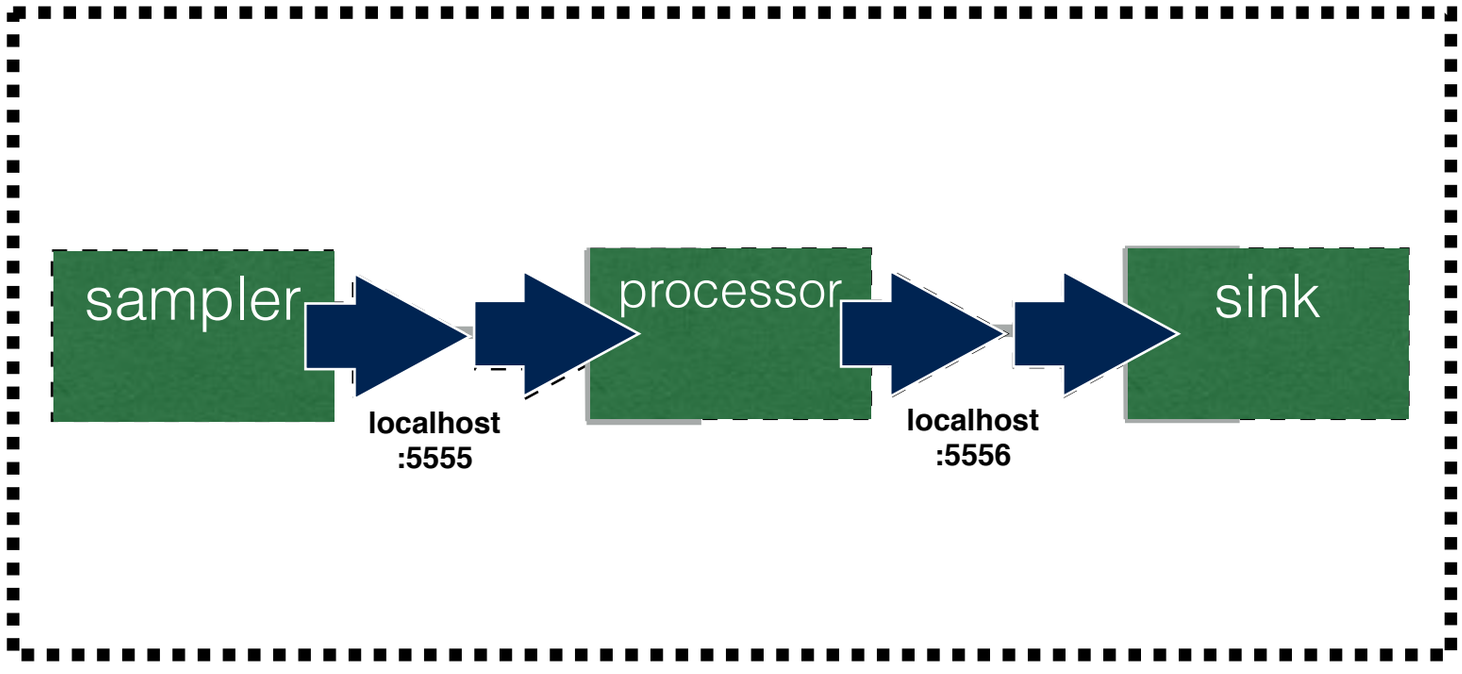
Day 1. Excercise 2

- run:

new topology file name

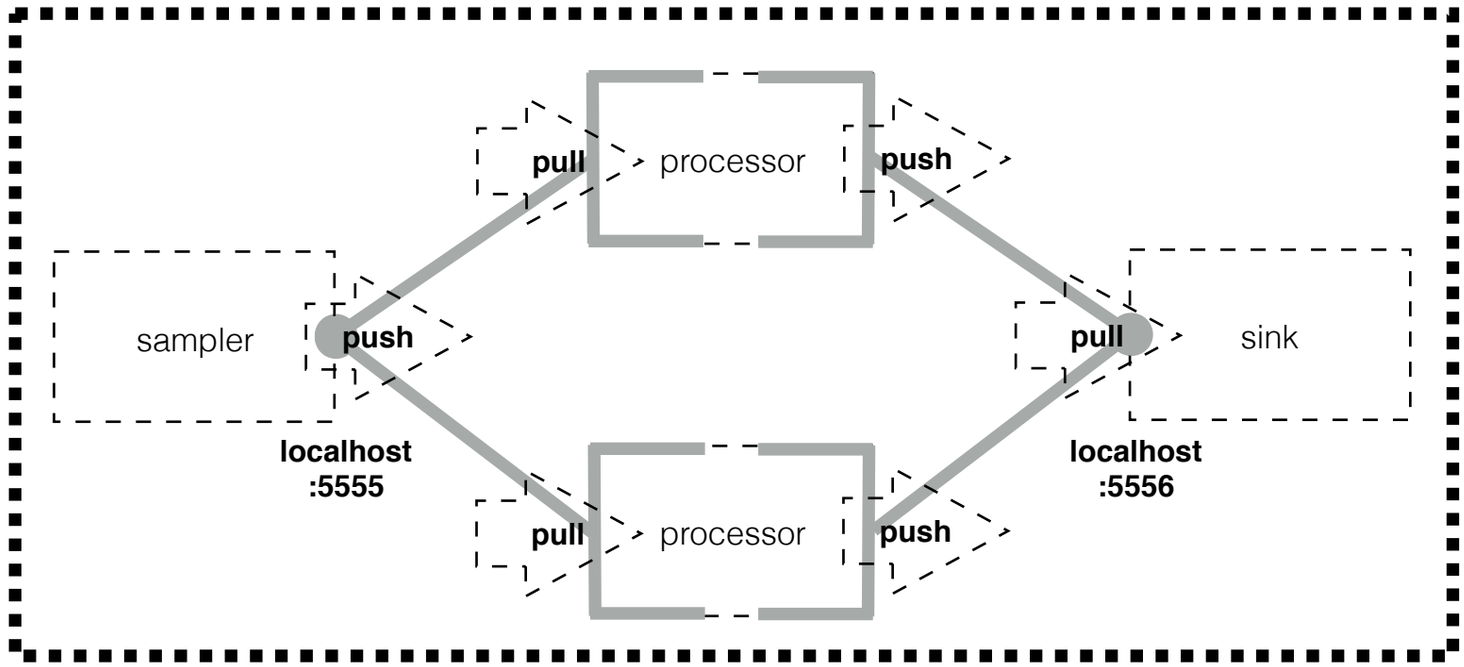
```
build$ ./bin/mq1-sink --id sink1 --mq-config ~/workshop/PandaRoot_trunk/MQ_samp_sink/options/MQ2.json
build$ ./bin/mq1-proc --id proc1 --mq-config ~/workshop/PandaRoot_trunk/MQ_samp_sink/options/MQ2.json
build$ ./bin/mq1-sampler --id sampler1 --mq-config ~/workshop/PandaRoot_trunk/MQ_samp_sink/options/MQ2.json
```

pandaroot build directory



Day 1. Excercise 3

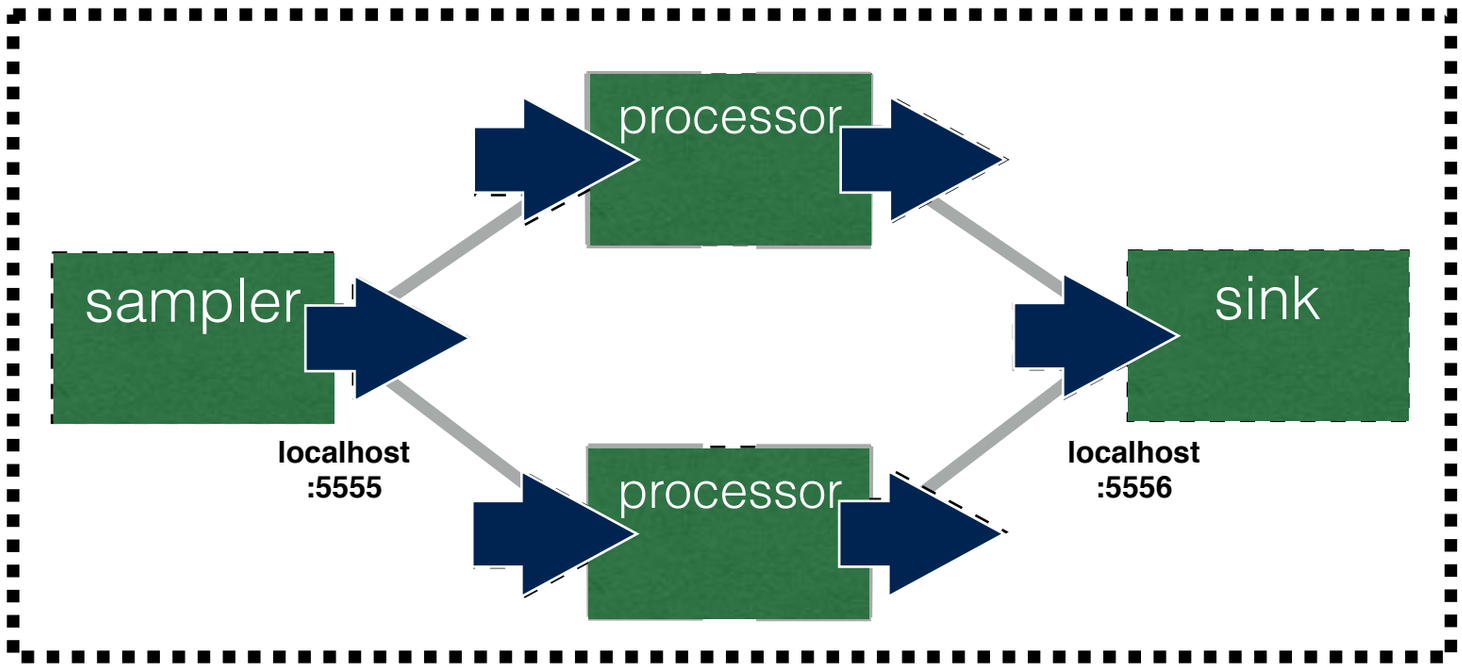
- modify topology to add one more processor:
  - sampler1 (binding transport channel to push on this channel)
  - proc1 (connect to sampler and sink channels)
  - proc1 (connect to sampler and sink channels)
  - sink1 (bind sink channel channel to pull from this channel)



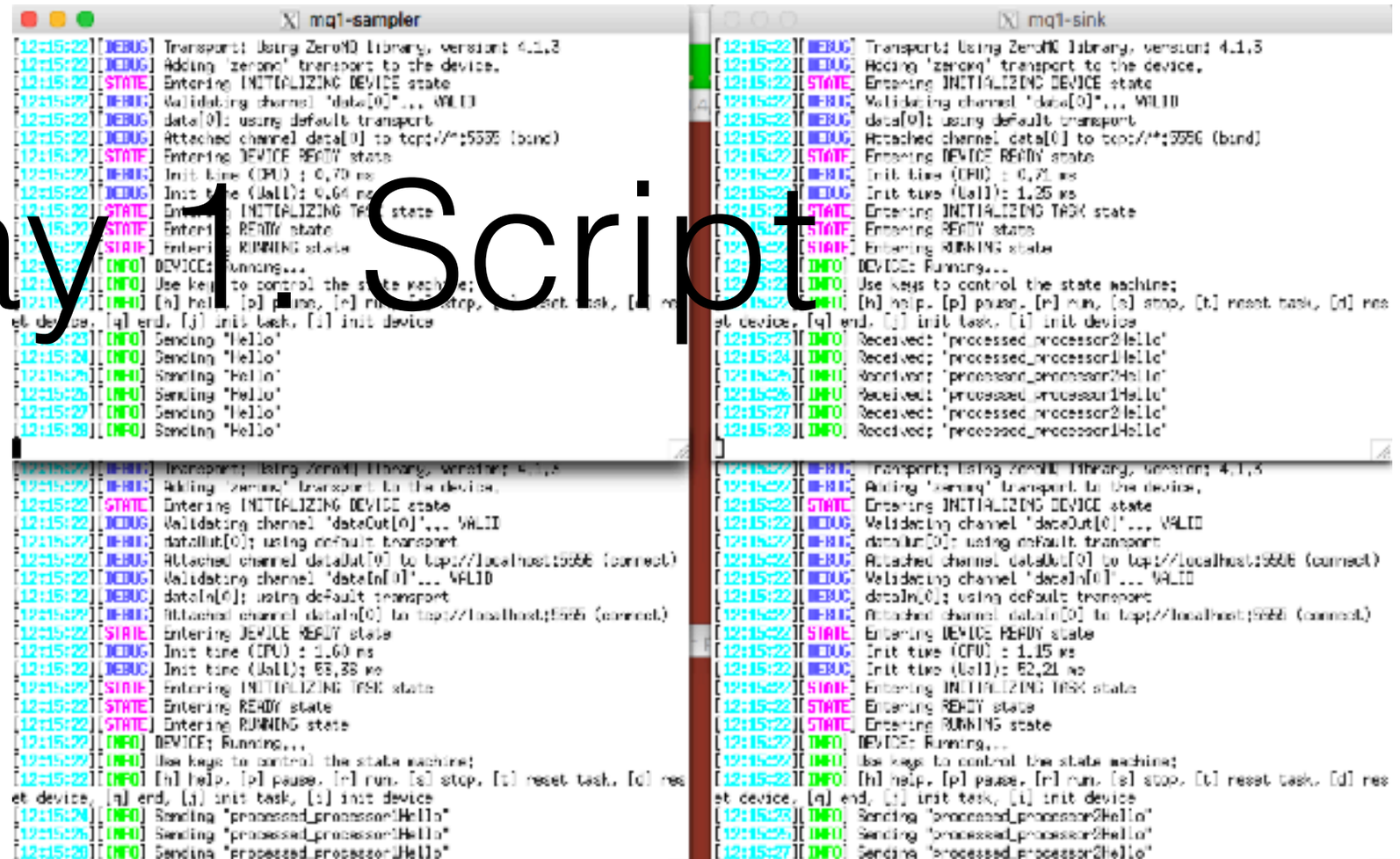
Day 1. Excercise 3

- run & play:

```
build$ ./bin/mq1-sink --id sink1 --mq-config ~/workshop/PandaRoot_trunk/MQ_samp_sink/options/MQ2.json
build$ ./bin/mq2-proc --id proc1 --mq-config ~/workshop/PandaRoot_trunk/MQ_samp_sink/options/MQ2.json
build$ ./bin/mq2-proc --id proc2 --mq-config ~/workshop/PandaRoot_trunk/MQ_samp_sink/options/MQ2.json
build$ ./bin/mq1-sampler --id sampler1 --mq-config ~/workshop/PandaRoot_trunk/MQ_samp_sink/options/MQ2.json
```



Day 1 Script




```
panda@panda-workshop:~/workshop/PandaRoot-trunk/MQ_samp_sink/run$ wget
http://web-docs.gsi.de/~karabowi/thailand/start_samp_2proc_sink.sh.in
panda@panda-workshop:~/workshop/PandaRoot-trunk/MQ_samp_sink/run$ cd ../options/
panda@panda-workshop:~/workshop/PandaRoot-trunk/MQ_samp_sink/options$ wget
http://web-docs.gsi.de/~karabowi/thailand/MQ2_samp_2proc_sink.json
```

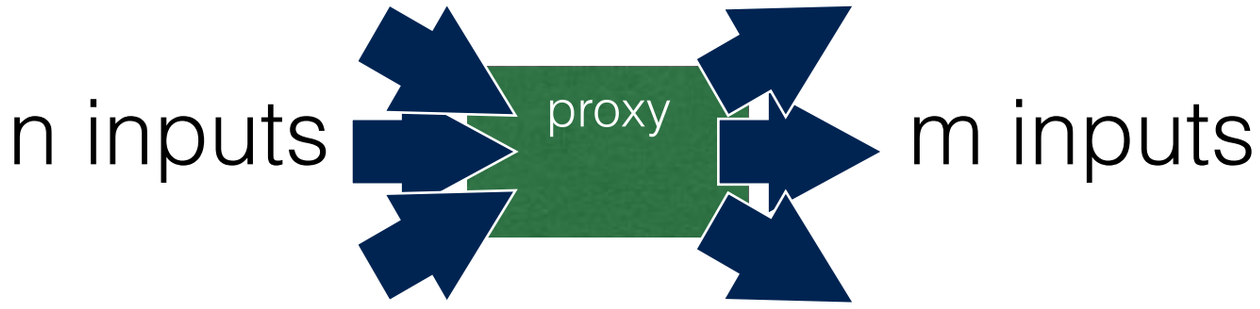
- add at the end of the MQ_samp_sink/CMakeLists.txt:

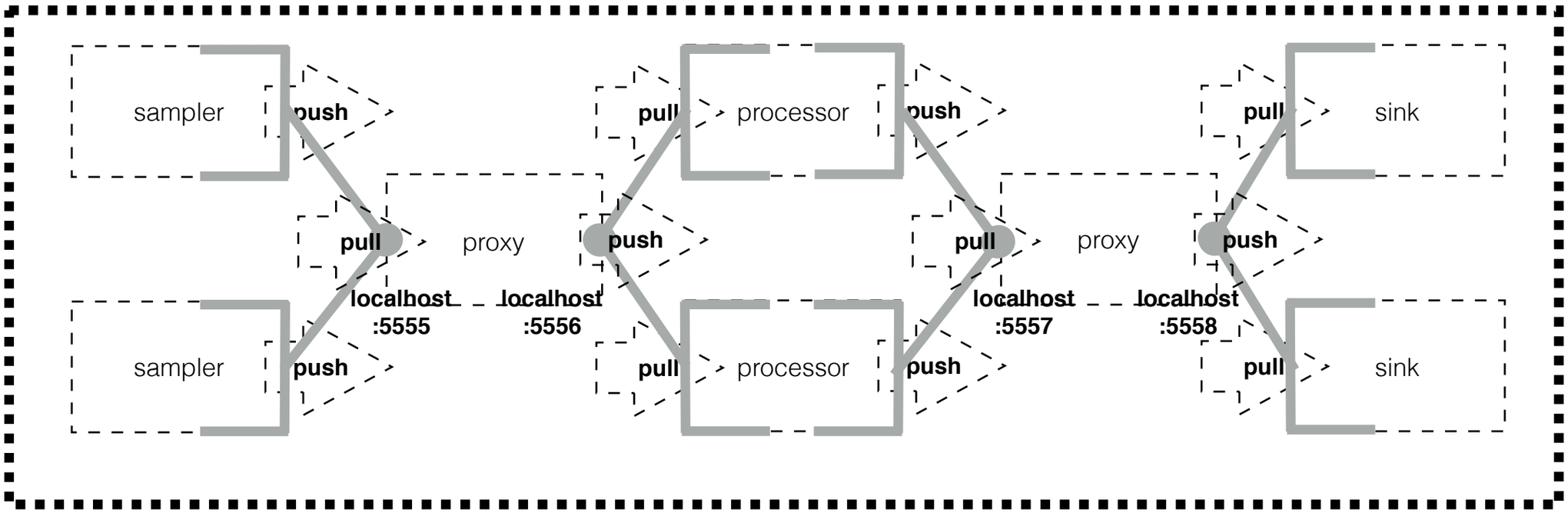
```
configure_file( scripts/start_samp_2proc_sink.sh.in
        ${CMAKE_BINARY_DIR}/bin/start_samp_2proc_sink.sh )
```

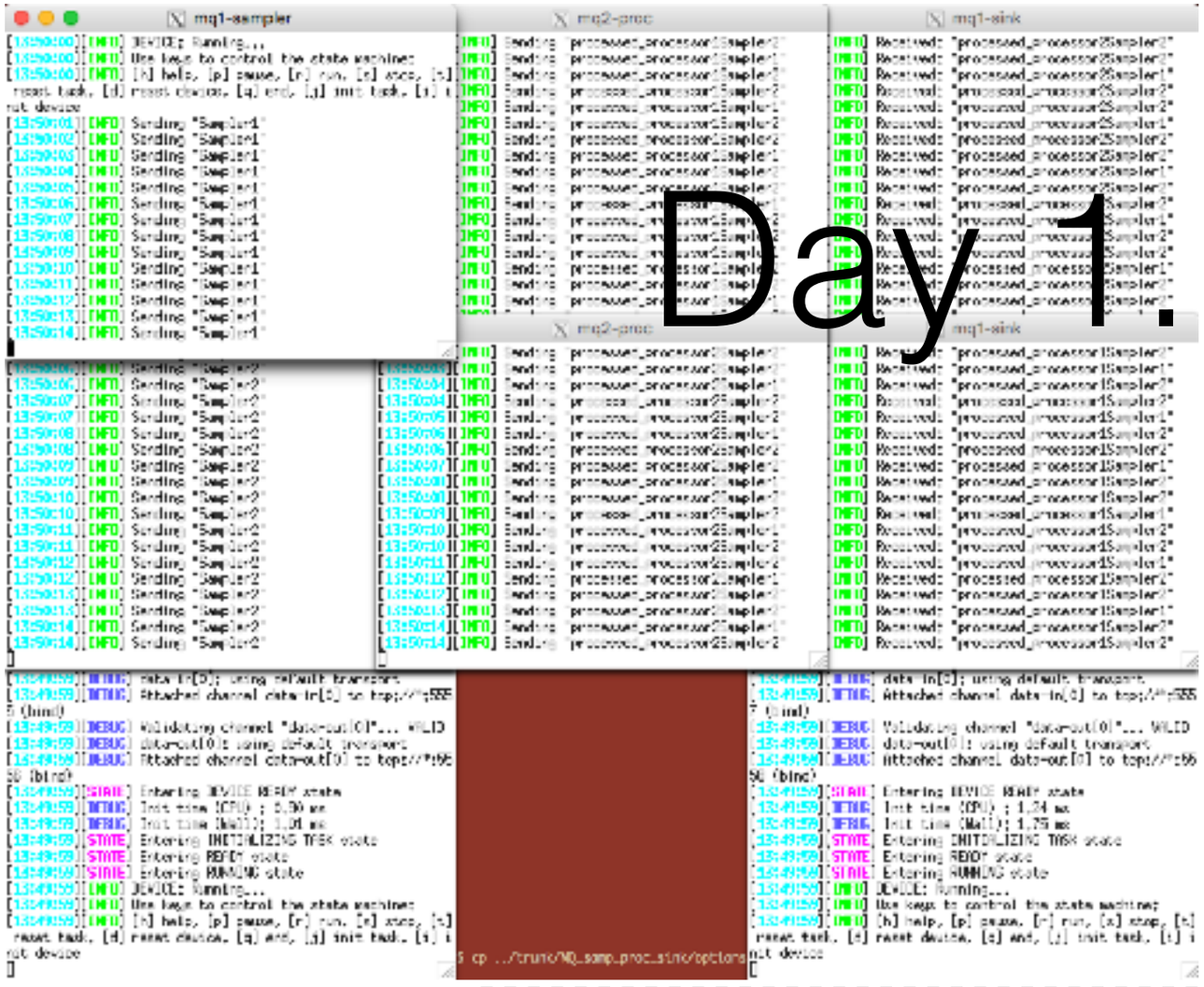
- in the build directory run: `cmake ~/workshop/PandaRoot_trunk/`
- and then: `./bin/start_samp_2proc_sink.sh`

Day 1. Proxy

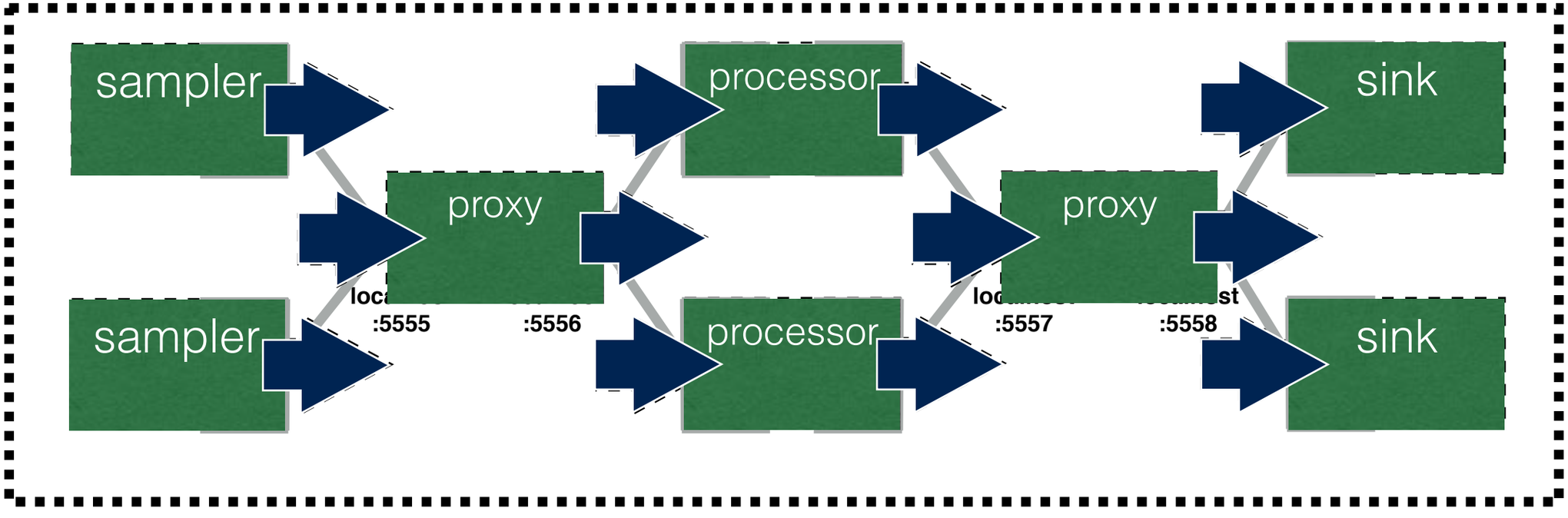
- use generic proxy device, able to connect nxm devices
- implement topology with 2 samplers, proxy, 2 processors, proxy, 2 sinks






mq1-sampler
mq2-proc
mq1-sink
mq2-proc
mq1-sink

Day 1. Proxy


sampler
sampler
proxy
:5555
:5556
processor
processor
proxy
:5557
:5558
sink
sink

Day 1 Finals

- Check if you can connect to my mac (ping 10.88.2.73
- I have two devices running: sampler (binds 10.88.2.73 : 5555) and sink (binds 10.88.2.73 : 5556)

- The sampler sends string: “String”
- The sink receives and prints out string;

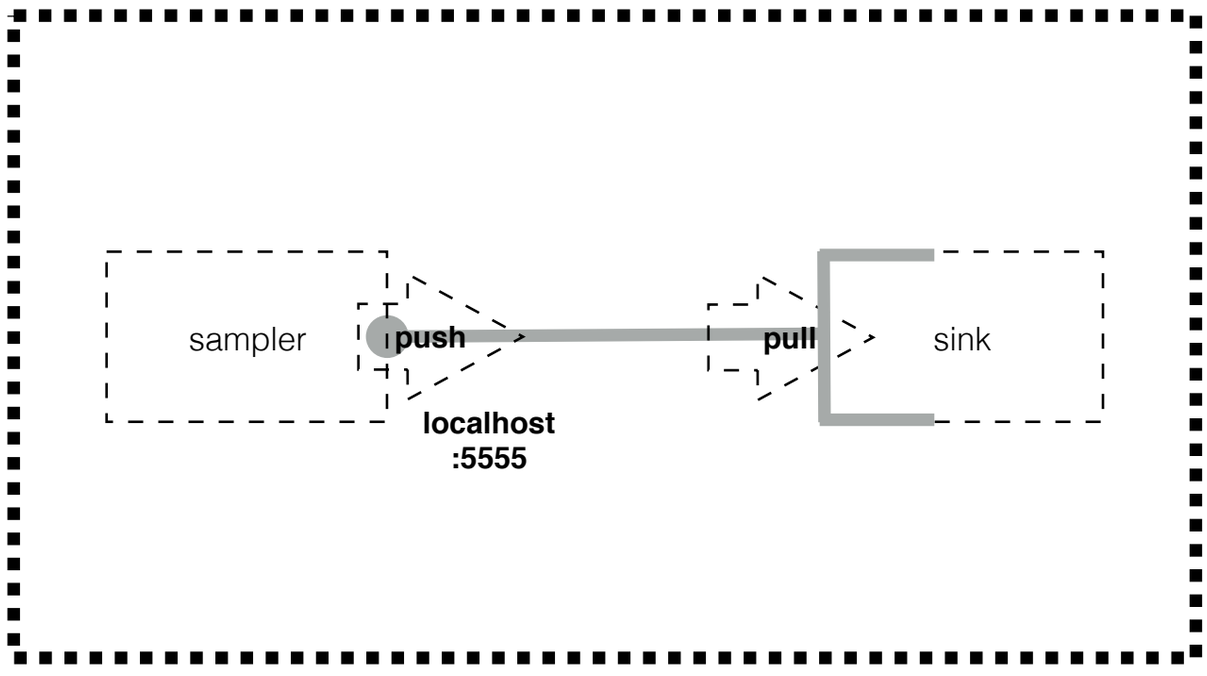
- modify your processor device to attach your name to the string
- create a new topology with only one processor, that will connect to my sampler and sink.
- run the processor

Day 2

- transport root data (TMessage)
- multipart message
- parmq server

Day 2. Excercise1

- create sampler and sink to send/receive a TMessage with TClonesArray
- create CMakeLists.txt (library, executables)
- create topology consisting of two devices:
  - sampler1 (binding transport channel to push on this channel)
  - sink1 (connecting to said channel to pull from this channel)



Implement sampler

- get the TClonesArray from the input root file
- wrap in the TMessage and send:

```
TMessage* message = new TMessage(kMESS_OBJECT);
```

create a TMessage

```
message->WriteObject(fObject);
```

write TObject (f.e. TClonesArray) to it

```
FairMQMessagePtr msg(NewMessage(message->Buffer(),
                                message->BufferSize(),
                                [](void* /*data*/, void* object){delete (TMessage*)(object);},
                                message));
```

create a FairMQMessage

```
Send(msg, "data");
```

send it

Check*

- on the receiver side (sink), you can just convert the message to string;
- —> use the existing sink (for string) to print it out;

Implement sink

- receive TMessage…

```
TMessage recMessage(msg->GetData(), msg->GetSize());
TObject*  recObject = (TObject*)recMessage.ReadObject(recMessage.GetClass());

LOG(INFO) << "message class is \"" << recMessage.GetClass() << "\" "
          << "(\"" << recMessage.GetClass()->GetName() << "\" : “
          << "\”" << recMessage.GetClass()->GetTitle() << "\")";
LOG(INFO) << "object name is \"" << recObject->GetName() << "\"";

if ( strcmp( recMessage.GetClass()->GetName(),"TClonesArray" ) == 0 ) {
  LOG(INFO) << " the TClonesArray has " << ((TClonesArray*)recObject)->GetEntries() << " entries.";
}
else {
  LOG(INFO) << "\"" << recMessage.GetClass()->GetName() << "\" != \"TClonesArray\"";
```

PndTMessage

- workaround for the lack of TMessage's public constructor:

```
panda@panda-workshop:~/workshop/PandaRoot_trunk/MQ_TMessage/devices$ cat PndTMessage.h
#ifndef PNDTMESSAGE_H_
#define PNDTMESSAGE_H_

#include "TMessage.h"

class PndTMessage : public TMessage
{
 public:
  PndTMessage(void* buf, Int_t len);
};

#endif /* PNDTMESSAGE_H_ */
panda@panda-workshop:~/workshop/PandaRoot_trunk/MQ_TMessage/devices$ cat PndTMessage.cxx
#include "PndTMessage.h"

PndTMessage::PndTMessage(void* buf, Int_t len)
  : TMessage(buf, len)
{
  ResetBit(kIsOwner);
}
panda@panda-workshop:~/workshop/PandaRoot_trunk/MQ_TMessage/devices$ grep PndTMessage
PndMQSinkTMessage.cxx
devices/PndMQSinkTMessage.cxx:#include "PndTMessage.h"
devices/PndMQSinkTMessage.cxx:  PndTMessage recMessage(msg->GetData(), msg->GetSize());
```

Processor

- receives TMessage with input TClonesArray;
- sends TMessage with output TClonesArray;

What about the params?

- the processor needs access to the parameters -> parmq-server
- access to parmq-server device -> request - reply pattern
- which parameters to request -> info available in the event header
- send event header to processor -> use multi-part message

```
TMessage* messageEH  = new TMessage(kMESS_OBJECT);
TMessage* messageTCA = new TMessage(kMESS_OBJECT);

messageEH ->WriteObject(fEventHeader);
messageTCA->WriteObject(fObject);

FairMQParts parts;
parts.AddPart(NewMessage(messageEH ->Buffer(),
                         messageEH ->BufferSize(),
                         [](void* /*data*/, void* object){delete (TMessage*)(object);},
                         messageEH ));
parts.AddPart(NewMessage(messageTCA->Buffer(),
                         messageTCA->BufferSize(),
                         [](void* /*data*/, void* object){delete (TMessage*)(object);},
                         messageTCA));

Send(parts, "data");
```

What about the params?

- receive parts:

```
bool PndMQProcessor::ProcessData(FairMQParts& parts, int /*index*/)
{
  TObject* tempObjects[10];
  for ( int ipart = 0 ; ipart < parts.Size() ; ipart++ )
    {
      PndTMessage tm(parts.At(ipart)->GetData(), parts.At(ipart)->GetSize());
      tempObjects[ipart] = (TObject*)tm.ReadObject(tm.GetClass());
      if ( strcmp(tempObjects[ipart]->GetName(),"EventHeader.") == 0 )
        {
          fEventHeader = (FairEventHeader*)tempObjects[ipart];
          fNewRunId = fEventHeader->GetRunId();

          if(fNewRunId!=fCurrentRunId)
            {
              fCurrentRunId=fNewRunId;
              UpdateParameters();
            }
        }
      else
        {
          // do proper things
        }
  }
}
```

Day 3

- brainstorming

- how to have one task that could be used both in FairRoot and in FairMQ?

Day 3. MQ/example9

- one idea (MQ/example9): implement three functions in a given FairTask:
  - `virtual void GetParList(TList* tempList);` to allow device to obtain list of parameters needed by a task
- `virtual void InitMQ (TList* tempList);` to supply the parameters received by device to the task
- `virtual void ExecMQ (TList* inputList,TList* outputList);` to run the Task::Exec on the input data (inside inputList) and create output data (transferred via outputList)

Day 3. ***FairDataManager*** ***FairParameterManager***

- currently we have hundrends of tasks that are running in FairRoot.
- they communicate with the framework scheleton via FairRootManager and FairRuntimeDb
- maybe replace them with:
- FairDataManager - it should simply be doing job of FairRootManager in FairRoot. In FairMQ, the data received by a device would be put in FairDataManager, so that it can be accessed in task
- FairParameterManager - it should simply be doing job of FairRuntimeDb in FairRoot. In FairMQ, the task will obtain the parameters via this FairParameterManager….