Embedded Software

A Message Distribution System

Agenda

- Current design
- Two approaches for decoupling
  - Specific receiver
  - Broadcasting - who the receiver is is irrelevant
- Singleton pattern
- In perspective
  - Publish/Subscriber schemes and coupling
  - Group and plenum discussion

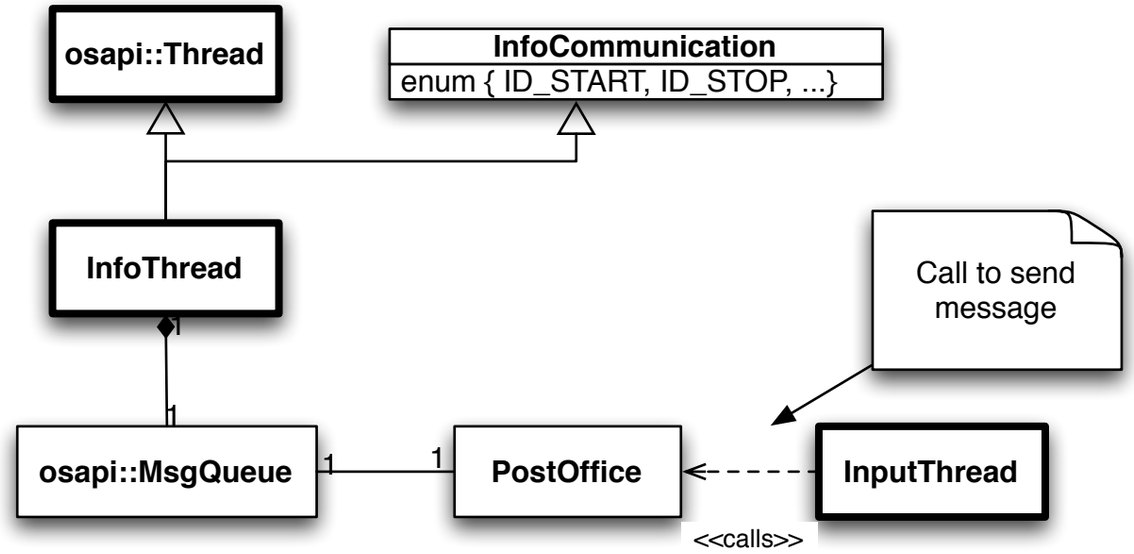
Current design and next step

ENGINEERING COLLEGE OF AARHUS

Current design

- A thread has a message queue through which other threads pass it messages
  - Consequence is that “other” threads need to have access to it’s message queue.
  - At application start these pointers (or references) must be passed around
- Problems - Potential Couplings issues
  - Challenges during creation - *chicken and the egg*
  - Leading to cyclic includes
  - Close relationships that are not needed

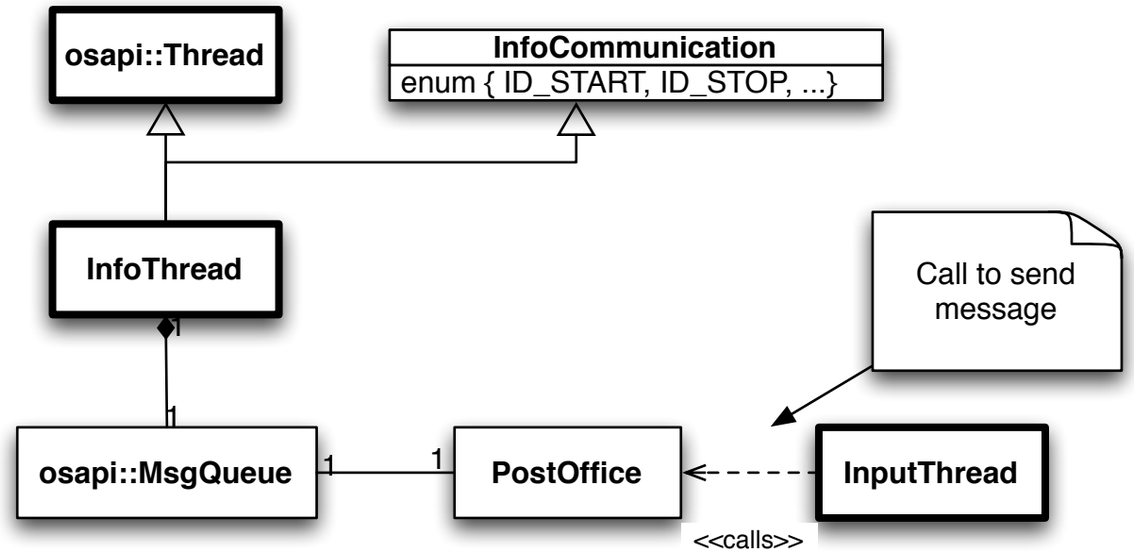
Design: Specific receiver



ENGINEERING COLLEGE OF AARHUS

Design: Specific receiver

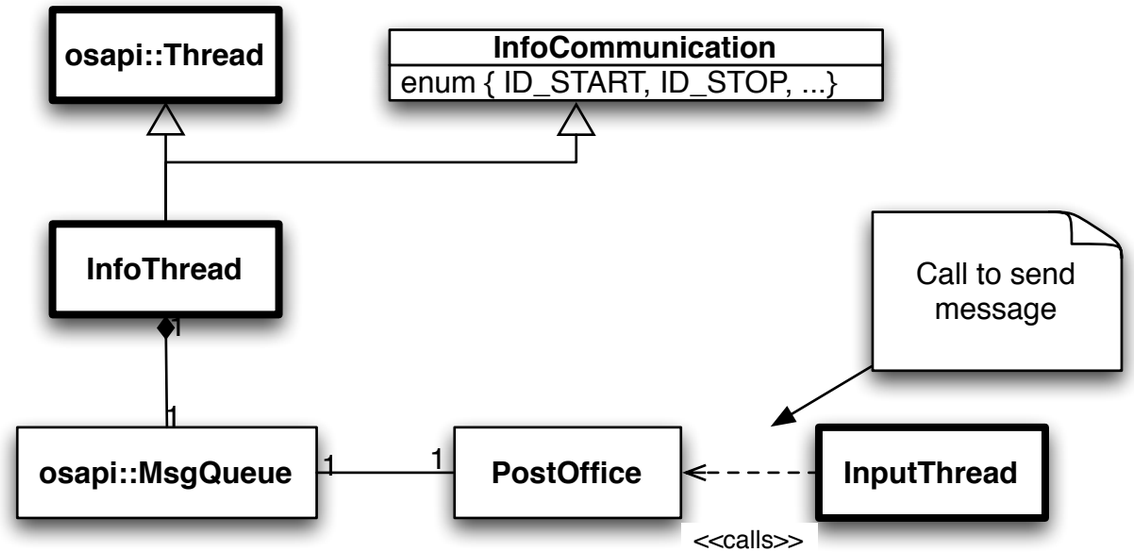
- Improve upon the include
  - Introduce another level



ENGINEERING COLLEGE OF AARHUS

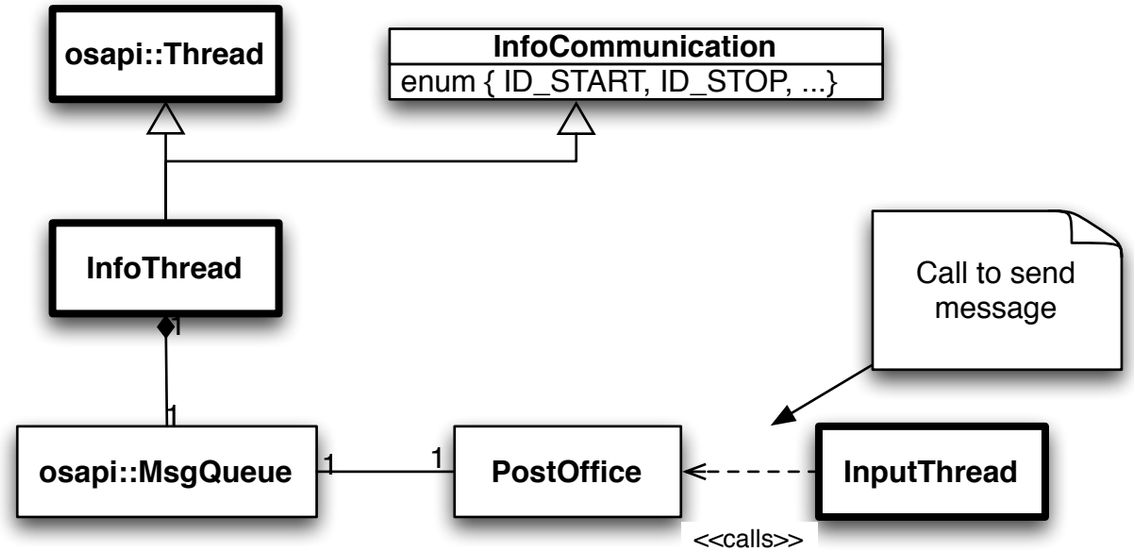
Design: Specific receiver

- Improve upon the include
  - Introduce another level
- Create a central postoffice
  - Send messages by *naming* (string format) the recipient
  - Or acquire a handle (speed up :-)



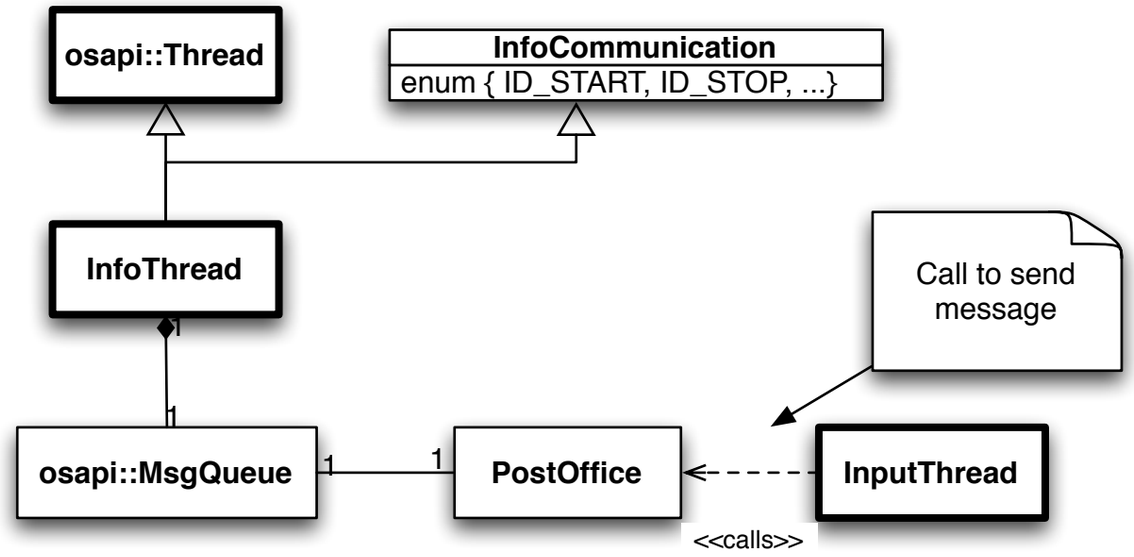
Design: Specific receiver

- Improve upon the include
  - Introduce another level
- Create a central postoffice
  - Send messages by *naming* (string format) the recipient
  - Or acquire a handle (speed up :-)
- Achieves
  - Low coupling since sender does not need to know receiver
  - Singleton usage or parsing around pointer/reference
  - Two-way communication possible
- Requires



Design: Specific receiver

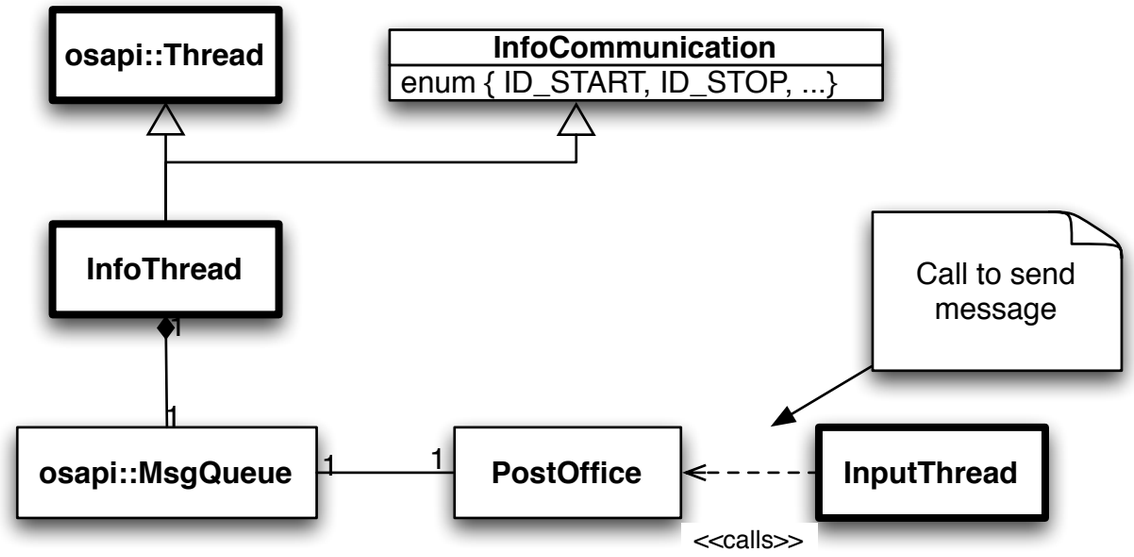
- Improve upon the include
  - Introduce another level
- Create a central postoffice
  - Send messages by *naming* (string format) the recipient
  - Or acquire a handle (speed up :-)
- Achieves
  - Low coupling since sender does not need to know receiver
  - Singleton usage or parsing around pointer/reference
  - Two-way communication possible
- Requires
  - A postoffice is up and running



ENGINEERING COLLEGE OF AARHUS

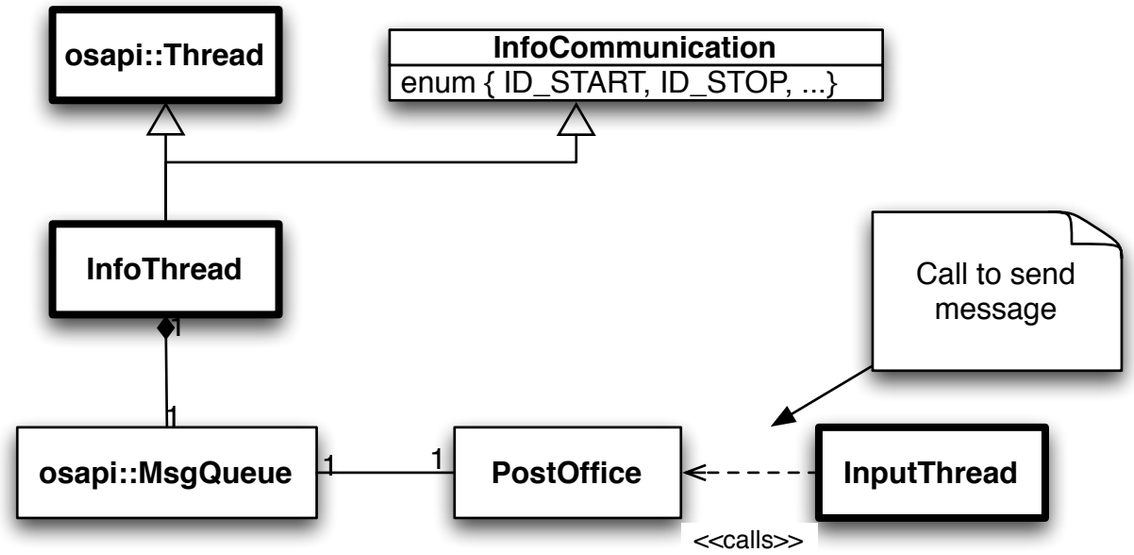
Design: Specific receiver

- Improve upon the include
  - Introduce another level
- Create a central postoffice
  - Send messages by *naming* (string format) the recipient
  - Or acquire a handle (speed up :-)
- Achieves
  - Low coupling since sender does not need to know receiver
  - Singleton usage or parsing around pointer/reference
  - Two-way communication possible
- Requires
  - A postoffice is up and running
  - Singleton usage or parsing around pointer/reference



Design: Specific receiver - Example

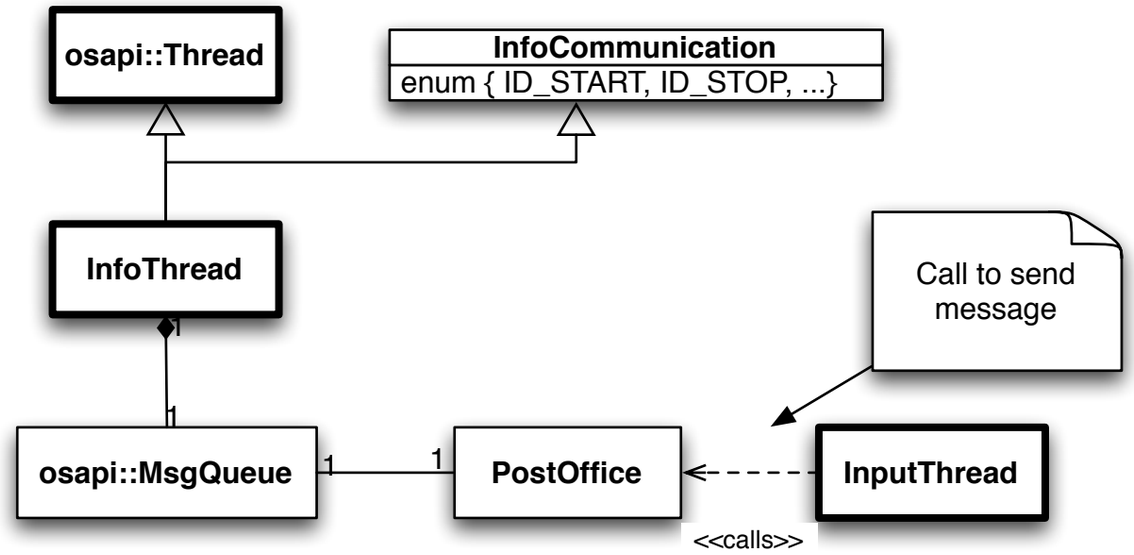
- Communication identification is done using a separate header file



ENGINEERING COLLEGE OF AARHUS

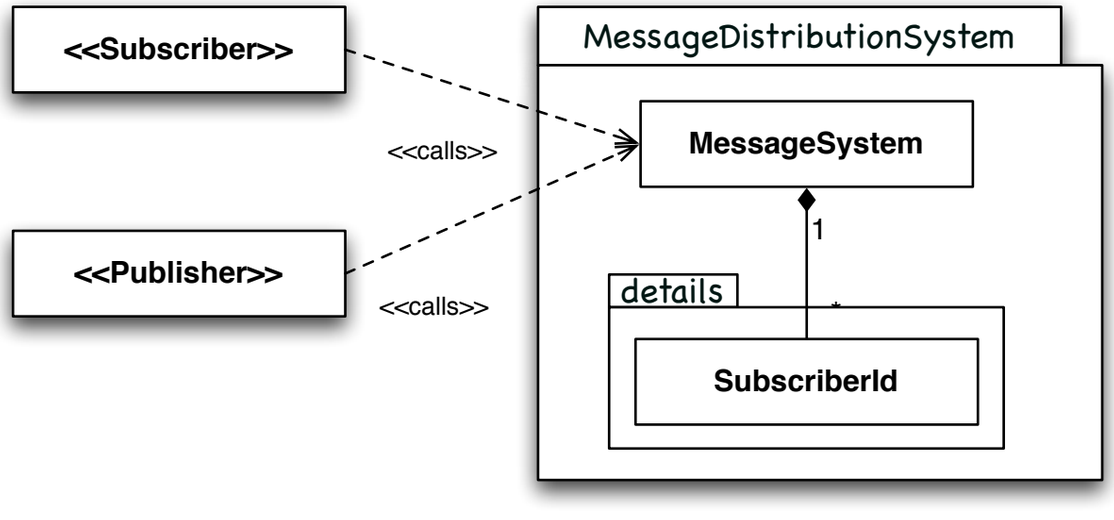
Design: Specific receiver - Example

- Communication identification is done using a separate header file



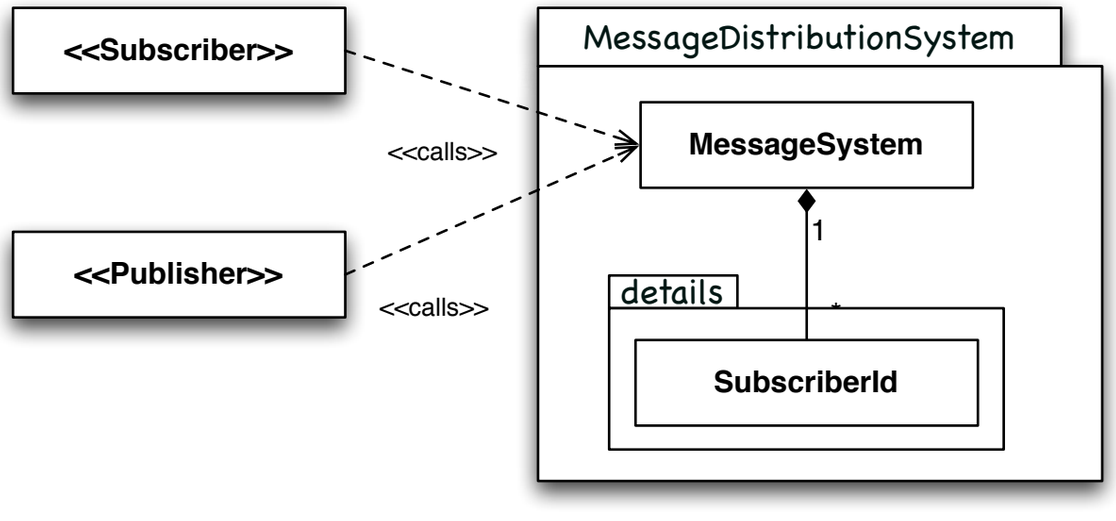

```
// InputCommunication.hpp
struct InputCommunication
{
    static const std::string QUEUE;
    enum { ID_START }
};

// In some thread...
#include "InputCommunication.hpp"

StartMsg* startMsg = new StartMsg;

PostOffice::send(InputCommunication::QUEUE, InputCommunication::ID_START, startMsg);
```

Design: Broadcasting - who the receiver is *is* irrelevant



Design: Broadcasting - who the receiver is *is* irrelevant

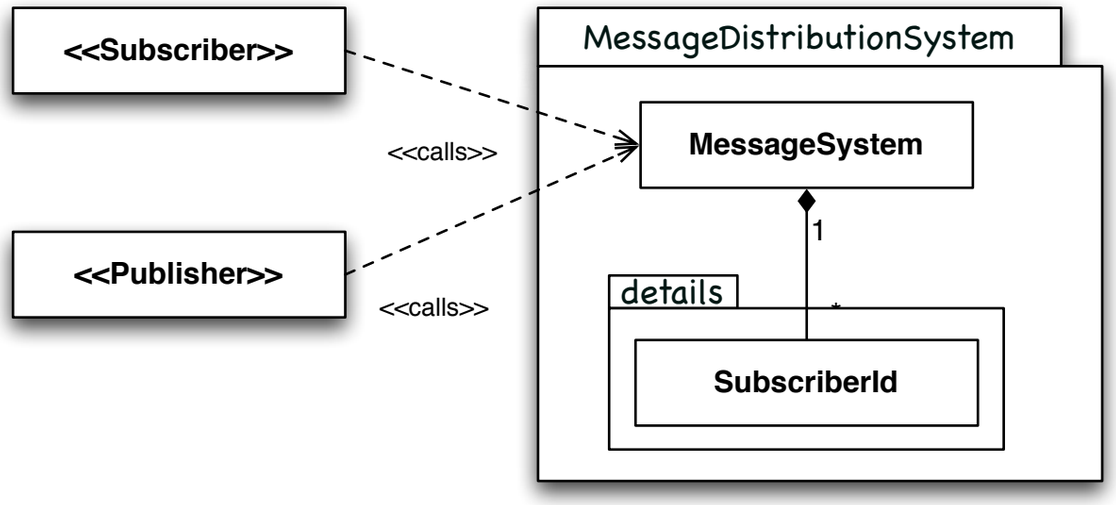
- Subscribes to a named message (std::string)
  - Using message queue pointer
  - ID to receive when a message is ready



ENGINEERING COLLEGE OF AARHUS

Design: Broadcasting - who the receiver is *is* irrelevant

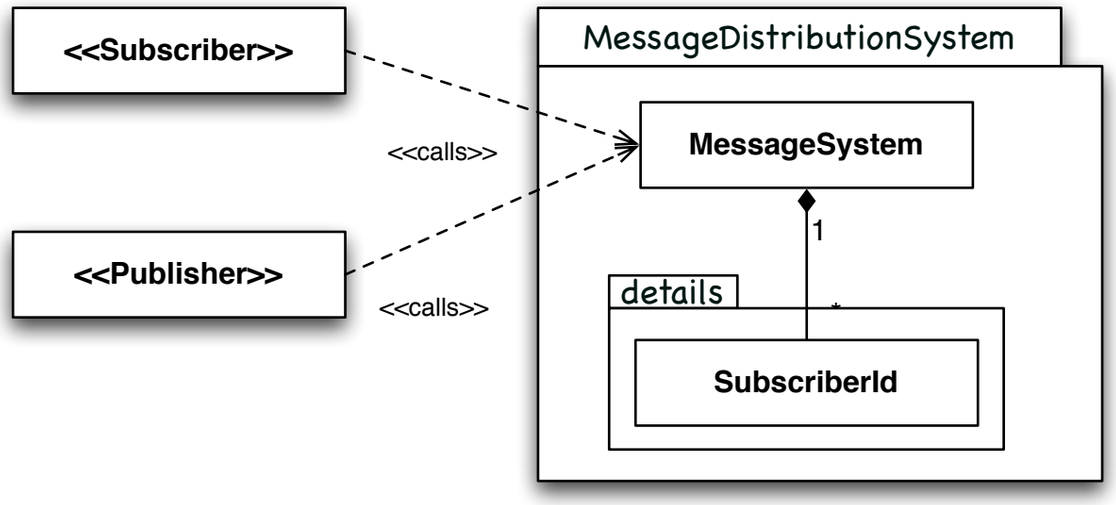
- Subscribes to a named message (std::string)
  - Using message queue pointer
  - ID to receive when a message is ready
- Publisher
  - Notifies all subscribers (if any), each will receive the message being distributed with their own desired id



ENGINEERING COLLEGE OF AARHUS

Design: Broadcasting - who the receiver is *is* irrelevant

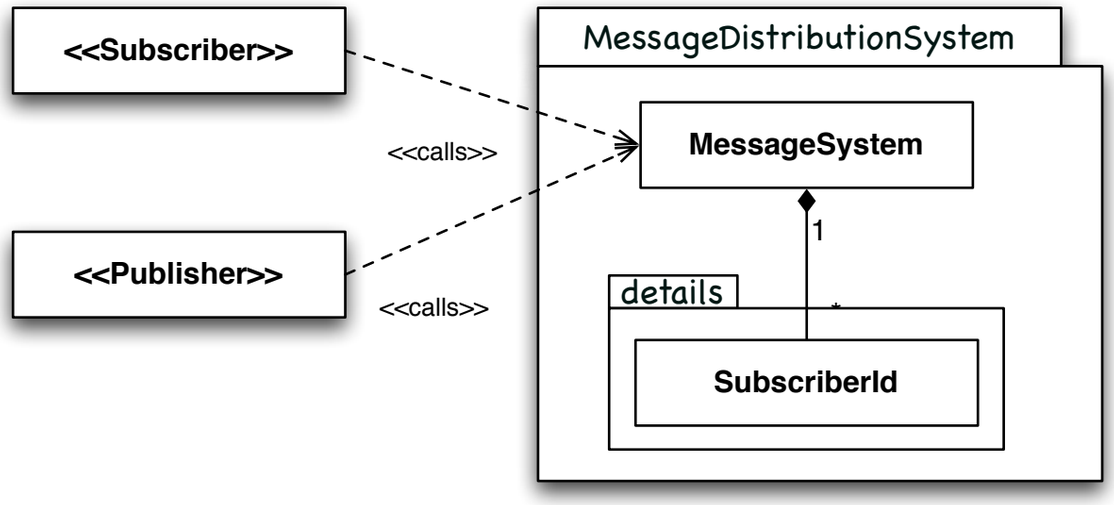
- Subscribes to a named message (std::string)
  - Using message queue pointer
  - ID to receive when a message is ready
- Publisher
  - Notifies all subscribers (if any), each will receive the message being distributed with their own desired id
- Achieves
  - Removes the need for the publisher to handle receiver(s)
  - Multiple receivers may get same message



ENGINEERING COLLEGE OF AARHUS

Design: Broadcasting - who the receiver is *is* irrelevant

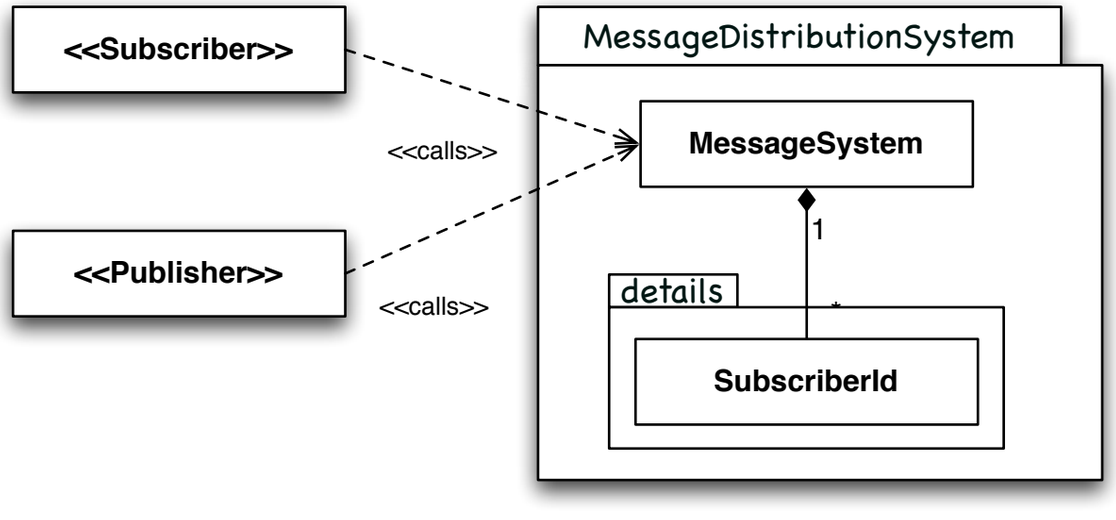
- Subscribes to a named message (std::string)
  - Using message queue pointer
  - ID to receive when a message is ready
- Publisher
  - Notifies all subscribers (if any), each will receive the message being distributed with their own desired id
- Achieves
  - Removes the need for the publisher to handle receiver(s)
  - Multiple receivers may get same message



- Requires
  - A MessageDistributionSystem is up and running prior to use
  - Singleton usage or parsing around pointer/reference
  - Message ***Globally*** identifiable by string
  - One way communication

Design: Broadcasting - who the receiver is *is* irrelevant Example

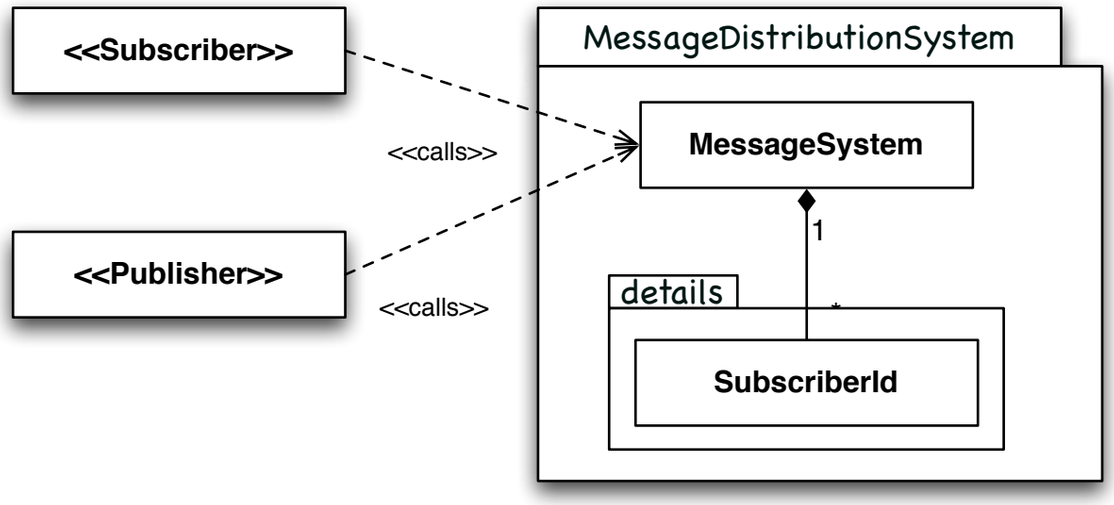
- Simple example using the MessageDistributionSystem directly



ENGINEERING COLLEGE OF AARHUS

Design: Broadcasting - who the receiver is *is* irrelevant Example

- Simple example using the MessageDistributionSystem directly




```
// Subscriber
MessageDistributionSystem::getInstance().subscribe(START_MSG, &mq_, ID_START);
...

void handleMsg(id, msg)
{
    switch(id_)
    {
        case ID_START:
            handleIdStart(msg);
    }
}

// Publisher
MessageDistributionSystem::getInstance().notify(START_MSG, startMsg);
```


ENGINEERING COLLEGE OF AARHUS

Design: Broadcasting - who the receiver is *is* irrelevant Example

- *Common* header file(s) contain message structures & declaration of globally string message ids
- Source file(s) contains the actual definition

Design: Broadcasting - who the receiver is *is* irrelevant Example

- *Common* header file(s) contain message structures & declaration of globally string message ids
- Source file(s) contains the actual definition

```
// hpp - file
struct StartMsg : public osapi::Message
{
    int x;
    int y;
};
extern const std::string START_MSG;

struct LogEntry : public osapi::Message
{
    char* filename_;
    int lineno_;
    std::string logStr_;
};
extern const std::string LOG_ENTRY_MSG;
```

Design: Broadcasting - who the receiver is *is* irrelevant Example

- *Common* header file(s) contain message structures & declaration of globally string message ids
- Source file(s) contains the actual definition

```cpp
// cpp - file
const std::string START_MSG = "StartMsg";
const std::string LOG_ENTRY_MSG = "LogEntryMsg";
```

```cpp
// hpp - file
struct StartMsg : public osapi::Message
{
    int x;
    int y;
};
extern const std::string START_MSG;

struct LogEntry : public osapi::Message
{
    char* filename_;
    int lineno_;
    std::string logStr_;
};
extern const std::string LOG_ENTRY_MSG;
```

Design: Broadcasting - who the receiver is *is* irrelevant Example

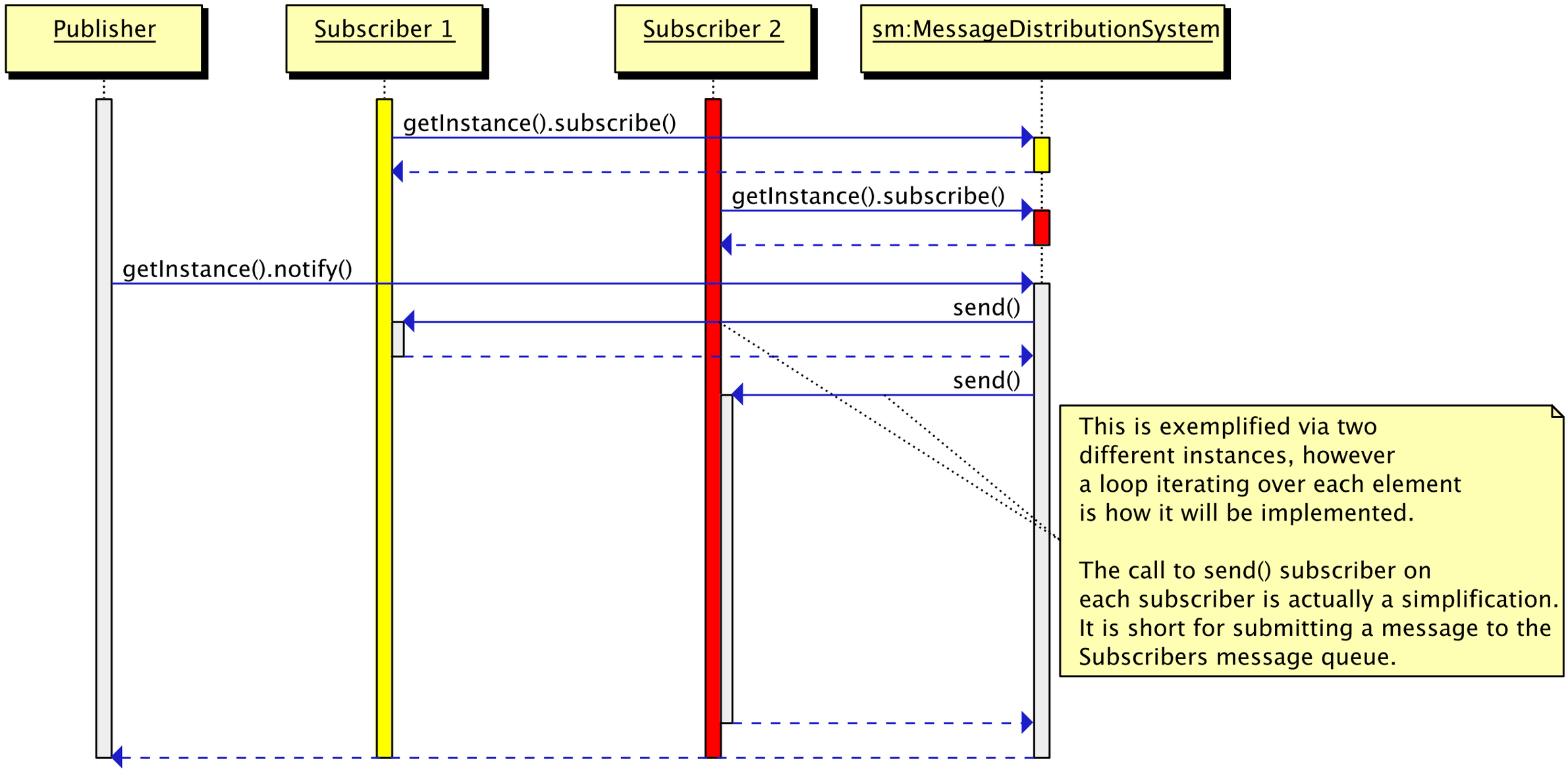
- *Common* header file(s) contain message structures & declaration of globally string message ids
- Source file(s) contains the actual definition

```cpp
// cpp - file
const std::string START_MSG = "StartMsg";
const std::string LOG_ENTRY_MSG = "LogEntryMsg";
```

```cpp
// hpp - file
struct StartMsg : public osapi::Message
{
    int x;
    int y;
};
extern const std::string START_MSG;

struct LogEntry : public osapi::Message
{
    char* filename_;
    int lineno_;
    std::string logStr_;
};
extern const std::string LOG_ENTRY_MSG;
```

```cpp
// Subscriber
MessageDistributionSystem::getInstance().subscribe(START_MSG,
                                                   &mq_, ID_START);
...
void handleMsg(id, msg)
{
    switch(id_)
    {
        case ID_START:
            handleIdStart(msg);
    }
}

// Publisher
MessageDistributionSystem::getInstance().notify(START_MSG, startMsg);
```

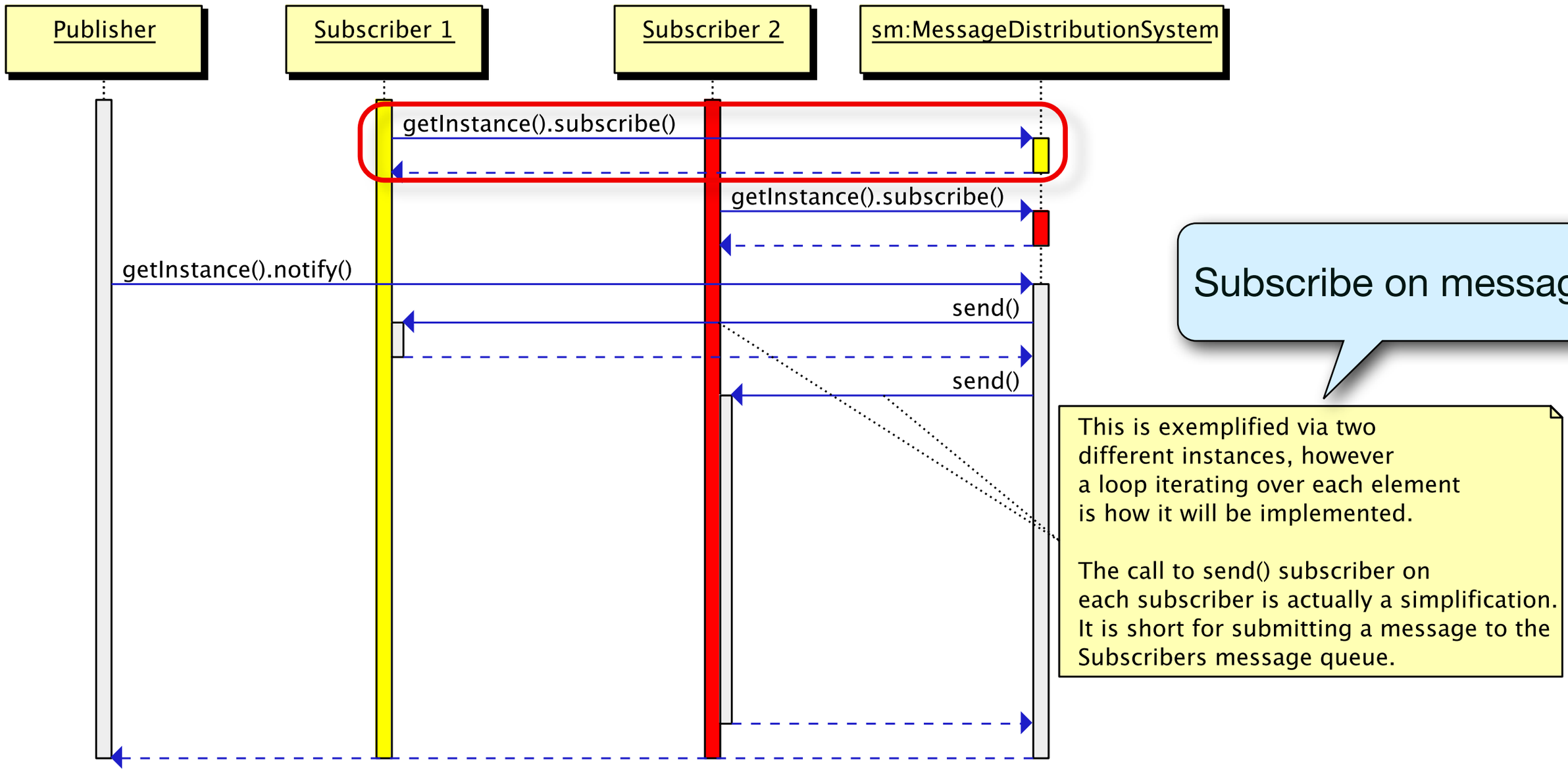
Message Distribution System in action



This is exemplified via two different instances, however a loop iterating over each element is how it will be implemented.

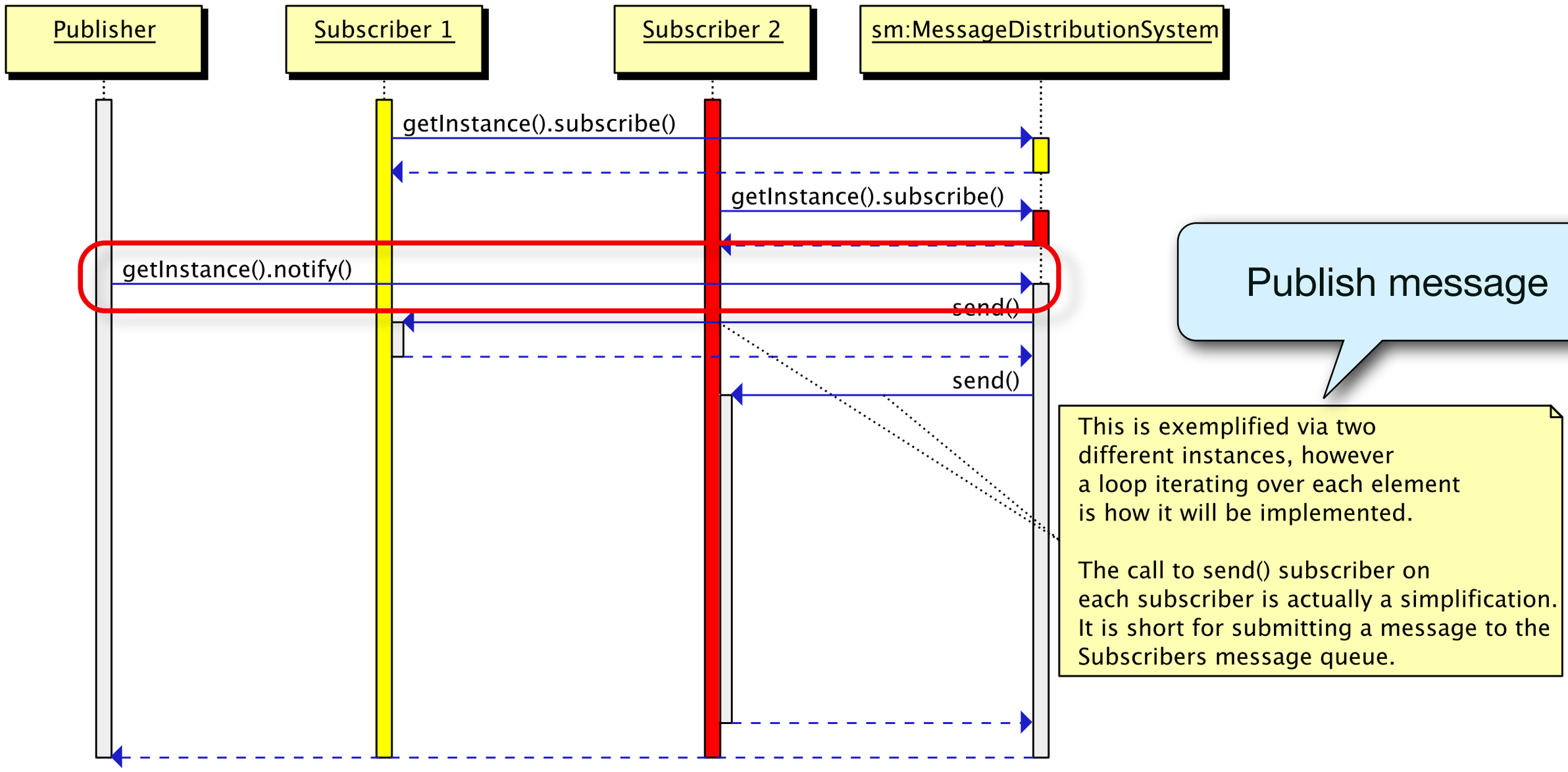
The call to send() subscriber on each subscriber is actually a simplification. It is short for submitting a message to the Subscribers message queue.

Message Distribution System in action



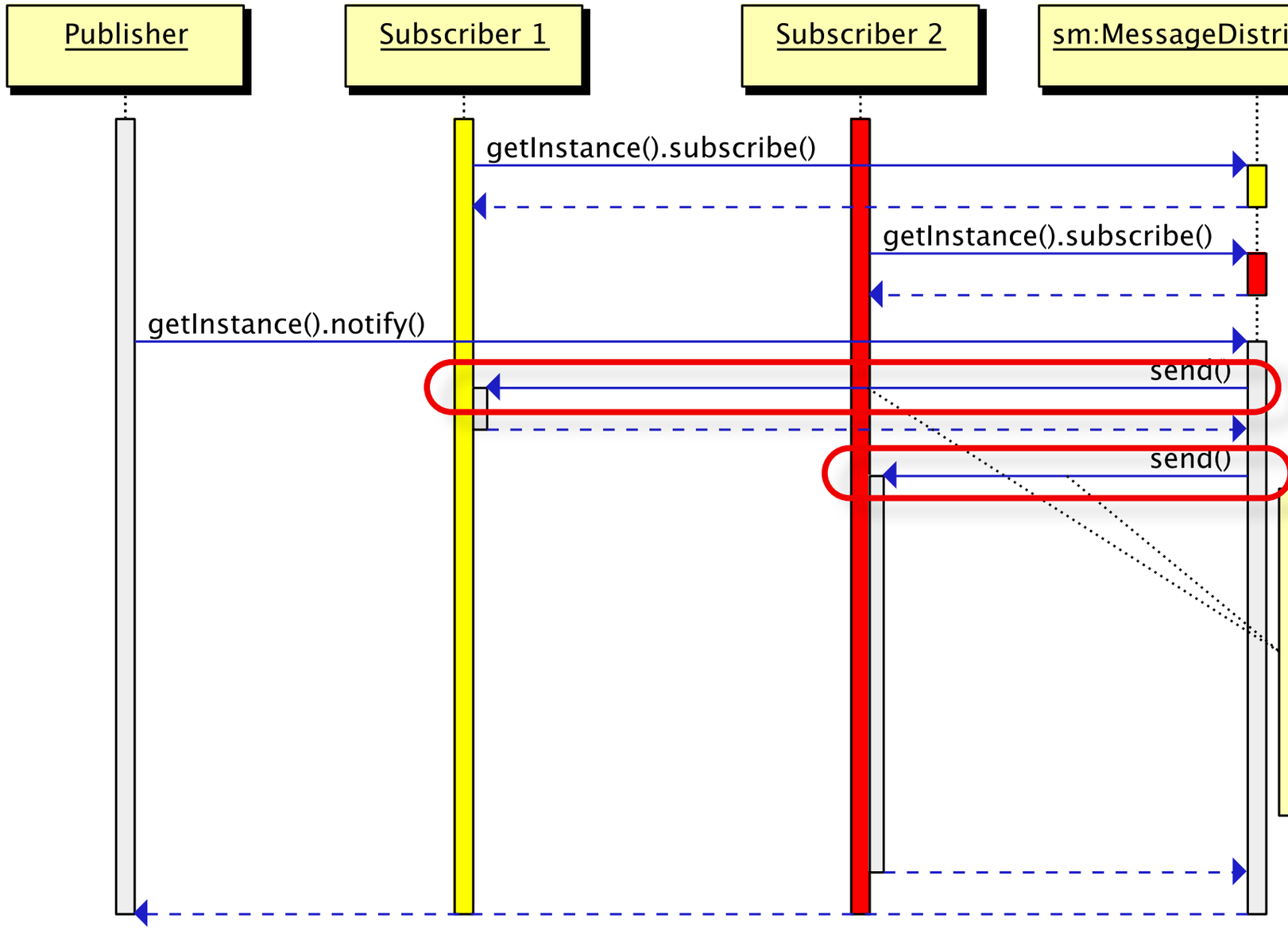
ENGINEERING COLLEGE OF AARHUS

Message Distribution System in action



ENGINEERING COLLEGE OF AARHUS

Message Distribution System in action



MSD notifies all

This is exemplified via two different instances, however a loop iterating over each element is how it will be implemented.

The call to send() subscriber on each subscriber is actually a simplification. It is short for submitting a message to the Subscribers message queue.

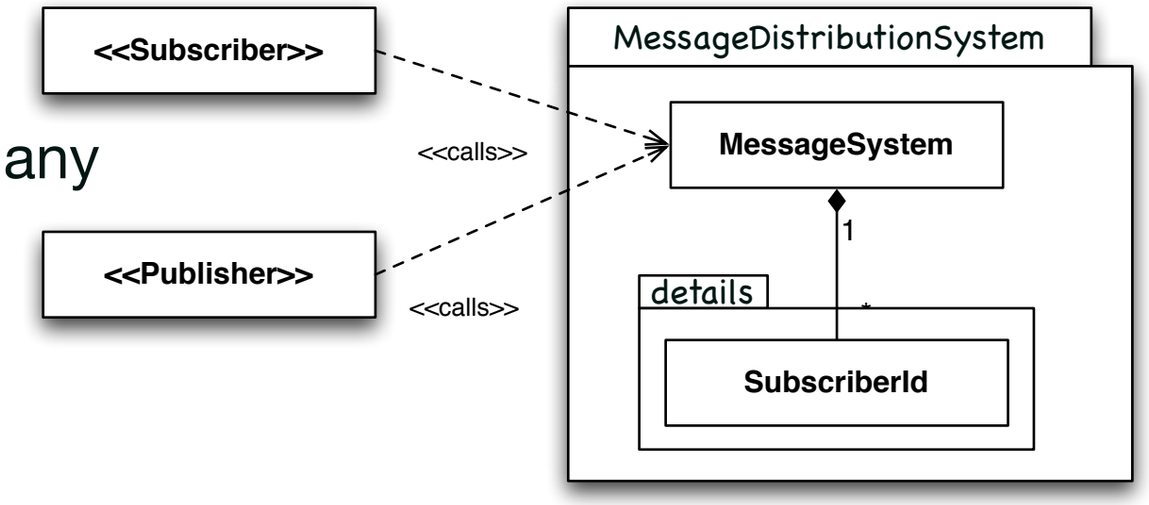
In this scenario - we choose...

- **Broadcasting - Who the receiver is *is* irrelevant**
  - One way communication
  - Knowledge of each other irrelevant
  - Lower coupling
- Usage scenarios
  - Indication that something has happened
    - Log entry
    - New temperature value

Singleton

Singleton pattern

- Challenge
  - System wide access to a given object ⇒Many pointers and/or references to be passed around
- Possible solution
  - ***Singleton***
- Usage could be
  - Message Distribution System
  - Config service
  - Log service
  - Any kind of application wide service



- Downsides
  - Global variable like
    - Serialized access needed
  - Lifetime
    - Who creates?
    - Who destroys and when?

Singleton pattern - Example

- Simple code example using the *static block initialization* approach
- Good
  - First access creates
  - Extremely easy to code and understand
  - No locks (in our approach)
- Downsides
  - First access creates - Multithreaded challenge
- Beware of “The double-checked locking” idiom
  - IT does not work!

Singleton pattern - Example

- Simple code example using the *static block initialization* approach
- Good
  - First access creates
  - Extremely easy to code and understand
  - No locks (in our approach)
- Downsides
  - First access creates - Multithreaded challenge
- Beware of “The double-checked locking” idiom
  - IT does not work!

```
// Singleton
class MessageDistributionSystem : osapi::NotCopyable
{
public:
    void subscribe(const std::string& msgId,
                   osapi::MsgQueue* mq, unsigned long id);
    void unSubscribe(const std::string& msgId,
                     osapi::MsgQueue* mq, unsigned long id);

    MessageDistributionSystem& getInstance()
    {
        MessageDistributionSystem mds;
        return mds;
    }

};

// Subscriber
MessageDistributionSystem::
       getInstance().subscribe(START_MSG, &mq_, ID_START);
```

Reflection - In perspective

Reflection - In perspective

- In relation to existing approaches - how does our choice fair?
  - Coupling
  - Publish/Subscriber schemes

- Group discussion followed by a plenum discussion

Decoupling

- Space decoupling
  - No references nor knowledge of each other
  - Number of subscribers not known
- Time
  - Do not need to be “online” at the same time
- Flow
  - Publishers are not blocked while creating messages
  - Subscribers do not *pull* new messages in a synchronously manner

Alternative schemes

- Topic-based
- Content-based
- Type-based

Topic-based scheme

- Idea
  - Identification/Key via a String
  - Topic abstraction
    - *topic == hierarchical addressing (recursive possible)*
  - Subscription via URL-like notation
    - “Msg:/system/input/*”
    - Possible to use wildcards
- Additional
  - Simplified Topic - Group abstraction
    - Group == flat addressing

Content-based scheme

- Idea
  - Identification/Key via properties (filtering)
    - String
      - name - value pairs - “device == ‘engine’ and temp > 90”
    - Template object
      - All attributes that match - wildcards are ignored
  - Executable code
    - Predicate implemented by developer
    - Created using *method value* pattern

Type-based scheme

- Idea
  - Identification/Key via type - E.g. you register a specific type and thus filter by it
  - Any instances of this type *or* derived instances will be received as well (recursive like)
  - Requires *static type system* - Types bound at compile time

Your task

- Consider the “The Message Distribution System” in groups of 2-3 for 5mins
  - Coupling wise?
    - Consider what we had and what we have now - What have we achieved?
  - Publish/Subscriber scheme variation
    - Which fits over approach best
- Discussion of what it is...

Summary

- Message Distribution System
- Singleton
- Reflection on our choice and setting it into perspective

Minute Paper

- Send me an email containing (deadline is today)
  - What you thought was important from todays lecture
  - What you didn’t quite get

- Do this in your respective groups
- I will collect and if needed elaborate on next meet