Algorithms and Distributed Systems 2023/2024 (Lab Two: Project Phase 1)

MIEI - Integrated Master in Computer Science and Informatics
MEI – Master in Computer Science and Informatics
Specialization block

Alex Davidson (a.davidson@fct.unl.pt)

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Class structure:

- Project Specification (Phase 1)
- The Babel Framework.

Project

- Research-oriented Project

- **Goal**: Write a Reliable Broadcast algorithm implementation that includes optimisations for propagating messages throughout the network

    1. Reliable broadcast based on gossip
    2. Anti-entropy optimisation for reducing number of messages in system
    3. HyParView optimisation to find peers more cleverly

- Your system should contain at least **100 processes**

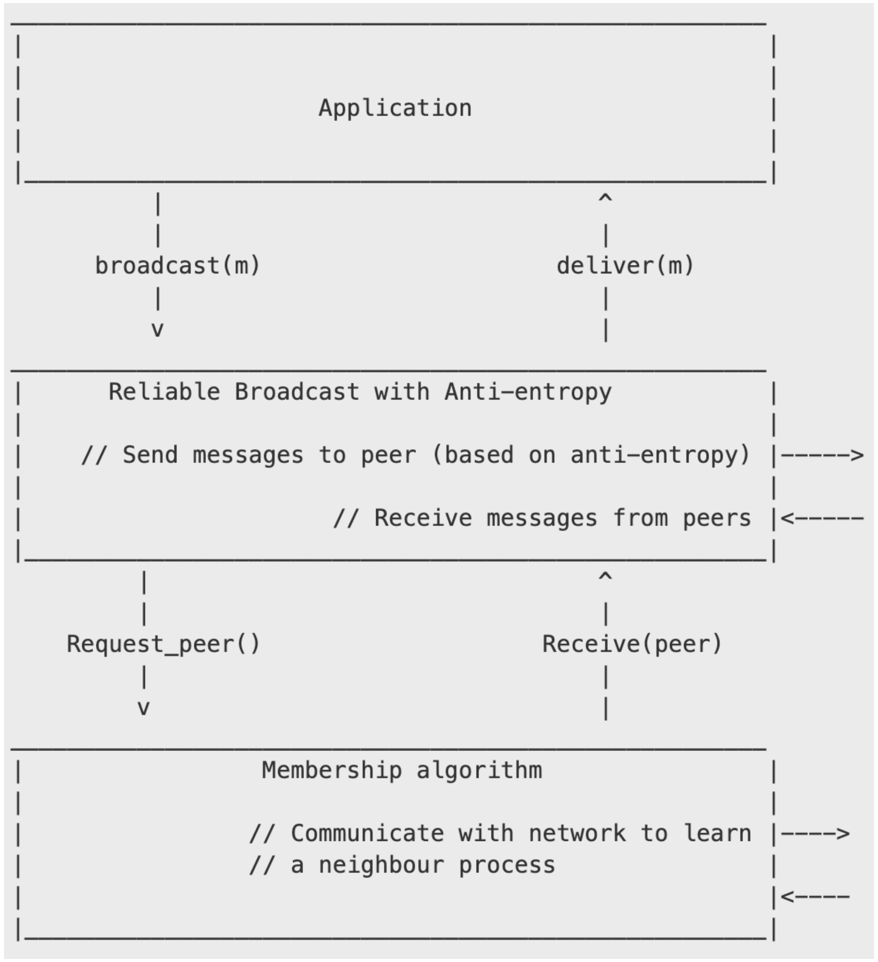
Project base solution

```
 ______________________________________________________
|                                                      |
|                                                      |
|                      Application                     |
|                                                      |
|______________________________________________________|
          |                               ^
          |                               |
      broadcast(m)                    deliver(m)
          |                               |
          v                               |
 ______________________________________________________
|                   Reliable Broadcast                 |
|                                                      |
|               // Send messages to gossip neighbours  |----->
|                                                      |
|          // Receive messages from gossip neighbours  |<-----
|______________________________________________________|
```

Should choose **t** gossip neighbours at random

Project optimisation #1

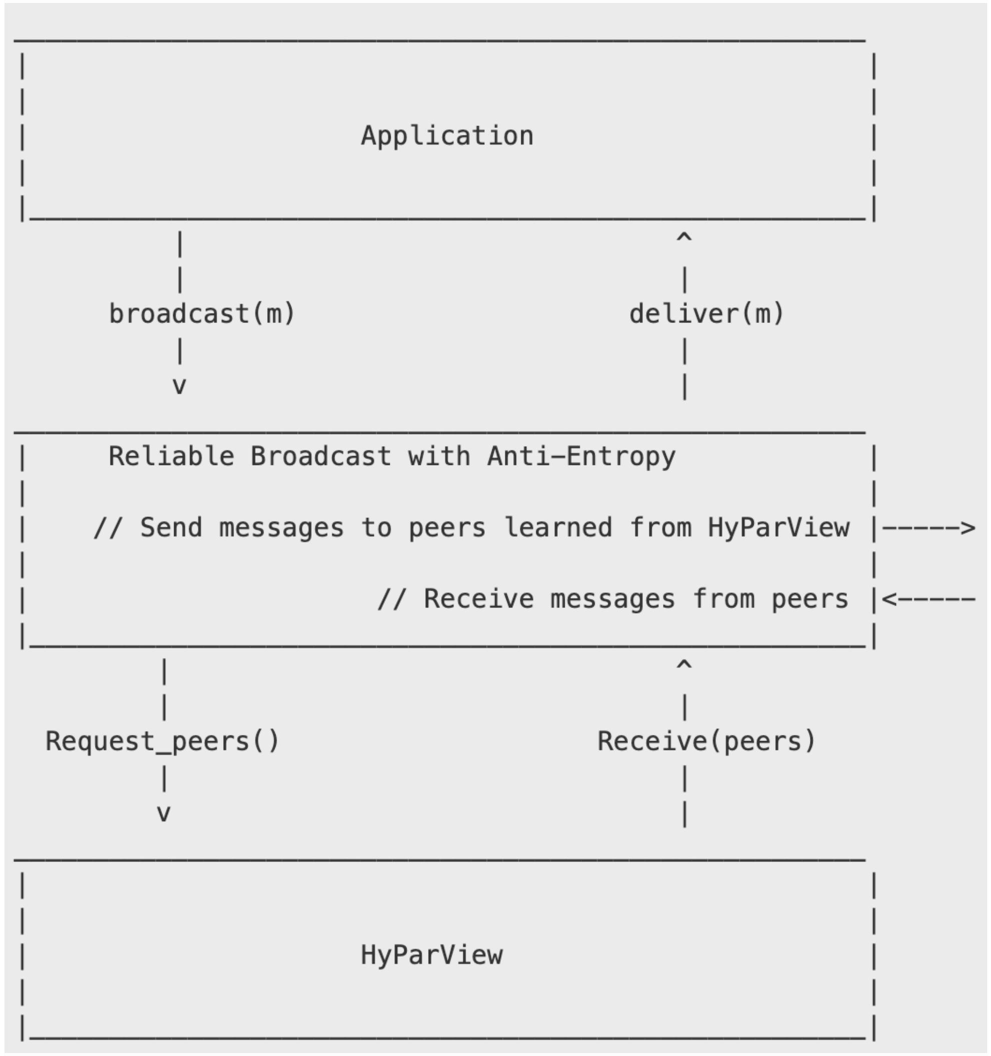
Should choose **1** neighbour at random to perform anti-entropy exchange



```
 ______________________________________________________
|                                                      |
|                                                      |
|                     Application                      |
|                                                      |
|______________________________________________________|
          |                                |
          |                                ^
          |                                |
      broadcast(m)                     deliver(m)
          |                                |
          v                                |
 ______________________________________________________
|      Reliable Broadcast with Anti-entropy            |
|                                                      |
|    // Send messages to peer (based on anti-entropy)  |----->
|                                                      |
|                      // Receive messages from peers  |<-----
|______________________________________________________|
         |                                 ^
         |                                 |
    Request_peer()                   Receive(peer)
         |                                 |
         v                                 |
 ______________________________________________________
|                Membership algorithm                  |
|                                                      |
|                 // Communicate with network to learn |---->
|                 // a neighbour process               |
|                                                      |<----
|______________________________________________________|
```

Project optimisation #2

Should choose **1** neighbour according to HyParView specification to perform anti-entropy exchange



```
 ______________________________________________________
|                                                      |
|                                                      |
|                     Application                      |
|                                                      |
|______________________________________________________|
          |                                ^
          |                                |
      broadcast(m)                     deliver(m)
          |                                |
          v                                |
 ______________________________________________________
|     Reliable Broadcast with Anti-Entropy             |
|                                                      |
|    // Send messages to peers learned from HyParView  |----->
|                                                      |
|                      // Receive messages from peers  |<-----
|______________________________________________________|
         |                                 ^
         |                                 |
   Request_peers()                    Receive(peers)
         |                                 |
         v                                 |
 ______________________________________________________
|                                                      |
|                                                      |
|                     HyParView                        |
|                                                      |
|______________________________________________________|
```

Base code

- https://classroom.github.com/a/QSCr0f1Y

- Application base code is offered for:
  - Generating and issuing requests (and logging them).
  - Receiving (and logging) replies.
  - Taking some metrics related with operation.

Conducting Experiments

- You will have access to the research computational cluster of DI and NOVA LINCS to conduct experiments for your project.
- The cluster **should not** be used to validate or test your implementations during development, only for conducting final evaluation.
- Groups must register so that an account is created (check the project definition in clip to see how)
- Each group will have a total budget of 3h x 2, so that they can run experiments with two machines for 3h.

Project: Conducting Experiments

- Information on how to use the cluster available at:

https://cluster.di.fct.unl.pt

- Online documentation explains how:
  - Login remotely using ssh (accessible from anywhere in the World)
  - Change Password
  - Create reservations to access computational machines
  - Cancel reservations
  - More…
- **Warning: You should never run your code on the frontend node to which you connect.**

Project: Conducting Experiments

Your experiments:

- Evaluate the performance of the two different system alternatives.
- Considering up to 4 combinations of application parameters:
  - Rate of Requests: slow and fast (i.e., 1 request 5 seconds VS 1 request every 500 milliseconds)
  - Size of content payload: small and big (i.e., 100b VS 1Mb)
- You should also measure (more if you want):
  - Reliability; Subscription latency; Publish latency; Number of messages/bytes sent; Number of messages/bytes received.

Project: Written report

- Short academic paper format.
- Up to 8 pages (including figures and tables, but excluding bibliography) with 10pt font and two column.
- **Highly suggested** that you use Latex (a template is provided in the repository).
- Pseudo-code describing your implementations should be present.
- Correctness arguments of implemented components is evaluated positively.
- You should present experimental results in a clear way, and discuss observed results.

Project: Evaluation Rules

- Everything is considered: correctness of the implementations, efficiency of the implementations, code quality, quality of the written report (up to a total of 20 points).
    - You will get at most 12 points if you implement a Reliable Broadcast algorithm (using an epidemic/Gossip model) and experimentally evaluate those protocols with a single set of experimental parameters.
    - You can get an additional 4 points if you implement the Anti-Entropy and HyParView optimisations and conduct experiments.
    - You can get an additional 4 points if you consider *dynamic memberships of processes, and* conduct experiments using multiple ranges of input parameters (see the project description in the README.md).

Babel

- Internal framework developed at NOVA LINCS.
- Developers: Pedro Fouto, Pedro Ákos Costa, João Leitão.
- Example Babel code provided in \`src/\` folder of repository

Babel: a bird's-eye view

- You implement protocols, where each protocol is a Java class (that extends an abstract class).
- Protocols are implemented by having:
  - A constructor that describes the types of events that are consumed by the protocols and the types of messages used in the protocol (and how they are serialized).
  - An **init** handler that receives initialization parameters with relevant configuration parameters for the protocol.
  - Handlers for all types of events that the protocol consumes.

Babel: Types of Events

- In Babel protocols can consume the following types of events:
  - Messages (the only event that can be sent through the network among processes)
  - Timers
  - Requests (that can generate Replies)
  - Notifications

- Each type of event has an abstract class associated with it. You define a new type of event by extending the appropriate abstract class.

Babel: Communication

- Similar to protocols in pseudo-code, protocols in Babel should only send (and receive) messages to and from the same type of protocols in other processes.

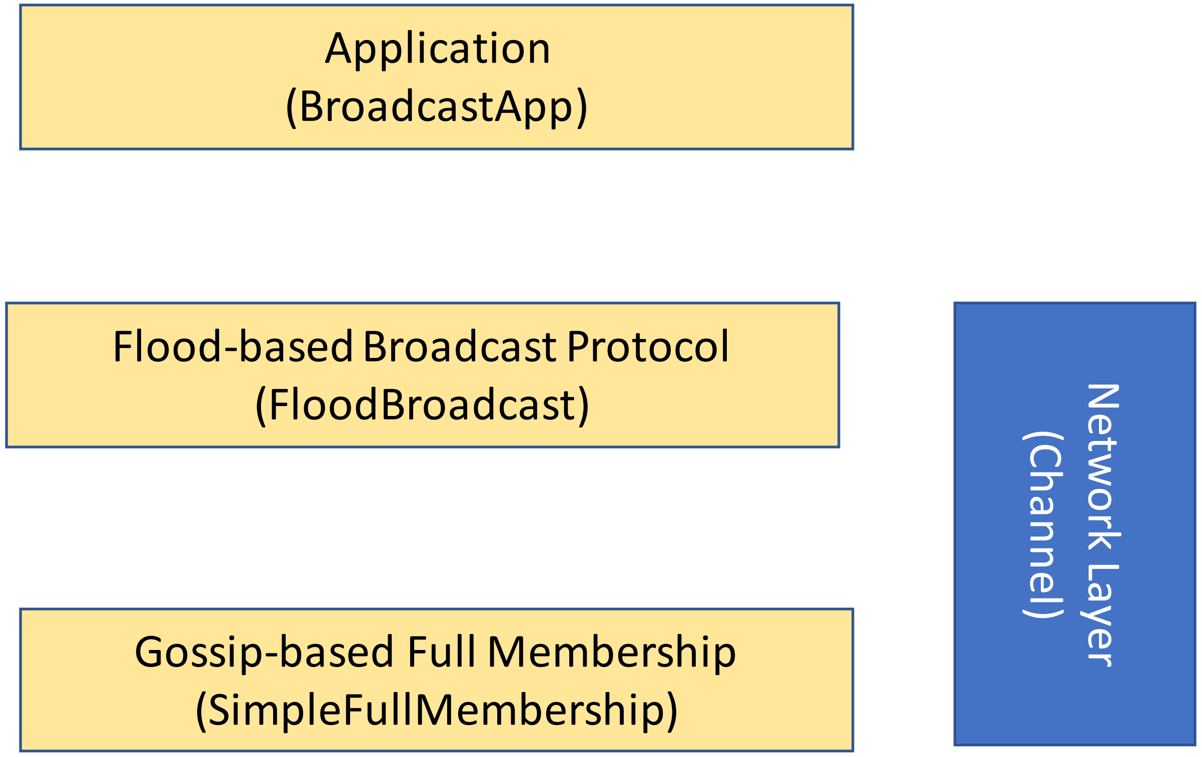
- Messages are exchanged using an abstraction called Channel.
- The Channel that you are going to use in the project requires you to explicitly open and close TCP connections. It will also issue notifications to your protocol for it to be aware when these events happen.

Babel: Example

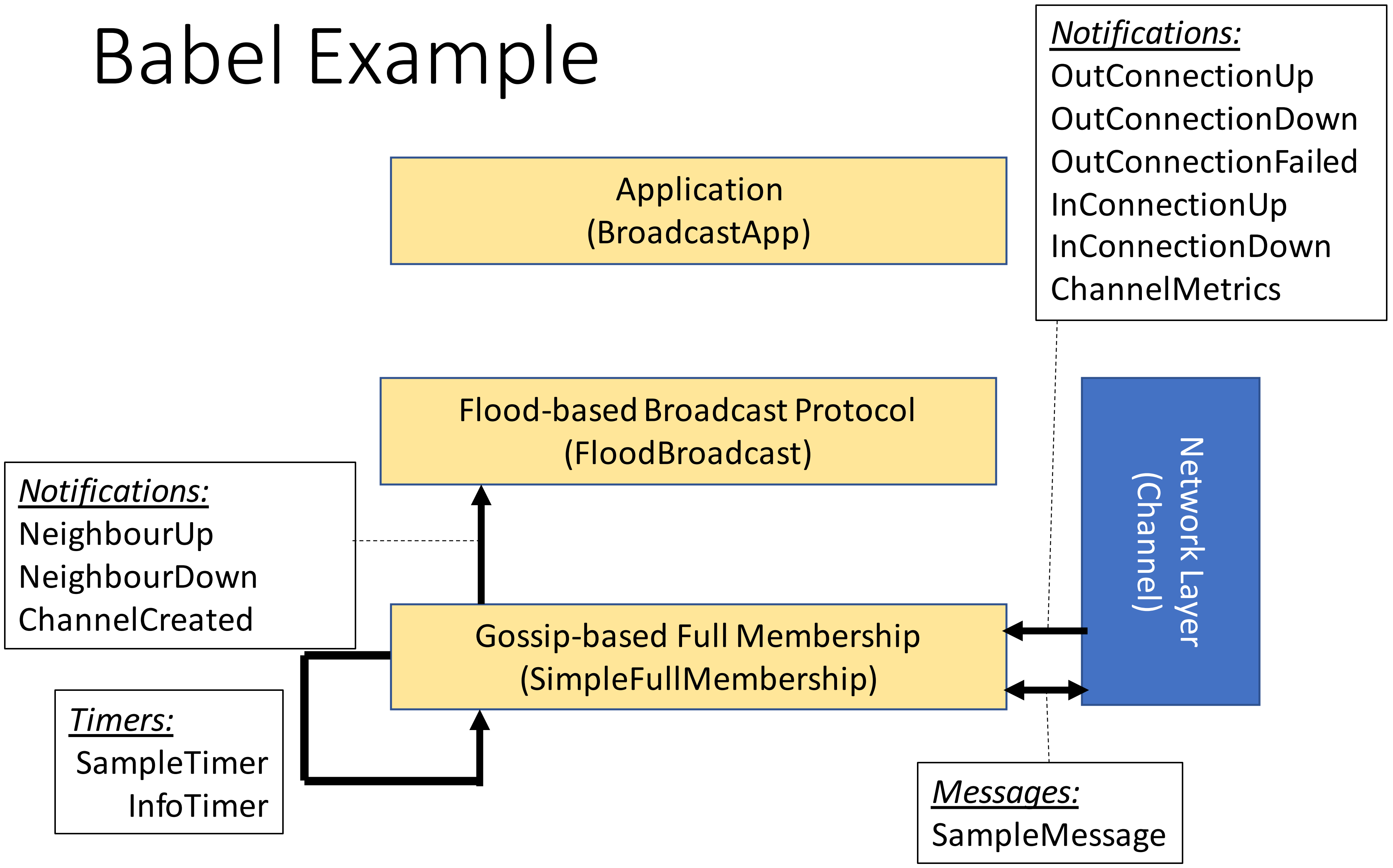
- An Example project is available in the GitHub repository.

- The example contains the following components:
  - It features an application that broadcasts and receives messages.
  - It features a simple flood broadcast protocol.
  - It features a gossip-based full membership protocol.

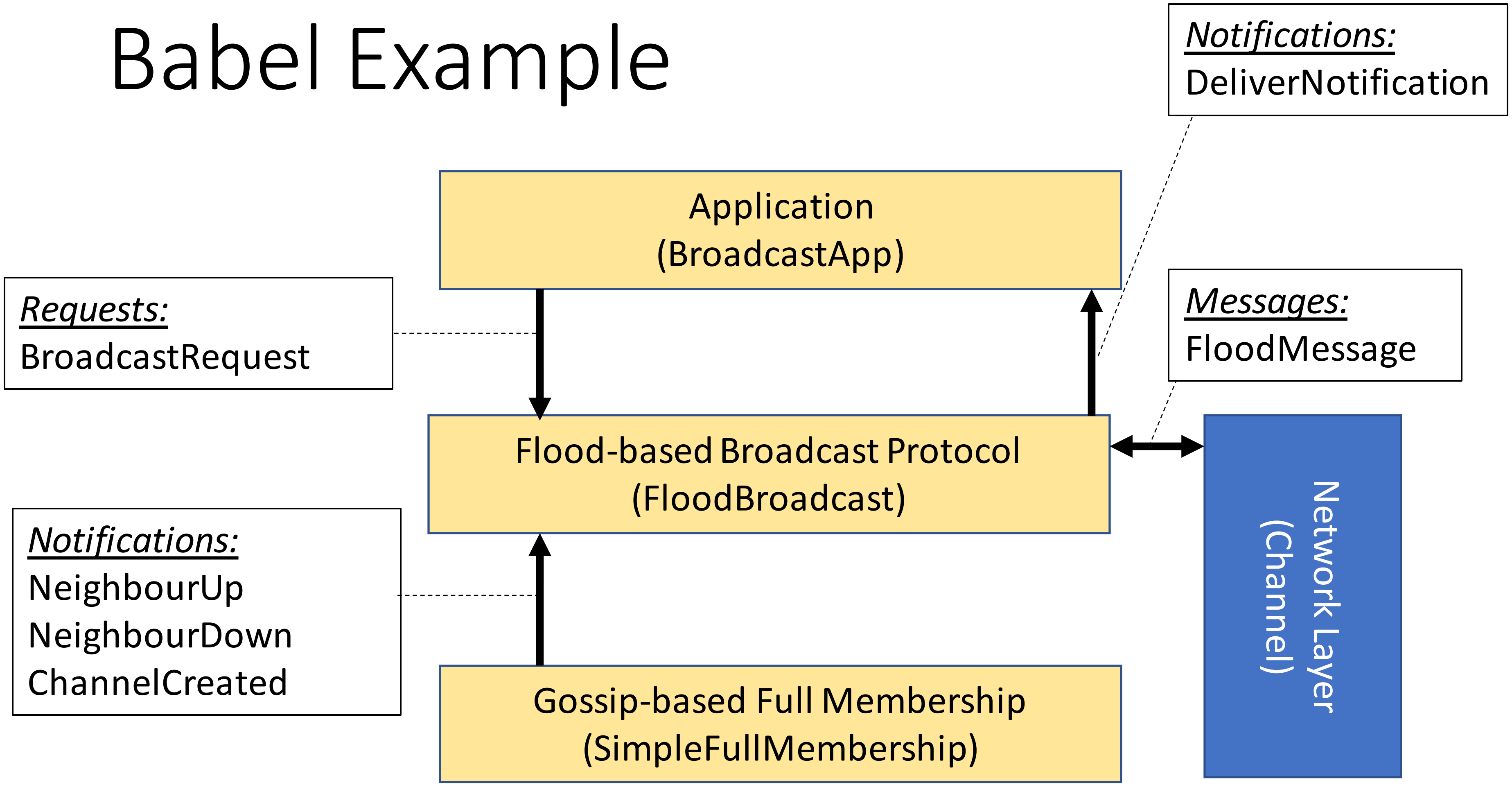
Babel Example



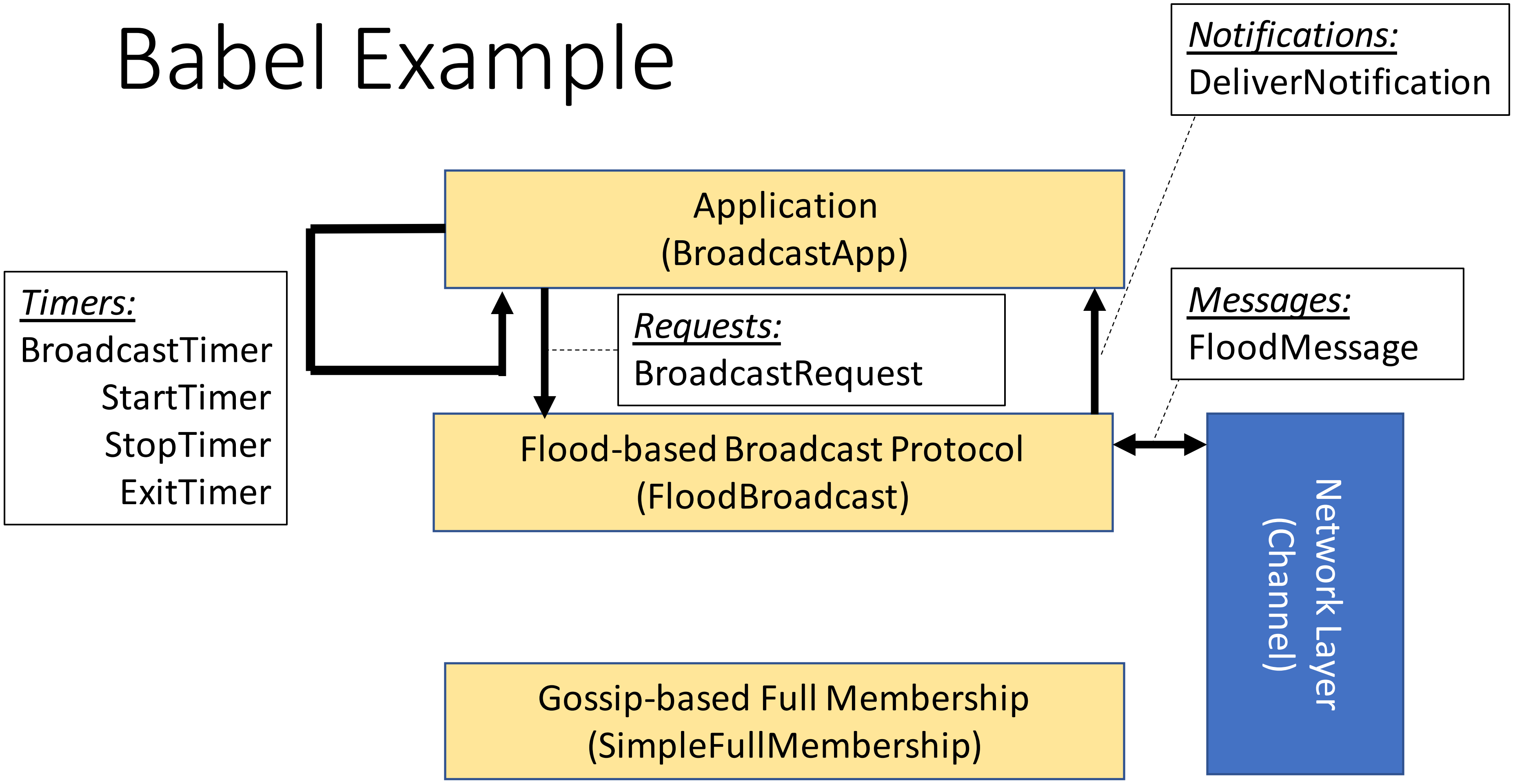
Babel Example



Babel Example



Babel Example



How to run the example

- First step compile and package the project with maven (in the directory with the pom.xml):

```
mvn compile package
```

- This will generate a jar file with all dependencies named **asdProj.jar** in directory **target**.

How to run the example

- In two different windows run the following commands in order:

```
java -cp target/asdProj.jar  Main interface=eth2  port=10101

java -cp target/asdProj.jar  Main interface=eth2  port=10102
contact=10.139.24.136:10101
```

How to run the example

- In two different windows run the following commands in order:

```
java -cp target/asdProj.jar  Main interface=eth2  port=10101

java -cp target/asdProj.jar  Main interface=eth2  port=10102
contact=10.139.24.136:10101
```

This depends on the interface that your machine has, in mac you can use “ifconfig” to check this, in linux “ip addr”, and windows “ipconfig”

How to run the example

- In two different windows run the following commands in order:

```
java -cp target/asdProj.jar Main interface=eth2 port=10101

java -cp target/asdProj.jar Main interface=eth2 port=10102
contact=10.139.24.136:10101
```

Processes in the same machine must use different ports. This is the port where the process will wait for TCP connections.

How to run the example

- In two different windows run the following commands in order:

```
java -cp target/asdProj.jar Main interface=eth2 port=10101
```

```
java -cp target/asdProj.jar Main interface=eth2 port=10102
contact=10.139.24.136:10101
```

All processes with the exception of the first must have an additional parameter with the contact node, whose value is IP:PORT of the first process.

Final notes

- Classroom link: https://classroom.github.com/a/QSCr0f1Y
- Babel Javadoc: https://asc.di.fct.unl.pt/~jleitao/babel/

First step

Use the flood-based broadcast protocol in the provided example as a template to create a Gossip-based protocol where instead of flooding the network you send the message only to t neighbors at random.

Second step

Work towards the intended optimisations and experimental framework