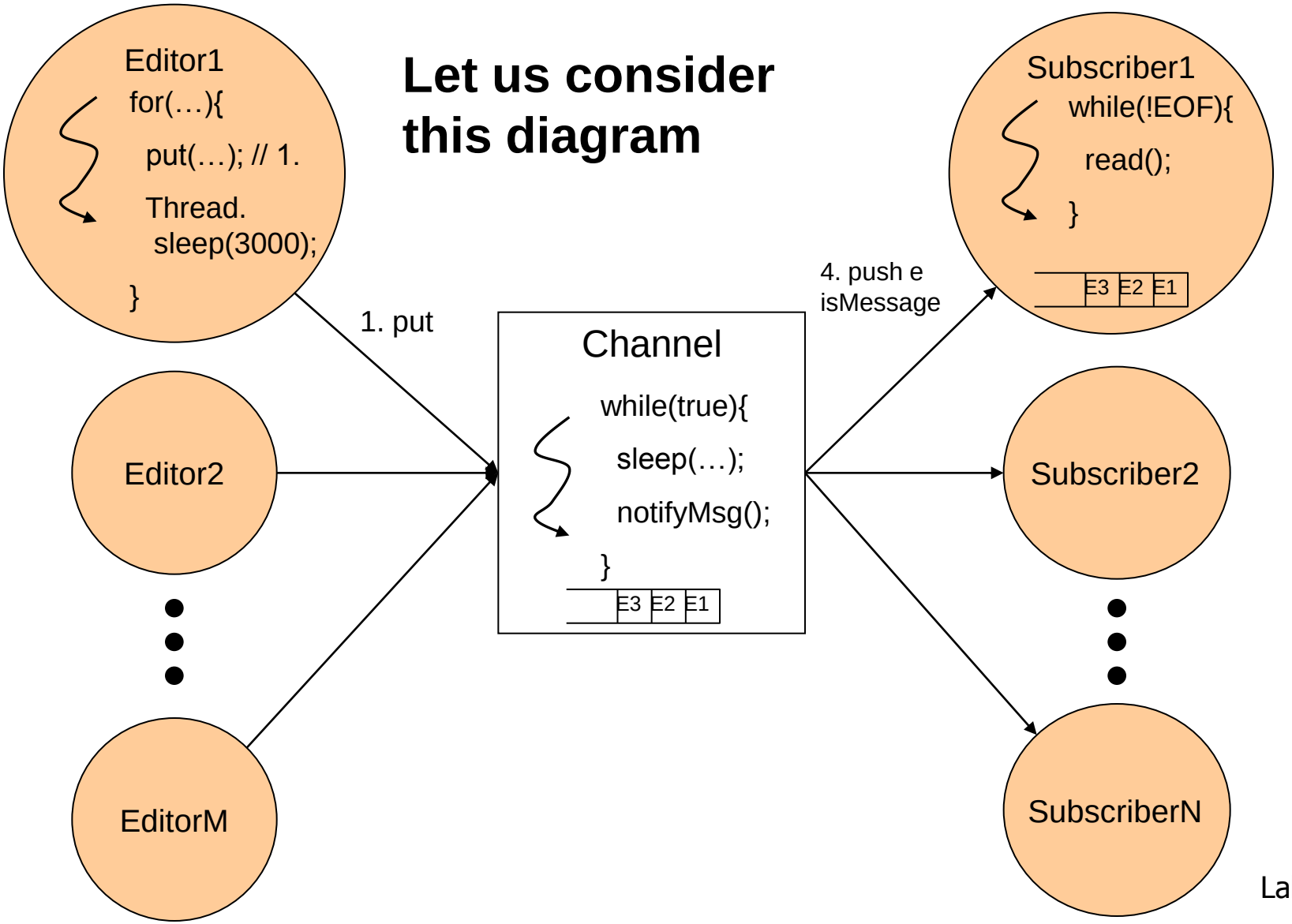
Università degli Studi di Bologna
Facoltà di Ingegneria

Principles, Models, and Applications for Distributed Systems M

Lab assignment 1
Java Multithreading

Luca Foschini

Reference architecture



Assignment (step-by-step): Step 1

Change the editor to allow multiple editors

add a new field to be used as Editor identifier, as already happens for Subscribers

- **Remove the input from console generation**, replacing it with automatic generation of strings. The new Editor executes N_PUBBLICATIONS, for example 7: each publication (String) contains **the identifier of the publisher** and **the number of the current iteration**; for example: "E1, publication 1".
- **The time** between one generation and one another **is 3 seconds.**
- **Modify the test program** and see what happens when several editors use the same channel.

Assignment (step-by-step): Step 2

Change the Channel architecture in order to decouple the event generation time and the event notification time.

Redefine the Channel as an **autonomous and active entity** to better separate other application entities.

The final operation to be obtained is the following: Publishers publish events, generated asynchronously, to the Channel and the **active** Channel, every 2 seconds, **notifies all the events received by editors during the last period** (i.e. 2 seconds) **to all registered subscribers**.

Assignment (step-by-step): Step 2 – Details

Let us note that to obtain the desired behavior it is necessary to change the Channel by making it ***an active process (a Java Thread)*** *and* ***by adding an event queue****.*

With a closer view to implementation details:

- The **Channel** is a thread and realizes the behavior described in the previous slide. The event queue is a String array with MAX_EVENTS elements (`String[MAX_EVENTS]`). Along with the updates of that data structure you will need to update *a counter (realized as an int*) that indicates the array filling level.
  Note: *the data structure containing the event has finite size*. Hence, the *put* in method should suspend when event queue full of the Channel (by using a `wait()`). In other words, Editor threads will suspend until there is (again) room in the queue.
- The **Editor** is the same as described in step 1. Let us note that, compared to the Editor designed in the worked-out assignment, this Editor *does not* publish the EOF, because the interaction with the human user has being replaced by an automatic generation.
- The **Subscriber** is unchanged: it prints to standard output (screen) a number of events, e.g., MAX_EVENTS, and then it ends.