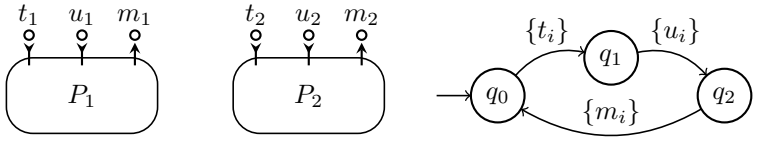
# Excercise for Coordinating Multicore Computing

In this exercise, we implement a small program that allows two players to play a small turn taking game. We model each player as a component $P_i$ shown in the figure below. The automaton describes the behaviour of the component $P_i$. Each component $P_i$ has three ports: input $t_i$ for determining when it is his turn,




output $m_i$ for executing a move, and input $u_i$ for receiving an update of the current state of the game.

**Set up Eclipse.** Download "Eclipse IDE for Java Developers" at: `http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/lunasr2`. Extract the archive and run Eclipse. Go to "Help" → "Install New Software...", and enter `http://www.cwi.nl/~jongmans/bertinoro` in the "Work with" field. Untick "Group items by category". Tick "Reo Tools" and "ReoText Tools". Complete the wizard (and restart Eclipse at the end).

**Create a new project.** Go to "File" → "New" → "Java Project", and complete the wizard (use a JRE for Java 7 or higher). In the Package Explorer (typically the left panel in the window), right-click the project just created → "New" → "Other...", then insert "Reo" as the filter text, then select "Reo Connector", and complete the wizard.

**Design your connector.** Draw a connector, and add the six nodes $t_1$, $t_2$, $m_1$, $m_2$, $u_1$, and $u_2$ to it. Design a Reo circuit that connects these ports such that if the players $P_1$ and $P_2$ are connected, then the system behaves 'naturally'.

**Animate your connector.** Go to "Window" → "Show View" → "Other", then insert "Reo" as the filter text, then select "Reo Animation". Enable guided animation to simulate a step by step execution of your connector.

**Compile your connector.** Go to "Window" → "Show View" → "Other", then insert "Reo" as the filter text, then select "Reo/ReoText Compiler". In the "Reo/ReoText Compiler" view just opened, click "Text" to convert the active Reo diagram to ReoText and click "Java" to generate code. If the "Run after compilation" box is ticked (at the bottom of the "Reo/ReoText Compiler" view, ticked by default), Eclipse immediately runs the generated code, and for every boundary node, a console should appear.

**Create the players.** In the Package Explorer, right-click "src" → "New" → "Class" and complete the wizard. Make sure that you put the new class outside the default package (i.e., put it in a package with a nonempty name). Then add the following method to this class.

```
public static void Player(OutputPort turn, OutputPort update, InputPort move, String name) {
   for (int i = 0; i < 4; i++) {
      System.out.println(name + " waits for his turn.");
      turn.getUninterruptibly();
      System.out.println(name + " waits for the last move.");
      Object lastmove = update.getUninterruptibly();
      System.out.println(name + " reads " + lastmove + " and tries to move.");
      move.putUninterruptibly("move" + i + ":" + name);
   }
   System.out.println("Player1 stopped playing.");
}
```

**Connect the players.** Drag-and-drop the class just created onto the canvas of the .reo file and select the method. Repeat this for the second player.

NOTE: This works only if (i) the .reo file and the .java file are in the same project, *and* (ii) the project is a Java project. Connect the sender and the receiver to the connector in the obvious way, generate code, and run (automatically). Make sure that both players follow the rules of your connector and that they terminate.