Using the conventional tools...

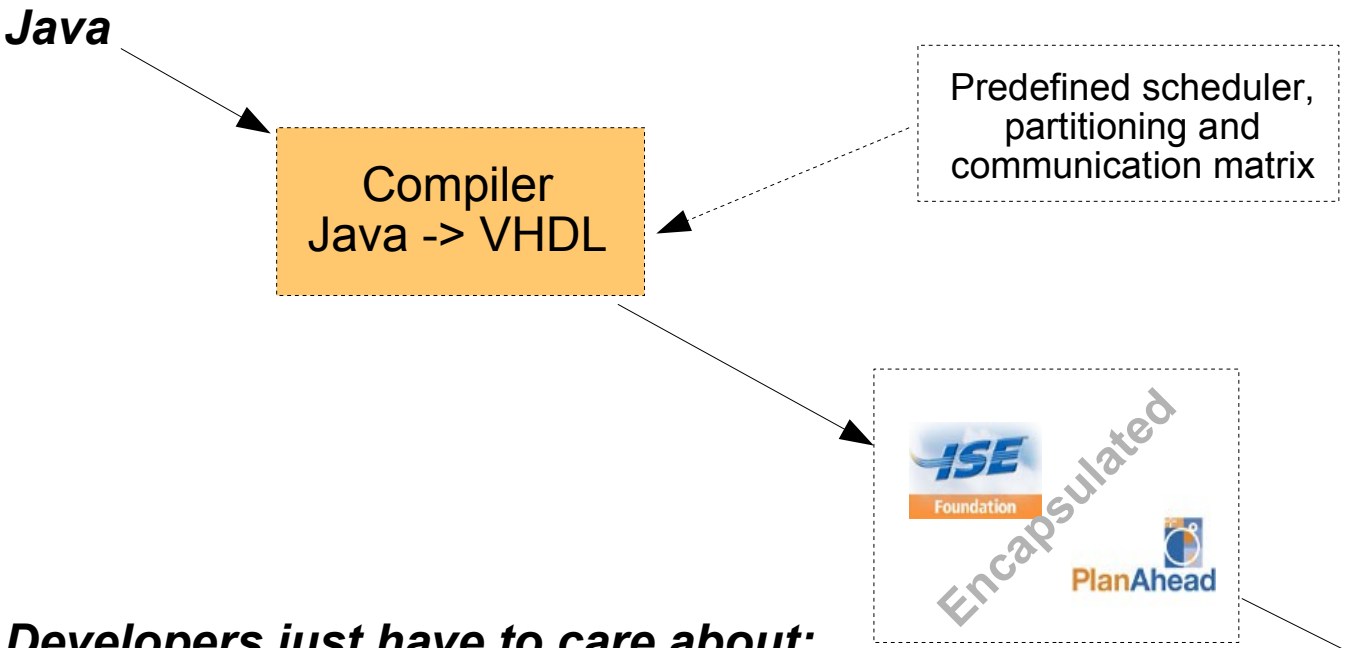
VHDL

Developers have to care about:

- Development of the dynamic and the static modules in VHDL
- Partitioning of the design in VHDL
- Implementation of the scheduler
- Logical implementation of the inter module communication (IMC)

- Partitioning of the Chip
- Physical implementation of the IMC

Using our DPR-Framework...

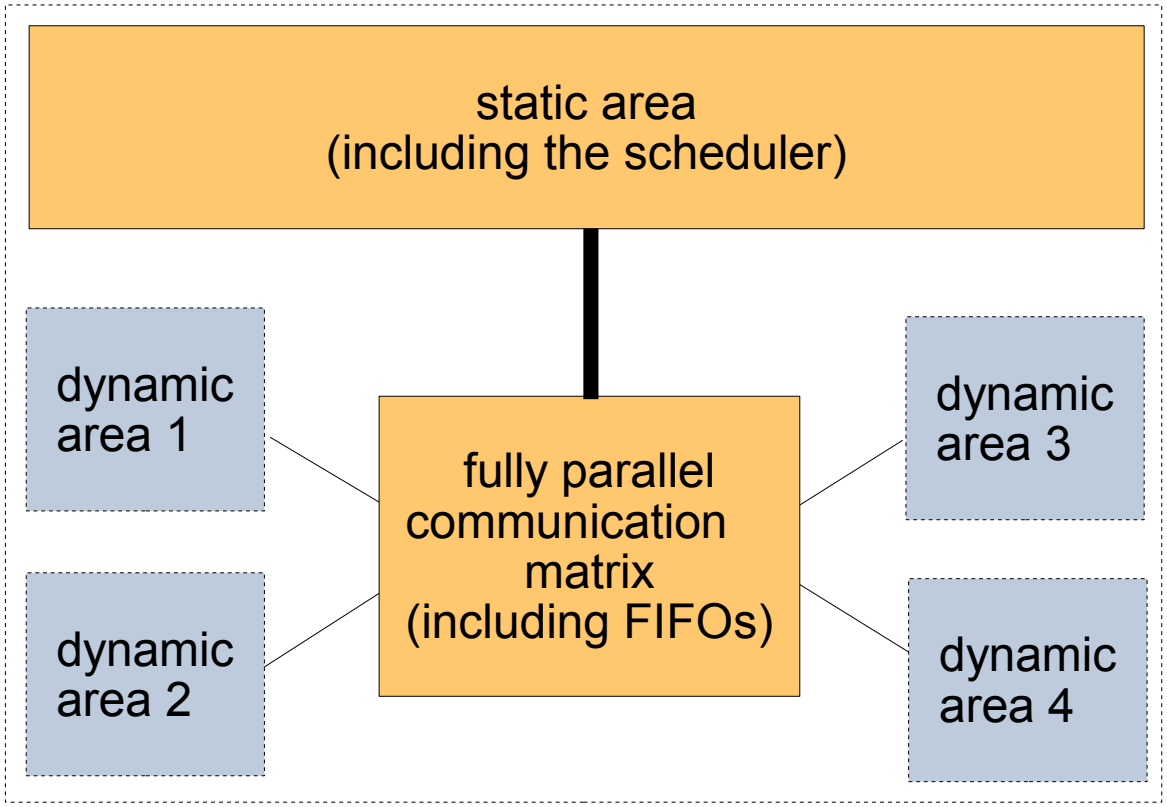


Developers just have to care about:

- Development of the dynamic and the static modules in Java

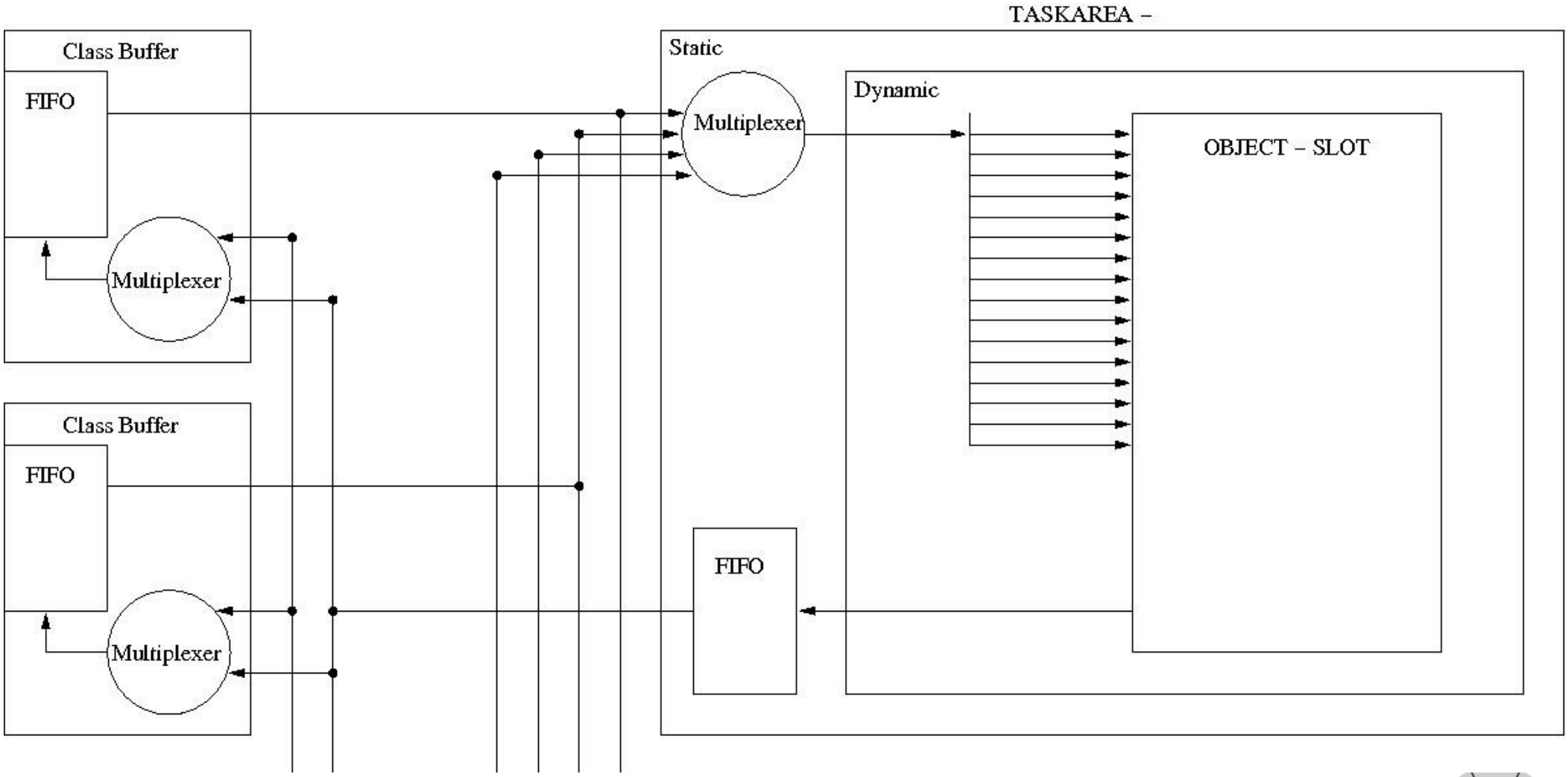
*The dynamically instantiation of a module is done with a simple **new***

A closer look to the Framework

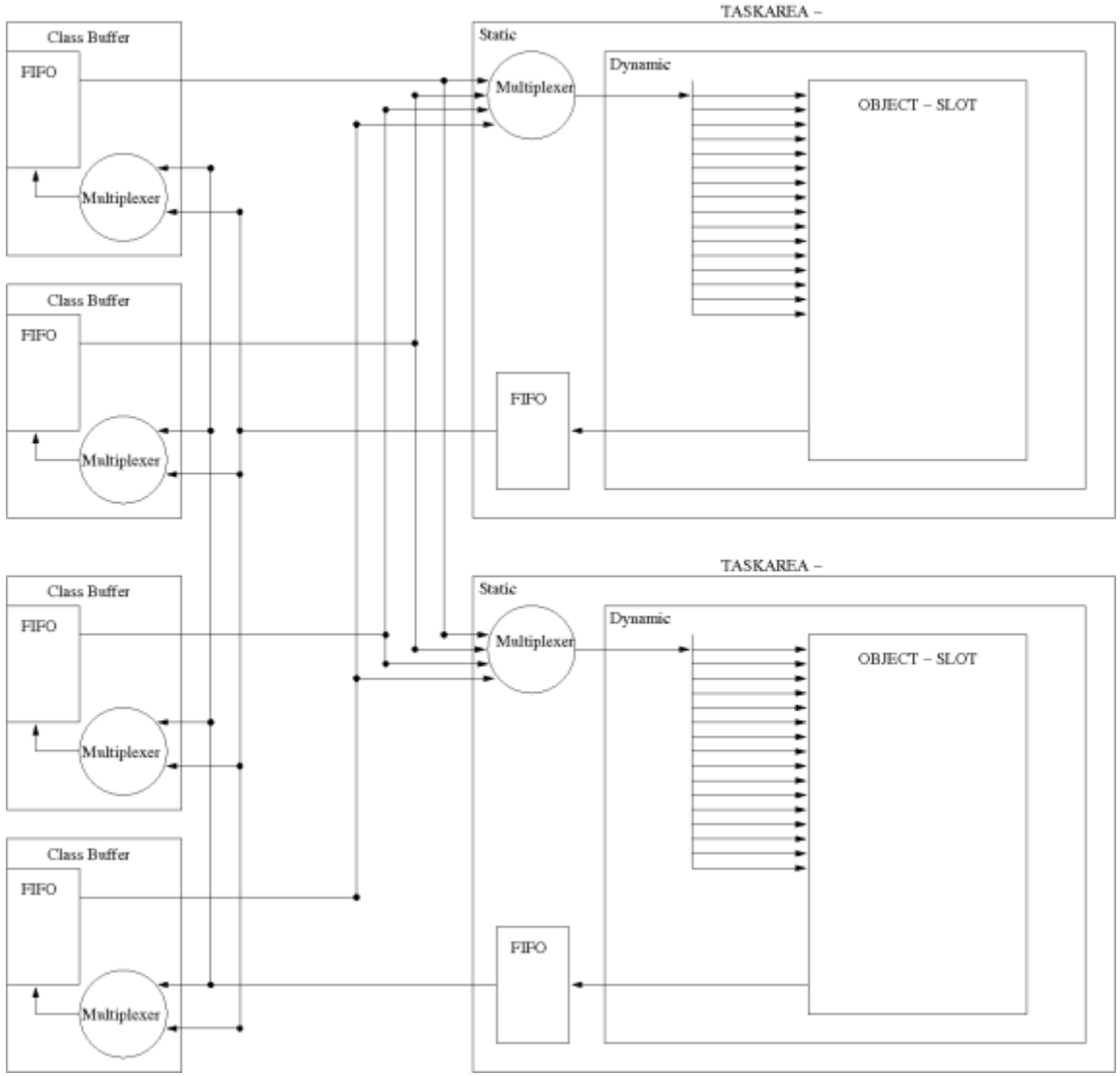


*Modules are loaded into the dynamic areas **on demand**. The communication matrix stores the data for every module in a FIFO. Since there is a farm of FIFOs the communication between several modules is fully parallel.*

The communication matrix



The communication matrix



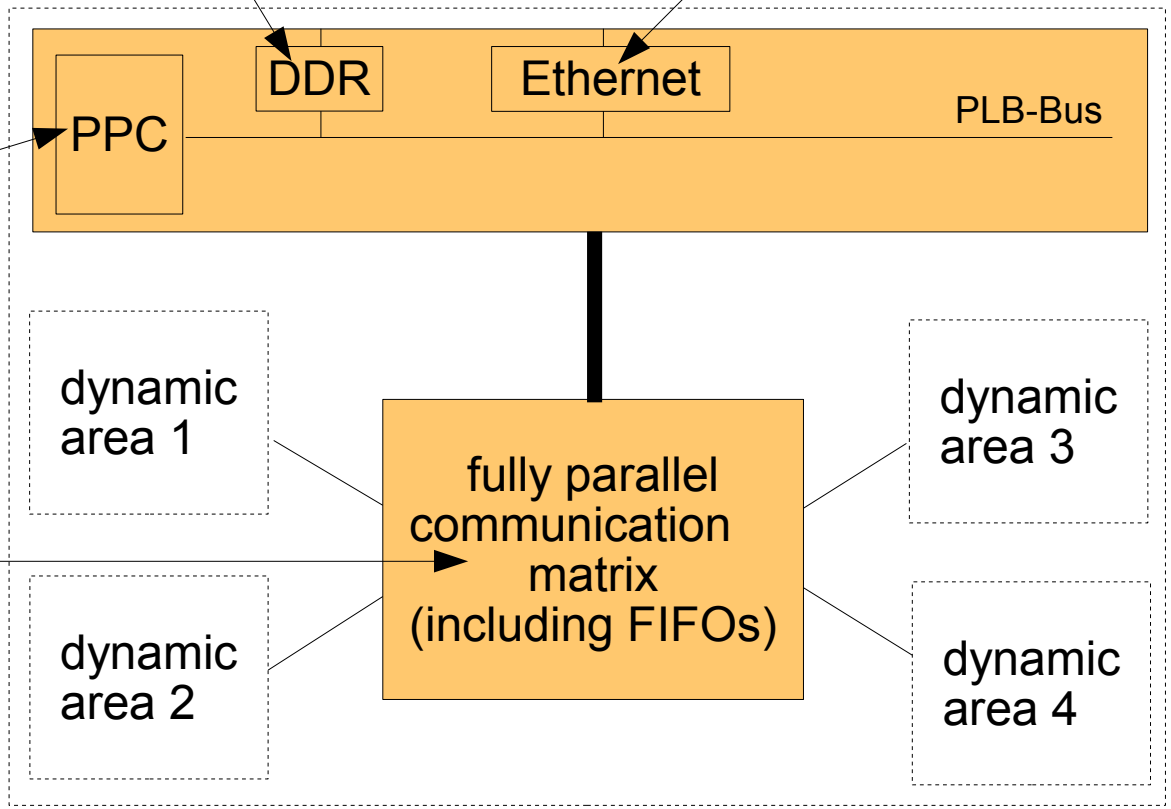
Java-Description of the static part

```
import system.*;
import system.architectures.*;
import system.basesystems.*;

public class MyXUPV2P {

  public static void  main (String[] args) throws SystemjavaException {

      Virtex4PPC mysystem = new Virtex4PPC(new Xilinx_ML503());

      mysystem.agent().addPlbEthernet();
      mysystem.agent().addPlbDDR();
      mysystem.agent().addDispatcher();

      new SystemJava().run(mysystem, args);

   }

}
```

Java-Description of the static part

mysystem.agent().addPlbDDR();

mysystem.agent().addPlbEhternet();



new Virtex4PPC(new Xilinx_ML503());

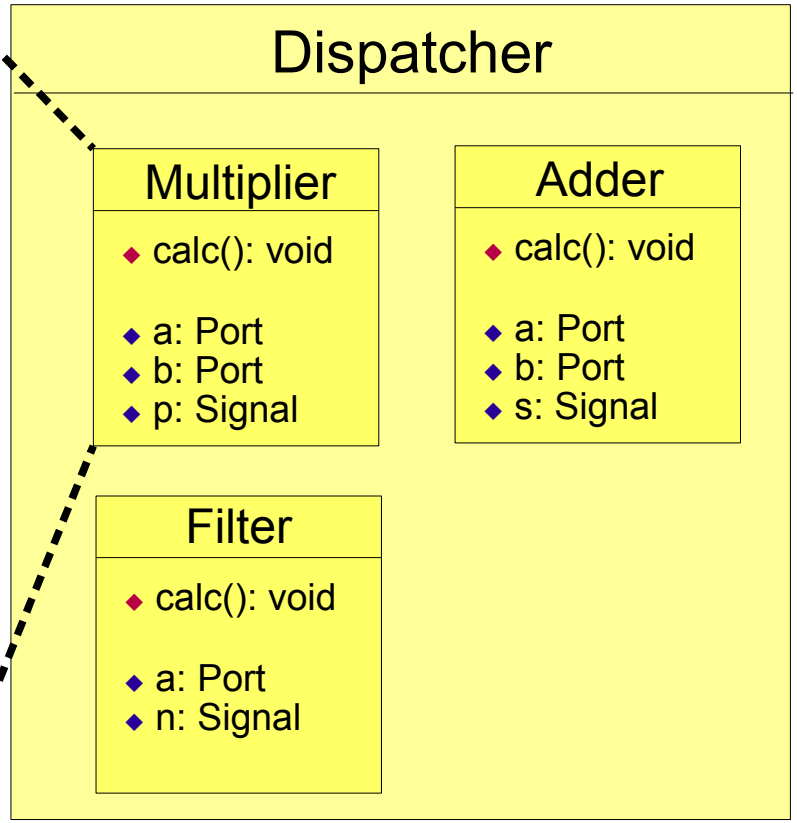
Generates the complete base system (including PPC, busses and clock controllers)

mysystem.agent().addDispatcher();

Generates the scheduler and the communcation matrix (including the FIFOs)

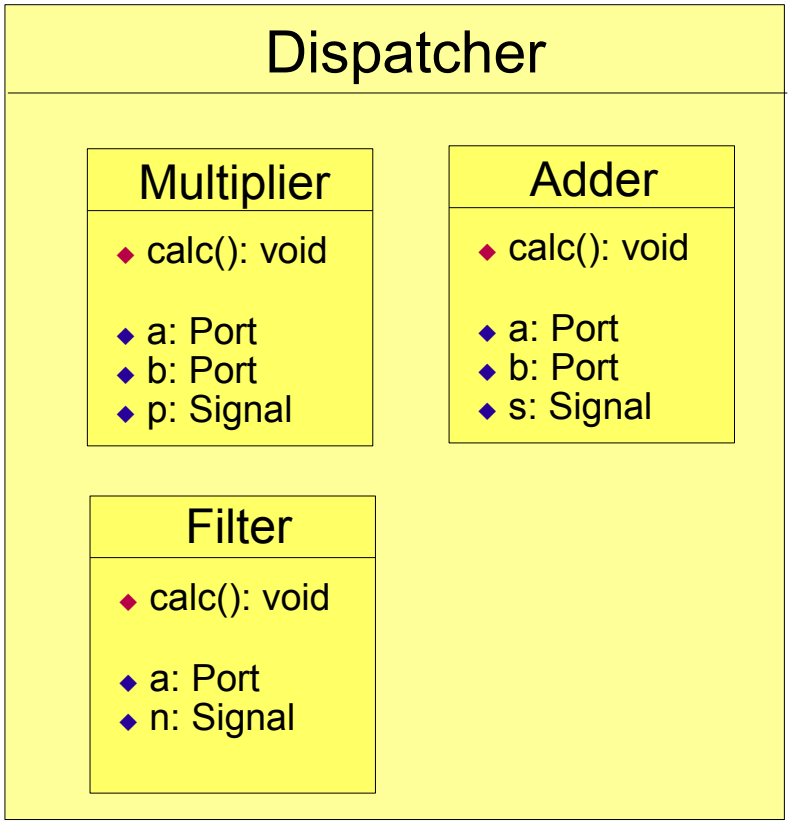
Java-Description of the dynamic part

```
class Multipier extends ParObj {
   Slot a,b;
   Signal p;

   int result;

   calc() {
      result = a.get() + b.get ();
      p.emit(result);
   }
}
```

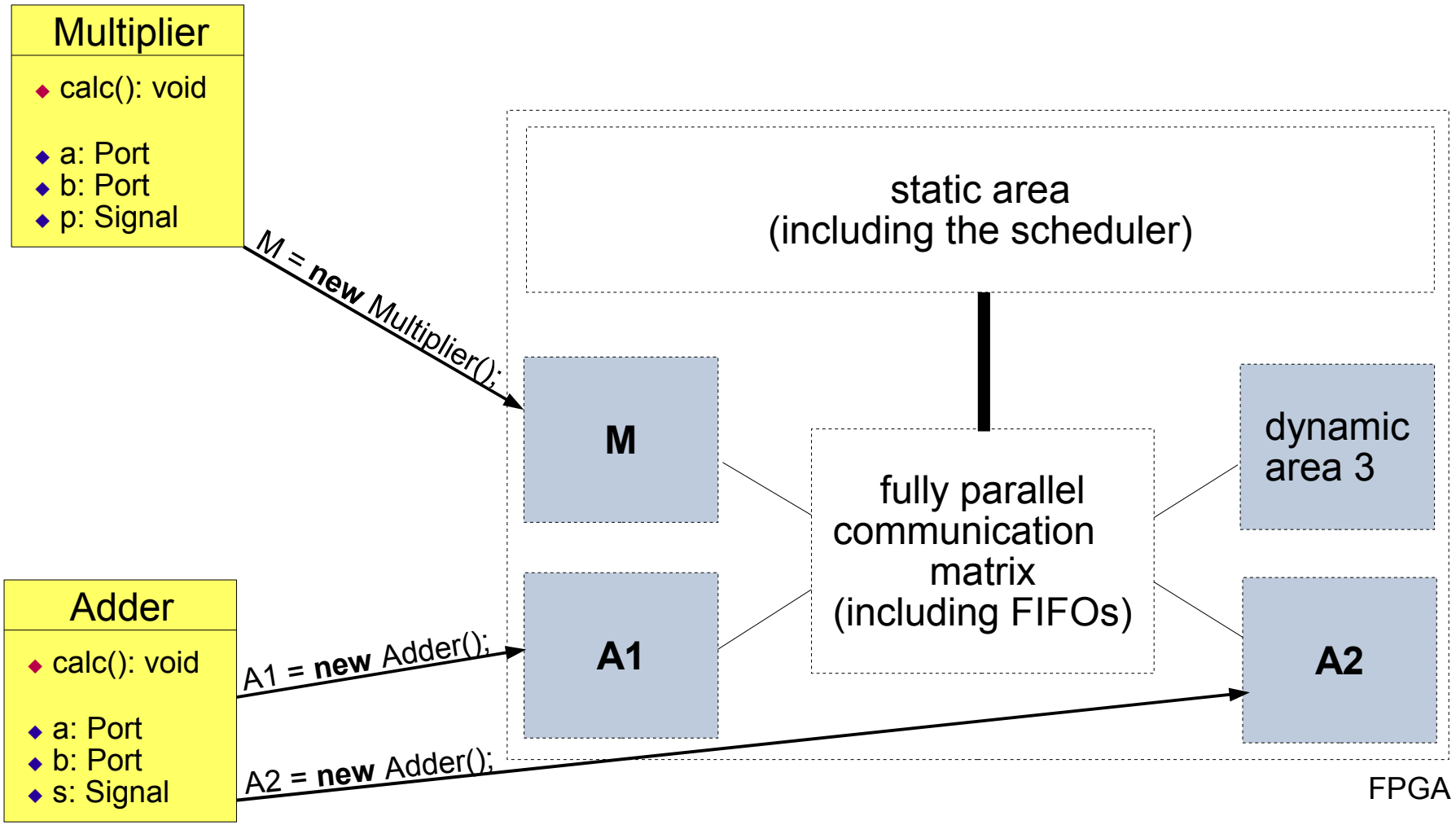


Java-Description of the dynamic part

```
class Dispatcher extends ParObj {

  ...

  Dispatcher () {
     M = new Multiplier();
     A1 = new Adder();
     A2 = new Adder();

     A1.s.connect(M.a);
     A2.s.connect(M.b);
   }

   calc() {
     ...
       if (stdin==”1”) F1 = new Filter();
     ...
   }
}
```



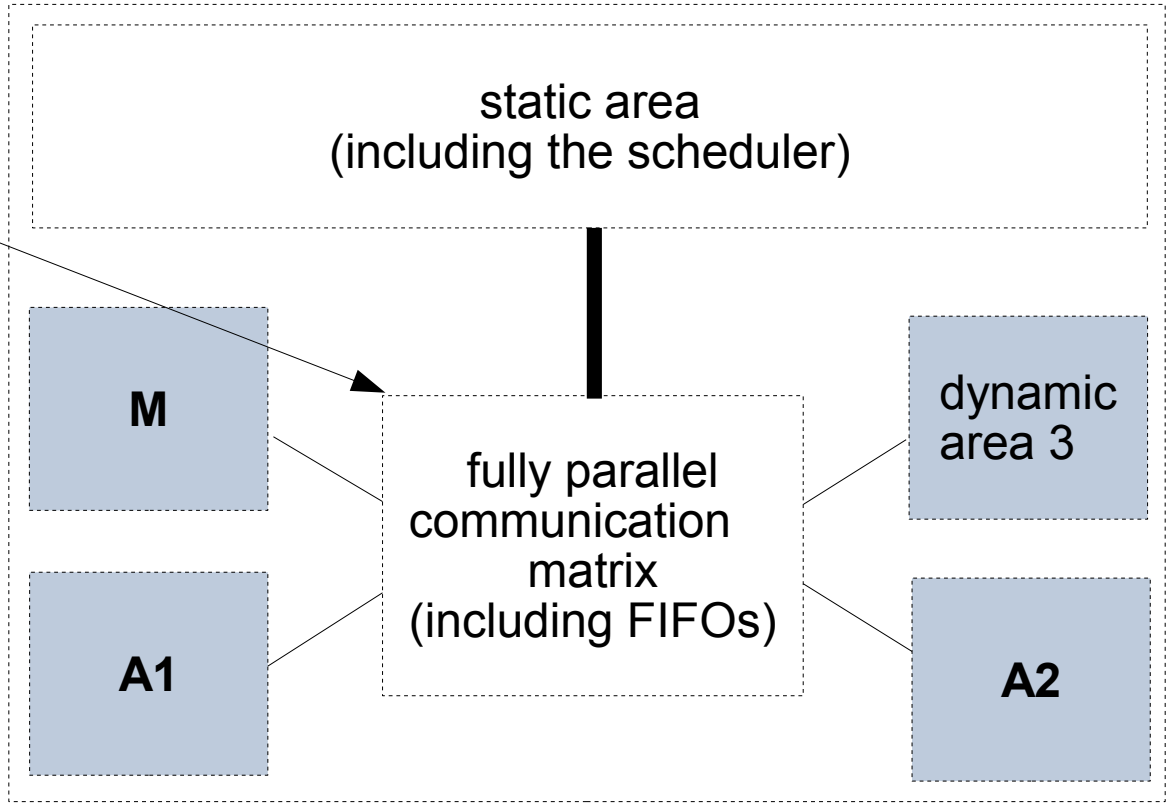
Java-Description of the dynamic part



Java-Description of the dynamic part

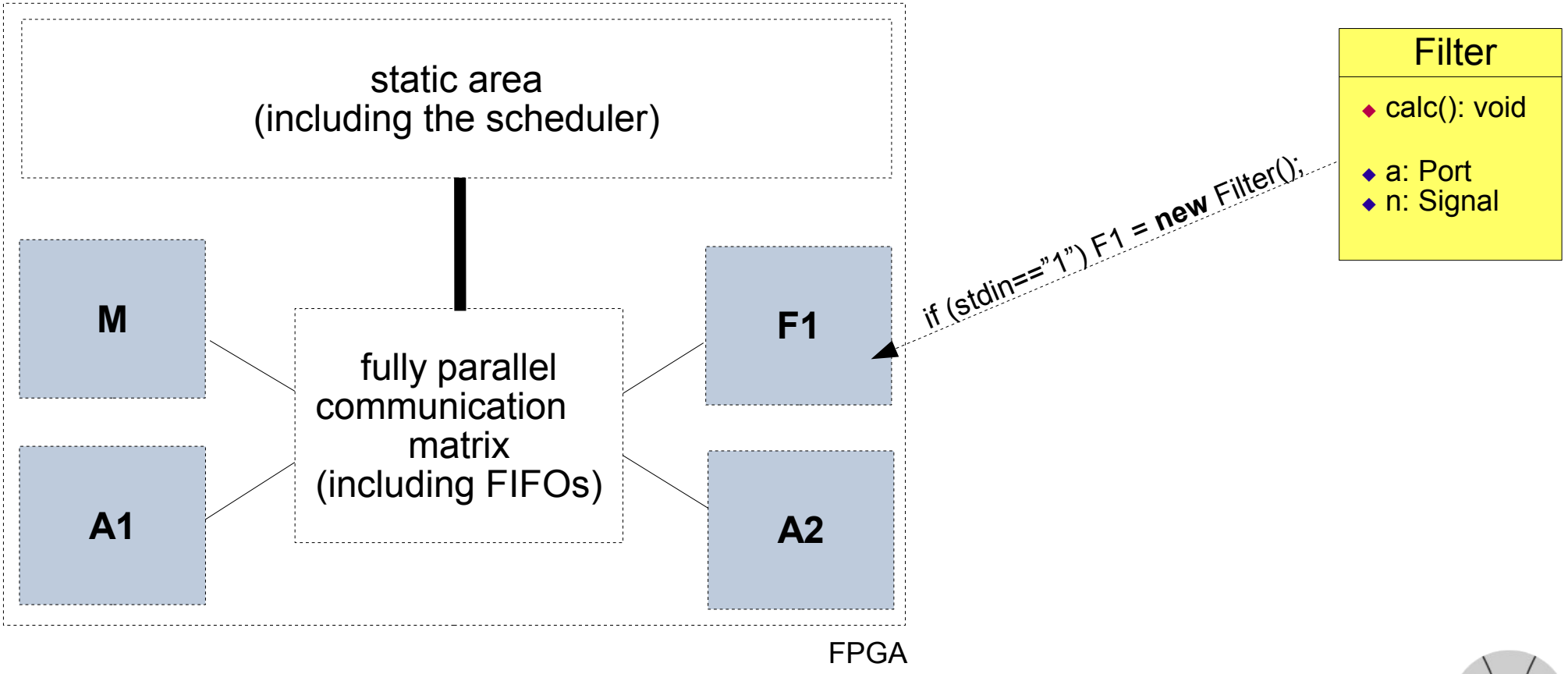
A1.s.connect(M.a);

*Tells **A1** via the communication matrix, that it shall send its calculations to **M**.*



Java-Description of the dynamic part

*Objects can be added and removed at runtime with a simple **new** or **finish()**.*



Java-Description of the dynamic part

If there are more instances than dynamic areas the system can schedule.
The figure illustrates our actual test setup.