KIP Heidelberg

Norbert Abel

Specification of POL and of the Reconfiguration-Framework

Communication Matrix:		Nick Meier
POL to VHDL	:	Frederik Grüll
POL to Java	:	Andreas Beyer

Using the conventional tools...

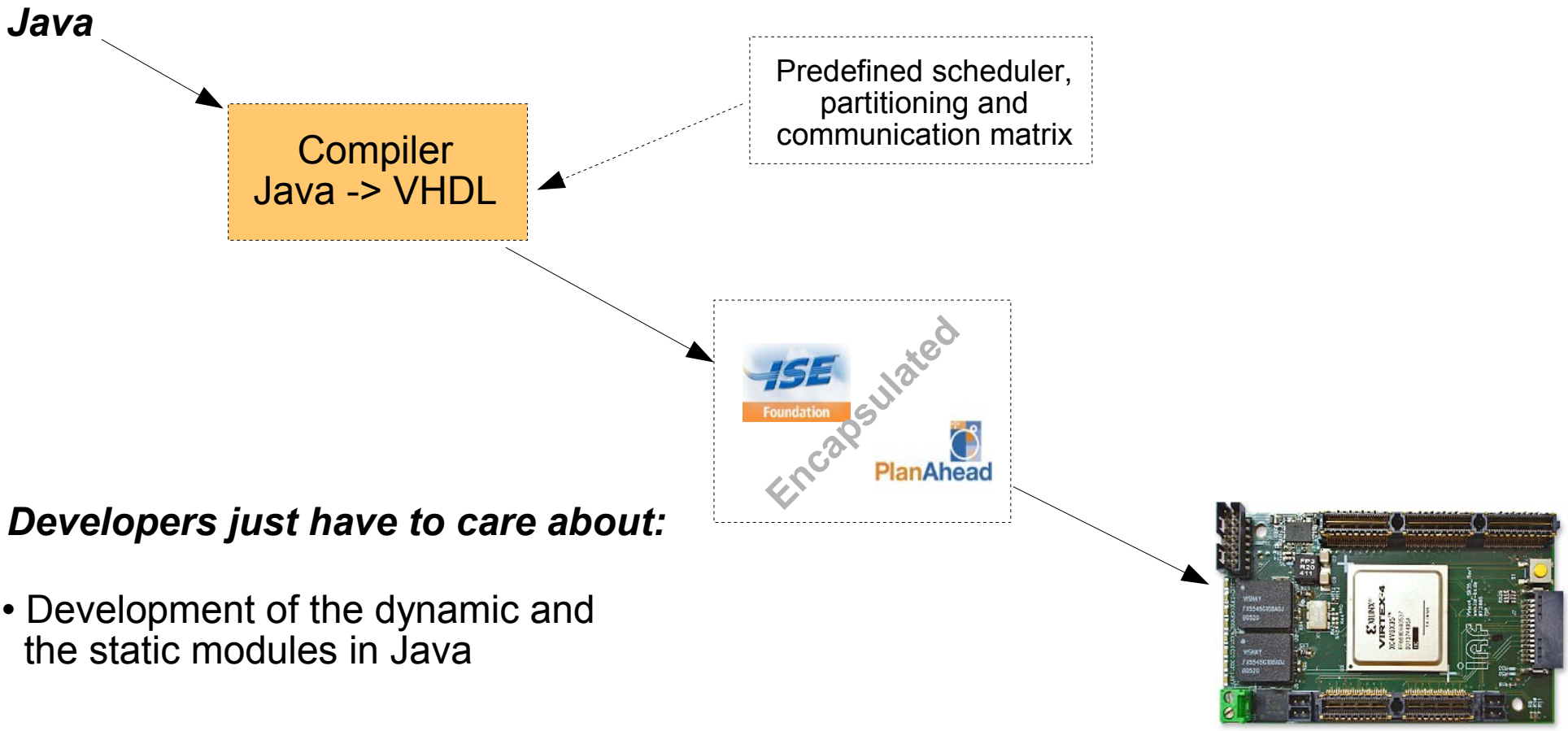
VHDL

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...

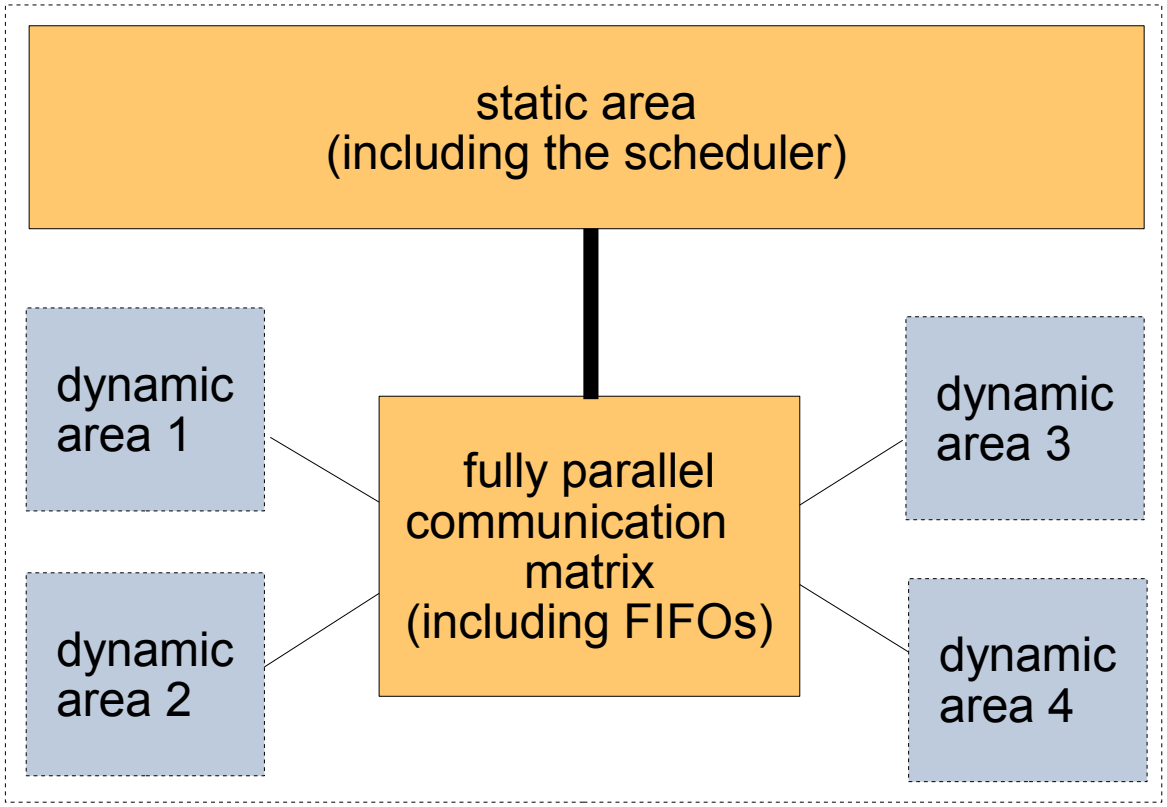


Developers just have to care about:

- 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

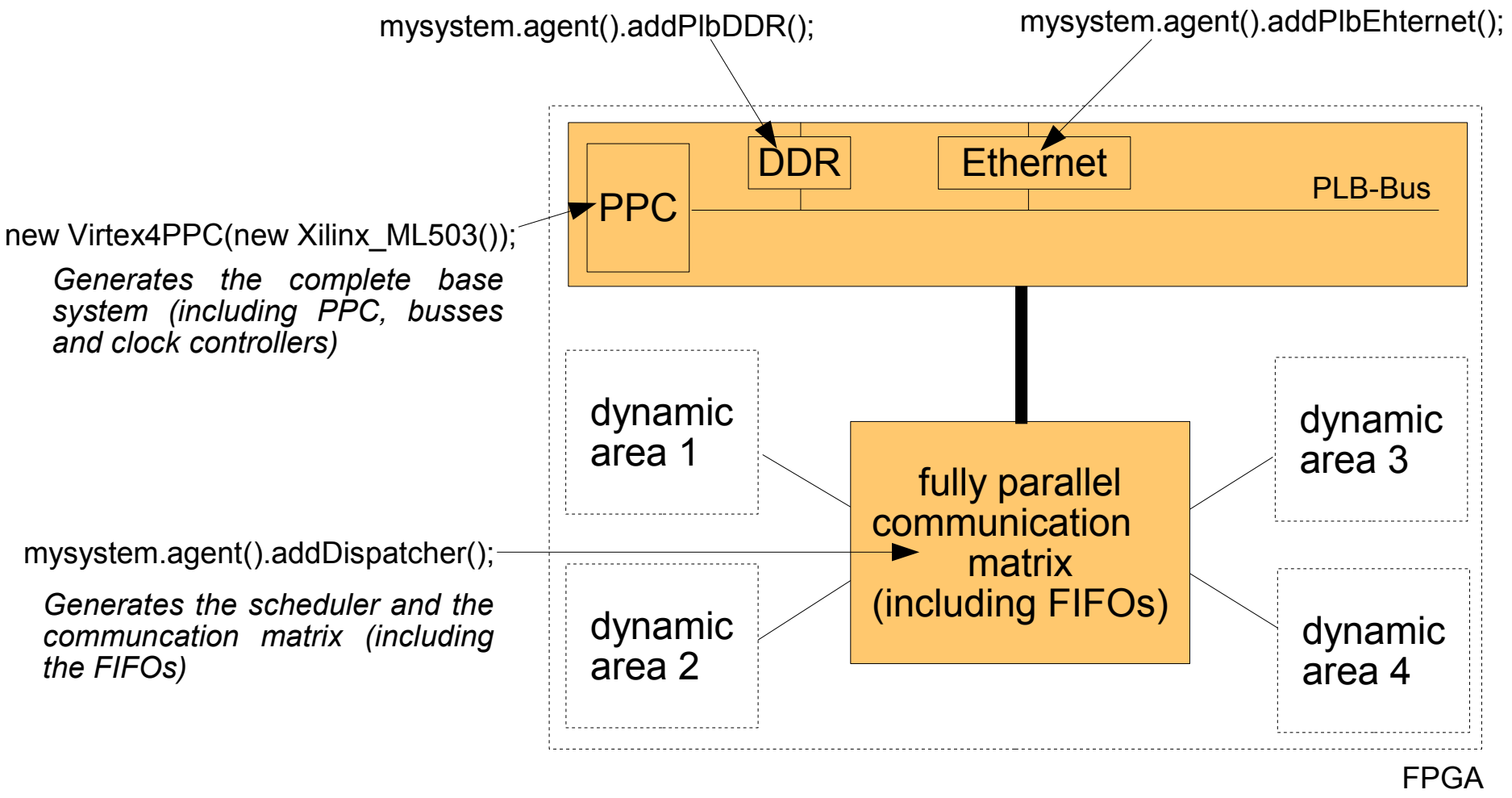


*Modules are loaded into the dynamic areas **on demand**. The communication matrix stores the data for every module in a FIFO. Since there is a farm of FIFOs the communication between several modules is fully parallel.*

Java-Description of the static part

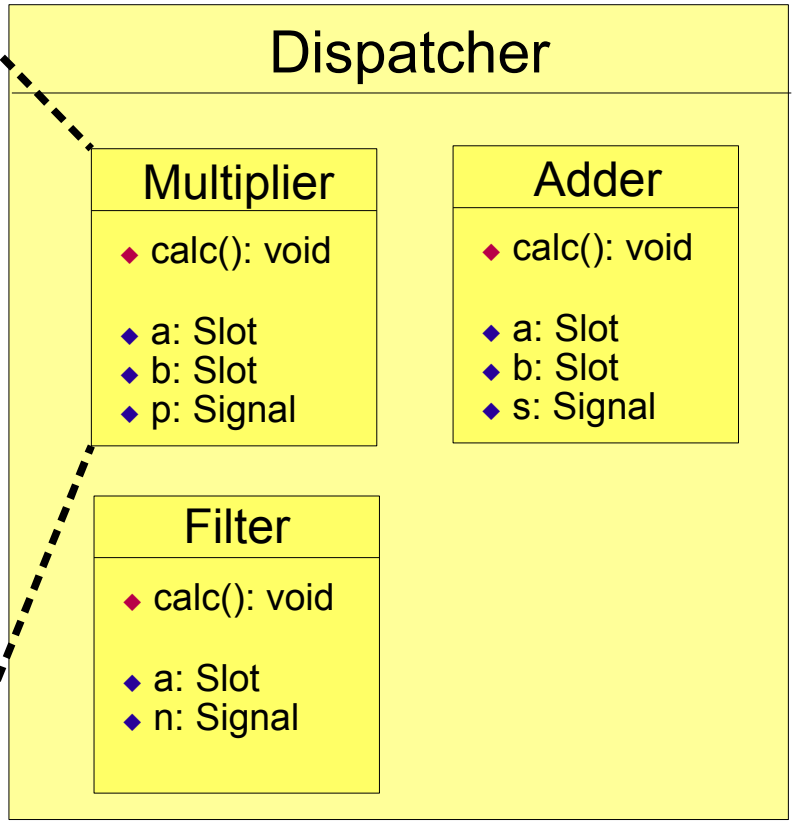
```
import system.*;
import system.architectures.*;
import system.basesystems.*;

public class MySystem {

  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



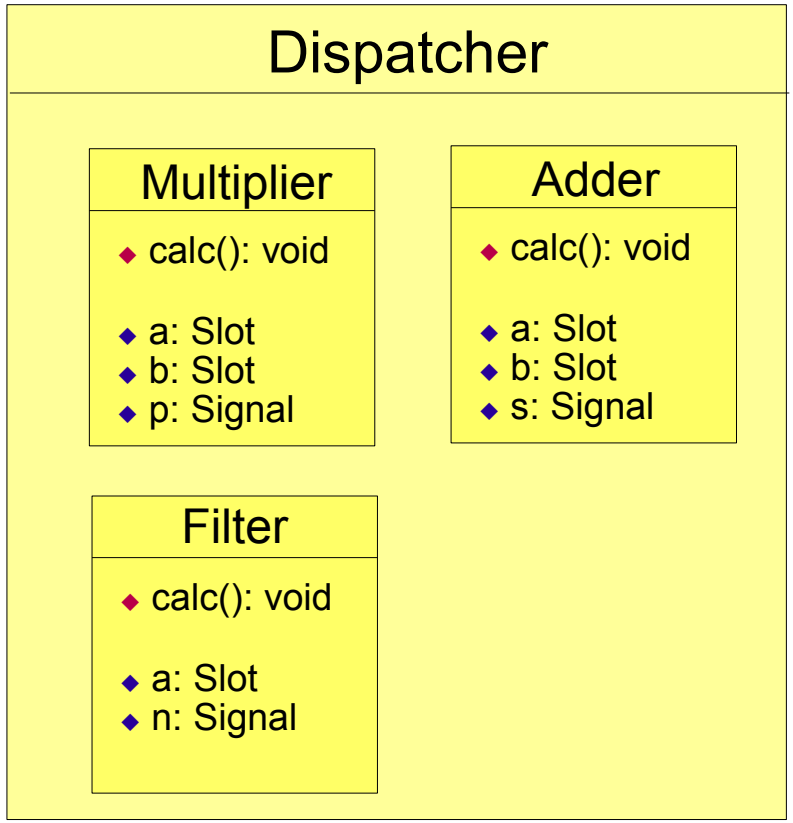
Java-Description of the dynamic part

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

   int result;

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

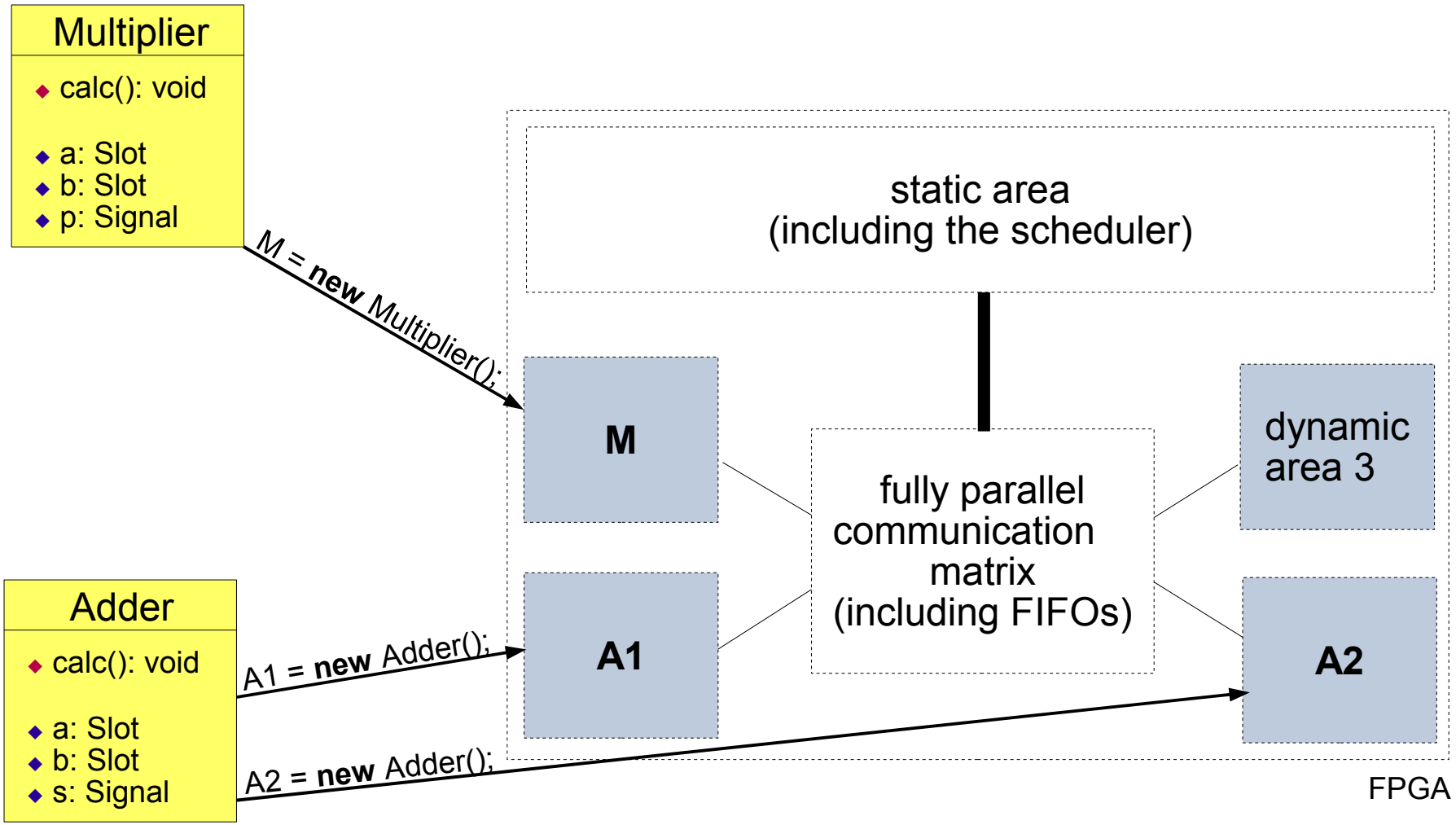


Java-Description of the dynamic part

```
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();
     ...
   }
}
```



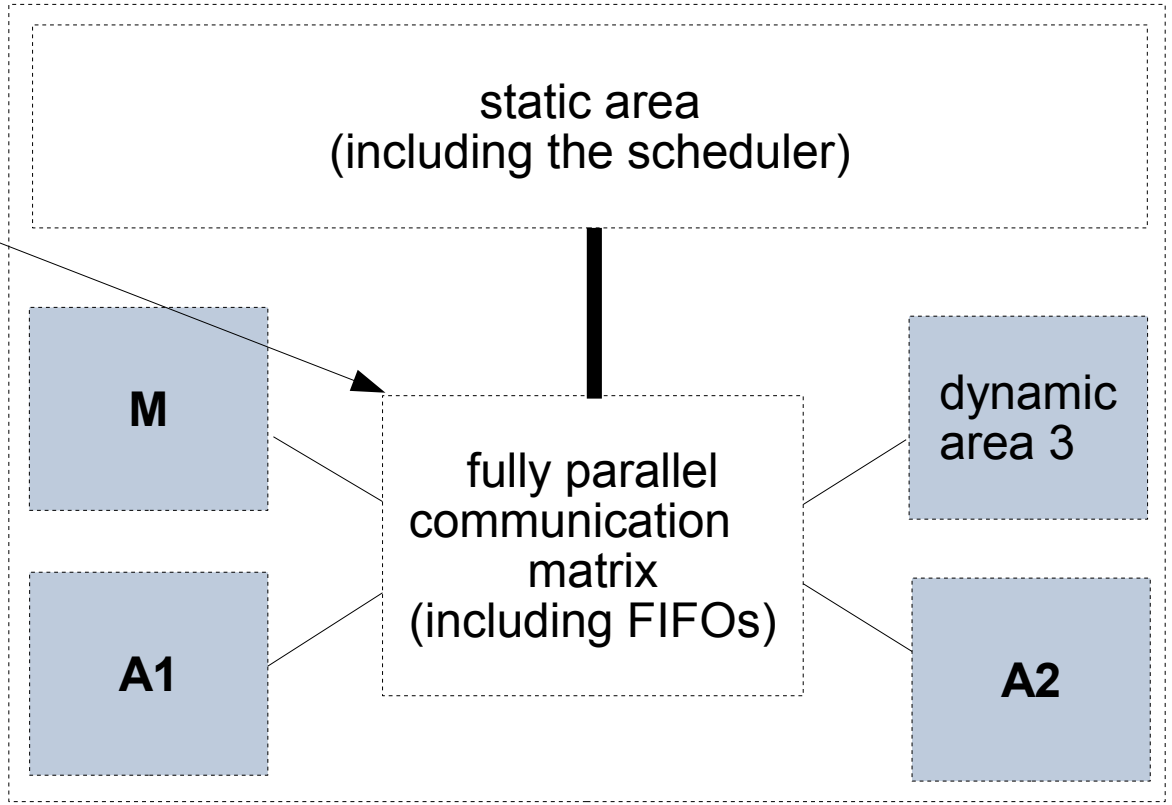
Java-Description of the dynamic part



Java-Description of the dynamic part

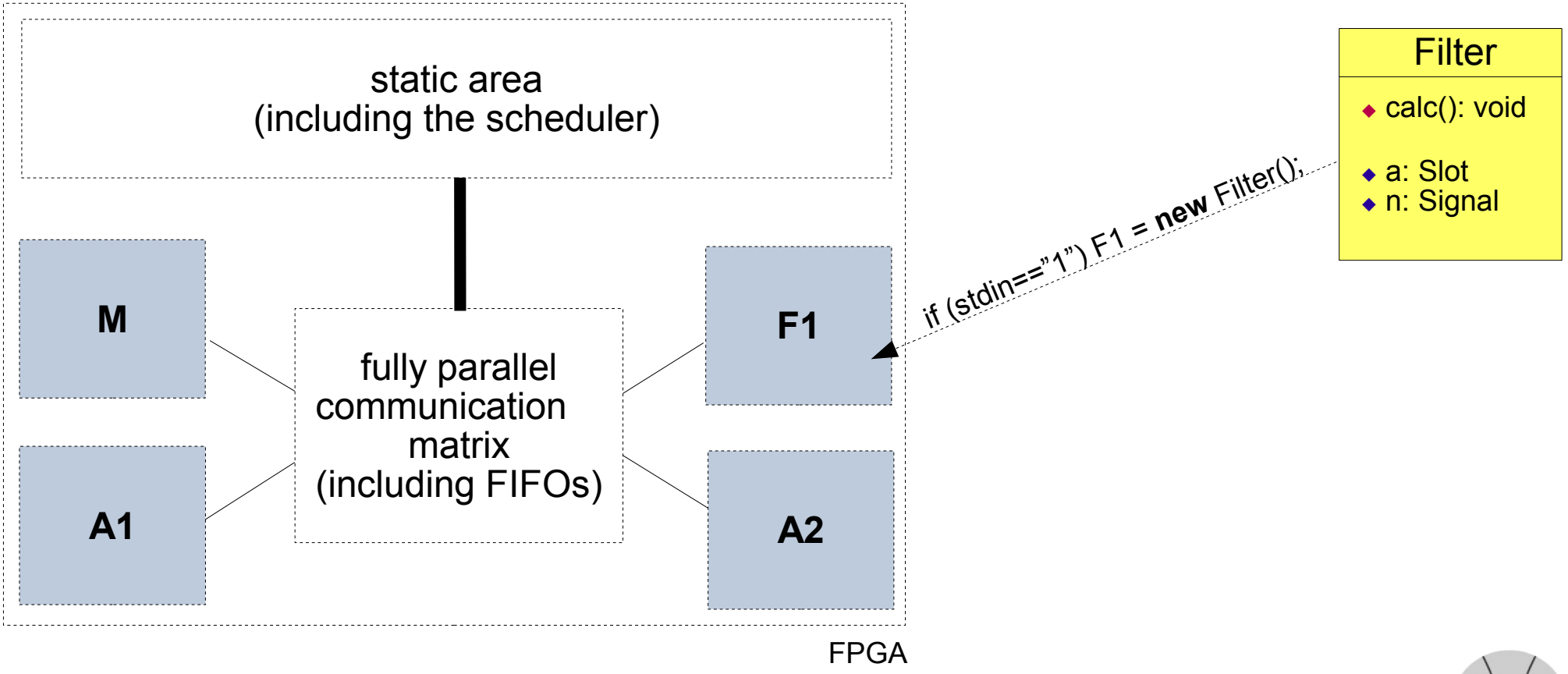
A1.s.connect(M.a);

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



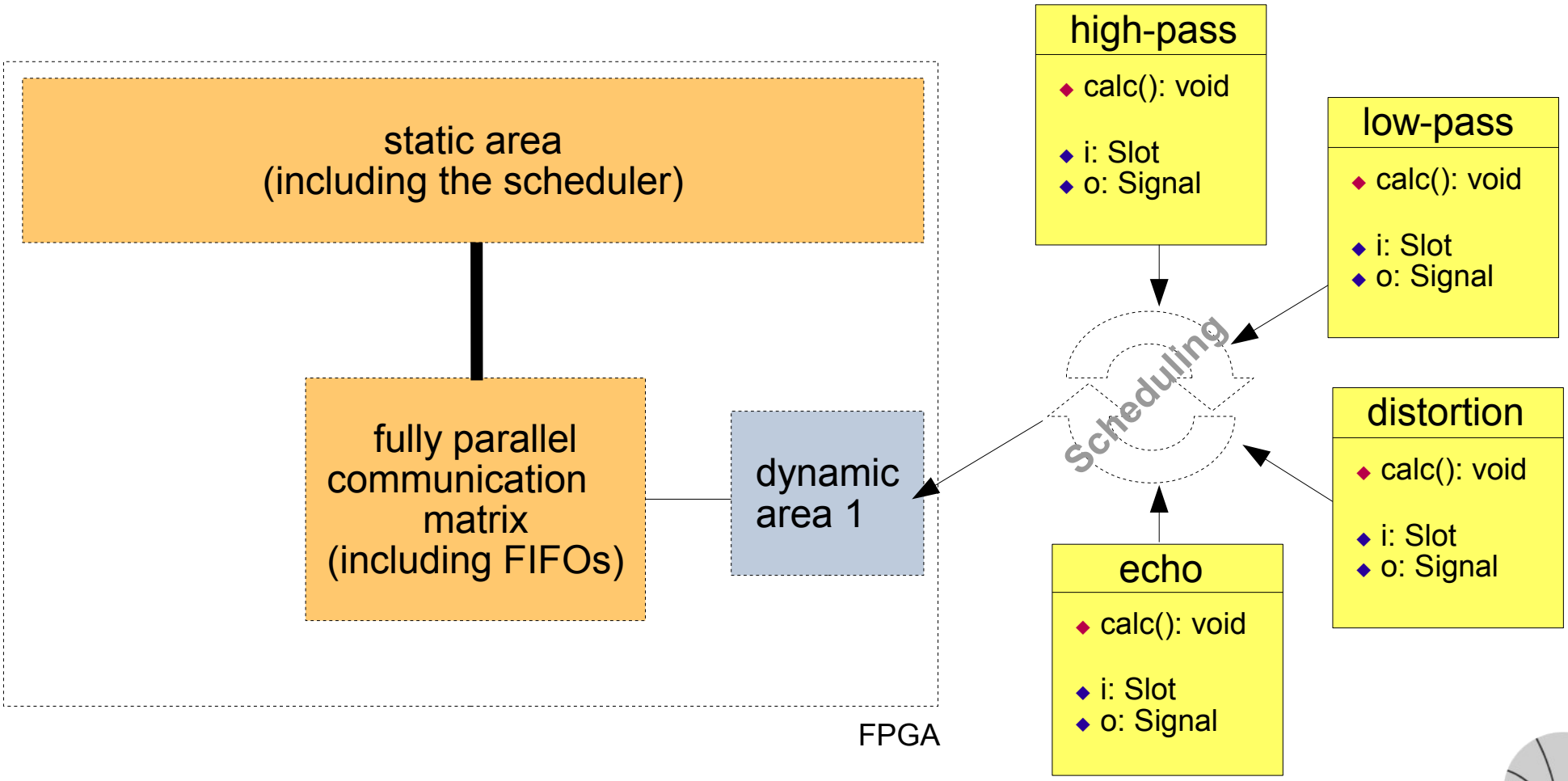
Java-Description of the dynamic part

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

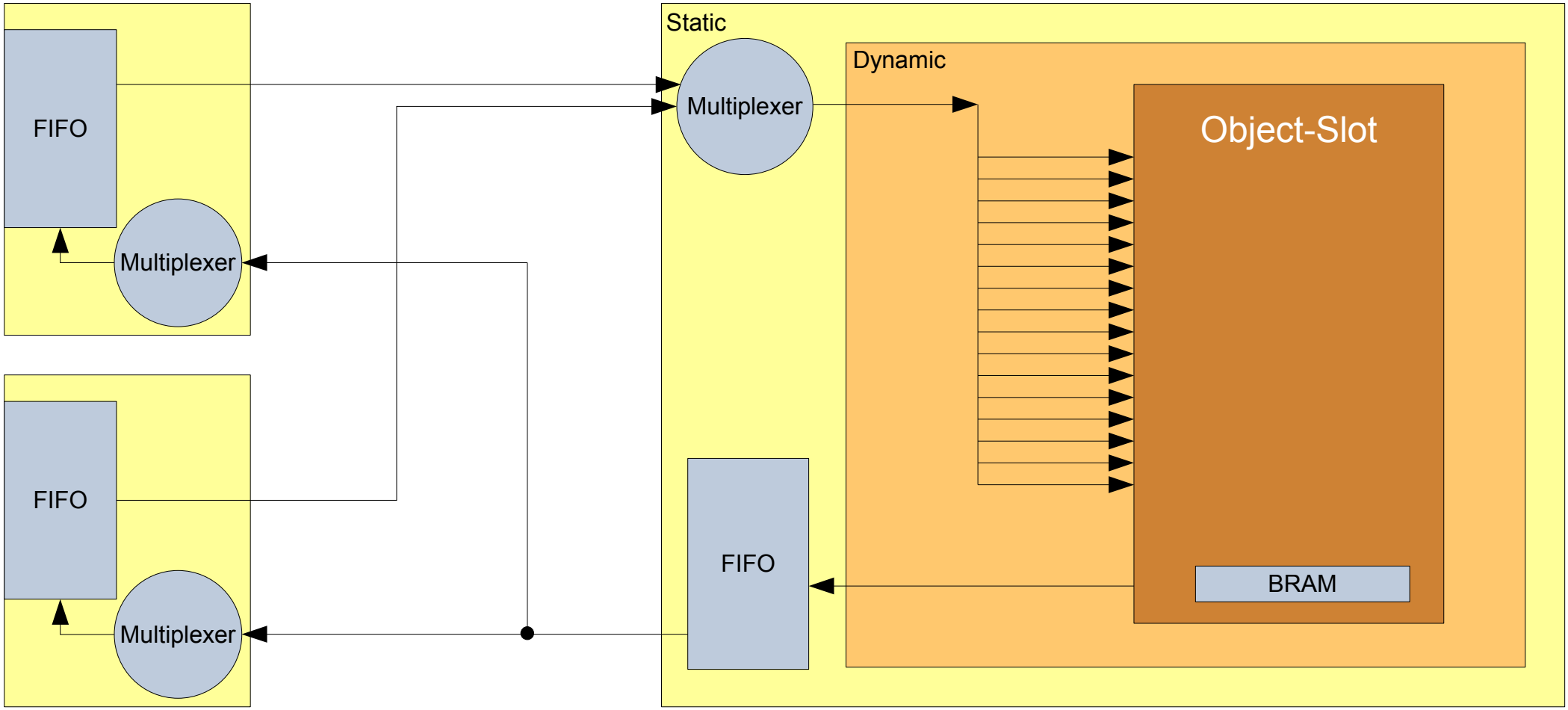


Java-Description of the dynamic part

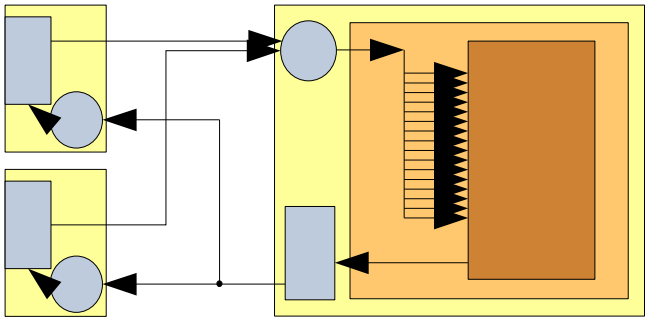
If there are more instances than dynamic areas the system can schedule.
The figure illustrates our actual test setup.

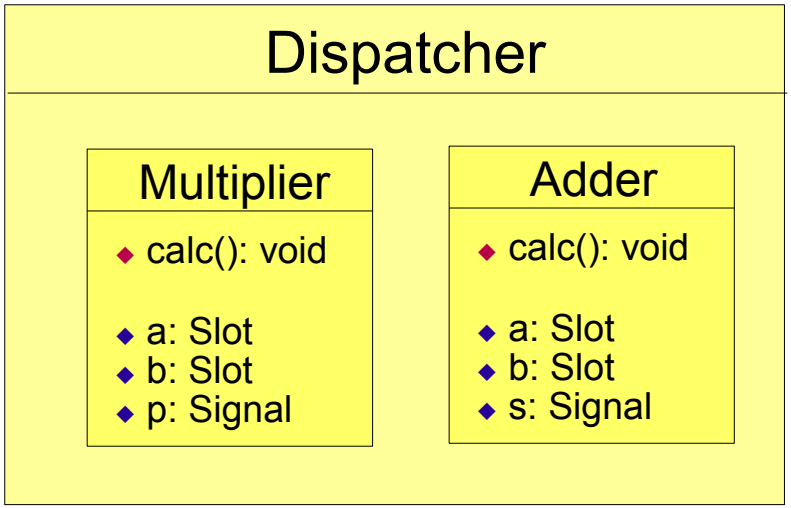


The communication matrix

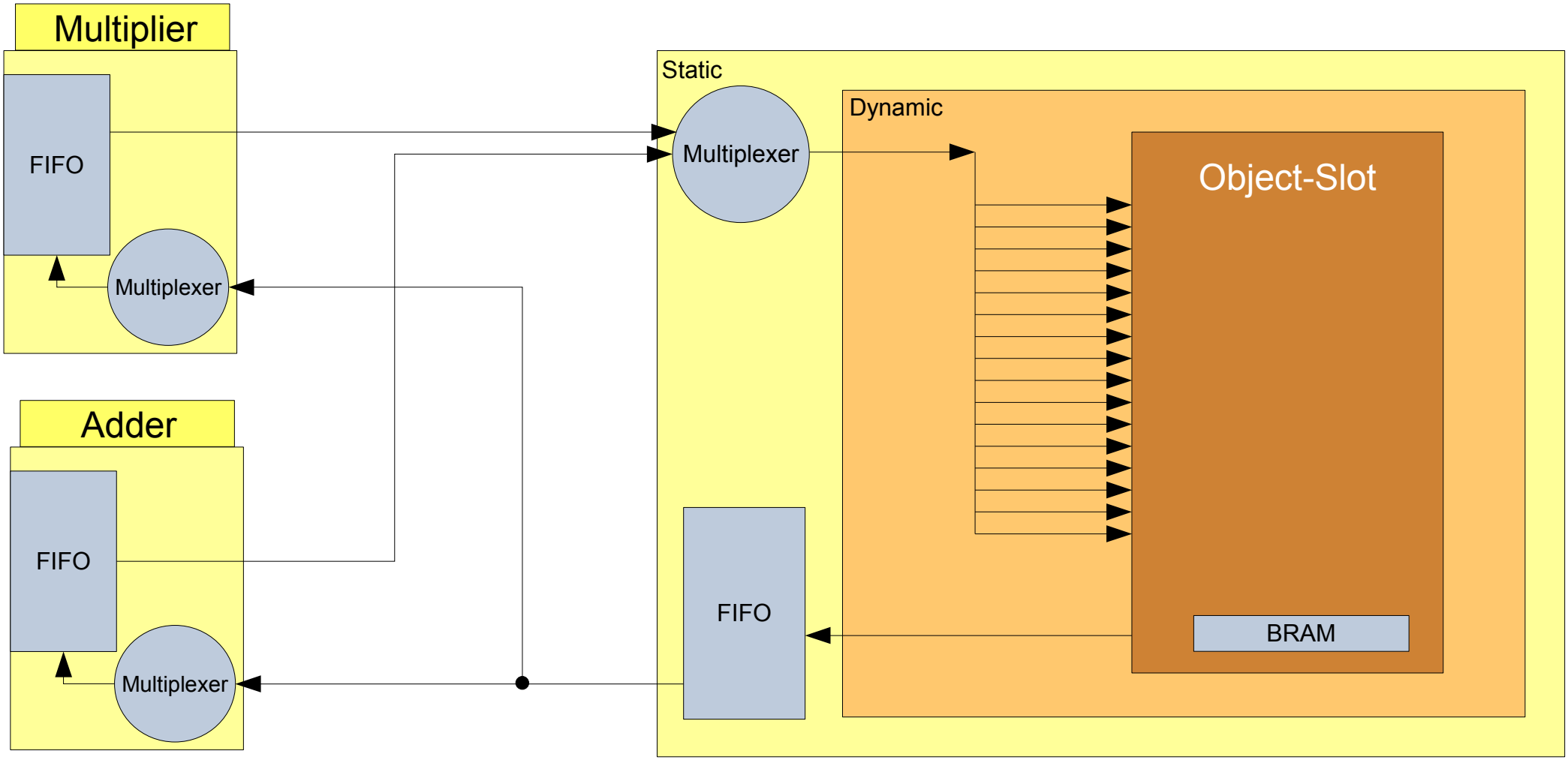


The communication matrix

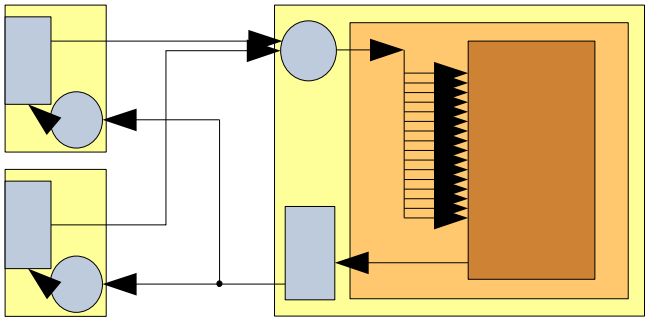


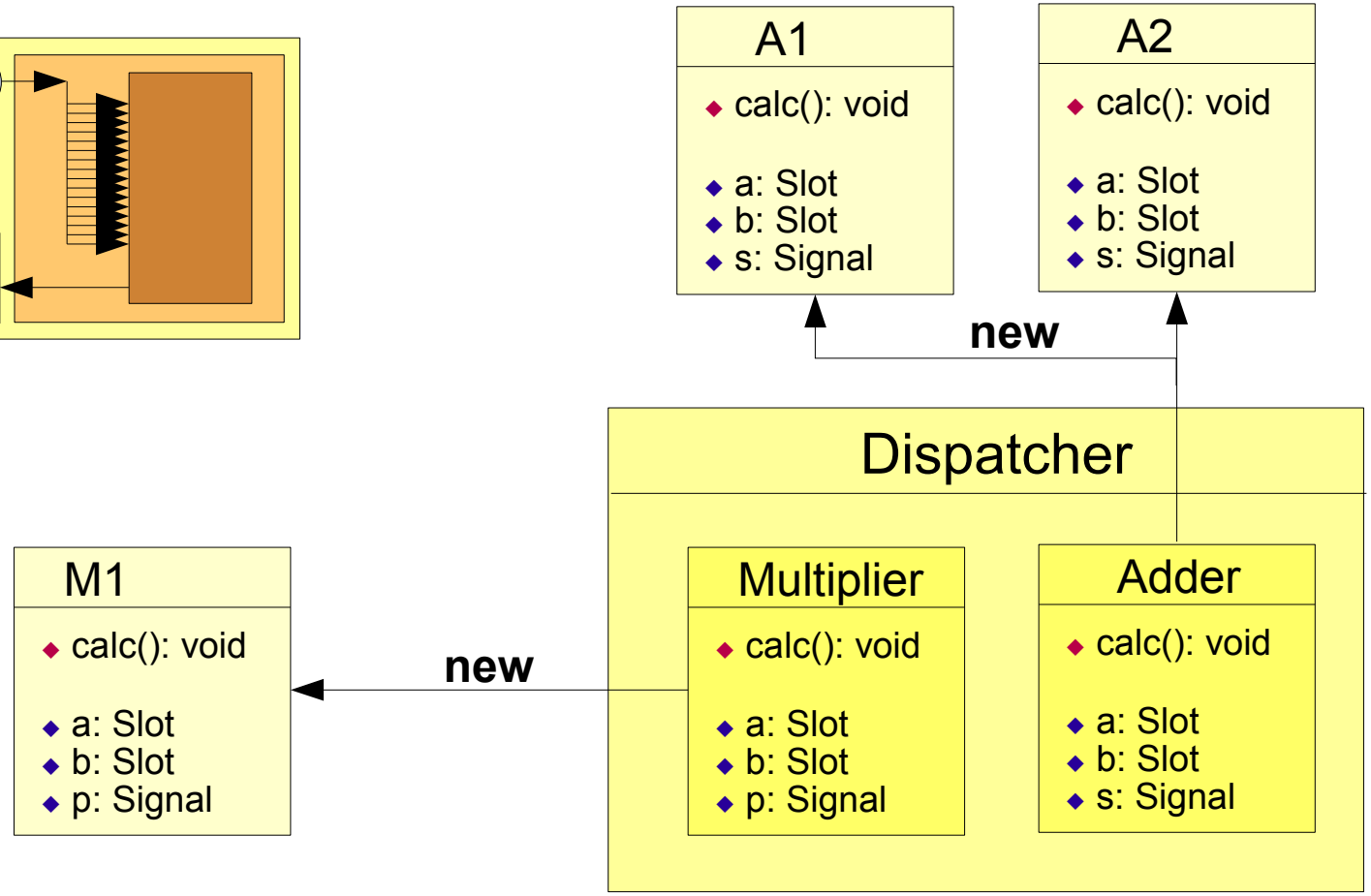


The communication matrix

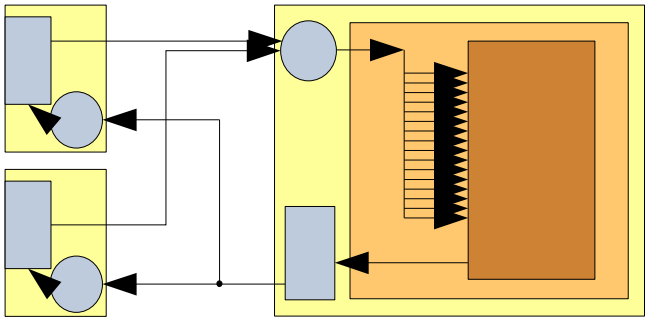


The communication matrix

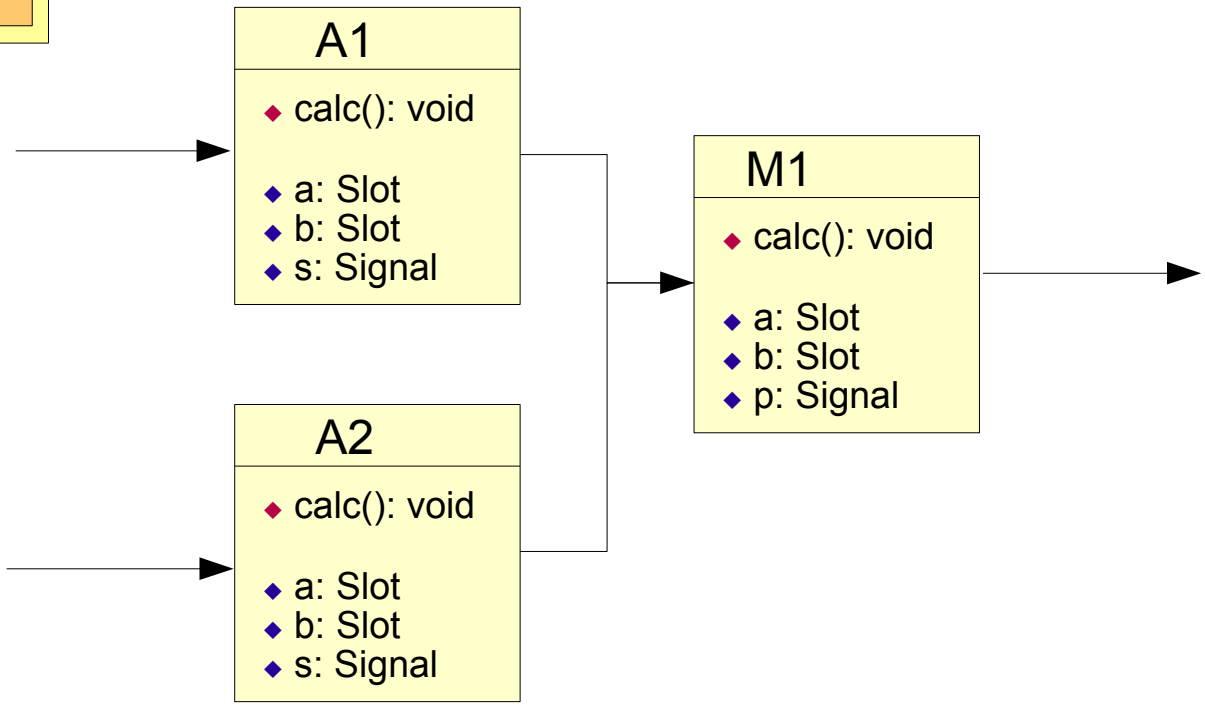




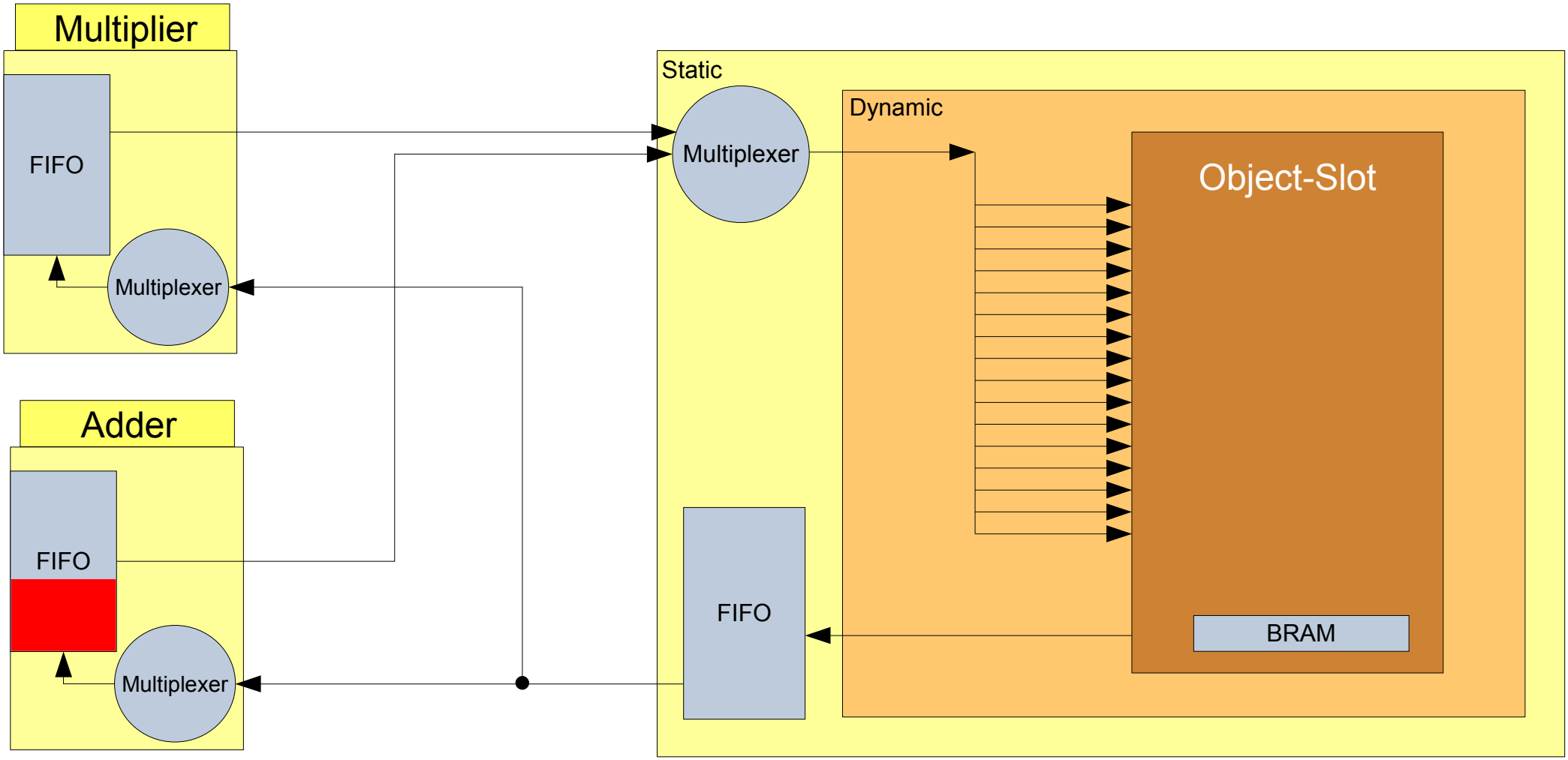
The communication matrix



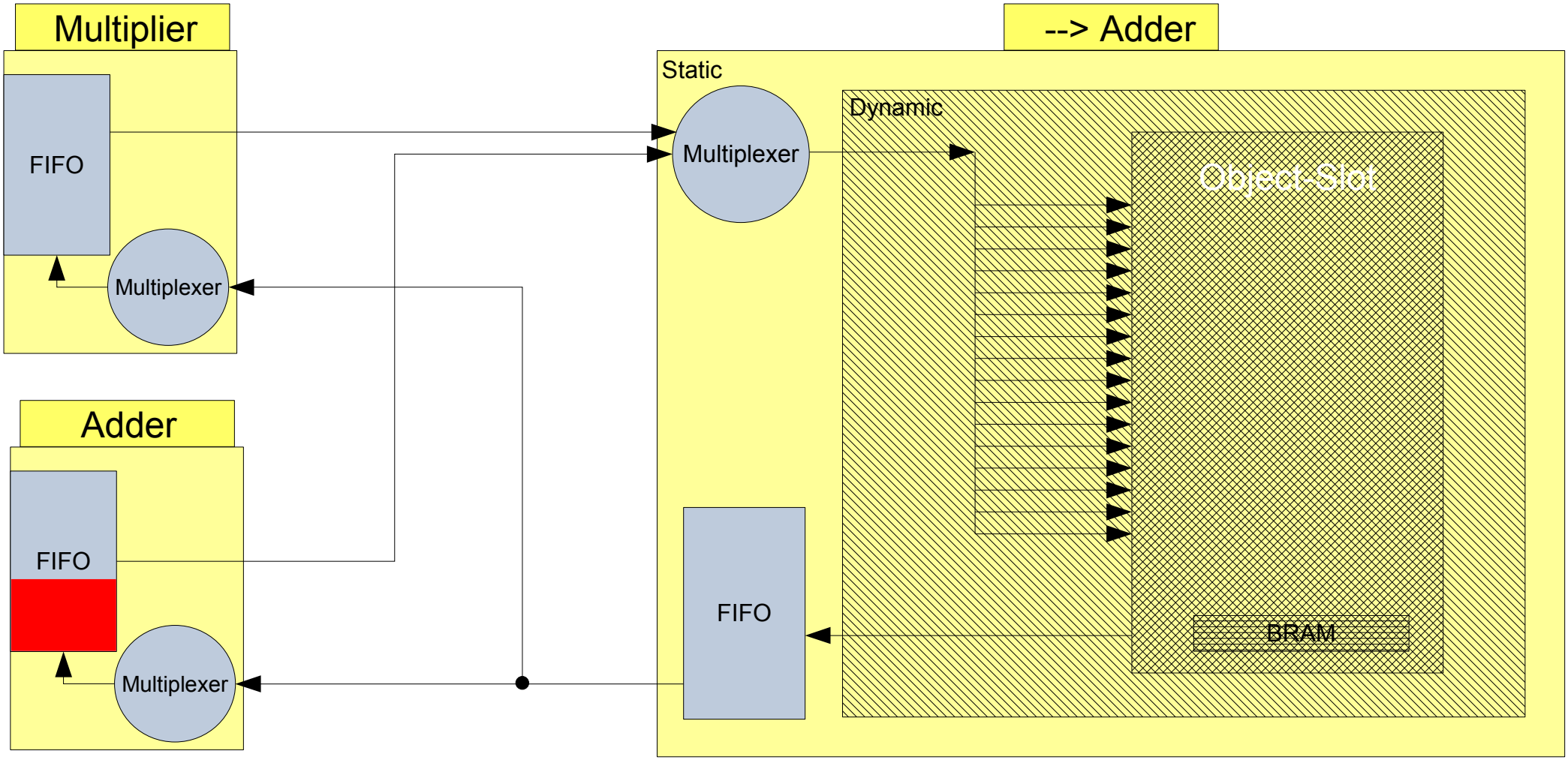
```
A1.s.connect(M.a);
A2.s.connect(M.b);
```



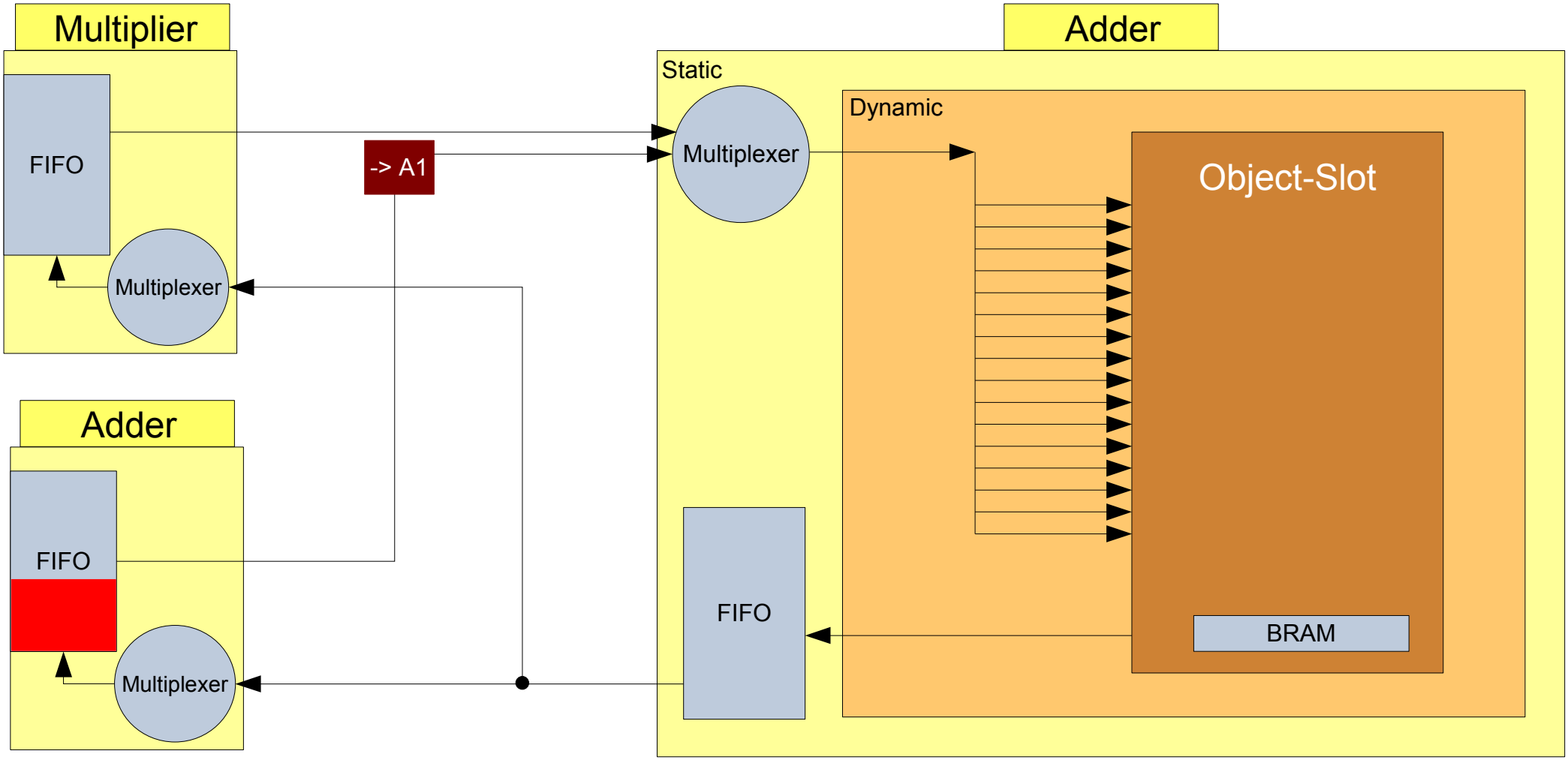
The communication matrix



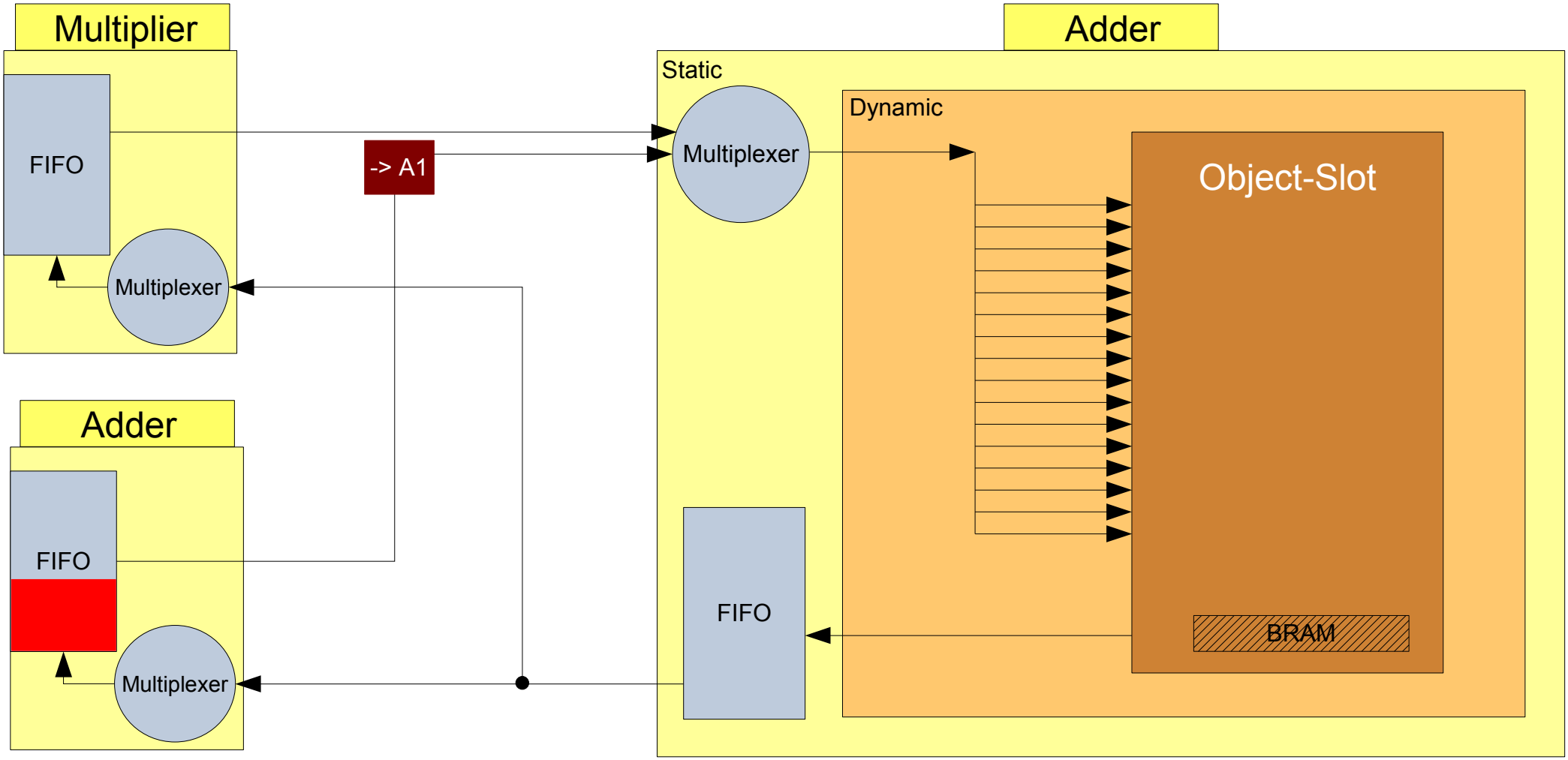
The communication matrix



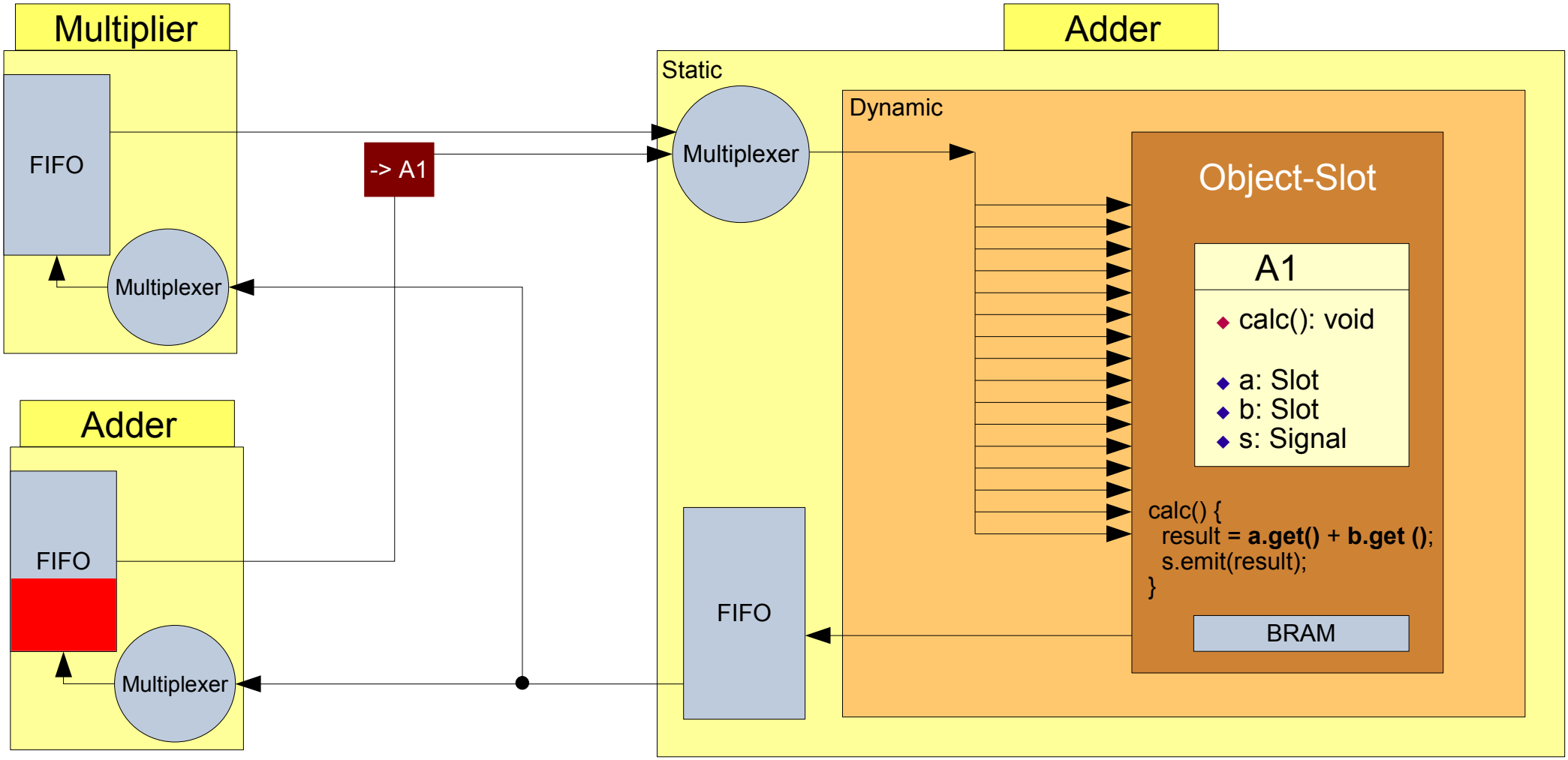
The communication matrix



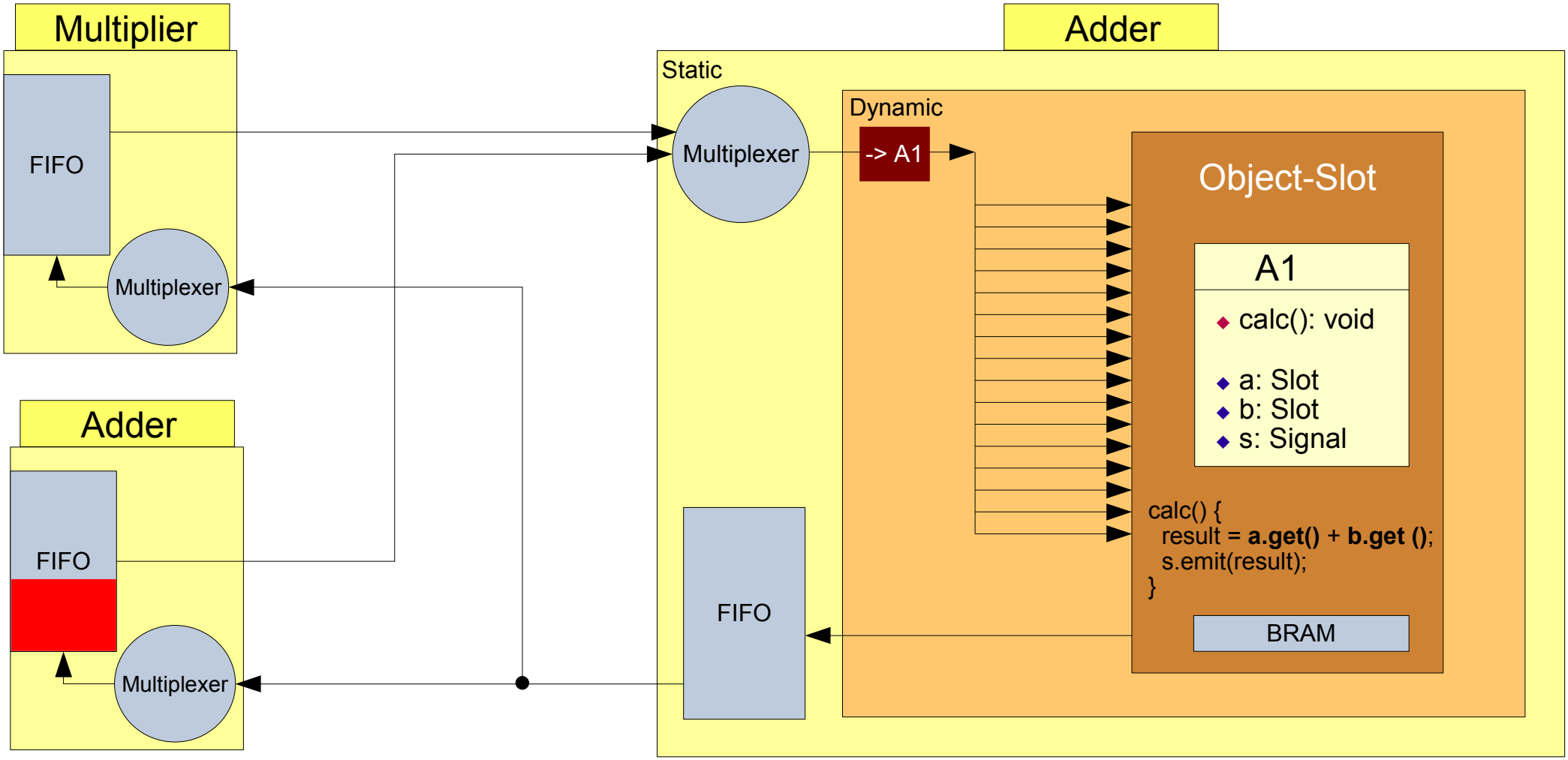
The communication matrix



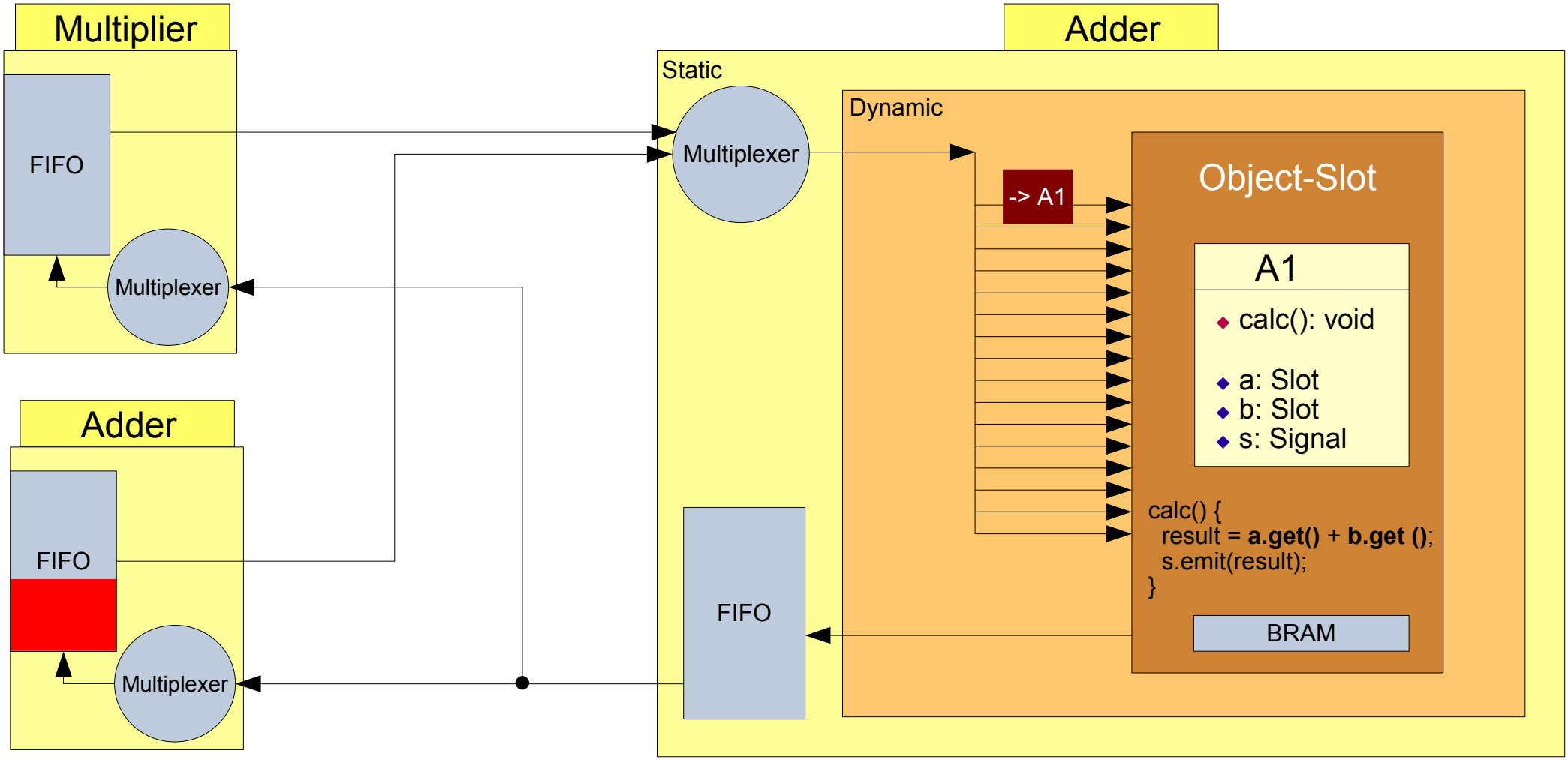
The communication matrix



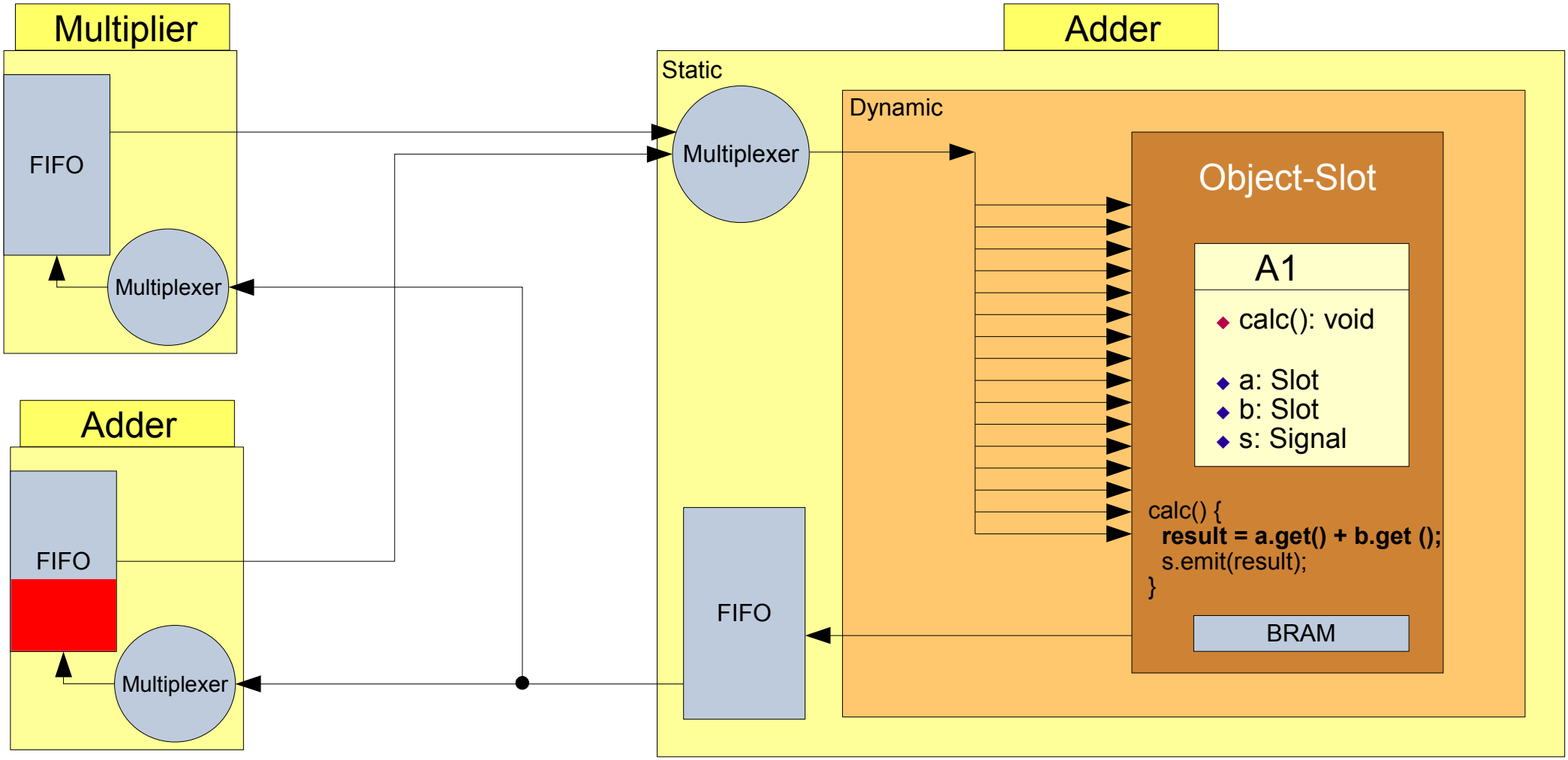
The communication matrix



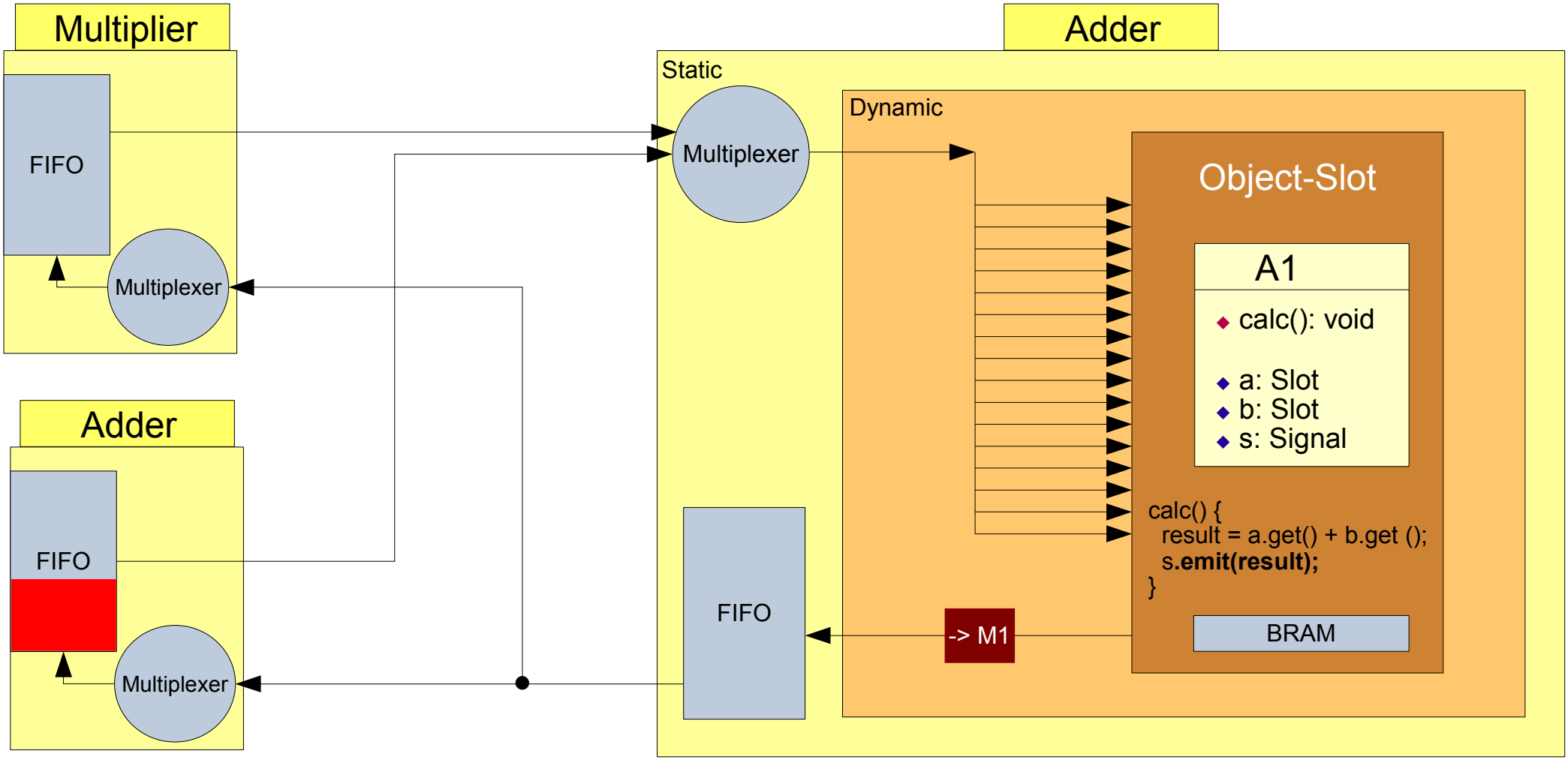
The communication matrix



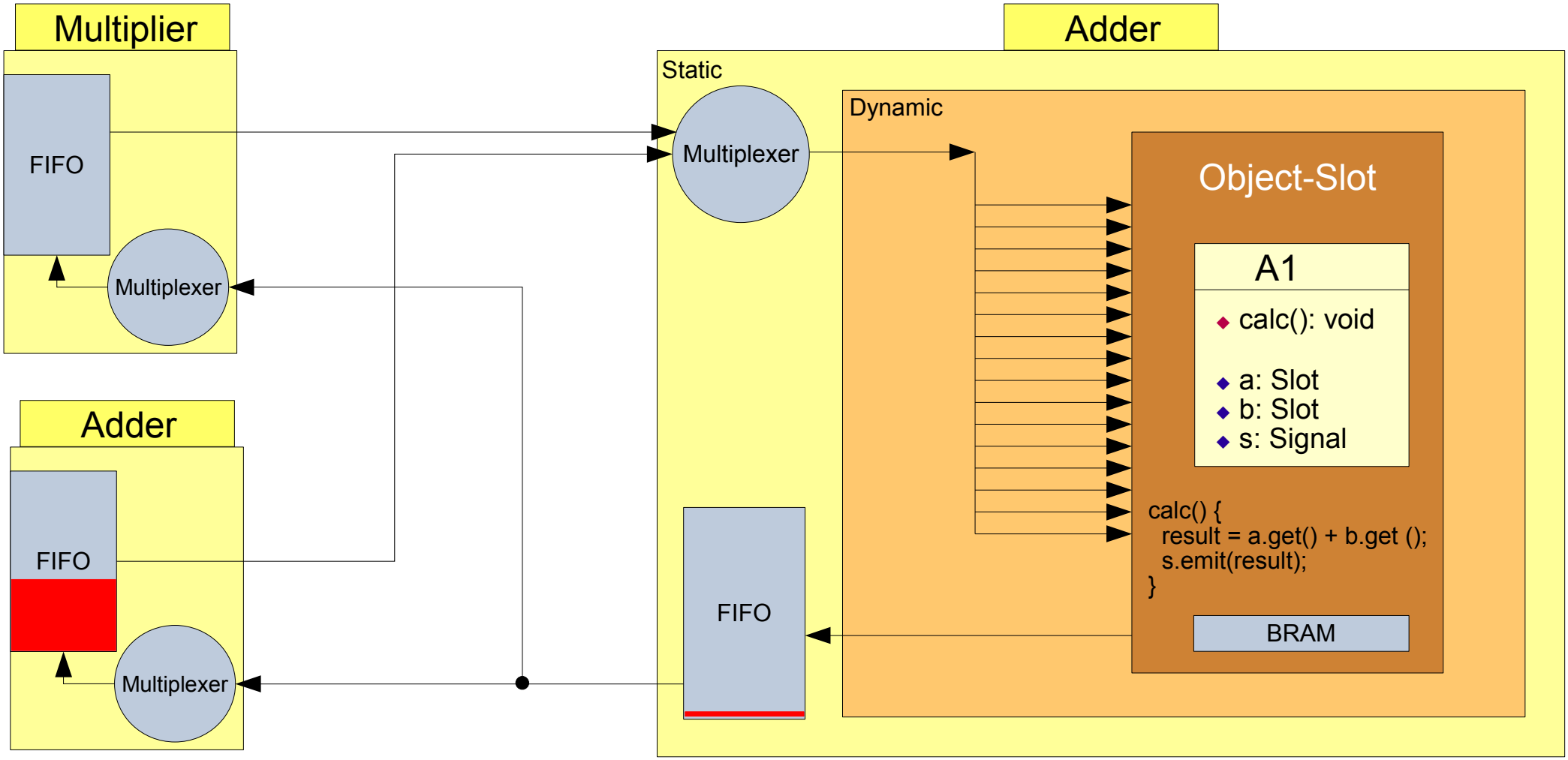
The communication matrix



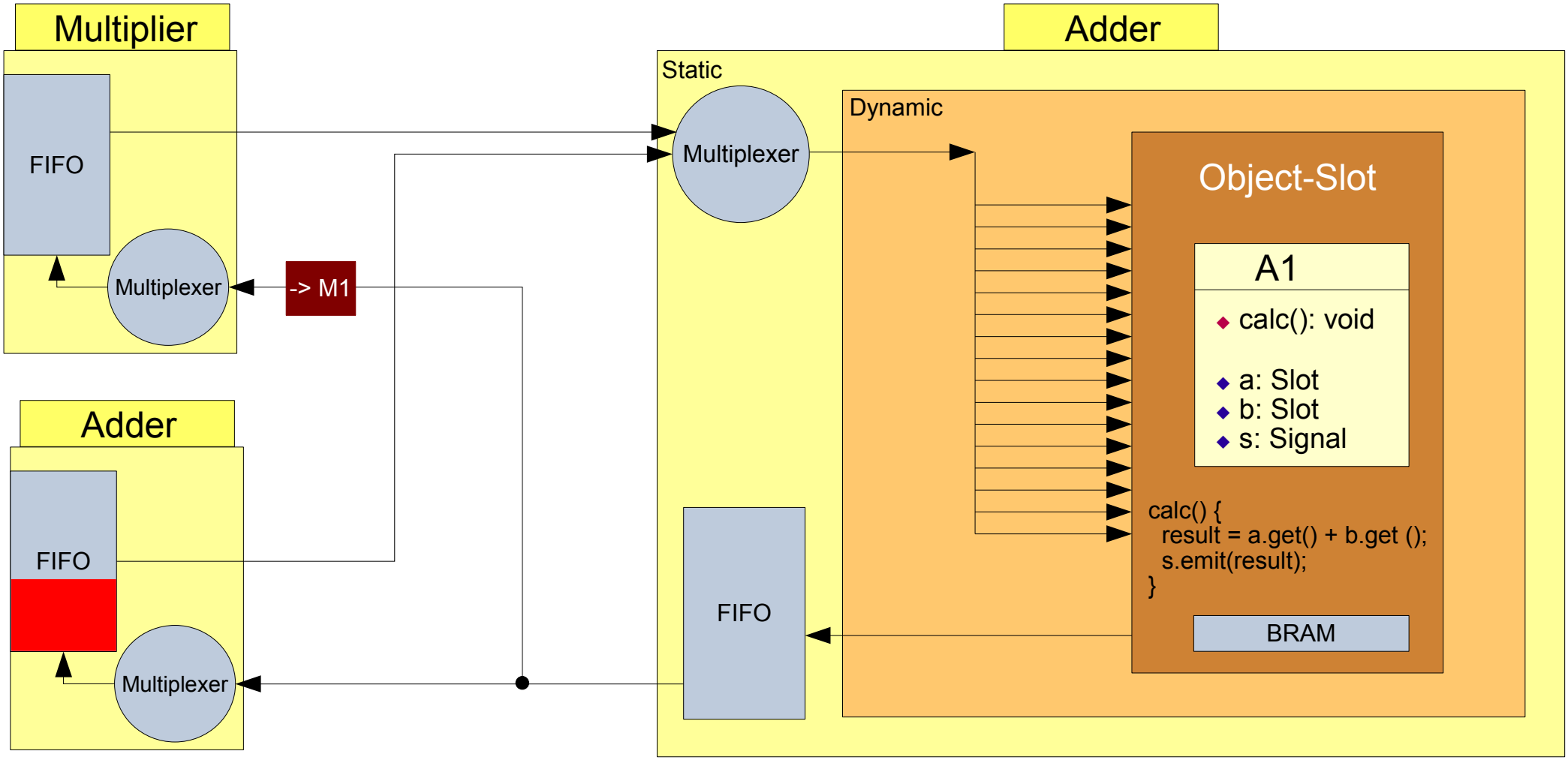
The communication matrix



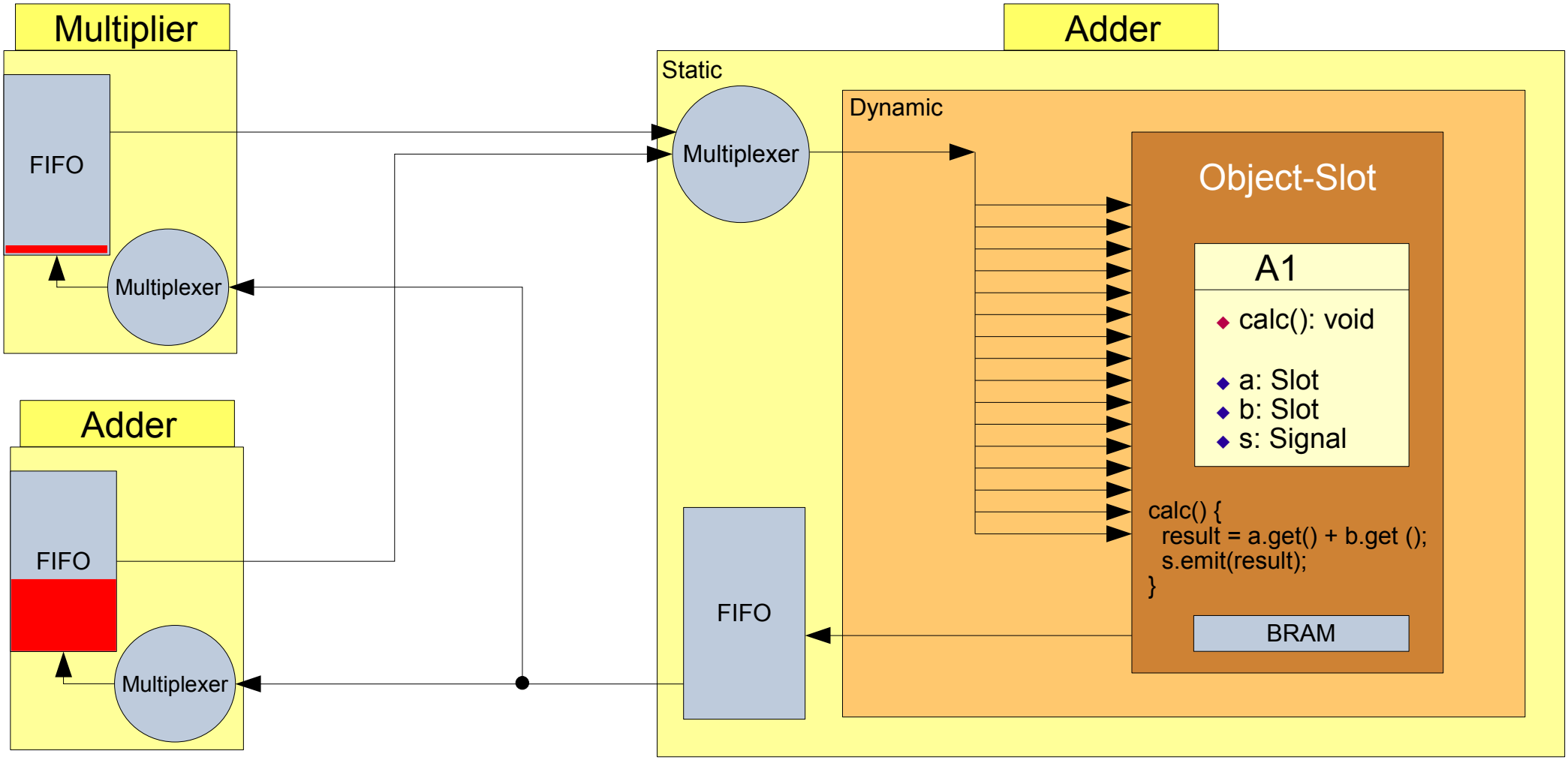
The communication matrix



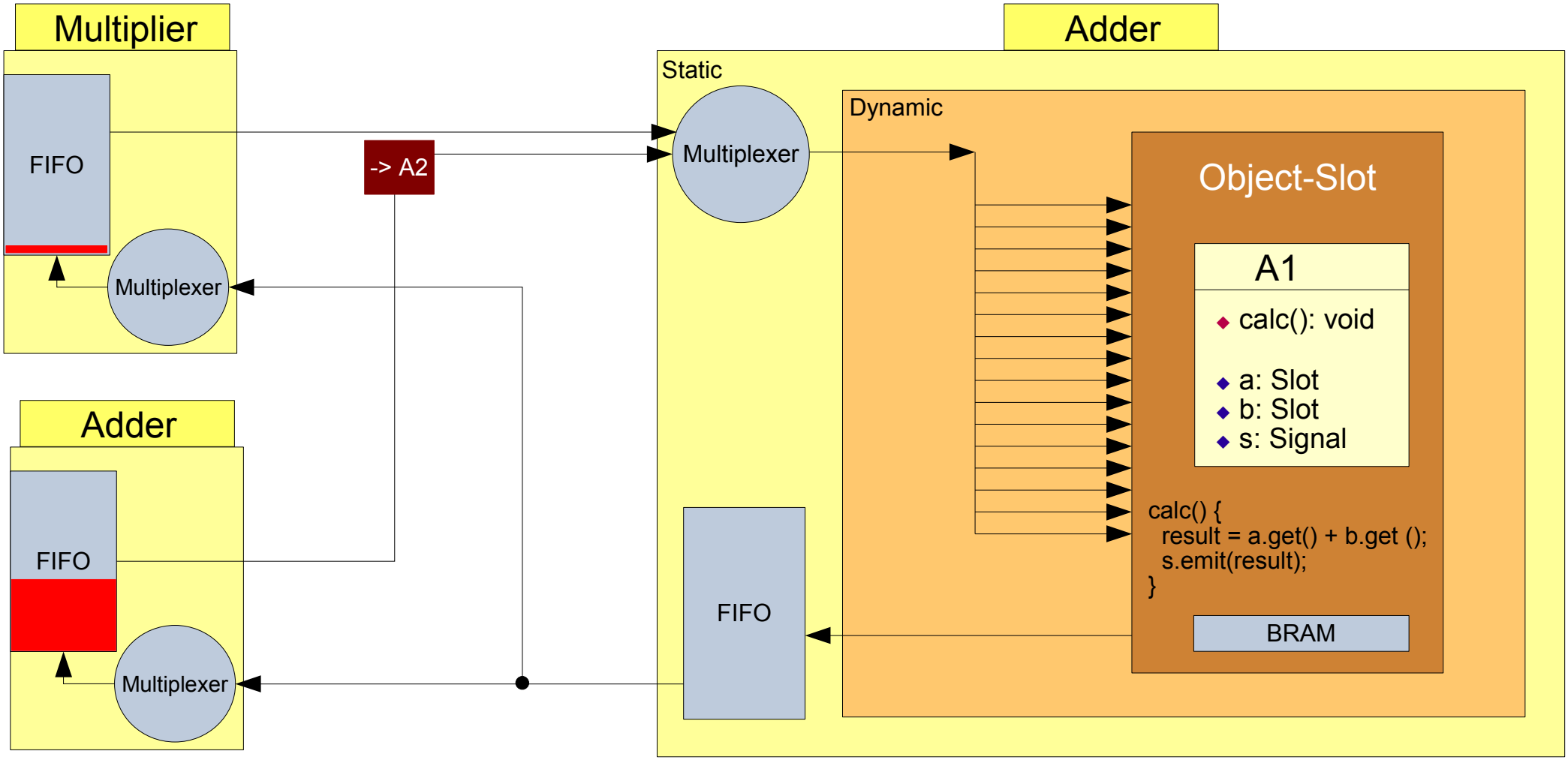
The communication matrix



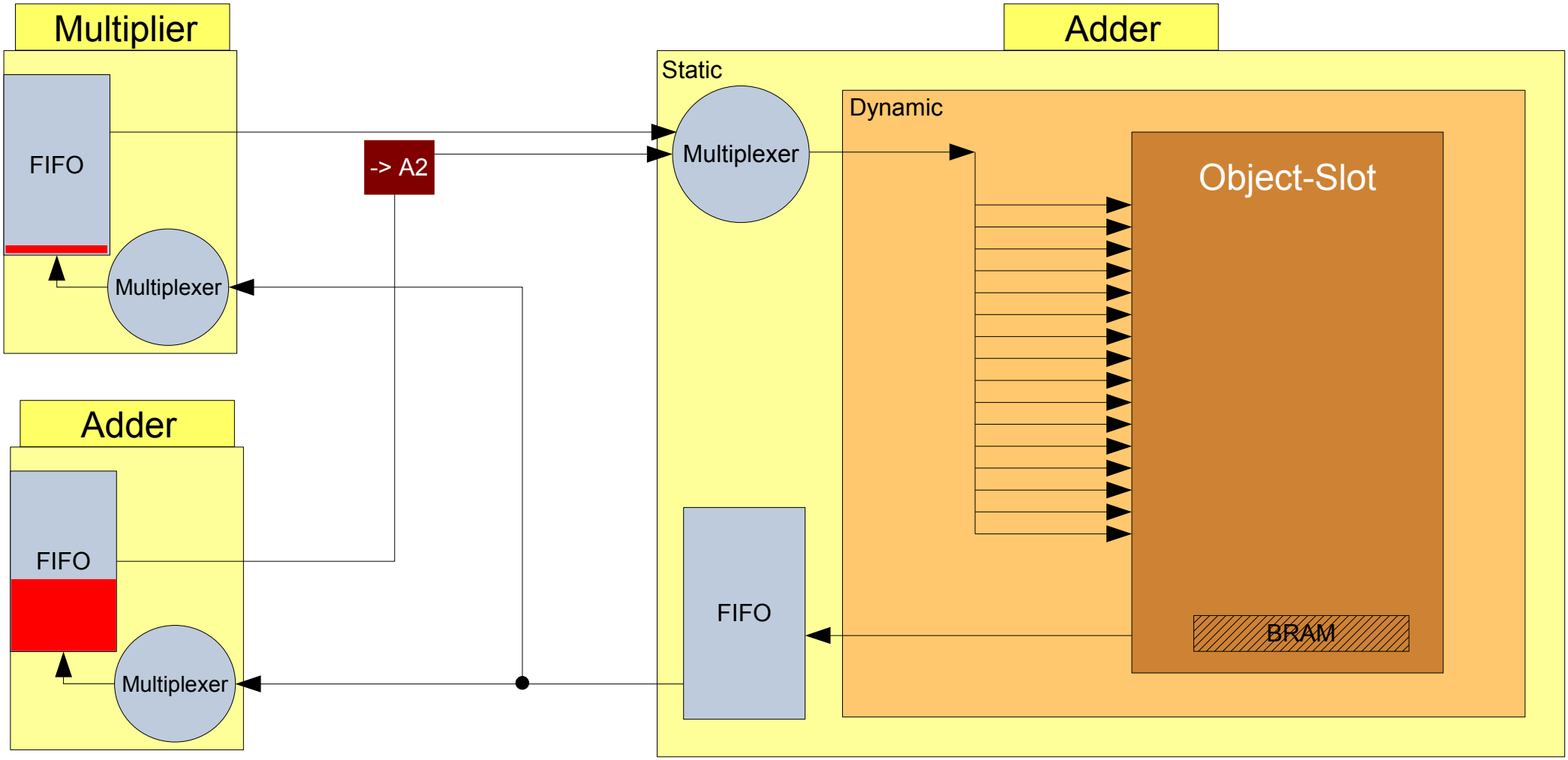
The communication matrix



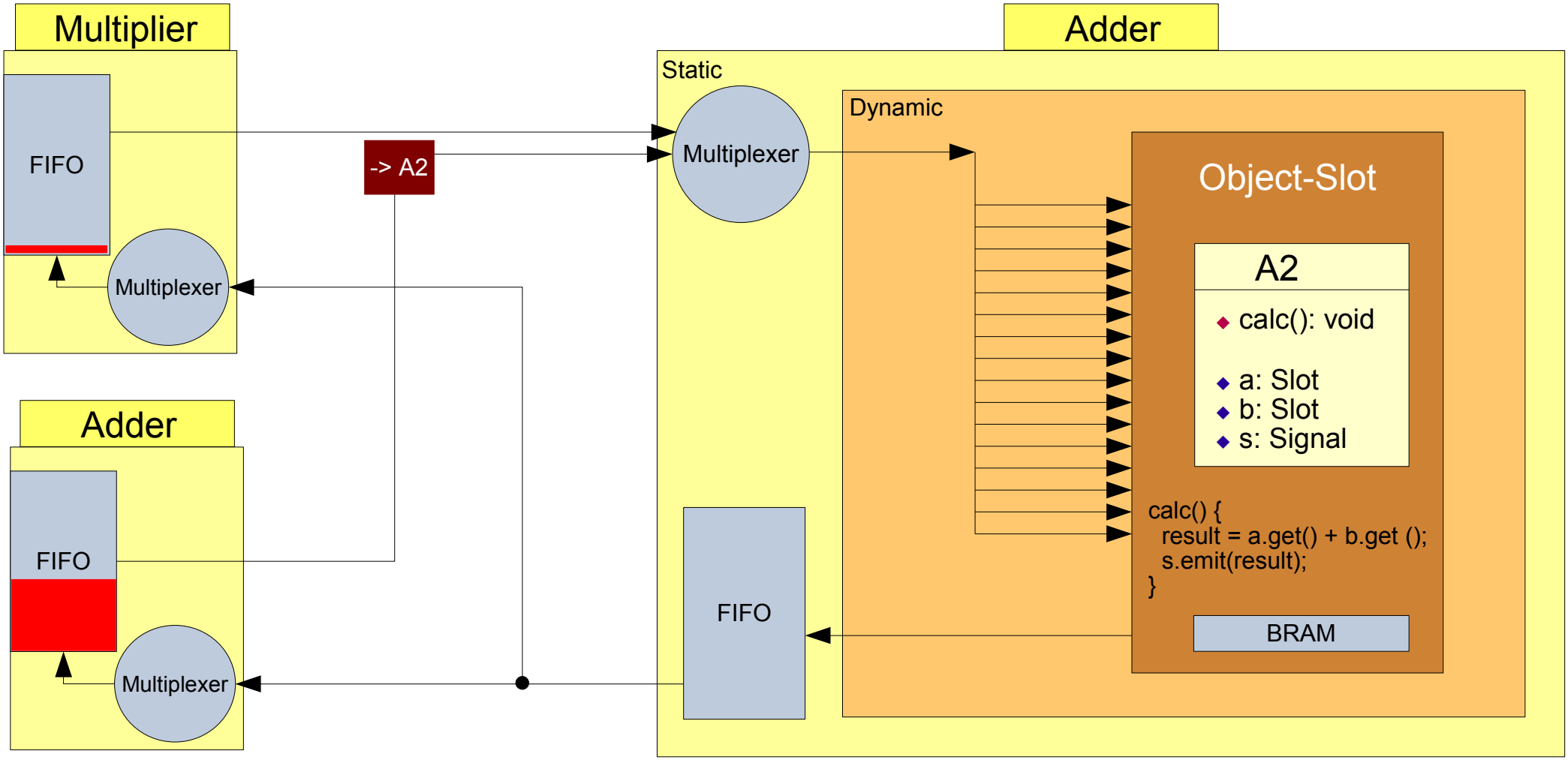
The communication matrix



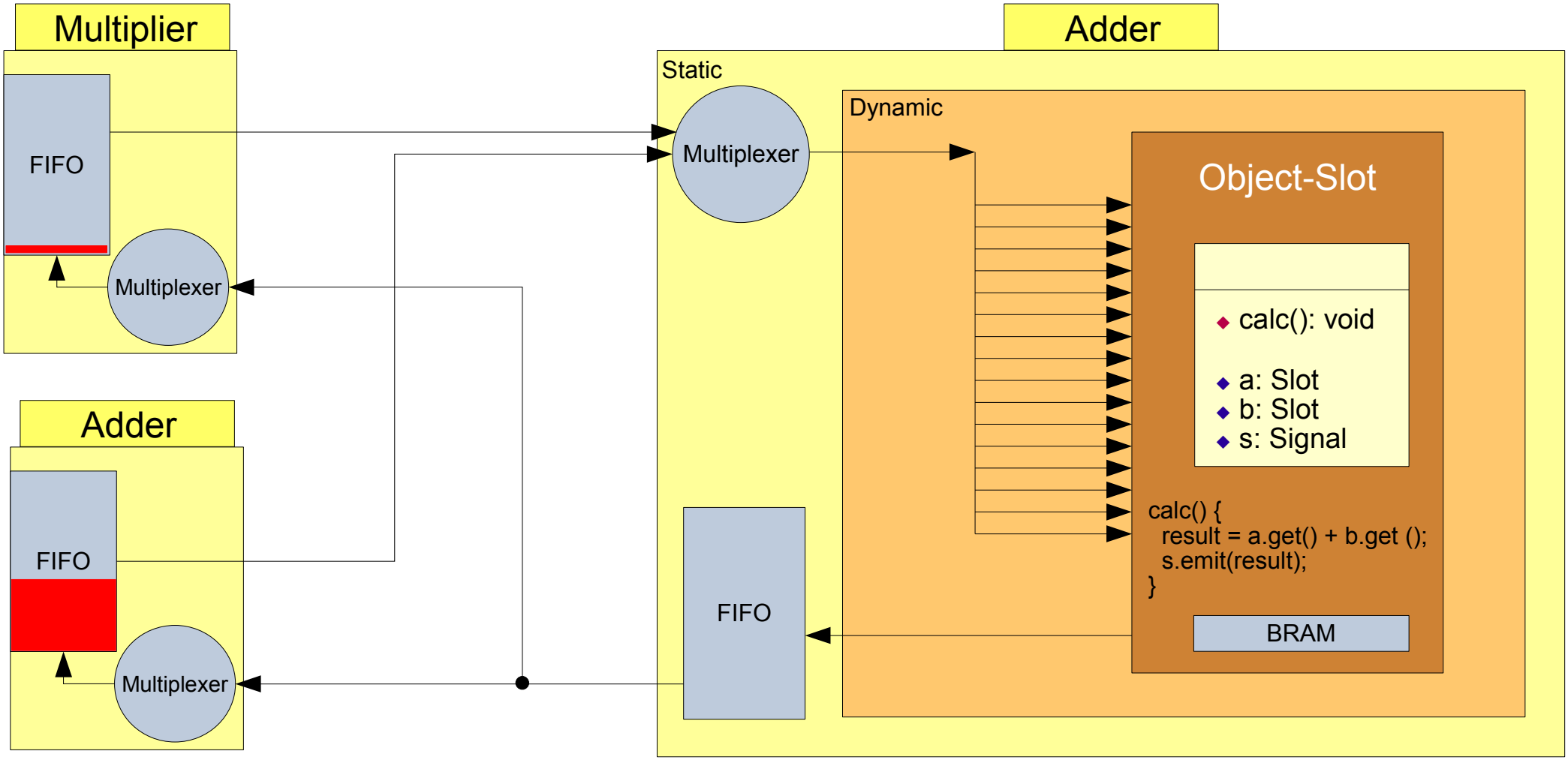
The communication matrix



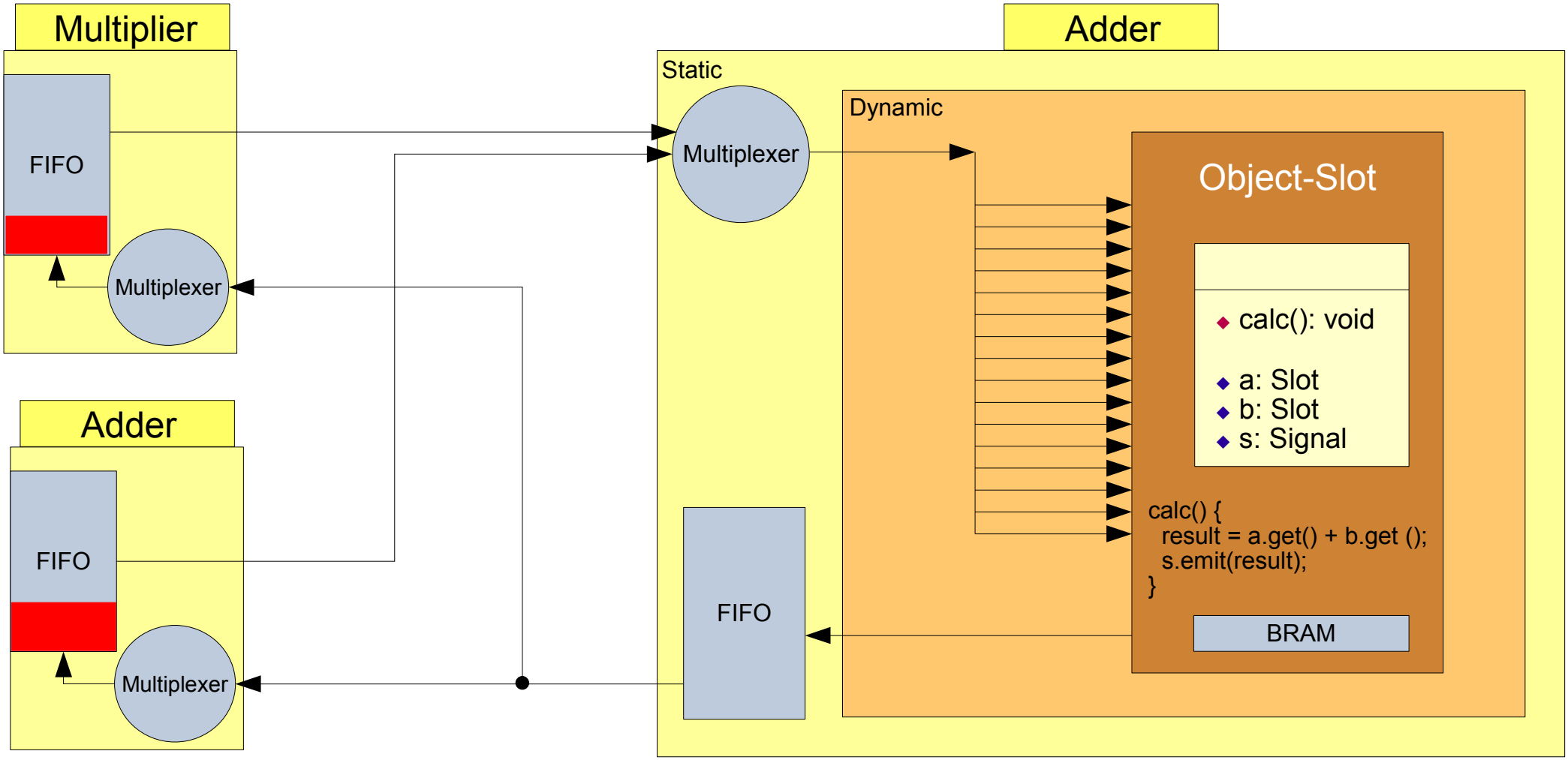
The communication matrix



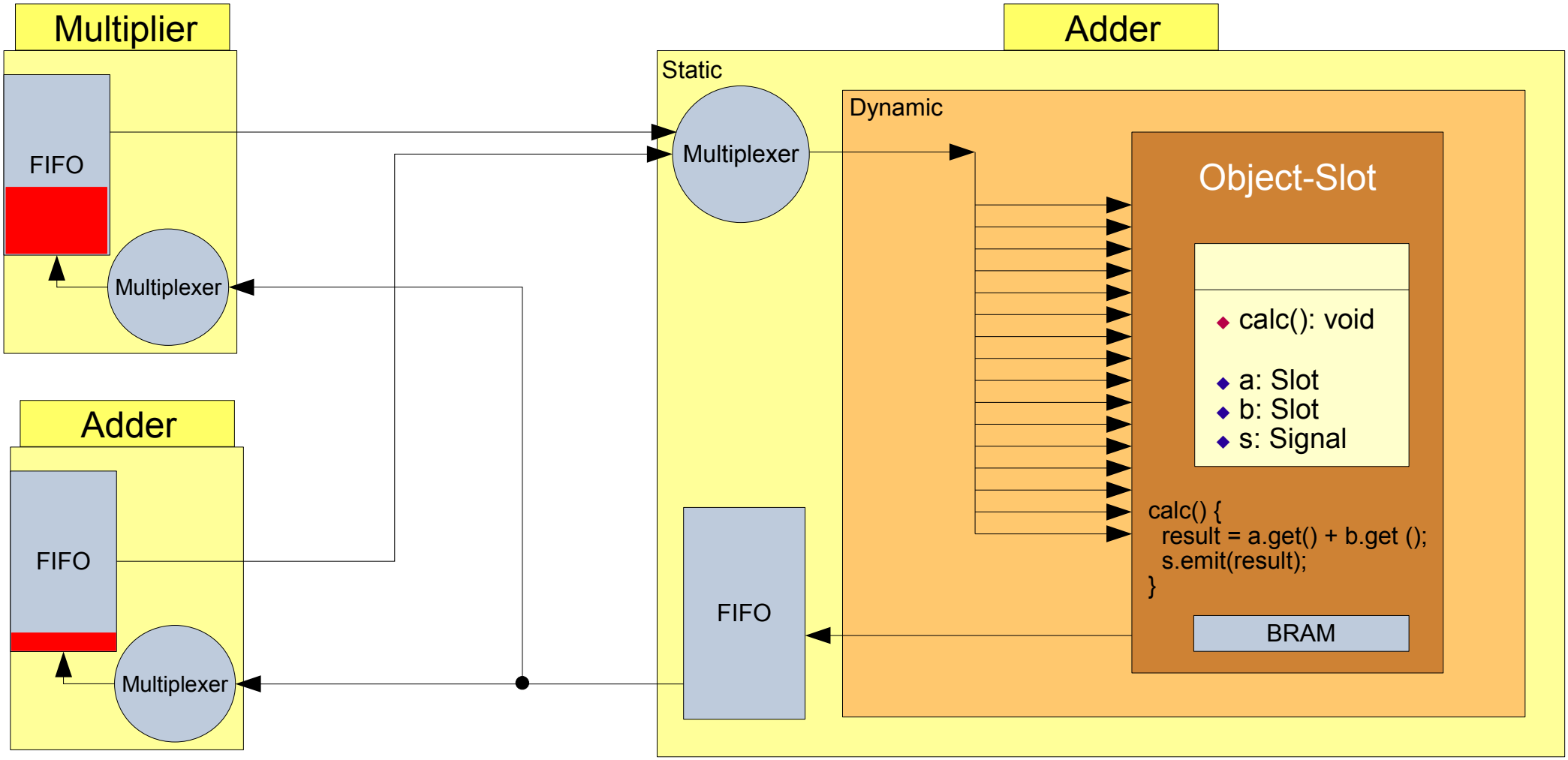
The communication matrix



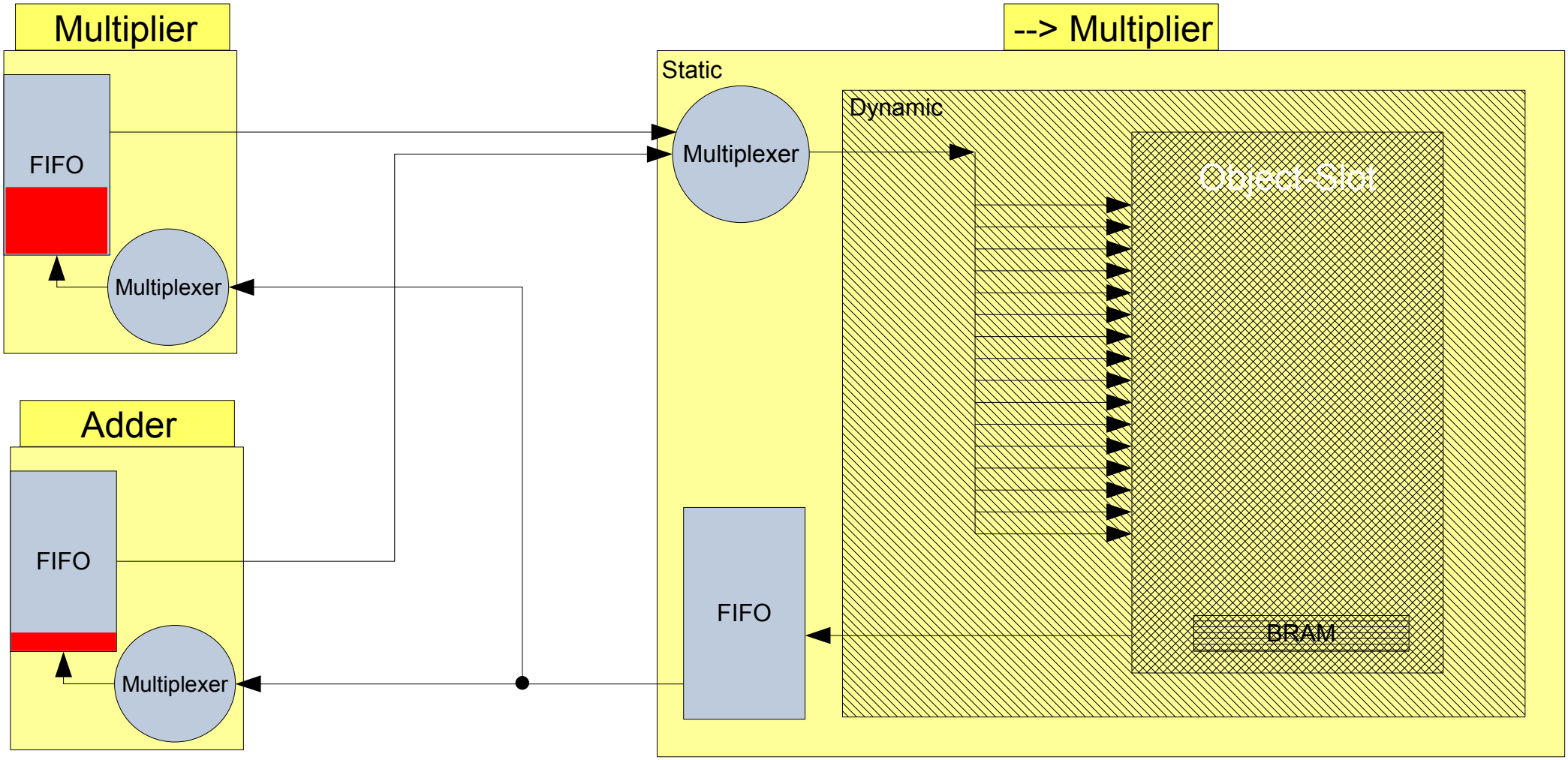
The communication matrix



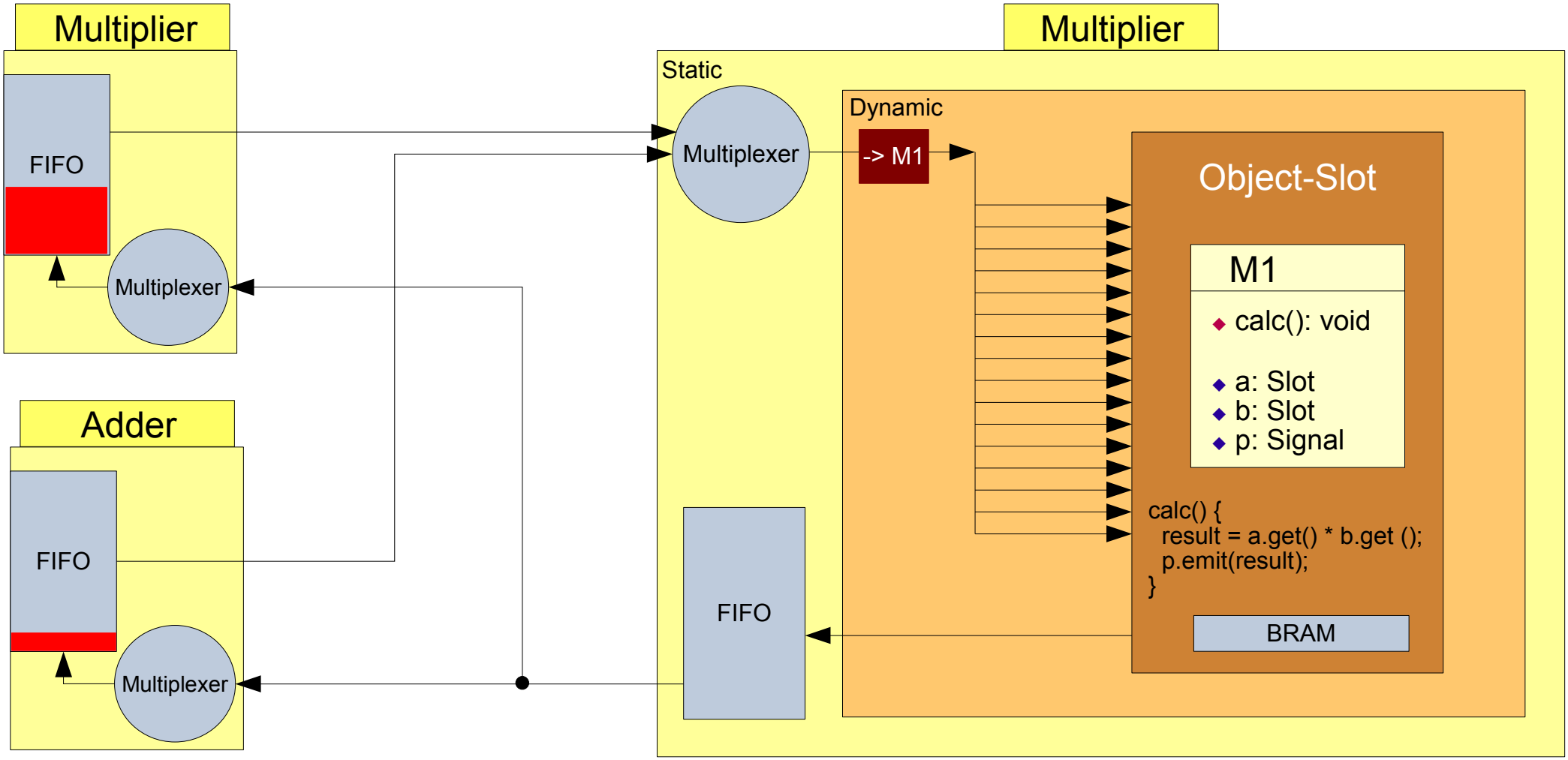
The communication matrix



The communication matrix



The communication matrix



The POL commands

Slot *in*;	Deklariert den Eingang *in* einer Klasse
Signal *out*;	Deklariert den Ausgang *out* einer Klasse
***Sender.out*.connect(*Receiver.in*);**	Verbindet das Signal *out* des Objekts *Sender* mit dem Slot *in* des Objekts *Receiver*
***Sender.out*.disconnect(*Receiver.in*);**	Trennt das Signal *out* des Objekts *Sender* von dem Slot *in* des Objekts *Receiver*
***out*.emit(*value*);**	Sendet eine Nachricht mit Inhalt *value* an alle mit dem Signal *out* verbundenen Slots
***in*.get();**	Blockiert, bis eine Nachricht an Slot in anliegt und holt diese ab
***in*.get(*default*);**	Holt eine Nachricht von Slot in ab oder gibt *default* zurück, falls keine Nachricht anliegt