



**Ilia Oussorov, Wolfgang Raab,
Ulrich Hachmann, Alex Kravtsov**

SystemC - based gradual development flow from abstract C++ to ISS integration

Motivation

Start with the functional spec

Architectural
exploration



Fast, easy to change
virtual prototype

☞ High level model

- Fast implementation
- Fast simulation
- Architectural details disregarded

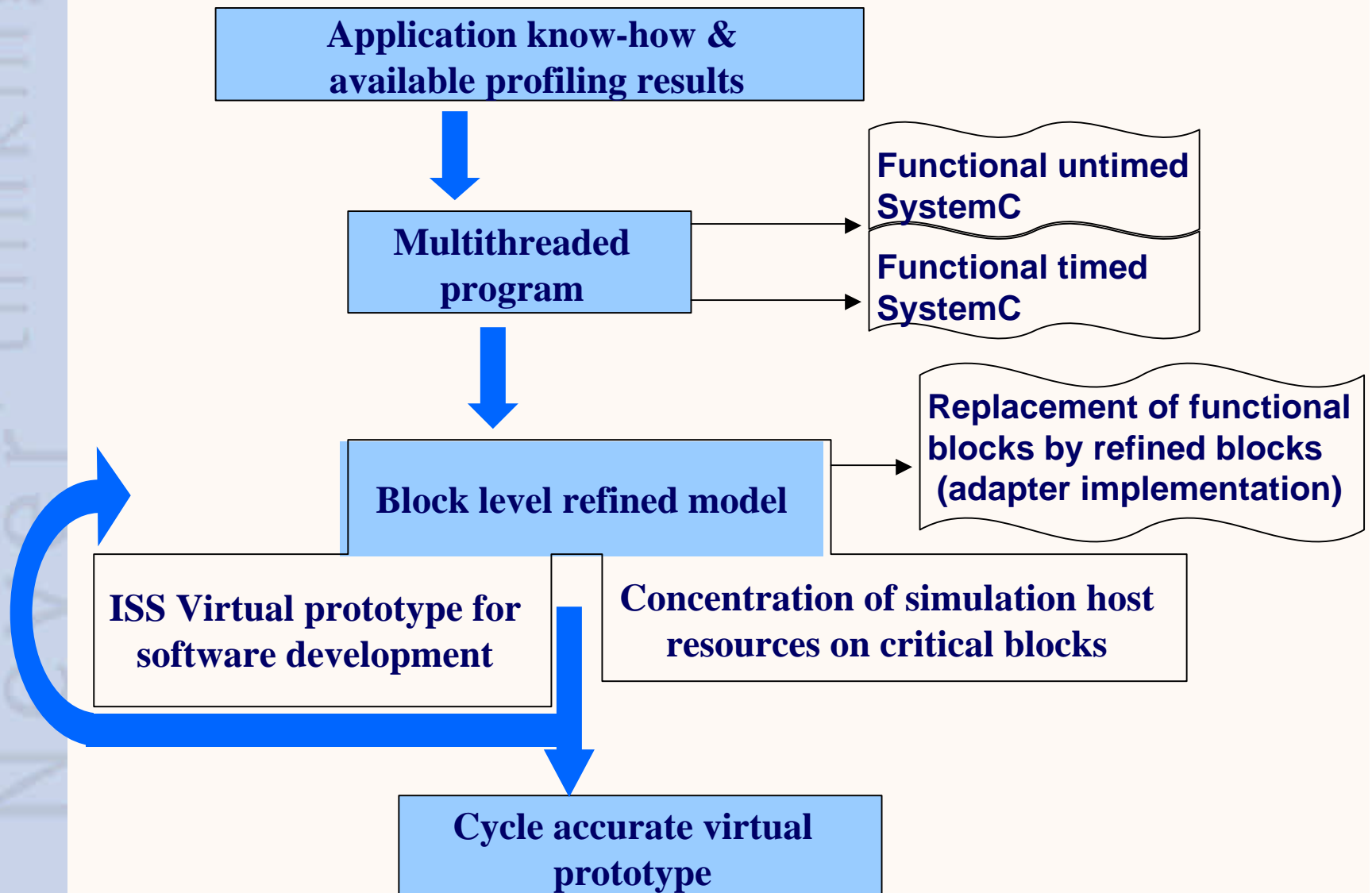
☞ Accurate virtual prototype

- Slow simulation techniques for accurate ISS
- At least linear dependency of simulation performance on the core number

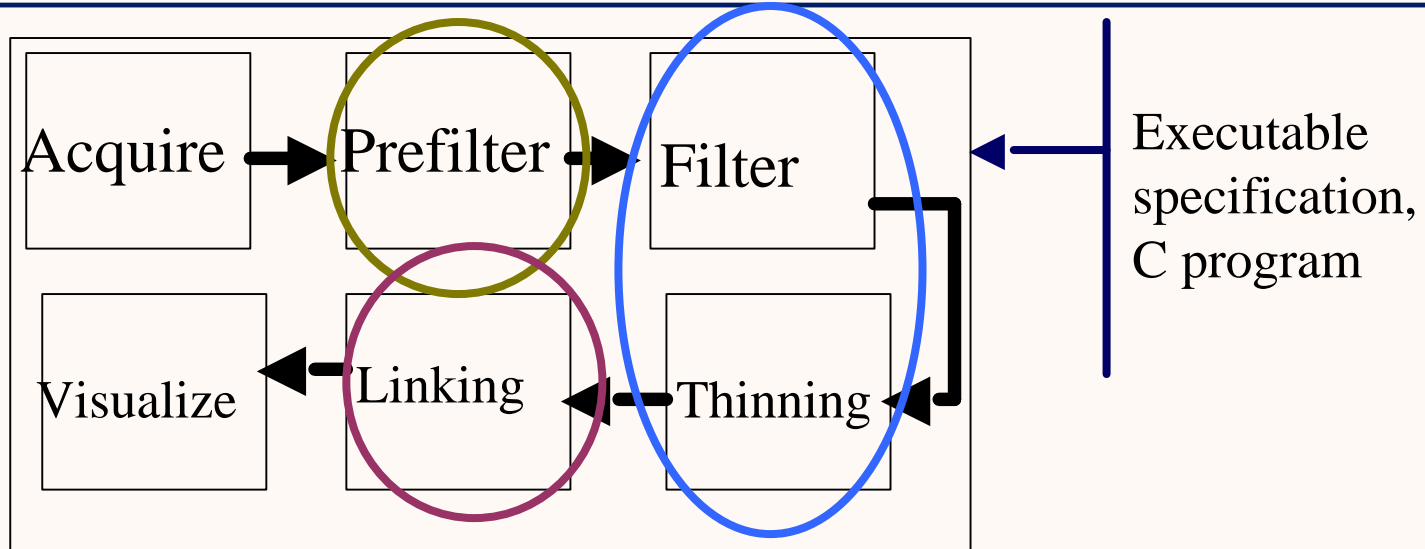
Problem:

Simulation Speed vs. Architectural Details

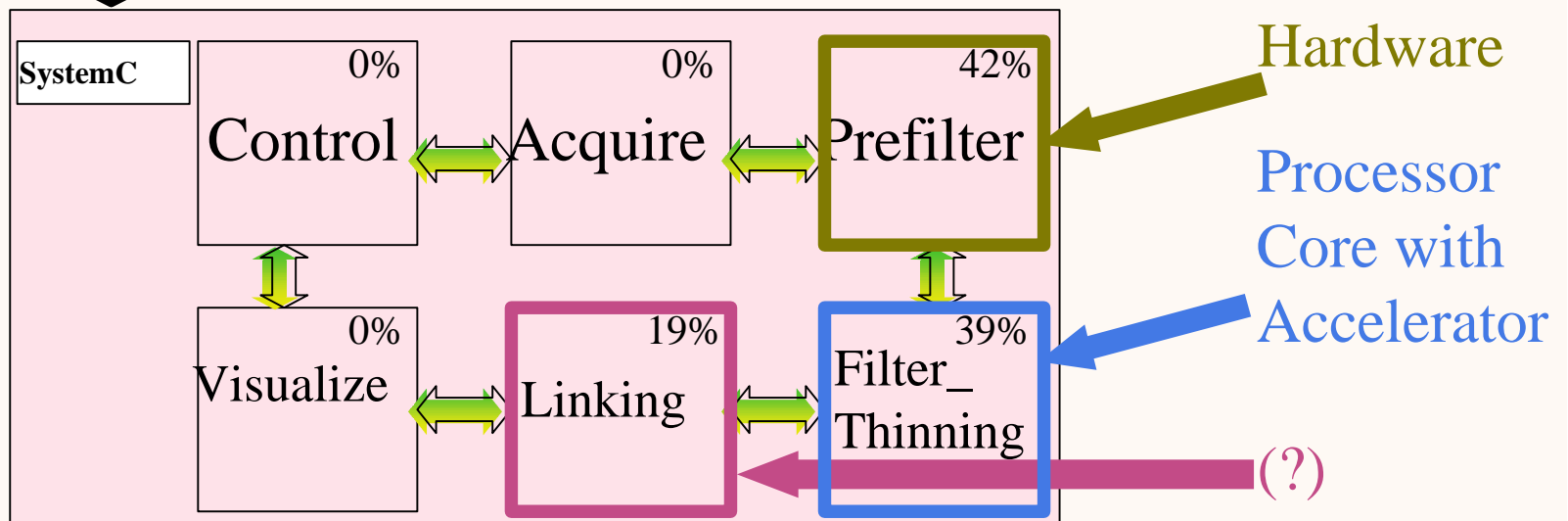
Concept of Architecture Development Flow



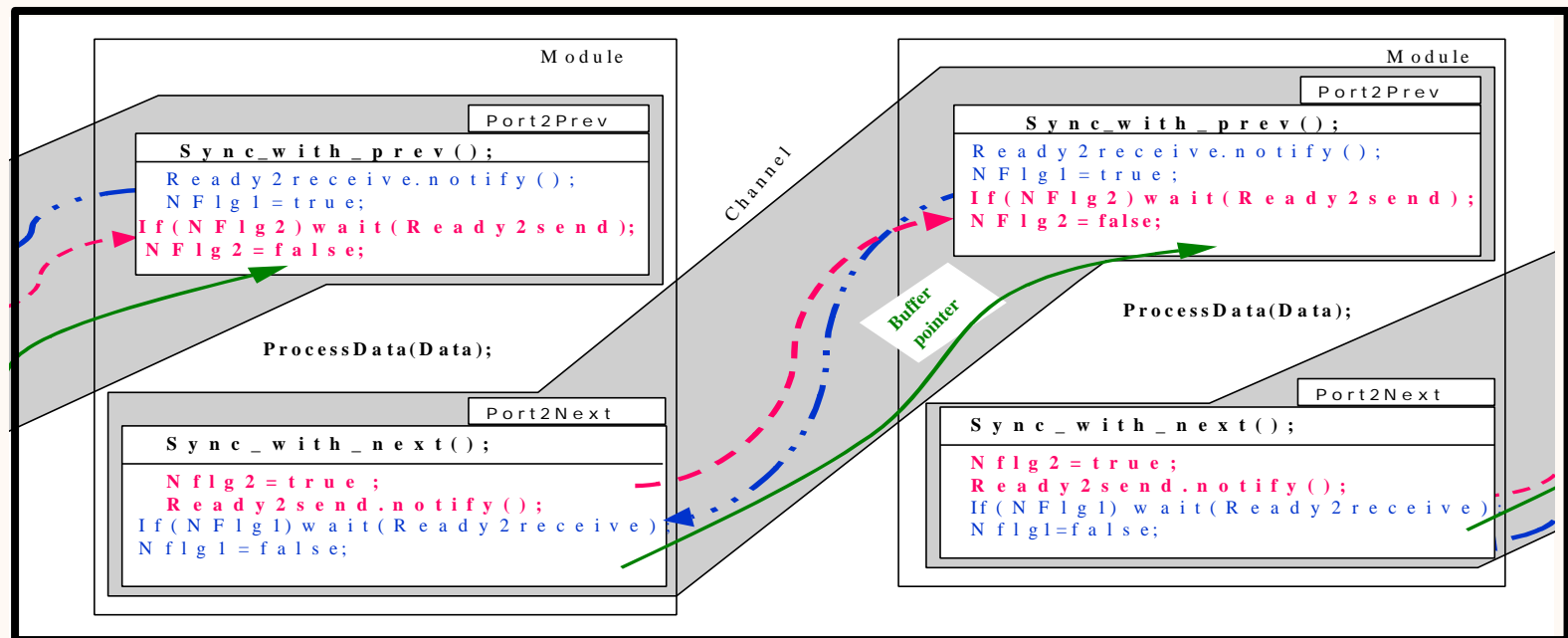
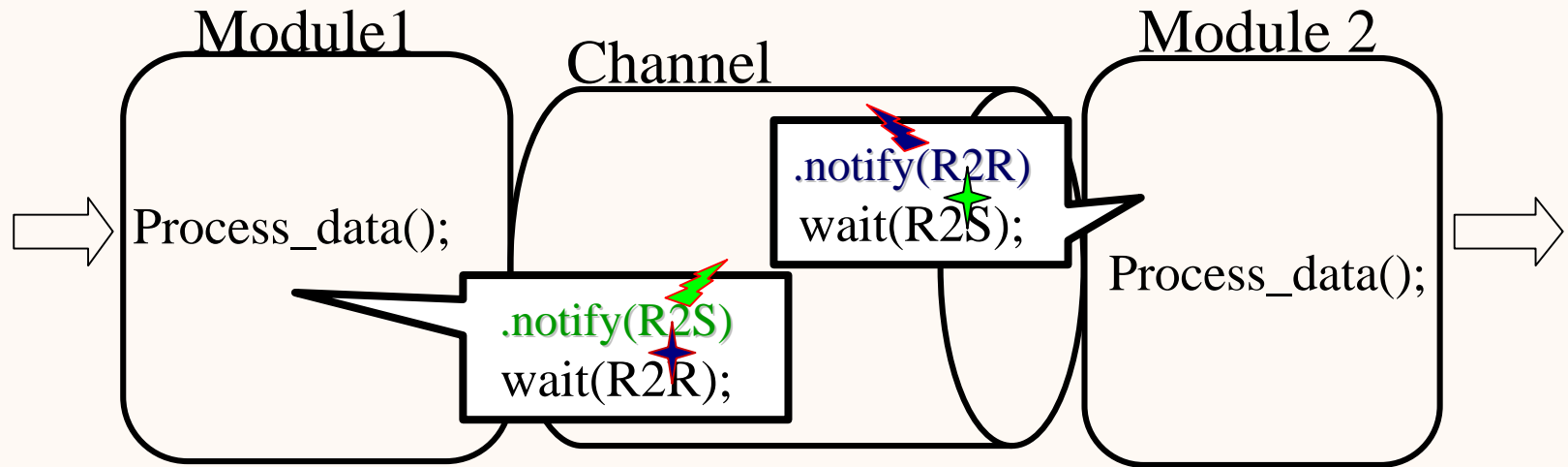
High Level Models



↓ Profiling on PC (about 1100MIPS)



Why SystemC events were not sufficient?



Initial Measurements

Host: Sun Ultra 10, cpu 333Mhz

C Specification: 0.035 sec per frame

Untimed SystemC model: 0.036 sec per frame

Paralleled timed SystemC model: 0.040 sec per frame

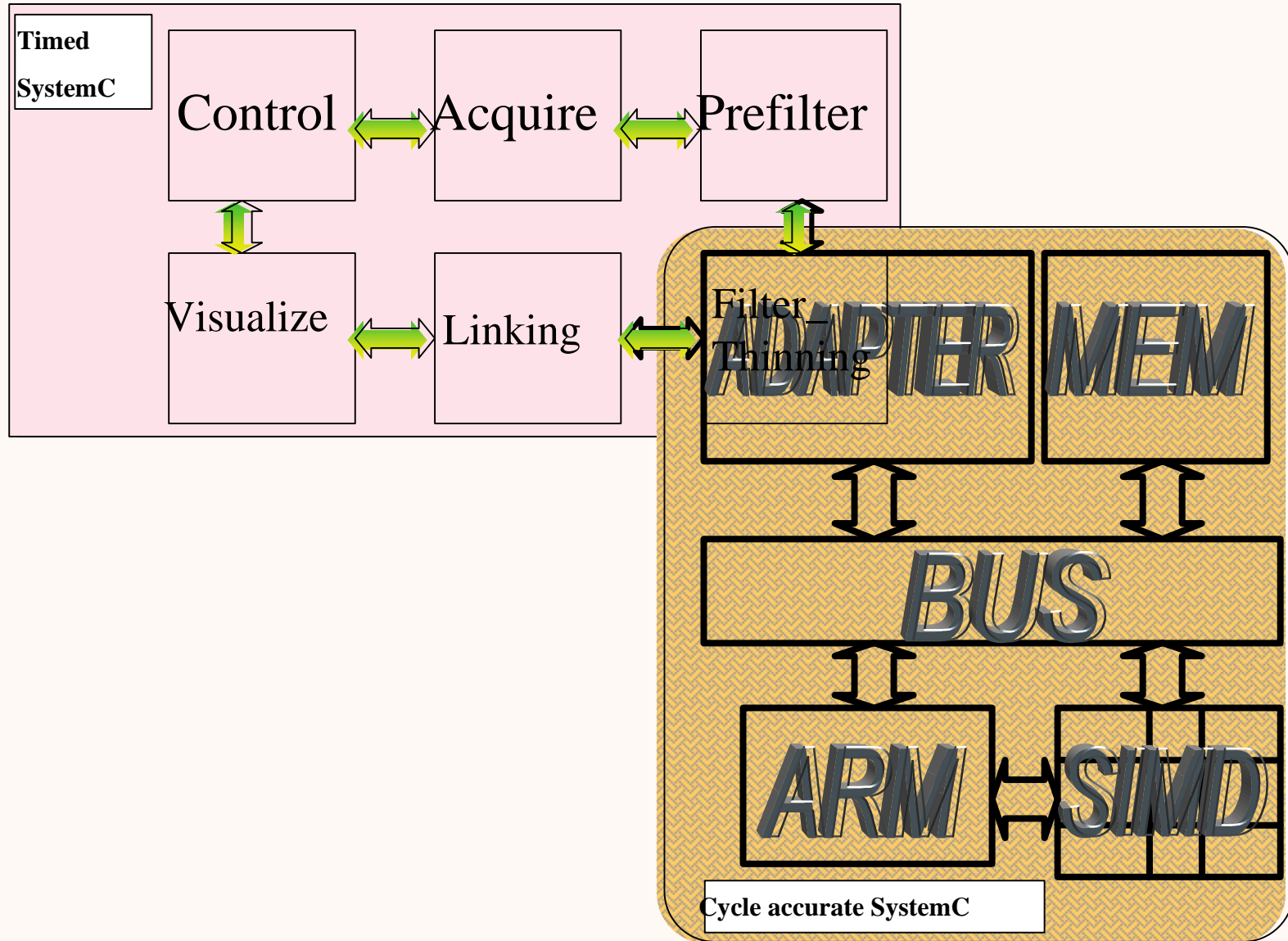
Armulator 920:

Specification: 45 sec per frame

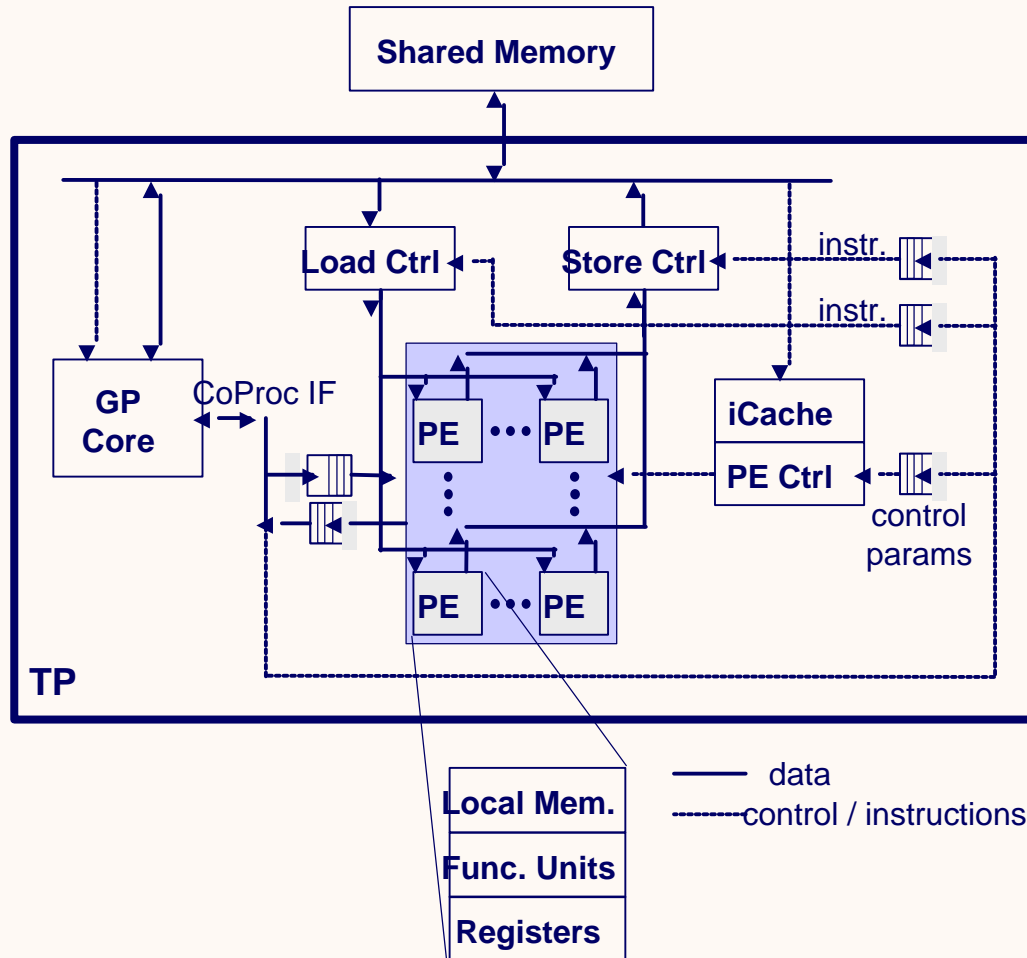
1 cycle of “accurate” ISS about 1000 host cycles

Refine partially, leave testbench for each block functional

Timed Functional- Architectural Cosimulation



SIMD Array



- Parameters:**
- Local memory size
 - Memory bus widths
 - Number of PEs
 - Number of registers

Core Integration Requirements

1. Instruction set simulators

- ⚙ SIMD array
- ⚙ ARM

2. Connection of high level model and refined model

- ⚙ Communication
- ⚙ Synchronization

3. Core software

- ⚙ Initialization routines
- ⚙ Synchronization drivers

Integration of SIMD model into SystemC

```

class simd_wrapper:public simd_if
{
    simd * pSimd;
    sc_port<bus_if> SU_port
SC_CTOR(simd_wrapper)
{
    pSimd=new simd(&SU_port);
    SC_METHOD (action);
    sensitive_pos<<clk;
}
}

void action(void) {simd->DoClock();}
void StoreFifo(...) {simd->StoreFifo(...);}

```

```

class simd
{
private:
    sc_port<bus_if> * SU_port;
public:
    StoreFifo (data_t &_data)
    write_bus(...);
    DoClock();
    simd(sc_port<bus_if> * _SU_port)
    {SU_port=_SU_port;}
}

bool write_bus(..)
{
    SU_port->write(..)
}

```

How to pass port pointer to the bus access function if the ISS API is fix?

```

sc_port<bus_if> * pSU_port;
class simd_wrapper:public simd_if
{
    simd * pSimd;
    sc_port<bus_if> SU_port
    SC_CTOR(simd_wrapper)
    {
        pSimd=new simd ();
        SC_METHOD (action);
        sensitive_pos<<clk;
    }
}
void action(void) {
    pSU_port=&SU_port;
    simd->DoClock();
}

```

```

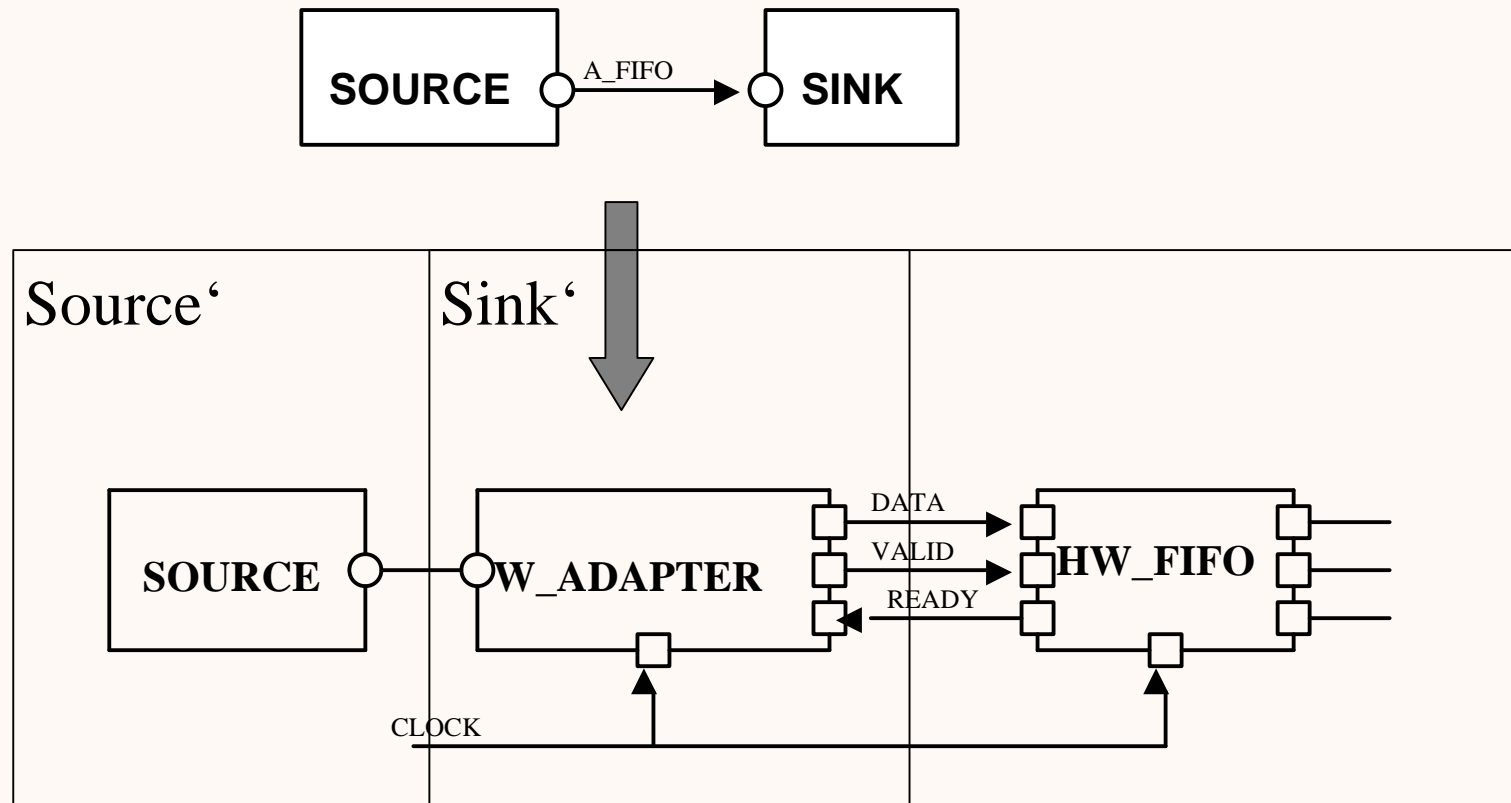
extern sc_port<bus_if> * pSU_port;
class simd
{
private:
    sc_port<bus_if> * SU_port;
public:
    StoreFifo (data_t &_data)
    write_bus(..);
    DoClock();
    simd()
}
bool write_bus(..)
{
    pSU_port->write(..);
}

```

Solution for multiple instantiation

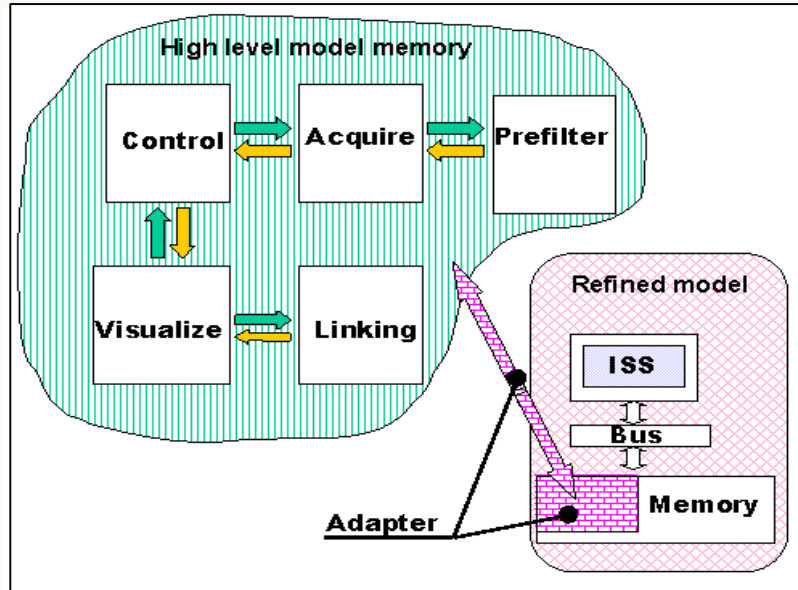
Communication Refinement

Functional specification for SystemC 2.0, chapt. 12:



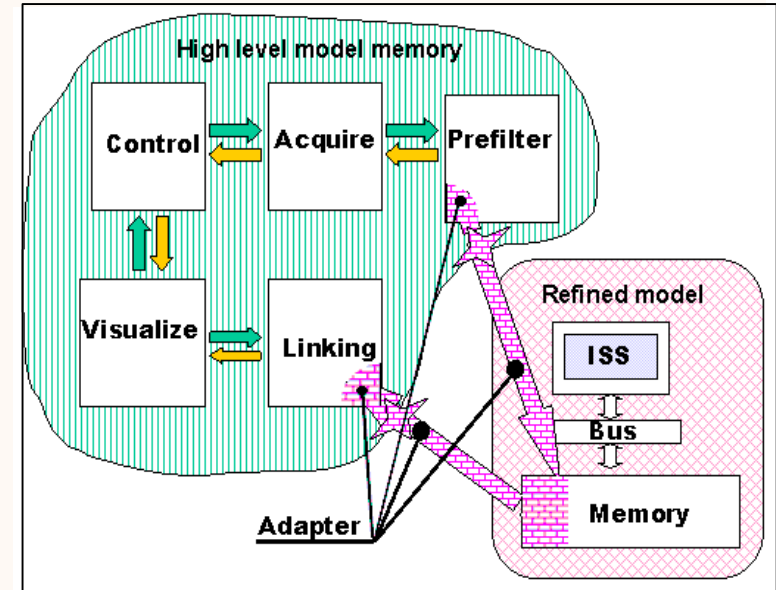
Examples of Communication Adapters

Shared Memory Adapter



- ⚙ Implemented within memory model
- ⚙ Maps the communication variables of ISS software into shared variables within the high level model
- ⚙ Converts transferred data

Transfer Adapter



- ⚙ Implemented as transfer routine within high level blocks
- ⚙ Transfers the communication data between high level model and core memory
- ⚙ Converts transferred data

Synchronization

■ High level model to low level model

- interrupts
- direct writing into memory of processor core (to be polled by core)

■ Low level model to high level model

- event activation by reading/writing of memory mapped event register

```
#define ADDR_EVENT_REG 12345
...
if (_address==(ADDR_EVENT_REG-start_addr))
  switch (*data)
    case 0:
      Event0.notify();
      break;
    case 1:
      Event1.notify();
      break;
    ...
    default:
      ReportUndefinedEventAndExit();
end;
```

What do we reach with this flow?

General benefit:

- Selective refinement → Fast virtual prototype (less accurate on the uncritical parts)
- Concentration on the module under exploration

Realistic testbench for refined blocks

Minor changes of application software

Architectural exploration:

Accurate profiling of SIMD array bus loading, scaling of PE array

Accurate profiling of SIMD instruction set

Verification of interworking between SIMD array and processor



Ilia Oussorov

Oussorov.external@infineon.com

Wolfgang Raab

Wolfgang.Raab@infineon.com

Ulrich Hachmann

Ulrich.Hachmann@infineon.com

Alexander Kravtsov

Alexander.Kravtsov@infineon.com



infineon **Technologies AG**

CPR ST

D-81730 Munich,

Germany

„Never stop thinking“