Framed Complexity Analysis in SystemC for Multi-level Design Space Exploration

Armin Wellig

Advanced System Technology

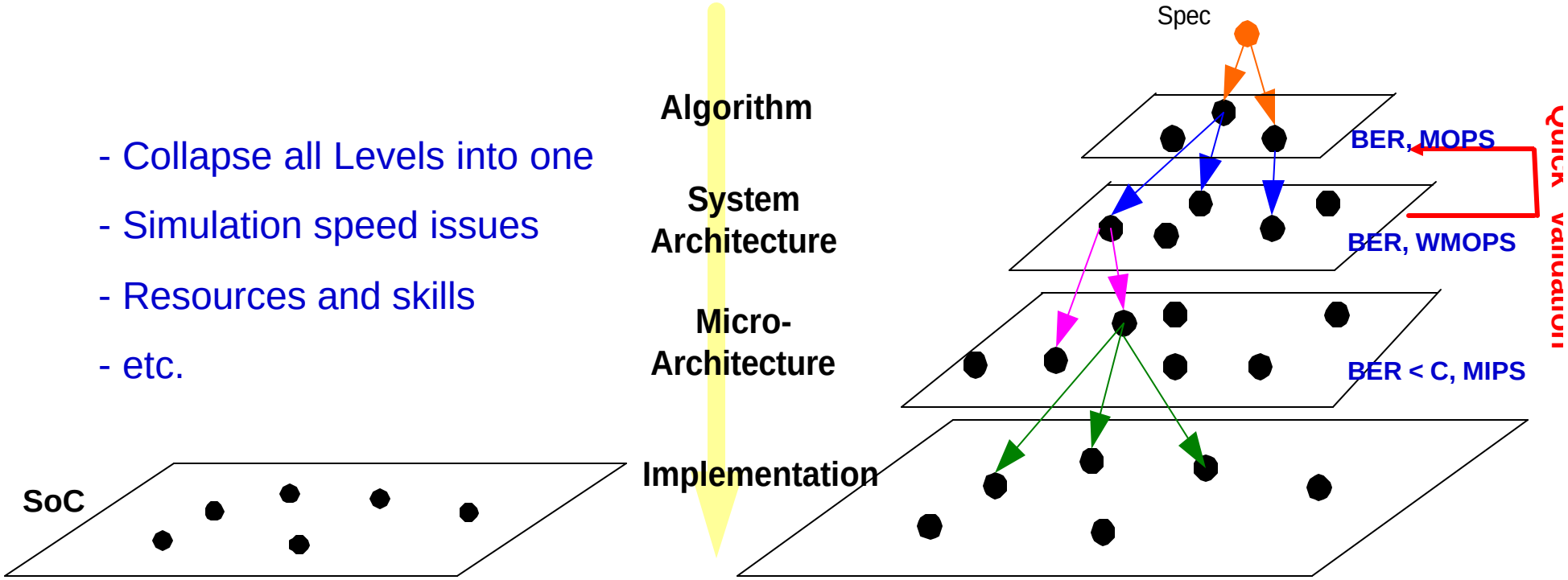
STMicroelectronics, Geneva Laboratory

Outline

- **Motivation**
- **Complexity Tool**
  - Motivation
  - Objectives of prototype
  - Classification in Behaviour, Storage and Communication Channels
  - Some implementation details
- **Application Example**
  - Stopping Criteria of Turbo Decoders
- **Conclusion**

Motivation

- Collapse all Levels into one
- Simulation speed issues
- Resources and skills
- etc.



- Communication performance (BER, QoS, etc.)
- Gate count (die area)
-Power consumption
- etc.

- Greedy resolution of several multi-objective problems
- Top-down metrics refinement process
- As short as possible feedback path

Complexity Tool: Overview

- Define a set of relevant **metrics** for each design level
  - Algo : number of operations, worst case memory size, …
  - Archi : bus traffic, required memory bandwith, ...
  - Impl : number of gates, silicon area, ...
- Specify a **methodology** to reuse and/or transform these metrics from one level to the next one
  - Be able to reuse the metrics during the transition phase
  - Use the metrics as feedback medium
  - Automate the complexity analysis **early on** in the system design phase
- Implement **monitoring techniques**
  - Within SystemC
  - Flexible enough to support different design level needs

Objectives of our Prototype

- **Compatibility** with the SystemC simulation kernel semantics
- **Seamless use** of the tool by the user
  - The user can use the present SystemC models with minimal change to the syntax
- **Transparent** to SystemC evolution
  - SystemC is constantly evolving – develop a Tool that is loosely coupled to SystemC so that both of them can evolve independently but cooperate afterwards
- **Limited** additional **cost** in simulation time
  - Limited control and data communication bandwidth
  - Well suited for distributed simulations

Functional entity classification

- **Behavior**
  - Set of program statements ***computing*** something
  - SystemC modules, external functions, …
- **Channel**
  - ***Communication*** behavioral entities
  - Fifo, signals, bus interface, …
- **Variable**
  - Data stored in ***memory***
  - Parameters, scalar var., arrays, containers, …

Behavior monitoring: Operations

- Integer operations
  - C++ basic integral types (e.g. unsigned 32 bit integer)
  - SystemC types (e.g. sc_int<5>)
  - Fixed-point
- Floating-point operations
  - C++ Basic floating-point types (e.g. float)
  - Complex types (e.g. template class)
- Functions
  - Mathematics (cosine, log, …)
  - Intrinsic library (shift1add operation, dot product,…)

Behavior monitoring: Example

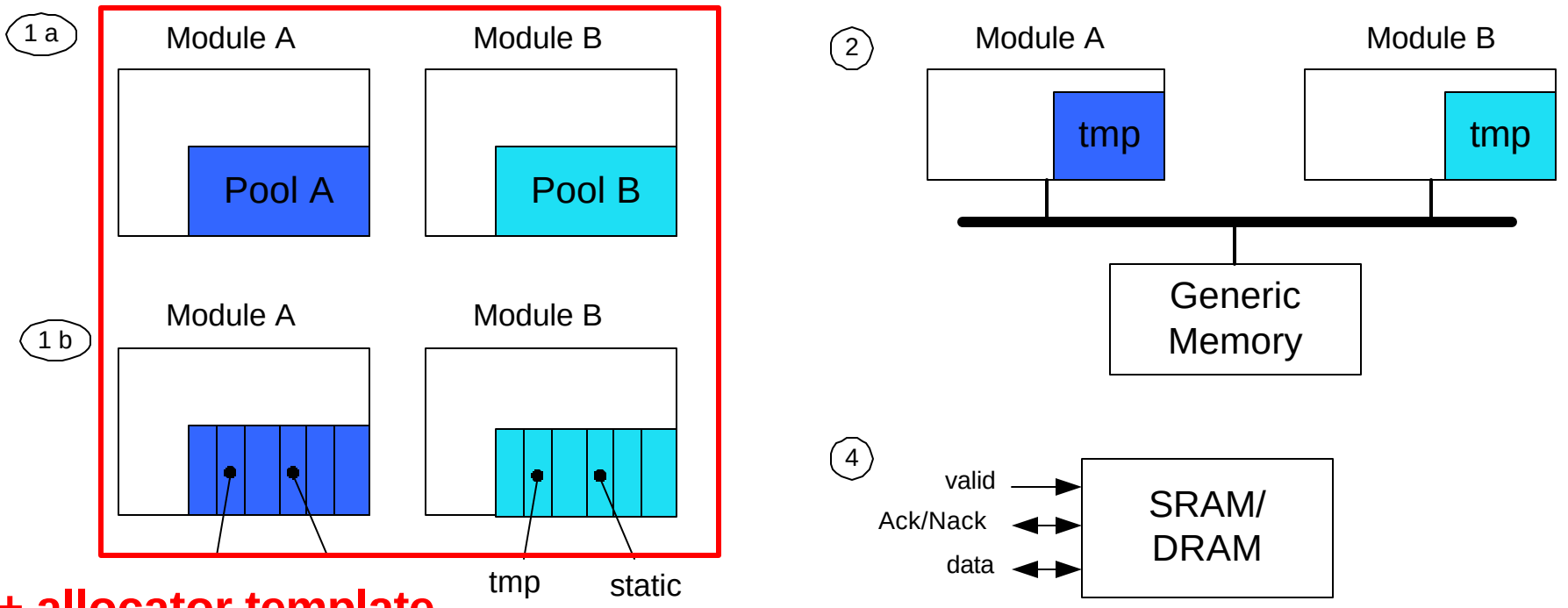
```
namespace wmops {
[ ...]
  template<int V>
   int_type operator + ( const sc_int<V>& v )
  {
    // Do the monitoring
    countOp( ADD, SC_INT, V );

    // Perform operation
    return ( *data + v );
  }
[ ...]
}
```

sc_dt::sc_int<W> *data;

Memory monitoring

- *Memory*:

  Memory pools → Functional memory modules → Transaction memory modules → Cycle-accurate memory modules

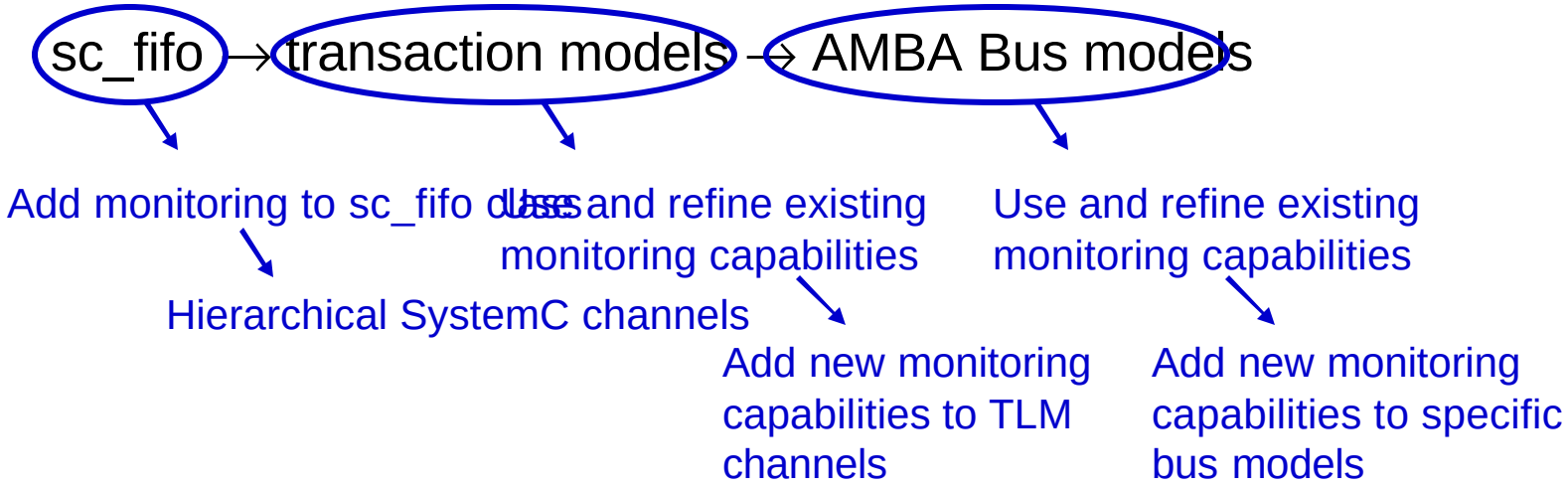


C++ allocator template

- Same implementation principles throughout the design levels: "*Record Events*"

Channel monitoring

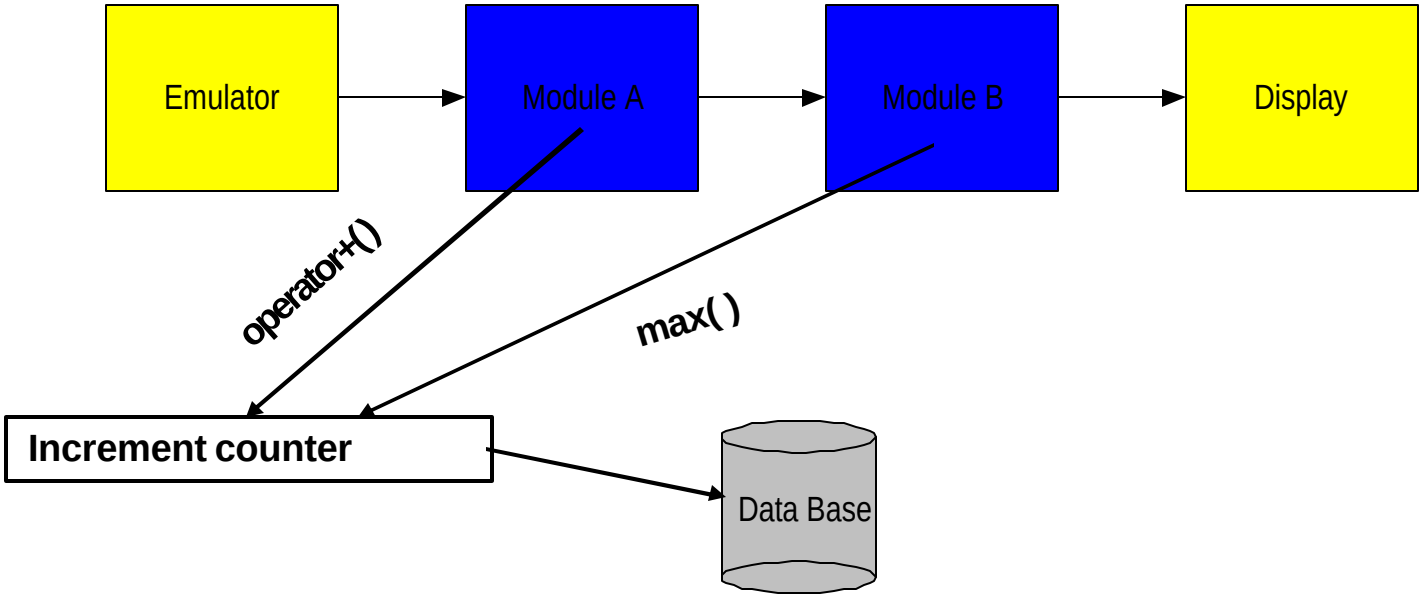
- Tool has to cope with different module refinements at each abstraction levels
  - *Communication channel*:



  - Same implementation principles throughout the design levels: "*Record Events*"

Implementation with global DB

1st Version

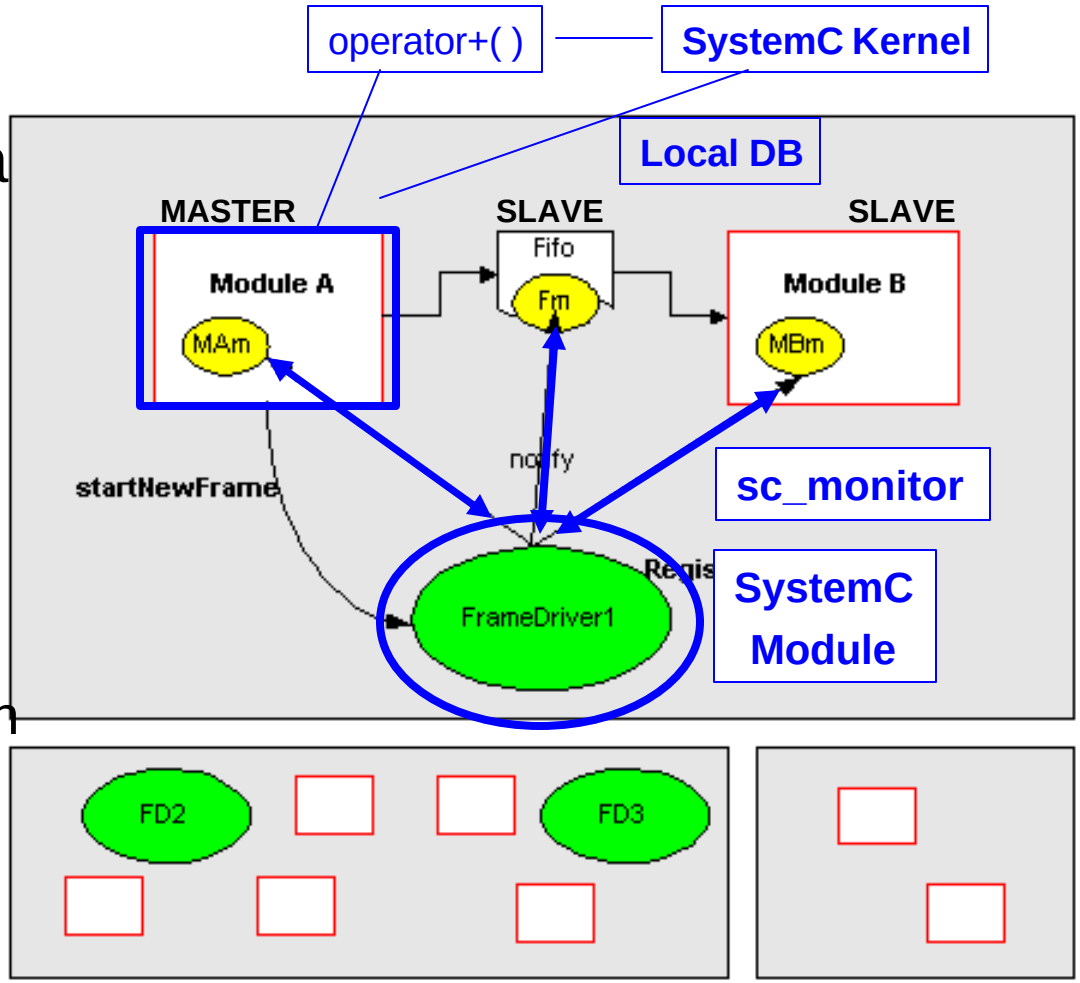


- Limited cost in simulation time
- No framing control (i.e., no statistical analysis)
- Global records (i.e., no Module specific information)
- Good for initial ‘compare & select phase’ of algorithms at Level 1

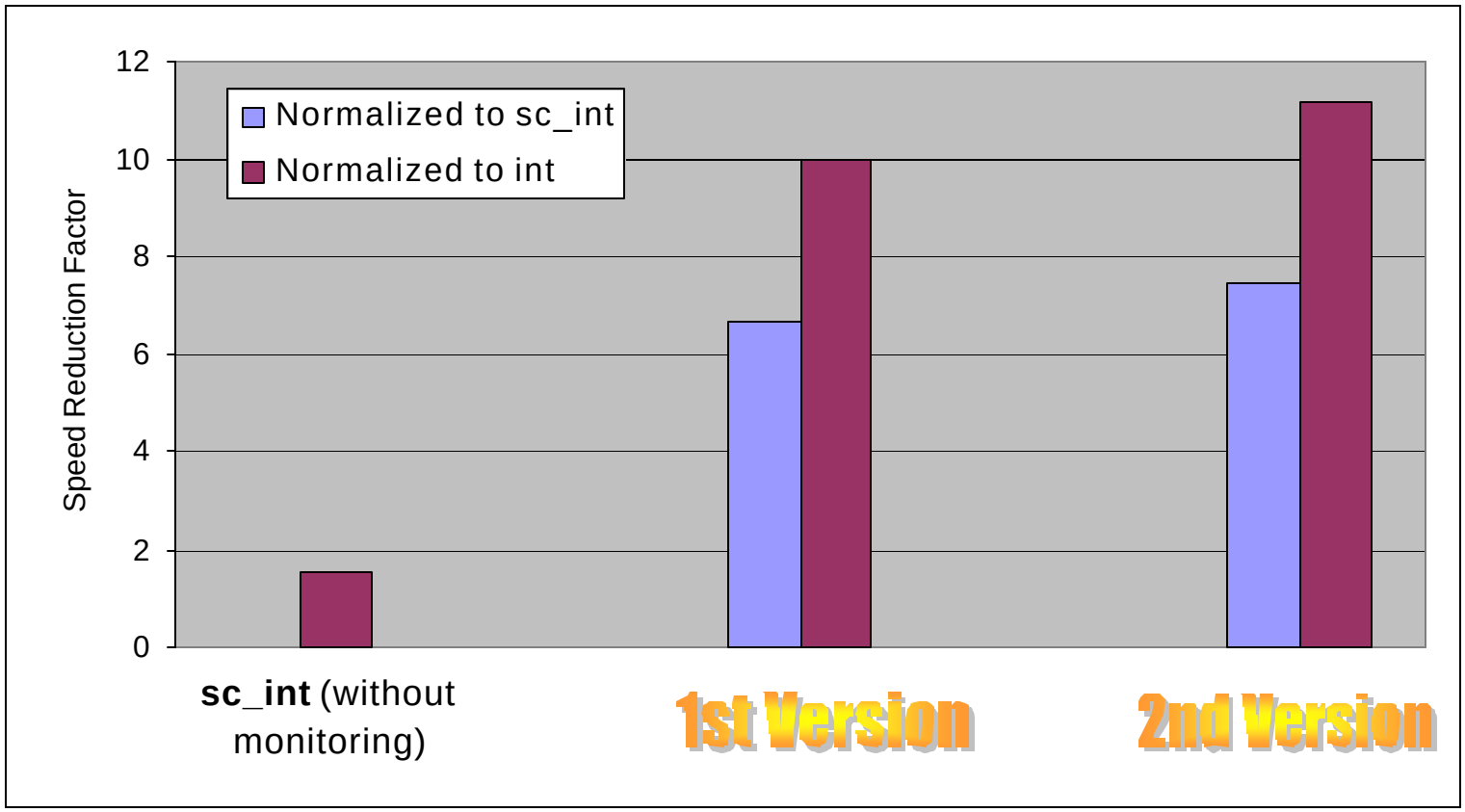
2nd Version

Frame monitoring

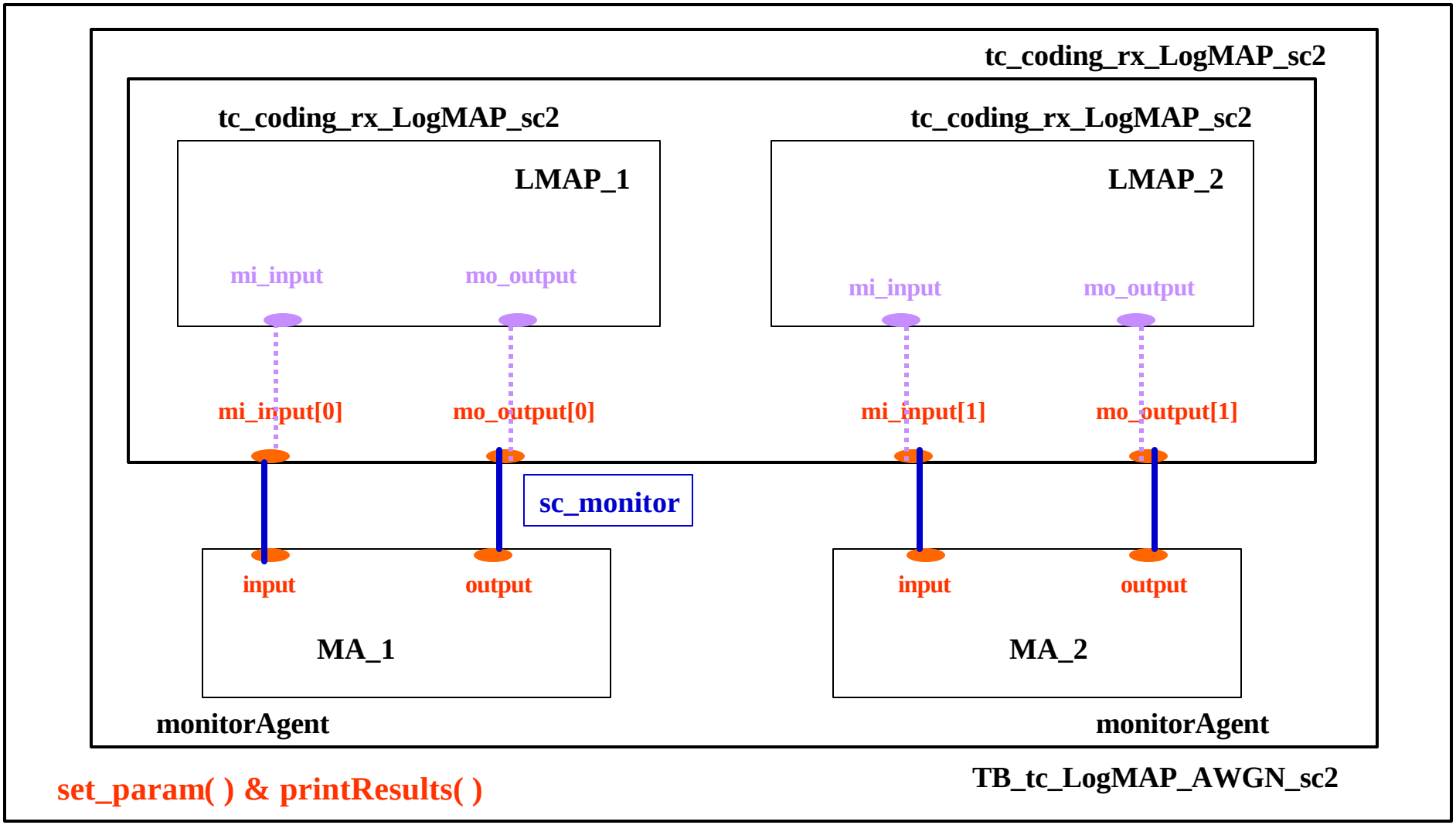
- Perform monitoring on a per Frame basis
- A frame could be :
  - Time (1 cycle, 4ns)
  - Data (8 bits)
  - Functional (loop iteration)
  - Parameters (e.g., whenever the TTI length changes)
  - etc.
- Support multiple frame “rates” and multiple frame drivers



Cost in Simulation Time



Example: Log-MAP (1)



main_AWGN

Example: Log-MAP (2)

```
void tc_LogMAP_sc2::main() {

   /* REGISTRATION
    * 1) Setup register protocol with various options
    * 2) Send the registration request
    * 3) Wait for registration acknowledgement from monitor agent */
   register_protocol rp = {sc_object()::name(), MASTER};
   mo_output -> registerModule(rp);
   wait( mi_input -> ackEvent() );
[...]

   // External function call
   tc_LogMap_sc2_funct(CBlk_Len_tmp, depth_tmp, z_buf, data_buf, … );
[...]

   /* FRAME INTERVAL CONTROL
    * 1) Send local DB and frame request protocol to monitor agent
    * 2) Reset local DB
    * 3) Wait for ack from monitor agent to resume processing */
   mo_output -> sendNewFrameRequest(rp,mv);
   resetLocalMonitorDB();
   wait( mi_input -> ackEvent() );
};
```

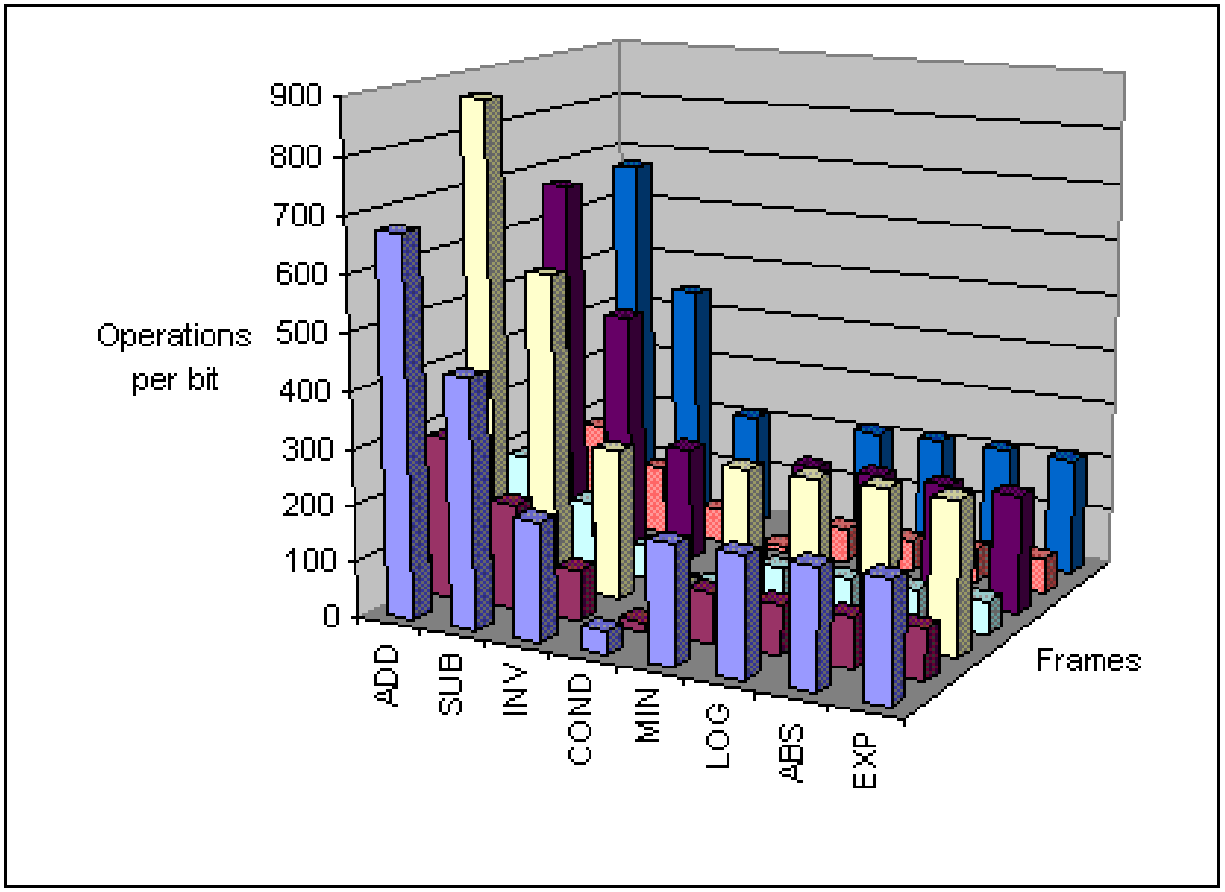
Example: Log-MAP (3)

```
void tc_LogMap_sc2_funct(CBlk_Len_tmp, depth_tmp, z_buf, data_buf, … )
{
   // Operations to be monitored
   wmops::WMOPS_SC_INT<20> alpha_sum;
   wmops::WMOPS_SC_INT<20> beta_sum;
   wmops::WMOPS_SC_INT<20> llr_comp;

[...]

   // Memory Size and Accesses to be monitored
   Alpha_type *ALPHA_MEMORY;
   Beta_type *BETA_MEMORY;

[...]

};
```

Monitoring

```
// Pools.h file
extern MMemPool pool_ALPHA_METRIC
typedef mmem::Array< sc_int<20>, 2, MMemAlloc< sc_int<20>, pool_ALPHA_METRIC> >
        Alpha_type;
[...]
```

Example: Log-MAP (4)
-
Computational Complexity Statistics of Turbo decoders implementing Stopping Criteria



Conclusion

- The ***wmops*** and ***MMPool*** libraries are used in several ongoing projects to verify and automate the ***complexity analysis*** early on in the design phase (of wireless systems)

- Prototype evolution
  - … on a as need basis

- 1st Prototype is used in MATRICE (European project)
  - http://www.ist-matrice.org