

# System Simulation with gem5, SystemC and other Tools

Christian Menard (TU Dresden, Germany)

Matthias Jung (Fraunhofer IESE, Germany)

# Detailed Description of Contents for this Talk

## System Simulation with gem5 and SystemC

The Keystone for Full Interoperability

Christian Menard, Jeronimo Castrillon  
Technische Universität Dresden  
Dresden, Germany  
Email: christian.menard@tu-dresden.de  
jeronimo.castrillon@tu-dresden.de

Matthias Jung  
Fraunhofer IESE  
Kaiserslautern, Germany  
Email: matthias.jung@iese.fraunhofer.de

Norbert Wehn  
University of Kaiserslautern  
Kaiserslautern, Germany  
Email: wehn@cit.uni-kl.de

**Abstract**—SystemC TLM based virtual prototypes have become the main tool in industry and research for concurrent hardware and software development, as well as hardware design space exploration. However, there exists a lack of accurate, free, changeable and realistic SystemC models of modern CPUs. Therefore, many researchers use the cycle accurate open source system simulator gem5, which has been developed in parallel to the SystemC standard. In this paper we present a coupling of gem5 with SystemC that offers full interoperability between both simulation frameworks, and therefore enables a huge set of possibilities for system level design space exploration. Furthermore, we show that the coupling itself only induces a relatively small overhead to the total execution time of the simulation.

### I. INTRODUCTION

Today's companies have to deal with complex hardware architectures such as multi-cores, sophisticated interconnects and memory systems. *Virtual Prototypes* (VPs) are widely used to allow for early design space exploration and to decrease the *Time-to-Market* (TTM), costs, and efforts by developing software and hardware concurrently. They are high-speed, fully functional software models of physical hardware systems that can simulate the exact behavior of real hardware. With their help, complete *Multi-Processor System-on-Chips* (MPSoCs) can be simulated with reasonable simulation speed and visibility and controllability over the entire system. Case studies have shown that by employing VPs it is possible to deliver more competitive products up to six months earlier [1].

In recent years, the SystemC TLM2.0 IEEE1666 standard [2] has become the main developing tool for VPs in industry and research. It allows to quickly simulate HW and SW systems on different levels of abstraction in order to estimate and optimize the performance and power for different applications. In contrast to pin accurate models, *Transaction Level Modeling* (TLM) abstracts the communication mechanisms from the actual hardware. It encapsulates communication between interacting components in so-called transactions, which are transferred by function calls.

The industry offers several SystemC based CPU core models that provide a trade-off between simulation speed and accuracy. For instance, the *FastModels*, distributed by ARM Ltd. or the OVP [3] models use *just-in-time-compilation* for code execution and model communication using the TLM *loosely-timed* coding style. However, loosely-timed models do not reflect a realistic timing behavior and thus can mainly be used for software development and high level explorations. Cycle

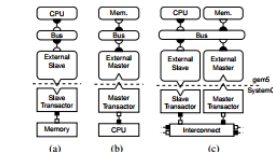


Figure 1: Possible scenarios for binding gem5 and SystemC.

accurate simulations can be performed with the commercially available Carbon models from ARM or the Aurix [4] TLM models from Infineon. However, these models are shipped as binary libraries, which makes them useless for micro-architectural research due to their inflexibility (i.e. they cannot be modified). Furthermore, they are slow and can, therefore, not be employed for fast design space exploration.

In contrast to industry, the academia lacks free, accurate and realistic SystemC models of modern CPUs. The most mature, cycle accurate open source system simulator is the gem5 framework, which is a modular platform for computer-system architecture research [5]. It is not only widely used in academia, but also the industry employs gem5 for research. For instance, ARM and AMD use gem5 internally for design space exploration and actively contribute to the open source project. However, gem5 is not implemented in SystemC as its development started before the IEEE ratified the official SystemC and TLM standard in 2005 [2]. Since this time, both frameworks, gem5 and SystemC have evolved extensively and independent in parallel. Therefore, gem5 is incompatible to TLM models that exist in industry and academia.

In this paper, we present for the first time a comprehensive coupling between SystemC and gem5 that provides full interoperability. Through this coupling, any SystemC module that implements the TLM base protocol can be connected to any gem5 module, as shown in Figure 1. To the best of our knowledge, there exists no reference which describes syntax and semantics of both frameworks and how both simulation kernels can be coupled in order to enable full interoperability.

```
Terminal - README + (~/.Programming/gem5/util/tlm) - VIM
File Edit View Terminal Tabs Help

This directory contains a demo of a coupling between gem5 and SystemC-TLM. It
is based on the gem5-systemc implementation in utils/systemc. This Readme gives
an overall overview (I), describes the source files in this directory (II),
explains the build steps (III), shows how to run example simulations (IV-VI)
and lists known issues (VII).

I. Overview
=====

The sources in this directory provide three SystemC modules that manage the
SystemC/gem5 co-simulation: Gem5SimControl, Gem5MasterTransactor, and
Gem5SlaveTransactor. They also implement gem5's ExternalMaster::Port interface
(SCMasterPort) and ExternalSlave::Port interface (SCSlavePort).

**SCMasterPort** and **Gem5MasterTransactor** together form a TLM-to-gem5
bridge. SCMasterPort implements gem5's ExternalMaster::Port interface and forms
the gem5 end of the bridge. Gem5MasterTransactor is a SystemC module that
provides a target socket and represents the TLM side of the bridge. All TLM
requests sent to this target socket, are translated to gem5 requests and
forwarded to the gem5 world through the SCMasterPort. Then the gem5 world
handles the request and eventually issues a response. When the response arrives
at the SCMasterPort it gets translated back into a TLM response and forwarded
to the TLM world through target socket of the Gem5MasterTransactor.
SCMasterPort and Gem5MasterTransactor are bound to each other by configuring
them for the same port name.

**SCSlavePort** and **Gem5SlaveTransactor** together form a gem5-to-TLM bridge.
Gem5SlaveTransactor is a SystemC module that provides an initiator socket and
represents the TLM end of the bridge. SCSlavePort implements gem5's
ExternalSlave::Port interface and forms the gem5 side of the bridge. All gem5
requests sent to the SCSlavePort, are translated to TLM requests and forwarded
to the TLM world through the initiator socket of the Gem5SlaveTransactor. Then
the TLM world handles the request and eventually issues a response. When the
response arrives at the Gem5SlaveTransactor it gets translated back into a
gem5 response and forwarded to the gem5 world through the SCSlavePort. SCSlavePort
and Gem5SlaveTransactor are bound to each other by configuring them for the
same port name.

6,0-1 Top
```

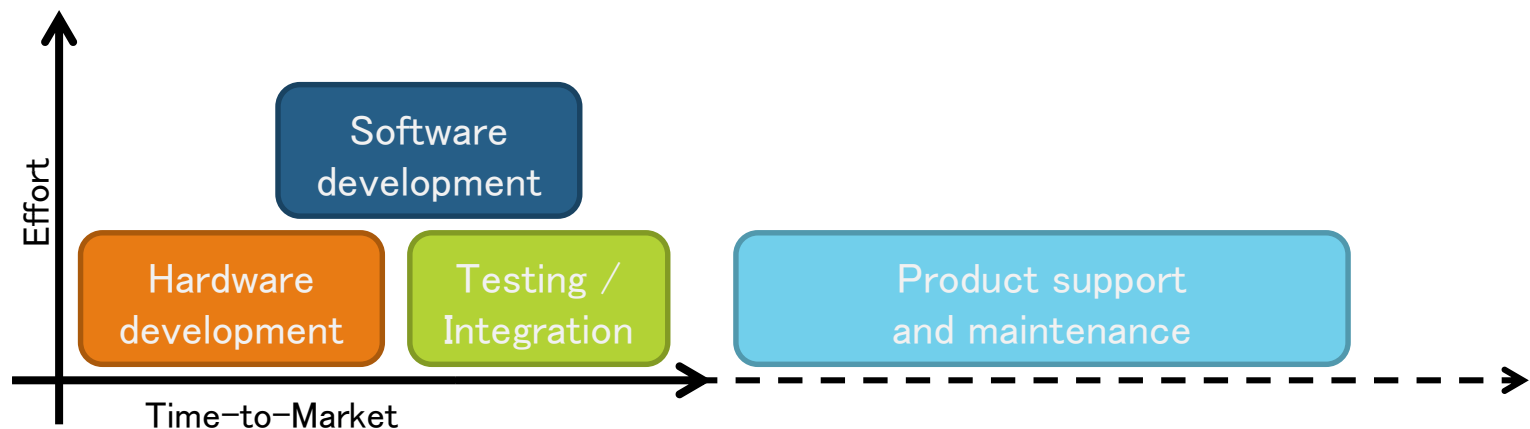
Paper: [Link](#) [1]

Sources: [/gem5/utils/tlm/README](#)

# Virtual Prototyping

Functional software models of physical hardware:

- ❑ Visibility and controllability over the entire system
- ❑ Powerful debugging and analysis tools
- ❑ Reuse of components for future projects
- ❑ Fast Design Space Exploration (for HW engineers)
- ❑ Easy to exchange, worldwide
- ❑ Concurrent HW and SW development:

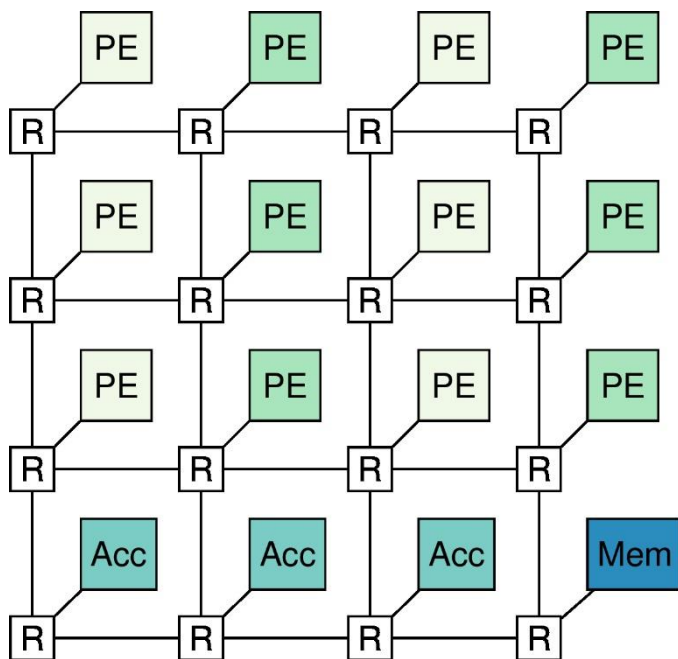


Functional software models of physical hardware:

- ❑ Visibility and controllability over the entire system
- ❑ Powerful debugging and analysis tools
- ❑ Reuse of components for future projects
- ❑ Fast Design Space Exploration (for HW engineers)
- ❑ Easy to exchange, worldwide
- ❑ Concurrent HW and SW development:

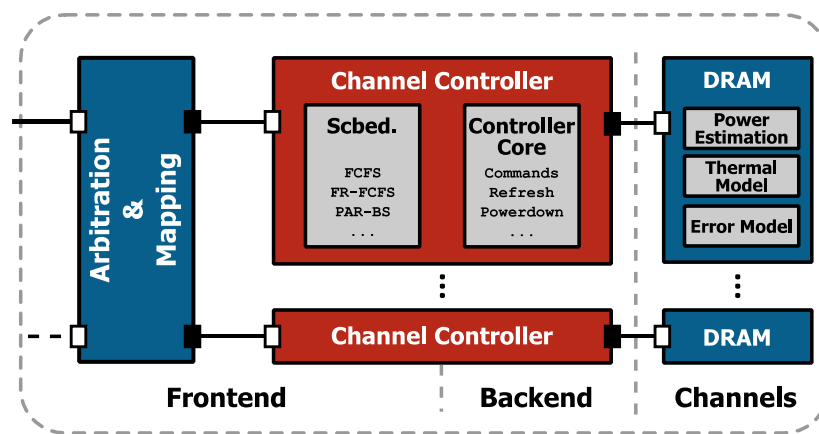


## Simulation of (widely) heterogeneous systems

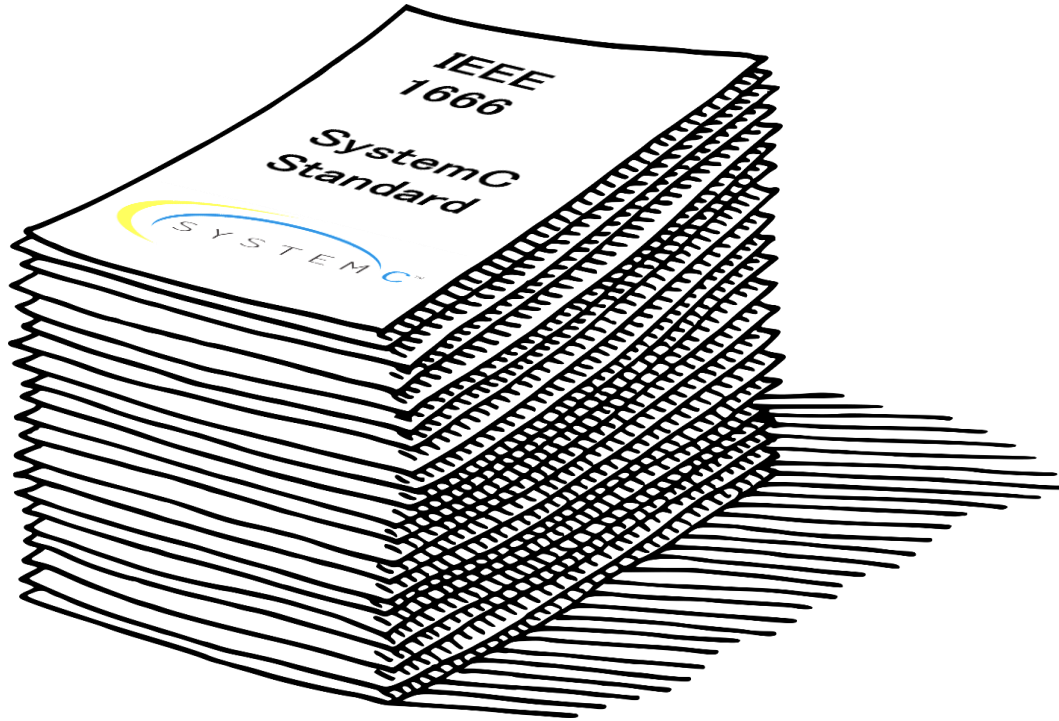


- Many different models of cores, accelerators, and communication infrastructure required.

## Simulation of the Memory Subsystem



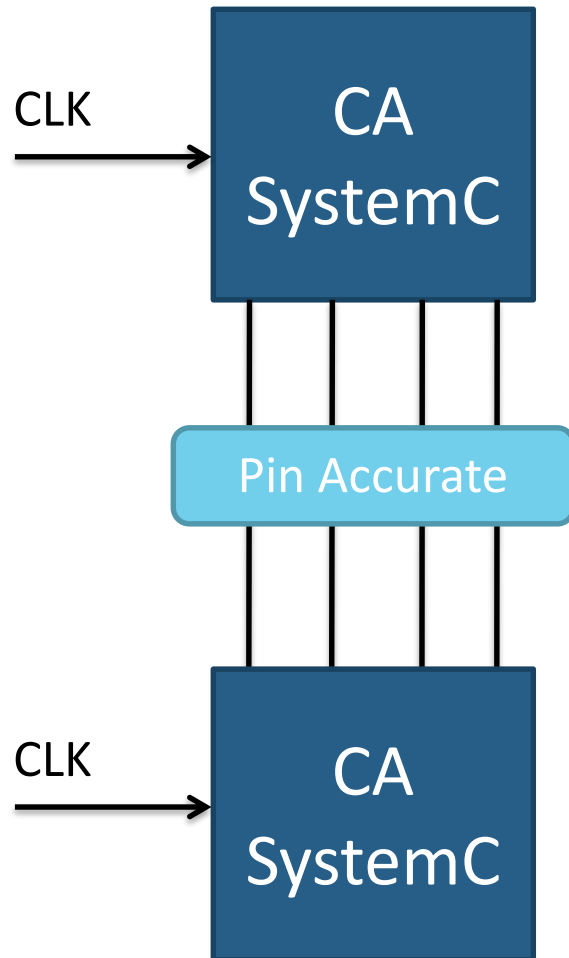
- Focuses on the memory subsystem, but detailed simulation of realistic workloads is required



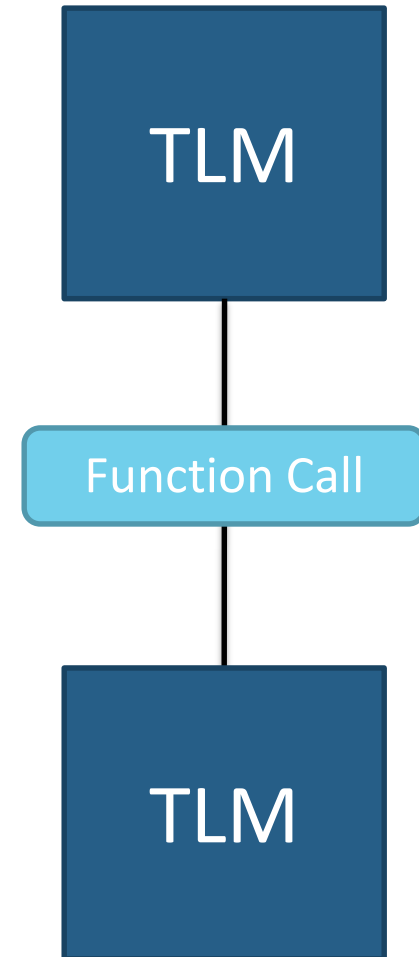
- ❑ Modeling language for HW and SW components
- ❑ Extends C++ to an event-driven simulation kernel
- ❑ Various levels of accuracy
- ❑ IEEE Standard, Maintained by Accellera
- ❑ 10-100x Faster than CA VHDL/Verilog Simulation

→ However, normal CA SystemC is not fast enough to, e.g., boot an OS.

# Transaction Level Modeling (TLM)



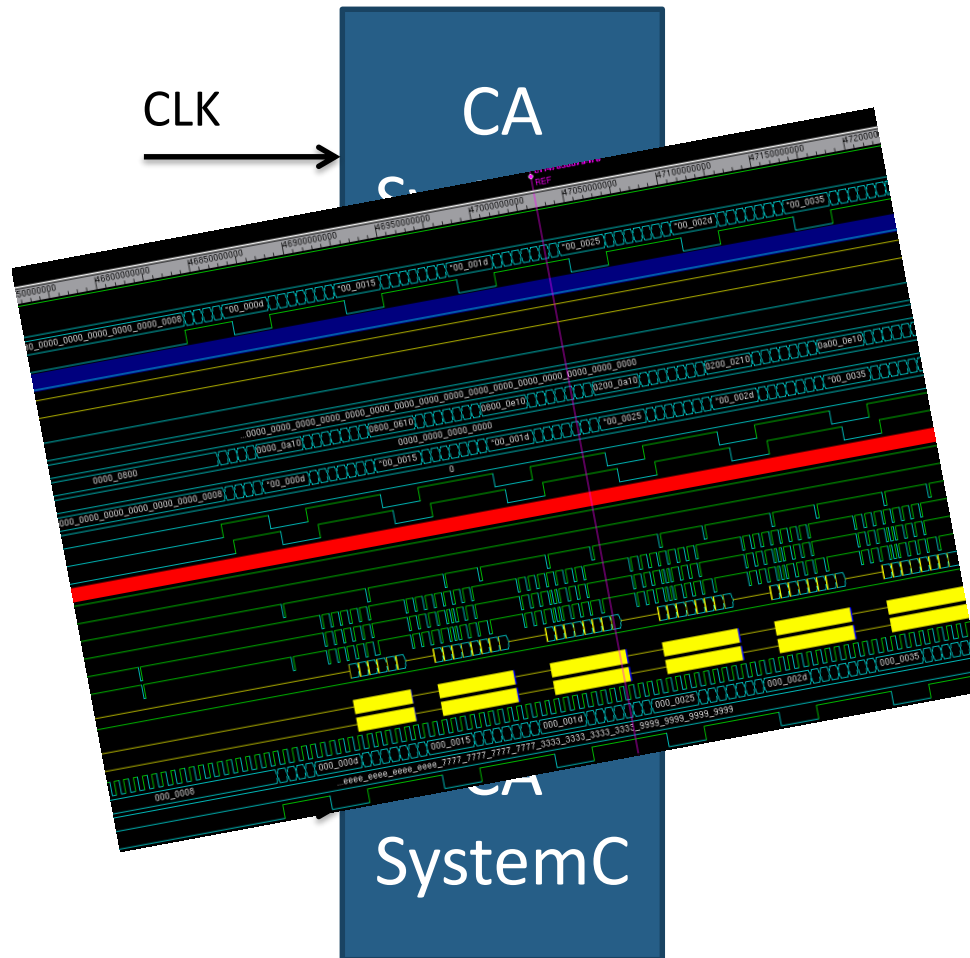
Simulate each pin  
separately



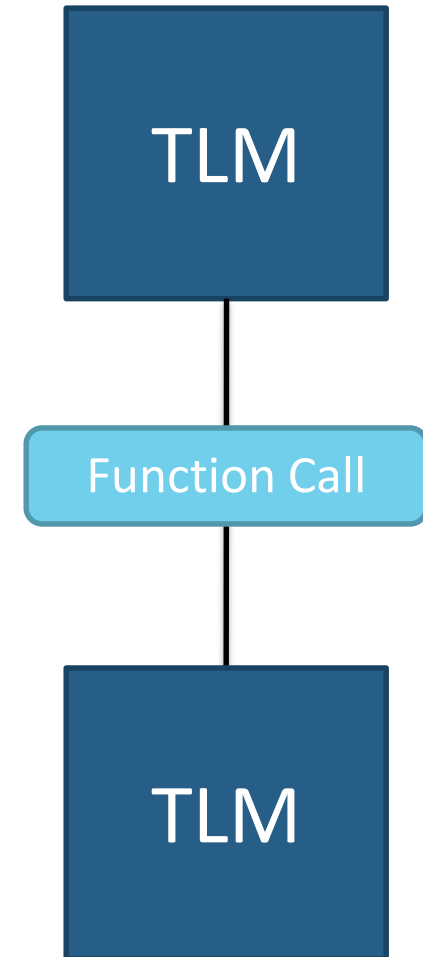
Simulate transactions  
up to 10,000x Faster



# Transaction Level Modeling (TLM)

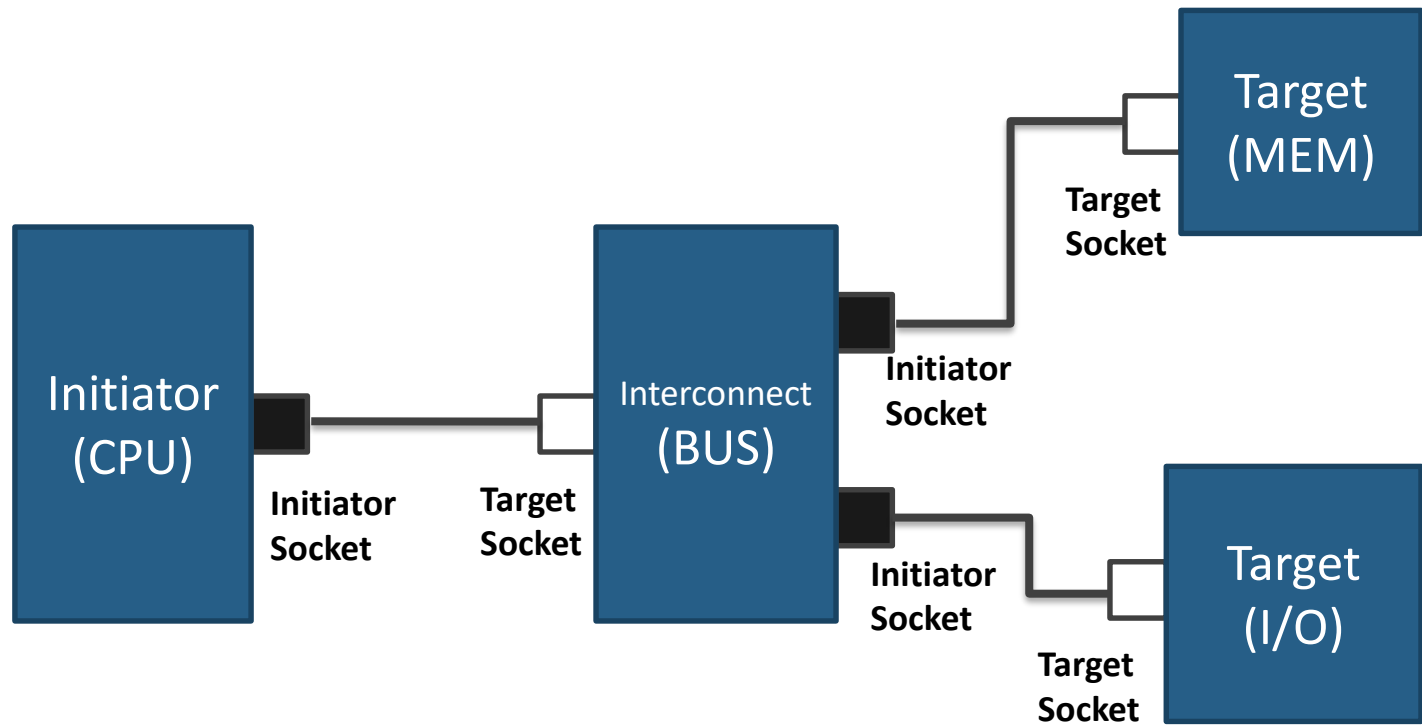


Simulate each pin  
separately



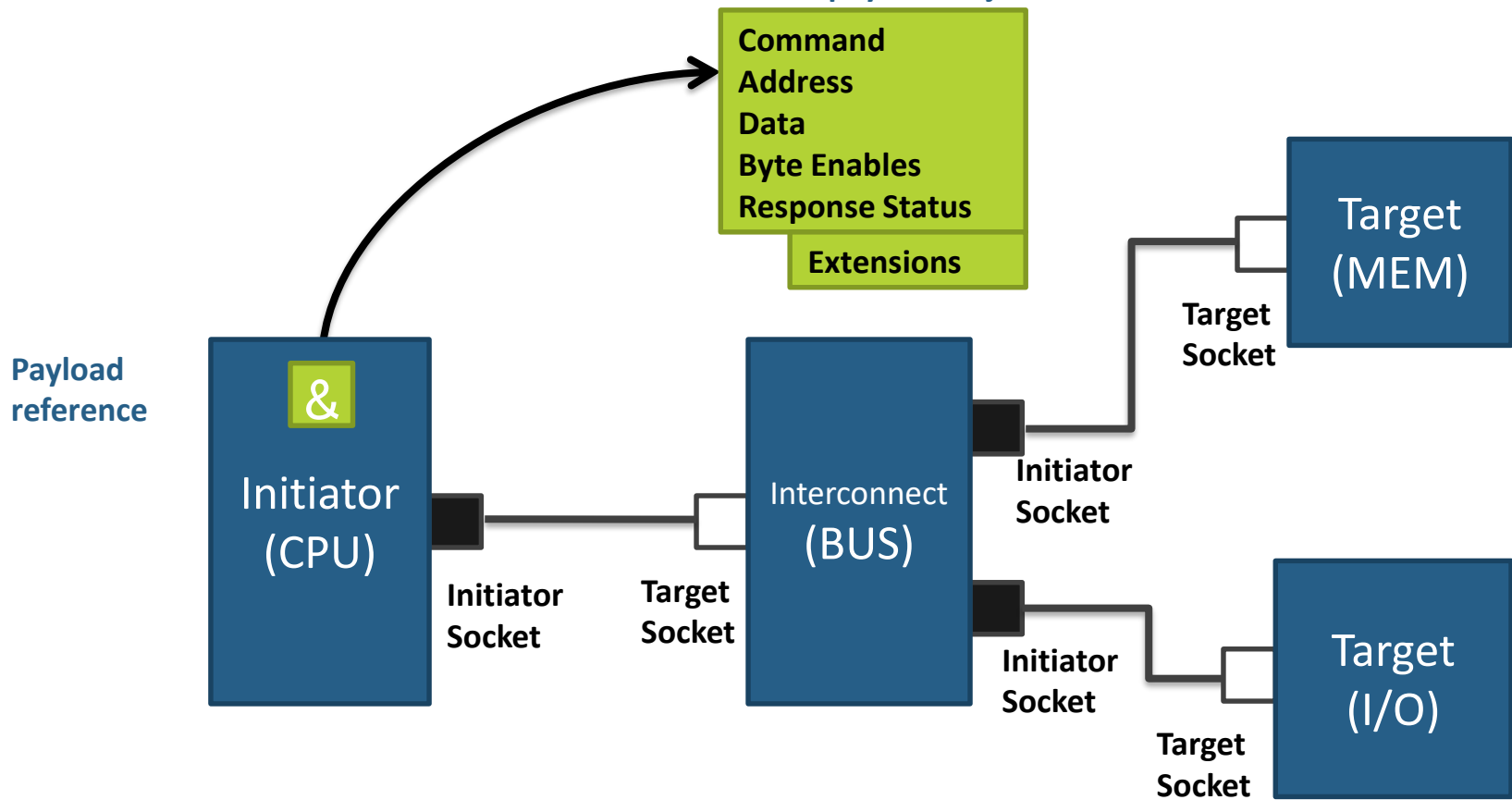
Simulate transactions  
up to 10,000x Faster

# Generic Payload



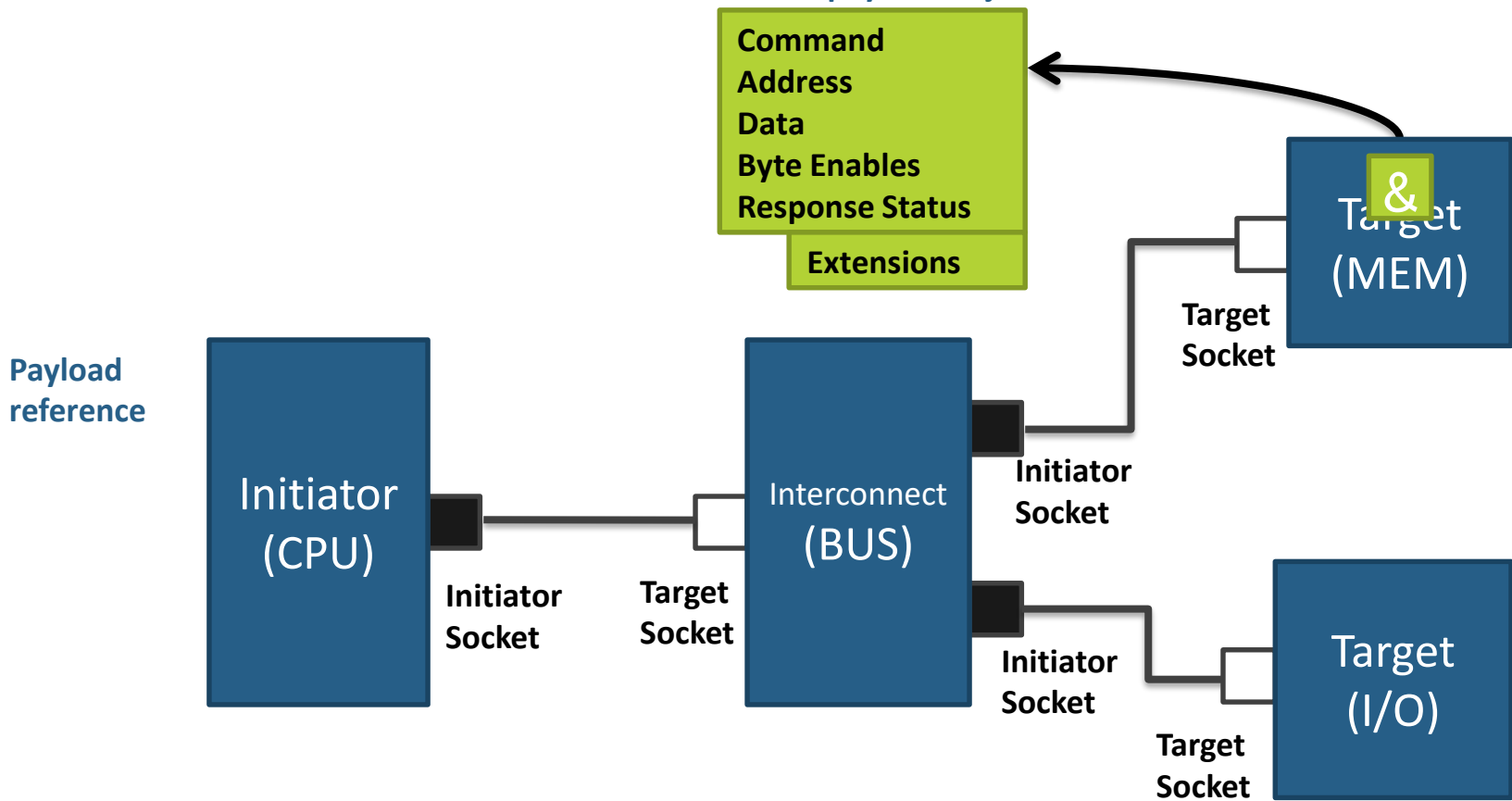
# Generic Payload

## Generic payload object



# Generic Payload

## Generic payload object



## TLM Use Cases

SW Application  
Development

SW Performance  
Analysis

Architecture  
Analysis

Hardware  
Verification

## TLM 2.0 Coding Style (*Just Guidelines*)

Loosely-timed

Single-phase, blocking API  
`debug_transport`, `b_transport`

`nb_transport`

Multi-phase, non-blocking API

Approximately -timed

## TLM Mechanisms (*Definitive API for enabling Interoperability*)

Blocking  
transport

DMI

Quantum

Sockets

Generic  
payload

Extensions

Phases

Non-  
blocking  
transport

TLM is widely used in Industry:

- ❑ The market of virtual platform tools:
  - ❑ Synopsys - Platform Architect
  - ❑ Cadence - Virtual System Platform
  - ❑ Mentor Graphics - Vista Virtual prototyping
  - ❑ Imperas - OpenVP
  - ❑ ASTC - VLAB Works
  
- ❑ Virtual Platform Core Models:
  - ❑ ARM (Fastmodels):
    - only LT models based on JIT, non-free, library
  - ❑ ARM Carbon (Former Carbon Design Systems):
    - Cycle Accurate (CA) Models in TLM Wrapper, non-free, library
  - ❑ Imperas / OVP:
    - only LT, Free



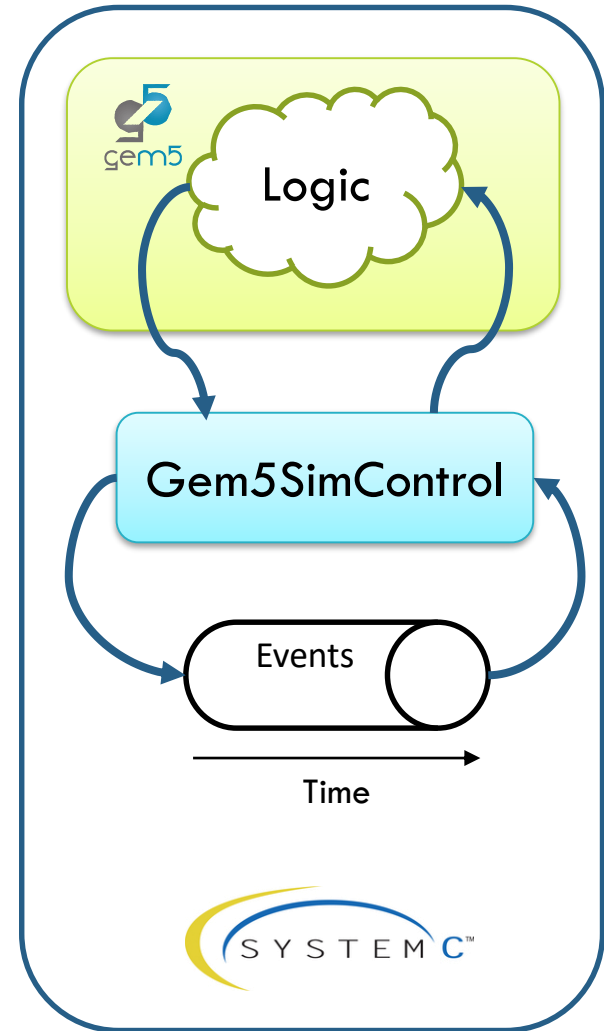
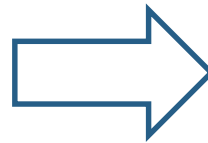
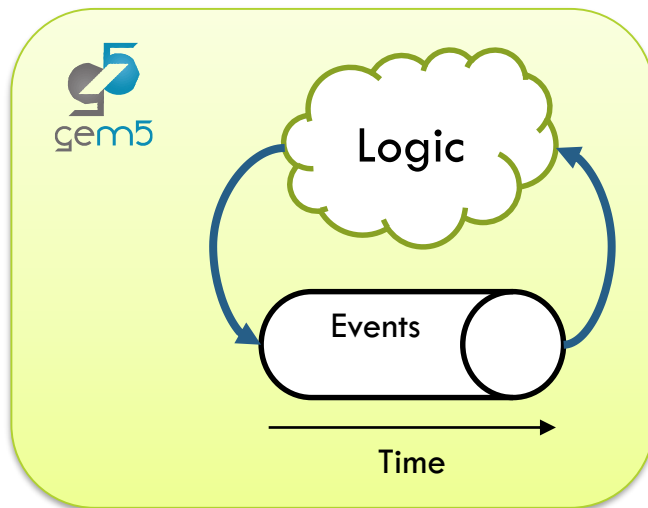
→ An accurate, free available and changeable core model is needed

# Coupling gem5 with SystemC

# Coupling gem5 with SystemC

gem5 supports a SystemC coupling:

- ❑ Gem5 is build as a C++ library.
- ❑ It is linked into a SystemC simulation.
- ❑ A SystemC object implements the gem5 event queue.
- ❑ **How can we communicate with other SystemC modules?**





## Timing

- ❑ The most detailed access: queuing delay + resource contention
- ❑ Similar to the TLM `nb_transport` interface.

## Atomic

- ❑ Accesses are a faster than detailed access
- ❑ Used for **fast forwarding** and **warming up caches**
- ❑ Similar to the TLM `b_transport` interface
- ❑ Not good for performance simulation

## Functional

- ❑ Similar to `transport_dbg` e.g. loading binaries, avoiding deadlocks in multi-level cache coherent networks

# Converting between TLM and gem5

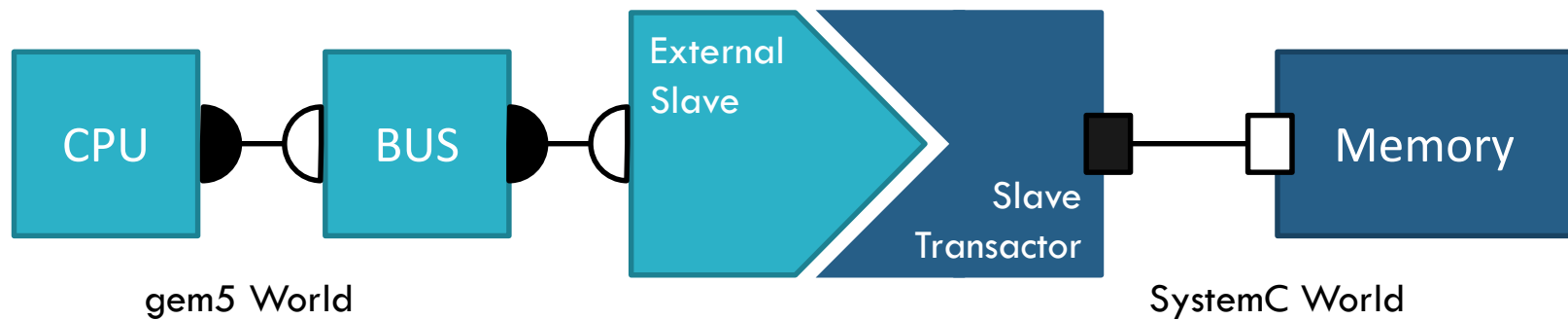


`recvFunctional(...)` → `transport_dbg(...)`  
`recvAtomic(...)` → `b_transport(...)`  
`recvTimingReq(...)` → `nb_transport(...)`

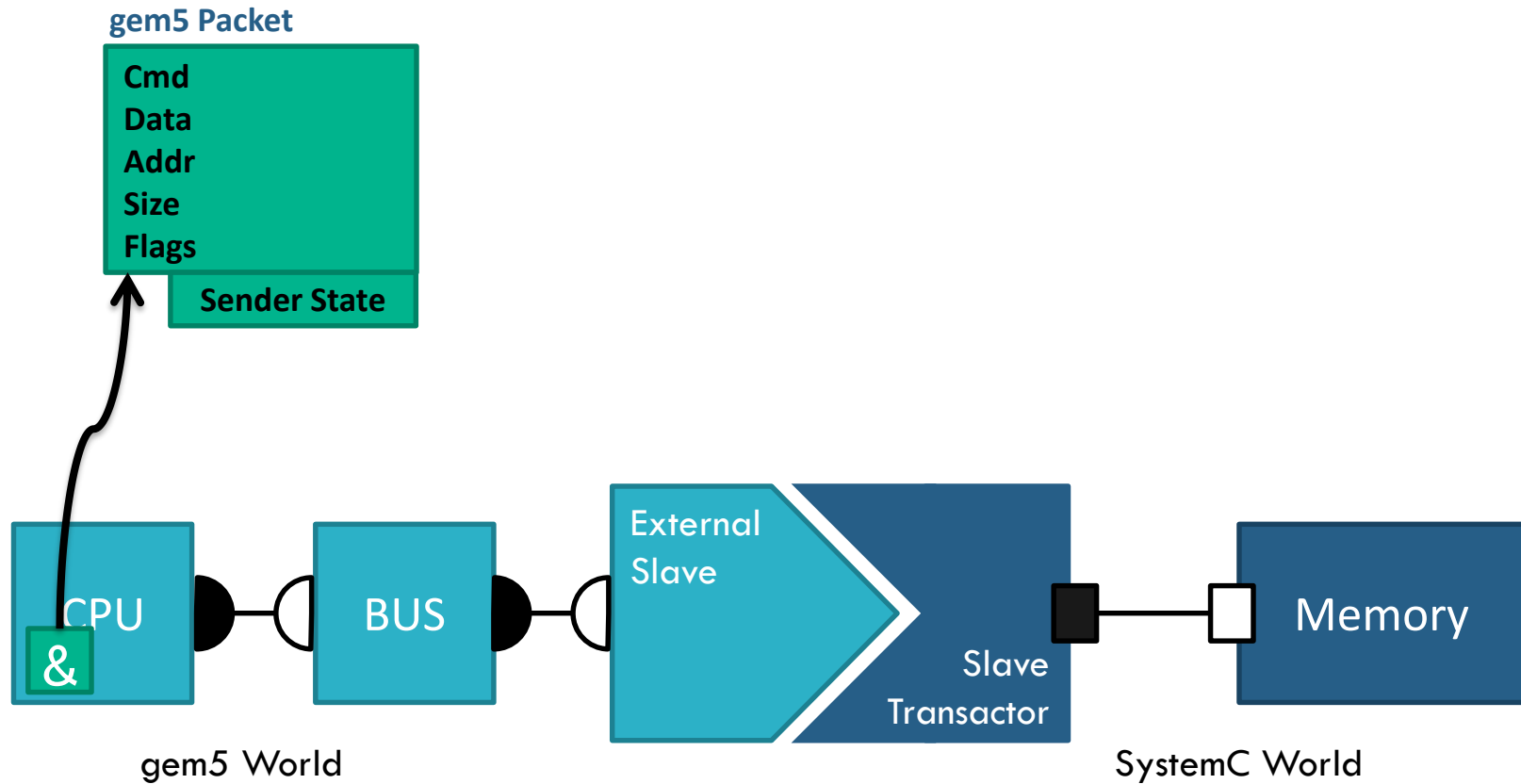


`transport_dbg(...)` → `recvFunctional(...)`  
`b_transport(...)` → `recvAtomic(...)`  
`nb_transport(...)` → `recvTimingReq(...)`

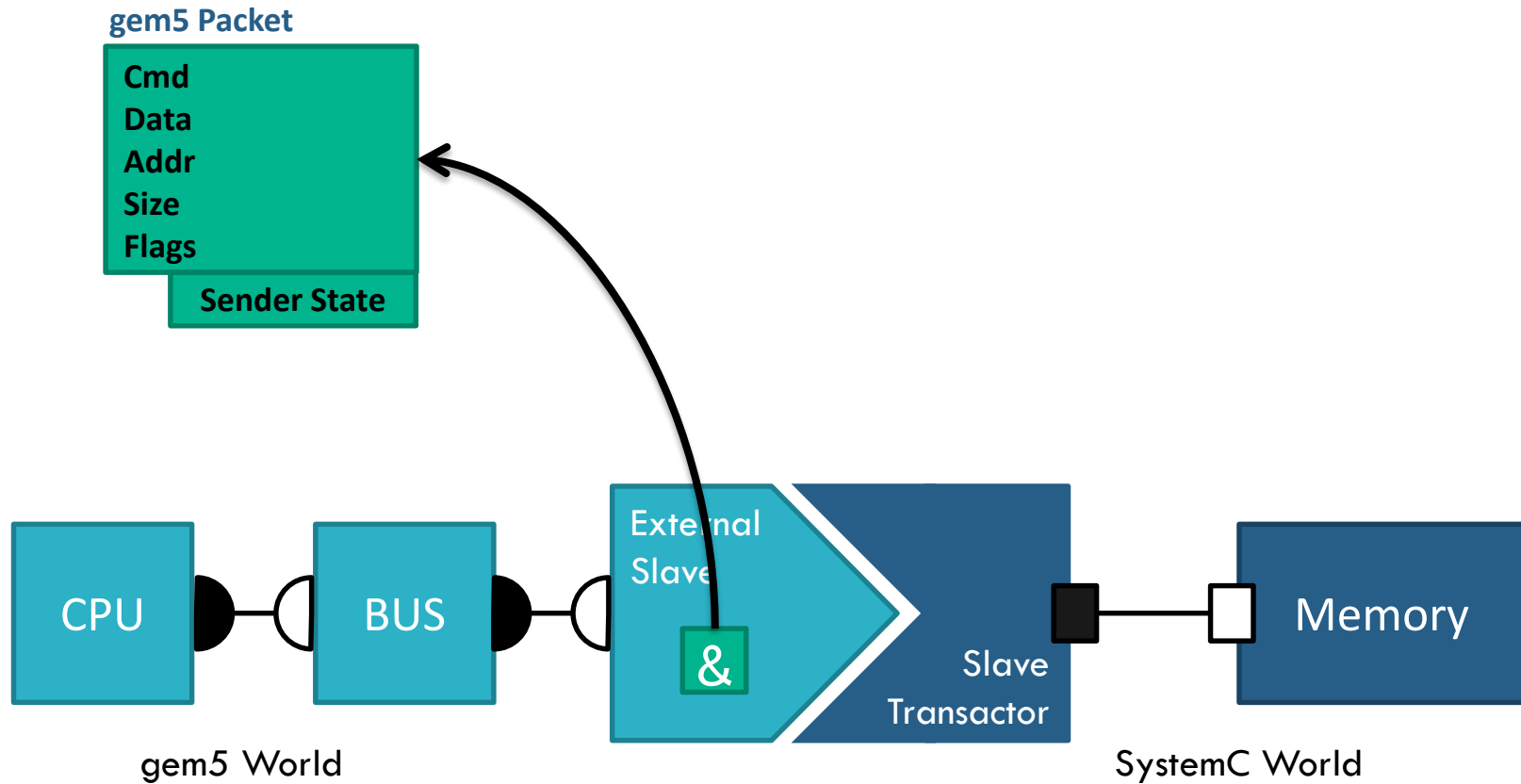
# Transaction Explained



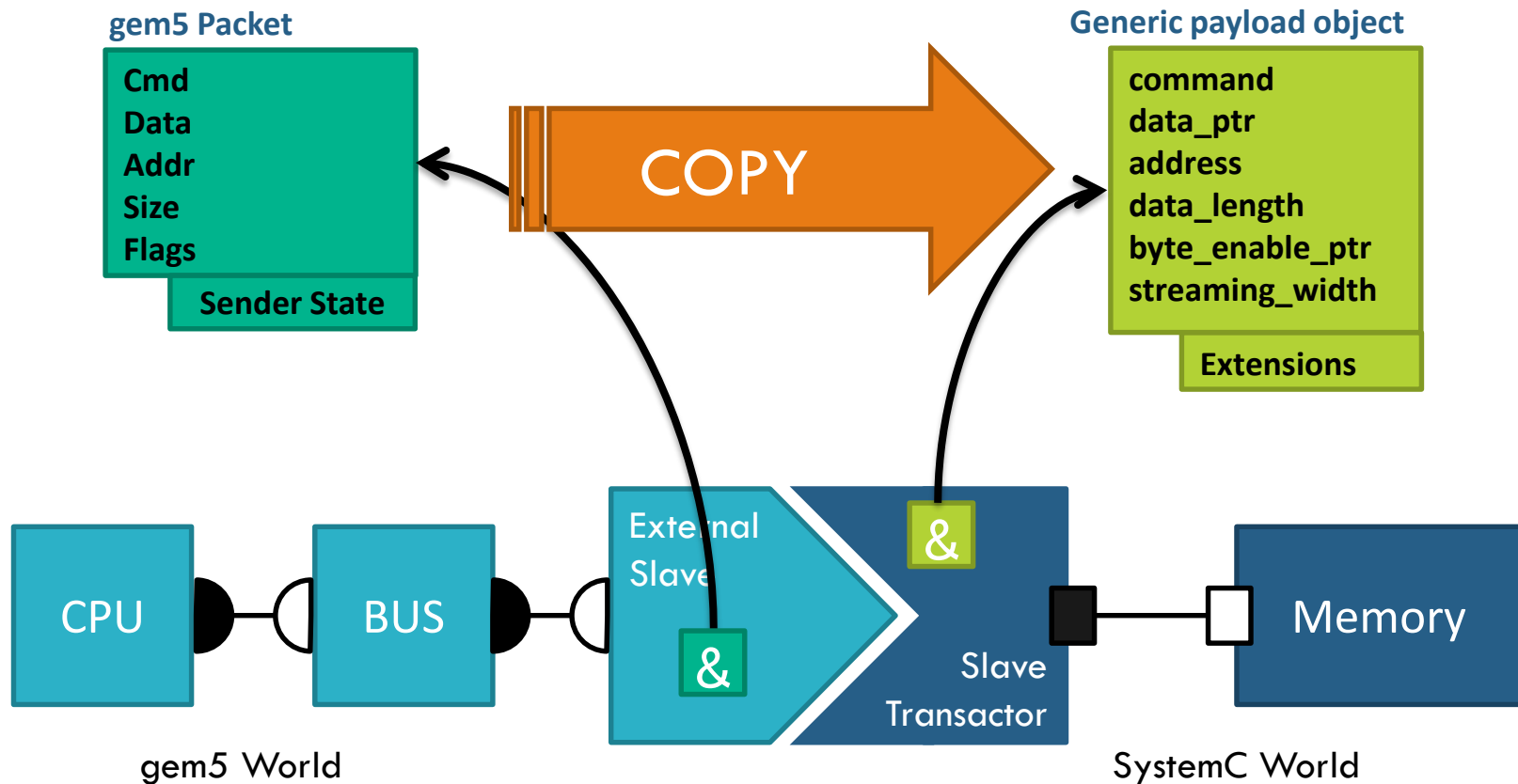
# Transaction Explained



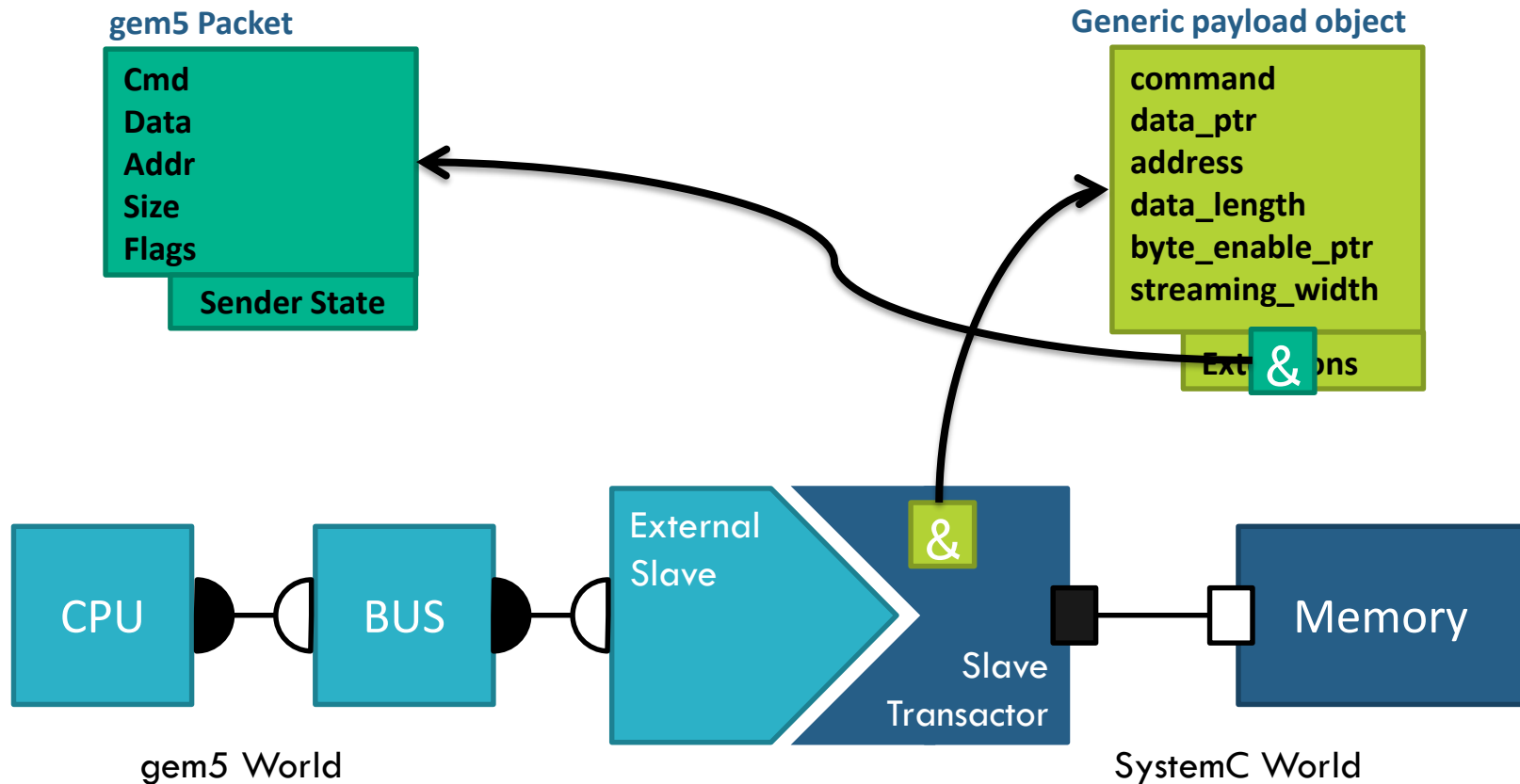
# Transaction Explained



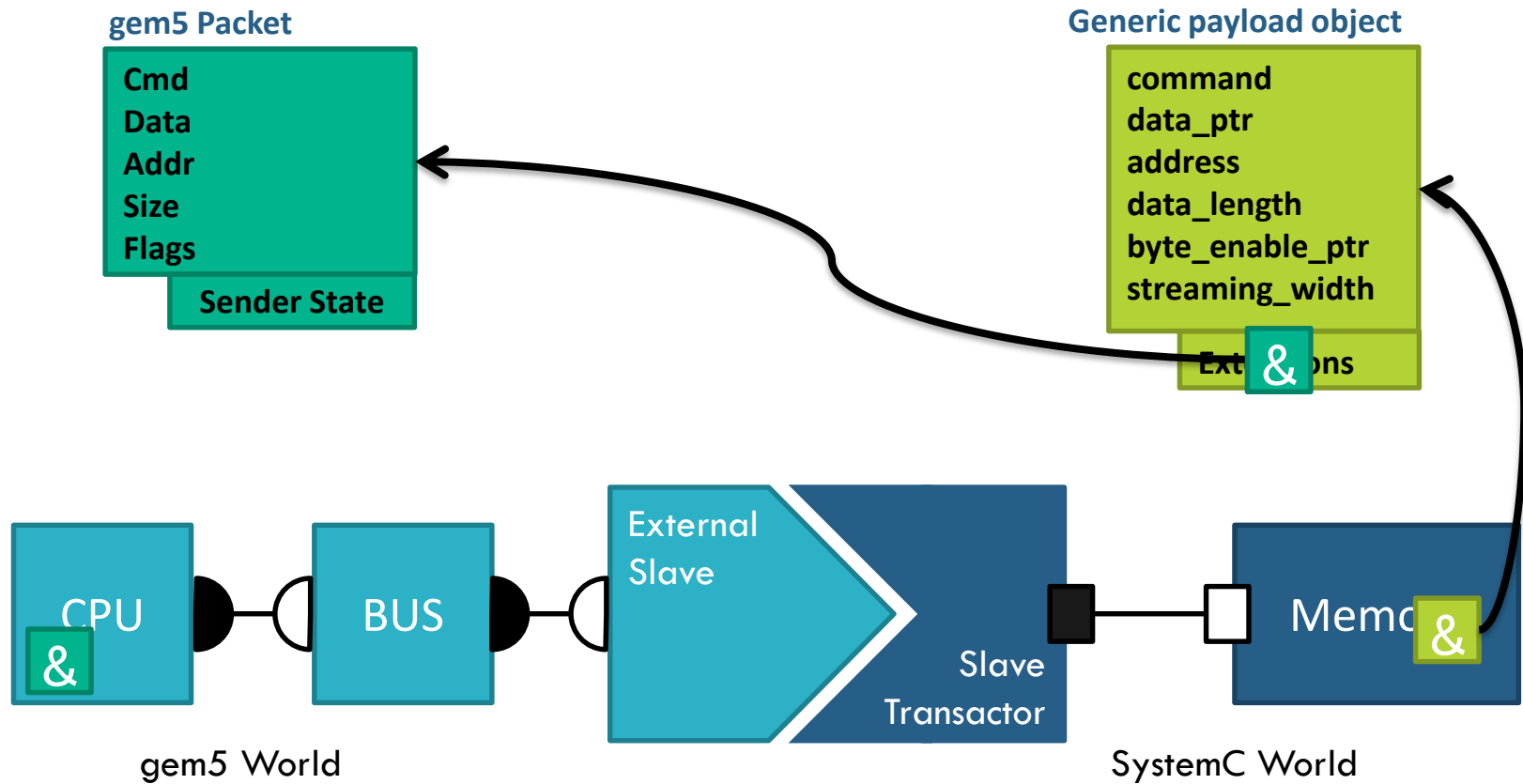
# Transaction Explained



# Transaction Explained

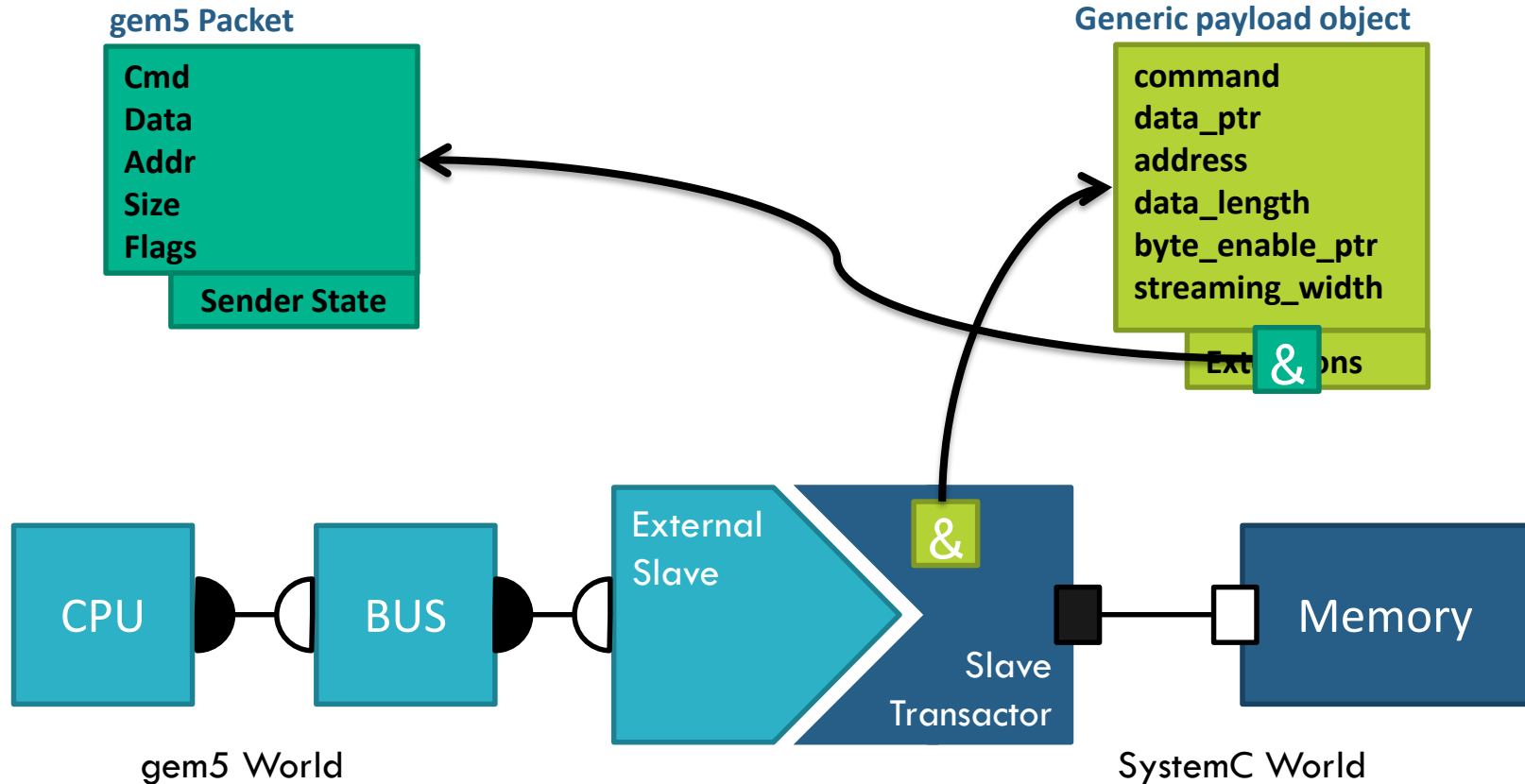


# Transaction Explained

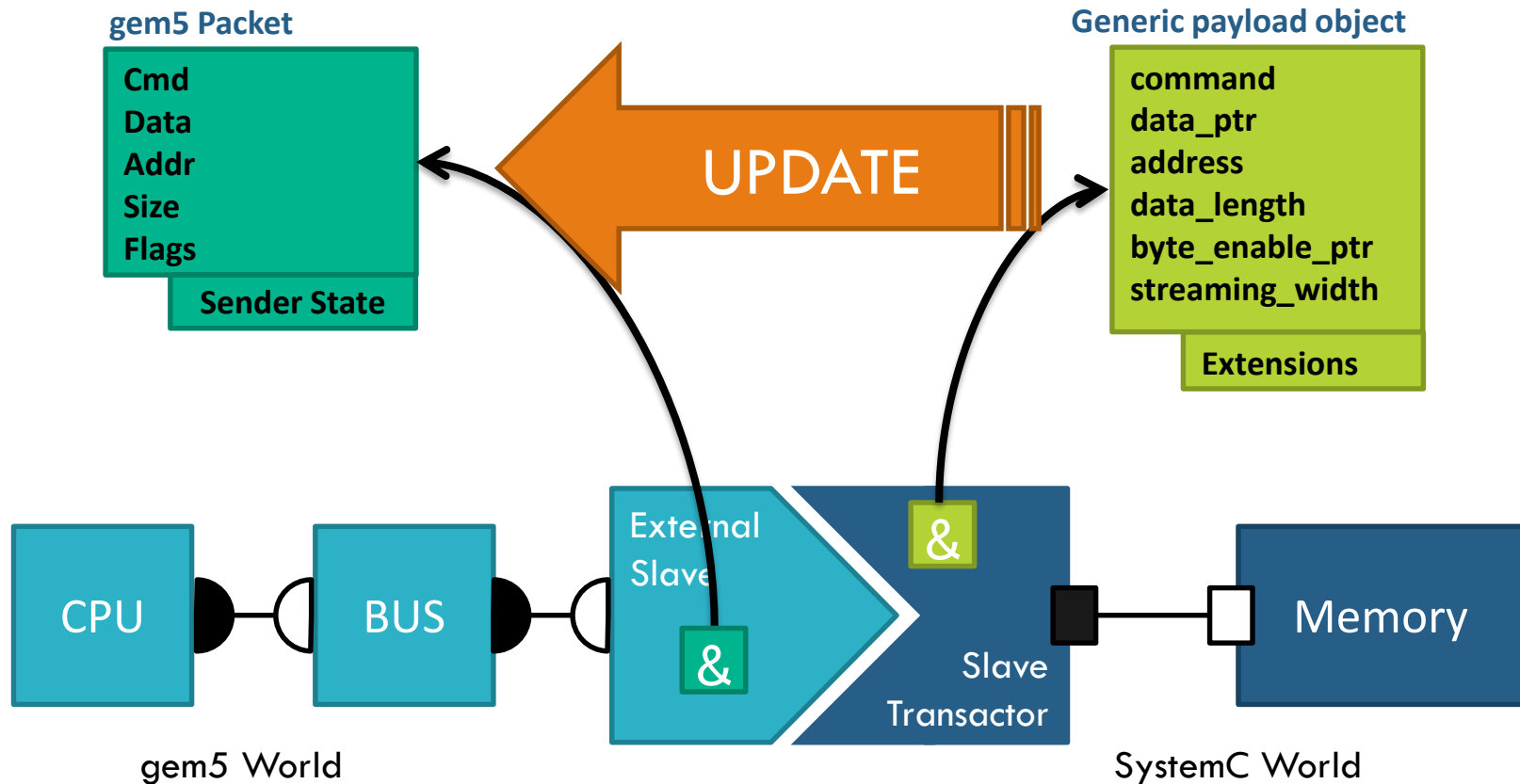




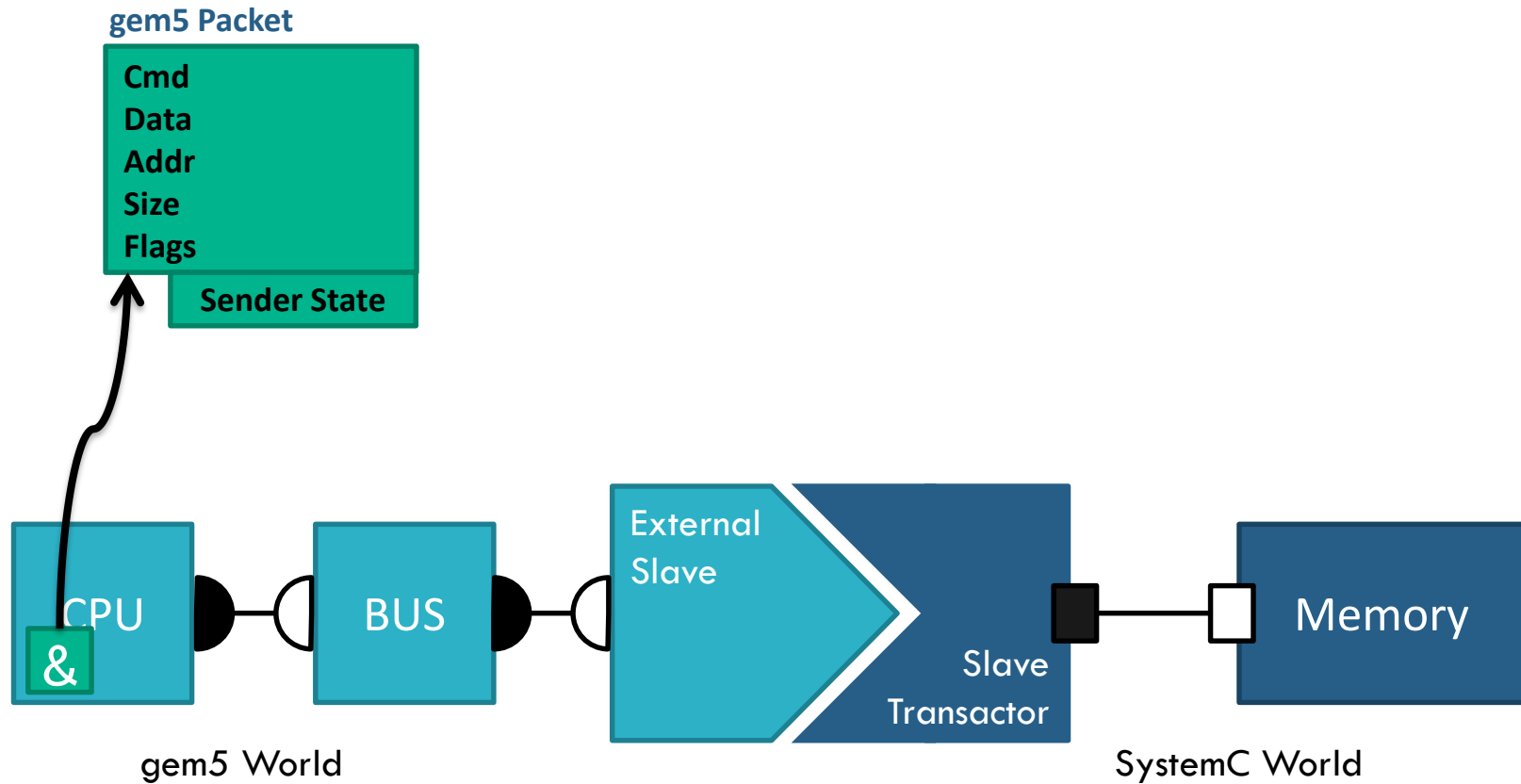
# Transaction Explained



# Transaction Explained



# Transaction Explained



# How to get Started?

# How to get Started?

## ❑ Study the Examples in </gem5/utils/tlm/>

### ❑ Slave Example:



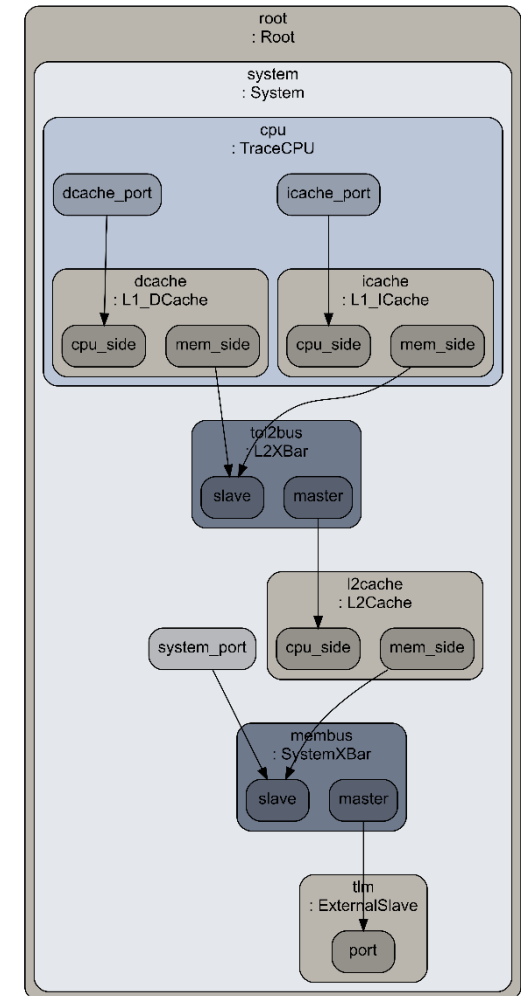
### ❑ Master Example:



### ❑ Elastic Trace Example [5] (see left)

### ❑ Full System Example:

```
../../../../build/ARM/gem5.opt ../../configs/example/fs.py \
--tlm-memory=transactor --cpu-type=TimingSimpleCPU --num-cpu=1 \
--mem-type=SimpleMemory --mem-size=512MB --mem-channels=1 --caches \
--l2cache --machine-type=VExpress_EMM \
--dtb-filename=vexpress.aarch32.11_20131205.0-gem5.1cpu.dtb \
--kernel=vmlinux.aarch32.11_20131205.0-gem5 \
--disk-image=linux-aarch32-ael.img \
```



1. Compile gem5 normally: `scons build/ARM/gem5.opt`
2. Compile gem5 as a library:  
`scons --with-cxx-config --without-python --without-tcmalloc \`  
`build/ARM/libgem5_opt.so`
3. Include the **gem5 modules** `Gem5SimControl` and `Gem5SlaveTransactor` and/or `Gem5MasterTransactor` in your SystemC project and connect them to your SystemC models. Be sure to pass an individual port name to the constructor of each transactor.
4. Compile your project and link against the gem5 library.
5. Run normal gem5 with a custom python script or `fs.py` with `--tlm-memory=<port-name>` to generate `m5out/config.ini`. Be sure to set the `tlm_data` attribute of the External Masters/Slaves to the port name of the corresponding SystemC transactor.
6. Run your SystemC project and pass the `m5out/config.ini` file to your `Gem5SimControl` object.

# Hands On: A Memory Module in SystemC

```
struct Target: public sc_module {
    // TLM interface socket:
    tlm_utils::simple_target_socket<Target> socket;

    // Storage
    unsigned char *mem;

    // Constructor
    Target(sc_core::sc_module_name name, /* ... */);
    SC_HAS_PROCESS(Target);

    // TLM interface functions
    virtual void b_transport(tlm::tlm_generic_payload& trans,
                             sc_time& delay);
    virtual unsigned int transport_dbg(tlm::tlm_generic_payload& trans);
    virtual tlm::tlm_sync_enum nb_transport_fw(
        tlm::tlm_generic_payload& trans,
        tlm::tlm_phase& phase,
        sc_time& delay);

    // ...
};
```

→ util/tlm/examples/slave\_port/sc\_target.hh

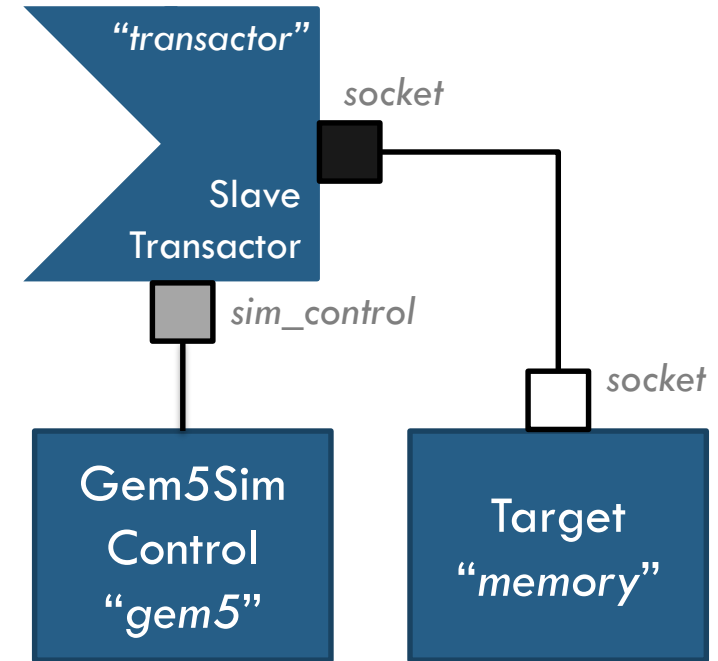
# Hands On: Connect the Memory to gem5

```
int sc_main(int argc, char **argv)
{
    // Instantiate all modules
    Gem5SystemC::Gem5SimControl
        sim_control("gem5", /* config ... */);
    Gem5SystemC::Gem5SlaveTransactor
        transactor("transactor", "transactor");
    Target memory("memory", /* config ... */);

    // Bind modules
    memory.socket.bind(transactor.socket);
    transactor.sim_control.bind(sim_control);

    // Start simulation
    sc_core::sc_start();

    return EXIT_SUCCESS;
}
```



➔ `util/tlm/examples/slave_port/main.cc`



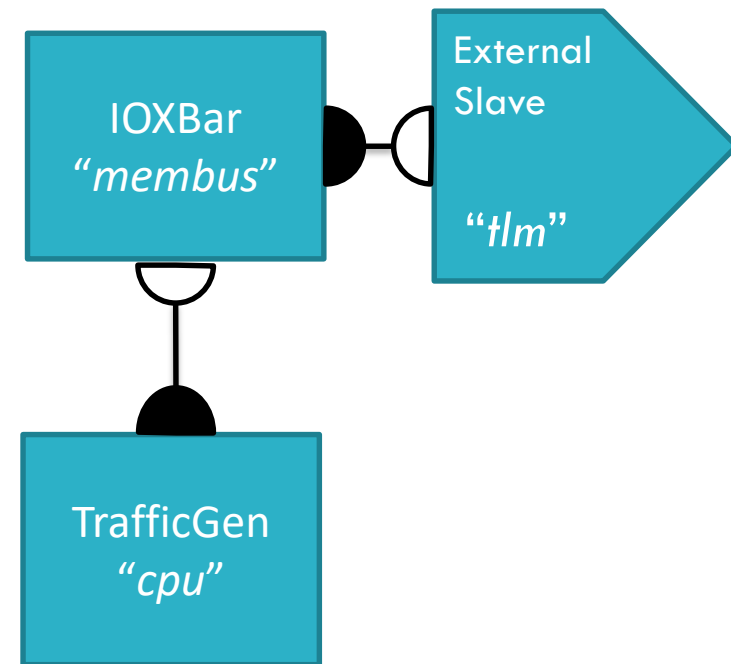
# Hands On: Configure gem5

```
# Create a system with a Crossbar and a
TrafficGenerator
system = System()
system.membus = IOXBar(width = 16)
# This must be instantiated, even if not needed
system.physmem = SimpleMemory()
system.cpu = TrafficGen(config_file = "tgen.cfg")
system.clk_domain = SrcClockDomain(clock = '1.5GHz',
    voltage_domain = VoltageDomain(voltage = '1V'))

# Create an external TLM port:
system.tlm = ExternalSlave()
system.tlm.addr_ranges = [AddrRange('512MB')]
system.tlm.port_type = "tlm_slave"
system.tlm.port_data = "transactor"

# Route the connections:
system.cpu.port = system.membus.slave
system.system_port = system.membus.slave
system.membus.master = system.tlm.port

# Start the simulation:
root = Root(full_system = False, system = system)
root.system.mem_mode = 'timing'
m5.instantiate()
m5.simulate()
```



# Hands On: Run the Simulation

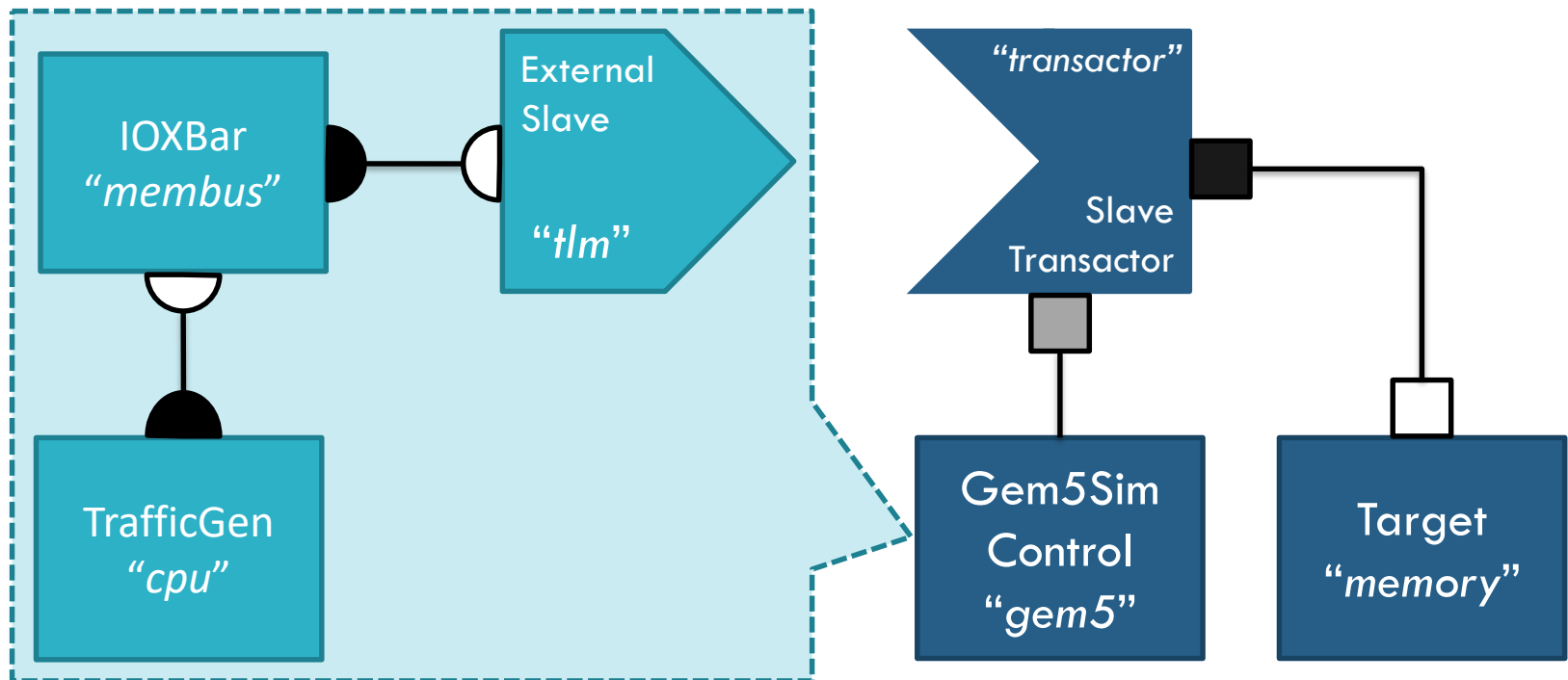
1. Build the example: `$ cd util/tlm && scons`

2. Create a gem5 config.ini file:

```
$ ../../build/ARM/gem5.opt conf/tlm_slave.py
```

3. Run the simulation:

```
$ build/examples/slave_port/gem5.sc m5out/config.ini
```



# Hands On: Simulation Output



```
$ build/examples/slave_port/gem5.sc m5out/config.ini -e 200000 -d TrafficGen
[...]  
0 s (=) : sc_main Start of Simulation  
info: Entering event queue @ 0. Starting simulation...  
5 ns (=) : system.cpu LinearGen::getNextPacket: r to addr 0, size 4  
5 ns (=) : system.cpu Next event scheduled at 10000  
10 ns (=) : system.cpu LinearGen::getNextPacket: w to addr 4, size 4  
15 ns (=) : system.cpu Received retry  
15 ns (=) : system.cpu LinearGen::getNextPacket: r to addr 8, size 4  
16675 ps (=) : system.cpu Received retry  
75 ns (=) : system.cpu Received retry  
75 ns (=) : system.cpu LinearGen::getNextPacket: r to addr c, size 4  
76038 ps (=) : system.cpu Received retry  
135 ns (=) : system.cpu Received retry  
135 ns (=) : system.cpu LinearGen::getNextPacket: r to addr 10, size 4  
136068 ps (=) : system.cpu Received retry  
195 ns (=) : system.cpu Received retry  
195 ns (=) : system.cpu LinearGen::getNextPacket: w to addr 14, size 4  
196098 ps (=) : system.cpu Received retry  
Exit at tick 200000, cause: simulate() limit reached
```

➔ The binary accepts various options:

- -e end of simulation at tick
- -d set a gem5 debug flag

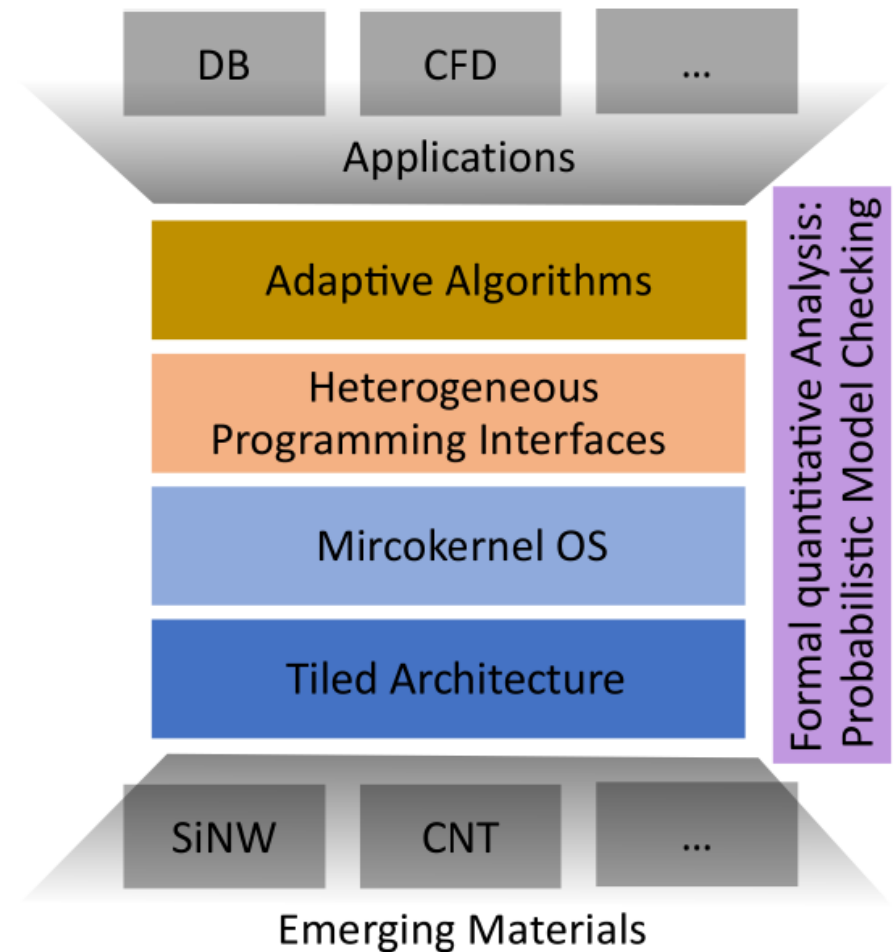
# Use cases

## The Orchestration Path at CFAED [2]

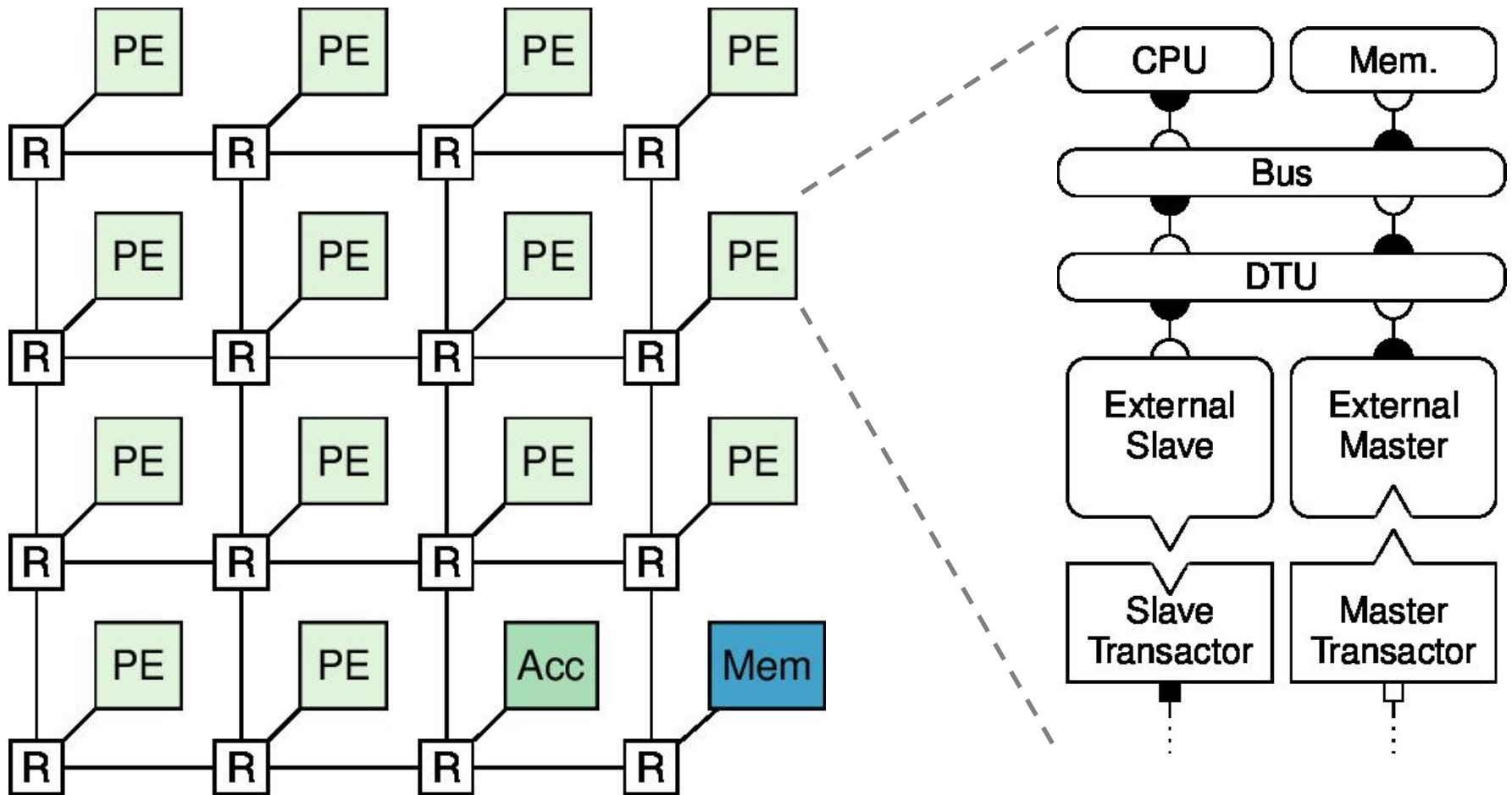
A programming stack for wildly heterogeneous systems including:

- ❑ dataflow programming models
- ❑ dataflow compiler
- ❑ adaptive runtime systems
- ❑ capability-based OS
- ❑ model checker

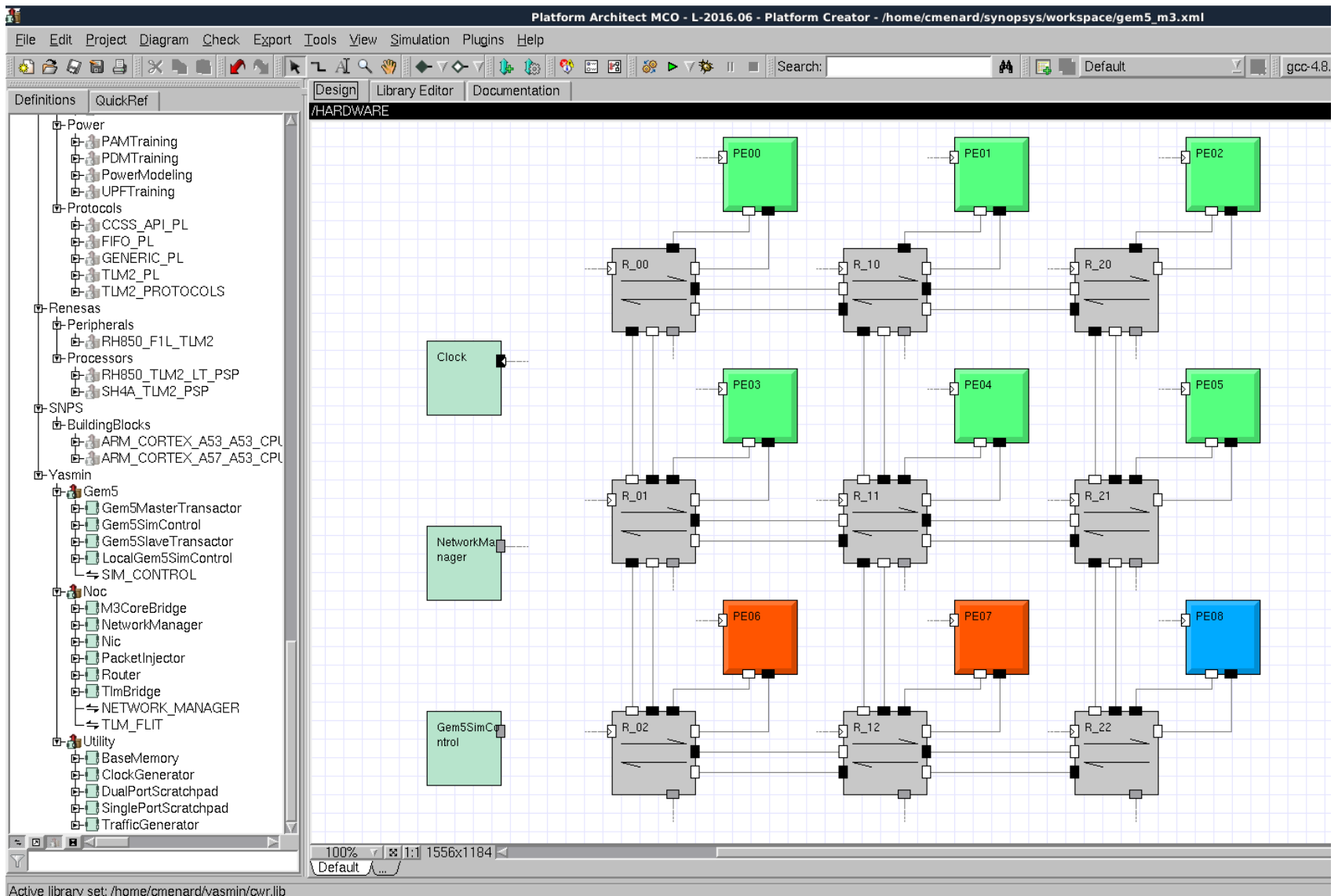
➔ A flexible simulation platform is required to try new designs and technologies.



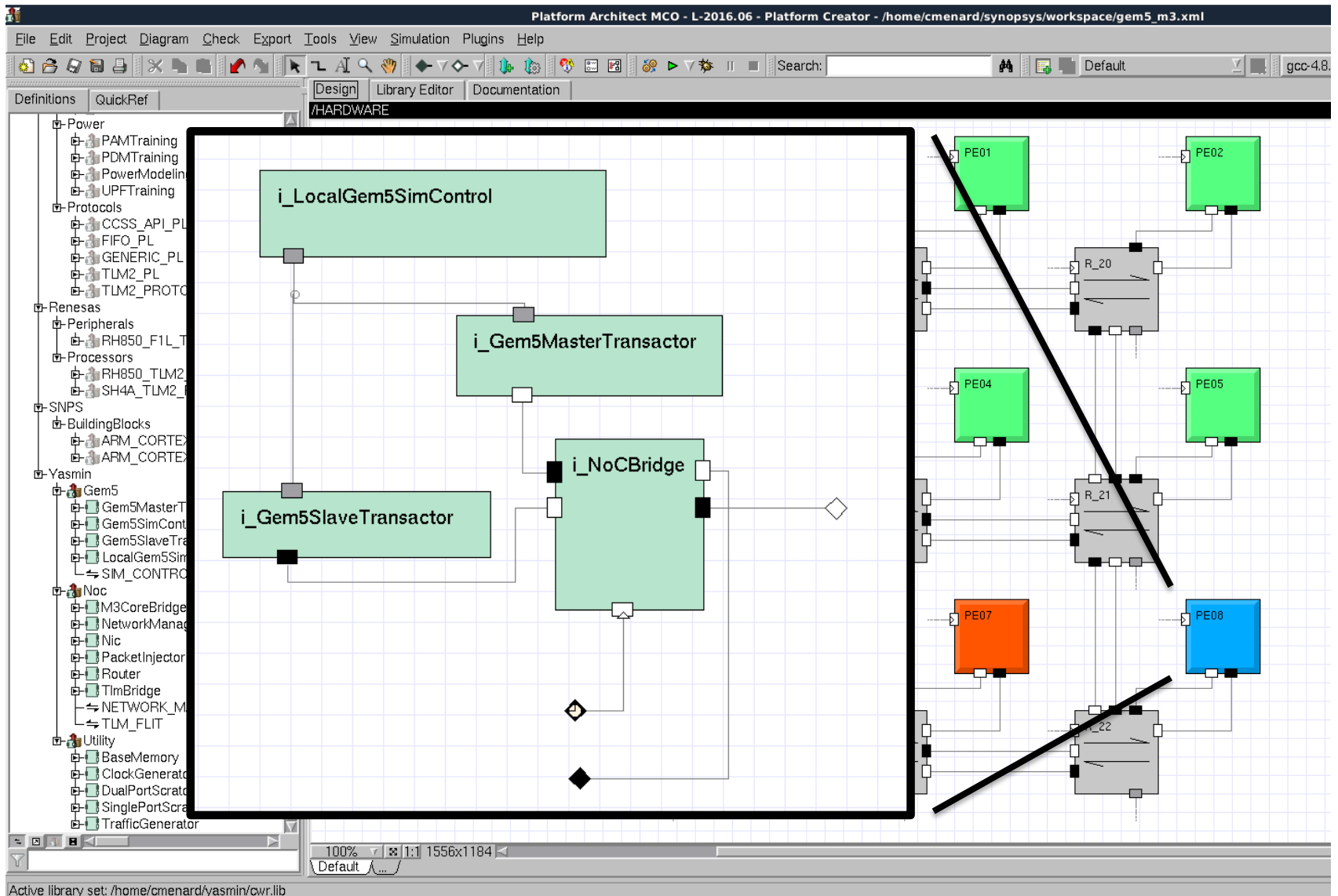
# Building Heterogeneous MPSoCs: gem5 as a Tile



# gem5 in Synopsys Platform Architect

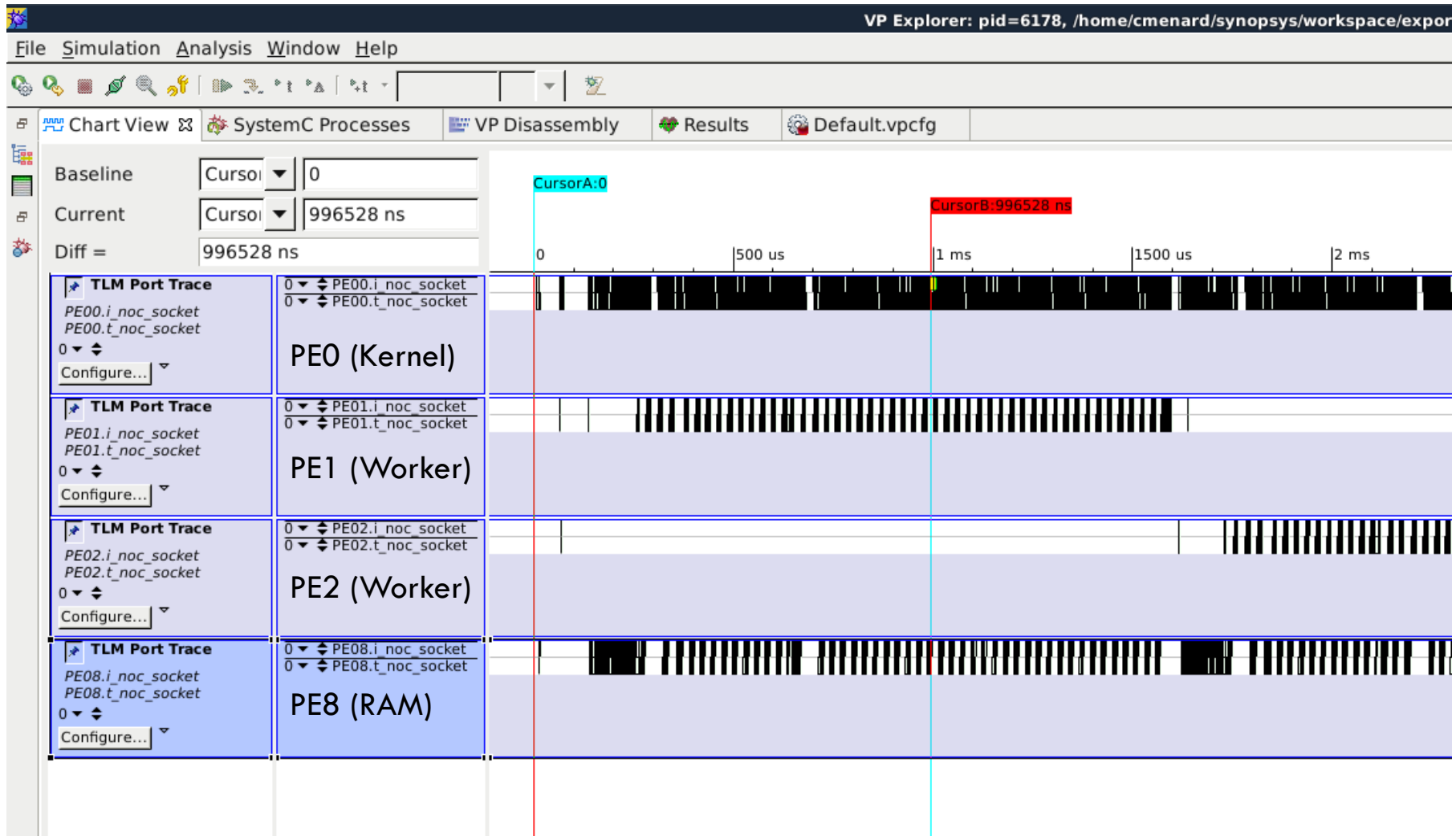


# gem5 in Synopsys Platform Architect

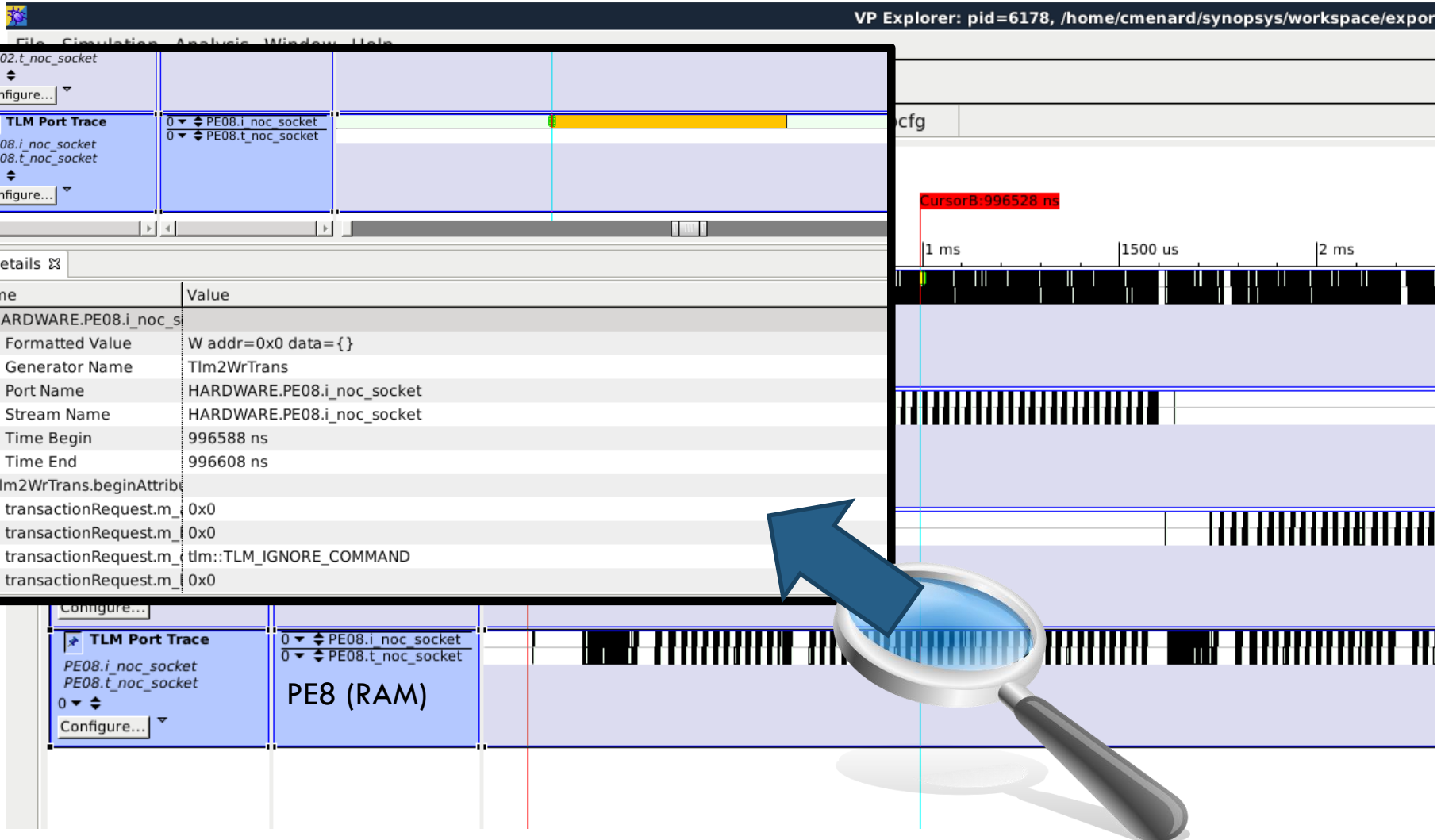




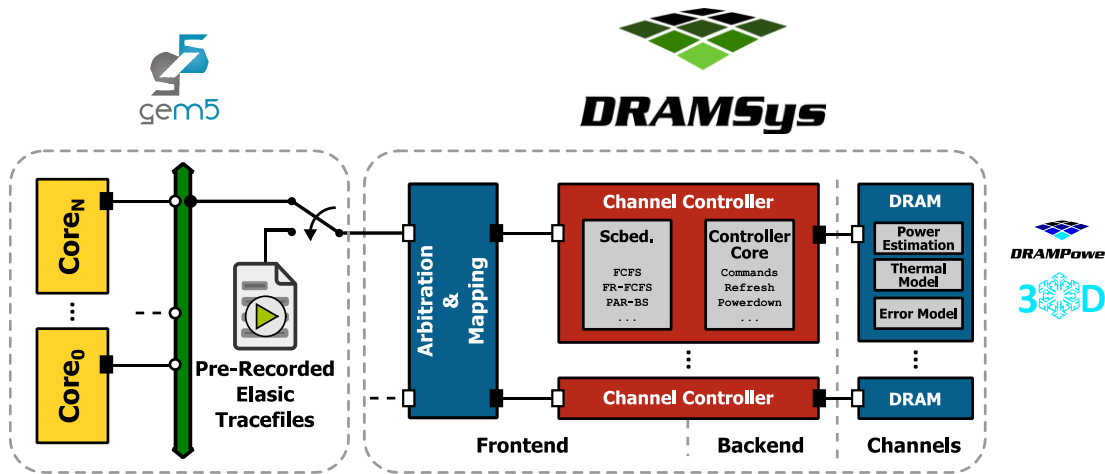
# gem5 in Synopsys Platform Architect: Trace Analysis



# gem5 in Synopsys Platform Architect: Trace Analysis



# Coupling gem5 with DRAMSys [3], [4]



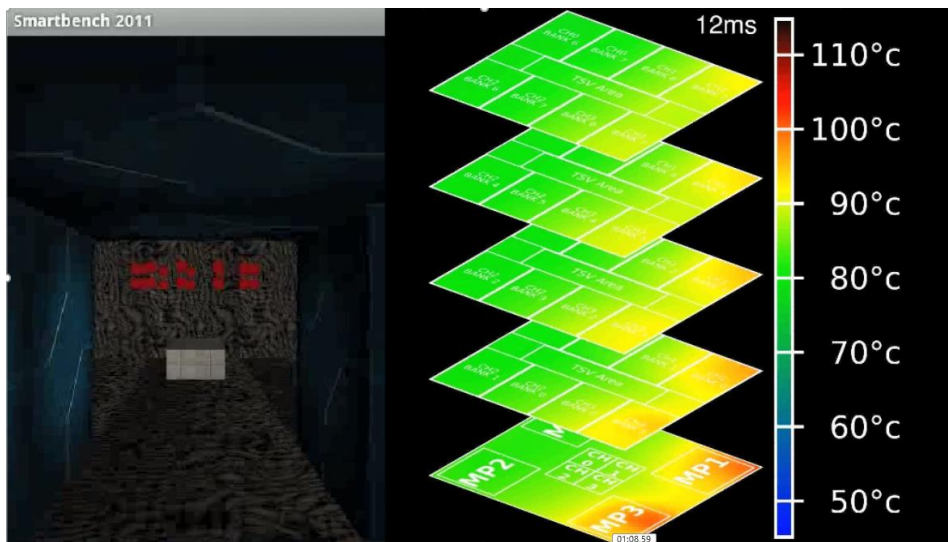
DRAMSys is a design space exploration framework for DRAM and memory controller

It includes:

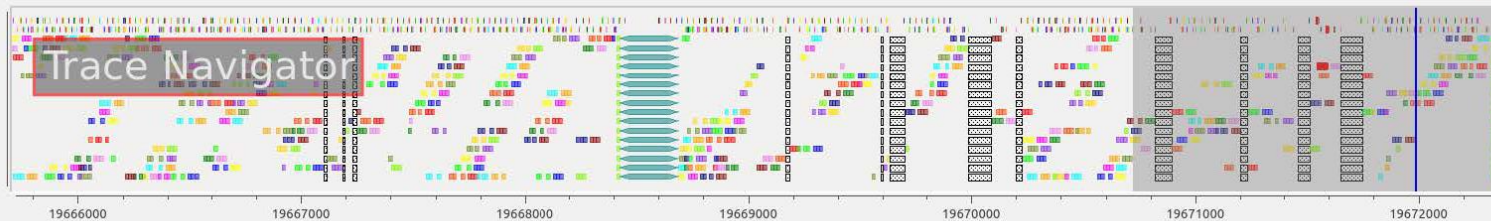
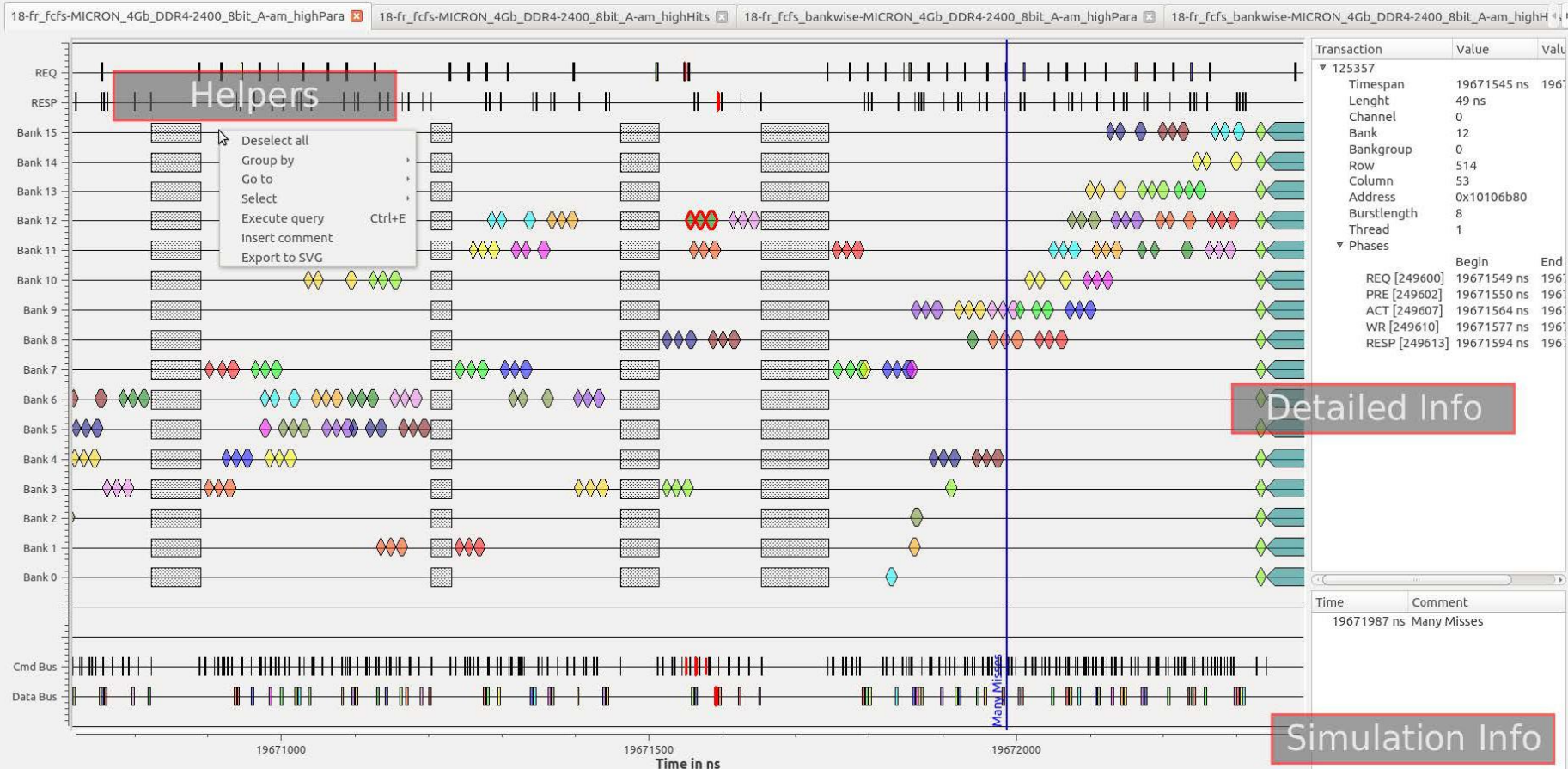
- ☐ Power model
- ☐ Thermal model
- ☐ Retention error model.

Linux boot (without thermal model) using the DRAMSys model → slowdown of 1.9×

This slowdown mostly comes from detailed DRAMSys model



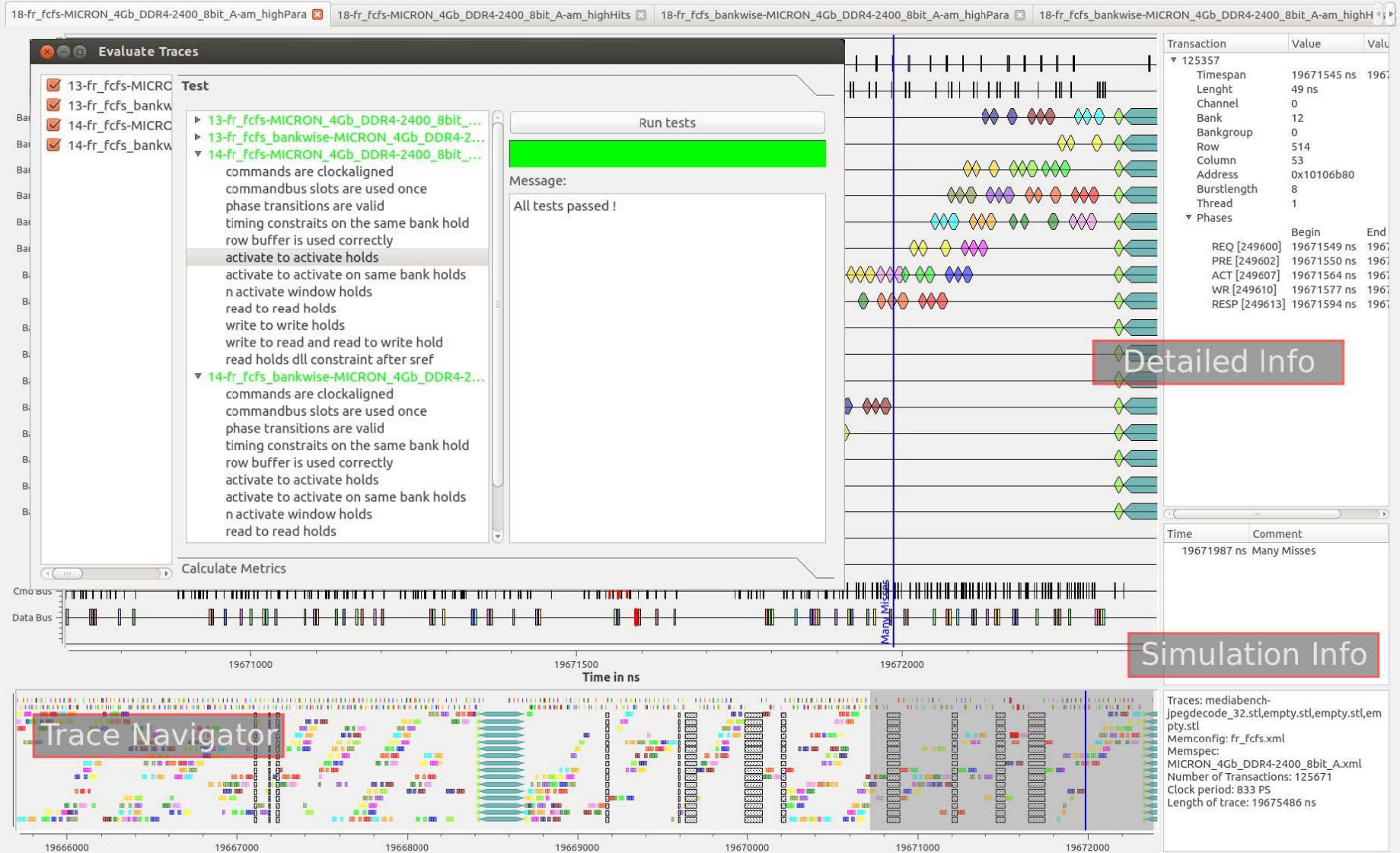
# Coupling gem5 with DRAMSys Continued



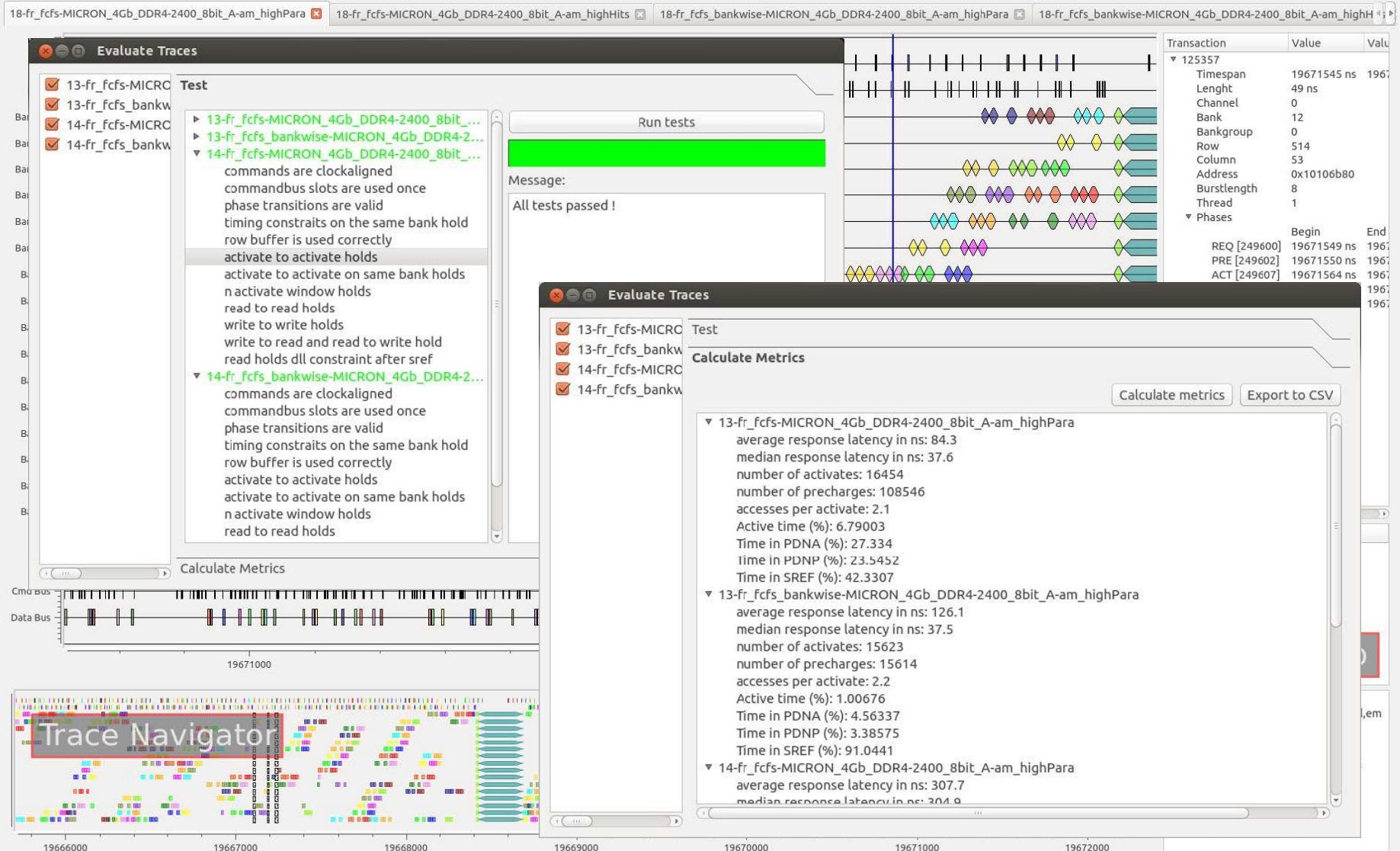
Traces: mediabench-jpegdecode\_32.stl,empty.stl,empty.stl,empty.stl  
Memconfig: fr\_fcfs.xml  
Memspec:  
MICRON\_4Gb\_DDR4-2400\_8bit\_A.xml  
Number of Transactions: 125671  
Clock period: 833 PS  
Length of trace: 19675486 ns



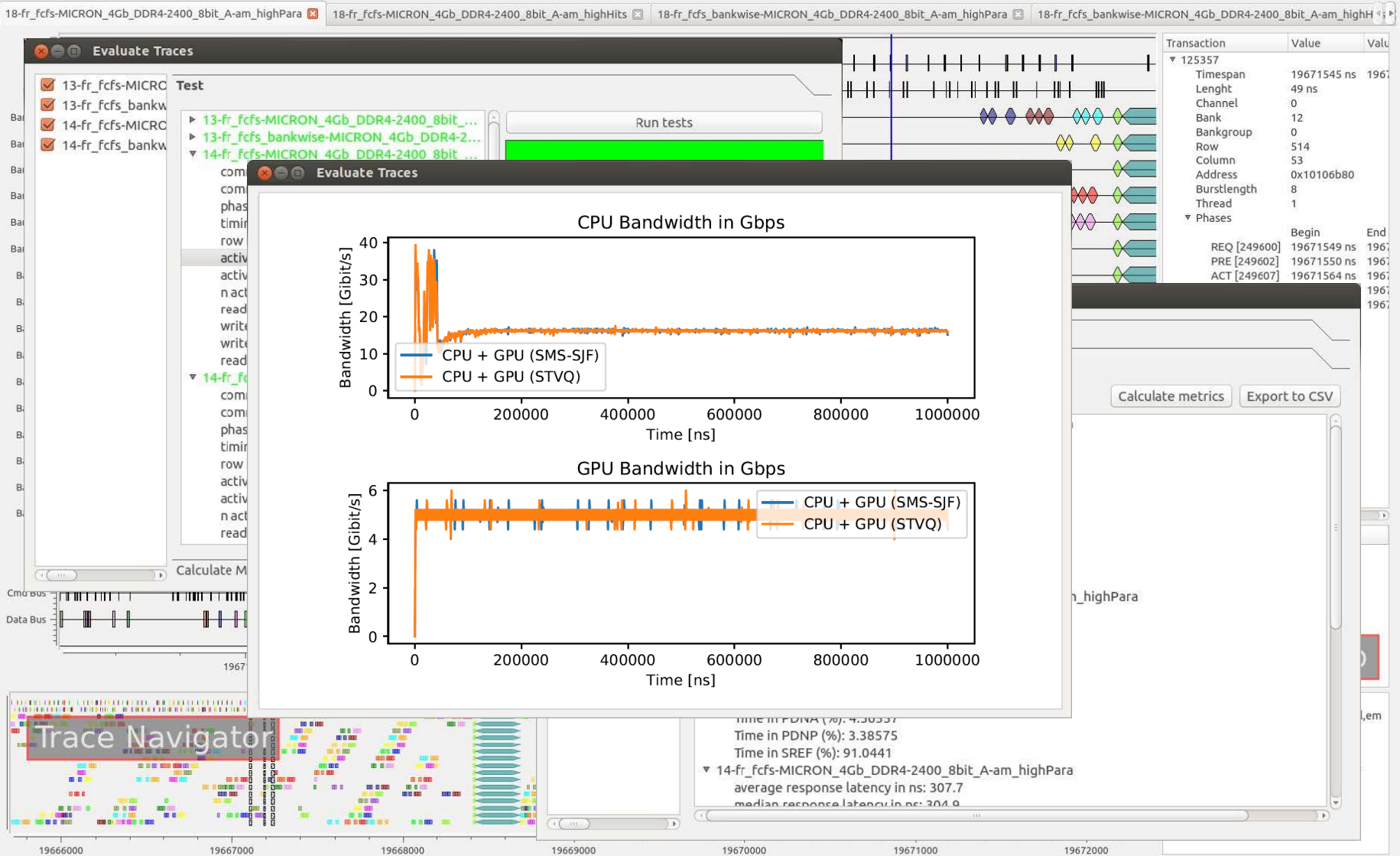
# Coupling gem5 with DRAMSys Continued



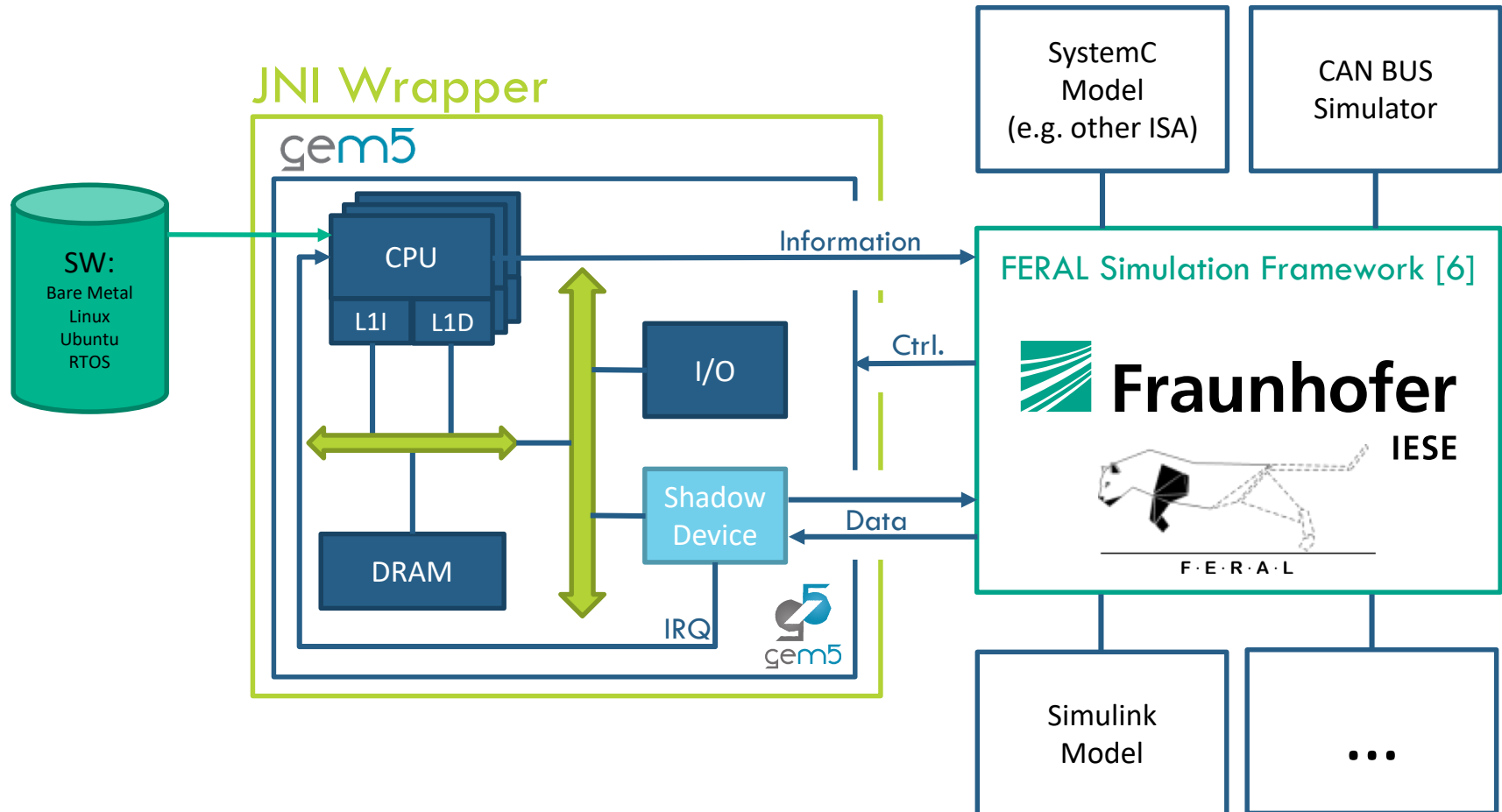
# Coupling gem5 with DRAMSys Continued



# Coupling gem5 with DRAMSys Continued



# Connecting gem5 to Non-SystemC Simulators





- ❑ Coupling of different Simulators with different models of computation (e.g. Simulink with gem5 or SystemC)
- ❑ gem5 compiled as C++ library and wrapped in JNI wrapper
  
- ❑ Development of software concepts
  - ❑ Simulation of systems of systems
  - ❑ Combining different levels of abstraction
  
- ❑ Software Testing:
  - ❑ Normal software testing instruments source code or binary → Intrusive
  - ❑ gem5 is instrumented instead
  - ❑ Supervised testing of concurrent software
  - ❑ Fault Injection
  - ❑ Coverage

- ❑ Full interoperability between gem5 and SystemC
- ❑ Fully compliant to the SystemC standard
- ❑ It is part of the gem5 repository!

→ Next steps:

- replace the entire simulation kernel and communication System by SystemC/TLM (?)
- Remove step for .ini generation (?)
- Suggestions? We are open!

**Thank you!**

- [1] Christian Menard, Matthias Jung, Jeronimo Castrillon, Norbert Wehn, **"System Simulation with gem5 and SystemC: The Keystone for Full Interoperability"**, *Proceedings of the IEEE International Conference on Embedded Computer Systems Architectures Modeling and Simulation (SAMOS)*, Jul 2017.
- [2] Marcus Völp, Sascha Klüppelholz, Jeronimo Castrillon, Hermann Härtig, et al. **"The Orchestration Stack: The Impossible Task of Designing Software for Unknown Future Post-CMOS Hardware"**, *Proceedings of the 1st International Workshop on Post-Moore's Era Supercomputing (PMES), Co-located with The International Conference for High Performance Computing, Networking, Storage and Analysis (SC16)*, Salt Lake City, USA, Nov 2016
- [3] Matthias Jung, Christian Weis, Norbert Wehn, Karthik Chandrasekar, **"TLM Modelling of 3D Stacked Wide I/O DRAM Subsystems, A Virtual Platform for Memory Controller Design Space Exploration"** *Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO)*, January, 2013, Berlin.
- [4] Matthias Jung, Christian Weis, Norbert Wehn, **"DRAMSys: A flexible DRAM Subsystem Design Space Exploration Framework"**, *IPSI Transactions on System LSI Design Methodology (T-SLDM)*, October, 2015.
- [5] Radhika Jagtap, Stefan Diestelhorst, Andreas Hansson, Matthias Jung, Norbert Wehn, **"Exploring System Performance using Elastic Traces: Fast, Accurate and Portable"**, *IEEE International Conference on Embedded Computer Systems Architectures Modeling and Simulation (SAMOS)*, July, 2016, Samos Island, Greece.
- [6] Thomas Kuhn, Thomas Forster, Tobias Braun, Reinhard Gotzhein, **"FERAL - Framework for Simulator Coupling on Requirements and Architecture Level"**, *IEEE Computer Society: Eleventh ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2013. Proceedings : 8-20 Oct. 2013, Portland, OR*