2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
OCTOBER 14-15, 2025

Efficient SoC Modeling, Architectural Exploration, and Result Analysis using TLM2 based IP

Rocco Jonack; MINRES Technologies GmbH

Mathias Jung; FHG

Jean-Blaise Pierre; Arteris

accellera SYSTEMS INITIATIVE

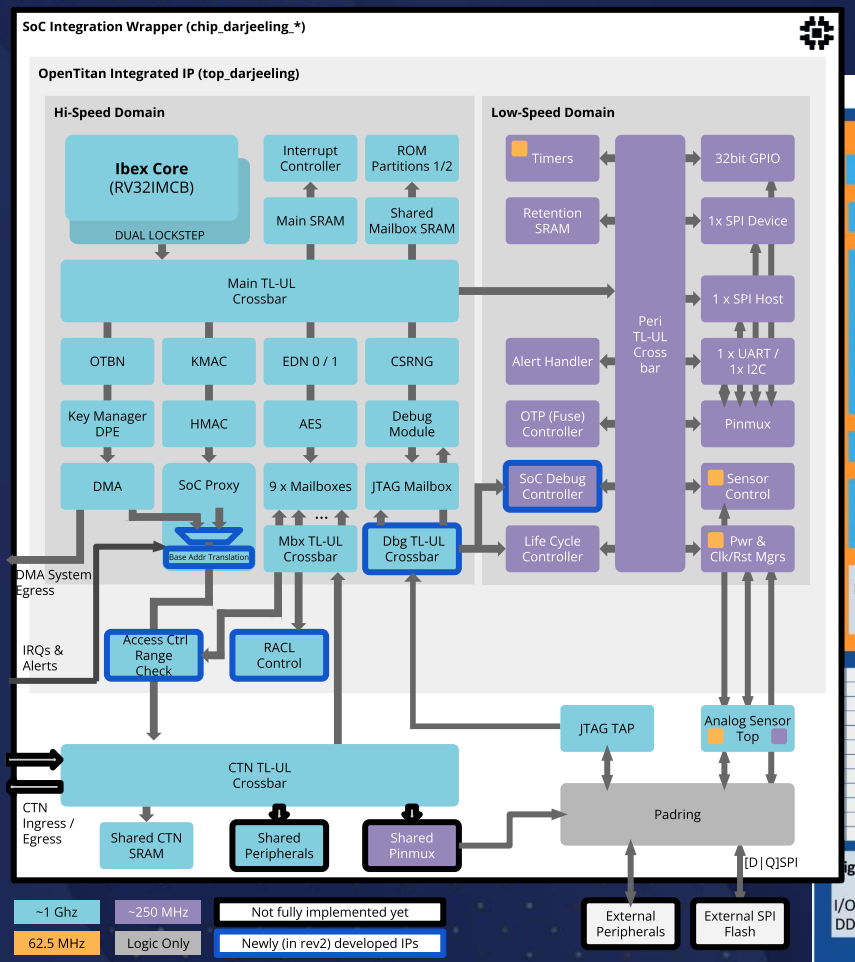
Agenda

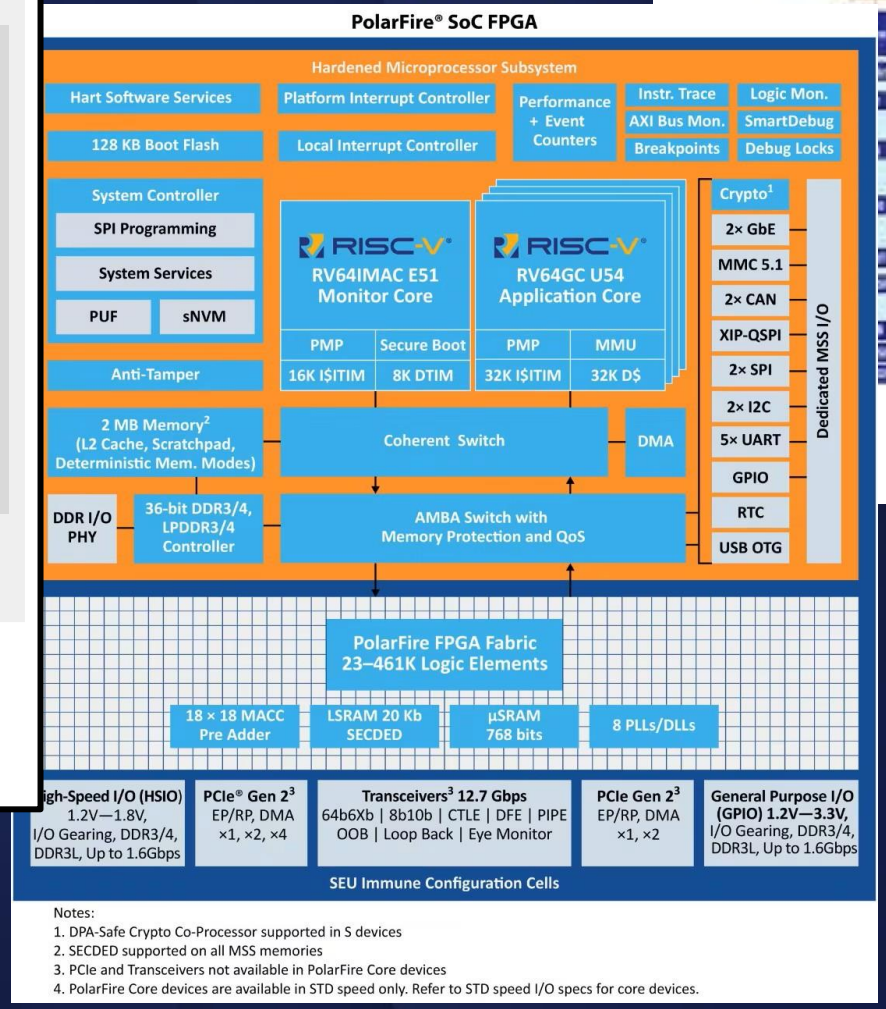
- **Motivation**
- Setting up platforms
- IP in platform modeling
- Performance analysis and visualization

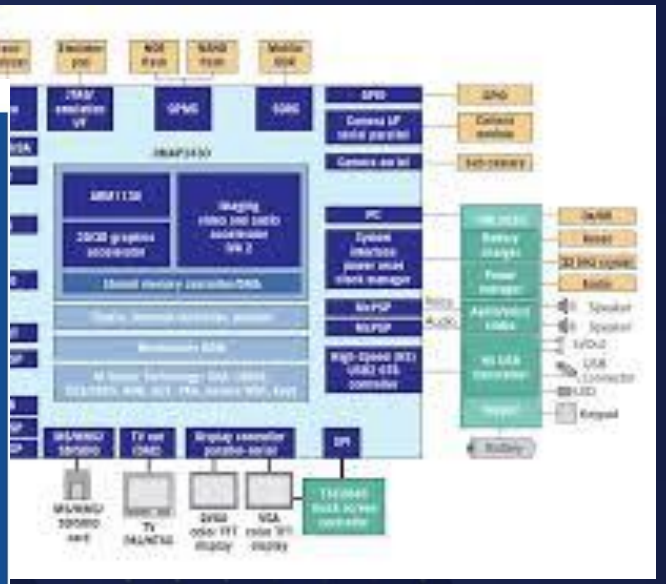
Motivation for models

- Model usage for SoC design has many use cases
  - No model is perfect
  - Various abstractions
    - SW development and SW driven testing
    - Architecture analysis
    - Power analysis
    - Augmentation of HW tests with realistic stimulus
    - Mixing abstraction levels
- Requirements for model-based platforms
  - Fast availability and flexibility with limited resources
  - Different host requirements
  - Infrastructure tasks
    - Tracing, logging, simulation control

Example SoC platforms





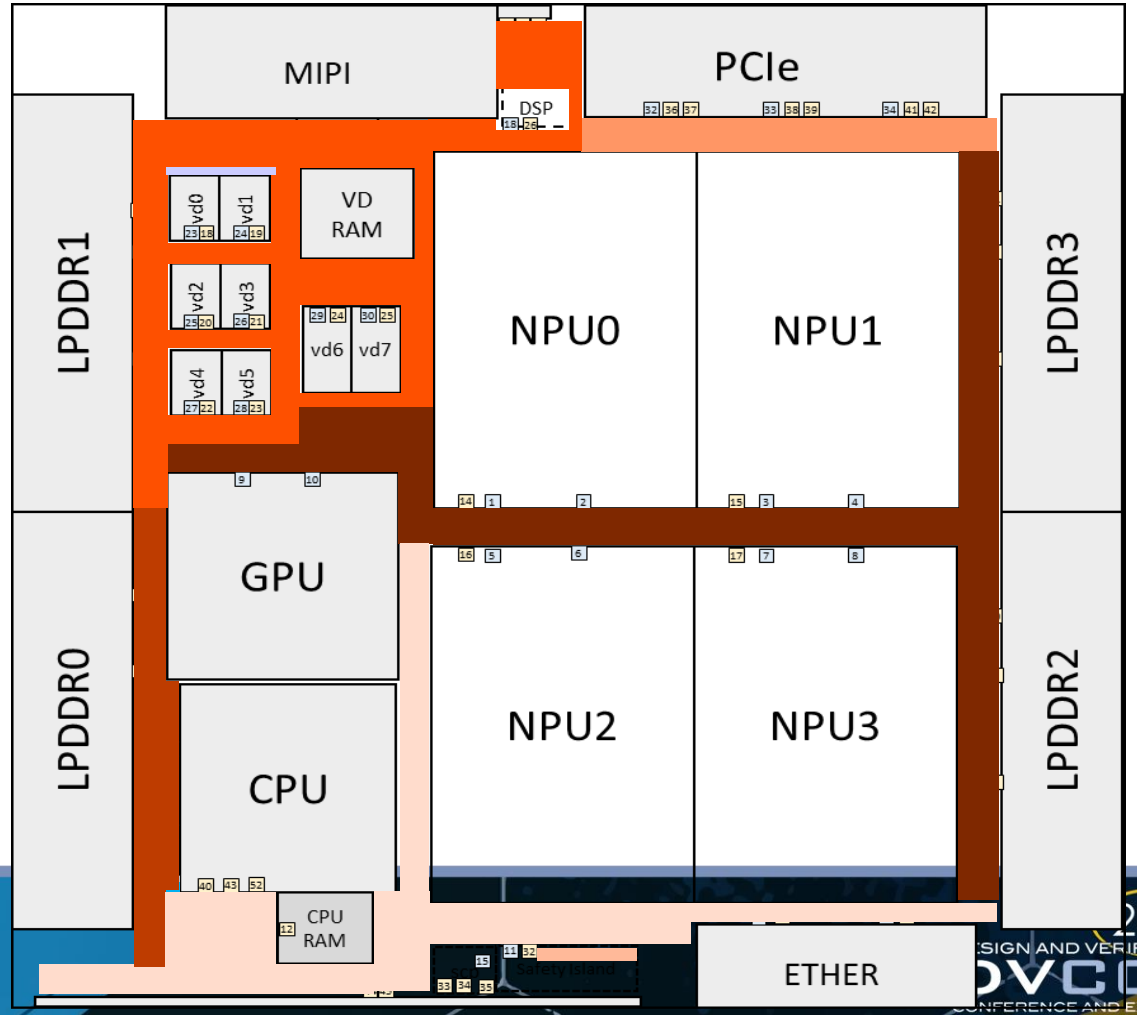


accellera SYSTEMS INITIATIVE

From IP to SoCs

- 1 NPU Subsystem
- CPU Subsystem
- Video Subsystem
- PCIe Subsystem
- Safety Subsystem
- Cache Coherency

- Common functional blocks
- Multiple instances
- Common infrastructure tasks

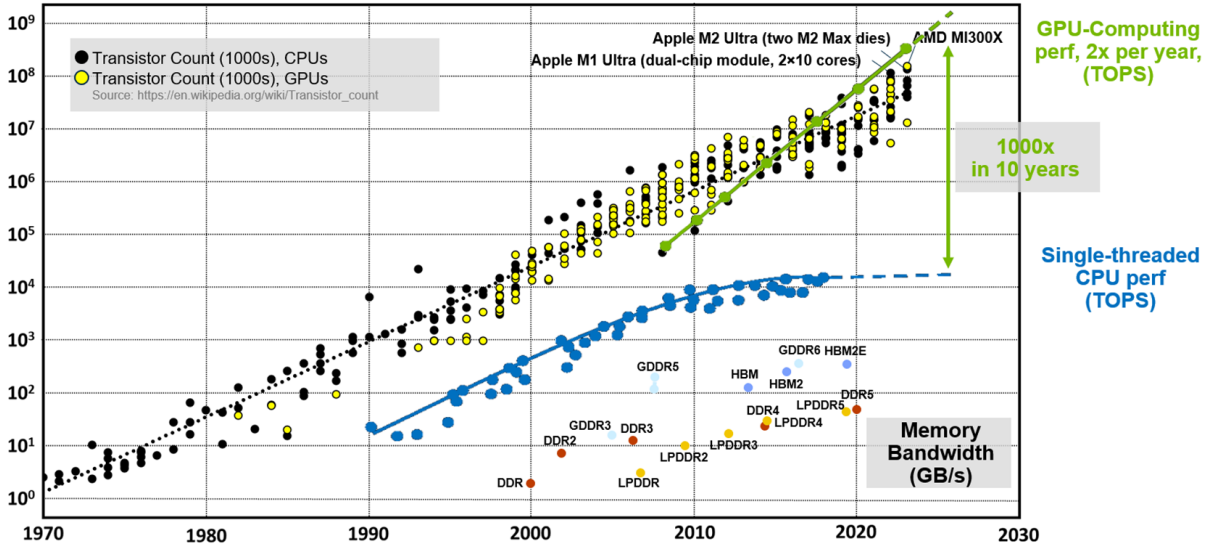


Data Storage and Transport

Memory and Data Transport Issues

- DRAM accesses kill performance and power consumption
- SoC architects must balance local data reuse vs. sharing within the chip and into the system
  - Closely coupled memories and internal SRAMs
  - Common buffer RAMs (managed at software level)
  - Hardware cache coherence
- Highly complex chips will have a mix of all of these
  - Recurrent neural networks (RNN) and other feedback networks can benefit from caches
- Paths between custom hardware elements must be optimized
  - **Bandwidth** –wide paths when needed, narrower when not
  - **Latency** –focus on paths that constrain near real-time performance

Growing SoC Complexity – CPU, GPU Performance and Memory

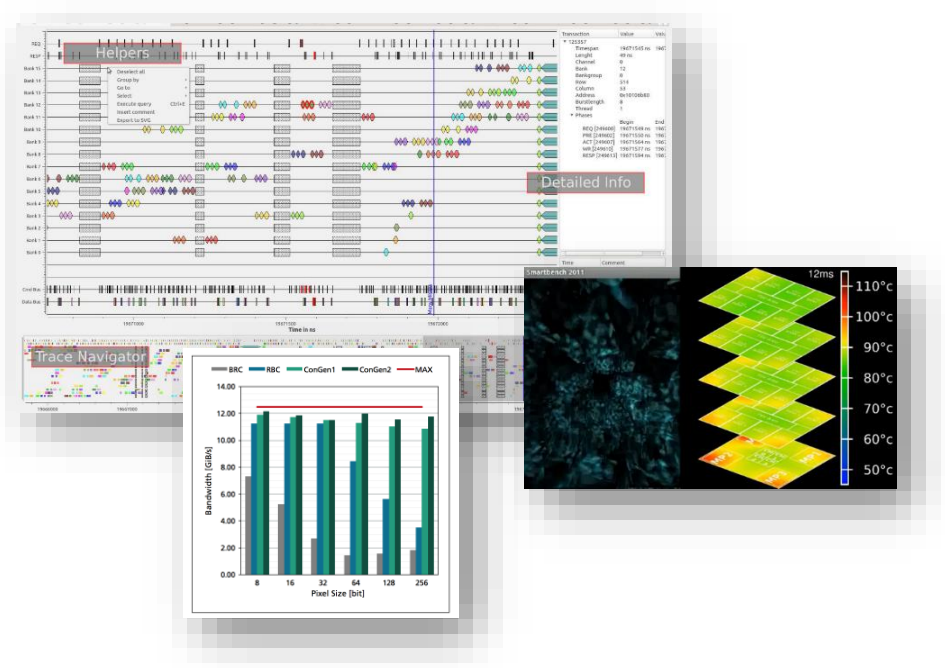


Source: Frank Schirrmeister, Arteris, plotted based on https://bit.ly/3P5y9OK, NVIDIA and "Fast Validation of DRAM Protocols with Timed Petri Nets", M. Jung et. al, MEMSYS 2019

DRAMSys in a Nutshell

Simulation and Design Space Exploration of Modern DRAM-based Memory Systems:

- Which DRAM configuration?
- When to support DDR5 or LPDDR5?
- How to configure the memory controller?
- What is the system-level application behavior?



DRAMSys Offers:

- High-speed and flexible models of all standards
- Fast and accurate design space exploration
- Early identification of bottlenecks
- Connection to cores (e.g. SystemC, gem5, …)

Agenda

- Motivation
- **Setting up platforms**
- IP in platform modeling
- Performance analysis and visualization

Mechanisms for platform creation

- Commercial and open-source tools for platform creation
  - License implications
  - Abstraction level, host and IP support
- Manual maintenance of top level
  - Capturing variety of use cases
- Meta data support for generation, configuration and analysis
  - IPXACT, RDL variations, abstract execution modeling
- Integration of open-source tools/libraries and Python
  - Large eco system
  - Planning required

System Composition using PySysC

- Python Binding for SystemC based platforms
- Composition of systems using Python scripts
  - support for structural construction, simulation control and dynamic model parametrization
  - Leveraging cppyy library for C++ inspection
- Leverage broad availability of Python integrations
  - Computational models using numpy/scipy/pandas
  - UIs and cockpits using GTK, wxWidgets or Qt
  - Visualization libraries like dash and mathplotlib

PySysC Example

1. Instantiation of a module
2. Instantiation of a templated module
3. Named signal connection
4. TLM2.0 socket connection
5. Simulation run

```
from cppyy import gbl as cpp
from cppyy.gbl import sc_core
from pysysc.structural import Connection, Signal, Module, Simulation
# loading required libraries
...
# instantiating modules
clk_gen = Module(cpp.ClkGen).create("clk_gen")                    ## (1)
initiator = Module(cpp.Initiator).create("initiator")
memories = [Module(cpp.Memory).create(name)
                for name in ["mem0", "mem1", "mem2", "mem3"]]
router = Module(cpp.Router[4]).create("router")                   ## (2)
# creating connections
clk = Signal("clk")
        .src(clk_gen.clk_o)
        .sink(initiator.clk_i)
        .sink(router.clk_i)                                       ## (3)
[clk.sink(m.clk_i) for m in memories]
Connection()
        .src(initiator.socket)
        .sink(router.target_socket)                               ## (4)
[Connection()
        .src(router.initiator_socket.at(idx))
        .sink(m.socket)
        for idx,m in enumerate(memories)]
# run simulation
sc_core.sc_start()                                                ## (5)
```

Meta data

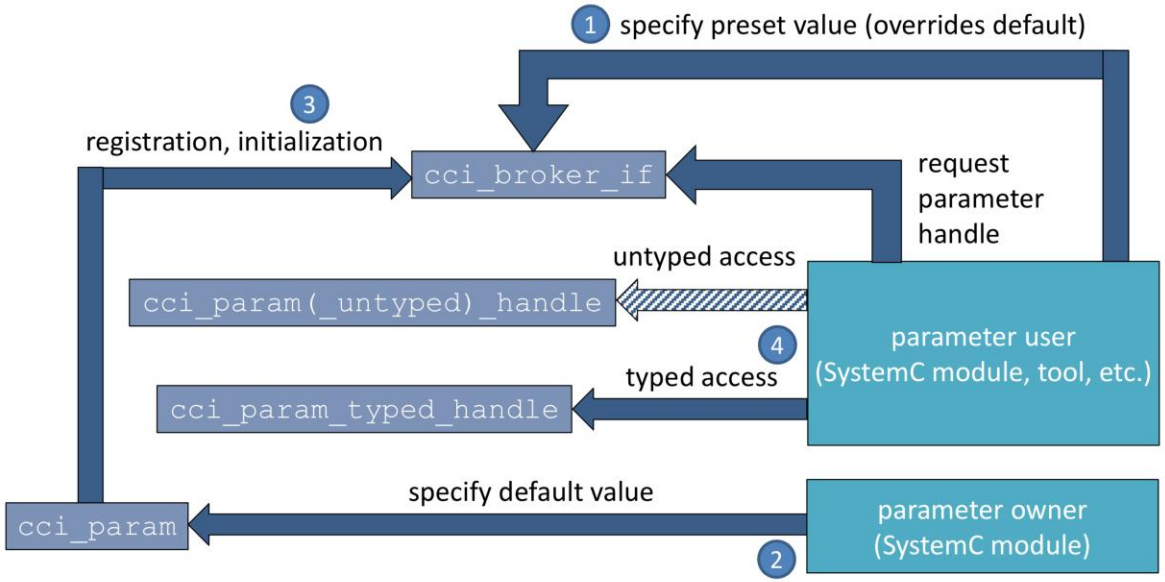
- Structural information
  - IPXACT as largest standard
  - Needs tools for management
  - Some simplified formats can be useful
- Register information
  - IPXACT and various RDL variations exist
  - Tools for management of input and output of formats
    - Documentation, SW access, verification
- Execution modeling
  - Formally specifying common runtime functions like boot or security setup

Build System

- CMake based build system
- Supports conan.io for 3rd party packages (even SystemC and CCI are available this way)
- Supports Windows and Linux (various compiler versions)
- Supports OSCI as well as SNPS PA, CDNS Xcelium, and MINRES RAVEN SystemC transparently
- Integrates Verilator, doxygen, clang-format, and clang-tidy

Parametrization with CCI

- Core concept is the pairing of parameters and brokers
- Parameter creation and direct access
- Parameter lookup and access via a handle
- (Sub-)system packaging and integration using local brokers

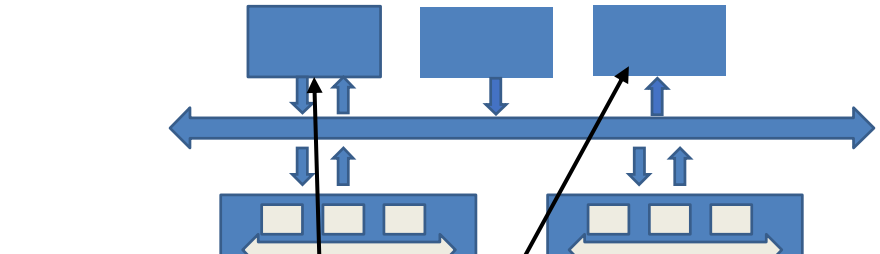


Logging

- Unifying logging aids model analysis and debug
  - Common formatting for post processing
  - Common information like timestamps and module names
  - Common logging levels
- Dynamically enable/disabling for logging mechanisms
  - Hierarchical enable/disable
- Filtering of log generation

Productivity Libraries

- Common elements should be modeled on top of common classes
  - Registers, Memories, etc.
  - Efficient for stubs or preliminary implementations
- Common infrastructure tasks should be modeled on top of common classes
  - Logging
  - Parametrization
  - Domain handling (Clock, reset, power)



```
static const char* MSGID = "MY_SIM";
void my_module::some_thread() {
  SC_REPORT_INFO(MSGID,"Starting");
  …
  if (error_condition) {
    SC_REPORT_ERROR(MSGID,
      "Oops something bad happened");
```

```
tParam;
/// Example scml unsigned int parameter.
scml_property<unsigned int> scmlUIntParam;
/// Example scml bool parameter.
scml_property<bool>         scmlBoolParam;
/// Example scml double parameter.
```

Tracing

- Flexibility is paramount
- Huge runtime impact
- Ties into parametrization

Demo

- Setting up some example configs
- Hierarchy viewer
- Config interaction

Agenda

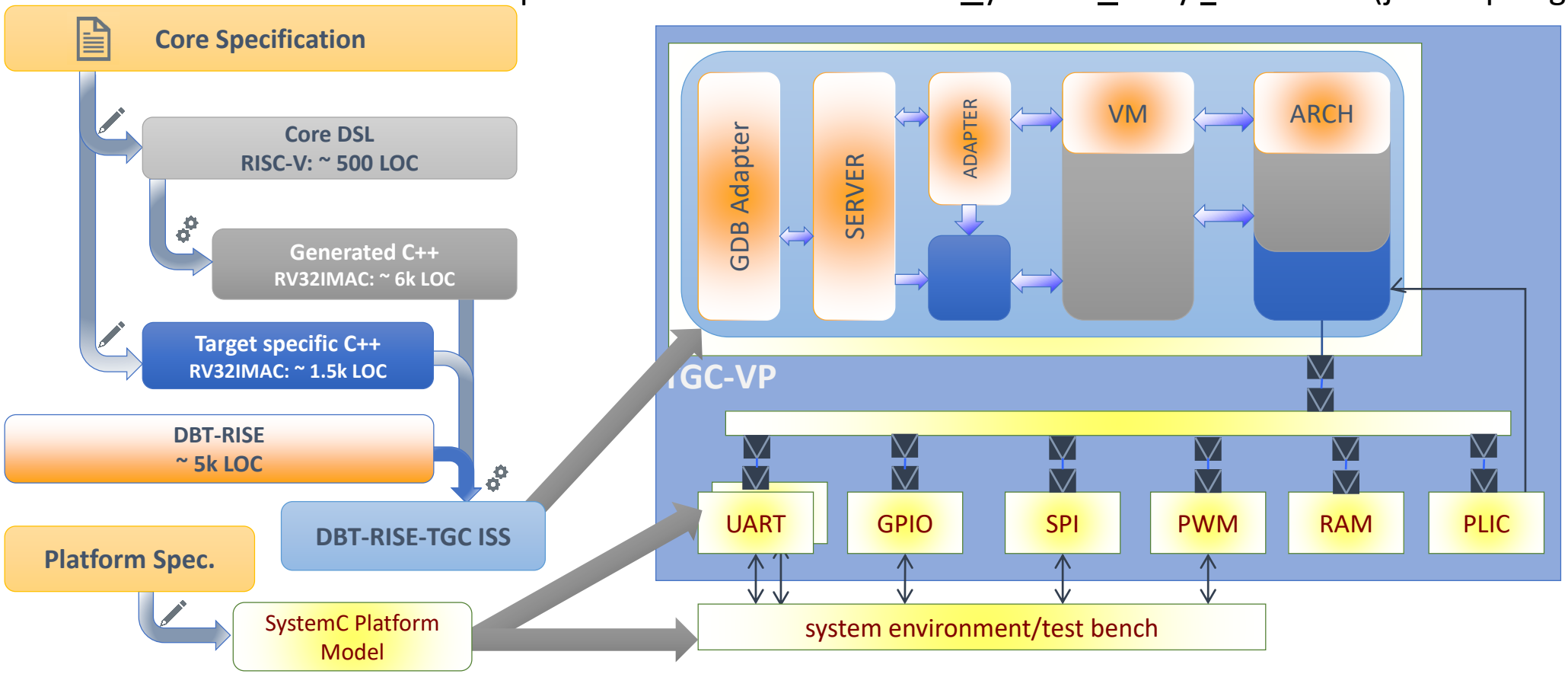
- Motivation
- Setting up platforms
- **IP in platform modeling**
- Performance analysis and visualization

Common functionality

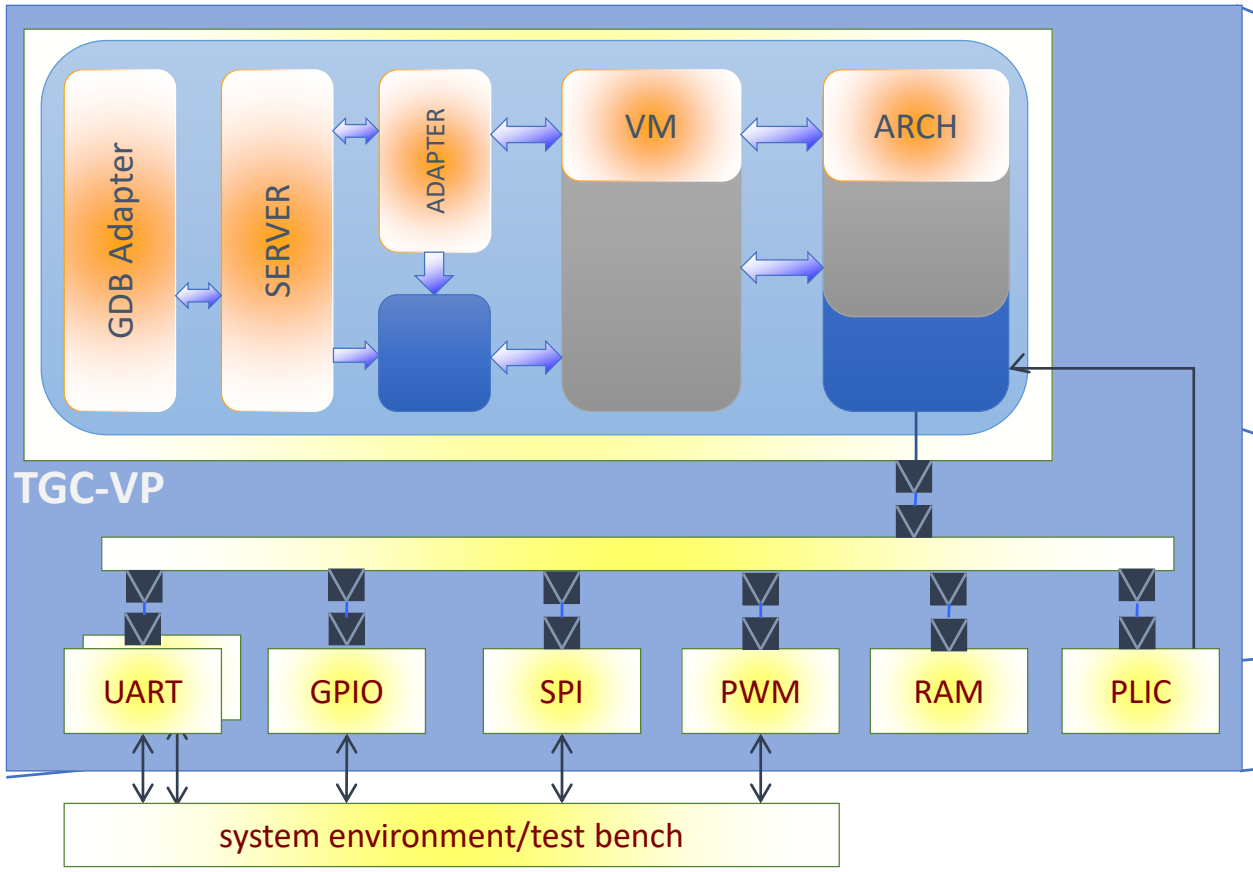
- Traffic source
  - CPU models
  - Workload execution
  - Trace replay
- Traffic sink
  - DRAM modeling
  - Register modeling
- Transport
  - Coherent and non coherent traffic support
  - Cache levels

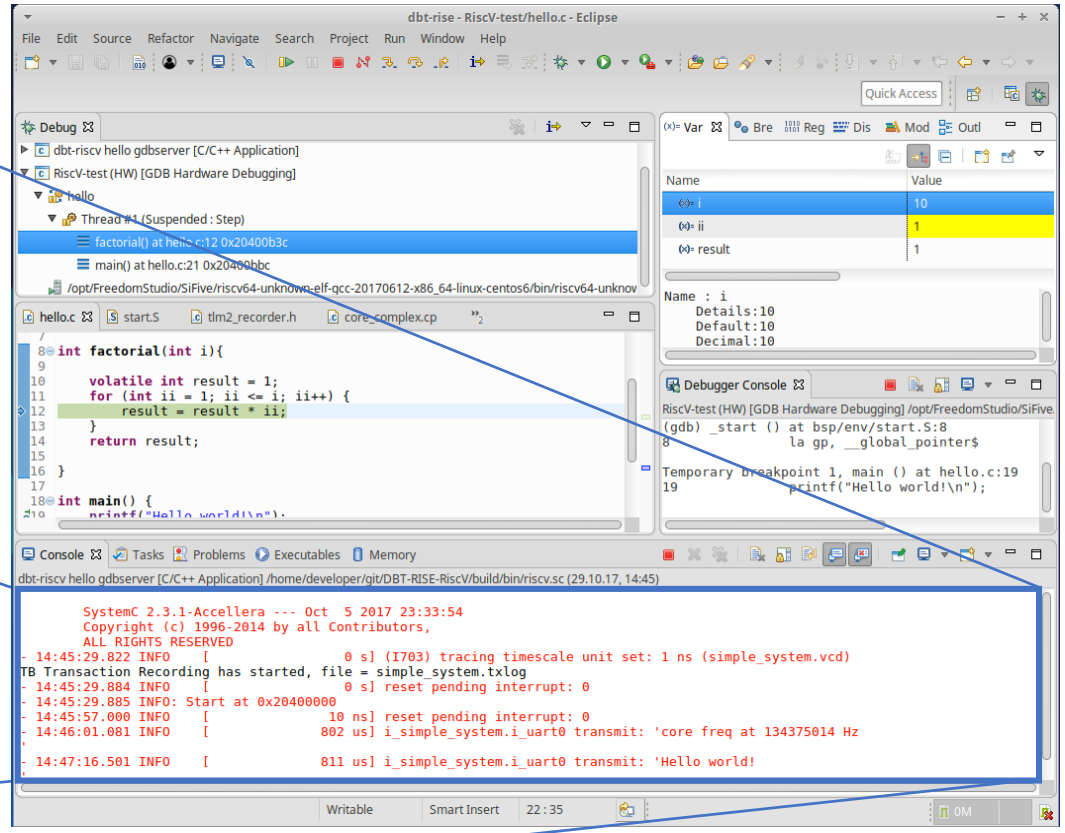
DBT-RISE based Platform

Open-Source Infrastructure for Dynamic Binary Translation (jit compiling) for ISS



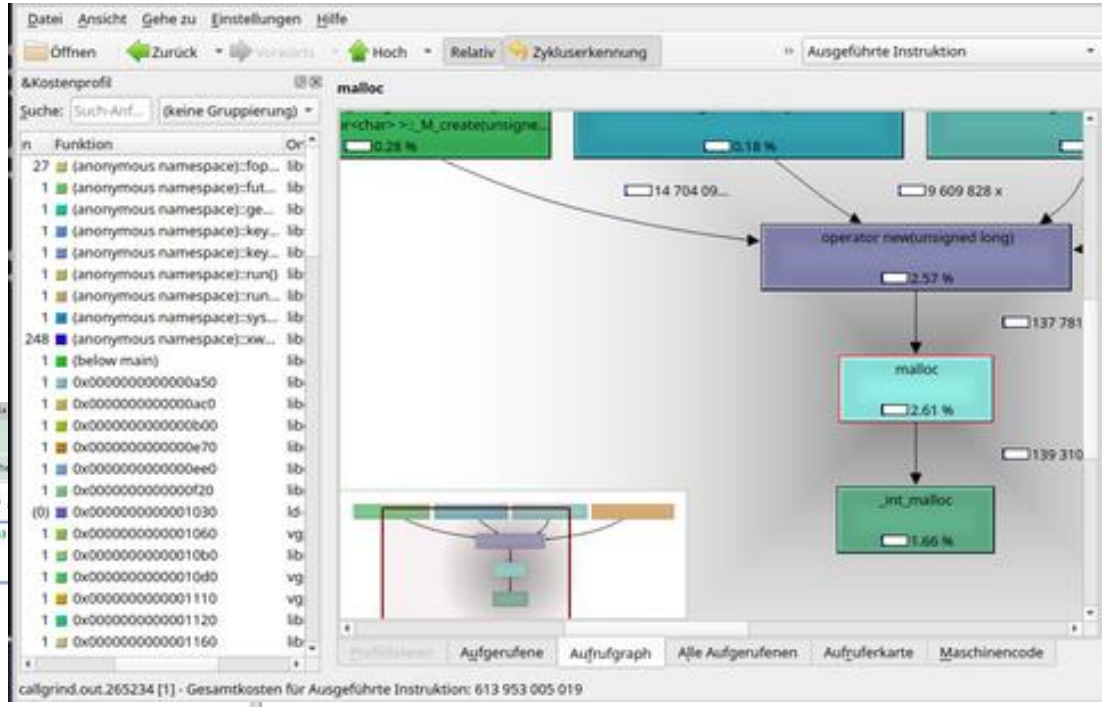
DBT-RISE RISC-V VP

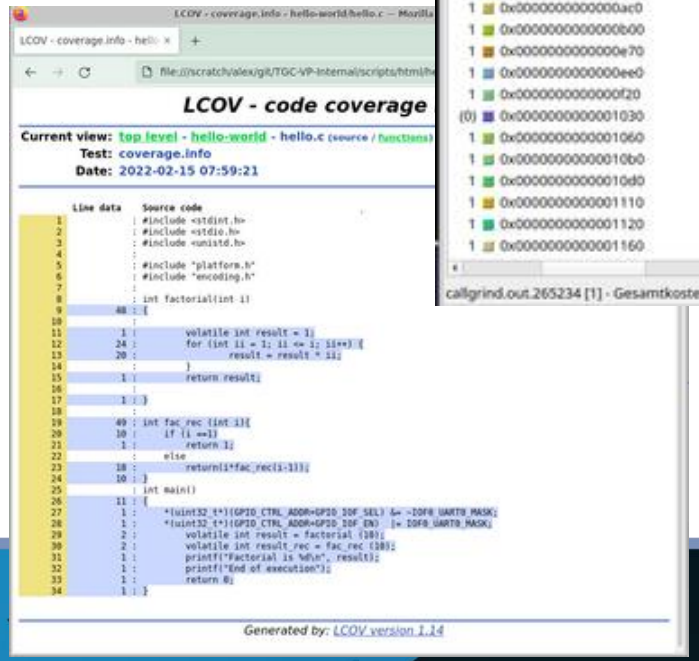




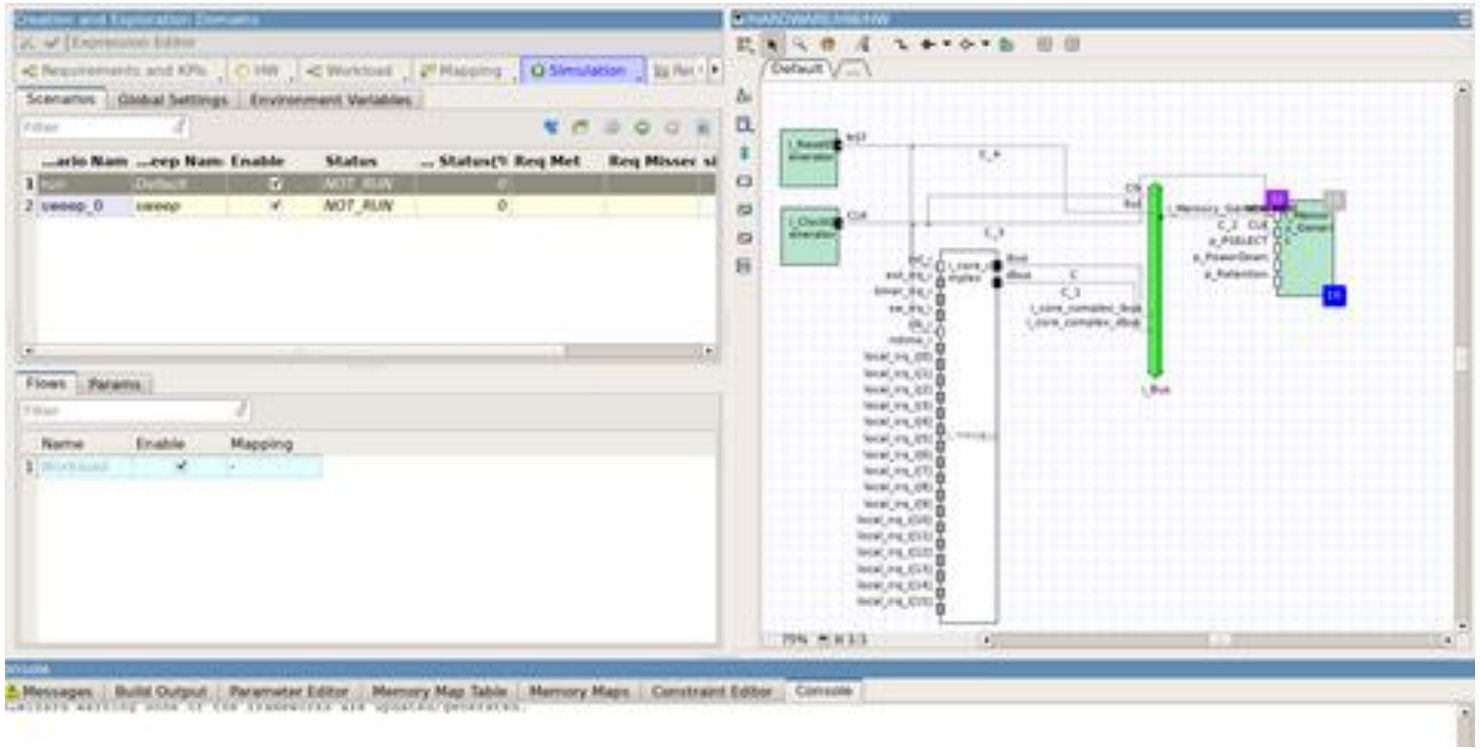
DBT-RISE - Plugins

- Existing Plugin infrastructure
- For example:
  - Instruction Tracing
    - Coverage Visualization with e.g. lcov
    - Profiling with kcachegrind
  - Cycle Annotation
  - Register Dumping





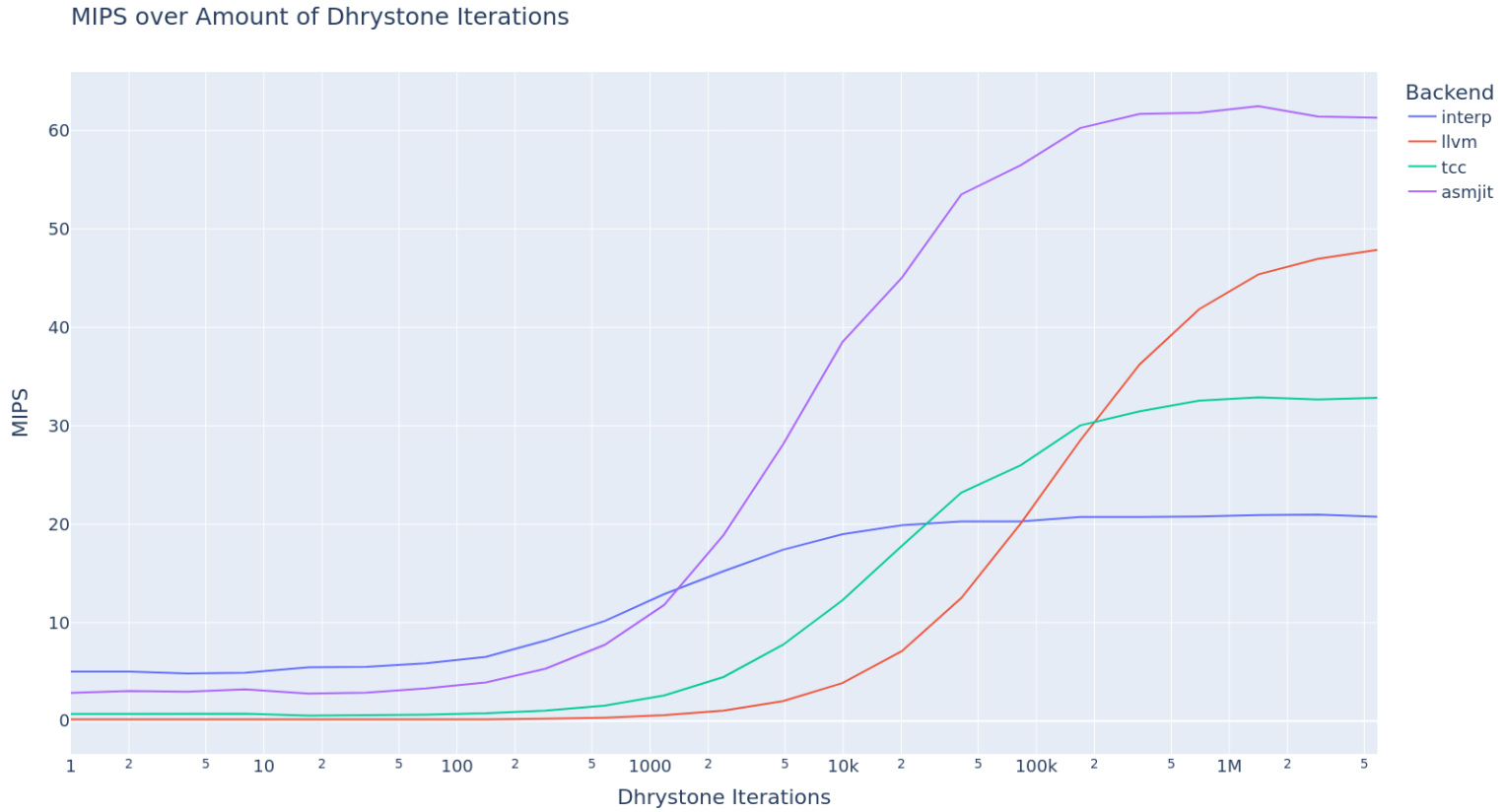
DBT-RISE - Integration



- Works in any TLM2 based tool environment, e.g. Platform Architect
- Plugins allow tailored integration

DBT-RISE - Backends

- Backend defines execution speed
  - Each curve shows a different backend
- Speed is measured as MIPS as function of iterations over benchmark Dhystone
- JIT techniques optimize SW sections which are executed repeatedly



Traffic generators

- Trace based models
  - STL reader
- Traffic profile models
  - Task based modeling
  - Queue based modeling
  - ATP profile reader
- Template based generators
  - Similar to VIP approach

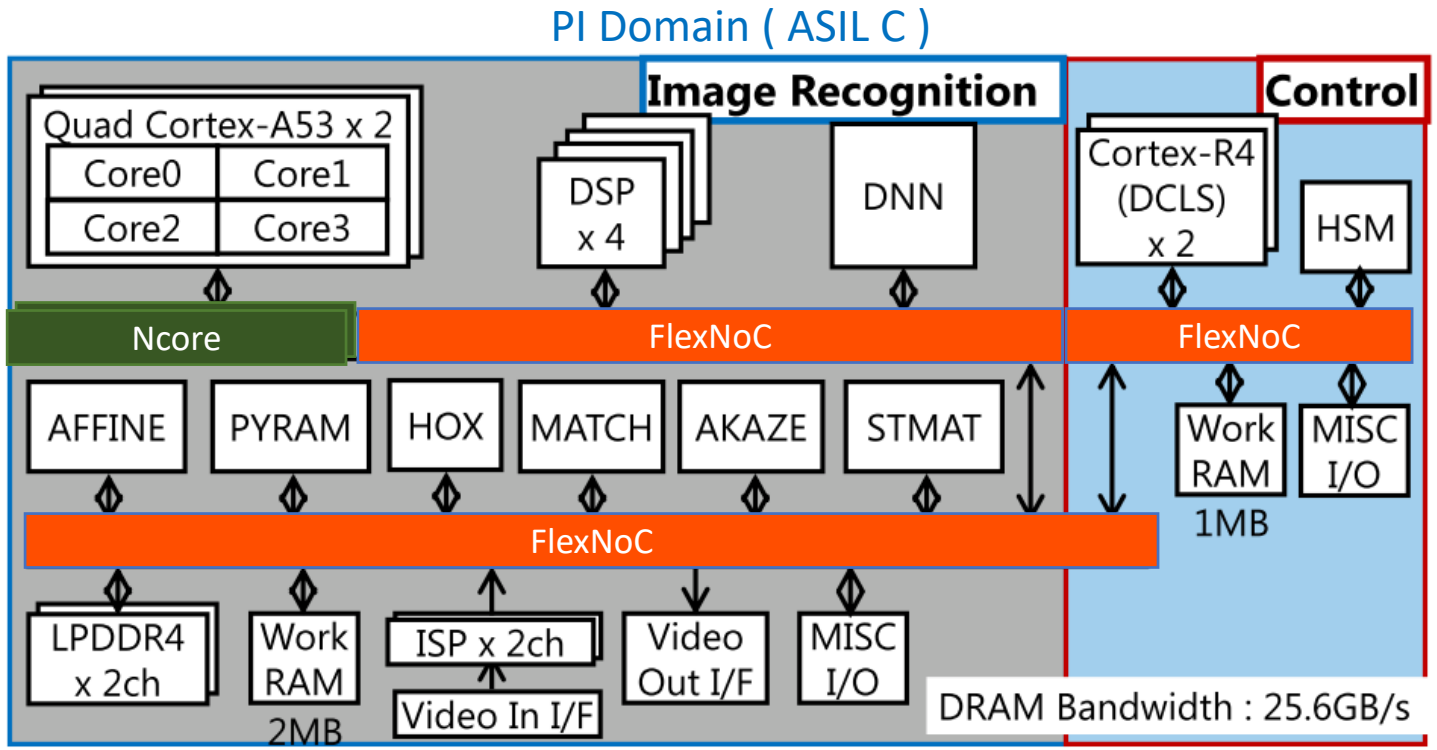
Transport models

- PV model - typically relatively simple
  - Prevent events; DMI; time quantum
  - Registers regions
  - Hooks to functionality like firewalls
- AT model - high impact on latency and BW analysis
- CA model – detailed and slow

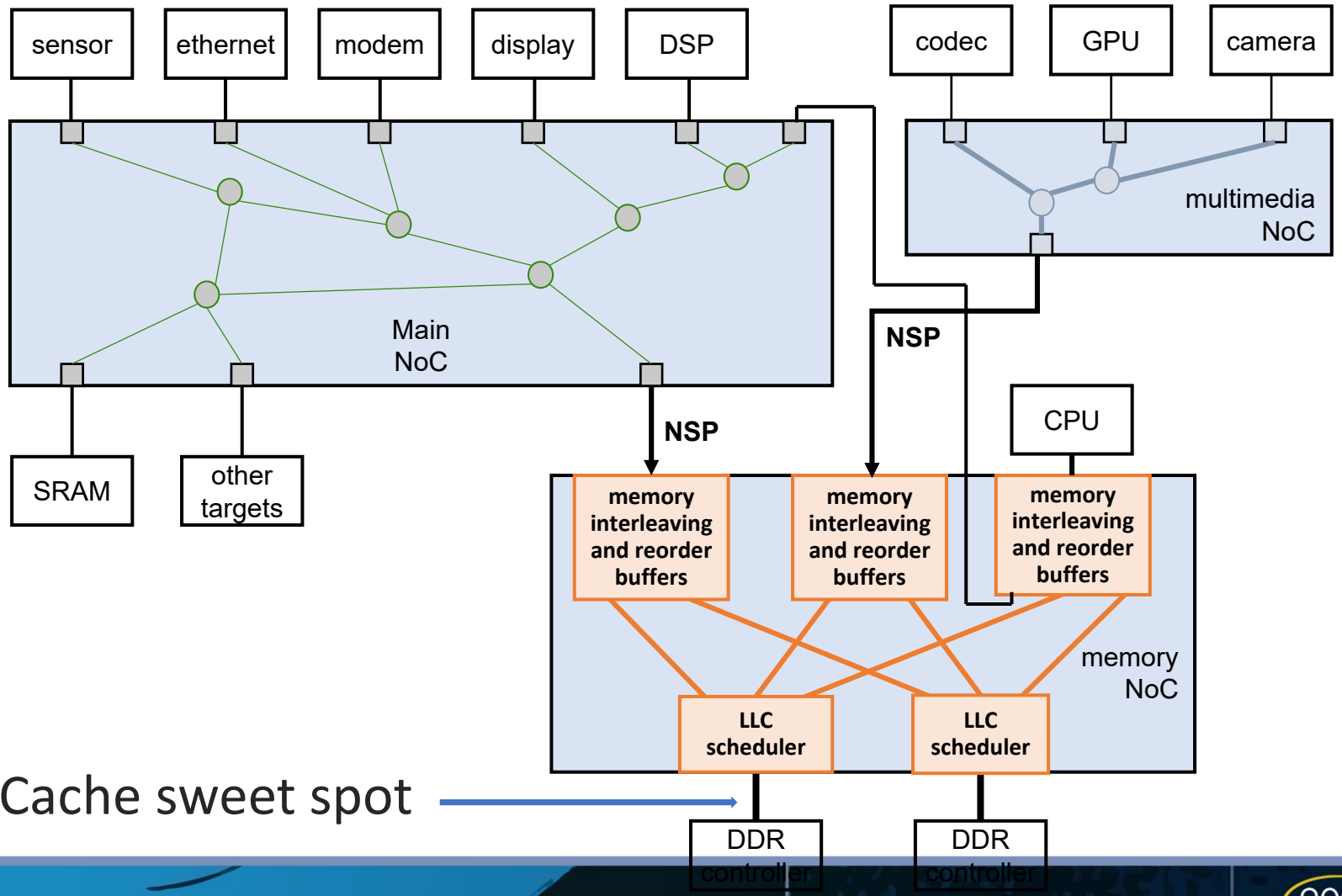
Mix of transport models

NoC Interconnect IP enables low power, high performance and safety

- 1 x Cache Coherent Interconnect
- 8 x Non-coherent Interconnect
  - 8 NoC instances and all of them use Resilience up to unit duplication.
  - Separation for SI (Safety Island) and PI (Processing Island).



Multi-NoC diagram

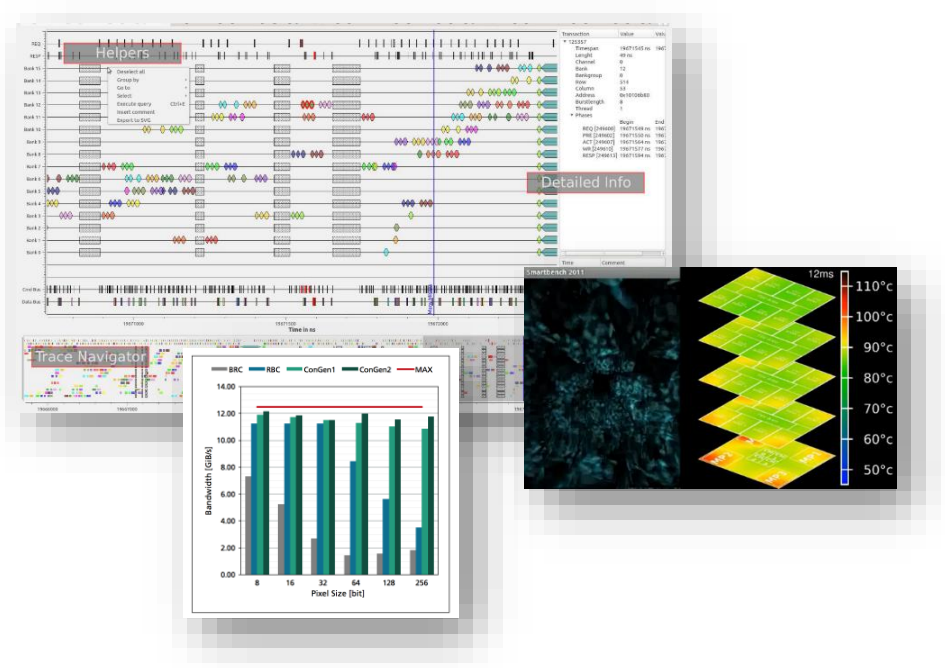


- CodaCache sweet spot

DRAMSys in a Nutshell

Simulation and Design Space Exploration of Modern DRAM-based Memory Systems:

- Which DRAM configuration?
- When to support DDR5 or LPDDR5?
- How to configure the memory controller?
- What is the system-level application behavior?

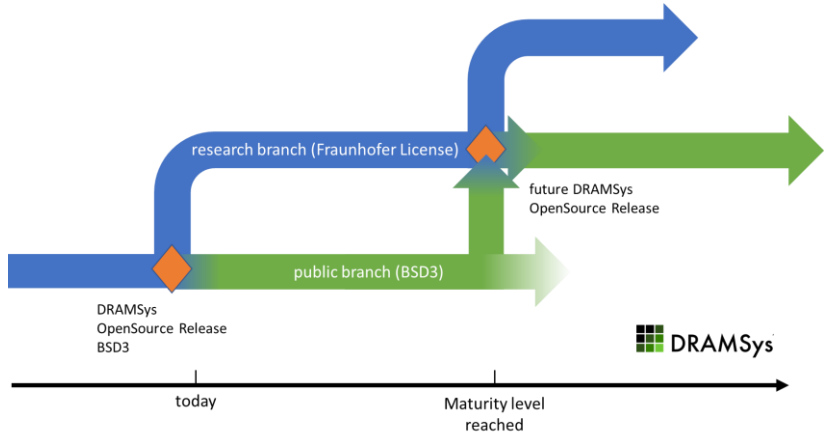


DRAMSys Offers:

- High-speed and flexible models of all standards
- Fast and accurate design space exploration
- Early identification of bottlenecks
- Connection to cores (e.g. SystemC, gem5, …)

DRAMSys Open Source Model

- Open source: DDR3/4, LPDDR4, Wide I/O 1/2, GDDR5/X, GDDR6, and HBM2
- Commercial/academic licenses: DDR5, LPDDR5, HBM3, Trace Analyzer tool
- New standard models will be open-sourced when a level of maturity is reached
- Customer-specific consulting, modifications and developments



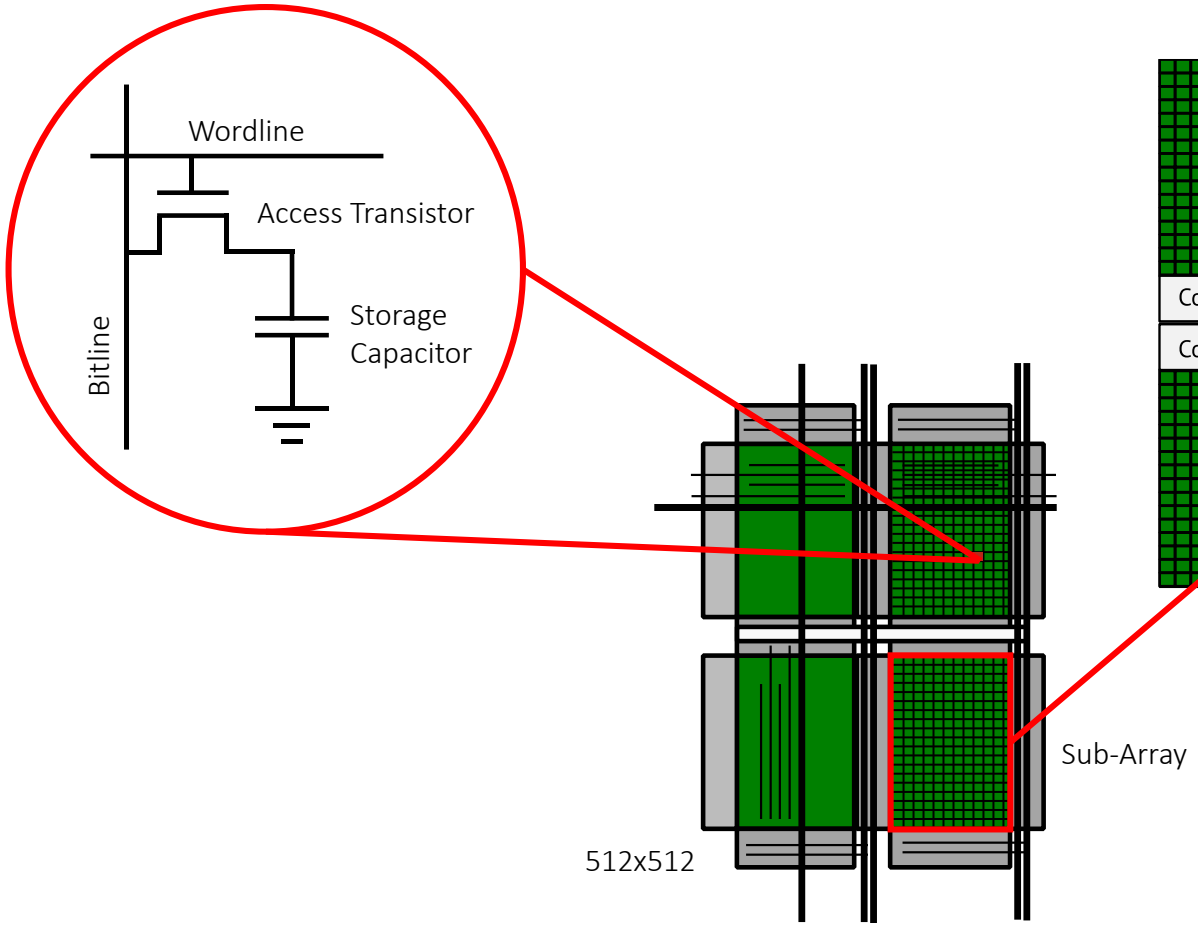
Thanks to our Key Partners:

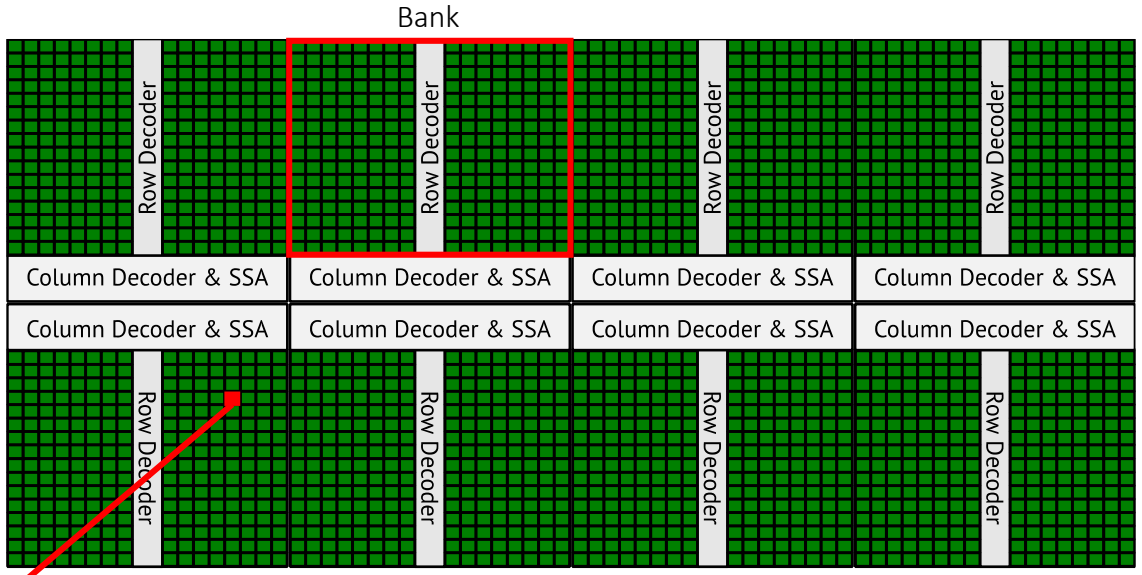
Rambus

Mercedes-Benz

ARTERIS IP

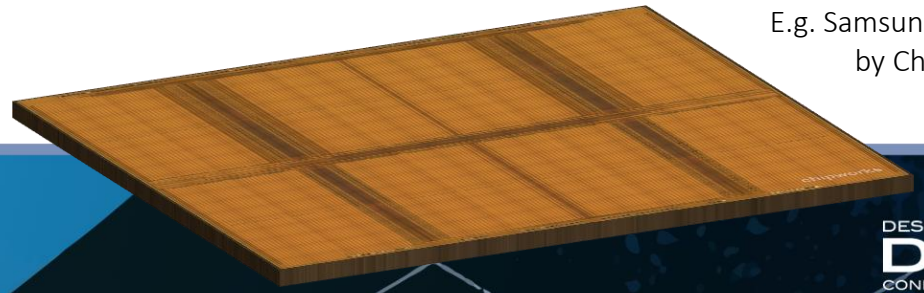
The DRAM Device / Operation



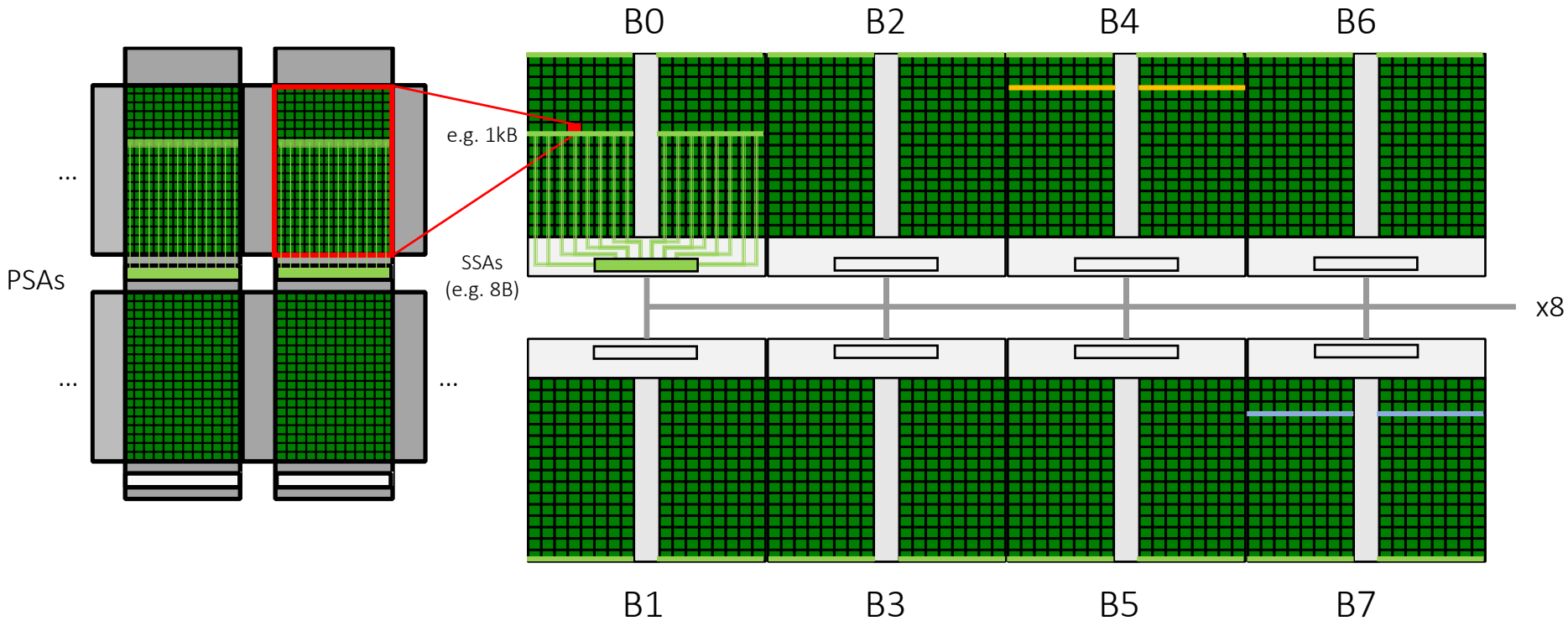


- Using Sub-Arrays for efficient wiring
- Bank parallelism, but banks share data and command bus

E.g. Samsung DDR3,
by Chipworks



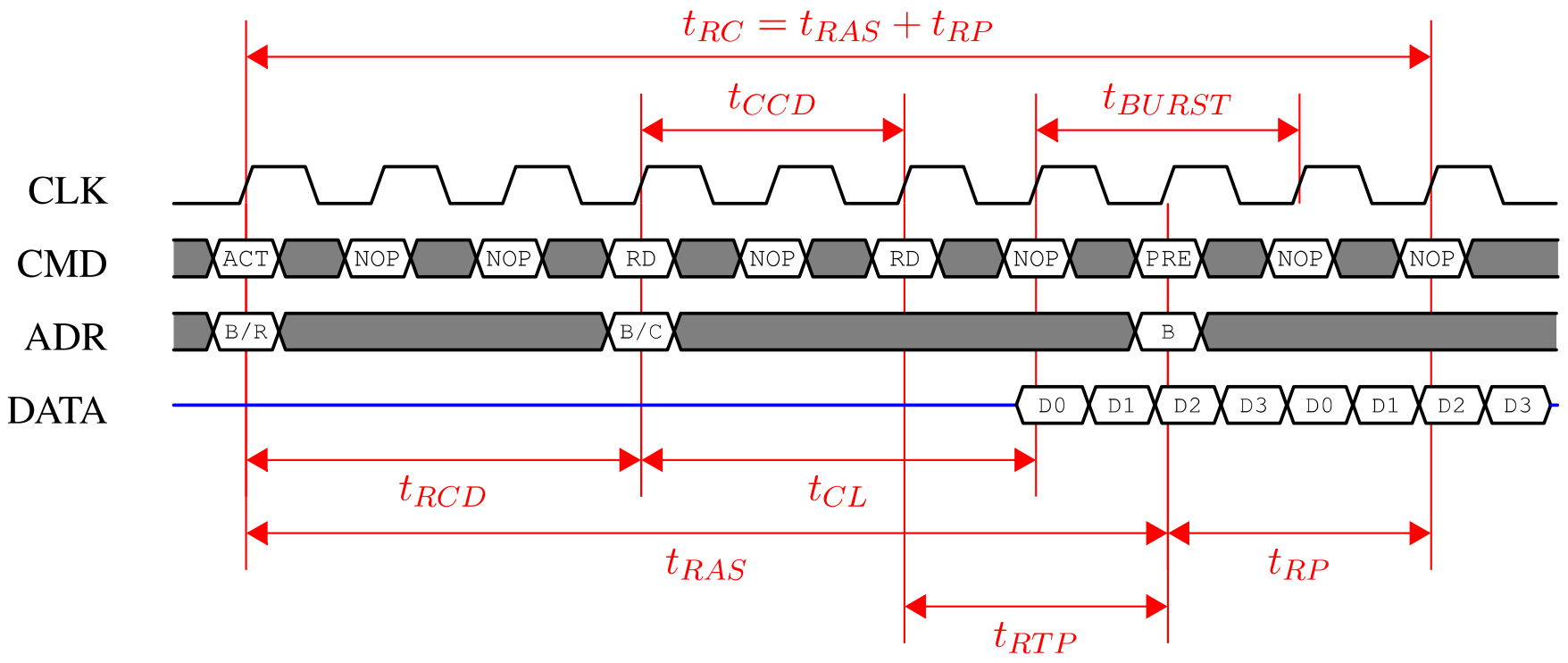
DRAMs Basic Operations



Important DRAM Commands:

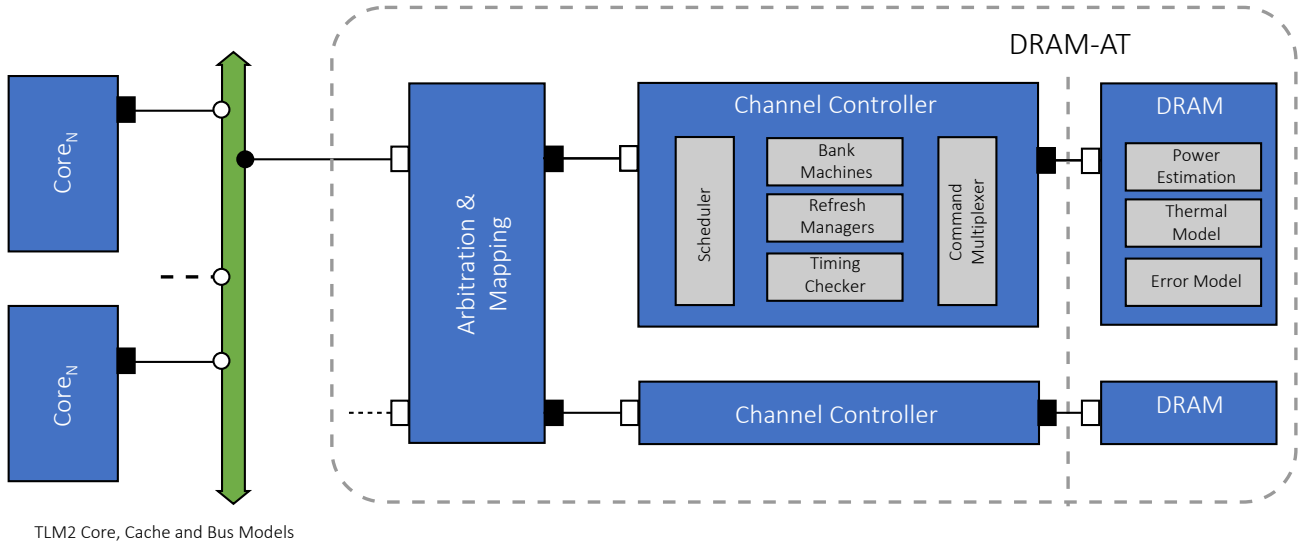
- **ACT**: Activates a specific row in a specific bank (sensing into PSA) [*tRCD*]
- **RD**: Read from activated row (prefetch from PSA to SSA and burst out) [*tCL + tBURST*]
- **PRE**: Precharges set LWL=0 set LBL=VDD/2 [*tRP*]
- **REFA:** DRAM cells are leaky and have to be refreshed [*tREFI & tRFC*]

JEDEC Standard: e.g. Timing Dependencies



Timing dependencies must be fulfilled by the DRAM controller

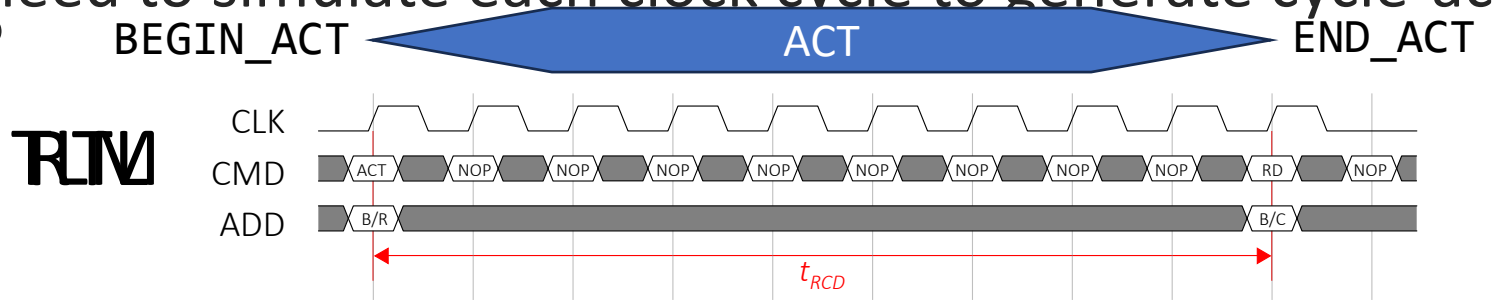
DRAMSys Architecture



- Based on SystemC TLM2, compliant with TLM-AT coding style
- Flexible SW-Architecture to support various JEDEC DRAM standards (e.g., DDR4, LPDDR4, GDDR6, HBM, …)
- For RTL-like accuracy a custom TLM protocol (DRAM-AT) is used

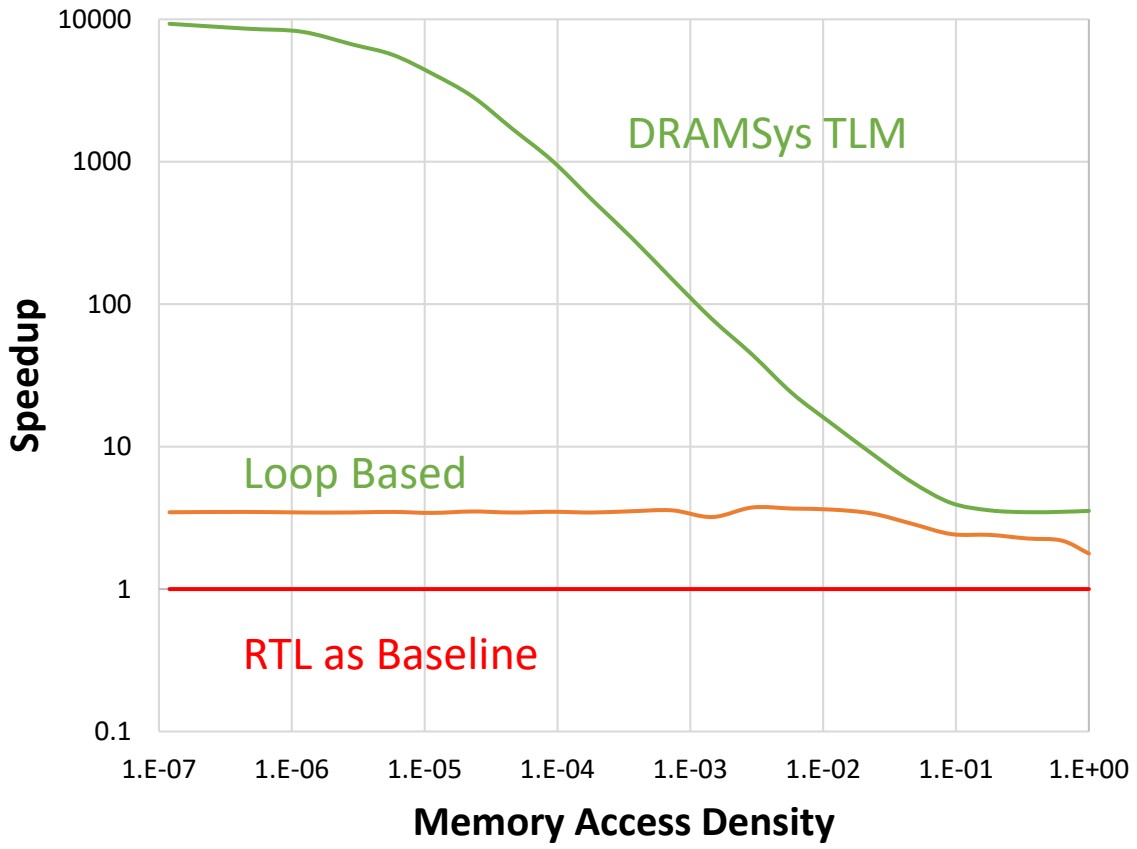
Custom TLM Protocol

- Simulation speed can be increased by reducing the number of events
- Clock signal has the highest event generation rate
- Do we need to simulate each clock cycle to generate cycle-accurate results?



- Simulation of state changes is sufficient, idle clock cycles can be skipped!
  - Large event reduction at low memory access densities
  - No loss of accuracy

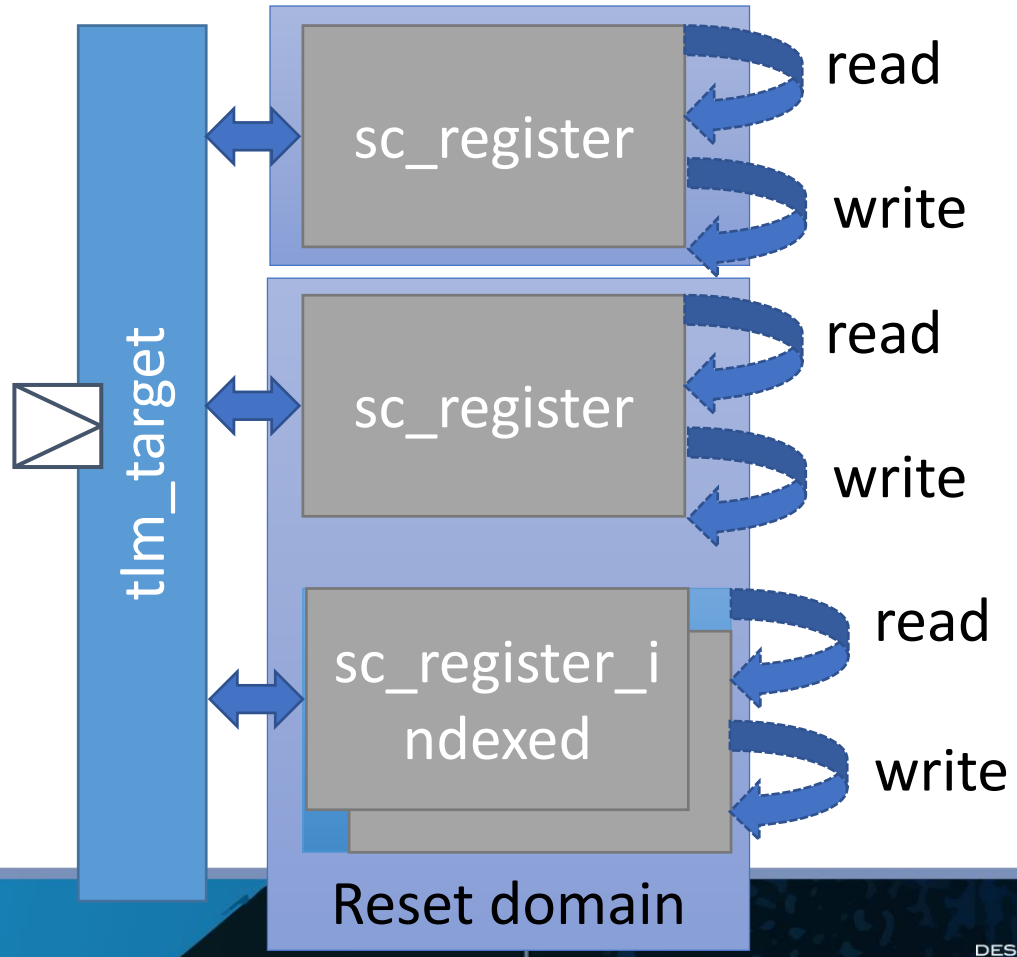
DRAMSys Simulation Speed



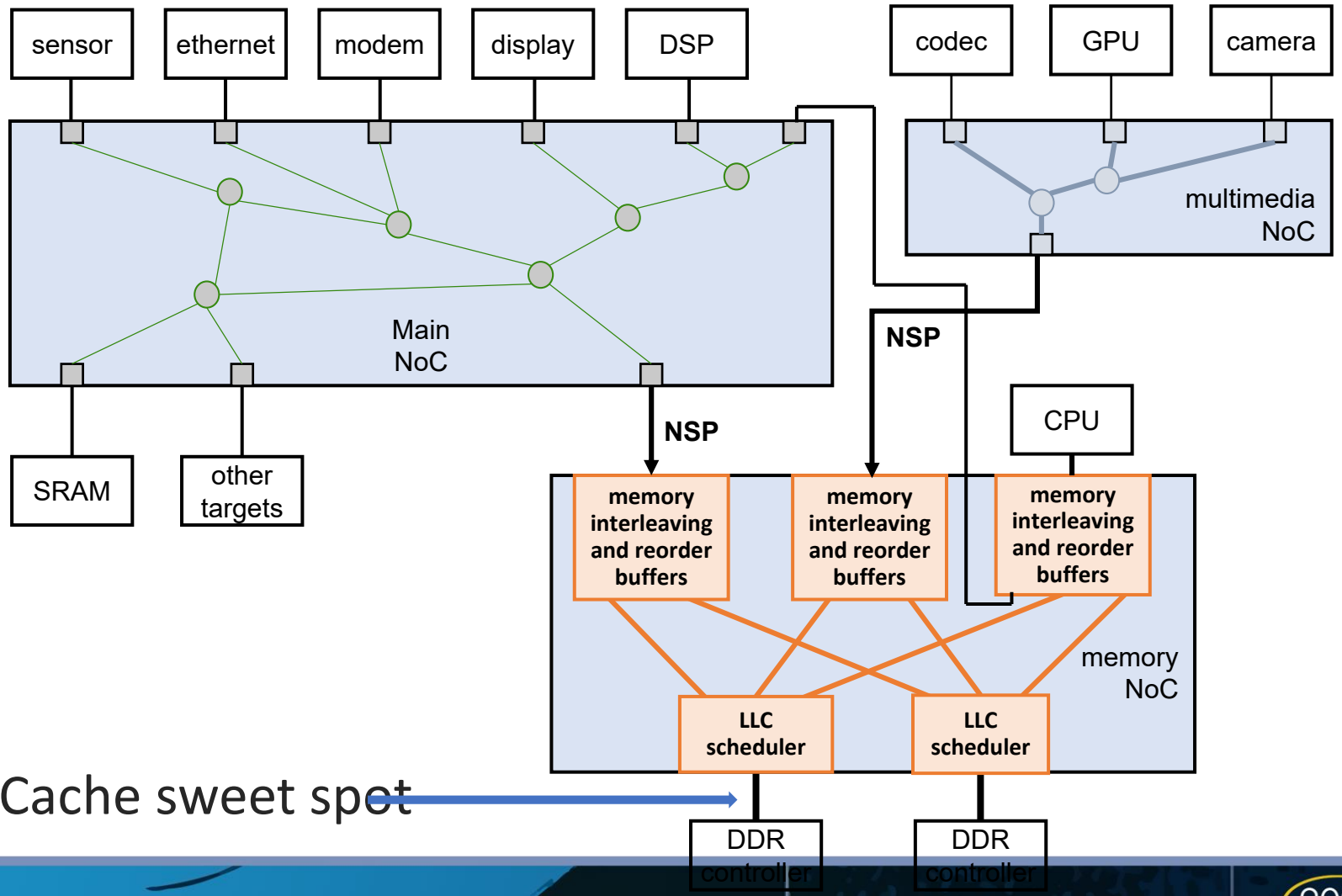
- Simulation of only the important events
- Speedup from 4x to 10.000x depending on trace density
- Average speedups depend on applications
- Typical values: 400x
- 100% RTL Accuracy

Modeling peripherals using SCC

- scc::tlm_target
  - the target socket to connect to
  - Does address decoding and forward to register
- scc::sc_register
  - Takes arbitrary data types
  - Provides means to register callbacks
- scc::sc_register_indexed
  - Array of registers



Multi-NoC diagram



- CodaCache sweet spot

Demo

- Running demonstrations with mix of DBT-RISE + FlexNoC
- Traffic generator + Ncore + FlexNoC + DRAMSys
- Traffic generator + CodaCache + FlexNoC + DRAMSys

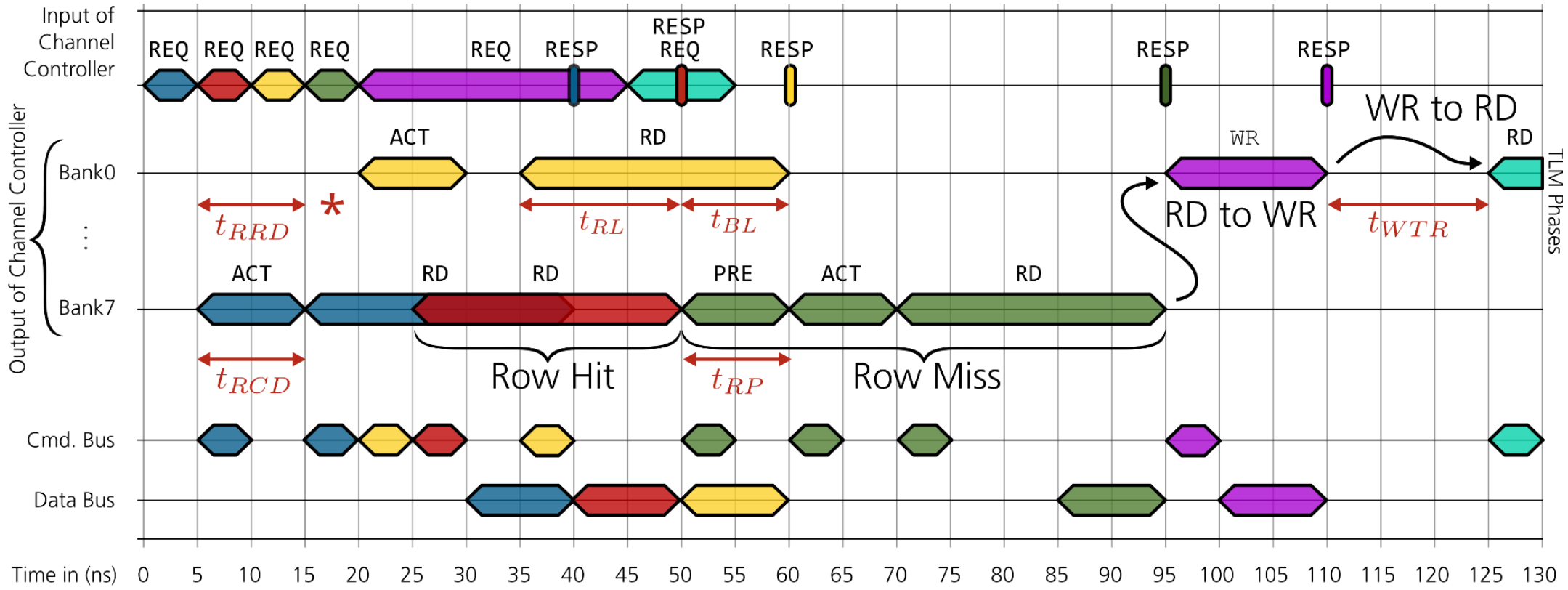
Agenda

- Motivation
- Setting up platforms
- IP as part of platforms
- **Performance analysis and visualization**

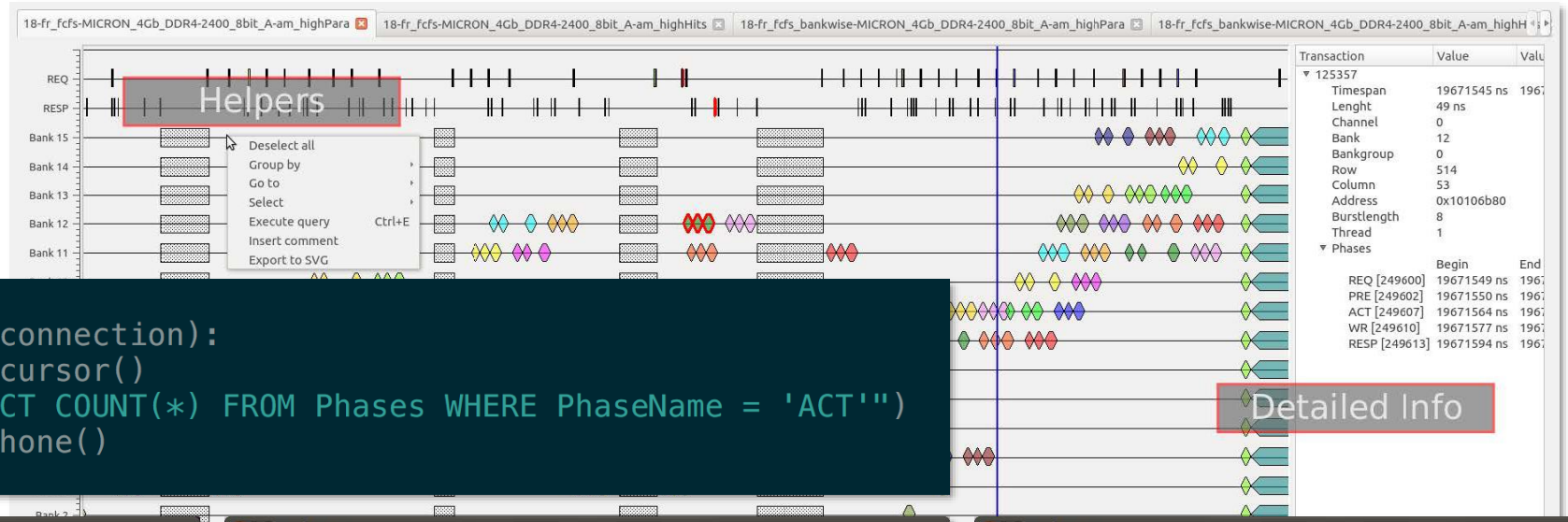
Analysis of results

- The goal of model simulations is the result analysis
- Type of analysis depends on accuracy of model
  - LT model focuses on SW execution (instruction profiling)
  - Latency, bandwidth with AT models
  - Cache and DRAM statistics
  - RTL/verilated simulations are precise, but tracing can be challenging
- Common, open-source formats for tracing are important
  - VCD, FST for signals
  - Transaction tracing
    - No standard format; SCC contains SCV and LWTR formats
- Reuse of frameworks for visualization, post processing and dashboarding
  - Dash, mathplotlib

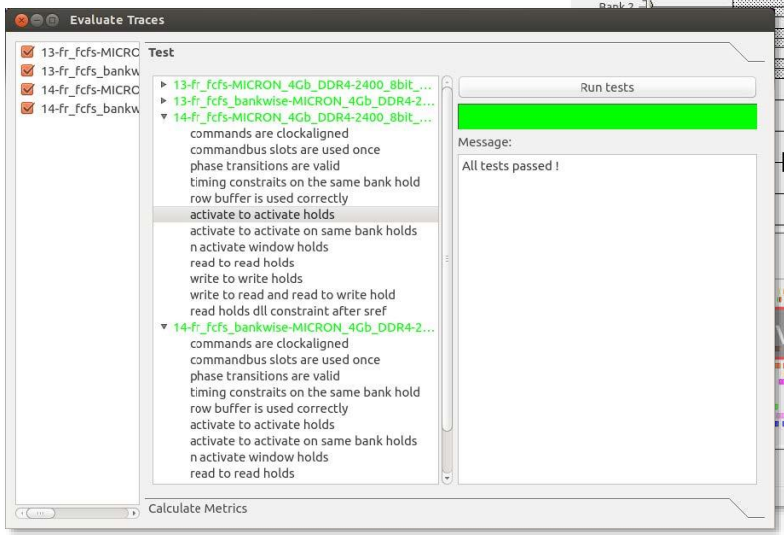
Custom TLM Protocol

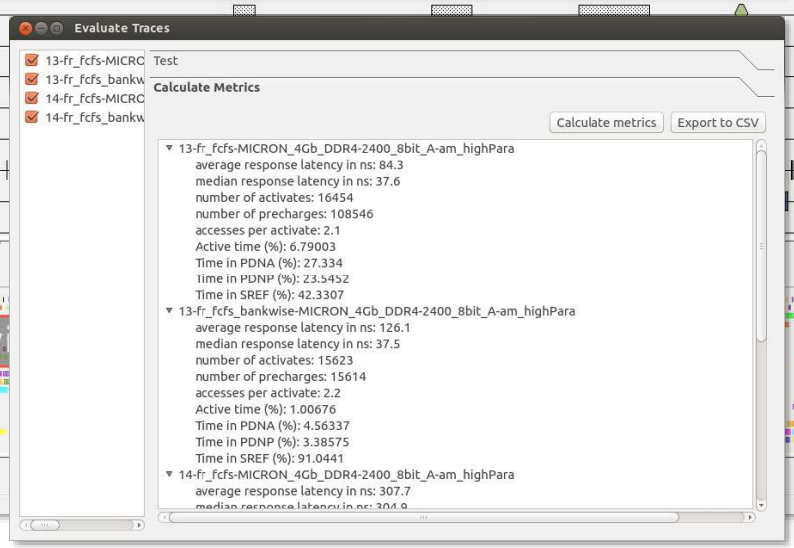


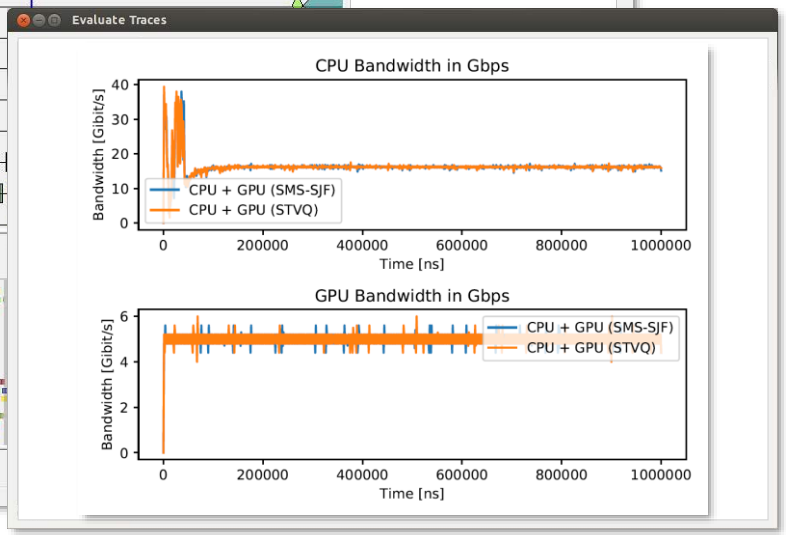
Trace Analyzer




```python
@metric
def number_of_activates(connection):
    cursor = connection.cursor()
    cursor.execute("SELECT COUNT(*) FROM Phases WHERE PhaseName = 'ACT'")
    result = cursor.fetchone()
    return result[0]
```







Waveform based analysis

- Correlation against RTL models for accuracy justification
- Requires performance point comparison
- Often VCD like DB as main result of RTL simulations
  - Waveform Analysis Language (WAL) to the rescue
    - Information under: https://wal-lang.org/
  - Allows post processing from signals to derive performance values or transactions

Process Satisfaction & Target Usage

- Self Checking Testbench
- Scenario also defines real time constraints
- Allow verifying the achieved QoS

Scenario result: /import.GamingQos_architecture.step1_1e_4.000.ScenarioResult

Process | Queue | Scen. Initiator | Scen. Target | Target Model

Column Clr
Row Clr
Helper

	Required Efficiency	Efficiency	Satisfaction	Throughput	State
Process_CPU0	0.0 %	27.1 %	PASS	74.56 MB/s	
Process_CPU1	0.0 %	26.5 %	PASS	72.64 MB/s	
Process_CPU2	0.0 %	25.2 %	PASS	70.08 MB/s	
Process_CPU3	0.0 %	28.0 %	PASS	69.12 MB/s	
Process_DispRd0	100.0 %	69.2 %	FAIL	222.7 MB/s	
Process_DispRd1	100.0 %	69.2 %	FAIL	222.7 MB/s	
Process_DispScan	100.0 %	100.0 %	PASS	80.64 MB/s	
Process_DispWr	100.0 %	100.0 %	PASS	320.6 MB/s	
Process_Eth	0.0 %	100.0 %	PASS	201 MB/s	
Process_Image	70.0 %	36.1 %	FAIL	819.2 MB/s	
Process_Modem_CPU	60.0 %	48.9 %	FAIL	53.12 MB/s	
Process_Modem_Data	100.0 %	42.6 %	FAIL	128 MB/s	
Process_Sata	0.0 %	41.0 %	PASS	92.16 MB/s	

Process | Queue | Scen. Initiator | Scen. Target | Target Model

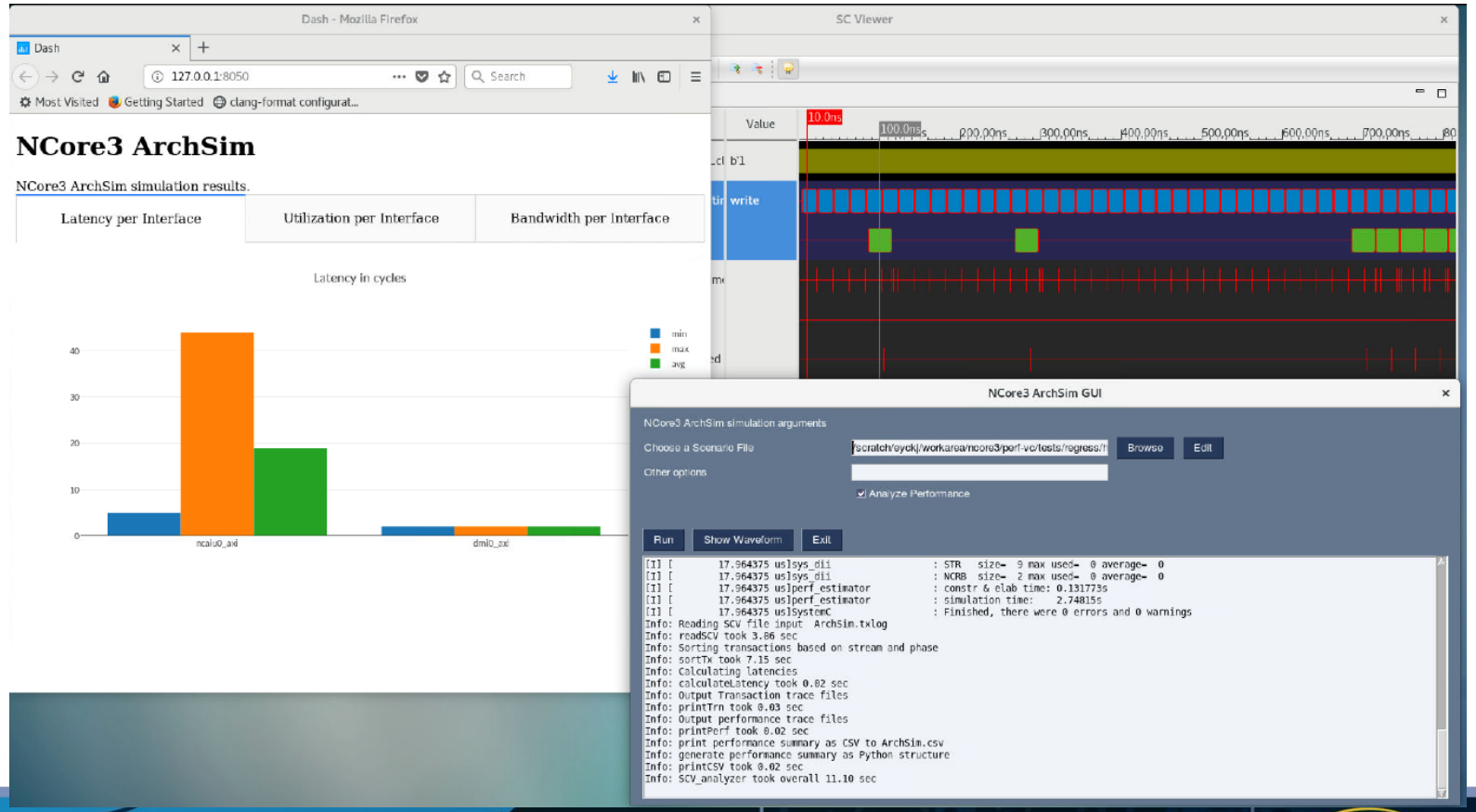
Column Clr
Row Clr
Helper

	Throughput	state	Read	Write	Busy	Wait	Throttle	Idle	Command Level	avg	max	Data Level
DramModel	1.957 GB/s		21.2 %	15.4 %	0.0 %	18.6 %	1.0 %	43.8 %	6	1.464	6	304 B

Source: Arteris FlexNoC, Internal Design Example

Dashboards

- Trace analysis output can be used by open-source visualization tools like dash
- Python libraries allow simple analysis and even simulation control interfaces



Demo

- Running demonstrations of DRAMSys
- FlexNoC Analysis
- Waveforms
- Dashboard

Open-soure offerings I

- SystemC Components Library (SCC)
  https://github.com/Minres/SystemC-Components
- PySysC: Python bindings for SystemC, adopted by Accellera
  https://github.com/Minres/PySysC/
- CoreDSL: a language to describe ISAs for ISS generation and HLS of RTL implementation
  https://minres.github.io/CoreDSL/

Questions

- Finalize slide set with questions slide