Comparison of TLM Modeling Styles Performance and Accuracy

Bill Bunton
LSI Networking Components Group
Austin, Texas

North American SystemC Users Group XVIII
San Francisco, California
June 6, 2013

Agenda

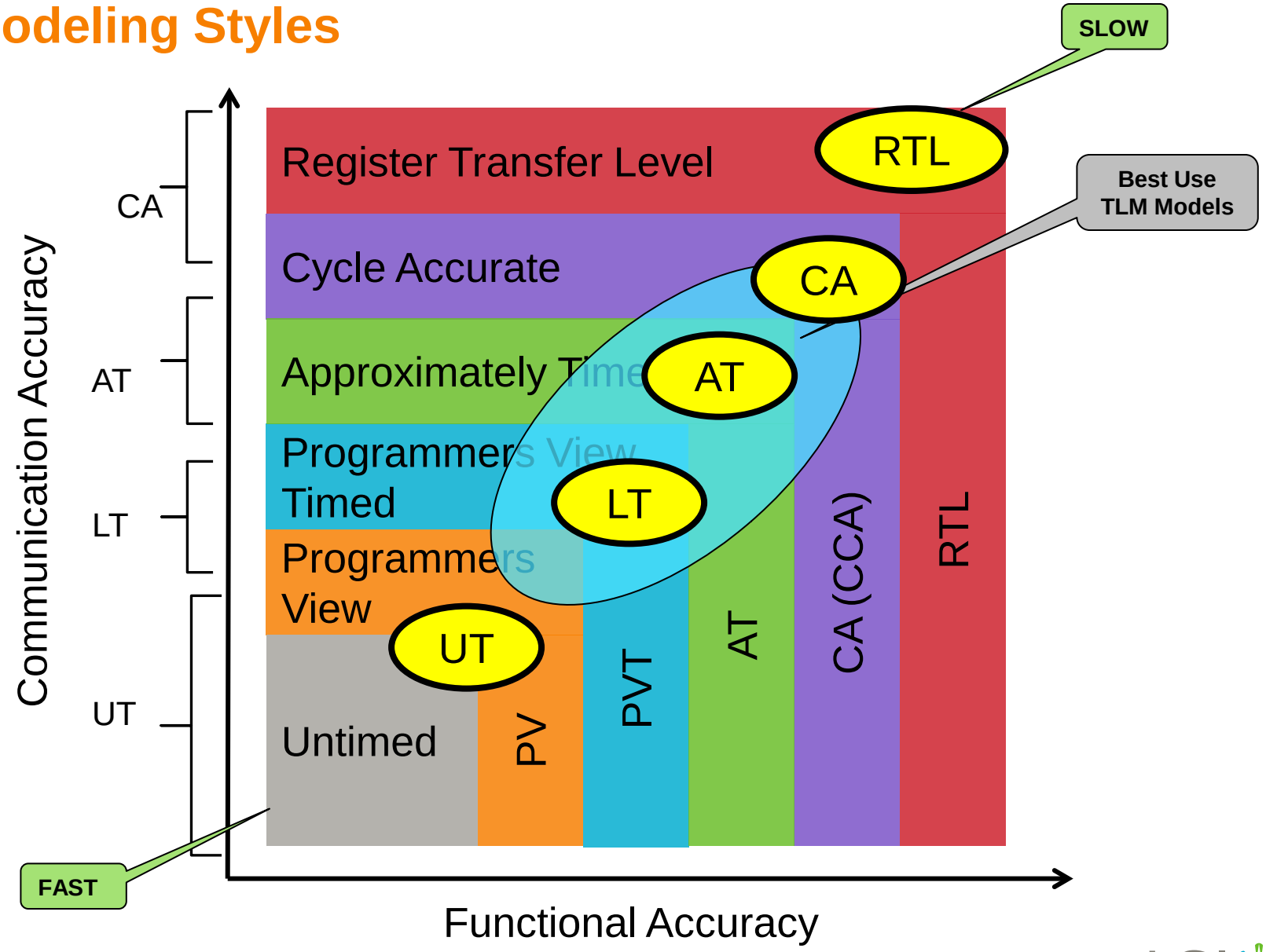
- Overview of Use-Cases and Modeling Styles
- SoC Design Flow
- Modeling Styles Guidelines
- Accuracy Requirements
- Example Models (AT and 3 LT versions)
- Performance Results
- Observations

Use Cases for SystemC Model

- Hardware Architectural Exploration and Analysis
- Virtual Prototype for Software Development
- System Performance Analysis
- Hardware Verification (Co-simulation SystemC and RTL)

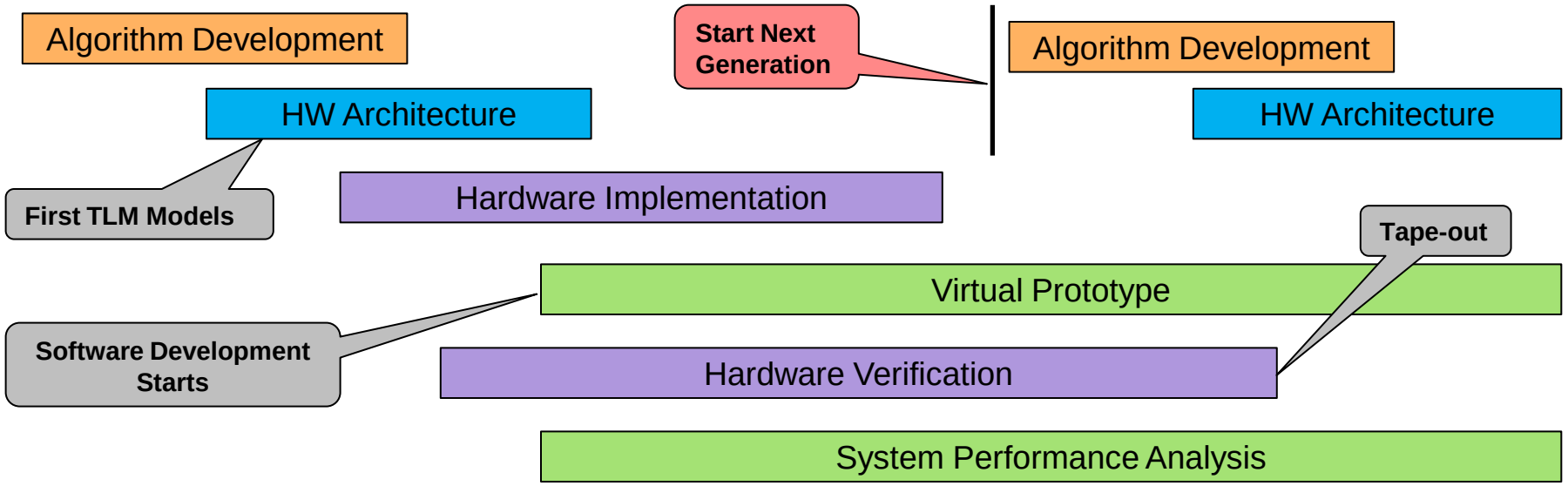
- SystemC TLM Model Attributes
  - Simulation Speed
  - Register Accuracy
  - Register Set Completeness
  - Functional Accuracy
  - Functional Timing
  - Functional Order of Execution
  - Communication Timing
  - Communication Order of Execution

SystemC Model Abstraction Levels or Modeling Styles



SoC Design Flow

▪ Functional and Algorithm Models	MATLAB SystemC
▪ Hardware Architectural Models	SystemC
▪ Hardware Implementation	HDL or SystemC with HLS
▪ Virtual Prototype	SystemC
▪ Hardware Verification	Mixed HDL and SystemC
▪ System Performance Analysis	SystemC



Modeling Styles Guidelines

- High Accuracy Model -- Approximately-timed (AT aka CA and CCA)
  - Processes run in lock-step with simulation time
  - Each transaction has 4 timing points and implement flow-control
  - All timing is a multiple of clock period (no sc_clock)
  - The following internal functions and blocks are modeled
    - Bus widths
    - FIFO
    - Arbiters
    - Pipelines
    - State Machines
    - RAMs
- High Speed Model -- Loosely-timed (LT)
  - Event ordered
  - Each transaction has 2 timing points: *begin* and *end*
  - Not demonstrated - Direct memory interface (DMI) (un-timed transactions)
  - Not demonstrated - Temporal decoupling

Requirements
High Accuracy (AT) and High Speed (LT) Models

Model Attributes	High Accuracy (AT)	High Speed (LT)
Simulation Speed	Don't Care	Fast Fast Fast
Register accuracy	Bit Accurate	Bit Accurate
Register set completeness	All Used by SW	All Used by SW
Functional accuracy	>90%	>90%
Functional timing	>90%	Best Effort (But Fast)
Functional order	>90%	Don't Care
Communication timing	>90%	Best Effort (But Fast)
Communication order	>90%	Don't Care

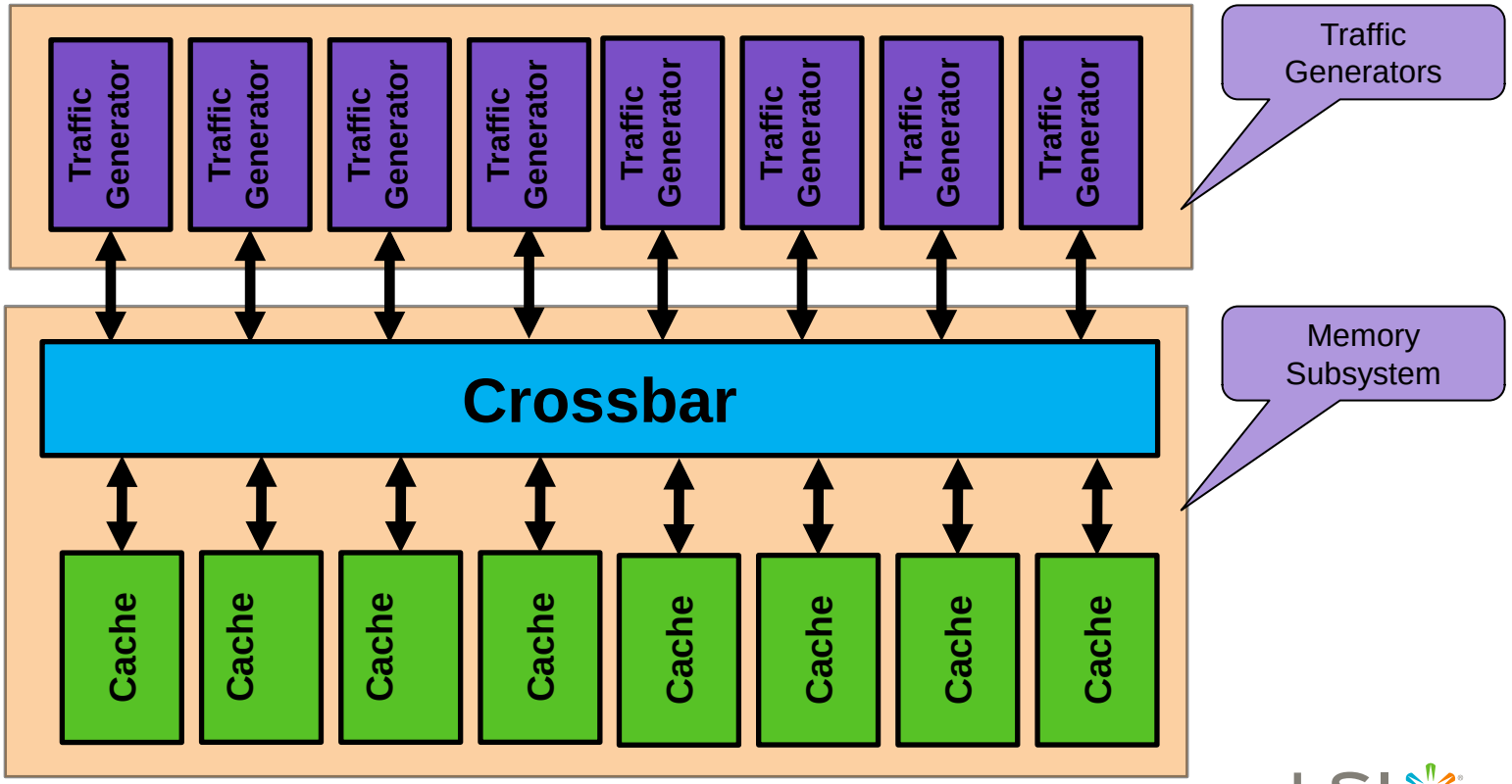
Common Requirements

Demonstration Plan

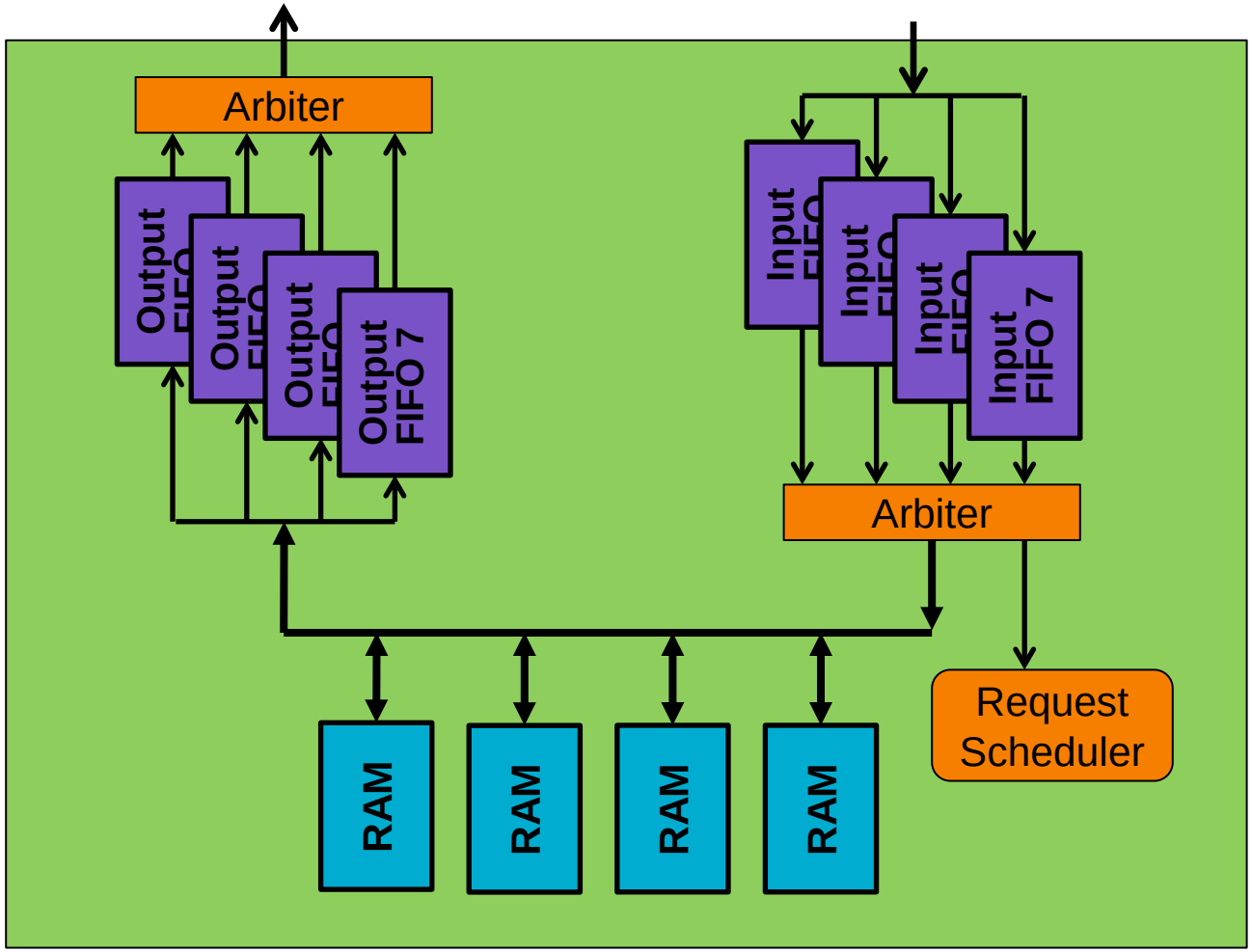
- Assume the existing model is complete and is High Accuracy
  - Traffic Generators
  - Simplified version of the ACP3448 Memory Subsystem
  - Excluded control and status registers (CSR) from test
- Configure the Traffic Generators to:
  - Generate a uniform request load
  - Four test runs per model configurations
    - 1, 2, 4, and 8 Generators enabled
  - Run each configuration of 4ms of simulated time
- Record execution time (CPU time)
- Record number of transactions completed
- Test four model configurations without changes to the Traffic Generators
  - Existing High Accuracy model
  - Simplified Crossbar
  - Simplified Cache and Crossbar
  - Replace Cache and Crossbar (One Block Implementation)

Demonstration Model
A Simplified System Memory Performance Model

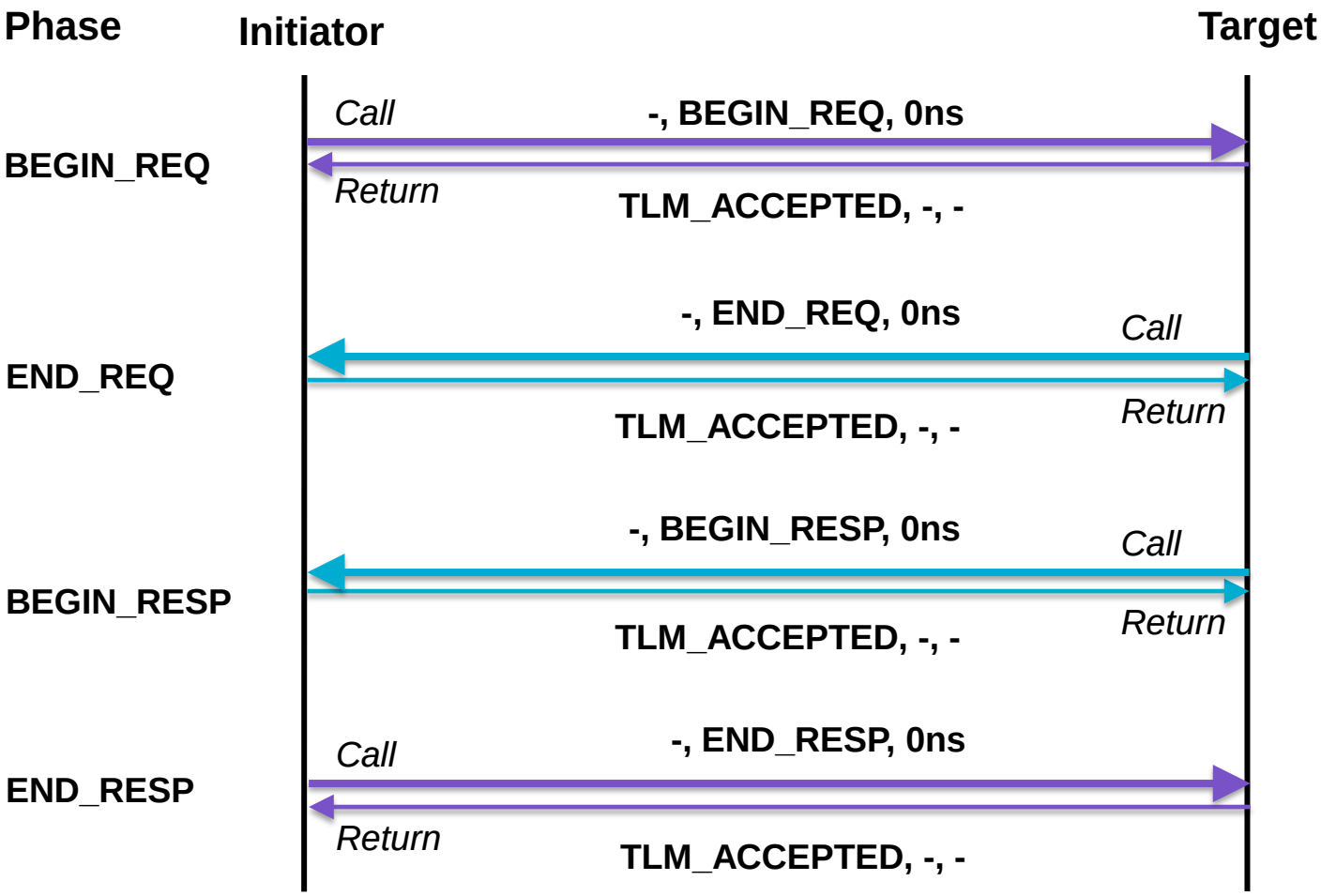
- Two top-level blocks
  - Traffic Generators and test control
  - Simplified version of the ACP3448 Memory Subsystem (High Accuracy)
    - Highly simplified System Cache model (all request modeled and cache hits)
    - DDR3 controllers and memory omitted



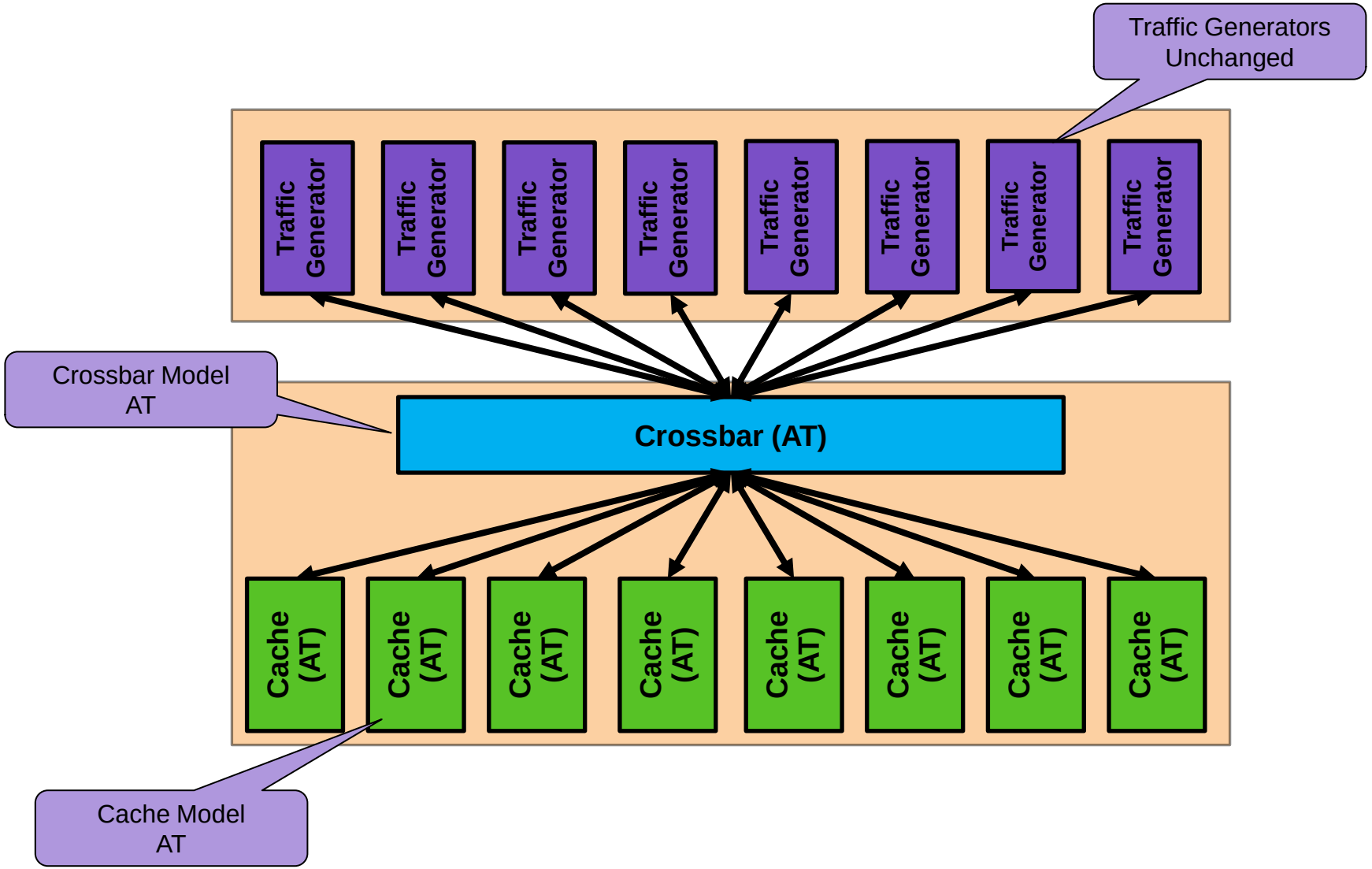
High Accuracy Cache Block Diagram (AT)



Non-Blocking Transport Using Backward Path

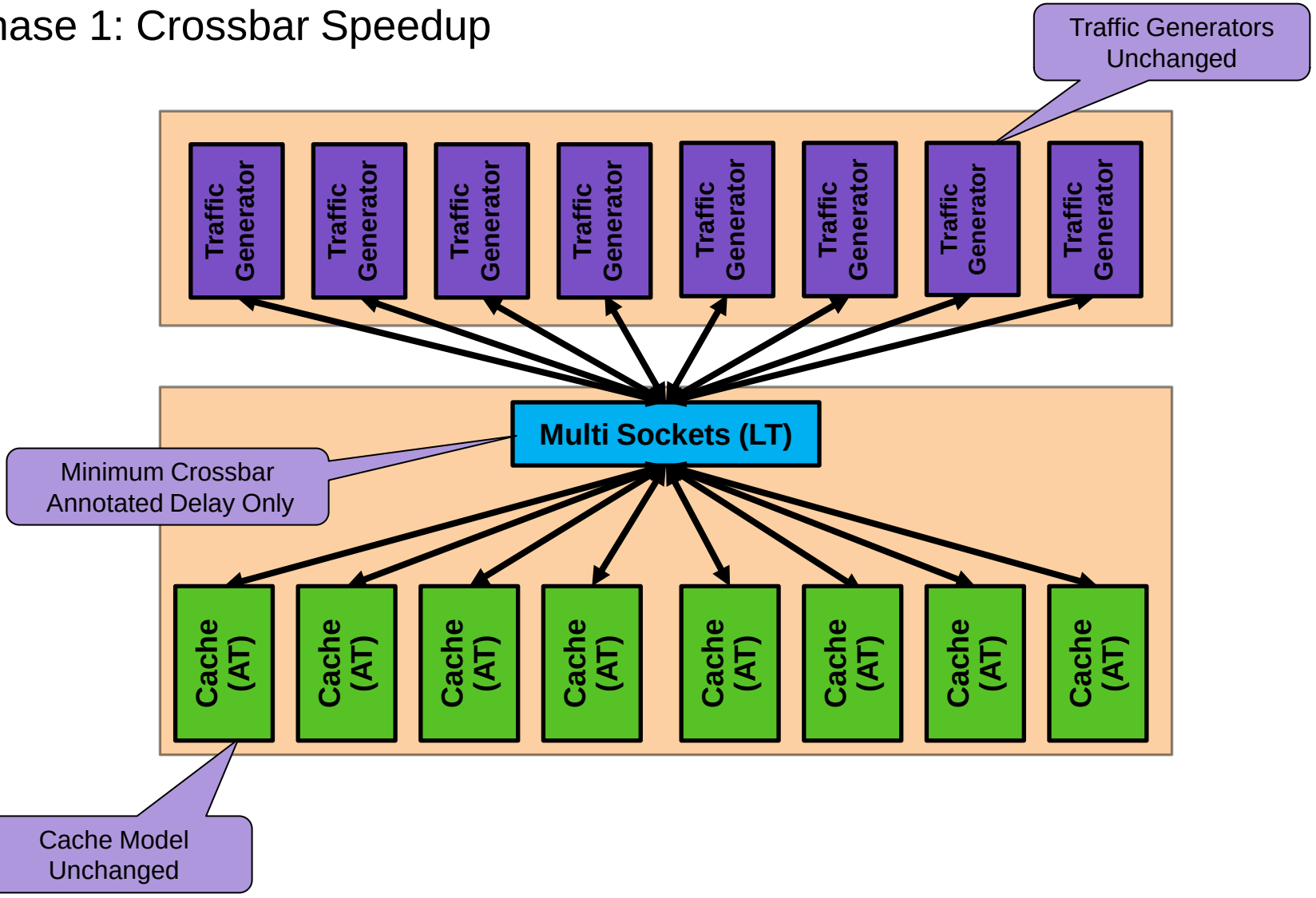


Existing High Accuracy System Memory Model (AT)



High Speed System Memory Model (LT v1)

Phase 1: Crossbar Speedup



Multi-Socket LT Crossbar

Forward Path

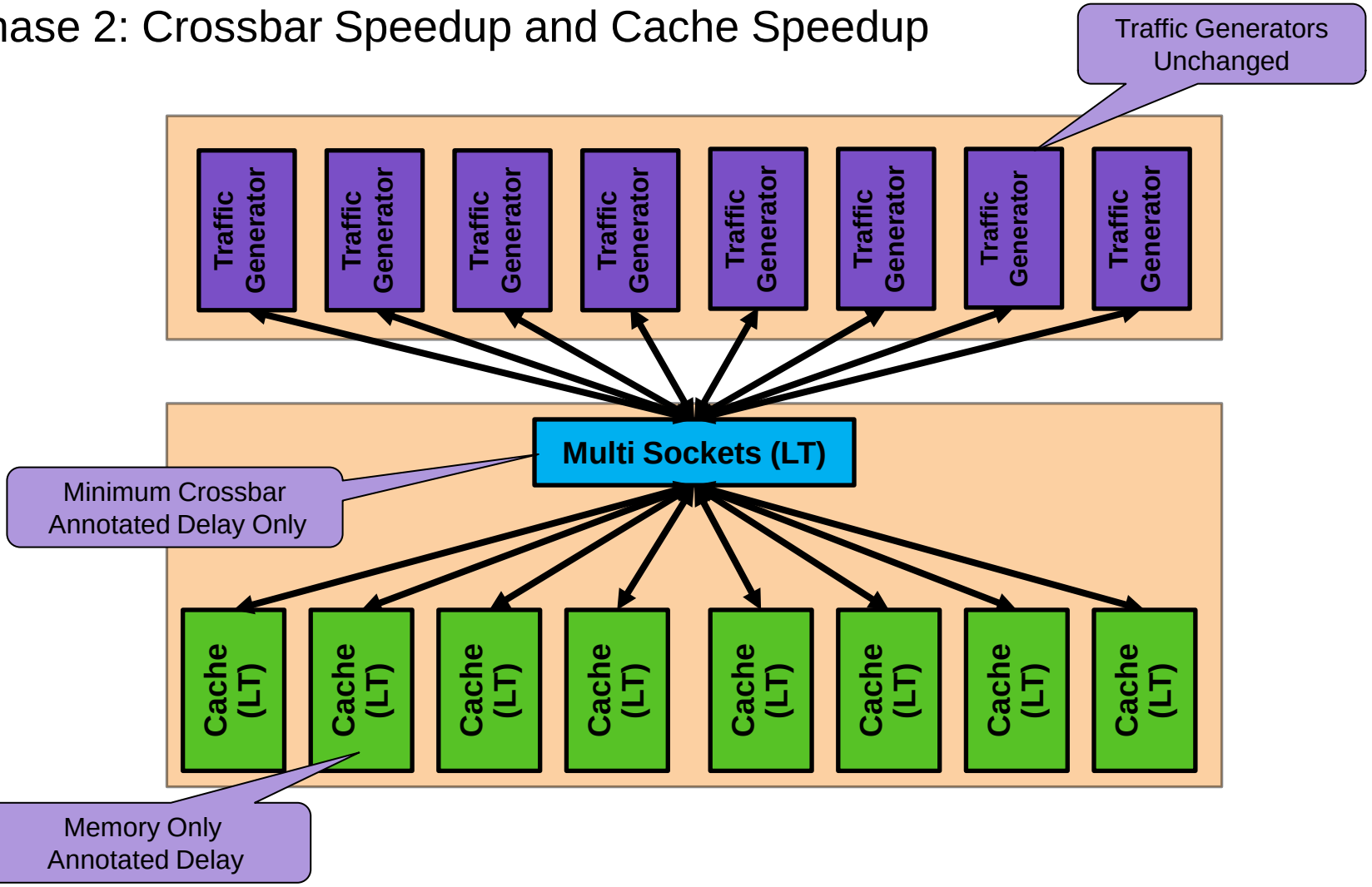
```
tlm::tlm_sync_enum NMSI_XBar_LT::nb_transport_fw_in
( int                          socket_idx
, NMSI::NMSI_Payload&          payload
, NMSI::NMSI_phase&            phase
, sc_core::sc_time&            delay_time
 ){

  payload.set_master_id( socket_idx );

  unsigned int socket_idx_fw = payload.get_slave_id();


  delay_time    = delay_time + (Average_Xbar_delay * core_clk);

  return_status = initiator_socket[socket_idx_fw]->
     nb_transport_fw(payload, phase, delay_time);

  return return_status ;
}
```

Backward Path

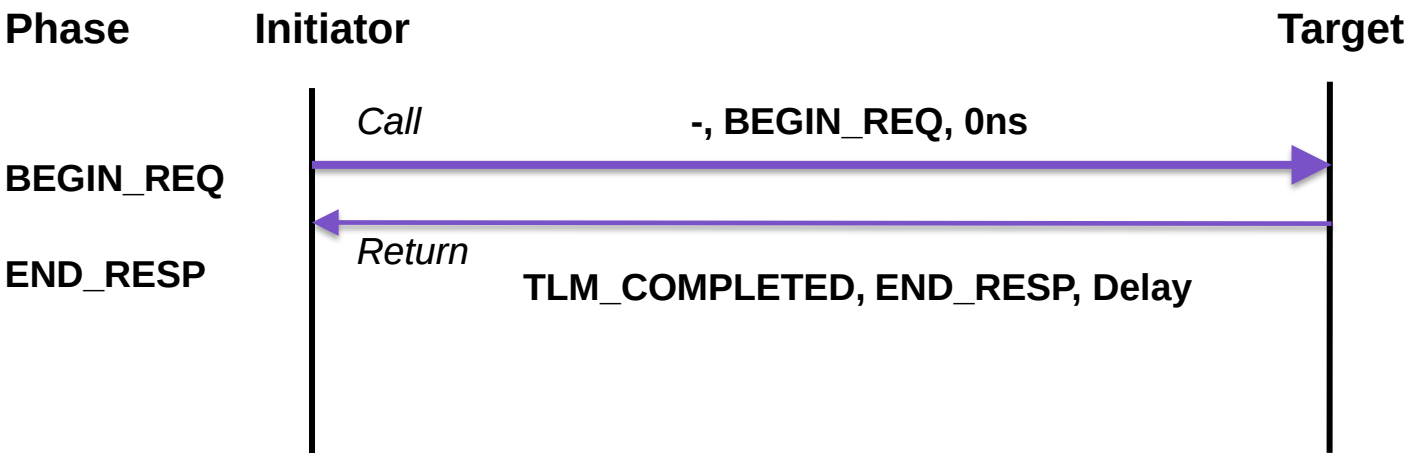
```
tlm::tlm_sync_enum  NMSI_XBar_LT::nb_transport_bw_in
( int                          socket_idx
, NMSI::NMSI_Payload&          payload
, NMSI::NMSI_phase&            phase
, sc_core::sc_time&            delay_time
){


  unsigned int socket_idx_bw = payload.get_master_id();


delay_time    = delay_time + (Average_Xbar_delay * core_clk);

  return_status = target_socket[socket_idx_bw]->
     nb_transport_bw(payload, phase, delay_time);

  return return_status ;
}
```

High Speed System Memory Model (LT v2)

Phase 2: Crossbar Speedup and Cache Speedup



Non-Blocking Transport Using Early Completion

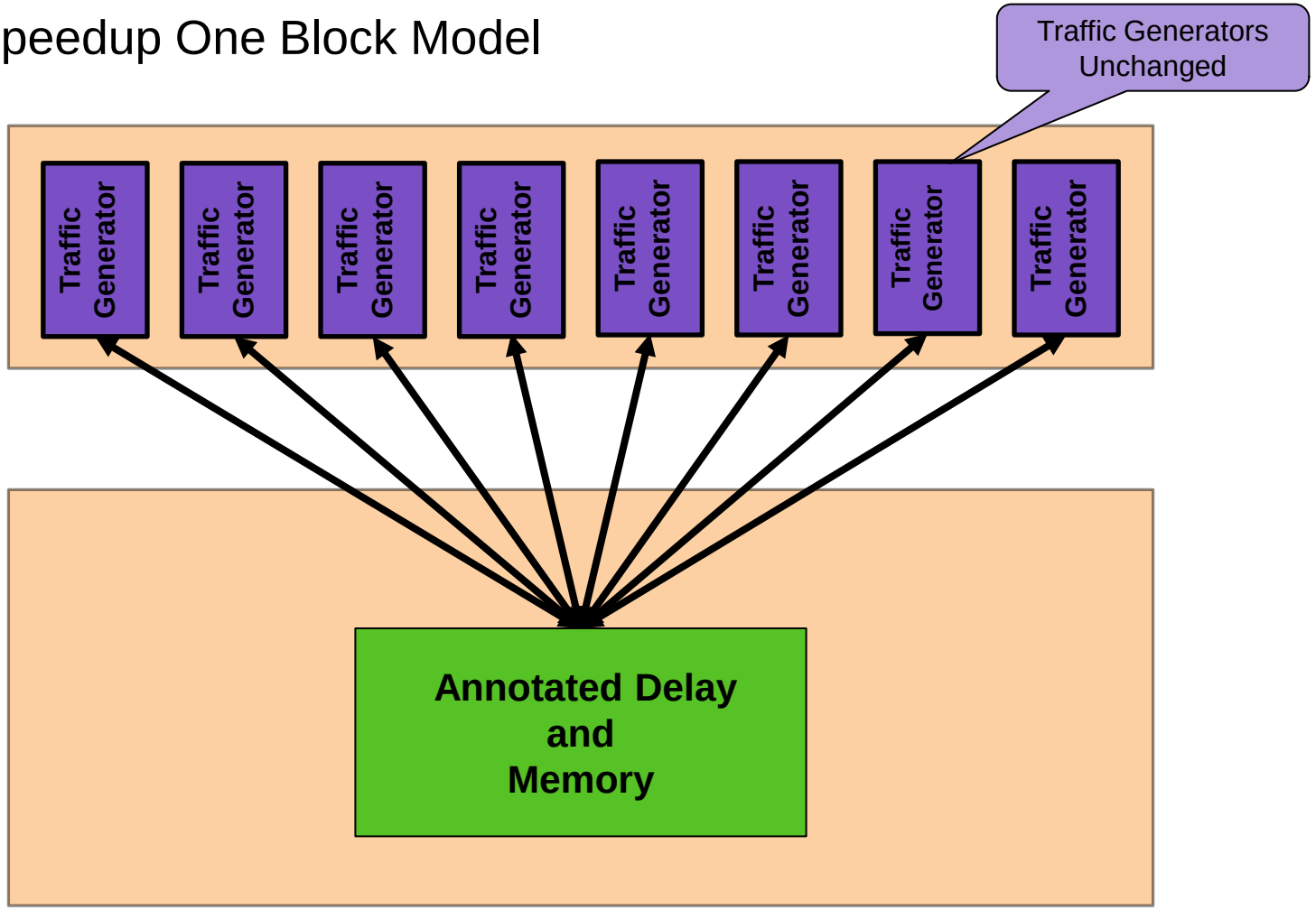


LT Cache

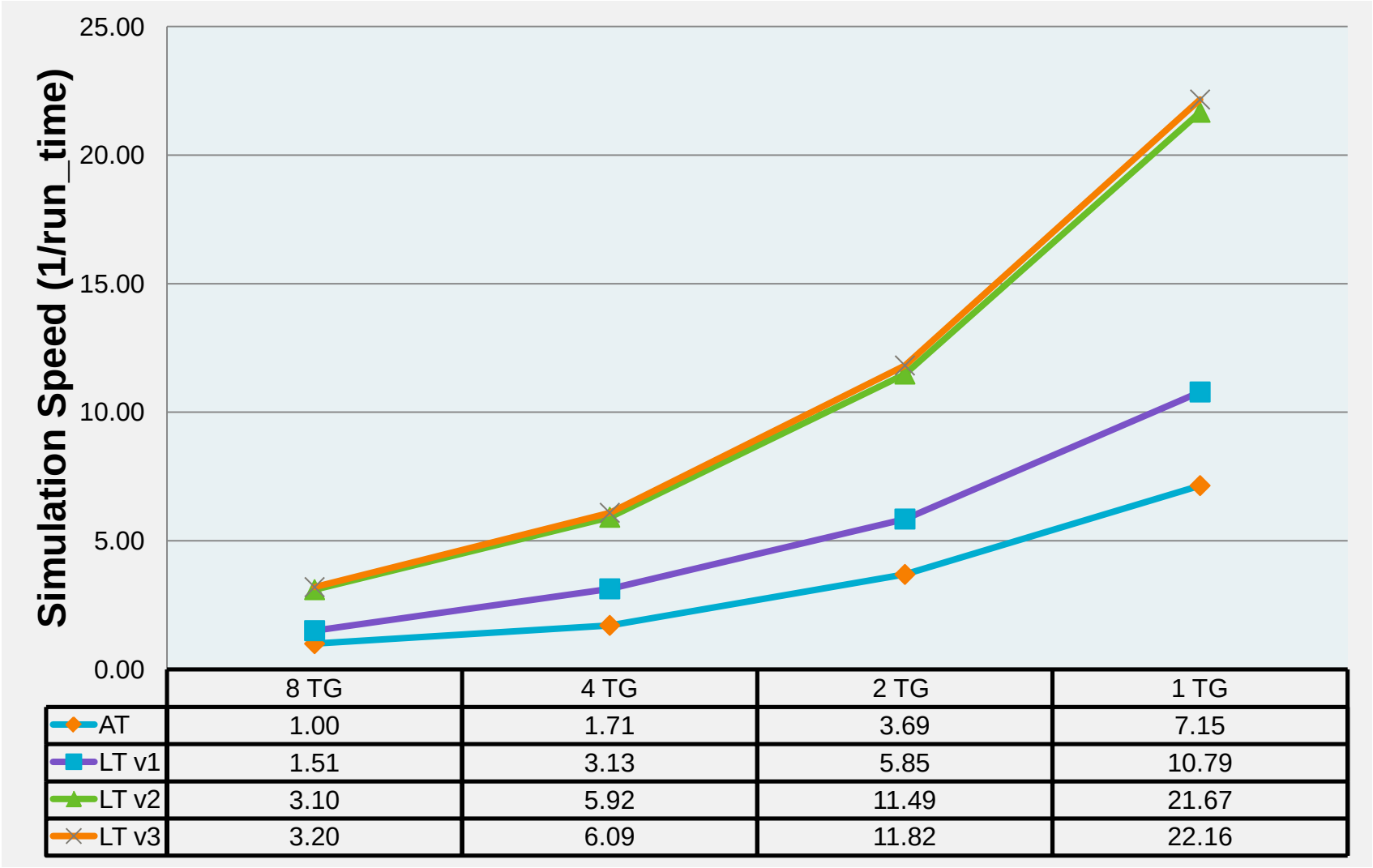
```
tlm::tlm_sync_enum Comb_X_C_LT::NMSI_nb_transport_fw
( int                              socket_idx
 , NMSI::NMSI_Payload&     payload_ref
 , NMSI::NMSI_phase&       phase
 , sc_core::sc_time&       delay
 ){
  switch (phase) {
    case NMSI::BEGIN_RdReq: {
      Req_ptr             = &payload_ref;
      Beat_cnt            = Req_ptr->get_Beat_cnt();
      storage_address = (Req_ptr->get_NMSI_address() & Address_Mask_const);
      Comb_X_C_LT::Storage_RW
        ( false                    // write = false
        , storage_address     // address
        , Beat_cnt                // beat count 16 byte per beat
        , Req_ptr->get_data_ptr()
        );
      delay             = delay + ((Avg_Rd_Delay_clks + Beat_cnt) * core_clk);
      return_status =  tlm::TLM_COMPLETED;
      break;
    }
    case NMSI::BEGIN_WrReq: {
       Code omitted see read case above
     }
  return return_status;
```

High Speed System Memory Model (LT v3)

Phase 3: Speedup One Block Model



Modeling Style Performance



	8 TG	4 TG	2 TG	1 TG
AT	1.00	1.71	3.69	7.15
LT v1	1.51	3.13	5.85	10.79
LT v2	3.10	5.92	11.49	21.67
LT v3	3.20	6.09	11.82	22.16

Observations

- Loosely-Timed (LT) models are clearly faster than more detailed Approximately-Timed (AT) models
- Large LT performance increases are possible without changing the structure of the system model
- Using annotated time can preserve model timing accuracy, LT models are easily adjusted to match average timing of AT or RTL models
- Temporal decoupling was not implemented and was not benchmarked
- Direct Memory Interface (DMI) was not benchmarked
- Implementing Quality Models is an iterative process
  - Correlation with the best available model
  - Completed RTL is the best and final reference model for correlation
  - Properly correlated models are ready when next project starts
- A complete IP package should include both an LT and AT model
  - The models should meet generally accepted model accuracy
  - All accuracy exception should be clearly documented


LSI
Storage. Networking. Accelerated.™