codesign environment

Schedule

- 1. Introduction
- 2. System Modeling Language: System C *
- 3. HW/SW Cosimulation *
- 4. C-based Design *
- 5. Data-flow Model and SW Synthesis
- 6. HW and Interface Synthesis
  (Midterm)
- 7. Models of Computation
- 8. Model based Design of Embedded SW
- 9. Design Space Exploration
  (Final Exam)
  (Term Project)

Reference

- **TLM Cosimulation**
  - CoWare ConvergenSC / ARM MaxSim
- **RTL Cosimulation**
  - Mentor Graphics Seamless CVE
- **Virtual Synchronization**
  - [3] Dohyung Kim, Chae-Eun Rhee, and Soonhoi Ha, "Combined Data-driven and Event-driven Scheduling Technique for Fast Distributed Cosimulation," IEEE Transactions on Very large Scale Integration(VLSI) Systems Vol. 10 pp 672-679 October 2002
  - [4] Youngmin Yi, Dohyung Kim, and Soonhoi Ha, "Fast and Accurate Cosimulation of MPSoC Using Trace-Driven Virtual Synchronization", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems Accepted in 2007

codesign environment

Contents

- Introduction
- RTL Cosimulation
- Co-simulation Performance Analysis
- Making Co-simulation Faster
- Virtual Synchronization

Validation

- **Validation Methods**
  - Emulation: fast, functional validation
  - Prototyping: more accurate, but time consuming and expensive
  - Simulation: **Co-simulation**
    - *concurrent software and hardware modules*
  - Formal verification
    - *Limited*
- **Validation Object**
  - Functional simulation
  - (Logic simulation)
  - Timing- accurate simulation: statistical vs. cycle-level vs. circuit level

HW/SW Cosimulation

- **Virtual-prototyping the target system with simulation models of components**

component simulator for uP1

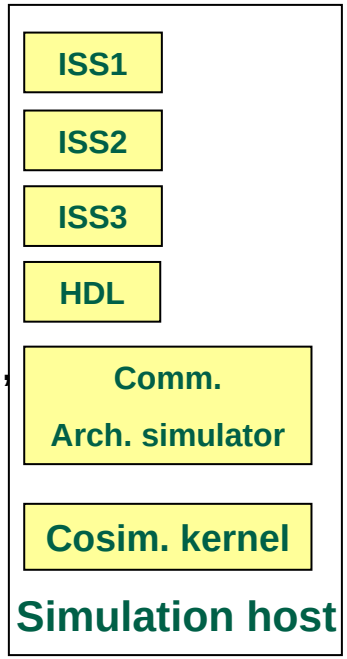
component simulator for uP2

component simulator for uP3

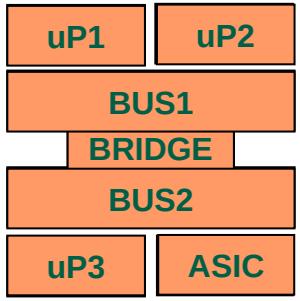
component simulator for ASIC

comm. arch. simulator for BUS1, BUS2, BRIDGE

Virtual Prototype of a System



Real Prototype of a System

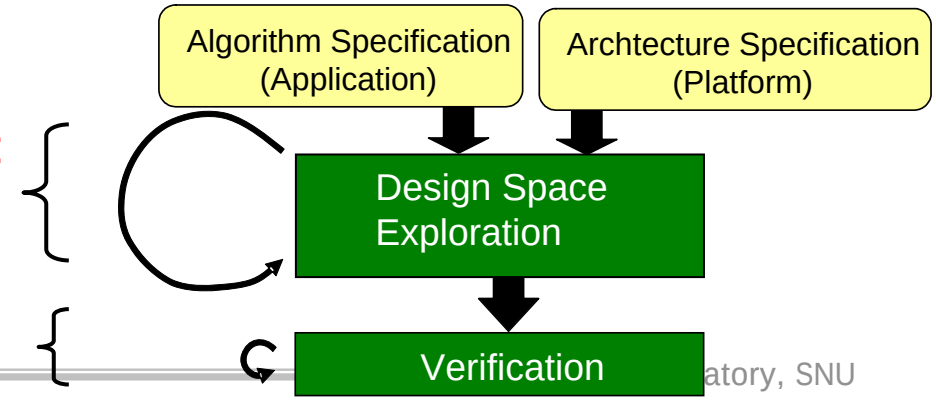


Objectives

- **Virtual prototyping for SW development**
  - Need fast simulation. Usually allow 20-30% accuracy error.
- **System performance estimation for design space exploration**
  - For design space exploration
  - Need reasonably fast cosimulation, sacrificing some accuracy
- **System verification before implementation**
  - Need cycle-accurate simulation for timing verification

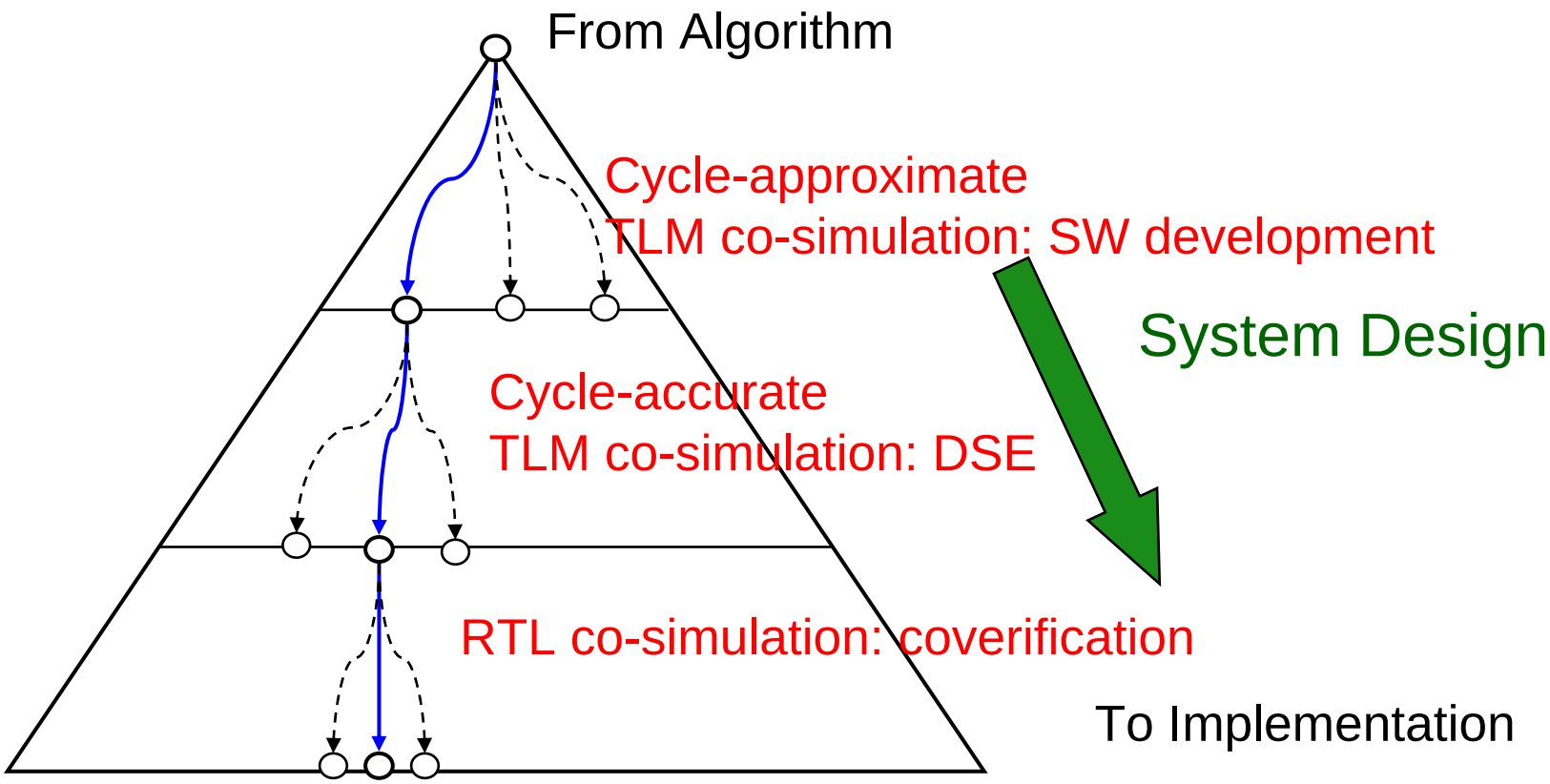
Estimate the system performance :
Fast and reasonably accurate

Verify the funtional and timing correctness :
Accurate and reasonably fast



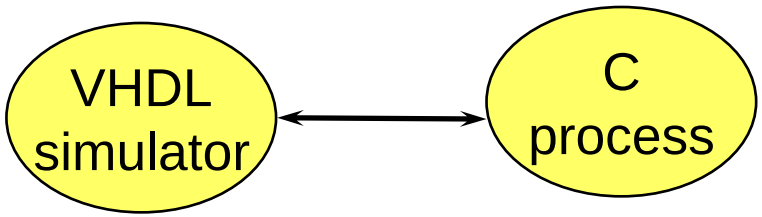
Design Flow and HW/SW Cosimulation

- **Tradeoff between speed and accuracy**



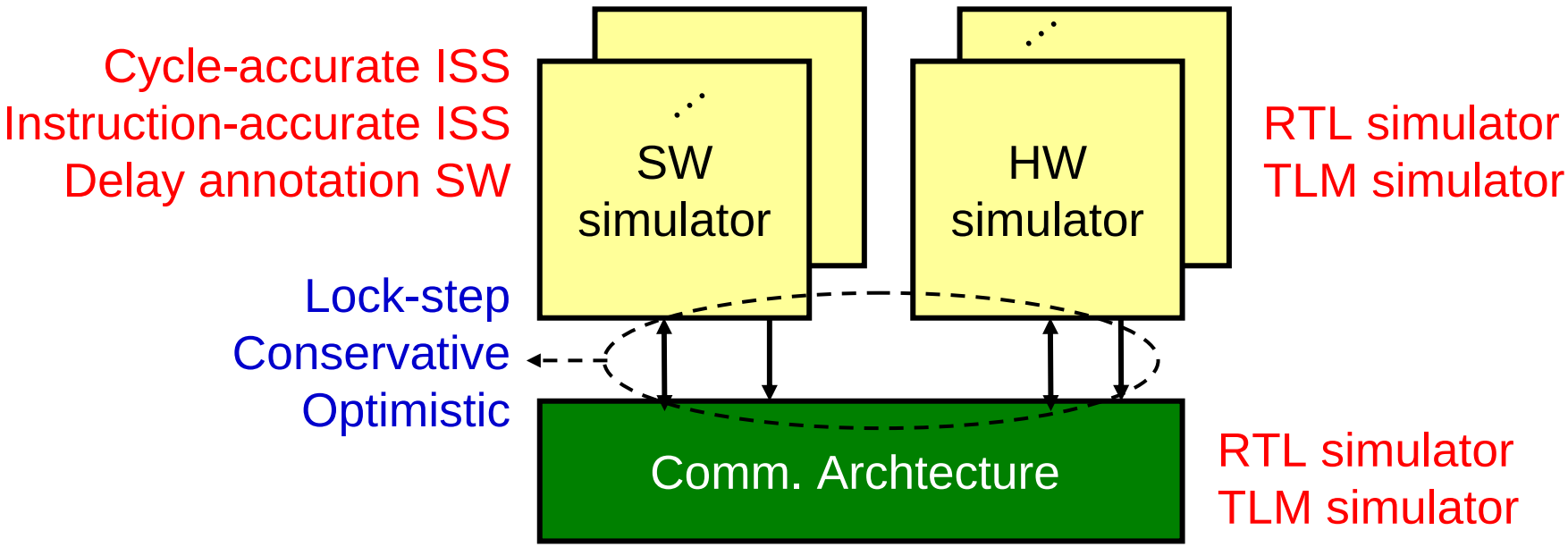
Functional Level Cosimulation

- Software : Host Compiled Model (a Unix process + socket communication)
- Hardware : VHDL Behavioral Model / SystemC Model
- No Channel Model
- **Why do we need this?**
  - No need of Cosimulation in PeaCE



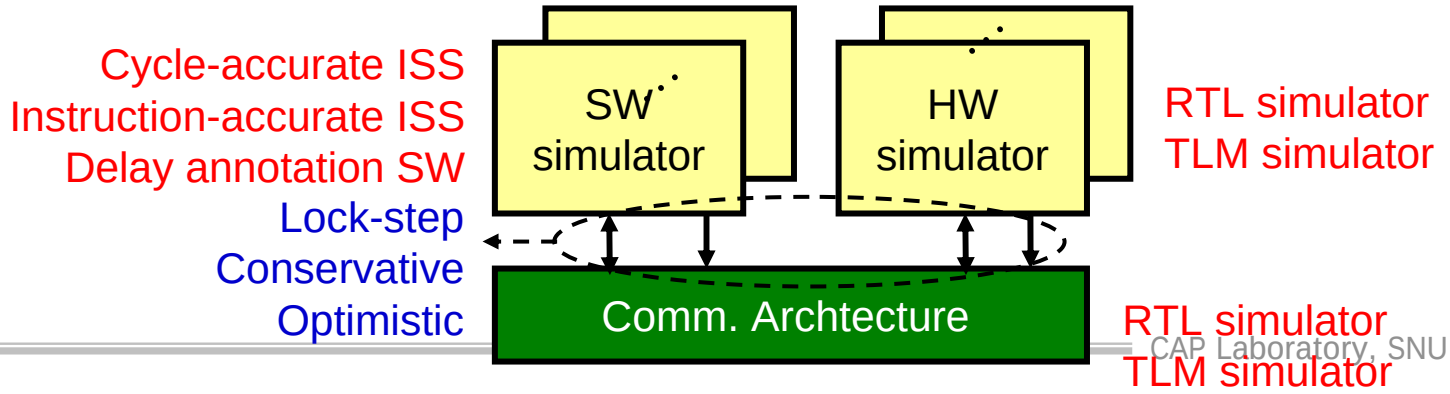
Integration of Component Simulators

- **Different simulation models according to the abstraction level**
- **Different synchronization methods between component simulators**



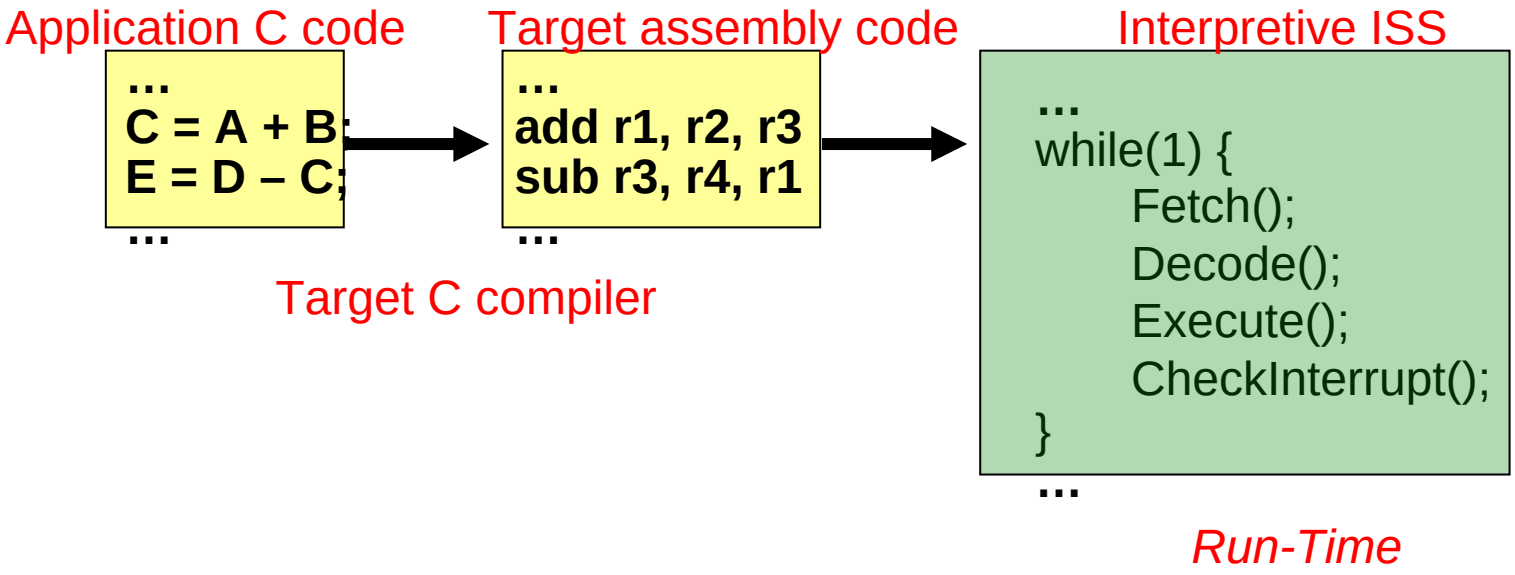
HW/SW Cosimulation Methods

- **Cosimulation for verification**
  - Cycle-accurate ISS + RTL(HDL) simulator + Lock-step synch.
  - SeamlessCVE™ (from Mentor Graphics)
- **Cosimulation for DSE**
  - ISS + TLM simulator + Lock-step synchronization
  - SoC Designer™ (from ARM), ConvergenSC™ (from CoWare)
- **Cosimulation for SW development**
  - Cycle-approximate (instruction-accurate) ISS + TLM simulator



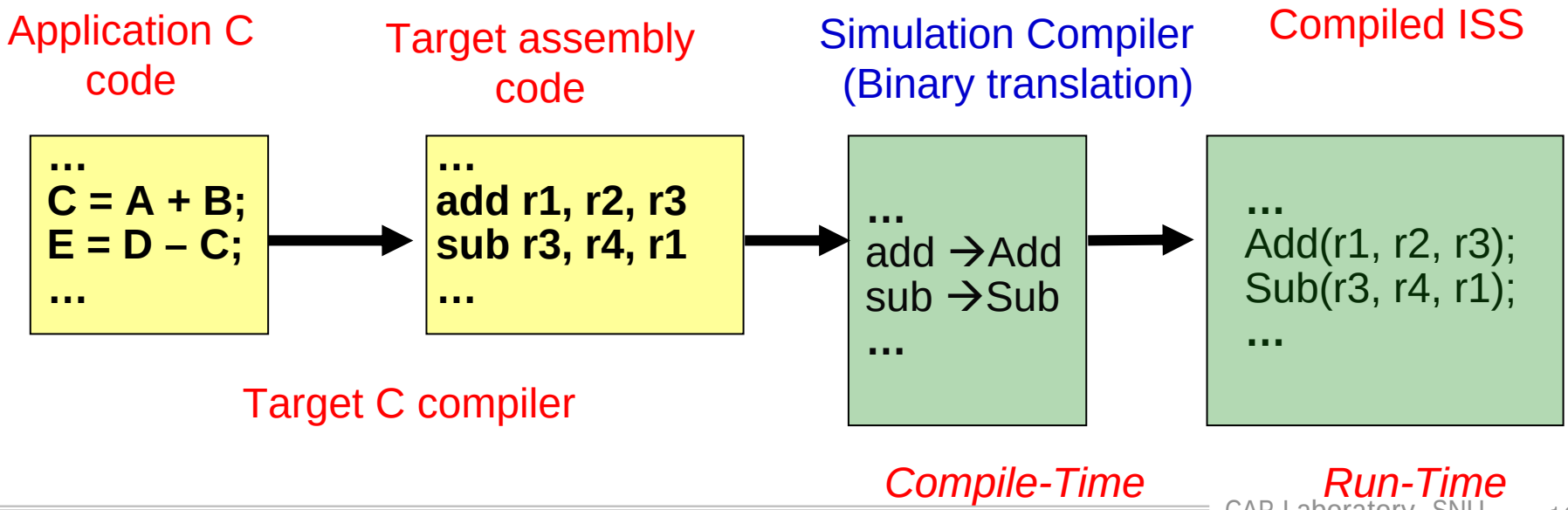
Processor(SW) Simulator: ISS

- **Interpretive ISS (Instruction Set Simulator)**
  - Interpret each line of binary and run it on the processor model
    - *(ex) ARMulator (axd, armsd, rvdebuger etc)*
  - Performance: 2~5 M inst. /sec for ARM processor as of 2007
    - *Instruction decoding time is huge*



Compiled ISS

- **Compiled ISS**
  - Simulation compiler compiles the target binary to produce a C code to be executed on the host machine: reduce decoding time.
  - Advantage: high performance (10-100 X)
  - Limitation : Support static code only. Large memory space (100-1000X)
    - *ex) cannot be used for OS, Boot loading code, ARM/Thumb ISA*



codesign environment

Fast ISS Techniques

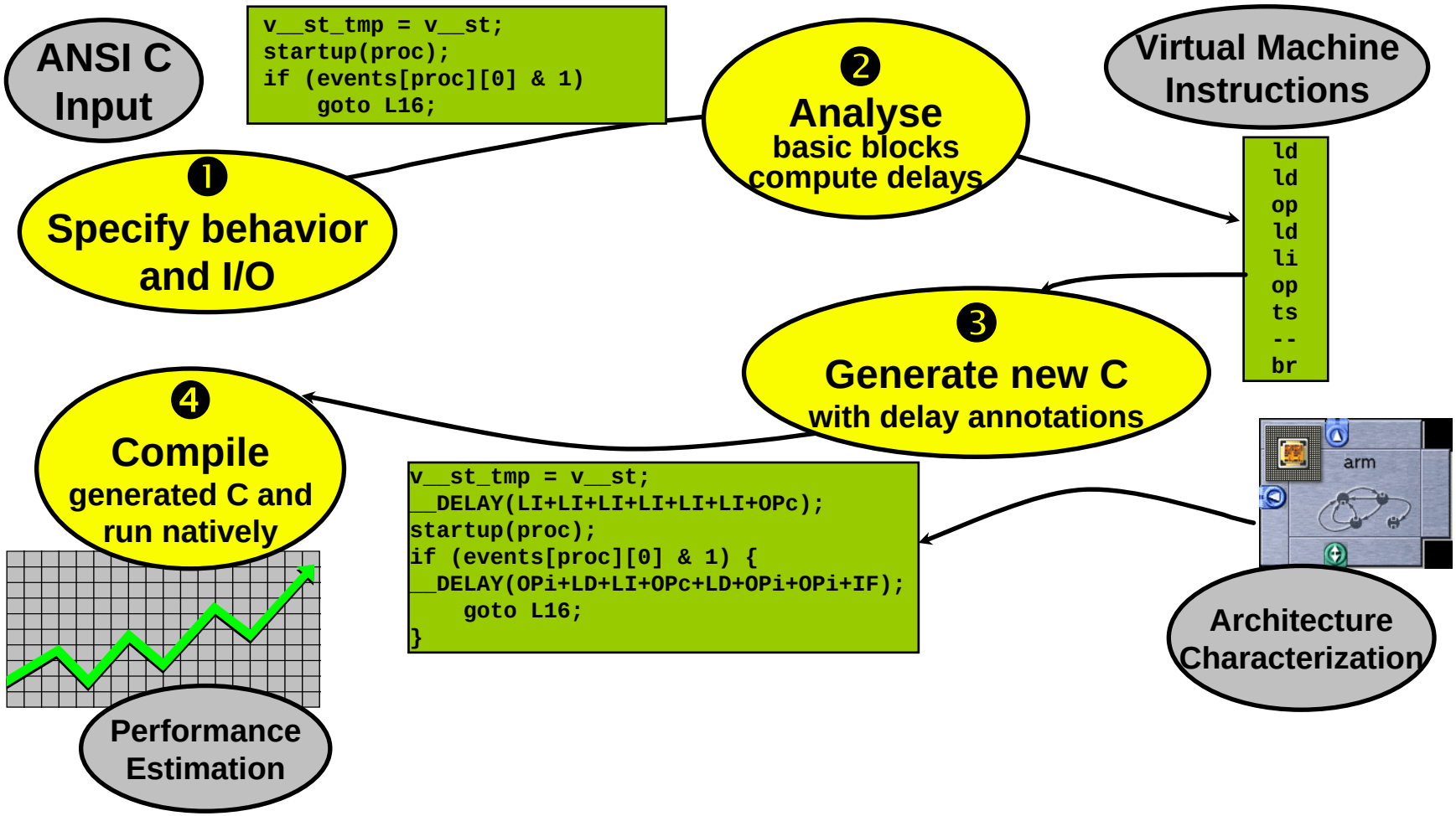
- **JIT-CCS: Just-In-Time Cache Compiled Simulation**
  - A. Nohl et. al., “Universal technique for fast and flexible instruction-set architecture simulation,” DAC 2002.
  - Run simulation compiler at run-time, just-in-time before the instruction is executed and save the extracted information in the simulation cache for direct reuse in a repeated execution.
  - Obtain about 10X speed gain

- **JIT-Binary translation**
  - Translate basic blocks into block translation cache (BTC) while running in an interpretive mode of execution.
  - Obtain 100-300 MIPS performance

Cycle-approximate Simulators

- **Instruction-accurate ISS**
  - Do not model the detailed micro-architecture of the processor
- **Delay Annotated SW**
  - Use host C compiler for the delay annotated SW. How to compute the delay is the key.
  - How to compute delays?
    - *(1) VCC: compile for virtual machine*
    - *(2) Source code analysis – inaccurate*
    - *(3) Assembly code + architecture modeling = assemly-level functionally equivalent C code*
  - Accuracy
    - *10% (3) < (1) < (2) 20% above*

VCC: Software Performance Estimation



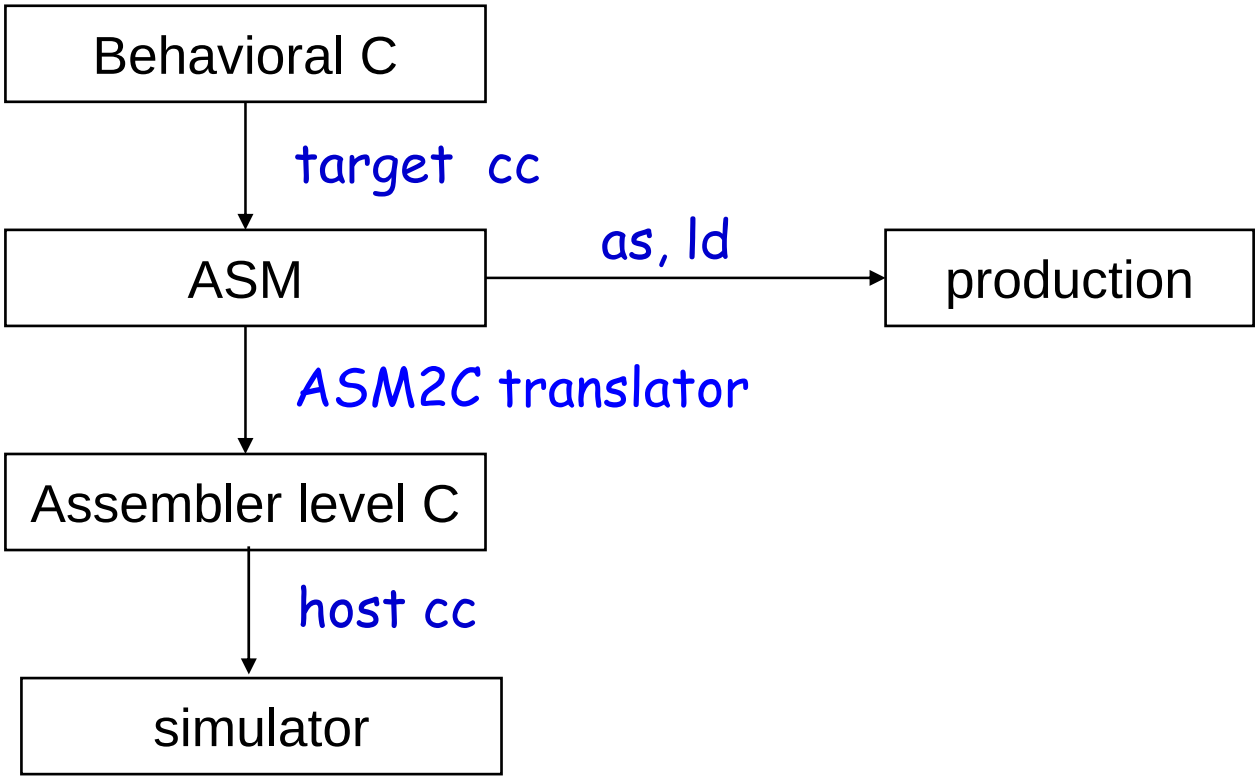
Usage of Virtual Instruction Set (VIS)

- **Constructing the delay table**
  - From the datasheet
  - Run benchmark programs on actual processor and measure
  - Run benchmark programs on cycle-accurate instruction set simulators
    - *Solve a set of linear equations*
- **Compile the function code (e.g. in C) to the VIS.**
- **Delay calculation by adding the delays of virtual instructions.**
- **Applicable to the estimation of power consumption**

<example>

OP	Description	Delay
LD	Load from Data Memory	3
LI	Load from Instruction Memory	1
ST	Store to Data Memory	2
RR	Register-to-Register Move	1
OP	Simple ALU Operation	1
OX1, OX2	Complex ALU Operation	17, 39
TS	Test and Branch	1
BR	Unconditional Branch	3
BS	Branch to Subroutine	19
RT	Return from Subroutine	18

Assembly-level Delay Annotation

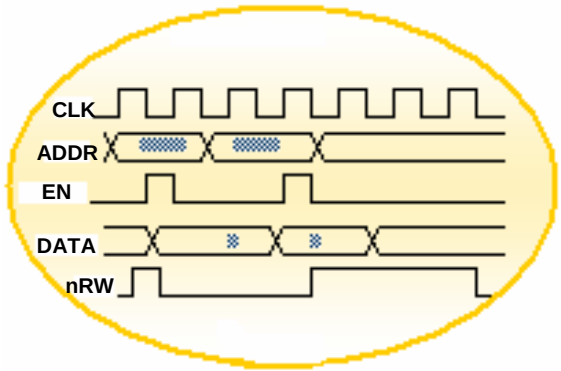


CFG Analysis

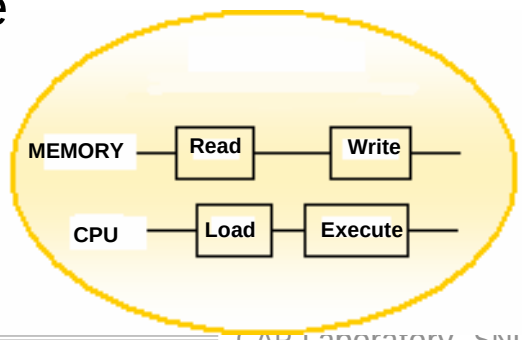
- **CFG construction by compiler**
  - Basic block is a node
  - Compiler optimization can be taken into account
  - For each basic block, pipeline state and cache memory state are recorded
- **Path analysis**
  - Find out the worst-case execution path
  - Can not take into account inter-dependent control structures
- **Exaggerated performance with infeasible path**
- **Not suitable for codesign procedure**
  - Not accurate enough
  - Worst-case behavior may not be of main concern
  - Only for single task

HW/Communication simulators

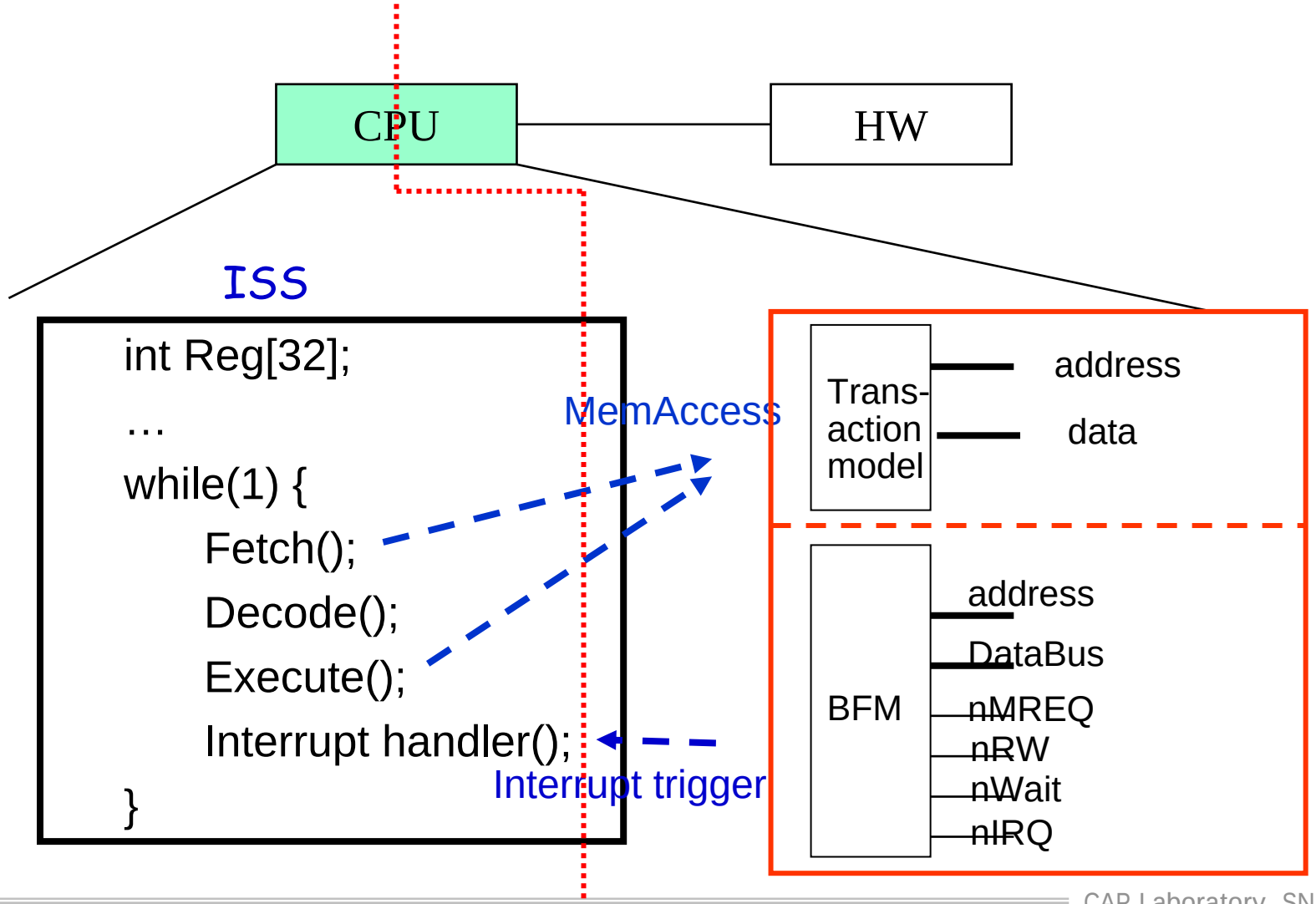
- **RTL(Register Transfer Level) Simulator**
  - Simulate RTL HDL (Hardware Description Language) code
  - Pin-accurate, bit-true, cycle-accurate
    - *ex) WEN, ADDR, DATA*
  - ModelSim™, VCS™, Incisive™ etc.



- **TLM (Transaction Level Modeling) Simulator**
  - Behavioral level description in SystemC
  - Not pin-accurate, but can be cycle-accurate



ISS + HW Simulator



Synchronization Methods

- **Synchronization Points**
  - Lock-step Synchronization: at every (bus) cycle
  - Conservative Synchronization: at future points without causality error
  - Optimistic Synchronization: when causality error is detected
  - Virtual Synchronization: when transaction occurs

- **Synchronization cost**
  - IPC (Inter-Process Communication)
    - *When simulators are implemented by separate processes, use system calls such as socket(), pipe(), etc.*
  - Thread context switch
    - *When simulators are implemented by separate threads within a process, use context switch.*
  - Function call
    - *When simulators are implemented by functions within a process, use function call.*

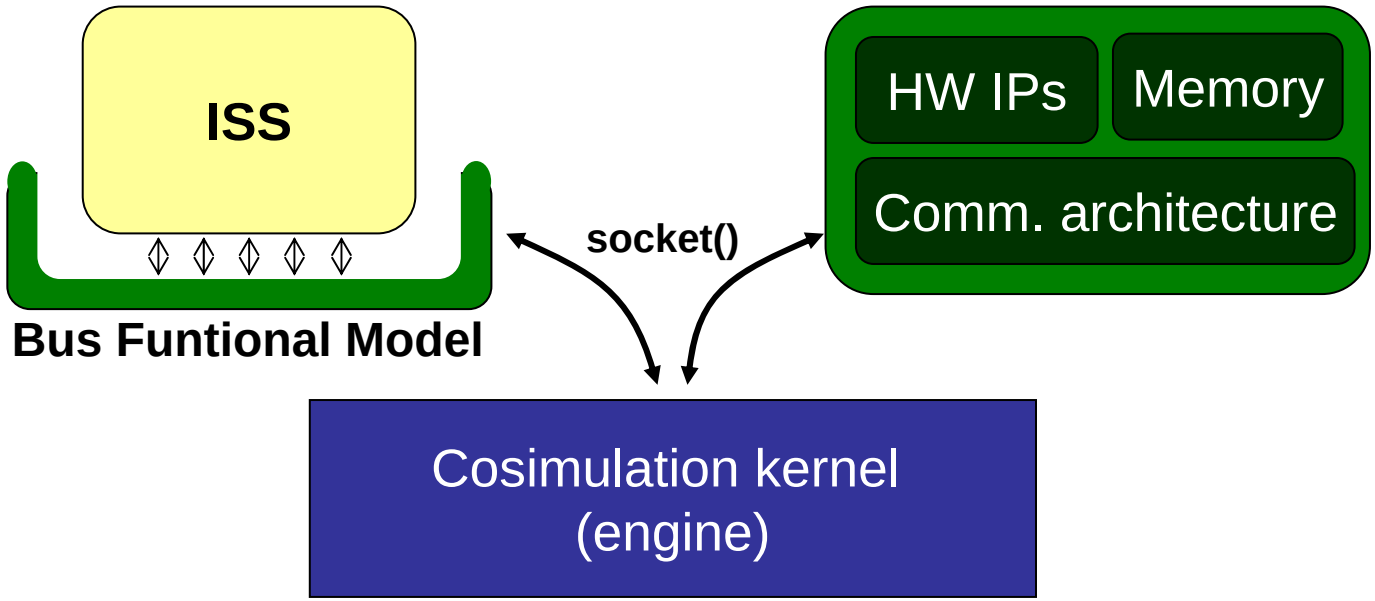
codesign environment

Contents

- Introduction
- RTL Cosimulation
- Co-simulation Performance Analysis
- Making Co-simulation Faster
- Virtual Synchronization

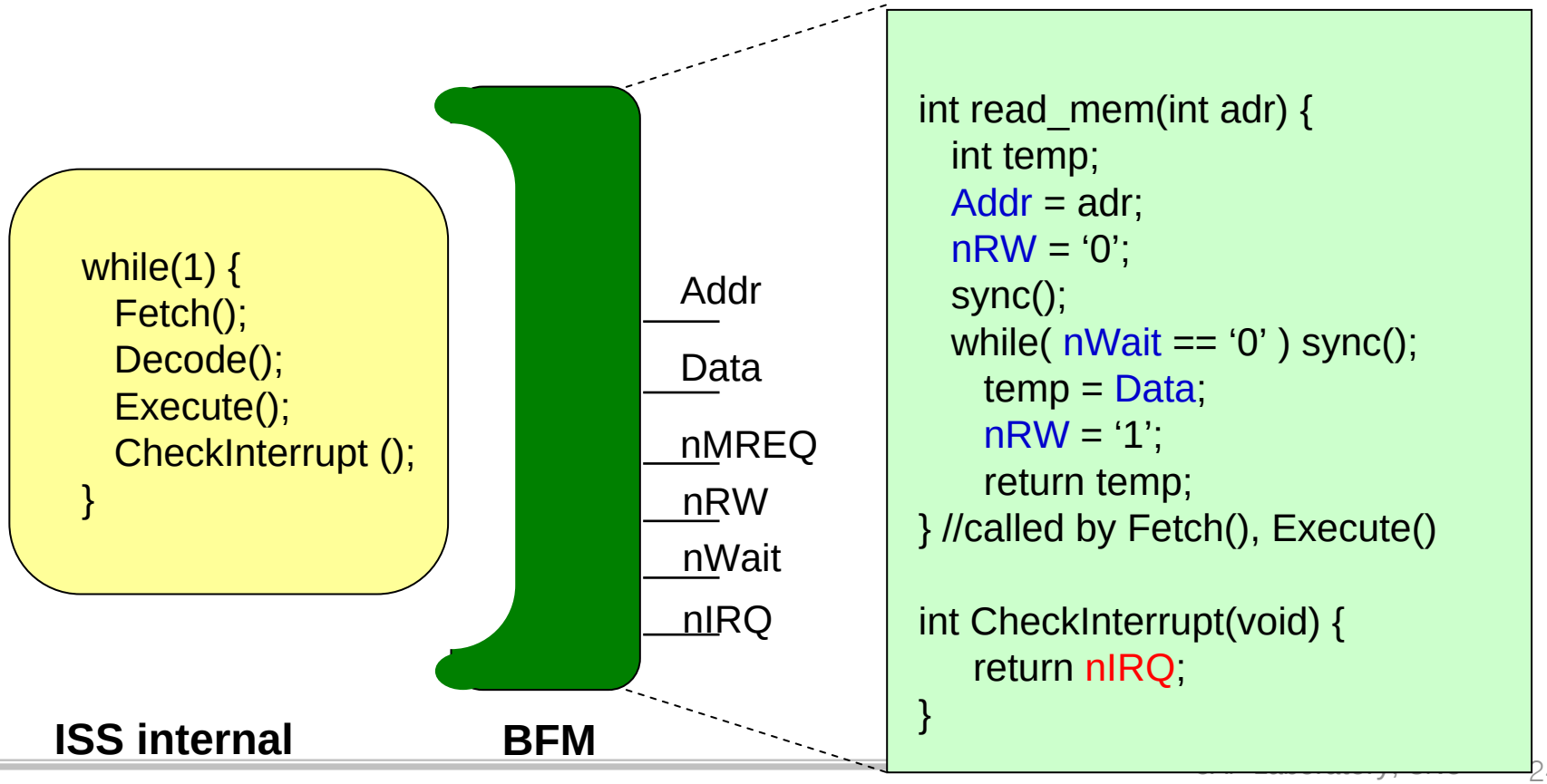
Cosimulation for Co-verification

- **RTL model of the entire system with binary SW**
  - too slow
- **Cycle-accurate cosimulation for verification**
  - Slow (< 1K cycles/sec)
  - ISS + RTL (HDL) simulator + Lock-step Synchronization



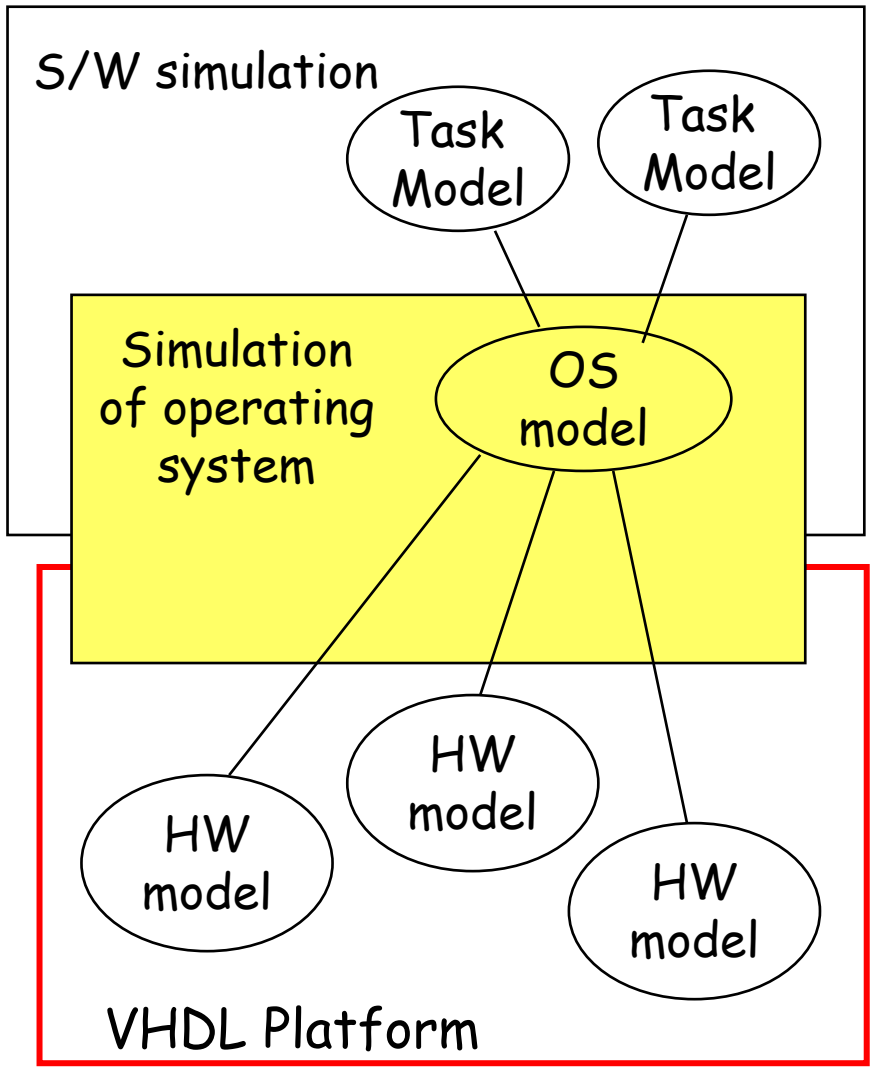
Bus Functional Model

- **Bus Functional Model (BFM)**
  - Transform memory access function call in ISS to a series of cycle-accurate events on processor pins



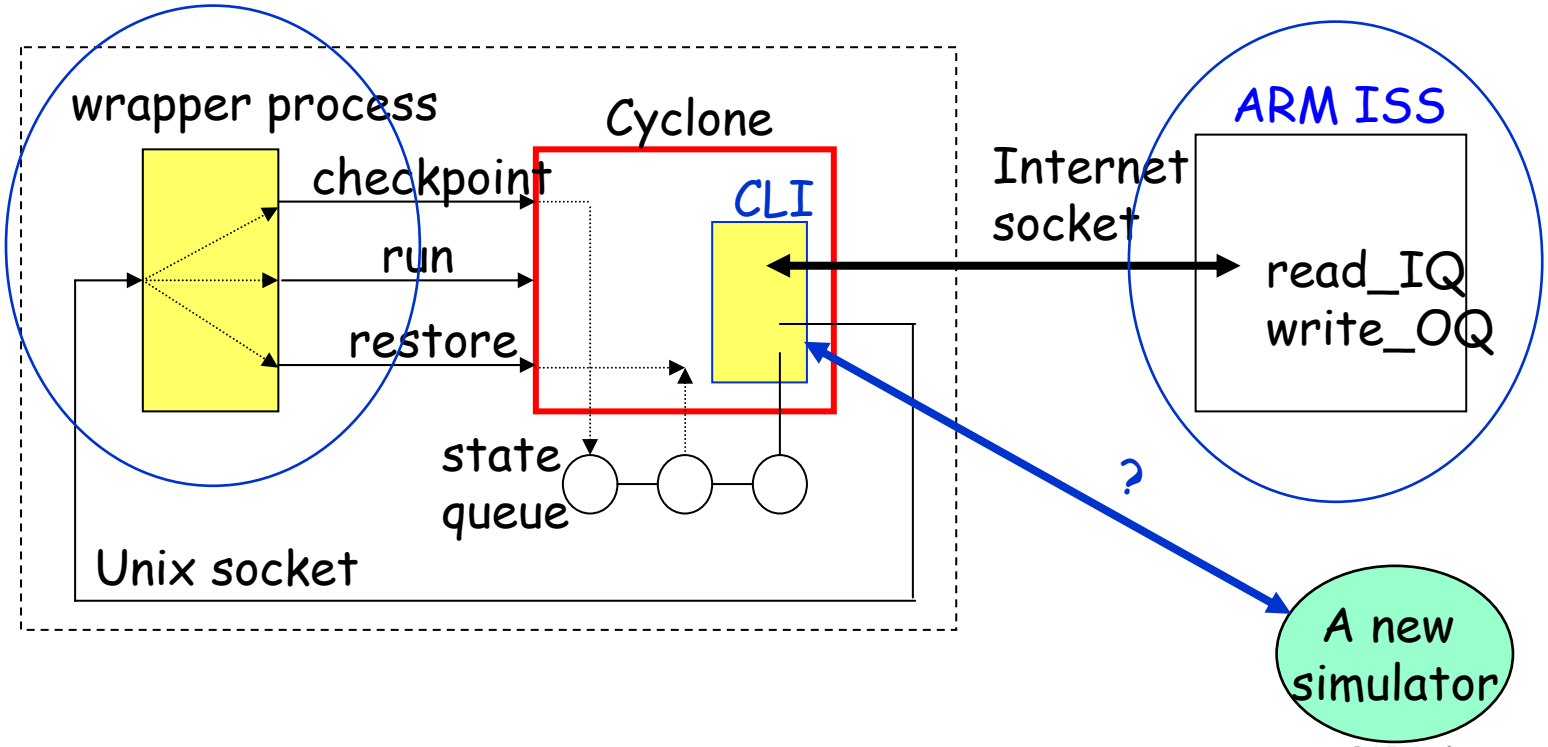
Old Unified Approach Example

- **Cosimulation of realtime control system**
  - by J.P. Soininen, et. al.
- **VHDL Simulator is the main simulation engine**
  - RTOS model is called through VHDL foreign interface
  - SW tasks are managed by OS modeler
- **Limitation**
  - Accuracy of OS modeler
  - Limited parallelism
  - Slow



Direct Coupling Approach

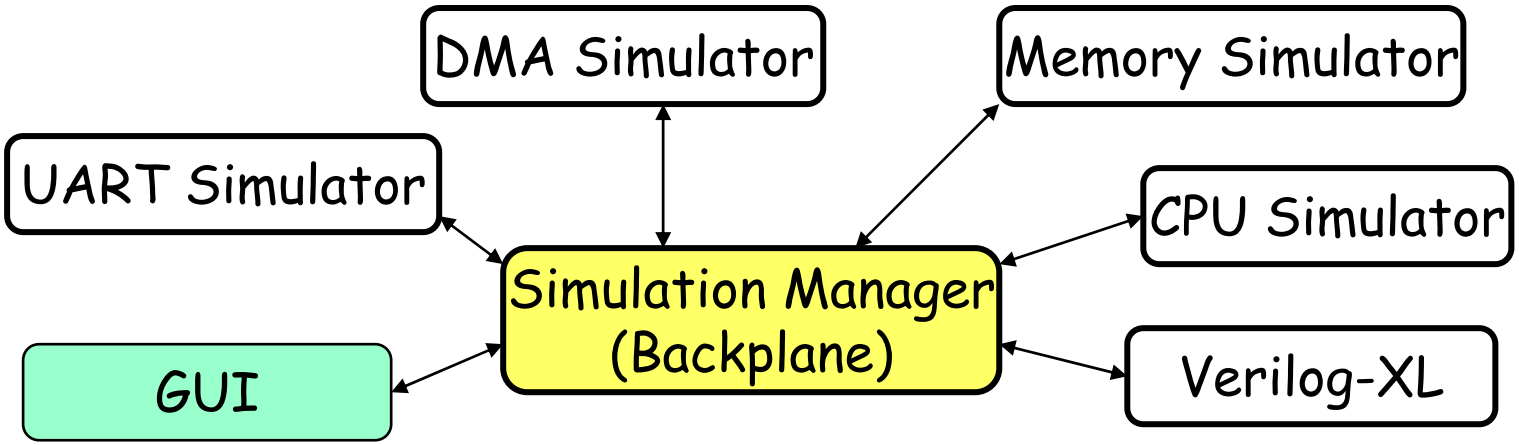
- **Not scalable**
- **No unified interface**



Co-simulation Backplane

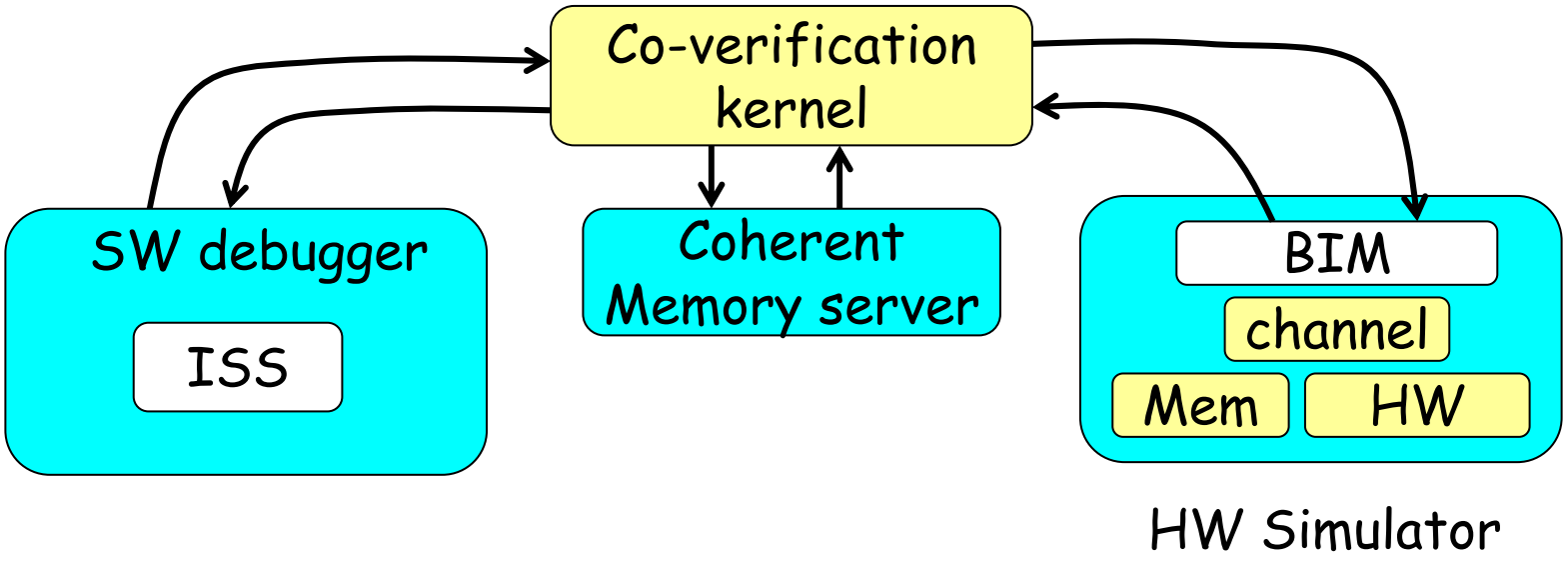
- **Seamless integration of new simulator**
  - O(N) complexity of integration
  - Standard interface
- **Backplane increases observability and controllability of the simulation.**

(ex) A. Ghosh et al. of Mitsubishi Electric Research Lab.



Current Practice: RTL Co-simulation

- **Mentor Graphics Seamless CVE**
- **Distributed co-simulation with multiple simulators**
  - Slow due to heavy time-synchronization overhead
- **BIM (Bus Interface Model) = BFM**



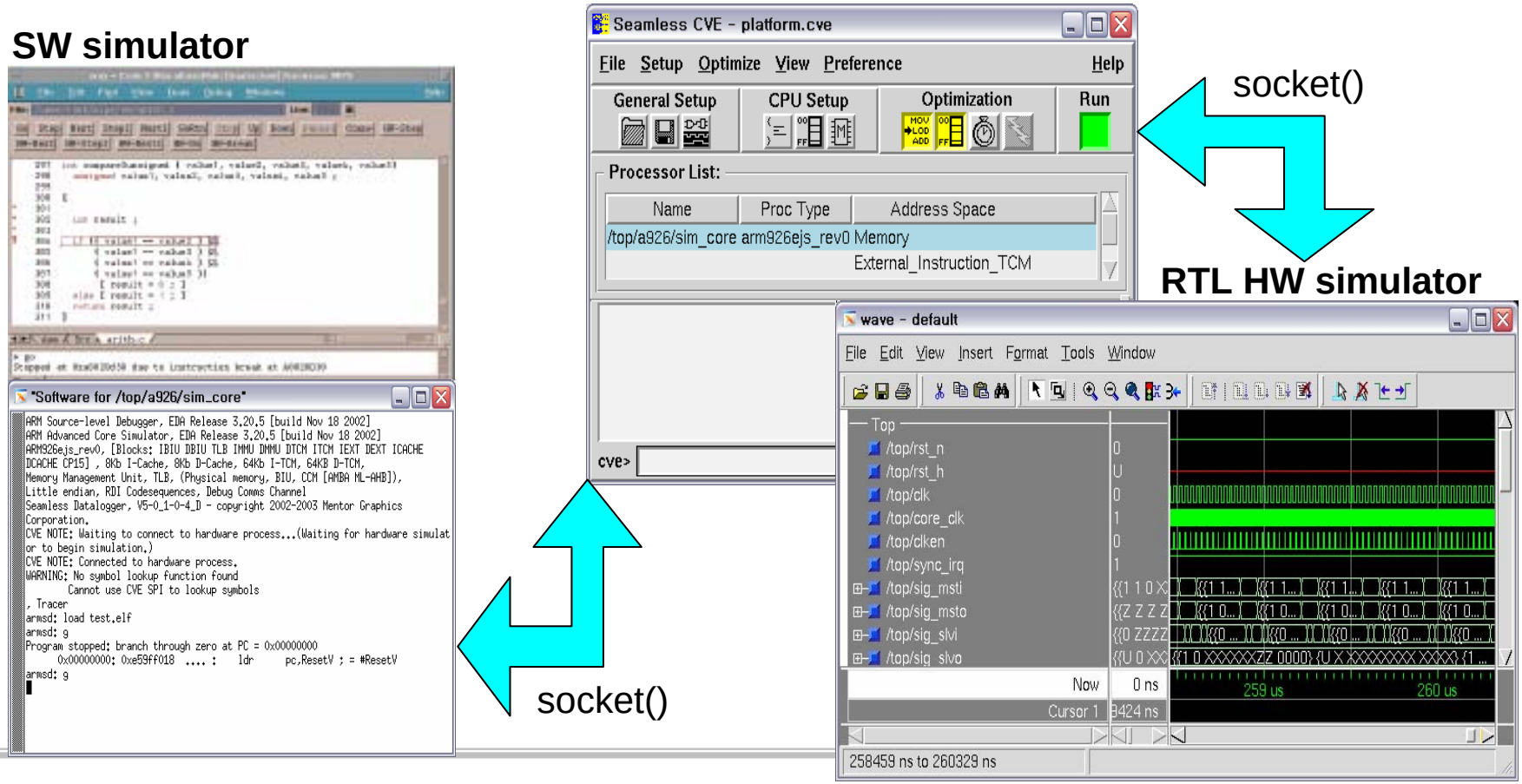
Seamless CVE™ Framework

- **Seamless CVE Framework**

cycle-accurate

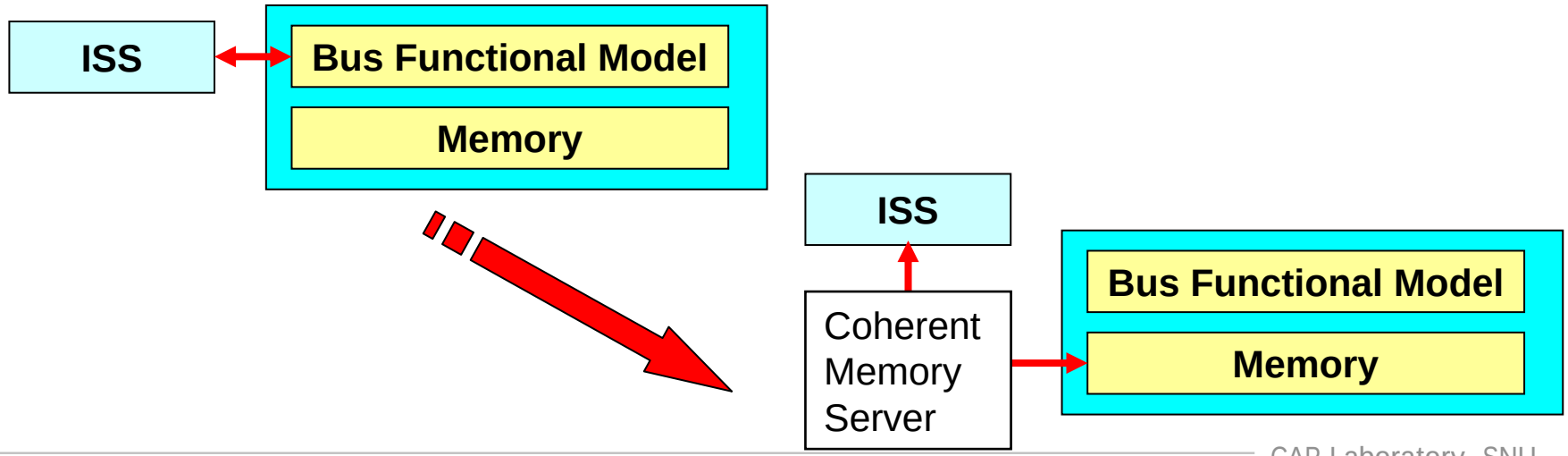
SW simulator

Cosimulation kernel (lock-step synchronization)



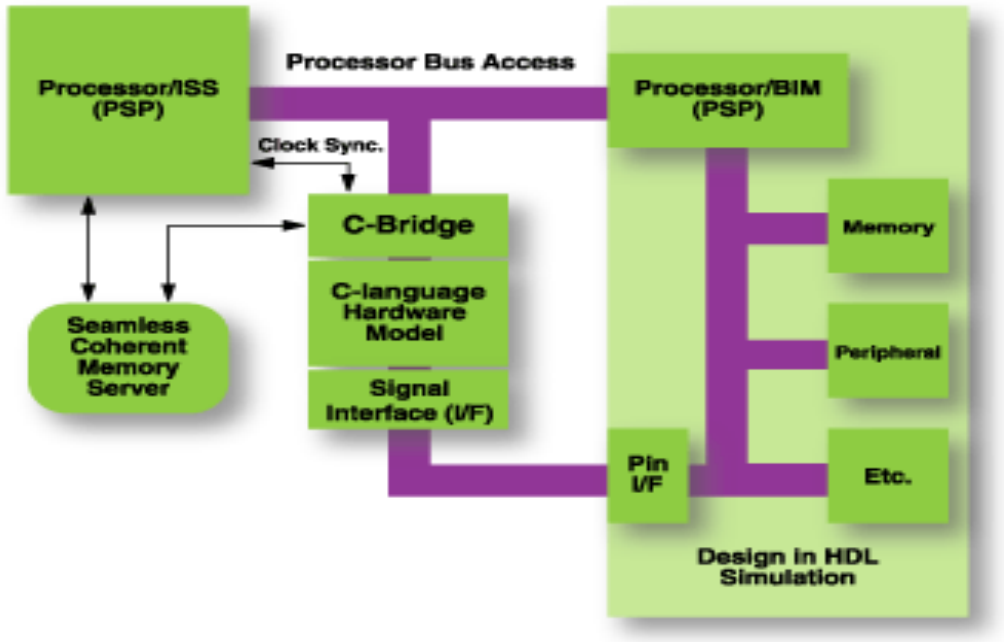
Performance Optimization

- **Fetch Optimization**
  - Instruction fetch is done locally without bus activity
  - ~ 200 instruction/sec
- **Data Access Optimization**
  - Local data access in ISS is done without bus activity
  - ~ 1K instruction/sec



Further Optimization

- **Time Synchronization Optimization**
  - No clock synchronization between SW and HW
  - Synchronization at the transaction level
  - ~ 10K instruction/sec
- **C-bridge Technology**
  - C/C++ Modeling of pre-verified HW block (SystemC + behavioral VHDL)



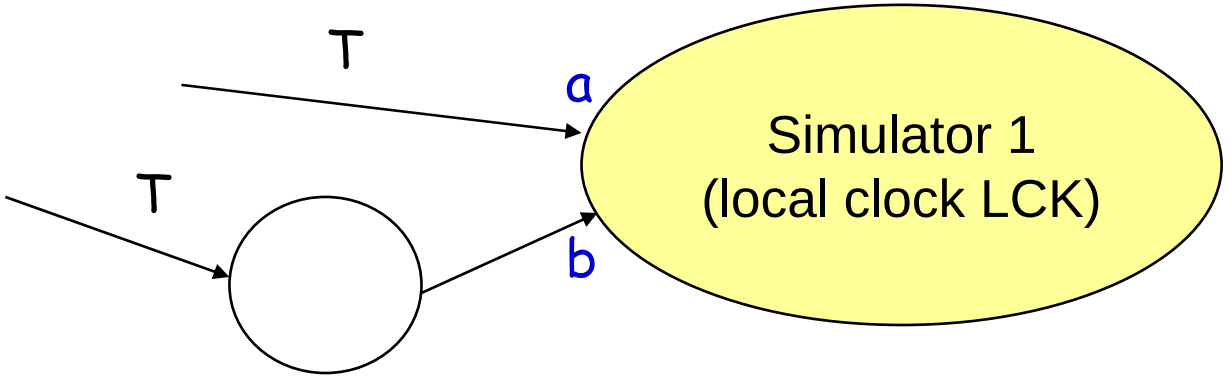
codesign environment

Contents

- Introduction
- RTL Cosimulation
- Co-simulation Performance Analysis
- Making Co-simulation Faster
- Virtual Synchronization

Timing Synchronization Problem

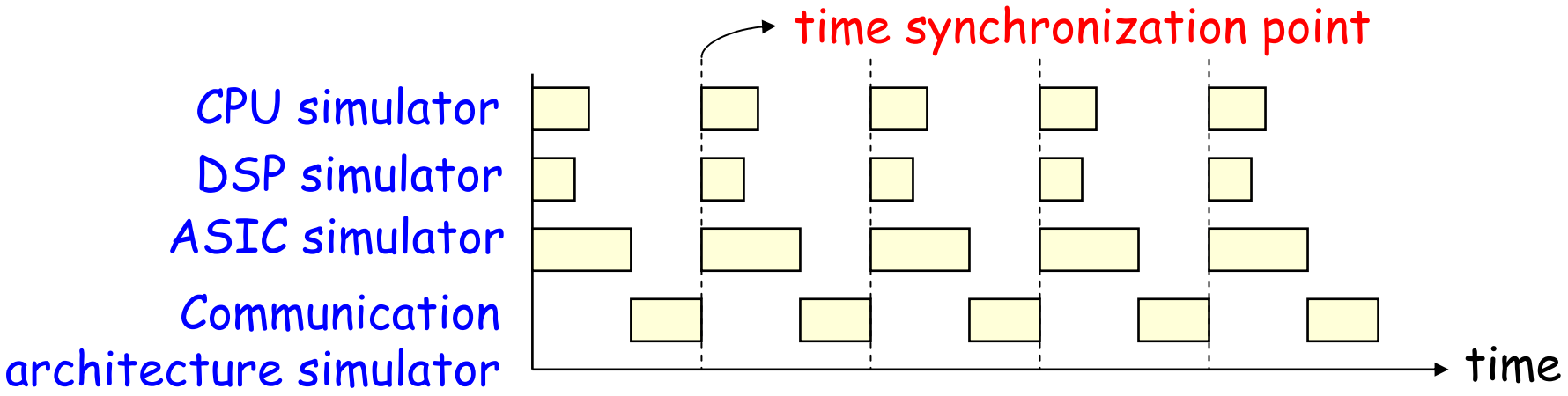
- **Causality error**
  - Local clock can not be ahead of the time stamp of a new event



- LCK < T
- Simulator 1 should wait until it receives an event from arc **b**.

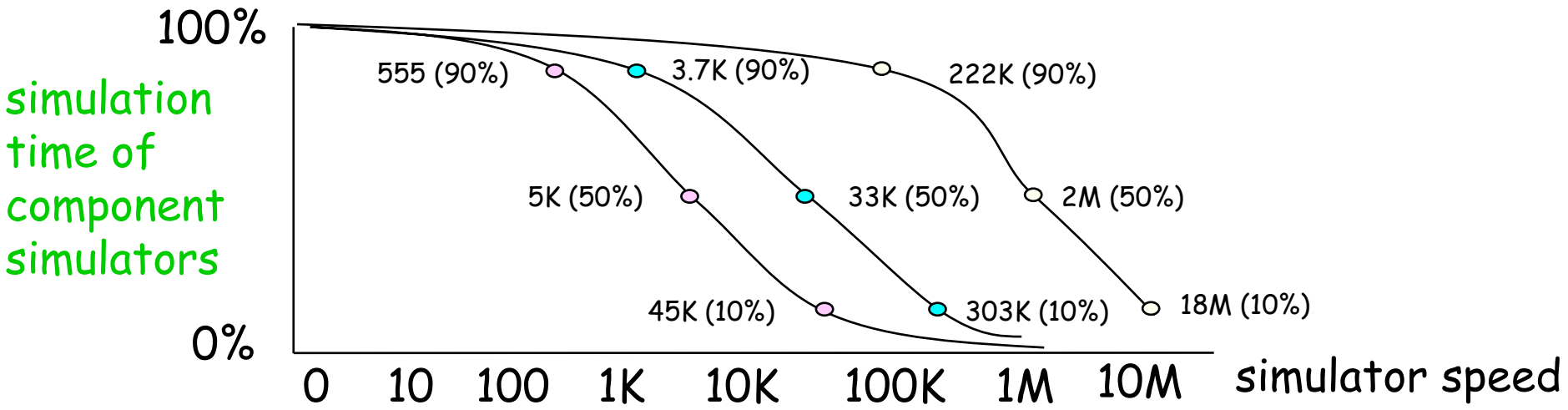
Lock-step Synchronization

- **For accurate functional behavior, each simulator should have valid data when it consumes or produces data**
- **For accurate timing behavior, each simulator should check events from other simulators at every clock which may affect timing behavior of the simulator**



Performance Problem in Cosimulation

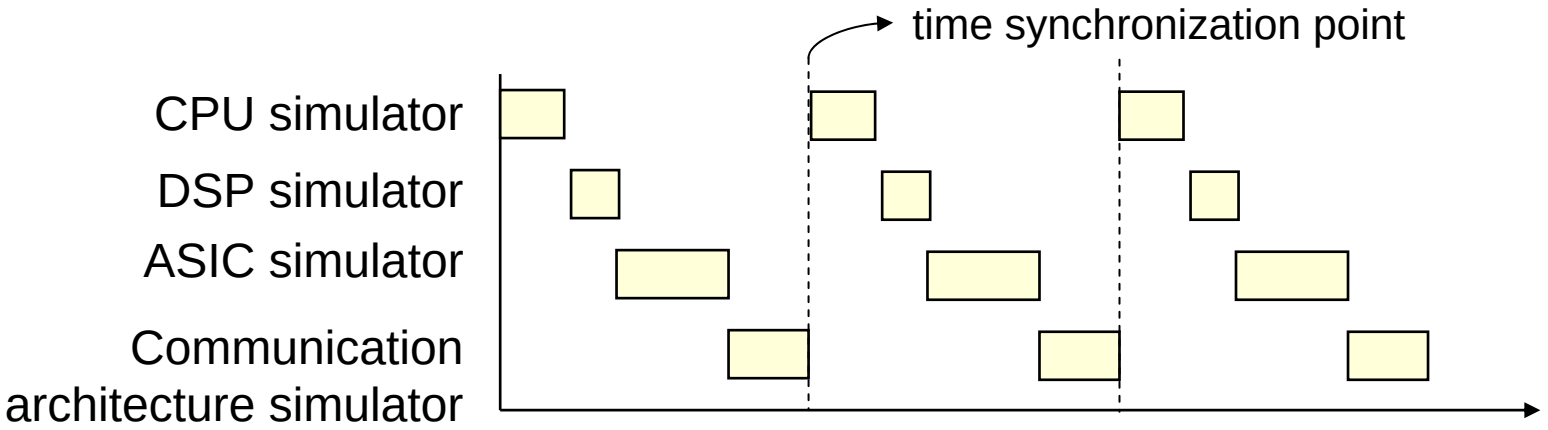
- **Slow speed of component simulators**
  - Processor simulator : 1M~10M cycles / second
  - Hardware simulator : 100 ~ 100K cycles / second
- **Time synchronization overhead : send data & receive data**
  - Function call : 0.5 us
  - TCP/IP (local) : 30 us (18 us using Pthread)
  - TCP/IP (remote) : 200 us *Linux 2.4, Pentium 1.8GHz dual, 100M LAN



Distributed vs. Serial Execution

- **Distributed simulation may be reasonable only when the simulation of a processing component is smaller than 10K-100K if lock-step synchronization is used**

- **Otherwise, simulators had better be serialized on the single machine**



Cosimulation Performance

- **Performance factors**
  - Simulation speed of processing component simulator
  - Simulation speed of communication architecture simulator
  - Synchronization overhead (cost)
  - Synchronization count
- **Cosimulation performance formula (lock-step synchronization)**

T : Total simulated cycles

st_i : Simulation time to advance one cycle of simulator i

$sync$: Synchronization overhead

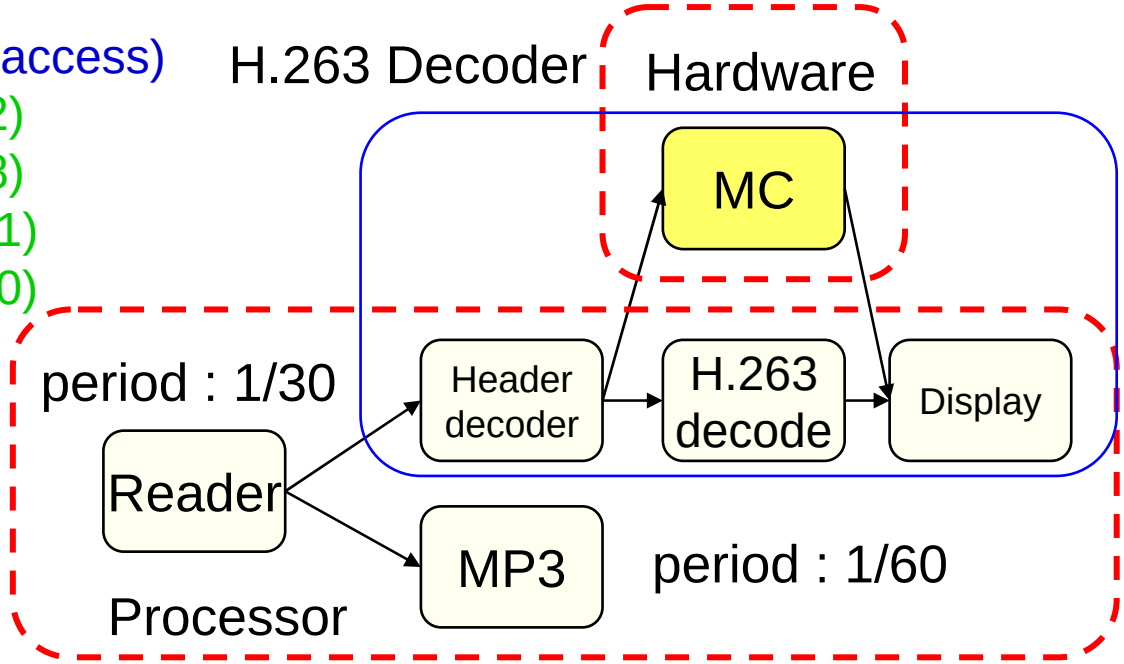
$tran_{num}$: The total number of communication transactions

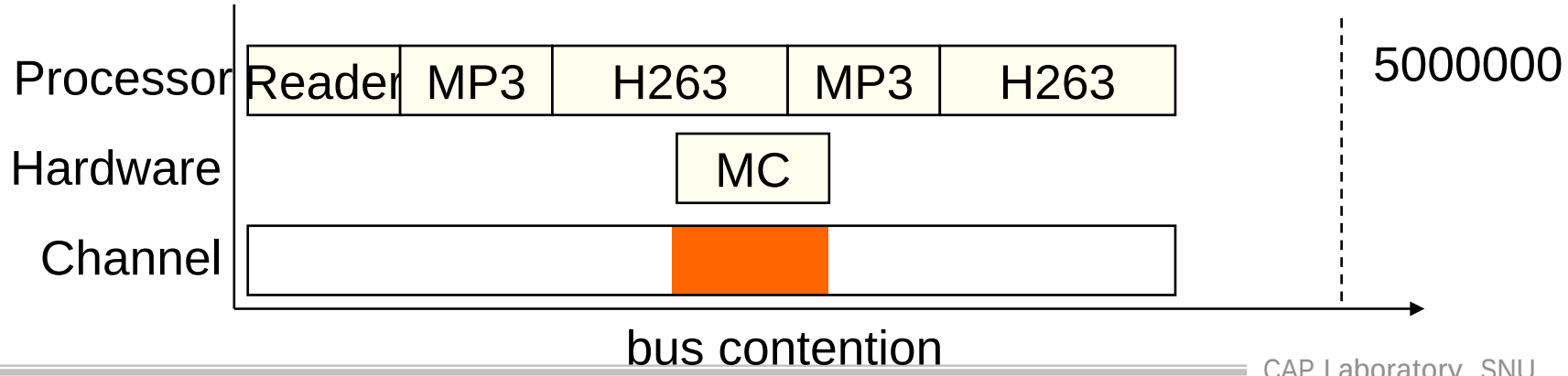
st_{tran} : Simulation time to process a transaction

$$\sum_{\forall i} \{ T \times (st_i + sync) \} + tran_{num} \times st_{tran} \quad \text{(eq. 1)}$$

Example (DIVX Player)

Block : cycles (memory access)
Reader : 10505 (2162)
MP3 : 217601 (22448)
H263 : 3965626 (262861)
MC : 178200 (44550)





Cosimulation Time

- **Hardware simulator (RTL) : 10K cycles / s**
- **Software simulator : 1M cycles / s**
- **Synchronization overhead : 40 us**
- **Communication architecture : 6.5K transactions / s**
  - due to RTL modeling
- **Parameters are 10 times faster than those from Seamless CVE**

- **Simulation time on processor**
  - 5000000 * (1/1000000 + 0.00004) = 5 + 200
- **Simulation time of hardware**
  - 5000000 * (1/10000 + 0.00004) = 500 + 200
- **Simulation time of communication architecture**
  - 332021/6500 = 51 (332021 = sum of memory access counts)
- **Total Simulation Time = 956 s**

codesign environment

Contents

- Introduction
- RTL Cosimulation
- Co-simulation Performance Analysis
- Making Co-simulation Faster
- Virtual Synchronization

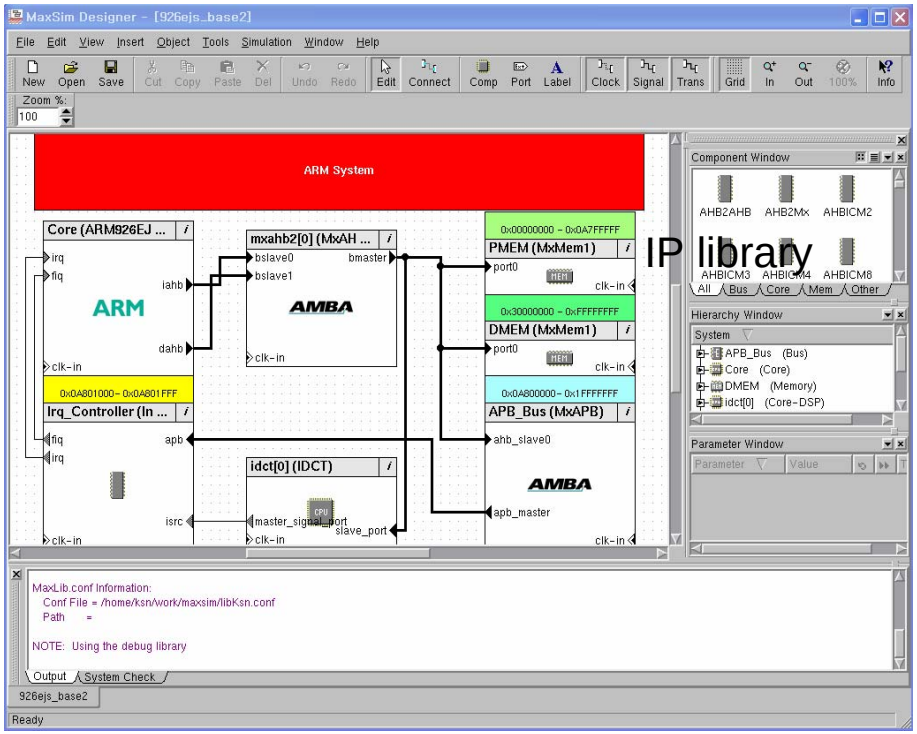
(1) TLM Co-simulation

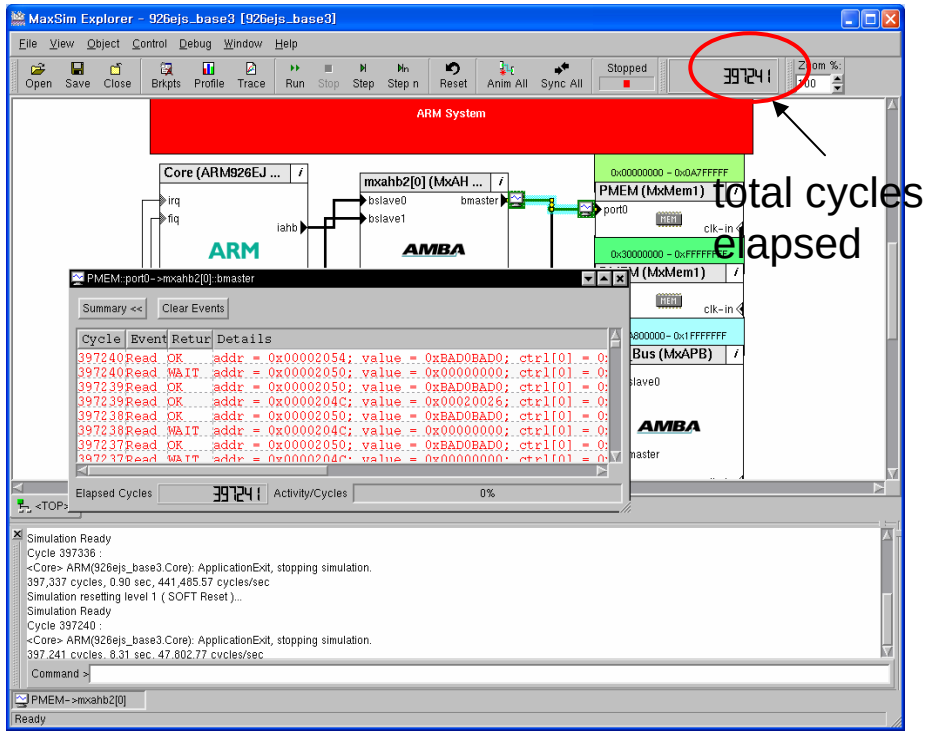
- **Making st_i, st_{trans} smaller**
  - Utilizes accuracy/performance tradeoff to enhance simulation performance on processing component simulator and communication architecture simulator separately
  - Hardware simulation: SystemC + time annotation
  - Transaction level model of channels
  - Has still large synchronization overheads particularly as the number of processors increases
  - Slow simulation speed by executing operating system on ISS

- **We will have a lab. with RealView SoC Designer from ARM.**

RealView SoC Designer™

- **RealView SoC Designer (Old: MaxSim) Designer**
  - Construct a platform using Hardware IPs from component library

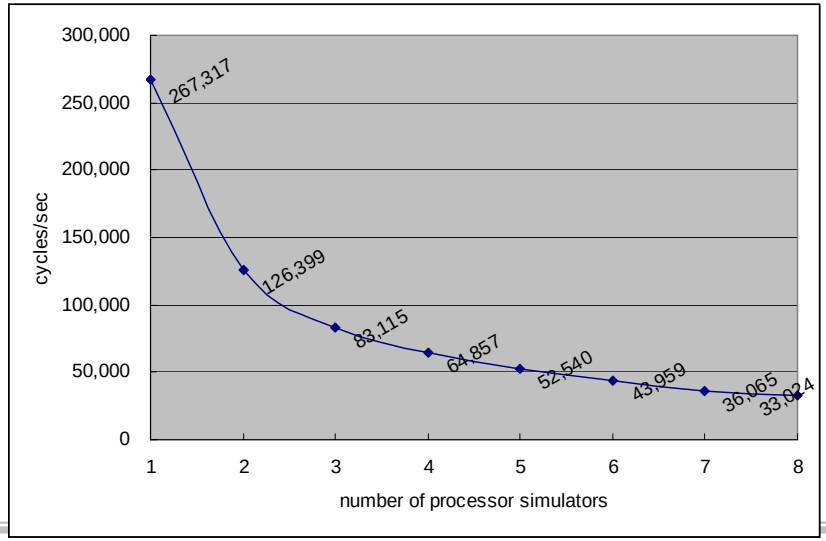




MPSoC Simulation

- **MPSoC**
  - # of processor cores increases
    - ➔ *Cosim. speed is inverse proportion to # of simulators.*
  - # of IPs (possibly in different abstraction levels) increases
    - ➔ *Mixed Abstraction Level Cosimulation ➔ IPC synchronization*

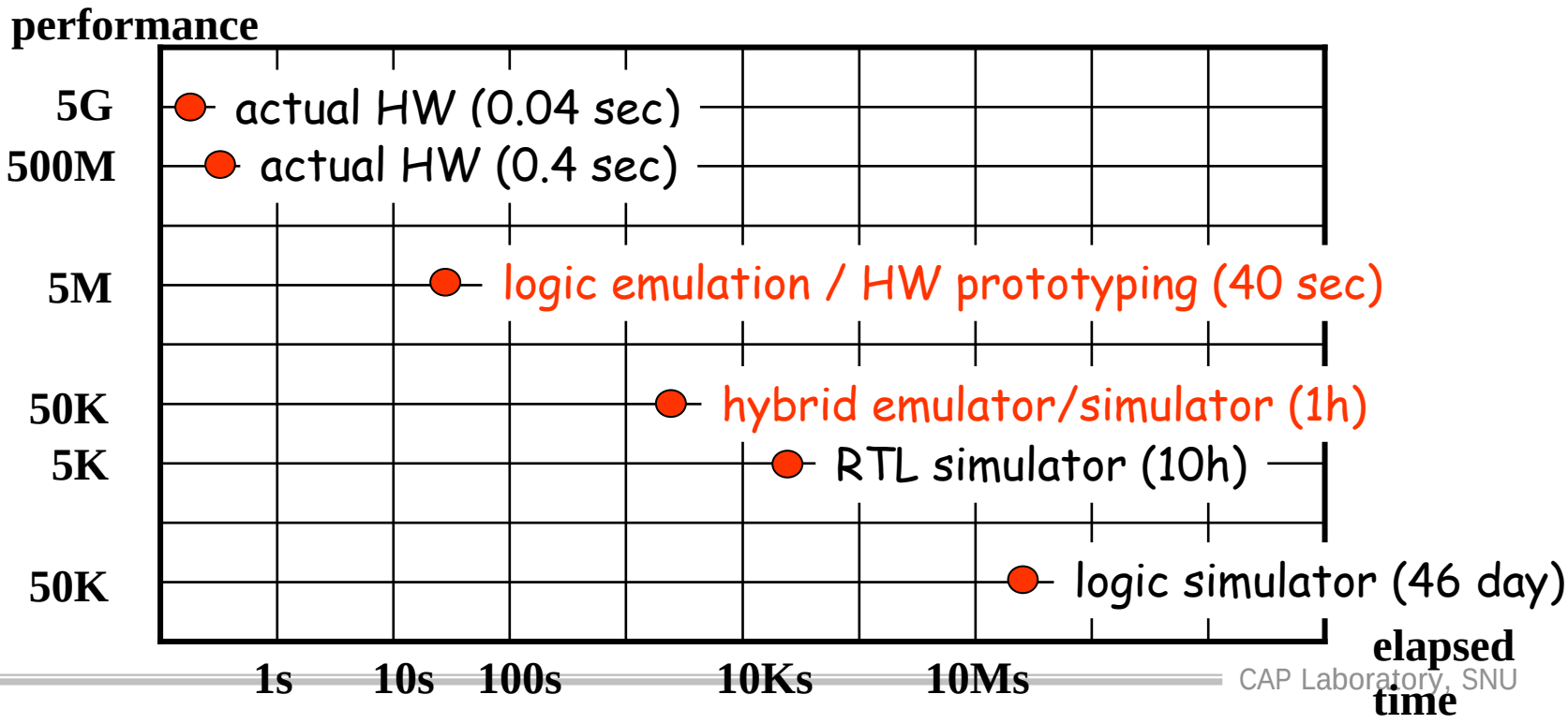
- **Current TLM with lock-step approach is not good enough**



MaxSim™ on Xeon 1.8GHz

(2) Hardware Emulation

- **Making st_i smaller**
  - Provide very fast simulation speed about 1M cycles / s
  - Cost too much
  - Do not solve time synchronization problem

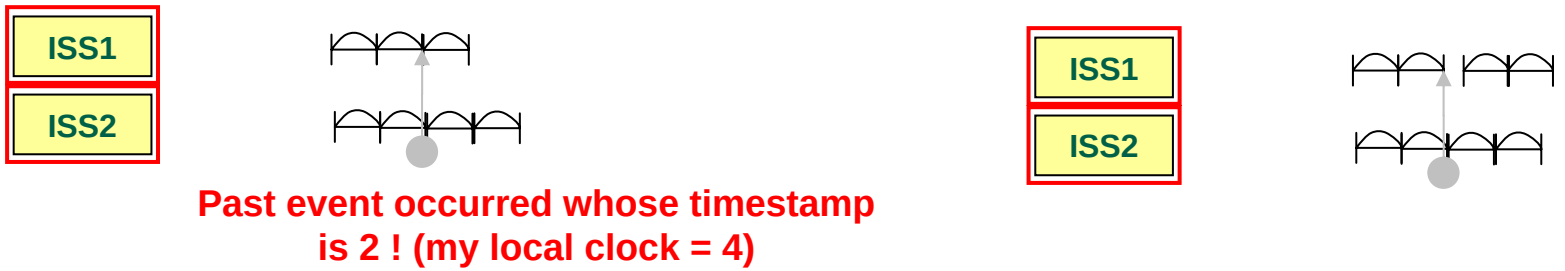


(3) Reduce Synchronization Points

- **Reduce T in $sync \times T$**
- **Conservative approach**
  - All simulators notify the next event time to one another. And then each simulator can safely advance its local time until the smallest next event time of simulators
- **Optimistic approach**
  - Each simulator advances its local clock optimistically assuming that no past event will arrive. If this assumption fails, it rolls back its local time to the event arrival time canceling all results that have been processed after that time
- **Their applications and performance enhancements are limited by their prerequisites**

Conservative Approach

- **Guarantees that no *past event* will occur**
  - Safely advance simulator clock to the minimum timestamp value of the event in the event queues



Past event occurred whose timestamp is 2 ! (my local clock = 4)

- **How to know it is safe?**
  - Optimized approach
    - *Examining the event queue*
  - Static analysis of SW codes
    - *Basic block analysis*
  - Static analysis of SW codes + Dynamic execution path prediction of SW + HW prediction

Optimistic Approach

- **Simulators advance the clock optimistically**
  - Assuming that no past event will arrive
  - If assumption fails, it rolls back its time to the latest check point time
    - *Checkpoint, state recovery overhead*
  - Pros : no need for the next event time prediction
  - Cons : need recovery time
    - *check-pointing and state saving overhead*
    - *simulator support is required*

Past event occurred whose timestamp is 2 ! (my local clock = 4) ➔ Rollback

(4) Trace-driven simulation

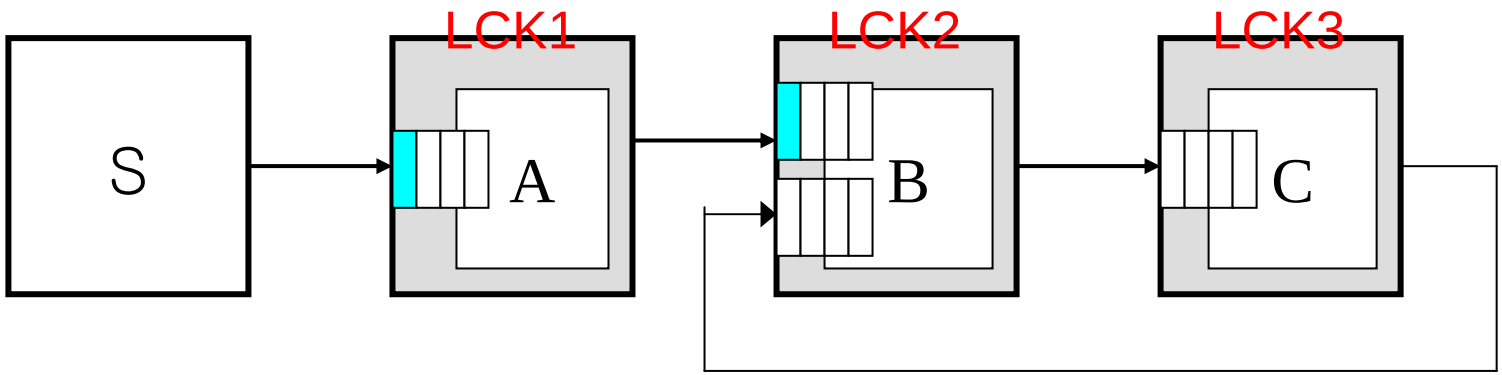
- Stores memory traces from cycle-accurate cosimulation without considering dynamic behavior from communication architecture
- Evaluates memory traces for different communication architectures very fast by only simulating dynamic behavior of communication architecture : small st_{trans}, $st_i \approx 0$
- Performance bottleneck of the approach is caused by accessing memory traces at the external storages
- Because it does not handle dynamic behavior which comes from data synchronization and operating system, the accuracy becomes worse when the system has operating system and is composed of multiple processors

Other Approaches

- **Instruction fetch optimization in Seamless CVE : smaller** $tran_{num}$
  - reduces the number of transactions to the communication simulator by making only shared memory accesses delivered to the communication simulator
  - loses time accuracy without considering bus contention for local memory accesses and still shows poor performance

- **Make synchronization overhead lighter : smaller** $sync$
  - Make simulators as shared library and perform time synchronization using a function call

Distributed Event-driven Simulation



- **No global synchronization of local clocks: pair-wise synchronization at data communication: only partial ordering is enforced**
  - **Speed-up due to parallel execution**
- **A (simulator) process processes an event when no causality error is guaranteed: intuitive solution is to wait until it receives events from all input ports**

Deadlock!?

How to Solve Deadlock Problem?

- **Deadlock avoidance by Null messages**
  - Whenever a process finishes processing an event, it sends a null message on each of its output ports indicating the lower bound of the next time stamp.
  - Estimation of this lower bound and generation of Null messages are all application programmer's duty.
- **Chandy and Misra's approach: Deadlock Detection and Recovery**
  - Deadlock detection: not an easy task
    - *Global deadlock*
    - *Local deadlock: starvation!*
  - Deadlock recovery: detect and process the system-wise earliest event – performance bottleneck
  - Assume a fully-distributed simulation.

codesign environment

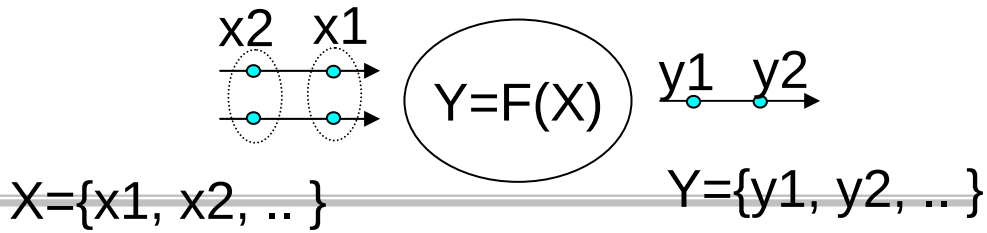
Contents

- Introduction
- RTL Cosimulation
- Co-simulation Performance Analysis
- Making Co-simulation Faster
- Virtual Synchronization

Core of Virtual Synchronization

- **Separate functional simulation and timing management**
- **Functional simulation and trace generation**
  - It acquires execution traces from processing component from HW/SW co-simulation ignoring the communication overhead.
- **Timing management**
  - For accurate timing behavior, it reconstructs timing correctness by adjusting time of execution traces using trace-driven architecture simulation

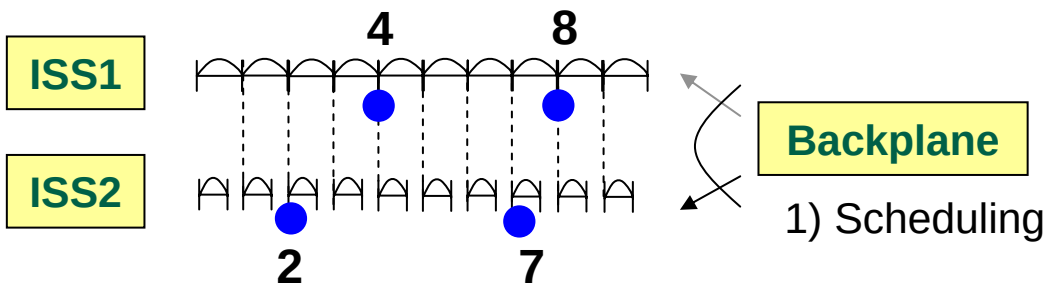
- **Assumption**
  - Behavior of a task is independent of the absolute timing of incoming data.



Principle of VS framework

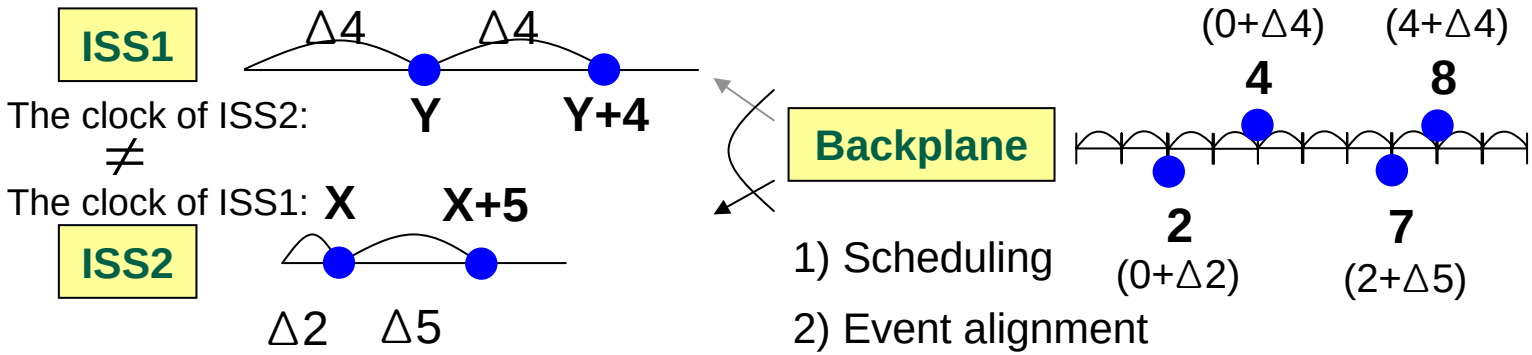
- **Increase cosimulation speed by minimizing synchronization overhead**

Conventional synch. scheme



1) Event generation
2) Event alignment ← Maintain synchronized clock

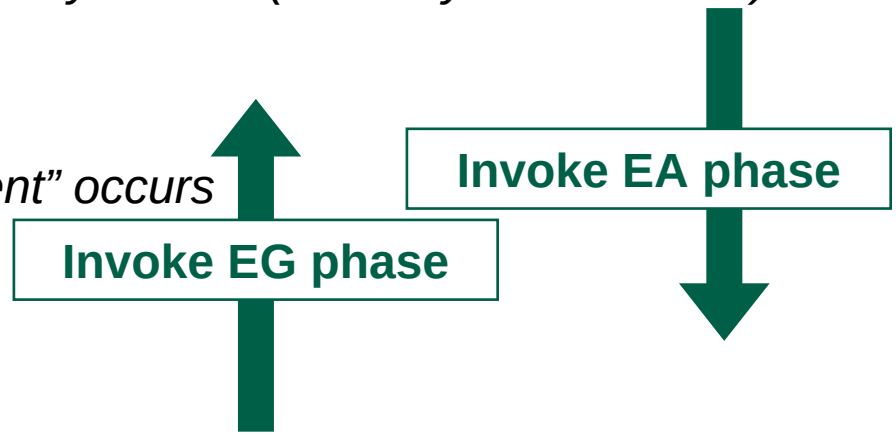
Virtual synch. scheme



1) Event generation

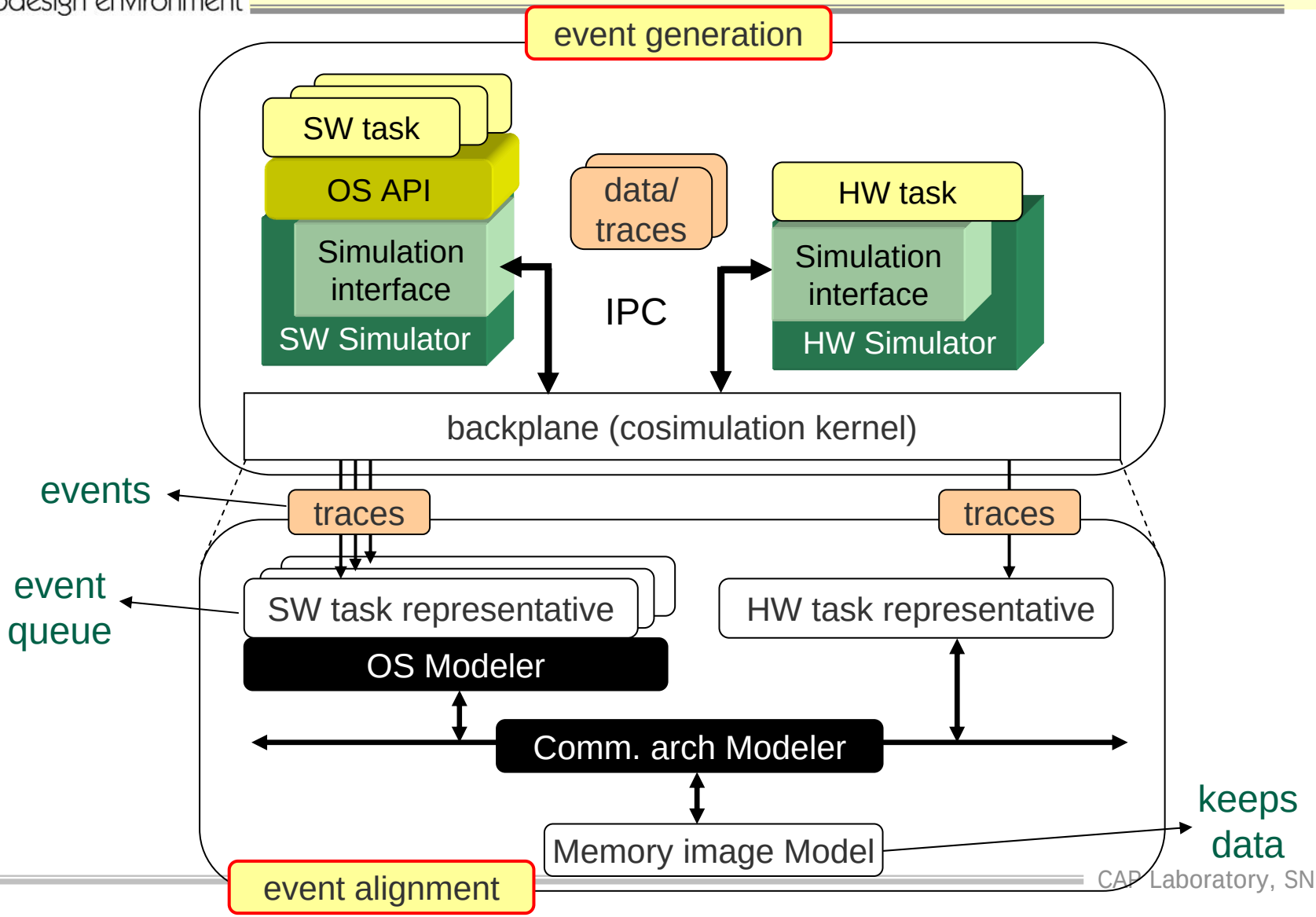
Basic Idea of VS

- **Optimistic event generation**
  - Simulators run optimistically generating events in form of traces
    - *Until it encounters shared memory access (=data synchronization)*
    - *Until it blocks*
    - *Until it ends*
    - *No rollback since no "past event" occurs*



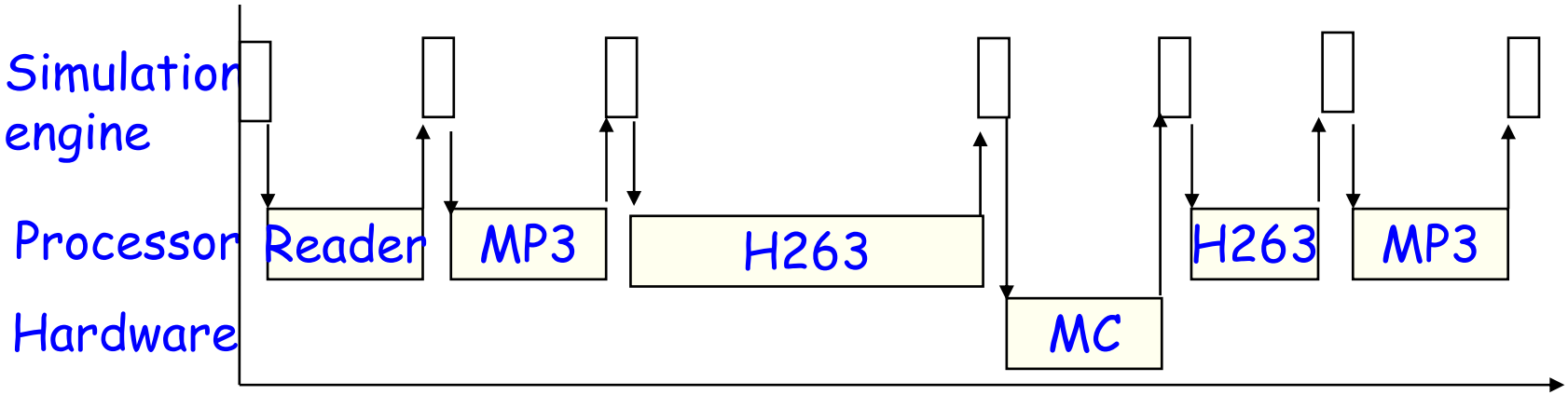
- **Conservative event alignment**
  - Cosim. kernel maintains global clock
    - *Reconstruction of relative difference into the global time*
  - Cosim. Kernel conservatively aligns the generated events (=traces)
  - If an event queue becomes empty, then it schedules the simulator

Virtual Synchronization Framework



Cosimulation for Trace Generation

- **Simulation engine executes simulators by functional dependency assuming that there is no dynamic behavior**
- **The execution continues until it is blocked by data or by period while the memory model stores execution traces**
  - Execution traces = memory traces + behavior traces
- **No time synchronization is needed during the execution**

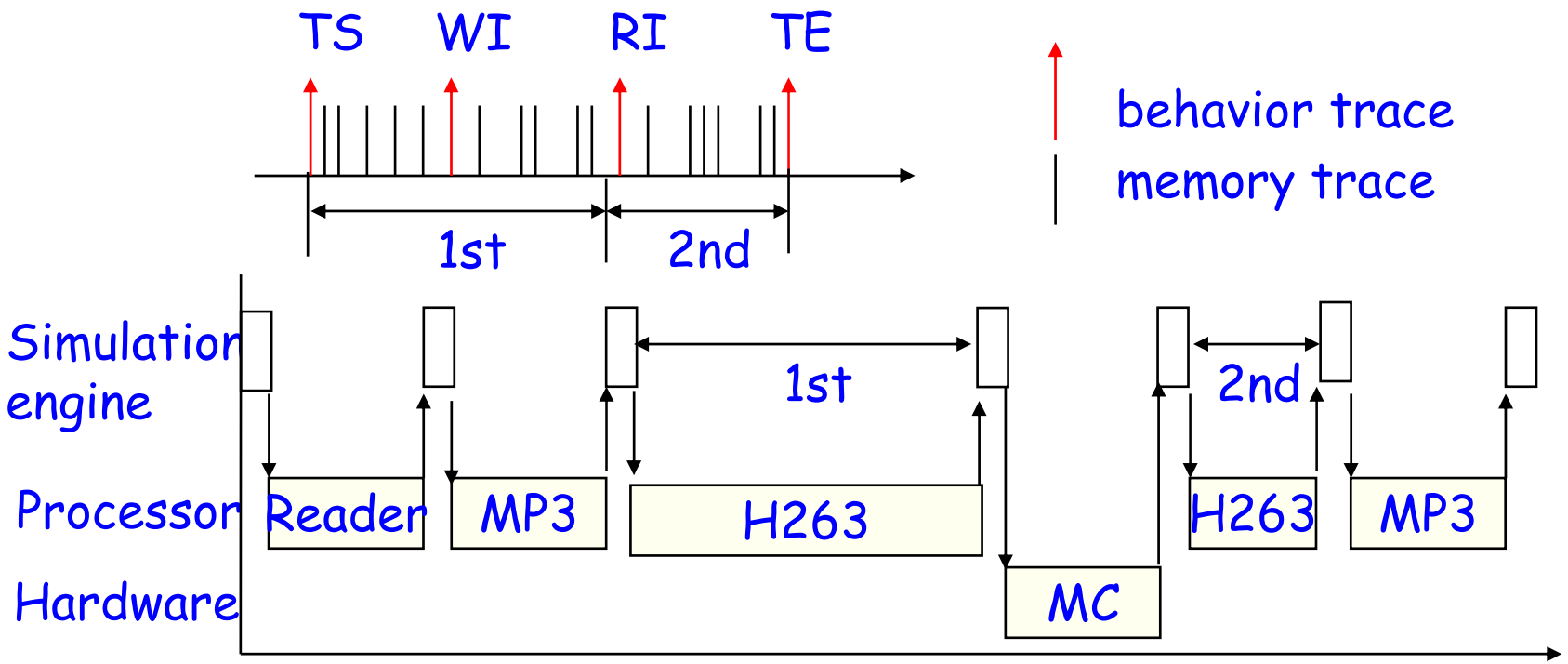


Trace Information

- **Memory model captures memory traces and behavior traces**
- **OS API and IF block writes behavior traces at the special address**
  - Dynamic behavior from data synchronization :
    - *blocking by unavailable data when receiving data*
    - *blocking by buffer overflow when sending data*
  - Dynamic behavior from operating system :
    - *tick interrupt, IO interrupt, preemption*
  - Dynamic behavior from communication architecture :
    - *bus arbitration, bus contention, memory delay*

Execution Traces from H.263

- Each trace does not have an absolute time but has a relative time (time difference) compared to the previous trace
- Therefore simulators do not need to advance its local clock when they have nothing to execute



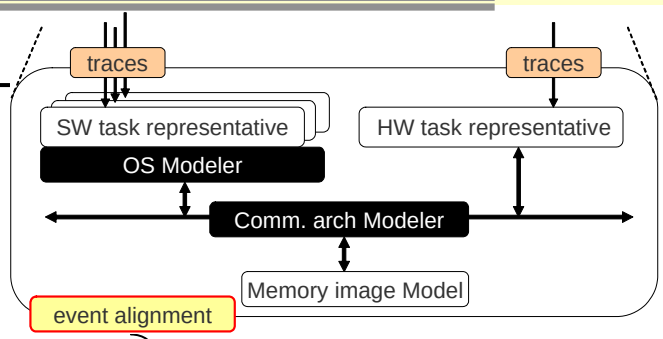
Event Alignment

- **OS Modeler**
  - Preemptive RTOS scheduler may change task execution sequence
  - Simulators execute task non-preemptively in TDVS
    - ← *No synchronization*
  - RTOS scheduling such as preemption and blocking is modeled while aligning events

- **Communication Architecture Modeler**
  - Memory latency or bus contention is modeled while aligning events or can be directly simulated

Scheduling Algorithm

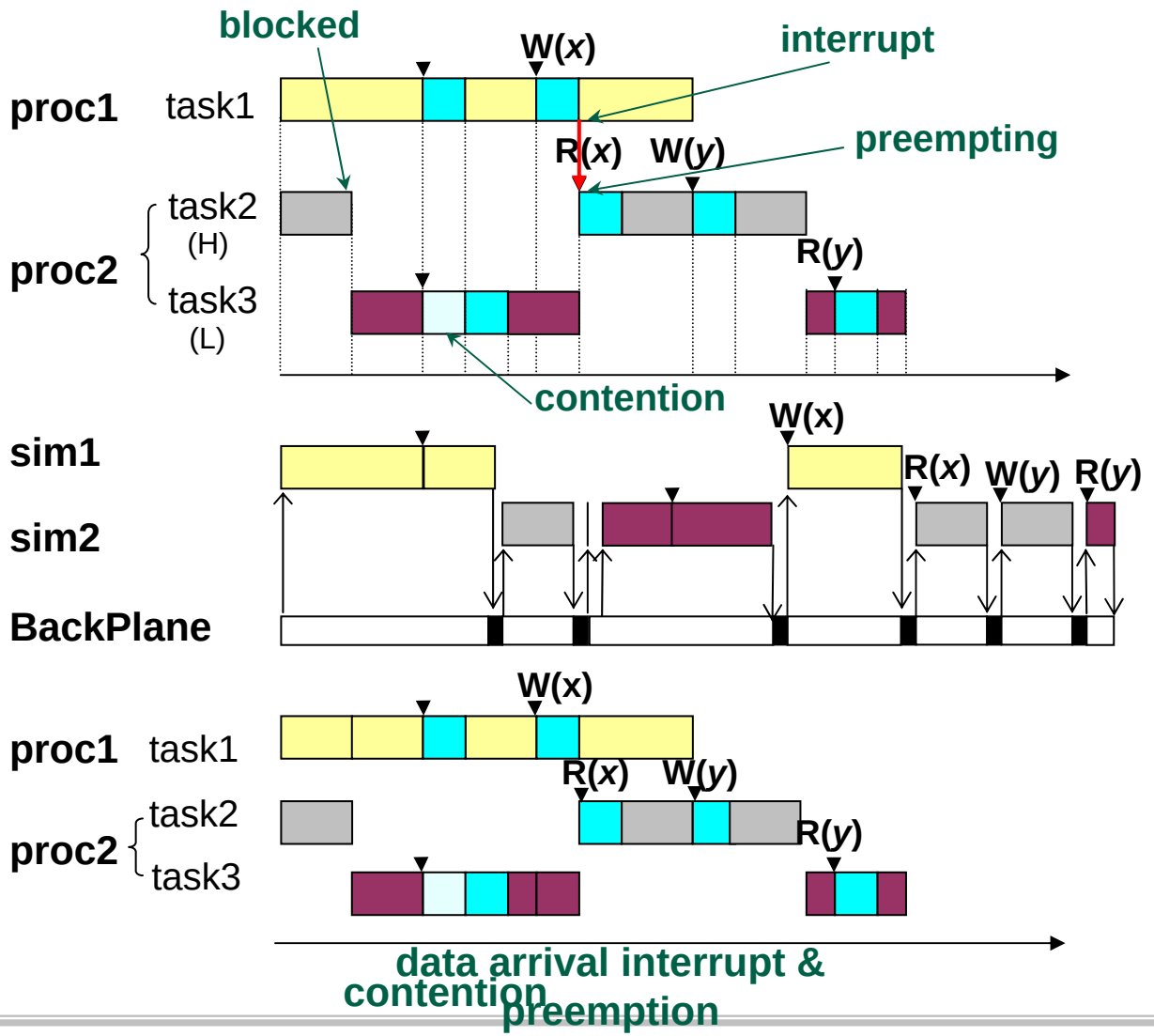
```
1    while (cosim_end==false) {       // for each trace
2        for (i=start_idx; i<numComp; i++) {
3            task = OS_Modeler( i )
4            if (task->trace == NULL) {
5                start_idx = i;
6                return;      // activate event-generation phase
7            }
8            if (GT(task->trace->time) < min_access_time)
9                min_access_time = GT(task->trace_time);
10               min_task = task;
11           }
12       }
13       CommArch_Modeler( min_task->trace );
14   }
```



Gaurantee no *past event*

Find out system-wide earliest event

Cosimulation Scheduling Example



Memory access latency

Contention delay

W(*x*): Shared memory write at address ***x***

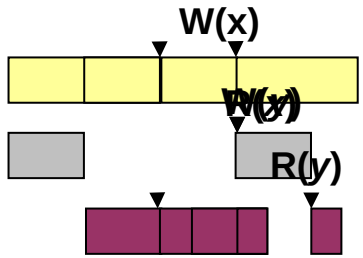
R(*x*) : Shared memory read at address ***x***

▼ : Local memory access

→ synchronization

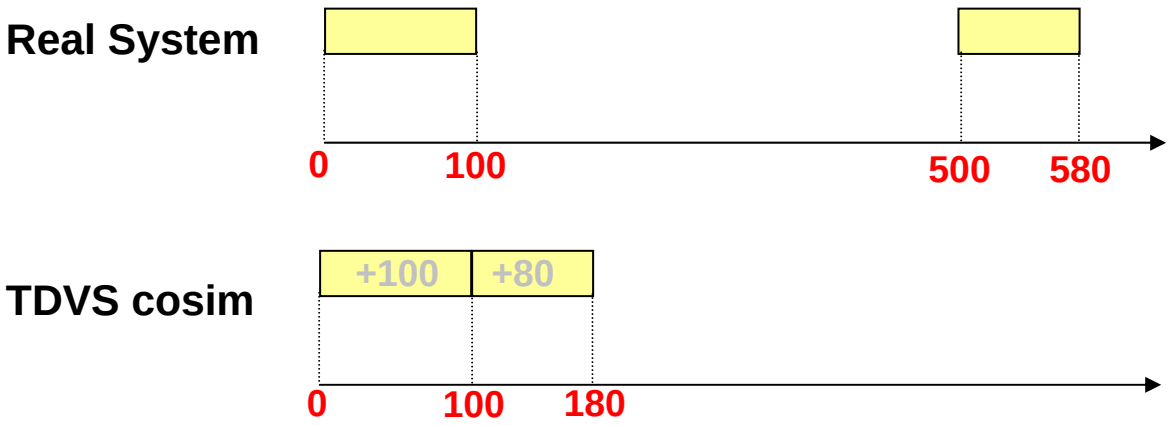
Event generation

Event-alignment



Removal of Idle Duration

- **Idle duration**
  - Task is not executed since it is blocked (=no input data)
  - Task is not executed since its period has not come yet
- **We can remove idle duration by executing simulators only when their tasks are active**
  - Fast forward effect of simulator clock



Performance Gain of VS

- **+ Reduction of # of synchronizations**
- **+ Removal of Idle duration simulation**
- **+ Removal of local memory transaction simulation**

- **- Trace management overhead**
  - Traces are not stored in files → very small overhead

u_i : Utilization of simulator i

sc_i : Synchronization counts of simulation i

$$\sum_{\forall i} \{T \times u_i \times st_i + sc_i \times sync\} + tran_{num} \times st_{tran}$$

Cosimulation Time with VS

- **Simulation time on processor**
  - 5000000 * 0.88 / 1000000 = 4.4 s
- **Simulation time of hardware**
  - 5000000 * 0.0356 / 10000 = 17.8 s
- **Simulation time of communication architecture**
  - 332021 / 1328922* = 0.2 s * from the experiment
- **Total Simulation Time = 22.4 s → 43 times better!!**

- **51% performance gain without idle duration**
  - 12% for ARM922T and 94% for FPGA,
- **42% without time synchronization**
- **5% for efficient implementation of communication architecture**

Assumption and Limitation

- **Assumption of Trace-Driven Virtual Synchronization**
  - Relative time difference between events is not changed as the memory system changes.
  - The assumption may not hold in case of out-of-order issue processor where instructions are scheduled dynamically

- **Possible timing inaccuracy due to cache in ISS**
  - Cache state in ISS becomes different from reality in case of multi-task system simulation with preemptive scheduler.
  - ➔ Disable cache simulation in ISS and perform cache simulation in the backplane
  -

Experiment

- **H.263 decoder example**
  - ARM926ej-s x 3, IDCT x 3 (Y,U,V)
    - *ARM(0) = H263Reader, ARM(1) = VLD, Dequant, InvZigzag, etc.*
    - *ARM(2) = MotionCompensation, DisplayFrame*
  - ARM926ej-s = ARMuator, IDCT = SystemC IP, simulation host = Xeon 1.8GHz
  - arm-elf-gcc 3.4.5 with O3 option
  - QCIF 3 frames (I, P, P)

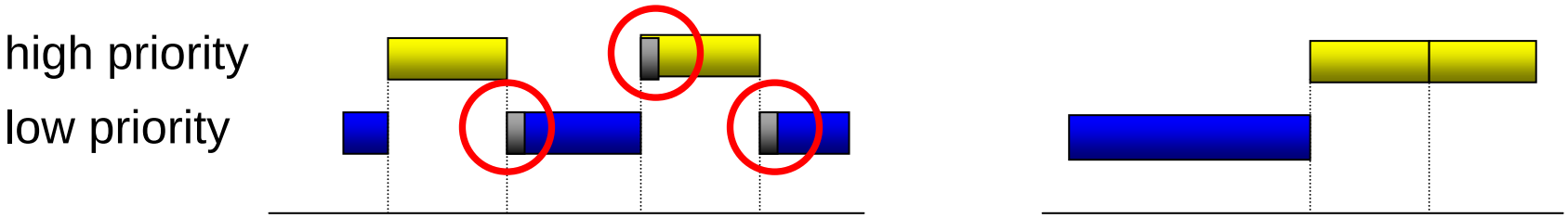
	Simulated Cycles	Simulation Time (sec)	Speed(cycles/sec)	Error	Speedup
MaxSim 6.0	**5,053,686**	**142.57**	**35,447**	**0%**	**1.00**
Proposed	**5,329,886**	**17.20**	**309,877**	**5%**	**8.74**

The RTOS Overheads

- **RTOS overheads under consideration**
  - Context switch and scheduler codes
    - *when a task voluntarily yields the control*
    - *when a task is preempted by another task*
  - Timer interrupt handler
- **The execution time of the timer interrupt handler varies slightly depending on the number of sleeping tasks**
- **The estimated values are obtained from the library after the initial measurement**

The Accuracy of the OS Modeling

- **Accuracy of this approach is dependent on the accuracy of the estimated RTOS overheads**
  - RTOS has the constant or bounded overhead
- **Cache is a major source of inaccuracy**
  - Cache related preemption delay
  - Smaller cache misses for the second instance when it preempts some task consecutively
  - This approach cannot accurately model the effect of temporal locality

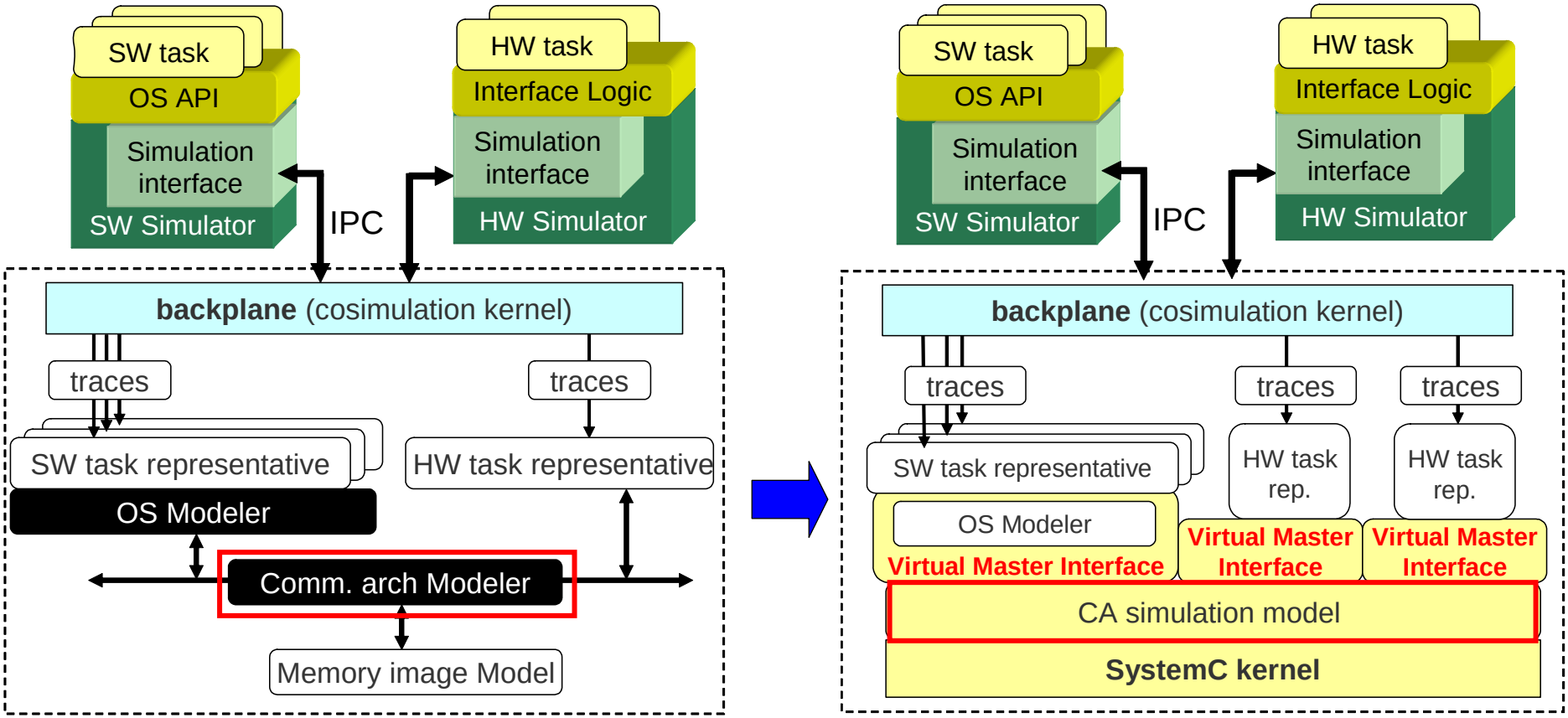


(a) In reality

(b) The proposed approach before time adjustment

Communication Architecture Modeling: Method 1

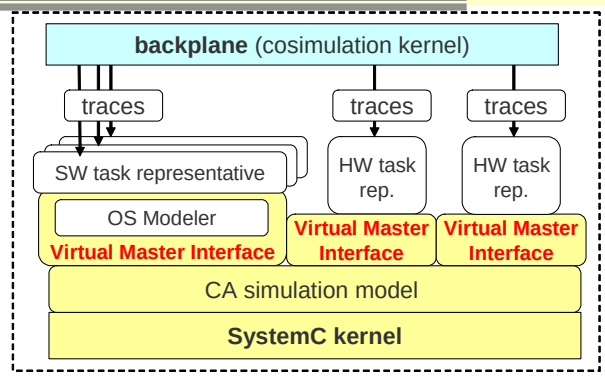
- **SystemC cosimulation environment is in charge of CA simulation**



Using SystemC with TDVS

Pseudo code of virtual master module

```
1    void MasterInter::run() {
2      while (!cosim_end) {
3        while (!sem_test (archi_sem));
4        // trace-driven architecture simulation phase
5        tr = get_earliest_trace(); // OS modeling included
6        if (tr->==NULL) {
7          sem_p (archi_sem);
8          //invoke trace generation phase
9          sem_v (tracegen_sem);
10         continue;
11       }
12       switch (tr->type)  {
13         case READ: ahbmst_read(addr, data); break;
14         case WRITE: ahbmst_write(addr, data); break;
15         ...
16       }
17       wait (tr->exec_time, SC_NS);
18   }}
```



- Choose trace to simulate
- If there is no more trace, request trace to backplane

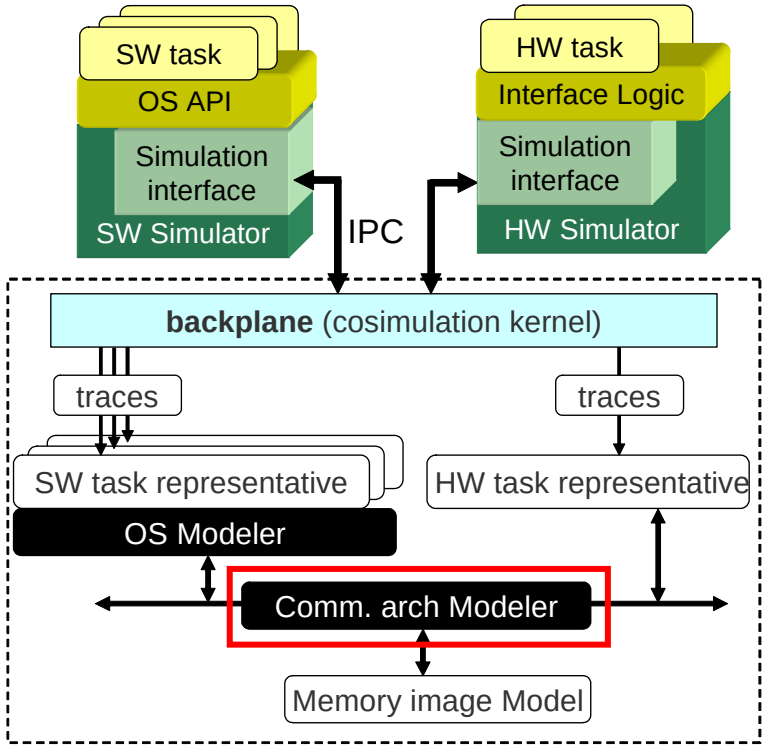
- Simulate SystemC CA model

- Clock increment in SystemC cosimulation; synchronization

CA Modeling: Method 2

- **SystemC suffers from low simulation speed, especially with BCA model**
  - Synchronization overhead within SystemC framework (i.e., thread context switch overhead)
- **C modeling approach**
  - Maintains high simulation speed of existing TDVS framework
  - Still provides cycle accuracy
  - Should consider transaction-order inversion due to bridge delay
- **CA details is specified in a XML file**
  - List of components in the platform
  - Attributes of each component (e.g. latency of memory)
  - Address map
  - Topology of platform (how components are connected)

Using C model with TDVS

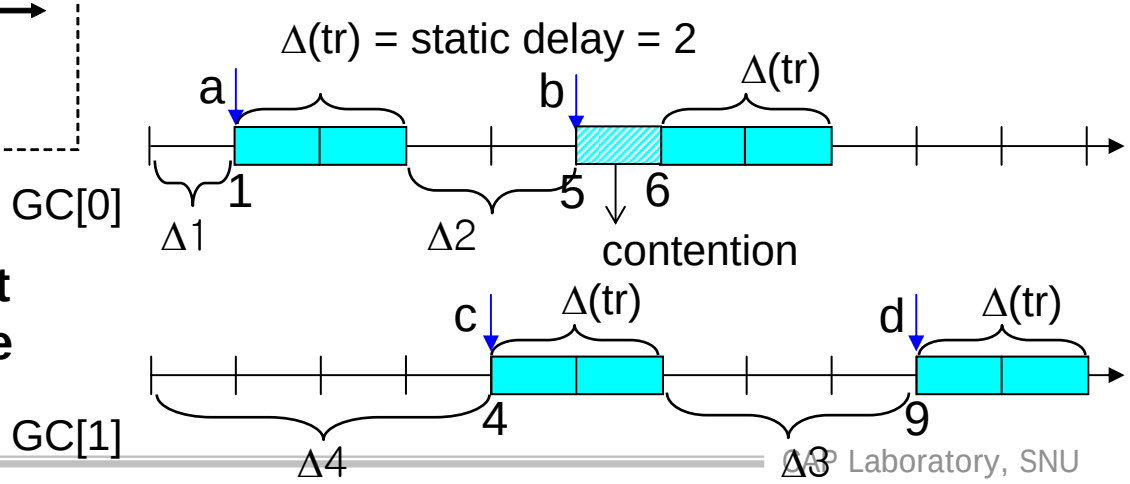


- GC[i] : global time of the component i
- Δ(tr) : static latency of the access, tr

1) Find GC[k] s.t. GC[k] = min(GC[0],..GC[n])

2) GC_ca = max(GC_ca, GC[k])+ Δ(tr)

eventQ[0] | **b** = Δ2, **a** = Δ1 | **d** = Δ3, **c** = Δ4 | **eventQ[1]**

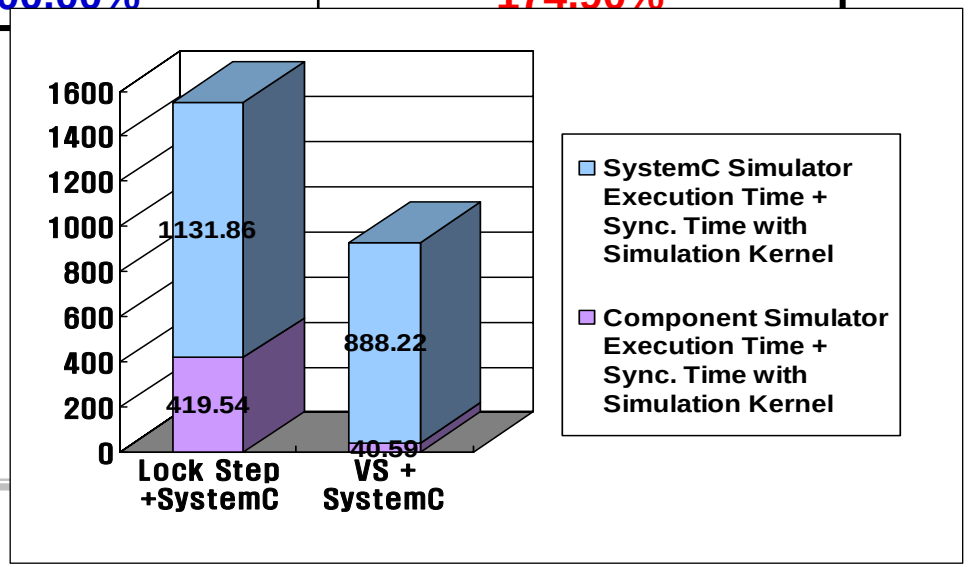


Event alignment in the backplane

Experimental result(1)

- **Verifying accuracy of VS**
  - JPEG decoder
  - Disable cache in order to simulate extensive contention

Dual Image	Lock Step + SystemC	VS + SystemC
Simulated Cycles	34,274,826	34,274,826
Simulation Time (sec.)	1551.401601	887.012023
Error	0.00%	0.00%
Perf. Improvement	100.00%	174.90%



Experimental result (2)

- **Verifying accuracy of C model : contention modeling**
  - JPEG decoder
  - Runs the same image on 2 processors

Single Image	VS + SystemC	VS + C Model
Simulated Cycles	20,997,500	20,531,185
Simulation Time	365.692 sec.	19.218 sec.
Error	0.00%	2.22%
Perf. Improvement	100.00%	1902.86%
Dual Image	**VS + SystemC**	**VS + C Model**
Simulated Cycles	26,008,700	26,136,879
Simulation Time	524.772 sec.	25.997 sec.
Error	0.00%	0.49%
Perf. Improvement	100.00%	2018.59%
Contention Effect	123.87%	127.30%

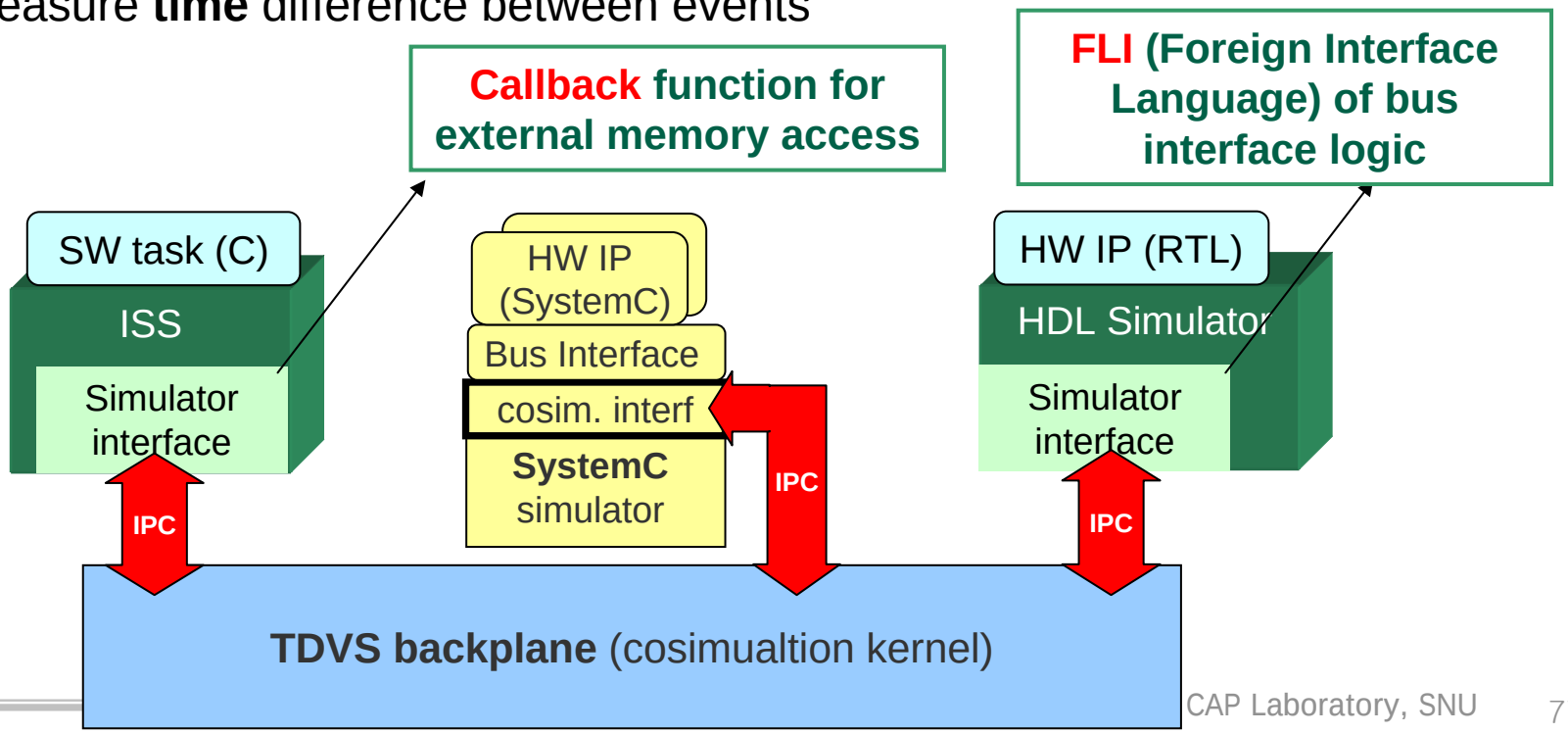
Experimental result (3)

- **Experiment on real-life example : H.263 decoder**
  - Cache enabled
  - Use 3 processors (IDCT1, IDCT2, all other jobs)

H.263 decoder	VS + SystemC	VS + C Model
Simulated Cycles	19,725,900	19,749,850
Simulation Time	332.119 sec.	20.359 sec.
Error	0.00%	0.12%
Perf. Improvement	100.00%	1631.31%

Mixed level Cosimulation

- **Cosimulation kernel = TDVS kernel**
- **Simulators interface**
  - Establish **connection** with the backplane
  - **Exchange** data and send traces
  - Generate **traces**
  - Measure **time** difference between events



Advance Issues of VS

- Parallel Cosimulation
- I/O Modeling
- CA modeling for NoC
- Support of Shared Memory Synchronization

VS Summary

- **VS Improves cosimulation performance**
  - By reducing # of synchronization
  - By not simulating idle duration
  - By parallel cosimulation

- **While mainitaining cosimulation accuracy with acceptable error below a few percents**
  - By OS modeling
  - By Communication Arch. modeling

This approach is complementary to

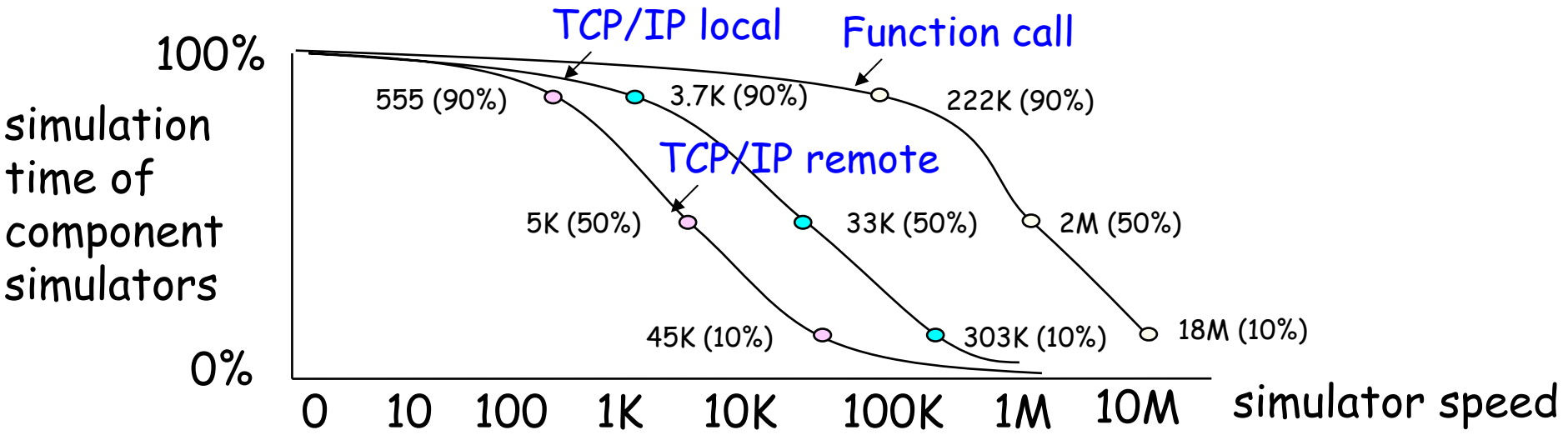
- Simulator speed improvement by TLM
- Synch. Cost : IPC→thread switch, function call

Summary

- **HW/SW Cosimulation is used for**
  - Virtual prototyping for SW development (Cycle-approximate TLM)
  - System performance estimation for DSE (Cycle-accurate TLM)
  - System verification before implementation (Cycle-accurate TLM or RTL)
- **Commercial tools for HW/SW Cosimulation**
  - TLM: SoC Designer from ARM, ConvergenSC from Coware, etc
  - RTL: Seamless CVE from Mentor Graphics
- **Cosimulation performance mainly depends on**
  - Component and communication architecture simulation speed
    - *(solution) TLM, Hardware emulator*
  - Time synchronization overhead
    - *(solution) conservative approach, optimistic approach, trace-driven simulation*
- **(Trace-driven) Virtual Synchronization Technique (TDVS)**
  - Separate functional simulation and timing arrangement
  - Remove the time synchronization overhead and idle duration of component simulation.
  - Enable mixed-level simulation and parallel simulation

Questions

- **1. Referring to the graph below, answer the followings assuming that the system consists of two components:**
  - (a) When can we gain speed-up by distributed simulation with two machines?
  - (b) What would be the maximum performance of cosimulation if the performance of component simulator is 2M cycle/sec ?



Continue…

- **2. Compare the SW simulation techniques in terms of speed, accuracy, and applicability**

- **3. Page 37 shows the performance equation of lock-step cosimulation method. Explain how the following technique can improve the performance.**
  - (a) TLM cosimulation
  - (b) Optimistic cosimulation
  - (c) Hardware emulation

- **4. For the virtual synchronization technique, answer the followings:**
  - (a) Basic idea
  - (b) Main cause of performance of improvement
  - (c) Sources of inaccuracy