Dist-gem5 Architecture

Illinois: Mohammad Alian , Daehoon Kim, Prof. Nam Sung Kim
ARM: Gabor Dozsa, Stephan Diestelhorst

Contents

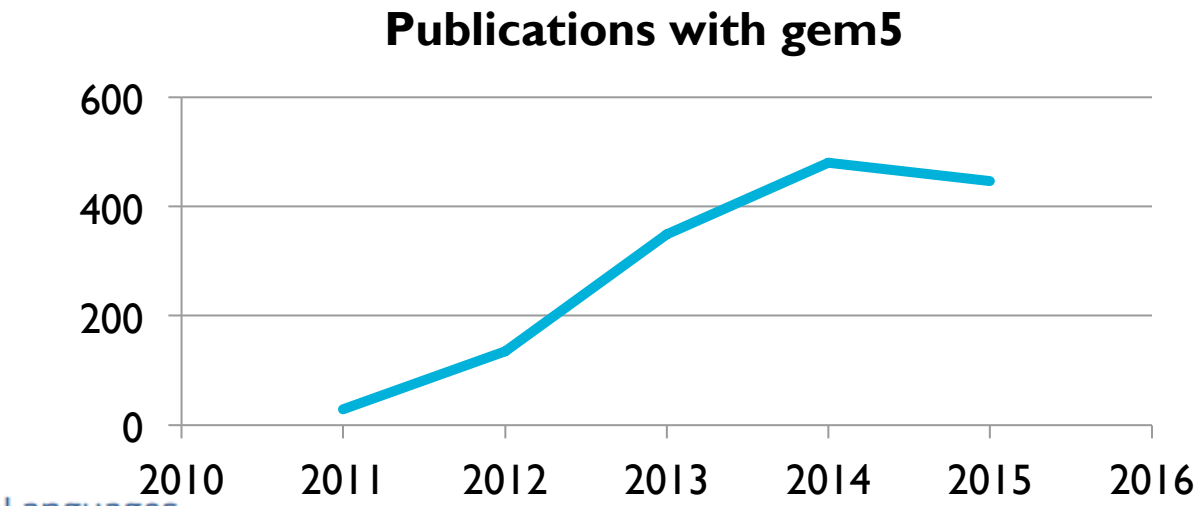
- Why distributed gem5?
- Core components
  - Packet forwarding
  - Synchronisation
  - Checkpointing
  - simulated Ethernet switch
- Deterministic execution
- Class and Object diagrams
- Use Case
- Conclusions and Future Work

What is gem5?

- Full-system, cycle-level simulator
  - Cores (in-order, out-of-order), caches, interconnects, DRAM, devices,…
  - ISAs: ARM, x86, …
  - Multi-core: Classical memory, GEMS / Ruby, …
  - Simulation modes: sampling, simpoints, traces, …
  - System topologies: single core … big.LiTTLE … multi-node HPC
- Generally plug-and-play
- Used extensively in universities and industry (ARM, AMD)
- Scenarios: Servers, mobile, client, HPC,

Users and Contributers

- Gem5 widely used in academia
- Gem5 contributions from
  - ARM, AMD, Google
  - Illinois, Michigan, BSC, ..



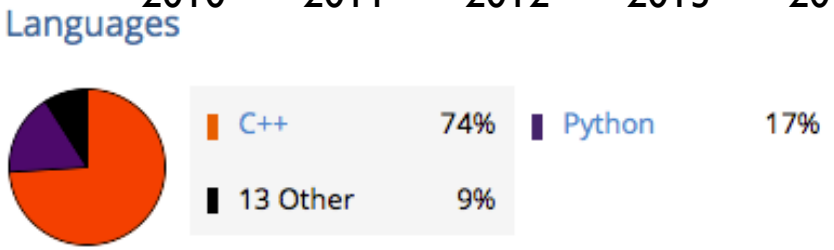
In a Nutshell, gem5...

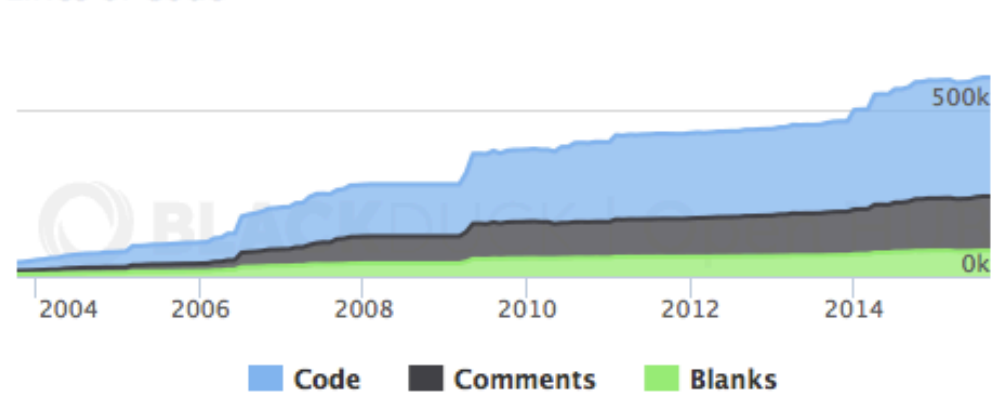
... has had 11,090 commits made by 167 contributors
representing 354,851 lines of code

... is mostly written in C++
with a well-commented source code

... has a well established, mature codebase
maintained by a very large development team
with increasing Y-O-Y commits

... took an estimated 95 years of effort (COCOMO model)
starting with its first commit in October, 2003
ending with its most recent commit 28 days ago



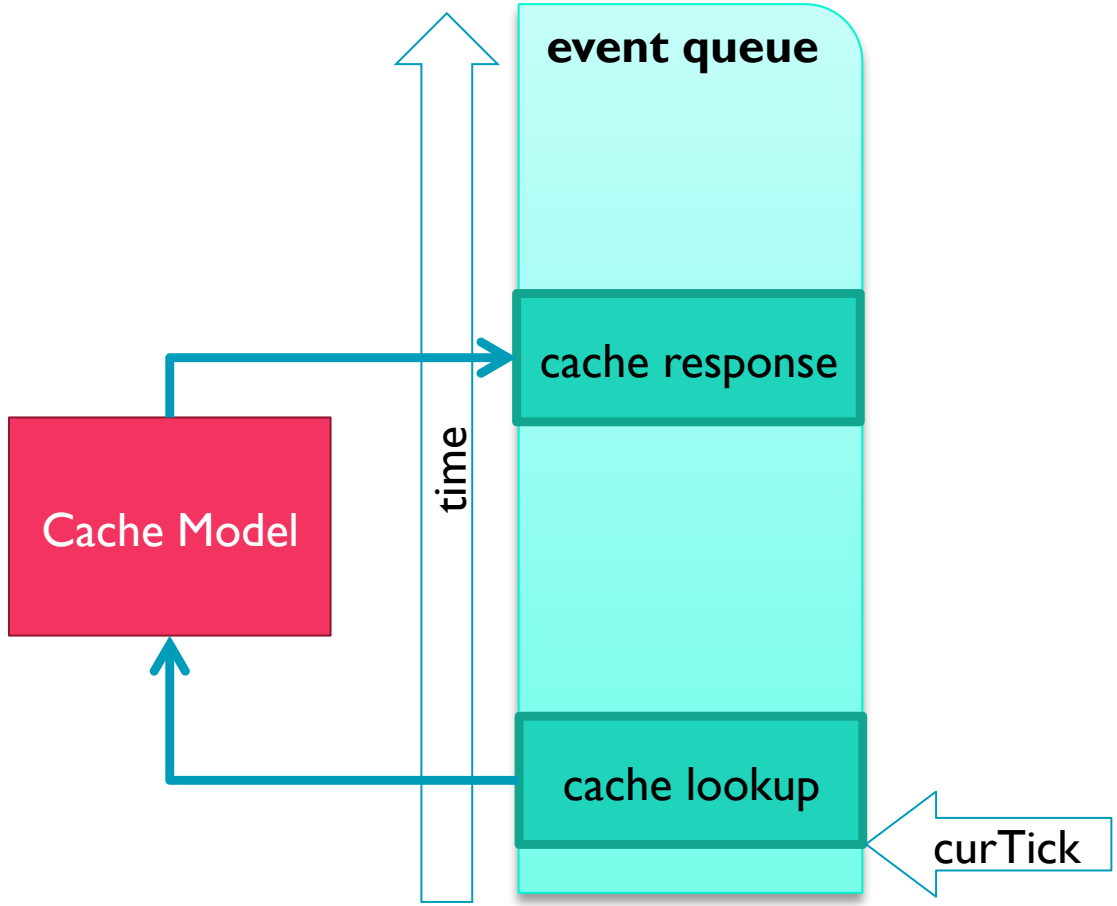


ECE IL
Department

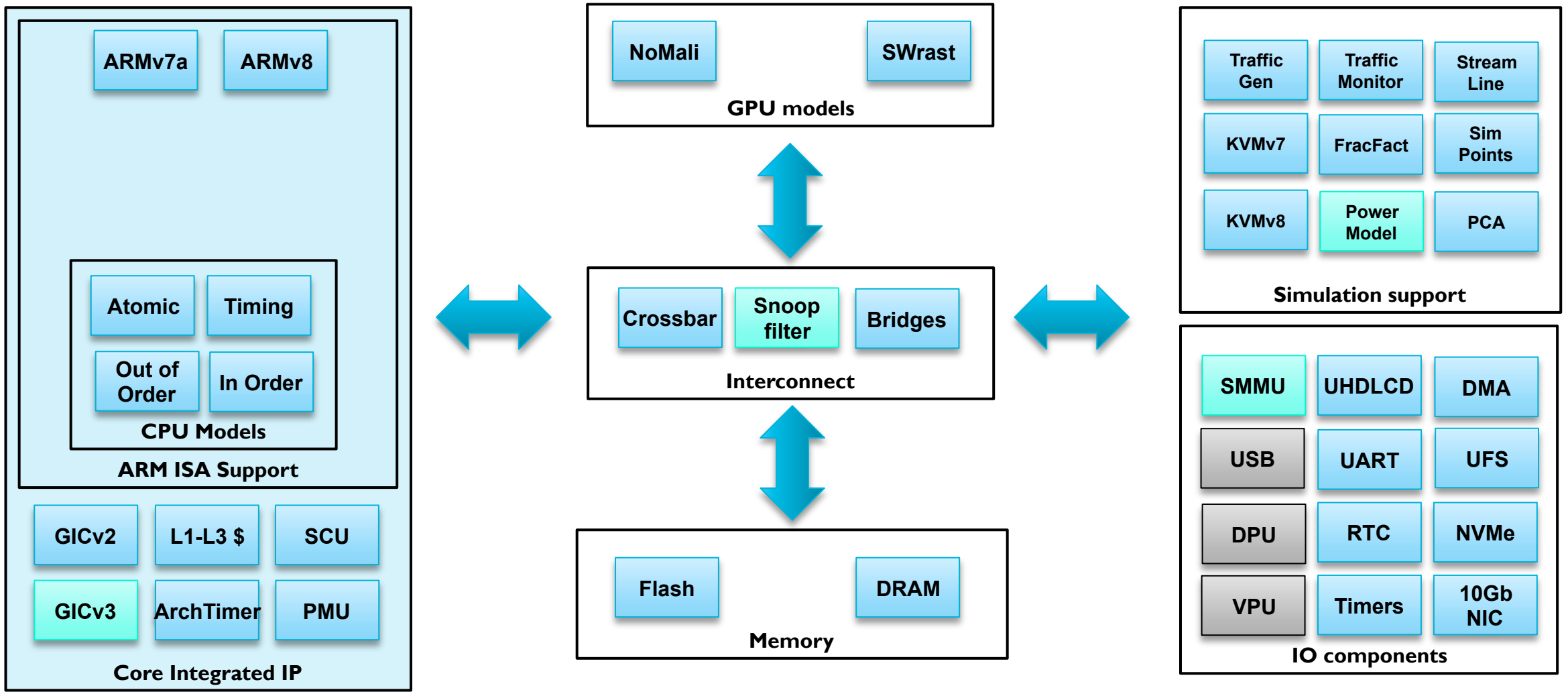
ARM

Event Simulation

- Event-driven
  - no activity -> no clocking
  - event queue
- Deterministic
  - fixed random number seed
  - no dependence on host addresses
- Multi-Queue
  - multiple workers



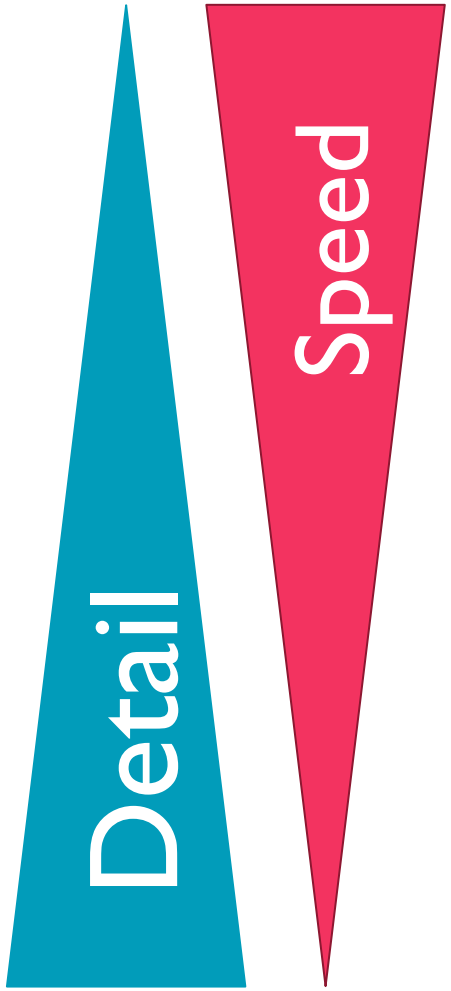
Component overview



Done | In Process | Planned

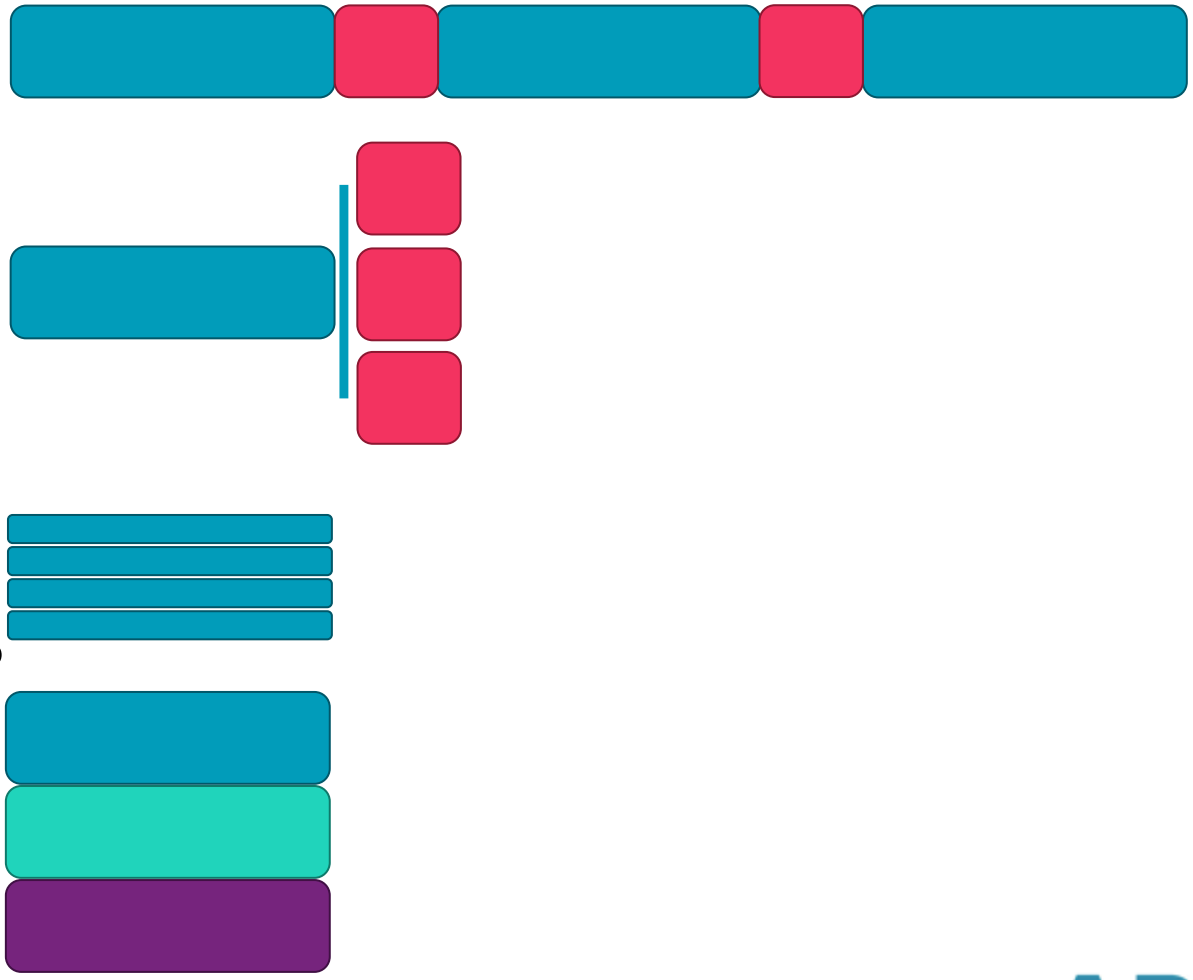
Level of Detail

- Functional mode
  - No timing, chain basic blocks of instrucitons
  - Can add cache models for warming
- Timing mode
  - Single time for execute and memory lookup
  - Advanced on bundle
- Detailed mode
  - Full out-of-order, in-order CPU models
  - Hit-under-miss, reodering, …



Accelerating gem5

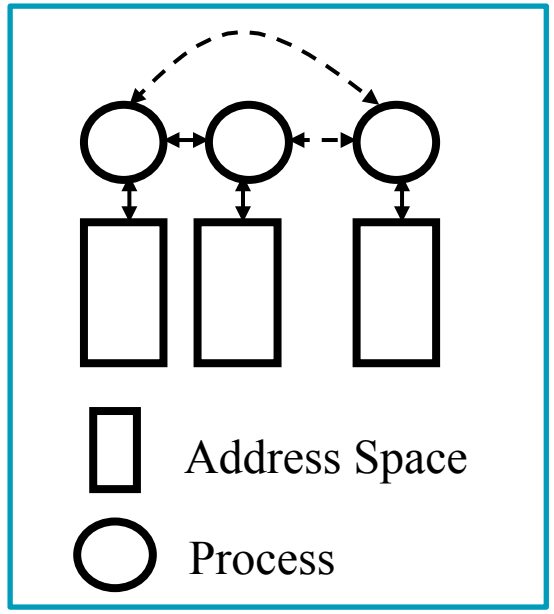
- Switching modes
  - (kvm +) functional + timing / detailed
- Checkpoints
  - boot Linux -> checkpoint
  - run multiple configurations in parallel
  - run multiple checkpoints in parallel
- Multi-threading
  - multiple queues
  - multiple workers execute events
  - data sharing and tight coupling limits speedup
- Multi-processed gem5
  - for design space explorations



Large-Scale Simulation

The Problem

- Design space exploration for future HPC systems requires simulators to cope with scalable benchmarks
  - e.g. MPI proxy apps from co-design centers (Lulesh, CoMD,…)
- Scale out efficiency related research questions
  - What would be the performance implications of using better/worse network links, NICs, etc. ?
  - What would be the optimal end-to-end latency of the system for a particular parallel application ?
- **Enable gem5 to simulate distributed memory systems on real clusters**

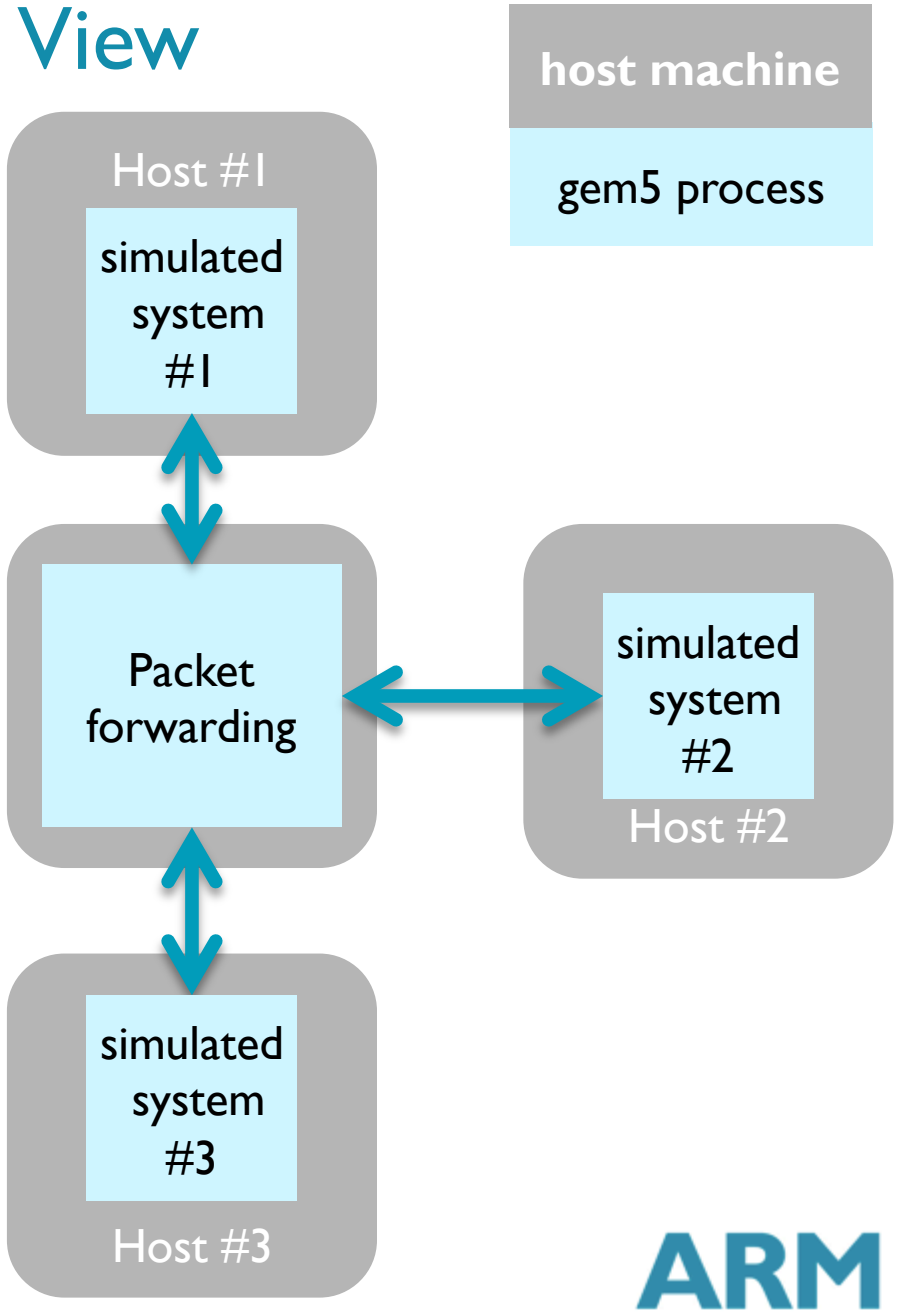


Distributed Memory

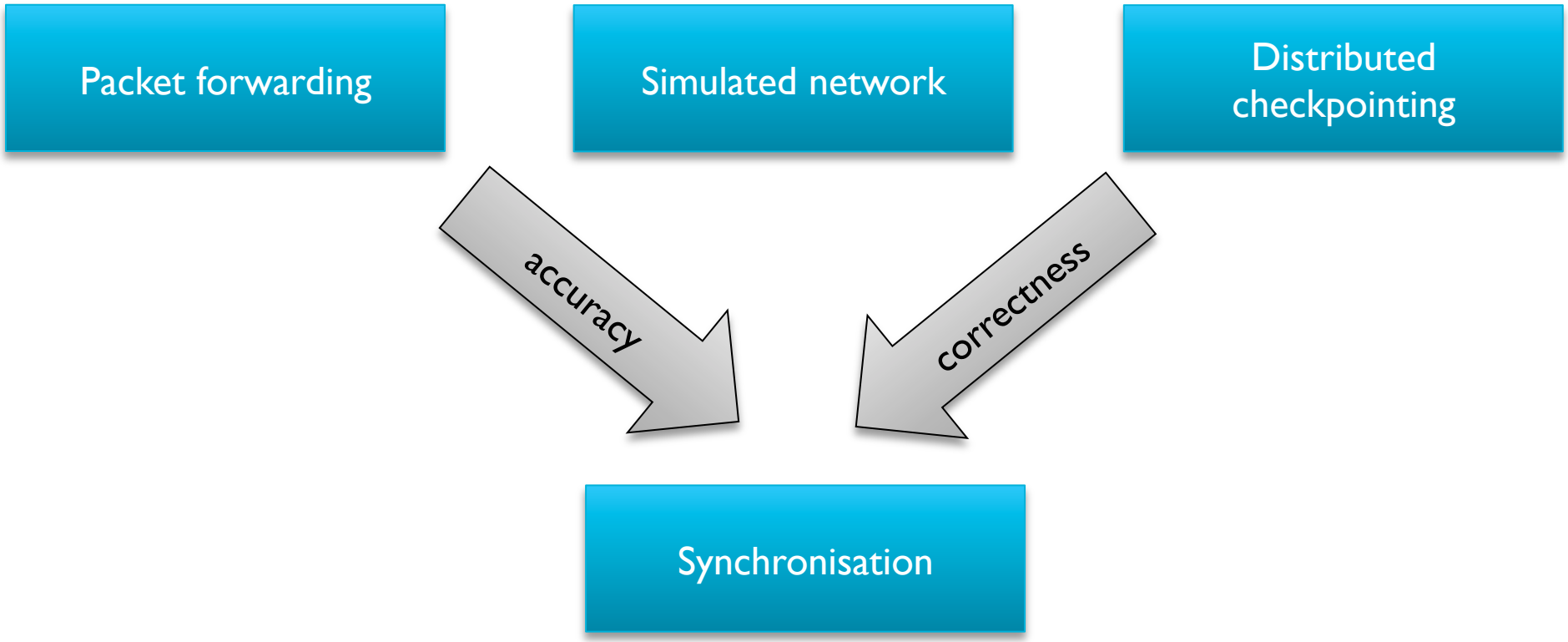
Message Passing

Distributed gem5 Simulation – High Level View

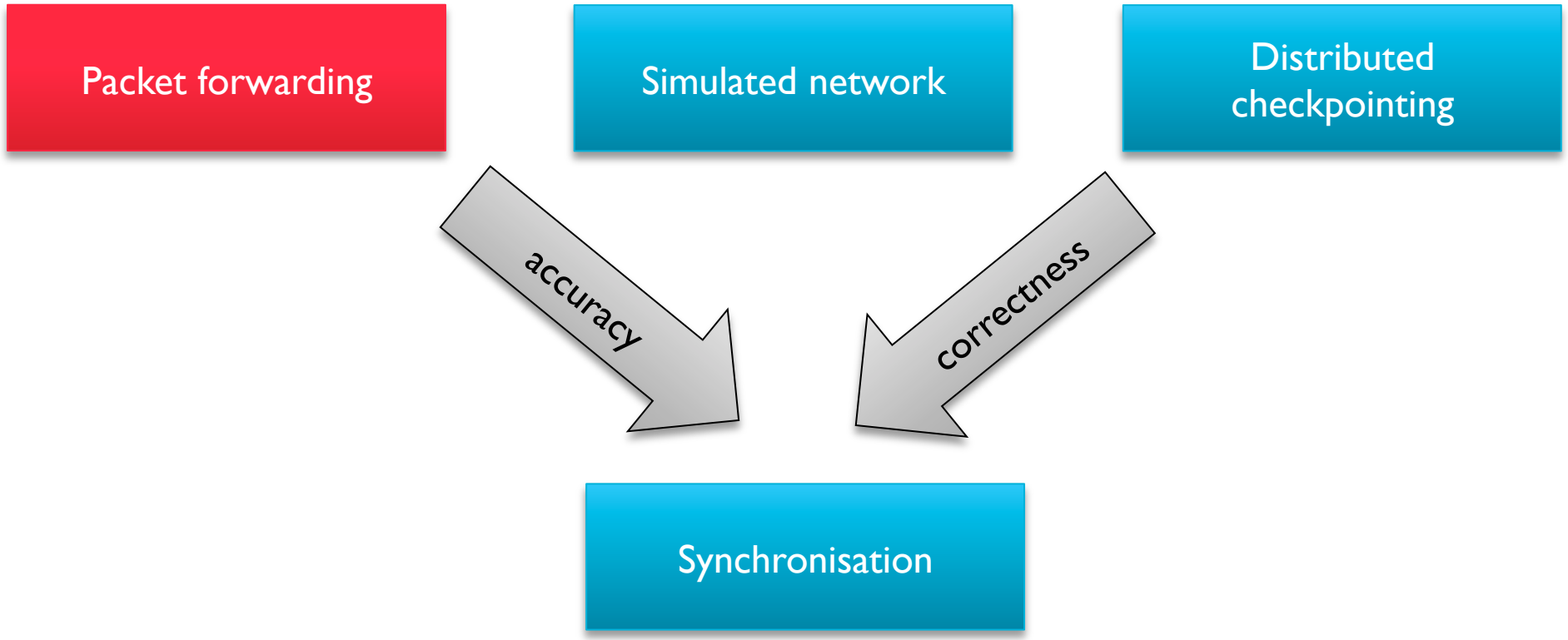
- gem5 processes modeling full systems run in parallel on a cluster of host machines
- Packet forwarding engine
  - Forward packets among the simulated systems
  - Synchronize the distributed simulation
  - Simulate network topology



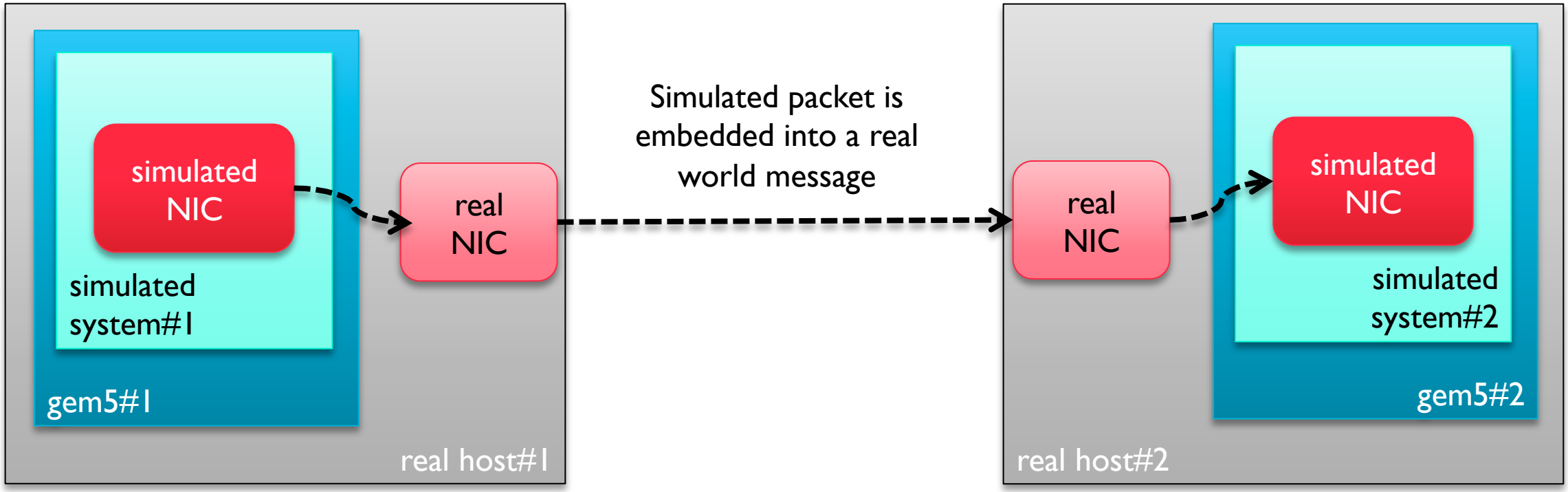
Core Components



Core Components

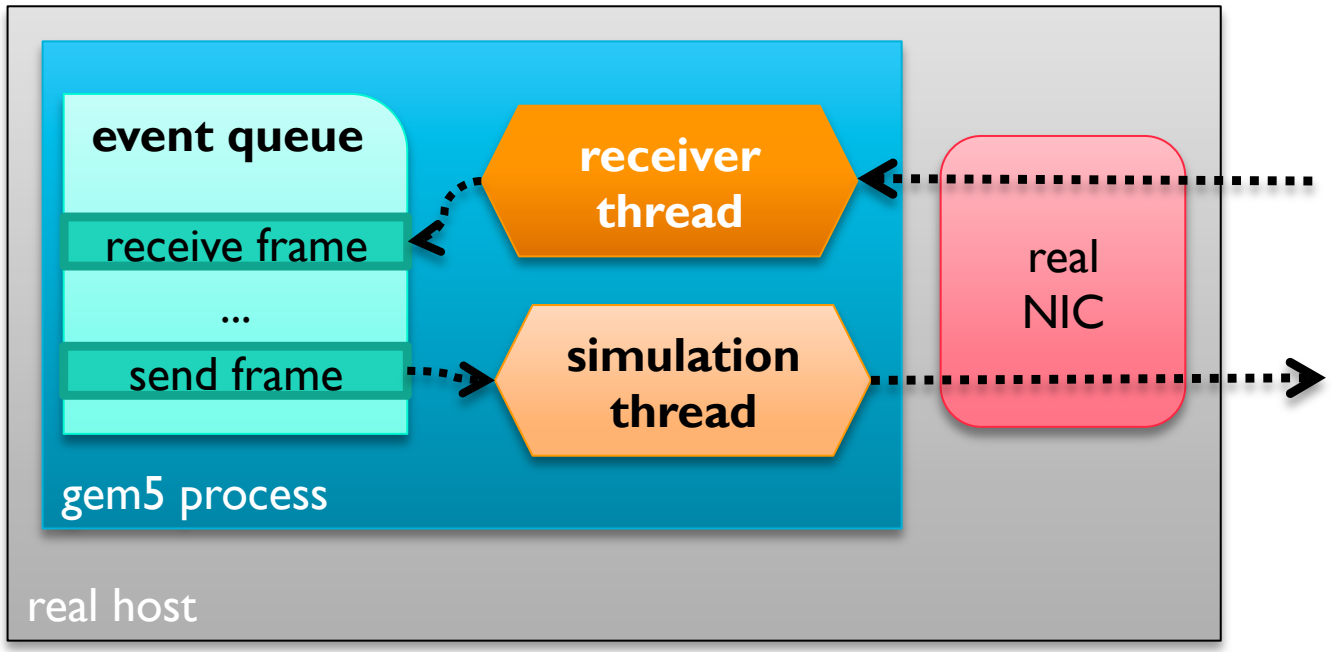


Packet Forwarding



Asynchronous Processing of Incoming Messages

- Simulation thread (aka main())
  - Part of vanilla gem5
  - Process events in the event queue (and inserts new events in the queue)
  - In case of a 'send frame' event encapsulates the simulated Ethernet frame in a message and send it out
- Receiver thread
  - Created for each dist-gem5 process
  - Waits for incoming messages
  - Create a 'receive frame' event for each incoming message and insert it in the event queue



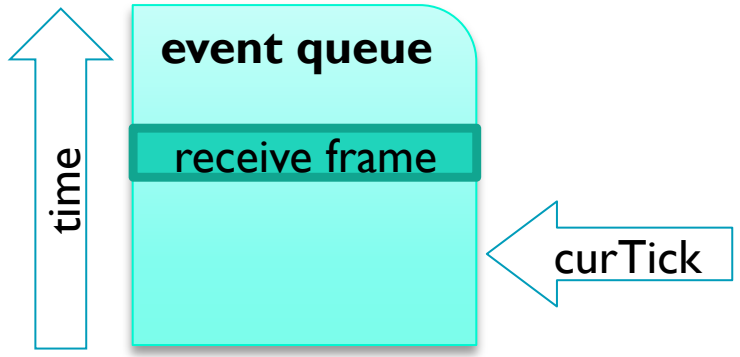
Simulation Accuracy and Packet Forwarding

- What is the correct tick for the receive event?
  - *st* : send tick
  - *lat*: simulated link latency
  - *bw*: simulated link bandwidth (bytes/tick)
  - *m*: simulated packet size (bytes)
  - *rt*: receive tick

Head

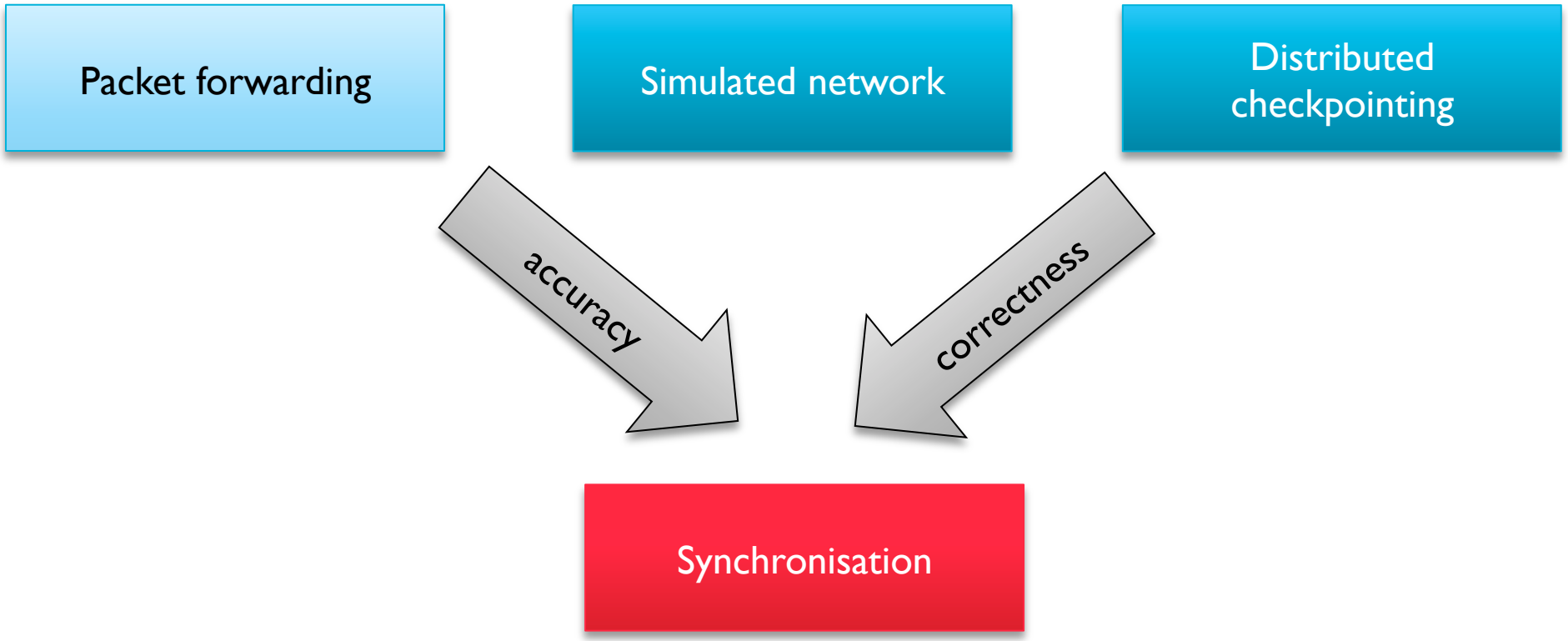
rt = st + lat + bw * m

Tail



- Accurate simulation
  - rt >= curTick() when the receiver gem5 gets the real message encapsulating the simulated packet
  - receiver gem5 can schedule the receive event for the simulated NIC

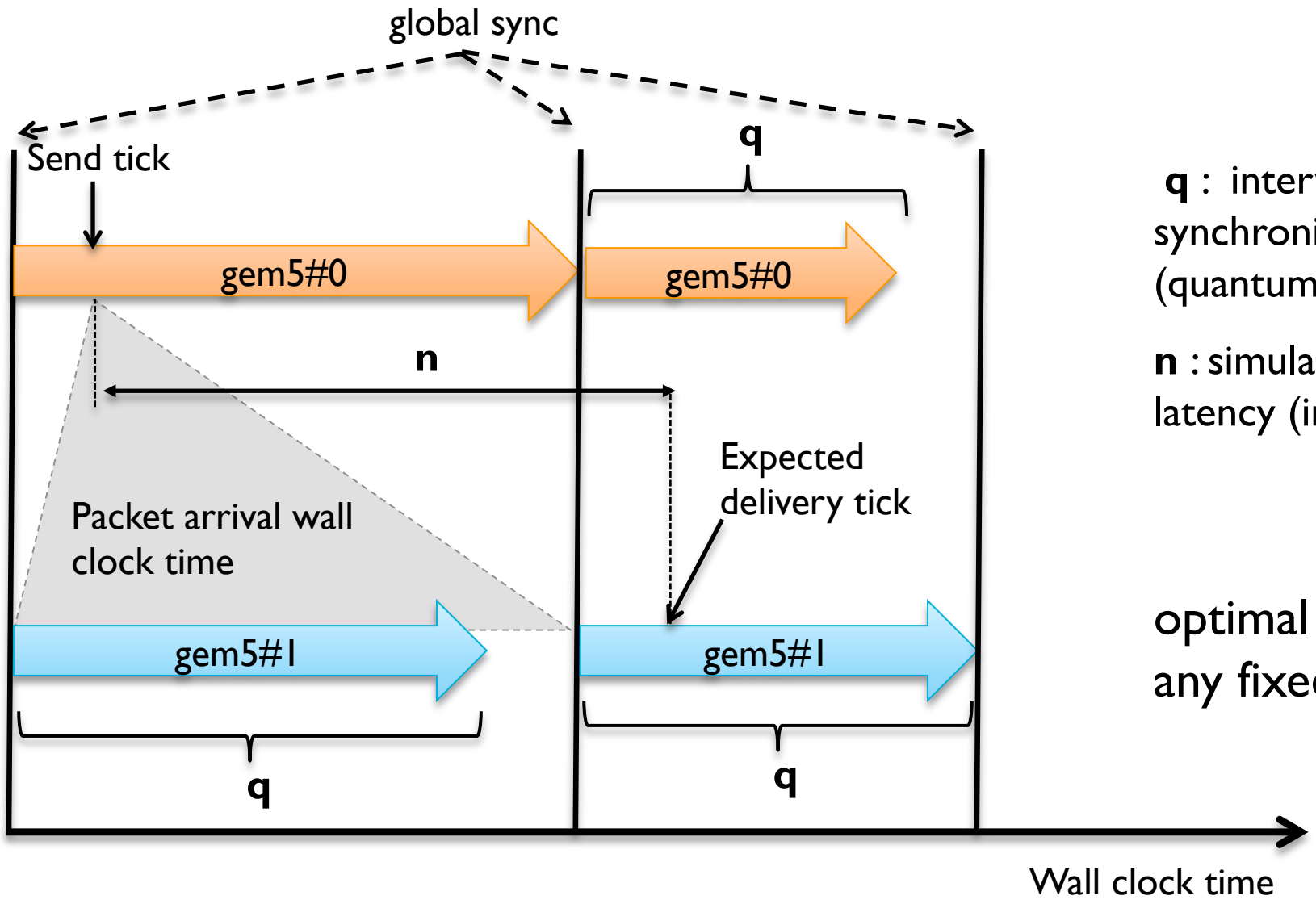
Core Components



Synchronisation

- Sender and receiver gem5 progress the simulation independently
  - Receiver may have less events to process => can run ahead of sender too much
  - curTick() may already be larger then the desired receive tick when message arrives
- Receiver gem5 may need to get "slowed down" to ensure simulation accuracy
- Synchronisation : periodic "barrier" to complete both by sender and receiver
  - global "sync" event
  - receiver and sender wait for each other at specific ticks
  - curTick() in sender and receiver are kept "close enough" at any point in (wall clock) time
- Synchronisation incurs overhead
  - Try to do as few global sync as possible while still maintain accuracy

Accurate Packet Forwarding



q : interval for periodic synchronisation in ticks (quantum)

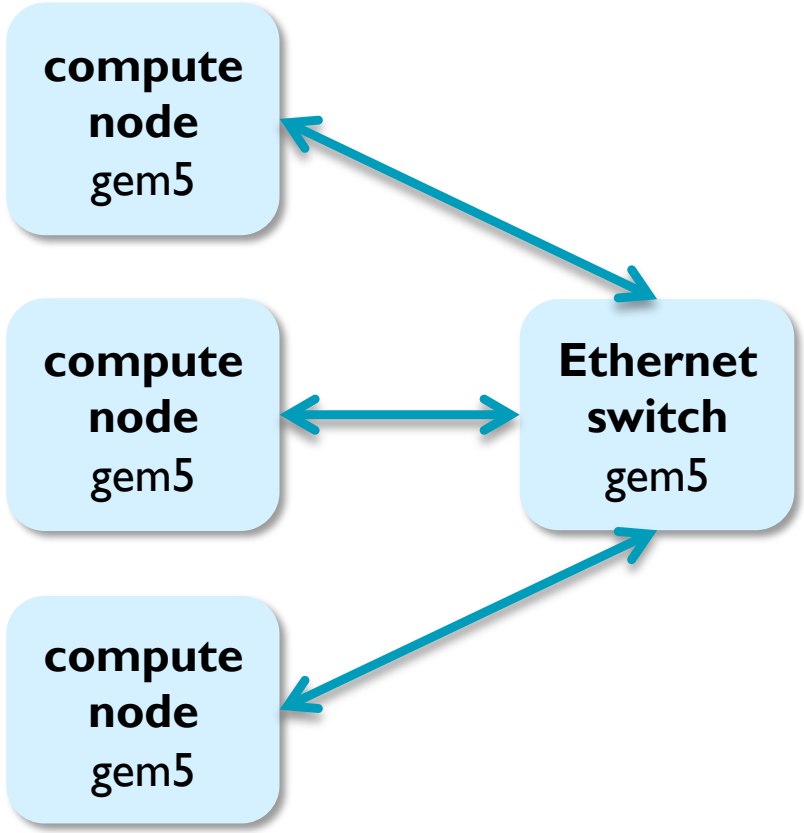
n : simulated network link latency (in ticks)

$$q \leq n$$

optimal **q**: **q** == **n** for any fixed **n**

Compute Nodes, Switch and Synchronisation

- Simulation progress gets stopped at each sync tick in each gem5 process
- Simulated compute node
  - Sends out 'synq request' message
  - Waits until 'sync ack' message comes back
- Simulated switch
  - Waits until it receives a 'sync request' message
  - Sends out 'sync ack' message



The Global Sync Event

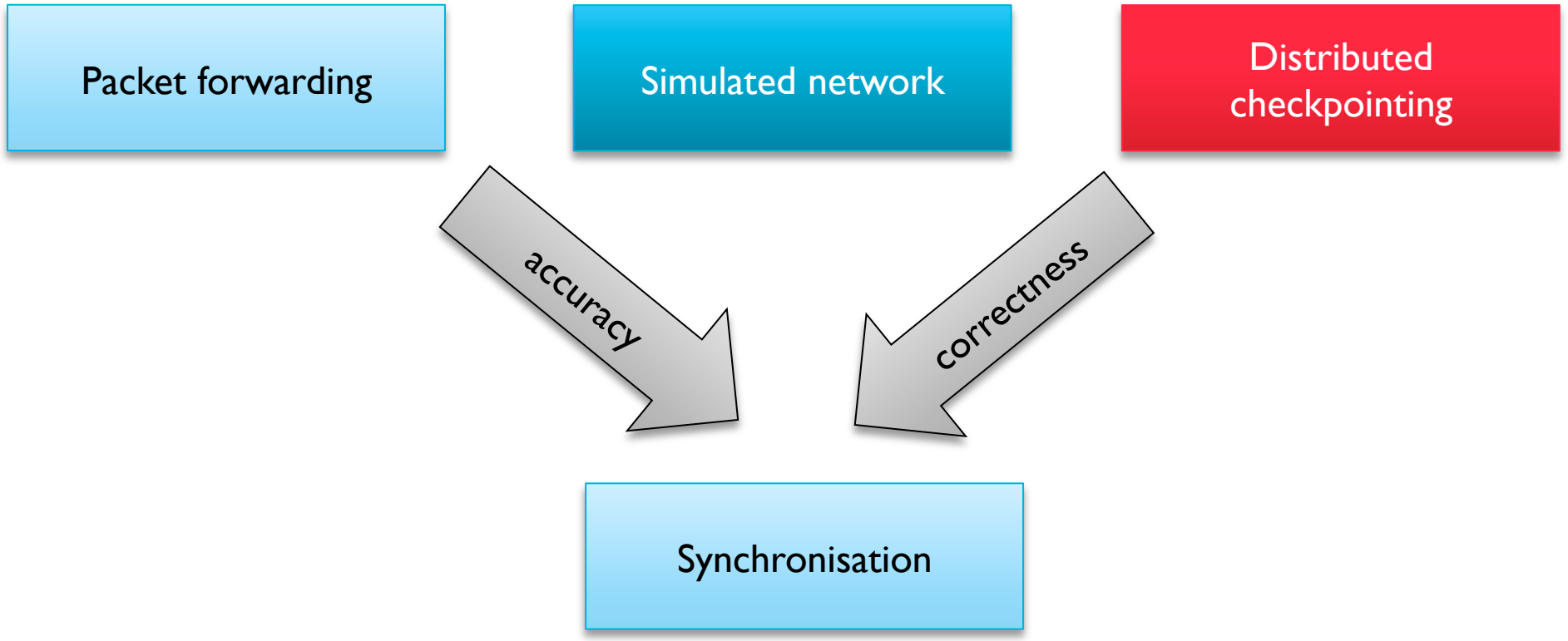
- A vanilla global gem5 event is scheduled at each sync tick in each gem5 process
  - A global gem5 event is a transparent thread barrier (in case of multiple simulation threads)
  - dist-gem5 global sync is prepared to work with multi-queue/multi-threaded gem5 simulations

- The process() method in a **compute node**
  - sends out '**sync request**' messages for each simulated link
  - waits on a condition variable to get notified about completion by the receiver thread

- The process() method in a **switch**
  - waits for completion notification from the receiver thread
  - sends out '**sync ack**' messages for each simulated link

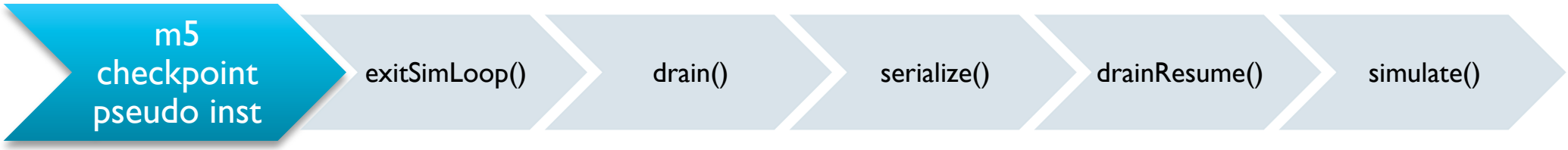
- Receiver thread keeps processing incoming messages while simulation thread is blocked
  - creates receive events in the event queue for simulated Ethernet frames
  - notifies blocked simulation thread when '**sync ack**' messages arrive
  - notifies blocked simulation thread when '**sync request**' messages arrive

Core Components

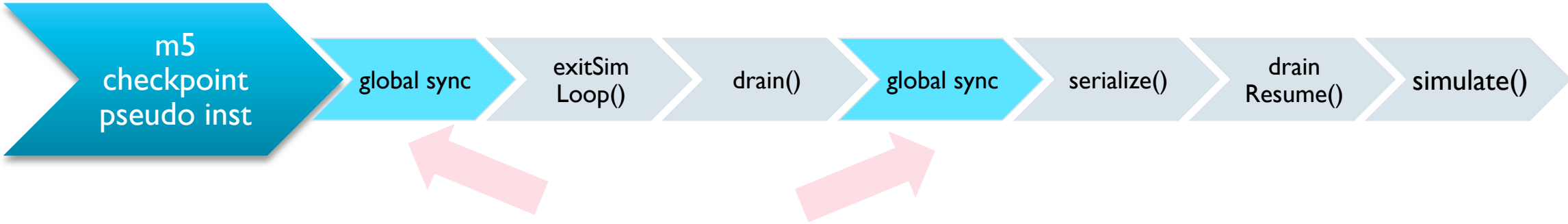


Distributed Checkpointing

- Checkpoint support for dist-gem5 relies on the mainline gem5 checkpoint support
  - Each gem5 process of a dist-gem5 run creates its own checkpoint



- dist-gem5 adds an extra co-ordination layer to ensure correctness
  - No in-flight message may exist among gem5 processes when the distributed checkpoint is taken



Distributed Checkpointing (cont.)

- Checkpoint can only be initiated at a periodic global sync
  - Simplifying implementation without scarifying usability

Checkpoint flavour	collaborative checkpoint	immediate checkpoint
Condition	all compute nodes signal intent	at least one compute node signals intent
Example use case	Instrumented MPI application source code to take a checkpoint at the MPI_barrier() before ROI	Taking a checkpoint from the bootscript before starting an MPI application (i.e. before calling 'mpirun')

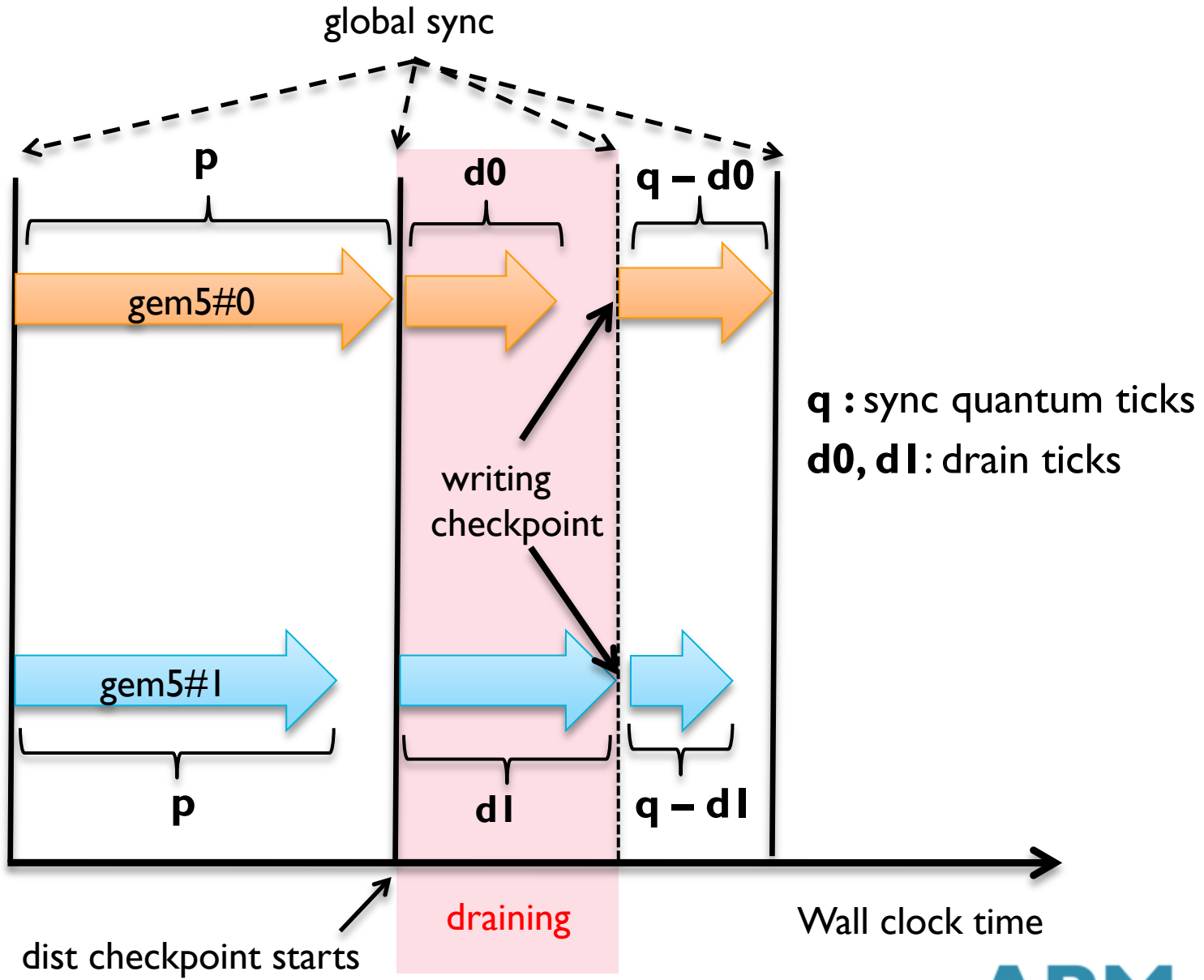
Checkpoint @ Global Sync

- In practical use cases a distributed checkpoint is taken "near" an application barrier (e.g. MPI_Barrier() or mpirun)
  - We want to take the checkpoint when all processes hit the barrier in the application code => desired application state can be captured even if we allow checkpoint writes only at global sync

- At a global sync
  - A compute node gem5 processes can signal its intention to take a checkpoint
    - 'm5 checkpoint' pseudo instruction => 'need checkpoint' meta info in the next 'sync request' message
  - Switch gem5 process can command to write a checkpoint
    - 'write checkpoint' meta info in the 'sync ack' message => exitSimLoop() in all gem5 processes

Writing Checkpoint

- Distributed checkpoint can start only at a global sync
- Draining may require different number of ticks in each gem5
- After drain complete, we flush out in-flight messages with an extra global sync
  - Global sync implements both an execution and a data (message) barrier

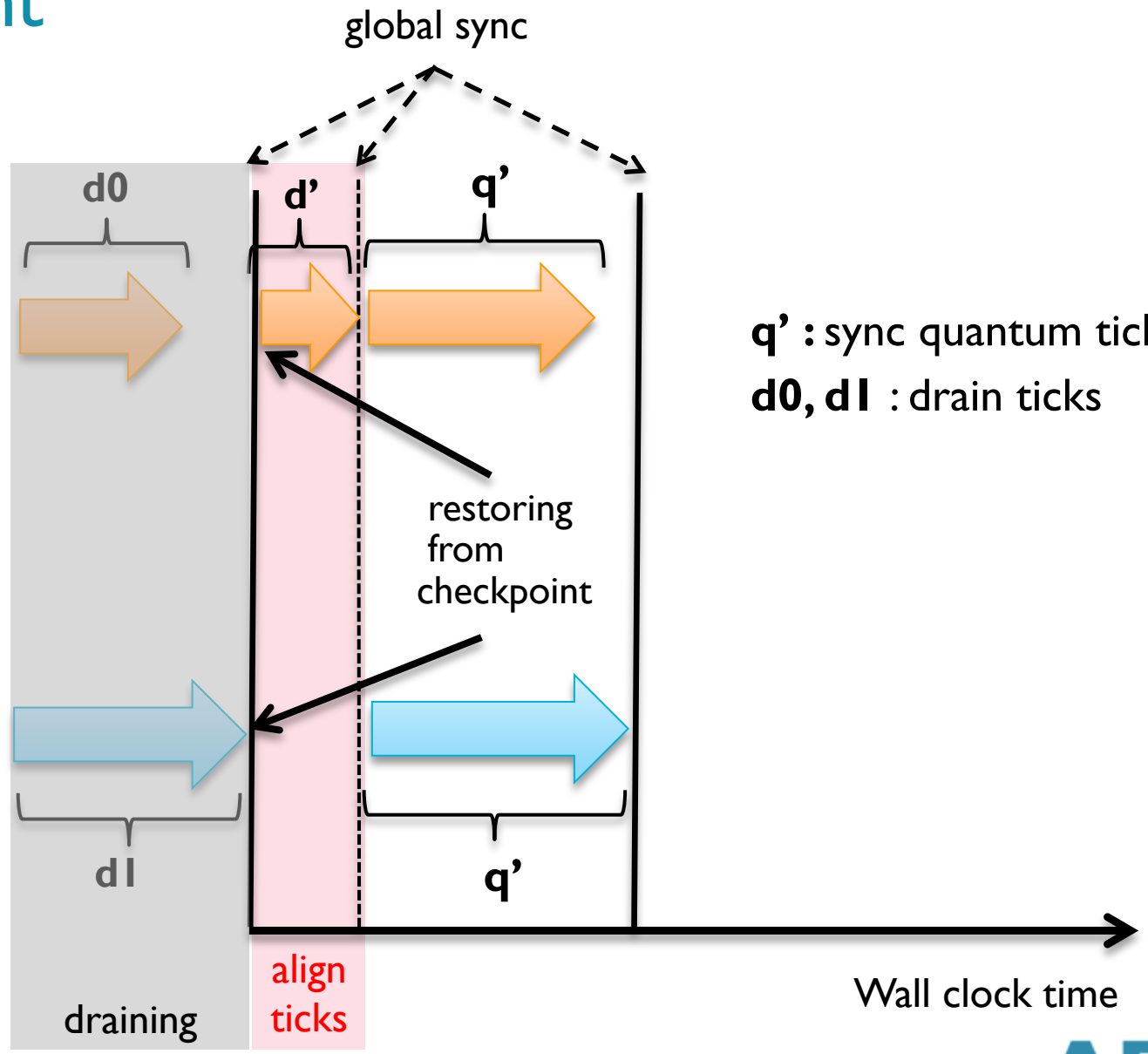


Restoring from Checkpoint

- Checkpoint might be written at different ticks in different gem5 processes
- An additional global sync to align the ticks:

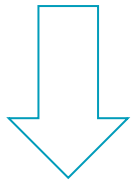
  d0 + d' = d1

  - Global sync delivers the max tick value to all gem5 processes
  - Periodic global sync always happens at the same tick in every gem5
- Global sync period may change at restore
  - Same checkpoint can be used to explore different network link latency/bandwidth effects later



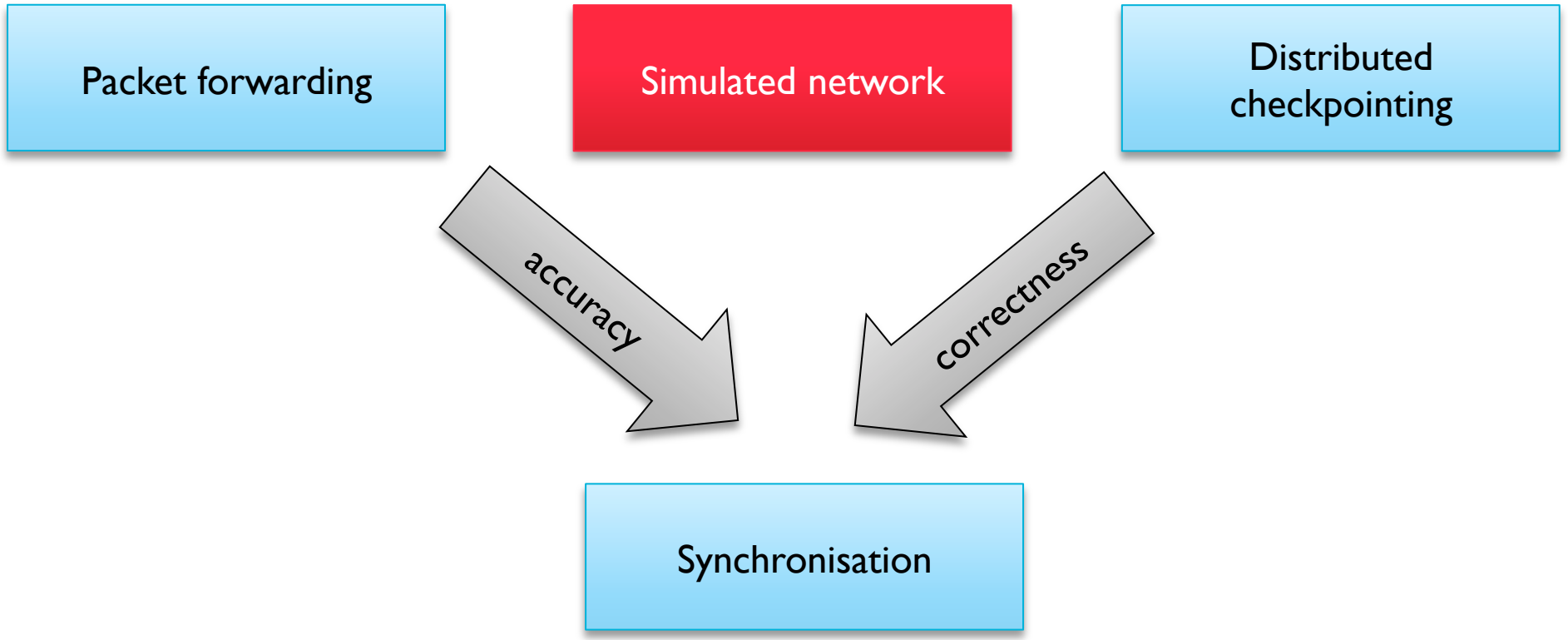
Restoring from Checkpoint (cont.)

- User is allowed to change simulated link parameters when restoring from a checkpoint
  - Same checkpoint can be used to explore different network link latency/bandwidth effects



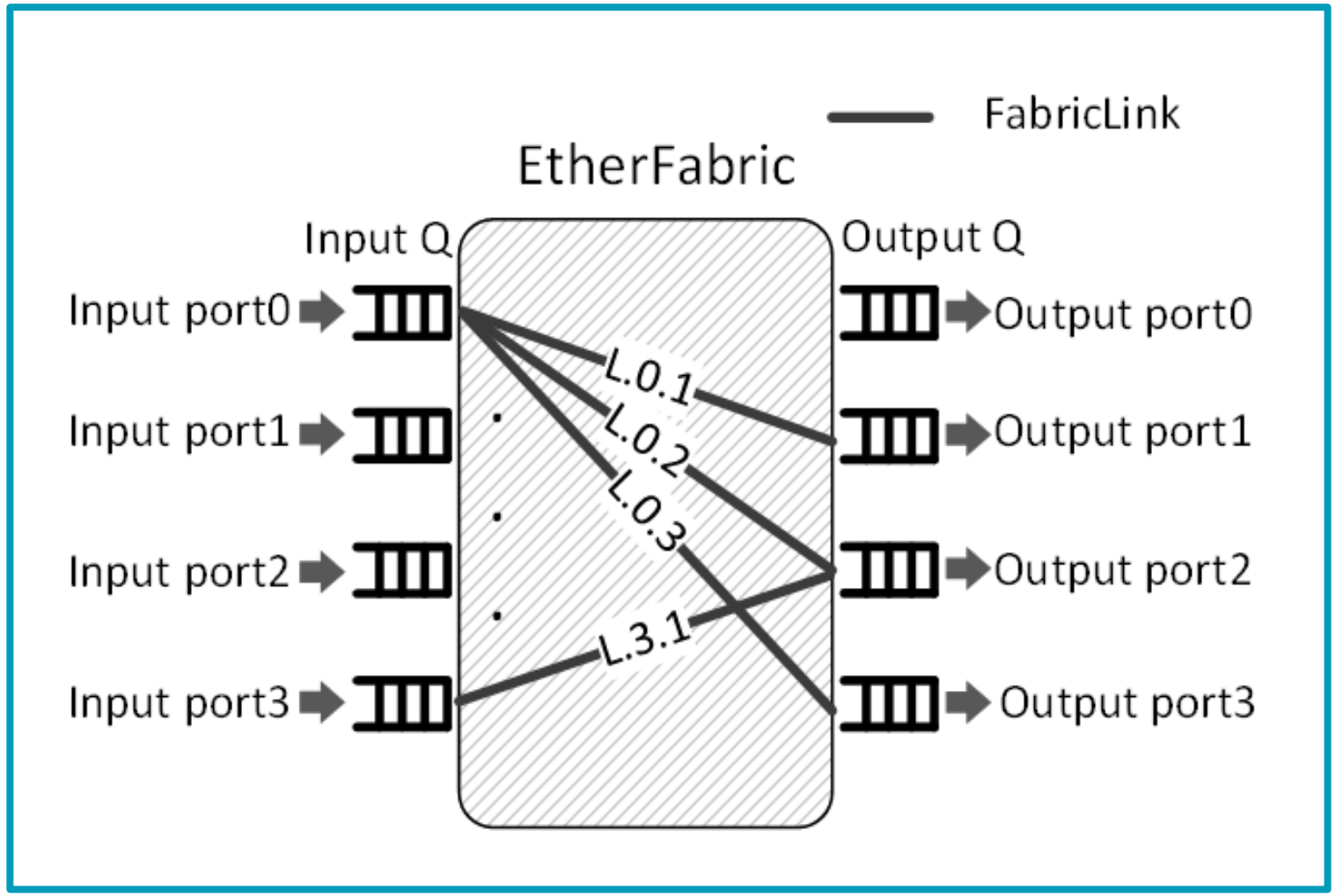
- Global sync period may change at restore (if the simulated link latency change)
  - Checkpoint may contain simulated packets to get received in the future
  - Receive ticks for such packets need to be adjusted to reflect the change of the simulated link parameters

Core Components



Architecture of the Simulated Ethernet Switch

- Interface
  - bi-directional port
  - input and output packet queues
  - it can connect to DistEtherLink (or EtherLink)
- EtherFabric
  - model a crossbar between all input and output ports
- ForwardEngine
  - move packets from input queues to output queues
  - schedule new attempt in the future in case of contention
  - map MAC addresses to ports



Contents

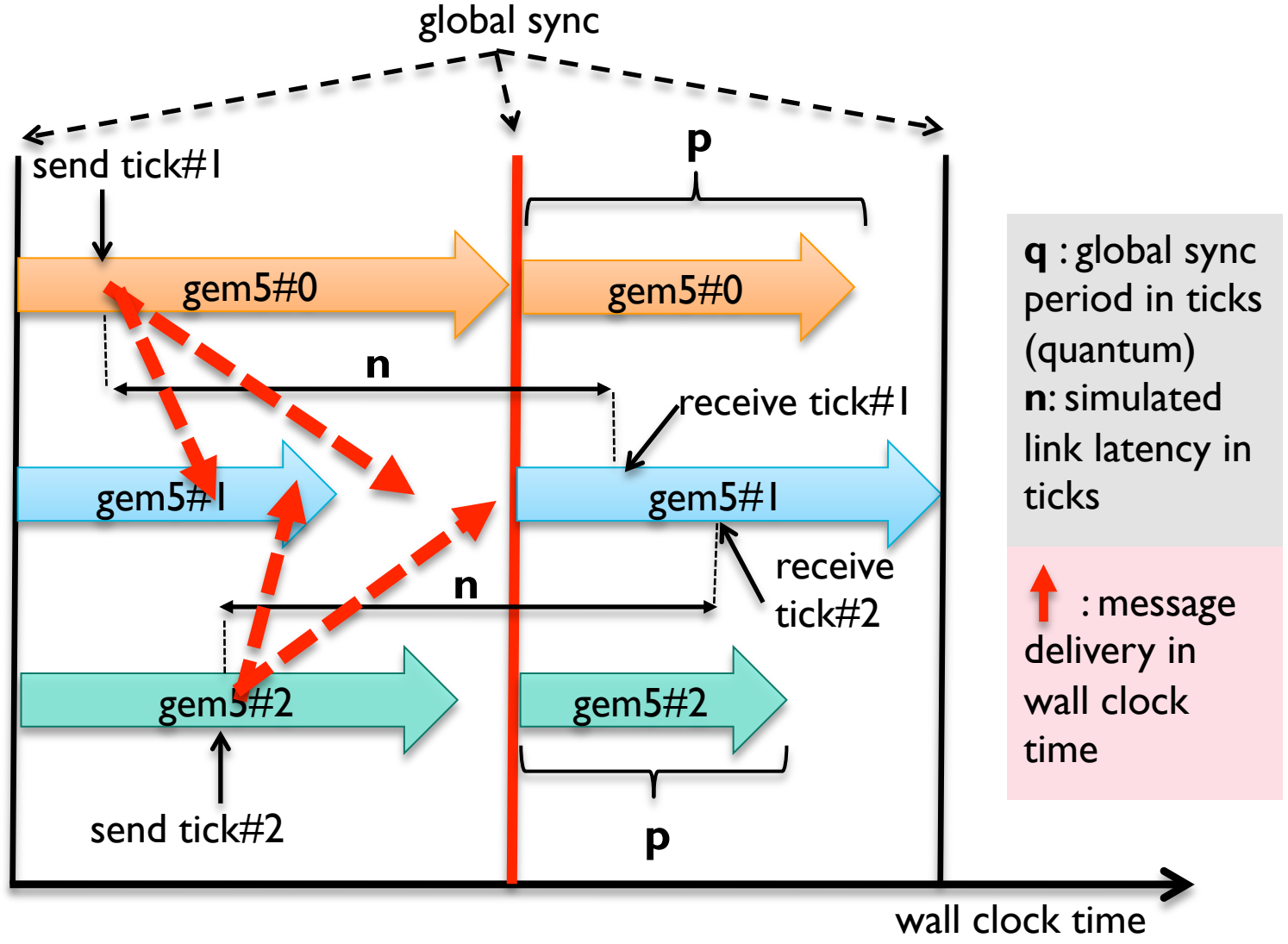
- Core components
  - Packet forwarding
  - Synchronisation
  - Checkpointing
  - simulated Ethernet switch
- **Deterministic execution**
- Class and Object diagrams
- Conclusions and Future Work

Deterministic Execution Issues

- We assume that a single compute node gem5 simulation is deterministic
- Ordering and speed of dist-gem5 messages in real world
  - Speed of gem5 processes (relative to each other) may vary
  - Communication speed among gem5 process may vary
- Global sync guarantees deterministic packet forwarding
  - **sync quantum <= simulated link latency**
  - **global sync is a message barrier**

Global Sync and Deterministic Packet Forwarding

- Receive tick for a simulated packet may not fall within the same quantum which the message gets received in
- A message is always gets sent and received within a single quantum



q : global sync period in ticks (quantum)
n: simulated link latency in ticks

: message delivery in wall clock time

Global Sync and Deterministic Packet Forwarding (cont.)

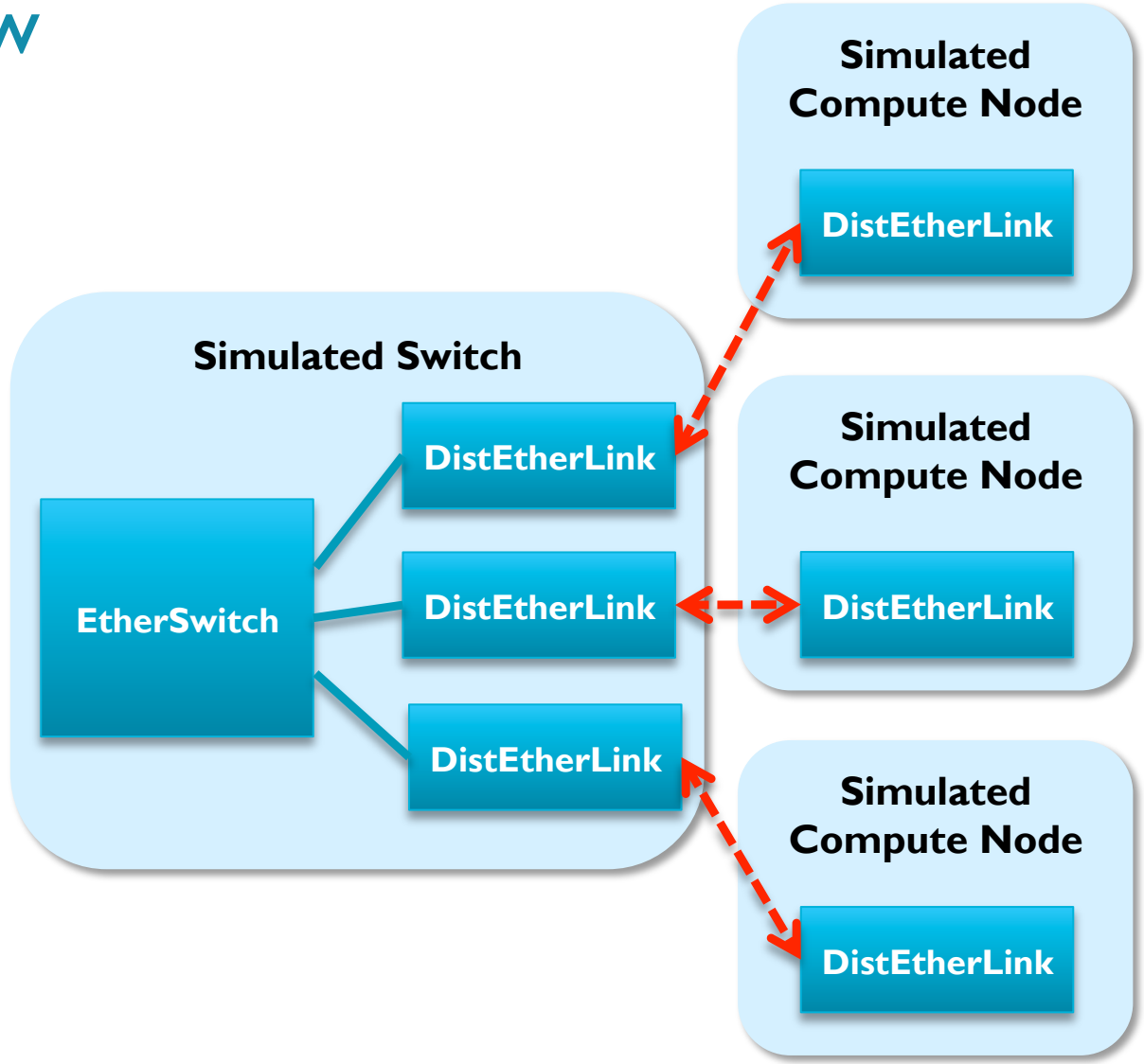
Pre-condiction	Invariant
quantum <= simulated link latency	**Receive order of messages within the same quantum does not matter**
	The ordered list of receive ticks falling within the active quantum will not change
global sync is a message barrier	**Each message will "happen" in exactly the same quantum across different runs**

Contents

- Core components
  - Packet forwarding
  - Synchronisation
  - Checkpointing
  - simulated Ethernet switch
- Deterministic execution
- **Class and Object diagrams**
- Conclusions and Future Work

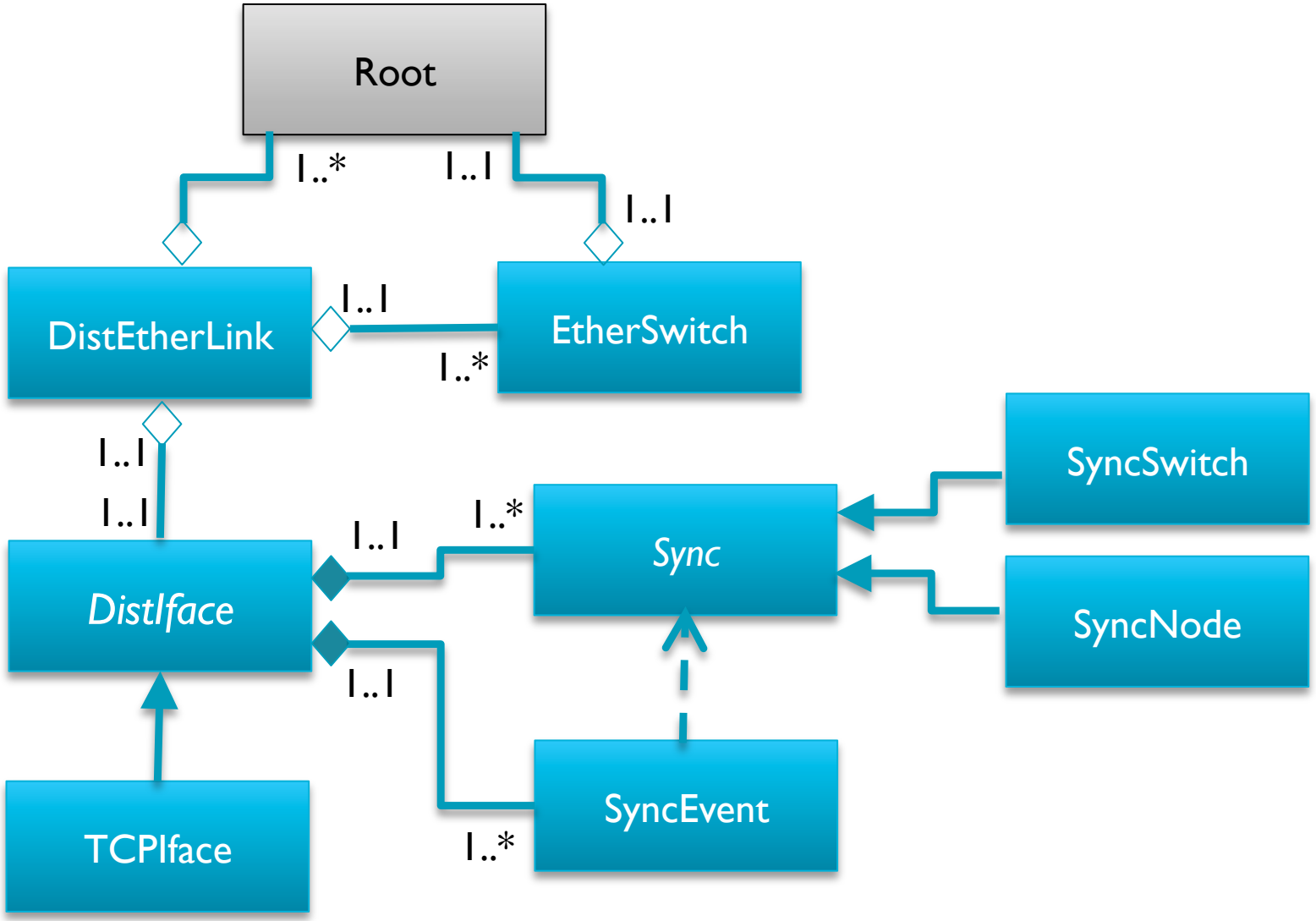
Implementation – High Level View

- New SimObjects to build dist-gem5 simulations
  - **DistEtherLink**
    - Simulated Ethernet link that connects systems simulated on different hosts
    - It can be used as a drop in replacement for the vanilla EtherLink object
  - **EtherSwitch**
    - Ethernet switch model
    - Crossbar topology
    - Forward packets from input to destination ports
    - Models port / fabric contention

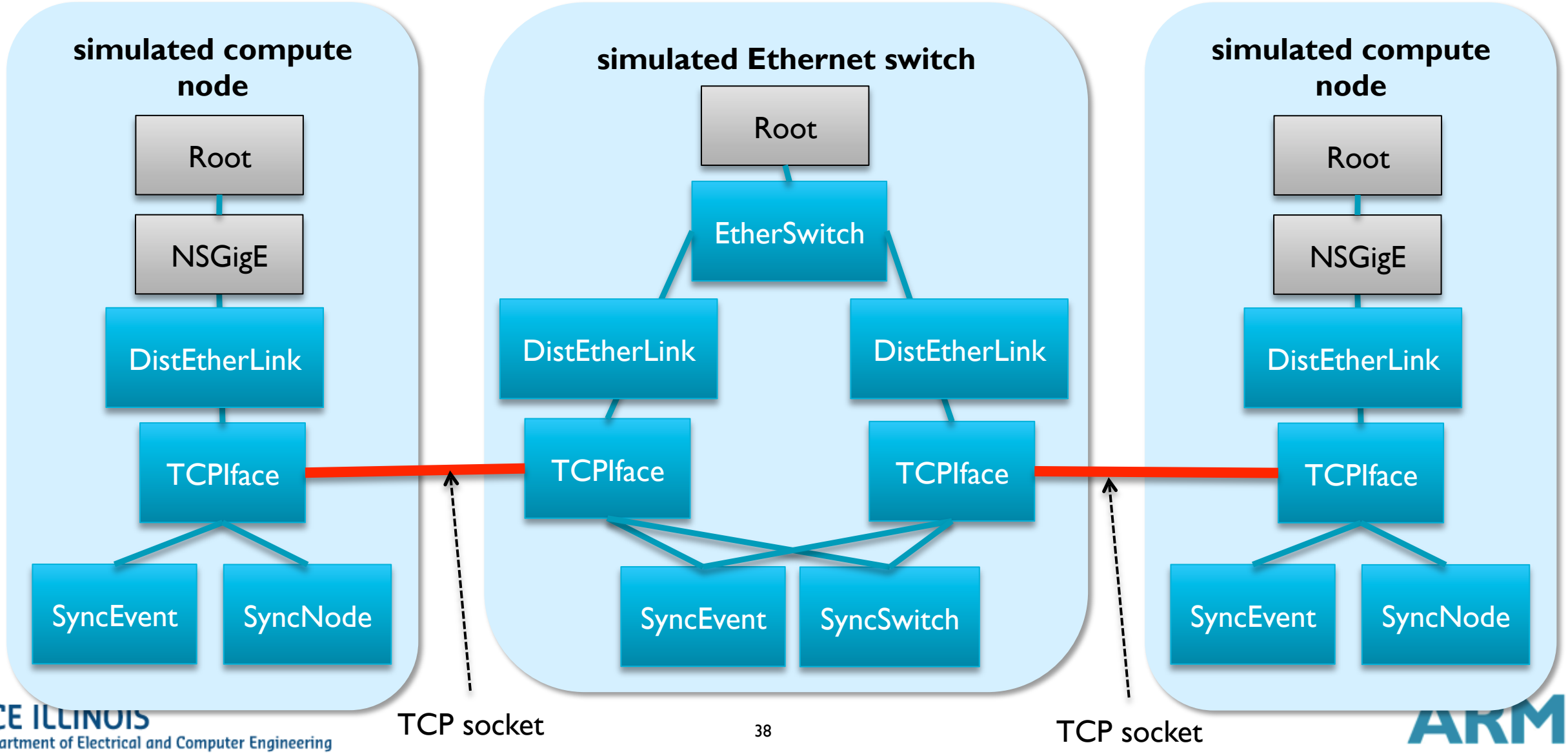


Class Diagram

- DistIface
  - packet forwarding
  - checkpoint co-ordinaton
  - synchronisation
  - pure virtual real world message transfer methods
- TCPIface()
  - TCP socket based implementation of a real world transport layer for DistIface() services



Object Diagram : Simulating a 2-node Cluster Example

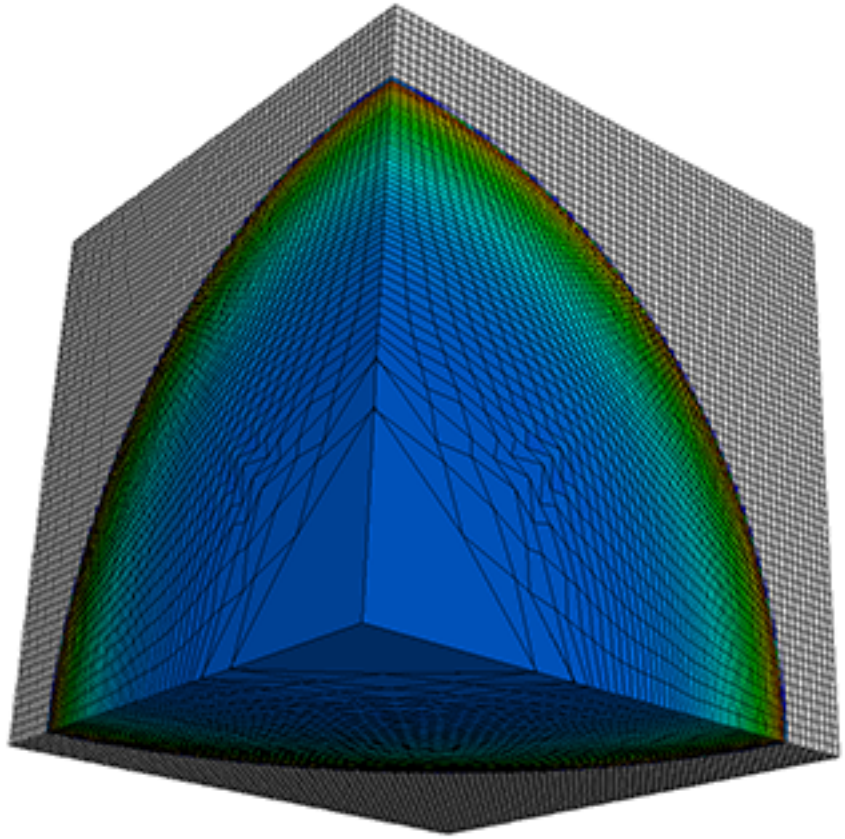


ARM Use Case

Case study : Network sensitivity of LULESH

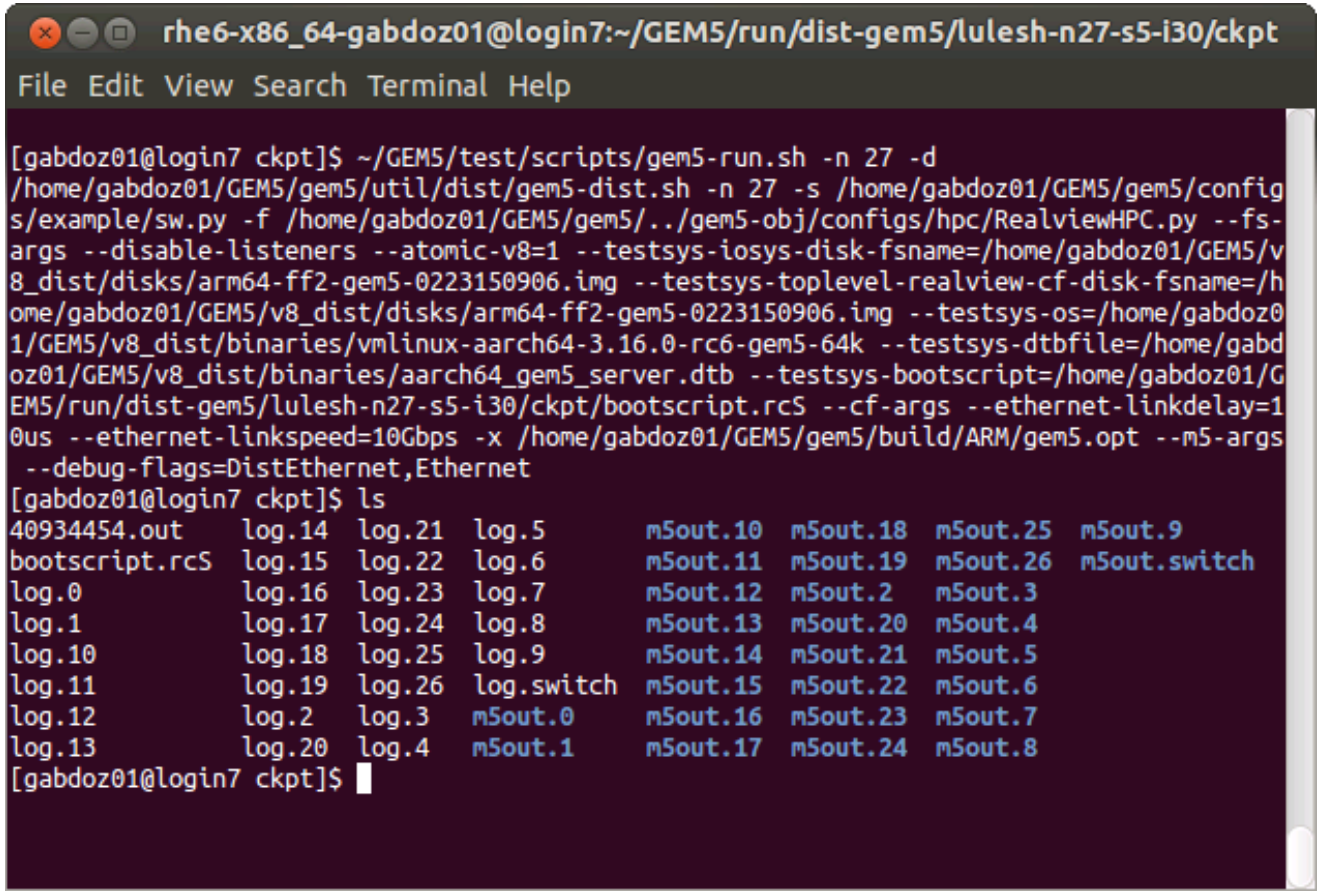
Lawrence Livermore National Laboratory

- What is LULESH?
  - Livermore Unstructured Lagrange Explicit Shock Hydrodynamics
  - A widely studied proxy application in DOE co-design efforts for exascale
  - Modeling hydrodynamics, which describes the motion of materials relative to each other when subject to forces
  - Highly simplified application that represents a typical hydrocode
  - Ported to a number of programming models (MPI, OpenMP, CUDA, Chapel, Charm++, etc.)



Running LULESH on distributed gem5

- Compute node config
  - ARMv8 single core CPU @ 1GHz, 2 GB DRAM
  - Ethernet NIC
- Switch config
  - 27-port Ethernet xbar
  - 1KiB input/output buffer per port
- LULESH command line
  - mpirun –n 27 lulesh-mpi –s 5 –i 30
    - -s : input data size per MPI process
    - -i : number of iterations in the main compute loop



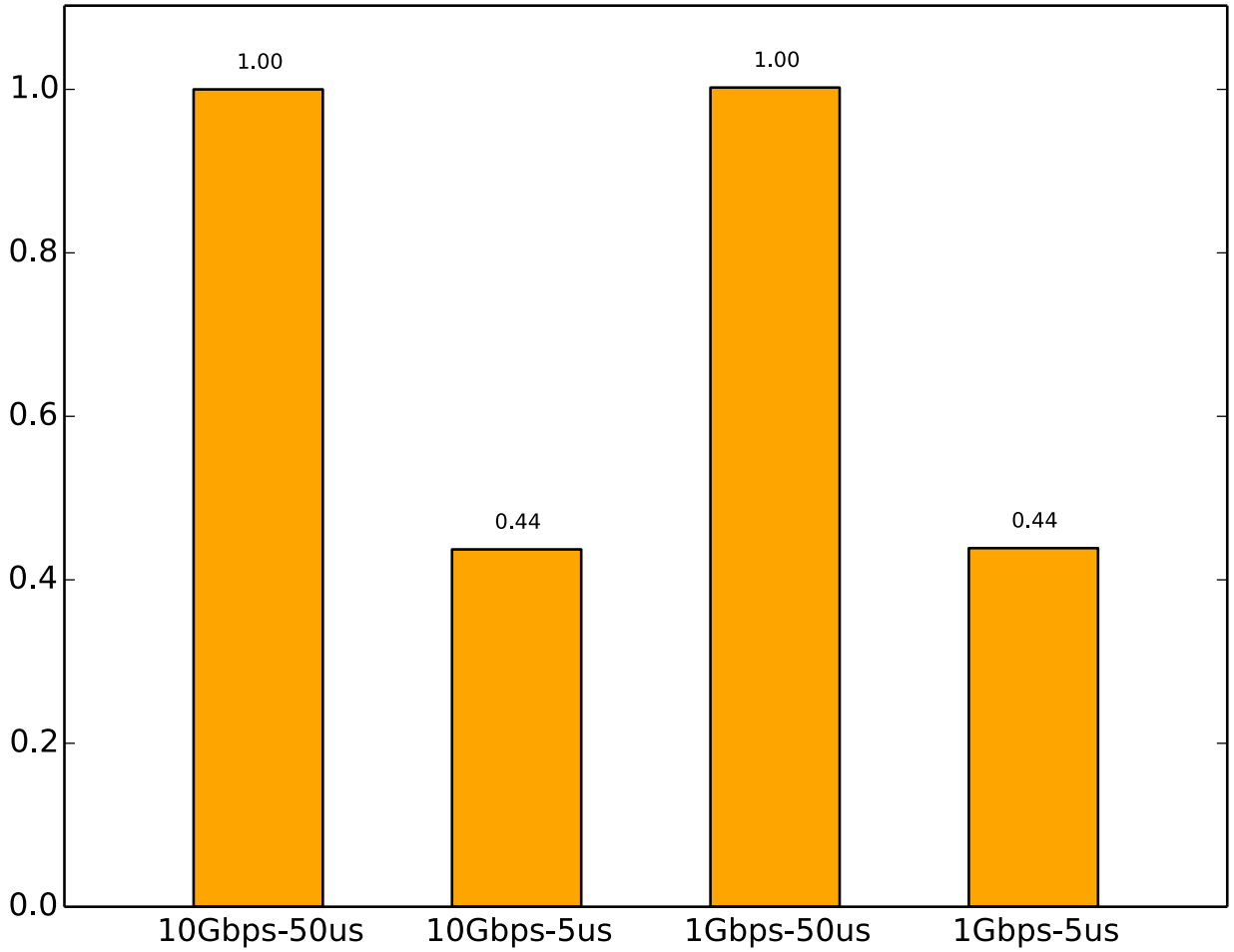
Running LULESH on distributed gem5 (cont.)

- Source code instrumentation to capture ROI
  - 'm5 checkpoint' pseudo instruction was inserted before main compute loop
  - 'm5 exit' pseudo instruction was inserted after the main compute loop
  - 'checkpoint' and 'exit' instructions can be collaborative : action is only taken when all participating gem5 processes complete the pseudo instruction
- Simulation runs
  1. Fast forwarding (atomic CPU) until the MPI_Barrier (before the ROI) was hit in all 27 processes
  2. Executing ROI in detailed (O3 CPU) mode by restoring from checkpoint
  - Change Ethernet link parameters at resume to explore latency/bandwith sensitivity

Ethernet link config	latency (us)	bandwidth (Gbps)
1.	50	10
2.	5	10
3.	50	1
4.	5	1

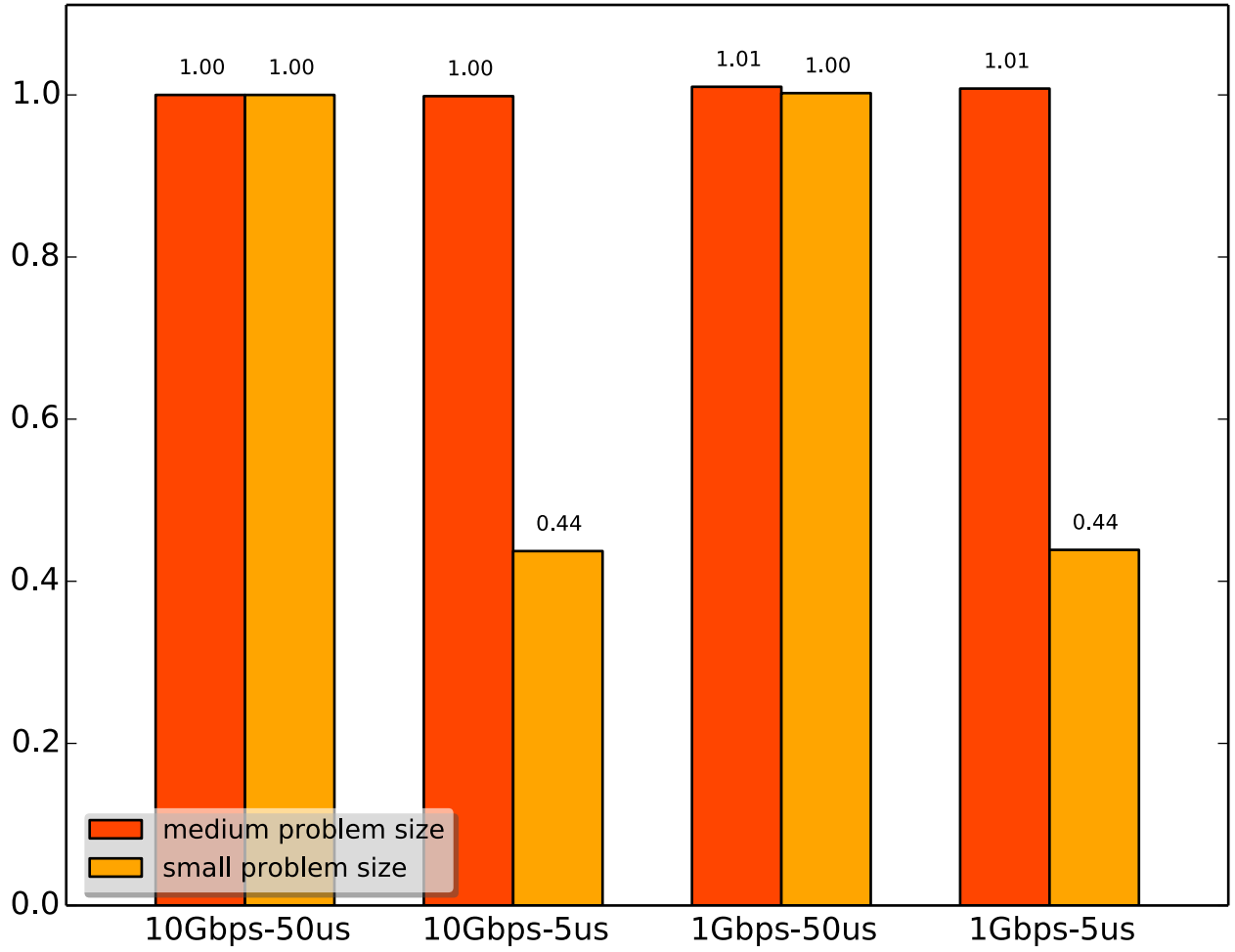
LULESH performance results – small input data size

- Performance is measured as run time of ROI
  - number of cycles from gem5 stats
  - max of the 27 compute nodes
- Results are normalized to the 1st config
  - 10Gbps bandwidth and 50us latency
- 5us link latency reduces run time by 55%



LULESH performance results – large vs. small input data size

- Results are normalized to the 1st config for both sets(10Gbps bandwidth and 50us latency)
- Sensitivity for link latency diminishes for large input data size
  - LULESH can overlap computation and communication



Conclusions

- Distributed gem5 enables scalable simulations of distributed systems
- Integrated part of the gem5 simulator

- Collaboration between ARM Research and University of Illinois (ex-Wisconsin)
  - Prof. Nam Sung Kim (nskim@illinois.edu)
  - Mohammad Alian (malian2@illinois.edu)
  - Gabor Dozsa (gabor.dozsa@arm.com)
  - Stephan Diestelhorst (stephan.diestelhorst@arm.com)

- Patches are on the gem5 review board
- Available to the gem5 community soon (in a few weeks …)

Future work

- Evaluate large scale simulation runs
  - Optimize real world communication transport for scale out simulations
    - MPI, OpenSHMEM,…
  - Optimize synchronisation to lower overhead
    - Relaxed synchronisation,…
- Model different networks
  - Hierarchical switches
    - Data center : blade switch, TOR switch …
    - synchronisation domains for different link latencies
  - HPC oriented network hardware
    - Infiniband (Mellanox) , BXI (Bull), …