A leading institution at the heart of the digital society

EURECOM
Sophia Antipolis

SIMUtools 2014

Newcom#

Large-scale Network Simulation over Heterogeneous Computing Architecture

Issues, Opportunities and Challenges

N. Nikaein and B. B. Romdhanne

Mobile Communication Department
Eurecom

Outline

Introduction & Background

Cunetsim

Hybrid scheduling

CMW & GP-CMW

NS-3 as proof of concept

Conclusion & Future work

About This Tutorial

- Explore **efficiency** and **scalability** horizons in network emulation and simulation field
  - **Execution time and runtime**
  - **Number of nodes, traffic load, and mobility**

- Applicability to popular simulation/emulation tools,
  - NS-3

Network Experiment

- **Simulation**
  - No interaction with the external entities (closed environment)
  - Part or all of the elements of a network/system is modeled or abstracted
- **Emulation**
  - Bring the external elements with their I/O streams (open environment)
  - Decision on which element is real or modeled depends on the use case and purpose of the experiments
  - At least one thing is modeled
- **Real testbed → field trial**
  - All the elements are real
  - Part of the testbed maybe controlled

Network Experiment Human Perspective

Human

Real

Simulated

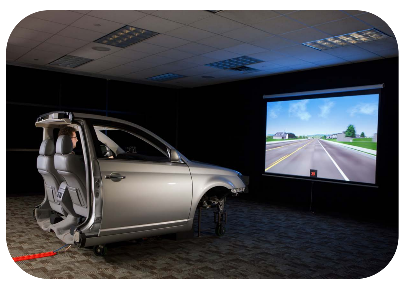
System

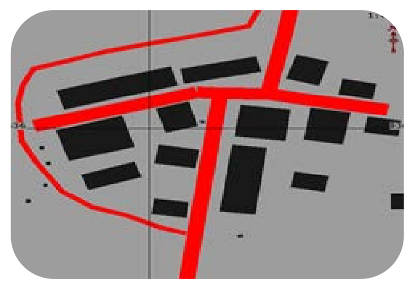
Real

Live

Cyber-Physical?

Simulated





Virtual

Constructive

Source: M. Loper

Network Experiment

Scalability — Reproducibility — Applicability

Analytical
- UML
- FreeMat
- IDL
- Matlab
- SciLab
- Octave

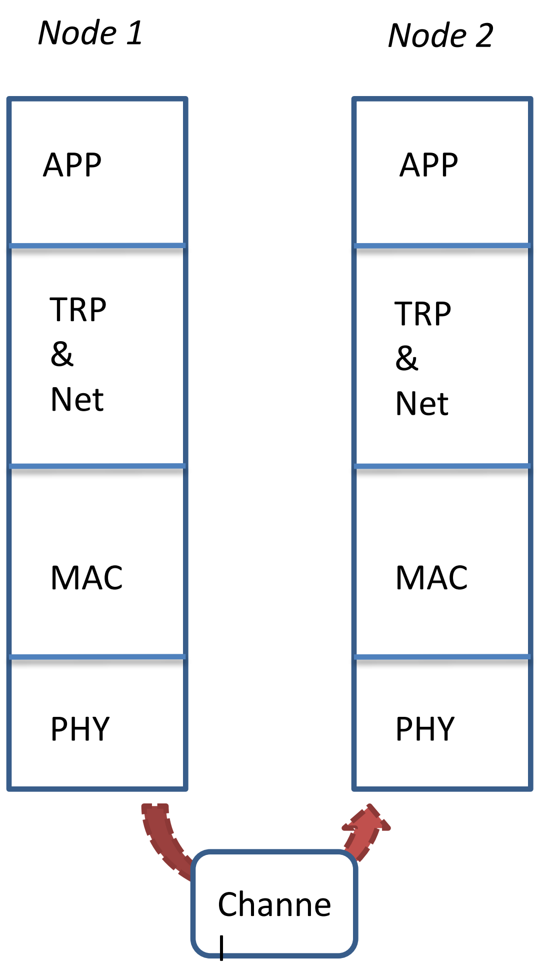
Simulation
- Sinalgo
- NetSim
- GloMoSim/Qualnet
- NS3
- Opnet
- Omnet++

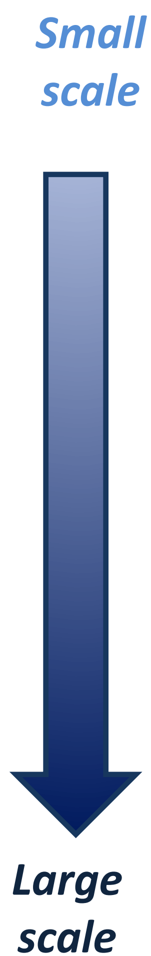
Emulation
- NS3
- NistNEt
- CORE
- USPR2
- WARP
- CMU-DSR
- ORBIT
- OAI

Real Testbed
- PlanetLab/OneLab
- NITOS
- GnuRadio
- WARP
- ORBIT
- Sundance, BEE2, WiTestLab
- USPR2
- OAI

Abstraction Level — Realism Level

Network Experiment





Evaluation methodology	Channel	PHY	MAC	TRP & NET	App
Real testbed	✓	✓	✓	✓	✓
Emulation Medium case1	✓	✓	✓	✗	✗
Emulation Medium case2	✗	✓	✓	✗	✗
Emulation App & protocol	✗	✗	✗	✓	✓
Simulation	✗	✗	✗	✗	✗

Network Experiment

- **Discrete event simulation model**
  - **Entity**, e.g. node, packet, channel, proto, models
  - **Link**, e.g. relationships among entities
  - **Event-driven discrete System**, e.g. Event occurs at discrete point of time changing the state of the system
- **Components**
  - State, clock, event list, counters, configure, time and event routines
- **Primitives**
  - run, stop, now, schedule, cancel, remove, release

Network Experiment
Discrete Event Simulation

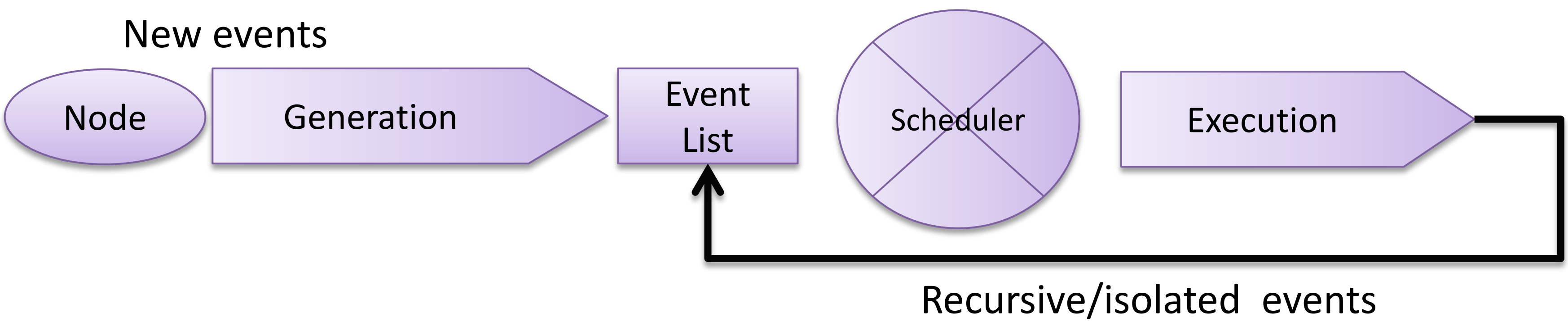
```
Void main () {
        mod model;
        ev event;
        initialize(&mod,&ev);
        configure(&mod,&ev);
        schedule(time, &my_func, &mod);
        …
        schedule_end_simu (ev);
        run();
        release ();
}
```

```
static void my_function (MyModel *model) {
//
}
```

```
void Run() {
   while (! end_of_simulation()) {
        time=get_timestamp();
        ev = extract_event (global_event_list);
        execute(ev);
}
```

Discrete Event Simulation

- A composition of a group of elementary entities
  - Finite state machine per component
  - Event triggers state changes
  - System state evolves over discrete and atomic time



- Sequential execution limits the scalability and efficiency

Parallel and Distributed Simulation

- Simulations executing over multiple computing systems
  - Tightly and/or loosely coupled multiprocessor systems

Parallel simulation involves the execution of a *single* simulation program on a collection of *tightly* coupled processors (e.g., a shared memory multiprocessor).

Distributed simulation involves the execution of a *single or multiple* simulation program on a collection of *loosely* coupled processors (e.g., PCs interconnected by a LAN or WAN).

Parallel and Distributed Simulation

- **Communication Mechanisms**:
  - Message Passing
  - Unicast, multicast, broadcast; publish/subscribe
  - Shared Memory
  - Remote Procedure Call (RPC)
  - Remote Method Invocation (RMI)
- **Event Synchronization**:
  - Clocks and Time
  - Event ordering

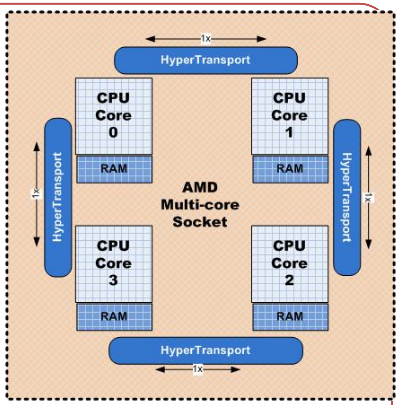
Source: M. Loper

Parallel and Distributed DES

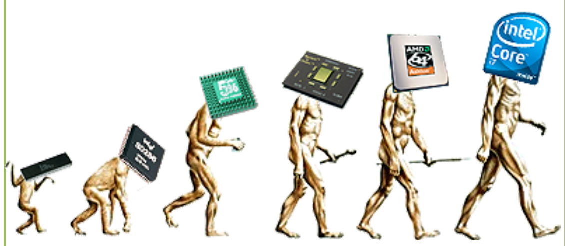
Parallel simulation	Distributed simulation
• Typically Shared memory context	• Typically several independent simulation instance
• Several execution resources	• Different machines
• Centralized scheduling	• Independent scheduling
• Memory-based communication mechanism	• Message-based communication
• Local synchronization	• Distributed synchronization

- Care must be taken for simulation correctness, synchronization overhead as well stability issues

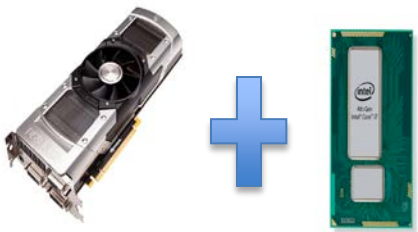
Hardware Context



Parallelism



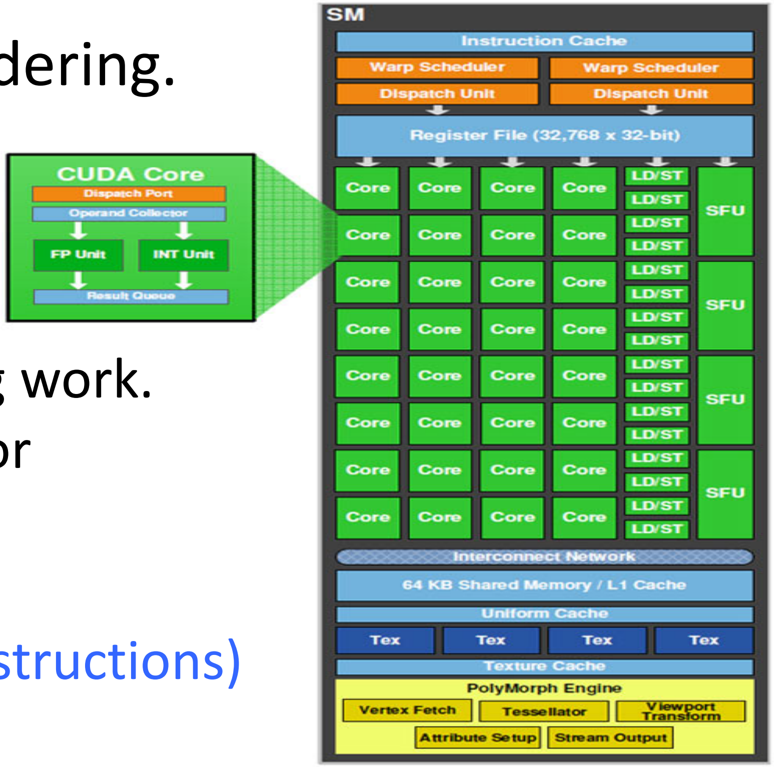
Efficiency and auto-scaling



Heterogeneity

Hardware Context
GPU Features and Specification

- Generally used for the graphical rendering.



Streaming Multiprocessor (SM)

- Current Trend
  - Able to ensures additional computing work.
  - Evolves on the sense of a co-processor
  - Large number of computing cores
  - Rapid dedicated memory
  - Hardware schedulers (threads and instructions)

- GPU cores are grouped into several streaming multi processor SM (like SIMD processors)

Software Context for HPC

IEEE POSIX
OpenMP
OpenACC
DIRECTIVES FOR ACCELERATORS
NVIDIA CUDA
OpenCL

Parallelism

GridMPI™
MPICH-V
MPI Implementation for Volatile resources
MVAPICH
openMPI

Distributed

Content of This Tutorial

- **Cunetsim:** GPU-based simulation framework
- **Hybrid-scheduler:** Conservative event scheduler targeting multi-target execution, both GPU & CPU.
- **CMW / GP-CMW:** optimized distributed and parallel simulation model targeting very large scale scenarios
- **NS-3:** proof-of-concept

Outline

Introduction & Background

Cunetsim

Hybrid scheduling

CMW & GP-CMW

NS-3 as proof of concept

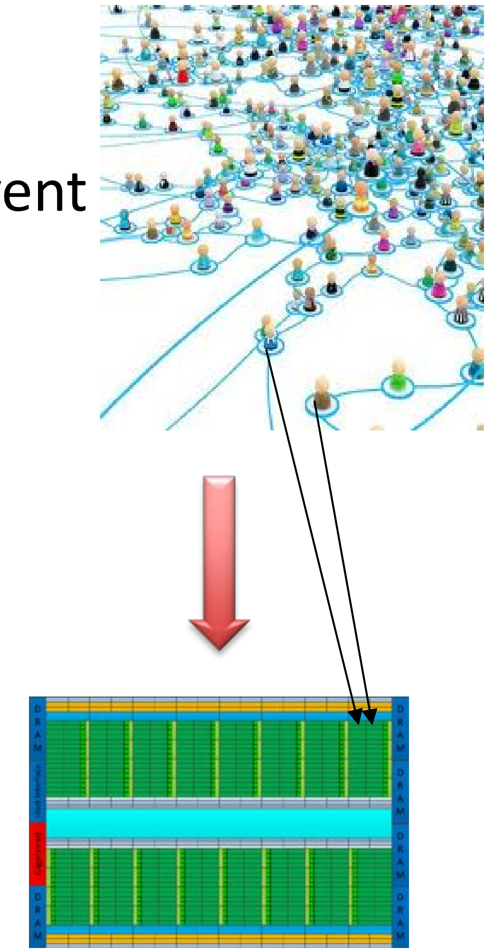
Conclusion & Future work

Cunetsim

General Idea

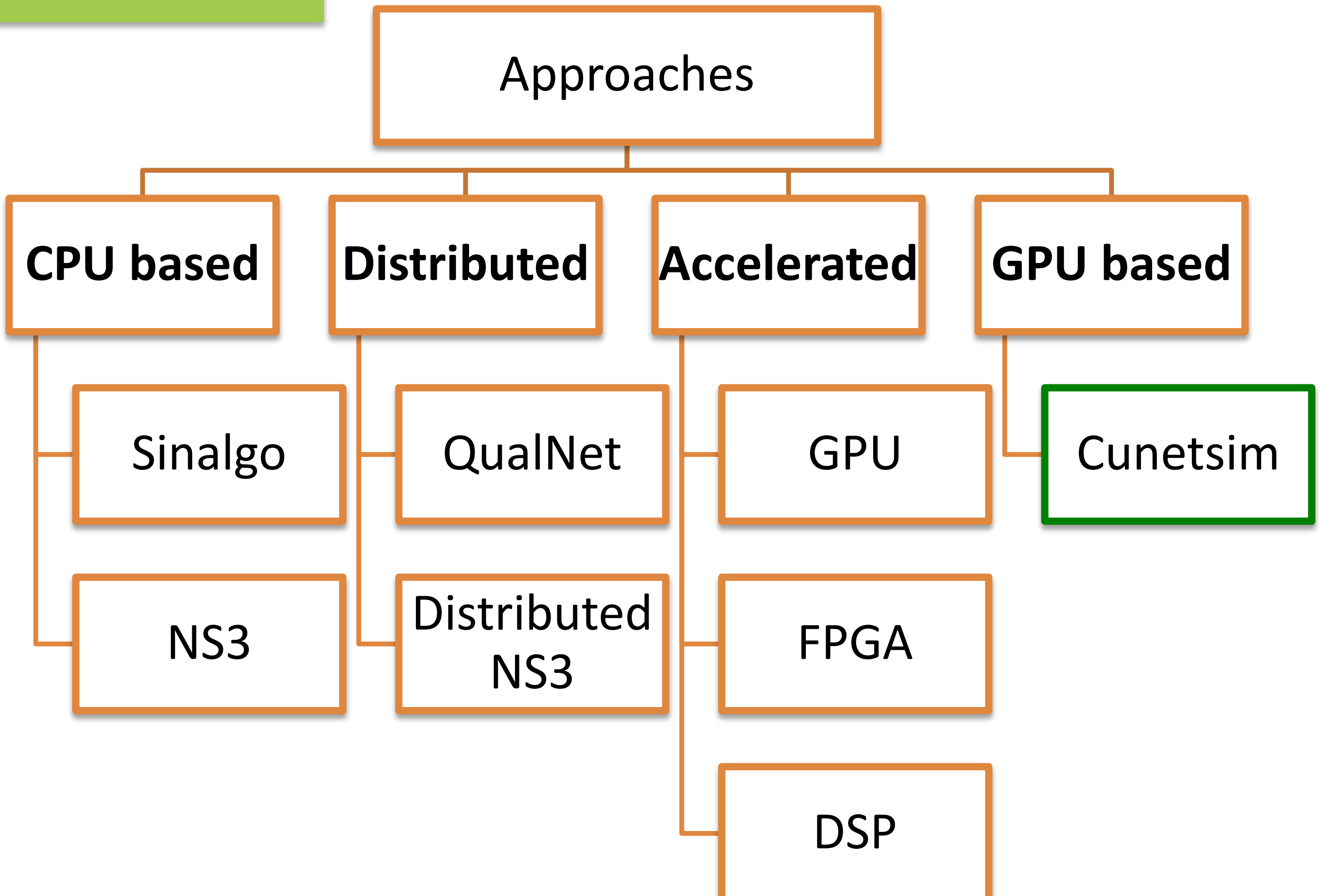
Fully GPU based simulator

- **One dedicated GPU core per node.**
  - At a given time Ti each node executes one event



- Only GPU executes, CPU controls
- Master-Worker simulation model
  – CPU is the master
  – GPU is the worker

Classification



Cunetsim

Event Descriptor

- Representation of an event used for management

Descriptor	Timestamps	callback	Arguments

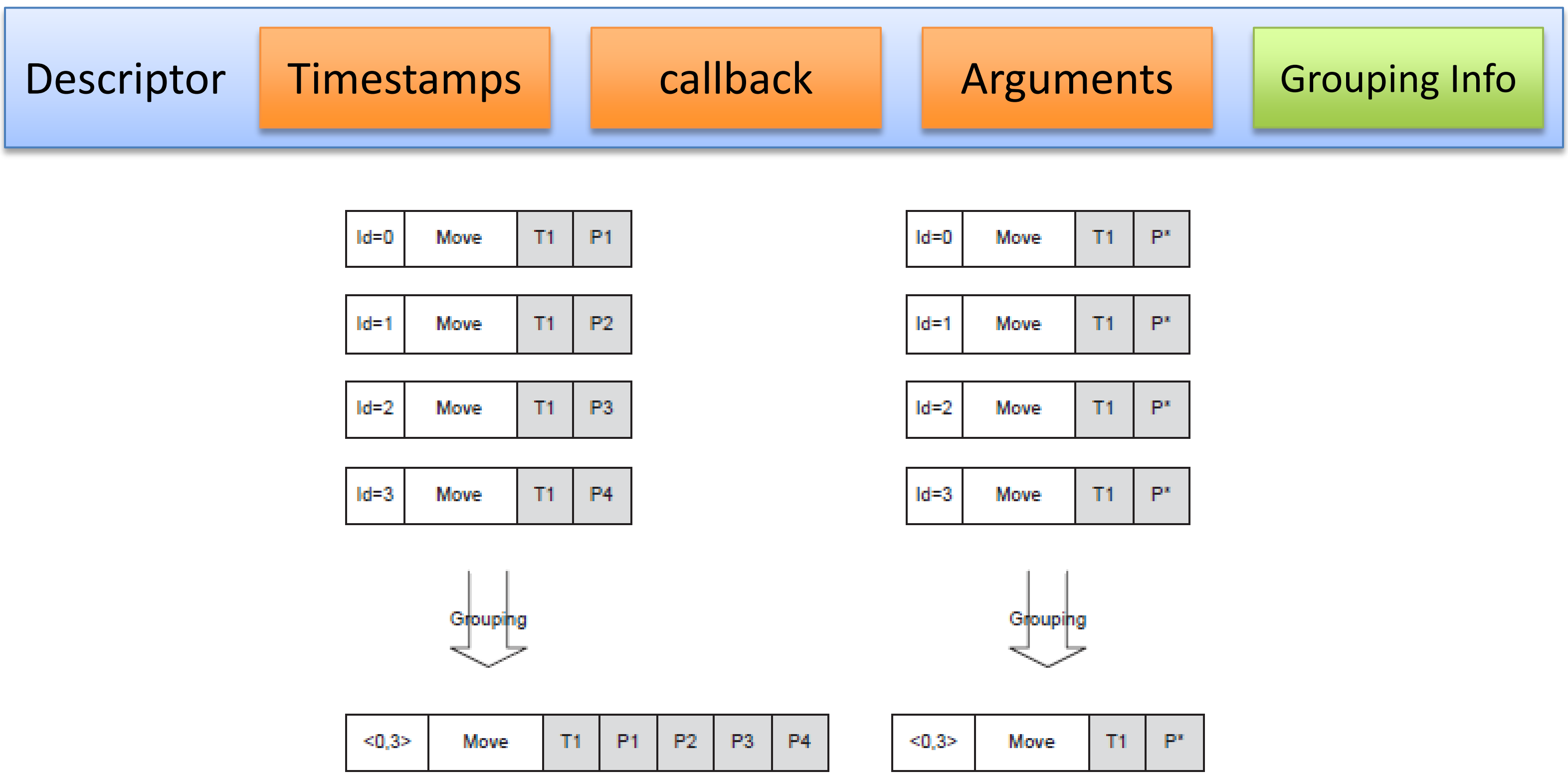
- Extend the event descriptor to support parallelism
  - **Grouping info for the events that only differs in their data**
  - **Cloned Independent Events** (CIE) represented as a single entry

Descriptor	Timestamps	callback	Arguments	Grouping Info

- Break the 1:1 relationship between an event and its descriptor

Event Descriptor

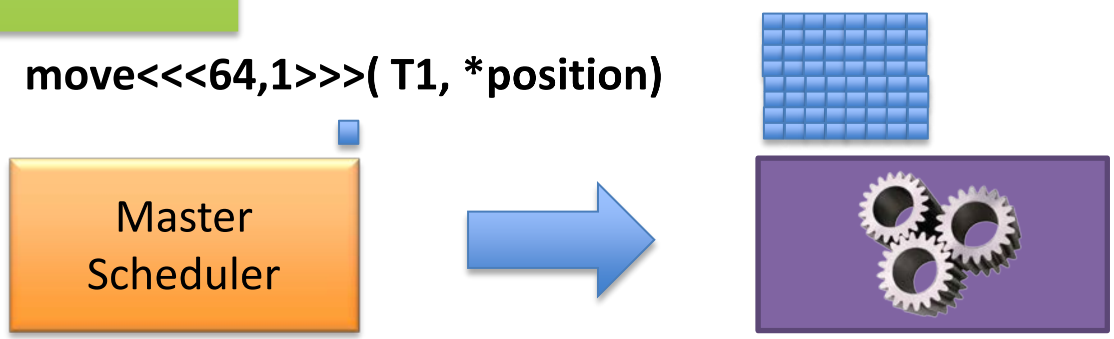
- **Grouping info for CIE events**

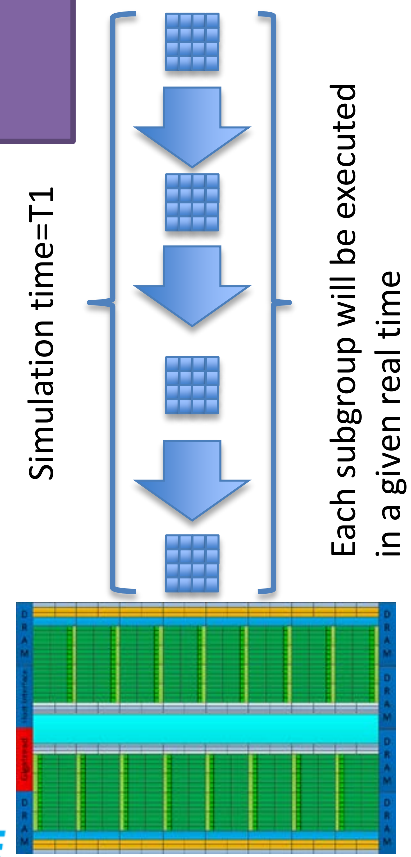


Cunetsim

Event Grouping

move<<<64,1>>>(T1, *position)



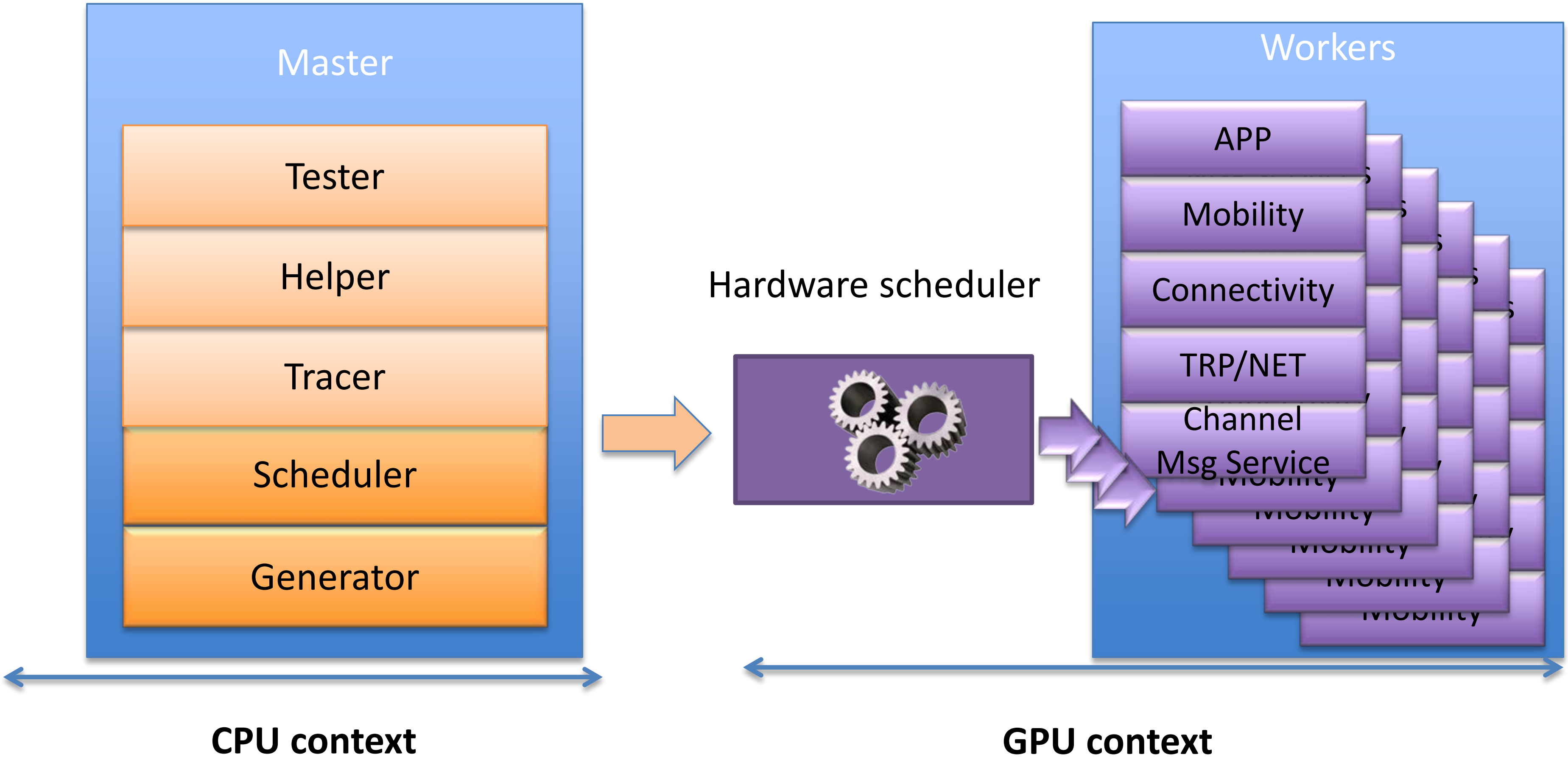


- Events are
  - Compressed during the generation (grouping info added)
  - Expanded during the execution by hardware scheduler

- **No strict order during execution inside a group**
  - Decision is made by the hardware scheduler
  - Events must be designed such that, the execution order of parallel events does not affect the correctness.

Cunetsim

Framework Architecture

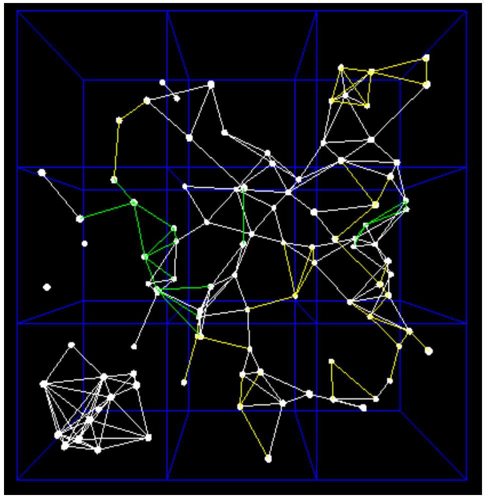


Cunetsim

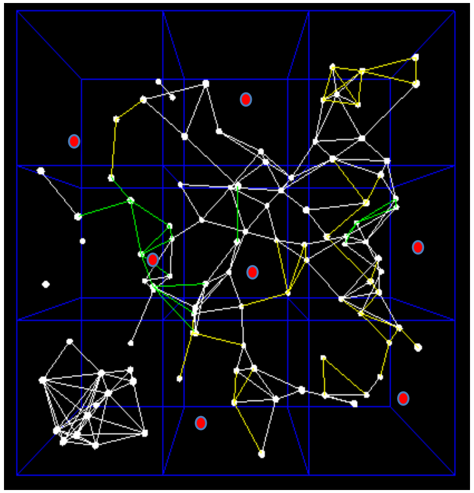
Evaluation Scenario

- Benchmarking Scenarios
  - 4-64k Nodes
  - 1600x1600x1600(3D)
  - 600 seconds
  - RWP mobility
  - UDG Conectivity
  - Flooding Proto
- Comparative Evaluation
  - NS-3 (distributed version- 6 LPs)
  - Sinalgo (asynchronous 6 threads)
  - Cunetsim-CPU (OMP – 6 threads) (via openACC)
  - Cunetsim-GPU (1 master+ 1 GPU)
- Software context
  - CUDA for GPU dev (GTX 460) and PGI for compilation

Scenario A

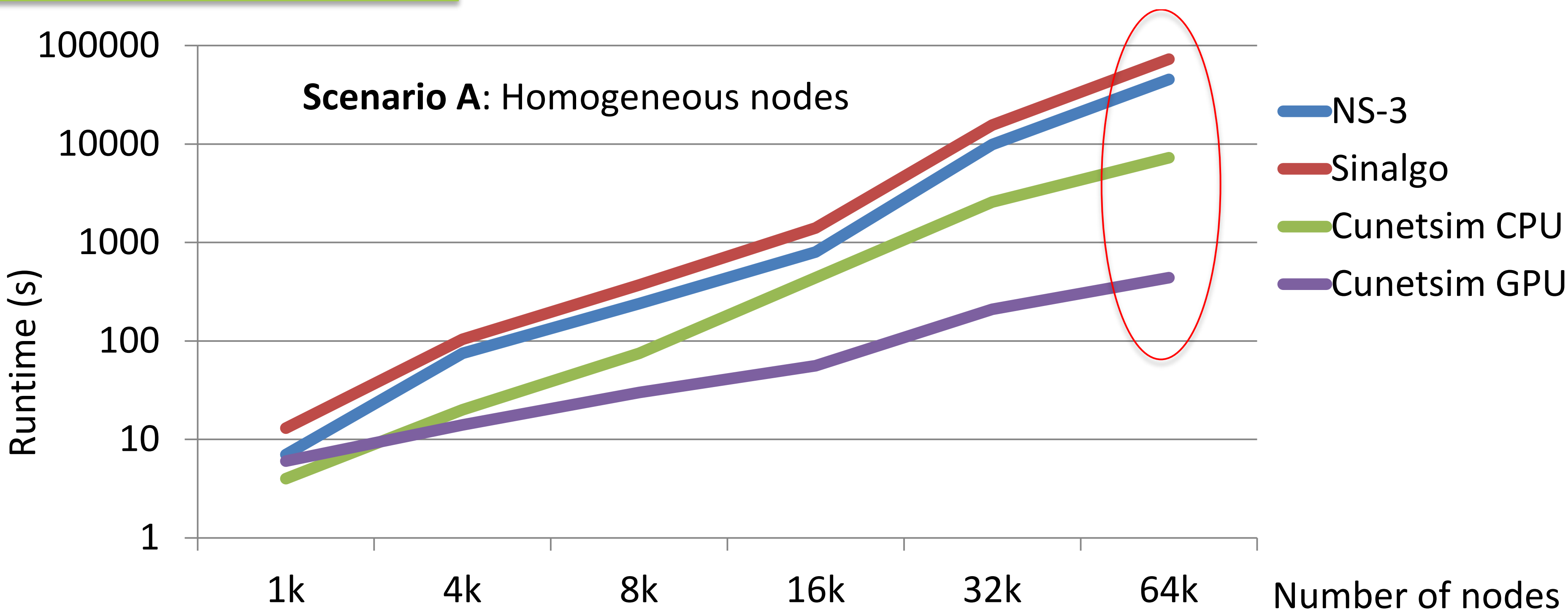


Scenario B



Cunetsim

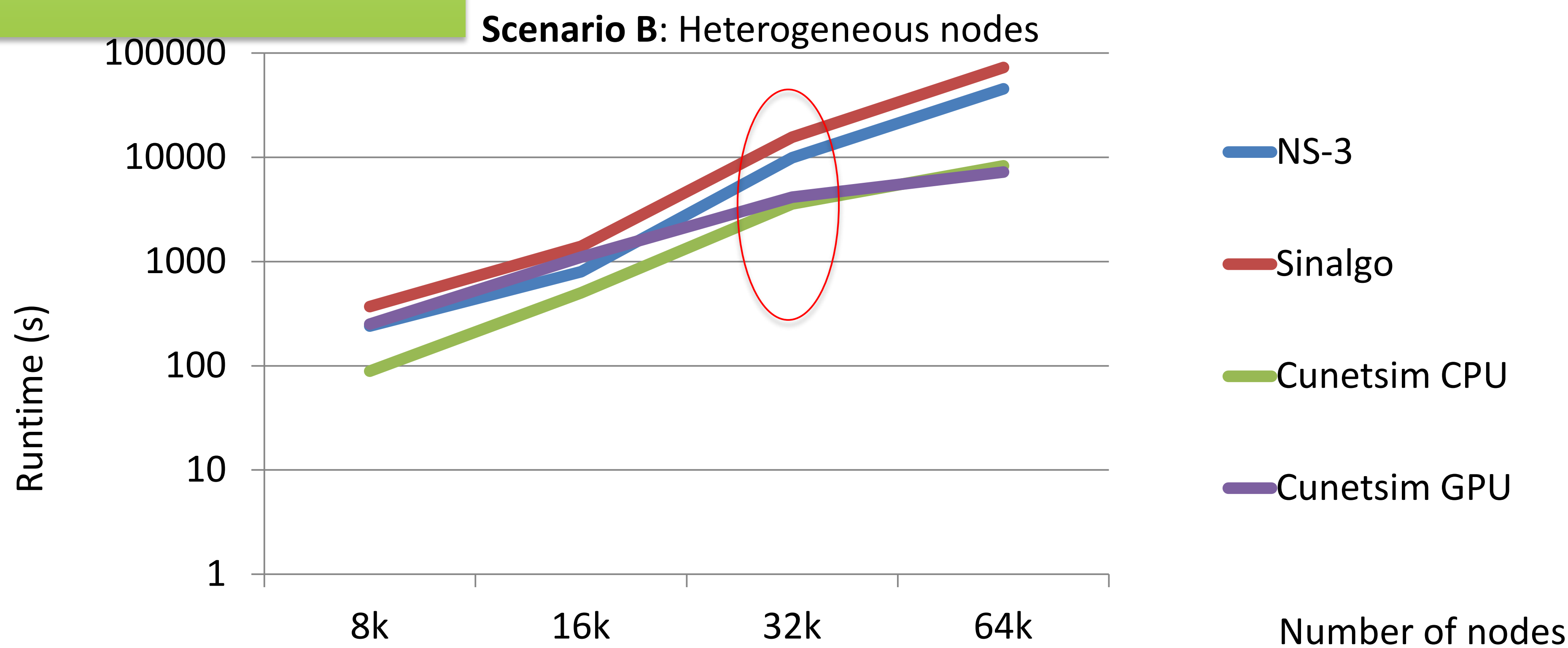
Performance Results



- **Gain obtained by grouped events**
  - 6x on the CPU target
  - 100x on the GPU target

Cunetsim

Performance Results



- CPU-based grouping remains stable
- GPU-based grouping runtime increased by a factor of 16
  - Higher number of isolated events
  - Cost of context switching and memory transfer

Outline

Introduction & Background

Cunetsim

Hybrid scheduling

CMW & GP-CMW

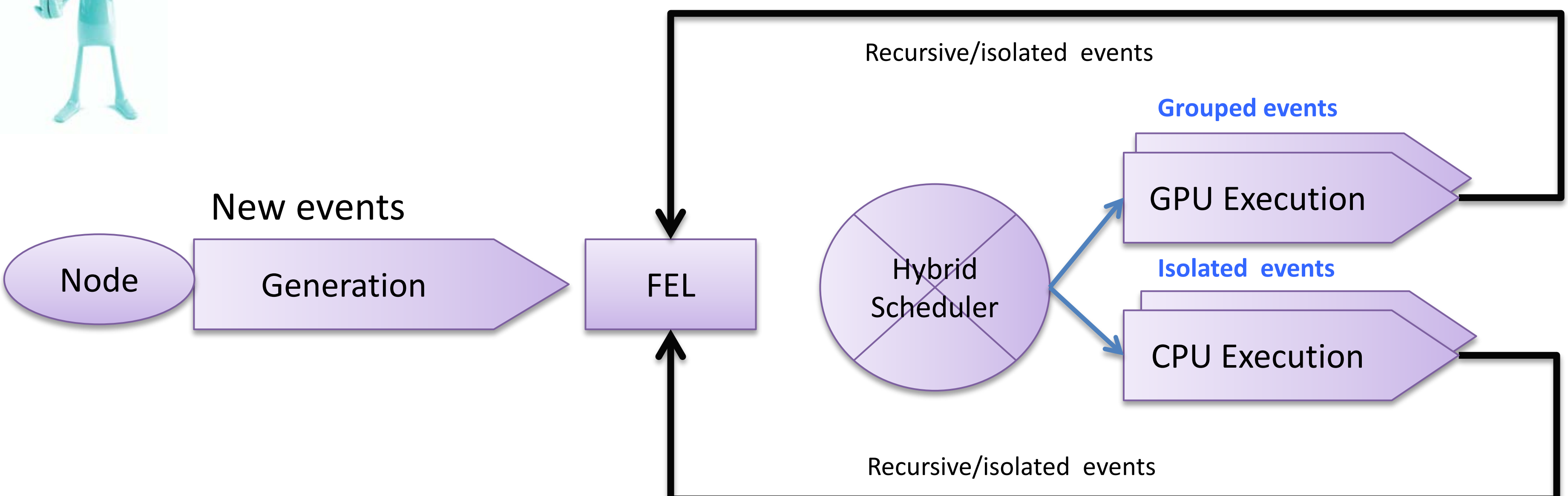
NS-3 as proof of concept

Conclusion & Future work

Hybrid scheduling

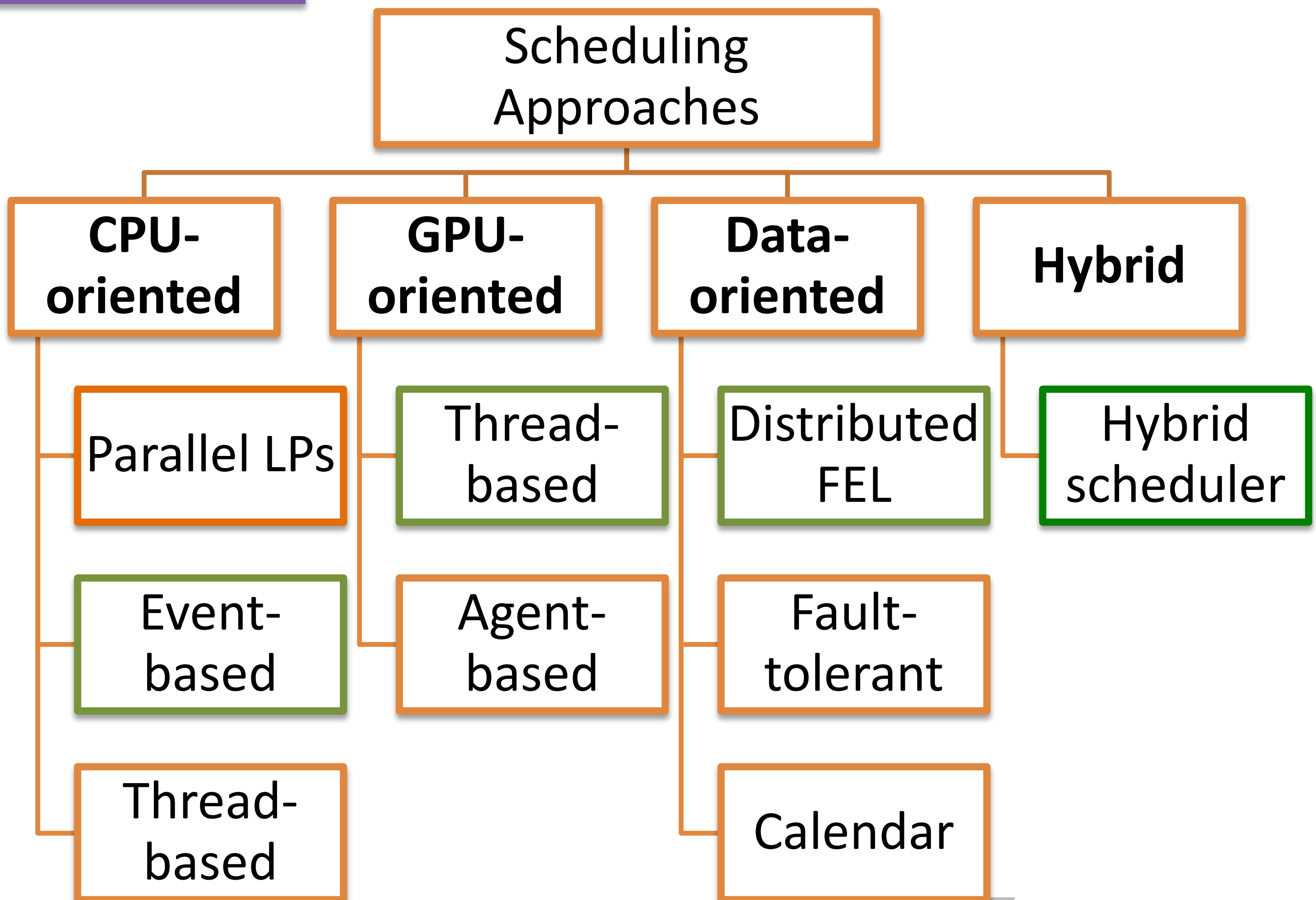
General Idea

- Maximize the hardware usage rate for both CPU and GPU targets
- Hybrid scheduling
  - Execute grouped events on the GPU
  - Execute isolated events on the CPU



Hybrid scheduling

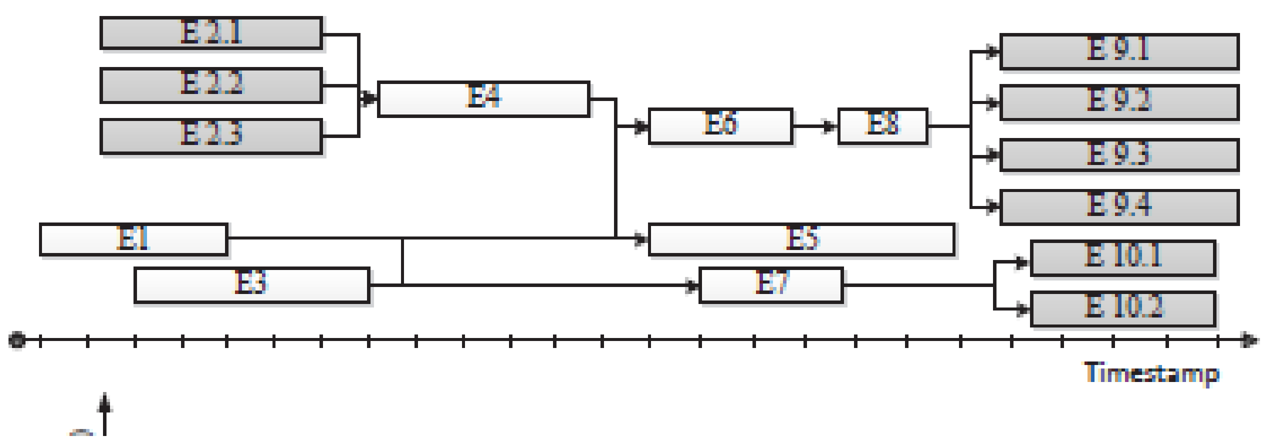
Related Works



Hybrid scheduling

Parallelize Events

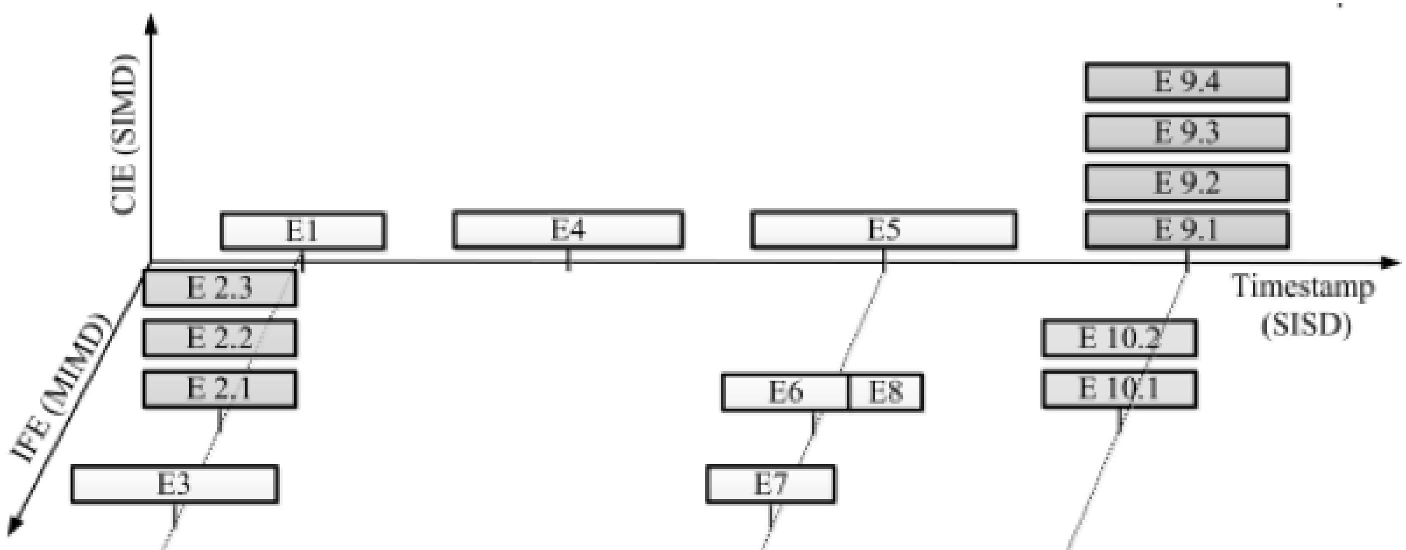
- 3-dimensional array list (3D-AL) data structure
  1. Timestamps : having sequential and strict order
  2. Foreign independent events: having the same timestamps
  3. Cloned independent events: having the same time stamp and instruction



Hybrid scheduling

Parallelize Events

- 3-dimensional array list (3D-AL) data structure
  1. Timestamps : having sequential and strict order
  2. Foreign independent events: having the same timestamps
  3. Cloned independent events: having the same time stamp and instruction



Events Flow and Stability

- Approach of dynamic system where events are flowing between producers and consumers sharing buffers

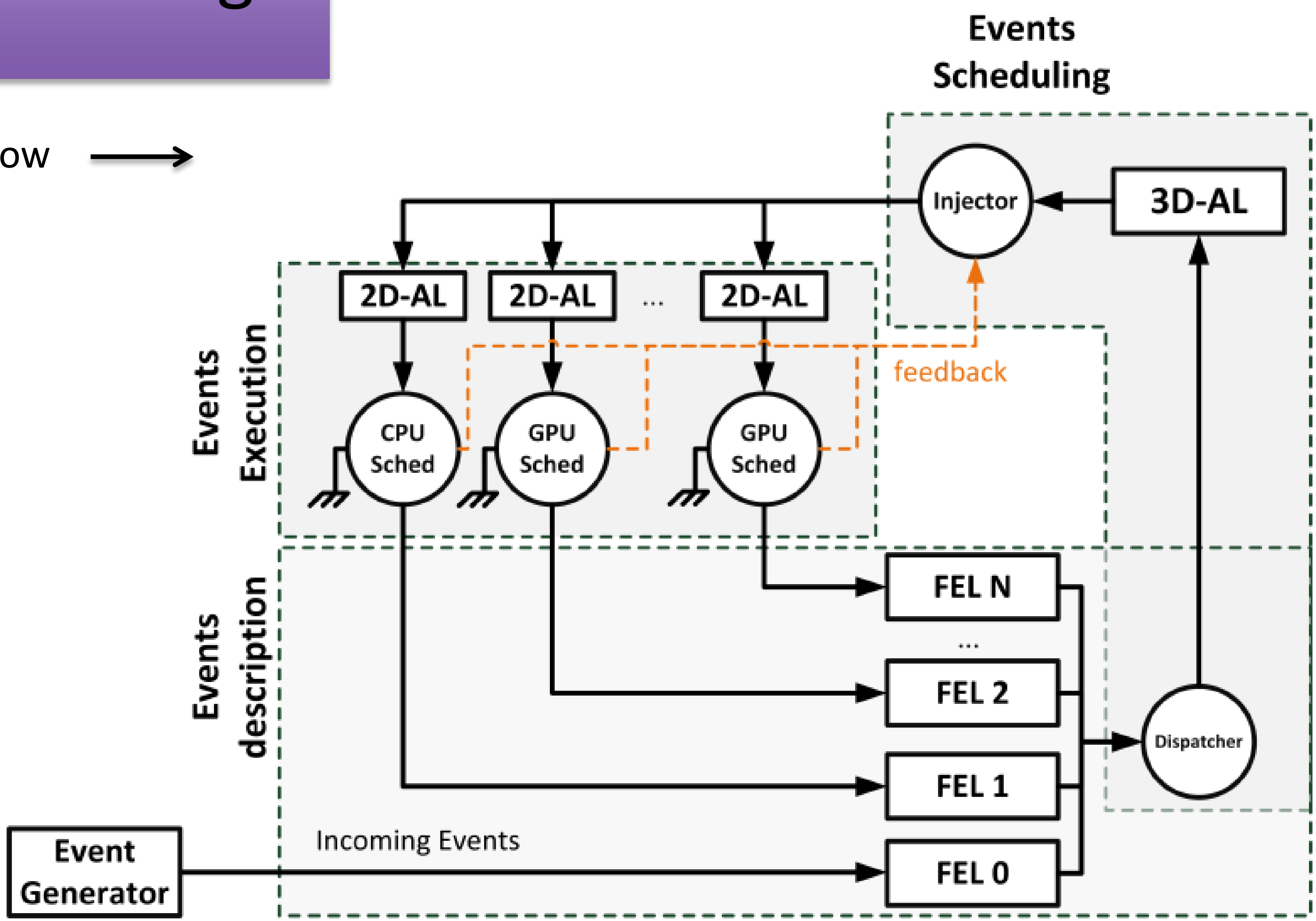
- System Bottleneck may change over time

➔ use feedback to maintain dynamically event rate stability to maximize the simulation efficiency

Hybrid scheduling

Architecture

Event flow →

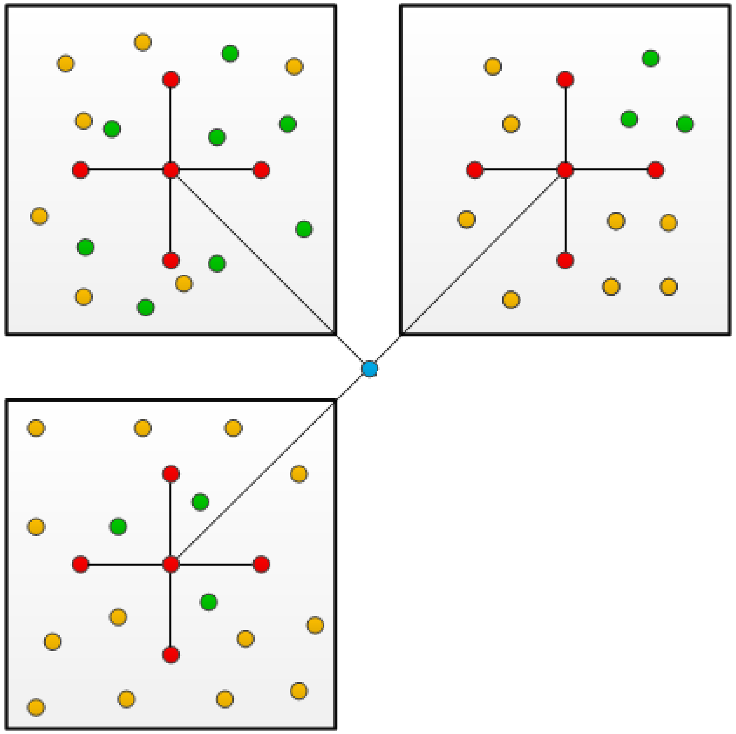


EURECOM

Hybrid scheduling

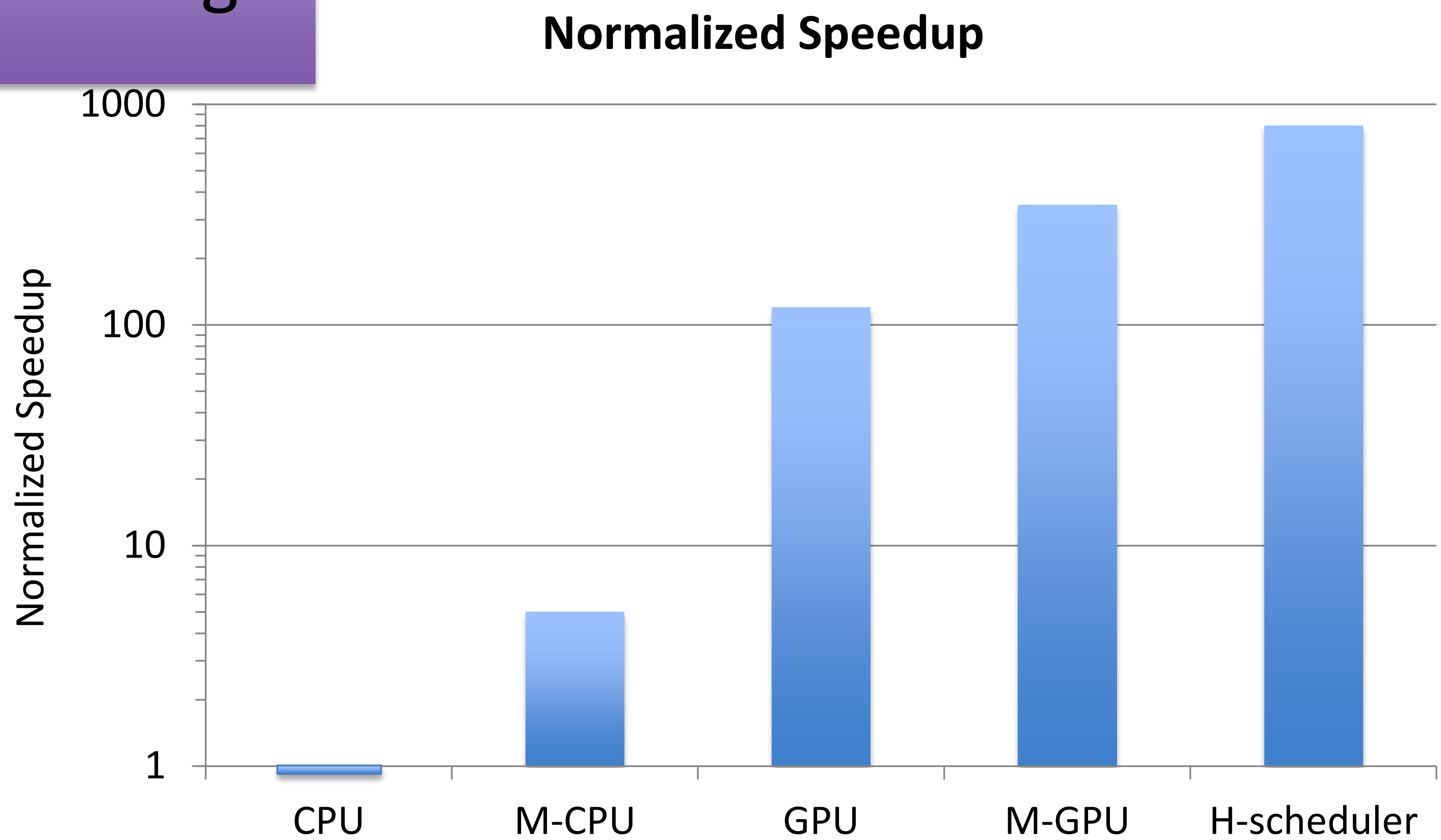
Validation Scenario

- Experiment setup
  - Cunetsim framework
  - 3 independent activity areas
  - 3 types of nodes
  - 525 K nodes per AA
  - 50 G packets each 128B
  - $600m^3$ Per AA
- Hardware setup
  - i7 3730k (6cores)
  - 64 Go of RAM
  - 3 GPUs : GTX 680
- Objective : Scalability



Hybrid scheduling

Comparative Results



- **H-Scheduler outperforms M-CPU by 150x and M-GPU by 2x**
- **Higher scheduling cost for H-Scheduler**

Hybrid scheduling

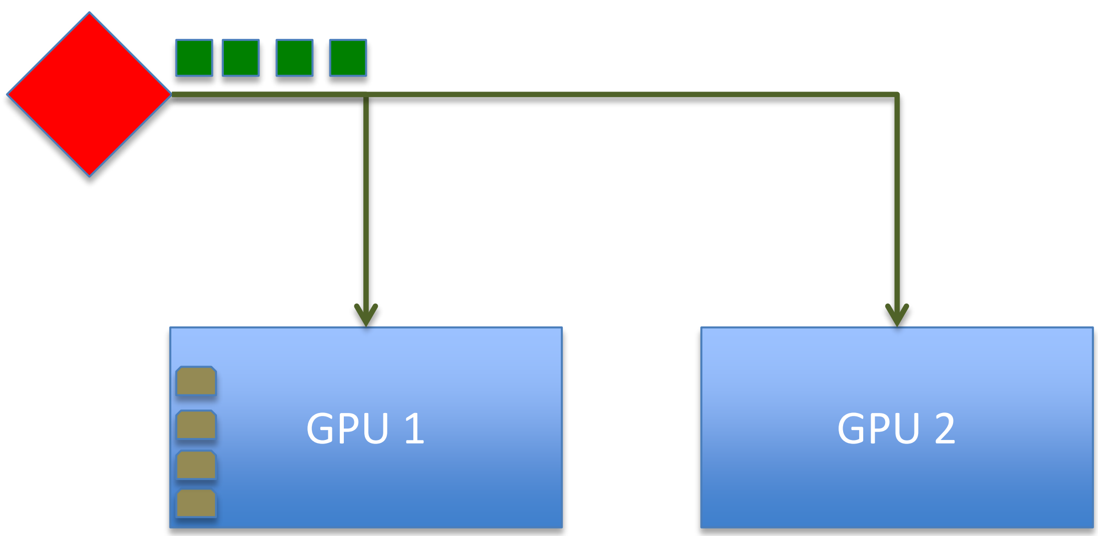
Discussion

- Scalability gain achieved by
  - Maximizing the hardware usage rate in a shared memory context
- Limitations
  - Simulation scalability due to limited memory size
  - Simulation instability due to data locality issue when swapping the target

Hybrid scheduling

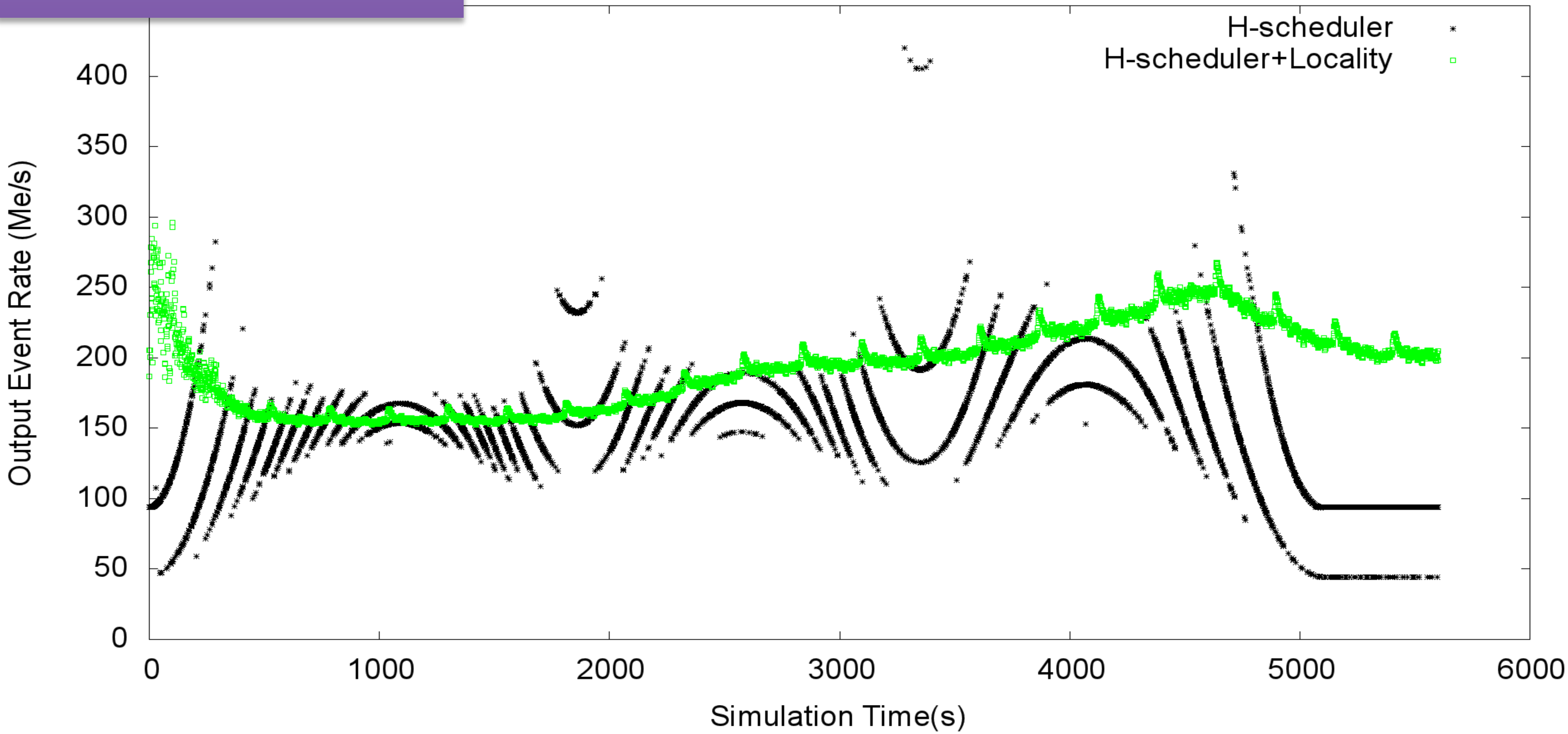
Locality Problem

- Consider the locality between the data and the event to determine the execution target



Hybrid scheduling

Comparative Results



- **Event rate stability is achieved (Event flow) → increase scalability**
- **Small Gain of 10%**

Outline

Introduction & Background

Cunetsim

Hybrid scheduling

CMW & GP-CMW

NS-3 as proof of concept

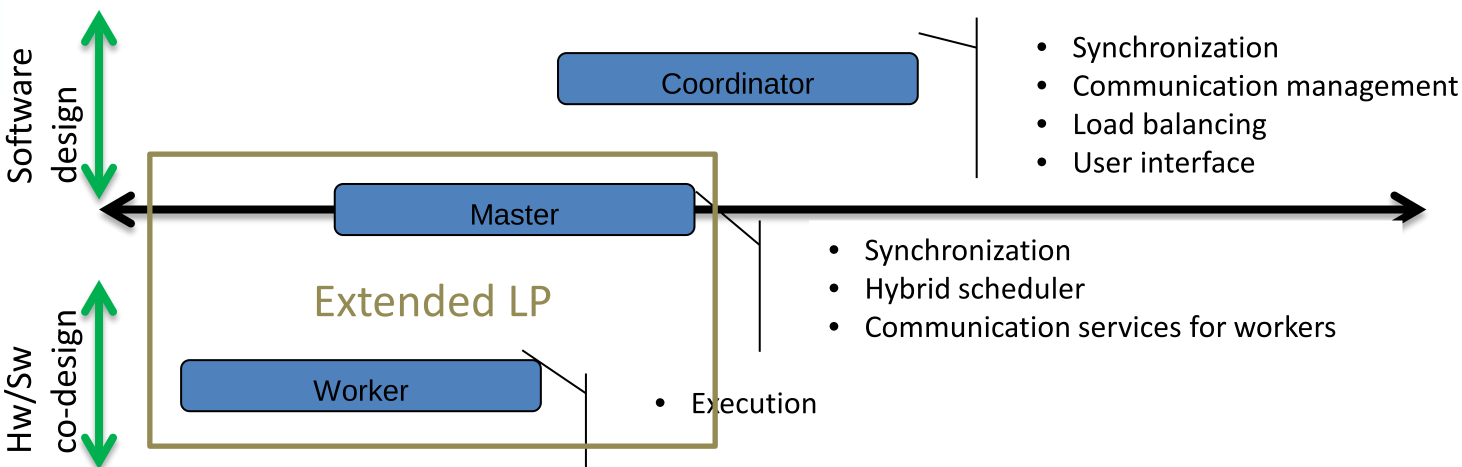
Conclusion & Future work

CMW
& GP-CMW

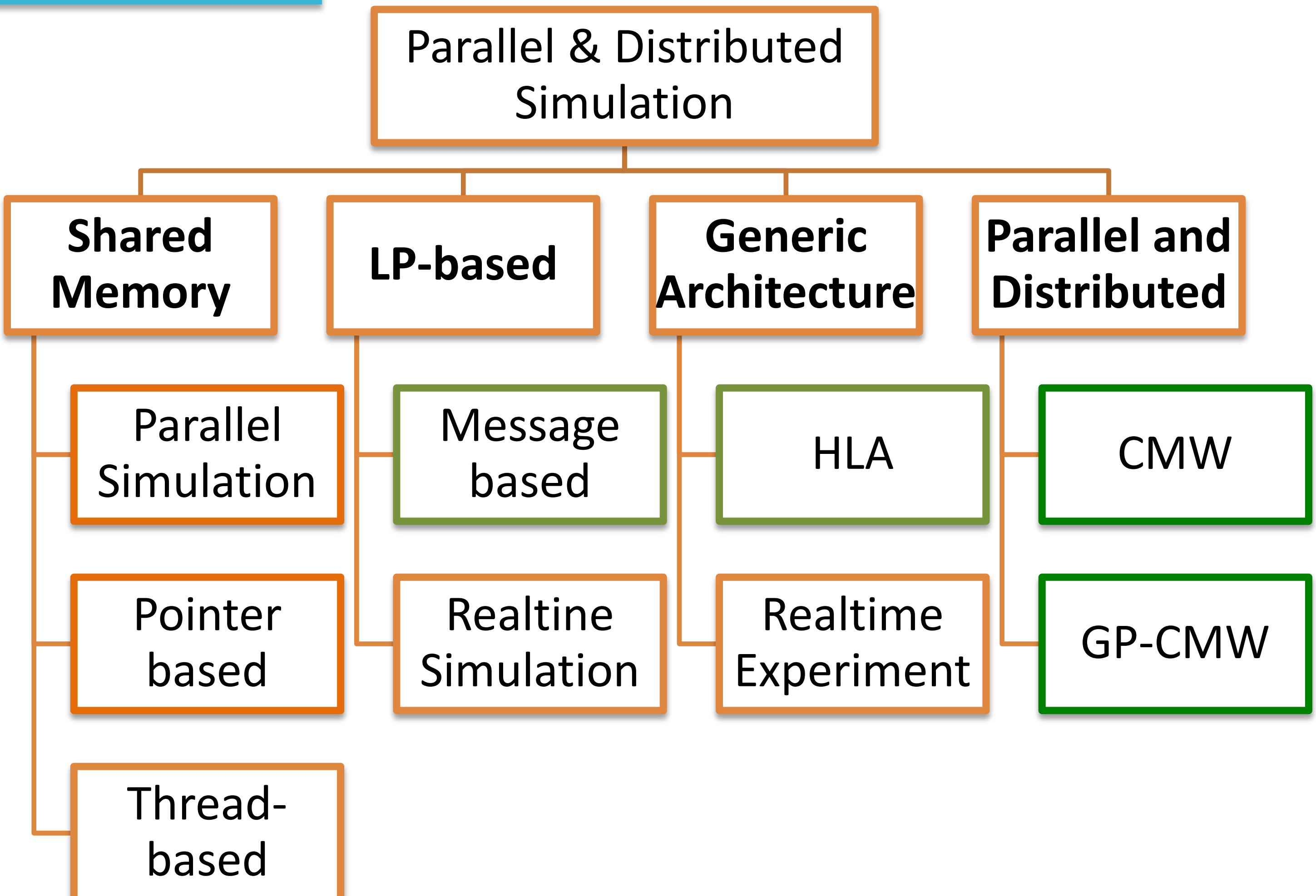
General Idea

- Shared memory context limits the scalability
- MW performs well in parallel or distributed simulation, **but not in both at the same time**

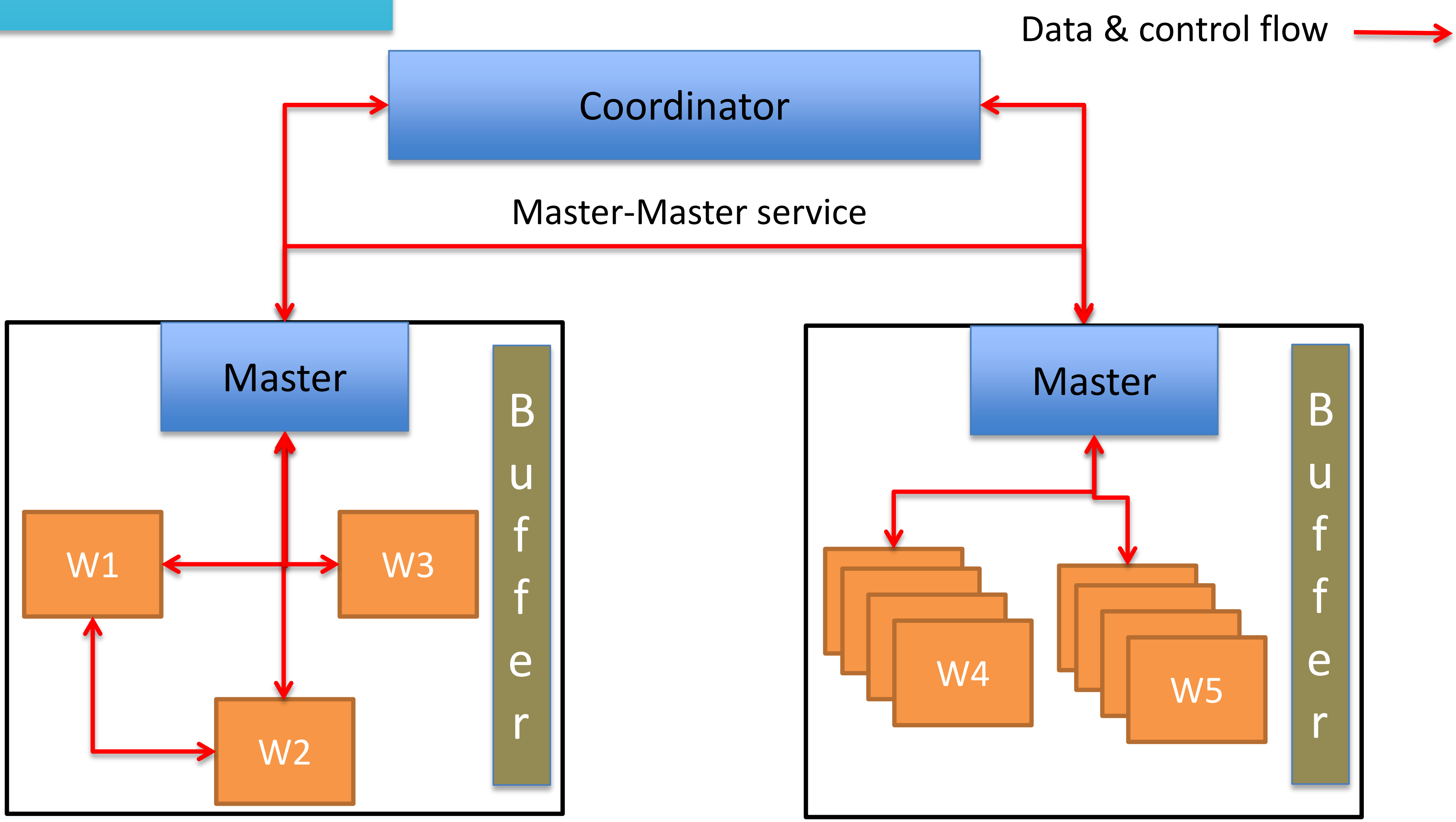
- Coordinator-Master-worker CMW
  - CM is optimized for distributed simulation
  - MW is optimized for parallel simulation



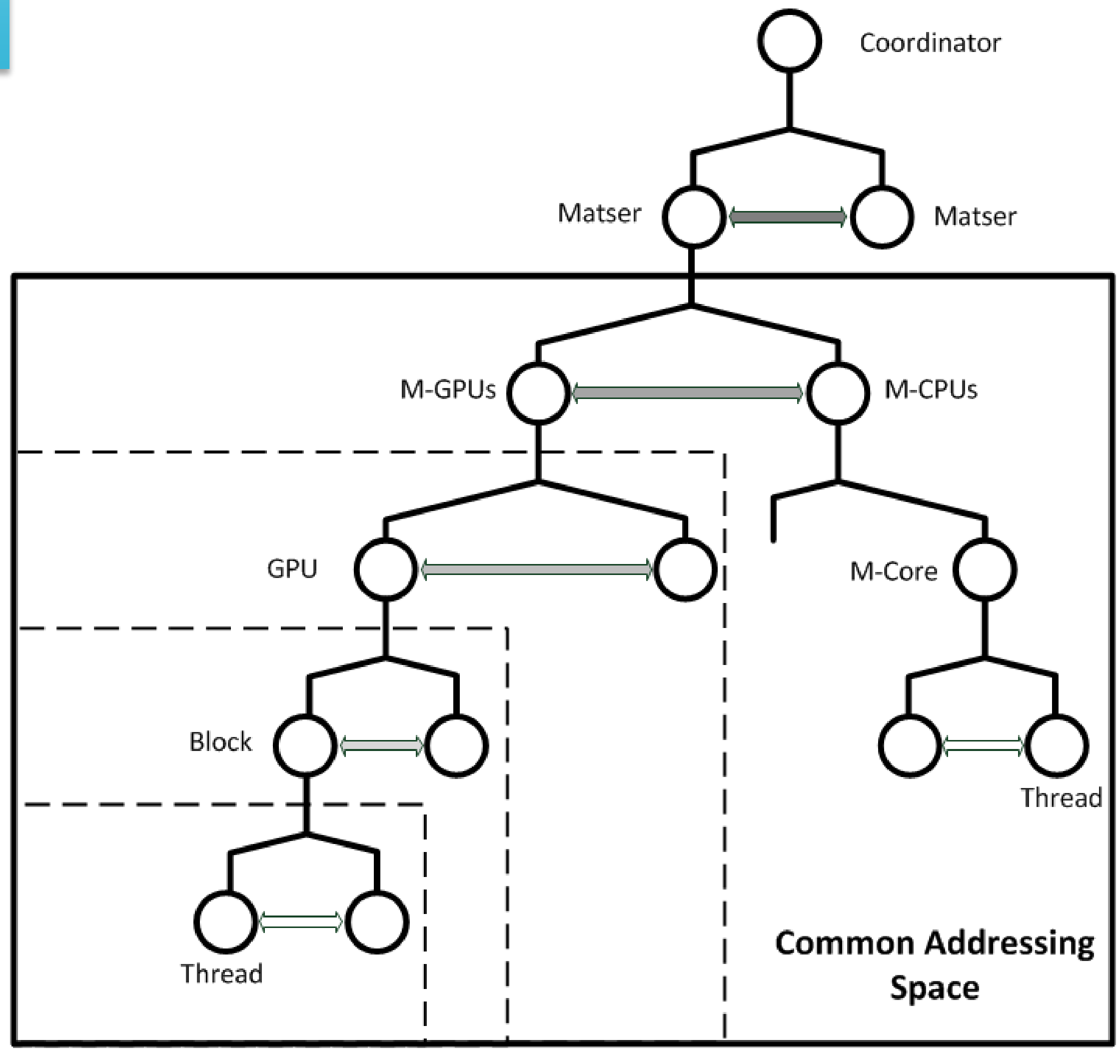
Related Works



CMW Model


Data & control flow
Coordinator
Master-Master service
Master
Buffer
W1
W3
W2
Master
Buffer
W4
W5

Addressing Space



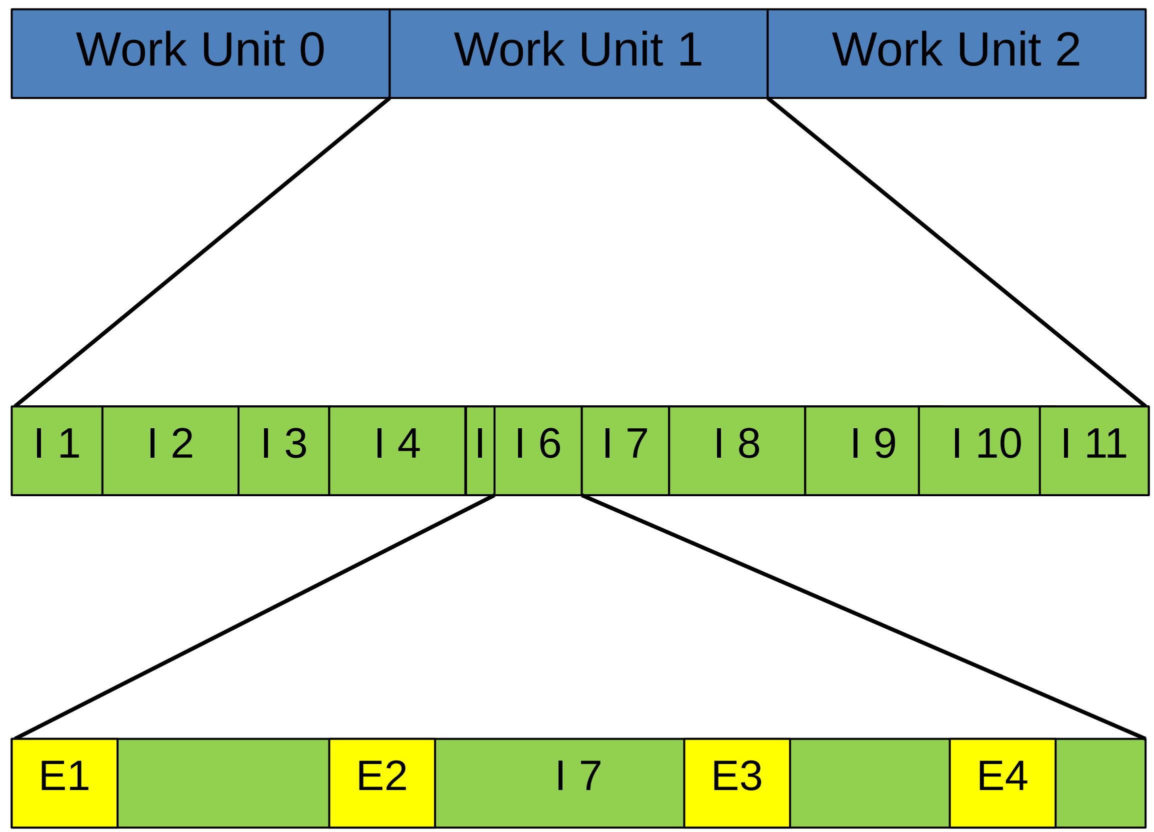
CMW & GP-CMW

CMW Synchronization

Coordinator
Time-driven clock

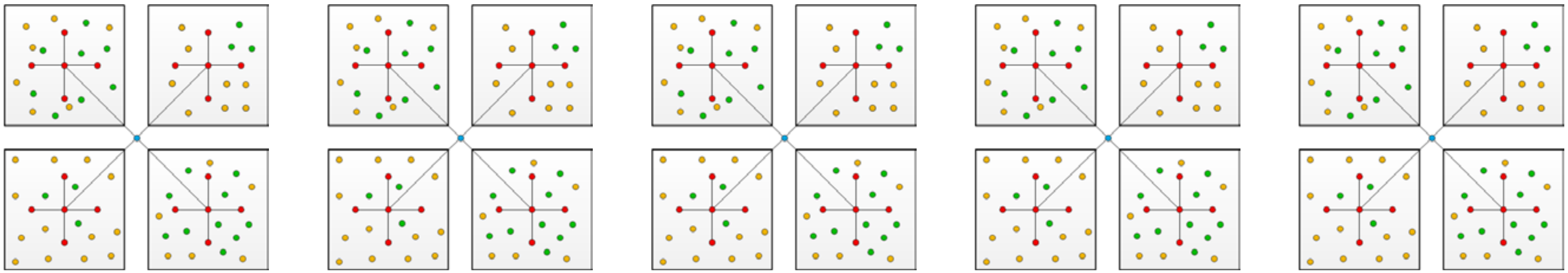
Master
Time-driven clock +
Lookahead protocol

Worker
Event-driven clock



CMW
& GP-CMW

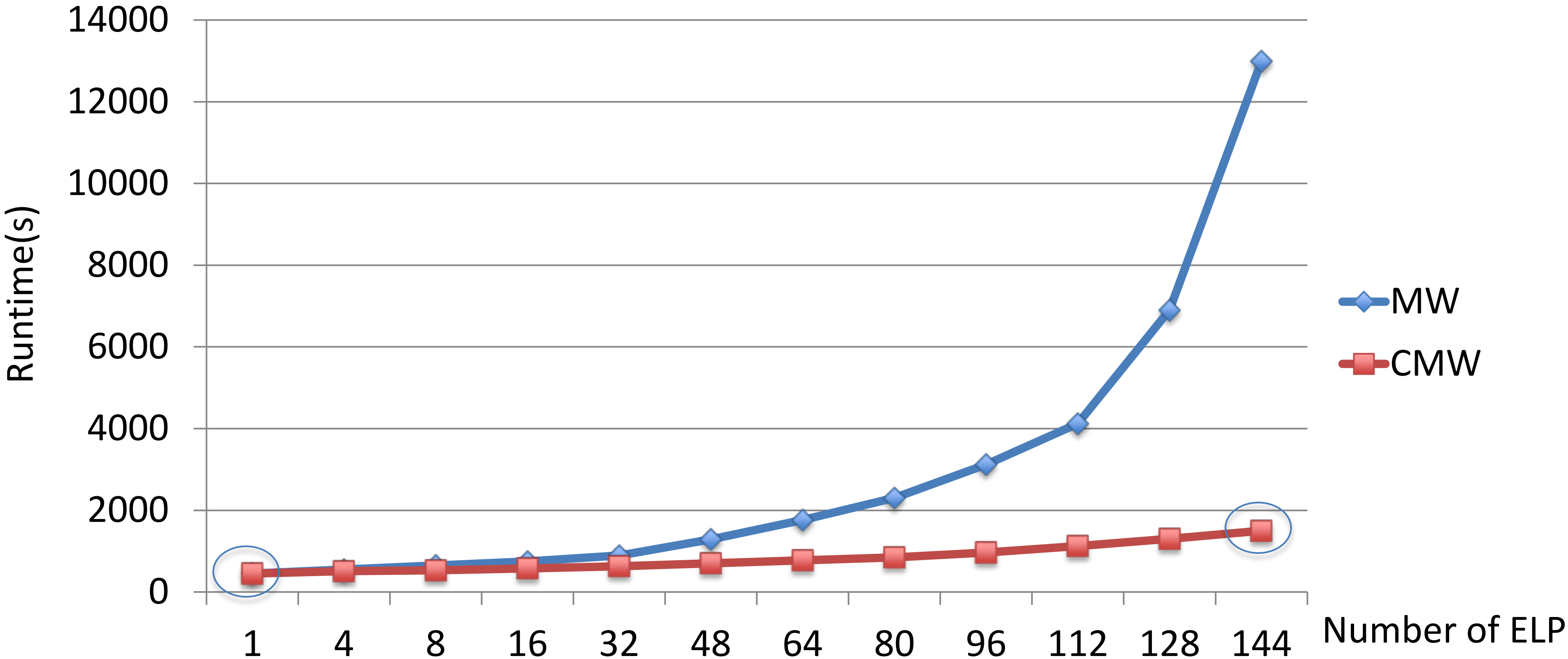
Validation Scenario



- Extension of the H-scheduler scenario.
- 250 M nodes
- P2P interconnection.
- 1-144 ELPs, 6002 seconds.
- TGCC Curie Infrastructure
  - Hybrid Nodes (144)
    - 2 CPU (8 cores)
    - 2 GPU (1024 Cores)
  - 192 Tflops , 528 Go GRAM, 46 To RAM
- Benchmarking: MW and CMW models

CMW
& GP-CMW

Result (simulation runtime)

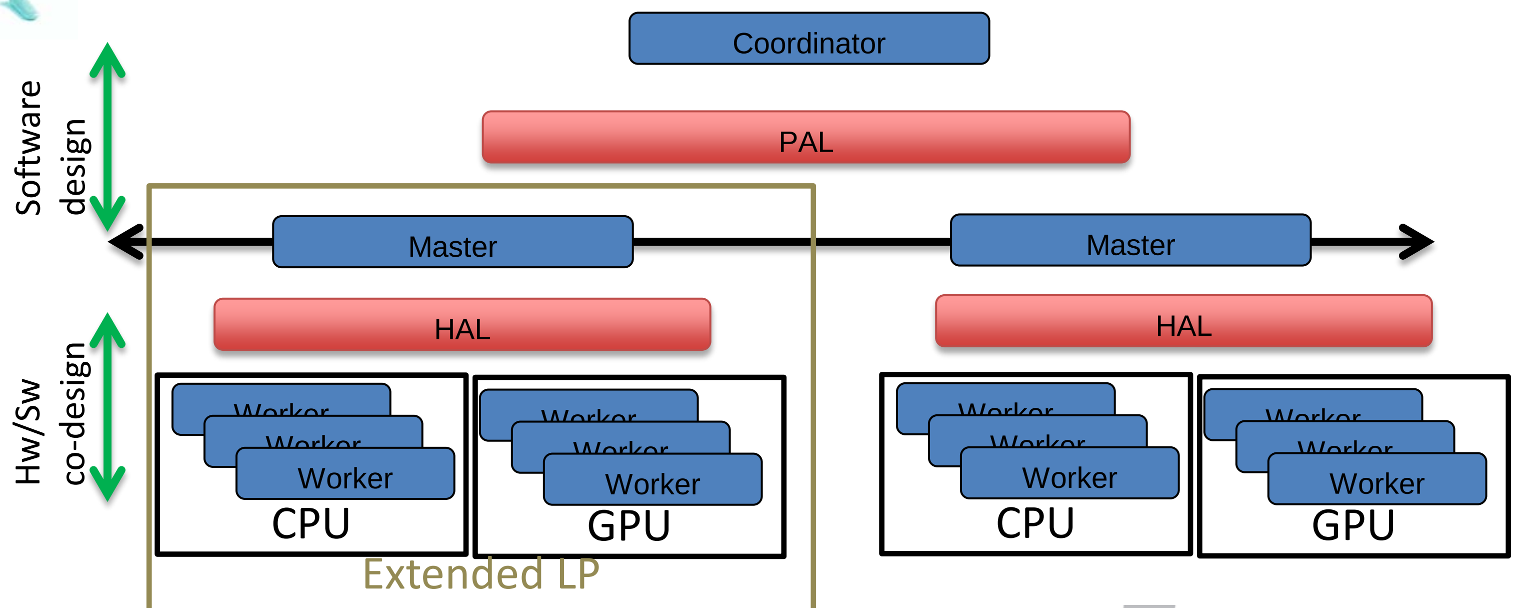


- Runtime stability is achieved
- Gain of 10 times compared to MW
- CMW introduces 3 times larger overhead as the number of ELP increases

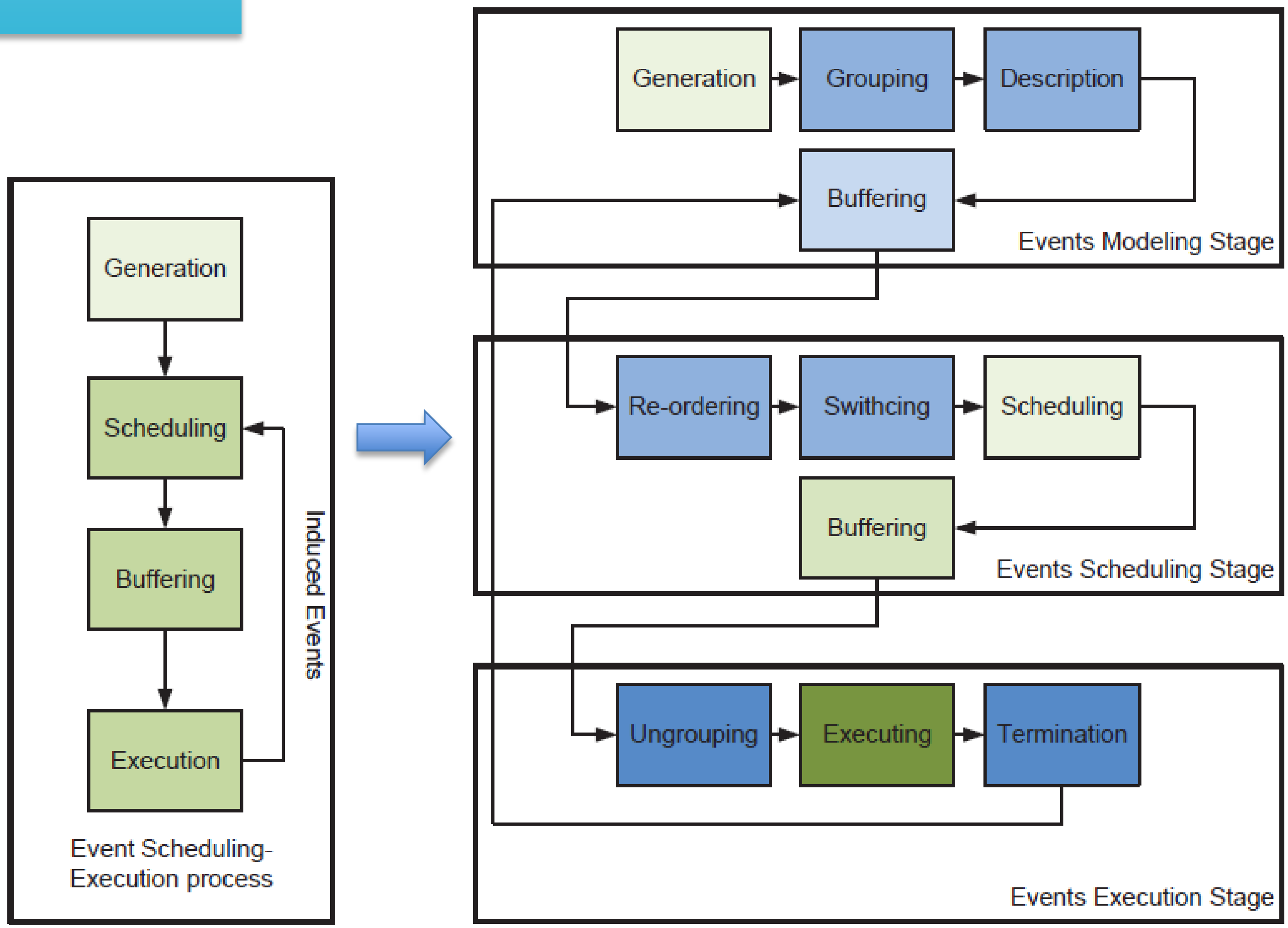
CMW
& GP-CMW

General Idea

- GP-CMW : Management overhead and locality
  - Priority Abstraction layer
    - Separates control plane from data plane
  - Hardware abstraction layer
    - Exploits data and communication locality



Event Lifecycle



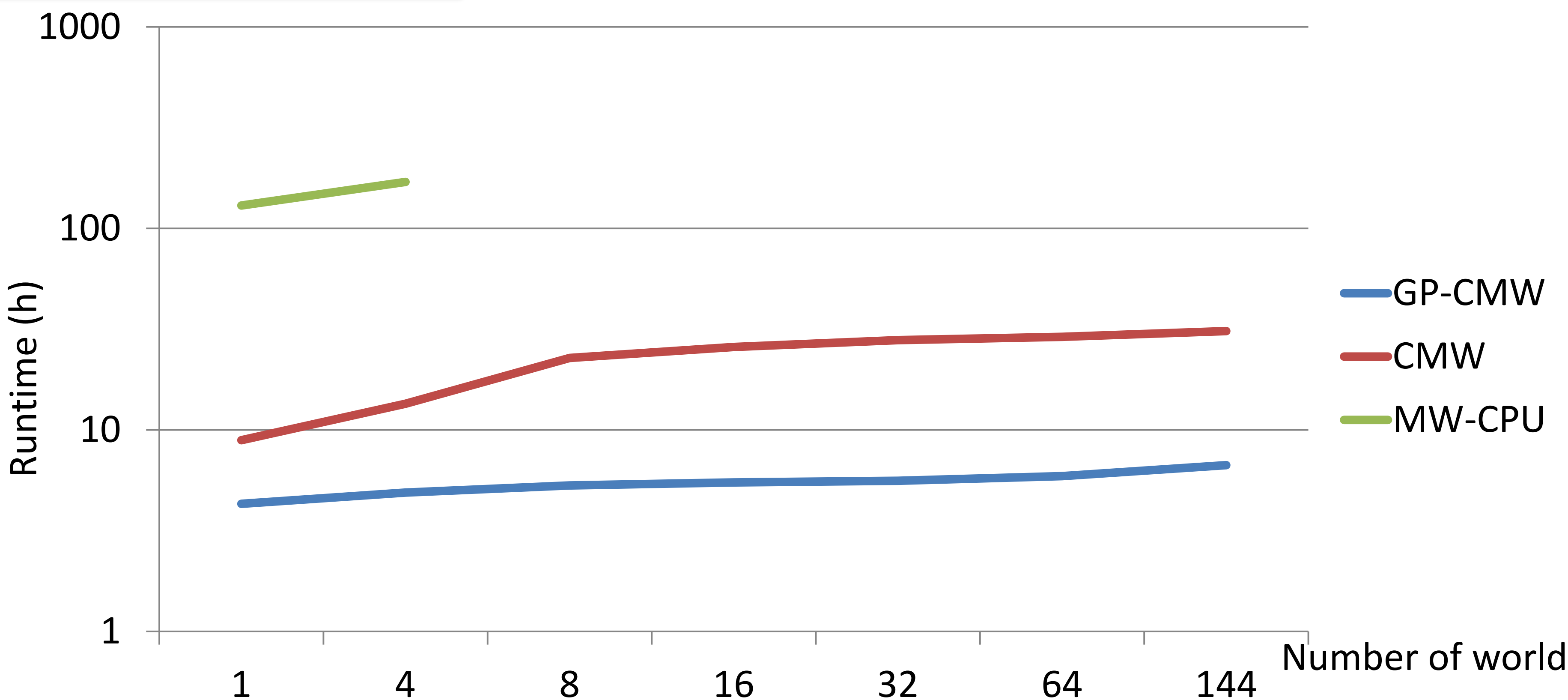
CMW
& GP-CMW

Experiment Setup

- Massive multi-player online game simulation
  - Command and conquer
- 144worlds, 25k-50k players/world, 1-20 bases/ players, 1-3 plans per base, 1-40 elements per plan
- Only 10% of players communicate with different worlds
  Simulating one year of the game with 144 ELPs
- Time stamp is 1 minute (1 Year is 525600)
- TGCC Curie Infrastructure

CMW & GP-CMW

Results



- GP-CMW outperforms CMW by 4.5 times

Conclusion

- GP-CMW combines parallel and distributed simulation in one optimized architecture
  - Introduce a Coordinator as a top level actor

- Limitation
  - Worker migration (mobility conditions)
  - Load balancing
  - System observation overhead

Outline

Introduction & Background	Cunetsim	Hybrid scheduling
CMW & GP-CMW	**NS-3 as proof of concept**	Conclusion & Future work

NS-3 as proof of concept

Highlights

- NS-3 : most popular DES network simulator.
  - Layered software architecture
  - Sequential execution
  - Scalability is achieved through distributed simulation (official branch)

- When targeting parallel simulation

the event scheduler is identified as the main bottleneck

NS-3 as proof of concept

General Idea

Explicit CPU parallelism

Implicit CPU parallelism

GPU offloading

Hybrid scheduling

NS-3 as proof of concept

Proposed Modifications

Event execution is made by a pool of threads.

The scheduler make the decision

The event execution is made by a pool of threads but the scheduler see only one event (framework: OpenMP)

Selected events will be offloaded to the GPU

(Framework: OpenACC)

Selected events will be forwarded to dedicated process that choose their executed target.

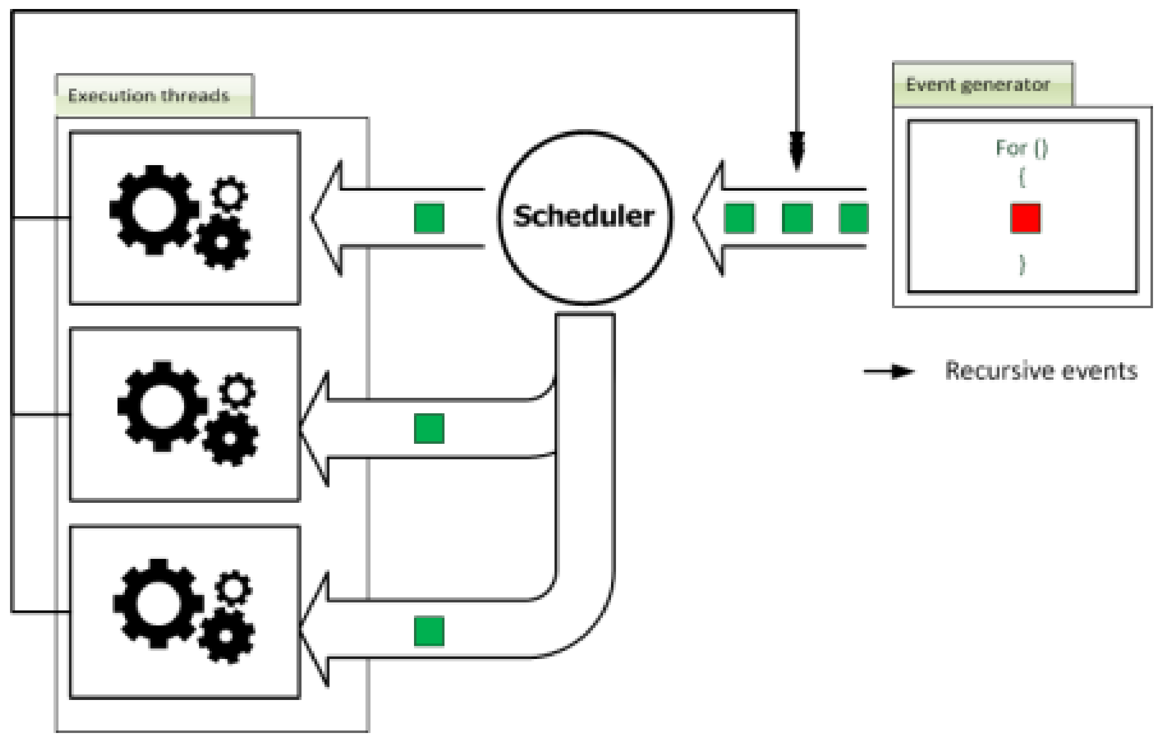
(OpenMP+MPI+OpenACC)

NS-3 as proof of concept

Explicit CPU Parallelism

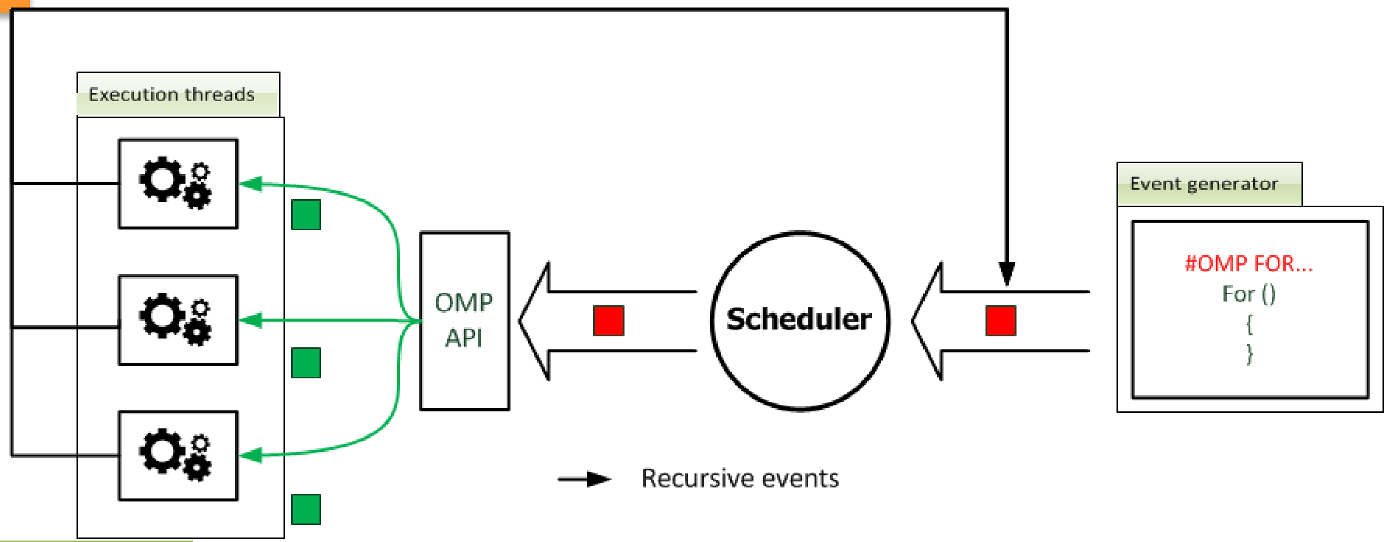
Event execution is made by a pool of threads.

The scheduler make the decision



NS-3 as proof of concept

Implicit CPU Parallelism



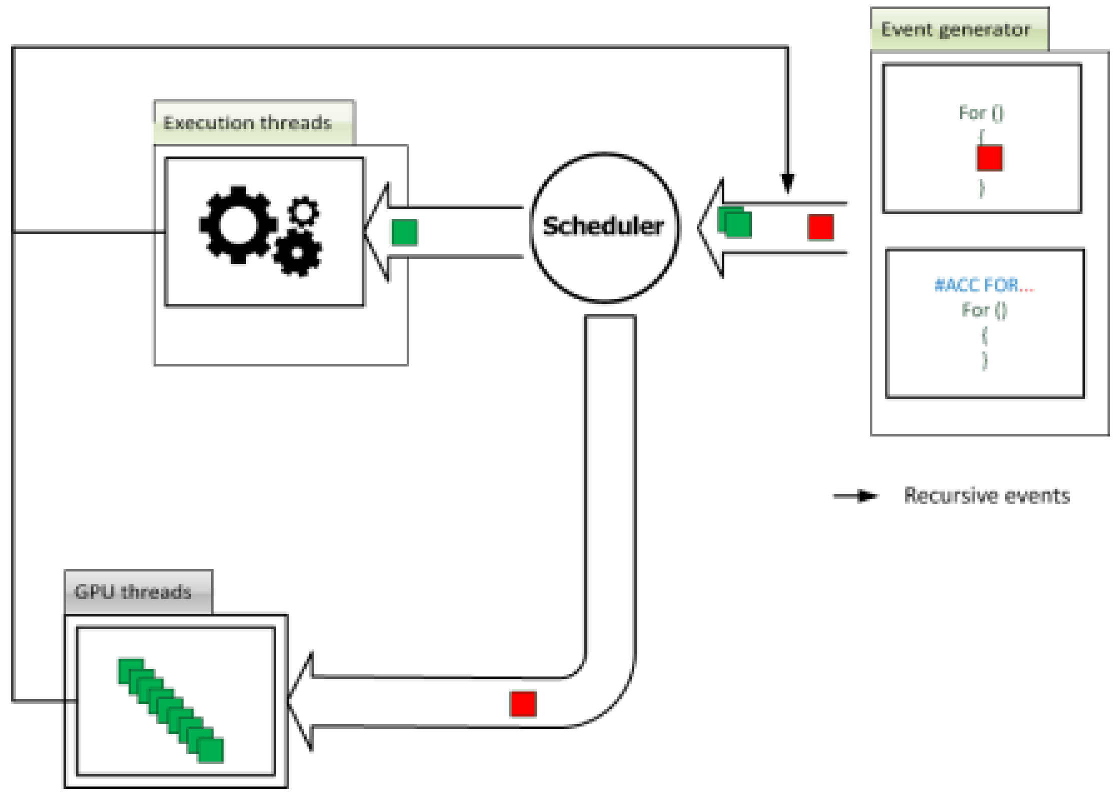
The event execution is made by a pool of threads but the scheduler see only one event (framework: OpenMP)

NS-3 as proof of concept

GPU Offloading

Selected events will be offloaded to the GPU

(Framework: OpenACC)

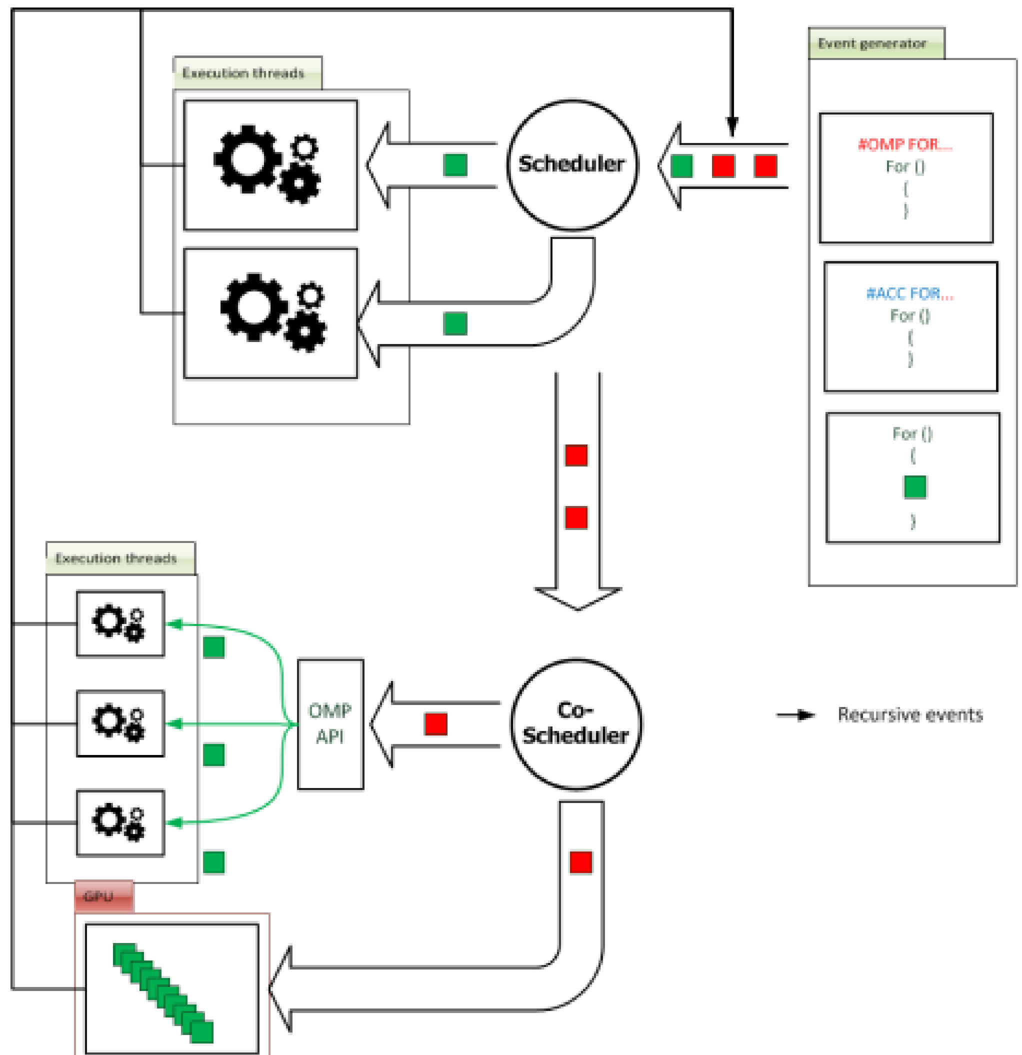


NS-3 as proof of concept

Hybrid Scheduler

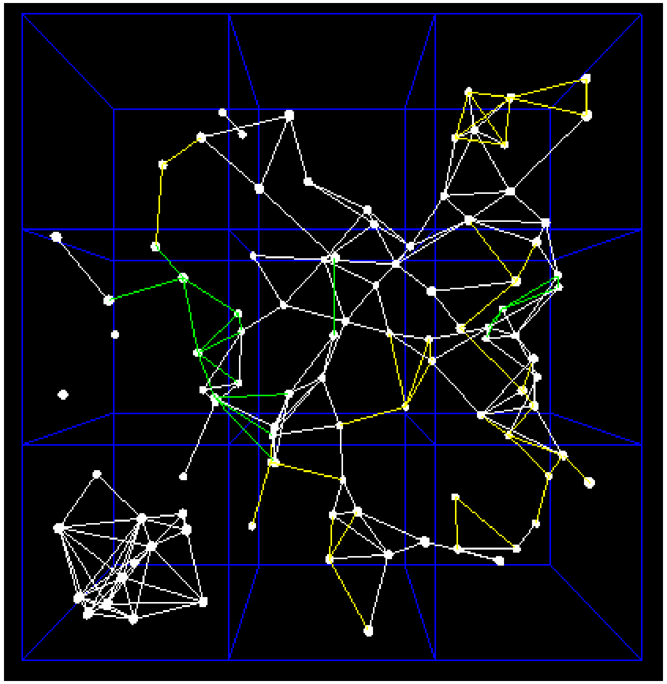
Selected events will be forwarded to dedicated process that choose their executed target.

(OpenMP+MPI+OpenACC)



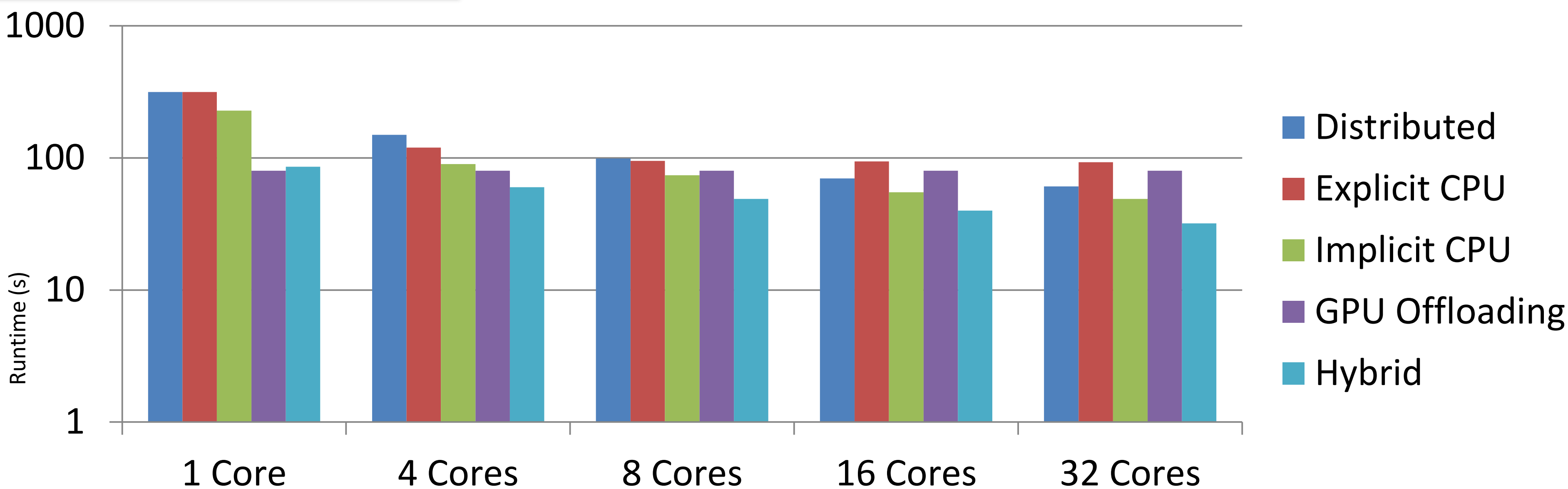
Experiment Setup

- Benchmarking Scenarios
  - 64k Nodes
  - 1600*1600*1600(3D)
  - RWP mobility,
  - UDG Connectivity
  - Flooding Proto
- Fair comparison
  - Stop the simulation when reaching 250 M events
  - Limiting the modification to the scheduler or the event generator
- Framework used
  - MPI,
  - OMP
  - OpenACC
  - PGI compiler



NS-3 as proof of concept

Performance Results (1K nodes, 250K events)



- Distributed architecture introduces an overhead but scales well
- Explicit CPU parallelism caps with 8 cores (due to the scheduling bottleneck)
- Implicit CPU parallelism handles easily large CPUs
- GPU offloading provides a real gain if used as a co-processor
- Hybrid approach maximize the hardware usage and scale well with heterogeneous resources.

Outline

Introduction & Background

Cunetsim

Hybrid scheduling

CMW & GP-CMW

NS-3 as proof of concept

Conclusion & Future work

Conclusion & Future work

Conclusion

Approach\ characteristic	Efficiency	Scalability	Overhead	Stability
Hybrid Scheduling	+++	++	--	--
GP-CMW		+++	+++	+++

1. Event grouping
2. Multi-target execution with locality consideration
3. Overhead management
   - Separation of control and data plane
   - Simulation data aggregation

Conclusion

- Recent hardware is heterogeneous and massively programmable.
  - Heterogeneous Execution
  - Smart usage of available resources allows a new scalability level.
  - To simplify the operation we need:
    - More high level API.
    - More intelligent IDE.
    - More integrated Hardware (SoC+ unified memory)

Future Work

- Massively parallel X86 processors
  - Hybrid execution of existing framework over such infrastructure guarantees a smooth migration to next software.
- Efficient hardware abstraction
  - Automatic parallelism and hardware detection
  - Automatic memory management
- Simulation as a service
  - Multi-target execution on virtual infrastructure without full knowledge of hardware
    - Hardware abstraction

Future Work

- Tracing and Data management
  - With the increasing number of events the data size and the required throughput becomes imposing.
    - E.g. Layers-based compression
- Worker migration
- Load balancing across ELPs