

# THE SNIPER MULTI-CORE SIMULATOR

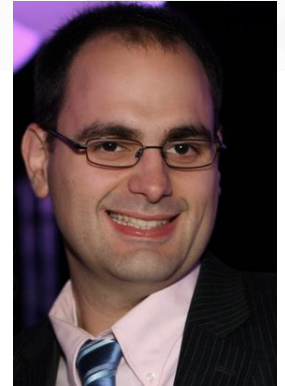
09:00	INTRODUCTION
09:30	INTERVAL SIMULATION
10:00	— COFFEE BREAK —
10:30	SIMULATOR INTERNALS
11:00	VALIDATION RESULTS
11:30	RUNNING SIMULATIONS AND PROCESSING RESULTS
12:00	— END —

# WHO WE ARE



Wim Heirman

- [wim.heirman@elis.ugent.be](mailto:wim.heirman@elis.ugent.be)
- Post-doctoral researcher at the Intel ExaScience Lab
- MS and PhD degrees from Ghent University in 2003 and 2008
- Interests
  - Fast and accurate simulation
  - Architecture exploration and software analysis through co-design
  - Energy efficient HPC

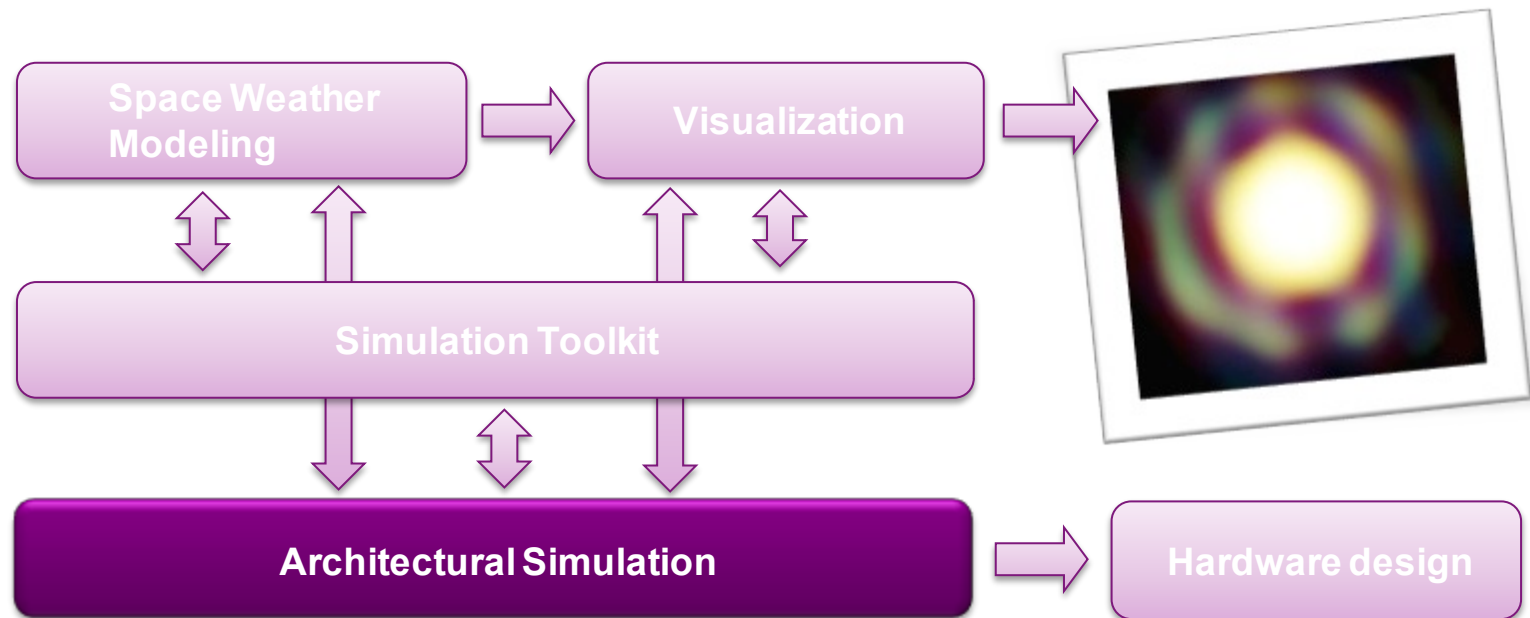


Trevor E. Carlson

- [trevor.carlson@elis.ugent.be](mailto:trevor.carlson@elis.ugent.be)
- Ph.D. student at Ghent University and part of the Intel ExaScience Lab
- BS and MS degrees from Carnegie Mellon University in 2002 and 2003
- Most recently worked as a researcher at IMEC where he investigated efficient embedded and 3D-stacked architectures
- Previously a Staff Engineer at IBM with 4 issued patents

# INTEL EXASCIENCE LAB

- Collaboration between Intel, imec and 5 Flemish universities
- Study Space Weather as an HPC workload





**ExaScience Lab**  
Intel Labs Europe



**EXASCALE COMPUTING**

# THE SNIPER MULTI-CORE SIMULATOR INTRODUCTION

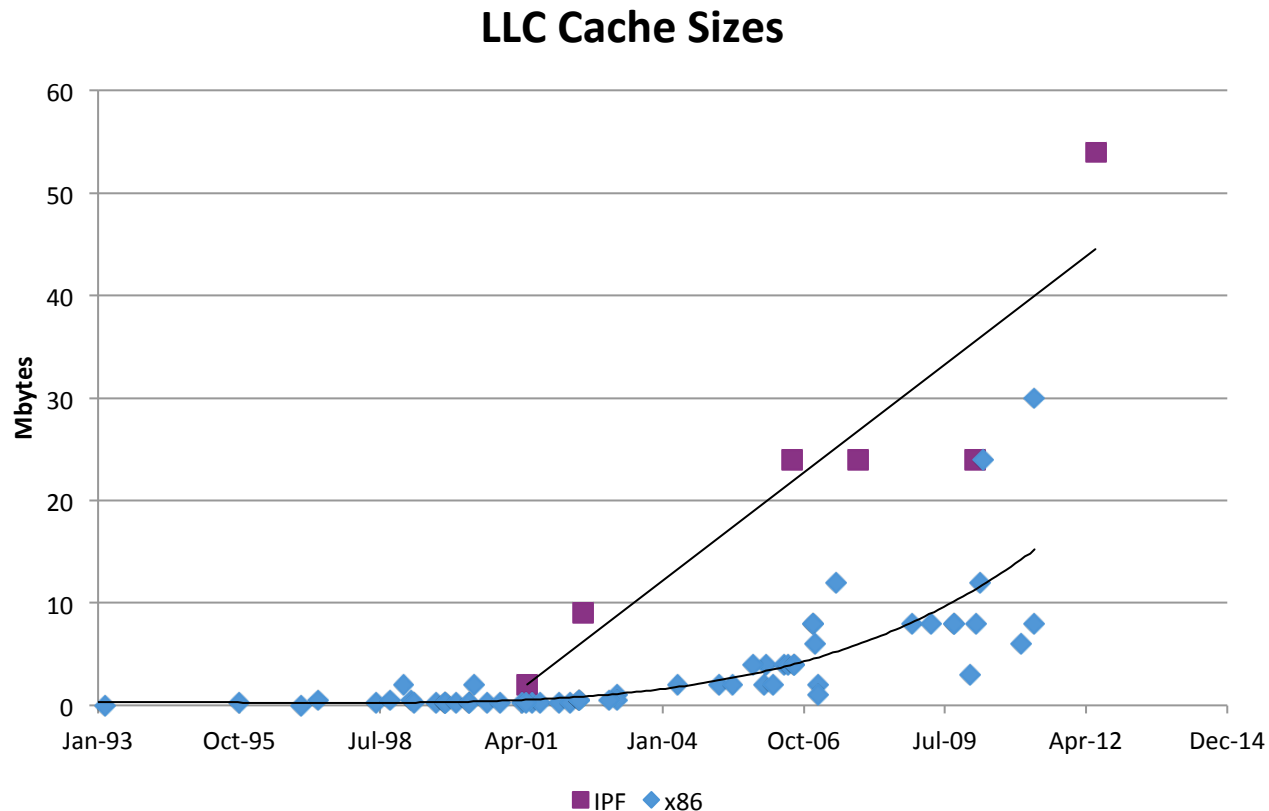
WIM HEIRMAN, TREVOR E. CARLSON  
AND LIEVEN EECKHOUT



[HTTP://WWW.SNIPERSIM.ORG](http://www.snipersim.org)  
SUNDAY, APRIL 1<sup>ST</sup>, 2012  
ISPASS, NEW BRUNSWICK, NJ

# TRENDS IN PROCESSOR DESIGN: CACHE

- Cache sizes are increasing



# TRENDS IN PROCESSOR DESIGN: CORES

---

- Number of cores per node is increasing
  - 2001: Dual-core POWER4
  - 2005: Dual-core AMD Opteron
  - 2011: 10-core Intel Xeon Westmere-EX
  - 201x: Intel MIC Knights Corner (50+ cores)

# SIMULATION

---

- Design tomorrow's processor using today's hardware
- Simulation
  - Obtain performance characteristics for new architectures
  - Architectural exploration
  - Early software optimization

# DEMANDS ON SIMULATION ARE INCREASING

---

- Increasing core counts
  - Linear increase in simulator workload
  - Single-threaded simulator sees a rising gap
    - workload: increasing target cores
    - available processing power: near-constant single-thread performance of host machine
  - Need to use all cores of the host machine
  - Parallel simulation



# DEMANDS ON SIMULATION ARE INCREASING

---

- Increasing cache size
  - Need a large working set to fully exercise a large cache
  - Scaled-down applications won't exhibit the same behavior
  - Long-running simulations are required

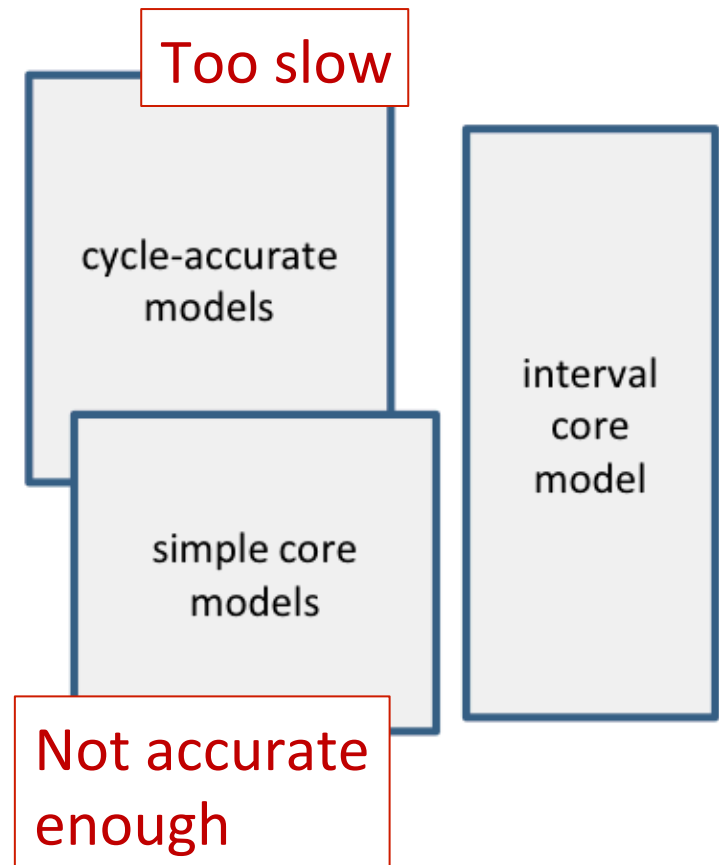
# UPCOMING CHALLENGES

---

- Future systems will be diverse
  - Varying processor speeds
  - Varying failure rates for different components
  - Homogeneous applications become heterogeneous
- Software and hardware solutions are needed to solve these challenges
  - Handle heterogeneity (reactive load balancing)
  - Be fault tolerant
  - Improve power efficiency at the algorithmic level (extreme data locality)
- Hard to model accurately with analytical models

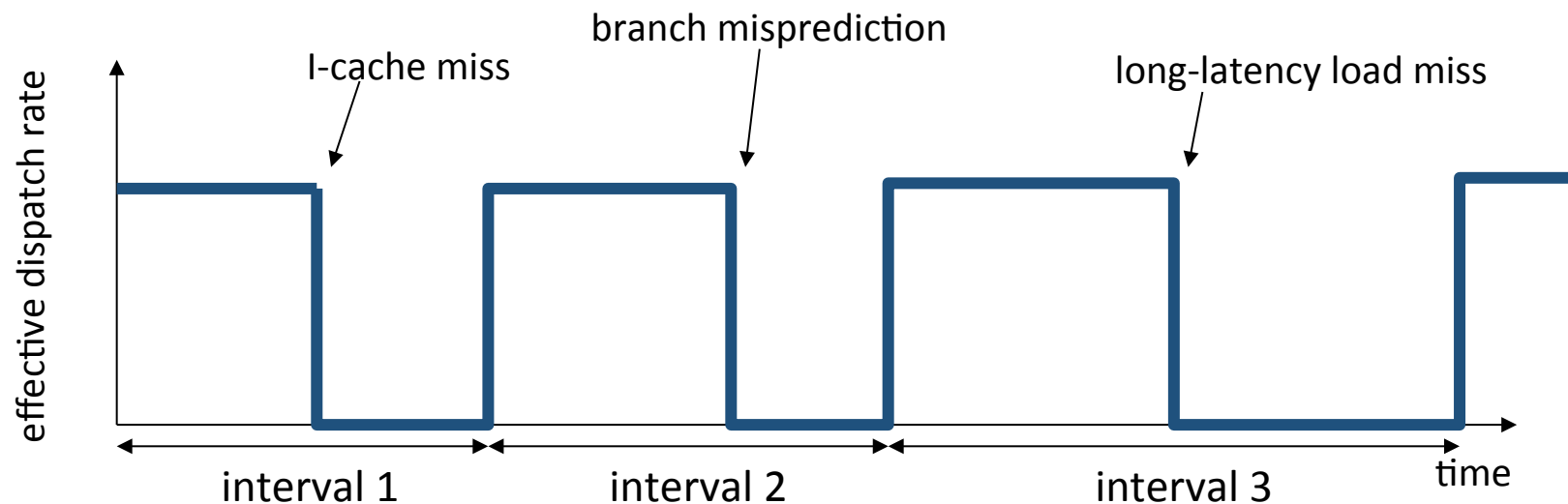
# NEEDED DETAIL DEPENDS ON FOCUS

Component	Single-event time scale	Required sim time
RTL	single clock cycle	millions of cycles
OOO execution		
Core memory ops		
L1 cache access		
LLC access		
Off-socket	microseconds	seconds



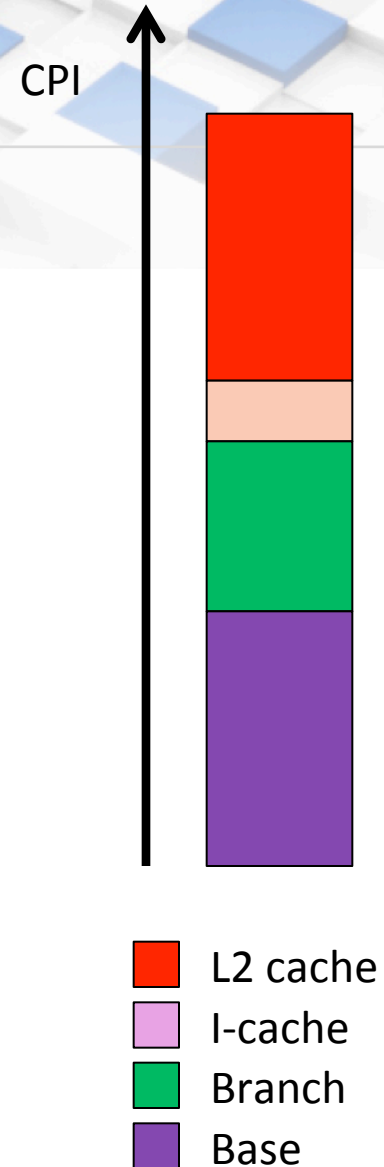
# INTERVAL SIMULATION

- Out-of-order core performance model with in-order simulation speed



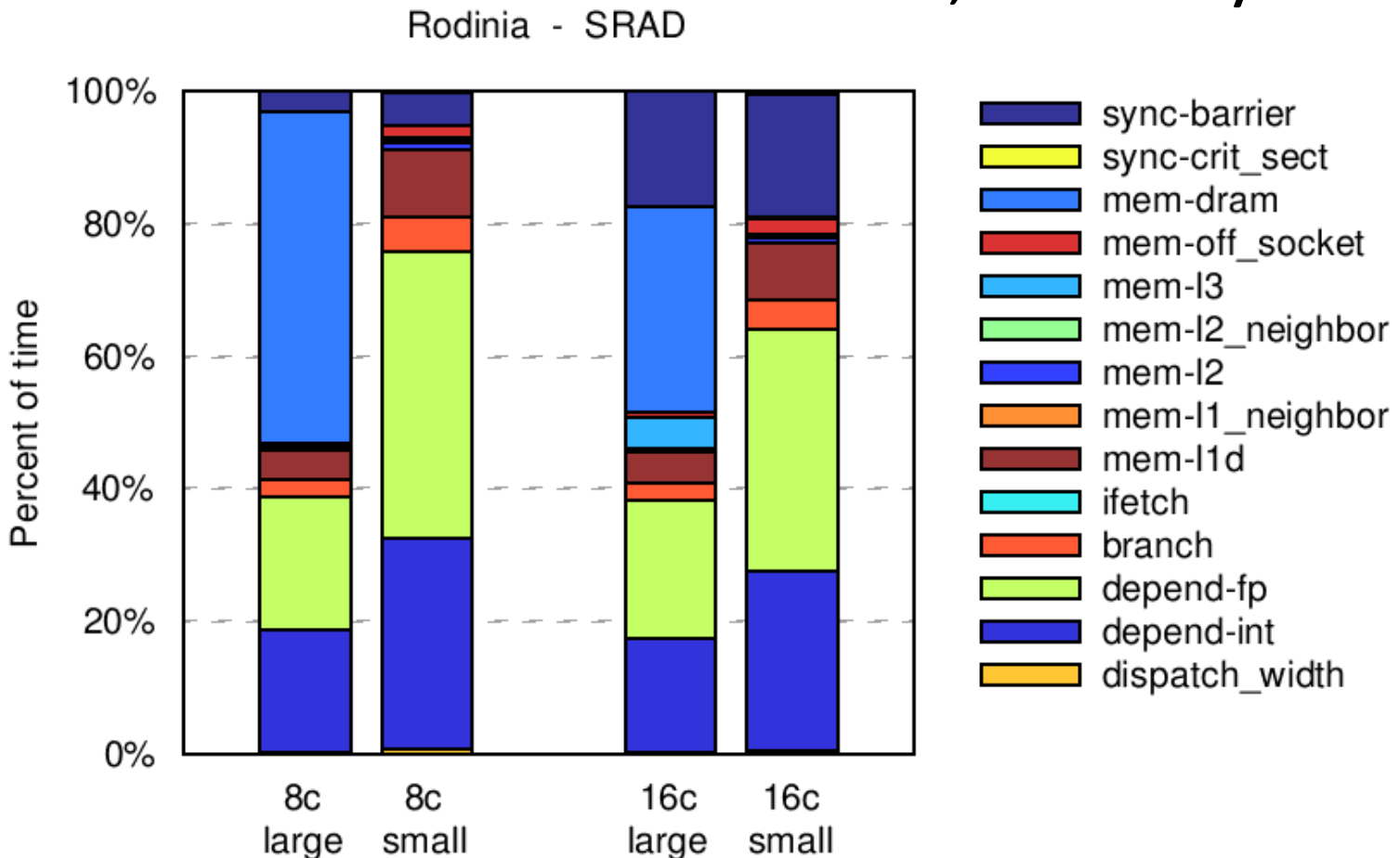
# CYCLE STACKS

- Where did my cycles go?
- CPI stack: cycles per instruction, broken up in components
- Normalize by either
  - Number of instructions (CPI stack)
  - Execution time (time stack)
- Different from miss rates as cycle stacks directly quantify the effect on performance



# CYCLE STACKS AND SCALING BEHAVIOR

- Scaling to more cores, larger input set size
- How does execution time scale, and why?

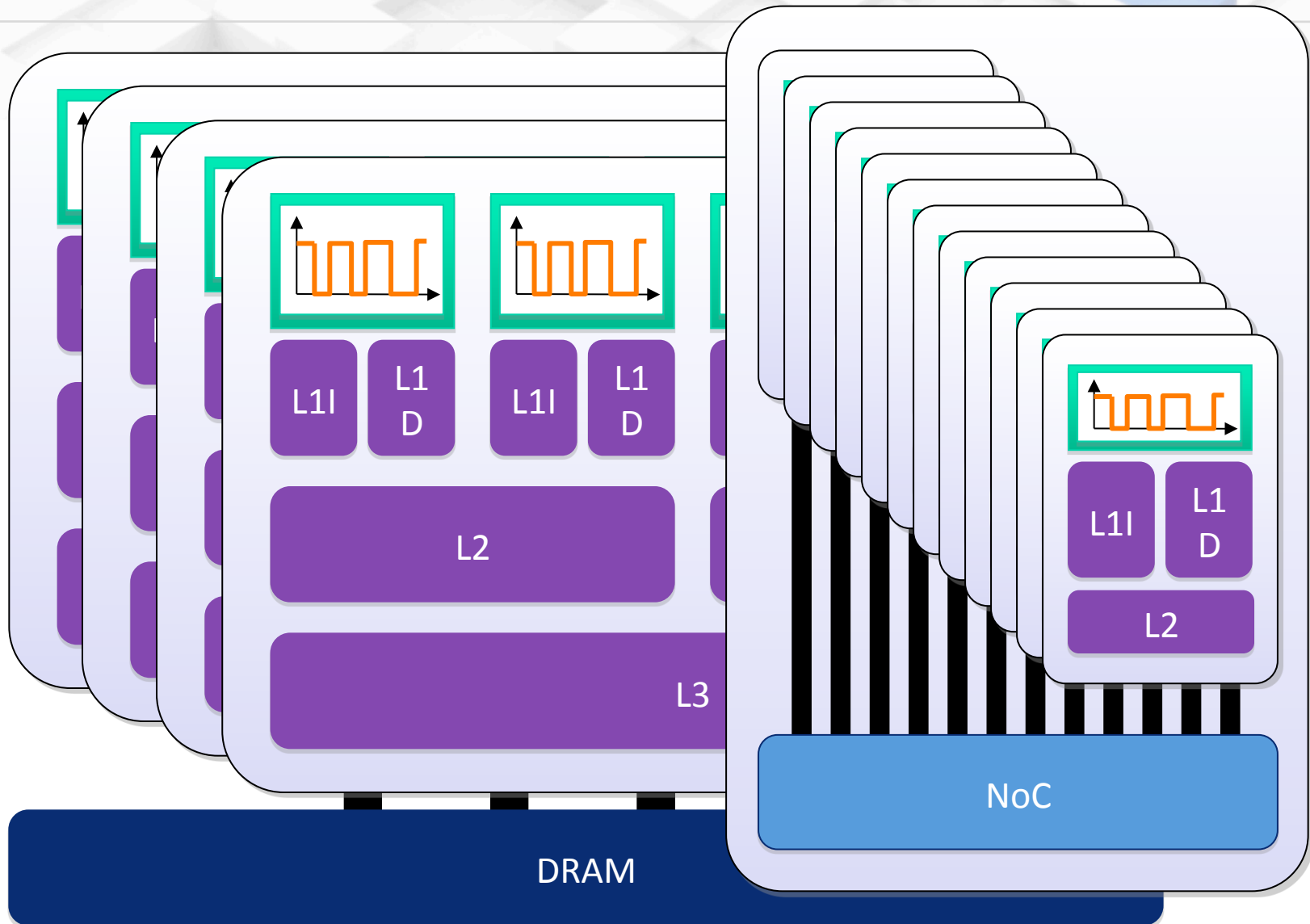


# FAST AND ACCURATE SIMULATION IS NEEDED

- Sniper Simulator
  - Interval core model
  - Accurate structures (caches, branch predictors, etc.)
  - Parallel simulator scales with the number of simulated cores
- Key Questions
  - What is the right level of abstraction?
  - When to use these abstraction models?



# MANY ARCHITECTURE OPTIONS



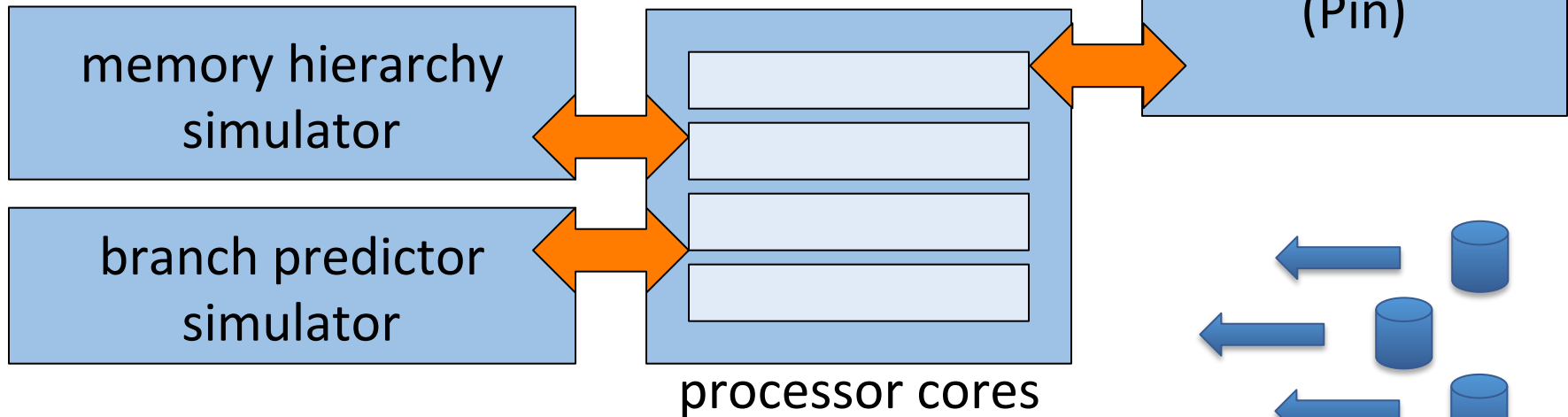


# SIMULATION IN SNIPER

Execution-driven simulation

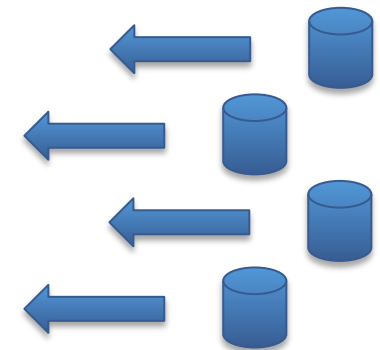
**A single-process,  
multithreaded  
Workload (v1.06)**

functional  
simulator  
(Pin)



Trace-driven simulation

**Multiple,  
single-threaded  
Workloads (v2.0)**



# TOP SNIPER FEATURES

- Interval Model
- CPI Stacks
- Parallel Multithreaded Simulator
- Based on Graphite infrastructure
- x86-64 and SSE2 support
- Validated against Core2, Nehalem
- Full DVFS support
- Shared and private caches
- Modern branch predictor
- Supports pthreads and OpenMP, TBB and OpenCL
- SimAPI and Python interfaces to the simulator
- Many flavors of Linux supported (Redhat, Ubuntu, etc.)



# SIMULATOR COMPARISON

	Sniper	Graphite	Gem5	COTSon	MARSSx86
Integrated			X		
Func-directed	X	X		X	X
User-level	X	X	X		
Full-system			X	X	X
Archs Supported	x64	x64	x64 Alpha SPARC	x64	x64
Parallel (in-node)	X	X			
Shared caches	X		X	X	X

# SNIPER LIMITATIONS

---

- User-level
  - Perfect for HPC
  - Not the best match for workloads with significant OS involvement
- Functional-directed
  - No simulation / cache accesses along false paths
- High-abstraction core model
  - Not suited to model all effects of core-level changes
  - Perfect for memory subsystem or NoC work
- x86-64 only



**ExaScience Lab**  
Intel Labs Europe



**EXASCALE COMPUTING**

# THE SNIPER MULTI-CORE SIMULATOR INTERVAL SIMULATION

TREVOR E. CARLSON, WIM HEIRMAN  
AND LIEVEN EECKHOUT



[HTTP://WWW.SNIPERSIM.ORG](http://www.snipersim.org)  
SUNDAY, APRIL 1<sup>ST</sup>, 2012  
ISPASS, NEW BRUNSWICK, NJ

# OVERVIEW

---

- Simulation Methodologies
  - Trace, Integrated, Functional-directed
- Core Models
  - One-IPC
  - Interval
- Interval Model and Simulation Detail
- CPI-Stacks

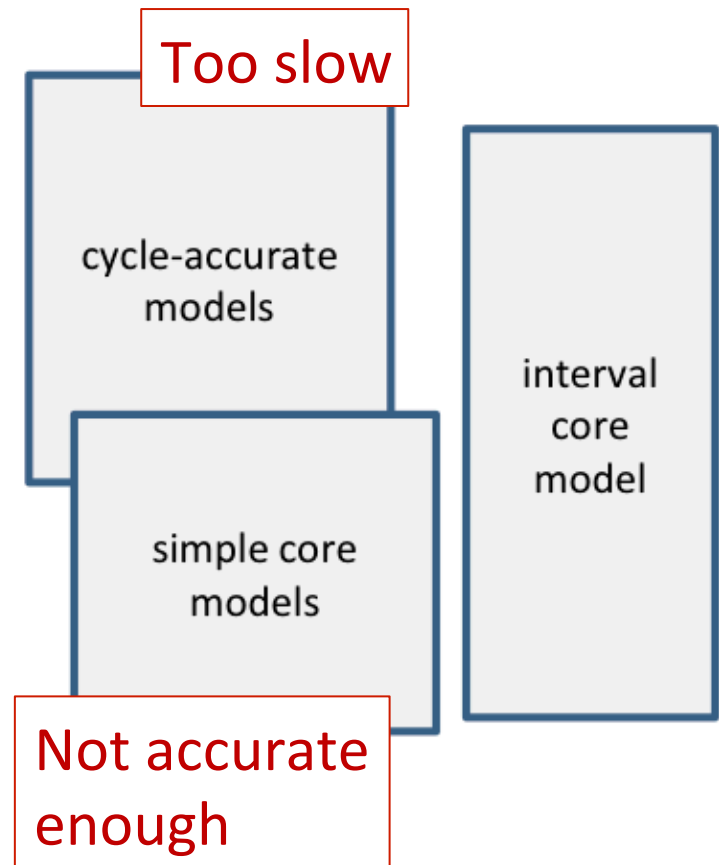
# SIMULATION METHODOLOGIES

---

- Trace-based Simulation
  - No wrong-path instructions nor timing-influenced results
  - Not the best for multithreaded applications
- Functional-First Simulation
  - The timing model controls wrong-path execution via checkpoints
  - Can be difficult to build
- Integrated Simulation
  - Timing and functional simulation are closely tied together
  - Timing of the core drives when instructions are fetched and executed
- Functional-Directed Simulation
  - Mispredicted path instructions are not taken into account
    - Rolling-back /check-pointing is therefore not needed
  - Timing model tends to be separate from the functional model

# NEEDED DETAIL DEPENDS ON FOCUS

Component	Single-event time scale	Required sim time
RTL	single clock cycle	millions of cycles
OOO execution		
Core memory ops		
L1 cache access		
LLC access		
Off-socket	microseconds	seconds





# ONE-IPC MODELING – TOO SIMPLE?

---

- Simple high-abstraction model
- Our definition of a One-IPC core model
  - Scalar, in-order issue
  - Account for non-unit instruction exec latencies
  - Perfect branch prediction
  - L1 D-cache hits are completely hidden
  - All other cache accesses incur penalty

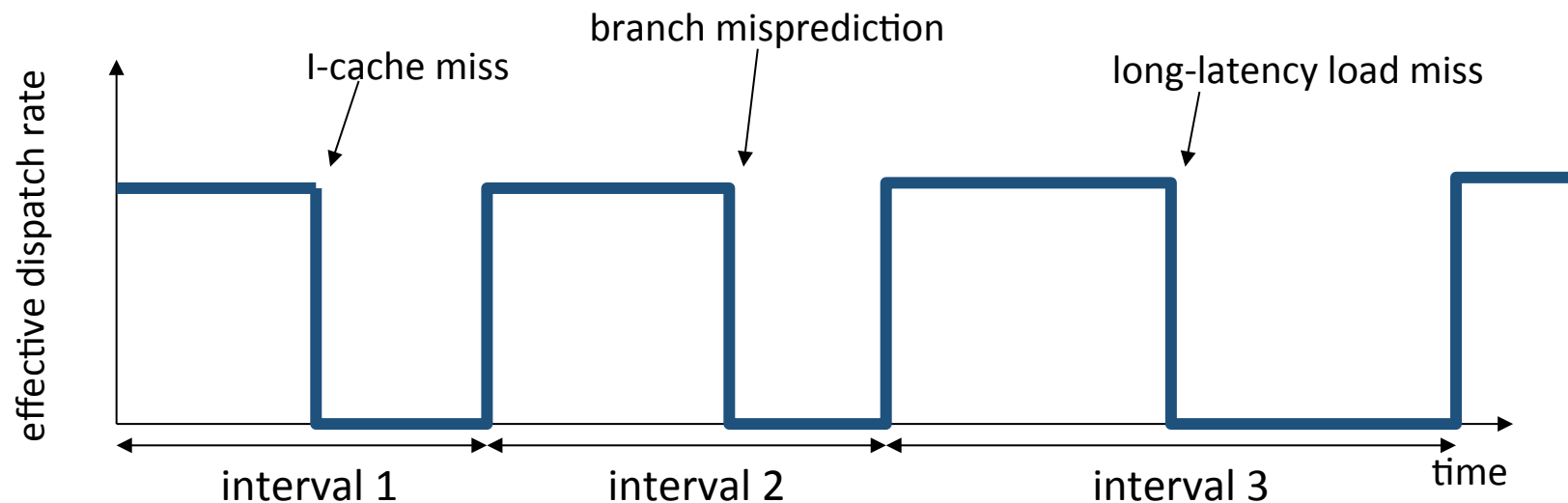
# ONE-IPC CORE MODEL

---

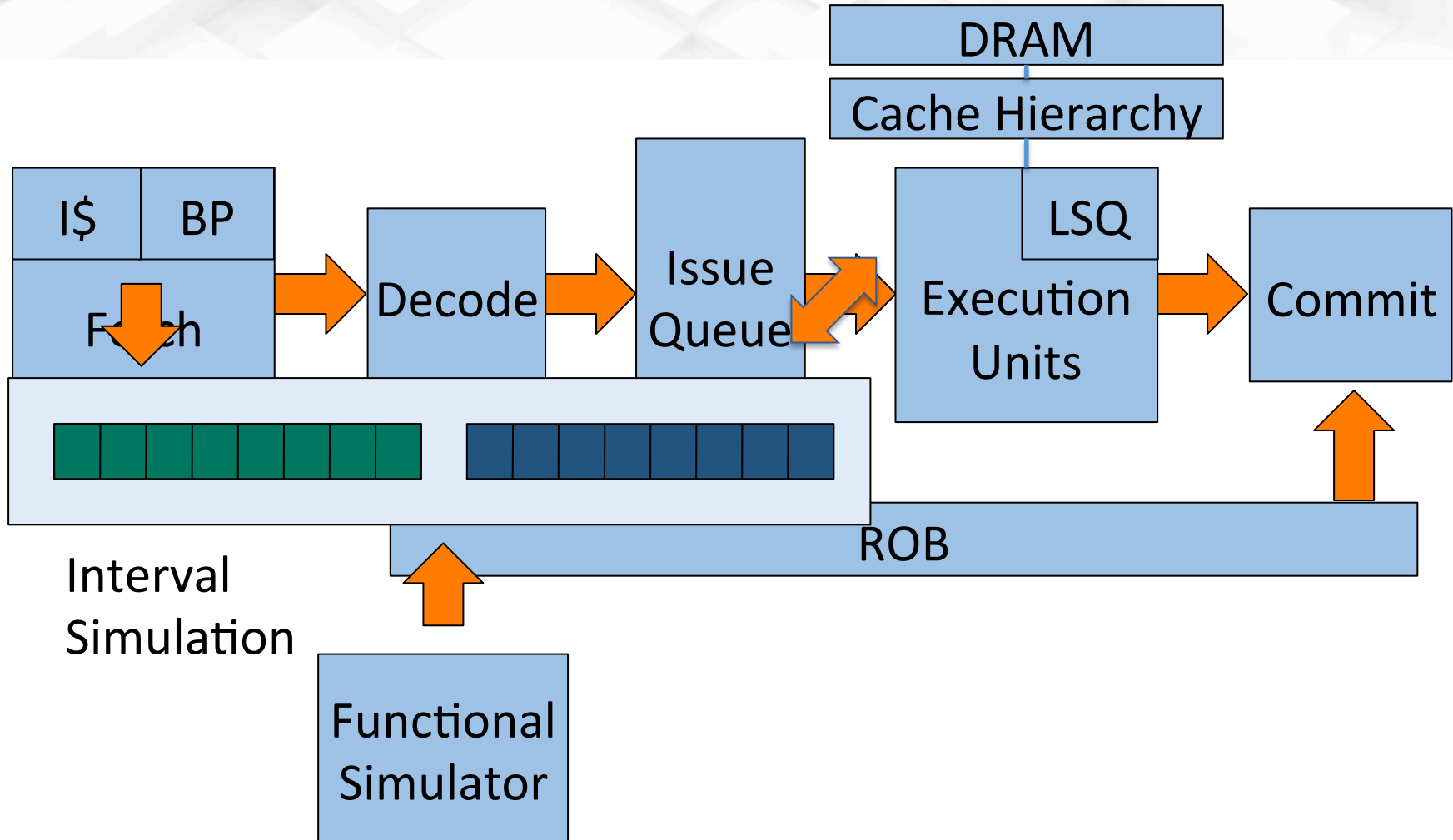
- Alternative for memory access traces
  - Aims to provide more-realistic access patterns
  - Allows for timing feedback
- Nevertheless, One-IPC core models do not exhibit MLP
  - Therefore, request rates are not as accurate as cycle-level simulators

# INTERVAL MODEL

- Out-of-order core performance model with in-order simulation speed



# DETAILED MODEL VS. INTERVAL SIM

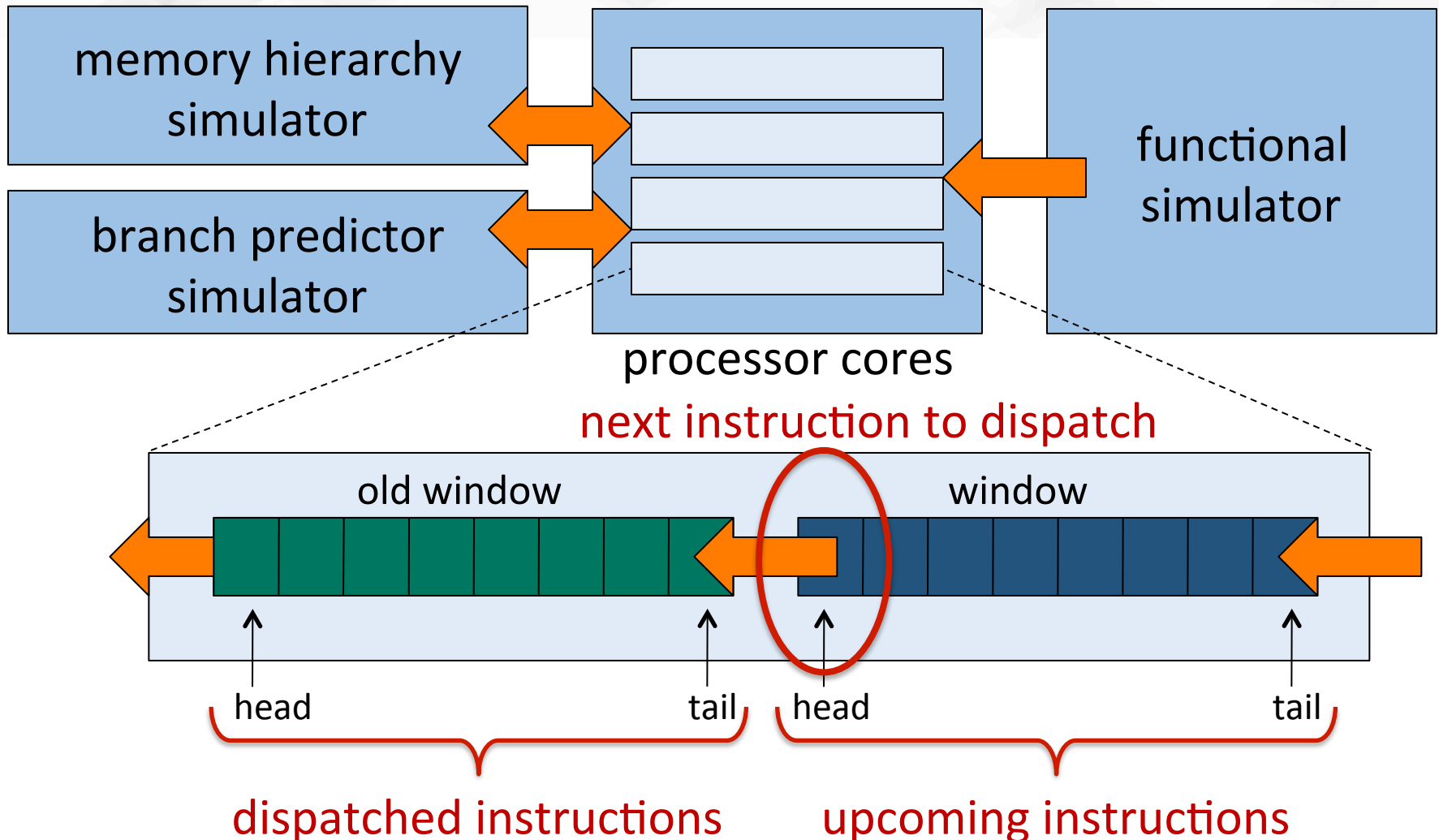


# KEY BENEFITS OF THE INTERVAL MODEL

---

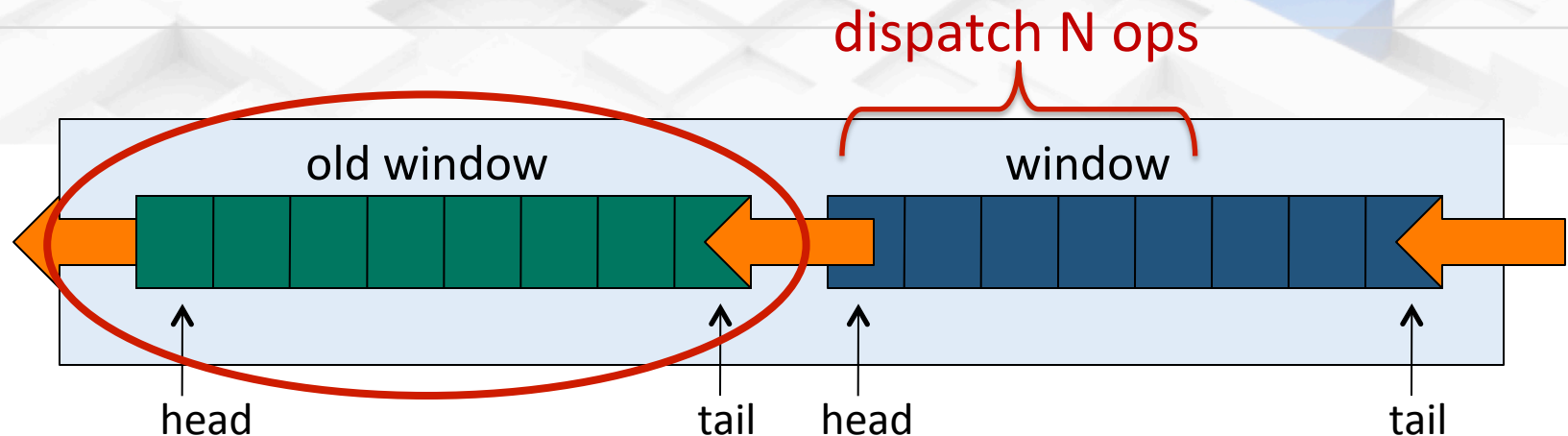
- Models superscalar OOO execution
- Models impact of ILP
- Models second-order effects: MLP
- Allows for constructing CPI stacks

# MULTI-CORE INTERVAL SIMULATION



# CORE-LEVEL TIMING

*NO MISS EVENTS*



Instantaneous dispatch rate is determined by the longest critical path in the old window:

$$\text{Instantaneous dispatch rate} = \min ( W / L, D )$$

Little's law

Assumes a balanced architecture

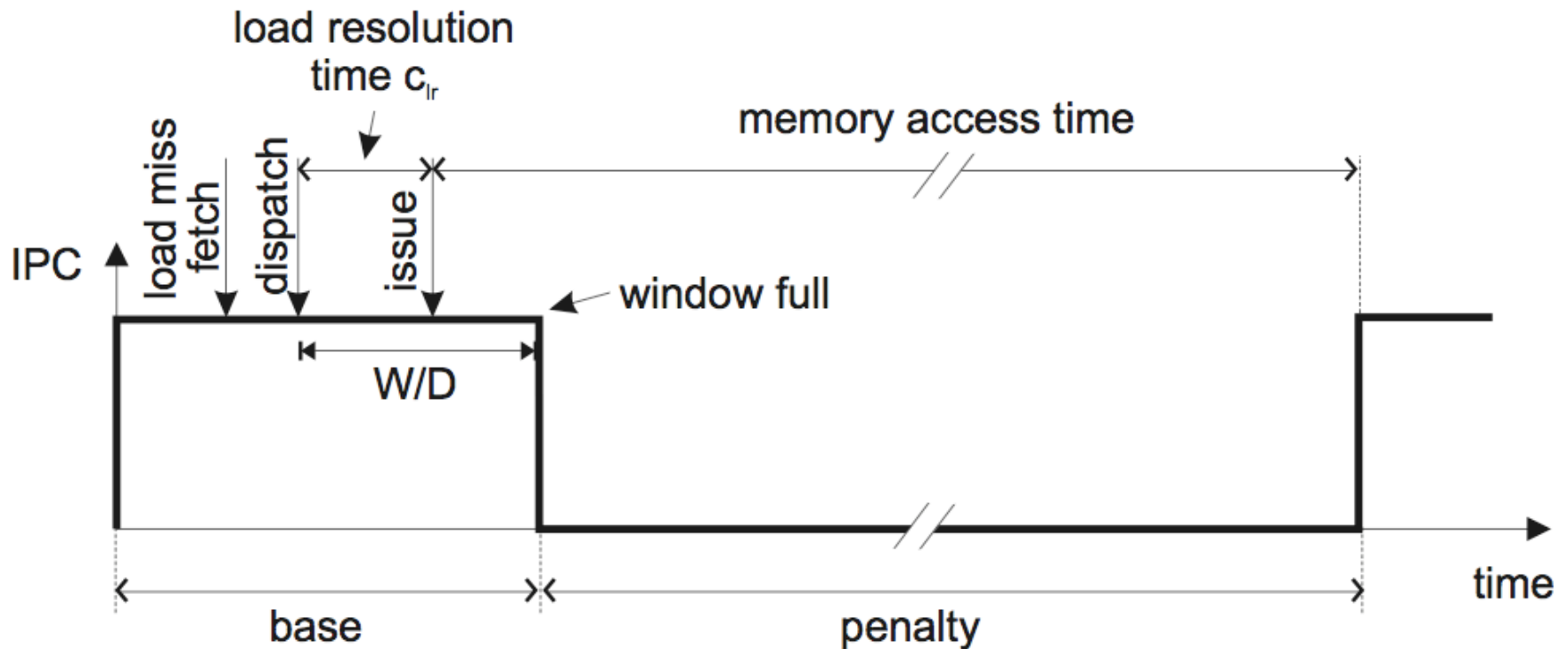
L = longest critical path length in cycles

W = instructions in the old window (max = ROB length)

D = maximum dispatch rate (processor width)

# LONG BACK-END MISS EVENTS

## *ISOLATED LONG-LATENCY LOAD*

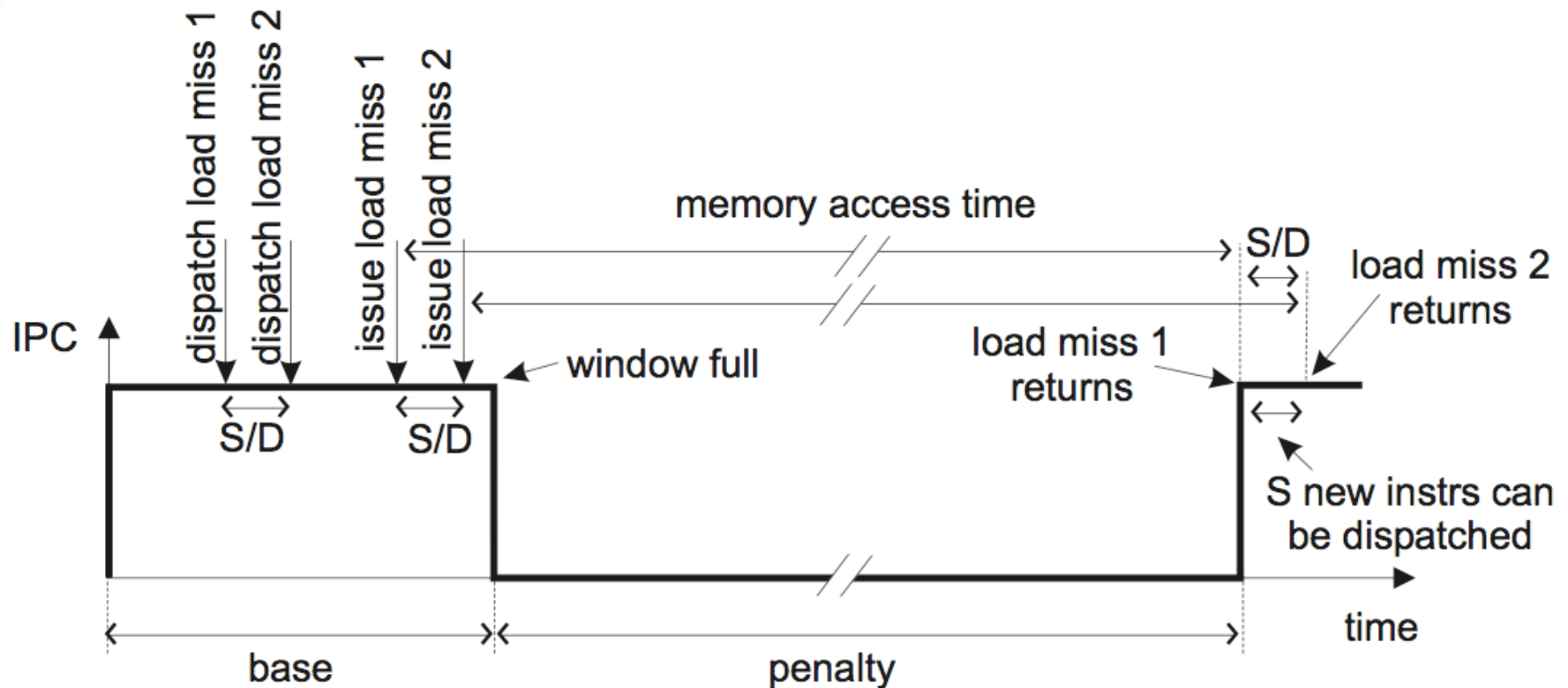


*S. Eyerman et al., ACM TOCS, May 2009*



# LONG BACK-END MISS EVENTS

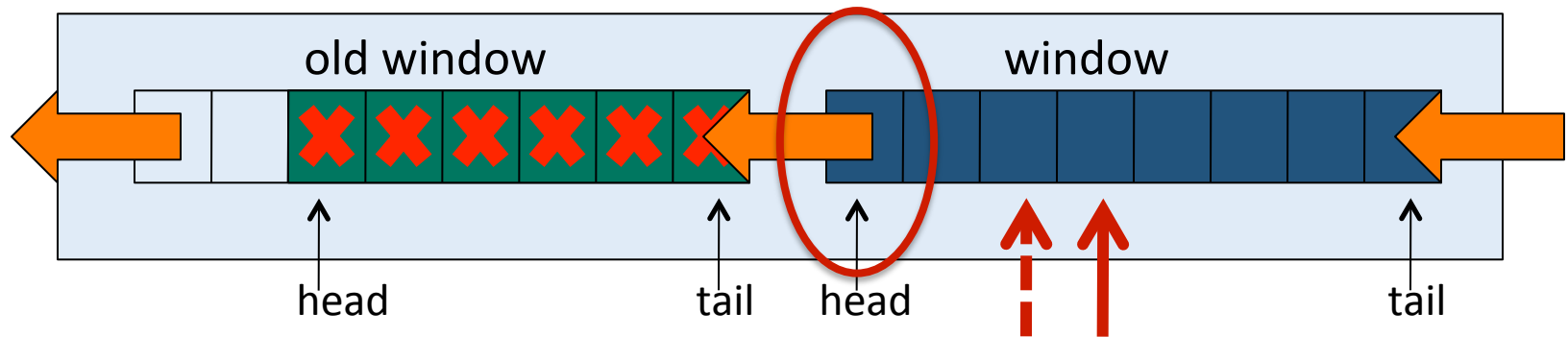
## *OVERLAPPING LONG-LATENCY LOADS*



*S. Eyerman et al., ACM TOCS, May 2009*

# CORE-LEVEL TIMING

## LONG-LATENCY LOAD



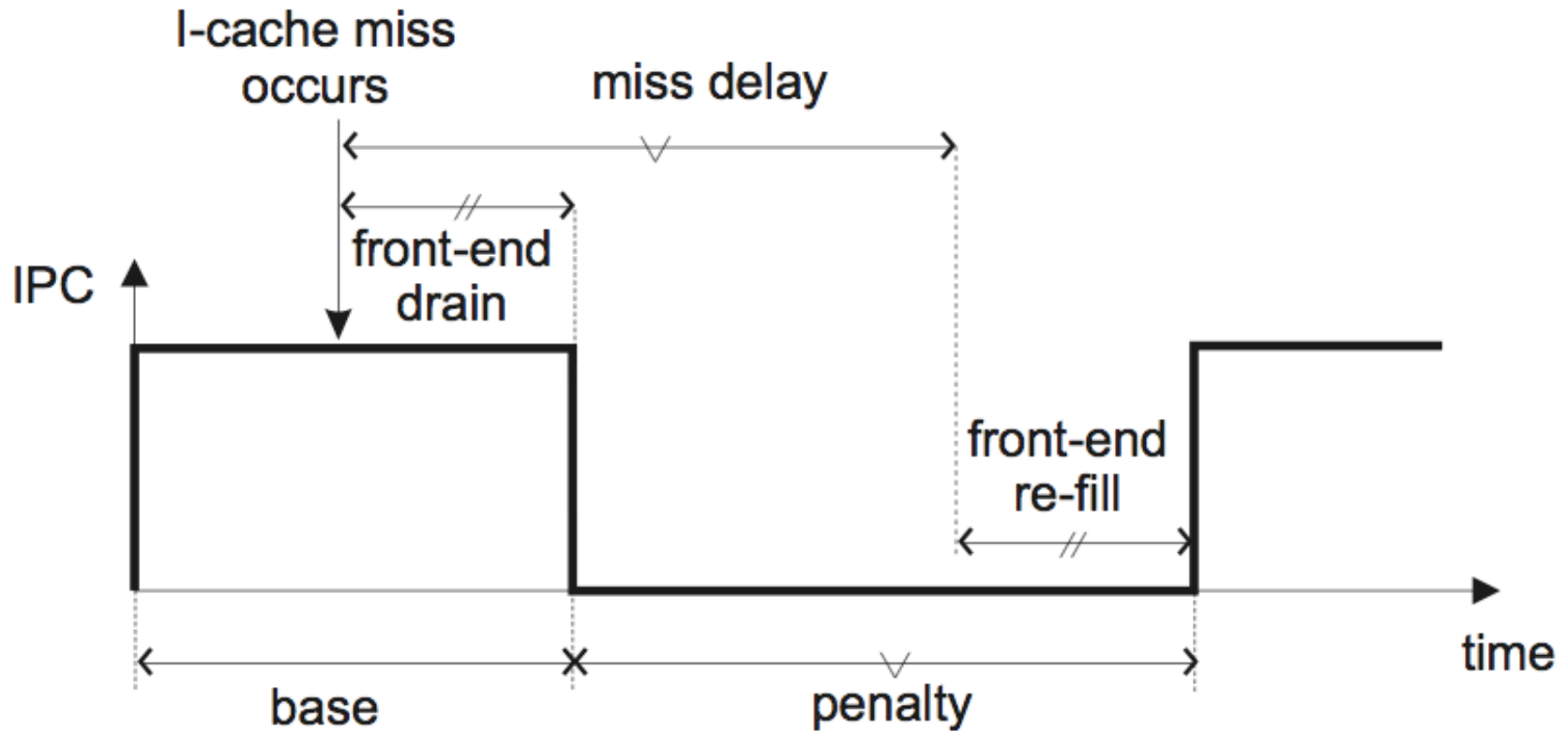
If long-latency load (LLC miss):

core sim time += miss latency

*AND* walk the window to issue independent miss events: these are hidden under the long-latency load  
– second-order effects

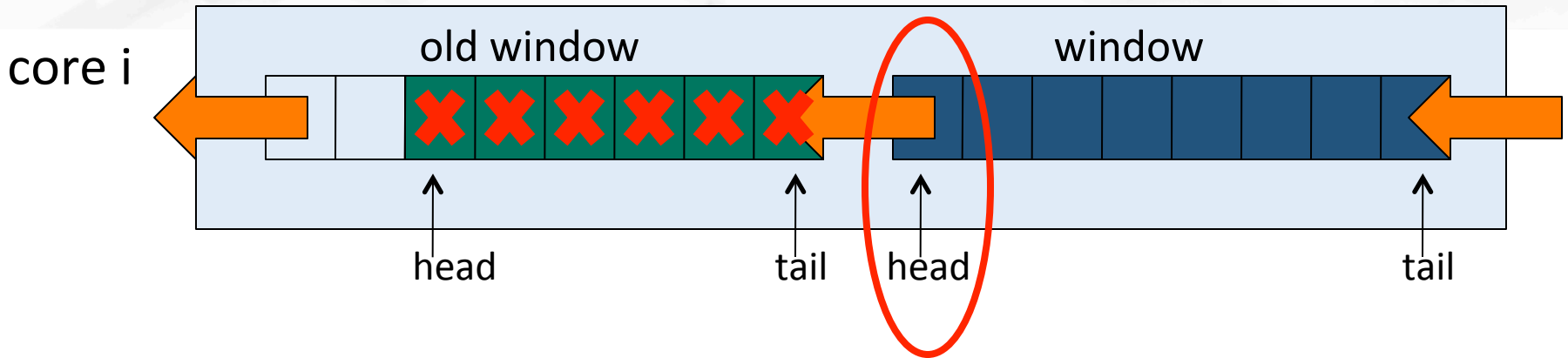
*AND* empty old window

# I-CACHE MISS (L1, L2, TLB)



*S. Eyerma et al., ACM TOCS, May 2009*

# CORE-LEVEL TIMING: I-CACHE/TLB

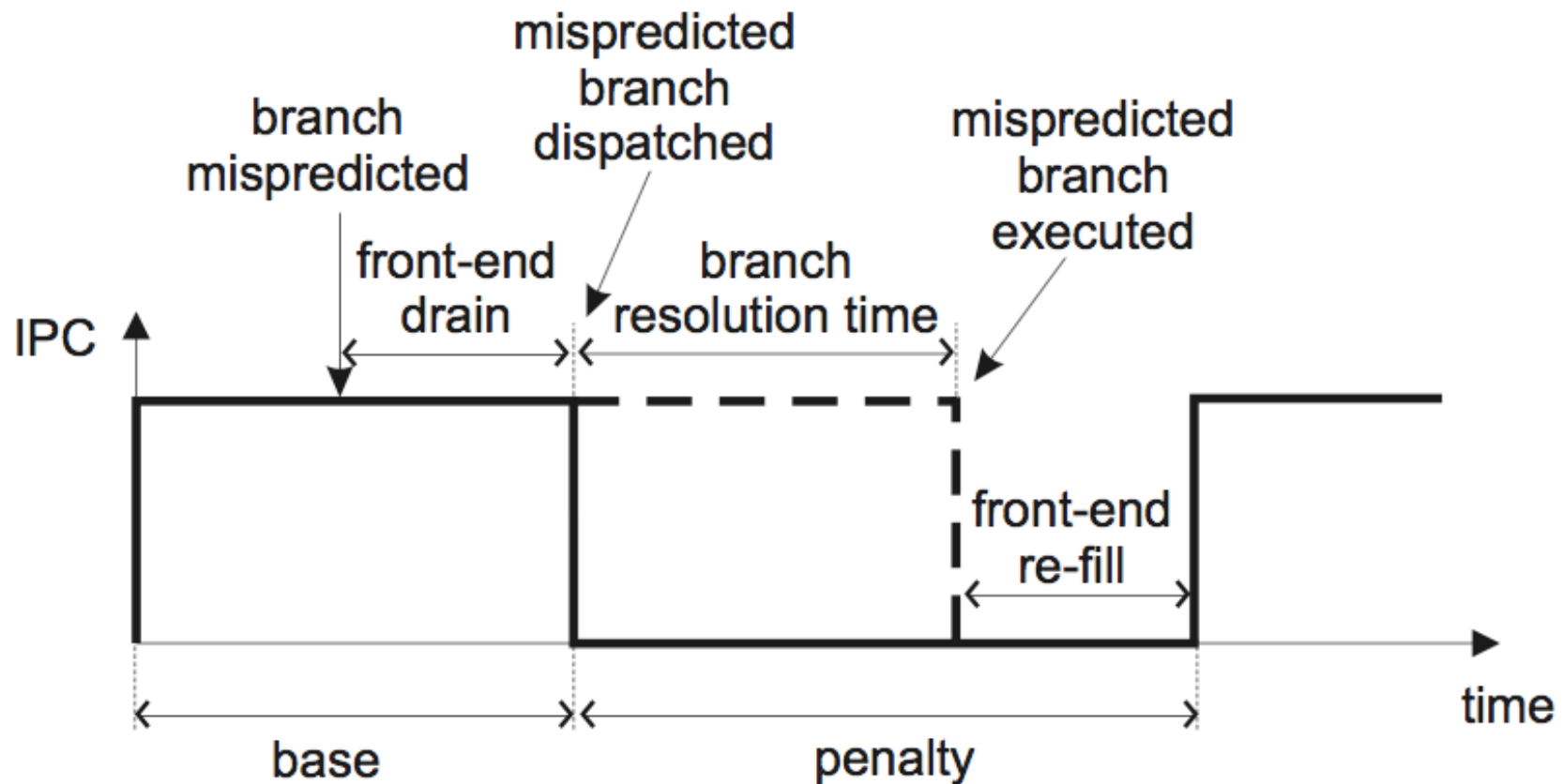


If I-cache or I-TLB miss:

core sim time += miss latency

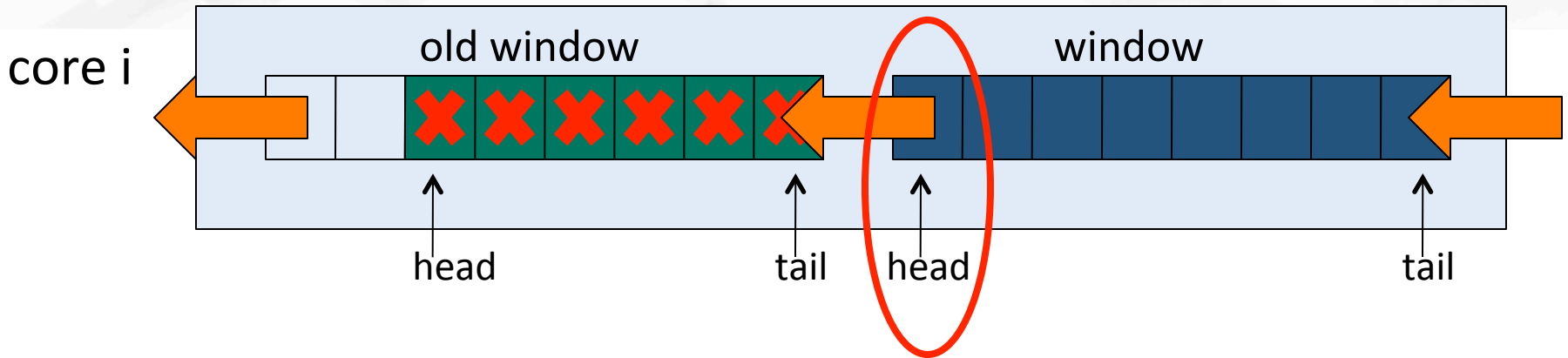
*AND* empty old window

# BRANCH MISPREDICTION



*S. Eyerma et al., ACM TOCS, May 2009*

# CORE-LEVEL TIMING: BRANCH MISPREDICT

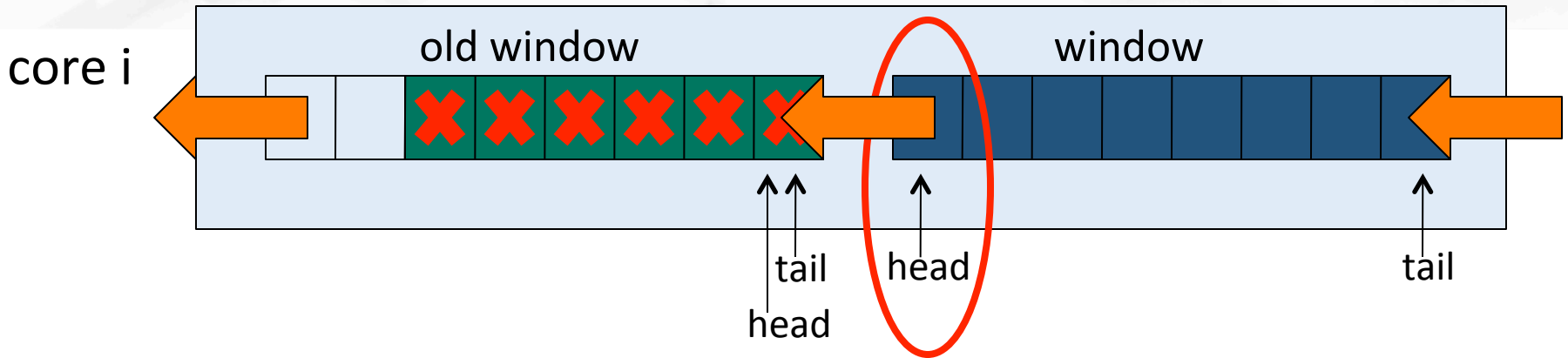


If branch misprediction:

core sim time += branch resolution time  
+ front-end pipeline depth

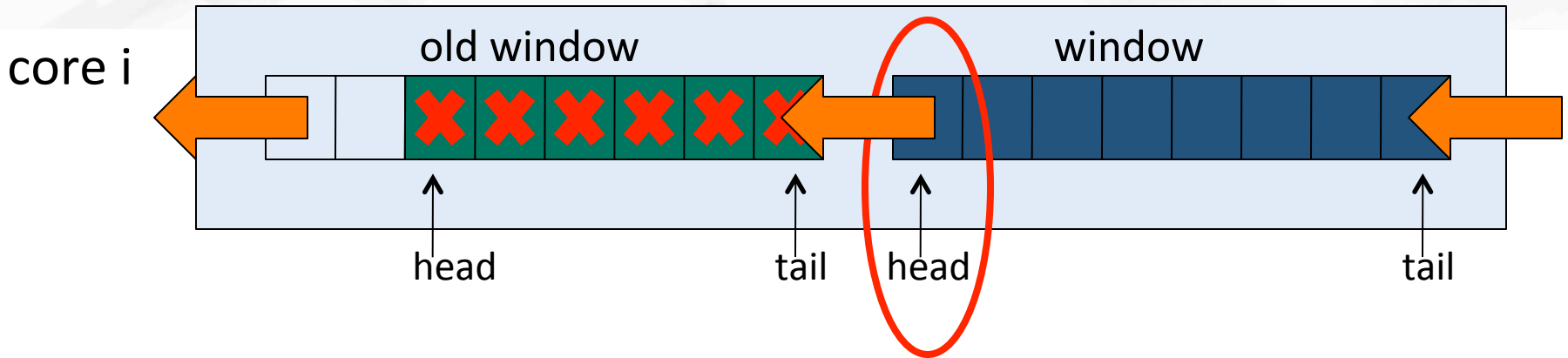
*AND* empty old window

# CORE-LEVEL TIMING: BRANCH MISPREDICT



Branch resolution time = longest critical path in 'old window' leading to the branch

# CORE-LEVEL TIMING: SERIALIZING INSN



If serializing instruction:

core sim time += window drain time

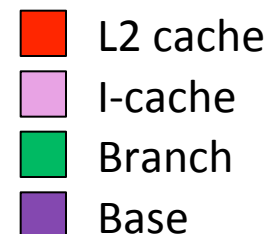
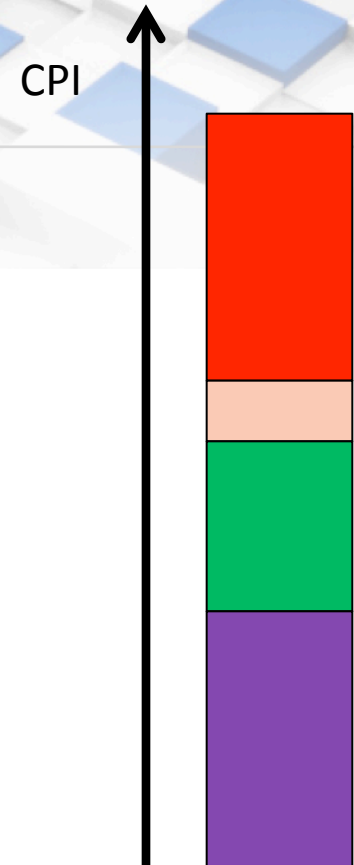
window drain time =  $\max ( W / D , L )$

*AND* empty the old window



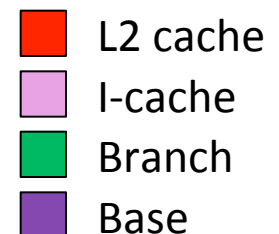
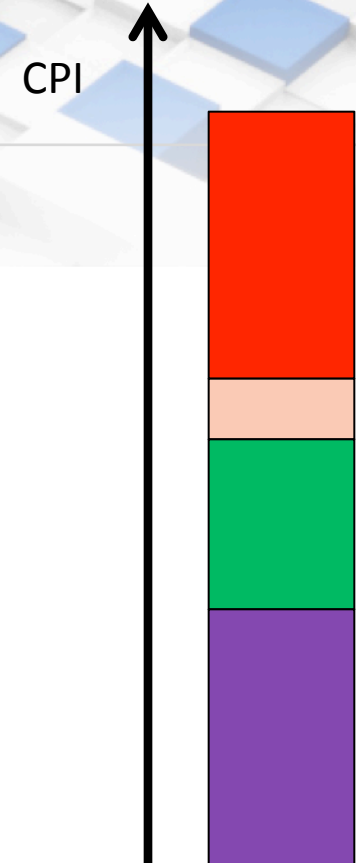
# CYCLE STACKS

- Where did my cycles go?
- CPI stack
  - Cycles per instruction
  - Broken up in components
- Normalize by either
  - Number of instructions (CPI stack)
  - Execution time (time stack)
- Different from miss rates:  
cycle stacks directly quantify  
the effect on performance



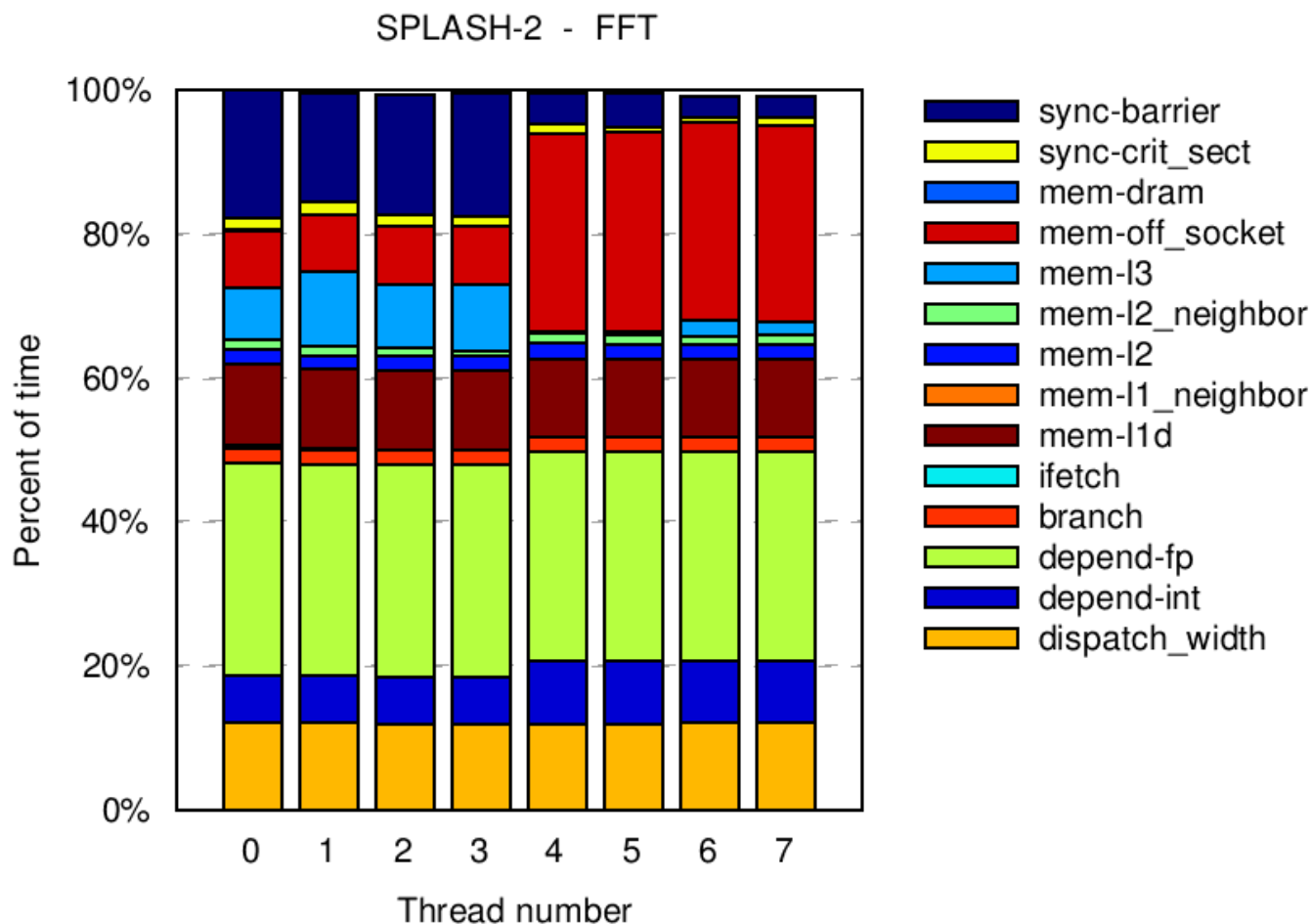
# CONSTRUCTING CPI STACKS

- Interval simulation:  
track why time is advanced
  - No miss events
    - Issue instructions at base CPI
    - Increment base component
  - Miss event
    - Fast-forward time by X cycles
    - Increment component by X

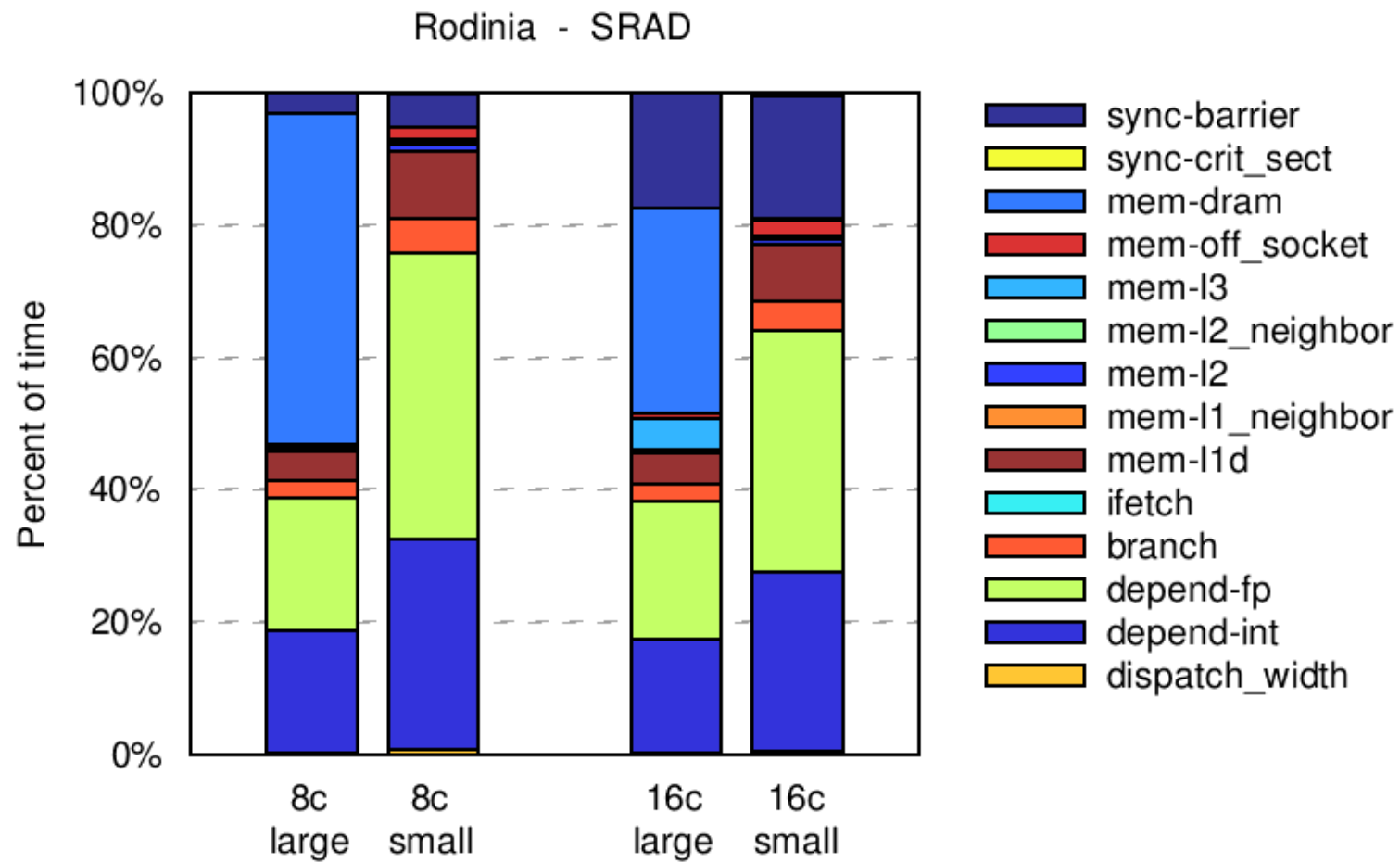


# CYCLE STACKS FOR PARALLEL APPLICATIONS

By thread: heterogeneous behavior  
in a homogeneous application?

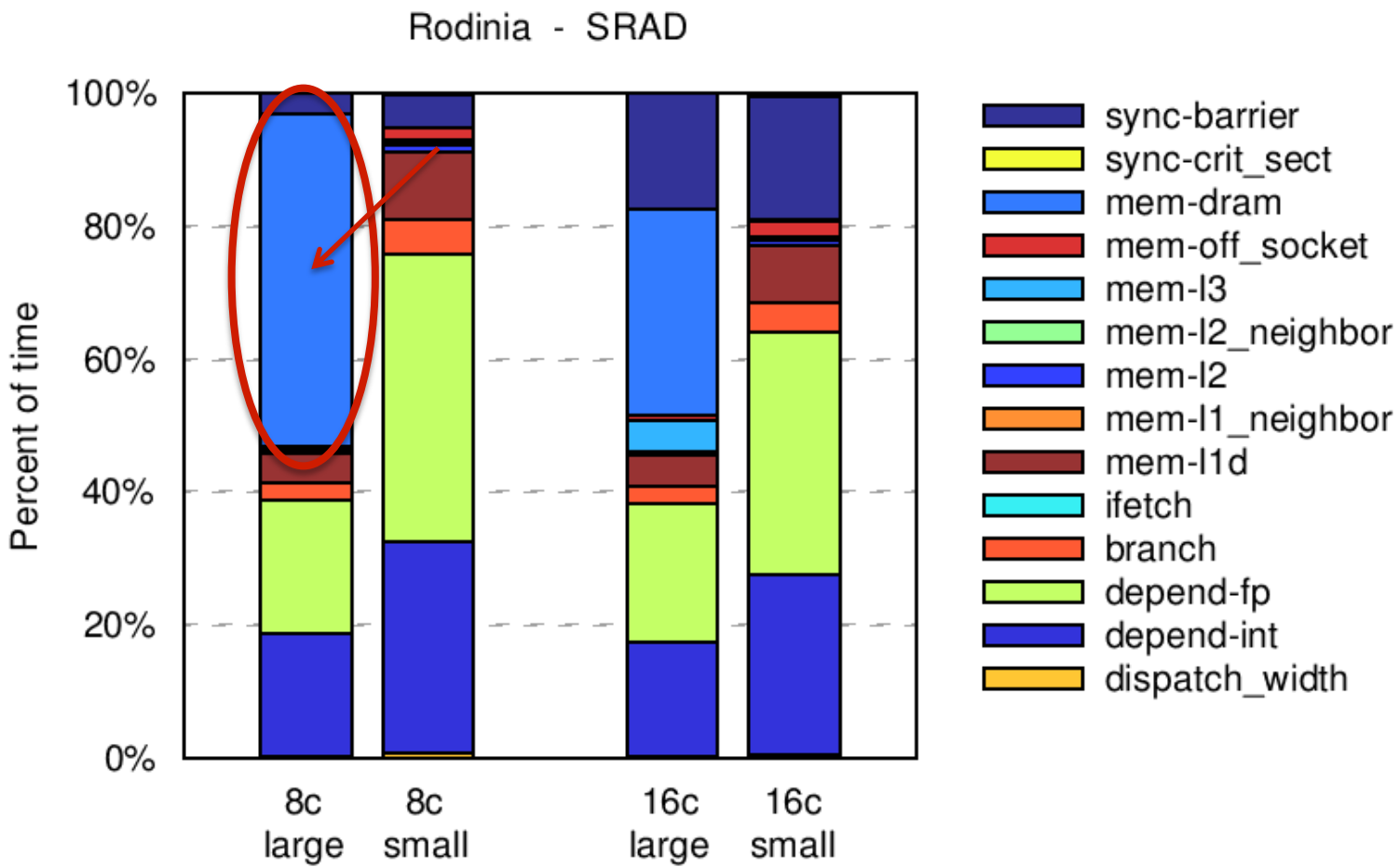


# USING CYCLE STACKS TO EXPLAIN SCALING BEHAVIOR



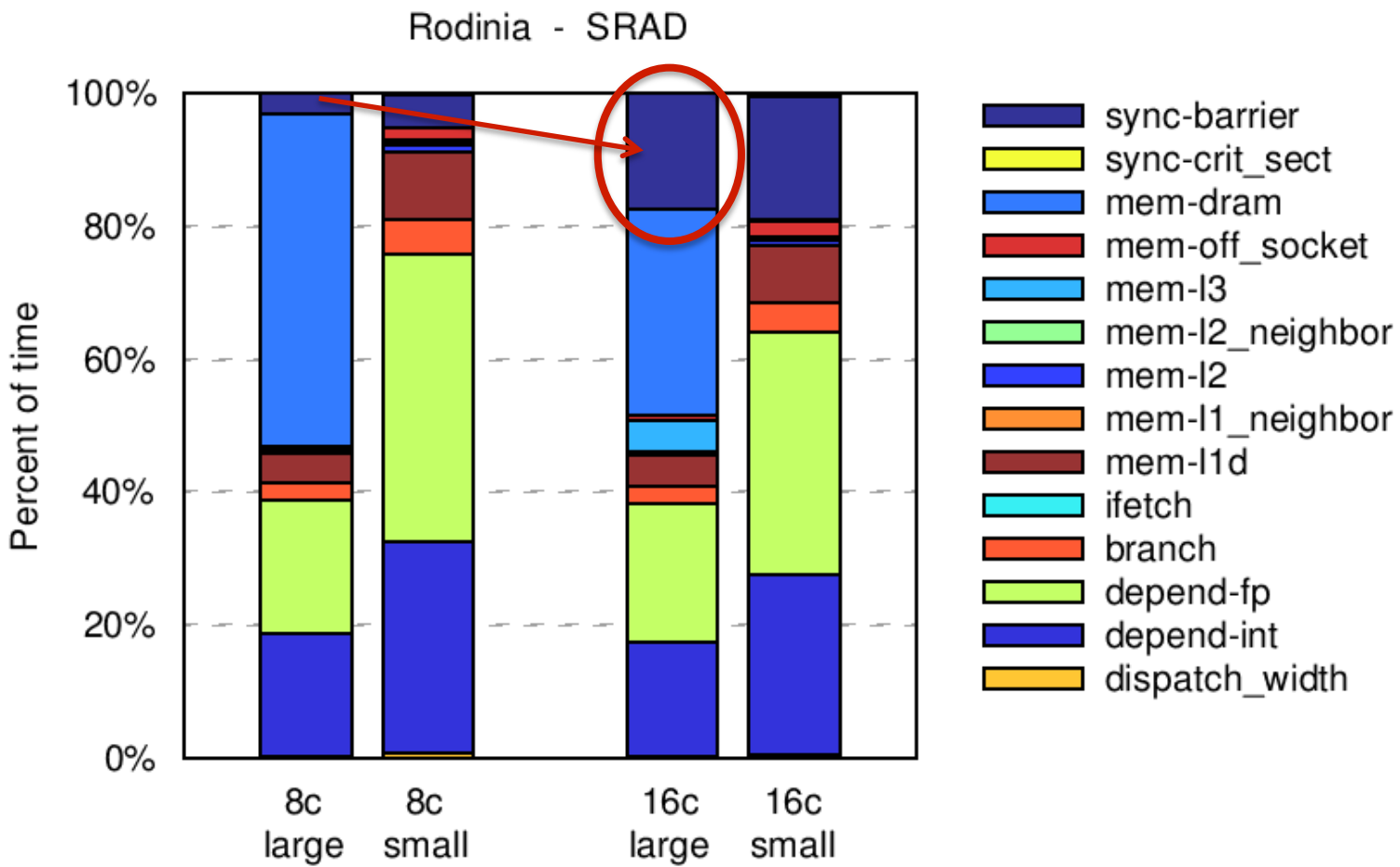
# USING CYCLE STACKS TO EXPLAIN SCALING BEHAVIOR

- Scale input: application becomes DRAM bound



# USING CYCLE STACKS TO EXPLAIN SCALING BEHAVIOR

- Scale input: application becomes DRAM bound
- Scale core count: sync losses increase to 20%





**ExaScience Lab**  
Intel Labs Europe

**EXASCALE COMPUTING**

# THE SNIPER MULTI-CORE SIMULATOR SIMULATOR INTERNALS

WIM HEIRMAN, TREVOR E. CARLSON  
AND LIEVEN EECKHOUT



[HTTP://WWW.SNIPERSIM.ORG](http://www.snipersim.org)  
SUNDAY, APRIL 1<sup>ST</sup>, 2012  
ISPASS, NEW BRUNSWICK, NJ

# OVERVIEW

---

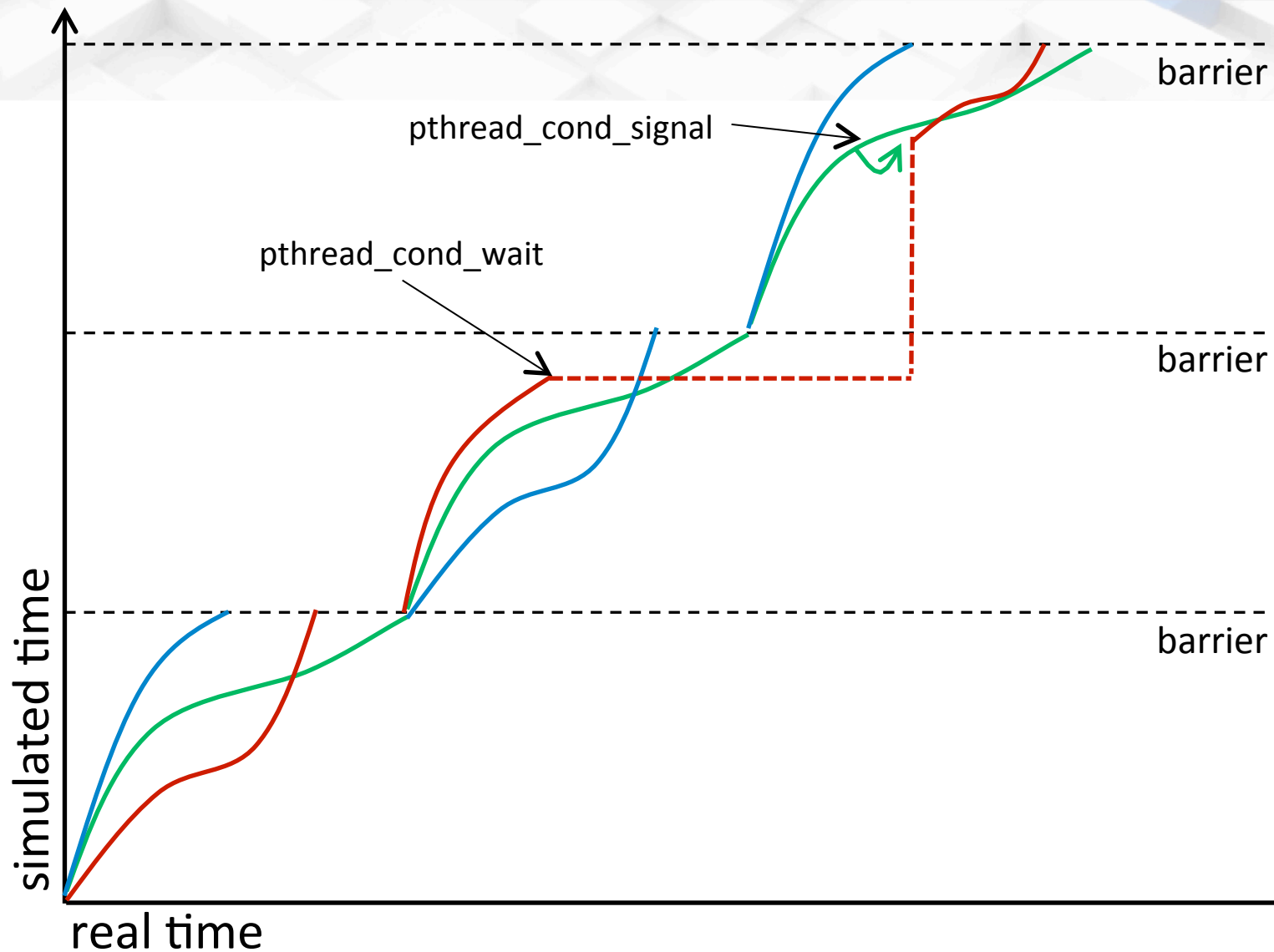
- Parallel simulation with relaxed synchronization
  - Flexible synchronization schemes between cores
  - Trade off causality errors for simulation speed
- Parallelism inside Sniper
- Hardware components



# RELAXED SYNCHRONIZATION

- Graphite introduced relaxed synchronization with a number of different synchronization schemes
  - none: only synchronizes when the application does; for pthread calls, etc.
  - random-pairs: synchronizes random pairs of threads
  - barrier: synchronizes all threads at a given simulated time interval
- Sniper defaults to barrier synchronization with 100ns intervals
  - Multi-machine mode not supported, so tight synchronization is easier

# BARRIER SYNCHRONIZATION IN ACTION



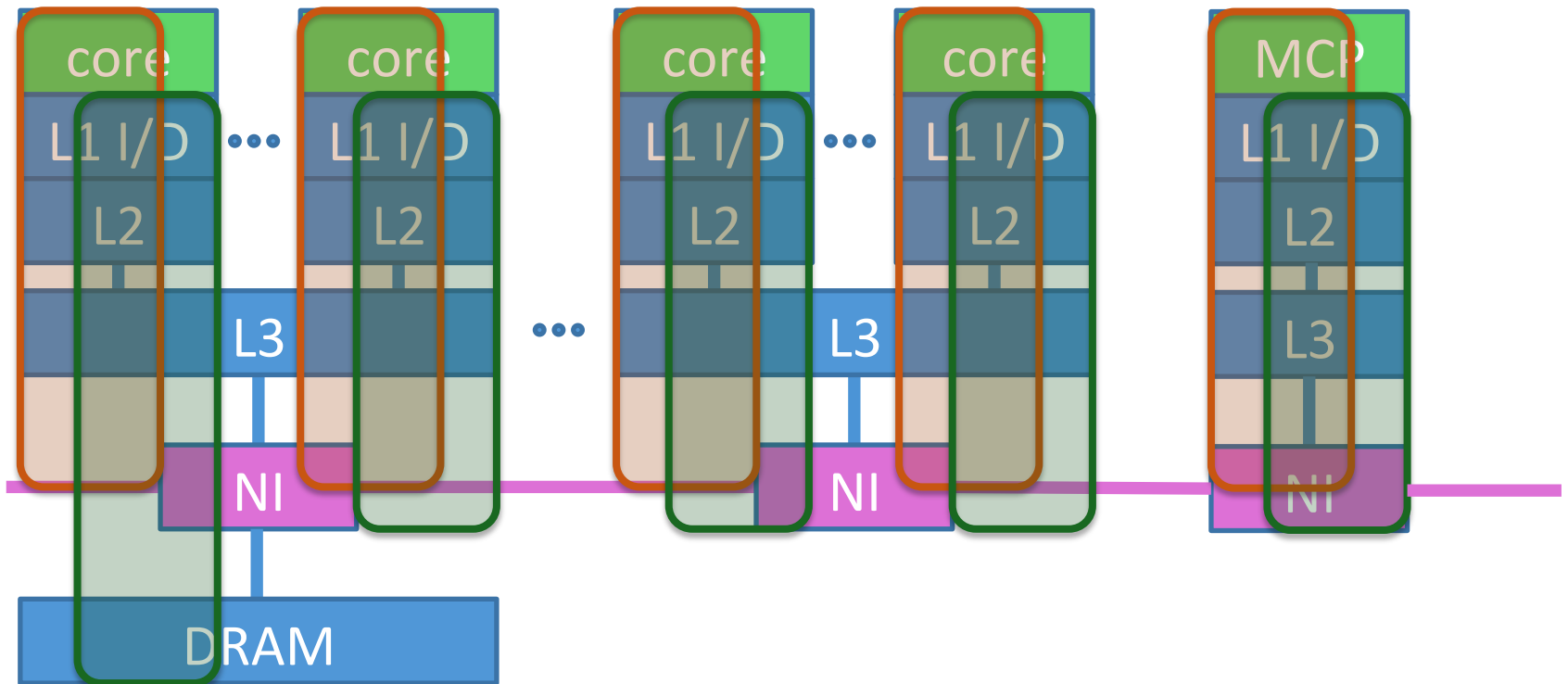
# PARALLELISM INSIDE SNIPER

- Each simulated core is run inside its own thread
  - Includes functional simulation, timing models for core and cache
  - Each core model maintains its own local time
- Extra threads for network and DRAM models
  - Can process invalidation requests without interrupting the core model
- Each thread is allowed to independently make progress
  - Causality errors can occur, no rollback
  - Skew is limited to 100ns

# THREADS IN SNIPER

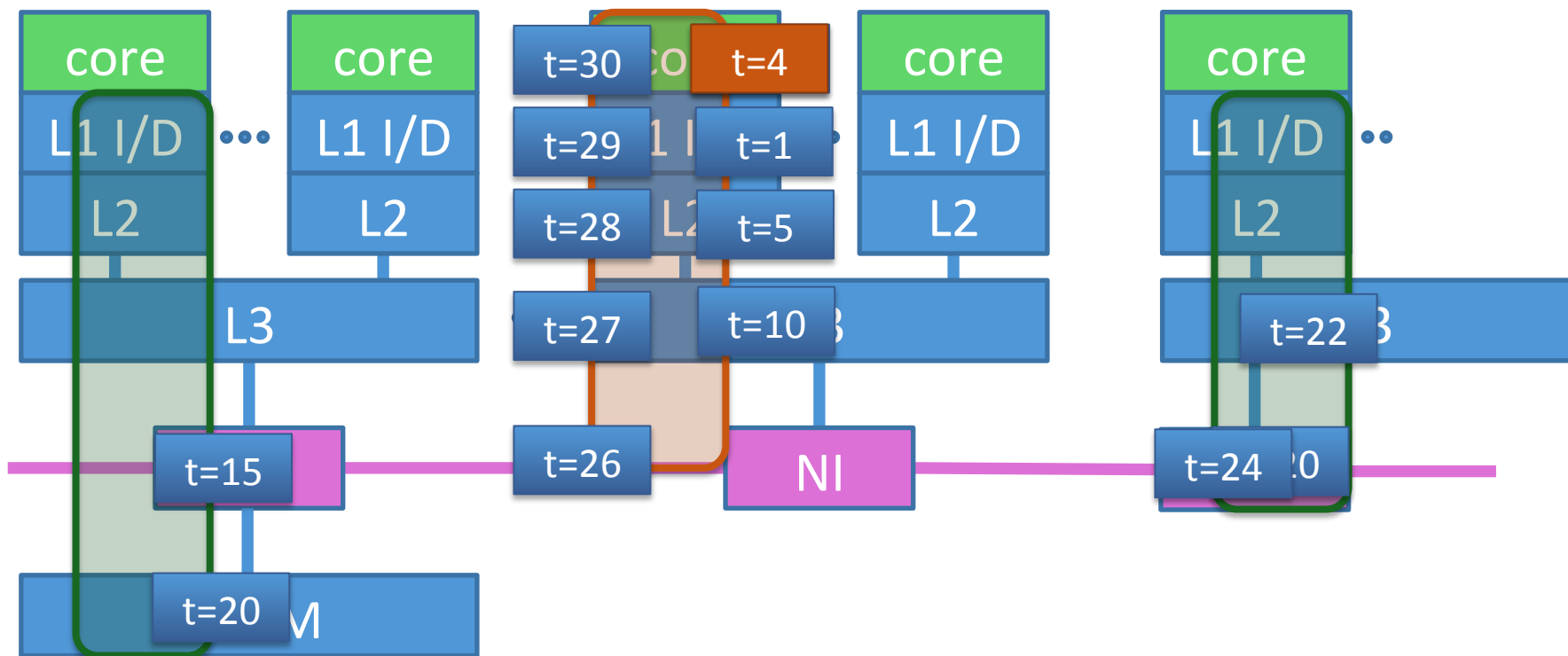
application threads

network threads



# TIME IN SNIPER

- Each memory access instantly returns latency
- Application threads maintain time
- Network threads reset time for each request



# MODELING CONTENTION

- Events may happen out of order
- How to model bandwidth / contention?
  - History list
    - Resource in use at times 0...10, 12...17, 25...30
    - Access at 15: delay = 2
    - Access at 8, length 5: ?
- Causality errors are possible
  - Effect is limited, as long as average bandwidth is OK
  - Allows for faster simulation, easier implementation
  - Speed versus accuracy trade-off

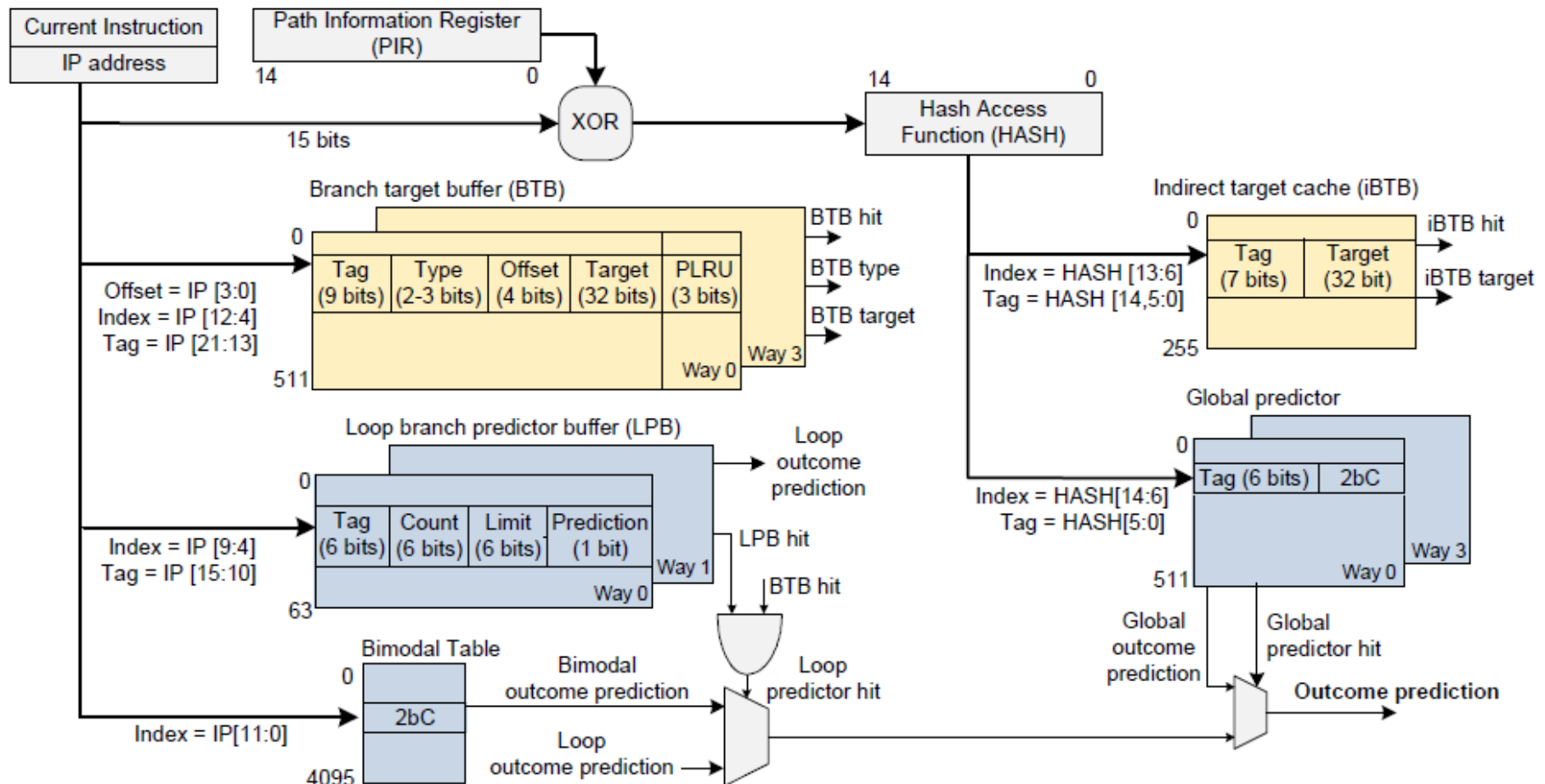
# CONFIGURABLE COMPONENTS

---

- Hardware options
  - Branch predictors
  - Cache hierarchies
- Core options
  - Core models: interval, one-IPC, Graphite legacy
  - DVFS
- Networks

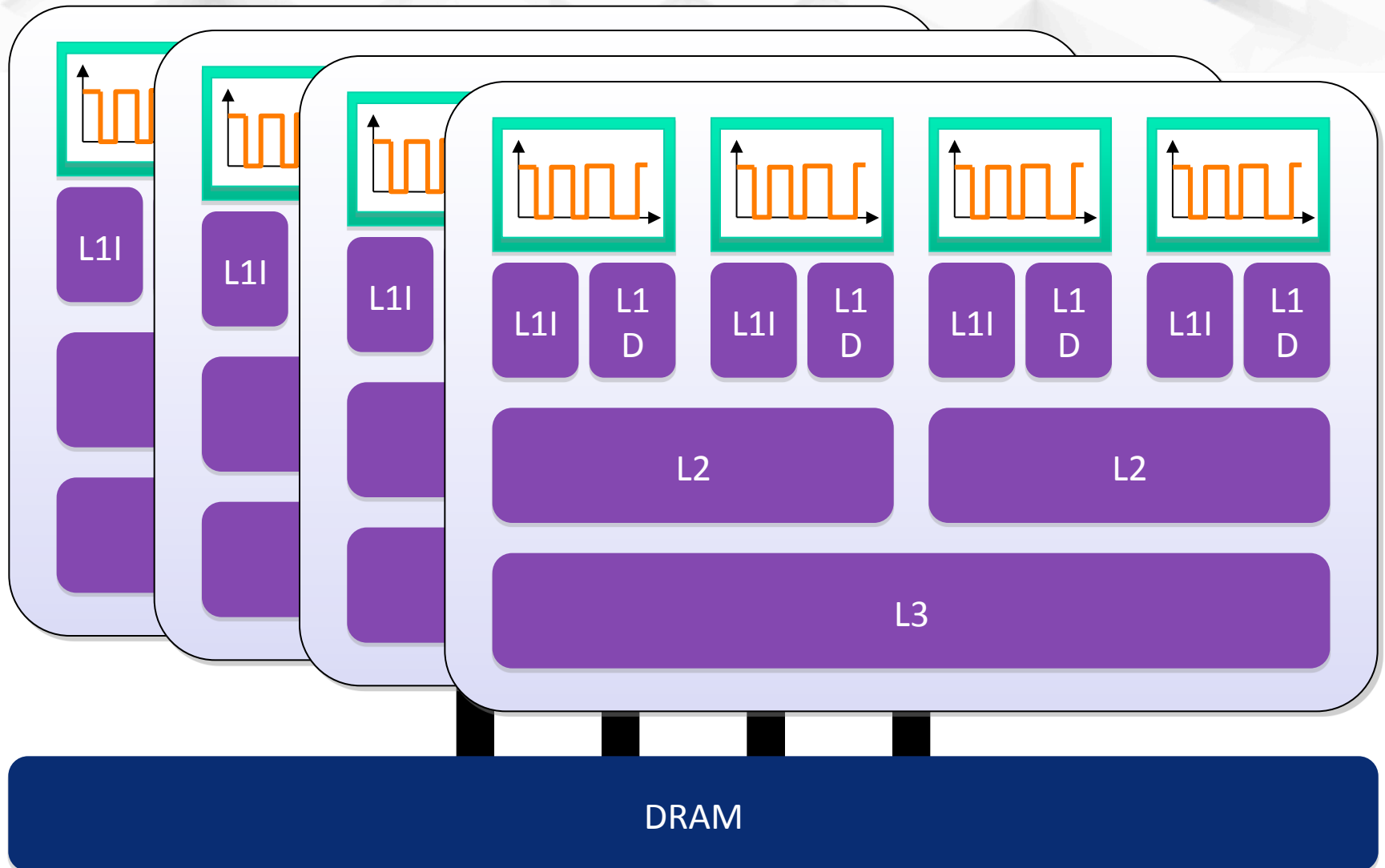
# BRANCH PREDICTOR

- Pentium-M-style branch predictor V. Uzelac, ISPASS'09





# PARAMETRIC SHARED CACHE HIERARCHY





**ExaScience Lab**  
Intel Labs Europe

**EXASCALE COMPUTING**

# **THE SNIPER MULTI-CORE SIMULATOR SIMULATOR ACCURACY AND HARDWARE VALIDATION**

TREVOR E. CARLSON, WIM HEIRMAN  
AND LIEVEN EECKHOUT



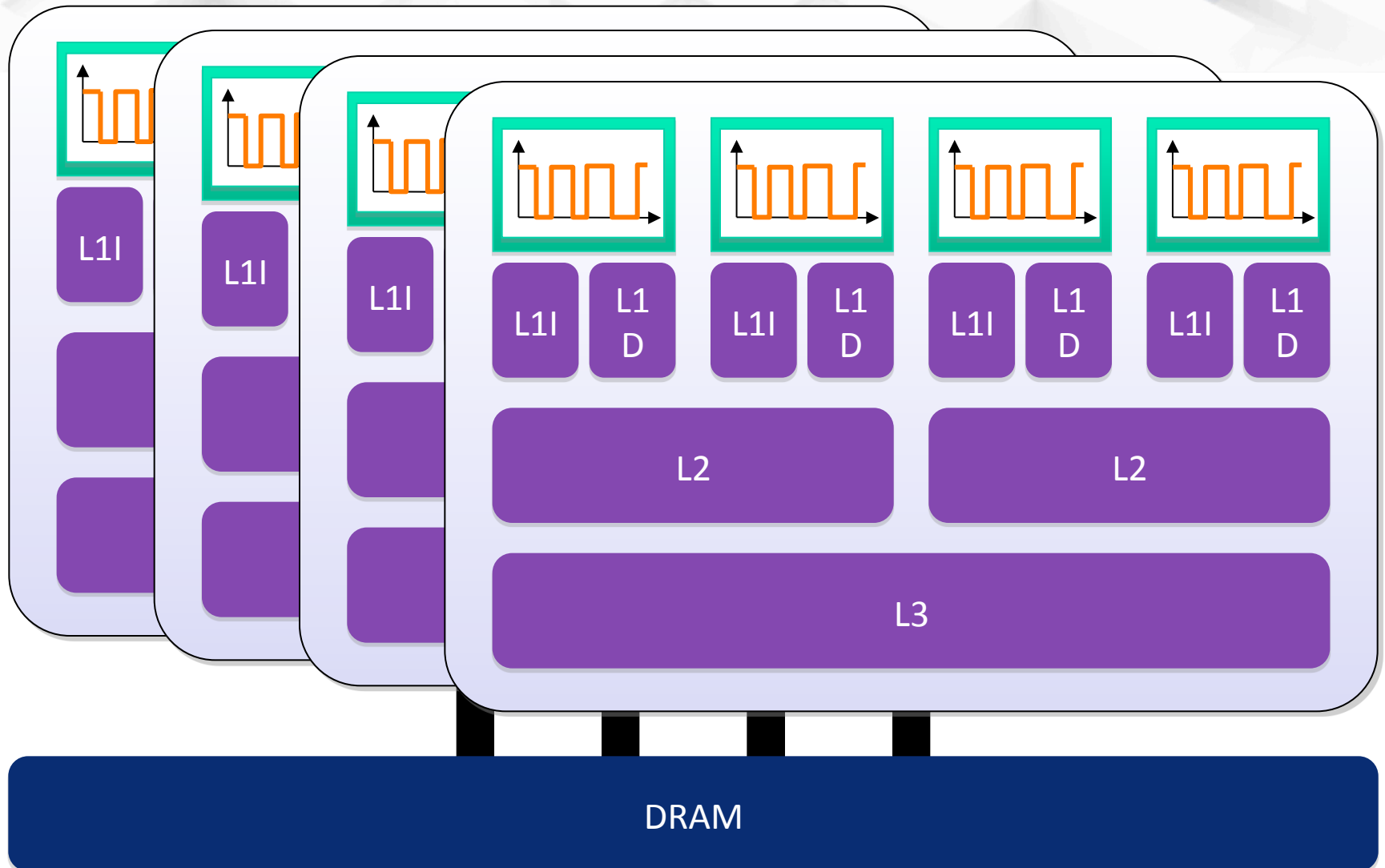
[HTTP://WWW.SNIPERSIM.ORG](http://www.snipersim.org)  
SUNDAY, APRIL 1<sup>ST</sup>, 2012  
ISPASS, NEW BRUNSWICK, NJ

# EXPERIMENTAL SETUP

---

- **Benchmarks**
  - Complete SPLASH-2 suite
    - 1 to 16 threads
    - Linux pthreads API
  - Extensive use of microbenchmarks to tune parameters and track down problems
- **Hardware**
  - Four-socket Intel Xeon X7460 machine
  - Core2 (45nm, Penryn) with 6 cores/socket

# EXPERIMENTAL SETUP: ARCHITECTURE



# HINTS FOR COMPARING TO HARDWARE

- Threads are pinned to their own core

```
pthread_setaffinity_np()
```

- Steepstep is disabled

```
echo performance > /sys/devices/system/cpu/*/cpufreq/  
scaling_governor
```

- Turbo mode, Hyperthreading disabled

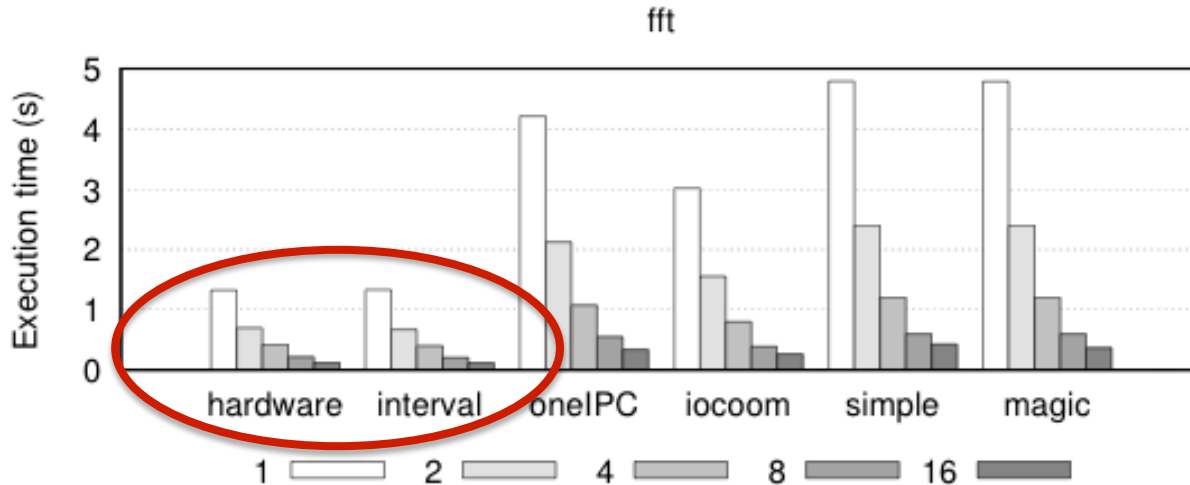
- BIOS setting

- Use hardware performance counters

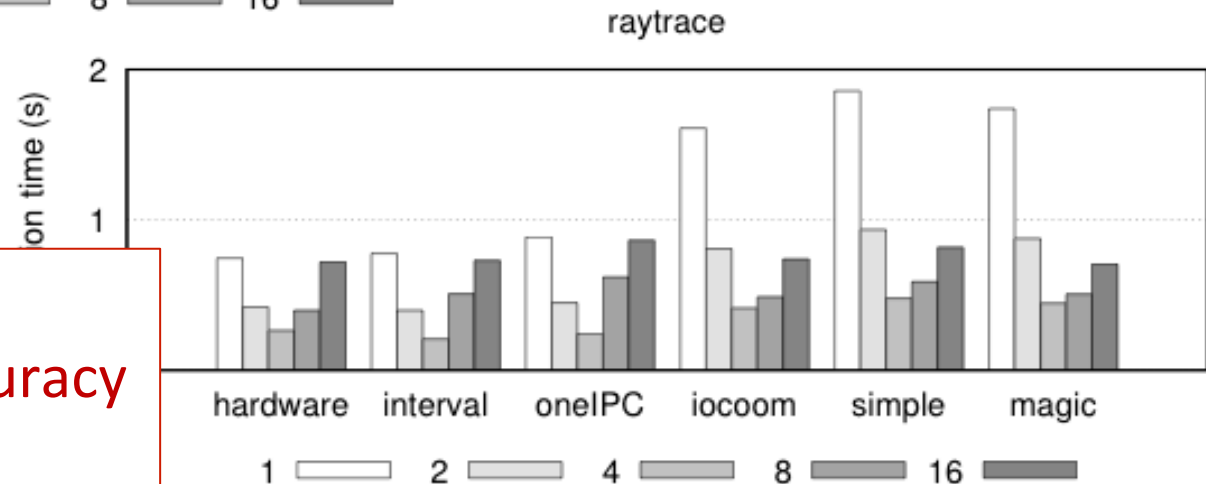
- But can be difficult to interpret

- Overlapping cache misses (HW) vs. hits (Sniper)

# INTERVAL PROVIDES NEEDED ACCURACY

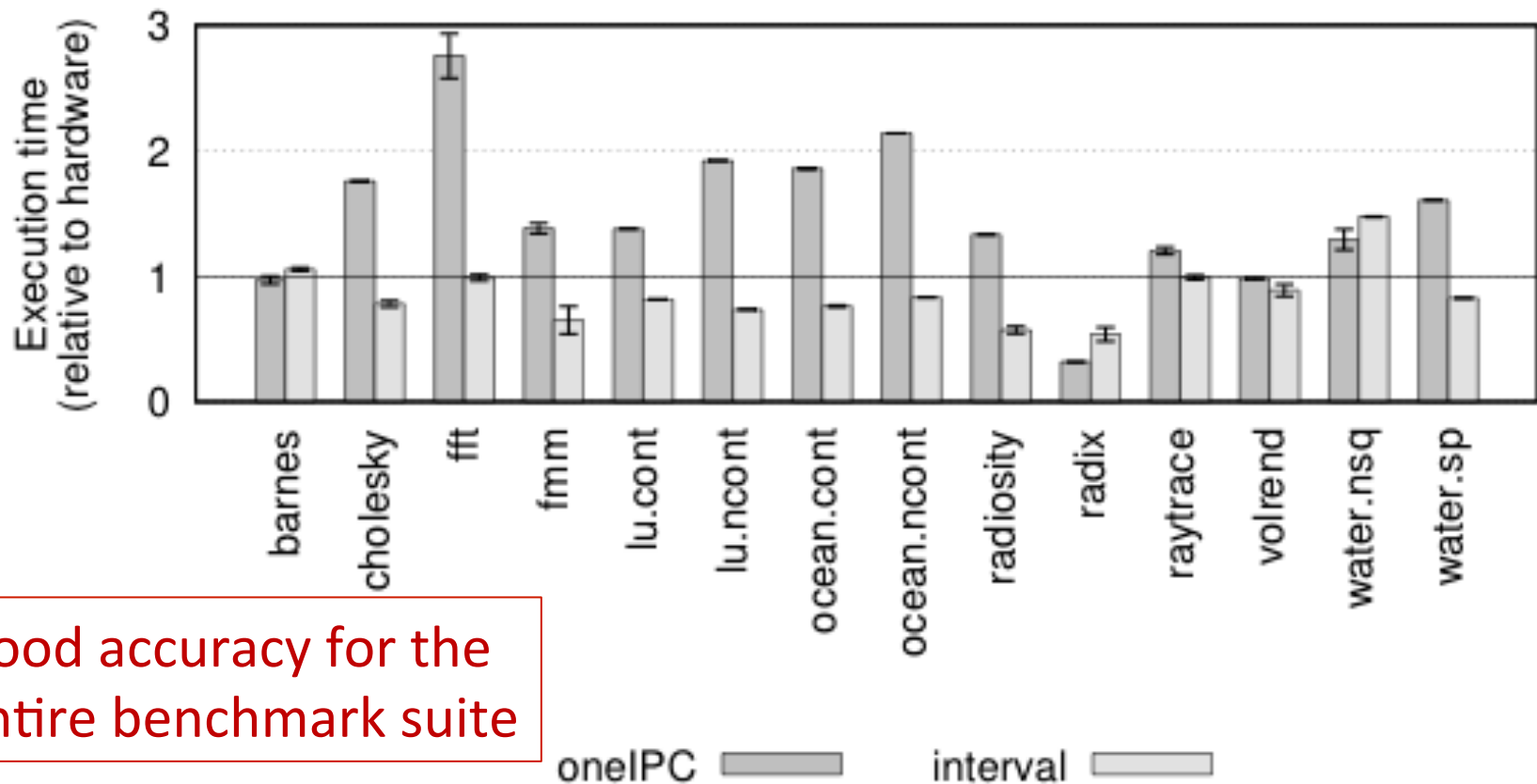


The interval core model provides consistent accuracy of 25% avg. abs. error, with a minimal slowdown



# INTERVAL: GOOD OVERALL ACCURACY

16 cores

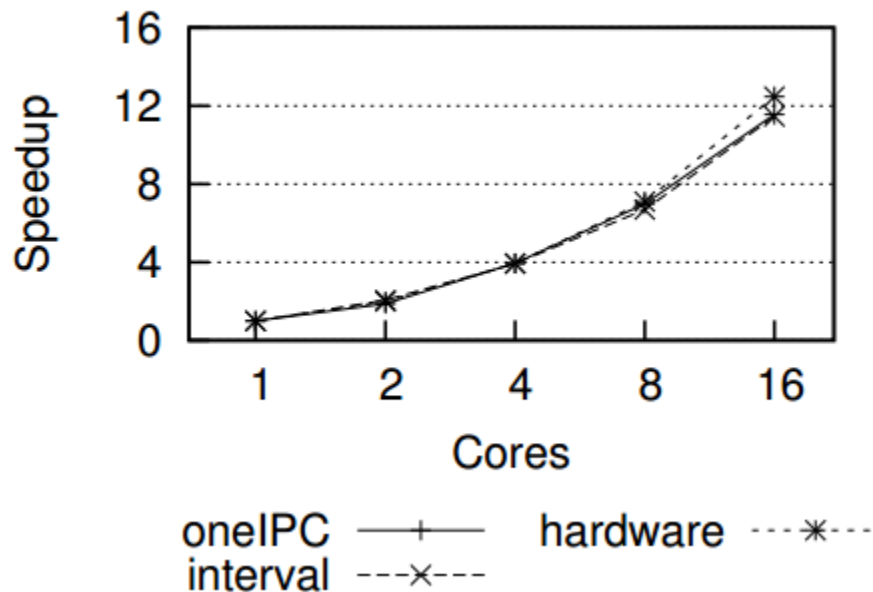


Good accuracy for the entire benchmark suite

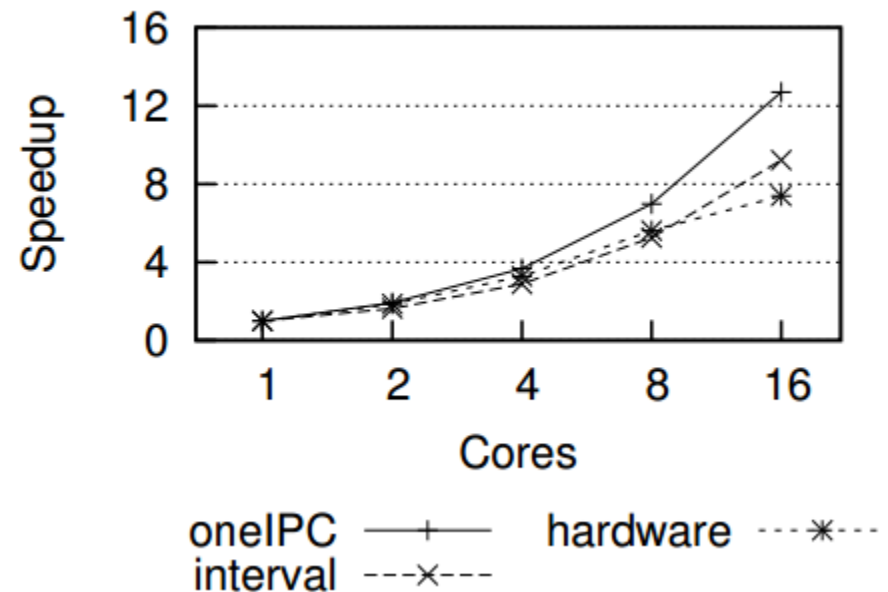
# INTERVAL: BETTER RELATIVE ACCURACY

- Application scalability is affected by memory bandwidth
- Interval model provides more realistic memory request streams, which results in a more accurate scaling prediction

barnes



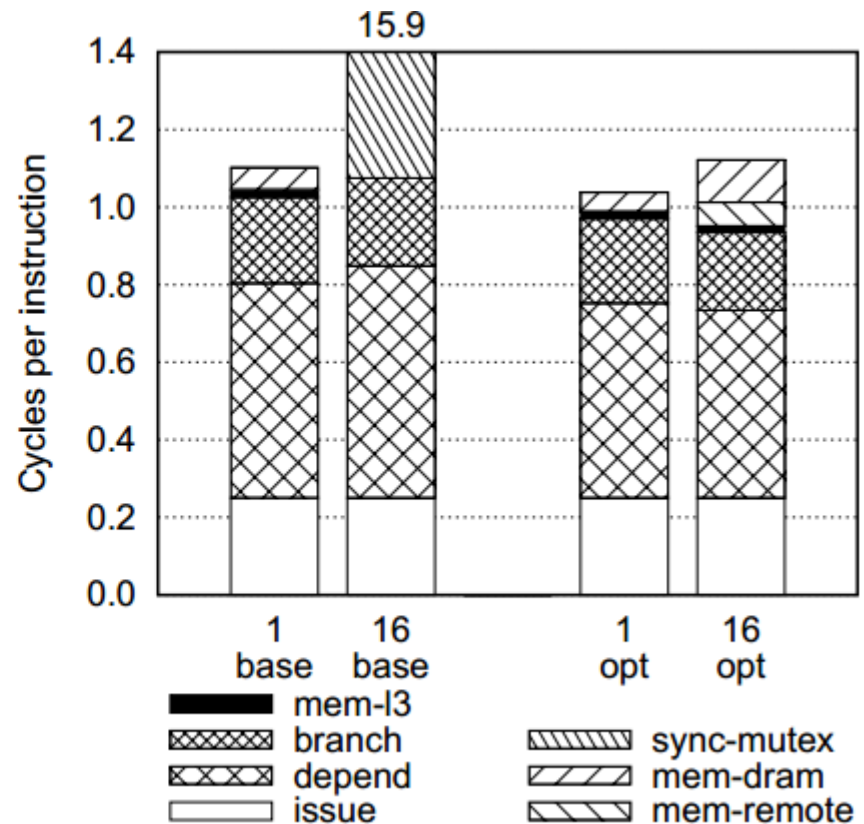
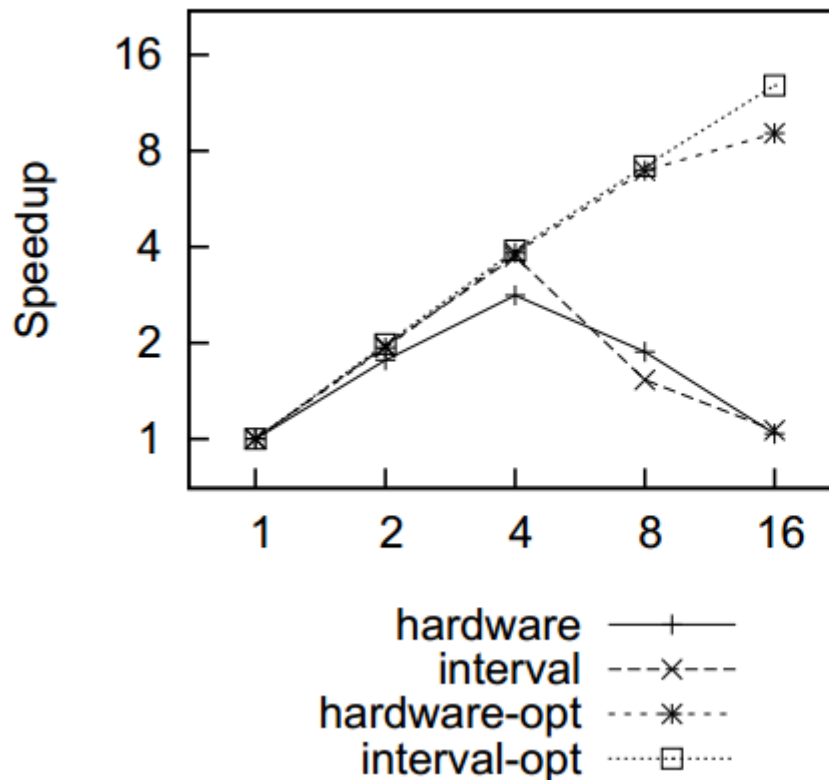
water.nsq



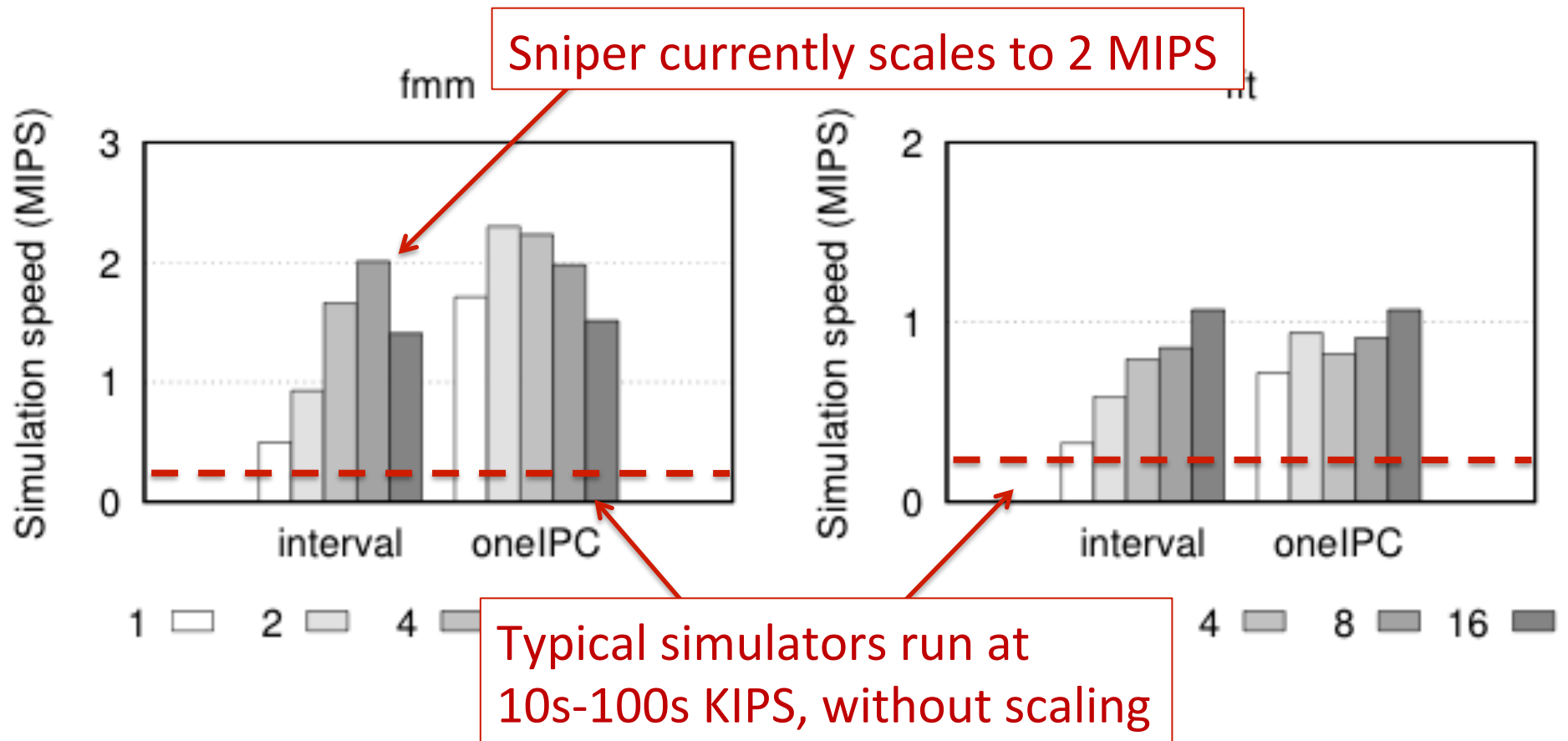


# APPLICATION OPTIMIZATION

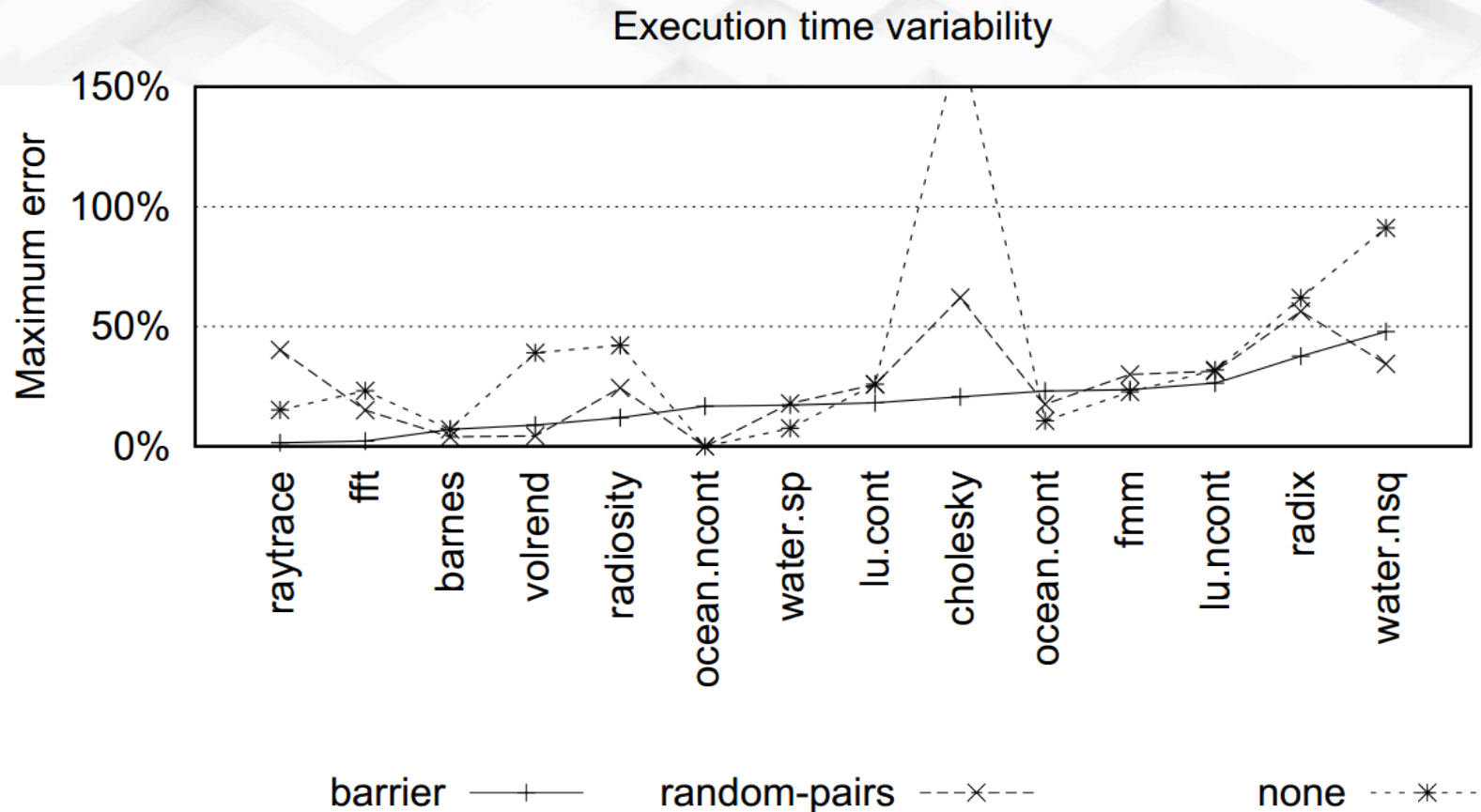
- Splash2-Raytrace shows very bad scaling behavior
- CPI stack shows why: heavy lock contention
- Conversion to use locked increment instruction helps



# SIMULATOR PERFORMANCE

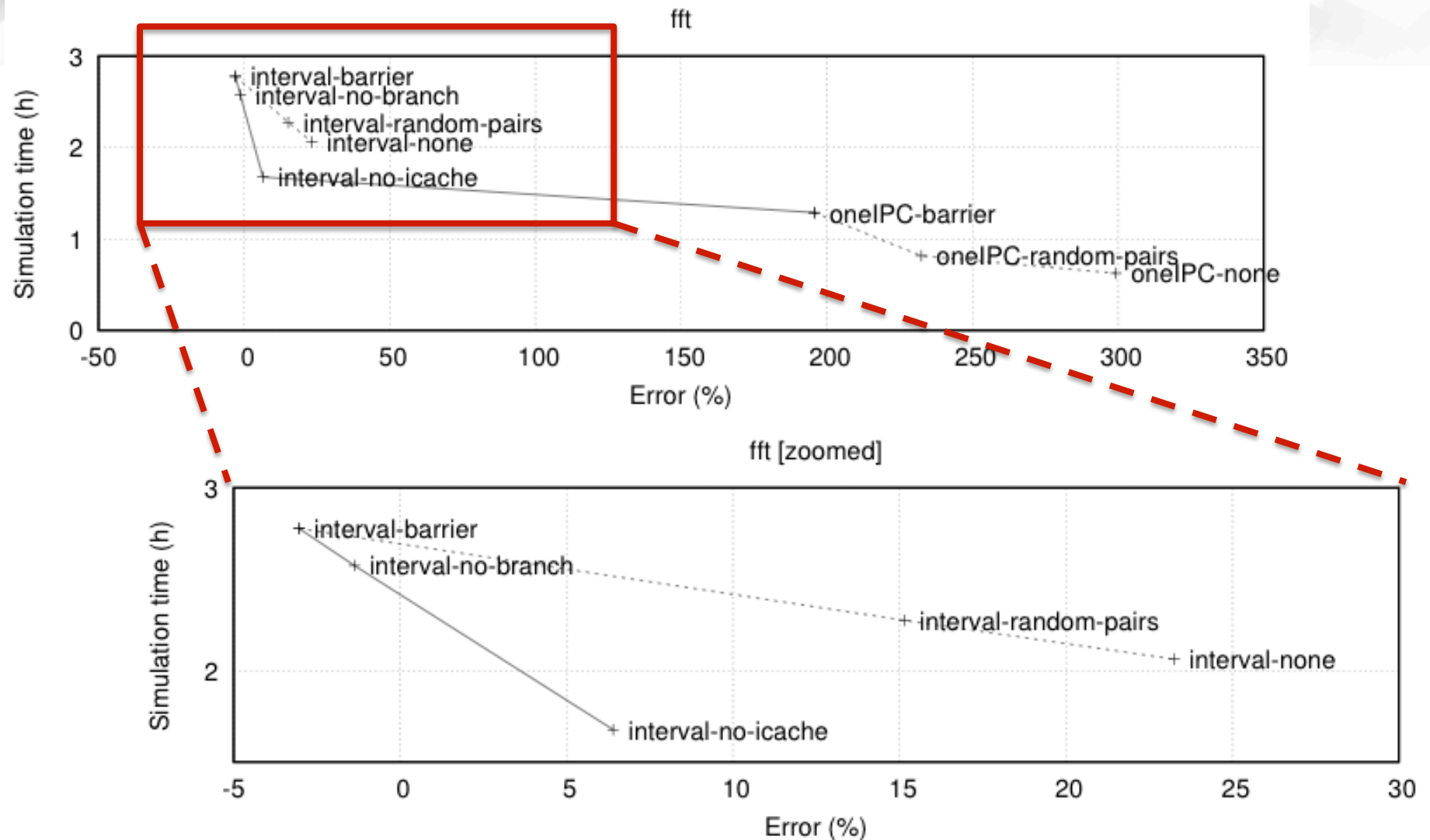


# SYNCHRONIZATION VARIABILITY



Variability due to relaxed synchronization is application specific

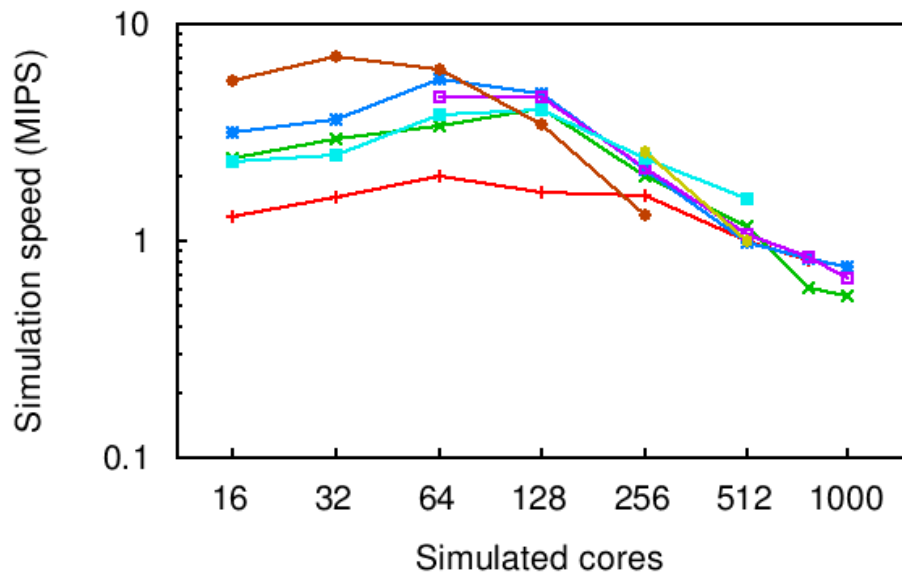
# FLEXIBILITY TO CHOOSE NEEDED FIDELITY



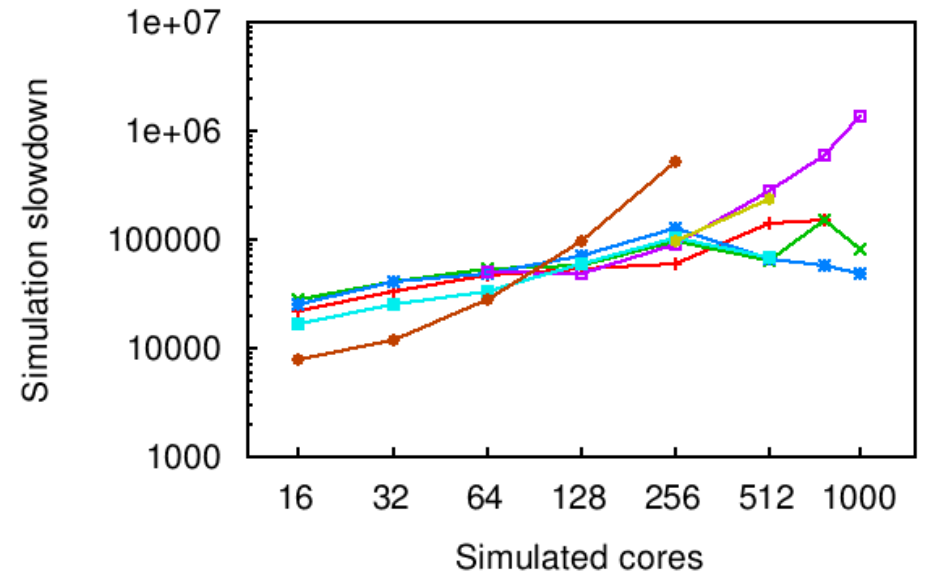
# MANY-CORE SIMULATIONS

## High simulation speed up to 1000 simulated cores

- Pin limitation (to be lifted shortly) at 1020 cores
- Efficient simulation: L1-based benchmarks execute faster
- Host system: dual-socket Xeon X5660 (6-core Westmere), 96 GB RAM



heat-1 —+—  
heat-5 —x—  
heat-9 —\*—  
heat-5-noic —□—  
fft —■—  
indep —●—  
fft-noicache —◆—



heat-1 —+—  
heat-5 —x—  
heat-9 —\*—  
heat-5-noic —□—  
fft —■—  
indep —●—  
fft-noicache —◆—



**ExaScience Lab**  
Intel Labs Europe



**EXASCALE COMPUTING**

# **THE SNIPER MULTI-CORE SIMULATOR RUNNING SIMULATIONS AND PROCESSING RESULTS**

WIM HEIRMAN, TREVOR E. CARLSON  
AND LIEVEN EECKHOUT



[HTTP://WWW.SNIPERSIM.ORG](http://www.snipersim.org)  
SUNDAY, APRIL 1<sup>ST</sup>, 2012  
ISPASS, NEW BRUNSWICK, NJ

# OVERVIEW

---

- Obtain and compile Sniper
- Running
- Configuration
- Simulation results
- Interacting with the simulation
  - SimAPI: application
  - Python scripting

# RUNNING SNIPER

- Download Sniper

- <http://snipersim.org/w/Download>

- Download tar.gz

- Git clone

- ```
~/sniper$ export GRAPHITE_ROOT=$(pwd)
```

- ```
~/sniper$ make
```

- Running an application

- ```
~/sniper$ ./run-sniper -- /bin/true
```

- ```
~/sniper/test/fft$ make run
```



# RUNNING SNIPER

- Integrated benchmarks distribution
  - [http://snipersim.org/w/Download\\_Benchmarks](http://snipersim.org/w/Download_Benchmarks)
  - ~/benchmarks\$ export BENCHMARKS\_ROOT=\$(pwd)
  - ~/benchmarks\$ make
  - ~/benchmarks\$ ./run-sniper -p splash2-fft \  
-i small -n 4
- Standardizes input sets and command lines
- Includes SPLASH-2, PARSEC

# INTEGRATION WITH BENCHMARKS

---

- To add a new benchmark
  - Add source code
  - Add `__init__.py` file
    - Provides application invocation details
    - Define input sets (e.g.: test, small, large)
  - Mark the ROI region
  - Simple example: see `local/pi`

# MULTI-PROGRAMMED WORKLOADS

- Recording traces (SIFT format)

```
$ ./record-trace -o fft -- test/fft/fft -p1
```

- Limited trace, by instruction count:

Fast-forward (-f), detailed length (-d), block size (-b)

```
$ ./record-trace -o fft -f 1e9 -d 1e9 -b 1e8 \  
-- test/fft/fft -p1 -m20
```

- Running traces

```
$ ./run-sniper -c gainestown -n 4 \  
--traces=gcc.sift,swim.sift,\  
swim.sift,quake.sift
```

# REGION OF INTEREST

- Skip benchmark initialization and cleanup
- Mark code with ROI begin / end markers
  - `SimRoiStart()` / `SimRoiEnd()` in your own application
  - `$ ./run-sniper --roi -- test/fft/fft`
- Already done in benchmarks distribution
  - `benchmarks/run-sniper` implies `--roi`
  - Use `--no-roi` to override
- Cache warming during pre-ROI period
  - Use `--no-cache-warming` to override

# CONFIGURATION

- Stackable configuration files (run-sniper -c) and explicit command-line options (-g)
  - Template configurations in sniper/config/\*.cfg (-c name)
  - Your own local configuration files (-c filename.cfg)
  - Explicit option: -g --section/key=value
- Multiple configuration files, and -g options, can be combined
  - Config files specified later on the command line take precedence
  - config/base.cfg is always included
  - If no -c option is provided, config/gainestown.cfg is the default (quad-core Nehalem-based Xeon)
- Complete configuration is stored in sim.cfg after each run

# CONFIGURATION

- Example configuration: largecache.cfg

```
[perf_model/l3_cache]  
cache_size = 16384 # KB
```

```
$ run-sniper -c gainestown -c largecache.cfg
```

- Equivalent to:

```
$ run-sniper -c gainestown \  
  -g --perfmodel/l3_cache/cache_size=16384
```

# SIMULATION RESULTS

---

- Files created after each simulation:
  - **sim.cfg**: all configuration options used for this run (includes defaults, all -c and -g options)
  - **sim.out**: basic statistics (number of cycles, instructions per core, cache access and miss rates, ...)
  - **sim.stats**: complete set of all recorded statistics at key points in the simulation (start, roi-begin, roi-end, stop)
- Use the `graphite_lib` Python package for parsing

# SIMULATION RESULTS

`graphite_lib.get_results()` parses `sim.cfg`, `sim.stats` and returns configuration and statistics (roi-end – roi-begin) for all cores

```
~/sniper/tools$ python
> import graphite_lib
> results = graphite_lib.get_results(resultsdir = '..')
> print results
{'config': {'general/total_cores': '64',
            'perf_model/core/frequency': '2.66', ...},
 'results': {'performance_model.instruction_count': [123],
             'performance_model.elapsed_time': [23000000], ...}}
```



# SIMULATION RESULTS

- Let's compute the IPC for core 0
- Core frequency is variable (DVFS)  
so cycle count has to be computed
  - Time is in femtoseconds, frequency in GHz

```
> instrs = results['results']  
           ['performance_model.instruction_count'][0]  
> cycles = results['results']  
           ['performance_model.elapsed_time'][0]  
           * float(results['config']['perf_model/core/frequency'])  
           * 1e-6 # femtoseconds -> nanoseconds  
> ipc = instrs / cycles  
2.0
```

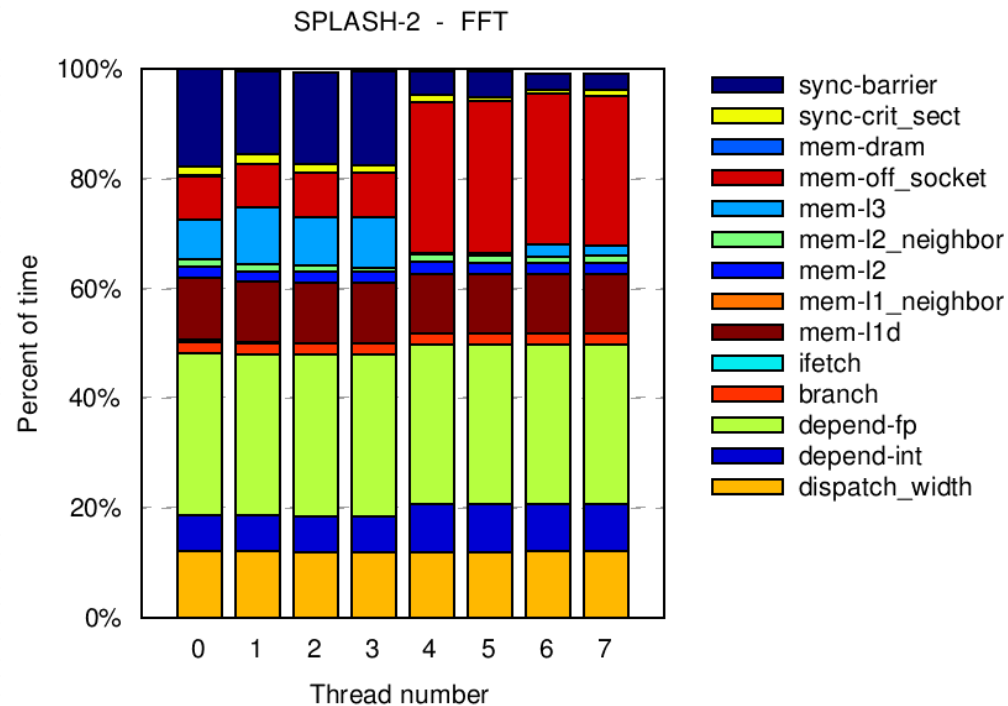
# SIMULATION RESULTS

- CPI stacks (user of graphite\_lib)

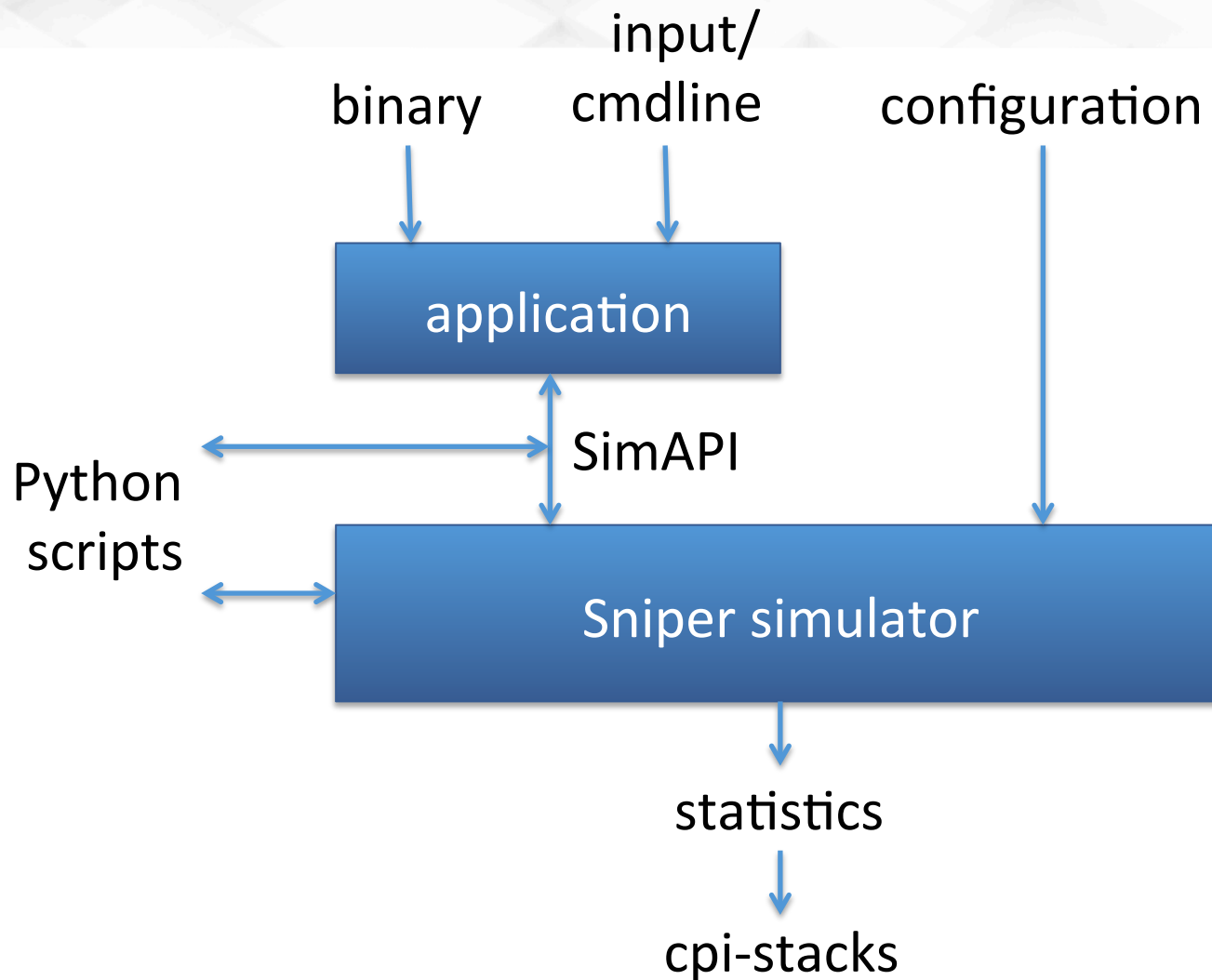
```
$ ./tools/cpistack.py [--time|--cpi|--abstime]
```

	CPI	CPI %	Time %
Core 0			
depend-int	0.20	23.42%	23.42%
depend-fp	0.16	18.94%	18.94%
branch	0.12	14.04%	14.04%
ifetch	0.04	4.16%	4.16%
mem-l1d	0.21	24.41%	24.41%
mem-l3	0.02	2.72%	2.72%
mem-dram	0.05	5.73%	5.73%
sync-mutex	0.02	2.59%	2.59%
sync-cond	0.03	3.01%	3.01%
other	0.01	0.97%	0.97%
total	0.84	100.00%	0.00s

Core 1			
depend-int	0.20	23.92%	23.92%
depend-fp	0.16	18.79%	18.79%
branch	0.12	13.72%	13.72%
mem-l1d	0.20	24.06%	24.06%
mem-l3	0.06	6.79%	6.79%
sync-mutex	0.04	5.22%	5.22%
sync-cond	0.05	5.60%	5.60%
other	0.02	1.89%	1.89%
total	0.85	100.00%	0.00s



# INTERACTING WITH SNIPER



# SIMAPI IMPLEMENTATION

- Magic instructions allow the application to talk to the simulator directly

```
__asm__ __volatile__ (  
    "xchg %%bx, %%bx\n"  
    : "=a" (_res)      /* output      */  
    : "a" (_cmd),  
      "b" (_arg0),  
      "c" (_arg1)      /* input       */  
    );                  /* clobbered */
```

- Pin intercepts this instruction and passes control to the simulator
- Command and arguments passed through rax/rbx/rcx registers, result in rax

# APPLICATION SIMAPI

- Calling simulator API functions from your C program

```
#include <sim_api.h>
```

- SimInSimulator()

- Return 1 when running inside Sniper, 0 when running natively

- SimGetProcid()

- Return processor number of caller

- SimRoiStart() / SimRoiEnd()

- Start/end detailed mode (when using ./run-sniper --roi)

- SimSetFreqMHz(proc, mhz) / SimGetFreqMHz(proc)

- Set / get processor frequency (integer, in MHz)

- SimUser(cmd, arg)

- User-defined function

# PYTHON SCRIPTING

- Scripts are run on simulator startup
  - Register hooks: callbacks when certain events happen during the simulation
  - See `common/system/hooks_manager.h` for all available hooks
- Use an existing script from `sniper/scripts/*.py`:  
`./run-sniper -s scriptname`
- Or your own script:  
`./run-sniper -s myscriptname.py`
- Use `sim` package for convenience wrappers

# PYTHON SCRIPTING

---

- Low-level script
- Execute “foo” at each barrier synchronization

```
import sim_hooks  
def foo(t):  
    print 'The time is now', t  
sim_hooks.register(sim_hooks.HOOK_PERIODIC, foo)
```

# PYTHON SCRIPTING

- Higher-level script
- Execute “foo” at each barrier synchronization

```
import sim
class Class:
    def hook_periodic(self, t):
        print 'The time is now', t
sim.util.register(Class())
```



# PYTHON SCRIPTING

- High-level script: execute “foo” every X ms
- Pass in parameter using  
`./run-sniper -s myscript.py:X`

```
import sim
class Class:
    def setup(self, args):
        sim.util.Every(long(args)*sim.util.Time.MS,
                        self.periodic)
    def periodic(self, t, t_delta):
        print 'The time is now', t
        print 'Elapsed time since last call', t_delta
sim.util.register(Class())
```

# PYTHON SCRIPTING

- Access configuration, statistics, DVFS
- Live periodic IPC trace:
  - See `scripts/ipctrace.py` for a more complete example

```
class IPCTracer:
    def setup(self, args):
        sim.util.Every(1*sim.util.Time.US, self.periodic)
        self.instrs_prev = 0
    def periodic(self, t, t_delta):
        freq = sim.dvfs.get_frequency(0)
        cycles = t_delta * freq * 1e-9 # fs * MHz -> cycles
        instrs = long(sim.stats.get('performance_model', 0,
                                    'instruction_count'))
        print 'IPC =', (instrs - self.instrs_prev) / cycles
        self.instrs_prev = instrs
```

# PYTHON & MAGIC INSTRUCTIONS

- Communicate information between application and Python script
  - E.g.: simulated hardware performance counters

- Application:

```
uint64_t ninstrs = SimUtil(0xdeadbeef, SimGetProcId())
```

- Python script:

```
class PerfCtr:  
    def setup(self):  
        sim.util.register_command(0xdeadbeef, self.compute)  
    def compute(self, arg):  
        return sim.stats.get('performance_model', arg,  
                              'instruction_count')
```

# NEAR TERM IDEAS

---

- Multiple processes
  - A number of multi-threaded applications
  - MPI support
- Heterogeneous cores at run-time
  - Big: 4-issue processor
  - Small: 2-issue processor
- Scheduling/Migration support
- Multiple processor configurations
  - Currently the simulator is compiled to support a single type of processor (Core2 vs. Nehalem vs. Sandy Bridge)

# REFERENCES

---

- Sniper website
  - <http://snipersim.org/>
- Download
  - <http://snipersim.org/w/Download>
  - [http://snipersim.org/w/Download Benchmarks](http://snipersim.org/w/Download_Benchmarks)
- Getting started
  - [http://snipersim.org/w/Getting Started](http://snipersim.org/w/Getting_Started)
- Questions?
  - <http://groups.google.com/group/snipersim>
  - [http://snipersim.org/w/Frequently Asked Questions](http://snipersim.org/w/Frequently_Asked_Questions)