Deterministic PinPoints:

Representative and Repeatable Simulation Region Selection with PinPlay and Sniper

Harish Patil (*Intel Corporation*)
Trevor E. Carlson (*Ghent University*)

February 23rd, 2013 HPCA, Shenzhen, People's Republic of China

Legal Disclaimer & Optimization Notice

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Executive Summary

PinPoints methodology effectively automates the tedious task of finding and check-pointing regions of programs for Pin-based simulation.*

** Creates check-points that are <u>representative</u> and <u>repeatable</u>.*

This Tutorial

IS about:

Tools and scripts for finding representative regions (***PinPoints***) of large programs, **check-pointing** them, and **feeding** them to Pin-based simulators.

IS NOT:

A detailed **description** of **Pin, PinPlay, or Sniper**

Outline

1. Introduction
2. An Overview of Pin, SimPoint, and PinPlay
3. PinPlay kit
4. Sniper:
   - Overview
   - PinPlay Integration
   - SPEC2006 Study
   - PinPoints Integration
5. Handling Parallel Programs

Pin-based Simulation: Two Usage Models

1. Pintool *is* the simulator

2. Pintool *feeds* the simulator

Simulation Challenges: Processor & Application Complexity

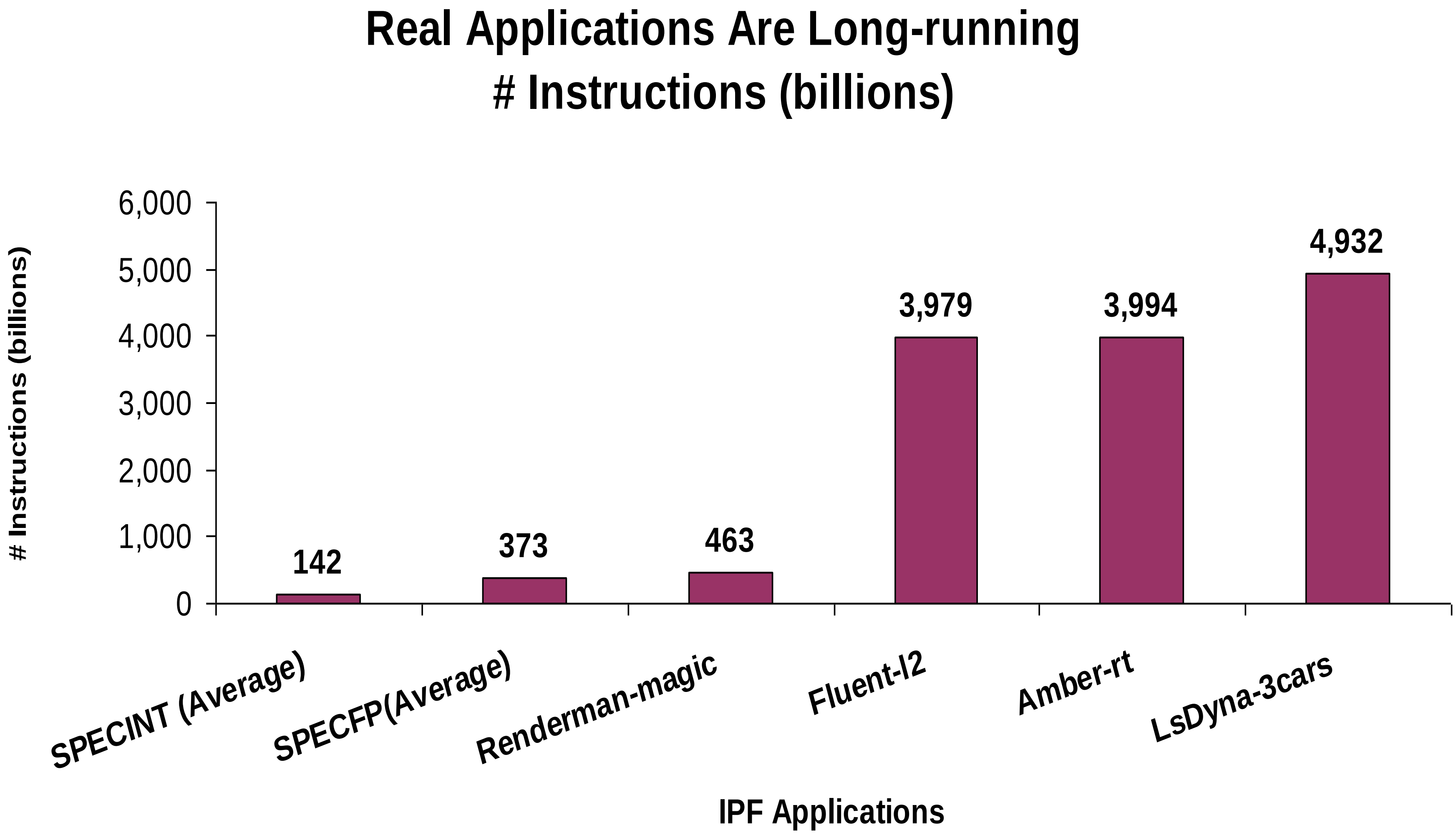
1. **Complex Processors/ Slower Simulation**

- Detailed, cycle-accurate, simulation is slow
- Typically few thousand instructions per second

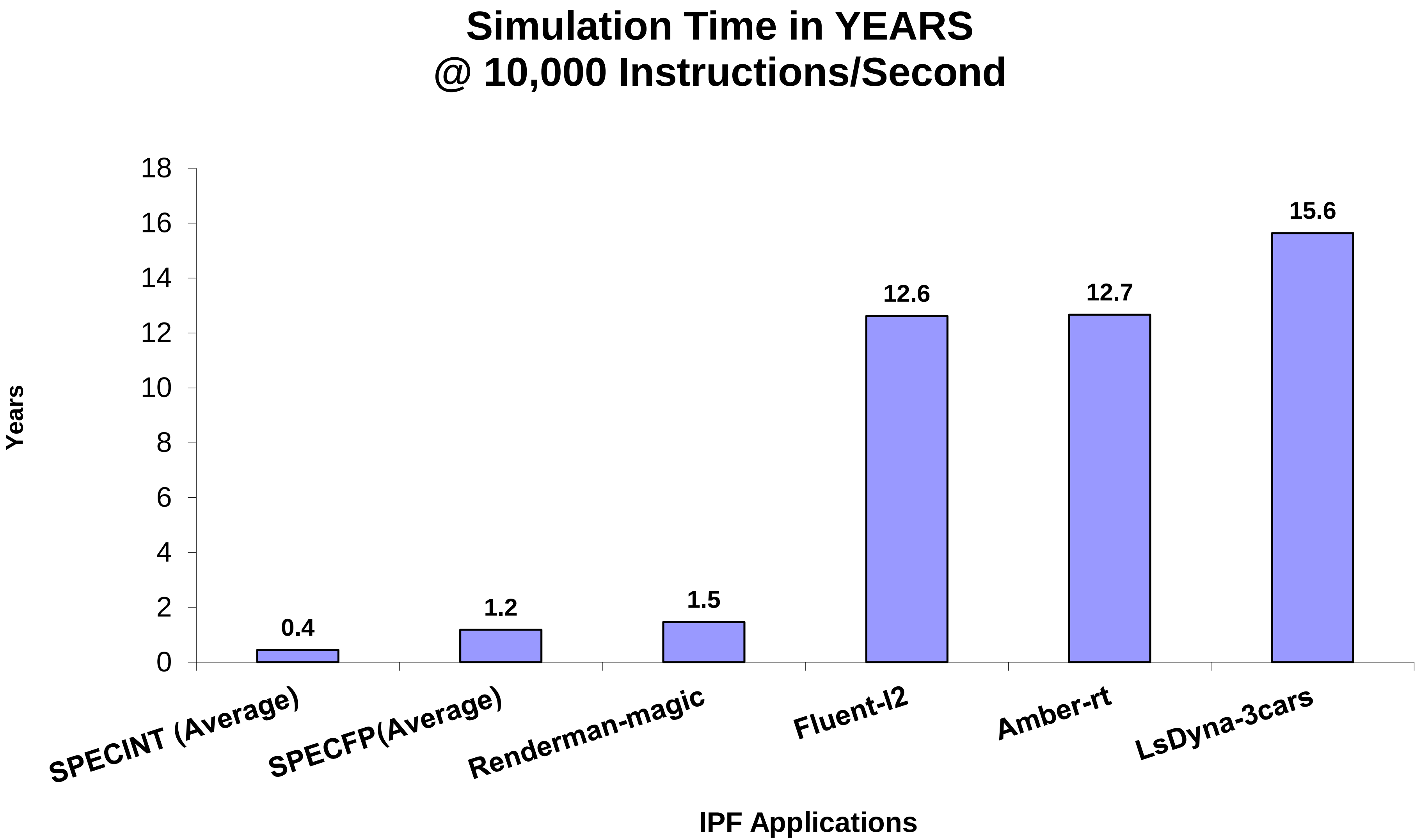
2. **Complex Applications/ Porting Difficulty**

- OS dependency; large memory/disk requirement
- Porting/Running on simulators impractical

Instruction Counts : Some Real Applications



Problem: Whole-Program Simulation is Slow



Optimization Notice

Solution: Select Regions to Simulate

Select One Region

- At the beginning (no skip)
- After 1 billion instructions
- After skipping a random number of instructions

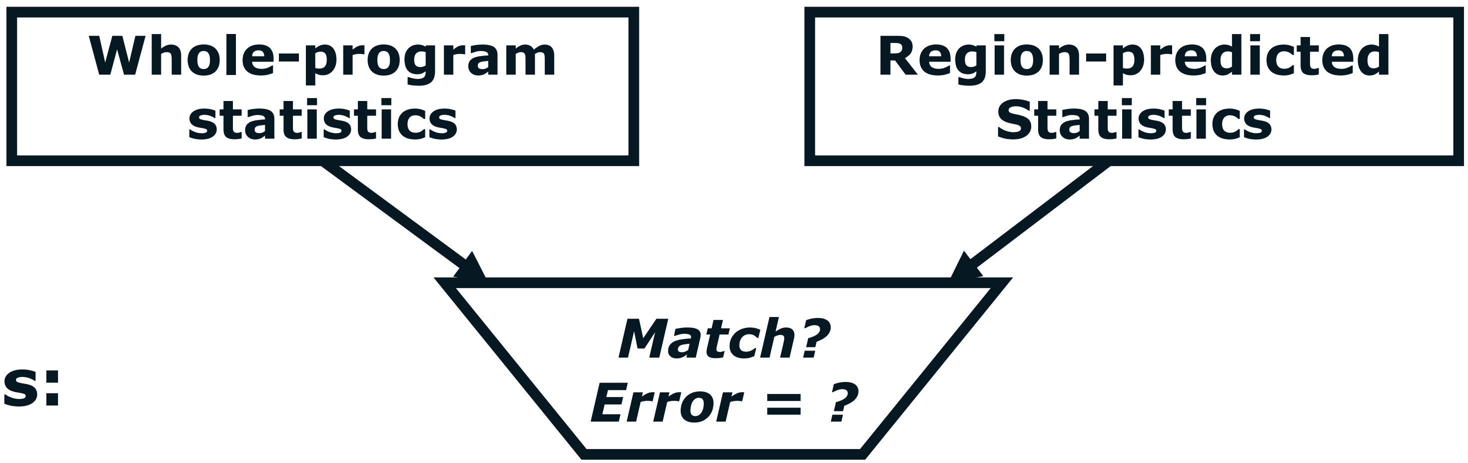
Select Multiple Regions

- Manually by looking at performance data – too tedious
- Randomly anywhere
- Randomly from uniform regions
- By program phase analysis (SimPoint:UCSD)
- Fine-grain sampling (SMARTS: CMU)

Do the selected regions represent whole-program behavior?

How Representative are Simulation Regions?

Compare Whole-program Statistics with Statistics Predicted with Simulation of Regions

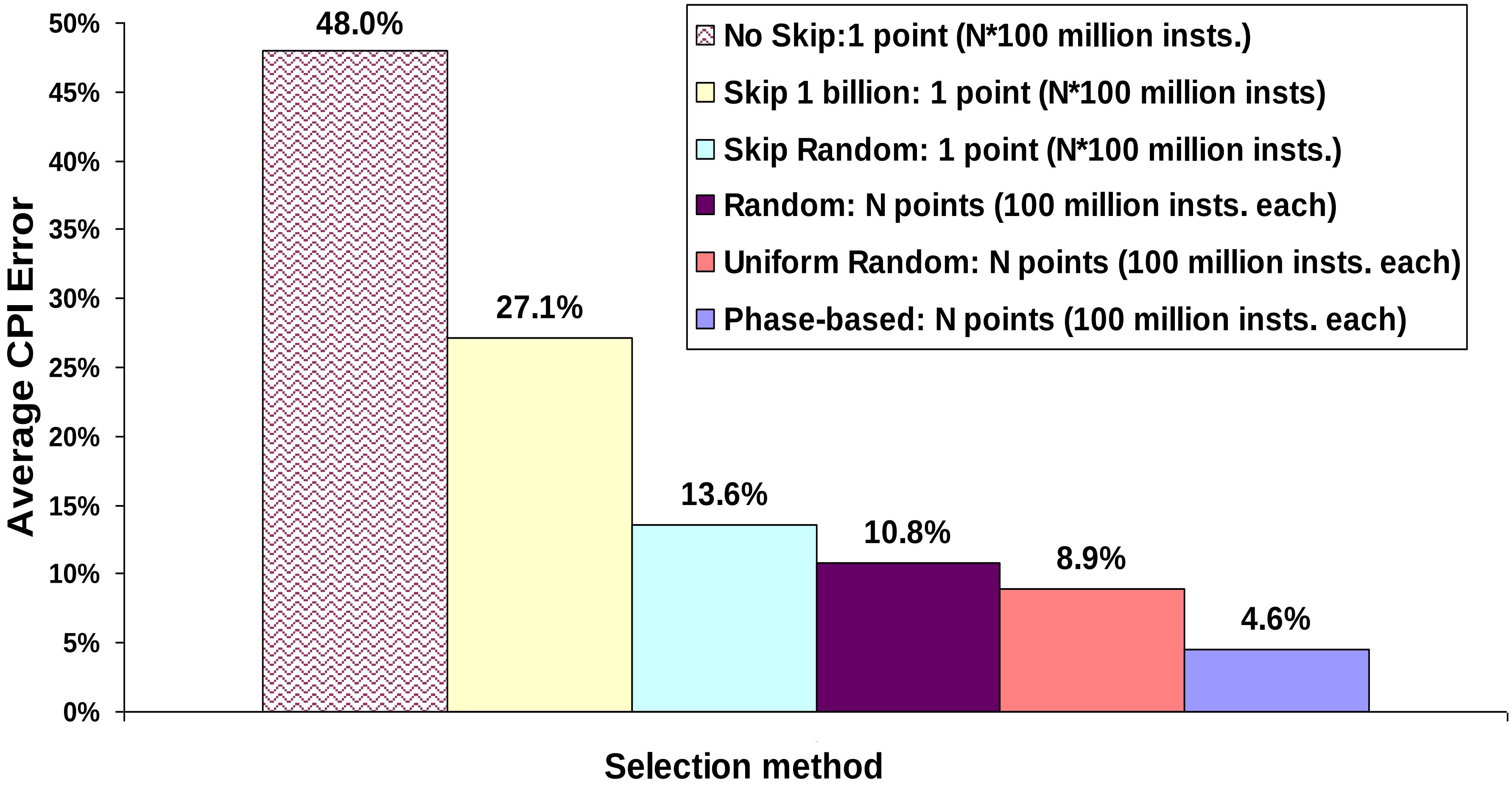


Caveats:

1. Can never "prove" region selection is "best possible"
2. Whole-program statistics computation should be "fast" (cannot use detailed simulation which could take years!)

Optimization Notice

CPI: *Average Error* SPEC2000(IA32) Whole Program vs. Selected Points



PinPoints =
Pin (Intel) + SimPoint (UCSD)*

- **What are PinPoints** : Representative Regions of Programs
  - Automatically chosen
  - Validated (multiple times: latest with Sniper from UGHent)
- **<u>PinPoints Methodology</u>**:
  - Use : Pin, SimPoint, a Pin-based Simulator (e.g. Sniper), multiple scripts
  - Find PinPoints, checkpoint them, validate them

** Acknowledgement: Timothy Sherwood, Erez Perelman, Greg Hamerly and Brad Calder*

Simulation Challenges: Our Solutions

1. **Detailed, cycle-accurate, simulation is slow**

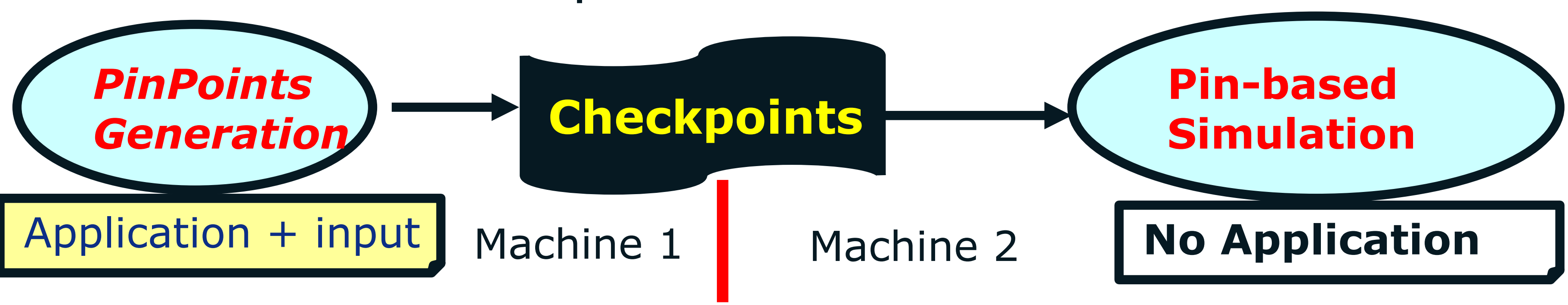
Solution: Find/simulate only representative regions

- PinPoints cover << 1% of whole-program execution

→ **vastly reduced simulation time**

2. **Complex Applications/ Porting Difficulty**

- Create check-points transfer to simulation



PinPoints-based Simulation + Prediction

1. **Simulate with check-points** for PinPoints

- Checkpoints capture region execution (details later)
- No fast-forwarding needed

2. **Predict : CPI, MPI,… Stats-per-instruction**

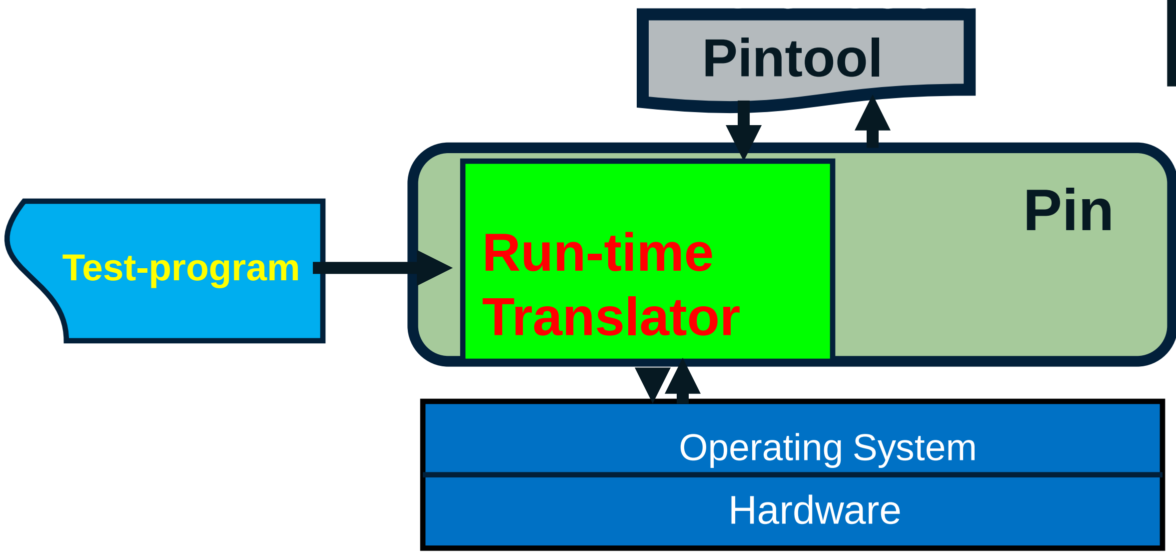
- Use statistics from PinPoints simulation, total instruction count, PinPoints weights…

Pin, SimPoint, PinPlay : Overview

Pin: A Tool for Writing Program Analysis Tools

```
sub     $0xff, %edx
movl    0x8(%ebp), %eax
jle     <L1>
```

```
counter++; print(IP)
sub     $0xff, %edx
counter++; print(EA)
movl  0x8(%ebp), %eax
counter++;print(br_taken)
jle     <L1>
```




```
$ pin -t pintool -- test-program
```

Normal output
+ *Analysis output*

Pin: A Dynamic Instrumentation Framework from Intel
http://www.pintool.org

SimPoint from UCSD

Goals

- The goals of this research are:
  - To create an automatic system that is capable of intelligently characterizing time-varying program behavior
  - To provide both analytic and software tools to help with program phase identification
  - To demonstrate the utility of these tools for finding places to simulate (SimPoints)
  - Without full program detailed simulation

ASPLOS : Sherwood et. al.

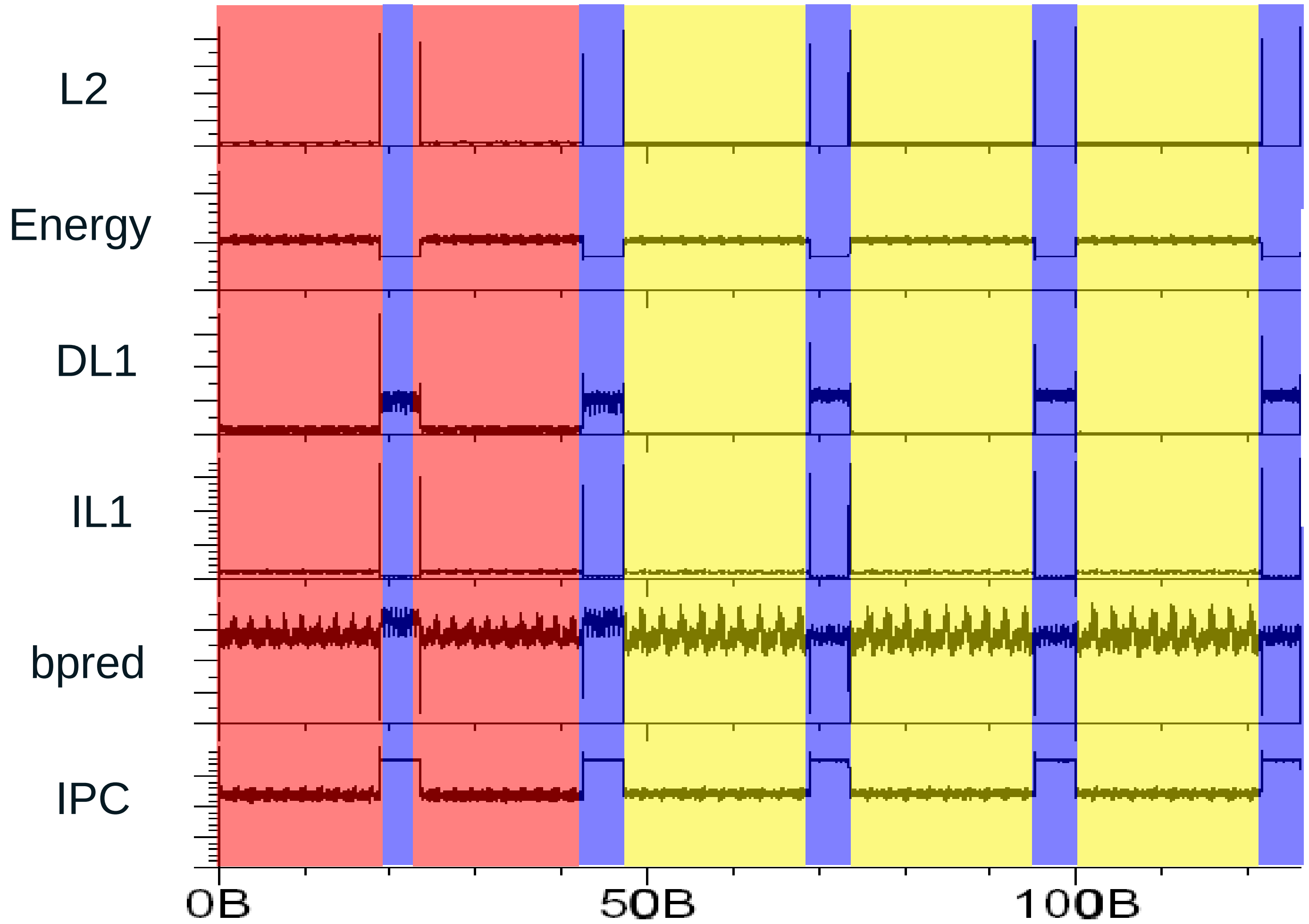
UCSD SimPoint Work: Key Idea

Simulation Points

- Should represent the program behavior *independent of hardware*

- Identified using program-dependent metric
  relative execution counts of basic blocks

Large Scale Behavior (gzip)

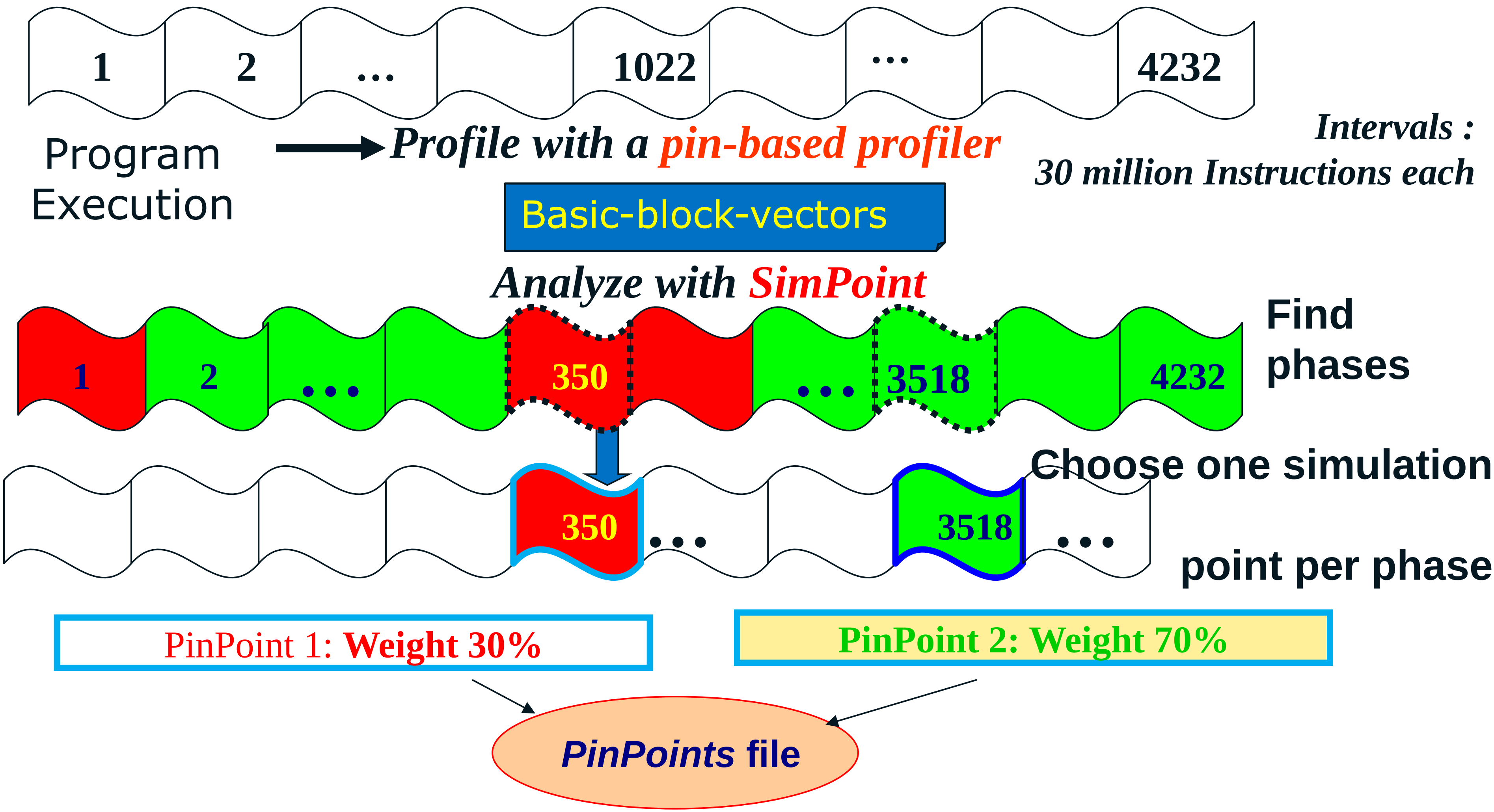


UCSD SimPoint Work: 3 Steps

1. Profile → ***Basic Block Vectors***
2. Analyze Profiles → ***Simulation Points***
   - ***SimPoints*** : (skip length, weight)
3. Verify → Compare Selective Statistics (ipc, miss rates) → ***Error rates***
   - Compare full run statistics vs SimPoints statistics

Optimization Notice

PinPoints = Pin + SimPoint



Two Phases => Two PinPoints

Optimization Notice

PinPoints: Estimating Total Execution Time

Total Execution Time = Total Cycles / Frequency

- We know the simulated Frequency; need to know Total Cycles for *full* run of the binary on the Simulator

Total Cycles Simulated = (Weighted CPI) * (Total Instructions)

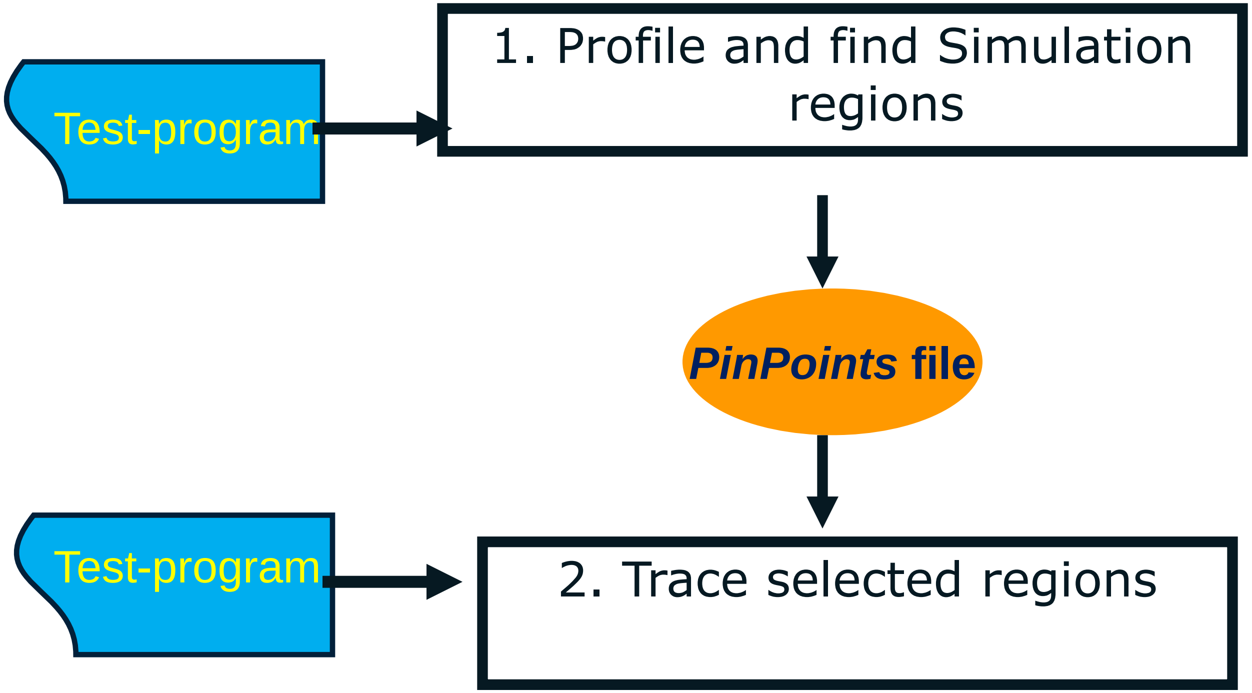
- PinPoints provides the Total number of instructions in the PinPoints file.

Weighted CPI can be determined through simulation of PinPoints regions and weighting of results:

$$\textit{Weighted CPI} = \sum \textit{Weight}_i * \textit{CPI}_i$$

- CAUTION: Use the formula only for statistics normalized by instructions : CPI computation OK; IPC computation is NOT OK

PinPoints : The Repeatability Challenge



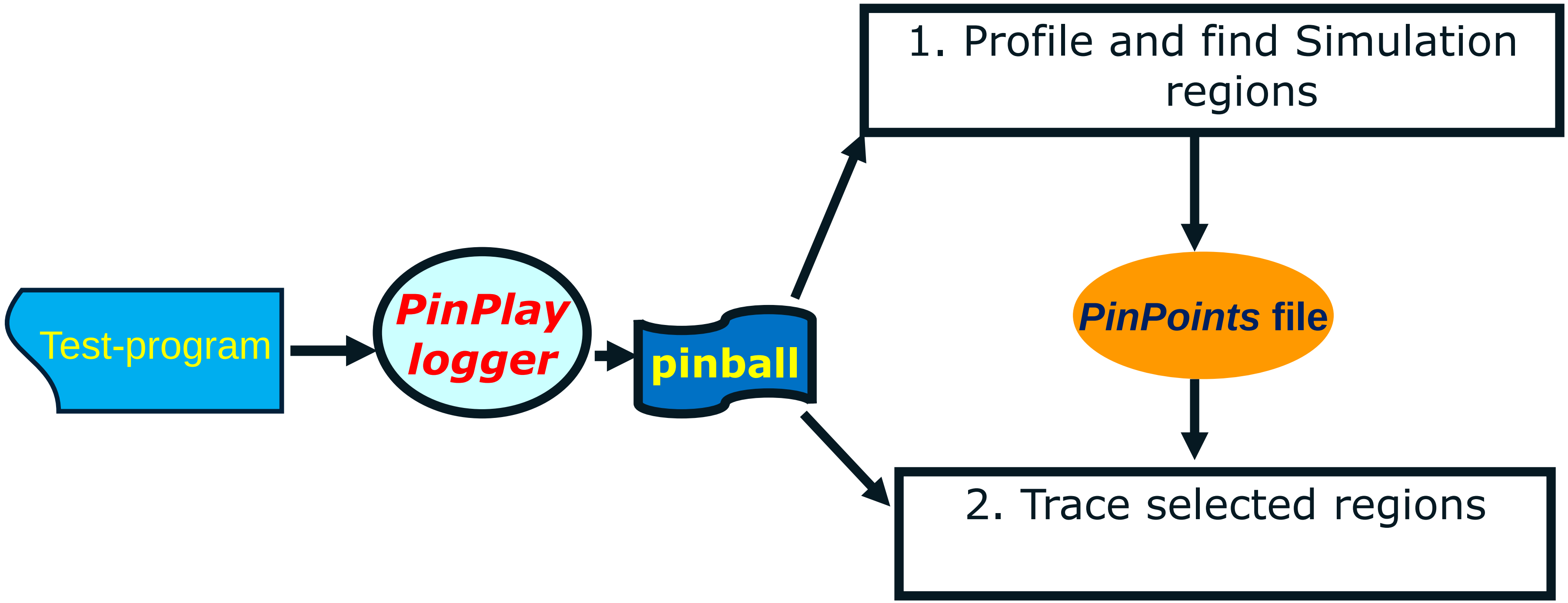
Problem: Two runs are not exactly same → PinPoints missed

Found this for 25/54 SPEC2006 runs!

[*"PinPoints out of order" "PinPoint End seen before Start"*]

Optimization Notice

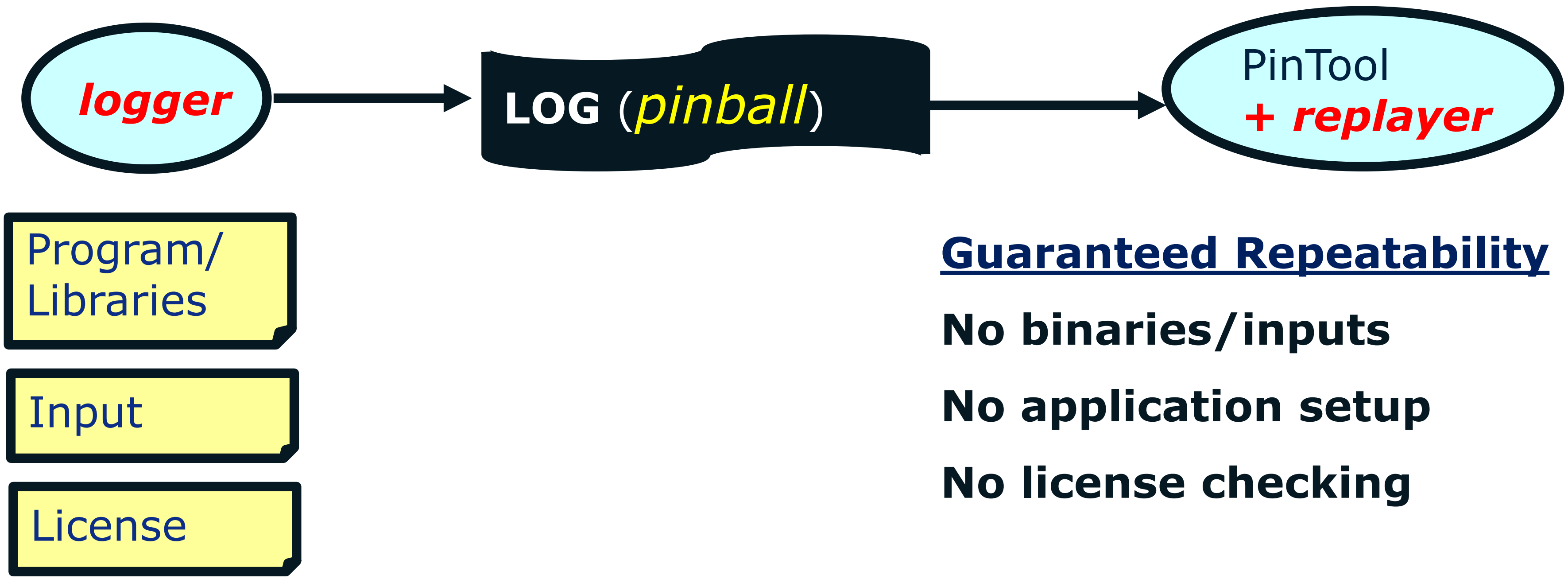
Enters PinPlay To Provide Repeatability



Two runs are same → PinPoints guaranteed to be reached

Optimization Notice

PinPlay*: Workload Capture and Deterministic Replay Framework



Guaranteed Repeatability

No binaries/inputs

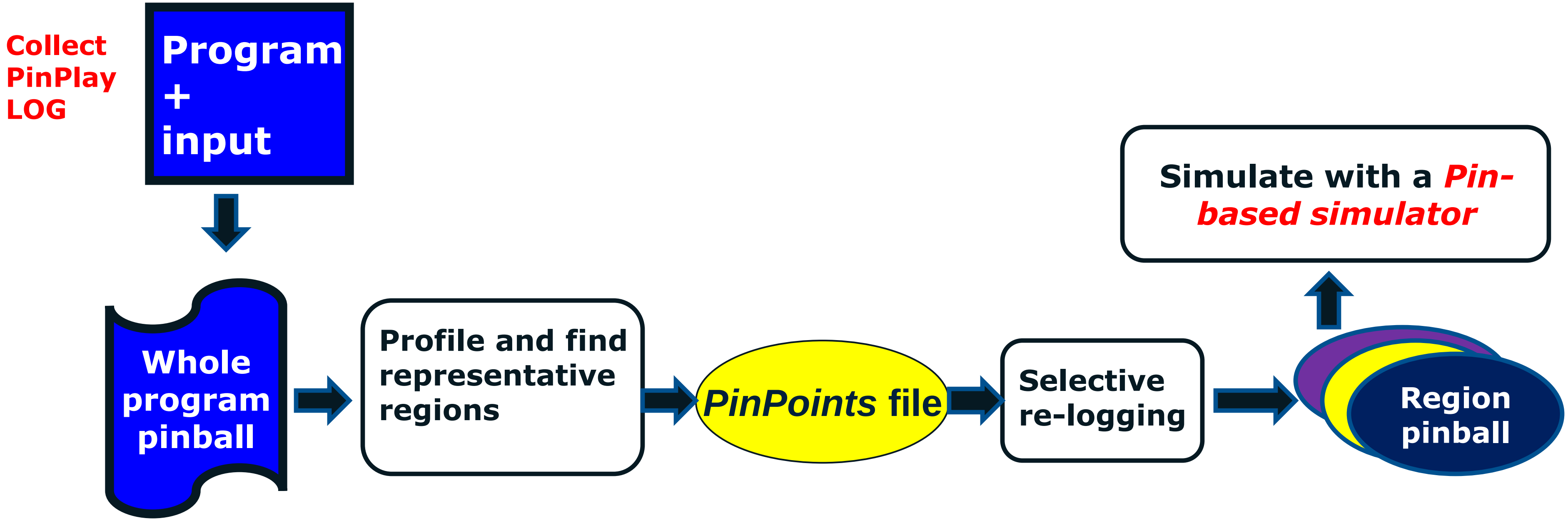
No application setup

No license checking

Record Once Replay + Analyze Multiple Times Anywhere!

* *Developers: Cristiano Pereira, James Cownie, Harish Patil*

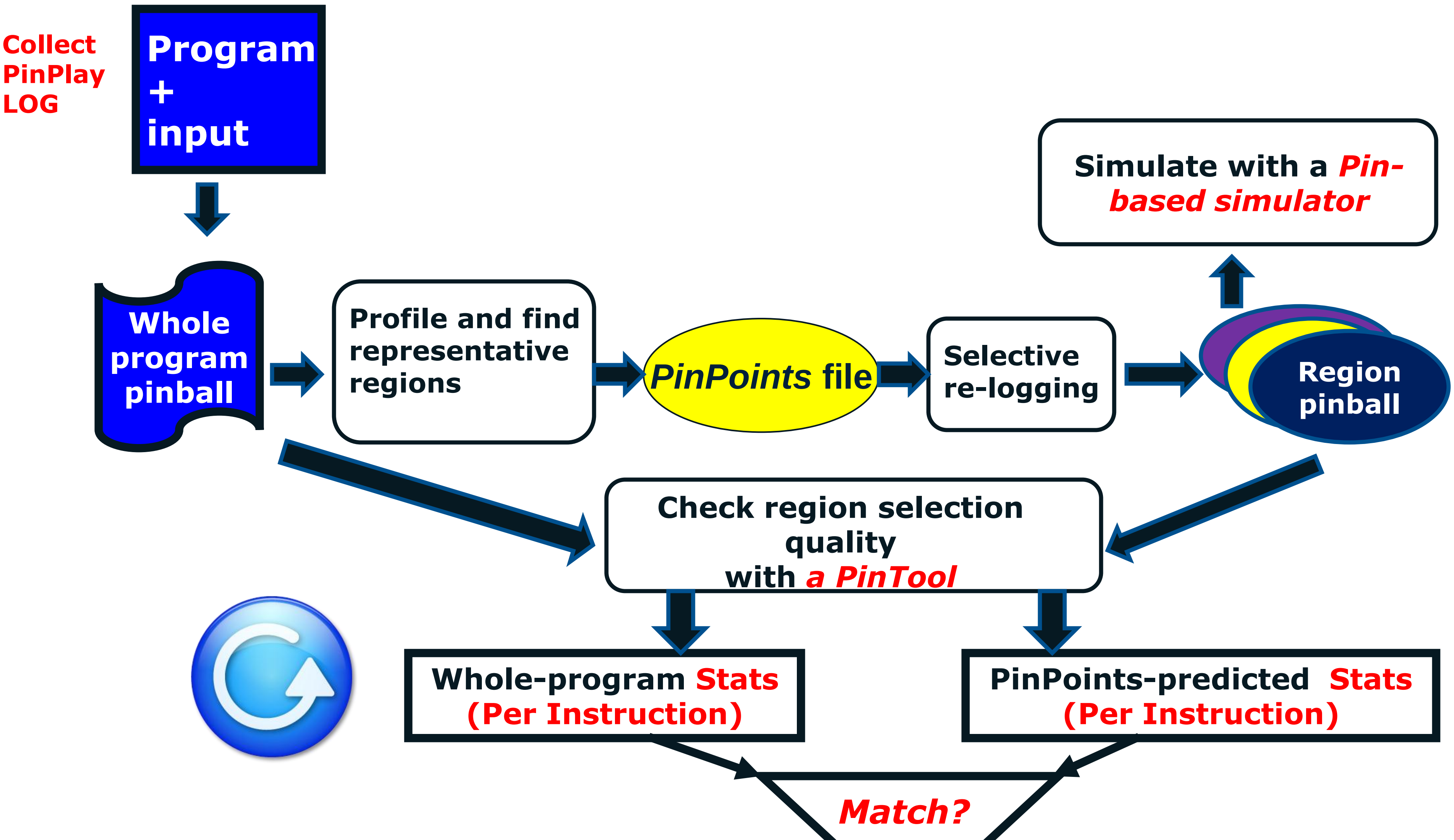
PinPlay + PinPoints: Basic Flow



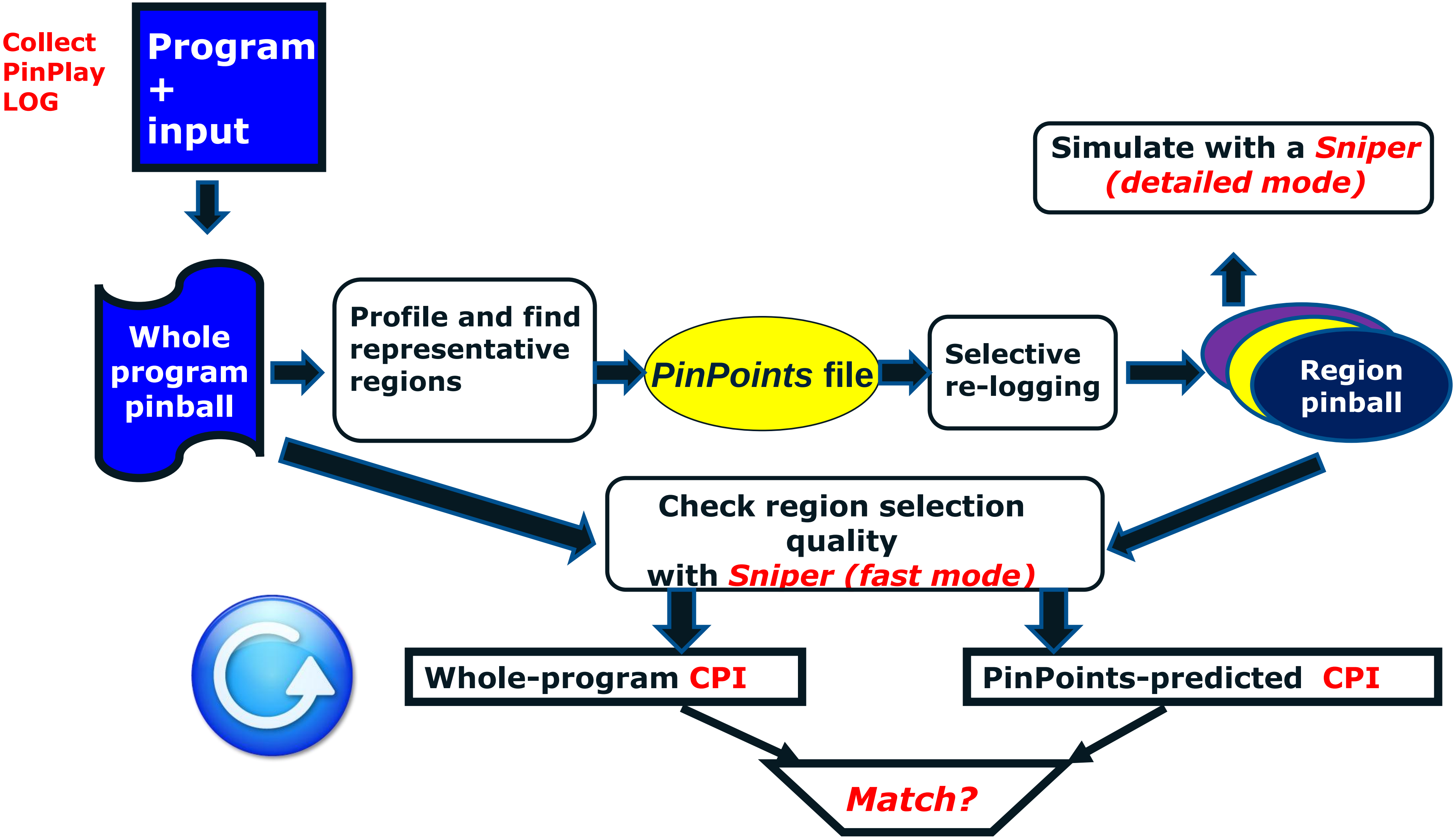
- PinPoints are representative (validation/tuning possible)
- PinPoints cover << 1% of whole-program execution

➔ **vastly reduced simulation time**

PinPoints validation: "*Functional Correlation*"



PinPlay + PinPoints + Sniper



The PinPlay kit

Download from http://www.pinplay.org

pinplay-1.0-pin-2.12-55942-gcc.4.4.7-linux

<PinPlay version> ***<base Pin kit used>***

PinPlay Kit = Pin kit + *extras/pinplay*

```
pinplay-1.0-pin-2.12-55942-gcc.4.4.7-linux/extras/pinplay/
|-- PinPoints
|    |-- bin
|    `-- scripts
|-- bin
|    |-- ia32
|    `-- intel64
|-- examples
|    `-- tests
|-- include
|-- include
|-- lib
|    |-- ia32
|    `-- intel64
`-- lib-ext
     |-- ia32
     `-- intel64
```

```
|-- README.PinPoints
|-- bin
|    |-- LICENSE.simpoint
|    `-- simpoint
```

```
|-- Makefile
|-- bimodal.H
|-- pinplay-branch-predictor.cpp
|-- pinplay-driver.cpp
```

```
`-- pinplay.H
```

```
`-- libpinplay.a
```

Installation : Linux-64

[Prerequisite: gcc/g++ available for both 32 and 64 bit]

```
%setenv PIN_KIT <root of pinplay kit>
%cd $PIN_KIT/extras/pinplay/examples
% make
```

Builds and tests pinplay-driver.so and pinplay-branch-predictor.so

```
extras/pinplay/bin/
|-- ia32
|    |-- nullapp
|    |-- pinplay-branch-predictor.so
|    `-- pinplay-driver.so
`-- intel64
     |-- nullapp
     |-- pinplay-branch-predictor.so
     `-- pinplay-driver.so
```

```
extras/pinplay/examples/
|-- Makefile
|-- bimodal.H
|-- foo.bimodal.ia32.out
|-- foo.bimodal.intel64.out
|-- hello32
|-- hello64
|-- pinball
|    |-- foo.0.dyn_text.bz2
|    |-- foo.0.race.bz2
|    |-- foo.0.reg.bz2
|    |-- foo.0.result
|    |-- foo.0.result_play
|    |-- foo.0.sel.bz2
|    |-- foo.0.sync_text.bz2
|    |-- foo.address
|    |-- foo.log.txt
|    |-- foo.procinfo.xml
|    |-- foo.replay.txt
|    `-- foo.text.bz2
```

Enabling a Pintool for PinPlay

```
#include "pinplay.H"
```

```
PINPLAY_ENGINE pinplay_engine;
```

```
KNOB<BOOL>KnobReplayer(KNOB_MODE_WRITEONCE, KNOB_FAMILY,
                       KNOB_REPLAY_NAME, "0", "Replay a pinball");
KNOB<BOOL>KnobLogger(KNOB_MODE_WRITEONCE,  KNOB_FAMILY,
                     KNOB_LOG_NAME, "0", "Create a pinball");
```

```
  pinplay_engine.Activate(argc, argv, KnobLogger, KnobReplayer);
```

Link in ***libpinplay.a, libzlib.a, libbz2.a***

Restrictions:

1. **PinTool shouldn't change application control flow**
2. **Image API not available during replay**

Example: pinplay-branch-predictor.cpp

```
#define KNOB_LOG_NAME  "log"
#define KNOB_REPLAY_NAME "replay"
#define KNOB_FAMILY "pintool:pinplay-driver"


PINPLAY_ENGINE pinplay_engine;

KNOB_COMMENT pinplay_driver_knob_family(KNOB_FAMILY, "PinPlay Driver Knobs");

KNOB<BOOL>KnobReplayer(KNOB_MODE_WRITEONCE, KNOB_FAMILY,
                       KNOB_REPLAY_NAME, "0", "Replay a pinball");
KNOB<BOOL>KnobLogger(KNOB_MODE_WRITEONCE,  KNOB_FAMILY,
                     KNOB_LOG_NAME, "0", "Create a pinball");

int main(int argc, char *argv[])
{
    if( PIN_Init(argc,argv) )
    {
        return Usage();
    }

    outfile = new ofstream(KnobStatFileName.Value().c_str());
    bimodal.Activate(KnobPhases, outfile);

    pinplay_engine.Activate(argc, argv, KnobLogger, KnobReplayer);

    PIN_AddThreadStartFunction(threadCreated, reinterpret_cast<void *>(0));


    PIN_StartProgram();
}
```

PinPlay-enabled PinTools : 3 Modes

1. **Regular Analysis mode**

```
$ pin -t pintool -- test-program
```

Normal output
+ *Analysis output*

2. **Logging Mode**

```
$ pin -t pintool -log -log:basename pinball/foo -- test-program
```

pinball

3. **Replay Mode**

```
$ pin -t pintool -replay -replay:basename pinball/foo -- nullapp
```


Optimization Notice

Example: pinplay-branch-predictor.so

```
% $PIN_KIT/pin -t
$PIN_KIT/extras/pinplay/bin/intel64/pinplay-
branch-predictor.so -- hello
```

Creates "bimodal.out"

```
%pin -t pinplay-branch-predictor.so -log -
log:basename pinball/foo - hello
```

Creates "biomodal.out" and "pinball/foo*"

```
%pin -xyzzy -reserve_memory pinball/foo.address -
replay -replay:basename pinball/foo --
$PIN_KIT/extras/pinplay/intel64/bin/nullapp
```

Creates "bimodal.out"

Using PinPoints/scripts* :Logging (uses pinplay-driver.so):

```
%$PIN_KIT/extras/pinplay/PinPoints/scripts/logger.py
--pinplayhome $PIN_KIT --mode st --logfile pinball/foo
hello

      Creates: pinball/foo_28293.*
                       foo_<pid>.*
```

```
Usage: logger.py [options] --logfile FILE application app_arguments
Version: 1.48
```

```
  -h, --help              show this help message and exit
```

```
--mode=MODE             MODE specifies the type of program to be logged. No
                        default. Must be defined in either the tracing
                        configuration file or this option.
                        st - single-threaded
                        mt - multi-threaded
                        mpi - MPI single-threaded
                        mpi_mt - MPI multi-threaded
```

** Developed by Mack Stallcup*

Optimization Notice

Using PinPoints/scripts: Replaying (uses pinplay-driver.so)

```
%$PIN_KIT/extras/pinplay/PinPoints/scripts/replayer.py
--pinplayhome $PIN_KIT -replay_file pinball/foo_28293
```

```
Usage: replayer.py [options] --replay_file=FILE
```

```
 -h, --help                show this help message and exit
```

Example: PinPoints on specrand (SPEC2006)

```
% $PIN_KIT/extras/pinplay/PinPoints/scripts/pinpoints.py -h
```

```
Usage: pinpoints.py phase [options]
```

```
  --cfg FILE, --config_file FILE
                        Give one, or more, file(s) containing the application
                        tracing parameters. Must use '--cfg' for each file.
```

```
% cat specrand.test.cfg
[Parameters]
program_name:    specrand
input_name:      test
command:         "./base.exe 1255432124 234923 > rand.234923.out"
epilog_length:   0
maxk:            5
mode:            st
num_proc:        1
prolog_length:   0
slice_size:      100000
warmup_length:   300000
```

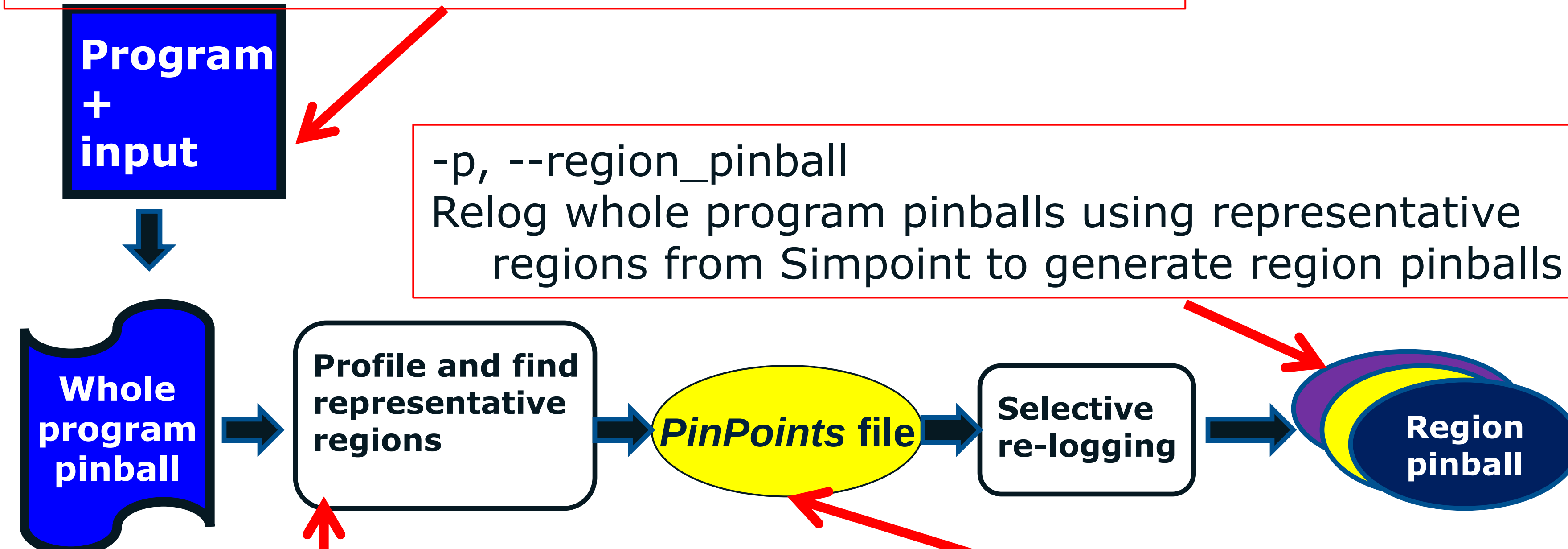
Warmup	Prolog	Simulation	Epilog

Optimization Notice

Specrand PinPoints: Basic Flow

-l, --log
Generate whole program pinballs for the application.

-p, --region_pinball
Relog whole program pinballs using representative regions from Simpoint to generate region pinballs.



-b, --basic_block_vector
Generate basic block vectors for whole program pinballs.

-s, --simpoint
Run Simpoint using whole program pinball basic block vectors.

```
% $PIN_KIT/extras/pinplay/PinPoints/scripts/pinpoints.py --cfg specrand.test.cfg
--pinplayhome=$PIN_KIT -l -b -s -p
```

Specrand PinPoints: Output

```
Script args:              --cfg specrand.test.cfg --pinplayhome=/nfs/mmdc/disks/pinplay/proj/PinPlayExternal/Wh
atIf/released/pinplay-1.0-pin-2.12-55942-gcc.4.4.7-linux -l -b -s -p
Tracing mode:              st
Program name:              specrand
Input name:                test
Command:                   "./base.exe 1255432124 234923 > rand.234923.out"
Maxk:                      20
Cutoff:                    1.0
Warmup length:             3,001,500
Prolog length:             0
Slice size (region):       1,000,000
Epilog length:             0
Pinplayhome:               /nfs/mmdc/disks/pinplay/proj/PinPlayExternal/WhatIf/released/pinplay-1.0-pin-2.12-55
942-gcc.4.4.7-linux
Whole program directory:   whole_program.test
Data/lit/pp directory:     specrand.1.test
Trace file name format:    specrand.1.test_t0rX_warmup3001500_prolog0_region1000000_epilog0
Number cores/system:       8
```

Warm-up: Extra 1500 instructions added for overlap avoidance due to basic-block level instruction counting

Trace file name format (for region pinballs):

specrand.1.test_**t0rX**_**warmup3001500**_**prolog0**_**region1000000**_**epilog0**

Warmup	**Prolog**	**Simulation**	**Epilog**

Optimization Notice

Specrand PinPoints: Results

```
999.specrand/
|-- specrand.1.test_31095.Data
|-- specrand.1.test_31095.pp
`-- whole_program.test
```

```
specrand.1.test_31095.Data
|-- create_region_file.out
|-- simpoint.out
|-- specrand.1.test_31095.T.0.bb
|-- specrand.1.test_31095.pinpoints.csv
|-- t.bb
|-- t.labels
|-- t.simpoints
`-- t.weights
```

```
whole_program.test/
|-- specrand.1.test_31095.0.dyn_text.bz2
|-- specrand.1.test_31095.0.reg.bz2
|-- specrand.1.test_31095.0.result
|-- specrand.1.test_31095.0.result_play
|-- specrand.1.test_31095.0.sel.bz2
|-- specrand.1.test_31095.address
|-- specrand.1.test_31095.log.txt
|-- specrand.1.test_31095.procinfo.xml
|-- specrand.1.test_31095.replay.txt
`-- specrand.1.test_31095.text.bz2
```

```
% du -h base.exe whole_program.test
492K    base.exe
192K    whole_program.test
```

```
% grep inscount whole_program.test/*
whole_program.test/specrand.1.test_31095.0.result:inscount: 617602978
whole_program.test/specrand.1.test_31095.0.result_play:inscount: 617602978
```

- Dynamic instruction count : 617 million
  Whole-program pinball size : 192K (base.exe 492K)

Optimization Notice

specrand.1.test_31095.pinpoints.csv

```
# Regions based on '/nfs/mmdc/disks/pinplay/proj/PinPlayExternal/WhatIf/released
/pinplay-1.0-pin-2.12-55942-gcc.4.4.7-linux/extras/pinplay/PinPoints/scripts/cre
ate_region_file.pl -seq_region_ids -tid 0 -region_file t.simpoints -weight_file
t.weights t.bb':
comment,thread-id,region-id,simulation-region-start-icount,simulation-region-end
-icount,region-weight
# Region = 1 Slice = 467 Icount = 467003527 Length = 1000005 Weight = 0.0016
cluster 0 from slice 467,0,1,467003527,468003532,0.001618

# Region = 2 Slice = 93 Icount = 93000803 Length = 1000004 Weight = 0.2006
cluster 1 from slice 93,0,2,93000803,94000807,0.200647

# Region = 3 Slice = 577 Icount = 577004084 Length = 1000007 Weight = 0.2427
cluster 2 from slice 577,0,3,577004084,578004091,0.242718

# Region = 4 Slice = 186 Icount = 186001529 Length = 1000001 Weight = 0.3447
cluster 3 from slice 186,0,4,186001529,187001530,0.344660

# Region = 5 Slice = 51 Icount = 51000481 Length = 1000006 Weight = 0.2104
cluster 4 from slice 51,0,5,51000481,52000487,0.210356

# Total instructions in 5 regions = 5000023.
# Total instructions in workload = 617602969.
# Total slices in workload = 618.
# Overall dynamic coverage of workload by these regions = 1.0000 (including lost
 instrs).
```

5 PinPoints → 5 Region Pinballs

```
% tree specrand.1.test_31095.pp | grep address
|-- specrand.1.test_31095_t0r1_warmup3001500_prolog0_region1000005_epilog0_001_0-00161.0.address
|-- specrand.1.test_31095_t0r2_warmup3001500_prolog0_region1000004_epilog0_002_0-20064.0.address
|-- specrand.1.test_31095_t0r3_warmup3001500_prolog0_region1000007_epilog0_003_0-24271.0.address
|-- specrand.1.test_31095_t0r4_warmup3001500_prolog0_region1000001_epilog0_004_0-34466.0.address
|-- specrand.1.test_31095_t0r5_warmup3001500_prolog0_region1000006_epilog0_005_0-21035.0.address
```

```
Warmup length:          3,001,500
Prolog length:          0
Slice size (region):    1,000,000
Epilog length:          0
```

Warm-up: Extra 1500 instructions

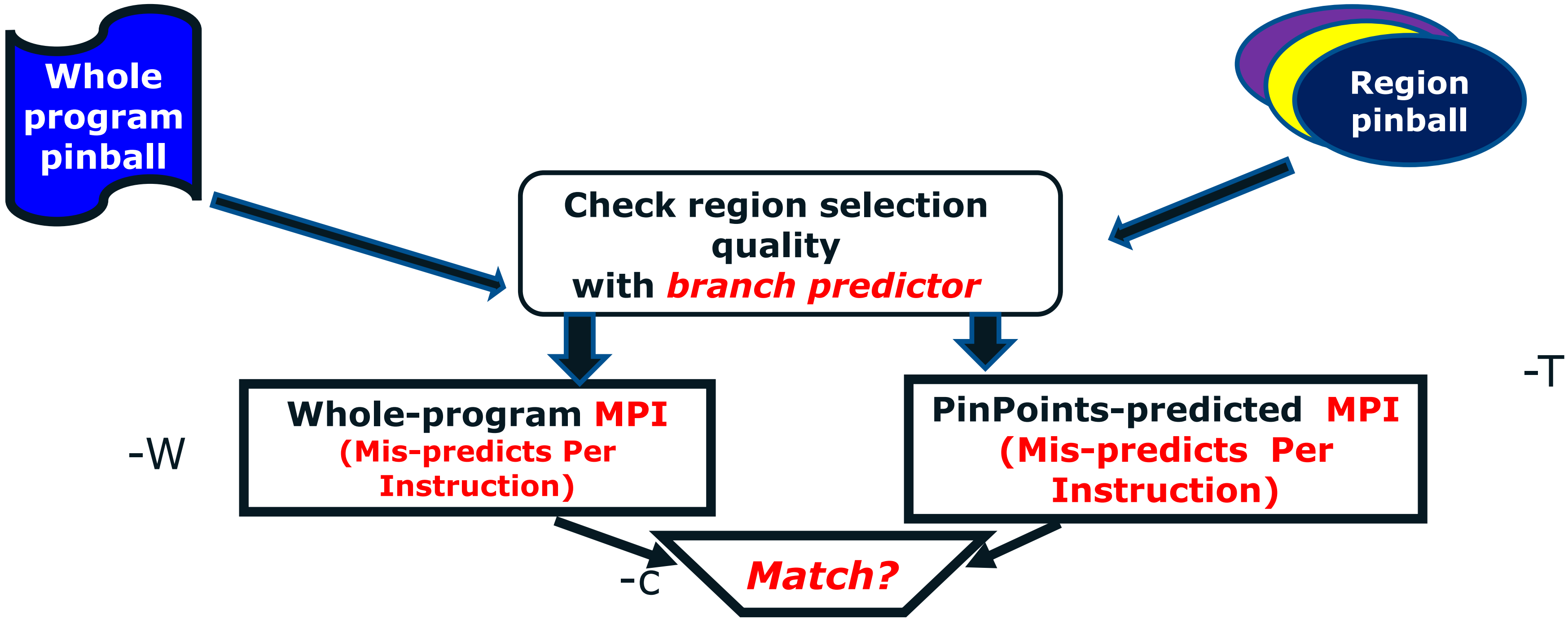
Trace file name format (for region pinballs):

specrand.1.test_**t0rX**_**warmup3001500**_**prolog0**_**region1000000**_**epilog0**

Warmup	Prolog	Simulation	Epilog

```
% grep inscount specrand.1.test_31095_t0r*.0.result
specrand.1.test_31095_t0r1_warmup3001500_prolog0_region1000005_epilog0_001_0-00161.0.result:inscount: 4001505
specrand.1.test_31095_t0r2_warmup3001500_prolog0_region1000004_epilog0_002_0-20064.0.result:inscount: 4001490
specrand.1.test_31095_t0r3_warmup3001500_prolog0_region1000007_epilog0_003_0-24271.0.result:inscount: 4001500
specrand.1.test_31095_t0r4_warmup3001500_prolog0_region1000001_epilog0_004_0-34466.0.result:inscount: 4001499
specrand.1.test_31095_t0r5_warmup3001500_prolog0_region1000006_epilog0_005_0-21035.0.result:inscount: 4001504
```

Functional Correlation with brpred_pinpoints.py

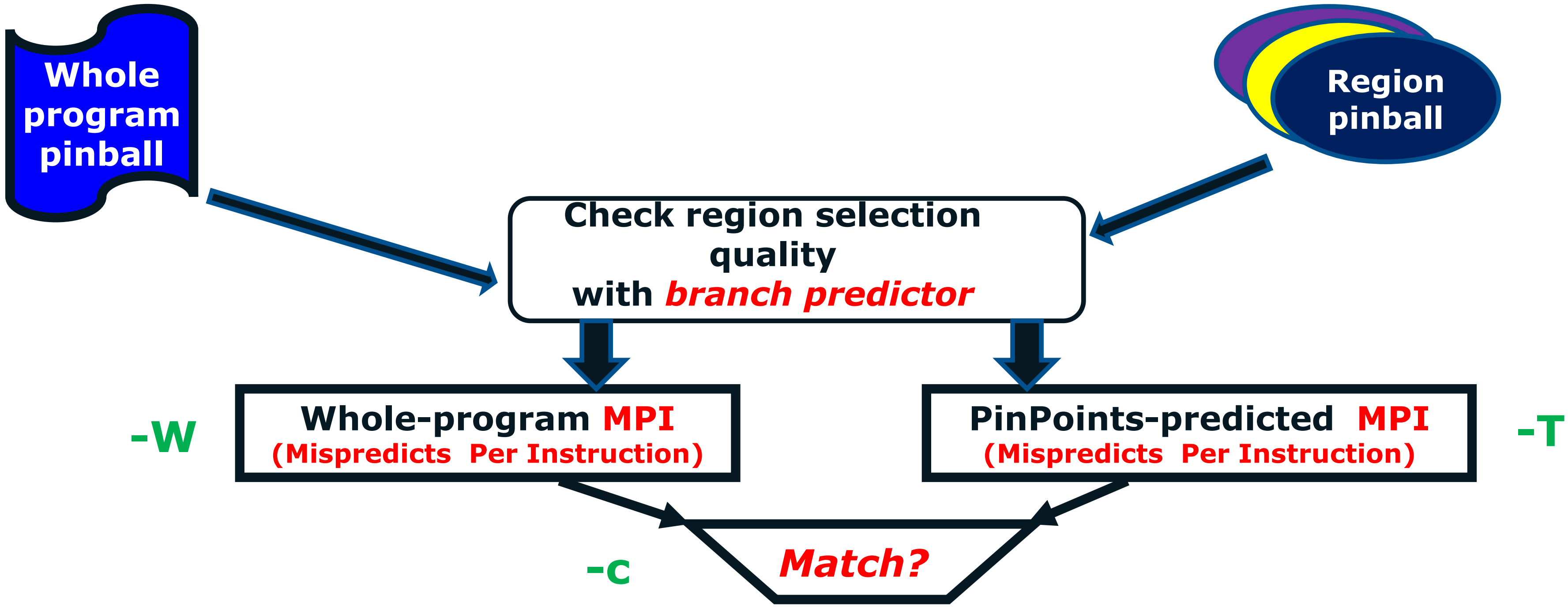



```
% $PIN_KIT/extras/pinplay/PinPoints/scripts/brpred_pinpoints.py -h
Usage: brpred_pinpoints.py phase [options]
```

```
  -T, --region_sim     Run the simulator on the whole program pinballs.
  -W, --whole_sim      Run the simulator on the region pinballs.
  -c, --calc_func      Calculate the functional correlation, using the metric
                       of interest, for a set of representative regions.
                       Must have already generated simulator data, either
                       using phases '--whole_sim' and '--region_sim' or the
                       appropriate options for your simulator,  before
                       running this phase.
```


Optimization Notice

Specrand PinPoints : *Functional Correlation* with pinplay-branch-predictor.so




```
% $PIN_KIT/extras/pinplay/PinPoints/scripts/brpred_pinpoints.py --cfg specrand.test.cfg --pinplayhome=$PIN_KIT
 -W -T -c
```

```
***  Calculating functional correlation  ***    February 18, 2013 20:55:17

PID: 31095
  Predicted MPI:             6.952073
  Measured MPI:              6.969056
  Functional correlation:    0.997563 (p/m)
```

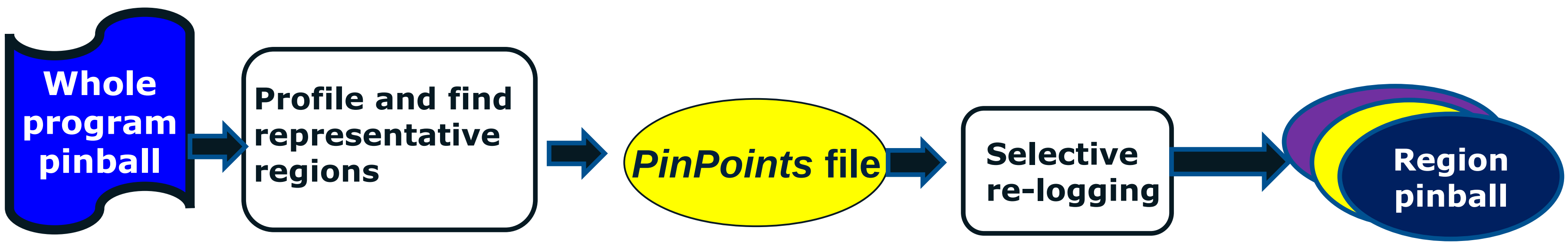
Tuning PinPoints Selection: bzip2: input “chicken”

Initial results (maxK=5): Error 31%

Re-generated PinPoints with maxK=20

```
% $PIN_KIT/extras/pinplay/PinPoints/scripts/pinpoints.py --pinplayhome=$PIN_KIT -b -s
-p --cfg bzip2.chicken.cfg
```

```
maxk:            20
```



-b, --basic_block_vector

-s, --simpoint

-p, --region_pinball

Tuned results (maxK=20) : Error 2.7%

```
% $PIN_KIT/extras/pinplay/PinPoints/scripts/brpred_metric.py --pinplayhome=$PIN_KIT
 --actual=whole_program.chicken/ --predicted=bzip2.1.chicken_9999.pp --calc_func
Predicted metric:           6.8173
Actual metric:              7.0338
Functional correlation:   0.969226 (p/a)
```


Optimization Notice

Logging a Specific Region

Use "PinPlay Controller" knobs:

```
% $PIN_KIT/pin -t $PIN_KIT/extras/pinplay/bin/intel64/pinplay-driver.so -help -- /bin/true
```

Start of region:

```
-log:skip  [default ]
          Number of instructions to skip from beginning
```

```
-log:start_address
          Address and count to trigger a start (e.g. 0x400000, main, memcpy:2,
          /lib/tls/libc.so.6+0x1563a:1)
```

End of region:

```
-log:stop_address
          Address and count to trigger a stop (e.g. 0x400000, main, memcpy:2,
          /lib/tls/libc.so.6+0x1563a:1)
```

```
-log:length  [default ]
          Number of instructions to execute before stopping
```

Uniform sampling

```
-log:uniform_count  [default ]
          Number of uniform samples to trigger.
-log:uniform_length  [default ]
          Number of instructions to capture periodically
-log:uniform_period  [default ]
          Number of instructions to skip periodically
-log:uniform_skip  [default ]
          Number of skip before uniform sampling starts.
```

Example: Logging a Specific Region

Skip 5000 instructions, log 1000, and then exit:

```
% $PIN_KIT/extras/pinplay/PinPoints/scripts/logger.py --pinplayhome=$PIN_KIT --mode st --logfile
pinball/foo --log_options="-log:skip 5000 -log:length 1000 -log:early_out" -- /bin/ls
```

```
% grep inscount pinball/*
pinball/foo_589.0.result:inscount: 996
```

pinball/*.log.txt:

```
[0] region-start
[0] + RECEIVED AND PROCESSED START event: ip:0x2aaaaaabd290 mcount: 0 icount: 0 time:Tue Feb 19 13:10:13 2013
[0]  Region# : 0
[0] Region has 1 existing threads(s)
[0] InitRegion called by thread: 0
[0] First thread in the region
[0] region-end
[0] + RECEIVED STOP event: mcount: 345 icount: 986 time:Tue Feb 19 13:10:13 2013
[0]  getting ready to terminate due to : log:early_out 345
[0] TerminateTrace: 348 0x2aaaaaaadef5
[0] EndRegion called by: 0 in_region: 1
[0] Exited_thread_count: 0
[0] Last thread, writting global files
[0] + FINISHED PROCESSING STOP event: mcount: 348 icount: 996
[0] Exiting due to log:early_out ic: 996 mc: 348
```

Sniper Overview

ExaScience Lab
Intel Labs Europe
EXASCALE COMPUTING

The Sniper Multi-Core Simulator

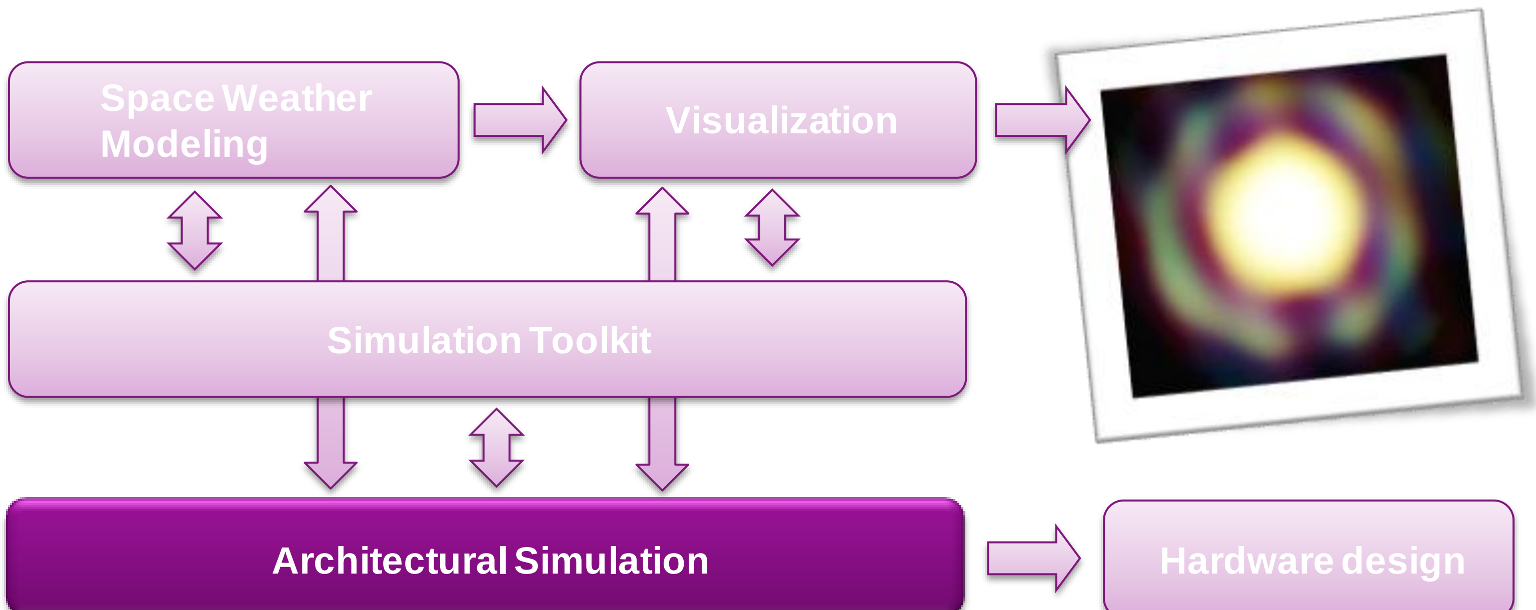
Trevor E. Carlson, Wim Heirman, Ibrahim Hur
Kenzo Van Craeynest and Lieven Eeckhout

http://www.snipersim.org
Saturday, February 23, 2013
HPCA 2013, Shenzhen, China

Universiteit Gent
intel

Intel ExaScience Lab

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



SIMULATION

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

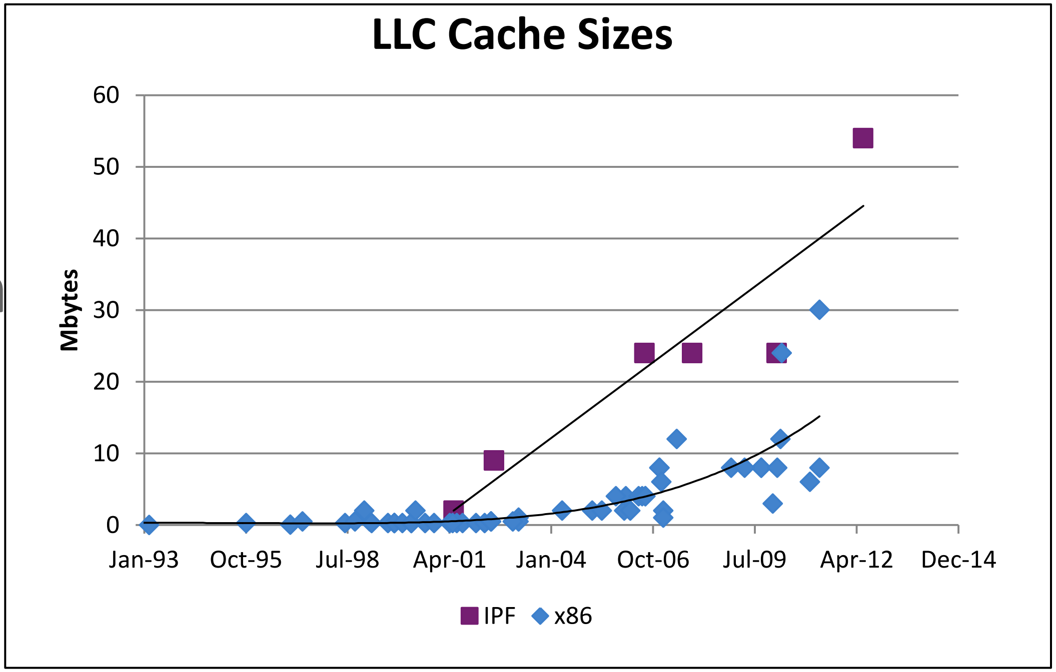
DEMANDS ON SIMULATION ARE INCREASING

- Increasing core counts
  - Hardware today is seeing large core-counts
    - 2011: 10-core Intel Xeon Westmere-EX
    - 2012: Intel MIC Knights Corner (60+ cores)
  - 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 app behavior
  - Long-running sim



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

Fast and Accurate Simulation is Needed

- Simulation use cases
  - Architecture exploration
  - Pre-silicon software optimization
  - [Validation]
- Cycle-accurate simulation is too slow for exploring multi/many-core design space and software
- Key questions
  - Can we raise the level of abstraction?
  - What is the right level of abstraction?
  - When to use these abstraction models?

Experiment Design in Architecture Exploration/Evaluation

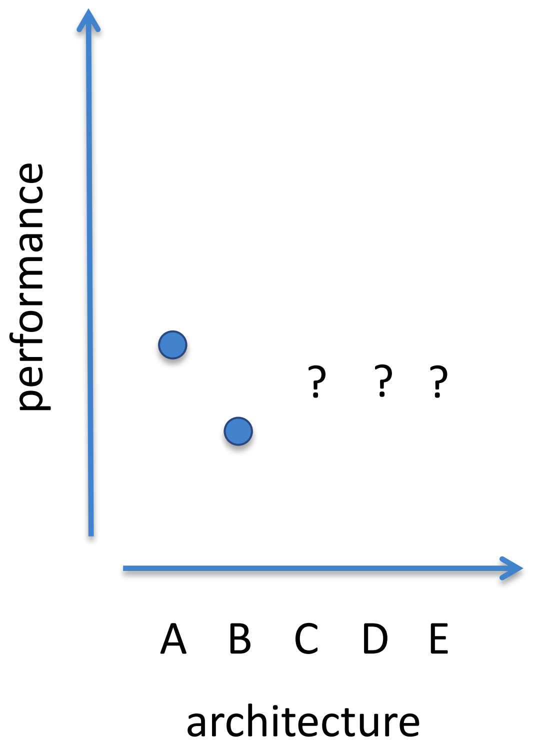
- Optimizing the probability of success (i.e., finding the best architecture/parameters):
  - Coverage: how many architecture configurations can I run
  - Confidence: # benchmarks, re-runs for variable applications
  - Accuracy: simulation model detail vs. runtime
- How many scenarios can I run?
  - N = total number of simulation scenarios
  - d = days until paper deadline
  - t = average time per simulation
  - B = number of benchmarks
  - A = number of architectures

$$N = \frac{d}{t \times B \times A}$$

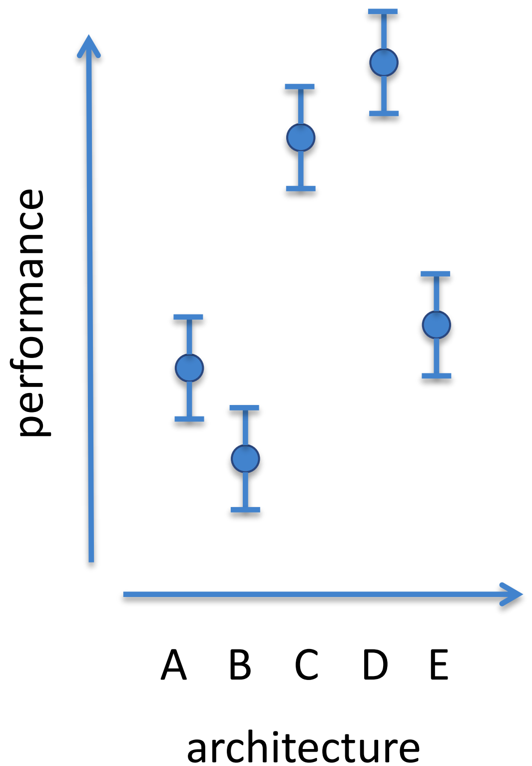
minimize *t* to maximize *N*

FAST OR ACCURATE SIMULATION?

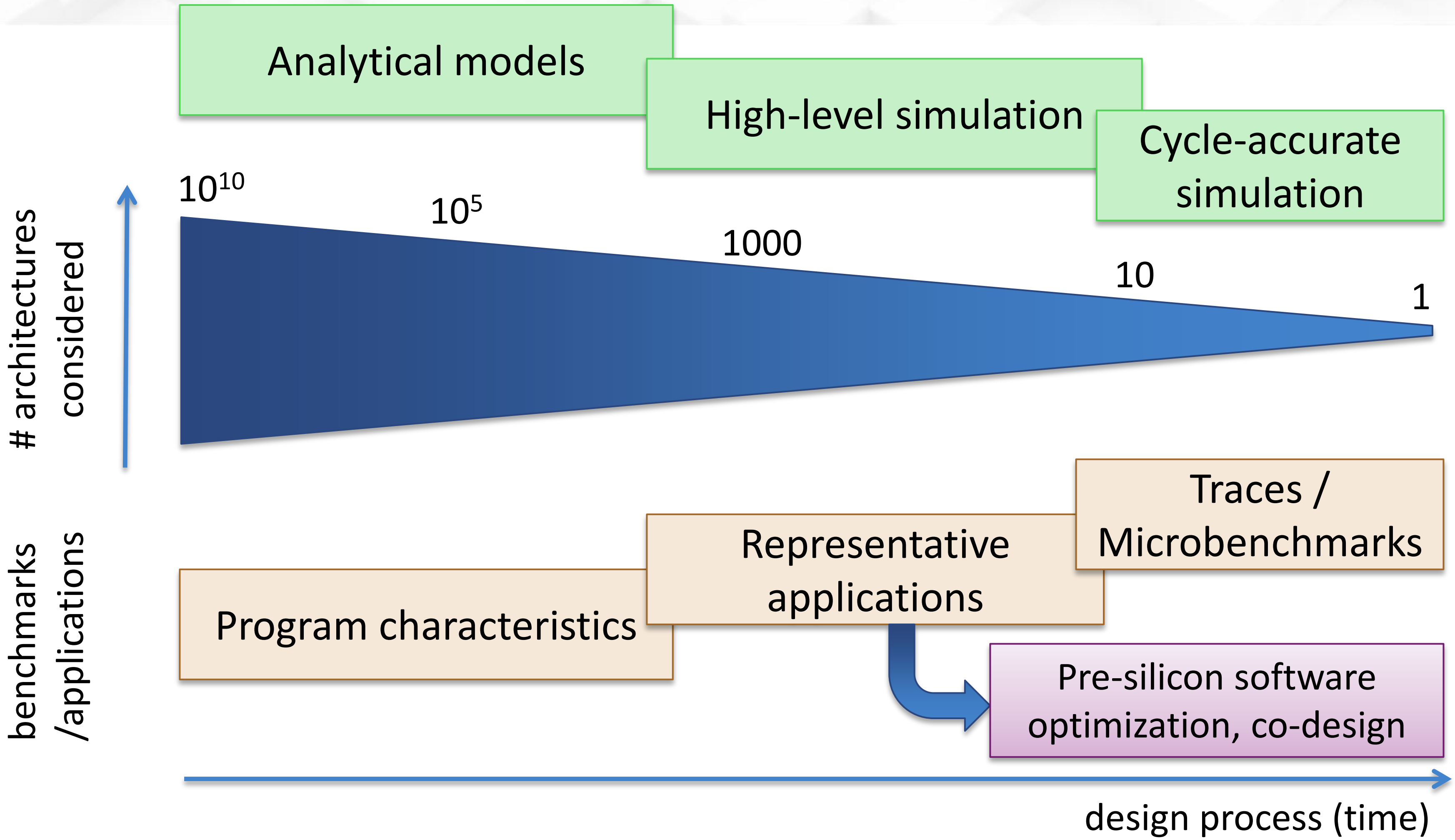
Cycle-accurate simulator



Higher-abstraction level simulator

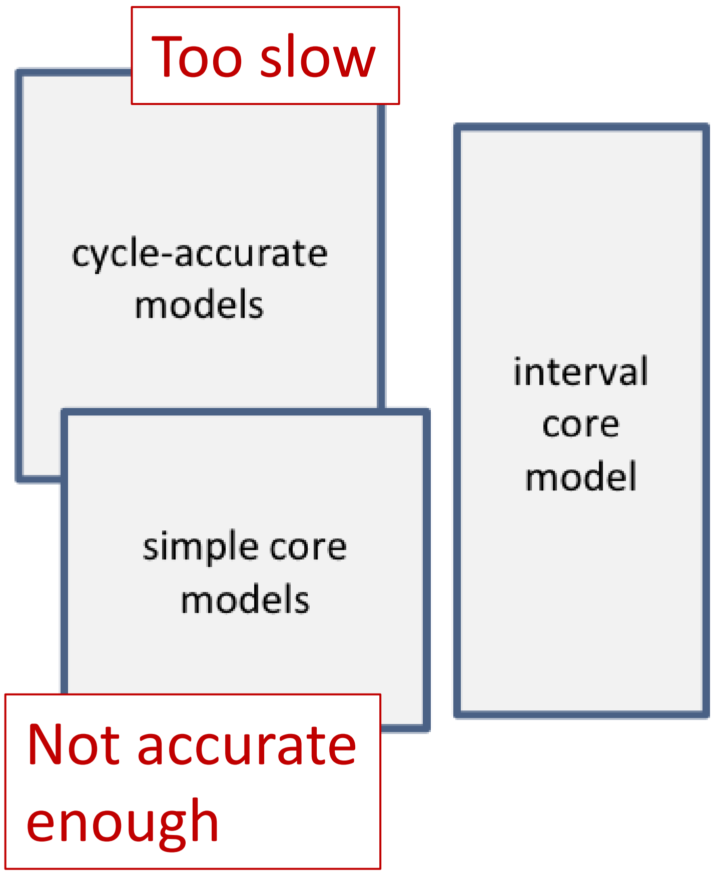


The Architecture Design Waterfall



NEEDED DETAIL DEPENDS ON FOCUS

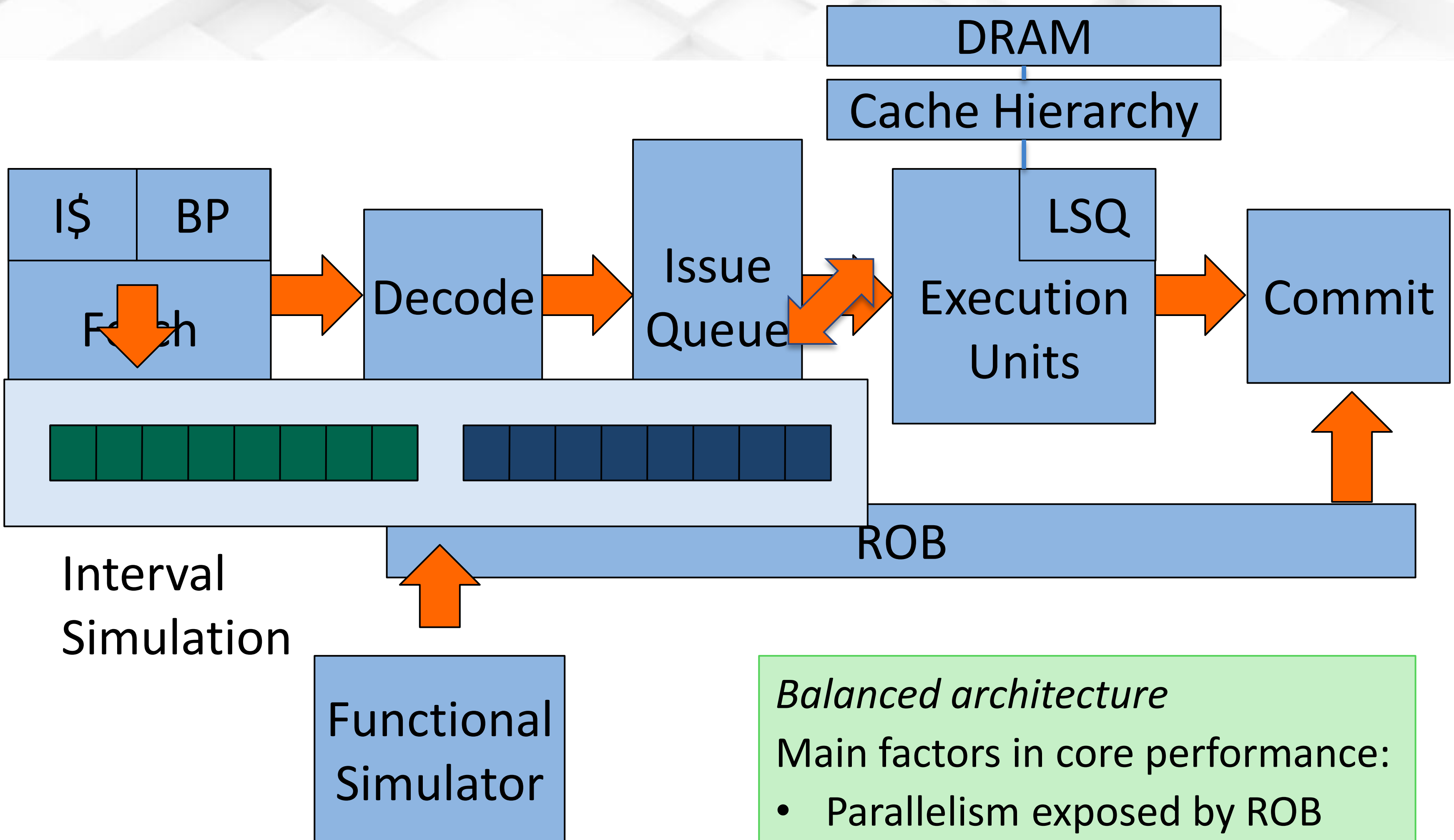
Component	Single-event time scale	Required sim time
RTL	single clock cycle	millions of cycles
OOO execution		
Core memory ops		
L1 cache access		
LLC access		
Off-socket	microseconds	seconds



SNIPER: A FAST AND ACCURATE SIMULATOR

- Hybrid simulation approach
  - Analytical interval core model
  - Micro-architecture structure simulation
    - branch predictors, caches (incl. coherency), NoC, etc.
- Hardware-validated, Pin-based
- Models multi/many-cores running multi-threaded and multi-program workloads
- Parallel simulator scales with the number of simulated cores
- Available at `http://snipersim.org`

Detailed Model vs. Interval Sim



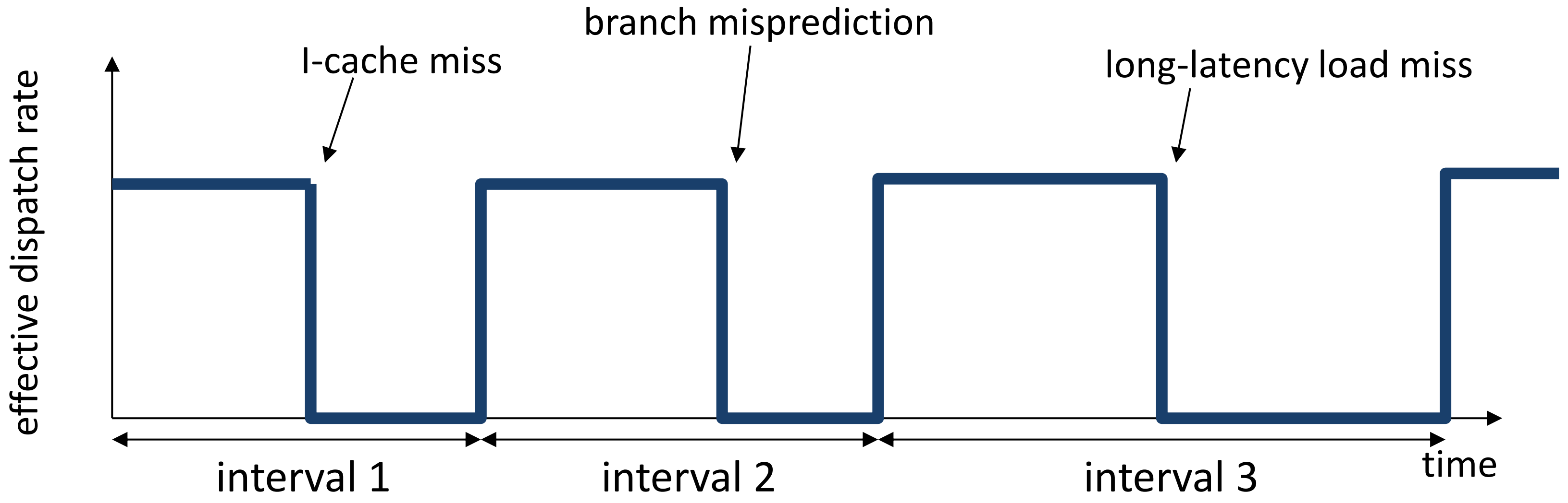
Balanced architecture

Main factors in core performance:

- Parallelism exposed by ROB
- Miss events

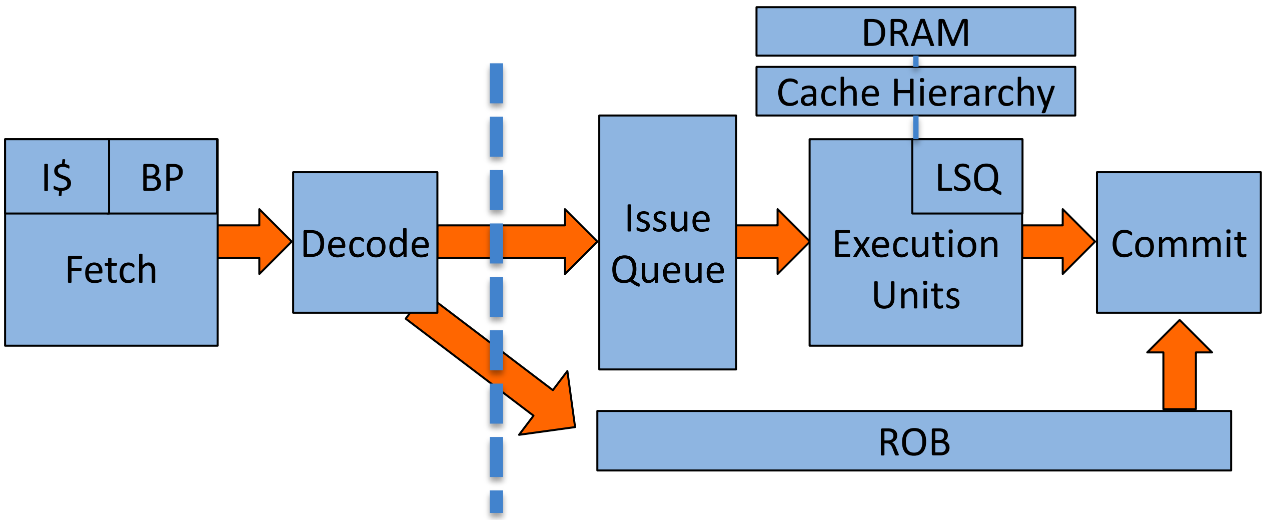
INTERVAL SIMULATION

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



D. Genbrugge et al., HPCA'10
S. Eyerman et al., ACM TOCS, May 2009
T. Karkhanis and J. E. Smith, ISCA'04, ISCA'07

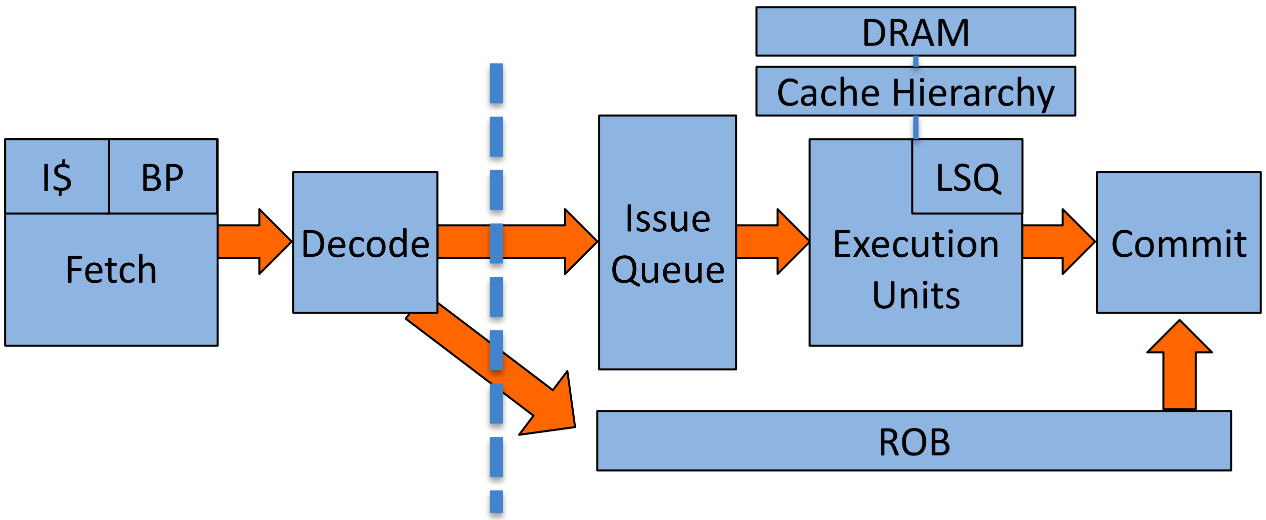
INTERVAL SIMULATION FROM 30,000 FEET



Interval simulation considers instructions (in-order) at *dispatch*

- dispatch not possible
  - Instruction cache / TLB miss
  - Branch misprediction (not dispatching *useful* instructions)
  - Front-end refill after misprediction
  - ROB full: long-latency miss at head of ROB

INTERVAL SIMULATION FROM 30,000 FEET

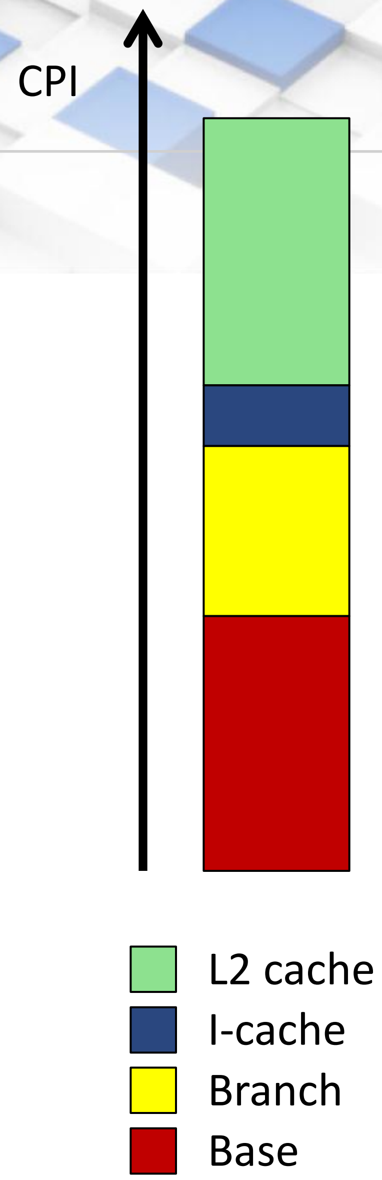


Interval simulation considers instructions (in-order) at *dispatch*

- dispatch not possible
- dispatch possible: at rate governed by ROB
  - Little’s law: progress rate = #elements / time spent in queue
  - Computed using ROB fill and critical path through ROB
    - Computed using dynamic instruction dependencies and latencies

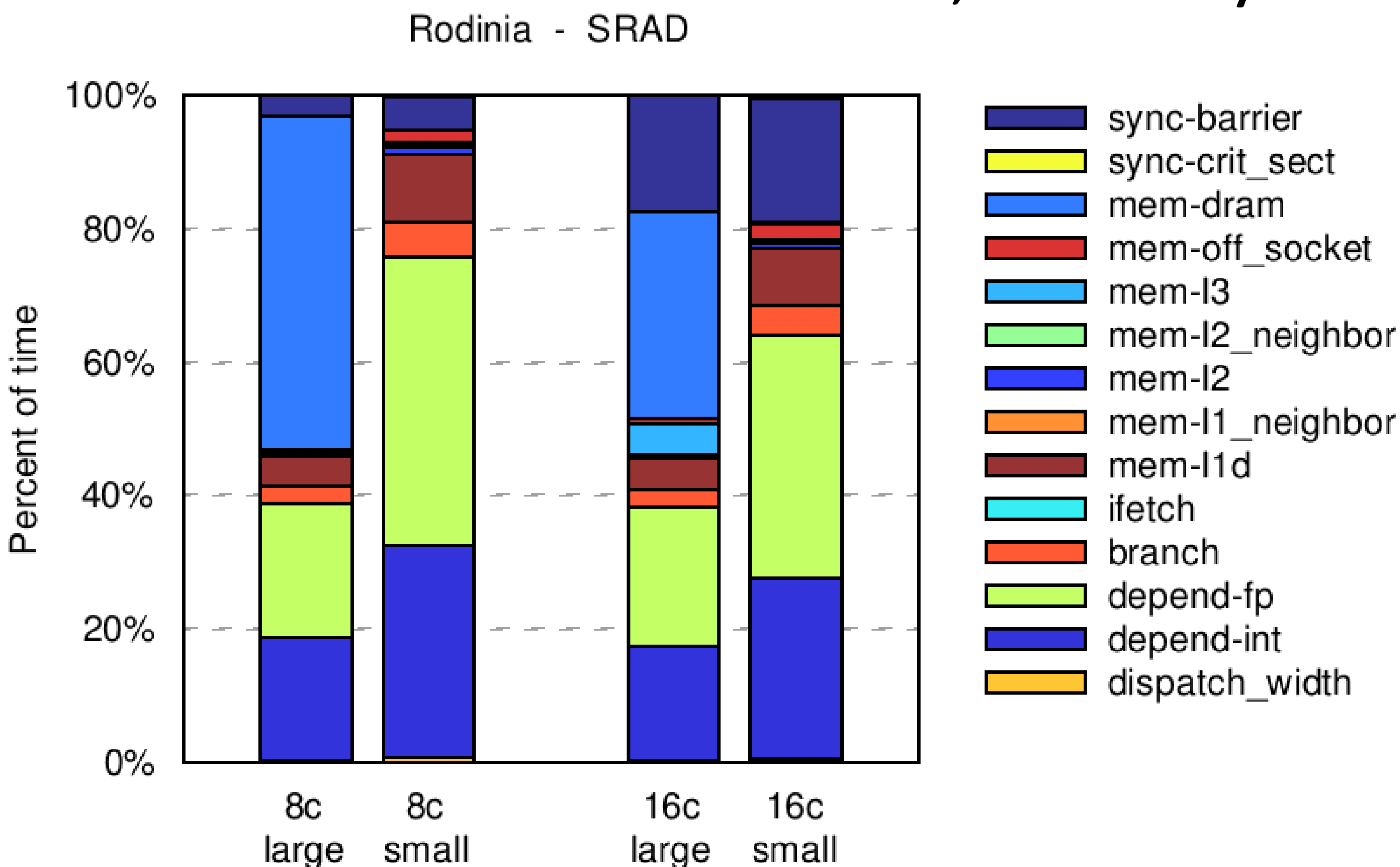
Cycle stacks

- 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

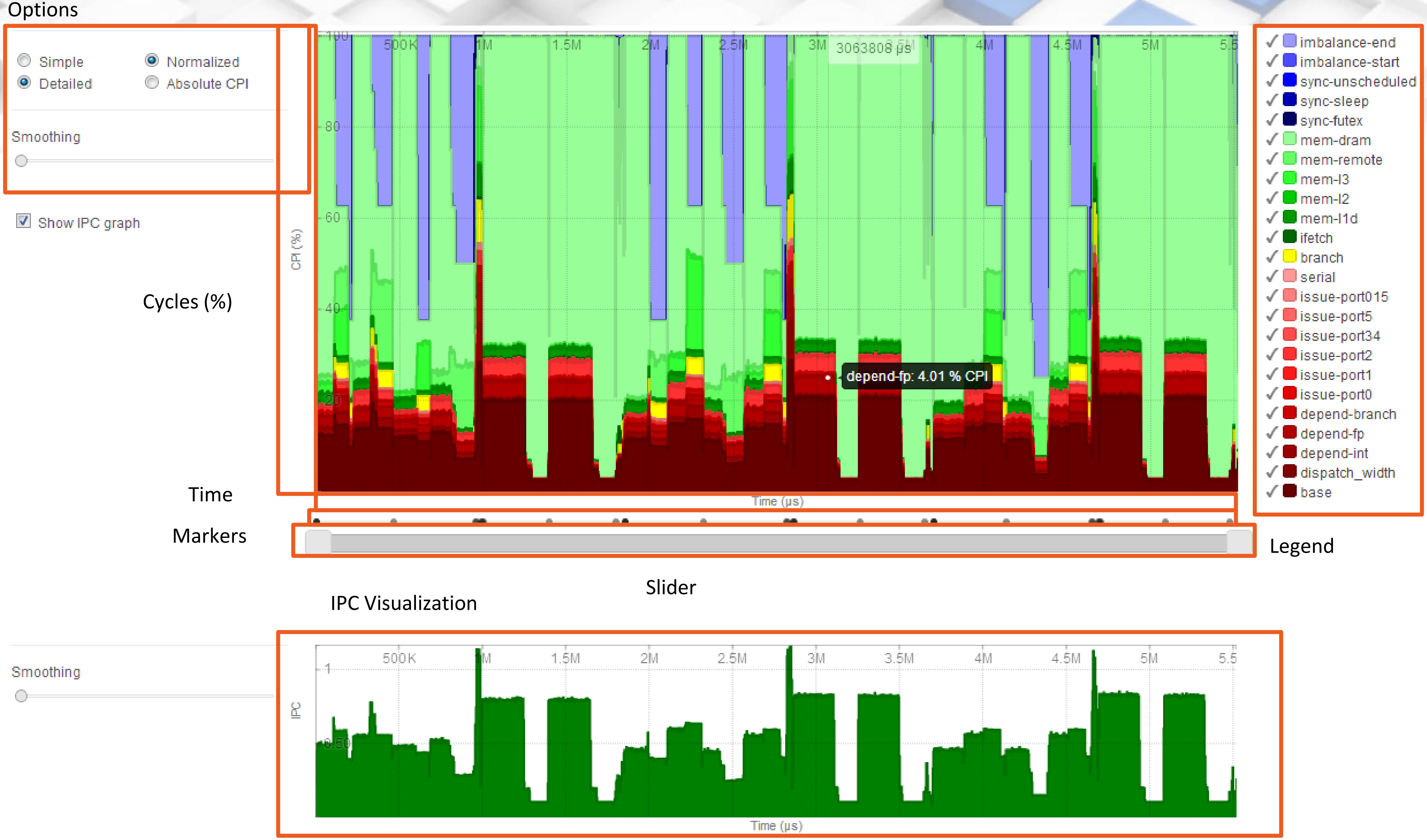


CYCLE STACKS AND SCALING BEHAVIOR

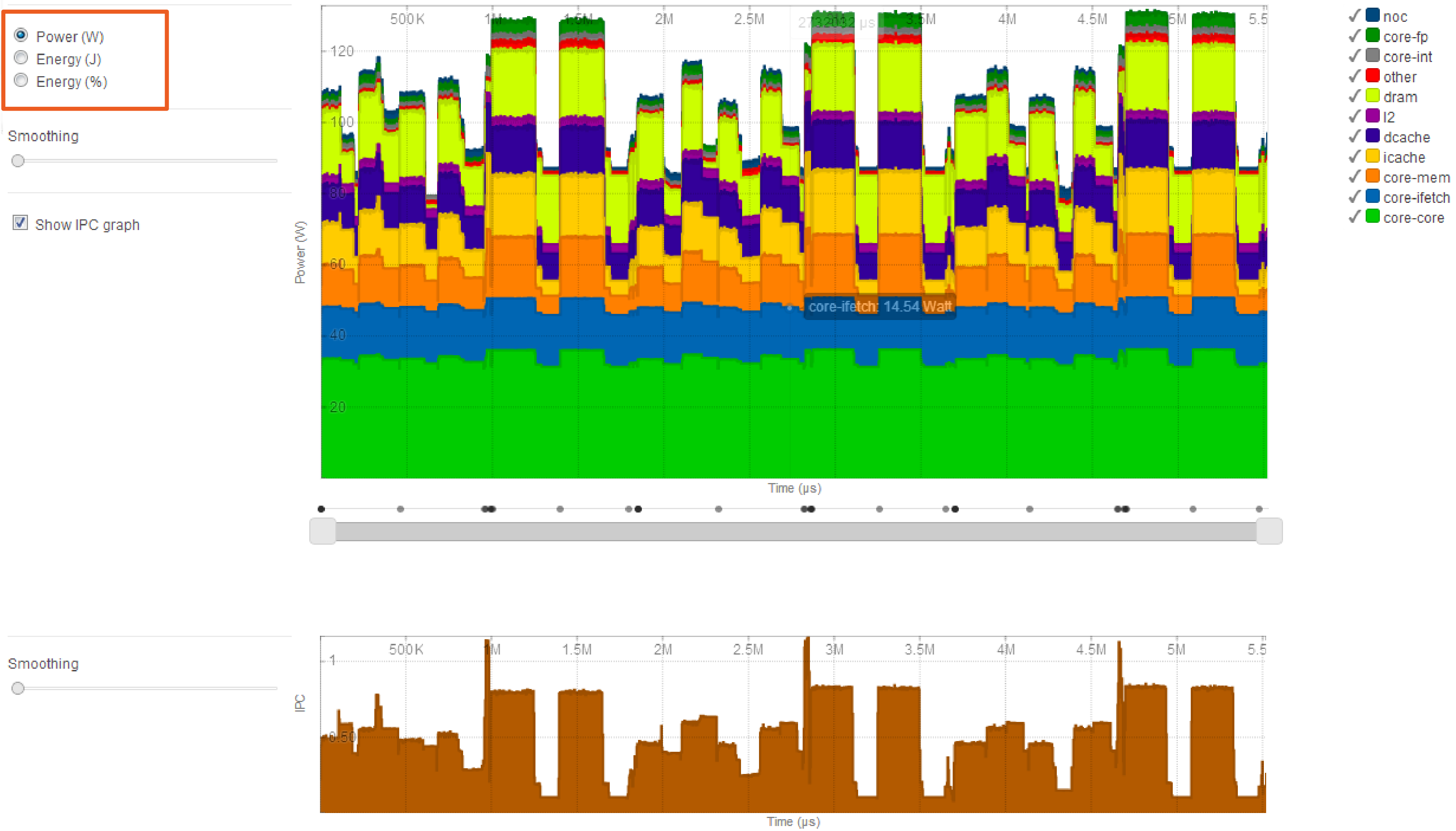
- Scaling to more cores, larger input set size
- How does execution time scale, and why?



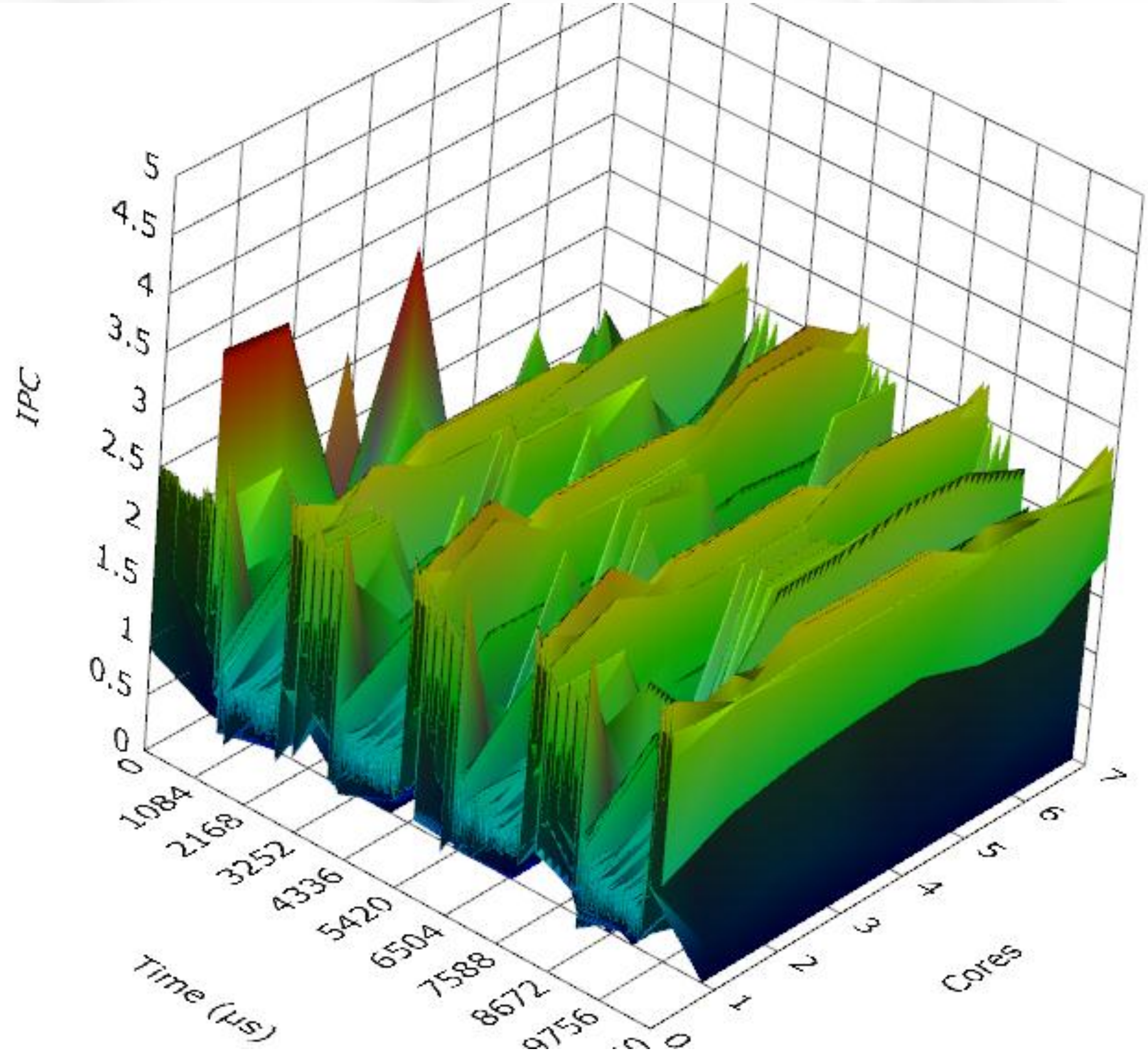
Viz: Cycles Stacks in Time



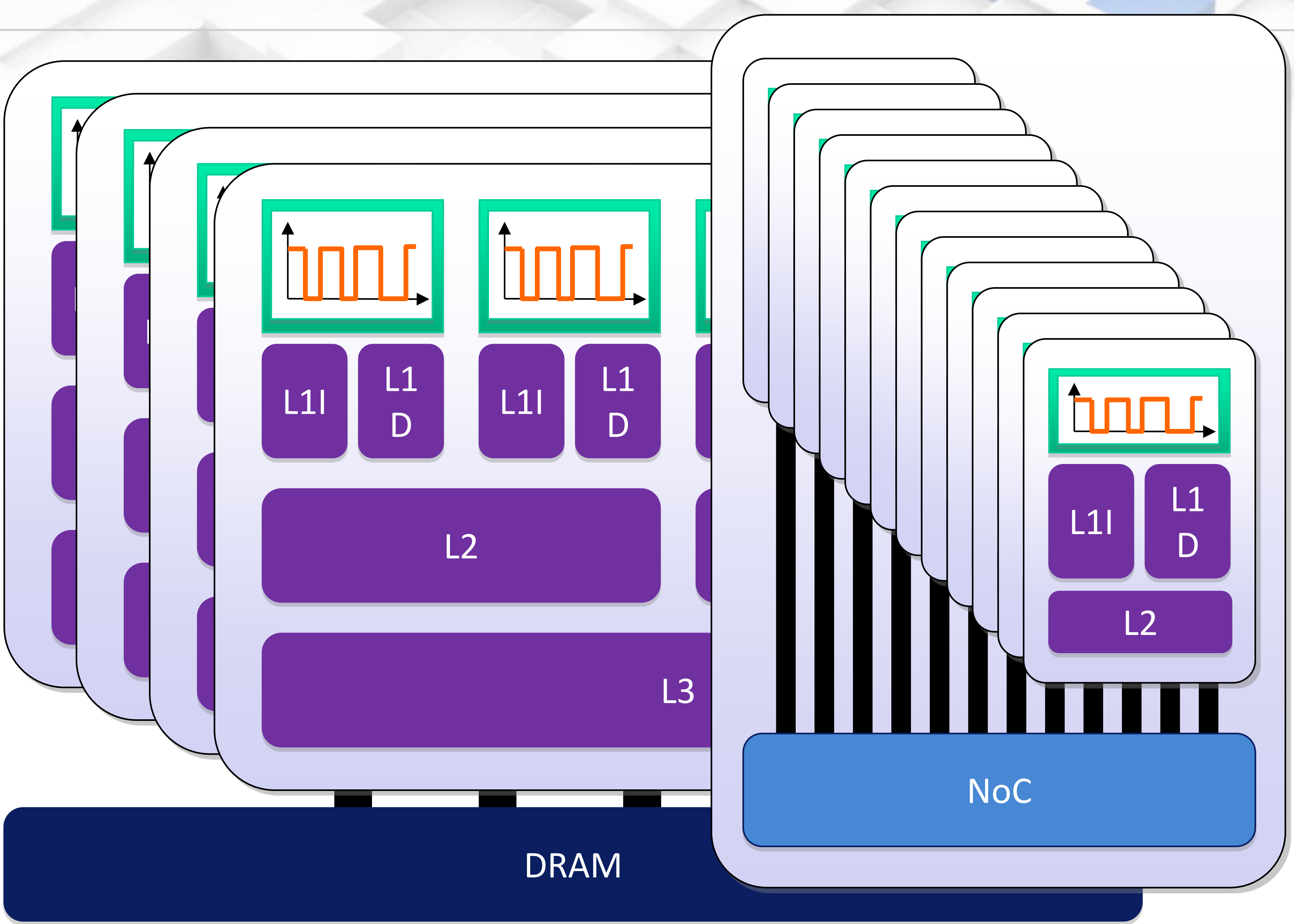
Viz: McPAT Output over Time



3D Visualization: IPC vs. Time vs. Core



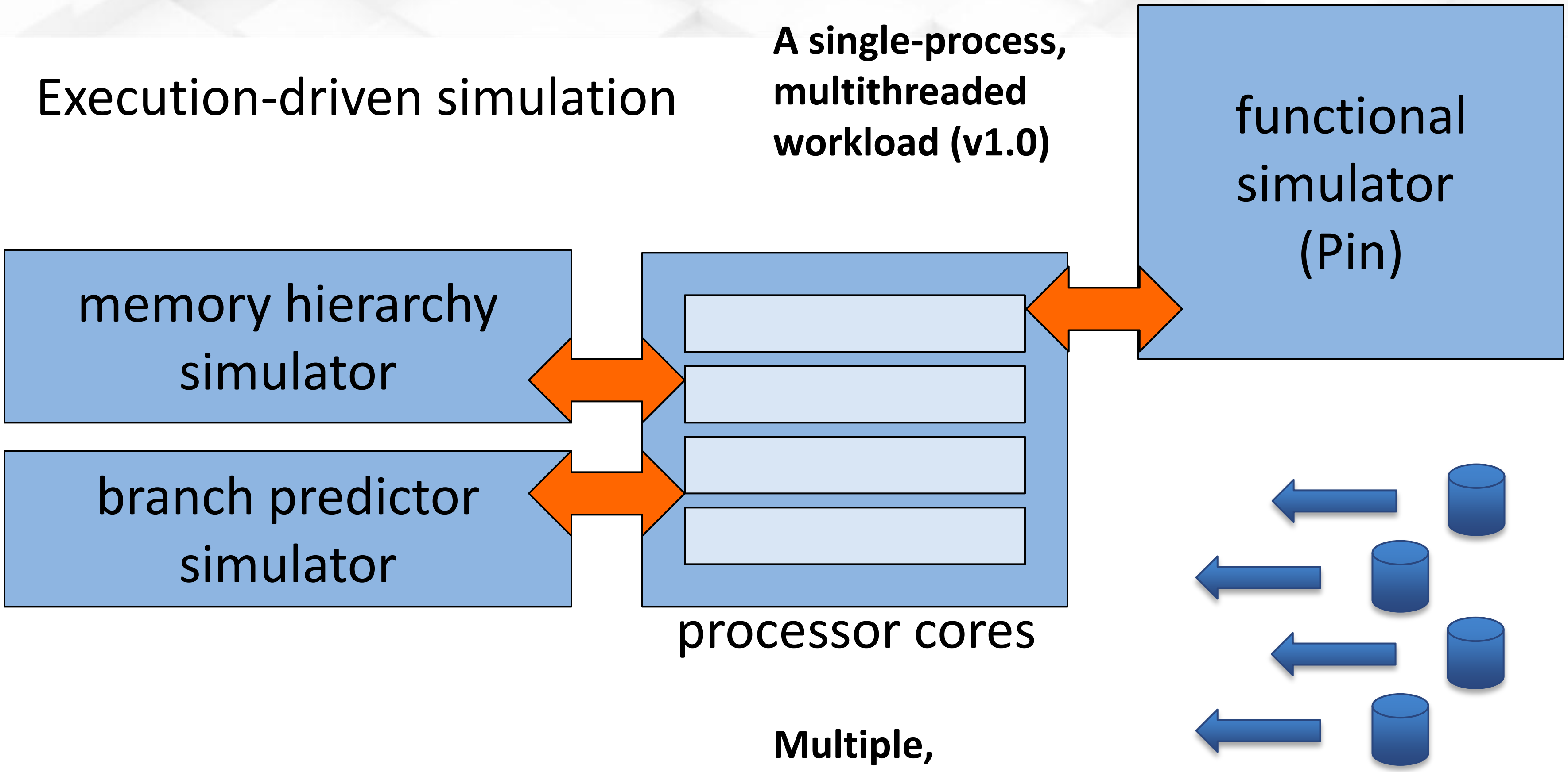
Many Architecture Options



Simulation in Sniper

Execution-driven simulation

A single-process, multithreaded workload (v1.0)

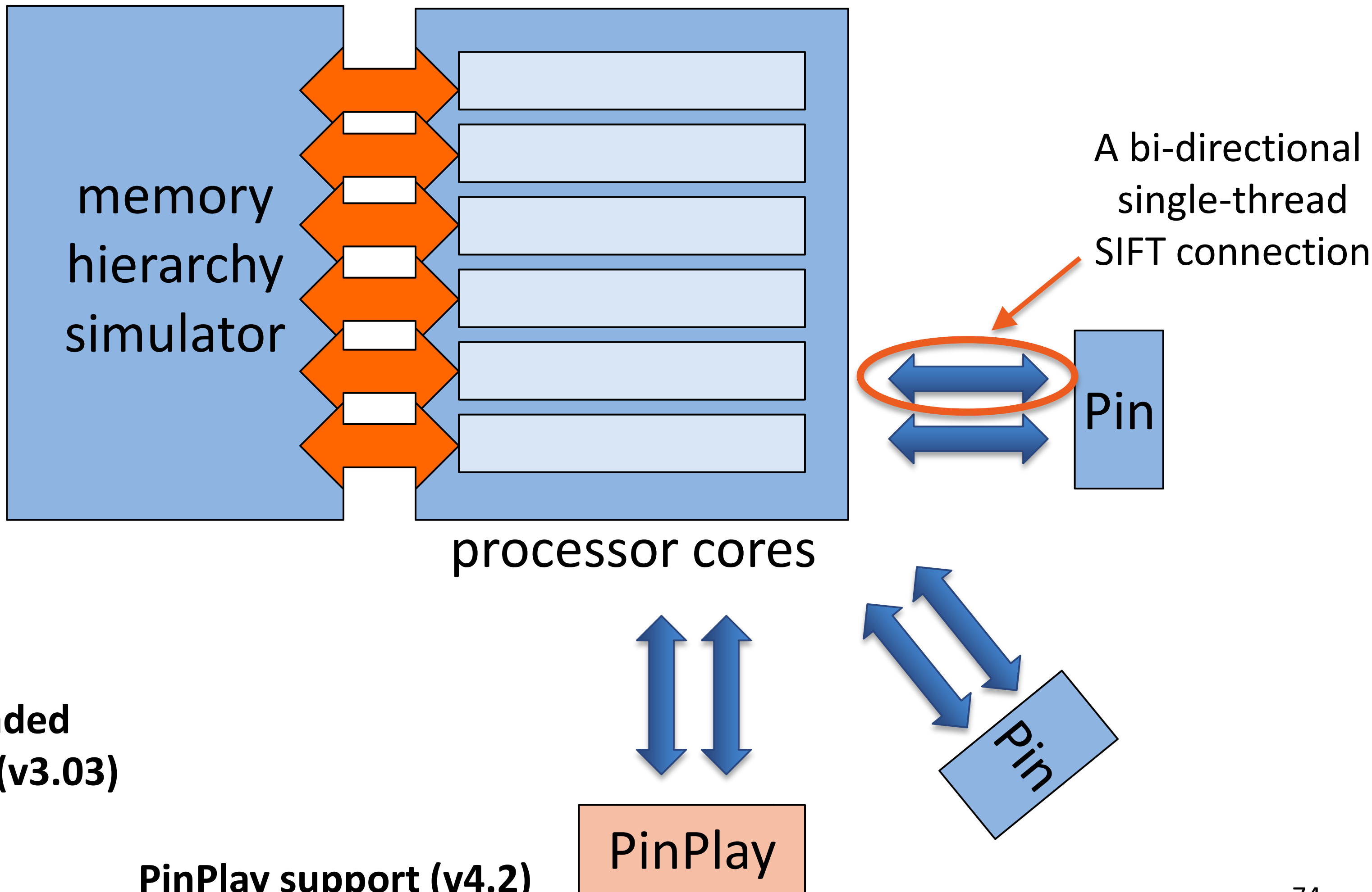


Trace-driven simulation

Multiple, single-threaded workloads (v2.0)

Simulation in Sniper with SIFT

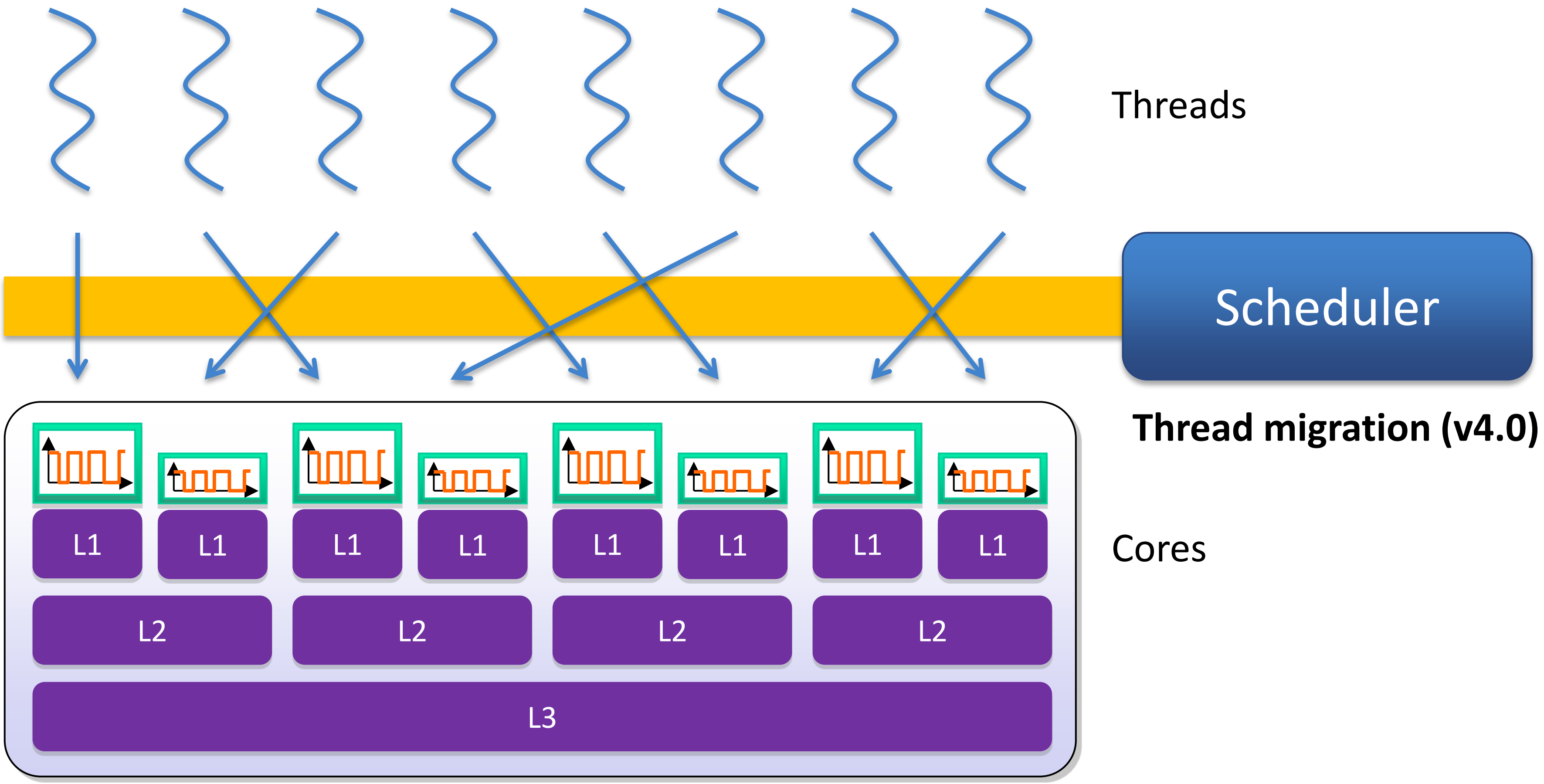
Functional-directed simulation + timing-feedback



Multiple, multi-threaded workloads (v3.03)

PinPlay support (v4.2)

THREAD SCHEDULING AND MIGRATION



Top Sniper Features

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

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 only

SNIPER HISTORY

- November, 2011: SC’11 paper, first public release
- May 2012, version 3.0: Heterogeneous architectures
- November 2012, version 4.0: Thread scheduling and migration
- December 2012, version 4.1: Visualization (2D and 3D)
- February 2012, version 4.2: PinPlay support
- Today: 300+ downloads from 45 countries

Sniper PinPlay Integration

Sniper PinPlay Integration

- Integration of PinPlay with Sniper was a perfect match
  - Since v2.0, Sniper has supported multi-program execution
  - We support functional-directed simulation
    - Pin generates an instruction stream
    - Sniper consumes that stream, but doesn't allow Pin to get to far ahead
    - Critical synchronization instructions halt functional execution
  - The SIFT format stores dynamic instruction trace information
    - Trace sizes are O(instruction count)
    - But Pinball data is executed directly; pinball sizes can be quite small O(binary-size+input-size)
    - 100's MBs to GBs for a SIFT trace can result in just MBs for a pinball
- Goal to Integrate PinPlay into Sniper
  - Code to insert is straight-forward
  - But, we ran into a few problems along the way

Optimization Notice

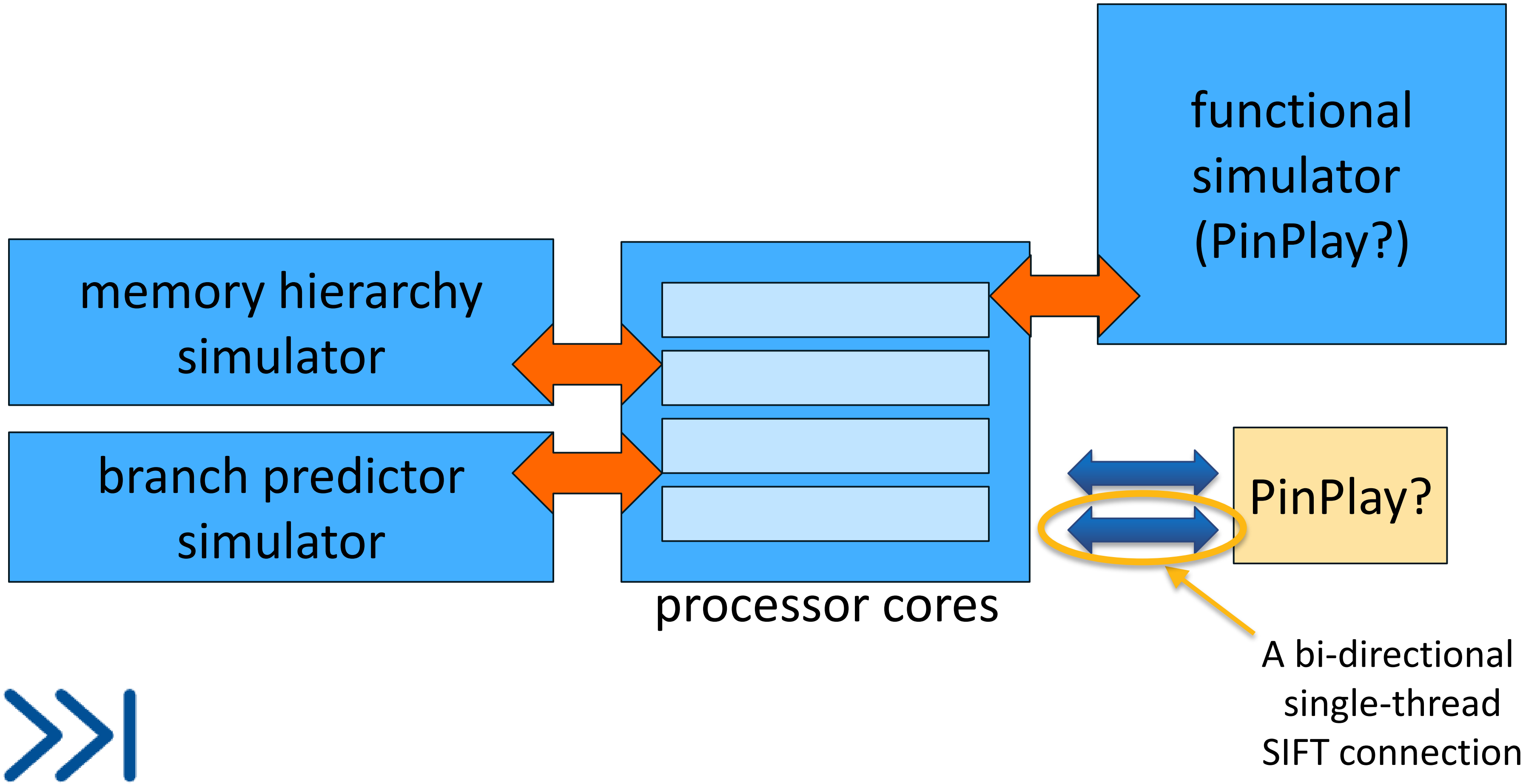
Sniper PinPlay Integration

- Would like to integrate PinPlay into Sniper
  - Code to insert is straight-forward
  - Initial goal was to integrate PinPlay into Sniper directly

```
#ifdef PINPLAY_SUPPORTED
# include "pinplay.H"
PINPLAY_ENGINE pinplay_engine;
#endif

bool pinplay_enabled = cfg->getBool("general/enable_pinplay");
#ifdef PINPLAY_SUPPORTED
   if (pinplay_enabled) {
      pinplay_engine.Activate(argc, argv, false /*logger*/, true /*replayer*/);
   }
#else
   if (pinplay_enabled) {
      LOG_PRINT_ERROR("PinPlay support not compiled in. Please use a compatible pin kit when compiling.");
   }
#endif
```

Initial Sniper PinPlay Integration



Sniper Integration Issue

- Sniper Pintool is slightly more complex than a typical Pintool
  - Per-basic-block callback with decoded instructions
    - Static information: list of micro-ops, loads, stores, memory accesses
    - Dynamic information: branch taken?, memory addresses, conditional execution status
  - Sniper also tries to control the state of running threads
    - Syscall interception and updates
    - gettimeofday() (returns simulated time)
    - get_nprocs() (returns number of simulated processors)
- Controlling the state of the thread is an issue with PinPlay

PinPlay Controls Execution

- Sniper would like to control execution
- PinPlay would like to control execution

```
$ ./run-sniper --pinballs=pin_kit/extras/pinplay/examples/pinball/foo
[SNIPER] Start
Running ['bash', '-c', ...]
[SNIPER] Enabling performance models
[SNIPER] Setting instrumentation mode to DETAILED

A:restore_mem.cpp:RestorePage:RestorePage:223:
  Entry is not a page for thread 0 @icount: 5720 @mcount: 1635
##############################################################################
## STACK TRACE
##############################################################################
...
Pin 2.12
Copyright (c) 2003-2012, Intel Corporation. All rights reserved.
...
Pin app terminated abnormally due to signal 6.
[SNIPER] End
[SNIPER] Elapsed time: 1.53 seconds
```

PinPlay Controls Execution

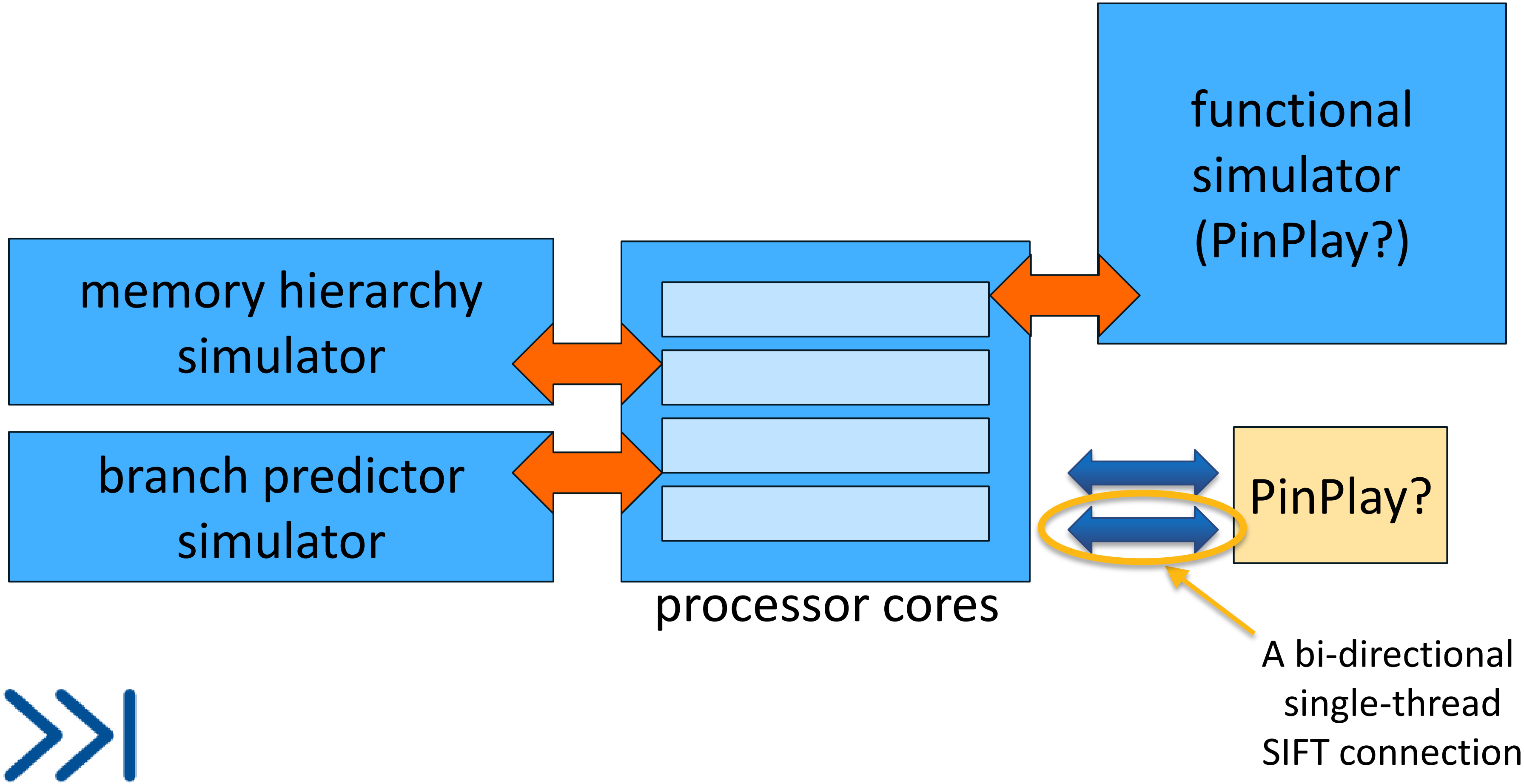
A:restore_mem.cpp:RestorePage:RestorePage:223:
Entry is not a page for thread 0 @icount: 5720 @mcount: 1635

- This is a PinPlay memory lookup error
- PinPlay needs to maintain control of the execution state of the application
  - The Pinball only contains memory, instruction data and syscalls that have been accessed during initial execution
  - Other data is not available, and therefore, changes in execution will cause PinPlay to return an error

Sniper Integration

- The end result took a different approach
  - Because of the deep integration with Pin, implementing PinPlay checks for these cases could be error prone
  - Instead, we implemented PinPlay support as a part of the SIFT trace generator (record-trace)
- Our trace recorder has a much lower impact on the execution traces
  - It supports command-line options to easily disable Linux syscall emulation
  - Behind the scenes, run-sniper --pinballs=foo automatically spawns a new record-trace process to generate a SIFT trace

Current Sniper-PinPlay Integration



sniper

Optimization Notice

Sniper PinPoints Evaluation

SPEC CPU2006 PinPoints Study

- We can use Sniper as a way to evaluate the accuracy of the Pinballs created using runtime as a metric

- Experiment with PinPoints inside of Sniper
  - Show results for a basic SimPoint configuration
  - Compare a matrix of results to improve outliers

sniper

SPEC CPU2006 PinPoints Study

- SPEC CPU2006
  - Reference inputs
  - 50 of the 55 benchmarks completed (5 full-simulations are still running)
- Environment
  - Compiled with GCC 4.3, -O2, SSE2 enabled
  - Sniper version 4.2, gainestown configuration, single core
  - PinPlay prerelease version

SPEC CPU2006 PinPoints Study

- Runtime
  - Whole-program pinball generation takes about a week per SPEC CPU2006 ref-input benchmark
  - PinPoints pinball generation takes an additional 1 to 5 days
  - Sniper Timing Simulation
    - 100M warmup + 30M detailed
      - 5 minutes
    - Full-reference inputs
      - Days to months
- Trace Sizes
  - Total of 3.8 GiB, from 2.7MiB (gamess) to 1.1GiB (milc)

sniper

Optimization Notice

SPEC CPU2006 CPI Error

30M detailed, 100M warmup, maxk=5

astar_1-ref-1	2.54	**gcc_1-ref-1**	0.30	**gobmk-ref-1**	1.12	**perlbench_2-ref-1**	2.24
astar-ref-1	8.11	**gcc_2-ref-1**	18.70	**gromacs-ref-1**	2.81	**perlbench-ref-1**	5.33
bzip2_1-ref-1	14.90	**gcc_3-ref-1**	9.37	**h264ref_1-ref-1**	1.03	**povray-ref-1**	2.25
bzip2_2-ref-1	0.57	**gcc_4-ref-1**	11.67	**h264ref_2-ref-1**	0.33	**sjeng-ref-1**	1.06
bzip2_3-ref-1	15.29	**gcc_5-ref-1**	1.66	**h264ref-ref-1**	2.36	**soplex_1-ref-1**	4.22
bzip2_4-ref-1	5.95	**gcc_6-ref-1**	3.14	**hmmer_1-ref-1**	0.04	**soplex-ref-1**	1.23
bzip2_5-ref-1	10.53	**gcc_7-ref-1**	16.26	**hmmer-ref-1**	0.08	**sphinx3-ref-1**	0.82
bzip2-ref-1	9.55	**gcc_8-ref-1**	9.23	**lbm-ref-1**	1.79	**tonto-ref-1**	0.20
calculix-ref-1	1.30	**gcc-ref-1**	9.79	**mcf-ref-1**	4.63	**wrf-ref-1**	14.27
dealII-ref-1	21.89	**gobmk_1-ref-1**	0.82	**milc-ref-1**	6.01	**xalancbmk-ref-1**	18.42
gamess_1-ref-1	0.56	**gobmk_2-ref-1**	1.09	**namd-ref-1**	31.79	**zeusmp-ref-1**	5.16
gamess_2-ref-1	2.15	**gobmk_3-ref-1**	0.76	**omnetpp-ref-1**	3.51		
gamess-ref-1	4.26	**gobmk_4-ref-1**	2.70	**perlbench_1-ref-1**	0.93	**avg abs err**	5.89

SPEC CPU2006 PinPoints Study

- Can we improve these timing error rates?
  - Compare a number of different maxK and warmup values

cpu2006-gcc_2-1, detailed = 30M insn		
	maxk 5	maxk 20
warmup 100M	18.70	6.99
warmup 300M	18.61	7.26

sniper

Optimization Notice

Sniper PinPoints Integration

Sniper PinPoints Integration

- The pinpoints.py script provides a basic infrastructure for creating PinPoints
- The sniper_pinpoints.py script enhances pinpoints.py to support automatic reference and PinPoint playback and comparison

- Provide a overview to extending the PinPoints scripts for your simulator

sniper

PinPoints Simulator Integration

- Register additional phases as options
- If those options have been selected, run the simulation

```
import pinpoints
class SniperPinPoints(pinpoints.PinPoints):
  def AddAdditionalOptions(self, parser):
    “””Add additional options specific for Sniper”””
  def AddAdditionalPhaseOptions(self, parser, phase_group):
    “””Add additional phase options specific for Sniper”””
  def RunAdditionalPhases(self, wp_log_dir, sim_replay_cmd, options):
    “””Run additional phases specific for Sniper”””
    return 0
if __name__ == “__main__”:
  sys.exit(SniperPinPoints().Run())
```

PinPoints Simulator Integration

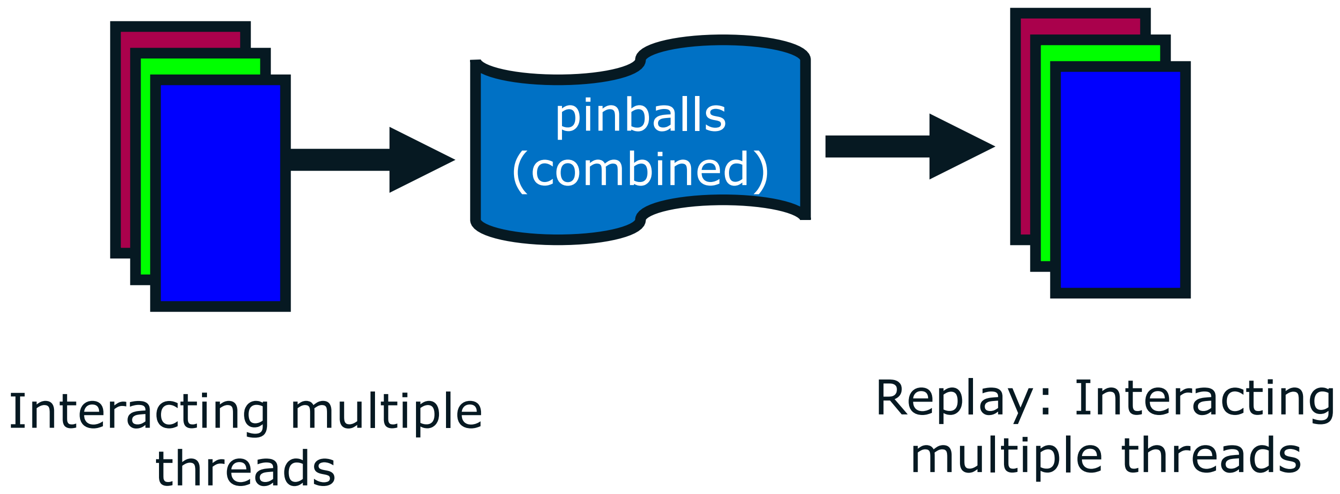
- Register additional options and phases

```
import pinpoints, cmd_options
def sniper_root(parser):
   parser.add_option("--sniper_root", dest="sniper_root",
         default=os.getenv("SNIPER_ROOT"), help="Sniper root")
def whole_sim(parser, group):
   method = cmd_options.GetMethod(parser, group)
   method("-W", "--whole_sim", dest="whole_sim", action="store_true",
         help="Run Sniper on the whole program pinballs.")
class SniperPinPoints(pinpoints.PinPoints):
 def AddAdditionalOptions(self, parser):
   sniper_root(parser)
 def AddAdditionalPhaseOptions(self, parser, phase_group):
   whole_sim(parser, phase_group)
 def RunAdditionalPhases(self, wp_log_dir, sim_replay_cmd, options):
   if options.whole_sim or options.default_phases:
     pass # Run the whole program simulation
   return 0
```


Optimization Notice

Handling Parallel Programs

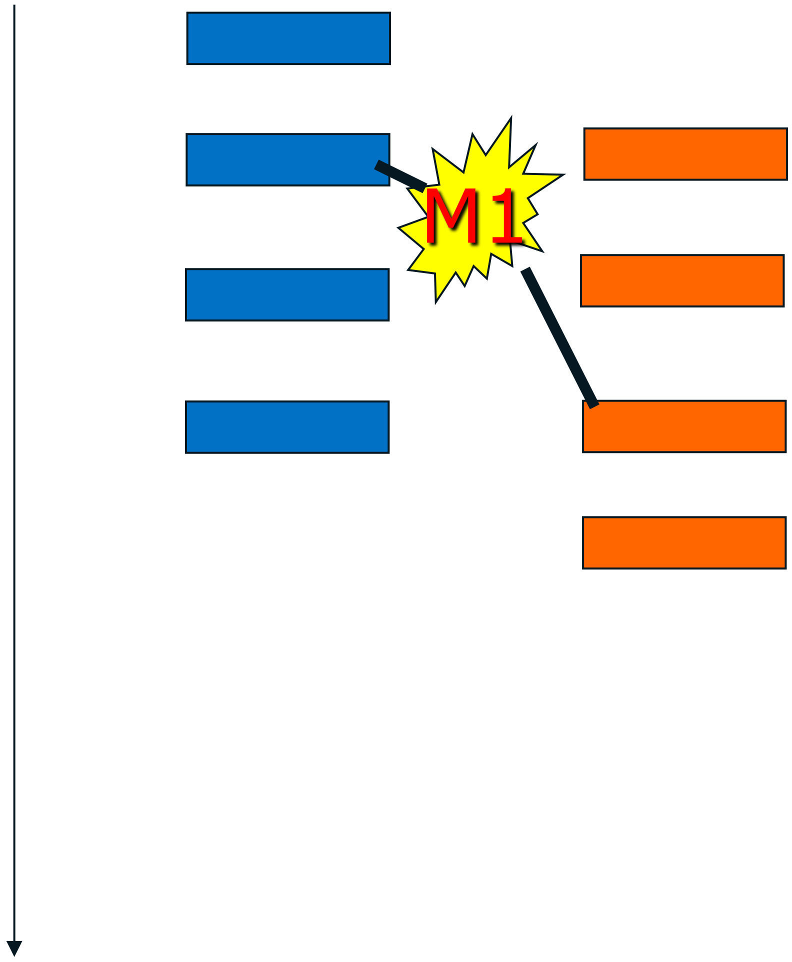
Model 1: Parallel Capture : Parallel Replay For Multi-threaded Programs



Useful for parallel analysis/simulation

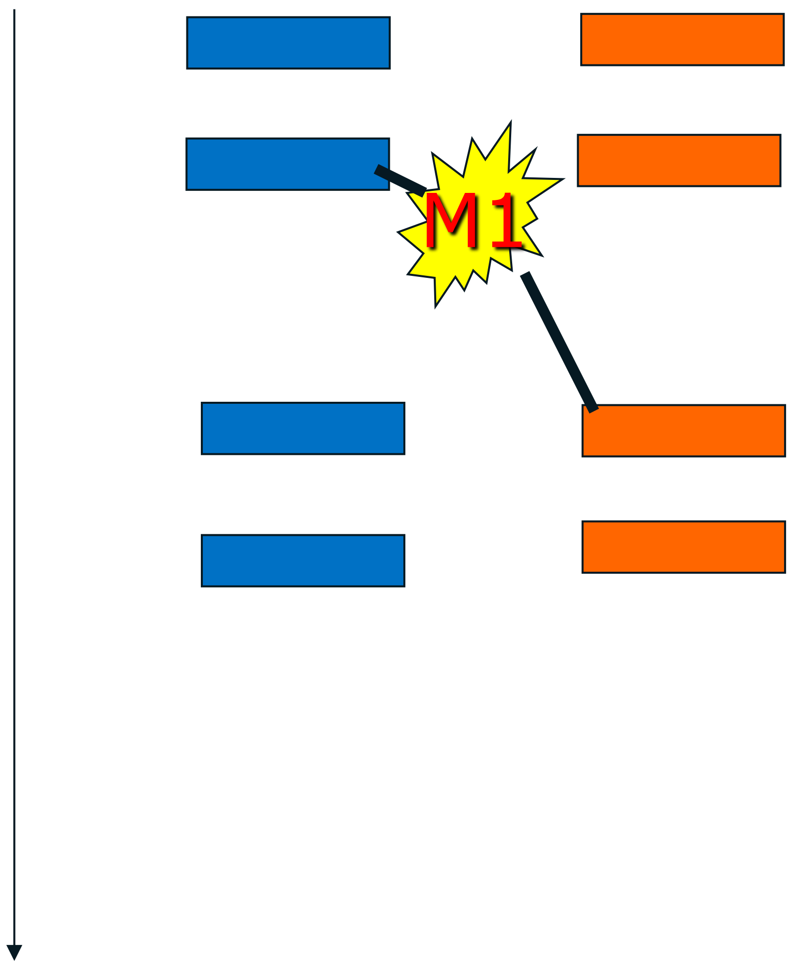
[Can focus on one thread with –log:focus_thread]

PinPlay's Determinism == Same Access Order for Conflicting Shared Memory Accesses



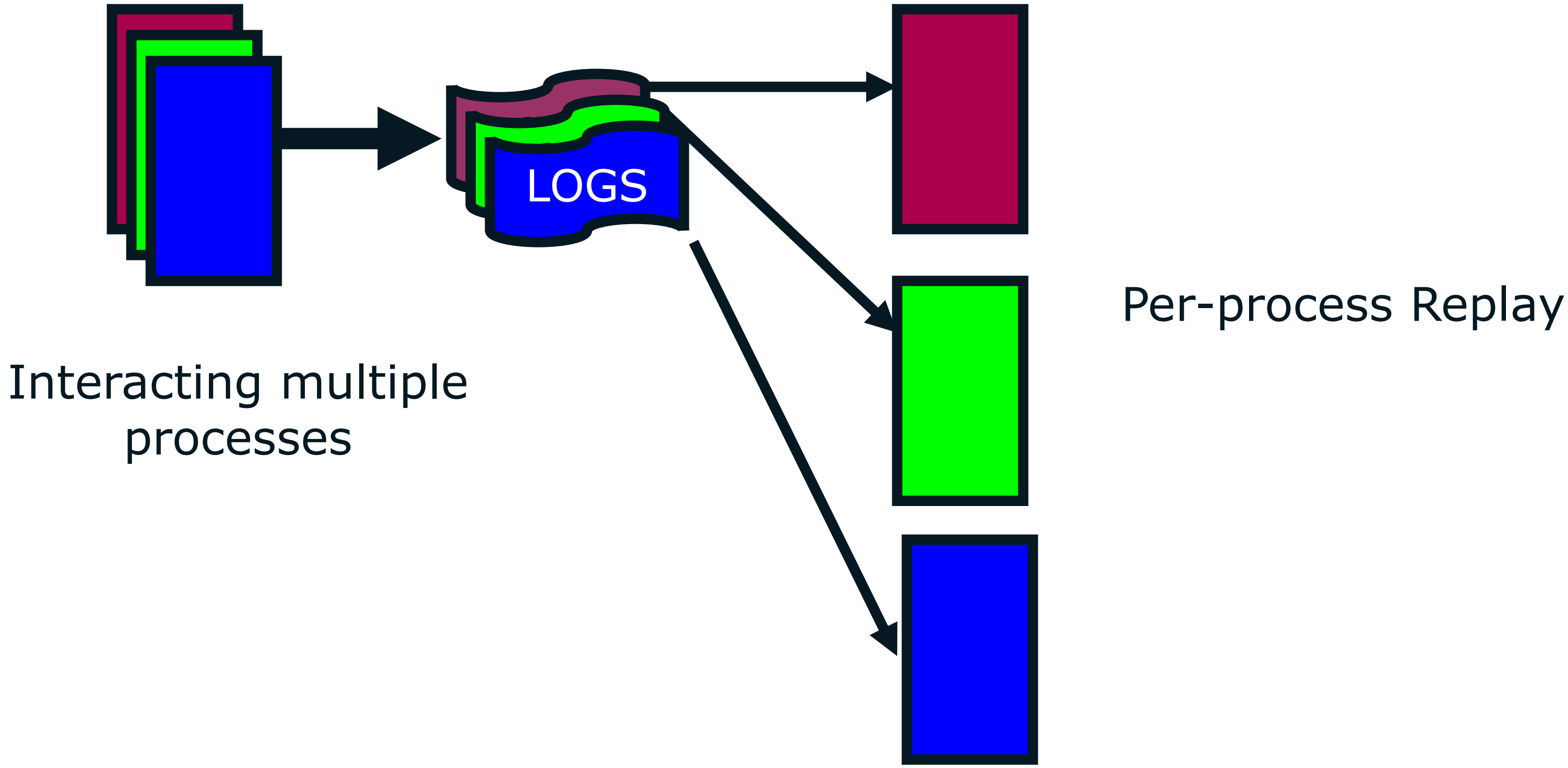
- Instructions from each thread replayed in program order
- RAW, WAR, WAW order for multiple threads is preserved

Relative Speed of Threads will Change During Replay



- Instructions from each thread replayed in program order
- RAW, WAR, WAW order for multiple threads is preserved

Model 2: Parallel Capture : Isolated Replay For Multi-process Programs



- **multi-process ➔ multi-programmed**

Extending PinPoints to Parallel Programs [Work In Progress]

- **<u>Multi-threaded Programs</u>**:
  - Per thread pinball: ***-log:focus_thread tid***
    Whole-program logging, PinPoints generation same as single-threaded program
  - Truly multi-threaded ("co-operative") pinball:
    - Use basic block vector from thread 0 (or any specified thread)
    - Generate multi-threaded PinPoint pinballs
    - Caveat: Simulator cannot change thread order!
      [Sniper runs into a problem trying to emulate futex()]
- **<u>Multi-process Programs:</u>** (per-process pinballs only)
  - Focus on one process pinball for PinPoints generation

Example: Multi-threaded PinPoints Generating Cooperative Pinballs

```
% pinpoints.py --pinplayhome=$PIN_KIT --cfg thread.coop.cfg --coop_pinball -l -b -s -p
```

```
% cat thread.coop.cfg
[Parameters]
program_name:   thread
input_name:     parallel
command:        "base.exe 3"
epilog_length:  0
maxk:           5
mode:           mt
num_proc:       1
prolog_length:  0
slice_size:     1000000
warmup_length:  3000000
```

```
% tree thread.1.parallel_14134.pp/ | grep t0r1 | grep race
|-- thread.1.parallel_14134_t0r1_warmup3001500_prolog0_region215307_epilog0_001_1-00000.0.race.bz2
|-- thread.1.parallel_14134_t0r1_warmup3001500_prolog0_region215307_epilog0_001_1-00000.1.race.bz2
|-- thread.1.parallel_14134_t0r1_warmup3001500_prolog0_region215307_epilog0_001_1-00000.2.race.bz2
|-- thread.1.parallel_14134_t0r1_warmup3001500_prolog0_region215307_epilog0_001_1-00000.3.race.bz2
```

- PinPoints pinballs:
  - truly multi-threaded (co-operative) for 1 (main) + 3 (created) threads
  - **Will not work with Sniper** currently

Example: Multi-threaded PinPoints Generating Per-thread Pinballs [thread 2]

```
% pinpoints.py --pinplayhome=$PIN_KIT --cfg thread.perthread.cfg --focus_thread=2 -l -b -s -p
```

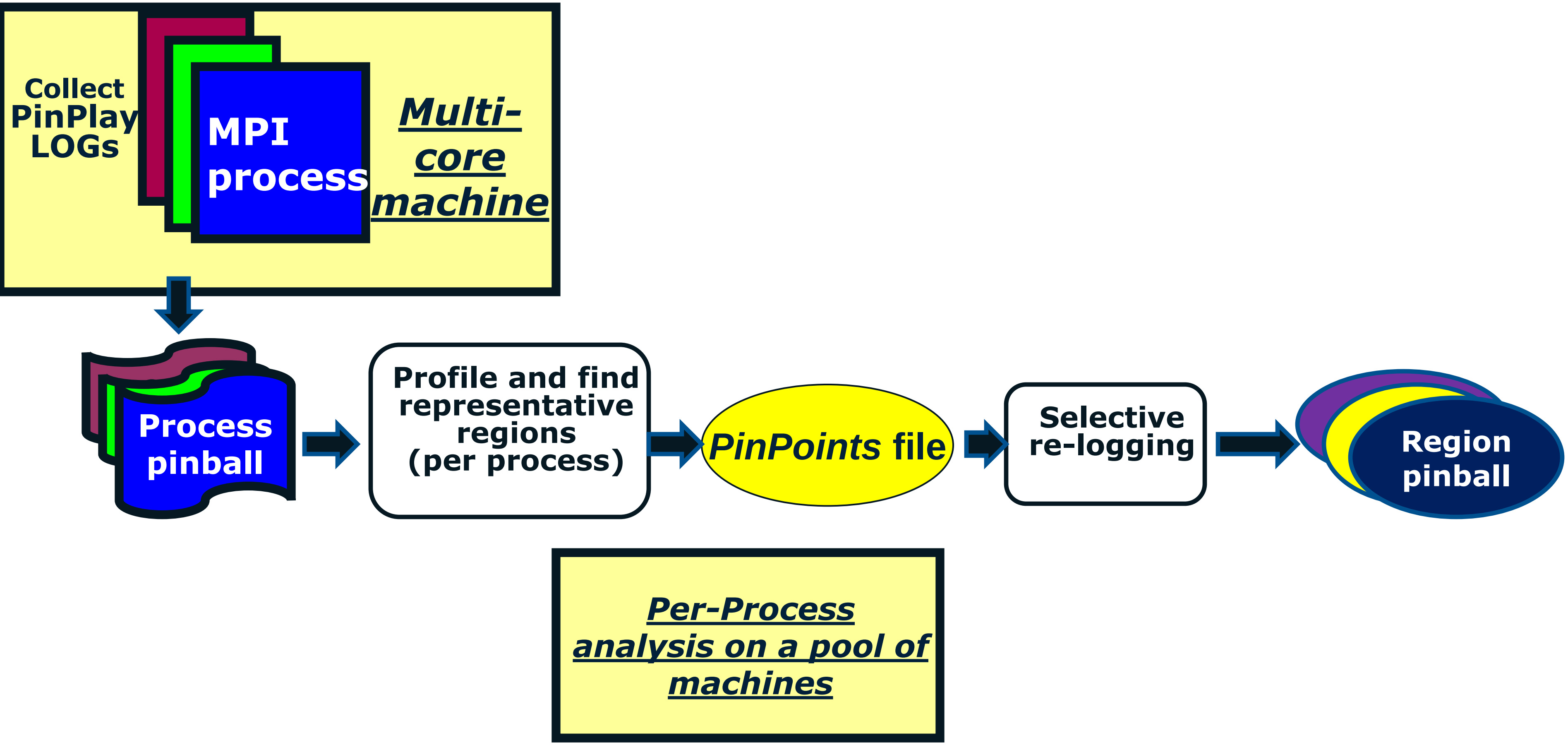
```
% cat thread.perthread.cfg
[Parameters]
program_name:   thread
input_name:     per-thread
command:        "base.exe 3"
epilog_length:  0
maxk:           5
mode:           mt
num_proc:       1
prolog_length:  0
slice_size:     1000000
warmup_length:  3000000
```

```
% tree thread.1.per-thread_22112.pp/ | grep t2r1 | grep race
|-- thread.1.per-thread_22112_t2r1_warmup0_prolog0_region1000002_epilog0_001_1-00000.2.race.bz2
```

- PinPoints pinballs:
  - Single-threaded
  - **Will work with Sniper** currently

Optimization Notice

MPI PinPoints Process Flow: Parallel Logging : Isolated Replay



Example: Intel MPI program with 2 processes (milc)

```
% $PIN_KIT/extras/pinplay/PinPoints/scripts/pinpoints.py --pinplayhome
=$PIN_KIT --cfg milc.mpi.cfg -l -b -s -p | & tee out
```

```
% cat milc.mpi.cfg
[Parameters]
program_name:   milc
input_name:     test
command:        "base.exe test < /dev/null"
epilog_length:  0
maxk:           5
mode:           mpi
num_proc:       2
prolog_length:  0
slice_size:     1000000
warmup_length:  3000000
```

```
% tree whole_program.test | grep address
|-- milc.2.test_22555.address
|-- milc.2.test_22556.address
```

```
milc/
|-- milc.2.test_22555.Data
|-- milc.2.test_22555.pp
|-- milc.2.test_22556.Data
|-- milc.2.test_22556.pp
`-- whole_program.test
```

- num_proc:2 ➔ “mpirun –n 2” was used
  2 whole-program pinballs for 2 processes
  PinPoints created for 2 processes per-process

Optimization Notice

Other Multi-process Models

1. **A program creating multiple processes**
   - Parent forks children → Children fork grand-children
   - Entire family tree communicate using inter-process shared memory

2. **Multiple programs/processes interacting:**

- Two independently invoked programs
- Communicate using inter-process shared memory

<..>/PinPoints/scripts/*.py do not currently support these models yet

Optimization Notice

Example: Program with fork()

```
% $PIN_KIT/pin -t $PIN_KIT/extras/pinplay/bin/intel64/pinplay-driver.so -log -log:mp_mode -log:basename pinbal
l/foo -log:pid -- fork_app
APPLICATION: Before fork
APPLICATION: After fork in parent
APPLICATION: After fork in child
```

```
pinball/
`-- fk0.fork_app
```

```
|-- fk0.fork_app
|     |-- foo_18786_18793.0.race
|     |-- foo_18786_18793.0.reg
|     |-- foo_18786_18793.0.result
|     |-- foo_18786_18793.address
|     |-- foo_18786_18793.procinfo.xml
|-- foo_18786.0.race
|-- foo_18786.0.reg
|-- foo_18786.0.result
|-- foo_18786.address
|-- foo_18786.procinfo.xml
```

```
% $PIN_KIT/pin -xyzzy -reserve_memory pinball/foo_18786.address -t $PIN_KIT/extras/pinplay/bin/intel64/pinplay
-driver.so -replay -replay:basename pinball/foo_18786 -- $PIN_KIT/extras/pinplay/bin/intel64/nullapp
APPLICATION: Before fork
APPLICATION: After fork in parent
```

```
% $PIN_KIT/pin -xyzzy -reserve_memory pinball/fk0.fork_app/foo_18786_18793.address  -t $PIN_KIT/extras/pinplay
/bin/intel64/pinplay-driver.so -replay -replay:basename pinball/fk0.fork_app/foo_18786_18793 -- $PIN_KIT/extra
s/pinplay/bin/intel64/nullapp
APPLICATION: After fork in child
```

One pinball per process

Optimization Notice

Recipe for Independent MP Logging

Generic model:

```
%pin-t pinplay-driver.so -log:mp_create_pool -- <any_valid_binary>
```

```
% pin -t pinplay-driver.so -log -log:basename pinball1/foo  -log:mp_mode -log:mp_attach --
<program1>
```

```
%pin -t pinplay-driver.so -log -log:basename pinball2/bar -log:mp_mode -log:mp_attach --
<program2>
```

Client/Server model (only Client to be traced):

```
%pin -t pinplay-driver.so -log -log:basename pinball1/foo -log:mp_mode_lock_only --
<server_program_to_be_ignored>
```

```
% pin -t pinplay-driver.so -log -log:basename pinball2/bar -log:mp_mode -log:mp_attach --
<client_program_of_interest>
```

Summary

PinPoints (Pin + SimPoint + Sniper) methodology effectively automates the tedious task of finding and check-pointing regions of programs for Pin-based simulation.*

- *Creates check-points that are representative and repeatable*

http://www.pinplay.org http://www.snipersim.org

Optimization Notice

References

Pin: Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation; Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*.

PinPoints: Pinpointing Representative Portions of Large Intel® Itanium® Programs with Dynamic Instrumentation; Patil, H., Cohn, R., Charney, M., Kapoor, R., Sun, A., and Karunanidhi, A. In Proceedings of the *37th Annual IEEE/ACM international Symposium on Microarchitecture* (Portland, Oregon, December 04 - 08, 2004).

PinPlay: PinPlay: A Framework for Deterministic Replay and Reproducible Analysis of Parallel Programs; Harish Patil, Cristiano Pereira, Mack Stallcup, Gregory Lueck, James Cownie. CGO 2010. **CGO 2010 Best Paper Award Winner!**

SimPoint : Automatically Characterizing Large Scale Program Behavior ; Timothy Sherwood, Erez Perelman, Greg Hamerly and Brad Calder. In proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, October 2002.

Sniper: Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation; Trevor E. Carlson; Wim Heirman; Lieven Eeckhout. In proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2011.

Optimization Notice

谢谢

xièxiè

Thank you!

sniper

PinPoints

intel®

UNIVERSITEIT GENT