北京开源芯片研究院
BEIJING INSTITUTE OF OPEN SOURCE CHIP

中国开放指令生态（RISC-V）联盟
China RISC-V Alliance

# XiangShan: An Open-Source High-Performance RISC-V Processor and Infrastructure for Architecture Research

*The XiangShan Team*

ASPLOS'24@San Diego, USA

April 27, 2024

# What we will cover in this tutorial

- **Introduce of XiangShan (30 minutes)**

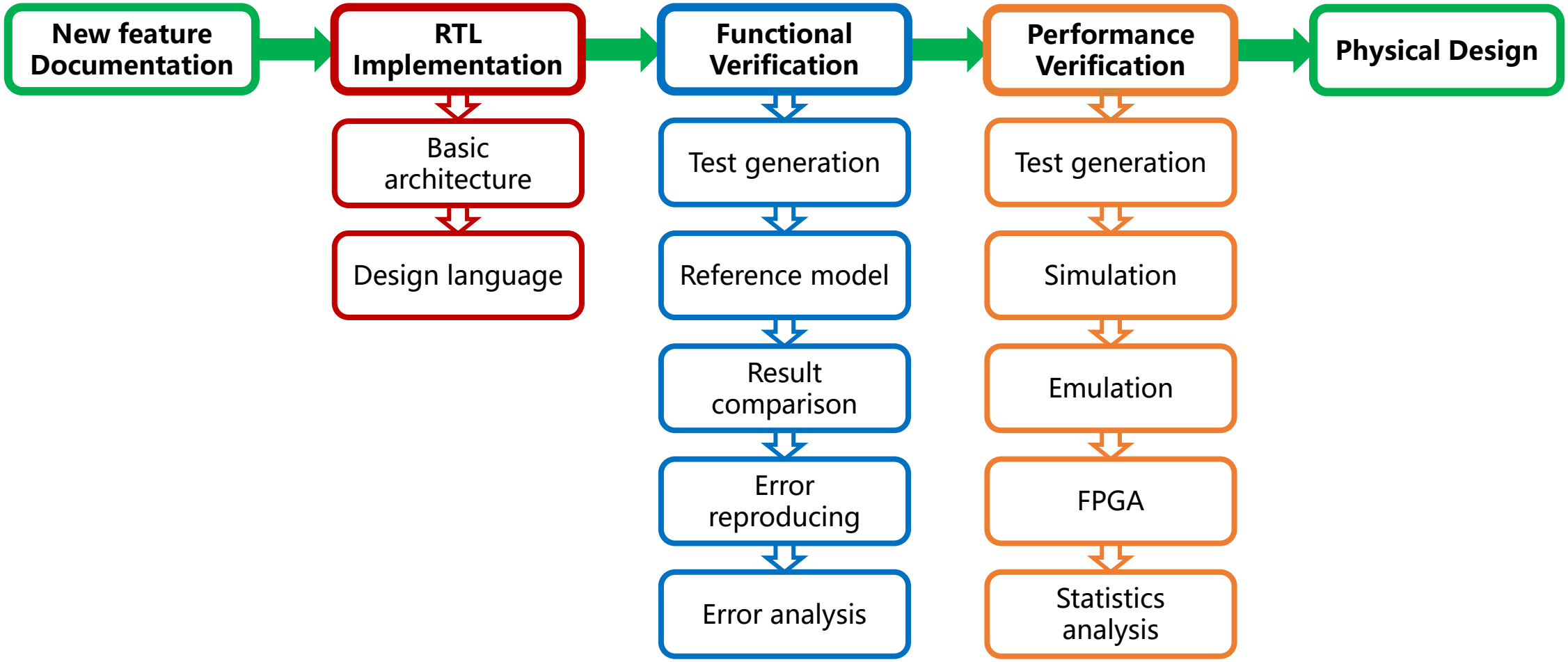
- **CPU Microarchitecture (60 minutes)**
  - *Design and implementation – How to implement novel ideas on XiangShan*
  - Frontend: branch prediction and instruction fetch
  - Backend: out-of-order scheduler, execution units
  - Load/Store Unit: LSQ, pipelines, TLBs, data caches
  - L2/L3 caches and prefetchers

- **Development workflows (60 minutes)**
  - *Introduction of their usages – How to develop on XiangShan with MinJie*
  - Simulation and Debugging
  - Research Demo

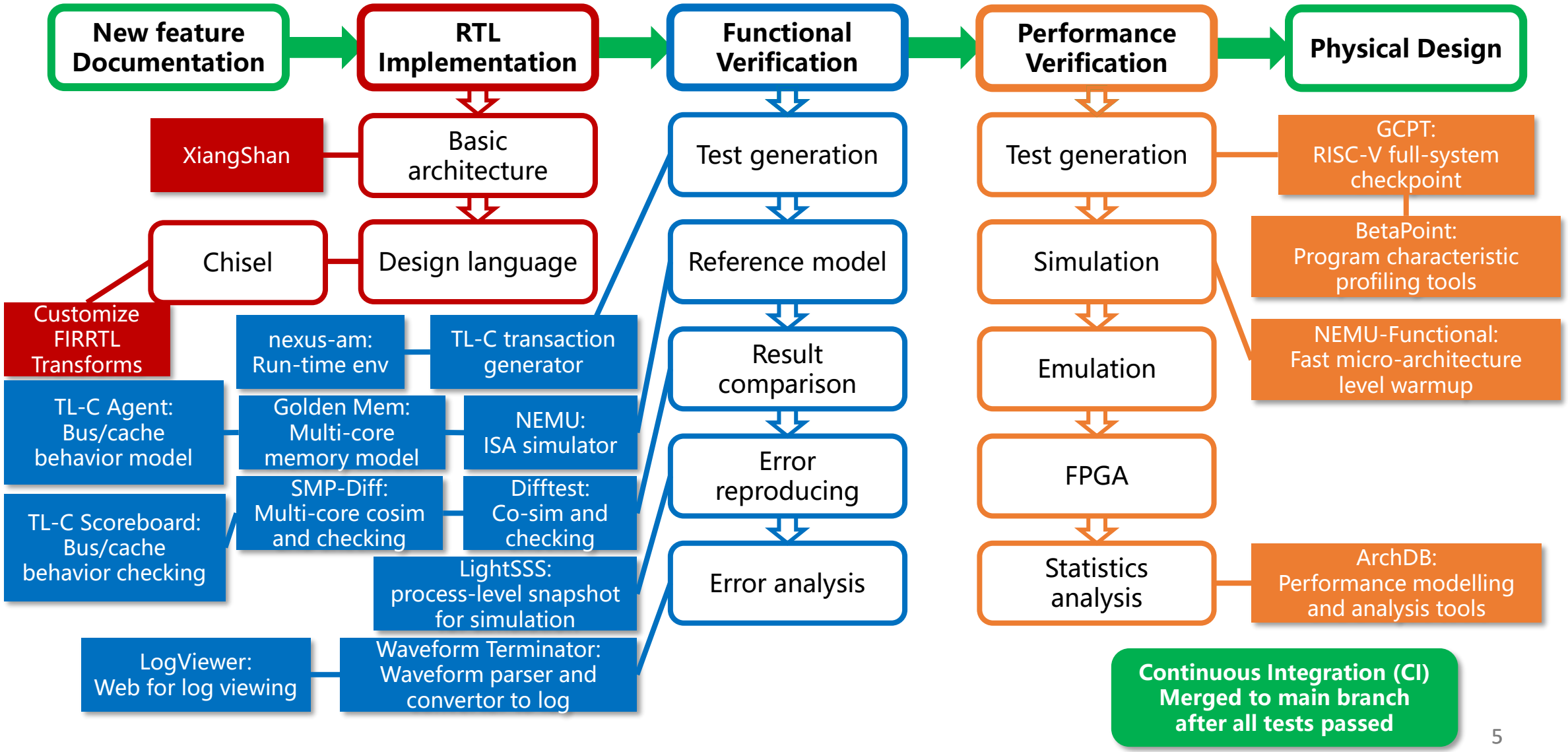
# Tutorial@ASPLOS'24 Schedule

| Time (PM) | Topic |
|---|---|
| 1:30 - 2:00 | Introduction of the XiangShan Project |
| 2:05 - 3:00 | Microarchitecture Design and Implementation |
| | Afternoon Break |
| 3:30 – 5:00 | **Hands-on Development & Discussions** |

# Roadmap: MinJie Development Flows and Tools



# Roadmap: MinJie Development Flows and Tools



# In this tutorial, we will

- Provide access to cloud servers with you
- Prepare the XiangShan development environment for you
- Go through the development workflows including:

| Simulation | Func. Verification | Perf. Verification | Research Demo |
|---|---|---|---|
| • Environment<br>• RTL Generation<br>• RTL Simulation<br>…… | • Nexus-AM<br>• NEMU<br>• Difftest<br>…… | • Perf. counter<br>• Constin<br>• Checkpoint<br>…… | • Oracle BP<br>• Top-Down<br>…… |

- ***Required: machines with networking and SSH***

# Demo Instructions

- **Interactive shell commands** are highlighted in **brown box** with prefix $

```
$ echo "Hello, XiangShan"
$ echo "Have a nice day"
```

- **Description and notes** are presented in **grey box** with prefix #

```
# Please prepare a laptop with an SSH client.
# Next, let's start the demo session !
```

## Let's Start!

# Prerequisites

- **Login to the provided cloud server**
  - **Windows (Windows Terminal / PowerShell is recommended)**

```
PS > ssh guest@47.254.126.201
# Password: asplos2024
```

  - **Linux / Mac**

```
$ ssh guest@47.254.126.201
# Password: asplos2024
```

For **offline users**, please refer to https://github.com/OpenXiangShan/xs-env/tree/tutorial-new

# Prerequisites

```
# Copy tutorial environment to your dir based on your name
$ cp -r /opt/xs-env ~/<YOUR_NAME>

# Enter your dir
$ cd ~/<YOUR_NAME>

# Set up environment variables
# DO IT AGAIN when opening a new terminal
$ source env.sh
# SET XS_PROJECT_ROOT:                    /home/guest/YOUR_NAME
# SET NOOP_HOME (XiangShan RTL Home):     $XS_PROJECT_ROOT/XiangShan
# SET NEMU_HOME:                          $XS_PROJECT_ROOT/NEMU
# SET AM_HOME:                            $XS_PROJECT_ROOT/nexus-am
```

# Prerequisites

*# Project Structure*

```
$ tree -d -L 1
```

```
.
├── env-scripts        Useful scripts
├── NEMU               NEMU
├── XiangShan          XiangShan processor
├── nexus-am           Abstract machine
└── tutorial           Scripts and patches for tutorial
```

*# Enter XiangShan directory*

```
$ cd XiangShan
```

# Chisel Compilation

- **Compile RTL and build simulator with Verilator**

```
$ make emu CONFIG=MinimalConfig MFC=1 -j8
Options:
    CONFIG=MinimalConfig        Configuration of XiangShan
    MFC=1                       Enable MLIR Firrtl Compiler
    // EMU_THREADS=2            Simulation threads
    // EMU_TRACE=1              Enable waveform dump
    // WITH_DRAMSIM=1           Enable DRAMSim3 for DRAM simulation
    // WITH_CHISELDB = 1        Enable ChiselDB feature
    // WITH_CONSTANTIN = 1      Enable Constantin feature
```

- **Compilation might take 10 mins~**

# Open Another Terminal

```
# login to the cloud server again
$ ssh guest@47.254.126.201
# Password: asplos2024

# Come back to XiangShan directory
$ cd ~/<YOUR_NAME> && source env.sh && cd XiangShan
```

# Run RTL Simulation with Verilator

- **After building, we can run the simulator**

```
# run pre-built simulator
$ ../tutorial/emu -i ../tutorial/hello.bin --no-diff 2>hello.err

# Some key options:
    -i                          Workload to run
    -C / -I / -W                Max cycles  /Max Insts  / Warmup Insts
    --diff=PATH / --no-diff     Compare with NEMU for difftest / disable difftest
```

**Great!**

**We have learned the basic simulation process of Xiangshan.**

Once we get the RTL simulator ready, what's next?

How can we

- **generate desired workloads**
- **find and fix functional bugs**
- **do performance analysis**
- **do research on micro-architecture**

# MinJie Functional Verification

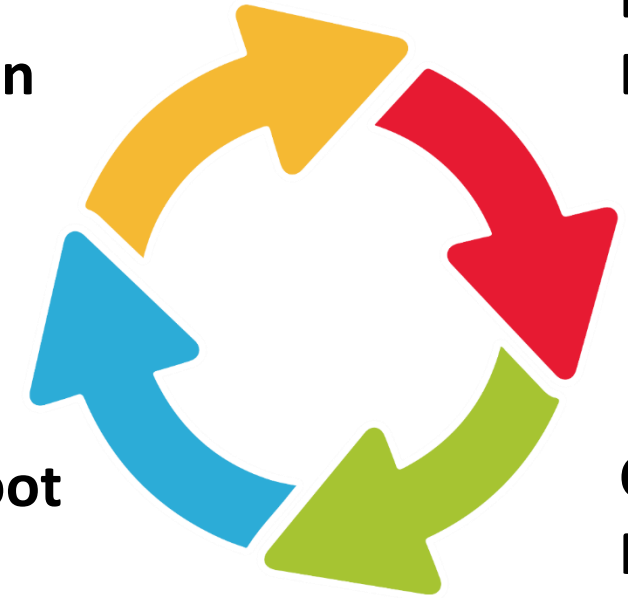
Nexus-AM

**Test Generation**

**Bug Detection**

Difftest

NEMU



Waveform

ChiselDB

**Troubleshoot & Fix bugs**

**Context Preservation**

LightSSS

# Bare Metal Runtime: Nexus-AM

- **Purpose**
  - Generate workloads **agilely** without an OS
  - Provide runtime framework for **bare metal machines** like XiangShan
- **We present a bare metal runtime called Nexus-AM**
  - Light-weight, easy to use
  - Implement basic **system call** interfaces and exception handlers
  - Support multiple **ISAs and configurations**

# Nexus-AM

- **Hands-on:** build Coremark workload
- **Showcase**

```
$ cd ~/<YOUR_NAME> && source env.sh && cd tutorial
$ bash build-am.sh
```

```
# cd $AM_HOME/apps/coremark
# make ARCH=riscv64-xs
# ls -l build
#     coremark-riscv64-xs.bin    Binary image of the program
#     coremark-riscv64-xs.elf    ELF of the program
#     coremark-riscv64-xs.txt    Disassembly of the program
```

# MinJie Functional Verification

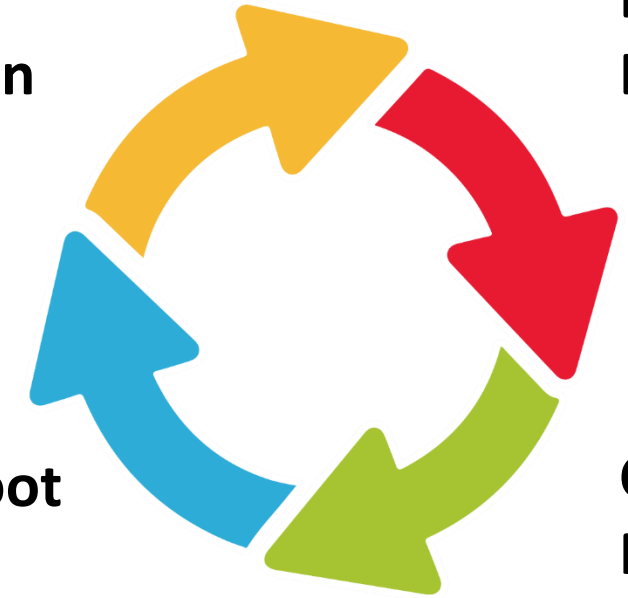
Nexus-AM

**Test Generation**

**Bug Detection**

Difftest

NEMU



Waveform

ChiselDB

**Troubleshoot & Fix bugs**

**Context Preservation**

LightSSS

# ISA REF: NEMU

- **Purpose**
  - Golden model for verification
  - Simple yet high performance
- **We present NEMU**
  - An **instruction set simulator**, similar to Spike
  - Through **optimization techniques**, achieve performance similar to QEMU.
  - **A set of APIs** is provided to assist XiangShan in comparing and verifying the **architectural states**

# NEMU

- **Showcase**

```
$ bash build-nemu.sh
```

```
# cd $NEMU_HOME
# make clean
# make riscv64-xs_defconfig
# make -j      build NEMU as the bare metal machine, can run the Coremark from the previous step
# make clean-softfloat
# make riscv64-xs-ref_defconfig
# make         build NEMU as the reference model for XiangShan
```

# NEMU

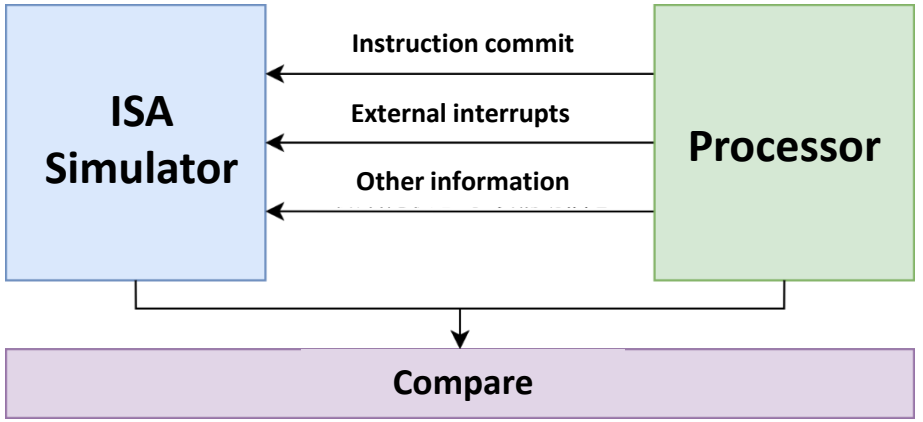
- **Hands-on:** run Coremark workload on NEMU
- **Showcase**

```
$ bash run-nemu.sh
```

```
# cd $NEMU_HOME
# ./build/riscv64-nemu-interpreter -b \                    run in batch mode, faster
$AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin       set workspace to Coremark
```

# ISA co-simulation using Difftest

- **Basic flows**
  - Instructions commit/other states update
  - The simulator executes the same instructions
  - Compare the architectural states between DUT and REF
  - Abort or continue
- **Provided as verification infrastructure for processors**
  - APIs for HDLs such as Chisel/Verilog
  - RTL simulators such as Verilator, VCS
  - RISC-V ISS such as **NEMU**, **Spike**
- **SMP-Difftest: co-simulation on an SMP processor**
  - SMP Linux kernel and multi-threading programs
  - Online checking of cache coherency and memory consistency




**Basic architecture of DiffTest**

```
while (1) {
  icnt = cpu_step();
  nemu_step(icnt);
  r1s = cpu _getregs();
  r2s = nemu_getregs();
  if (r1s != r2s) { abort(); }
}
```

**Online checking**

# Difftest

- **Hands-on:** run Coremark workload on XiangShan, difftest with NEMU
- **Showcase**

```
# Running Coremark takes about 2 minutes
$ bash run-emu.sh
```

```
# ./emu \          if the simulator is successfully compiled, you can use $NOOP_HOME/build/emu
-i $AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin \   use coremark.bin
--diff $NEMU_HOME/build/riscv64-nemu-interpreter-so \       difftest with NEMU
2>perf.out       redirect stderr to file
```

# Difftest

```
./emu -i $AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin --diff $NEMU_HOME/build/riscv64-nemu-interpreter-so
Emu compiled at Mar 24 2023, 20:33:03
Using simulated 32768B flash
[warning]no valid flash bin path, use preset flash instead
The image is /home/guest/xs-env-asplos2023/nexus-am/apps/coremark/build/coremark-riscv64-xs.bin
Using simulated 8192MB RAM
NemuProxy using /home/guest/xs-env-asplos2023/NEMU/build/riscv64-nemu-interpreter-so
The first instruction of core 0 has commited. Difftest enabled.
[NEMU] Can not find flash image: (null)
[NEMU] Use built-in image instead
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'flash' at [0x0000000010000000, 0x000000000100fffff]
Running CoreMark for 1 iterations
2K performance run parameters for coremark.
CoreMark Size    : 666
Total time (ms)  : 4288
Iterations       : 1
Compiler version : GCC9.4.0
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0xe714
Finised in 4288 ms.
==================================================
CoreMark Iterations/Sec 233
Core 0: HIT GOOD TRAP at pc = 0x80001c52
total guest instructions = 398,372
instrCnt = 398,372, cycleCnt = 517,099, IPC = 0.770398
Seed=0 Guest cycle spent: 517,103 (this will be different from cycleCnt if emu loads a snapshot)
Host time spent: 77,559ms
```

# Difftest

- **Hands-on:**
  - We injected a bug in a pre-built XiangShan processor
  - Now, let's trigger it on the pre-built buggy XiangShan by Difftest
- **Showcase**

```
$ bash run-emu-diff.sh
```

```
# ./emu-bug \
  -i $AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin \
  --diff $NEMU_HOME/build/riscv64-nemu-interpreter-so    # difftest with NEMU
```

# Difftest

- Difftest will report the error once failed
  - Dump info like PC, GPRs, etc.
  - Point out the differences

```
============== Commit Group Trace (Core 0) ==============
commit group [00]: pc 0080000118 cmtcnt 7
commit group [01]: pc 008000012c cmtcnt 6
commit group [02]: pc 00800013b2 cmtcnt 6
commit group [03]: pc 0080000124 cmtcnt 7
commit group [04]: pc 0080000136 cmtcnt 6
commit group [05]: pc 00800013c0 cmtcnt 6
commit group [06]: pc 00800013da cmtcnt 6
commit group [07]: pc 0080001400 cmtcnt 7
commit group [08]: pc 0080001414 cmtcnt 7
commit group [09]: pc 0080001428 cmtcnt 6
commit group [10]: pc 0080001436 cmtcnt 6
commit group [11]: pc 00800017fe cmtcnt 6
commit group [12]: pc 008000180c cmtcnt 6 <--
commit group [13]: pc 008000139a cmtcnt 6
commit group [14]: pc 0080000124 cmtcnt 7
commit group [15]: pc 0080000154 cmtcnt 6
```

```
==============  REF Regs  ==============
  $0: 0x0000000000000000   ra: 0x00000000800013b8   sp: 0x000000008000cf00   gp: 0x0000000000000000
  tp: 0x0000000000000000   t0: 0x0000000000000000   t1: 0x0000000800039d0   t2: 0x0000000000000000
  s0: 0x000000008000cf10   s1: 0x0000000000000000   a0: 0x000000000000029a   a1: 0x0000000000000001
  a2: 0x0000000000000001   a3: 0x0000000000000002   a4: 0x0000000000000002   a5: 0x0000000000000007
  a6: 0x0000000000000003   a7: 0x0000000000000001   s2: 0x0000000000000000   s3: 0x0000000000000000
  s4: 0x0000000000000000   s5: 0x0000000000000000   s6: 0x0000000000000000   s7: 0x0000000000000000
  s8: 0x0000000000000000   s9: 0x0000000000000000  s10: 0x0000000000000000  s11: 0x0000000000000000
  t3: 0x000000800039d0   t4: 0x0000000000000001   t5: 0x000000008000cdc1   t6: 0x0000000000000000
 ft0: 0xffffffff00000000  ft1: 0xffffffff00000000  ft2: 0xffffffff00000000  ft3: 0xffffffff00000000
 ft4: 0xffffffff00000000  ft5: 0xffffffff00000000  ft6: 0xffffffff00000000  ft7: 0xffffffff00000000
 fs0: 0xffffffff00000000  fs1: 0xffffffff00000000  fa0: 0xffffffff00000000  fa1: 0xffffffff00000000
 fa2: 0xffffffff00000000  fa3: 0xffffffff00000000  fa4: 0xffffffff00000000  fa5: 0xffffffff00000000
 fa6: 0xffffffff00000000  fa7: 0xffffffff00000000  fs2: 0xffffffff00000000  fs3: 0xffffffff00000000
 fs4: 0xffffffff00000000  fs5: 0xffffffff00000000  fs6: 0xffffffff00000000  fs7: 0xffffffff00000000
 fs8: 0xffffffff00000000  fs9: 0xffffffff00000000 fs10: 0xffffffff00000000 fs11: 0xffffffff00000000
 ft8: 0xffffffff00000000  ft9: 0xffffffff00000000 ft10: 0xffffffff00000000 ft11: 0xffffffff00000000
pc: 0x000000008000143a mstatus: 0x8000000a00006000 mcause: 0x0000000000000000 mepc: 0x0000004fe6a8c3dc
                       sstatus: 0x8000000200006000 scause: 0x0000000000000000 sepc: 0x0000000000000000
satp: 0x0000000000000000
mip: 0x0000000000000000 mie: 0x0000000000000000 mscratch: 0x0000000000000000 sscratch: 0x0000000000000000
mideleg: 0x0000000000000000 medeleg: 0x0000000000000000
mtval: 0x0000000000000000 stval: 0x9738662f59a3300f mtvec: 0x0000000000000000 stvec: 0x0000000000000000
privilege mode:3  pmp: below
 0: cfg:0x00 addr:0x0000000000000000| 1: cfg:0x00 addr:0x0000000000000000
 2: cfg:0x00 addr:0x0000000000000000| 3: cfg:0x00 addr:0x0000000000000000
 4: cfg:0x00 addr:0x0000000000000000| 5: cfg:0x00 addr:0x0000000000000000
 6: cfg:0x00 addr:0x0000000000000000| 7: cfg:0x00 addr:0x0000000000000000
 8: cfg:0x00 addr:0x0000000000000000| 9: cfg:0x00 addr:0x0000000000000000
10: cfg:0x00 addr:0x0000000000000000|11: cfg:0x00 addr:0x0000000000000000
12: cfg:0x00 addr:0x0000000000000000|13: cfg:0x00 addr:0x0000000000000000
14: cfg:0x00 addr:0x0000000000000000|15: cfg:0x00 addr:0x0000000000000000
priviledgeMode: 3
     a3 different at pc = 0x008000180c, right= 0x0000000000000002, wrong = 0x0000000000000000
Core 0: ABORT at pc = 0x80000752
instrCnt = 1,547, cycleCnt = 12,103, IPC = 0.127820
Seed=0 Guest cycle spent: 12,106 (this will be different from cycleCnt if emu loads a snapshot)
Host time spent: 11,908ms
```

# Difftest

- **Hands-on:** Dump waveform after Difftest detects the bug

- **Showcase**

```
$ bash run-emu-dumpwave.sh
$ ls ../XiangShan/build | grep "vcd"   # There should be a .vcd waveform
```

```
# ./emu-bug \
-i $AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin \
--diff $NEMU_HOME/build/riscv64-nemu-interpreter-so \
-b 12000 \      the begin cycle of the wave, you may find XiangShan trap at 12103s cycle in previous step
-e 13000 \      the end cycle of the wave
--dump-wave \   set this option to dump wave, you will find *.vcd on $NOOP_HOME/build
```

# MinJie Functional Verification

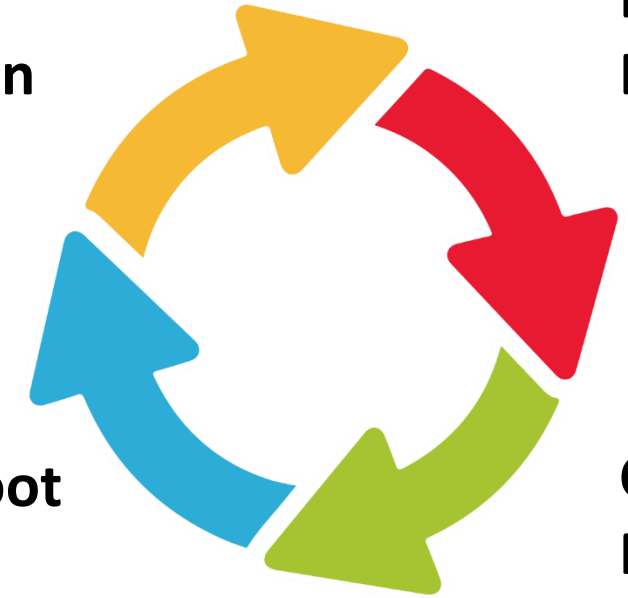
Nexus-AM

**Test Generation**

**Bug Detection**

Difftest

NEMU



Waveform

ChiselDB

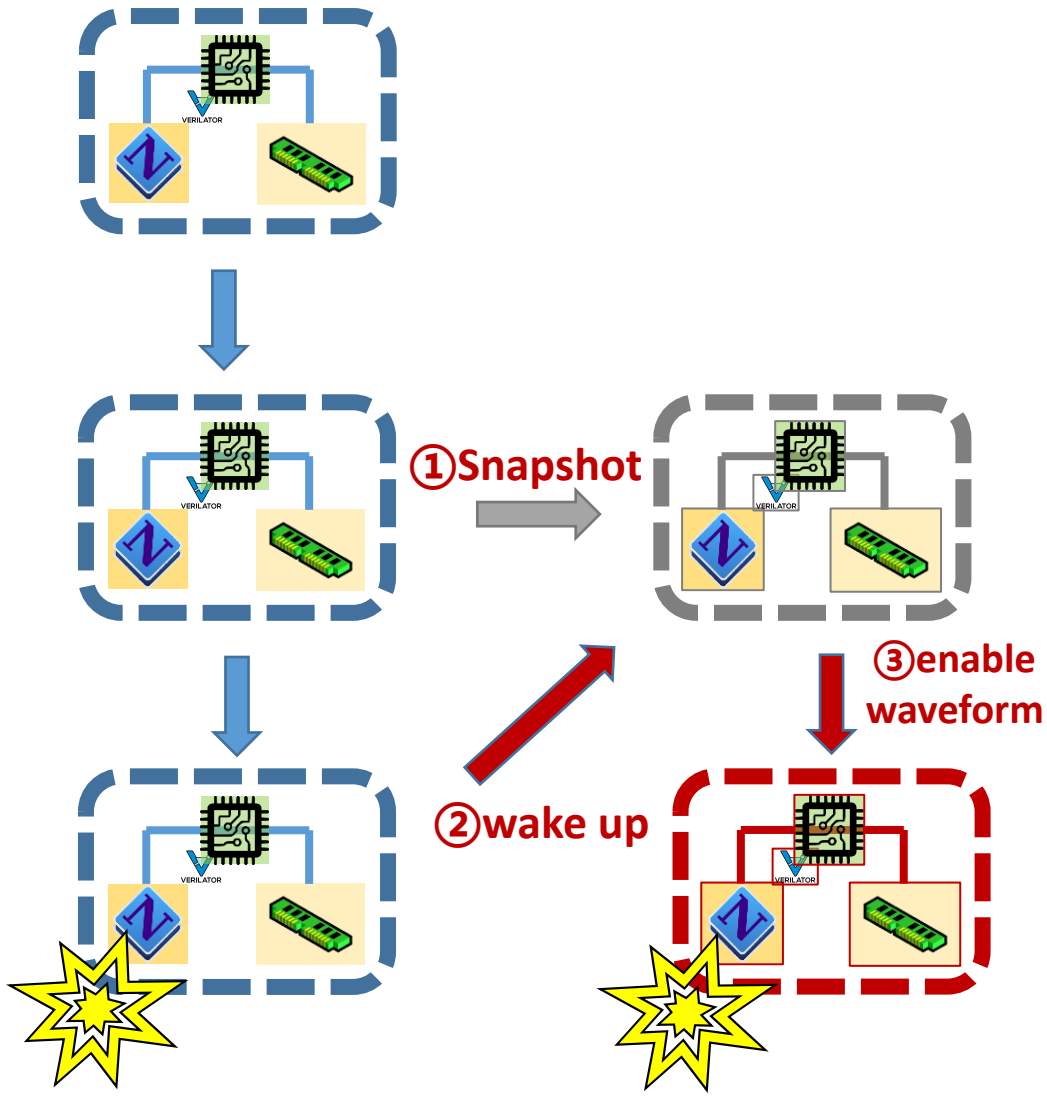
**Troubleshoot & Fix bugs**

**Context Preservation**

LightSSS

# Light-weight Simulation SnapShot: LightSSS

- **Re-run simulation is time-consuming!**
  - **Snapshot is the way out**
- **LightSSS: snapshots of the process with fork()**
  - Copy-on-write on Linux Kernel
- **Advantage 1: Good portability and scalability**
  - Snapshots for any external models (such as C++)
  - No need to understand details of external models
- **Advantage 2: Low overhead of snapshots**
  - ~500us for taking a snapshot
  - Far less overhead than RTL snapshots by Verilator



# LightSSS for on-demand debugging (waveform)

- **Hands-on:** use lightSSS to dump waveform
- **Showcase**

```
$ bash run-emu-lightsss.sh
```

```
# ./emu-bug \
-i $AM_HOME/apps/coremark/build/coremark-riscv64-xs.bin \
--diff $NEMU_HOME/build/riscv64-nemu-interpreter-so \
--enable-fork \      No longer need to use complex parameters
2 > lightsss.err
```

# LightSSS for on-demand debugging (waveform)

- **Hands-on:** use lightSSS to dump waveform
- LightSSS is working if you see:

```
[FORK_INFO pid(NUMBER)] the oldest checkpoint start to dump wave and dump nemu log...
[FORK_INFO pid(NUMBER)] dump wave to /SOME/PATH/XXX.vcd...
```

- After that, simulation will start again from the existing latest snapshot

# MinJie Functional Verification

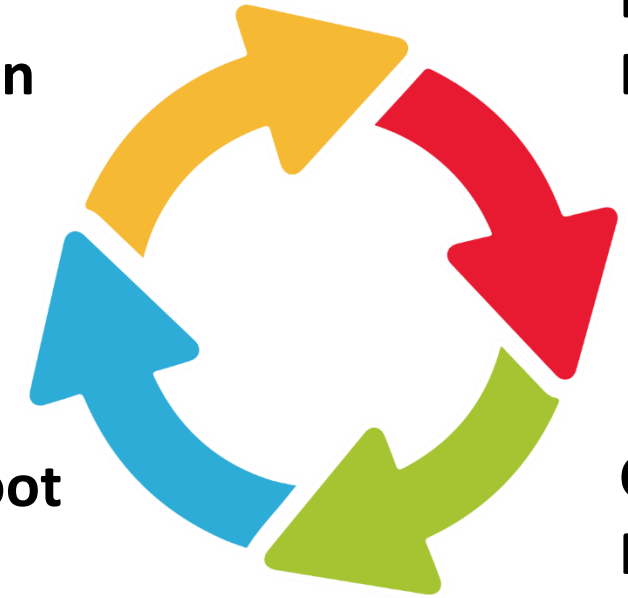
Nexus-AM

**Test Generation**

**Bug Detection**

Difftest

NEMU



Waveform

ChiselDB

**Troubleshoot & Fix bugs**

**Context Preservation**

LightSSS

# Debug-friendly Structured Database: ChiselDB

- **Motivation:**
  - Waveforms are **large** in size and **hard to apply** further analysis
  - Need to analyze structured data like memory transaction trace
- **We present ChiselDB**
- **Highlights:**
  - Inserting probes between module interfaces in hardware
  - DPI-C: Using C++ function in Chisel code to transfer data
  - Persist in database, SQL queries supported

# ChiselDB

- **Design source code:** *XiangShan/utility/src/main/scala/utility/ChiselDB.scala*
- **Usage:** Create table

```
// API: def createTable[T <: Record](tableName: String, hw: T): Table[T]
import huancun.utils.ChiselDB

class MyBundle extends Bundle {
        val fieldA = UInt(10.W)
        val fieldB = UInt(20.W)
}
val table = ChiselDB.createTable("MyTableName", new MyBundle)
```

# ChiselDB

- **Usage:** Register probes

```
/* APIs
def log(data: T, en: Bool, site: String = "", clock: Clock, reset: Reset)
def log(data: Valid[T], site: String, clock: Clock, reset: Reset): Unit
def log(data: DecoupledIO[T], site: String, clock: Clock, reset: Reset): Unit
*/
table.log(
  data = my_data /* hardware of type T */,
  en = my_cond,   site = "MyCallSite",
  clock = clock,   reset = reset
)
```

# ChiselDB

- **Example:** *XiangShan/src/main/scala/xiangshan/cache/mmu/L2TLB.scala*

```
class L2TlbMissQueueDB(implicit p: Parameters) extends TlbBundle {
   val vpn = UInt(vpnLen.W)
}

val L2TlbMissQueueInDB, L2TlbMissQueueOutDB = Wire(new L2TlbMissQueueDB)
L2TlbMissQueueInDB.vpn := missQueue.io.in.bits.vpn
L2TlbMissQueueOutDB.vpn := missQueue.io.out.bits.vpn

val L2TlbMissQueueTable = ChiselDB.createTable(
        "L2TlbMissQueue_hart" + p(XSCoreParamsKey).HartId.toString, new L2TlbMissQueueDB)

L2TlbMissQueueTable.log(L2TlbMissQueueInDB, missQueue.io.in.fire, "L2TlbMissQueueIn", clock, reset)
L2TlbMissQueueTable.log(L2TlbMissQueueOutDB, missQueue.io.out.fire, "L2TlbMissQueueOut", clock, reset)
```

# ChiselDB

- **Hands-on: Analysis of Cache Coherence Violation**
  - Add ChiselDB
  - Inject bug
  - Compile and run
  - Analyze ChiselDB

# ChiselDB

- **Hands-on: Analysis of Cache Coherence Violation**

```
# Inject bug – We set all Release Data as a constant value
$ cd chiseldb && cat cc_err.patch
```

```
--- a/src/main/scala/huancun/noninclusive/SinkC.scala
+++ b/src/main/scala/huancun/noninclusive/SinkC.scala
@@ -93,7 +93,7 @@ class SinkC(implicit p: Parameters) extends
BaseSinkC {
-       buffer(insertIdx)(count) := c.bits.data
+       buffer(insertIdx)(count) := 0xABCDEF.U
```

# ChiselDB

- **Hands-on: Analysis of Cache Coherence Violation**

```
# simulate using pre-built emu (ChiselDB enabled)
$ bash step1-run.sh
```

```
# ./emu-cc-err -i $NOOP_HOME/ready-to-run/microbench.bin \
  --diff $NOOP_HOME/ready-to-run/riscv64-nemu-interpreter-so \
  --dump-db    # set this argument to dump data base
  2>linux.err
```

# ChiselDB

- **Hands-on: Analysis of Cache Coherence Violation**

```
# Analyze
$ bash step2-analyze.sh
```

```
# 1. sqlite query for all transactions on Eaddr
# 2. use script to parse TLLog
# sqlite3 $NOOP_HOME/build/*.db \
  "select * from TLLOG where ADDRESS=0x80008fc0" | \
  sh $NOOP_HOME/scripts/utils/parseTLLog.sh
```

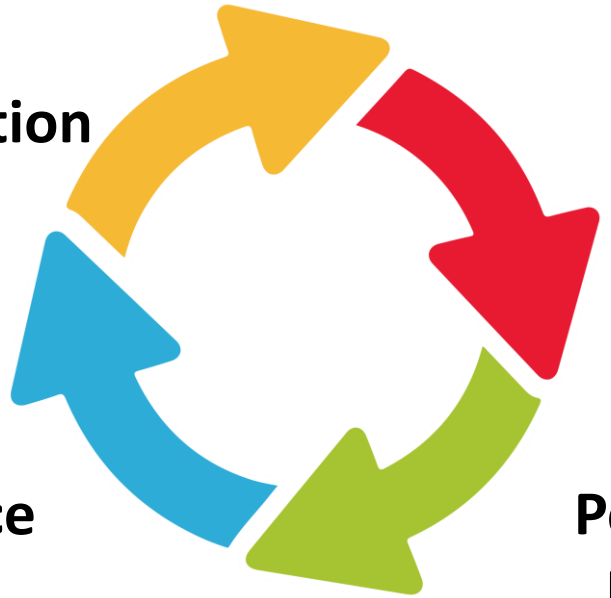
# ChiselDB:

```
# Result: [Time | To_From | Channel | Opcode | Permission | Address | Data(0)]

# Data successfully transferred from L1D to L2
186297 L2_L1D_0 C ReleaseData Shrink TtoN 80008fc0 000000000000000a

# But when L1D acquires addr again, data loaded from L2 is wrong
187022 L2_L1D_0 A AcquireBlock Grow NtoB 80008fc0
187029 L2_L1D_0 D GrantData Cap toT 80008fc0 0000000000abcdef

# So there must be something wrong when L2 records Release Data
```

# MinJie Performance Verification

Chisel-based prototypes

Implementation



Run Tests

VERILATOR

umd-memsys/
DRAMsim3

DRAMsim3: a Cycle-accurate, Thermal-Capable DRAM Simulator

XSPerf

Constantin

Performance Analysis

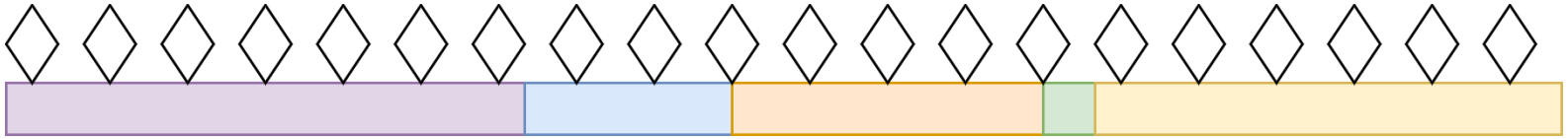
Performance Evaluation

RISC-V Checkpoint

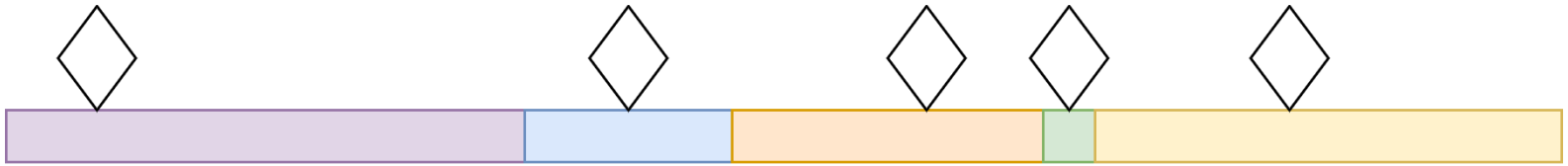
SimPoint

# Checkpoints

- Uniform Checkpoint
- Divide a program to small parts which can be simulated in parallel



- Simpoint Checkpoint
- Select representative parts from a program with cluster analysis

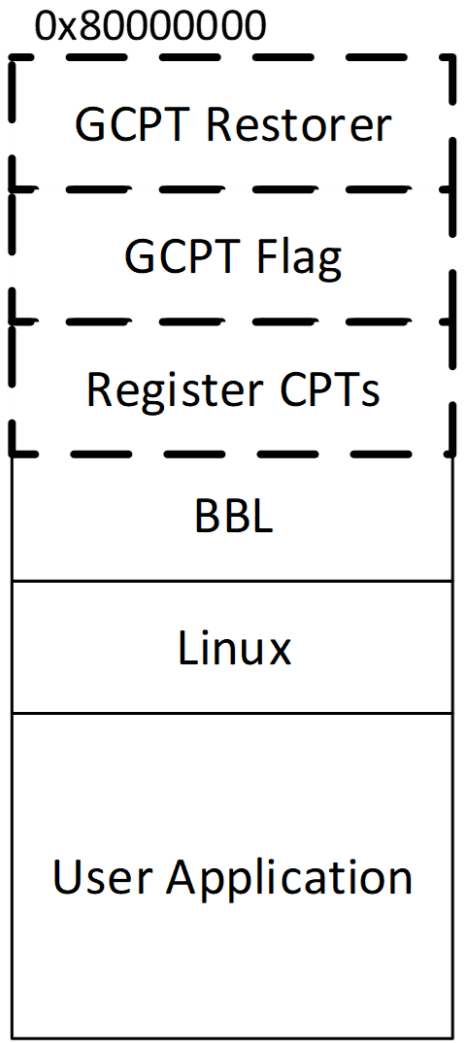


# Checkpoints: Uniform Checkpoint

- **step0: Prepare NEMU environment for checkpoints**

```
$ cd ../checkpoint          # tutorial/checkpoint
$ bash uniform-step0-prepare.sh
```

```
# uniform-step0-prepare.sh

# cd $NEMU_HOME

# git checkout asplos2023-tutorial
# git submodule update --init
# make clean
# make tutorial_defconfig
# make -j4

// generate gcpt restorer binary
# cd resource/gcpt_restore && make -j4
```




Basic format of RISC-V Checkpoint

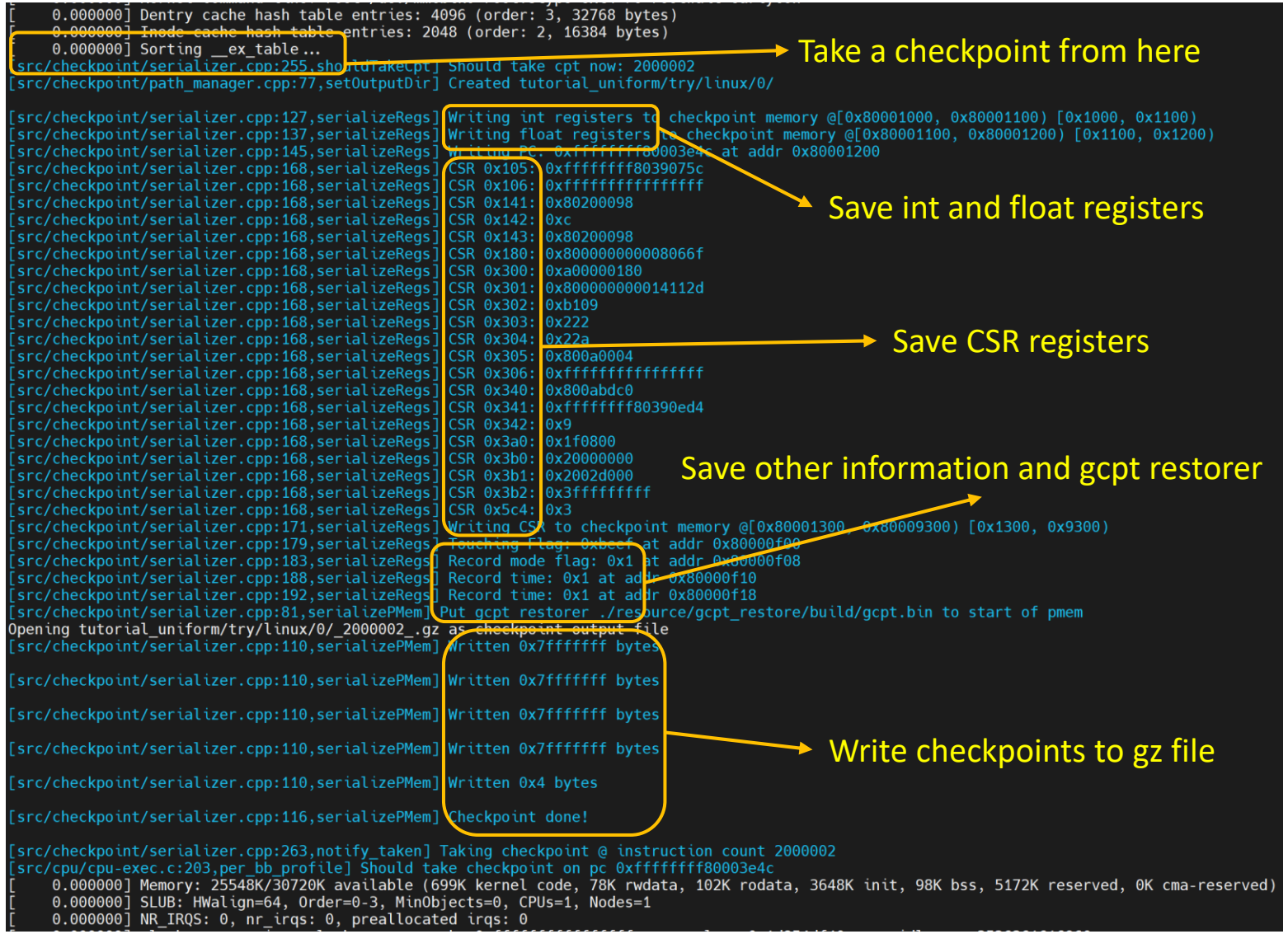
# Checkpoints: Uniform Checkpoint

- **step1: Generate uniform checkpoints in NEMU**

```
$ bash uniform-step1-gen.sh
```

```
# uniform-step1-gen.sh

# cd $NEMU_HOME
# rm -rf tutorial_uniform
# ./build/riscv64-nemu-interpreter
#     --cpt-interval 2000000   the profiling period for simpoint
#     -u                       uniformly take cpt with fixed interval
#     -b                       run with batch mode
#     -D tutorial_uniform      directory where the checkpoint is generated
#     -C try                   name of this task
#     -w linux                 name of workload
#     -r ./resource/gcpt_restore/build/gcpt.bin  binary of gcpt restorer
#     --dont-skip-boot         checkpoint immediately after boot
#     -I 3000000               max number of instructions executed
#     ./ready-to-run/linux-0xa0000.bin
```

# Checkpoints: Uniform Checkpoint



# Checkpoints: Uniform Checkpoint

- **step2: Run the uniform checkpoint in NEMU or XiangShan**
- Here we take NEMU as an example

```
$ bash uniform-step2-run-nemu.sh
```

```
# uniform-step2-run-nemu.sh

# cd $NEMU_HOME

# ./build/riscv64-nemu-interpreter
#     -b                    run with batch mode
#     --restore             restoring from CPT FILE
#     -I 1000000            max number of instructions executed
#     `find tutorial_uniform/try/linux/0/ -type f -name "*.gz" | tail -1`
                            path of the checkpoint files just generated
```

# Checkpoints: Uniform Checkpoint

```
[src/monitor/monitor.c:103,parse_args] Restoring from checkpoint
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'clint' at [0x0000000038000000, 0x000000003800ffff]
[src/isa/riscv64/init.c:100,init_isa] NEMU will start from pc 0x80000000
[src/monitor/image_loader.c:69,load_img] Loading Gcpt file form cmdline: tutorial_uniform/try/linux/0/_2000002_.gz

[src/monitor/image_loader.c:77,load_img] Loading GZ image tutorial_uniform/try/linux/0/_2000002_.gz
[src/monitor/image_loader.c:69,load_img] Loading Gcpt restorer form cmdline: (null)

[src/monitor/image_loader.c:71,load_img] No image is given. Use the default built-in image/restorer.
[src/device/io/port-io.c:30,add_pio_map] Add port-io map 'uartlite' at [0x00000000000003f8, 0x0000000000000404]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'uartlite' at [0x0000000040600000, 0x000000004060000c]
[src/device/io/port-io.c:30,add_pio_map] Add port-io map 'rtc' at [0x0000000000000048, 0x000000000000004f]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'rtc' at [0x00000000a1000048, 0x00000000a100004f]
[src/device/io/port-io.c:30,add_pio_map] Add port-io map 'screen' at [0x0000000000000100, 0x0000000000000107]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'screen' at [0x0000000040001000, 0x0000000040001007]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'vmem' at [0x0000000050000000, 0x00000000500752ff]
[src/device/io/port-io.c:30,add_pio_map] Add port-io map 'keyboard' at [0x0000000000000060, 0x0000000000000063]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'keyboard' at [0x00000000a1000060, 0x00000000a1000063]
[src/device/io/mmio.c:38,add_mmio_map] Add mmio map 'sdhci' at [0x0000000040002000, 0x000000004000207f]
[src/device/sdcard.c:137,init_sdcard] Can not find sdcard image:
[src/monitor/monitor.c:44,welcome] Debug: OFF
[src/monitor/monitor.c:49,welcome] Build time: 19:35:10, Mar 25 2023
Welcome to riscv64-NEMU!
For help, type "help"
[    0.000000] Memory: 25548K/30720K available (699K kernel code, 78K rwdata, 102K rodata, 3648K init, 98K bss, 5172K reserved, 0K cma-reserved)
[    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
[    0.000000] NR_IRQS: 0, nr_irqs: 0, preallocated irqs: 0
[    0.000000] clocksource: riscv_clocksource: mask: 0xffffffffffffffff max_cycles: 0x1d854df40, max_idle_ns: 3526361616960 ns
[    0.000000] console [hvc0] enabled
[    0.000000] console [hvc0] enabled
[    0.000000] bootconsole [early0] disabled
[    0.000000] bootconsole [early0] disabled
[    0.000000] Calibrating delay loop (skipped), value calculated using timer frequency.. 2.00 BogoMIPS (lpj=10000)
[    0.000000] pid_max: default: 4096 minimum: 301
[    0.000000] Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
[    0.000000] Mountpoint-cache hash table entries: 512 (order: 0, 4096 bytes)
[    0.010000] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 19112604462750000 ns
[    0.020000] clocksource: Switched to clocksource riscv_clocksource
```

Restore from checkpoint and load gz files

NEMU will start execution from the moment of that checkpoint

# Checkpoints: Simpoint Checkpoint

- **step0: Prepare NEMU environment for Simpoint checkpoint**

```
$ bash simpoint-step0-prepare.sh
```

```
# simpoint-step0-prepare.sh

# cd $NEMU_HOME

# git checkout cpt-bk
# git submodule update --init
# cd resource/simpoint/simpoint_repo/analysiscode && make simpoint -j4
                                                    generate simpoint generator binary
# cd resource/gcpt_restore && make -j4   generate gcpt restorer binary

# cd $NEMU_HOME
# make clean
# make ISA=riscv64 XIANGSHAN=1 -j4       compile NEMU
```

# Checkpoints: Simpoint Checkpoint

- **step1: Execute the workload and collect program behavior**

```
$ bash simpoint-step1-profiling.sh
```

```
# simpoint-step1-profiling.sh

# cd $NEMU_HOME
# rm -rf tutorial_simpoint

# ./build/riscv64-nemu-interpreter
#     -b                                  run with batch mode
#     -D $NEMU_HOME/tutorial_simpoint     directory where the checkpoint is generated
#     -C profiling                        name of this task
#     -w stream                           name of workload
#     --simpoint-profile                  is simpoint profiling
#     --interval 50000000                 simpoint interval instructions
#     $XS_PROJECT_ROOT/tutorial/bin/stream-0xa0000.bin
```

# Checkpoints: Simpoint Checkpoint

```
[src/isa/riscv64/exec/special.c,38,exec_nemu_trap] Start profiling, resetting inst count

-------------------------------------------------------------
STREAM version $Revision: 5.10 $
-------------------------------------------------------------
This system uses 4 bytes per array element.
-------------------------------------------------------------
Array size = 2000000 (elements), Offset = 0 (elements)
Memory per array = 7.6 MiB (= 0.0 GiB).
Total memory required = 22.9 MiB (= 0.0 GiB).
Each kernel will be executed 10 times.
 The *best* time for each kernel (excluding the first iteration)
 will be used to compute the reported bandwidth.
-------------------------------------------------------------
Your clock granularity/precision appears to be 2047 microseconds.
Each test below will take on the order of 102400 microseconds.
   (= 50 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:            710.2     0.023438     0.022528     0.024576
Scale:           181.7     0.089884     0.088064     0.092160
Add:             217.0     0.112868     0.110592     0.114688
Triad:           172.3     0.140402     0.139264     0.141312
-------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-06 on all three arrays
-------------------------------------------------------------
[src/monitor/cpu-exec.c,110,cpu_exec] nemu: HIT GOOD TRAP at pc = 0x0000002aaaaaa5f6
```

NEMU will do profiling while executing workload STREAM

# Checkpoints: Simpoint Checkpoint

- **step2: Cluster and obtain multiple representitive slices**

```
$ bash simpoint-step2-cluster.sh
```

```
# simpoint-step2-cluster.sh

# cd $NEMU_HOME

# mkdir -p $NEMU_HOME/tutorial_simpoint/cluster/stream
# export CLUSTER=$NEMU_HOME/tutorial_simpoint/cluster/stream

# ./resource/simpoint/simpoint_repo/bin/simpoint
#    -loadFVFile ./tutorial_simpoint/profiling/simpoint_bbv.gz  load a frequency vector file
#    -saveSimpoints $CLUSTER/simpoints0                         file to save simpoints
#    -saveSimpointWeights $CLUSTER/weights0                     file to save simpoint weights
#    -inputVectorsGzipped                                       input vectors have been compressed with gzip
#    -maxK 3                 maximum number of clusters to use
#    -numInitSeeds 2         times of different random initialization for each run, taking only the best clustering
#    -iters 1000             maximum number of iterations that should perform
#    -seedkm 123456          random seed for choosing initial k-means centers
#    -seedproj 654321        random seed for random linear projection
```

# Checkpoints: Simpoint Checkpoint

```
------------------------------------------------------------
Run number 3 of at most 4, k = 2
------------------------------------------------------------
  ------------------------------------------------------------
  Initialization seed trial #1 of 2; initialization seed = 123460
  ------------------------------------------------------------
  Initialized k-means centers using random sampling: 2 centers
  Number of k-means iterations performed: 1
  BIC score: 123.793
  Distortion: 1.61246
  Distortions/cluster: 1.54841 0.0640483
  Variance: 0.161246
  Variances/cluster: 1.54841 0.00711647
  ------------------------------------------------------------
  Initialization seed trial #2 of 2; initialization seed = 123461
  ------------------------------------------------------------
  Initialized k-means centers using random sampling: 2 centers
  Number of k-means iterations performed: 1
  BIC score: 123.793
  Distortion: 1.61246
  Distortions/cluster: 0.0640483 1.54841
  Variance: 0.161246
  Variances/cluster: 0.00711647 1.54841
  The best initialization seed trial was #1

------------------------------------------------------------
------------------------------------------------------------
Post-processing runs
------------------------------------------------------------
------------------------------------------------------------
  For the BIC threshold, the best clustering was run 3 (k = 2)
  Post-processing run 3 (k = 2)
```

Compute the information of K-means clustering

Obtain multiple program representitive checkpoints

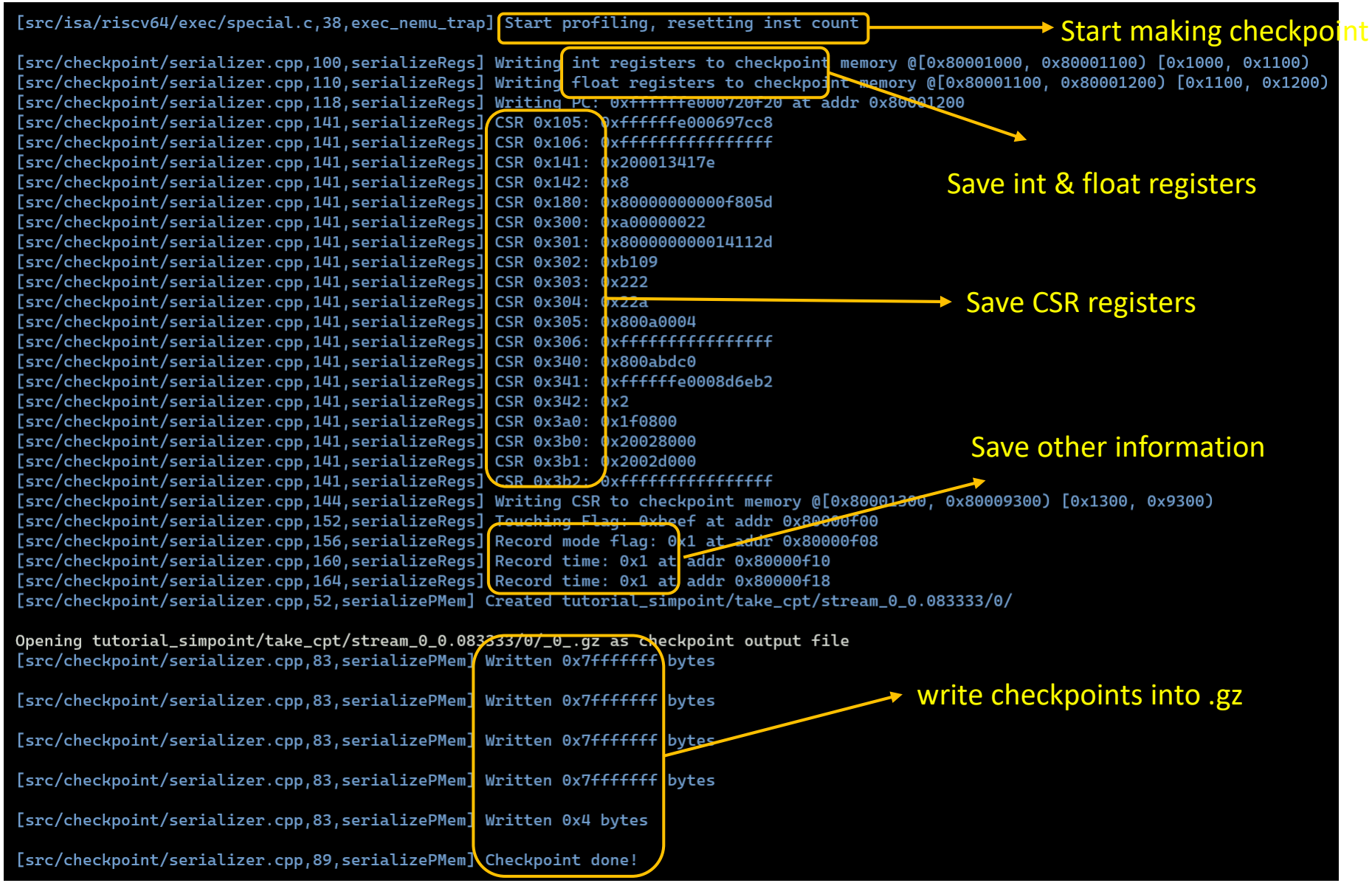
# Checkpoints: Simpoint Checkpoint

- **step3: Generate corresponding Checkpoints based on clustering results**

```
$ bash simpoint-step3-take-cpt.sh          # It may take about 3 minutes
```

```
# simpoint-step3-take-cpt.sh

# cd $NEMU_HOME
# rm -rf $NEMU_HOME/tutorial_simpoint/stream/take_cpt

# ./build/riscv64-nemu-interpreter
#     -b                                  run with batch mode
#     -D tutorial_simpoint                directory where the checkpoint is generated
#     -C take_cpt                         name of this task
#     -w stream                           name of workload
#     -S ./tutorial_simpoint/cluster      simpoints dir
#     --checkpoint-interval 50000000      simpoint interval instructions
#     $XS_PROJECT_ROOT/tutorial/bin/stream-0xa0000.bin
```

# Checkpoint: Simpoint Checkpoint



# Checkpoints: Simpoint Checkpoint

- **step4: Run the simpoint checkpoint in NEMU or XiangShan**
- Here we take XiangShan as an example

```
$ bash simpoint-step4-run-xs.sh
```

```
# simpoint-step4-run-xs.sh

$XS_PROJECT_ROOT/tutorial/emu
    -i `find $NEMU_HOME/tutorial_simpoint/ -type f -name "*.gz" | tail -1`
    --diff $NOOP_HOME/ready-to-run/riscv64-nemu-interpreter-so
    --max-cycles=100000
    2>simpoint.err
```

# Simulation Performance Counter

- **Two common types:**
  - **Accumulation**： basic accumulator counter
  - **Histogram**： count the distribution

**Accumulation:**

*[PERF][time=10000] ctrlBlock.rob: commitInstr, 1500*

*[PERF][time=10000] ctrlBlock.rob: waitLoadCycle, 800*

**Histogram:**

*[PERF][time=10000] l2cache: acquire_period_10_20, 15*

*[PERF][time=10000] l2cache: acquire_period_20_30, 200*

*[PERF][time=10000] l2cache: acquire_period_30_40, 60*

# Simulation Performance Counter

- **Usage:**
  - Add *XSPerfAccumulate('name', signal)* or *Histogram* wherever you want
  - **By default**, it is printed after simulation finishes
  - Set *DebugOptions(EnablePerfDebug = false)* to turn off
- **Showcase**

```
$ cd ..
$ bash run-emu-perf.sh
```

```
# ./emu -i hello.bin \
  --diff $NEMU_HOME/build/riscv64-nemu-interpreter-so 2>perf.err
```

```
$ less perf.err
```

# Simulation Performance Counter

```
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.ctrlBlock.decode: utilization,                1183
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.bpu.predictors.components_3.tables_0.wrbypass: wrbypass_hit,                    0
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.icache.missUnit.entries_1: entryPenalty1,                  714
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.ctrlBlock.decode: waitInstr,                414
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.bpu.predictors.components_3.tables_0.wrbypass: wrbypass_miss,                    0
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.ctrlBlock.decode: stall_cycle,                393
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.icache.missUnit.entries_1: entryReq1,                   14
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.bpu.predictors.components_3.tables_1.wrbypass: wrbypass_hit,                    0
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend: FrontendBubble,               4085
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.icache.missUnit.entries_0: entryPenalty0,                  248
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.bpu.predictors.components_3.tables_1.wrbypass: wrbypass_miss,                    2
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: utilization,                1826
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.icache.missUnit.entries_0: entryReq0,                    6
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: util_0_1,               2029
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: util_1_2,                541
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.misc.busPMU: dcache_bank_0_A_channel_fire,                      8
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: util_2_3,                 45
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.misc.busPMU: dcache_bank_0_A_channel_stall,                      0
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: util_3_4,                107
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.exuBlocks.scheduler.rs_4.staRS_0.oldestSelection: oldest_override_0,                    0
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.misc.busPMU: dcache_bank_0_A_channel_PutFullData_fire,                    0
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: util_4_5,                 44
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.exuBlocks.scheduler.rs_4.staRS_0.oldestSelection: oldest_same_as_selected_0,                    0
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.misc.busPMU: dcache_bank_0_A_channel_PutFullData_stall,                    0
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: util_5_6,                 85
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.exuBlocks.scheduler.rs_4.staRS_0.oldestSelection: oldest_override_1,                    0
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.misc.busPMU: dcache_bank_0_A_channel_PutPartialData_fire,                    0
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: util_6_7,                 13
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.exuBlocks.scheduler.rs_4.staRS_0.oldestSelection: oldest_same_as_selected_1,                    0
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.misc.busPMU: dcache_bank_0_A_channel_PutPartialData_stall,                    0
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: util_7_8,                 27
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.memBlock.dtlb_st_tlb_st.entries: port0_np_sp_multi_hit,                    0
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.misc.busPMU: dcache_bank_0_A_channel_ArithmeticData_fire,                    0
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: full,                862
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.memBlock.dtlb_st_tlb_st.entries: port1_np_sp_multi_hit,                    0
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.misc.busPMU: dcache_bank_0_A_channel_ArithmeticData_stall,                    0
[PERF ][time=               2893] TOP.SimTop.l_soc.core_with_l2.core.frontend.ibuffer: exHalf,                  125
```

*Find more parsing scripts at https://github.com/OpenXiangShan/env-scripts/blob/main/perf/perf.py*

# Top-Down: Method for Performance Analysis

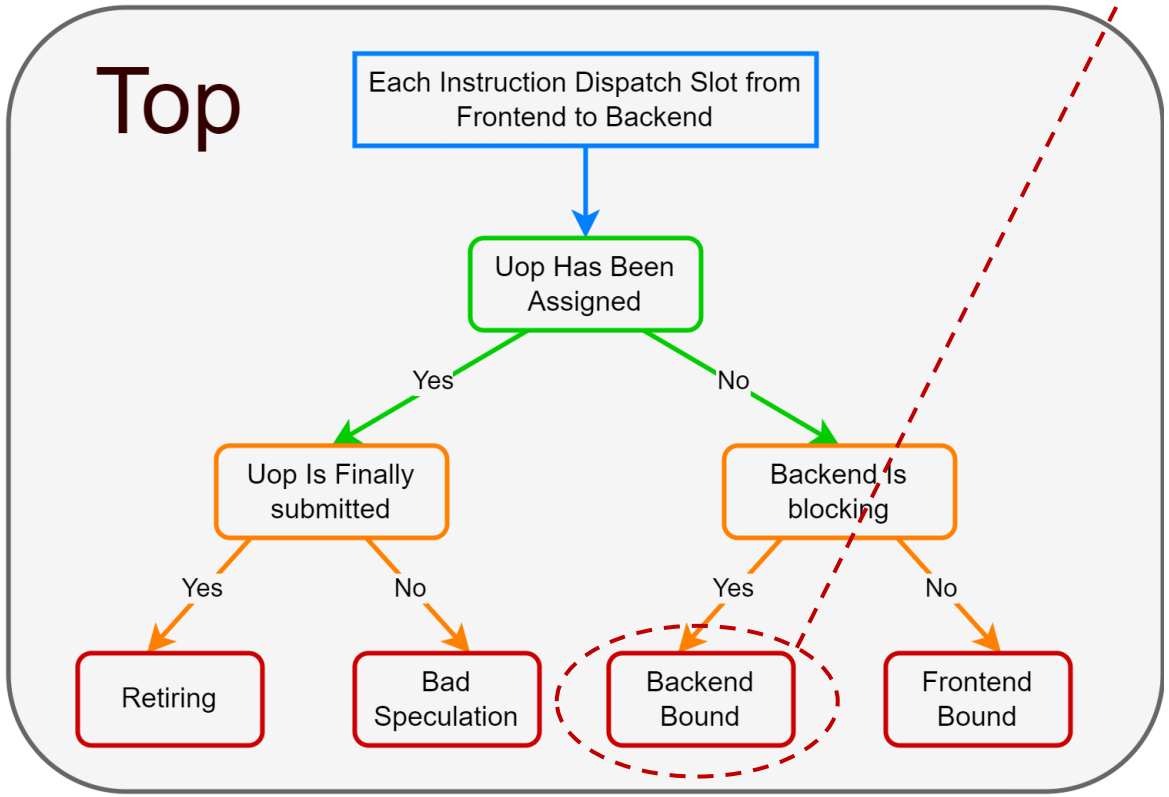
- Organize scattered performance events in a **hierarchical manner**
- **Accurately** calculate one event's impact on processor performance

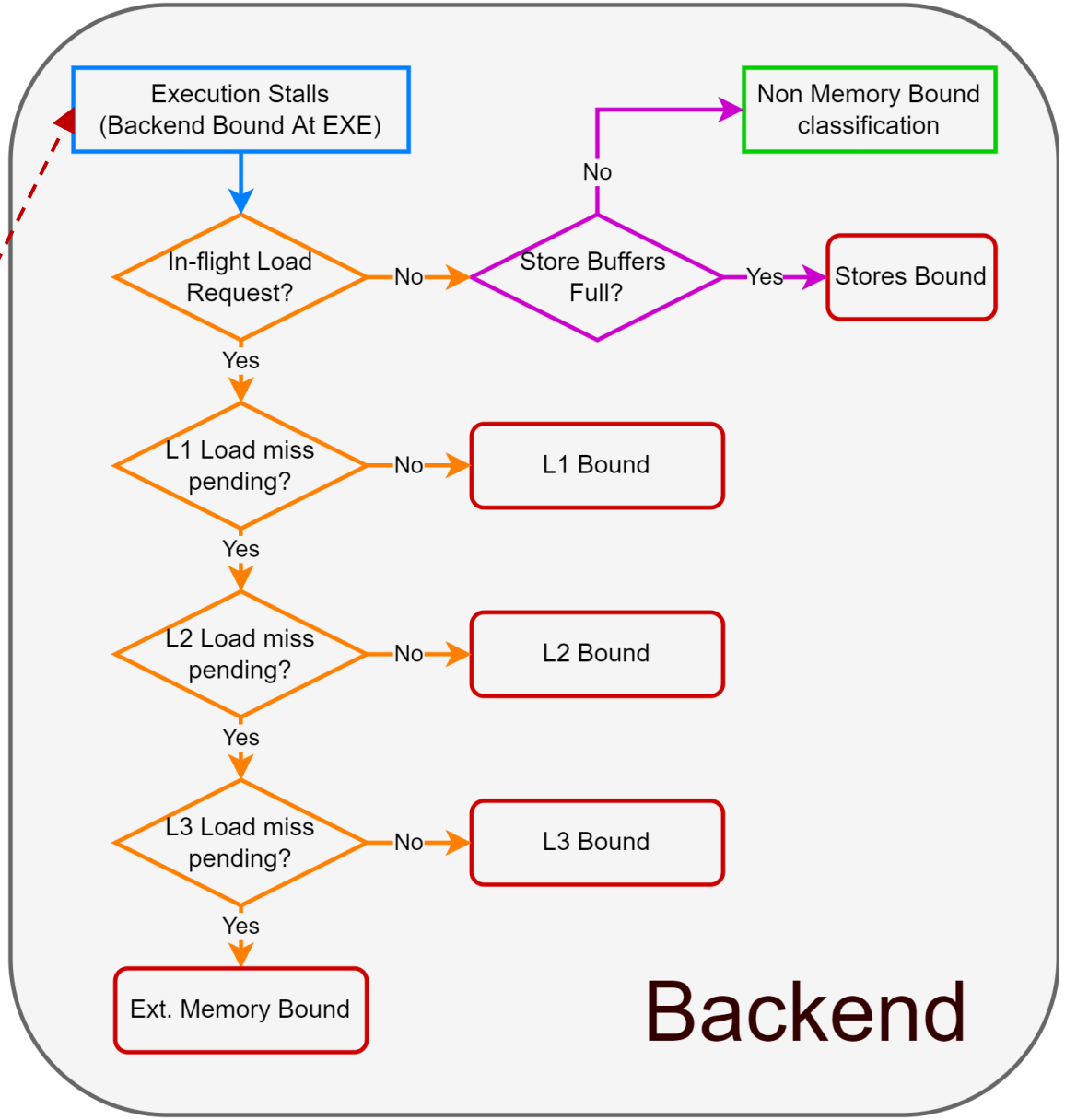
- We apply the Top-Down approach to XiangShan
  - Complete **targeted optimization adaptation** for RISC-V instruction set
  - Optimize **performance counters** according to XiangShan microarchitecture
  - Further refine the **hierarchical design** of Top-Down model
  - Without missing or duplicating any events
  - Without assuming the blocking cycles of performance events in advance

- Due to the limited time, we will not arrange a hands-on section here
  - Please refer to https://github.com/OpenXiangShan/XiangShan/tree/master/scripts/top-down

# Top-Down

- Use Top and Backend as examples



# Top-Down

- Setting Performance Counters in RTL Code

*Dispatch.scala*

```
val stallReason = Wire(chiselTypeOf(io.stallReason.reason))
// ...
TopDownCounters.values.foreach(ctr =>
  XSPerfAccumulate(ctr.toString(), PopCount(stallReason.map(_ === ctr.id.U)))
)
```

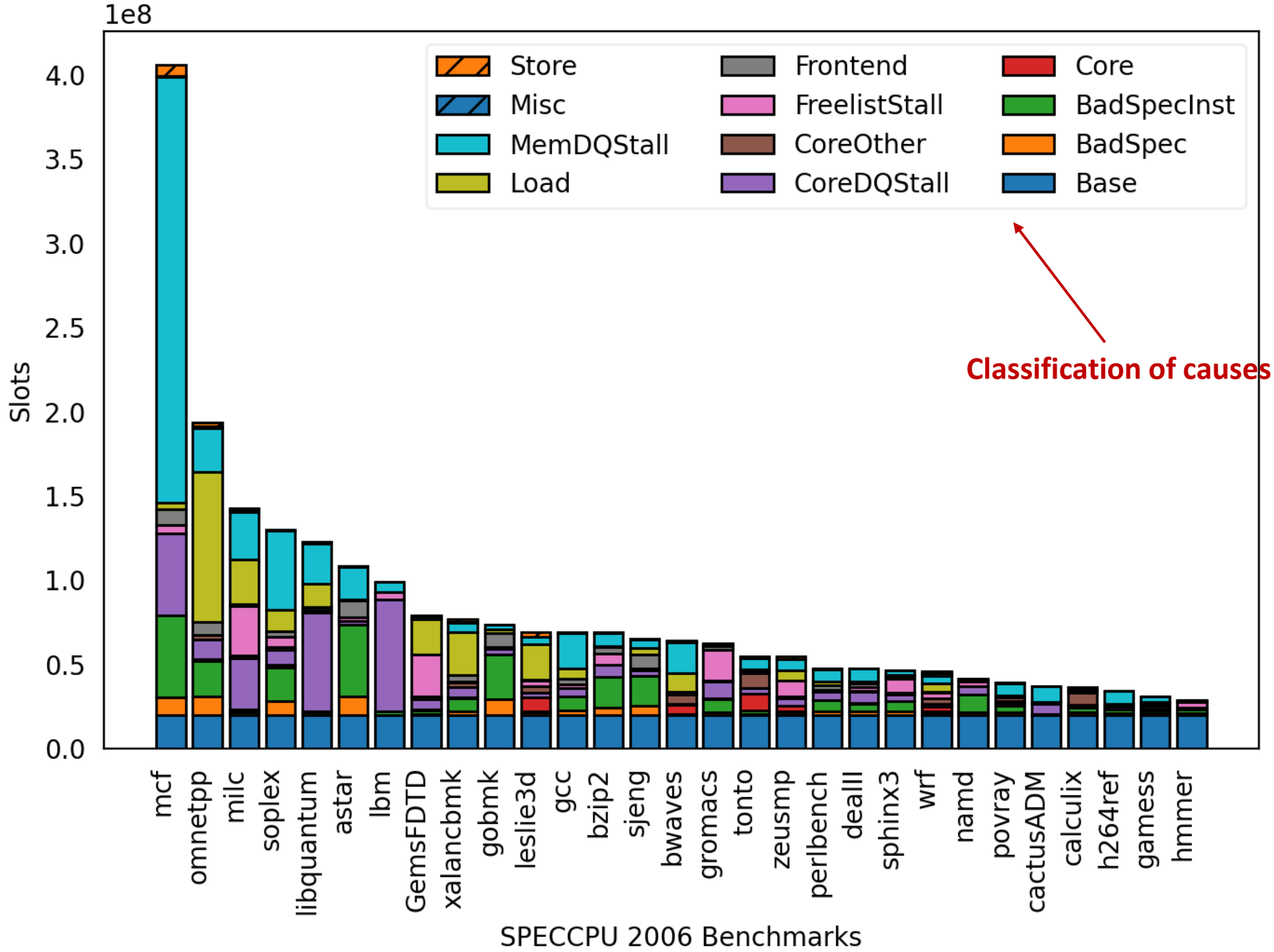
- Do simulation and collect performance counter data

# Top-Down

- Analyze through Top-Down method

```
env-scripts: perf/top_down/configs.py


xs_coarse_rename_map = {
  'OverrideBubble': 'MergeFrontend',
  'FtqFullStall': 'MergeFrontend',
  ‘IntDqStall': 'MergeCoreDQStall',  # attribute perf-counters to Top-Down hierarchy
  'FpDqStall': 'MergeCoreDQStall'
}
```

- Obtain analysis results and make targeted optimizations



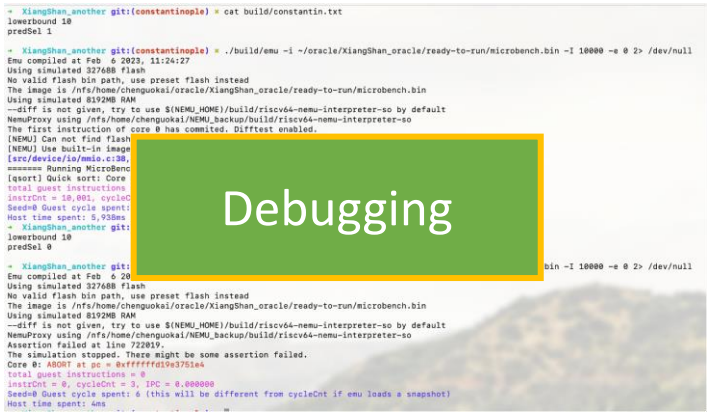

**Classification of causes of congestion**

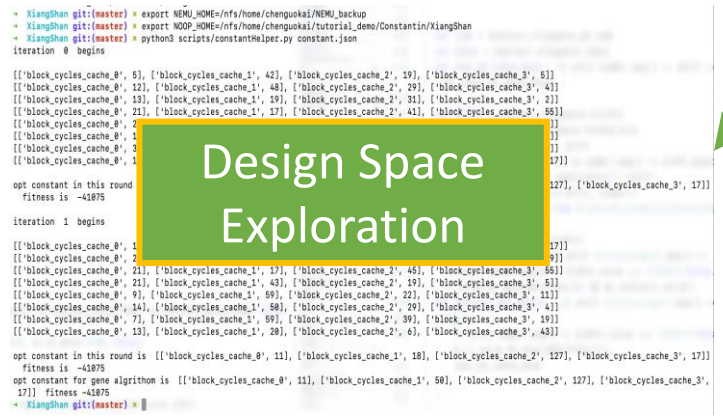
# ConstantIn

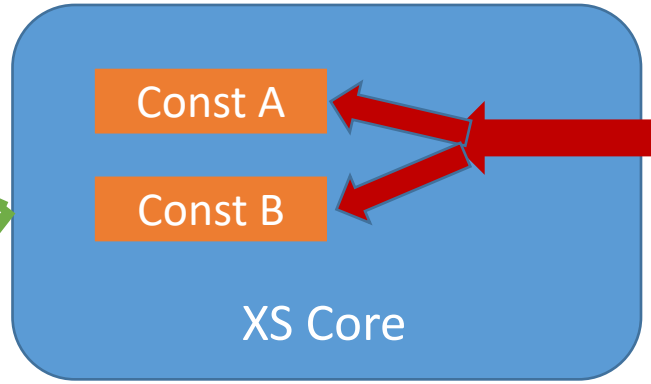
- **Motivation**
  - Wanna test the performance under different parameters
  - But re-compilation is **time consuming**
  - Can we change parameters(constants) at runtime?
- **We present Constantin** (Constant - in)
- **Design/Highlights:**
  - DPI-C： Using C++ function in Chisel code (through Verilog using BlackBox)
  - Signal values configured at runtime rather than compile time

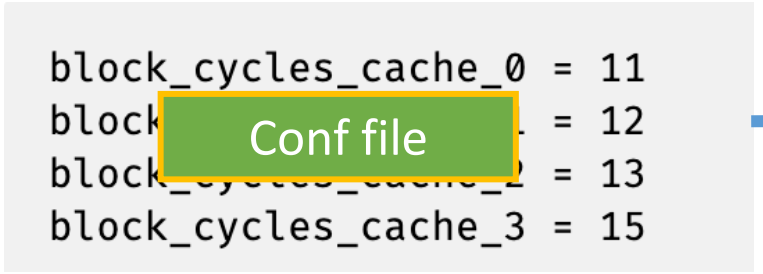
# Constantin

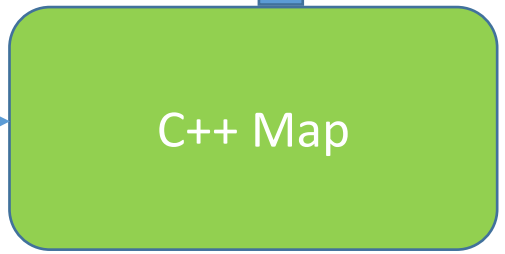
- **Constants keep constant only during runtime**





# Constantin

- **Usage:** Create signal
- XiangShan/coupledL2/src/main/scala/coupledL2/prefetch/TemporalPrefetch.scala

```
139   require(cacheParams.hartIds.size == 1)
140   val hartid = cacheParams.hartIds.head
141   // 0 / 1: whether to enable temporal prefetcher
142   private val enableTP = WireInit(Constantin.createRecord("enableTP"+hartid.toString, initValue = 1.U))
143   // 0 ~ N: throttle cycles for each prefetch request
144   private val tpThrottleCycles = WireInit(Constantin.createRecord("tp_throttleCycles"+hartid.toString, initValue = 4.U(3.W)))
145   // 0 / 1: whether request to set as trigger on meta hit
146   private val hitAsTrigger = WireInit(Constantin.createRecord("tp_hitAsTrigger"+hartid.toString, initValue = 1.U))
147   // 1 ~ triggerQueueDepth: enqueue threshold for triggerQueue
148   private val triggerThres = WireInit(Constantin.createRecord("tp_triggerThres"+hartid.toString, initValue = 1.U(3.W)))
149   // 1 ~ tpEntryMaxLen: record threshold for recorder and sender (storage size will not be affected)
150   private val recordThres = WireInit(Constantin.createRecord("tp_recordThres"+hartid.toString, initValue = tpEntryMaxLen.U))
151   // 0 / 1: whether to train on vaddr
152   private val trainOnVaddr = WireInit(Constantin.createRecord("tp_trainOnVaddr"+hartid.toString, initValue = 0.U))
153   // 0 / 1: whether to eliminate L1 prefetch request training
154   private val trainOnL1PF = WireInit(Constantin.createRecord("tp_trainOnL1PF"+hartid.toString, initValue = 0.U))
```

# Constantin

- **Hands-on:** Pass constant via standard input stream

```
$ cd constantin                    # now in tutorial/constantin
```

```
# Build emu with constantin (time consuming, use pre-built emu instead)
# bash step0-build.sh
```

```
# Run emu and pass constant
$ bash step1-basic.sh
```

```
please input total constant number
```

```
2
```

```
please input each constant ([constant name] [value])
```

```
tp_recordThres0 14
tp_throttleCycles0 3
```

# Constantin

- **Hands-on:** Pass constant via standard input stream

```
~/tu/test/t/constantin tutorial-new = ?4 > bash ./step1-basic.sh
emu-constantin compiled at Oct 28 2023, 17:38:43
please input total constant number
2
please input each constant ([constant name] [value])
tp_recordThres0 14
tp_throttleCycles0 3
Using simulated 32768B flash
isWriteICacheTable0 = 0
isWriteFetchToIBufferTable0 = 0
isWriteIfuWbToFtqTable0 = 0
isWritePrefetchPtrTable0 = 0
isWriteFTQTable0 = 0
isWriteBankConflictTable0 = 0
isWriteL1MissQMissTable0 = 0
isWriteLoadMissTable0 = 0
isFirstHitWrite0 = 0
isWriteLoadAccessTable0 = 0
isWriteL2TlbPrefetchTable0 = 0
isWriteL1TlbTable0 = 0
isWritePageCacheTable0 = 0
isWritePTWTable0 = 0
isWriteL2TlbMissQueueTable0 = 0
CorrectMissTrain0 = 0
isWriteInstInfoTable0 = 0
l1_stride_ratio0 = 2
l2_stride_ratio0 = 5
tp_trainOnL1PF0 = 0
tp_hitAsTrigger0 = 1
tp_triggerThres0 = 1
enableTP0 = 1
enableL3StreamPrefetch0 = 0
enableL1StreamPrefetcher0 = 1
nMaxPrefetchEntry0 = 14
always_update0 = 1
ForceWriteUpper_0 = 60
ForceWriteLower_0 = 55
tp_recordThres0 = 14
tp_throttleCycles0 = 3
tp_trainOnVaddr0 = 0
StoreWaitThreshold_0 = 0
ColdDownThreshold_0 = 12
enableDynamicPrefetcher0 = 1
StoreBufferThreshold_0 = 7
StoreBufferBase_0 = 4
l2DepthRatio0 = 2
l3DepthRatio0 = 3
depth0 = 32
Using simulated 8192MB RAM
The image is ./ready-to-run/linux.bin
The reference model is ./ready-to-run/riscv64-nemu-interpreter-so
The first instruction of core 0 has commited. Difftest enabled.
Core 0: EXCEEDING CYCLE/INSTR LIMIT at pc = 0x80000e4c
instrCnt = 1,000, cycleCnt = 10,235, IPC = 0.097704
Seed=0 Guest cycle spent: 10,239 (this will be different from cycl
Host time spent: 44,688ms
```

```
please input total constant number
2
please input each constant ([constant name] [value])
tp_recordThres0 14
tp_throttleCycles0 3
```

```
ForceWriteUpper_0 = 60
ForceWriteLower_0 = 55
tp_recordThres0 = 14
tp_throttleCycles0 = 3
tp_trainOnVaddr0 = 0
StoreWaitThreshold_0 = 0
ColdDownThreshold_0 = 12
enableDynamicPrefetcher0 = 1
```

# Constantin

- **Feature:** Auto Solving of better constant values
- **Process**
  - Enable AutoSolving & prepare signals
  - Compile
  - Run basic demonstration
  - Set parameters for AutoSolving
  - Auto solve
  - Cleanup

# Constantin

- **Feature:** Auto Solving of better constant values

```
# Automatical parameter solver configuration file
$ cat constantin.json
```

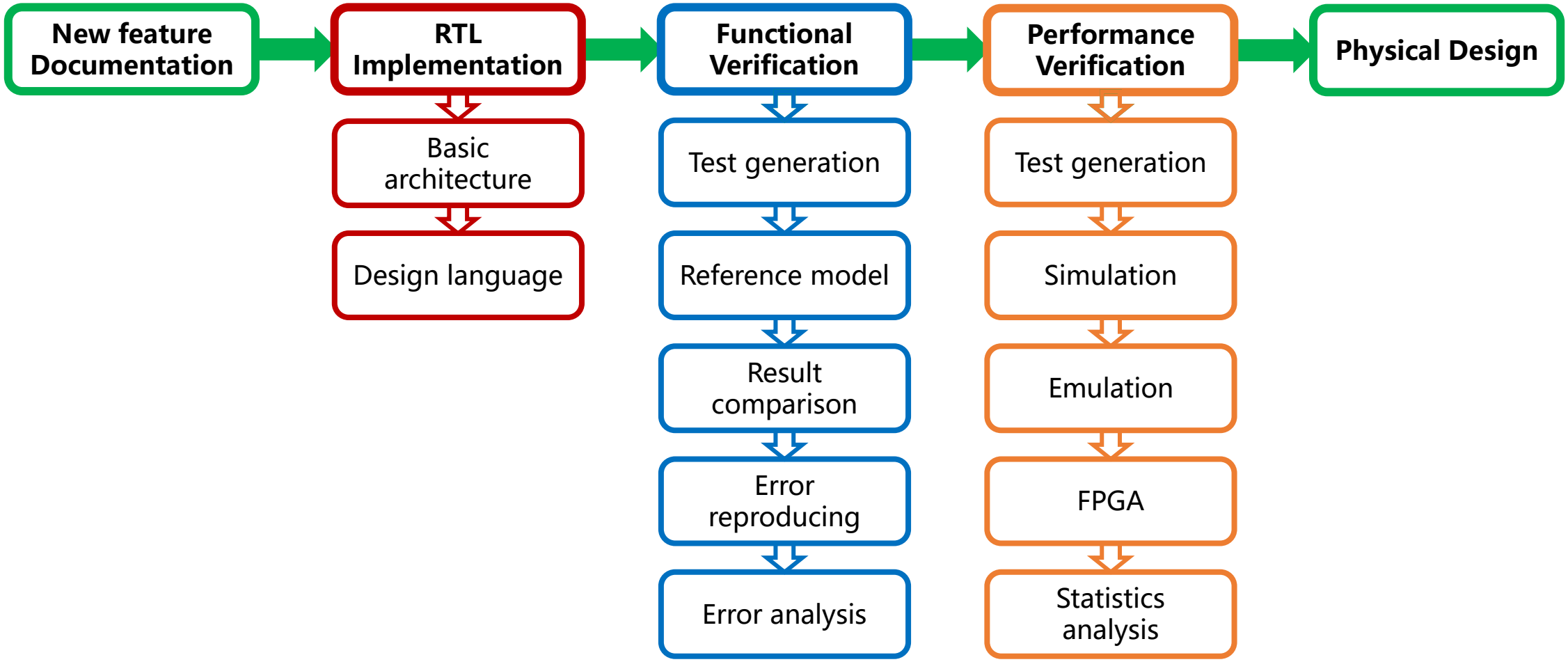
- **Parameters defined**
  - properties of signals
  - target of Auto Solving
  - parameters of genetic algorithm
  - parameters of XS simulator

# Constantin

- **Usage: Auto Solving of better constant values**

```
# Run auto solver, output will be the optimal constant currently found
$ bash step2-solve.sh


# The solver generates optimal parameters
opt constant for gene algrithom is
[['block_cycles_cache_0', XXX], ['block_cycles_cache_1', XXX],
 ['block_cycles_cache_2', XXX],  ['block_cycles_cache_3', XXX]]  fitness 0
```

# Summary: MinJie Development Flows and Tools



# Thanks!

*This tutorial is organized by Xi Chen, Zefan Xu and Haojin Tang.*