



# XS-Gem5 Simulator

---

The XiangShan Team  
HPCA 26 @ Sydney, Australia  
January 31, 2026

## What we will cover in this tutorial

- **Introduction:**
- **Background:** Why build such a simulator?
- **Calibration & Architecture:** How is the simulator designed?
- **Development Tools:** What tools are we using at the current stage to improve development efficiency?
- **Case Study:** A concrete example demonstrating how the simulator is used.

# What we will cover in this tutorial

- **Introduction:**
- Background: Why build such a simulator?
- Calibration & Architecture: How is the simulator designed?
- Development Tools: What tools are we using at the current stage to improve development efficiency?
- Case Study: A concrete example demonstrating how the simulator is used.

# Introduction

- **XS-Gem5 Architectural Simulator**
  - Built on the open-source GEM5 simulator
  - Simulation speed: ~100 KIPS
  - supports V & H
  - An **open-source, industrial-grade** architecture simulator
  - Enables rapid design space exploration

# Introduction

## • Features

- Se simulation
- Full-system simulation
  - RVGCpt: XiangShan's RISC-V generic checkpoint
  - Estimated **SPECCPU 2006 & 2017** with **SimPoint**
  - Online Difttest

## • Frontend

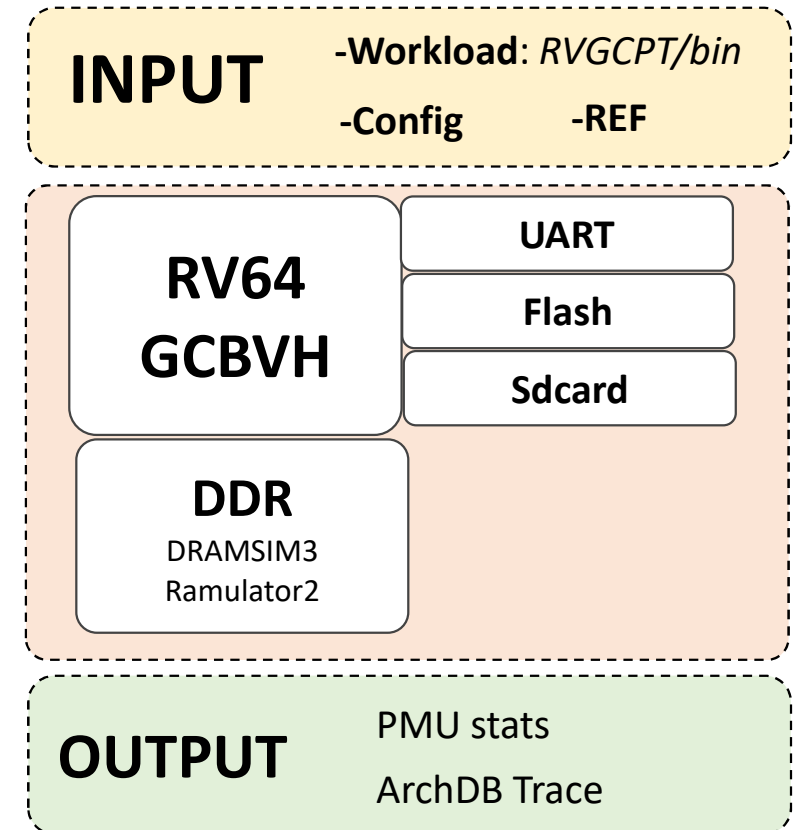
- micro-architecture calibration

## • Backend

- latency/throughput calibration
- Pipeline & Issue queue & Regfile

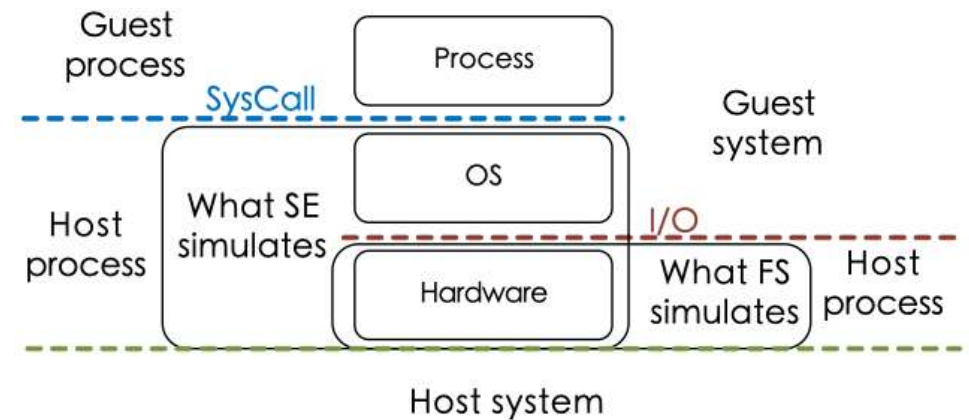
## • Memory system

- Prefetchers
- Parallel PTW / L2 TLB / TLB prefetching
- Memory bandwidth and memory controller calibration
- 1MB L2
- 16MB L3
- DDR4 **OR** DDR5



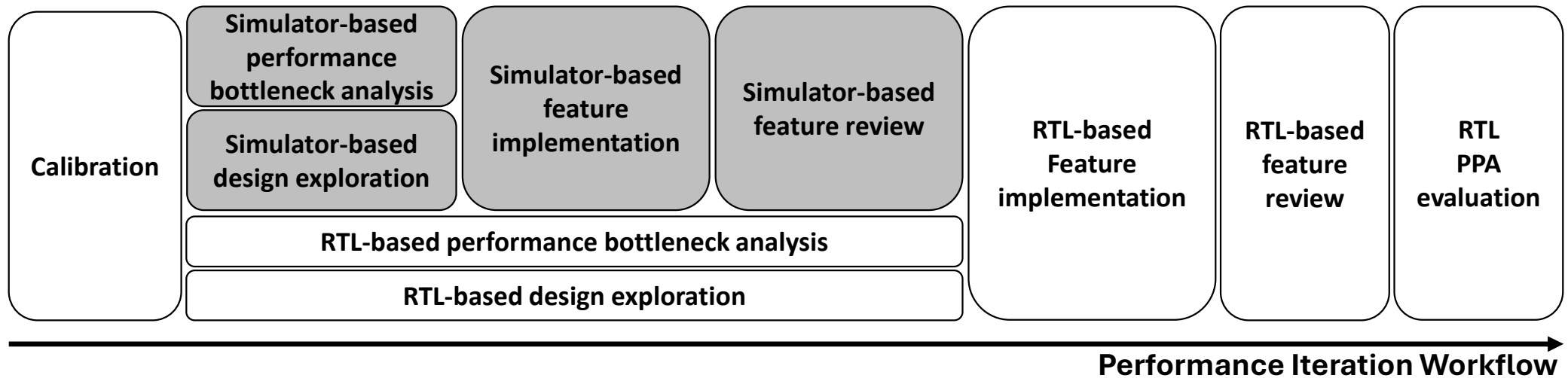
# Support Full-system & Se simulation

- **Se mode**
  - Support part microbenchmark
- **Full-system**
  - Avoid hacky system calls in SE (system call emulation)
  - **Accurate system call modeling**
    - Performance of serializing instructions
  - **Accurate virtual memory behavior**
    - Random VA → PA mapping
    - Accurate TLB/PTW modelling
    - Recommend using fs.



# ❖ Performance Exploration Workflow on an Architectural Simulator

- The results obtained from architectural simulator-based exploration should provide **guidance** for performance iteration on XiangShan.
- Findings derived from a calibrated simulator are more likely to be validated on RTL.
- Performance iteration workflow guided by the simulator:



## What we will cover in this tutorial

- Introduction:
- **Background: Why build such a simulator?**
- Calibration & Architecture: How is the simulator designed?
- Development Tools: What tools are we using at the current stage to improve development efficiency?
- Case Study: A concrete example demonstrating how the simulator is used.

## **Background - Rapid performance iteration**

- Three generations of architectural upgrades within three years:
  - Yanqihu → Nanhu → Kunminghu
  - SPEC CPU 2006: 7 /GHz → 10 /GHz → 15 /GHz
- High development efficiency is achieved by architectural simulation co-design
  - Flexible, easy to modify
  - Fast simulation time
  - Enable evaluation of new features prior to RTL implementation
  - Enable evaluation of optimal configurations in vast design space



# Background - Pitfalls in simulation methodologies

“While absolute accuracy for simulator may be low, relative performance trend can be accurate”

**The assumption that "the overall trend remains reliable" does not always hold true [1]**

## Problem: The “trend myth” of simulators

### Performance gain:

GEM5: BOP < SMS

XS-RTL: BOP = SMS

- High-performance processors have complex microarchitectural details.
- Uncalibrated simulators fail to reflect the correct design trends.

• Uncalibrated simulators → **fail to reflect performance characteristics:**

- Misleading performance insights
- Wasted engineering efforts
- Low return-on-cost design

**Calibration is crucial for the simulator.**

## What we will cover in this tutorial

- Introduction:
- Background: Why build such a simulator?
- **Calibration & Architecture: How is the simulator designed?**
- Development Tools: What tools are we using at the current stage to improve development efficiency?
- Case Study: A concrete example demonstrating how the simulator is used.



# Microbenchmarks for micro-arch calibration

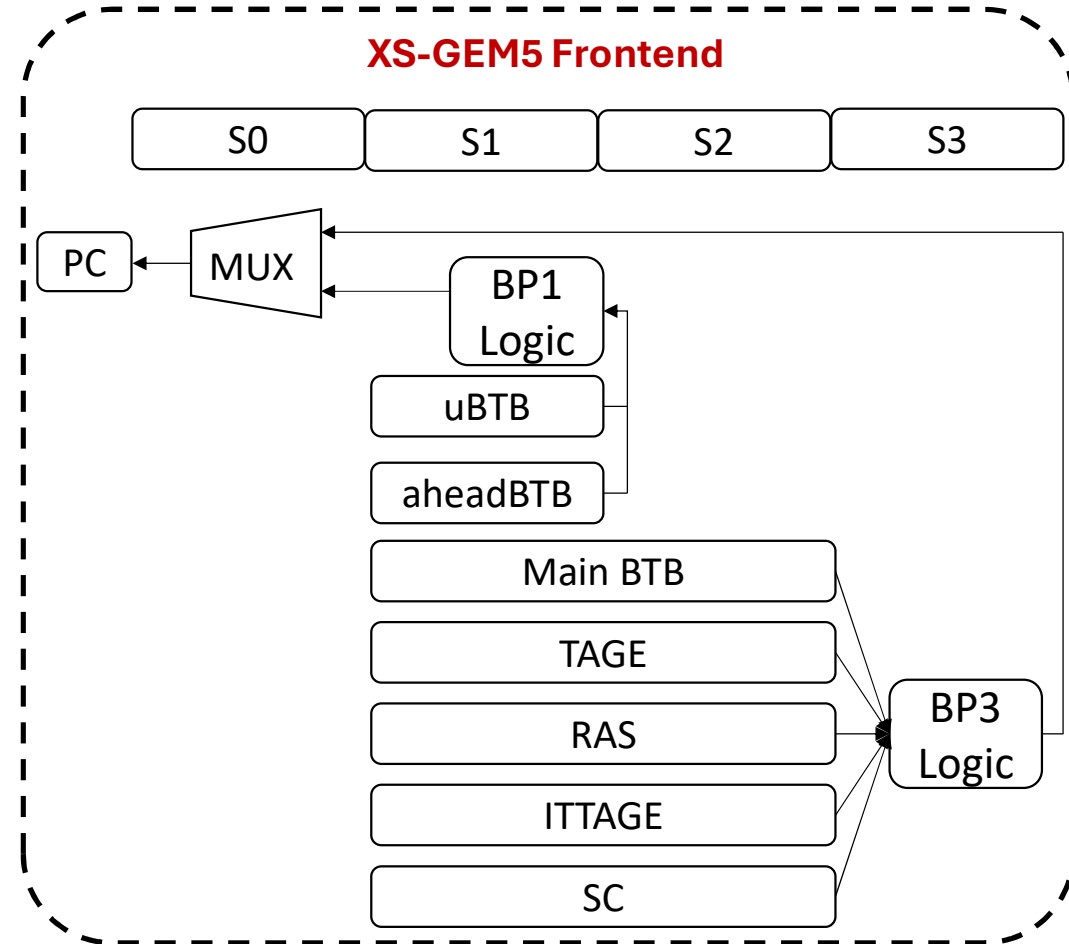
- **Principle:** Calibrate the architectural model by extracting key microarchitectural parameters of the processor through external observation.
- **Work:** Designed and implemented a set of microbenchmarks for simulator RTL calibration.

| Pipeline  | Cache   | Virtual mem/Prefetch  | Branch prediction  | Uncalibrated point   |
|---|---|---|--|--|
| <ul style="list-style-type: none"><li>• Instruction operation latency</li><li>• Operand-dependent division operations, scheduling latency</li><li>• Instruction scheduling latency</li><li>• Decode bandwidth</li><li>• Rename bandwidth</li><li>• Dispatch bandwidth</li></ul> | <ul style="list-style-type: none"><li>• Load-to-use latency (L1/L2/L3 hit)</li><li>• Load-Load schedule latency (L1/L2/L3hit)</li><li>• L1/L2/L3 access latency</li><li>• L1/L2/L3 access bandwidth</li><li>• Stream store access pattern</li></ul> | <ul style="list-style-type: none"><li>• L2 TLB access latency</li><li>• L2 TLB access latency to cache and memory at all levels</li><li>• Stride Prefetch</li><li>• Stream Prefetch</li></ul> | <ul style="list-style-type: none"><li>• Penalty of flushing instructions after branch misprediction</li><li>• Long instruction sequence test without jumps</li></ul> | <ul style="list-style-type: none"><li>• Ideal dispatch logic</li><li>• Wrong division latency and operand relationship</li><li>• Wrong CSR_MINSTRET in RTL</li><li>• Wrong store-load features</li><li>• Conflict detection logic</li><li>• Inconsistent memory access latency</li><li>• Wrong prefetch parameters</li><li>• ...</li></ul> |

**Infrastructure: the entire process of automated test running, and reporting results**

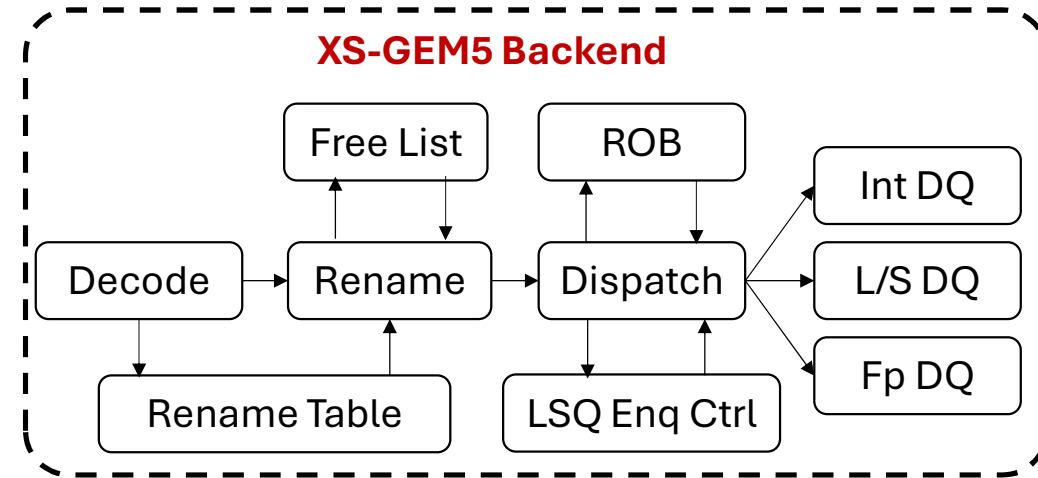
# Frontend

- **Decoupled Frontend**
  - Hides latency and boosts pipeline efficiency
- **Advanced Predictor Suite**
  - Integrates TAGE, SC, and others
  - TAGE
    - Expanded from 4 to 10 tables
    - History up to 500 bits
  - SC
    - Six correction tables
- **Highly scalable**
  - Supports multiple decoupled branch predictors
  - Configurable latency

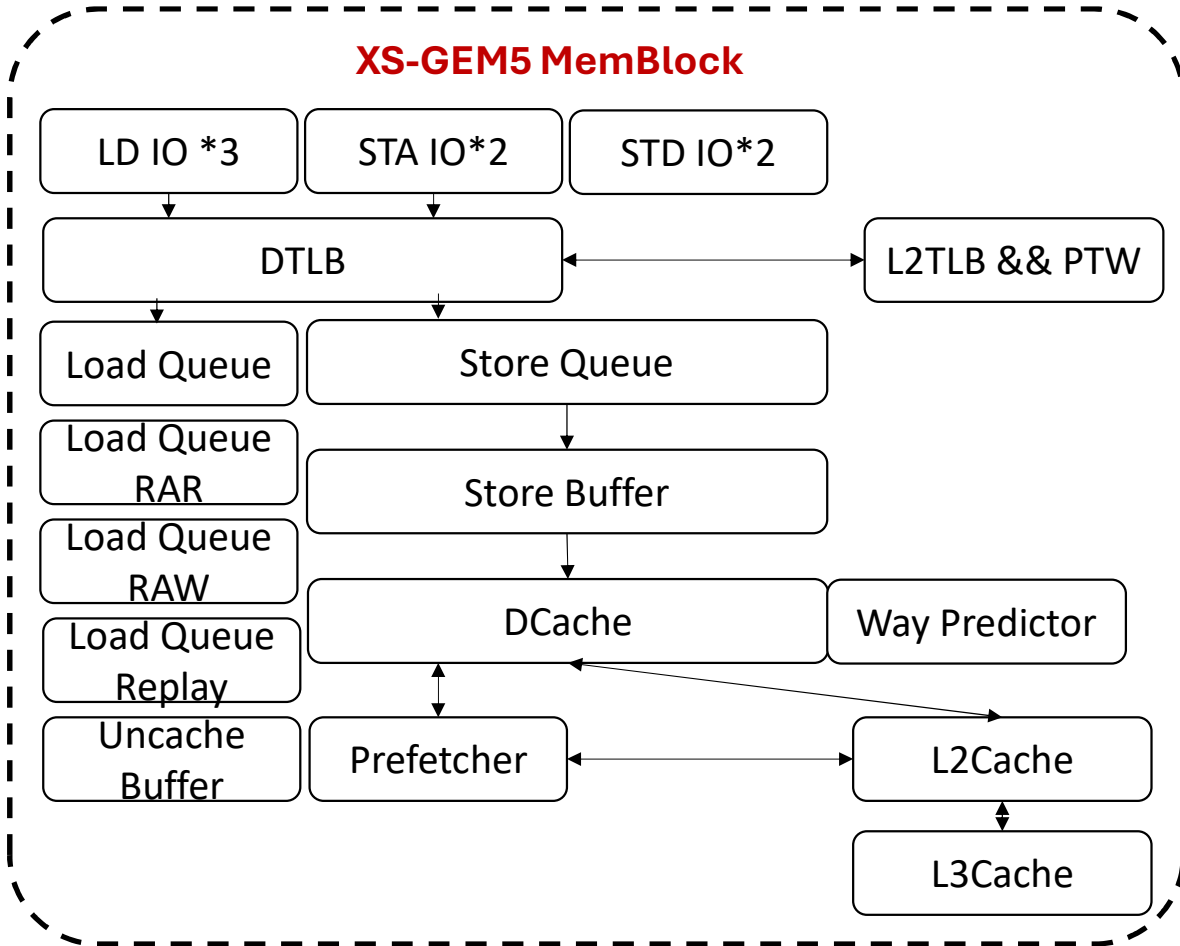


# Backend

- Instruction fusion
- Move elimination
- Efficient and configurable scheduler
  - Accurate calibration:
    - Dispatch
    - Issue
    - Replay
    - Wakeup network
  - Highly abstract and parameterized
    - Execute unit
    - Fast/spec wakeup channel
    - Regfile read/write port



# MemBlock



- Calibrated
  - Load & Store pipeline details<sup>[1]</sup>
  - Store STA/STD split<sup>[1]</sup>
  - Most Load replay scenarios modeling<sup>[1]</sup>
  - Prefetcher<sup>[4]</sup> & Way Predictor<sup>[5]</sup>
  - StoreBuffer<sup>[1]</sup>
  - MMU<sup>[2,3]</sup>
- Differences still
  - No detailed LoadQueue split
  - L3 Cache hardware logic details

Code details

[1] `src/cpu/o3/lsq*`

[2] `src/arch/riscv/tlb*`

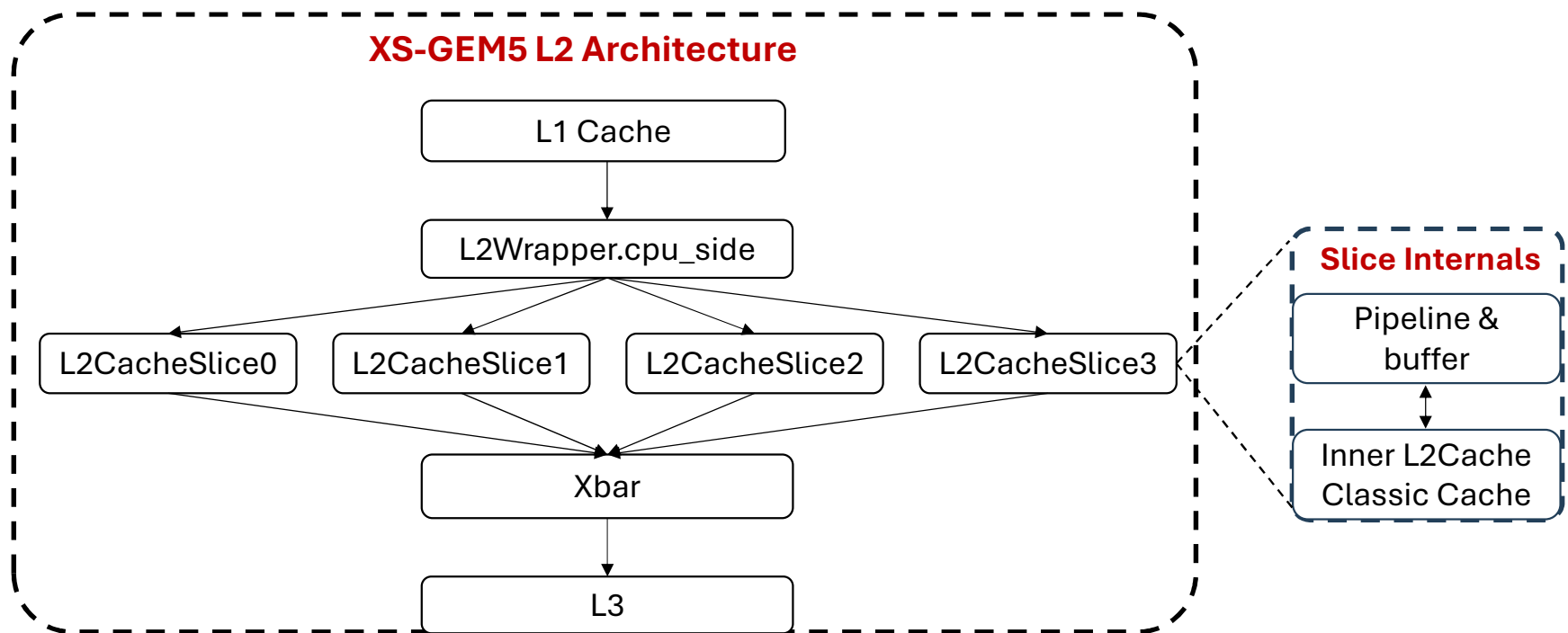
[3] `src/arch/riscv/pagetable_walker*`

[4] `src/mem/cache/prefetch/*`

[5] `src/mem/cache/way_prediction_policies/*`

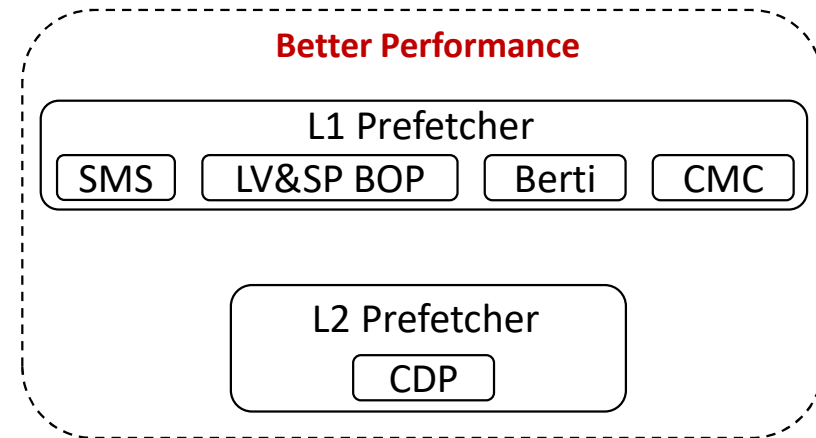
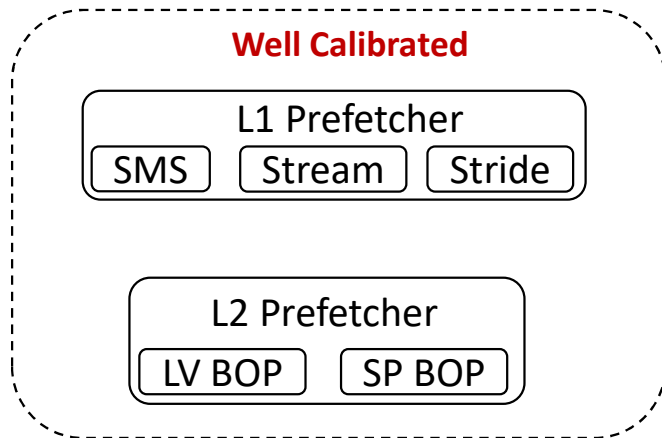
## L2 Cache

- We have implemented a L2 Cache calibrated with XiangShan CoupledL2
  - Based on GEM5 classic cache, a wrapper is added to simulate hardware behavior
    - Detailed pipeline arbitration/stall/schedule/control
    - Slice/prefetch mechanism calibrated with RTL



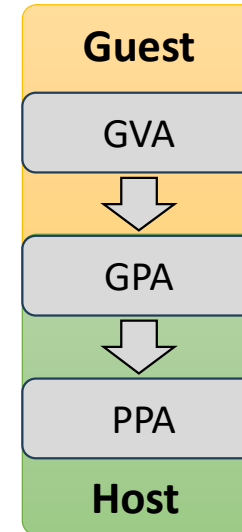
# Prefetcher

- Calibrated configuration with RTL (vias --kmh-align)
  - L1 Prefetchers: Stream + Stride
  - L2 Prefetchers: SMS + Large Virtual BOP + Small Physical BOP
- SOTA prefetch algorithms & mechanisms
  - IPCP, SPP, Berti, CDP, and temporal prefetcher
  - Prefetching request **offloading**: passive & active
  - Dynamic Prefetching control



# Support Hypervisor Extension

- **Functionality**
- **Architecture**
  - Supports SV39 and SV39x4
  - Supports SV48 and SV48x4
  - Supports two-stage address translation
  - Supports H-extension exception handling flow
  - Supports execution of H-extension CSR instructions
  - Supports interrupt controller Infra
- **Architectural calibration**
  - L1 TLB: 48-entry ITLB and DTLB
  - L2 TLB: L0, L1, and L2 store corresponding levels of the page table
- **Difference**
  - PTW (Page Table Walker): Idealized PTW model with no restriction on parallelism



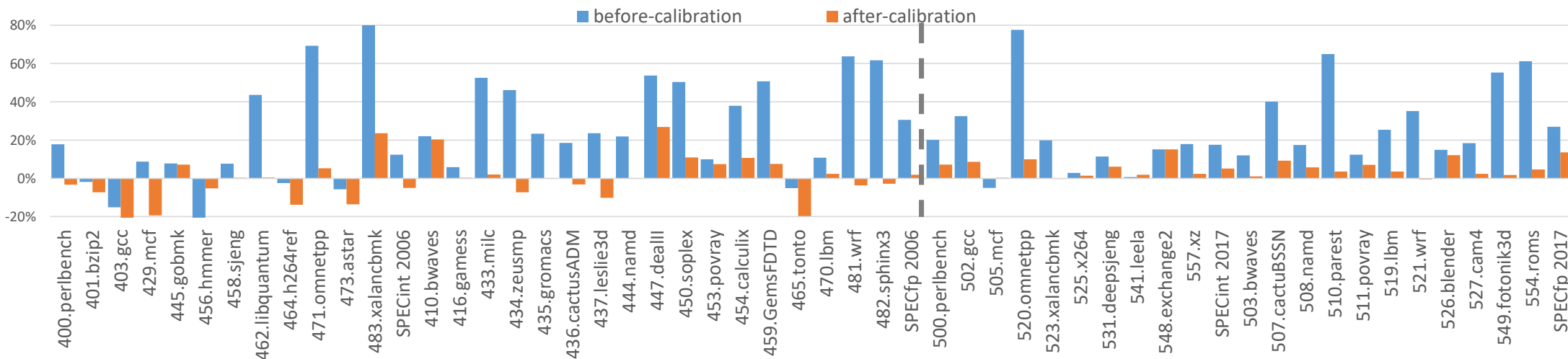
## Support for multiple DDR models

- Integration of **DRAMSim3**
  - Provides mainstream open-source DRAM modeling capability
  - Enables basic latency and bandwidth modeling for DDR3/DDR4
- Integration of **Ramulator2**
  - A new generation of flexible and extensible DRAM simulator
  - Supports multiple scheduling and row policies (e.g., FRFCFS, OpenRowPolicy)
  - Capable of modeling DDR3/DDR4/DDR5
- **Flexible configuration mechanism**
  - Switch between DRAMSim3 and Ramulator2 with a simple toggle
  - One-click configuration to switch between DDR4 and DDR5 models
  - Supports diverse simulation scenarios to meet different requirements



# Calibration

- Performance deviation between calibrated GEM5 and RTL on SPEC06:
  - SPEC06 Integer: 1.79% deviation between GEM5 and RTL
  - SPEC06 Floating Point: 2.69% deviation between GEM5 and RTL
  - SPEC17 Integer: 5.21% deviation between GEM5 and RTL
  - SPEC17 Floating Point: 13.61% deviation between GEM5 and RTL



**The gap in performance difference is gradually decreasing**

## Current SPEC CPU 2006 Score

- Estimated with RVGCpt + SimPoint
- Memory simulated with DRAMSim3:
  - DDR4 3200 Dual-channel
- Cache:
- 64kB Dcache+1 MB L2 +16MB L3
- Prefetcher:
  - **Stream + Stride + SMS + BOP**
  - With IPCP/SPP/Temporal **off**
- **SPEC int 2006 score: 15.40/GHz**
- **SPEC fp 2006 score: 15.04/GHz**
  - Compilation: **GCC 12 o3 base**

|             |             |            |             |
|-------------|-------------|------------|-------------|
| perlbench   | 12.619      | bwaves     | 22.405      |
| bzip2       | 8.89        | gamess     | 13.879      |
| gcc         | 15.855      | milc       | 16.25       |
| mcf         | 21.418      | zeusmp     | 15.056      |
| gobmk       | 10.594      | gromacs    | 11.663      |
| hmmer       | 14.178      | cactusADM  | 14.761      |
| sjeng       | 10.711      | leslie3d   | 14.864      |
| libquantum  | 40.5        | namd       | 9.508       |
| h264ref     | 19.019      | dealII     | 25.328      |
| omnetpp     | 14.516      | soplex     | 20.564      |
| astar       | 10.207      | povray     | 18.502      |
| xalancbmk   | 25.439      | calculix   | 5.575       |
| Int per GHz | 15.40004723 | GemsFDTD   | 16.887      |
|             |             | tonto      | 12.005      |
|             |             | lbm        | 28.414      |
|             |             | wrf        | 13.383      |
|             |             | sphinx3    | 13.497      |
|             |             | FP per GHz | 15.04328747 |



# Architecture exploration case: SMS algorithm optimization

- **Propose several SMS prefetch optimization algorithms**
  - Timeliness: PHT prefetch update algorithm guided by access timing
  - Coverage: PHT prefetch confidence calculation method guided by access frequency
- **Choose the features with better effect on the simulator and complete the RTL implementation**
- **The total SPEC06 score increased by 0.35% on the simulator and 0.4% on RTL**

|                 | baseline    | sms improve |        |
|-----------------|-------------|-------------|--------|
| SPECINT         |             |             |        |
| 400.perlbench:  | 13.043      | 13.085      | 0.32%  |
| 401.bzip2:      | 8.808       | 8.81        | 0.02%  |
| 403.gcc:        | 17.436      | 17.484      | 0.27%  |
| 429.mcf:        | 20.979      | 21.072      | 0.44%  |
| 445.gobmk:      | 9.92        | 9.921       | 0.01%  |
| 456.hmmer:      | 13.594      | 13.594      | 0.00%  |
| 458.sjeng:      | 10.364      | 10.334      | -0.29% |
| 462.libquantum: | 42.172      | 42.092      | -0.19% |
| 464.h264ref:    | 20.327      | 20.337      | 0.05%  |
| 471.omnetpp:    | 12.793      | 13.004      | 1.62%  |
| 473.astar:      | 19.124      | 19.066      | -0.30% |
| 483.xalancbmk:  | 24.653      | 24.672      | 0.08%  |
|                 | 16.101485   | 16.12901676 | 0.17%  |
|                 |             |             |        |
| SPECFP          |             |             |        |
| 410.bwaves:     | 25.078      | 25.723      | 2.51%  |
| 416.gamess:     | 14.693      | 14.698      | 0.03%  |
| 433.milc:       | 12.871      | 13.297      | 3.20%  |
| 434.zeusmp:     | 18.688      | 18.497      | -1.03% |
| 435.gromacs:    | 12.796      | 12.869      | 0.57%  |
| 436.cactusADM:  | 16.398      | 16.51       | 0.68%  |
| 437.leslie3d:   | 15.441      | 15.834      | 2.48%  |
| 444.namd:       | 11.554      | 11.569      | 0.13%  |
| 447.dealII:     | 19.355      | 19.399      | 0.23%  |
| 450.soplex:     | 16.908      | 16.848      | -0.36% |
| 453.povray:     | 18.049      | 18.093      | 0.24%  |
| 454.Calculix:   | 6.205       | 6.204       | -0.02% |
| 459.GemsFDTD:   | 11.336      | 11.36       | 0.21%  |
| 465.tonto:      | 11.689      | 11.679      | -0.09% |
| 470.lbm:        | 33.972      | 33.938      | -0.10% |
| 481.wrf:        | 11.712      | 11.799      | 0.74%  |
| 482.sphinx3:    | 18.279      | 18.294      | 0.08%  |
|                 | 15.15381473 | 15.24000599 | 0.57%  |



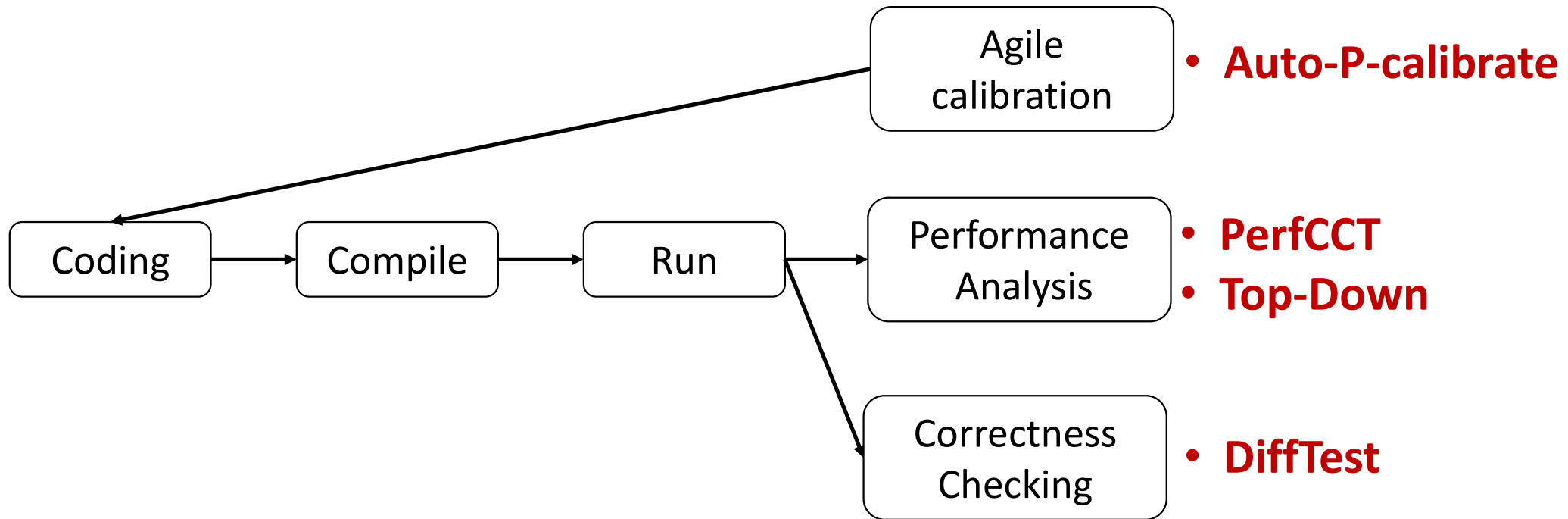
## Processor architecture exploration guided by simulator end to end

## What we will cover in this tutorial

- Introduction:
- Background: Why build such a simulator?
- Calibration & Architecture: How is the simulator designed?
- **Development Tools: What tools are we using at the current stage to improve development efficiency?**
- Case Study: A concrete example demonstrating how the simulator is used.

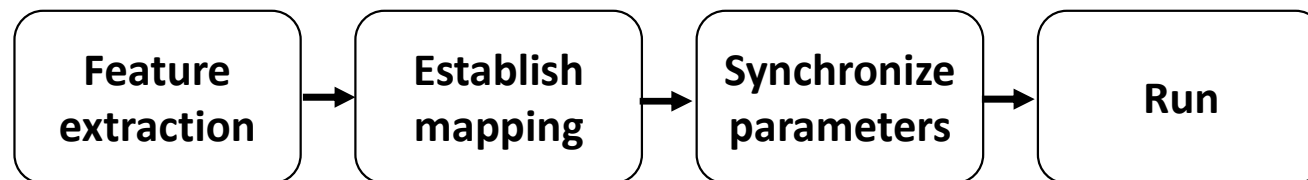
# 🏔 Introduction to Development Tools

- Development Workflow on an Architectural Simulator



## Automated parameter calibration

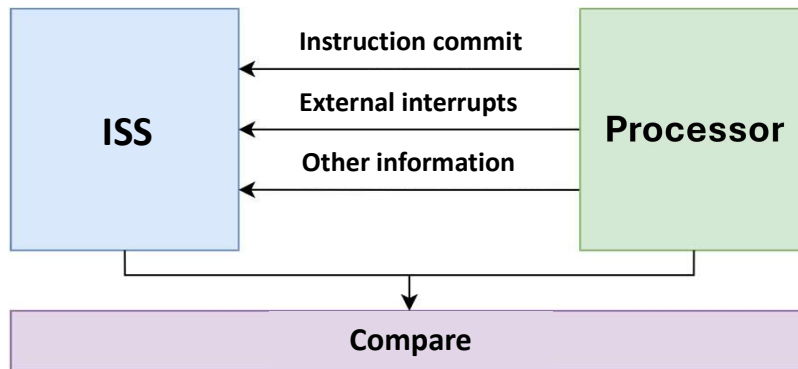
- RTL has many parameters, enabling dynamic calibration during development
- Automated scripts extract key architectural parameters from RTL
- Establish XS-RTL - XS-GEM5 parameter mapping
- Synchronize parameters to XS-GEM5 automatically
- Basic functionality implemented, supporting dynamic parameter calibration



# DiffTest: Efficient Debugging

## • Accelerating Architectural Simulator Debugging

- Implemented the DIFFTEST framework
- Online ISA-level behavior verification
- Uses NEMU/Spike as the reference model (REF)
- Compares register states between the architectural simulator and REF → triggers error or continues execution
- Quickly captures the context of instruction-level mismatches



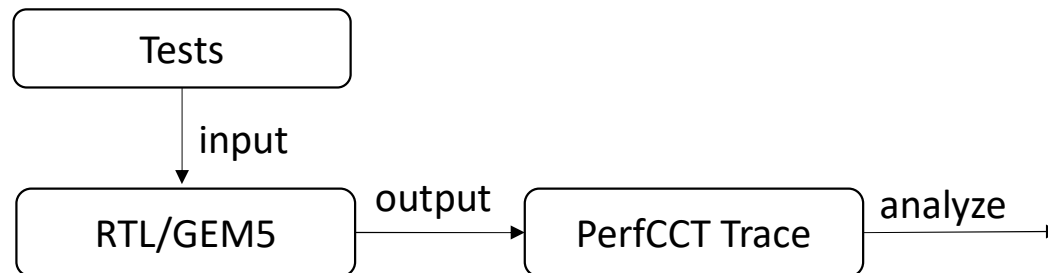
Basic architecture

```
while (1) {  
    icnt = gem5_step();  
    ref_step(icnt);  
    r1s = gem5_getregs();  
    r2s = ref_getregs();  
    if (r1s != r2s) { abort(); }  
}
```

Online Comparison Mechanism

# PerfCCT: Trace analysis tool

- Track the life cycle of an instruction
- Life cycle:
  - fetch/decode/rename/dispatch/issue/arbitration/execute/writeback/commit
- Extract Hot loops from Traces after running which output:
  - How many times each basic block appears
  - The proportion of each basic block in the total execution cycle
  - The Instructions of the basic block



The XiangShan Team

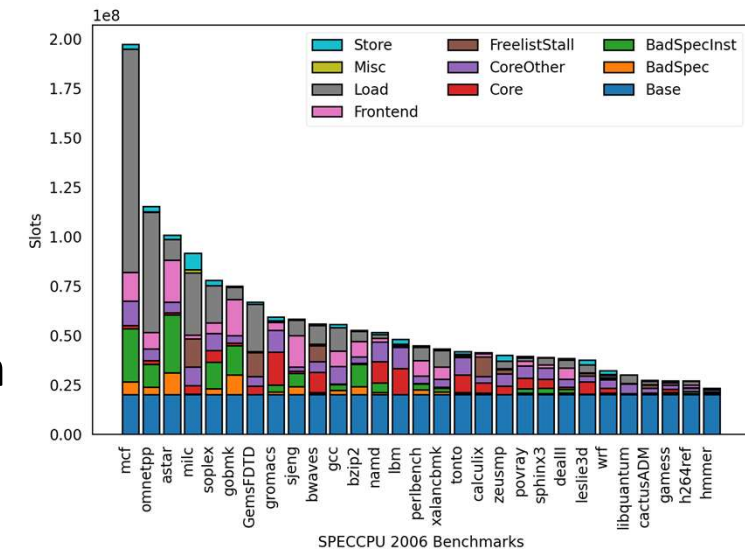
**It is convenient for us to understand the characteristics of the test**

```
Top 10 most common basic blocks:
Total cycle: 3040633
Count: 65265 (42.37%) cycle: 2182167 ratio (71.77%)
Instructions:
0xaae98: add s1, a7, t0
0xaae9c: fld fa4, -8(t4)
0xaaea0: fld ft10, -32(a7)
0xaaea4: fld fa1, -104(s1)
0xaaea8: fld fa6, -96(t1)
0xaaeac: fmul_d ft10, fa4, ft10
0xaaeb0: fmul_d fa1, fa4, fa1
0xaaeb4: fld ft4, -24(a1)
0xaaeb8: fmul_d fa6, fa4, fa6
0xaaebc: fld ft0, 48(a1)
0xaaec0: fmul_d ft4, fa4, ft4
0xaaec4: c_fld fa0, 120(a1)
0xaaec6: fmul_d ft10, ft10, fs5
0xaaeca: fmul_d fa1, fa1, fs11
0xaaece: fmul_d ft0, fa4, ft0
0xaaed2: fmul_d fa6, fa6, fs8
0xaaed6: fmul_d fa0, fa4, fa0
0xaaeda: fmul_d ft4, ft4, fs8
0xaaede: fld fs0, 40(a7)
0xaaee2: fmul_d ft10, ft10, fs11
0xaaee6: fadd_d fa1, fa1, fa5
0xaaeea: fmul_d ft0, ft0, fs8
0xaaeee: fmul_d fa6, fa6, fs11
0xaaef2: fmul_d fa0, fa0, fs8
0xaaef6: fmul_d ft4, ft4, fs5
0xaaefa: fmul_d fs10, fa4, fs0
0xaaefe: fld fa5, -88(t1)
0xaaf02: fadd_d fa1, fa1, ft10
0xaaf06: fmul_d ft0, ft0, fs3
```



# Top-Down: quickly identify performance bottlenecks

- Hierarchical Performance Counters
- Attribute instruction stalls to architectural components
- Top-down decomposition of processor performance events
- Quickly and accurately identify performance bottlenecks
- **Adapt Intel's Top-Down methodology to XiangShan design**
  - Optimize for RISC-V ISA
  - Enhance counters for XiangShan microarchitecture
  - Refine hierarchical Top-Down model design



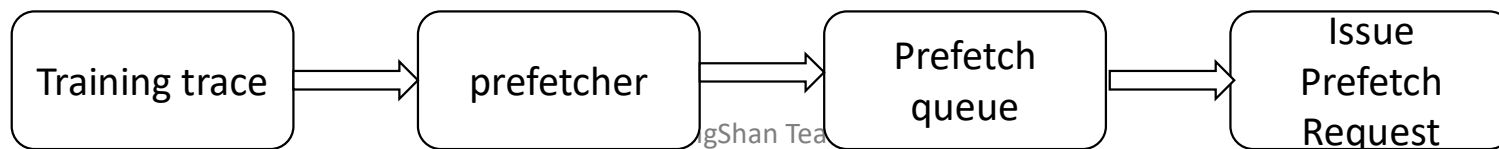
Performance Bottleneck Statistics  
Visualization Example  
Identify Performance Bottlenecks at  
the Macro Level

## What we will cover in this tutorial

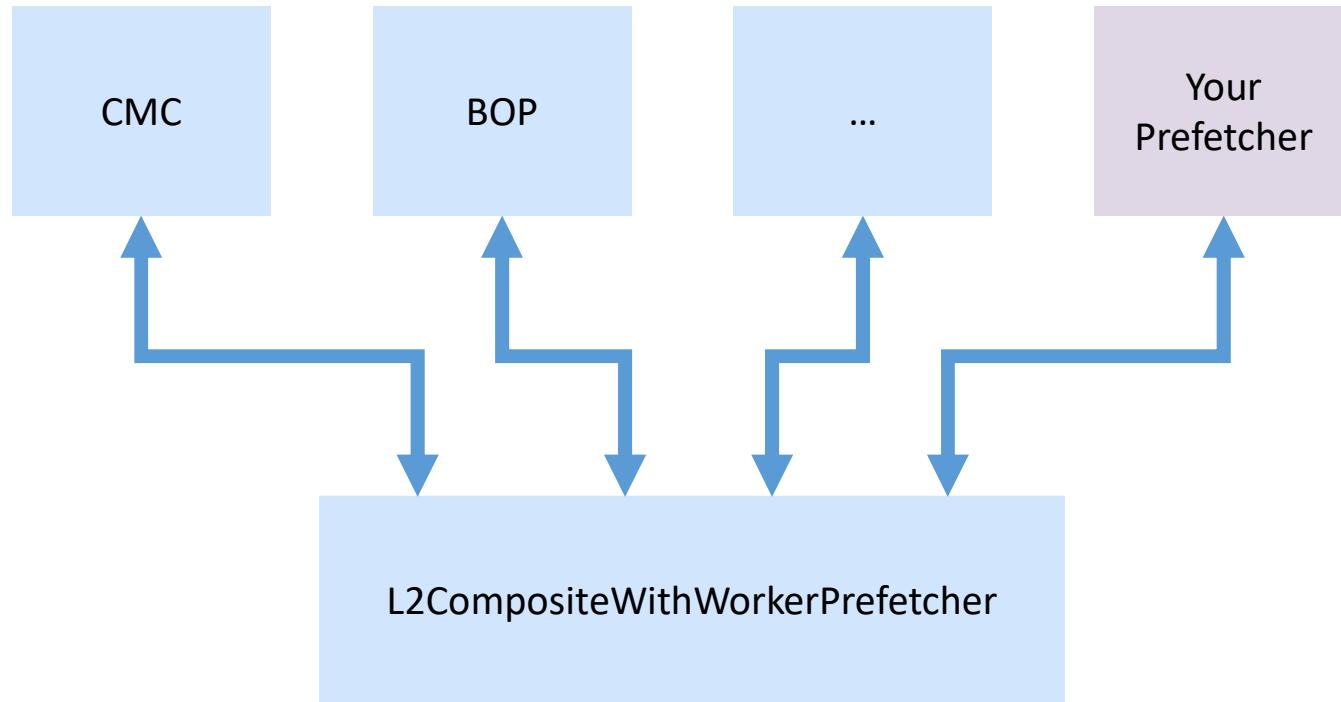
- Introduction:
- Background: Why build such a simulator?
- Calibration & Architecture: How is the simulator designed?
- Development Tools: What tools are we using at the current stage to improve development efficiency?
- **Case Study: A concrete example demonstrating how the simulator is used.**

## Case Study: How to add a new feature

- Example: Integrating a new prefetching algorithm for evaluation
- In XS-GEM5, prefetch execution is organized into several components. Adding a new prefetcher involves the following steps:
  - **Definition**
    - Define the new prefetcher in prefetcher.py.
  - **Configuration Switch**
    - Set whether the prefetcher is enabled by default in the configuration system.
  - **Declaration**
    - Declare the prefetcher within the simulator's prefetching framework.
  - **Implementation**
    - Implement the logic of the prefetching algorithm.
  - **Invocation**
    - Integrate the prefetcher into the execution flow by registering the call logic within the existing prefetching framework.



## Case Study: How to add a new feature



# Case Study: How to add a new feature

- 1. Define the prefetcher in prefetcher.py
  - **Prefetcher Specification:**
    - Specify key parameters such as prefetch distance, trigger conditions.
  - **Replacement Policy:**
    - Implement internal management strategies such as entry replacement or priority handling within the prefetcher.

```
diff --git a/src/mem/cache/prefetch/Prefetcher.py b/src/mem/cache/prefetch/Prefetcher.py
index d2cce7eb4d..a566c2a648 100644
--- a/src/mem/cache/prefetch/Prefetcher.py
+++ b/src/mem/cache/prefetch/Prefetcher.py
@@ -929,6 +929,57 @@ class CMCPrefetcher(QueuedPrefetcher):
     "Enable prefetch database"
 )

+class DespacitoStreamPrefetcher(QueuedPrefetcher):
+    type = "DespacitoStreamPrefetcher"
+    cxx_class = "gem5::prefetch::DespacitoStreamPrefetcher"
+    cxx_header = "mem/cache/prefetch/despacito_stream.hh"
+
+    use_virtual_addresses = False
+
 class XSCompositePrefetcher(QueuedPrefetcher):
     type = "XSCompositePrefetcher"
     cxx_class = 'gem5::prefetch::XSCompositePrefetcher'
@@ -1108,9 +1159,12 @@ class L2CompositeWithWorkerPrefetcher(CompositeWithWorkerPrefetcher):
     "Large BOP used in composite prefetcher ")
     bop_small = Param.BOPPrefetcher(SmallBOPPrefetcher(is_sub_prefetcher=True),
                                     "Small BOP used in composite prefetcher ")
+    despacito_stream = Param.DespacitoStreamPrefetcher(DespacitoStreamPrefetcher(is_sub_prefetcher=True),
+                                                       "DespacitoStream used in composite prefetcher")
     enable_bop = Param.Bool(False, "Enable BOP")
     enable_cdp = Param.Bool(True, "Enable CDP")
     enable_cpc = Param.Bool(False, "Enable CPC")
+    enable_despacito_stream = Param.Bool(True, "Enable despacito stream")

 class L3CompositeWithWorkerPrefetcher(CompositeWithWorkerPrefetcher):
     type = 'L3CompositeWithWorkerPrefetcher'
```

## Case Study: How to add a new feature

- 2. Enable or disable the prefetcher in prefetcherConfig.py

```
diff --git a/configs/common/PrefetcherConfig.py b/configs/common/PrefetcherConfig.py
index 9467763251..20d873d3da 100644
--- a/configs/common/PrefetcherConfig.py
+++ b/configs/common/PrefetcherConfig.py
@@ -65,6 +65,7 @@ def create_prefetcher(cpu, cache_level, options):
     prefetcher.enable_cmc = False
     prefetcher.enable_bop = True
     prefetcher.enable_cdn = False
+    prefetcher.enable_despacito_stream = False
     prefetcher.bop_large = XSVirtualLargeBOP(is_sub_prefetcher=True)
     prefetcher.bop_small = XSPhysicalSmallBOP(is_sub_prefetcher=True)
     if options.l1_to_l2_pf_hint:
```

## Case Study: How to add a new feature

- 3. Declare the prefetcher in SConscript


```
diff --git a/src/mem/cache/prefetch/SConscript b/src/mem/cache/prefetch/SConscript
index c77c5494ad..61756a1bf6 100044
--- a/src/mem/cache/prefetch/SConscript
+++ b/src/mem/cache/prefetch/SConscript
@@ -37,7 +37,7 @@ SimObject('Prefetcher.py', sim_objects=[
    'SignaturePathPrefetcherV2', 'AccessMapPatternMatching', 'AMPMPrefetcher',
    'DeltaCorrelatingPredictionTables', 'DCPTPrefetcher',
    'IrregularStreamBufferPrefetcher', 'SlimAMPMPrefetcher',
-   'WorkerPrefetcher',
+   'WorkerPrefetcher', 'DespacitoStreamPrefetcher',
    'BUFPrefetcher', 'SBOUEPrefetcher', 'SiEMSPrefetcher', 'PIFPrefetcher', 'IPCPrefetcher',
    'CompositeWithWorkerPrefetcher', 'L2CompositeWithWorkerPrefetcher'])

@@ -58,6 +58,7 @@ DebugFlag('CDPHotVpns')
    DebugFlag('CDPdebug')
    DebugFlag('CDPdepth')
    DebugFlag('CMCPrefetcher')
+   DebugFlag('DespacitoStreamPrefetcher')

    Source('access_map_pattern_matching.cc')
    Source('base.cc')
@@ -86,4 +87,5 @@ Source('worker.cc')
    Source('cmc.cc')
    Source('composite_with_worker.cc')
    Source('l2_composite_with_worker.cc')
+   Source('despacito_stream.cc')
```

## Case Study: How to add a new feature

- 4. Implement the prefetching
  - Write the core logic of the prefetcher in the source files:
  - `src/mem/cache/prefetch/prefetch1.cc`
  - `src/mem/cache/prefetch/prefetch1.hh`

```
> src/mem/cache/prefetch/despacito_stream.cc 
```

```
> src/mem/cache/prefetch/despacito_stream.hh 
```



## Case Study: How to add a new feature

- 5. Invoke the prefetcher in the L1/L2 prefetching framework

```
diff --git a/src/mem/cache/prefetch/L2_composite_with_worker.cc b/src/mem/cache/prefetch/L2_composite_with_worker.cc
index 31ba47bf01..b7ede79a17 100644
--- a/src/mem/cache/prefetch/L2_composite_with_worker.cc
+++ b/src/mem/cache/prefetch/L2_composite_with_worker.cc
@@ -16,14 +16,17 @@ L2CompositeWithWorkerPrefetcher::L2CompositeWithWorkerPrefetcher(const L2Composi
    largeBOP(p.bop_large),
    smallBOP(p.bop_small),
    cmc(p.cmc).
+   despacitoStream(p.despacito_stream),
    enableBOP(p.enable_bop),
    enableCDP(p.enable_cdp),
-   enableCMC(p.enable_cmc)
+   enableCMC(p.enable_cmc),
+   enableDespacitoStream(p.enable_despacito_stream)
    {
        cdp->pflRUFilter = &pflRUFilter;
        largeBOP->filter = &pflRUFilter;
        smallBOP->filter = &pflRUFilter;
        cmc->filter = &pflRUFilter;
+       despacitoStream->filter = &pflRUFilter;
        cdp->parentKio = p.sys->getrequestorio(this);
    }

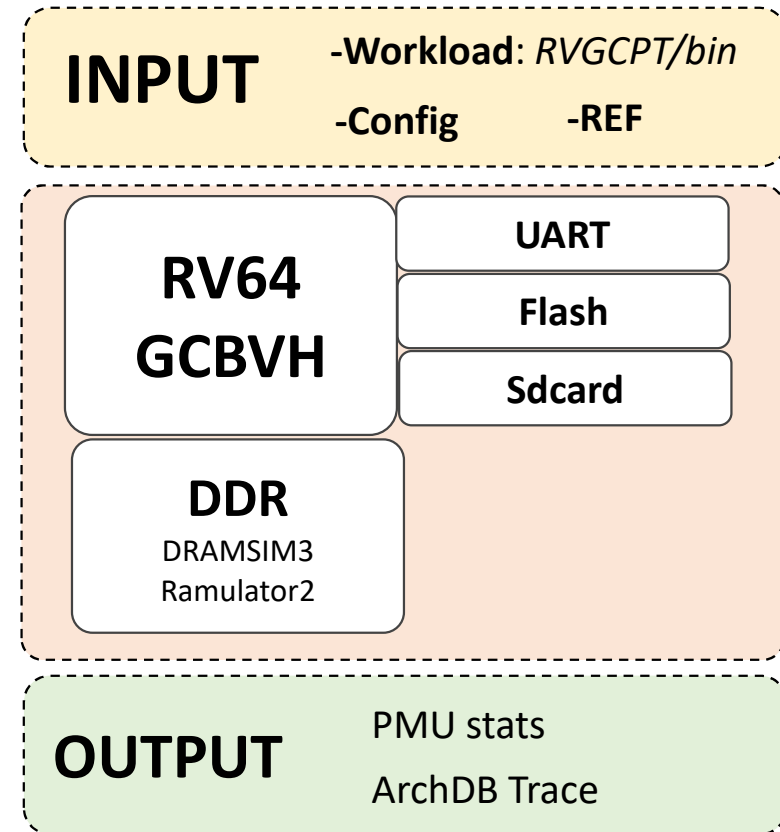
@@ -65,6 +68,9 @@ L2CompositeWithWorkerPrefetcher::calculatePrefetch(const PrefetchInfo &pfi, std:
    largeBOP->calculatePrefetch(pfi, addresses, late && pf_source == PrefetchSourceType::HWP_BOP);
    smallBOP->calculatePrefetch(pfi, addresses, late && pf_source == PrefetchSourceType::HWP_BOP);
+   if (enableDespacitoStream) {
+       despacitoStream->calculatePrefetch(pfi, addresses, late && pf_source == PrefetchSourceType::DespacitoStream);
+   }
}

void
@@ -121,6 +127,7 @@ L2CompositeWithWorkerPrefetcher::setParentInfo(System *sys, ProbeManager *pm, Ca
    largeBOP->setParentInfo(sys, pm, _cache, blk_size);
    smallBOP->setParentInfo(sys, pm, _cache, blk_size);
    cmc->setParentInfo(sys, pm, _cache, blk_size);
+   despacitoStream->setParentInfo(sys, pm, _cache, blk_size);
    CompositeWithWorkerPrefetcher::setParentInfo(sys, pm, _cache, blk_size);
}
```

# Summary

- An effective and calibrated simulator for research on microarchitecture design.
- Agile development tools for performance analysis and correctness checking.

 <https://github.com/OpenXiangShan/GEM5>



# Thanks!