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

香山开源处理器社区
XiangShan Open-Source Processor Community

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

The XiangShan Team

HPCA'25@Las Vegas, USA

March 02, 2025

What we will cover in this tutorial

- Introduce of XiangShan (8:30 – 9:00, 30 minutes)

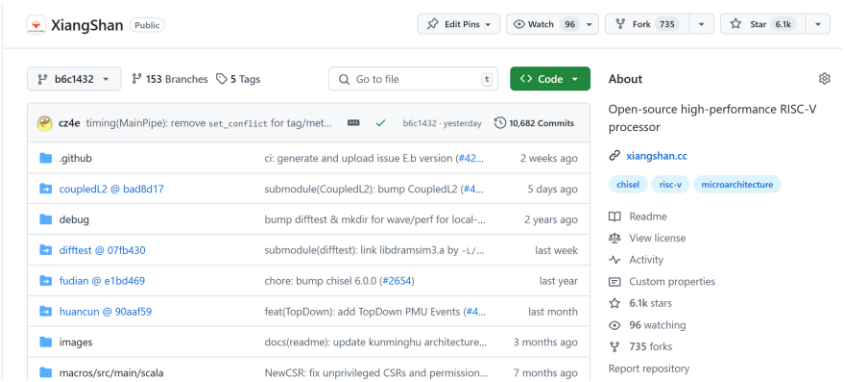
- **CPU Microarchitecture (9:00 – 10:00, 60 minutes)**
  - *Design and implementation – How to implement novel ideas on XiangShan*
  - Frontend: branch prediction and instruction fetch
  - Backend: out-of-order scheduler, execution units
  - Load/Store Unit: LSQ, pipelines, TLBs, data caches
  - L2/L3 caches and prefetchers

- Development workflows (10:30 – 12:00, 90 minutes)
  - *Introduction of their usages – How to develop on XiangShan with MinJie*
  - Simulation and Debugging
  - Research Demo

XiangShan: Open-Source High-Performance Processor

- **1st generation: YQH (Yanqihu)**
  - 2020/6: first commit of design RTL
  - 2021/7: **28nm tape-out, 1.3GHz**
  - Performance: **SPEC CPU2006 7.01@1GHz, DDR4-1600**
- **2nd generation: NH (Nanhu)**
  - 2021/5: starting design exploration and RTL design
  - 2023/4: GDSII delivery
  - **14-nm tape-out, estimated SPEC CPU2006 20@2GHz**
- **3rd generation: KMH (Kunminghu)**
  - ISA feature: **Vector (V) extension, Hypervisor (H) extension**
  - Support RVA23 profile and L2 Cache with CHI protocol
  - **estimated SPEC CPU2006 45@3GHz**
- **Open-sourced at https://github.com/OpenXiangShan/XiangShan**

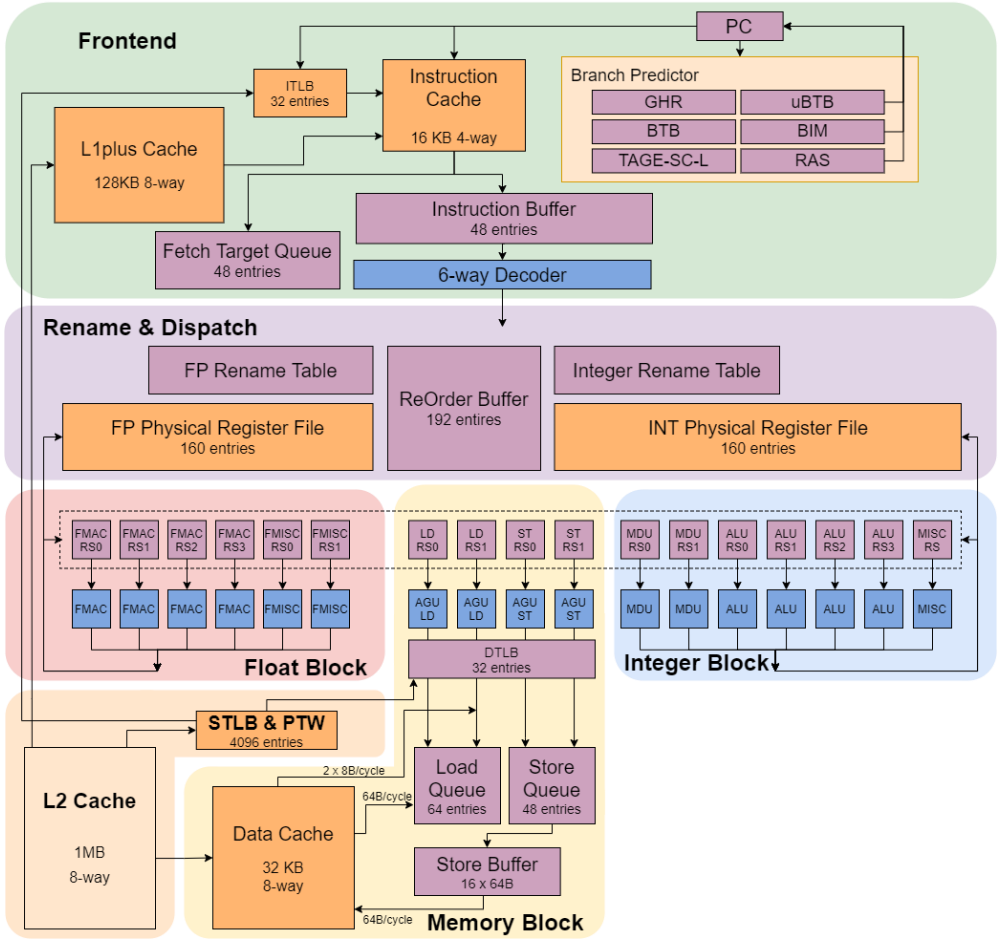
Fragrant Hills in Beijing



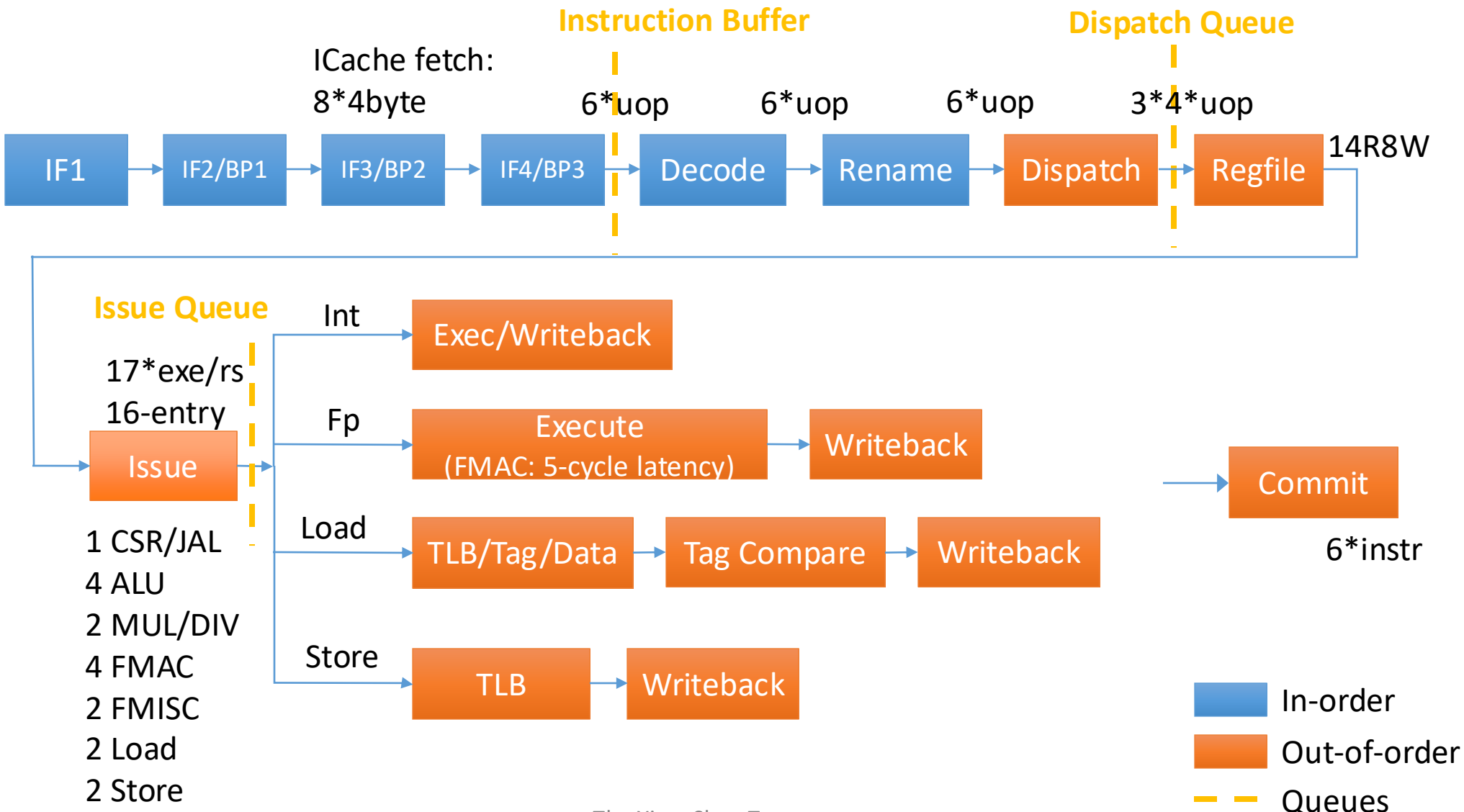
>6.1K stars, >735 forks on GitHub

Yanqihu : 1st generation of XiangShan

- **11**-stage, **6**-wide decode/rename
- **TAGE-SC-L** branch prediction
- **160** Int PRF + **160** FP PRF
- **192**-entry ROB, **64**-entry LQ, **48**-entry SQ
- **16**-entry RS for each FU
- **16KB** L1 Cache, **128KB** L1plus Cache for instruction
- **32KB** L1 Data Cache
- **32**-entry ITLB/DTLB, **4K**-entry STLB
- **1MB** inclusive L2 Cache



Yanqihu µArch Overview

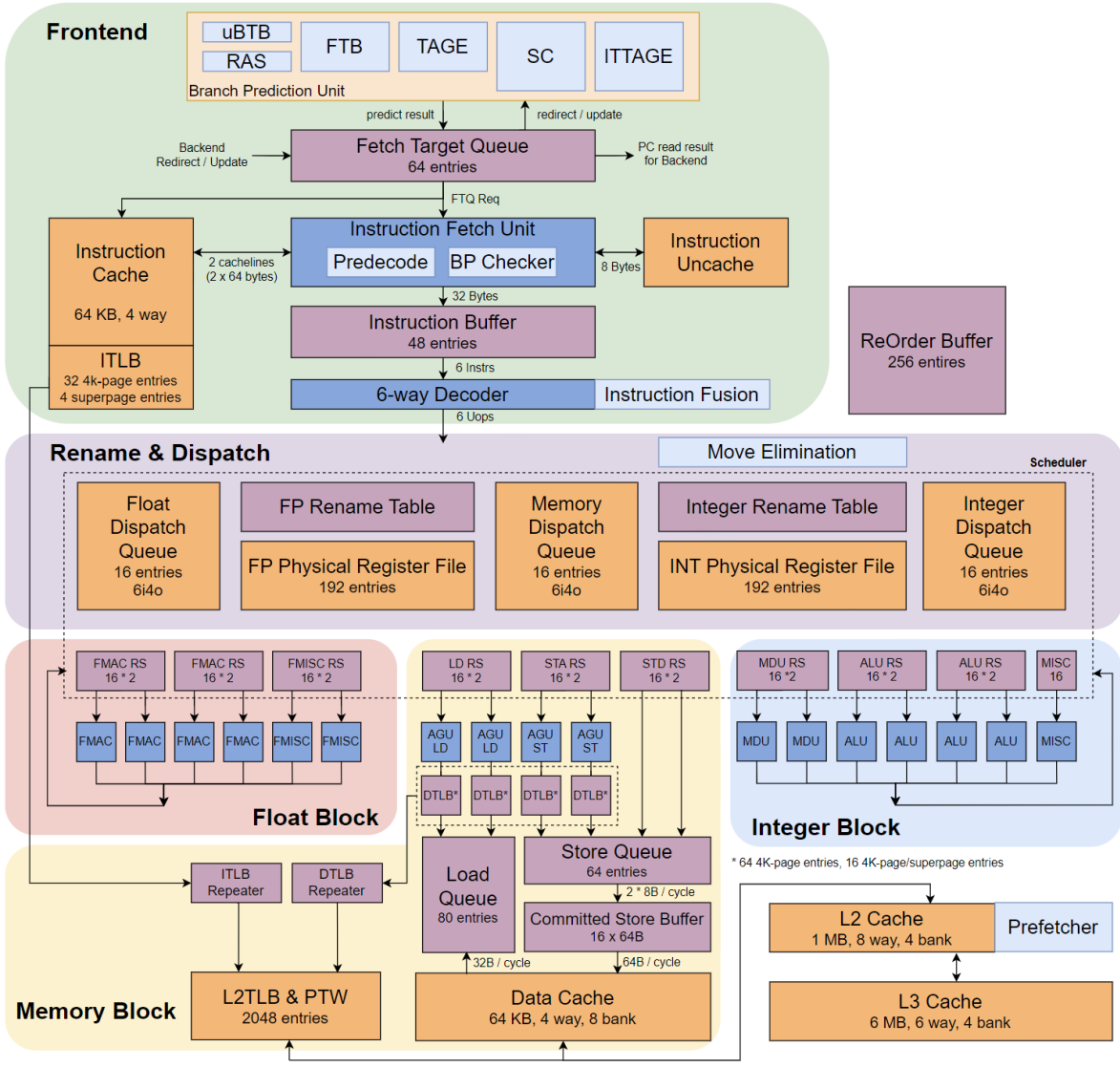


Overview: XiangShan Gen 2 Nanhu

- Named after a lake in Jiaxing, Zhejiang, China
- **Fetch & Branch Prediction**
  - Decoupled fetch and branch prediction unit in Frontend architecture
  - Higher throughput and prediction accuracy
- **New feature and Functional Support**
  - Support RISC-V Bit-Manipulation (B) and Cryptography (K) extensions
  - Support Physical Memory Protection (PMP) and configurable PMA
- **Load Store Unit**
  - Increase the capacity of TLB, Dcache and LSQ entries
  - Redesign DCache pipeline to reduce bank conflicts and improve efficiency
- **L2/L3 Cache**
  - Develop an open-source high performance L2 Cache / LLC named **HuanCun**

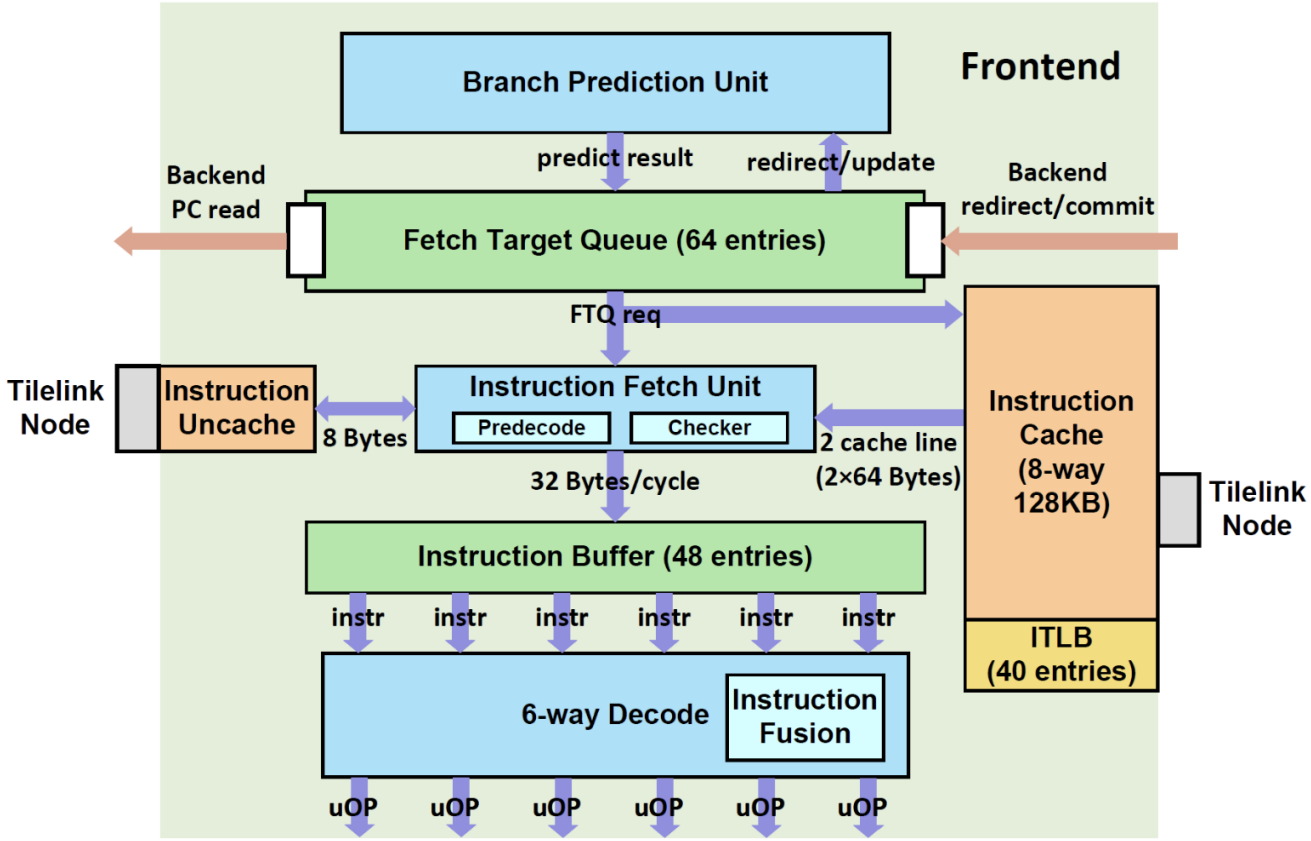
Nanhu μArch Overview

- **192** Int PRF + **192** FP PRF
- **256**-entry ROB, **80**-entry LQ, **64**-entry SQ
- **32**-entry RS for two FUs
- **64KB** L1 Instr. Cache, **64KB** L1 Data Cache
- **64**-entry DM + **16**-entry FA DTLB
- **32**-entry ITLB, **2K**-entry STLB
- **1MB** non-inclusive L2 Cache
- **6MB** non-inclusive L3 Cache (LLC)



Frontend Overview

- Decoupling of Fetch and Branch Prediction Units
  - **Producer**: branch predictor unit
  - **Consumer**: instruction fetch unit
- Merit
  - Performance
    - Hide branch prediction bubbles
    - Guide instruction prefetching
  - Timing
    - Avoid interact of BPU and ICache
- Other Detailed Optimization



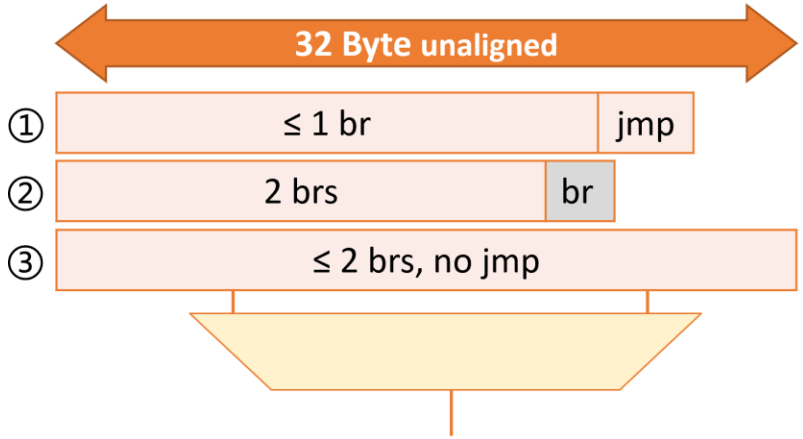
* Frontend design of YQH is partially learnt from BOOM

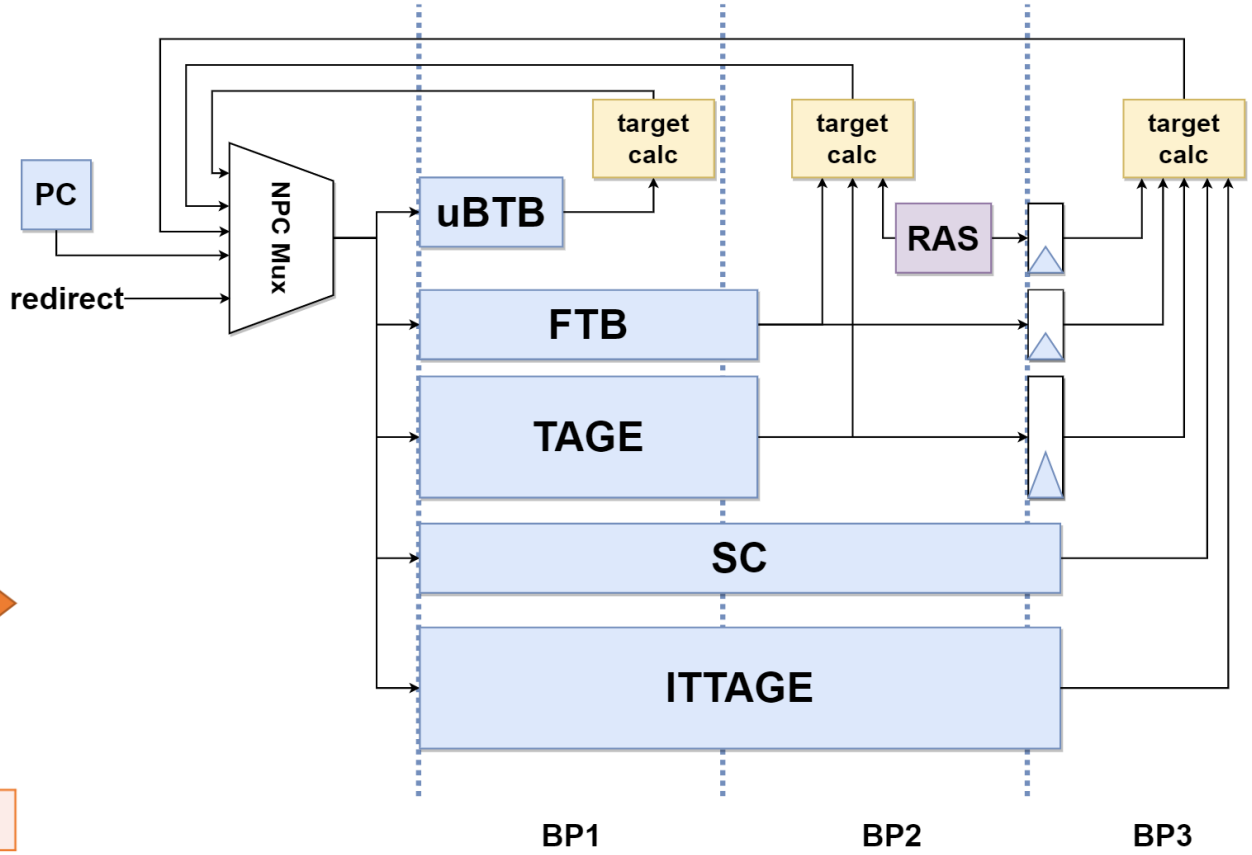
Frontend — Branch Predictor

Three-stage overriding prediction

- **Stage 1**： uBTB
- **Stage 2**： FTB + TAGE + RAS
- **Stage 3**： ITTAGE + SC

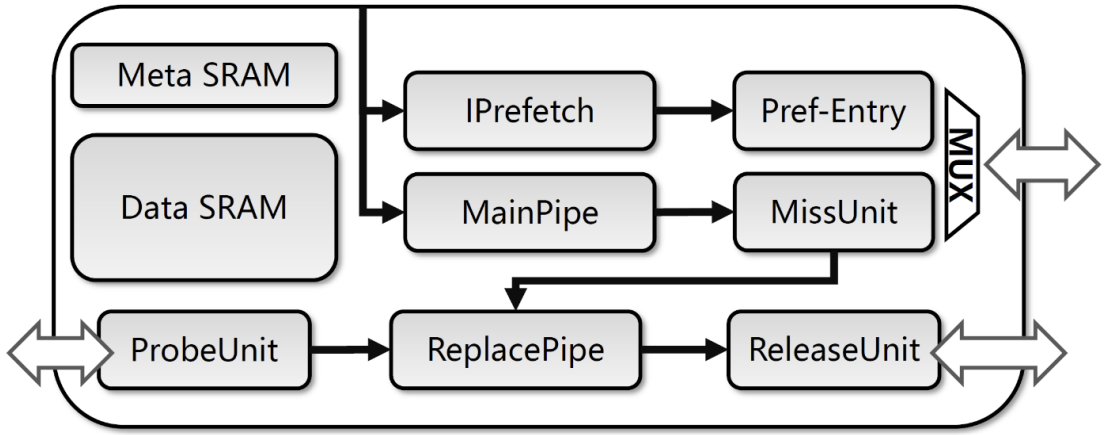
- FTB: Fetch Target Buffer





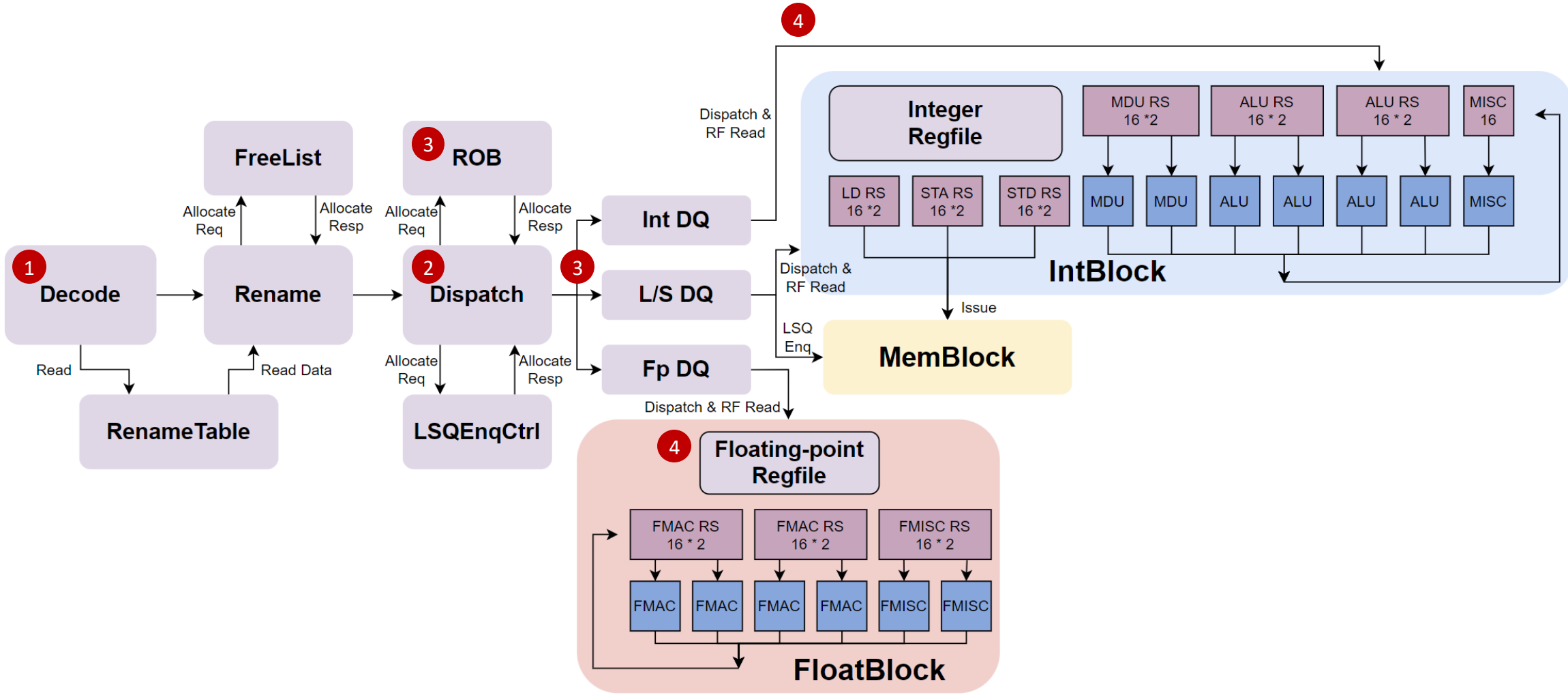
Frontend — Instruction Cache

- **Main feature**
  - 64KB 4-w VIPT blocking cache
  - Even/odd bank interleaving read
  - **Support TileLink protocol**
  - FDIP instruction prefetching



	Yanqihu	Nanhu	
L1 ICache	16KB, 4-w	64KB, 4-w	4×
L1plus Cache	128KB, 8-w	-	Reduce
Read bandwidth	64 B	128 B	2×
TileLink Support	No	Yes	New
Instruction prefetch	Stream	L2 FDIP	New

Backend Overview



Backend — Instruction Fusion

- Fusion of adjacent UOPs
  - Improve backend efficiency
  - Reduce data bypass delay
- Fusion Types
  1. Reuse of the original calculation
  2. Add new type of calculation
- **Reduce dynamic instruction count**

```
slli   r1, r0, 32
srli   r1, r1, 32
```

⇒ `add.uw r1, r0, zero`

Type1: Reusing the original calculation

```
srli    r1, r0, 8
andi r1, r1, 255
```

⇒ `byte2 r1, r0`

Type2: Creating new type of calculation

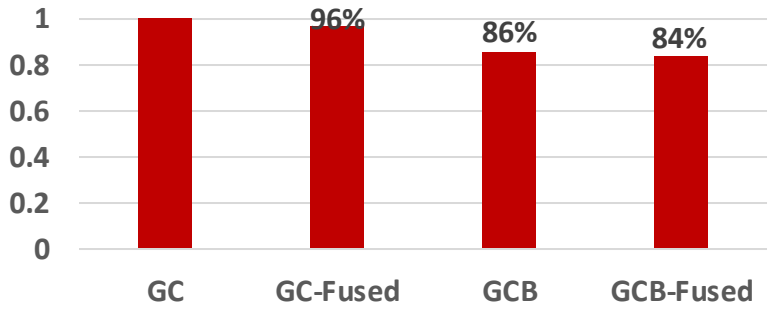
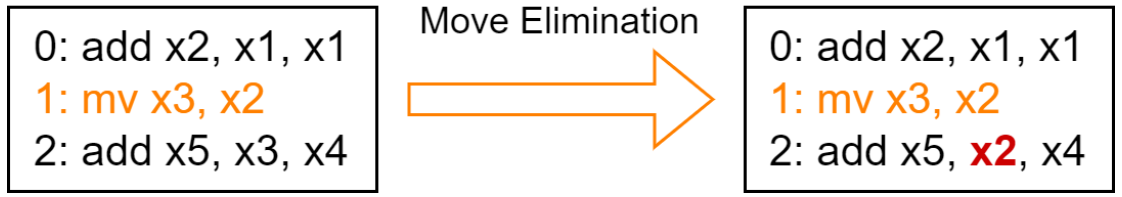


Figure. Normalized Dynamic Instruction Count for CoreMark (-O2)

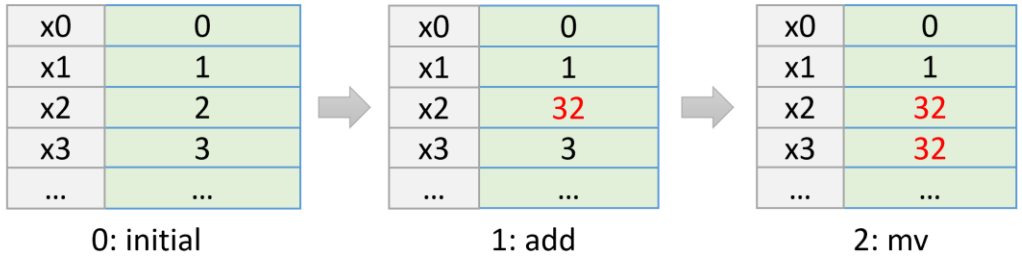
Backend — Rename & Move Elimination

- RenameMap: the mapping between **arch registers** and **physical registers**
- Move Elimination
  - No need to allocate new physical register by just changing the RenameMap
  - Mark as **Complete** when Move instruction enters ROB
  - Use **Reference-Counter** to record mapping counts of physical registers

program execution flow:



rename table mapping:

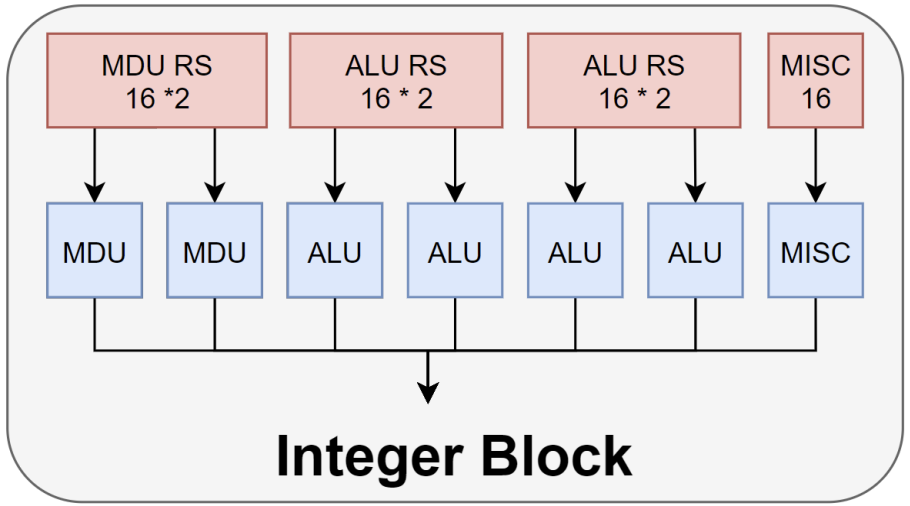


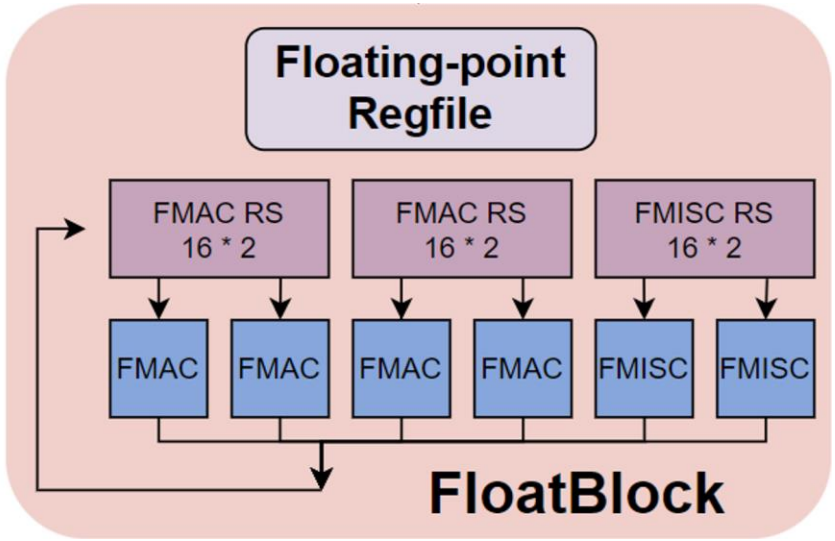
Backend — Reservation Station

- Main Changes:
  - 1i1o -> 2i2o
  - based selection

0	true			
1	true	true		
2	true	false	true	
3	true	false	false	true
	0	1	2	3

Age Matrix

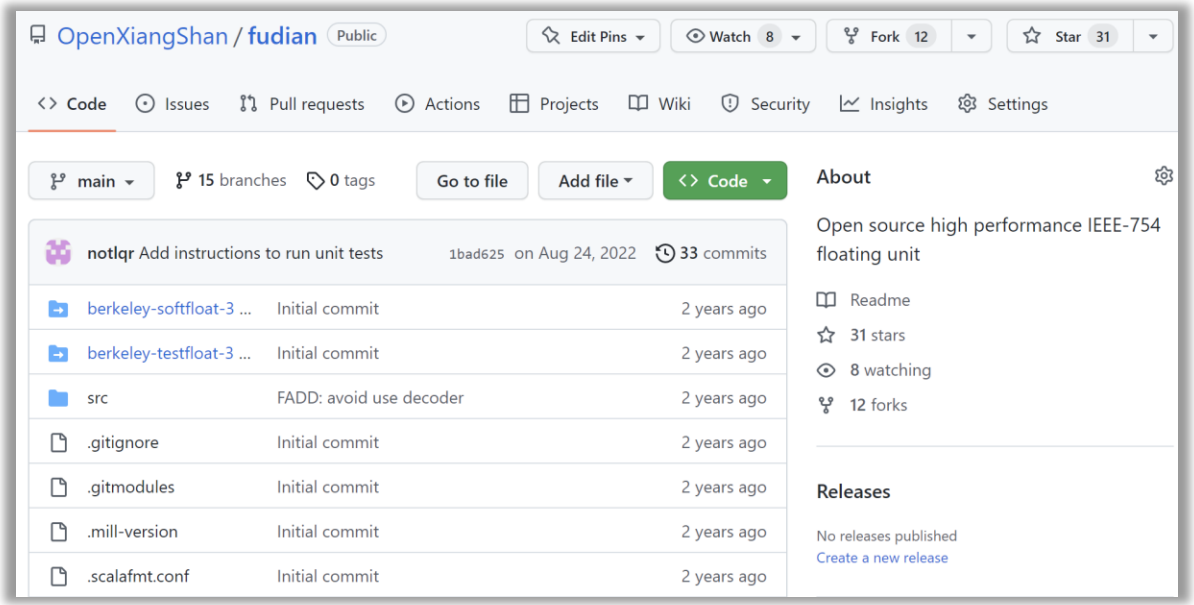




Backend — Floating Point Units (FPUs)

- Industry competitive Floating Point Unit, **IEEE 754** compatible

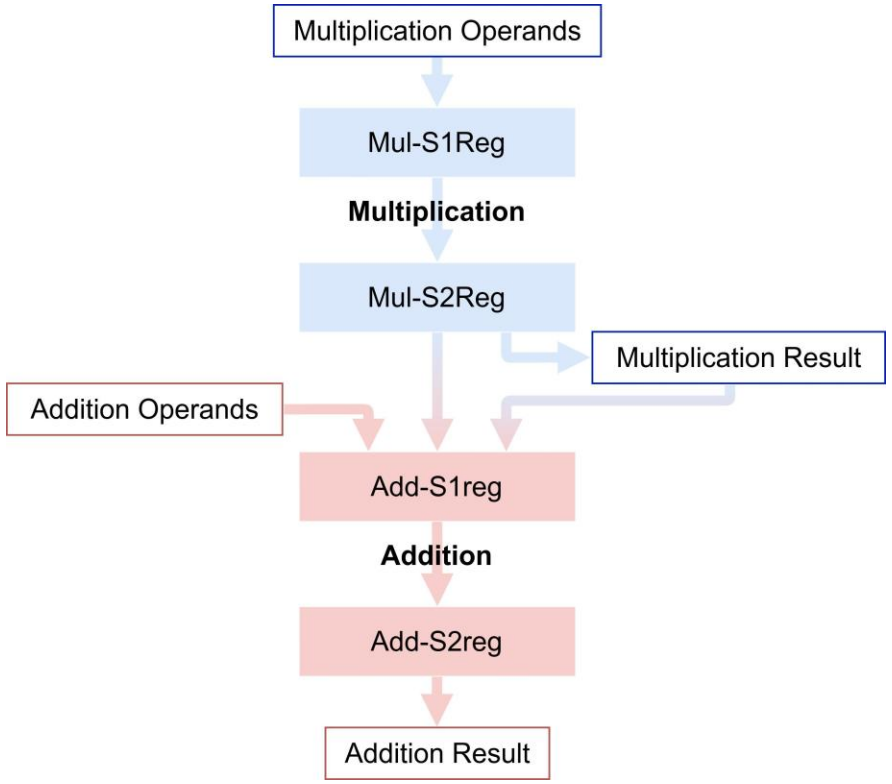
Operations	Latency
FADD	3
FMUL	3
FMA	5
FDIV	≤ 11 (float) ≤ 18 (double)
FSQRT	≤ 17 (float) ≤ 31 (double)
FCVT, FCMP	3

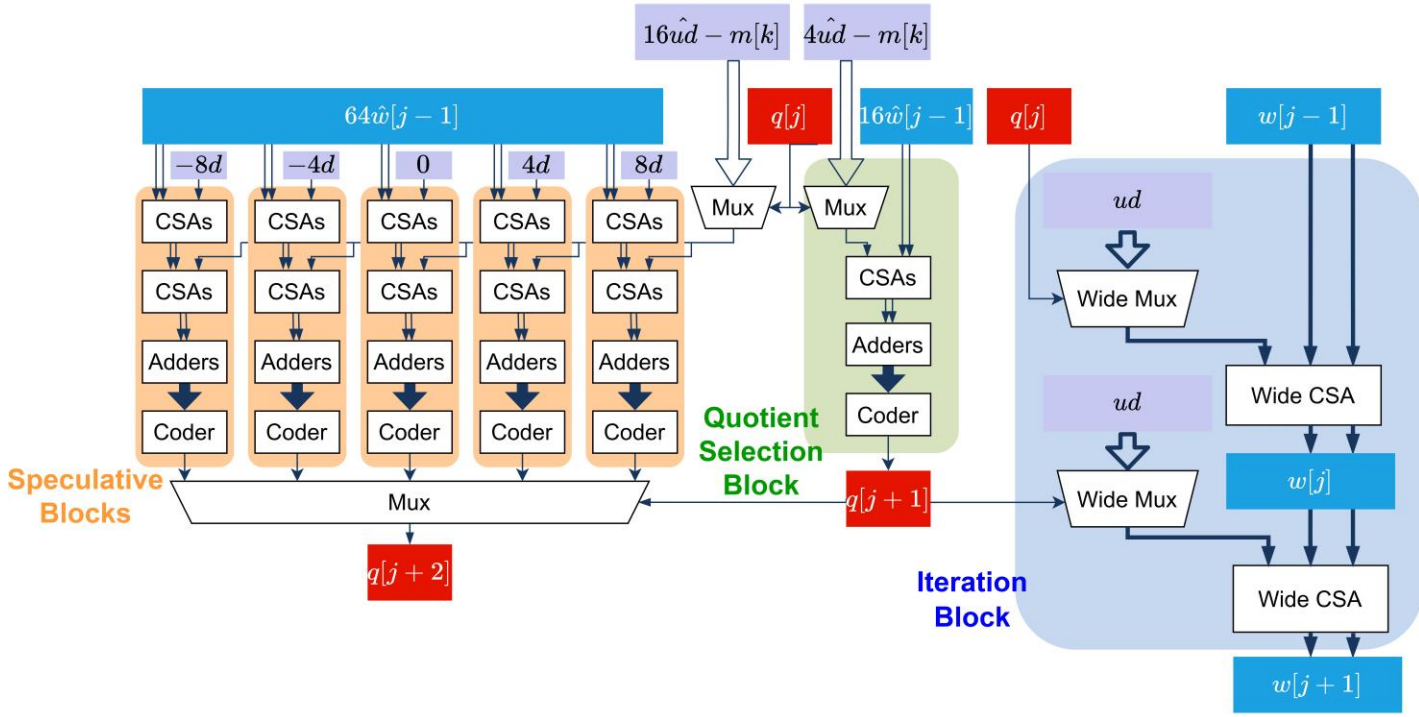


https://github.com/OpenXiangShan/fudian

Backend — FPU

- **Cascade FMA**
  - Reduce FADD delay 5->3
- **SRT16 Division units**
  - 1/2 delay compared with YQH (SRT4)





Backend — B/K Extension Implemented

- **B**: Bit-Manipulation extension
- **K**: Cryptographic extension

1. Extensions
   1.1. Zba extension
   1.2. Zbb: Basic bit-manipulation
      1.2.1. Logical with negate
      1.2.2. Count leading/trailing zero bits
      1.2.3. Count population
      1.2.4. Integer minimum/maximum
      1.2.5. Sign- and zero-extension
      1.2.6. Bitwise rotation
      1.2.7. OR Combine
      1.2.8. Byte-reverse
   1.3. Zbc: Carry-less multiplication
   1.4. Zbs: Single-bit instructions

B Extension

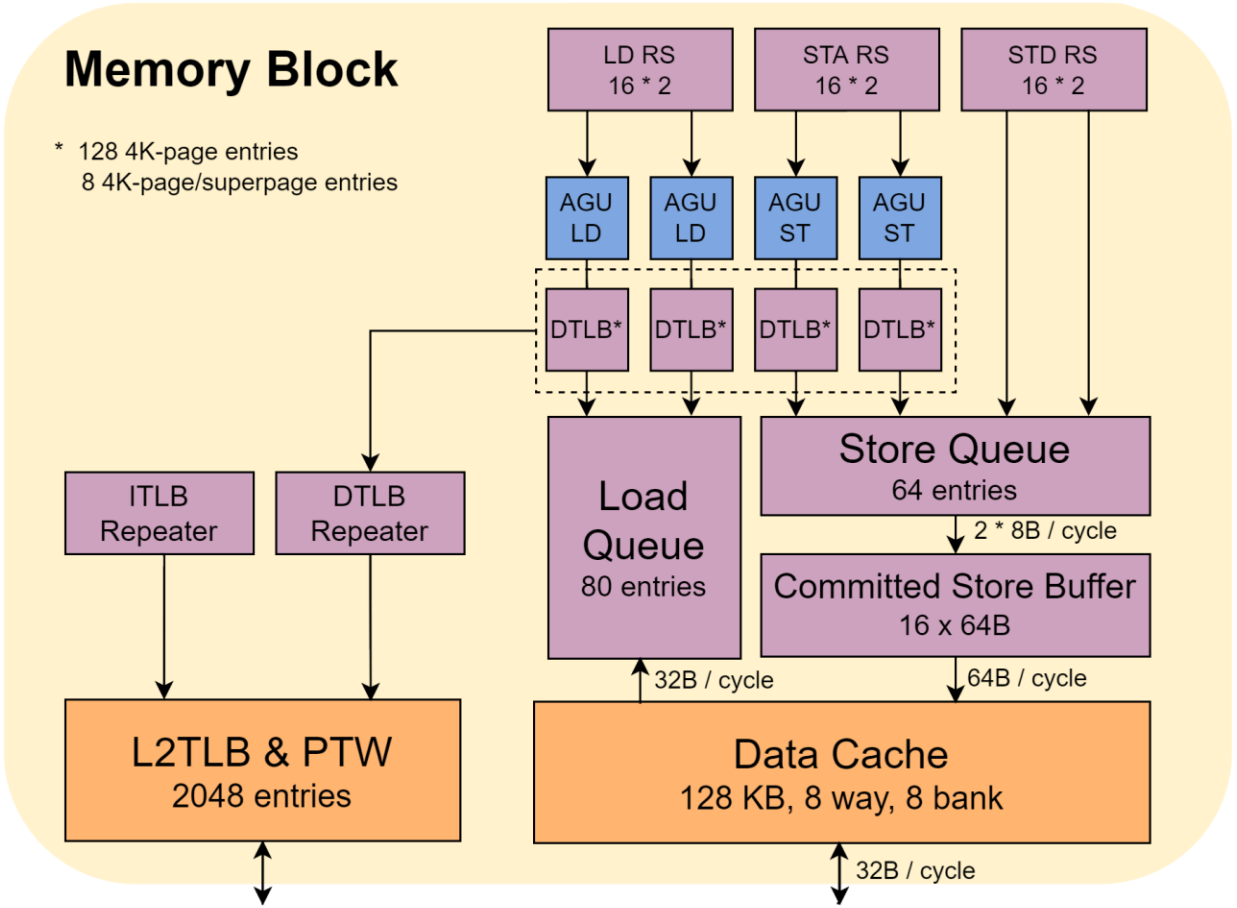
2. Extensions Overview
   2.1. Zbkb - Bitmanip instructions for Cryptography
   2.2. Zbkc - Carry-less multiply instructions
   2.3. Zbkx - Crossbar permutation instructions
   2.4. Zknd - NIST Suite: AES Decryption
   2.5. Zkne - NIST Suite: AES Encryption
   2.6. Zknh - NIST Suite: Hash Function Instructions
   2.7. Zksed - ShangMi Suite: SM4 Block Cipher Instructions
   2.8. Zksh - ShangMi Suite: SM3 Hash Function Instructions
   2.9. Zkr - Entropy Source Extension
   2.10. Zkn - NIST Algorithm Suite
   2.11. Zks - ShangMi Algorithm Suite
   2.12. Zk - Standard scalar cryptography extension
   2.13. Zkt - Data Independent Execution Latency

K Extension

Memory Block Overview

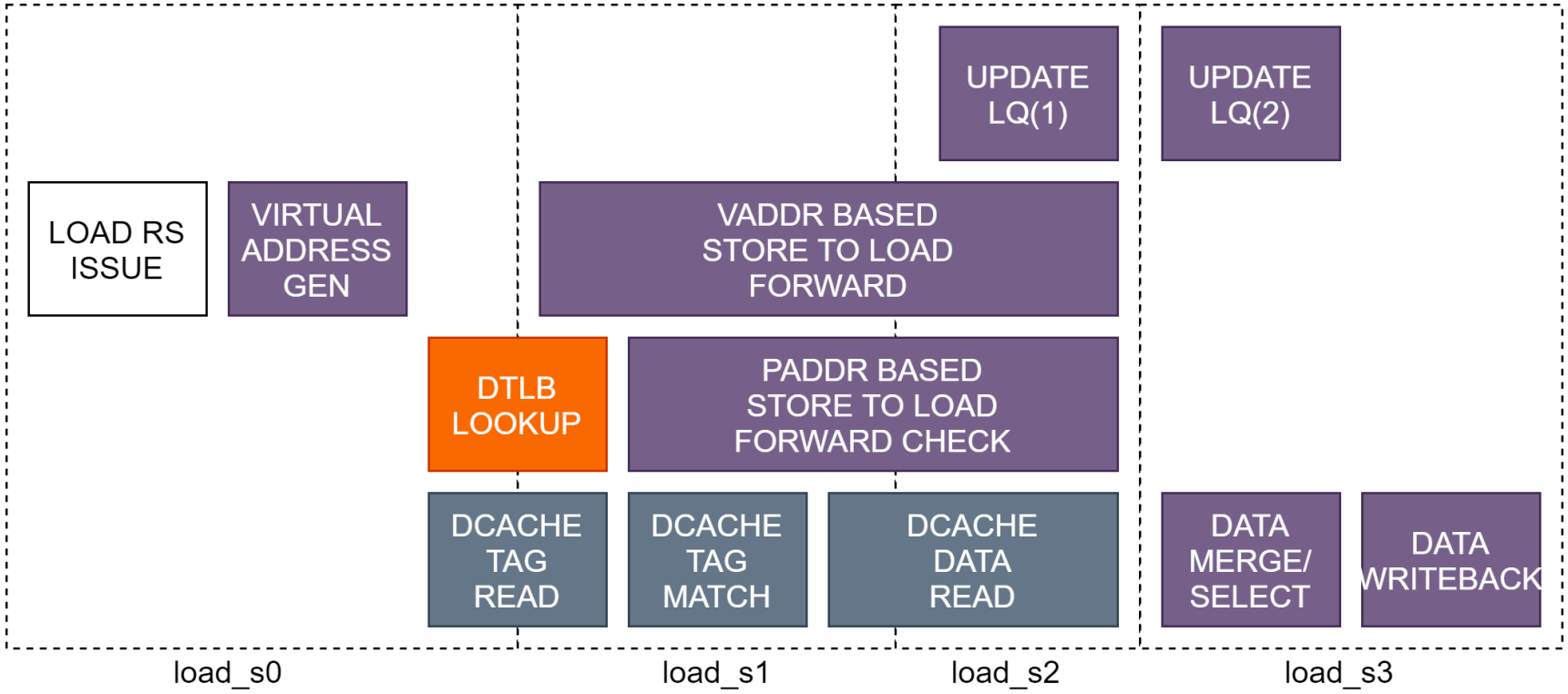
- Datasheets

Load pipeline	**2**
L1 LD hit bandwidth	2x8B/cycle
L1 LD hit latency	**4**
Store address pipeline	**2**
Store data pipeline	2
Store data bandwidth	2x8B/cycle
Load Queue Entry	80
Store Queue Entry	64
Merged Store Buffer	16 cachelines
L1 Data cache size	**64KB/128KB-8w**
DTLB size	**64 + 16**
L2 TLB size	2k entries



MemBlock — Load Pipeline

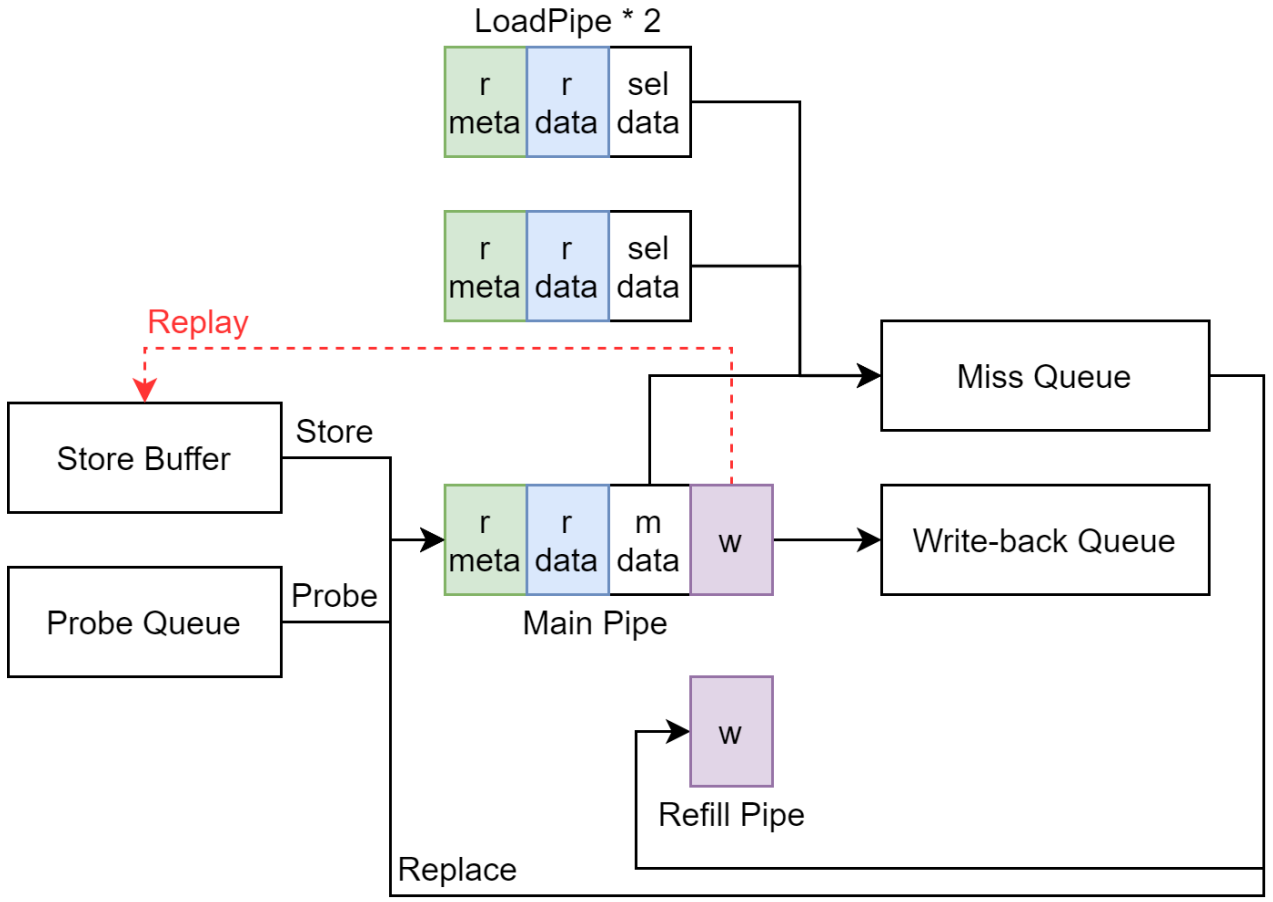
- Load Pipeline stages: three(YQH) -> **four** (NH)
- Adjust the read & write logic of tag / data / LQ
- Optimized for load-load bypass & load-store forward



MemBlock — L1 DCache

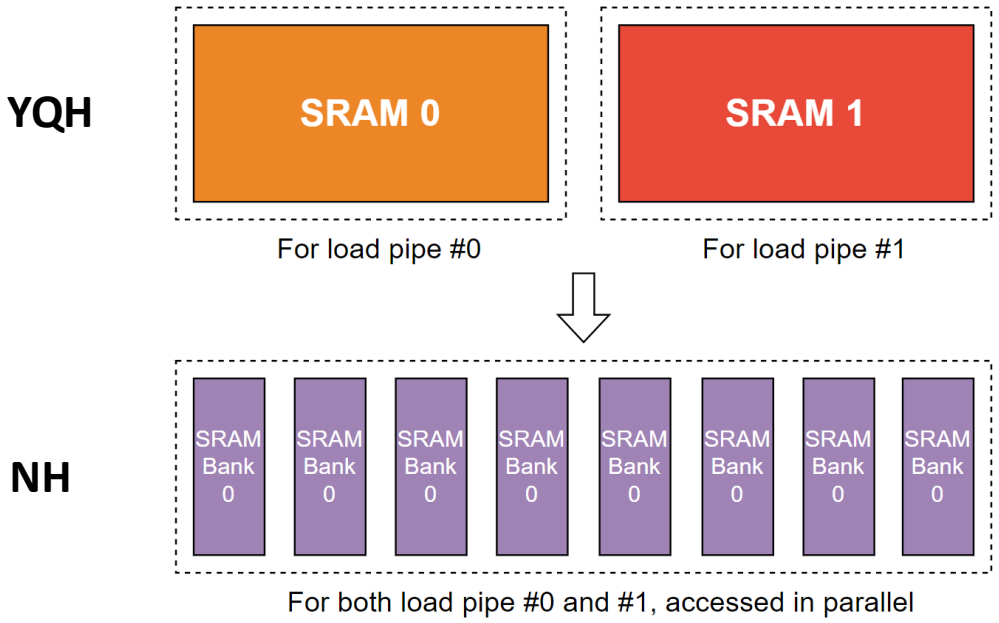
- Optimize store buffer and miss handling logic
- Support larger capacities
- I/D Cache coherence

L1 Data cache size	**64KB/128KB-8w**
L1/L2 Bus Width	256bit
Store Buffer	16
Probe Queue	8
Miss Queue	16
Write-back Queue	18
Software Prefetch	**Support**
ECC	**Support**

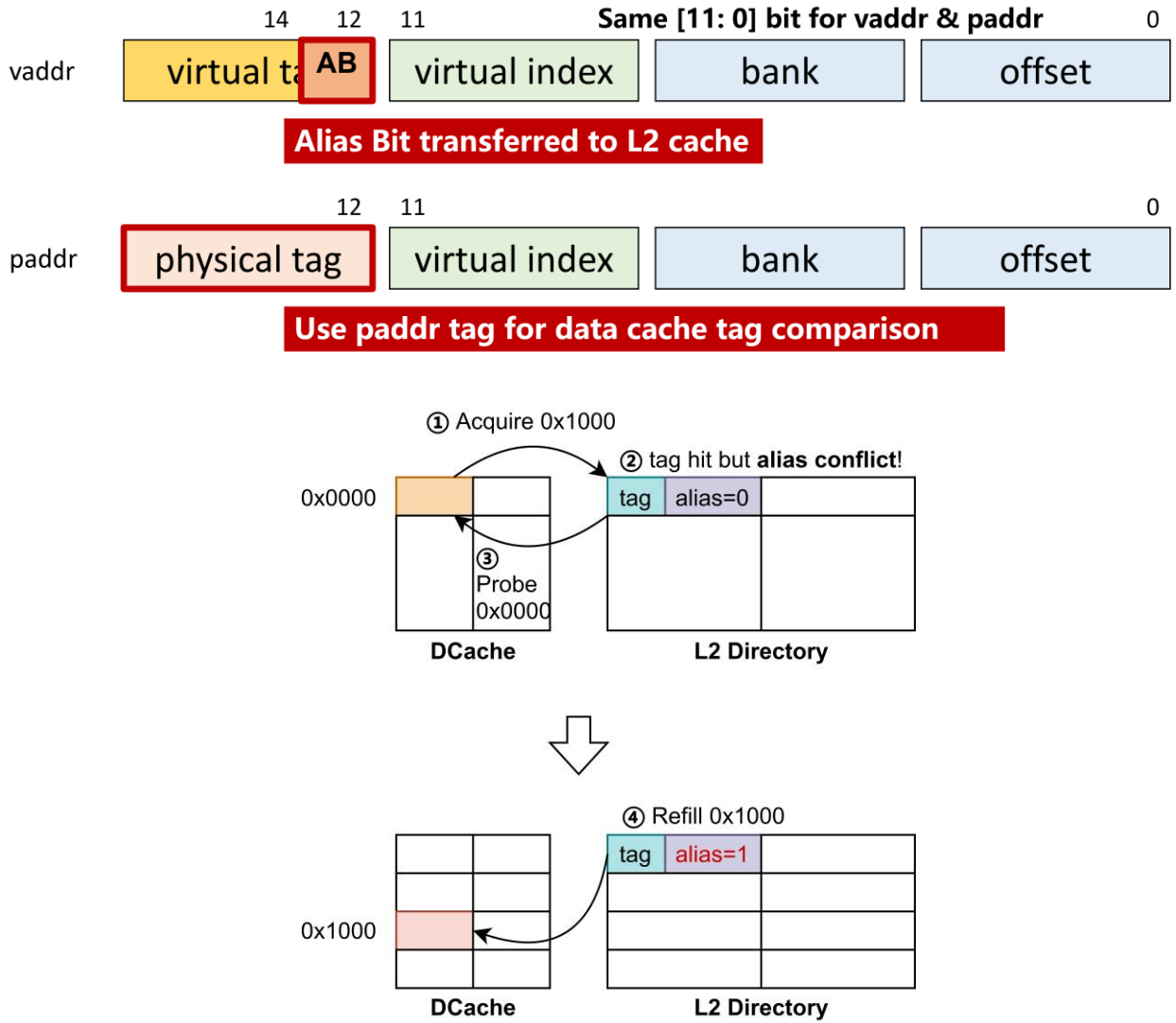


MemBlock — L1 DCache

- Support larger cache capacities
  - Multi-bank for multi read ports
  - Anti-alias under larger size



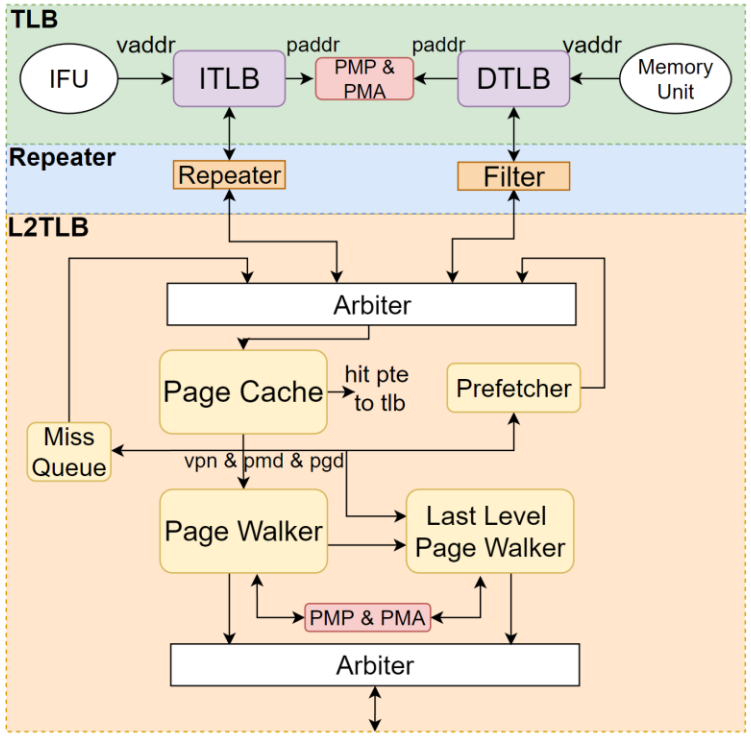
Multi-Bank

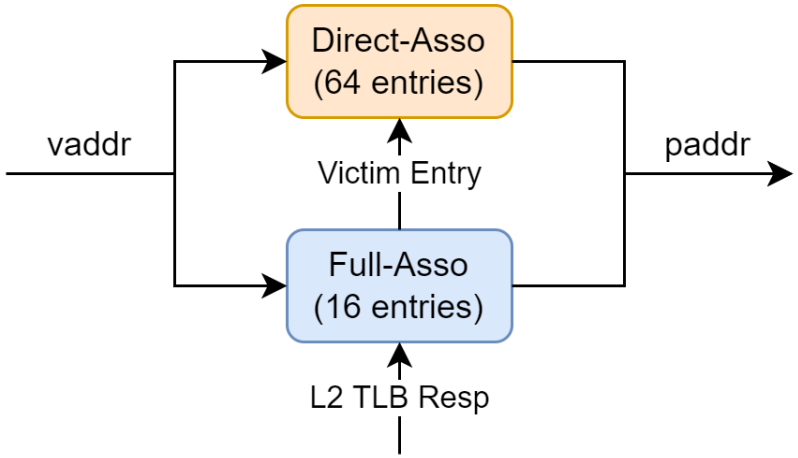


Anti-Alias processing flow

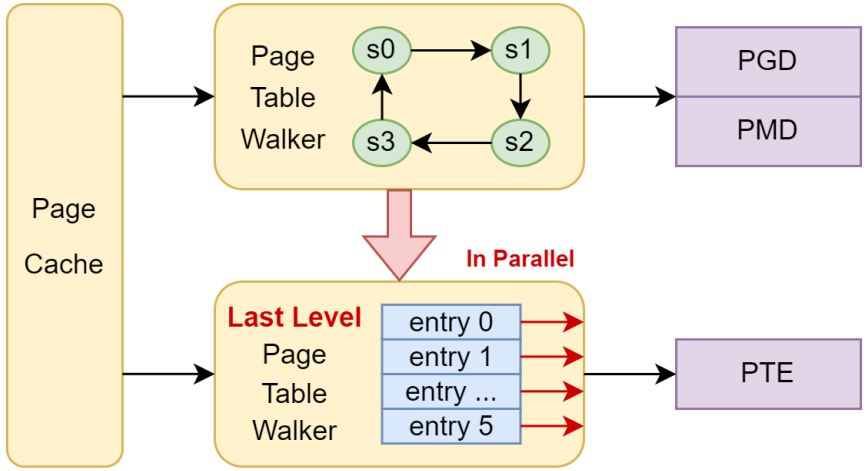
MemBlock — Memory Management Unit

- Principal Improvement:
  - Larger Data TLB size
  - Parallel Page Table Walks





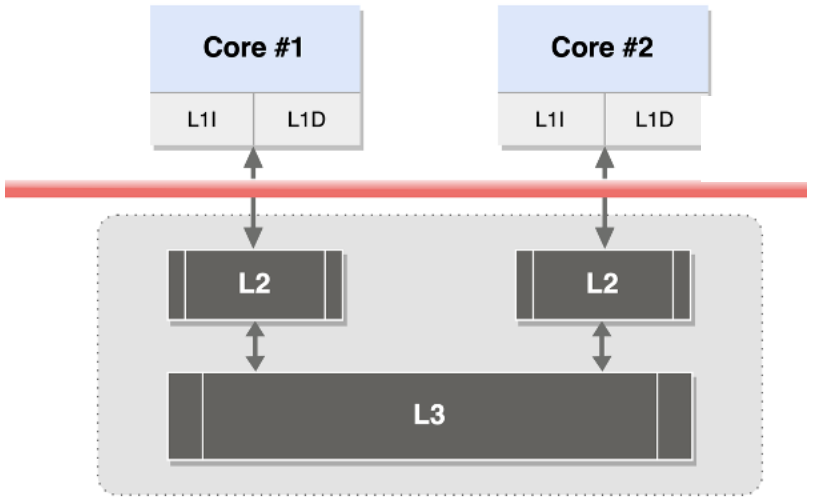
DTLB Structure



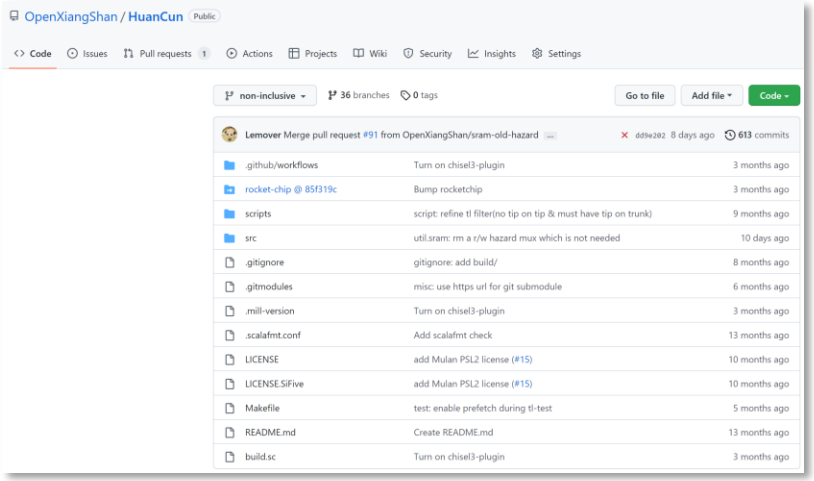
Parallel Page Table Walk

L2/L3 Cache (HuanCun)

- High performance non-blocking **L2/L3** cache
- Design features
  - **Directory based coherence**
  - **Optional Inclusive/Non-inclusive**
  - **TileLink protocol**
  - **Optimise latency and improve concurrency**
- Highly configurable parameters
  - #Size, #Banks, #Ways, #MSHRs
  - Replacement: **Random/PLRU/RRIP**
  - Prefetch: **BOP/SMS**



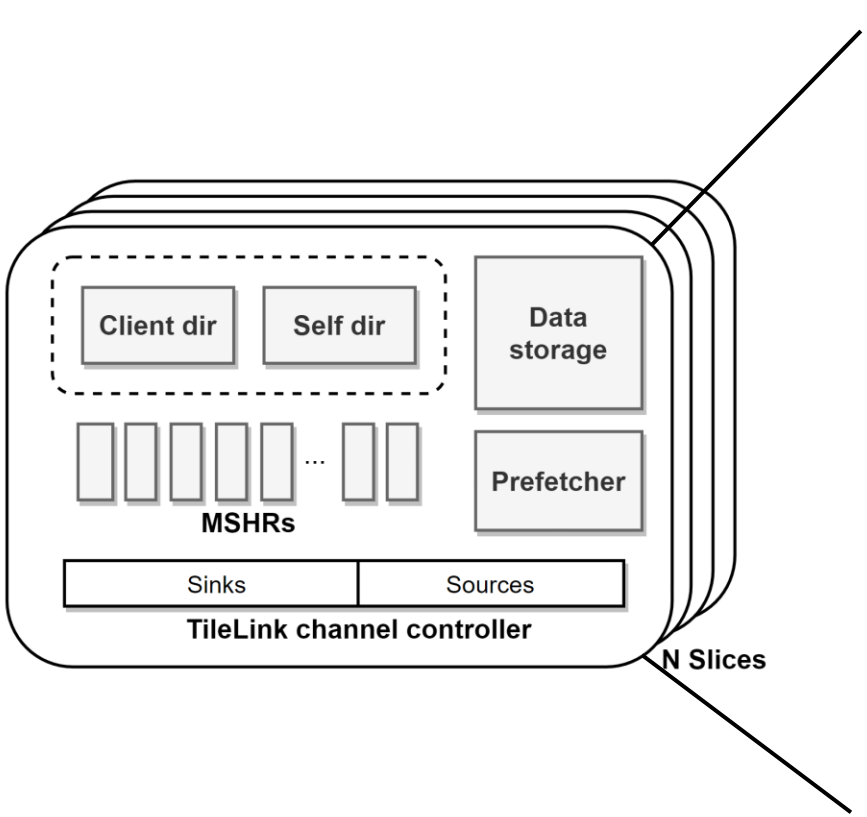
Nanhu Dual-core

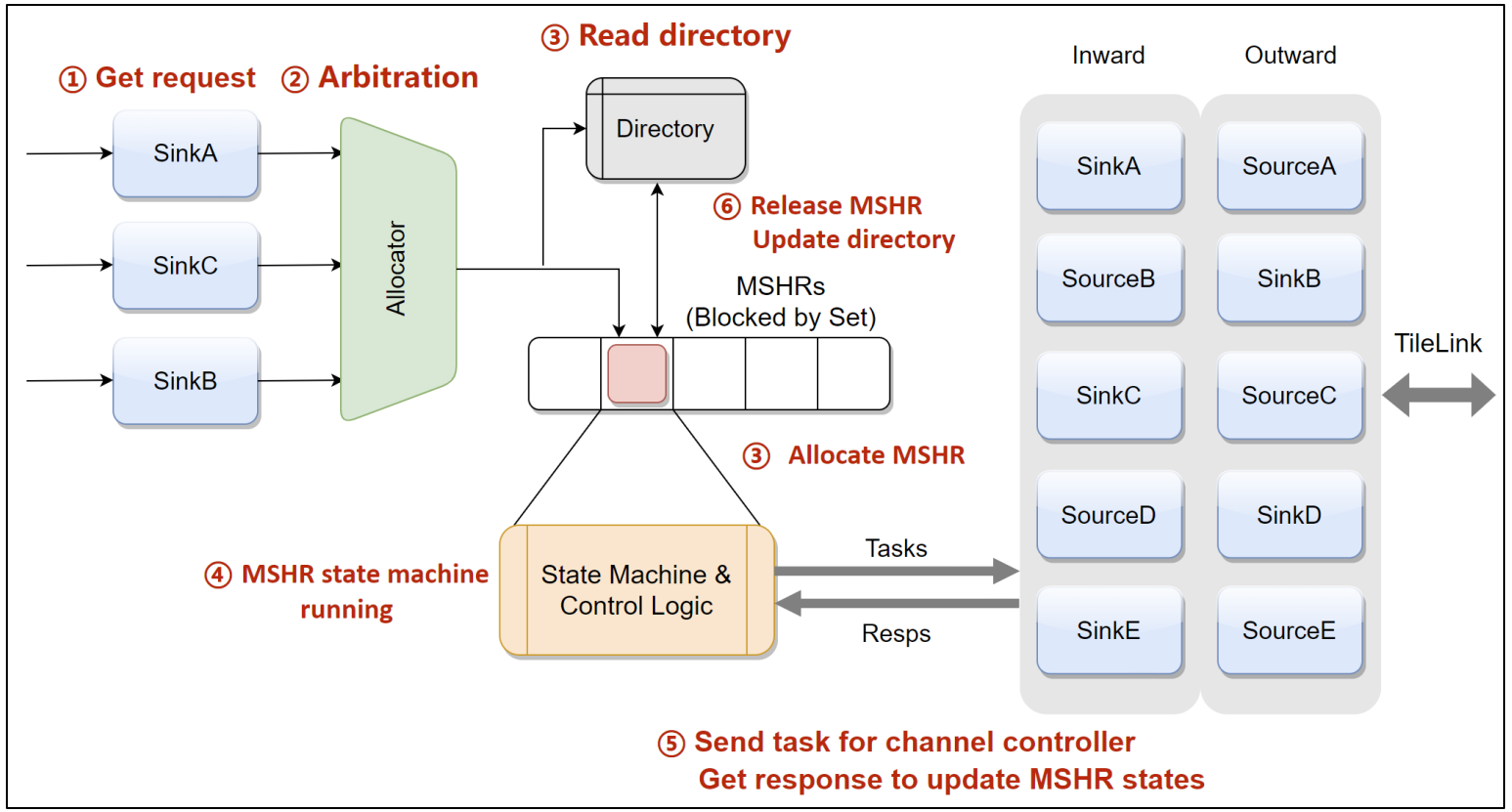


https://github.com/OpenXiangShan/HuanCun

L2/L3 Cache (HuanCun)

- Transaction processing flow

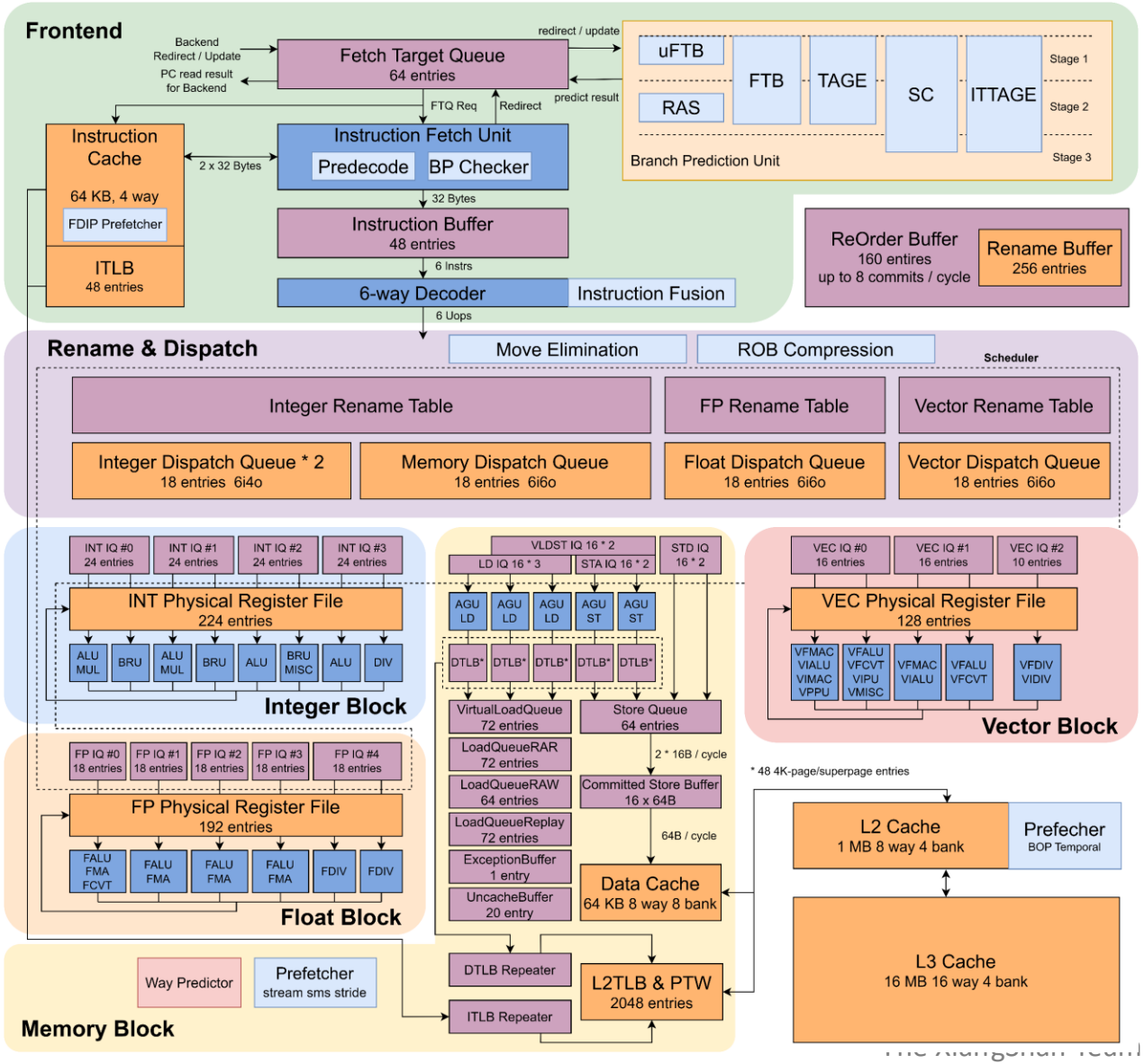




Overview: XiangShan Gen 3 Kunminghu (KMH)

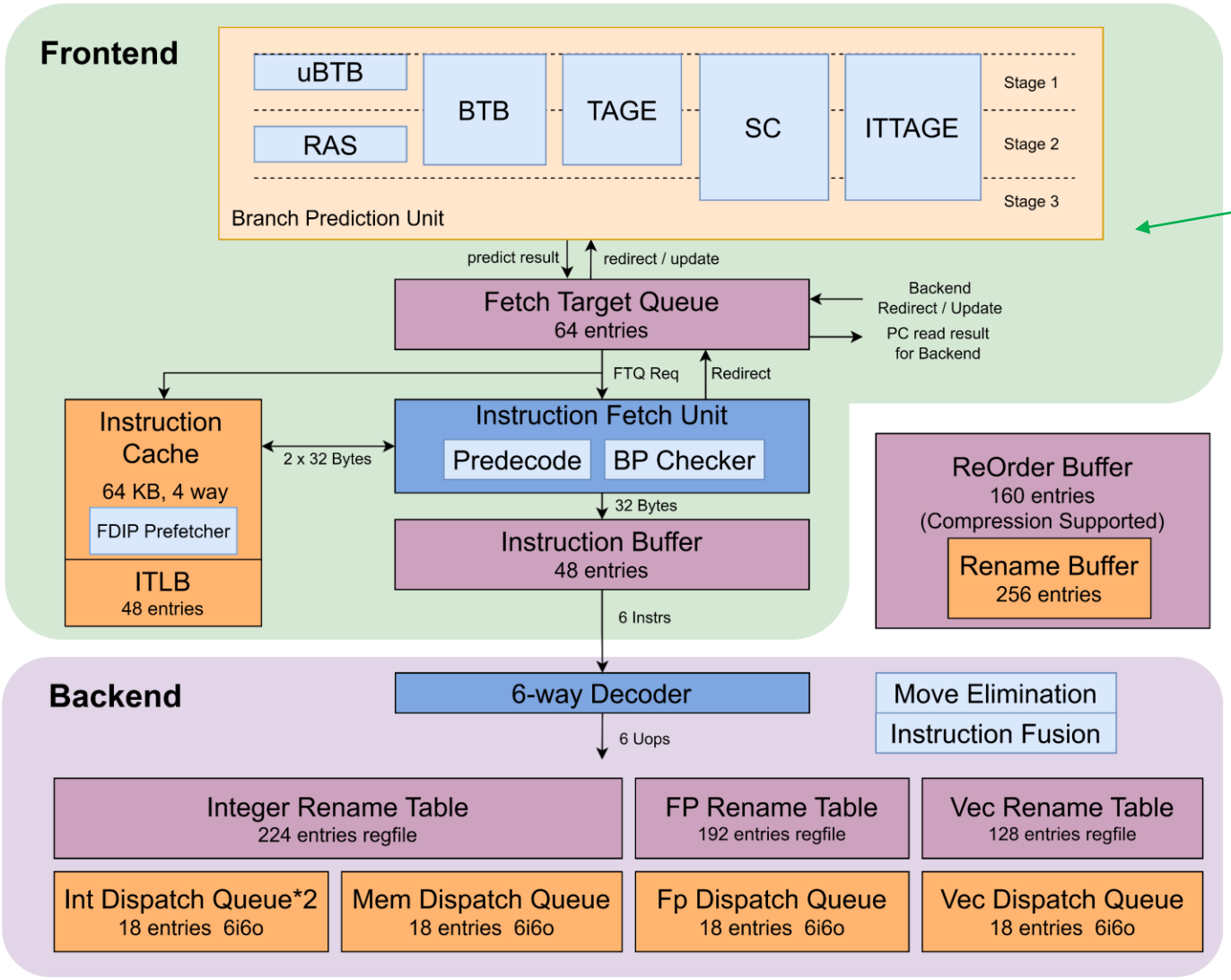
- **Work with Beijing Institute of Open Source Chip (BOSC)**
  - Joint validation with partner companies based on open source
- **Functional Improvement**
  - Support RISC-V **RVA23 profile**
  - RISC-V **Vector** and **Hypervisor** extension implementation
- **Performance Exploration**
  - Performance upgrade of Frontend, Backend, load-store unit and cache
  - DSE on performance model (**XS-Gem5**) calibrated with RTL
- **Functional Verification & Physical Design**
  - Hierarchical verification flow including **UT, IT, ST + FPGA**
  - Industrial-grade verification process
  - Simultaneous iteration of RTL coding with physical design timing evaluation

3rd Kunminghu µArch Overview



- **Decoupled frontend**
  - Reduce fetch bubbles
  - Instruction prefetching friendly
- **Aggressive outstanding instruction window**
  - Large ROB/LQ/SQ
- **Low latency & high BW $ access**
  - Closely-coupled load/store pipe & L1/L2$
  - Highly-banked design
  - Multi-level composite prefetching
- **ISA support**
  - Support RISC-V RVA23 profile
  - RVWMO (RISC-V Weak Memory Ordering)

3rd Kunminghu µArch Overview

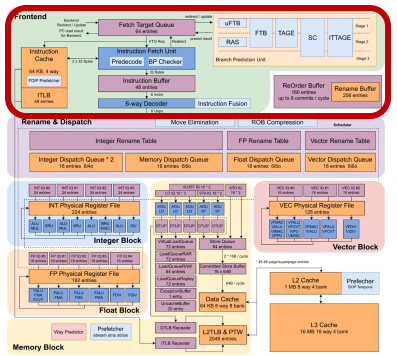


Multi-level branch predictors

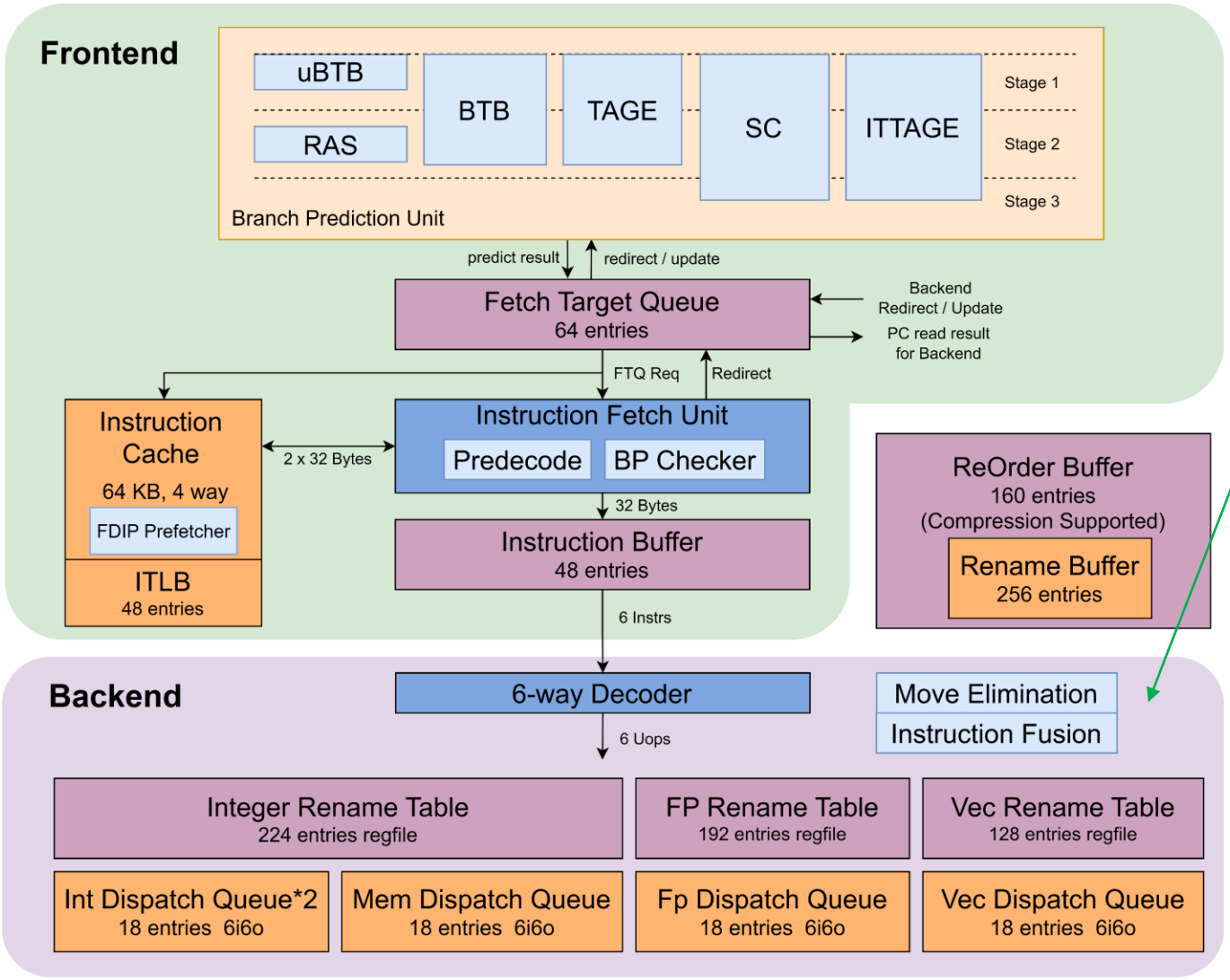
- **256**-entry µBTB
- **2K**-entry BTB, optional L2 BTB
- **16K** TAGE-SC direction predictor
- **2K** ITTAGE indirection predictor
- **48**-entry return address stack

Instruction cache and TLB

- **64KB 4-way** ICache
- **48**-entry ITLB
- Fetch-directed instruction prefetcher



3rd Kunminghu µArch Overview



6-wide decode/rename/dispatch

Register rename

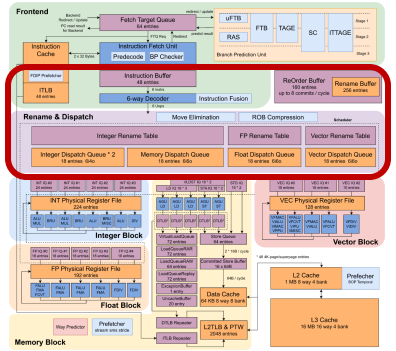
- **224**-entry Int regfile
- **192**-entry FP regfile
- **128**-entry Vec regfile
- Move elimination
- Instruction fusion

160 entry ROB

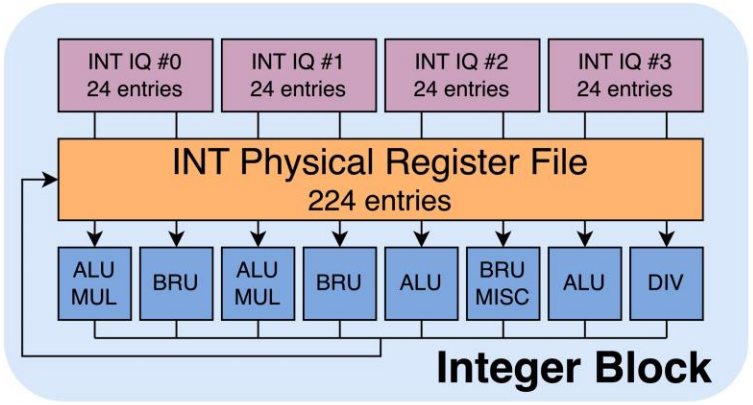
- Support compression (up to **6**-µop/entry)
- **8**-entry retire/cycle
- Recovery via checkpoint + rollback

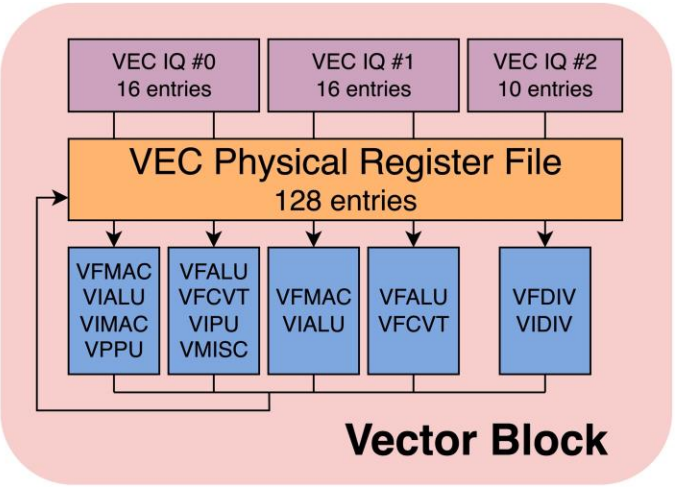
256 entry Rename Buffer

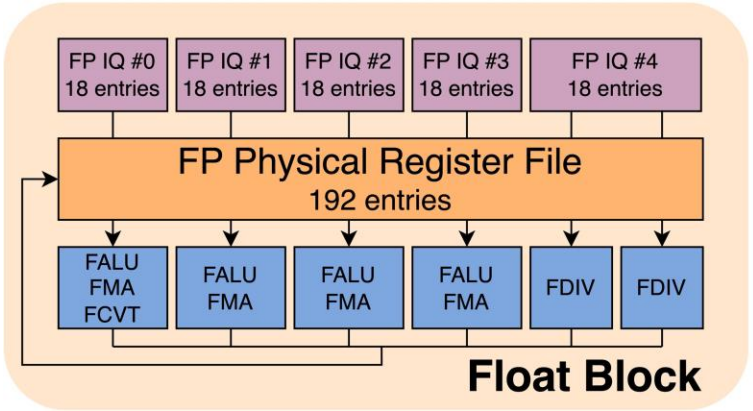
- Bridge the gap between commit & rename table update



3rd Kunminghu µArch Overview







Integer block

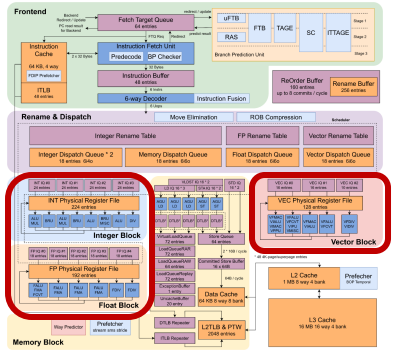
- **4** ALU (2 with 3-stage multiplier)
- **3** BRU for branch resolution
- **1** SRT radix-16 divider

Float-point block

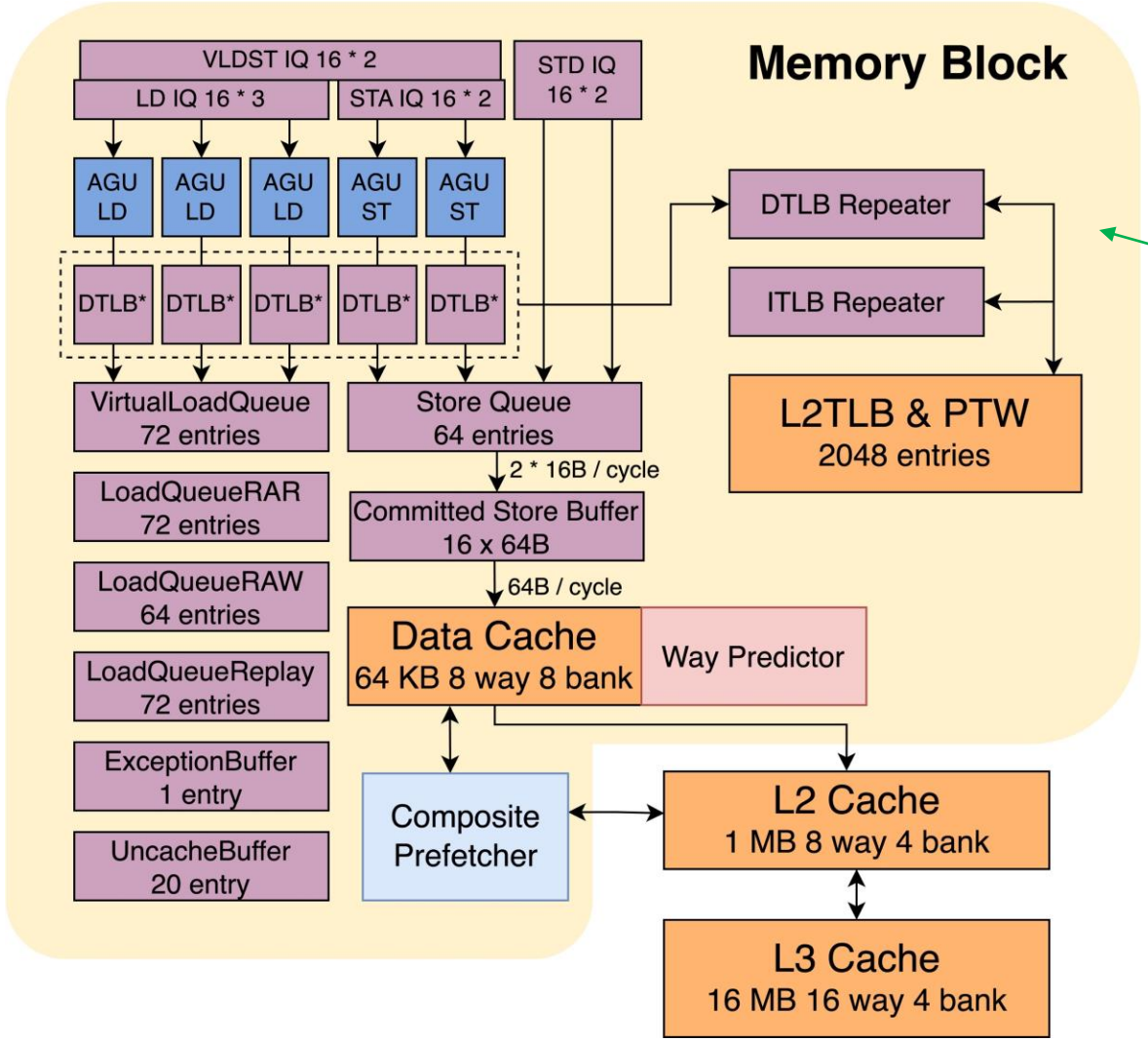
- **4** FPU + **2** FP Div

Vector block

- **4** VPU + **1** Vec Div
- **V1.0** SPEC (VLEN = **128**)



3rd Kunminghu µArch Overview



Load/Store pipeline & queue

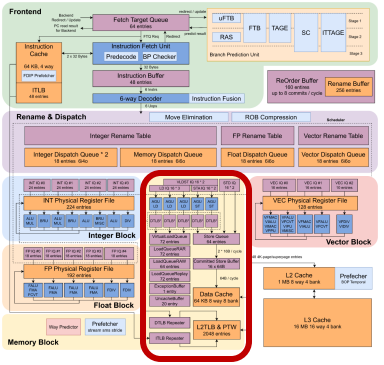
- **3** load pipes, **2** store pipes
- **72** inflight load, **64** inflight store

MMU

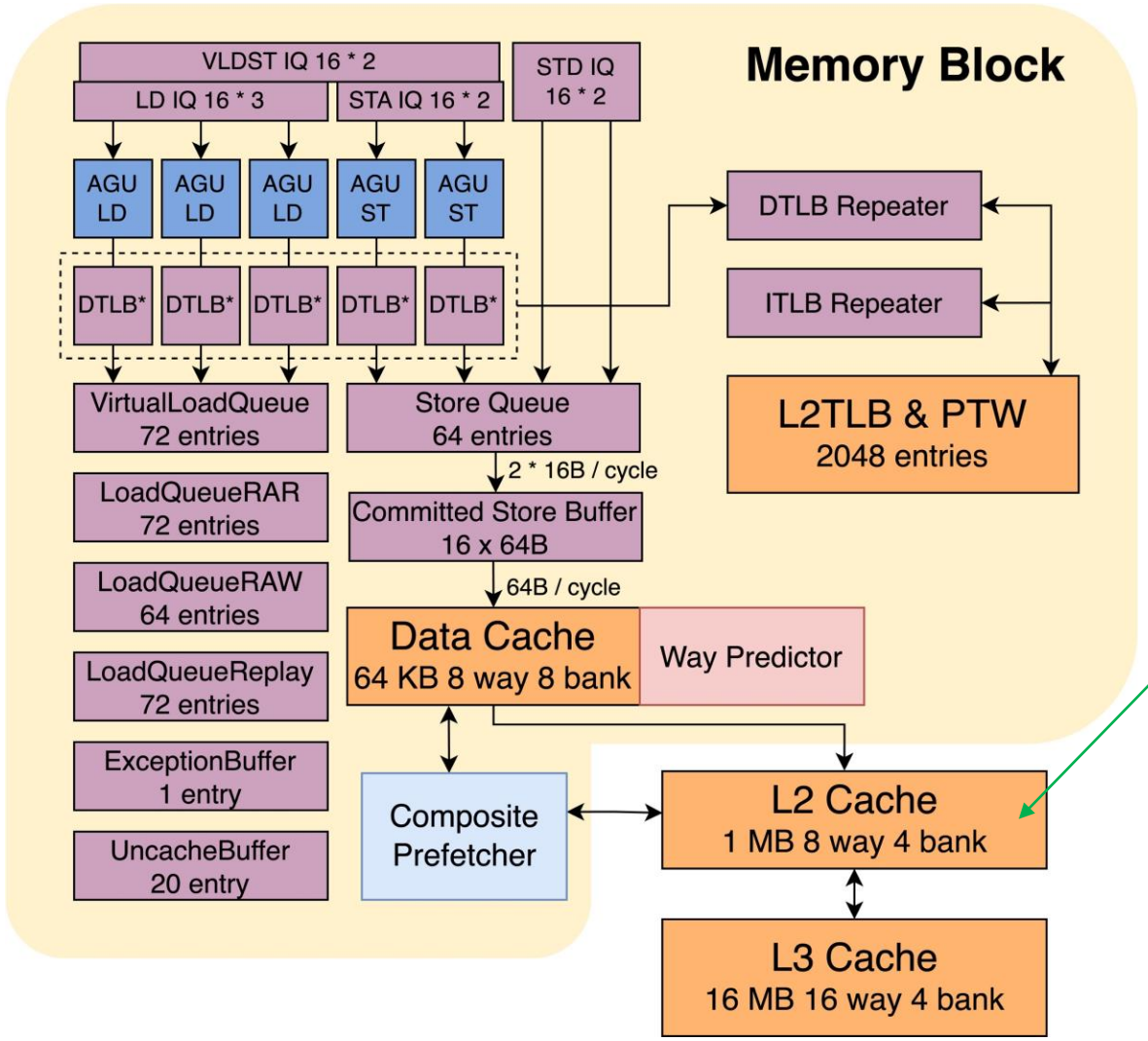
- **48b** virtual/**52b** physical addressing
- **48**-entry L1 DTLB, **2K**-entry L2 TLB
- Up to **7** outstanding page table walks
- Nested walk for virtualization

Data cache

- **64KB 8-way** VIPT (HW anti-aliasing)
- Way predictor for power efficiency
- Composite prefetcher
  - Stream & Stride prefetching
  - SMS & BOP prefetching
  - Temporal prefetching



3rd Kunminghu µArch Overview

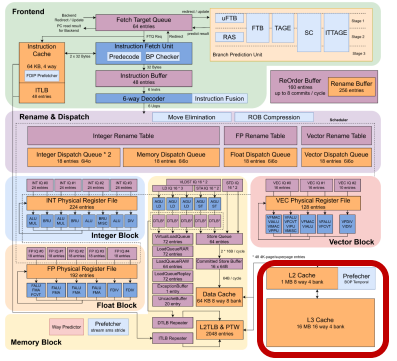


Private L2 cache

- **8-way**, up to **1MB** per core
- Inclusive to D$, non-inclusive to I$
- **64**+ OTs (configurable)
- **10-cycle** load-use latency
- PLRU/RRIP replacement

Shared L3 cache

- **16-way**, up to **16MB**
- Inclusive/Non-inclusive to L2$
- **150**+ OTs (configurable)
- **~30-cycle** load-use latency
- PLRU/RRIP replacement



Performance Evaluation of XiangShan 3rd Kunminghu

- **Method: SPEC CPU checkpoints selected by SimPoint**
  - Compiler: GCC 12 -O3, RV64GCB, jemalloc
  - TileLink bus protocol
  - Cache: 64KB I$/D$ + 1MB L2$ + 16MB L3$
  - Memory modeled by DRAMsim3 DDR4@3200MHz
    - 70ns latency
    - Dual channel, 2x64
- **Evaluation results**

Base score	SPECint 2006	SPECfp 2006
Nanhu@2GHz	**16.94**	**19.42**
Kunminghu*@3GHz	**44.17**	**44.52****

SPECint 2006 est.@ 3GHz		SPECfp 2006 est.@ 3GHz	
400.perlbench	35.88	410.bwaves	66.62
401.bzip2	25.55	416.gamess	40.91
403.gcc	46.67	433.milc	45.21
429.mcf	58.13	434.zeusmp	51.96
445.gobmk	30.34	435.gromacs	33.61
456.hmmer	41.60	436.cactusADM	46.26
458.sjeng	30.50	437.leslie3d	46.10
462.libquantum	122.57	444.namd	28.91
464.h264ref	56.66	447.dealII	73.69
471.omnetpp	39.35	450.soplex	52.00
473.astar	29.24	453.povray	53.42
483.xalancbmk	72.01	454.Calculix	16.38
GEOMEAN	**44.17**	459.GemsFDTD	36.01
		465.tonto	36.74
		470.lbm	91.18
		481.wrf	40.62
		482.sphinx3	48.61
		GEOMEAN	**44.52**

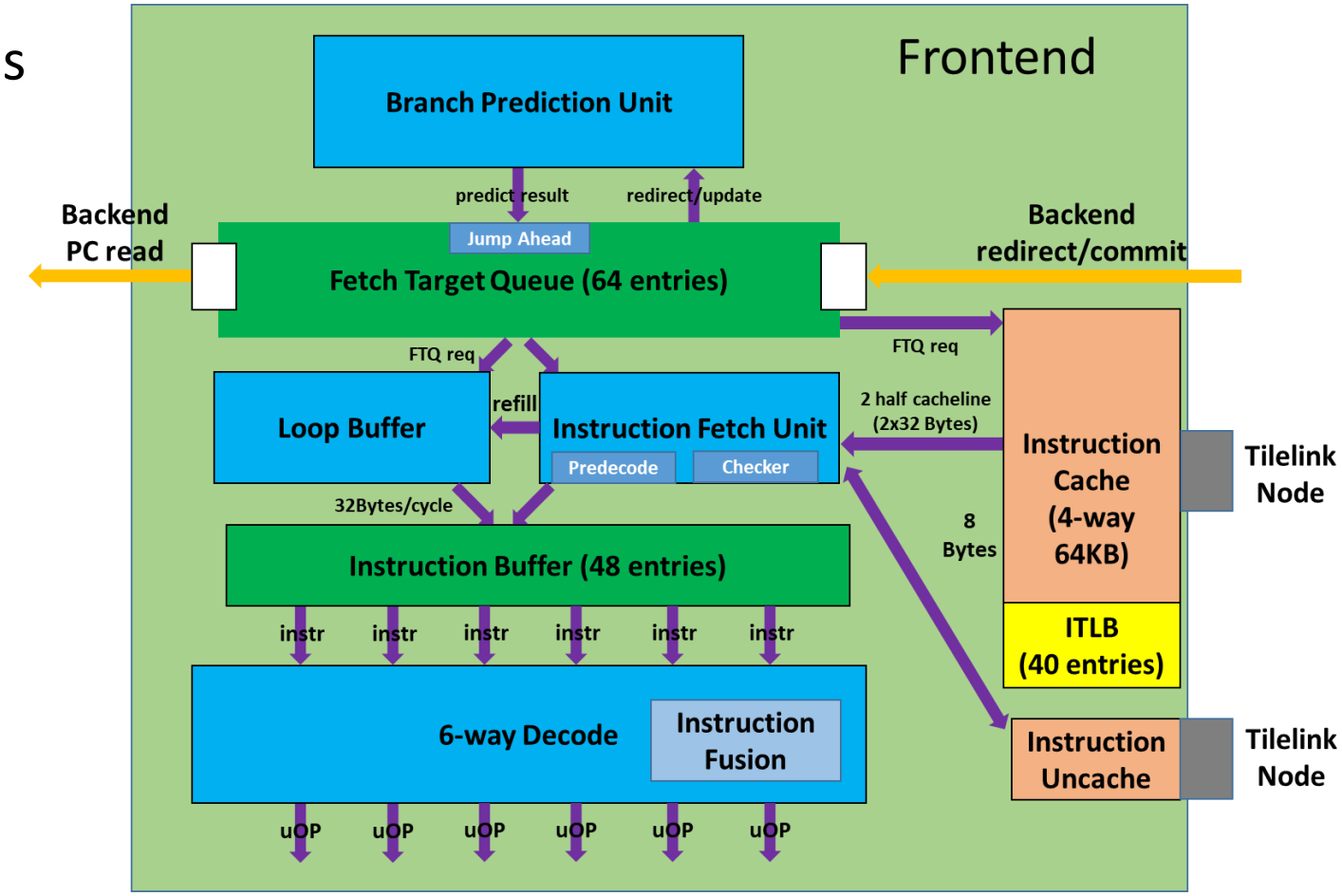
Published by XiangShan Biweekly 70

* Updated in February 2025, XiangShan commitID 7f475a2

** FP performance slightly degraded mainly due to PPA optimisation

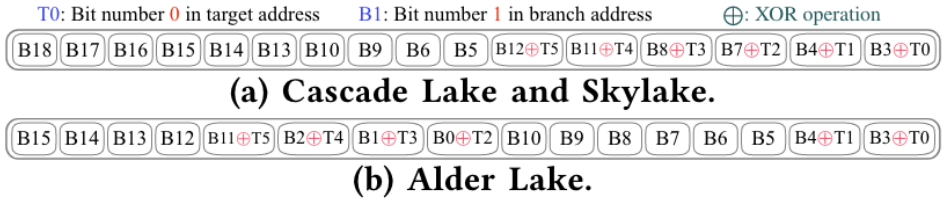
Frontend Upgrade Overview

- Inherit NH decoupled architecture
  - Hiding branch prediction bubbles
  - Guiding instruction prefetch
- Iterative optimization
  - Lower prediction miss rate
  - Larger fetch/prediction width
  - More accurate branch predictor



Frontend Feature 1: TAGE Predictor Fine Tuning

- ❶ **Branch History Hashing**



Hash algorithm affects performance

Genetic and particle swarm algorithms are employed to adjust parameters automatically.

Param of Nanhu → Param tuned: IPC +1.44%

Bonus!

- ❷ **#Table & History Length**

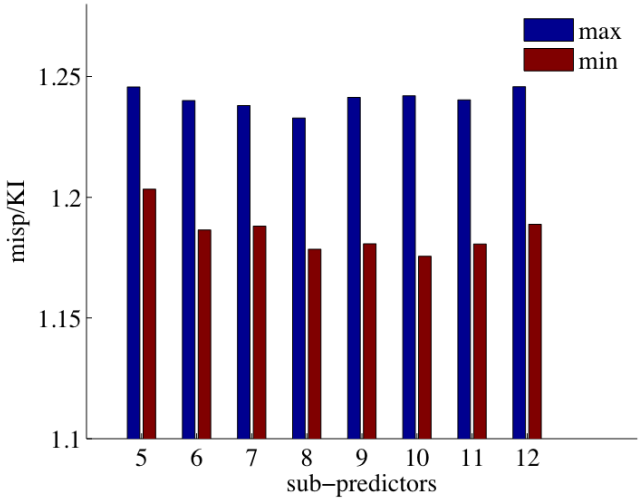


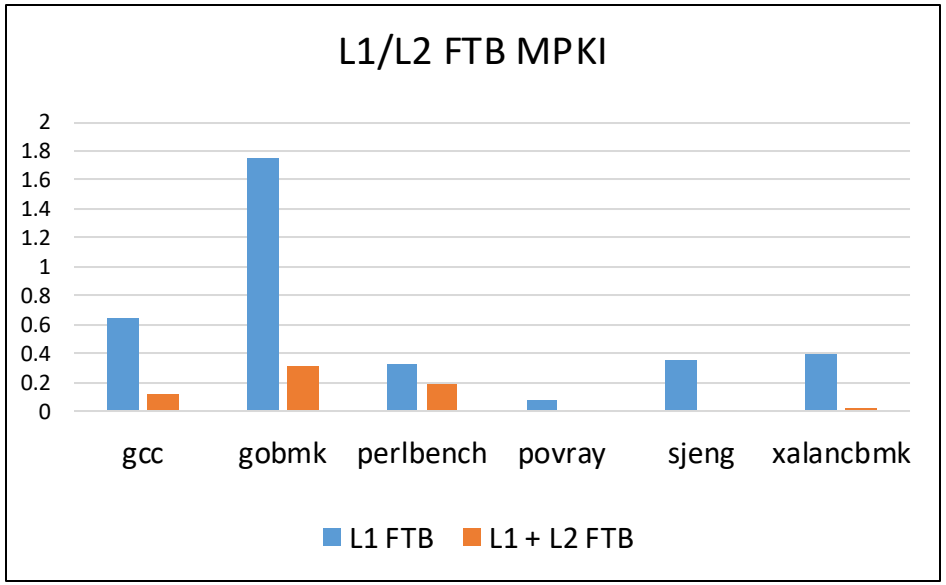
Figure 2: Performance of random parameters

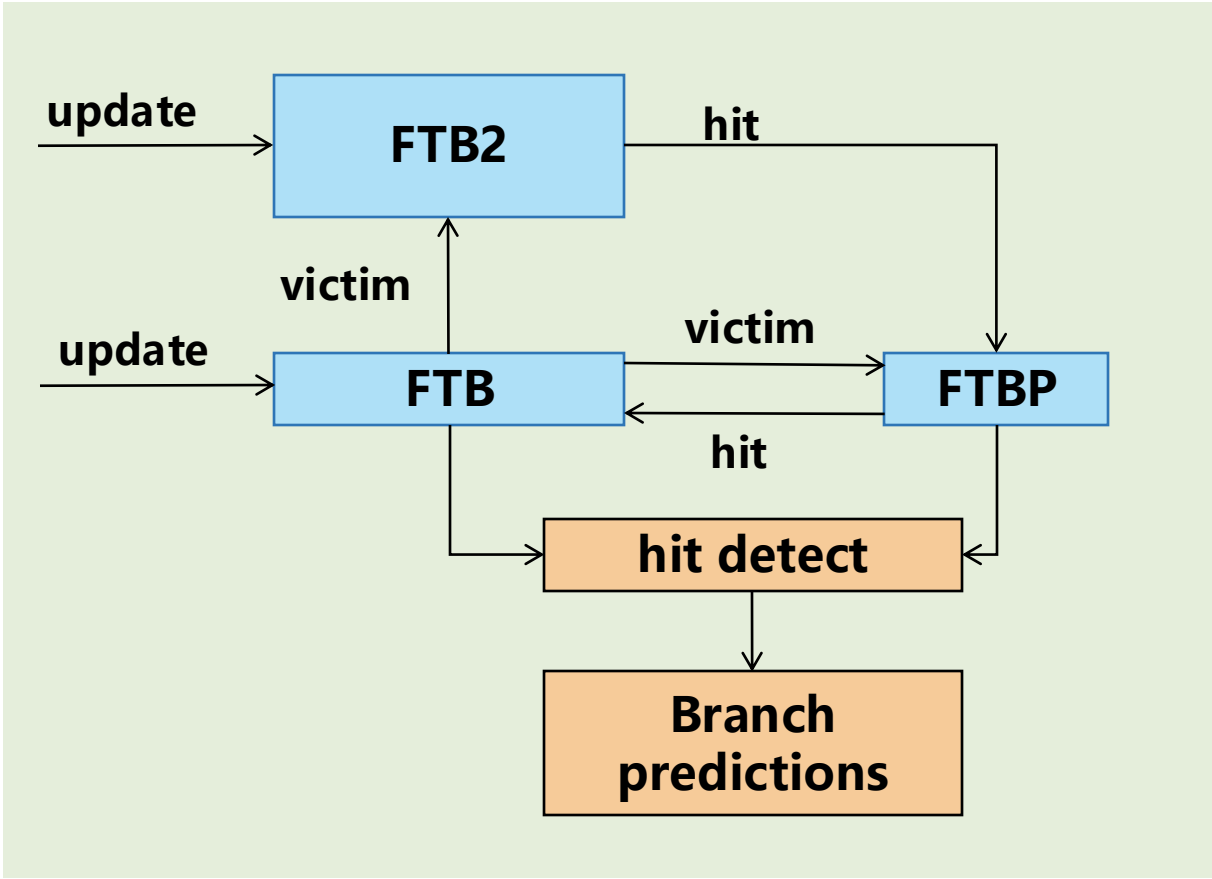
Previous studies have shown that the #tables and history length have a significant effect on TAGE performance, up to 5.64% in MPKI*

* C. Zhou, L. Huang, Z. Li, T. Zhang, and Q. Dou, "Design Space Exploration of TAGE Branch Predictor with Ultra-Small RAM," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, Banff Alberta Canada: ACM, May 2017, pp. 281–286.

Frontend Feature 2: L2 FTB (BTB)

- Hierarchical FTB
  - Support 4K L2 FTB
  - Semi-exclusive
  - FTBP acts as a buffer between L1 & L2 FTB





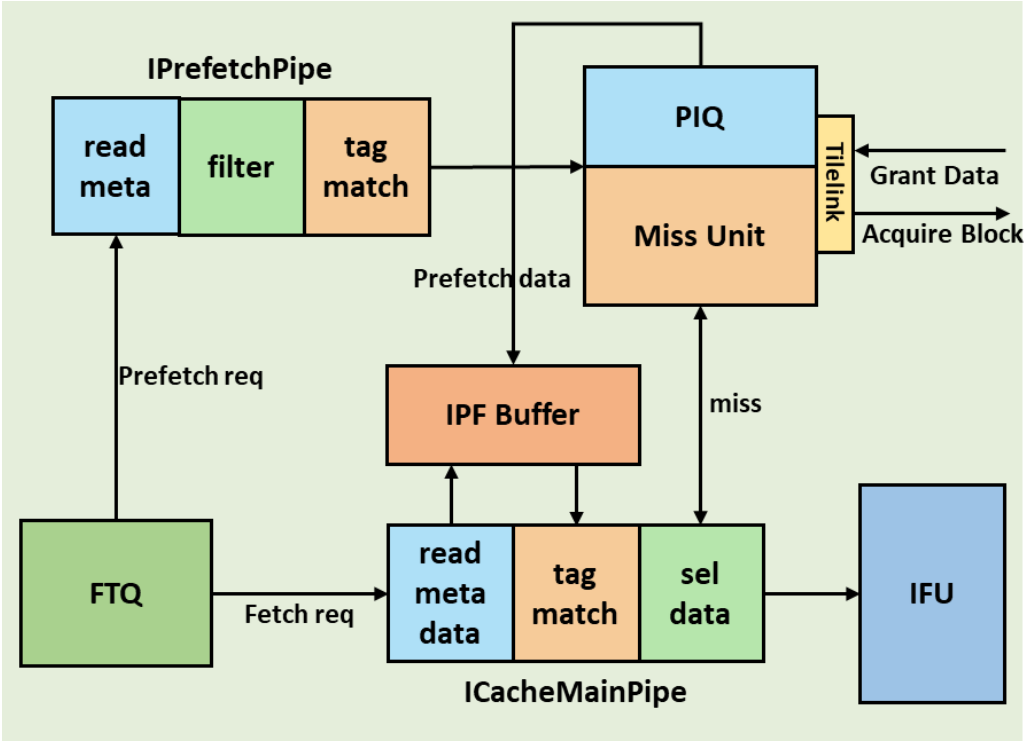
L2 FTB Structure

Frontend Feature 3: ICache Smarter Prefetcher

- Based on decoupled frontend, FDIP algorithm is implemented
- **Prefetcher workflow**

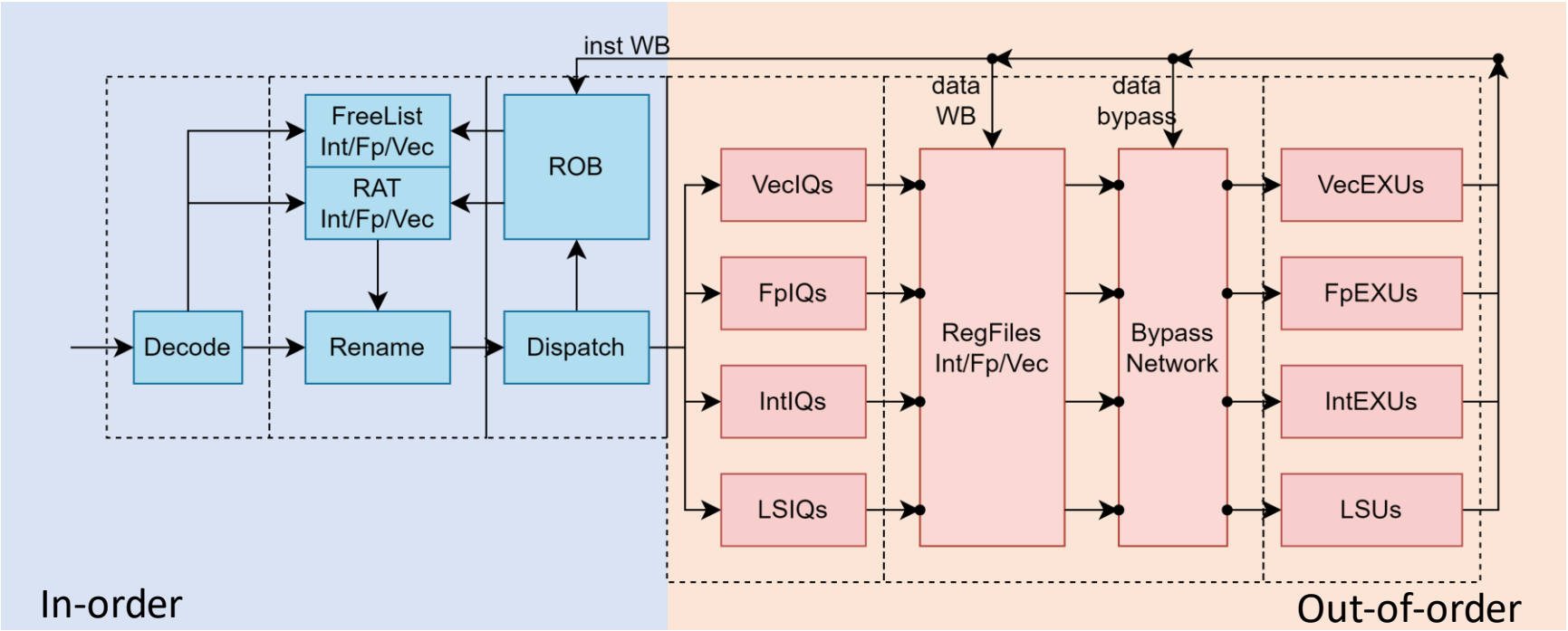
1. Select prefetch request from PF-queue
2. Write refilled data into prefetch buffer
3. Fetch query icache & prefetch buffer parallelly

- **Optimisation compared to nanhu**
  - Prefetch position: L2 Cache -> **L1 ICache**
  - **Dual prefetching pipelines** to enlarge bandwidth



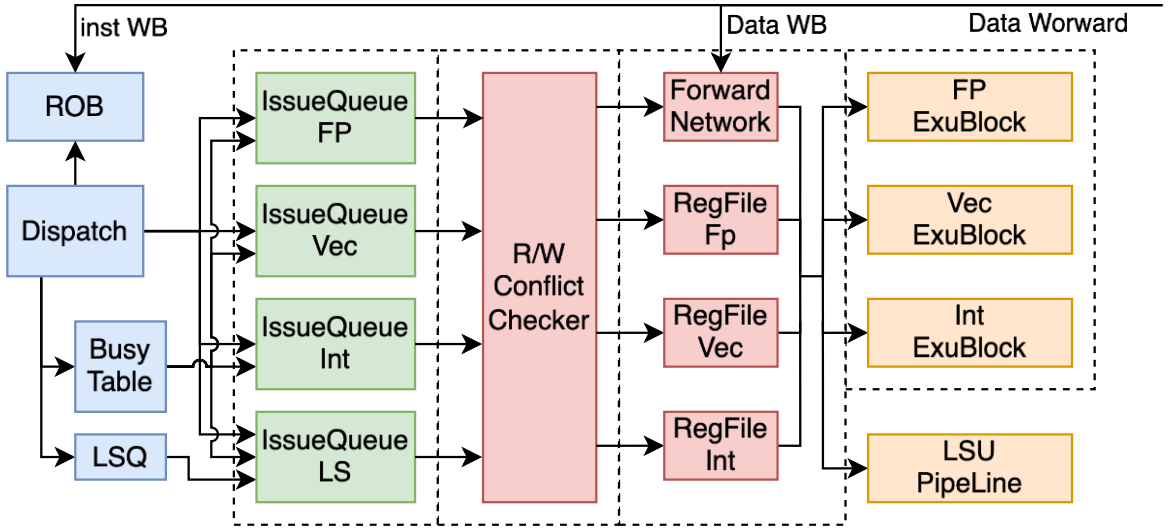
Backend Upgrade Overview

- Read regfile after issue
- Faster µop commit (ROB compression) and redirect (Rename snapshot)
- Vector extension and Hypervisor extension support



Backend Feature 1: Read Regfile after Issue

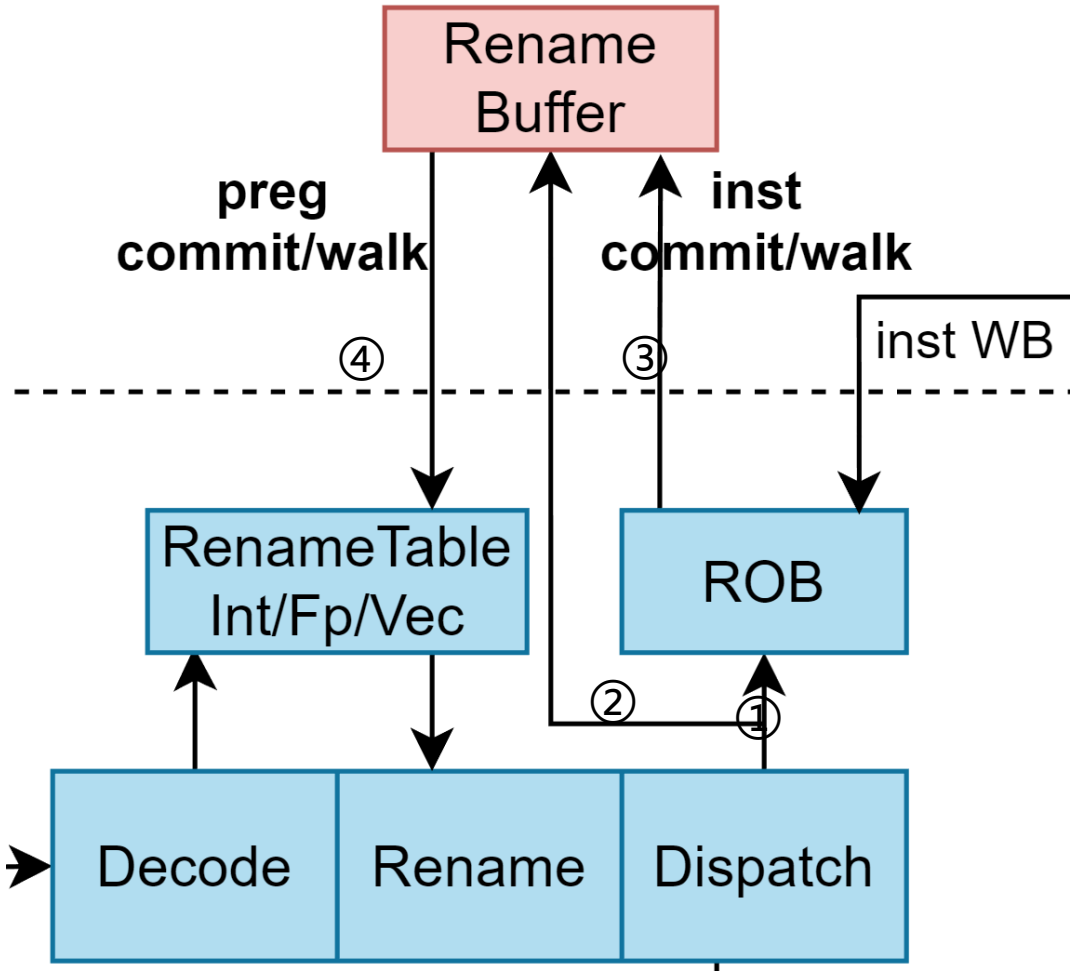
- Advantage
  - Bottleneck is shifted to after µop issue
  - Reduce RS storage to save SRAM
  - Reduce latency and increase RS capacity
- Optimization
  - Efficient arbitration of regfile reading ports
  - Accurate speculative wakeup and cancelling



Read regfile after issue

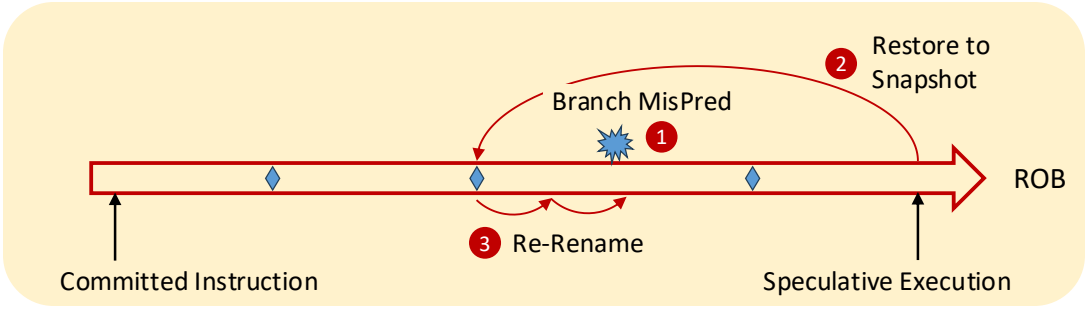
Backend Feature 2: Decouple µop & Regfile Commit

- New Rename Buffer (**RAB**) design: record regfile changing history
  - ① Save Inst in ROB after dispatch
  - ② Save regfile mapping info to Rename Buffer
  - ③ On Inst commit/redirect, wake RAB up
  - ④ RAB is responsible to update Rename Table
- Benefits
  - Faster ROB entry releasing
  - Optimize timing
  - Support µop split of vector instructions
  - Support ROB compression



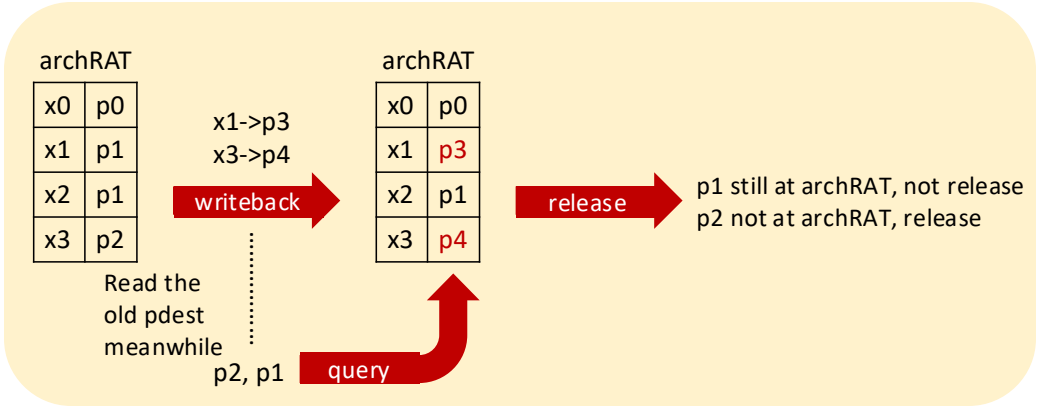
Backend Feature 3: Register Renaming Optimizations

- ① Rename Snapshot: Reduce The Cost of Backend Redirect
- ② Move Elimination Based on RAT: Optimize Area and Power
- ③ ROB Compression: Reduce ROB Consuming and Enlarge OoO Windows

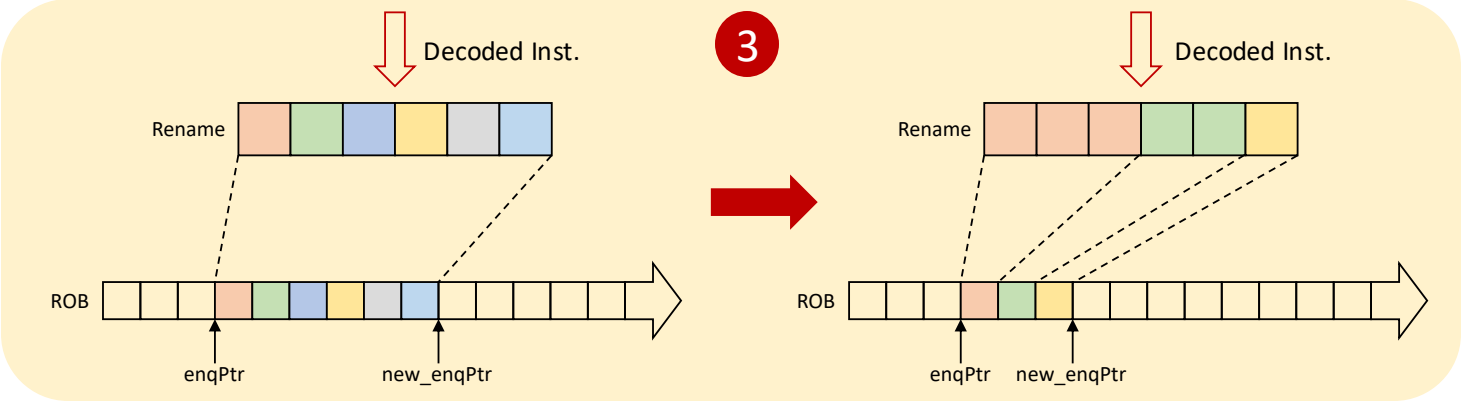


1

2

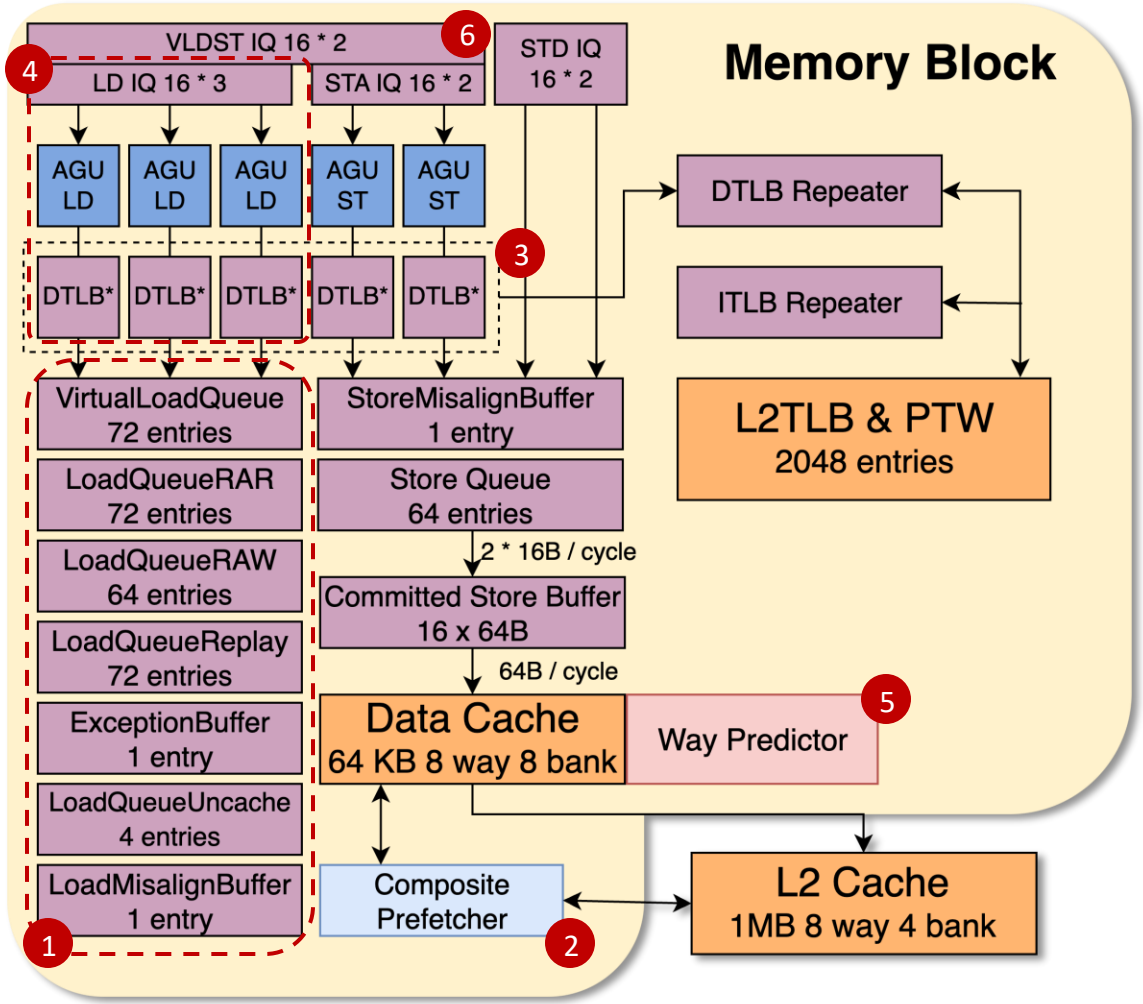


3



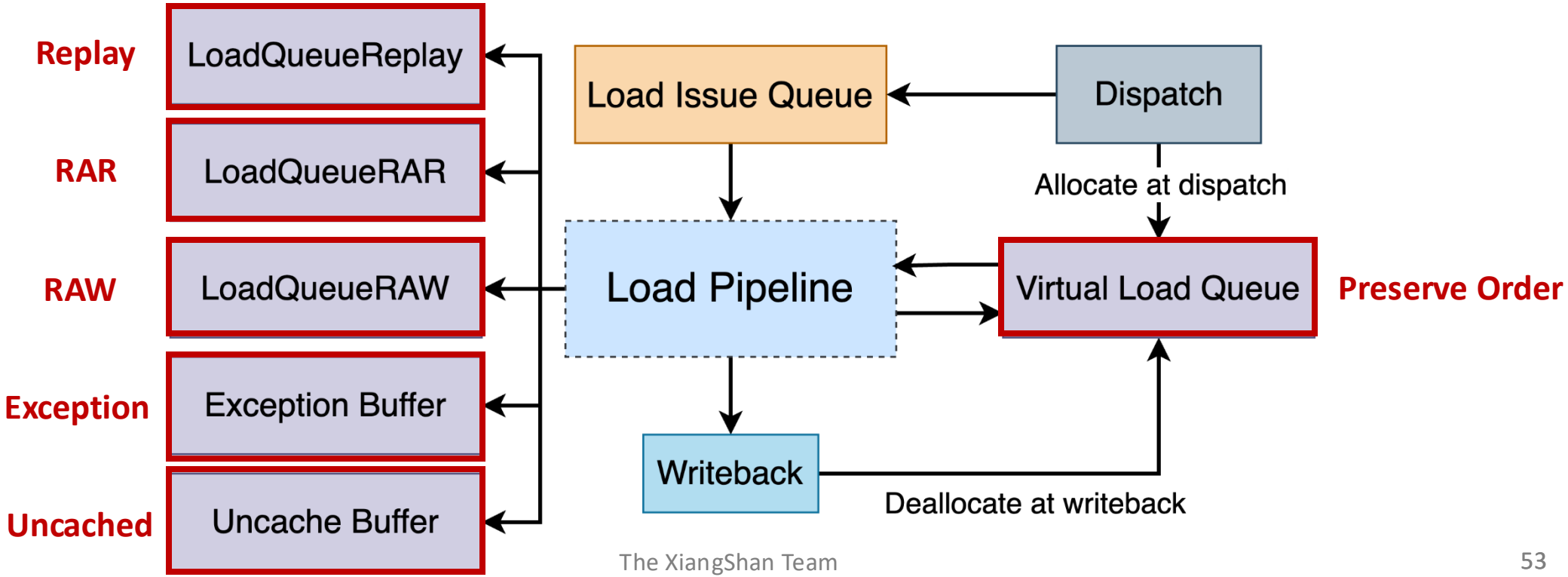
MemBlock Upgrade Overview

- Optimization compared to NH
  - ① Divide load queue
  - ② Prefetch to L1D
  - ③ Increase effective TLB capacity
  - ④ Enlarge load bandwidth
  - ⑤ Implement way predictor
  - ⑥ Vector LSU



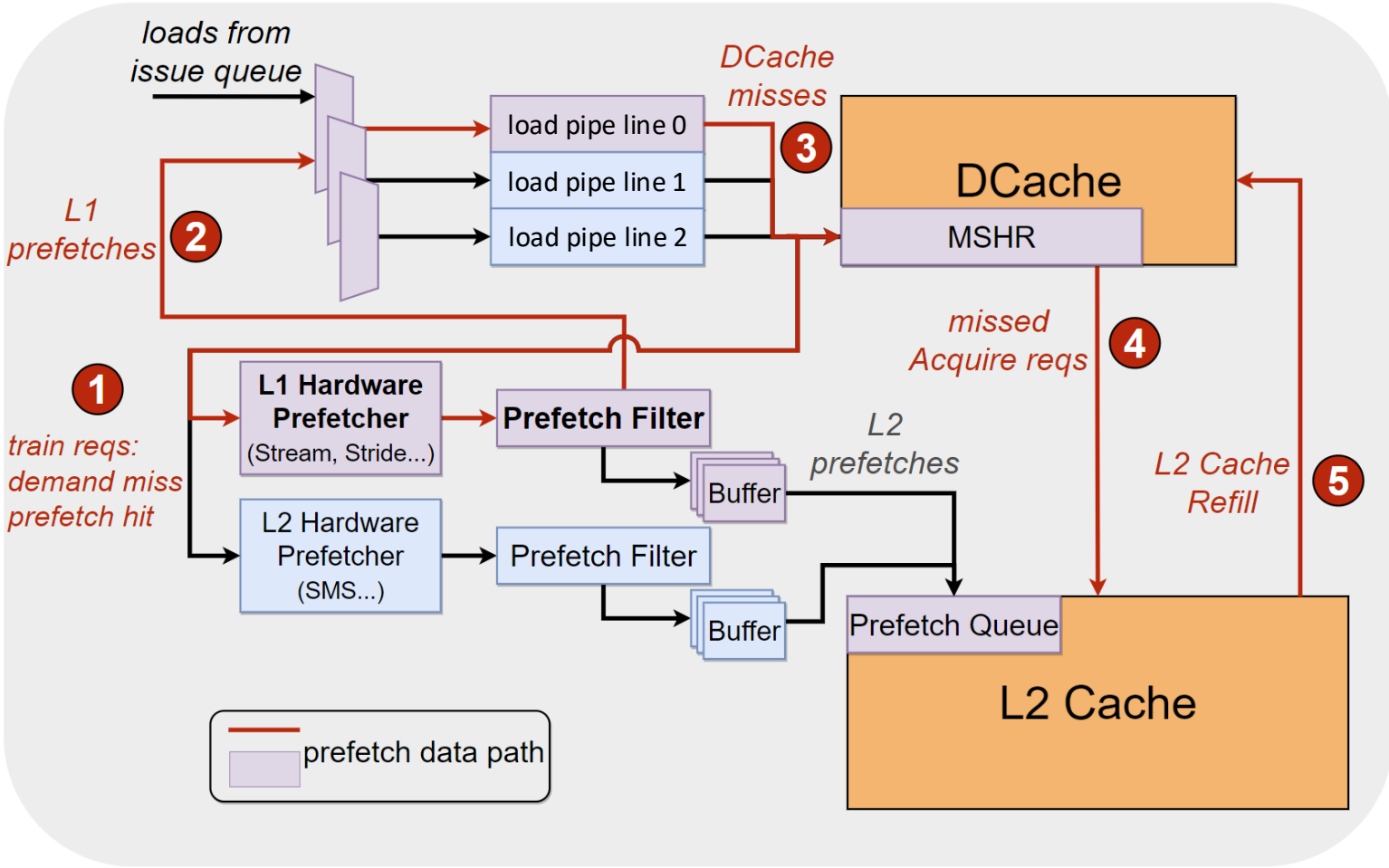
MemBlock Feature 1: Split Load Queue

- Split load queue **based on different responsibilities**
  - Logic simpler, timing friendly and area acceptable
  - **Early commit**: load instruction can be retired sequentially from lq after write-back



MemBlock Feature 2: L1D Prefetch

- L1D prefetch workflow
  - Training pattern detection
  - **Reuse load pipelines**
  - Fetch data from D$ MSHR
- Prefetch algorithm
  - **Stream[1]**
    - (x,x+1,x+2,x+3...)
  - **Stride[2]**
    - (x,x+n,x+2n,x+3n...)

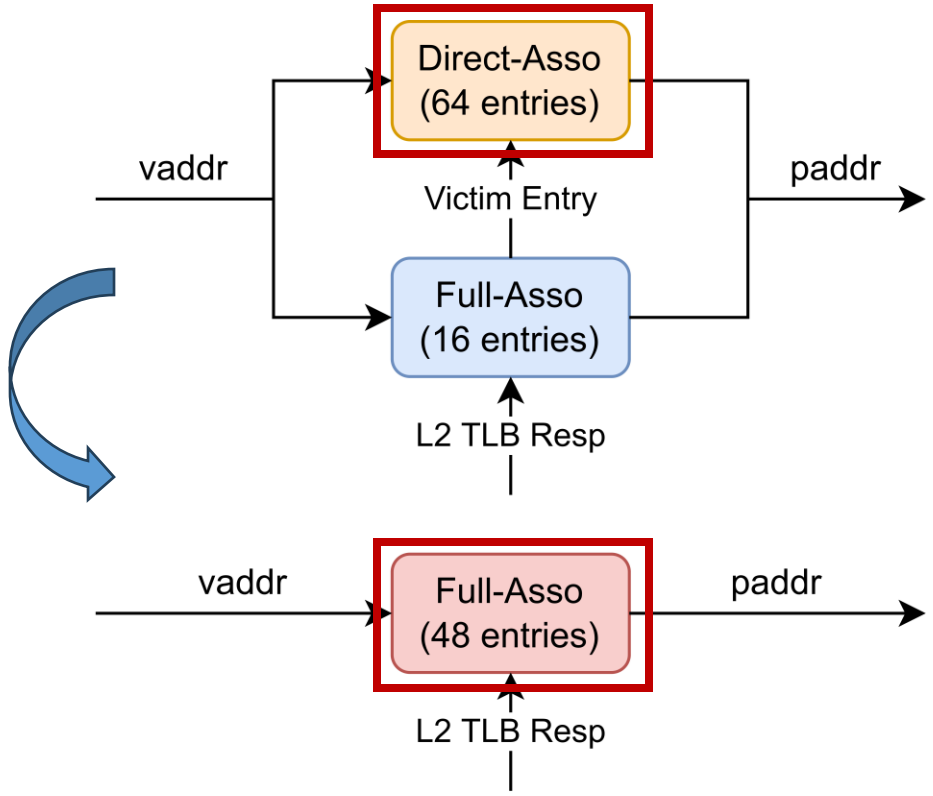


[1] S. Srinath, O. Mutlu, H. Kim and Y. N. Patt, "Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers," 2007 IEEE 13th International Symposium on High Performance Computer Architecture, Scottsdale, AZ, USA, 2007

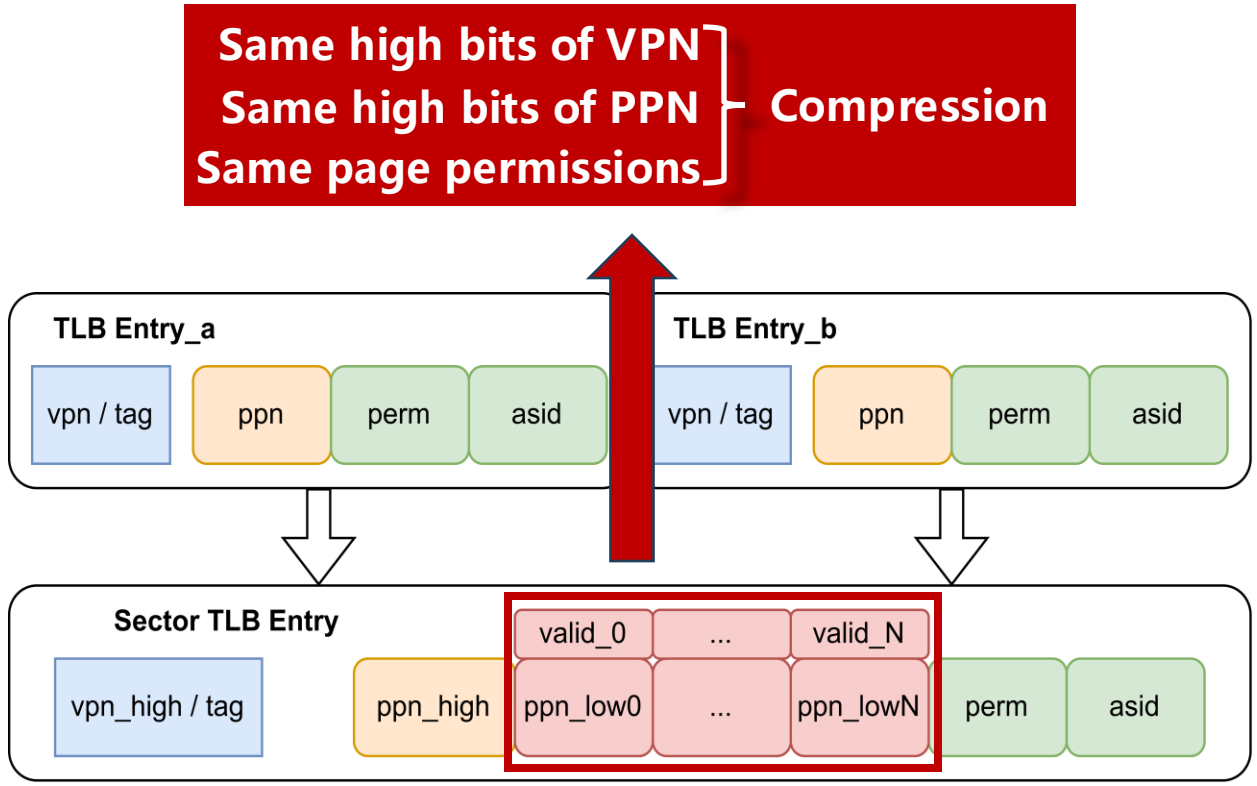
[2] J. -L. Baer and T. -F. Chen, "An effective on-chip preloading scheme to reduce data access penalty," Supercomputing '91:Proceedings of the 1991 ACM/IEEE Conference on Supercomputing, Albuquerque, NM, USA, 1991

MemBlock Feature 3: Increase effective TLB capacity

- 16 fully associative entry + 64 direct mapping entry → **48 fully associative entry**
- Support **TLB compression**, merging contiguous page table entries



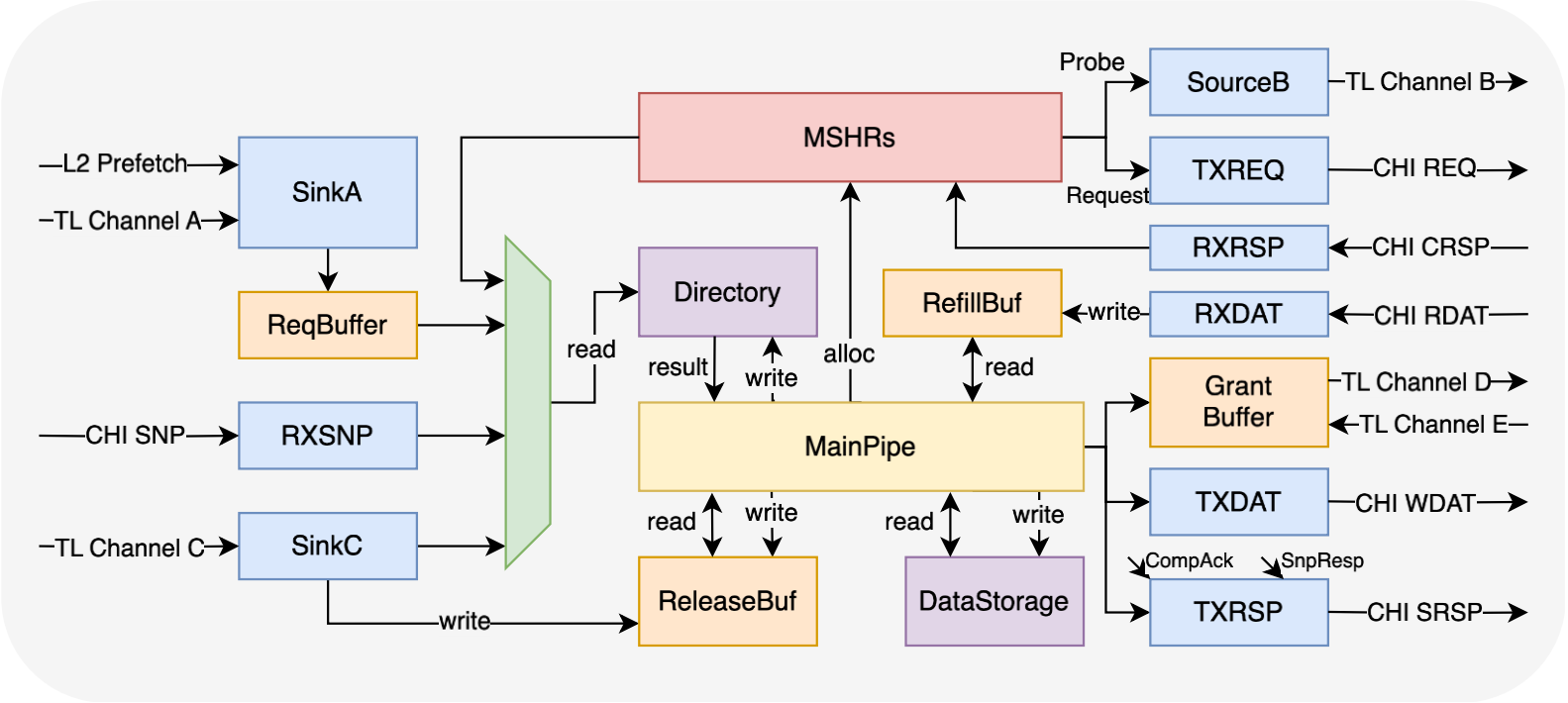
DTLB organization in Nanhu / Kunminghu



TLB compression

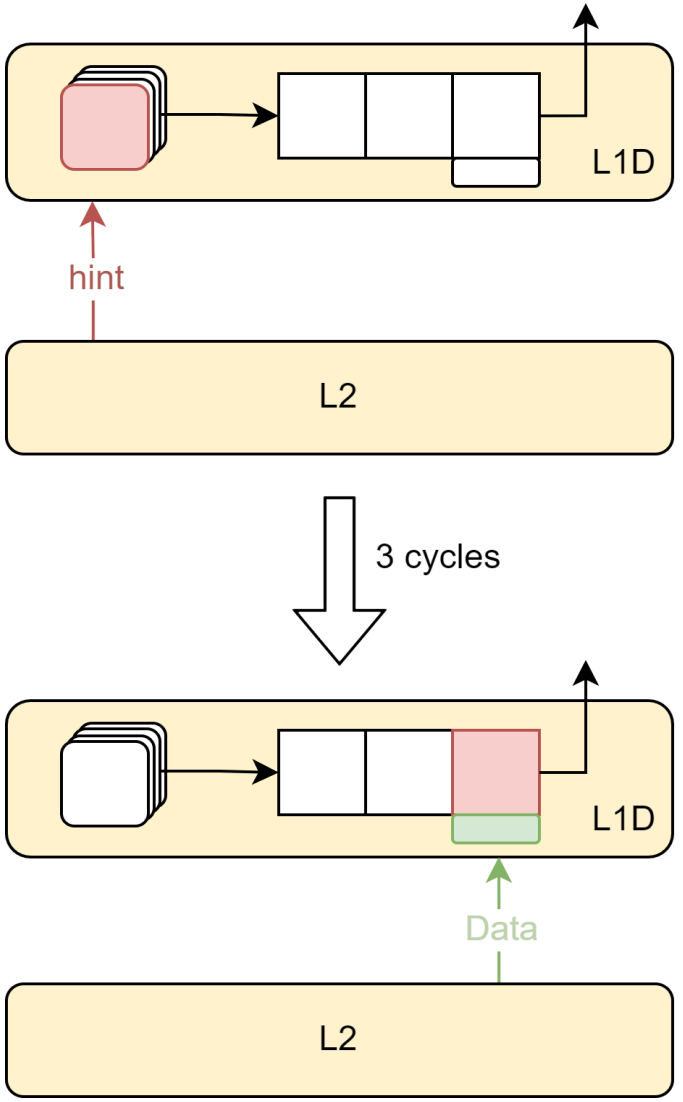
L2 Cache Upgrade Overview

- Multiple concurrent pipelines → Non-blocking main pipeline
- L1-L2 Refill wake-up collaboratively
- Eliminate transaction serialization in the same set & Request merge
- Supported protocols
  - CHI Issue E.b
  - CHI Issue B
  - CHI Issue C
  - TileLink



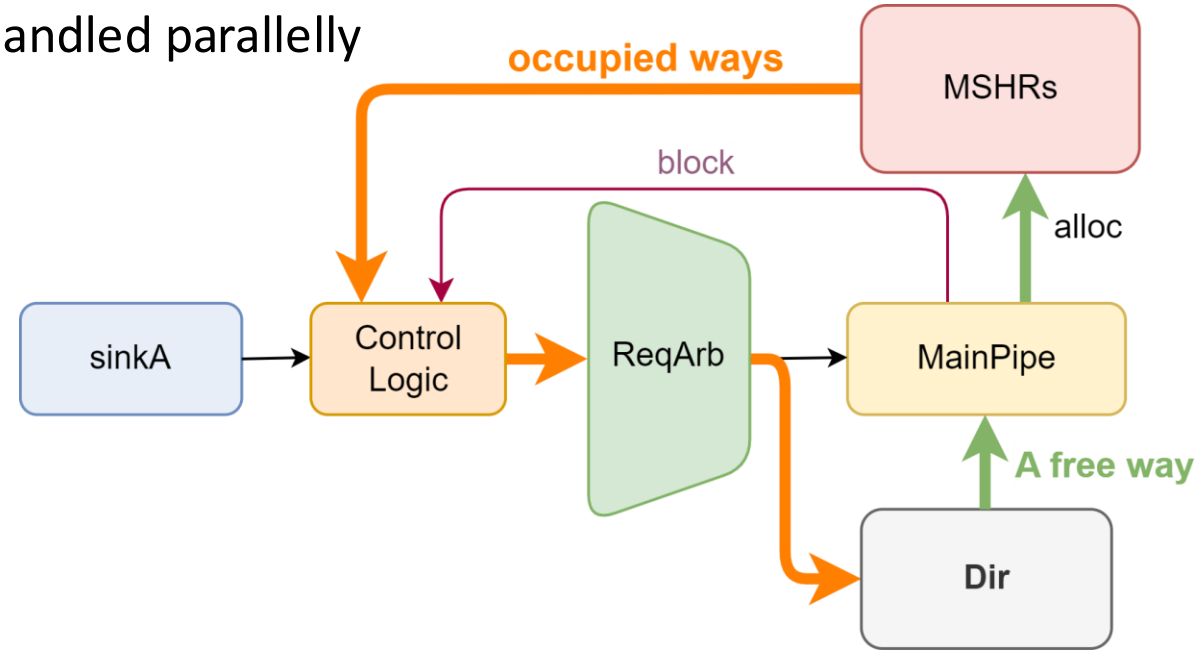
L2 Cache Feature 1: L1-L2 Collaborative Optimization

- Observation：
  - Interval between **load wake-up** and **use-refill-data** is 3 cycles
  - There’s chance to hint LSU to replay before refill
- New Design：
  - Let L2 give **Hint signal in advance**
  - Speedup the wakeup and replay of load replay queue
- Implementation：
  - Set up monitor to track info from main pipeline
  - Need to calculate the exact Hint timing
  - Send hint signal to L1 3 cycles before refill



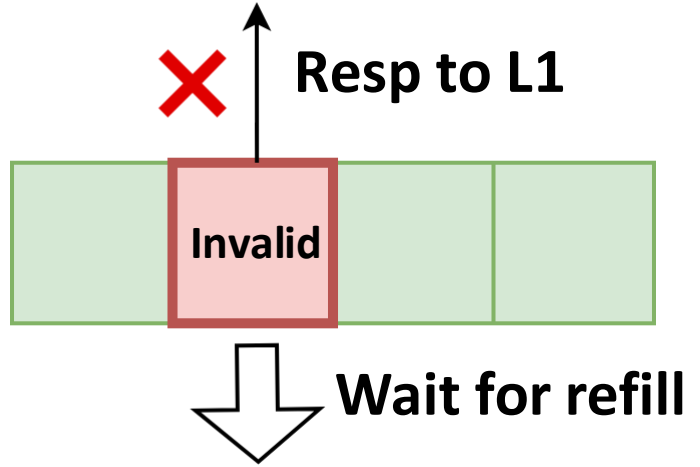
L2 Cache Feature 2: Eliminate Txns Serialization in same set

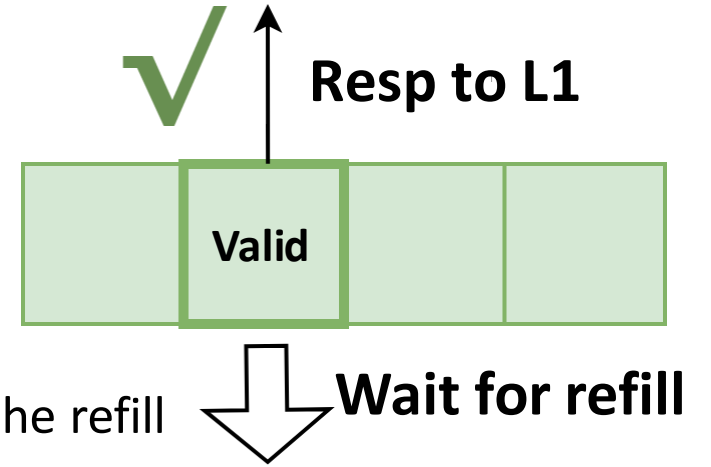
- Background: Txns to **same sets** were handled serially, reduces parallelism
- Difficulty: Same-set txns can affect each other, due to replacement
- New Design
  - Txns of **same-set but different-addr** can be handled parallelly
  - Txns of same-addr are handled serially
- Implementation
  - Record occupied ways in active MSHRs
  - **Assign a free way** to the new request



L2 Cache Feature 3: Evict on Refill

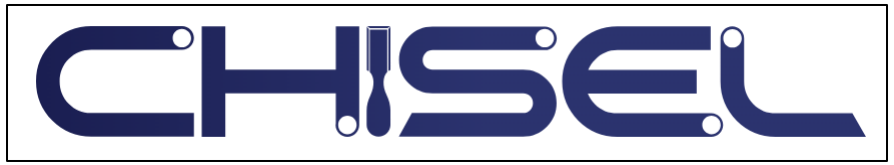
- **Observation：**
  - On L1 acquire, L2 chooses a way for replacement
  - But this way is **occupied until refill,** unable to serve L1
- **Improvement：**
  - New design chooses replaced-way **after** data is refilled from L3
  - So, it can still serve L1 when waiting for refill
- **Implementation：**
  - Transfer it to L3 on acquire miss, make replacer untouched
  - Wait for L3 to refill
  - **Read directory again** and assign one way when MSHR handles the refill





Beyond RTL Design: Agile development & infrastructure

- **Implement by Chisel HDL**
  - High readability
  - High modifiability



Learn more from https://www.chisel-lang.org/

- we designed some open-source high-performance utilities / components

BinaryArbiterNode.scala	ExtractVerilogModules.scala	LookupTree.scala	RegMap.scala
BitUtils.scala	FastArbiter.scala	MIMOQueue.scala	Replacement.scala
ChiselDB.scala	FileRegisters.scala	Misc.scala	ResetGen.scala
CircularQueuePtr.scala	GTimer.scala	ParallelMux.scala	SRAMTemplate.scala
Constantin.scala	Hold.scala	Pipeline.scala	StopWatch.scala
DataModuleTemplate.scala	IntBuffer.scala	PipelineConnect.scala	TLClientsMerger.scala
ECC.scala	LFSR64.scala	PriorityMuxDefault.scala	TLEdgeBuffer.scala
ExcitingUtils.scala	LatencyPipe.scala	PriorityMuxGen.scala	

Check it out! https://github.com/OpenXiangShan/Utility

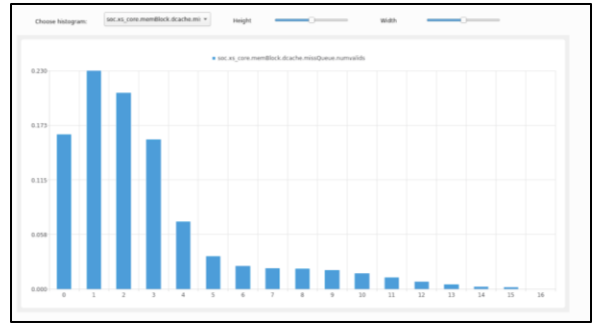
Beyond RTL Design: Agile development & infrastructure

- **Highly configurable**
  - setup parameters as you wish

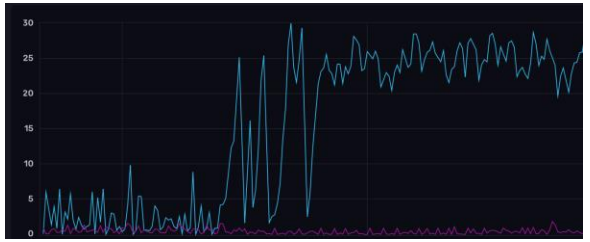
```
IBufSize: Int = 48,
DecodeWidth: Int = 6,
RenameWidth: Int = 6,
CommitWidth: Int = 6,
FtqSize: Int = 64,
EnableLoadFastWakeUp: Boolean = true,
IssQueSize: Int = 16,
NRPhyRegs: Int = 192,
LoadQueueSize: Int = 80,
LoadQueueNWriteBanks: Int = 8,
StoreQueueSize: Int = 64,
StoreQueueNWriteBanks: Int = 8,
RobSize: Int = 256,
dpParams: DispatchParameters =
  DispatchParameters(
    IntDqSize = 16,
    FpDqSize = 16,
    LsDqSize = 16,
    IntDqDeqWidth = 4,
    FpDqDeqWidth = 4,
    LsDqDeqWidth = 4
  ),
exuParameters: ExuParameters =
  ExuParameters(
    JmpCnt = 1,
    AluCnt = 4,
    MulCnt = 0,
    MduCnt = 2,
    FmacCnt = 4,
    FmiscCnt = 2,
    FmiscDivSqrtCnt = 0,
    LduCnt = 2,
    StuCnt = 2
  ),
prefetcher: Option[PrefetcherParams] =
  Some(SMSParams()),
LoadPipelineWidth: Int = 2,
StorePipelineWidth: Int = 2,
StoreBufferSize: Int = 16,
StoreBufferThreshold: Int = 7,
EnableLoadToLoadForward: Boolean = true,
EnableFastForward: Boolean = false,
EnableLdVioCheckAfterReset: Boolean = true,
EnableSoftPrefetchAfterReset: Boolean = true,
EnableCacheErrorAfterReset: Boolean = true,
EnablePTWPreferCache: Boolean = true,
EnableAccurateLoadError: Boolean = true,
```

Some key parameters

- **Easy-to-use performance counter**
- Multiple collection types
  - Cumulative counter
  - Delay counter
  - Transaction counter
  - Queue counter



- Multiple analysis style
  - Key-value list
  - Histogram
  - Database
  - Time series visualization



XiangShan: Open-Source High-Performance Processor

XiangShan Project

- **Embraces innovative agile development workflow**
- **Fills the gap in open-source high-performance processors**
- **Addresses the needs from both industry and academia**

- 1st generation: YQH (Yanqihu) & 2nd generation: NH (Nanhu)
- **3rd generation: KMH (Kunminghu)**
  - **Support RVA23 profile and L2 Cache with CHI protocol**
  - **3GHz @7nm, estimated SPEC CPU2006 45@3GHz**
- **Open-sourced at https://github.com/OpenXiangShan/XiangShan**
- **NEXT PART: Hands-on Development**
  - Scan the QR to get the env introduction and slides

Thanks!

This tutorial is organized by Zeyu Gao, Haoyuan Feng, YueXiao Ma, Yanqin Li.