中国科学院计算技术研究所 INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

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

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

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

The XiangShan Team

Institute of Computing Technology (ICT)

Chinese Academy of Sciences (CAS)

HPCA'24@Edinburgh, Scotland

March 2, 2024

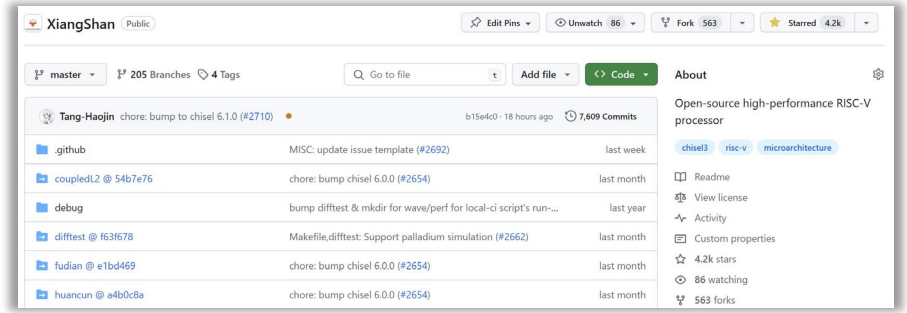
Tutorial@HPCA'24 Schedule

Time (AM)	Topic
8:30 - 9:00	Introduction of the XiangShan Project
9:05 - 10:00	***Microarchitecture Design and Implementation***
10:05 - 10:50	Hands-on Development
	Coffee Break
11:20 - 12:00	Hands-on Development & Discussions (Cont.)

XiangShan: Open-Source High-Performance Processor

- **1st generation: YQH**
  - 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**
  - 2021/5: starting design exploration and RTL design
  - 2023/4: GDSII delivery
  - **14-nm tape-out, estimated SPEC CPU2006 20@2GHz**
- **3rd generation: KMH**
  - ISA feature: **Vector (V) extension, Hypervisor (H) extension**
  - Comprehensive performance improvement

Fragrant Hills in Beijing

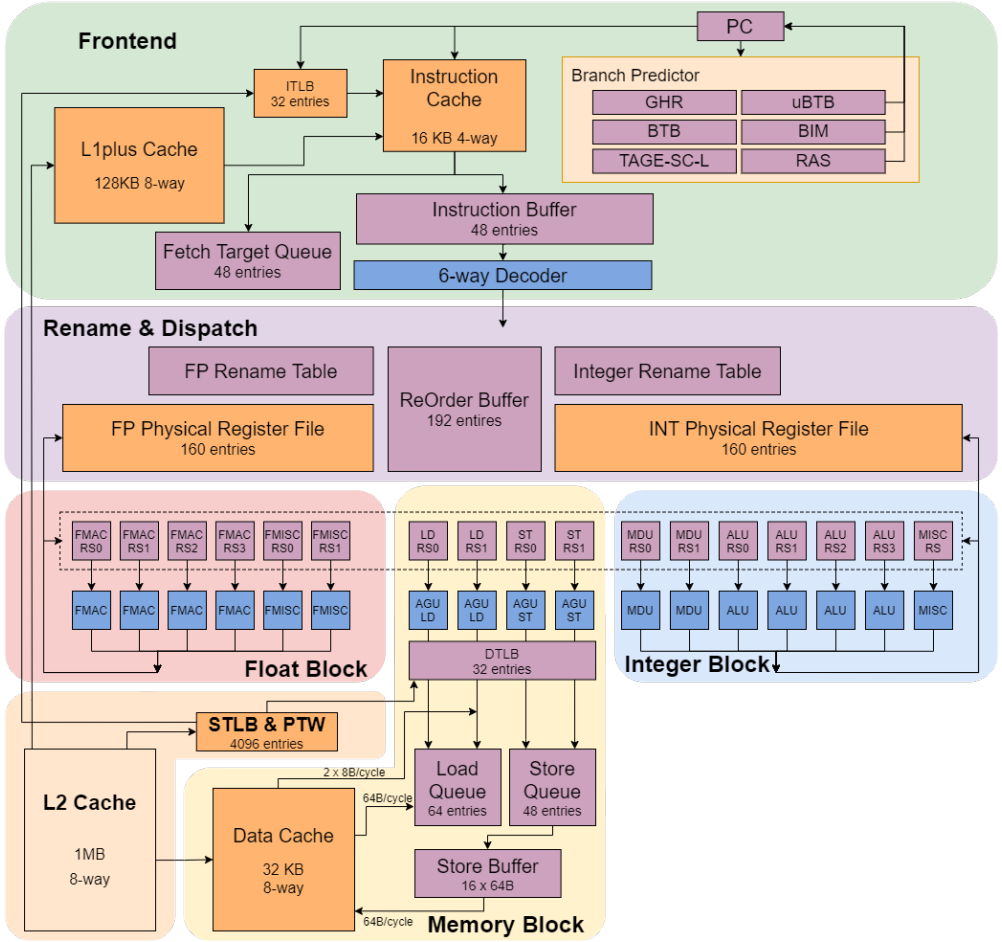


>4.2K stars, >500 forks on GitHub

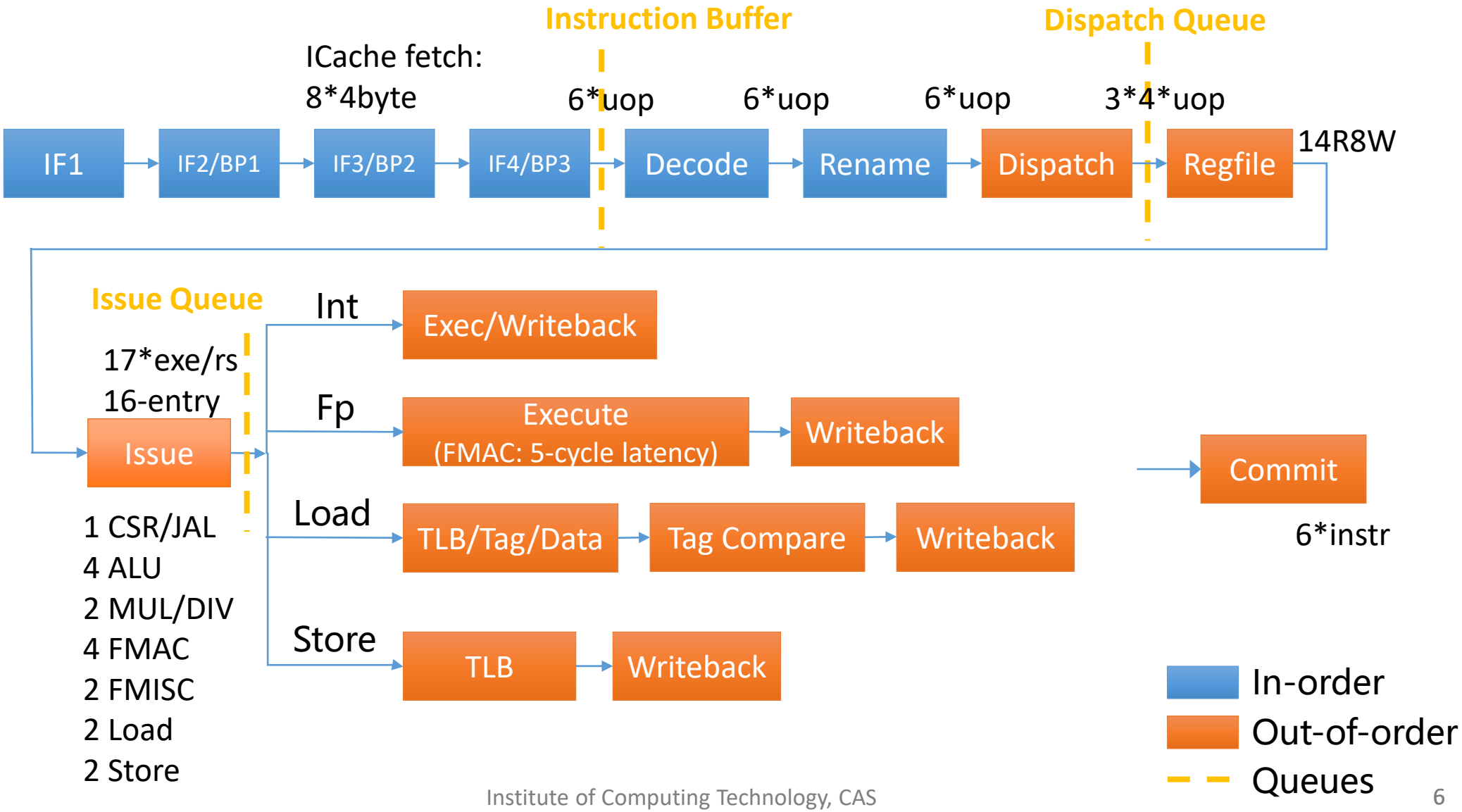
- Open-sourced at https://github.com/OpenXiangShan/XiangShan

XiangShan Microarchitecture (V1-YQH)

- **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

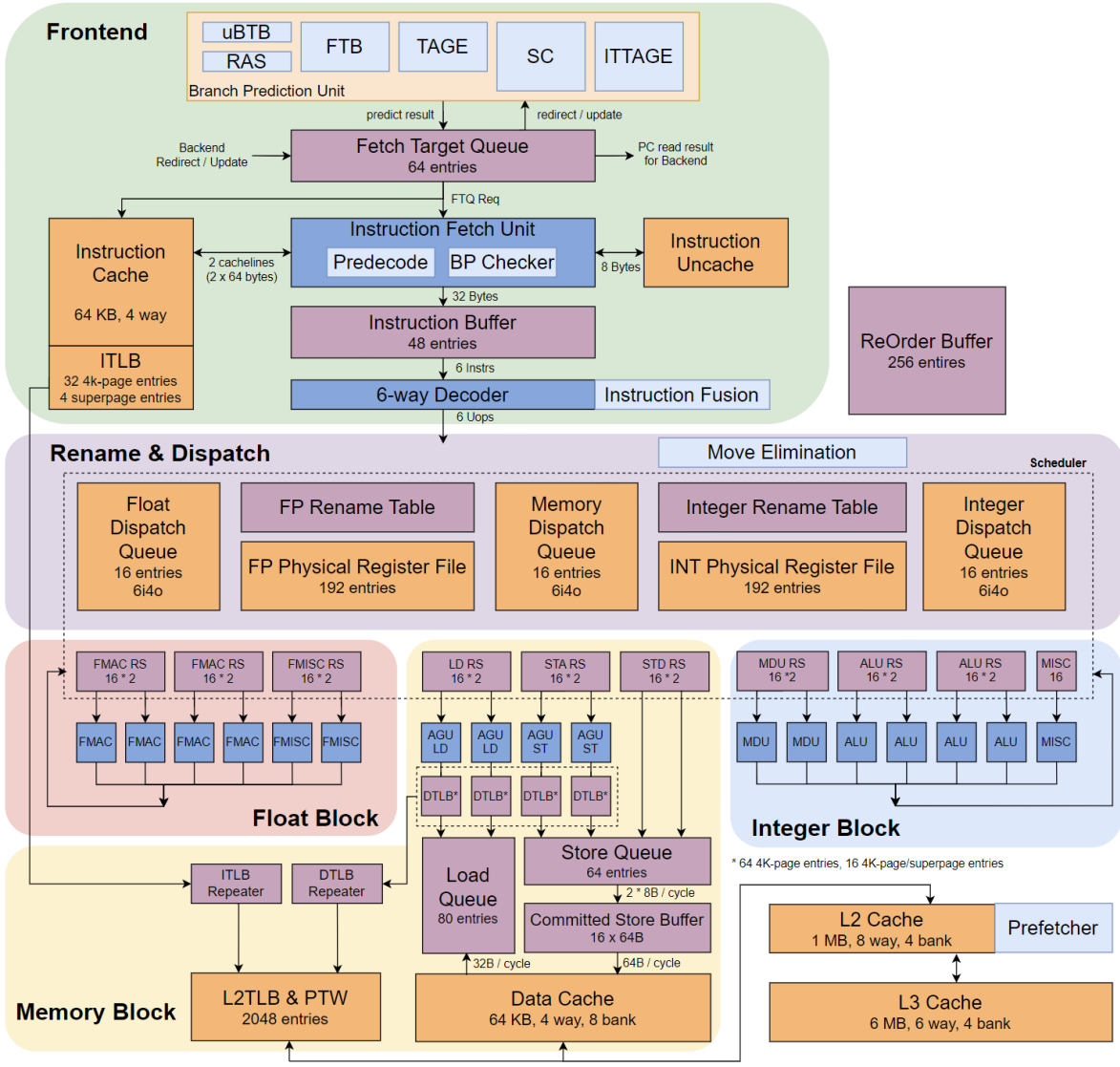


XiangShan Microarchitecture (V1-YQH)



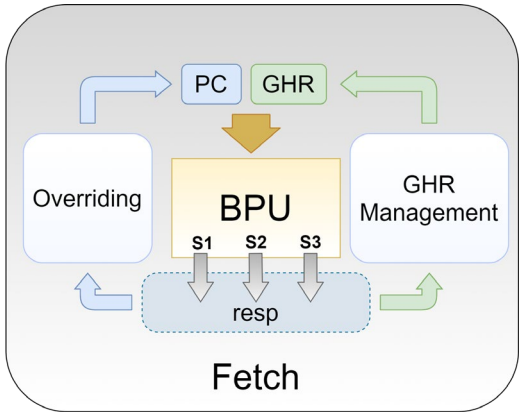
XiangShan Microarchitecture (V2-NH)

- **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
- **80**-entry DTLB, **36**-entry ITLB, **2K**-entry STLB
- **1MB** non-inclusive L2 Cache
- **6MB** non-inclusive L3 Cache (LLC)



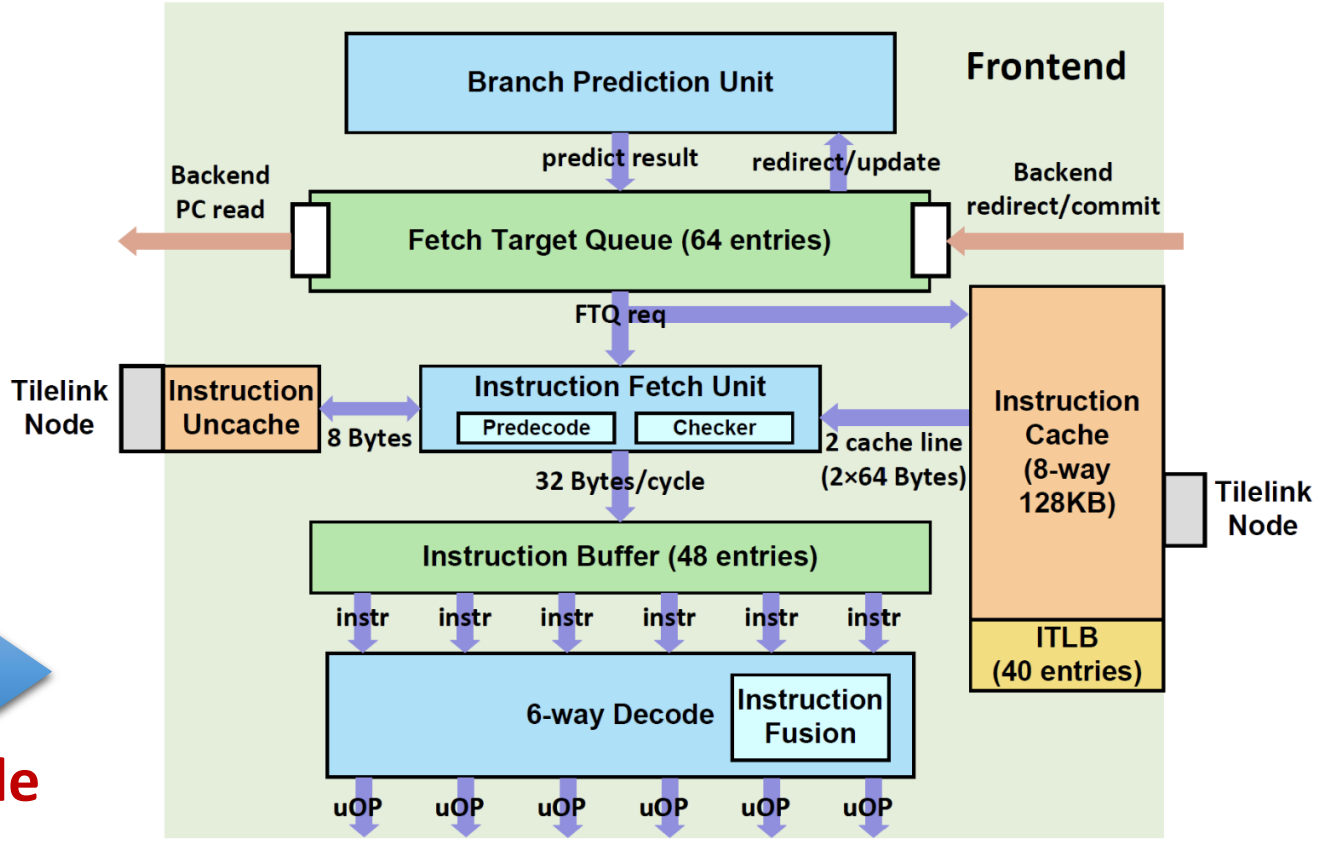
Frontend

- Decoupled frontend
  - **Producer**: branch predictor unit
  - **Consumer**: instruction fetch unit
- Merit
  - Fewer fetch bubbles
  - Fetch directed instr. prefetching



Yanqihu

Upgrade



Nanhu

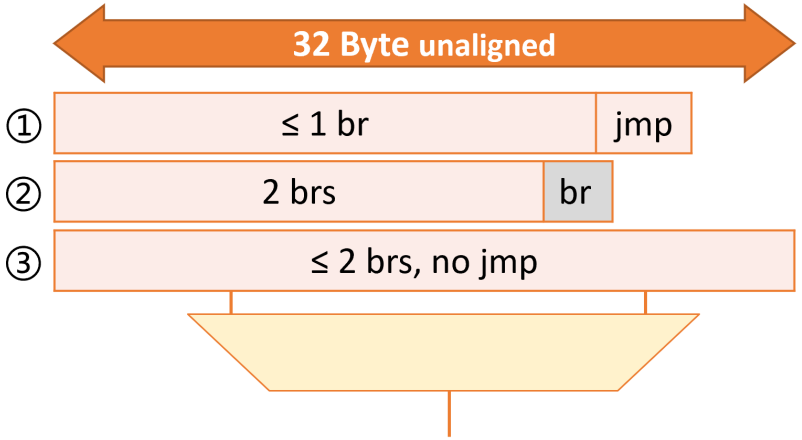
* Frontend design of YQH is partially learnt from BOOM

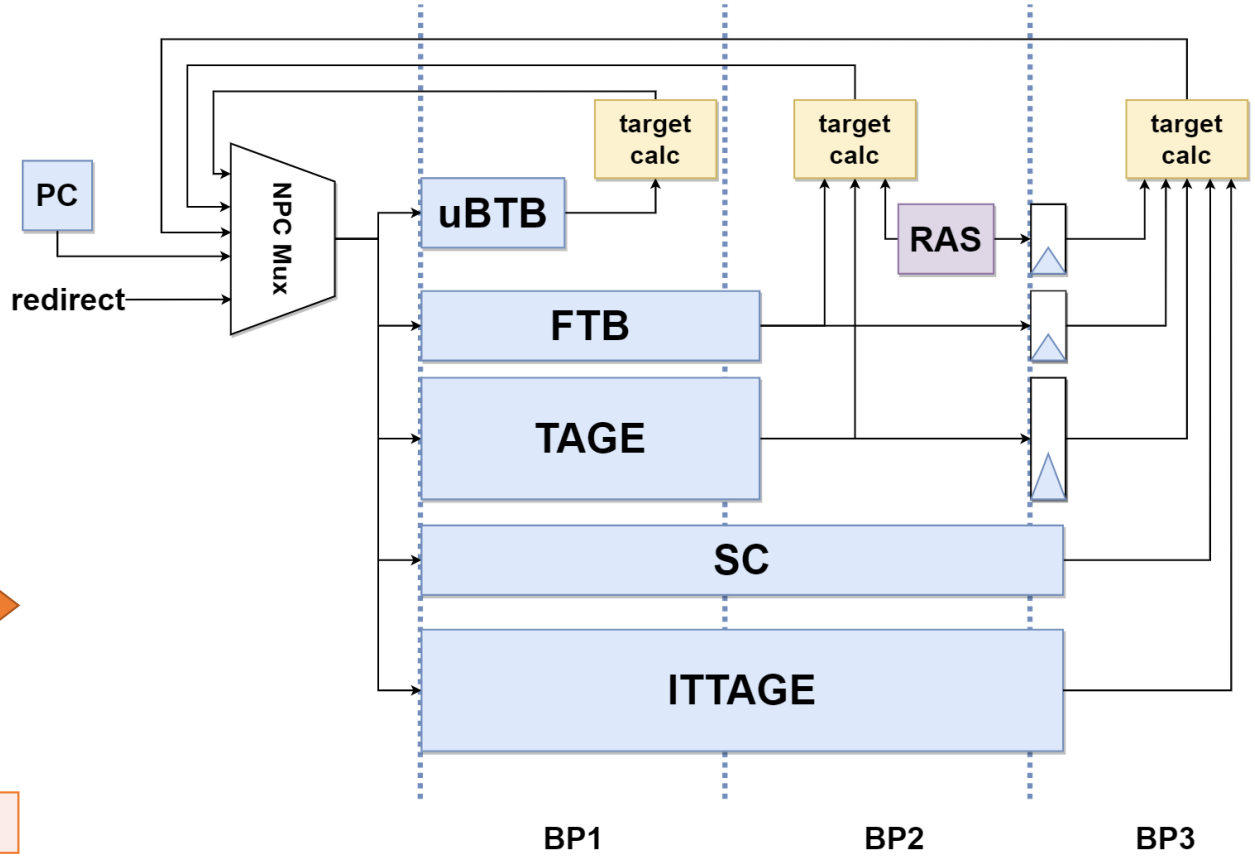
Branch Predictor

Three-stage overriding prediction

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

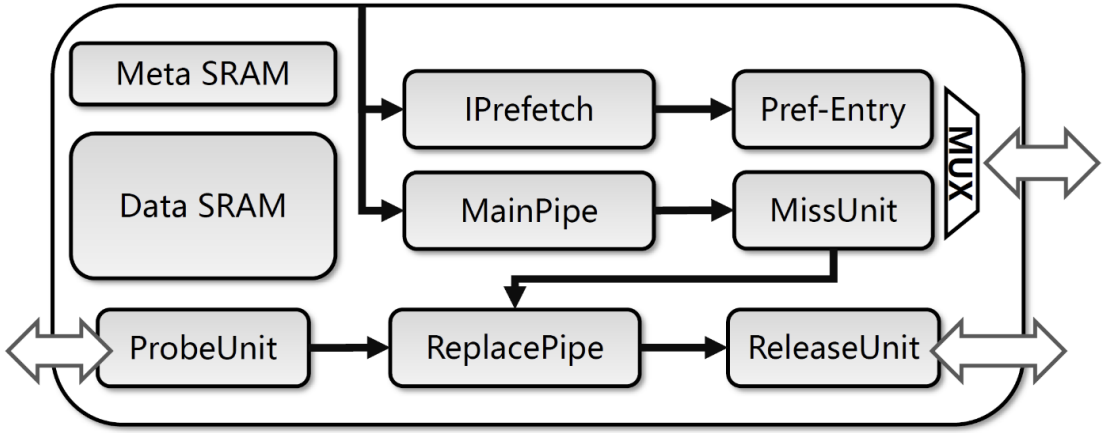
- FTB: Fetch Target Buffer





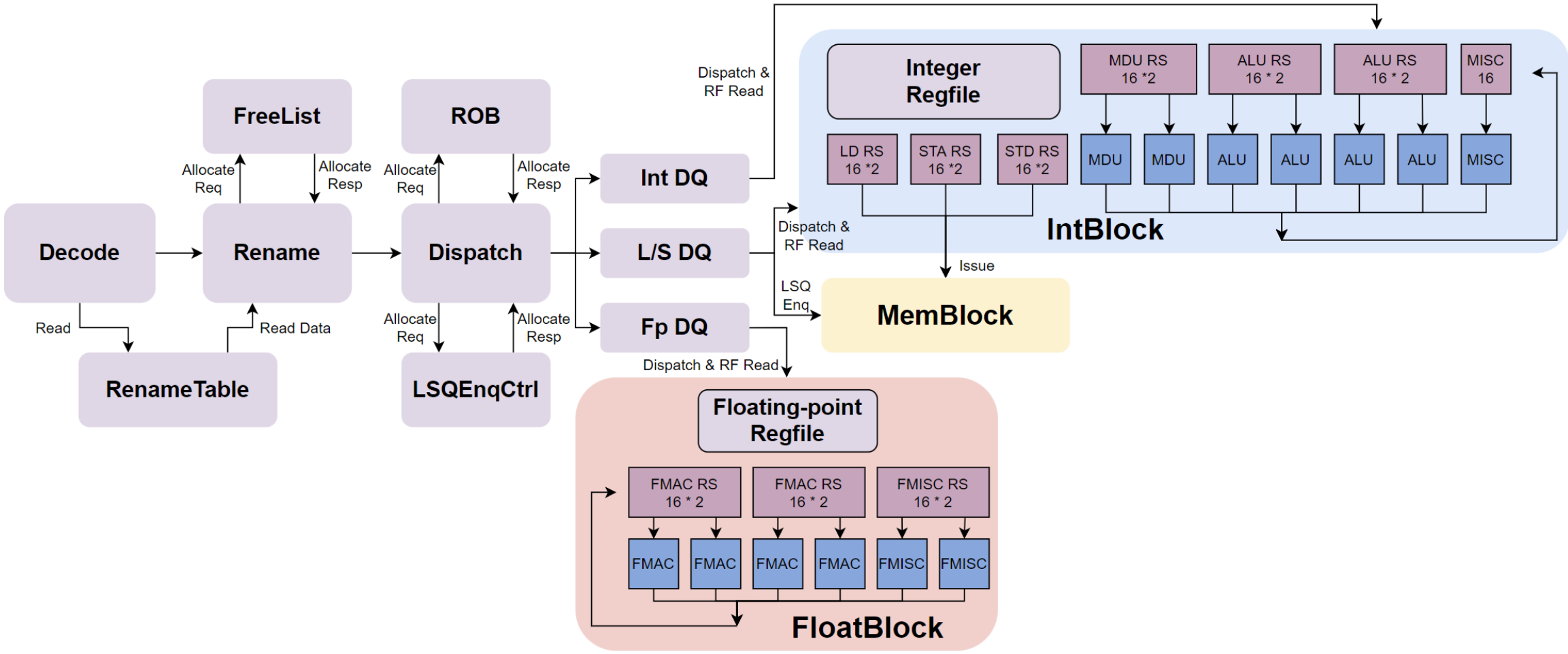
Instruction Cache

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



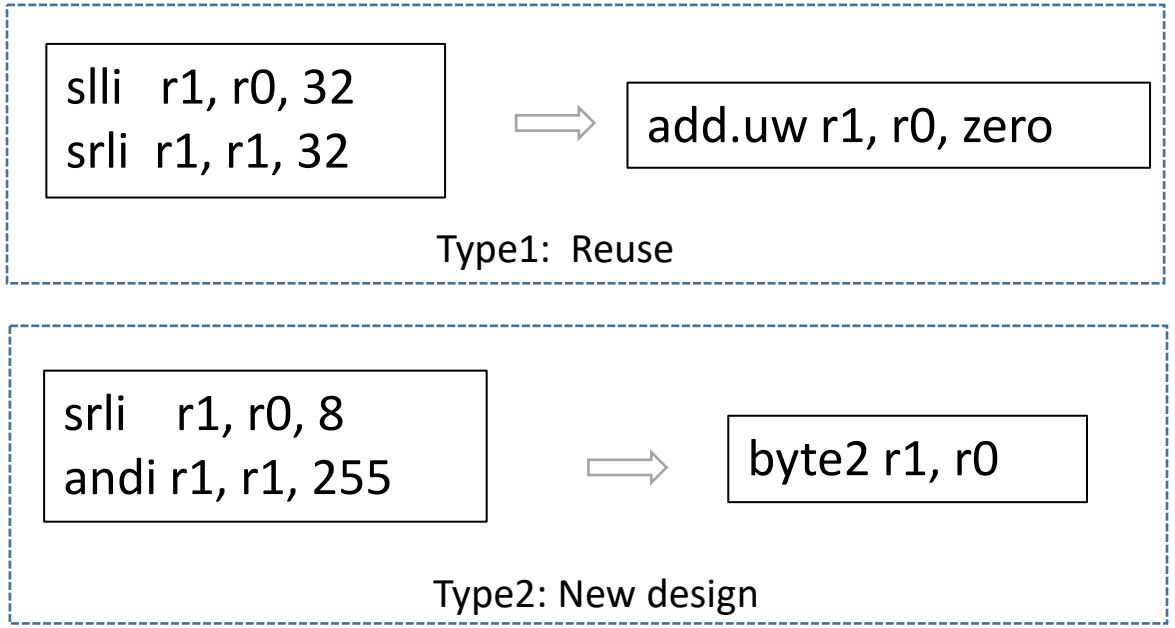
	Yanqihu	Nanhu	
L1 ICache	16KB, 4-w	64KB, 4-w	4×
L1plus Cache	128KB, 8-w	-	Reduce
Read bandwidth	64 B	128 B	2×
TileLink Support	No	Yes	New
Instruction prefetch	Stream	L2 FDIP	New

Backend


FreeList
ROB
Allocate Req
Allocate Resp
Allocate Req
Allocate Resp
Decode
Rename
Dispatch
Int DQ
L/S DQ
Fp DQ
Read
Read Data
Allocate Req
Allocate Resp
RenameTable
LSQEnqCtrl
Dispatch & RF Read
Integer Regfile
MDU RS 16 *2
ALU RS 16 * 2
ALU RS 16 * 2
MISC 16
LD RS 16 *2
STA RS 16 *2
STD RS 16 *2
MDU
MDU
ALU
ALU
ALU
ALU
MISC
IntBlock
Dispatch & RF Read
Issue
LSQ Enq
MemBlock
Dispatch & RF Read
Floating-point Regfile
FMAC RS 16 * 2
FMAC RS 16 * 2
FMISC RS 16 * 2
FMAC
FMAC
FMAC
FMAC
FMISC
FMISC
FloatBlock

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 instr. greatly**



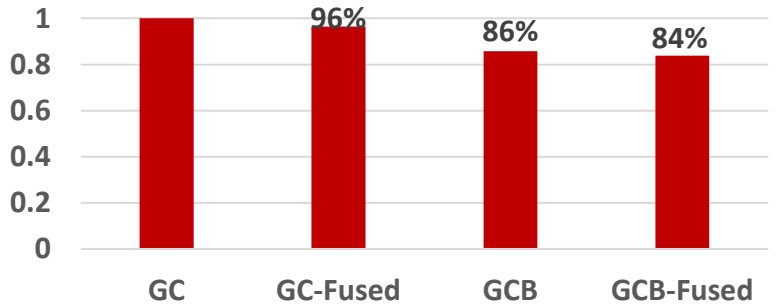
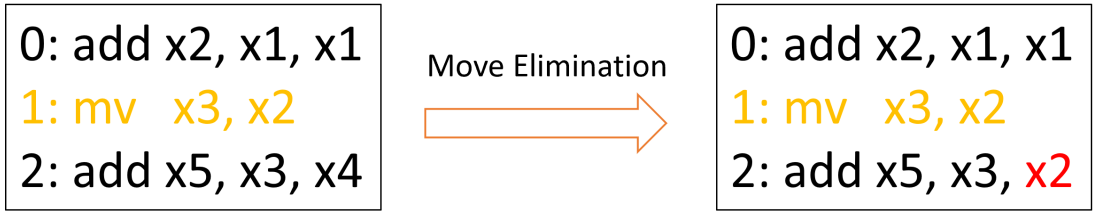
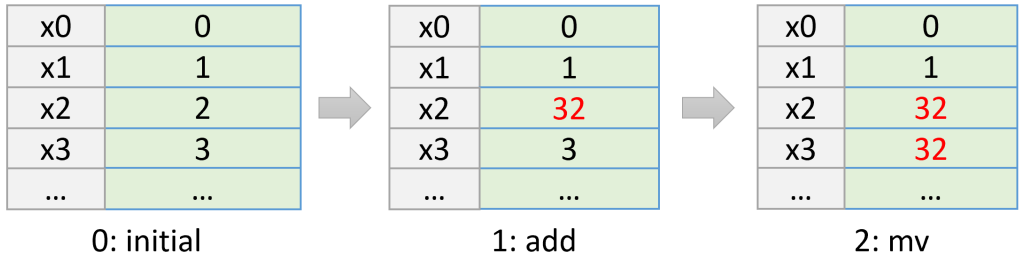


Figure. Normalized Dynamic Instruction Count for CoreMark (-O2)

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



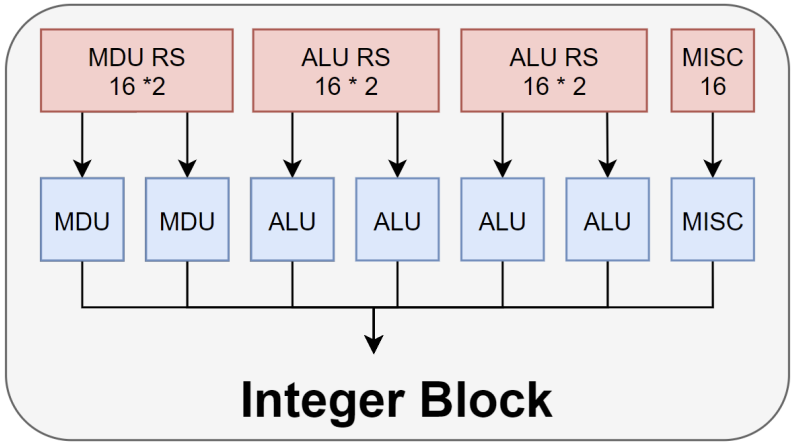


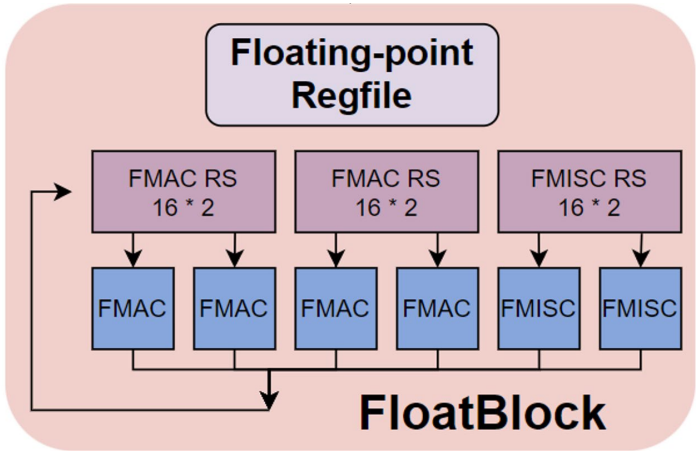
Reservation Station

- Main Changes:
  - 2i2o & 1i1o
  - Age Matrix based selection

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

Age Matrix

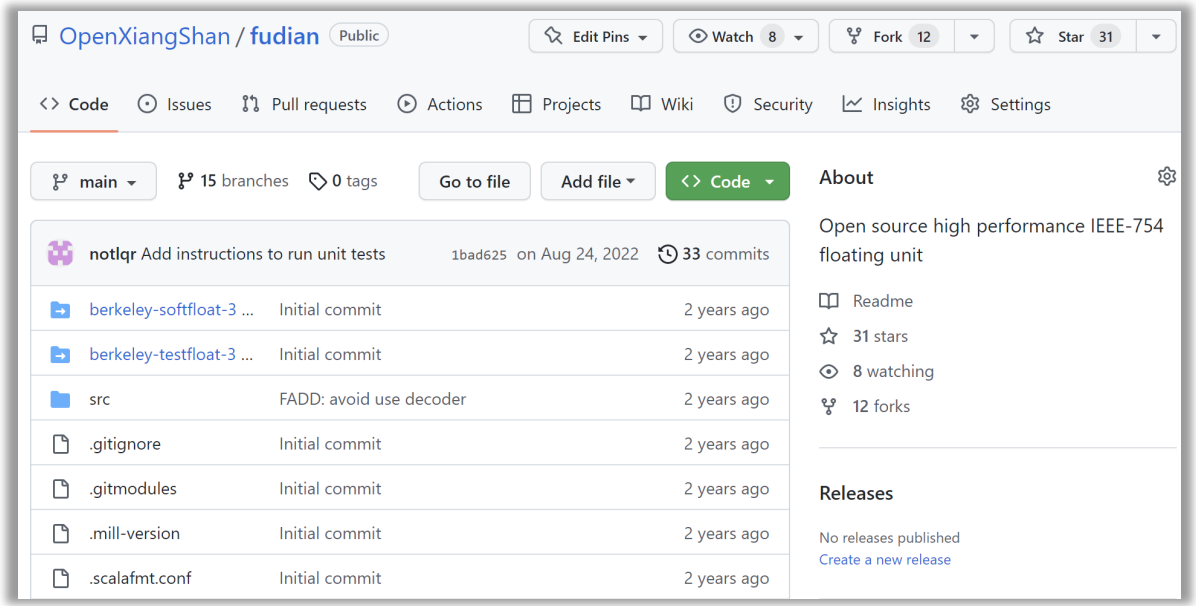




Floating Point Units (FPUs)

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

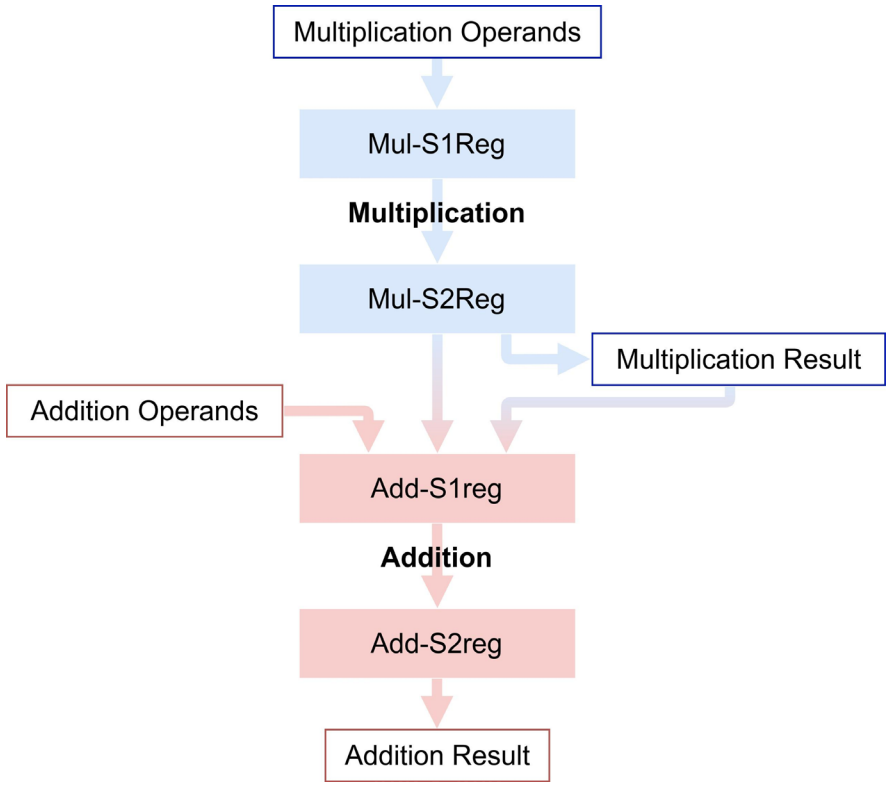
Operations	Latency
FADD	3
FMUL	3
FMA	5
FDIV	≤ 11 (float) ≤ 18 (double)
FSQRT	≤ 17 (float) ≤ 31 (double)
FCVT, FCMP	3



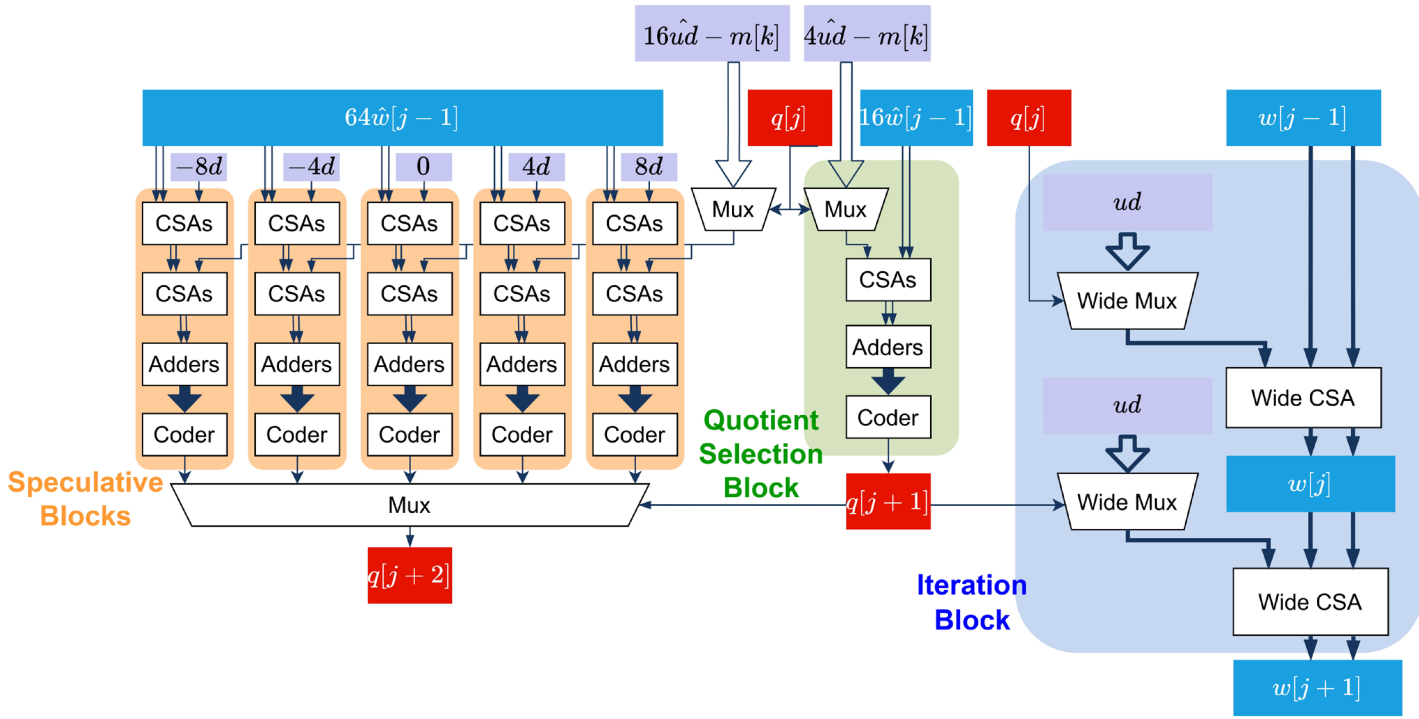
https://github.com/OpenXiangShan/fudian

FPU

- **Cascade FMA**
  - Reduce FADD delay 5->3



- **SRT16 Division units**
  - ½ delay compared with YQH (SRT4)



B/K Extension Implemented

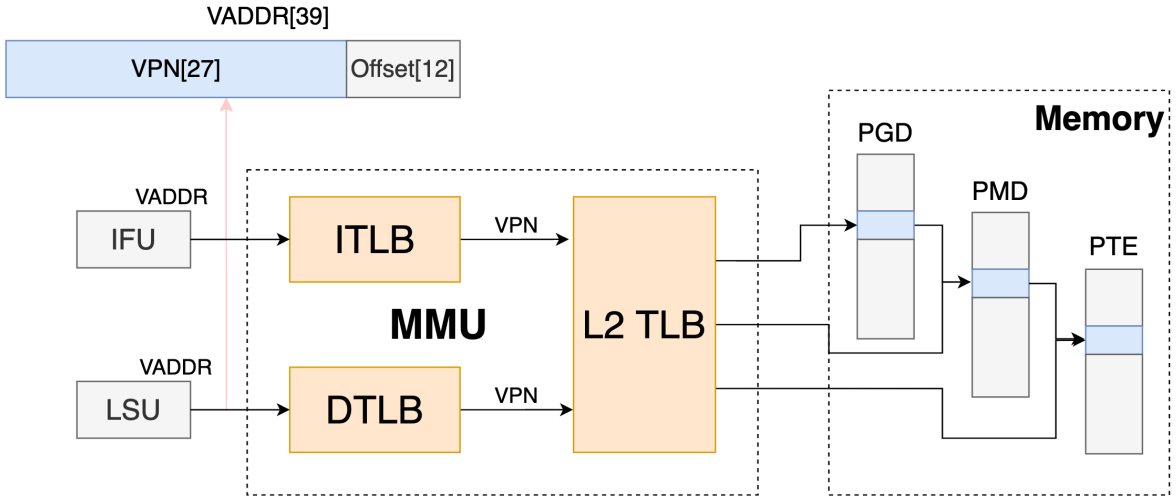
- **B**: Bit-Manipulation extension
- **K**: Cryptographic extension

B Extension

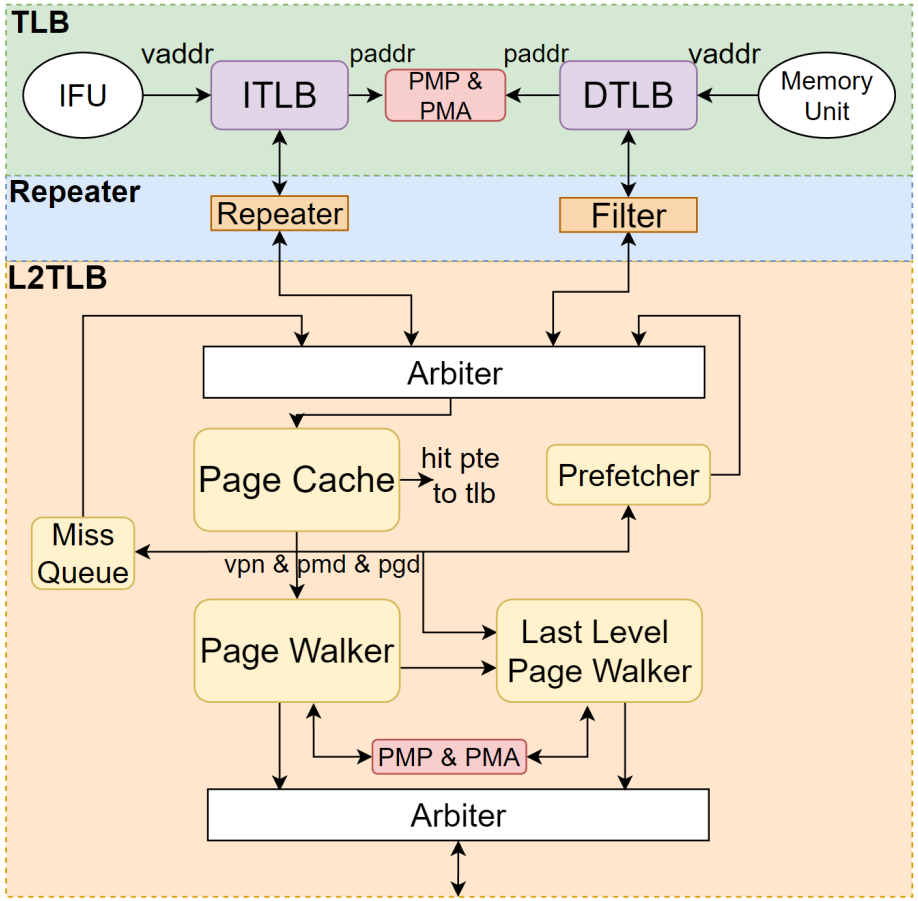
K Extension

Memory Management Unit

- Improvement:
  - Larger Data TLB size
  - Next-Line Prefetch for STLB
  - Parallel Page Table Walks
  - PMP/PMA support



Translation Flow when miss. From L1TLB to Memory

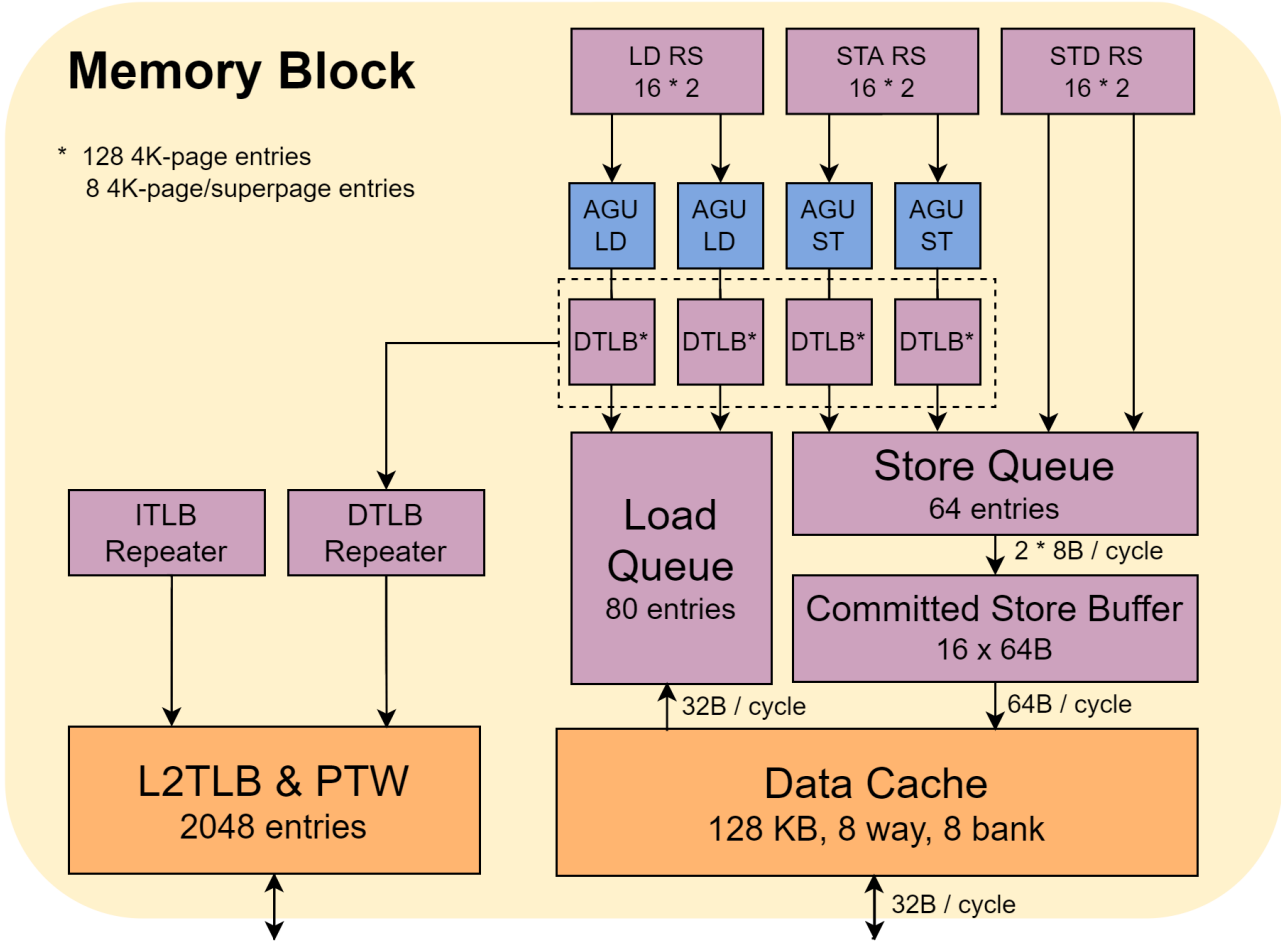


MMU

Memory Block

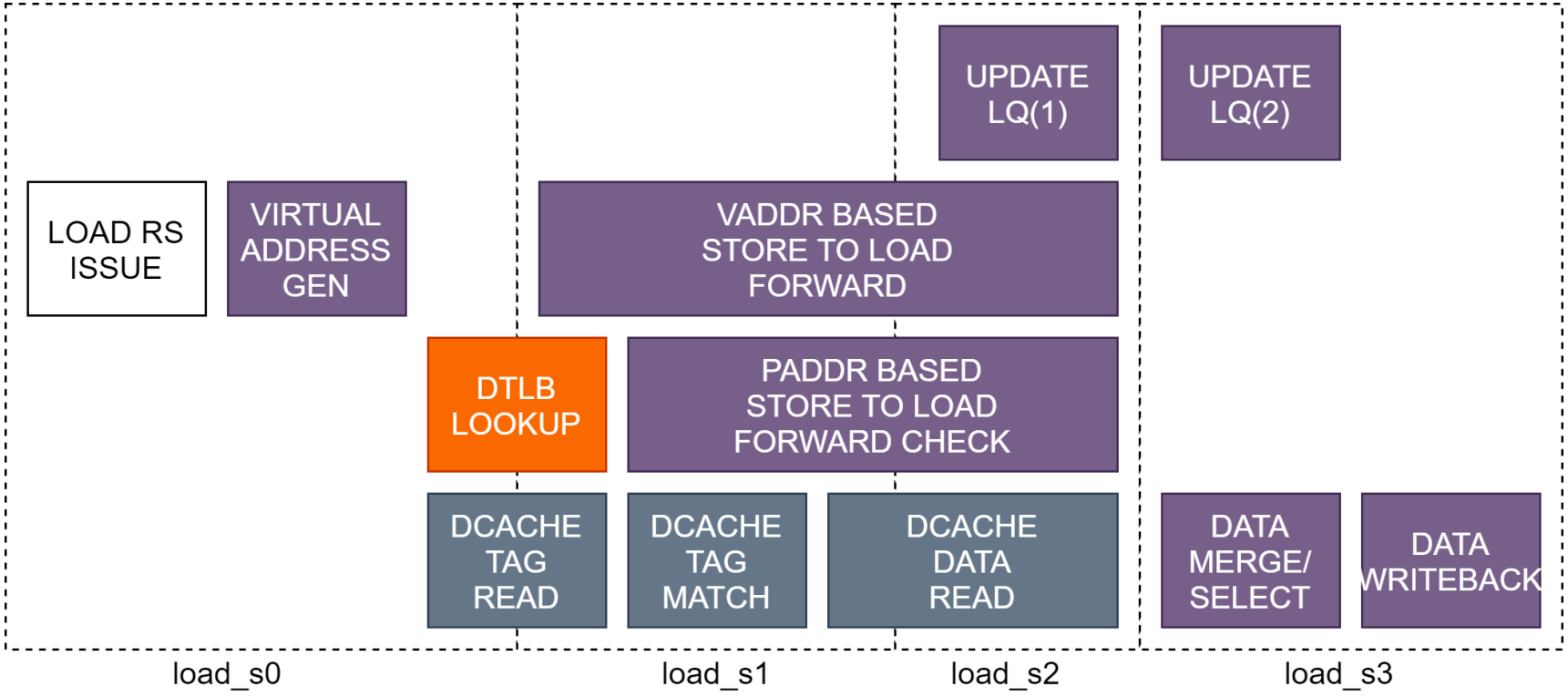
- Datasheets

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**



Load Pipeline

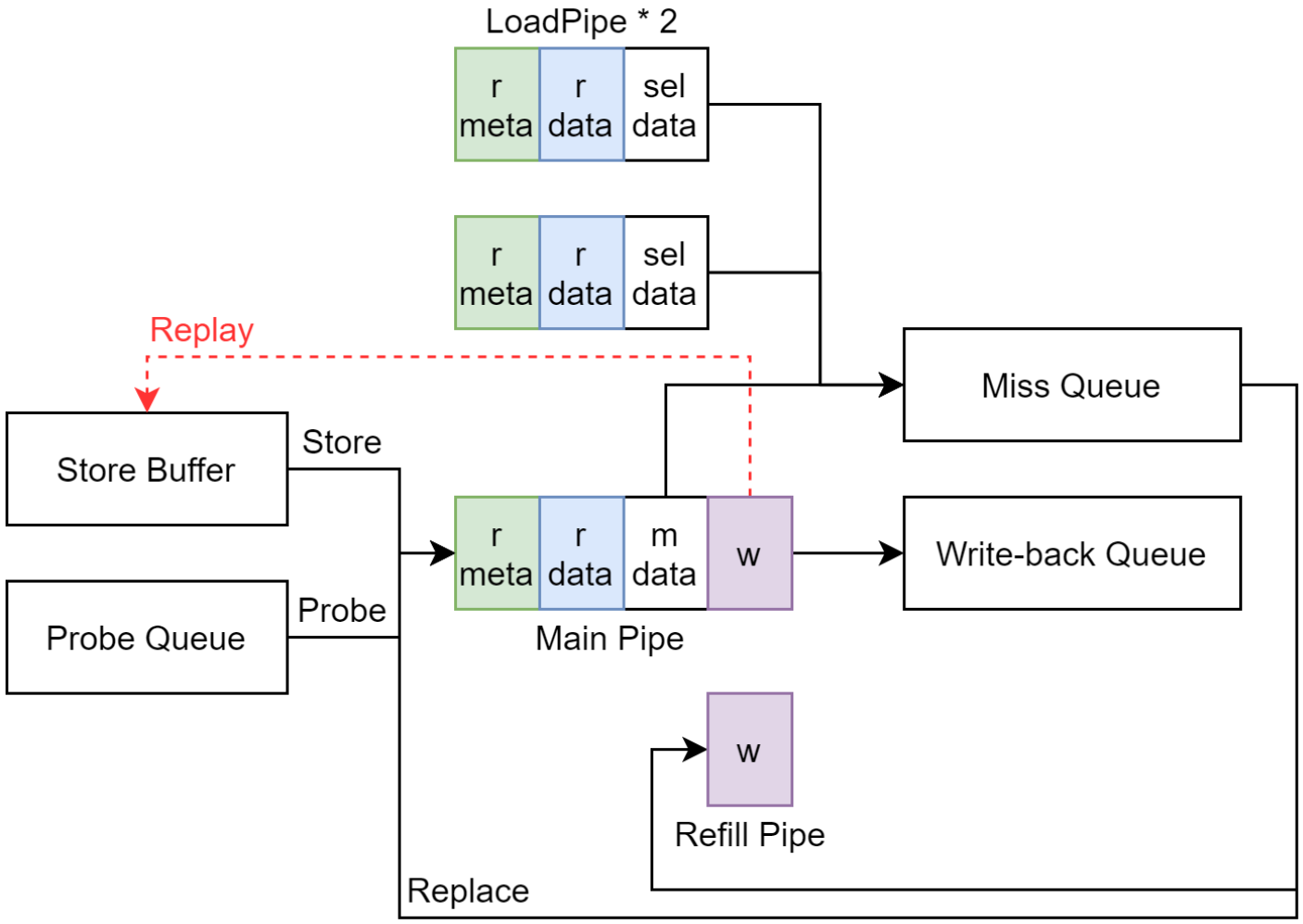
- Four stage
- Optimized for load-to-load & load-store bypass



L1 DCache

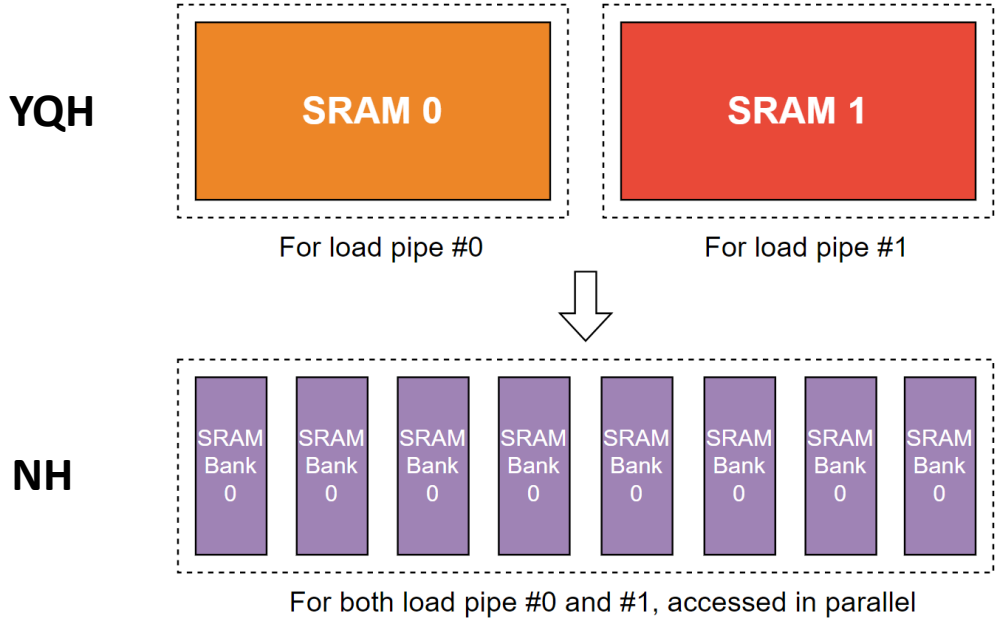
- Coupled with load pipeline
- TileLink protocol

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

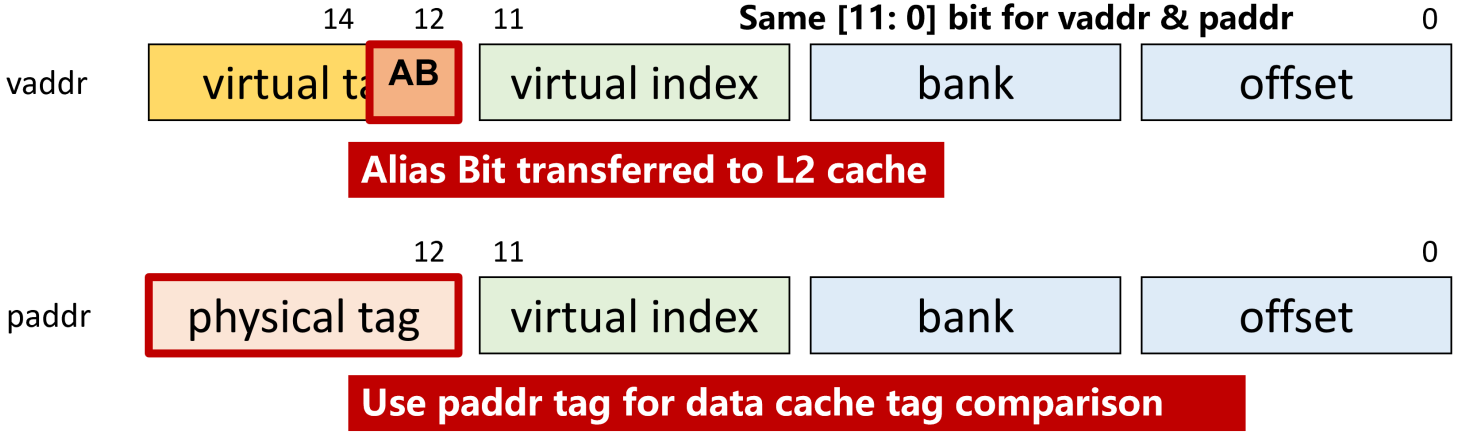


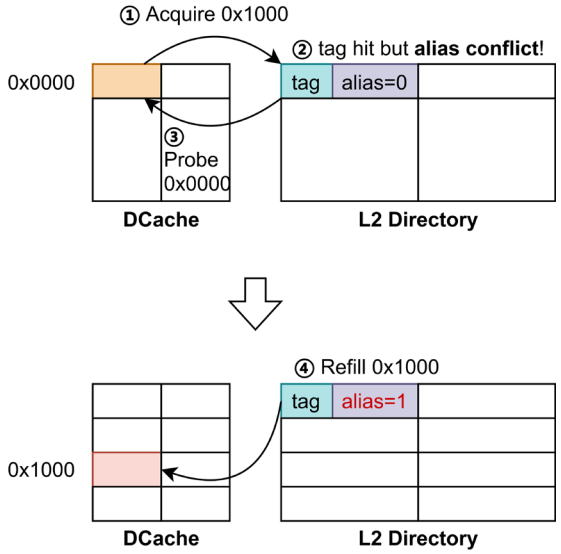
L1 DCache

- Noteworthy features
  - Multi-bank for multi read ports
  - Anti-alias for larger size
  - MSHR request merge
  - Hardware L1 data prefetcher



Multi-Bank

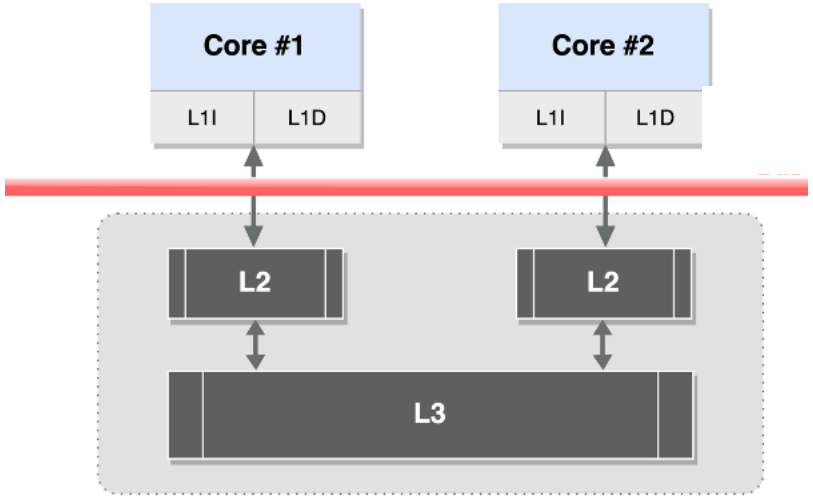




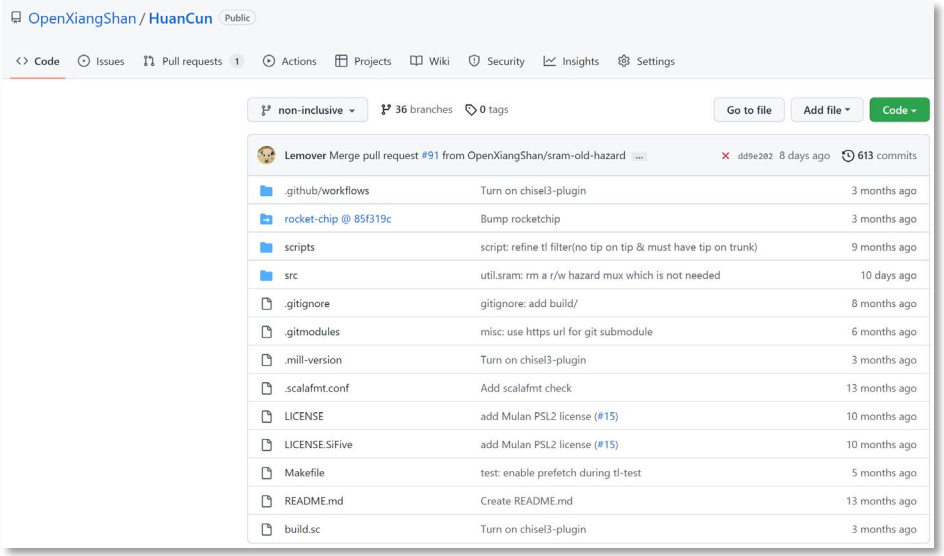
Anti-Alias processing flow

L2/L3 Cache

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



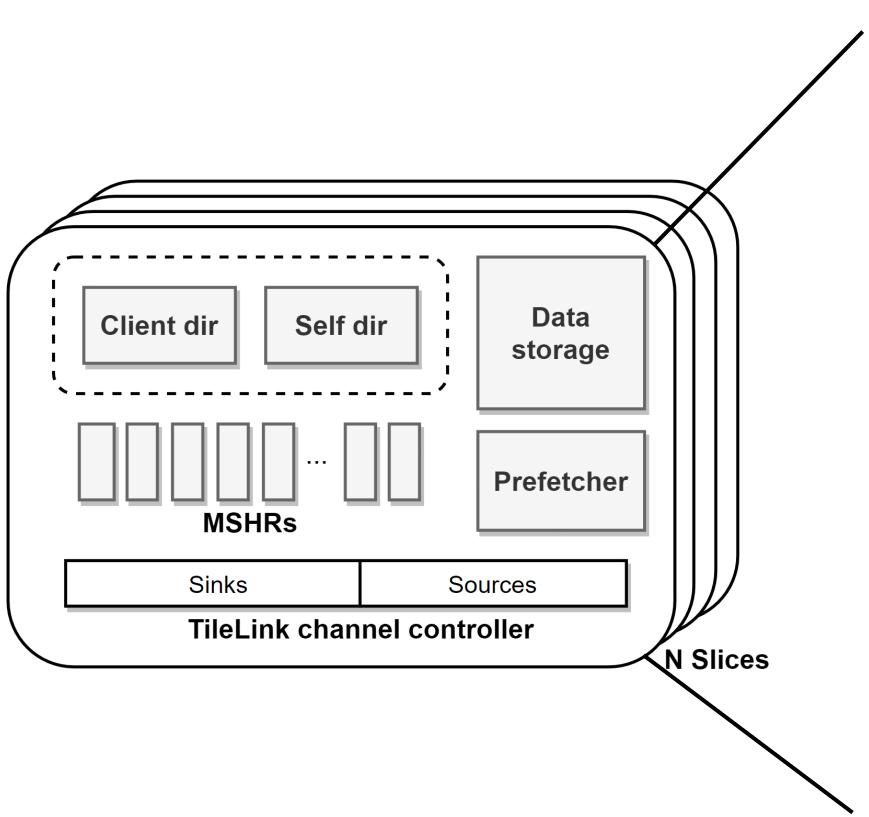
Nanhu Dual-core

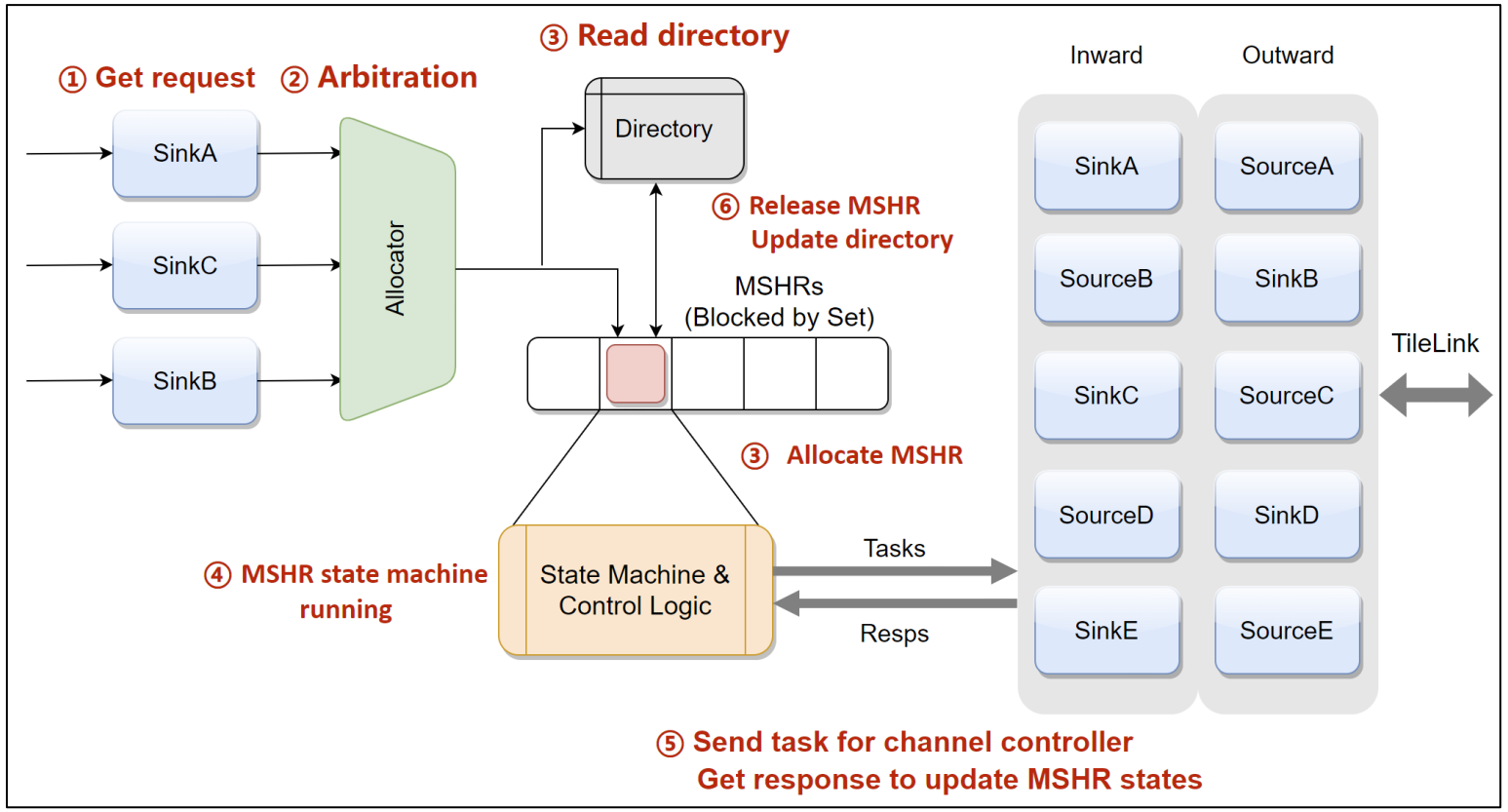


https://github.com/OpenXiangShan/HuanCun

L2/L3 Cache

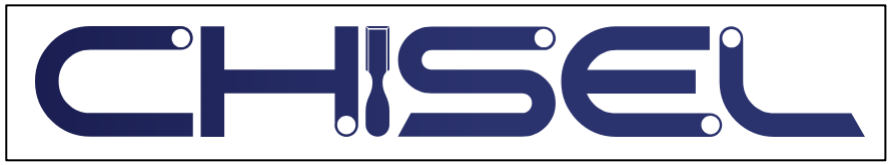
- Concurrent request processing flow





Microarchitecture Implementation

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



- Beyond built-ins, 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

Microarchitecture Implementation

- **Highly configurable**
  - You can 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,
```

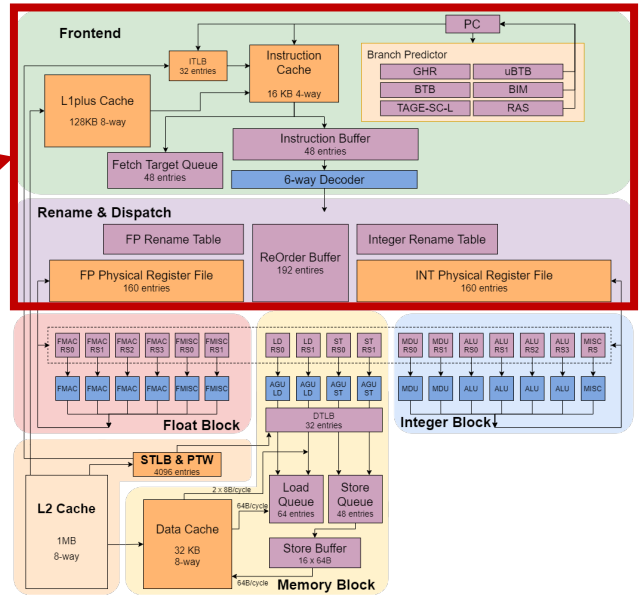
Some key parameters

e.g.
How to change rename/decode/issue width?

All you need is to change a number

```
23  // Synthesizable minimal XiangShan
24  // * It is still an out-of-order, super-scalaer arch
25  // * L1 cache included
26  // * L2 cache NOT included
27  // * L3 cache included
28  class MinimalConfig(n: Int = 1) extends Config(
29    new DefaultConfig(n).alter((site, here, up) => {
30      case SoCParamsKey => up(SoCParamsKey).copy(
31        cores = up(SoCParamsKey).cores.map(_.copy(
32          DecodeWidth = 2,
33          RenameWidth = 2,
34          FetchWidth = 4,
35          IssQueSize = 8,
36          NRPhyRegs = 80,
37          LoadQueueSize = 16,
38          StoreQueueSize = 16,
39          RoqSize = 32,
40          BrqSize = 8,
41          FtqSize = 16,
42          IBufSize = 16,
43          StoreBufferSize = 4,
44          StoreBufferThreshold = 3,
45          dpParams = DispatchParameters(
46            IntDqSize = 8,
47            FpDqSize = 8,
48            LsDqSize = 8,
```

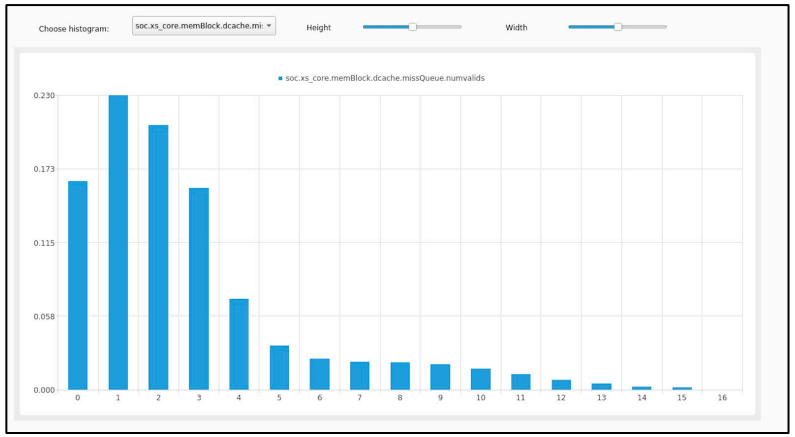
All affected
But it works

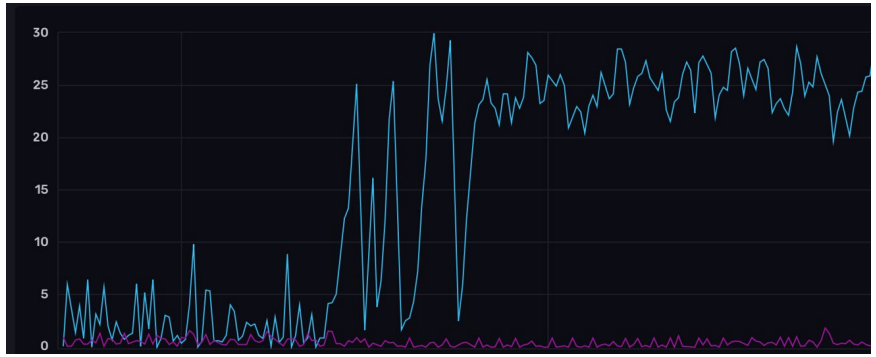


Microarchitecture Implementation

- **Easy-to-use performance counter system**
- Multiple collection types
  - Cumulative counter
  - Delay counter
  - Transaction counter
  - Queue counter
- Multiple analysis style
  - Key-value list
  - Histogram
  - Database
  - Time series visualization

```
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpBInstr,              57899
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpBRight,              54695
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpBWrong,               3204
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpCRight,               3615
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpCWrong,                 34
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpInstr,               78176
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpIRight,               5471
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpIWrong,               6099
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpJRight,               8707
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpJWrong,                  0
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpRight,               68873
```



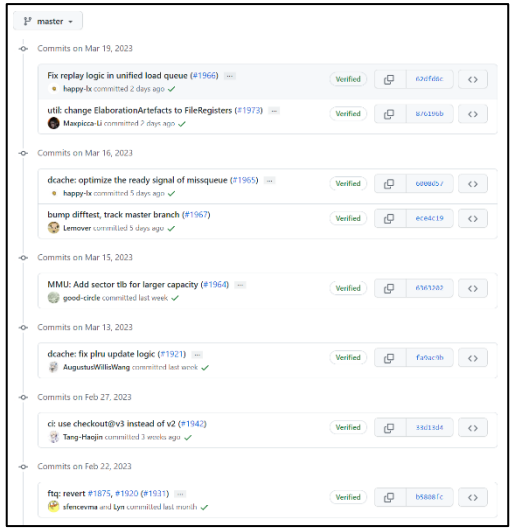


KUNMINGHU: 3rd generation microarchitecture

- Work with *Beijing Institute of Open Source Chip (BOSC)*
  - Non-profit organization with 18 founding members from the industry

- RISC-V Vector Extension 1.0 Support
- RISC-V Hypervisor Extension 1.0 Support
- Loop predictor, loop buffer
- Refactored issue queues, execution units, load/store units/queues
- L1D prefetcher, cross-level cache optimizations
- AMBA CHI compatible

- Comprehensive verification plan
- Advanced process node and EDA flows
- GEM5 simulator with aligned microarchitecture



Active development on GitHub

Roadmap of Kunminghu

- **Functional Improvement**
  - Support RISC-V **Vector** extension
  - Support RISC-V **Hypervisor** extension
- **Performance Exploration**
  - Performance upgrade of front-end, back-end, load-store unit and cache
  - Performance model calibrated with RTL
  - Workflow：**DSE on perf Model => Impl. & fine tunning on RTL**
- **Functional Verification**
  - **Hierarchical veri. flow** including ST、IT、UT + FPGA
  - Industrial-grade verification process
- **Physical Design**
  - Experienced physical design team
  - Simultaneous iteration of RTL coding with physical design timing evaluation

Performance Evaluation of Kuminghu

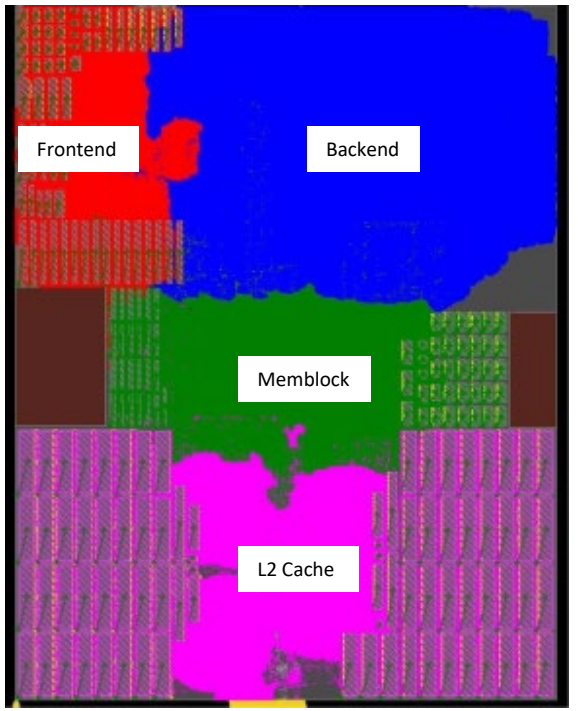
- Method: SPEC CPU checkpoints selected by Simpoint
  - Base: GCC 12 –O3, RV64GCB, jemalloc
  - 1MB L2 and 16MB L3
  - Simulated@3GHz with DRAMsim3 DDR4-3200

SPECint 2006	
400.perlbench	36.648
401.bzip2	24.283
403.gcc	47.692
429.mcf	57.852
445.gobmk	31.711
456.hmmer	39.567
458.sjeng	31.500
462.libquantum	125.487
464.h264ref	57.376
471.omnetpp	42.243
473.astar	30.738
483.xalancbmk	75.535
SPECint2006@3GHz	44.977

Estimated SPECint 2006 Base
44.98@3GHz

SPECint 2006	
400.perlbench	42.143
401.bzip2	24.283
403.gcc	47.692
429.mcf	57.772
445.gobmk	31.711
456.hmmer	60.826
458.sjeng	31.500
462.libquantum	222.412
464.h264ref	60.243
471.omnetpp	42.243
473.astar	30.738
483.xalancbmk	81.213
SPECint2006@3GHz	49.964

Estimated SPECint 2006 Peak
49.96@3GHz

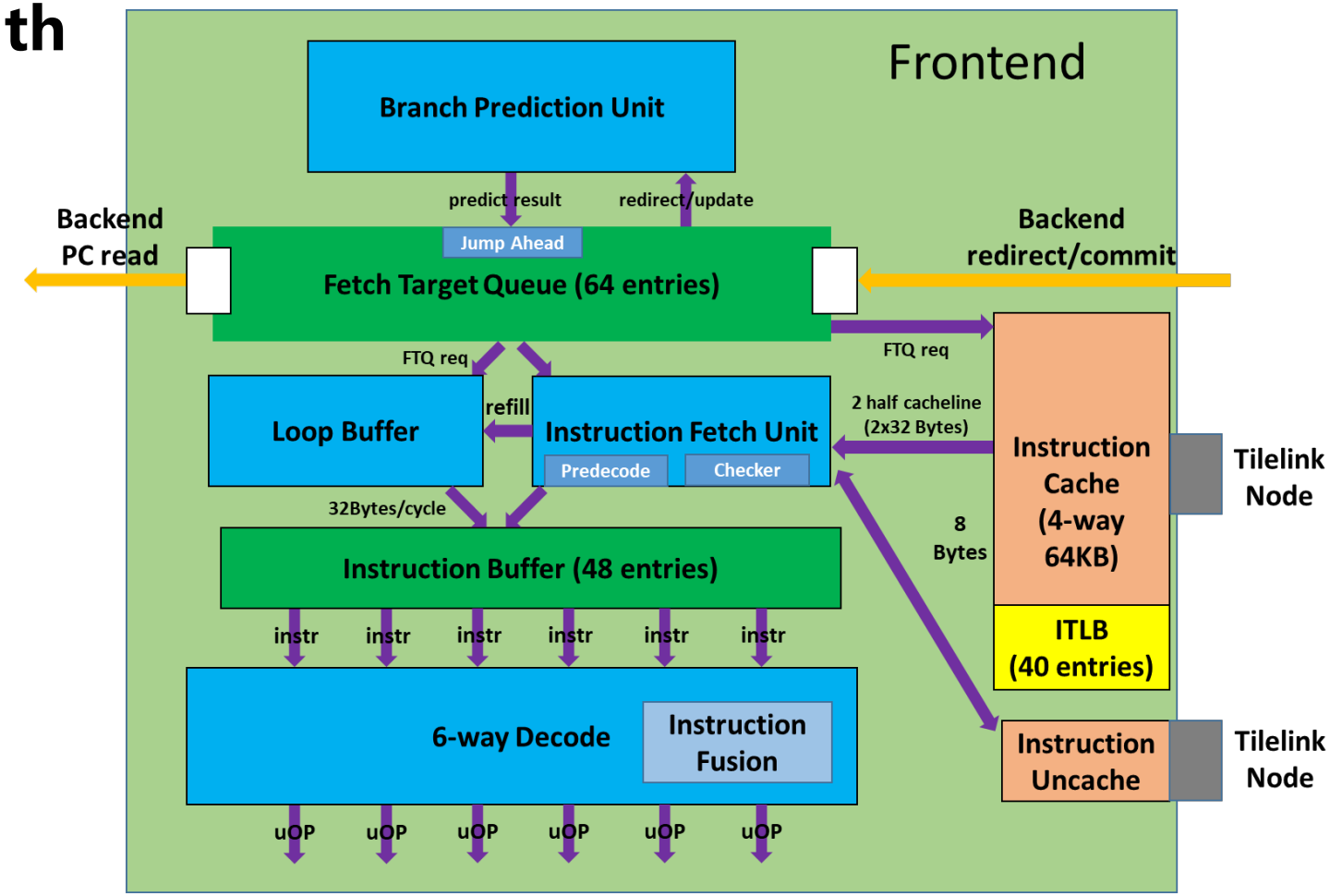


Floorplan of V3 (single core)

* Updated in December 2023

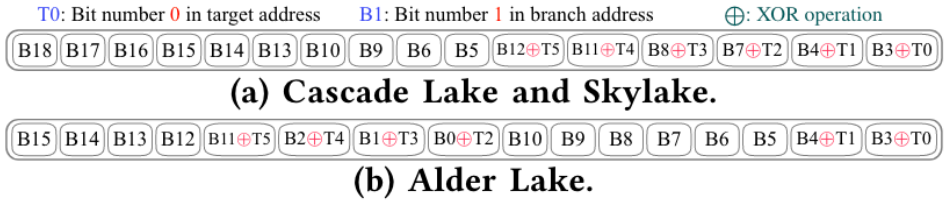
① Frontend Upgrade

- Larger fetch/prediction width
- Lower prediction miss rate
- Higher frequency



Feature 1: TAGE Predictor Fine Tuning

- ❶ **Branch History Hashing**



(a) Cascade Lake and Skylake.

(b) Alder Lake.

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**

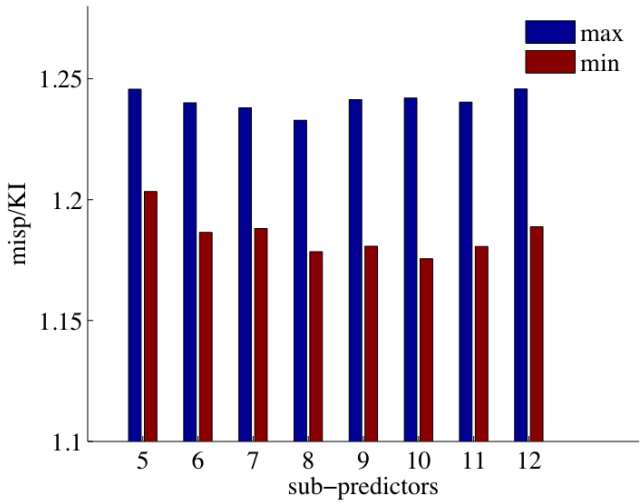


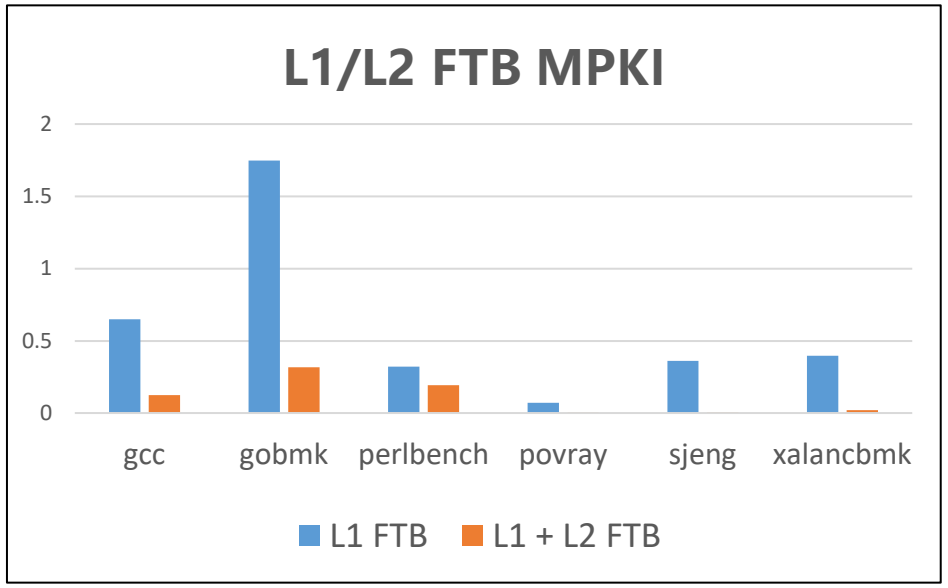
Figure 2: Performance of random parameters

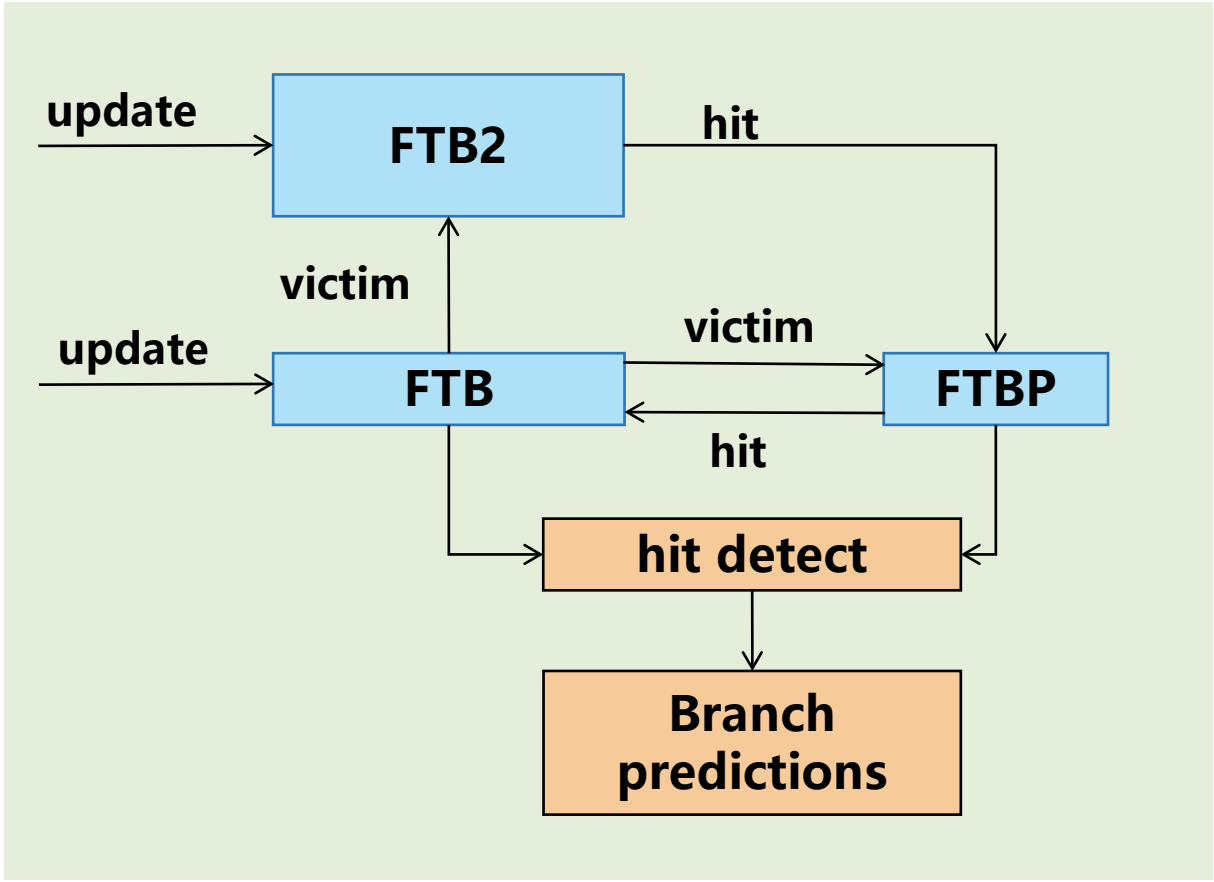
Previous studies have shown that the #tables and history length have a significant effect on TAGE performance, up to 5.64% in MPKI [1]

[1] 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.

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

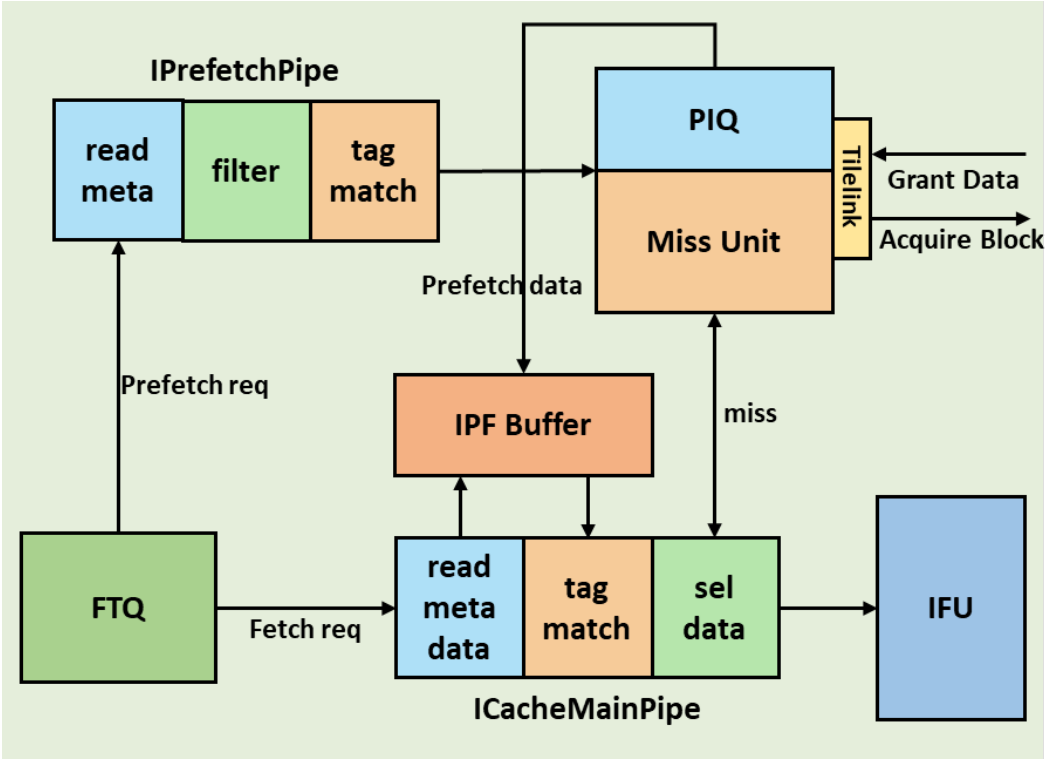
Feature 3: Smart ICache Prefetcher

- Based on decoupled frontend, FDIP prefetching algorithm is implemented

Prefetcher workflow

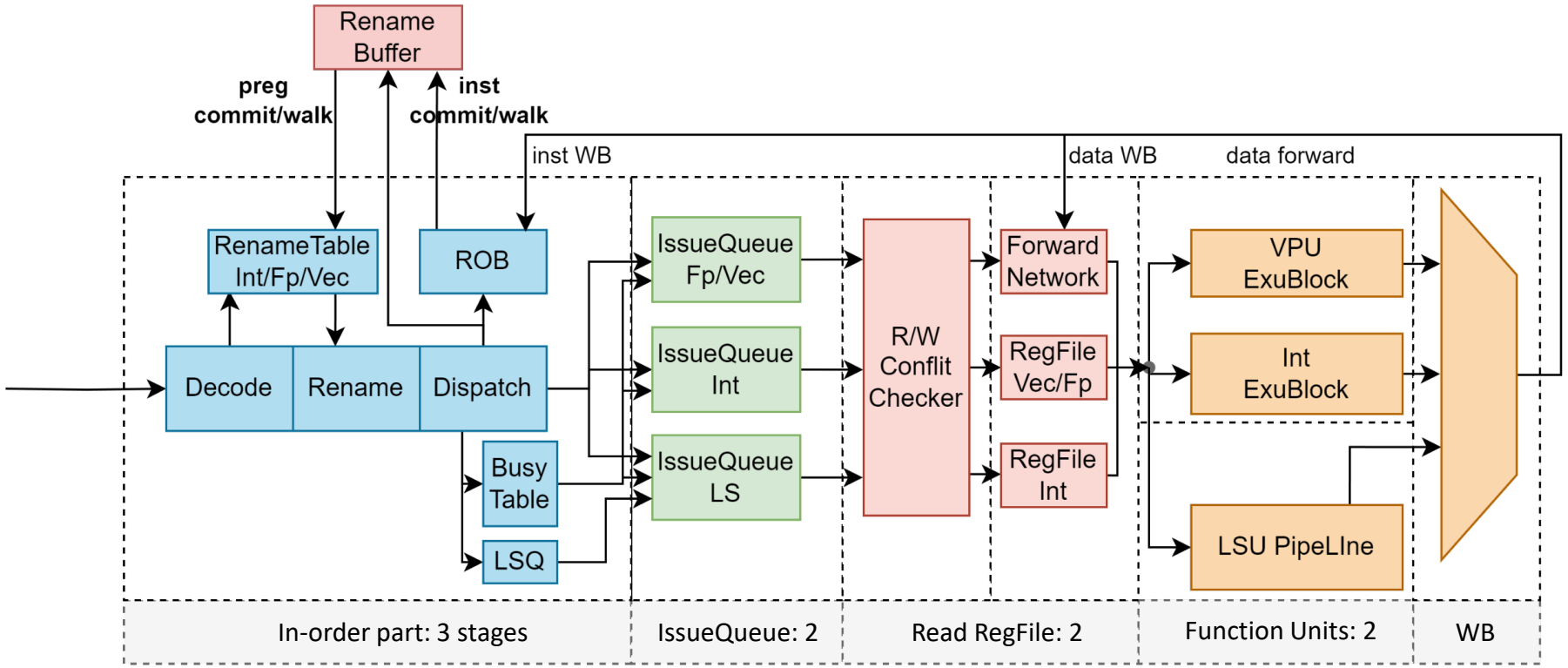
1. Select prefetch request from PF-queue
2. Write refilled data into prefetch buffer
3. Fetch query icache & prefetch buffer parallelly

- Prefetch position: L2 Cache or L1 ICache
- Dual prefetching pipelines to enlarge bandwidth



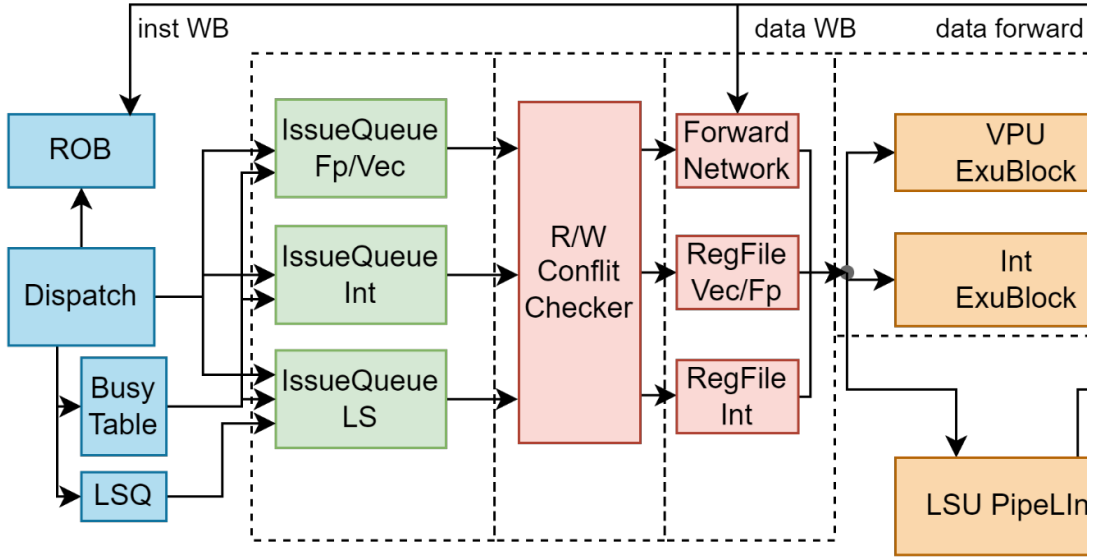
② OoO Backend Upgrade

- **Read regfile after issue**
- **Faster µop commit and redirect**
- **Vector execution unit**



Feature 1: Read Regfile after issue

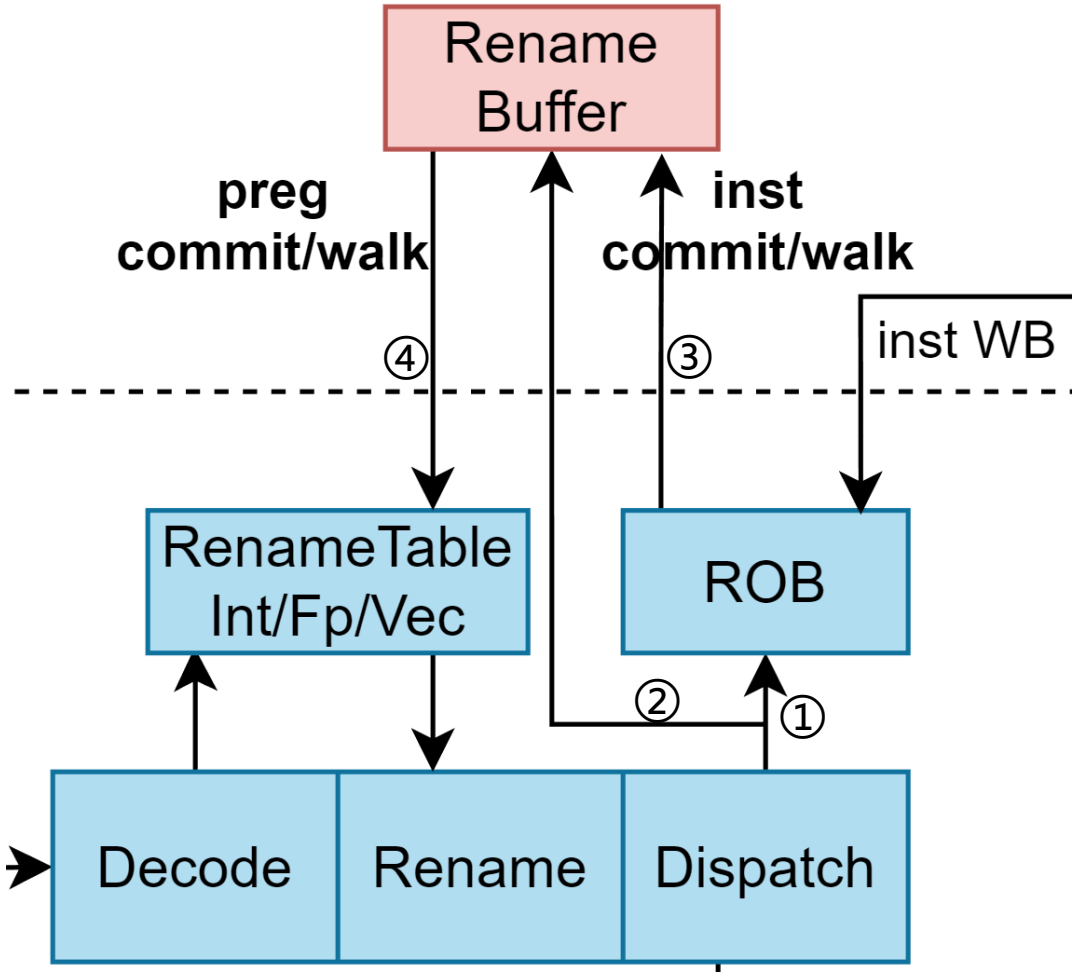
- Advantage
  - Bottleneck is shifted to after µop issue
  - Reduce RS storage to save SRAM
  - Reduce latency and increase RS capacity
- Optimization
  - Efficient arbitration of regfile reading ports
  - Accurate speculative wakeup and cancelling



Read regfile after issue

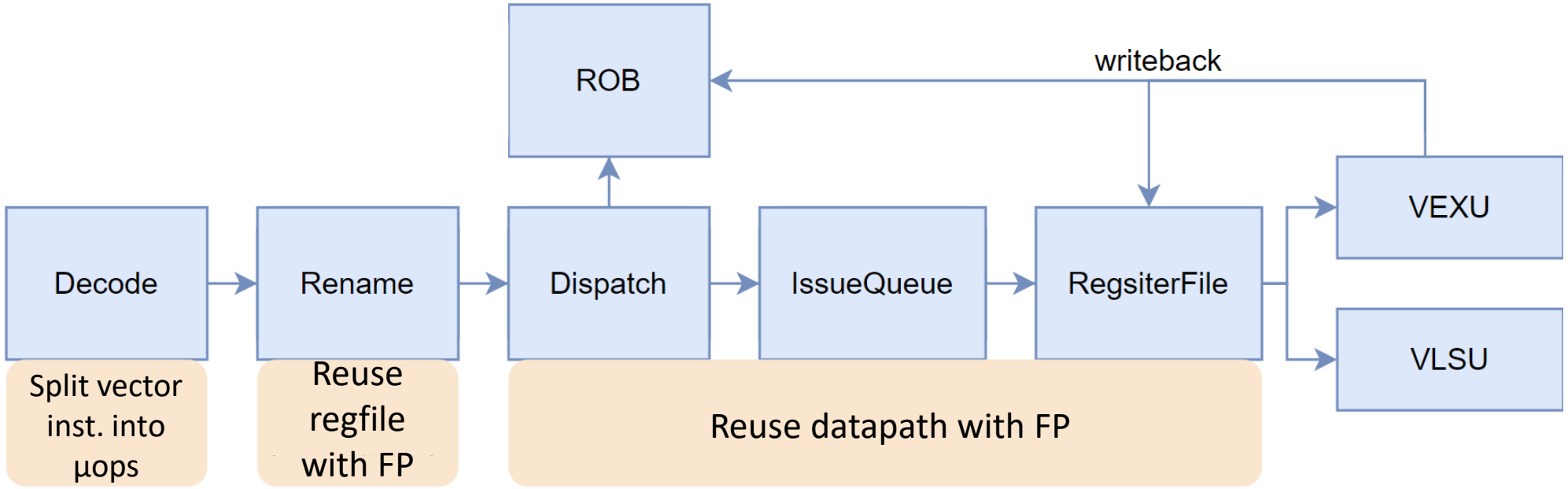
Feature 2: Decouple µop commit & regfile commit

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



Feature 3： Support Vector Extension

- Coupled vector unit design, compatible with RISC-V Vector 1.0
  - VLEN = 128
- Reuse backend pipeline： Decode/Dispatch/OoO/Execute



③ Load-Store Unit Upgrade

Cons: High L1D SRAM power

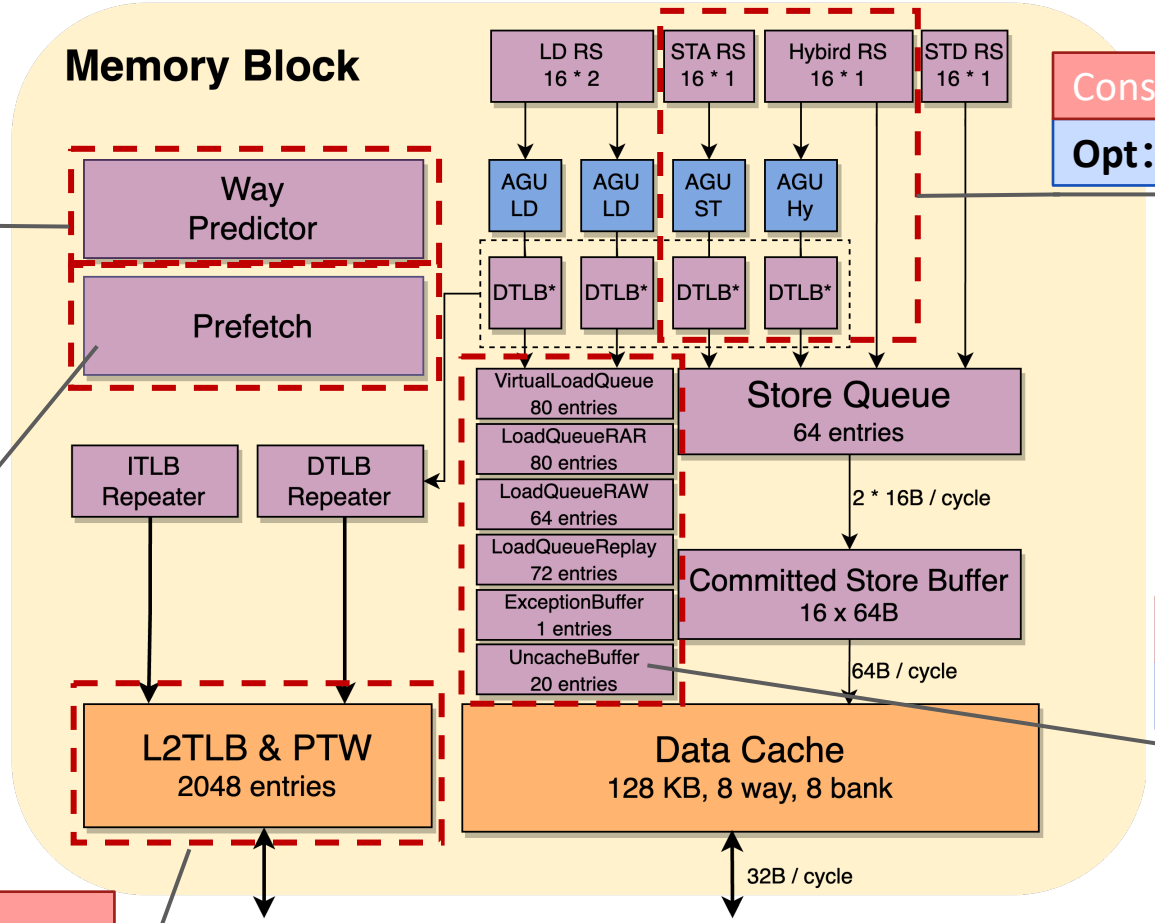
Opt： Support way prediction

Cons: High miss rate in L1D

Opt： Support L1D prefetch

Cons: Bad timing in TLB

Opt： Re-design structure of L1/L2 TLB



Cons: Low load/store bandwidth

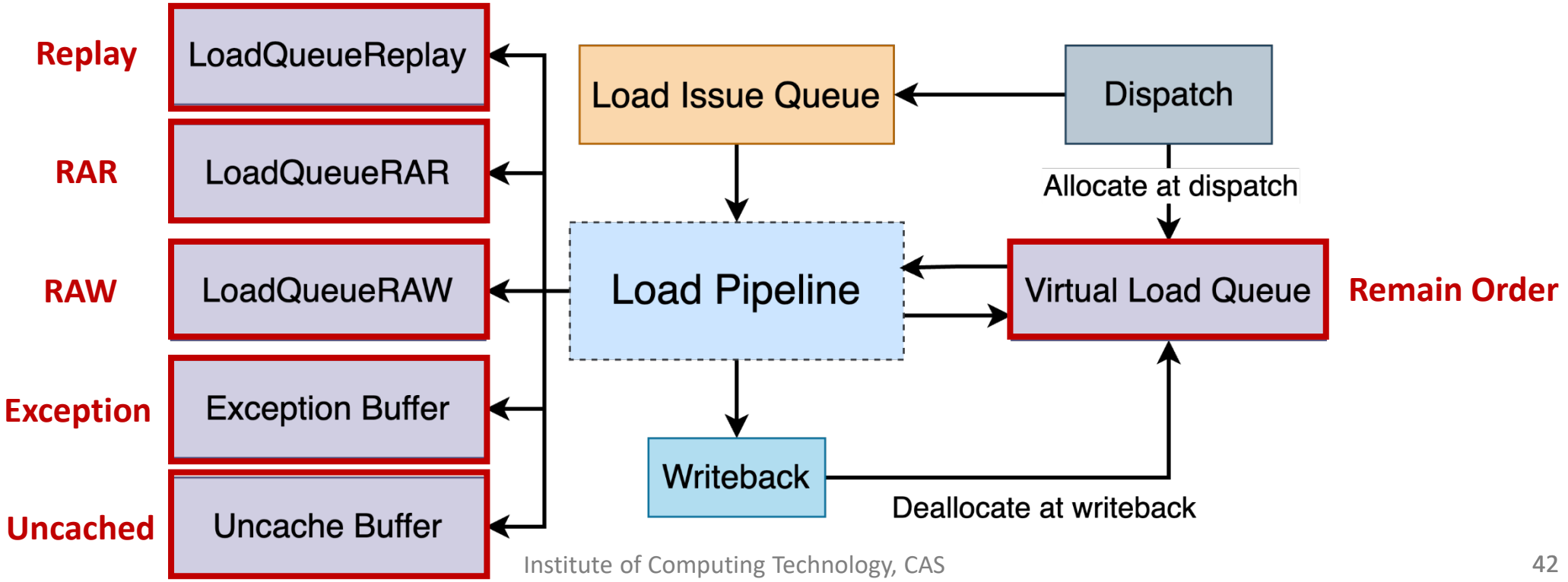
Opt： Support hybrid ld/st unit

Cons: bad timing in load queue

Opt： Split load queue

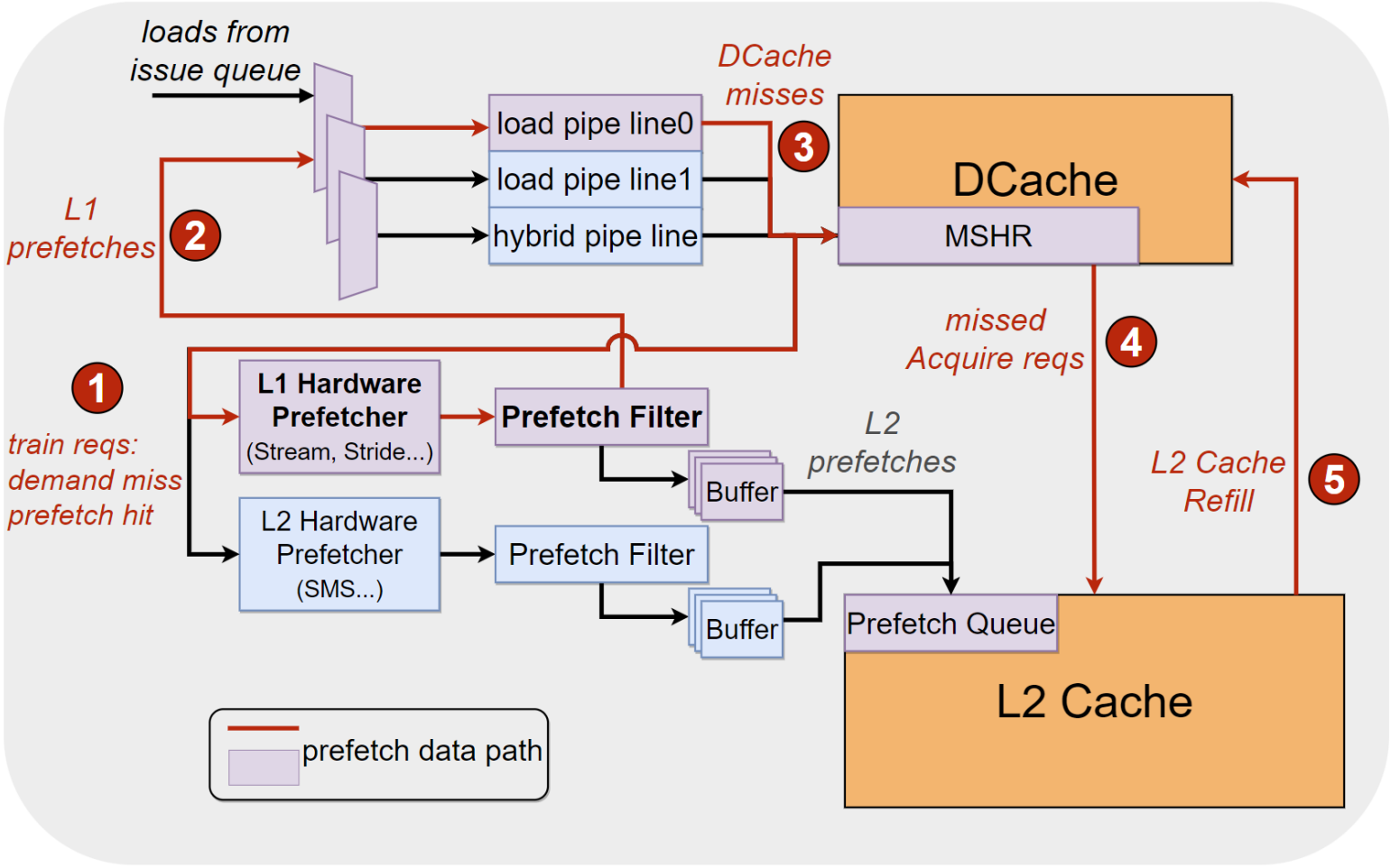
Feature 1: Split Load Queue

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



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...)

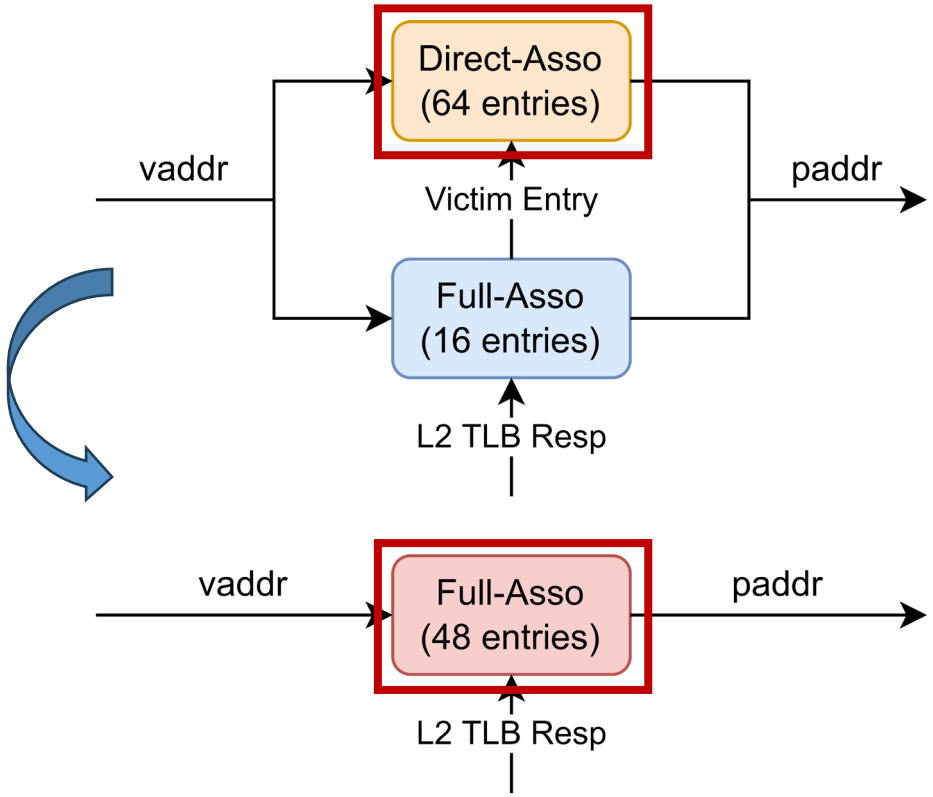


[1] S. Srinath, O. Mutlu, H. Kim and Y. N. Patt, "Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers," 2007 IEEE 13th International Symposium on High Performance Computer Architecture, Scottsdale, AZ, USA, 2007

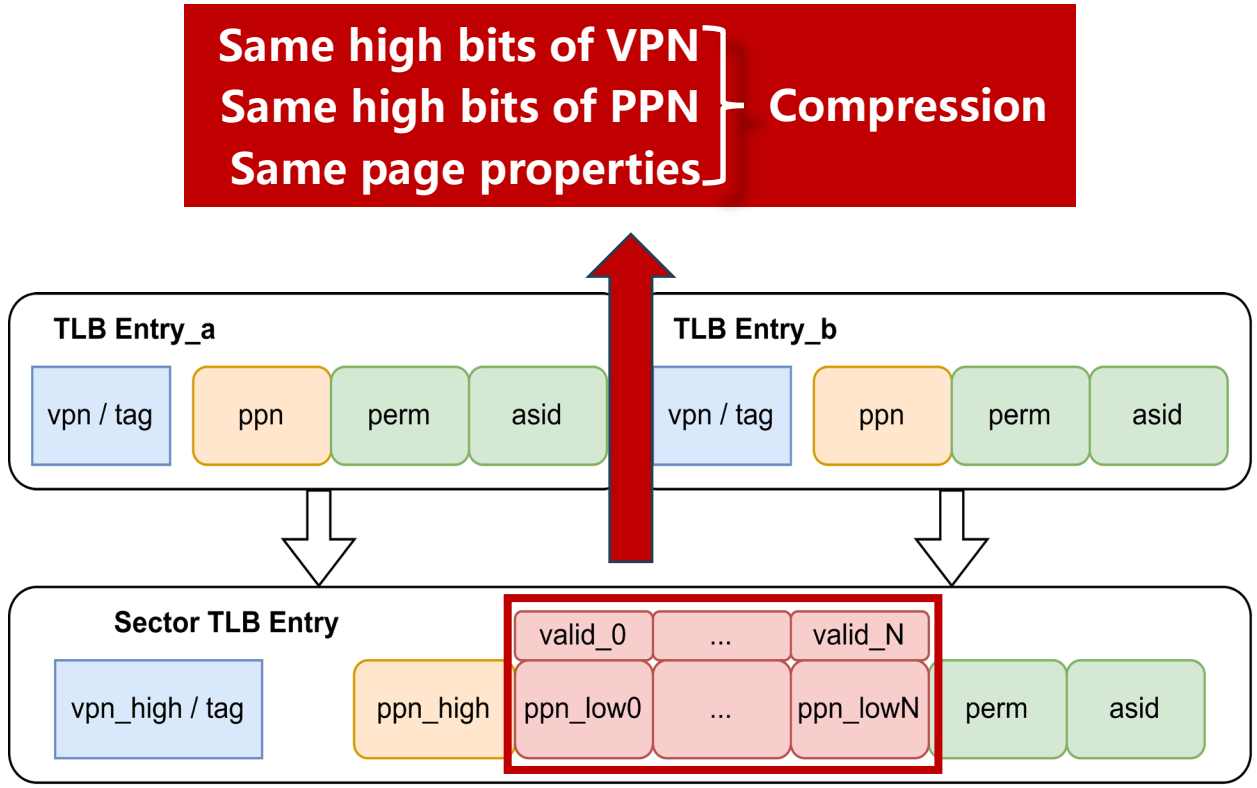
[2] J. -L. Baer and T. -F. Chen, "An effective on-chip preloading scheme to reduce data access penalty," Supercomputing '91:Proceedings of the 1991 ACM/IEEE Conference on Supercomputing, Albuquerque, NM, USA, 1991

Feature 3: Powerful MMU

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



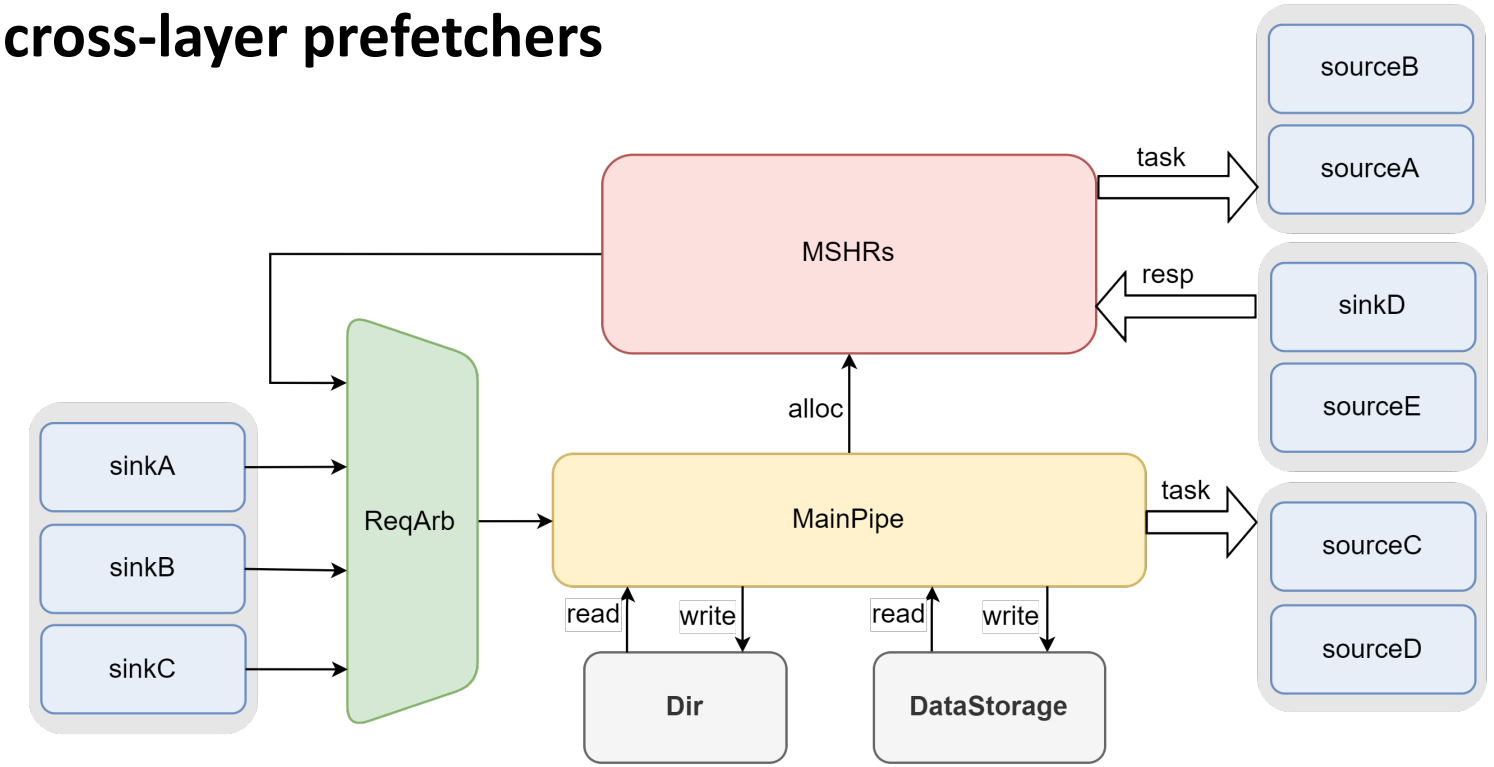
DTLB organization in Nanhu and Kunminghu



TLB compression

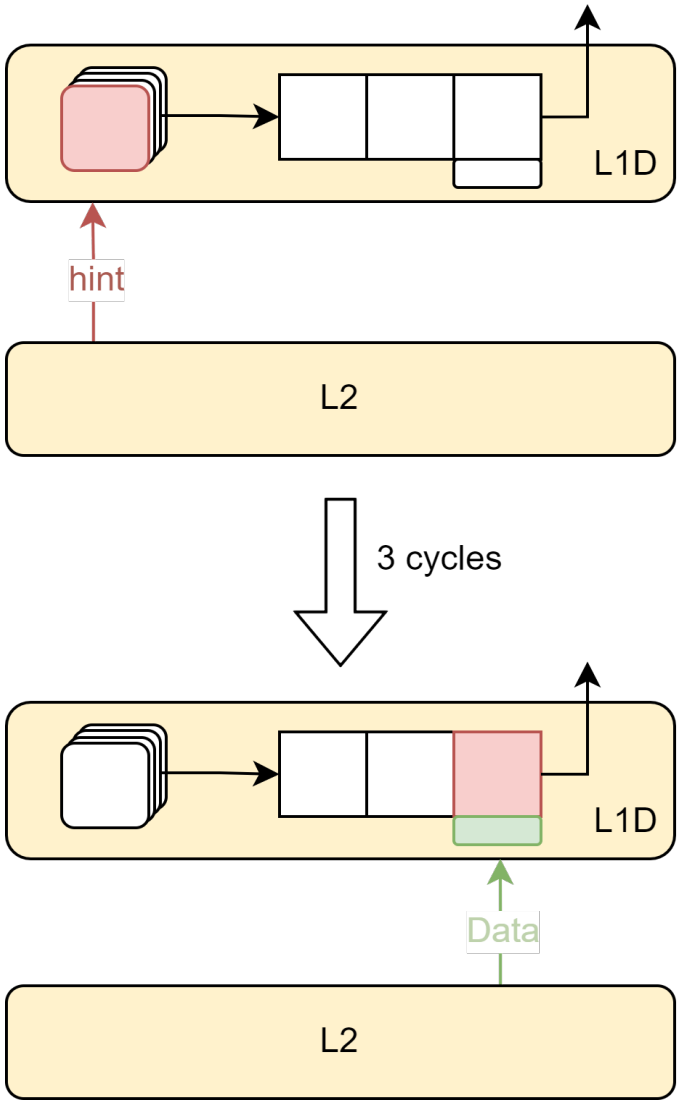
④ L2 Cache Upgrade

- **Multiple concurrent pipelines → Non-blocking main pipeline**
- **L1-L2 Refill wake-up collaboratively**
- **Eliminate transaction serialization in the same set & Request merge**
- **Aggressive cross-layer prefetchers**



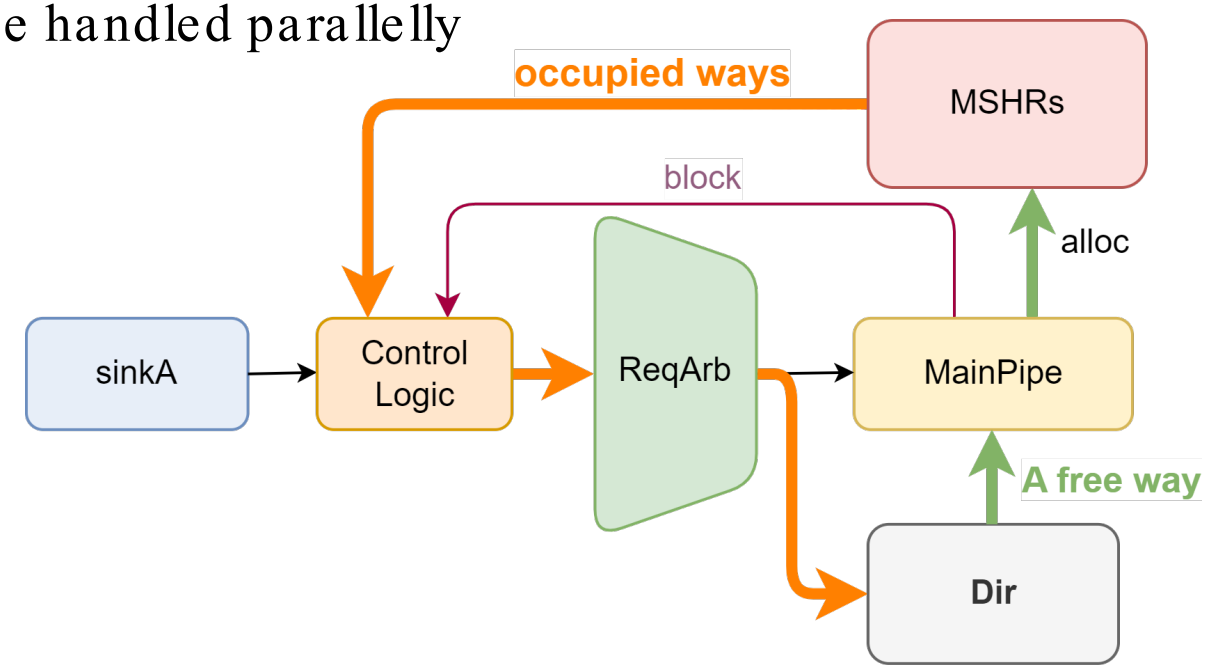
Feature 1: L1-L2 Collaborative Optimization

- 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 wake and replay of load replay queue
- Implementation:
  - Set up monitor to track info from main pipeline
  - Send hint signal to L1 3 cycles before refill
  - Need to calculate the exact Hint timing considering possible pipeline delays



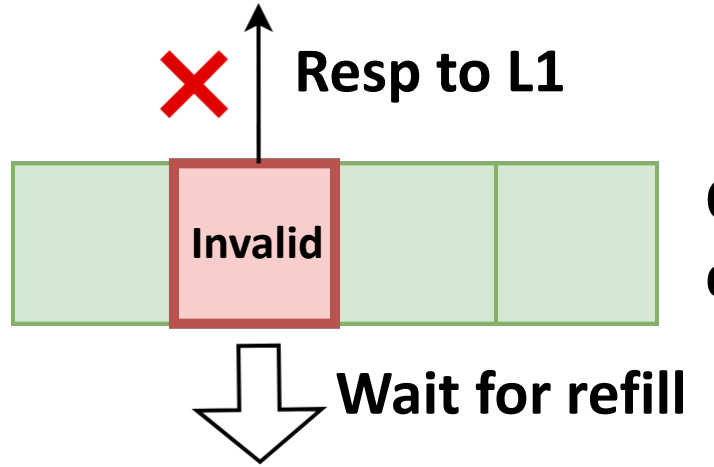
Feature 2: Eliminate Txns Serialization in the same set

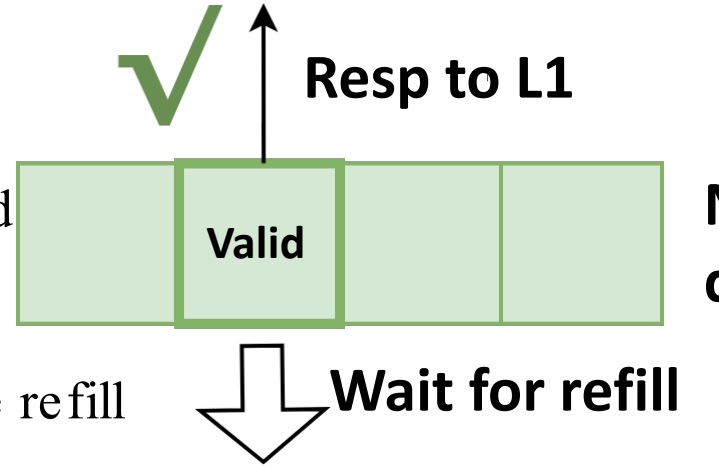
- Background: Txns to the same set were handled serially, which reduces parallelism
- Difficulty: Same-set txns can affect each other, due to replacement
- New Design
  - Txns of same-set but different-addr can be handled parallelly
  - Txns of same-addr are handled serially
- Implementation
  - Record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



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 and assign one way when MSHR handles the refill

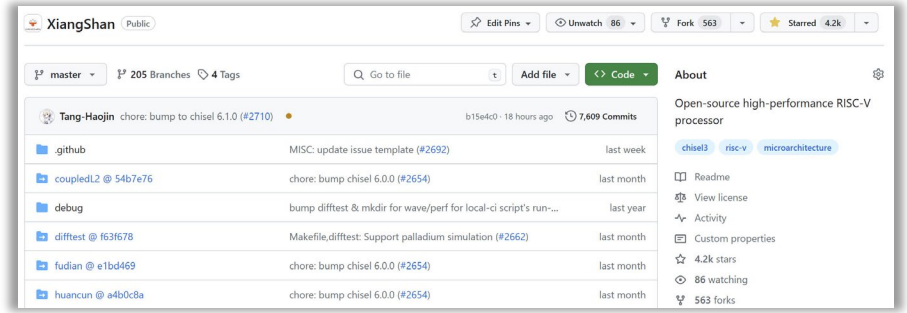




XiangShan: Open-Source High-Performance Processor

- **1st generation: 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: NANHU**
  - 2021/5: starting design exploration and RTL design
  - 2023/4: GDSII delivery
  - **14-nm tape-out, estimated SPEC CPU2006 20@2GHz**
- **3rd generation: KUNMINGHU**
  - ISA feature: **Vector (V) extension, Hypervisor (H) extension**
  - Comprehensive performance improvement

Fragrant Hills in Beijing



>4.2K stars, >500 forks on GitHub

- Open-sourced at https://github.com/OpenXiangShan/XiangShan

Thanks!

This tutorial is organized by Yinan Xu, Kaifan Wang, Zefan Xu, and Zeyu Gao.