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)

MICRO'23@Toronto, Canada

October 28, 2023

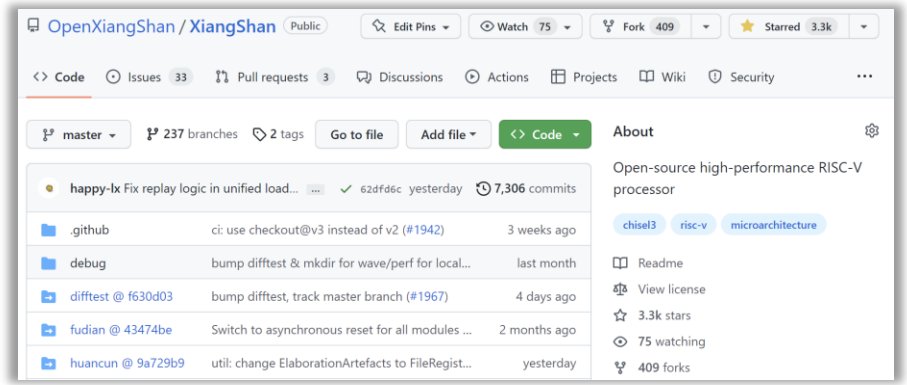
Tutorial@MICRO'23 Schedule

Time (PM)	Topic
1:00 - 1:25	Introduction of the XiangShan Project
1:25 - 1:30	Tutorial Overview and Highlights
1:30 - 2:30	***Microarchitecture Design and Implementation***
2:30 - 3:00	Hands-on Development
	Coffee Break
3:30 - 4:00	Hands-on Development & Discussions

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
  - 2022/12: RTL Freeze
  - Plan: **14-nm tape-out soon, estimated SPEC CPU2006 20@2GHz**
- **3rd generation: KUNMINGHU**
  - ISA feature: **Vector (V) extension, Hypervisor (H) extension**
  - Comprehensive performance improvement

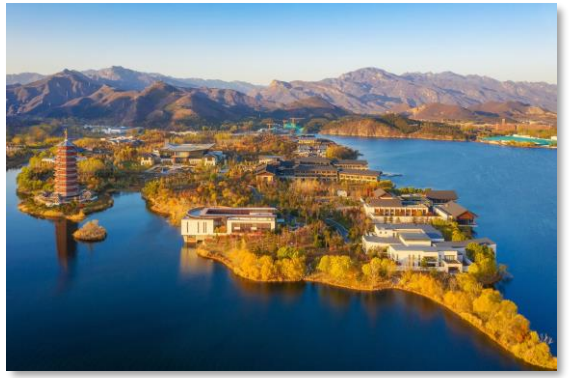
Fragrant Hills in Beijing



>3.2K stars, >400 forks on GitHub

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

Yanqihu: 1st generation of XiangShan



Yanqi Lake in Beijing

- **Yanqihu**: named after a lake in Beijing, China
  - RV64GC, 11-stage, superscalar, out-of-order
  - 5.3 CoreMark/MHz (gcc-9.3.0 –O2)
  - Real chip: SPEC CPU2006 7@1GHz with DDR4-1600 (DDR not fully optimized)
- **Tape-out**: single XiangShan core (commit hash ccbca07) with 1MB L2 Cache

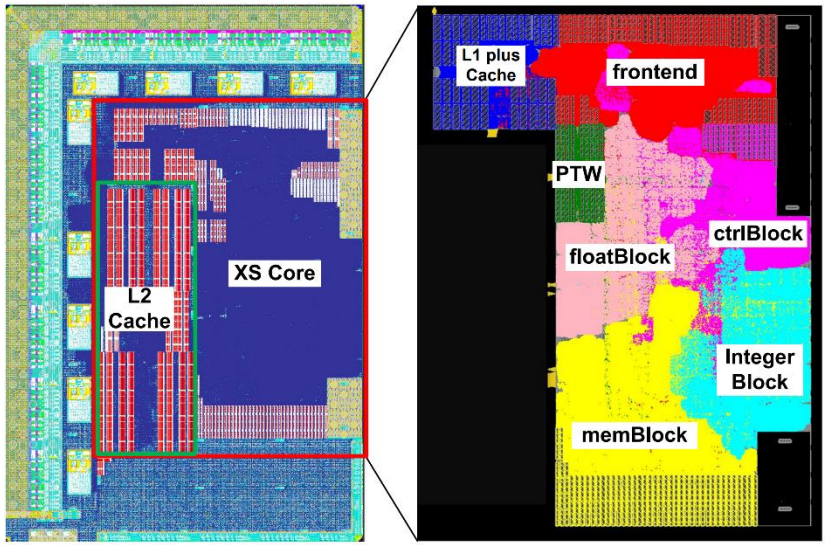
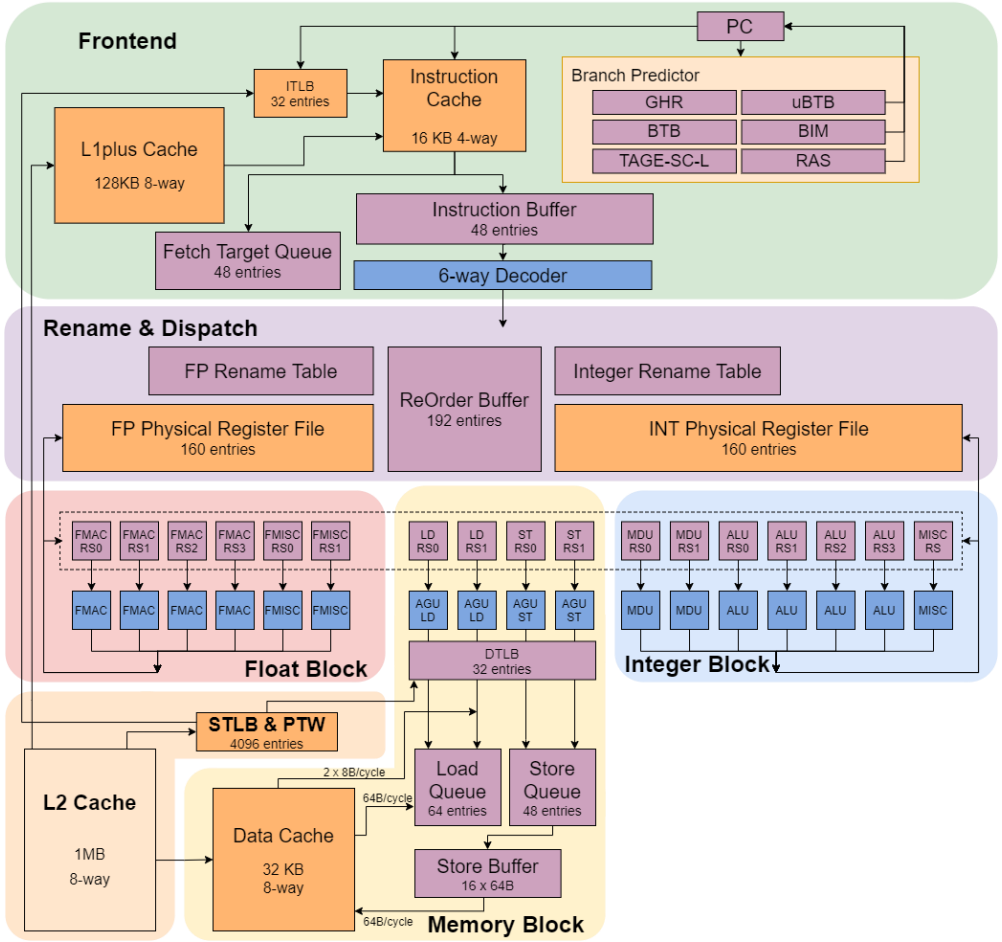


Figure. Layout of (a) the entire chip; (b) the core

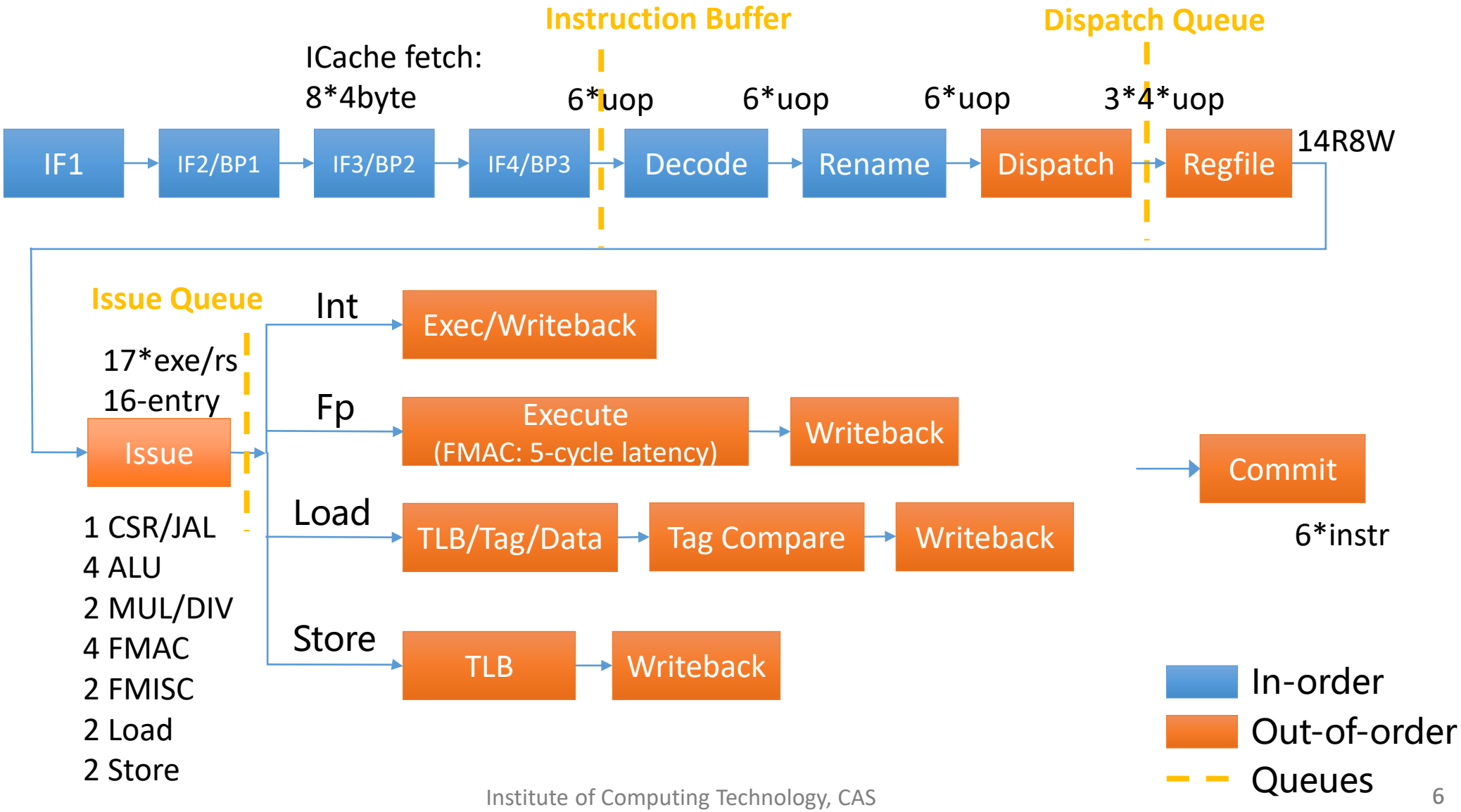
Tape-out information for the processor core	
Process Node	28nm
Die Size	8.6 mm^2
Std Cell	5.05M, 4.27 mm^2
Mem	261, 1.7mm^2
Density	66%
Cell	ULVT 1.04%, LVT 19.32%, SVT 25.19%, HVT 53.67%
Estimated Power	5W
Frequency	1.3GHz, TT85C

XiangShan Microarchitecture (Yanqihu)

- **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 (Yanqihu)



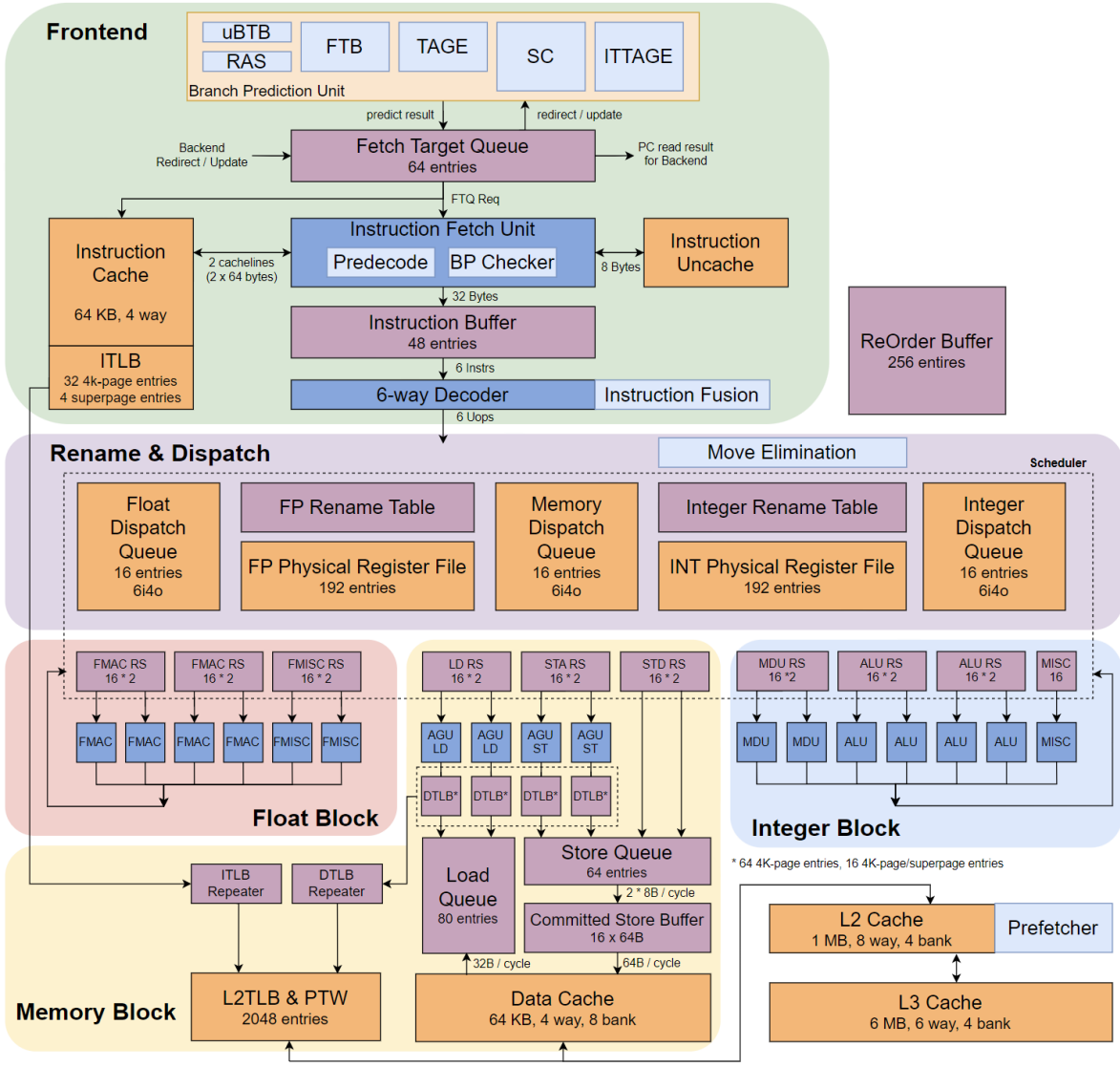
NANHU: 2nd generation microarchitecture

- Named after a lake in Jiaxing, Zhejiang, China
- **2GHz@14nm, taped out**
- Estimated **SPECint 2006 19.10, SPECfp 2006 22.18**

- **Major changes**
  - New frontend design: decoupled BP and instruction fetch
  - Improved backend: better scheduler, instruction fusions, and more
  - New L2 cache: designed for high frequency and high performance
  - Tape-out with dual cores (RV64GCBK), more devices support in SoC (PCIe, USB, …)

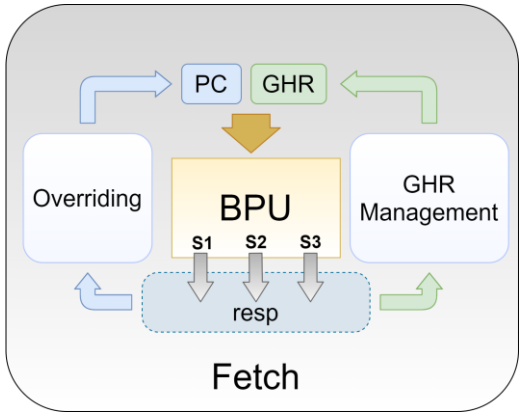
XiangShan Microarchitecture (Nanhu)

- **192** Int PRF + **192** FP PRF
- **256**-entry ROB, **80**-entry LQ, **64**-entry SQ
- **16**-entry RS for each FU (32-entry as an RS)
- **64KB** L1 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)



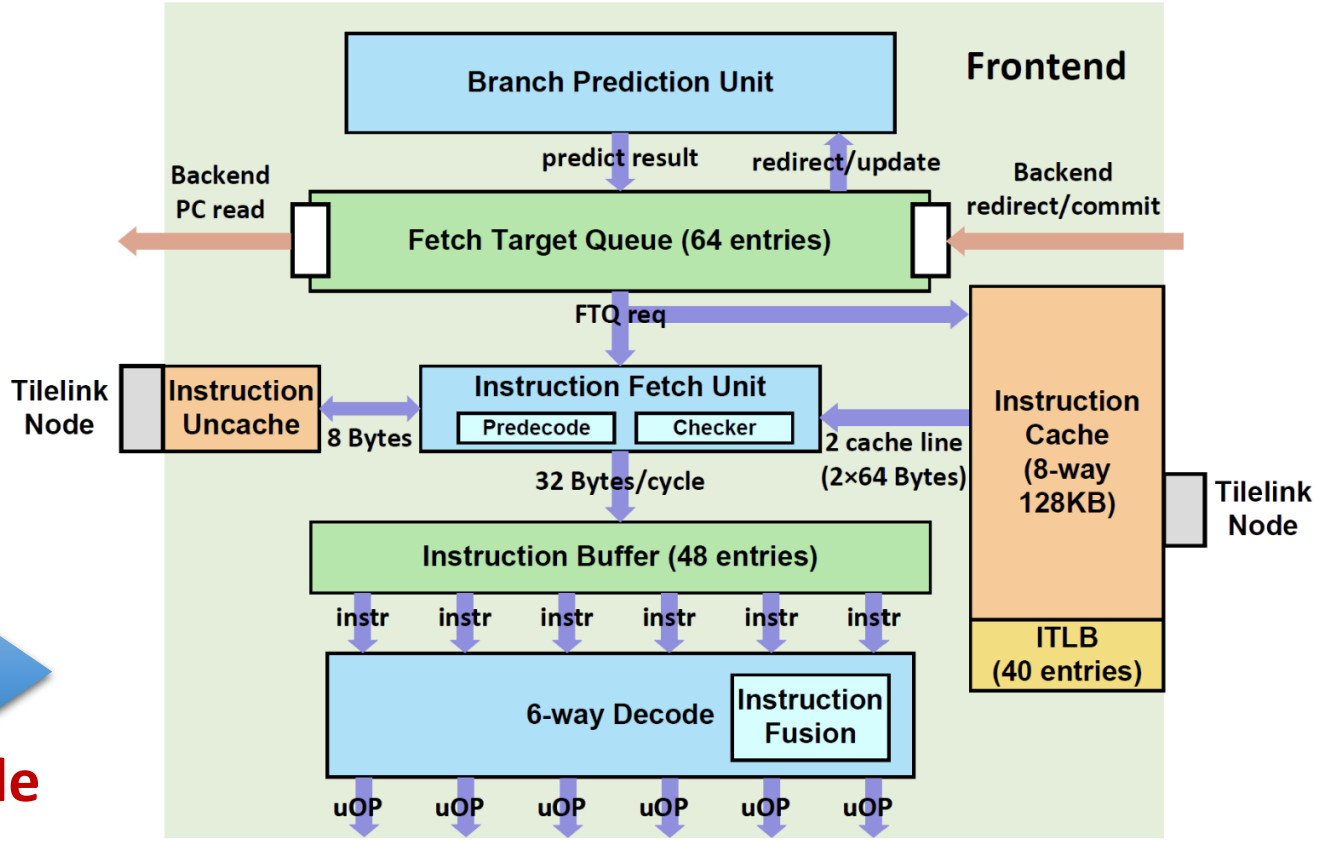
Frontend

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



Yanqihu

Upgrade



Nanhu

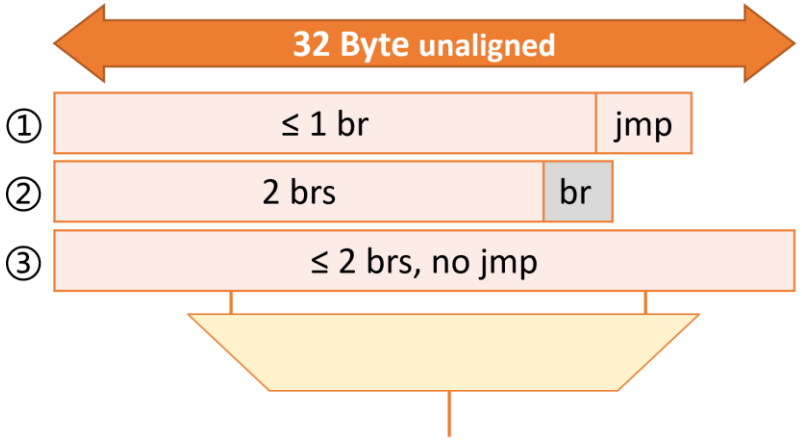
* Frontend design of YQH is partially learnt from BOOM

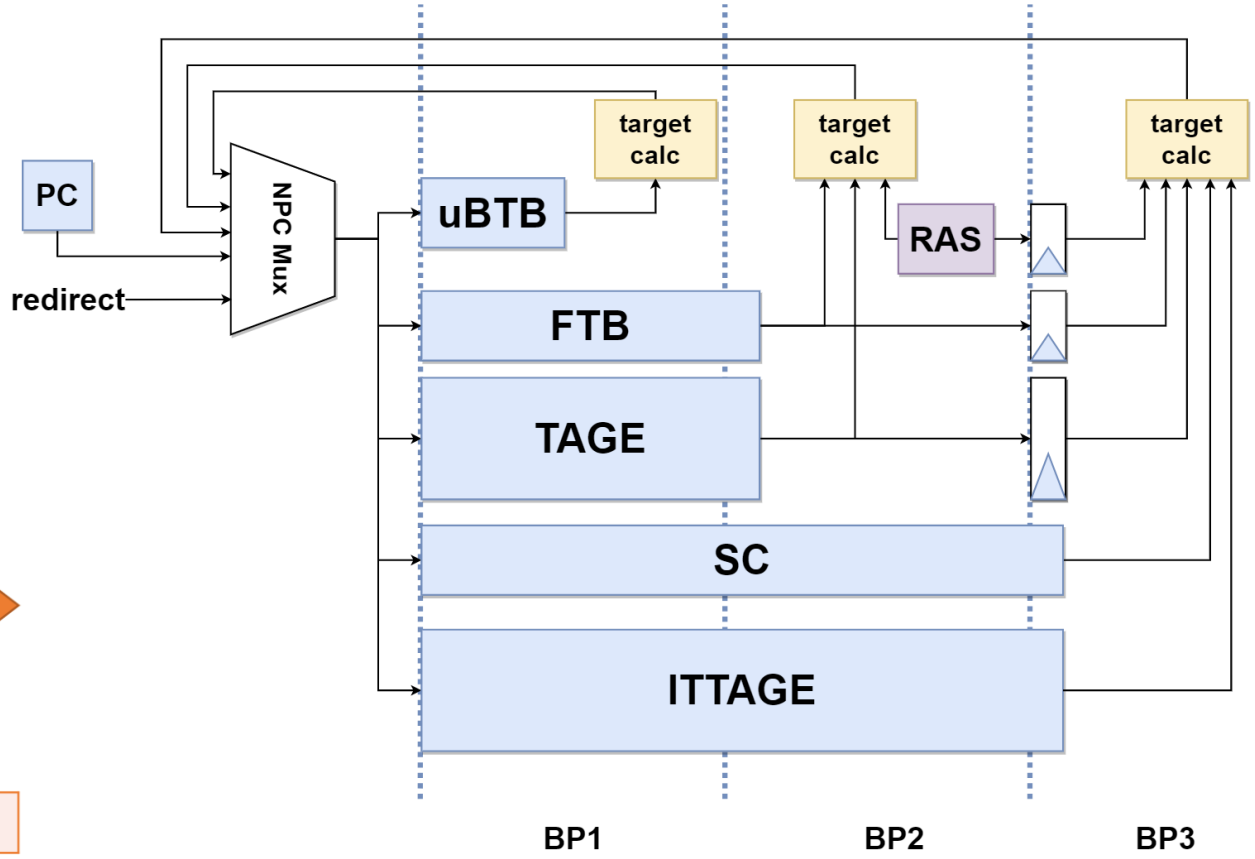
Branch Predictor

Three-stage overriding prediction

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

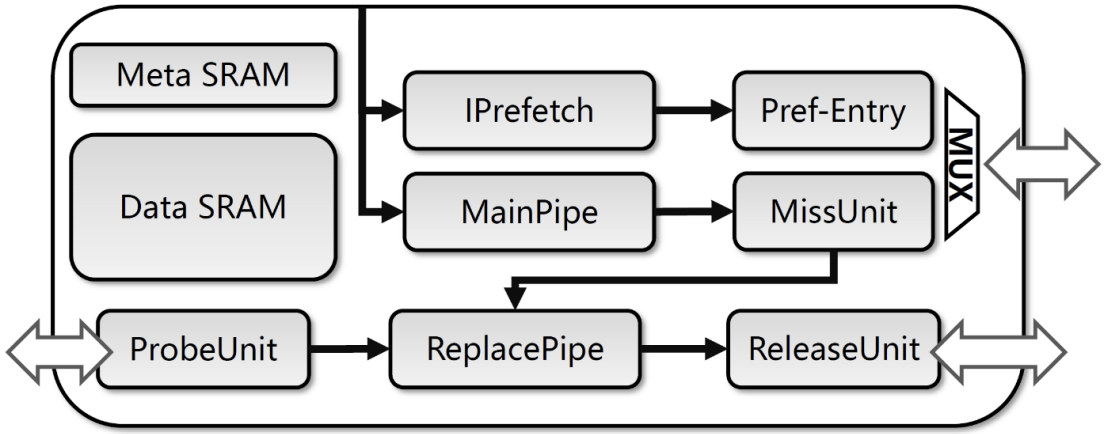
- FTB: Fetch Target Buffer





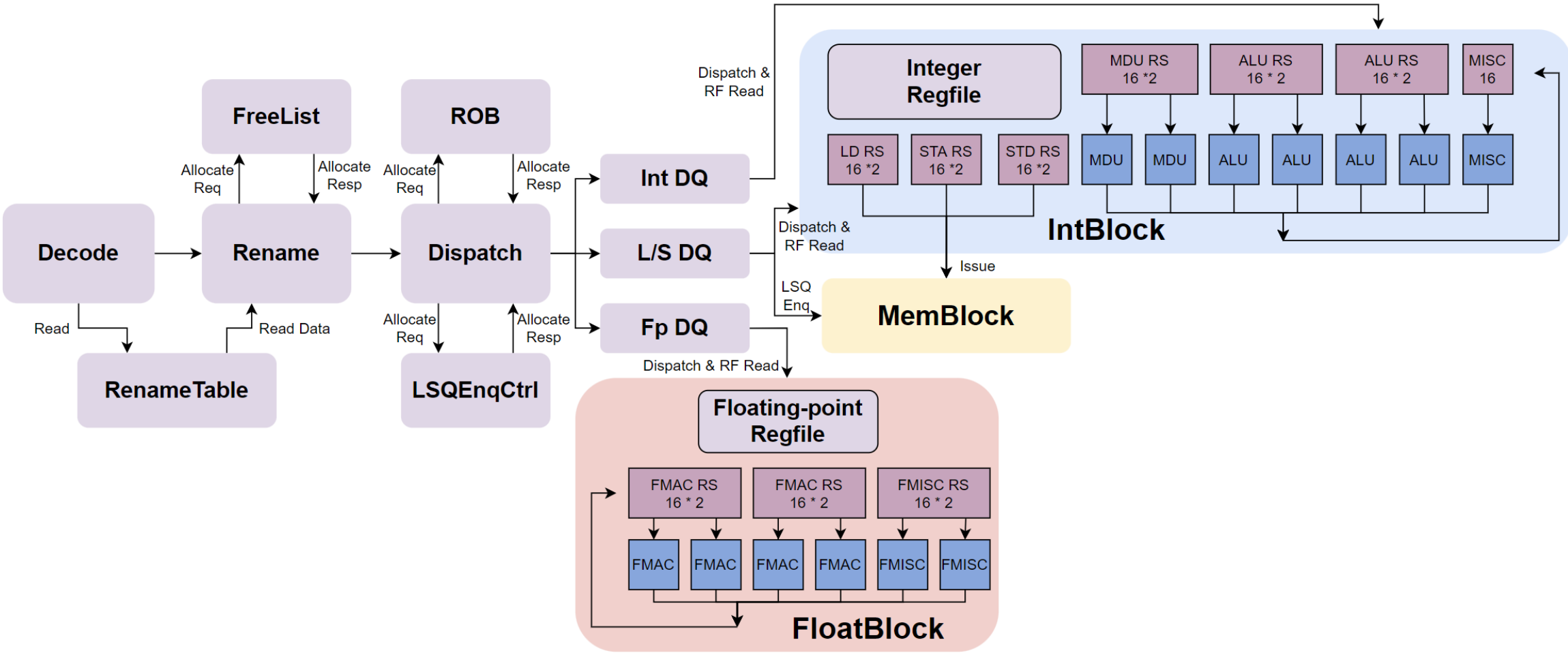
Instruction Cache

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



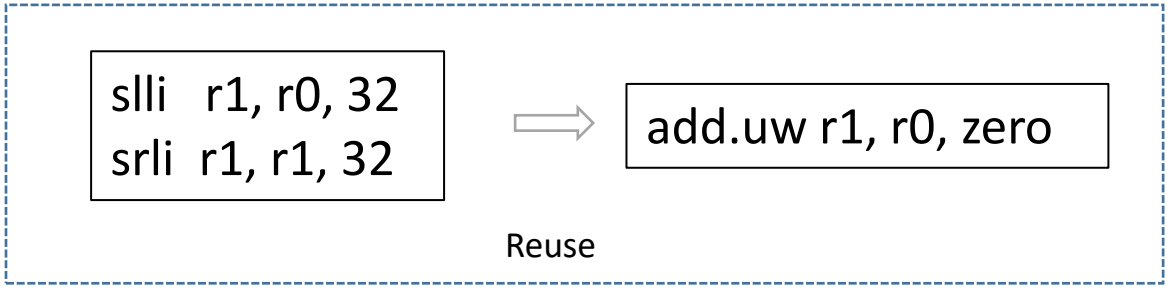
	Yanqihu	Nanhu	
ICache level	Two	One	Reduce
Size	16KB, 4-w	64KB, 4-w	4×
L1plus Cache	128KB, 8-w	-	Reduce
Read bandwidth	64 B	128 B	2×
TileLink Support	No	Yes	New
Instruction prefetch	Stream	L2 FDIP	New

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
- How can we do that?
  1. Reuse of the original calculation
  2. Designing new types of calculations
- **Reduce dynamic instr. greatly**



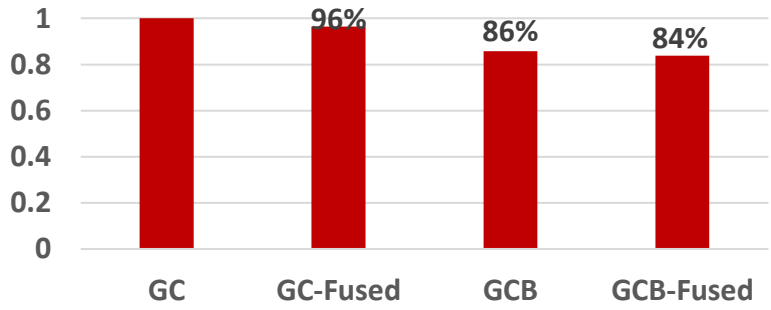
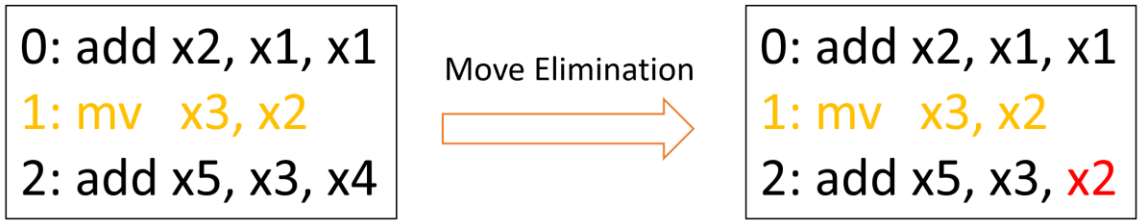
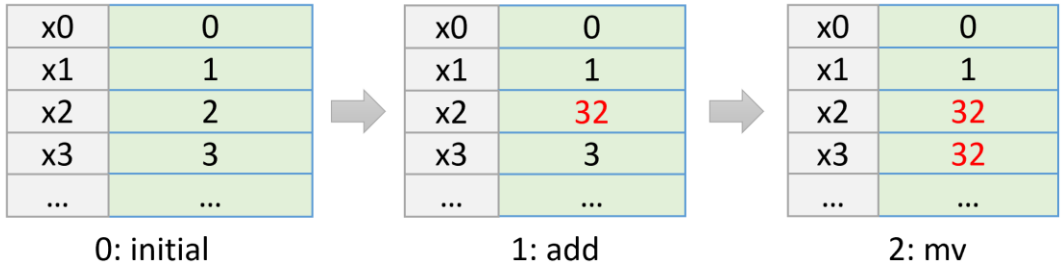


Figure. Normalized Dynamic Instruction Count for CoreMark (-O2)

Rename & Move Elimination

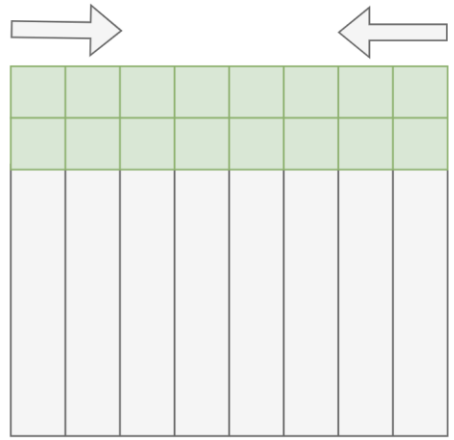
- Maintain the mapping between **arch registers** and **physical registers**
- Move Elimination
  - Mark as **Complete** when Move instruction enters ROB
  - Use **Reference-Counter** to record mapping counts of physical registers



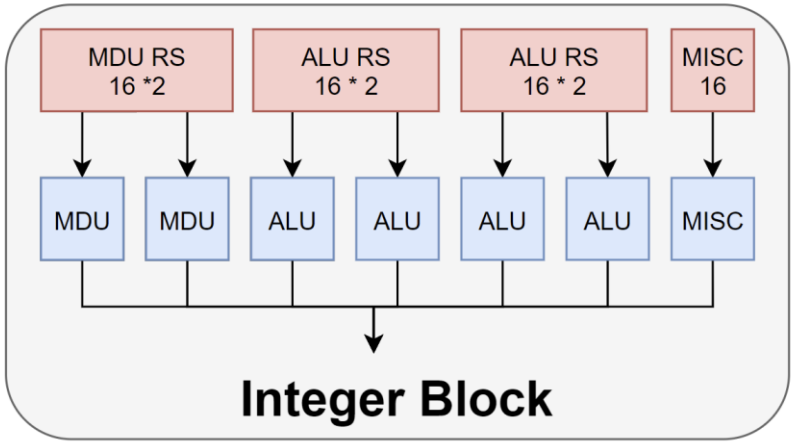


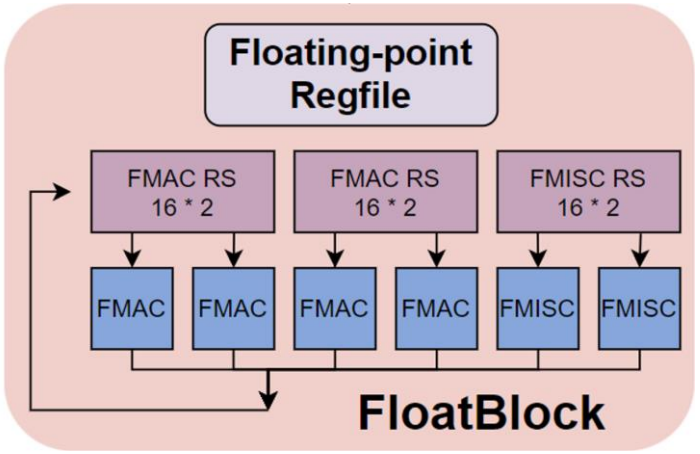
Reservation Station

- Hybrid reservation station design
  - 2i1o & 1i1o
  - normal selection & age matrix base selection



Normal Select





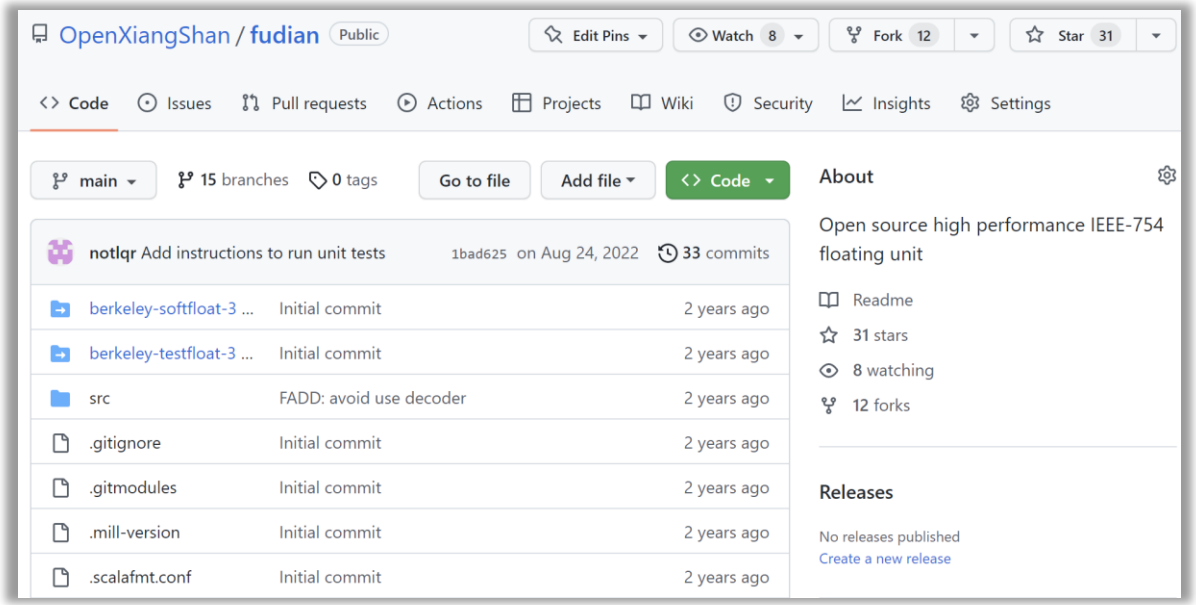
0	true			
1	true	true		
2	true	false	true	
3	true	false	false	true
	0	1	2	3

Age Matrix

Floating-Point Units (FPUs)

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

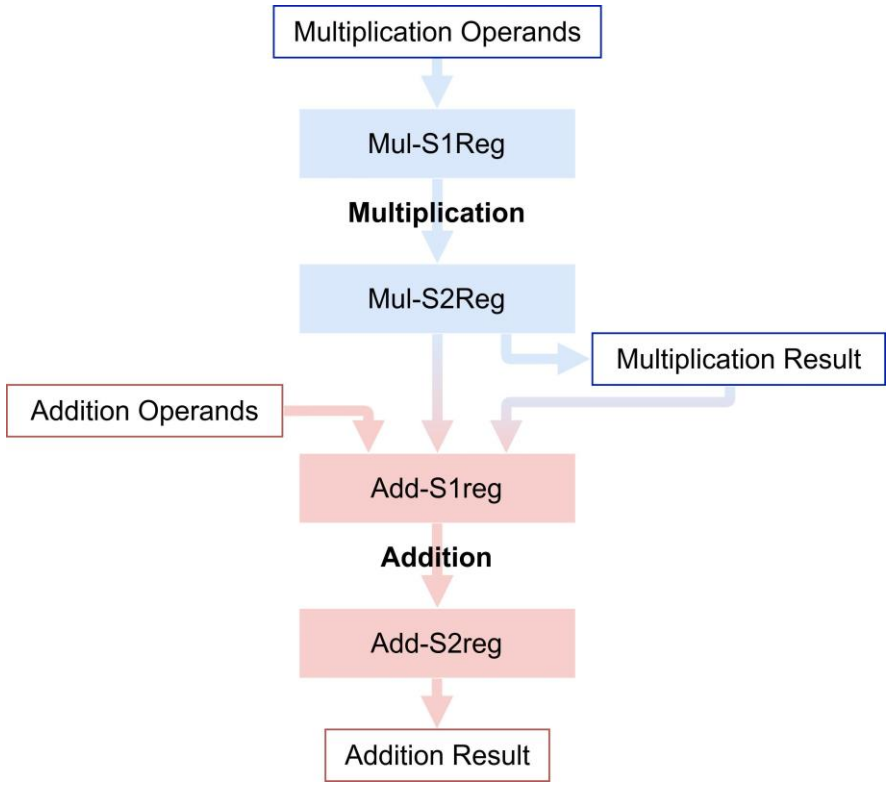
Operations	Latency
FADD	3
FMUL	3
FMA	5
FDIV	≤ 11 (float) ≤ 18 (double)
FSQRT	≤ 17 (float) ≤ 31 (double)
FCVT, FCMP	3

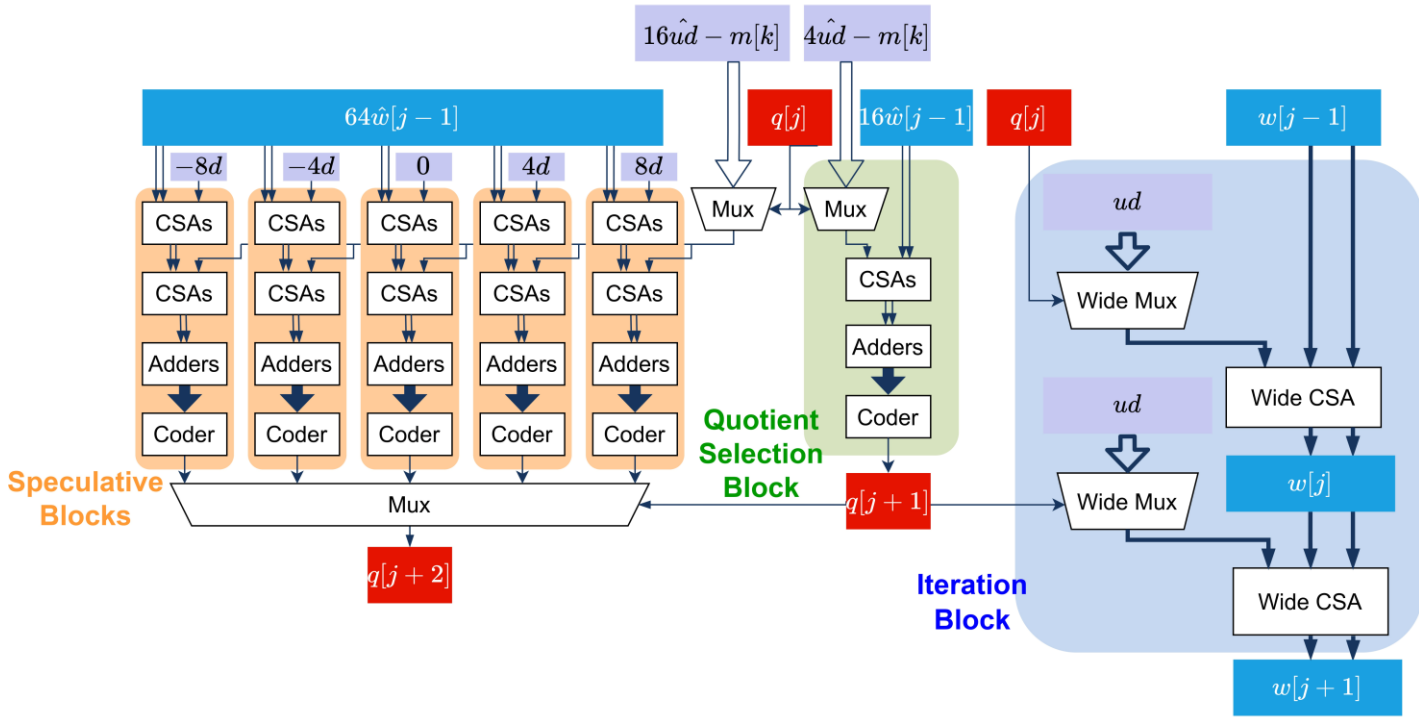


https://github.com/OpenXiangShan/fudian

FPU

- **Cascade FMA**
  - Reduce FADD delay 5->3
- **SRT16 Division units**
  - ½ delay compared with Yanqihu (SRT4)





B/K Extension Implemented

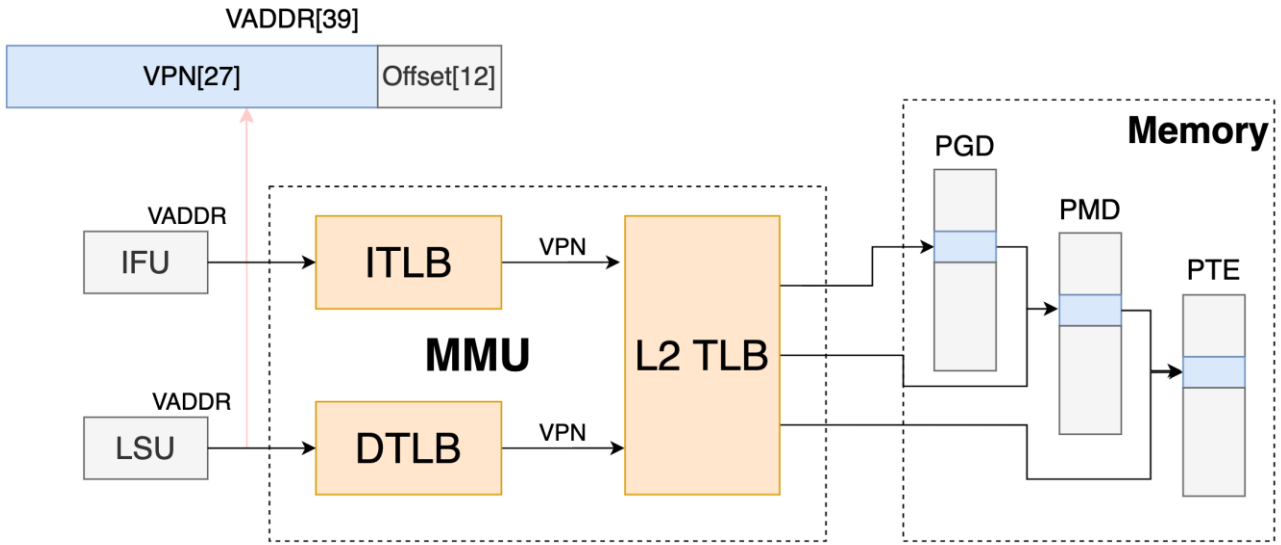
- **B**: Bit-Manipulation extension
- **K**: Cryptographic extension
- * **V**(Vector)/**H**(Hypervisor) extensions are in development!

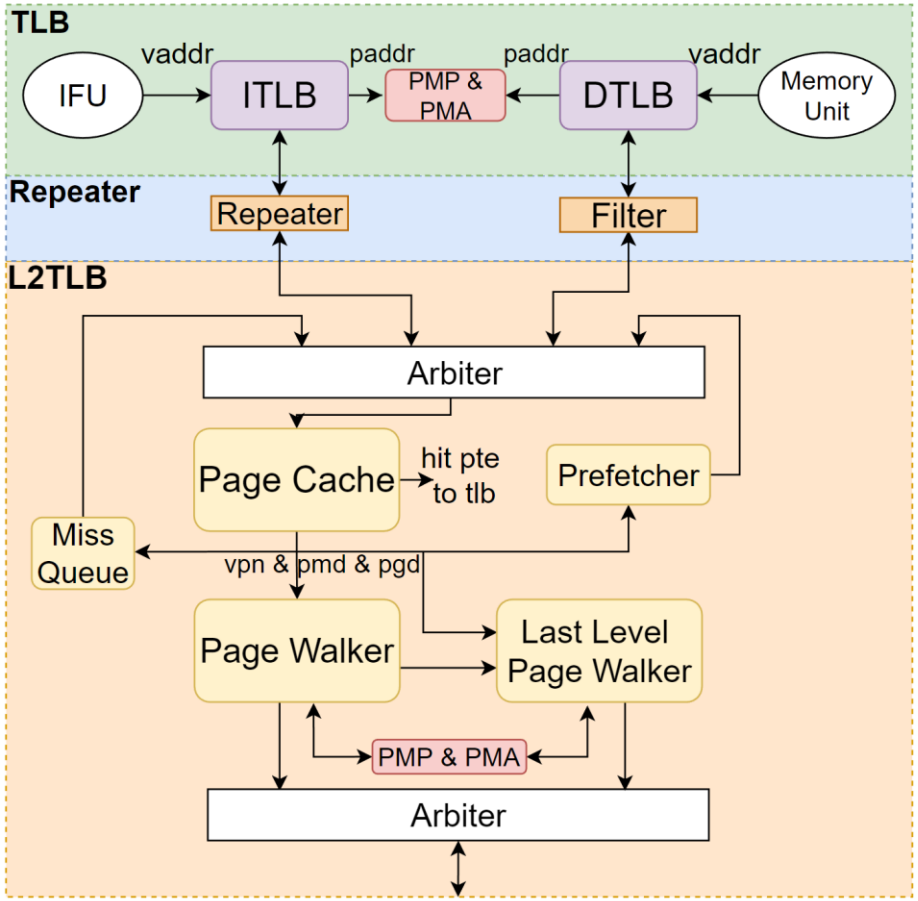
B Extension

K Extension

Memory Management Unit

- Translate from virtual address to physical address
- Privilege checking
- Support Sv39

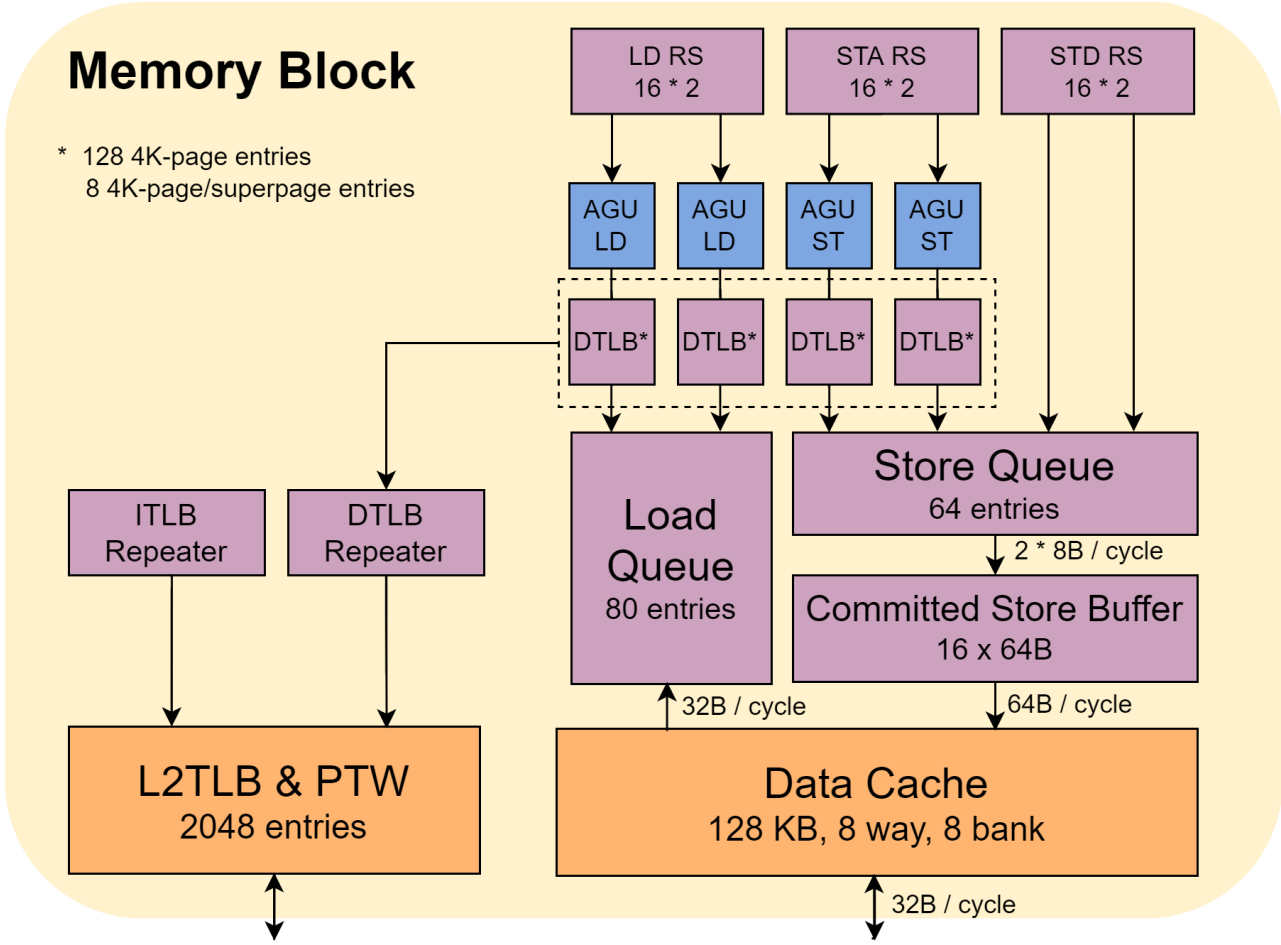




Memory Block

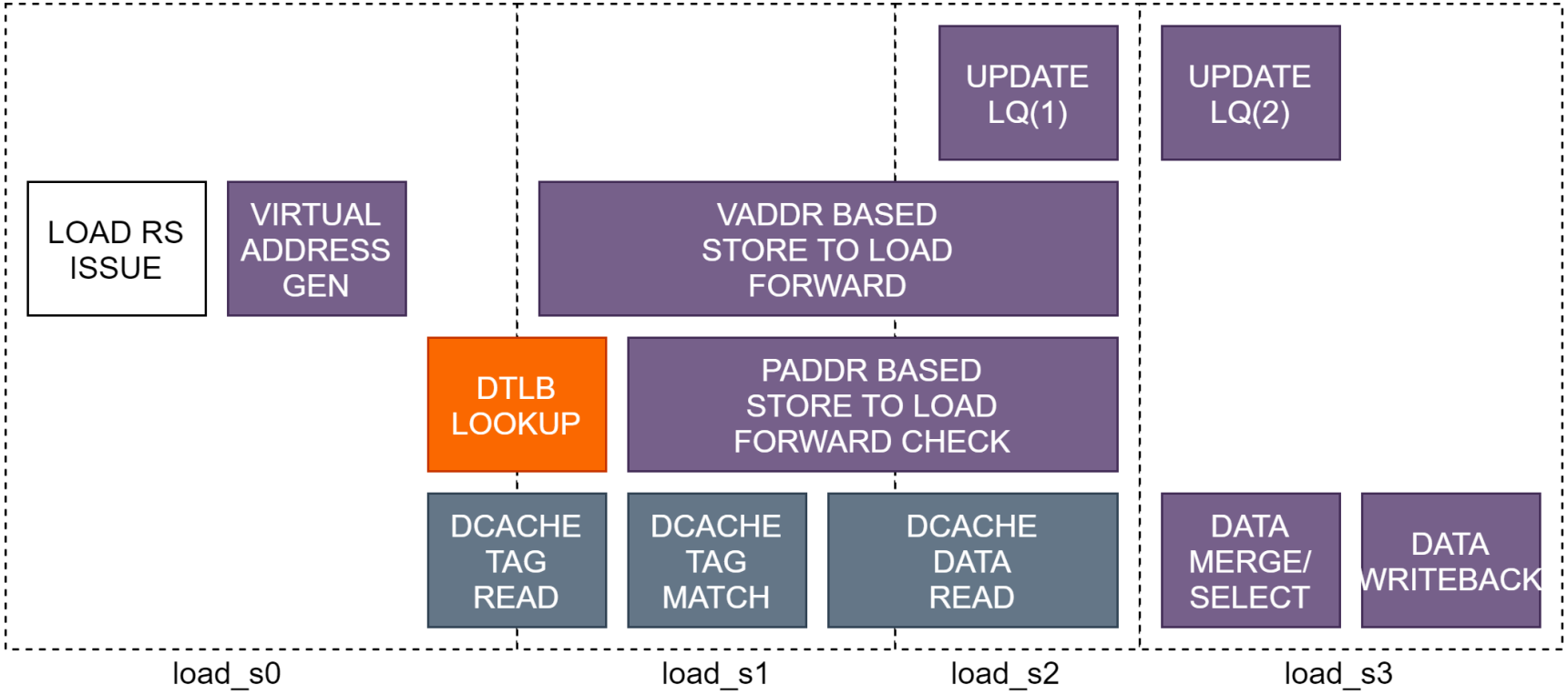
- Datasheets

Load pipeline	2
Store address pipeline	2
Store data pipeline	2
L1 LD hit latency	4
L1 LD hit bandwidth	2x8B/cycle
Store data bandwidth	2x8B/cycle
Load Queue Entry	80
Store Queue Entry	64
Merged Store Buffer	16 cachelines



Load Pipeline

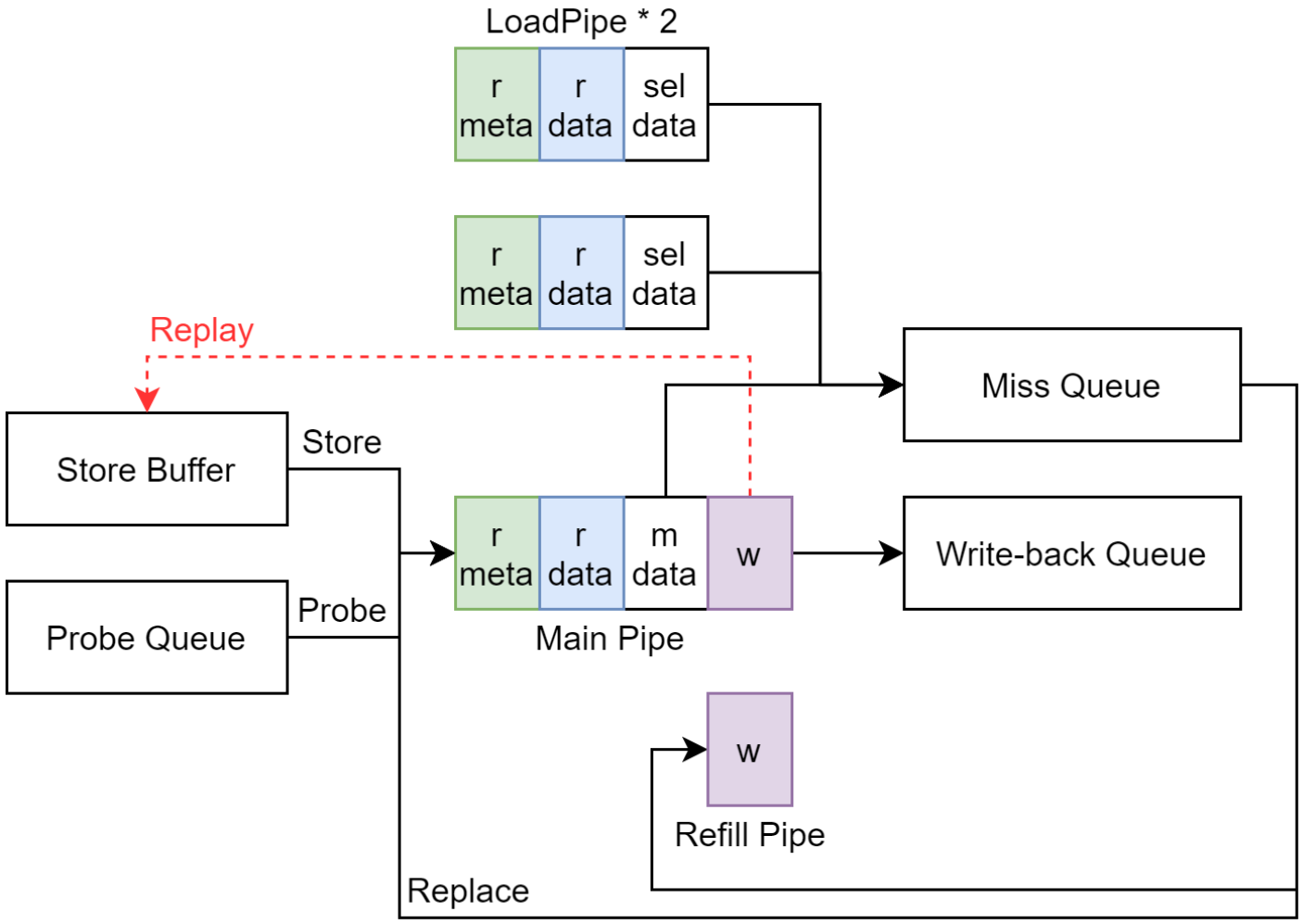
- Four stage
- Optimized for load-to-load & load-store bypass



L1 DCache

- Coupled with load pipeline
- TileLink protocol

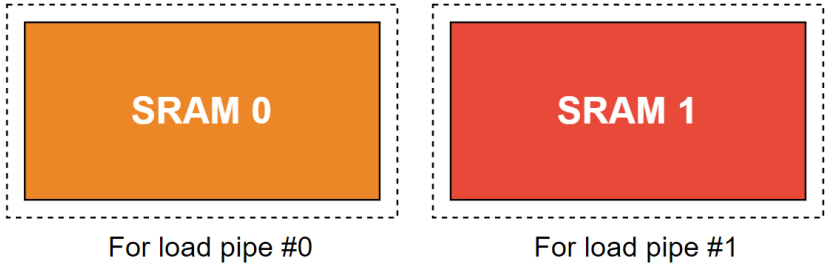
L1 Data cache size	**64KB**
L1/L2 Bus Width	256bit
Store Buffer	16
Probe Queue	8
Miss Queue	16
Write-back Queue	18
Prefetch Instruction	**Support**
ECC	**Support**



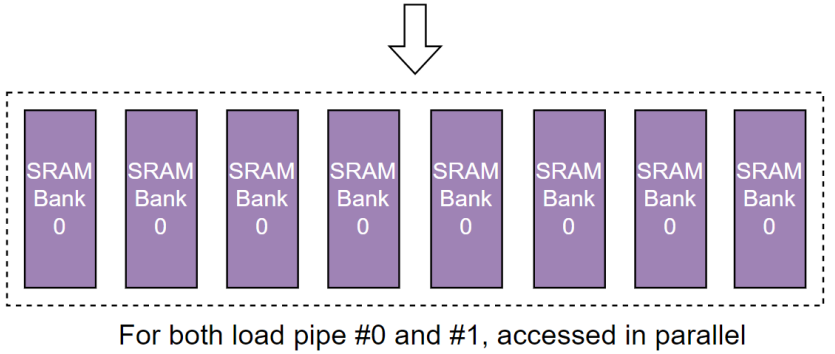
L1 DCache

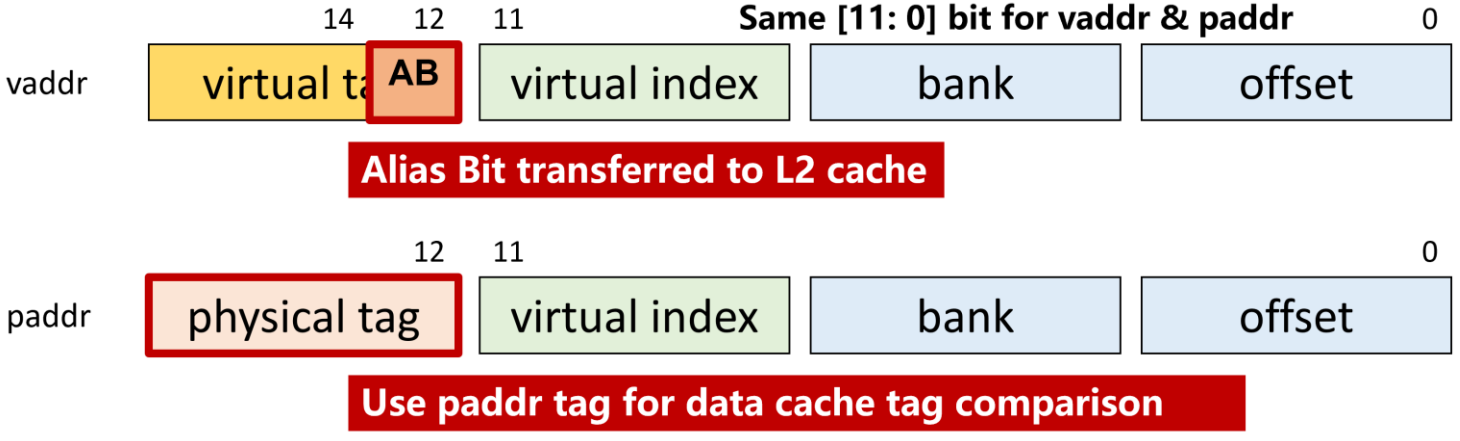
- Noteworthy features
  - Multi-bank
  - Anti-alias
  - MSHR request merge
  - Optional L1D prefetcher

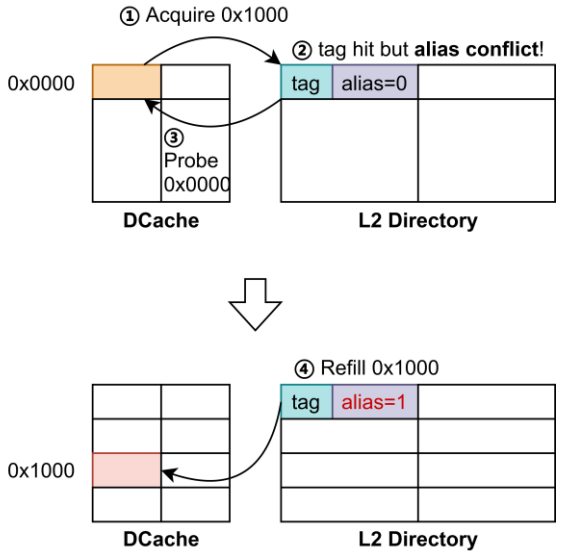
Yanqihu



Nanhu

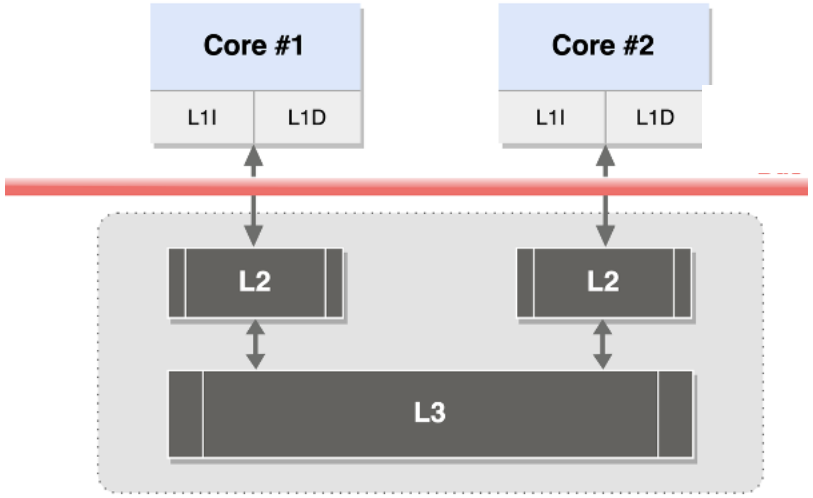




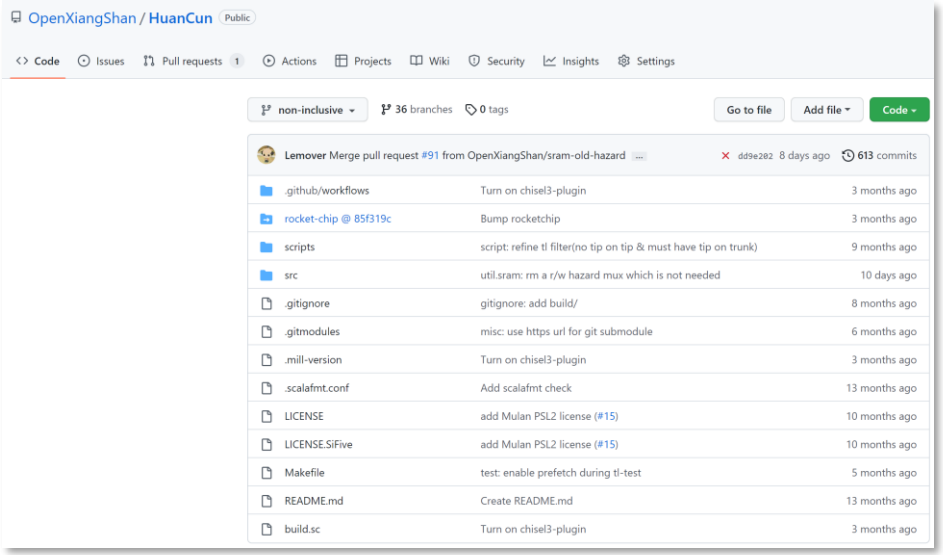


L2/L3 Cache

- 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**
- Maintain coherence with **both** L1I & L1D



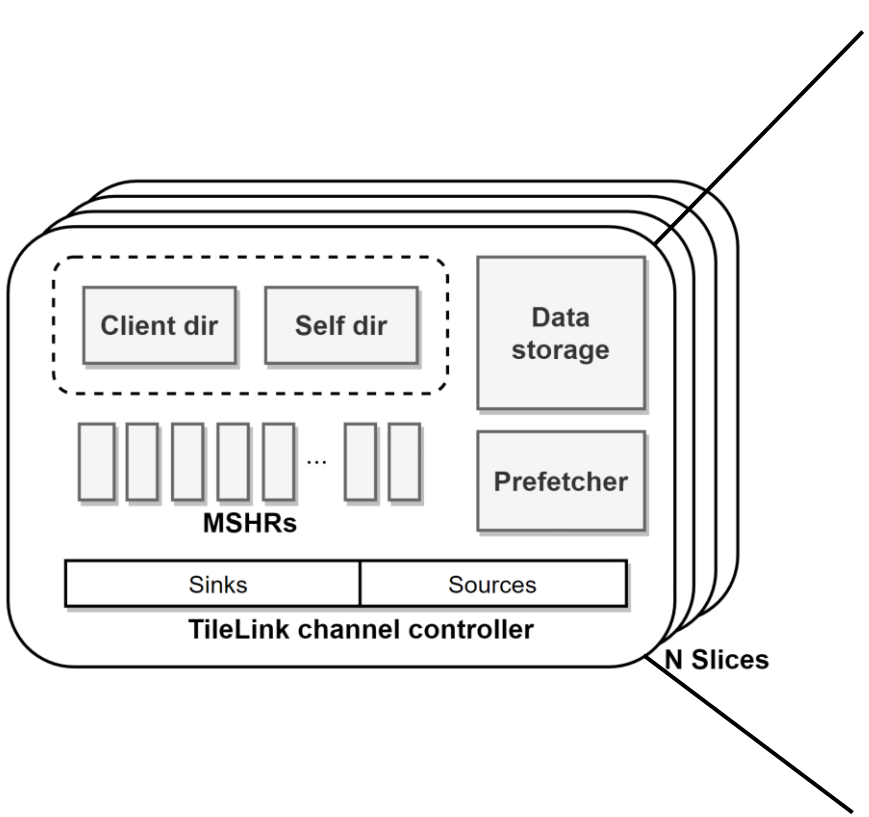
Nanhu Dual-core

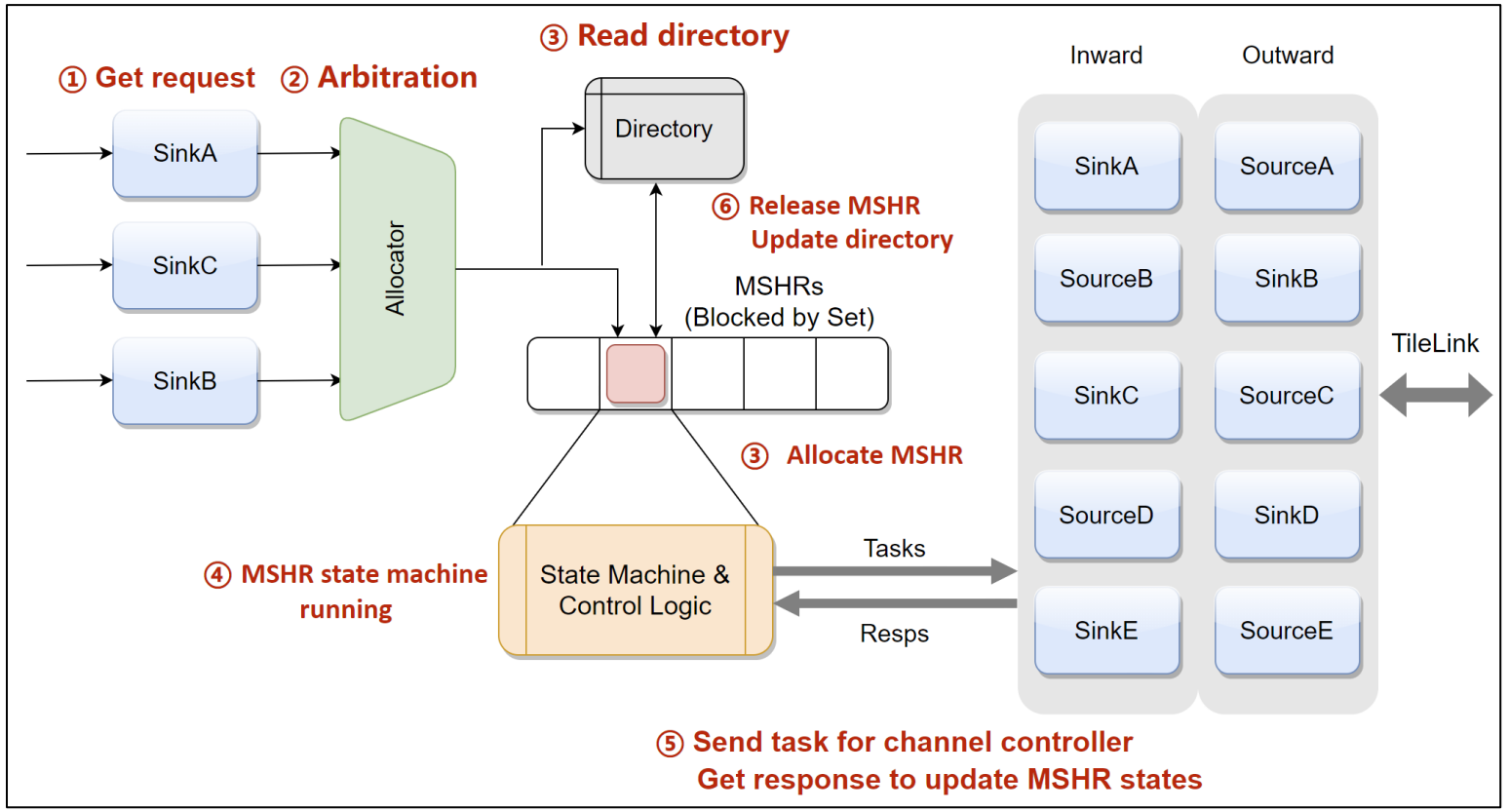


https://github.com/OpenXiangShan/HuanCun

L2/L3 Cache

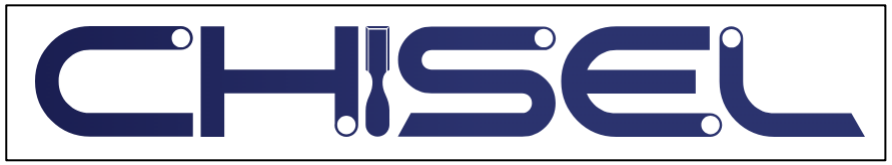
- Concurrent request processing flow





Microarchitecture Implementation

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



- Beyond built-ins, we designed some open-source high-performance utilities/components

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

Check it out!

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,
```

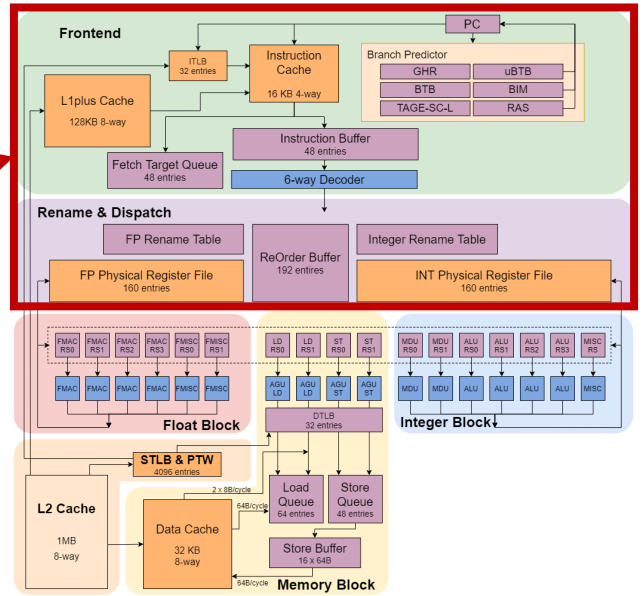
Some key parameters

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

All you need is to change a number

```
// Synthesizable minimal XiangShan
// * It is still an out-of-order, super-scalaer arch
// * L1 cache included
// * L2 cache NOT included
// * L3 cache included
class MinimalConfig(n: Int = 1) extends Config(
  new DefaultConfig(n).alter((site, here, up) => {
    case SoCParamsKey => up(SoCParamsKey).copy(
      cores = up(SoCParamsKey).cores.map(_.copy(
        DecodeWidth = 2,
        RenameWidth = 2,
        FetchWidth = 4,
        IssQueSize = 8,
        NRPhyRegs = 80,
        LoadQueueSize = 16,
        StoreQueueSize = 16,
        RoqSize = 32,
        BrqSize = 8,
        FtqSize = 16,
        IBufSize = 16,
        StoreBufferSize = 4,
        StoreBufferThreshold = 3,
        dpParams = DispatchParameters(
          IntDqSize = 8,
          FpDqSize = 8,
          LsDqSize = 8,
```

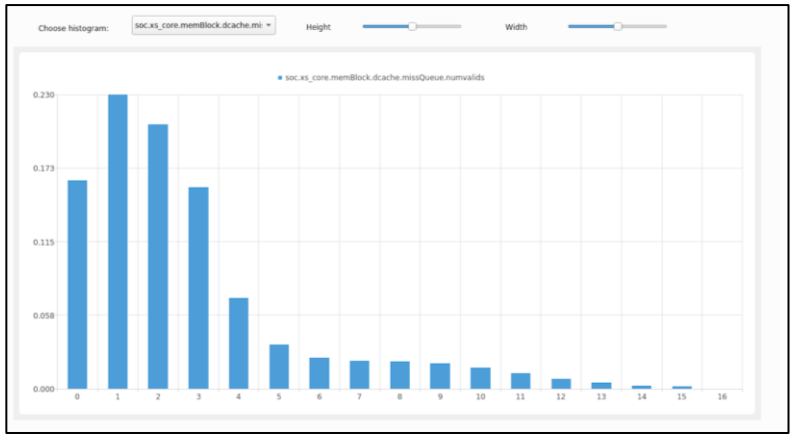
All affected
But it works

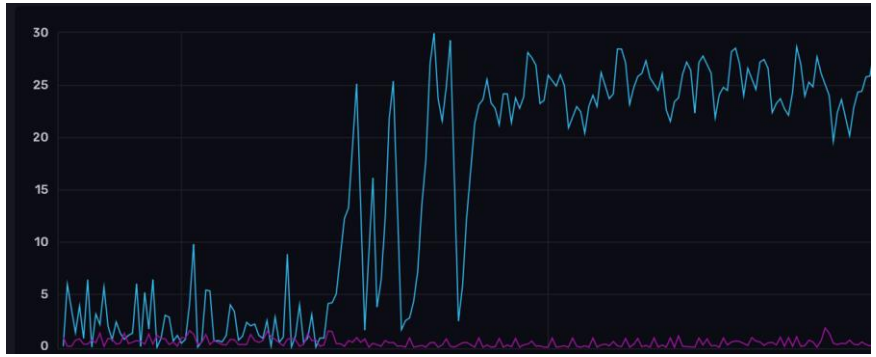


Microarchitecture Implementation

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

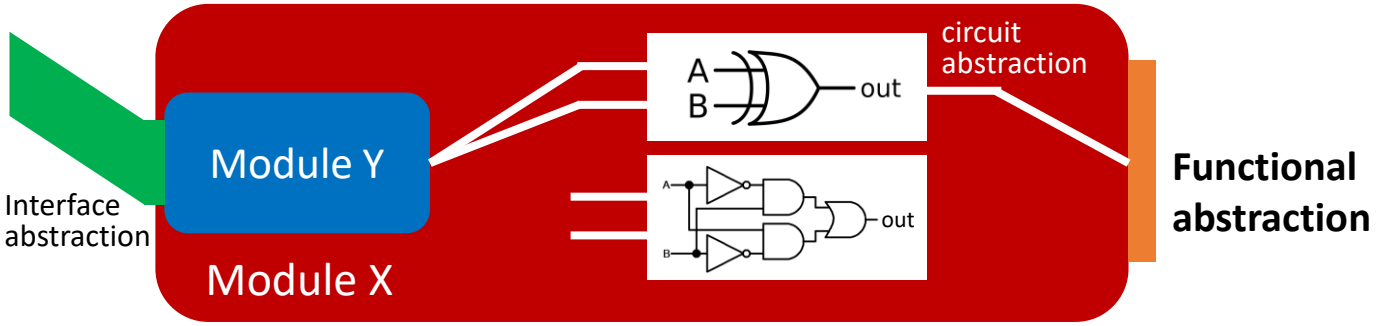
```
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpBInstr,              57899
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpBRight,              54695
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpBWrong,               3204
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: BpCRight,               3615
TOP.XSSimSoC.soc.xs_core.ctrlBlock.ftq: 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
```





Microarchitecture Implementation

- Modules are designed according to object-oriented programming
- Support multi-level abstraction



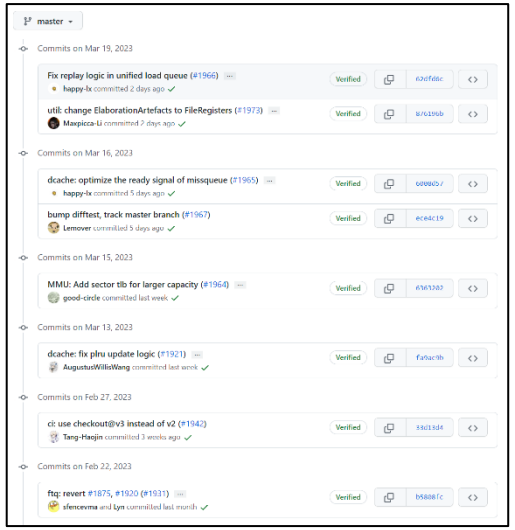
- e.g., to implement a new prefetcher
  - We provide a clean set of interfaces that allow the developer to focus on the implementation of the algorithm itself

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

Estimated Performance of Kuminghu

- **Estimated SPECint 2006 40.95, SPECfp 2006 43.96@3GHz**
  - Compile with GCC 12.0, –O3, RV64GCB
  - SPEC CPU2006 checkpoints selected by Simpoint
  - RTL simulation, 1MB L2 + 16MB L3, DDR4-3200MHz under DRAMsim3 (66 cycle access padding)

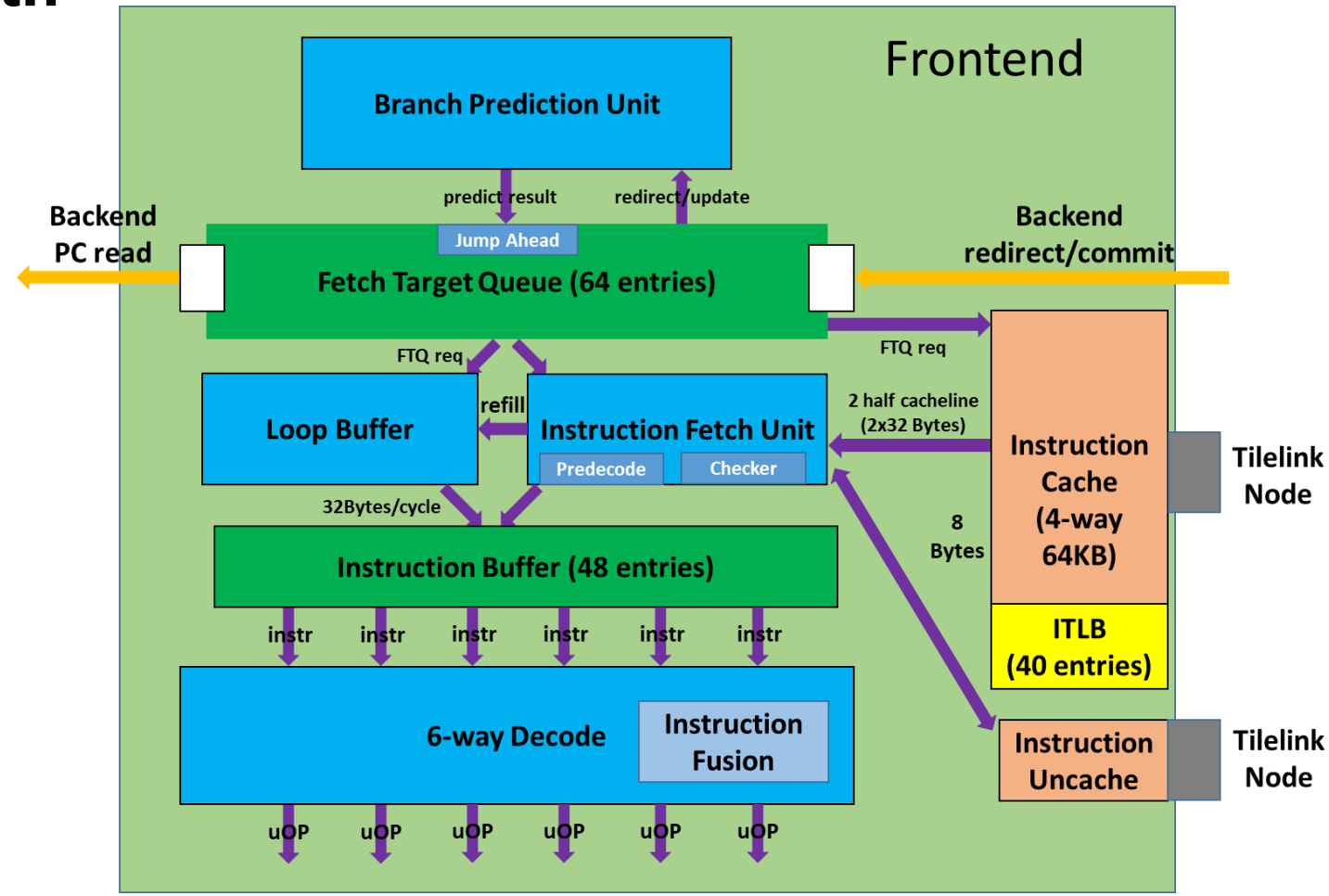
SPECint 2006	
400.perlbench	37.587
401.bzip2	23.673
403.gcc	46.215
429.mcf	53.832
445.gobmk	29.61
456.hmmer	40.236
458.sjeng	30.219
462.libquantum	123.249
464.h264ref	57.426
471.omnetpp	32.748
473.astar	28.965
483.xalancbmk	41.511
SPECint2006@3GHz	40.9462

SPECfp 2006	
410.bwaves	76.941
416.gamess	44.178
433.milc	33.402
434.zeusmp	45.684
435.gromacs	31.035
436.cactusADM	47.286
437.leslie3d	42.138
444.namd	38.427
447.dealII	57.177
450.soplex	55.869
453.povray	58.761
454.Calculix	15.135
459.GemsFDTD	37.728
465.tonto	33.228
470.lbm	91.608
481.wrf	37.389
482.sphinx3	56.82
SPECfp2006@3GHz	43.9578

More feature to be merged yet

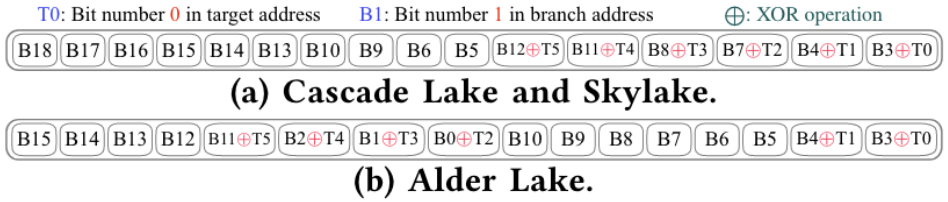
① Frontend Upgrade

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



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**

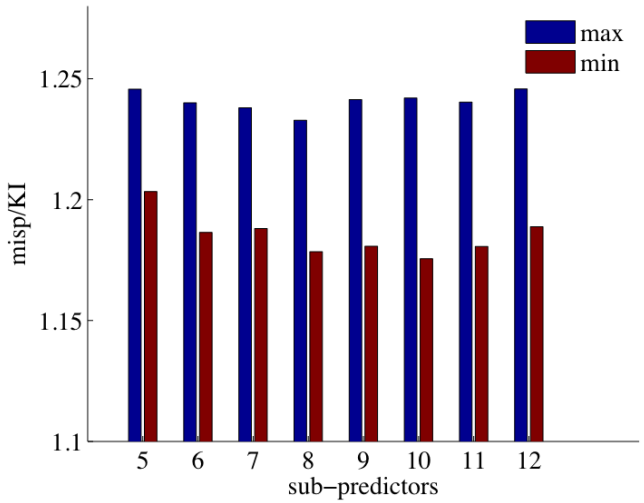


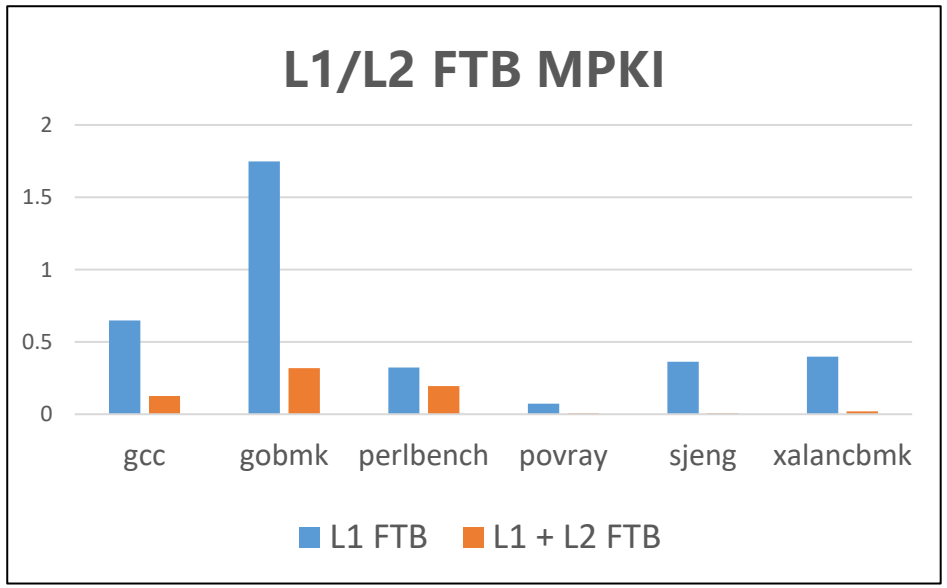
Figure 2: Performance of random parameters

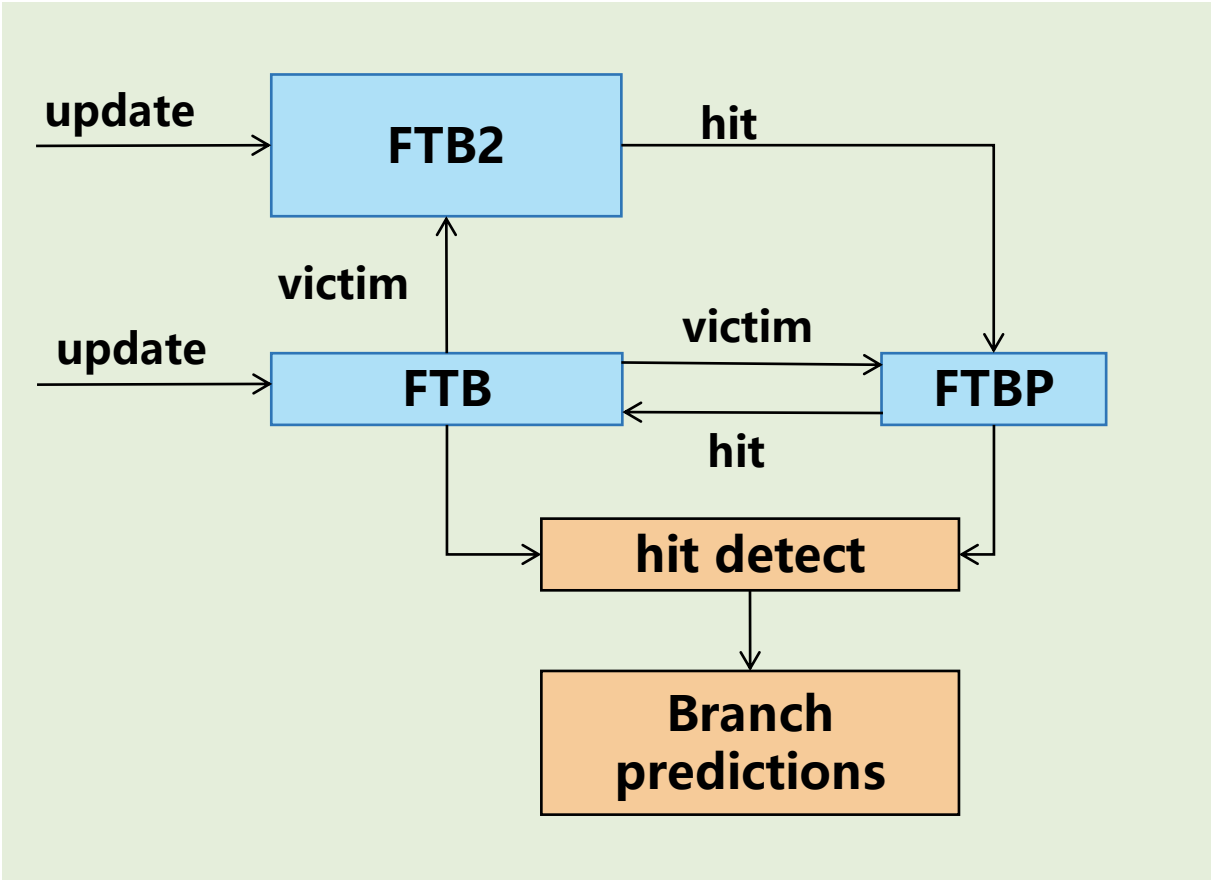
Previous studies have shown that the #tables and history length have a significant effect on TAGE performance, up to 5.64% in MPKI [1]

[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

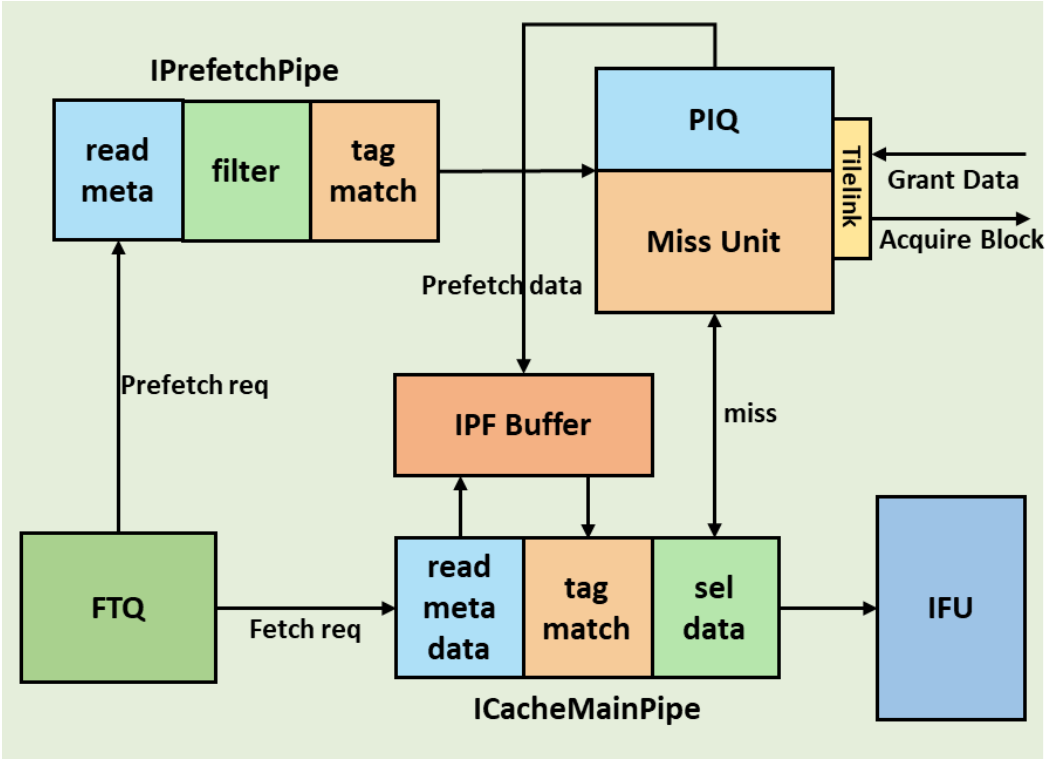
Feature 3: Smart ICache Prefetcher

- Based decoupled front-end, FDIP prefetching algorithm is implemented

Prefetcher workflow

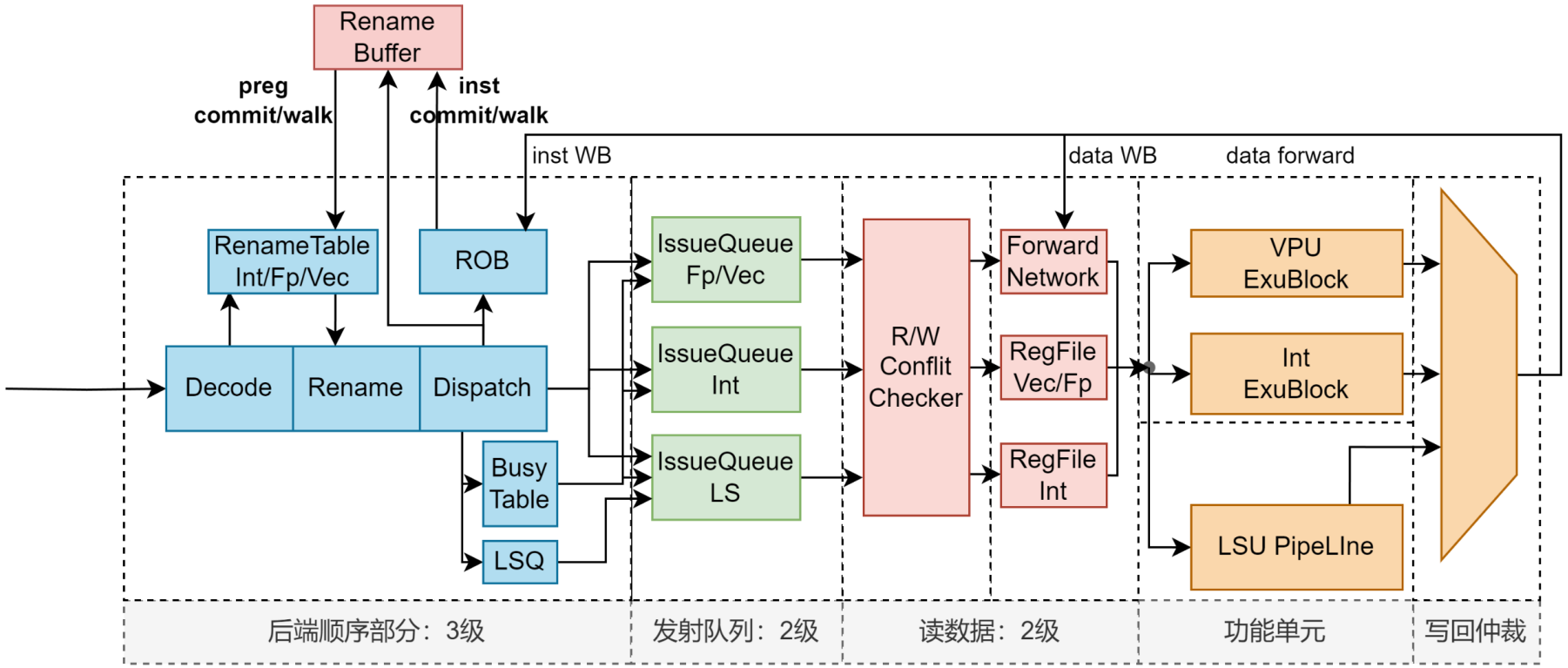
1. Select prefetch request from PF-queue
2. Write refilled data into prefetch buffer
3. Query icache & prefetch buffer parallelly

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



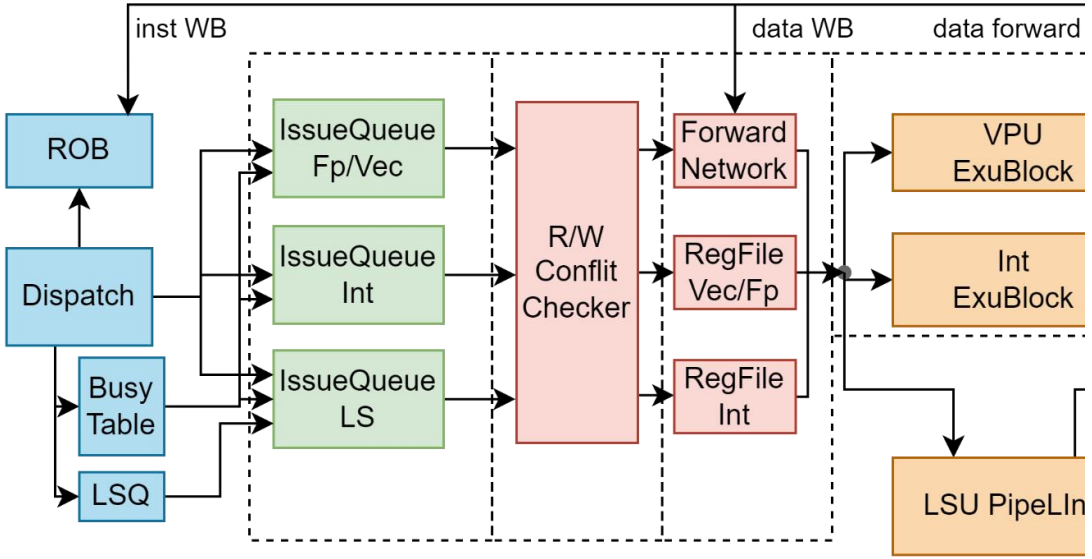
② OoO Backend Upgrade

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



Feature 1: Read regfile after issue

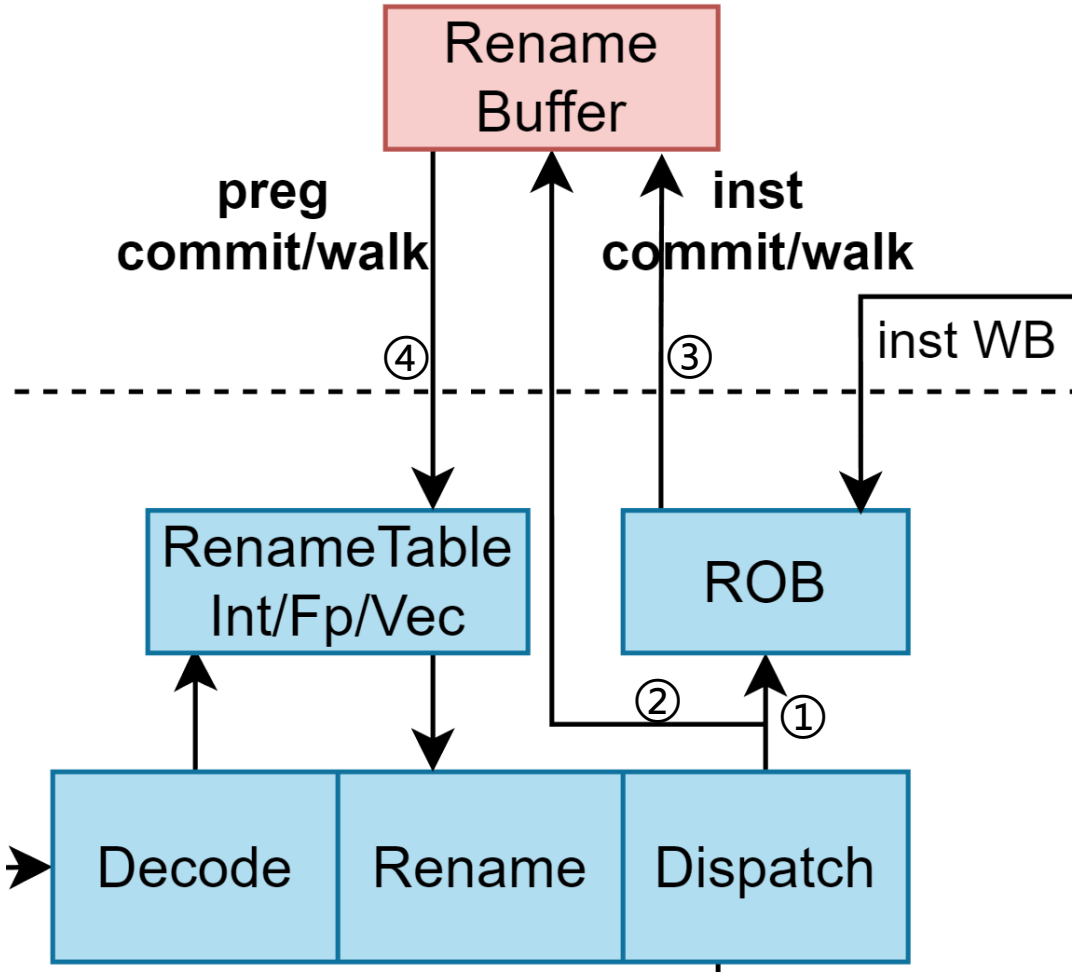
- Advantage
  - Bottleneck is shifted back to µop issue
  - Reduce RS storage to save SRAM
  - Reduce latency and increase RS capacity
- Optimization
  - Efficient arbitration of regfile reading ports
  - Accurate speculative wakeup and cancelling



Read regfile after issue

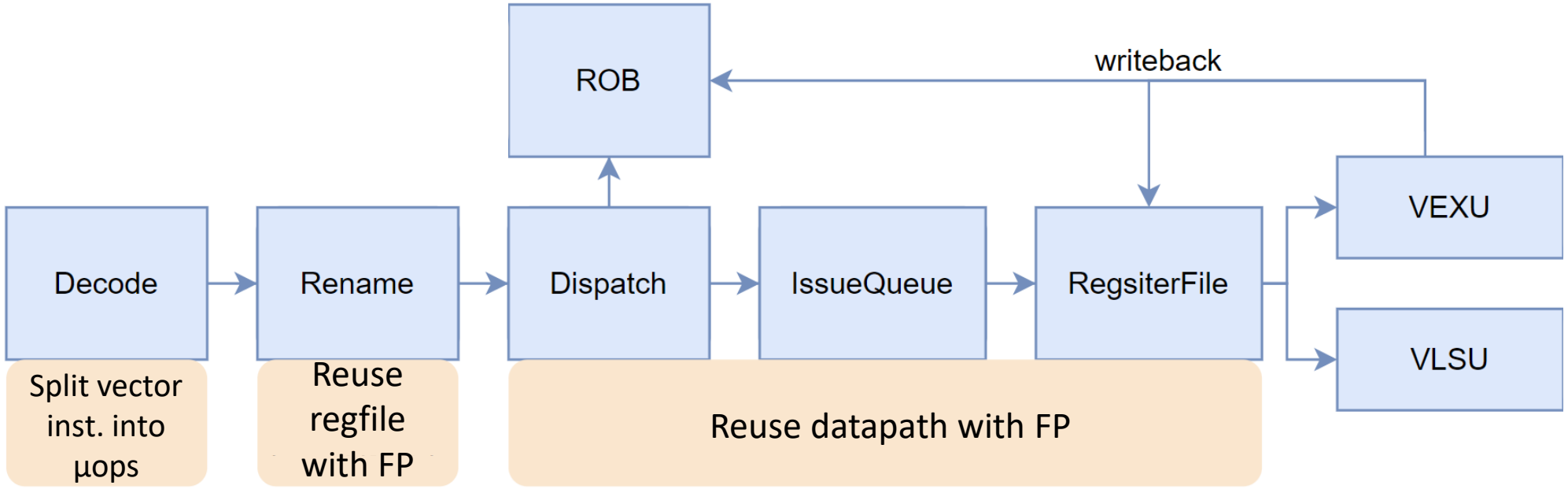
Feature 2: Decouple µop commit & rename-table update

- New Rename Buffer (**RAB**) design: record µops that have been committed but not updated the RAT
  - 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
  - Reduce ROB entry and optimize timing
  - Support µop split of vector instructions
  - Support ROB compression



Feature 3： Support Vector Extension

- Decoupled 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

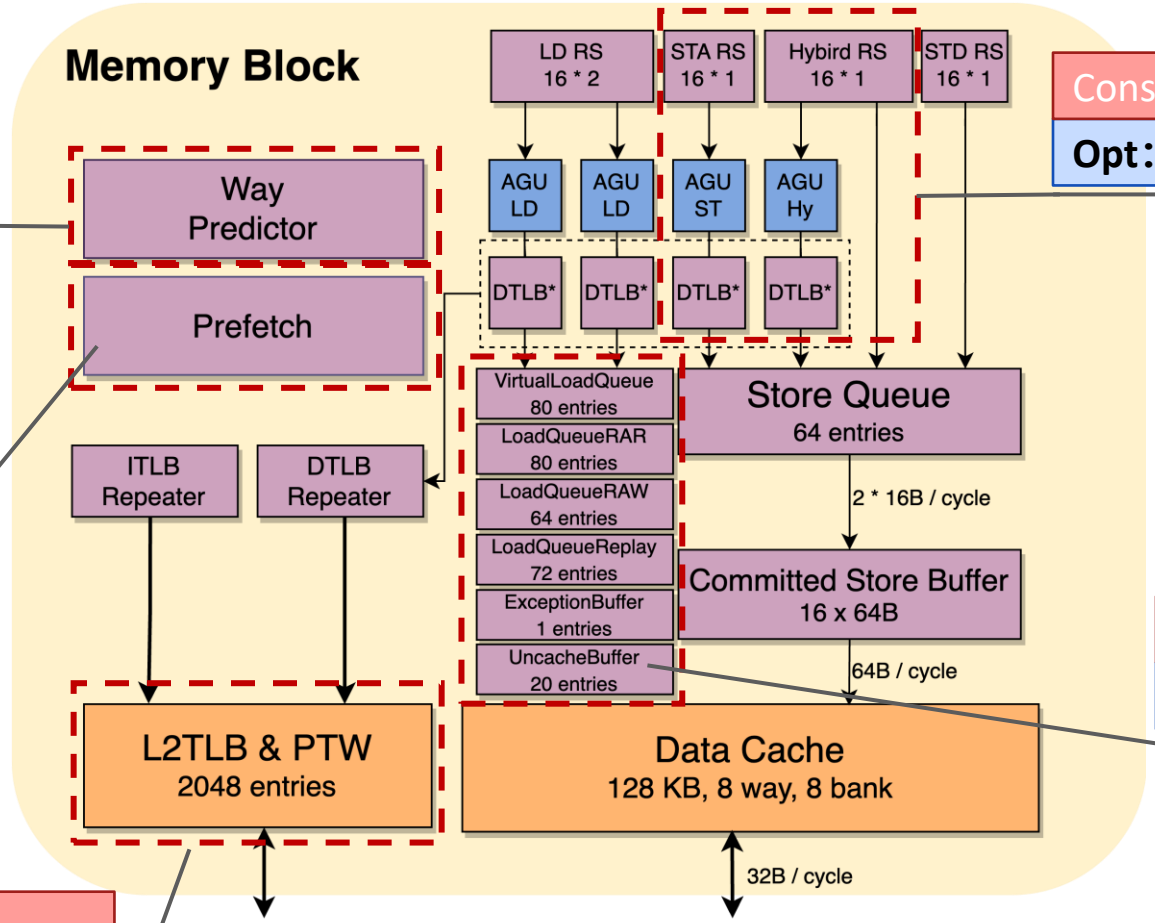
Opt： Support way prediction

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

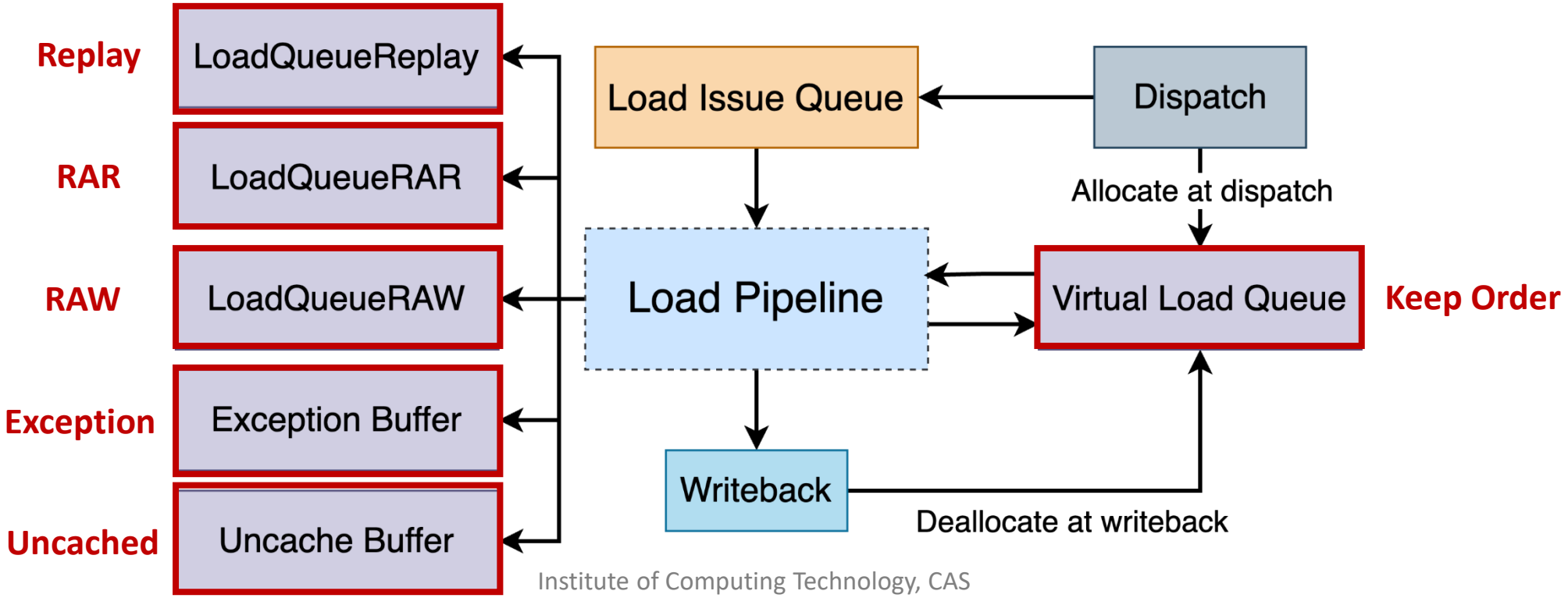
Opt： Support hybrid ld/st unit

Cons: bad timing in load queue

Opt： Split load queue

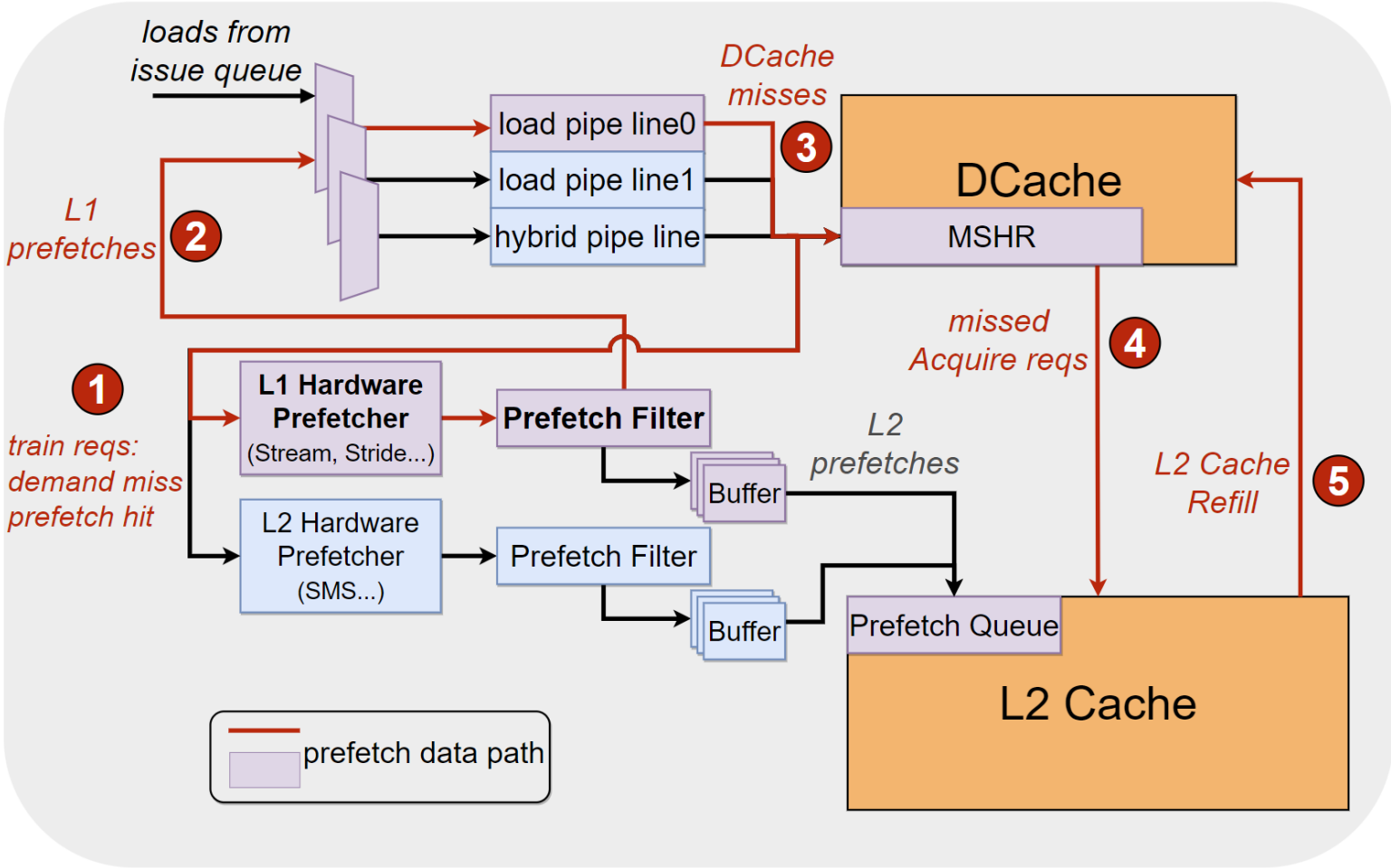
Feature 1: Split Load Queue

- 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...)

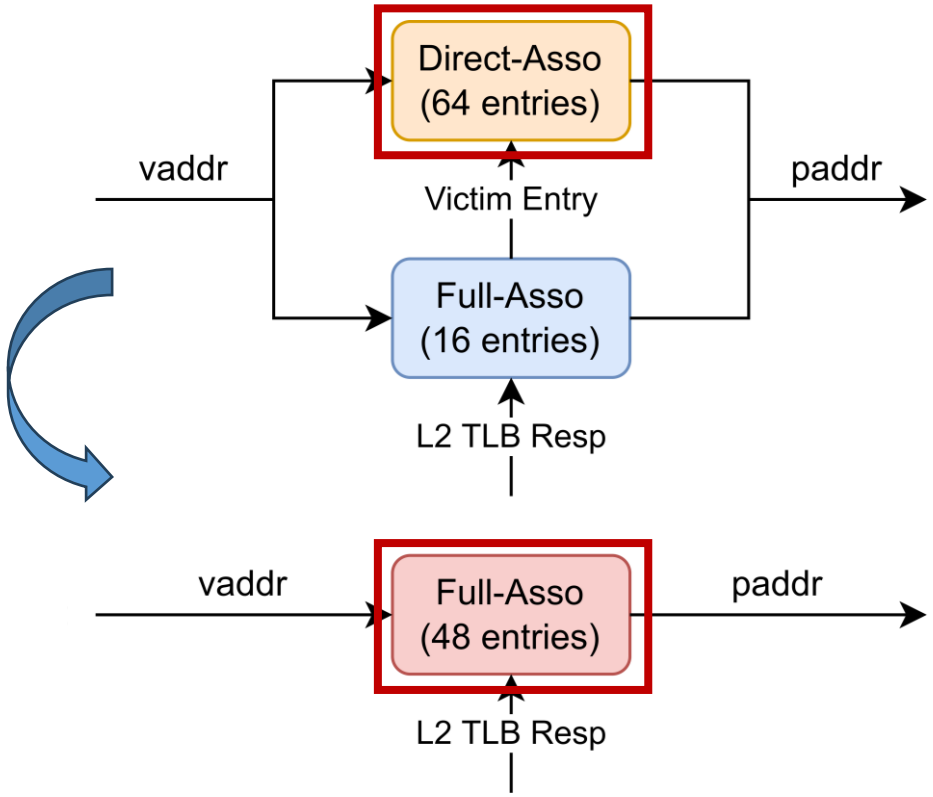


[1] S. Srinath, O. Mutlu, H. Kim and Y. N. Patt, "Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers," 2007 IEEE 13th International Symposium on High Performance Computer Architecture, Scottsdale, AZ, USA, 2007

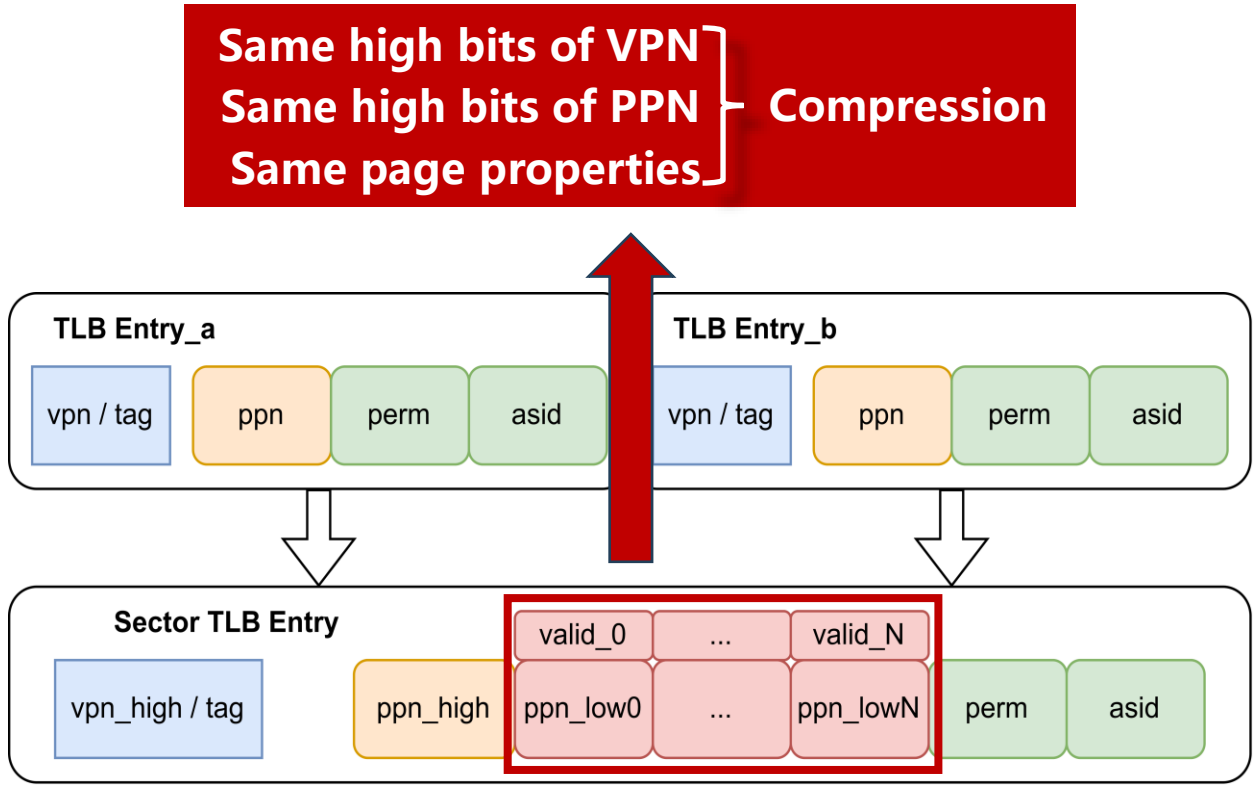
[2] J. -L. Baer and T. -F. Chen, "An effective on-chip preloading scheme to reduce data access penalty," Supercomputing '91:Proceedings of the 1991 ACM/IEEE Conference on Supercomputing, Albuquerque, NM, USA, 1991

Feature 3: Powerful MMU

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



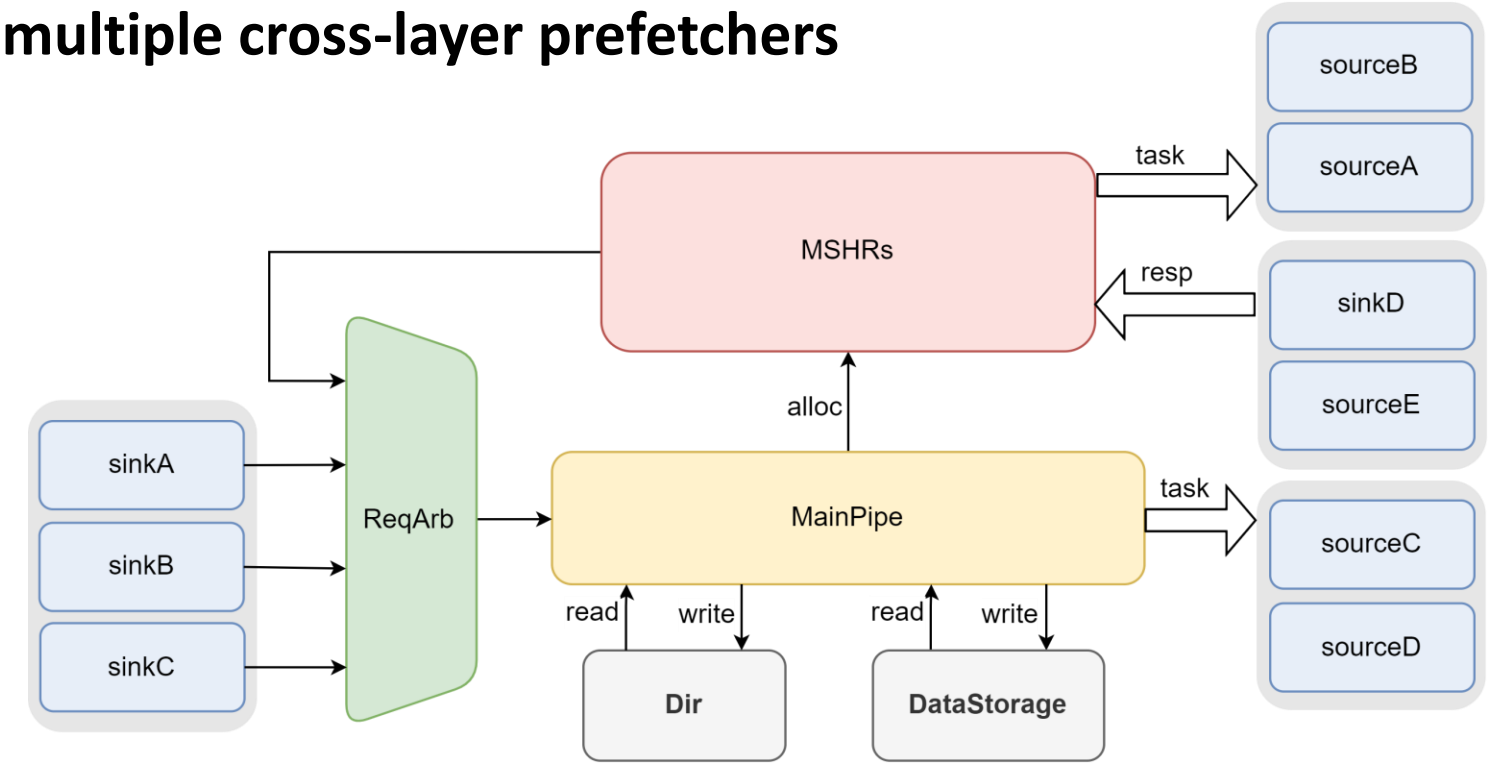
DTLB organization in Nanhu and Kunminghu



TLB compression

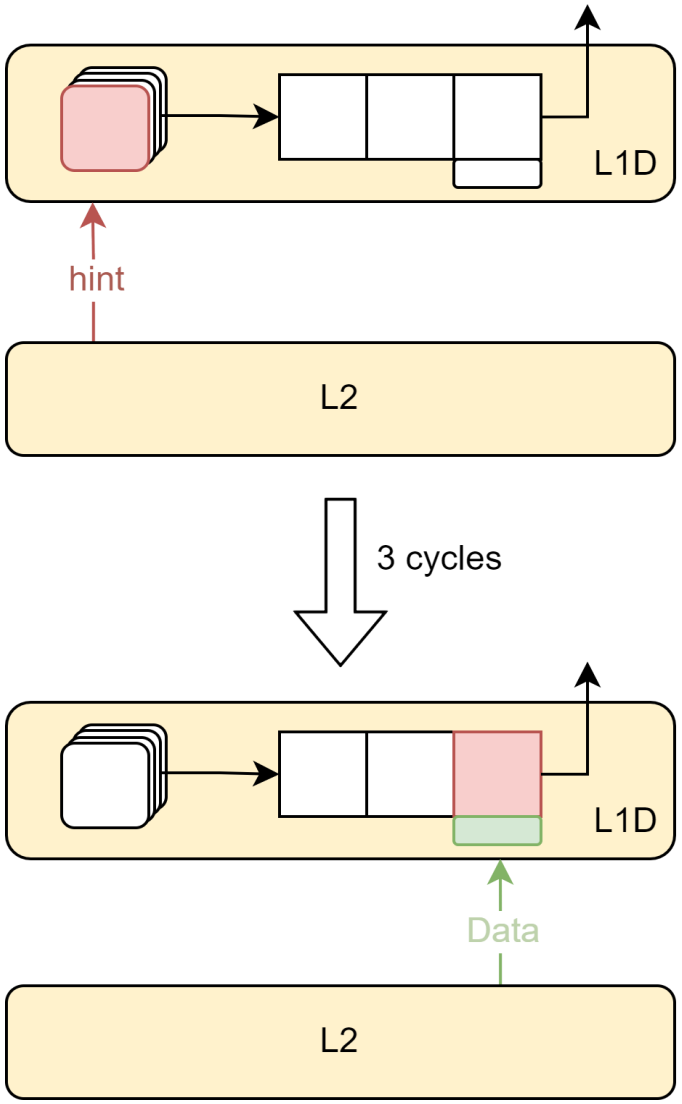
④ L2 Cache Upgrade

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



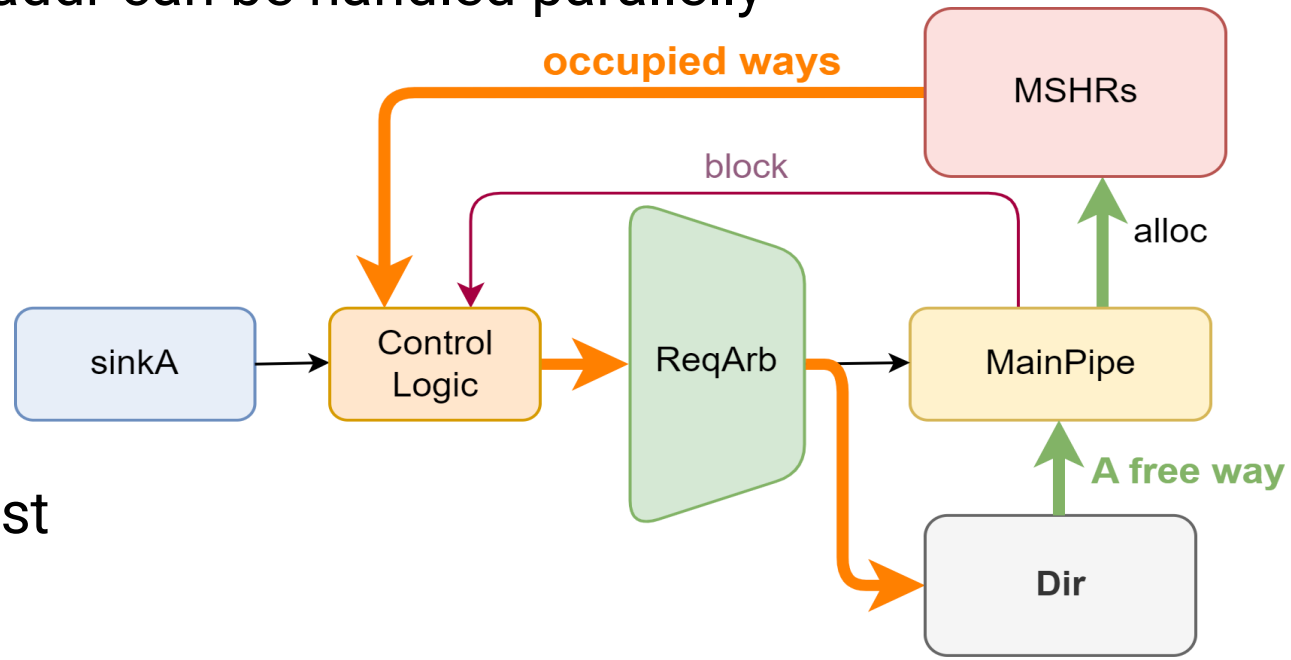
Feature 1: L1-L2 Collaborative optimization

- **Observation:**
  - There's chance to hint LSU to replay before refill
  - Interval between hint and refill signal is 3 cycles
- **New Design:**
  - Let L2 give the 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



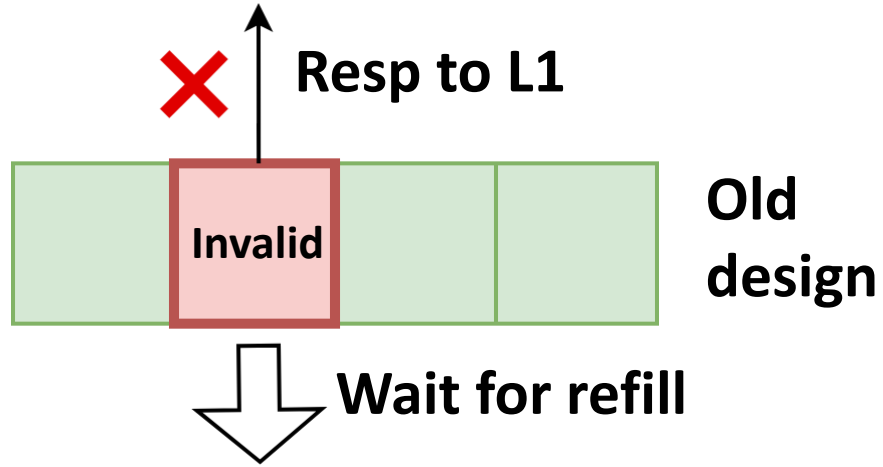
Feature 2: Eliminate txns serialization in the same set

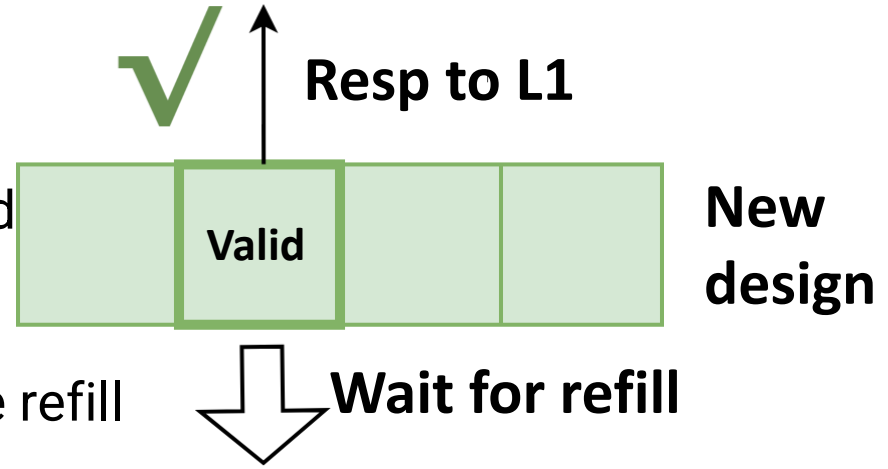
- **Background:** Txns to the same set were handled serially, which reduces parallelism
- **Difficulty:** Txns to the same set can affect each other due to replacement
- **New Design**
  - Txns to the same set but different addr can be handled parallelly
  - Block txns to same addr only
- **Implementation**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



Feature 3: Evict on Refill

- **Observation:**
  - On L1 acquire, L2 assigns one way by replacement policy
  - But this way is occupied until refill, unable to serve L1
- **Improvement:**
  - New design makes the way be assigned until 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





Thanks!

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