



Why You Should Choose XiangShan

- Industry-competitive, high-performance, fully-verified and open-source processor
- First-tier performance in RISC-V
- Industrial-grade verification flow including UT, IT, ST + FPGA
- The **only** open-source implementation of RISC-V RVA23 profile
 - An ideal choice for **Vector** and **Hypervisor** research

XiangShan Microarchitecture Design Philosophy

The XiangShan Team

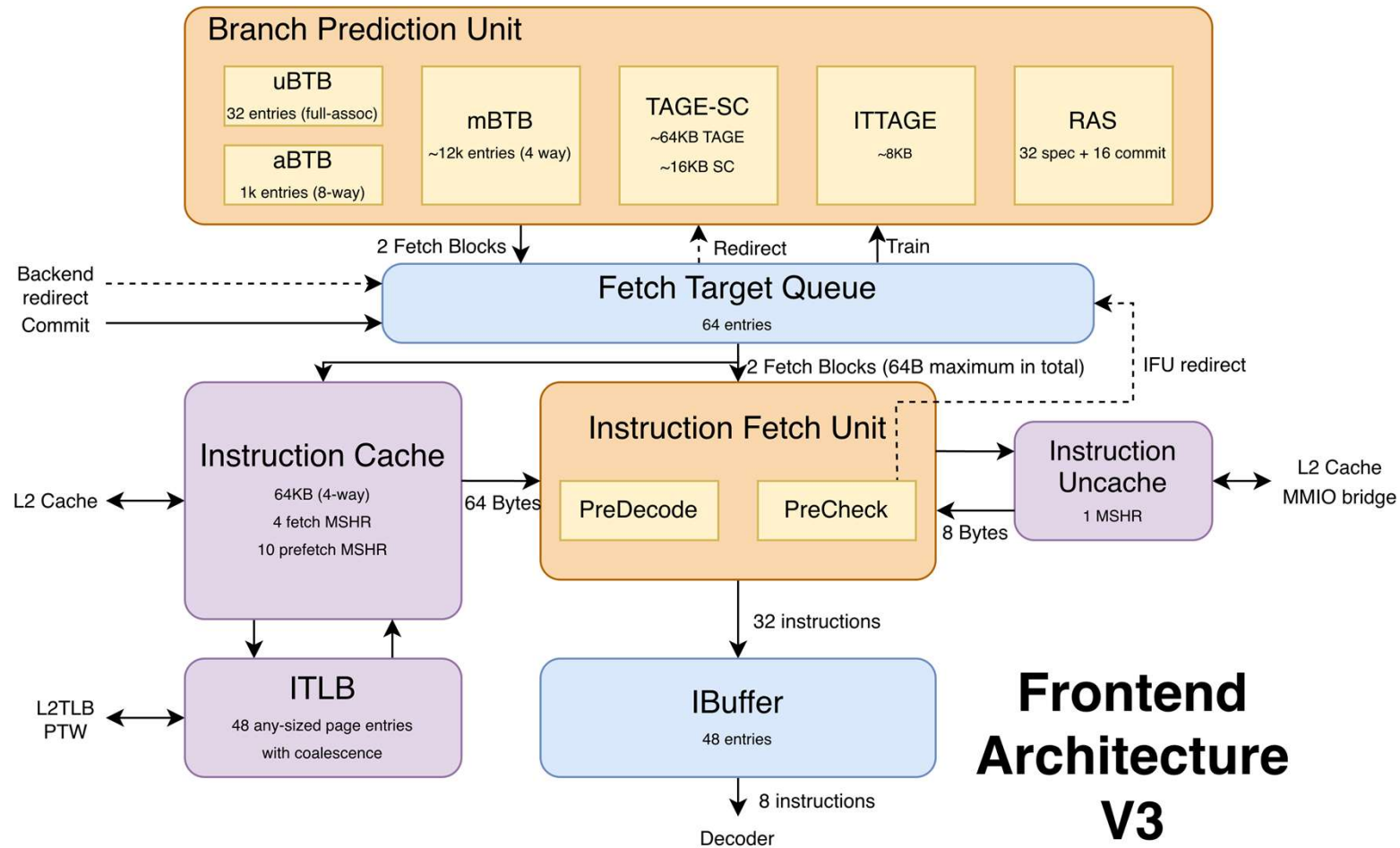
XiangShan Roadmap

- 1st generation: 雁栖湖 (Yanqihu)
 - 2020/6: first commit of RTL design
 - 2021/7: 28nm tape-out, 1.3GHz
 - SPEC CPU2006 7.01@1GHz, DDR4-1600
- 2nd generation: 南湖 (Nanhu)
 - 2021/5: performance exploration and RTL design
 - 2023/4: GDSII delivery
 - 14-nm tape-out
 - SPEC CPU2006 20@2GHz

XiangShan Roadmap

- 3rd generation: 昆明湖 (Kunminghu)
 - ISA feature: Vector (V) extension, Hypervisor (H) extension
 - Support RVA23 profile and L2 Cache with CHI protocol
 - Estimated SPEC CPU2006 45@3GHz
- Future: Kunminghu V3
 - Targeted at SPEC 2006 22/GHz
 - Extended multi core support

Frontend Architecture



Frontend Architecture V3

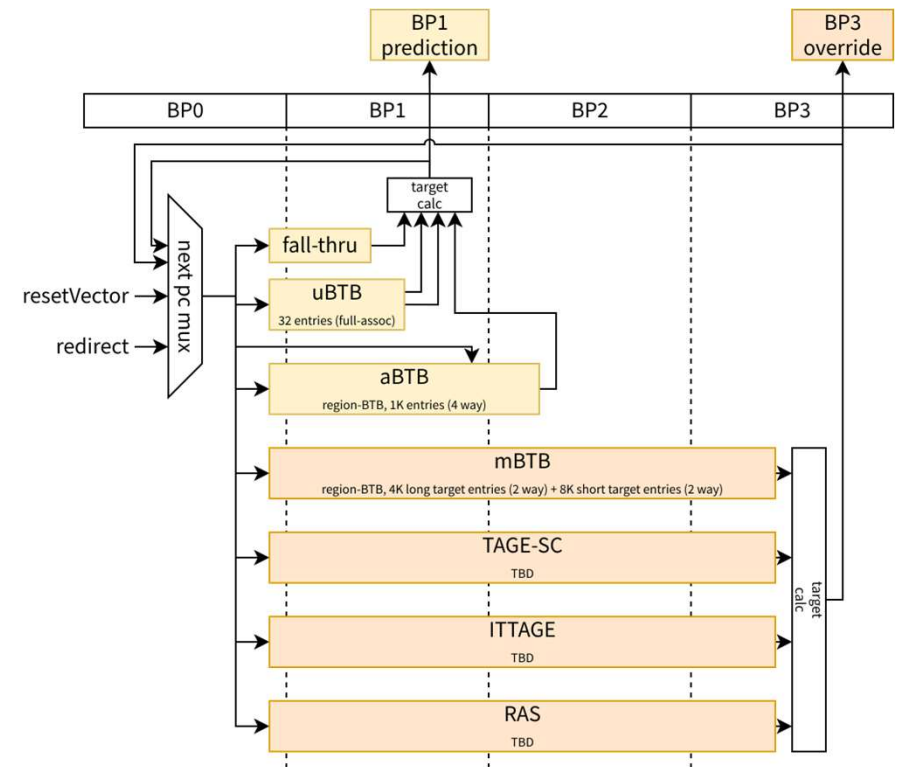
Frontend Design Philosophy

- Primary goal: provide enough instructions for 8-wide backend
- We need accurate BP and high bandwidth IF
- Decoupled frontend
- 64B fetch block, maximum 32 instructions
- BTB structure, maximum 8 branches in 64B
 - 4 branches in every aligned 32B
- Preliminary support for 2-taken/cycle and 2-fetch block/cycle



Branch Prediction Pipeline Overview

- 3-stage prediction pipeline
 - BP1: uBTB + aBTB
 - BP3: mBTB + RAS + TAGE-SC + ITTAGE
- Maximum 8 branches in 64B
- Interleaved banking
- Limited 2-taken support
- Removed cross-page predictions



❖ Preliminary 2-Taken Support

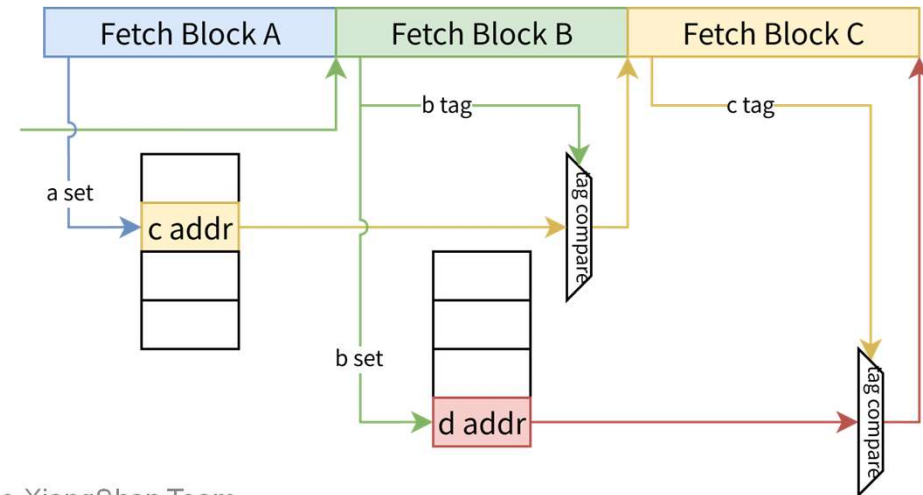
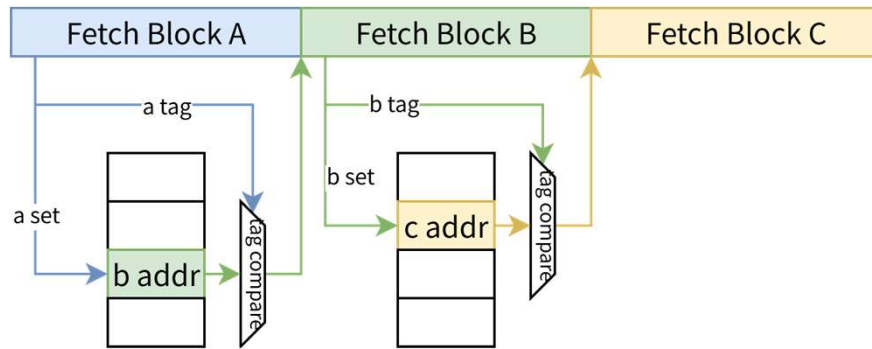
- Gain substantial performance benefit with minimal extra area
- Only uBTB will generate 2-taken predictions under some certain conditions
- First branch: static target
 - Same target indirect jump
 - Direct jump
 - Branch
- Second branch: no conflict
 - Branch: TAGE-SC read conflict
 - Indirect: ITTAGE read conflict
 - Consecutive call: RAS read conflict

BR1 \ BR2	Branch	Call	Other direct jumps	Ret	Other static indirect jumps	Other dynamic indirect jumps
Branch						
Stable call						
Dynamic call						
Other direct jumps						
Ret						
Other static indirect jumps						
Other dynamic indirect jumps						

🏔️ Ahead Pipelining

- Large capacity zero-bubble predictor with ahead pipelining
- 1K entries (4-way)

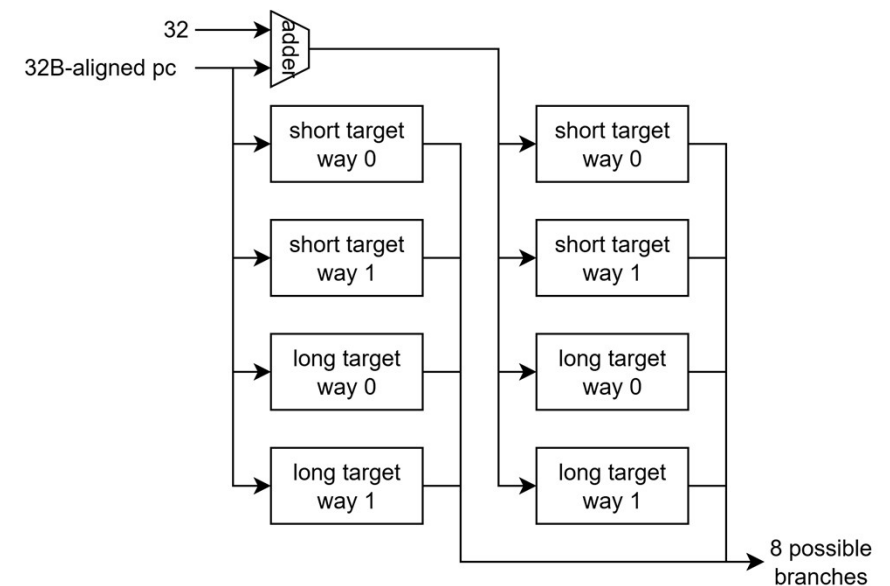
Predictor	Index	Tag	Prediction
uBTB (left)	A	A	B
aBTB (right)	A	B	C



Accurate Predictors

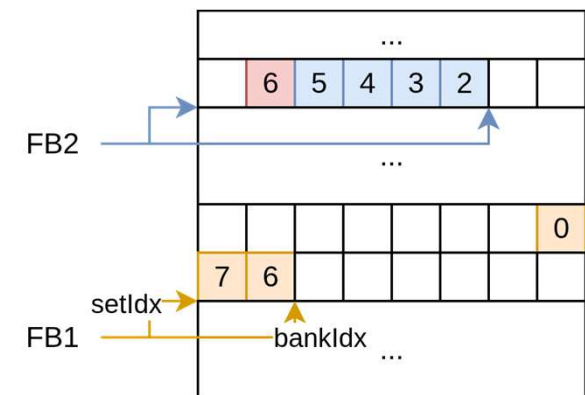
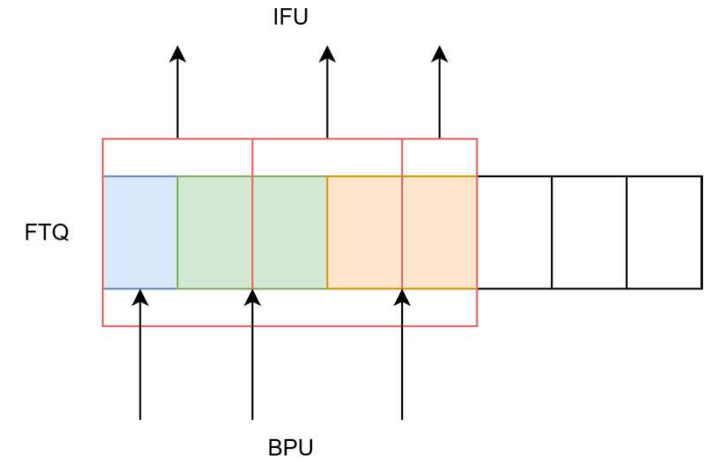
- Main-BTB:
 - 4K entries with long target, 8K entries with short target
 - Interleaved banking

- TAGE-SC & ITTAGE & RAS:
 - Standard design with some tricks
 - Inherited from Kunminghu V2
 - Modified for BTB structure
 - Parameter optimization



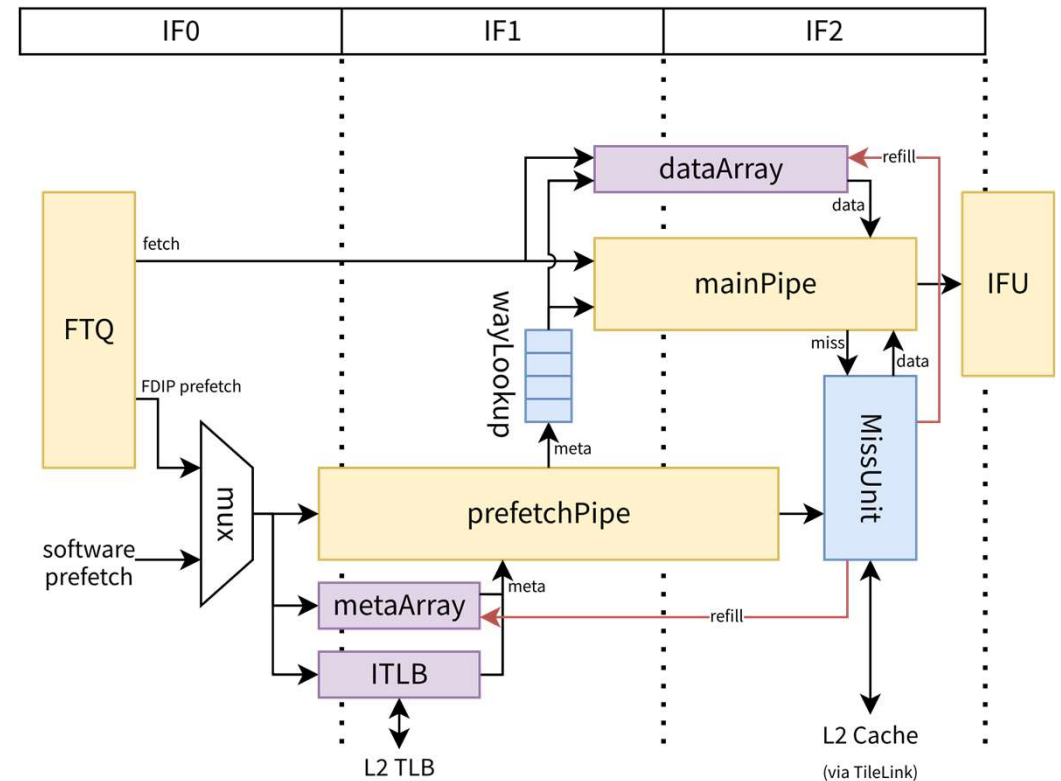
Fetch Target Queue

- Support for 2-taken and 2-fetch
 - Decoupled enqueue and dequeue
 - Flow control when dequeue
- Reduced redirect latency
- Train BPU when branches are resolved
 - Faster training usually means higher accuracy
 - Add a new resolve queue to cache branches
 - Complex control in resolve queue
- Manage queues according to life cycle

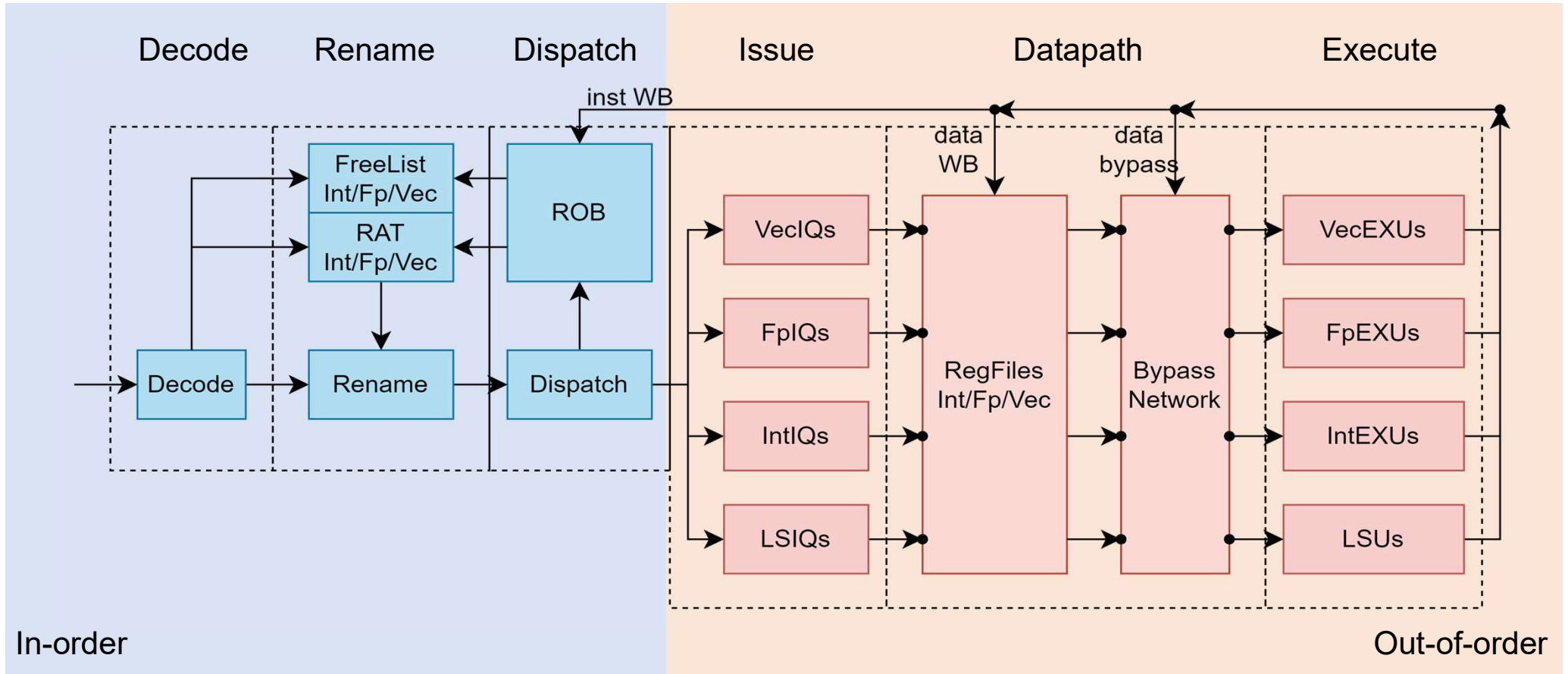


ICache

- 64KB (4-way)
- 64B cacheline
- 8B bank fine-grained storage
- 4 + 10 MSHR
- Read hit banks only for low power
- Support Zicbop prefetch.i
- Support parity check



Backend Architecture



Backend Design Philosophy

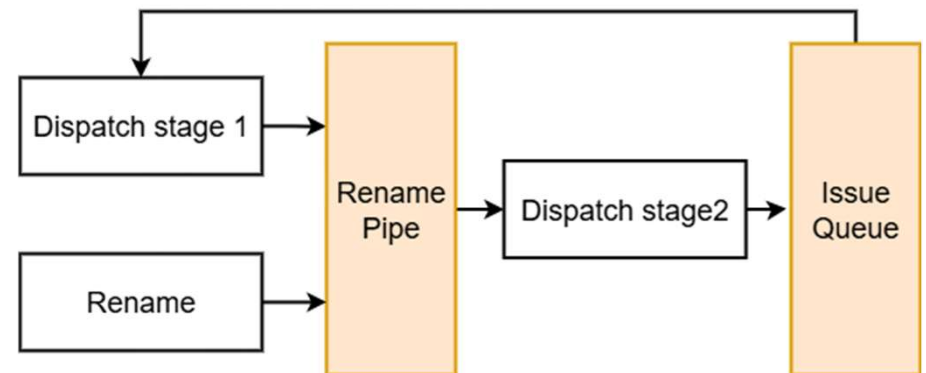
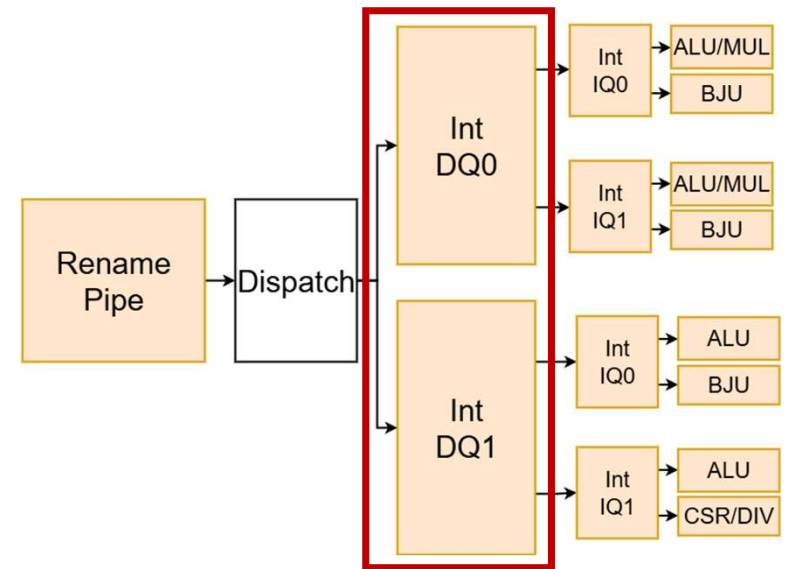
- Primary goal:
 - Throughput, with Timing, Power & Area
 - Functionality
- 8-wide out-of-order backend
 - Instruction fusion, move elimination, rename snapshot, ROB compression
 - Read regfile after issue, speculative wake-up, Regfile Cache
- Closely-coupled vector extension
- New CSR for privileged extensions like hypervisor

Decode and Rename

- Instruction fusion
 - Fuse adjacent μ ops to reduce dynamic instruction count
- Move elimination
 - Change the rename map directly
 - Mark as complete when move instruction enters ROB
- Rename snapshot
 - Re-rename from snapshot instead of committed instructions
 - Faster recovery after redirect
- Rob compression
 - Compress ROB entries according to certain rules

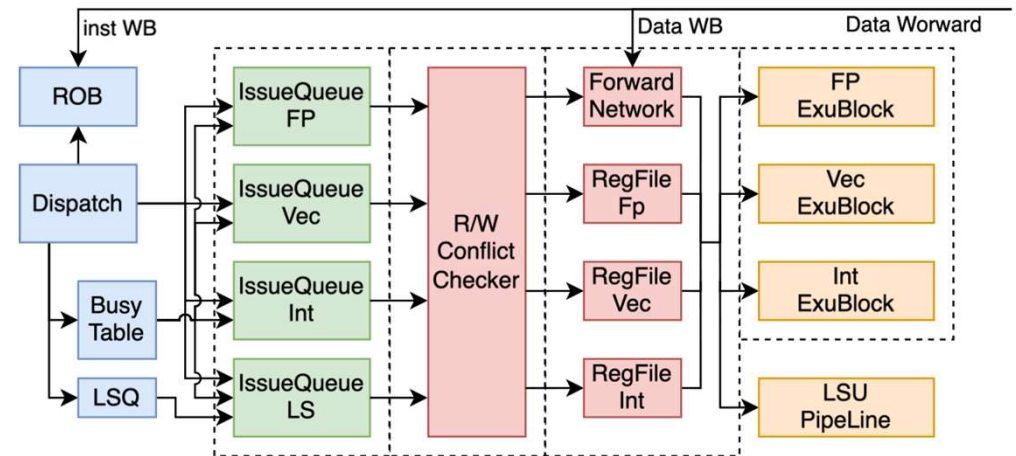
Dispatch: Optimization

- Distributed issue queues require fine dispatch algorithms
- Less cycles:
 - Remove useless dispatch queue between rename and issue
 - Move some logic to rename
- More balanced:
 - Better issue queue load evaluation based on dequeuing μ op type



🏔️ Read regfile after issue

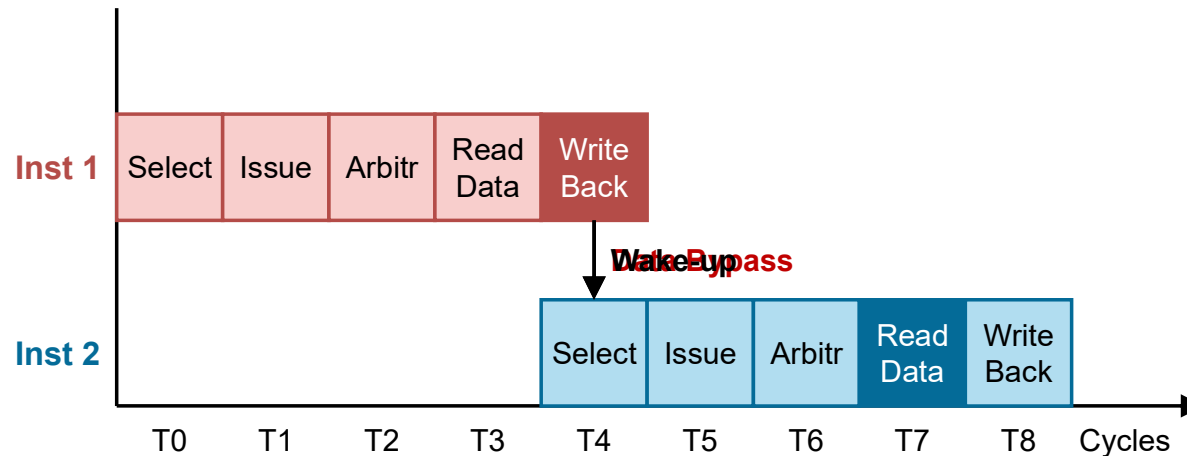
- Compared with otherwise:
 - Better timing and area
 - More function units and registers
 - More complex control path
- To mitigate:
 - Accurate speculative wakeup when issue
 - Efficient arbitration of regfile read ports



Issue: Speculative wake-up

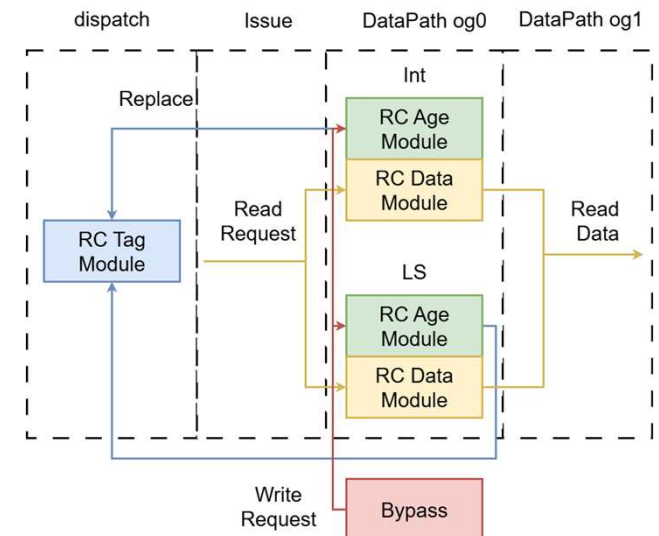
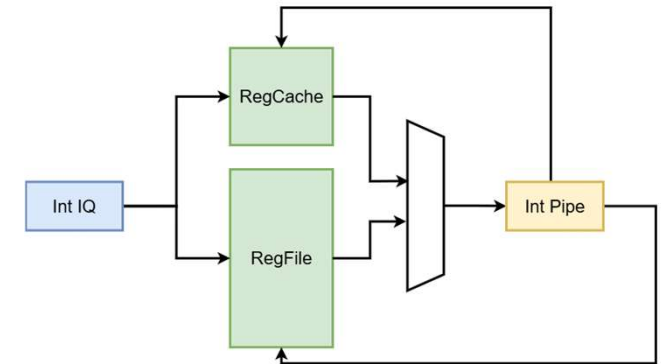
- Speculative wake-up

- More stages between wake-up and read-data \Rightarrow more bubbles
- For most μ ops, easy to know when it finished
- Speculative wake up μ op, so that it could read data just when last μ op writes back



Data Path: Regfile Cache

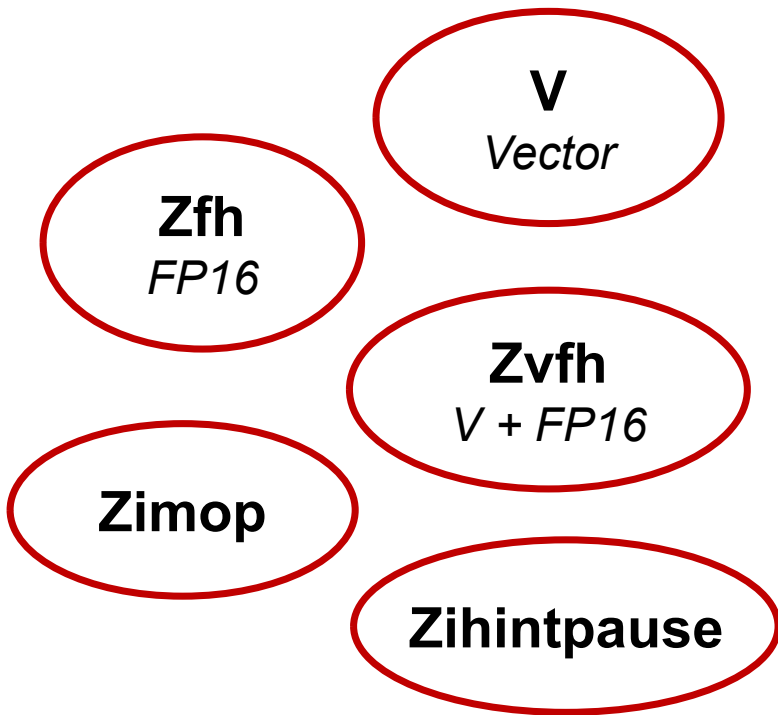
- Multiple functional units share a regfile read port
 - Conflicts on read ports
- Regfile is too big to add more read ports
- Add a small cache for regfile
 - Small \Rightarrow more read ports
- Reduce conflicts on original regfile
 - Same performance with fewer read ports
 - Fewer read port \Rightarrow better timing



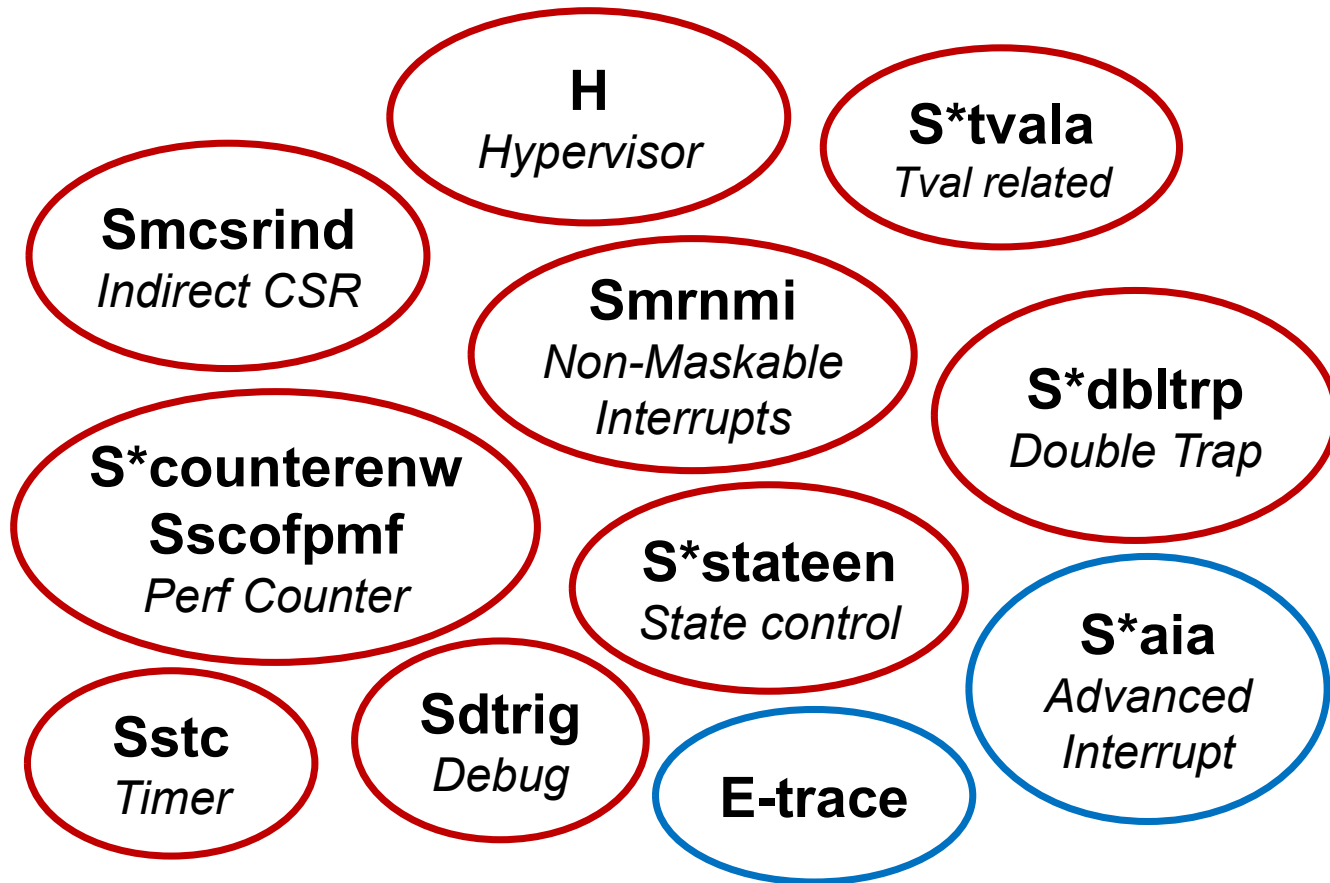


Backend Related Functionality

Unprivileged



Privileged



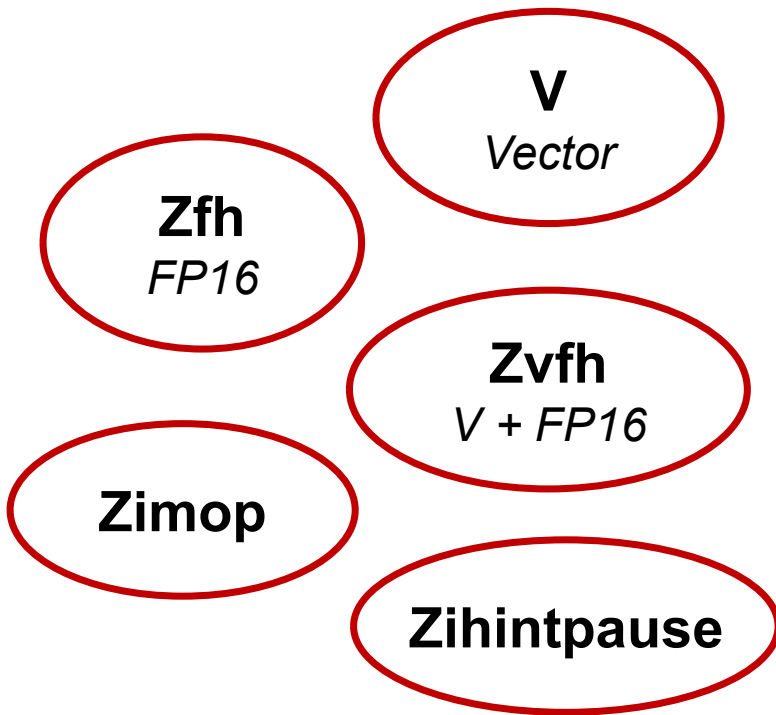
 required by RVA23

 required by Industrial



Backend Related Functionality

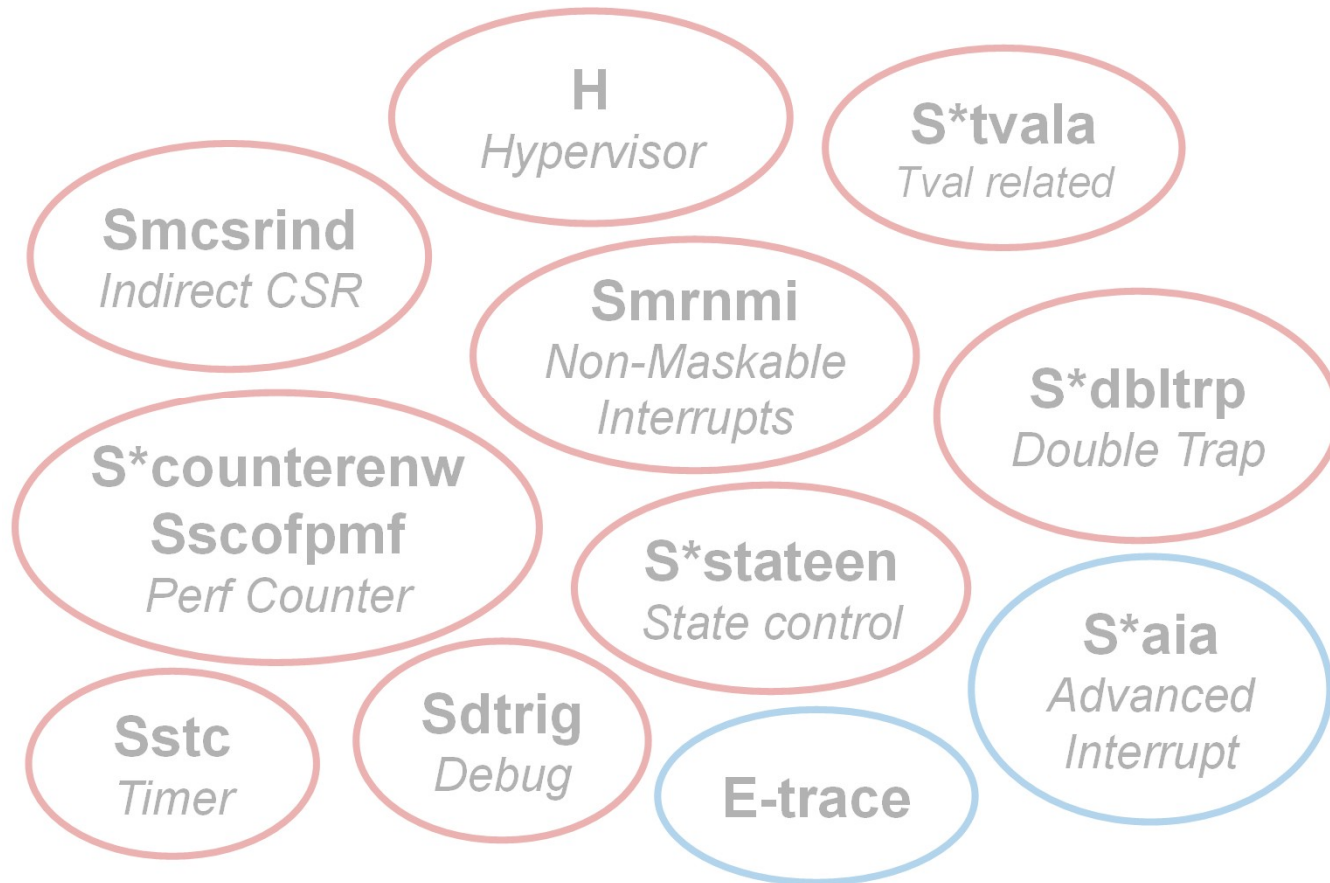
Unprivileged



 required by RVA23

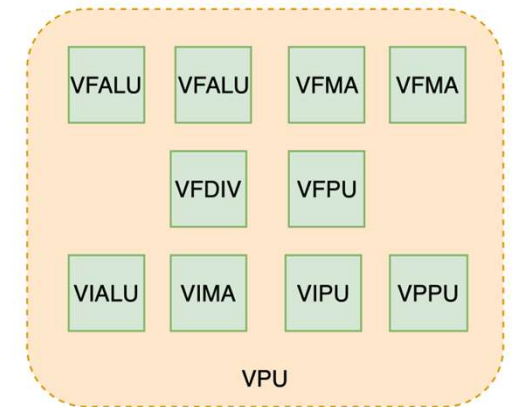
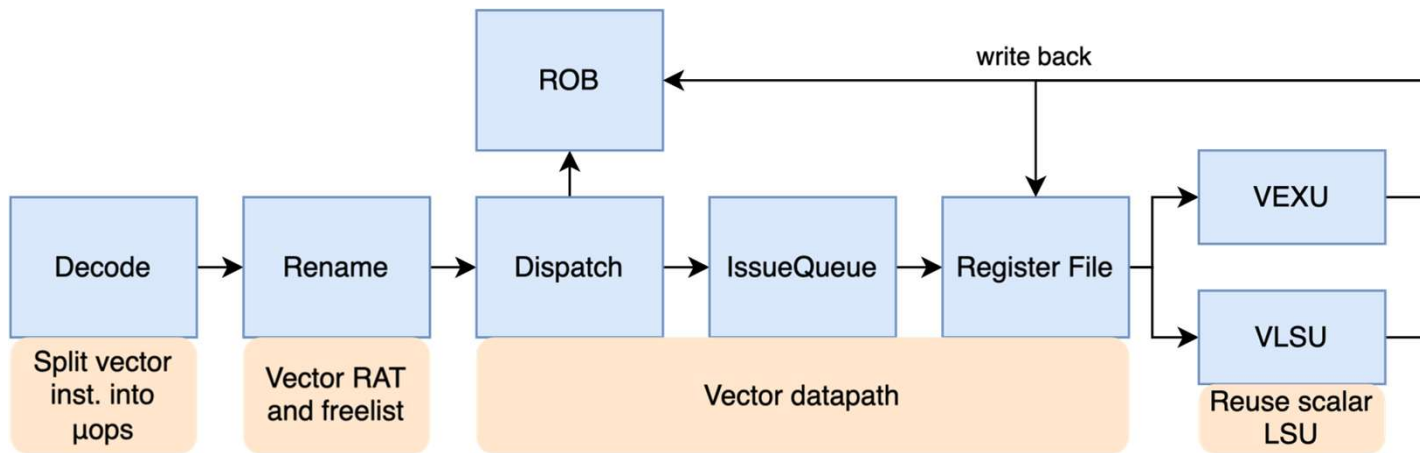
 required by Industrial

Privileged



Vector Extension

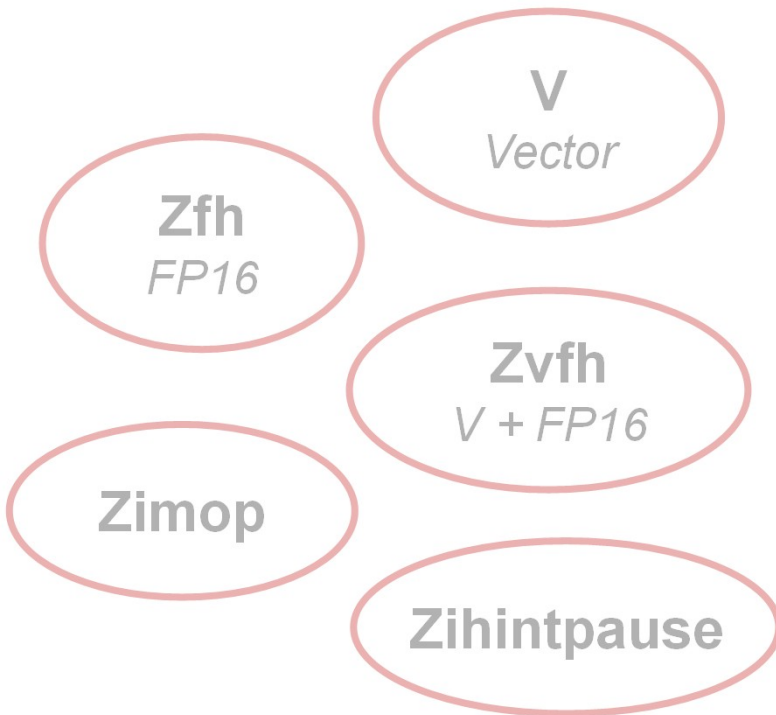
- Closely-coupled vector design, reuse existing pipeline
- Split vector instructions into micro-operations (μop) at the granularity of vector registers
- Separate issue queue and vector regfile
- VPU also process floating-point calculations



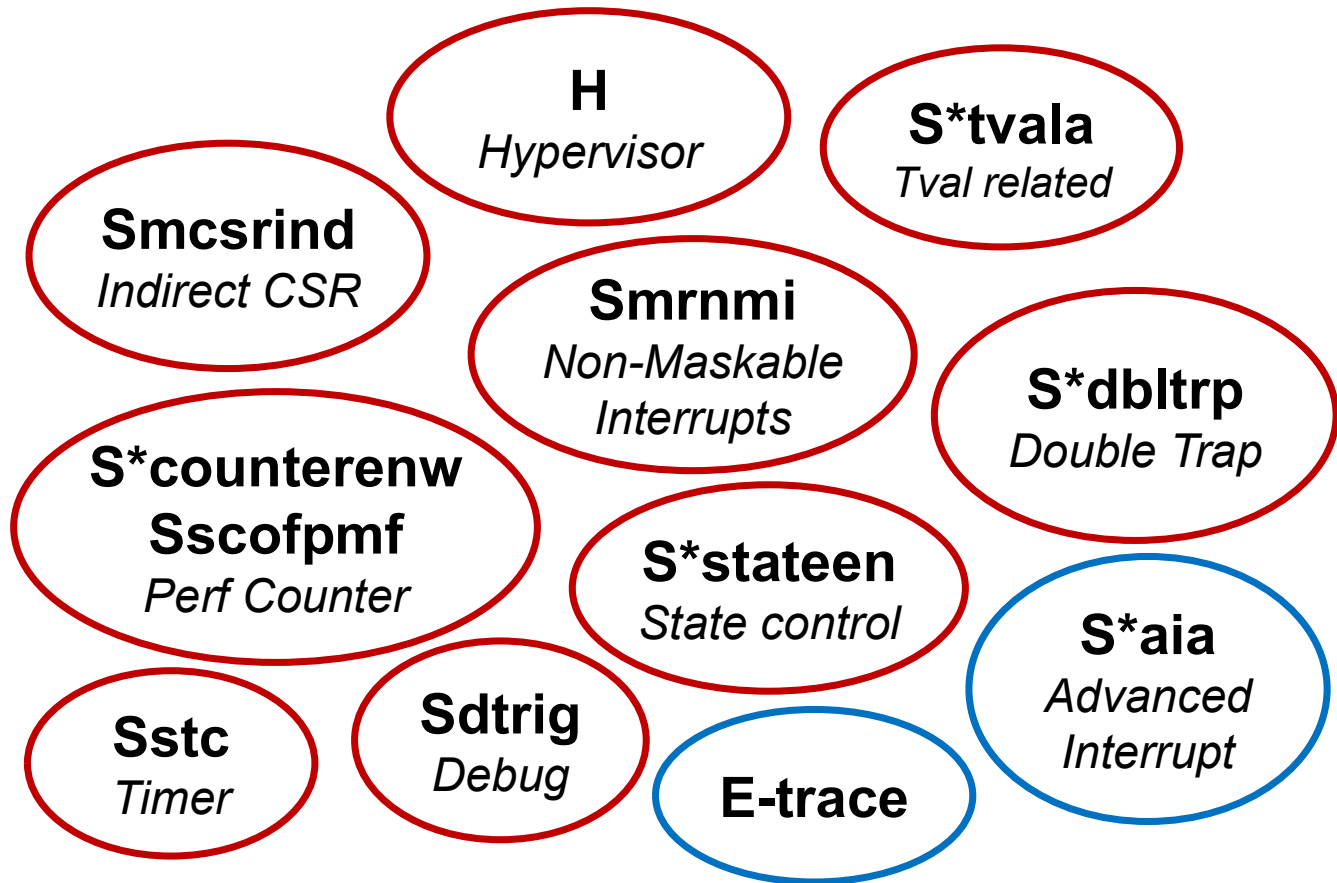


Backend Related Functionality

Unprivileged



Privileged



required by RVA23

required by Industrial

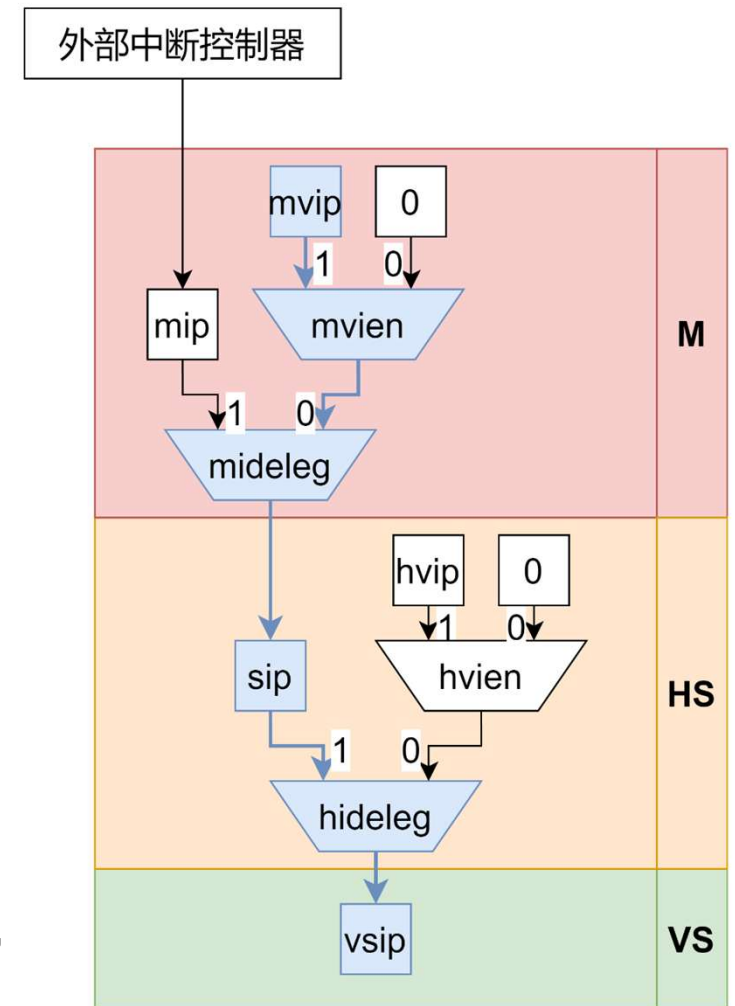
CSR: Core of Privilege-level

- Most privilege extensions introduce **new CSR**
 - For H-ext, it introduces dozens of CSRs
- Complex extensions define complex behaviors of CSR
 - It's not easy to correctly implement CSRs in traditional way
- Use **Object Oriented** to abstract
 - Chisel and Scala enable us to implement CSRs in a more systematic way



New CSR Design

- CSR fields are defined as four types
 - RW, RO, WARL, WLRL
 - OOP allows us to easily specify the type
- More accurate field descriptions for WARL
 - Alias (Read & Write)
 - `sstatus.VS` is an alias of `mstatus.VS`
 - View (Read-only)
 - `sip.SEIP` is a view of `mip.SEIP`
 - Conditional Alias (View)
 - when `mvien=1 && mideleg=0 && hideleg=1`, `vsip` is an alias of `mvip`



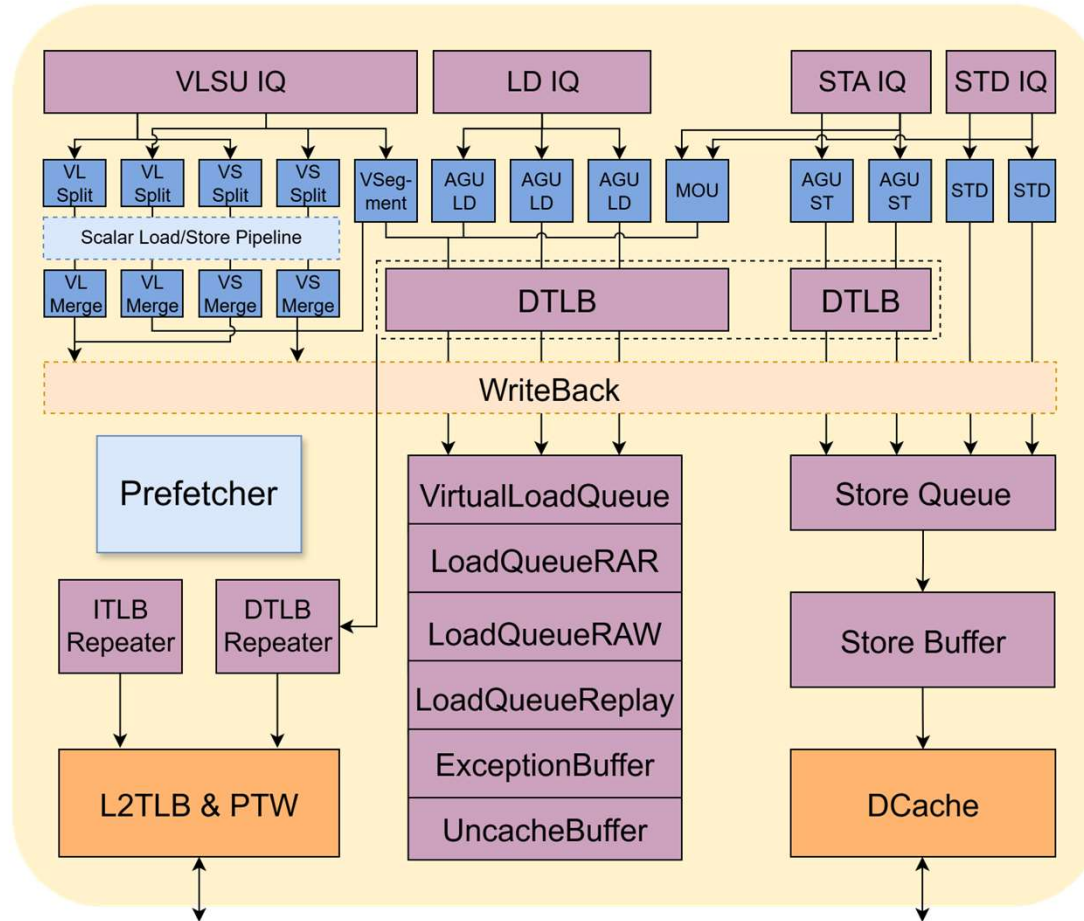
Interrupt injection and proxy CSRs with H-ext and AIA 33



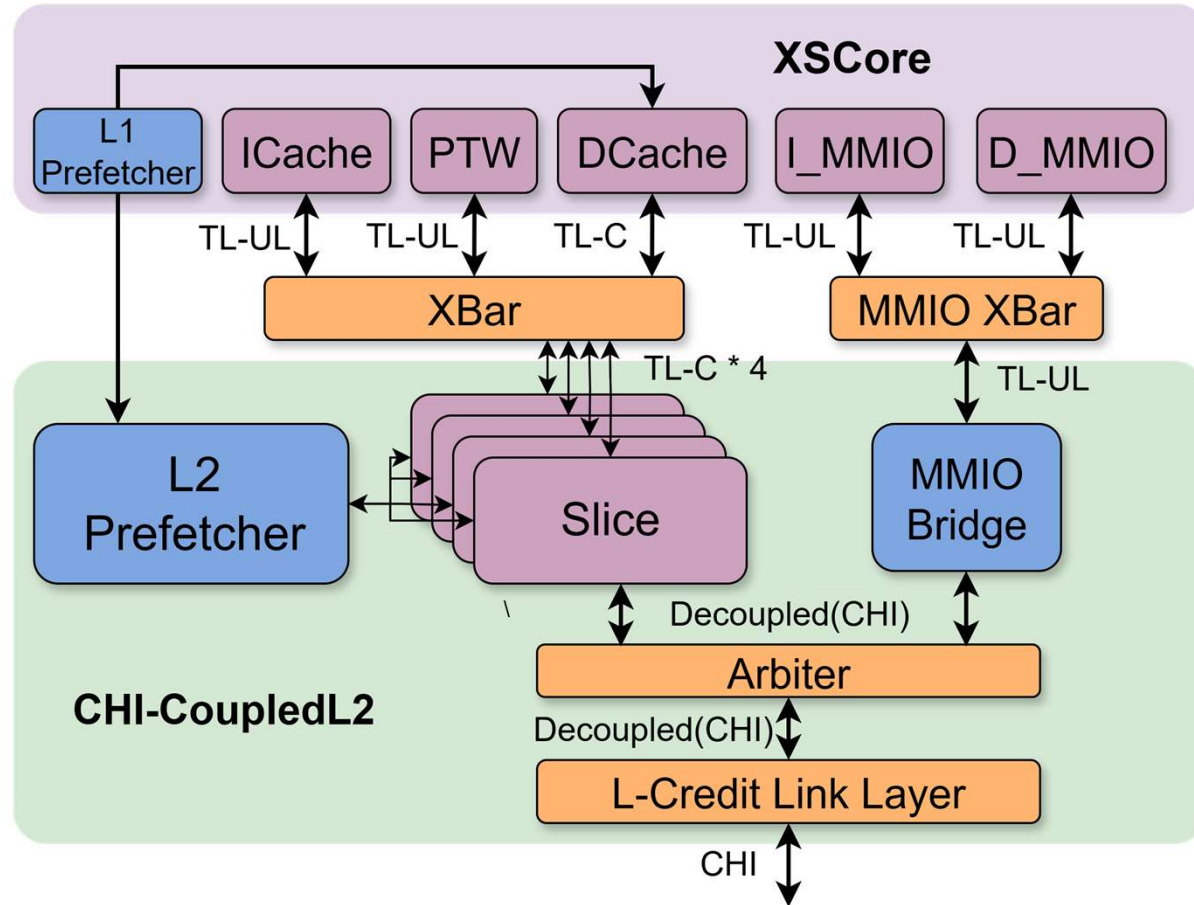
New CSR Design

- Modularize trap event handler
 - 5 privilege levels with trap enter and return
 - Debug、MN、M、S、VS
 - Trap enter & Trap return
 - Events could update the CSR fields directly
- Fine permission check
 - CSR Access permission check
 - Privilege level check, direct/indirect access restrict check
 - Instructions execution check
 - Privilege level check, execution condition check
- Conducive to design expansion and unit verification.

Memblock Architecture



Cache System Architecture



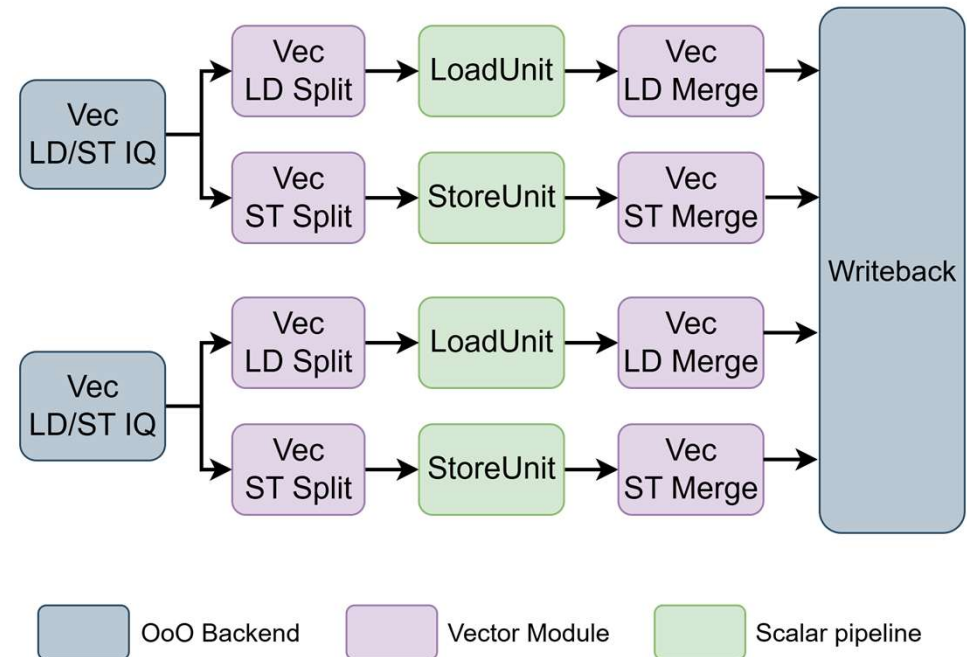


Memblock and Cache Design Philosophy

- Primary goals:
 - Functionality and safety
 - Latency and bandwidth
- Support for V and H extension
- Features for high performance Memblock
- Agile integration framework for prefetchers
- Optimized low-latency cache system
- Open-Source CHI framework (Support Issue E, C, B)

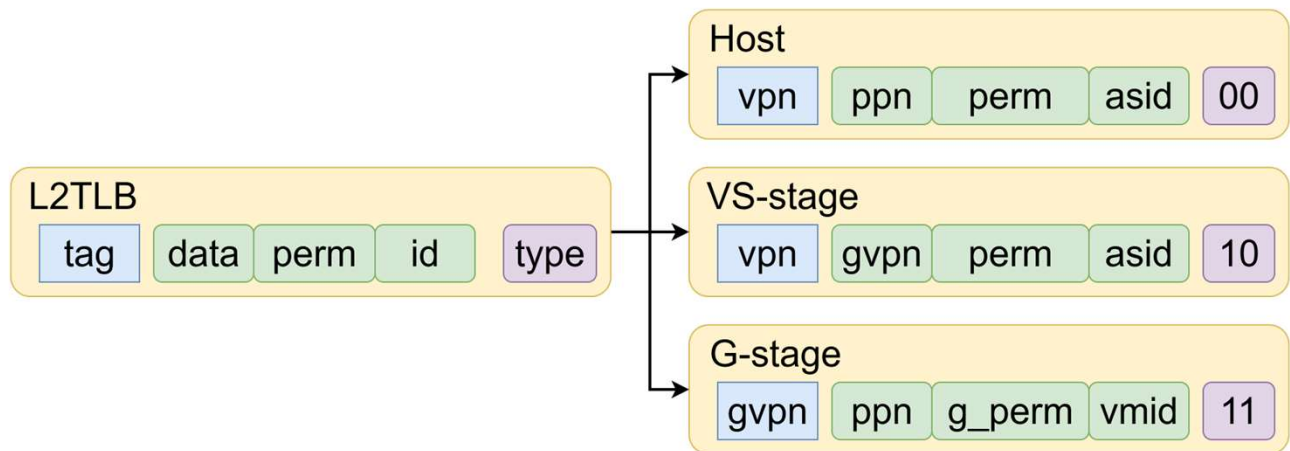
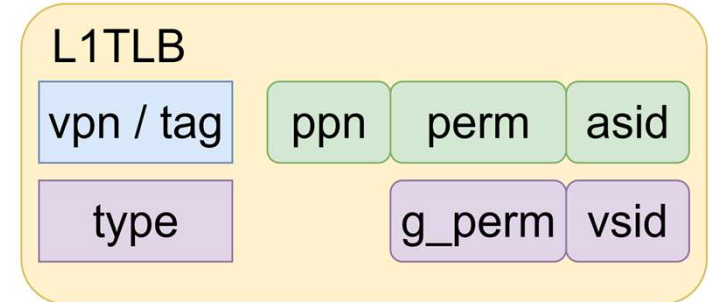
V Extension

- Split vector instructions to scalar load/store operations
- Reuse scalar pipeline
- Violation detection at split memory op granularity for mixed vector/scalar memory accesses .
- Bandwidth optimization for unit-stride access
 - One translation and cache request for contiguous memory access



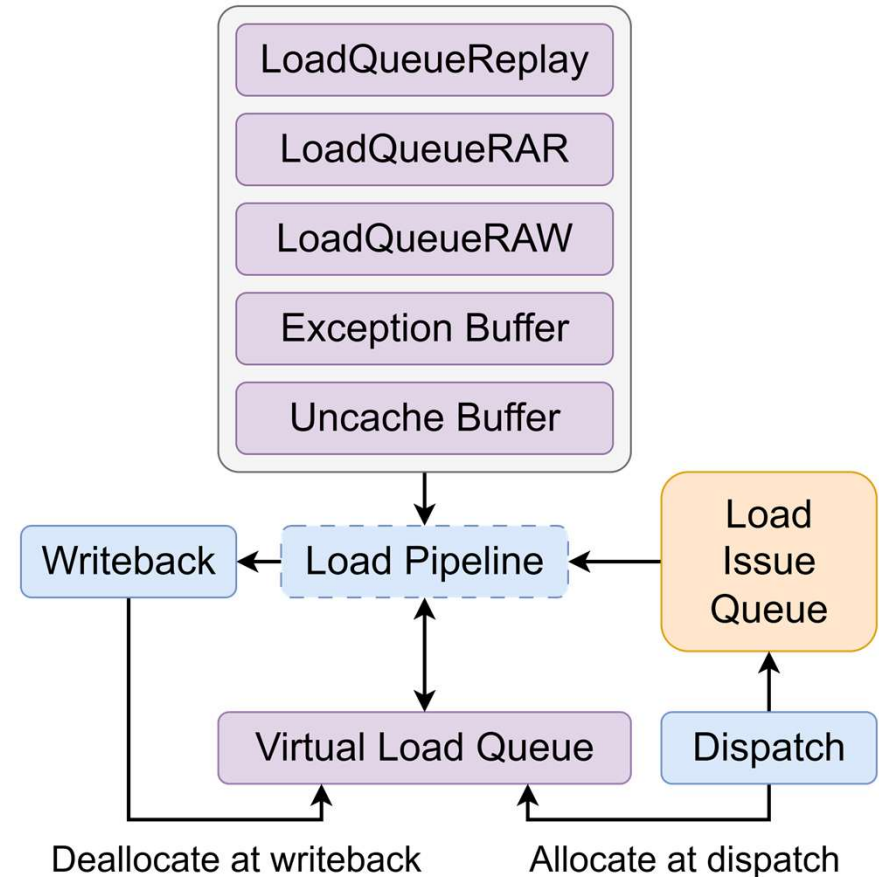
H Extension

- 2-stage address translation (Sv48x4)
 - Maximum 16 pages walked before getting physical address
- Add new fields in TLB entry
- New hypervisor PTW



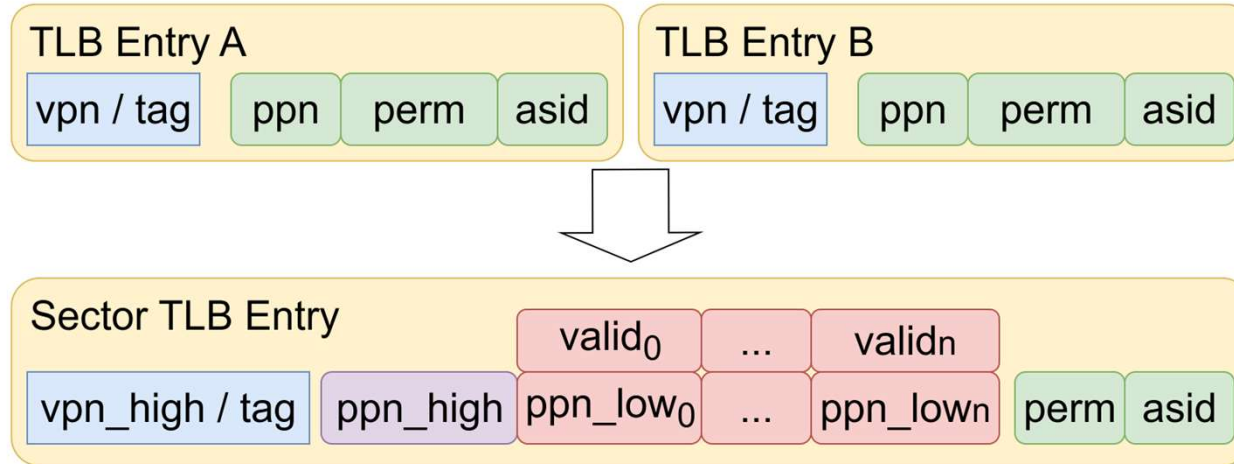
❖ Distributed Load Queues

- Split load queue based on different responsibilities
- Simpler control, better timing and acceptable area
- Early commit: retire loads sequentially from virtual load queue after write-back, not after issue
- Free entries for younger loads



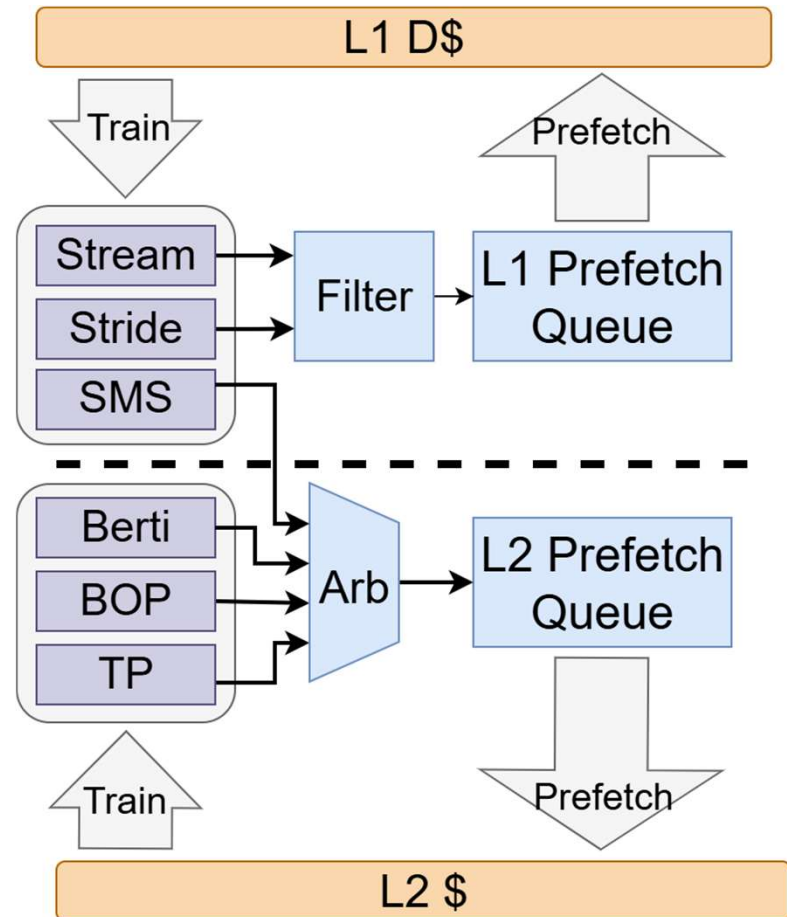
TLB Compression

- TLB compression for larger capacity
- Merge contiguous page table entries
 - Same high bits of VPN and PPN
 - Same page permissions



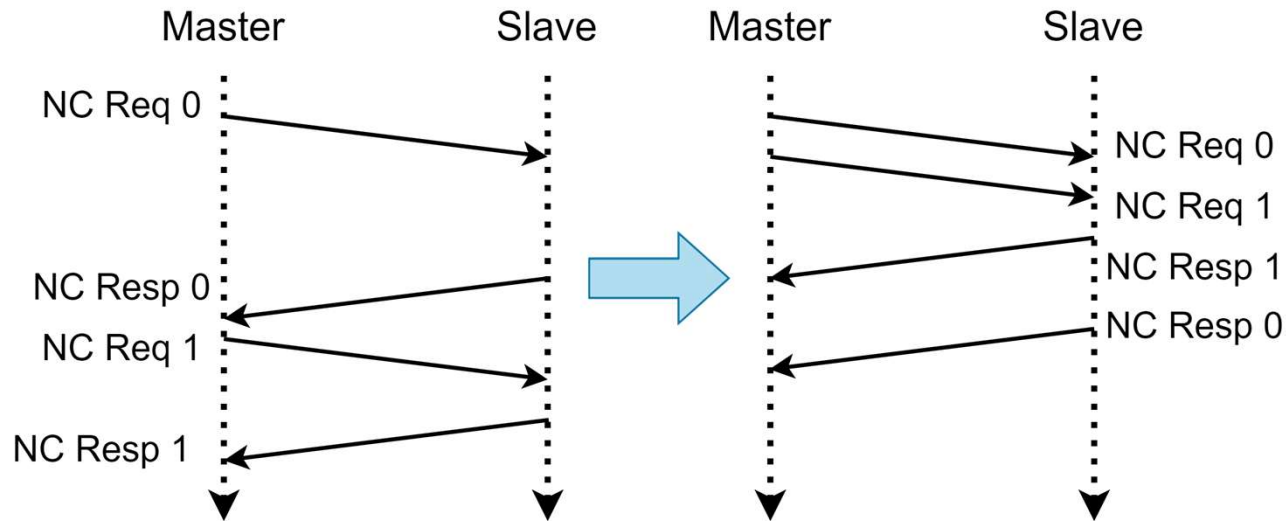
❖ Prefetch Framework

- Multi-prefetcher Collaboration
- Prefetching and Training in L1 D\$ and L2\$
- Agile Integration of New Prefetchers
 - Integrate new prefetcher with minimal changes (~4 lines of code)
 - Ideal platform for academic research on prefetching algorithms



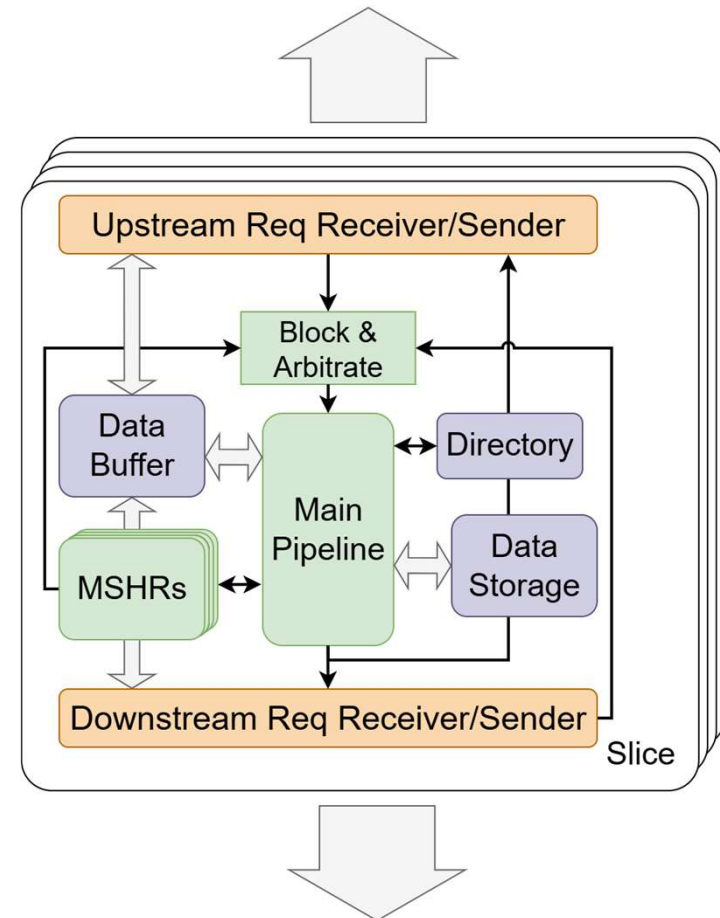
❖ Outstanding IO Performance

- Svpbmt: memory attribute based on pages
 - IO: non-cacheable, sequential
 - NC: non-cacheable, RVWMO
- Make NC area Out-of-order



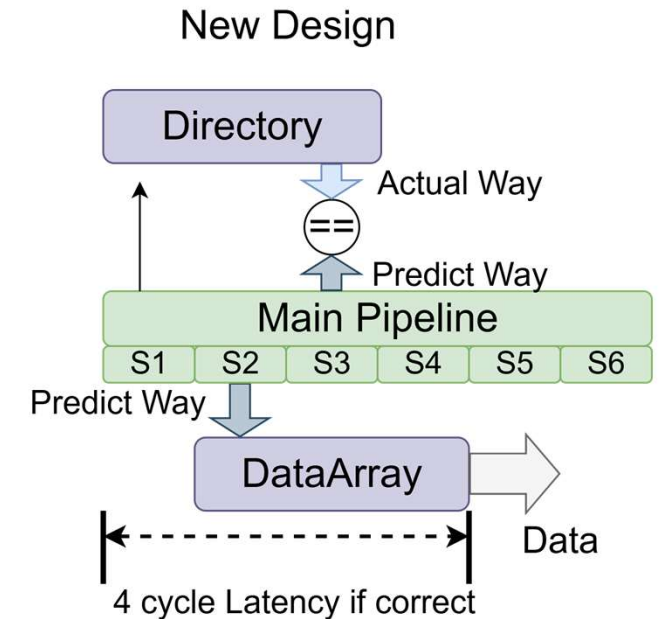
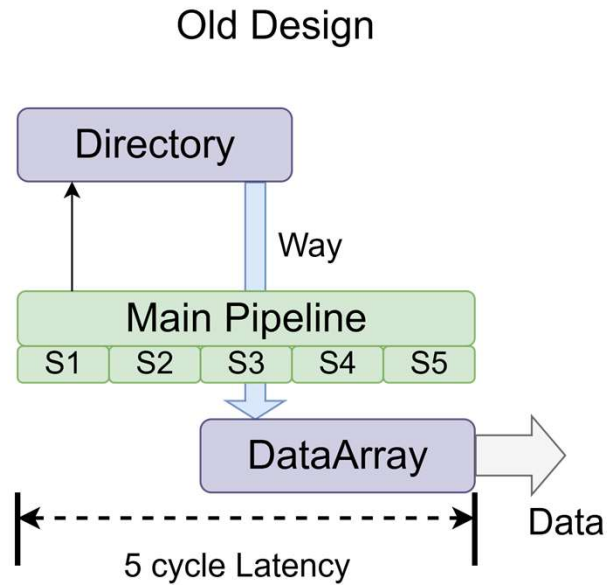
Cache Design Principles

- Common design principles of CoupledL2 and OpenLLC
- Directory-based coherence
- High-concurrency design
 - Multiple slices and MSHRs
 - Non-blocking main with back-pressure control at entry
 - Selects victim way after data return from downstream, permitting same-set access overlap
 - Support data bypass using data buffers



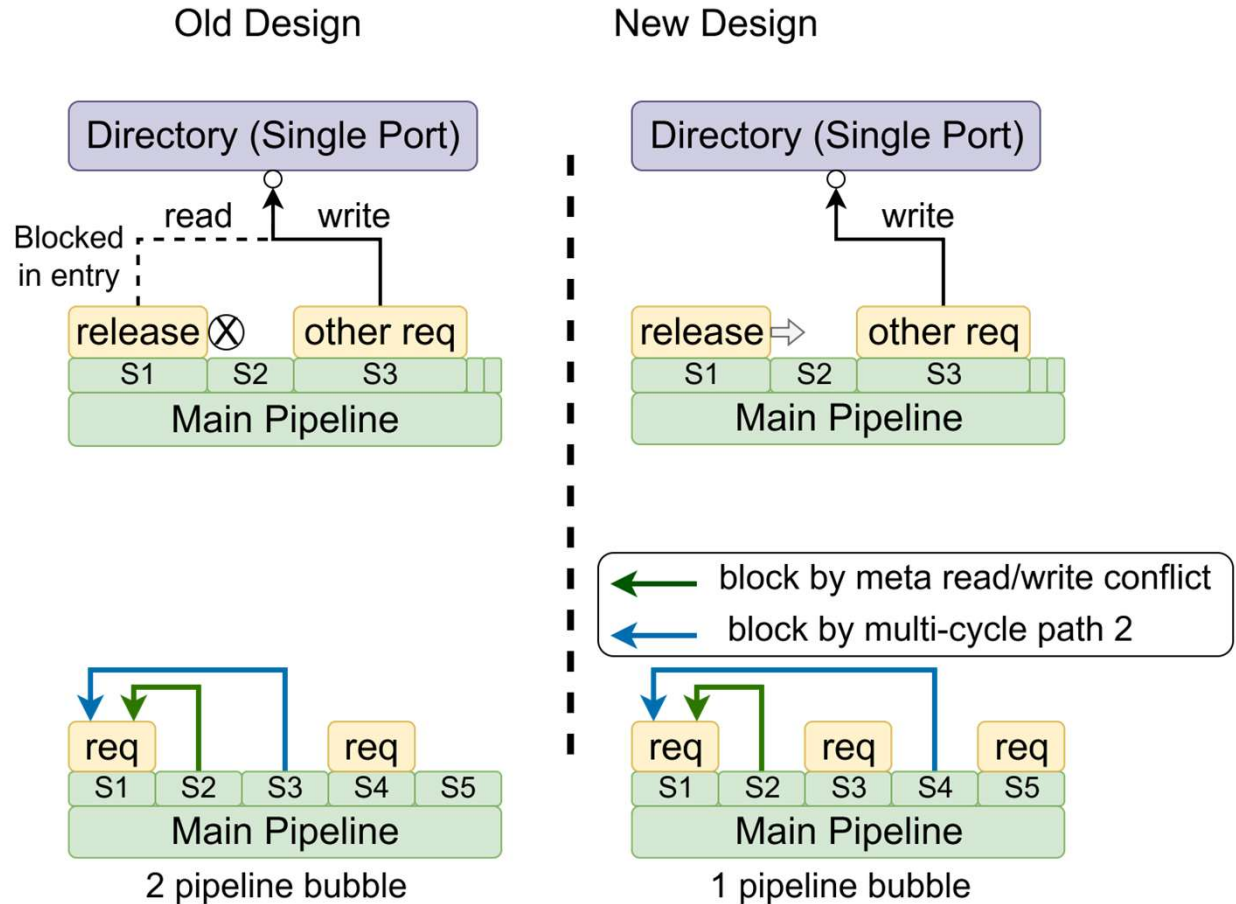
❖ CoupledL2 Design

- Basic design refer to previous page
- Optimization to reduce latency
 - Employ way prediction to read data array in advance



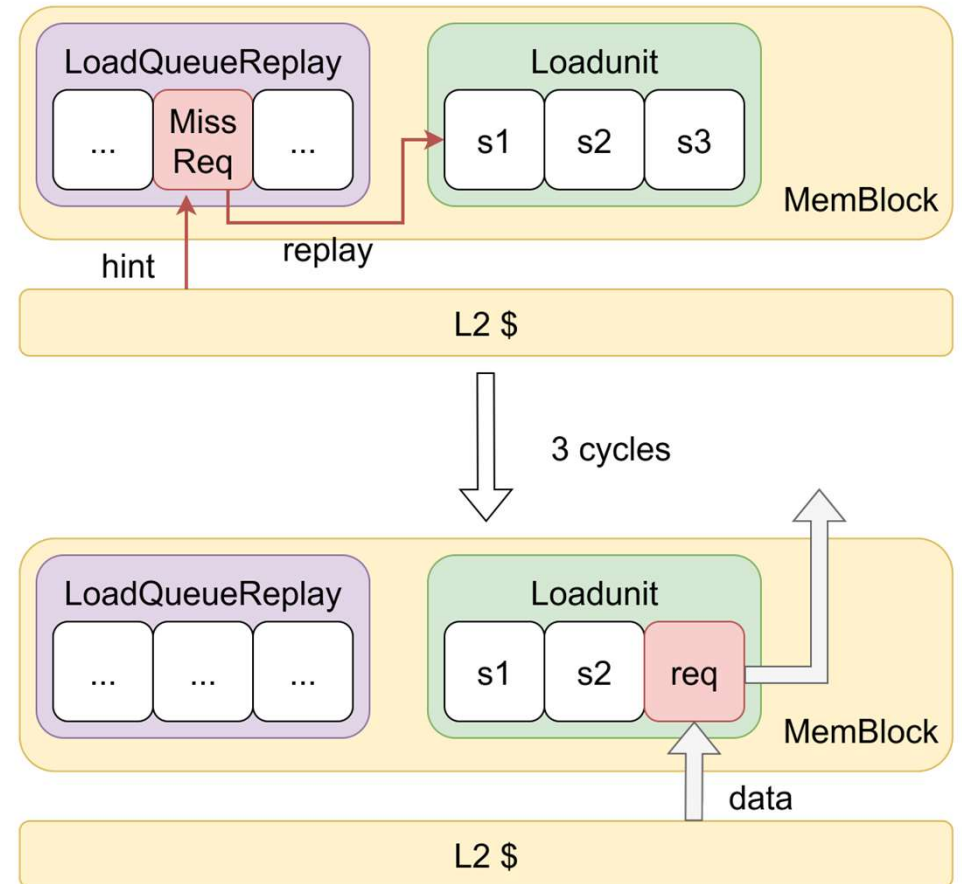
❄️ CoupledL2 Design

- Optimizations to reduce pipeline bubble
 - Record L2 way in L1 to eliminate tag read on L1-to-L2 release
 - Defer directory write to main pipeline stage 4



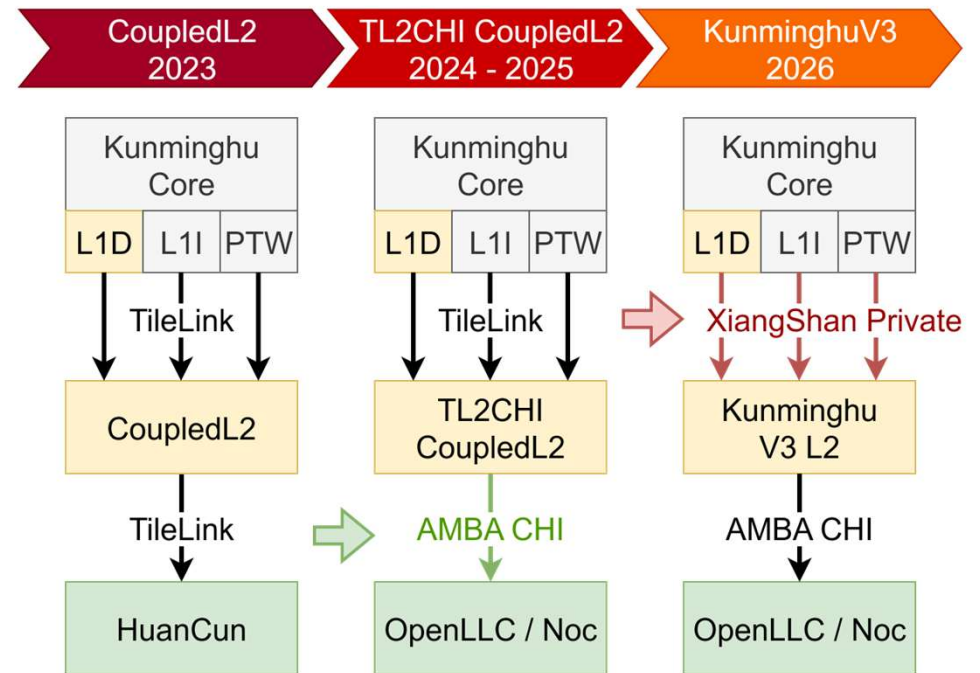
❄️ L1-L2 Co-optimization

- 3 fixed cycles between load waking up and using refill data
- L2 send hint signal to L1 3 cycles before refill
- L1 directly use refill data 3 cycles after
- Reduce load-to-use latency



Private Protocol between L1 and L2

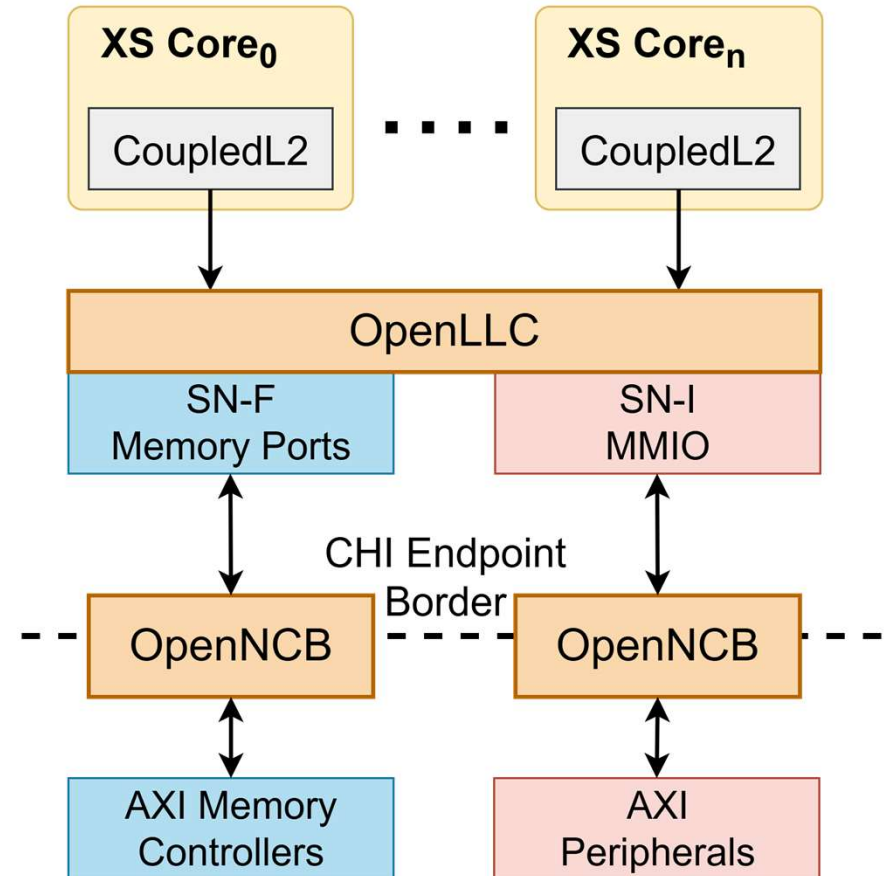
- Modifications have been made based on TileLink
 - To support CMO instructions
 - To support L1-L2 hint
- Conversions between different coherence protocols are complex
 - More complicated verifications
 - Result in conservative design
- TileLink is functionally limited compared with CHI
- New private protocol in KunminghuV3





Open-Source CHI framework: OpenLLC

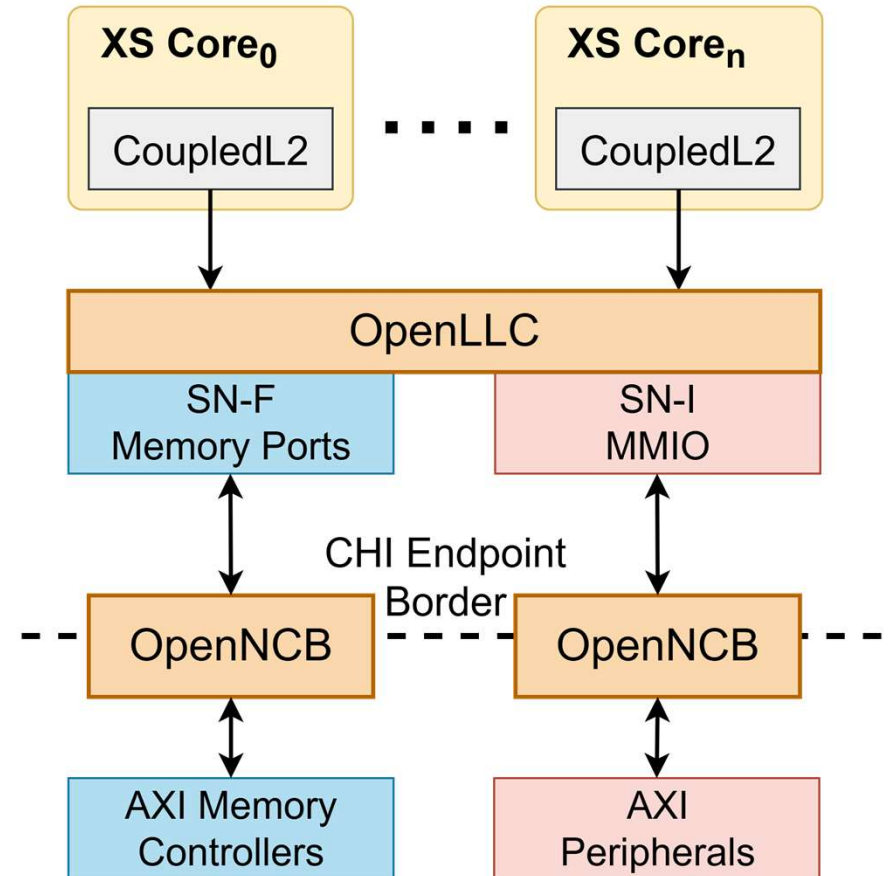
- Last level cache as HN-F
- Support multiple inclusion policy
 - Exclusive for single core, non-inclusive for multi cores
- Support necessary CHI transactions, including CMO and AMO request





Open-Source CHI framework: OpenNCB

- Non-Coherent Bridge to AXI4
- Credit-controlled Zero Retry
- Configurable outstanding request depth, bus data width and transaction ordering
- Early return of observationally ordered transactions, allowing aggressive out-of-order execution in HN





Conclusion

- High-bandwidth, decoupled frontend
 - High-accuracy BPU and high-bandwidth IFU
 - 2-taken and 2-fetch support in BPU and IFU
- Full-featured, high-bandwidth backend
 - Support for V extension
 - New CSR Design
- Secure and low-latency memory subsystem
 - Support for V and H extension
 - Optimized L2 cache and prefetcher
- Refactored for higher code quality
 - easier to understand existing design and implement your ideas
- Questions are welcomed!