

# GPGPU L1 Data Cache

金楚丰

# Menu

- CPU L1 Data Cache Features Overview
  - Ports
  - Flows
- GPGPU L1D Incremental Features Overview
- Function Units Introduction
  - Data Access
  - Arbiter & Data Crossbar
  - Tag Access
  - Miss Status Holding Registers
  - Write Data Buffer

# Menu

- Typical Work Flows, Pipelining
  - READ hit flow
  - READ miss flow
  - WRITE hit flow(sub-word or not)
  - WRITE miss flow, without sub-word
  - WRITE miss flow, with sub-word
- Control Hazard

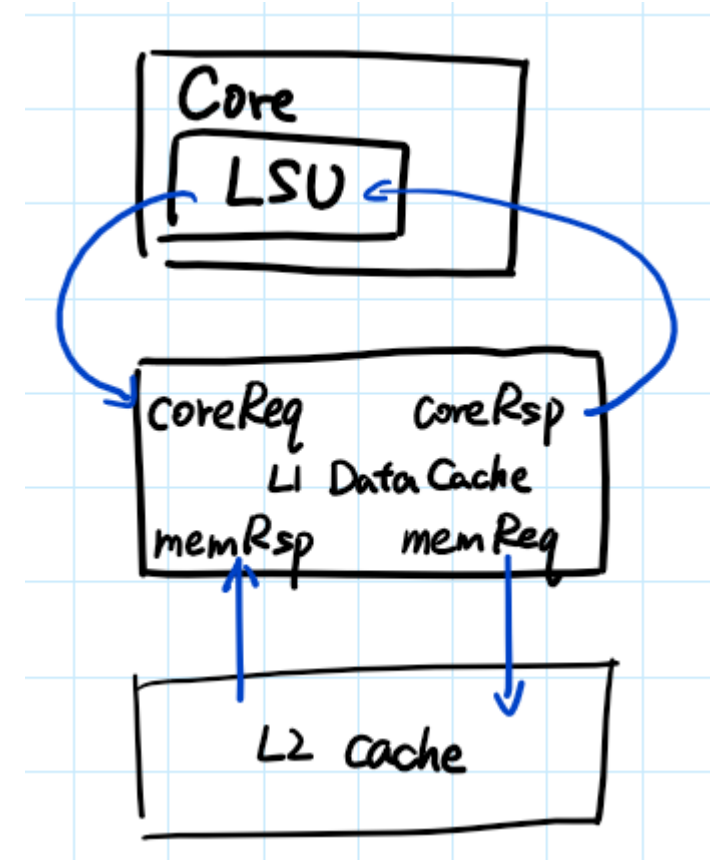
# CPU L1 Data Cache Features Overview

Function/Implementation Perspective

Typical Work Flows: 1 Behavioral Classification 2 Path Classification

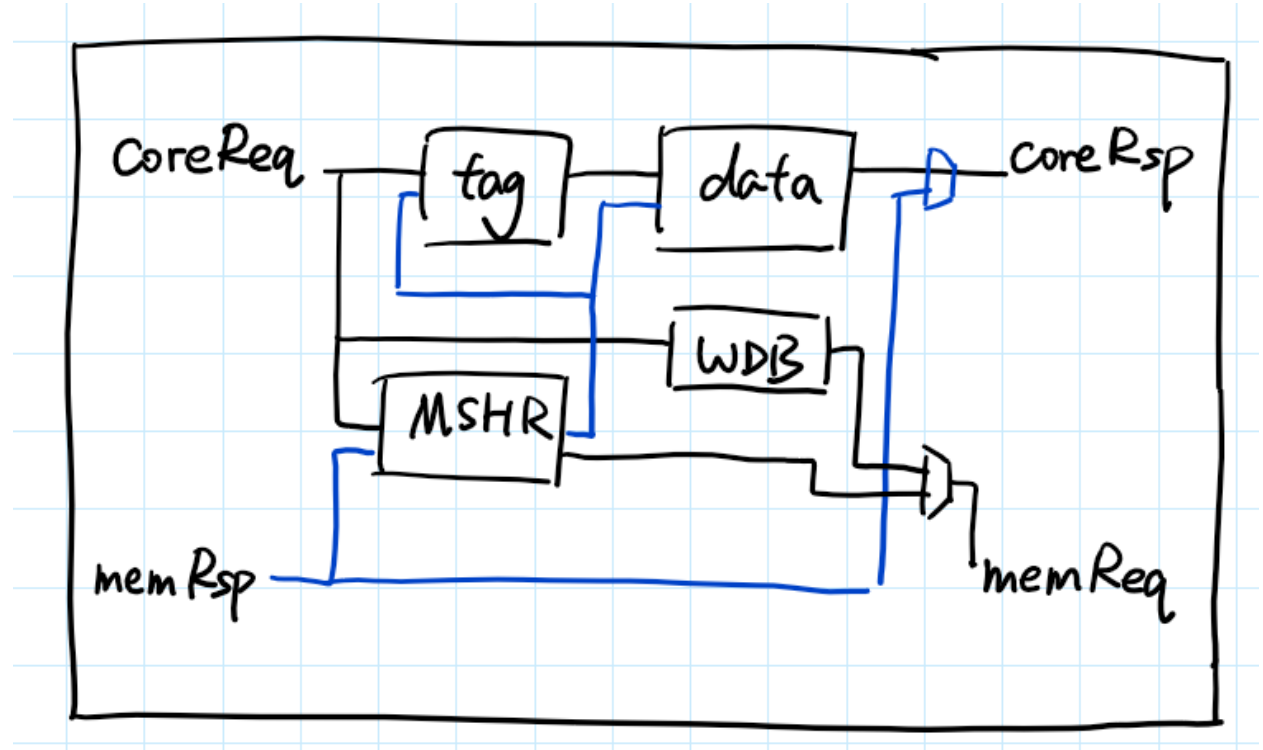
# CPU L1 Data Cache Features Overview

- Functional speaking
- Handle **READ** request and **WRITE** request
- Handle **miss** and **hit**
- WRITE back or **WRITE through**
- If WRITE through, **WRITE around** or WRITE allocate when miss happens?
- Support **sub-word**(byte or half word) WRITE, while L2 do not



# CPU L1 Data Cache Features Overview

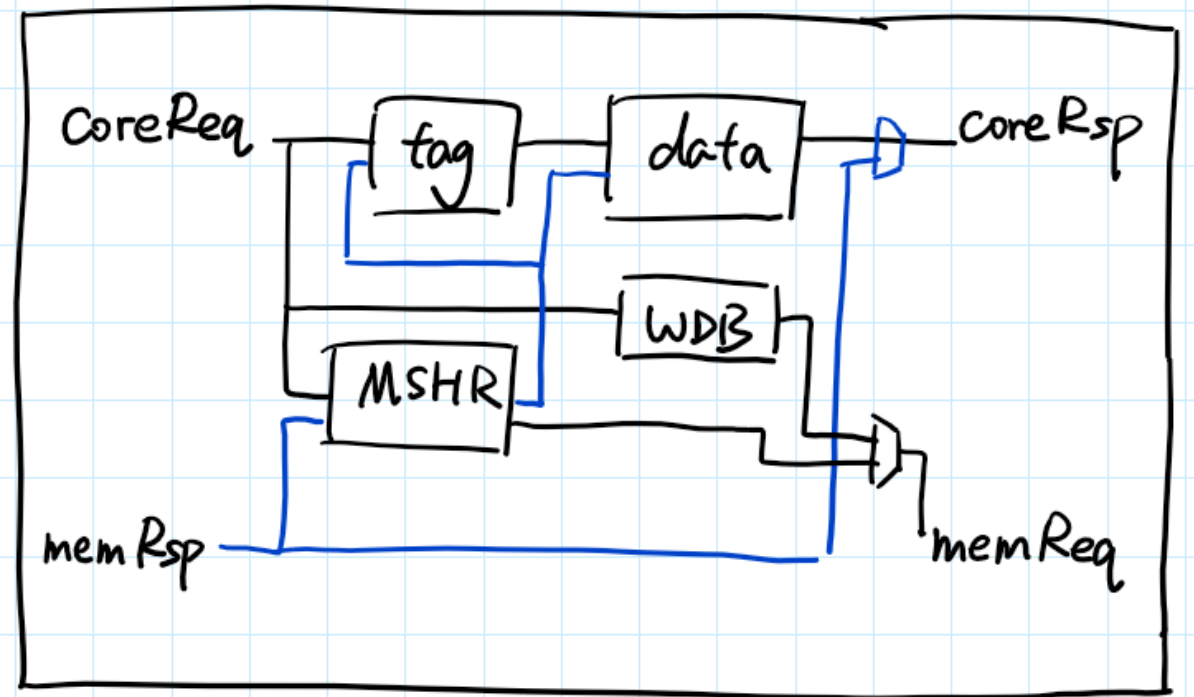
- In an implementation perspective
- Two input ports:
  - coreReq
  - memRsp
- Two output ports:
  - coreRsp
  - memReq
- MSHR
  - as a bridge connecting miss request event and miss response even



# CPU L1 Data Cache Features Overview

- Typical work flows, behavioral classification

- READ hit
- READ miss
- WRITE hit
  - without sub-word
  - with sub-word
- WRITE miss without sub-word
- **WRITE miss with sub-word**

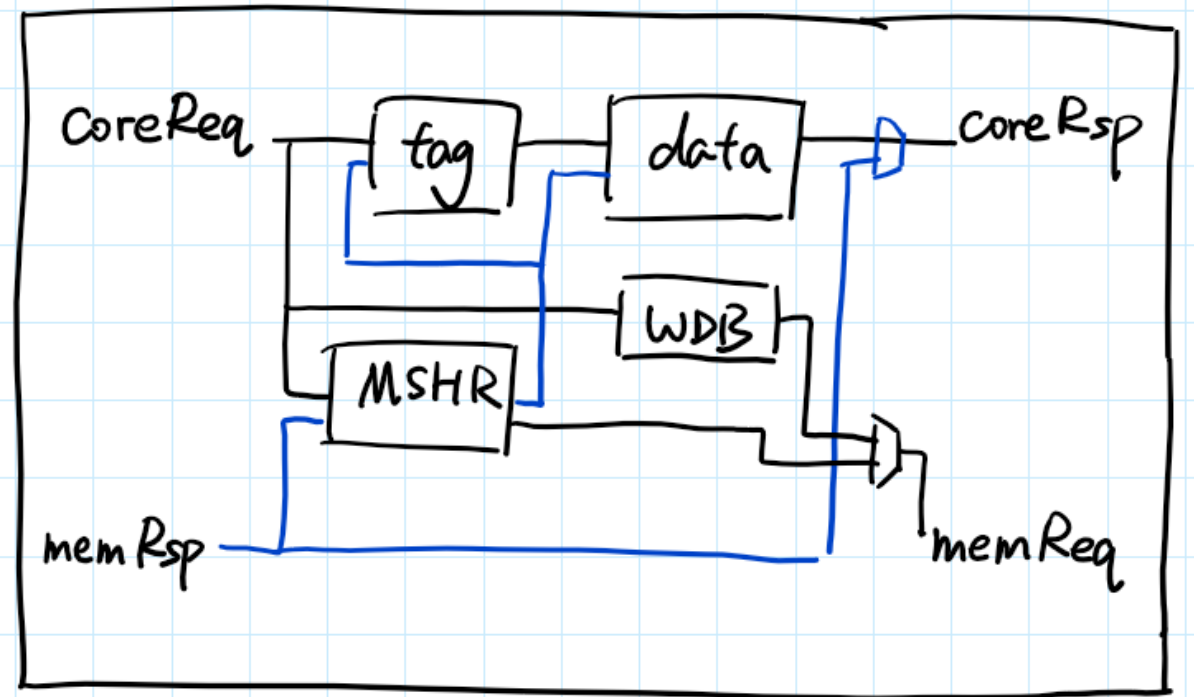


- “common things fast”
  - -- **uncommon things** cursed

# CPU L1 Data Cache Features Overview

- Typical work flows, path classification

- coreReq-coreRsp
- coreReq-memReq
- coreReq-Data Access
- memRsp-coreRsp
- memRsp-memRsp
- memRsp-Data Access




- one behavioral flow may contain multiple path flows
- one path flow may contribute to multiple behavioral flows



# CPU L1 Data Cache Features Overview

Latency estimation for each flows



	R h	R m	W h	W m ns	W m s
cReq-cRsp	4~+a				
cReq-mReq		3	3~+a	3~+a	3~+a
cReq-data					
mRsp-cRsp		4~+b			
mRsp-mReq					3
mRsp-data					

READ hit  
READ miss  
WRITE hit  
WRITE miss no sub-word  
WRITE miss sub-word

BlockSize/Nlanes=a

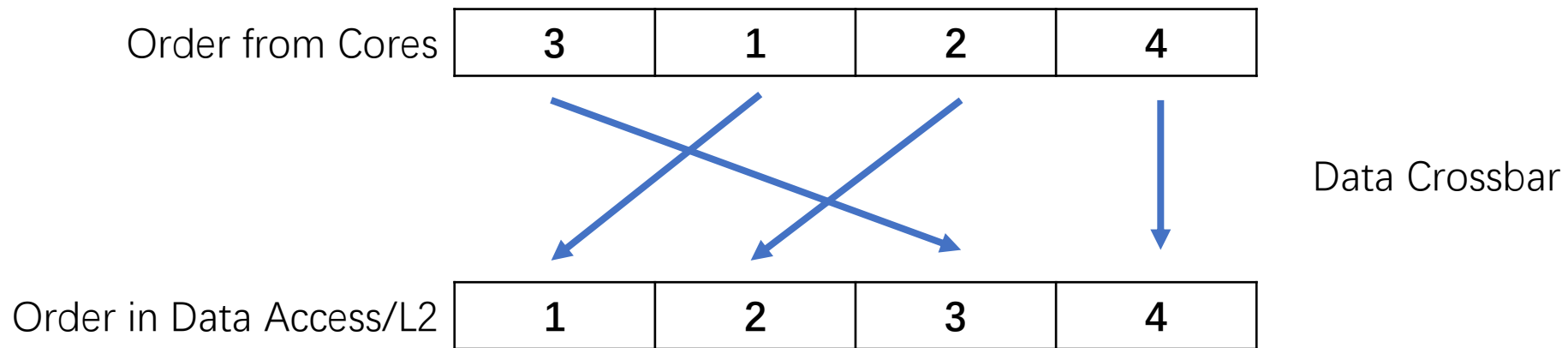
MSHR # of valid subentries=b

# GPU Incremental Features

And Their Implemental Considerations

# GPU Incremental Features

- Handle **concurrent, scattered** request from multiple hardware threads
- Add an Arbiter, a Data Crossbar, and enlarge the bandwidth of WDB to support this feature



# GPU Incremental Features

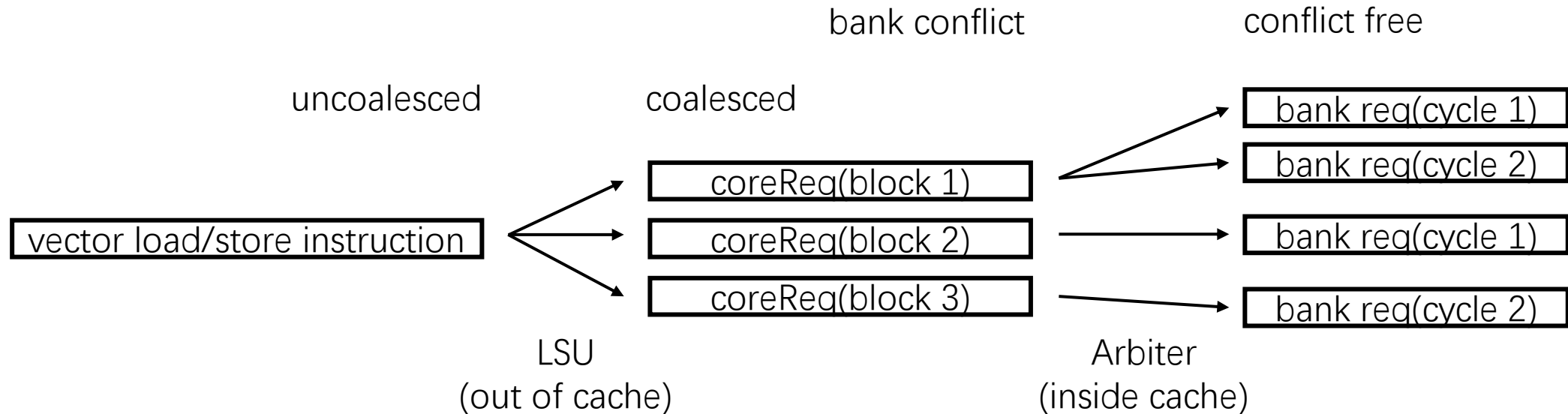
- Data Crossbar is quite expensive
- **Bidirectional, NLanes \* NLanes**-in-1 MUX, with each data channel width of **WordLength**
  - assume NLanes = Nbanks
  - typical values assigned: 32 \* 32-in-1 MUX, each channel has 32 bits
- non-CMOS design might be attractive

# GPU Incremental Features

- Data Crossbar is quite expensive
- But 2 different jobs for it
  - Scattered – Ordered (Core – Mem)
  - Ordered – Scattered (Mem – Core)
- **Reuse** for cost, or duplicate for perf?
  - reuse will cause pipeline stall when different flows adjoin

# GPU Incremental Features

- Limitations: coreReq must be coalesced
  - Benefit: for tag related operations(Tag Access, MSHR), keep same as CPU style.



# Function Units Introduction

Data Access

Arbiter & Data Crossbar

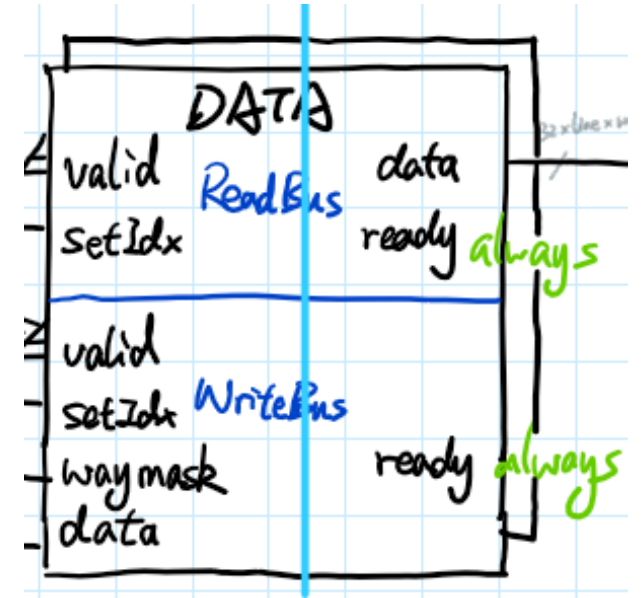
Tag Access

Miss Status Holding Registers

Write Data Buffer

# Data Access

- CPU D\$ features:
- Hold blocks of data, in a set associative manner
- Write and Read port are separate, can work concurrently
- Support sub-word write
- To index a block:
  - setIdx, bitfield of original addr
  - wayIdx, managed by Tag Access

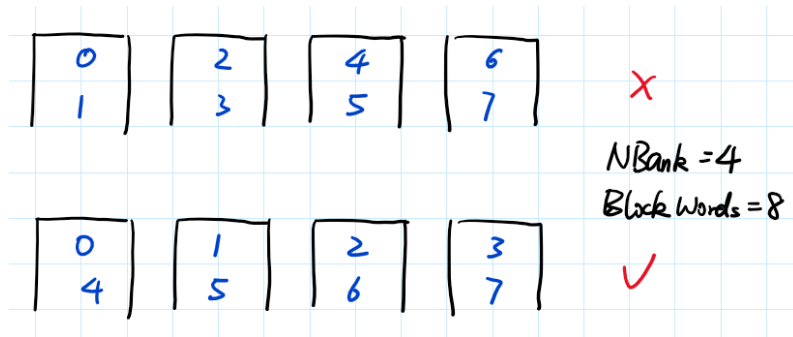


tag	setIdx	blockOffset	WordOffset
-----	--------	-------------	------------



# Data Access

- Multiple Banks to support concurrent accesses
- How does a Block distributed among Banks?



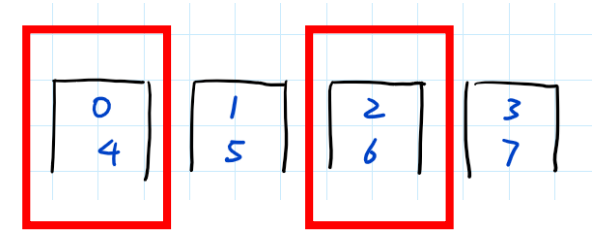
- one req one bank, for unit step accesses
- output  $BW(NBanks) < BlockWords$ , cause **“bank conflict”**

- To index within a block:
  - lower bits to select bank
  - higher bits to index within a bank

tag	setIdx	bankOffset	bankIdx	WordOffset
-----	--------	------------	---------	------------

# Arbiter

- Bank conflict?
  - assume a stride access with stride=2, base =0
  - this request is served using 2 cycles



- Arbiter detect the conflict, and decide which part of the original request should go straight, which part should reserve and wait.
- This procedure happens both on coreReq-coreRsp path, and memRsp-coreRsp path
  - so Arbiter has 2 sources: coreReq and MSHR

# Arbiter

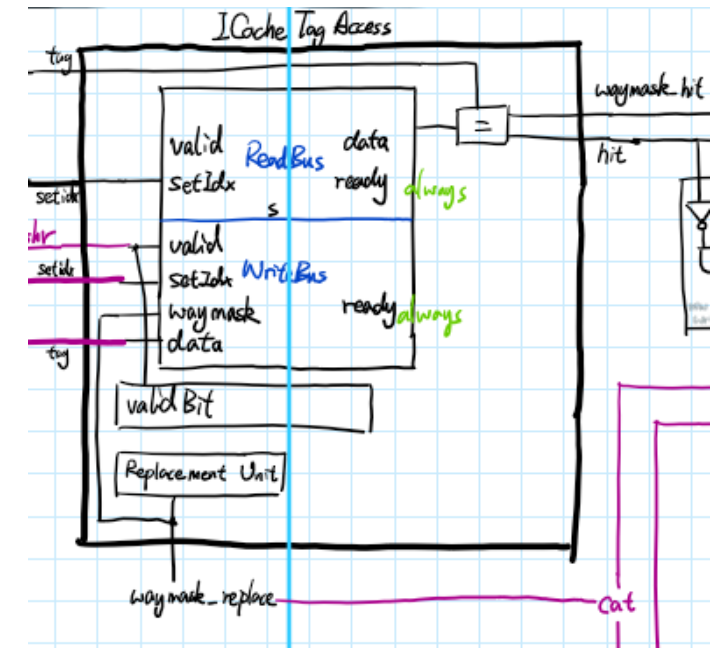
- Arbiter provide Data Array and Data Crossbar with control signal
- Pure combinational
- For Data Array:
  - Bank enable, bank offset, byte mask
  - These addr & ctrl also has order transformation, thus an Addr Crossbar is embedded in Arbiter
- For Data Crossbar

# Data Crossbar

- Transform between two data orders
  - Core-Mem
  - Mem-Core
  - NLanes = NBanks assumption takes effect here
- Composed of massive MUXs
  - $\text{NLanes} * (\text{WordLength})\text{bits}$  NLanes-in-1 MUX
- Control always from Arbiter
- Data source and sink varies

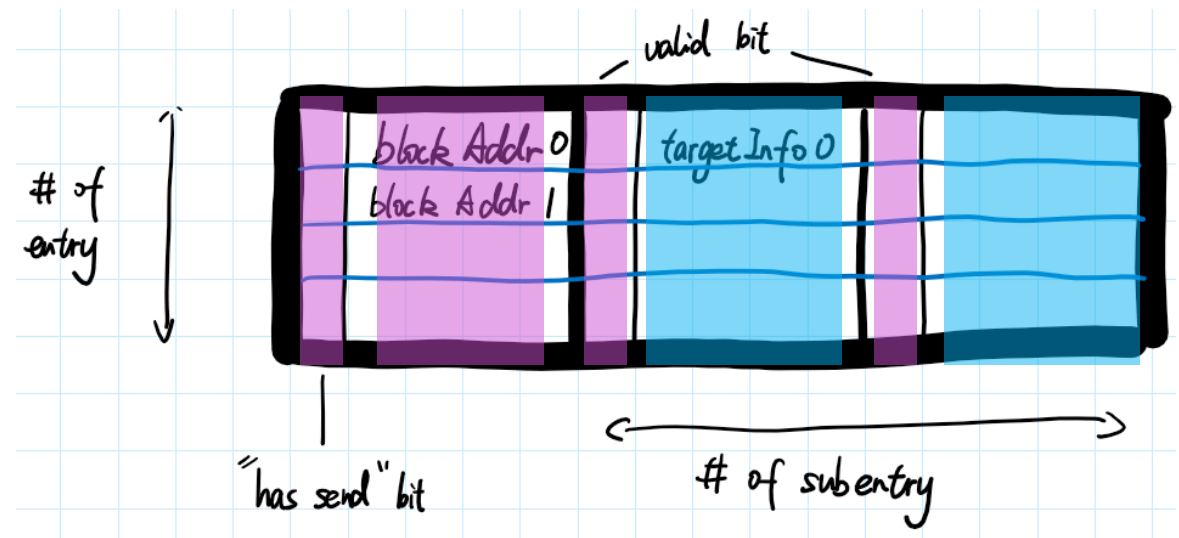
# Tag Access

- Record Tags of existing blocks in Data Access, so as to determine hit or miss
- Allocate way entries when miss requests return
  - employ replace unit
- Same to CPU data cache or instruction cache



# Miss Status Holding Register

- Record miss requests, and send them to L2
  - Merge unreturned L2 request if they aim at the same block
  - When miss response, resume unfinished requests
- 
- 4 independent ports
  - 2 coupled **flows**
  - 3 memory instances

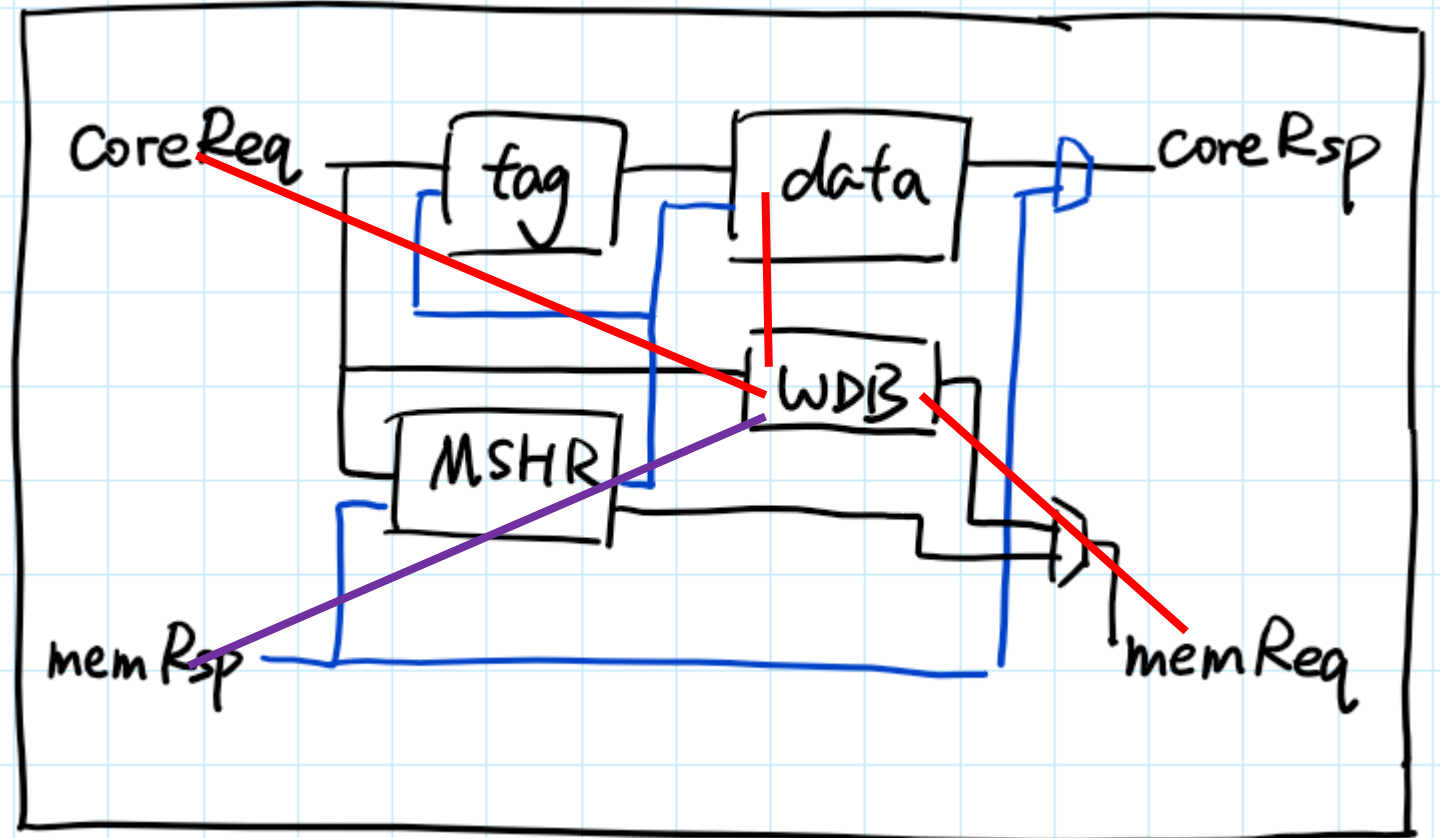


# Write Data Buffer

- Buffer the data of WRITE request and send it to L2 by memReq port
- Without sub-word WRITE, whether hit or miss, ease
- With a sub-word WRITE and hit, partially from coreReq, partially from DataAccess(old data)
- With a sub-word WRITE and miss, shift from write around to “write allocate”, WDB reserve new data at missReq, fill rest part with old data at missRsp, then send it by memReq

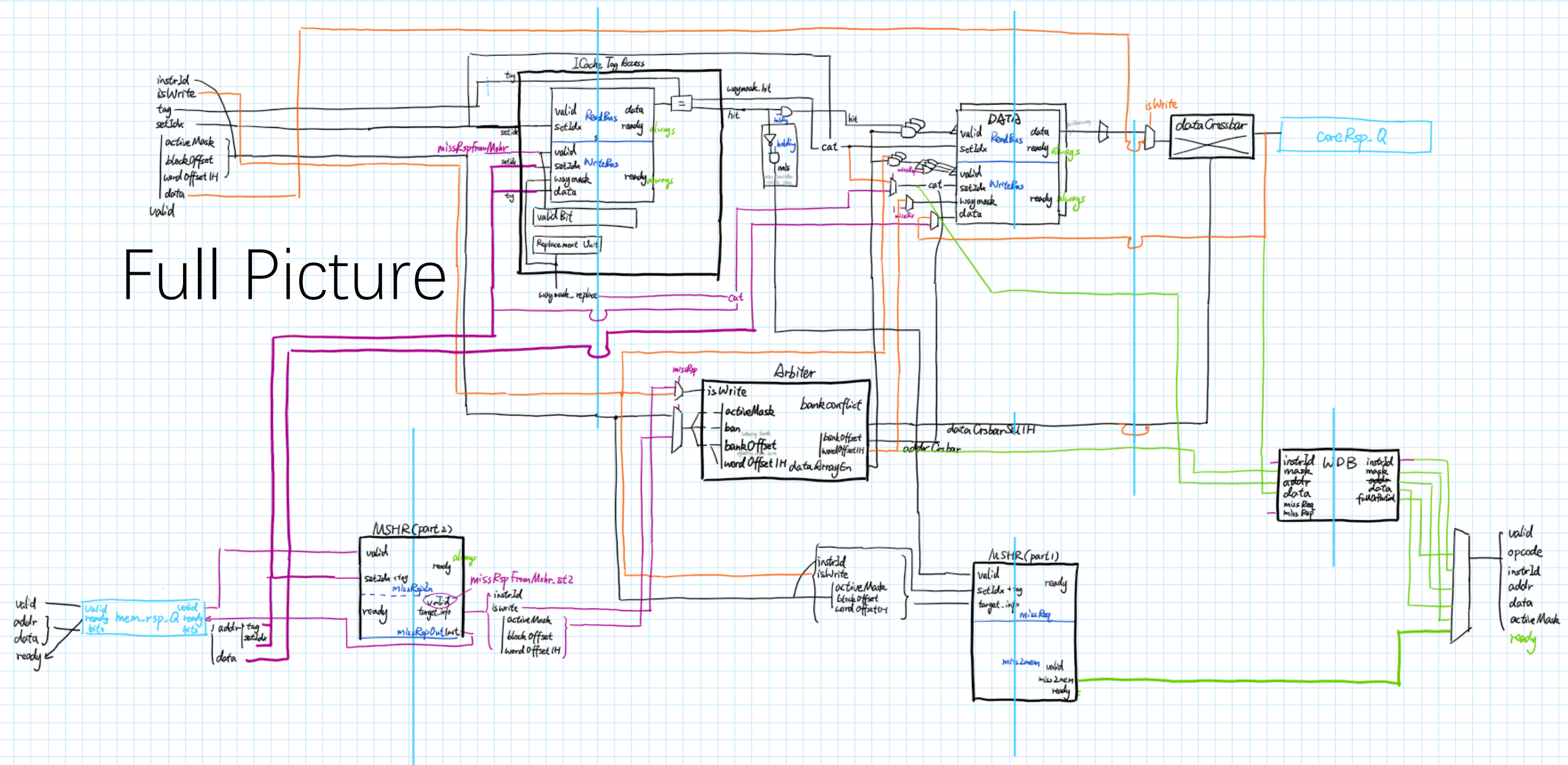
# Write Data Buffer

- ✓ • WRITE no sub-word
- ✓ • WRITE hit, sub-word
- ✓ • WRITE miss, sub-word





# Full Picture



# Thanks for listening

Hope this is not too suffering for everyone

