Discussion X: Cache

from a system designer's perspective

Jiahui Li lijh12025@shanghaitech.edu.cn

CAS4ET LAB & TOAST LAB &

SI²

Outline

1. Cache(s) on-the-fly
2. Mapping Rules
3. Cache Tiering
4. Replacement Policy

CS110

Cache(s) on-the-fly

- **What is a cache?** **[from Webster dictionary]**
  - a hiding place especially for concealing and preserving provisions or implements
  - a secure place of storage
    - *discovered a cache of weapons*
  - a computer memory with very short access time used for storage of frequently or recently used instructions or data
    - called also *cache memory*

CS110

Cache(s) on-the-fly

➢ Why do we need cache?

- Program/Service(s) run slowly without an **architectural design support**
  - As a designer (not programmer yet), we hope to provide a fast path for users
  - Cache as a mechanism, provided to users
- With cache, users could perform more efficiently with cache enabled
  - Programmer's program could take the new quick path
  - Who could resist a place of caching?
    - Grab some snacks from the garage, like those candies you stashed
    - Or drive miles to purchase at Sam's Club then drive back home to eat them
  - Even that fast path is forced (compared to raw design), programmers should think how to make their program on that path faster

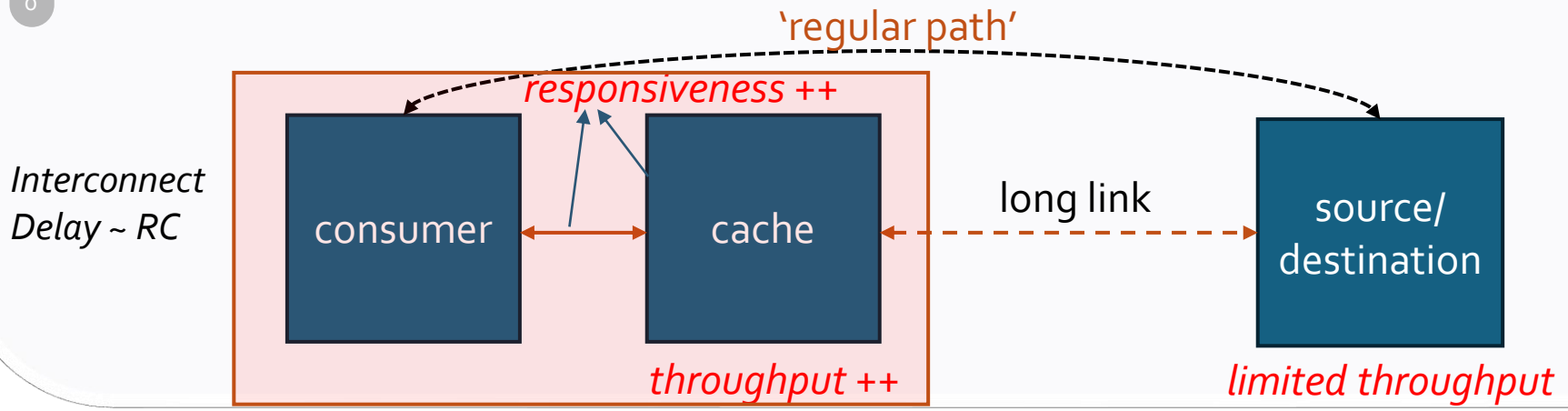
CS110

Cache(s) on-the-fly

➢ Cache Principles

- **Close enough to its consumer (cache is physical-entity backed)**
- **Store data as (main) memory does**
- **Fast enough compares to 'regular path'**
  - In an average sense, cache memory only maintain a set of copies of data in 'remote memory' (original source/destination), so the regular path is not eliminated

CS110



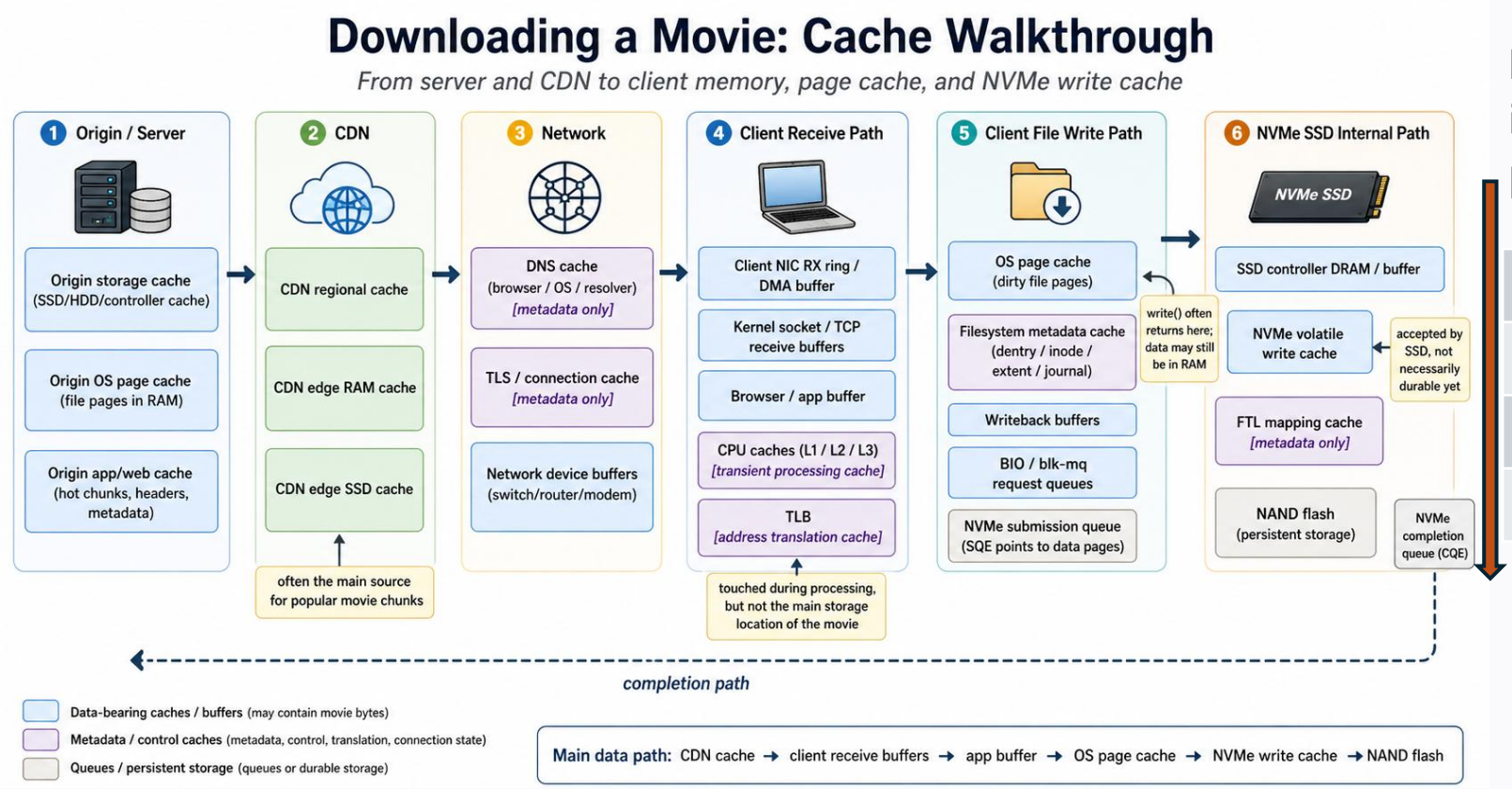
It's consumer who initiate a request, based on the direction of data transmission, the other side is either source or destination

Cache(s) on-the-fly

➢ Cache(s) walkthrough

- Scenario: downloading a movie

Figure for illustrative use, might not be accurate



Everything is built-on CPU cache as it's the backbone of high-performance CPU to access *memory*

CDN Cache	SW managed Storage
Page Cache	OS managed Memory
TLB	OS managed HW
CPU Cache	HW managed HW

CA Studies

xx managed HW means it's backed by specialized HW
(both logics and storage medium, it's a complex)

Cache(s) on-the-fly

Performance Monitoring Unit (built-in in your CPU)

➢ Cache(s) walkthrough

- **CPU Cache** (which the *cache* refer to in CA)

What your OS knows about your CPU's cache?

```
ljh@ljh-VMware-Virtual-Platform:~/Desktop$ lscpu | grep cache
L1d cache:                            192 KiB (4 instances)
L1i cache:                            128 KiB (4 instances)
L2 cache:                             5 MiB (4 instances)
L3 cache:                             24 MiB (1 instance)
```

Intel i7-11800H

```
ljh@ljh-VMware-Virtual-Platform:~/Desktop$ perf list pmu | grep cache
cache:
       [Counts the number of cache lines replaced in L1 data cache. Unit: cpu]
       [L2 cache lines filling L2. Unit: cpu]
       [Modified cache lines that are evicted by L2 cache when triggered by an
        L2 cache fill. Unit: cpu]
       [Non-modified cache lines that are silently dropped by L2 cache. Unit:
       [Demand Data Read access L2 cache. Unit: cpu]
       [RFO requests to L2 cache. Unit: cpu]
       [L2 cache hits when fetching instructions,code reads. Unit: cpu]
       [L2 cache misses when fetching instructions. Unit: cpu]
       [Demand Data Read requests that hit L2 cache. Unit: cpu]
       [Demand Data Read miss L2 cache. Unit: cpu]
       [Read requests with true-miss in L2 cache. Unit: cpu]
       [All accesses to L2 cache. Unit: cpu]
       [RFO requests that hit L2 cache. Unit: cpu]
       [RFO requests that miss L2 cache. Unit: cpu]
       [SW prefetch requests that hit L2 cache. Unit: cpu]
       [SW prefetch requests that miss L2 cache. Unit: cpu]
       [L2 writebacks that access L2 cache. Unit: cpu]
```

Intel Processor Events Reference

Intel® VTune™ Profiler provides a set of hardware event-based analysis types that help you estimate how effectively your application uses hardware resources. These analysis types monitor hardware events supported by your system's *Performance Monitoring Unit (PMU)*. The PMU is hardware built inside a processor to measure its performance parameters such as instruction cycles, cache hits, cache misses, branch misses and many others.

NOTE:
For more information on Intel® 64 and IA-32 architectures, explore *Intel Software Developer Manuals* available at https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html.

For details on hardware events supported by your system's PMU, use any of the following options:

- When adding new events to your custom configuration, select an event in the table and explore its short description, or click the **Explain** button to open the *Intel Processor Events Reference* for more details:

Events configured for CPU: Intel(R) Processor code named Skylake ULT

NOTE: For analysis purposes, Intel VTune Amplifier 2018 may adjust the Sample After values in the table below by a multiplier. The multiplier depends on the value of the Duration time estimate option specified in the target configuration window.

Search... Explain

Event Name	Sample After	Description
CPU_CLK_UNHALTED.THREAD	2400000	Core cycles when the thread is not in ...
CPU_CLK_UNHALTED.REF_TSC	2400000	Reference cycles when the core is not...
INST_RETIRED.ANY	2400000	Instructions retired from execution.
MEM_TRANS_RETIRED.LOAD_LATENCY...	10003	Counts loads when the latency from fir...
CYCLE_ACTIVITY.STALLS_L1D_MISS	2000003	Execution stalls while L1 cache miss d...
CYCLE_ACTIVITY.STALLS_L2_MISS	2000003	Execution stalls while L2 cache miss d...
CYCLE_ACTIVITY.STALLS_L3_MISS	2000003	Execution stalls while L3 cache miss d...
CYCLE_ACTIVITY.STALLS_MEM_ANY	2000003	Execution stalls while memory subsys...
EXE_ACTIVITY.1_PORTS_UTIL	2000003	Cycles total of 1 uop is executed on al...
EXE_ACTIVITY.2_PORTS_UTIL	2000003	Cycles total of 2 uops are executed on...
EXE_ACTIVITY.BOUND_ON_STORES	2000003	Cycles where the Store Buffer was full...
EXE_ACTIVITY.EXE_BOUND_0_PORTS	2000003	Cycles where no uops were executed,...

Cache(s) on-the-fly

➢ Cache(s) walkthrough

- **CPU Cache** (which the *cache* refer to in CA)

1. CPU cache sits at LSU (Load/Store Unit) path

- so it takes memory address as its input, and may return valid data

2. CPU cache starts at cold (why compulsory miss)

- it's volatile, just likes main memory
- it's capacity is much smaller than main memory, it's not a 1:1 replica
- data cached dynamically

Job: To design a high speed memory, which *play* as the main memory, aligning with the memory model. But to achieve these at a limited capacity (but at the same addressing space!)

We have much design space (mapping, tiering, replacement,…)

CS110

Mapping Rules

➢ Why mapping?

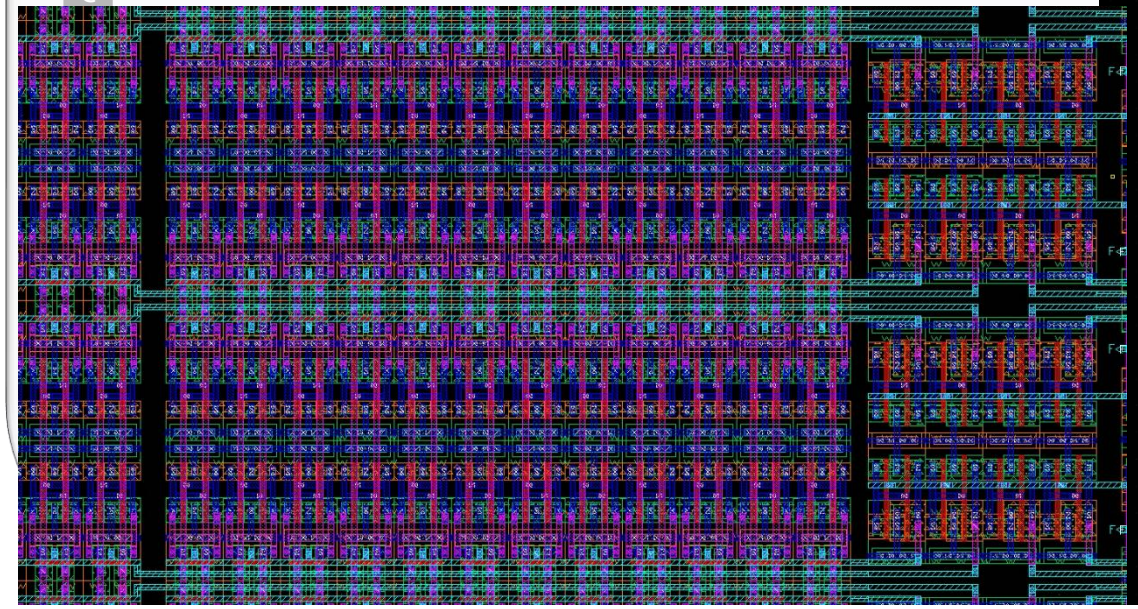
- All stored data needs to be assigned a physical location
- Program/OS handle address to storage device
  - Storage device's controller to decide where to store/fetch
  - Storage device is built of basic storage *blocks*, i.e. byte-addressable
- With building block's param in mind, could design better access model
  - Memory access pattern should maximize device's limits

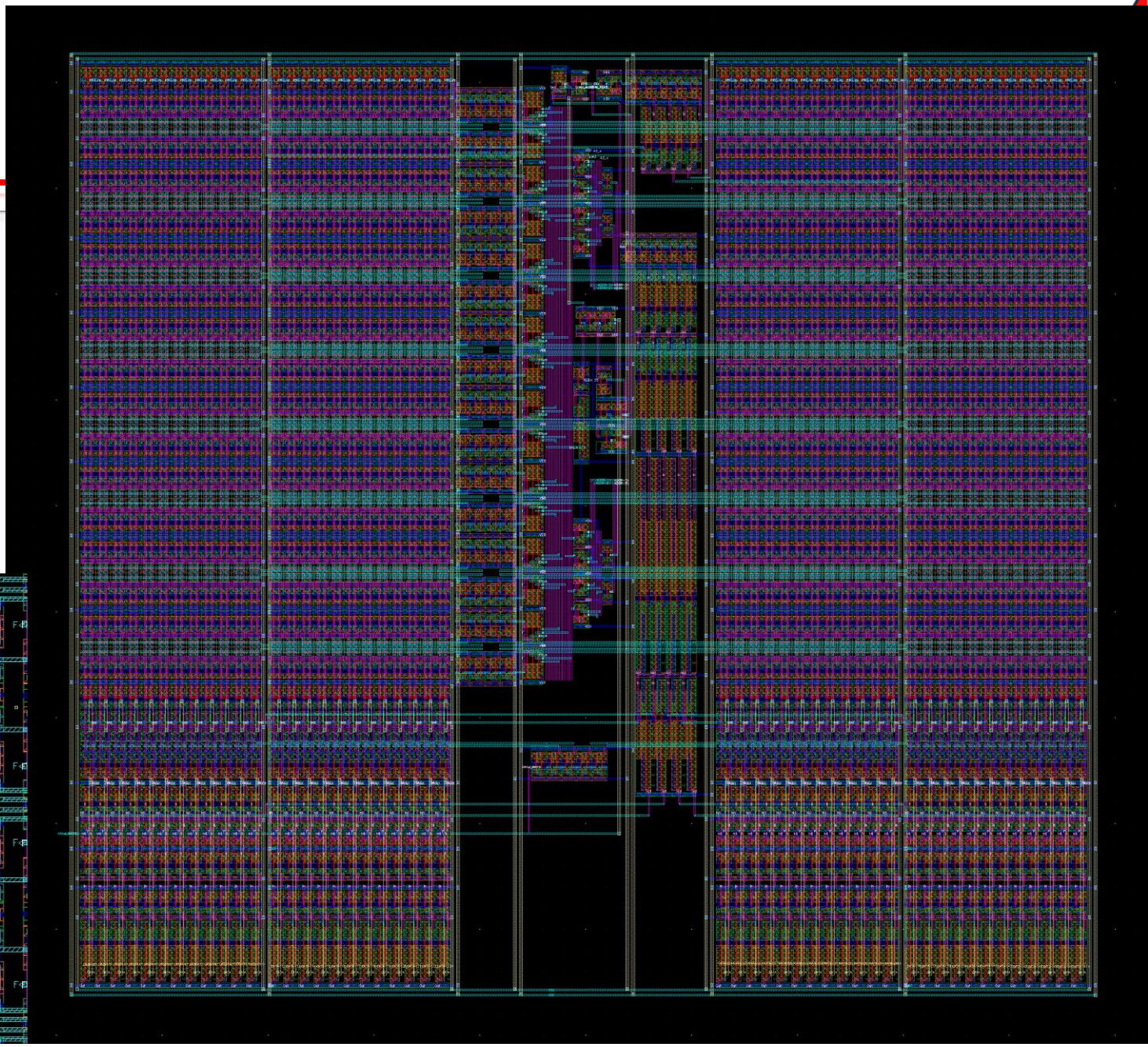
Btw. We are not talking about address translation, i.e. virtual memory's VA to PA, SSD's internal indirection. After these translations, how ***address (util) and their associated data (goal) map to real physical medium*** *is indispensable*

CS110

Mapping Rules

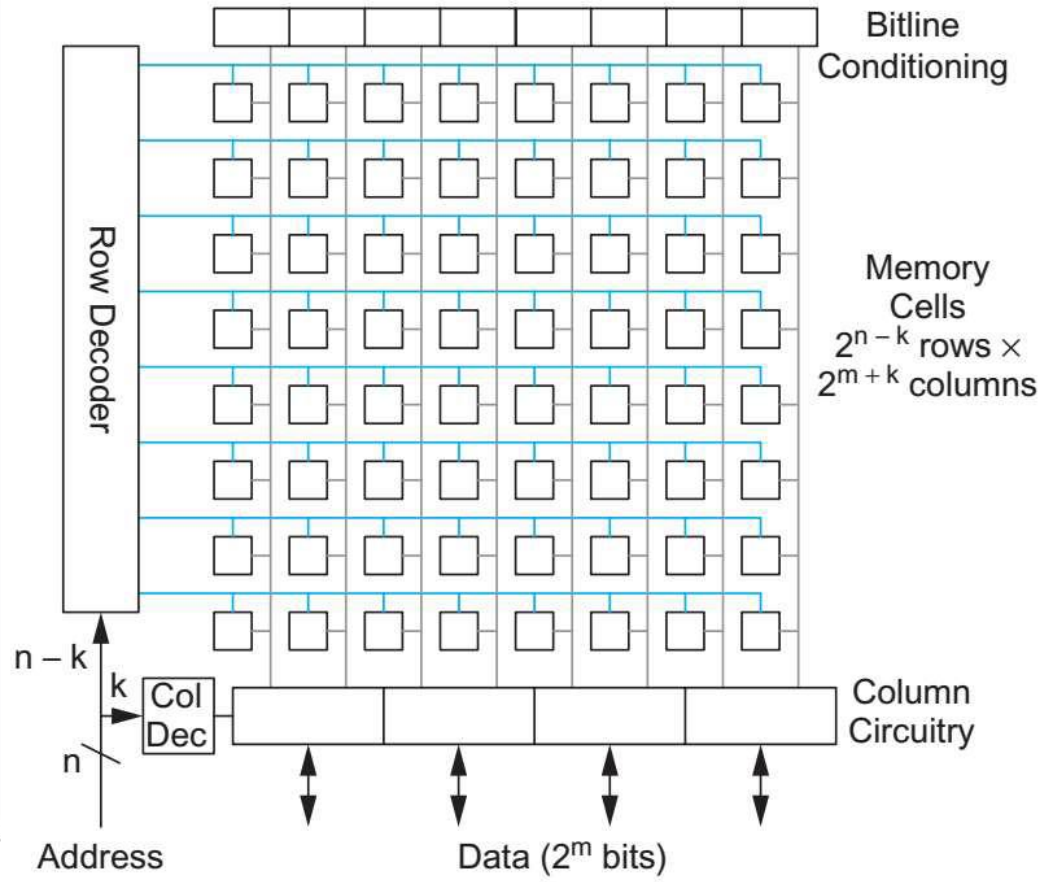
- **Basic Building Blocks**
  - 48*16 SRAM (from EE213 Prj)
  - Here SRAM = 6T SRAM Cell +
    Row decoder +
    Sense Amplifier (on Cols) +
    Controller

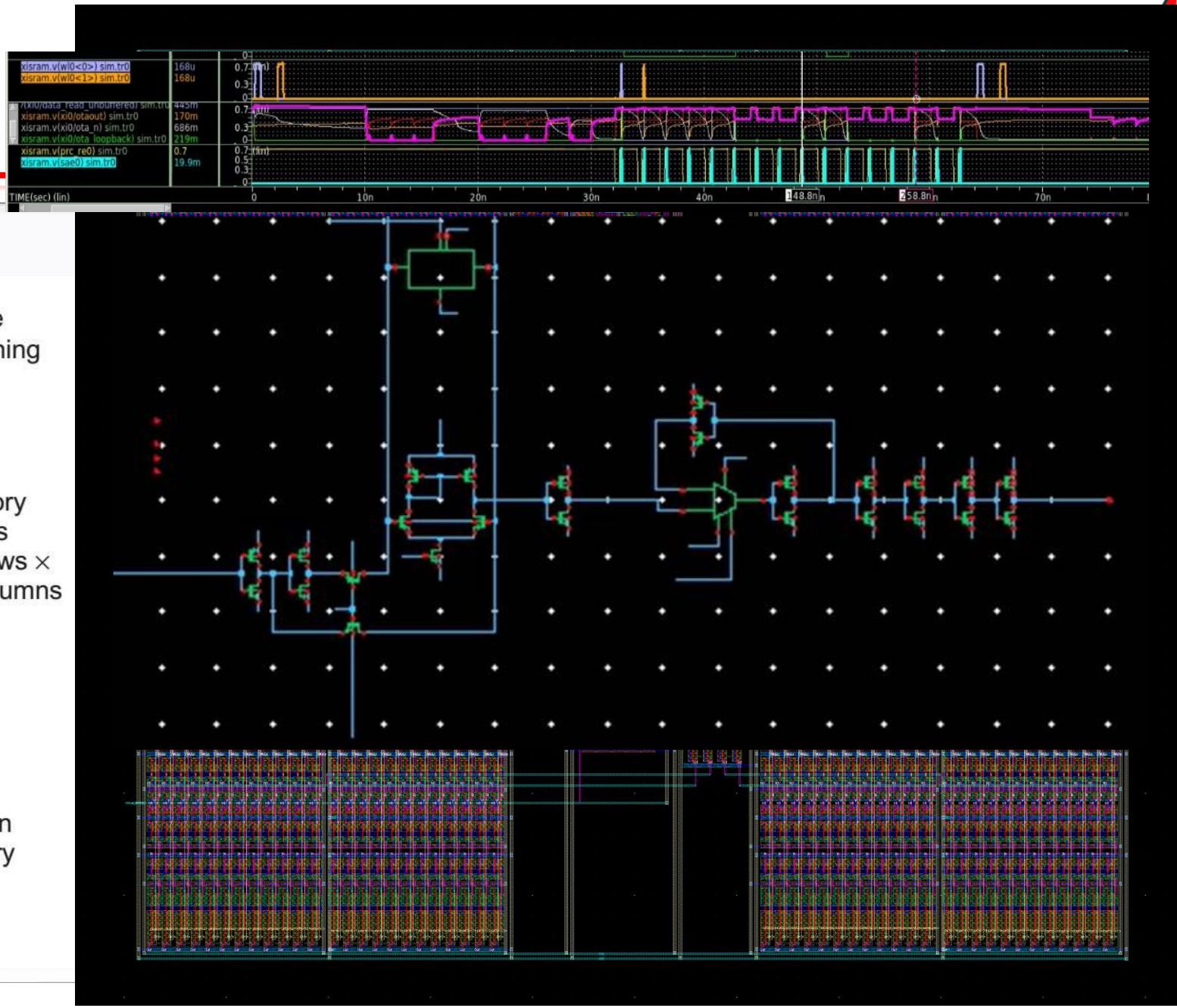




Mapping Rules

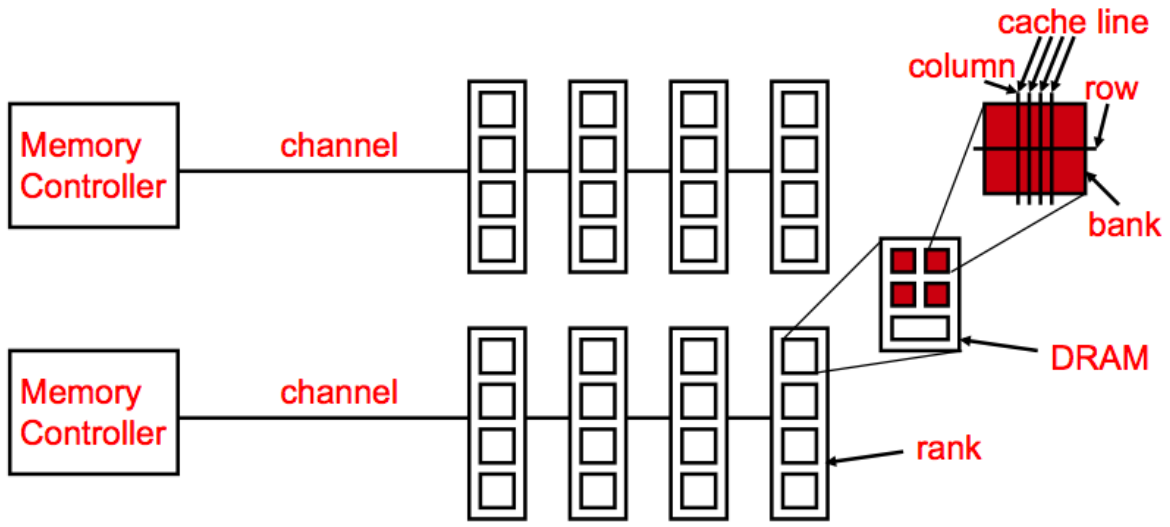
- **Basic Building Blocks**





Mapping Rules (RAM)

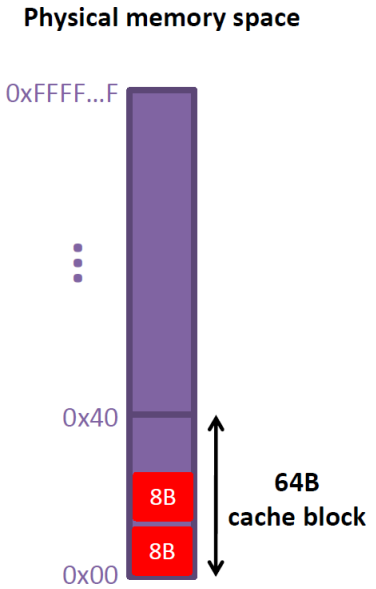
- Need pay attention to bus specs too
  - Banks share command/address/data buses
  - The chip itself has a narrow interface

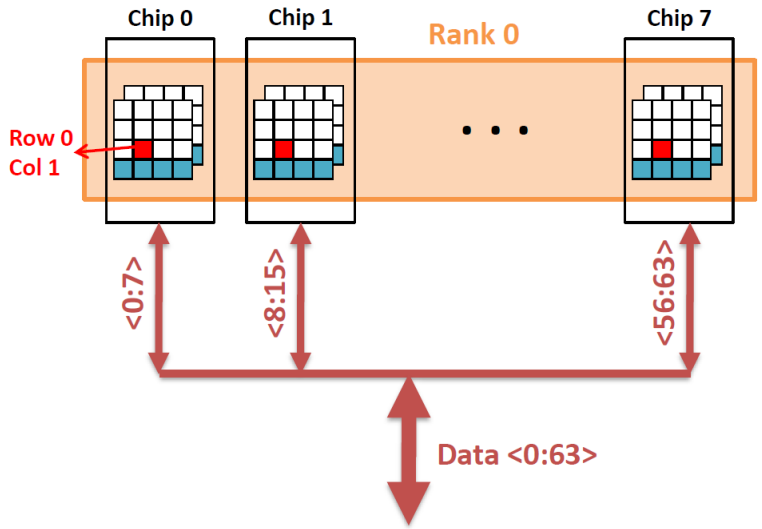


Latency = Medium access time + Bus IO time (bottleneck)

DIMM (Dual in-line memory module)

Example: Transferring a cache block

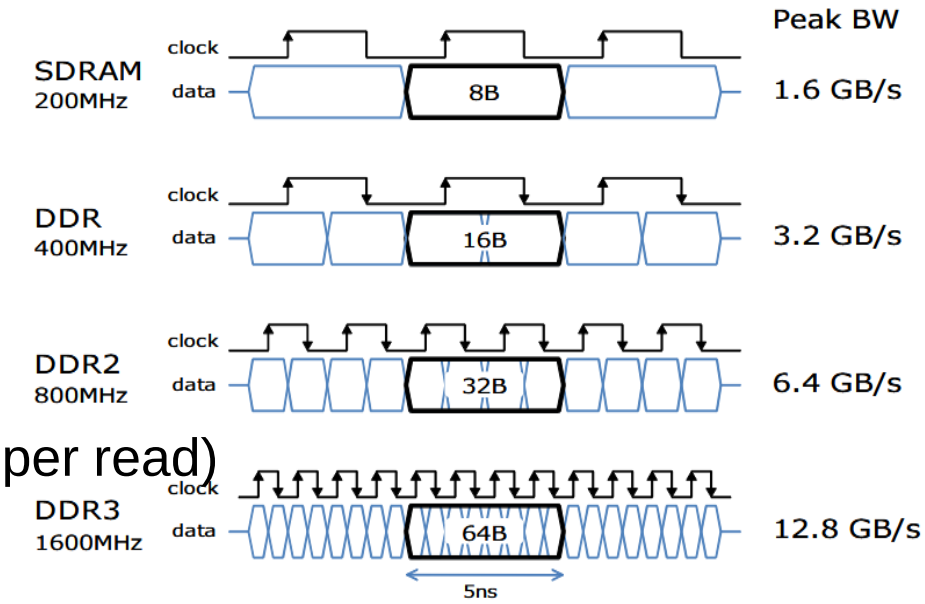




A 64B cache block takes 8 I/O cycles to transfer.

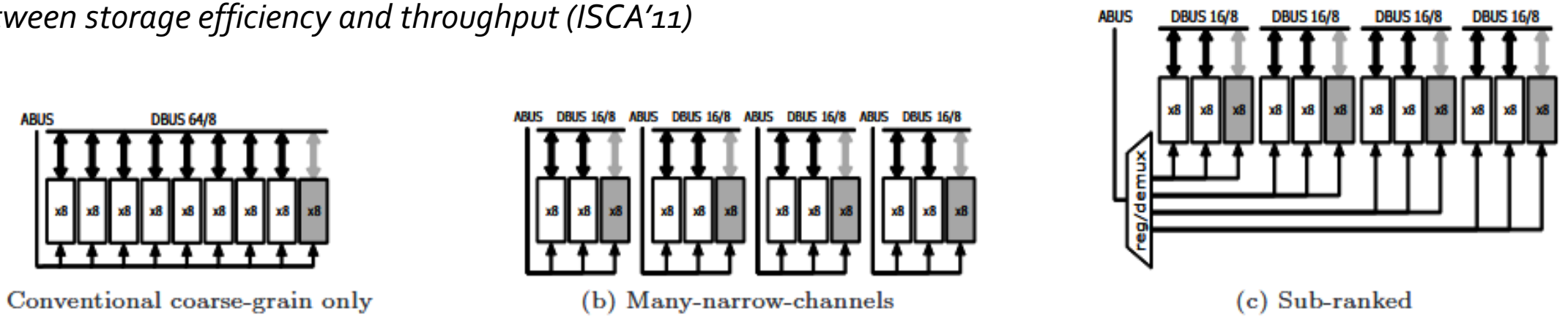
During the process, 8 columns are read sequentially.

Mapping Rules (RAM)



- Need pay attention to bus specs too
  - Banks share command/address/data buses
  - The chip itself has a narrow interface (4 -16 bits per read)

Figures from *Adaptive granularity memory systems: A tradeoff between storage efficiency and throughput (ISCA'11)*



(a) Conventional coarse-grain only (b) Many-narrow-channels (c) Sub-ranked

Figure 2: Comparison of a conventional memory system, a many-narrow-channels approach, and a sub-ranked memory system similar to MC-DIMM. The many-narrow-channels and sub-ranked in this figure provide 16B access granularity with DDR3. Gray boxes and arrows represent ECC storage and transfers, which increase significantly in (b) and (c) as a result of finer access granularity support. *ABUS* represents address/command bus and *DBUS X/Y* represents data bus, where *X* bits are for data and *Y* bits are for ECC.

Mapping Rules

Figs Source:CMOS VLSI Design A Circuits and Systems Perspective (Weste, Neil H EHarris, David Money)

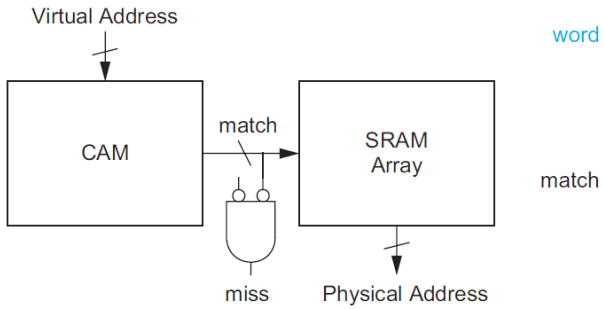


FIGURE 12.68 Translation Lookaside Buffer (TLB) using CAM

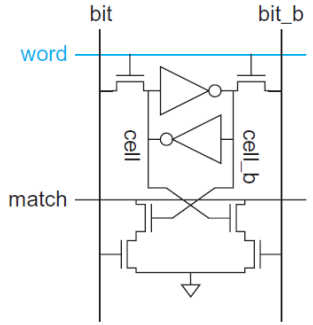
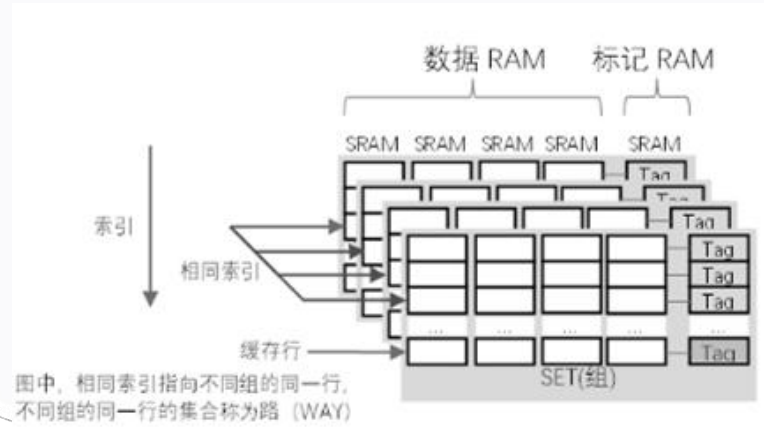


FIGURE 12.69 CAM cell implementation

➢ At a higher level

- Single SRAM only suits for **direct-mapped**
  - Not flexible, store cached data AS-IS (static bookkeeping SRAM arrays)
  - Accessing path is fixed
- *On-demand Caching* requires additional lookup HWs
  - **FA (Fully Associated)** & **SA (Set Associated)** add degree of freedom to lookup
  - Indirection occurs, need 'translation' -- at very low-level HW



注：在编写这部分内容参考相关资料时，对高速缓存相关术语的解释不尽相同，尤其是路和组的概念，为了更准确的说明这些术语，专门分析了龙芯1B处理器的底层HDL设计以及MIPS授权开源的MicroAdaptive IP的HDL设计，明确了路和组的概念。

lookup
- CAM (Content-Addressable Memory, 9T+)
- **Double RAM Access (Tag + Data)**

中文资料来源：微型计算机系统原理及应用：国产龙芯处理器的软件和硬件集成（何宾，2022）

Mapping Rules (Associativity)

Double RAM Access (Tag + Data)

➢ Cache Coloring

- Used in OS's memory management
  - This shows how to optimize performance of SW by taking leverage of cache
  - Use (associative) cache wisely
- OS manage memory in *page*
  - *Page* is the similar concept of cache *block* (cache *line*) in SW's view
  - To avoid internal defragmentration, a page could not be too big (typically 4kB)
  - Do the math: [a *4-way* set associative $ has a size of 64kB → 256 set]
    - $ line = 64B, 6 bit
    - A page requires 4k / 64 = 64 $ lines, 6 bit
    - We have 8 bit to store $ lines, 2 bit left (coloring bit)

CS110

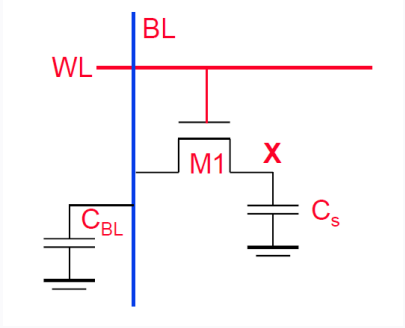
Cache Tiering

➢ Cache Behavior

- Allocation (to bring back data to this tier from next tier on a $ miss)
  - For write: decide on if the cache use Write-Allocation scheme
  - For **read**: decide on if the cache use Read-Allocation scheme
- Modification (a $ line becomes stale)
  - Caused by **write**
  - Flush to next tier or not depends on it's Write-Through or Write-Back
- Eviction
  - When current tier $ is FULL, either Allocation or Modification triggers an eviction
  - Eviction could propagate among tiers due to Allocation and Modification schemes

CS110

Cache Tiering



➢ On Cache Miss

- Write-Back cache
  - Mainly as write buffer, flush on demand
  - Ultimate goal: never touches DRAM (1.energy efficiency, 2. *update write* friendly)
- Read cache
  - Previous Read-Allocated data got evicted (as a write to next tier $)
  - We need that data again
  - L1 size < L2 size < L3 size: wither fewer conflicts in next tier, it hopefully resides (one more important reason: L2 and L3 typically use associative cache)
  - Ultimate goal: never touches DRAM, again (a read to DRAM cost more than write)

CS110

Replacement Policy

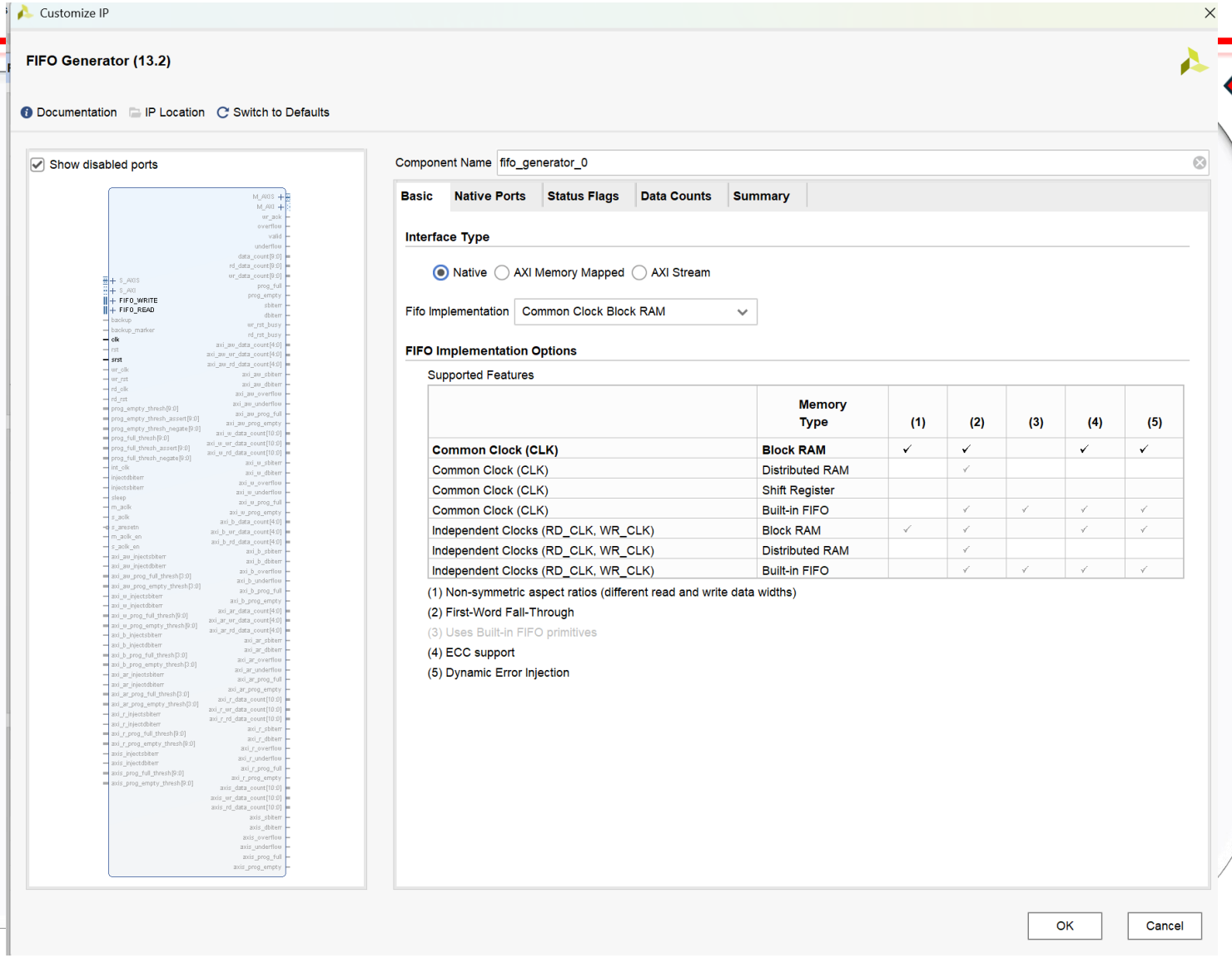
```
Data_Cache_Slot_Type Data_Cache_Flash::Evict_one_dirty_slot()
{
    asser  lots.size() > 0);
    auto it  lru_list.rbegin();
    while (it  = lru_list.rend()) {
        if ((*i  .second->Status =  ache_Slot_Status::DIRTY_NO_F
            break
        }
        itr++;
    }

    Data_Cache_Slot_Typ  v  ed_item = *lru_list.back().second;
    if (itr == lru_li  rend(
        evicted_ite  tatus = C  e_Slot_Status::EMPTY;
        return ev  ed_item;
    }

    slots.e  e(lru_list.back().first);
    delet  ru_list.back().second;
    lru_list.pop_back();

    return evicted_item;
}
```

- **Policy & Mechanism**
  - Policy decide what (to evict)
  - Mechanism decide how
- **CPU $ is managed by HW!**
  - HW implementation of replacement algorithm
    - Could not manage in linked-list
    - Decision must be instantaneous
  - Replacement decision from circuit

Replacement Policy

➢ Replacement Algorithm

- **FIFO (First In, First Out)**
  - Easy to implement and control
  - Bélády's anomaly
    - Larger size, Lower hit ratio
- **LRU (Least Recent Used)**
- **Random**



Replacement Policy

➢ Replacement Algorithm

- **FIFO (First In, First Out)**
  - Easy to implement and control
  - Bélády's anomaly
    - Larger size, Lower hit ratio
- **LRU (Least Recent Used)**
  - Bélády's MIN algorithm (OPT)
    - Not HW achievable, but as baseline
    - Requires information of future
  - Better temporal locality
    - Scan resistant (cuz it's stateful)
- **Random**

CS110

Policy	Predictor Structures	Cache Meta-data	Hardware Budget
LRU	None	16KB	16KB
DRRIP	8 bytes	8KB	8KB
SHiP	4KB SHCT 2KB PC tags	8KB	14KB
SDBP	8KB sampler 3KB predictor	16KB	27KB
Hawkeye	12KB sampler 1KB OPTgen 3KB predictor	12KB	28KB

Table 4: Comparison of hardware overheads.

Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement (ISCA' 16)

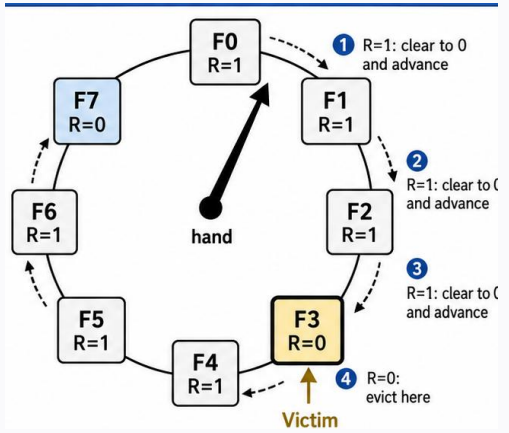
Replacement Policy

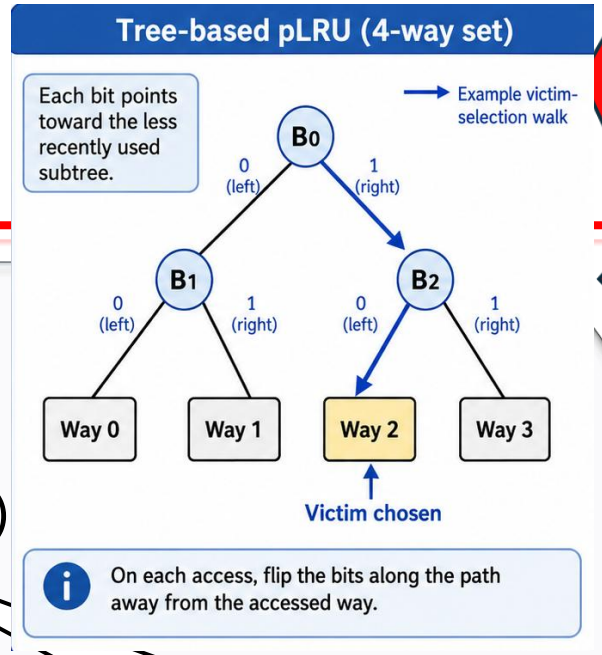
➢ Replacement Algorithm

Real LRU → **pLRU (use a tree)**

CLOCK (use 'minute hand')

- **FIFO (First In, First Out)**
  - Easy to implement and control
  - Bélády's anomaly
    - Larger size, Lower hit ratio
- **LRU (Least Recent Used)**
  - Bélády's MIN algorithm (OPT)
    - Not HW achievable, but as baseline
    - Requires information of future
  - Better temporal locality
    - More robust (cuz it's stateful)
- **Random**





Still too complex to implement in HW

On hit: set R=1

On replacement: scan circularly