Trying to design a simple yet efficient L1 cache

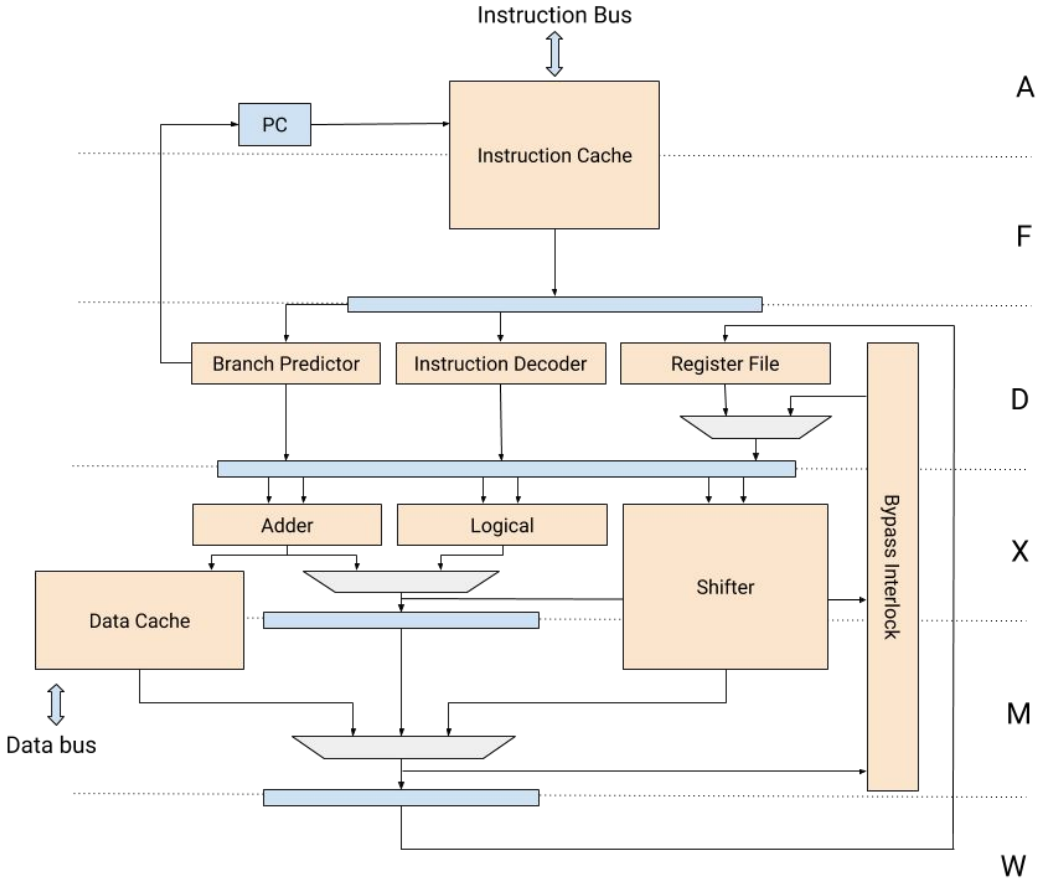
Jean-François Nguyen

Background

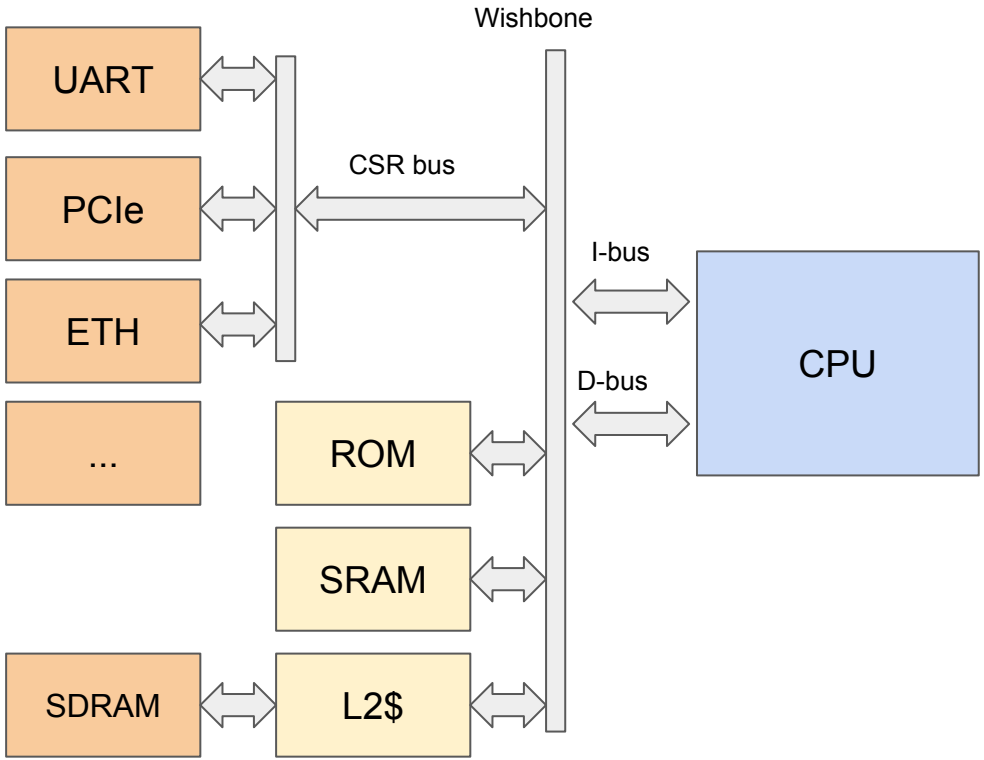
- Minerva is a 32-bit RISC-V soft CPU
- It is described in plain Python using nMigen
- FPGA-friendly
- Designed for reasonable performance in embedded use cases

This talk will discuss our approach to improve our current biggest performance bottleneck: memory access latency.

Minerva

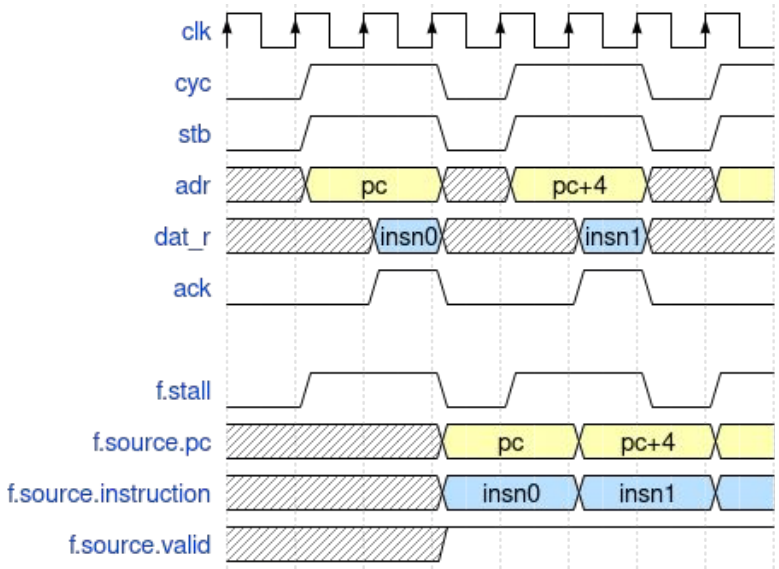

Instruction Bus
A
PC
Instruction Cache
F
Branch Predictor
Instruction Decoder
Register File
D
Adder
Logical
X
Shifter
Bypass Interlock
Data Cache
Data bus
M
W

MiSoC/LiteX

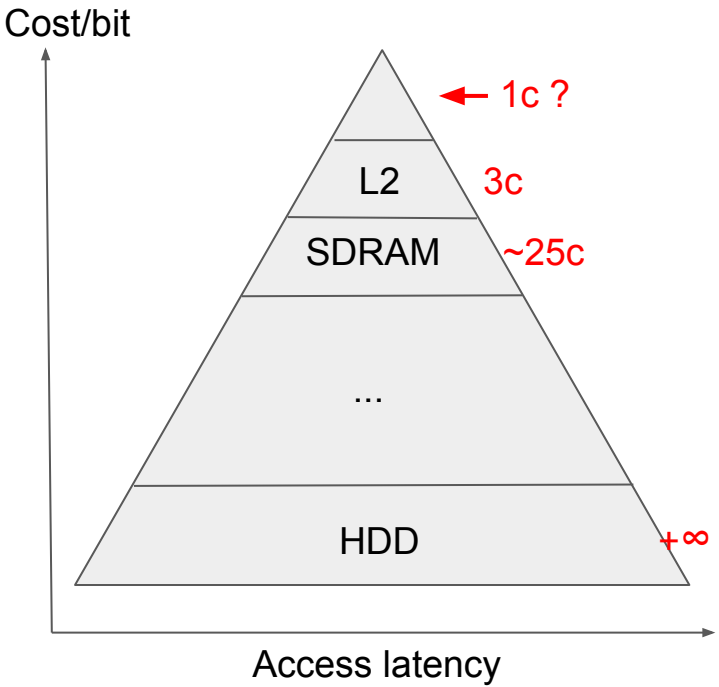

Wishbone
UART
PCIe
ETH
...
SDRAM
CSR bus
ROM
SRAM
L2$
I-bus
D-bus
CPU

Memory access latency

- Memory accesses must stall the CPU pipeline until completion.
- Direct accesses to FPGA BRAM take 1 clock cycle, but are size constrained.
  - XC7A35T has ~1.8M of BRAM
- Standard Wishbone transactions take 2 cycles (+1 to register).
  - throughput = 3c, latency = 18c



Memory as a hierarchy

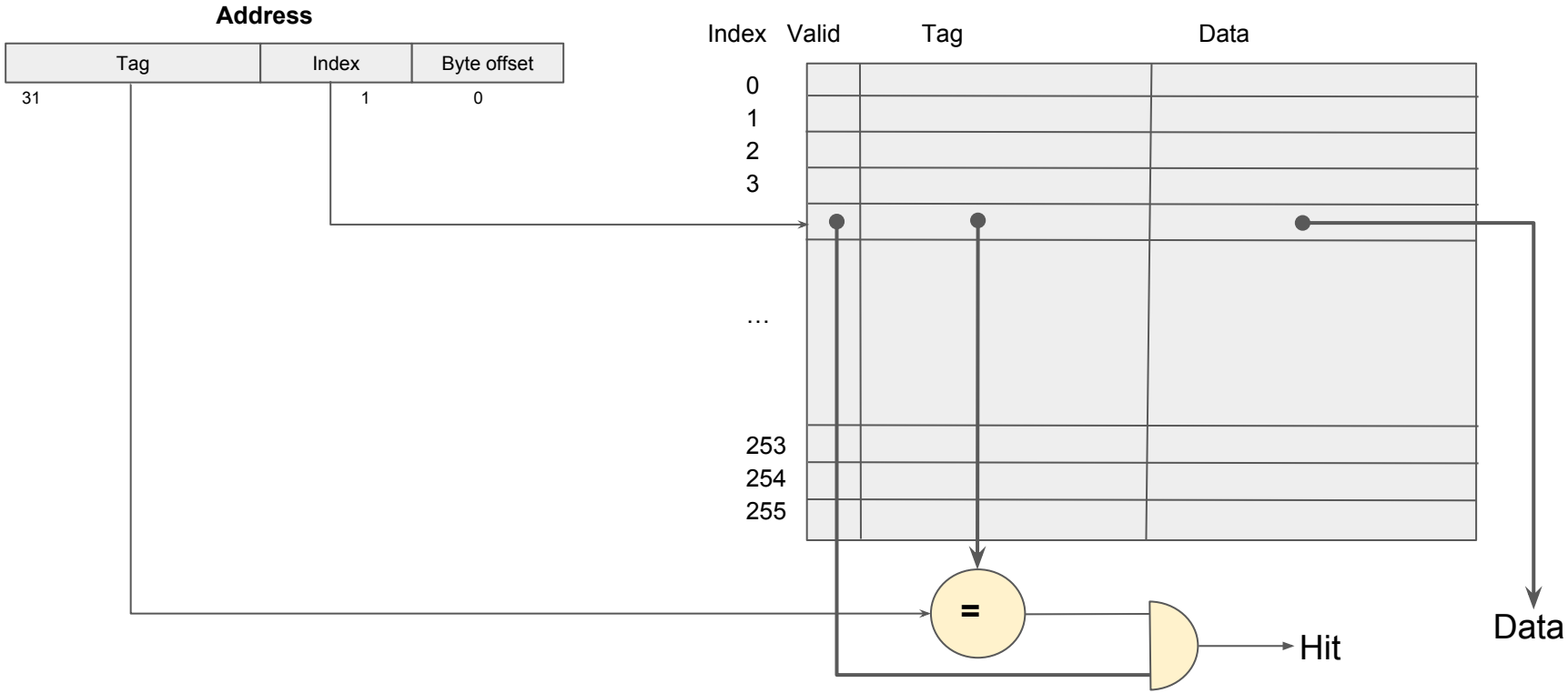


Step 1: basic cache design

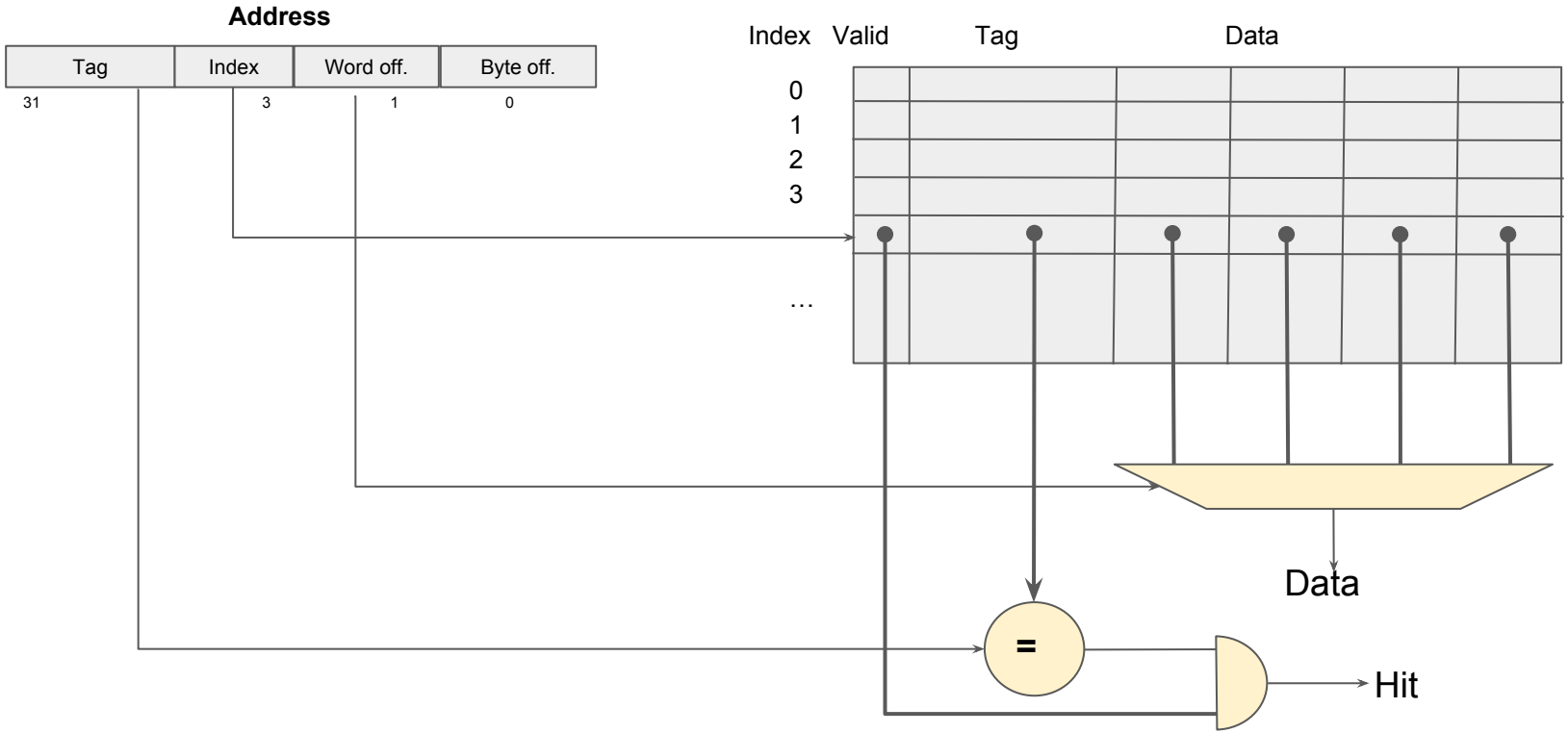
Goal: exploiting the memory hierarchy

- Provide the illusion of a large and fast memory
- Temporal locality:
  - If an item is referenced, it will tend to be referenced again soon.
- Spatial locality:
  - If an item is referenced, nearby items will tend to be referenced again soon.

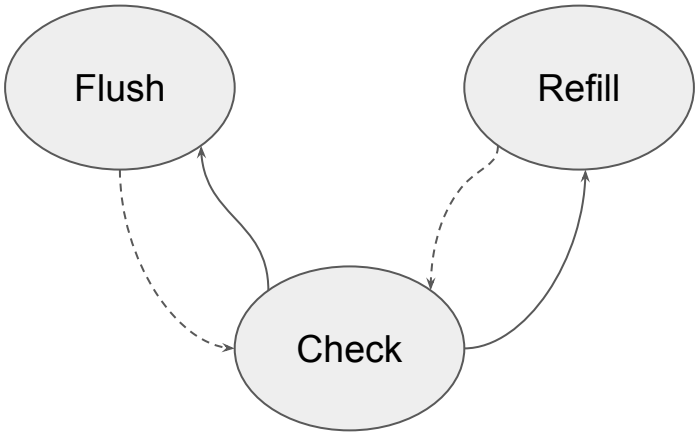
Accessing a cache



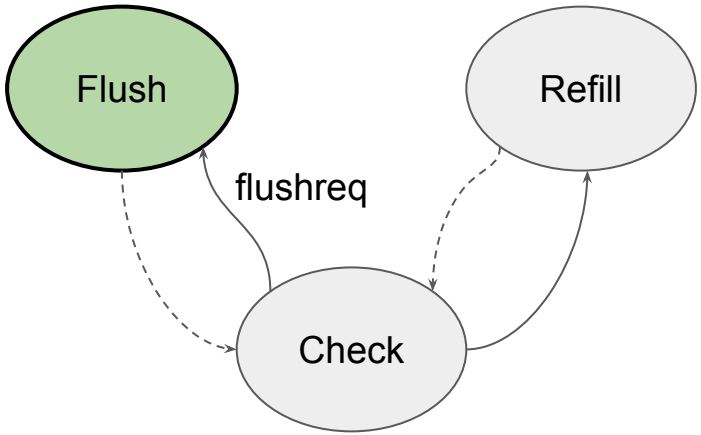
Multiple words in a line



Cache control FSM



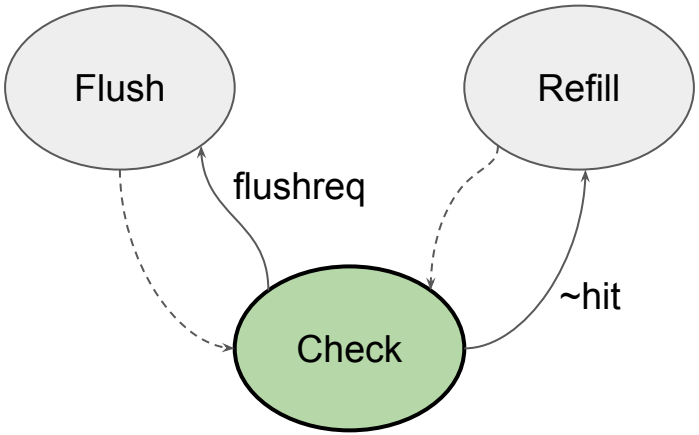
Cache control FSM: Flush



Flush the cache.
This state is entered on reset.

- Stall the pipeline
- Walk over each line to clear the valid bit

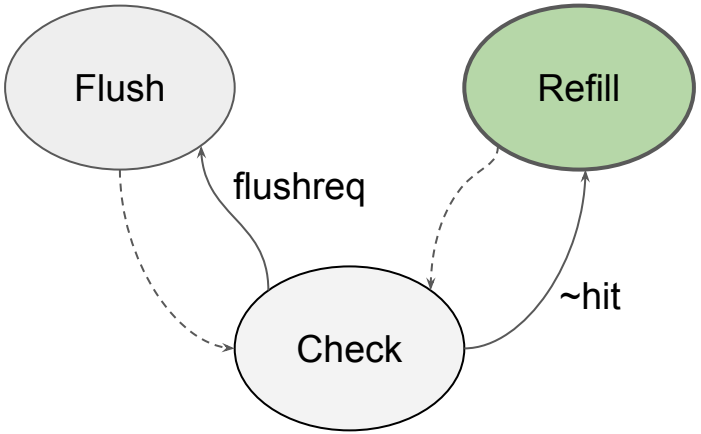
Cache control FSM: Check



Check for cache misses.

- In case of a miss, send a refill request and go to “Refill”.
- In case of a flush request, go to “Flush”.

Cache control FSM: Refill



Operate cache refills.

- Stall the pipeline.
- Wait for data from memory
- Set data, tag and valid bits
- Resume execution

Handling write accesses

- Write-through
  - Always write data to both memory and cache
  - Cost-effective when paired with a write buffer
  - Simple but slow

- Write-back
  - Only write data to cache. Memory is updated on refills
  - Cache contents are not consistent with memory
  - Faster, but more complex

Step 2: Optimizations

Improving cache performance

Average memory access time:

Hit latency **+** miss rate ***** miss penalty

Two strategies:

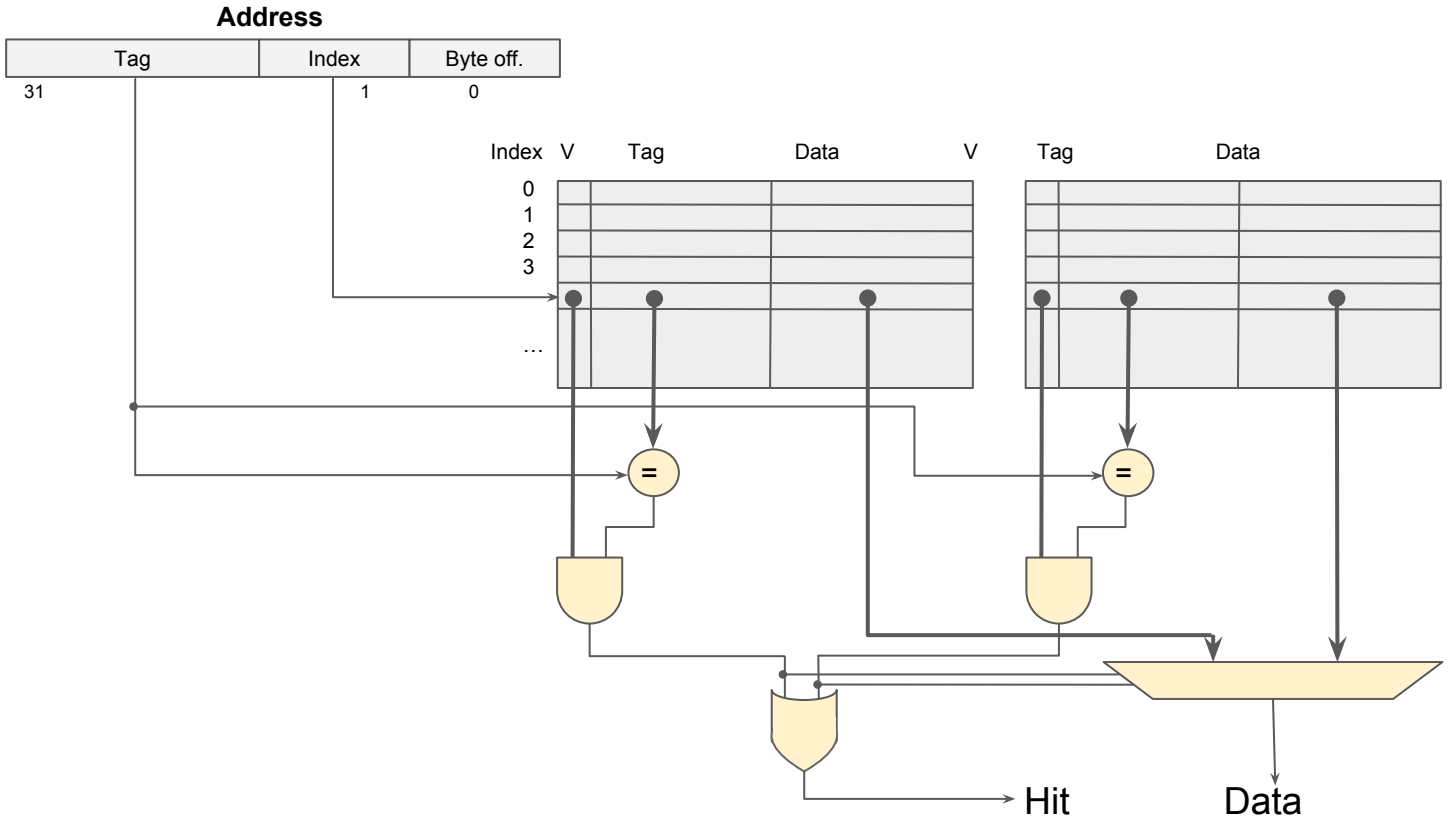
- Reduce miss rate
- Mitigate miss penalties

Associativity

Associativity is the number of locations where a cache line can be placed.

- Direct-mapped
  - A line can be placed in exactly **one** location.
- n-way associative
  - A line can be placed in **n** locations.
- Fully associative
  - A line can be placed in **any** location.

Example: 2-way associativity



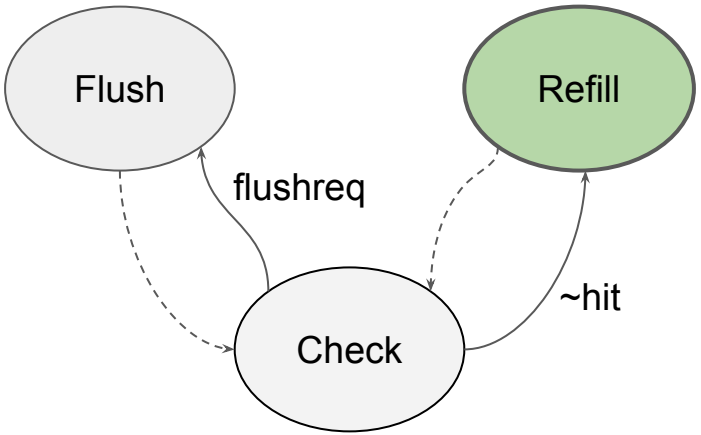
Refill policy

In a n-way associative cache, n lines can match a given index.
In case of a refill, which line should we choose to replace ?
Multiple choices:

- pseudo-LRU (Least Recently Used)
- MRU
- Pseudo-random
- ...

In a 2-way associative cache, Round-Robin is equivalent to LRU.

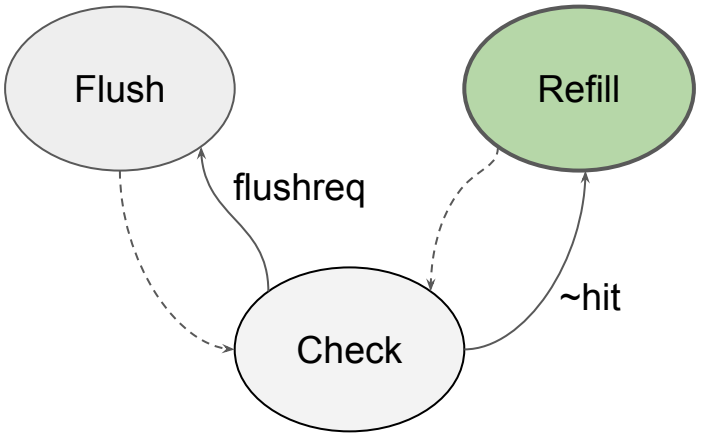
Early restart



- Stall the pipeline
- Wait for the requested word
- Un-stall the pipeline but continue refilling

What happens if we miss again ?

Early restart



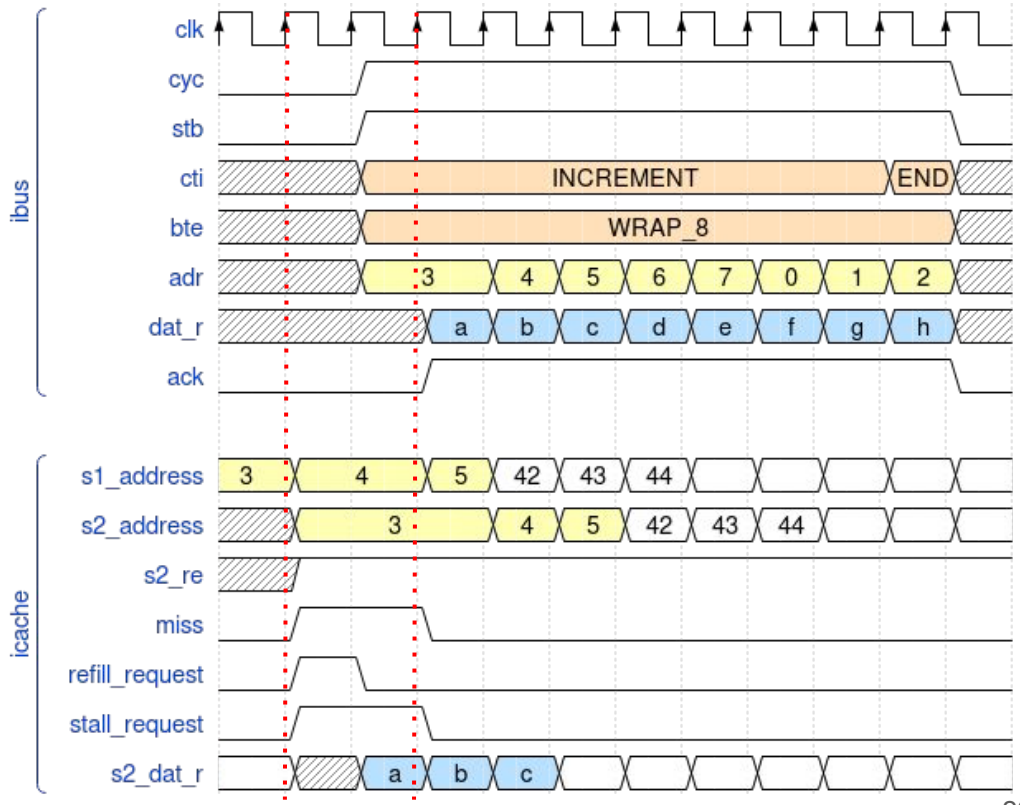
- Stall the pipeline
- Wait for the requested word
- Un-stall the pipeline but continue refilling

What happens if we miss again ?

- Stall the pipeline again
- Wait for the requested word

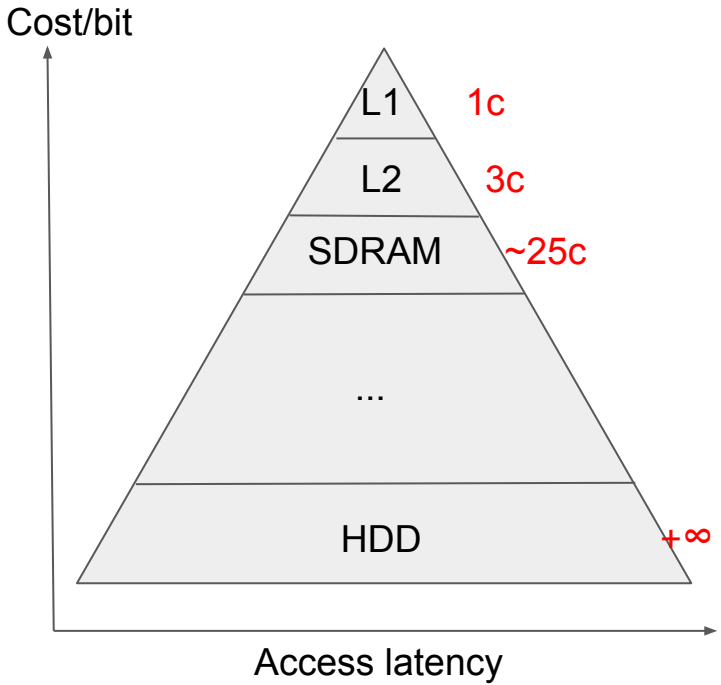
Critical Word First

- We start the refill burst with the address of the requested word first.
- Wishbone incremental bursts can wrap around the address' LSB.
- Lines of 4, 8 or 16 words are supported.



Conclusion

- In case of L1 hit:
  - 1 cycle read latency
- In case of L1 miss, assuming L2 hits:
  - 2 cycle miss penalty
  - N+1 cycles for a complete refill (N is number of words per line)
- Lots of tuning knobs:
  - Words per line
  - Number of lines
  - Associativity (1 or 2 ways)



Demo!

Thanks !

Code: https://github.com/lambdaconcept/minerva