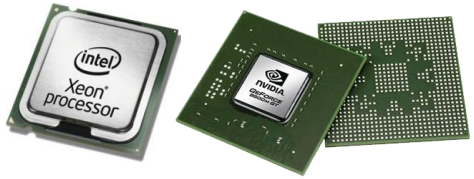
Reorder Buffer: An Energy-Efficient Multithreading Architecture for Hardware MIMD Ray Traversal

SAMSUNG Advanced Institute of Technology

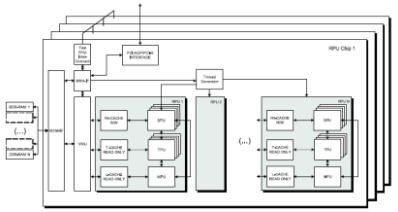
Won-Jong Lee, Youngsam Shin, Seok Joong Hwang, Seok Kang, Jeong-Joon Yoo, Soojung Ryu

Background: Mobile ray tracing H/W revisit

- Early desktop CPU/GPU (`00~09)
  - Packet tracing [Gunther 07][Overbeck 08][Benthin 09]



- HW Specialization (`02~06)
  - SarrCor [Schmitter `02], RPU, D-RPU [Woop `05, `06]
  - Not commericalized



- Modern GPUs and MICs (`10~)
  - OptiX [Steven `10], Embree [Wald `14]
  - Professional graphics

- **Mobile GPU and H/W revisit (`13~present)**
  - SGRT [Lee `13], GR6500 [McCombe `14], RayCore [Nah `14]
  - Targeted for real-time applications (Game, UX, AR/VR)

Key features of modern ray tracing H/W are..

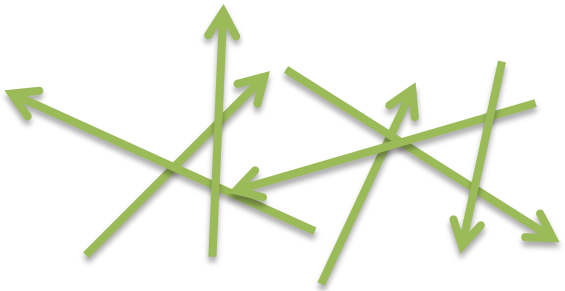
- MIMD Traversal Architecture ← Branch Divergence
  - Independent, single ray based, parallel processing
  - Better parallelism than SIMD for incoherent rays

  [Lee `13] [Kopta `10] [Nah `14] [Keely `14]

- Ray Scheduling & Multithreading ← Memory Divergence
  - Schedule the rays to increase memory locality

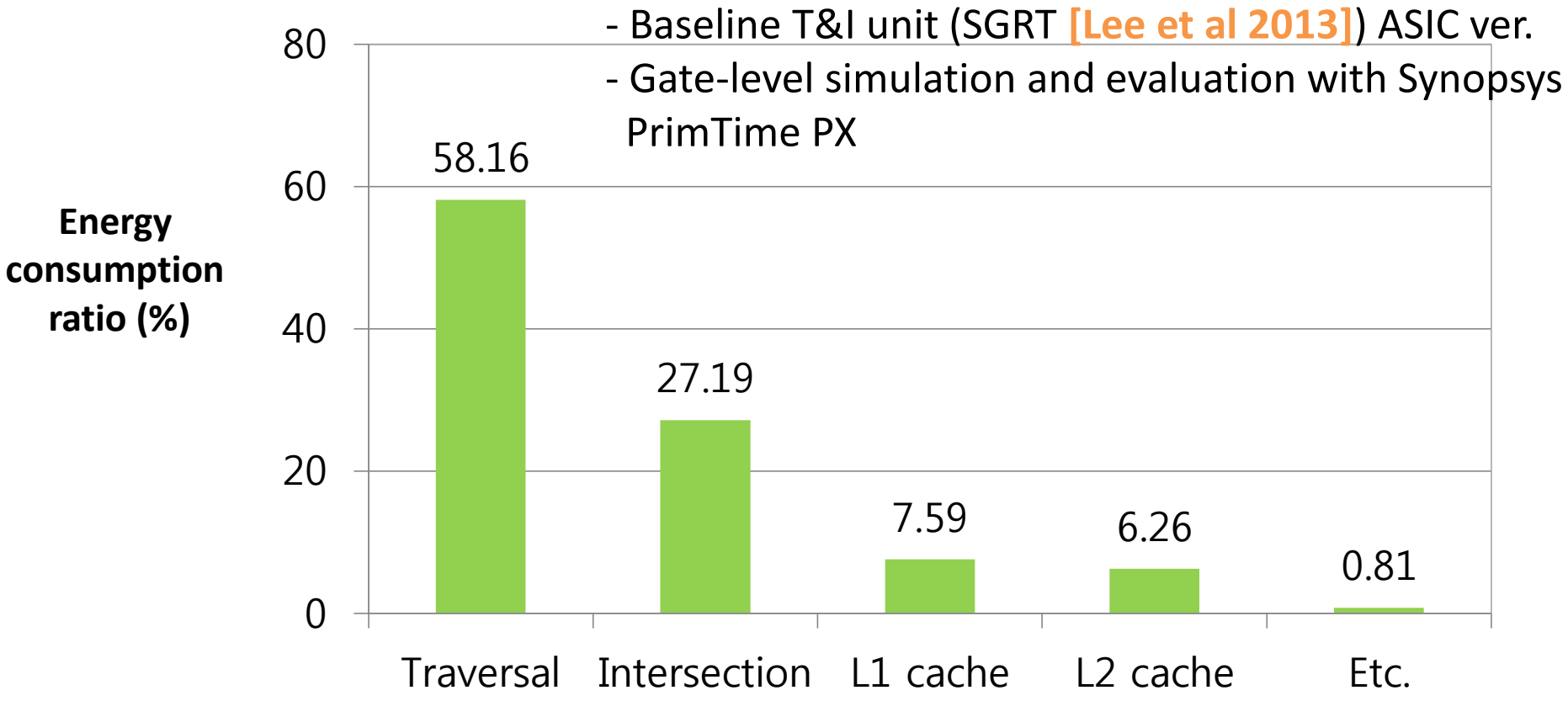
    [Aila `10] [Moon `11] [McCombe `14] [Kopta `14] [Keely `14]
  - Hide memory latencies caused by miss penalties

    [Nah `11] [Lee `13] [Kwon `13][Nah `14]



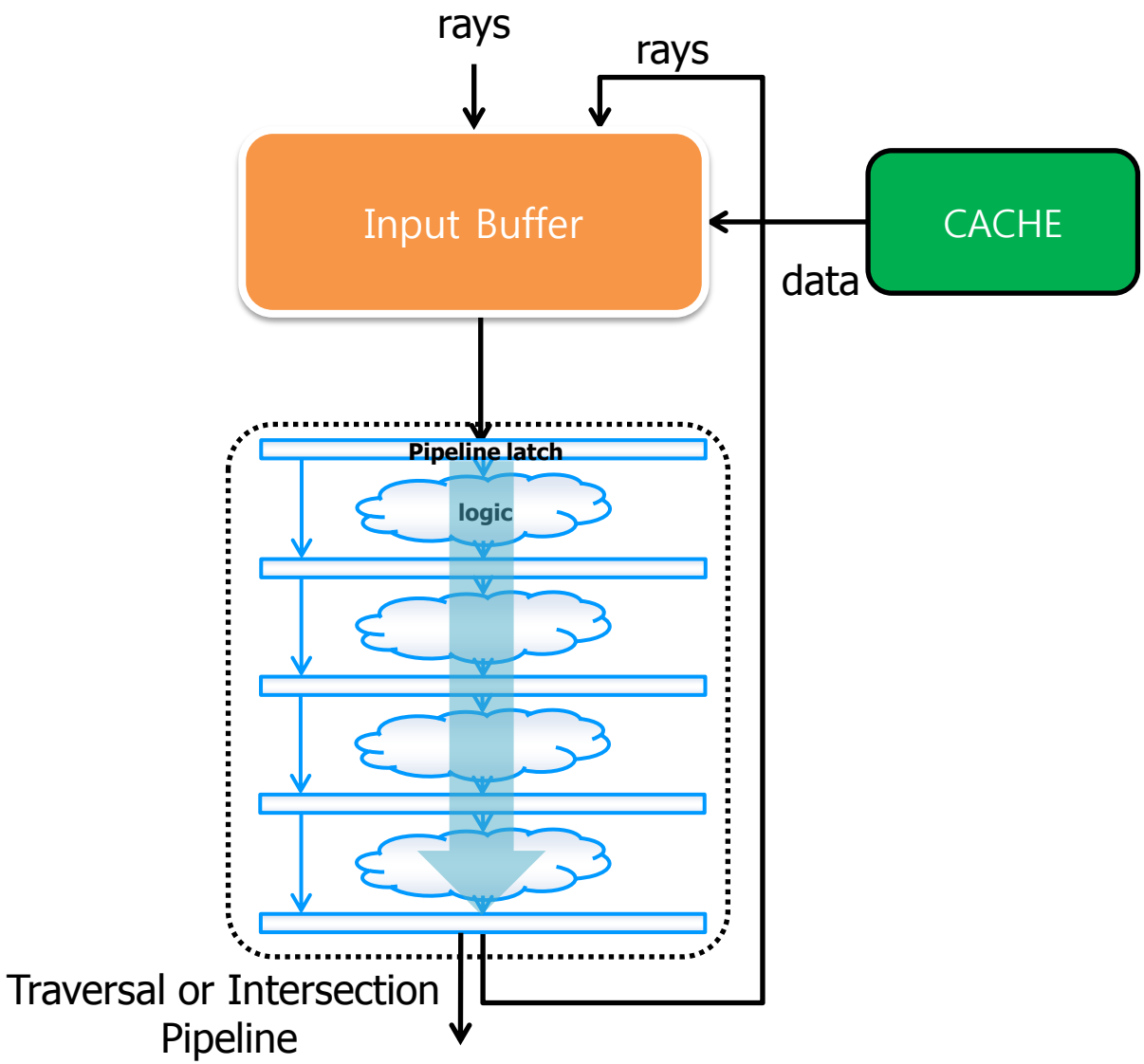
Background: Low Power Consumption

- MIMD traversal pipeline consumes energy of 58.16% in the ray tracing logic (except DRAM).

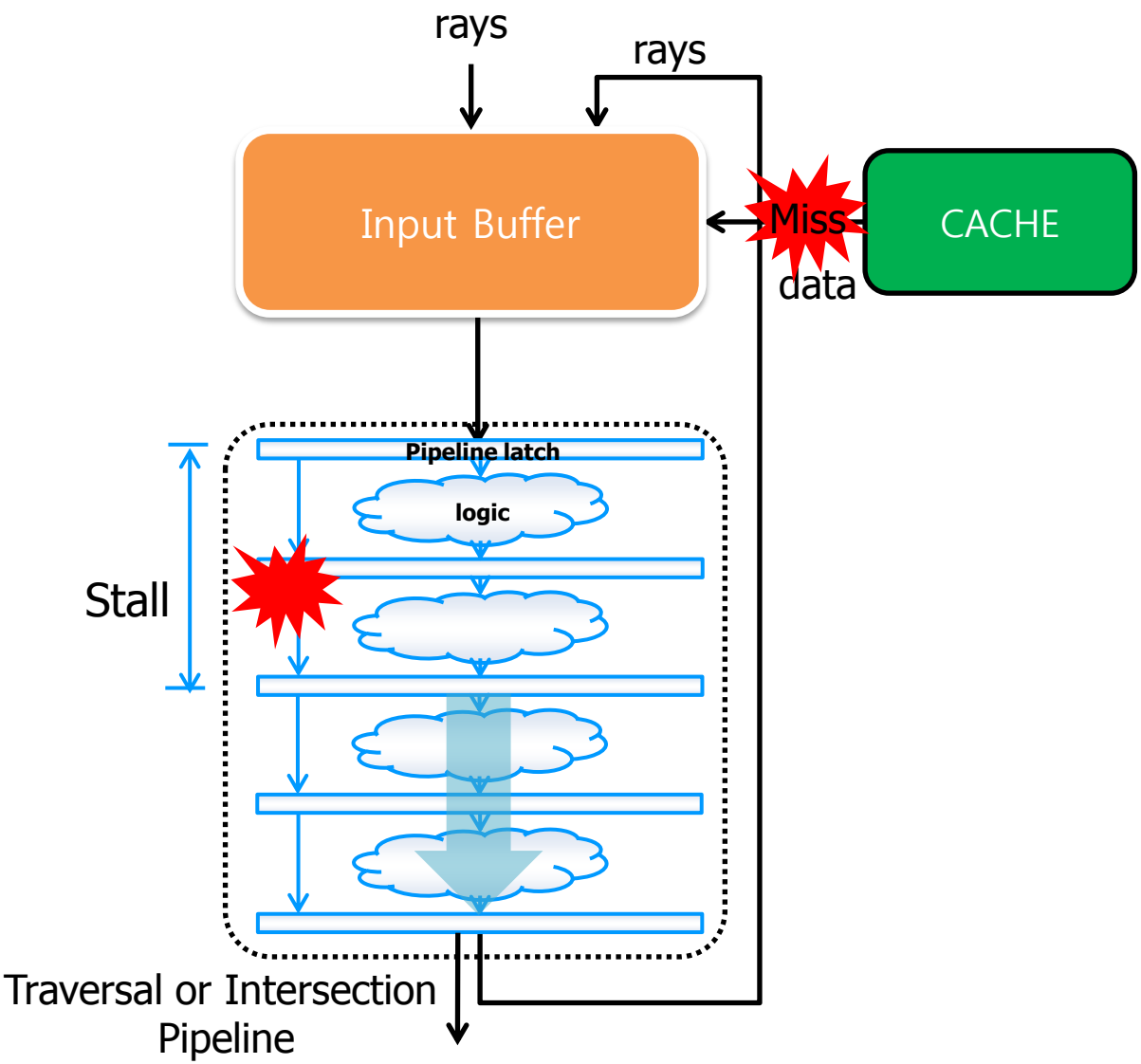


Traversal (or Intersection) pipeline

- Consists of input buffer, cache, pipeline and feed back



Classical problem:
Cache miss causes pipeline stall



Previous hardware multithreading (latency hiding) for ray tracing

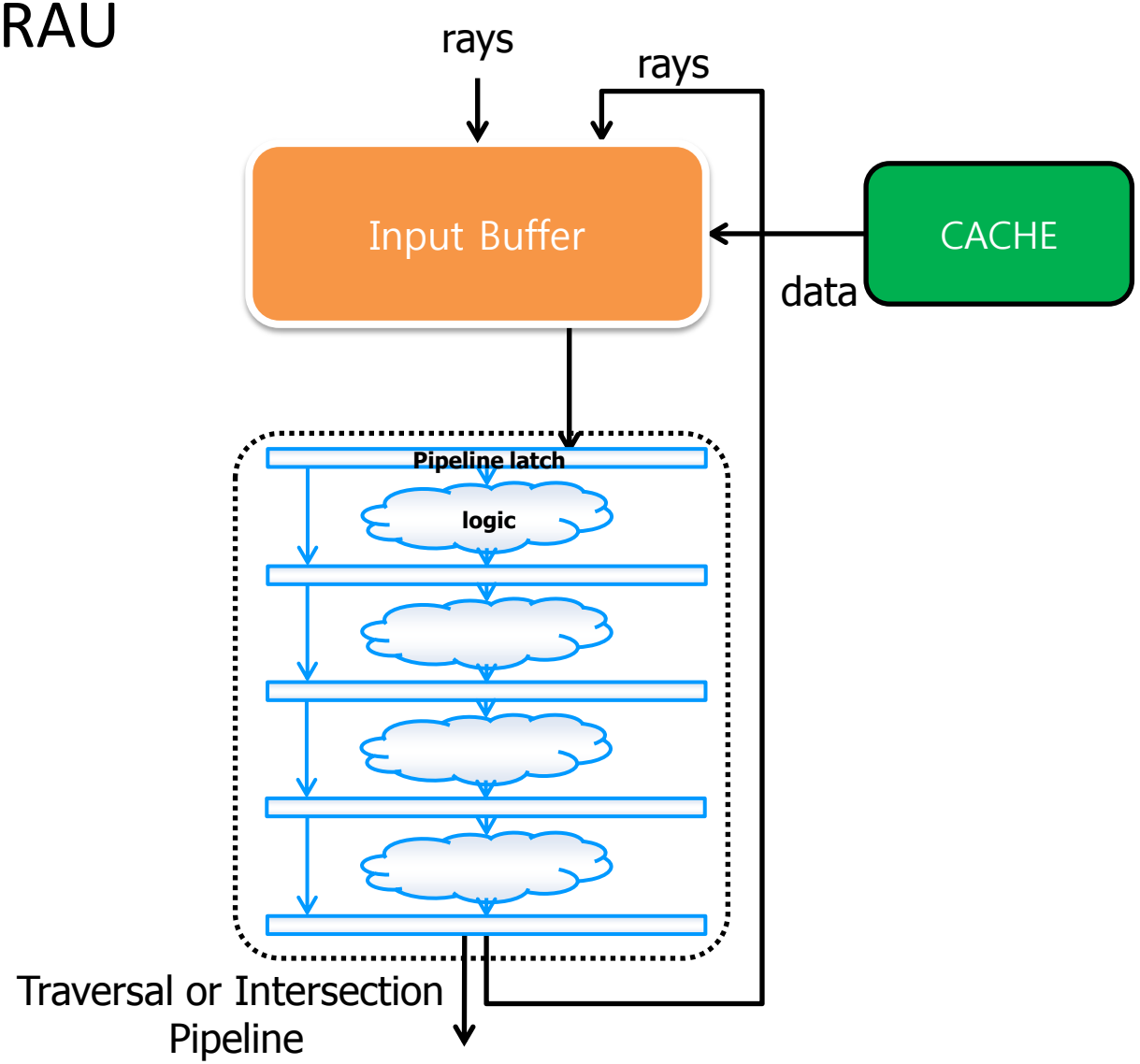
- Ray Accumulation Unit (RAU) [Nah `11 – T&I Engne] [Lee `13 - SGRT]
  - Prevents pipeline stall by storing rays that induce a cache miss to the buffer and processing other threads
- Retry [Kwon `13][Nah `14 - RayCore]
  - Invalidates cache-missed rays and feeds them to the hardware without any pipeline stalls being incurred
  - A.K.A “Looping for next chance”

PROBLEM

The RAU and the Retry deal with rays inefficiently

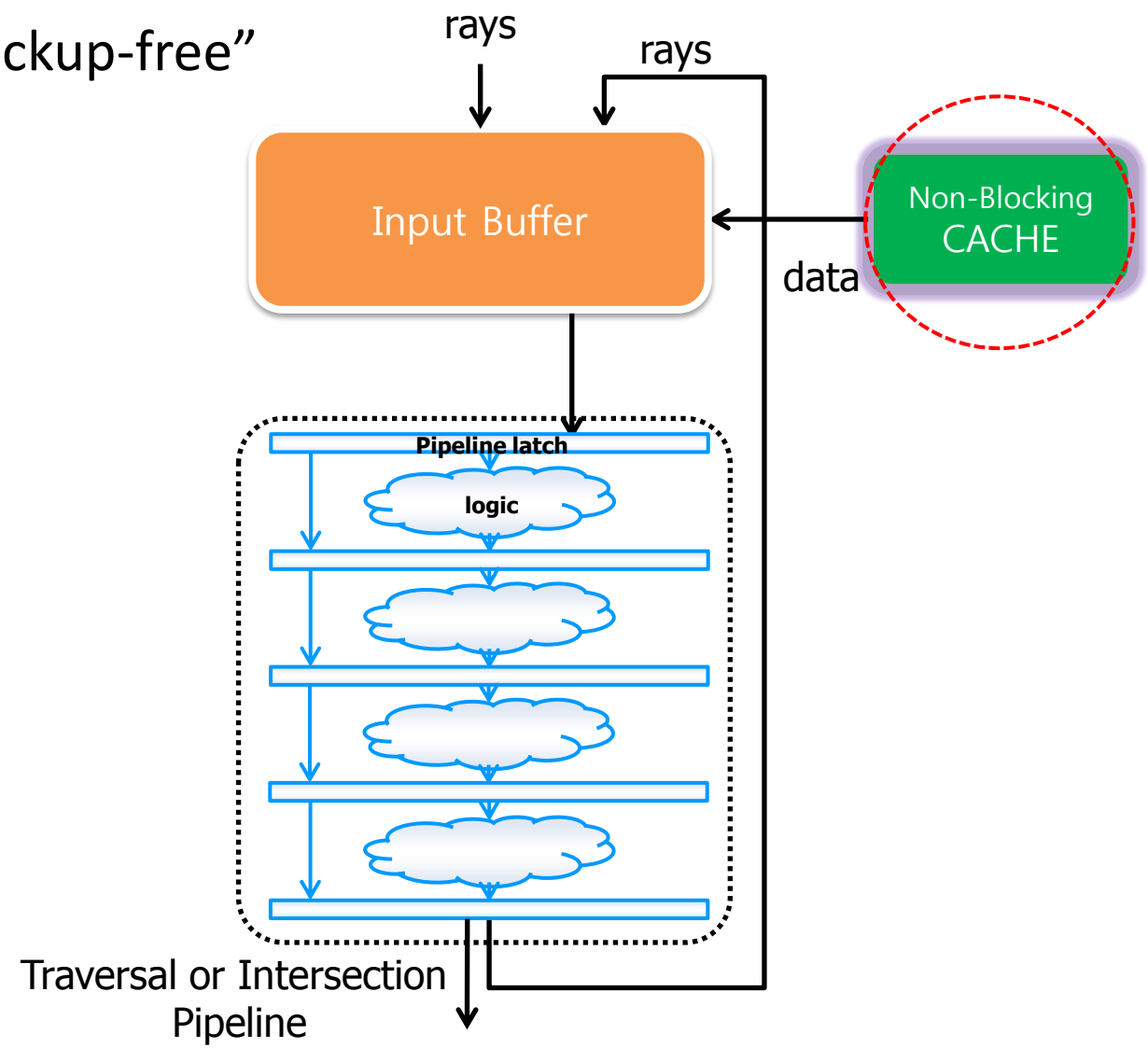
Ray Accumulation Unit

- Let's make RAU



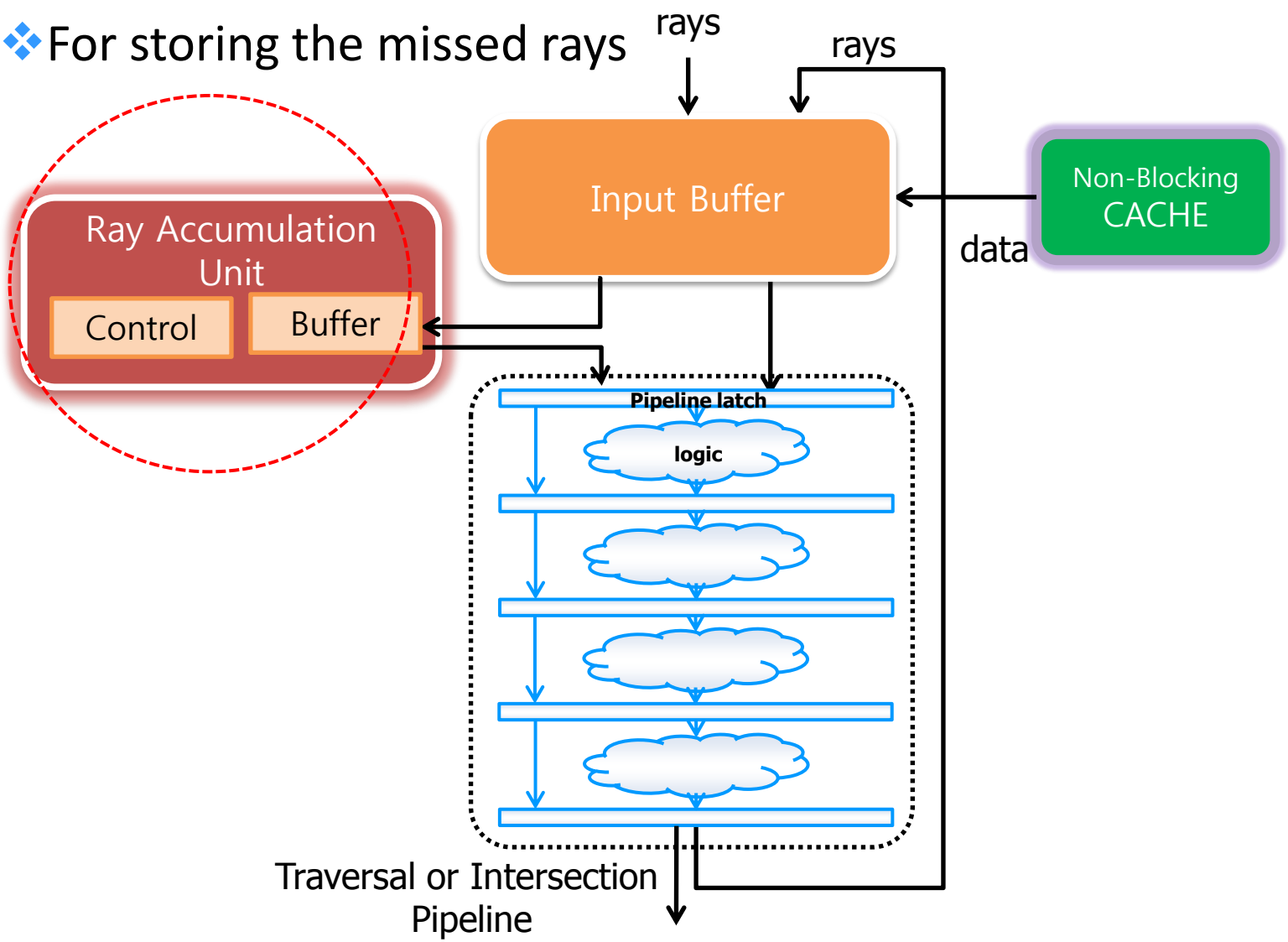
Ray Accumulation Unit

- We need a non-blocking cache to successive service
  - A.K.A “Lockup-free”



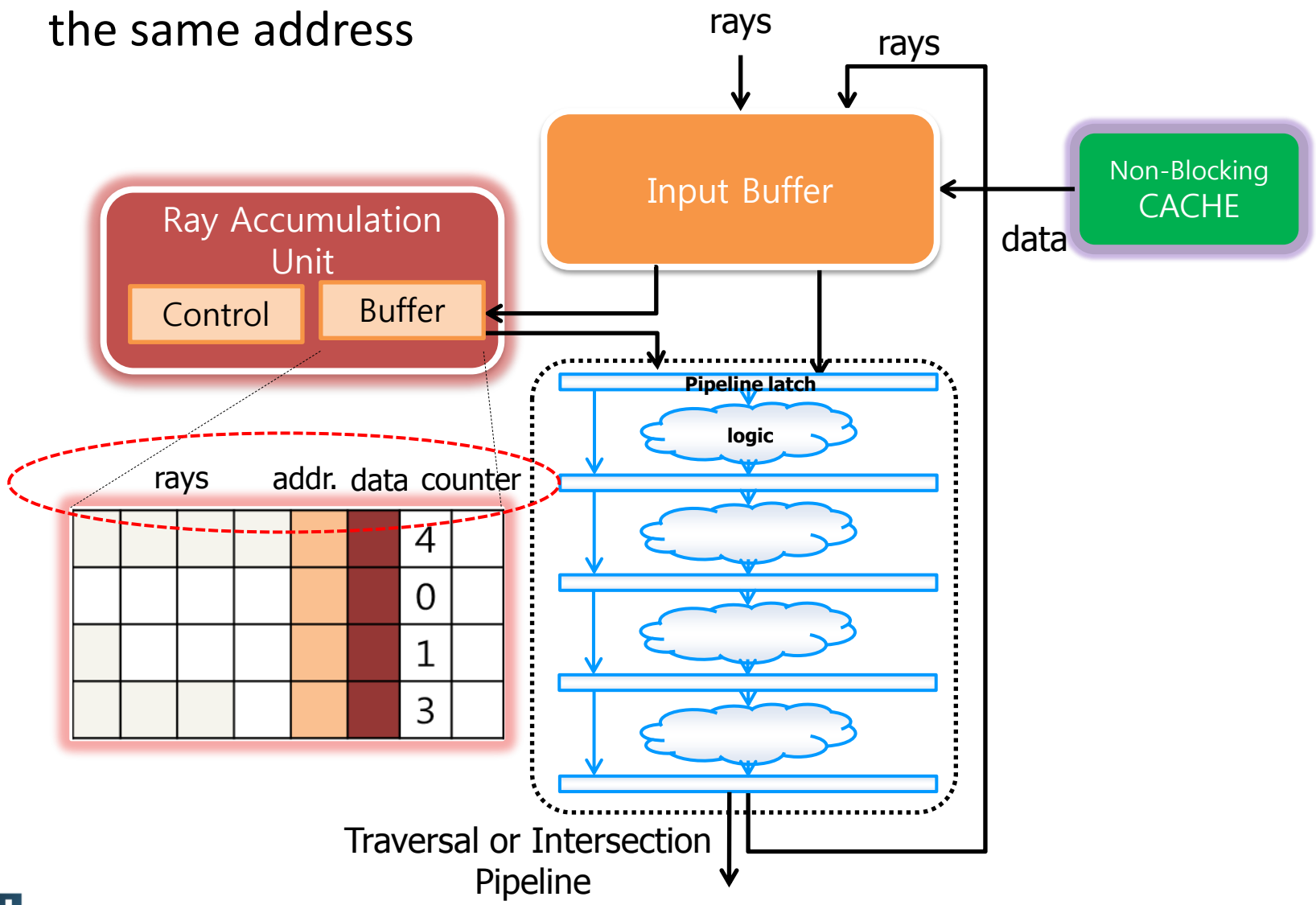
Ray Accumulation Unit

- Accumulation buffer & control logic
  - For storing the missed rays



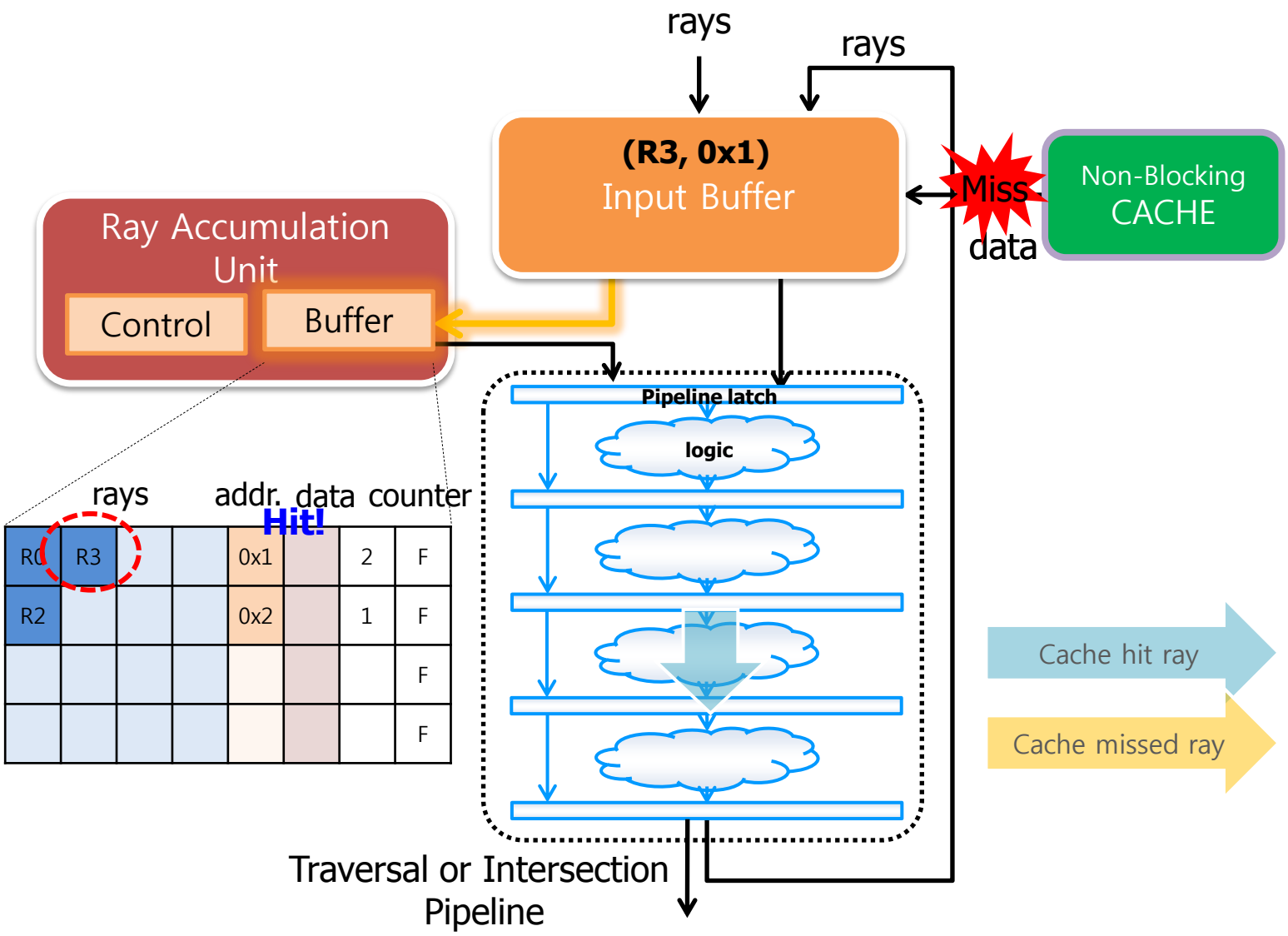
Ray Accumulation Unit

- Buffer is configured in two dimensionally to group rays that have the same address



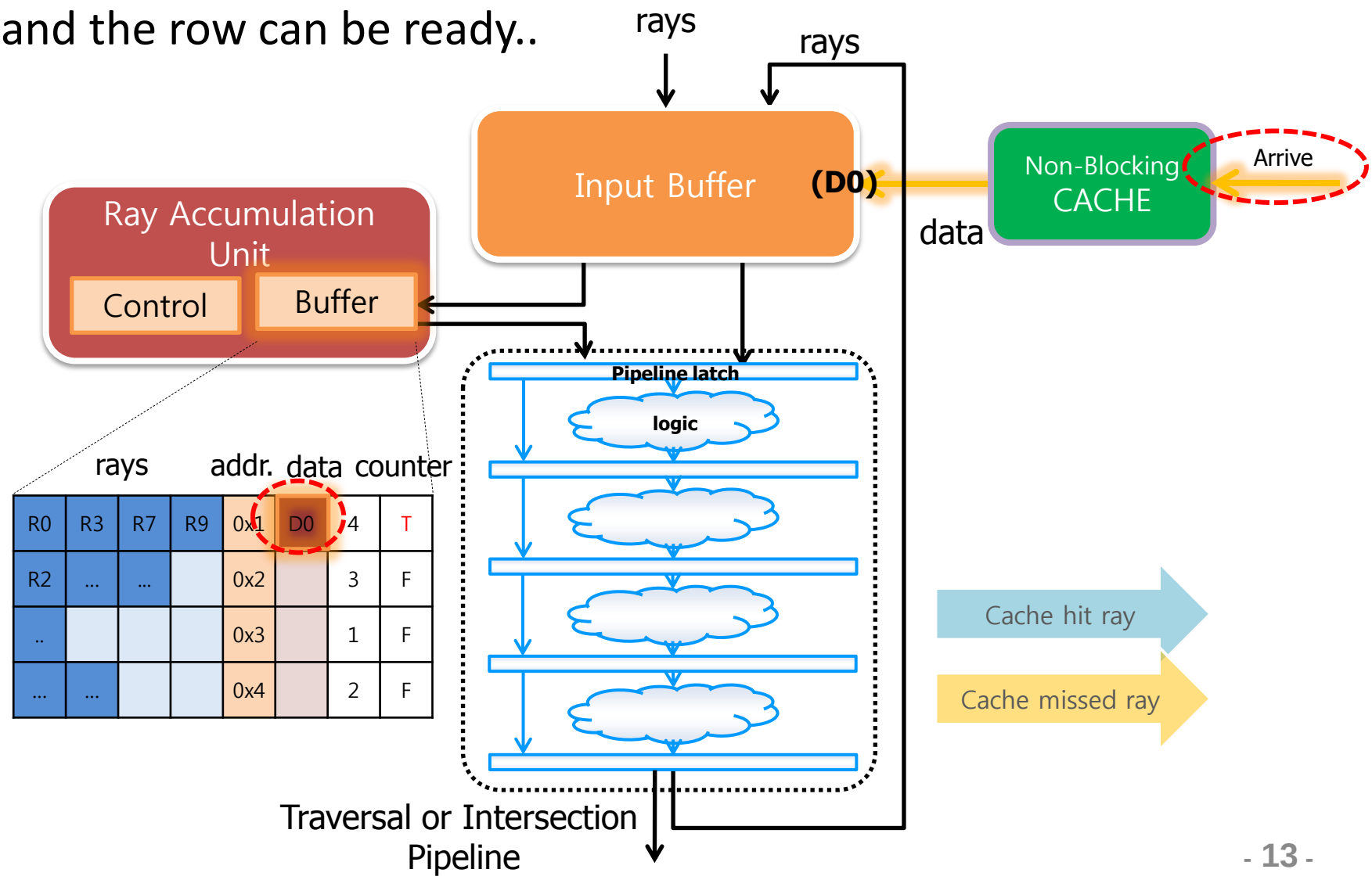
When the cache miss is occurred..

- stored in same row (0x1 (R0) == 0x1 (R3))



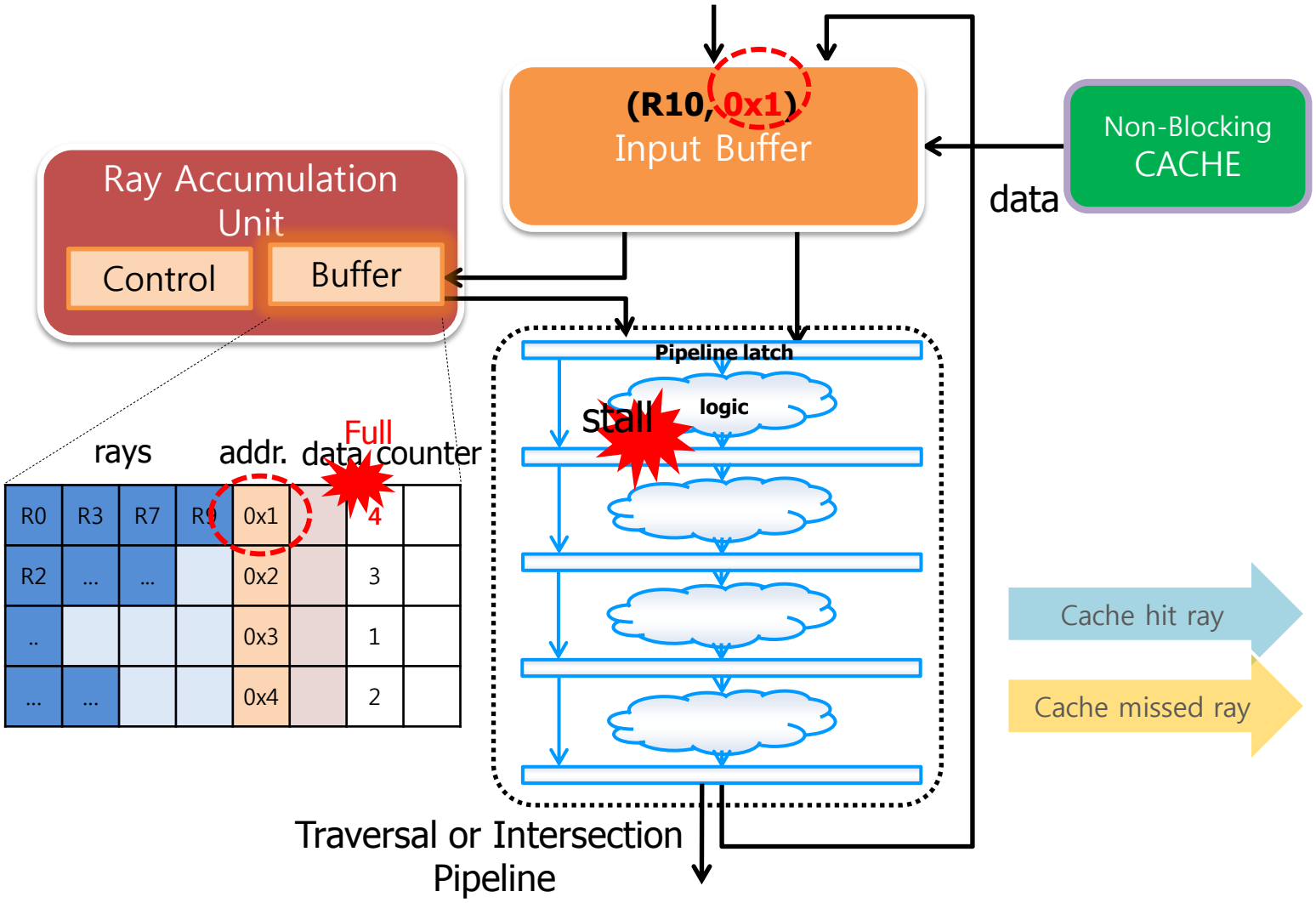
When the cache miss is complete….

- the cache data is copied to the corresponding row of RAU and the row can be ready..



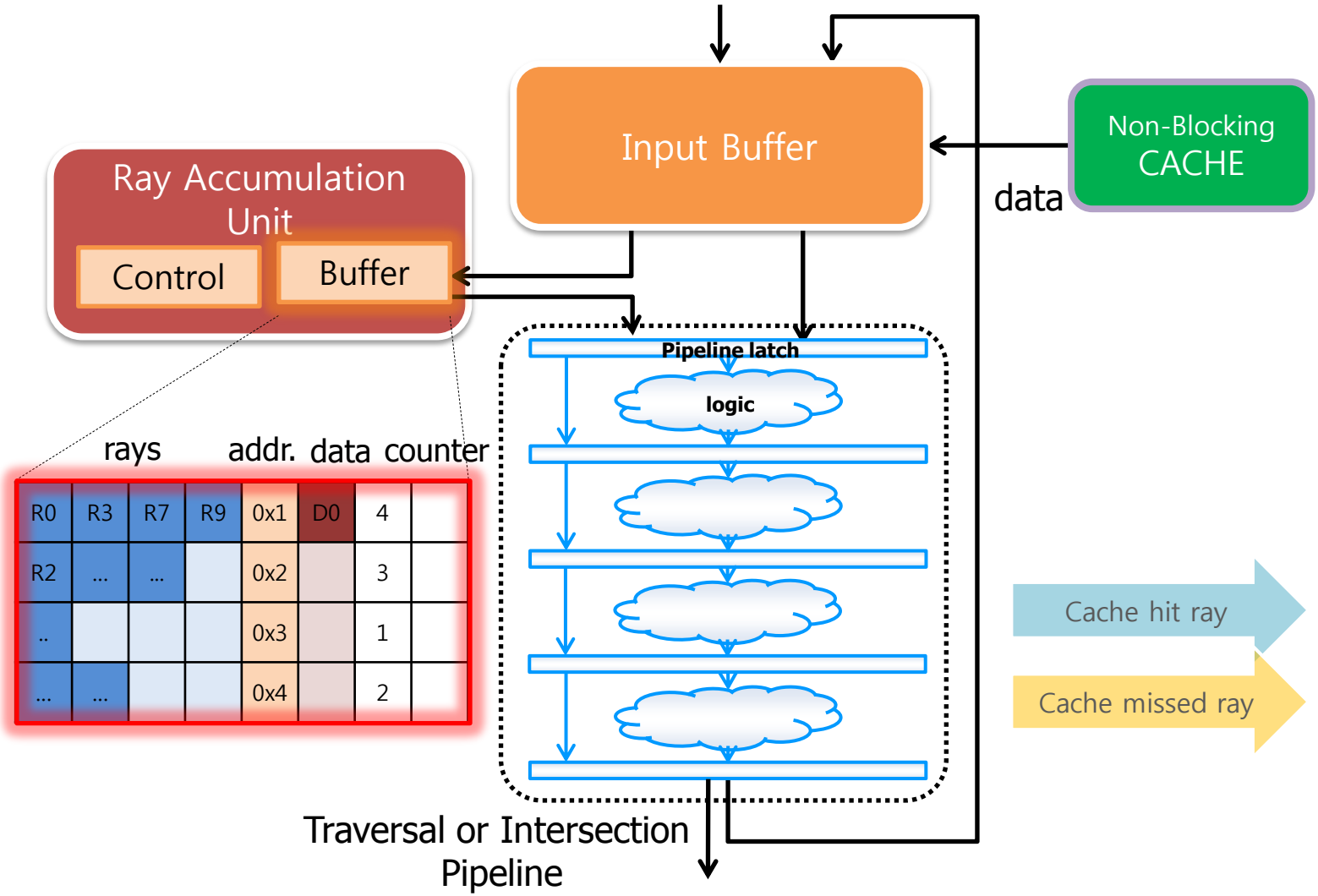
Problem #1: What if a row is full and the ray with same address is arrived at input buffer?

- Capacity miss even if there's a room



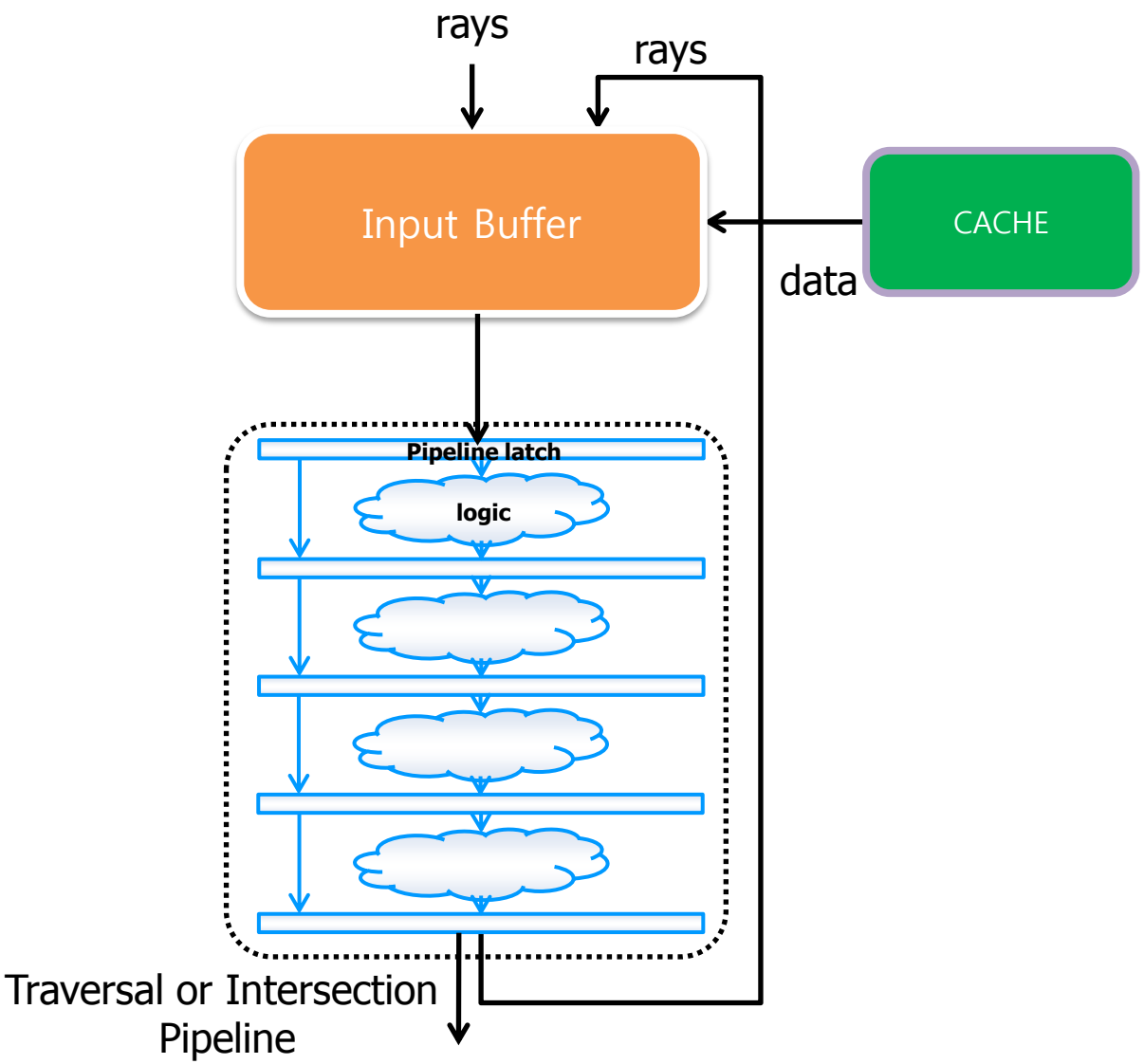
Problem#2: RAU needs a relatively big SRAM buffer (32-64KB)

- Ray payloads (org, dir,..), data, address.. etc



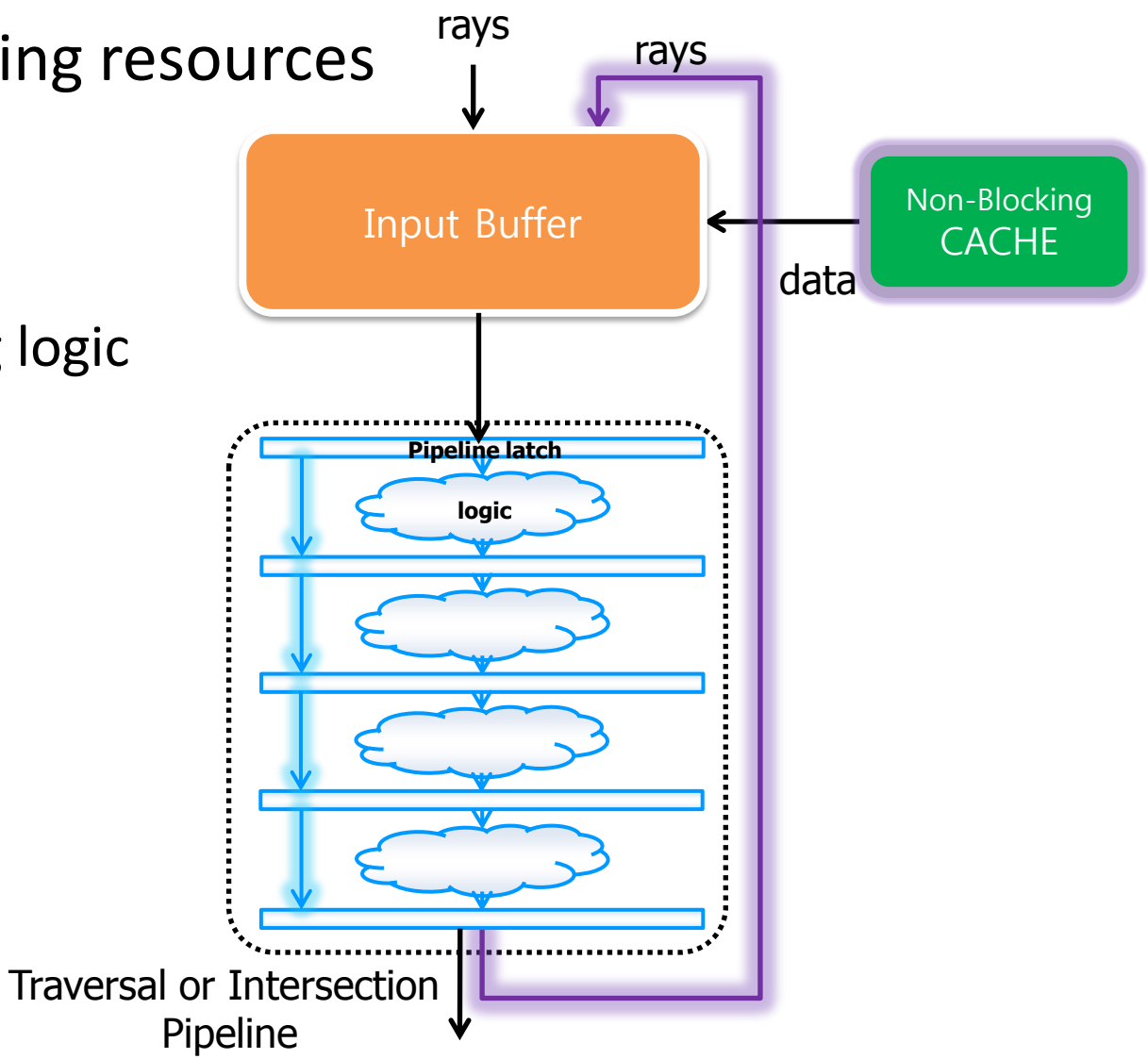
Retry method

- Let's talk about alternative! – Retry



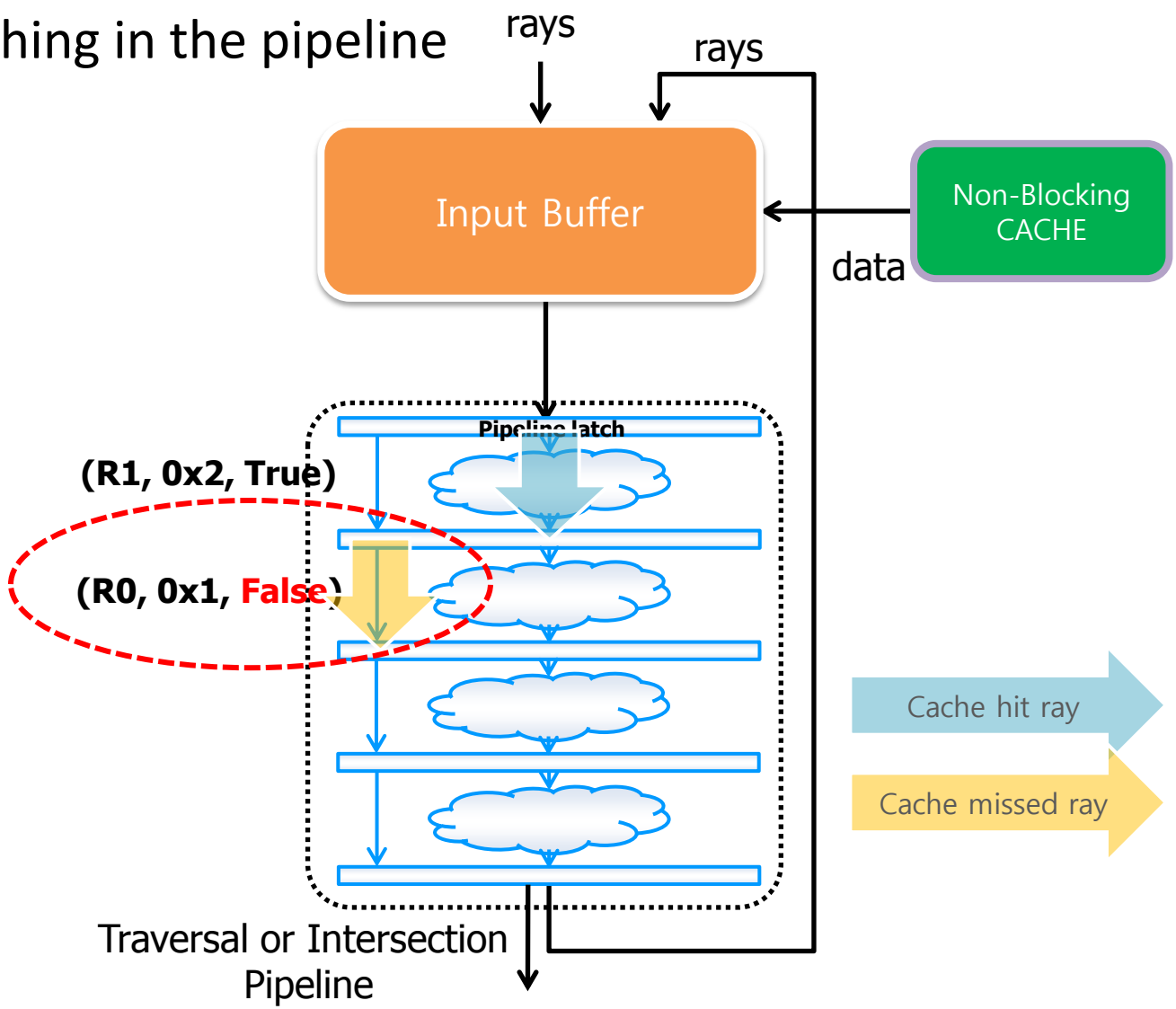
Retry method

- No needs for additional buffer
- Utilize existing resources
  - NB cache
  - Feedback
  - Bypassing logic



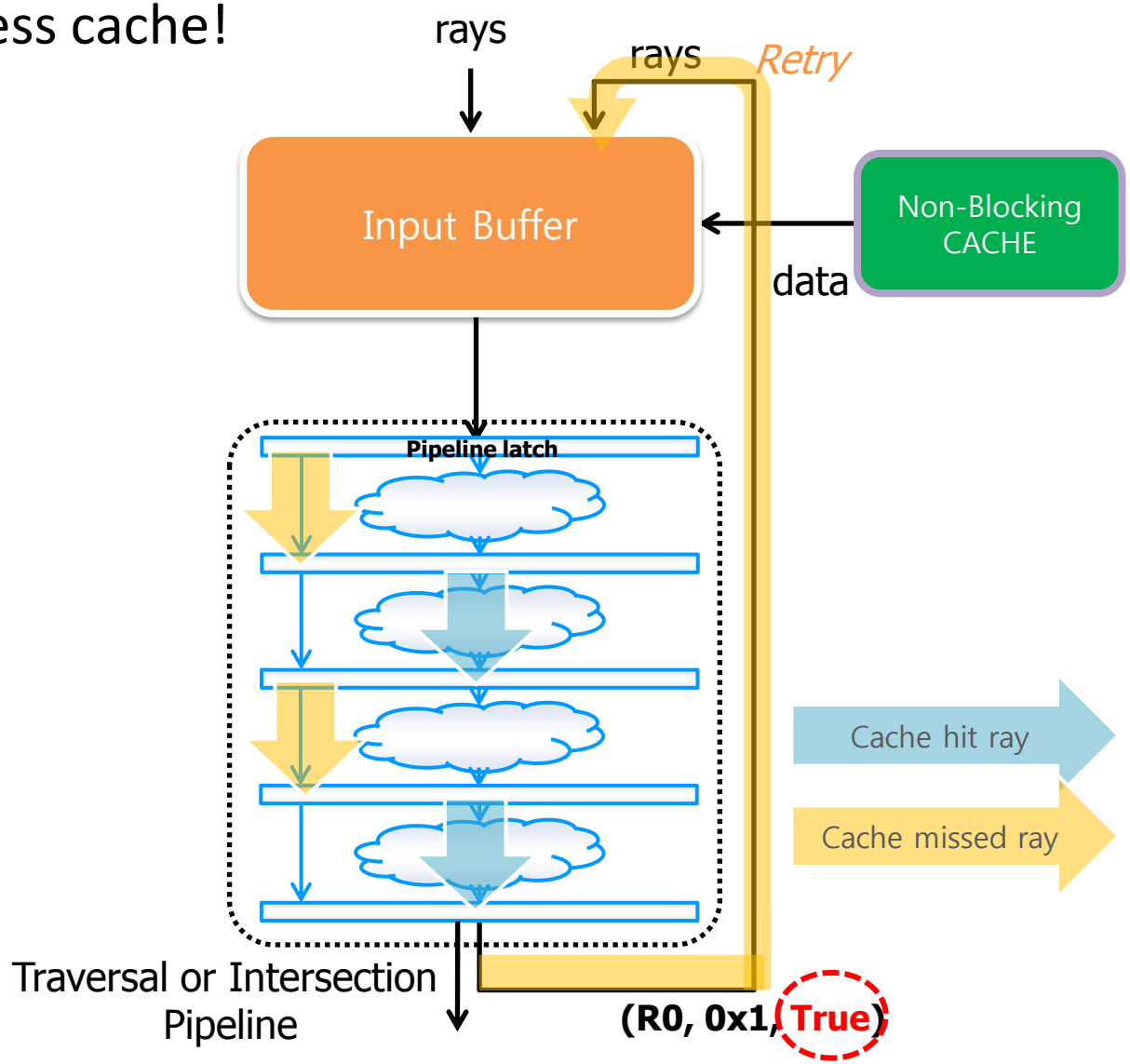
When the cache miss is occurred…

- The ray is just invalided and fed to the pipeline without stall.
- Does nothing in the pipeline



When the cache-missed-ray arrives to the end of the pipeline…

- Retry to access cache!



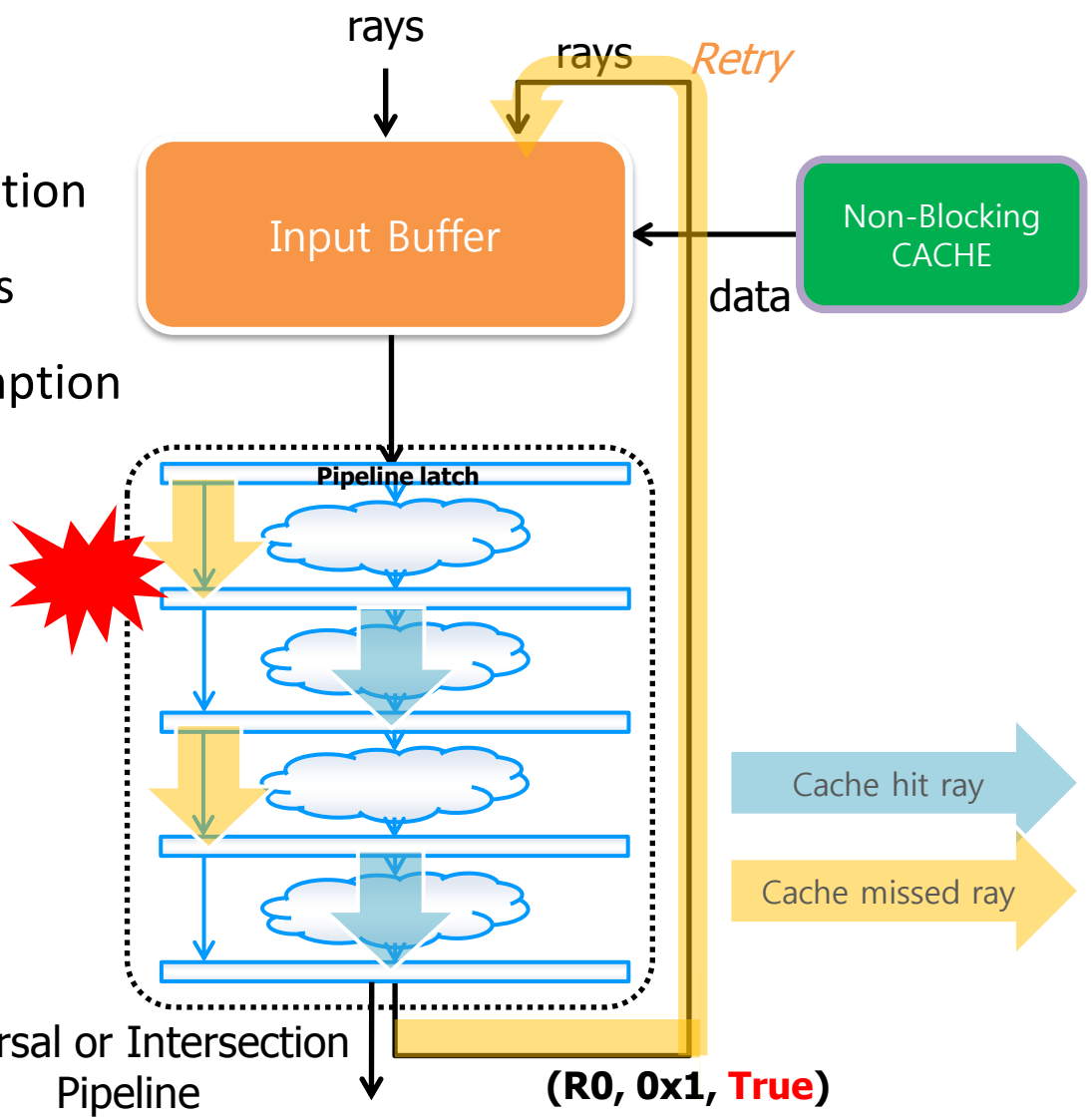
Problem: Bypassing causes higher energy consumption by R2R transfer and switching

Longer DRAM latency

→ more bypassing iteration

→ more cache re-access

→ more energy consumption



Goal & Approach

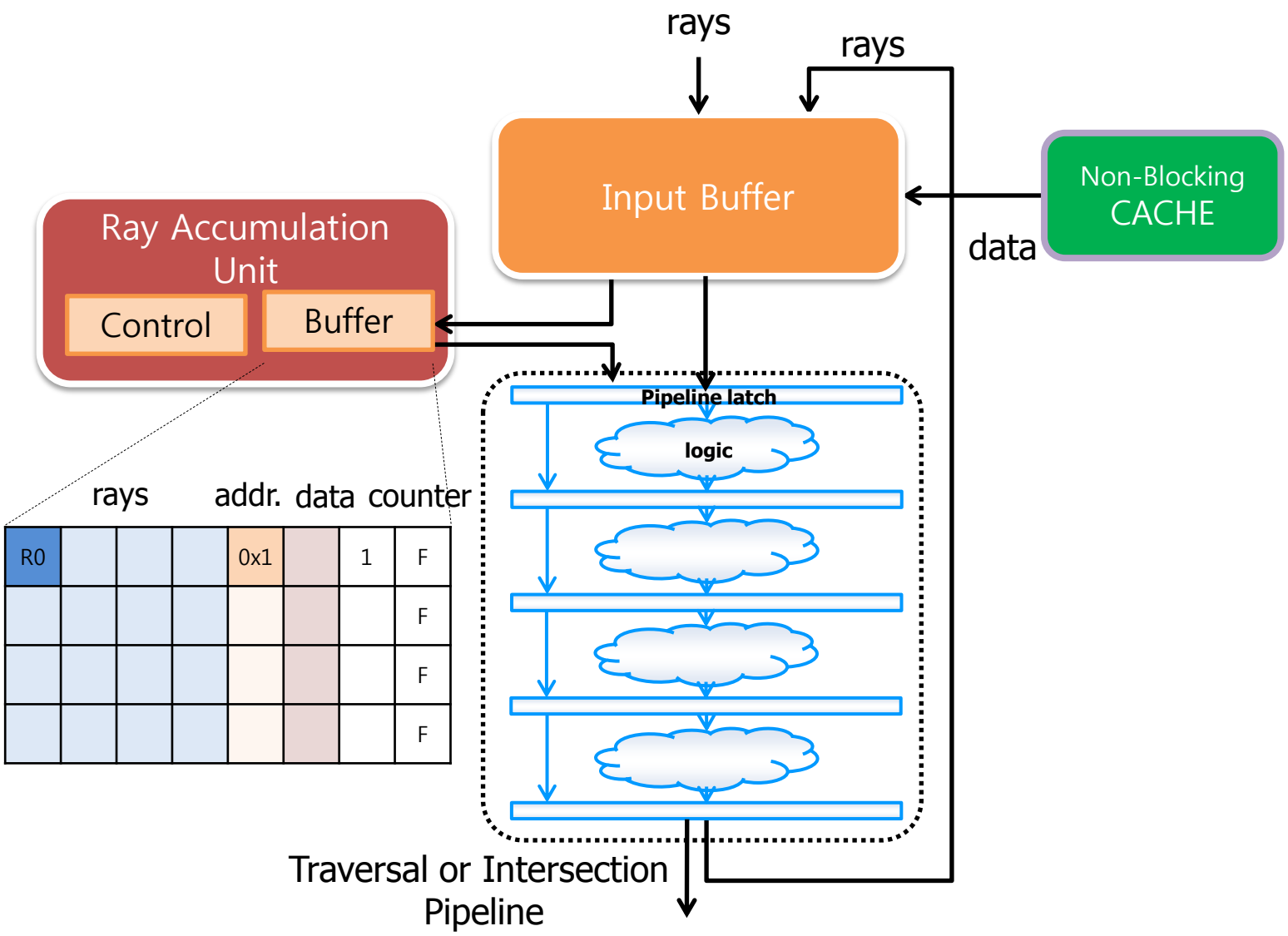
- Goal: Efficient hardware multithreading for MIMD traversal with minimal cost and energy consumption

- Approach:
  - Eliminate dedicated buffers and avoid bypassing
  - Utilize existing buffer with minimal modification

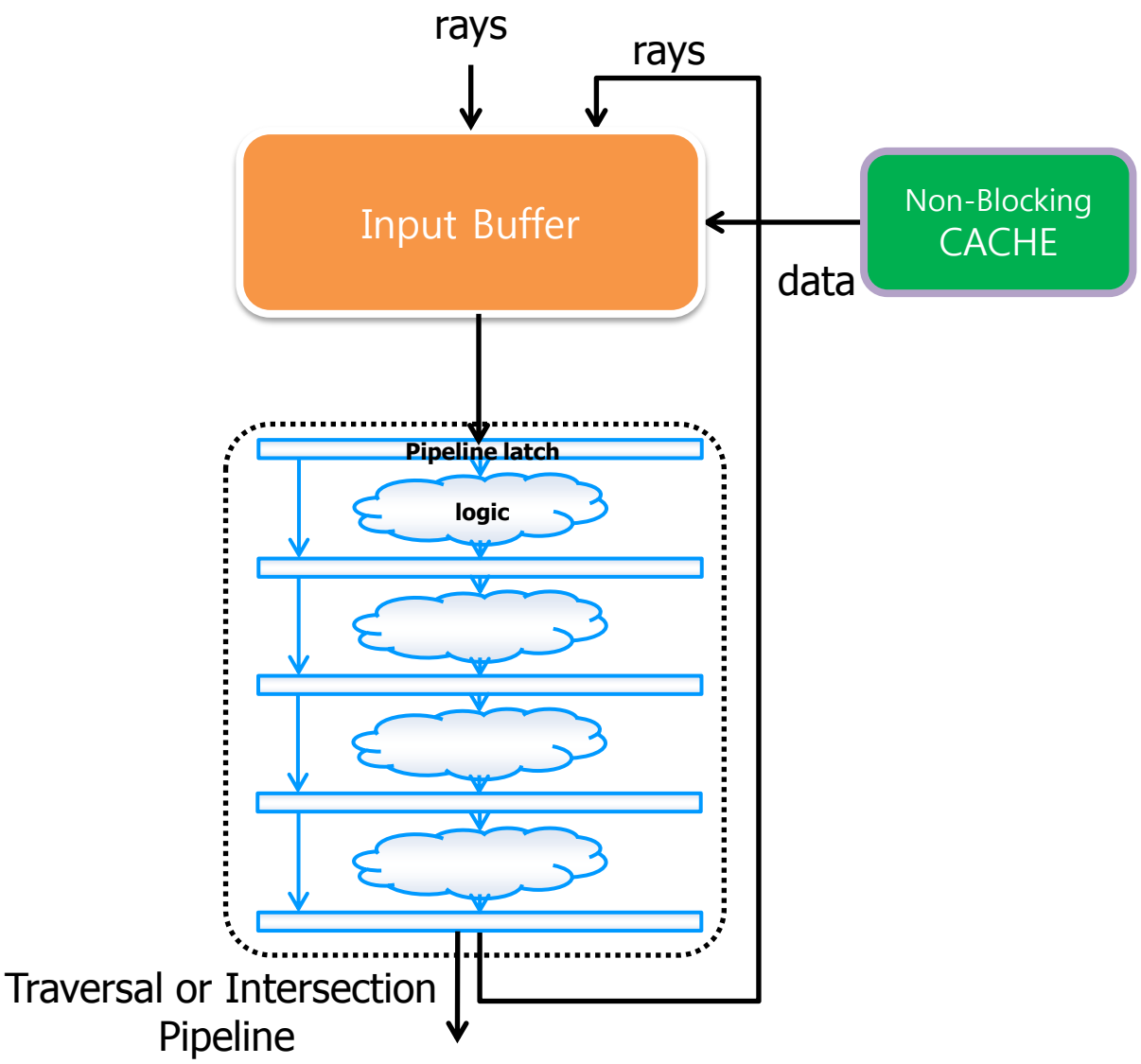
REORDER BUFFER

Now, we propose a new method to resolve previous problems

Eliminate dedicated buffers



Avoid bypassing



Approach

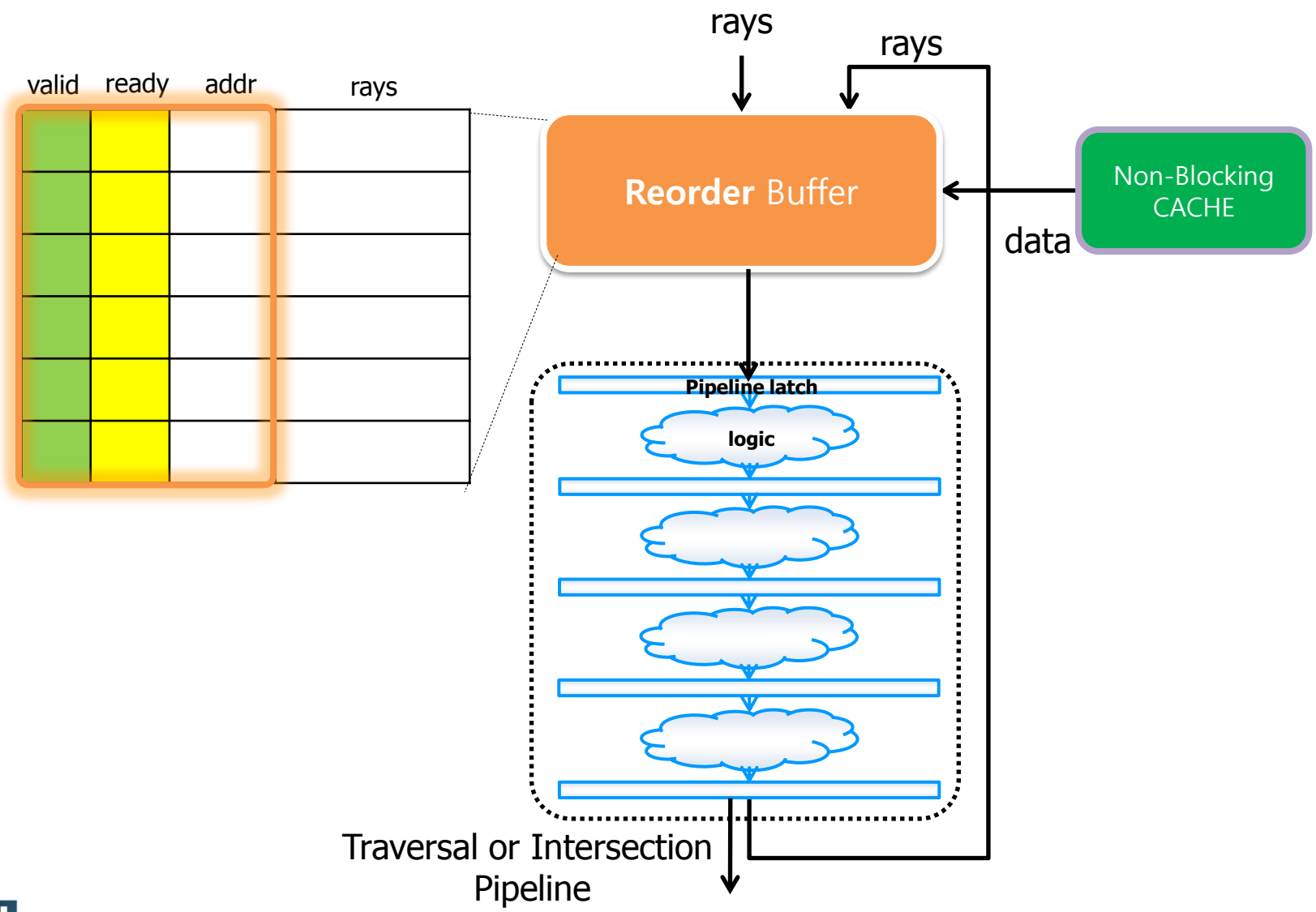
- How can we retaining the cache missed ray without any further resources, bypassing and pipeline stall

→ Input Buffer with small extension

: Valid (1bit) + Ready (1bit) + Address (26bits)

Reorder Buffer - Configuration

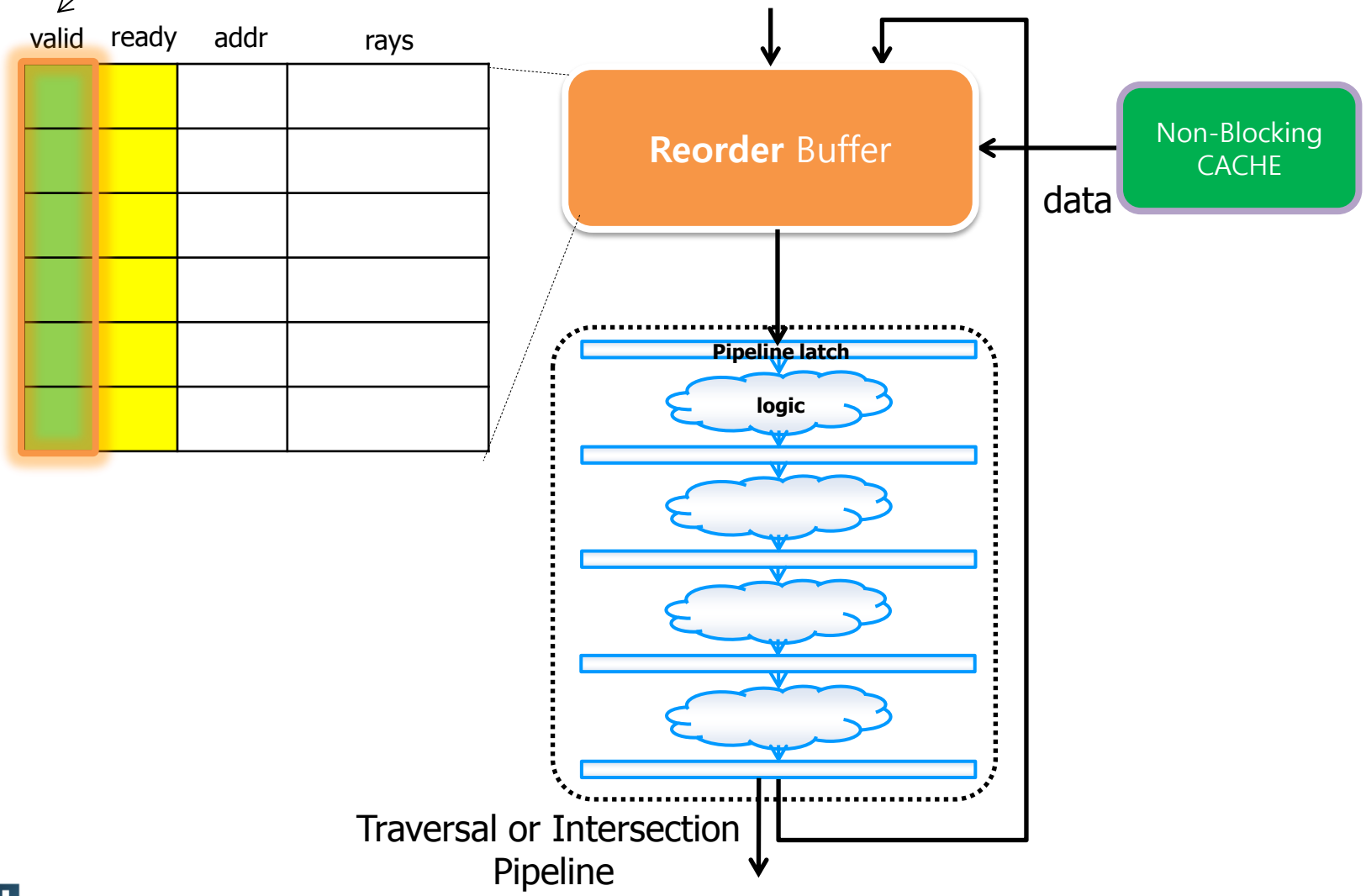
- Utilize input buffer for latency hiding



Reorder Buffer - Configuration

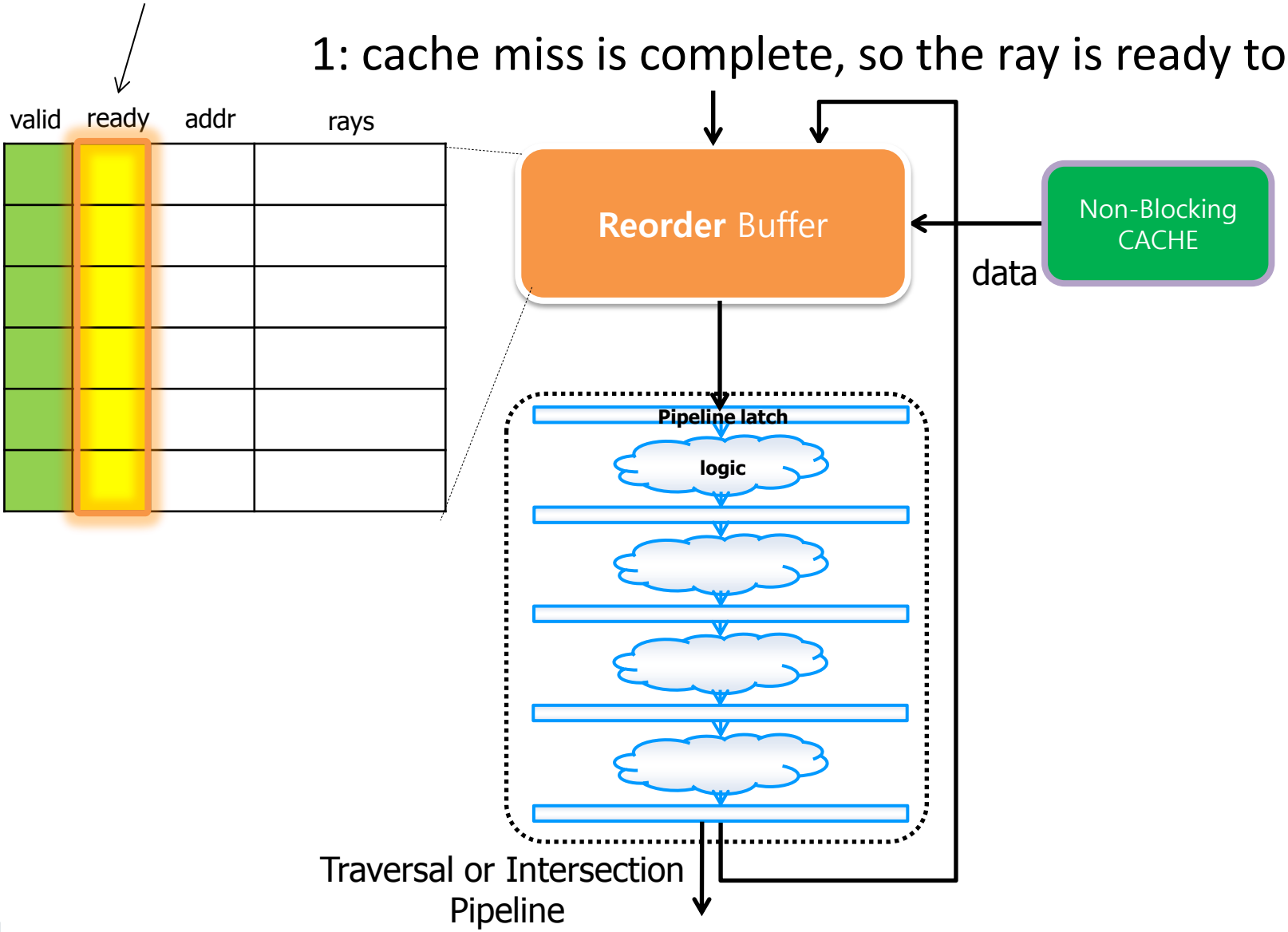
- Type of ray, 1: newly arrived & not yet accessed the cache

  0: accessed the cache and missed



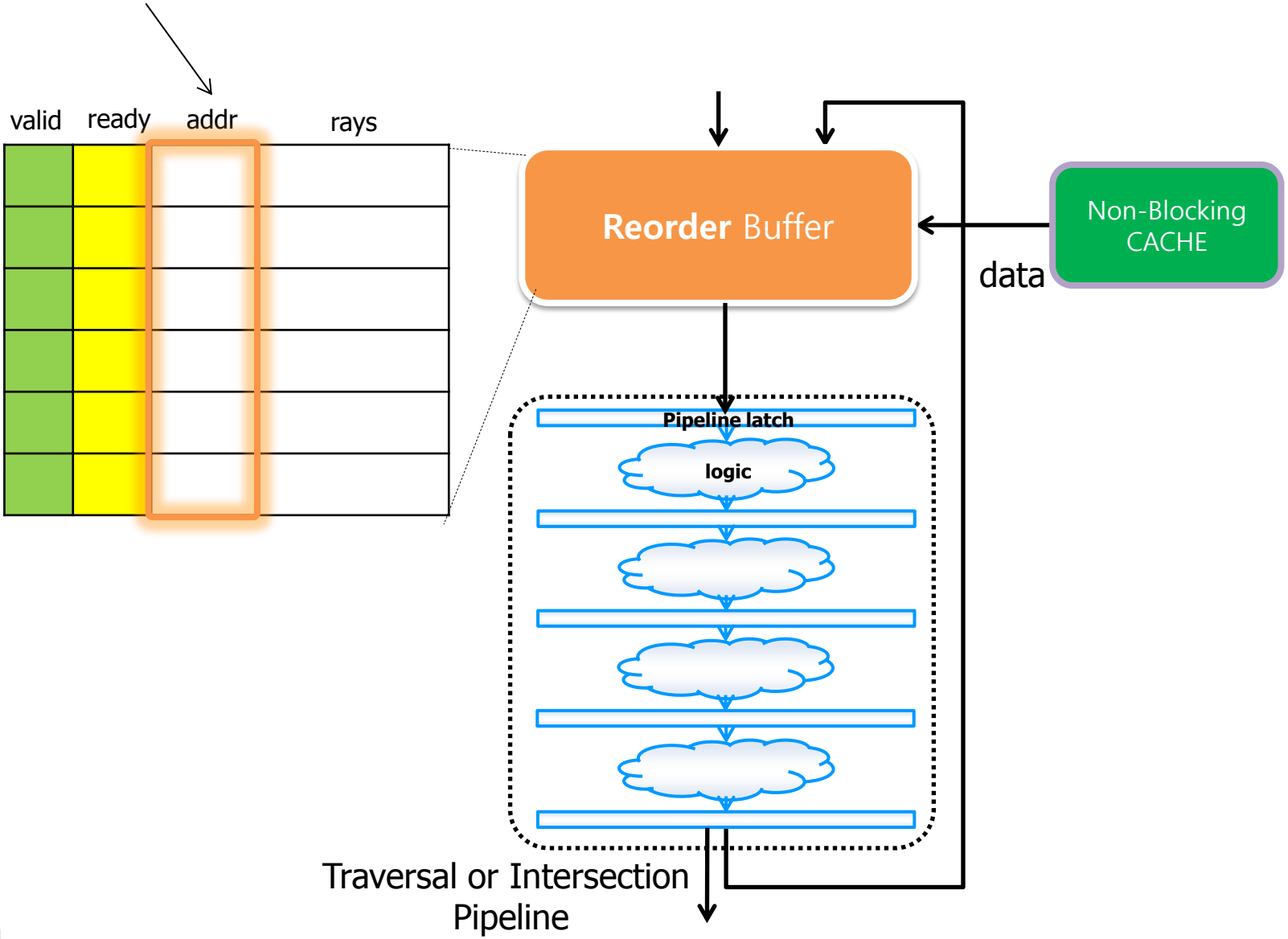
Reorder Buffer - Configuration

- Ray is ready? 0: cache missed and waiting for miss complete



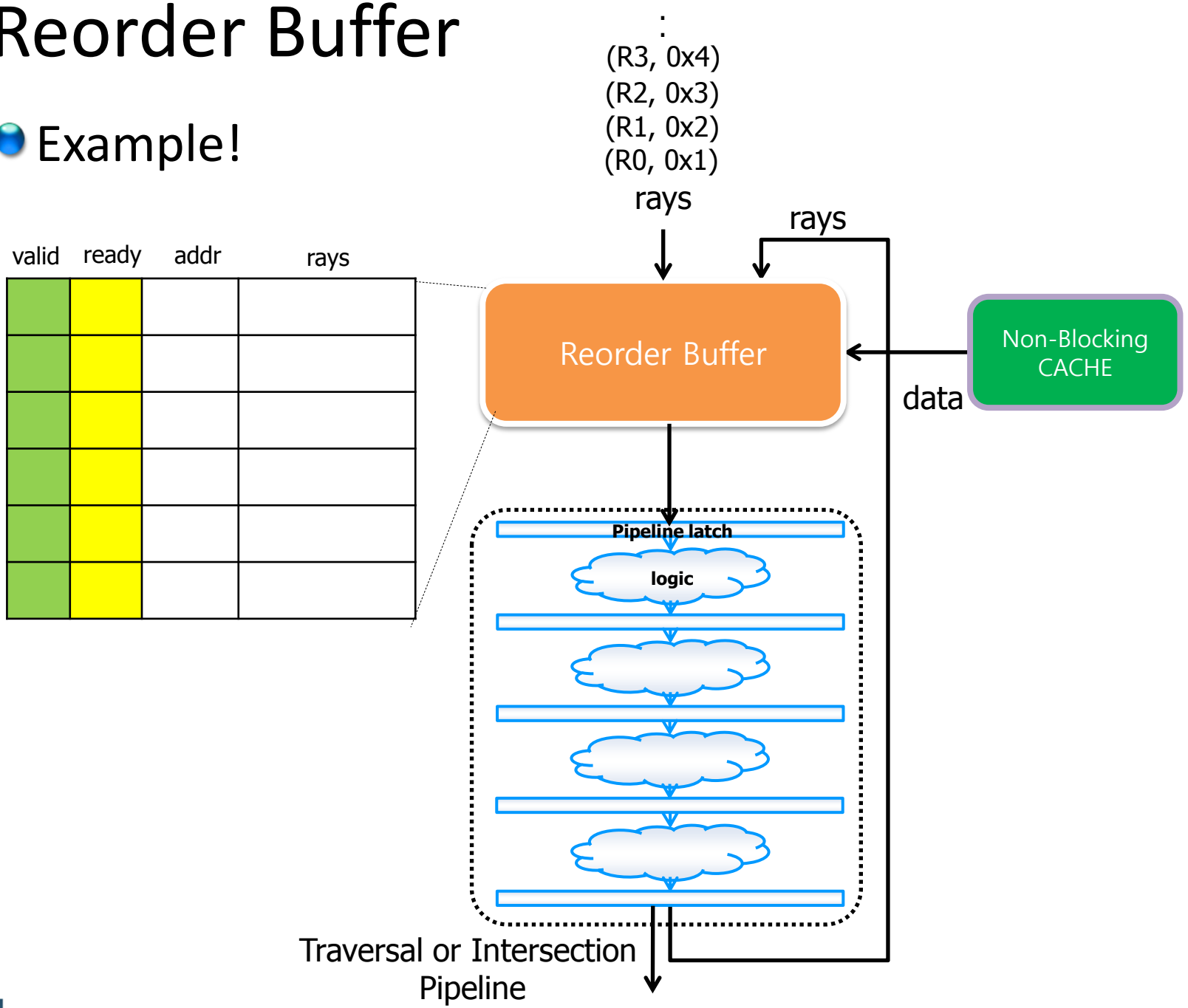
Reorder Buffer - Configuration

- Cache address references data, and used for searching



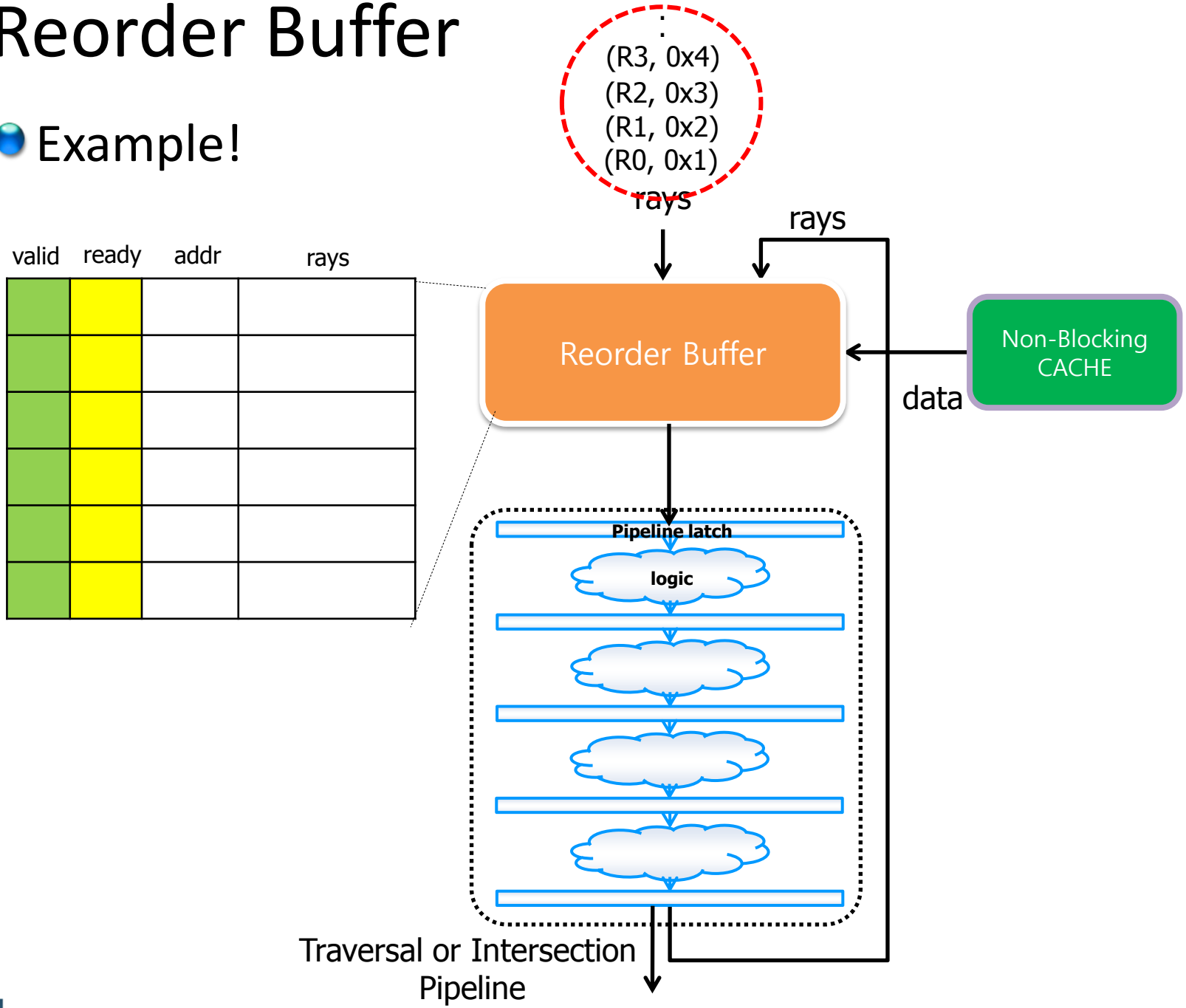
Reorder Buffer

- Example!



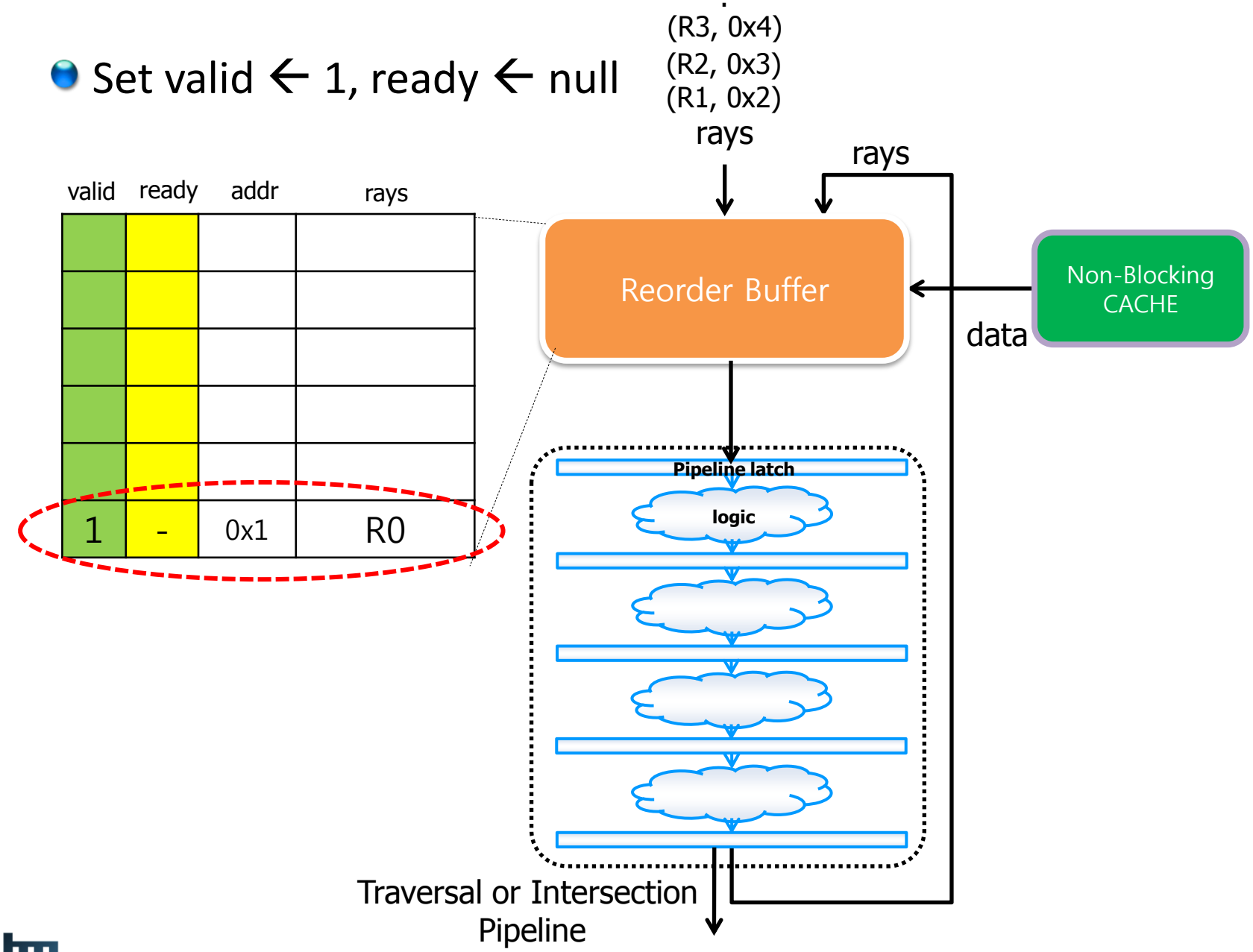
Reorder Buffer

- Example!



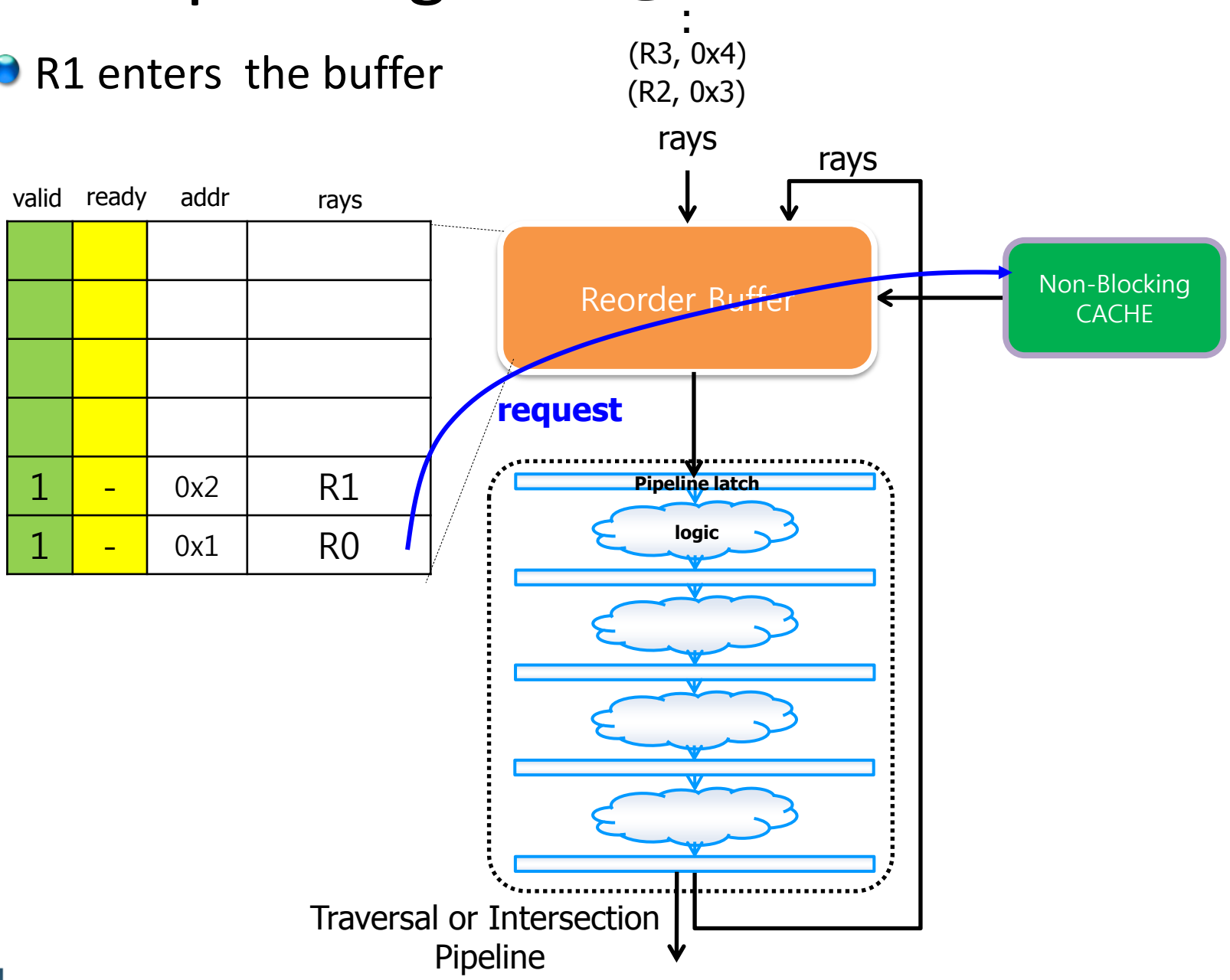
R0 enters the buffer..

- Set valid ← 1, ready ← null



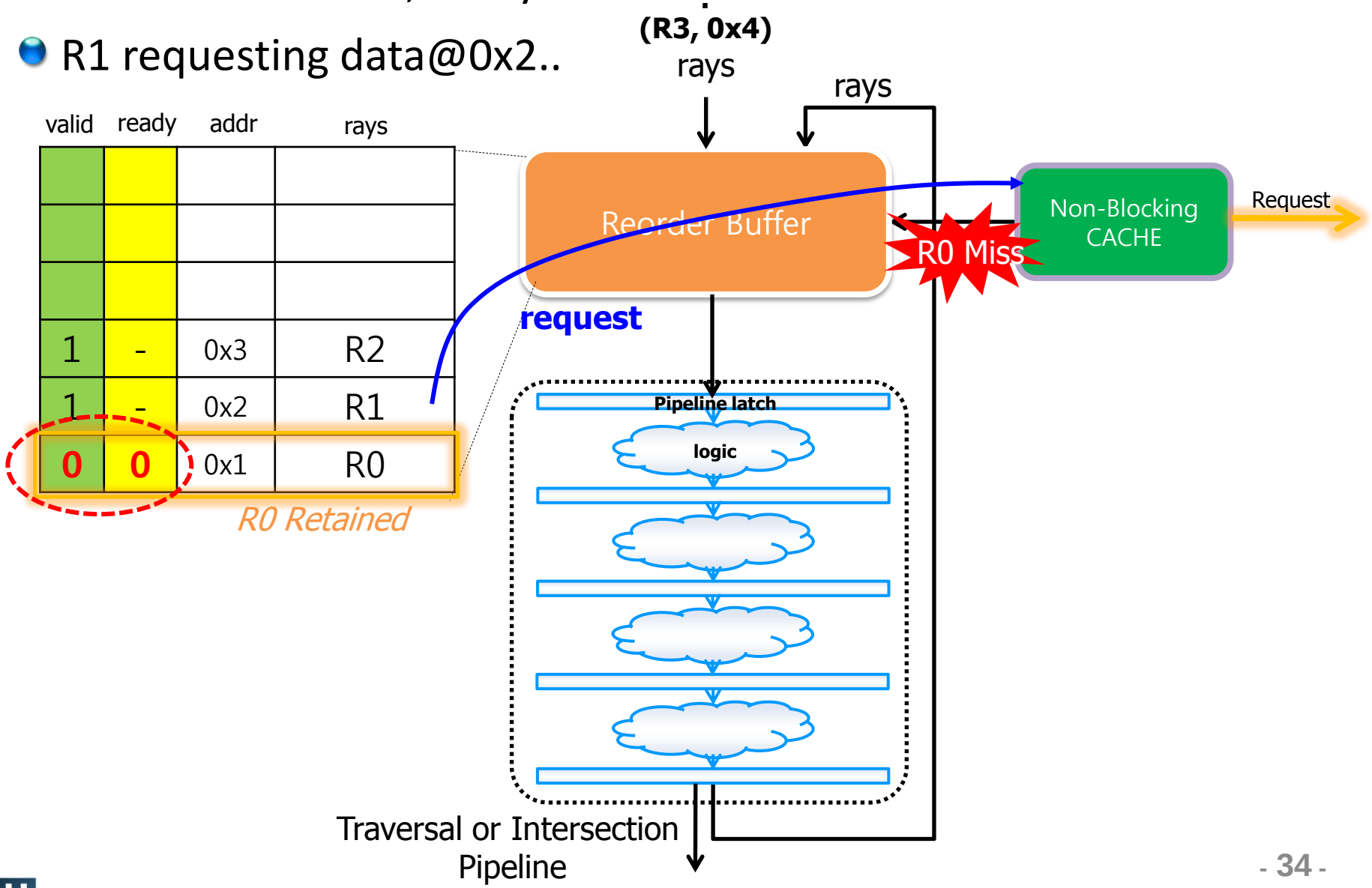
R0 requesting data@0x1

- R1 enters the buffer



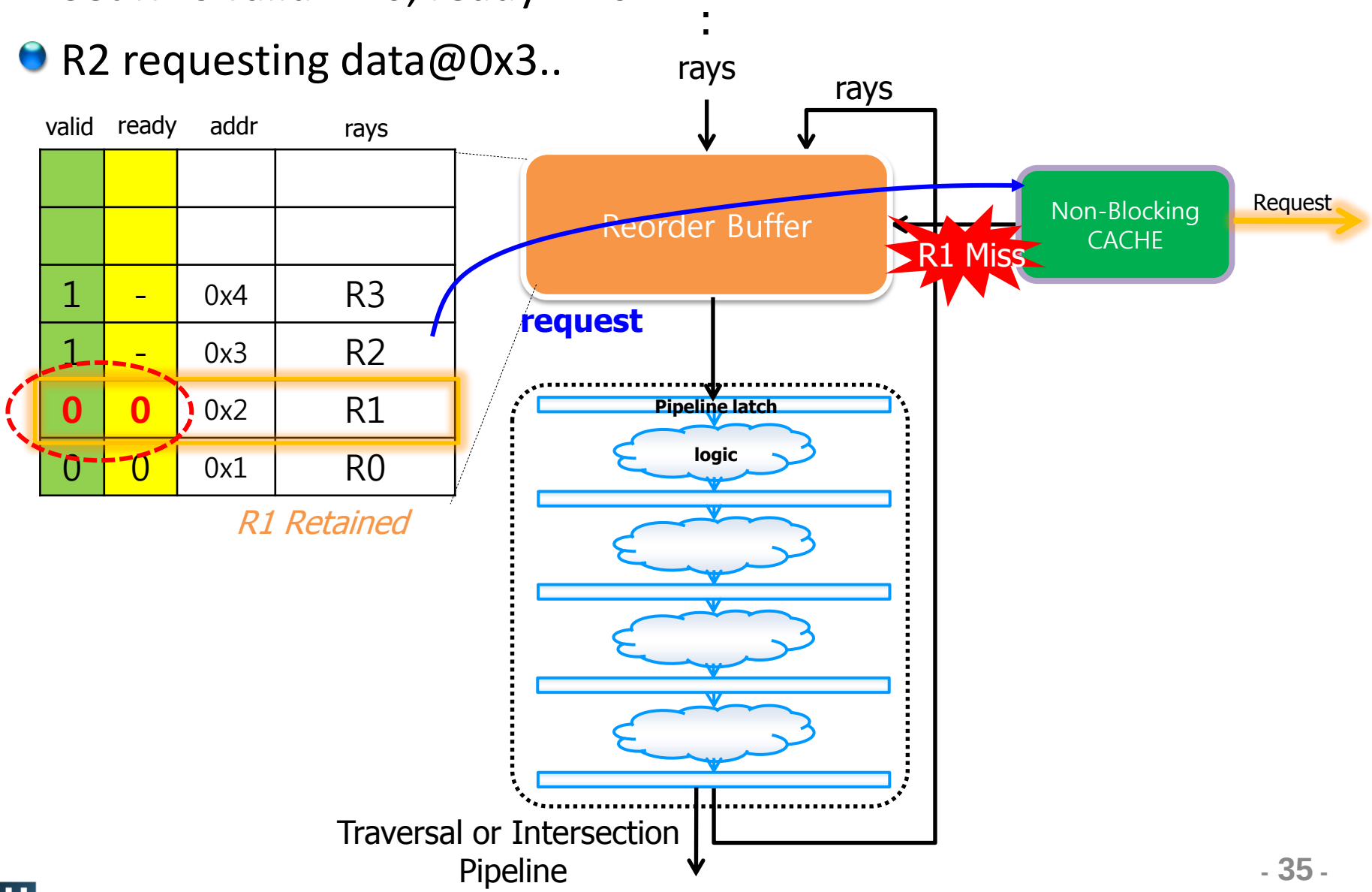
R0 missed the cache, so it is retained

- Set R0's valid ← 0, ready ← 0
- R1 requesting data@0x2..



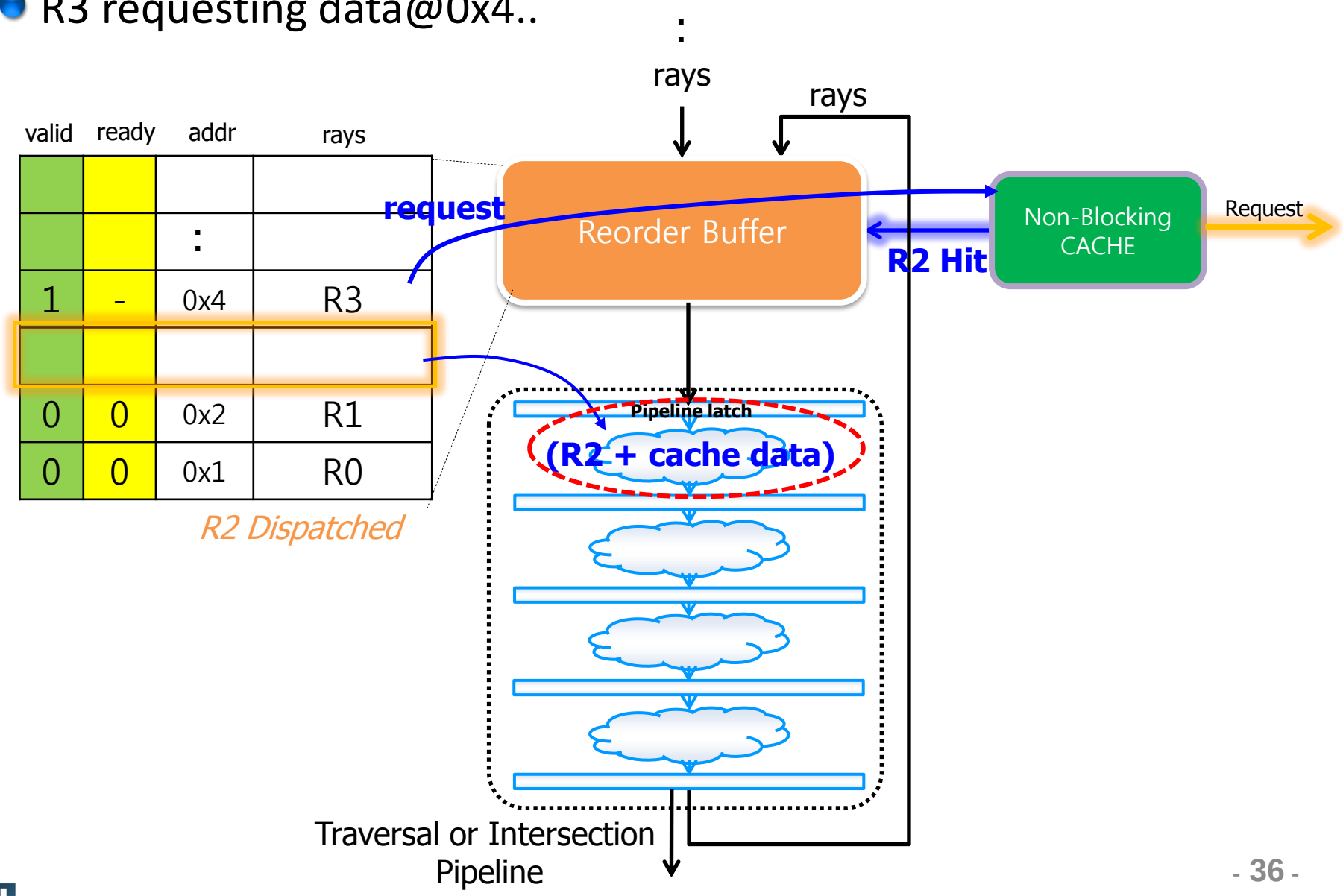
R1 missed the cache and is retained

- Set R1's valid ← 0, ready ← 0
- R2 requesting data@0x3..



R2 hits the cache, so immediately dispatched

- R3 requesting data@0x4..



Optimization

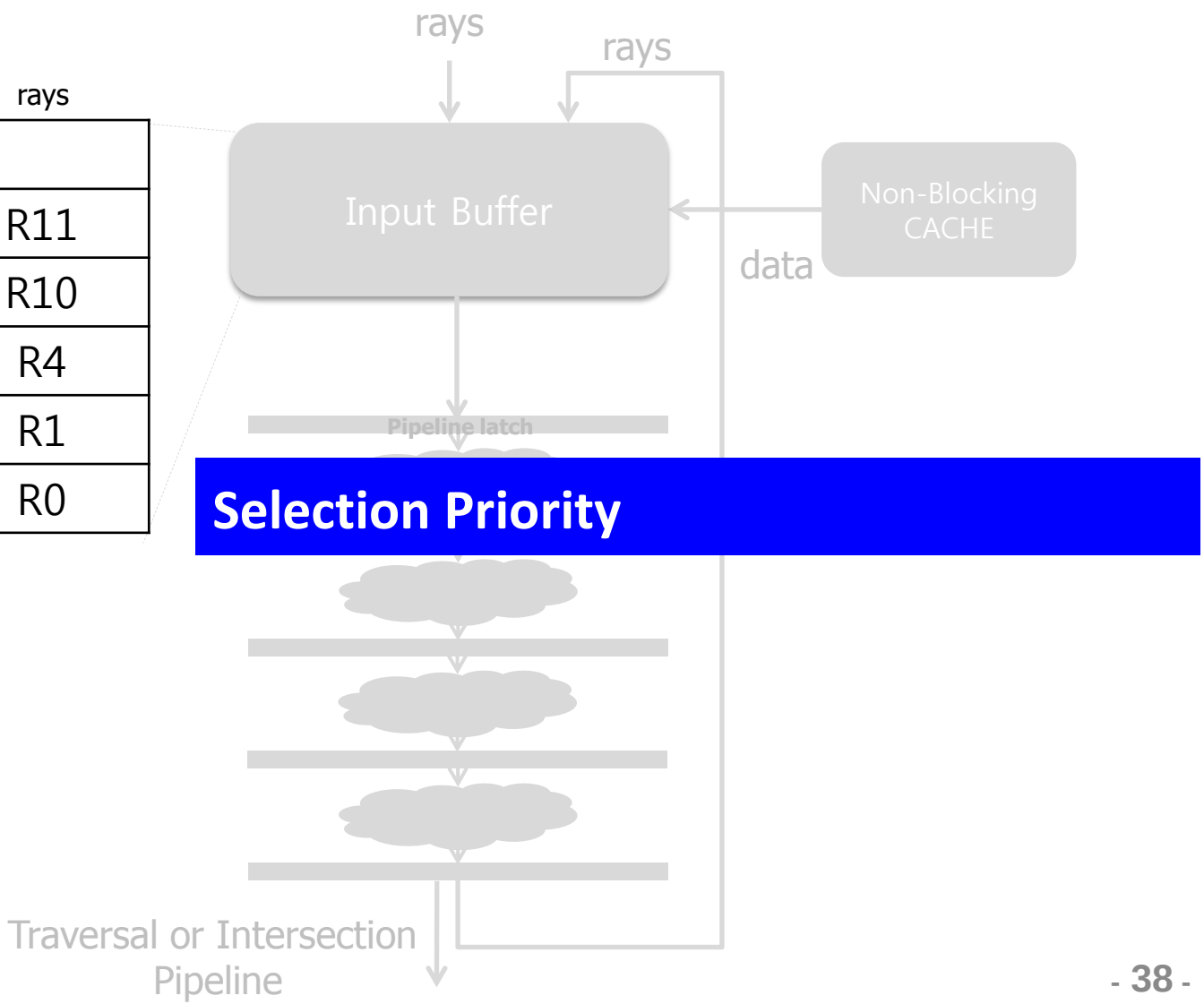
- When do the retained rays re-access the cache?
- How we prioritize the rays to fetch from buffer - the retained rays or the newly arrived rays?

→ Valid & Ready bits

Selection Priority

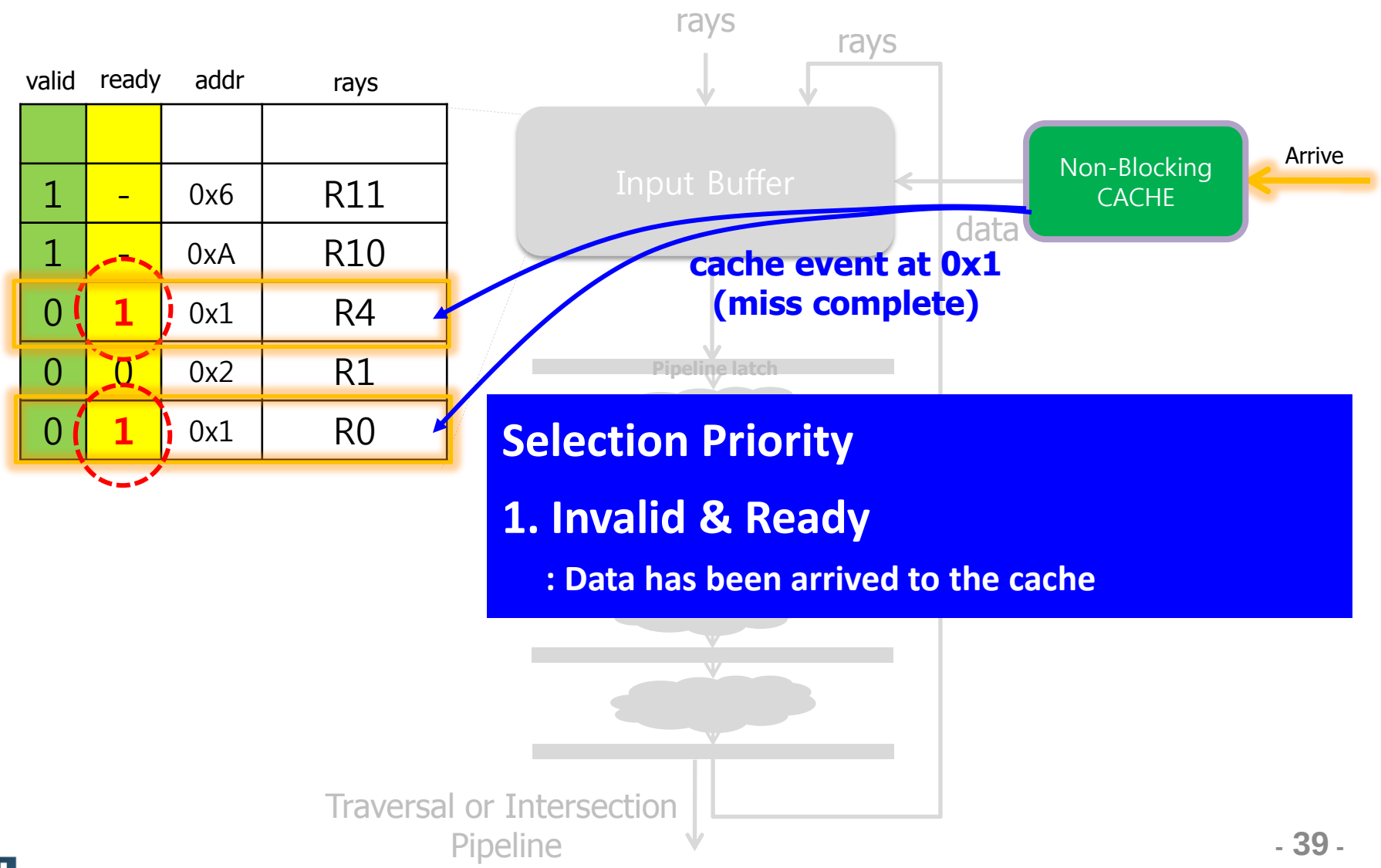
- Check the valid & ready bits:

valid	ready	addr	rays
1	-	0x6	R11
1	-	0xA	R10
0	0	0x1	R4
0	0	0x2	R1
0	0	0x1	R0



Selection Priority

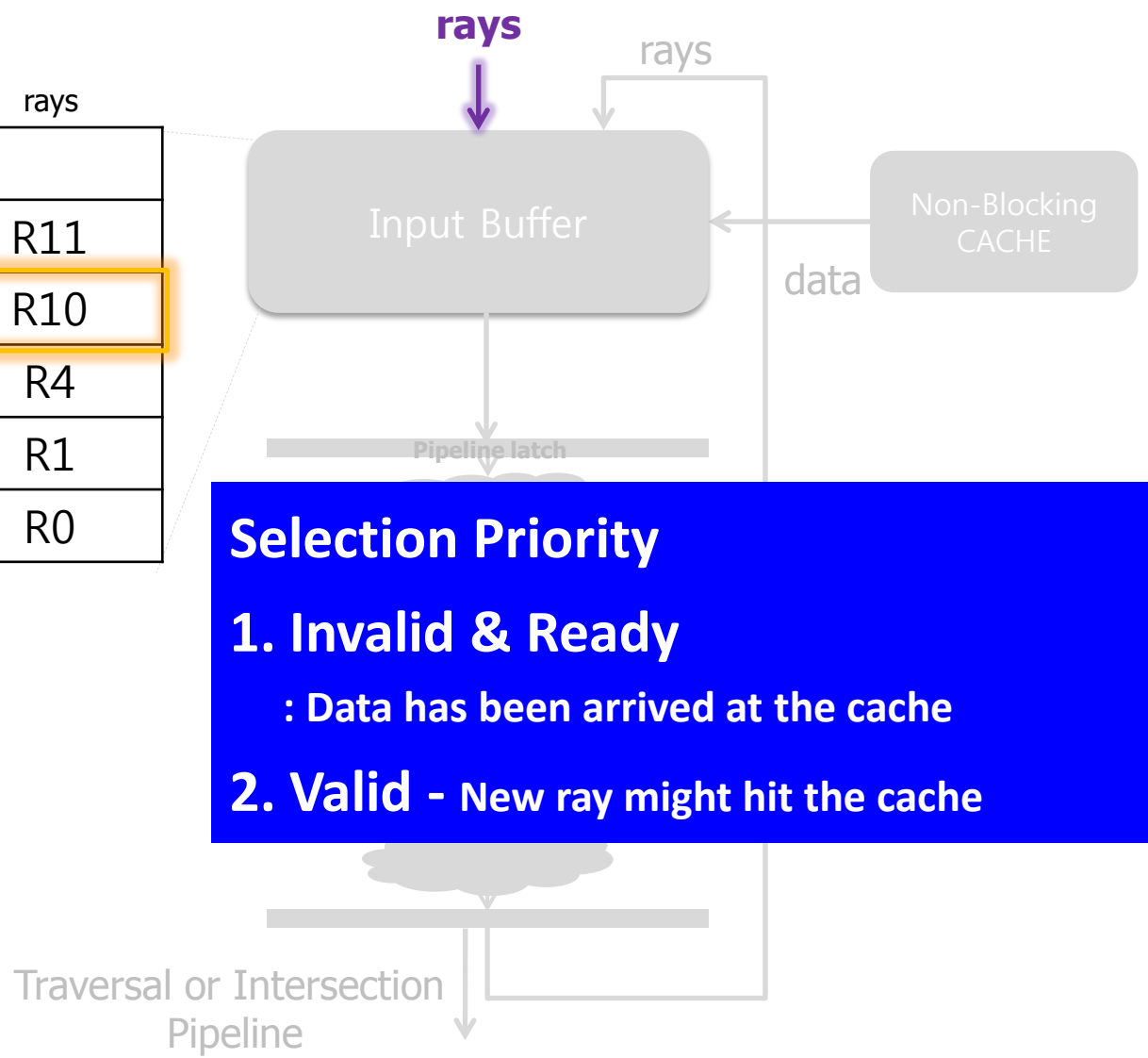
- Check the valid & ready bits:



Selection Priority

- Check the valid & ready bits:

valid	ready	addr	rays
1	-	0x6	R11
1	-	0xA	R10
0	0	0x1	R4
0	0	0x2	R1
0	0	0x1	R0



Selection Priority

1. Invalid & Ready

: Data has been arrived at the cache

2. Valid - New ray might hit the cache

Optimization

- What if the address of the newly arrived ray would be the same with the one of the retained rays?

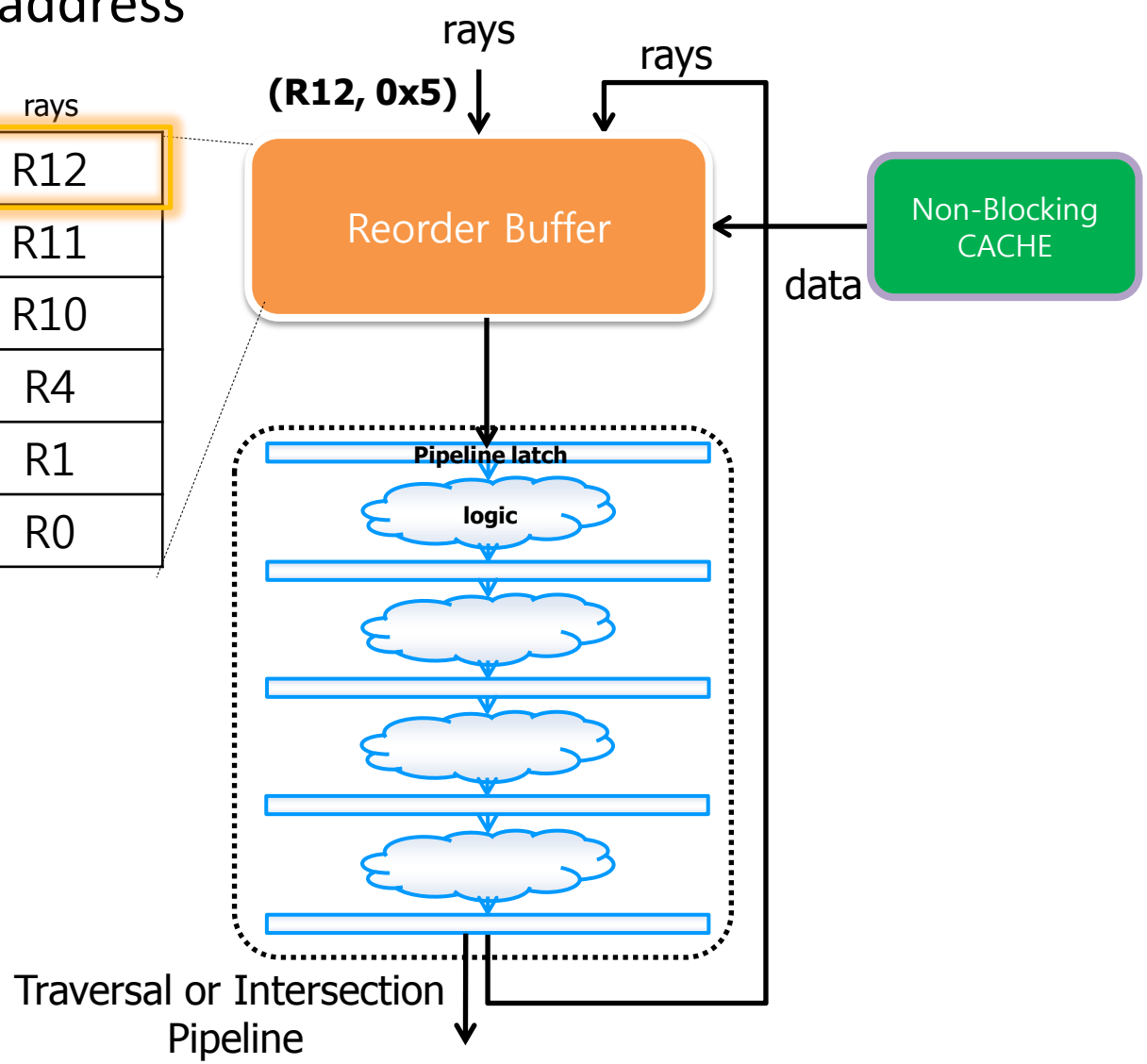
→The corresponding data was already requested and under delivery from external memory.

→Do not re-access the cache!

Redundancy control

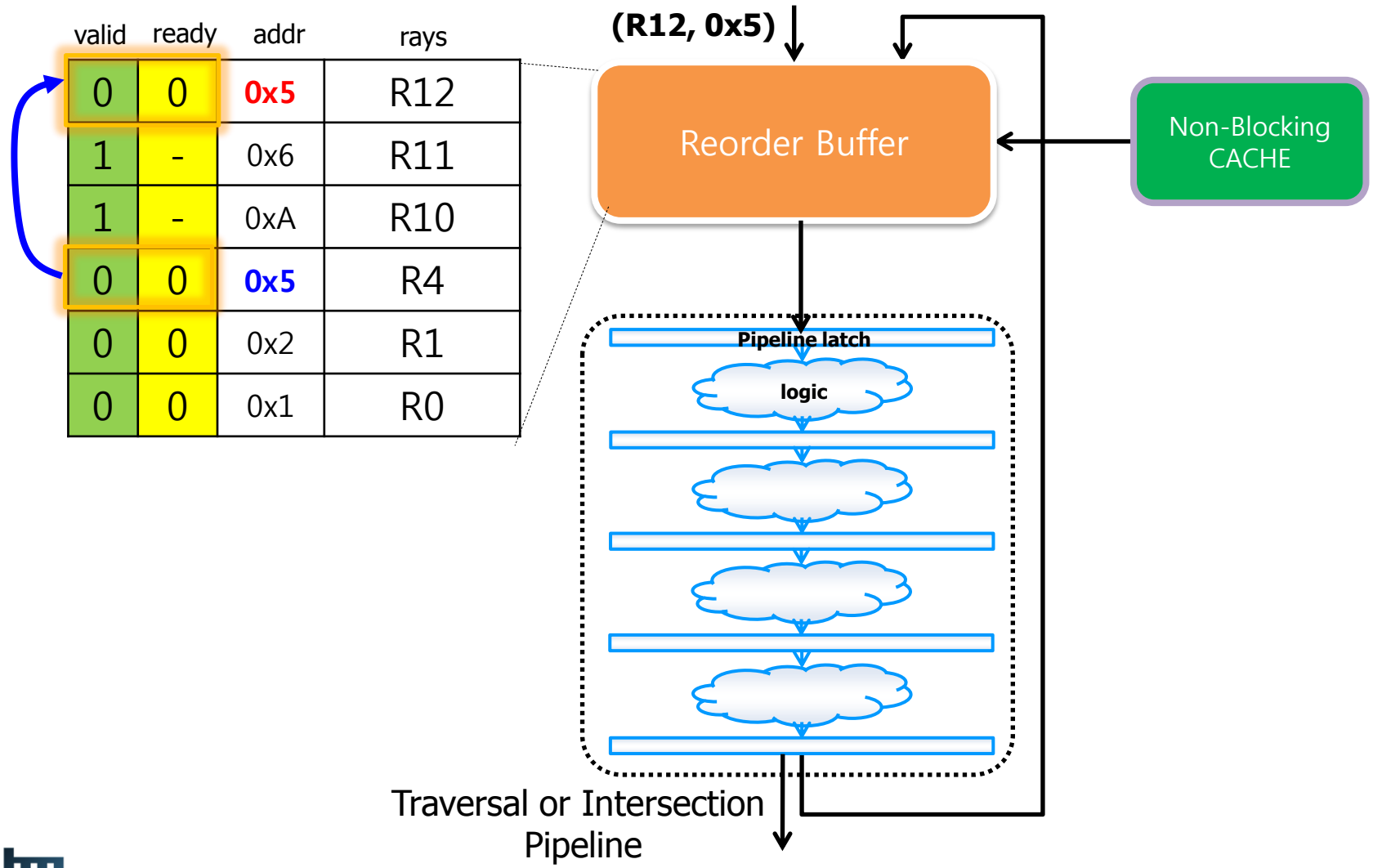
- R12 is newly arrived and referencing data@0x5, and R4 has same address

valid	ready	addr	rays
1	-	0x5	R12
1	-	0x6	R11
1	-	0xA	R10
0	0	0x5	R4
0	0	0x2	R1
0	0	0x1	R0



Redundancy control

- Update valid bit & counter to avoid unnecessary cache access and increase ray coherency (R4, R12)



Advantages

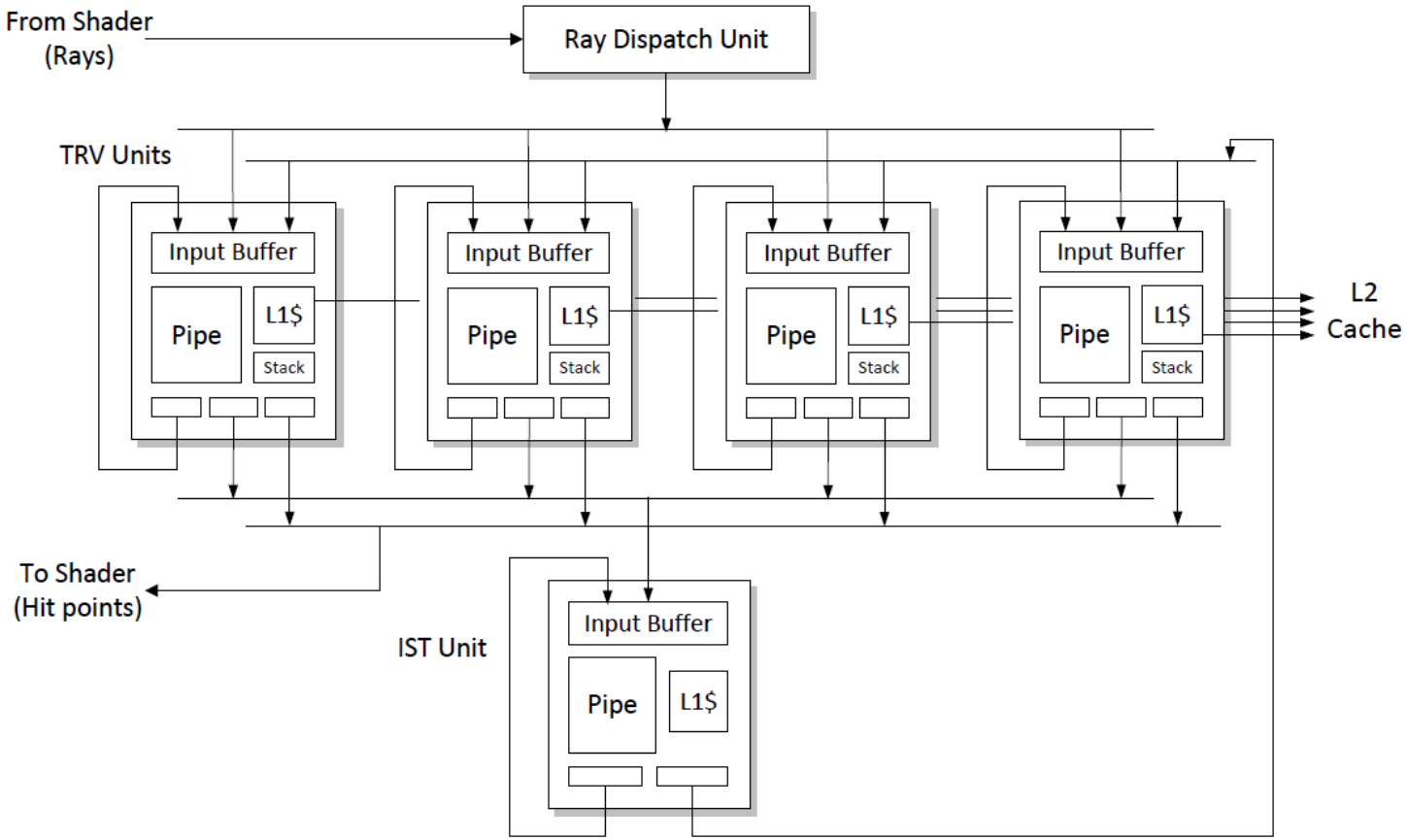
- Cost-effective and energy-efficient
  - Minimal extension of existing H/W
  - No bypass

- *Reordering* in input buffer
  - Multithreading
  - Increase ray coherency

EVALUATION

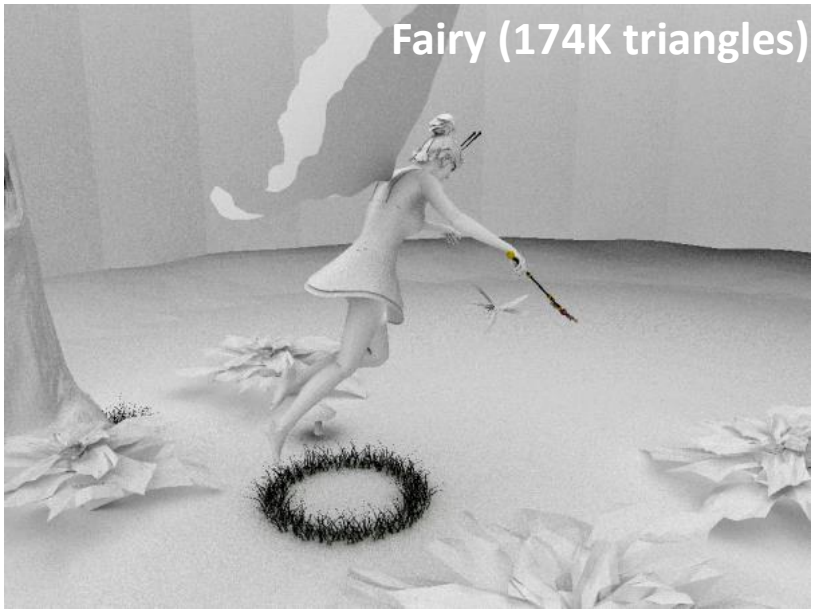
Experimental setup – baseline architecture

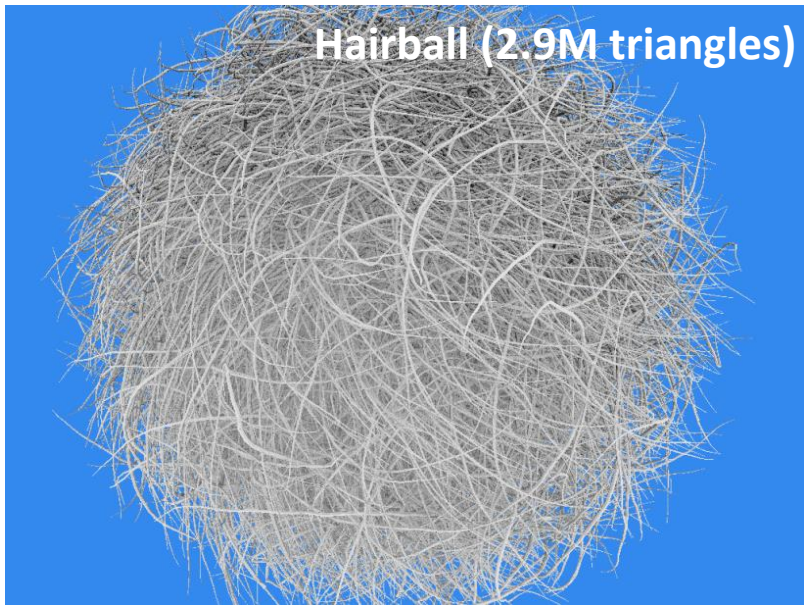
- T&I Units, SGRT [Lee et al 2013] without any multithreading.



Experimental setup – test scenes

All scenes are rendered by diffused path tracing





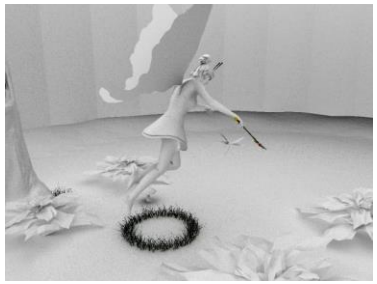
Experimental Setup - Simulation

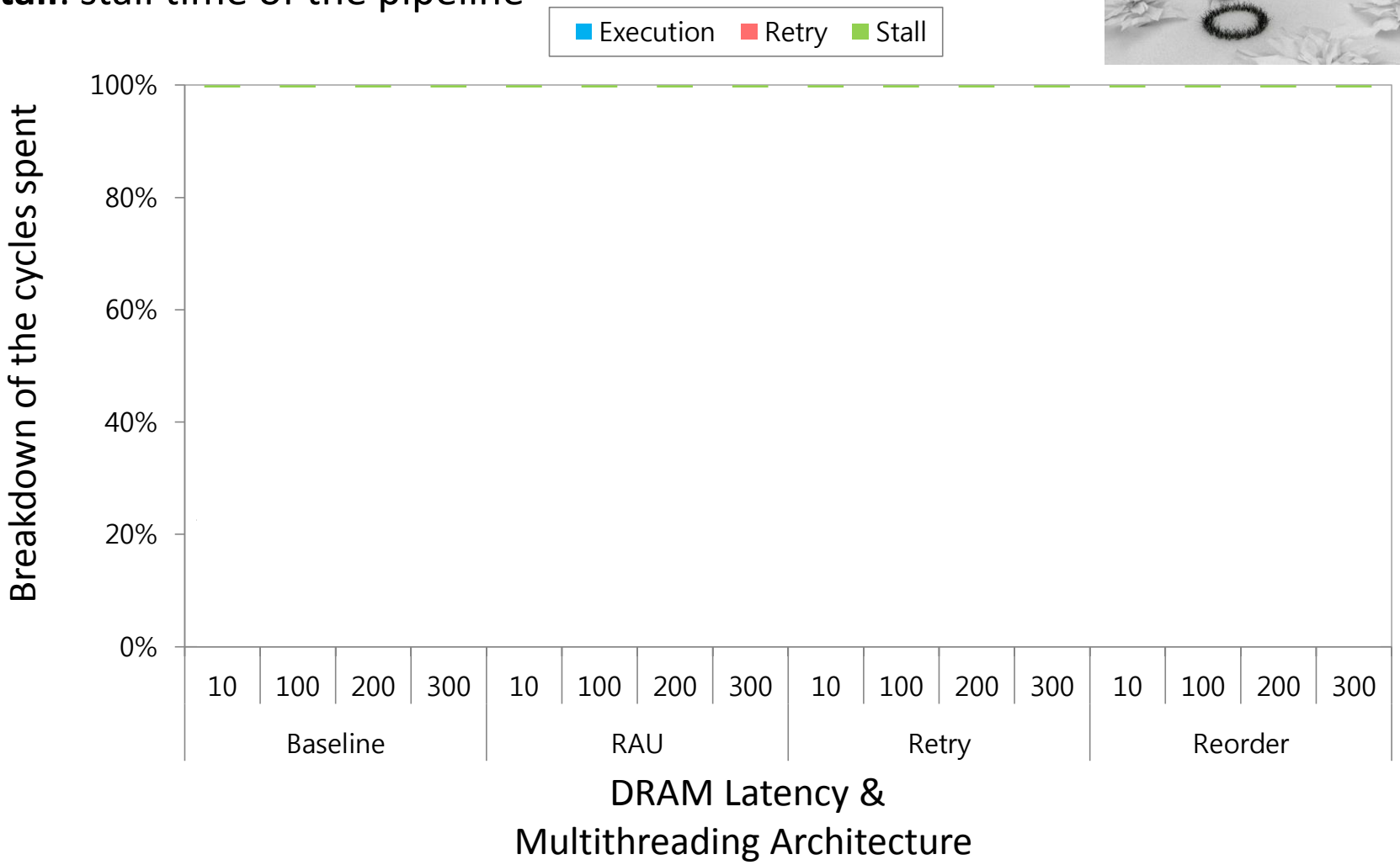
- Comparison: different multithreading scheme..
  - Baseline, RAU, Retry, Reorder
- Cycle accurate, performance simulation
  - Based on SGRT [Lee et al 2013]
- Energy simulation of memory elements
  - CACTI 6.5 [Muralimanohar 2007]

- Configuration of MIMD Traversal H/W
  : Clock (500MHz), Latency L1(1) / L2(20) / DRAM (10, 100, 200, 300)
- All scenes are rendered by diffused path tracing

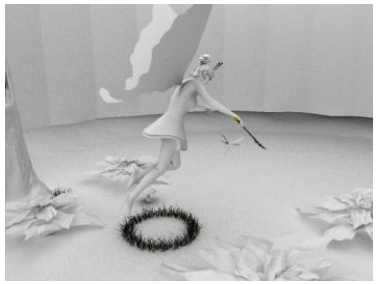
Pipeline utilization with varying the DRAM latency

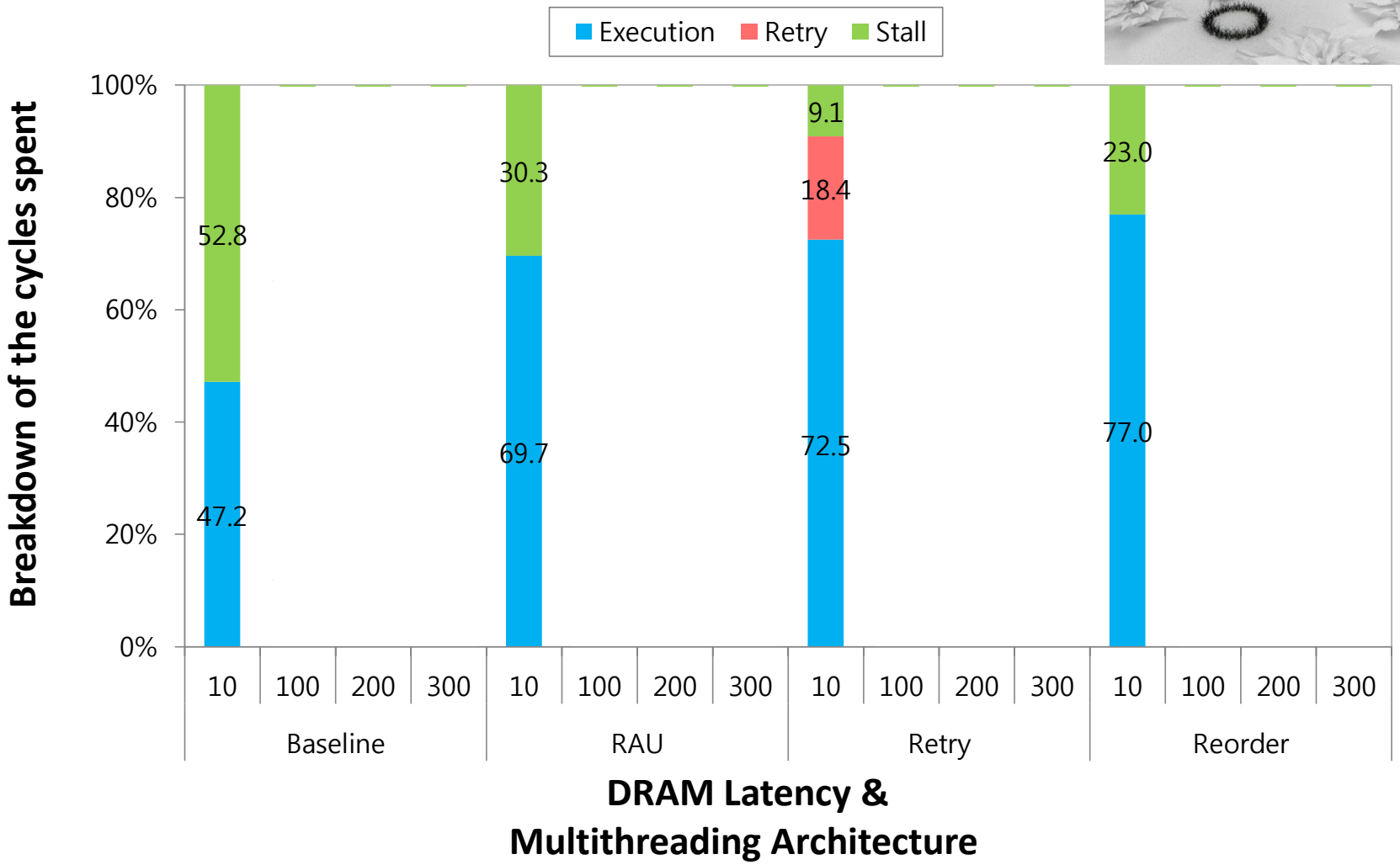
- **Execution**: effective running time in the pipeline
- **Retry**: bypass time of the invalidated ray in the pipeline
- **Stall**: stall time of the pipeline



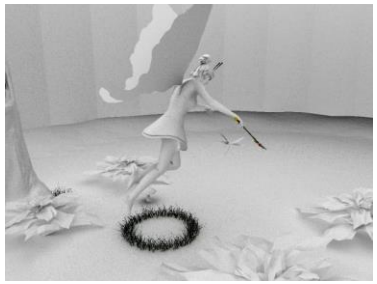


At latency 10: RAU, Retry, and Reorder recorded similar utilization level

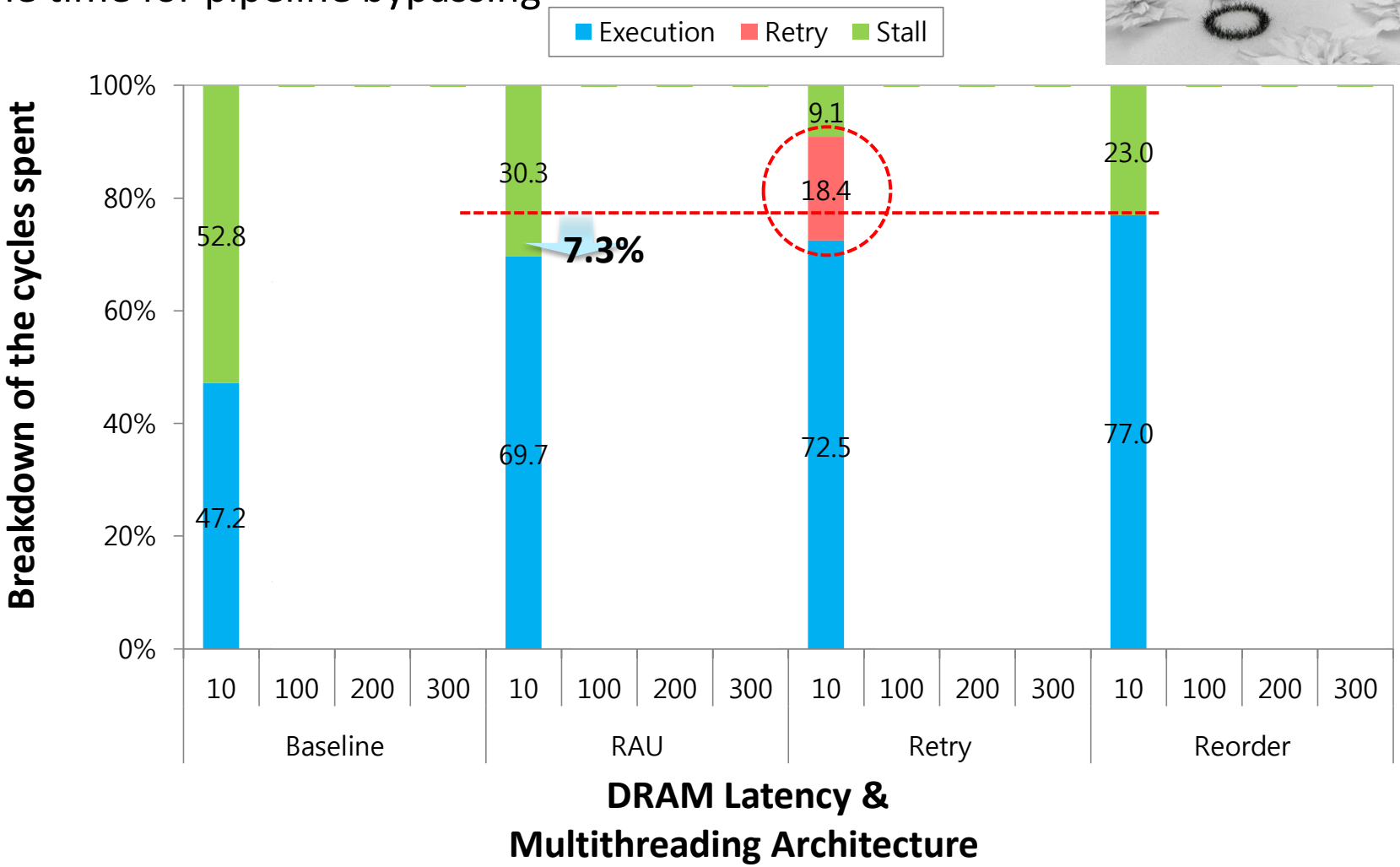




At latency 10: RAU, Retry, and Reorder recorded similar utilization level

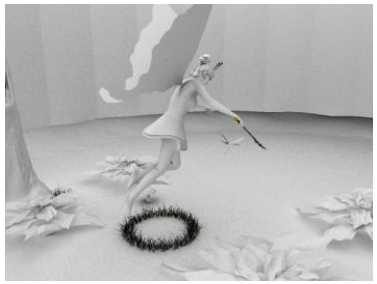


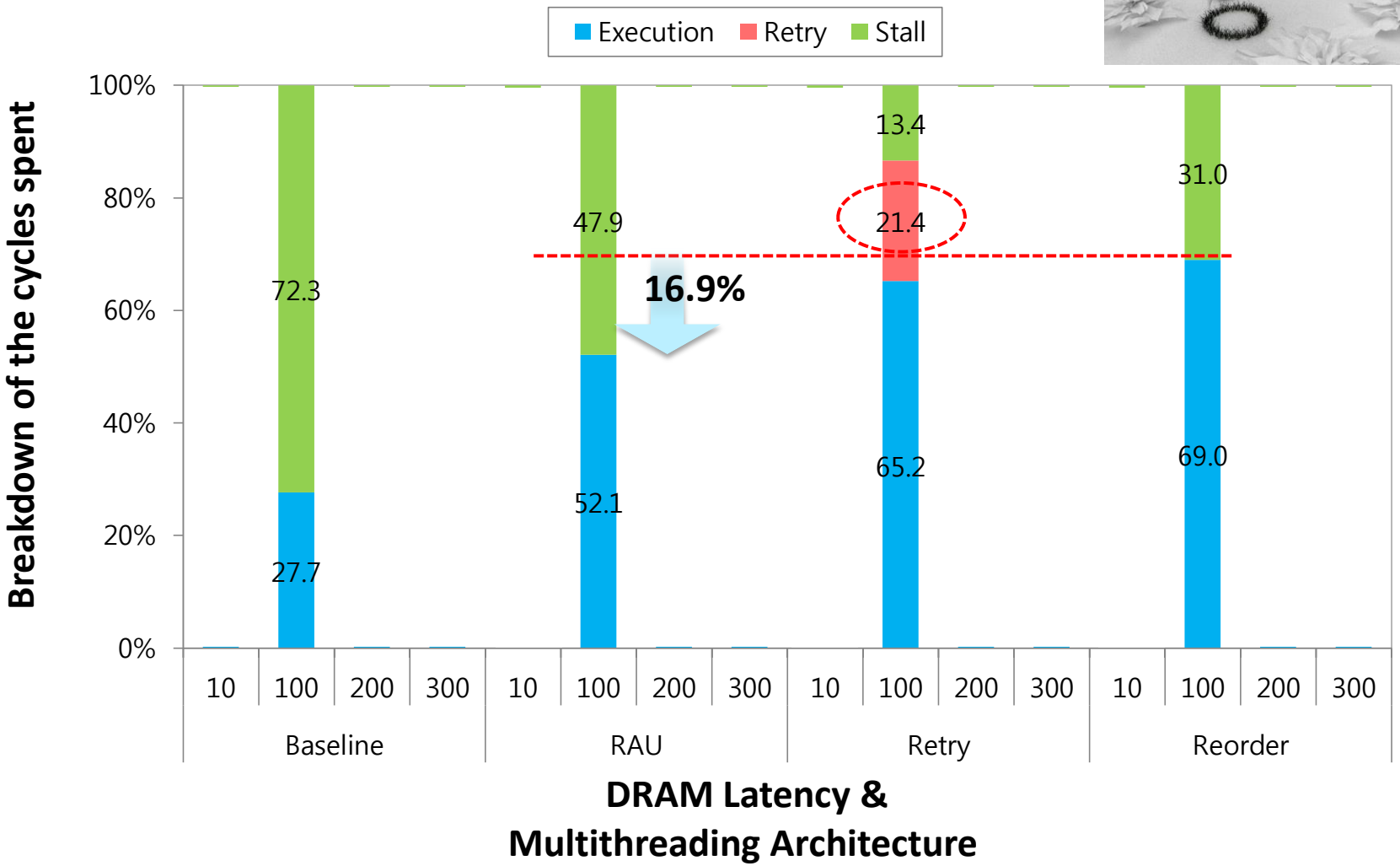
- But, RAU stalls pipeline a bit more, and Retry consumes the time for pipeline bypassing



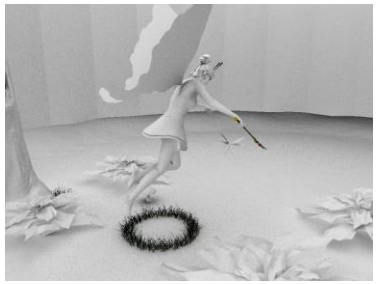
At latency 100: Retry, and Reorder recorded similar utilization level

- RAU stalls pipeline more, and Retry retries a bit more

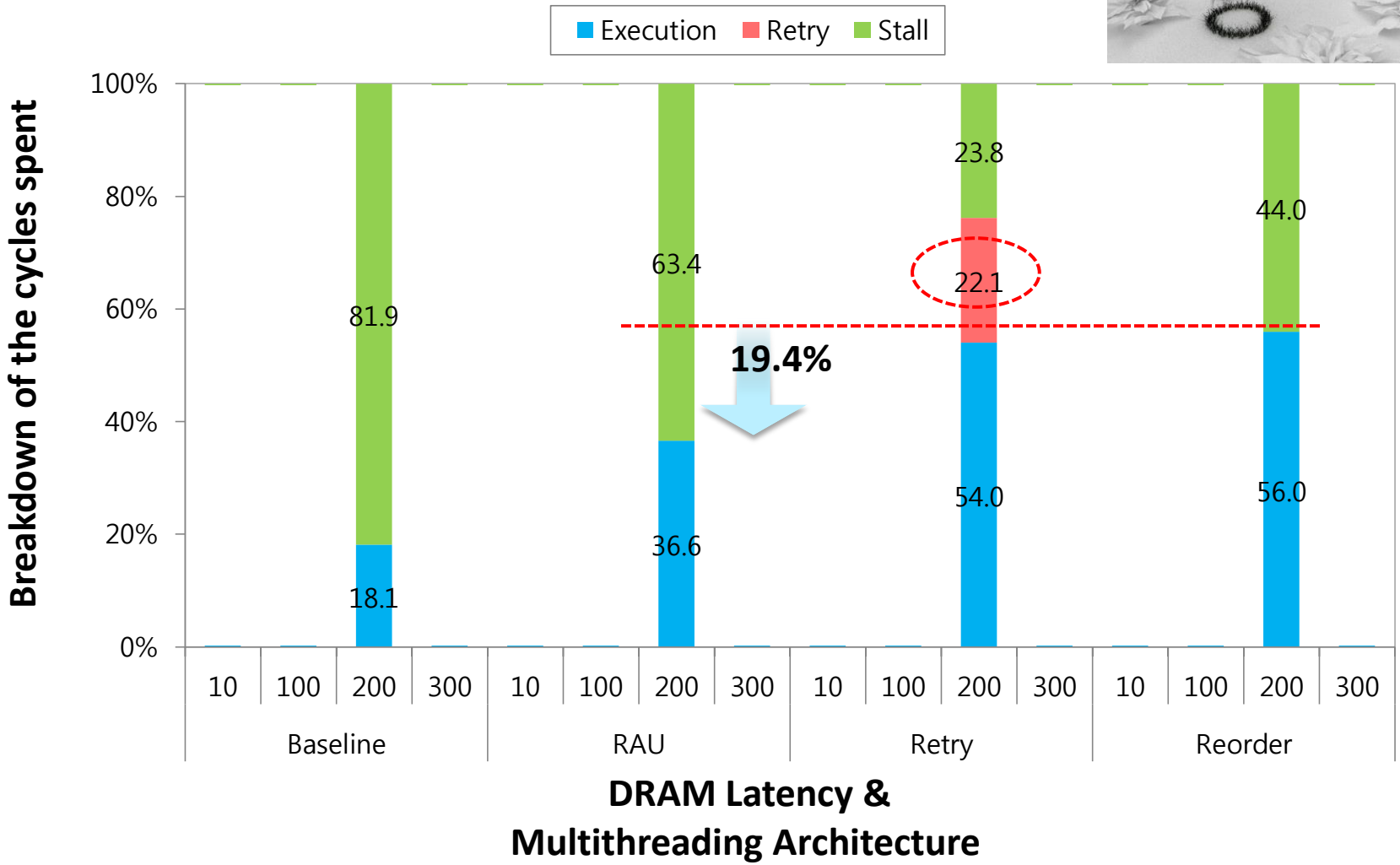




At latency 200: Retry, and Reorder recorded similar utilization level

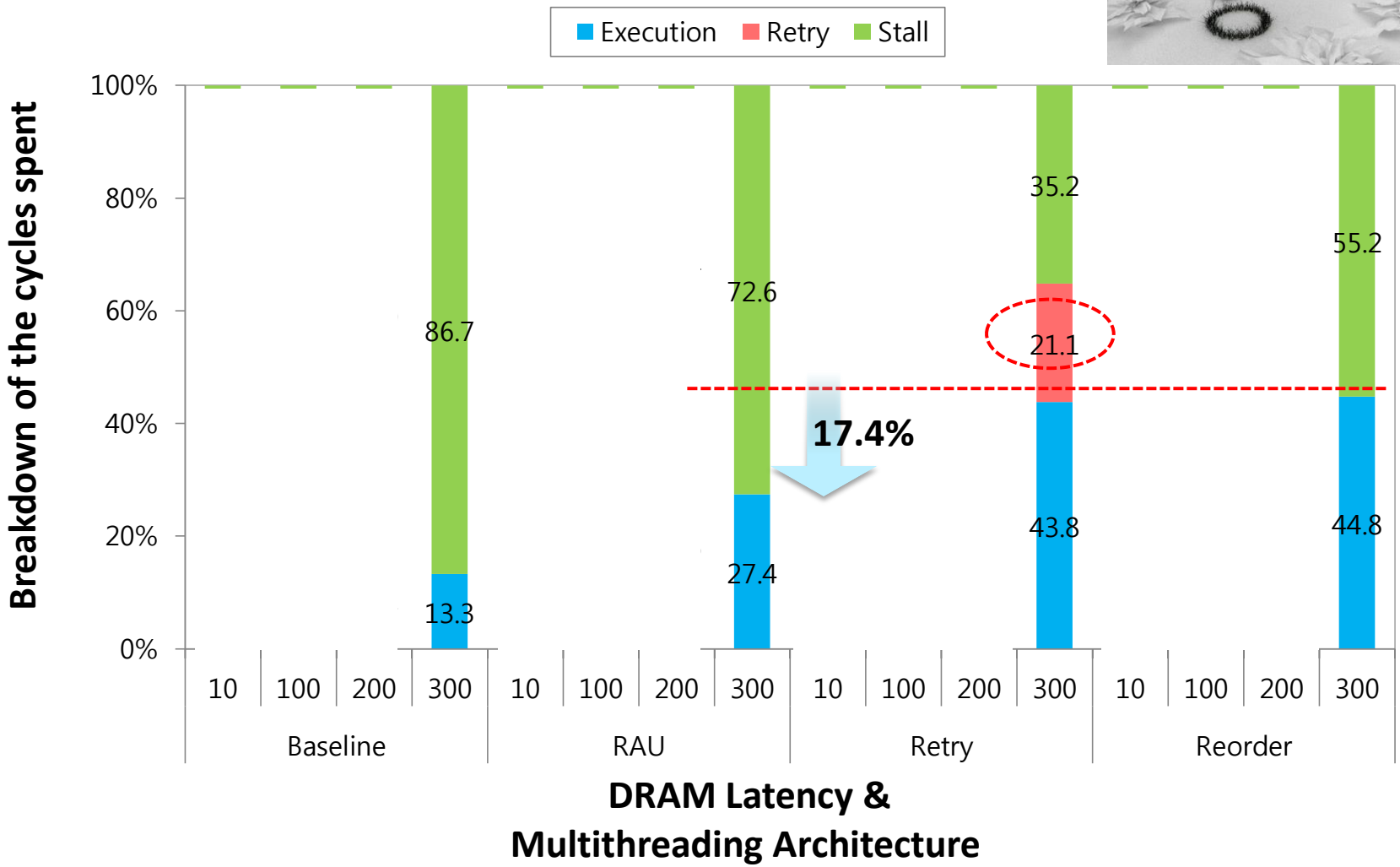


- RAU spent more time for stall than execution and Retry retries caching a bit more

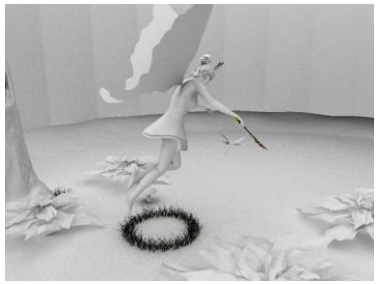


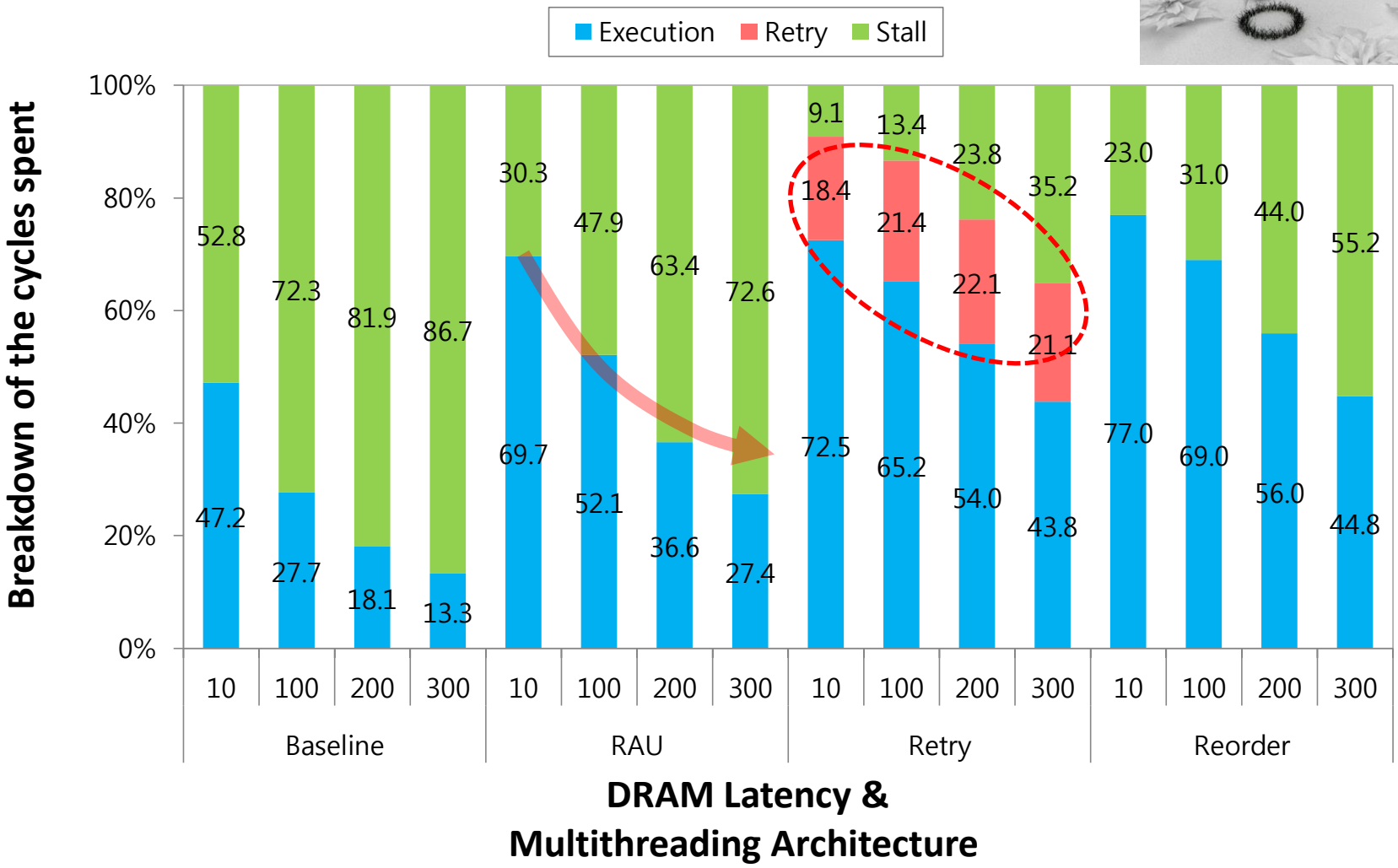
At latency 300: RAU spent most stall time and Retry spent the 21% of the time for pipeline iteration

- Stall time → lower performance
- Retry time → higher energy consumption



As the DRAM latency is increased, RAU drops sharply and Retry consumes more retry times

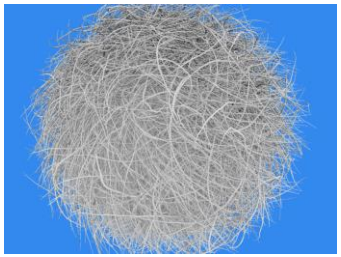


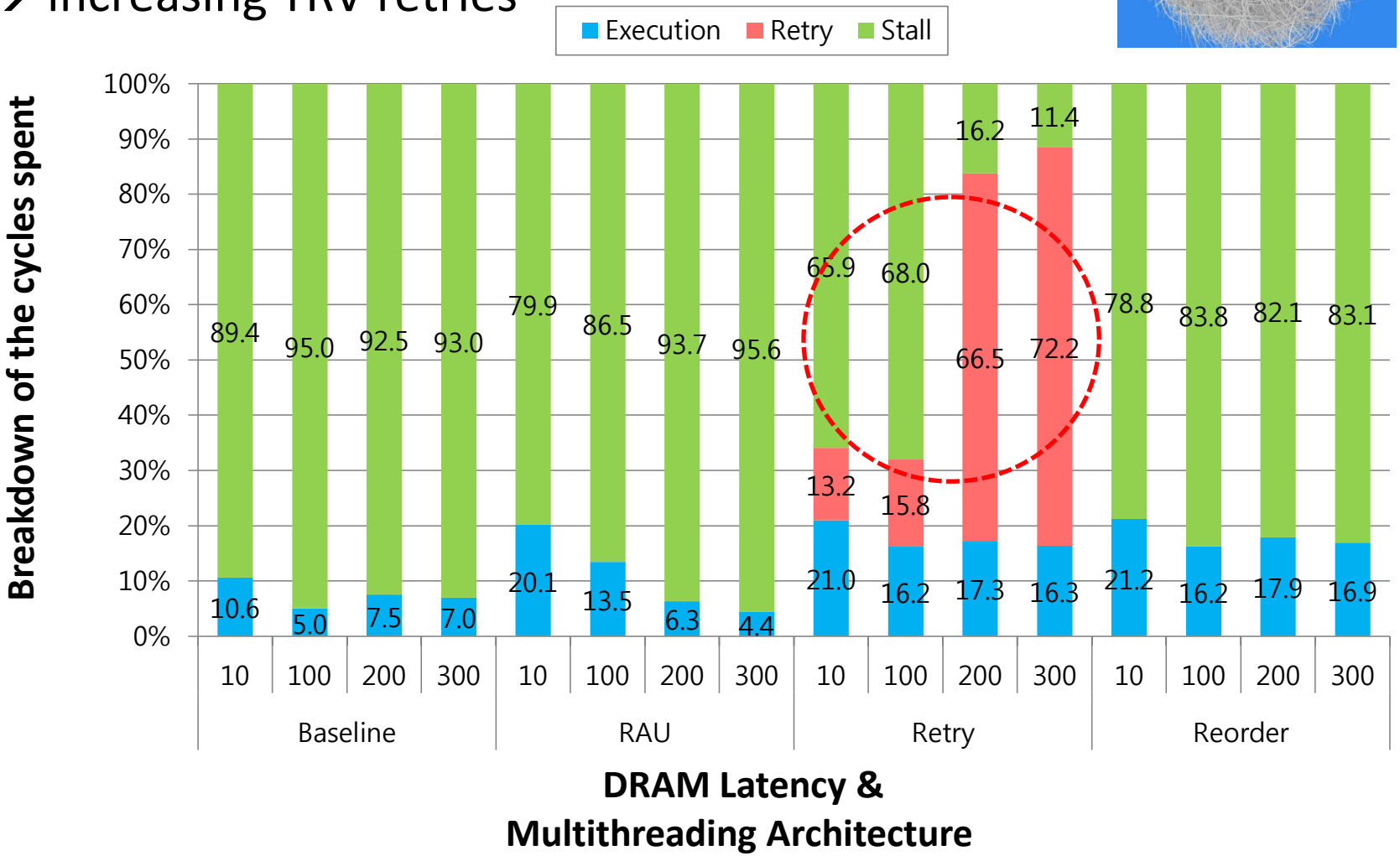


In high complex scene, retry times are increased sharply (latency is more than 200)

- Longer DRAM latency caused to IST bottleneck

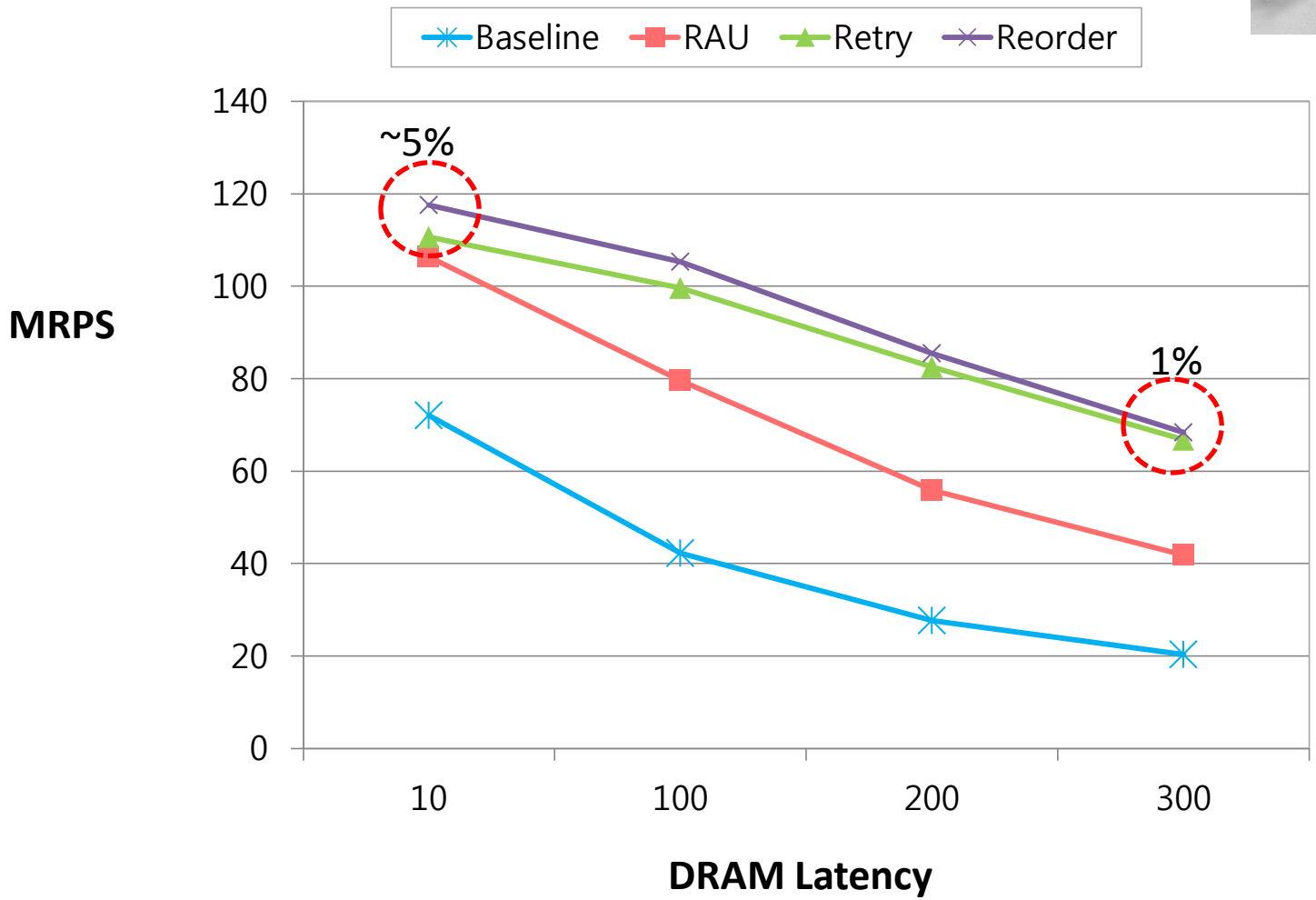
  → increasing TRV retries





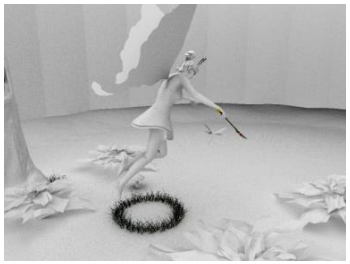
Performance with varying the DRAM latency

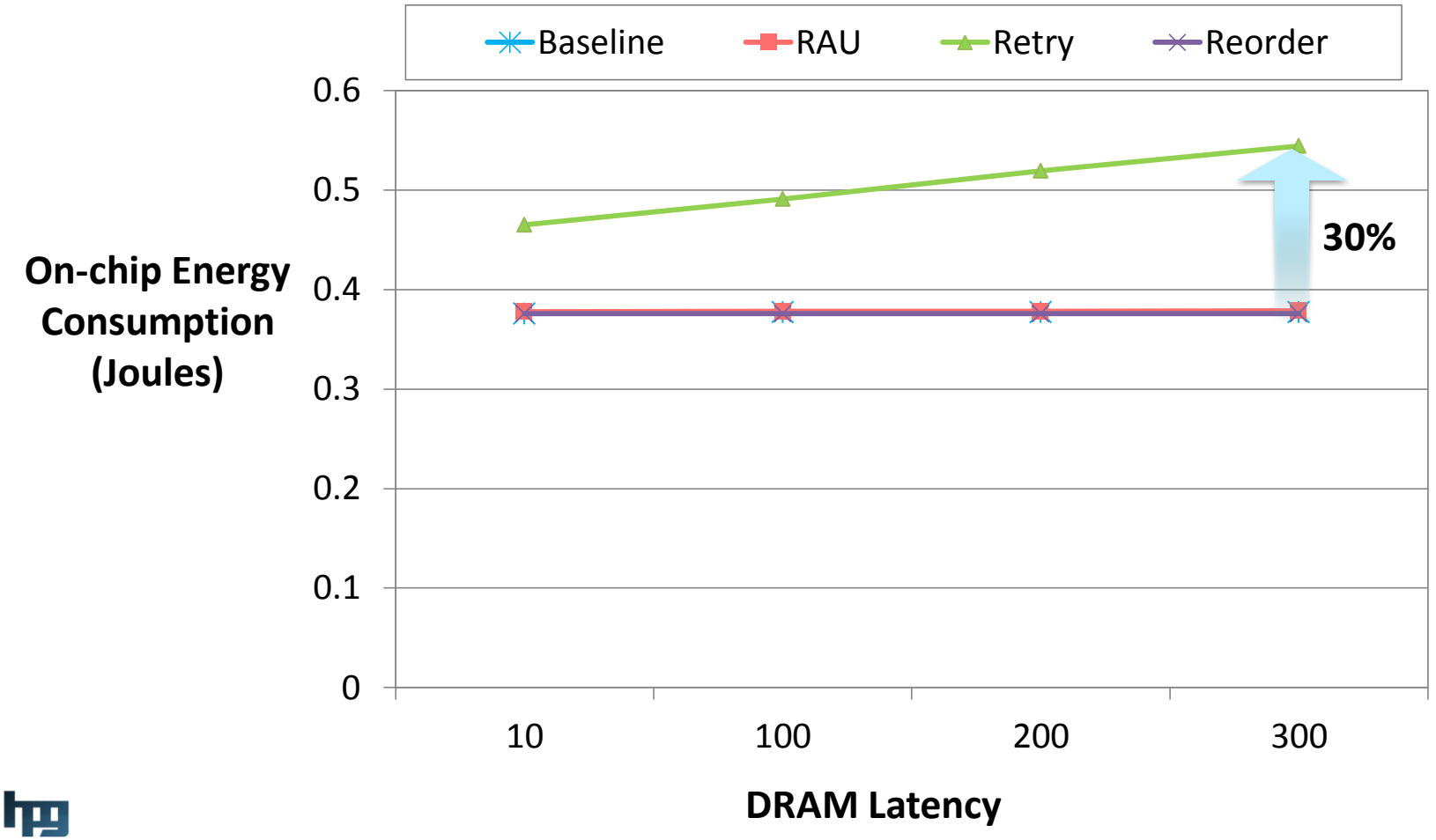
- Reorder Buffer slightly outperforms Retry
  due to the better cache utilization with ray-reordering (1~5%)
- RAU drops sharply by lower cache & pipeline utilization



On-chip memory energy (registers, buffer, SRAM, L1/L2 cache) consumption with varying the DRAM latency

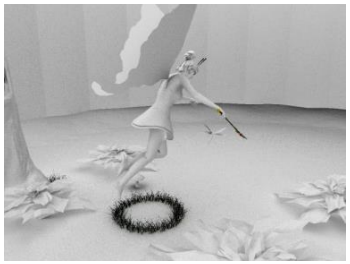
- Retry records the highest consumption because the bypass makes the number of accesses to the pipeline register and the SRAM buffer increases.

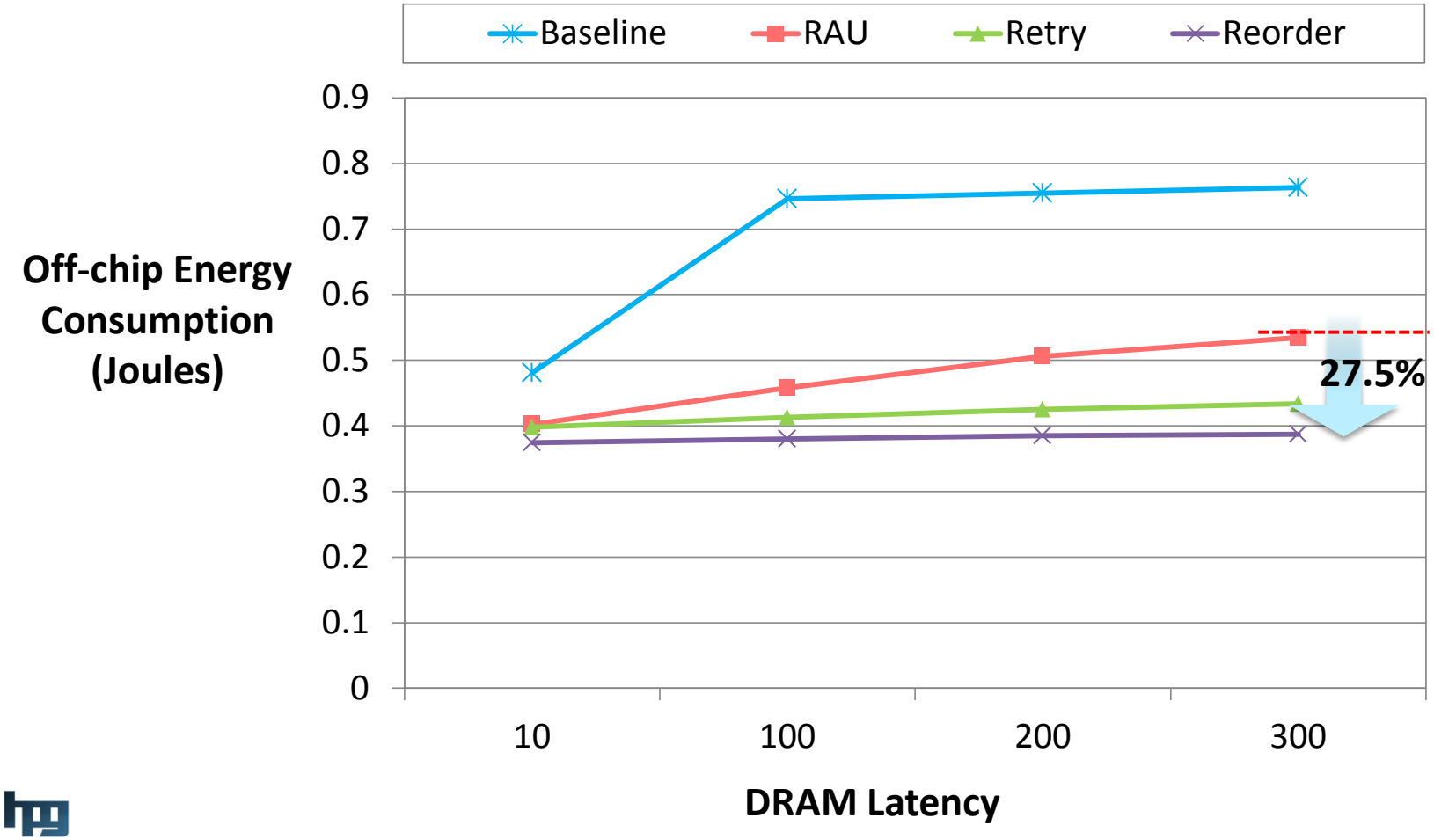




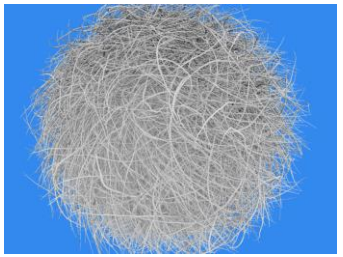
Off-chip memory energy (DRAM) consumption with varying the DRAM latency

- It is purely proportional to the miss ratio of the L2 cache.
- Reorder Buffer could reduce the power consumption up to 27.5% by the better cache utilization (vs. RAU)

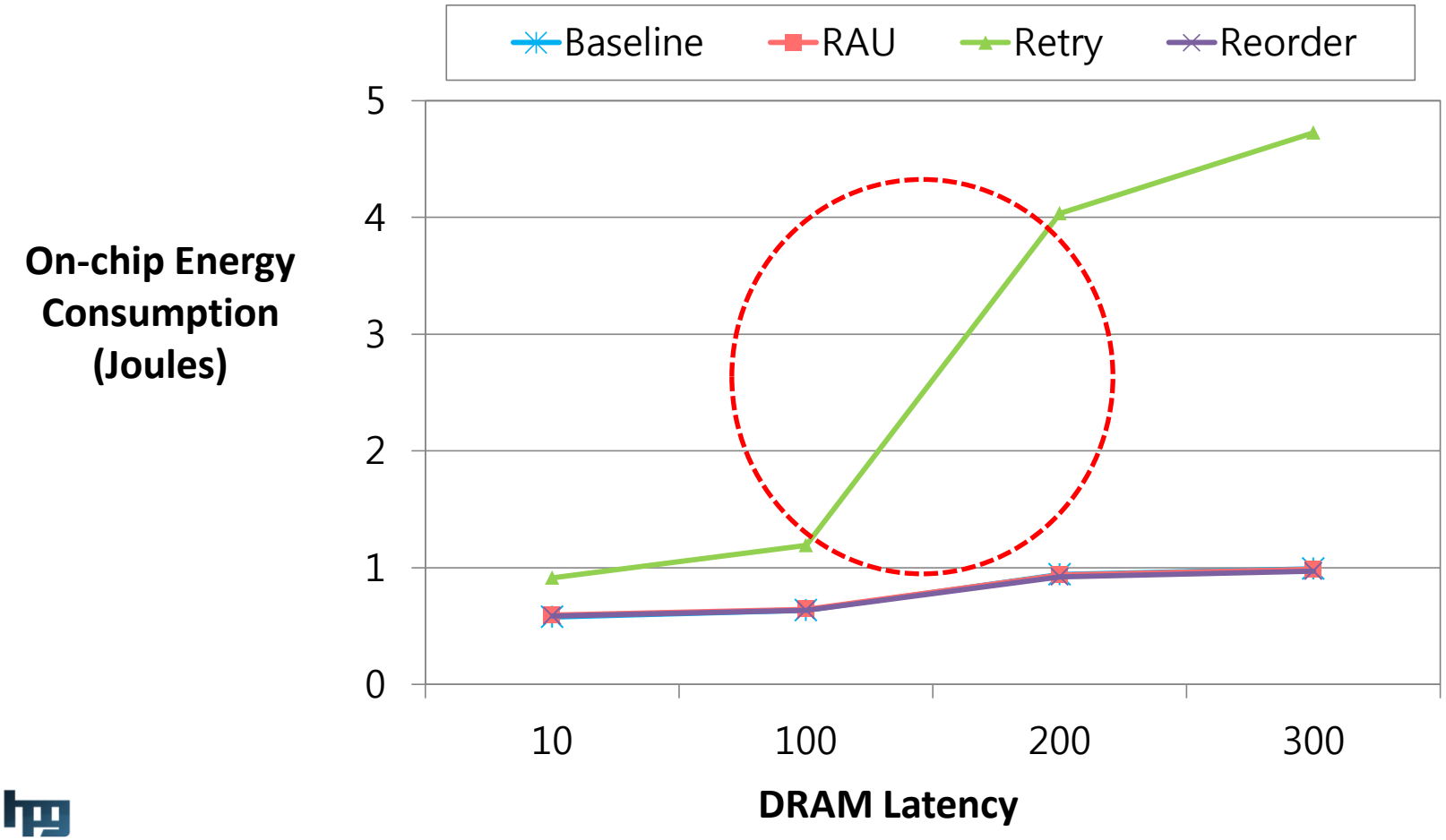




On-chip memory energy consumption with varying the DRAM latency (complex scene)

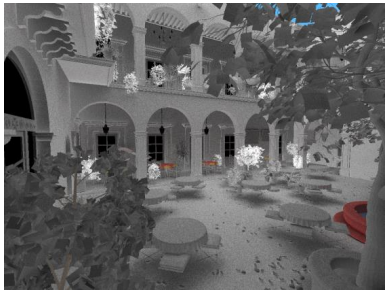


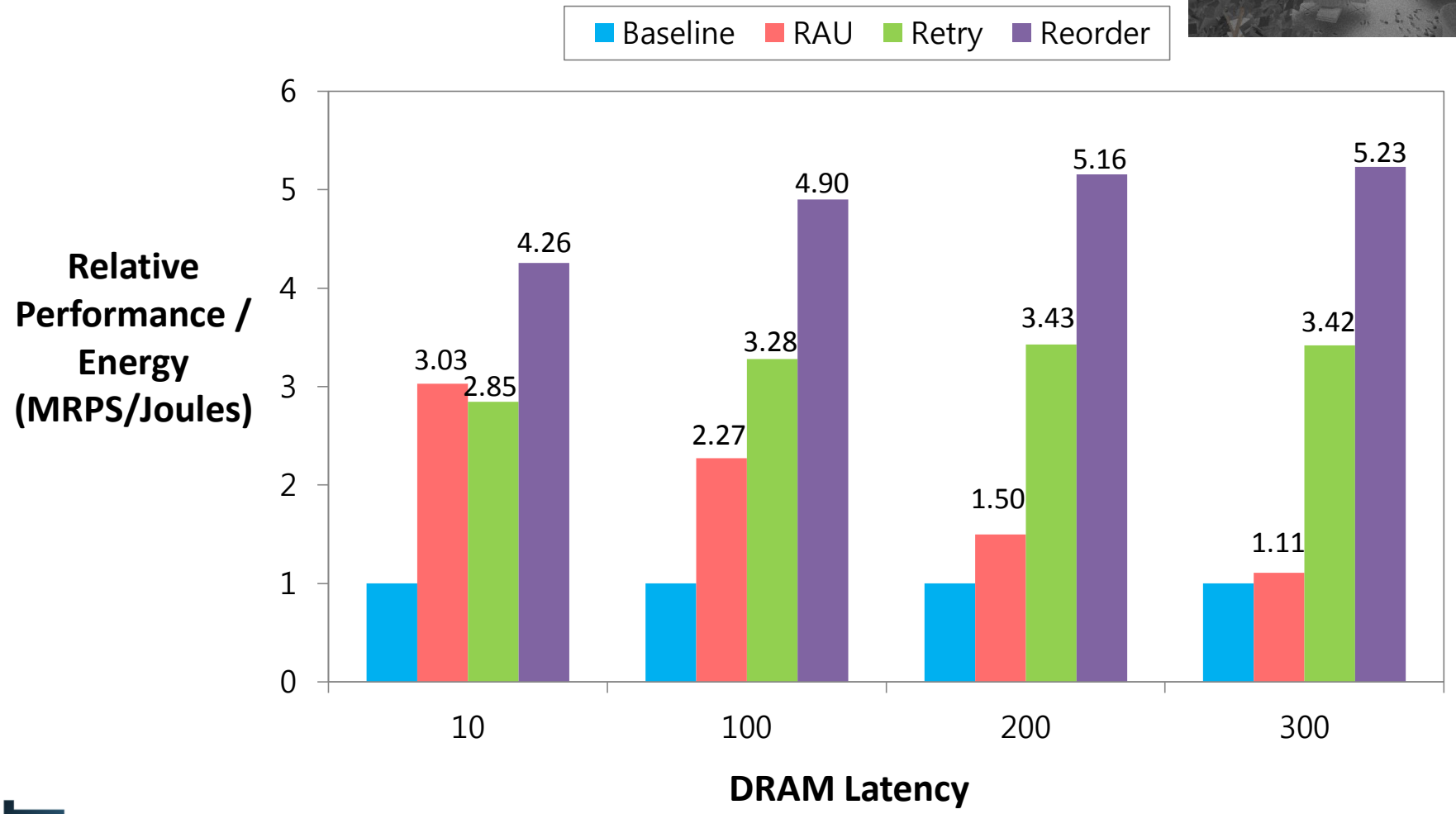
- Retry ratio sharply increases in more than 200 cycles, which causes 3.4 times more energy to be consumed when the latency moves from 100 to 200.



Relative Performance / Energy

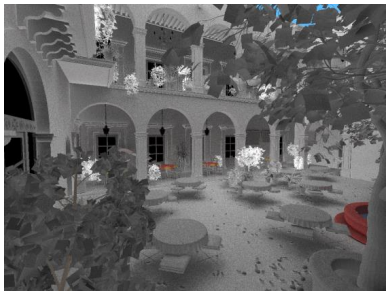
- It is the relative number to the baseline arch.

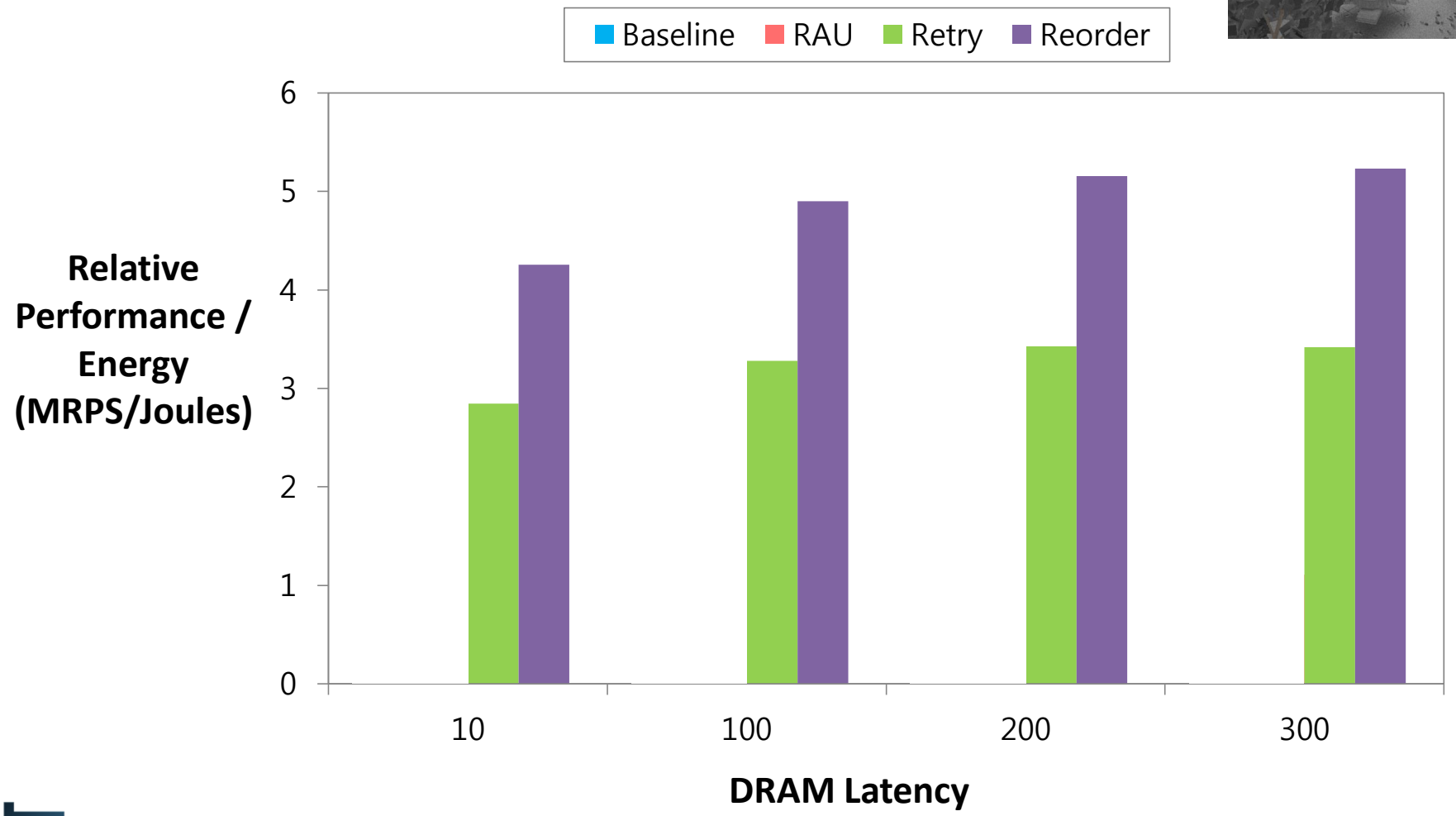




Relative Performance / Energy

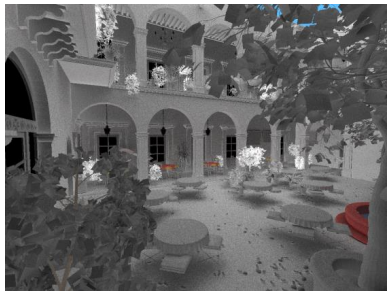
- Retry vs. Reorder…

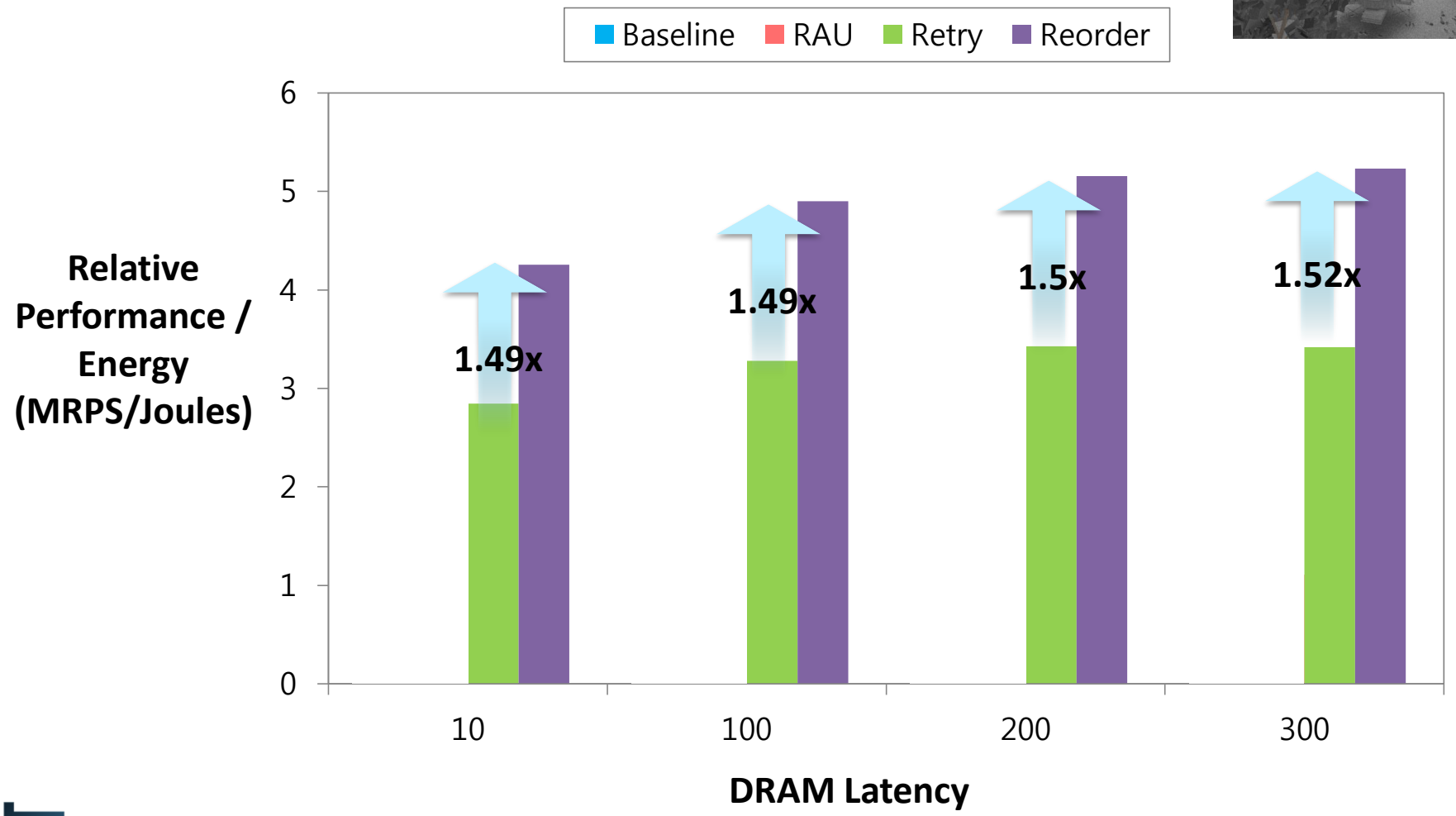




Relative Performance / Energy

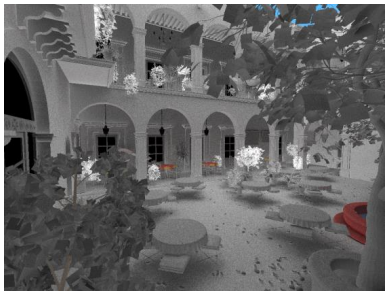
- Reorder Buffer achieves up to 1.52x better efficiency.
- Retry records a similar performance with the Reorder Buffer, but, its bypassing feature consumes much more on-chip energy.

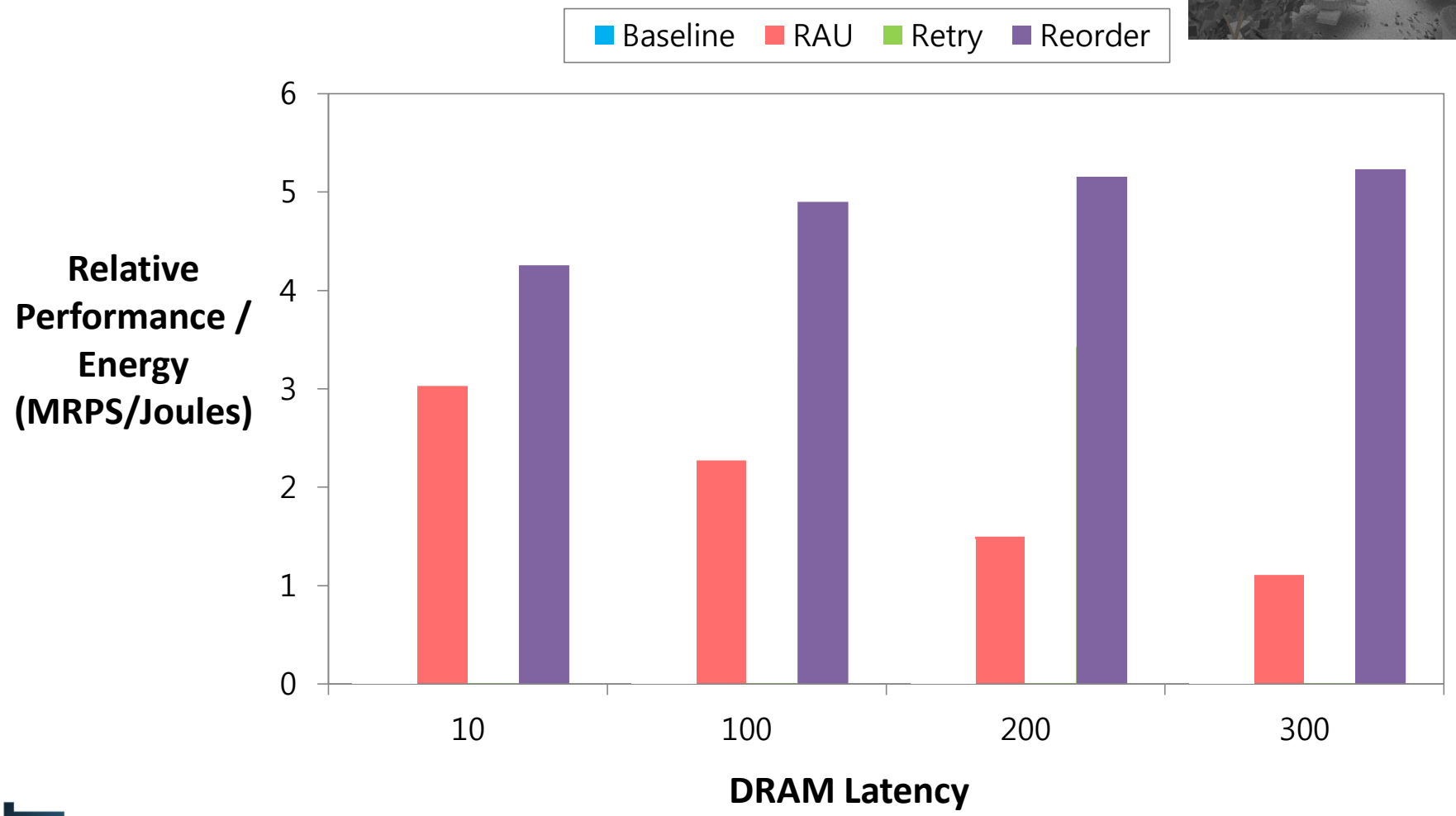




Relative Performance / Energy

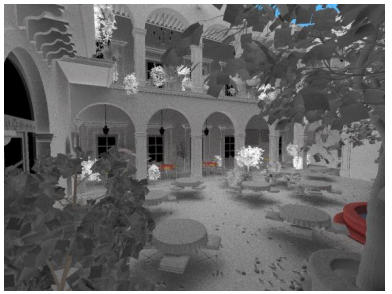
- RAU vs. Reorder…

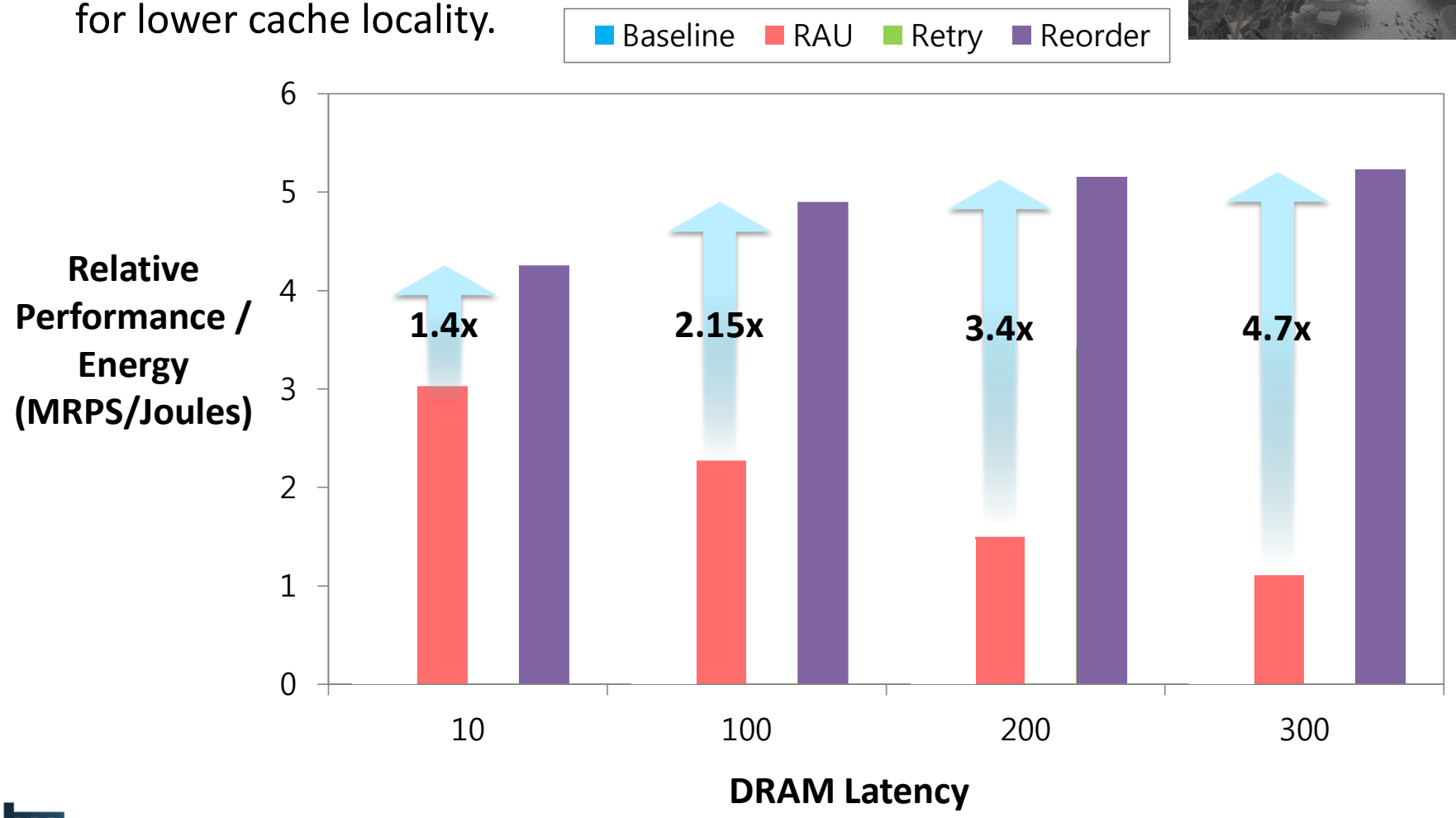




Relative Performance / Energy

- Reorder achieves up to 4.7x better efficiency.
- RAU records much lower throughputs with more pipeline stalls. Further, it consumes slightly more off-chip energy for lower cache locality.





SUMMARY

Reorder Buffer

- New hardware multithreading for MIMD ray traversal with minimal cost and energy consumption.
  (Does not need additional buffer and pipeline bypassing)

- Reschedule the order of the input rays in buffer to latency hiding and increase ray coherency (based on the cache hit/miss)

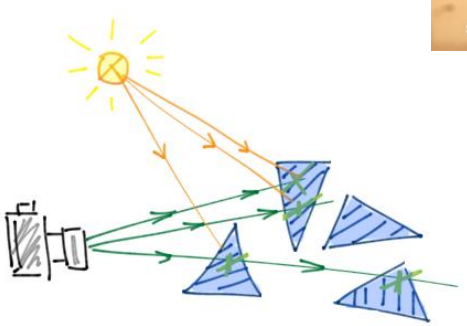
→ Up to 11.7% better cache utilization and 4.7x better efficiency.

Ray tracing and mobile

- Ray tracing provides a potential rendering technique for future mobile applications that require photorealistic graphics…









Thank you