Memory & Cache

Jinyang Li

based on Tiger Wang's slides

Question

How many memory access are needed to execute

- `movq (%rax), %rbx`
- `addq %rax, %rbx`

1 memory access.
Instruction takes 100 CPU cycles to execute

0 memory access.
Instruction takes ~1 CPU cycle to execute

How to reduce the cost of memory access?

Principle of locality

Temporal locality

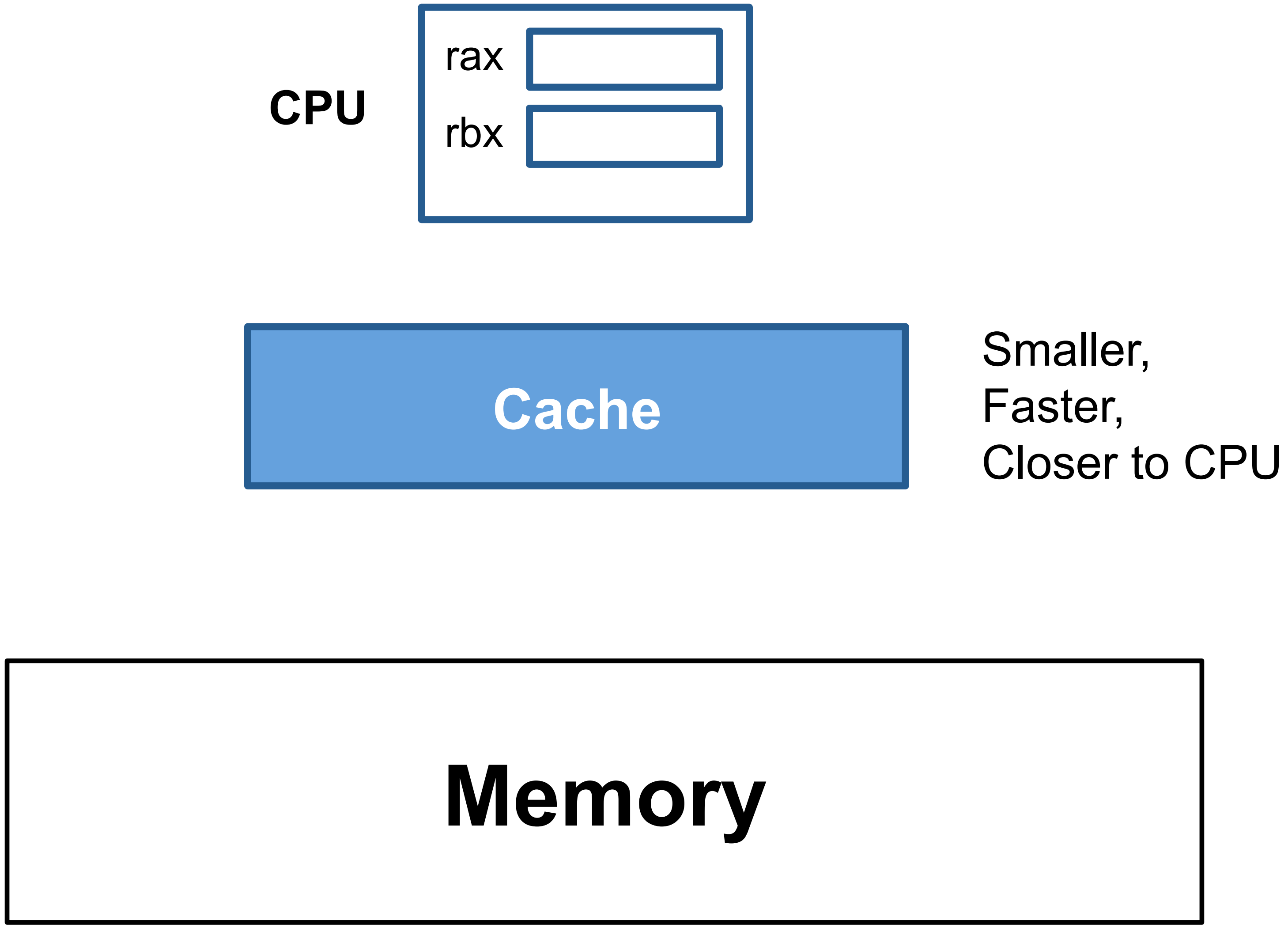
- If memory location x is referenced, then x will likely be referenced again in the near future.

Spatial locality

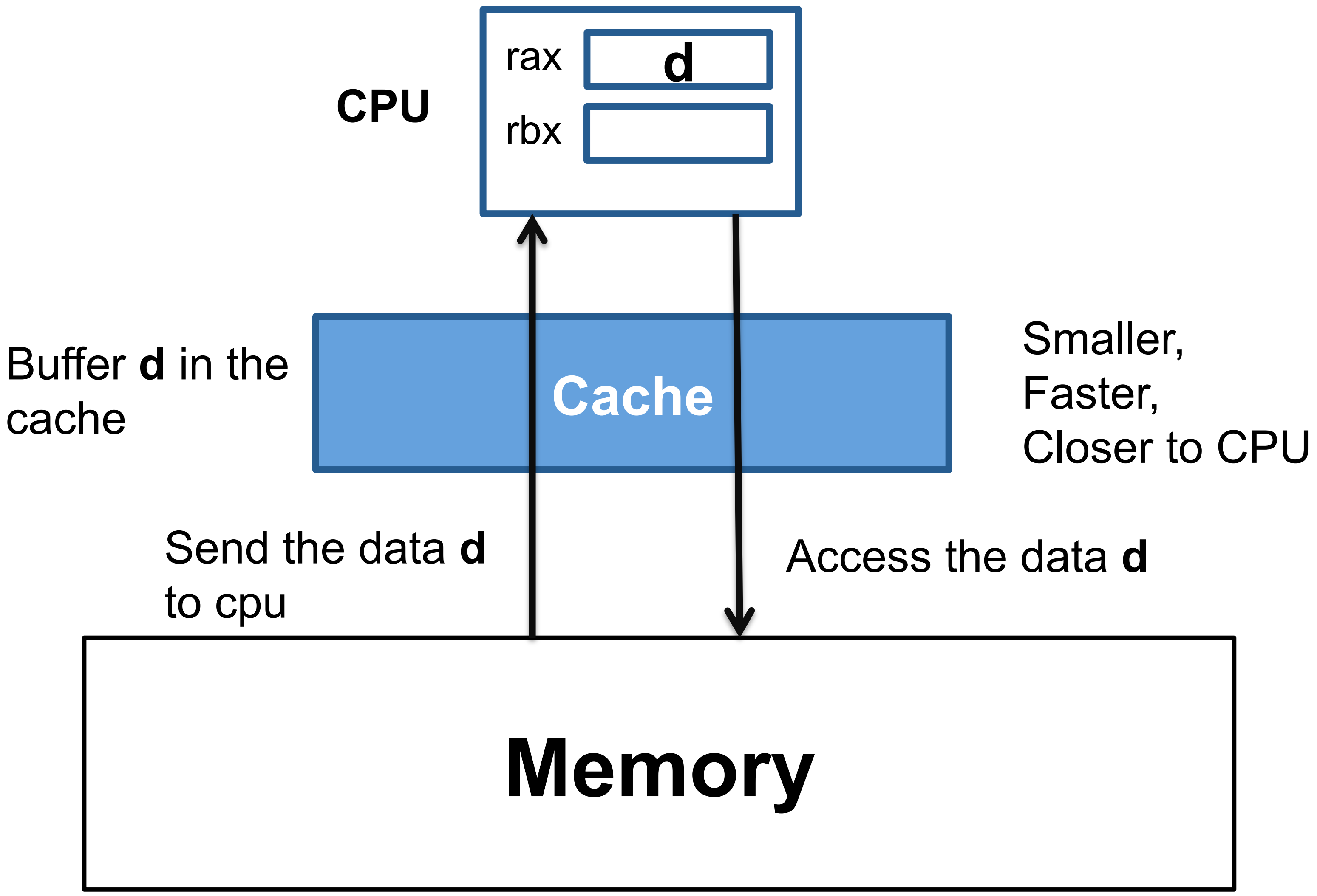
- If memory location x is referenced, then locations near x will likely be referenced in the near future.

Idea: buffer recently accessed data in cache close to CPU

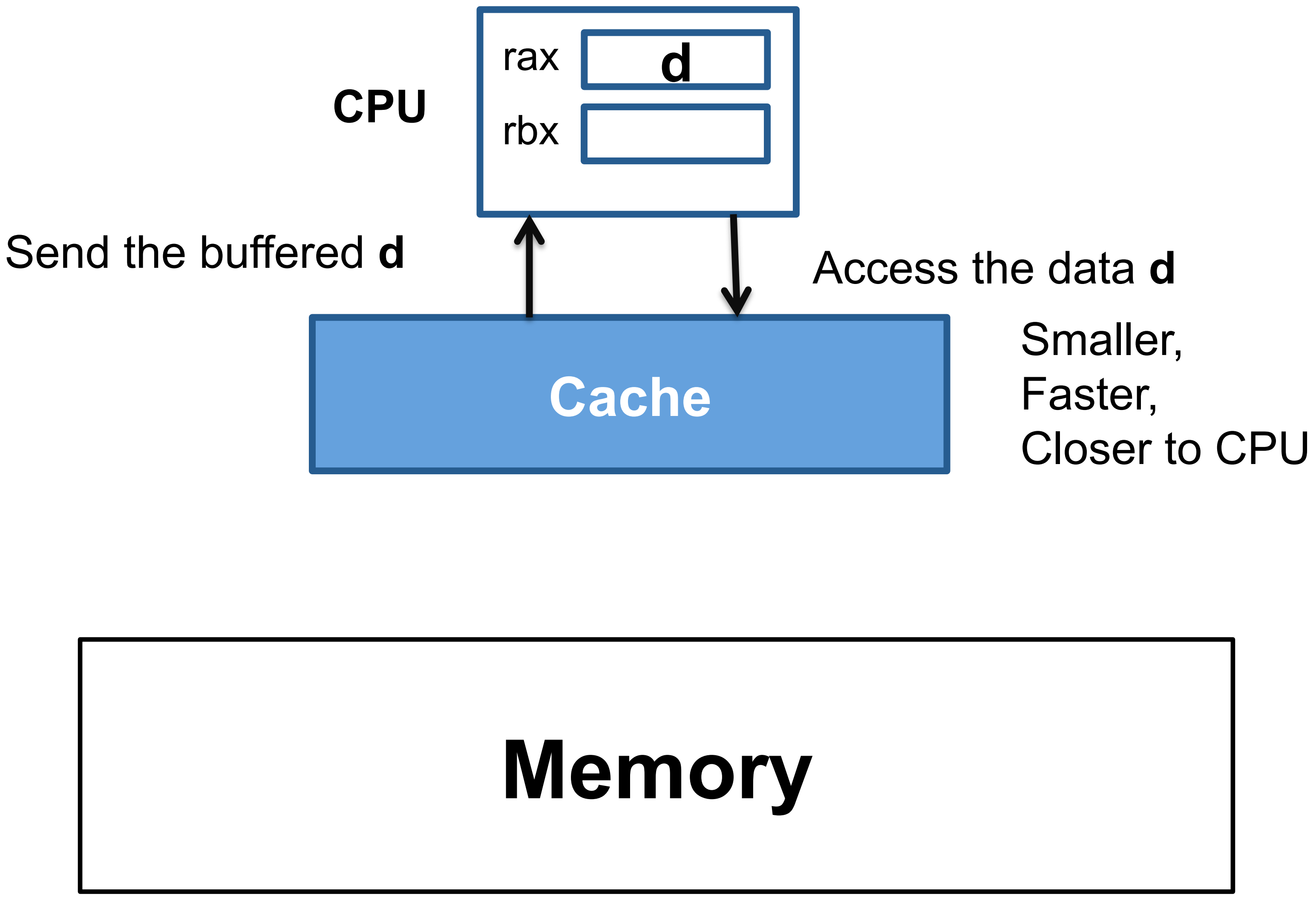
Basic idea - caching



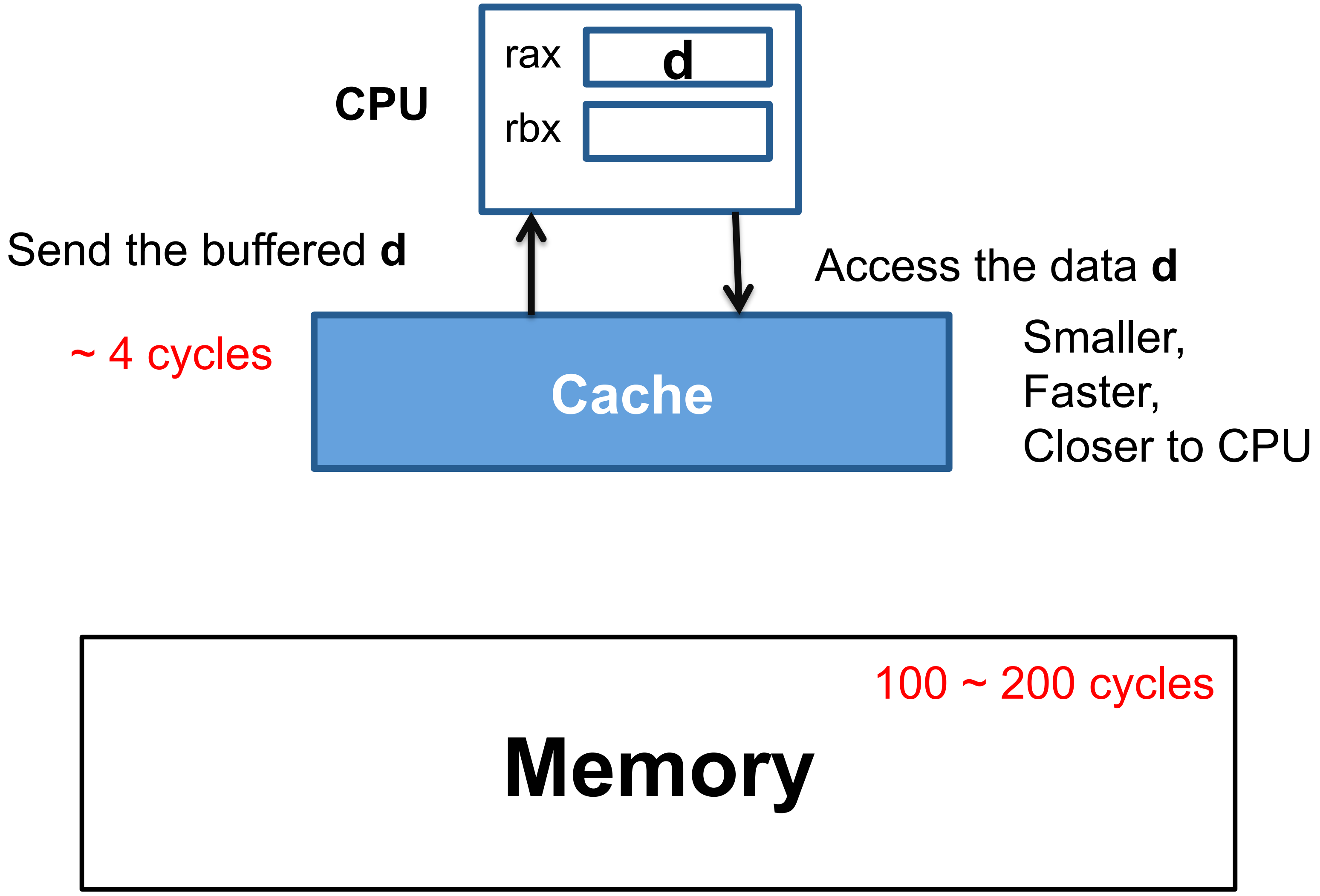
Basic idea – caching



Basic idea – caching



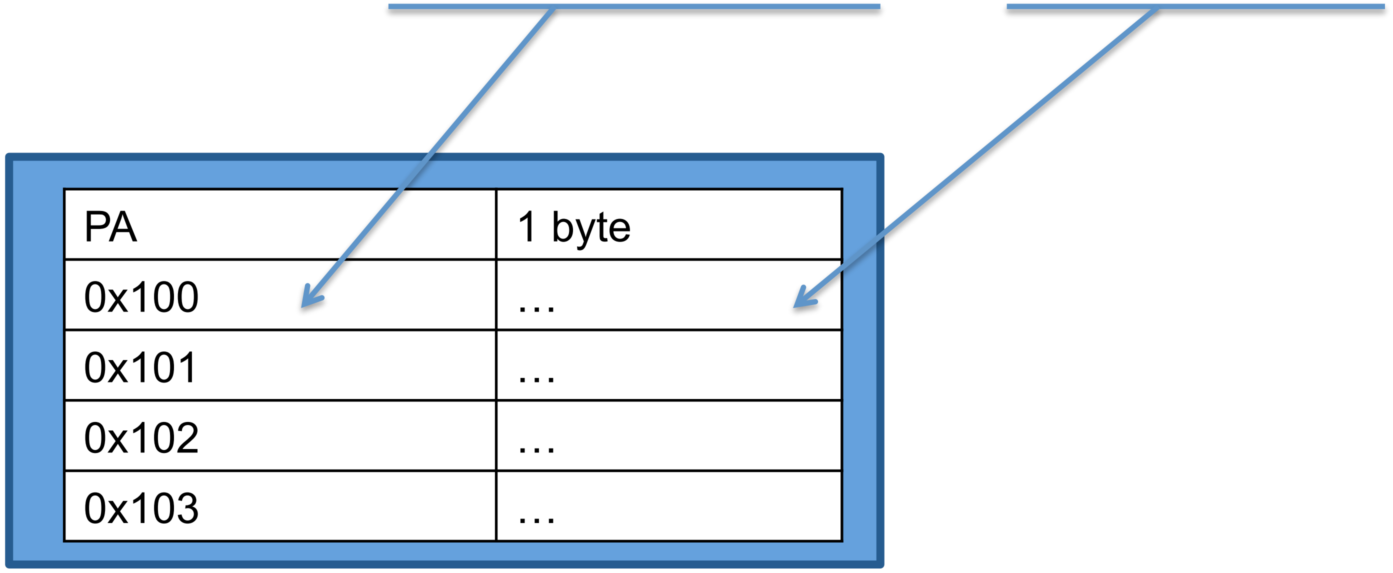
Basic idea – caching



Intuitive implementation

Caching at byte granularity:

- Search the cache for each byte accessed
  - movq (%rax), %rbx $\rightarrow$ checking 8 times
- High bookkeeping overhead
  - each cache entry has 8 bytes of address and 1 byte of data



PA	1 byte
0x100	…
0x101	…
0x102	…
0x103	…

Caching at block granularity

Solution:

- Cache one block (cacheline) at a time.
- A typical cacheline size is 64 bytes

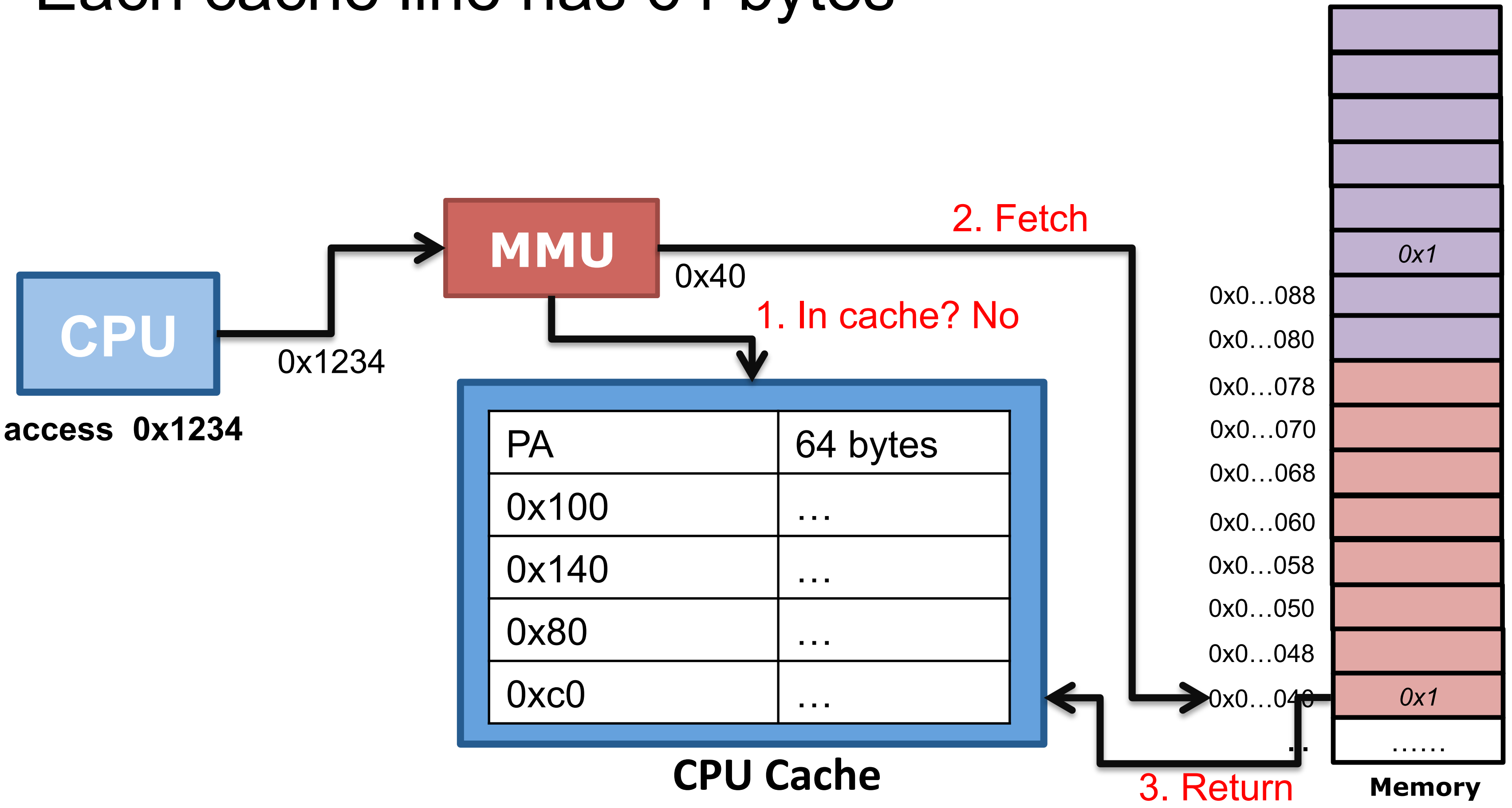
Advantage:

- Lower bookkeeping overhead
  - A cache line has 8 byte of address and 64 byte of data
- Exploits spatial locality
  - Accessing location x causes 64 bytes around x to be cached

Direct-mapped cache

Caching at block granularity

- Each cache line has 64 bytes

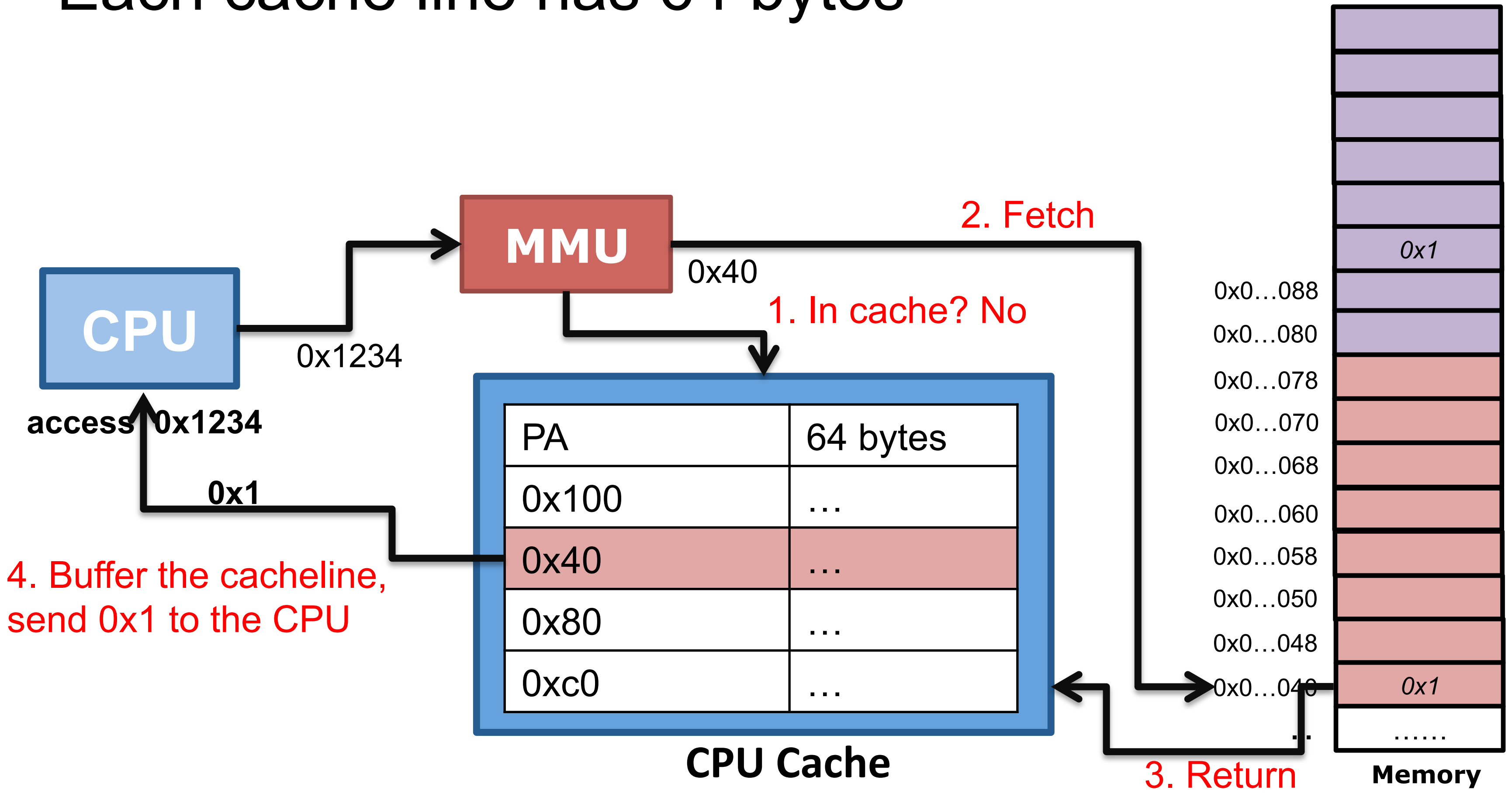


PA	64 bytes
0x100	…
0x140	…
0x80	…
0xc0	…

Direct-mapped cache

Caching at block granularity

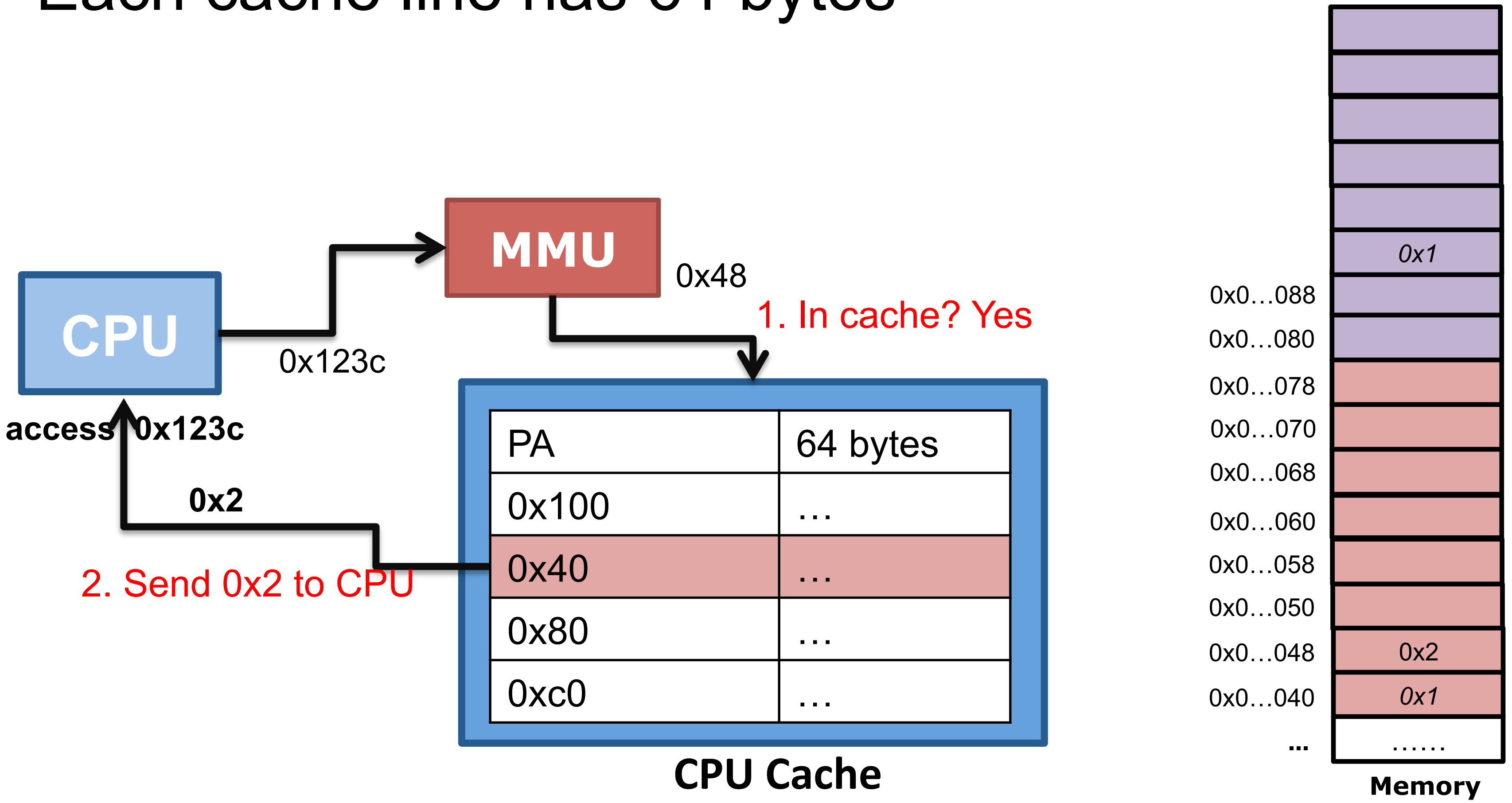
- Each cache line has 64 bytes

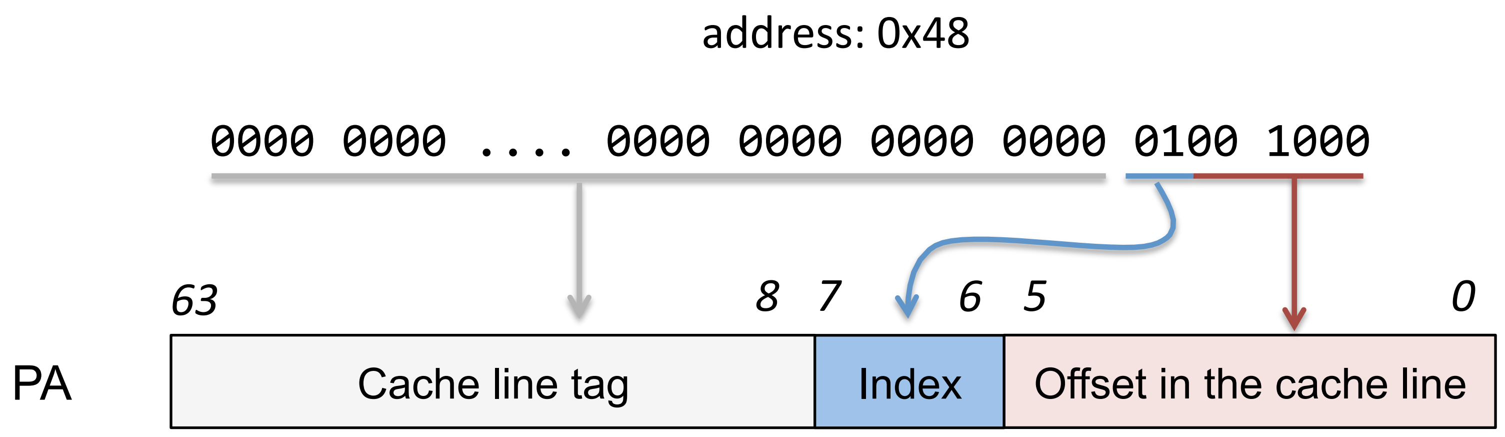


Direct-mapped cache

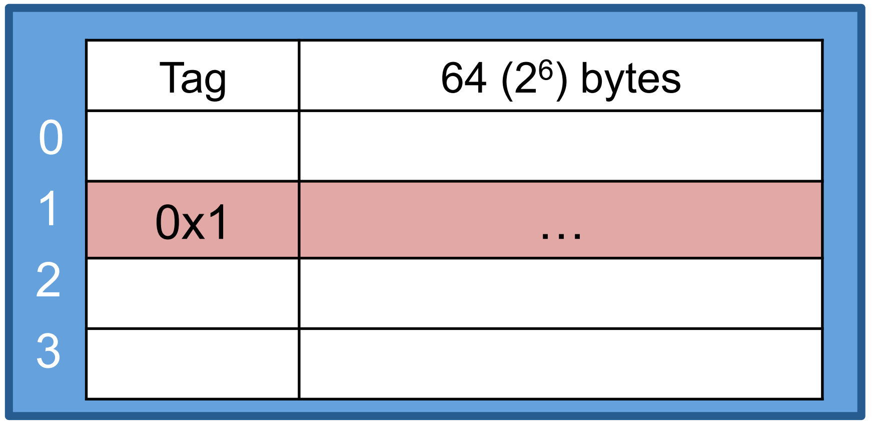
Caching at block granularity

– Each cache line has 64 bytes





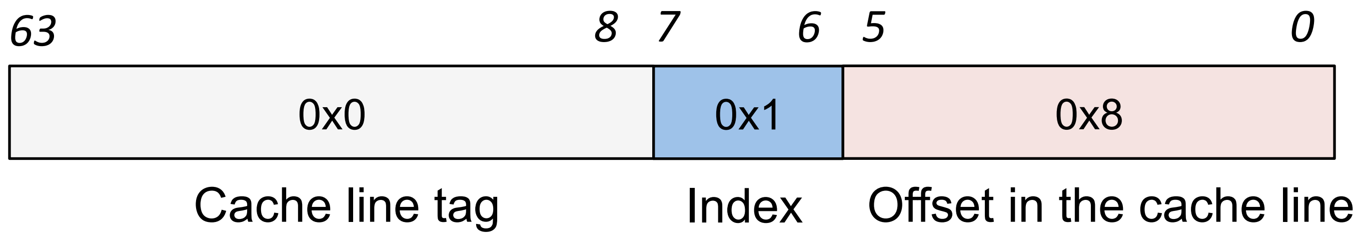
CPU access data at (PA)



	Tag	64 (2^6) bytes
0		
1	0x1	…
2		
3		

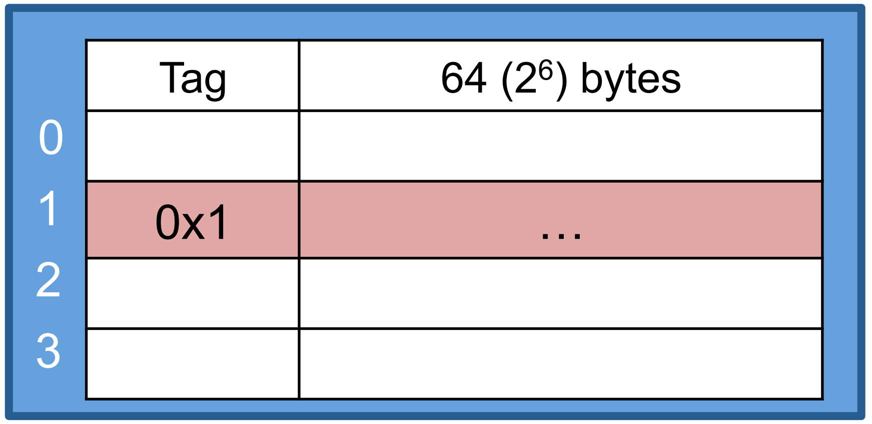
CPU Cache (64 bytes cache line)

Check and identify the location



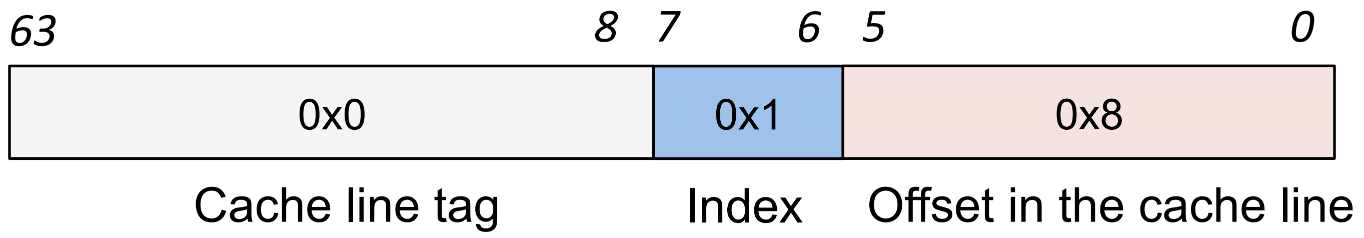
1. Use bits[6:7] to find the cache line which may buffer the data.

CPU access data at (0x48 PA)

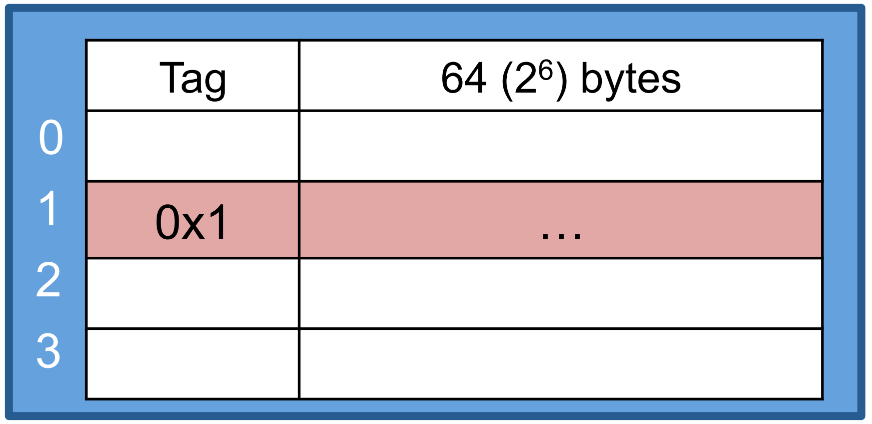


CPU Cache (64 bytes cache line)

Check and identify the location



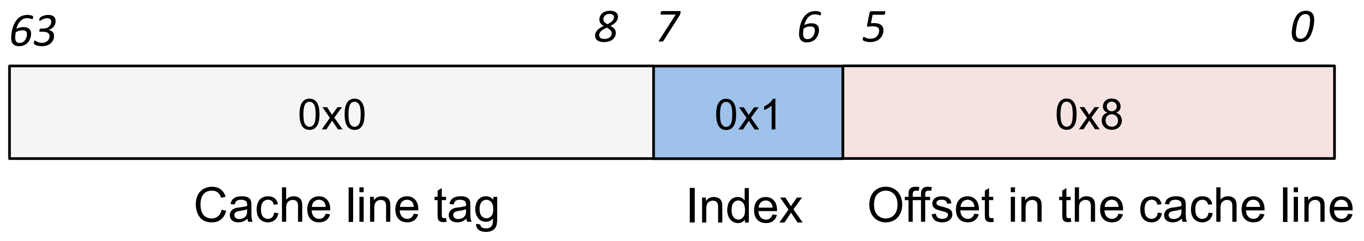
CPU access data at (0x48 PA)



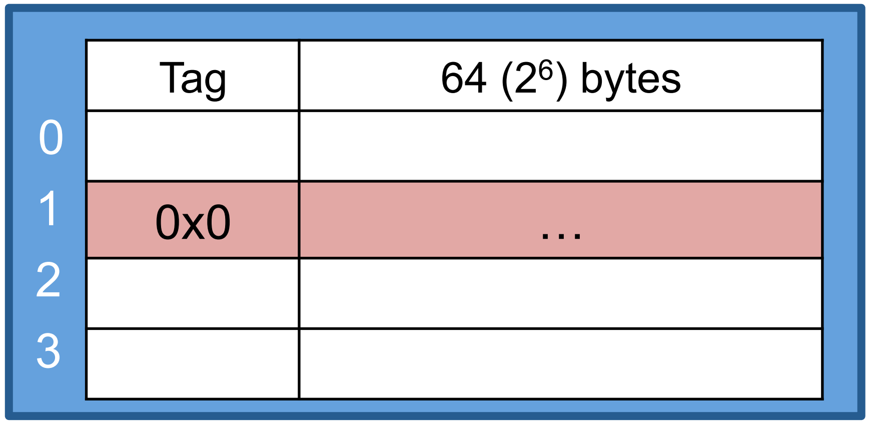
CPU Cache (64 bytes cache line)

1. Use bits[6:7] to find the cache line which may buffer the data.
2. Compare the cache line tag bits[8:63] (Cache miss)

Check and identify the location



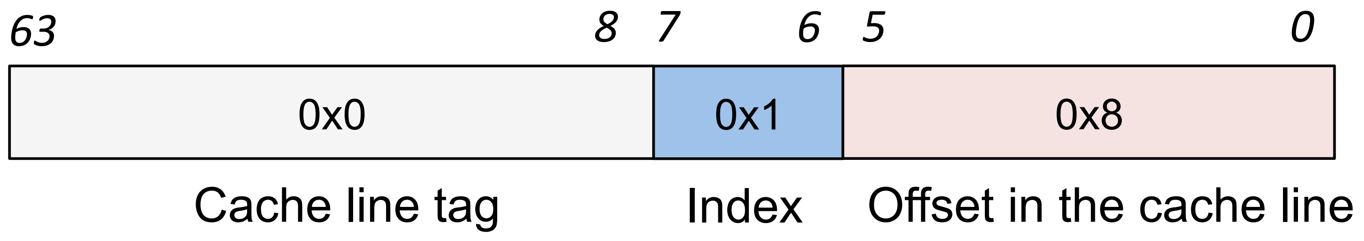
CPU access data at (0x48 PA)



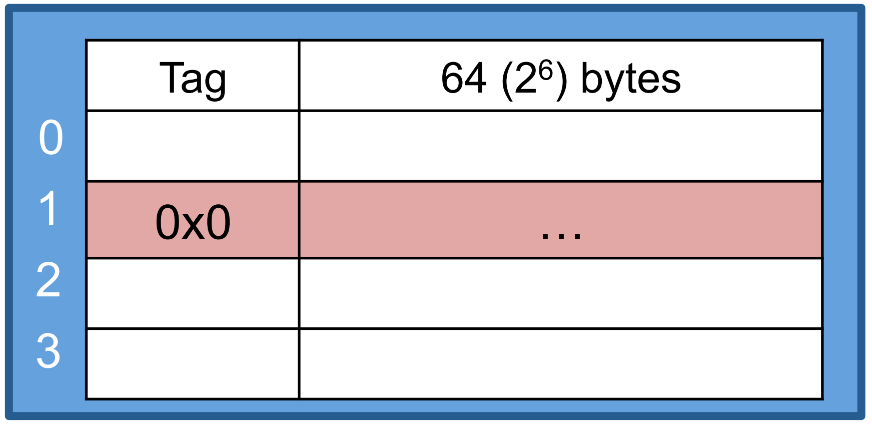
CPU Cache (64 bytes cache line)

1. Use bits[6:7] to find the cache line which may buffer the data.
2. Compare the cache line tag bits[8:63] (Cache miss)
3. Load 64 bytes from 0x40

Check and identify the location



CPU access data at (0x48 PA)



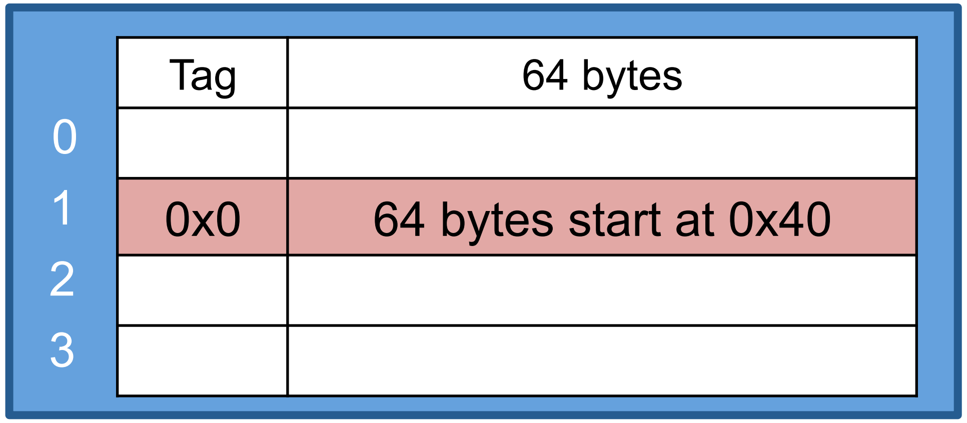
CPU Cache (64 bytes cache line)

1. Use bits[6:7] to find the cache line which may buffer the data.
2. Compare the cache line tag bits[8:63] (Cache miss)
3. Load 64 bytes from 0x40
4. Send the data at the offset of 0x8 in buffered cache line.

Issue I

Access pattern:

→ access **0x40** **cache miss**
access **0x140**
access **0x40**
access **0x140**



	Tag	64 bytes
0		
1	0x0	64 bytes start at 0x40
2		
3		

CPU Cache

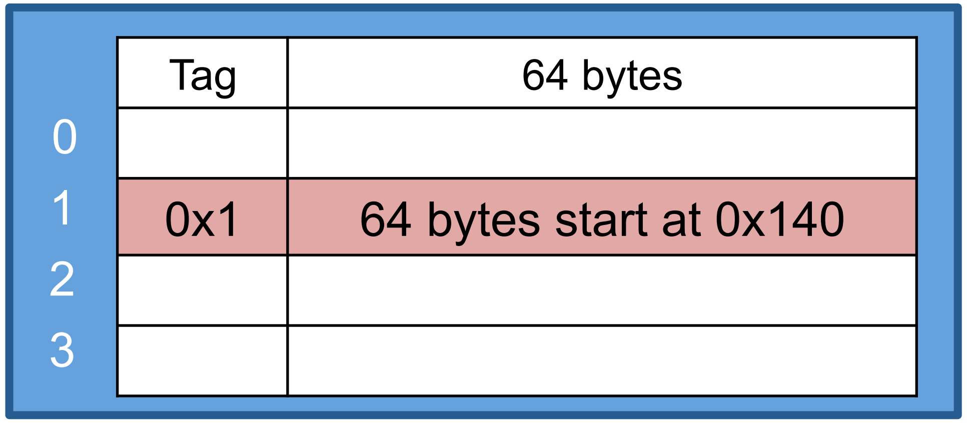
Cache line tag: 0
Cache line index: 1

Load 64 bytes start at 0x40 into 1st entry

Issue I

Access pattern:

access **0x40** **cache miss**
→ access **0x140** **cache miss**
access **0x40**
access **0x140**



	Tag	64 bytes
0		
1	0x1	64 bytes start at 0x140
2		
3		

CPU Cache

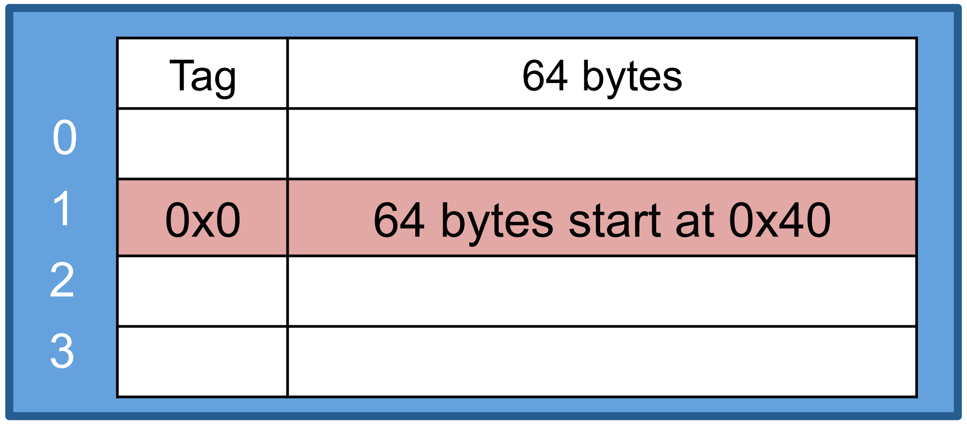
Cache line tag: 1
Cache line index: 1

Evict old cache line (1st entry),
Load 64 bytes start at 0x140 into 1st entry

Issue I

Access pattern:

access **0x40**	**cache miss**
access **0x140**	**cache miss**
→ access **0x40**	**cache miss**
access **0x140**	



	Tag	64 bytes
0		
1	0x0	64 bytes start at 0x40
2		
3		

CPU Cache

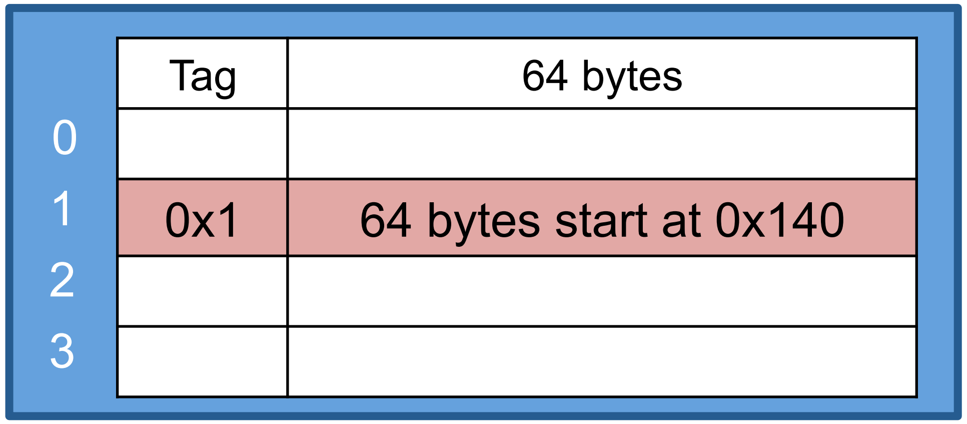
Cache line tag: 0
Cache line index: 1

Evict old cache line (1st entry),
Load 64 bytes start at 0x40 into 1st entry

Issue I

Access pattern:

access **0x40** cache miss
access **0x140** cache miss
access **0x40** cache miss
→ access **0x140** cache miss



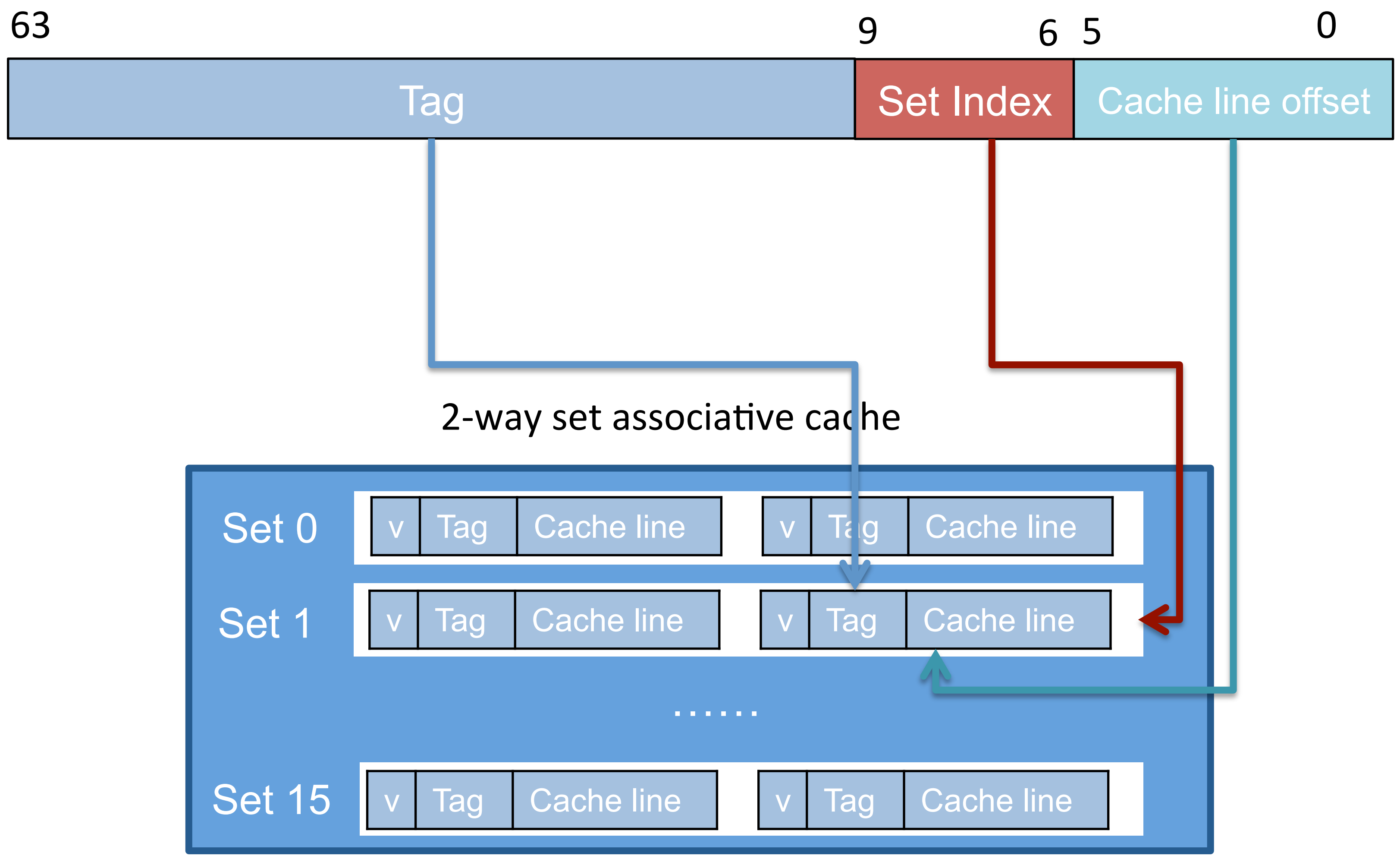
	Tag	64 bytes
0		
1	0x1	64 bytes start at 0x140
2		
3		

CPU Cache

Cache line tag: 1
Cache line index: 1

Evict old cache line (1st entry),
Load 64 bytes start at 0x140 into 1st entry

Multi-way set associative cache



CPU Cache

Multi-way set associative cache

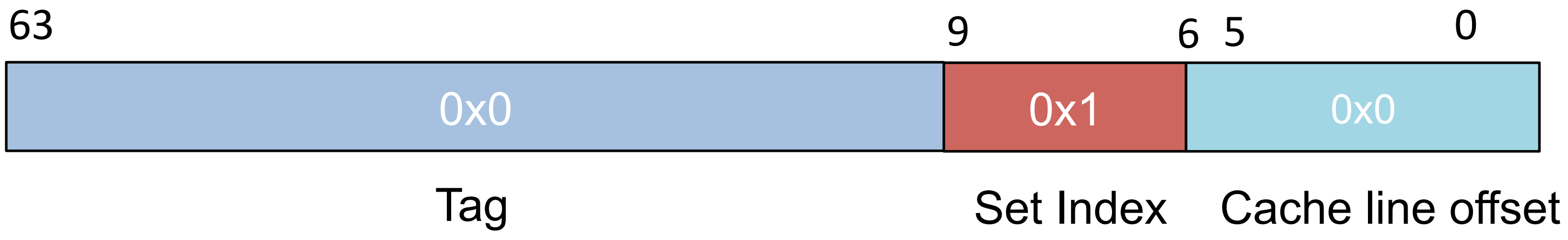
Access pattern:

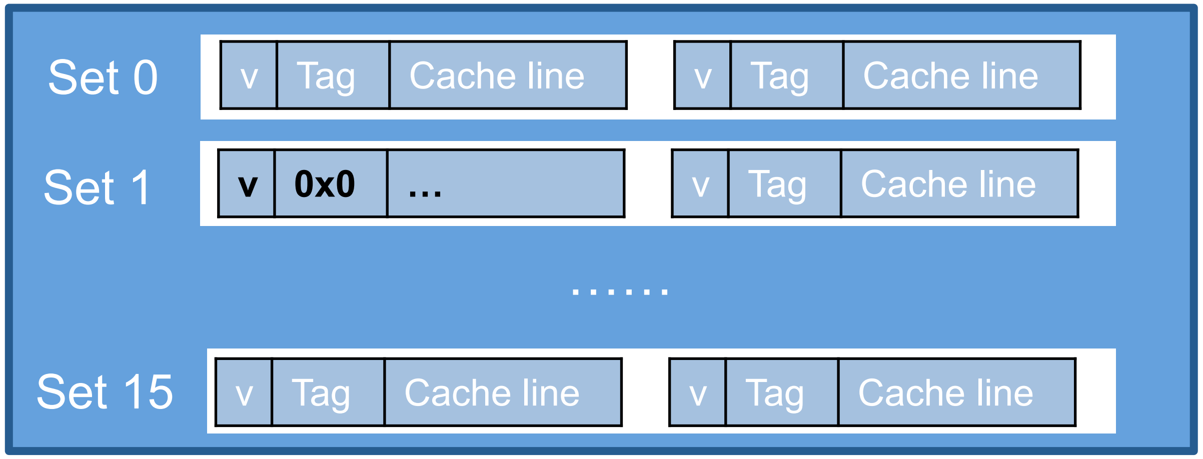
→ access **0x40** **miss**

access **0x440**

access **0x40**

access **0x440**



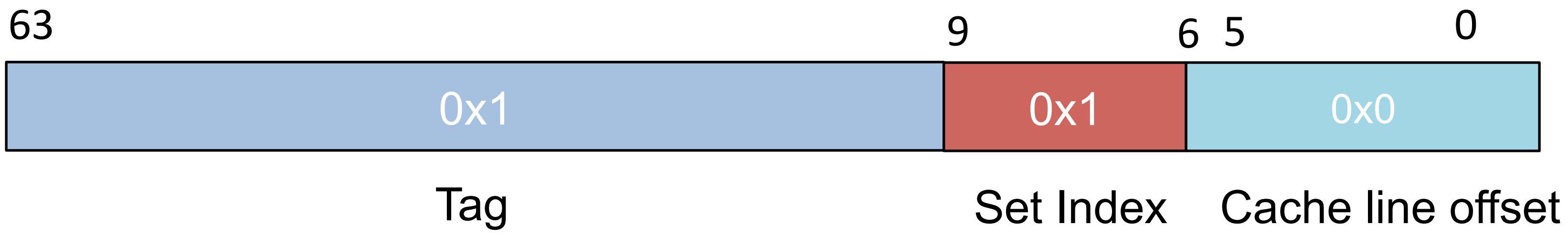


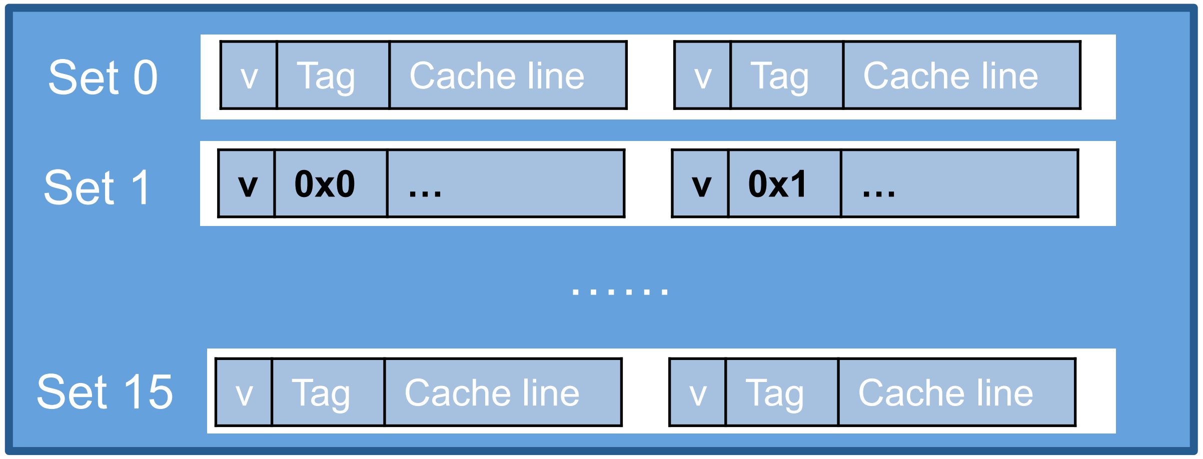
CPU Cache

Multi-way set associative cache

Access pattern:

access **0x40** **miss**
→ access **0x440** **miss**
access **0x40**
access **0x440**



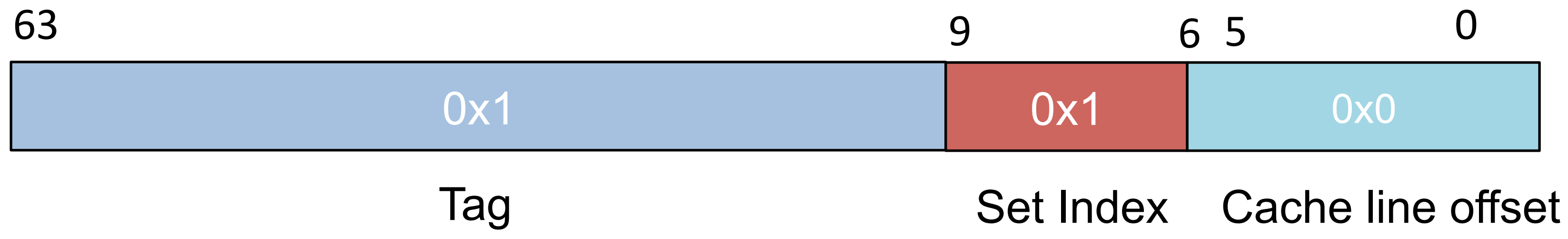


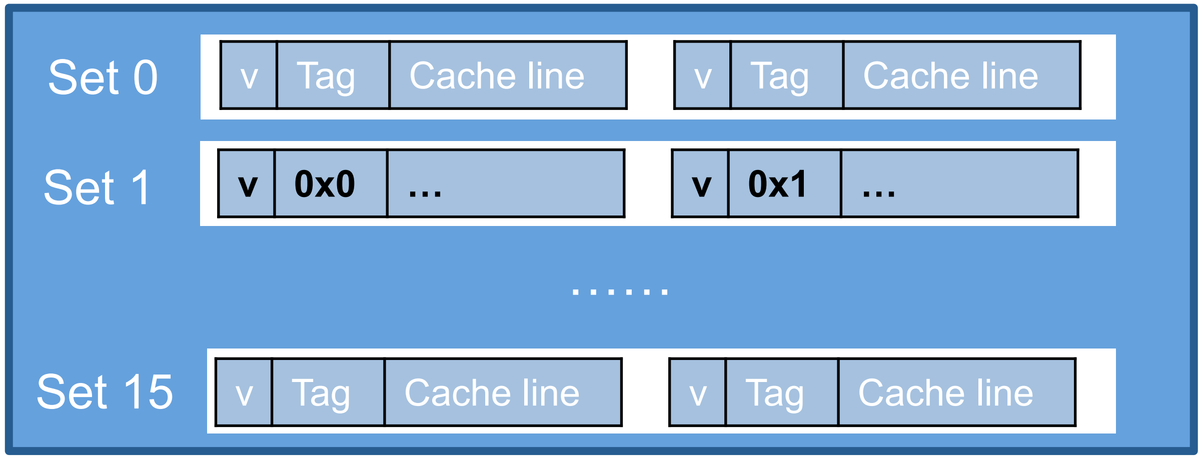
CPU Cache

Multi-way set associative cache

Access pattern:

access **0x40** **miss**
access **0x440** **miss**
access **0x40** **hit**
access **0x440** **hit**





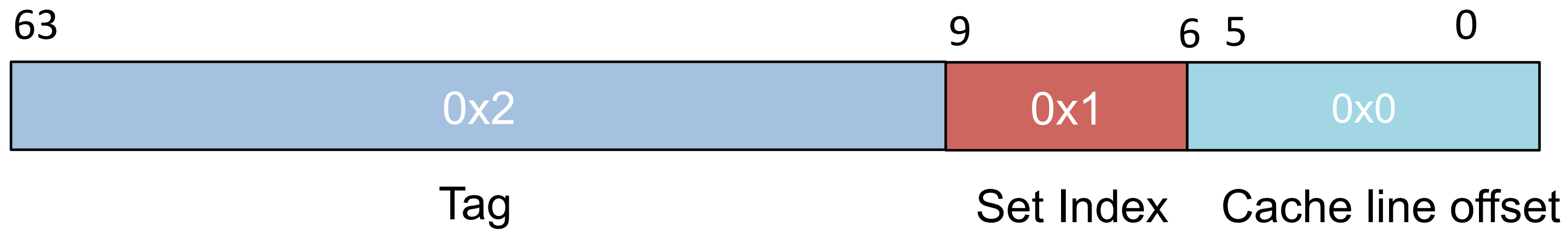
CPU Cache

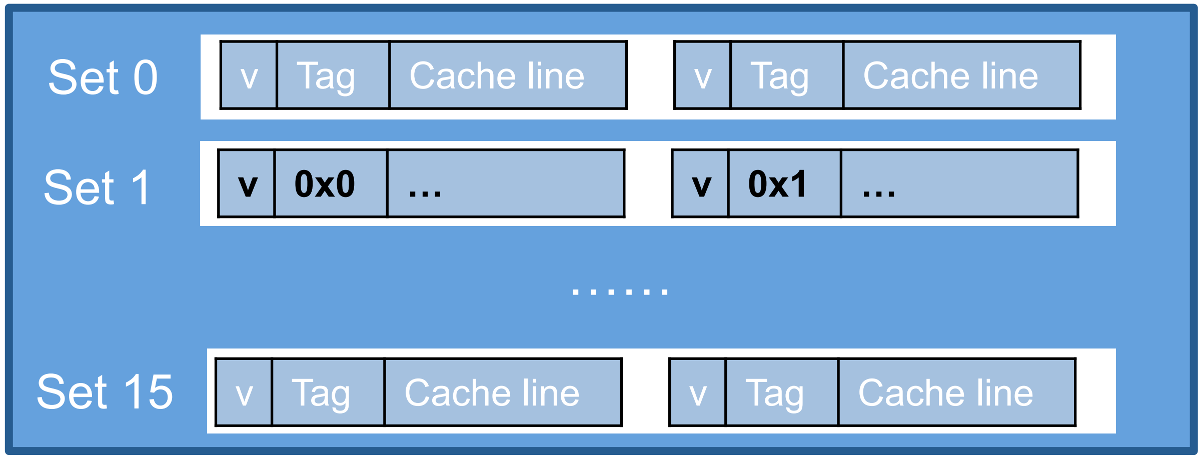
Multi-way set associative cache

Access pattern:

access **0x40**
access **0x440**
→ access **0x840**

Which cache line in set 1 should be evicted?





CPU Cache

Cache line replacement policy

LFU (least-frequently-used)

– Replace the line that has been referenced the fewest times over some past time window

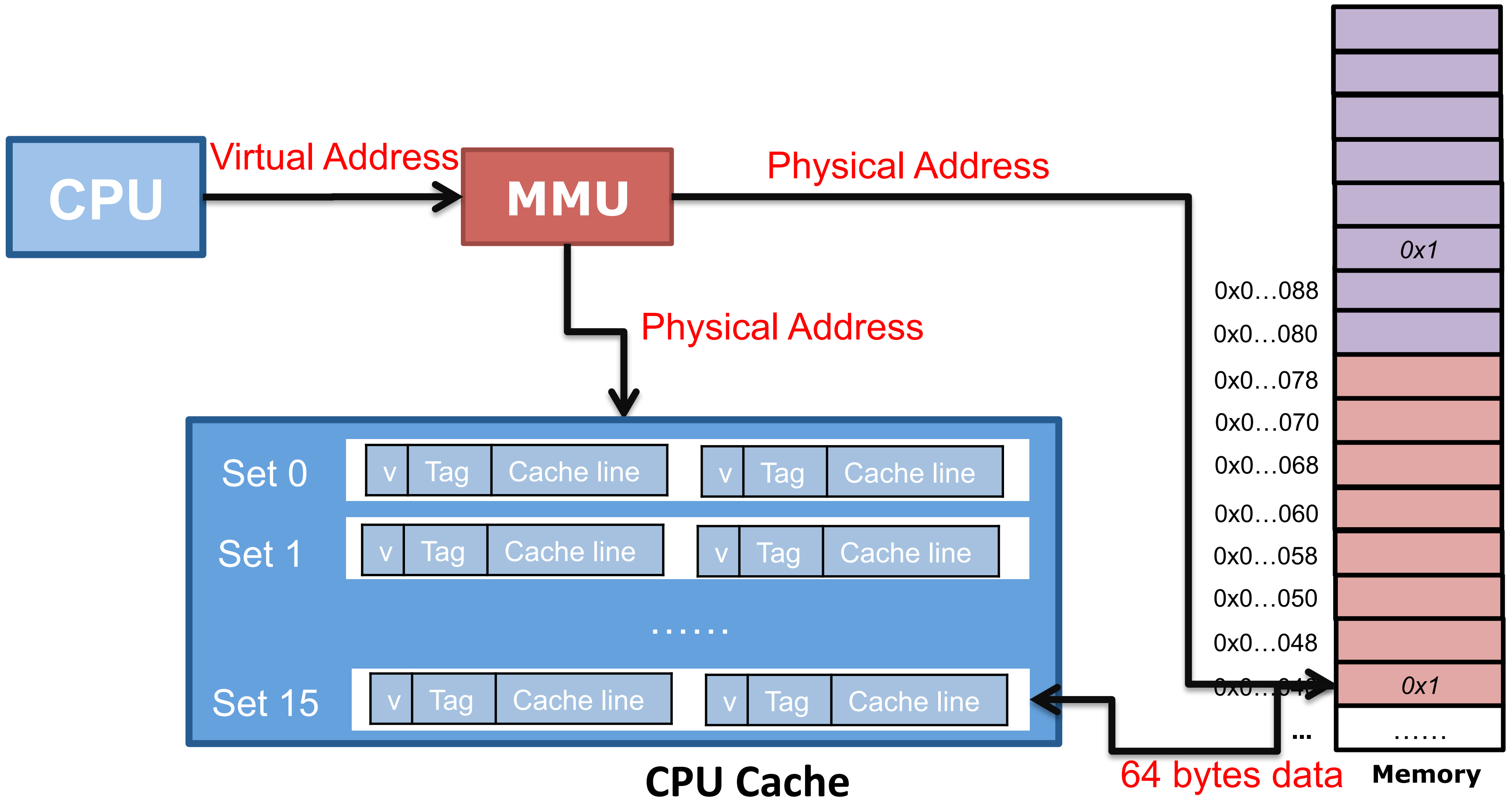
LRU (least-recently-used)

– Replace the line that has the furthest access in the past

These policies require additional time and hardware

Issue II

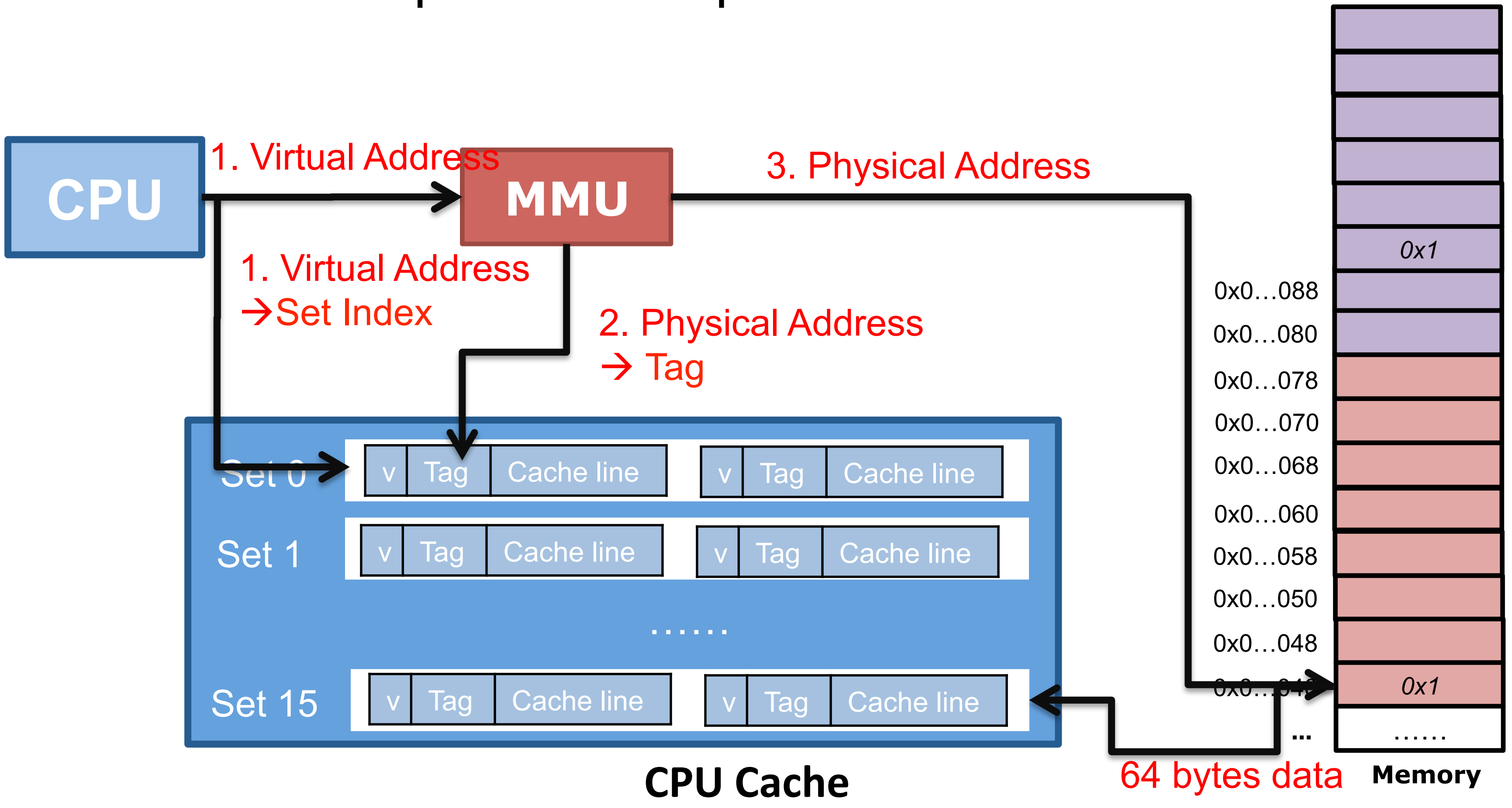
- Current design: address translation → cache access
  - The two steps are performed sequentially



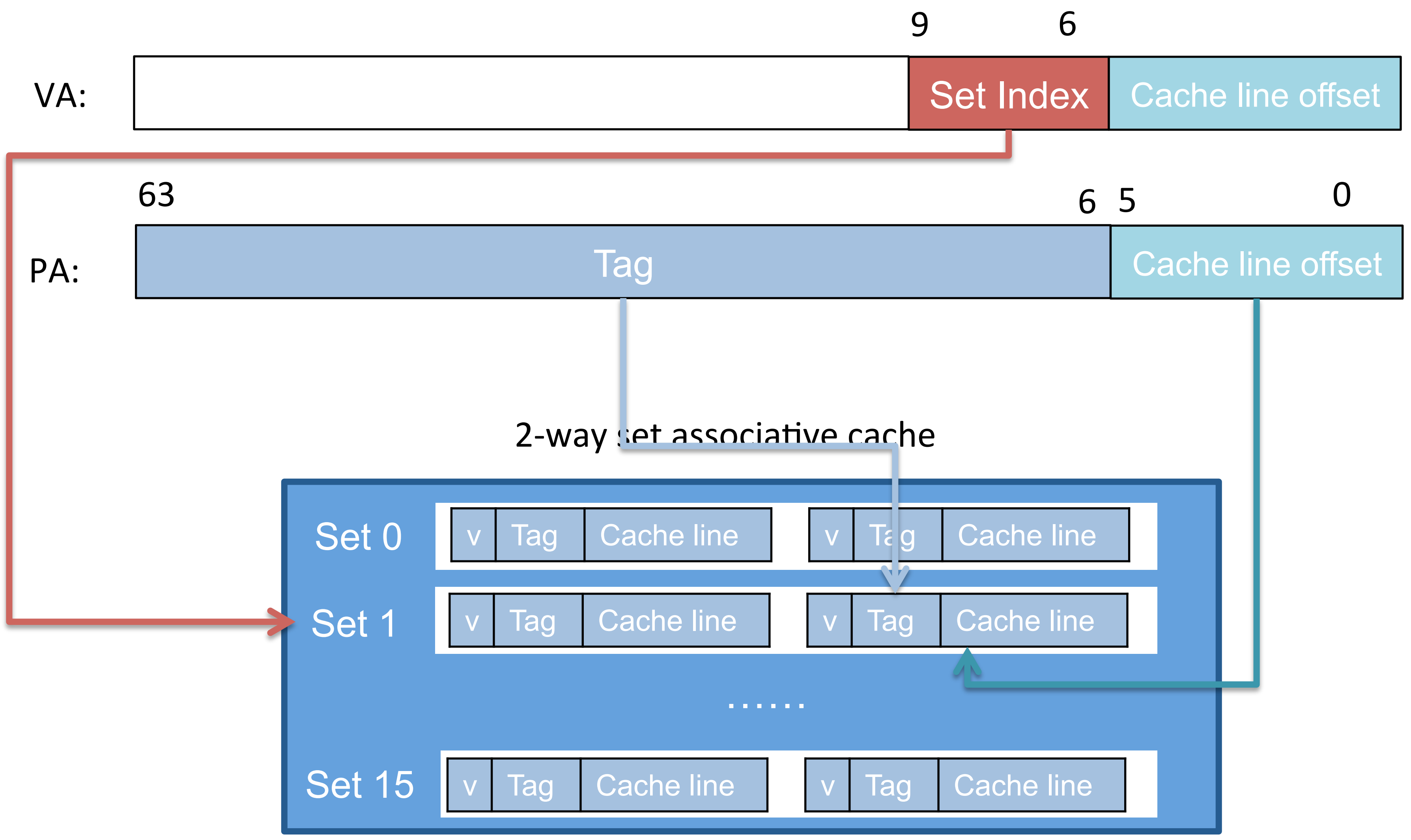
How to parallelize address translation & cache access?

Virtual Index and Physical Tag

- Use VA to index set, calculate the tag from PA
- Cache set lookup is done in parallel with address translation



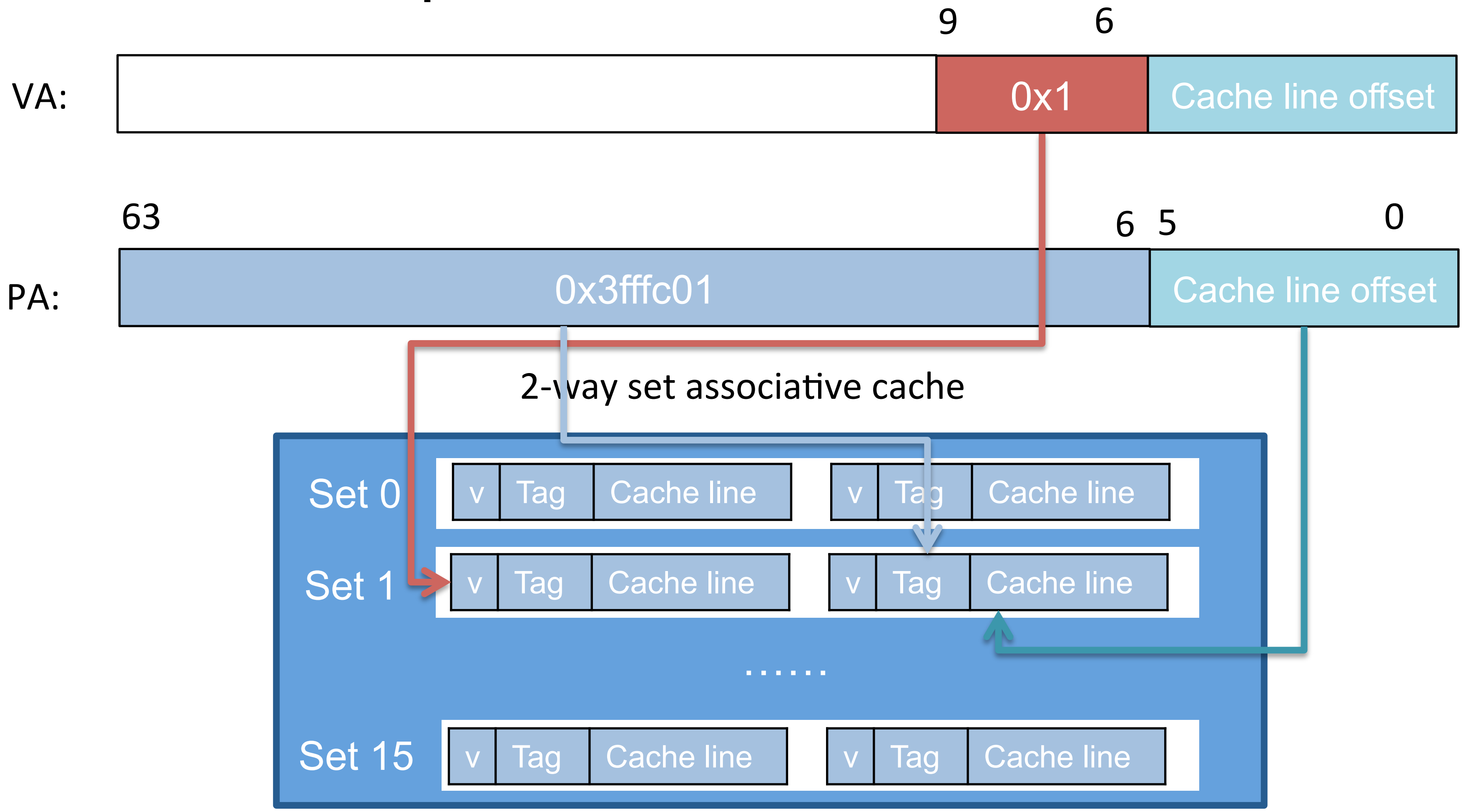
Virtual Index and Physical Tag



CPU Cache

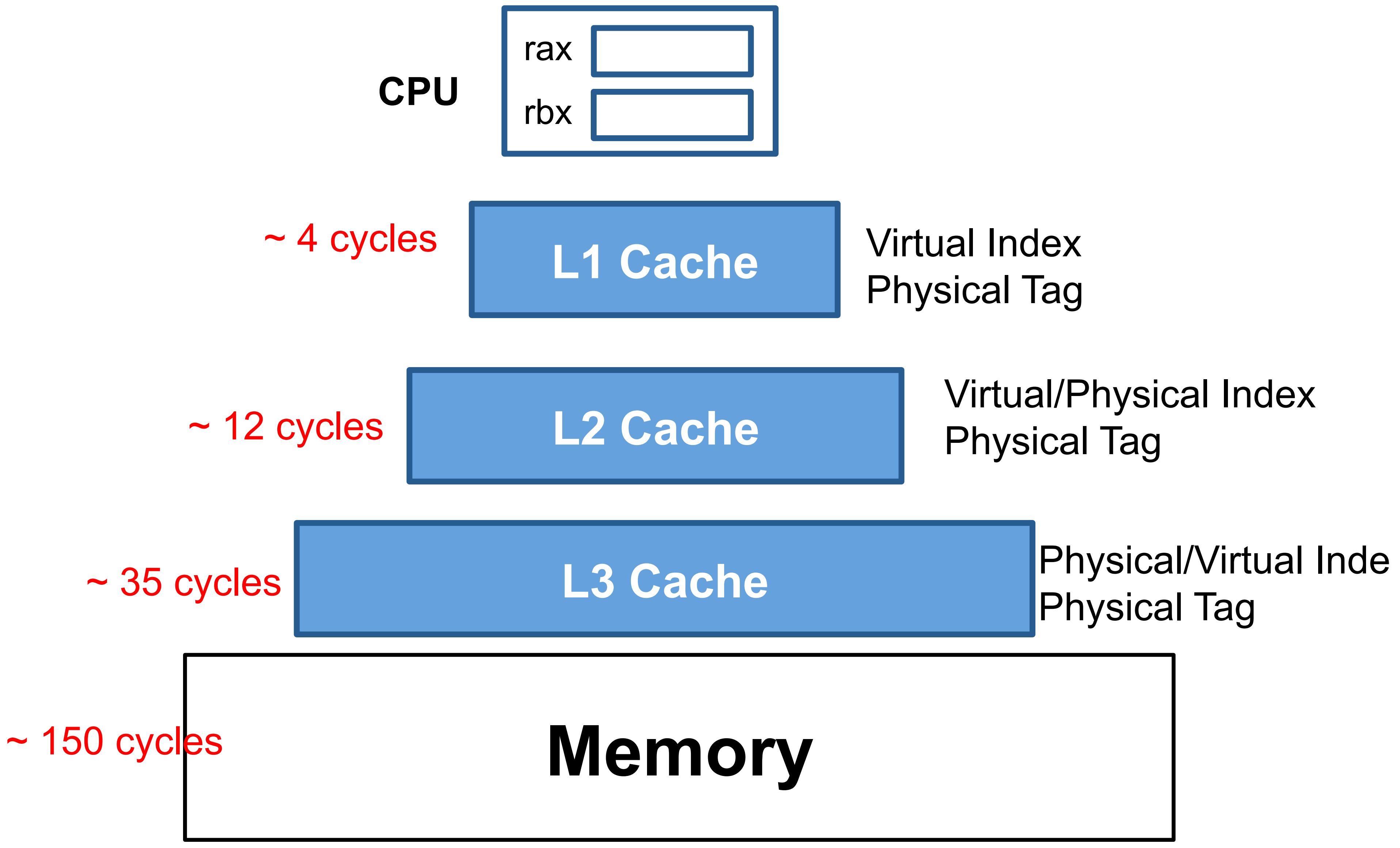
Virtual Index and Physical Tag

access **va 0x1040 pa 0xffff0040**



CPU Cache

Memory hierarchy



Cache summary

- Caching can speed up memory access
- L1/L2/L3 cache data
- TLB cache address translation
- Cache design:
  - Direct mapping
  - Multi-way set associative
  - Virtual index + physical tag parallelize cache access and address translation