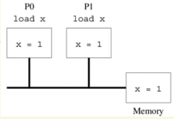
Cache Coherence

Cache 일관성

Shared address space

- modern processors replicate memory in local cache



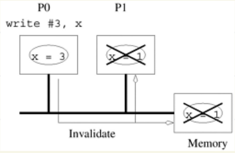
⇒ need to coordinate such shared data (local copy)

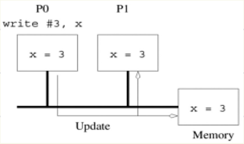
- multiple copies can easily become inconsistent

  ⇒ due to write on local copy

- to guarantee consistency, one need to coordinate it.

  ⇒ 즉, reading at value == last value written by any processor





(다양한 policy 존재)

sequential consistency

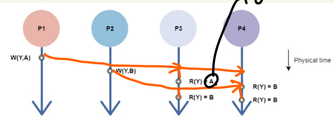
: 이벤트 간의 인과관계만 생각하는 것

- All data access = executed atomically, in some sequential order
- Consistency를 위해서 임의로 sequential order를 설정할 수 있다

  last write = read

- 예

physical timeline에 위배됨



예시 sequentially consistent 하려면

W(Y,A) → R(Y) = A → W(Y,B) → R(Y) = B

즉, each variable must have only single value at a time

write 일어나고 나면 이전 값은 가질 수 없음 !

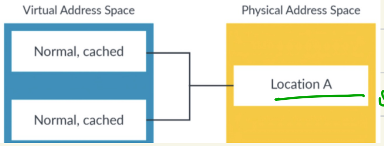
Approaches to Cache Coherence

Hardware : cache Implements coherence protocols to ensure global consistency

Software

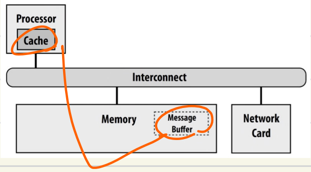
- compiler / runtime support
- must be conservative (경계, 보수적) to be safe
  - memory aliasing 주의

memory aliasing : one physical address space has multiple virtual addresses



shared memory 주의

on single CPU : I/O device with DMA data transfer



Copy!

① when DMA write

- DMA write 하는동안 Memory는 cache data 이용.
- If write on cache, flush back to Memory
  ⋮ stated data (old data)
  old data로 Main Memory overwritten 발생

② when cache write

- DMA transfer from buffer to cache는 stale data transfer하고 끝

Solution

① CPU writes using uncached stores

② OS makes pages containing shared buffer as uncached

즉, DMA transfer 포함시만 write 하도록!

DMA가 그렇게 frequent 하진 않아서 bottleneck은 별로

③ explicit flush / Invalidate before DMA transfer