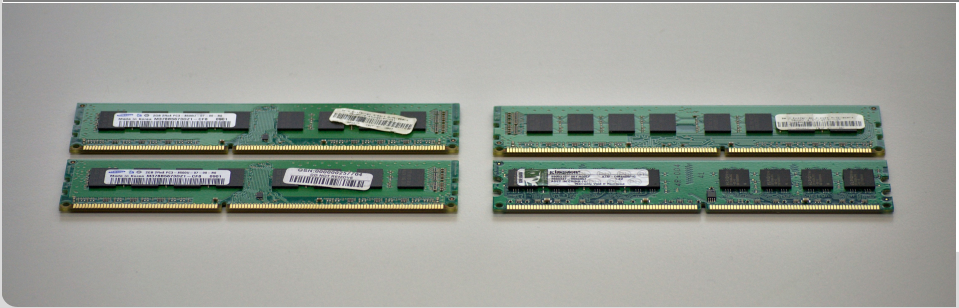
Karlsruhe Institute of Technology

Operating Systems

10. CPU Caches

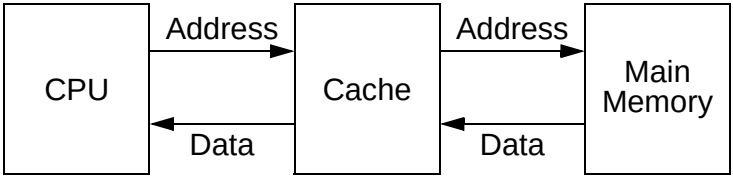
Prof. Dr. Frank Bellosa | WT 2020/2021



Managing Memory

- Ideal memory properties
  - Large
  - Fast
  - Nonvolatile
  - Cheap
- Real memory: trade off properties above
  - Tape/Disk = large + slow + cheap + nonvolatile
  - SSD = not too large + not too cheap + nonvolatile + low endurance
  - PCRAM, Optane® = fast (Read) + slow (write) + nonvolatile
  - DRAM, SRAM = fast + volatile + expensive
  - Registers = very fast + volatile + very expensive
  - ...

CPU-Cache Location [Sch99]



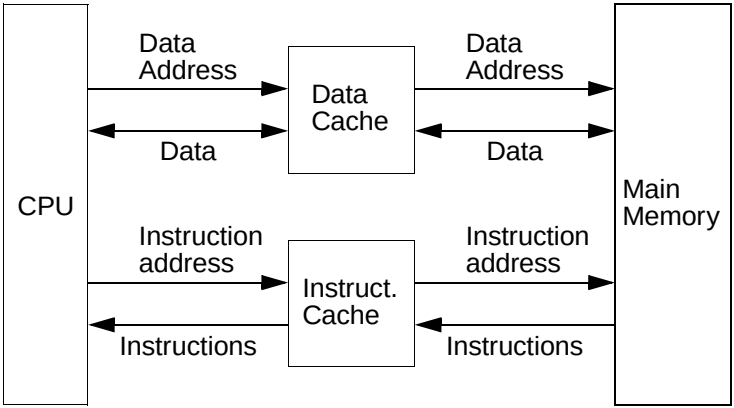
- Buffer memory for exploiting temporal and spatial locality
- Low latency, high bandwidth
- Reduction of main memory accesses
- Reduction of memory bus traffic (important for multiprocessor systems)
- Buffer for asynchronous prefetch operations

Types of Cache Misses

- Compulsory miss
  - Cold start, first reference
  - Data block was not cached before
- Capacity miss
  - Not all required data fits into the cache
  - Accessed data was previously evicted to make room for different data
- Conflict miss
  - Collision, interference
  - Depending on the cache organization, data items interfere with each other
  - Fully associative caches are not prone to conflict misses

Harvard Architecture[Sch99]

- Separate buffer memory for data and instructions



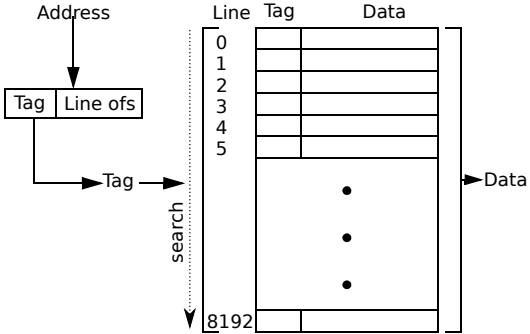
Write and Replacement Policies

- Cache hit
  - Write-through
    - Main memory is always up-to-date
    - Writes may be slow
  - Write-back
    - Data is written only to the cache
    - Main memory temporarily inconsistent

- Cache miss
  - Write-allocate:
    - To-be-written data item is read from main memory into the cache
    - Write performed afterwards according to the write policy
  - Write-to-memory:
    - Modification is performed only in main memory

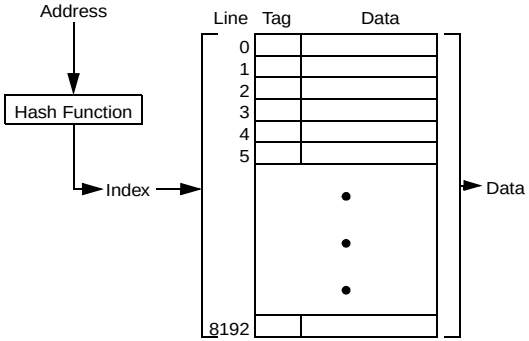
Cache Organization: Fully Associative Cache[Sch99]

- Cache-lines with fixed length (e.g., 32/64 Bytes)
- Data identified by tag-field
- Memory access: searching all tags for address tag



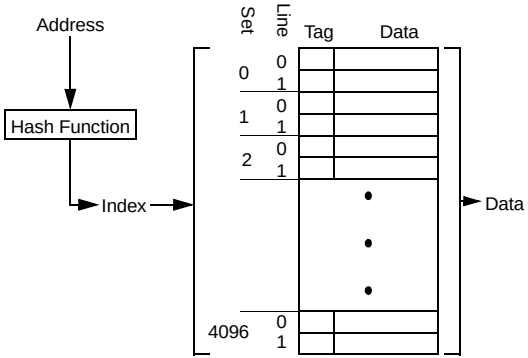
Cache Organization: Direct Mapped Cache[Sch99]

- Cache-lines with fixed length (e.g., 32/64 Bytes)
- Mapping from address to cache lines
- Data identified by tag-field



Cache Organization: Set Associative Cache[Sch99]

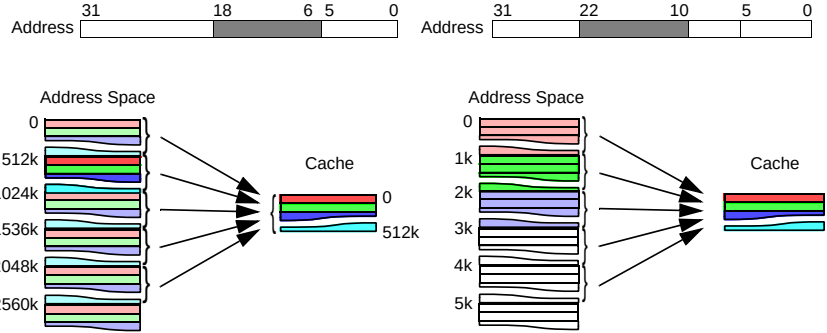
Example: 2-way set associative cache



- If #sets == #lines → Direct Mapped Cache
- If #sets == 1 → Fully Associative Cache

Hash Functions[Sch99]

- Modulo hashing: e.g., bits 0 to 5 as byte-offset within a cache line, bits 6 to 18 to select a cache line, bits 19 to 31 as tag.
- Arbitrary cut-out of address to select cache line, e.g. bits 10 to 22. Suboptimal mapping of addresses to cache lines and therefore not used.

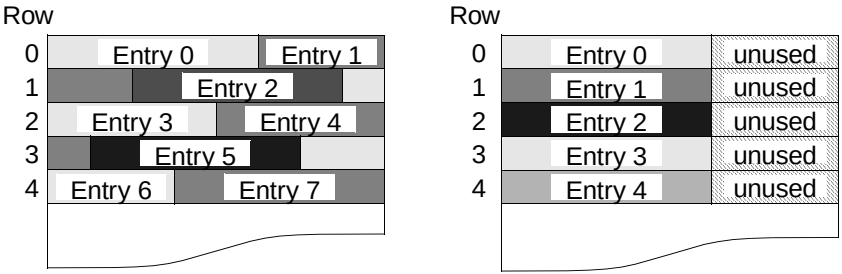


Cache Design Parameters

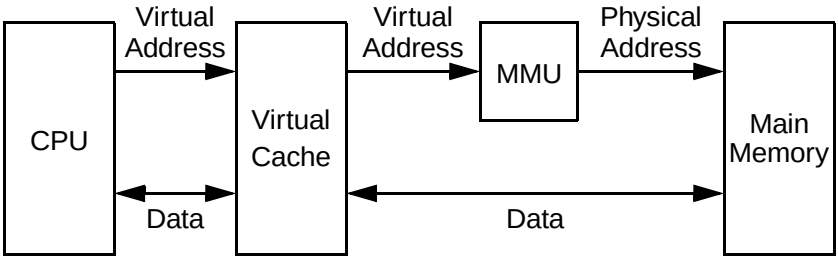
- Size and set size
  - Small cache → set-associative implementation with large sets
- Line length
  - Spatial locality → long cache lines
- Write policy
  - Temporal locality → write-back
- Replacement policy
- Using virtual or physical addresses for tagging/indexing

Long Cache Lines vs. Cache Alignment

- Cache line length influences what a good size for data structures is
- Problem: Multiple misses caused by “non-aligned” data structures spanning multiple cache lines
- Solution: Padding of structures to a multiple of line length

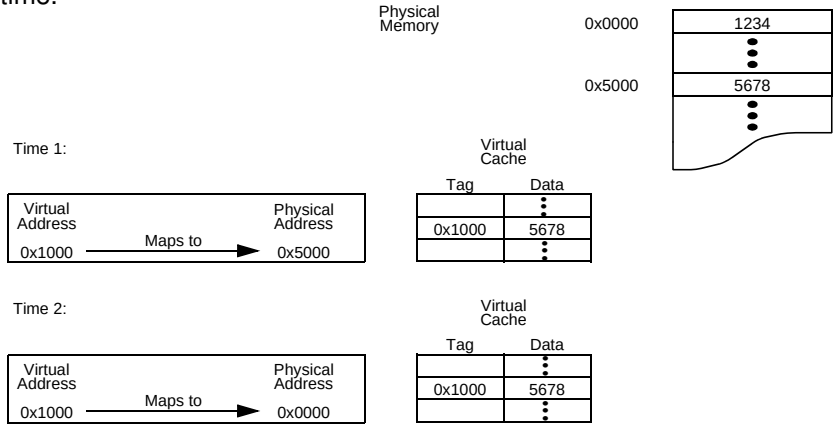


Virtually Indexed, Virtually Tagged (ARMv4/v5)[Sch99]



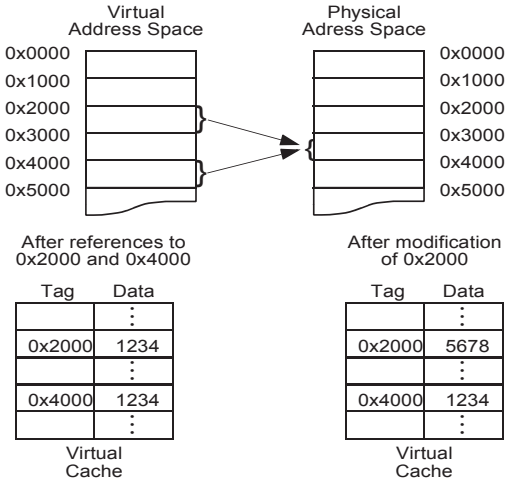
Ambiguity Problem[Sch99]

- Identical virtual addresses point to different physical addresses at different points in time.



Alias Problem[Sch99]

- Different virtual addresses point to the same physical memory location



Virtually Indexed, Virtually Tagged Cache Operations

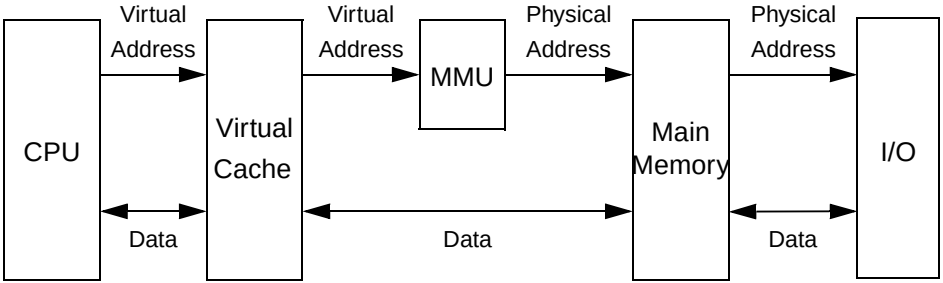
- **Context switch**: Cache must be invalidated (and written back if write-back is used), as identical virtual addresses of different address spaces might (likely) reference different physical addresses
- **fork**: The child needs a complete copy of the parent's address space.
  If copy operation runs in the context of the parent, special cache management is required.
- **exec**: Invalidate cache to prevent ambiguities. Not necessary to write content back, as memory is overwritten anyway.
- **exit**: Flush cache
- **brk**/**sbrk**: Growing requires no action, shrinking requires (selective) cache invalidations

Virtually Indexed, Virtually Tagged Cache Operations

- Shared memory and memory mapped files → alias problem
  (multiple virtual address refer to the same physical memory)
  - Disallow
  - Do not cache
  - Only allow addresses that map to the same cache line
    (if cache is direct-mapped and uses write-allocate)
  - Each frame accessible from exactly one virtual address at each point in time (requires invalidation of all alias pages in the page table)

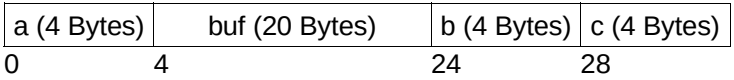
Virtually Indexed, Virtually Tagged I/O[Sch99]

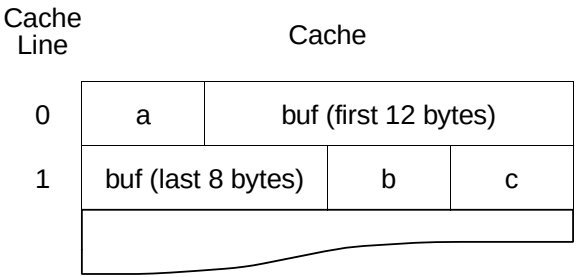
- Buffered I/O
  - No problems
- Unbuffered I/O
  - Write: Information might still be in the cache → write back before I/O starts
  - Read: Cache must be invalidated



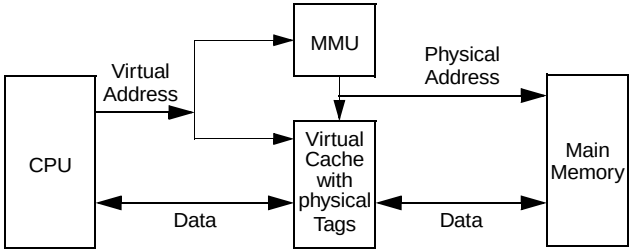
Virtually Indexed, Virtually Tagged: Unbuffered I/O[Sch99]

- Additional problems if I/O region is not aligned with length of cache line





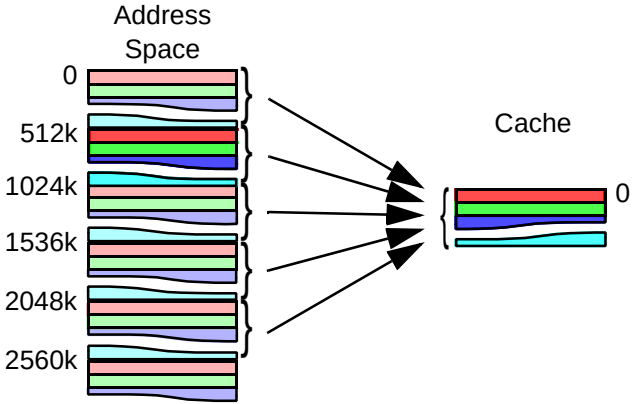
Virtually Indexed, Physically Tagged (ARMv6/v7)[Sch99]



- Often used as first-level caches
  - e.g., UltraSPARC II: 16 kB direct mapped
- Cache management in VIPT caches
  - No ambiguities
  - No cache flush on context switch
  - Shared memory/memory mapped files:
    Virtual starting addresses must be mapped to the same cache line
  - I/O: Cache flush required as with virtually indexed, virtually tagged cache

VIPT Conflicts

- Data structures whose address distance is a multiple of the cache size are mapped to the same cache line



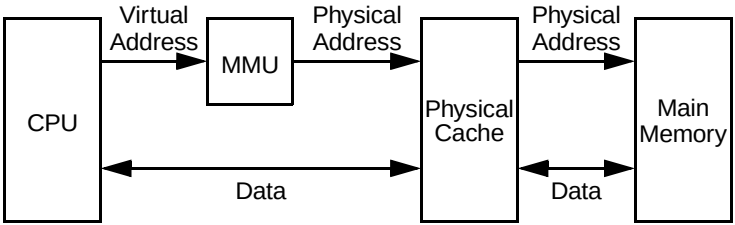
VIPT Runtime Properties

- Cache flush can be avoided most of the time
  - Fast context switches, interrupt-handling and system calls
- Deferred write-back after context switch

→ Avoids write accesses (performance gain)
→ Variable execution time caused by compulsory misses (depends on access pattern of previous thread)

- Variable execution time in case of dynamic memory management caused by conflict misses
- Variable search time caused by address translation in MMU
- Problematic in multiprocessor systems with shared memory: Which line to invalidate?
  - Cache size is a small multiple of page size (factor 1-4)
  - Requires to only invalidate/flush 1-4 cache lines by cache coherency HW

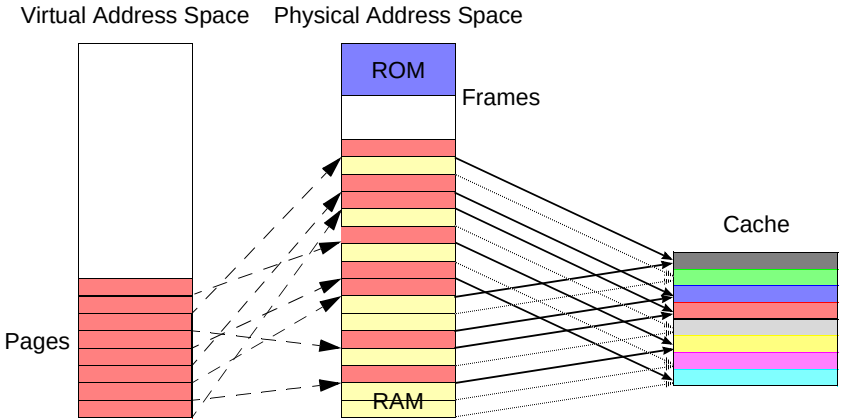
Physically Indexed Physically Tagged Caches[Sch99]



- Completely transparent to processor
- No performance-critical system support required (including I/O)
- SMPs with shared memory can use coherency protocol implemented in hardware

PIPT Random Allocation Conflicts

- Contiguous virtual memory is normally mapped to arbitrary free physical pages
- Page conflicts caused by random allocation of physical memory



PIPT Random Coloring Conflicts

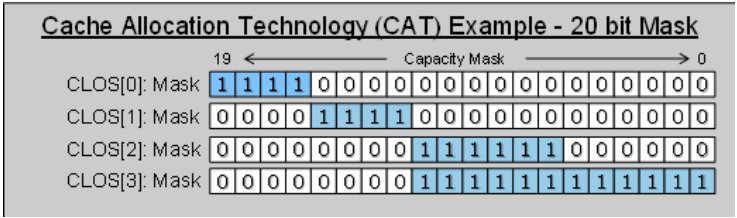
- Consequences of random page coloring:
  - Cache Conflicts
  - Cache only partially used
  - Significant variations in runtime

PIPT Conflict Mitigation

- Sequential page colors for individual memory segments
  - E.g., red-yellow-green-blue-red-yellow . . .
- Cache partitioning [LHH97]
  - Divide physical memory in disjoint subsets.
  - All pages of a subset are mapped to the same cache partition
  - Example:
    All red and blue pages for operating system
    All yellow pages for “the” real-time application
    All green pages for background processes

PIPT Conflict Mitigation: Intel® Cache Allocation Technology

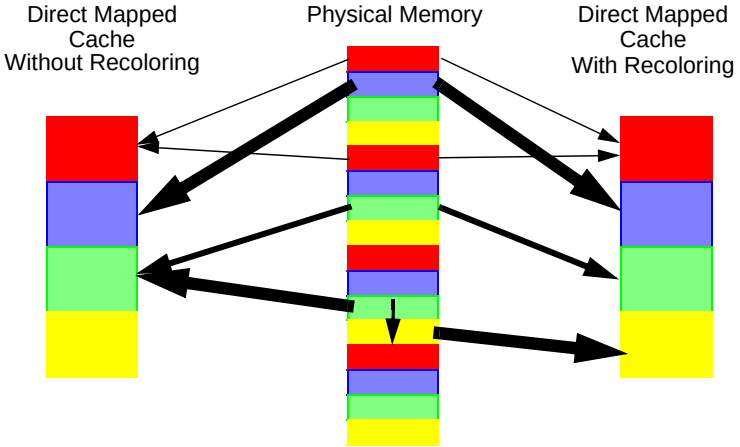
- Class of service (CLOS) defines the amount of resources (cache space) available via MSRs(Model Specific Register).
- Associate each logical thread with an available logical CLOS
- As the OS swaps a thread onto a core, update the CLOS on the core via a CLOS-specific MSR



[Ngu16] [int20]

PIPT Conflict Mitigation: Page Recoloring

- Analysis of access patterns (WS analysis) and page-recoloring



References I

[int20] *Four-Volume Set of Intel® 64 and IA-32 Architectures Software Developer's Manuals*, 2020. retrieved from `https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html#three-volume`.

[LHH97] J. Liedtke, H. Härtig, and M. Hohmuth. Os-controlled cache predictability for real-time systems. In *Proceedings Third IEEE Real-Time Technology and Applications Symposium*, pages 213–224, 1997.

[Ngu16] K. T. Nguyen. Usage models for cache allocation technology in the intel® xeon® processor e5 v4 family. *Intel Development Topics & Technologies*, 2016.

[Sch99] Curt Schimmel. *UNIX systems for modern architectures : symmetric multiprocessing and caching for kernel programmers*. Addison-Wesley professional computing series. Addison-Wesley, Reading, Mass. [u.a.], 3rd print. edition, 1999.