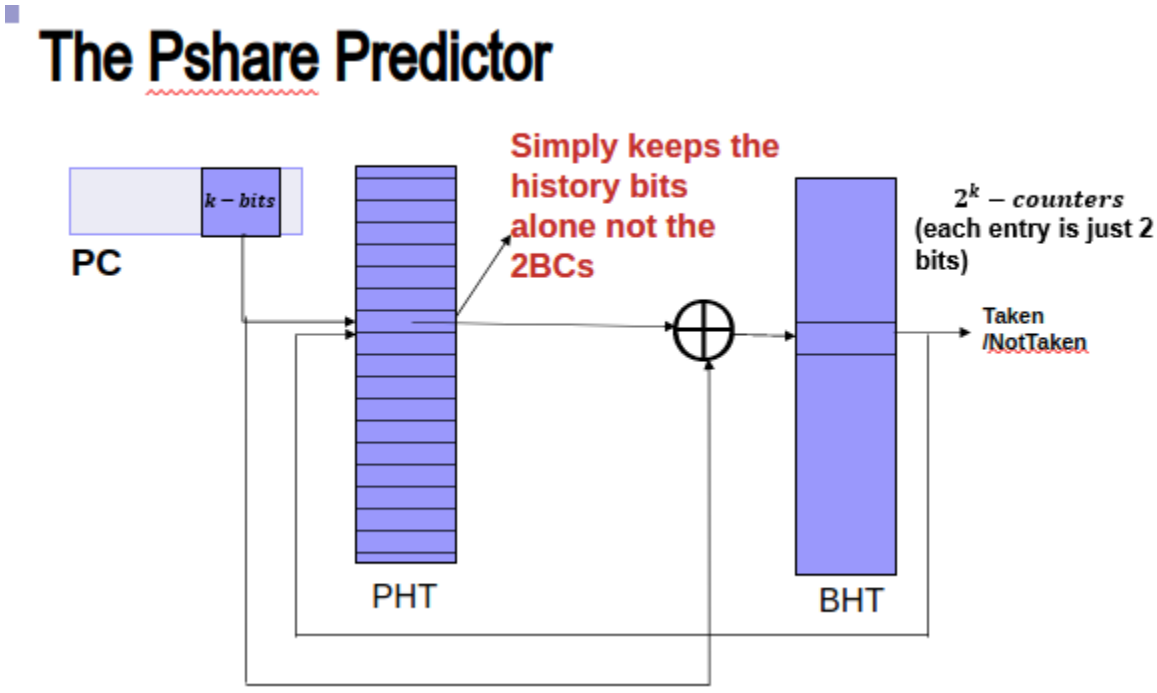
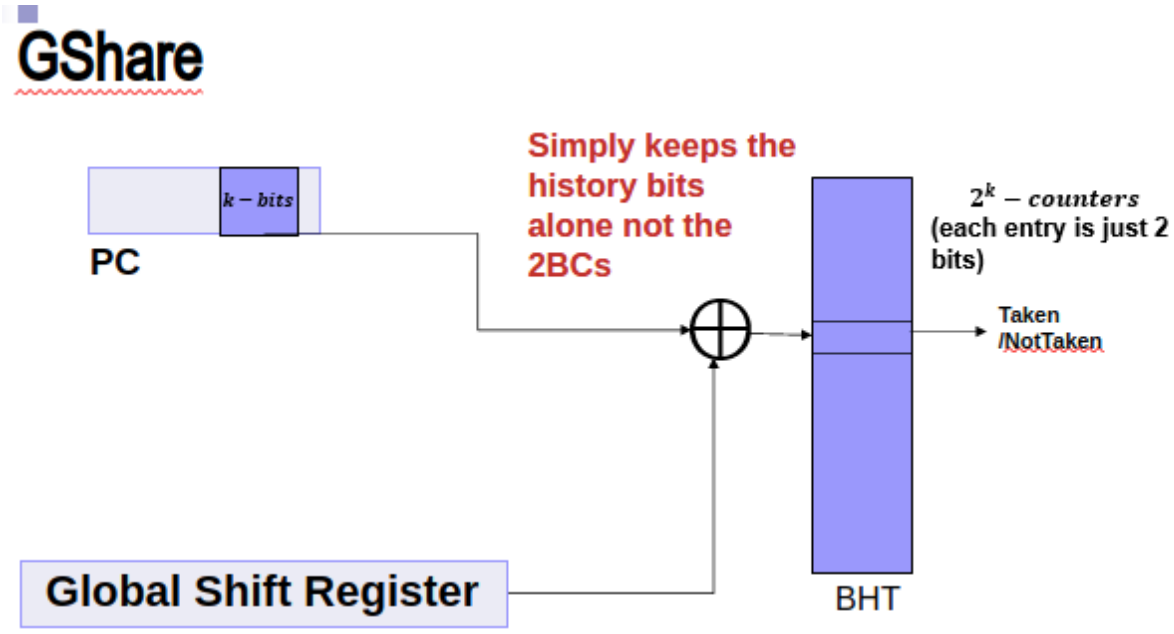
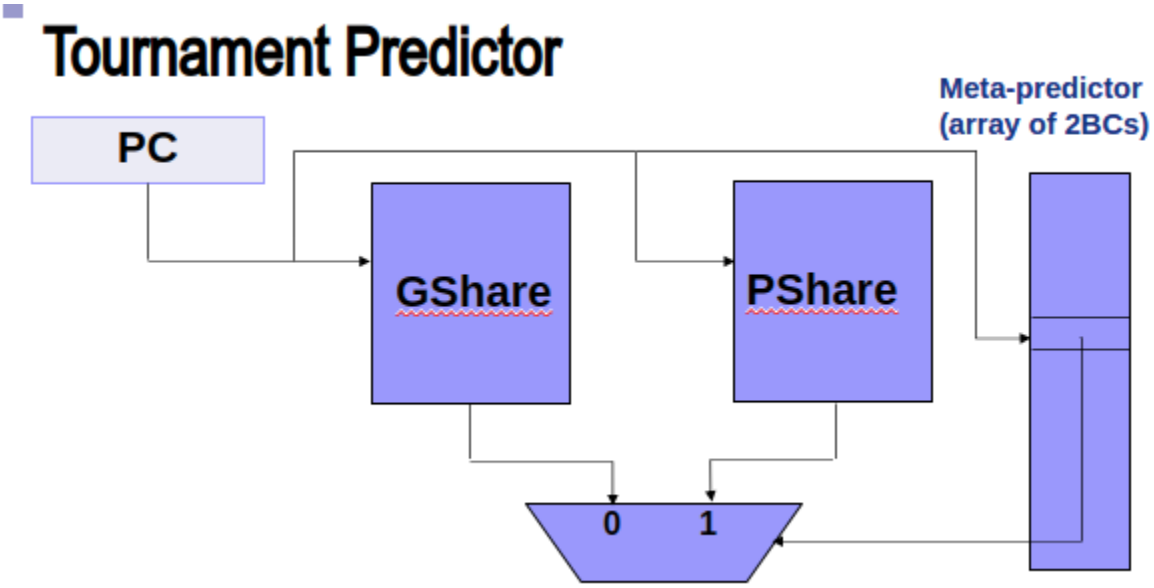
1. Latency: delay from start to done for one output
2. Throughput: no. of outputs/second
3. Iron Law of Performance: execution time = Number of Instructions × CPI × cycle time
4. Amdalh's law: Overall Speedup= 1/[1-p + p/s] where s is the speedup for p proportion of the task.
5. LHADMA's Law: Cautions that in pursuit of optimizing the common case, the uncommon case shouldn't be slowed down too much.
6. Diminishing returns with respect to Amdahl's Law: continuing to optimize a specific part of the execution eventually provides less and less gains in speedup
7. Branch Target Buffer (BTB): cache like structure, that records the addresses of the branches and the target addresses associated.
8. Least Signification Bits of Program counter (PC) are used to index BTB
9. Branch History Table (BHT): Tells whether branch is taken or not
10. Branch predictors: 1-bit predictor, 2-bit predictor
11. The previous predictors have flaw. Hence, use 1-bit history with two 2-bit counters. So if branch was not taken the use counter 1 otherwise use counter 2
12. Similarly, 2-bit History Predictor has four 2-bit counters associated with it
13. N-bit History Predictor requires O(2^N) counters but only few of them are used in reality leading to wastage of counters
14. Pattern History Table (PHT): Each entry in a PHT is a 2-bit counter. Indexed by some bits of Program counter (PC). Multiple PCs share 2-bit counters.
15. Bimodal predictor uses PHT to make predictions
16. Local history predictor structure: PHT->BHT. PHT contains history [indexed by PC], BHT contains 2 bit counters [indexed by history]
17. PShare has local history predictor structure with additional property that hash function is used to determine the index of BHT. Input to hash function is history bits and PC bits



18. Local history predictor fail to detect correlated branches. In this case use global shift register instead, which makes it a global predictor
19. GShare: History is stored in a single global register. Hash function is used to determine the index of BHT. Input to hash function is history bits and PC bits. No PHT is present here. Cons: GShare requires bigger history.

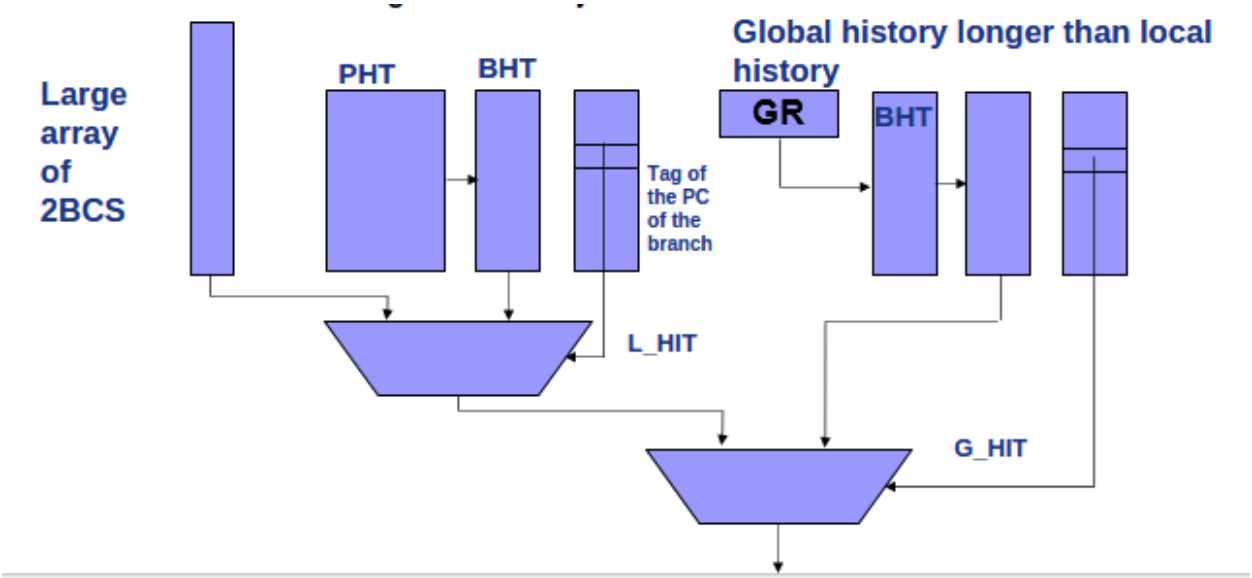


20. Tournament predictor uses both GShare and PShare.

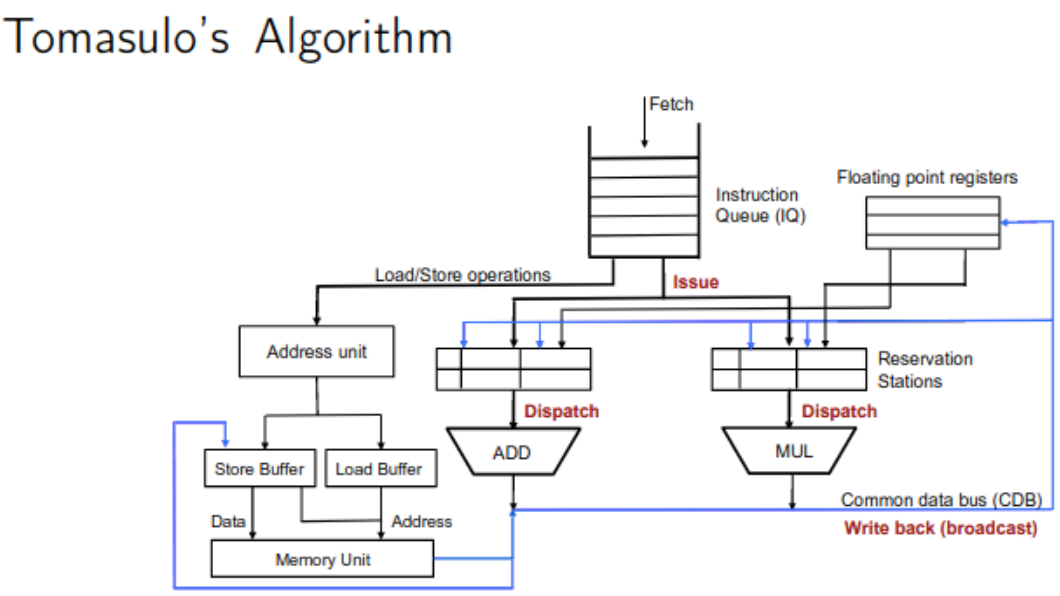


21. Meta-predictor is used in Tour. Pred. which tells which one is more accurate between the GShare and PShare.It is made up of 2 bit counters
22. Hierarchical: 1 Good, 1 OK predictor. we update the OK predictor on each decision, we update the Good predictor only if the OK prediction is not good.

23. Example of Hier. Pred.- Pentium M uses a Cheap predictor with 2BCs, Predictor with local history, Predictor with global history.

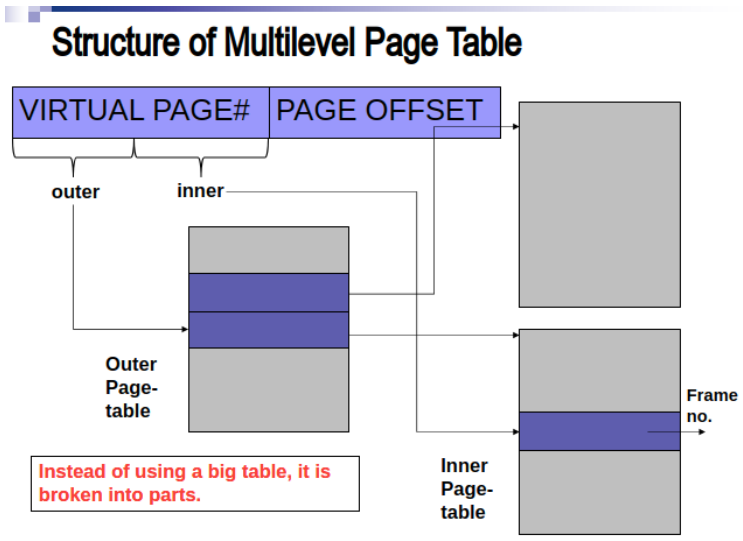


24. Repair Mechanisms: Non-speculative and speculative [see slides for more details, not so important in my opinion]
25. To improve performance, the following is done for each hazard, control-> Branch Predictor, WAR / WAW -> renaming, RAW -> OOO (out of order), structural deps -> wider issue HW design, add more resources
26. Tomasulo′s Algorithm: Used for OOO execution
27. Schematic diagram of Tomasulo's algorithm:



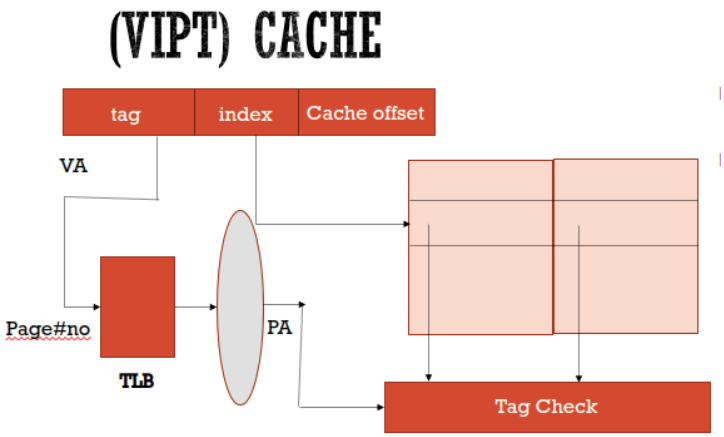
28. Reorder buffer (ROB): Used to deal with branch misprediction and exceptions. Contains Two pointers: COMMIT : from where to commit (basically write into register), ISSUE: from where to write in ROB. Commit takes place in-order.

29. A superscalar processor is a CPU that implements a form of parallelism called instruction-level parallelism within a single processor.
30. Load-store queue (LSQ) in Tomasulo: In order execution is done. Data is usually forwarded from a store instruction to a future load instruction if both are referring to the same memory. In this case, the load instruction immediately broadcasts the result to the matching Reservation Station.
31. Predication: all possible branch paths in if-else are coded inline, but some instructions execute while others do not. The basic idea is that each instruction is associated with a predicate (MOVN, MOVZ in mips) and that the instruction will only be executed if the predicate is true. Used when the if-else size is small.
32. Compiler support for Instruction level parallelism (ILP): Instruction scheduling (reschedule instructions to decrease data hazards) and loop unrolling (if loop runs for say 100 times, then by unrolling the loop once, the number of iterations becomes 50 [we run 2 loops in one go] and this decreases the number of cycles per instruction).
33. Virtual memory: Programmer's view of memory, Physical memory: actual memory,Processor's View of Memory.
34. Virtual mem is divided into equal size chunks called pages. Physical mem is divided into equal size chunks called frames. Page size=frame size
35. Page table: maps virtual address (VA) to physical address (PA). Virtual address = Virtual page number+ offset [this is not addition, just split the bits into 2 parts], Physical address = Physical page number + offset. No. of Offset bits = k, where 2^k is the page size. Same offset bits for both VA and PA.
36. Sometimes the page table cannot fit in memory because of the large virtual address space. In such a case, a multi-level page table is used.
37. Schematic diagram of multi-level page table:



38. In mutli-level page table, if inner table has no entry, then it is eliminated and the corresponding outer page table entry is marked as no entry required.
39. Translation Look-Aside Buffer (TLB): Cache for page table.
40. How is TLB better than using the cache itself?: TLB storing only translations can be very small and hence fast.

41. TLB Associativity: usually fully or highly associative
42. Average Memory Access Time (AMAT)=HIT TIME + MISS RATE x MISS PENALTY
43. Similar to processor pipelining, cache can be pipelined. It contains three stages: Finding the entry in cache where the data might be there, checking if it is an HIT or not, using the offset to select the corresponding byte.
44. A cache that is accessed with the Physical Address (PA) is called Physically Accessed Cache or Physically Indexed Physically Tagged (PIPT) cache.
45. Schematic diagram of Virtually indexed physically tagged (VIPT) cache:



46. Aliasing in VIPT Cache: Since index is coming from Virtual address(VA), two VAs with same PAs may end up writing in two different entries of the cache. This is called aliasing.
47. Avoiding Aliasing: Choose index from offset. It is same for VAs which have the same PA.
48. The idea of “way prediction” is based on an additional feature to predict/guess which line in a set is the most likely to result in a HIT. Reduces HIT time in set associative cache.
49. Replacement policy in cache: LRU (Least Recently Used), Not Most Recently Used (NMRU), Pseudo LRU (PLRU),
50. Flaw of LRU: we need to update many counters to keep track of the recency. This will consume power!
51. NMRU: We track which block in a set is MRU (Most Recently Used), On replacement, we pick a non-MRU block randomly.
52. PLRU: Here, all the bits are initialized to 0. Every time, a line is accessed it becomes a 1. If we need to replace a block, we choose among the blocks whose corresponding bits are 0. PLRU is a perfect balance between LRU and NMRU
53. Types of cache misses: Compulsory misses, capacity misses, conflict misses.
54. Prefetching: Predictive technique attempting to bring data or instructions to the cache before their usages, to reduce delay
55. Hardware prefetchers: Sequential prefetcher- prefetches line adjacent to the one that has just been referenced, stride prefetchers- used in a program with nested loops where the memory access patterns exhibit constant strides., correlating prefetchers- say we know that when data 1 is accessed then data 2 is accessed. Then it is better to prefetch data 2 whenever data 1 is accessed.

56. In blocking caches, the second load cannot be done before 1st load is finished.However, modern processors have caches that allow several concurrent misses. These are called non-blocking or lockup-free caches.
57. Missing Status Handling Registers (MSHR): When a cache miss occurs, an MSHR is associated with it. They remember the misses that are in progress. If there is another cache miss for the same data (also known as half miss), the present memory access should not be again done. Instead, wait for the data to arrive as already it is being requested earlier.
58. Cache coherence problem explained with the help of an example:

Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1	0

Figure 5.3 The cache coherence problem for a single memory location (X), read and written by two processors (A

59. Cache coherence protocols- directory and snooping.
60. Directory based—The sharing status of a particular block of physical memory is kept in one location, called the directory.
61. Snooping—Rather than keeping the state of sharing in a single directory, every cache that has a copy of the data from a block of physical memory could track the sharing status of the block. This is done by monitoring the broadcast medium (also known as snooping)
62. Snooping based protocols: Write update and write invalidate
63. One method is to ensure that a processor has exclusive access to a data item before writing that item. This style of protocol is called a write invalidate protocol because it invalidates other copies on a write. It is by far the most common protocol. Exclusive access ensures that no other readable or writable copies of an item exist when the write occurs: all other cached copies of the item are invalidated.
64. The alternative to an invalidate protocol is to update all the cached copies of a data item when that item is written. This type of protocol is called a write update or write broadcast protocol. Because a write update protocol must broadcast all writes to shared cache lines, it consumes considerably more bandwidth. For this reason, virtually all recent multiprocessors have opted to implement a write invalidate protocol.
65. In MSI, each block contained inside a cache can have one of three possible states:

    Modified: The block has been modified in the cache. The data in the cache is then inconsistent with the backing store (e.g. memory). A cache with a block in the "M" state has the responsibility to write the block to the backing store when it is evicted.
    Shared: This block is unmodified and exists in read-only state in at least one cache. The cache can evict the data without writing it to the backing store.

Invalid: This block is either not present in the current cache or has been invalidated by a bus request, and must be fetched from memory or another cache if the block is to be stored in this cache

These coherency states are maintained through communication between the caches and the backing store. The caches have different responsibilities when blocks are read or written, or when they learn of other caches issuing reads or writes for a block.

When a read request arrives at a cache for a block in the "M" or "S" states, the cache supplies the data. If the block is not in the cache (in the "I" state), it must verify that the block is not in the "M" state in any other cache. Different caching architectures handle this differently. For example, bus architectures often perform snooping, where the read request is broadcast to all of the caches. Other architectures include cache directories which have agents (directories) that know which caches last had copies of a particular cache block. If another cache has the block in the "M" state, it must write back the data to the backing store and go to the "S" or "I" states. Once any "M" line is written back, the cache obtains the block from either the backing store, or another cache with the data in the "S" state. The cache can then supply the data to the requester. After supplying the data, the cache block is in the "S" state.

When a write request arrives at a cache for a block in the "M" state, the cache modifies the data locally. If the block is in the "S" state, the cache must notify any other caches that might contain the block in the "S" state that they must evict the block. This notification may be via bus snooping or a directory, as described above. Then the data may be locally modified. If the block is in the "I" state, the cache must notify any other caches that might contain the block in the "S" or "M" states that they must evict the block. If the block is in another cache in the "M" state, that cache must either write the data to the backing store or supply it to the requesting cache. If at this point the cache does not yet have the block locally, the block is read from the backing store before being modified in the cache. After the data is modified, the cache block is in the "M" state.

For any given pair of caches, the permitted states of a given cache line are as follows:

	M	S	I
M	✗	✗	✓
S	✗	✓	✓
I	✓	✓	✓

66. MESI adds the state Exclusive to the basic MSI protocol, yielding four states (Modified, Exclusive, Shared, and Invalid). The exclusive state indicates that a cache block is resident in only a single cache but is clean. If a block is in the E state, it can be written without generating any invalidates, which optimizes the case where a block is read by a single cache before being written by that same cache

67. See this for MOESI:

Adding an "Owned" State: MOESI

- MESI must write-back to memory on M→S transition (a.k.a. ***downgrade***)
  - Because protocol allows "silent" evicts from shared state, a dirty block might otherwise be lost
  - But, the writebacks might be a waste of bandwidth
    - *e.g.*, if there is a subsequent store (common in producer-consumer scenarios)
- Solution: add an "Owned" state
  - Owned – shared, but dirty; only one owner (others enter S)
  - Owner is responsible for replying to read requests
  - Owner is responsible for writeback upon eviction
    - Or should transfer "ownership" to another cache