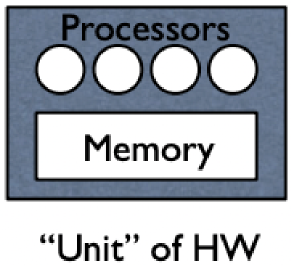
CAS CS 350
Lec 17

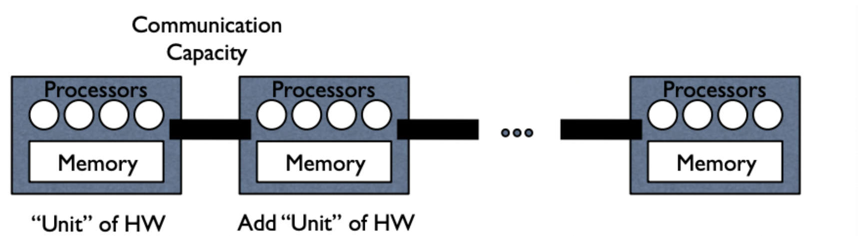
Shared Memory Systems

1. Today's Paper
    a. How do we take a bunch of workstations on a LAN and make them into a system that can be easily programmed
    b. Their approach - Distributed Shared Memory (DSM)
        i. Take familiar parallel programming model - threads - and move it over to a network of computers
        ii. Our focus intro DSM and explore consistency - two key ideas from the paper
            1. Lazy-release consistency
                a. Consistency is correctness condition for read/write operations
                b. Linearizability: every read will get the most recent write
                c. Serializability: hypothetical serial execution that has the same effect as the concurrent transaction (two phase locking ensures serializability)
                d. Strict serializability: serializability that hypothetical serial execution is run in respect to real time (combination of linearizability and serializability)
            2. Version vectors (common implementation technique in distributed systems)
2. The big idea
    a. Computational models for distributed computing (hides all complexity of distribution, fault tolerance, etc. so that users can focus on the logic of the program)
        i. Treadmarks
            1. Can just execute their program in a cluster of machines
        ii. Mapreduce
    b. How to implement a framework that hides details of all machines?
    c. How to keep programming model "simple"?
    d. How to achieve the performance goals?
3. Consider
    a. Machine with 4 physical threads in memory



b. → know how to write parallel codes (channels, etc.)... fork off threads that work together by reading and updating variables - data structures

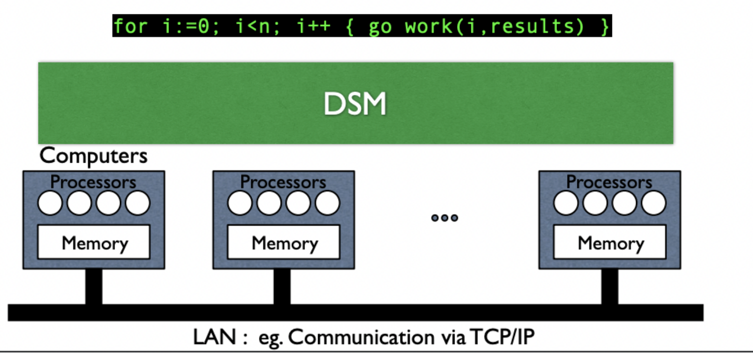
c. Cluster of machines with threads, memory, and etc. to a network → distribute the code depending on the availability sources



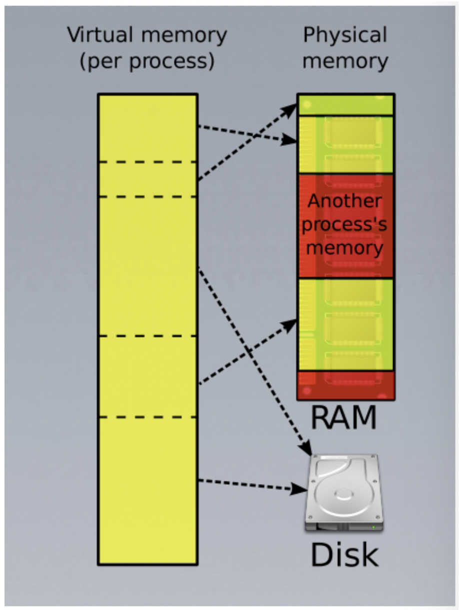
d.

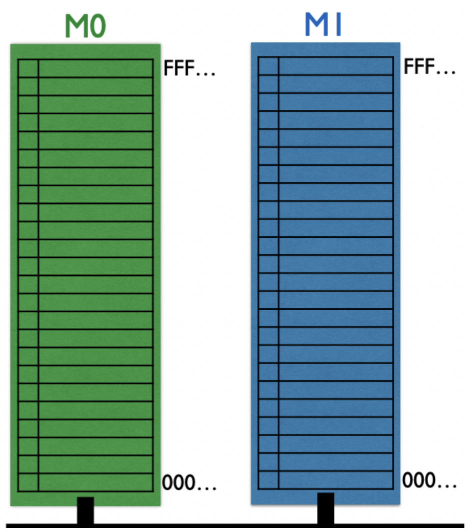
e. We need Shared Memory! → provide all goroutines illusions that they are connected (although in reality it is not the case)

4. DSM
    a. Simulate shared memory on a distributed collection of computers
    b. Apps don’t directly do communication… instead DSM software provide the primitives for a language like go - Shared Address space, threads, locks, channels, etc.

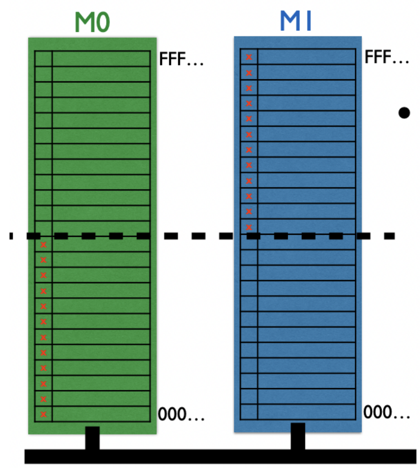


c.
    i. Familiar model (do not need to modify or rewrite the program) → just execution
    ii. Very general purpose (no restrictions in program → can write any program)
    iii. Huge collection of existing threaded libraries

5. DSM Basic Plan (Slow)
    a. Exploit the fact that hardware already has support for "virtual memory"
    b. Paging hardware and os software trampoline to DSM SW
    c. 

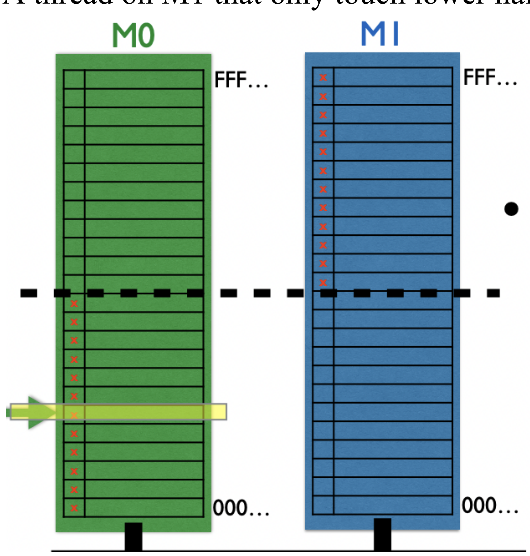
    d. Managing the memory requires tasks
    e. 

        i. Two machines on a network
        ii. Address space controlled by page table
        iii. Pages can be marked R, W, RW, NONE



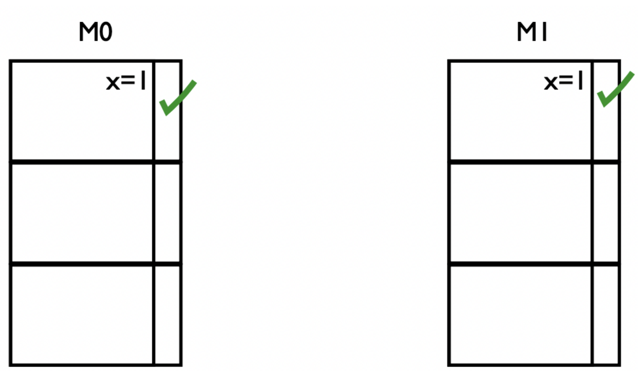
f.

g. Simple setup where at first we split the “shared” address space in half
    i. Top accessible “owned” by M0
    ii. Bottom accessible “owned” by M1
h. For any one page only accessible on a single machine
i. A thread on M0 that only touch upper half is fine
j. A thread on M1 that only touch lower half is fine

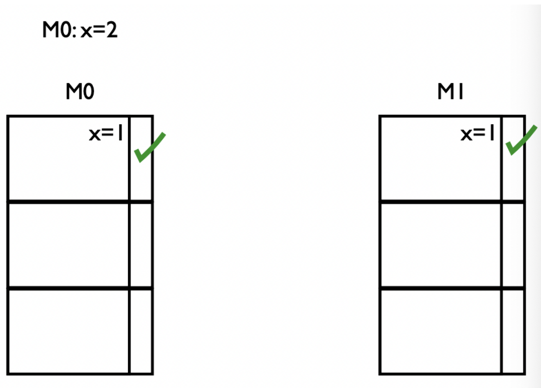


k.

l. When a thread on M0 touches a page in lower half - DSM software kicks in
    i. Hw generates fault
    ii. DSM code handles
        1. Copies page and marks page inaccessible on M1
        2. And accessible on M0
m. Threaded code just works! - eg. Matrix multiplication, sort, etc.

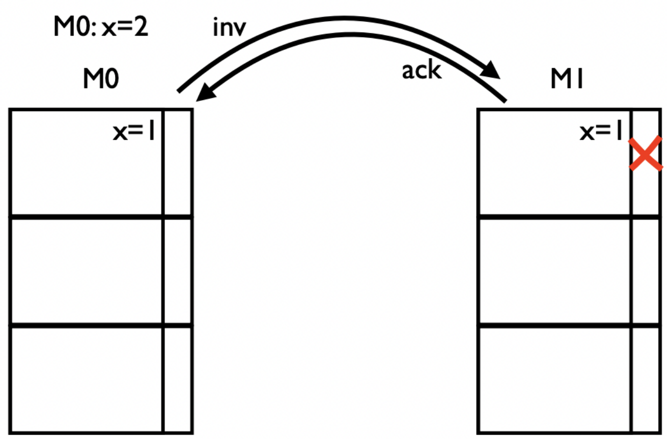
6. An example to think about
    a.
    ```
    X = Y = 0

    Thread 0              Thread 1
       X=1                    Y=1
       if Y==0 {              if X==0 {
         print "yes"            print "yes"
       }                      }
    ```
    b. Can both these print yes?
        i. Could allow one to see two yes
    c. In other words, go's memory model is not the one you might have expected (like many modern languages/machines)
7. Memory model
    a. Explains how reads and writes in different threads interact with each other
    b. Its a contract - there are many memory models out there due to tension between
        i. Give compiler/hw freedom to optimize - the more relaxed the MM greater optimization
        ii. Programmer Guranties - things people can reason about that when writing code - stricter easier - no funny business
8. TreadMarks is trying to address these issues
    a. Write Amplification
    b. False sharing
9. Write amplification
    a. Write amplification: a one byte write turns into a while-pager transfer
        i. Not good → transfer data over network and there are many bugs on the network
    b. How could we tackle this problem?
        i. Just transfer the delta (changes)
10. Fix for Write Amplification - write diffs
    a. 

    b. Both machines see the same variable for variable x

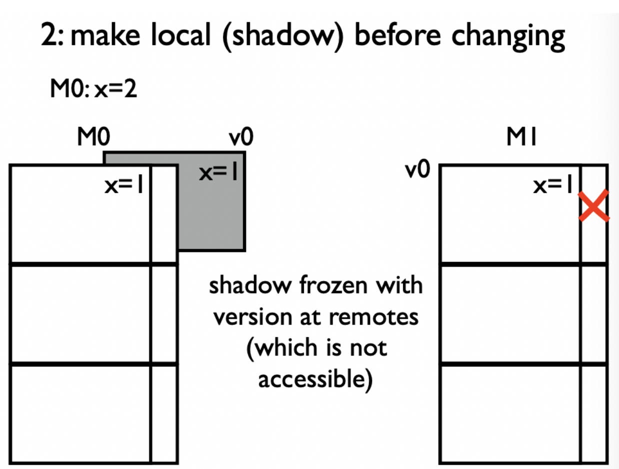


c.

d. At some point, programming running on M0 changes the value of x

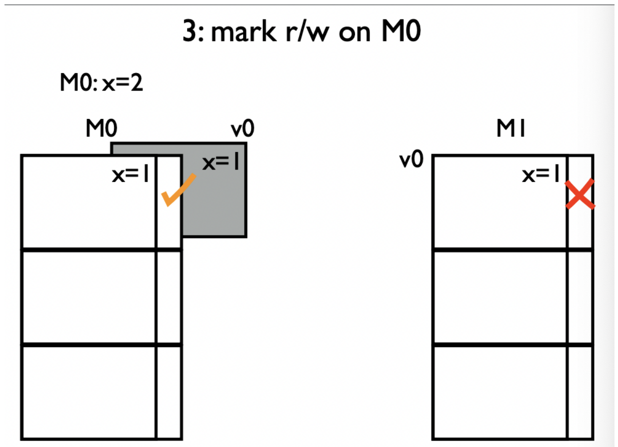


e.

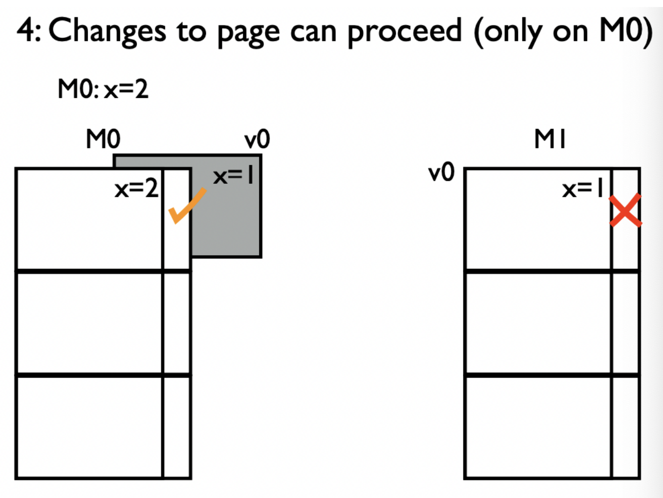


f.

g. Make a subtle copy of x before changing the value of x

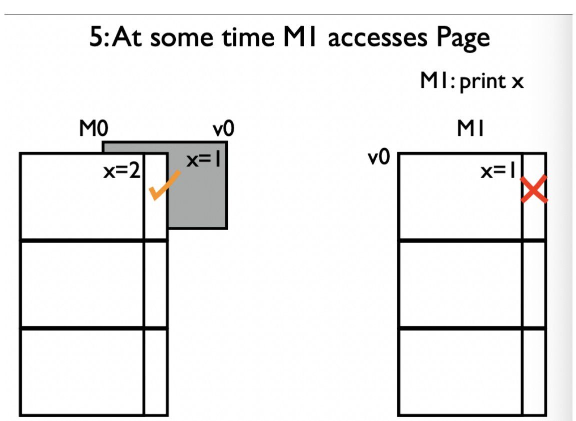
h. 

i. In most case, there are read only data. If the user wants to write, it has to send request

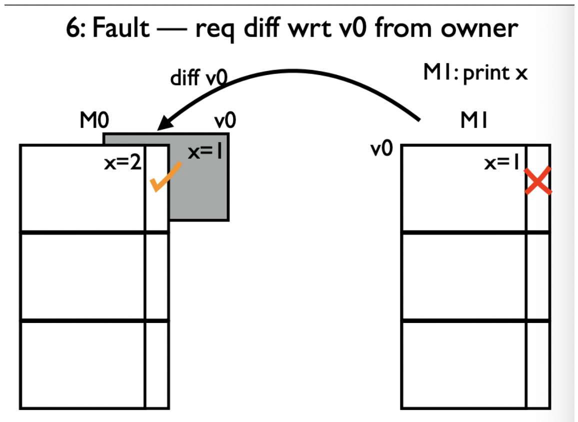
j. 

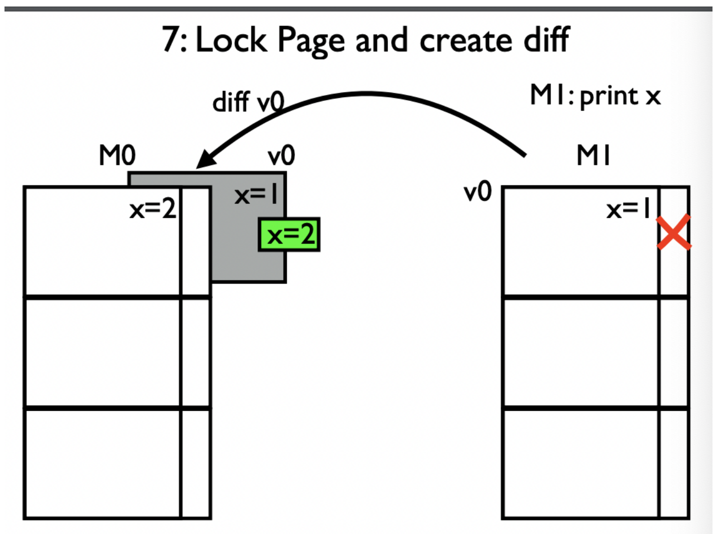
k. Happens on the real memory (not on subtle copy) → not accessible to any other machines except M0

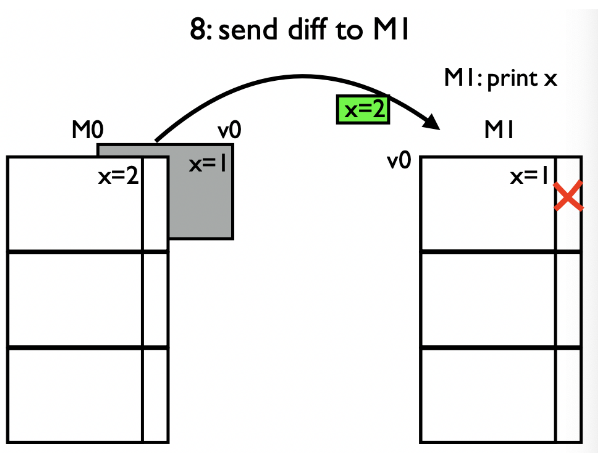
l. Subtle copy → to be able to compute delta and send only the delta to other machines

m. 

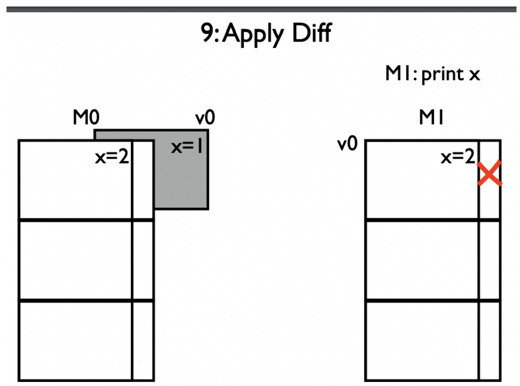
n. M1 wants to read value of x → request req diff from write0

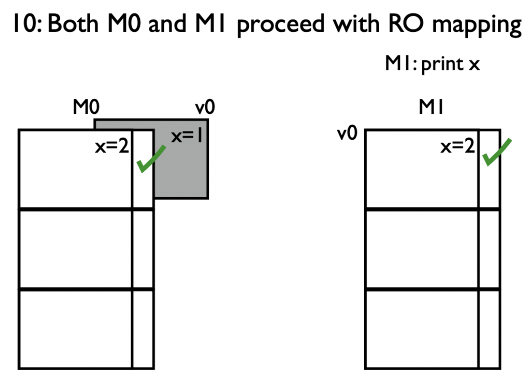
o. 

p. 

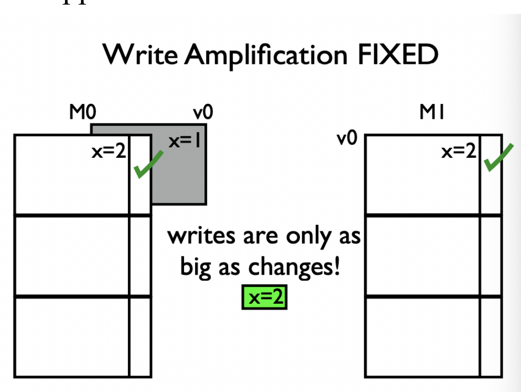
q. 

r. Send to only Machine 1

s. 

t. 

u. M1 applies diff to its machine

v. 

w. After this, both machines have only read-access to the page

11. Do write diffs change the consistency model?
    a. No
    b. Consistency model remains exactly the same
    c. Performance is better → move fewer data since data is transferred only when it is requested (more efficient)
    d. At most one writable copy, so writes are ordered
    e. No writing while any copy is readable, so not stale reads
    f. Readable copies are up to date, so no stale reads