CS 425 / ECE 428
Distributed Systems
Fall 2026

Aishwarya Ganesan & Ram Kesavan

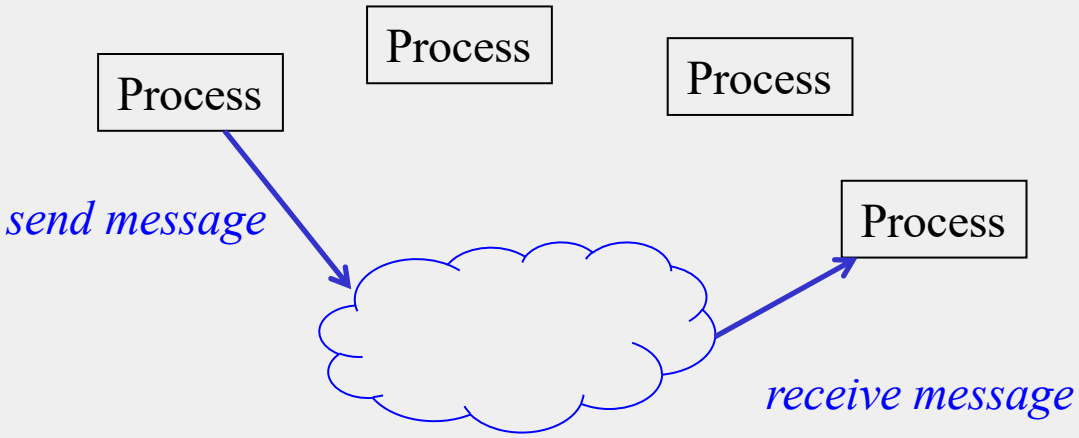
Lecture 25: Distributed Shared Memory

Ack: Indranil Gupta

All slides © IG

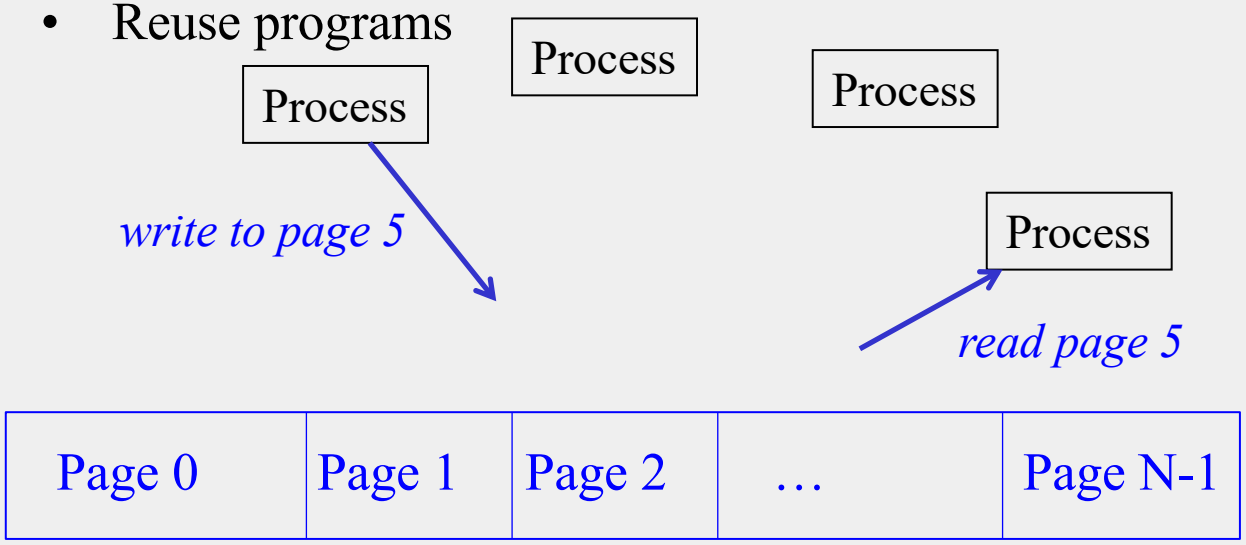
Thus Far …

- Message passing network



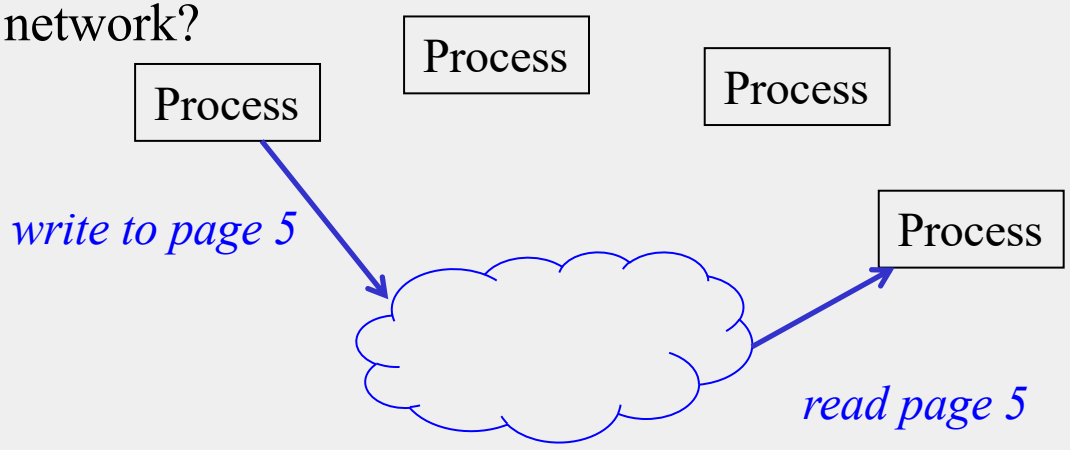
But what if …

- Processes could *share* memory pages instead?
- Makes it convenient to write programs
- Reuse programs



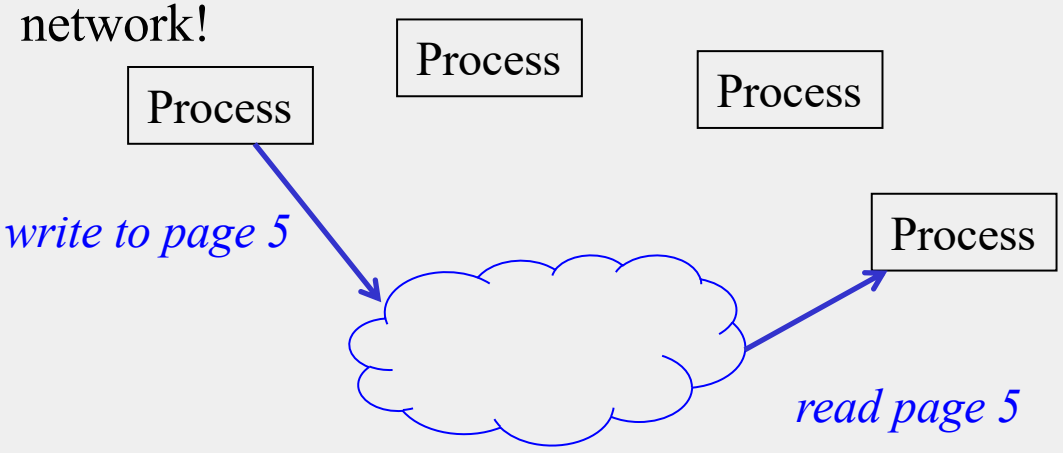
Distributed Shared Memory

- Distributed Shared Memory = processes virtually share pages
- How do you implement DSM over a message-passing network?



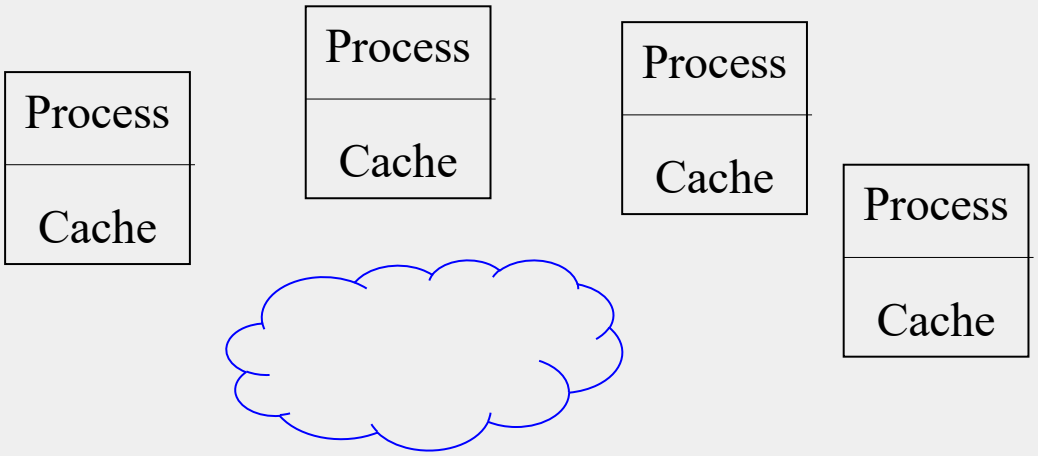
In fact …

1. Message-passing can be implemented over DSM!
   - Use a common page as buffer to read/write messages
2. DSM can be implemented over a message-passing network!



DSM over Message-Passing Network

- ***Cache*** maintained at each process
  - Cache stores pages accessed recently by that process
- Read/write first goes to cache



DSM over Message-Passing Network (2)

- Pages can be mapped in local memory
- When page is present in memory, page hit
- Otherwise, *page fault* (kernel trap) occurs
  - Kernel trap handler: invokes the DSM software
  - May contact other processes in DSM group, via multicast

DSM: Some Historical Perspective

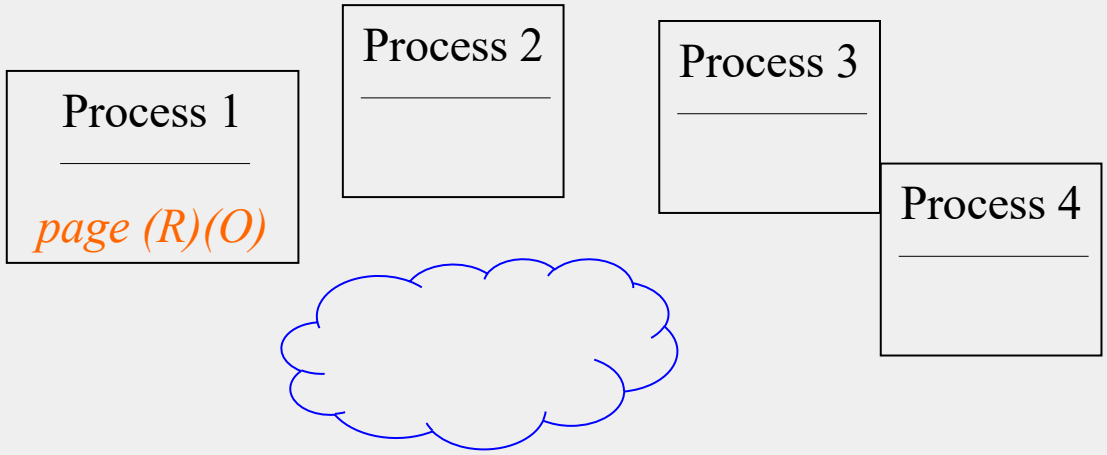
- Trendy research topic in 80/90s
  - Multiprocessor workstations became popular
  - Large computations without cost of supercomputing hardware
- Decline in 00/10s
  - Due to network bottlenecks; Ethernet was too slow
  - Industry pivoted to ”shared-nothing” architecture & message passing models
- Modern resurgence

DSM: Invalidate Protocol

- Owner = Process with latest version of page
- Each page is in either R or W state
- When page in R state, owner has an R copy, but other processes may also have R copies
  - but no W copies exist
- When page is in W state, only owner has a copy

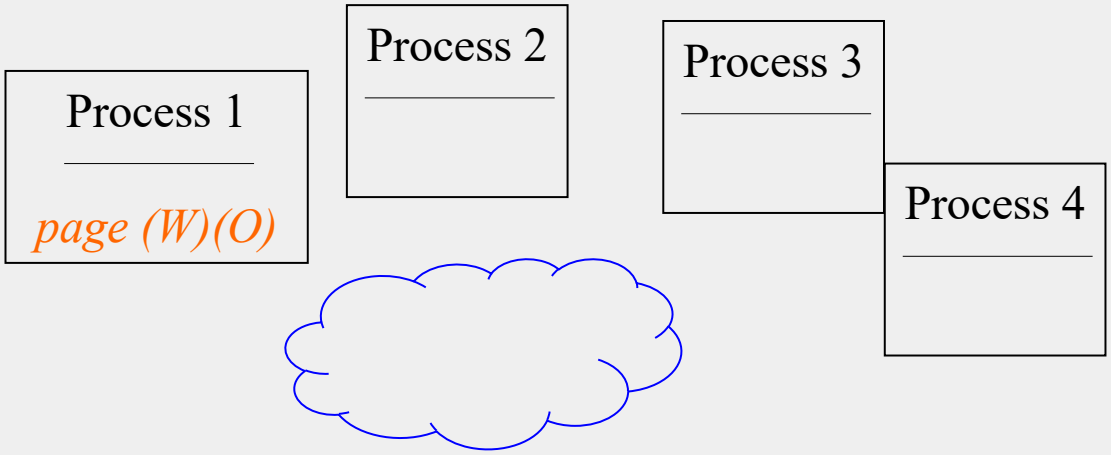
Process 1 Attempting a Read: Scenario 1

- Process 1 is owner *(O)* and has page in R state
- *Read from cache. No messages sent.*



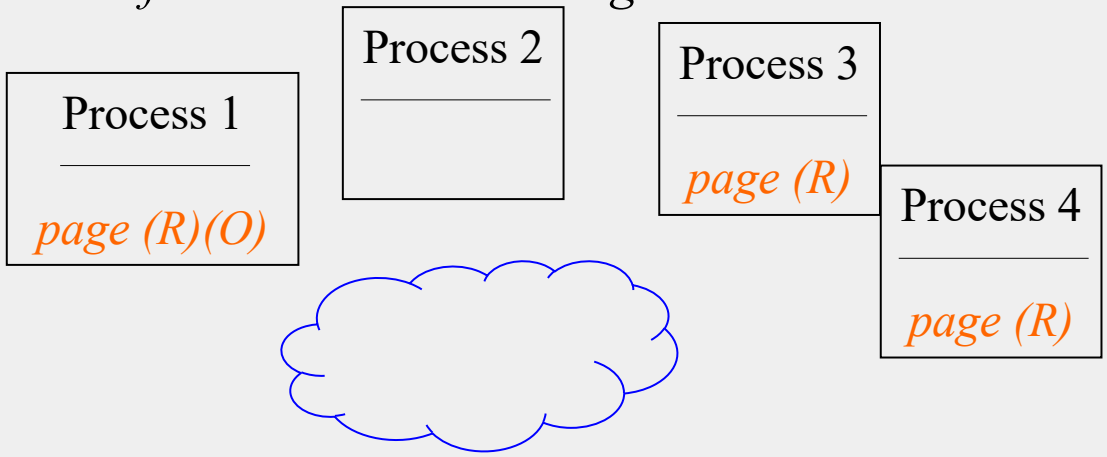
Process 1 Attempting a Read: Scenario 2

- Process 1 is owner *(O)* and has page in W state
- *Read from cache. No messages sent.*



Process 1 Attempting a Read: Scenario 3

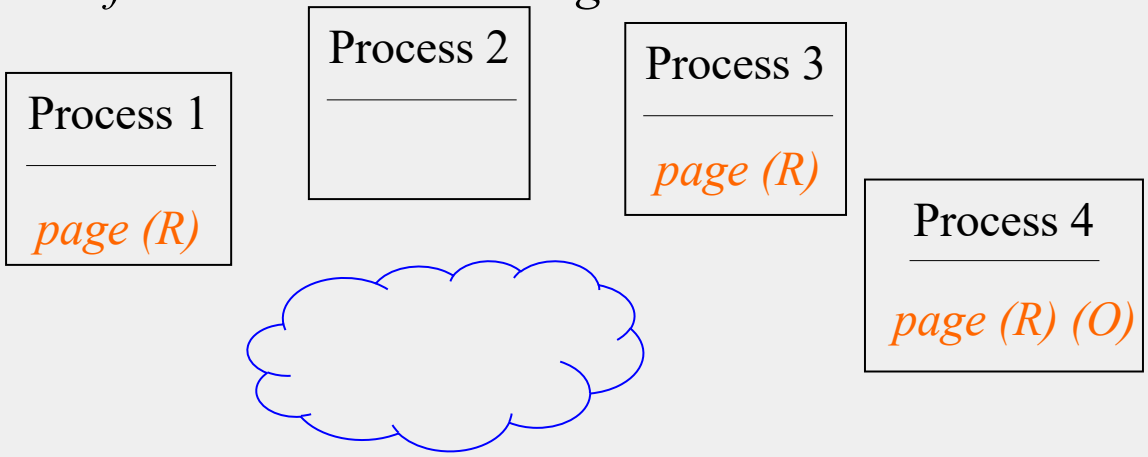
- Process 1 is owner *(O)* and has page in R state
- Other processes also have page in R state
- *Read from cache. No messages sent.*



Process 1 Attempting a Read: Scenario 4

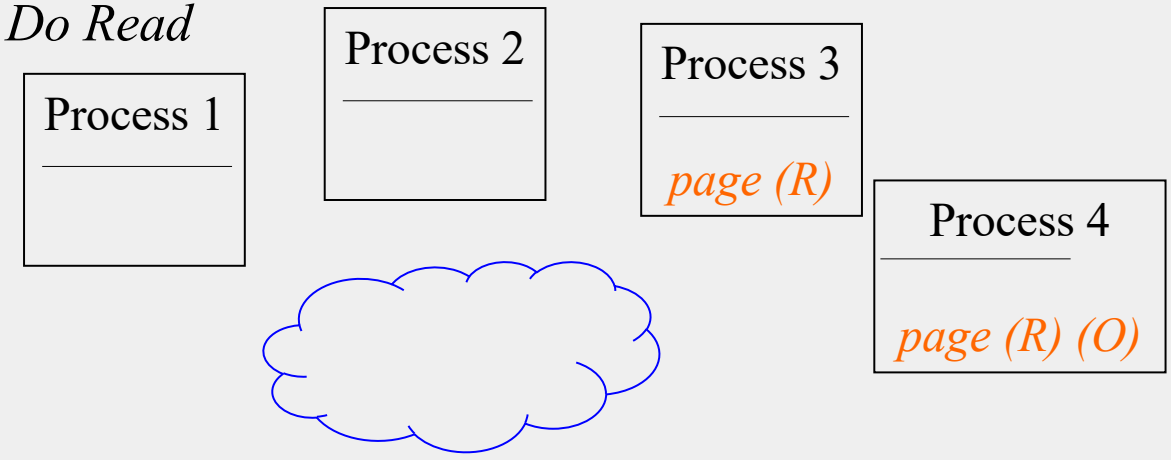
Can process1 read from cache? Why or why not?

- Process 1 has page in R state
- Other processes also have page in R state, and someone else is owner
- *Read from cache. No messages sent.*



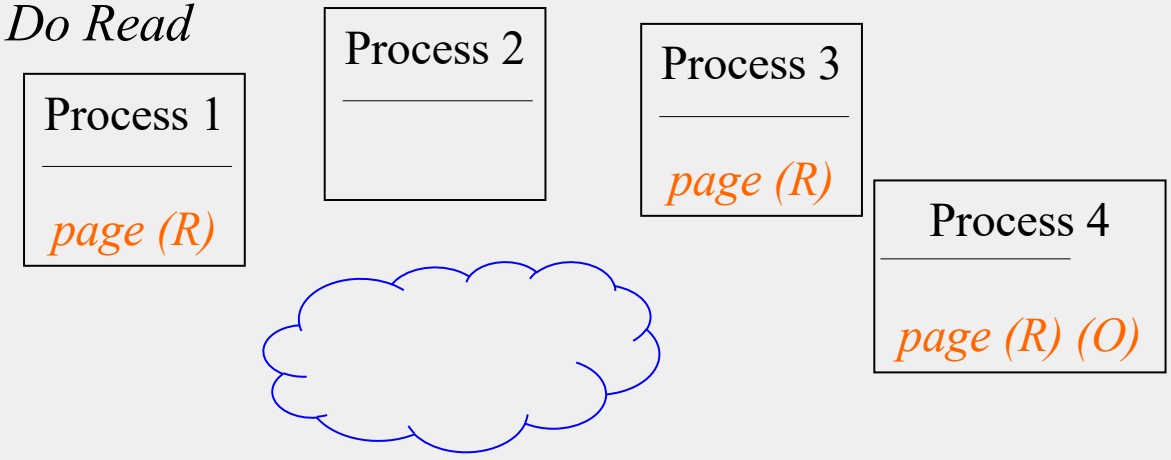
Process 1 Attempting a Read: Scenario 5

- Process 1 does not have page
- Other process(es) has/have page in (R) state
- *Ask for a copy of page. Use multicast.*
- *Receive page. Mark it as R*
- *Do Read*



End State: Read Scenario 5

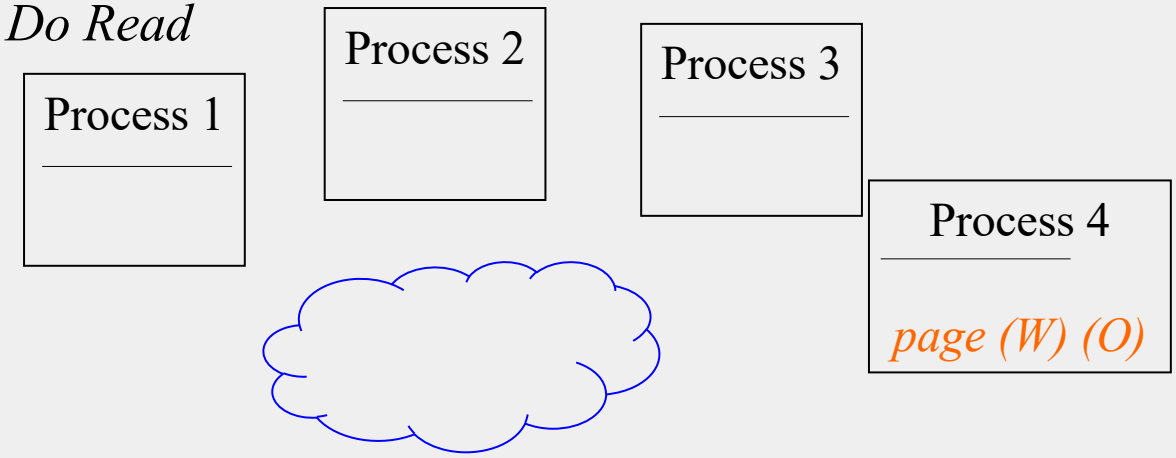
- Process 1 does not have page
- Other process(es) has/have page in (R) state
- *Ask for a copy of page. Use multicast.*
- *Receive page. Mark it as R*
- *Do Read*



Process 1 Attempting a Read: Scenario 6

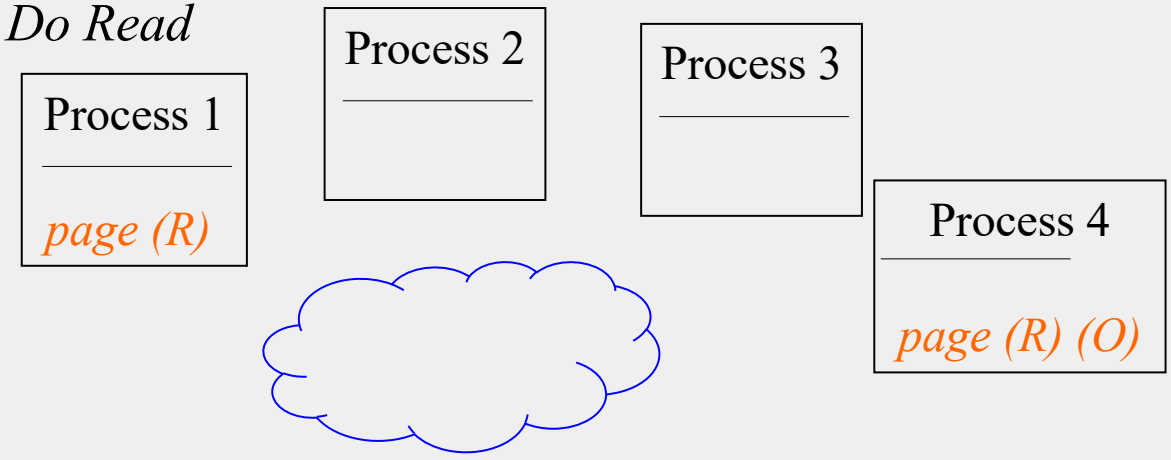
Can Process 1 get a copy from Process 4, simply mark it as R, and do the read?

- Process 1 does not have page
- Another process has page in (W) state
- *Ask for copy of page. Use multicast. P4 downgrades from W to R.*
- *Receive page; mark it as R*
- *Do Read*



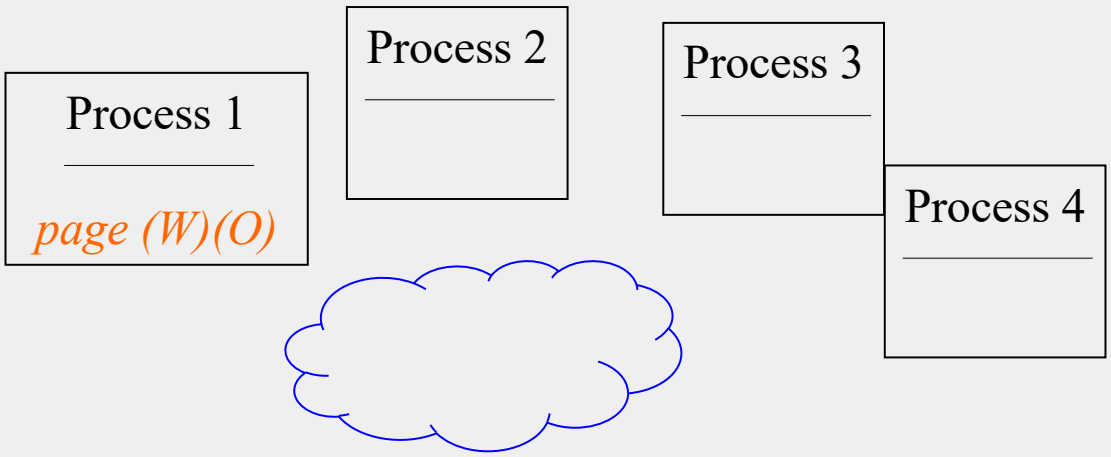
End State: Read Scenario 6

- Process 1 does not have page
- Another process has page in (W) state
- *Ask for copy of page. Use multicast. P4 downgrades from W to R.*
- *Receive page; mark it as R*
- *Do Read*



Process 1 Attempting a Write: Scenario 1

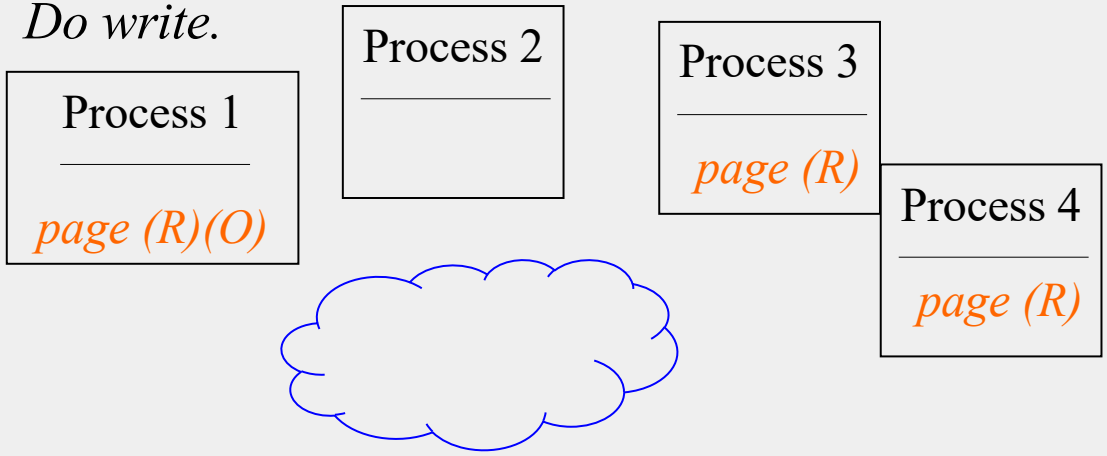
- Process 1 is owner *(O)* and has page in W state
- *Write to cache. No messages sent.*



Process 1 Attempting a Write: Scenario 2

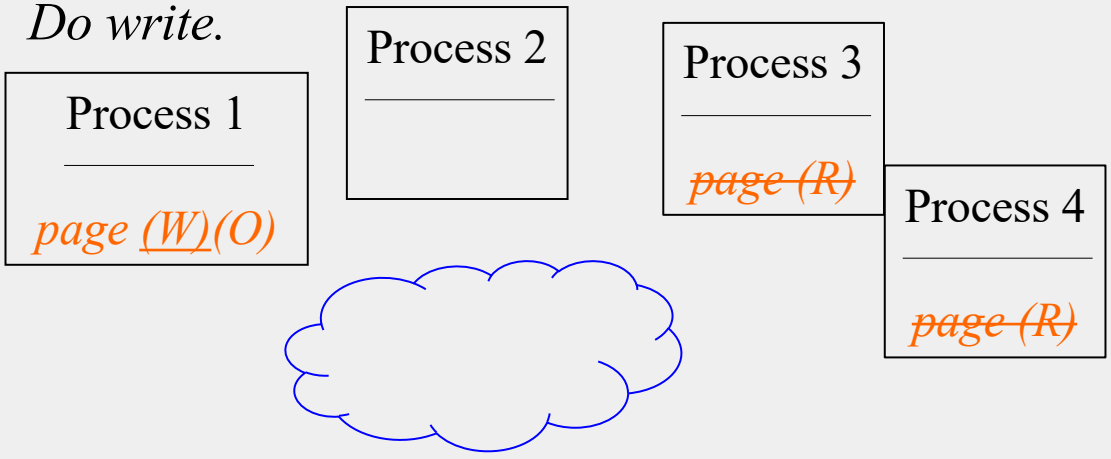
Can Process 1 simply mark the page as W and write to it because it is the owner?

- Process 1 is owner *(O)* has page in R state
- Other processes may also have page in R state
- *Ask other processes to invalidate their copies of page. Use multicast.*
- *Mark page as (W).*
- *Do write.*



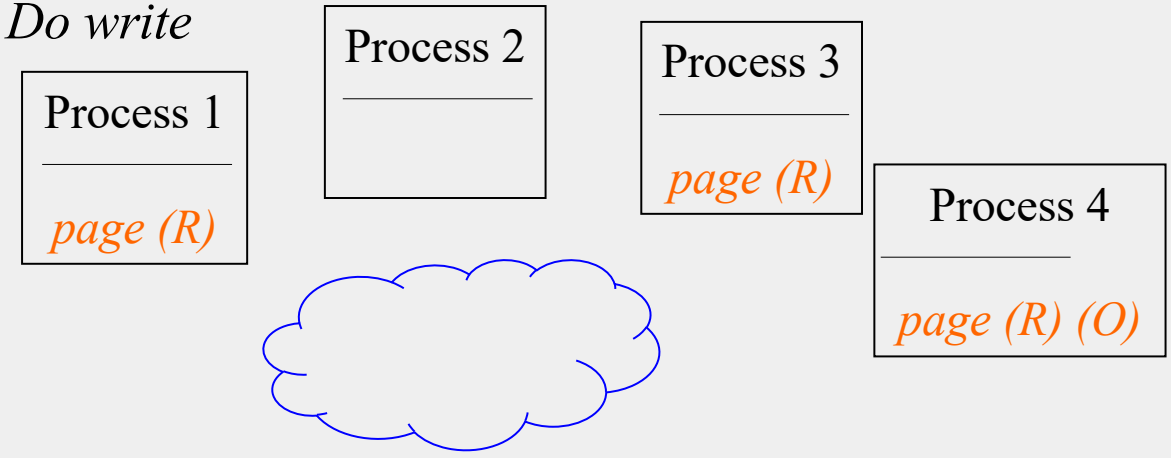
End State: Write Scenario 2

- Process 1 is owner *(O)* has page in R state
- Other processes may also have page in R state
- *Ask other processes to invalidate their copies of page. Use multicast.*
- *Mark page as (W).*
- *Do write.*



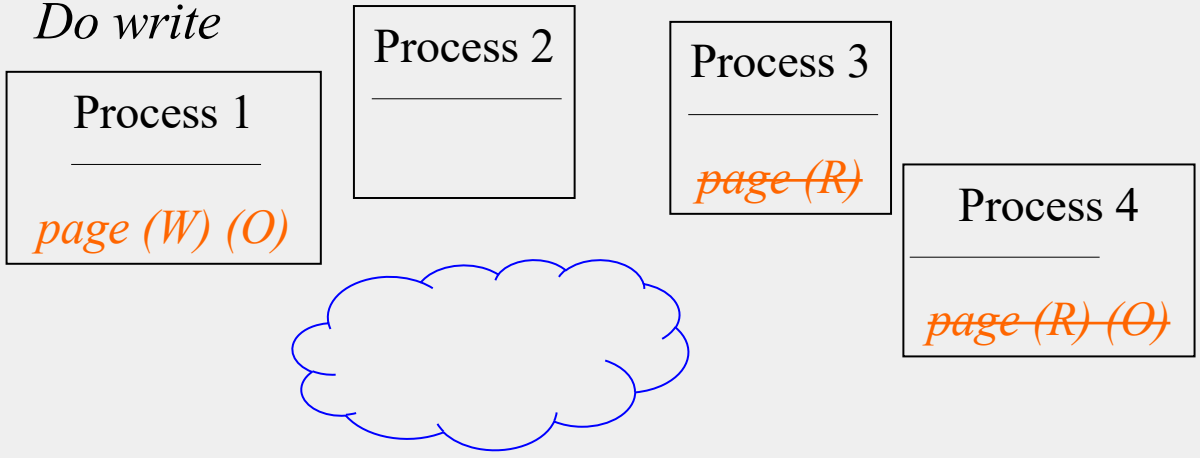
Process 1 Attempting a Write: Scenario 3

- Process 1 has page in R state
- Other processes may also have page in R state, and someone else is owner
- *Ask other processes to invalidate their copies of page. Use multicast.*
- *Mark page as (W), become owner*
- *Do write*



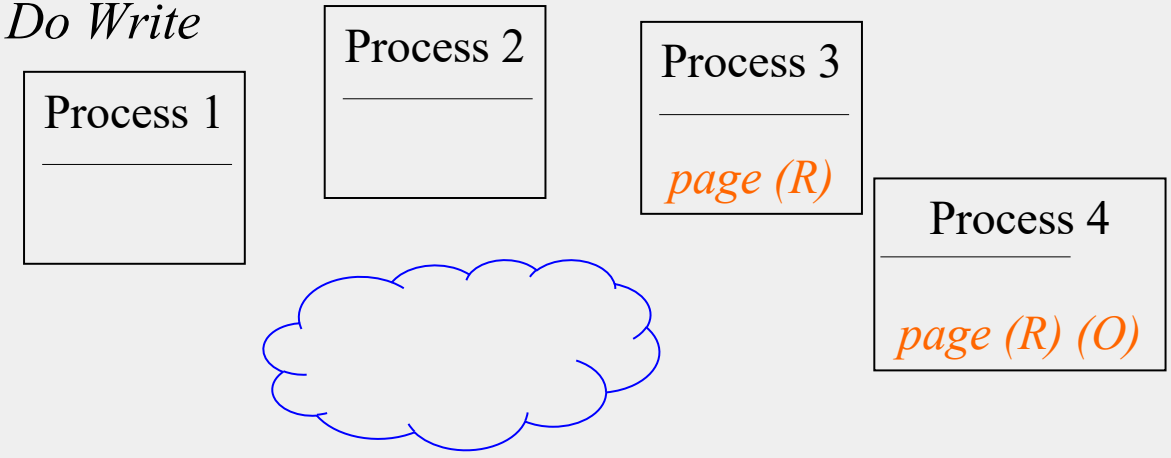
End State: Write Scenario 3

- Process 1 has page in R state
- Other processes may also have page in R state, and someone else is owner
- *Ask other processes to invalidate their copies of page. Use multicast.*
- *Mark page as (W), become owner*
- *Do write*



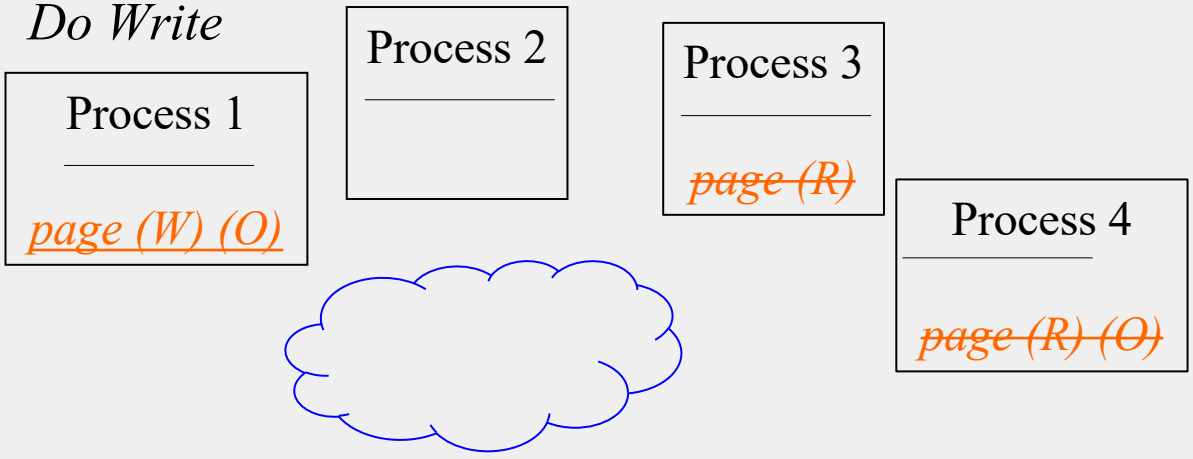
Process 1 Attempting a Write: Scenario 4

- Process 1 does not have page
- Other process(es) has/have page in (R) or (W) state
- *Multicast: ask other processes to invalidate their copies and request a copy*
- *Receive copy (with ack) from P4; mark it as (W); become owner*
- *Do Write*



End State: Write Scenario 4

- Process 1 does not have page
- Other process(es) has/have page in (R) or (W) state
- *Multicast: ask other processes to invalidate their copies and request a copy*
- *Receive copy (with ack) from P4; mark it as (W); become owner*
- *Do Write*



Invalidate Downsides

- That was the invalidate approach
- If two processes write same page concurrently
  - Back-n-forth messages where one process invalidates the other
  - Lots of network transfer
  - Can happen when unrelated variables fall on same page
  - Called false sharing
- Need to set page size to capture a process' *locality of interest*
  - If page size much larger, then have false sharing
  - If page size much smaller, then too many page transfers => also inefficient

An Alternative Approach: Update

- Instead: could use Update approach
  - Multiple processes allowed to have page in W state
  - When process P writes to a page, DSM multicasts it (or the newly written fraction) to all other holders of that page
  - Other processes can then continue reading and writing page
  - Many mechanisms used to resolve races and guarantee consistency
- Update preferable over Invalidate
  - When lots of sharing among processes
  - Page size is large and/or writes are to small variables
- In general, Invalidate better and preferred option

Consistency

- Whenever multiple processes share data, consistency comes into picture
- DSM systems can be implemented with:
  - Linearizability
  - Sequential consistency
  - Causal consistency
  - Pipelined RAM (PRAM) consistency, aka FIFO consistency
  - Eventual consistency
  - (Also, other models like Release consistency)
  - These should be familiar to you from the course!
- As one goes down this list, speed increases while consistency gets weaker

Trendy Again

- DSM has not been popular the last 2 decades
- But is making a comeback
  - Ultra-fast hardware interconnects & SSDs
  - RDMA: Remote Data Memory Access
  - CXL (Compute Express Link) with in-built cache coherency
  - A server can read/write remote memory at near-local speeds
  - AI model training needs enormous amounts of RAM
  - DSM trendy again for pooling memory across server racks
  - Poised for growth

Summary

- DSM = Distributed Shared Memory
  - Processes share pages, rather than sending/receiving messages
  - Useful abstraction: allows processes to use same code as if they were all running over the same OS (multiprocessor OS)
- DSM can be implemented over a message-passing interface
- Invalidate vs. Update protocols
- Has become trendy recently