Consistency
What is consistency?

- **Consistency model:**
  - A constraint on the system state observable by applications

- **Examples:**
  - Local/disk memory:
    - write \( x = 5 \)
    - read \( x \) (should be 5)

  - Database:
    - \( x := x + 1; \ y := y - 1 \)
    - assert \( x + y = \text{const} \)

Single object consistency, also called “coherence”

Consistency across \( >1 \) objects
Consistency challenges

• No right or wrong consistency models
  – Tradeoff between ease of programmability and efficiency
  – E.g. what’s the consistency model for web pages?

• Consistency is hard in (distributed) systems:
  – Data replication (caching)
  – Concurrency
  – Failures
Example application program

- Is this program correct?
Example

• CPU0’s instruction stream: \(W(x) \ R(y)\)
• CPU1’s instruction stream: \(W(y) \ R(x)\)
• Enumerate all possible inter-leavings:
  – \(W(x)1 \ R(y)0 \ W(y)1 \ R(x)1\)
  – \(W(x)1 \ W(y)1 \ R(y)1 \ R(x)1\)
  – \(W(x)1 \ W(y)1 \ R(x)1 \ R(y)1\)
  – ....
• None violates mutual exclusion
An example distributed shared memory

- Each node has a local copy of state
- Read from local state
- Send writes to the other node, but do not wait
Consistency challenges: example

\[ W(x)1 \]
\[ W(y)1 \]

\[ x=1 \]
If \( y==0 \)
critical section

\[ y=1 \]
If \( x==0 \)
critical section
Does this work?

If $y = 0$

- critical section

If $x = 0$

- critical section
What went wrong?

CPU0 sees:
W(x)1
R(y)0

CPU1 sees:
W(y)1
R(x)0
Strict consistency

- Each operation is stamped with a global wall-clock time
- Rules:
  1. Each read gets the latest write value
  2. All operations at one CPU have time-stamps in execution order
Strict consistency gives “intuitive” results

• No two CPUs in the critical section
• Proof: suppose mutual exclusion is violated

CPU0: W(x)1 R(y)0
CPU1: W(y)1 R(x)0

W must have timestamp later than R

• Rule 1: read gets latest write

CPU0: W(x)1 R(x)0
CPU1: W(y)1 R(x)0

Contradicts rule 1: R must see W(x)1
Sequential consistency

- Strict consistency is not practical
  - No global wall-clock available
- Sequential consistency is the closest
- Rules: There is a total order of ops s.t.
  - Each CPUs’ ops appear in order
  - All CPUs see results according to total order (i.e. reads see most recent writes)
Sequential consistency is also intuitive

• Recall our proof for correctness
• Enumerate all possible total orders s.t.:
  – Each CPUs’ ops appear in order
  – All CPUs see results according to total order (i.e. reads see most recent writes)
• Show no total order violates mutual excl
Native DSM example gives no sequential consistency

CPU0 sees:
W(x)1
R(y)0
W(y)1

CPU1 sees:
W(y)1
R(x)0
W(x)1

No total order can explain this execution
Sequential consistency is easier to implement

- There’s no notion of real time
- System is free to order concurrent events
- However, when the system finishes a write or reveals a read, it commits to certain partial orders
Requirement for sequential consistency

1. Each processor issues requests in the order specified by the program
   - Do not issue the next request unless the previous one has finished

2. Requests to an individual memory location (storage object) are served from a single FIFO queue.
   - Writes occur in a single order
   - Once a read observes the effect of a write, it’s ordered behind that write
Native DSM violates R1,R2

- Read from local state
- Send writes to the other node, but do not wait

$R(x)0 \rightarrow W(x)1 \rightarrow W(y)1 \rightarrow R(y)0 \rightarrow R(x)0$
Ivy distributed shared memory

- What does Ivy do?
  - Provide a shared memory system across a group of workstations

- Why shared memory?
  - Easier to write parallel programs with than using message passing
  - We’ll come back to this choice of interface in later lectures
Ivy architecture

- Each node caches read pages
  - Why?
- Can a node directly write cached pages?

If page not found in local memory, request from remote node

Each processor’s local memory keeps a subset of all pages
Ivy aims to provide sequential consistency

- **How?**
  - Always read a fresh copy of data
    - Must invalidate all cached pages before writing a page.
    - This simulates the FIFO queue for a page because once a page is written, all future reads must see the latest value
  - Only one processor (owner) can write a page at a time
Ivy implementation

• The ownership of a page moves across nodes
  – Latest writer becomes the owner
  – Why?

• Challenge:
  – how to find the owner of a page?
  – how to ensure one owner per page?
  – How to ensure all cached pages are invalidated?
Ivy: centralized manager

manager

<table>
<thead>
<tr>
<th>Page#, copy_set, owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1, {..}, {A}</td>
</tr>
</tbody>
</table>

Page#, access

<table>
<thead>
<tr>
<th>p1, read</th>
</tr>
</thead>
</table>

A
B
C
Ivy: read

1. Page fault for p1 on C
2. C sends RQ(read request) to M
3. M sends RF(read forward) to A, M adds C to copy_set
4. A sends p1 to C, C marks p1 as read-only
5. C sends RC(read confirmation) to M
Ivy: write

1. Page fault for p1 on B
2. B sends WQ(write request) to M
3. M sends IV(invalidate) to copy_set = {C}
4. C sends IC(invalidate confirm) to M
5. M clears copy_set, sends WF(write forward) to A
6. A sends p1 to B, clears access
7. B sends WC(write confirmation) to M
Ivy properties

- Does Ivy work with our example app?
- Why does it need RC and WC messages?

x=1
If y==0
   critical section

y=1
If x==0
   critical section
How about failures?

• How does Ivy recover from failure?
• Do Ivy’s invariants hold across failure?