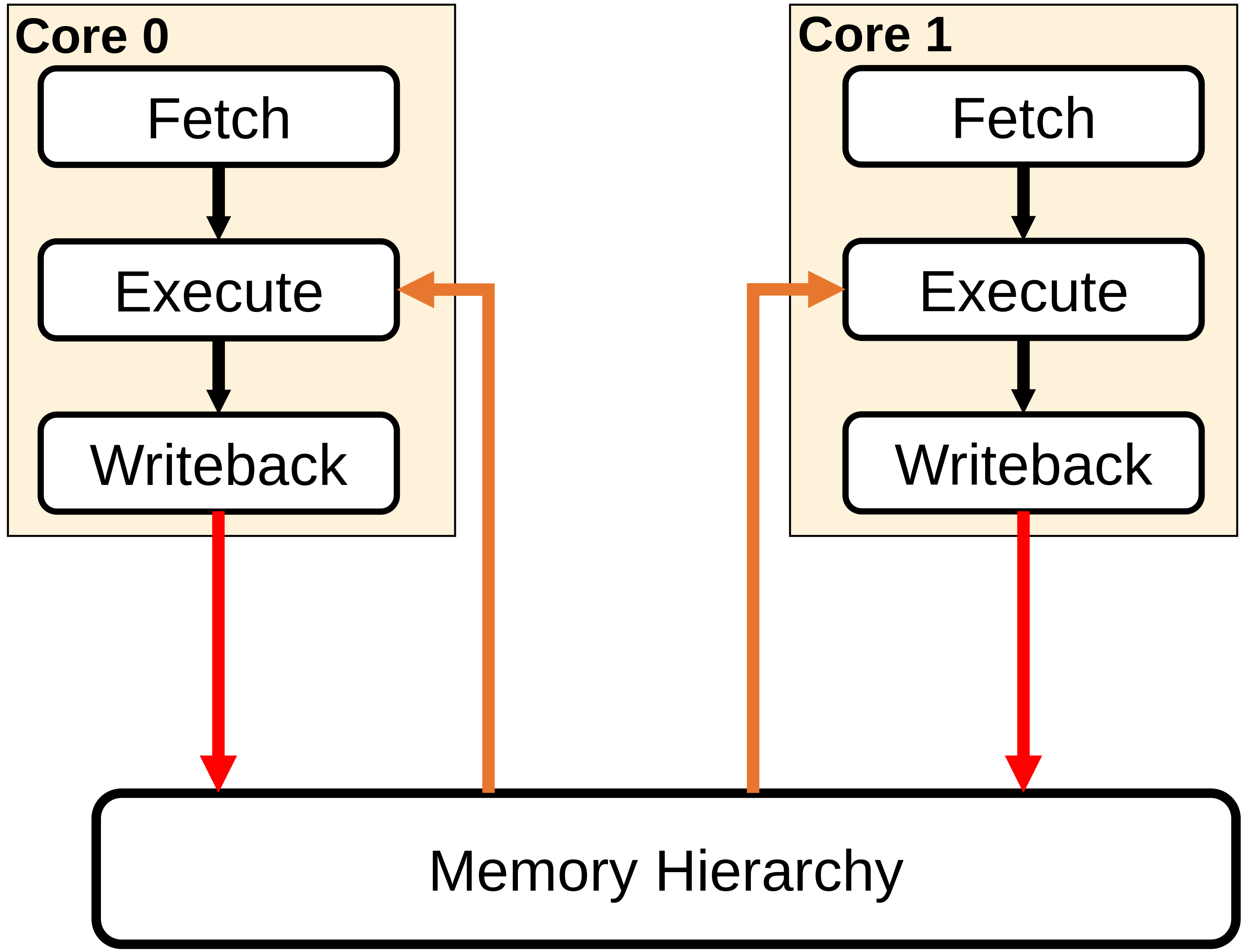
Extending SC uarch. to TSO

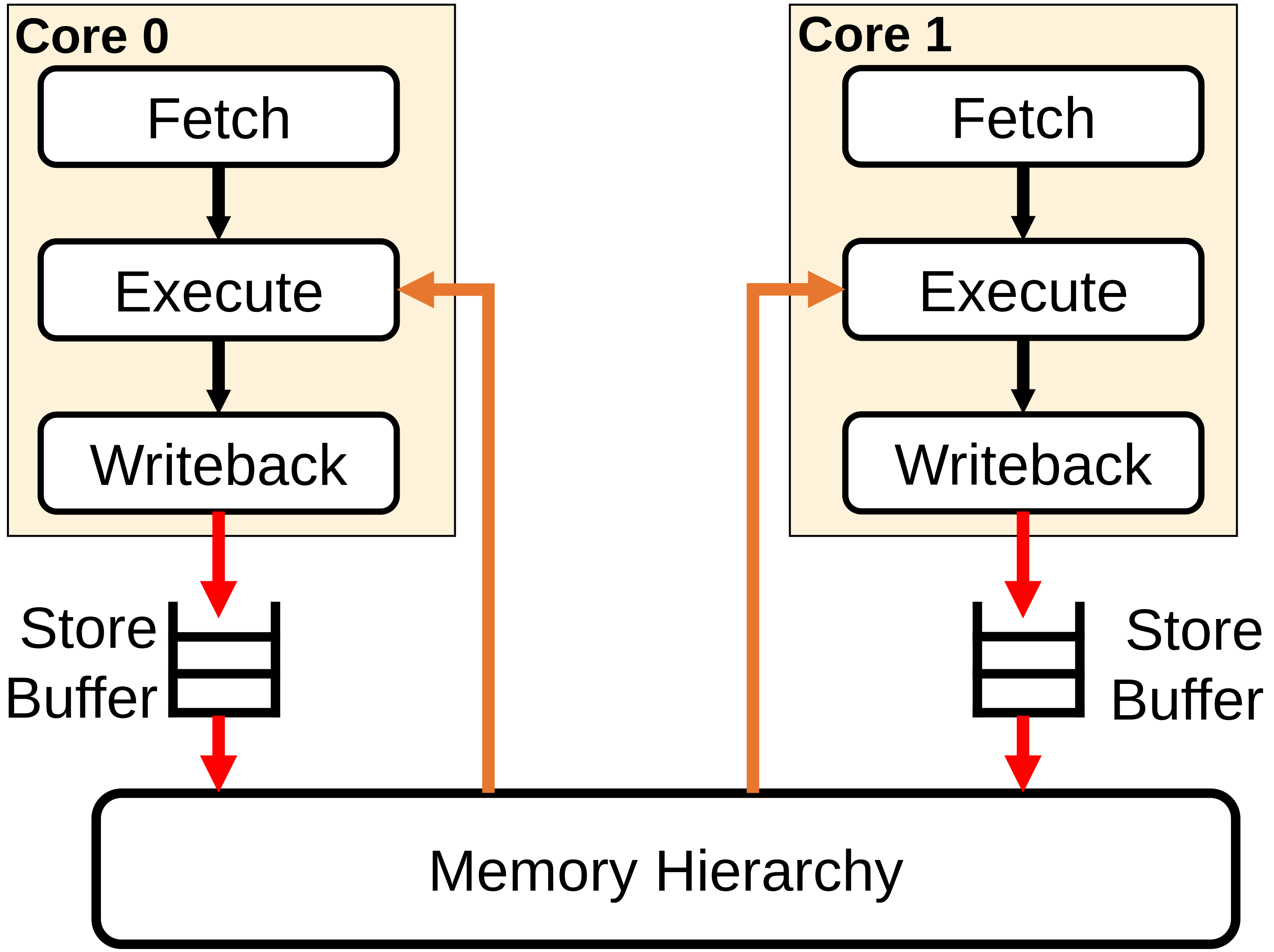
Hands-on: Moving from SC to TSO

- Reads must currently wait for prior writes to reach memory
  - **EnforceWritePPO** axiom
- Main motivation for TSO: store buffers to hide write latency
- Also want to allow reads to bypass value from store buffer (before value made visible to other cores)
  - Known as “read your own write early”
- **How to model this in μSpec?**

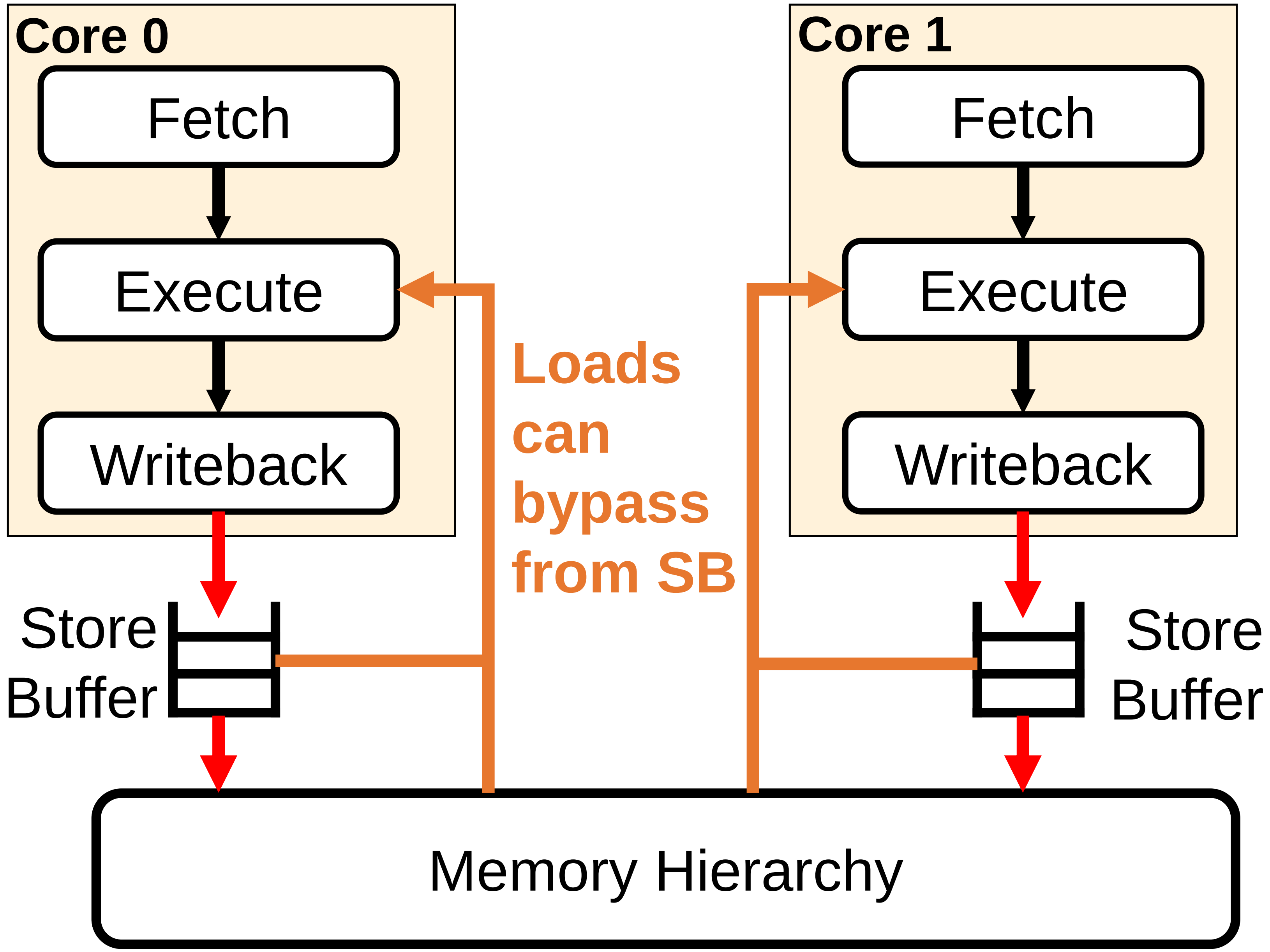
Moving from SC to TSO



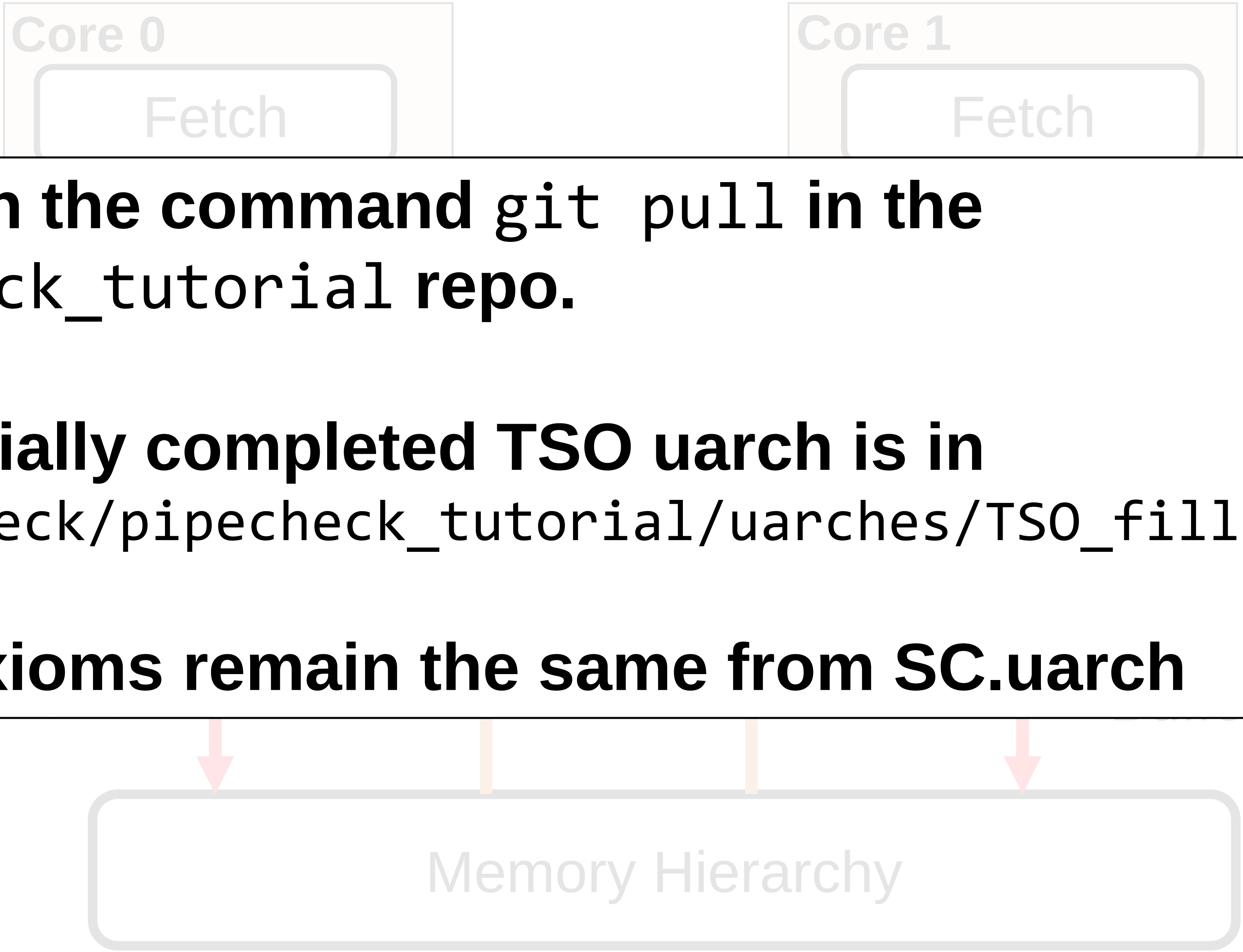
Moving from SC to TSO



Moving from SC to TSO



Moving from SC to TSO



First, run the command `git pull` **in the** `pipecheck_tutorial` **repo.**

The partially completed TSO uarch is in
`/home/check/pipecheck_tutorial/uarches/TSO_fillable.uarch`

Some axioms remain the same from SC.uarch

Hands-on: Moving from SC to TSO

- 8 changes needed to SC.uarch:
  1. **Add StoreBuffer and MemoryHierarchy stages**
  2. Split Instr_Path into Reads_Path and Writes_Path, writes go through SB
  3. Ensure that same-core writes go through SB in order
  4. Modify WriteSerialization and EnforceFinalWrite to enforce coherence order at MemoryHierarchy rather than Writeback
  5. Enforce that write is released from SB only after all prior same-core writes have reached memory
  6. Ensure that if load is reading from memory, that core’s store buffer has no entries for address of load (includes modifying Read_Values macros)
  7. (Advanced) Allow a core to read value of a write from its store buffer before write is made visible to other cores
  8. (Advanced) Implement fence operation that flushes all prior writes to memory before any succeeding instructions can perform

Add StoreBuffer and MemoryHierarchy Stages

- Solution:

```
StageName _ "____________".
StageName _ “________________".
```

Add StoreBuffer and MemoryHierarchy Stages

- Solution:

```
StageName 3 "StoreBuffer".
StageName 4 "MemoryHierarchy".
```

Hands-on: Moving from SC to TSO

- 8 changes needed to SC.uarch:
  1. Add StoreBuffer and MemoryHierarchy stages
  2. **Split Instr_Path into Reads_Path and Writes_Path, writes go through SB**
  3. Ensure that same-core writes go through SB in order
  4. Modify WriteSerialization and EnforceFinalWrite to enforce coherence order at MemoryHierarchy rather than Writeback
  5. Enforce that write is released from SB only after all prior same-core writes have reached memory
  6. Ensure that if load is reading from memory, that core's store buffer has no entries for address of load (includes modifying Read_Values macros)
  7. (Advanced) Allow a core to read value of a write from its store buffer before write is made visible to other cores
  8. (Advanced) Implement fence operation that flushes all prior writes to memory before any succeeding instructions can perform

Split Instr_Path into Reads_Path and Writes_Path

- Complete Writes_Path axiom so stores go WB → SB →MemHier
- Solution:

```
Axiom “Reads_Path":
...
Axiom “Writes_Path”:
forall microops "i",
IsAnyWrite i =>
AddEdges [((i, Fetch), (i, Execute),     "path");
          ((i, Execute), (i, Writeback),    "path");
          ((i, _________), (i, ___________),    "path");
          ((i, ___________), (i, _______________),
         "path")
          ].
```

Split Instr_Path into Reads_Path and Writes_Path

- Complete Writes_Path axiom so stores go WB → SB →MemHier
- Solution:

```
Axiom “Reads_Path":
...
Axiom “Writes_Path”:
forall microops "i",
IsAnyWrite i =>
AddEdges [((i, Fetch), (i, Execute),     "path");
          ((i, Execute), (i, Writeback),     "path");
          ((i, Writeback), (i, StoreBuffer),     "path");
          ((i, StoreBuffer), (i, MemoryHierarchy), "path")
          ].
```

Hands-on: Moving from SC to TSO

- 8 changes needed to SC.uarch:
  1. Add StoreBuffer and MemoryHierarchy stages
  2. Split Instr_Path into Reads_Path and Writes_Path, writes go through SB
  3. **Ensure that same-core writes go through SB in order**
  4. Modify WriteSerialization and EnforceFinalWrite to enforce coherence order at MemoryHierarchy rather than Writeback
  5. Enforce that write is released from SB only after all prior same-core writes have reached memory
  6. Ensure that if load is reading from memory, that core’s store buffer has no entries for address of load (includes modifying Read_Values macros)
  7. (Advanced) Allow a core to read value of a write from its store buffer before write is made visible to other cores
  8. (Advanced) Implement fence operation that flushes all prior writes to memory before any succeeding instructions can perform

Same-Core Writes Go Through SB in order

- If same-core writes go through WB in order, they should go through SB in order too
- Hint: Use Writeback_stage_is_in_order axiom as a starting point
- Solution:

```
Axiom "StoreBuffer_stage_is_in_order":
forall microops "i1",
forall microops "i2",
__________ i1 /\ __________ i2 /\ SameCore i1 i2 /\
EdgeExists ((i1, _________),  (i2, _________), "") =>
AddEdge ((i1, ___________), (i2, ___________), "PPO",
"darkgreen").
```

Same-Core Writes Go Through SB in order

- If same-core writes go through WB in order, they should go through SB in order too
- Hint: Use Writeback_stage_is_in_order axiom as a starting point
- Solution:

```
Axiom "StoreBuffer_stage_is_in_order":
forall microops "i1",
forall microops "i2",
IsAnyWrite i1 /\ IsAnyWrite i2 /\ SameCore i1 i2 /\
EdgeExists ((i1, Writeback), (i2, Writeback), "") =>
AddEdge ((i1, StoreBuffer), (i2, StoreBuffer), "PPO",
"darkgreen").
```

Hands-on: Moving from SC to TSO

- 8 changes needed to SC.uarch:
  1. Add StoreBuffer and MemoryHierarchy stages
  2. Split Instr_Path into Reads_Path and Writes_Path, writes go through SB
  3. Ensure that same-core writes go through SB in order
  4. **Modify WriteSerialization and EnforceFinalWrite to enforce coherence order at MemoryHierarchy rather than Writeback**
  5. Enforce that write is released from SB only after all prior same-core writes have reached memory
  6. Ensure that if load is reading from memory, that core's store buffer has no entries for address of load (includes modifying Read_Values macros)
  7. (Advanced) Allow a core to read value of a write from its store buffer before write is made visible to other cores
  8. (Advanced) Implement fence operation that flushes all prior writes to memory before any succeeding instructions can perform

Enforce Coherence Order at MemoryHierarchy

```
Axiom "WriteSerialization":
forall microops "i1",
forall microops "i2",
  ( ~(SameMicroop i1 i2) /\ IsAnyWrite i1 /\ IsAnyWrite i2 /\
     SamePhysicalAddress i1 i2) =>
  (EdgeExists ((i1, _______________), (i2, _______________), "ws”)
\/ EdgeExists ((i2, _______________), (i1, _______________), "ws")
  ).

Axiom "EnforceFinalWrite":
  forall microop "w",
  forall microop "w’”,
 IsAnyWrite w /\ IsAnyWrite w' /\ SamePhysicalAddress w w’
 /\ ~SameMicroop w w' /\ DataFromFinalStateAtPA w') =>
      AddEdge ((w, _______________), (w', _______________),
           "ws_final", "red").
```

Enforce Coherence Order at MemoryHierarchy

```
Axiom "WriteSerialization":
forall microops "i1",
forall microops "i2",
  ( ~(SameMicroop i1 i2) /\ IsAnyWrite i1 /\ IsAnyWrite i2 /\
     SamePhysicalAddress i1 i2) =>
  (EdgeExists ((i1, MemoryHierarchy), (i2, MemoryHierarchy), "ws”)
\/ EdgeExists ((i2, MemoryHierarchy), (i1, MemoryHierarchy), "ws")
  ).

Axiom "EnforceFinalWrite":
  forall microop "w",
  forall microop "w’”,
 IsAnyWrite w /\ IsAnyWrite w' /\ SamePhysicalAddress w w’
 /\ ~SameMicroop w w' /\ DataFromFinalStateAtPA w') =>
      AddEdge ((w, MemoryHierarchy), (w', MemoryHierarchy),
           "ws_final", "red").
```

Hands-on: Moving from SC to TSO

- 8 changes needed to SC.uarch:
  1. Add StoreBuffer and MemoryHierarchy stages
  2. Split Instr_Path into Reads_Path and Writes_Path, writes go through SB
  3. Ensure that same-core writes go through SB in order
  4. Modify WriteSerialization and EnforceFinalWrite to enforce coherence order at MemoryHierarchy rather than Writeback
  5. **Enforce that write is released from SB only after all prior same-core writes have reached memory**
  6. Ensure that if load is reading from memory, that core's store buffer has no entries for address of load (includes modifying Read_Values macros)
  7. (Advanced) Allow a core to read value of a write from its store buffer before write is made visible to other cores
  8. (Advanced) Implement fence operation that flushes all prior writes to memory before any succeeding instructions can perform

Same-Core Writes Reach Memory In Order

- For two same-core writes in program order, first write must reach memory before second can leave store buffer
- Solution:

```
Axiom "EnforceWriteOrdering":
  forall microop "w",
  forall microop "w'",
  (IsAnyWrite w /\ IsAnyWrite w' /\ SameCore w w’
   /\ EdgeExists((w, Fetch), (w', Fetch), "")) =>
    AddEdge ((w, _______________), (w', ___________),
     "EWO", "green").
```

Same-Core Writes Reach Memory In Order

- For two same-core writes in program order, first write must reach memory before second can leave store buffer
- Solution:

```
Axiom "EnforceWriteOrdering":
  forall microop "w",
  forall microop "w'",
  (IsAnyWrite w /\ IsAnyWrite w' /\ SameCore w w’
   /\ EdgeExists((w, Fetch), (w', Fetch), "")) =>
    AddEdge ((w, MemoryHierarchy), (w’, StoreBuffer),
     "EWO", "green").
```

Hands-on: Moving from SC to TSO

- 8 changes needed to SC.uarch:
  1. Add StoreBuffer and MemoryHierarchy stages
  2. Split Instr_Path into Reads_Path and Writes_Path, writes go through SB
  3. Ensure that same-core writes go through SB in order
  4. Modify WriteSerialization and EnforceFinalWrite to enforce coherence order at MemoryHierarchy rather than Writeback
  5. Enforce that write is released from SB only after all prior same-core writes have reached memory
  6. **Ensure that if load is reading from memory, that core's store buffer has no entries for address of load (includes modifying Read_Values macros)**
  7. (Advanced) Allow a core to read value of a write from its store buffer before write is made visible to other cores
  8. (Advanced) Implement fence operation that flushes all prior writes to memory before any succeeding instructions can perform

Only read from Mem if SB has no same addr writes

- Create a **macro** enforcing that all writes “w” before instr “i” in program order to address of “i” have reached mem before “i” **Execute**s
- Solution:

```
DefineMacro "STBEmpty":
  % Store buffer is empty for the address we want to read.
  forall microop "w", (
    (IsAnyWrite w /\ SamePhysicalAddress w i /\
     ProgramOrder w i) =>
      AddEdge ((w, _______________), (i, _______),
     "STBEmpty", "purple")).
```

Only read from Mem if SB has no same addr writes

- Create a **macro** enforcing that all writes “w” before instr “i” in program order to address of “i” have reached mem before “i” **Execute**s
- Solution:

```
DefineMacro "STBEmpty":
  % Store buffer is empty for the address we want to read.
  forall microop "w", (
    (IsAnyWrite w /\ SamePhysicalAddress w i /\
     ProgramOrder w i) =>
      AddEdge ((w, MemoryHierarchy), (i, Execute),
     "STBEmpty", "purple")).
```

Only read from Mem if SB has no same addr writes

- Modify Read_Values axiom’s macros to enforce orderings on writes with respect to MemoryHierarchy stage rather than Writeback stage

```
DefineMacro "BeforeAllWrites":
  % Read occurs before all writes to same PA & Data
  DataFromInitialStateAtPA i /\
  forall microop "w", (
    (IsAnyWrite w /\ SamePhysicalAddress w i /\ ~SameMicroop i w) =>
    AddEdge ((i, Execute), (w, _______________), "fr", "red")).

DefineMacro "Before_Or_After_Every_SameAddrWrite":
  % Either before or after every write to the same physical address
  forall microop "w", (
    (IsAnyWrite w /\ SamePhysicalAddress w i) =>
    (AddEdge ((w, _______________), (i, Execute), "wsrf", "crimson") \/
     AddEdge ((i, Execute), (w, _______________), "fr", "red"))).

DefineMacro "No_SameAddrWrites_Btwn_Src_And_Read":
  % Read from "w", and there must not exist any writes w' in between w and i
  exists microop "w", (
    IsAnyWrite w /\ SamePhysicalAddress w i /\ SameData w i /\
    AddEdge ((w, _______________), (i, Execute), "rf", "red") /\
    ~(exists microop "w'",
      IsAnyWrite w' /\ SamePhysicalAddress i w' /\ ~SameMicroop w w' /\
      EdgesExist [((w , _______________), (w', _______________), "");
                  ((w', _______________), (i, Execute), "")])).
```

```
DefineMacro "BeforeAllWrites":
  % Read occurs before all writes to same PA & Data
  DataFromInitialStateAtPA i /\
  forall microop "w", (
    (IsAnyWrite w /\ SamePhysicalAddress w i /\ ~SameMicroop i w) =>
    AddEdge ((i, Execute), (w, MemoryHierarchy), "fr", "red")).

DefineMacro "Before_Or_After_Every_SameAddrWrite":
  % Either before or after every write to the same physical address
  forall microop "w", (
    (IsAnyWrite w /\ SamePhysicalAddress w i) =>
    (AddEdge ((w, MemoryHierarchy), (i, Execute), "wsrf", "crimson") \/
     AddEdge ((i, Execute), (w, MemoryHierarchy), "fr", "red"))).

DefineMacro "No_SameAddrWrites_Btwn_Src_And_Read":
  % Read from "w", and there must not exist any writes w' in between w and i
  exists microop "w", (
    IsAnyWrite w /\ SamePhysicalAddress w i /\ SameData w i /\
    AddEdge ((w, MemoryHierarchy), (i, Execute), "rf", "red") /\
    ~(exists microop "w'",
      IsAnyWrite w' /\ SamePhysicalAddress i w' /\ ~SameMicroop w w' /\
      EdgesExist [((w , MemoryHierarchy), (w', MemoryHierarchy), "");
                  ((w', MemoryHierarchy), (i, Execute), "")])).
```

Only read from Mem if SB has no same addr writes

- Now expand the STBEmpty macro in the Read_Values axiom to ensure that SB has no entries for a load’s address if it is reading from memory

Only read from Mem if SB has no same addr writes

```
Axiom "Read_Values":
forall microops "i",
IsAnyRead i => (
% Uncomment the commented lines if you add the (advanced) store buff forwarding.
%  ExpandMacro ______ \/
%  (
       ExpandMacro ________ /\
       (
          ExpandMacro BeforeAllWrites
          \/
          (
            ExpandMacro No_SameAddrWrites_Btwn_Src_And_Read
            /\
            ExpandMacro Before_Or_After_Every_SameAddrWrite
          )
       )
%  )
).
```

Only read from Mem if SB has no same addr writes

```
Axiom "Read_Values":
forall microops "i",
IsAnyRead i => (
% Uncomment the commented lines if you add the (advanced) store buff forwarding.
%  ExpandMacro ______ \/
%  (
       ExpandMacro STBEmpty /\
       (
          ExpandMacro BeforeAllWrites
          \/
          (
            ExpandMacro No_SameAddrWrites_Btwn_Src_And_Read
            /\
            ExpandMacro Before_Or_After_Every_SameAddrWrite
          )
       )
%  )
).
```

Hands-on: Moving from SC to TSO

- 8 changes needed to SC.uarch:
  1. Add StoreBuffer and MemoryHierarchy stages
  2. Split Instr_Path into Reads_Path and Writes_Path, writes go through SB
  3. Ensure that same-core writes go through SB in order
  4. Modify WriteSerialization and EnforceFinalWrite to enforce coherence order at MemoryHierarchy rather than Writeback
  5. Enforce that write is released from SB only after all prior same-core writes have reached memory
  6. Ensure that if load is reading from memory, that core's store buffer has no entries for address of load (includes modifying Read_Values macros)
  7. **(Advanced) Allow a core to read value of a write from its store buffer before write is made visible to other cores**
  8. (Advanced) Implement fence operation that flushes all prior writes to memory before any succeeding instructions can perform

Forward Value from SB (Advanced)

- Create a **macro** that checks for a write on the same core to forward from (Execute stage -> Execute stage), and ensures the forwarding occurs **before** the write reaches memory
- Macro must also check that forwarding occurs from the latest write in program order (no intervening writes)
- Solution:

Forward Value from SB (Advanced)

```
DefineMacro "STBFwd":
  % Forward from the store buffer
  exists microop "w", (
    __________ w /\
    ________ w i /\
    ___________________ w i /\
    ________ w i /\
    AddEdges [((w, Execute), (i, Execute), "STBFwd", "red");
              ((i, Execute), (w, MemoryHierarchy), "STBFwd",
           "purple")]) /\
    % Ensure the STB entry is the latest one.
    ~exists microop "w'",
    __________ w' /\ ___________________ w w' /\
    ____________ w w' /\ ____________ w' i.
```

Forward Value from SB (Advanced)

```
DefineMacro "STBFwd":
  % Forward from the store buffer
  exists microop "w", (
    IsAnyWrite w /\
    SameCore w i /\
    SamePhysicalAddress w i /\
    SameData w i /\
    AddEdges [((w, Execute), (i, Execute), "STBFwd", "red");
              ((i, Execute), (w, MemoryHierarchy), "STBFwd",
           "purple")]) /\
    % Ensure the STB entry is the latest one.
    ~exists microop "w'",
    IsAnyWrite w' /\ SamePhysicalAddress w w' /\
    ProgramOrder w w' /\ ProgramOrder w' i.
```

Forward Value from SB (Advanced)

- Expand the macro in the Read_Values axiom so that forwarding from the SB is an *alternative* choice to reading from memory
- Solution:

```
Axiom "Read_Values":
forall microops "i",
IsAnyRead i =>
(
  ExpandMacro ______ \/
  (
    ExpandMacro STBEmpty /\
    (
      ExpandMacro BeforeAllWrites
      \/
      (
        ExpandMacro No_SameAddrWrites_Btwn_Src_And_Read
        /\
        ExpandMacro Before_Or_After_Every_SameAddrWrite
      )
    )
  )
).
```

```
Axiom "Read_Values":
forall microops "i",
IsAnyRead i =>
(
  ExpandMacro STBFwd \/
  (
    ExpandMacro STBEmpty /\
    (
      ExpandMacro BeforeAllWrites
      \/
      (
        ExpandMacro No_SameAddrWrites_Btwn_Src_And_Read
        /\
        ExpandMacro Before_Or_After_Every_SameAddrWrite
      )
    )
  )
).
```

Hands-on: Moving from SC to TSO

- 8 changes needed to SC.uarch:
  1. Add StoreBuffer and MemoryHierarchy stages
  2. Split Instr_Path into Reads_Path and Writes_Path, writes go through SB
  3. Ensure that same-core writes go through SB in order
  4. Modify WriteSerialization and EnforceFinalWrite to enforce coherence order at MemoryHierarchy rather than Writeback
  5. Enforce that write is released from SB only after all prior same-core writes have reached memory
  6. Ensure that if load is reading from memory, that core's store buffer has no entries for address of load (includes modifying Read_Values macros)
  7. (Advanced) Allow a core to read value of a write from its store buffer before write is made visible to other cores
  8. **(Advanced) Implement fence operation that flushes all prior writes to memory before any succeeding instructions can perform**

Fence Instruction Orders Write-Read pairs

- Add a fence instruction that flushes all prior writes in program order to memory before the fence's execute stage
- Solution:

Fence Instruction Orders Write-Read pairs

- Add a fence instruction that flushes all prior writes in program

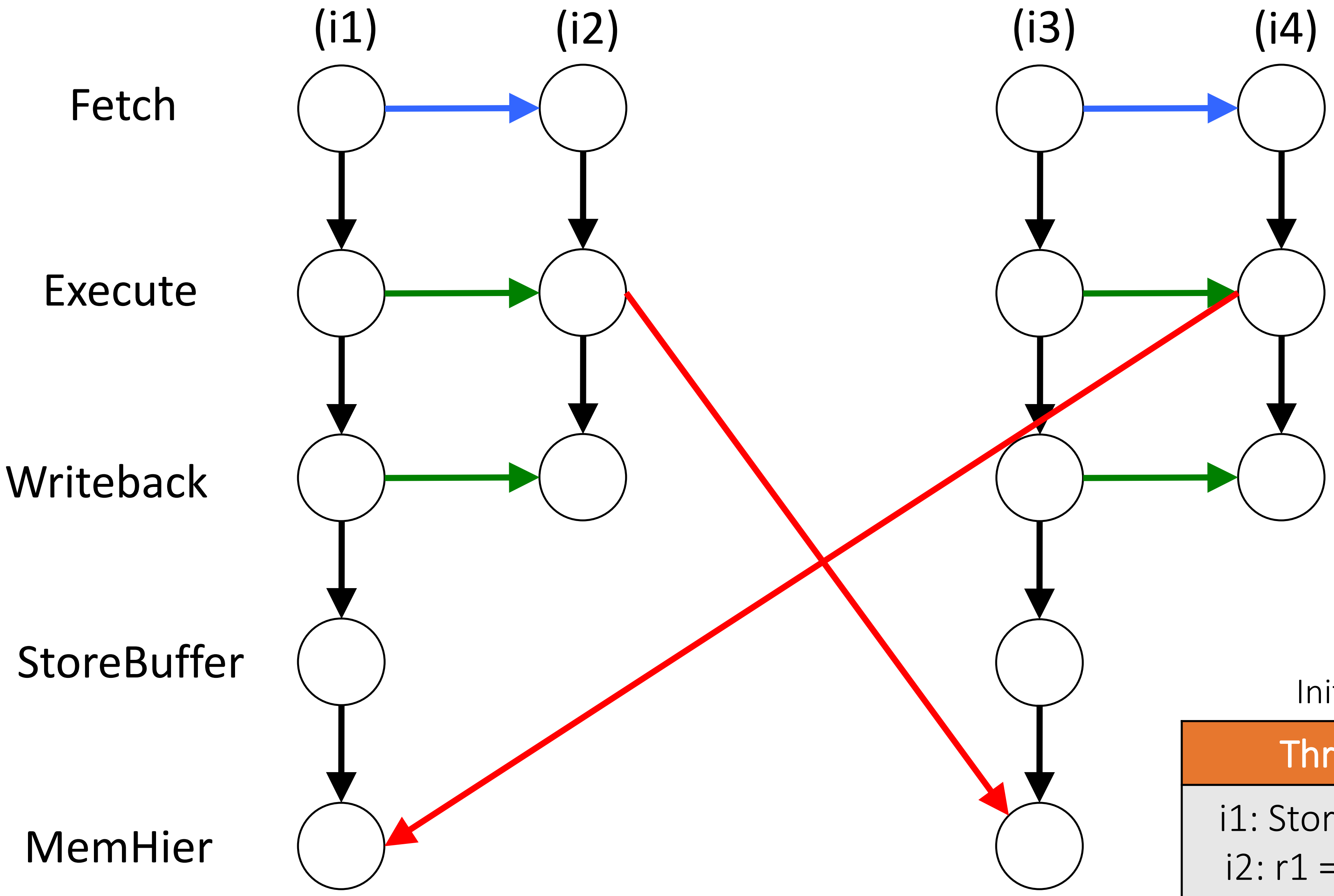
```
Axiom "Fence_Ordering":
forall microops "f",
IsAnyFence f =>
AddEdges [((f, Fetch),       (f, Execute),      "path");
          ((f, Execute),     (f, Writeback), "path")]
/\
(
  forall microops "w",
    (__________ w /\ ____________ w f) =>
      AddEdge ((w, _______________), (f, _______),
     "fence", "orange")
).
```

Fence Instruction Orders Write-Read pairs

- Add a fence instruction that flushes all prior writes in program

```
Axiom "Fence_Ordering":
forall microops "f",
IsAnyFence f =>
AddEdges [((f, Fetch),       (f, Execute),      "path");
          ((f, Execute),     (f, Writeback), "path")]
/\
(
  forall microops "w",
    (IsAnyWrite w /\ ProgramOrder w f) =>
      AddEdge ((w, MemoryHierarchy), (f, Execute),
     "fence", "orange")
).
```

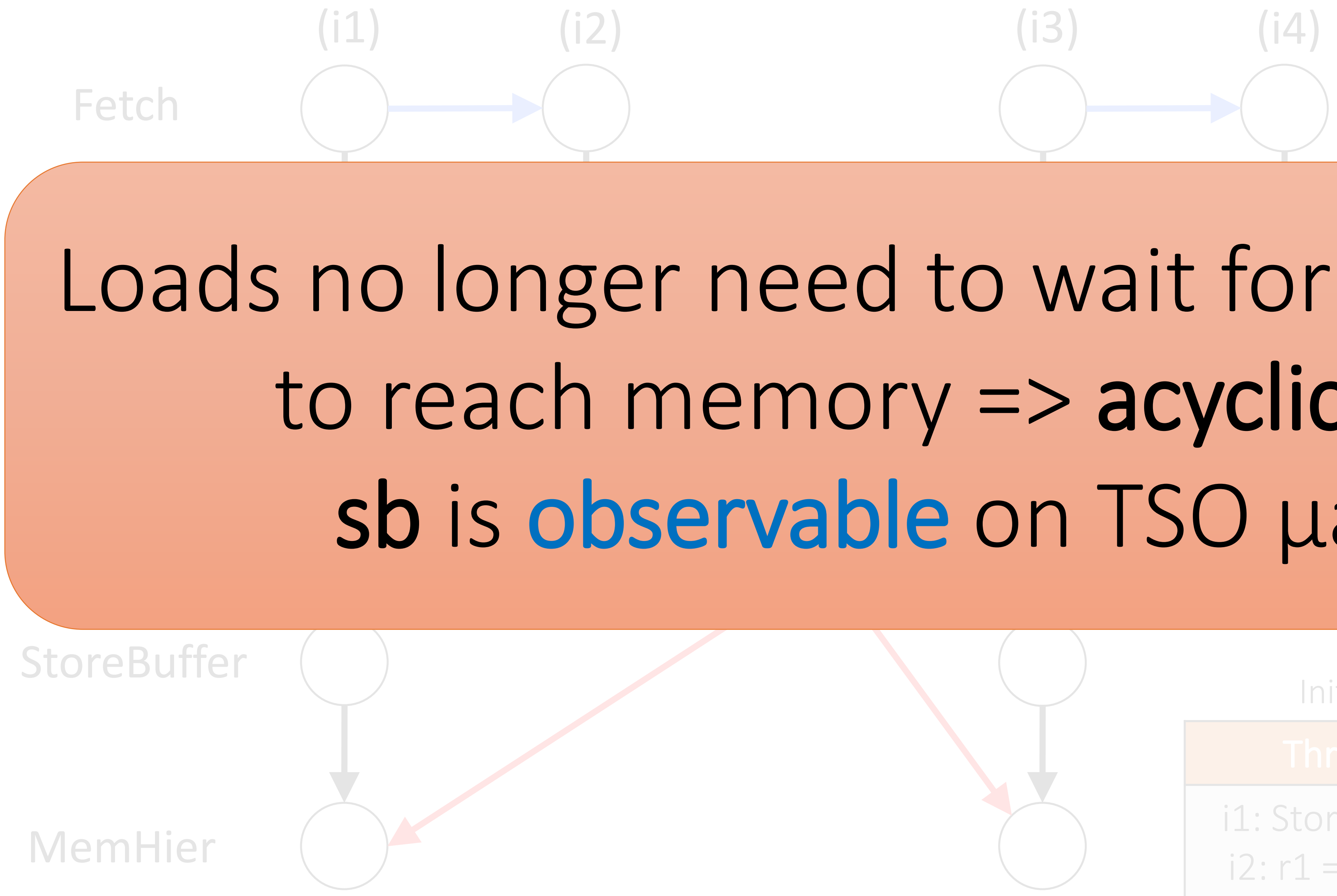
μhb Graph for sb On TSO μarch.



Initially, Mem[x] = Mem[y] = 0

Thread 0	Thread 1
i1: Store [x] ← 1 i2: r1 = Load [y]	i3: Store [y] ← 1 i4: r2 = Load [x]
SC Forbids: r1=0, r2=0	

μhb Graph for sb On TSO μarch.



Loads no longer need to wait for prior writes to reach memory => **acyclic graph**

sb is observable on TSO μarch!

Initially, Mem[x] = Mem[y] = 0

Thread 0	Thread 1
i1: Store [x] ← 1 i2: r1 = Load [y]	i3: Store [y] ← 1 i4: r2 = Load [x]

SC Forbids: r1=0, r2=0

Test your completed TSO uarch!

```
# Assuming you are in ~/pipecheck_tutorial/uarches/
$ check -i ../tests/TSO_tests/sb.test -m TSO_fillable.uarch

# If your uarch is valid, the above will create sb.pdf in your
# current directory (open pdfs from command line with evince)
# To run the solution version of the TSO uarch on this test:
# (Note: this will overwrite the sb.pdf in your current folder)
$ check -i ../tests/TSO_tests/sb.test -m TSO.uarch –d solutions/

# If you get an error (cannot parse uarch, ps2pdf crashes, etc),
# examine your syntax or ask for help.
# If the outcome is not observable (“Strict”), compare the graphs
# generated by the solution uarch to those of your uarch.

# To compare the uarches themselves:
$ diff TSO_fillable.uarch solutions/TSO.uarch
```

Run the entire suite of TSO litmus tests!

```
# Assuming you are in ~/pipecheck_tutorial/uarches/
$ run_tests –v 2 -t ../tests/TSO_tests/ -m TSO_fillable.uarch

# The above will generate *.gv files in ~/pipecheck_tutorial/out/
# for all TSO tests, and output overall statistics at the end. If
# the count for “Buggy” is non-zero, your uarch is faulty. Look for
the tests that output “BUG” to find out which tests fail.

# You can use gen_graph to convert gv files into PDFs:
$ gen_graph –i <test_gv_file>

# Compare your uarch with the solution TSO uarch using diff to find
# discrepancies:
$ diff TSO_fillable.uarch solutions/TSO.uarch
```