Transactions in Relaxed Memory Architectures

Brijesh Dongol       Radha Jagadeesan       James Riely
Transactional Memory

- Replace locks with transactions
- Well studied …
Transactional Memory

- Replace locks with transactions
- Well studied …
  - *Atomicity* = all or nothing
Transactional Memory

- Replace locks with transactions
- Well studied …
  - Atomicity = all or nothing
  - Committeds: What order?
    - Standard serializability: $\exists$ total order (arbitrary)
    - Strict serializability: $\exists$ total order respecting real-time order
    - Causal serializability: $\exists$ partial order respecting causality

... assuming memory is sequentially consistent (SC)
What about relaxed memory?
Transactional Memory

- Replace locks with transactions
- Well studied …
  - Atomicity = all or nothing
  - Committed: What order?
    - Standard serializability: $\exists$ total order (arbitrary)
    - Strict serializability: $\exists$ total order respecting real-time order
    - Causal serializability: $\exists$ partial order respecting causality
  - Aborted: Can affect client?
    - Yes: Opacity — Aborted must fit in committed order
    - No: TMS1, VWC, … — Intuition less clear

... assuming memory is sequentially consistent (SC)

What about relaxed memory?
Transactional Memory

- Replace locks with transactions
- Well studied …
  - Atomicity = all or nothing
  - Committeds: What order?
    - Standard serializability: ∃ total order (arbitrary)
    - Strict serializability: ∃ total order respecting real-time order
    - Causal serializability: ∃ partial order respecting causality
  - Aborteds: Can affect client?
    - Yes: Opacity — Aborteds must fit in committed order
    - No: TMS1, VWC, … — Intuition less clear
  - How does nontransactional code see transaction?
    - Atomically: Strong isolation
    - As individual operations: Weak isolation
Transactional Memory

- Replace locks with transactions
- Well studied …
  - *Atomicity* = all or nothing
  - Committeds: What order?
    - *Standard serializability*: $\exists$ total order (arbitrary)
    - *Strict* serializability: $\exists$ total order respecting real-time order
    - *Causal* serializability: $\exists$ partial order respecting causality
- Aborteds: Can affect client?
  - Yes: *Opacity* — Aborteds must fit in committed order
  - No: *TMS1, VWC*, … — Intuition less clear
- How does nontransactional code see transaction?
  - Atomically: *Strong isolation*
  - As individual operations: *Weak isolation*

- … assuming memory is *sequentially consistent* (SC)
- What about *relaxed* memory?
Transactional Memory … Relaxed

- Atomicity, as before
Transactional Memory … Relaxed

- Atomicity, as before
- Order for committeds?
  - Idea: Use order from underlying memory model
Transactional Memory … Relaxed

- Atomicity, as before
- Order for committeds?
  - Idea: Use order from underlying memory model
  - $\implies$ causal serializability
  - $\iff$ strict serializability
  - $\not\implies$ standard serializability, in general
    - Respects causality: ✓ us ✗ standard
    - Single total order: ✗ us ✓ standard
Transactional Memory ... Relaxed

- Atomicity, as before
- Order for committeds?
  - Idea: Use order from underlying memory model
  - ⇒ causal serializability
  - ⇐ strict serializability
  - ⇔ standard serializability, in general
    - Respects causality: ✓ us ❌ standard
    - Single total order: ❌ us ✓ standard
  - ⇒ standard serializability, for GHB models, e.g. TSO and ARMv8
    - Respects causality: ✓ us ❌ standard
    - Single total order: ✓ us ✓ standard

In paper: Observational serializability ⇒ causal & standard
Transactional Memory … Relaxed

- Atomicity, as before
- Order for committed transactions?
  - Idea: Use order from underlying memory model
  - $\Rightarrow$ causal serializability
  - $\Leftarrow$ strict serializability
  - $\not\Rightarrow$ standard serializability, in general
    - Respects causality: $\checkmark$ us $\times$ standard
    - Single total order: $\times$ us $\checkmark$ standard
  - $\Rightarrow$ standard serializability, for GHB models, e.g. TSO and ARMv8
    - Respects causality: $\checkmark$ us $\times$ standard
    - Single total order: $\checkmark$ us $\checkmark$ standard

In paper: *Observational* serializability $\Rightarrow$ causal & standard

- Aborted transactions: Can affect the client?
  - Natural formalization of opacity (Ignoring realtime)
  - New perspective on weaker conditions (TMS1, VWC, …)
Transactional Memory … Relaxed

- Atomicity, as before
- Order for committeds?
  - Idea: Use order from underlying memory model
  - ⇒ causal serializability
  - ⇐ strict serializability
  - ⇔ standard serializability, in general
    - Respects causality: ✓ us ✗ standard
    - Single total order: ✗ us ✓ standard
  - ⇒ standard serializability, for GHB models, e.g. TSO and ARMv8
    - Respects causality: ✓ us ✗ standard
    - Single total order: ✓ us ✓ standard

In paper: Observational serializability ⇒ causal & standard

- Aborteds: Can affect client?
  - Natural formalization of opacity (Ignoring realtime)
  - New perspective on weaker conditions (TMS1, VWC, …)

- Nontransactional code?
  - Natural formalization of isolated and relaxed
Herding Cats!

- Axiomatic model Alglave, Maranget and Tautschnig (AMT)
  - Unifying framework for TSO, Power, ARMv7, etc
Herding Cats!

- Axiomatic model Alglave, Maranget and Tautschnig (AMT)
  - Unifying framework for TSO, Power, ARMv7, etc
- Events labelled by \textit{action} (R_{x1}, W_{x1})
Herding Cats!

- Axiomatic model Alglave, Maranget and Tautschnig (AMT)
  - Unifying framework for TSO, Power, ARMv7, etc
- Events labelled by action (Rx1, Wx1)
- Relations over events, including
  - Program generated
    - Program order \[ Wx1 \xrightarrow{\text{po}} Wy1 \] (ML syntax) e.g., \( x:=1; y:=1 \)
Herding Cats!

- Axiomatic model Alglave, Maranget and Tautschnig (AMT)
  - Unifying framework for TSO, Power, ARMv7, etc
- Events labelled by action (Rx1, Wx1)
- Relations over events, including
  - Program generated
    - Program order
      - Program order: $Wx1 \xrightarrow{\text{po}} Wy1$  
        - e.g., $x:=1; y:=1$
    - Data dependency
      - Data dependency: $Rx1 \xrightarrow{\text{data}} Wy1$  
        - e.g., $y:=!x$
    - Address dependency
      - Address dependency: $Rxy \xrightarrow{\text{addr}} Wy1$  
        - e.g., $!x:=1$
    - Control dependency
      - Control dependency: $Rx1 \xrightarrow{\text{ctrl}} Wy1$  
        - e.g., if !x then $y:=1$
Herding Cats!

- Axiomatic model Alglave, Maranget and Tautschnig (AMT)
  - Unifying framework for TSO, Power, ARMv7, etc
- Events labelled by action (Rx1, WX1)
- Relations over events, including
  - Program generated (ML syntax)
    - Program order    \( WX1 \xrightarrow{\text{po}} WY1 \)  e.g., \( x:=1; y:=1 \)
    - Data dependency  \( RX1 \xrightarrow{\text{data}} WY1 \)  e.g., \( y:=!x \)
    - Address dependency \( Rxy \xrightarrow{\text{addr}} WY1 \)  e.g., \( !x:=1 \)
    - Control dependency \( RX1 \xrightarrow{\text{ctrl}} WY1 \)  e.g., if \(!x\) then \( y:=1 \)
  - Resolving nondeterminism
    - Reads-from     \( WX1 \rightarrow RX1 \)  e.g., \( x:=1 || y:=!x \)
Herding Cats!

- Axiomatic model Alglave, Maranget and Tautschnig (AMT)
  - Unifying framework for TSO, Power, ARMv7, etc

- Events labelled by action (Rx1, Wx1)

- Relations over events, including
  - Program generated
    - Program order $Wx1 \xrightarrow{po} Wy1$ e.g., $x:=1; y:=1$
    - Data dependency $Rx1 \xrightarrow{data} Wy1$ e.g., $y:=!x$
    - Address dependency $Rxy \xrightarrow{addr} Wy1$ e.g., $!x:=1$
    - Control dependency $Rx1 \xrightarrow{ctrl} Wy1$ e.g., if $!x$ then $y:=1$
  - Resolving nondeterminism
    - Reads-from $Wx1 \xrightarrow{rf} Rx1$ e.g., $x:=1||y:=!x$
    - From-read $Rx0 \xrightarrow{fr} Wx1$ e.g., $x:=1||y:=!x$
Herding Cats!

- Axiomatic model Alglave, Maranget and Tautschnig (AMT)
  - Unifying framework for TSO, Power, ARMv7, etc
- *Events* labelled by *action* ($Rx_1$, $Wx_1$)
- Relations over events, including
  - Program generated
    - Program order
      - $Wx_1$ \xrightarrow{po} $Wy_1$  
        e.g., $x:=1; y:=1$
    - Data dependency
      - $Rx_1$ \xrightarrow{data} $Wy_1$  
        e.g., $y:=!x$
    - Address dependency
      - $Rxy$ \xrightarrow{addr} $Wy_1$  
        e.g., $!x:=1$
    - Control dependency
      - $Rx_1$ \xrightarrow{ctrl} $Wy_1$  
        e.g., if $!x$ then $y:=1$
  - Resolving nondeterminism
    - Reads-from
      - $Wx_1$ \xrightarrow{rf} $Rx_1$  
        e.g., $x:=1||y:=!x$
    - From-read
      - $Rx_0$ \xrightarrow{fr} $Wx_1$  
        e.g., $x:=1||y:=!x$
    - Coherence
      - $Wx_1$ \xrightarrow{co} $Wx_2$  
        e.g., $x:=1||x:=2$
Herding Cats!

- Axiomatic model Alglave, Maranget and Tautschnig (AMT)
  - Unifying framework for TSO, Power, ARMv7, etc
- Events labelled by action (Rx1, Wx1)
- Relations over events, including
  - Program generated
    - Program order $Wx1 \xrightarrow{\text{po}} Wy1$ e.g., $x:=1; y:=1$
    - Data dependency $Rx1 \xrightarrow{\text{data}} Wy1$ e.g., $y:=!x$
    - Address dependency $Rxy \xrightarrow{\text{addr}} Wy1$ e.g., $!x:=1$
    - Control dependency $Rx1 \xrightarrow{\text{ctrl}} Wy1$ e.g., if !x then $y:=1$
  - Resolving nondeterminism
    - Reads-from $Wx1 \xrightarrow{\text{rf}} Rx1$ e.g., $x:=1||y:=!x$
    - From-read $Rx0 \xrightarrow{\text{fr}} Wx1$ e.g., $x:=1||y:=!x$
    - Coherence $Wx1 \xrightarrow{\text{co}} Wx2$ e.g., $x:=1||x:=2$
  - Architecture generated
    - Preserved program order For SC: $\text{ppo} = \text{po}$
      For TSO: $\text{ppo} = \text{po} \setminus \text{WR}$
Herding Cats!

- Axiomatic model Alglave, Maranget and Tautschnig (AMT)
  - Unifying framework for TSO, Power, ARMv7, etc
- Execution is valid if it satisfies certain acycliclicity requirements
Herding Cats!

- Axiomatic model Alglave, Maranget and Tautschnig (AMT)
- Unifying framework for TSO, Power, ARMv7, etc
- Execution is *valid* if it satisfies certain acyclic requirements
- Load buffering example: *Forbidden* under SC, where $ppo = po$

Initially: $x=y=0$

Thread 1: $x:=1; \text{ read } y$

Thread 2: $y:=1; \text{ read } x$
Herding Cats!

- Axiomatic model Alglave, Maranget and Tautschnig (AMT)
  - Unifying framework for TSO, Power, ARMv7, etc
- Execution is *valid* if it satisfies certain acyclicity requirements
- Load buffering example: *Forbidden* under SC, where $ppo = po$

Initially: $x=y=0$
Thread 1: $x:=1$; read $y$
Thread 2: $y:=1$; read $x$

- *Allowed* under TSO, where $ppo = po \setminus WR$
Herding Cats!

- Axiomatic model Alglave, Maranget and Tautschnig (AMT)
  - Unifying framework for TSO, Power, ARMv7, etc
- Execution is *valid* if it satisfies certain acyclicity requirements
- Load buffering example: *Forbidden* under SC, where $ppo = po$

Initially: $x=y=0$

Thread 1: $x:=1; FF; read\ y$
Thread 2: $y:=1; FF; read\ x$

- *Allowed* under TSO, where $ppo = po \setminus WR$
- To get a cycle under TSO, **add fences**
A simple idea

- Load buffering example: **Allowed** under TSO

Initially: $x=y=0$

- Thread 1: $x:=1;\text{read } y$
- Thread 2: $y:=1;\text{read } x$

Transaction shown as boxes

To achieve atomicity, **Li/f_t** relations across transactions

Independent discovery by Chong, Sorensen and Wickerson

Not AMT valid: Cycle appears between the reads

Consequences:

- AMT valid $\Rightarrow$ acyclicity $\Rightarrow$ causal serializability

Ignores real time

Erase empty transactions, singletons

Li/f_t includes nontransactional $\Rightarrow$ strong isolation
A simple idea

- Load buffering example: **Allowed** under TSO, without atomics

Initially: \( x=y=0 \)
Thread 1: \( \text{atomic}\{x:=1;\text{read } y}\} 
Thread 2: \( \text{atomic}\{y:=1;\text{read } x}\} 

- Transaction shown as boxes
A simple idea

- Load buffering example: **Allowed** under TSO, without atomics

Initially: $x = y = 0$
Thread 1: atomic{$x:= 1; \text{read } y$}
Thread 2: atomic{$y:= 1; \text{read } x$}

- Transaction shown as boxes
- To achieve atomicity, *lift* relations across transactions
  - Independent discovery by Chong, Sorensen and Wickerson
A simple idea

- Load buffering example: **Allowed** under TSO, without atomics

Initially: \(x=y=0\)

Thread 1: `atomic{x:=1;read y}`

Thread 2: `atomic{y:=1;read x}`

- Transaction shown as boxes

- To achieve atomicity, *lift* relations across transactions
  - Independent discovery by Chong, Sorensen and Wickerson
  - Not AMT valid: **Cycle** appears between the reads
A simple idea

▶ Load buffering example: **Allowed** under TSO, without atomics

Initially: \( x=y=0 \)
- Thread 1: atomic\{\( x:=1; \)read \( y \)\}
- Thread 2: atomic\{\( y:=1; \)read \( x \)\}

▶ Transaction shown as boxes
▶ To achieve atomicity, *lift* relations across transactions
  ▶ Independent discovery by Chong, Sorensen and Wickerson
  ▶ Not AMT valid: **Cycle** appears between the reads

▶ Consequences:
  ▶ AMT valid \(\implies\) acyclicity \(\implies\) Causal serializability
A simple idea

▶ Load buffering example: **Allowed** under TSO, without atomics

Initially: \( x = y = 0 \)
Thread 1: \( \text{atomic}\{x:=1;\text{read } y\} \)
Thread 2: \( \text{atomic}\{y:=1;\text{read } x\} \)

▶ Transaction shown as boxes
▶ To achieve atomicity, *lift* relations across transactions
  ▶ Independent discovery by Chong, Sorensen and Wickerson
  ▶ Not AMT valid: **Cycle** appears between the reads

▶ Consequences:
  ▶ AMT valid \( \Rightarrow \) acyclicity \( \Rightarrow \) Causal serializability
  ▶ Ignores real time
A simple idea

- Load buffering example: **Allowed** under TSO, without atomics

Initially: x=y=0
Thread 1: atomic{x:=1;read y}
Thread 2: atomic{y:=1;read x}

- Transaction shown as boxes
- To achieve atomicity, *lift* relations across transactions
  - Independent discovery by Chong, Sorensen and Wickerson
  - Not AMT valid: *Cycle* appears between the reads

- Consequences:
  - AMT valid ⇒ acyclicity ⇒ Causal serializability
  - Ignores real time
  - Erase empty transactions, singletons
A simple idea

- Load buffering example: **Allowed** under TSO, without atomics

  Initially: \(x=y=0\)
  Thread 1: \(\text{atomic}\{x:=1; \text{read } y\}\)
  Thread 2: \(\text{atomic}\{y:=1; \text{read } x\}\)

- Transaction shown as boxes
- To achieve atomicity, *lift* relations across transactions
  - Independent discovery by Chong, Sorensen and Wickerson
  - Not AMT valid: **Cycle** appears between the reads

- Consequences:
  - AMT valid \(\Rightarrow\) acyclicity \(\Rightarrow\) Causal serializability
  - Ignores real time
  - Erase empty transactions, singletons
  - Lift includes nontransactional \(\Rightarrow\) Strong isolation
Some goals

- Nested transactions
- Weak isolation (Example under TSO)

![Diagram 1: Wx1 po Wx2 R x1]

- Abort models (Example under TSO)

![Diagram 2: init Wx2 Rx0 Wy1 Ry0]

\[\text{\(\times\) strong} \quad \text{\(\checkmark\) weak}\]

\[\text{\(\times\) opaque} \quad \text{\(\checkmark\) non-opaque}\]
Definition

- Execution is **correct** if AMT valid with **lifted** relations

- $e \xrightarrow{\text{lift}(o)} d$ when either
  1. $e \xrightarrow{o} d$
  2. or $e' \xrightarrow{o} d$ for some $e' \in \text{trans}(e)$, $d \notin \text{trans}(e)$
  3. or symmetrically for $d'$

- Refinements:
  - **Nesting:** $e'$ in same or sub-transaction of $e$
  - **Weak isolated not seen atomically**
  - **Opacity:** aborteds ordered w.r.t. commi/t_teds

- **Consequences:** Causal serializability, No real time, Singletons

- What about standard serializability?
Definition

- Execution is **correct** if AMT valid with **lifted** relations

- $e \xrightarrow{\text{lift(o)}} d$ when either
  1. $e \xrightarrow{o} d$
  2. or $e' \xrightarrow{o} d$ for some $e' \in \text{descend}(e)$, $d \notin \text{descend}(e)$
  3. or symmetrically for $d'$

- Refinements:
  - Nesting: $e'$ in same or sub-transaction of $e$
Definition

- Execution is **correct** if AMT valid with **lifted** relations

- \( e \xrightarrow{\text{lift(o)}} d \) when either
  1. \( e \xrightarrow{o} d \)
  2. or \( e' \xrightarrow{o} d \) for some \( e' \in \text{descend}(e) \), \( d \notin \text{descend}(e) \), \( e' \in \text{StrongIsolated} \)
  3. or symmetrically for \( d' \)

- Refinements:
  - Nesting: \( e' \) in same or sub-transaction of \( e \)
  - Weak isolated not seen atomically

- Opacity: aborteds ordered w.r.t. commi/t_teds \( \Rightarrow \) No changed to li/f_t
- Aborteds only affect aborteds: \( \text{rwdep} = \text{rf} \cup \text{data} \cup \text{addr} \cup \text{ctrl} \)
- Consequences: Causal serializability, No real time, Singletons
- What about standard serializability?
Definition

- Execution is **correct** if AMT valid with **lifted** relations

  - $e \xrightarrow{\text{lift(o)}} d$ when either
    1. $e \xrightarrow{o} d$
    2. or $e' \xrightarrow{o} d$ for some $e' \in \text{descend}(e)$, $d \notin \text{descend}(e)$, either $e' \in \text{StrongIsolated}$ or $d \in \text{Transactional}$
    3. or symmetrically for $d'$

- Refinements:
  - Nesting: $e'$ in same or sub-transaction of $e$
  - Weak isolated not seen atomically, except by transactions
Definition

- Execution is **correct** if AMT valid with **lifted** relations

- \( e \xrightarrow{\text{lift}(o)} d \) when either
  1. \( e \xrightarrow{o} d \)
  2. or \( e' \xrightarrow{o} d \) for some \( e' \in \text{descend}(e) \), \( d \notin \text{descend}(e) \),
     either \( e' \in \text{StrongIsolated} \) or \( d \in \text{Transactional} \)
  3. or symmetrically for \( d' \)

- Refinements:
  - Nesting: \( e' \) in same or sub-transaction of \( e \)
  - Weak isolated not seen atomically, except by transactions
  - Opacity: aborteds ordered w.r.t. committeds \( \Rightarrow \) No changed to lift
Definition

- Execution is correct if AMT valid with lifted relations and 
  \( \forall d \in \text{Aborted}. \forall e \in E. d \xrightarrow{\text{li/f_t}} e \) implies \( e \in \text{Aborted} \)

- \( e \xrightarrow{\text{lift(o)}} d \) when either
  1. \( e \xrightarrow{o} d \)
  2. or \( e' \xrightarrow{o} d \) for some \( e' \in \text{descend}(e) \), \( d \notin \text{descend}(e) \),
     either \( e' \in \text{StrongIsolated} \) or \( d \in \text{Transactional} \)
  3. or symmetrically for \( d' \)

- Refinements:
  - Nesting: \( e' \) in same or sub-transaction of \( e \)
  - Weak isolated not seen atomically, except by transactions
  - Opacity: aborteds ordered w.r.t. committeds \( \Rightarrow \) No changed to lift
    Aborteds only affect aborteds
Definition

Execution is **correct** if AMT valid with **lifted** relations and
\[ \forall d \in \text{Aborted}. \forall e \in E. \ d \xrightarrow{\text{rwdep}} e \text{ implies } e \in \text{Aborted} \]

- \( e \xrightarrow{\text{lift}(o)} d \) when either
  1. \( e \xrightarrow{o} d \)
  2. or \( e' \xrightarrow{o} d \) for some \( e' \in \text{descend}(e) \), \( d \notin \text{descend}(e) \),
     either \( e' \in \text{StrongIsolated} \) or \( d \in \text{Transactional} \)
  3. or symmetrically for \( d' \)

**Refinements:**
- **Nesting:** \( e' \) in same or sub-transaction of \( e \)
- **Weak isolated** not seen atomically, except by transactions
- **Opacity:** aborteds ordered w.r.t. committeds \( \Rightarrow \) No changed to lift
  Aborteds only affect aborteds: \( \text{rwdep} = \text{rf} \cup \text{data} \cup \text{addr} \cup \text{ctrl} \)
Definition

- Execution is **correct** if AMT valid with **lifted** relations and
  \[ \forall d \in \text{Aborted.} \forall e \in E. \ d \xrightarrow{\text{rwdep}} e \implies e \in \text{Aborted} \]

- \[ e \xrightarrow{\text{lift}(\circ)} d \] when either
  1. \[ e \xrightarrow{\circ} d \]
  2. or \[ e' \xrightarrow{\circ} d \] for some \[ e' \in \text{descend}(e) \], \[ d \notin \text{descend}(e) \],
     either \[ e' \in \text{StrongIsolated} \] or \[ d \in \text{Transactional} \]
  3. or symmetrically for \[ d' \]

- **Refinements:**
  - **Nesting:** \[ e' \] in same or sub-transaction of \[ e \]
  - **Weak isolated not seen atomically, except by transactions**
  - **Opacity:** aborteds ordered w.r.t. committeds \[ \Rightarrow \] No changed to lift
    Aborteds only affect aborteds: \[ \text{rwdep} = \text{rf} \cup \text{data} \cup \text{addr} \cup \text{ctrl} \]

- **Consequences:** Causal serializability, No real time, Singletons
Definition

- Execution is **correct** if AMT valid with **lifted** relations and 
  \( \forall d \in \text{Aborted}. \forall e \in E. d \xrightarrow{\text{rwdep}} e \) implies \( e \in \text{Aborted} \)

- \( e \xrightarrow{\text{lift}(o)} d \) when either
  1. \( e \xrightarrow{o} d \)
  2. or \( e' \xrightarrow{o} d \) for some \( e' \in \text{descend}(e) \), \( d \notin \text{descend}(e) \), either \( e' \in \text{StrongIsolated} \) or \( d \in \text{Transactional} \)
  3. or symmetrically for \( d' \)

- **Refinements:**
  - Nesting: \( e' \) in same or sub-transaction of \( e \)
  - Weak isolated not seen atomically, except by transactions
  - Opacity: aborteds ordered w.r.t. committeds \( \Rightarrow \) No changed to lift
    Aborteds only affect aborteds: \( \text{rwdep} = \text{rf} \cup \text{data} \cup \text{addr} \cup \text{ctrl} \)

- **Consequences:** Causal serializability, No real time, Singletons

- What about standard serializability?
Standard Serializability?

- Independent Reads of Independent Writes (IRIW)
  Forbidden for *Multi-copy atomic*, e.g. SC, TSO, ARMv8

Diagram:
- **Wx1** → **Rx1** → **Ry0**
- **Wy1** → **Rx0** → **Ry1**
- **init** → **Wx1** → **Wy1**

Formalized using Global Happens Before [Alglave 2010]
Standard Serializability?

- Independent Reads of Independent Writes (IRIW)
  Forbidden for *Multi-copy atomic*, e.g. SC, TSO, ARMv8

- Allowed under ARMv7: Writes seen in different orders
Standard Serializability?

- Independent Reads of Independent Writes (IRIW)
  Forbidden for *Multi-copy atomic*, e.g. SC, TSO, ARMv8

- Allowed under ARMv7: Writes seen in different orders
- With transactions: ✓ causal serializable ✗ serializable
  Lift ⇒ Standard serializability, in general
Standard Serializability?

- Independent Reads of Independent Writes (IRIW)
  Forbidden for *Multi-copy atomic*, e.g. SC, TSO, ARMv8

- Allowed under ARMv7: Writes seen in different orders
- With transactions: ✔ causal serializable  ❌ serializable
  Lift ⇒ Standard serializability, in general
- Lift ⇒ Standard serializability, for multi-copy atomic

Formalized using *Global Happens Before* [Alglave 2010]
NonOpaque Aborts

- Forbidden if all commit (Example under TSO)
NonOpaque Aborts

- Forbidden if all commit (Example under TSO)

- What if bottom transaction aborts?
NonOpaque Aborts

- Forbidden if all commit (Example under TSO)

What if bottom transaction aborts?
- Forbidden under opacity: Aborteds ordered w.r.t. committeds
- Allowed under weaker conditions, e.g. VWC (and possibly TMS1)
NonOpaque Aborts

▷ Forbidden if all commit (Example under TSO)

What if bottom transaction aborts?
▷ Forbidden under opacity: Aborteds ordered w.r.t. committeds
▷ Allowed under weaker conditions, e.g. VWC (and possibly TMS1)

Our solution:
▷ Check committeds and opaques together, ignoring non-opaques
NonOpaque Aborts

- Forbidden if all commit (Example under TSO)

- What if bottom transaction aborts?
  - Forbidden under opacity: Aborteds ordered w.r.t. committeds
  - Allowed under weaker conditions, e.g. VWC (and possibly TMS1)

- Our solution:
  - Check committeds and opaques together, ignoring non-opaques
  - Check each non-opaque w.r.t. its causal history
NonOpaque Aborts

- Forbidden if all commit (Example under TSO)

- What if bottom transaction aborts?
  - Forbidden under opacity: Aborteds ordered w.r.t. committeds
  - Allowed under weaker conditions, e.g. VWC (and possibly TMS1)

- Our solution:
  - Check committeds and opaques together, ignoring non-opaques
  - Check each non-opaque w.r.t. its causal history

- New formal footing for weaker conditions, e.g. VWC and TMS1
In the paper

- Non-Opaques: Comparison with VWC and TMS1
In the paper

- Non-Opaques: Comparison with VWC and TMS1
- Automaton to check violations of Global Happens Before
  Used to prove lift \(\Rightarrow\) total order on transactions (for GHB)
In the paper

- Non-Opaques: Comparison with VWC and TMS1
- Automaton to check violations of Global Happens Before
  Used to prove lift $\Rightarrow$ total order on transactions (for GHB)
- Formalized in Memalloy [Wickerson, et al 2017]
  - TSO, Power and ARMv8 using non-opaque aborts
  - Compared to HW transactions ($\leq 5$ events)
In the paper

- Non-Opaques: Comparison with VWC and TMS1
- Automaton to check violations of Global Happens Before
  Used to prove lift $\Rightarrow$ total order on transactions (for GHB)
- Formalized in *Memalloy* [Wickerson, et al 2017]
  - TSO, Power and ARMv8 using non-opaque aborts
  - Compared to HW transactions ($\leq 5$ events)
  - HW hides aborted from different aborted

**HW allows**

**We allow**
In the paper

- Non-Opaques: Comparison with VWC and TMS1
- Automaton to check violations of Global Happens Before
  Used to prove $\text{li/f}_t \Rightarrow \text{total order on transactions}$ (for GHB)
- Formalized in *Memalloy* [Wickerson, et al 2017]
  - TSO, Power and ARMv8 using non-opaque aborts
  - Compared to HW transactions ($\leq 5$ events)
  - HW hides aborted from different aborted
  - Otherwise, our model strictly more expressive
    - HW enforces coherence with aborted
    - HW places fences before/after each transaction

HW allows

- $E_0: W[x]=1$
- $E_1: R[x]=1$
- $E_2: R[x]=0$

We allow

- $E_0: W[x]=1$
- $E_1: R[x]=1$
- $E_0: W[x]=2$
- $E_1: W[x]=1$
- $E_2: R[x]=0$
- $E_3: R[y]=0$
- $E_0: W[y]=1$
- $E_1: W[y]=1$
- $E_2: R[x]=0$

We allow

- $E_0: W[y]=0$
- $E_1: W[y]=1$
- $E_2: R[x]=0$

We allow

- $E_0: W[y]=0$
- $E_1: W[y]=1$
- $E_2: R[x]=0$
Inspiration

▶ *What do High-Level Memory Models Mean for Transactions?*
  Grossman, Manson and Pugh, 2006

▶ *Transactions As the Foundation of a Memory Consistency Model*
  Dalessandro, Scott and Spear, 2010
Inspiration

- **What do High-Level Memory Models Mean for Transactions?**
  Grossman, Manson and Pugh, 2006

- **Transactions As the Foundation of a Memory Consistency Model**
  Dalessandro, Scott and Spear, 2010

- **A Shared Memory Poetics**
  Alglave, 2010

- **Herding Cats: Modeling, Simulation, Testing, and Data Mining …**
  Alglave, Maranget and Tautschnig, 2014
Inspiration

- *What do High-Level Memory Models Mean for Transactions?*  
  Grossman, Manson and Pugh, 2006

- *Transactions As the Foundation of a Memory Consistency Model*  
  Dalessandro, Scott and Spear, 2010

- *A Shared Memory Poetics*  
  Alglave, 2010

- *Herding Cats: Modeling, Simulation, Testing, and Data Mining …*  
  Alglave, Maranget and Tautschnig, 2014

- *Automatically comparing memory consistency models,*  
  Wickerson, Batty, Sorensen and Constantinides, 2017

- *The Semantics of Transactions … in x86, Power, ARMv8, and C++*  
  Chong, Sorensen and Wickerson, 2017
Inspiration

- What do High-Level Memory Models Mean for Transactions? Grossman, Manson and Pugh, 2006
- Transactions As the Foundation of a Memory Consistency Model Dalessandro, Scott and Spear, 2010
- A Shared Memory Poetics Alglave, 2010
- Herding Cats: Modeling, Simulation, Testing, and Data Mining … Alglave, Maranget and Tautschnig, 2014
- Automatically comparing memory consistency models, Wickerson, Batty, Sorensen and Constantinides, 2017
- The Semantics of Transactions … in x86, Power, ARMv8, and C++ Chong, Sorensen and Wickerson, 2017
- Our contribution: High-level view of low-level model