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

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

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*: ∃ 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*
- … 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
  - $\Rightarrow$ causal serializability
  - $\Leftarrow$ strict serializability
  - $\not\Leftrightarrow$ 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
  - $\Rightarrow$ causal serializability
  - $\Leftarrow$ strict serializability
  - $\not\Leftrightarrow$ standard serializability, in general

Respects causality:	✓ us	✗ standard
Single total order:	✗ us	✓ standard

  - $\Rightarrow$ standard serializability, for *GHB* models, e.g. TSO and ARMv8

Respects causality:	✓ us	✗ standard
Single total order:	✓ us	✓ standard

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

Transactional Memory … Relaxed

- Atomicity, as before
- Order for committeds?
  - Idea: Use order from underlying memory model
  - $\Rightarrow$ causal serializability
  - $\Leftarrow$ strict serializability
  - $\not\Leftarrow$ standard serializability, in general

Respects causality:	✓ us	✗ standard
Single total order:	✗ us	✓ standard

  - $\Rightarrow$ standard serializability, for *GHB* models, e.g. TSO and ARMv8

Respects causality:	✓ us	✗ standard
Single total order:	✓ us	✓ standard

  In paper: *Observational* serializability $\Rightarrow$ causal & standard
- Aborteds: Can affect 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
  - $\Rightarrow$ causal serializability
  - $\Leftarrow$ strict serializability
  - $\not\Leftarrow$ standard serializability, in general

Respects causality:	✓ us	✗ standard
Single total order:	✗ us	✓ standard

  - $\Rightarrow$ standard serializability, for *GHB* models, e.g. TSO and ARMv8

Respects causality:	✓ us	✗ standard
Single total order:	✓ us	✓ standard

  In paper: *Observational* serializability $\Rightarrow$ 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 *action* (Rx1, Wx1)

Herding Cats!

- Axiomatic model Alglave, Maranget and Tautschnig (AMT)
  - Unifying framework for TSO, Power, ARMv7, etc
- *Events* labelled by *action* ($\mathrm{R}x1$, $\mathrm{W}x1$)
- Relations over events, including
  - Program generated (ML syntax)
    - Program order $\mathrm{W}x1 \xrightarrow{\text{po}} \mathrm{W}y1$ 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 (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`

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 $\xrightarrow{\text{rf}}$ 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 (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 $\xrightarrow{\text{rf}}$ Rx1 e.g., `x:=1`$\|$`y:=!x`
    - From-read Rx0 $\xrightarrow{\text{fr}}$ 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* ($\mathrm{R}x1$, $\mathrm{W}x1$)
- Relations over events, including
  - Program generated (ML syntax)
    - Program order $\mathrm{W}x1 \xrightarrow{\text{po}} \mathrm{W}y1$ e.g., `x:=1;y:=1`
    - Data dependency $\mathrm{R}x1 \xrightarrow{\text{data}} \mathrm{W}y1$ e.g., `y:=!x`
    - Address dependency $\mathrm{R}xy \xrightarrow{\text{addr}} \mathrm{W}y1$ e.g., `!x:=1`
    - Control dependency $\mathrm{R}x1 \xrightarrow{\text{ctrl}} \mathrm{W}y1$ e.g., `if !x then y:=1`
  - Resolving nondeterminism
    - Reads-from $\mathrm{W}x1 \xrightarrow{\text{rf}} \mathrm{R}x1$ e.g., `x:=1`$\|$`y:=!x`
    - From-read $\mathrm{R}x0 \xrightarrow{\text{fr}} \mathrm{W}x1$ e.g., `x:=1`$\|$`y:=!x`
    - Coherence $\mathrm{W}x1 \xrightarrow{\text{co}} \mathrm{W}x2$ 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* ($\mathrm{R}x1$, $\mathrm{W}x1$)
- Relations over events, including
  - Program generated (ML syntax)
    - Program order $\mathrm{W}x1 \xrightarrow{\text{po}} \mathrm{W}y1$ e.g., `x:=1;y:=1`
    - Data dependency $\mathrm{R}x1 \xrightarrow{\text{data}} \mathrm{W}y1$ e.g., `y:=!x`
    - Address dependency $\mathrm{R}xy \xrightarrow{\text{addr}} \mathrm{W}y1$ e.g., `!x:=1`
    - Control dependency $\mathrm{R}x1 \xrightarrow{\text{ctrl}} \mathrm{W}y1$ e.g., `if !x then y:=1`
  - Resolving nondeterminism
    - Reads-from $\mathrm{W}x1 \xrightarrow{\text{rf}} \mathrm{R}x1$ e.g., `x:=1`‖`y:=!x`
    - From-read $\mathrm{R}x0 \xrightarrow{\text{fr}} \mathrm{W}x1$ e.g., `x:=1`‖`y:=!x`
    - Coherence $\mathrm{W}x1 \xrightarrow{\text{co}} \mathrm{W}x2$ 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 acyclicity requirements

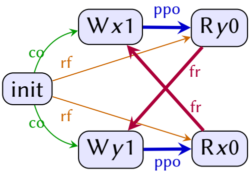
Herding Cats!

- 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;`

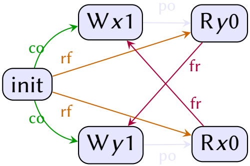
Thread 2: `y:=1;` `read x;`



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 \ WR

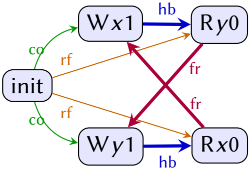
Herding Cats!

- 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 \ WR
- To get a cycle under TSO, add fences

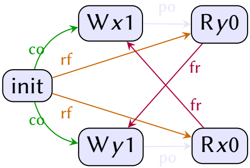
A simple idea

- Load buffering example: Allowed under TSO

Initially: `x=y=0`

Thread 1: `x:=1;read y`

Thread 2: `y:=1;read x`



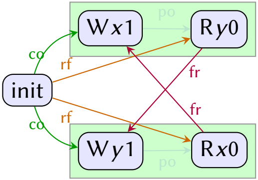
A simple idea

- 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

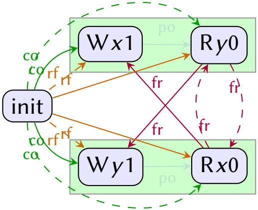
A simple idea

- 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

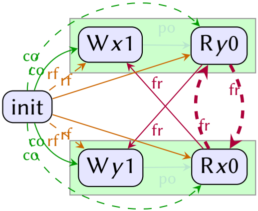
A simple idea

- 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

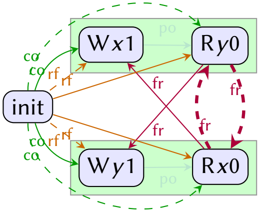
A simple idea

- 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 ⇒ acyclicity ⇒ Causal serializability

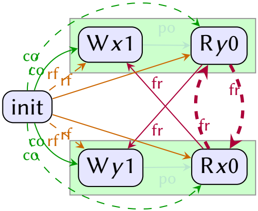
A simple idea

- 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

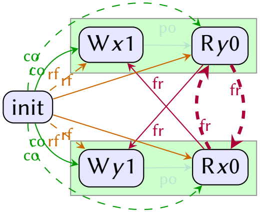
A simple idea

- 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

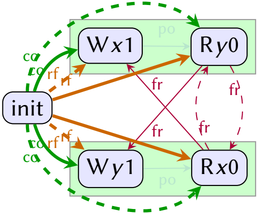
A simple idea

- Load buffering example: Allowed under TSO, without atomics

Initially: `x=y=0`

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

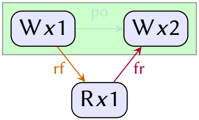
Thread 2: `atomic{y:=1;read x}`



- 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
  - Lift includes nontransactional ⇒ Strong isolation

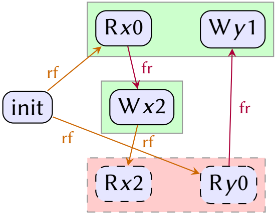
Some goals

- Nested transactions
- Weak isolation (Example under TSO)



✗ strong ✓ weak

- Abort models (Example under TSO)



✗ opaque ✓ non-opaque

Definition

- Execution is correct if AMT valid with lifted relations

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

Definition

- Execution is correct if AMT valid with lifted relations
- $e \xrightarrow{\text{lift(o)}} d$ when either
  1. $e \xrightarrow{\text{o}} d$
  2. or $e' \xrightarrow{\text{o}} d$ 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

Definition

- Execution is correct if AMT valid with lifted relations

- $e \xrightarrow{\text{lift(o)}} d$ when either
  1. $e \xrightarrow{\text{o}} d$
  2. or $e' \xrightarrow{\text{o}} d$ 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. $e \xrightarrow{\text{o}} d$
  2. or $e' \xrightarrow{\text{o}} d$ 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 $\forall d \in \text{Aborted}. \forall e \in E.\ d \longrightarrow e$ implies $e \in \text{Aborted}$
- $e \xrightarrow{\text{lift(o)}} d$ when either
  1. $e \xrightarrow{\text{o}} d$
  2. or $e' \xrightarrow{\text{o}} d$ 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 $\forall d \in \mathrm{Aborted}.\forall e \in E.\ d \xrightarrow{\mathrm{rwdep}} e$ implies $e \in \mathrm{Aborted}$
- $e \xrightarrow{\mathrm{lift(o)}} d$ when either
  1. $e \xrightarrow{\mathrm{o}} d$
  2. or $e' \xrightarrow{\mathrm{o}} d$ for some $e' \in \mathrm{descend}(e)$, $d \notin \mathrm{descend}(e)$, either $e' \in \mathrm{StrongIsolated}$ or $d \in \mathrm{Transactional}$
  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: $\mathrm{rwdep} = \mathrm{rf} \cup \mathrm{data} \cup \mathrm{addr} \cup \mathrm{ctrl}$

Definition

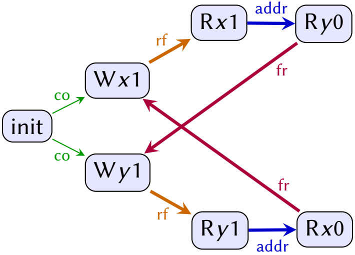
- Execution is correct if AMT valid with lifted relations and $\forall d \in \text{Aborted}. \forall e \in E.\ d \xrightarrow{\text{rwdep}} e$ implies $e \in \text{Aborted}$
- $e \xrightarrow{\text{lift(o)}} d$ when either
  1. $e \xrightarrow{\text{o}} d$
  2. or $e' \xrightarrow{\text{o}} d$ for some $e' \in \text{descend}(e)$, $d \notin \text{descend}(e)$, 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

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. $e \xrightarrow{\text{o}} d$
  2. or $e' \xrightarrow{\text{o}} d$ 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?

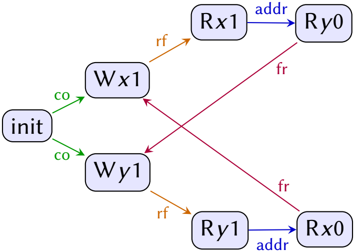
Standard Serializability?

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



Standard Serializability?

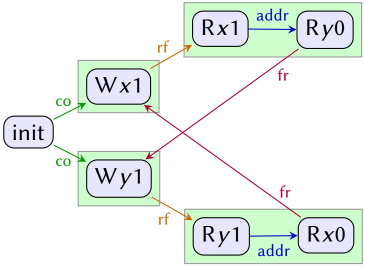
- Independent Reads of Independent Writes (IRIW)
  Forbidden for *Multi-copy atomic*, e.g. SC, TSO, ARMv8



- Allowed under ARMv7: Writes seen in different orders

Standard Serializability?

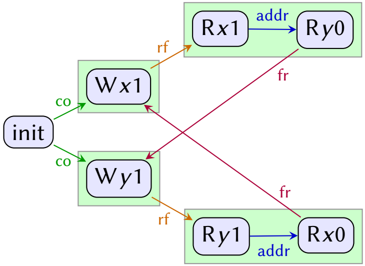
- Independent Reads of Independent Writes (IRIW)
  Forbidden for *Multi-copy atomic*, e.g. SC, TSO, ARMv8



- Allowed under ARMv7: Writes seen in different orders
- With transactions: ✓ causal serializable ✗ serializable
  Lift $\not\Rightarrow$ Standard serializability, in general

Standard Serializability?

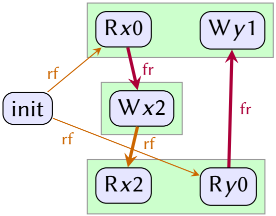
- Independent Reads of Independent Writes (IRIW)
  Forbidden for *Multi-copy atomic*, e.g. SC, TSO, ARMv8



- Allowed under ARMv7: Writes seen in different orders
- With transactions: ✓ causal serializable ✗ serializable
  Lift $\not\Rightarrow$ Standard serializability, in general
- Lift $\Rightarrow$ Standard serializability, for multi-copy atomic
  Formalized using *Global Happens Before* [Alglave 2010]

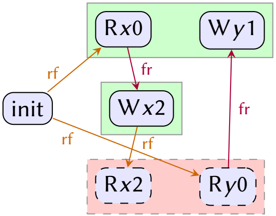
NonOpaque Aborts

- Forbidden if all commit (Example under TSO)



NonOpaque Aborts

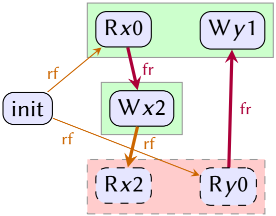
- Forbidden if all commit (Example under TSO)



- What if bottom transaction aborts?

NonOpaque Aborts

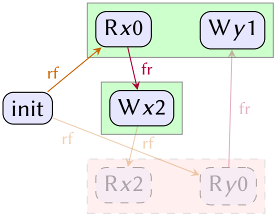
- Forbidden if all commit (Example under TSO)



- 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

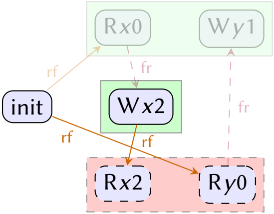
- Forbidden if all commit (Example under TSO)



- 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

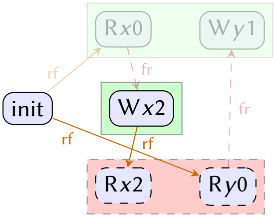
- Forbidden if all commit (Example under TSO)



- 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)

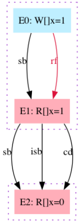
In the paper

- 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)

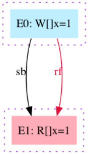
In the paper

- 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



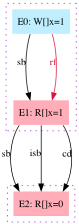
We allow



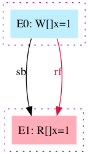
In the paper

- Non-Opaques: Comparison with VWC and TMS1
- Automaton to check violations of Global Happens Before
  Used to prove 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
  - Otherwise, our model strictly more expressive
    - HW enforces coherence with aborted
    - HW places fences before/after each transaction

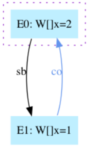
HW allows



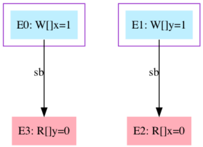
We allow



We allow



We allow



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