Hopscotch: A Scalable Flow-Graph Based Approach to Verify CPU Store Execution

Abhinav Sethi, Madhu Iyer, Sai Komaravelli, Vikram Khosa
Arm Austin Design Center
Outline

Background

• Design and Formal Environment

• Complexity Mitigation

• Memory-Tagging Extension (MTE)

Case-Study: MTE Store-Execution

• Verification Requirements

• *Hopscotch*: A Flow-Graph Framework

• Store-Execution Checker (STEXEC) Implementation

• Results

• Conclusions
LoadStore Unit
Includes L1D Cache

- **Key Structures**
  - AGU + TLB
  - Load Pipe
  - LRQ (Load Replay Queue)
  - RAR/RAW (Read-After-Write Ordering) Buffers
  - SAB + SDB (Store Addr & Data Buffers)
  - Tag/Data Arbitration & RAMs
  - RST (Recent Store Tags)
  - MB (Merge Buffer)
  - FB (Fill Buffer)
  - L2 (Arb) Interface
  - Snoop Interface
  - Prefetcher
LoadStore Unit
Includes L1D Cache

- Key Structures
  - AGU + TLB
  - Load Pipe
  - LRQ (Load Replay Queue)
  - RAR/RAW (Read-After-* Ordering) Buffers
  - SAB + SDB (Store Addr & Data Buffers)
  - Tag/Data Arbitration & RAMs
  - RST (Recent Store Tags)
  - MB (Merge Buffer)
  - FB (Fill Buffer)
  - L2 (Arb) Interface
  - Snoop Interface
  - Prefetcher

* Store Path in BOLD
LoadStore E2E Formal Testbench Architecture
Anti-Complexity Strategies

- Design Mutations
- Over-Constraint based State Reduction
- Abstractions
- Structured Case Splitting
Memory-Tagging Extension (MTE)

Overview

Problem

• Memory-unsafe languages allow
  • Unintended data corruption, or
  • Unauthorized access to sensitive data

• Bounds-Overflow
  • References outside allocation bounds

• Use-After-Free
  • Reallocating dangling memory references

Solution

• Support *coloring* of memory-allocation
  • Low-overhead
  • Probabilistic

• Algorithm
  • Assign a *color* (“tag”) to each memory-allocation
  • Store *color* in unused high bits of address pointer
  • Match *color* for each reference to stored *color* prior to access
  • Reassign *color* when freeing allocation
Memory-Tagging Extension (MTE)

Key Architectural Rules

Location, Control and Maintenance

• All Checked Loads/Stores carry a Logical Address Tag (LAT)

• 4 bits of Physical Address Tag (PAT) per 16B granule of memory

• MTE-enable granularities (Exception-Level, Pages)

• For a checked access, LAT compared against PATs for all overlapping granules in memory (cache)

• Separate instructions (LDGs and STGs) to read and write PATs

Tag-Checking for Checked Accesses

MTE Tag-Check Modes for Store Instructions:

• Precise Mode
  • Requires Tag-Check success before merge
  • Synchronous abort if Tag-Check fails
  • High perf-overhead
    • software testing

• Imprecise Mode
  • Execution not gated by Tag-Check success
  • Asynchronous abort if Tag-Check fails
  • Low perf-overhead
    • production mode
MTE Verification Complexity

- Cache-Line/Page-Crossing Ops span multiple RST entries
- Variable Store-Buffer and Merge-Buffer Occupancy (32B vs. non-32B, MBX)
- Variable Tag-Check Behavior (triggered by multiple events in Imprecise Mode, delays merge/resolve in Precise Mode)
- Checked/Unchecked Accesses to Tagged/Untagged Pages
- Separate Non-Atomic Tag and Data Accesses Per Op
- Checked Accesses and Tag Stores may span multiple Tag Granules (QWX)
# Project Intercept

## MTE Implemented on Project X

- Novel, High-Complexity Feature
- Tacked onto existing µ-arch
- Predominant impact on Store-Path
- Pre-Release Bug-Rate High

## State of LoadStore Formal Environment

- Bring-up complete with basic stimulus
  - Loads + Stores + Snoop-Requests
- Read(Load)-Value Checker
  - Only E2E Checker to cover Store-Path
  - Longer path + high formal complexity
    - includes *Store-to-Load-Forwarding* path
- Expect insufficient proof-coverage
Checking Requirements

• Initial planning indicates
  
  • Multiple independent architectural/μ-arch checkers to verify Store Path
  
  • Across different Tag-Check modes, MTE attributes, instruction-types, alignments etc.
  
  • Significant overlap in tracking required across checkers
  
  • Duplication of effort
## Rationale for *Hopscotch*

**Abstracting Execution Flows**

### What If?

We instead develop a Unified Framework

- span the entire lifecycle of a Store µop
- identify key events
  - architectural & micro-architectural
- specify legal (or illegal) event orderings
  - for each flow
    - mode, instruction-type, other attributes
- at an event occurrence
  - invoke checks for both safety and progress
  - key off other functional checks

### Potential Advantages?

- Flexible
- Modular
- Iterative
- Scalable
Store-Execution Flow as a Multigraph

- Nondeterministic Abstraction of all Legal Store-Executions
  - for a given Flow-Type (+Flow-Attributes)

- For a given **Store-Execution Flow** $F$

  - Each Event in $F \leftrightarrow$ A visit to an Event-Node $N$
    - on a set of Directed Flow-Graphs $FG_0, FG_1, \ldots, FG_R$ for Visit-Relations $VR_0, VR_1, \ldots, VR_R$

  - Each Directed Edge (Path) on a Flow-Graph captures a single Visit-Relation

  - Each Event-Node is mapped to a Node-Type and Node-Attributes

  - A subset of Event-Nodes can be grouped into a Rendezvous-Node
    - Has Barrier semantics
    - Used to define forward path-requirements from any Node.
Example Flow-Graph (Precise-Mode Store-Exclusive)

Legal Hop
Illegal Hop
Mutex
Example Flow-Graph (Precise-Mode Store-Exclusive)
Hopscotch : Design Goals

• Clean separation between
  • A user-defined/maintained layer of functional specification
    • express ordering requirements for abstracted design events using simple temporal operators
  • A static, concise code-substrate operating on domain-agnostic, configurable, regular structures
    • translate, store and execute the user-specification
    • enable auto-generation of checkers and coverage

• Ease of maintenance
  • Allow spec updates to be clearly captured and interactively tested (CEX-guided)
  • Little impact on underlying codebase

• Ease of decomposition
  • Supports automated case-splitting and path-decomposition
  • Both key to mitigating formal complexity.
Sample Node Specification (Exclusive-Check Fail)

```c
//************
//STREX FAIL
//************

//STREX Fail can occur after RST-allocation if monitor unarmed
LEGAL_OPEN_HOP( STEXEC_NODE_EVENT_SB_RST_ALLOC_LNK_OW, STEXEC_NODE_EVENT_FAIL_EXMONCHK)

`LEGAL_RACE( STEXEC_NODE_EVENT_SB_MB_ALLOC_LNK_OW, STEXEC_NODE_EVENT_FAIL_EXMONCHK)

//STREX Fail can also occur after TSB_MERGE (if monitor was armed at the time of RST allocation??)
`LEGAL_RACE( STEXEC_NODE_EVENT_TSB_MERGE, STEXEC_NODE_EVENT_FAIL_EXMONCHK)

`LEGAL_RACE( STEXEC_NODE_EVENT_RST_TAG_LOOKUP_OW, STEXEC_NODE_EVENT_FAIL_EXMONCHK)

`LEGAL_CLOSED_HOP( STEXEC_NODE_EVENT_FAIL_EXMONCHK, STEXEC_NODE_EVENT_FAIL_LATCH_OW)

`LEGAL_CLOSED_HOP( STEXEC_NODE_EVENT_FAIL_EXMONCHK, STEXEC_NODE_EVENT_PASS_LATCH_OW)

//At least OWN RST-alloc must have passed before STREX-FAIL
`LEGAL_OPEN_PATH_MUST_INCLUDE( STEXEC_NODE_EVENT_SB_RST_ALLOC_LNK_OW, STEXEC_NODE_EVENT_FAIL_EXMONCHK)

//A STREX cannot fail or have failed before it writes L1D
`LEGAL_CLOSED_PATH_MUST_EXCLUDE( STEXEC_NODE_EVENT_FAIL_EXMONCHK, STEXEC_NODE_EVENT_L1D_WR)
```
Hopscotch Implementation

Filtered DUT Events

Event Nodifier

Tracked-Txn State

Flow-Multigraph

Trace Graph
Hopscotch-based STEXEC Checker

- Filtered DUT Events
- Event Nodifier
- Tracked-Txn State
- Tag-Check Safety

Flow Safety
Flow Progress
Data & Tag Consistency
Flow-Safety Checkers

- Trace-State
- Node-State
- Hop Safety
- Concurrence Safety
- Path Safety
Trace and Node State Checkers

Invoked at each Event-Node visit against Trace-Graph

- Check that *Trace-State* never becomes *ILLEGAL*
- Check that *Node-State* does not become *ILLEGAL*
  - Based on *Node-Type* and *Trace-State*
  - Includes revisit-bounds.
Hop and Concurrence Flow Safety Checkers

Invoked for each of N nodes currently visited

- Check against M nodes visited last
  - all $M \times N$ node hops are \textit{LEGAL}

- Check that no positive or negative concurrence relations are being violated
  - w.r.t. the remaining $N-1$ visited nodes
Path Flow-Safety Checkers
Invoked for all N nodes currently visited

• Check against all other nodes ➔
  • No defined path-inclusion relations are violated
  • No defined path-exclusion relations are violated
Hop & Concurrence Flow-Safety-Check Animation
Hop & Concurrence Flow-Safety Check Animation
Hop & Concurrence Flow-Safety Check Animation
Hop & Concurrence Flow-Safety Check Animation
Hop & Concurrence Flow-Safety Check Animation
Hop & Concurrence Flow-Safety Check Animation
Hop & Concurrence Flow-Safety Check Animation
Hop & Concurrence Flow - Safety Check Bug Animation
Hop & Concurrence Flow-Safety Check Bug Animation
Path Flow-Safety Check Animation
Path Flow-Safety Check Animation (Bug)
Flow-Progress Checkers (Assume-Guarantee based)

- Hop Progress
- Rendezvous Progress
- Stall-Clear Guarantee
Hop Flow-Progress Checkers
Invoked for each of N non-leaf atomic nodes currently visited

Starting at a visit to node $N_i$, unless escaped or aborted:

- **Liveness Variants (require fairness):**
  - Will always eventually visit at least one atomic node $N_j$ to which a hop is legally defined

- **Bounded Safety Variant:**
  - Compare a predefined threshold against a count of cycles during which
    - no legally defined *internal stalls* (assume bounded) or *external waits* defined for $N_i$ are active, AND
    - we have not legally hopped to another atomic node $N_j$
Rendezvous Flow-Progress Checkers

Invoked for each of N non-leaf atomic or rendezvous nodes currently visited

- Rendezvous Nodes
  - Composite nodes with barrier semantics (all visited, any visited etc.)

- Starting at a visit to node $N_i$, unless escaped or aborted:
  - **Liveness Variants (require fairness):**
    - will *always eventually* visit all required downstream rendezvous nodes ($R_a, R_b, ...$)
  - **Bounded Safety Variant:**
    - Compare a predefined threshold against a count of cycles during which:
      - no legally defined *internal stalls* (assume bounded) or *external waits* defined for $N_i$ are active, AND
      - we have not legally hopped to each required downstream rendezvous nodes ($R_a, R_b, ...$)
Stall-Clear Guarantee Checkers
Invoked for each of N non-leaf atomic nodes currently visited

• Coupled (assumed) with Progress Checkers.

• For any *internal-stalls* assumed as bounded for a given node, independently check *(guarantee)* that they will clear:
  
  • *Liveness variant*: eventually, assuming fairness on *external-wait* dependencies

  • *Bounded Safety variant*: within a specified number of cycles
Hop Flow-Progress Check Animation
Hop Flow-Progress Check Animation
Hop Flow-Progress Check Animation
Hop Flow-Progress Check Animation
Hop Flow-Progress Check Animation
Hop Flow-Progress Check Animation
Rendezvous Flow-Progress Check Animation
Rendezvous Flow-Progress Check Animation
Rendezvous Flow-Progress Check Animation
Results

- Total of late bugs hit by STEXEC: 19
  - 6 post-release
  - 10 formal-only, 9 reproductions
Sample Bugs Found by Flow-Safety Checkers

**Safety Bug 1** (Node-State Check)

Each node in the flow-multigraph is allowed to be visited only a specified number of times.

*Bug:* RTL was writing Tracked STG op’s PAT and DAT at different times (writing the same PAT twice).

Also hit by *Hop-Safety Check*

**Safety Bug 2** (Concurrence Check)

If node A is visited, then node B should also get visited.

*Bug:* RTL was writing Tracked STGP op’s PAT without writing its DAT.
Sample Bug Found by Flow-Progress Checkers

If node $A$ is visited, then at least one hop-related node $B$ and all required downstream rendezvous nodes $C$ (from $A$) should eventually always get visited.

- **Bug:**
  - In Precise Mode, RTL executes a *Tag-Check* for a Store-Exclusive micro-op which *fails* due to hitting a poisoned *PAT*.
  - Since unarmed, it signals *STREX* complete (failure) but never arbitrates for broadcasting the failure because the *Merge-Buffer* was never written due to the *Tag-Check* error above.
  - *Progress-Checker* fails because a required Rendezvous Hop from *STREX_FAIL* to *STREX_RESULT* was never taken (*Progress-Counter* saturation in the absence of transient stalls / external waits)
  - Multiple variants hit
Key Observations

**Hopscotch**

- **Accelerated overall Store-Execution bring-up**
  - Safety checks valuable for sanity testing
    - Caught lots of otherwise-subtle TB issues quickly
  - Debug productivity boost from visual logging of node visits

- **Performance of matrix implementation for flow-multigraphs**
  - Sensitive to total number of nodes:
  - Added nodes increase complexity + memory requirements

**Checkers**

- **Required variable level of white-boxing**
  - Limit to high-level events at first
  - Model u-arch events as needed
  - Tradeoff: checker precision vs. modeling effort

- **Little payoff from liveness checkers**
  - Different source of complexity?
Looking Ahead

Enhancements/Extensions

Flowgraphs are naturally extendable to generate lower-level coverage models (for signoff +/- DBH)

- Pairwise hop coverage
- Path coverage
- Per-node Stall coverage

Optimizations

- Reduce graph size/complexity
  - Sparse matrix implementations

- New/improved traversal/check algorithms used in safety and progress checkers

- Liveness checking
Conclusions

• Investments in complexity reduction → huge impact on baseline TB performance

• Focusing on the right problems with the right toolset critical to adding value with formal
  • Time + effort in careful planning of both scope and implementation well-spent
  • *Hopscotch* framework: speedy, flexible and scalable way to build and test E2E checkers

• Developed a mature formal environment for the LS Store-Path
  • within a couple of months
  • added confidence to RTL release quality
Questions?
Backup Slides
# Key Compound Oracles

<table>
<thead>
<tr>
<th>$O_{\text{line}}$</th>
<th>$O_{\text{uop}}$</th>
<th>$O_{\text{data}}$</th>
<th>$O_{\text{init}}$</th>
<th>$O_{\text{check}}$</th>
<th>$O_{\text{gatekeeper}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracked Cache-line Addresses (VA including VA-alias)</td>
<td>(Instruction) micro-op(s)</td>
<td>Size and Byte-Offsets of Tracked Data Granules within Tracked Cache-line Addresses chosen by $O_{\text{line}}$</td>
<td>Initial-State choices for IVAs &amp; Abstraction Models bound to the DUT</td>
<td>Unique enumerated choice of Checker to activate from among the set of supported Checkers</td>
<td>Non-deterministic choice of which eligible event(s) are picked when to be reported to Checker</td>
</tr>
<tr>
<td>Tracked Translations (PA, Memory Type, Cacheability etc.)</td>
<td>Op-Type</td>
<td>Cache-State, Way, Value Tracked Data Granule, Exclusive Monitor etc.</td>
<td></td>
<td></td>
<td>Not Stable</td>
</tr>
<tr>
<td>Context</td>
<td>Size</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Structured Case-Splitting
Precondition Conjugations

- Implemented in source for key, hard E2E Checkers

- Higher bounds and full proofs achieved for **Extreme Case-Splits**
  - Concretized values for symmetrical or interesting oracle choices
    - Pick bit 0 of byte 0 to check
    - Pick only cacheable addresses to only 1 bank to check
    - Pick only cases with hits to check (TLB, Tag-RAM abstraction-model policy oracle choices)
    - Pick only one specific op-type to check

- Accelerated by helper assertions

- Appropriate to be included in smoke testing

- Each case-split (value) enables sensitivity analysis
  - effect on proof-convergence & contribution to complexity
Compile-time Transaction-Limiting Profiles

Static Over-Constraint Recombining

- Create a reference set of OCs (Over-Constraints)
  - Sources
    - Interfaces
    - Address-Space
    - Checker Oracles
    - Abstraction Oracles
  - Types
    - Disable types of stimulus
    - Limit number of transactions of each type of enabled stimulus
    - Narrowed/unique choice of oracle values

- Methodology
  - Map each OC into corresponding `define
  - Concatenate `defines into a set of named, unique profiles
  - Allow profiles to be specified at build time
  - Omit out-of-focus (UNR) properties on a per-profile basis ("waiver" flow)
Runtime Transaction-Limiting Profiles
Dynamic Over-Constraint Recombining (Loc-K-Picker)

• Create a static pool of *weighted, abstracted OCs* (Over-Constraints) with attributes and dependencies
  • Inclusion
  • Mutual-exclusion

• On each invocation, pick a random set of *K* concretized, mutually-consistent Local Over-Constraints
  • Solve the knapsack problem
  • Dynamically applied on a per-task basis
    • Each task/proof-thread gets a unique set of selected OCs
    • Supported for both proof and DBH threads
Store-Execution Flow as a Multigraph

- Non-deterministic Abstraction of all Legal Store-Executions
  - for a given Flow-Type (+Flow-Attributes)

- For a given Store-Execution Flow $F$
  - Each Event in $F \leftrightarrow$ Visit to an Event-Node $N$ on a set of Directed Flow-Graphs
    - $FG_0, FG_1, ..., FG_R$ for $R$ Visit-Relations $VR_0, VR_1, ..., VR_R$
  - Each Directed Edge (Path) between Event-Nodes on a Flow-Graph $FG_i$
    - Captures a unique Direct (Transitive) Visit-Relation $VR_i$
  - Each Event-Node is mapped to a set of Node-Attributes
## Atomic Node Types

<table>
<thead>
<tr>
<th>CODE</th>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>Initialized</td>
<td>Default</td>
</tr>
<tr>
<td>ASYNC</td>
<td>Asynchronous</td>
<td>No ordering relation w.r.t. any other node</td>
</tr>
<tr>
<td>ROOT</td>
<td>Root</td>
<td>First node(s) visited e.g., Tracked Txn accepted</td>
</tr>
<tr>
<td>FORK</td>
<td>Perform Boolean Test</td>
<td>Checker Invocation (PASS/FAIL)</td>
</tr>
<tr>
<td>IHOP</td>
<td>Intermediate Hop</td>
<td>Intermediate Event (neither root nor leaf)</td>
</tr>
<tr>
<td>FAIL</td>
<td>Failure</td>
<td>Tracked Txn Failure/Abort</td>
</tr>
<tr>
<td>ESCP</td>
<td>Escape</td>
<td>Cannot disambiguate Tracked Txn in future (aliasing event)</td>
</tr>
<tr>
<td>ENDP</td>
<td>Endpoint</td>
<td>Tracked Txn success; No outbound edges</td>
</tr>
</tbody>
</table>
### Primitive and Composite Visit-Relations

**Directed** $(A \rightarrow B)$

#### Hop-Relation
- **ILLEGAL**
  - $A$ can never be immediately followed by $B$
- **OPEN**
  - Legal if $A$ visited strictly before $B$
- **CLOSED**
  - Legal if $A$ visited before or concurrently with $B$
- **NONE**
  - Don’t care

#### Concurrence-Relation

**Primitives:**
- **POSitive Implication**
  - $A \rightarrow B$
- **NEGative Implication**
  - $A \rightarrow \neg B$

**Composites:**
- **MUTEX**
  - $A$ and $B$ never concurrent
- **CONDITIONAL** (One-Way)
  - $A$ implies $B$ concurrent but not vice-versa
- **COUPLED**
  - $A$ and $B$ always concurrent

#### Path Relation

**Primitives:**
- **NONE**
- **OPEN**
- **CLOSED**
- **NONHOP**

**Two Flavors:**
- **INCLUSION**
  - $A$ visited on every path to $B$
- **EXCLUSION**
  - $A$ never visited on a path to $B
Rendezvous Nodes
Composite Nodes with “Barrier” semantics

Rendezvous Type

- **ANY**
  - Reached once any member Node(s) visited
- **ONE**
  - Reached once exactly one member Node visited (one-hot)
- **ALL**
  - Reached once all member Nodes visited
- **NONE**
  - Don’t-care

Progress Checks

- **Node-to-Rendezvous**
  - Define progress from a visited node once all downstream rendezvous nodes mapped are subsequently visited
- **Cross-Rendezvous**
  - Define progress from one rendezvous node to another
Flow Multigraph Representation

- Implemented via enumerations and matrices (2-D arrays) in Verilog

- Represented as a set of:

  - *Event Node* Declarations with Attributes and Membership
    - Node-Type [Atomic | Rendezvous]
    - Revisitability [NEVER | ONCE | UNLIMITED]
    - Thread + Strand

  - Flow-Graphs
    - Hop-Relation and Concurrence Relations
    - Path-Inclusion & Path-Exclusion Relations

- *Threads* and *Strands*
  - Enable concise specification via support for node-affinity
  - Initialize all Relations to *Don’t-Care* across nodes in different *Threads/Strands*
Store-Execution Trace Graph

- **Concrete Trace**

  - Represents a single deterministic Store-Execution
    - of a Tracked Store op to the Tracked Data Bit on the Tracked Cacheline

  - Overlays static Flow-Multigraph with dynamic Trace-State
    - Update for a set of Event-Nodes visited in a cycle:
      - **Global-State**
        - [ IDLE | ACTIVE | ESCAPED | ABORTED | DONE | ILLEGAL ]
      - **Node-States**
        - [ IDLE | VISITED | REVISITED | ILLEGAL]
      - **Last (Multi-) Hop**
        - Set of Event-Nodes Last Visited

  - Enables checks automatically triggered for one or more Event-Nodes defined in the Flow Multigraph
    - Safety Checks (Retrospective)
      - against Event-Nodes that (ought to) have been visited so far
    - Liveness Checks (Progressive)
      - against Event-Nodes (ought) to be visited in the future
Data & Allocation-Tag Consistency

**DAT (Store Data)**

- MTE-mode agnostic.
- For all Store-Types with Data:
  - Check consistency of *Tracked DAT Bit* of *Tracked Store Op* against
    - merge-data at Merge
    - write at L1$ or L2 interface

**PAT (Allocation-Tag)**

- For *STGs (Stores to Allocation-Tag)*:
  - Check consistency of *Tracked PAT Bit* of *Tracked Store Op* against
    - *PAT* written to L1 cache.
    - *PAT* streamed to L2
Tag-Check Correctness
Predict Tag-Check Occurrence and Outcome

Precise Mode

• For a Tracked Checked Store op, check for:
  • on a clean resolve, LAT must have matched the latest PATs in memory for all spanned QW granules
  • If LAT doesn’t match the latest PATs in memory for all spanned QW granules, we must resolve with a u-arch abort ("nuke")
  • Consider all alignments and SBX/MBX cases.
  • High-Level, triggered at merge/resolve time.

Imprecise Mode

• For a Tracked Checked Store op,
  • Check correctness of Tag-Checks at different points
    • triggered at RST-lookup, store-merge, fill.
  • Allow for accumulation of older stores to the same line towards Tag-Check result
  • Special handling for CLX/PGX cases and poison/SEI
  • Requires partial implementation-choice modeling for precision.
Timeline

March:
- Project Y early release

Q1-Q2:
- LS formal bring-up on Project X -> Project Y

Mid-May:
- STEXEC planning

June:
- STEXEC bring-up on Project Y

Early July:
- Hit Project Y 1st Bug

July 1st:
Event Logging for STEXEC Traces
Sample Bug From Imprecise-Mode Tag-Check Checker

• On cycle 11, we do a tag-lookup for the Tracked Store with QW=0 but set Tag-Checked indicator even though QWs are not enabled

• On cycle 28, when the Tracked Store merges but following a line state transition from EVICT to SHARED, we do another Tag-Check, which indicates a mismatch.

• However, we don’t flag the Tag-Check fail correctly because Tag-Checked indicator is previously set

• We miss reporting the result of the Tag-Check the 2nd time around
Time-to-Cover (Raw Hop vs. Hop-Safety Checker Witness)

**Precise-Mode TXN=16**

**Imprecise-Mode TXN=16**

- **MIN-BOUND**
- **T(RAW-NODE)**
- **T(PC)**