TransForm: Formally Specifying Transistency Models and Synthesizing Enhanced Litmus Tests

Naorin Hossain
Princeton University

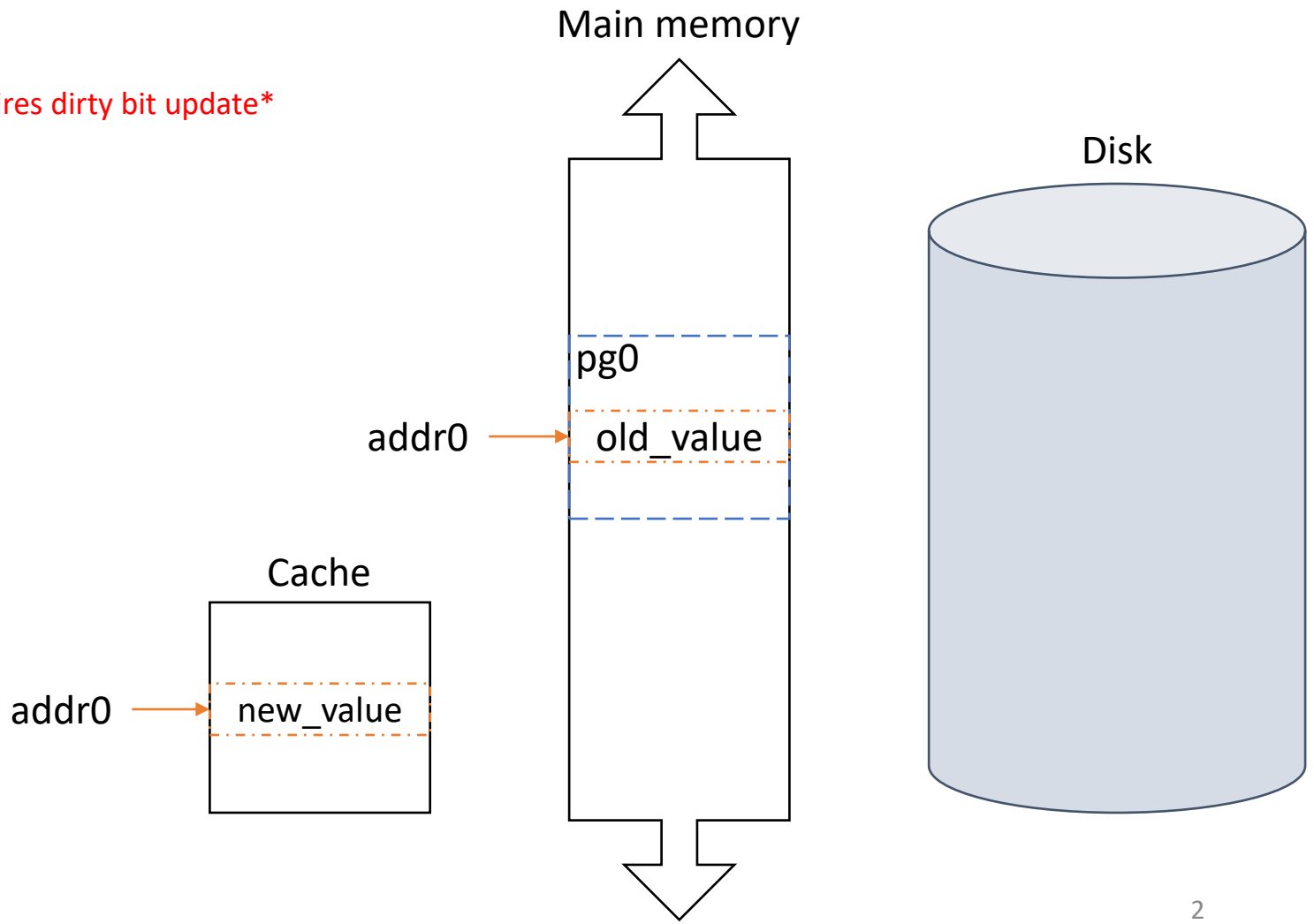
FOCA 2020
October 30, 2020

Naorin Hossain, Caroline Trippel, Margaret Martonosi. "TransForm: Formally Specifying Transistency Models and Synthesizing Enhanced Litmus Tests", ISCA '20.

Page replacement needed when memory is full and data from disk is being accessed

```
*addr0 = new_value    *requires dirty bit update*
         *
         *
         *
data = *addr1
```

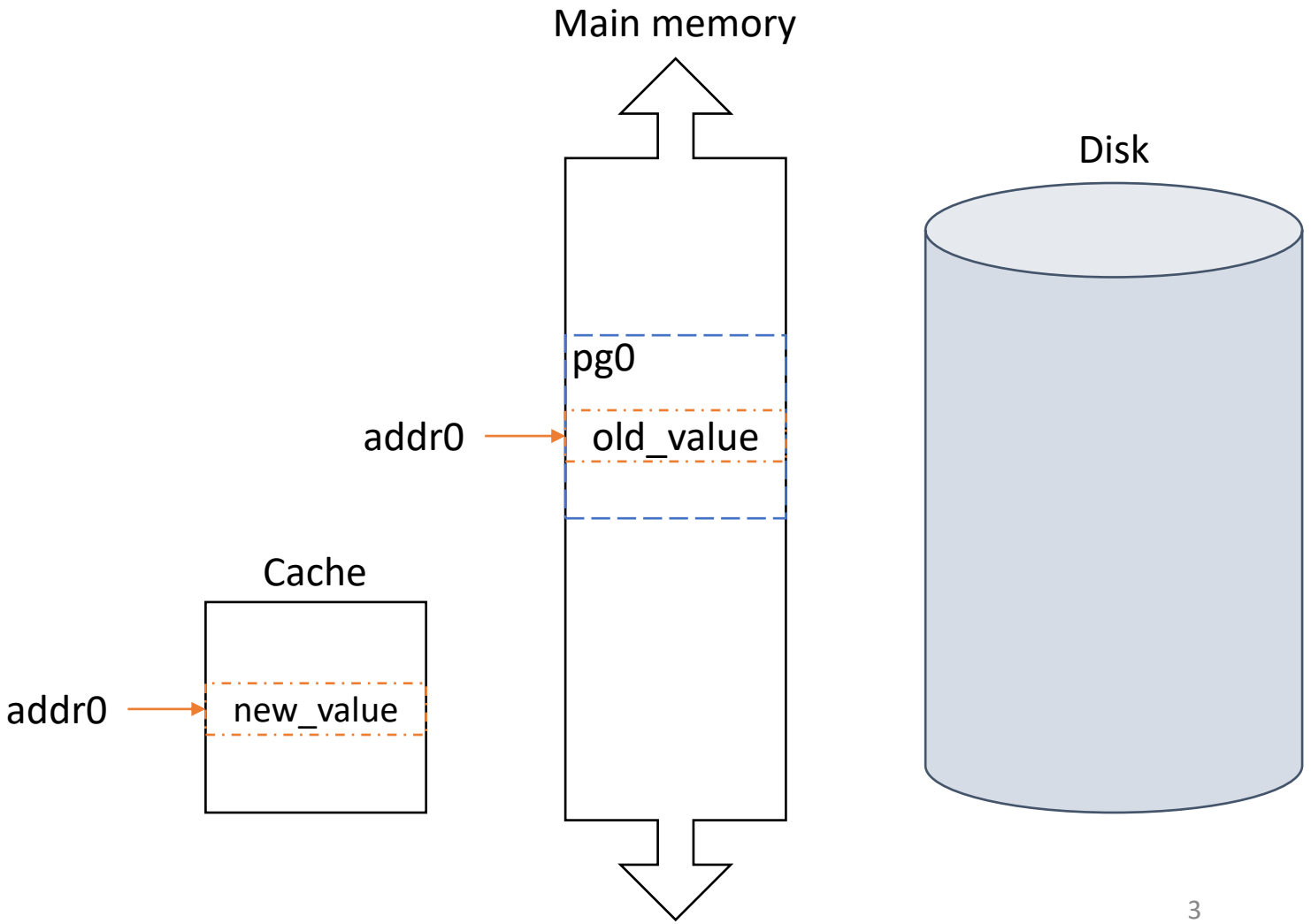
	Page Table				
	A	D	R	W	Physical page
addr0	0	1	1	1	pg0
addr1	0	0	1	0	disk



Page replacement needed when memory is full and data from disk is being accessed

```
*addr0 = new_value
         *
         *
         *
data = *addr1
```

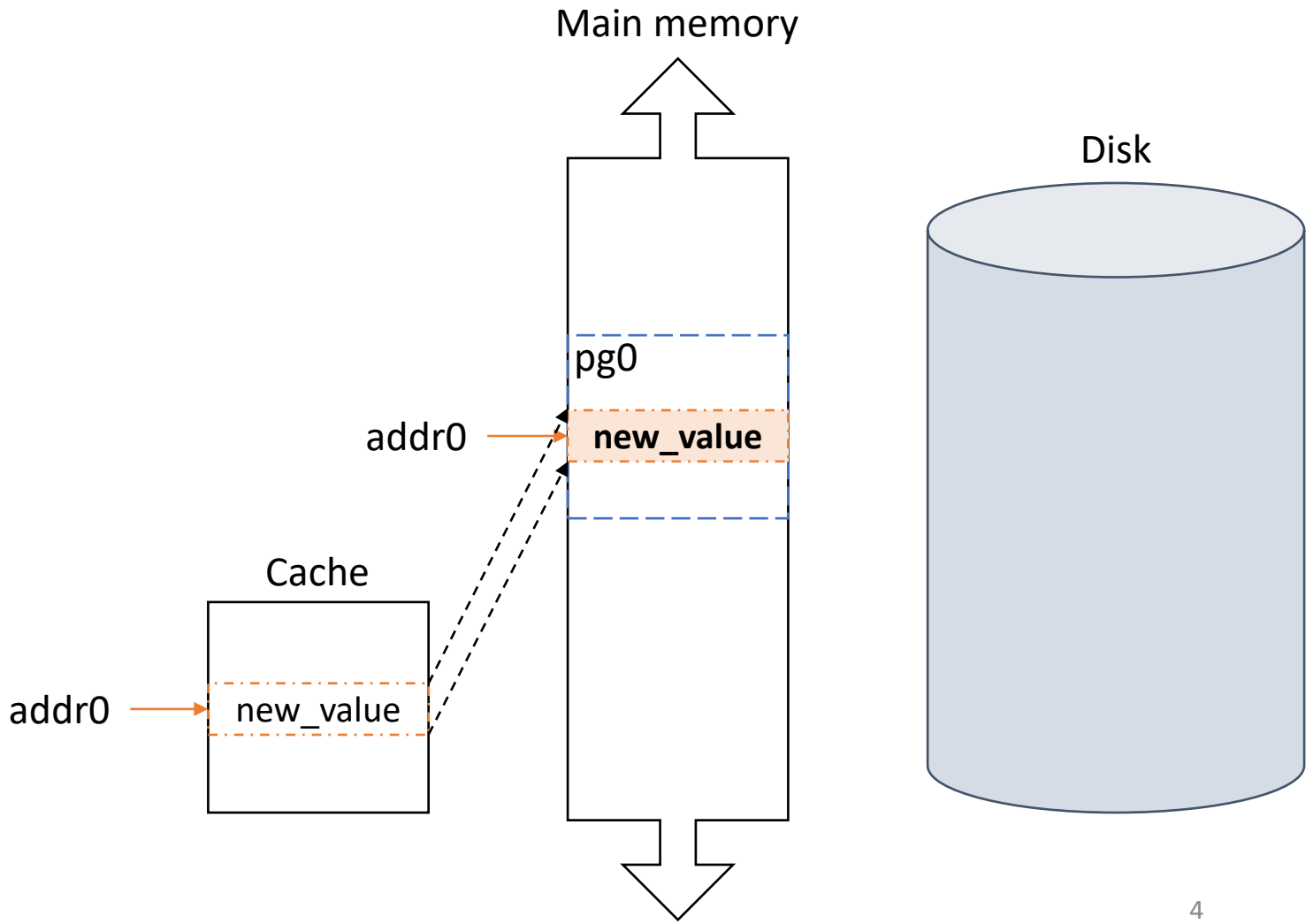
	Page Table				
	A	D	R	W	Physical page
addr0	0	1	1	1	pg0
addr1	0	0	1	0	disk



Page replacement needed when memory is full and data from disk is being accessed

```
*addr0 = new_value
         *
         *
         *
data = *addr1
```

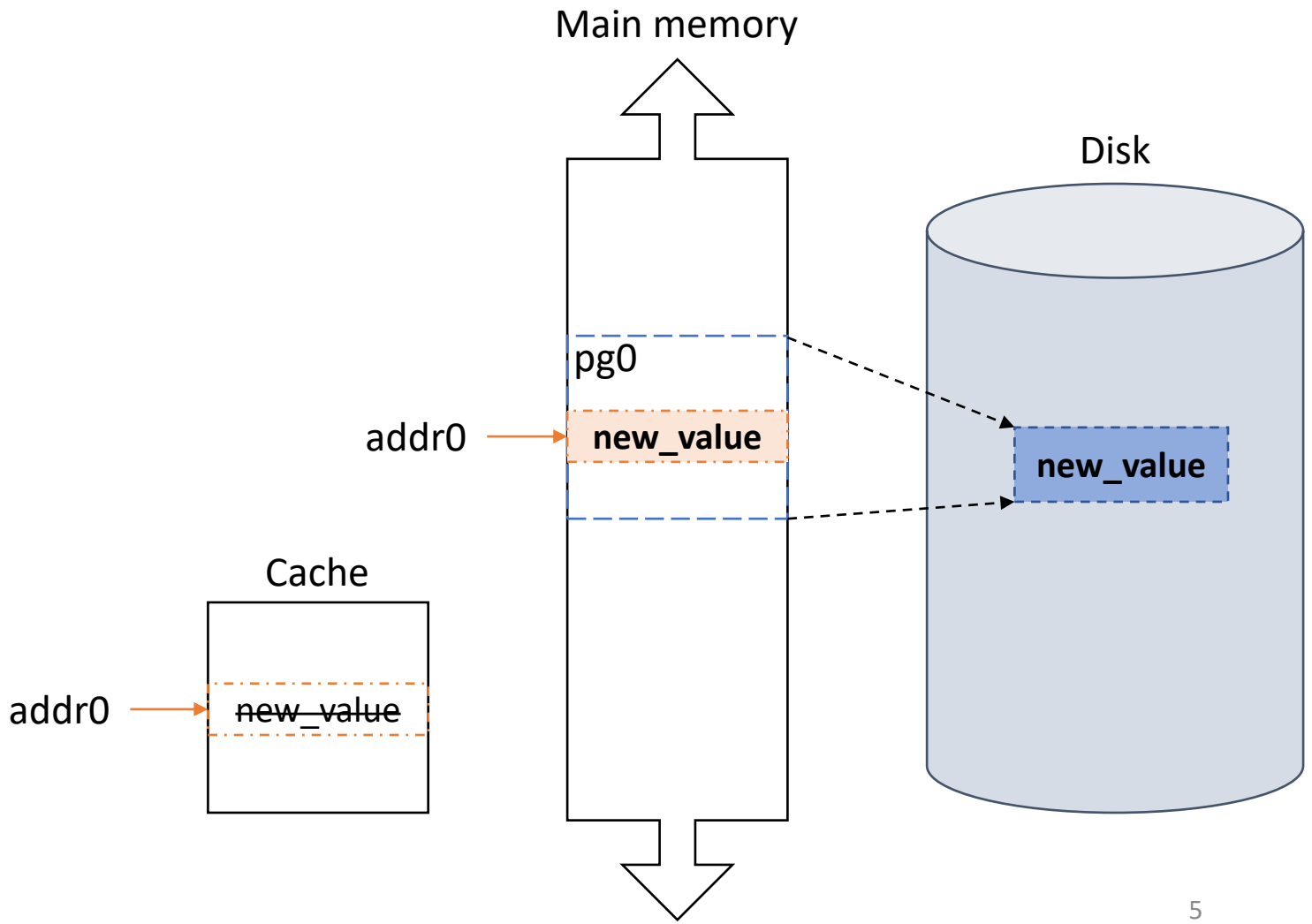
	Page Table				
	A	D	R	W	Physical page
addr0	0	**0**	1	1	pg0
addr1	0	0	1	0	disk



Page replacement needed when memory is full and data from disk is being accessed

```
*addr0 = new_value
         *
         *
         *
data = *addr1
```

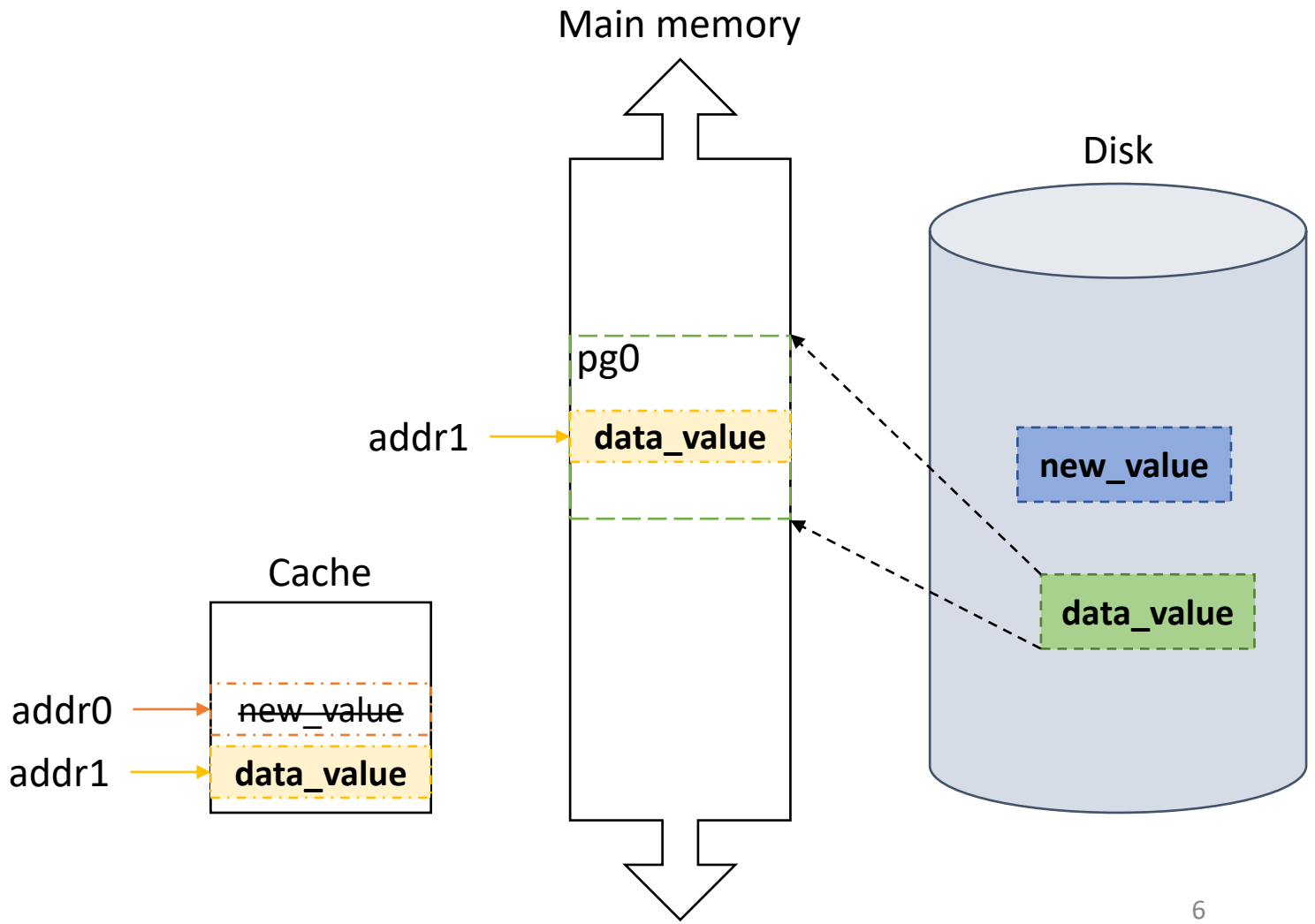
	Page Table				
	A	D	R	W	Physical page
addr0	0	0	1	1	disk
addr1	0	0	1	0	disk



Page replacement needed when memory is full and data from disk is being accessed

```
*addr0 = new_value
         *
         *
         *
data = *addr1
```

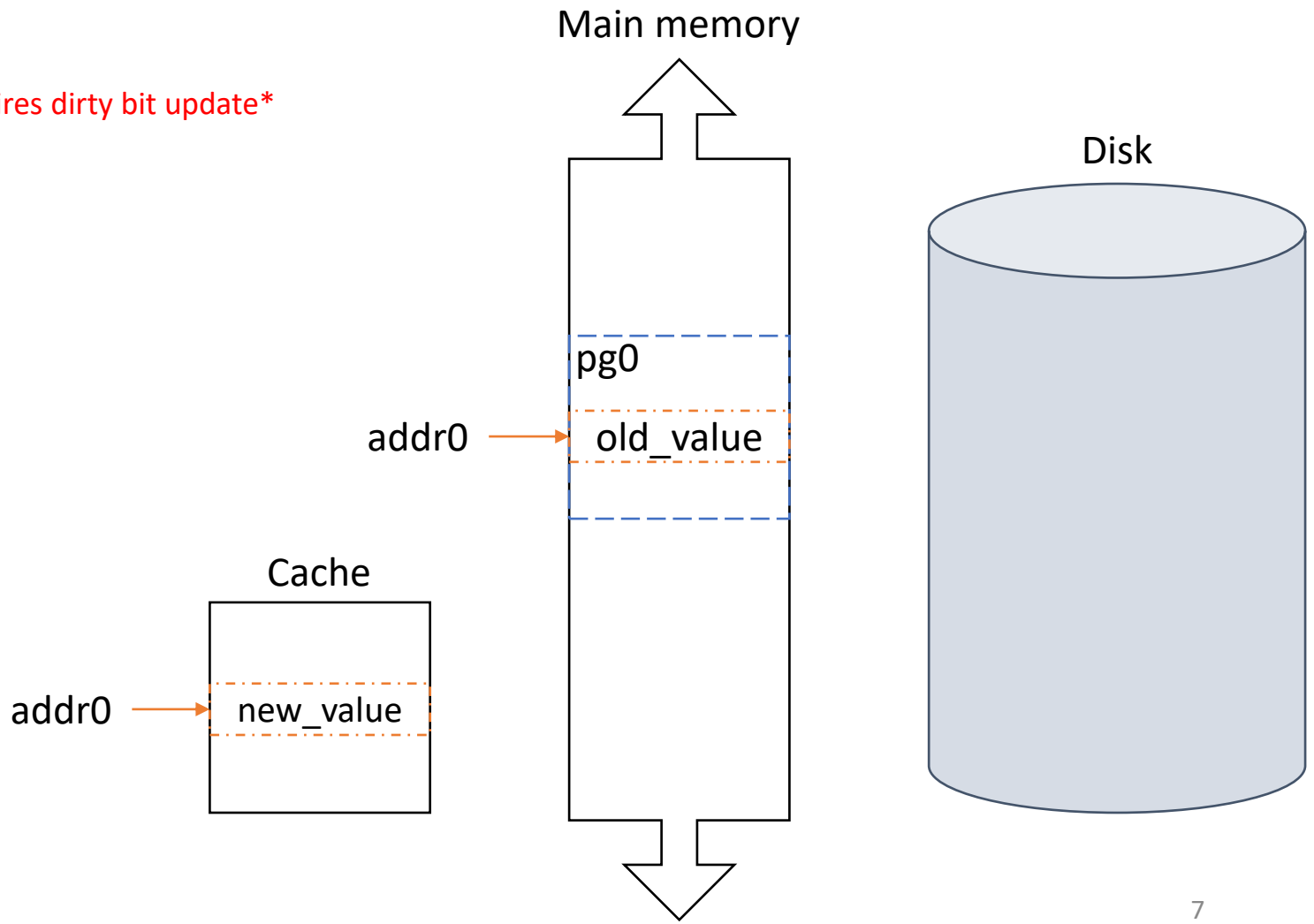
	Page Table				
	A	D	R	W	Physical page
addr0	0	0	1	1	disk
addr1	0	0	1	0	pg0



What if dirty bit is not updated before page swapped to disk?

```
*addr0 = new_value    *requires dirty bit update*
         *
         *
         *
data = *addr1
```

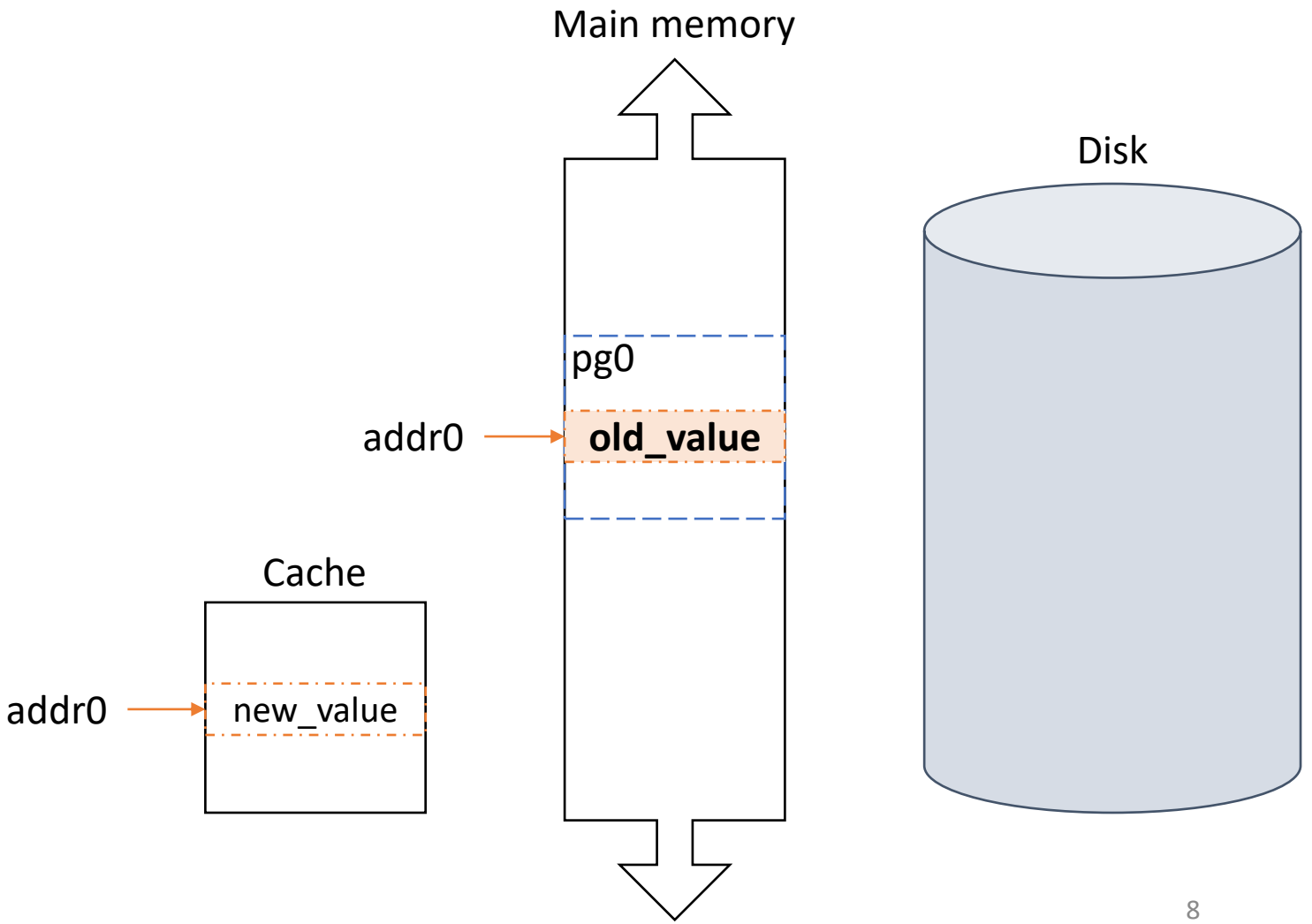
	Page Table				
	A	D	R	W	Physical page
addr0	0	0	1	1	pg0
addr1	0	0	1	0	disk



What if dirty bit is not updated before page swapped to disk?

```
*addr0 = new_value
         *
         *
         *
data = *addr1
```

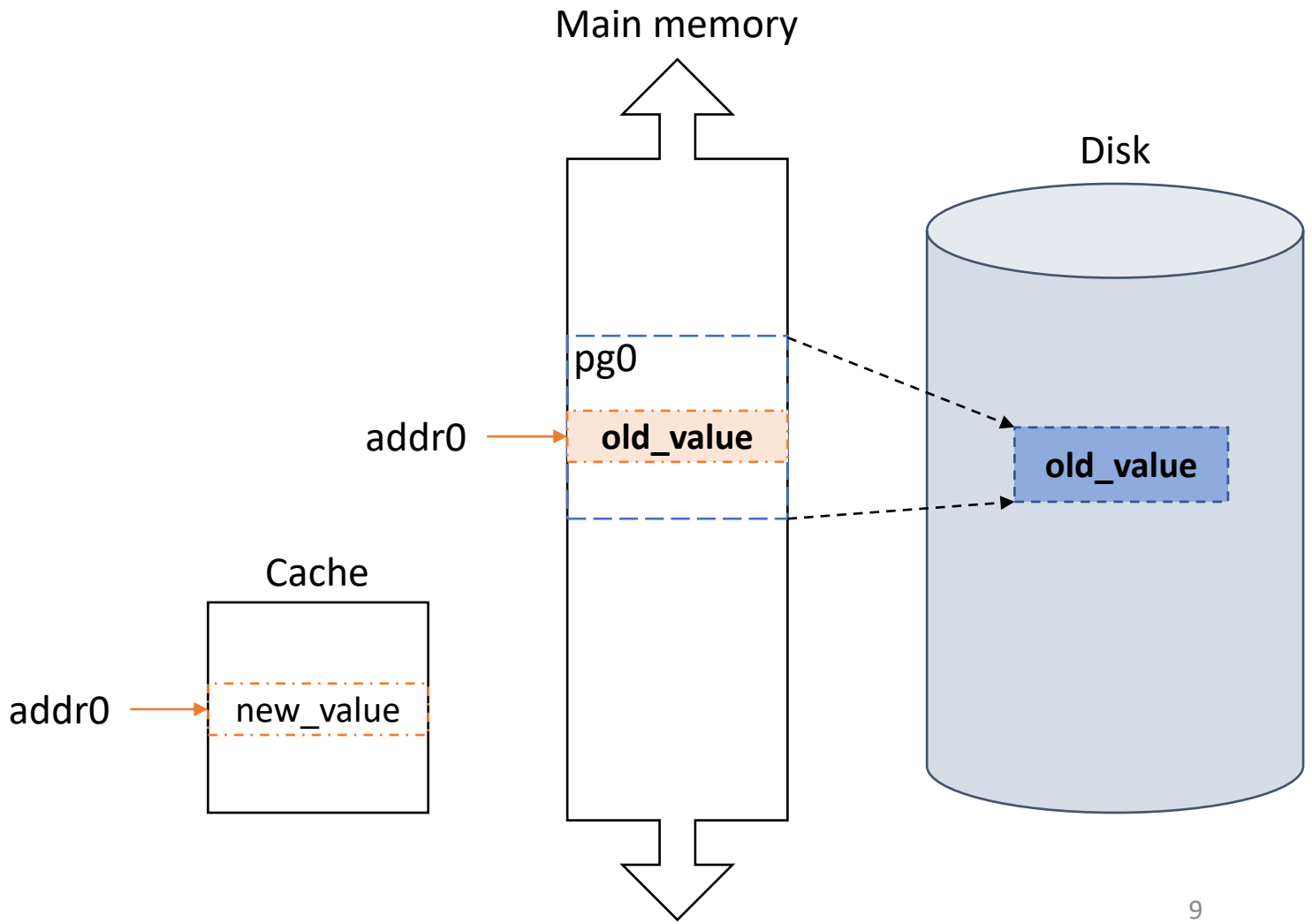
	Page Table				
	A	D	R	W	Physical page
addr0	0	0	1	1	pg0
addr1	0	0	1	0	disk



What if dirty bit is not updated before page swapped to disk?

```
*addr0 = new_value
         *
         *
         *
data = *addr1
```

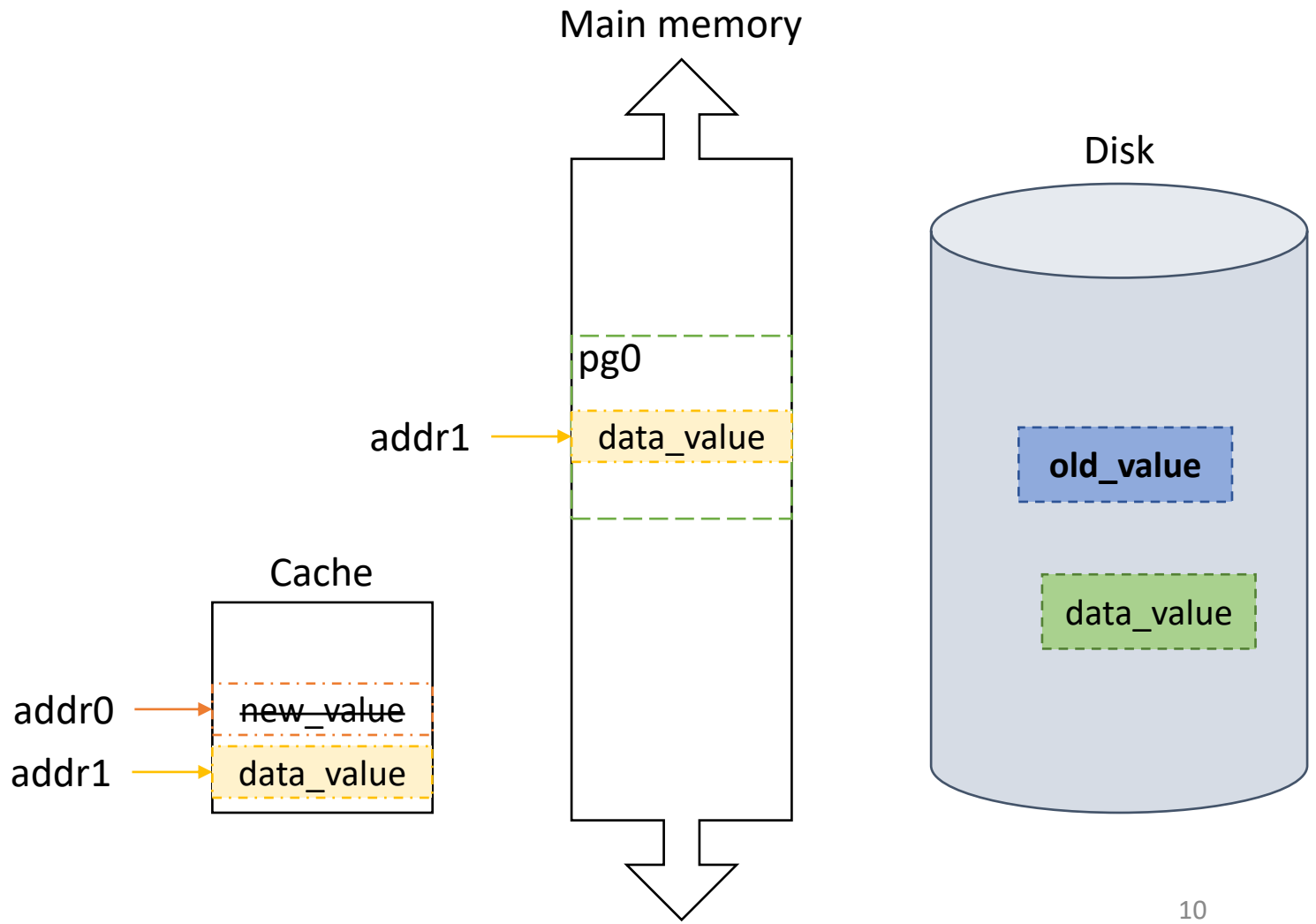
	Page Table				
	A	D	R	W	Physical page
addr0	0	0	1	1	disk
addr1	0	0	1	0	disk



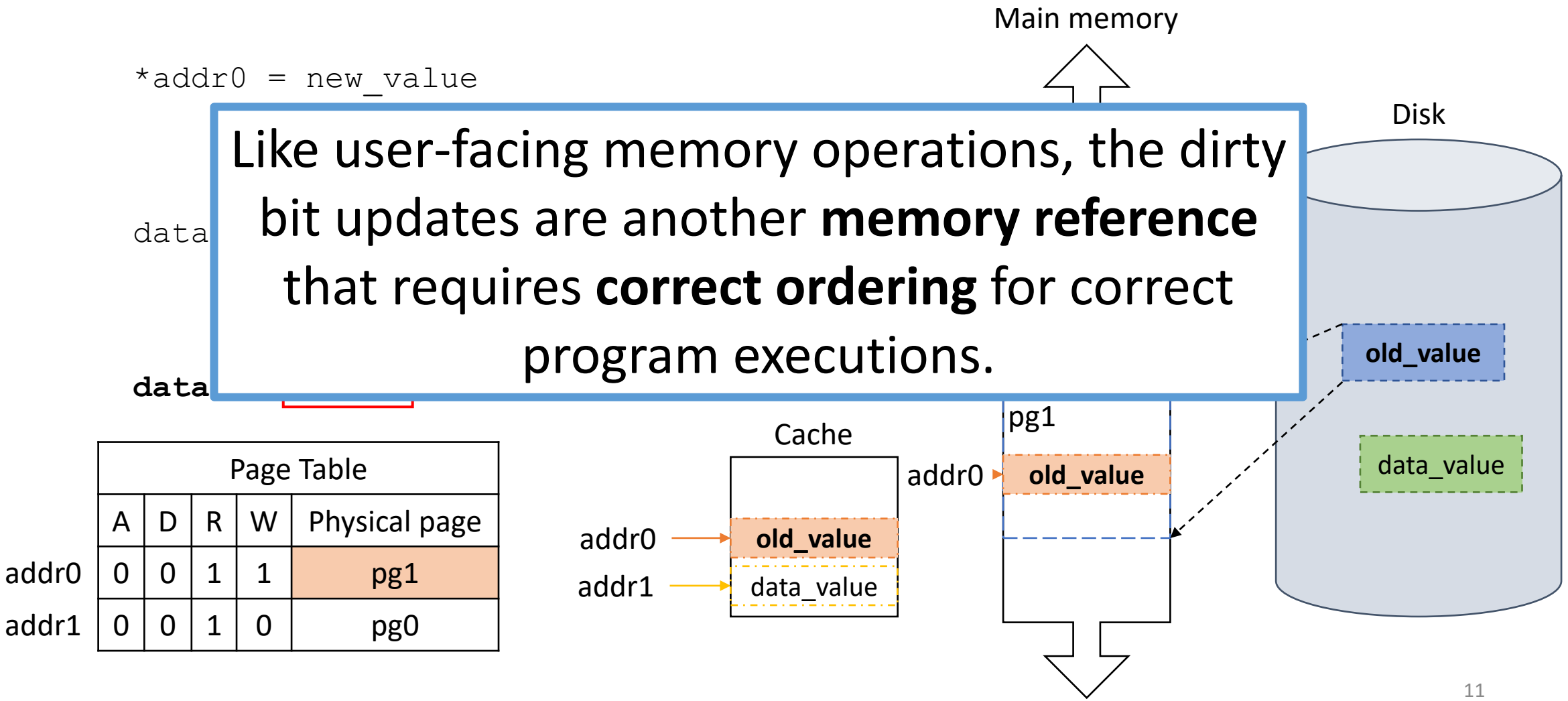
What if dirty bit is not updated before page swapped to disk?

```
*addr0 = new_value
         *
         *
         *
data = *addr1
         *
         *
         *
data2 = *addr0
```

	Page Table				
	A	D	R	W	Physical page
addr0	0	0	1	1	disk
addr1	0	0	1	0	pg0



What if dirty bit is not updated before page swapped to disk?

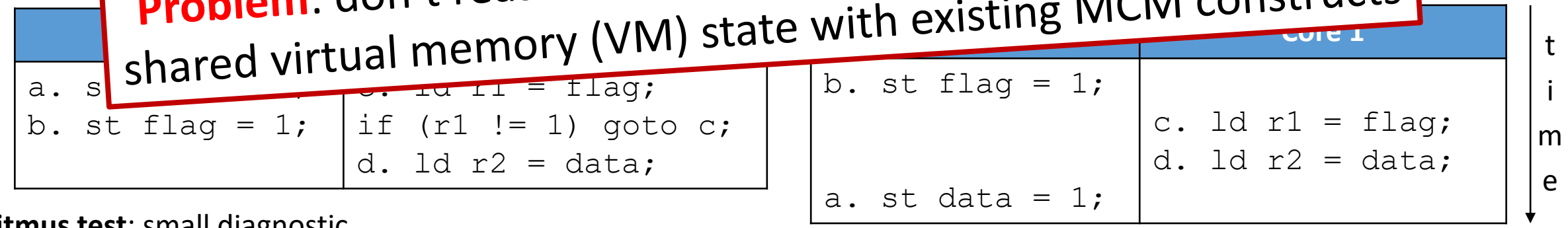


	Page Table				
	A	D	R	W	Physical page
addr0	0	0	1	1	pg1
addr1	0	0	1	0	pg0

Memory Consistency Models (MCMs) specify correct event orderings for concurrent programs

- MCMs specify rules for legal values that can be returned when software loads from memory on a shared memory system

Problem: don't reason about program executions impacted by shared virtual memory (VM) state with existing MCM constructs

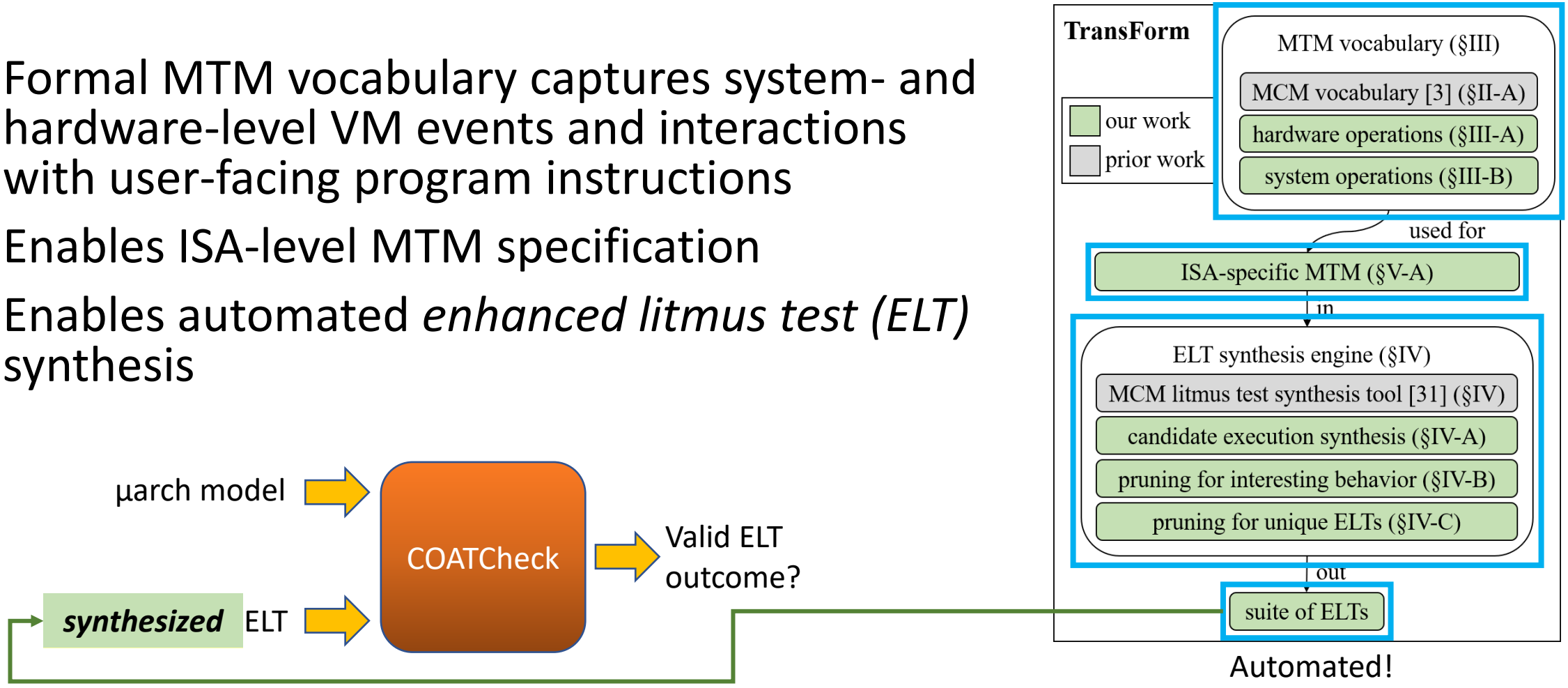


Litmus test: small diagnostic

This work: Memory Transistency Models (MTMs) – the superset of MCMs that additionally capture VM-aware ordering specifications

TransForm introduces constructs for ISA-level MTM specification and ELT synthesis

- Formal MTM vocabulary captures system- and hardware-level VM events and interactions with user-facing program instructions
- Enables ISA-level MTM specification
- Enables automated *enhanced litmus test (ELT)* synthesis



Allows for verification against *formally specified* MTM

Outline

- Background on ISA-level MCM vocabulary
- Background on virtual memory systems

Prior Work

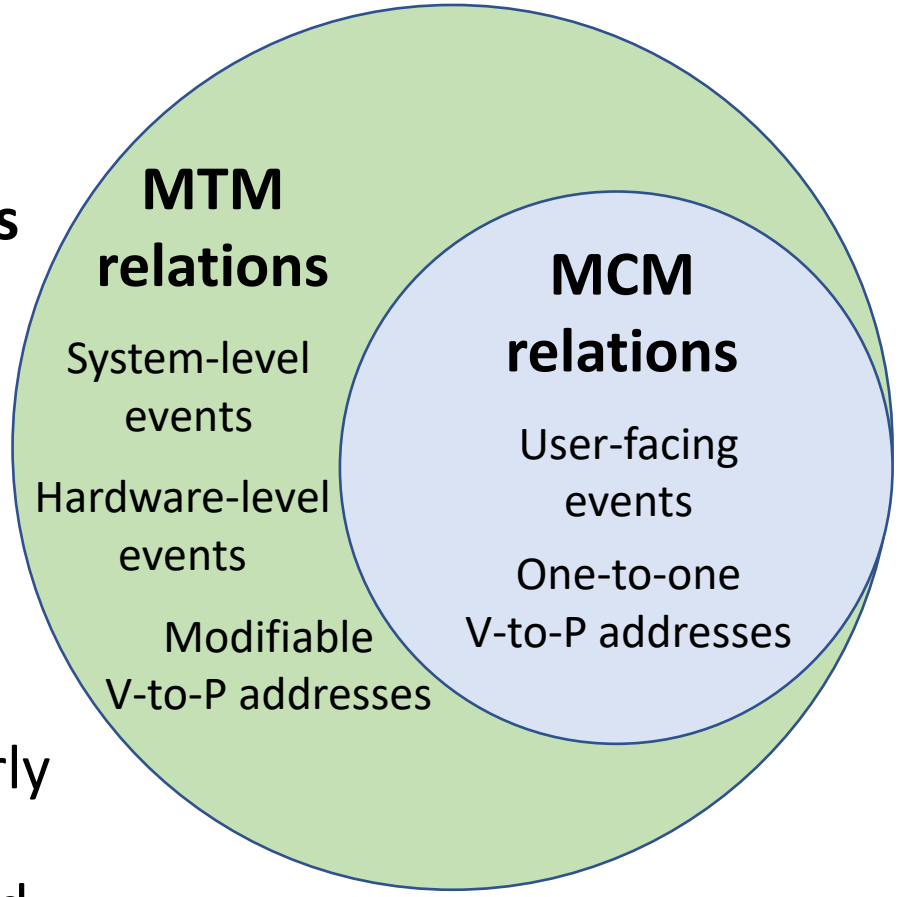
- Novel ISA-level MTM vocabulary
- Automating synthesis of ELTs
- Case Study: an estimated MTM for x86

My Work

- Conclusions

Approach to defining vocabulary for formally reasoning about MTMs

- MCMs can be defined **axiomatically**
  - Axiomatic MCM specifications use sets of **relations** that can describe user-facing program executions
  - MCM relations describe user-facing event executions of programs with one-to-one V-to-P address mappings
- MTMs are *superset* of MCMs
  - Axiomatic MTM specifications can use MCM relations but require additional relations to similarly describe transistency events and V-to-P address mappings that can have synonyms and be modified



ISA-level MCM relations can describe programs and their *candidate executions*

Candidate executions – set of possible executions of a program and their outcomes

Program

Core 0	Core 1
W_0 x	R_2 y
W_1 y	R_3 x

Instructions

Event = {W_0, W_1, R_2, R_3}

MemoryEvent = {W_0, W_1, R_2, R_3}

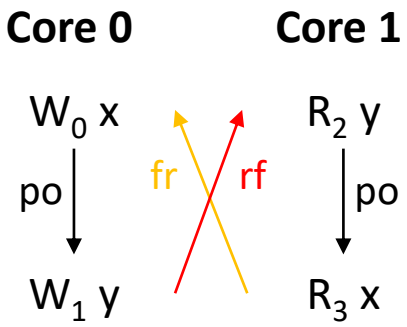
Location = {x, y}

address {W_0 → x, W_1 → y, R_2 → y, R_3 → x}

program order (po)

po = {W_0 → W_1, R_2 → R_3}

Graph



Candidate execution

Core 0	Core 1
W_0 x = 1	R_2 y = 1
W_1 y = 1	R_3 x = 0

Communication (com) relations

reads from (rf)

rf = {W_1 → R_2}

coherence order (co)

co = {}

from reads (fr)

fr = {R_3 → W_0}

Accessed data (outcome) symbolically represented by com relations

[Shasha & Snir, 1988]
[Alglave et al., 2014]

ISA-level MCM relations can describe programs and their *candidate executions*

Candidate executions – set of possible executions of a program and their outcomes

Program

Core 0	Core 1
W_0 x	R_2 y
W_1 y	R_3 x

Instructions

Event = {W_0, W_1, R_2, R_3}

MemoryEvent = {W_0, W_1, R_2, R_3}

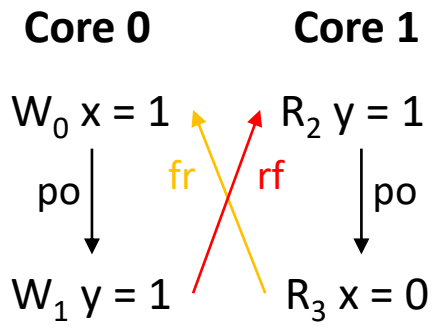
Location = {x, y}

address {W_0 → x, W_1 → y, R_2 → y, R_3 → x}

program order (po)

po = {W_0 → W_1, R_2 → R_3}

Graph



Candidate execution

Core 0	Core 1
W_0 x = 1	R_2 y = 1
W_1 y = 1	R_3 x = 0

Communication (com) relations

reads from (rf)

rf = {W_1 → R_2}

coherence order (co)

co = {}

from reads (fr)

fr = {R_3 → W_0}

Accessed data (outcome) symbolically represented by com relations

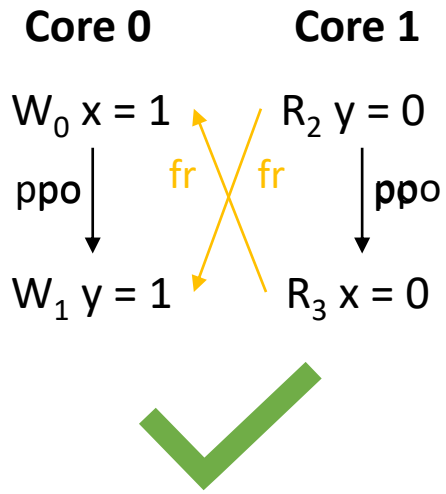
[Shasha & Snir, 1988]
[Alglave et al., 2014]

MCM specifications place constraints on permitted execution behaviors

Axiomatic MCM specifications use MCM relations to describe axioms that constrain candidate execution behaviors

Intel x86 processors use the **total store order (TSO)** memory model (**x86-TSO**) [Owens et al., 2009]: strict sequential memory access orderings but relaxed Store->Load orderings to allow for store buffering

Causality – axiom for x86-TSO:
acyclic(rfe + co + fr + ppo + fence)



MCM specifications place constraints on permitted execution behaviors

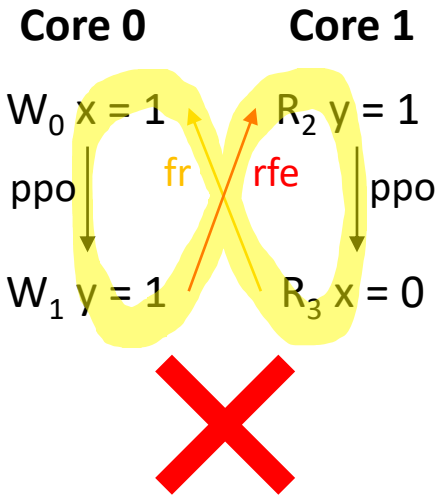
Axiomatic MCM specifications use MCM relations to describe axioms that constrain candidate execution behaviors

Sequential consistency (SC) [Lamport, 1979]: outcome must be representative of executing instructions in order

Intel x86 processors use the **total store order (TSO)** memory model (**x86-TSO**) [Owens et al., 2009]: like SC but relaxes Store->Load orderings to allow for *store buffering*

Causality – axiom for x86-TSO:
acyclic(rfe + co + fr + ppo + fence)

mp ("message passing") litmus test



Outline

- Background on ISA-level MCM vocabulary
- Background on virtual memory systems

Prior Work

- Novel ISA-level MTM vocabulary
- Automating synthesis of ELTs
- Case Study: an estimated MTM for x86

My Work

- Conclusions

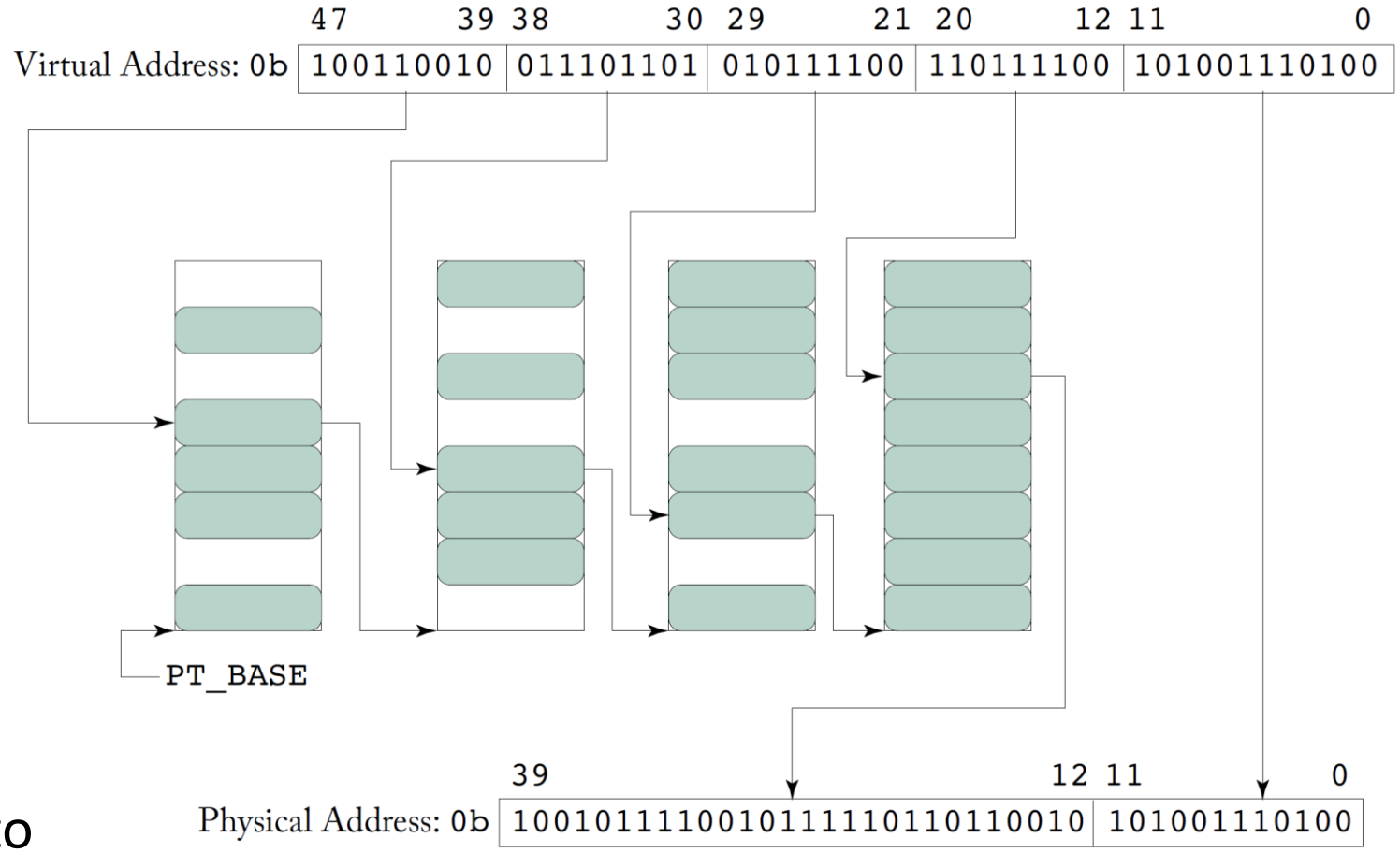
V-to-P address mappings need to be stored and accessed during memory events

V-to-P address mappings are stored in **page tables**.

Page table entries (PTEs) hold address mapping and status bits (permissions, access, dirty).

Page tables are usually structured hierarchically.

When address translation is needed, a **page table walk** traverses the page table levels to find the desired address mapping.

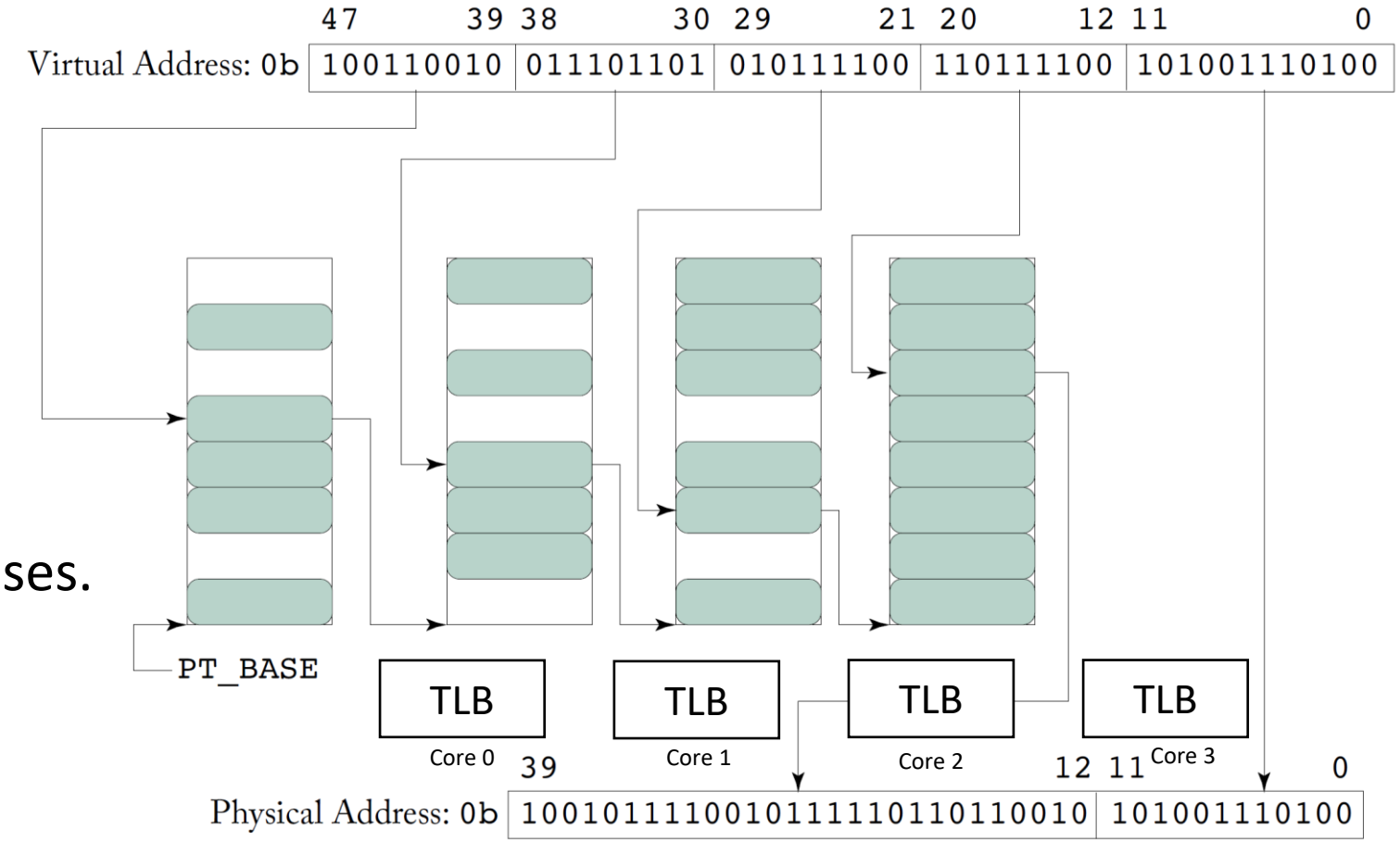


A. Bhattacharjee and D. Lustig, "Architectural and operating system support for virtual memory", *Synthesis Lectures on Computer Architecture*, 2017.

V-to-P address mappings are cached in the *translation lookaside buffer* (TLB)

Hierarchical page tables require **additional** memory accesses during address translation – big performance hit.

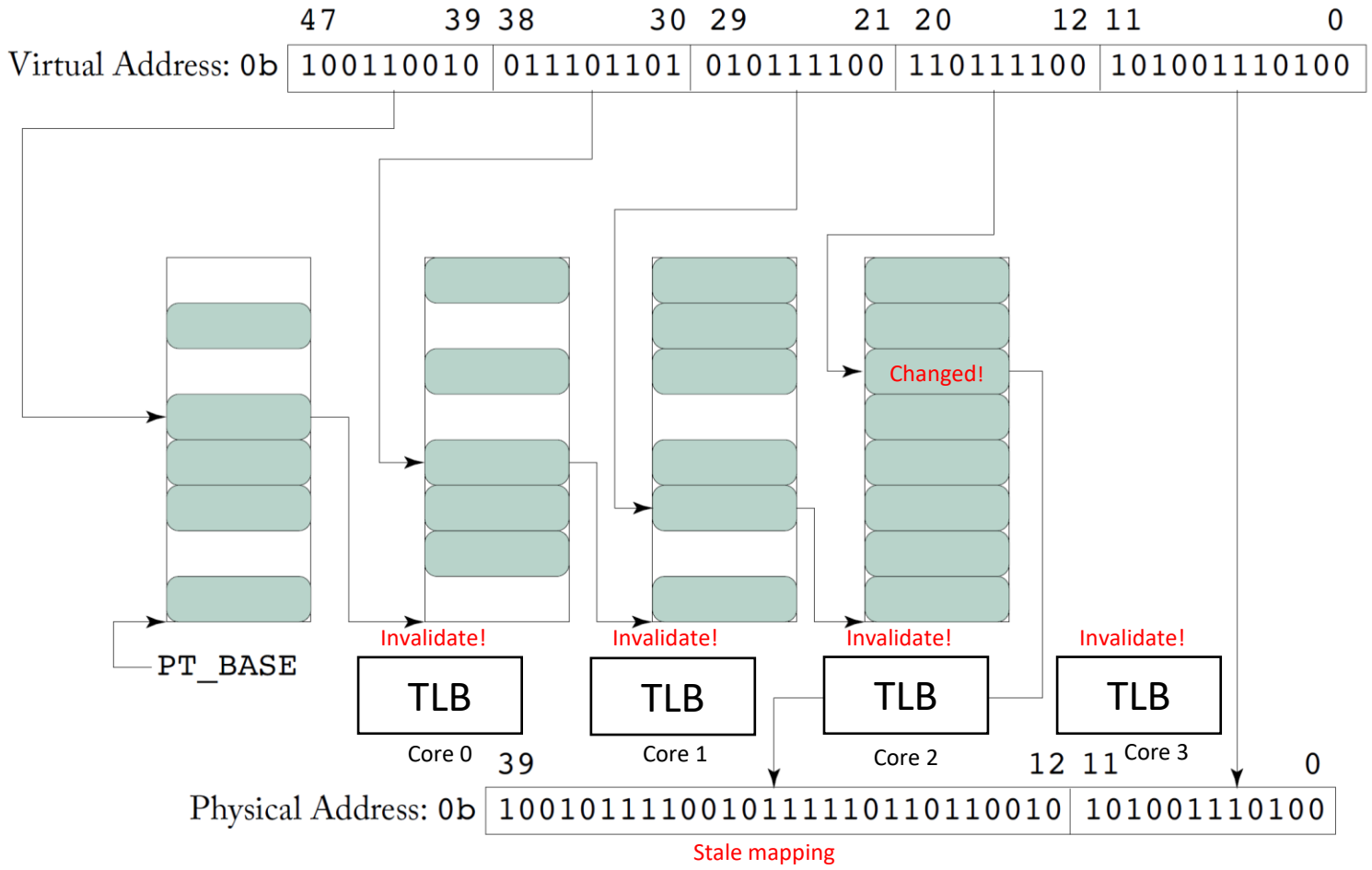
Page mappings cached in TLB to reduce latency of memory accesses.



V-to-P address mappings can be changed by OS

Operating system (OS) may change address mappings in page tables.

Corresponding TLB entries must be invalidated on *each core* to prevent stale mapping accesses.

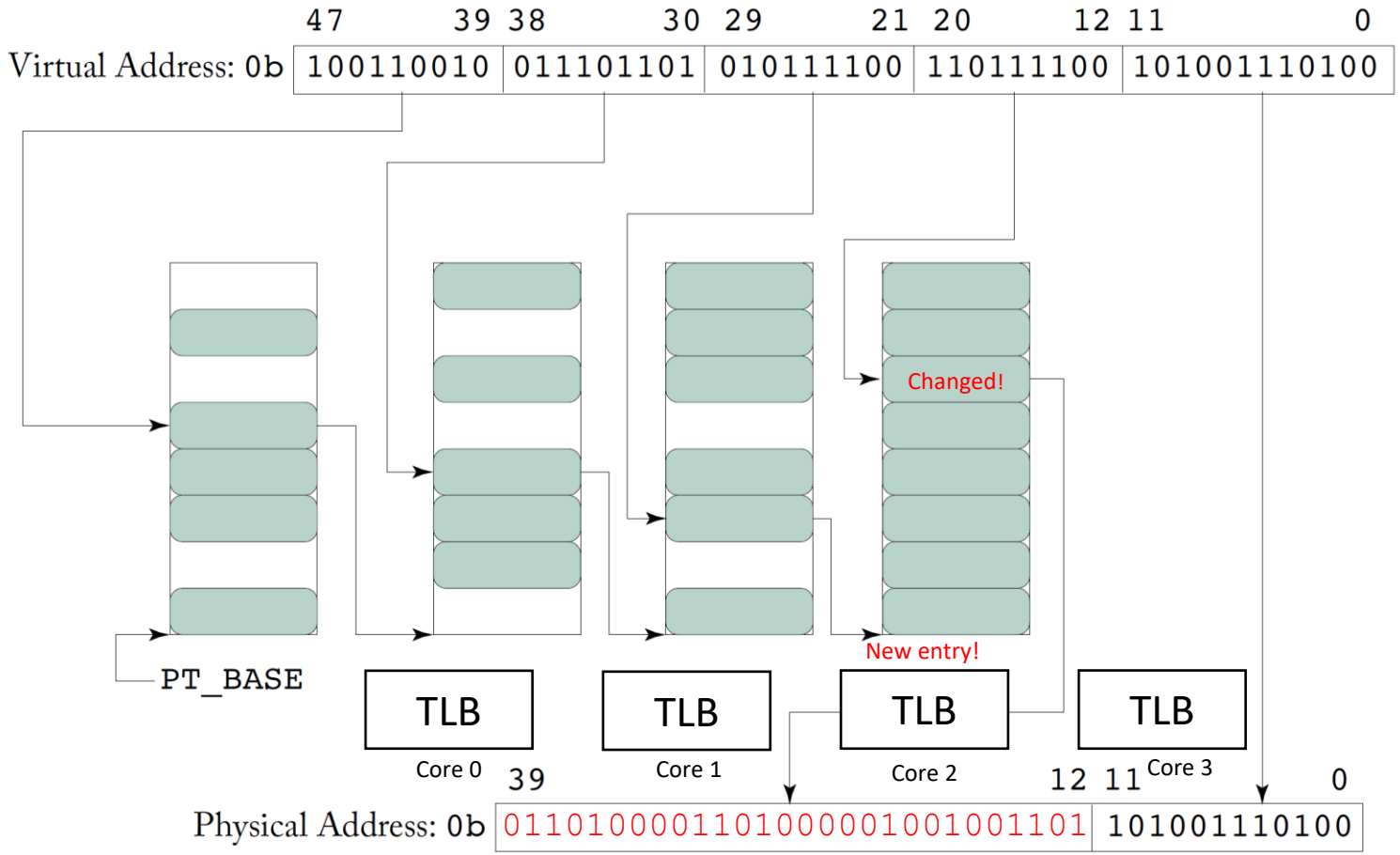


V-to-P address mappings can be changed by OS

Operating system (OS) may change address mappings in page tables.

Corresponding TLB entries must be invalidated on *each core* to prevent stale mapping accesses.

New page table walk needed to load new mapping into TLB.



Virtual memory events TransForm needs to support

Hardware-level events	System-level events
Page table walk Loads TLB entries on memory access	Address mapping changes V-to-P address mapping must be modifiable like data
PTE status bit updates TransForm supports dirty bit updates on memory stores	TLB entry invalidations May be invoked on multiple cores by address mapping changes

Outline

- Background on ISA-level MCM vocabulary
- Background on virtual memory systems
- Novel ISA-level MTM vocabulary
- Automating synthesis of ELTs
- Case Study: an estimated MTM for x86
- Conclusions

Prior Work

My Work

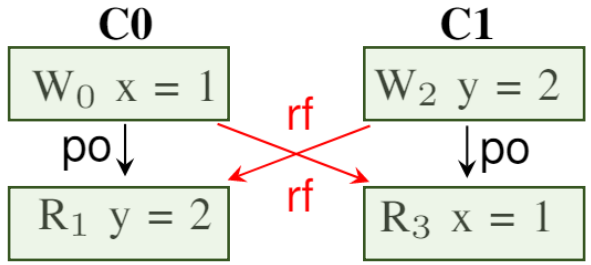
MTM Vocabulary: Hardware-level events

TransForm supports **page table walks (PTW)** and **dirty bit updates**

Ghost instructions

PTW: loads translation lookaside buffer (TLB) entry

dirty bit update: modifies dirty bit in PTE



MTM Vocabulary: Hardware-level events

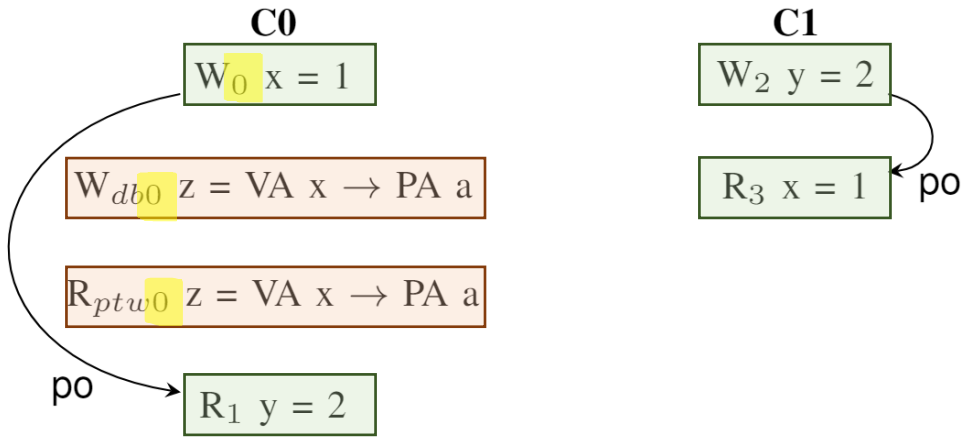
TransForm supports **page table walks (PTW)** and **dirty bit updates**

Ghost instructions

PTW: loads translation lookaside buffer (TLB) entry

dirty bit update: modifies dirty bit in PTE

ghost – relates user-facing MemoryEvent to invoked ghost instructions (numerical subscripts)



MTM Vocabulary: Hardware-level events

TransForm supports **page table walks (PTW)** and **dirty bit updates**

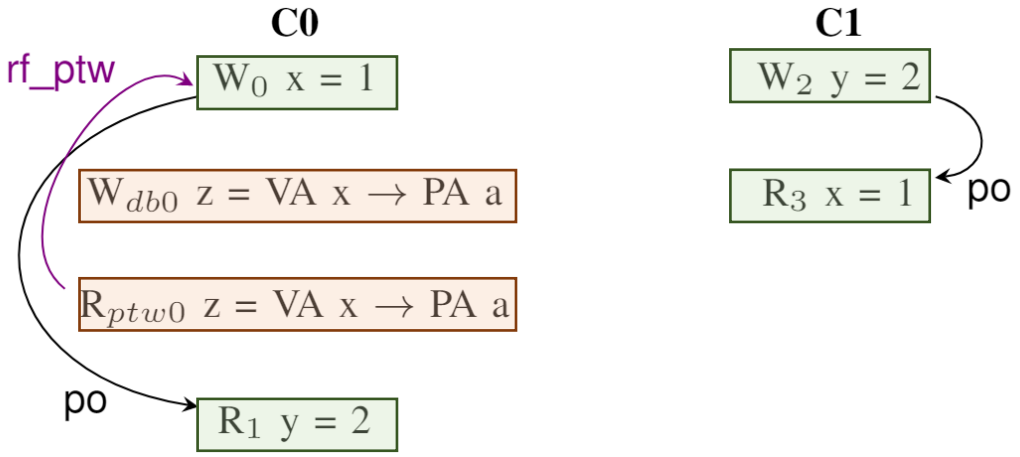
Ghost instructions

PTW: loads translation lookaside buffer (TLB) entry

dirty bit update: modifies dirty bit in PTE

ghost – relates user-facing MemoryEvent to invoked ghost instructions (numerical subscripts)

rf_ptw – relates PTW to user-facing MemoryEvents that access loaded TLB entry



MTM Vocabulary: Hardware-level events

TransForm supports **page table walks (PTW)** and **dirty bit updates**

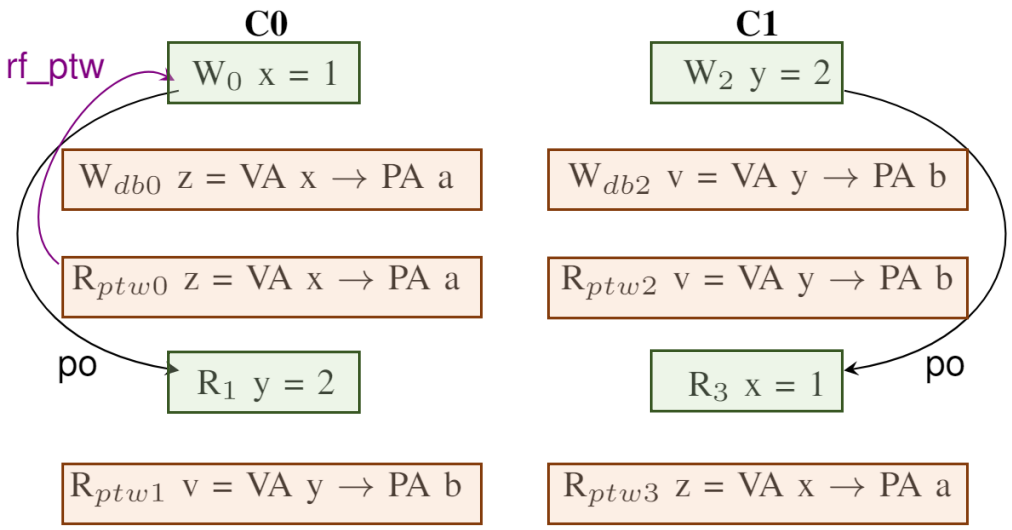
Ghost instructions

PTW: loads translation lookaside buffer (TLB) entry

dirty bit update: modifies dirty bit in PTE

ghost – relates user-facing MemoryEvent to invoked ghost instructions (numerical subscripts)

rf_ptw – relates PTW to user-facing MemoryEvents that access loaded TLB entry



MTM Vocabulary: Hardware-level events

TransForm supports **page table walks (PTW)** and **dirty bit updates**

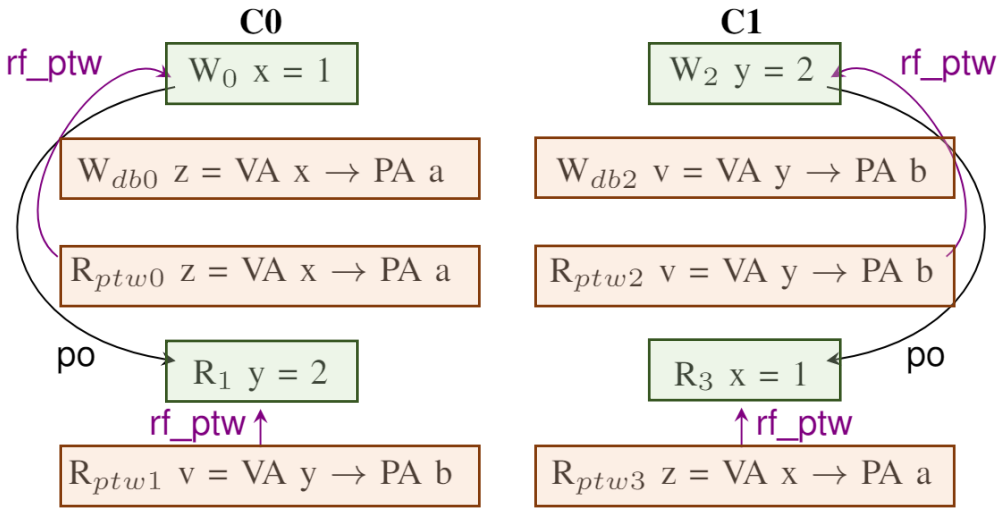
Ghost instructions

PTW: loads translation lookaside buffer (TLB) entry

dirty bit update: modifies dirty bit in PTE

ghost – relates user-facing MemoryEvent to invoked ghost instructions (numerical subscripts)

rf_ptw – relates PTW to user-facing MemoryEvents that access loaded TLB entry



MTM Vocabulary: Hardware-level events

TransForm supports **page table walks (PTW)** and **dirty bit updates**

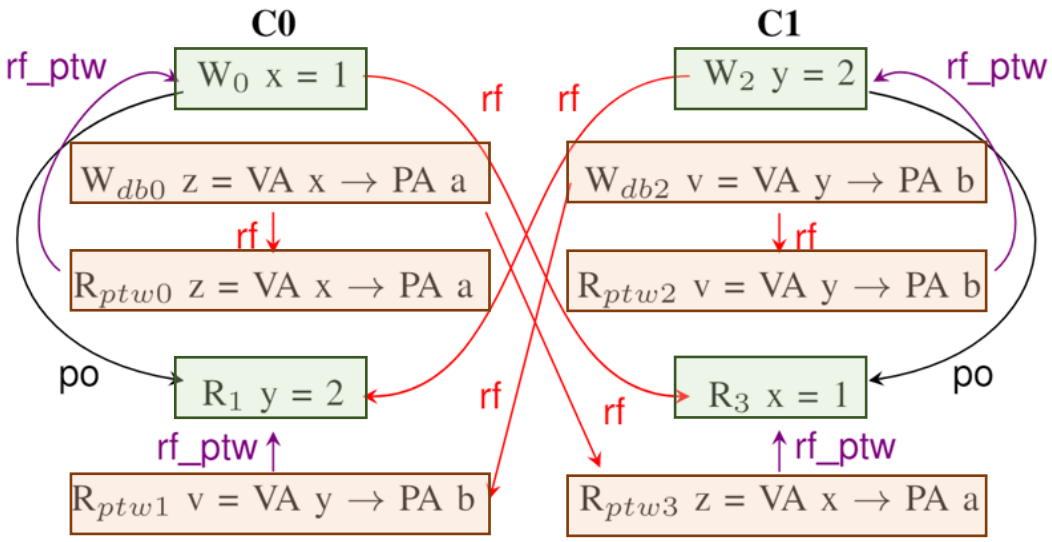
Ghost instructions

PTW: loads translation lookaside buffer (TLB) entry

dirty bit update: modifies dirty bit in PTE

ghost – relates user-facing MemoryEvent to invoked ghost instructions (numerical subscripts)

rf_ptw – relates PTW to user-facing MemoryEvents that access loaded TLB entry

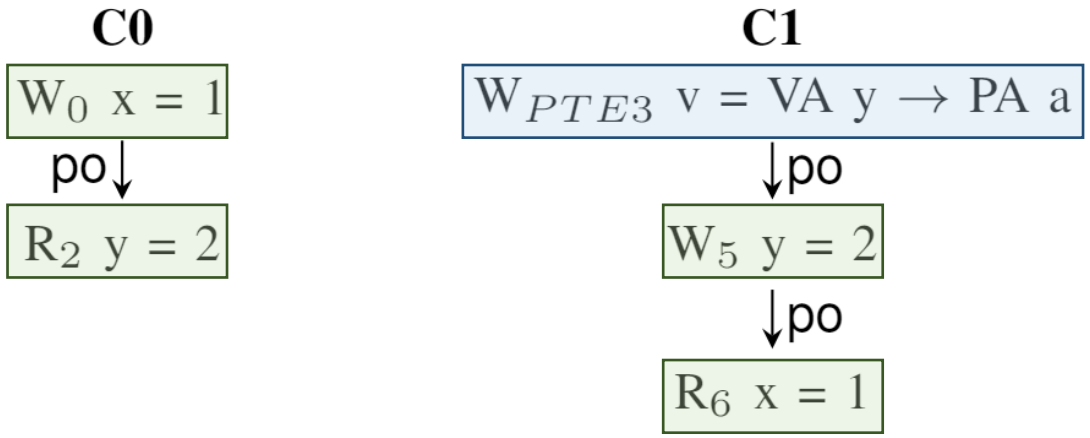


MTM Vocabulary: System-level events

TransForm supports **address remappings via PTE Writes** and **TLB entry invalidations**

Support instructions

PTE Write: changes address mapping stored in a PTE for some VA v



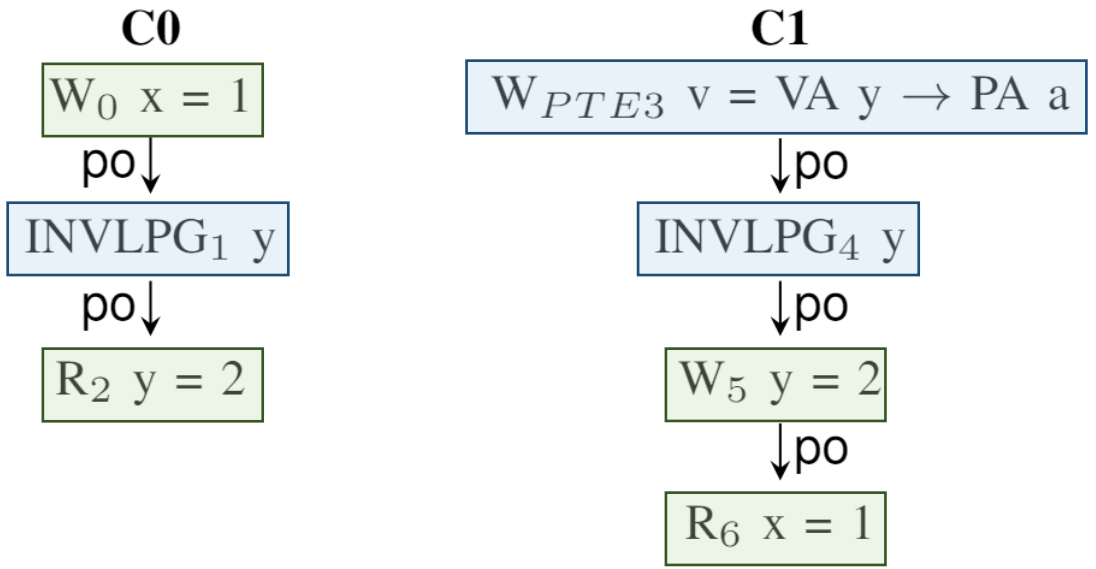
MTM Vocabulary: System-level events

TransForm supports **address remappings via PTE Writes** and **TLB entry invalidations**

Support instructions

PTE Write: changes address mapping stored in a PTE for some VA v

INVLPG: invalidates TLB entry



MTM Vocabulary: System-level events

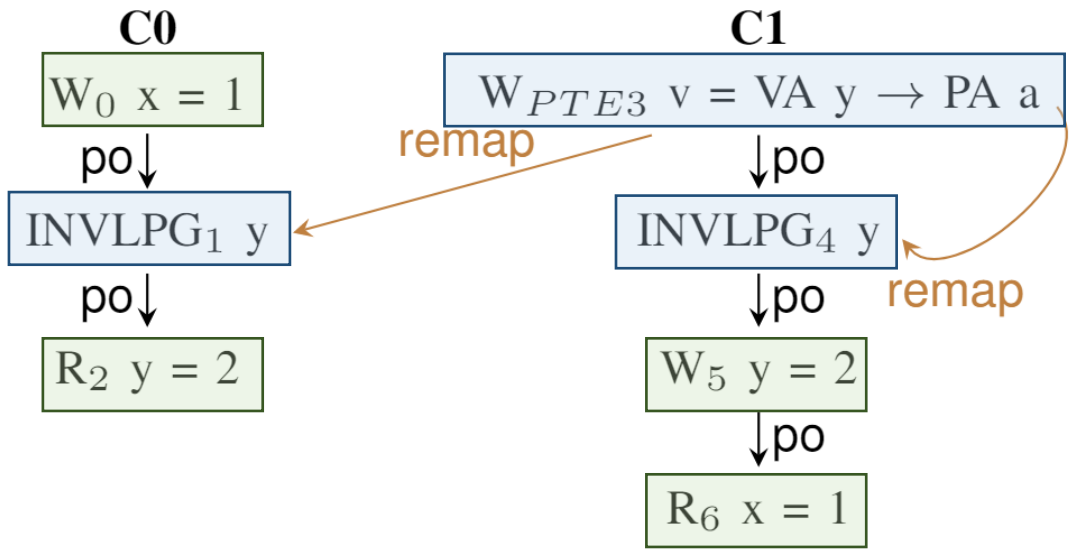
TransForm supports **address remappings via PTE Writes** and **TLB entry invalidations**

Support instructions

PTE Write: changes address mapping stored in a PTE for some VA v

INVLPG: invalidates TLB entry

remap – relates PTE Writes to invoked INVLPGs



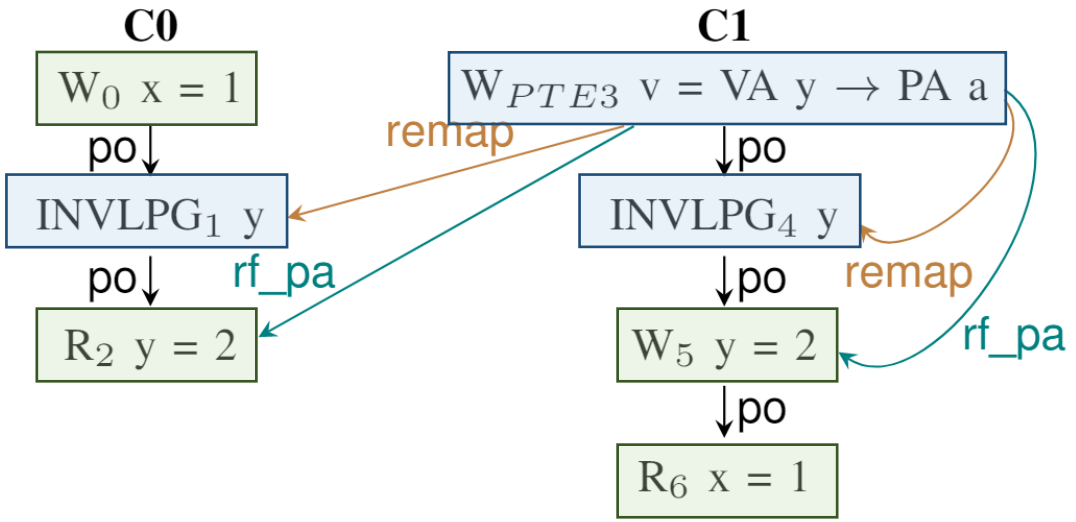
MTM Vocabulary: System-level events

TransForm supports **address remappings via PTE Writes** and **TLB entry invalidations**

Support instructions

PTE Write: changes address mapping stored in a PTE for some VA v

INVLPG: invalidates TLB entry



remap – relates PTE Writes to invoked INVLPGs

rf_pa – relates PTE Write for VA v → PA p to user-facing MemoryEvents accessing PA p via VA v

fr_pa – relates user-facing MemoryEvents accessing PA p via VA v to PTE Writes for VA v' → PA p

co_pa and **fr_va** follow similarly

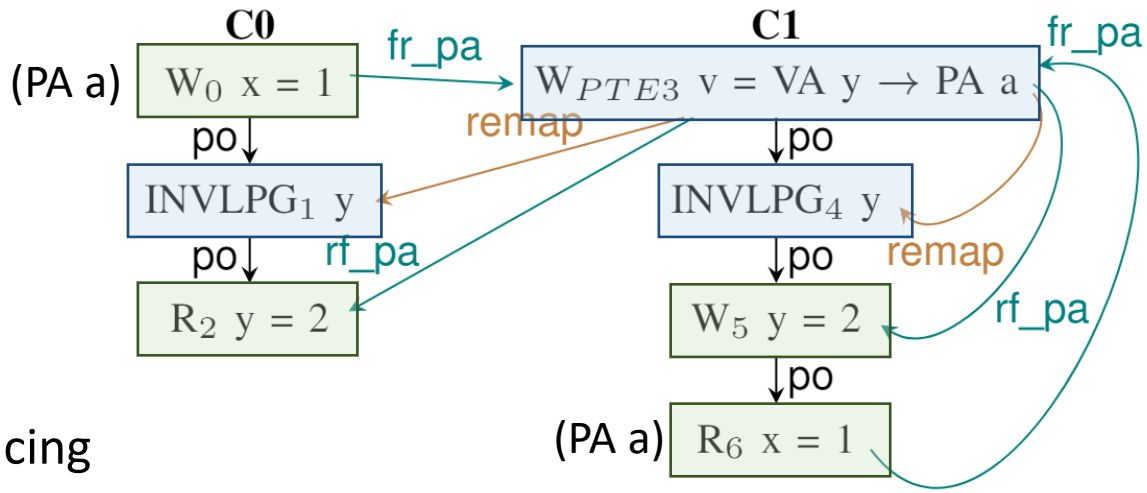
MTM Vocabulary: System-level events

TransForm supports **address remappings via PTE Writes** and **TLB entry invalidations**

Support instructions

PTE Write: changes address mapping stored in a PTE for some VA v

INVLPG: invalidates TLB entry



remap – relates PTE Writes to invoked INVLPGs

rf_pa – relates PTE Write for VA v → PA p to user-facing MemoryEvents accessing PA p via VA v

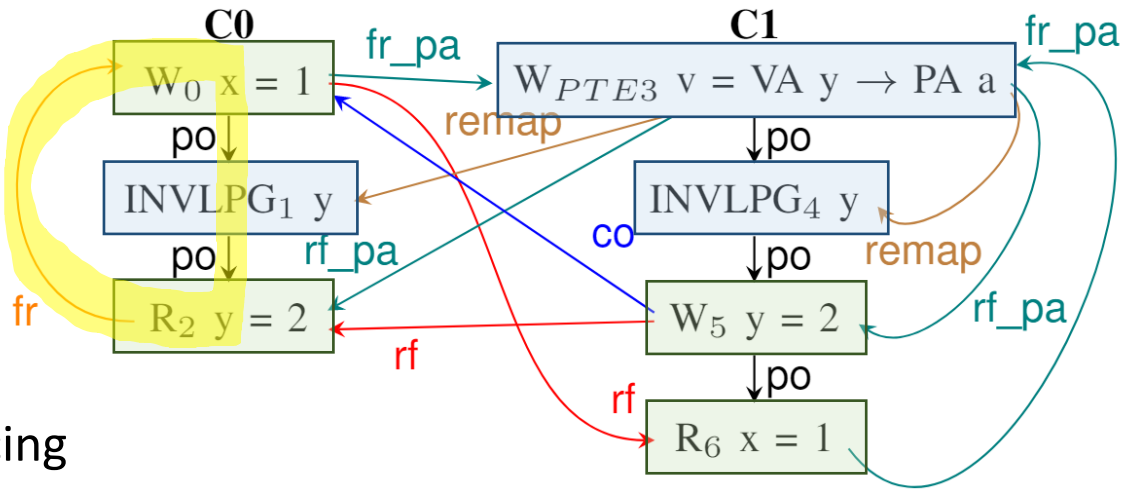
fr_pa – relates user-facing MemoryEvents accessing PA p via VA v to PTE Writes for VA v' → PA p

co_pa and **fr_va** follow similarly

MTM Vocabulary: System-level events

TransForm supports **address remappings via PTE Writes** and **TLB entry invalidations**

These new com relations can be used to derive same PA accesses.

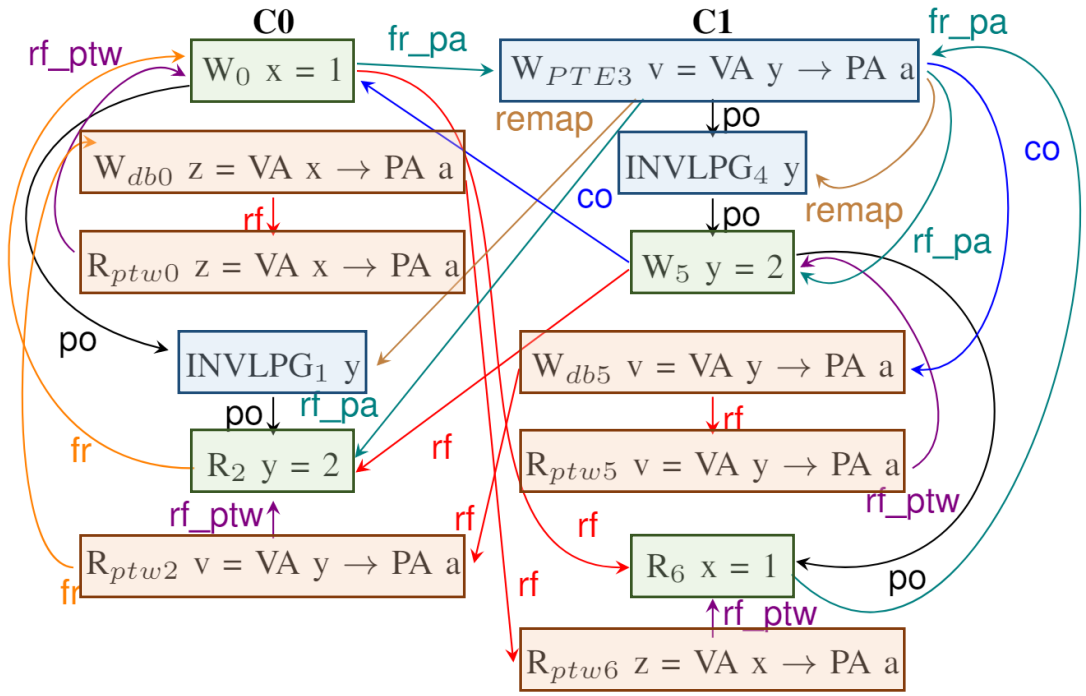


Forbidden!

rf_pa – relates PTE Write for VA v → PA p to user-facing MemoryEvents accessing PA p via VA v
fr_pa – relates user-facing MemoryEvents accessing PA p via VA v to PTE Writes for VA v' → PA p
co_pa and **fr_va** follow similarly

MTM Vocabulary: Putting it all together

Program executions with transistency events and relations can get quite complex but they allow us to capture these additional interactions that can occur and impact the program's execution.



Outline

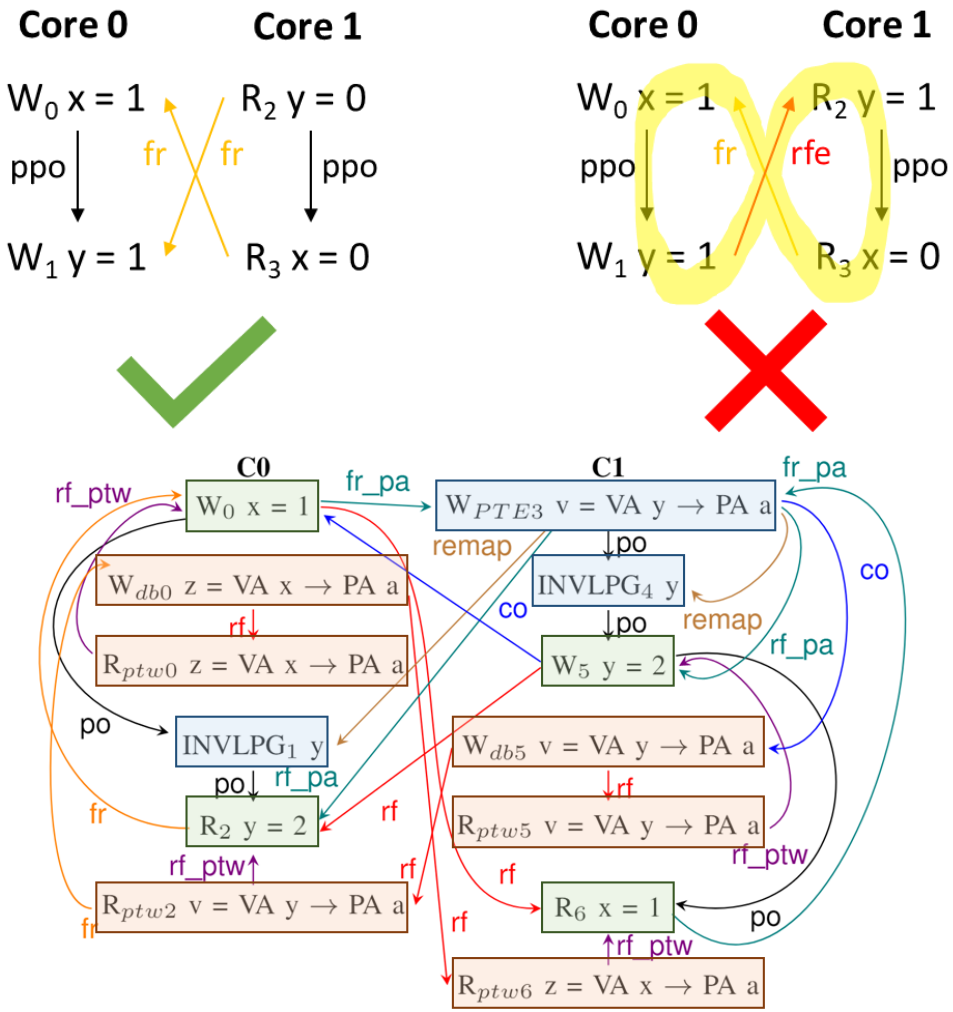
- Background on ISA-level MCM vocabulary
- Background on virtual memory systems
- Novel ISA-level MTM vocabulary
- Automating synthesis of ELTs
- Case Study: an estimated MTM for x86
- Conclusions

Prior Work

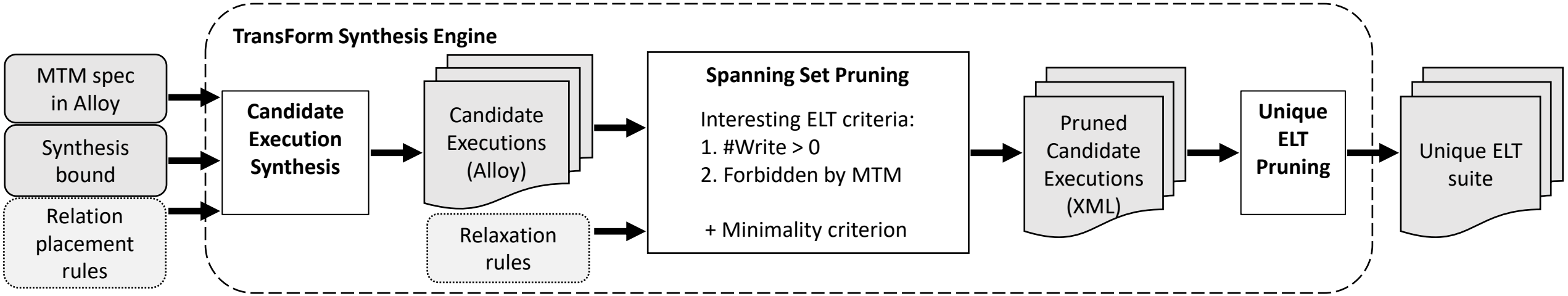
My Work

MCMs can be verified with litmus tests. MTMs can be verified with *enhanced* litmus tests.

- **Litmus tests**: small diagnostic programs for validating MCM behaviors
  - Executions and their outcomes can be deemed **permitted** or **forbidden** by MCM specification
- **Enhanced litmus tests (ELTs)**: litmus tests enhanced with system- and hardware-level events that facilitate address translation

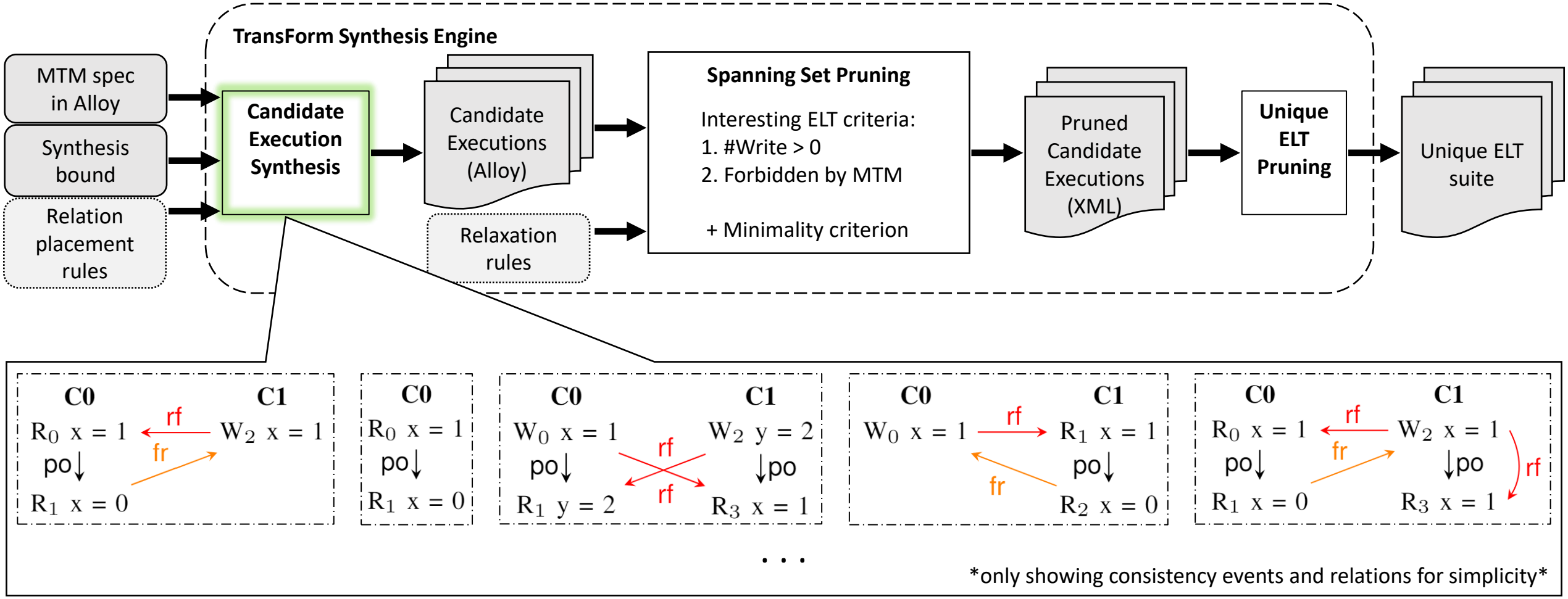


From Specification to Test Synthesis

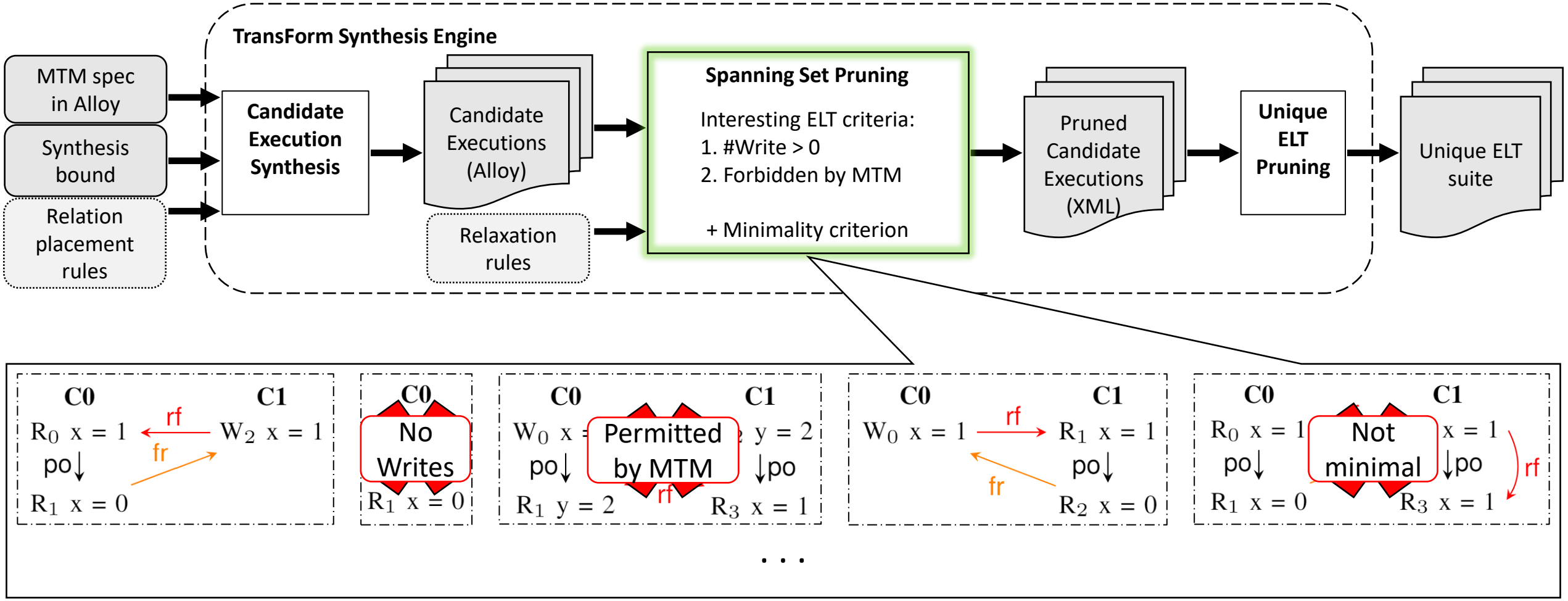


- ELTs can be described with MTM relations and support verification against an MTM spec
  - Goals:
    - Automated
    - Interesting and minimal (“Spanning set”)
    - Deduplicated
    - Comprehensive (to a bound)

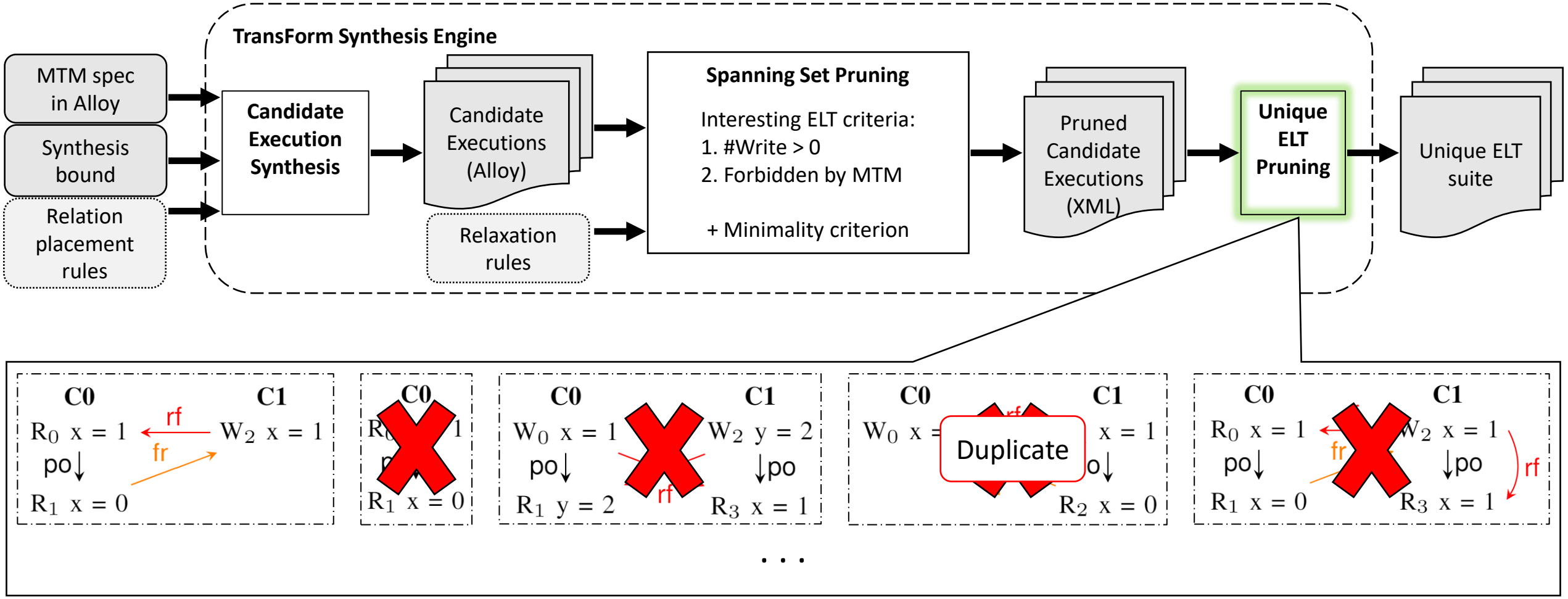
TransForm's synthesis engine starts by synthesizing all possible candidate executions up to a bound



Candidate executions are pruned for interesting ELT behaviors and checked for minimality



Unique ELTs are found by deduplicating synthesized ELTs with a post-processing script



Outline

- Background on ISA-level MCM vocabulary
- Background on virtual memory systems

Prior Work

- Novel ISA-level MTM vocabulary
- Automating synthesis of ELTs
- Case Study: an estimated MTM for x86

My Work

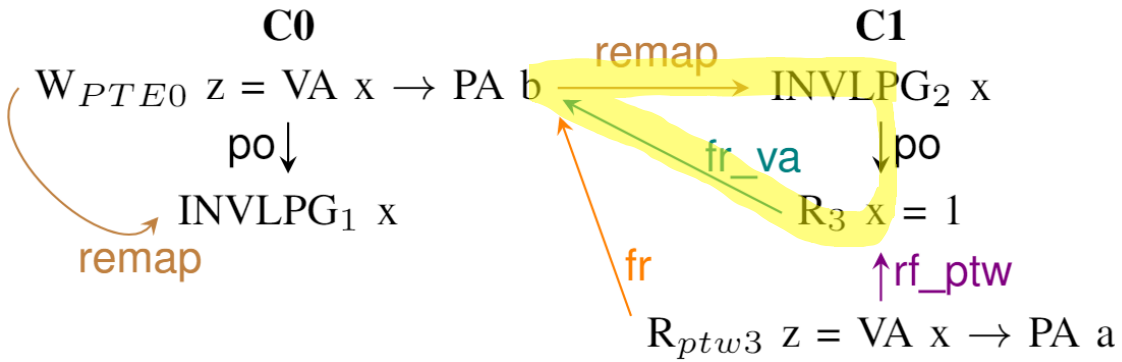
- Conclusions

x86t_elt transistency predicates are composed of TSO axioms and new transistency-specific axioms

- **x86t_elt**: an approximate x86 transistency model based on prior work and publicly available documentation
- **x86-TSO**: sc_per_loc, rmw_atomicity, causality

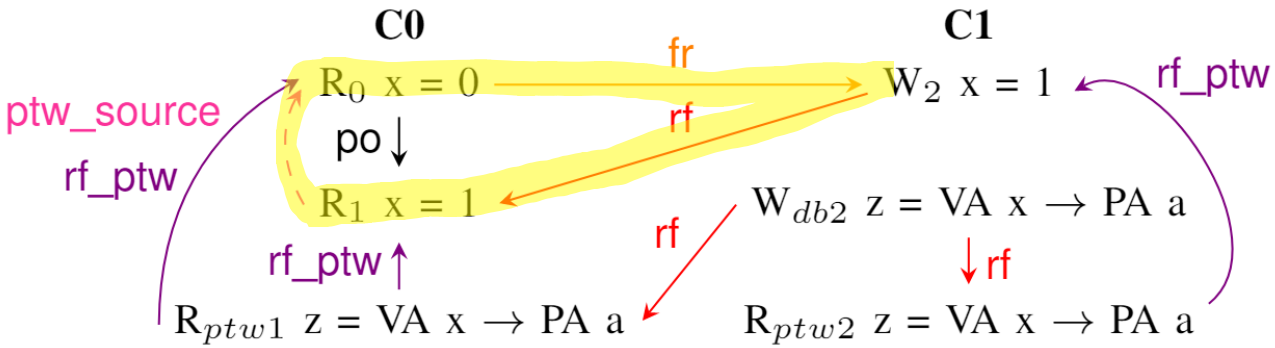
invlpg (required)

acylic[fr_va + remap + ^po]

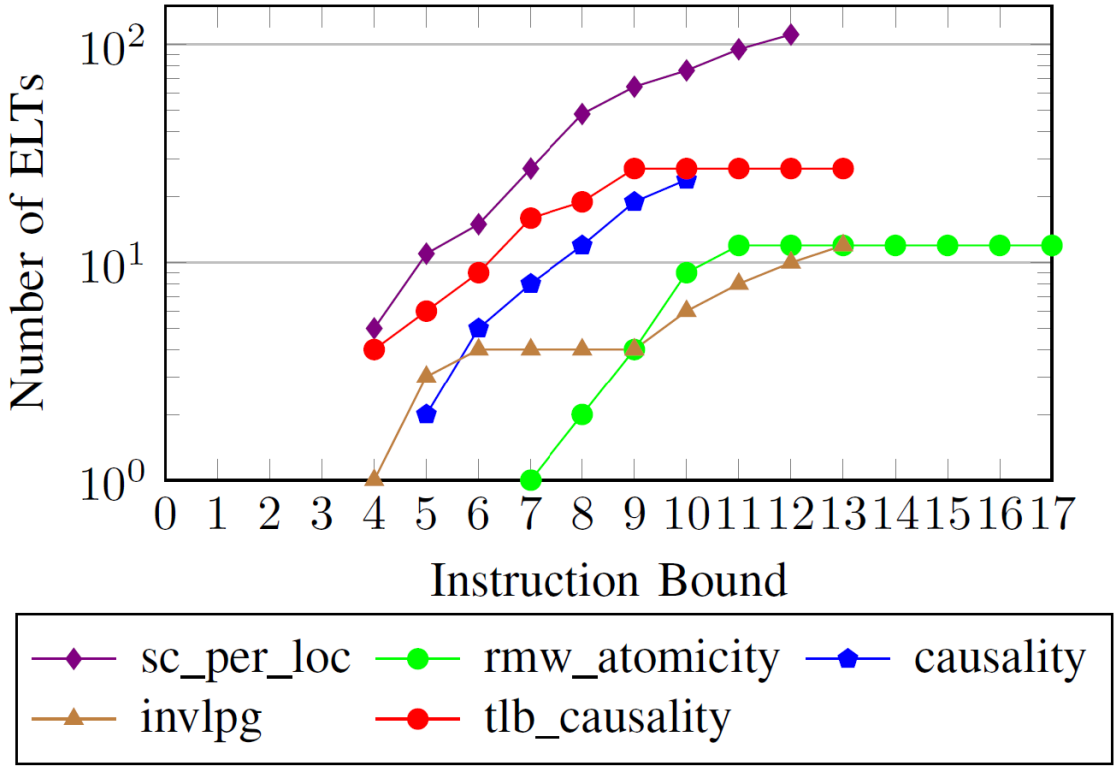


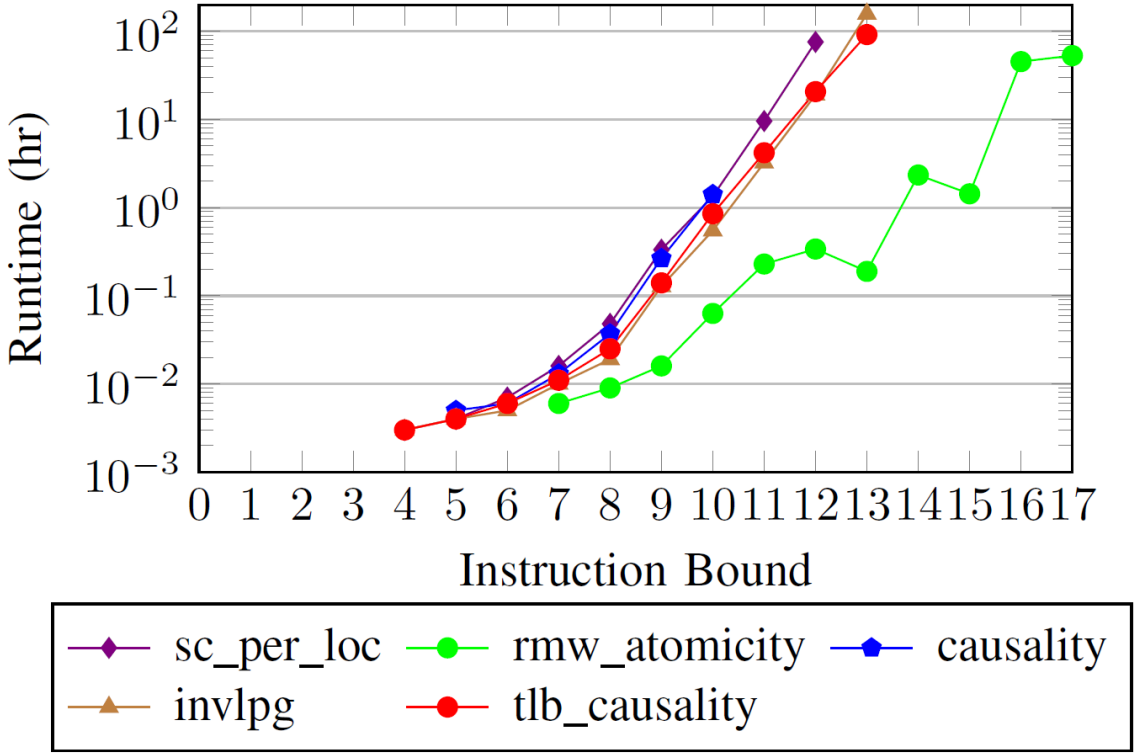
tlb_causality (auxiliary)

acyclic[ptw_source + com]



A per-axiom suite was synthesized for each **x86t_elt** axiom





140 total unique ELTs!

The synthesized x86t_elt suite consisted of all relevant ELTs from COATCheck and more

- All 22 relevant ELTs from COATCheck synthesized
  - 7 ELTs synthesized verbatim → map to 4 ELT programs in x86t_elt suite
  - 15 ELTs can be reduced to a minimal ELT that is synthesized
- 4 ELTs from COATCheck, 136 new ELTs

Conclusions

- **TransForm**: framework for formal specification of MTMs and ELT synthesis
- Enables modern ISAs to have a formal specification that includes VM
- Offers systems programmers and hardware designers a stronger opportunity for verification of full systems
- **Future work**:
  - Empirical MTM testing to validate/verify with x86t_elt
  - Specify other MTMs (e.g., RISC-V)
  - Model additional transistency interactions (e.g., updating permission bits)
- Available at: https://github.com/naorinh/TransForm

TransForm: Formally Specifying Transistency Models and Synthesizing Enhanced Litmus Tests

Naorin Hossain
Princeton University

FOCA 2020
October 30, 2020

https://github.com/naorinh/TransForm