Synthesizing Memory Models from Framework Sketches and Litmus Tests

James Bornholt
Emina Torlak
University of Washington

Memory consistency models define memory reordering behaviors on multiprocessors

Memory consistency models define memory reordering behaviors on multiprocessors

…correctness of my compiler…

Compiler writers

Memory consistency models define memory reordering behaviors on multiprocessors

…correctness of my compiler…

Compiler writers 👨‍💻

…rules to verify against…

Verification tools 🤖

Memory consistency models define memory reordering behaviors on multiprocessors

…correctness of my compiler…

Compiler writers 👨‍💻

…rules to verify against…

Verification tools 🤖

…possible low-level behaviors…

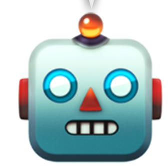
Kernel/library developers 👩‍🔬

Memory consistency models define memory reordering behaviors on multiprocessors

…correctness of my compiler…

Compiler writers 

…rules to verify against…

Verification tools 

…possible low-level behaviors…

Kernel/library developers 

Litmus tests and prose

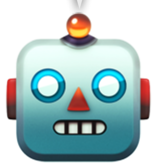
Memory consistency models define memory reordering behaviors on multiprocessors

...correctness of my compiler...

Compiler writers

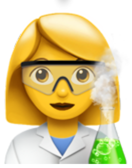
...rules to verify against...

Verification tools

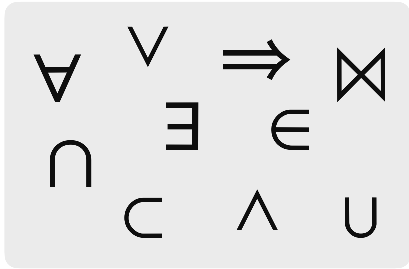


...possible low-level behaviors...

Kernel/library developers



Litmus tests and prose



Formal specifications

Memory consistency models define memory reordering behaviors on multiprocessors

...correctness of my compiler...

Compiler writers

...rules to verify against...

Verification tools

...possible low-level behaviors...

Kernel/library developers

Litmus tests and prose

x86 [Sewell et al, CACM'10] →

PowerPC [Alglave et al, CAV'10, etc] →

ARM [Flur et al, POPL'16] →

$\forall \quad \vee \quad \Rightarrow \quad \bowtie$

$\exists \quad \in$

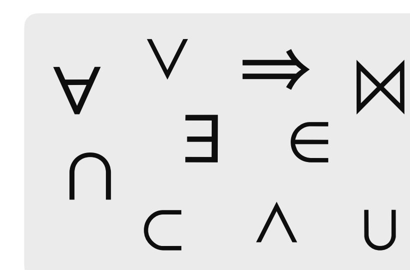
$\cap$

$\subset \quad \wedge \quad \cup$

Formal specifications

MemSynth

Litmus tests



Formal
specifications

MemSynth

Synthesize specifications

$\forall \quad \vee \quad \Rightarrow \quad \bowtie$

$\cap \quad \exists \quad \in$

$\subset \quad \wedge \quad \cup$

Litmus tests

Formal specifications

MemSynth

Framework sketch

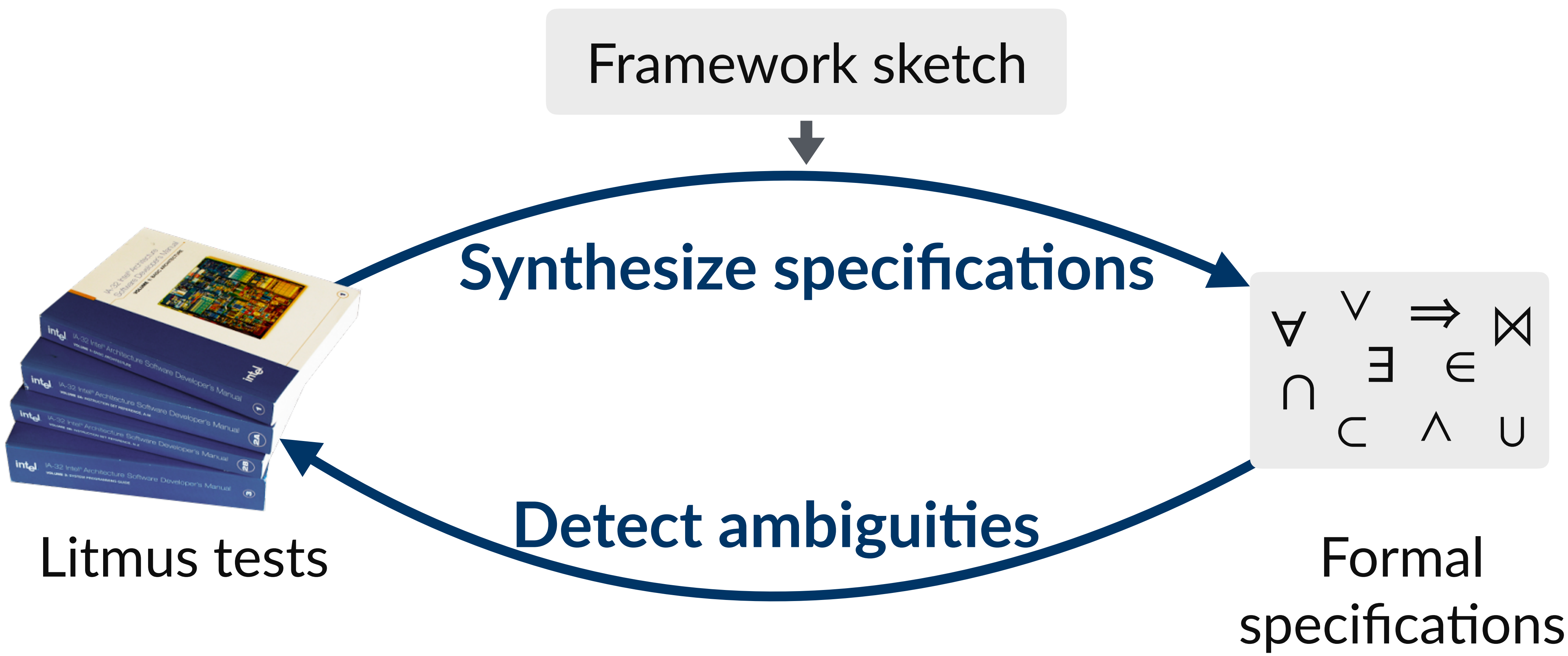
Synthesize specifications

Litmus tests

$\forall$ $\vee$ $\Rightarrow$ $\bowtie$
$\exists$ $\in$
$\cap$
$\subset$ $\wedge$ $\cup$

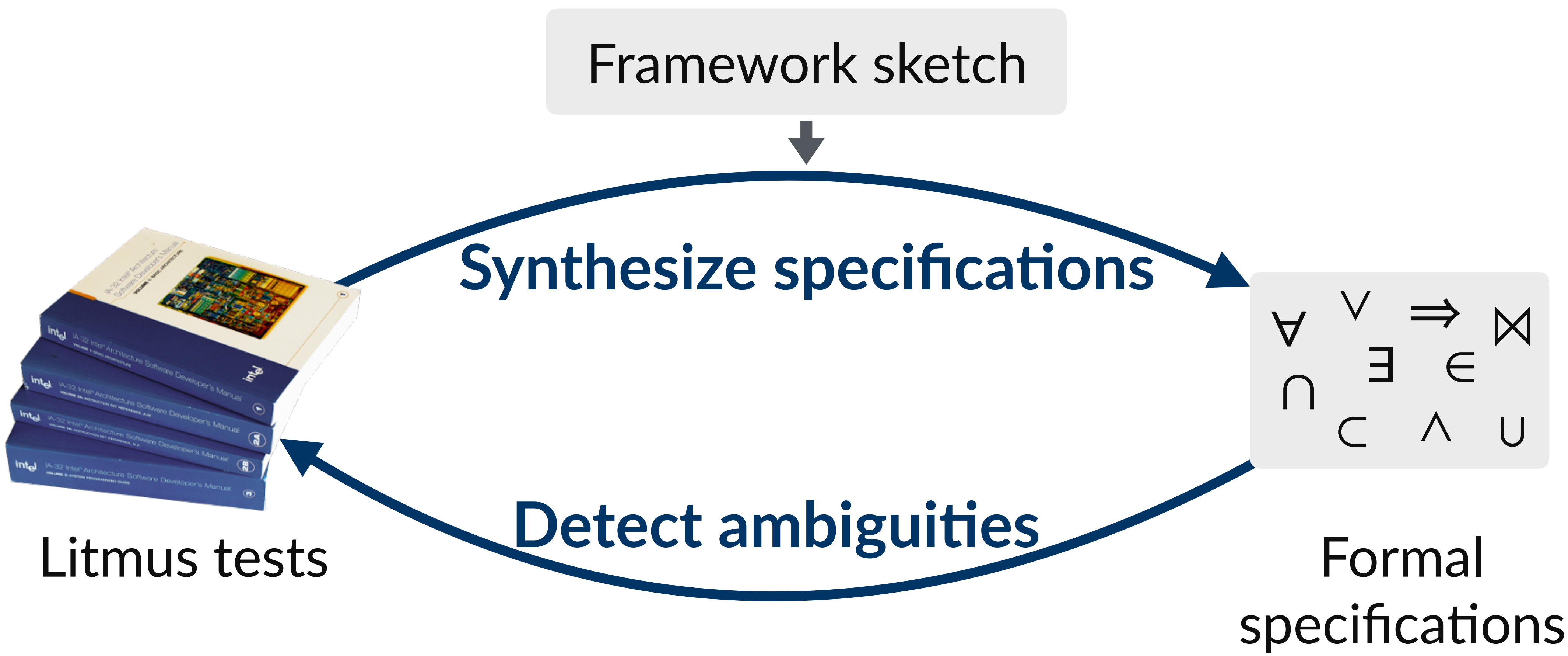
Formal specifications

MemSynth

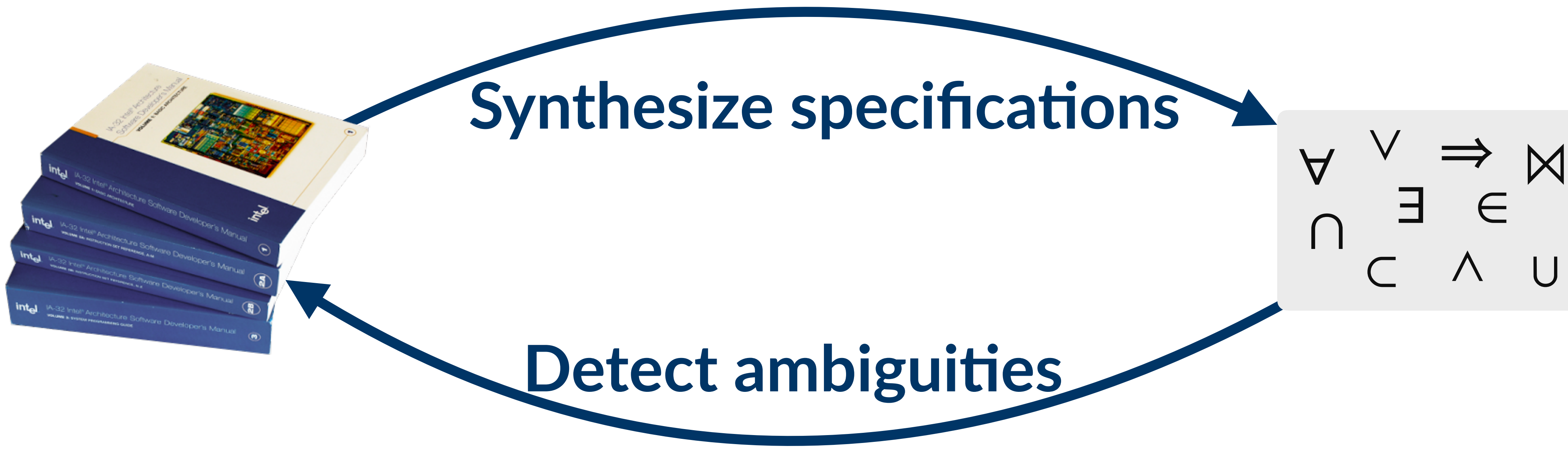


Framework sketch
Synthesize specifications
Detect ambiguities
Litmus tests
∀ ∨ ⇒ ⋈ ∃ ∈ ∩ ⊂ ∧ ∪
Formal specifications

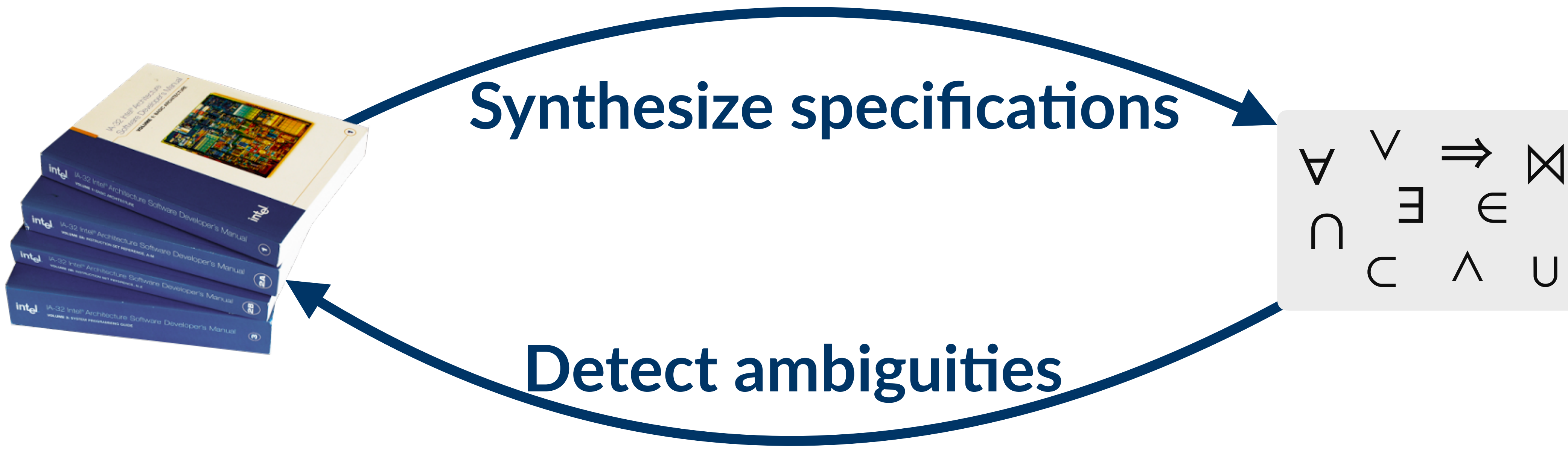
MemSynth


Framework sketch
Synthesize specifications
Detect ambiguities
Litmus tests
$\forall$ $\vee$ $\Rightarrow$ $\bowtie$ $\exists$ $\in$ $\cap$ $\subset$ $\wedge$ $\cup$
Formal specifications

MemSynth



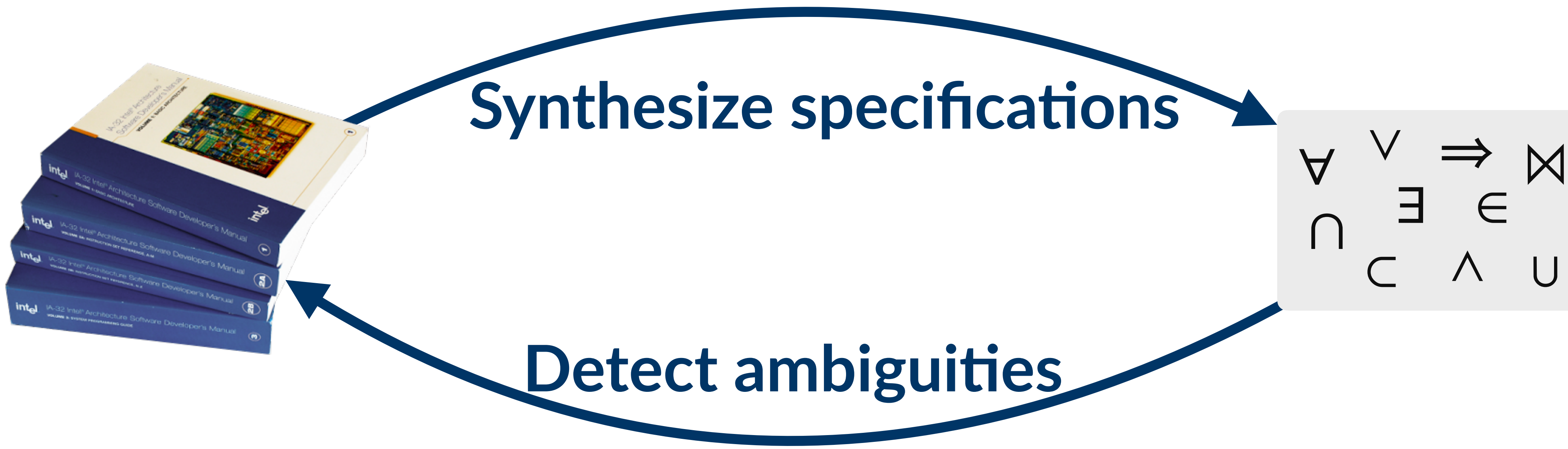
MemSynth



Framework sketches

define a class of memory models

MemSynth



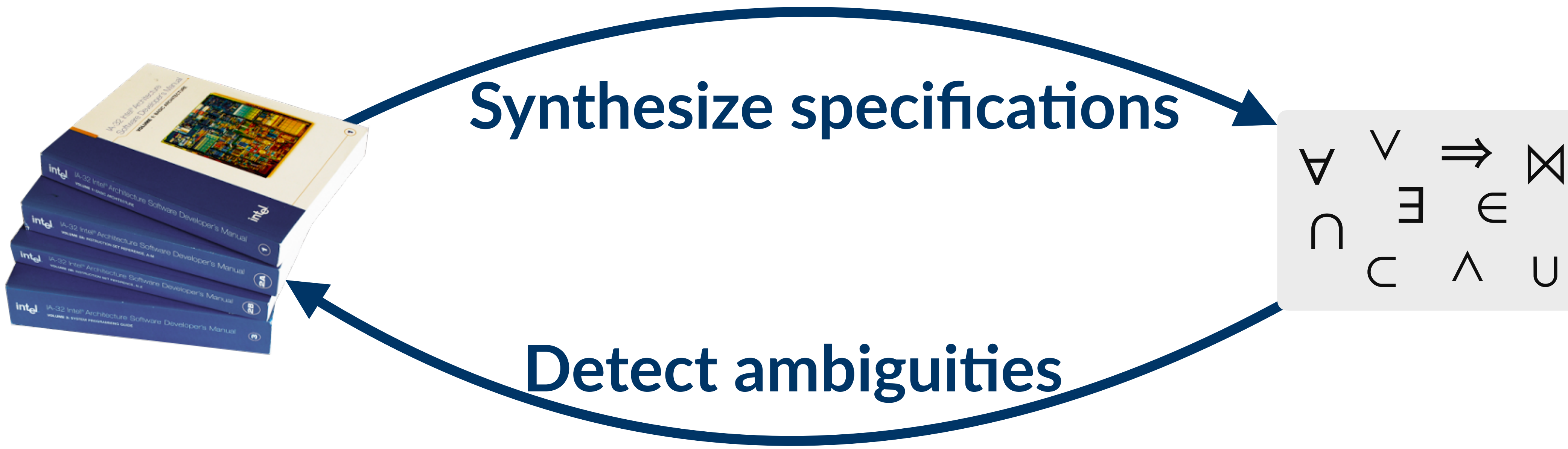
Framework sketches

define a class of memory models

MemSynth engine

verification, equivalence, synthesis, ambiguity

MemSynth



Framework sketches

define a class of memory models

MemSynth engine

verification, equivalence, synthesis, ambiguity

Results

synthesize real-world memory model specs

Memory models and framework sketches

Litmus tests illustrate memory model behavior

Thread 1	Thread 2
1 `X = 1`	3 `Y = 1`
2 `r1 = Y`	4 `r2 = X`

Can $r1 = 0 \wedge r2 = 0$?

Litmus tests illustrate memory model behavior

Thread 1	Thread 2
1 `X = 1`	3 `Y = 1`
2 `r1 = Y`	4 `r2 = X`

Can $r1 = 0 \land r2 = 0$?

Sequential consistency: no

Litmus tests illustrate memory model behavior

Thread 1	Thread 2
1 `X = 1`	3 `Y = 1`
2 `r1 = Y`	4 `r2 = X`

Can $r1 = 0 \wedge r2 = 0$?

Sequential consistency: no

x86: yes!

Litmus tests illustrate memory model behavior

Thread 1	Thread 2
1 `X = 1`	3 `Y = 1`
2 `r1 = Y`	4 `r2 = X`

Can $r1 = 0 \land r2 = 0$?

Sequential consistency: no

x86: yes!

A **memory model** M is a set of constraints that define the possible executions (outcomes) of a program.

Litmus tests illustrate memory model behavior

Thread 1	Thread 2
1 `X = 1`	3 `Y = 1`
2 `r1 = Y`	4 `r2 = X`

Can $r1 = 0 \wedge r2 = 0$?

Sequential consistency: no

x86: yes!

A **memory model** M is a set of constraints that define the possible executions (outcomes) of a program.

Memory model M **allows** litmus test T if there exists an execution that satisfies M's constraints.

Litmus tests illustrate memory model behavior

Memory model M allows test T: $\exists$ E. M(T,E)

Thread 1	Thread 2
1 `X = 1`	3 `Y = 1`
2 `r1 = Y`	4 `r2 = X`

Can $r1 = 0 \wedge r2 = 0$?

Sequential consistency: no

x86: yes!

A **memory model** M is a set of constraints that define the possible executions (outcomes) of a program.

Memory models, formally

Memory model M allows test T: ∃ E. M(T,E)

Common formalizations based on **relational logic**

Example for **sequential consistency**:

```
no ^(ws + fr + po + rf + fences) & iden
```

[Alglave et al, CAV'10]

Memory models, formally

Memory model
M allows test T:
∃ E. M(T,E)

Common formalizations based on **relational logic**

Example for **sequential consistency**:

```
no ^(ws + fr + po + rf + fences) & iden
```

Binary relations over program instructions

[Alglave et al, CAV'10]

Memory models, formally

Memory model M allows test T: ∃ E. M(T,E)

Common formalizations based on **relational logic**

Example for **sequential consistency**:

happens-before order

```
no ^(ws + fr + po + rf + fences) & iden
```

Binary relations over program instructions

[Alglave et al, CAV'10]

Memory models, formally

Memory model M allows test T: ∃ E. M(T,E)

Common formalizations based on **relational logic**

Example for **sequential consistency**:

$$\textbf{no}\ \underbrace{\texttt{\^{}(ws + fr + po + rf + fences)}}_{\text{happens-before order}}\ \underbrace{\texttt{\& }\textbf{iden}}_{\text{is acyclic}}$$

Binary relations over program instructions

[Alglave et al, CAV'10]

Memory models, formally

Memory model M allows test T: ∃ E. M(T,E)

Common formalizations based on **relational logic**

Example for **sequential consistency**:

$$\texttt{no}\ \overbrace{\texttt{\^{}(ws + fr + po + rf + fences)}}^{\text{happens-before order}}\ \overbrace{\texttt{\& iden}}^{\text{is acyclic}}$$

From program syntax

Binary relations over program instructions

[Alglave et al, CAV'10]

Memory models, formally

Memory model M allows test T: ∃ E. M(T,E)

Common formalizations based on **relational logic**

Example for **sequential consistency**:

$$\texttt{no}\ \overbrace{\texttt{\^{}(ws + fr + po + rf + fences)}}^{\text{happens-before order}}\ \overbrace{\texttt{\& iden}}^{\text{is acyclic}}$$

From program syntax

Binary relations over program instructions

Thread 1	Thread 2
1 `X = 1`	3 `Y = 1`
2 `r1 = Y`	4 `r2 = X`

Can r1 = 0 ∧ r2 = 0?

[Alglave et al, CAV'10]

Memory models, formally

Memory model M allows test T: ∃ E. M(T,E)

Common formalizations based on **relational logic**

Example for **sequential consistency**:

$$\texttt{no}\ \underbrace{\texttt{\^{}(ws + fr + po + rf + fences)}}_{\text{happens-before order}}\ \underbrace{\texttt{\& iden}}_{\text{is acyclic}}$$

From program syntax

Binary relations over program instructions

Thread 1	Thread 2
1 X = 1	3 Y = 1
2 r1 = Y	4 r2 = X

Can r1 = 0 ∧ r2 = 0?

Program order:

po = {(1,2), (3,4)}

[Alglave et al, CAV'10]

Memory models, formally

Memory model M allows test T: ∃ E. M(T,E)

Common formalizations based on **relational logic**

Example for **sequential consistency**:

Part of execution; implicitly existentially quantified

$$\texttt{no}\ \overbrace{\texttt{\^{}(ws + fr + po + rf + fences)}}^{\text{happens-before order}}\ \overbrace{\texttt{\& iden}}^{\text{is acyclic}}$$

From program syntax

Binary relations over program instructions

Thread 1	**Thread 2**
1 `X = 1`	3 `Y = 1`
2 `r1 = Y`	4 `r2 = X`

Can r1 = 0 ∧ r2 = 0?

Program order:

`po` = {(1, 2), (3, 4)}

[Alglave et al, CAV'10]

Framework sketches

A **framework sketch** defines the search space for synthesizing a memory model *M* by including *holes* in constraints

```
no ^(ws + fr + po + rf + fences) & iden
```

Framework sketches

A **framework sketch** defines the search space for synthesizing a memory model *M* by including *holes* in constraints

Expression holes
for a synthesizer to complete

```
no ^(ws + fr + ?? + ?? +    ??   ) & iden
```

Framework sketches

A **framework sketch** defines the search space for synthesizing a memory model *M* by including *holes* in constraints

Expression holes for a synthesizer to complete

```
no ^(ws + fr + ?? + ?? +    ??   ) & iden
```

Framework sketches are the **key design tool** for synthesizing memory model specifications — they define the "interesting" candidate models

Memory model frameworks

```
no ^(ws + fr + ?? + ?? +    ??   ) & iden
```

[Alglave et al, CAV'10]

Memory model frameworks

```
no ^(ws + fr + ppo + grf + fences) & iden
```

Preserved program order (same-thread reorderings)

Global reads from (inter-thread order)

Fence cumulativity (for Power, ARM, etc)

[Alglave et al, CAV'10]

Memory model frameworks

`no ^(ws + fr + ppo + grf + fences) & iden`

	ppo	grf	fences
	Preserved program order (same-thread reorderings)	**Global reads from** (inter-thread order)	**Fence cumulativity** (for Power, ARM, etc)
Sequential consistency	`po`	`rf`	`∅`

[Alglave et al, CAV'10]

Memory model frameworks

`no ^(ws + fr + ppo + grf + fences) & iden`

	Preserved program order (same-thread reorderings)	**Global reads from** (inter-thread order)	**Fence cumulativity** (for Power, ARM, etc)
Sequential consistency	`po`	`rf`	`∅`
Total store order (x86)	`po - (Wr→Rd)`	`rf & SameThd`	`∅`

[Alglave et al, CAV'10]

Memory model frameworks are common

Global time relational model
[Alglave et al, CAV'10]

Axiomatic "must-not-reorder" functions
[Mador-Haim et al, DAC'11]

Exexcutable distributed consistency models
[Yang et al, IPDPS'04]

…

Ocelot: relational logic with holes

A **relational logic DSL** with synthesis support

Built on the Rosette solver-aided language [Torlak & Bodik, PLDI'14]

Expression holes
for a synthesizer
to complete

```
no ^(ws + fr + ?? + ?? + ?? ) & iden
```

Available as a Racket package: `raco pkg install ocelot`

Ocelot: relational logic with holes

A **relational logic DSL** with synthesis support

Built on the Rosette solver-aided language [Torlak & Bodik, PLDI'14]

Expression holes for a synthesizer to complete

```
no ^(ws + fr + ?? + ?? + ?? ) & iden
```

Completions are expressions in relational logic with chosen operators, terminals, and depth.

Available as a Racket package: `raco pkg install ocelot`

Ocelot: relational logic with holes

A **relational logic DSL** with synthesis support

Built on the Rosette solver-aided language [Torlak & Bodik, PLDI'14]

Expression holes for a synthesizer to complete

```
no ^(ws + fr + ?? + ?? + ?? ) & iden
```

Completions are expressions in relational logic with chosen operators, terminals, and depth.

operators = `{+, &}`

terminals = `{po, ws}`

depth = 1

Available as a Racket package: `raco pkg install ocelot`

Ocelot: relational logic with holes

A **relational logic DSL** with synthesis support

Built on the Rosette solver-aided language [Torlak & Bodik, PLDI'14]

Expression holes for a synthesizer to complete

```
no ^(ws + fr + ?? + ?? + ?? ) & iden
```

Completions are expressions in relational logic with chosen operators, terminals, and depth.

operators = `{+, &}`

terminals = `{po, ws}`

depth = 1

```
po
ws
po + ws
po & ws
```

Available as a Racket package: `raco pkg install ocelot`

Queries

- Verification
- Equivalence
- Synthesis
- Ambiguity

Verification and equivalence

Memory model M allows test T: ∃ E. M(T,E)

Common queries for automated memory model reasoning tools

Herd [Alglave et al, CAV'10]; MemAlloy [Wickerson et al, POPL'17]; etc.

Verification and equivalence

Memory model M allows test T: $\exists$ E. M(T,E)

Common queries for automated memory model reasoning tools

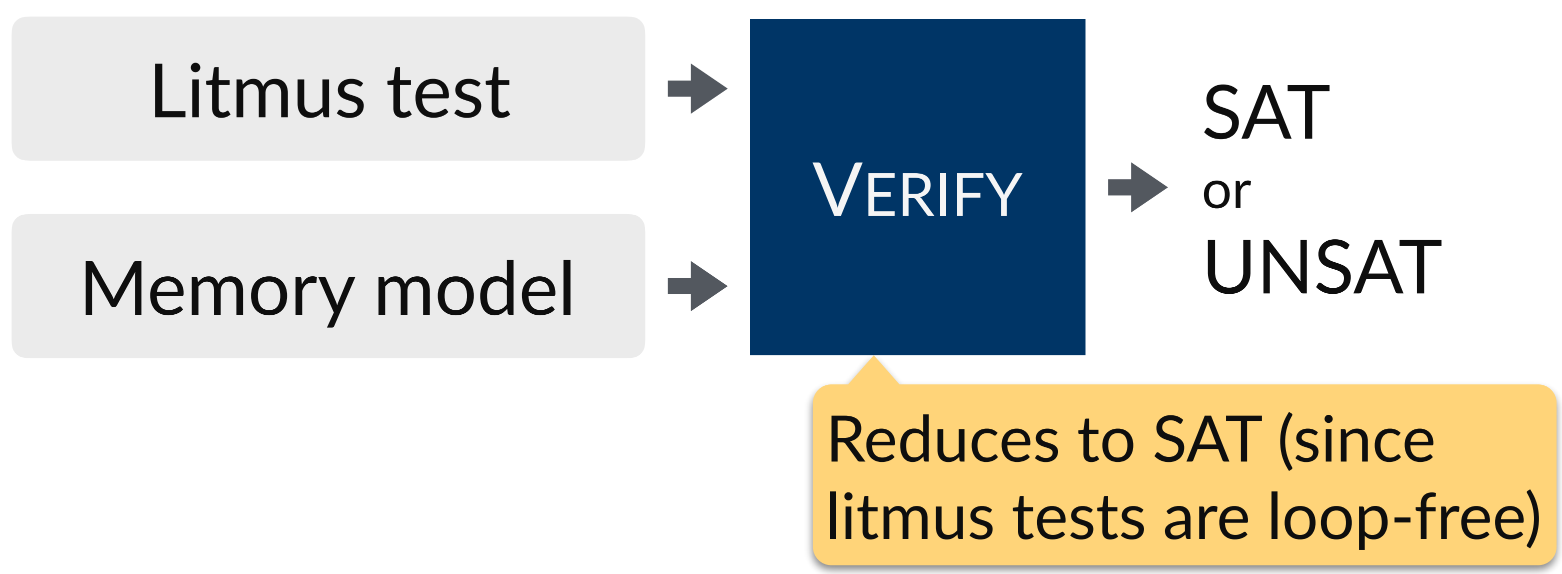
Herd [Alglave et al, CAV'10]; MemAlloy [Wickerson et al, POPL'17]; etc.

Verification and equivalence

Memory model M allows test T: ∃ E. M(T,E)

Common queries for automated memory model reasoning tools

Herd [Alglave et al, CAV'10]; MemAlloy [Wickerson et al, POPL'17]; etc.



Verification and equivalence

Memory model M allows test T: $\exists$ E. M(T,E)

Common queries for automated memory model reasoning tools

Herd [Alglave et al, CAV'10]; MemAlloy [Wickerson et al, POPL'17]; etc.

Litmus test → VERIFY

Memory model → VERIFY

VERIFY → SAT or UNSAT

Reduces to SAT (since litmus tests are loop-free)

Memory model M_A → EQUIV

Memory model M_B → EQUIV

EQUIV → Litmus test or UNSAT

Verification and equivalence

Memory model M allows test T: ∃ E. M(T,E)

Common queries for automated memory model reasoning tools

Herd [Alglave et al, CAV'10]; MemAlloy [Wickerson et al, POPL'17]; etc.

Litmus test → VERIFY
Memory model → VERIFY
VERIFY → SAT or UNSAT

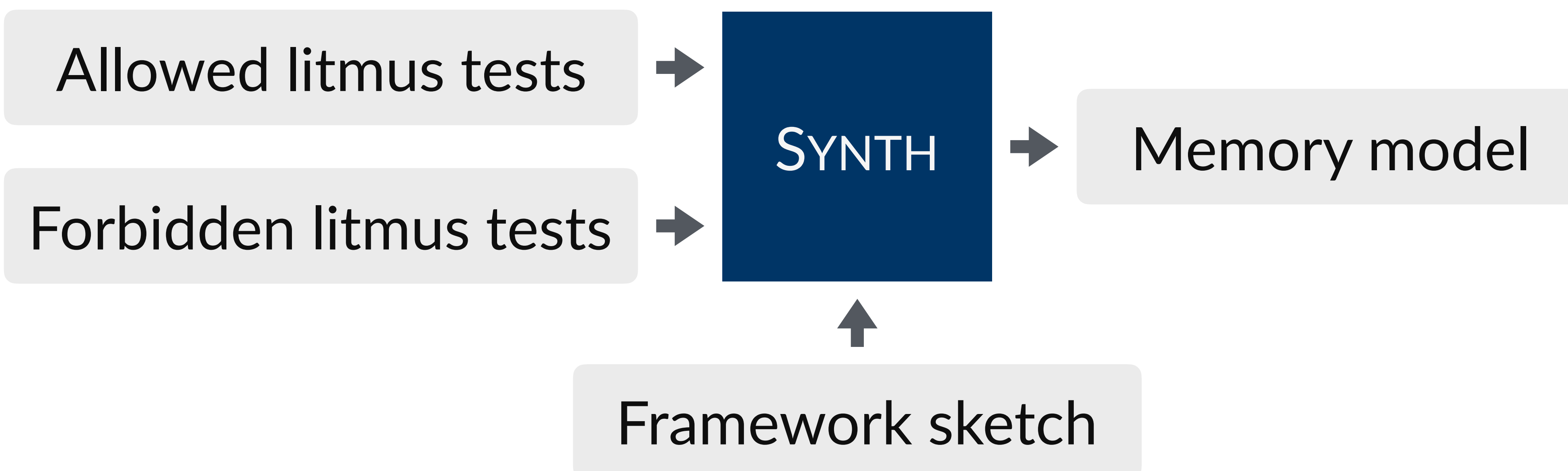
Reduces to SAT (since litmus tests are loop-free)

Memory model M_A → EQUIV
Memory model M_B → EQUIV
EQUIV → Litmus test or UNSAT

UNSAT = **bounded** equivalence ("equivalent up to tests of size *k*")

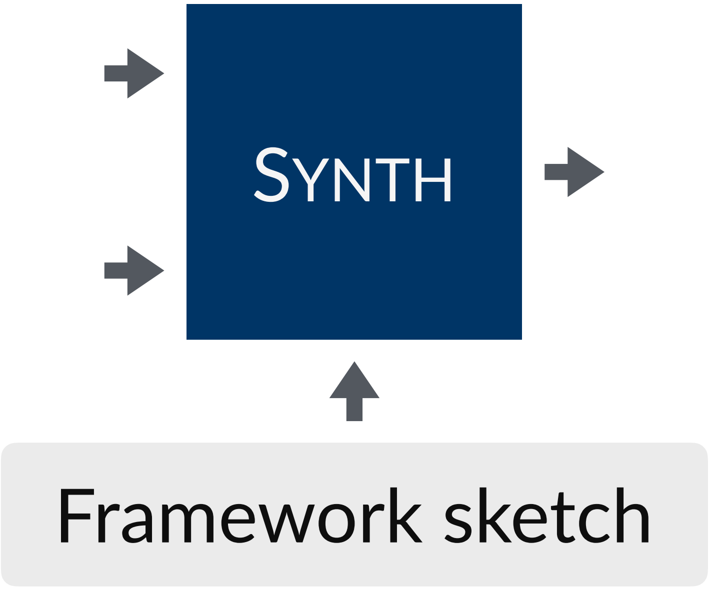
Synthesis

Find a memory model consistent with a set of litmus tests



Synthesis

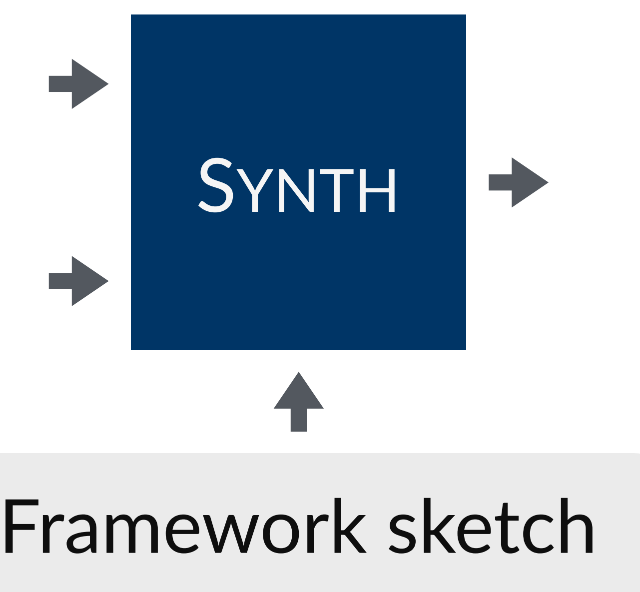
Find a memory model consistent with a set of litmus tests



Synthesis

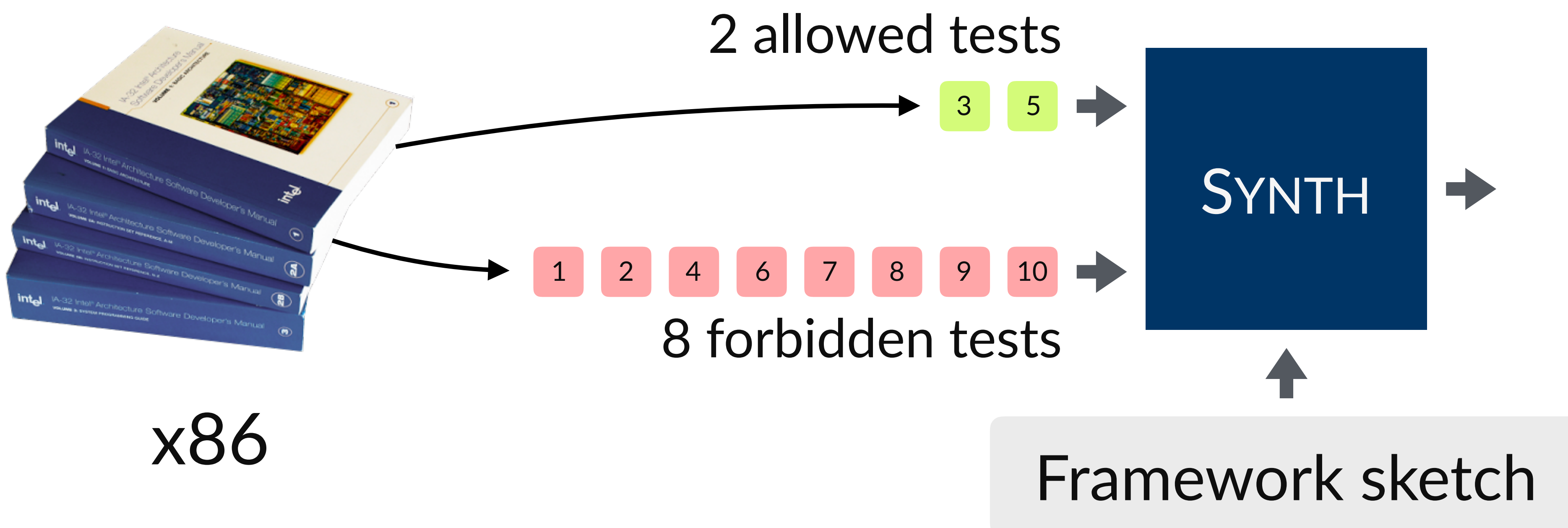
Find a memory model consistent with a set of litmus tests

x86



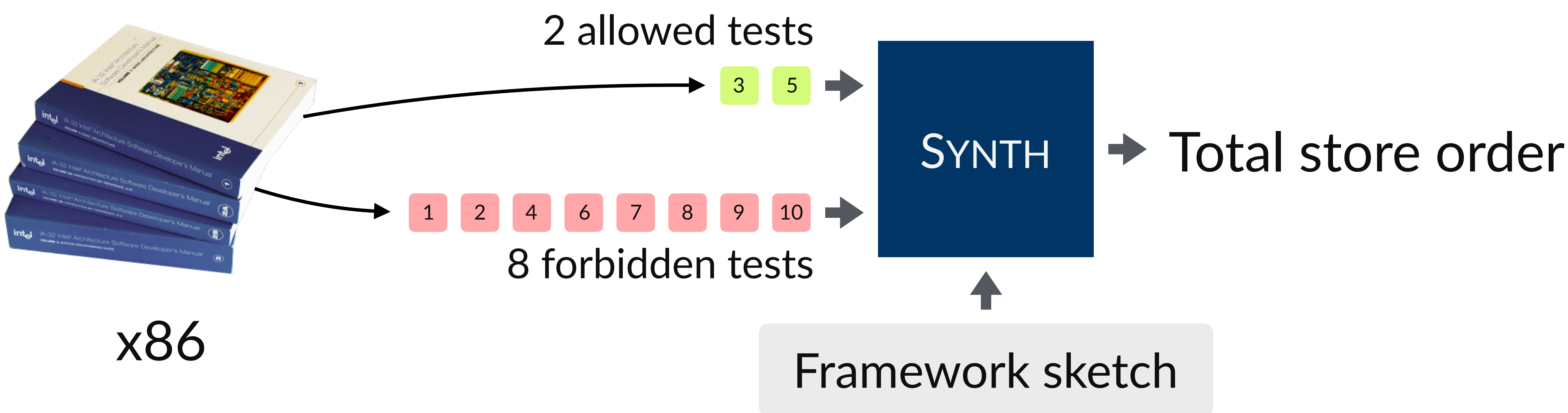
Synthesis

Find a memory model consistent with a set of litmus tests



Synthesis

Find a memory model consistent with a set of litmus tests



Synthesis

Find a memory model consistent with a set of litmus tests

Memory model M allows test T: $\exists E.\ M(T,E)$

Allowed litmus tests T^+

Forbidden litmus tests T^-

Memory model

M

Framework sketch

Synthesis

Find a memory model consistent with a set of litmus tests

Memory model M allows test T: $\exists$ E. M(T,E)

Allowed litmus tests T^+ →

Forbidden litmus tests T^- →

$$\bigwedge_{T \in T^+} \exists E.\, M(T,E)$$

→ Memory model

M

Framework sketch

Synthesis

Find a memory model consistent with a set of litmus tests

Memory model M allows test T: $\exists$ E. M(T,E)

Allowed litmus tests $T^{+} \rightarrow \bigwedge_{T \in T^{+}} \exists E.\ M(T,E)$

Forbidden litmus tests $T^{-} \rightarrow \bigwedge_{T \in T^{-}} \forall E.\ \neg M(T,E)$

$\rightarrow$ Memory model

M

Framework sketch

Synthesis

Find a memory model consistent with a set of litmus tests

Memory model M allows test T: $\exists$ E. M(T,E)

Allowed litmus tests $T^+ \rightarrow \bigwedge_{T \in T^+} \exists E.\ M(T,E)$

Forbidden litmus tests $T^- \rightarrow \bigwedge_{T \in T^-} \forall E.\ \neg M(T,E)$

$\rightarrow$ Memory model

Solved **incrementally**, like counterexample-guided inductive synthesis (CEGIS)

M

Framework sketch

Ambiguity

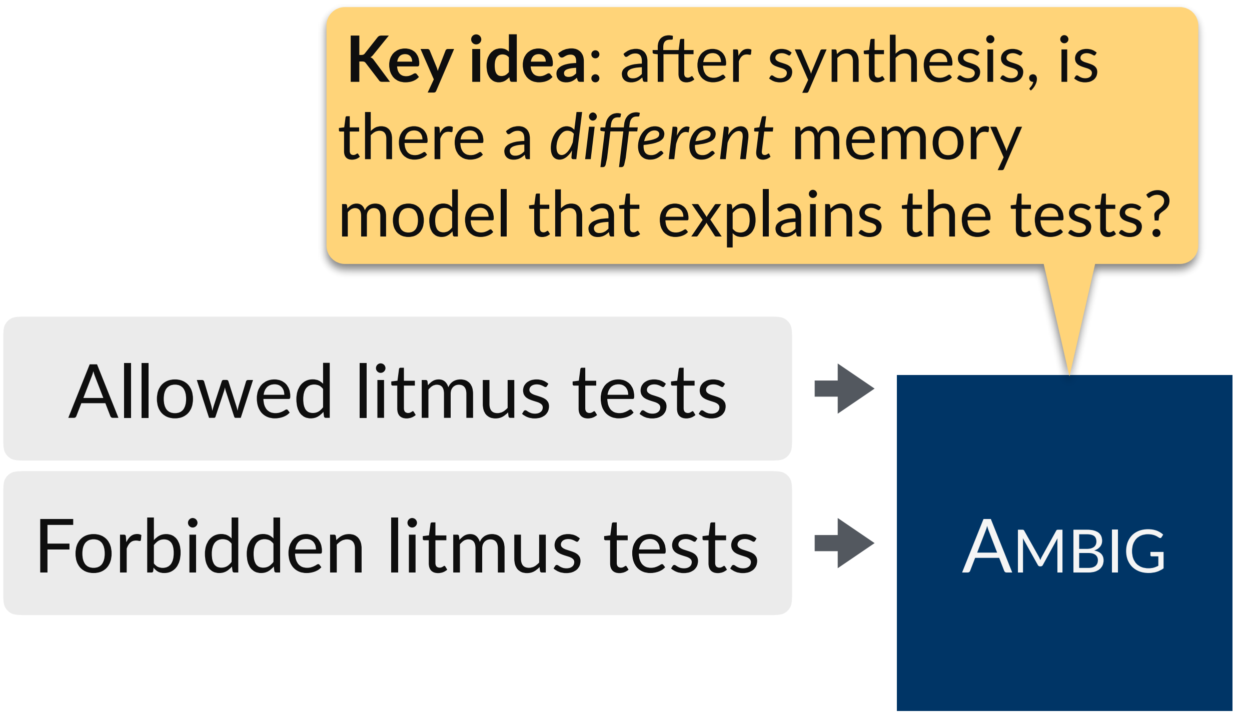
Find a distinguishing litmus test that exposes an ambiguity in a model

Key idea: after synthesis, is there a *different* memory model that explains the tests?

AMBIG

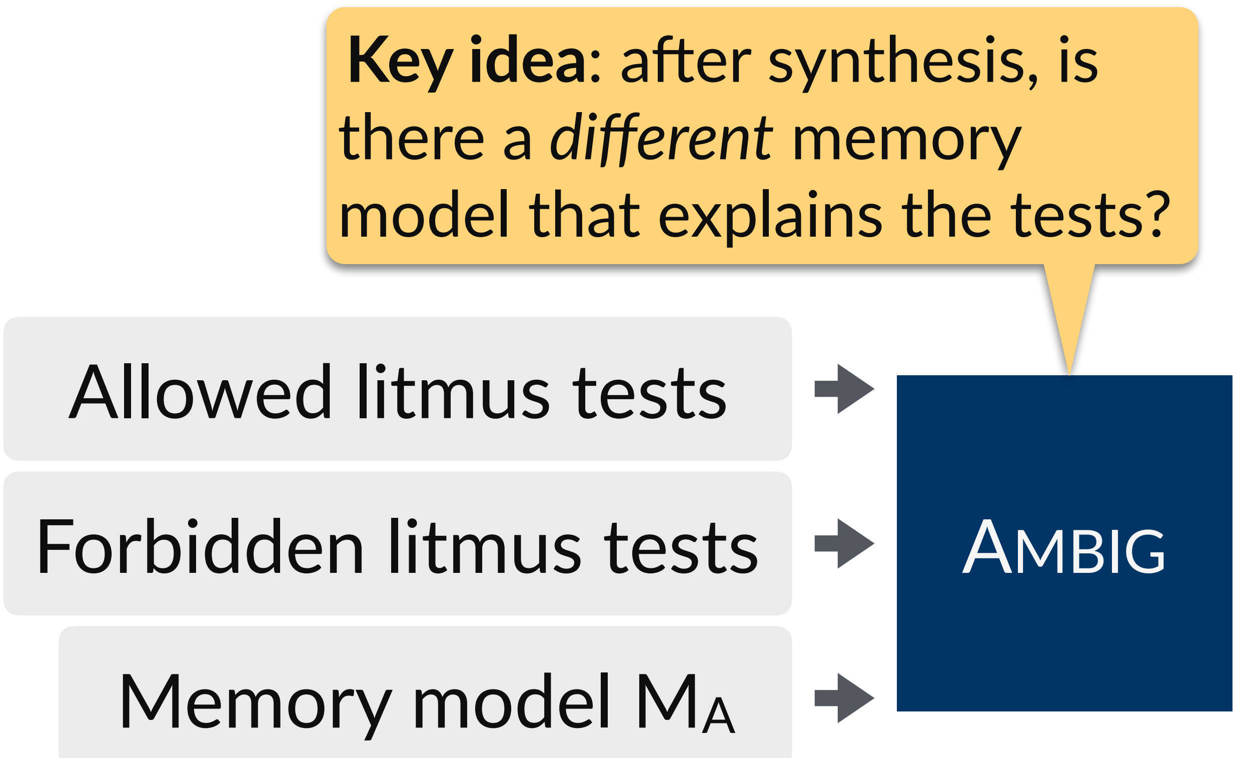
Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model



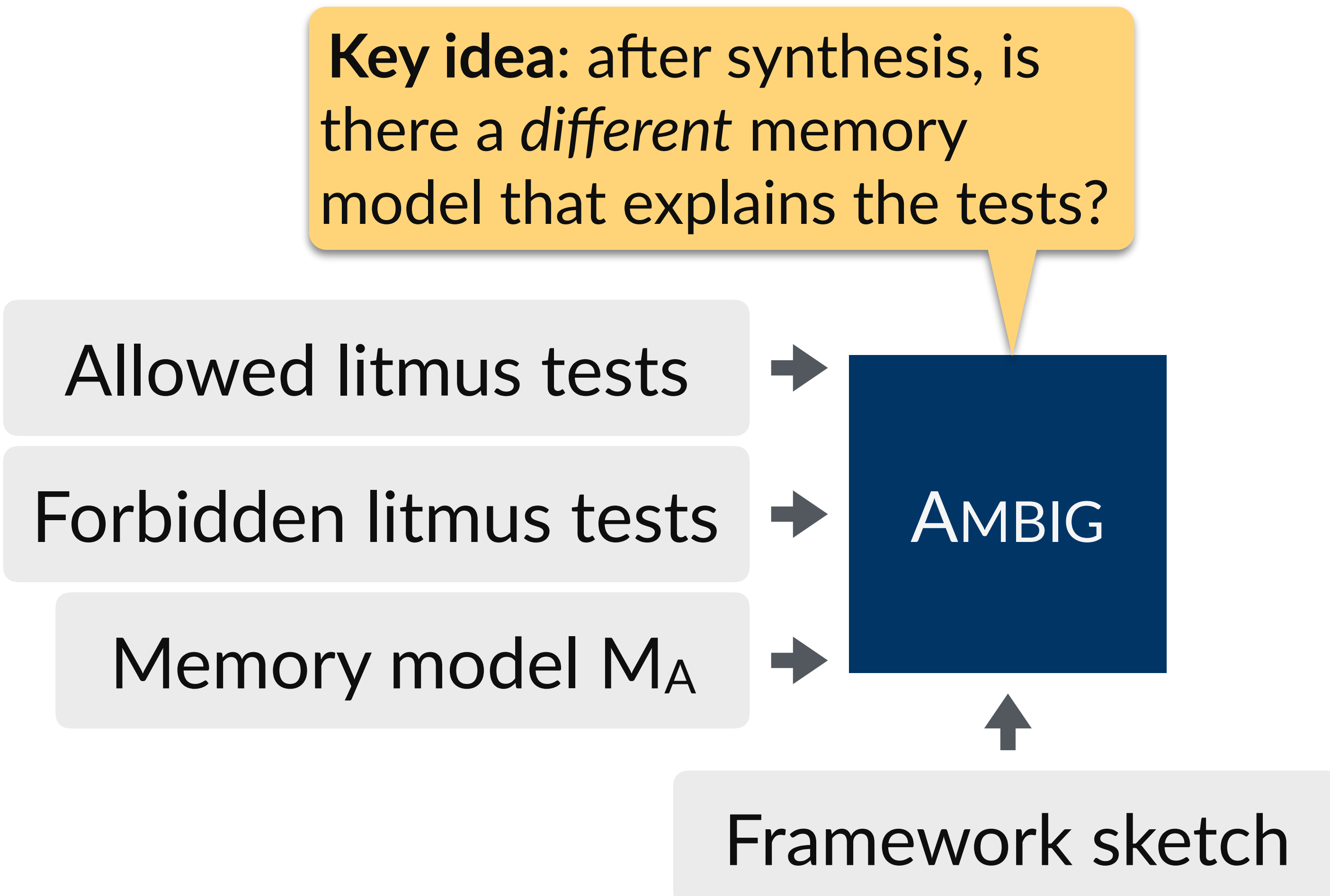
Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model



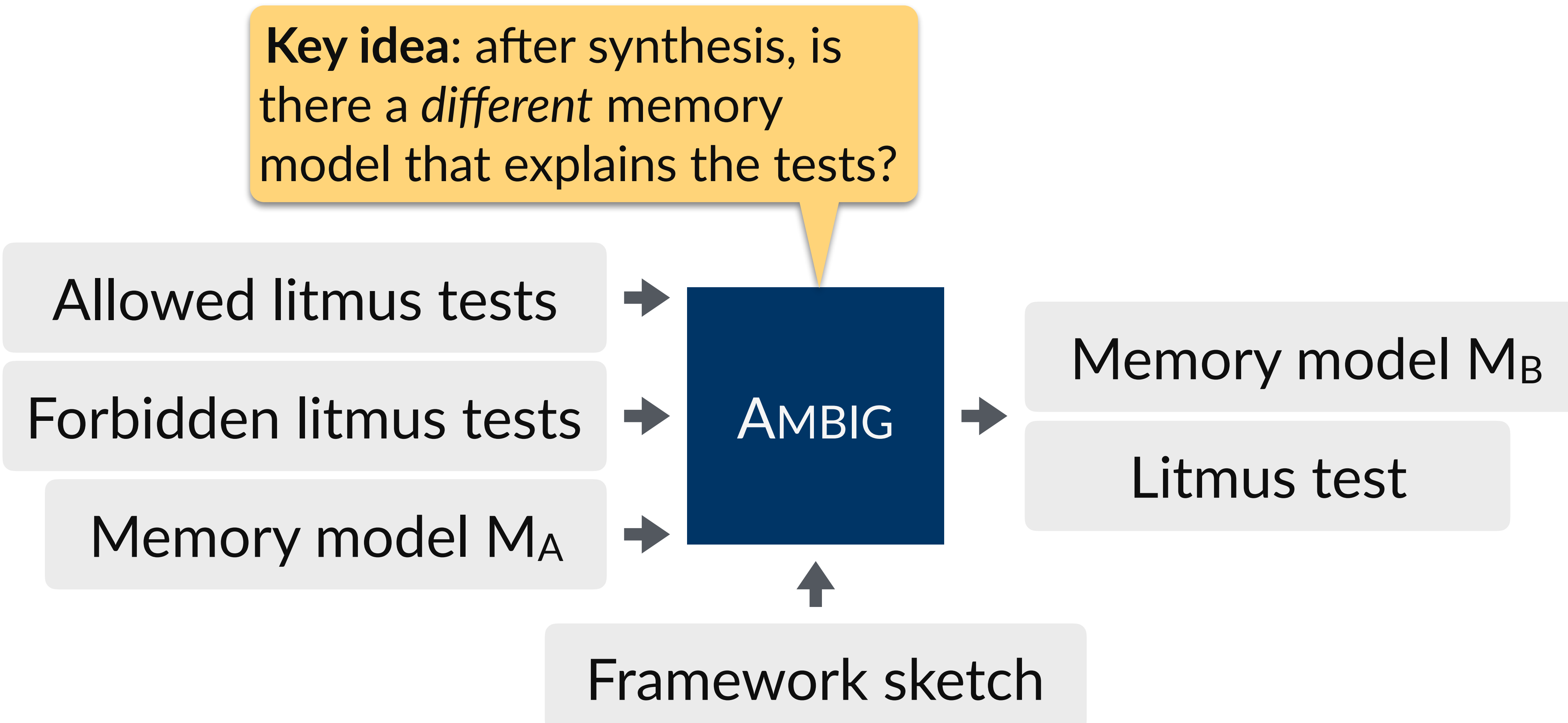
Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model



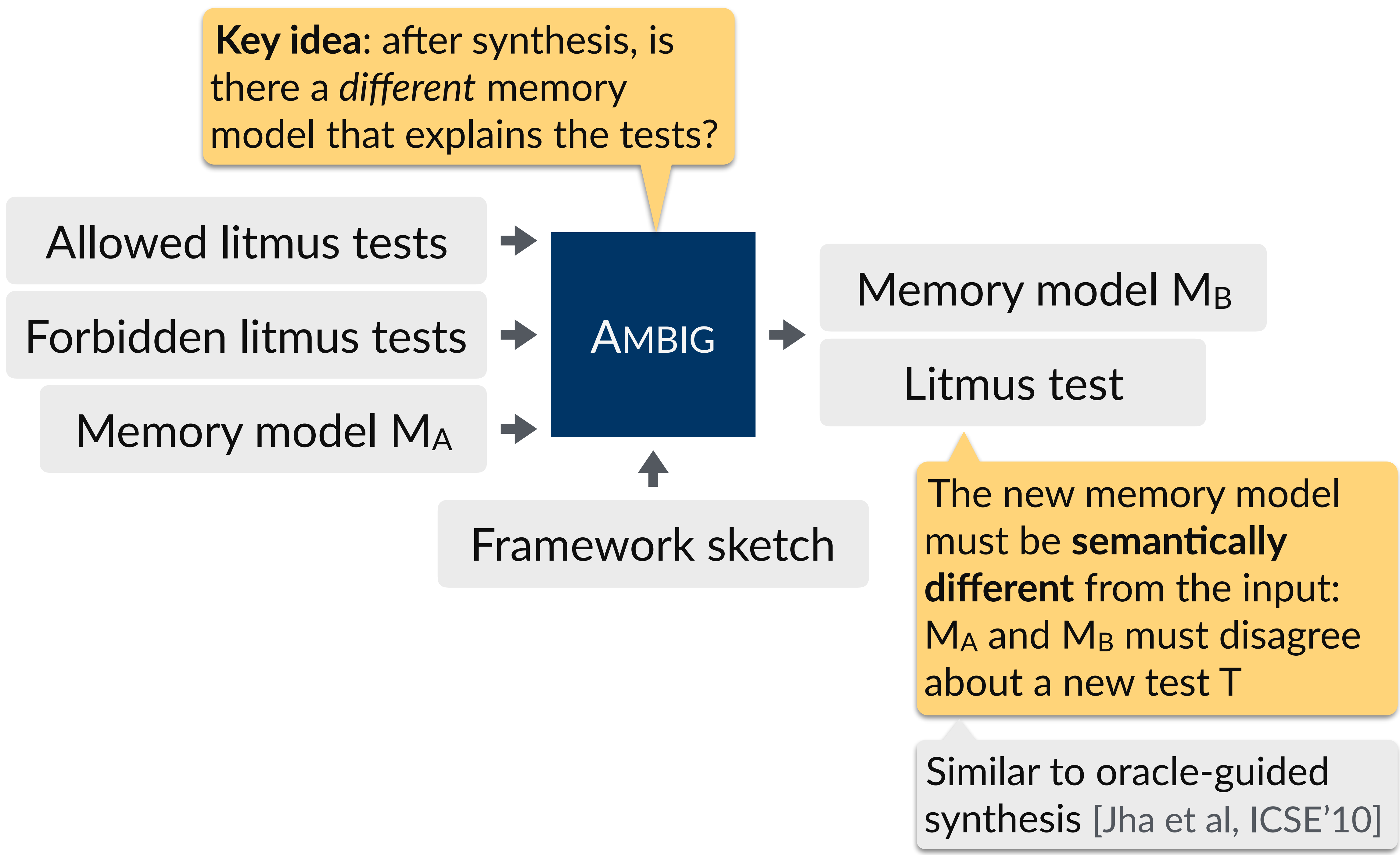
Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model



Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model



Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model

Thread 1	Thread 2
1 `X = 1`	3 `Y = 1`
2 `r1 = Y`	4 `r2 = X`

Can $r1 = 0 \wedge r2 = 0$?

Total store order (x86)

Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model

Is there another **semantically different** memory model that also allows this test?

Thread 1	Thread 2
1 `X = 1`	3 `Y = 1`
2 `r1 = Y`	4 `r2 = X`

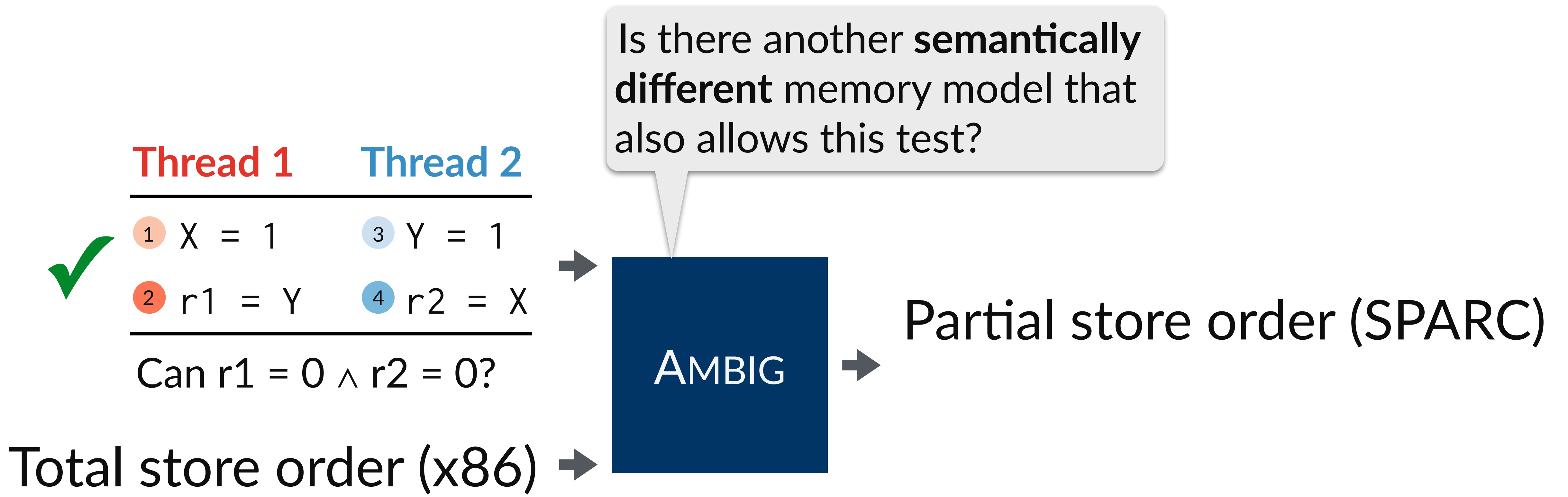
Can r1 = 0 ∧ r2 = 0?

Total store order (x86)

AMBIG

Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model



Ambiguity

Find a distinguishing litmus test that exposes an ambiguity in a model

Is there another **semantically different** memory model that also allows this test?

✓

Thread 1	Thread 2
1 `X = 1`	3 `Y = 1`
2 `r1 = Y`	4 `r2 = X`

Can r1 = 0 $\wedge$ r2 = 0?

Total store order (x86)

AMBIG

Partial store order (SPARC)

Thread 1	Thread 2
1 `X = 1`	3 `r1 = Y`
2 `Y = 1`	4 `r2 = X`

Can r1 = 1 $\wedge$ r2 = 0?

✓ PSO

✗ TSO

The Synthesis-Ambiguity Cycle

3 5

1 2 4

Litmus tests

The Synthesis-Ambiguity Cycle

Documentation

Random/systematic generation

Architects

3 5

1 2 4

Litmus tests

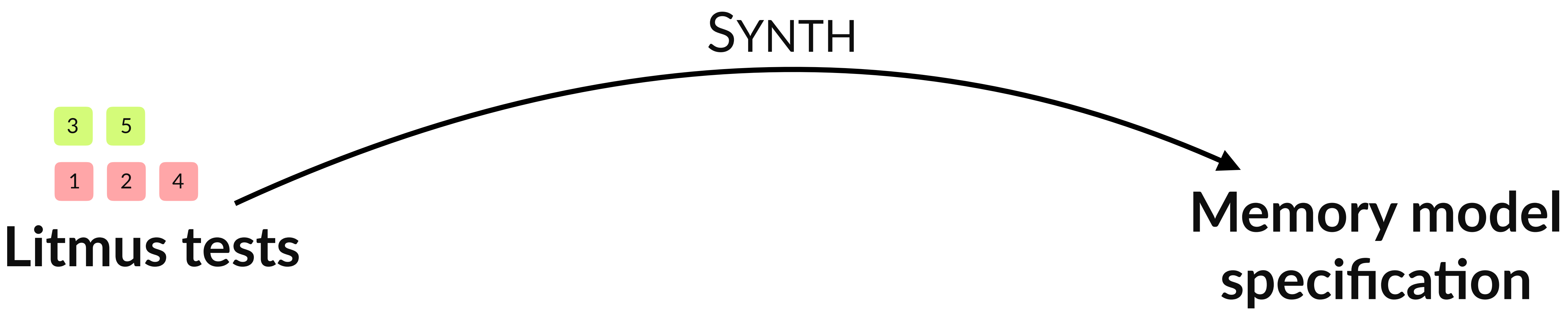
The Synthesis-Ambiguity Cycle

3 5

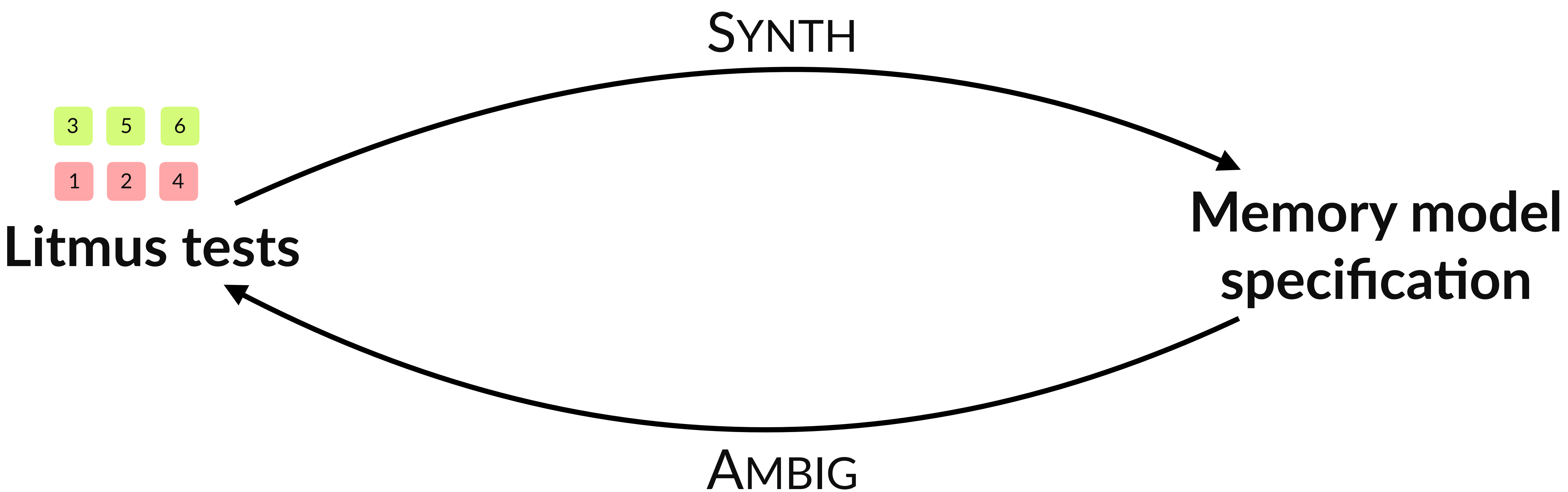
1 2 4

Litmus tests

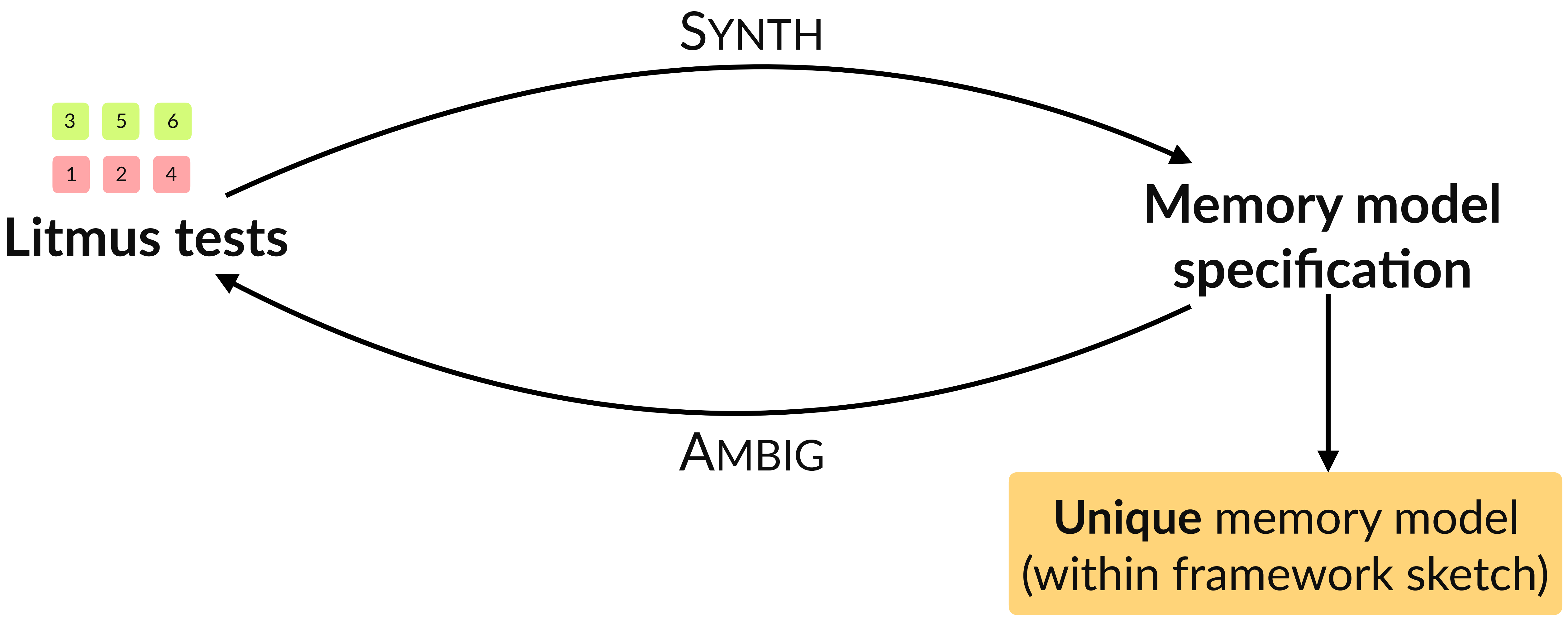
The Synthesis-Ambiguity Cycle



The Synthesis-Ambiguity Cycle



The Synthesis-Ambiguity Cycle



Results

Synthesizing existing memory models

PowerPC

x86

Synthesizing existing memory models

PowerPC

768 tests
[Alglave et al, CAV’10]

x86

10 tests

Synthesizing existing memory models

		Synthesis
PowerPC	768 tests [Alglave et al, CAV'10]	✓ 12 seconds Search space: 2^{1406}
x86	10 tests 	✓ 2 seconds Search space: 2^{624}

Synthesizing existing memory models

		Synthesis
PowerPC	768 tests [Alglave et al, CAV'10]	✓ 12 seconds Search space: 2^{1406}
x86	10 tests 	✓ 2 seconds Search space: 2^{624}

Not equivalent to published model!

Synthesizing existing memory models

Synthesis

PowerPC	768 tests [Alglave et al, CAV'10]	✓ 12 seconds Search space: 2^{1406} Not equivalent to published model!
x86	10 tests 	✓ 2 seconds Search space: 2^{624} Not equivalent to TSO!

Synthesizing existing memory models

		Synthesis	Ambiguity
PowerPC	768 tests [Alglave et al, CAV'10]	✓ 12 seconds Search space: 2^{1406}	9 new tests `sync`, `lwsync`, etc.
		Not equivalent to published model!	
x86	10 tests 	✓ 2 seconds Search space: 2^{624}	4 new tests `mfence`, `xchg`
		Not equivalent to TSO!	

Other results

Implemented another **framework sketch** [Mador-Haim et al, DAC'11]

- Found typo in paper; couldn't fix by hand, but synthesized repair

Other results

Implemented another **framework sketch** [Mador-Haim et al, DAC'11]

- Found typo in paper; couldn't fix by hand, but synthesized repair

Order of magnitude faster than the Alloy **general-purpose relational solver** for verification and equivalence

- Ocelot offers finer-grained control over relational constraints

Other results

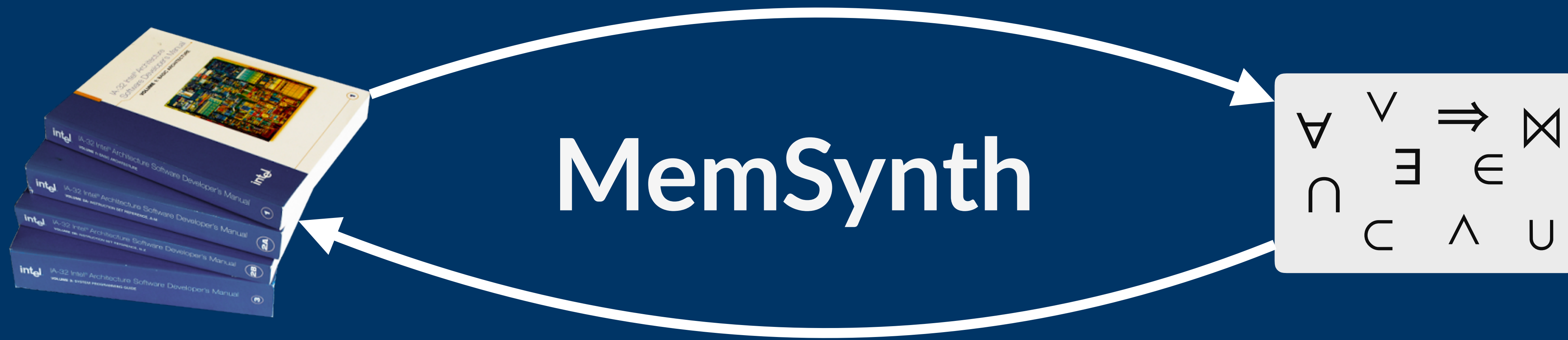
Implemented another **framework sketch** [Mador-Haim et al, DAC'11]

Found typo in paper; couldn't fix by hand, but synthesized repair

Order of magnitude faster than the Alloy **general-purpose relational solver** for verification and equivalence

Ocelot offers finer-grained control over relational constraints

Comparable performance to existing **custom memory model tool** for verification (Herd [Alglave et al, CAV'10])



Framework sketches
define a class of memory models

MemSynth engine
verification, equivalence, synthesis, ambiguity

Results
synthesize real-world memory model specs

memsynth.uwplse.org