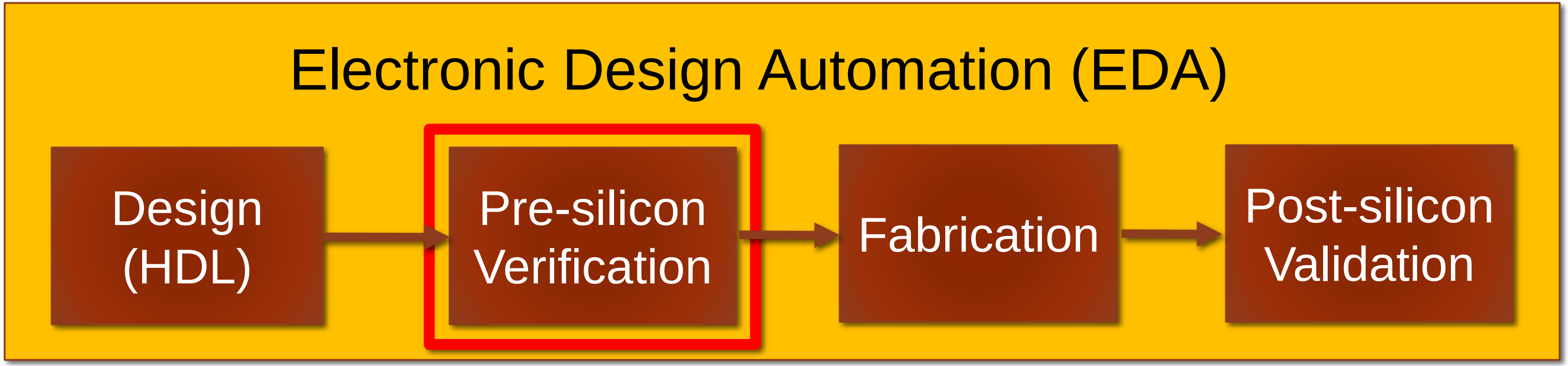
A Theoretical Framework for Symbolic Quick Error Detection

FLORIAN LONSING
SUBHASISH MITRA
CLARK BARRETT

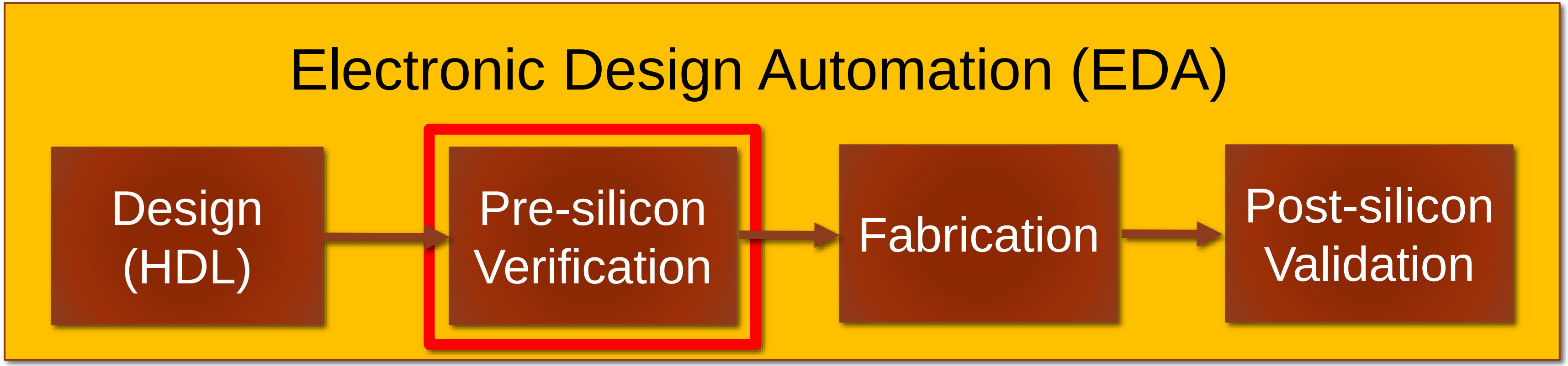
Paper published at Formal Methods in Computer-Aided Design (FMCAD) 2020
Preprint: https://arxiv.org/abs/2006.05449

Context: Pre-Silicon Verification



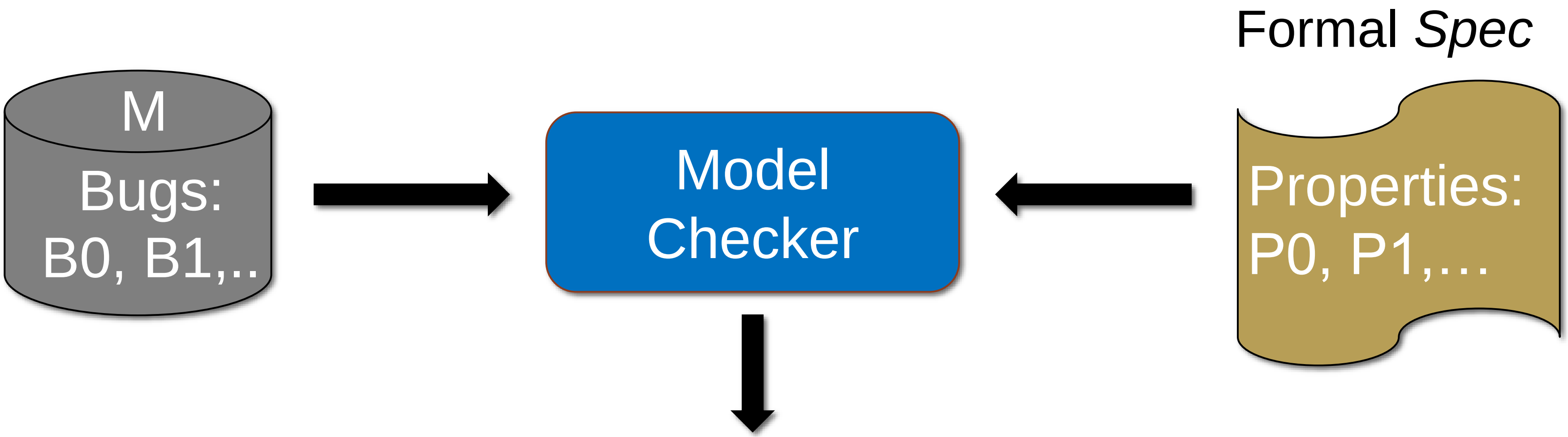
- Our focus: processor designs.
- Formally verify model of a design (e.g. Verilog).
- Model checking vs. non-formal simulation or testing.

Context: Pre-Silicon Verification



- Our focus: processor designs.
- Formally verify model of a design (e.g. Verilog).
- Model checking vs. non-formal simulation or testing.

Soundness of Bug-Finding

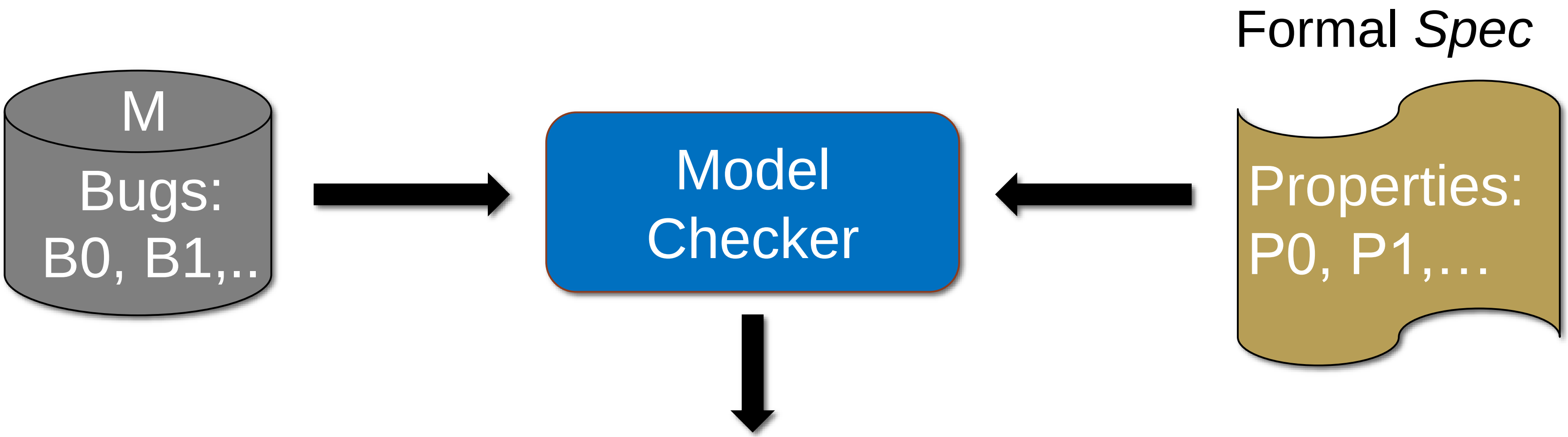


Does P ∈ *Spec* hold in M?

- If P ∈ *Spec* fails then B ∈ M.
- Property P covers bug B.

Soundness ≈
no spurious cex

Completeness of Bug-Finding

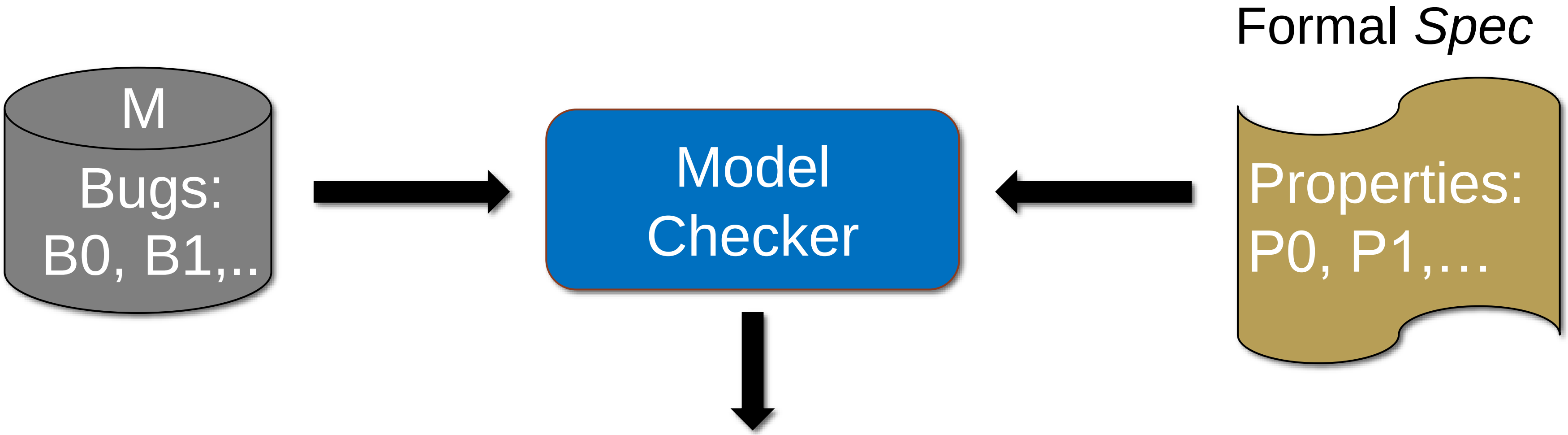


Does P ∈ *Spec* hold in M?

- If B ∈ M then P ∈ *Spec* fails.
- Property P covers bug B.

Completeness ≈
Spec covers all bugs

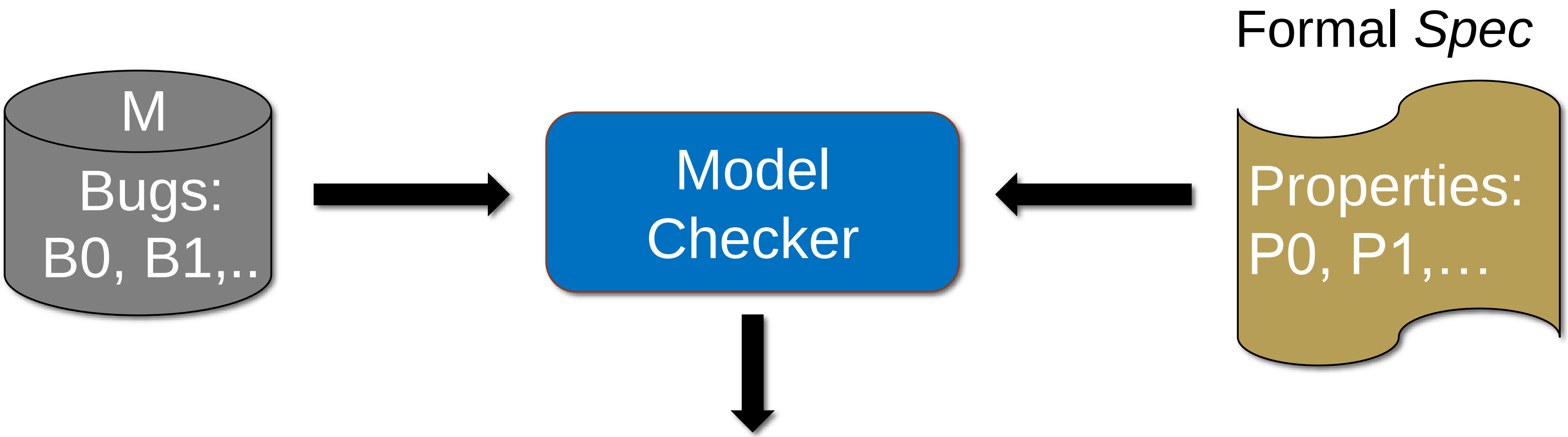
Completeness of Bug-Finding



Does P ∈ *Spec* hold in M?

- "Have I written enough properties?" [Katz et al. CHARME'99].
- Challenge: making *Spec* complete.

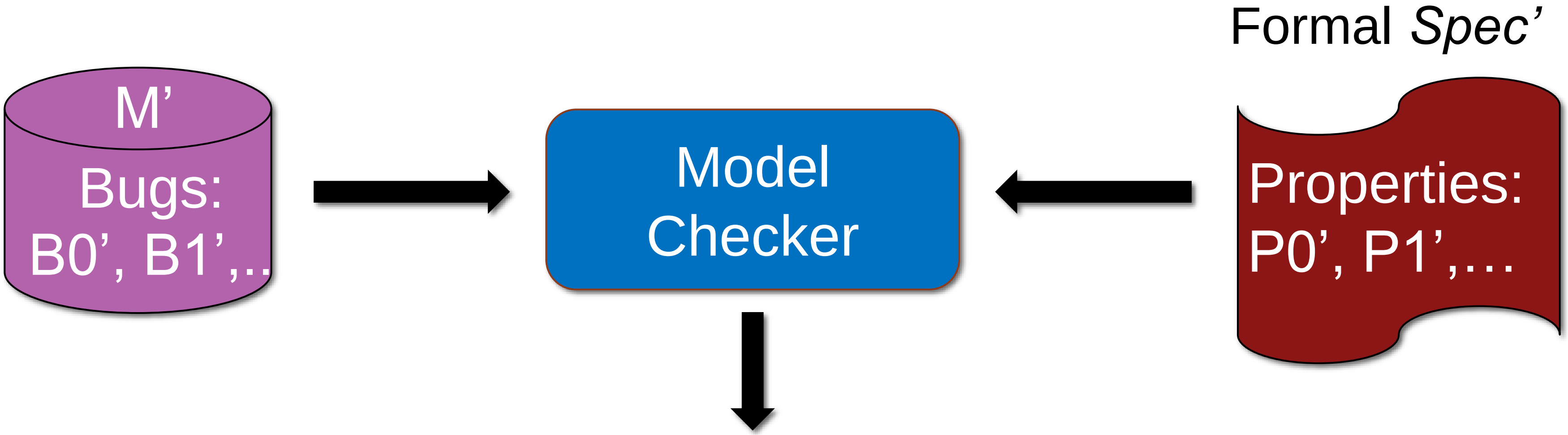
Completeness of Bug-Finding



Does P ∈ *Spec* hold in M?

- *Spec*: manual writing of implementation-specific properties.

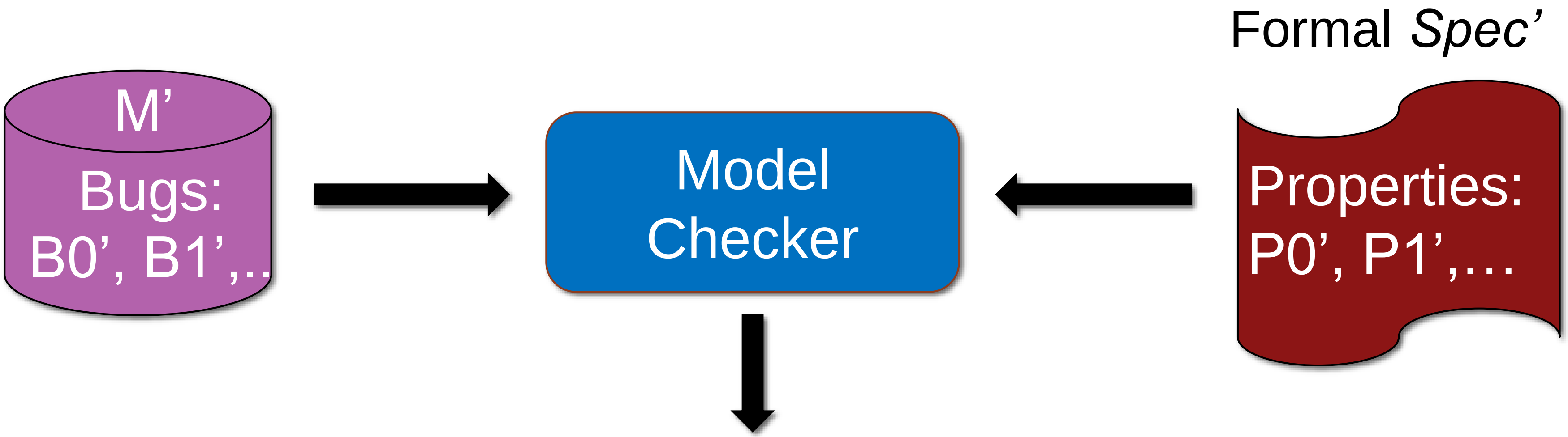
Completeness of Bug-Finding



Does P ∈ *Spec*’ hold in M’?

- *Spec*: manual writing of implementation-specific properties.
- Model/design changes → *Spec* to be adapted (manually).

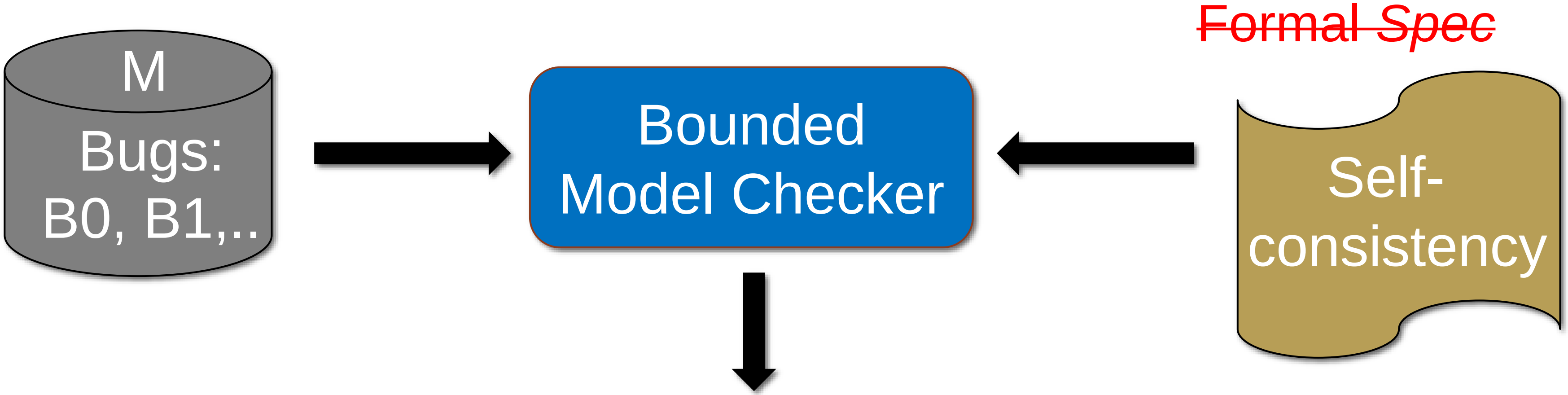
Completeness of Bug-Finding



Does P ∈ *Spec'* hold in M'?

- *Spec*: manual writing of implementation-specific properties.
- Model/design changes → *Spec* to be adapted (manually).
- Completeness depending on *Spec.*

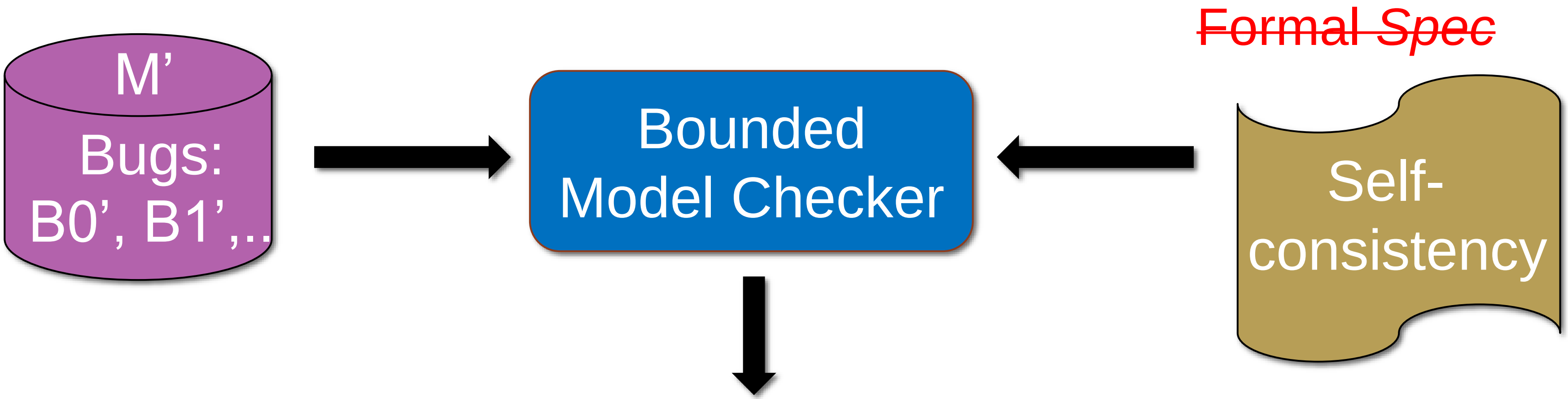
Symbolic Quick Error Detection (SQED)



Is M self-consistent?

- No need for *Spec* or implementation-specific properties.
- Leverages bounded model checking (BMC).

Symbolic Quick Error Detection (SQED)



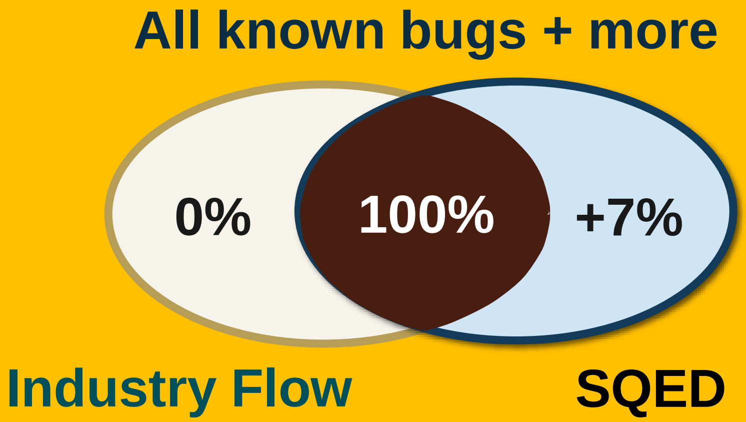
Is M self-consistent?

- No need for *Spec* or implementation-specific properties.
- Leverages bounded model checking (BMC).
- Self-consistency: universal property, no manual writing.

SQED: Industrial Strength

INFINEON case study: automotive IP versions [Singh et al DATE'19]

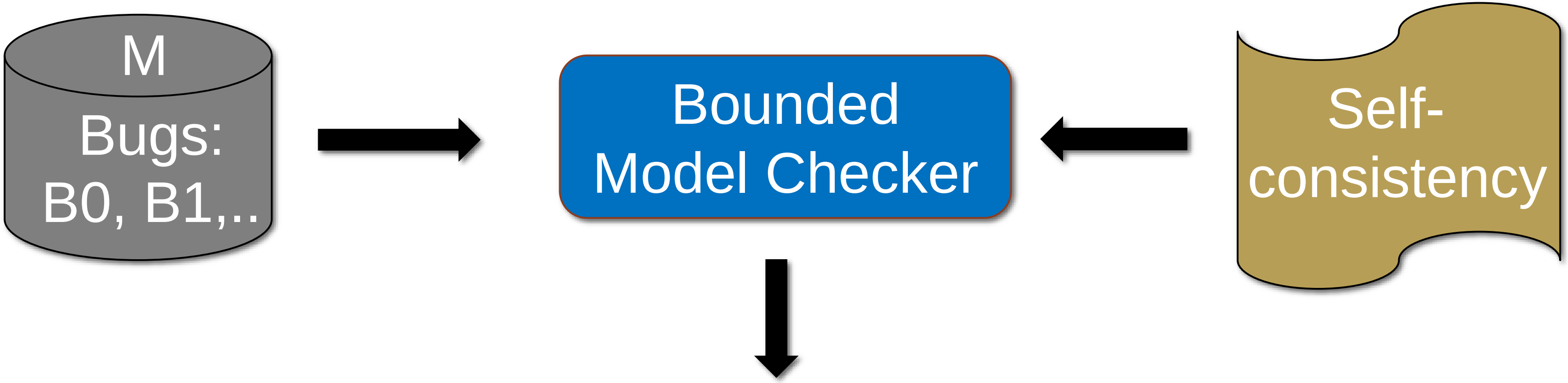
Bug detection



Traditional verification:

(Constrained) random simulation, directed tests, formal.

Our Contributions: SQED Formal Proofs



Does P ∈ *Spec* hold in M? ≈ Is M self-consistent?

1. Soundness: no spurious cex.
2. (Conditional) completeness: all bugs covered (BMC depth).
3. Formal framework: abstract processor model.

Self-Consistency

- Function f: equivalent inputs → equivalent outputs.
- Functional congruence property:

$$\forall x, x' : x = x' \rightarrow f(x) = f(x')$$

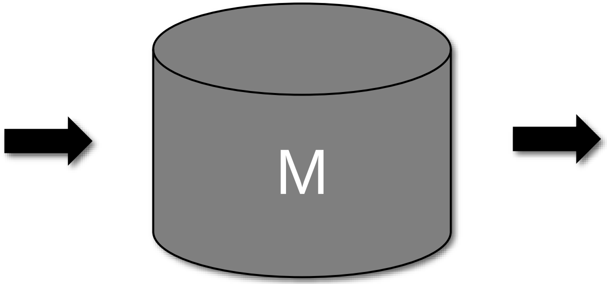
Self-Consistency

- Processor Design M:

$i_1, i_2, \ldots, i_n$ + inputs (regs/mem)

=

$i_1', i_2', \ldots, i_n'$ + inputs' (regs/mem)



outputs (regs/mem)

=

outputs' (regs/mem)

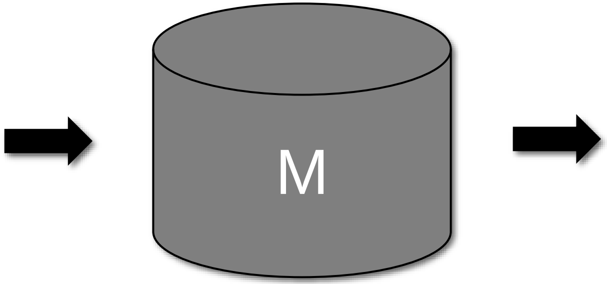
Self-Consistency

- Processor Design M:

$i_1, i_2, \dots, i_n$ + inputs (regs/mem)

=

$i_1', i_2', \dots, i_n'$ + inputs’ (regs/mem)



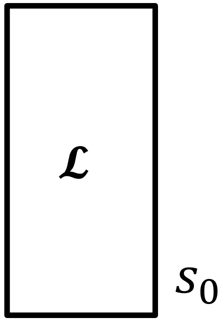
outputs (regs/mem)

=

outputs’ (regs/mem)

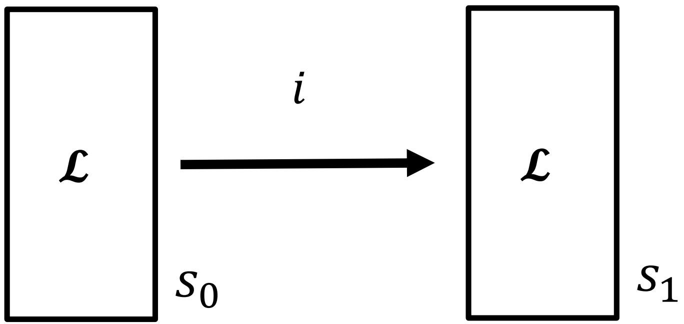
HW designs have complex internal state (pipeline,…).

Formal Model of Processors and SQED



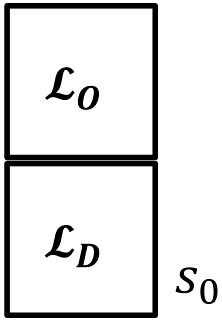
- State s_0: mapping from locations $\mathcal{L}$ to values.
- (Non-)architectural parts of $s_0 = (\mathrm{s_a}, \mathrm{s_{na}})$.
- $\mathcal{L}$: regs. and mem. locations, value $s_0(l) = \mathrm{s_a}(l) = v$.

Formal Model of Processors and SQED



- Instruction $i = \left(op, l, (l', l'')\right)$, one-step execution.
- Opcode op, input locations (l', l''), output location l.
- Transition: $\mathrm{T}(s_0, i) = \mathrm{s}_1$, $s_0 = (\mathrm{s_a}, \mathrm{s_{na}})$, $s_1 = (\mathrm{s_a}', \mathrm{s_{na}}')$.

Formal Model of Processors and SQED

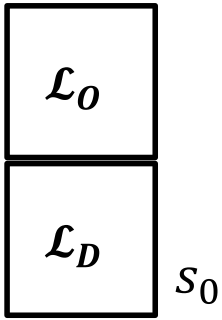


Example: register identifiers

$$\mathcal{L} = \{0,1,\ldots,31\}$$
$$\mathcal{L}_O = \{0,\ldots,15\}$$
$$\mathcal{L}_D = \{16,\ldots,31\}$$
$$\mathrm{L_D}(l) = l + 16$$

- Partition of $\mathcal{L}$: original and duplicate locations $\mathcal{L}_O$, $\mathcal{L}_D$.
- Arbitrary, fixed bijective mapping $\mathrm{L_D} : \mathcal{L}_O \rightarrow \mathcal{L}_D$.
- Self-consistency property based on mapping $\mathrm{L_D}$.

Formal Model of Processors and SQED

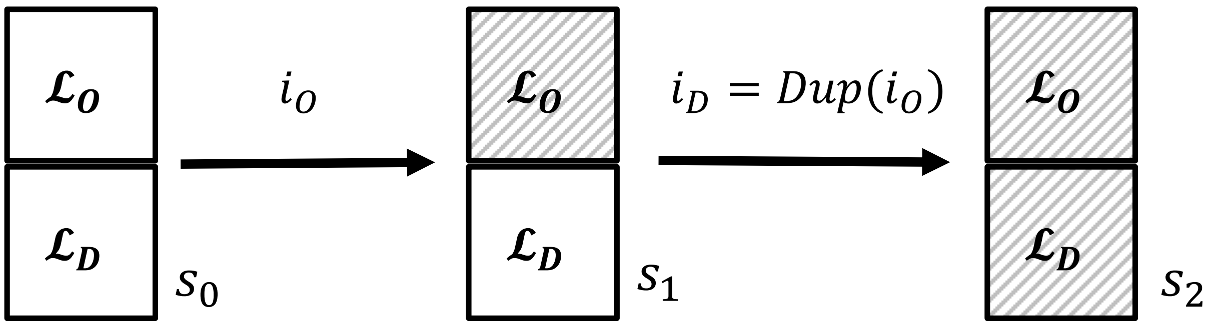


Example:

$$i_O = (\text{ADD}, l_{12}, (l_4, l_8))$$
$$\text{L}_\text{D}(l) = l + 16$$
$$i_D = (\text{ADD}, l_{28}, (l_{20}, l_{24}))$$

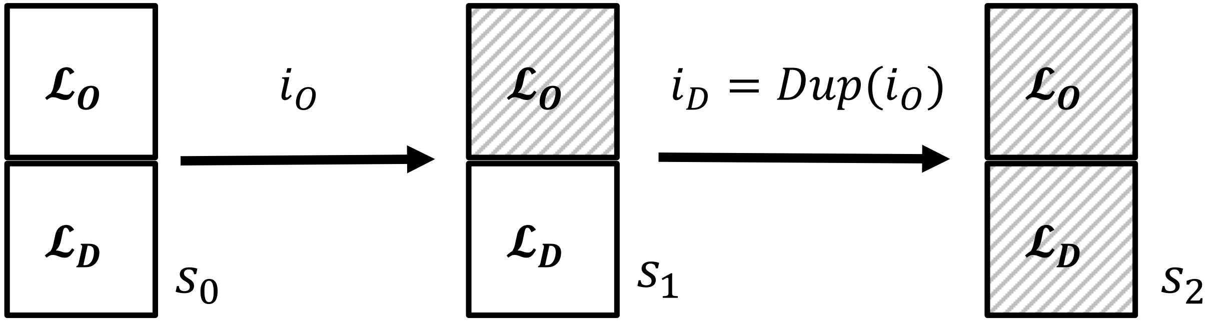
- Original instruction $i_O = (op, l, (l', l''))$.
- Duplicate $i_D = Dup(i_O) = (op, \text{L}_\text{D}(l), \text{L}_\text{D}(l', l''))$.

Formal Model of Processors and SQED



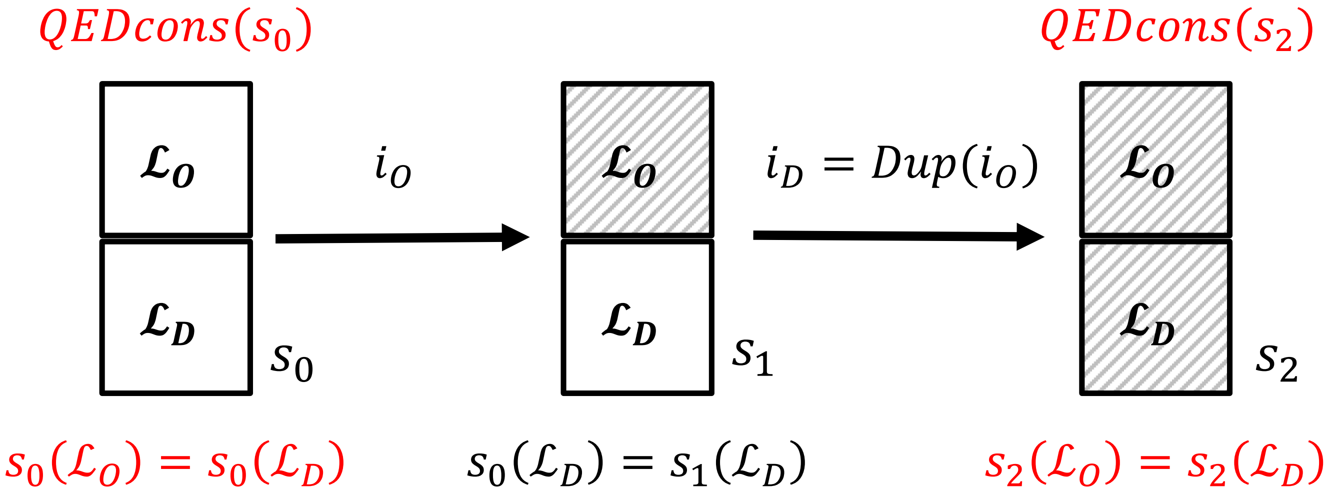
- Original instruction $i_O = (op, l, (l', l''))$.
- Duplicate $i_D = Dup(i_O) = (op, \mathrm{L_D}(l), \mathrm{L_D}(l', l''))$.
- Original/duplicate i_O/ i_D operates on $\boldsymbol{\mathcal{L}_O}/\boldsymbol{\mathcal{L}_D}$ only.

Formal Model of Processors and SQED



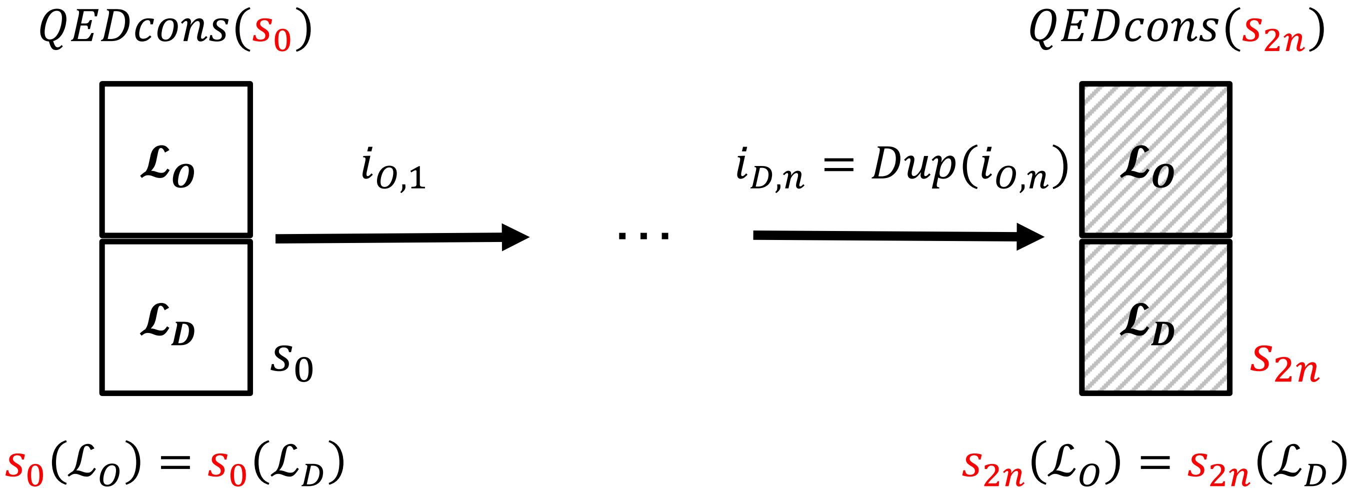
- Given L_D, state s_0 QED-consistent $\leftrightarrow s_0(\mathcal{L}_O) = s_0(\mathcal{L}_D)$.
- Matching values at original/duplicate locations.

Formal Model of Processors and SQED



- Given $\mathrm{L_D}$, state s_0 QED-consistent $\leftrightarrow s_0(\mathcal{L}_O) = s_0(\mathcal{L}_D)$.
- Matching values at original/duplicate locations.
- Correct execution of i_O/ i_D preserves QED-consistency.

Formal Model of Processors and SQED

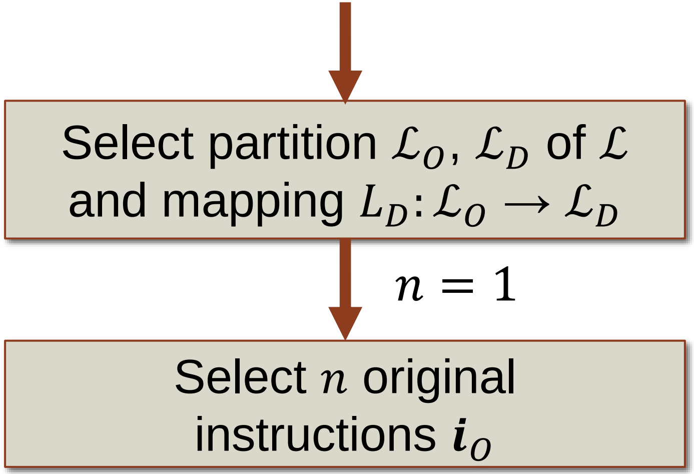


- $\boldsymbol{i}_O = i_{O,1}, \ldots, i_{O,n}$ and $\boldsymbol{i}_D = i_{D,1}, \ldots, i_{D,n}$ with $\boldsymbol{i}_D = Dup(\boldsymbol{i}_O)$.
- QED test: concatenation $\boldsymbol{i} = \boldsymbol{i}_O :: \boldsymbol{i}_D$ of $2n$ instructions.
- Correct execution of $\boldsymbol{i}$ preserves QED-consistency.

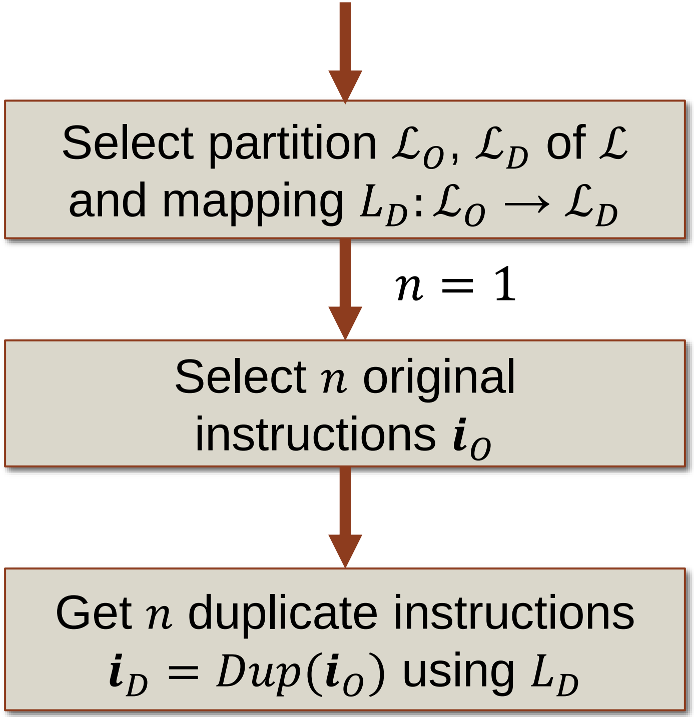
Using BMC in SQED

Select partition $\mathcal{L}_O$, $\mathcal{L}_D$ of $\mathcal{L}$
and mapping $L_D\colon \mathcal{L}_O \rightarrow \mathcal{L}_D$

Using BMC in SQED



Using BMC in SQED



Using BMC in SQED

Select partition $\mathcal{L}_O$, $\mathcal{L}_D$ of $\mathcal{L}$ and mapping $L_D: \mathcal{L}_O \rightarrow \mathcal{L}_D$

↓ $n = 1$

Select n original instructions $\boldsymbol{i}_O$

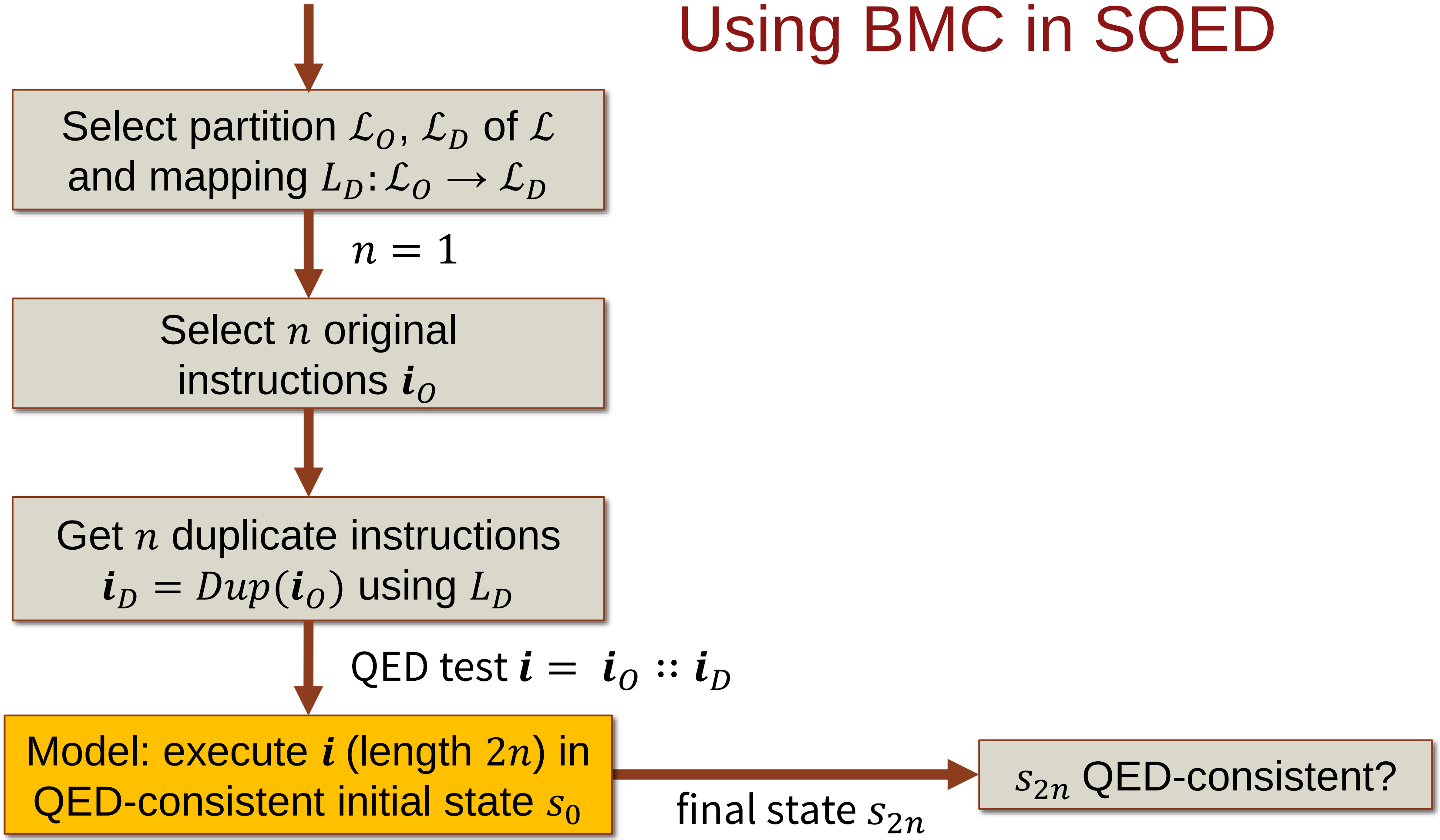
↓

Get n duplicate instructions $\boldsymbol{i}_D = Dup(\boldsymbol{i}_O)$ using L_D

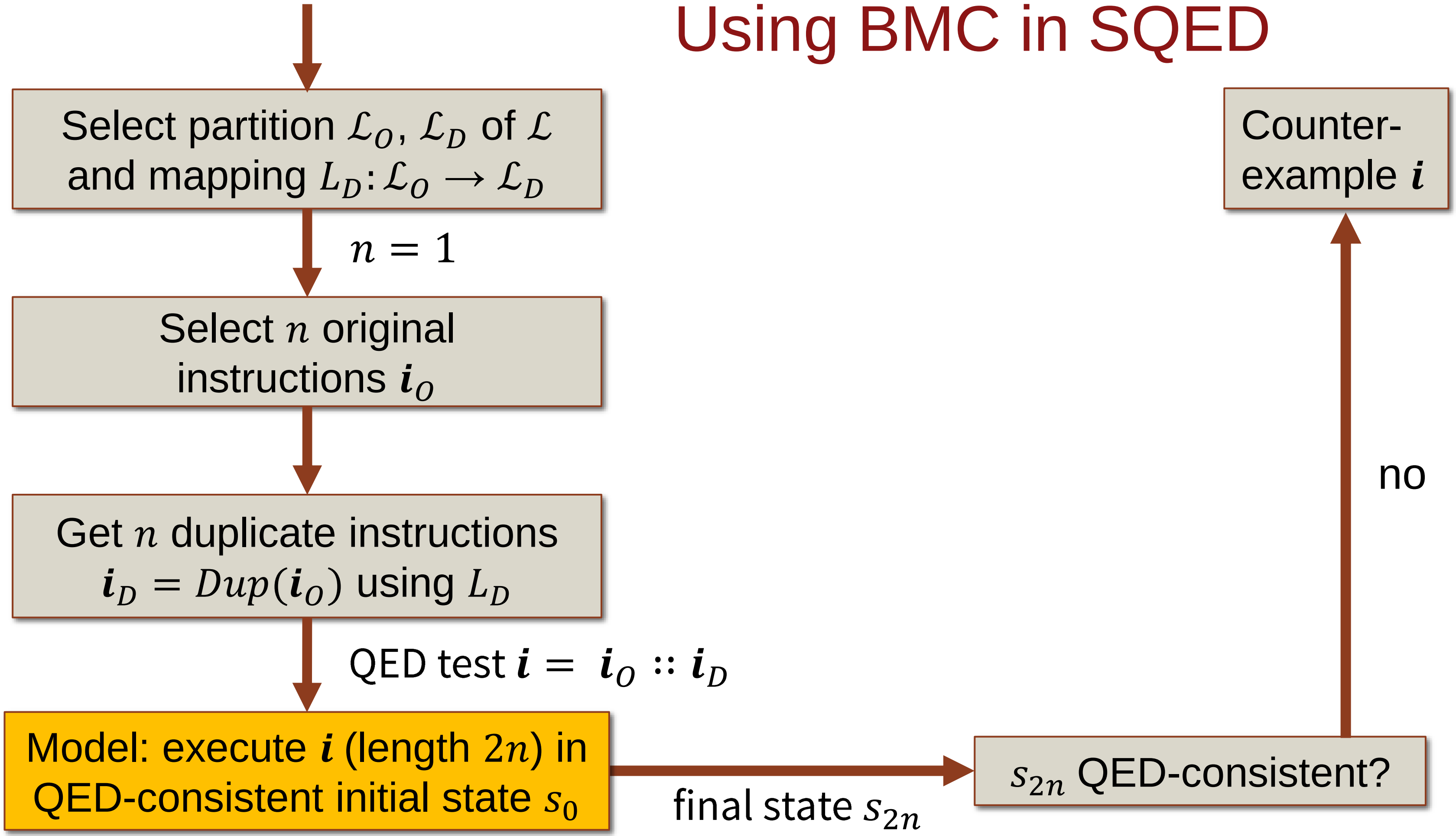
↓ QED test $\boldsymbol{i} = \boldsymbol{i}_O :: \boldsymbol{i}_D$

Model: execute $\boldsymbol{i}$ (length $2n$) in QED-consistent initial state s_0

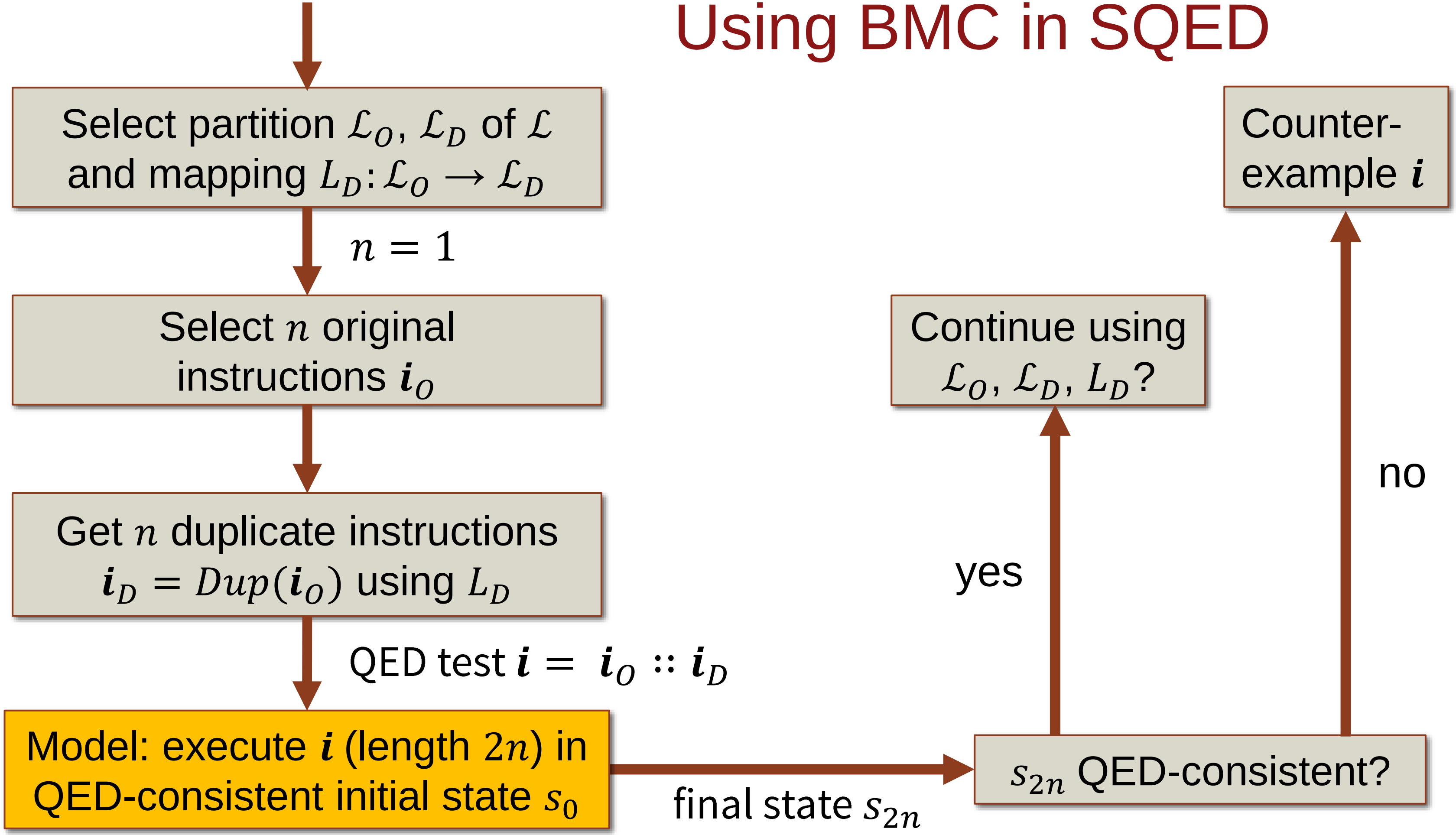
Using BMC in SQED



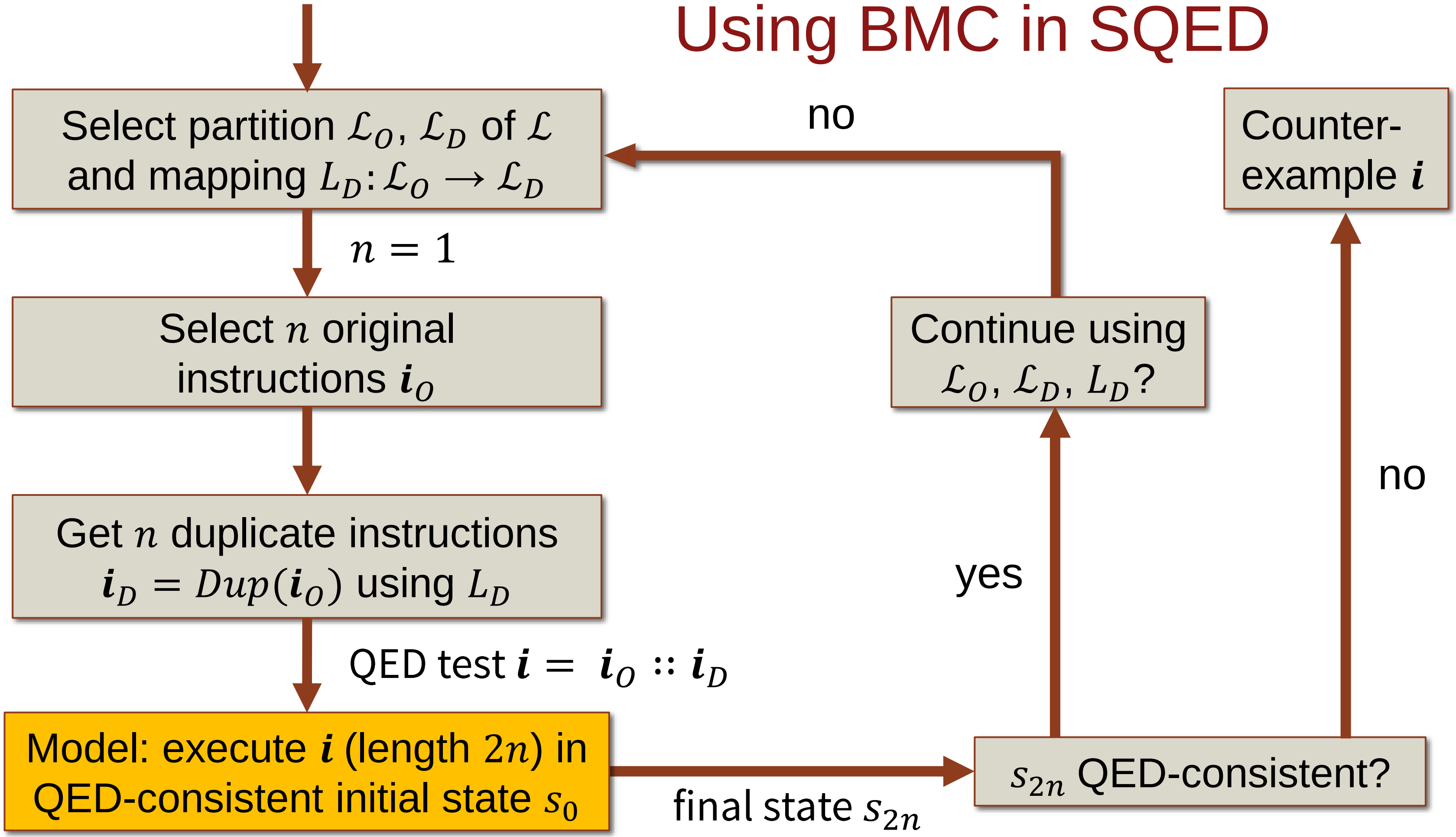
Using BMC in SQED



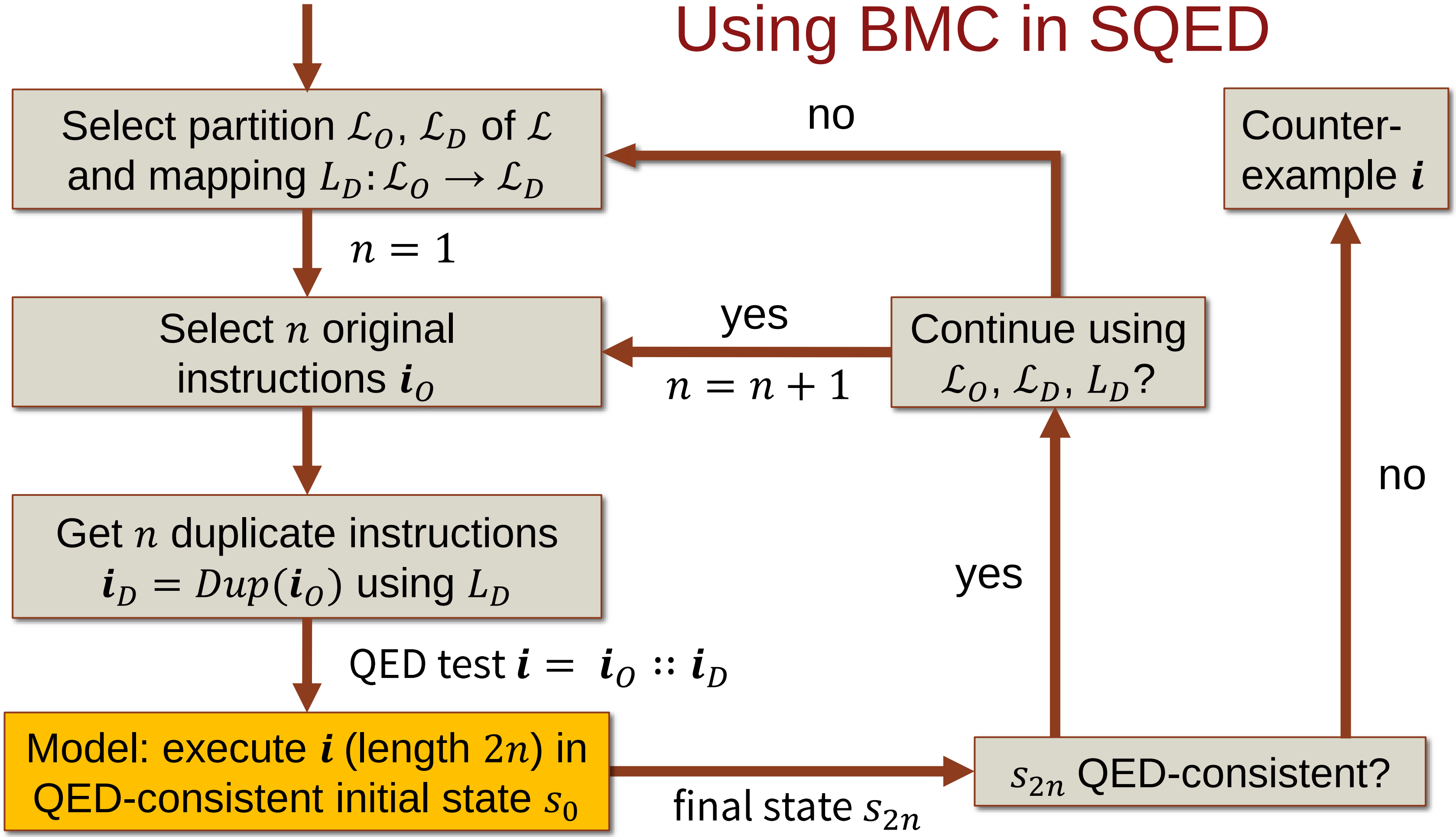
Using BMC in SQED



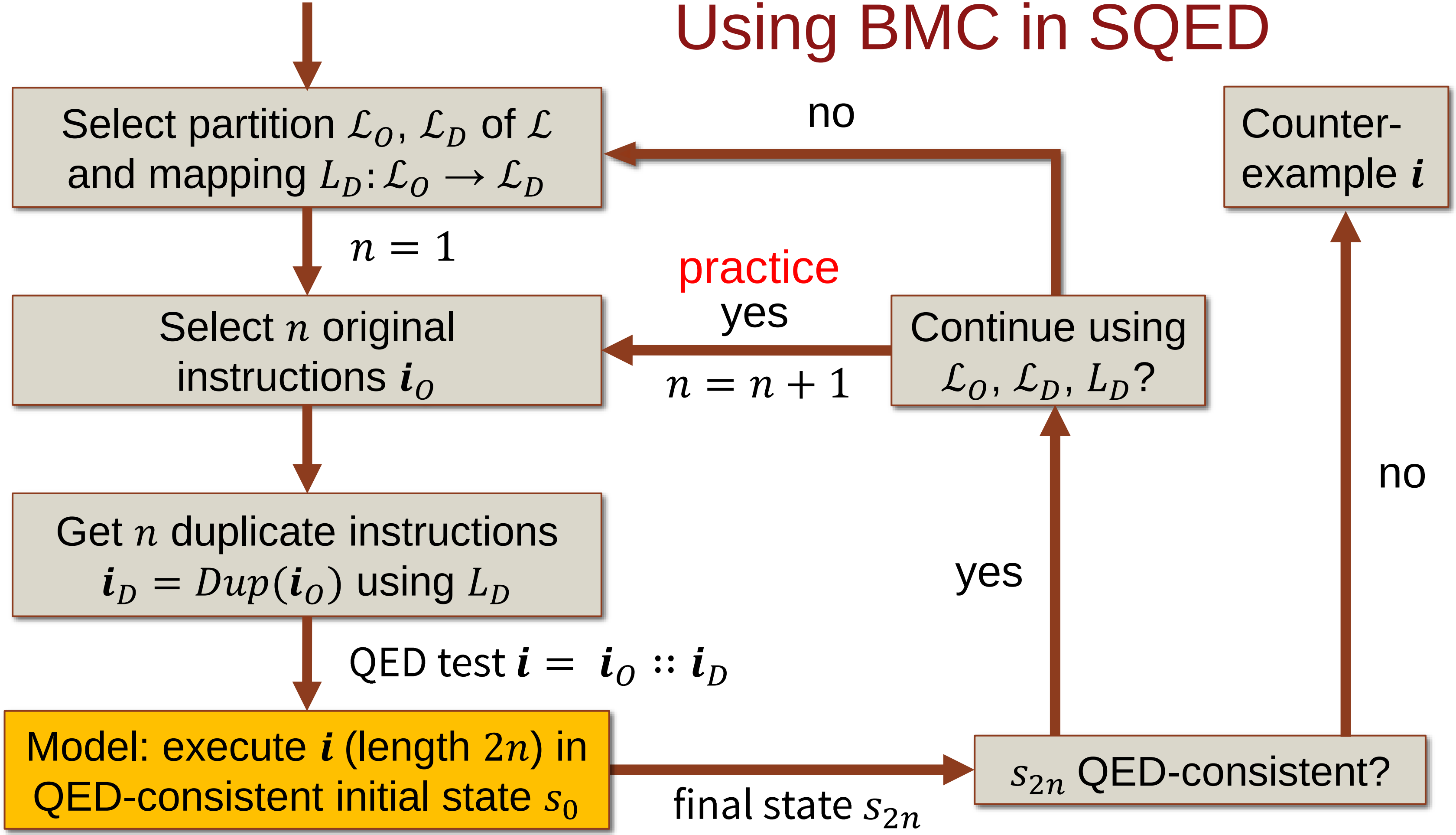
Using BMC in SQED



Using BMC in SQED



Using BMC in SQED



Abstract Specification Relation

$$\forall s, s' \in S, i \in I.\ Spec(s, i, s') \leftrightarrow \forall\, l \in \mathcal{L}.$$
$$\left(l \ \neq LocOut(i) \rightarrow s(l) = s'(l)\right) \wedge$$
$$(l = LocOut(i) \rightarrow s'(l) = SpecOut(i, s(LocIn(i))))$$

- Transition $\mathrm{T}(s, i) = s'$

Abstract Specification Relation

$$\forall s, s' \in S, i \in I.\ Spec(s, i, s') \leftrightarrow \forall\ l \in \mathcal{L}.$$
$$\big(l \neq LocOut(i) \rightarrow s(l) = s'(l)\big) \wedge$$
$$(l = LocOut(i) \rightarrow s'(l) = SpecOut(i, s(LocIn(i))))$$

- Transition $\mathrm{T}(s, i) = s'$ according to $Spec \subseteq S \times I \times S$ iff:

Abstract Specification Relation

$$\forall s, s' \in S, i \in I.\ Spec(s, i, s') \leftrightarrow \forall\, l \in \mathcal{L}.$$
$$\big(l \neq LocOut(i) \rightarrow s(l) = s'(l)\big) \wedge$$
$$(l = LocOut(i) \rightarrow s'(l) = SpecOut(i, s(LocIn(i))))$$

- Transition $\mathrm{T}(\mathrm{s}, i) = \mathrm{s}'$ according to $Spec \subseteq S$ x I x S iff:
  1. all non-output locations unchanged, and

Abstract Specification Relation

$$\forall s, s' \in S, i \in I.\ Spec(s, i, s') \leftrightarrow \forall\, l \in \mathcal{L}.$$
$$\big(l \neq LocOut(i) \rightarrow s(l) = s'(l)\big) \wedge$$
$$(l = LocOut(i) \rightarrow s'(l) = SpecOut(i, s(LocIn(i))))$$

- Transition $\mathrm{T}(\mathrm{s}, i) = \mathrm{s}'$ according to $Spec \subseteq S$ x I x S iff:
  1. all non-output locations unchanged, and
  2. correct output produced for given input values.
     - Output specification: $SpecOut\colon I$ x $\mathcal{V}^2 \rightarrow \mathcal{V}$.

Abstract Specification Relation

$$\forall s, s' \in S, i \in I.\ Spec(s, i, s') \leftrightarrow \forall\, l \in \mathcal{L}.$$
$$\big(l \neq LocOut(i) \rightarrow s(l) = s'(l)\big) \wedge$$
$$(l = LocOut(i) \rightarrow s'(l) = SpecOut(i, s(LocIn(i))))$$

- Transition $\mathrm{T}(\mathrm{s}, i) = \mathrm{s}'$ according to $Spec \subseteq S \times I \times S$ iff:
  1. all non-output locations unchanged, and
  2. correct output produced for given input values.
     - Output specification: $SpecOut: I \times \mathcal{V}^2 \rightarrow \mathcal{V}$.
- Abstract spec needed only for theory, not practice.

Bugs and Processor Correctness

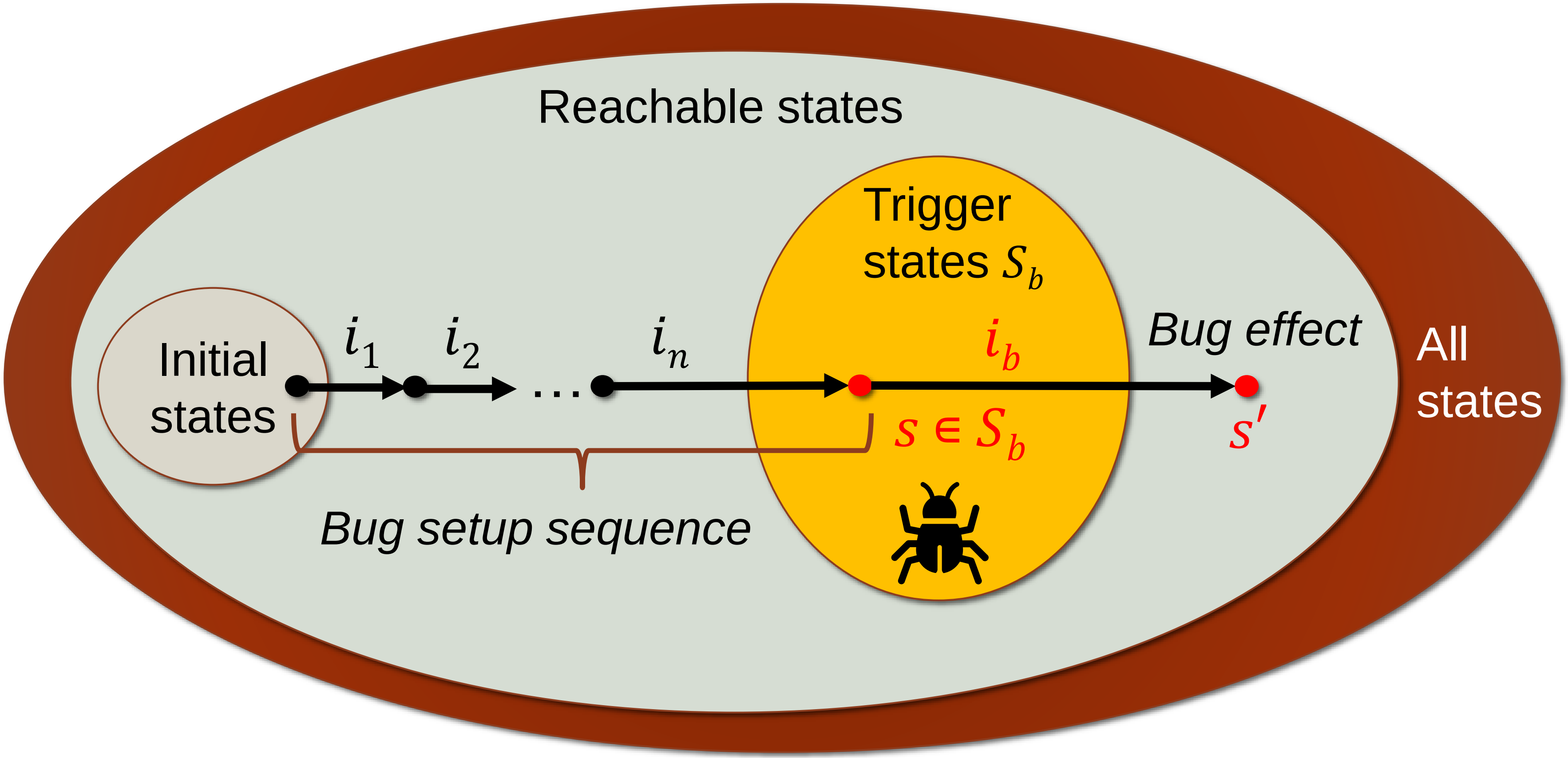
Processor P is correct wrt. Spec:

- $\forall s \in S, i \in I.\ reach(s) \rightarrow Spec(s, i, T(s, i))$.
- All instructions execute correctly in all reachable states.

Bug:

- Instruction i_b and set $S_b \subseteq S$ of bug-triggering states.
- $S_b = \{s \in S \mid reach(s) \land \sim Spec(s, i_b, T(s, i_b))\}$

Bug Triggering



Single-Instruction Bugs and Correctness

Processor P is single-instruction (SI) correct wrt. Spec:

- $\forall s \in Init, i \in I.\ Spec(s, i, T(s, i))$.
- All instructions execute correctly in all initial states $Init$.

Single-Instruction Bugs and Correctness

Processor P is single-instruction (SI) correct wrt. Spec:

- $\forall s \in Init, i \in I.\ Spec(s, i, T(s, i))$.
- All instructions execute correctly in all initial states $Init$.

Single-instruction (SI) bug:

- $\exists s \in Init, i \in I.\ {\sim}Spec(s, i, T(s, i))$.
- No setup sequence, well-studied approaches to checking.

Single-Instruction Bugs and Correctness

Processor P is single-instruction (SI) correct wrt. Spec:

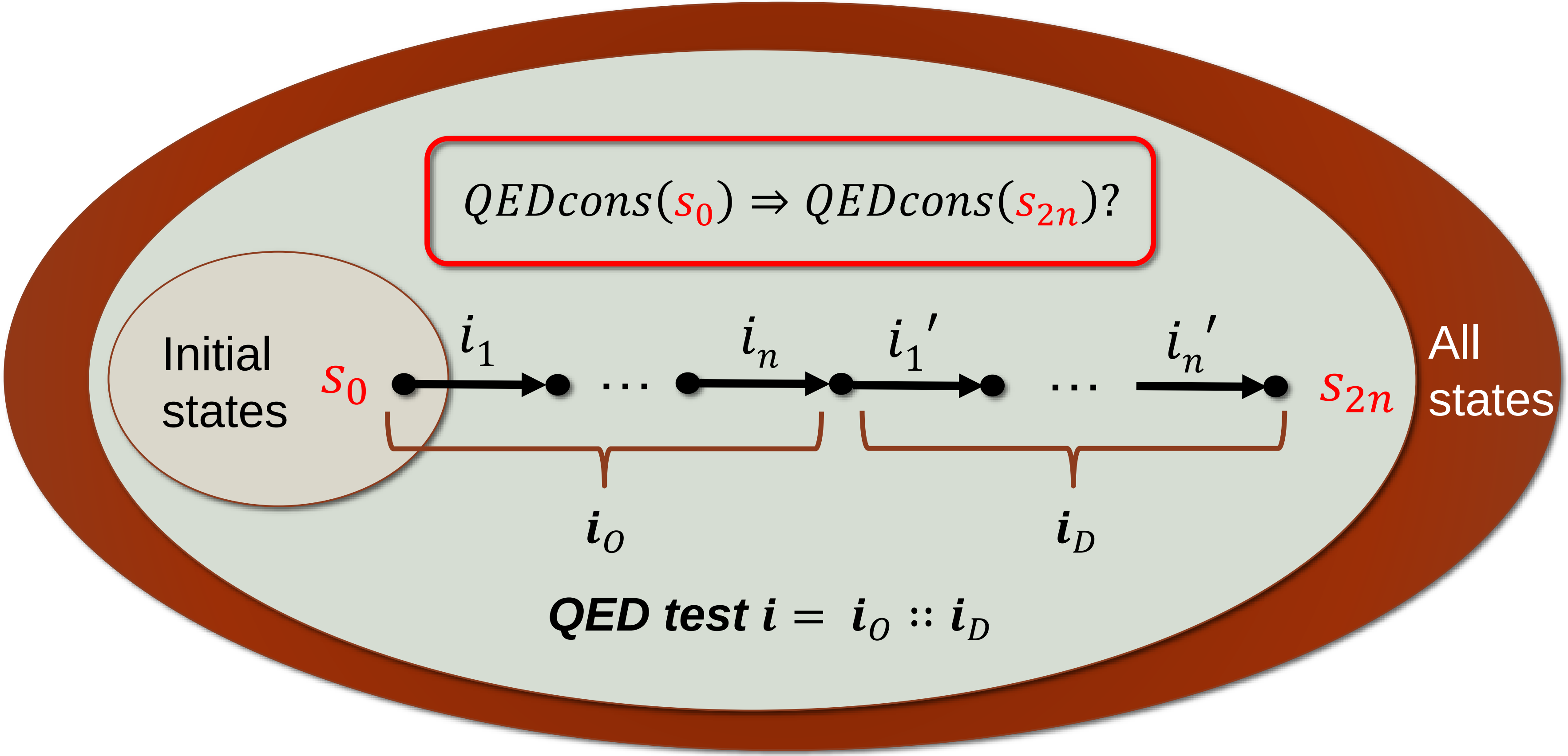
- $\forall s \in Init, i \in I.\ Spec(s, i, T(s, i))$.
- All instructions execute correctly in all initial states $Init$.

Single-instruction (SI) bug:

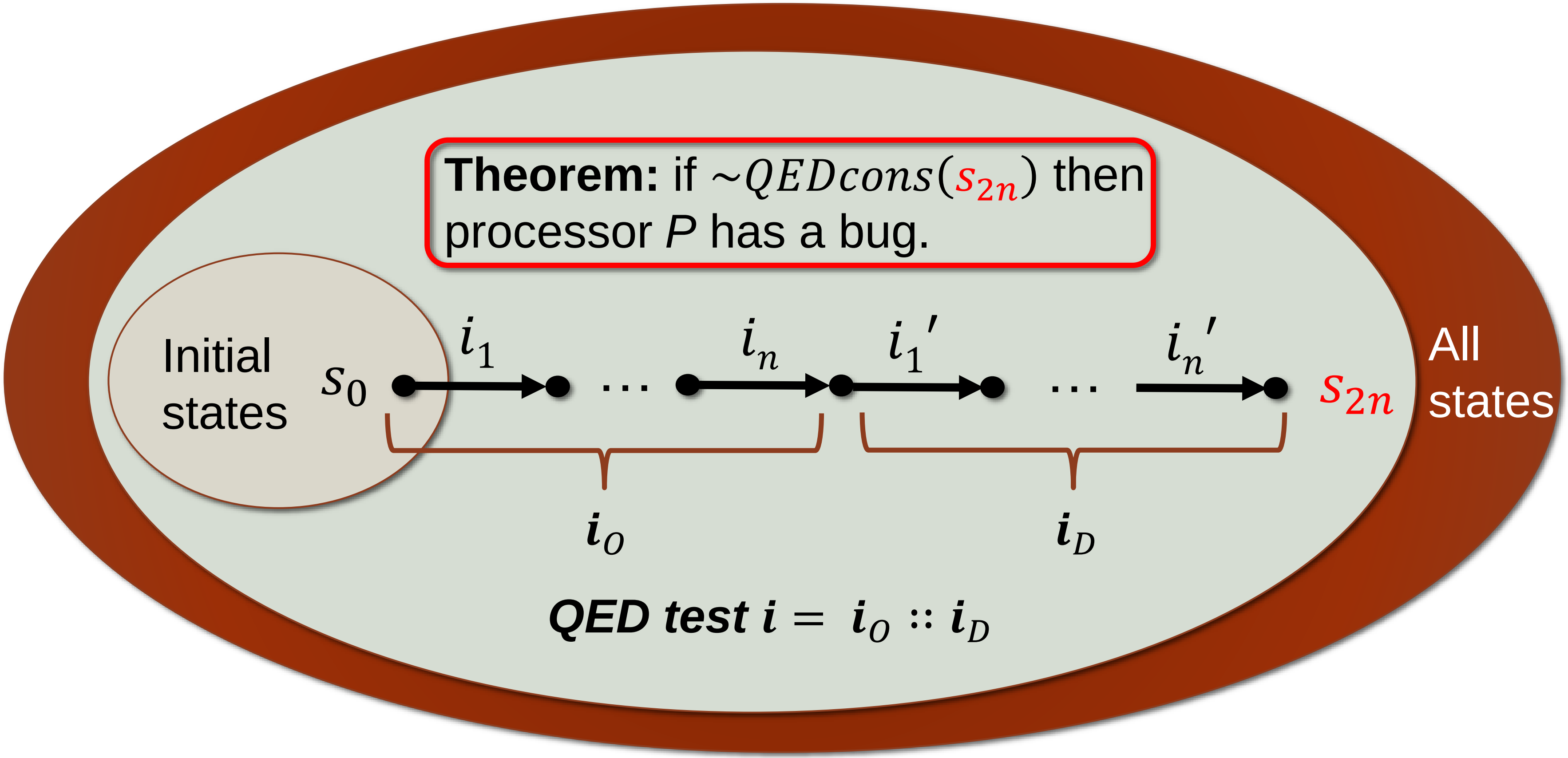
- $\exists s \in Init, i \in I.\ \sim Spec(s, i, T(s, i))$.
- No setup sequence, well-studied approaches to checking.

Assumption: P is SI-correct.

Soundness of SQED

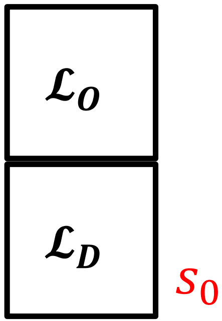


Soundness of SQED



Towards Completeness: Bug-Specific QED Test

$QEDcons(s_0)$



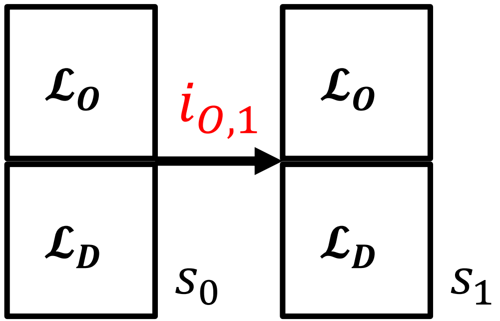
Initial state

QED test $\boldsymbol{i} = (i_{O,1}, \ldots, i_{O,n}) :: (i_{D,1}, \ldots, i_{D,n})$ for some L_D.

- Flexibility in choosing L_D.
- QED-consistent initial state s_0.

Towards Completeness: Bug-Specific QED Test

$QEDcons(s_0)$

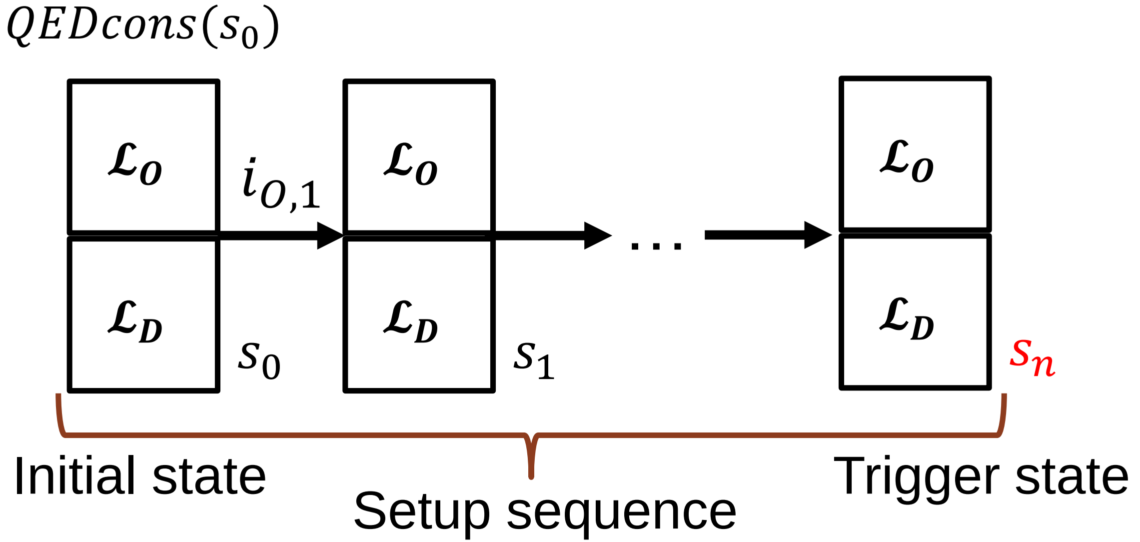


Initial state

QED test $\boldsymbol{i} = (i_{O,1}, \ldots, i_{O,n}) :: (i_{D,1}, \ldots, i_{D,n})$ for some L_D.

- QED-consistent initial state s_0.
- Let $i_b = Dup(i_{O,1})$: $i_{O,1}$ meets *Spec* due to SI-correctness.

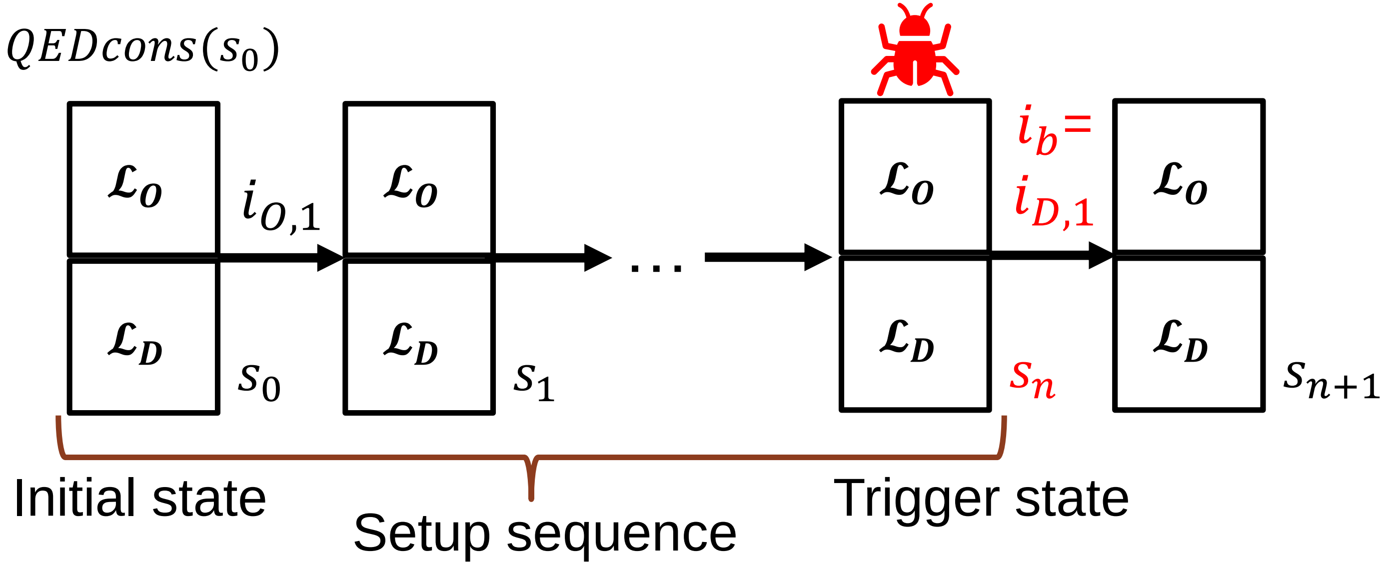
Towards Completeness: Bug-Specific QED Test



QED test $\boldsymbol{i} = (i_{O,1}, \ldots, i_{O,n}) :: (i_{D,1}, \ldots, i_{D,n})$ for some L_D.

- Setup sequence $i_{O,1}, \ldots, i_{O,n}$ to reach triggering state $s_n \in S_b$.

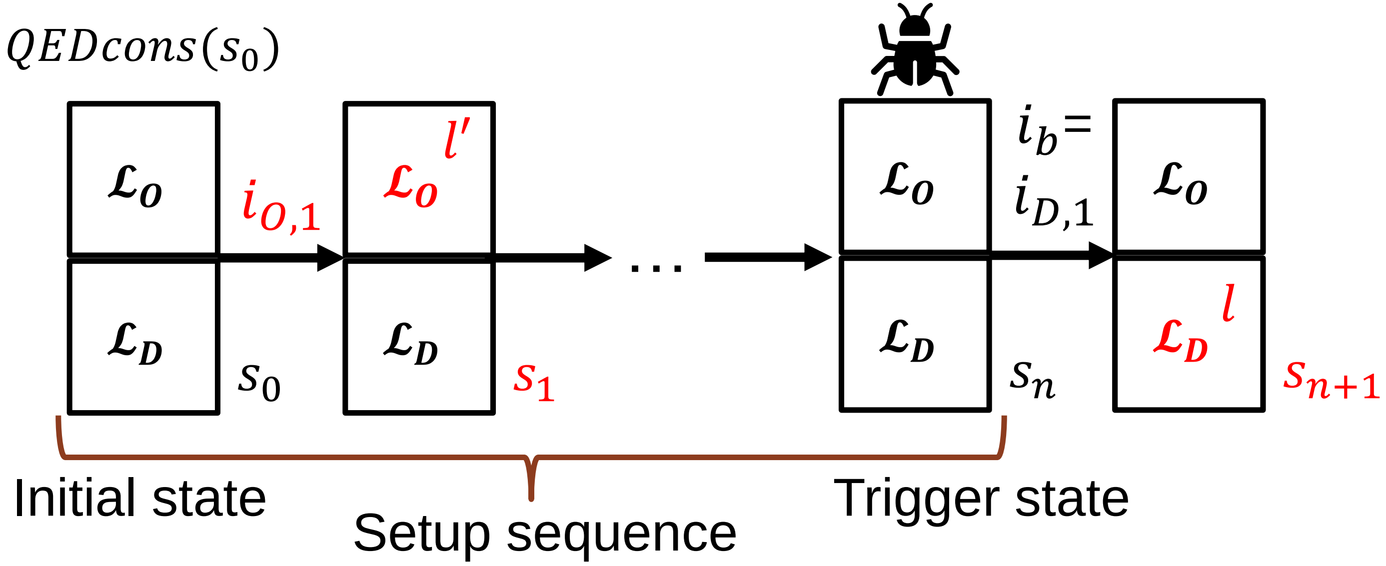
Towards Completeness: Bug-Specific QED Test



QED test $\boldsymbol{i} = (i_{O,1}, \dots, i_{O,n}) :: (i_{D,1}, \dots, i_{D,n})$ for some L_D.

- Setup sequence $i_{O,1}, \dots, i_{O,n}$ to reach triggering state $s_n \in S_b$.
- Bug instruction $i_b = Dup(i_{O,1})$ fails in s_n.

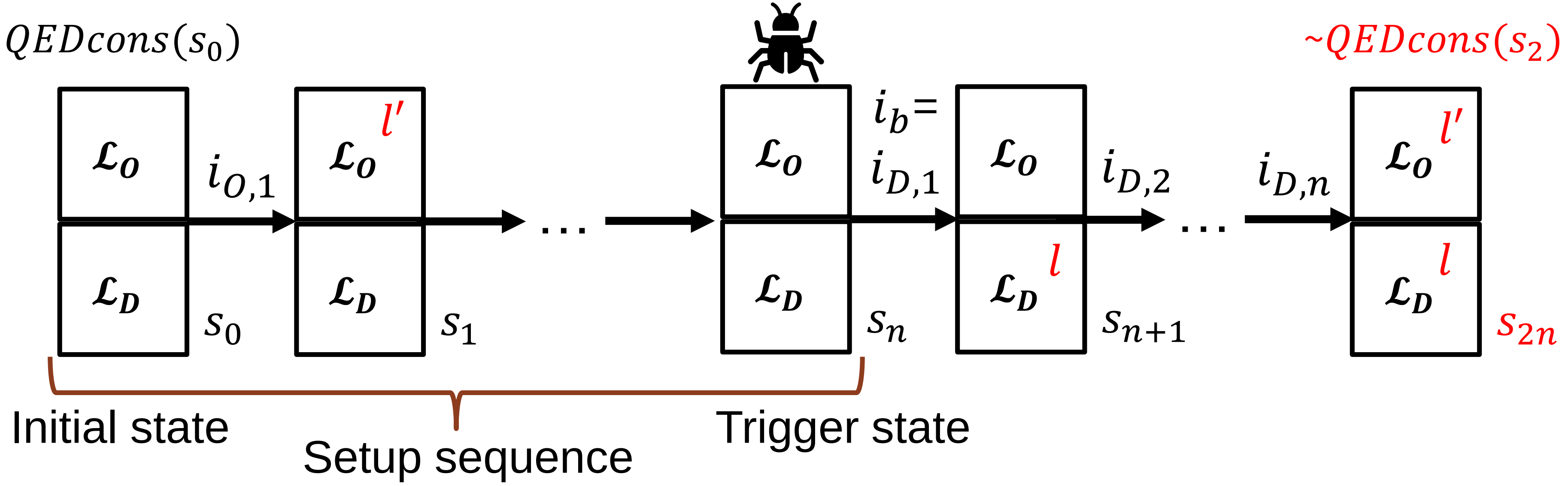
Towards Completeness: Bug-Specific QED Test



QED test $\boldsymbol{i} = (i_{O,1}, \dots, i_{O,n}) :: (i_{D,1}, \dots, i_{D,n})$ for some L_D.

- E.g. wrong value at output location l of i_b in s_{n+1}.
- Correct value at original output location l' of $i_{O,1}$ in s_1.

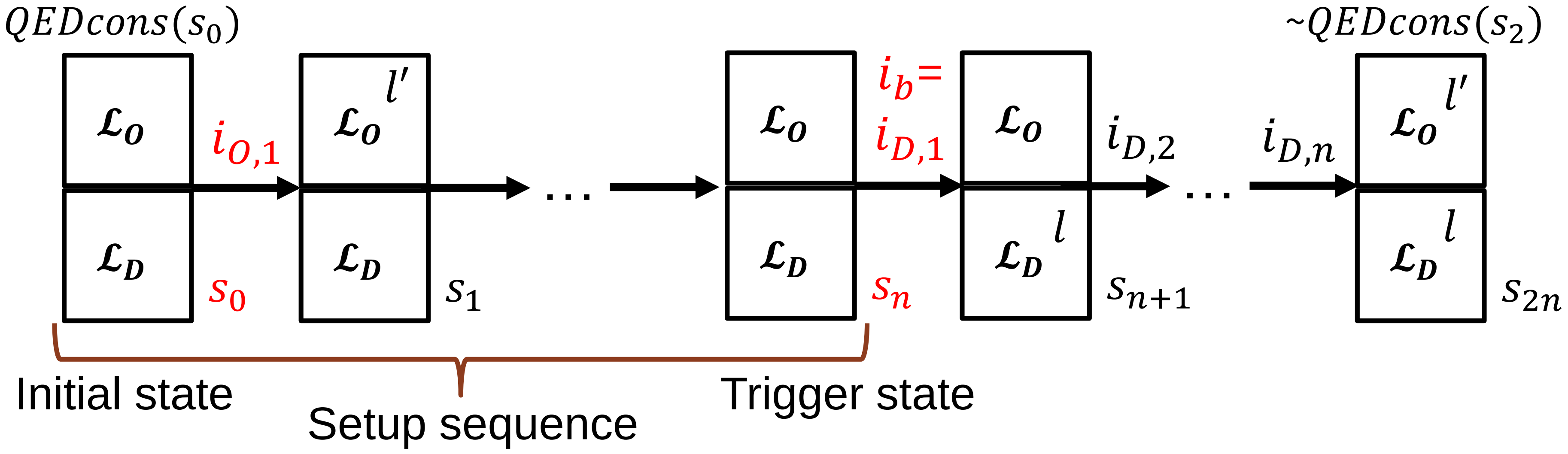
Towards Completeness: Bug-Specific QED Test



QED test $\boldsymbol{i} = (i_{O,1}, \ldots, i_{O,n}) :: (i_{D,1}, \ldots, i_{D,n})$ for some L_D.

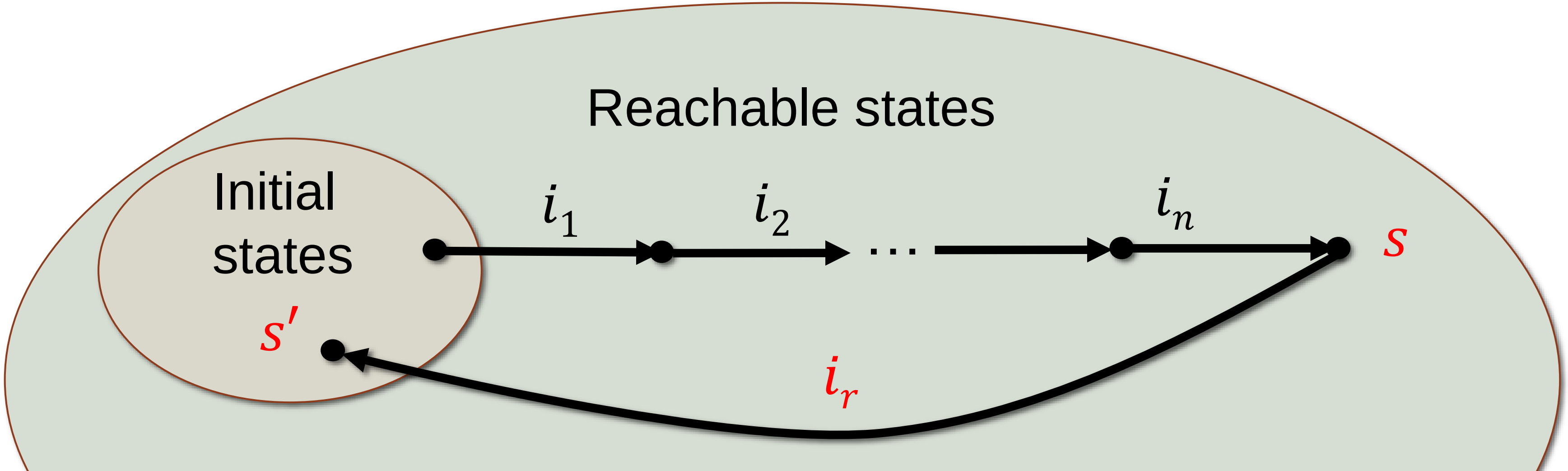
- Mismatching values at locations l and l' in s_{2n}.
- Final state s_{2n} QED-inconsistent.

Conditional Completeness



Theorem: *if a bug-specific QED test **i** exists, then **i** fails.*

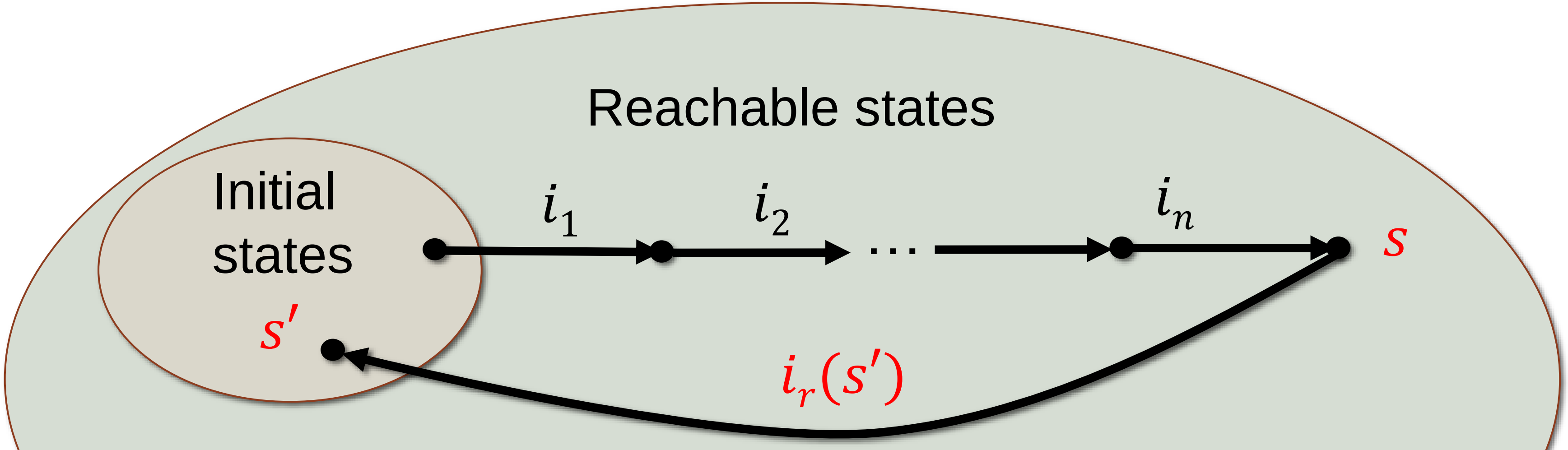
Extensions: Reset Instructions



Soft-reset instruction i_r:

- $s = (s_a, s_{na})$, $s' = (s_a', s_{na}')$.
- Keep arch. part: $s_a = s_a'$.

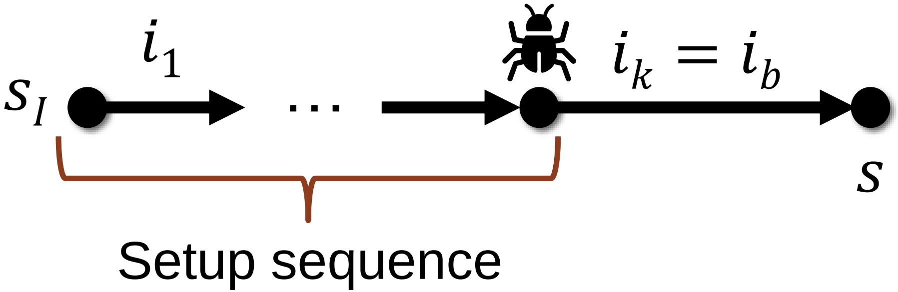
Extensions: Reset Instructions



Hard-reset instruction $i_r(s')$:

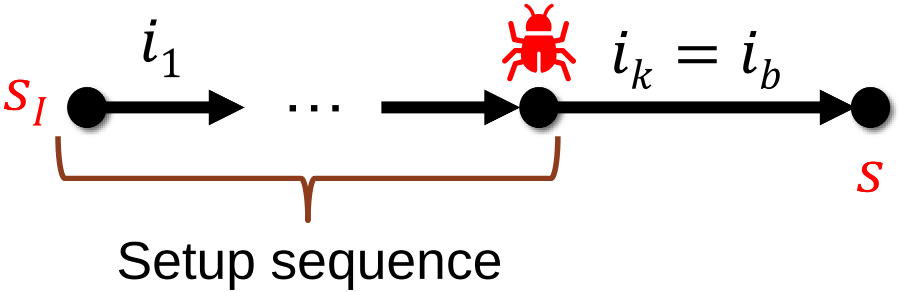
- $s = (s_a, s_{na})$, $s' = (s_a{}', s_{na}{}')$.
- Change s_a and s_{na} arbitrarily.

Extensions: QED Test with Reset



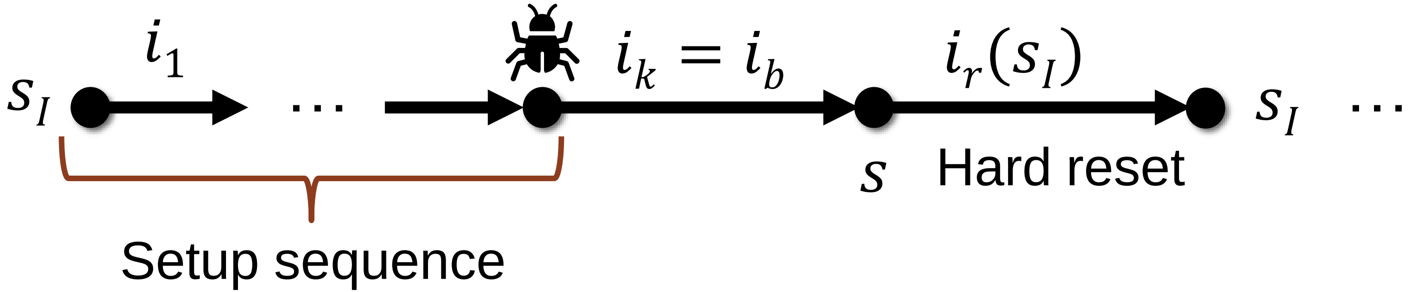
- Bug set up and triggered by $i_1, \ldots, i_k = i_b$.
- No duplication: check states after $i_k = i_b$ with(out) reset.

Extensions: QED Test with Reset



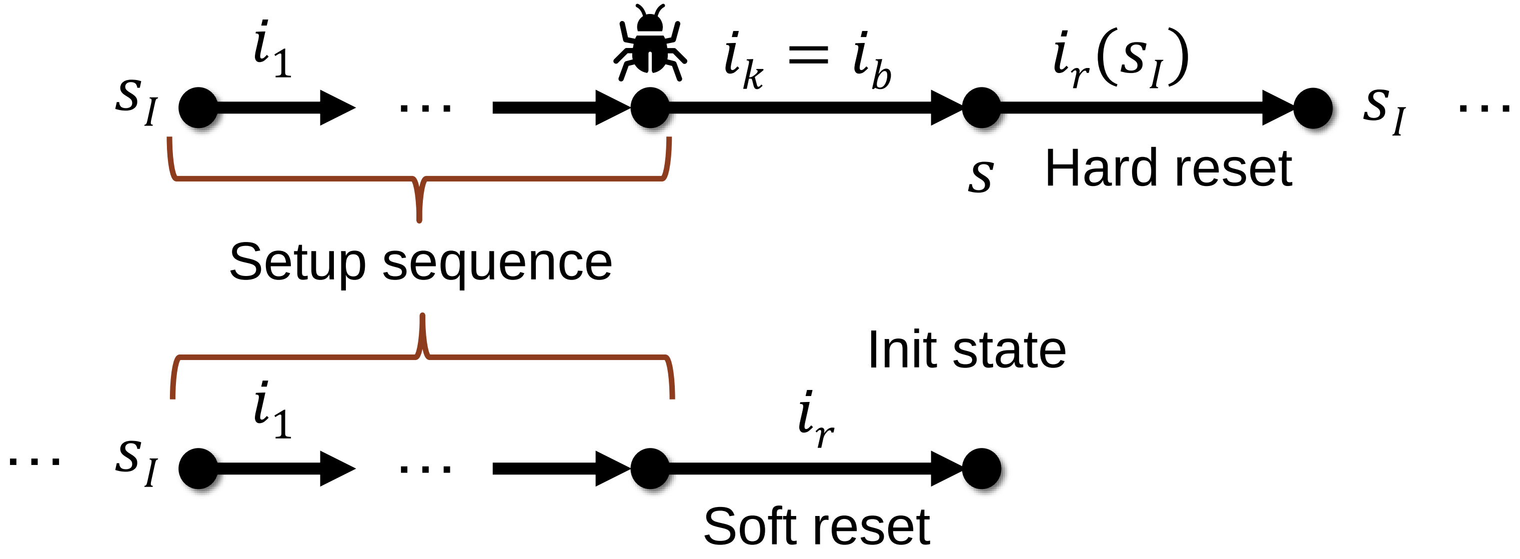
- Bug set up and triggered by $i_1, \ldots, i_k = i_b$.
- Execute $i_1, \ldots, i_k = i_b$ from $s_I \in Init$: wrong value in state s.

Extensions: QED Test with Reset



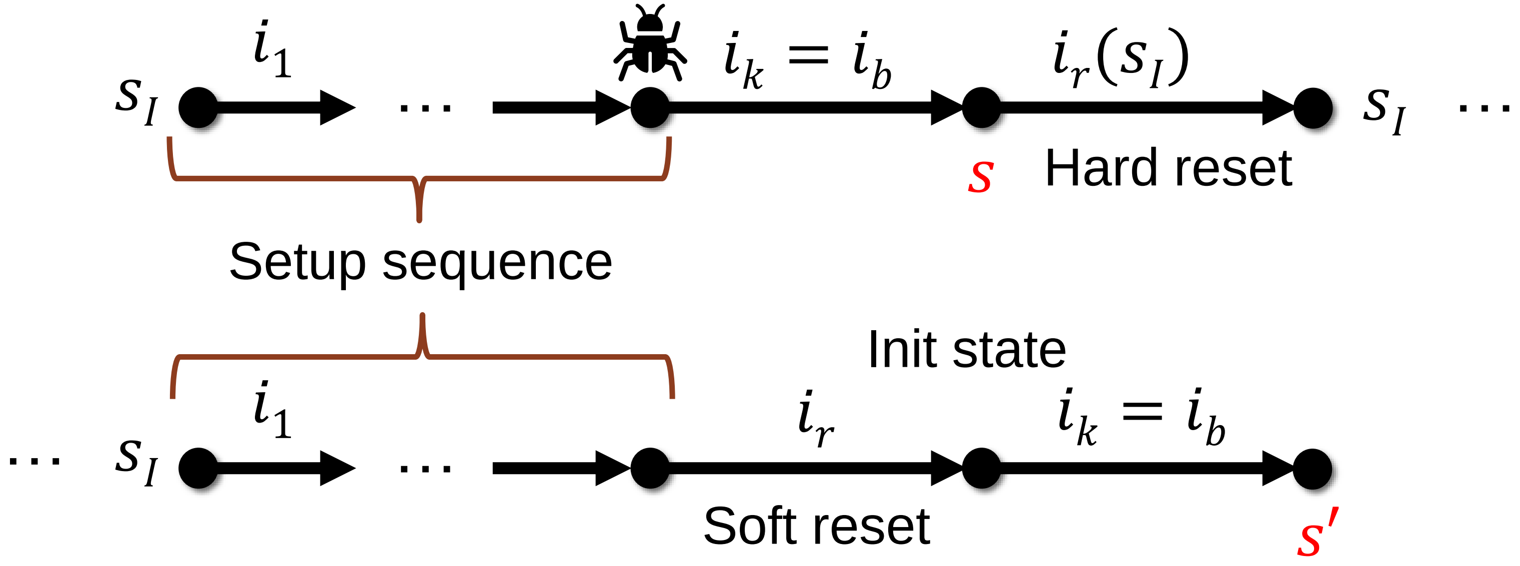
- Execute hard reset in state s, get back to s_I.
- Idea: execute $i_1, \ldots, i_k = i_b$ again with soft reset before i_b.

Extensions: QED Test with Reset



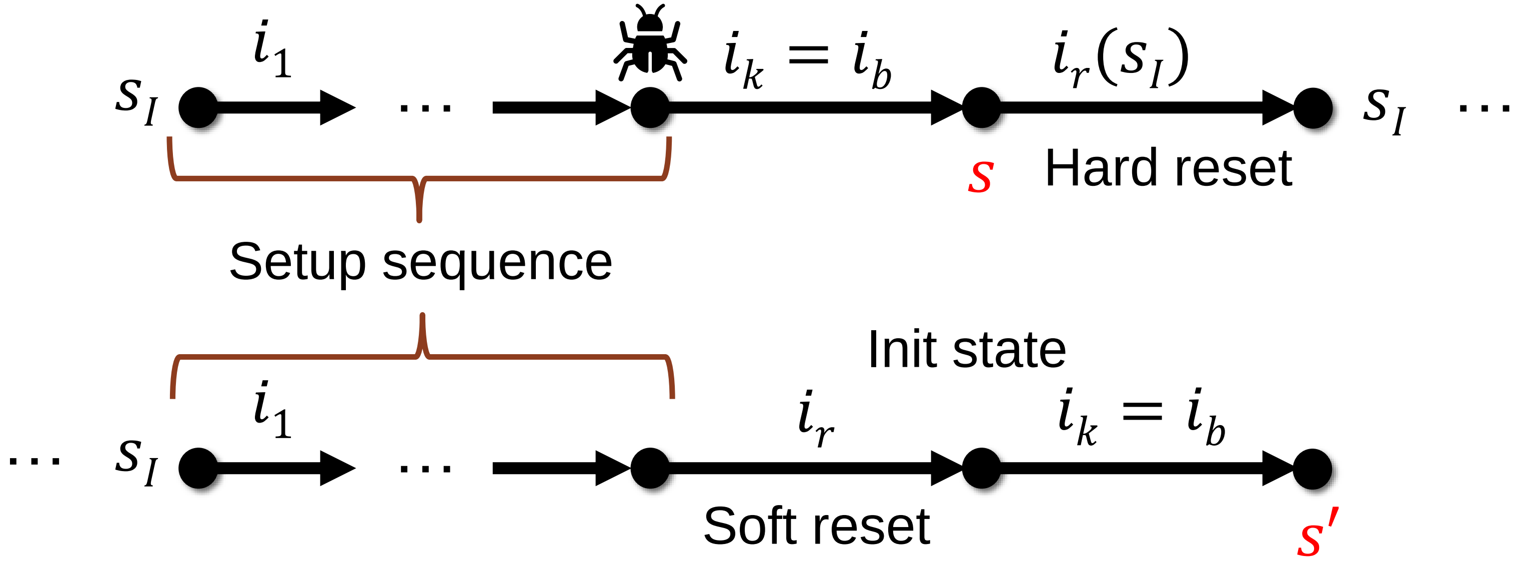
- Execute soft reset in bug-triggering state before $i_k = i_b$.
- Make use of SI correctness.

Extensions: QED Test with Reset



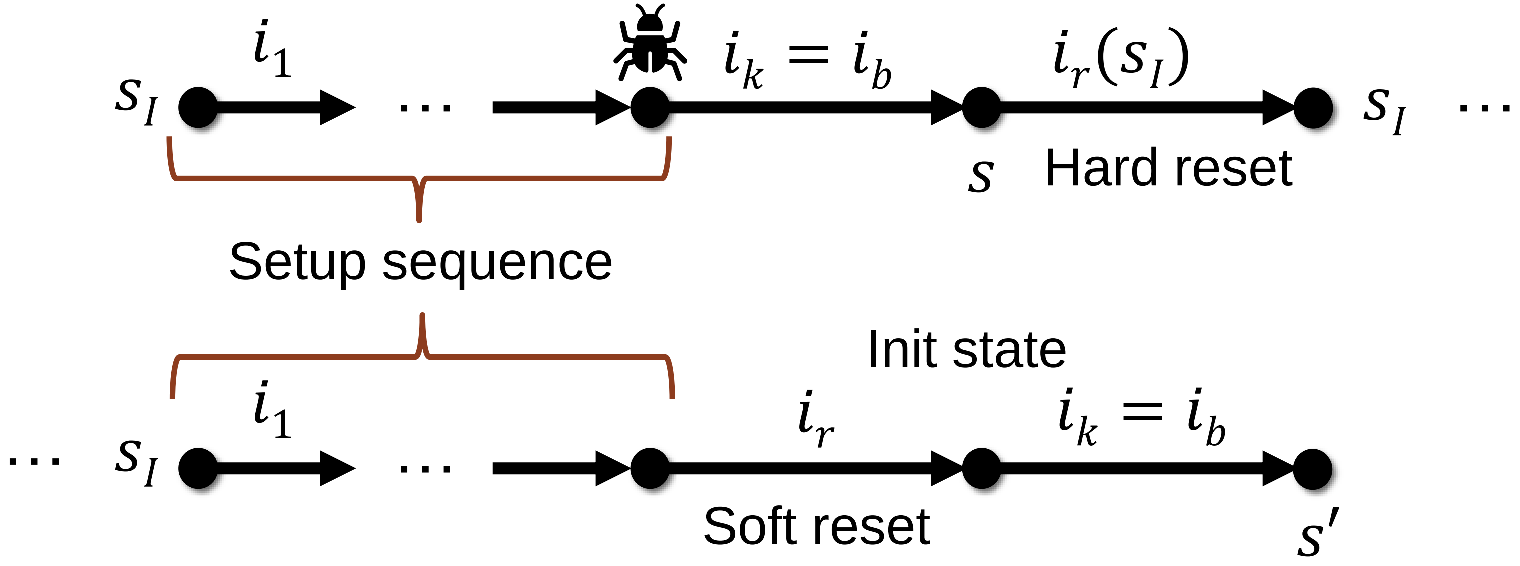
- Bug instruction $i_k = i_b$ executes correctly.
- Compare s and final state s'.

Extensions: QED Test with Reset



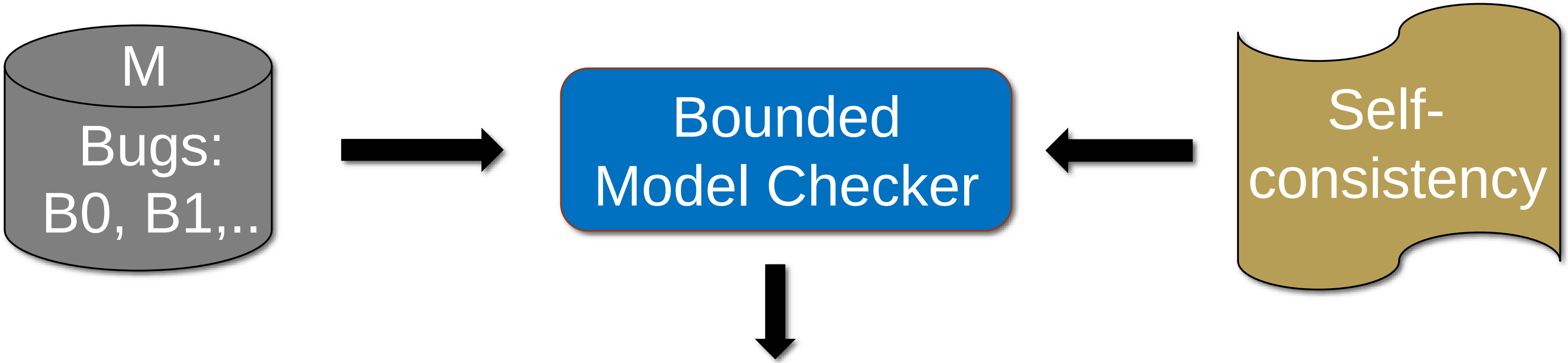
- QED test with reset fails iff $s(l) \neq s'(l)$ for a location l.

Extensions: QED Test with Reset



Theorem (full completeness): *if P is SI correct and has no failing QED test with reset, then P is correct.*

Summary: SQED Soundness and Completeness

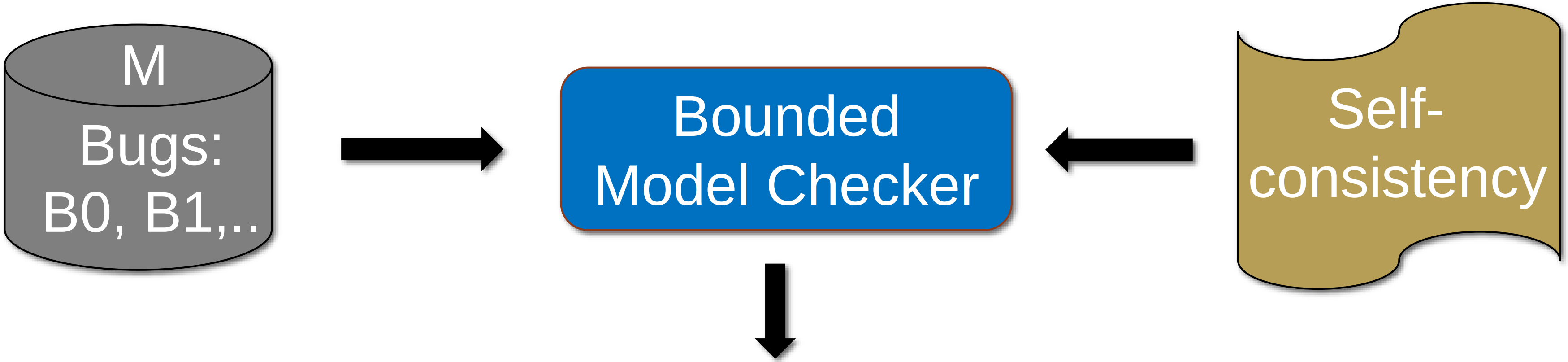


Does P ∈ *Spec* hold in M? ≈ Is M self-consistent?

- If M not self-consistent then B ∈ M.
- Self-consistency covers bug B.

No spurious cex

Summary: SQED Soundness and Completeness



Does P ∈ *Spec* hold in M? ≈ Is M self-consistent?

- If B ∈ M then M not self-consistent.
- Self-consistency covers bug B.

Conditional/full
Completeness

Future Work

Leveraging QED test extensions:

- Soft/hard reset not yet applied in practice.
- Design-for-verification approach.

Formal model refinements:

- Instruction pipelines, multiprocessor systems.
- Deadlock detection.
- Symbolic starting states.

Thank you for your attention!