

A Theoretical Framework for Symbolic Quick Error Detection

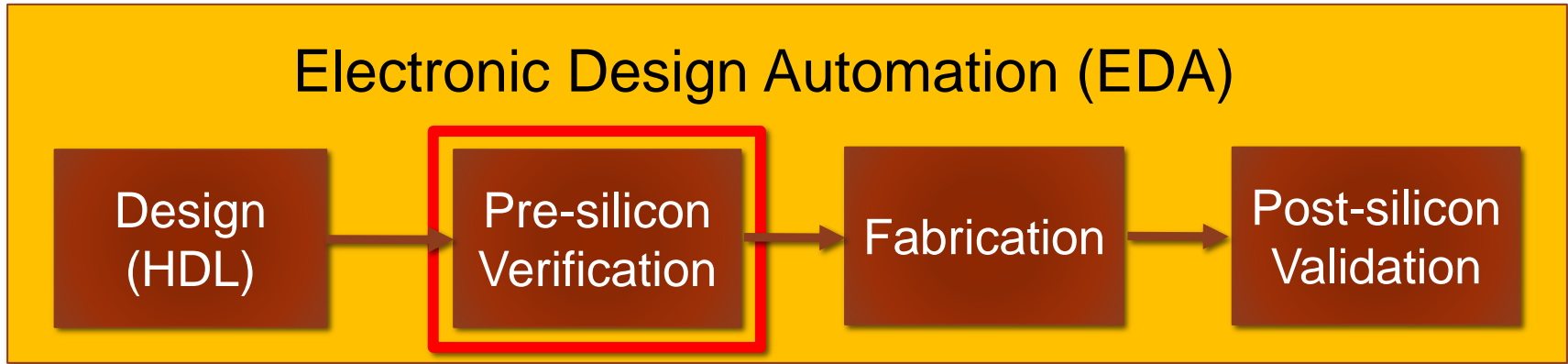


FLORIAN LONSING
SUBHASISH MITRA
CLARK BARRETT

Paper published at Formal Methods in Computer-Aided Design (FMCAD) 2020

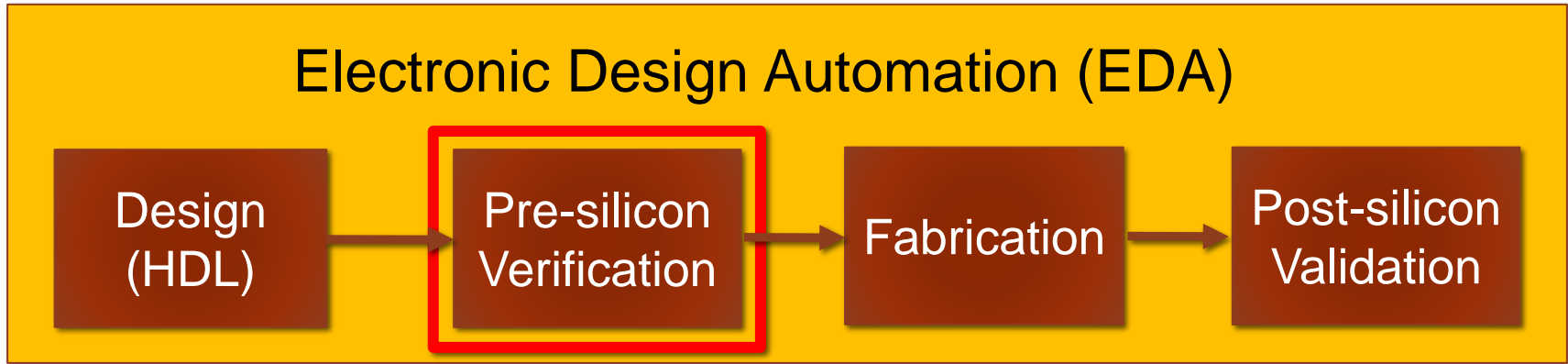
Preprint: <https://arxiv.org/abs/2006.05449>

Context: Pre-Silicon Verification



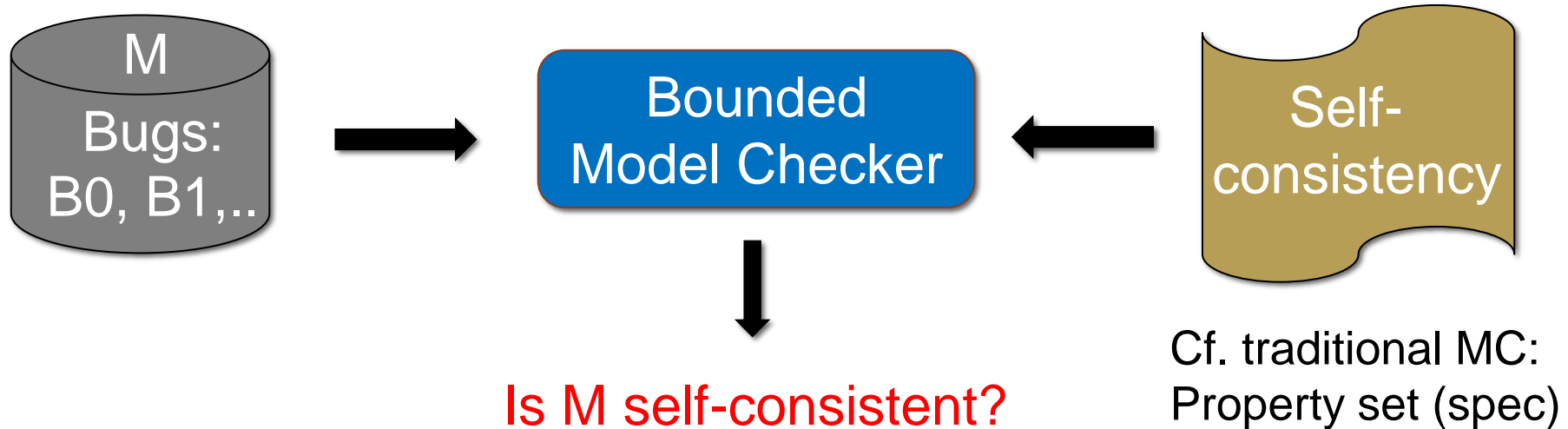
- Our focus: processor designs.
- Formally verify model of a design (e.g. Verilog).
- Model checking vs. non-formal simulation or testing.

Context: Pre-Silicon Verification



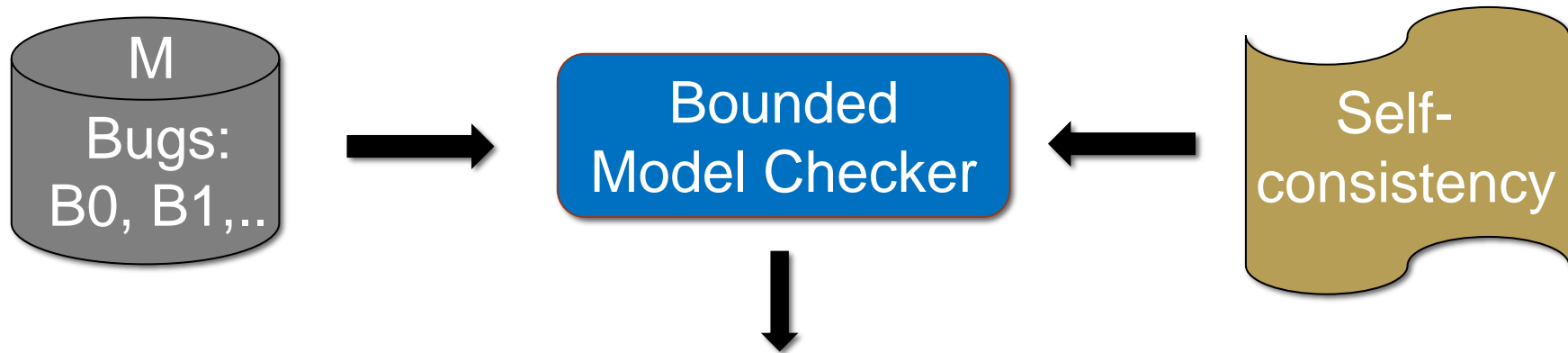
- Our focus: processor designs.
- **Formally** verify model of a design (e.g. Verilog).
- **Model checking** vs. non-formal simulation or testing.

Symbolic Quick Error Detection (SQED)



- **No** formal spec or implementation-specific properties.
- **Self-consistency**: universal property.
- Industrial-strength technique.

Our Contributions: SQED Soundness Proof

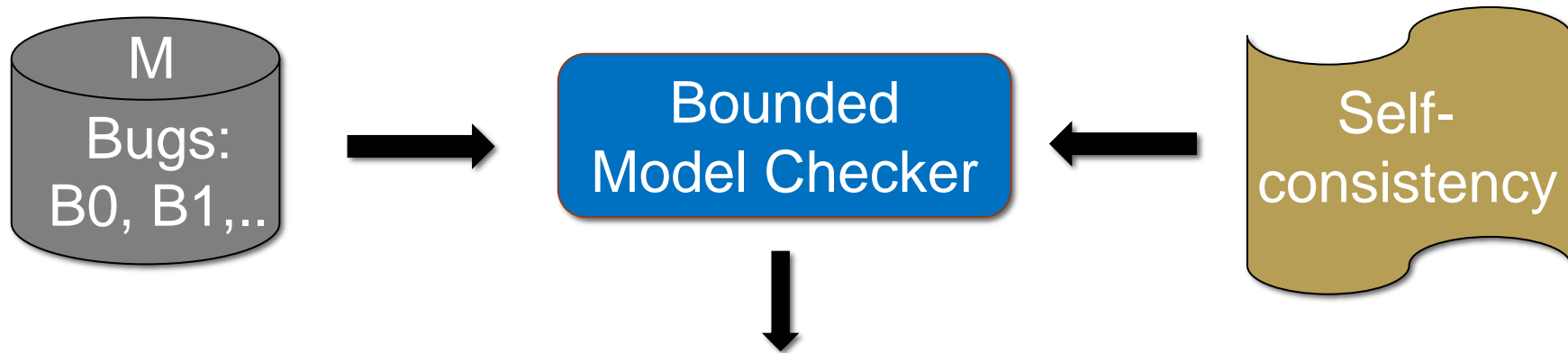


Is M self-consistent?

- If M not self-consistent then $B \in M$.

No spurious cex

Our Contributions: SQED Completeness Proof



Is M self-consistent?

- If $B \in M$ then M not self-consistent.

Conditional/full
Completeness

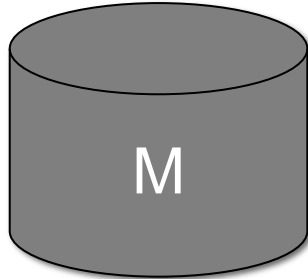
Self-Consistency

- 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)

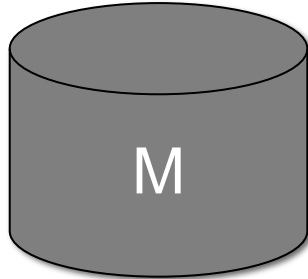
Self-Consistency

- 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)

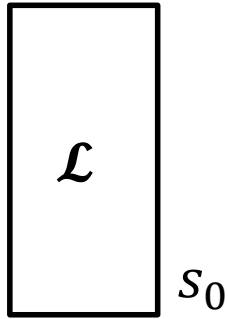
=

outputs' (regs/mem)



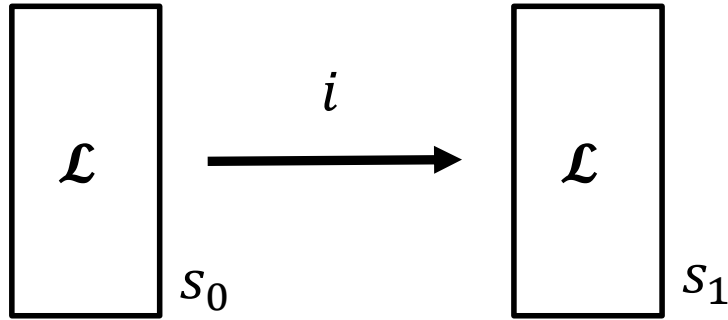
HW designs have complex internal state (pipeline,...).

Formal Model of Processors and SQED



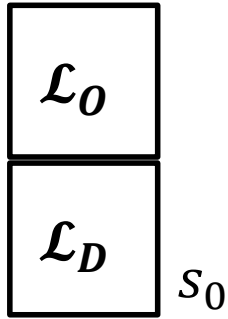
- State s_0 : mapping from locations \mathcal{L} to values.
- \mathcal{L} : register and memory locations.

Formal Model of Processors and SQED



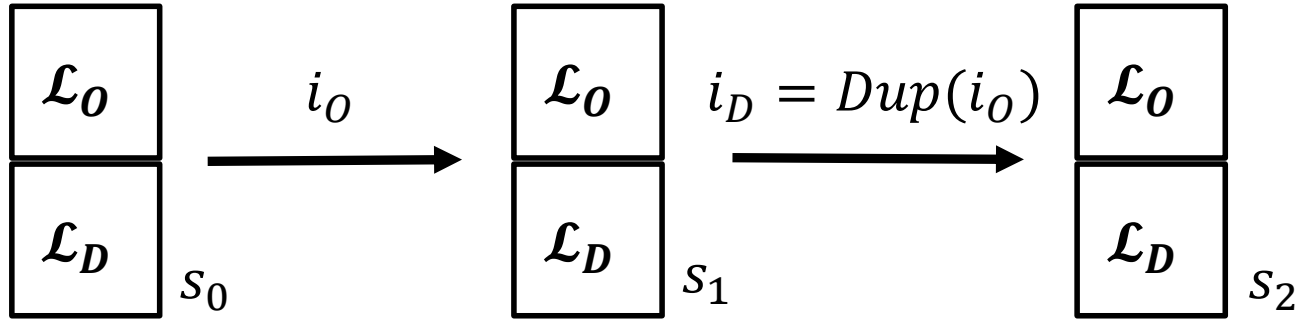
- Executing instruction i : read from input locations in s_0 .
- Update output location in s_1 by opcode.

Formal Model of Processors and SQED



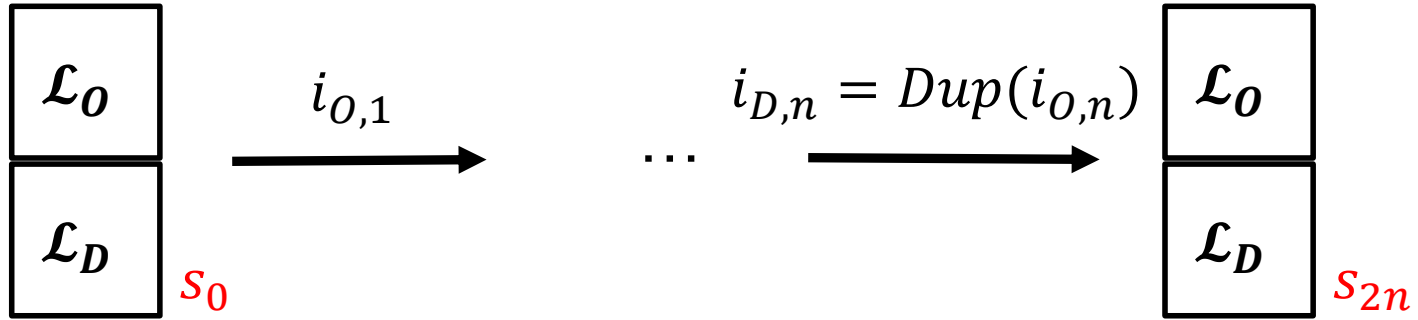
- Original locations \mathcal{L}_O .
- Duplicate locations \mathcal{L}_D .
- Arbitrary, fixed bijective mapping $L_D : \mathcal{L}_O \rightarrow \mathcal{L}_D$.

Formal Model of Processors and SQED



- Original instruction i_0 : read/write \mathcal{L}_0 only.
- Duplicate i_D : same opcode, read/write \mathcal{L}_D only.

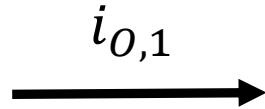
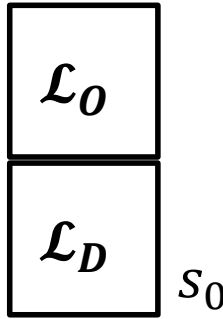
Formal Model of Processors and SQED



- $\mathbf{i}_O = i_{O,1}, \dots, i_{O,n}$, $\mathbf{i}_D = i_{D,1}, \dots, i_{D,n}$, $\mathbf{i}_D = Dup(\mathbf{i}_O)$.
- **QED test:** concatenation $\mathbf{i} = \mathbf{i}_O :: \mathbf{i}_D$ of $2n$ instructions.

Formal Model of Processors and SQED

$QEDcons(s_0)$

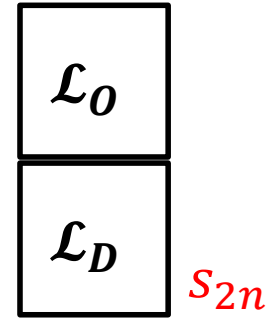


...

$i_{D,n} = Dup(i_{O,n})$



$QEDcons(s_{2n})?$



$$s_0(\mathcal{L}_O) = s_0(\mathcal{L}_D)$$

$$s_{2n}(\mathcal{L}_O) = s_{2n}(\mathcal{L}_D)?$$

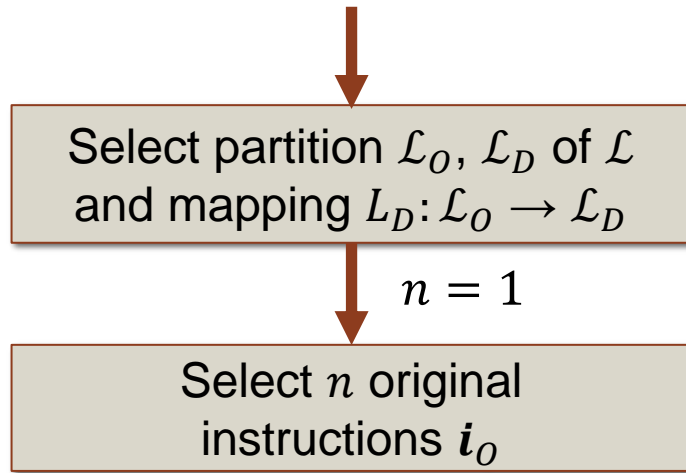
- $\mathbf{i}_O = i_{O,1}, \dots, i_{O,n}$, $\mathbf{i}_D = i_{D,1}, \dots, i_{D,n}$, $\mathbf{i}_D = Dup(\mathbf{i}_O)$.
- QED test: concatenation $\mathbf{i} = \mathbf{i}_O :: \mathbf{i}_D$ of $2n$ instructions.
- **QED-consistent state**: matching values at \mathcal{L}_O and \mathcal{L}_D .

Using BMC in SQED

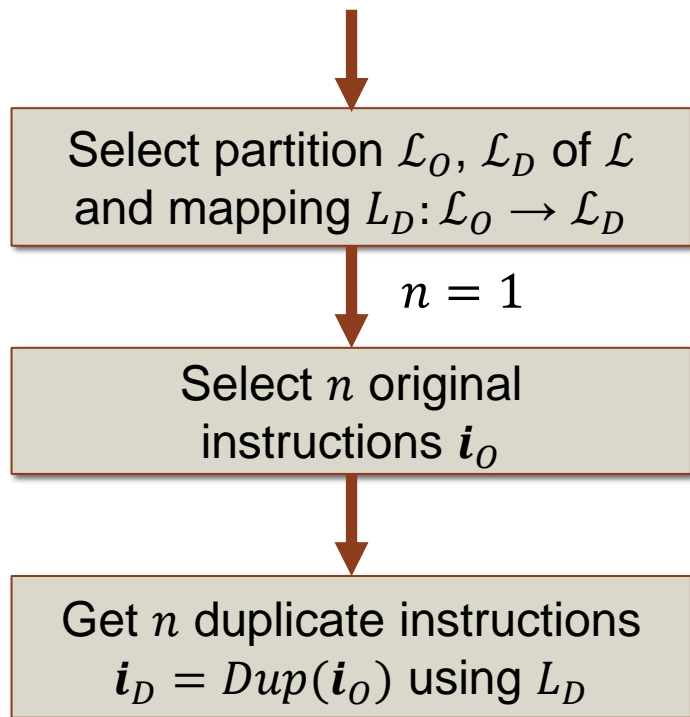


Select partition $\mathcal{L}_O, \mathcal{L}_D$ of \mathcal{L}
and mapping $L_D: \mathcal{L}_O \rightarrow \mathcal{L}_D$

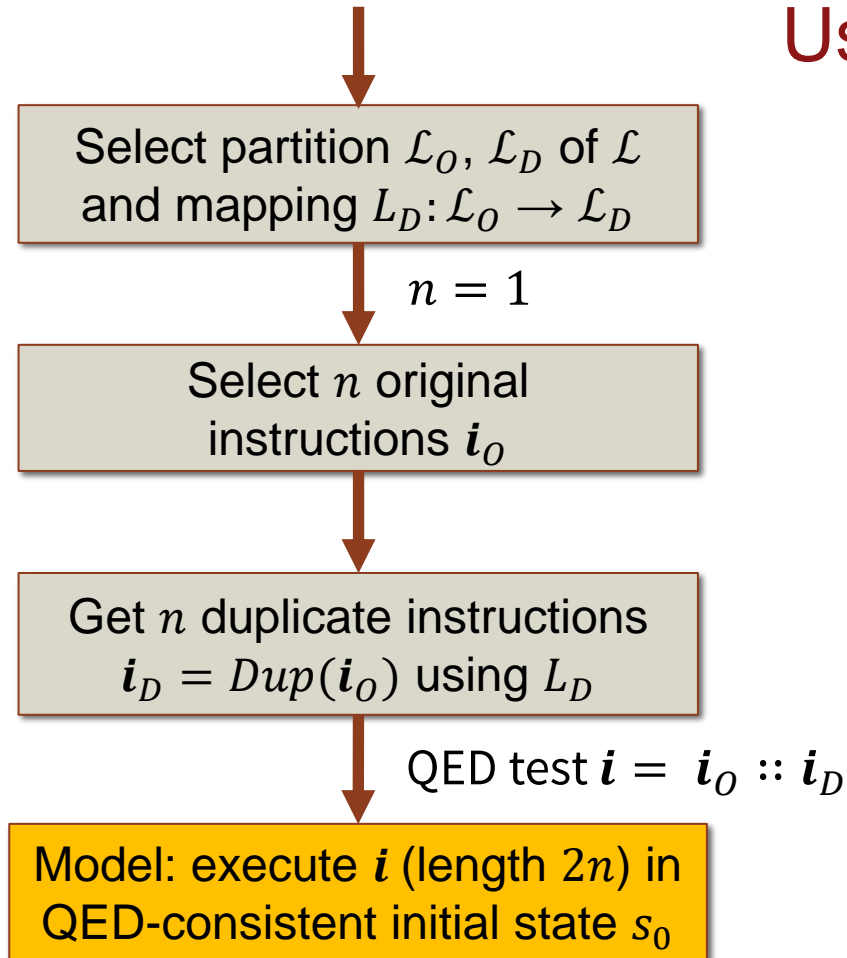
Using BMC in SQED



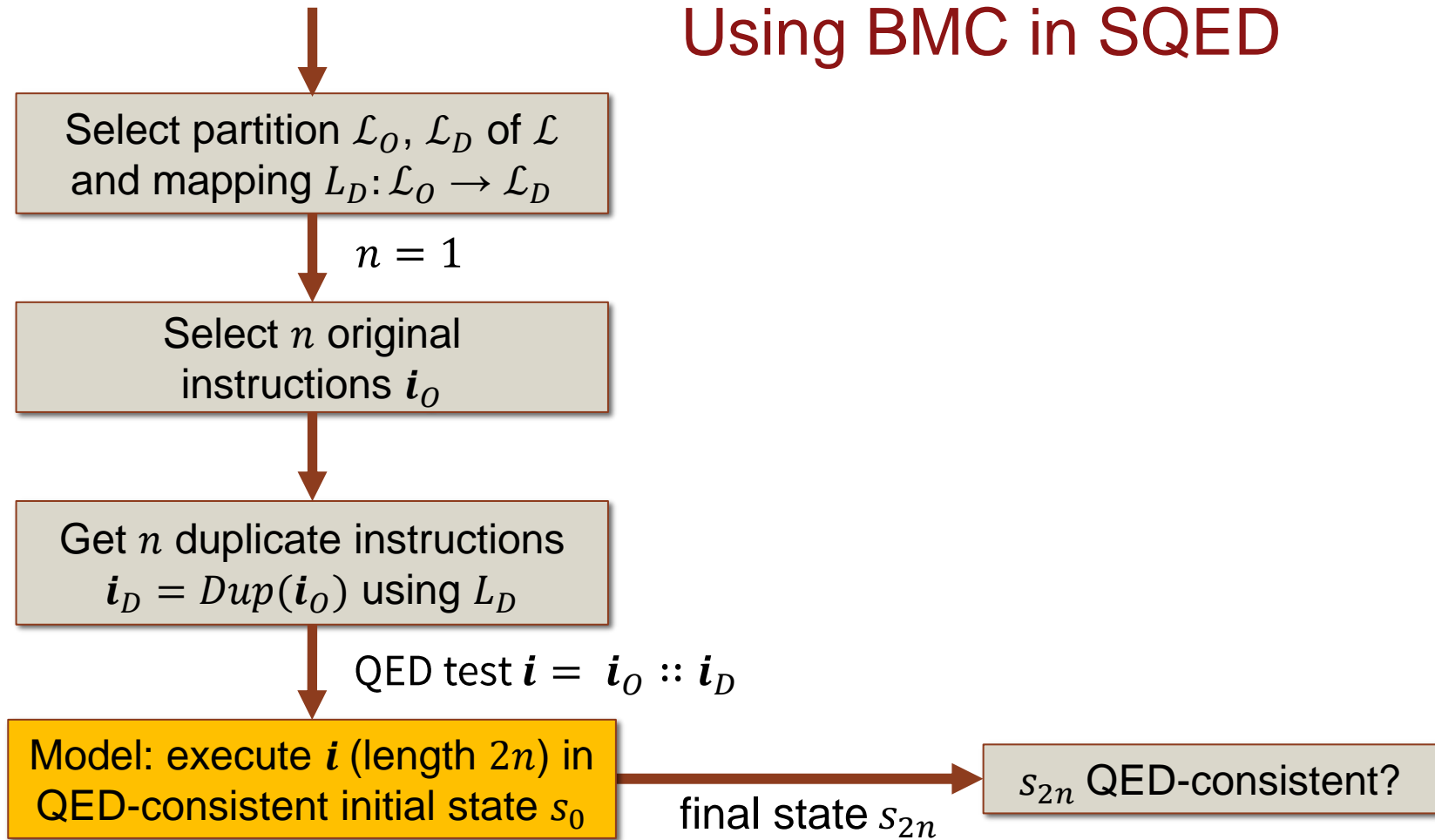
Using BMC in SQED



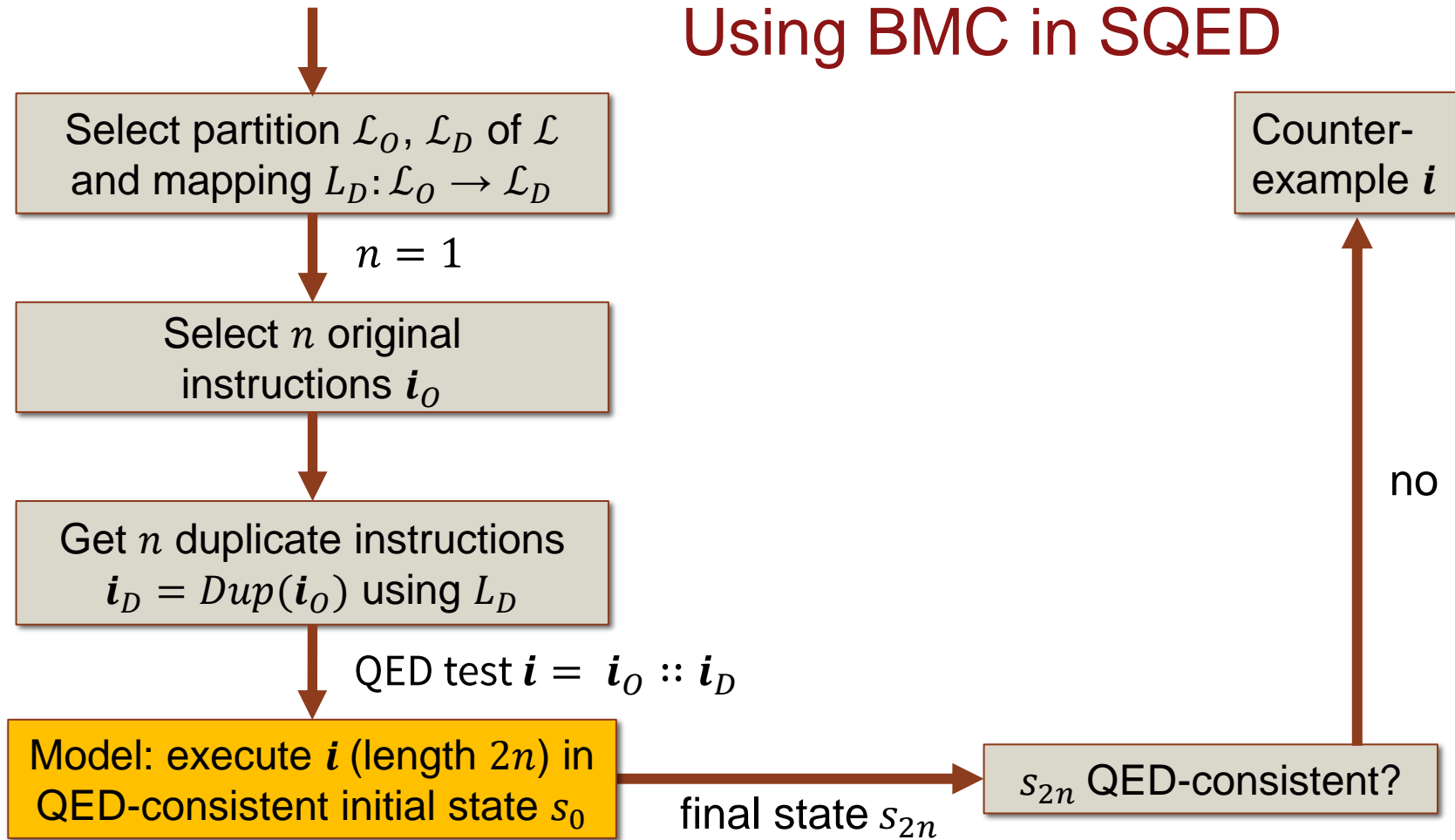
Using BMC in SQED



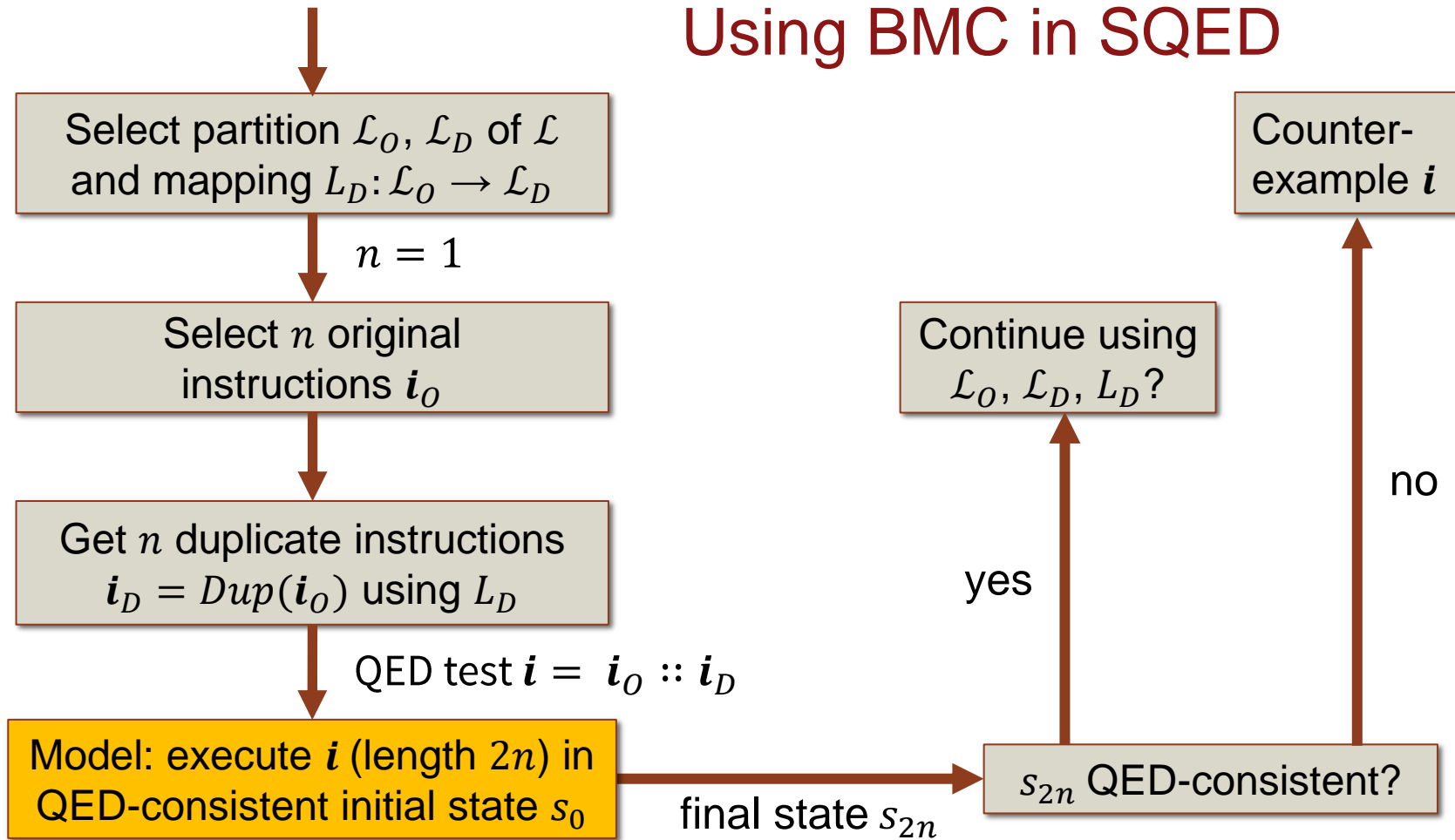
Using BMC in SQED



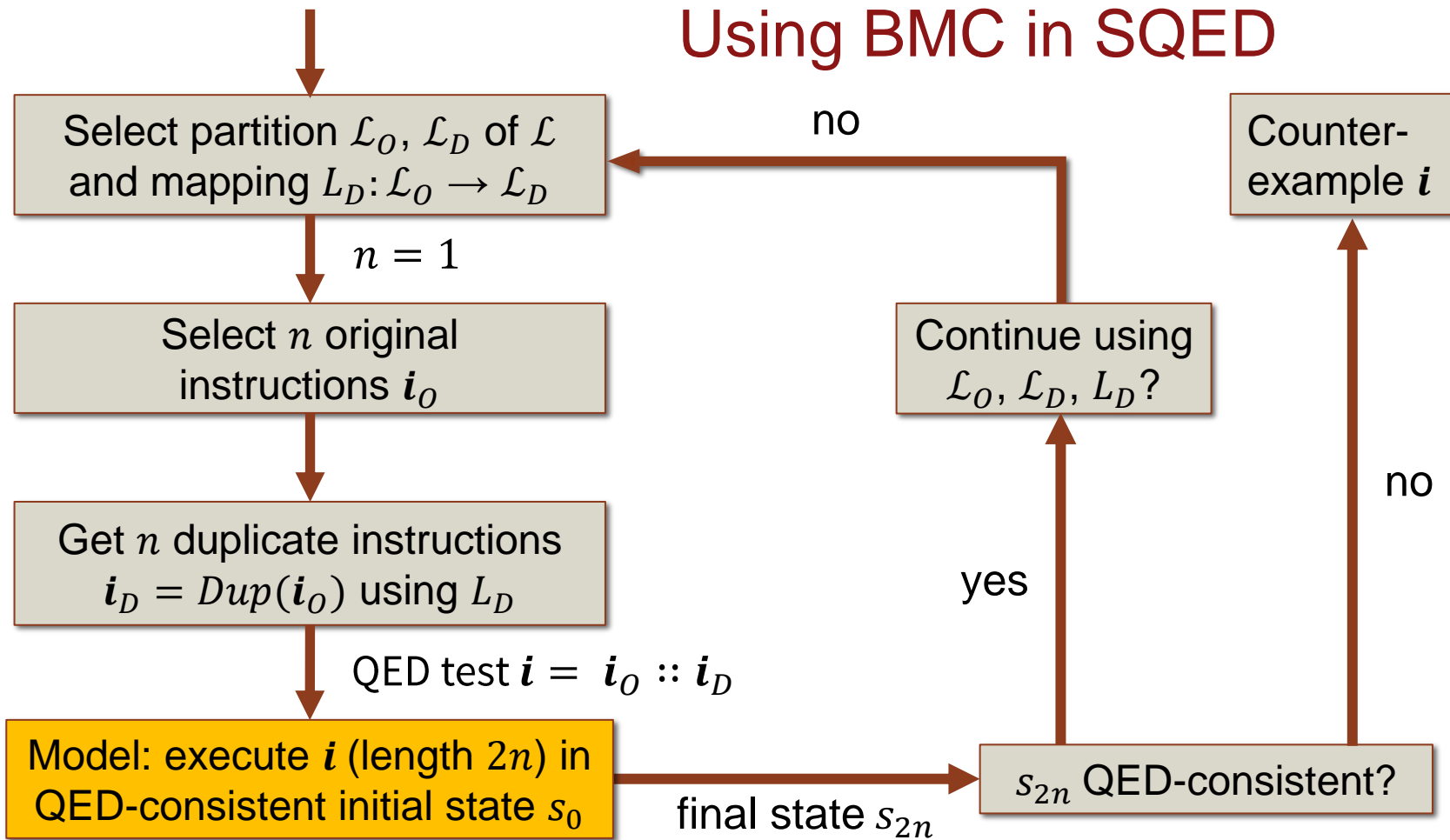
Using BMC in SQED



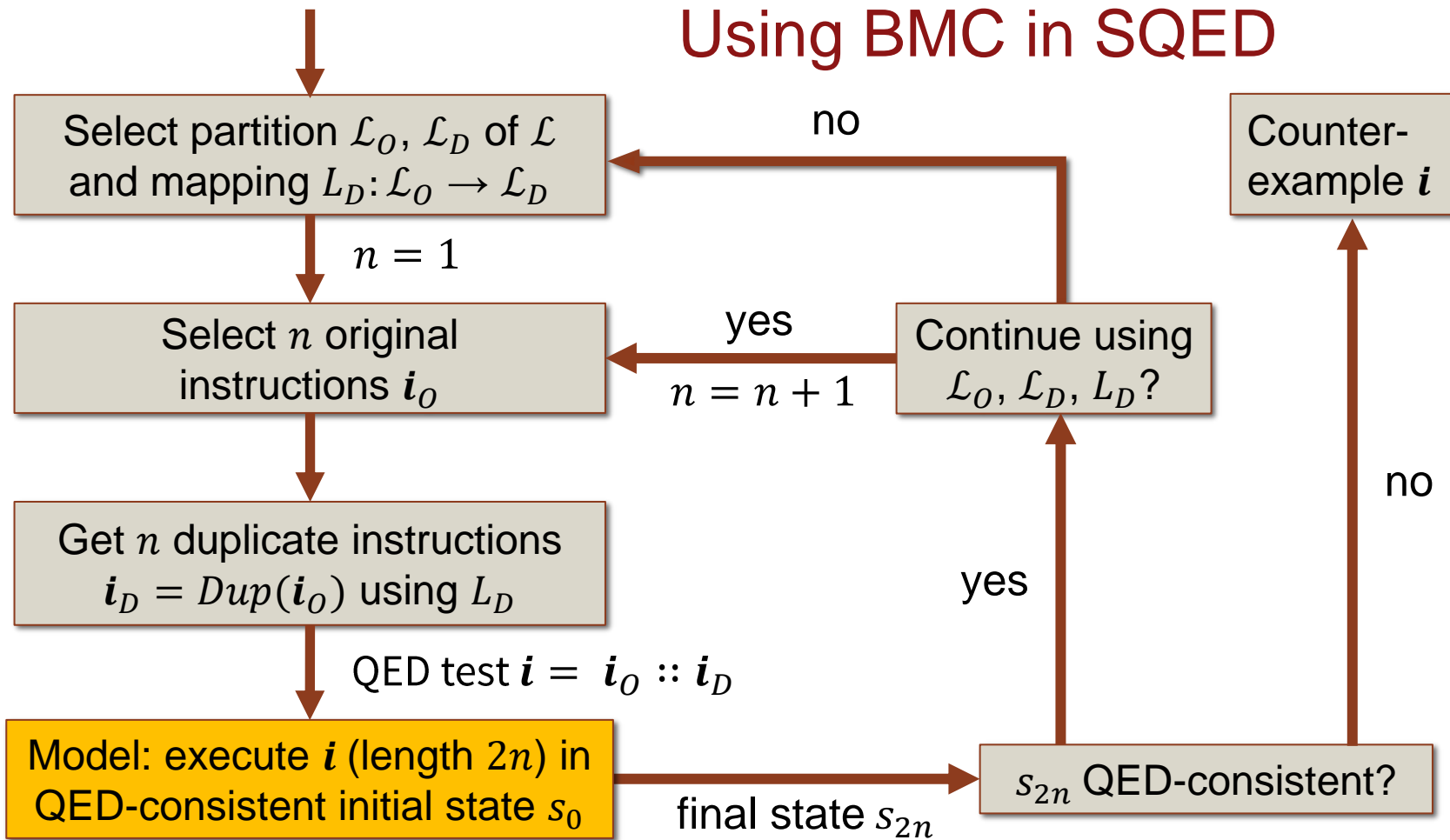
Using BMC in SQED



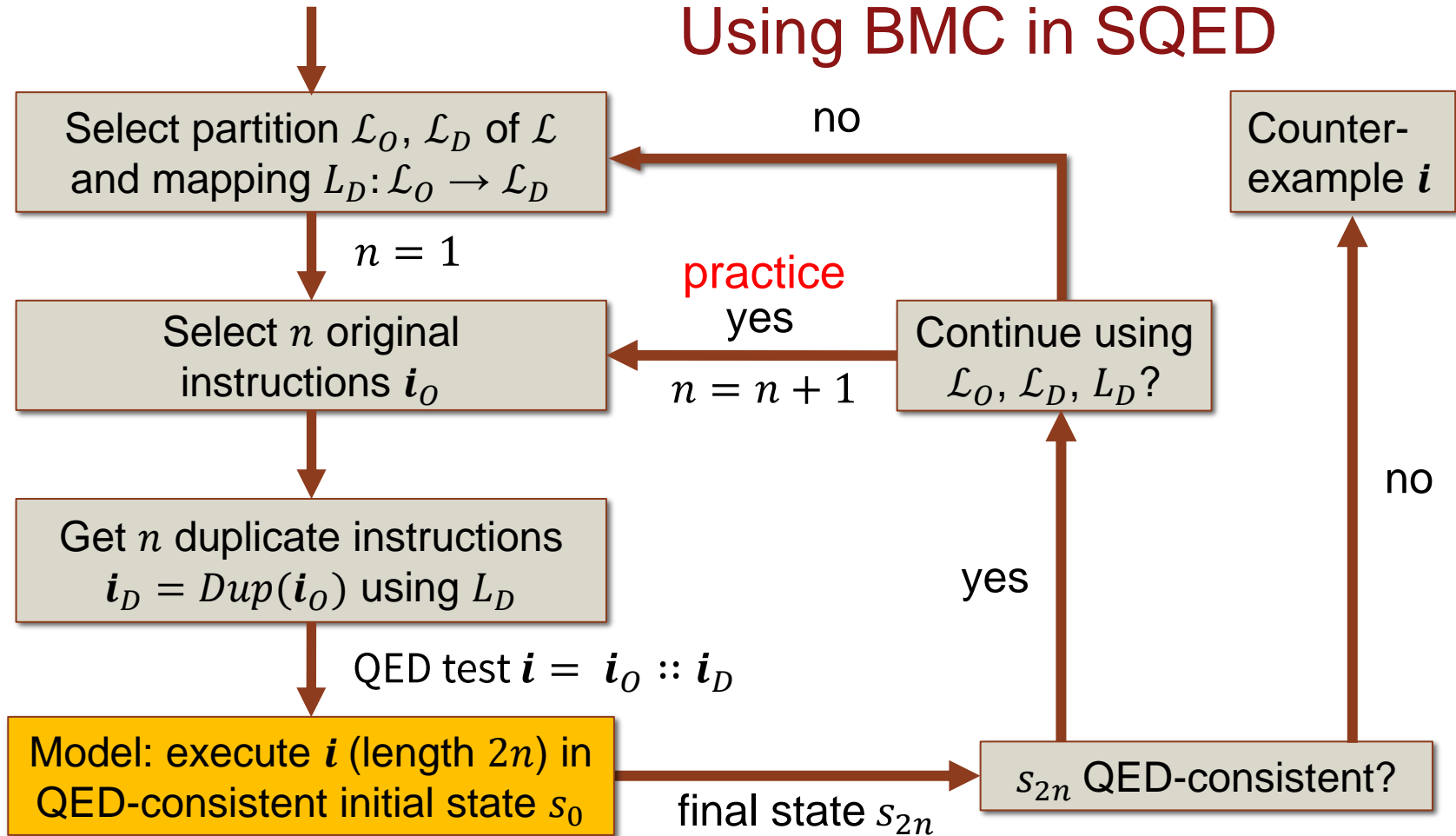
Using BMC in SQED



Using BMC in SQED



Using BMC in SQED



Processor Correctness

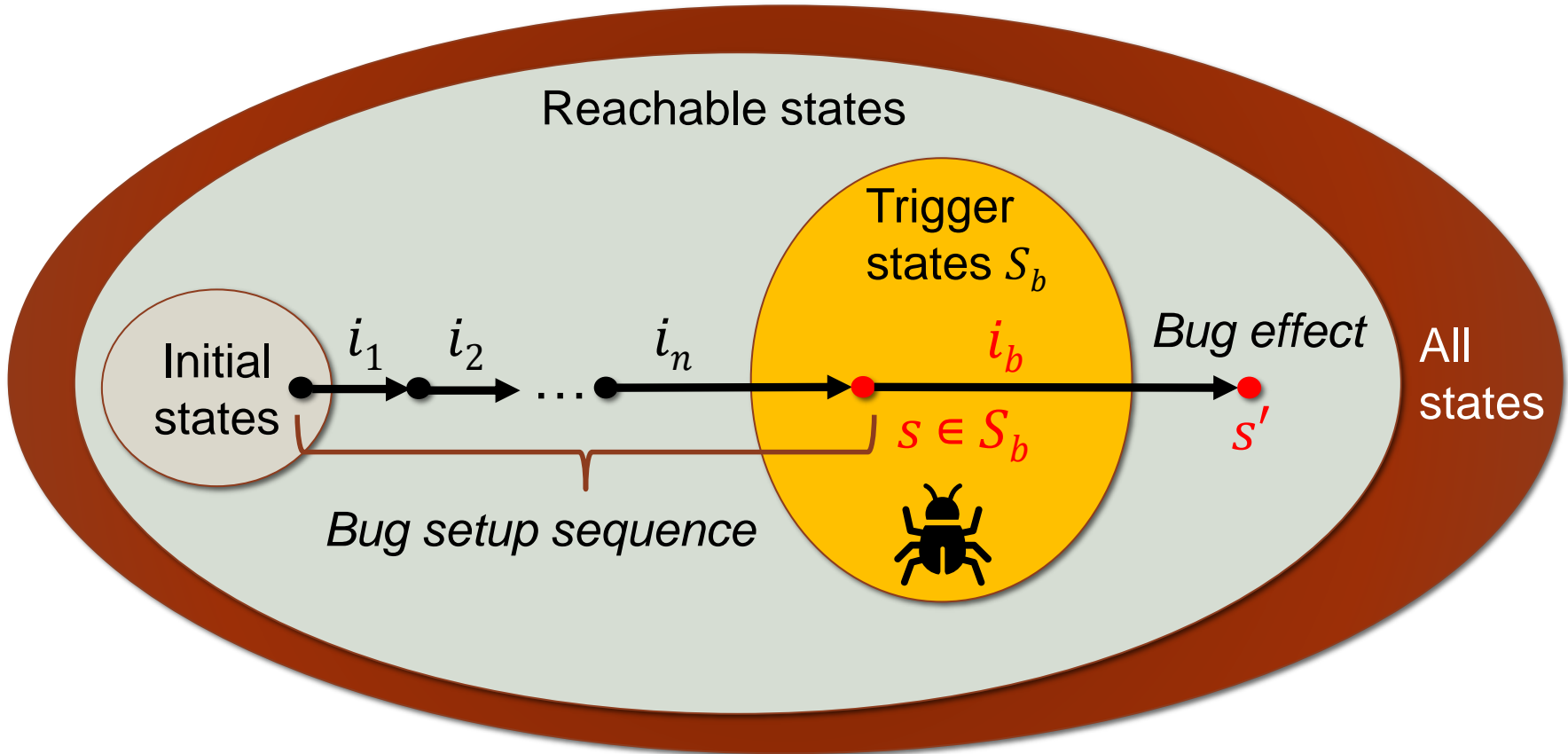
Processor P is **correct**:

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

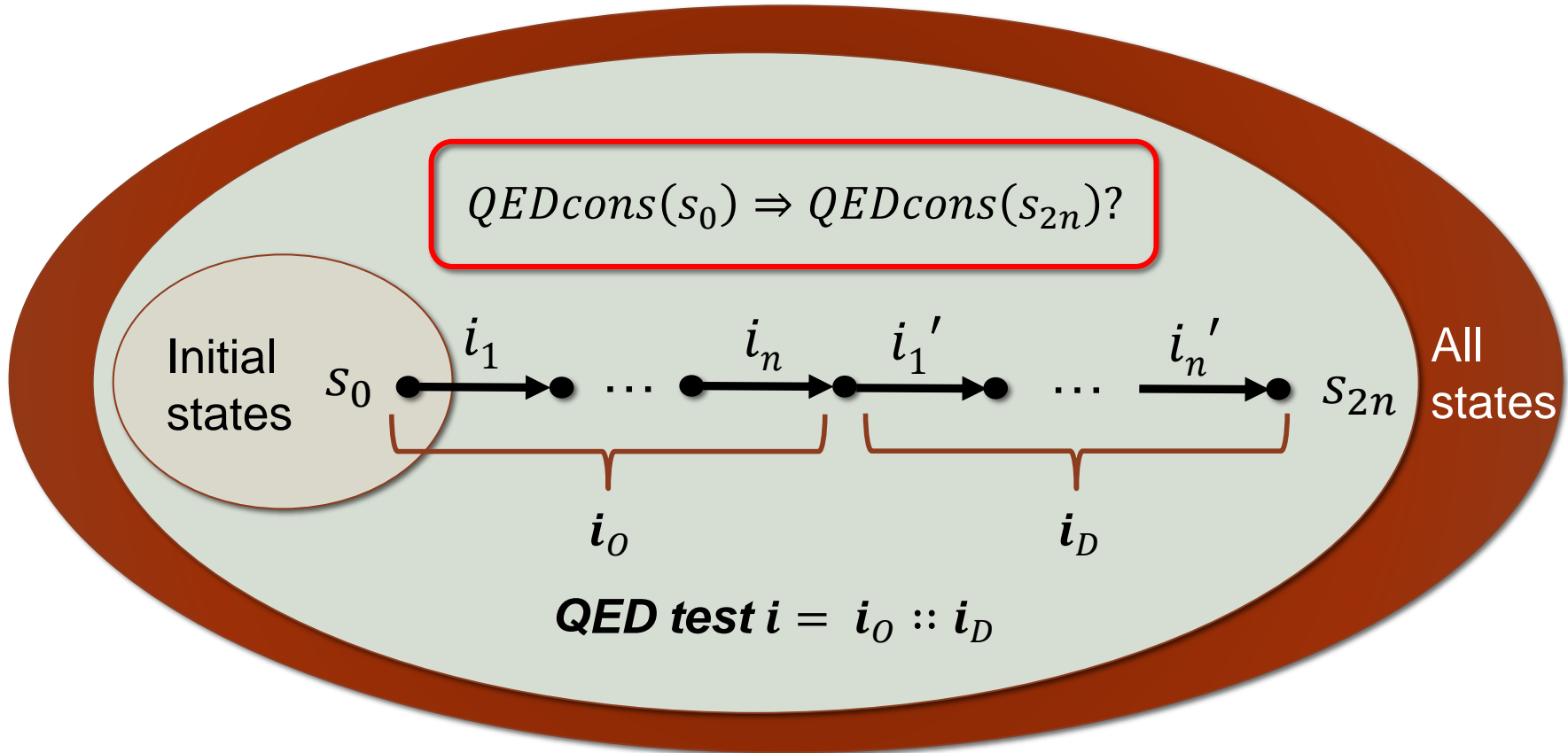
Abstract specification *Spec* (theory only):

1. Correct output value product.
2. All non-output locations unchanged.

Bugs



Soundness of SQED



Soundness of SQED

Theorem: if $\sim QEDcons(s_{2n})$ then processor P has a bug.

Initial states

s_0

i_1

i_n

i_1'

i_n'

s_{2n}

All states

i_0

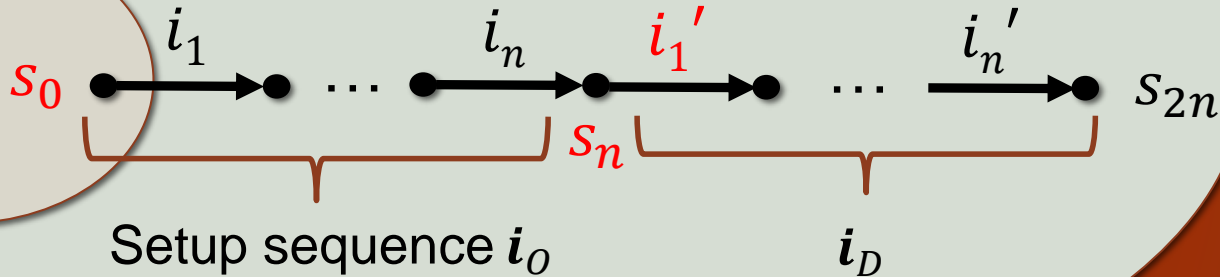
i_D

QED test $i = i_0 :: i_D$

Conditional Completeness of SQED

- Set up and trigger bug.
- $s_n \in S_b, i_1' = \text{Dup}(i_1) = i_b$.
- Freedom in choosing L_D .

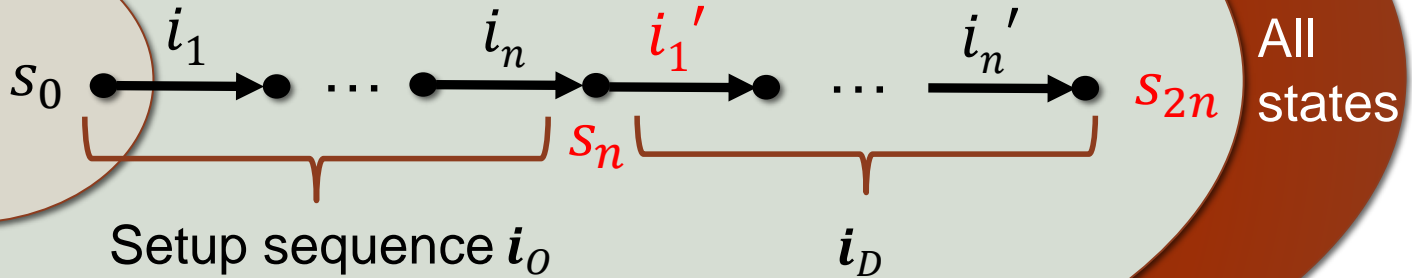
Initial
states



Conditional Completeness of SQED

- $i_1' = Dup(i_1) = i_b$ fails in s_n .
- Goal: bug effect appears in s_{2n} .

Initial
states

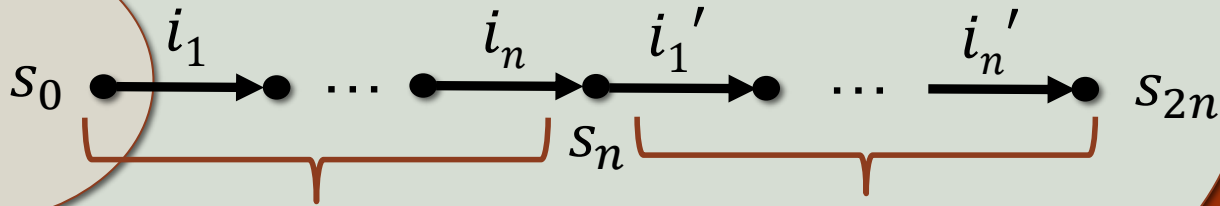


Bug-specific QED test $i = i_0 :: i_D$

Conditional Completeness of SQED

Theorem: if *bug-specific QED test i* exists then $\sim QEDcons(s_{2n})$.

Initial states



Bug-specific QED test $i = i_0 :: i_D$

Full Completeness of SQED

- No duplication: run twice.
- Run reset instructions.
- Assume: *Spec* in initial states.

Initial
states

s_0

i_1

...

i_b

reset

s_0

1st run

All
states

Setup sequence

QED test i with reset

Full Completeness of SQED

- No duplication: run twice.
- Run reset instructions.
- Assume: *Spec* in initial states.

Initial states

s_0

i_1

reset

s_0'

i_b

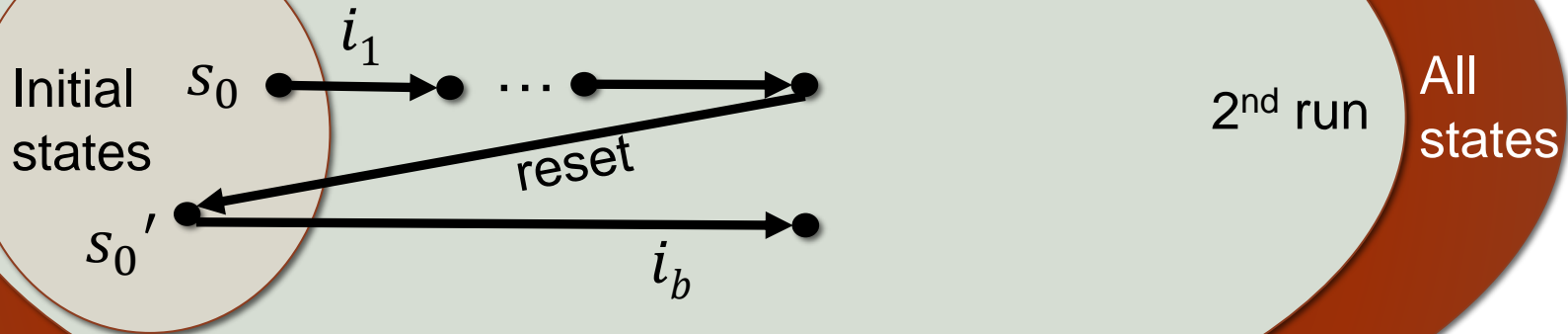
2nd run

All states

QED test i with reset

Full Completeness of SQED

Theorem: *if P has no failing QED test with reset, then P is correct.*



QED test i with reset

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!