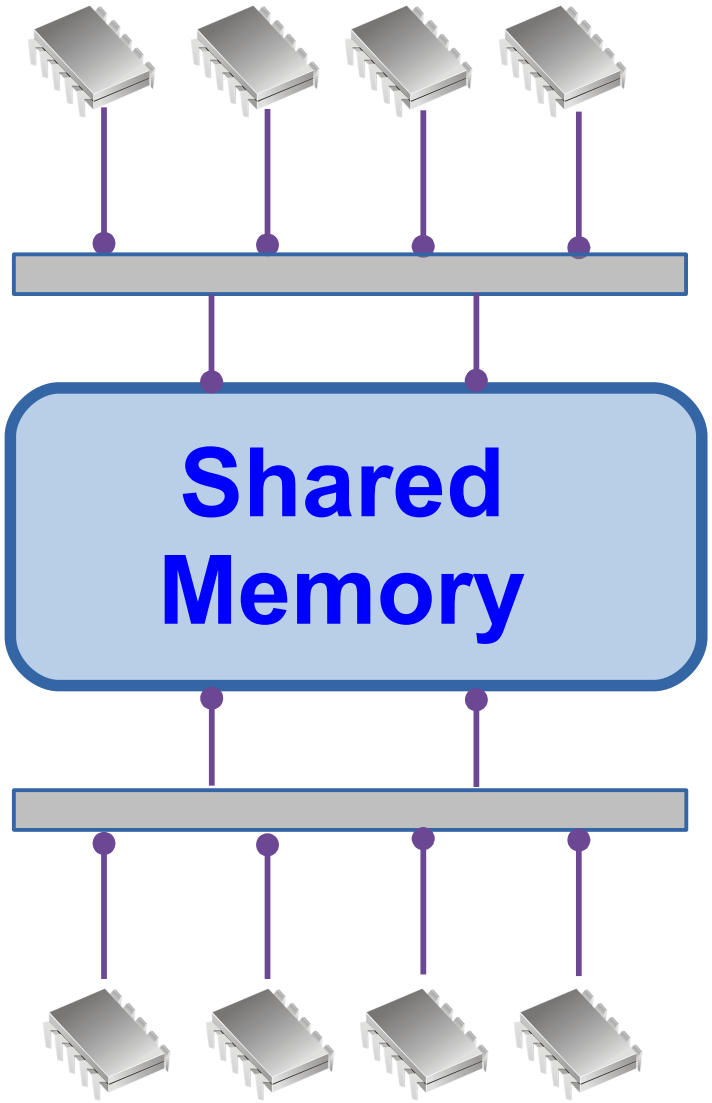
# AutoMO: Automatic Inference of Memory Order Parameters for C/C++11

**Peizhao Ou** and Brian Demsky

University of California, Irvine

Oct 28, 2015

# Programming with Multi-cores




*Building Blocks*

**Concurrent Data Structures with Atomics**

# C/C++11 Memory Model

```
int x, y;
// Thread 1                 // Thread 2
x = 1;                      r1 = y;
y = 1;                      r2 = x;
```

**Pseudo code**

```
atomic_int x, y;
// Thread 1                 // Thread 2
x.store(1, relaxed);        r1 = y.load(acquire);
y.store(1, release);        r2 = x.load(relaxed);
```

**C/C++11**

**memory order parameters**

# C/C++11 Memory Model

- Language-level atomics → portability

- Memory order parameters → complicated

Pseudo code

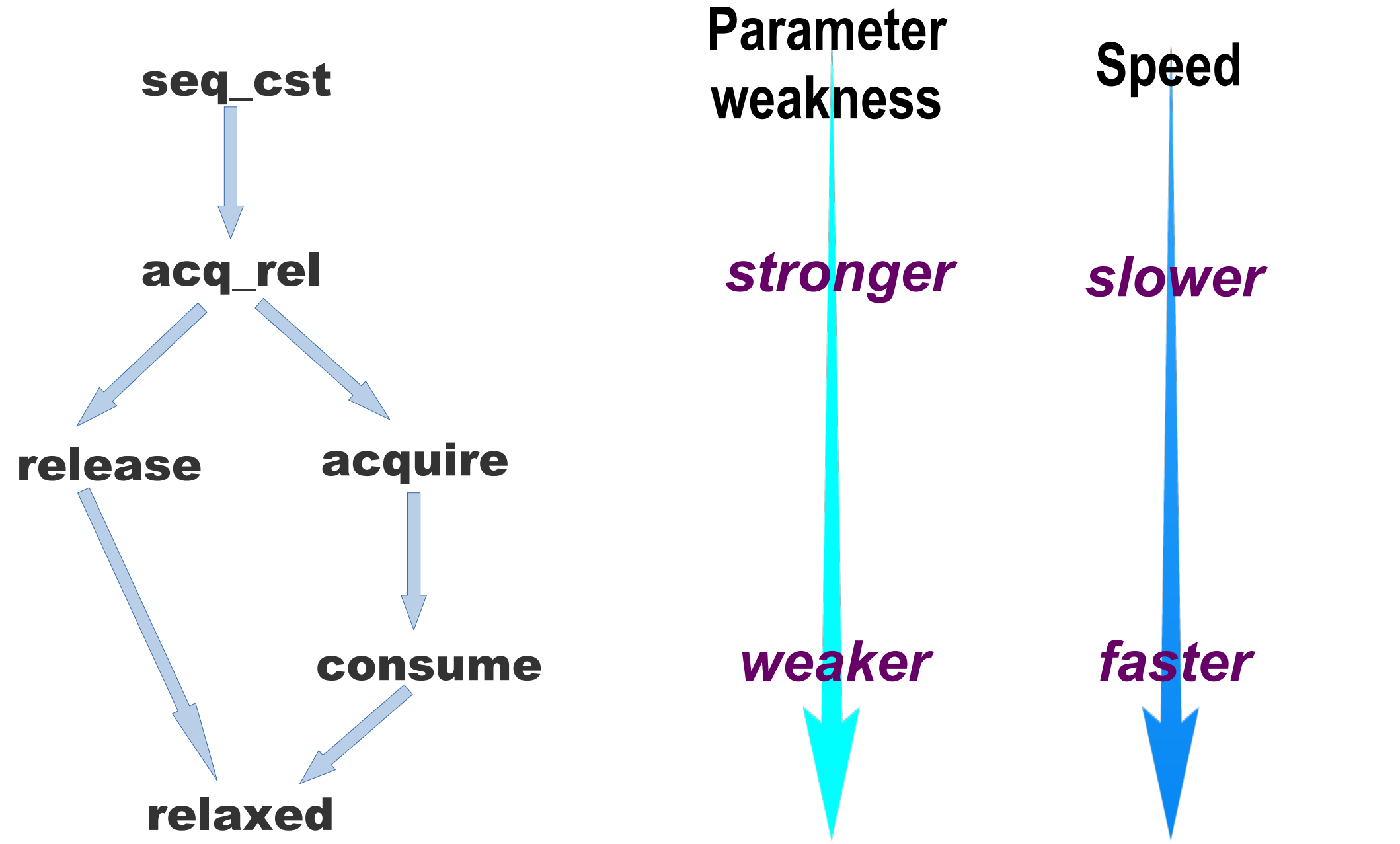
```
int x, y;
// Thread 1
x = 1;
y = 1;
// Thread 2
r1 = y;
r2 = x;
```

C/C++11

```
atomic_int x, y;
// Thread 1
x.store(1,          );
y.store(1,          );
// Thread 2
r1 = y.load(        );
r2 = x.load(        );
```

memory order parameters

# Memory Order Parameters



# Choosing Memory Order Parameters

```
atomic_int x, y;
// Thread 1                          // Thread 2
x.store(1, ____?____);               r1 = y.load(____?____);
y.store(1, ____?____);               r2 = x.load(____?____);
```

# Choosing Memory Order Parameters

```
atomic_int x, y;
// Thread 1                    // Thread 2
x.store(1, seq_cst);           r1 = y.load(seq_cst);
y.store(1, seq_cst);           r2 = x.load(seq_cst);
```

Overly strong parameters
→ **hurt performance**

# Choosing Memory Order Parameters

```
atomic_int x, y;
// Thread 1                    // Thread 2
x.store(1, relaxed);           r1 = y.load(relaxed);
y.store(1, relaxed);           r2 = x.load(relaxed);
```


Too weak parameters → ***bugs***

# Choosing Memory Order Parameters

```
atomic_int x, y;
// Thread 1                    // Thread 2
x.store(1, relaxed);           r1 = y.load(acquire);
y.store(1, release);           r2 = x.load(relaxed);
```

While **ensuring correctness**

→ seek for weaker parameters

**Difficult**

# What We Propose

**A tool that AUTOMATES** the process of configuring *memory order parameters*

# A Motivational Scenario


A fairly exhaustive test suite

Port

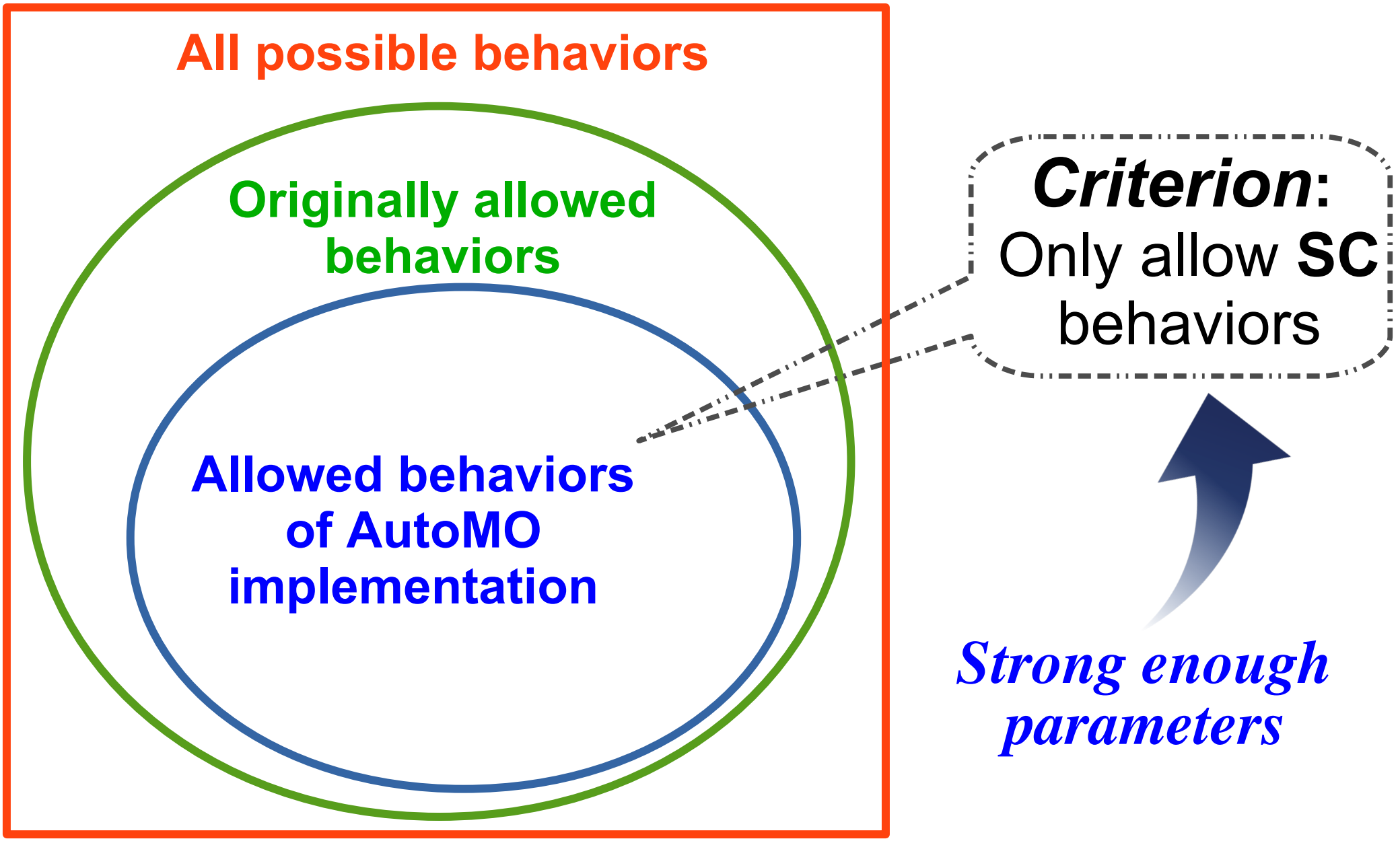
Some Memory Model

C/C++11 Memory Model


Same test suite has correct behaviors

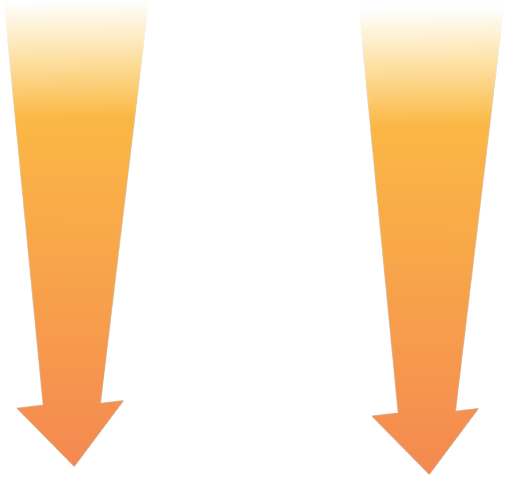
***What parameters??***

# Our Solution



# It Boils down to

Infer *memory order parameters*



*G*uarantee **SC** for provided test cases

# A Search Problem

```
atomic_int x, y;
// Thread 1                    // Thread 2
x.store(1, ____?____);         r1 = y.load(____?____);
y.store(1, ____?____);         r2 = x.load(____?____);
```

Fill out these blanks with parameters

# A Search Problem

```
atomic_int x, y;
// Thread 1                    // Thread 2
x.store(1, relaxed);           r1 = y.load(relaxed);
y.store(1, relaxed);           r2 = x.load(relaxed);
```

Start with the weakest parameters (relaxed)

# A Search Problem

```
atomic_int x, y;
// Thread 1                    // Thread 2
x.store(1, relaxed);           r1 = y.load(relaxed);
y.store(1, relaxed);           r2 = x.load(relaxed);
```

Such parameter assignment allows **non-SC** behaviors

# A Search Problem

```
atomic_int x, y;
// Thread 1                    // Thread 2
x.store(1, relaxed);           r1 = y.load(acquire);
y.store(1, release);           r2 = x.load(relaxed);
```

Try some stronger parameter assignment

# A Search Problem

```
atomic_int x, y;
// Thread 1                    // Thread 2
x.store(1, relaxed);           r1 = y.load(acquire);
y.store(1, release);           r2 = x.load(relaxed);
```

This parameter assignment **only** allows SC behaviors: **terminate**

# A Search Problem

```
atomic_int x, y;
// Thread 1                    // Thread 2
x.store(1, relaxed);           r1 = y.load(acquire);
y.store(1, release);           r2 = x.load(relaxed);
```

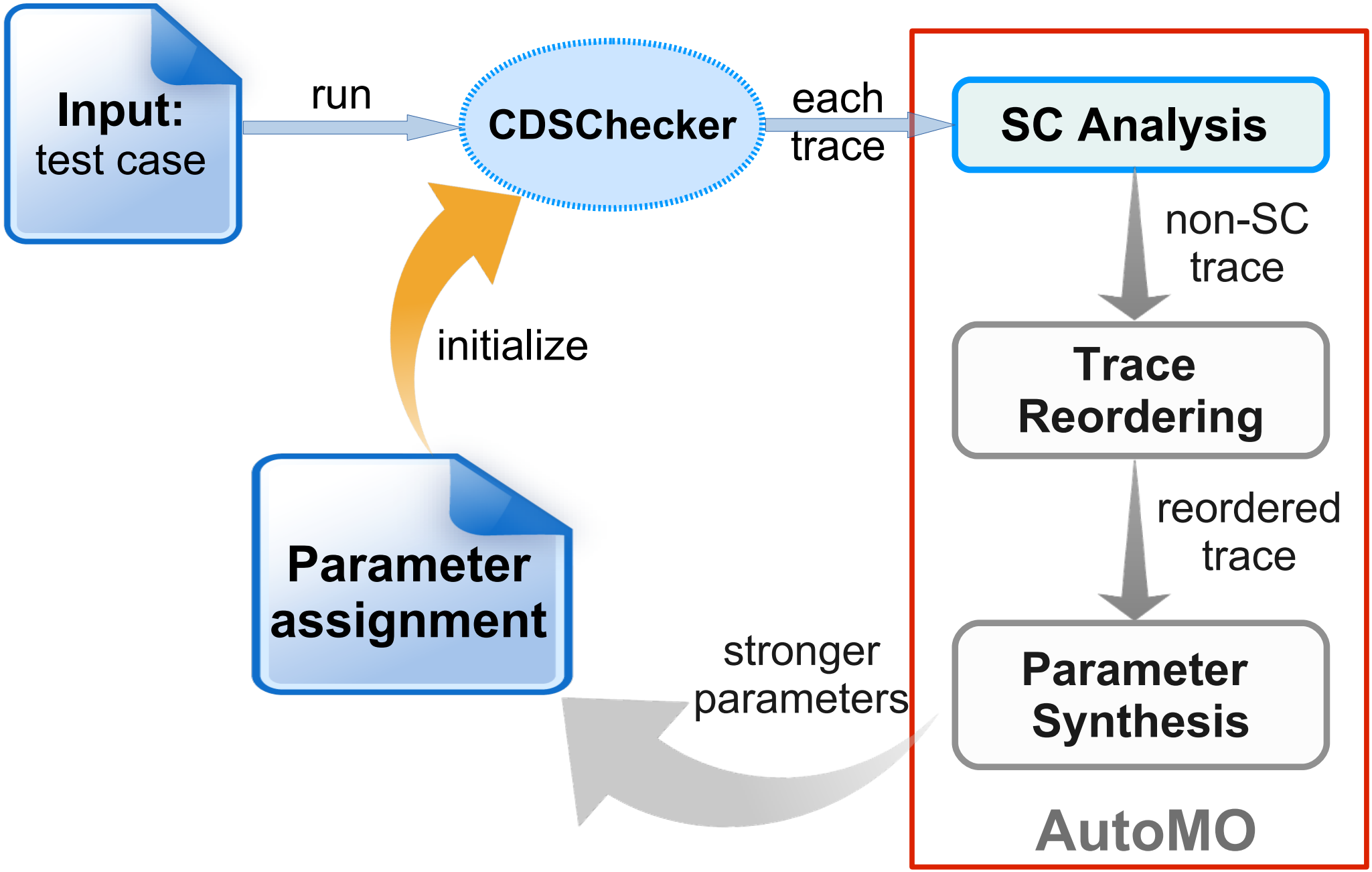
**Question 1:**

How to **detect** SC violations

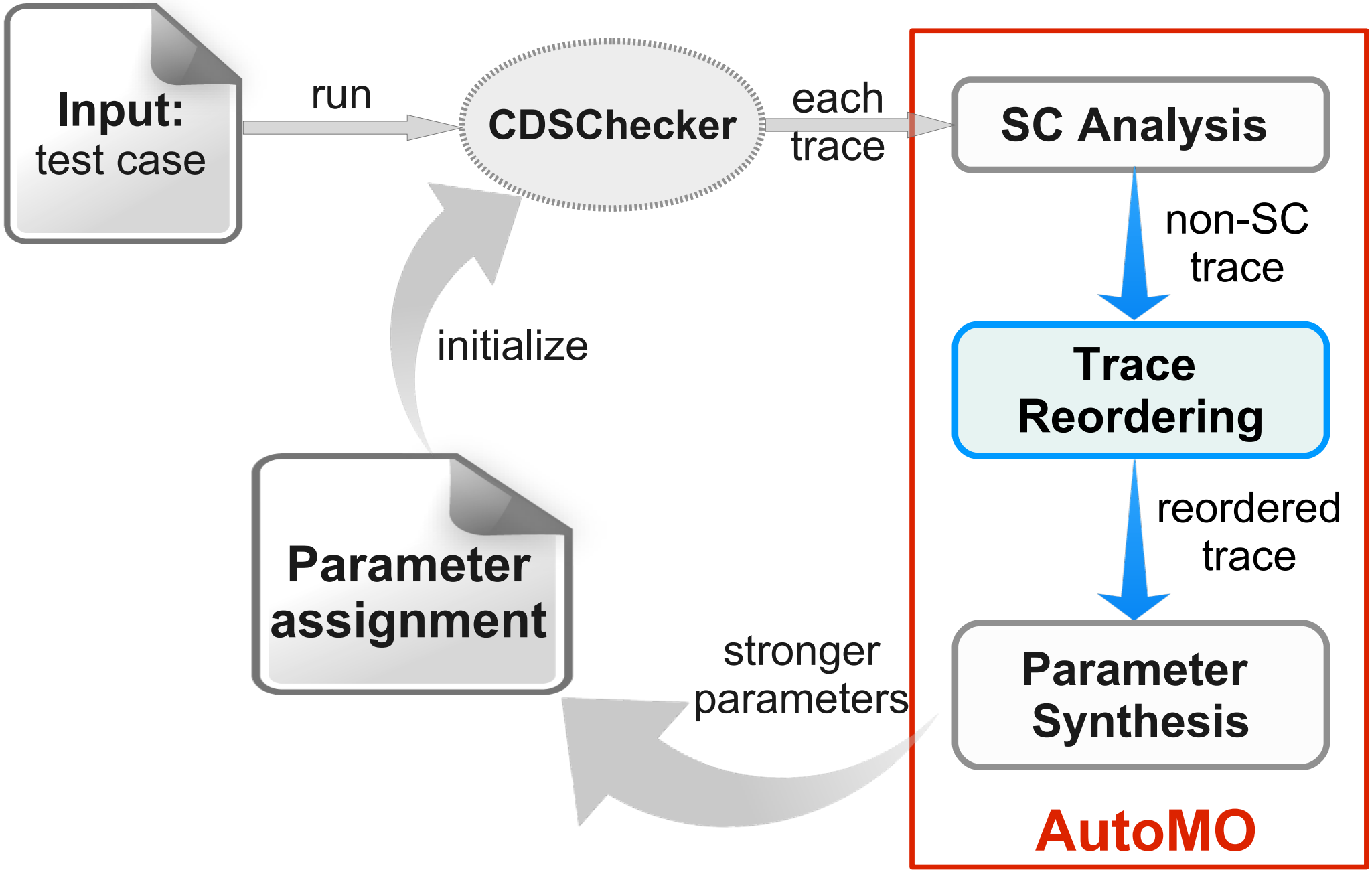
***Question 2:***

How to **repair** SC violations

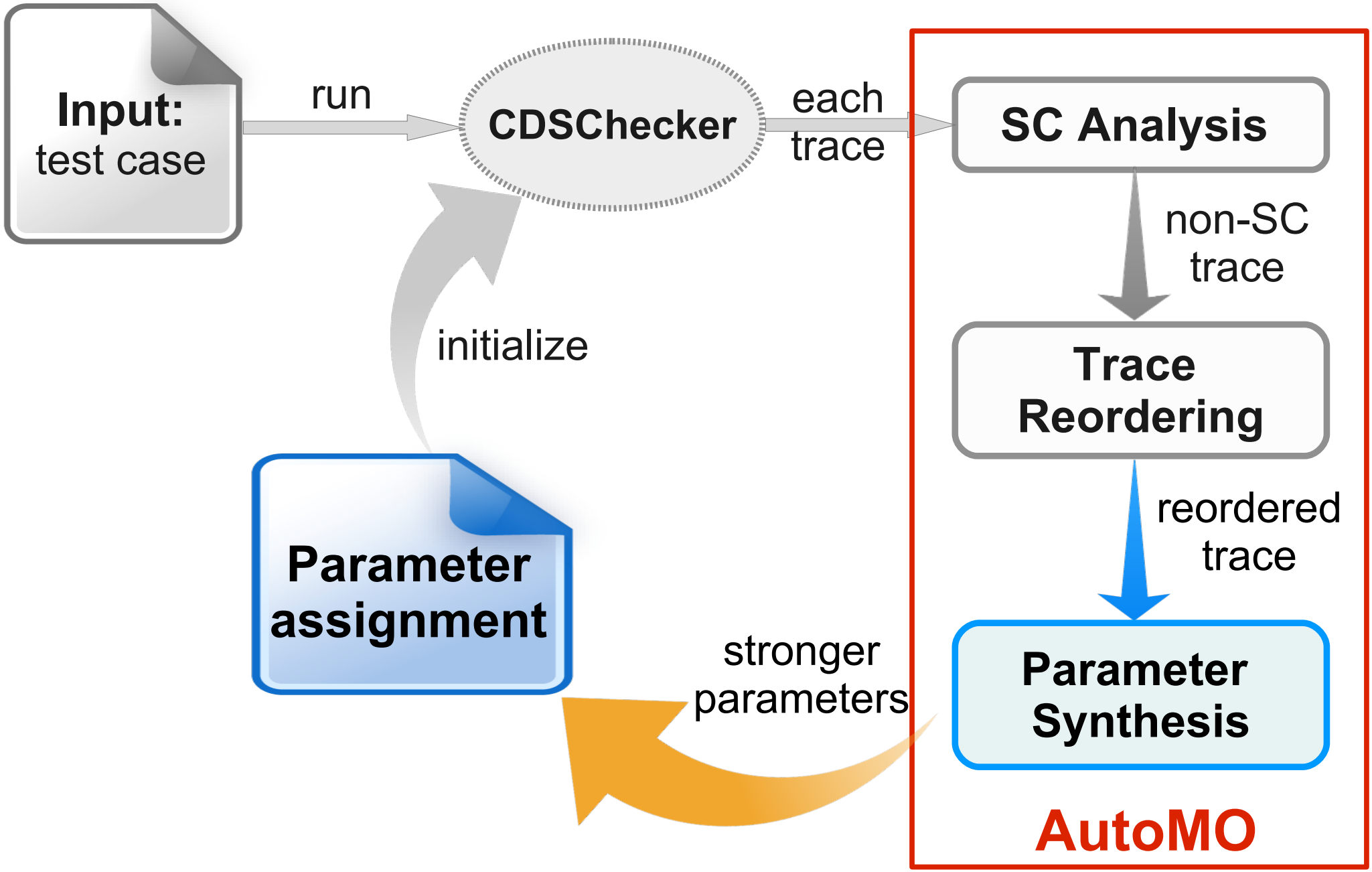
# Overview of Approach



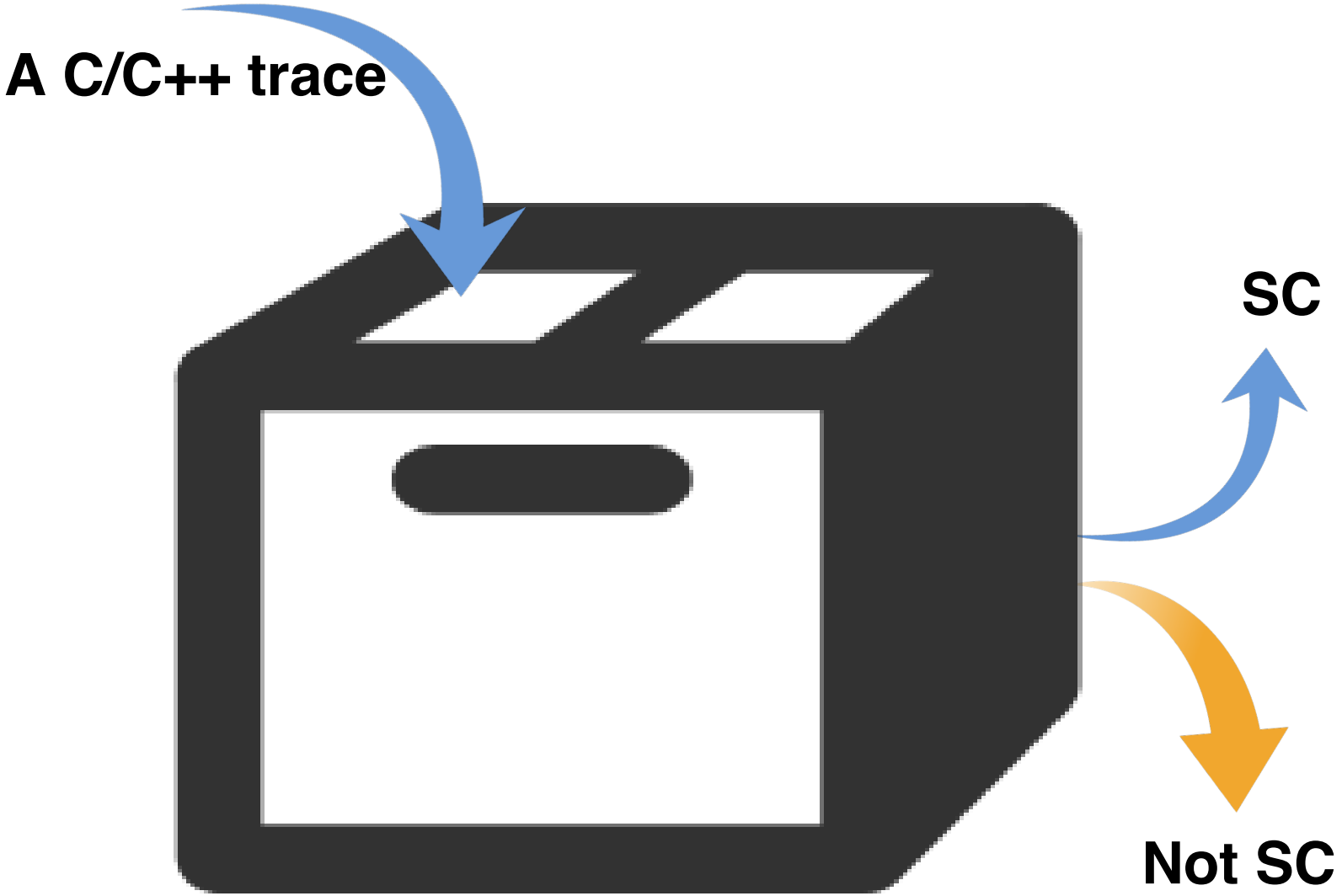
# Overview of Approach



# Overview of Approach



# SC Analysis



# Trace Reordering

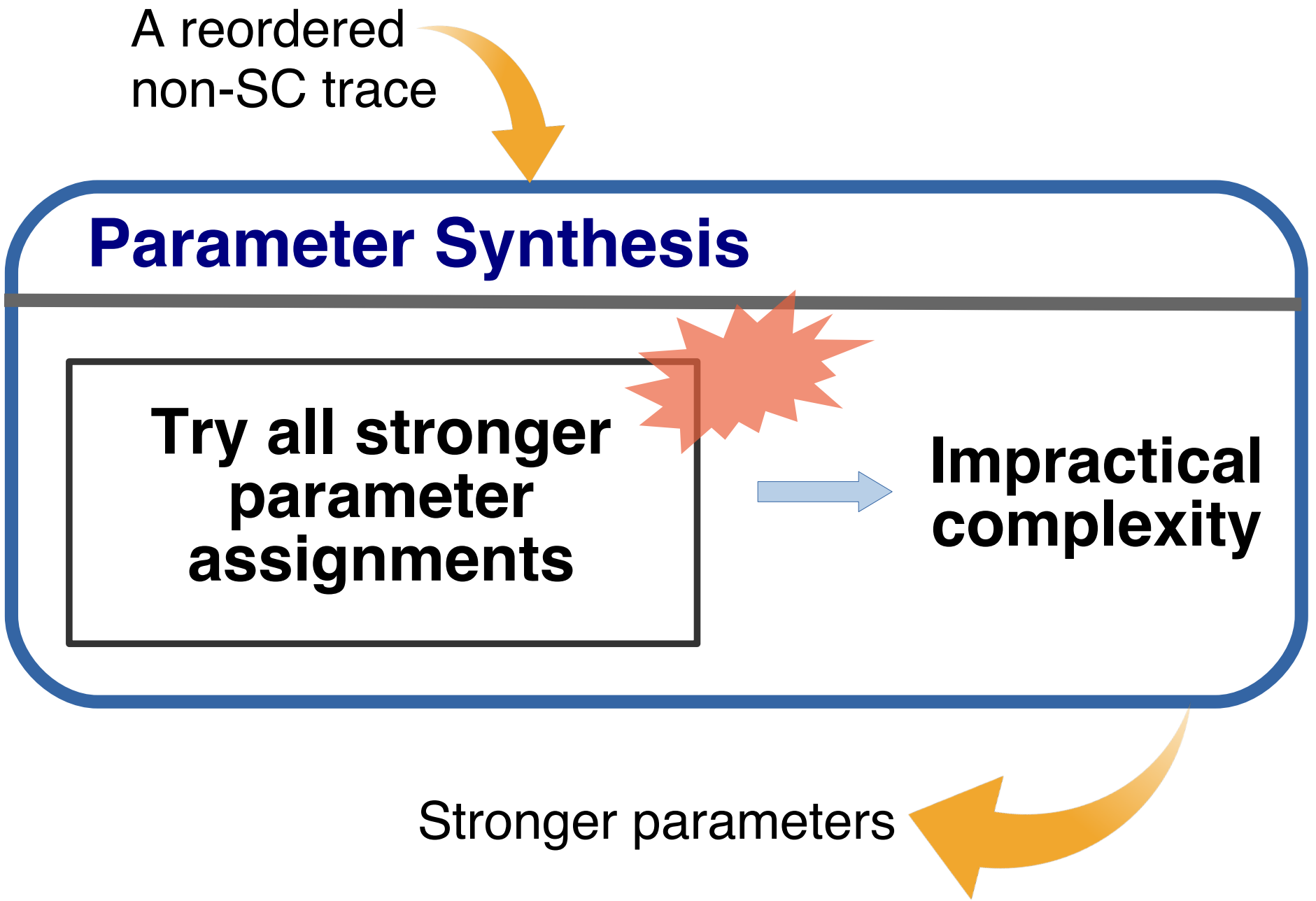
A non-SC trace

## Trace Reordering

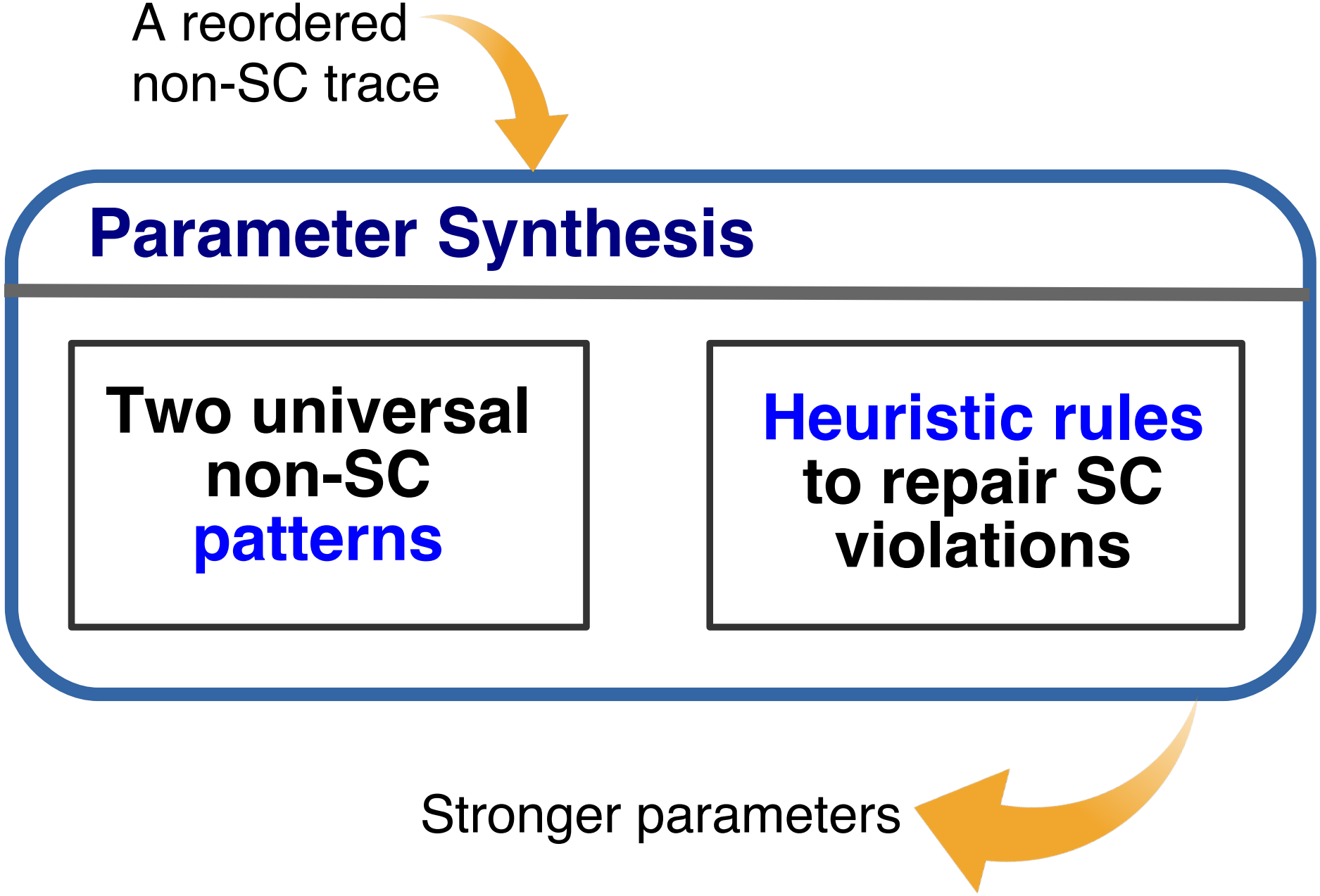
- Rearrange non-SC trace to mostly SC
  - expose real SC violations
- Ensure the involved SC violations are repairable
  - preserve *hb* & *SC* relation

A reordered trace

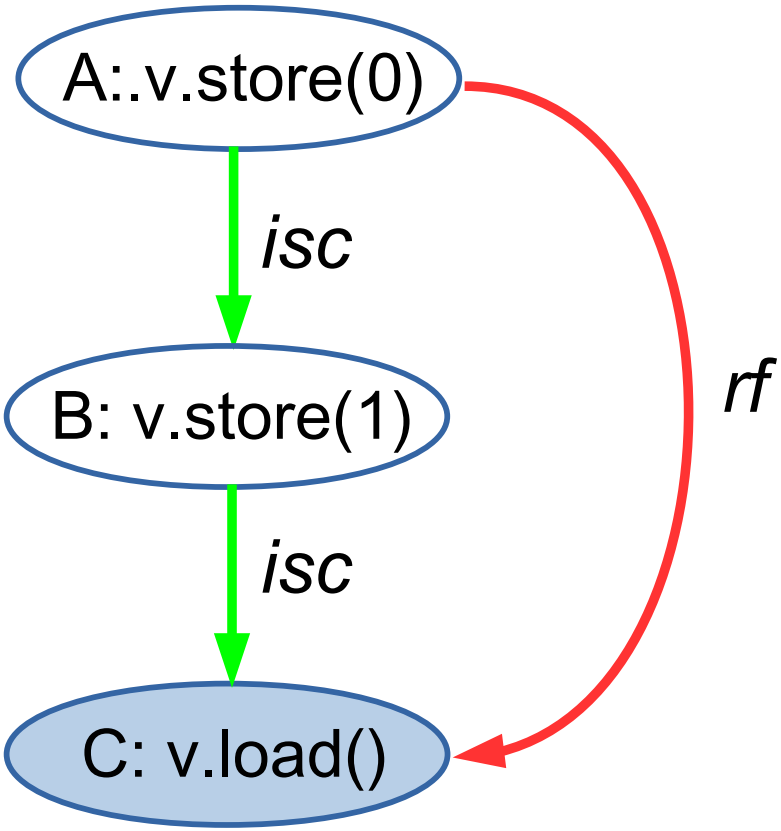
# Parameter Synthesis – Naïve Approach



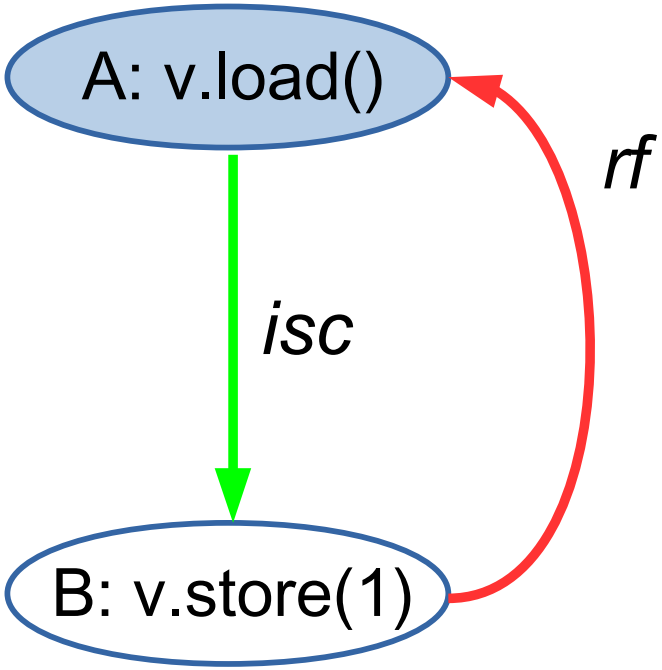
# Parameter Synthesis – Our Approach



# Two Universal Non-SC Patterns

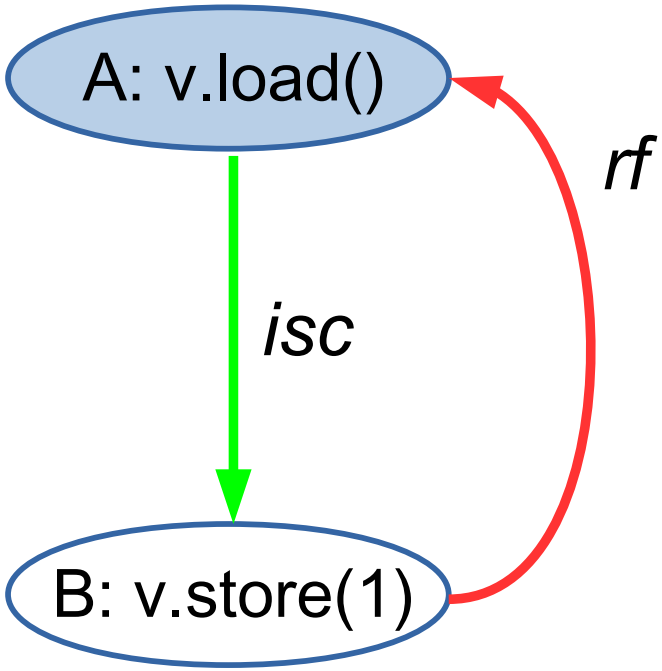



***Stale Read***

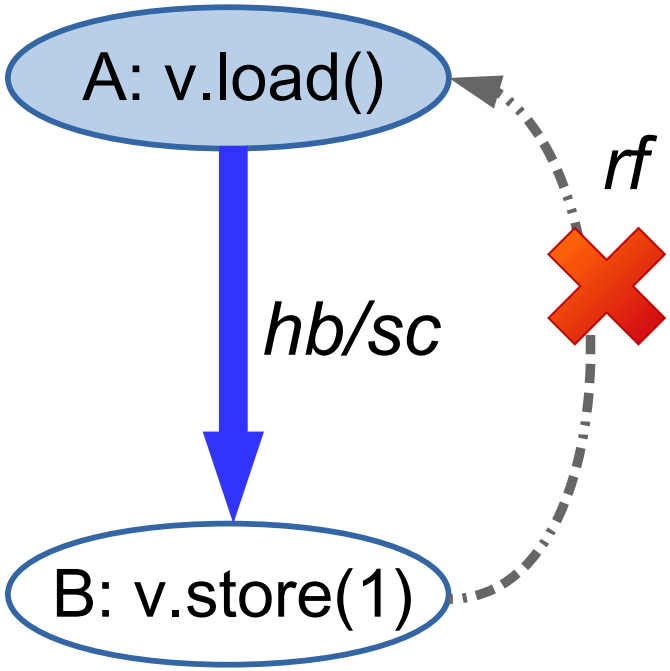



***Future Read***

# Inference Rule Example



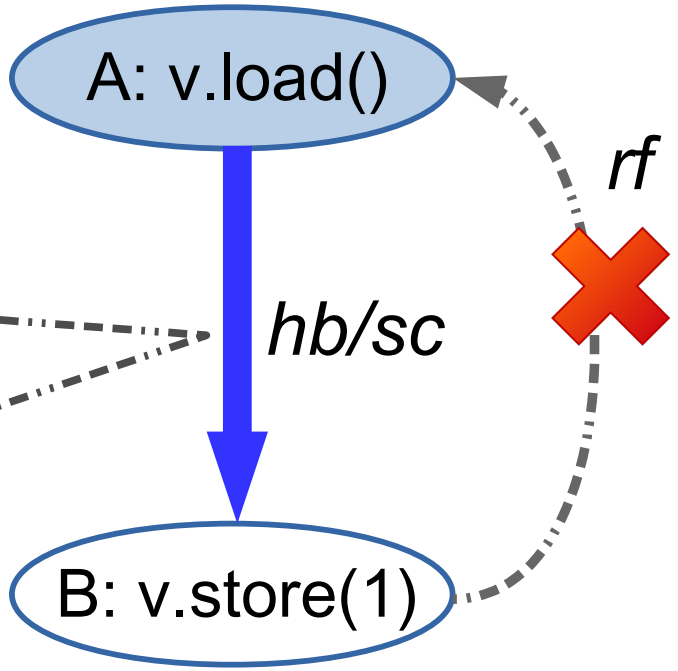

***Future Read***




***Inference Rule***

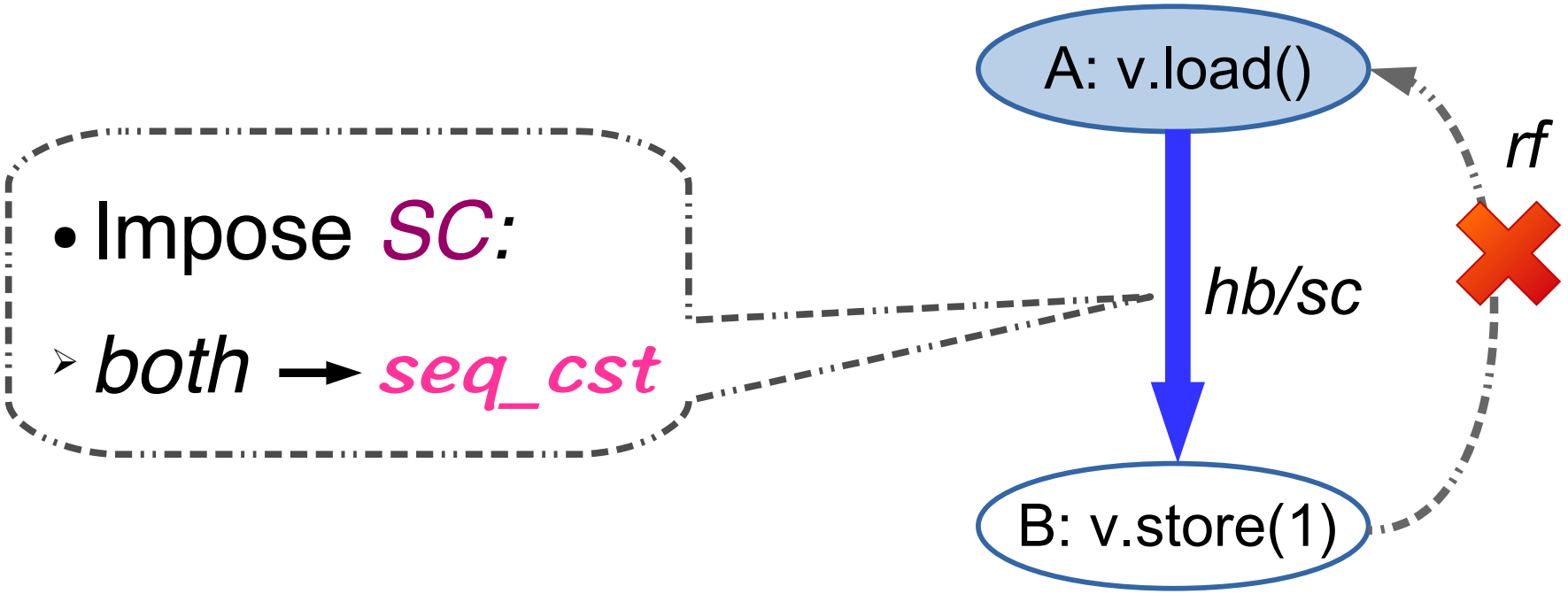
# Inference Rule Example

- *happens-before*:
  - *a chain of reads-from & sequenced-before:* ***establish synchronization***




***Inference Rule***

# Inference Rule Example




***Inference Rule***

# Termination of Inference Process

- Able to apply some rule for each violation
- Each rule application strengthens at least some parameter
- Finite number of parameter assignments

# Benchmarks

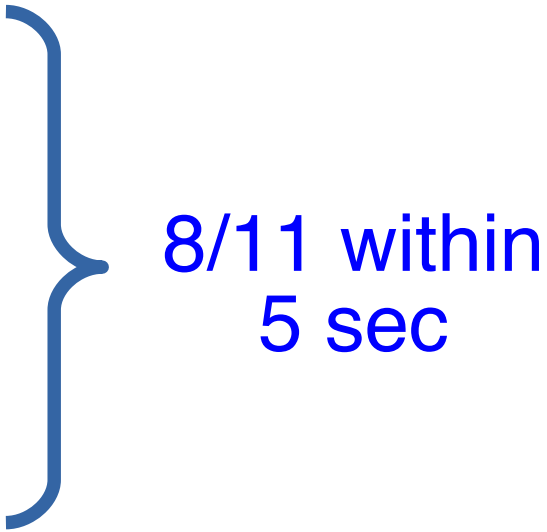
- **11 real-world data structures**

  - Barrier
  - Dekker's algorithm
  - Three concurrent queues: SPSC, M&S queue & MPMC
  - Three locks: Linux RW lock, Seqlock & MCS lock
  - Treiber stack
  - Chase-Lev deque
  - Concurrent hashtable

# Inference Algorithm Performance

- On Intel Core i7 3770

| Benchmarks | Inference Time (sec) |
|---|---|
| Chase-Lev deque | 536.322 |
| Dekker | 396.756 |
| Linux RW lock | 24.982 |
| M&S queue | 4.808 |
| MCS lock | 4.056 |
| MPMC | 0.143 |
| Seqlock | 0.095 |
| Barrier | 0.019 |
| Treiber's stack | 0.018 |
| Concurrent hashtable | 0.016 |
| SPSC | 0.015 |

# As Good As Manual Version

- Dekker
- Linux RW lock
- Treiber's stack
- Seqlock

# Better Than Manual Version

- MCS lock
- Barrier

# Expose Bugs of Manual Version

- M&S queue implementation
  - AutoMO infers two stronger parameters
  - Both are necessarily stronger (fixed two bugs)

# Close to Manual Version

- Chase-Lev deque
  - Only take 9 min to finish
  - Found an incorrect claim

# Overly Strong Parameters

- MPMC & Concurrent hashtable
  - Take advantage of SC-violations

# Related Work

- **Test behaviors for relaxed language models**
  - C/C++: CPPMEM, Nitpick, CDSChecker, Relacy...
  - Axiomatic model: MemSAT
- **Detect data race (lock-based)**
  - Glodilocks, RacerX, FastTrack, Eraser...
- **Automatic parallelization (sequential → parallel)**
- **Check SC**
  - Complexity: At least NP-Complete
  - Hardware M.M.: TRF, Herding cat, CheckFence, DFence...
- **Automatically infer SC**
  - Dfence (infer fences for hardware memory model)

# Conclusion

- **AutoMO**
  - Automatically infers memory order parameters for C/C++11 programs
  - Available on our site (http://plrg.eecs.uci.edu/automo/)

# Questions