z13: Simultaneous Multithreading and System Level Testing

Ali Duale
Dennis Wittig
Shailesh Gami

Systems Assurance Kernel (SAK)

- Main system level test for z Systems Architecture Implementations
  - Enterprise/Business Class Hardware
  - Software emulators
- Test Oriented Operating System and system level test exercisers
- In use for decades
- Architecture Compliance
- Backward compatible
- Tolerant for unstable implementations/early development
- Capability of error/test case reproducibility
- Operator controllable

Systems Assurance Kernel (SAK) - continued

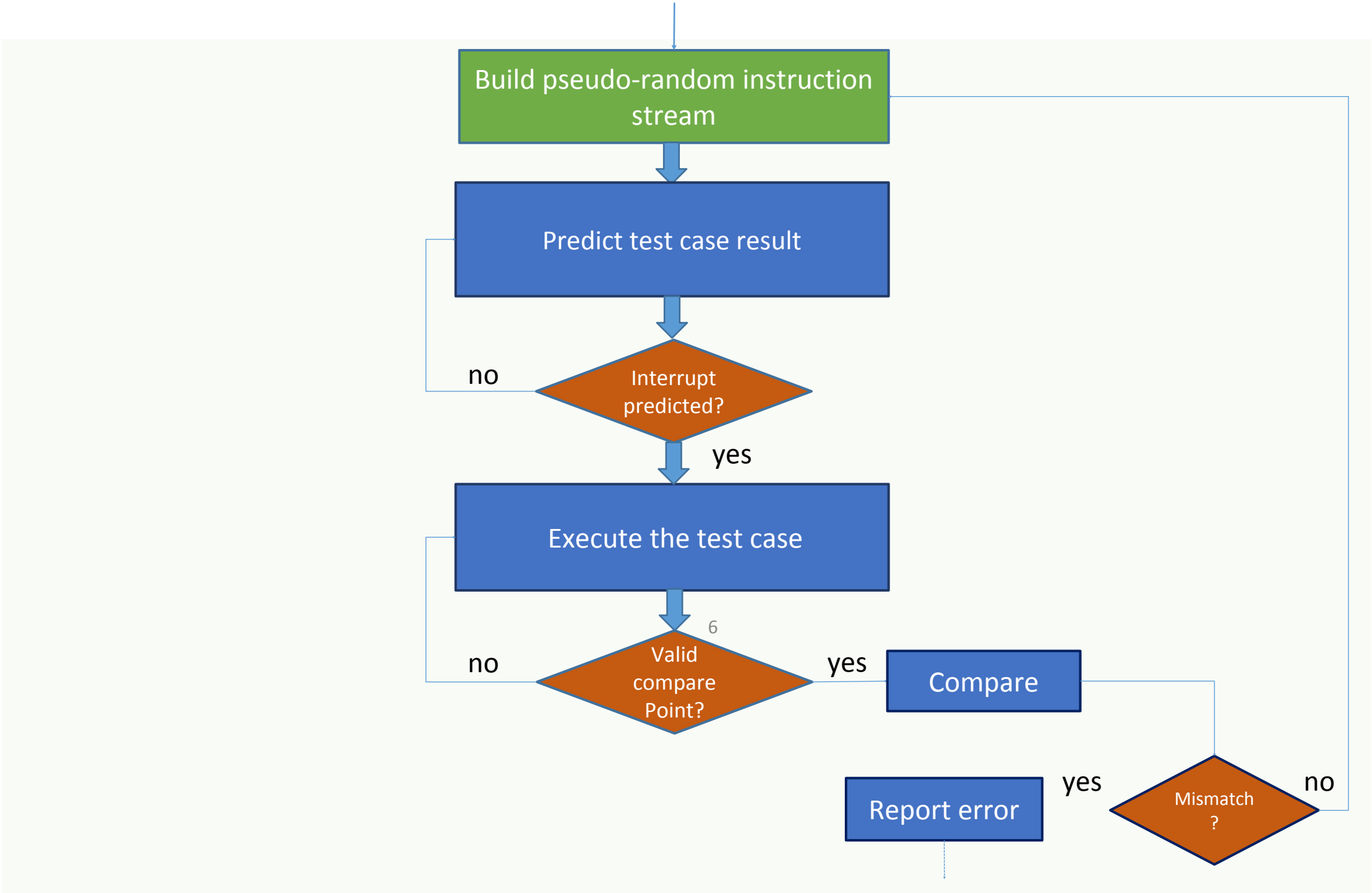
- Test exercisers:
  - CPU
  - Memory hierarchy
  - IO
  - Crypto
  - Any combinations of the above
  - Pseudo-random test generation
- Capable of running *n* copies of same/different test exercisers
  - Resource availability - limit
- Pseudo-Random Test case build and result prediction are done with full speed of the machine under test
  - Millions of test cases per minute
  - Endless combinations of test

Exerciser Focuses

- Single-processor (UNI)
- Multiple-Processor (MP)
- General and Control Instructions
- Branching
- Floating-Point (HFP, BFP and DFP)
- SIMD
- Cache coherency, Serializations, Memory access ordering
- IO Subsystem and Devices
- Crypto Devices
- etc

General Test Flow

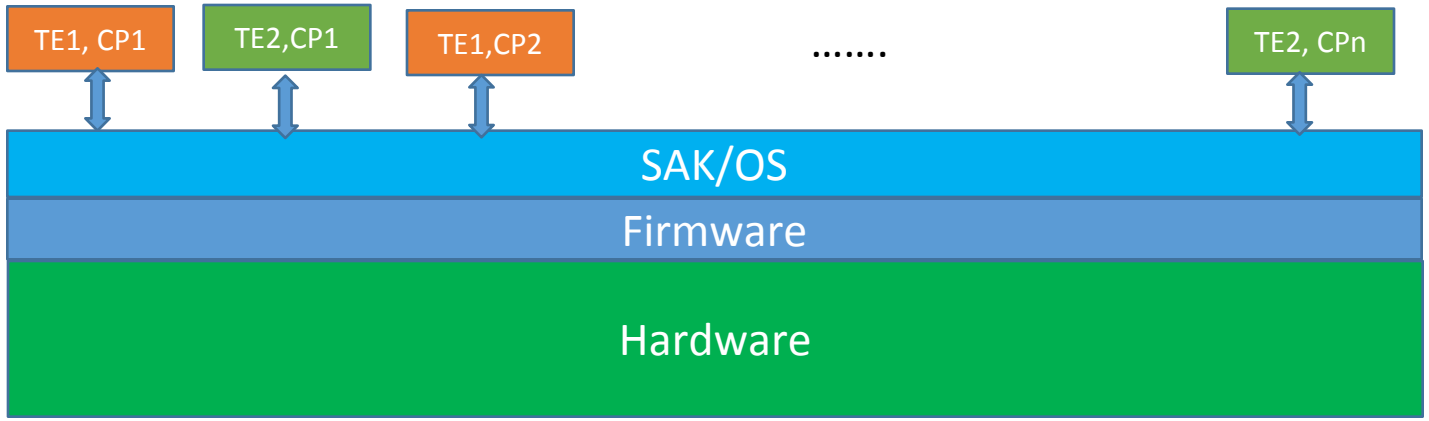
1. Build Instruction Stream
2. Predict result to interruption point
3. Execute Instruction Stream
4. Post process result as required
5. Compare result
6. Report any errors
7. Repeat the process until stopped


Build pseudo-random instruction stream
Predict test case result
Interrupt predicted?
no
yes
Execute the test case
Valid compare Point?
no
yes
Compare
Mismatch ?
yes
no
Report error

Simultaneous Multi Threading - SMT

- With SMT each core supports more than one instruction stream
  - Each thread maintains a unique and complete CPU context
  - All threads share/compete for core execution resources
- Each thread executes its own instruction stream
  - Streams do not have to be based on same architectures
  - Streams are totally independent of one another
    - Each appears as if it runs on a complete CPU in non SMT systems
- Test challenge:
  - Build basic infrastructure for SMT
  - Build instruction streams for each thread
  - SMT with random number of threads enabled
  - SMT with streams of same/different architectures running on threads
  - Capability of multiple instruction stream rebuild with a single seed
  - Direct and indirect testing

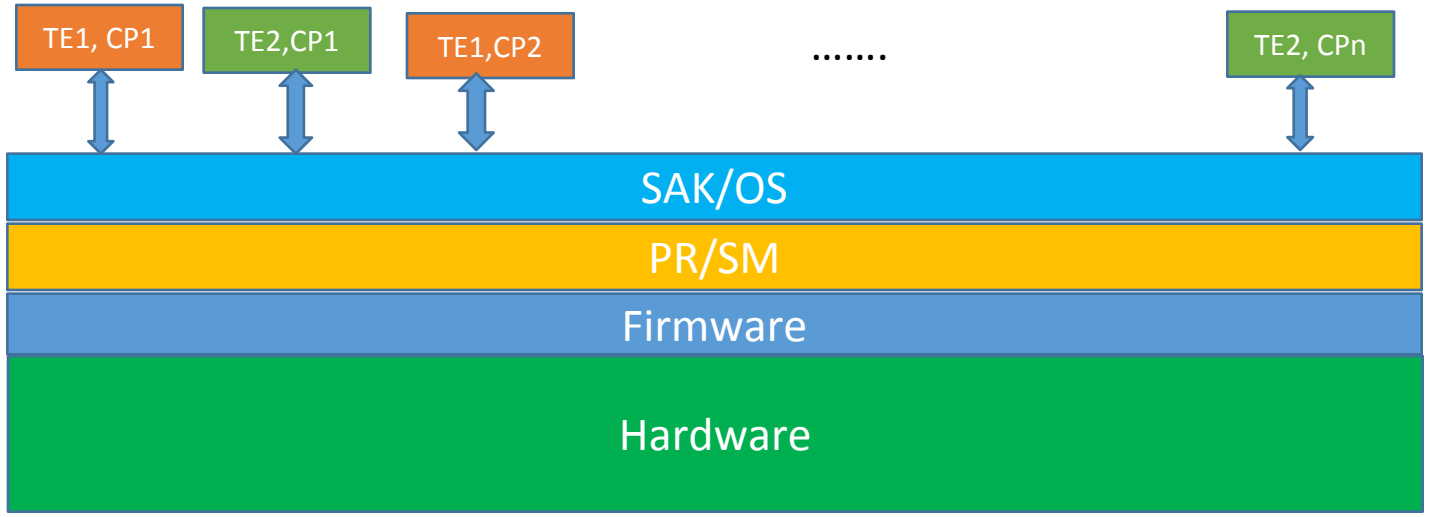
SMT: Direct Testing



TEi= Test Exerciser i

CPj = Copy j

SMT: Indirect Testing



TEi= Test Exerciser i

CPj = Copy j

SMT Test Case Build: Overview

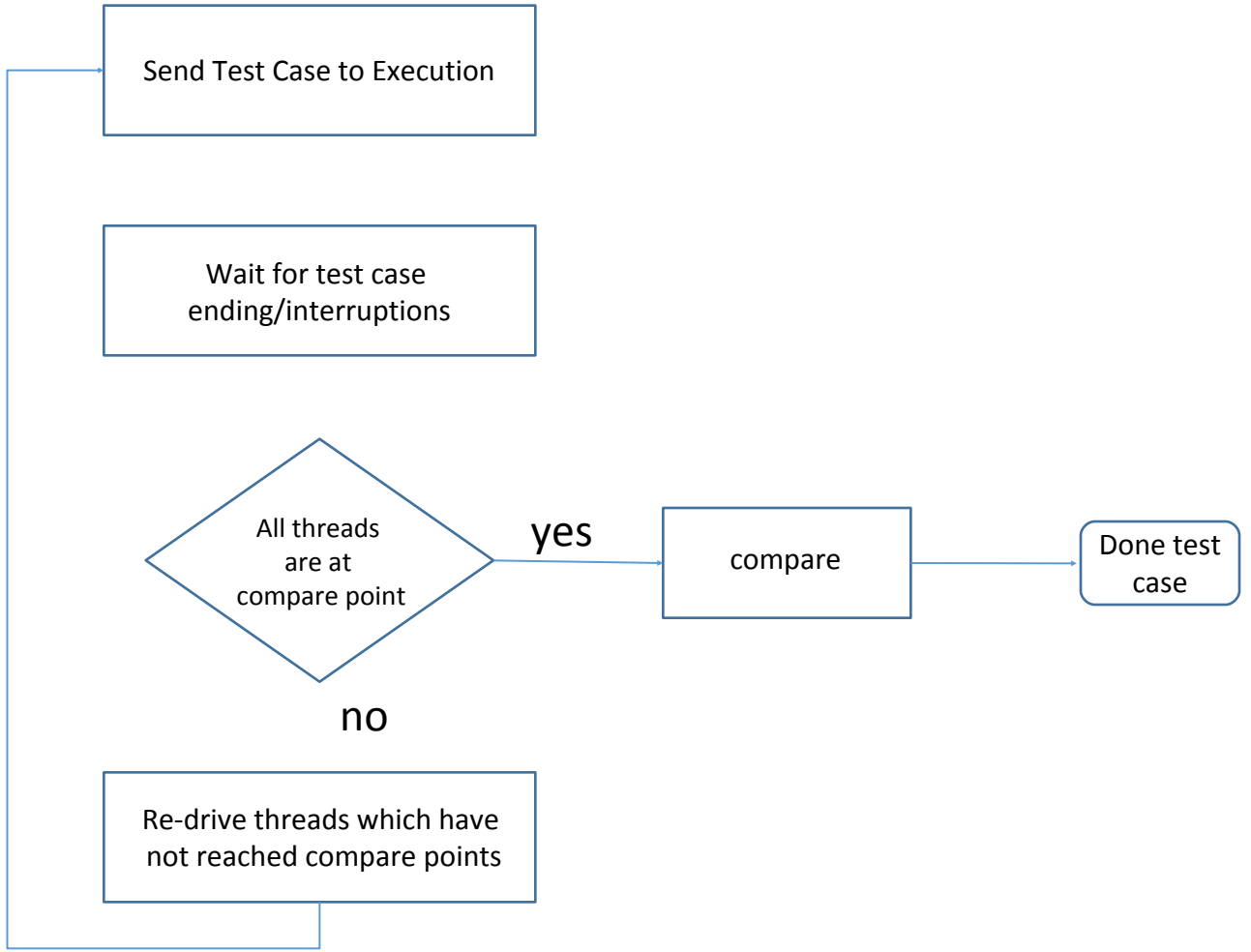
- Enable Test Exercisers to test SMT
- Keep backward compatibility
  - Exerciser must remain capable of testing prior z Systems implementations
- Stress new architecture features
  - New test case build methodology
- Alternate SMT being on and off
  - Randomly decided/operator controlled
- Transparency in result prediction
  - Result prediction processor: oblivious to thread number etc

SMT: Test Case Generation

- Obtain needed memory buffers for all threads
- Randomize test environment for each thread
  - Addressing mode
  - Exceptions
  - Address translation means
  - Functions enablement/disablement
  - Architecture mode
    - Allow building different architectures for different threads
  - etc
- Accommodate common control fields (as required by architecture)

SMT Test Sequence

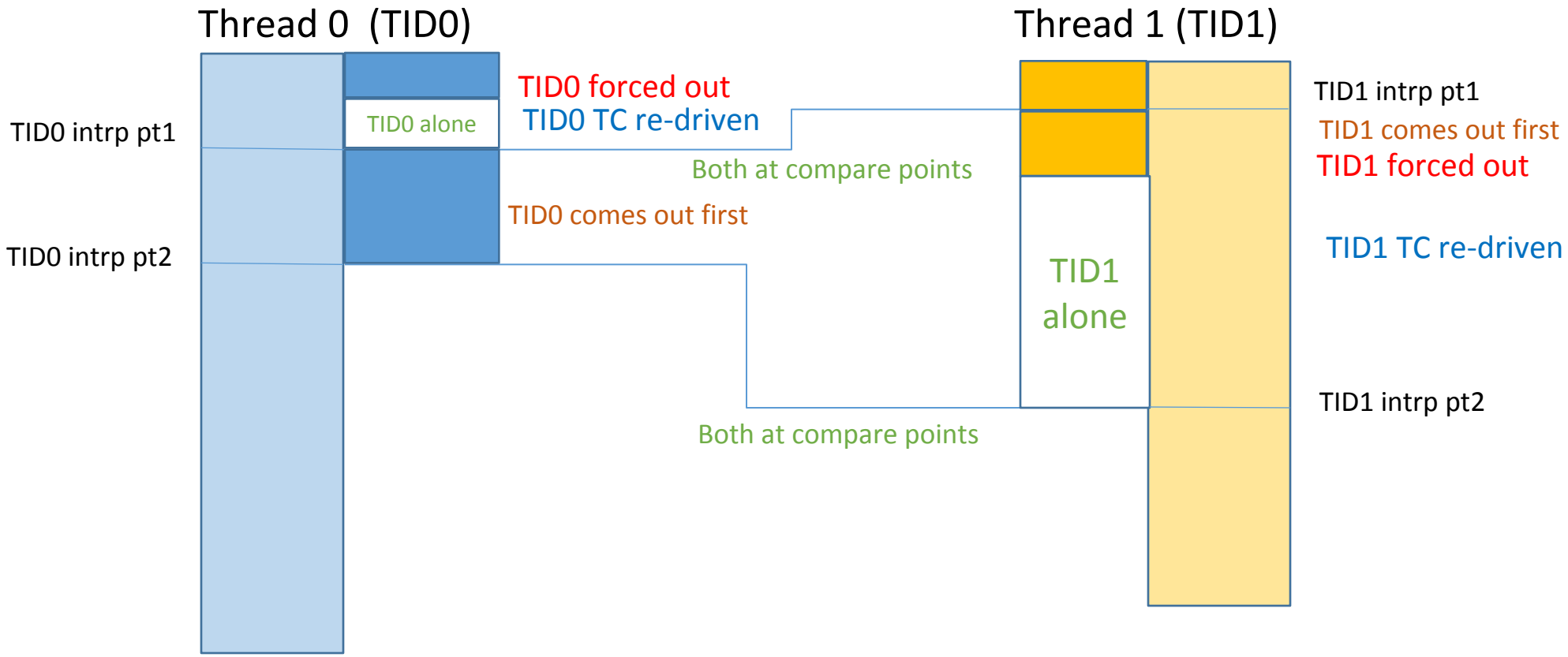
- Build Test case for each enabled thread
- Predict results for each enabled thread (up to a compare point)
- Execute the test cases in parallel (SMT in action here)
  - Drive all thread execution to the next compare point
    - all threads may not reach the end of their test case at the same time
  - Maintain execution history for error analysis
- Compare results
- Output human readable error and trace reports


Send Test Case to Execution
Wait for test case ending/interruptions
All threads are at compare point
yes
compare
Done test case
no
Re-drive threads which have not reached compare points

SMT Test Case Execution Flow

- Execute the test case until all threads exit test cases
- Gather context of each thread
- Check for “early exit” of test case due to:
  - Interruptible instruction in any thread
  - Forced to exit: One of the threads completed before others
  - Transaction Execution abort in any thread and may need retry
- “Early exit” cases require test case re-drive in one to *n* threads until all reach the next SMT compare point
- A completed thread is not re-dispatched into execution while other threads are re-driven into test case execution

Execution and Test Case Re-drive Flow Example



THANK YOU

Q/A