BREKER™ THE LEADER IN PORTABLE STIMULUS

Advanced RISC-V Verification Technique Learnings for SoC Validation

Using Breker SystemVIP for RISC-V System Ready

Adnan Hamid, CTO, Breker Verification Systems

Verification Futures Austin 2023

Agenda

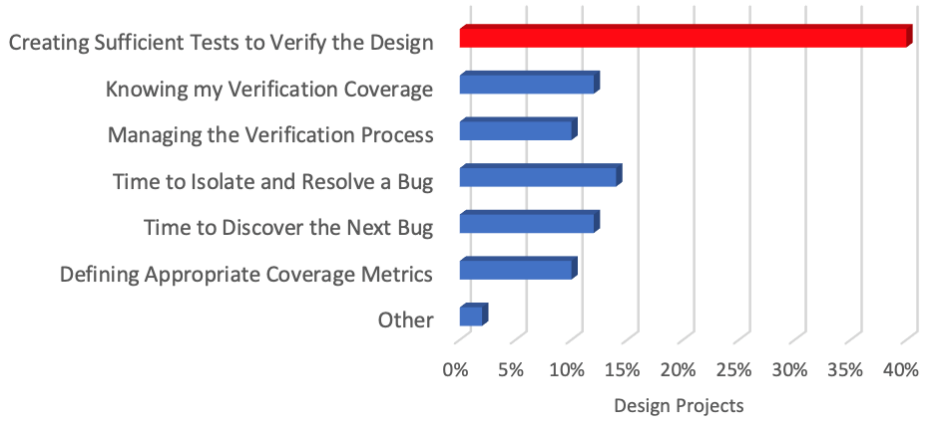
- Test Suite Synthesis and SystemVIP
- RISC-V Core Verification SystemVIP
- RISC-V SoC Verification SystemVIP

Agenda

- Test Suite Synthesis and SystemVIP
- RISC-V Core Verification SystemVIP
- RISC-V SoC Verification SystemVIP

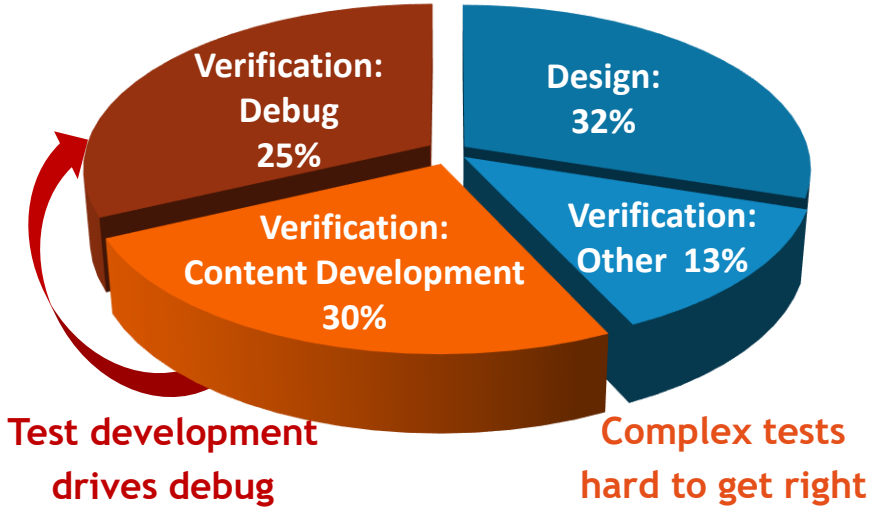
The High Cost of Developing Test Content

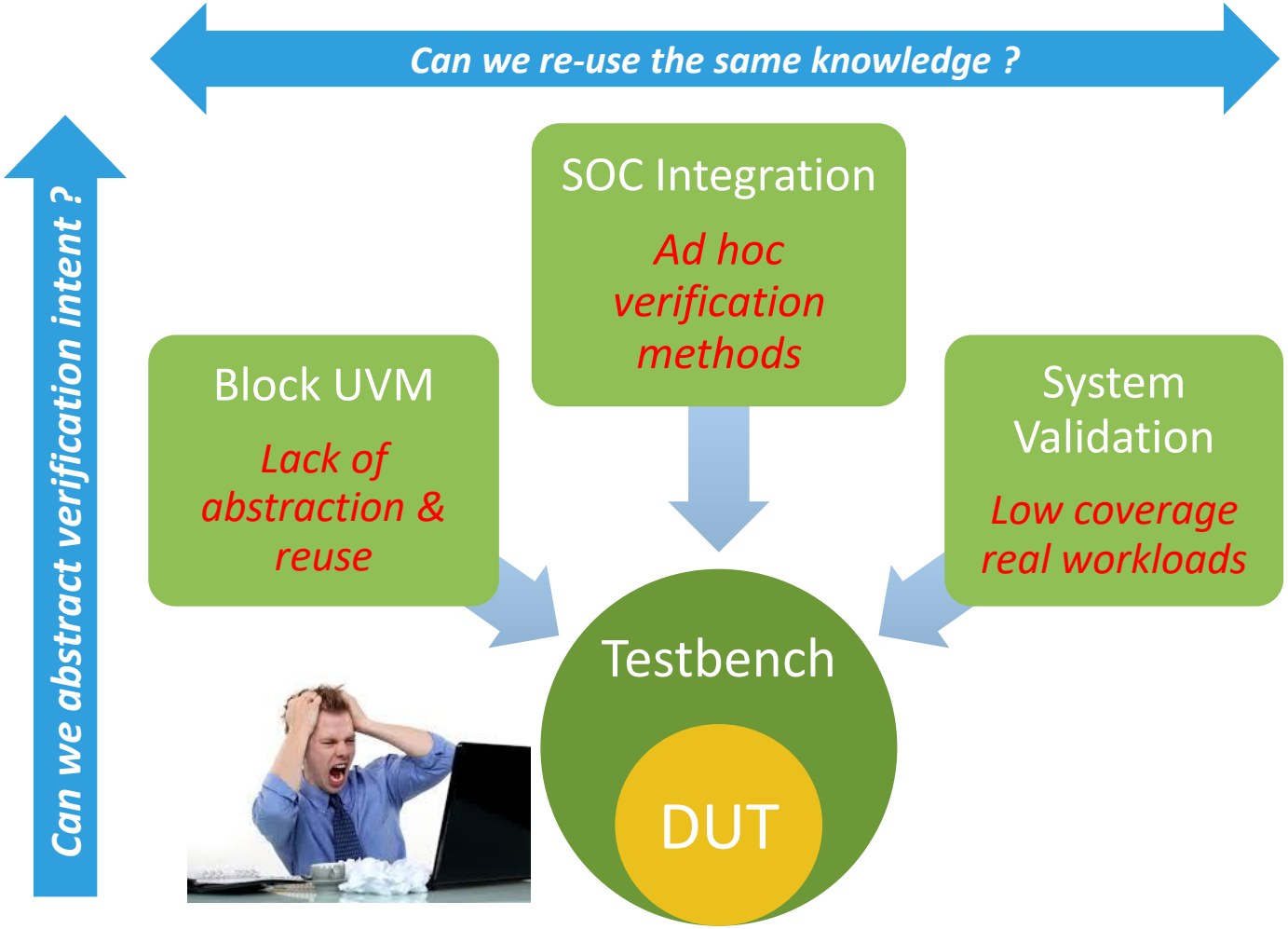
Largest Functional Verification Challenge



Source: Wilson Research 2020

Project Resource Deployment



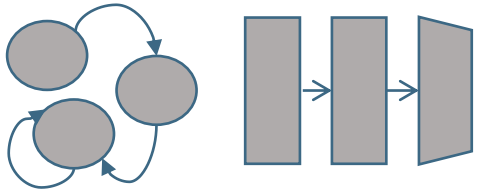


Why? Resource Intensive Test Content

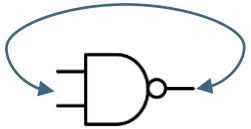
Test Suite Synthesis… Analogous to Logic Synthesis

BREKER™

Design Synthesis

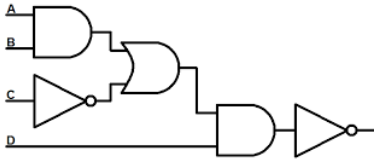


Constrain

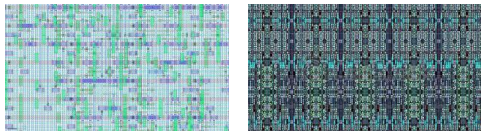


Timing/Area Constraints

Synthesize

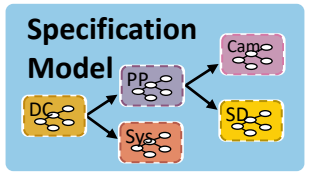


Optimize



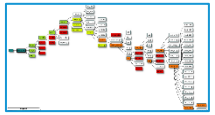
Test Suite Synthesis

Describe intent



Constrain

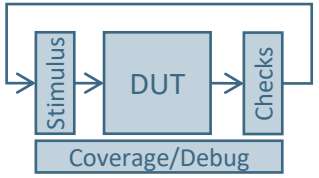
Specify goals



Coverage Constraints

Synthesize

Generate implementation



Optimize

Map to platform

Breker Core Technology

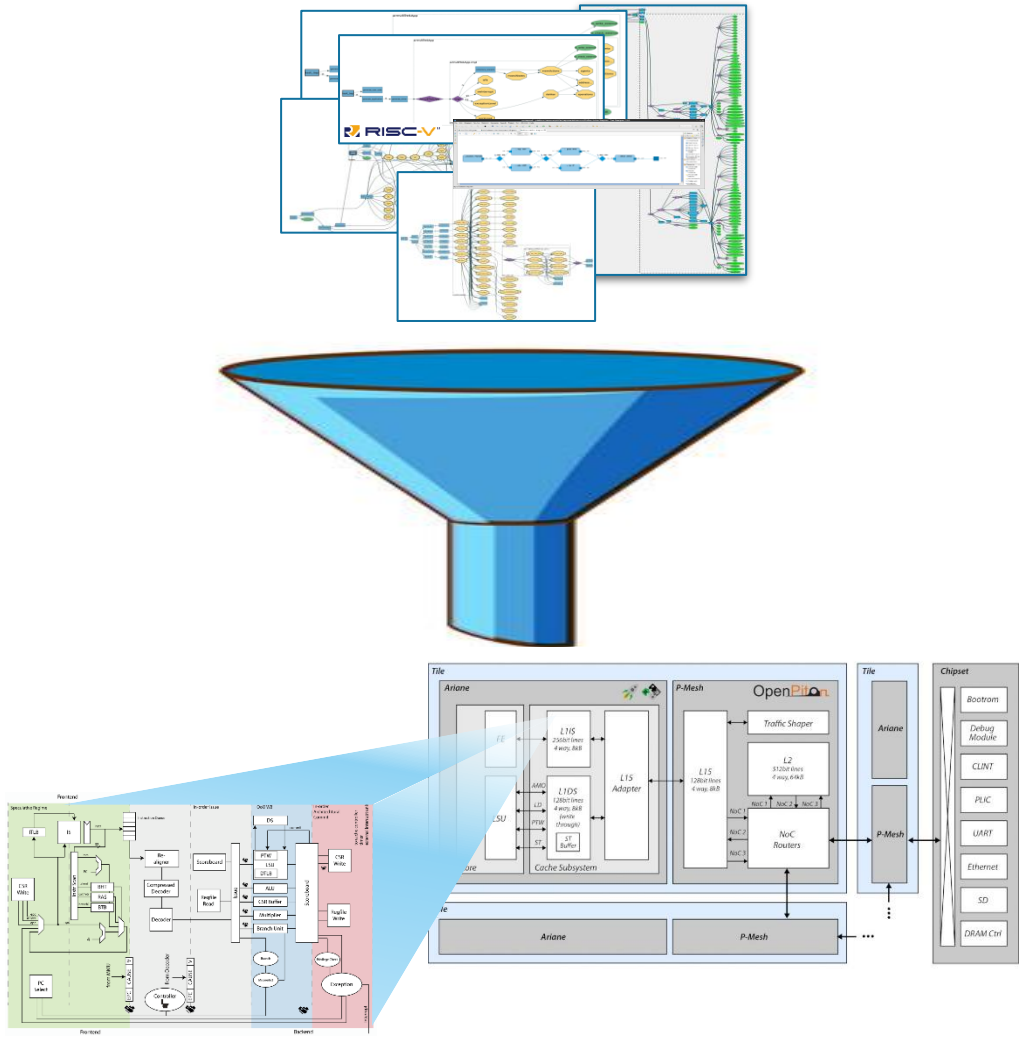
AI Planning Algorithms

3D Coverage Closure

Synthesizable VerificationOS

AI Planning Algorithms

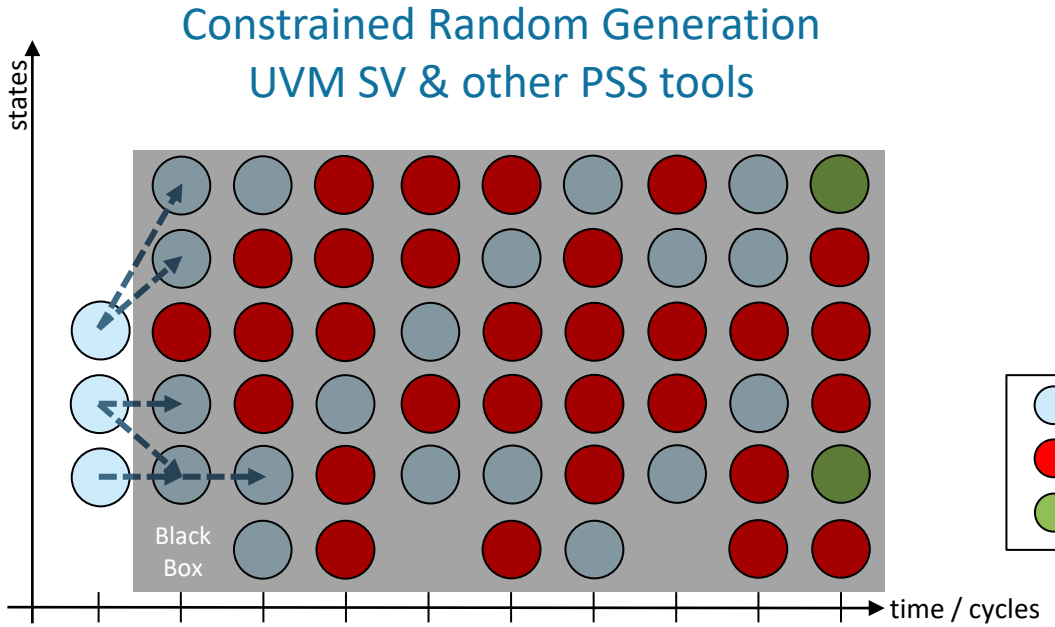
Breker SystemVIP Library

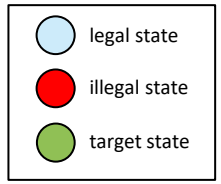


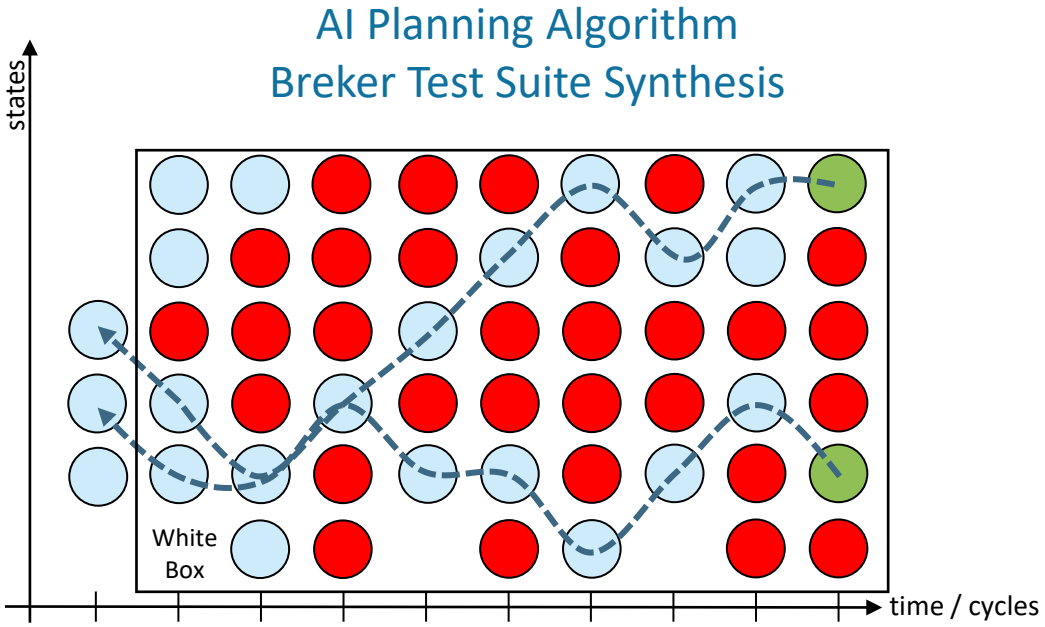
SoC SystemVIP Library

- The ***RISC-V Core TrekApp*** provides fast, pre-packaged tests for RISC-V Core and SoC integrity issues
- The ***Coherency TrekApp*** verifies cache and system-level coherency in a multiprocessor SoC
- The ***End-to-end IP TrekApp*** IP test sets ported from UVM to SoC
- The ***Power Management TrekApp*** automates power domain switching verification
- The ***Security TrekApp*** automates testing of hardware access rules for HRoT fabrics
- The ***Networking & Interface TrekApp*** automates packet generation, CXL, UCIe interface tests

Constrained Random vs AI Planning Algorithm Synthesis







Design black box, shotgun tests to search for key state

Low probability of finding complex bug

Starts with key state and intelligently works backward through space

Deep sequential, optimized test discovers complex corner-cases

White Paper Discussing AI Planning Algorithm Test Generation on Breker Website

A Look At RISC-V

- Open Instruction Set Architecture (ISA) creating a discontinuity in the market
- Appears to be gaining significant traction in multiple applications
- Significant verification challenges
  - Arm spends $150M per year on 10^{15} verification cycles per core
  - Hard for RISC-V development group to achieve this same quality
  - Lots of applications expands verification requirements
  - Requires automation, reuse and other new thinking

RISC-V Verification & Validation Tasks

BREKER™

Complexity

Core Integrity

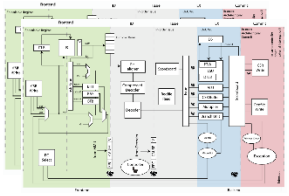
- Interrupts/Paging/Memory Order
- Micro-architecture functionality
- ISA compliance
- First Instruction Completion

Complexity

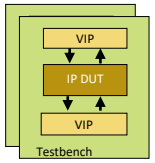
IP Integrity

- End-to-End Use Cases
- Concurrency Testing
- IP Configuration

CPU Cores



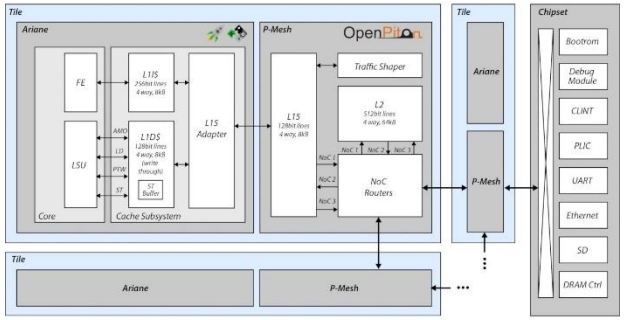
Uncore IPs



Complexity

SoC Integrity

- Power/Performance Profiling
- Security
- Power Management
- Interrupts/Paging/Memory Order
- Cache Coherency



Firmware

Firmware Integrity

- End-to-End Use Cases
- Concurrency Testing
- HW/Firmware Compatibility

Complexity

Breker RISC-V SystemVIP Portfolio

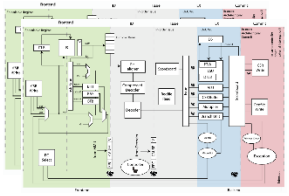
SVIPs for SoC Integrity

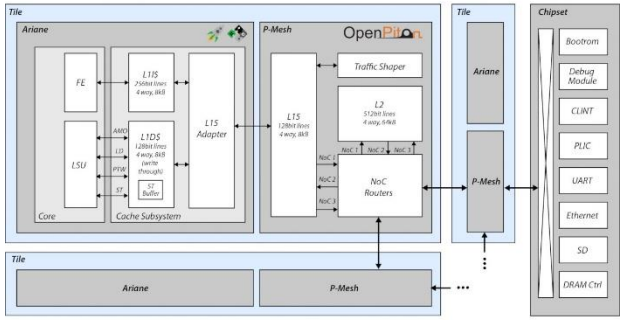
- SoC Cache Coherency
- Memory Ordering
- Power Management
- System Interrupts
- …

SVIPs for Core Integrity

- Register Hazards
- Load/Store
- Core Cache Coherency
- Core Interrupts
- …

CPU Cores



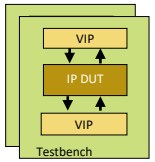


Firmware

SVIPs for Firmware Integrity

- Mem2Mem (dma)
- IO Offload (PCIE/Eth)
- WQ Servicing
- …

Uncore IPs



SVIPs for IP Integrity

- Mem2Mem (dma)
- IO Offload (PCIE/Eth)
- WQ Servicing
- …

Single Source of Truth for all stages of Verification & Validation

BREKER™

SVIPs for IP Integrity

- Mem2Mem (dma)
- IO Offload (PCIE/Eth)
- WQ Servicing
- …

SVIPs for Core Integrity

- Register Hazards
- Load/Store
- Core Cache Coherency
- Core Interrupts
- …

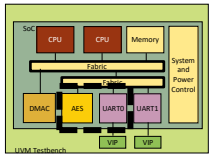
SVIPs for SoC Integrity

- SoC Cache Coherency
- Memory Ordering
- Power Management
- System Interrupts
- …

SVIPs for FW Integrity

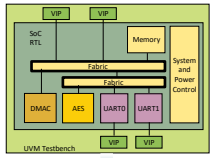
- Mem2Mem (dma)
- IO Offload (PCIE/Eth)
- WQ Servicing
- …

Test Suite Synthesis

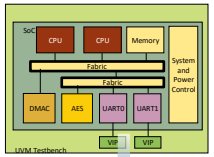


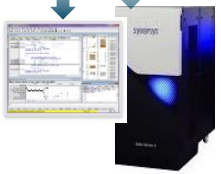
Virtual Platform Environment

UVM Block Environment

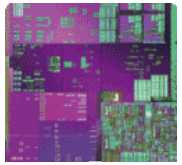


Simulation Acceleration





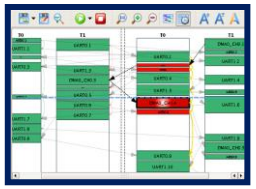
Hybrid Emulation Environment



Silicon / Prototyping Environment

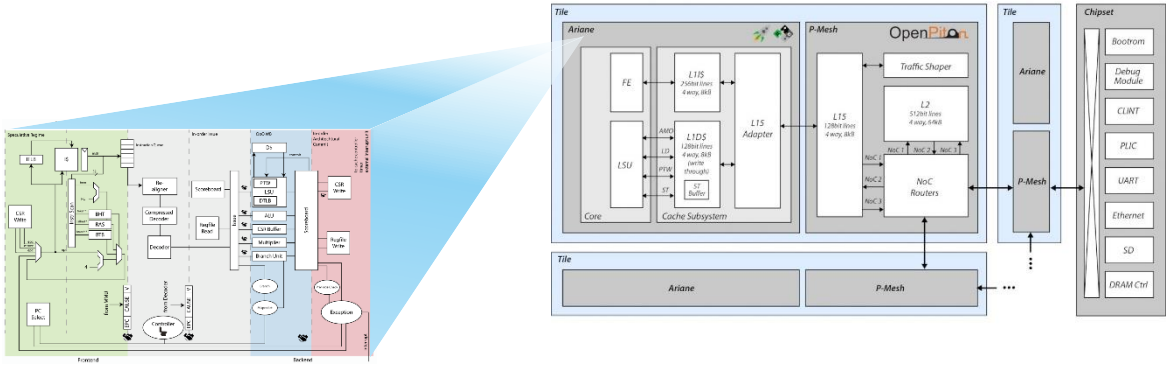
High Level Debug

Coverage Analysis



Performance Profiling

Different Challenges for Core vs SoC Verification



RISC-V Core Verification Challenges

Random Instructions	Do instructions yield correct results
Register/Register Hazards	Pipeline perturbations dues to register conflicts
Load/Store Integrity	Memory conflict patterns
Conditionals and Branches	Pipeline perturbations from synchronous PC change
Exceptions	Jumping to and returning from ISR
Asynchronous Interrupts	Pipeline perturbations from asynchronous PC change
Privilege Level Switching	Context switching
Core Security	Register and Memory protection by privilege level
Core Paging/MMU	Memory virtualization and TLB operation
Sleep/Wakeup	State retention across WFI
Voltage/Freq Scaling	Operation at different clock ratios
Core Coherency	Caches, evictions and snoops

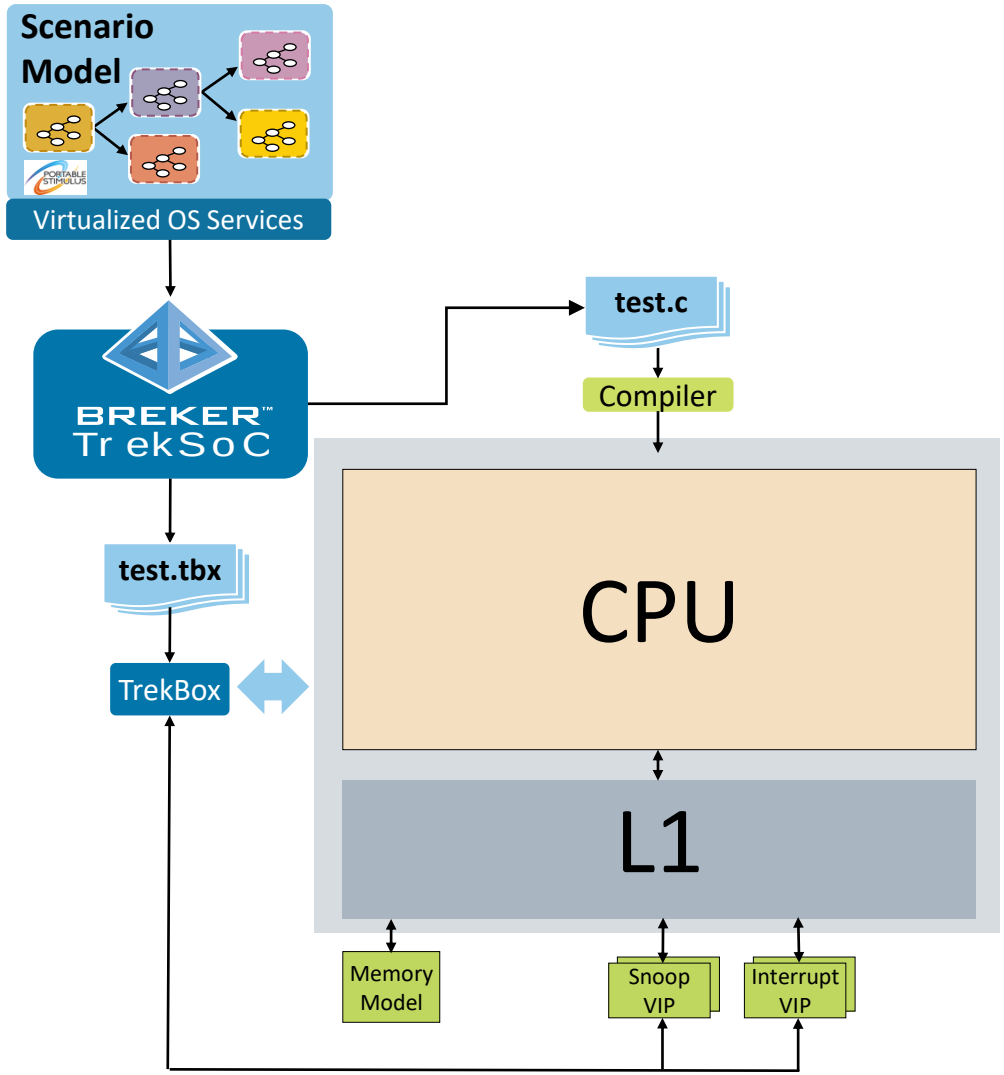
RISC-V SoC Verification Challenges

System Coherency	Cover all cache transitions, evictions, snoops
System Paging/IOMMU	System memory virtualization
System Security	Register and Memory protection across system
Power Management	System wide sleep/wakeup and voltage/freq scaling
Packet Generation	Generating networking packets for I/O testing
Interface Testing	Analyzing coherent interfaces including CXL & UCIe
Random Memory Tests	Test Cores/Fabrics/Memory controllers across DDR, OCRAM, FLASH etc
Random Register Tests	Read/write test to all uncore registers
System Interrupts	Randomized interrupts through CLINT
Multi-core Execution	Concurrent operations on fabric and memory
Memory Ordering	For weakly order memory protocols
Atomic Operation	Across all memory types

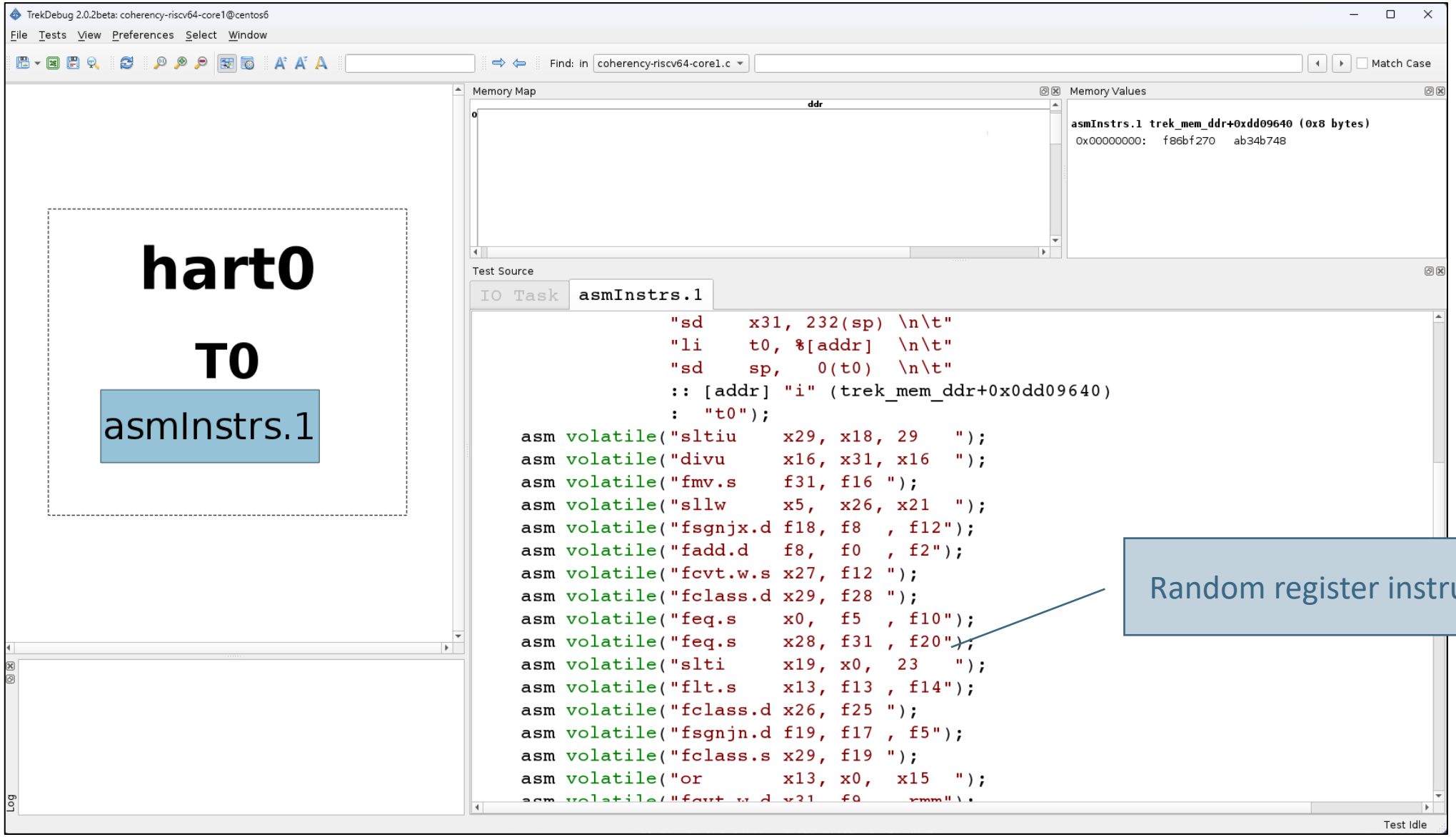
Agenda

- Test Suite Synthesis and SystemVIP
- RISC-V Core Verification SystemVIP
- RISC-V SoC Verification SystemVIP

RISC-V Core Testbench Integration

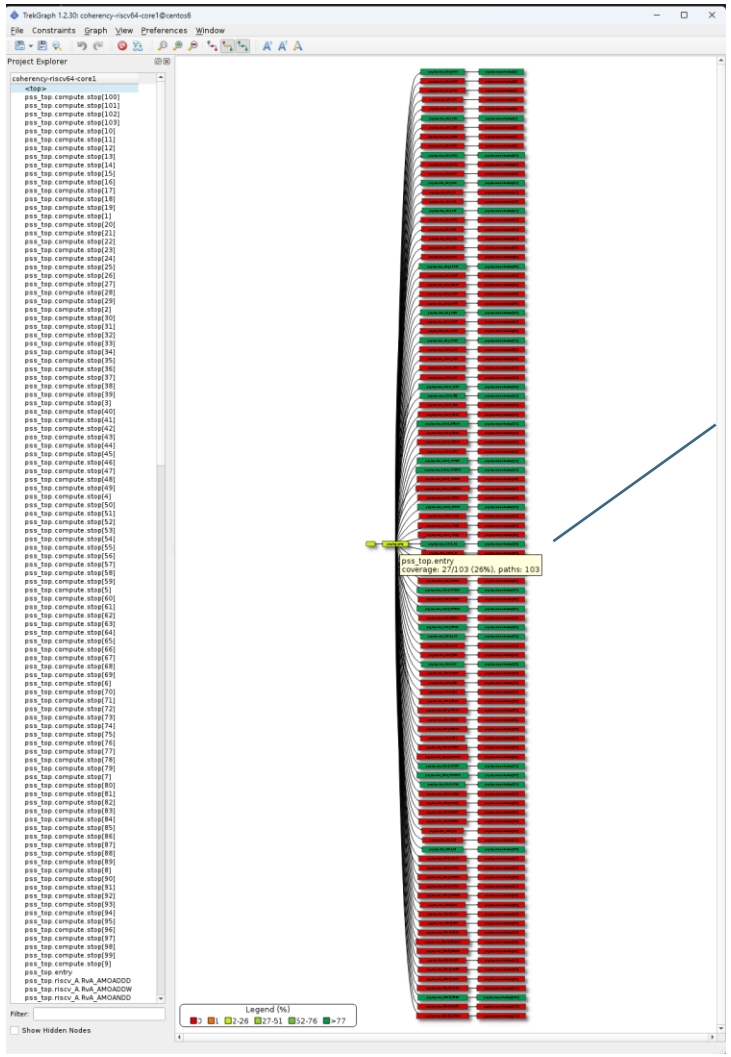


RV64 Core Instruction Generation



Random register instructions

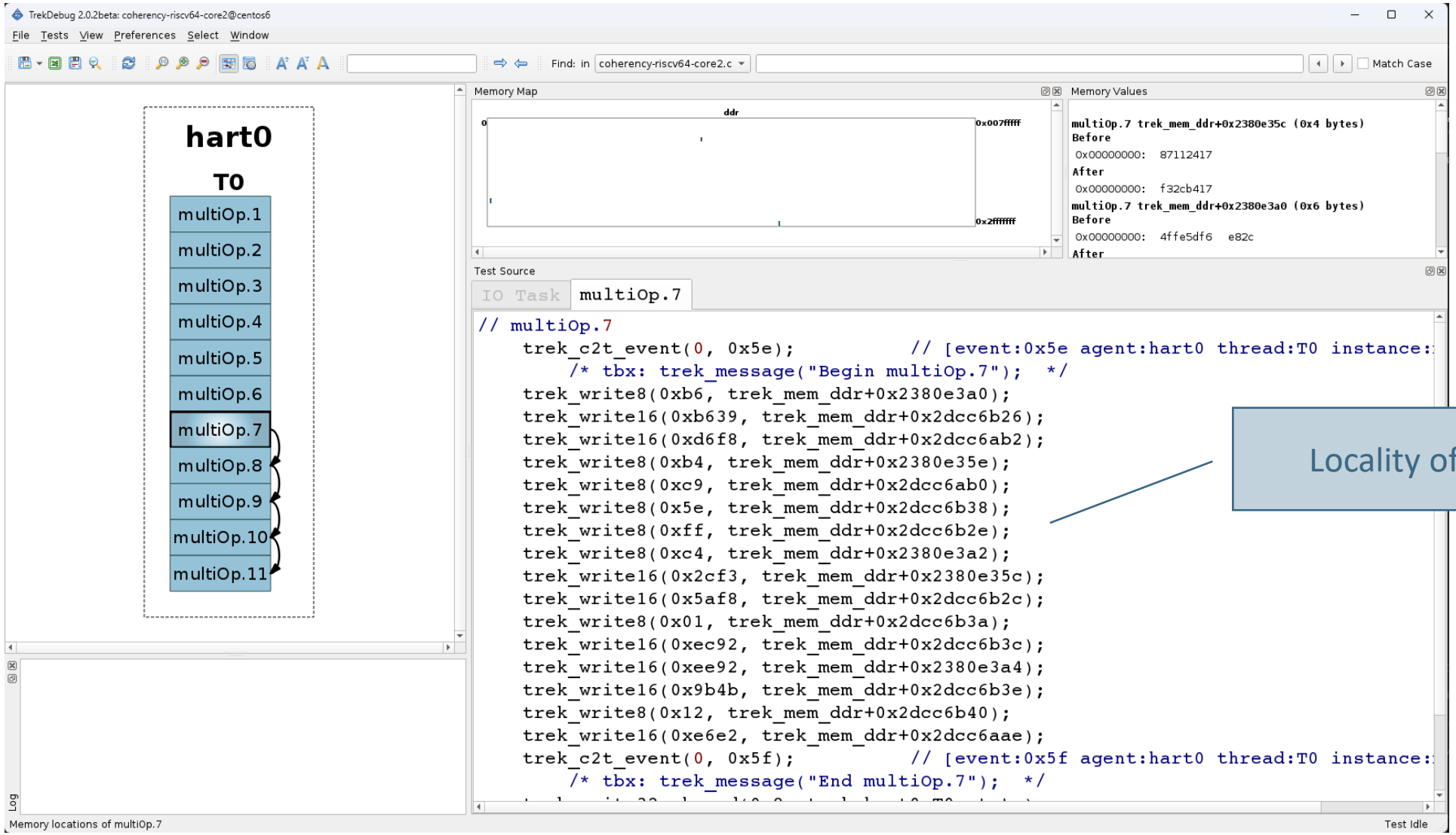
Instruction Coverage Analysis



27/103 reachable opcode have been exercised

Atomics, loads and stores not reachable in register only test

RV64 Core Load/Store



Locality of write addrs

Example Address Allocation Patterns

- Random Clusters with locality of reference

```
// memAllocAddrSlice allocated setId:0x1 of 0x4 blocks
// memAllocAddrRand size:0x8 addr: trek_mem_ddr+0x08b810c8
// memAllocAddrRand size:0x8 addr: trek_mem_ddr+0x2380e378
// memAllocAddrRand size:0x8 addr: trek_mem_ddr+0x2380e380
// memAllocAddrRand size:0x8 addr: trek_mem_ddr+0x2380e370
```

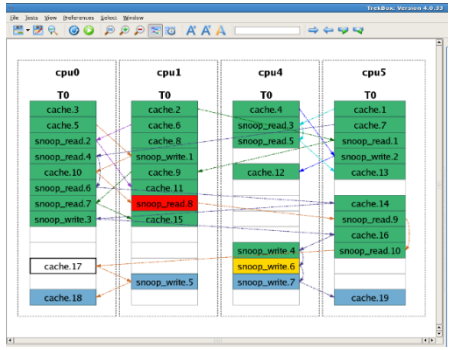
- Stride Patterns across fixed address distances

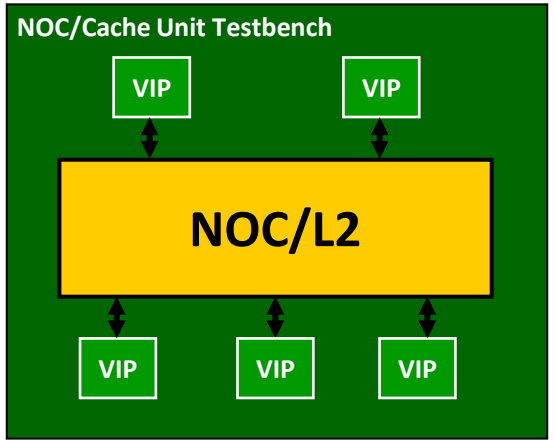
```
// memAllocAddrSlice allocated setId:0x1 of 0x4 blocks
// memAllocAddrStride stride_len:0x2000 size:0x8 addr: trek_mem_ddr+0x08b830c8
// memAllocAddrStride stride_len:0x2000 size:0x8 addr: trek_mem_ddr+0x08b850c8
// memAllocAddrStride stride_len:0x2000 size:0x8 addr: trek_mem_ddr+0x08b870c8
// memAllocAddrStride stride_len:0x2000 size:0x8 addr: trek_mem_ddr+0x08b890c8
```

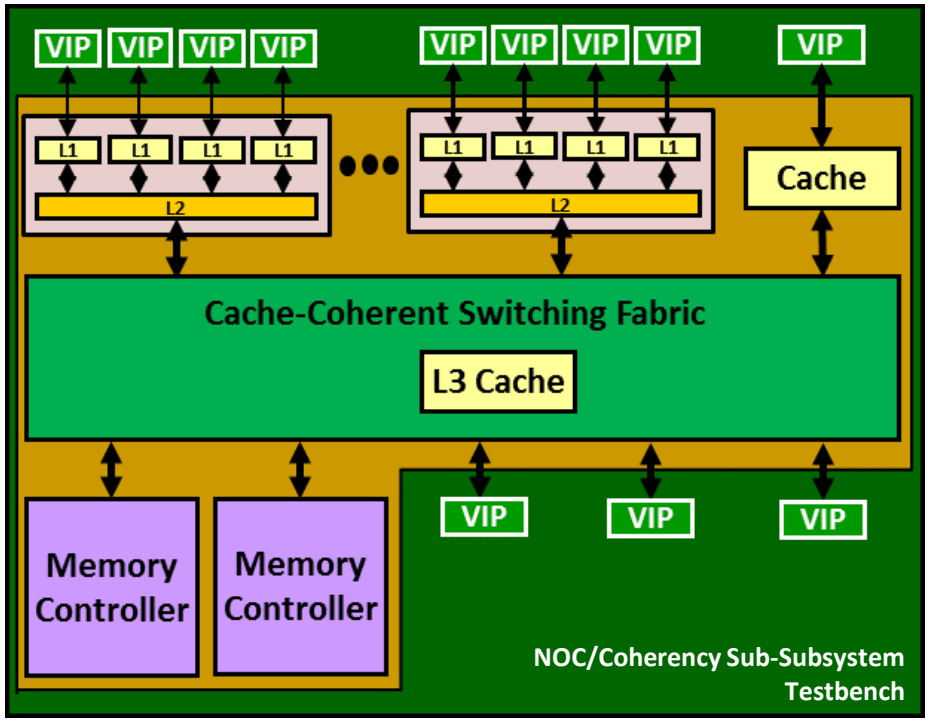
- Sequential Addresses matching a specific Hash

```
// memAllocAddrSlice allocated setId:0x1 of 0x4 blocks
// memAllocAddrHash hash:0x44 size:0x100 addr: trek_mem_ddr+0x08bc1100
// memAllocAddrHash hash:0x44 size:0x100 addr: trek_mem_ddr+0x08c01100
// memAllocAddrHash hash:0x44 size:0x100 addr: trek_mem_ddr+0x08c41100
// memAllocAddrHash hash:0x44 size:0x100 addr: trek_mem_ddr+0x08c81100
```

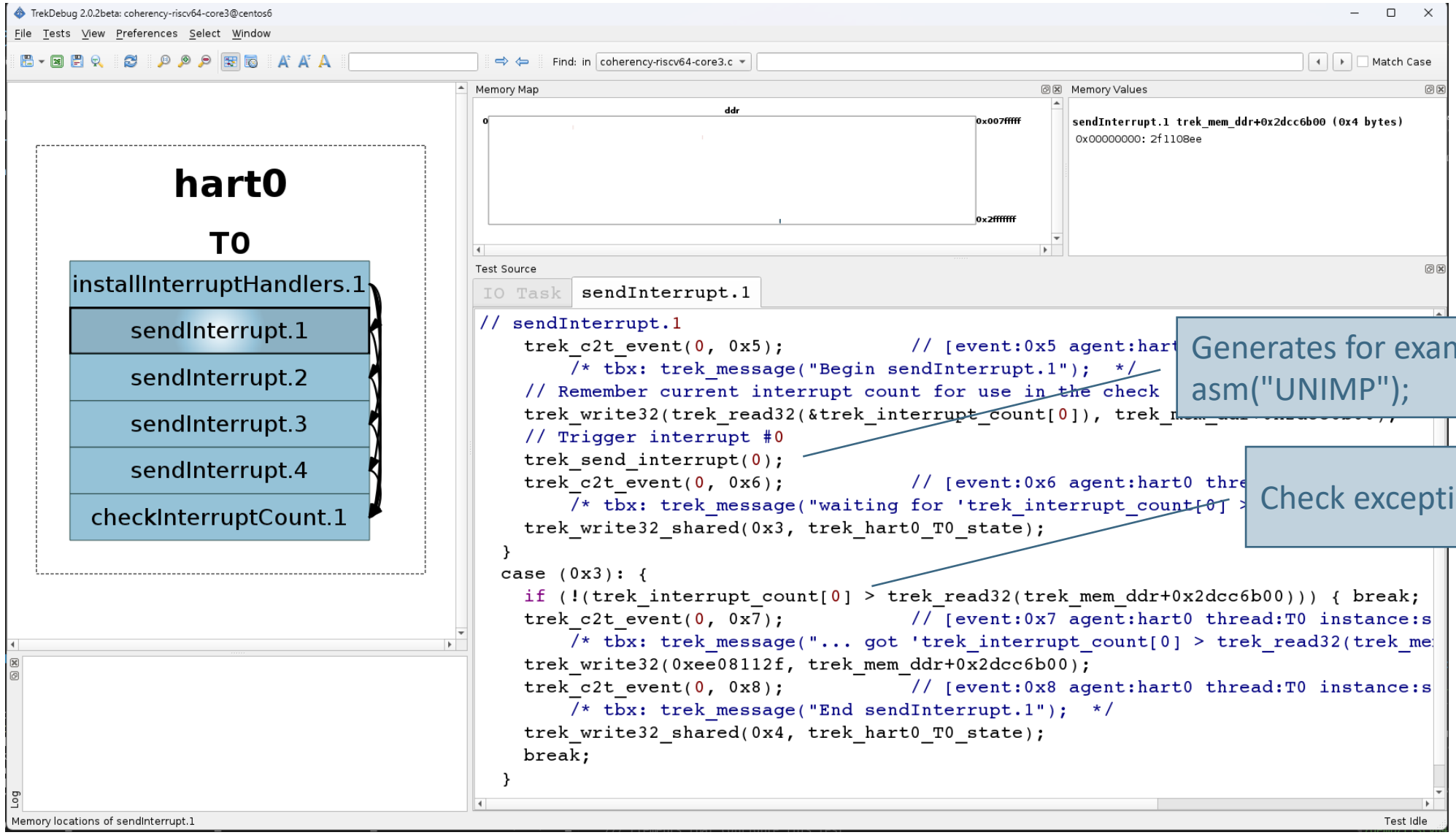
Application to Unit Bench and Sub-System Bench







RV64 Core Exception Testing



Generates for example, asm("UNIMP");

Check exception counts

Page Based Virtual Memory Tests

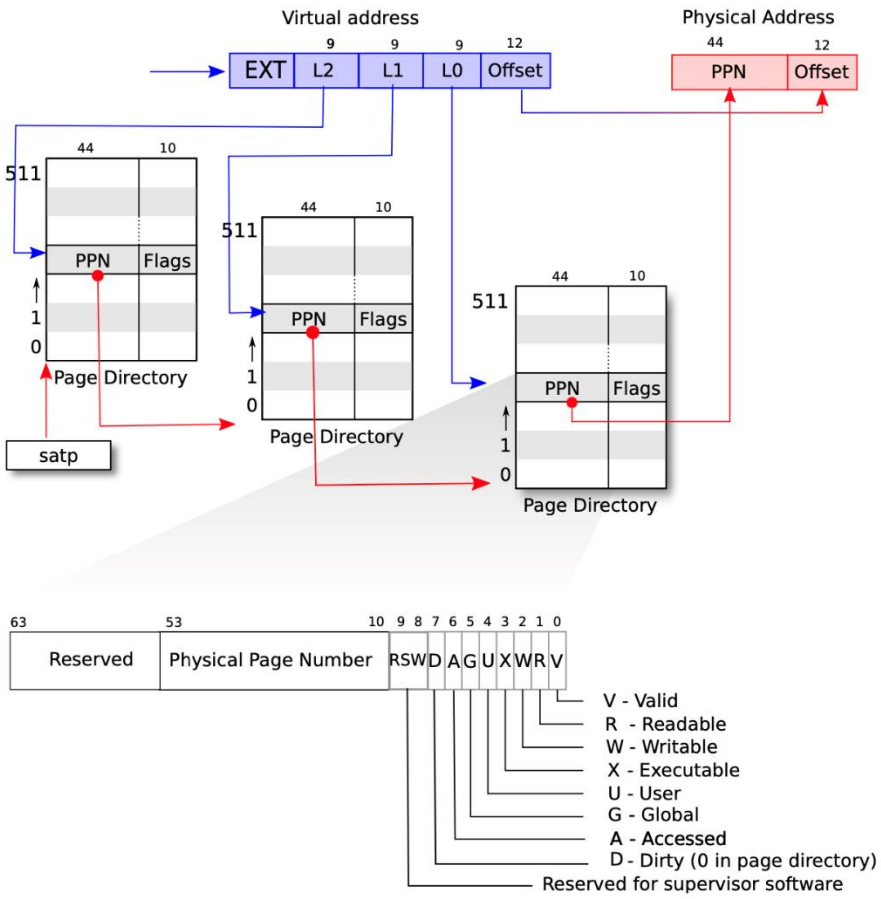


Figure 3.2: RISC-V address translation details.

RV64 Core Page Based MMU Tests

TrekDebug 2.0.2beta: coherency-riscv64-core4@centos6

File Tests View Preferences Select Window

Find: in coherency-riscv64-core4.c Match Case

multiOp.4
multiOp.5
multiOp.6
doCopy.3
doCheck.2
doCopy.4
checkSet.1
writeSet.1
multiOp.7
multiOp.8
swapOne.2
sFenceVMA.2
dekkerOp.2
dekkerCheck.2
writeSet.2

Memory Map

ddr
0 0x007fffff
0x2fffffff

Memory Values

swapOne.2 trek_mem_ddr+0xdd09000 (0x2000 bytes)
Before

0x00000000:	f8a5a11c	a0b95c2d	388a8a43	082f3264
0x00000010:	78fa7f77	d09c03bf	b82a209a	b87211f6
...	...	...	...	...
0x00001fe0:	f8e47c60	c1769a09	38d52197	a9e46b40
0x00001ff0:	78050aba	715a799b	b875ffe9	d9cb4ed2

After

Test Source

IO Task swapOne.2

```
// swapOne.2
    trek_c2t_event(0, 0x38);          // [event:0x38 agent:hart0 thread:T0 instance:s
        /* tbx: trek_message("Begin swapOne.2");  */
    /* Swapping Pages: trek_mem_ddr+0x0dd0a000 and trek_mem_ddr+0x0dd09000 */
    const trek_uint64_t addrA = trek_mem_ddr+0x0dd0a000ULL;
    const trek_uint64_t addrB = trek_mem_ddr+0x0dd09000ULL;
    // Find table entries for each address.
    trek_uint64_t* const pte1 = trek_find_pte(addrA);
    trek_uint64_t* const pte2 = trek_find_pte(addrB);
    const trek_uint64_t entry1 = *pte1;
    const trek_uint64_t entry2 = *pte2;
    // Insert the new table entries with addrA and addrB swapped.
    *pte1 = entry2;
    *pte2 = entry1;
    trek_c2t_event(0, 0x39);          // [event:0x39 agent:hart0 thread:T0 instance:s
        /* tbx: trek_message("End swapOne.2");  */
    trek_write32_shared(0x17, trek_hart0_T0_state);
    break;
  }
```

Swap MMU PTE's and Check memory access

Log

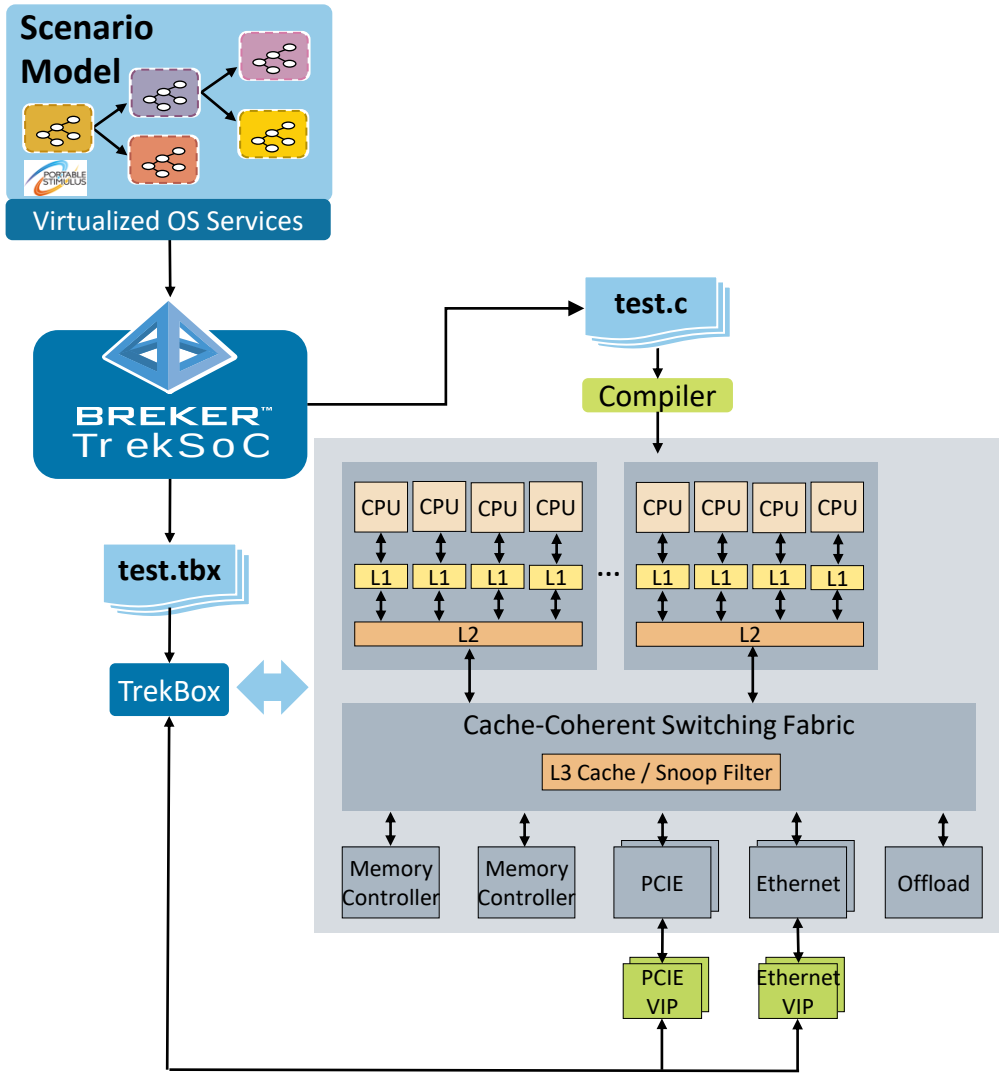
Memory locations of swapOne.2

Test Idle

Agenda

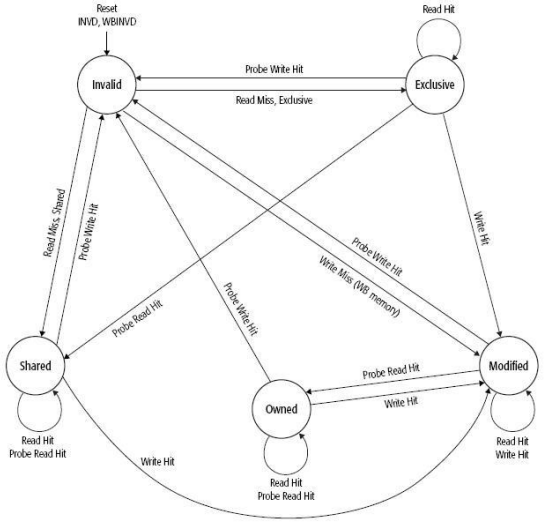
- Test Suite Synthesis and SystemVIP
- RISC-V Core Verification SystemVIP
- RISC-V SoC Verification SystemVIP

RISC-V SoC Testbench Integration



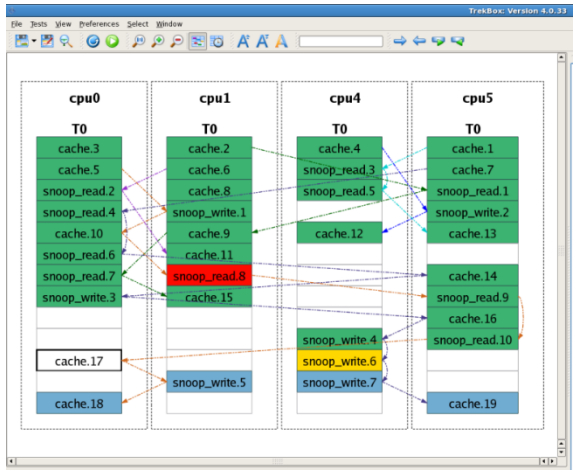
Multi-Agent Scheduling Plans: Overview

- True Sharing within scenario
- False Sharing across scenarios

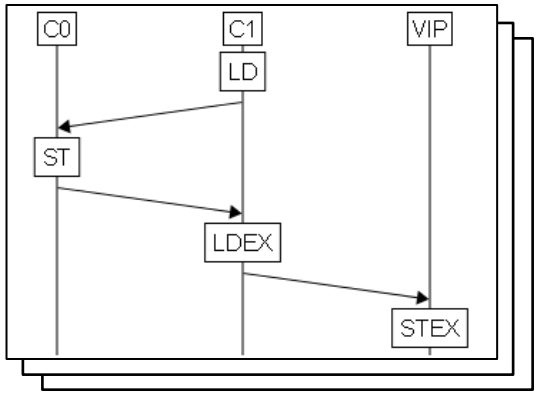


N Transition Sequences

Concurrent Scenario Test Case



N Transition Scenarios



Schedule Memory
Interleave & Pack
Resolve Dependencies

RV64 MultiCore MoesiStates

TrekDebug 2.0.2beta: coherency-riscv64-moesiStates@centos6

File Tests View Preferences Select Window

Find: in coherency-riscv64-moesiStates.c Match Case

hart0 T0 T1 | hart1 T0 T1 | hart2 T0 T1 | hart3 T0 T1

multiOp.1 multiOp.4 multiOp.5 multiOp.6 multiOp.7 multiOp.8 multiOp.9 multiOp.10 multiOp.11 multiOp.12 multiOp.13 multiOp.14 multiOp.15 multiOp.16 multiOp.17 multiOp.18 multiOp.19 multiOp.20 multiOp.21 multiOp.22 multiOp.23 multiOp.24 multiOp.25 multiOp.26 multiOp.27 multiOp.28 multiOp.29 multiOp.30 multiOp.31 multiOp.32 multiOp.33 multiOp.34 multiOp.35 multiOp.36 multiOp.37 multiOp.38 multiOp.39 multiOp.40 multiOp.41 multiOp.42 multiOp.43 multiOp.44 multiOp.45 multiOp.46 multiOp.47

Memory Map

ddr

Memory Values

multiOp.36 trek_mem_ddr+0xff95020 (0x4 bytes
0x00000000: 4b22f054
multiOp.36 trek_mem_ddr+0xff9503c (0x4 bytes
0x00000000: 4b22f054

Test Source

IO Task | multiOp.36

```
// multiOp.36
    trek_write32_shared(0x8, trek_hart0_T0_state);
  }
  case (0x8): {    // wait for multiOp.35
    if (trek_read32_shared(trek_hart3_T0_state) < 0x
    trek_c2t_event(0, 0xb);          // [event:0xb
        /* tbx: trek_message("Begin multiOp.36");  */
    // memAllocSingle with 0x1 blocks of 0x4 bytes
    // pss_top.moesiStates.EXCLUSIVE_to_MODIFIED_8_i / pss_top.op
    // trek_copy_memory_block(trek_mem_ddr+0x0ff9503c, trek_mem_
    trek_write32(trek_read32(trek_mem_ddr+0x0ff9503c), trek_mem_
    trek_c2t_event(0, 0xc);          // [event:0xc agent:hart0 t
        /* tbx: trek_message("End multiOp.36");  */
    trek_write32_shared(0x9, trek_hart0_T0_state);
    break;
  }
```

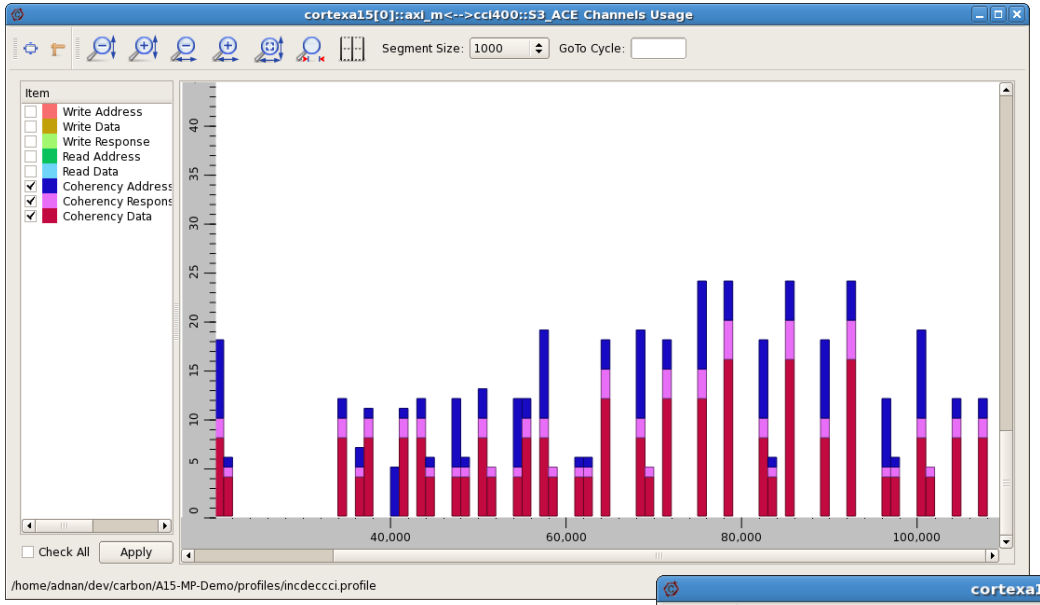
Log

Memory locations of multiOp.36

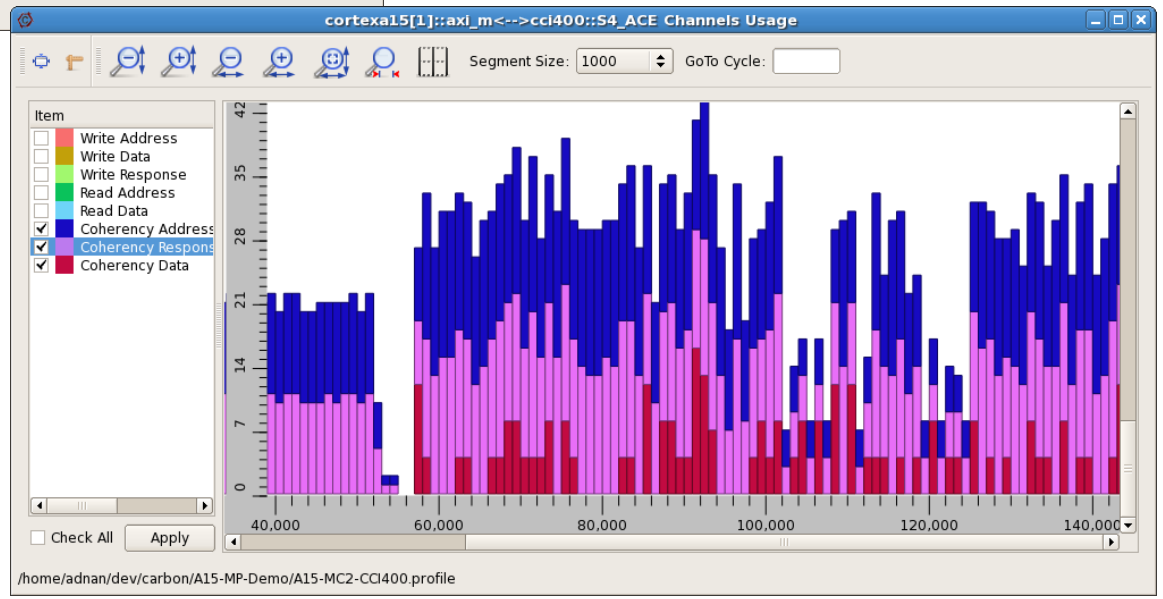
Test Idle

Planned Cache State Transitions

Efficacy of System-Integrity Testing using the RISC-V TrekApp



Typical directed coherency test …



… vs. RISC-V TrekApp automated Sys-Integrity tests

Atomics Testing

TrekDebug 2.0.2beta: coherency-riscv64-atomics@centos6

File Tests View Preferences Select Window

Find: in coherency-riscv64-atomics.c Match Case

Check result is aggregate of synchronized atomic operations

hart0 T0 T1 hart1 T0

Memory Map

ddr 0 0x007ffff 0x2fffffff

Memory Values

multiOp.19 trek_mem_ddr+0xff95038 (0x4 bytes
Before
0x00000000: b86b7386
After
0x00000000: a3c75b72

Test Source

IO Task multiOp.19

```
// multiOp.19
    trek_write32_shared(0x7, trek_hart1_T1_state);
  }
  case (0x7): {    // wait for sync tasks
    if ((trek_read32_shared(trek_hart2_T0_state) < 0x10) ||
        (trek_read32_shared(trek_hart3_T0_state) < 0x8) ||
        (trek_read32_shared(trek_hart3_T1_state) < 0xf)) {
      break;
    }
    trek_c2t_event(3, 0x4c);          // [event:0x4c agent:hart1
        /* tbx: trek_message("Begin multiOp.19");  */
    {
      trek_uint32_t *ptr  = (trek_uint32_t*)(trek_mem_ddr+0x0ff9!
      (void)__atomic_fetch_xor(ptr, 0xf428ac1b, __ATOMIC_SEQ_CST
    }
    trek_c2t_event(3, 0x4d);          // [event:0x4d agent:hart1
        /* tbx: trek_message("End multiOp.19");  */
    trek_write32_shared(0x8, trek_hart1_T1_state);
    break;
  }
```

Log

Memory locations of multiOp.19

Test Idle

RISC-V SoC Memory Ordering: Dekker Algorithm

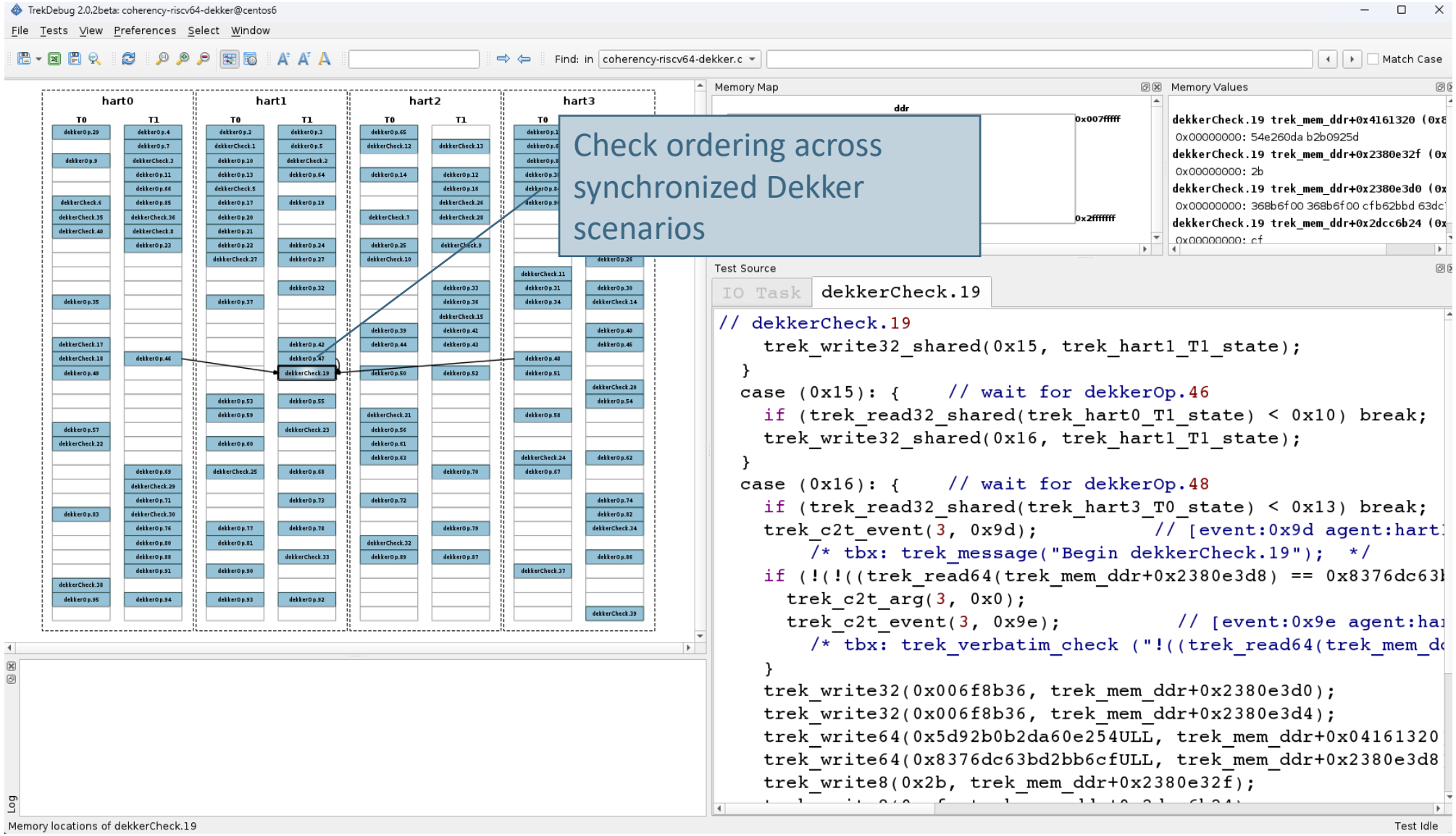
- Assume initial state A=0 , B=0

- The Dekker Algorithm States

```
core 0: ST A, 1; MEM_BARRIER; LD B
core 1: ST B, 1; MEM_BARRIER; LD A
error iff ( A == 0 && B == 0 )
```

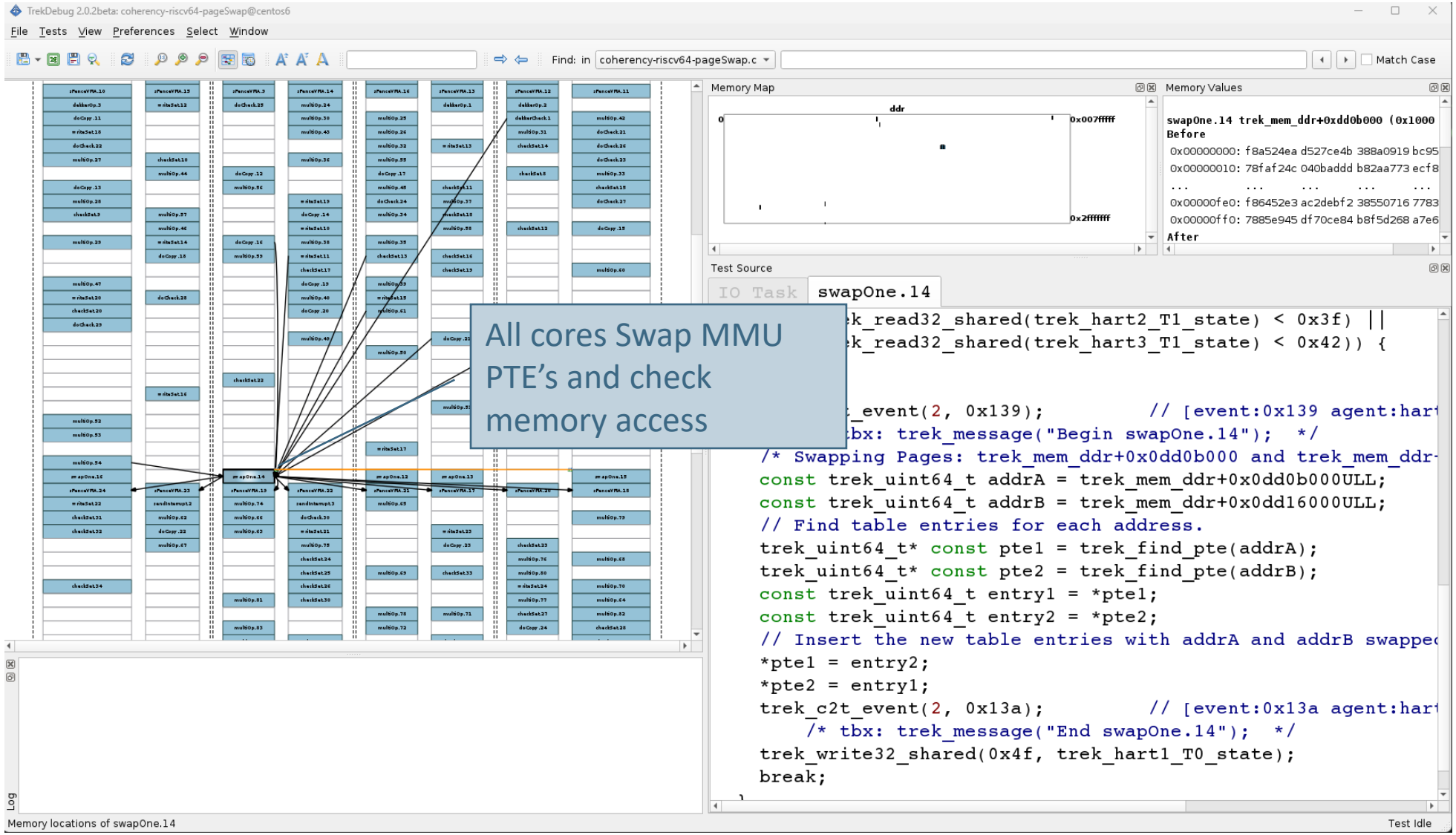
- This is a test for a weakly ordered memory system
  - Such a system must preserve the property that a LD may not reorder ahead of a previous ST from the same agent

Dekker Memory Ordering



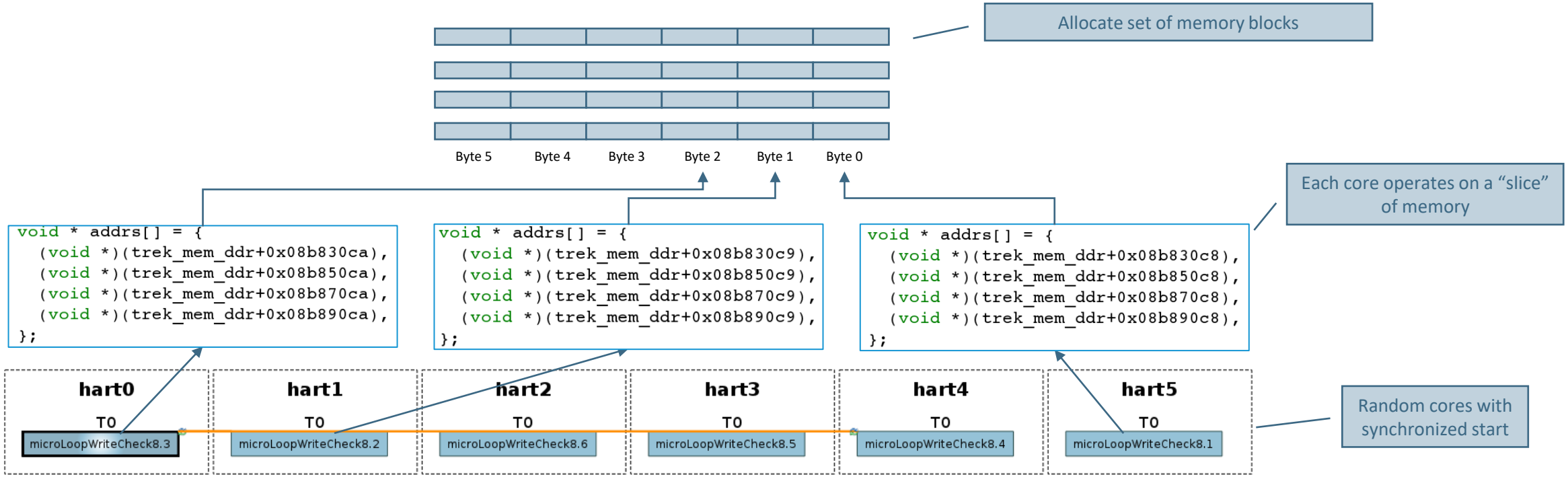
Check ordering across synchronized Dekker scenarios

MultiCore MMU Tests



All cores Swap MMU PTE's and check memory access

False-Share Memory Stress Tests




```
int trek_microloop_write_check8( void * addrs[], int count, trek_uint8_t pattern){
  int errorCount = 0;
  int ii;
  for ( ii = 0; ii < count; ++ii){
    trek_write8(pattern, addrs[ii]);
  }
  for ( ii = 0; ii < count; ++ii){
    if (trek_read8(addrs[ii]) != pattern) {
      ++errorCount;
      trek_runtime_error("trek_microloop_write_check8", addrs[ii], pattern, trek_read8(addrs[ii]));
    };
  }
  return errorCount;
}
```

```
for ( int ii = 0; ii < 1000; ++ii ){
  errorCount += trek_microloop_write_check8(addrs, 4, 184);
}
```

Each core has free running loop

Thanks for Listening!
Any Questions?