BREKER™ *The Leader in Portable Stimulus*

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION CHINA

An advanced method for RISC-V load-store and register hazard verification, leveraging PSS

Adnan Hamid, Breker Verification Systems

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

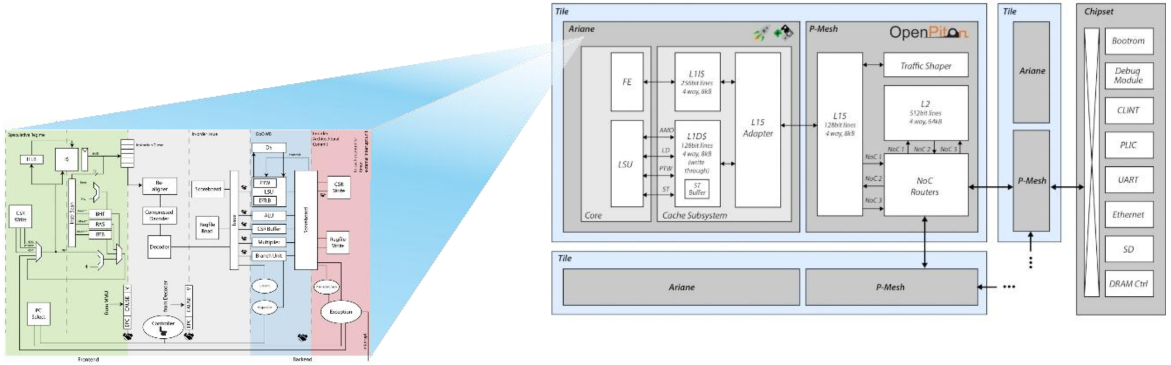
RISC-V Verification “Stack”

Test Complexity

- Performance/power profiling
- SW Execution, OS Boot
- System integration integrity
- Core operation integrity
- Micro-architecture functionality
- ISA compliance
- Up & running “Hello World”

Different Challenges for Core vs SoC Verification

BREKER™



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

RISC-V Verification & Validation Tasks

Core Integrity

Complexity

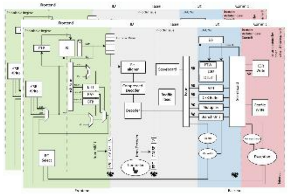
- Interrupts/Paging/Memory Order
- Micro-architecture functionality
- ISA compliance
- First Instruction Completion

IP Integrity

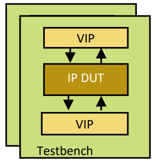
Complexity

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

CPU Cores



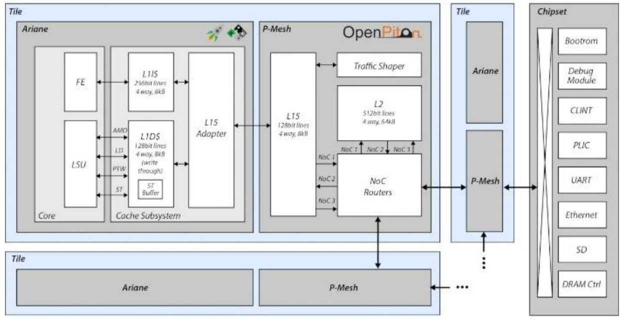
Uncore IPs



SoC Integrity

Complexity

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



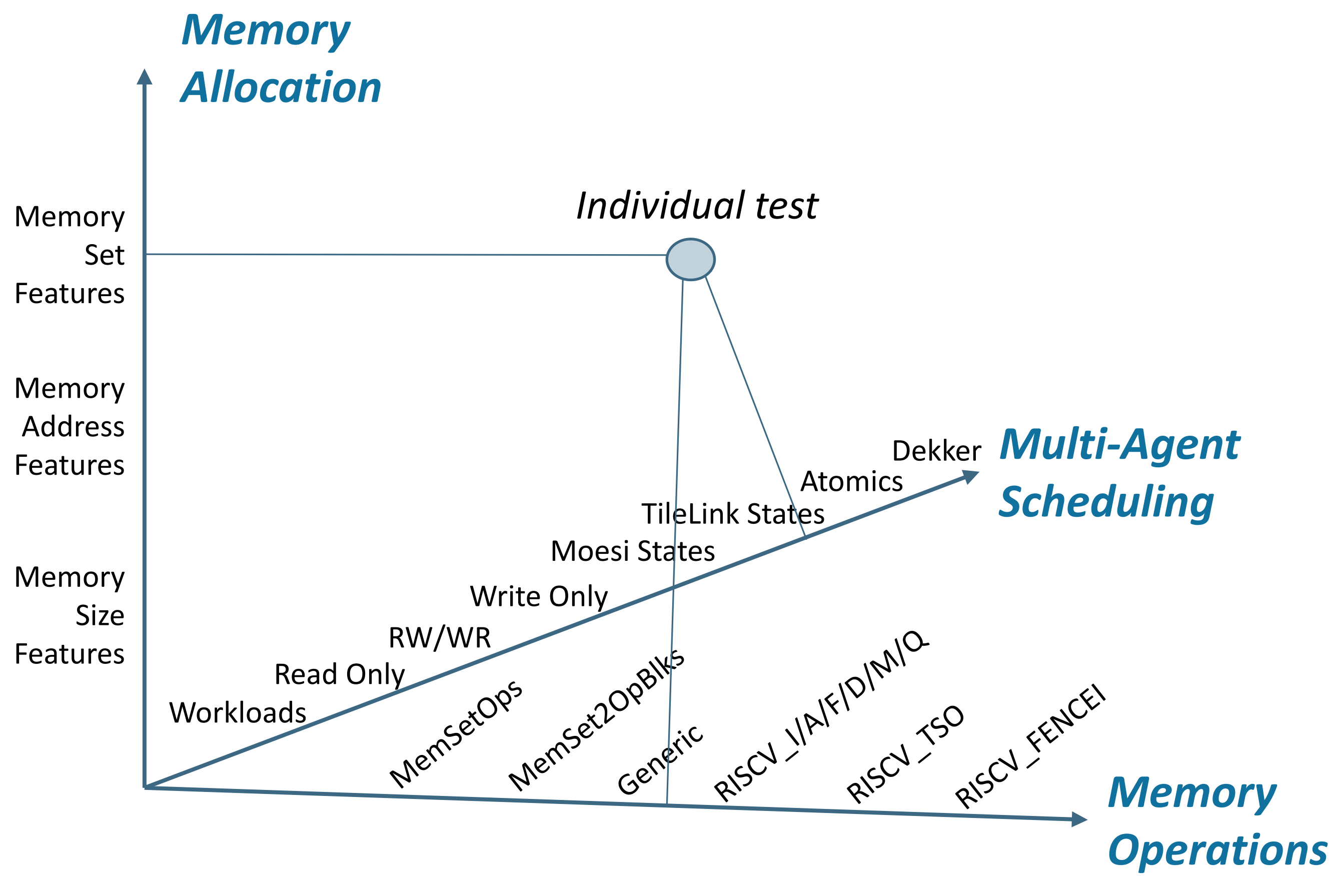
Firmware

Firmware Integrity

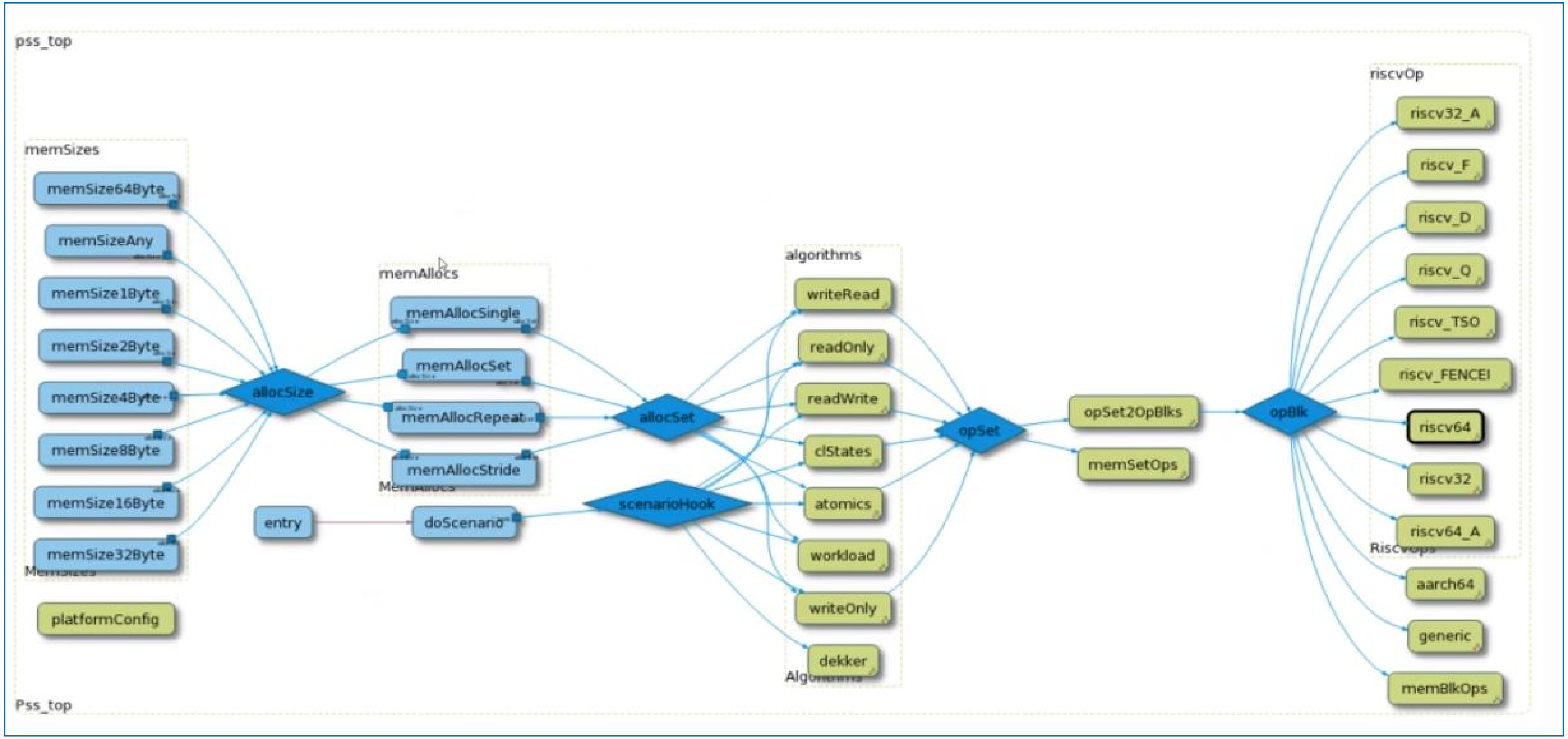
Complexity

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

Activity Categories for Load Store Verification



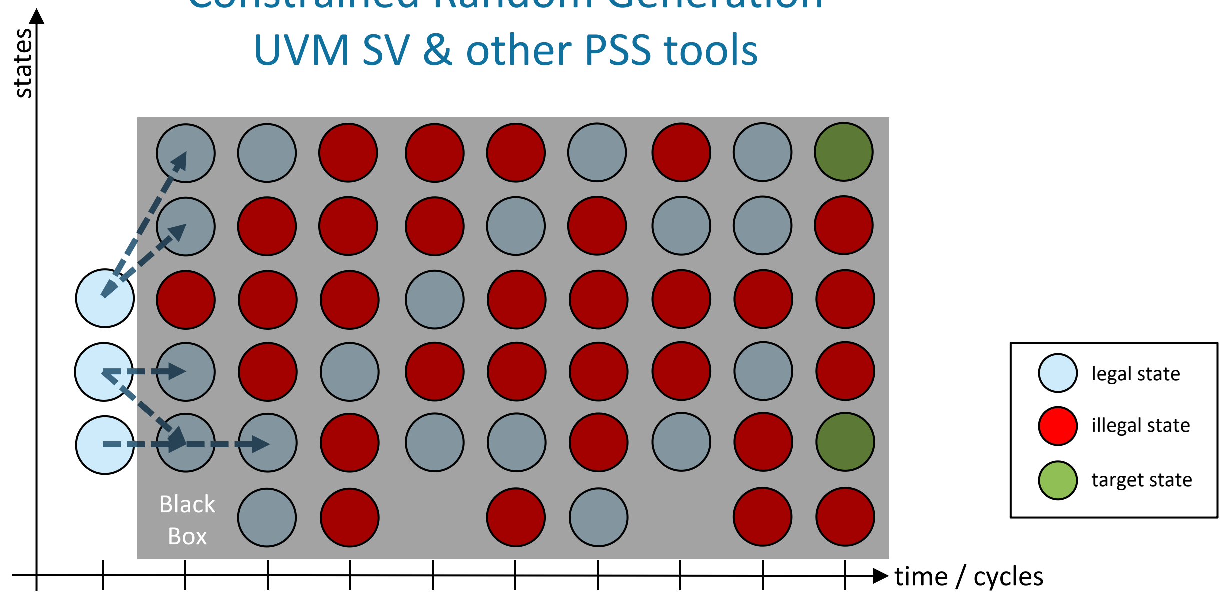
Activities coded as a graph



Constrained Random vs AI Planning Algorithm Synthesis

BREKER™

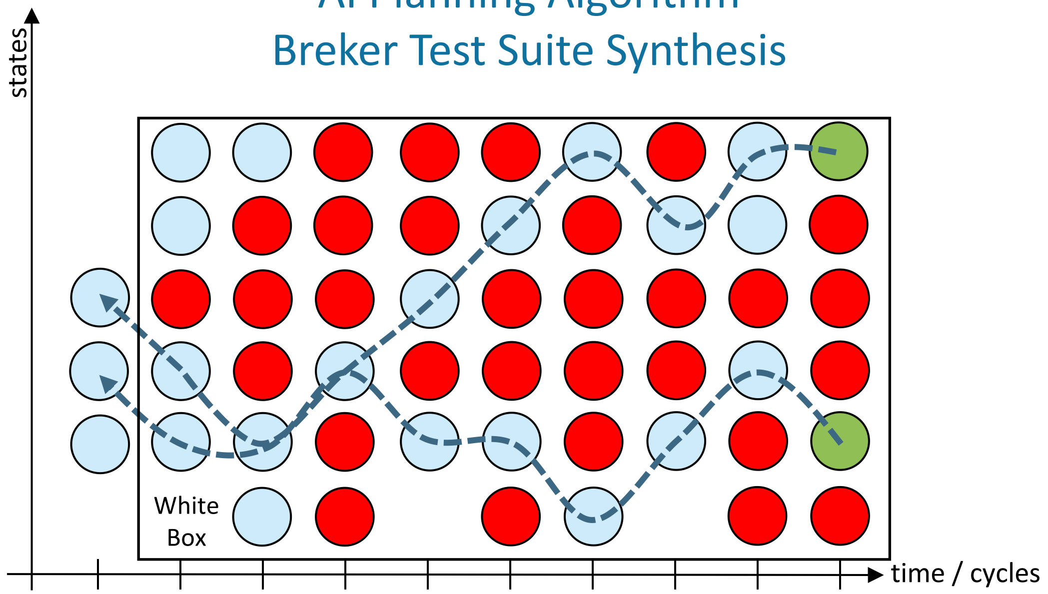
Constrained Random Generation
UVM SV & other PSS tools



Design black box, shotgun tests to search for key state

Low probability of finding complex bug

AI Planning Algorithm
Breker Test Suite Synthesis



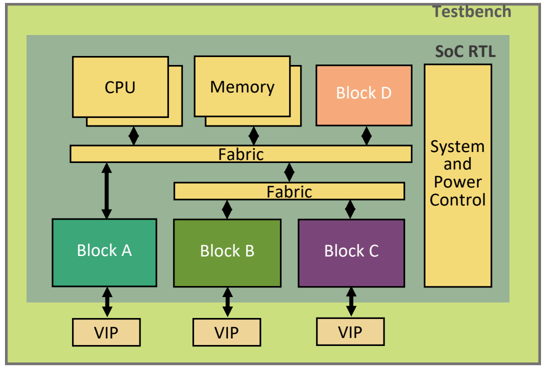
Starts with key state and intelligently works backward through space

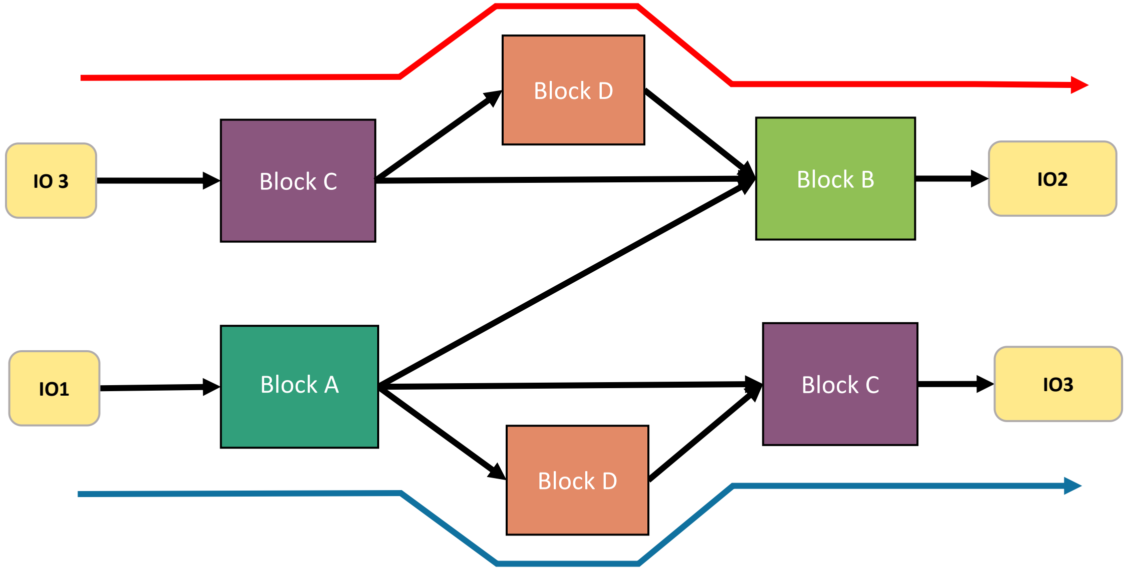
Deep sequential, optimized test discovers complex corner-cases

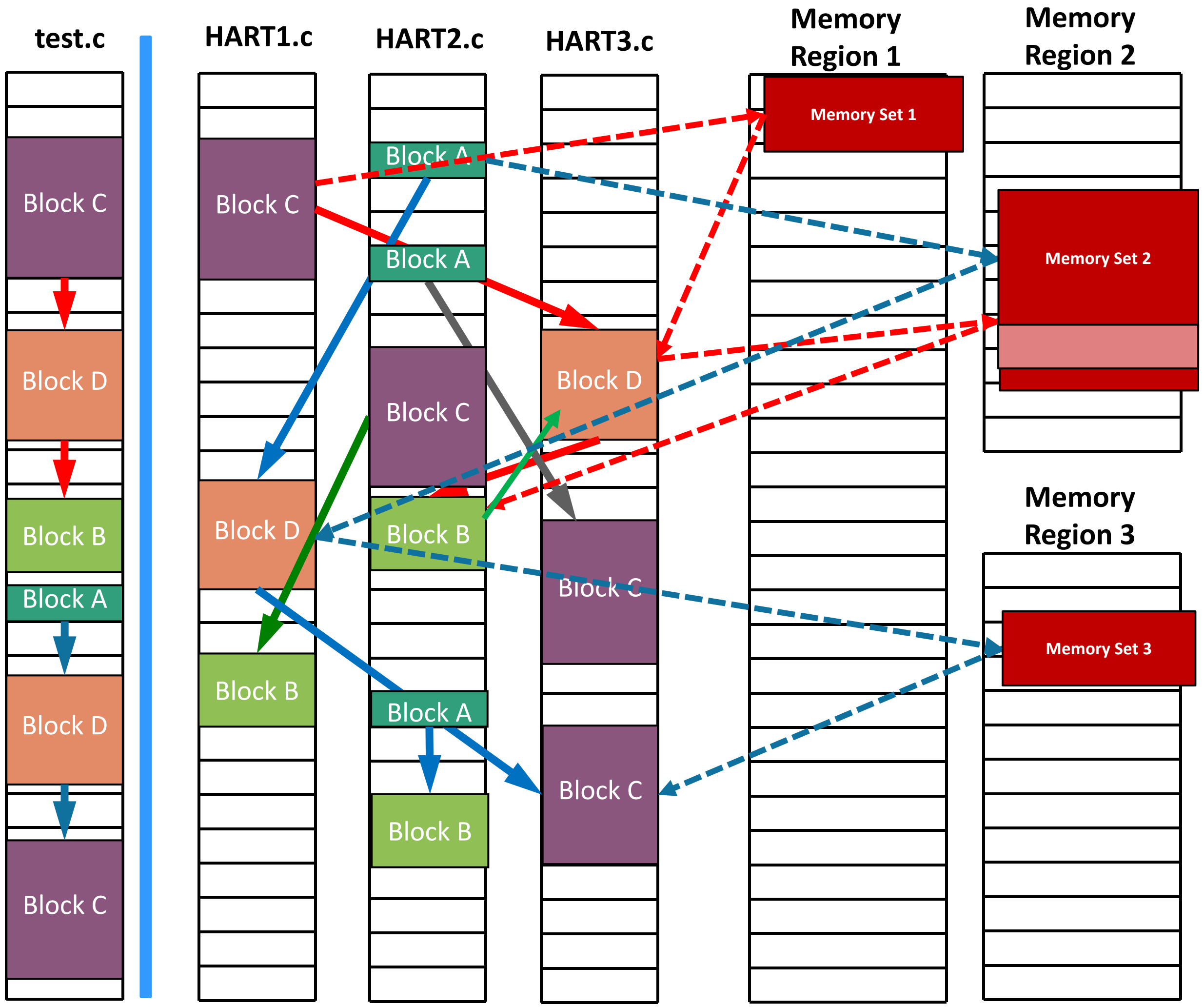
IBM

White Paper Discussing AI Planning Algorithm Test Generation on Breker Website

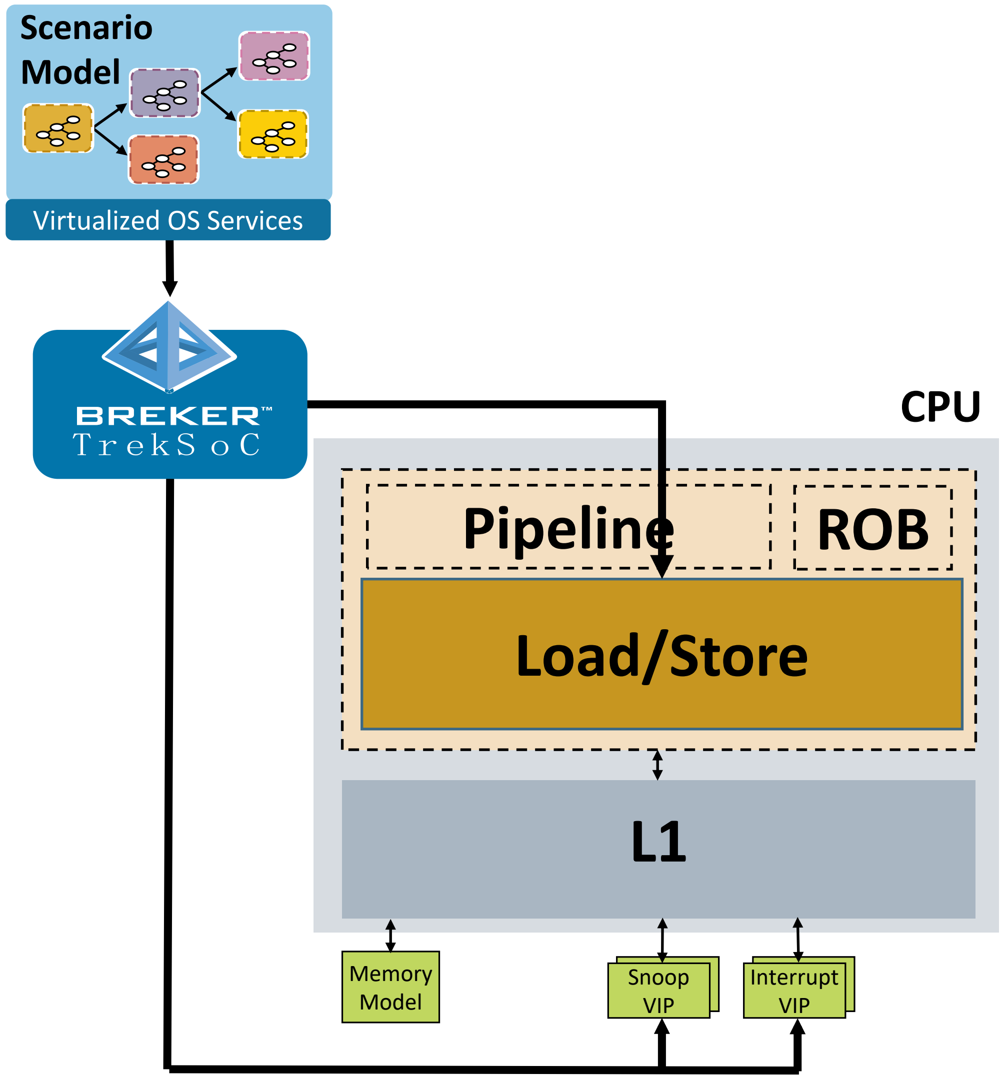
Concurrent Test Content to Torture Load Store



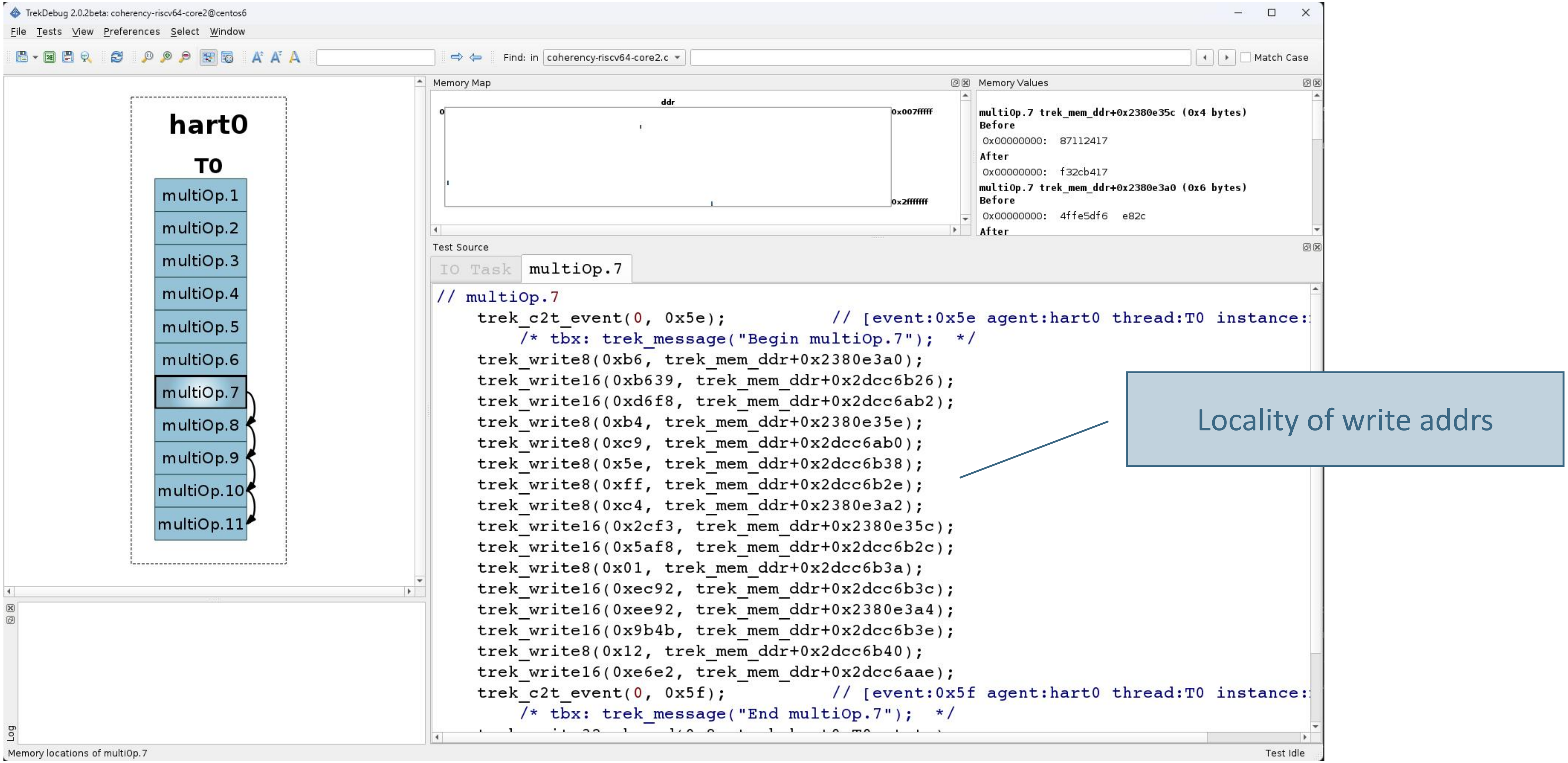




Load/Store is 80% of High Performance CPU Design



RV64 Core Load/Store



Example Address Allocation Patterns

BREKER™

- 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

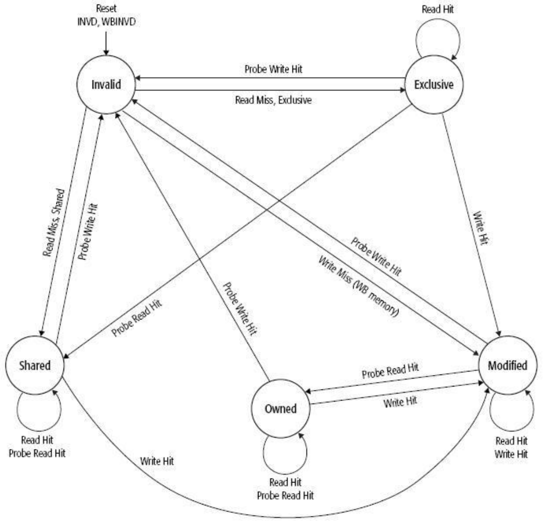
```
// memAllocAddrSlice allocated setId:0x1 of 0x4 blocks
// memAllocAddrStride stride_len:0x2000 size:0x8 addr: trek_mem_ddr+0x08b830c8
// memAllocAddrStride stride_len:0x2000 size:0x8 addr: trek_mem_ddr+0x08b850c8
// memAllocAddrStride stride_len:0x2000 size:0x8 addr: trek_mem_ddr+0x08b870c8
// memAllocAddrStride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

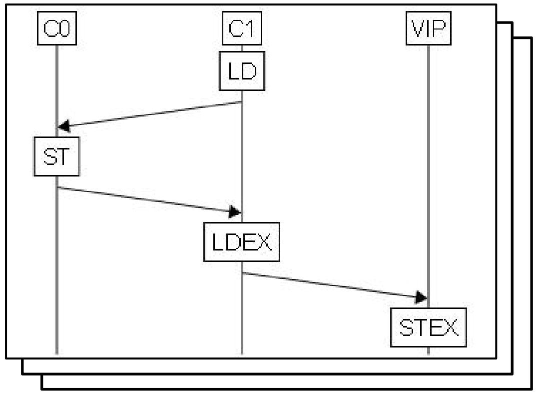
Multi-Agent Scheduling Plans: Overview

- True Sharing within scenario
- False Sharing across scenarios

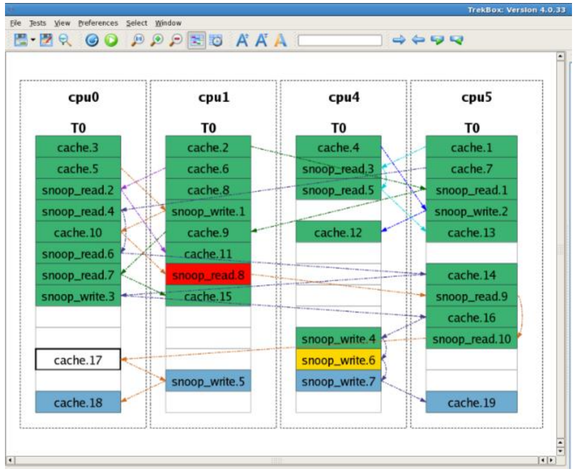


N **Transition Sequences**

N Transition Scenarios

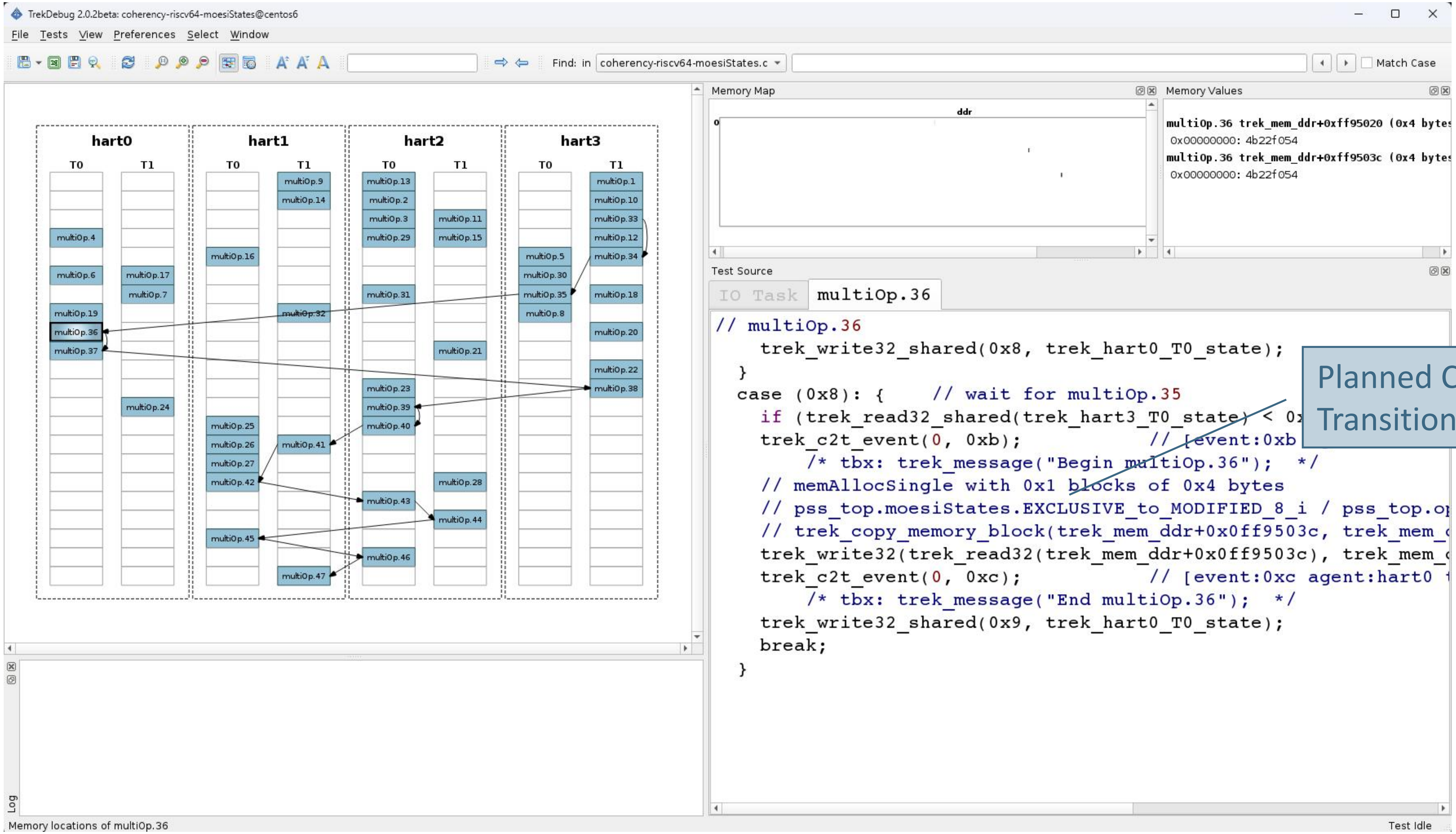


Concurrent Scenario Test Case



Schedule Memory Interleave & Pack
Resolve Dependencies

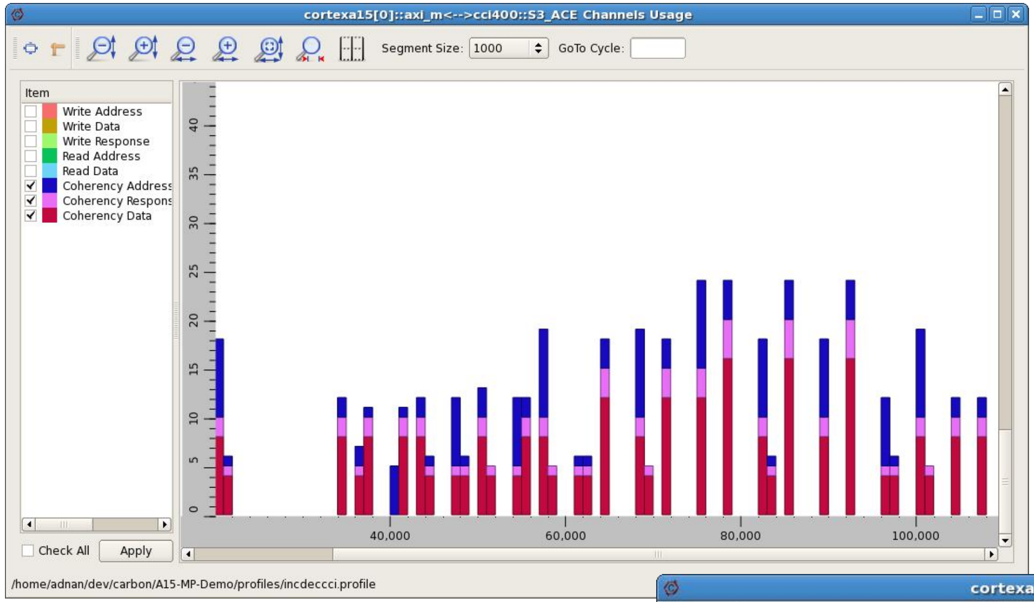
RV64 MultiCore MoesiStates



Planned Cache State Transitions

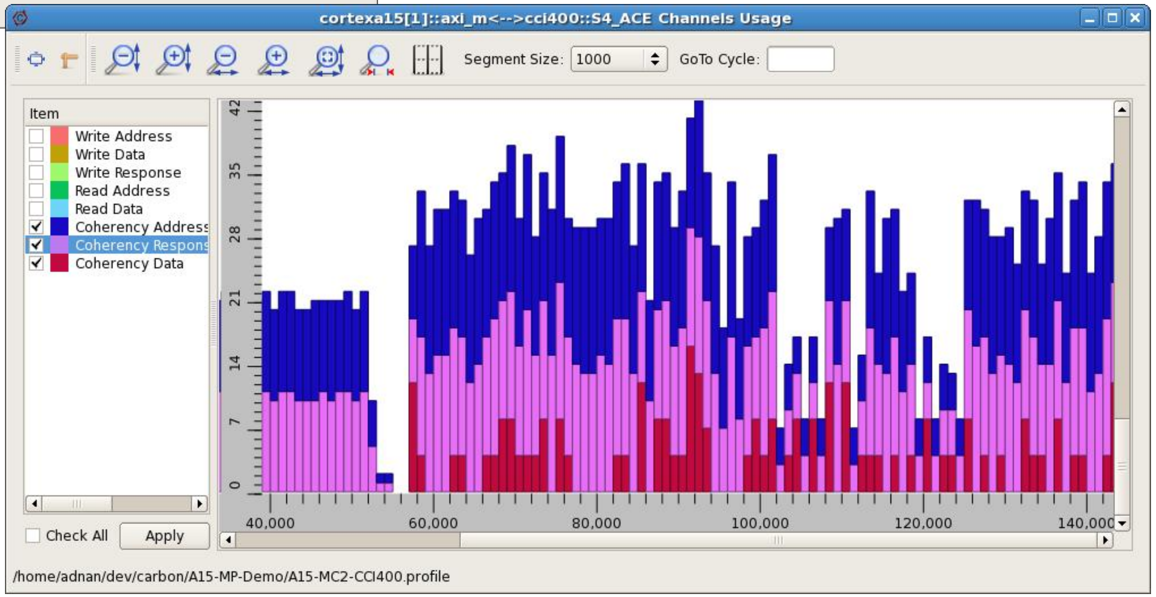
Efficacy of System-Integrity Testing using the RISC-V TrekApp

BREKER™

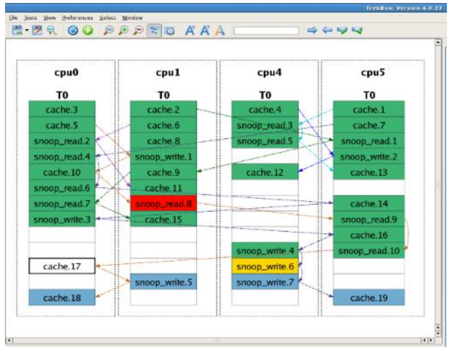


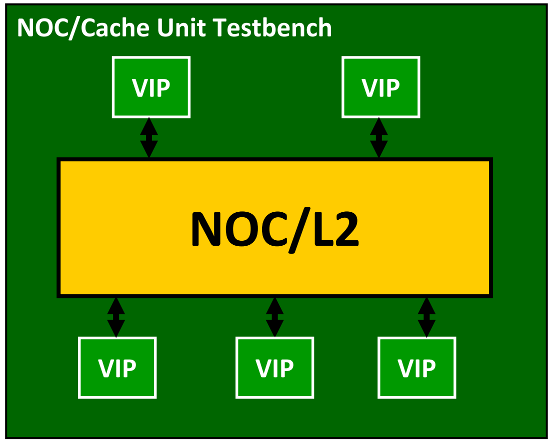
Typical directed coherency test …

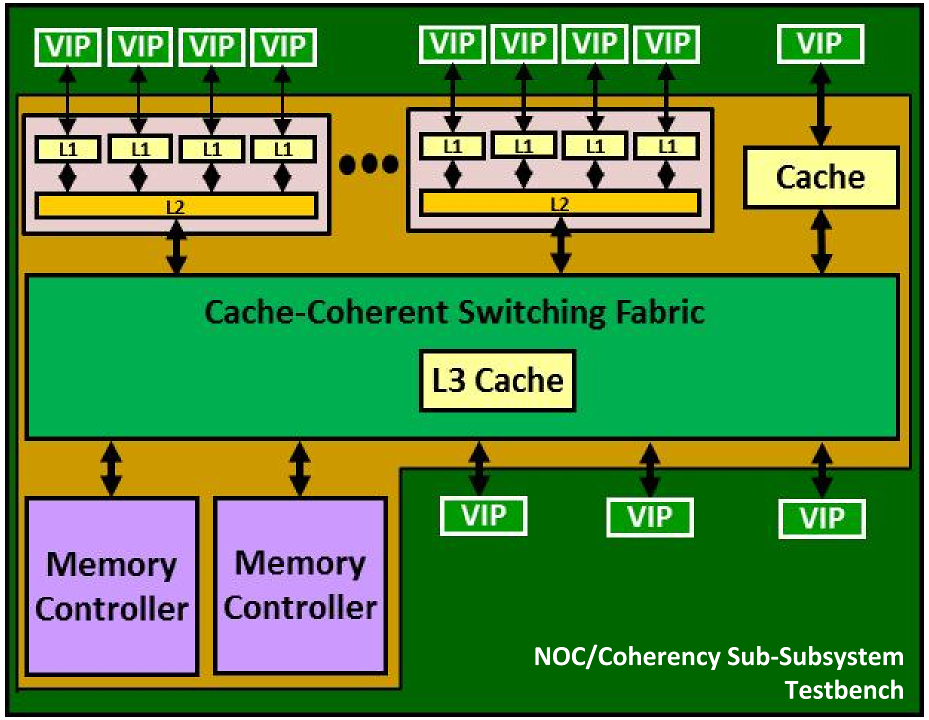
… vs. RISC-V TrekApp automated Sys-Integrity tests



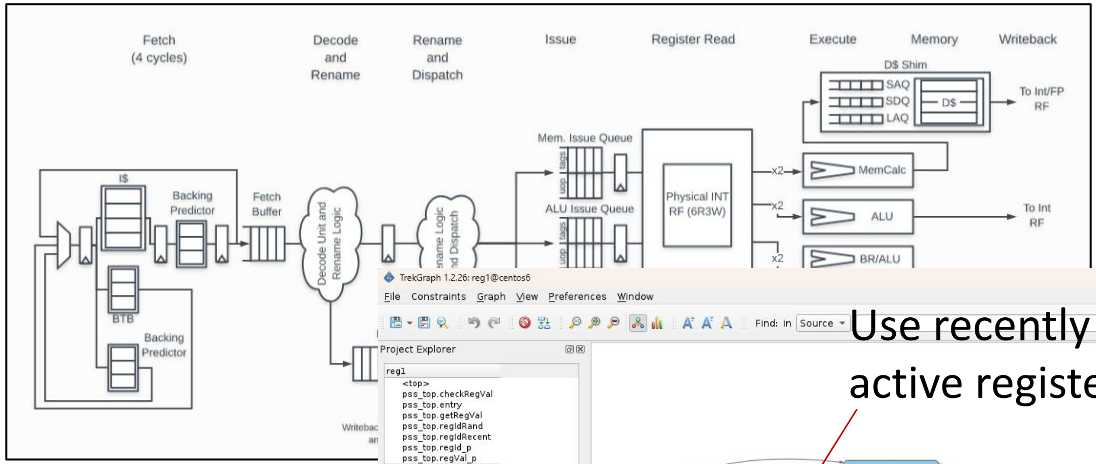
Application to Unit Bench and Sub-System Bench



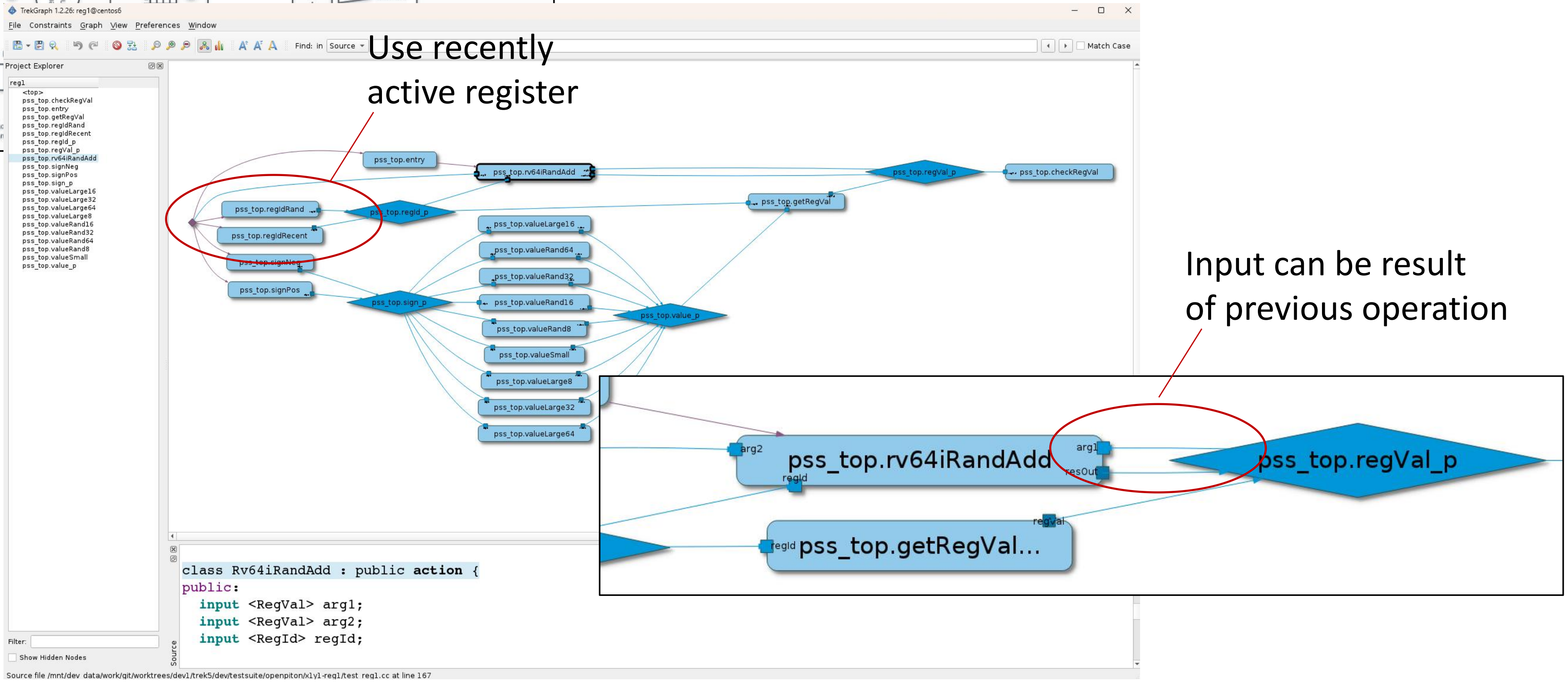




RISC-V Register Hazards Model



Hazard between read from stage N and write from stage N+m



Use recently active register

Input can be result of previous operation

RISC-V SoC Memory Ordering: Dekker Algorithm

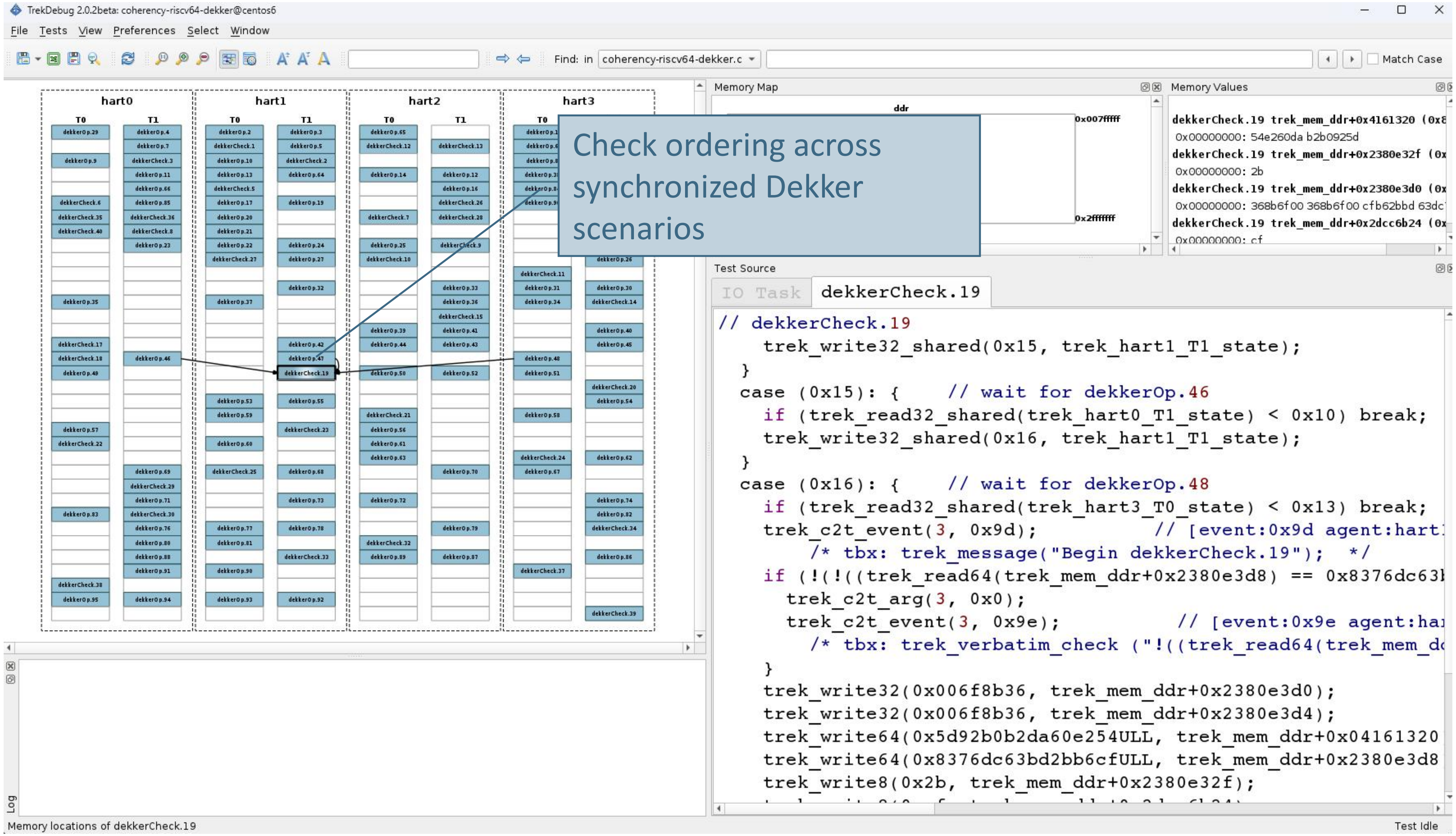
- Assume initial state A=0 , B=0

- The Dekker Algorithm States

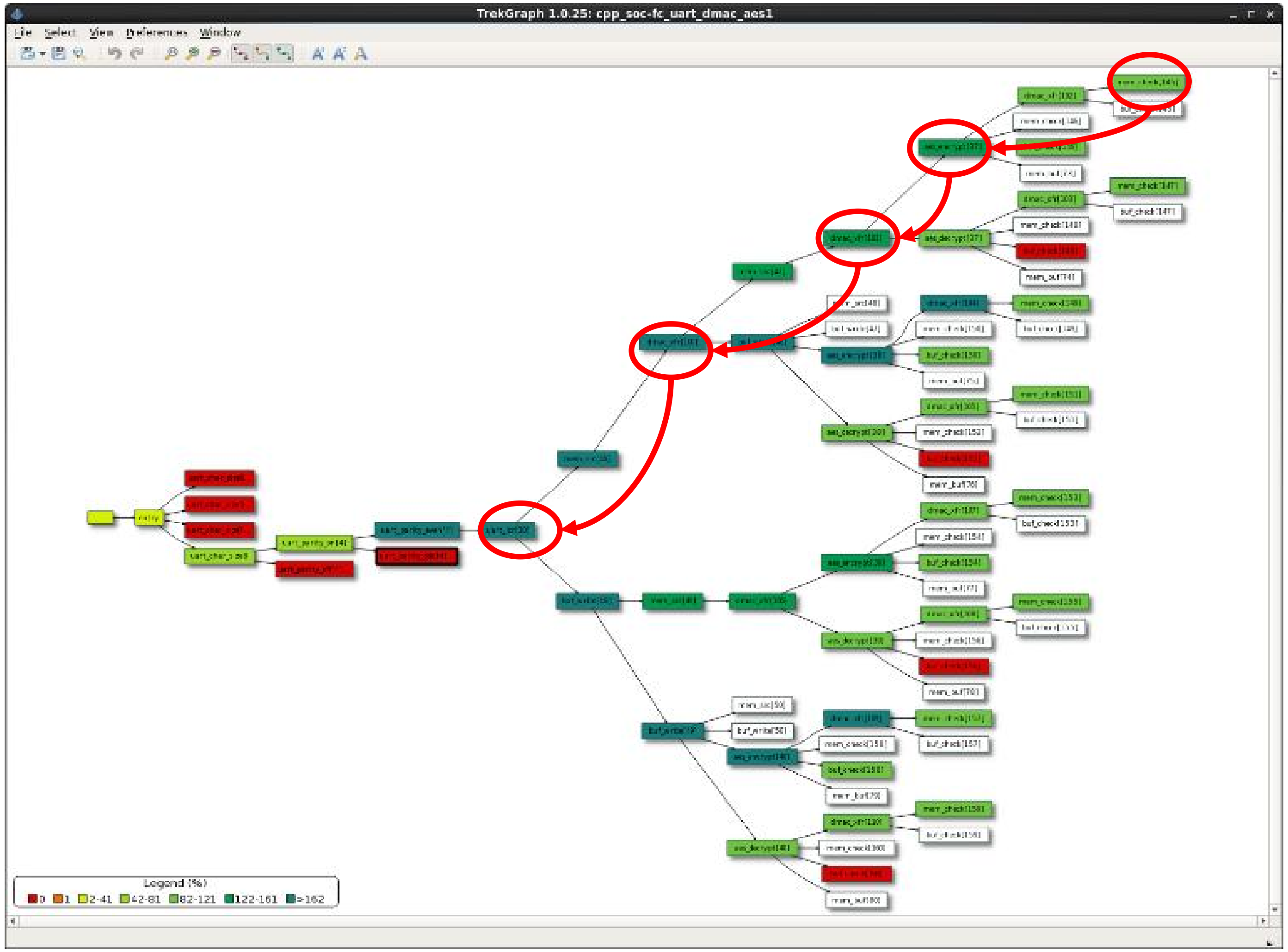
```
core 0: ST A, 1; MEM_BARRIER; LD B
core 1: ST B, 1; MEM_BARRIER; LD A
error iff ( A == 0 && B == 0 )
```

- 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

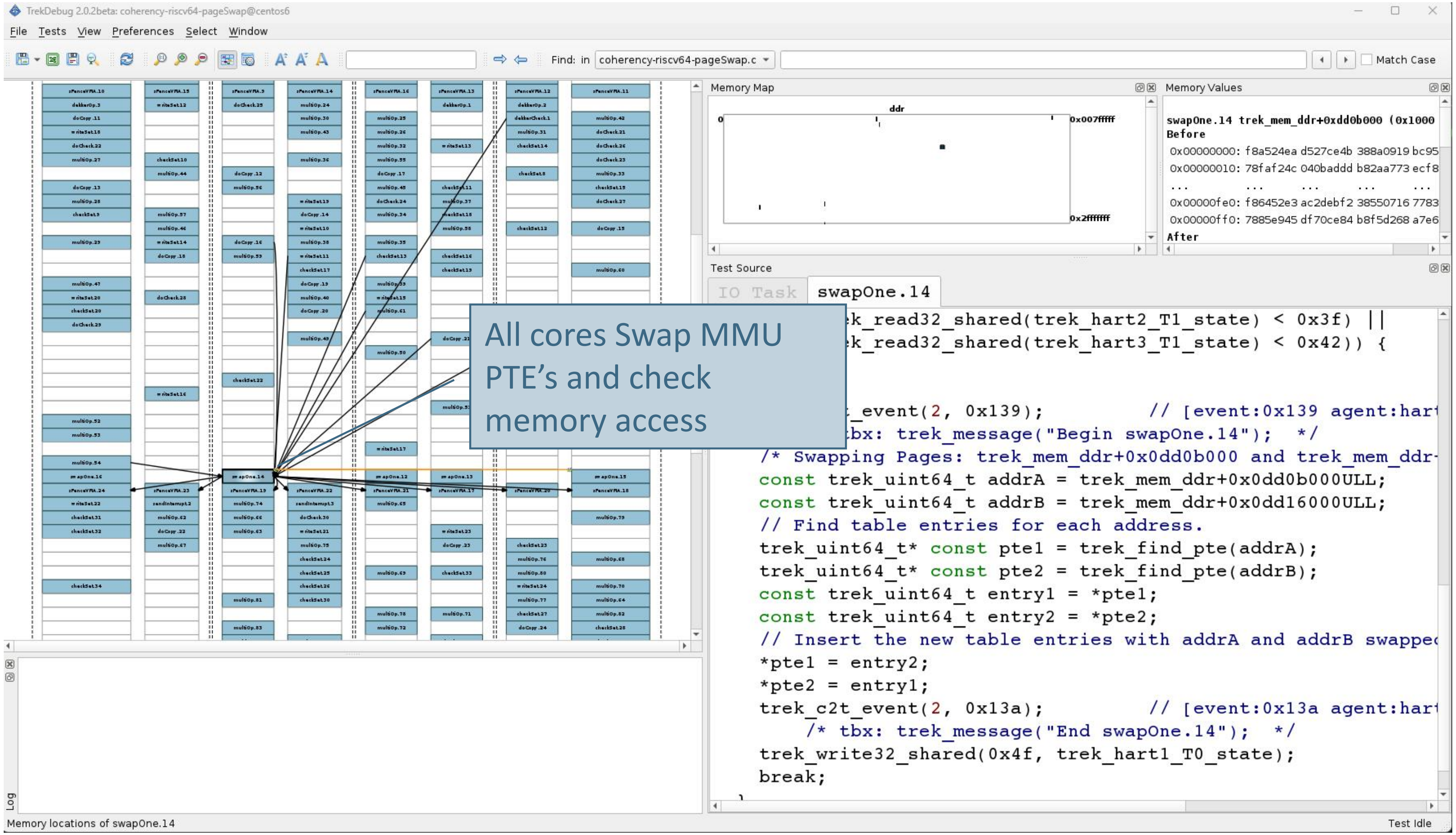
Dekker Memory Ordering



Graph Based Scenario Coverage



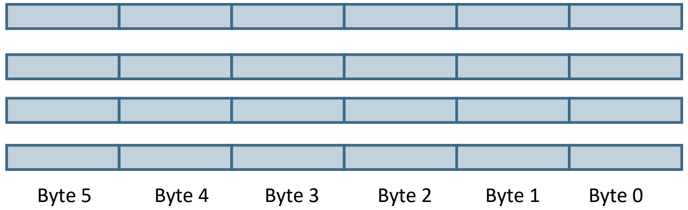
MultiCore MMU Tests



False-Share Memory Stress Tests

BREKER™

Allocate set of memory blocks

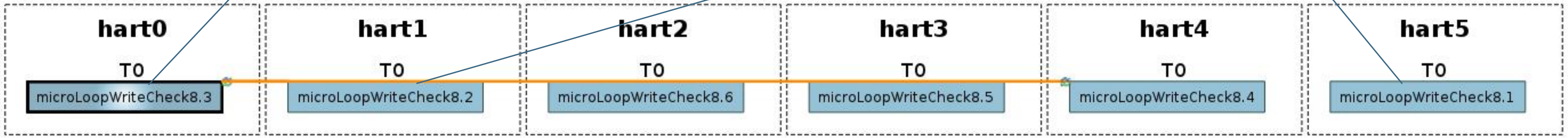


Each core operates on a "slice" of memory

```
void * addrs[] = {
  (void *)(trek_mem_ddr+0x08b830ca),
  (void *)(trek_mem_ddr+0x08b850ca),
  (void *)(trek_mem_ddr+0x08b870ca),
  (void *)(trek_mem_ddr+0x08b890ca),
};
```

```
void * addrs[] = {
  (void *)(trek_mem_ddr+0x08b830c9),
  (void *)(trek_mem_ddr+0x08b850c9),
  (void *)(trek_mem_ddr+0x08b870c9),
  (void *)(trek_mem_ddr+0x08b890c9),
};
```

```
void * addrs[] = {
  (void *)(trek_mem_ddr+0x08b830c8),
  (void *)(trek_mem_ddr+0x08b850c8),
  (void *)(trek_mem_ddr+0x08b870c8),
  (void *)(trek_mem_ddr+0x08b890c8),
};
```



Random cores with synchronized start

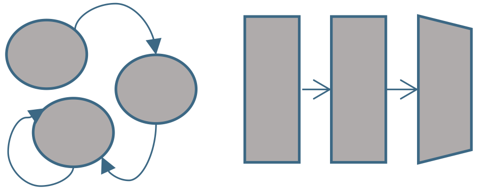
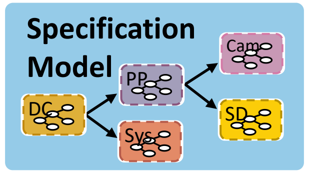
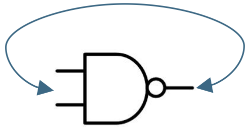
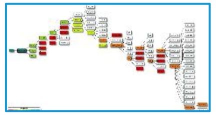
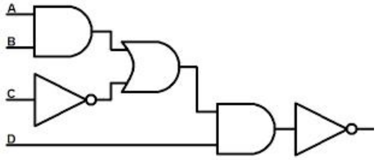
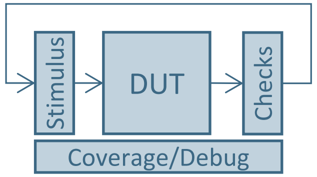
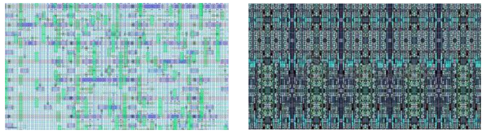
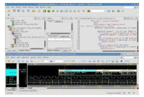
```
int trek_microloop_write_check8( void * addrs[], int count, trek_uint8_t pattern){
  int errorCount = 0;
  int ii;
  for ( ii = 0; ii < count; ++ii){
    trek_write8(pattern, addrs[ii]);
  }
  for ( ii = 0; ii < count; ++ii){
    if (trek_read8(addrs[ii]) != pattern) {
      ++errorCount;
      trek_runtime_error("trek_microloop_write_check8",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

Test Suite Synthesis… Analogous to Logic Synthesis

BREKER™

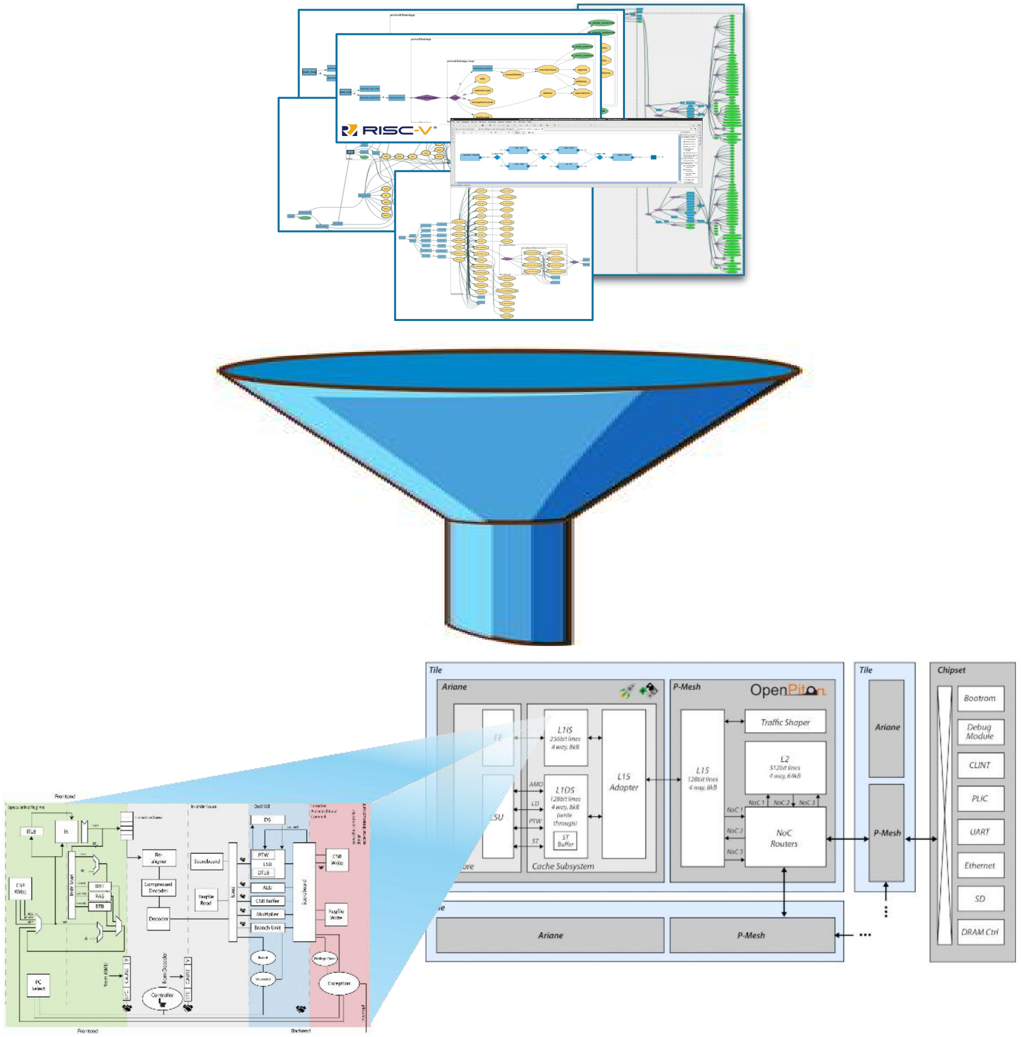
Design Synthesis		Test Suite Synthesis
	Describe intent	
Constrain		Constrain
 *Timing/Area Constraints*	Specify goals	 *Coverage Constraints*
Synthesize		Synthesize
	Generate implementation	
Optimize		Optimize
	Map to platform	***UVM***  ***SoC***  ***Silicon*** 

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

Thanks for Listening!
Any Questions?