



What is a “System”?

- **System Specification**

**collection of
interacting
behaviors**

- **System Mapping**

**collection of
interacting
resources**

- **System Architecture**



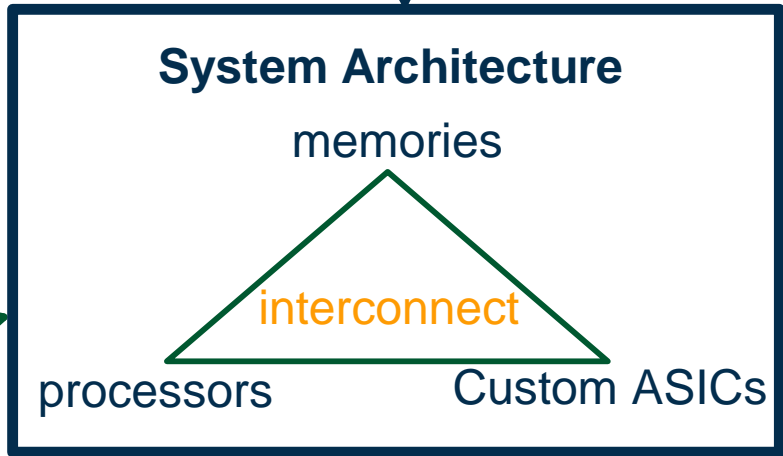
What is a “Computer System”?

Classically more about creativity than restriction in:

- System Specification
- System Mapping
- System Architecture

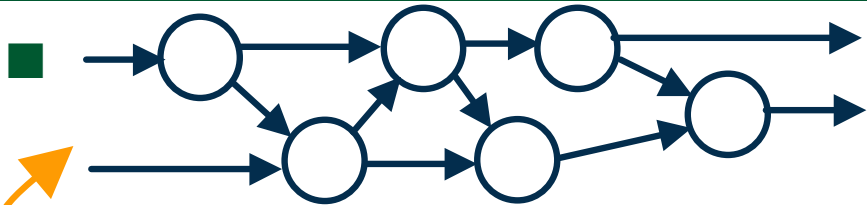


Lots of this!





The “It’s All Really HW Even If It Uses a CPU and Memory” Approach



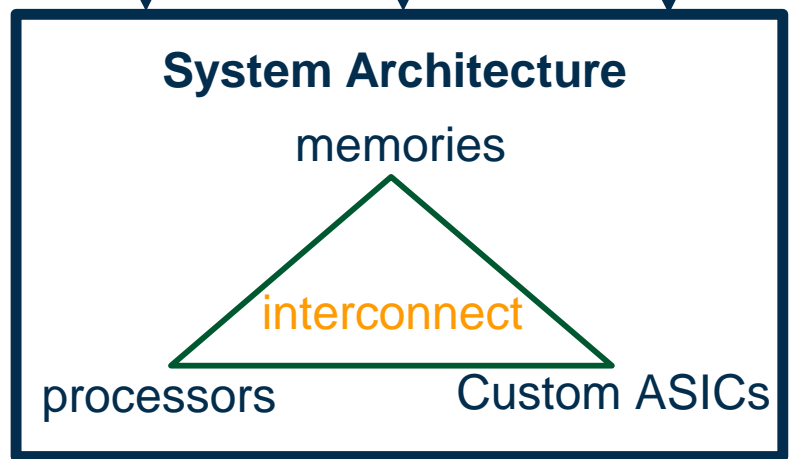
SW, HW Synthesized

Architecture Spec “Up Front”

Unified Modeling Environment

Behavioral Software Domain

Behavioral Hardware Domain



- The hardware designer’s view of system design, reactive systems, not data-intensive
 - Software is limited, sometimes for verification purposes (boundedness)
 - looks a lot like HW models executing on a processor
 - does not handle shared memory



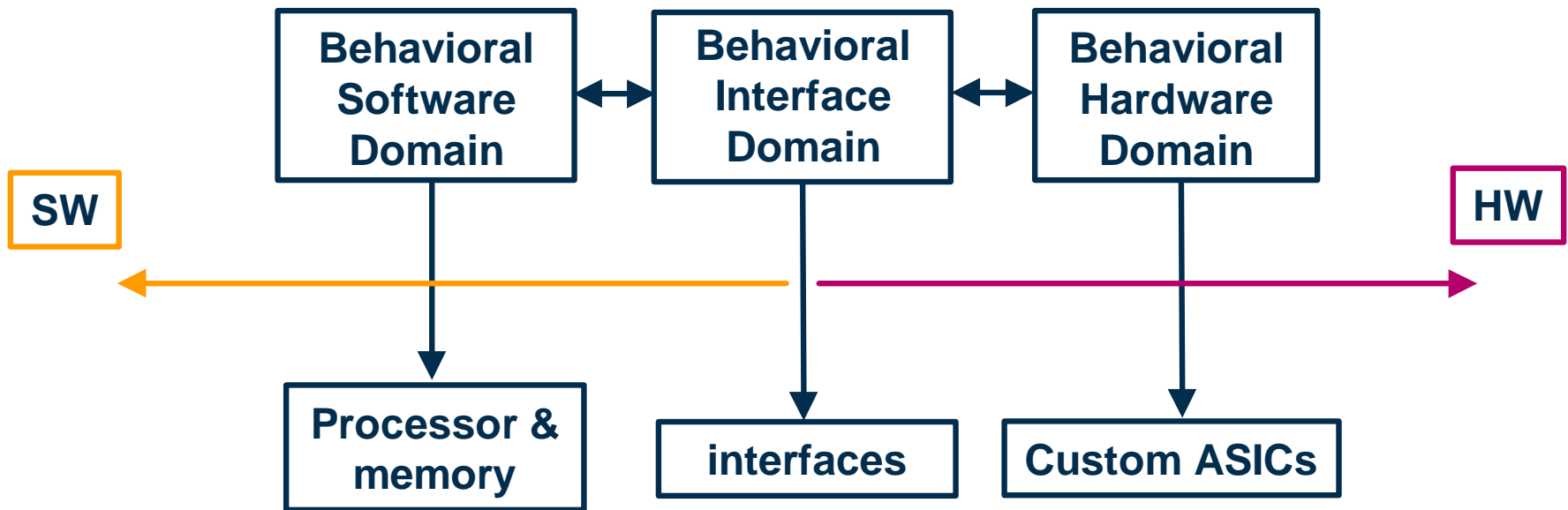
Just Do It -- All

<i>HW, Unbounded Structural Resource Model</i>	<i>SW, Unbounded Memory Model</i>
co-specified behavior/resources	dynamic, addressable, de-referencable memory
synchronous global state update	global state addressability
synchronous state advancement	time independence
implicit state updates via modular interconnect	explicit state updates via compiled library function calls
time-based simulation	interactive, operational input
fan-out/fan-in	unbounded buffers, mutexes, semaphores
threads always eligible for execution	dynamic threads
feedback	recursion

If You Can't Model *All* of This Behavior, You Can't Call It "Codesign"



Another Approach



- **H/S Behaviors Pre-Partitioned**

- so system architecture pre-determined also
 - often limited to one processor, unithreaded software

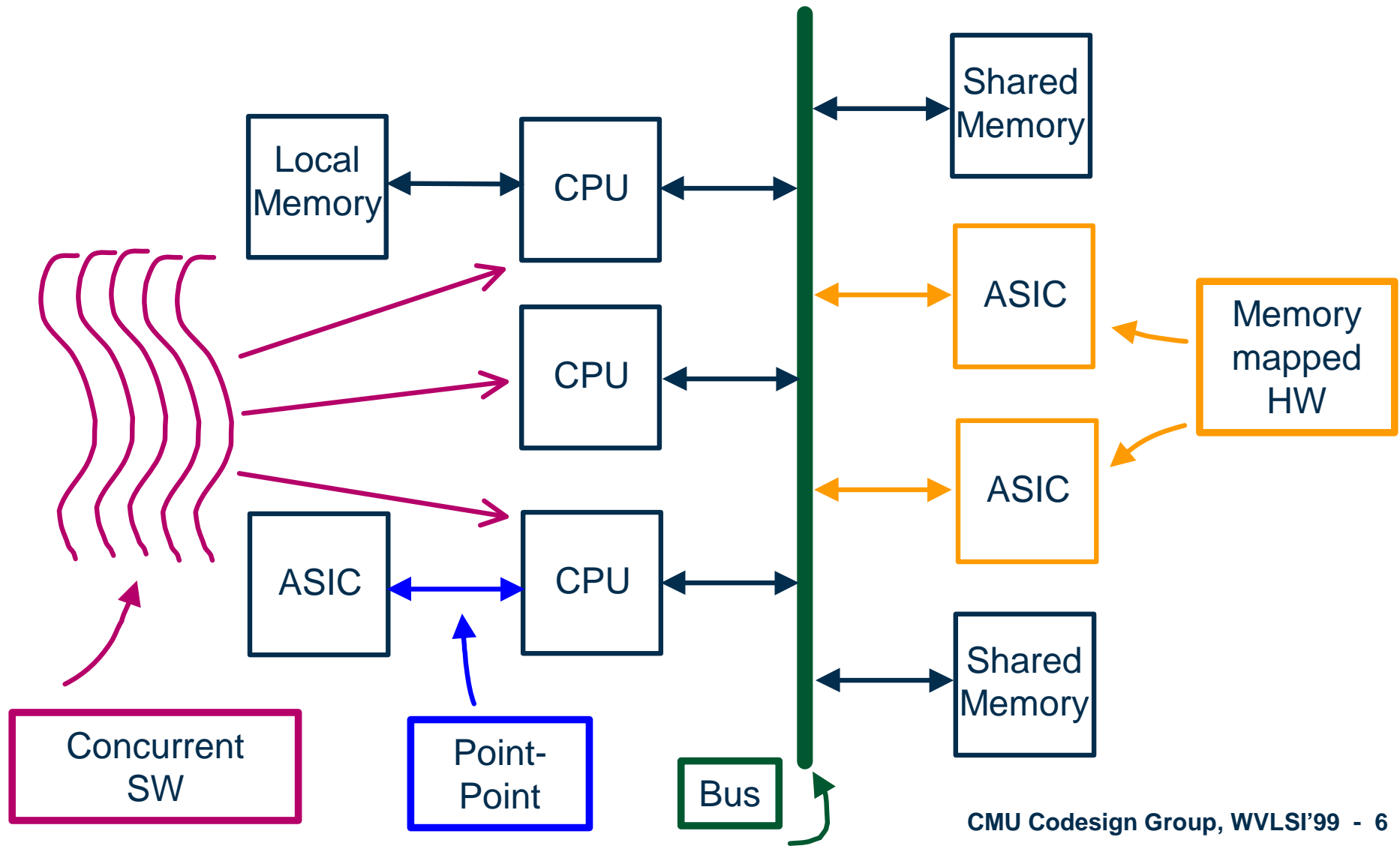
- **Multi-Level Interfaces Described as Objects**

- using a third domain

- isn't there a more natural way to describe H/S interfaces?

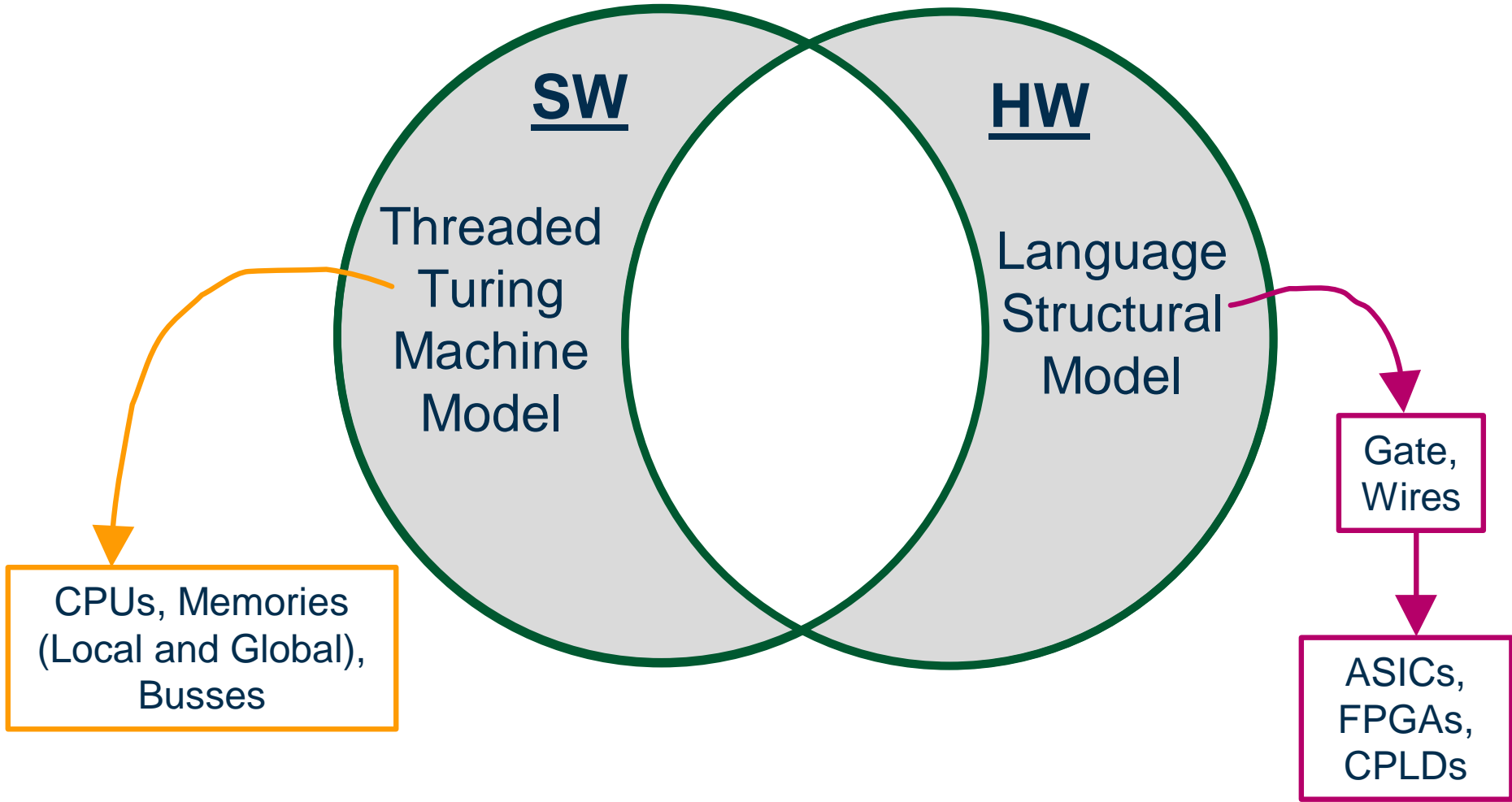


Heterogeneous Components and Interconnect: Where is the H/S?



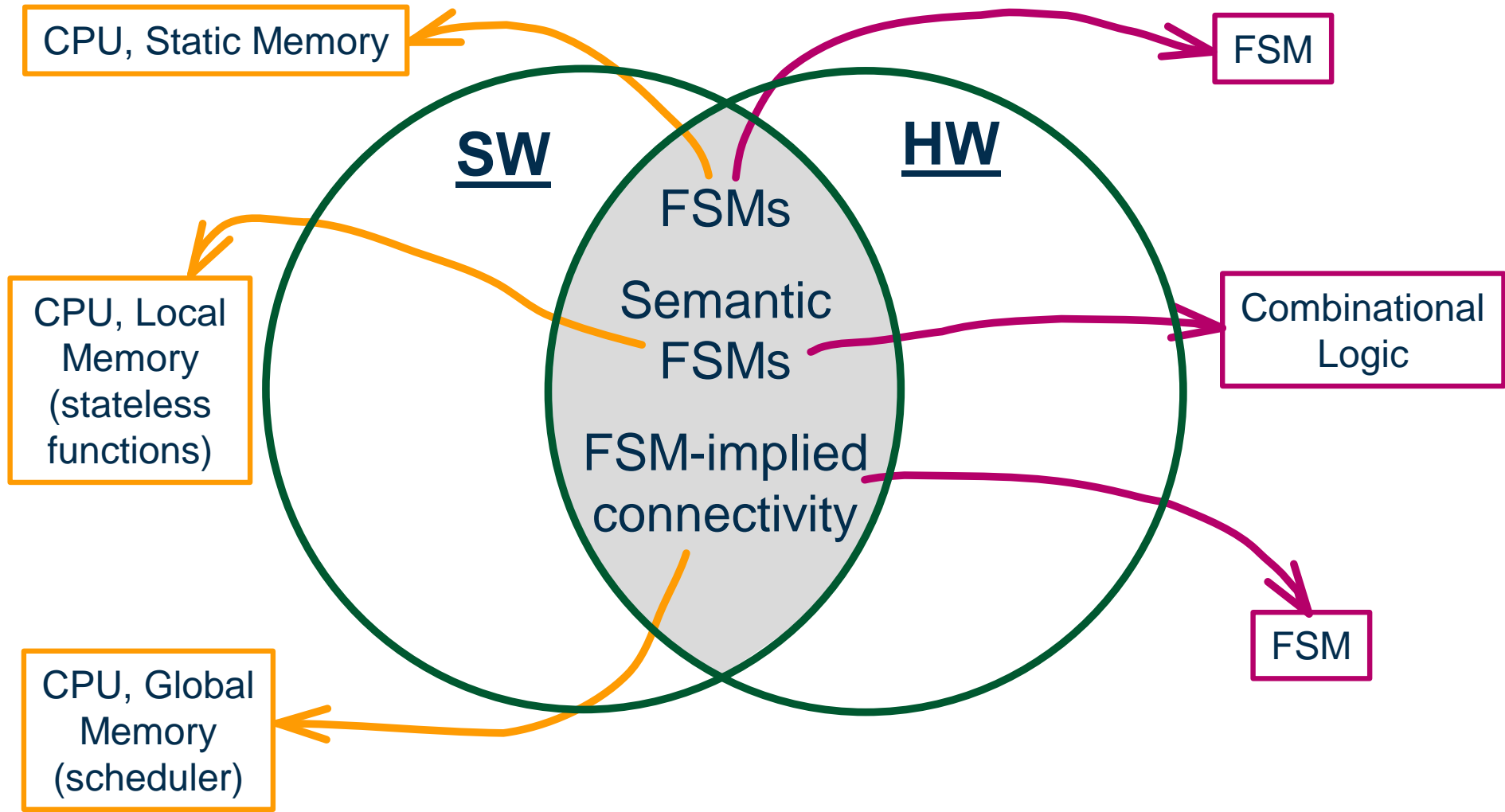


Don't These Two Domains Overlap?



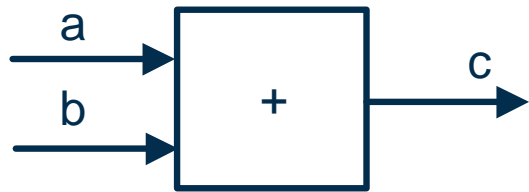


Common Specification, Dual Physical Inferences

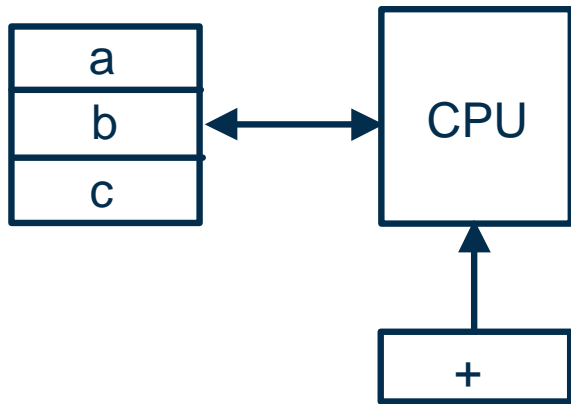




Behavioral Duals: Computation Advancing State



$$c = a + b;$$



- Consider assignment:
 - has both computation and state
 - programmer's view
- a, b, c are all system state
 - shared memory for SW
 - wires for HW
- HW is guaranteed to activate and propagate; SW has unbounded workspace (memory) for the computation, if it needs it
- Once encapsulated what behaviors may have a dual means of contributing to the system composition?
- State is the boundary



Modeling Duals and System Architectural Composition

- **Can all behaviors be considered duals?**

- ... maybe someday all computations will be done combinatorially
 - ... or we will have enough time for all computations to be modeled in software

- in the meantime, some behaviors are far more convenient than others to model in software and some behaviors are far better than others to be modeled in HW for architectural/system resource reasons

- but there are lots of behaviors in the middle
 - and for those in the middle

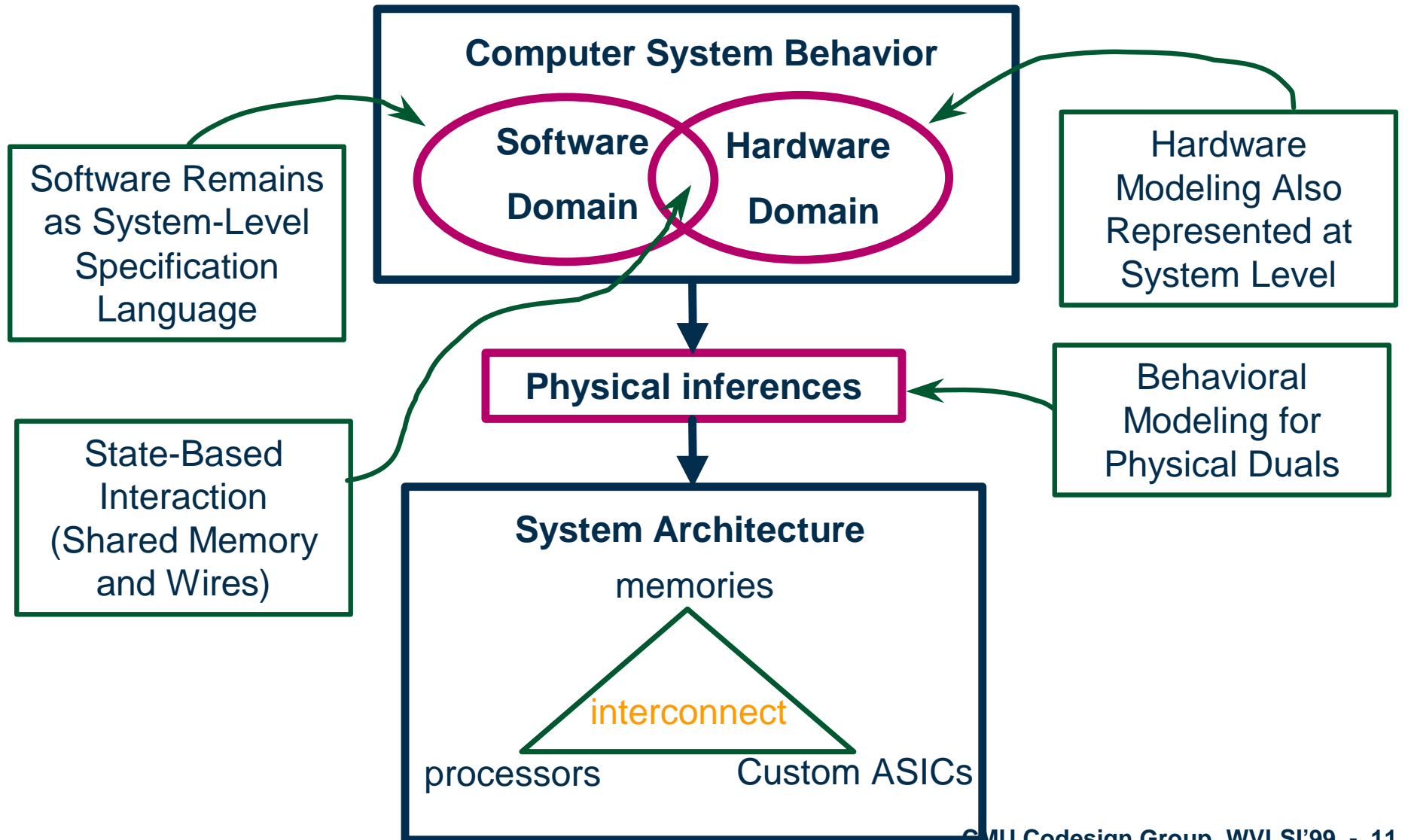
- **The most architectural possibilities exist for behaviors that can be considered duals**

- **Codesign can be considered dual modeling**

- Extracting HW from SW models!

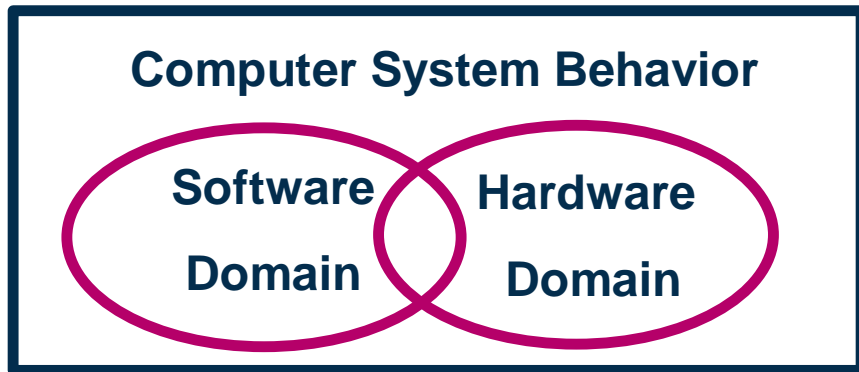


Our Approach

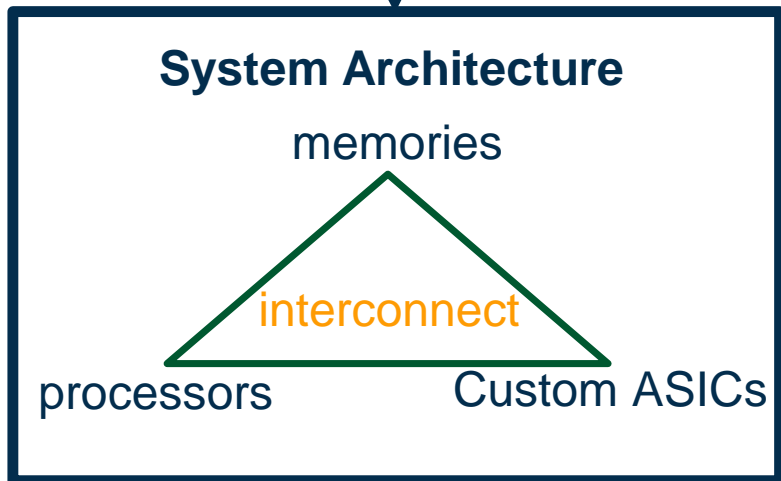




When Less Specification Is More



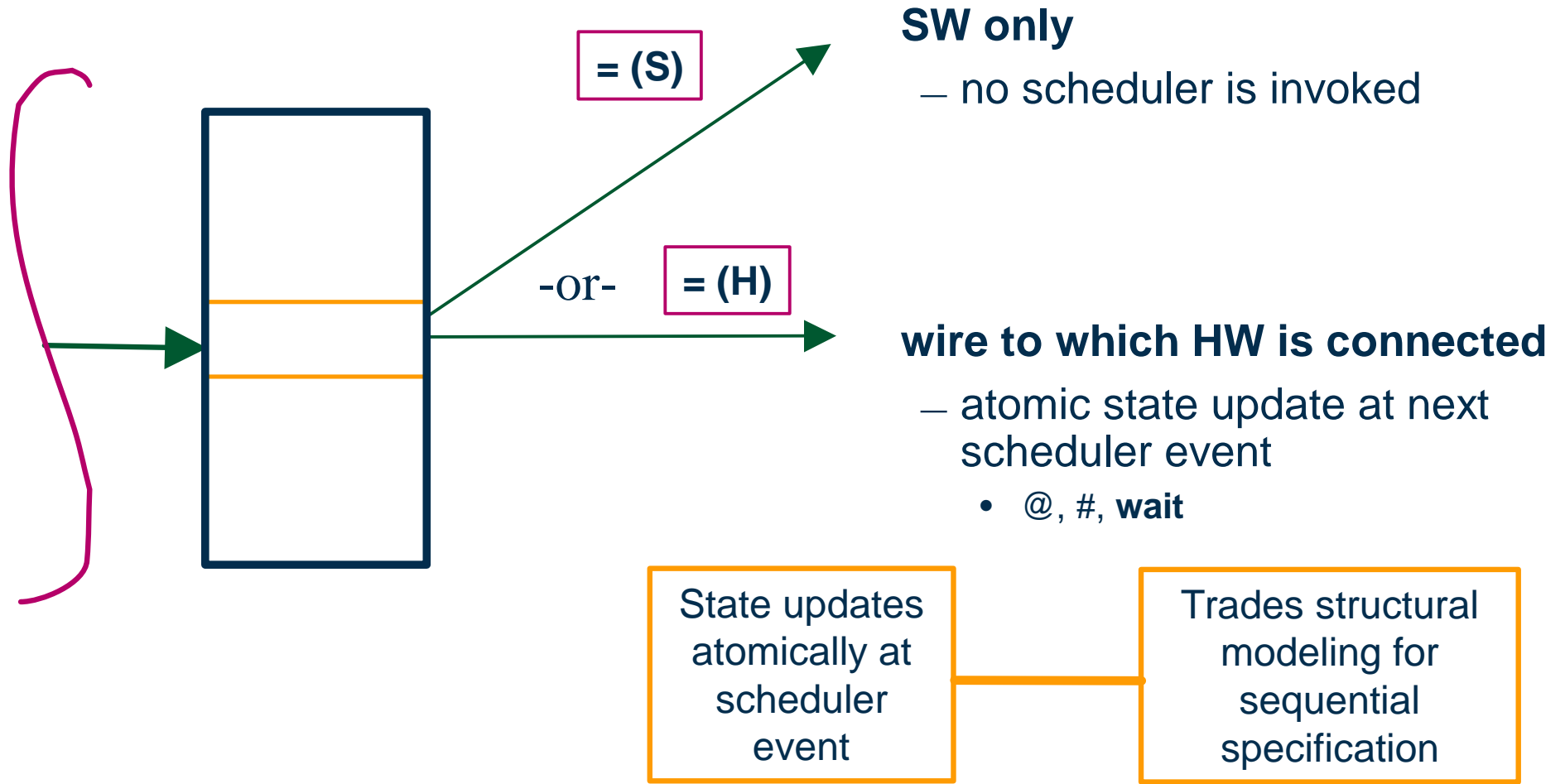
Physical inferences



- **No new languages**
C/pthreads & Verilog
already widely used
similar syntax
not already “bloated”
- **No restrictions on domain-specific modeling situations!**
- **Domain Pre-Partitioning Not Required**
- **Richest Possible Set of Architectural Possibilities**



In Search of Dual Models: Behavioral Migration Starting With Threads





Threads as Dual Computation Resource Models

```
while(1) { /* SW1 */
  pthread_mutex_lock(&mtx);
  C = A + F;
  B = D + E;
  pthread_mutex_unlock(&mtx);
}
while(1) { /* SW2 */
  pthread_mutex_lock(&mtx);
  A = B + C;
  F = D + E;
  pthread_mutex_unlock(&mtx);
}
```

```
always @(A or D or E or F) // HW1
begin
  C = A + F;
  B = D + E;
end

always @(B or C or D or E) // HW2
begin
  A = B + C;
  F = D + E;
end
```

■ What's the same here?

Behavior inside corresponds left <--> right
Top to bottom its some sort of feedback loop

Same model,
different physical implementations:
DUALS

■ What's different?

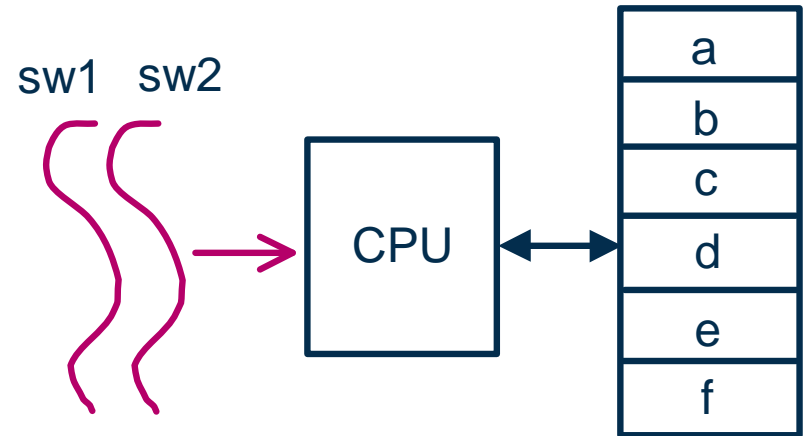
Architectural inferences due to different resource models



SW Threads

```
while(1) { /* SW1 */
  pthread_mutex_lock(&mtx);
  C = A + F;
  B = D + E;
  pthread_mutex_unlock(&mtx);
}

while(1) { /* SW2 */
  pthread_mutex_lock(&mtx);
  A = B + C;
  F = D + E;
  pthread_mutex_unlock(&mtx);
}
```



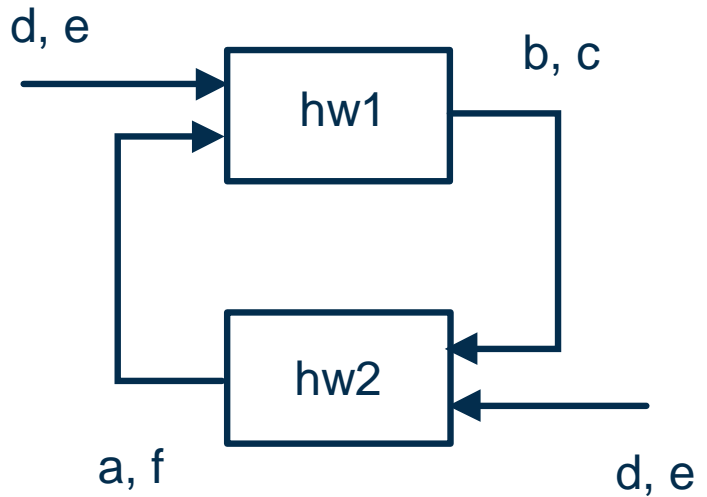
Resource Efficient Model of Concurrency
But: Concurrency/Activation not Guaranteed!



HW Threads: Guaranteed Activation and Propagation

```
always @(A or D or E or F) // HW1
begin
    C = A + F;
    B = D + E;
end

always @(B or C or D or E) // HW2
begin
    A = B + C;
    F = D + E;
end
```

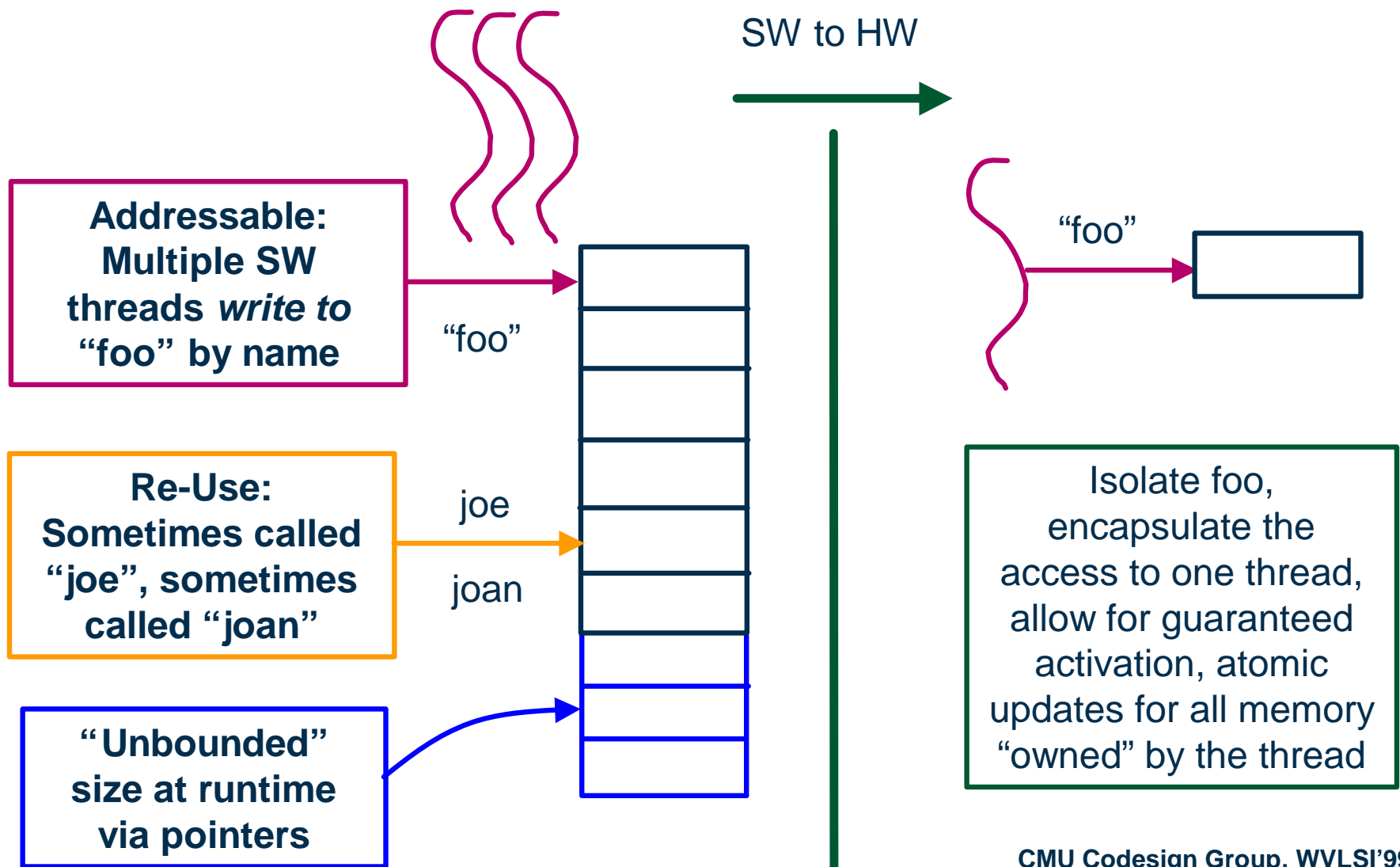


Specifies System Resources Behaviorally
But: Requires Simulation in Lieu of Actual Resources

No memory is contributed to the system composition,
but wired state and structure is contributed

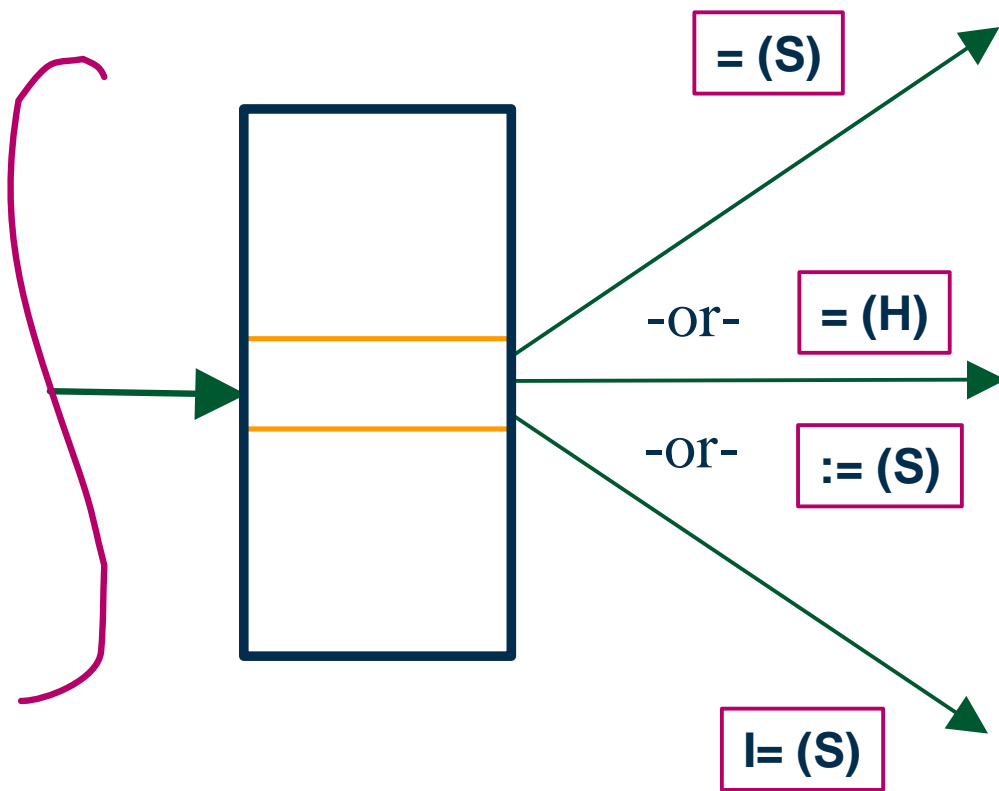


Domain Migration





Extended Resource Modeling in SW



SW only

- no scheduler is invoked

wire to which h/s is connected

- atomic state update at next scheduler event
 - @, #, wait
 - lock, wait, semaphore

HW residing in shared memory

- cosim scheduler immediately invoked



Our Cosimulator

- **Peer-based Co-execution**

 - Common Threading Model

 - Common State Model

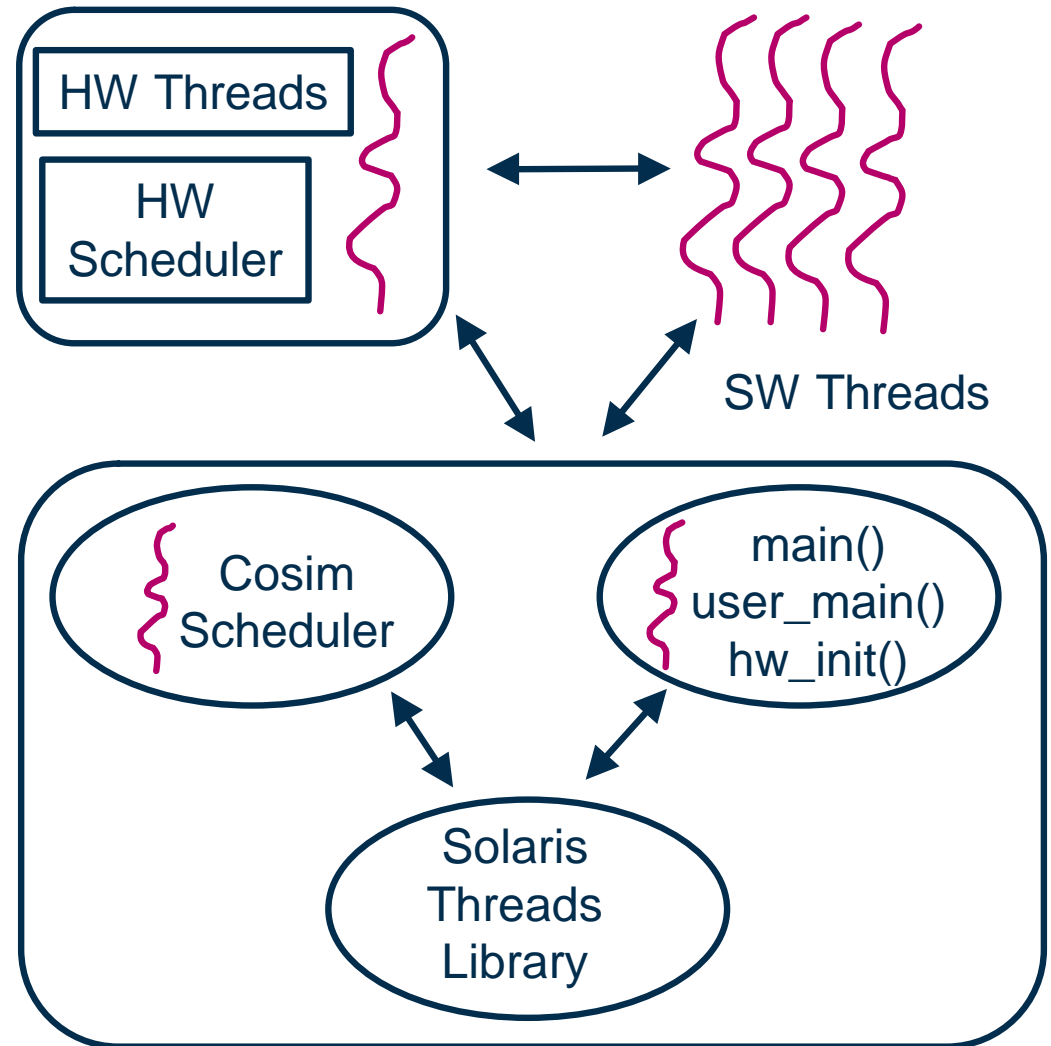
 - Extended Updates

- **Uses Solaris Threads**

 - eventual custom cosim scheduler

- **Currently defining common co-execution syntax**

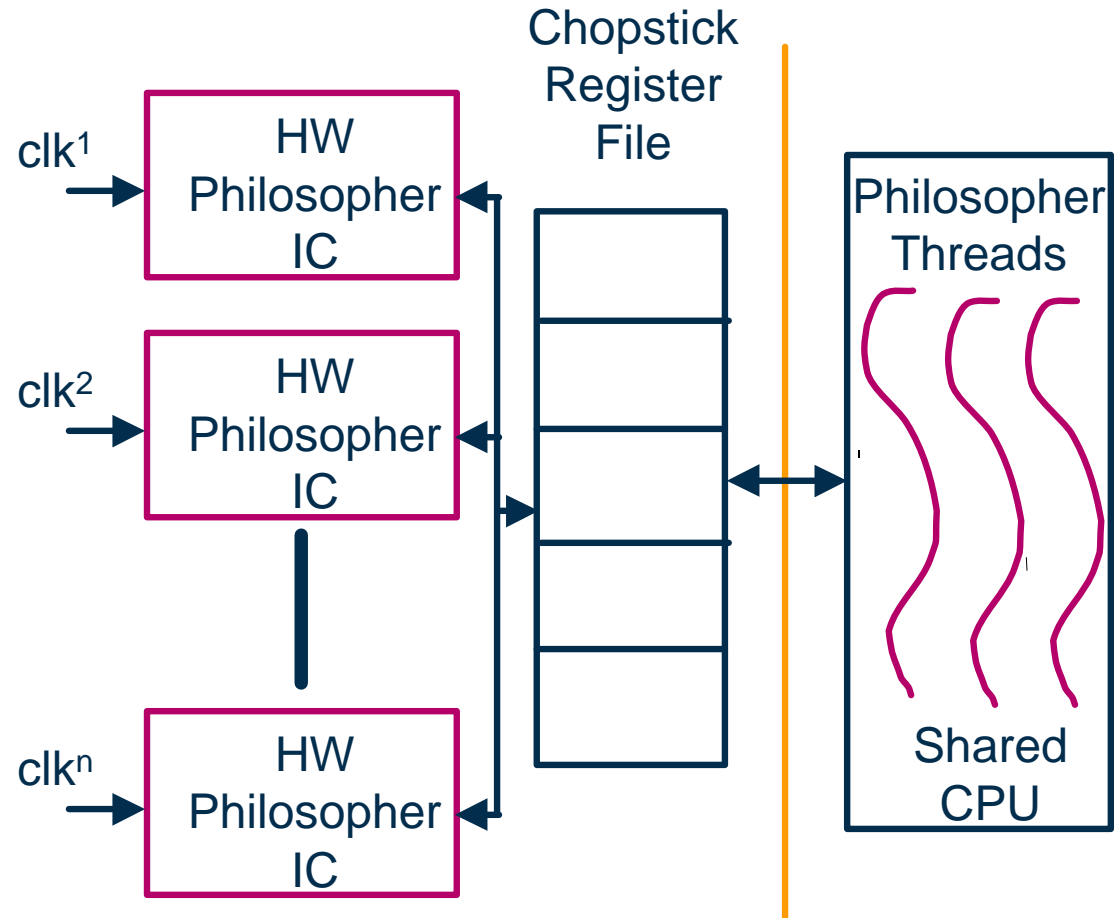
 - parsers in each domain will generate





Dining Philosophers Example for Mixed Domain Resource Modeling

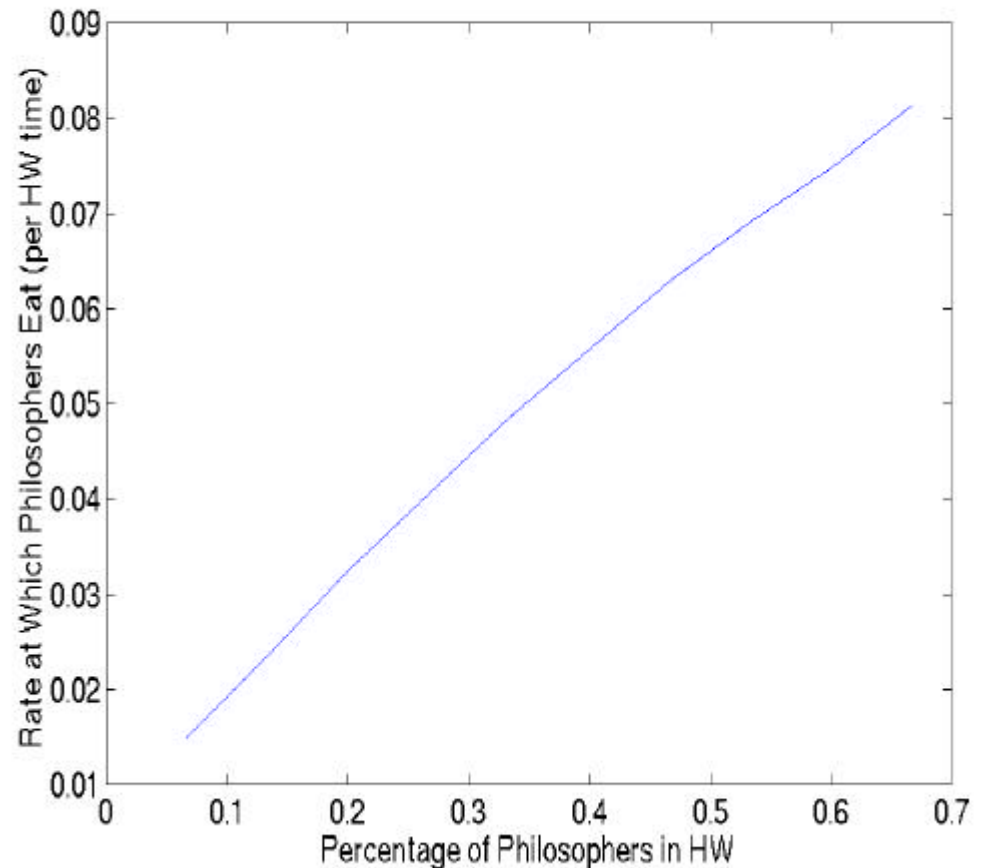
- **Philosophers share chopsticks to eat**
need two to eat
otherwise think
- **HW, SW philosophers**
in SW share single CPU
but each HW philosopher is a resource
- **chopsticks in HW**
clock phasing in HW
mutexes in SW





Resource Modeling Behavior

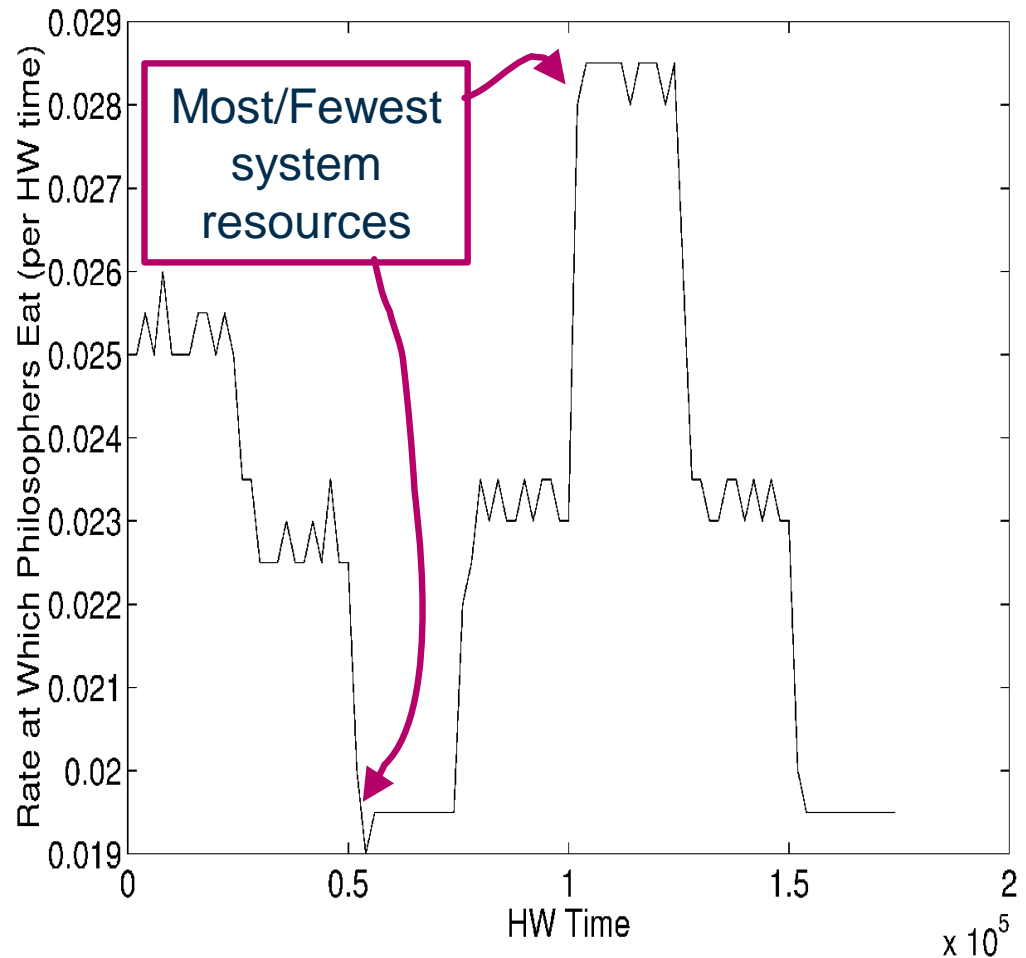
- **Single CPU resource for SW**
- **As ratio of HW/SW philosophers - , more philosophers eat, overall**
 - more resource threads have been added to the system architecture
- **simulation is required to capture resource models**





Software Resource Threads == Dynamic System Architecture

- **A mixed computation resource**
guaranteed activation
but also dynamic
- **Create/destroy one philosopher at a time**
- **Food consumption follows number of resource philosophers in the dynamic system architecture**





Unified H/S Testbenches

HW (Eat, Think Time) in Clock Cycles	SW (Eat, Think Time) in HW Time	Number of Times SW Eats	Number of Times HW Eats
(1,2)	(20,40)	74.3	78.0
(2,4)	(20,40)	69.3	69.5
(4,8)	(20,40)	63.7	39.5
(1,2)	(40,80)	57.0	82.0
(1,2) (2,4)	(20,40) (40,80) (80,160) (160,320)	38.0	65.5

■ system simulation

testbenches are representative of time, sequencing native to each design domain

how does the overall behavior of my system change in what resources should I place what behavior?

■ Small system with lots of contention

2-h, 3-s threads



Summary

■ Peer-Based Executable Co-Specification Using

shared memory and wires as fundamental models of state and inter-domain interaction

threads as common model of encapsulation of concurrent behavior

■ Dual Modeling Permits Codesign Architectural Inferences for a Rich Set of System Architectures

Infer Alternate System Architectures by Altering the way a behavior is specified in the HW, SW co-executing domains