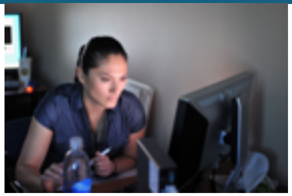
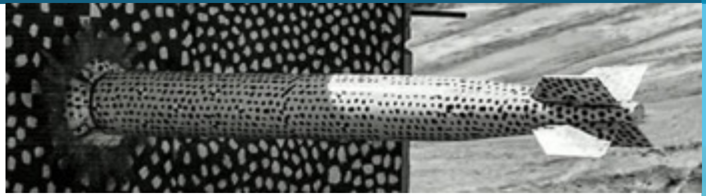
Sandia National Laboratories

Simulation by Composition:
Using models as building blocks to enable simulation of complex node architectures





PRESENTED BY

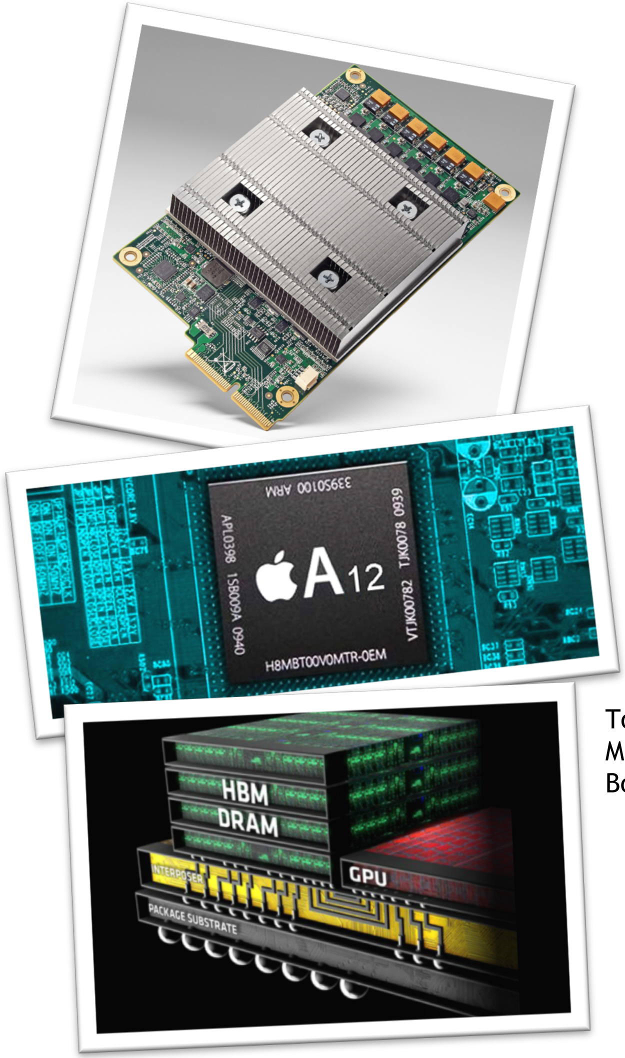
Gwen Voskuilen

U.S. DEPARTMENT OF ENERGY   NNSA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.
SAND2019-9588C

Architecture innovations drive models

- As scaling slows, architectures become more creative
  - Instead of just bigger
  - Increasingly complex
    - Heterogeneous processors, GPUs, other accelerators
    - Processing at memory and/or throughout cache hierarchy
    - Customization
- **Simulation:**
  - The more "stuff" to simulate, the slower it gets
  - New models, not just scaling existing ones



Top: Google TPU
Middle: Apple A12 SoC
Bottom: AMD

Architecture trends are leading to slower development of slower simulations

Impact of Complexity in Node Models

- Complex models are difficult to work with
  - Noisy, masks cause & effect
  - Difficult to debug
  - Slow

- Often models are tightly integrated
  - Modifying the model becomes complex
    - E.g., pervasive assumptions about caching or address mapping

Approaches

- Simpler simulators
  - Faster simulation
  - Less accurate
  - What can be simplified is problem dependent → makes reuse difficult
- Accept the complexity
  - Slow simulation
  - Slower
  - More accurate (?)
    - Complex models are hard to validate
- ***Or, build composable models***

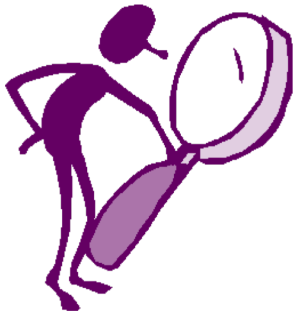
Composability is Key

*"A highly composable system provides **components** that can be **selected** and **assembled** in various combinations to satisfy specific **user requirements**"*

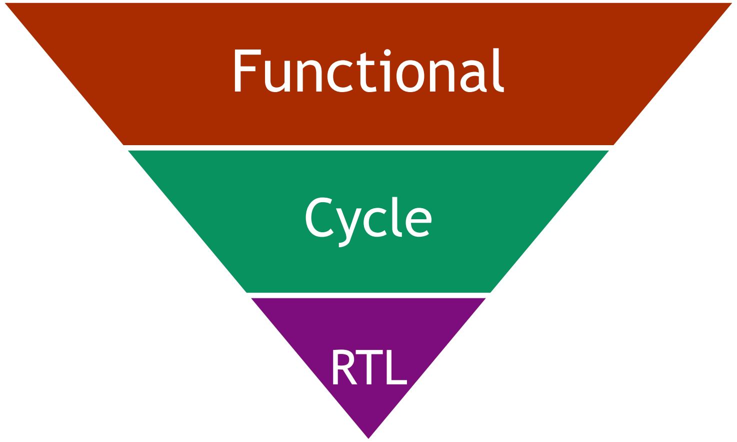
-Wikipedia

- Benefits
  - Rapid creation of new architectures
  - Minimize the work to explore new concepts
    - Only add/modify the new parts
    - Minimal disturbance to existing infrastructure
  - Tune tradeoff between fidelity and simulation overhead to specific instances
    - E.g., simplify the core model, keep the caches detailed
    - OR simplify the cache hierarchy and use detailed core models

Composability makes research better

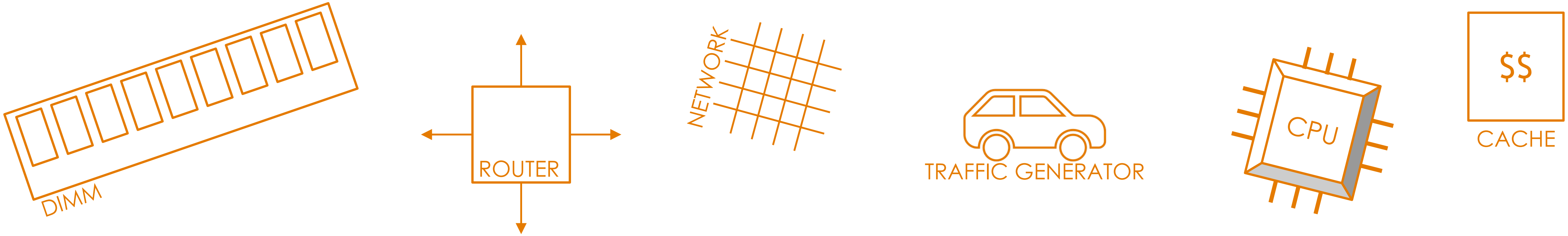


- Fair comparisons from point changes
  - Single change between base system and comparison point
  - E.g., swap prefetch model at runtime without disturbing any other part of the simulation
- Validating models is time consuming but necessary
  - Breaks much of validation into manageable chunks
  - Create new systems in which many of the pieces have already been validated
- Workflow benefits
  - Continuous path from high-level to detailed models
    - Build hierarchically
  - Collaborative development



Achieving composability

- Defined APIs between classes of components
  - Cores and caches
  - Instruction stream and pipeline model
  - Must be flexible enough to adapt to future ideas
    - Hard part
- Fast simulation
  - Leveraging composable model properties to facilitate parallel simulation
    - Capture, synchronize interactions between models
  - Enable seamless transition from fast low-detail models to slow high-detail ones

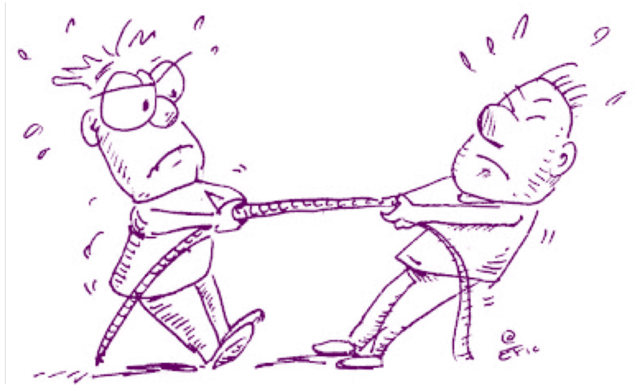


Composability versus interoperability

- Many simulators support interoperability
  - Bridge between simulators
  - Problems:
    - Interface is often simulator-specific → rewritten for every integration
      - Inflexible because not written to be generic
    - Models can rely on information not encoded in their interface API
      - E.g., one simulator expects in-order requests, other breaks that, first one has problems!

The Challenges of Composability

- Inefficiencies
  - Engineering overhead to design models to existing APIs and build APIs into models
  - Code/runtime overhead from designing models to APIs
    - Must be careful to build API such that it is flexible but doesn't impose too much burden
  - Have to work through APIs instead of around them
    - No shortcuts
- Never completely sufficient
  - Interfaces unable to capture arbitrary future ideas
    - Always be a need to hack simulators



Composable Node Models in SST

The Structural Simulation Toolkit

Goals

- Create a standard architectural *simulation framework* for HPC
- Ability to evaluate future systems on DOE/DOD workloads
- *Use supercomputers to design supercomputers*

Status

- Parallel framework (*SST Core*)
- Integrated libraries of components (*Elements*)
- Current Release (9.0)
  - https://sst-simulator.org
  - https://github/sstsimulator

Technical Approach

- Parallel
  - Parallel Discrete Event core with <u>conservative optimization over MPI/Threads</u>
- Interoperability
  - Node: memory, cores, caches, NoCs
  - System: routers, NICs, schedulers
- Multi-scale
  - Detailed and simple models that interoperate
- Open
  - Open Core, non-viral, modular

Consortium

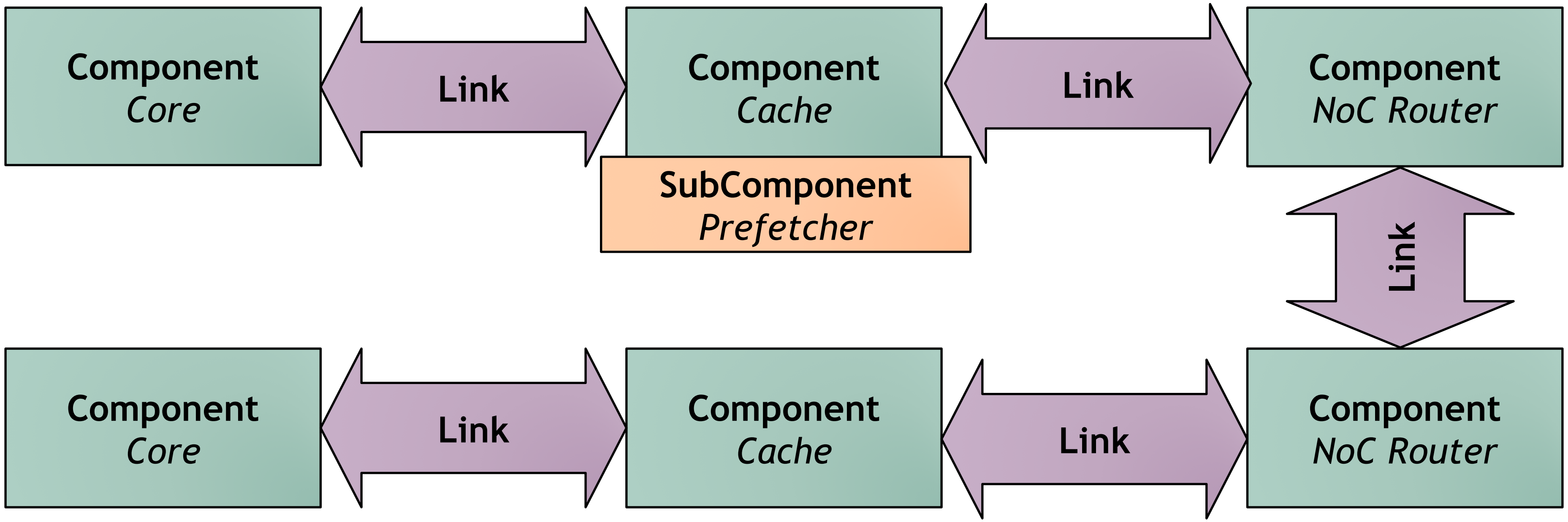
- “Best of Breed” simulation suite
- Combine Lab, Academic & Industry

Key Capabilities

- **Parallel**
  - Built from the ground up to be scalable
  - Conservative, distance-based optimization
  - MPI + Threads
- **Flexible**
  - Enables “mix and match” of simulation components
  - Custom architectures
  - Multiscale tradeoff between accuracy and simulation time
    - E.g., cycle-accurate network with trace-driven endpoints
- **Open API**
  - Easily extensible with new models, modular framework and open source

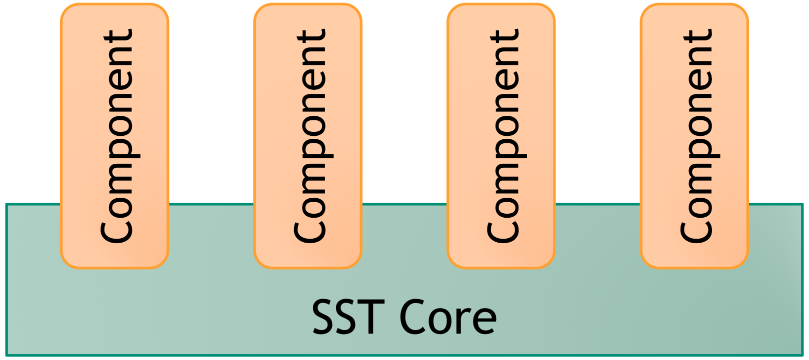
SST Building Blocks

- SST simulations are comprised of **components** connected by **links**
- Components communicate by sending **events** over the links
  - Components define **ports** which are valid connection points for a link
- Components can use **subComponents** and **modules** to expose composable functionality internally



SST Architecture

- SST **Core** Framework
  - The backbone of simulation, parallel, high-performance, multi-threaded
  - Provides utilities and interfaces for simulation components (models)
    - Clocks, event exchange, statistics and parameter management, parallelism support, etc.
- SST **Element** libraries
  - Libraries of components that perform the actual simulation
  - Elements include processors, memory, network, etc.
  - Compatible with many existing simulators: DRAMSim2, HMCSim, Spike, Ramulator, etc.

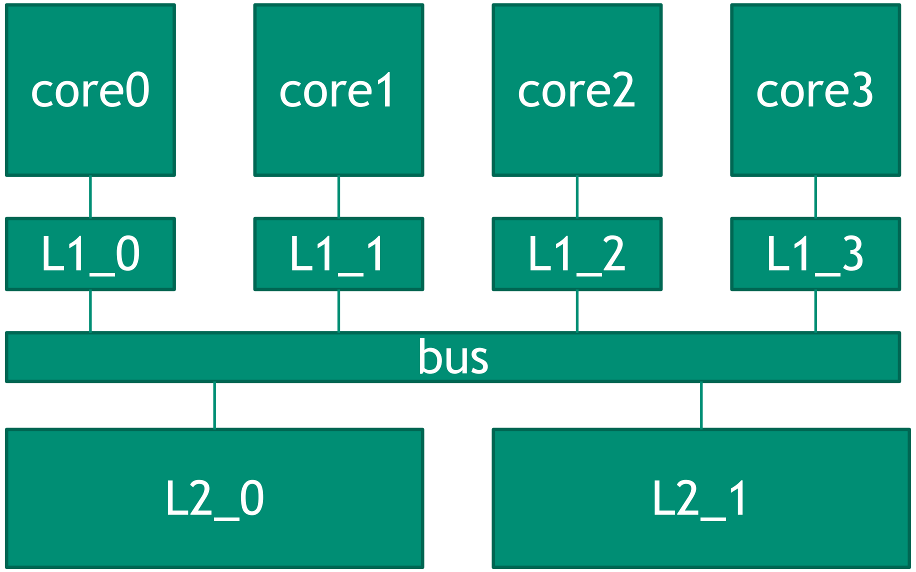


Breadth and Depth…

- Detailed Caches
- Multiscale Memory Models
- Dynamic Trace-based Processors
- Functional Processors
- High-level Program Communication Models
- Cycle-based Networks
- High-level System Workflows

- memHierarchy - Cache and memory
- cassini - Prefetchers
- CRAMSim - DDR, HBM
- NVDIMMSim - Emerging Memories
- GoblinHMC - HMC
- SimpleDRAM - Low-fidelity DDR model
- ariel - PIN-based tracing
- MacSim - GPGPU
- GPGPUSim - GPGPU
- Spike - RISC-V ISA
- ember - State-machine message generation
- firefly - Communication protocols
- hermes - MPI-like interface
- merlin - Network router model and NIC
- kingsley - Network-on-chip model
- scheduler - Job-scheduler simulation models

Composable Node Modeling in SST: Building bigger models



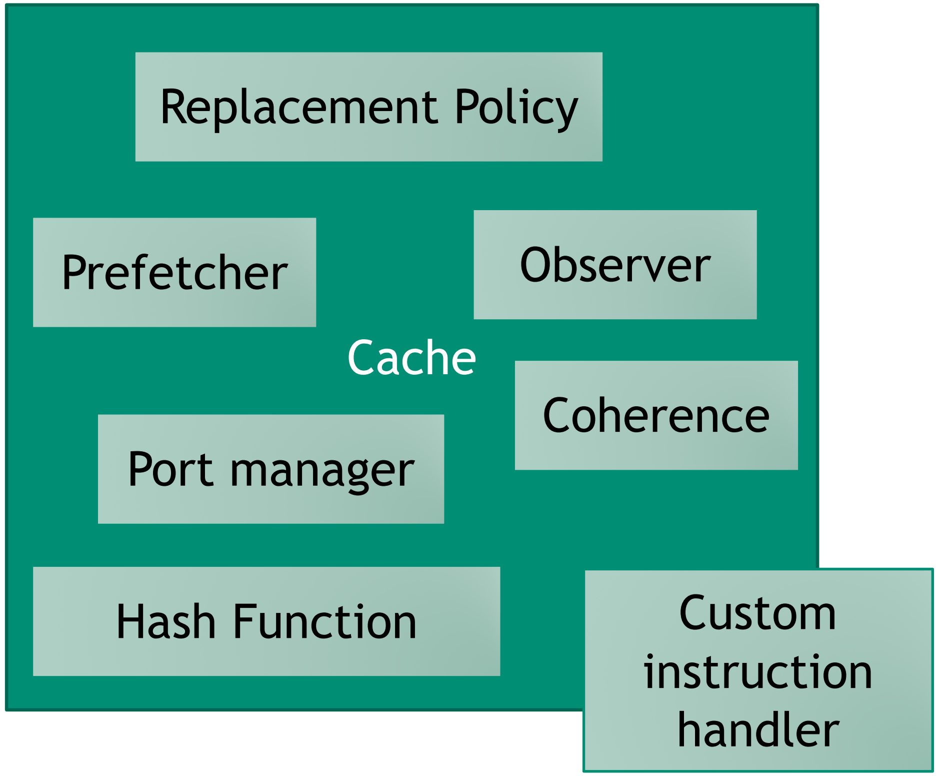

```
bus = sst.Component("bus", "lib.bus")

for x in range(0,4):
     core = sst.Component("core" + str(x), "lib.cpu")
     l1 = sst.Component("L1_" + str(x), "lib.cache")
     link = sst.Link("core_to_cache_" + str(x)
     link.connect(core, l1)
     linkb = sst.Link("cache_to_bus_" + str(x)
     linkb.connect(bus, l1)


l2_0 = sst.Component("L2_0", "lib.cache")
l2_1 = sst.Component("L2_1", "lib.cache")

Link0 = sst.Link("bus_to_l2_0")
Link1 = sst.Link("bus_to_l2_1")
Link0.connect(bus, l2_0)
Link1.connect(bus, l2_1)
```

Composable Node Modeling in SST: Building deeper models



- SubComponent
  - Slot in a component for loading some function
  - Example: cache replacement policy
  - Subcomponents can live in any library; allows users to customize without hacking the component
- *Enables*
  - Hierarchical models
    - Successive refinement
  - Customizable model outlines
  - Model re-use

SST: Future directions in node modeling

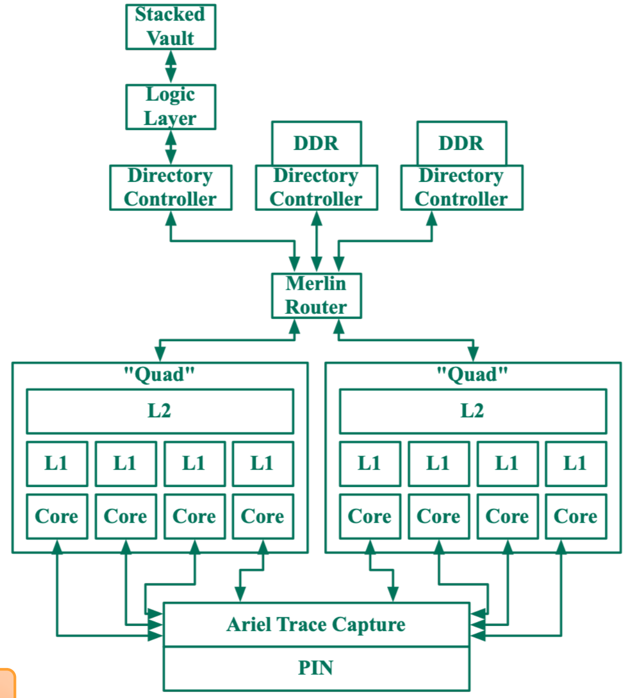
- Increasing composability within existing node models
  - Accelerator interfaces in core models
  - Expanded support for drop-in addition of custom instructions
- Support for composing RTL models with C++ models
- Growing the eco-system

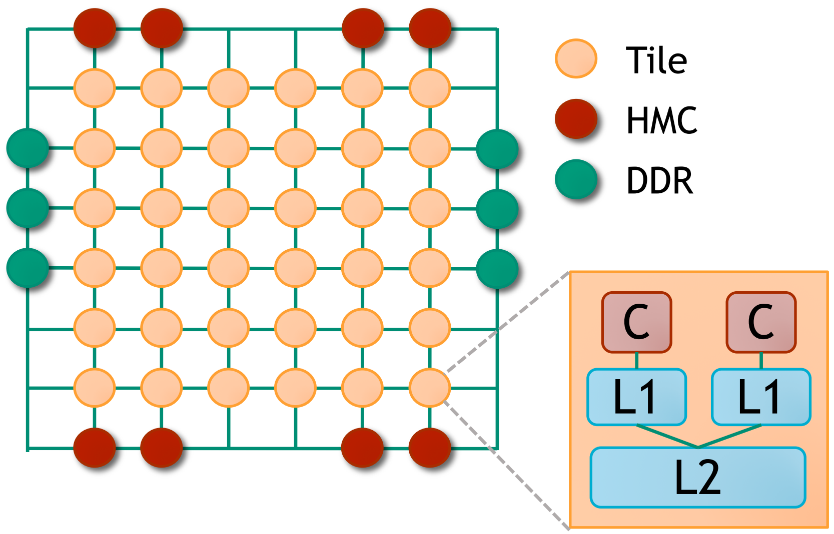
Revisiting the composability challenges

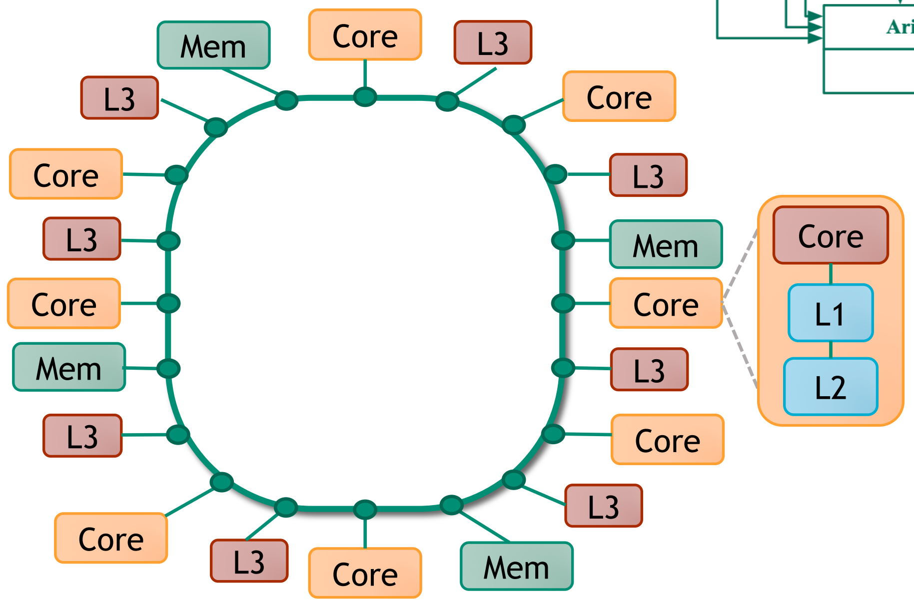
- Component/SubComponent APIs designed to be lightweight
  - Minimize runtime overheads
  - While enabling SST Core to manage parallel execution between components
    - Benefit from forcing components to interact through APIs
- Interfaces
  - Network
  - Memory (core ←→ cache/memory)

Closing thoughts

- Architectures are evolving quickly ➔ slower simulation
- Building simulations out of composable pieces
  - Amortizes investment in simulation infrastructure
  - Speeds up innovation
  - Reduces the validation burden







Sandia
National
Laboratories