SST and Cycle-accurate Simulation of xBGAS

Jie Li
Ph.D. Student, TTU
05/03/2022

Overview

- SST Overview
- Simulating with SST
- SST for xBGAS
- Summary & Future Work

SST Overview

The Structural Simulation Toolkit

- Why SST?
  - Many simulators are serial and unable to simulate very large systems.
  - Significant performance issues with tying many simulators together.

  Lack of scalability

  - Modification on tightly-coupled simulations is difficult.
  - Difficult to simulate at different levels of accuracy.

  Lack of flexibility

**The Structural Simulation Toolkit:
an open, modular, parallel, multi-objective,
multi-scale simulation framework for
scalability and flexibility.**

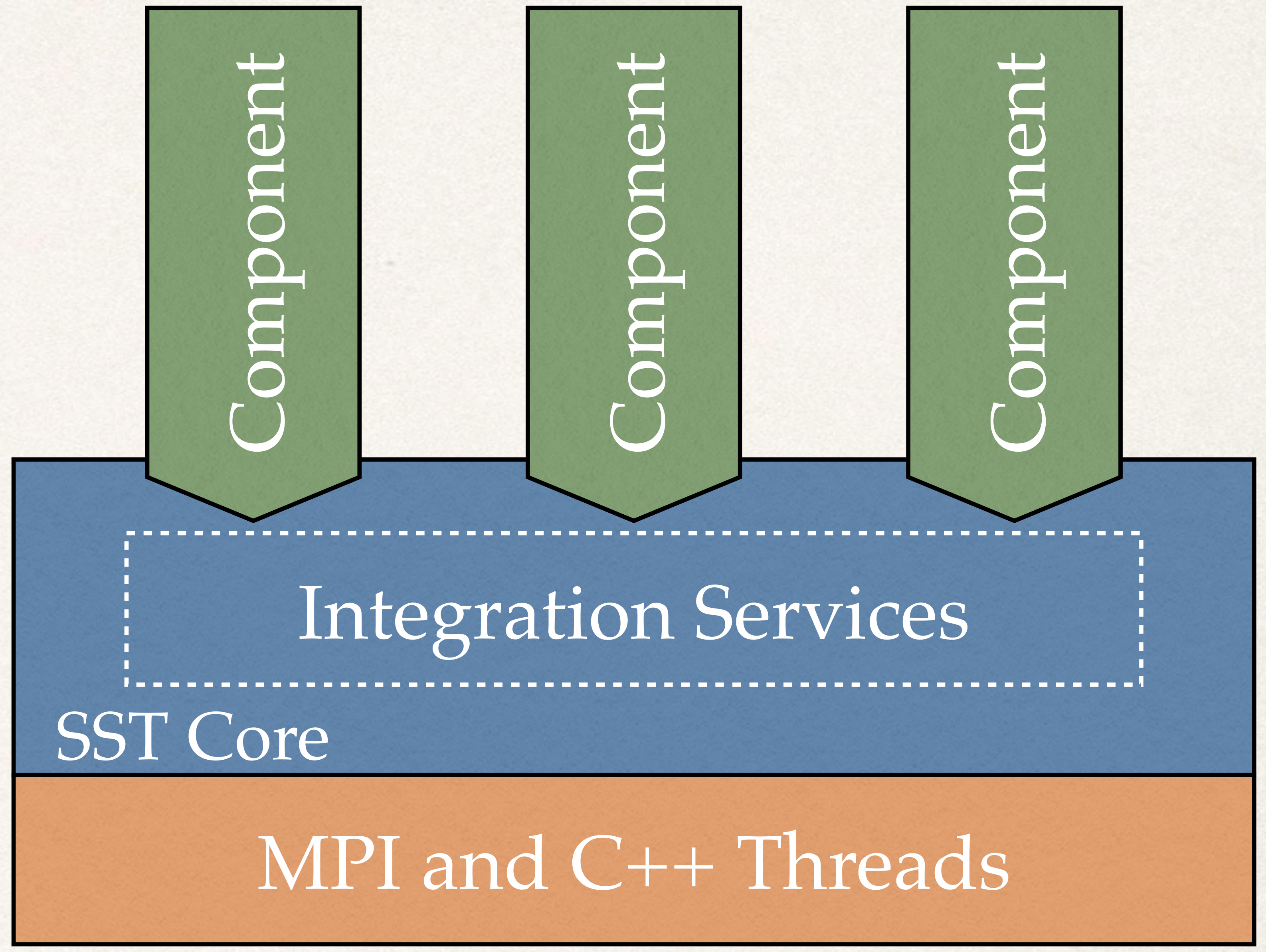
The Structural Simulation Toolkit

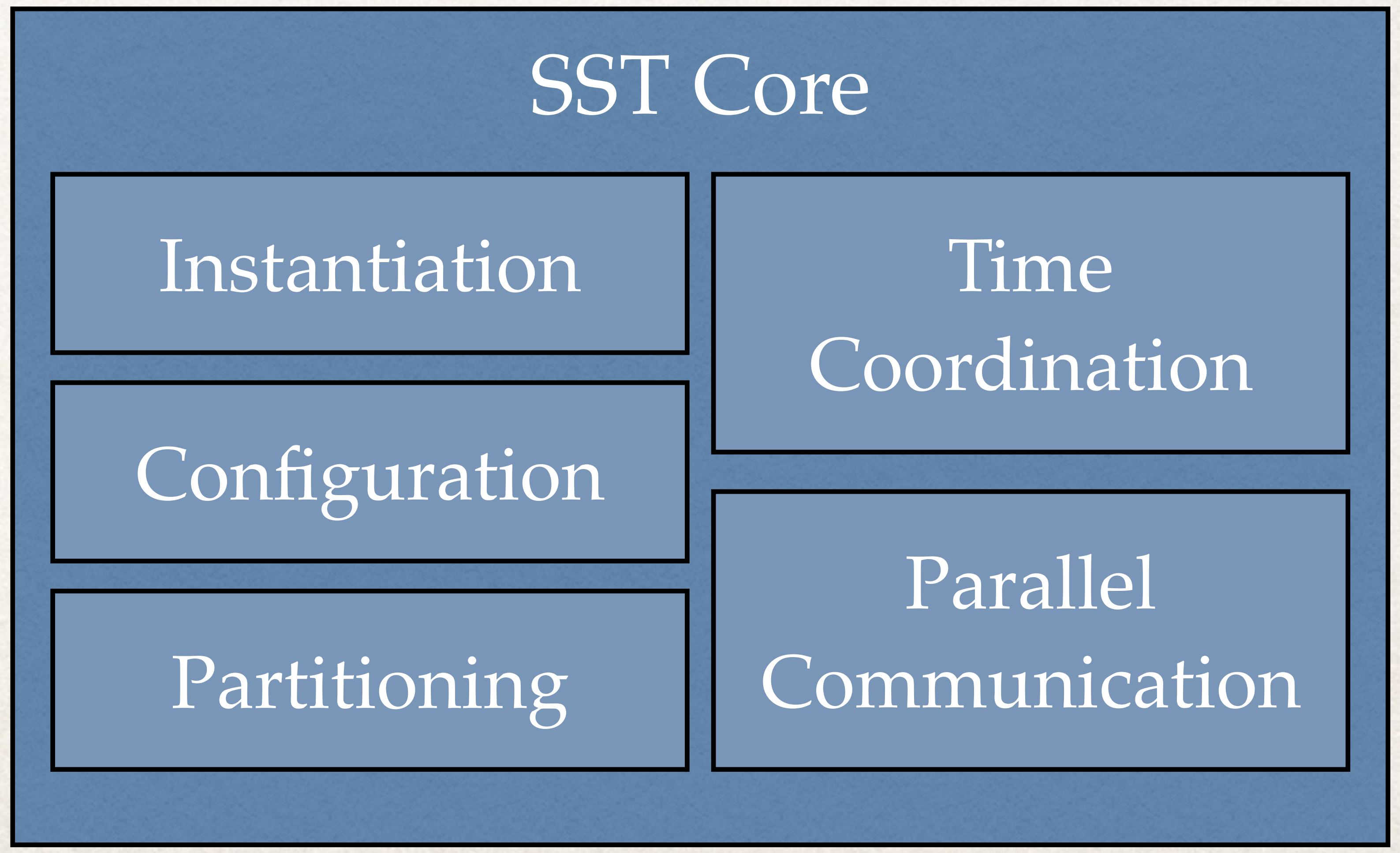
- Goals:
  - Create a standard architectural simulation framework for HPC
  - Enable "mix and match" of simulation components.
  - Custom architectures and custom tradeoff between accuracy and simulation time.
  - Use supercomputers to design supercomputers.
    - The simulation performance is optimized over MPI/Threads.

Sandia National Laboratories

SST Architecture

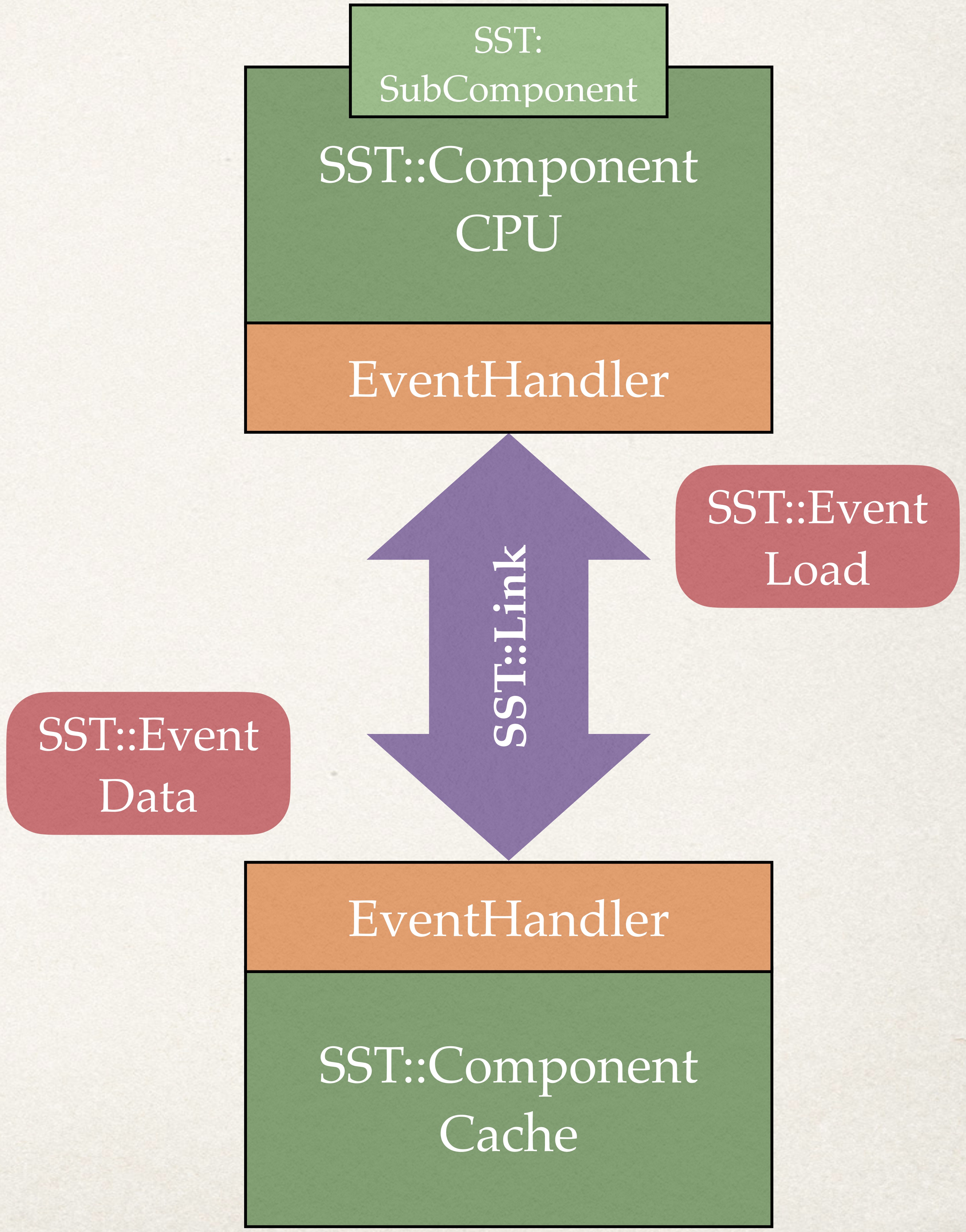
- **SST Core** (framework):
  - The backbone of simulation.
  - Provides utilities and interfaces for simulation components.
  - Clocks, event exchange, statistics and parameter management, parallelism support, etc.
- **Components** (SST element libraries):
  - Components perform the actual simulation.
  - Built-in models include processors, memory, network, etc.
  - Compatible with many existing simulators: DRAMSim2, Spike, HMC-Sim, etc.





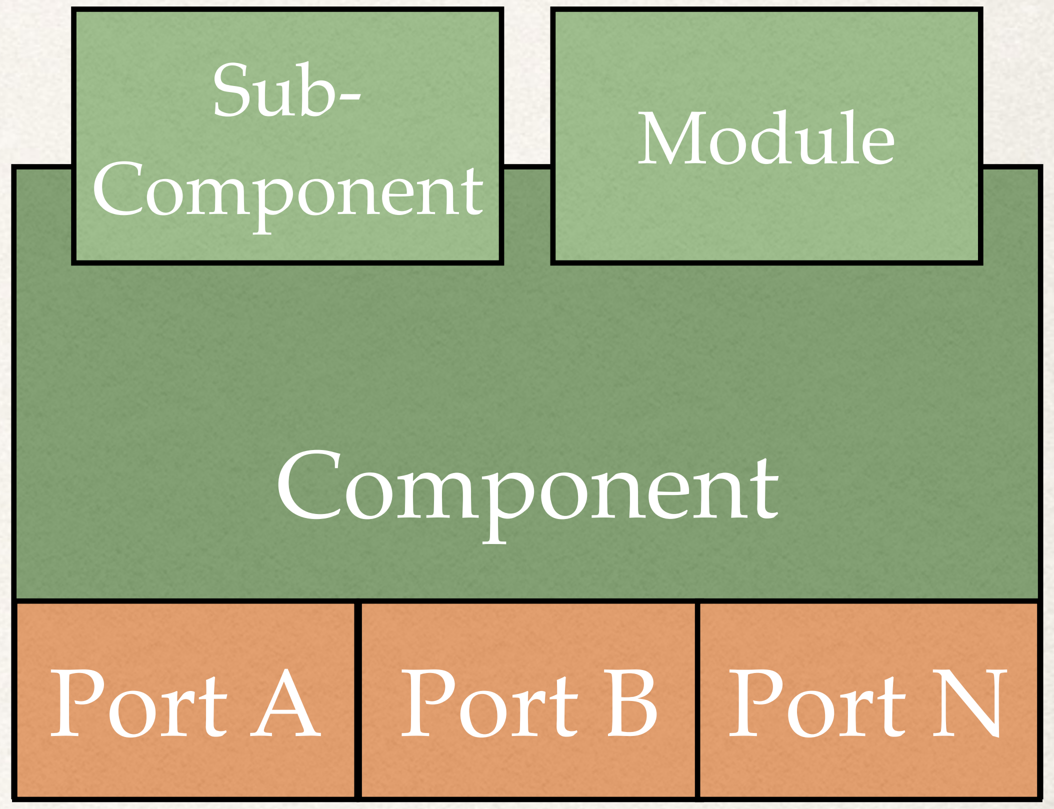
Key Objects

- **SST::Component**
  - Simulation model
- **SST::Link**
  - Communication path between two components.
  - Poll or interrupt with EventHandler.
- **SST::Event**
  - A discrete event
- **SST::Clock::Handler**
  - Function to handle a clock tick.
- **SST::SubComponent**
  - Add functionality to Components



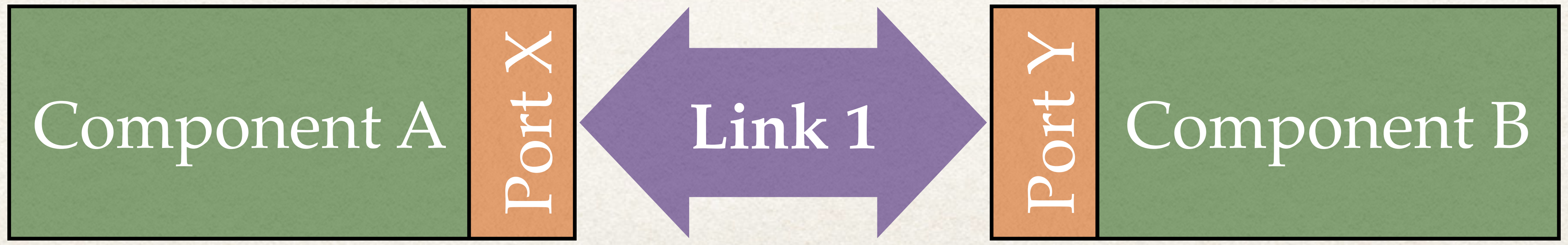
Component

- Basic building block of a simulation model. It performs the actual simulation.
- Use links and ports to communicate with other components.
  - Components define ports, links connect ports between components.
  - Polled: register a clock handler to poll the link.
  - Interrupt: register an event handler to be called when an event arrives.
  - Both: receive events on interrupt, send events on clock.
- Loads SubComponents and Modules for additional functionality.
  - Useful for parts of a component that should be swappable between simulations. E.g., scheduling algorithms, address mapping schemes, cache prefetchers.
  - Useful for exposing interfaces to other components. E.g., network provides a subcomponent that can be loaded to handle traffic into/out of network.



Link

- Connects two components.
  - Connect a specific named “Port X” on component A to a “Port Y” on component B.
- The only mechanism by which components communicate.
  - Necessary for parallel simulation.
- Has minimum, non-zero latency for communication.
  - Except self-links.
  - Except during init/complete phases.
- Transparently handles any MPI/threaded communication.

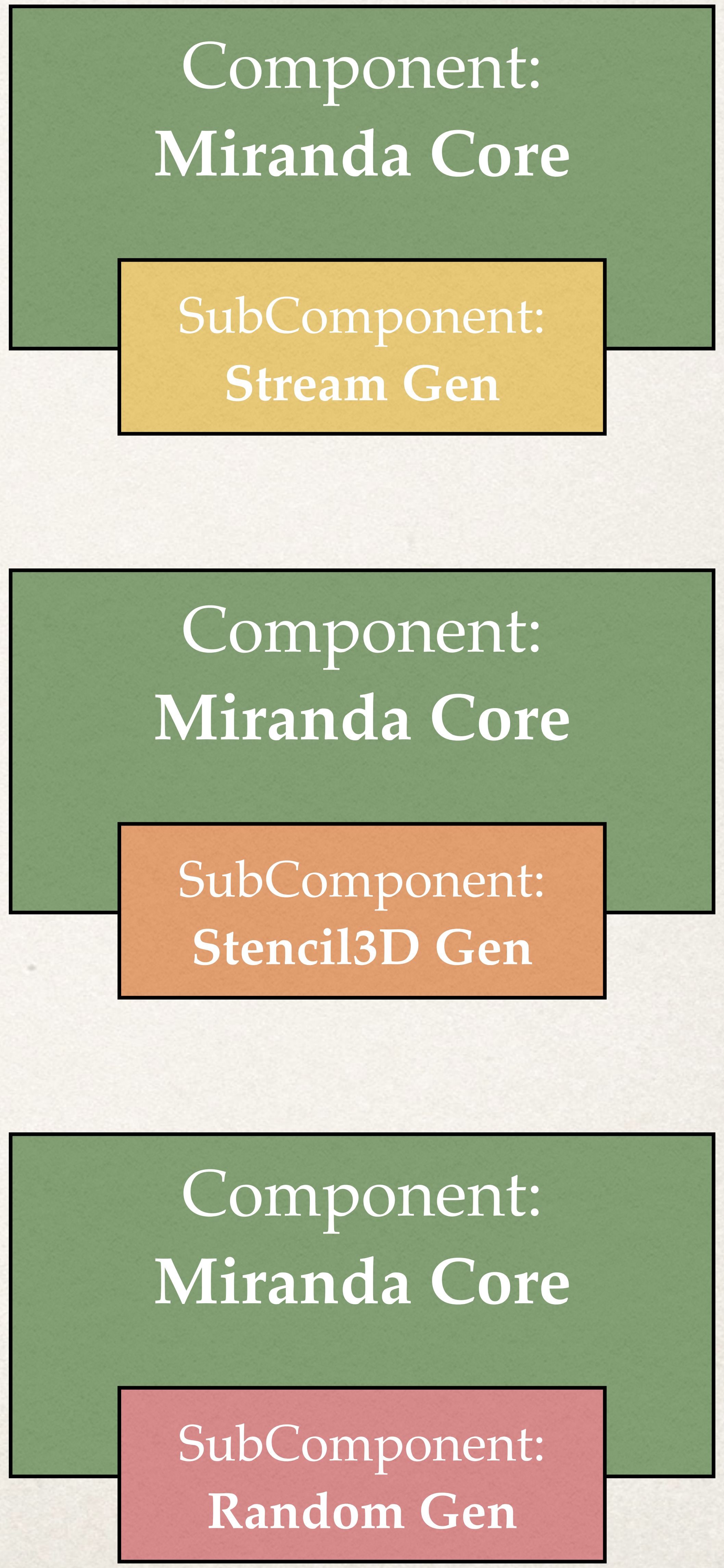


Event

- Unit of communication between two components.
  - Packet format is up to the communicating components.
- Some standardized interfaces.
  - Facilitates "mix and match" capability.
  - sst/core/interfaces/
  - Memory (stdMem)
    - Defines commands & event format for communication with caches & memory.
  - Network (simpleNetwork)
    - Defines a header for events sent through a network component.

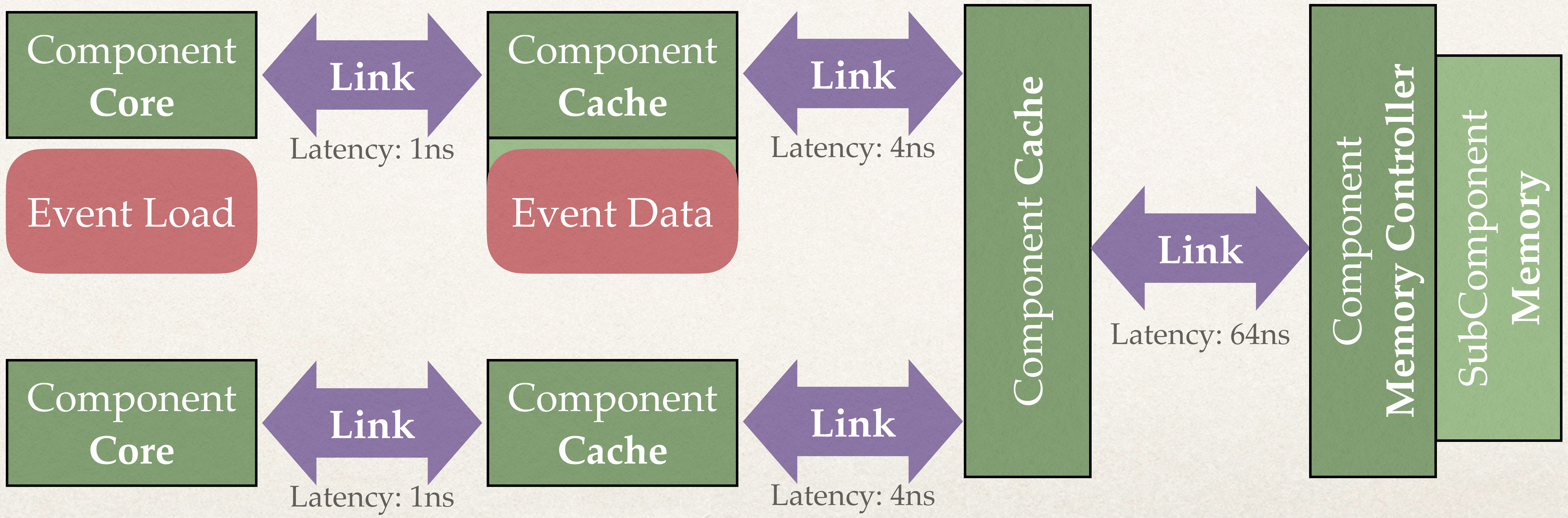
SubComponents and modules

- Add additional functionality to a Component.
  - SubComponent (SC): shares common base class with Component (friend class of component).
  - Module (M): self-contained functionality (no access to Component class functionality).
- Provide modularity.
  - Generic interface which can be used by multiple SC/M.
  - Loaded by Components at runtime .
- Tightly-coupled with a component.
  - Components call SC/M as a C++ class instance.
  - Do not need to communicate via Links.
  - SC/M cannot exist by themselves.
- Example:
  - Miranda core loads a pattern generator subcomponent to generate a sequence of memory operations.



Building Blocks

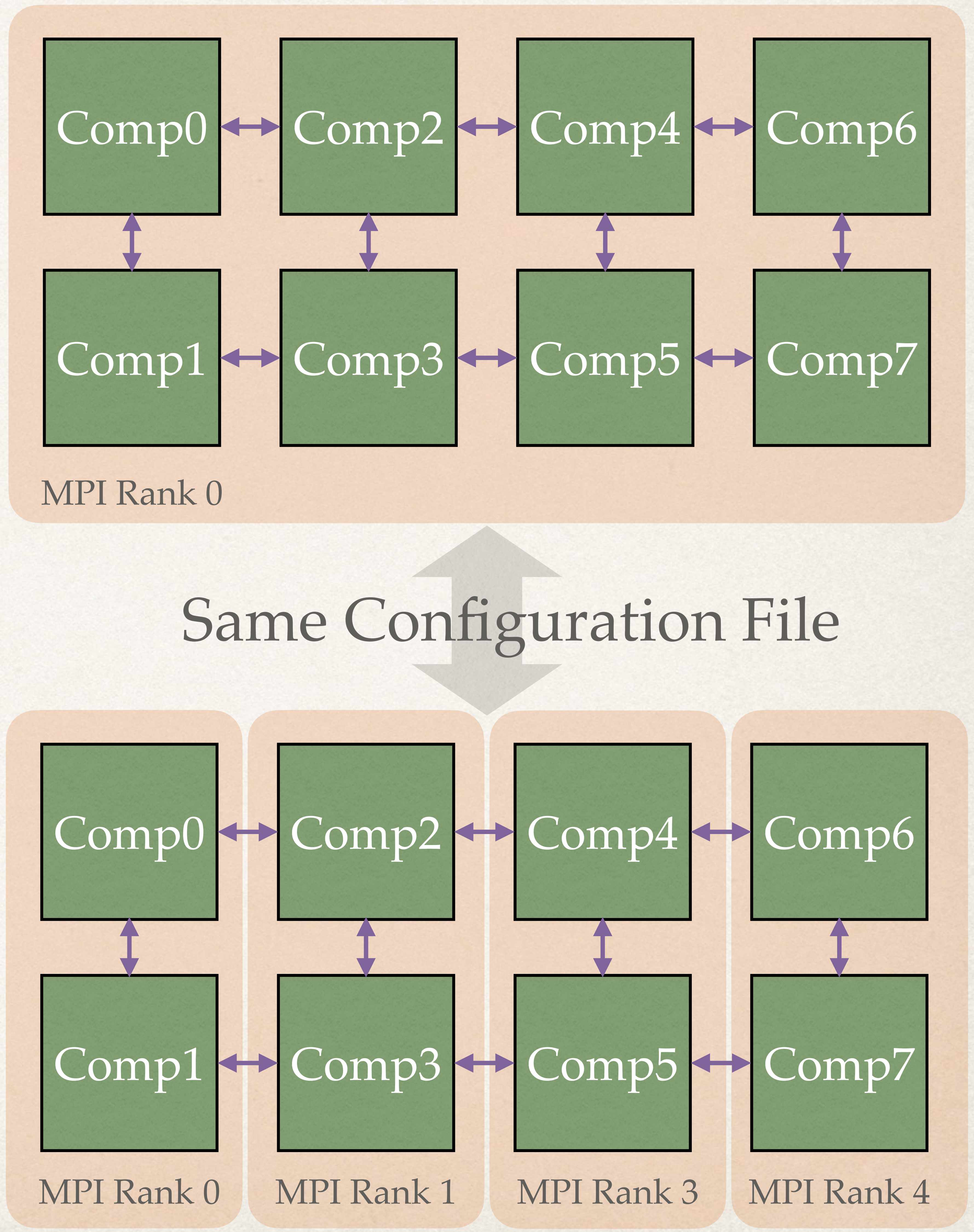
- SST simulations are comprised of **components** connected by **links**.
- Components communicate by sending **events** over links.
- Components define **ports** which are valid connection points for a link.
- Each link has a minimum latency.
- Components can use **subComponents** and **modules** for additional/customizable functionality.



SST in Parallel

- SST was designed from the ground up to enable scalable, parallel simulations.
- Components are distributed among MPI ranks and/or threads.
- Links allow parallelism:
  - Components should communicate via links ONLY.
  - Transparently handle any MPI/thread communication.
  - Specified link-latency determines synchronization rate.

```
# Two ranks
$ mpirun -np 2 sst demo.py
# Two threads
$ sst -n 2 demo.py
# Two ranks with two threads
$ mirin -np 2 sst -n 2 demo.py
```



Simulation Lifecycle

Birth

- Create Graph
- Partition
- Instantiate
- Connect
- Initialize
- Setup

Life

- Send Events
- Manage Clock and Event Handlers

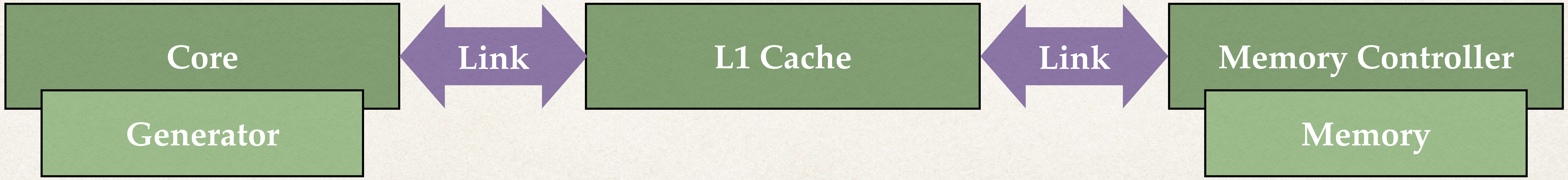
Death

- Finalize
- Output
- Cleanup

Simulating with SST

Simulating with SST

- Element libraries in the simulation:
  - **Miranda** - Simple core model that runs generated instruction streams.
    - **Generators** (SubComponents) produce memory access patterns, e.g., GUPSGenerator, RandomGenerator, STREAMBenchGenerator, etc.
  - **memHierarchy** - Various cache/memory system related subcomponents and modules.
    - **Cache** (Component) with coherence protocol SubComponent.
    - **Memory Controller** (Component) that loads a memory timing model (SubComponent).



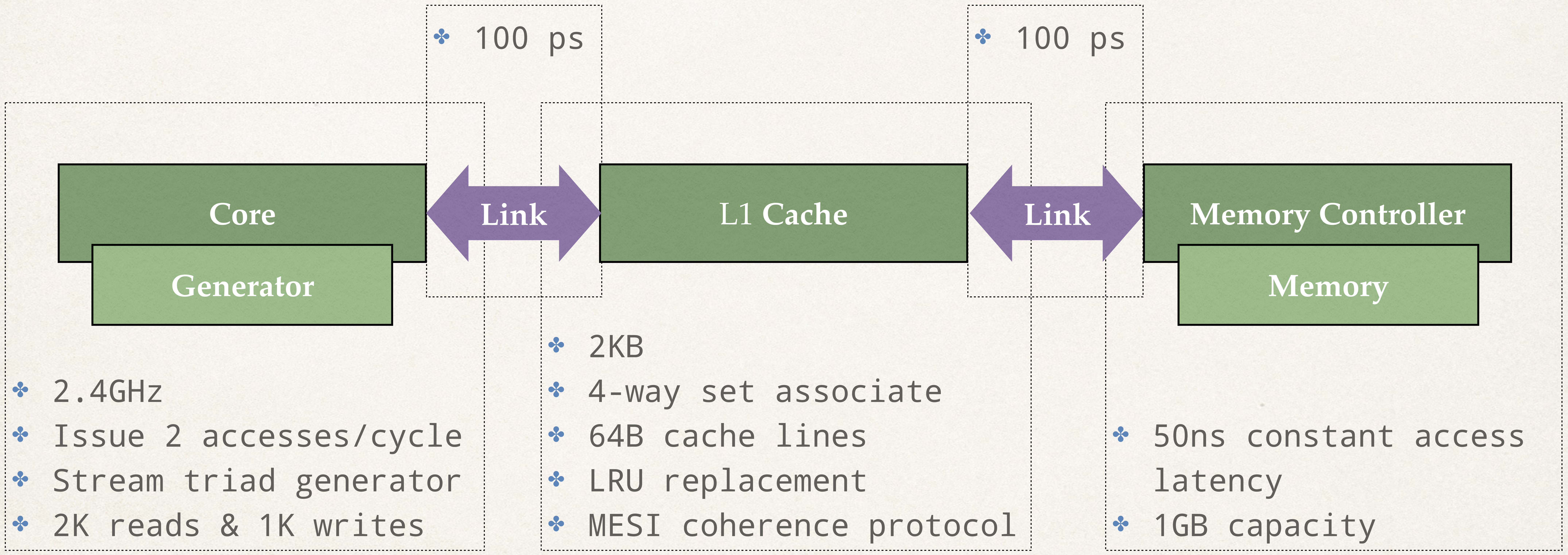
SST-Info

```
SubComponent 7: STREAMBenchGenerator
       Interface: SST::Miranda::RequestGenerator
          NUM STATISTICS = 0
          NUM PORTS = 0
          NUM SUBCOMPONENT SLOTS = 0
          NUM PARAMETERS = 7
             PARAMETER 0 = verbose (Sets the verbosity output of the generator) [0]
             PARAMETER 1 = n (Sets the number of elements in the STREAM arrays) [10000]
             PARAMETER 2 = n_per_call (Sets the number of iterations to generate per call to the generation function) [1]
             PARAMETER 3 = operandwidth (Sets the length of the request, default=8 (i.e. one double)) [8]
             PARAMETER 4 = start_a (Sets the start address of the array a) [0]
             PARAMETER 5 = start_b (Sets the start address of the array b) [1024]
             PARAMETER 6 = start_c (Sets the start address of the array c) [2048]
    STREAMBenchGenerator: Creates a representation of the STREAM benchmark
    Using ELI version 0.9.0
    Compiled on: Apr 13 2022 12:48:52, using file: generators/streambench.h
```

“**sst-info** element_name” to extract element and component information.

```
Stream Triad Kernel:
a[i] = b[i] + c[i] * SCALAR;
```

Simulating with SST



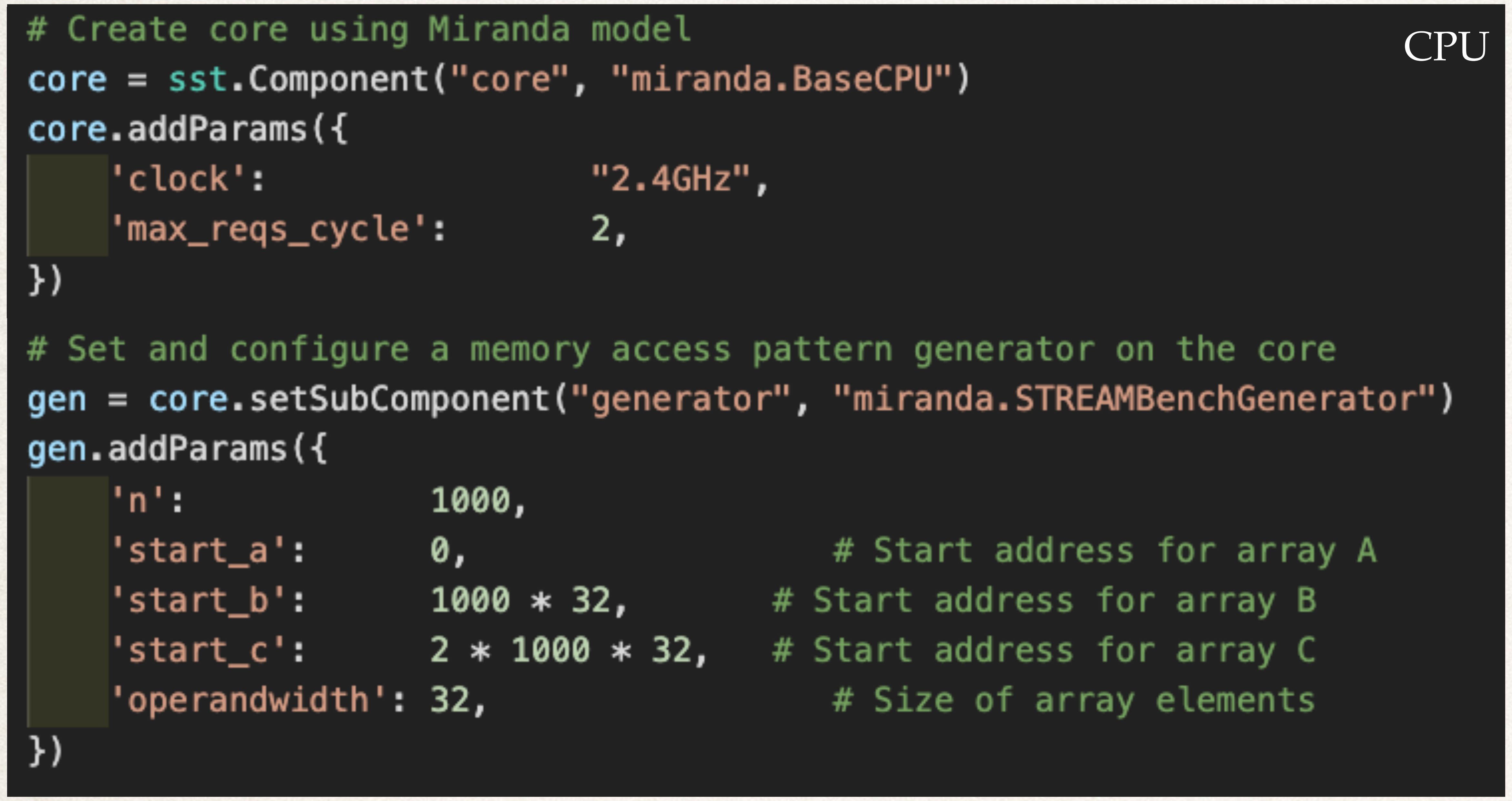
- 2.4GHz
- Issue 2 accesses/cycle
- Stream triad generator
- 2K reads & 1K writes

- 2KB
- 4-way set associate
- 64B cache lines
- LRU replacement
- MESI coherence protocol

- 50ns constant access latency
- 1GB capacity

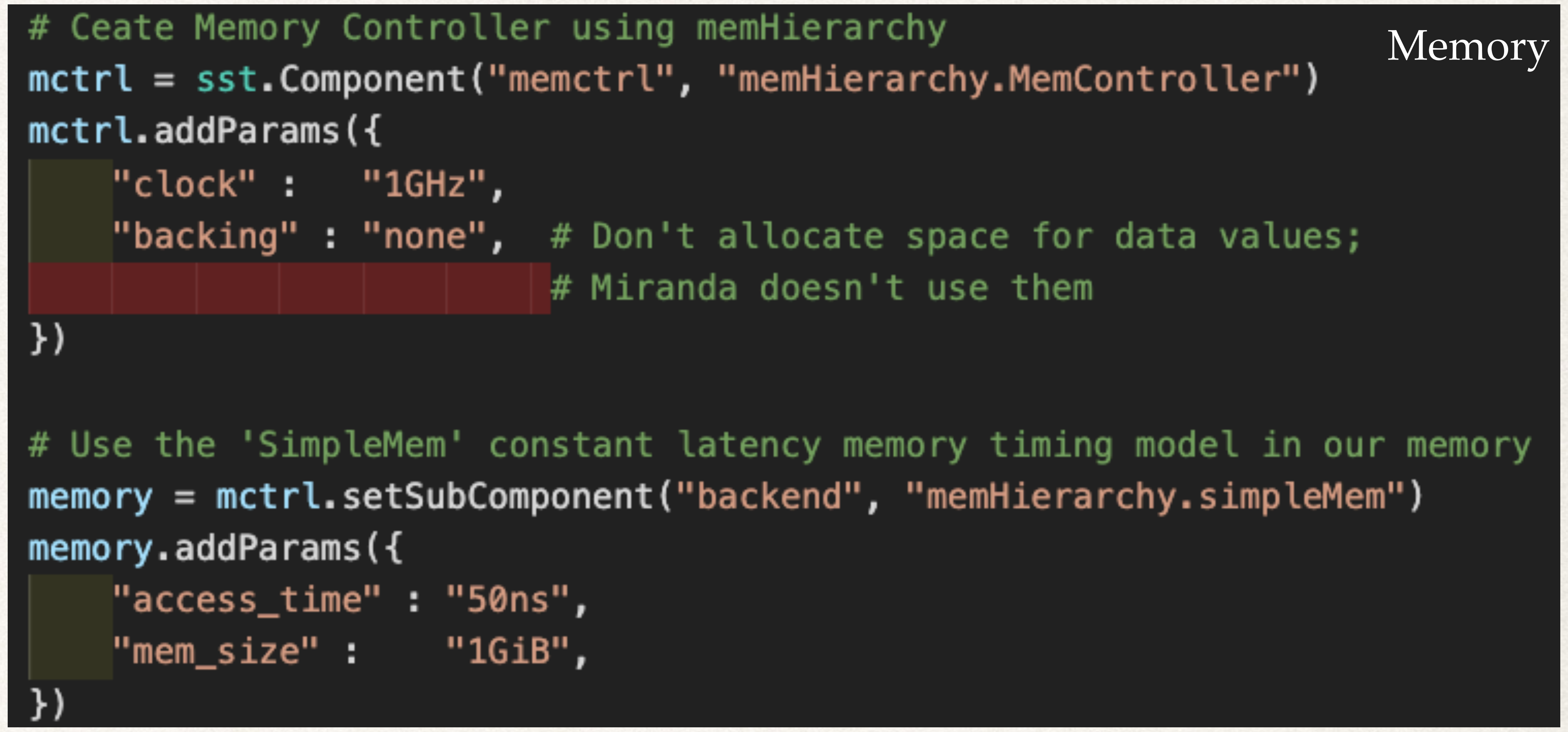
```
Stream Triad Kernel:
a[i] = b[i] + c[i] * SCALAR;
```

Configuration



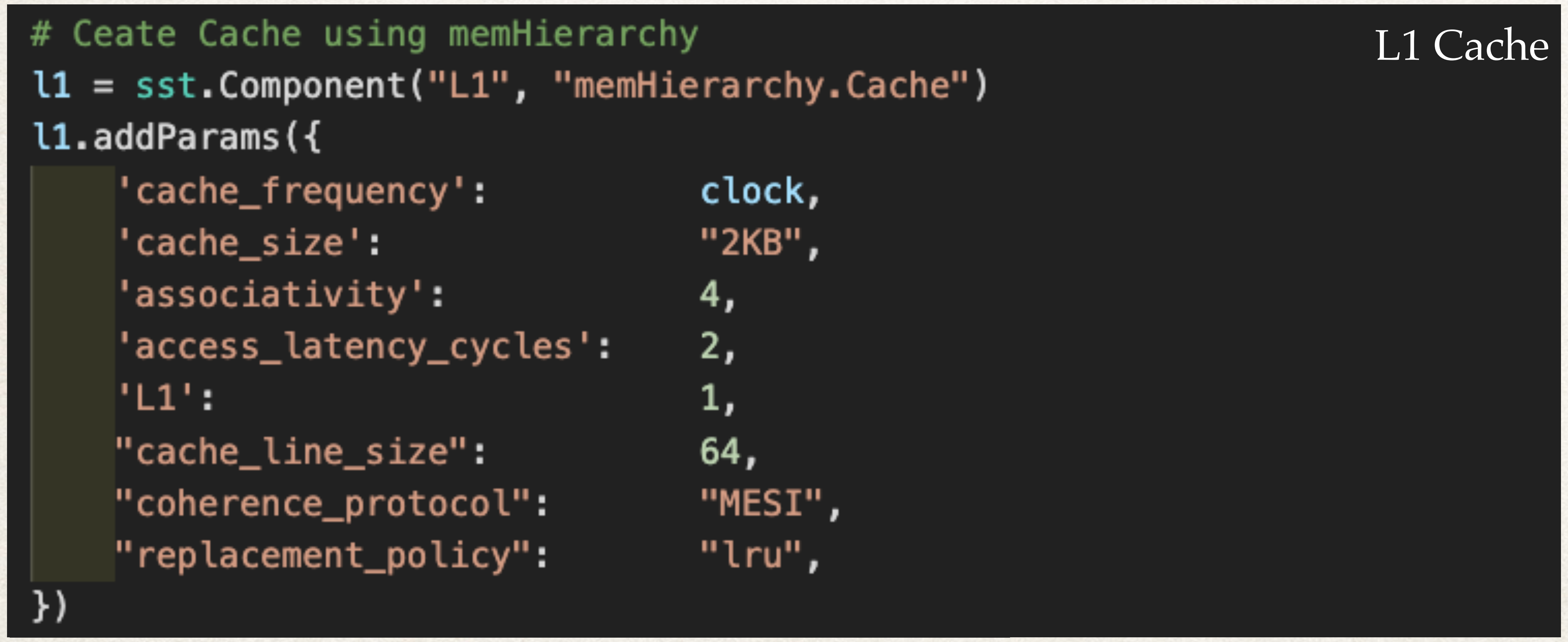
CPU

```python
# Create core using Miranda model
core = sst.Component("core", "miranda.BaseCPU")
core.addParams({
    'clock':             "2.4GHz",
    'max_reqs_cycle':    2,
})

# Set and configure a memory access pattern generator on the core
gen = core.setSubComponent("generator", "miranda.STREAMBenchGenerator")
gen.addParams({
    'n':            1000,
    'start_a':      0,                    # Start address for array A
    'start_b':      1000 * 32,        # Start address for array B
    'start_c':      2 * 1000 * 32,    # Start address for array C
    'operandwidth': 32,                   # Size of array elements
})
```



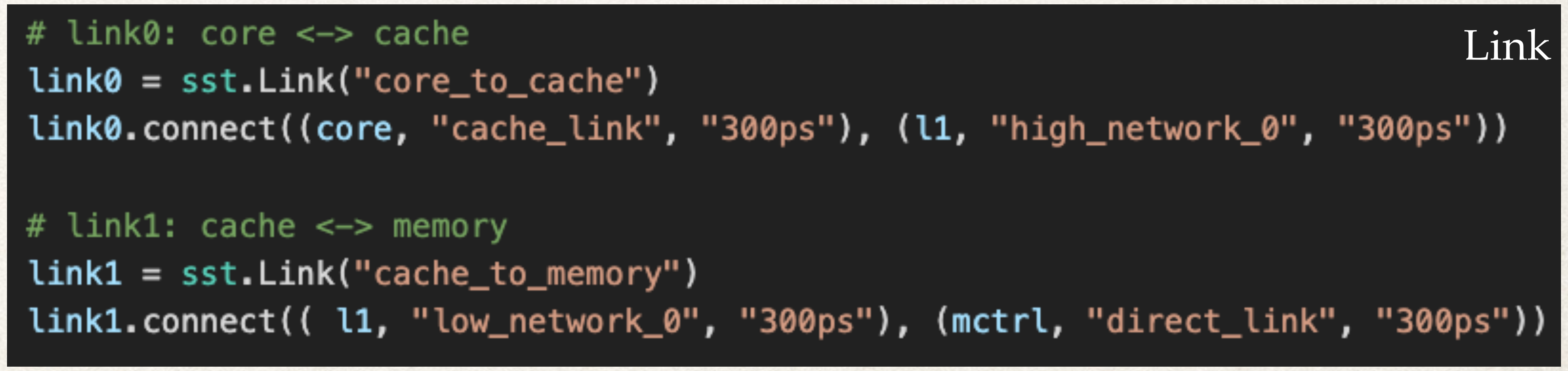
Memory

```python
# Ceate Memory Controller using memHierarchy
mctrl = sst.Component("memctrl", "memHierarchy.MemController")
mctrl.addParams({
    "clock" :   "1GHz",
    "backing" : "none",  # Don't allocate space for data values;
                         # Miranda doesn't use them
})

# Use the 'SimpleMem' constant latency memory timing model in our memory
memory = mctrl.setSubComponent("backend", "memHierarchy.simpleMem")
memory.addParams({
    "access_time" : "50ns",
    "mem_size" :    "1GiB",
})
```



L1 Cache

```python
# Ceate Cache using memHierarchy
l1 = sst.Component("L1", "memHierarchy.Cache")
l1.addParams({
    'cache_frequency':         clock,
    'cache_size':              "2KB",
    'associativity':           4,
    'access_latency_cycles':   2,
    'L1':                      1,
    "cache_line_size":         64,
    "coherence_protocol":      "MESI",
    "replacement_policy":      "lru",
})
```



Link

```python
# link0: core <-> cache
link0 = sst.Link("core_to_cache")
link0.connect((core, "cache_link", "300ps"), (l1, "high_network_0", "300ps"))

# link1: cache <-> memory
link1 = sst.Link("cache_to_memory")
link1.connect(( l1, "low_network_0", "300ps"), (mctrl, "direct_link", "300ps"))
```

Enabling statistics

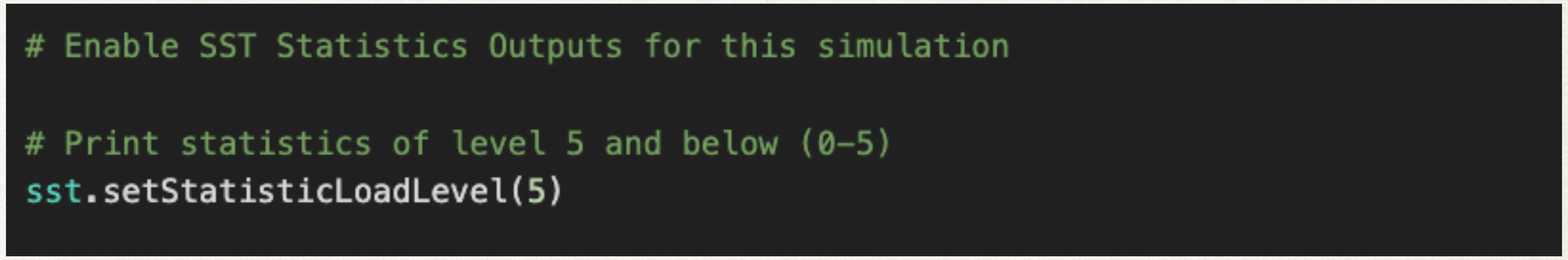
Most Components and SubComponents define statistics.

$ sst-info miranda

```
ELEMENT 0 = miranda ()
Num Components = 1
      Component 0: BaseCPU
      CATEGORY: PROCESSOR COMPONENT
         NUM STATISTICS = 17
            STATISTIC 0 = split_read_reqs [Number of read requests split over a cache line boundary] (requests) Enable level = 2
            STATISTIC 1 = split_write_reqs [Number of write requests split over a cache line boundary] (requests) Enable level = 2
            STATISTIC 2 = split_custom_reqs [Number of custom requests split over a cache line boundary] (requests) Enable level = 2
            STATISTIC 3 = read_reqs [Number of read requests issued] (requests) Enable level = 1
            STATISTIC 4 = write_reqs [Number of write requests issued] (requests) Enable level = 1
            STATISTIC 5 = custom_reqs [Number of custom requests issued] (requests) Enable level = 1
            STATISTIC 6 = total_bytes_read [Count the total bytes requested by read operations] (bytes) Enable level = 1
            STATISTIC 7 = total_bytes_write [Count the total bytes requested by write operations] (bytes) Enable level = 1
            STATISTIC 8 = total_bytes_custom [Count the total bytes requested by custom operations] (bytes) Enable level = 1
            STATISTIC 9 = req_latency [Running total of all latency for all requests] (ns) Enable level = 2
            STATISTIC 10 = cycles_with_issue [Number of cycles which CPU was able to issue requests] (cycles) Enable level = 1
            STATISTIC 11 = cycles_no_issue [Number of cycles which CPU was not able to issue requests] (cycles) Enable level = 1
            STATISTIC 12 = time [Nanoseconds spent issuing requests] (ns) Enable level = 1
            STATISTIC 13 = cycles_hit_fence [Number of issue cycles which stop issue at a fence] (cycles) Enable level = 2
            STATISTIC 14 = cycles_max_reorder [Number of issue cycles which hit maximum reorder lookup] (cycles) Enable level = 2
            STATISTIC 15 = cycles_max_issue [Cycles with maximum operation issue] (cycles) Enable level = 2
            STATISTIC 16 = cycles [Cycles executed] (cycles) Enable level = 1
```

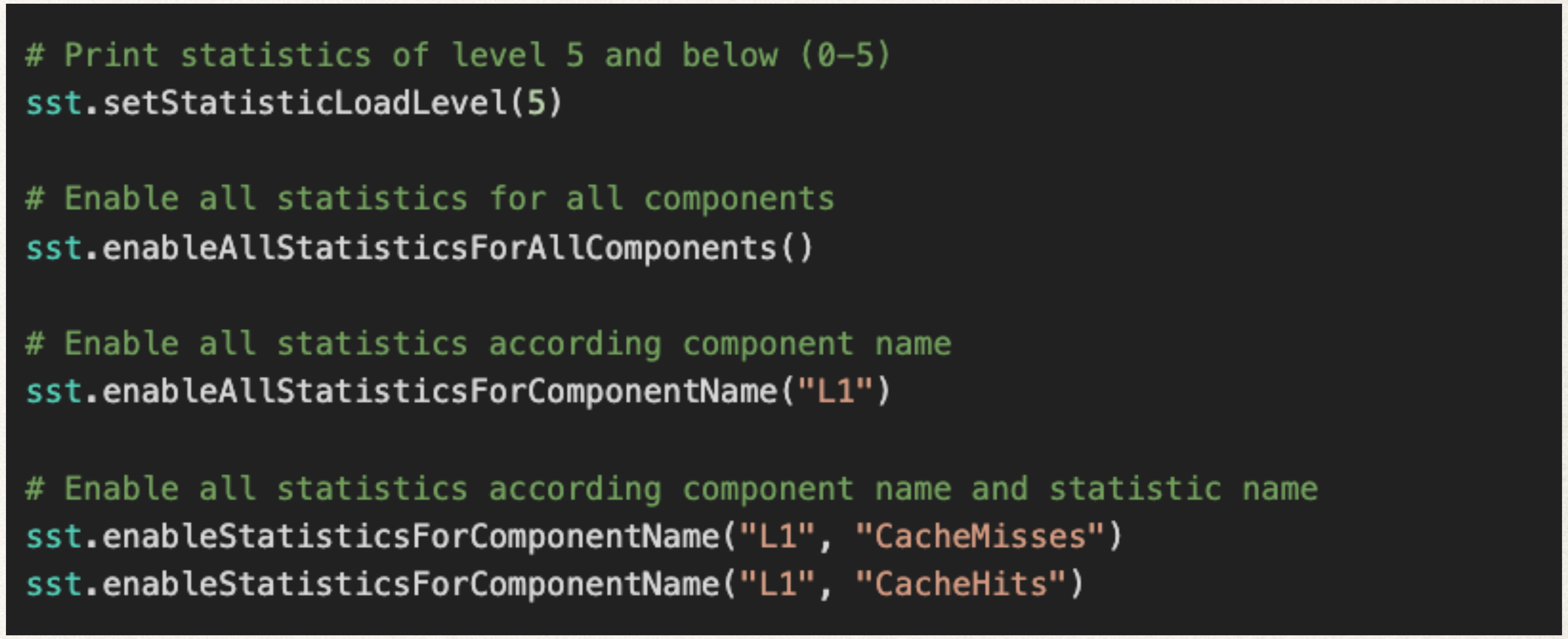
Enabling statistics

Set Statistics Level



```
# Enable SST Statistics Outputs for this simulation

# Print statistics of level 5 and below (0-5)
sst.setStatisticLoadLevel(5)
```

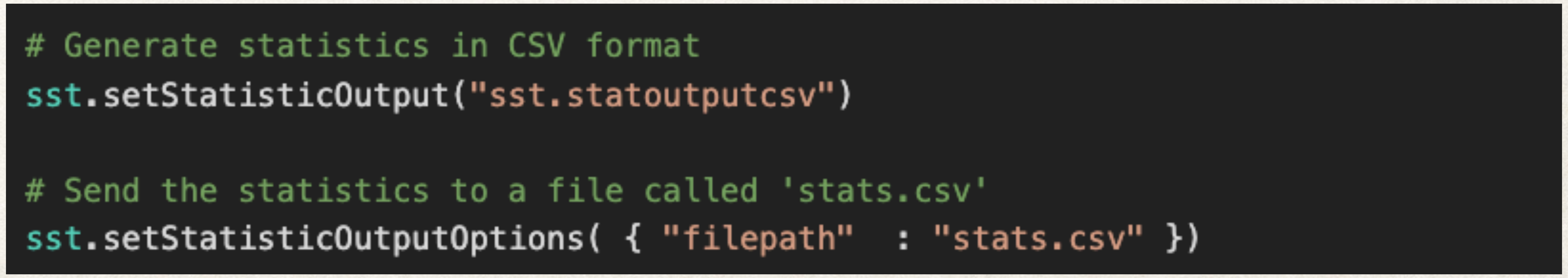
Select Statistics of Components



```
# Print statistics of level 5 and below (0-5)
sst.setStatisticLoadLevel(5)

# Enable all statistics for all components
sst.enableAllStatisticsForAllComponents()

# Enable all statistics according component name
sst.enableAllStatisticsForComponentName("L1")

# Enable all statistics according component name and statistic name
sst.enableStatisticsForComponentName("L1", "CacheMisses")
sst.enableStatisticsForComponentName("L1", "CacheHits")
```

Configure Statistic output

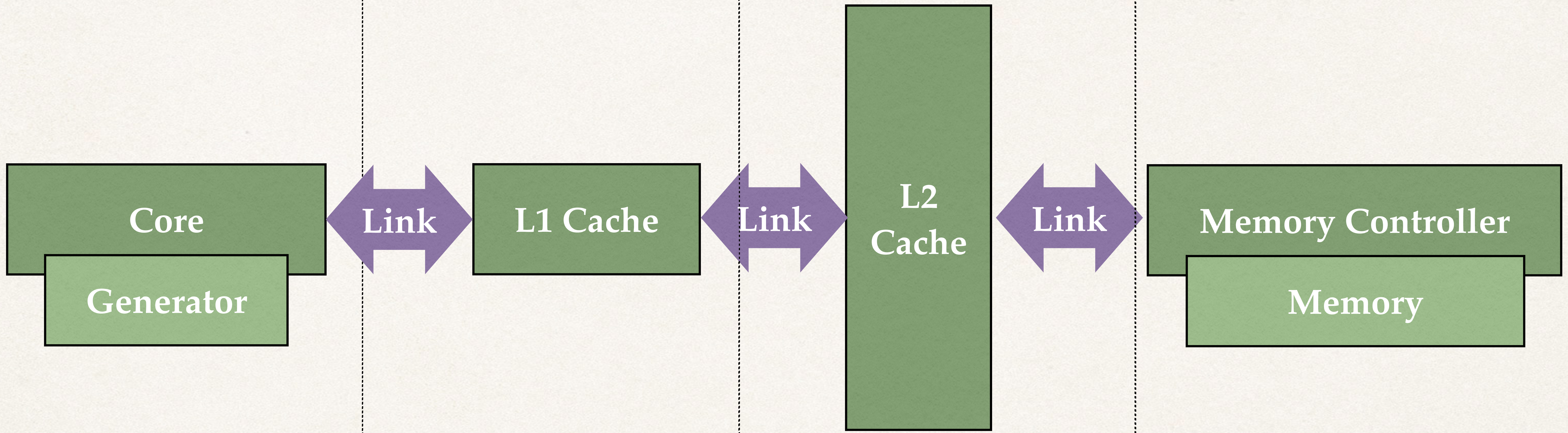
Supported outputs: console, txt, json, hdf5



```
# Generate statistics in CSV format
sst.setStatisticOutput("sst.statoutputcsv")

# Send the statistics to a file called 'stats.csv'
sst.setStatisticOutputOptions( { "filepath"  : "stats.csv" })
```

Demo

Simulating with SST - demo2



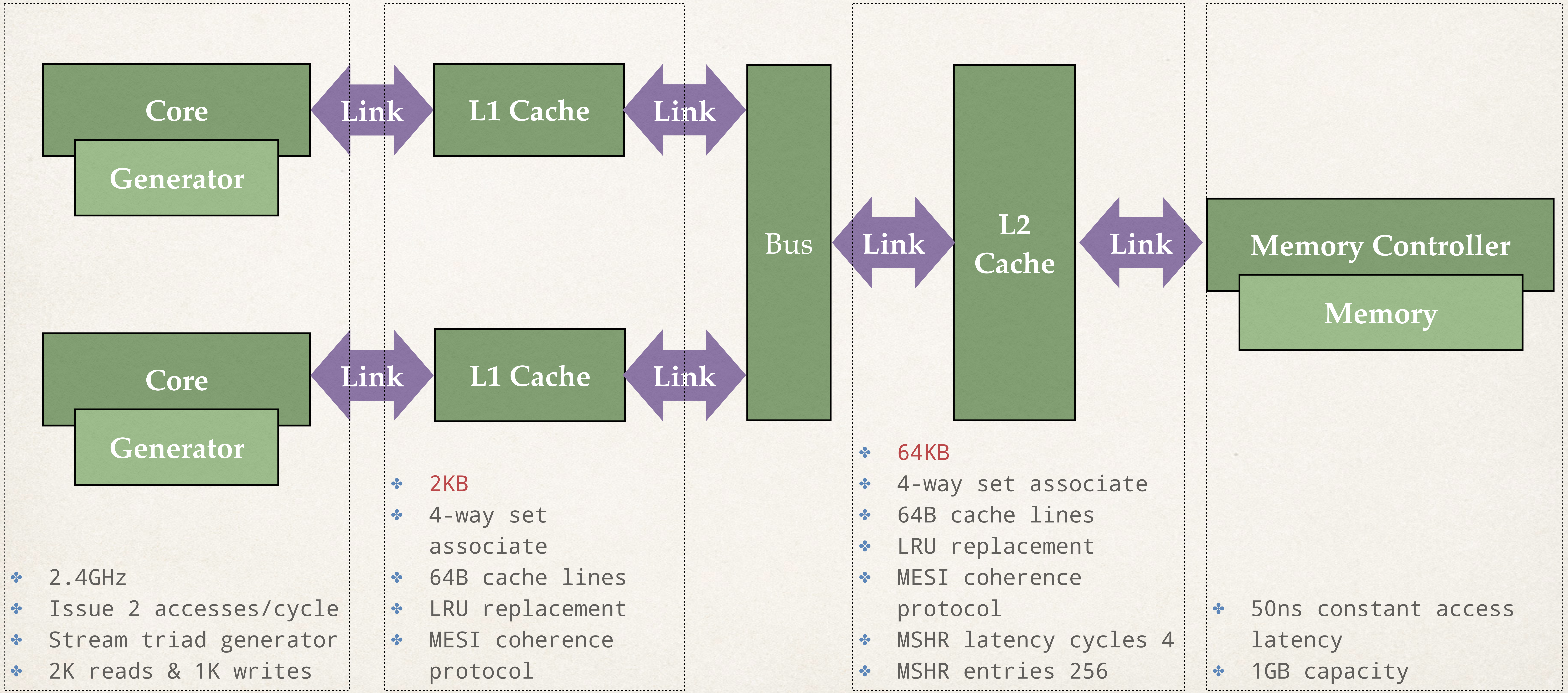
- 2.4GHz
- Issue 2 accesses/cycle
- Stream triad generator
- 2K reads & 1K writes

- 2KB
- 4-way set associate
- 64B cache lines
- LRU replacement
- MESI coherence protocol

- 64KB
- 4-way set associate
- 64B cache lines
- LRU replacement
- MESI coherence protocol
- MSHR latency cycles 4
- MSHR entries 256

- 50ns constant access latency
- 1GB capacity

Simulating with SST - demo3



- 2.4GHz
- Issue 2 accesses/cycle
- Stream triad generator
- 2K reads & 1K writes

- 2KB
- 4-way set associate
- 64B cache lines
- LRU replacement
- MESI coherence protocol

- 64KB
- 4-way set associate
- 64B cache lines
- LRU replacement
- MESI coherence protocol
- MSHR latency cycles 4
- MSHR entries 256

- 50ns constant access latency
- 1GB capacity

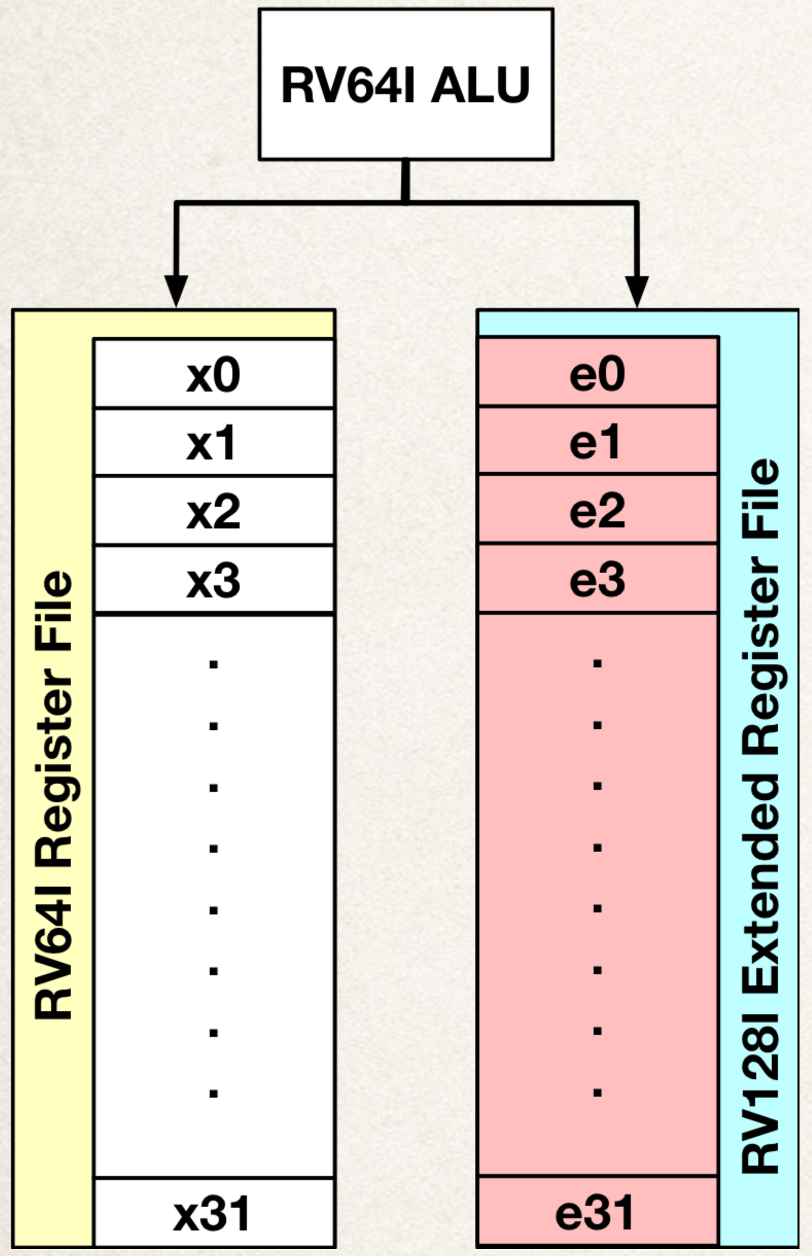
SST for xBGAS

Rev — RISC-V CPU Model

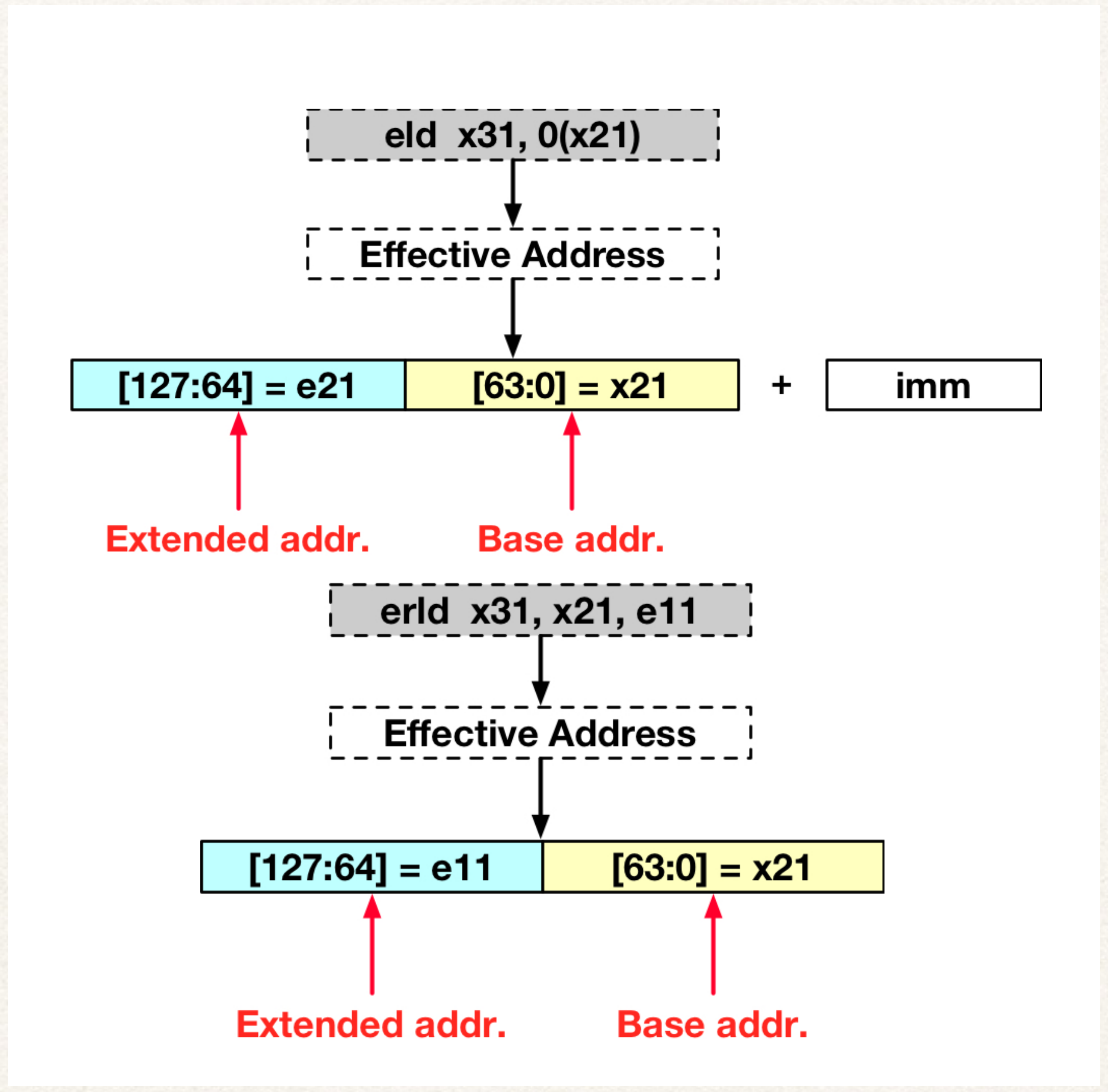
- Rev SST component:
  - Designed to provide cycle-based simulation capabilities of **RISC-V** core/cores.
  - Built upon the standard SST "core" component libraries.
  - Can be attached to any other existing SST component for full system simulations.
  - Provides the ability to **load compiled binaries** (ELF binaries).
  - Supports for **custom instruction extension**.

Towards xBGAS simulation

- Register files

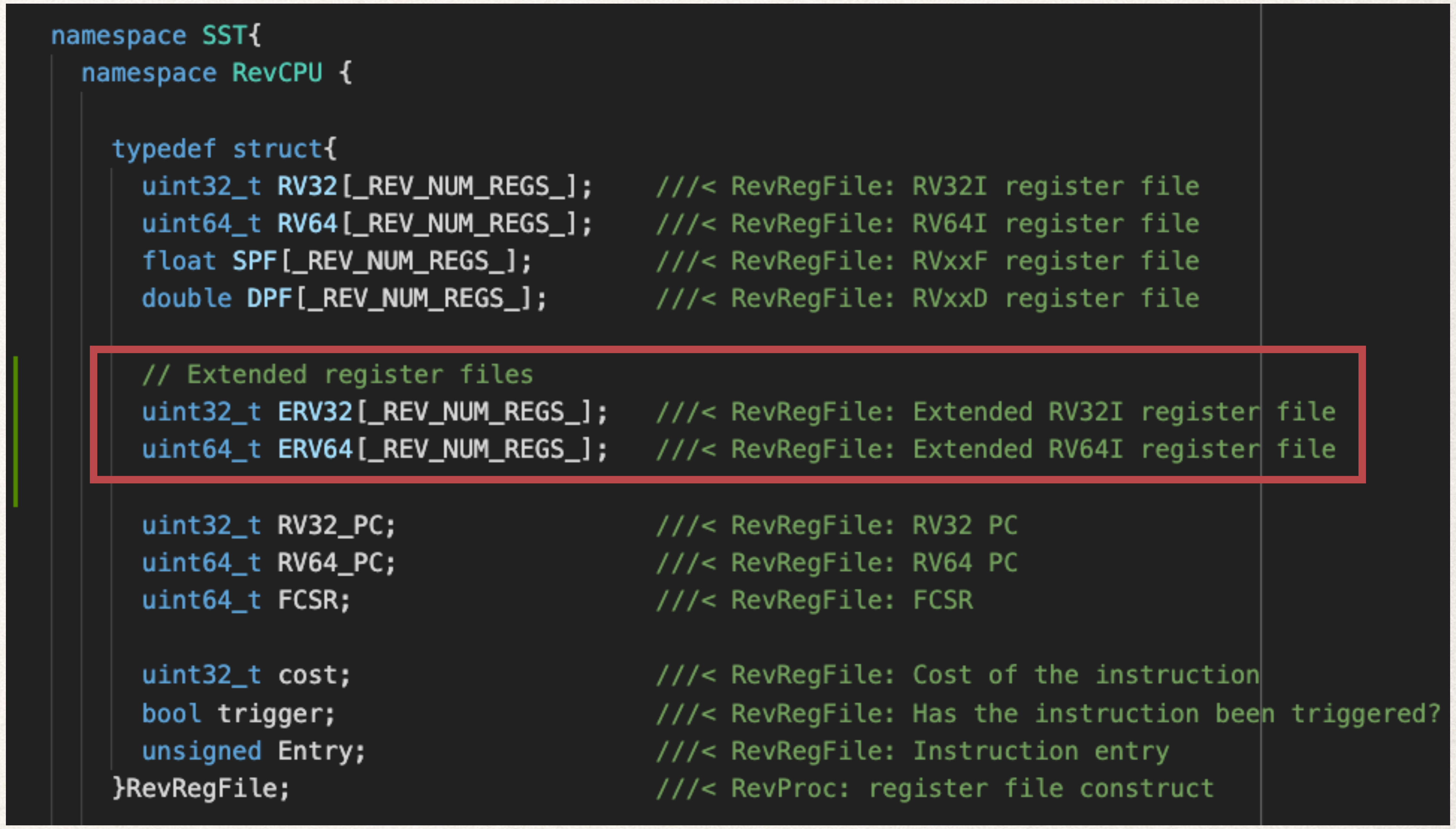


Extended Registers



xBGAS Addressing Model

Rev/src/RevInstTable.h

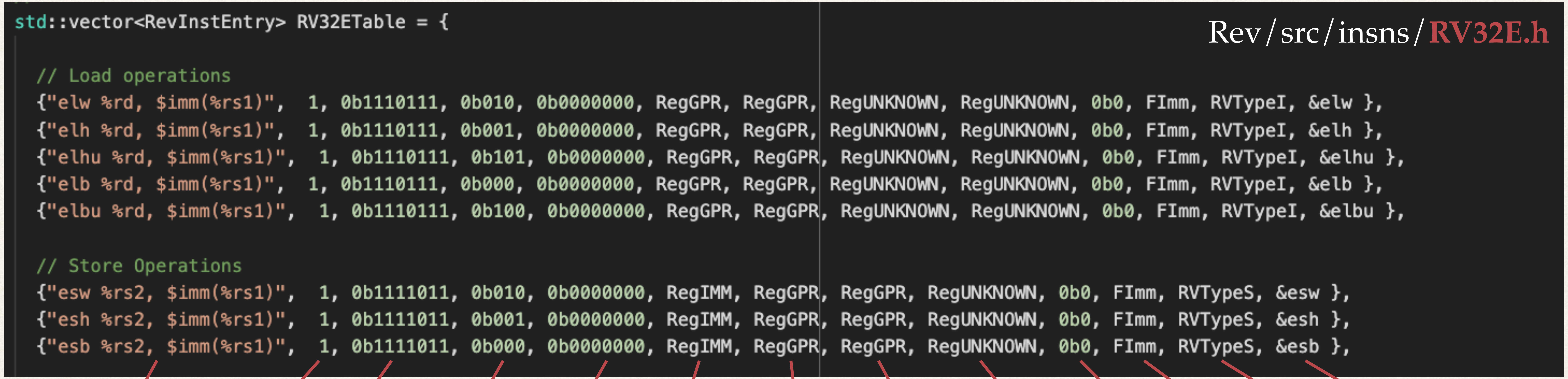


```
namespace SST{
  namespace RevCPU {

    typedef struct{
      uint32_t RV32[_REV_NUM_REGS_];    ///< RevRegFile: RV32I register file
      uint64_t RV64[_REV_NUM_REGS_];    ///< RevRegFile: RV64I register file
      float SPF[_REV_NUM_REGS_];        ///< RevRegFile: RVxxF register file
      double DPF[_REV_NUM_REGS_];       ///< RevRegFile: RVxxD register file

      // Extended register files
      uint32_t ERV32[_REV_NUM_REGS_];   ///< RevRegFile: Extended RV32I register file
      uint64_t ERV64[_REV_NUM_REGS_];   ///< RevRegFile: Extended RV64I register file

      uint32_t RV32_PC;                 ///< RevRegFile: RV32 PC
      uint64_t RV64_PC;                 ///< RevRegFile: RV64 PC
      uint64_t FCSR;                    ///< RevRegFile: FCSR

      uint32_t cost;                    ///< RevRegFile: Cost of the instruction
      bool trigger;                     ///< RevRegFile: Has the instruction been triggered?
      unsigned Entry;                   ///< RevRegFile: Instruction entry
    }RevRegFile;                        ///< RevProc: register file construct
```

Register file struct in RevCPU

Towards xBGAS simulation

- Instruction encoding tables



Rev/src/insns/RV32E.h

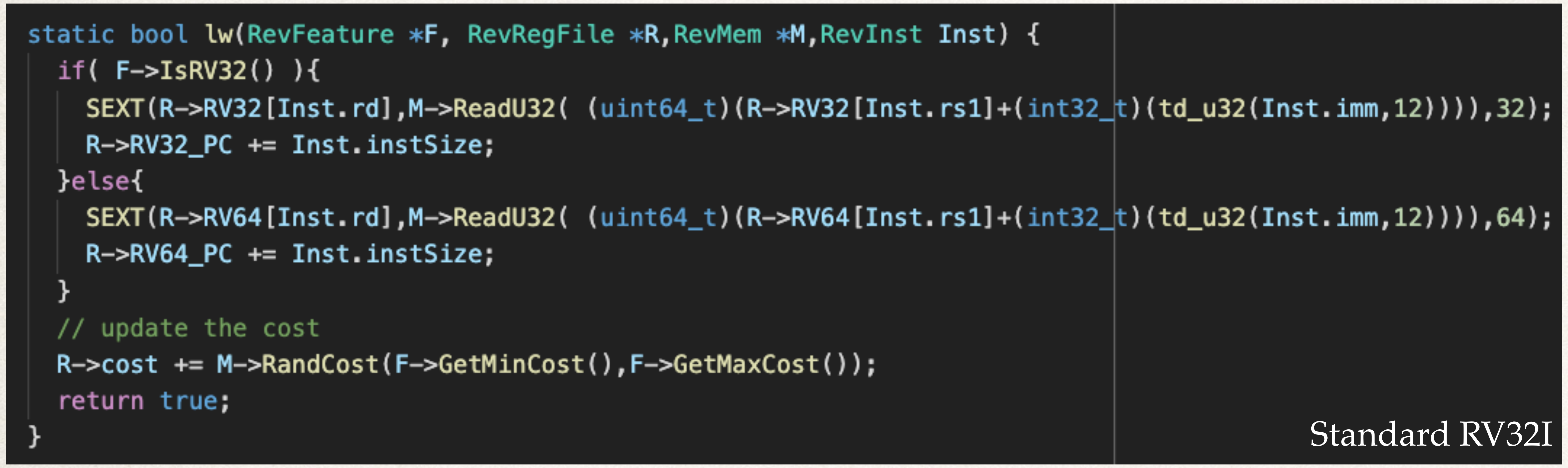
```
std::vector<RevInstEntry> RV32ETable = {

  // Load operations
  {"elw %rd, $imm(%rs1)",  1, 0b1110111, 0b010, 0b0000000, RegGPR, RegGPR, RegUNKNOWN, RegUNKNOWN, 0b0, FImm, RVTypeI, &elw },
  {"elh %rd, $imm(%rs1)",  1, 0b1110111, 0b001, 0b0000000, RegGPR, RegGPR, RegUNKNOWN, RegUNKNOWN, 0b0, FImm, RVTypeI, &elh },
  {"elhu %rd, $imm(%rs1)",  1, 0b1110111, 0b101, 0b0000000, RegGPR, RegGPR, RegUNKNOWN, RegUNKNOWN, 0b0, FImm, RVTypeI, &elhu },
  {"elb %rd, $imm(%rs1)",  1, 0b1110111, 0b000, 0b0000000, RegGPR, RegGPR, RegUNKNOWN, RegUNKNOWN, 0b0, FImm, RVTypeI, &elb },
  {"elbu %rd, $imm(%rs1)",  1, 0b1110111, 0b100, 0b0000000, RegGPR, RegGPR, RegUNKNOWN, RegUNKNOWN, 0b0, FImm, RVTypeI, &elbu },

  // Store Operations
  {"esw %rs2, $imm(%rs1)",  1, 0b1111011, 0b010, 0b0000000, RegIMM, RegGPR, RegGPR, RegUNKNOWN, 0b0, FImm, RVTypeS, &esw },
  {"esh %rs2, $imm(%rs1)",  1, 0b1111011, 0b001, 0b0000000, RegIMM, RegGPR, RegGPR, RegUNKNOWN, 0b0, FImm, RVTypeS, &esh },
  {"esb %rs2, $imm(%rs1)",  1, 0b1111011, 0b000, 0b0000000, RegIMM, RegGPR, RegGPR, RegUNKNOWN, 0b0, FImm, RVTypeS, &esb },
```

Mnemonic, cost, opcode, funct3, funct7, <u>rdClass, rs1Class, rs2Class, rs3Class</u>, imm12, imm, format, func

Define the register class for the corresponding slot.
Possible values: RegGPR, RegCSR, RegIMM, RegUNKNOWN
RegUNKNOWN if the field is not utilized.

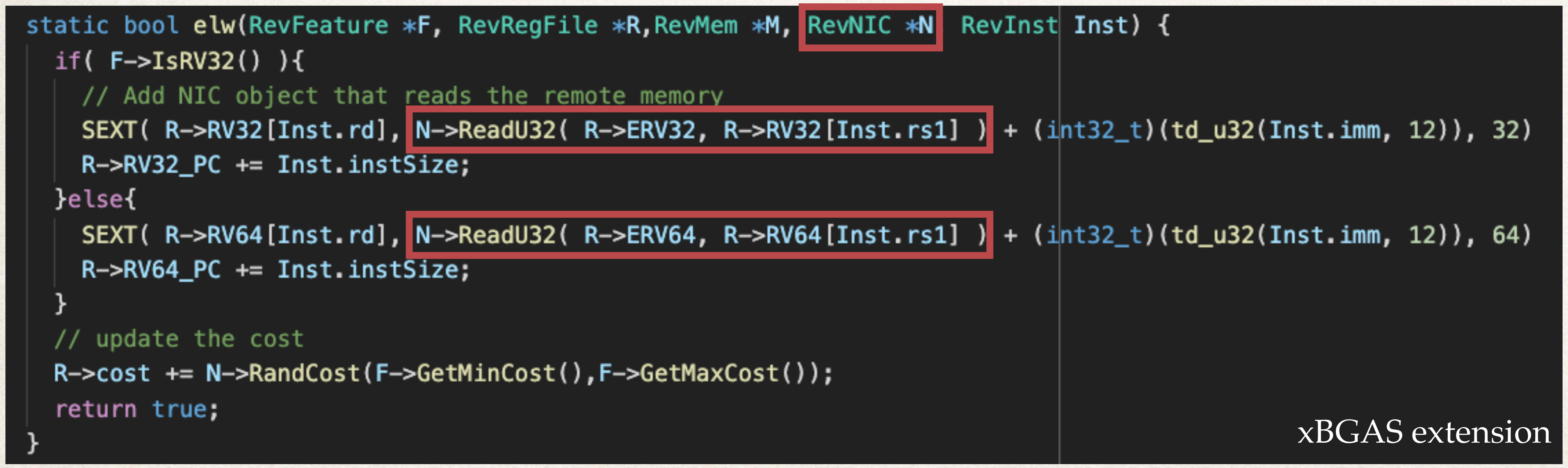
Towards xBGAS simulation

- Instruction function implementation



```
static bool lw(RevFeature *F, RevRegFile *R,RevMem *M,RevInst Inst) {
  if( F->IsRV32() ){
    SEXT(R->RV32[Inst.rd],M->ReadU32( (uint64_t)(R->RV32[Inst.rs1]+(int32_t)(td_u32(Inst.imm,12)))),32);
    R->RV32_PC += Inst.instSize;
  }else{
    SEXT(R->RV64[Inst.rd],M->ReadU32( (uint64_t)(R->RV64[Inst.rs1]+(int32_t)(td_u32(Inst.imm,12)))),64);
    R->RV64_PC += Inst.instSize;
  }
  // update the cost
  R->cost += M->RandCost(F->GetMinCost(),F->GetMaxCost());
  return true;
}
```

- Utilize the **RevMem** object to read the desired number of bytes via the ReadU32 routine.



```
static bool elw(RevFeature *F, RevRegFile *R,RevMem *M, RevNIC *N  RevInst Inst) {
  if( F->IsRV32() ){
    // Add NIC object that reads the remote memory
    SEXT( R->RV32[Inst.rd], N->ReadU32( R->ERV32, R->RV32[Inst.rs1] ) + (int32_t)(td_u32(Inst.imm, 12)), 32)
    R->RV32_PC += Inst.instSize;
  }else{
    SEXT( R->RV64[Inst.rd], N->ReadU32( R->ERV64, R->RV64[Inst.rs1] ) + (int32_t)(td_u32(Inst.imm, 12)), 64)
    R->RV64_PC += Inst.instSize;
  }
  // update the cost
  R->cost += N->RandCost(F->GetMinCost(),F->GetMaxCost());
  return true;
}
```

- Utilize the **RevNIC** object to read the desired number of bytes via the ReadU32 routine.
- Read routines are not exist in RevNIC.
- NLB (Namespace Lookaside Buffer)
- Initialization of PE id and its corresponding address

https://github.com/tactcomplabs/rev

Summary

- Presented the design of SST and its key objects.
- Simulated a multi-level memory system with three different configurations.
- Discussed the approaches to modify the RevCPU model for supporting xBGAS simulation.

- Will dive deep into the RevNIC component to understand how it works.
- Will implement the remote memory operation routines to support the instruction functions.

- SST is not limited to simulating HPC Systems; SST GitHub repo provides an example of carwash simulation.
  - https://github.com/sstsimulator/sst-elements/tree/master/src/sst/elements/simpleSimulation
  - https://www.youtube.com/watch?v=ipxoVb2dEcw&list=PLgehegDe4T2y1badxrxcuvIsX42V64t2x

Thank you!

Q&A