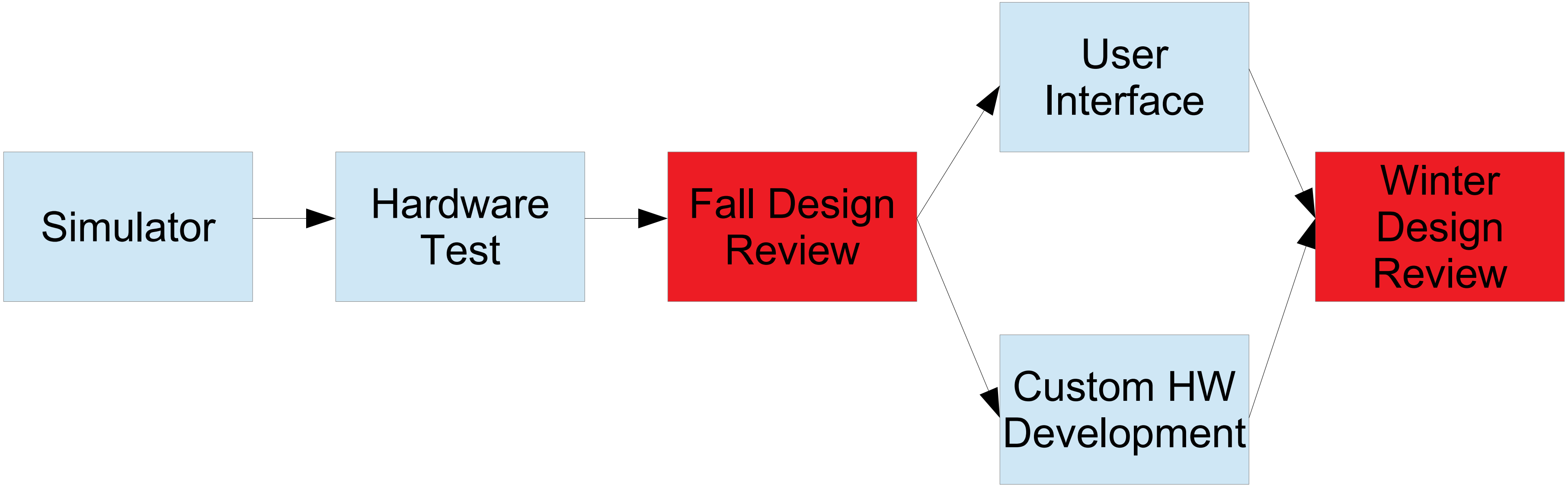
Senior Design Update

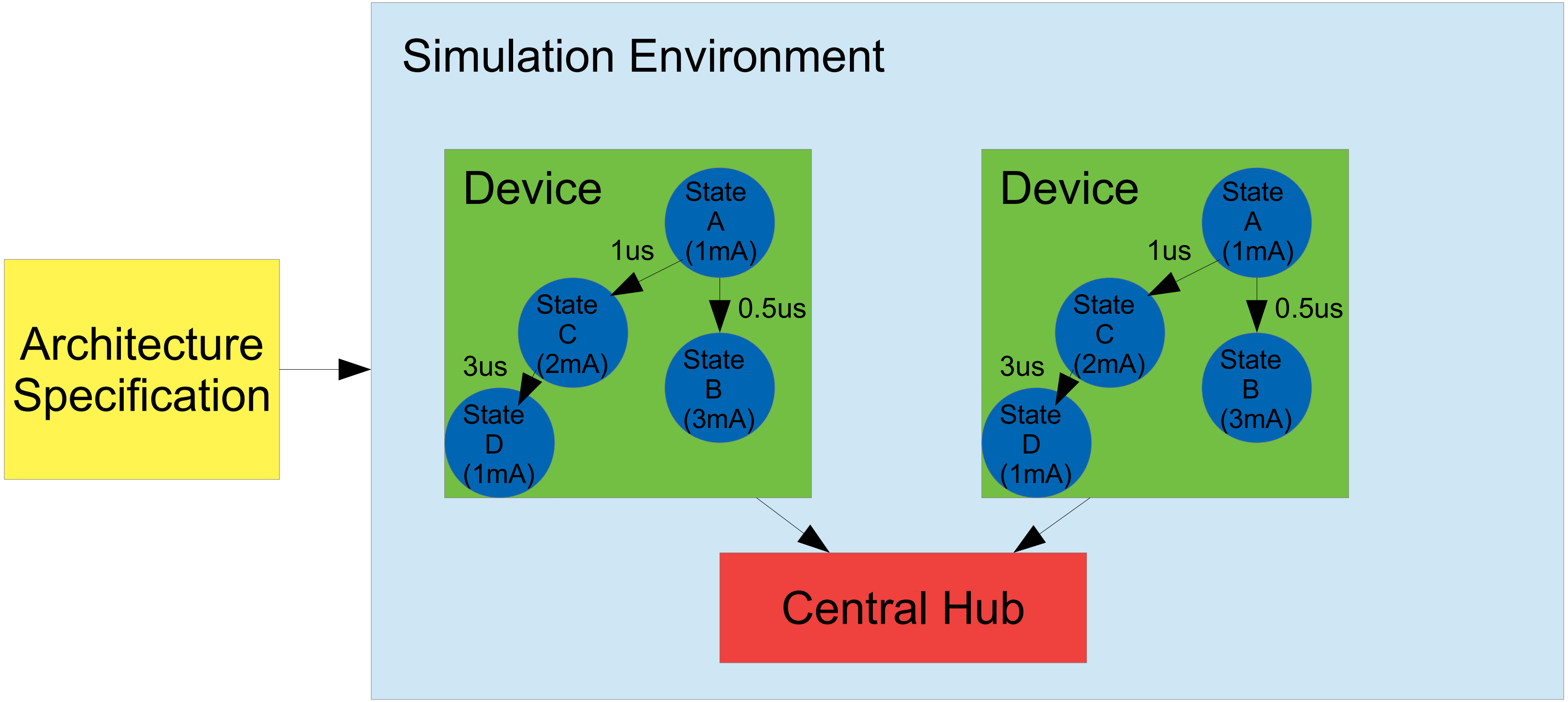
Current Implementation Plan



Simulator

- The current plan is to develop a system-level simulator.
- The simulator has two main applications:
  - Determine a scalable algorithm in a more efficient and automated manner than manually testing small numbers of boards and extrapolating
  - Determine the necessary specifications for custom hardware implementations
  - Compare the performance impact that different pieces of hardware have on the system’s performance
- Some of the simulator code, such as keeping track of the positions of devices, can likely be used in the user-interface later.

Simulator Design

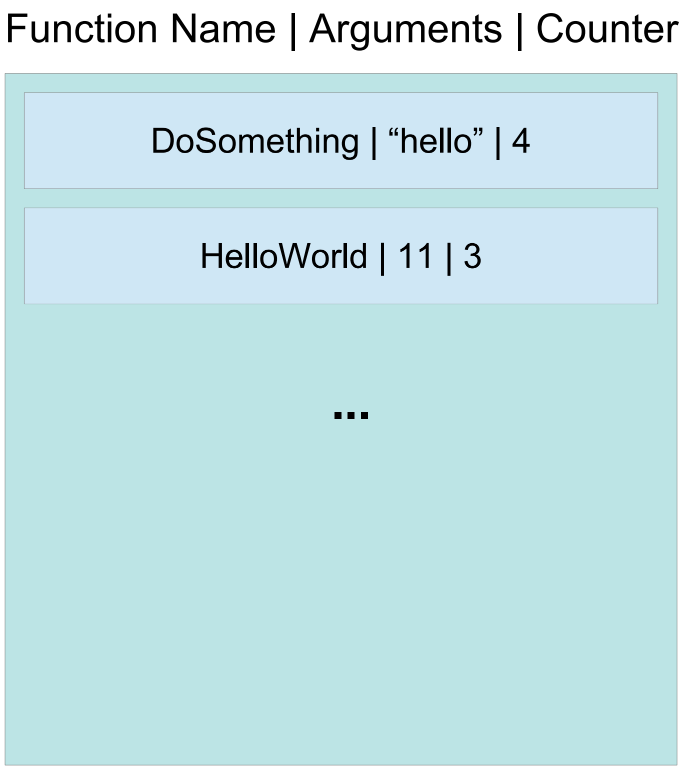


Simulator Design (Cont’d)

- The simulation occurs within a simulation environment that keeps track of the position, battery life, and other properties of the devices in the system.
- The simulation environment knows what kinds of devices are being used and what algorithms they are using from the architecture specification.

Action Queue

- At a very fundamental level, the simulator works by keeping track of the number of timesteps that have passed.
- As the simulator progresses through time, it calls different functions. These called functions determine what actually happens in the simulation.
- An action queue is a FIFO queue that keeps track of the different functions that need to be called in order.
- The decrement() function associated with this class, decrements the counter of ONLY the element at the top of the queue. When the counter reaches 0, the function and its associated arguments are called.



Function Name | Arguments | Counter

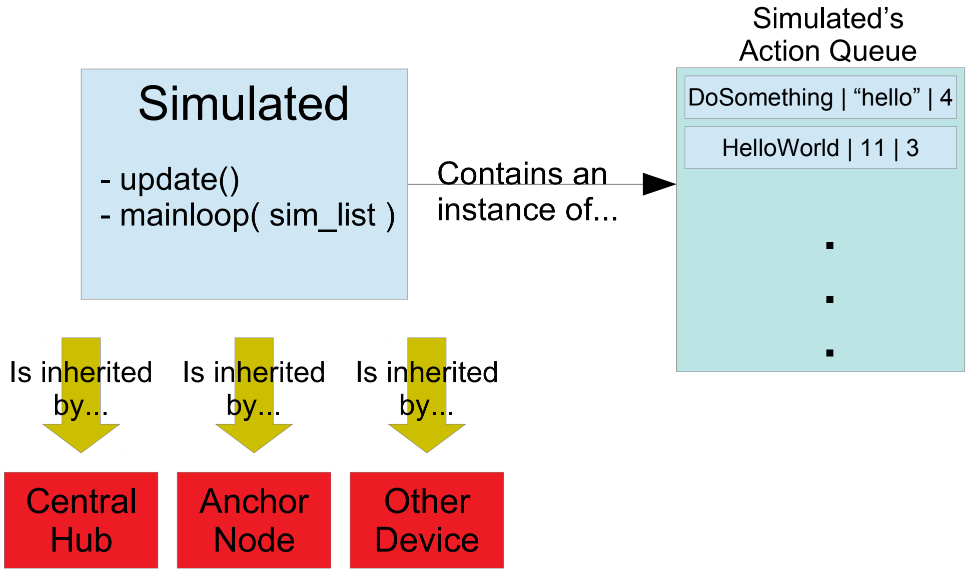
DoSomething | “hello” | 4

HelloWorld | 11 | 3

...

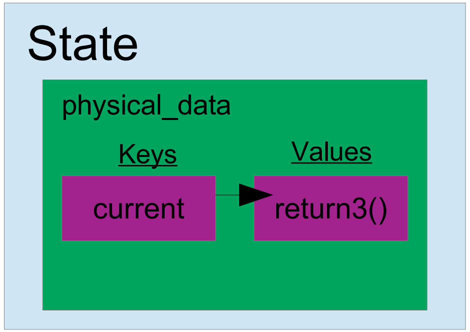
Simulated Class

- Simulated is simply an abstract class that generalizes the notions of a device, central hub, anchor node, etc.
- Simulated contains an action queue that keeps track of what is going to be happening with the object (ex. 3 more timesteps until the anchor node sends a packet to the central hub).
- Simulated contains two essential functions:
  - update() :: call the decrement function
  - mainloop(sim_list) :: a function intended to be called by the simulation environment during every timestep; sim_list is a list of Simulated Types (ex. a list of all the anchor nodes would be passed to the central hub's Simulated object); the goal is for mainloop() to call a C library and for the same C library's code to be used on the actual device



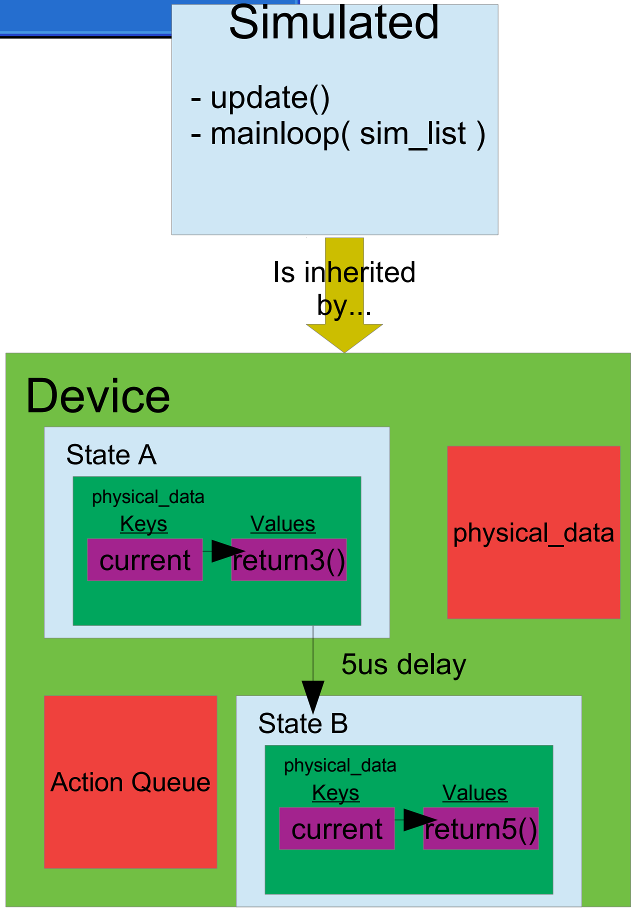
State Class

- Let’s take a brief detour to the state class.
- The state class is a single node in a finite state machine (ex. Deep sleep mode for the DW1000).
- Each state object has a physical_data dictionary associated with it.
  - Keys :: name of a particular value (ex. how much current is consumed in deep sleep mode)
  - Values :: functions that take one argument, the physical_data dictionary; the return value of these functions is the value of the corresponding parameter specified in Keys (ex. physical_data[“current”]() returns 3 for 3 mA)



Device Class Structure

- The device class contains different functions, which are explained in the spec on Github. This class inherits from the Simulated abstract class and acquires an action queue as a result.
- The general idea is the “Device” is an FSM that describes the hardware, the DW1000 in our current case.
- Each node in the FSM is a State object and each transition has a delay associated with it (ex. 10us to transition from sleep to deep sleep mode).
- Whenever a state transition occurs, we push a call to the setState function to the Device’s action queue. The simulation environment will call the device’s update() function every timestep to decrement the counter. This is how the transition delay is simulated.
- The device class also contains its own physical_data dictionary. For information on the device’s battery life, for instance, one looks at the Device’s physical_data dictionary. To know the current consumption in the current state, we would check the device’s current state’s physical_data dictionary.



Simulation Environment

- The simulation environment instantiates the different device objects.
- It accesses physical_data dictionaries depending on whether it wants the device’s battery life or the state’s current consumption.
- Each timestep, the simulation environment can do this with each device and determine the device’s lifetime.
- The simulation environment also instantiates central hub and anchor objects, which are simply different classes that inherit the Simulated abstract class (or the device class that is used by all the other nodes in the network).
- The simulation environment iterates through each device, central hub, and anchor and calls their mainloop function each timestep. This is how the simulation progresses through time.

Architecture Specification

- The architecture specification keeps track of different parameters of the current simulation environment.
- Specifically, the architecture spec tells the simulation environment what the types of the central hub, device, and anchor nodes are. For instance, the architecture spec is responsible for telling the simulation environment that the device should be an instance of the Device class we’ve written.
- It also determines the distance between each anchor node and other parameters.

Remaining Questions

- What is the mechanism by which the simulation environment keeps track of the timestep?
- How does the user configure the simulation settings (ex. simulation time length and number of nodes)?
- How does the simulation environment keep track of the position of each node?
- How does the simulation environment calculate the error in position determined by the central hub?