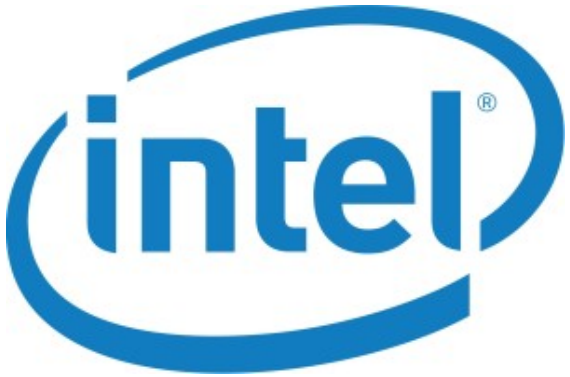
2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Multithreading a UVM Testbench for Faster Simulation

Benjamin Applequist
Vijayakrishnan Rousseau
Andrew Tan
Jayasrinivas Sesham

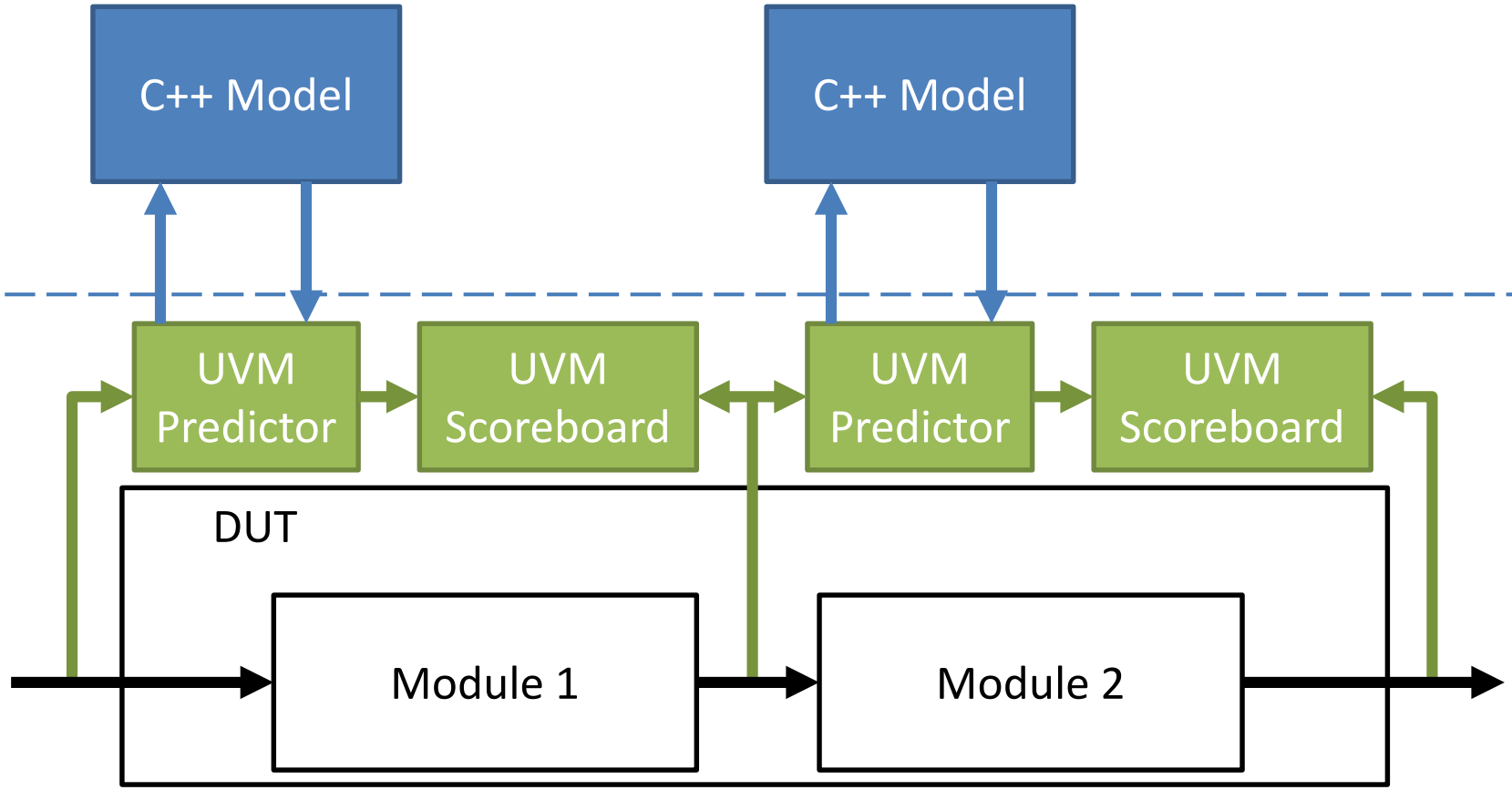


accellera SYSTEMS INITIATIVE

UVM Testbench with C++ Predictors

C++

SystemVerilog



Basic DPI Predictor

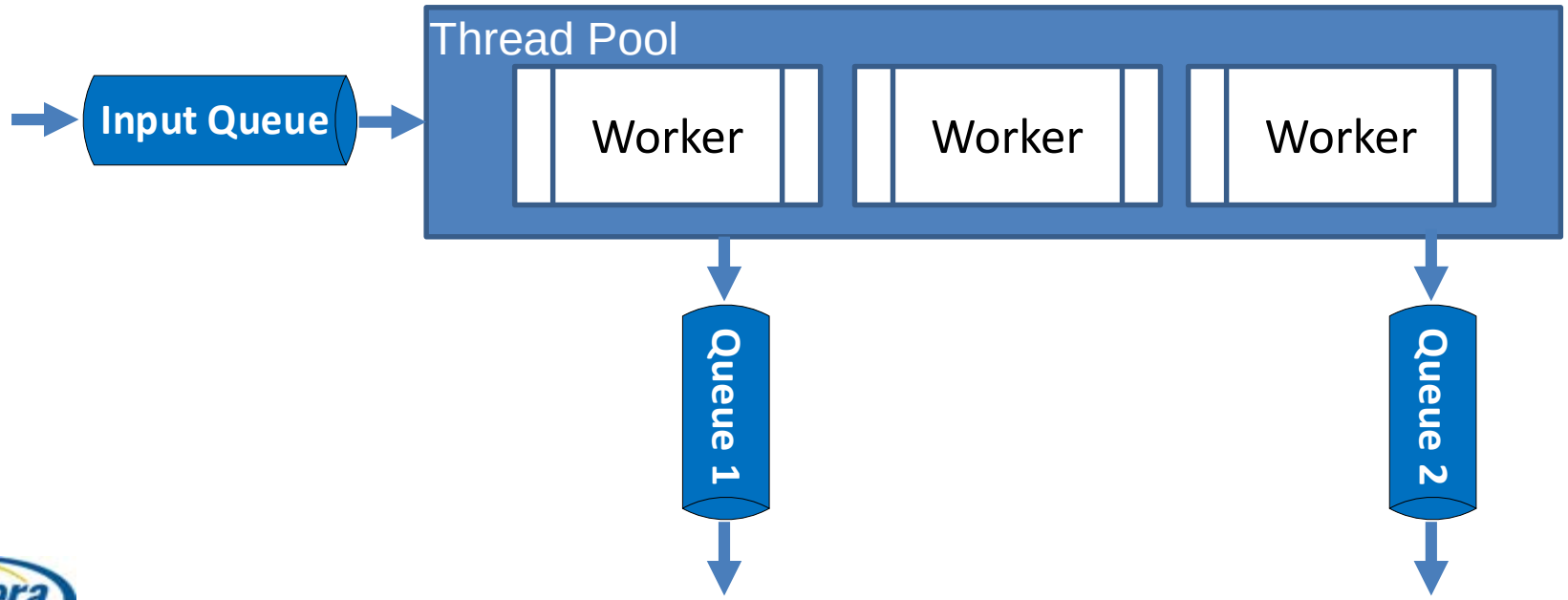
```
//Function to perform predictor work
int calc_result(int num) {
    …
}
```

```
import "DPI-C" function int
calc_result(input int count);

class my_predictor
  …
  function void write(my_item item);
    my_item out_item;
    $cast(out_item, item.clone());
    out_item.data = calc_result(item.data);
    predicted_ap.write(out_item);
  endfunction
```

UVM Testbench with Asynchronous Predictors

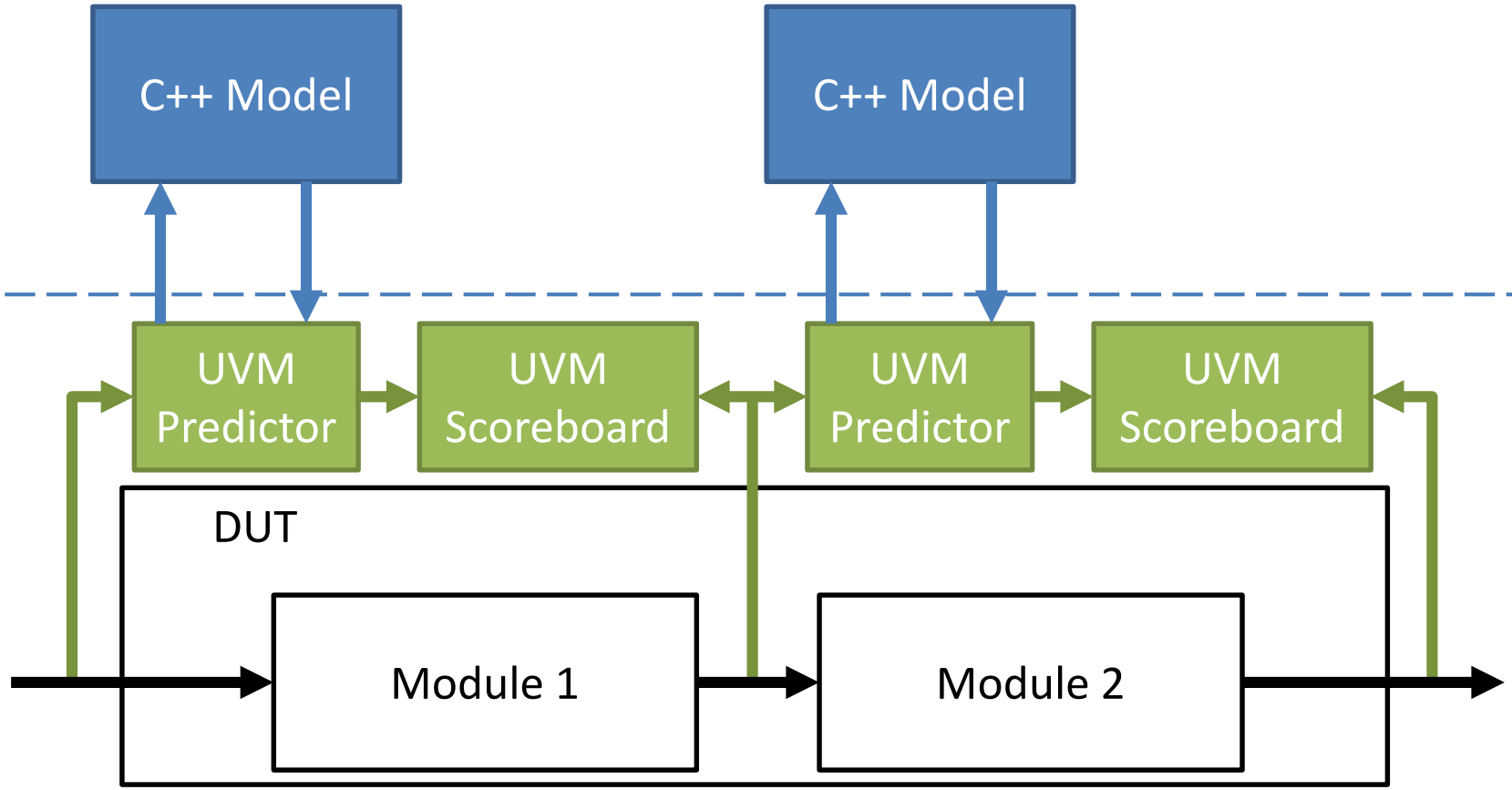
- Too much C++ code can cause simulation slowdowns
- C++ tasks can be multithreaded using a thread pool



UVM Testbench with Async Predictors

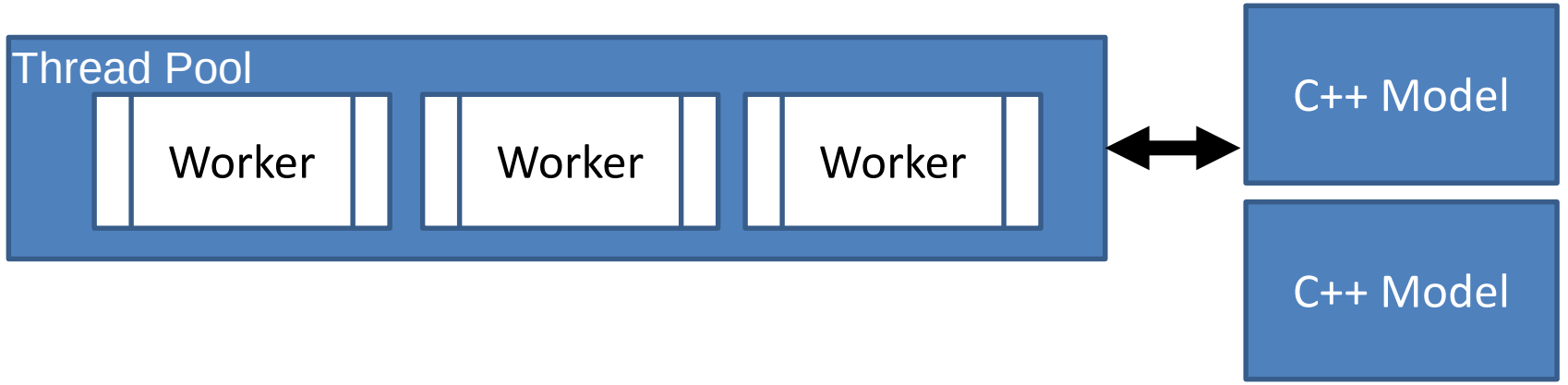
C++

SystemVerilog

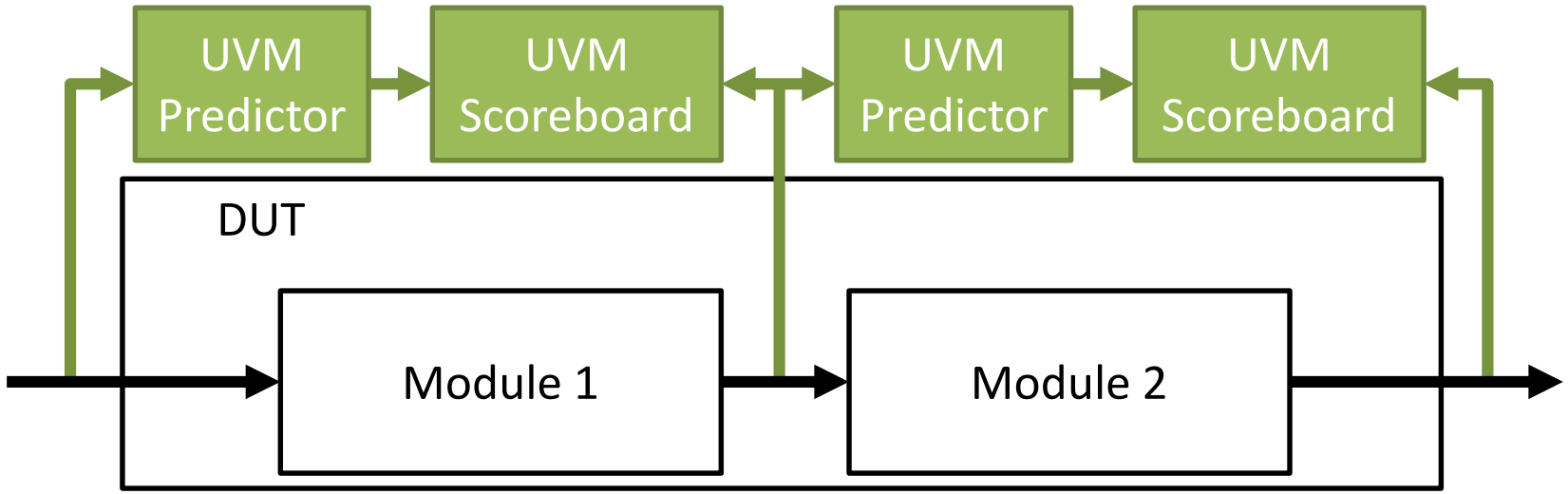


Add a Thread Pool

C++

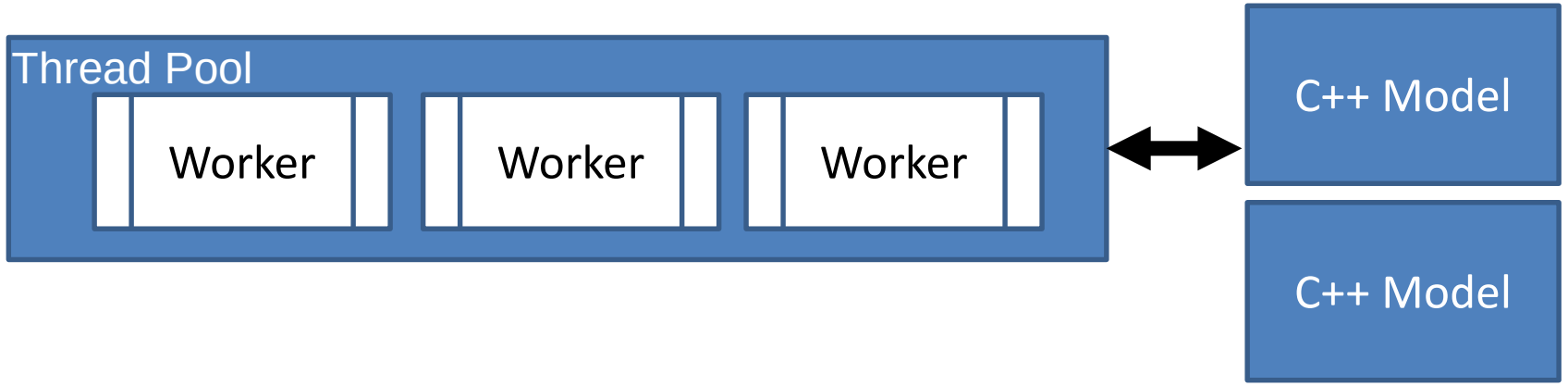


SystemVerilog

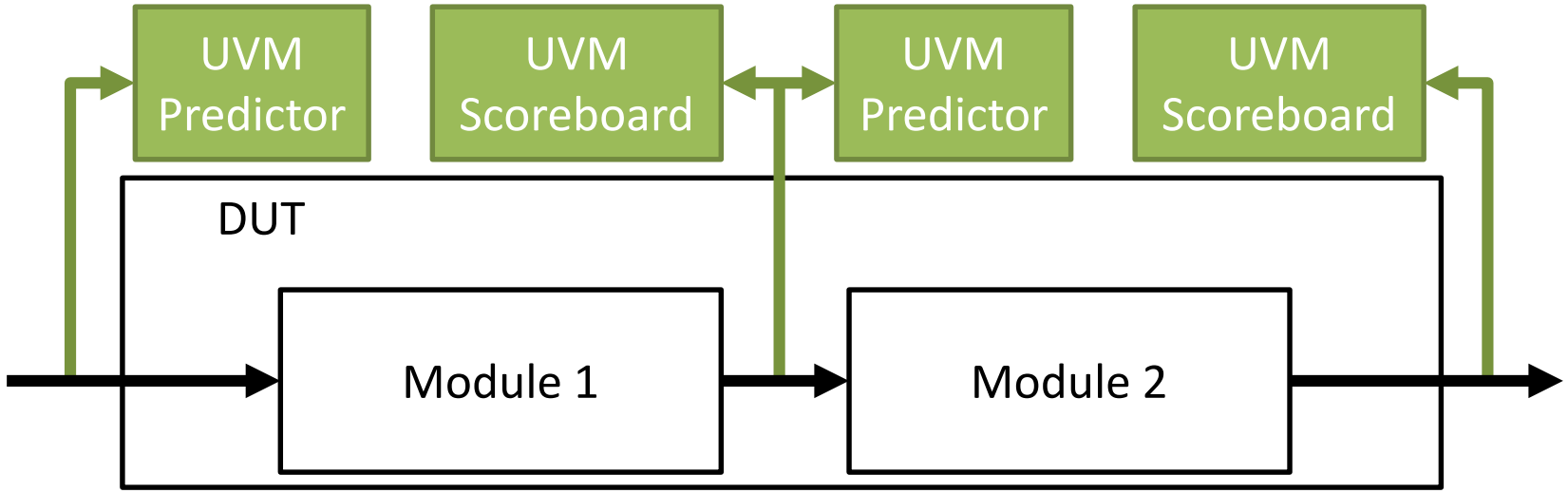


Disconnect Predictor from Scoreboard

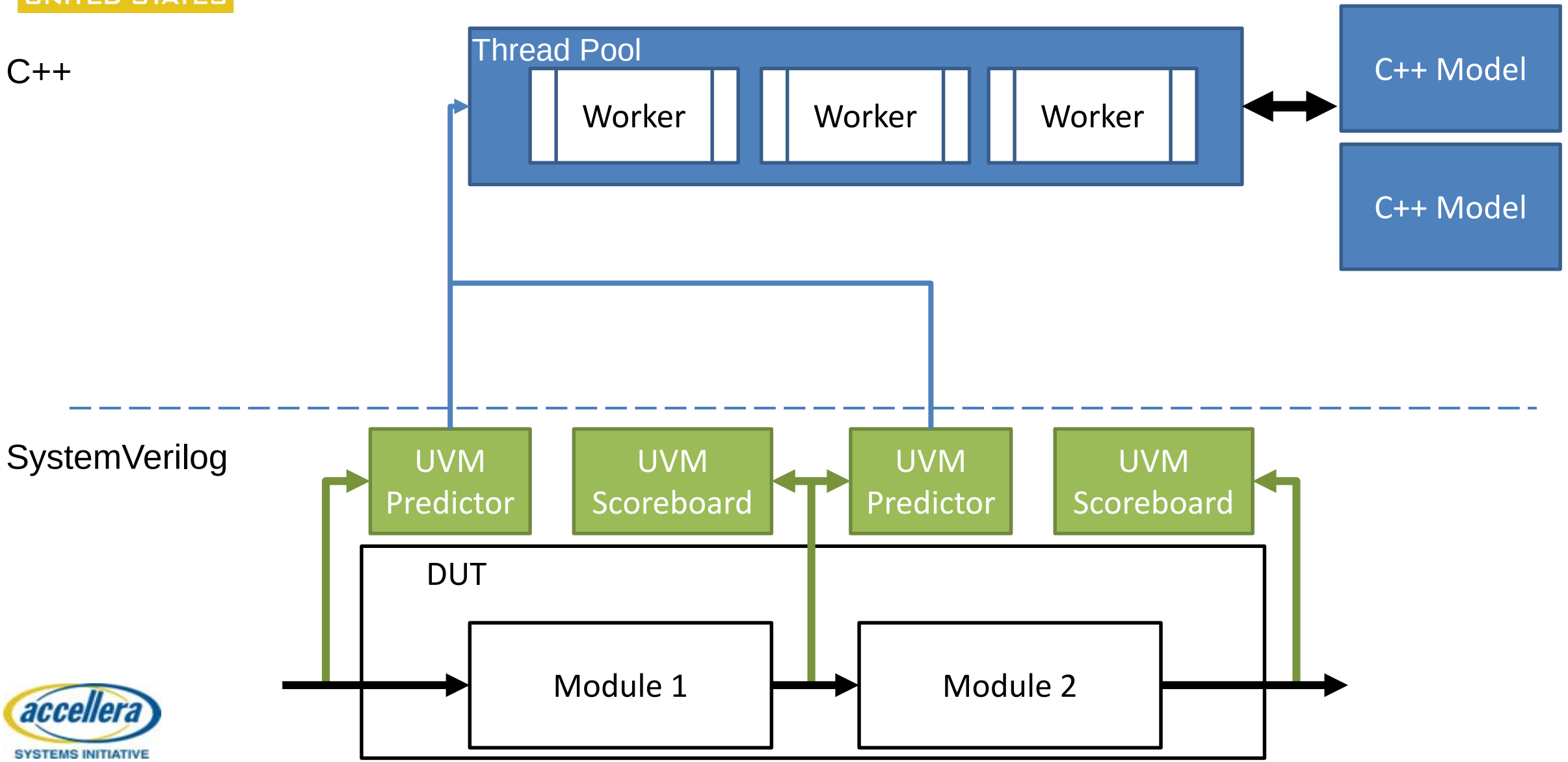
C++



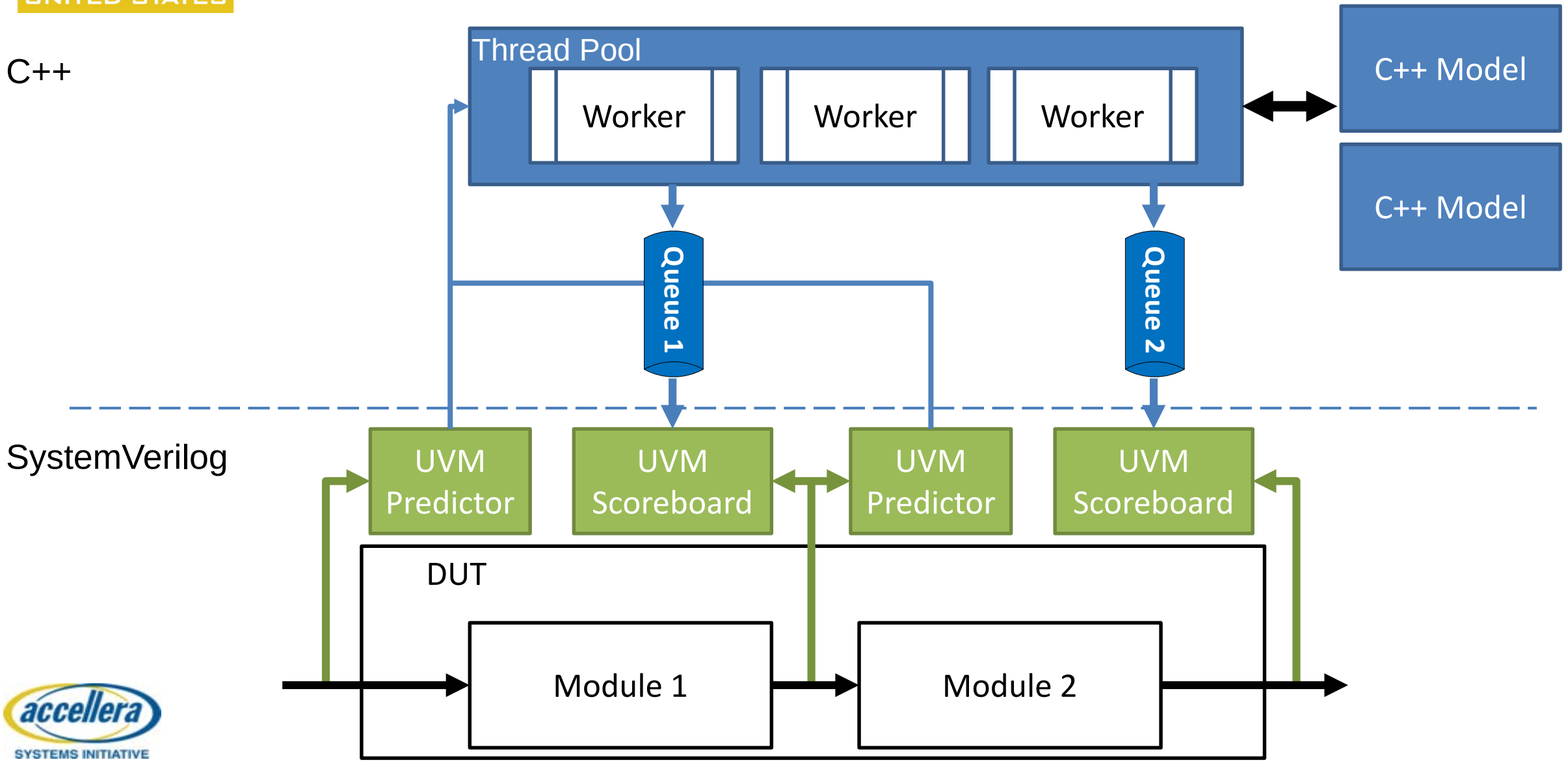
SystemVerilog



Send Predictor Data to Thread Pool



Connect Scoreboards to Output Queues



Thread Pool Initialization

```
class thread_pool {
public:
  static thread_pool& get_instance(){
    static thread_pool inst;
    return inst;
  }
private:
  thread_pool() = default;
}
```

```
thread_pool& tp = thread_pool::get_instance();
```

Thread Pool Work Submission

```
//Store futures
std::queue<std::future<int>> futures;

//Function to do actual predictor work
int calc_result(int num) {
    …
}

//Predictor function
extern "C" void predict_call(const int num) {
  thread_pool& tp =
    thread_pool::get_instance();
  futures.emplace(
    tp.add_job(calc_result, num));
}
```

Thread Pool Work Retrieval

```
//Scoreboard function
extern "C" int sb_call() {
  int num = futures.front().get();
  futures.pop();
  return num;
}
```

Enhanced Predictor

```
//DPI call to submit work
import "DPI-C" function void
   predict_call(input int num);

class my_predictor extends
  uvm_subscriber #(my_item);

  …

  function void write(my_item item);
    predict_call(item.data);
  endfunction
endclass : my_predictor
```

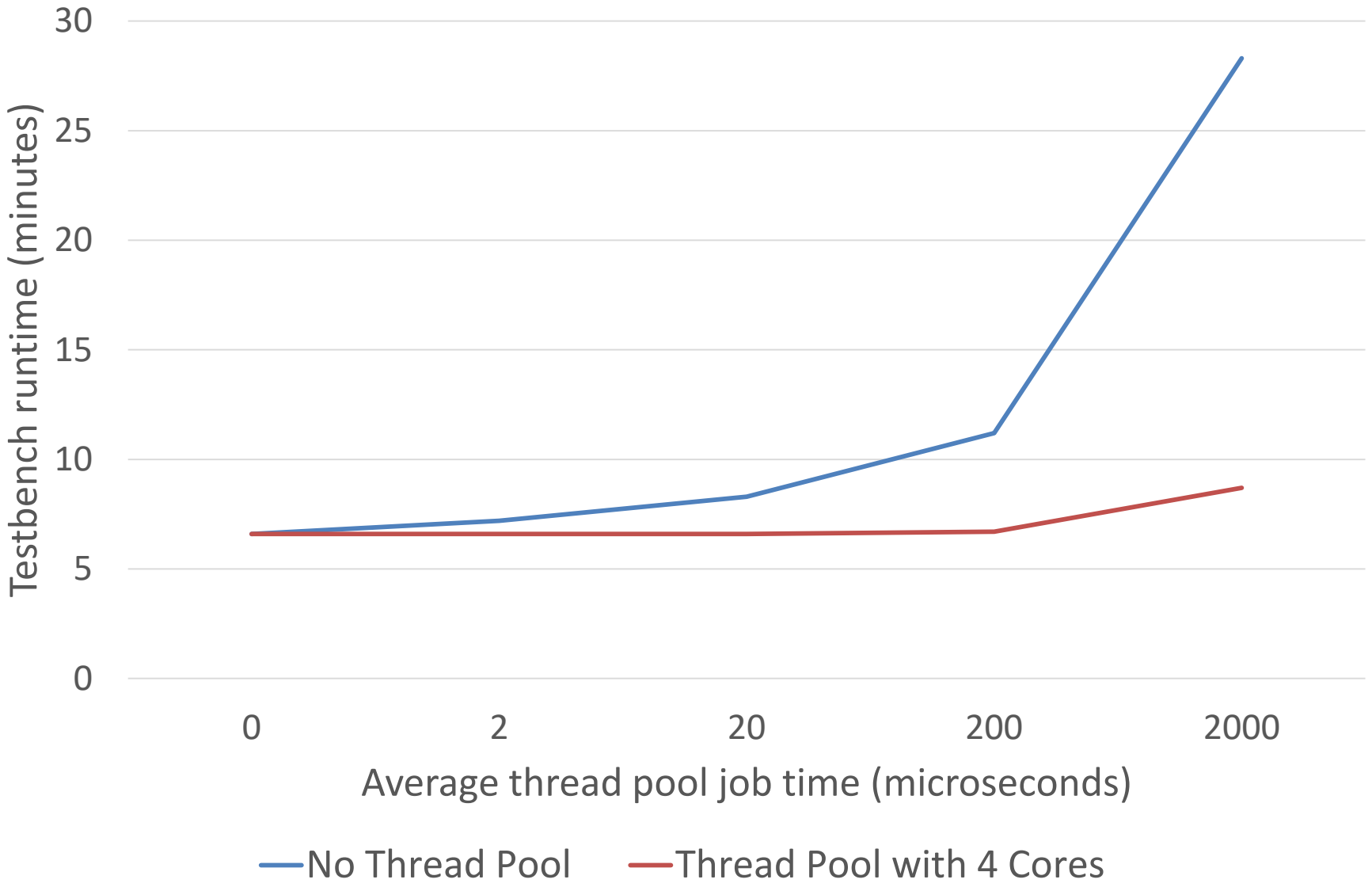
Enhanced Scoreboard

```
//DPI call to retrieve value
import "DPI-C" function int sb_call();

class my_scoreboard #(type T = my_item)
  extends uvm_scoreboard;

  virtual function void write_rcv(T txn);
    int rx_data;
    int pred = scoreboard_call();
    rx_data = txn.data;
    if (rx_data != pred) begin
      `uvm_error("Scoreboard", "Failure")
    end
  endfunction
endclass : my_scoreboard
```

Results


30
25
20
15
10
5
0
Testbench runtime (minutes)
0
2
20
200
2000
Average thread pool job time (microseconds)
No Thread Pool
Thread Pool with 4 Cores

Profile Your Testbench!

Dealing with State

- Asynchronous behavior of a thread pool prevents calls with persistent state
- Single threads can be allocated instead of thread pools for calls with state
  - Allocate up to one thread per type of call
  - Offload file I/O