Initial Discussions on AGILE Runtime

February 7th, 2023

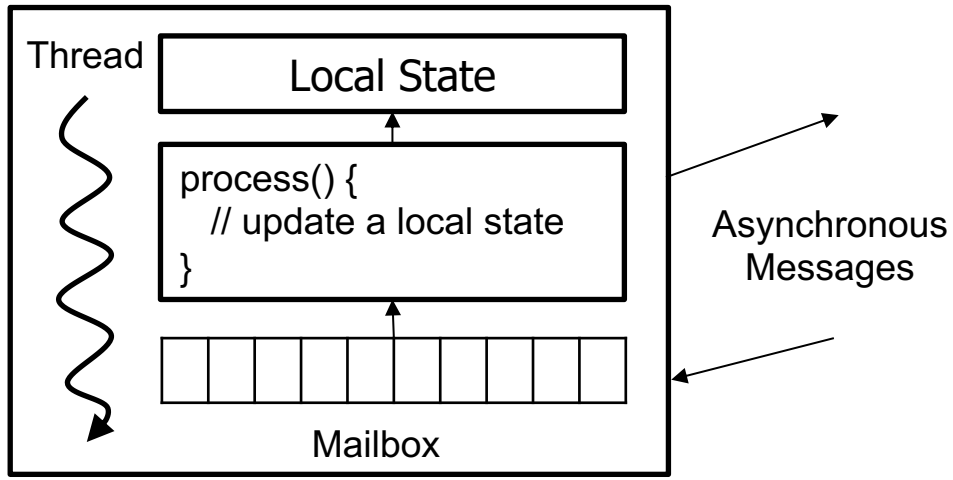
Akihiro Hayashi

Outline

- What is the Actor model?
- The current PGAS-Actor Runtime Implementation
- Potential Extensions to the current runtime for the FORZA HW

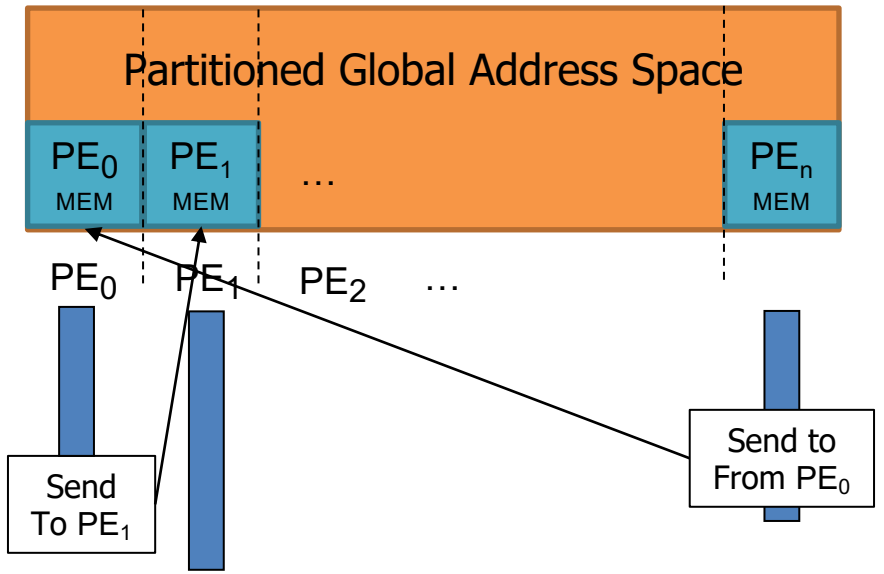
What is the Actor model?

- Actor
  - An immutable identity
  - Encapsulated mutable local state
  - Procedures to manipulate the local state
  - A logical thread of control
- Other important features
  - Process one message at a time
  - Asynchronous message passing
  - Non-deterministic ordering of messages
- Limitations
  - Data races possible when messages include shared objects
- Note: Selector is an Actor with multiple mailboxes



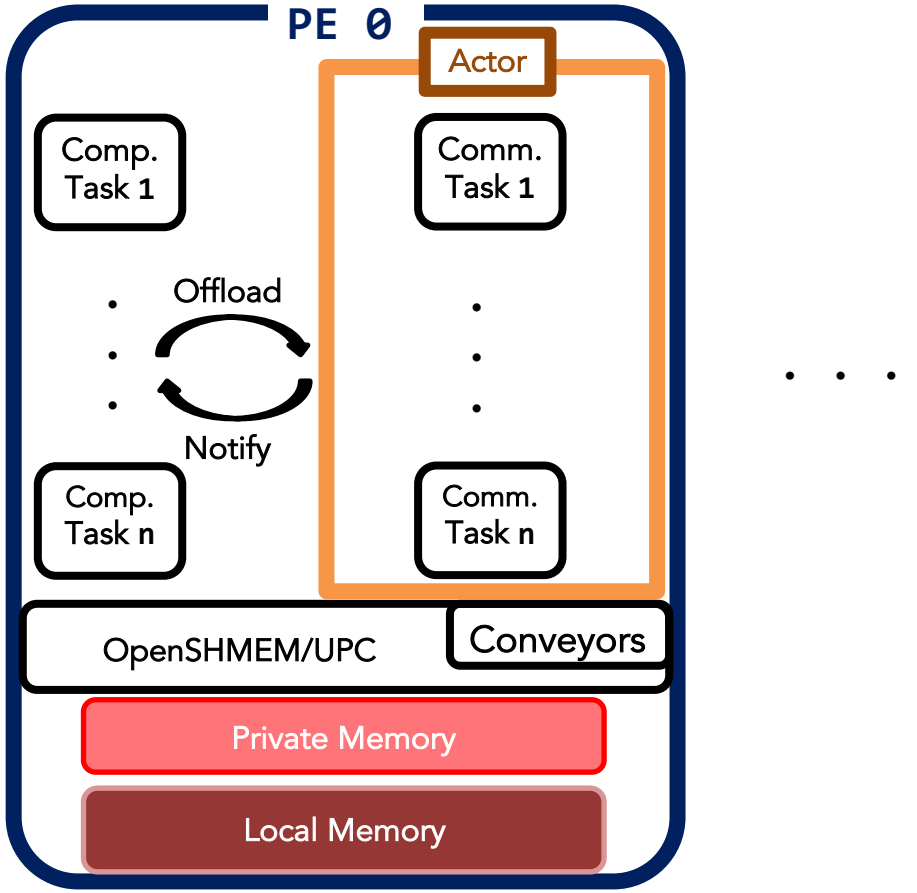
The PGAS Actor Model

- Each PE has a slice of a global address space
  - PE = a process/rank that owns an own address space
  - Communications between PEs require explicit communication API (i.e., send())



The anatomy of the current Actor runtime

- PGAS-based distributed actor/selector runtime system with a high-performance message aggregation library
- Dependencies
  - Actor/Selector + Tasking
    - Habanero-C Library (HClib)
      - https://github.com/srirajpaul/hclib/tree/bale3_actor/modules/bale_actor
  - Communication
    - Conveyors library
      - https://github.com/jdevinney/bale
    - OpenSHMEM/UPC
      - http://www.openshmem.org/site/
      - https://upc.lbl.gov/

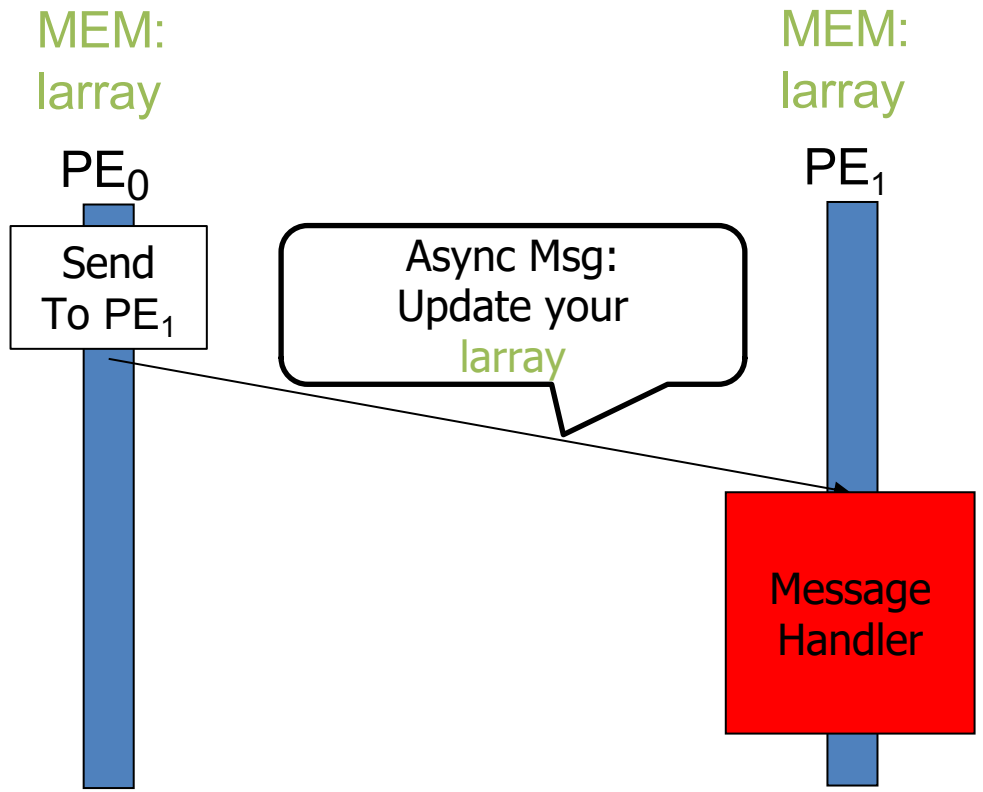


FORZA

The Histogram (PUT) pattern

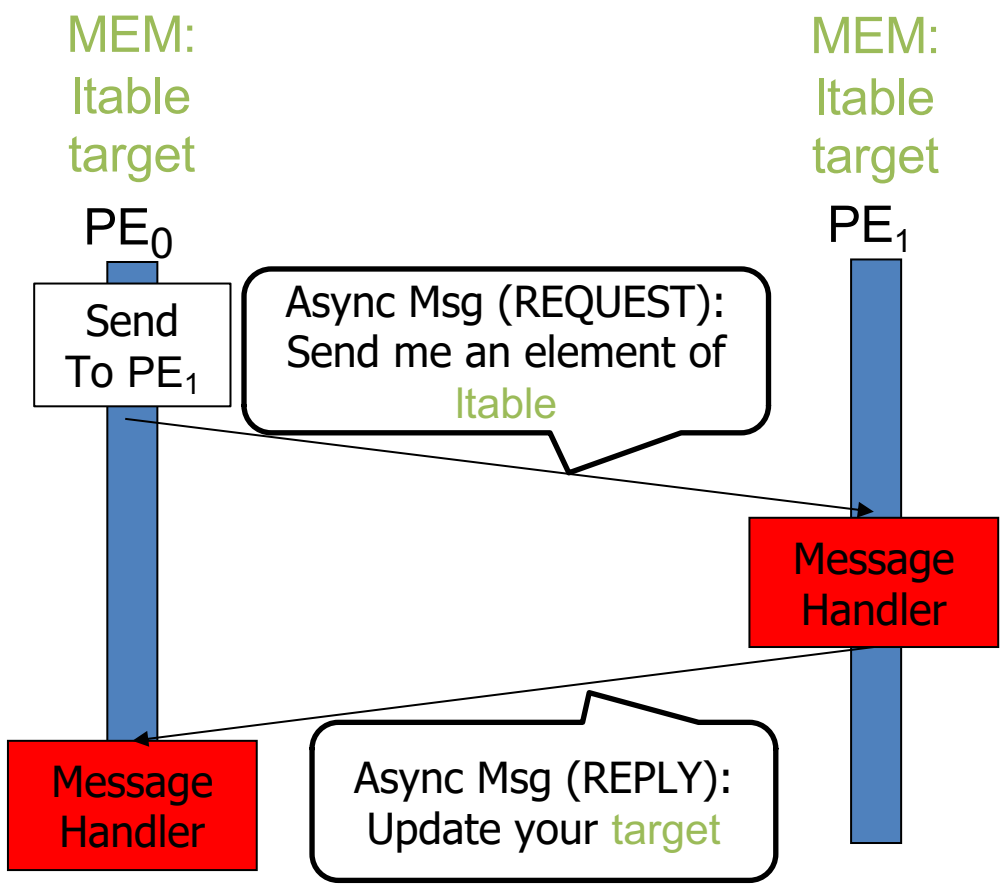
```
1  // Main Program (SPMD)
2  MyActor* actor_ptr = new MyActor(larray);
3  hclib::finish([=]() {
4    actor_ptr->start();
5    for (int i = 0; i < N; i++) {
6      int pe = …;
7      // non-blocking SEND
8     actor_ptr->send(i, pe);
9    }
10   actor_ptr->done(0);
11 });
```

```
1  // Actor Class
2  class MyActor: public hclib::Selector<1, int> {
3      int *larray;
4      // Message Handler
5      void process(int idx, int sender_rank) {
6          larray[idx] += 1;
7      }
8  public:
9      MyActor(int *larray) : larray(larray) { … } };
```



The Index-Gather (GET) pattern

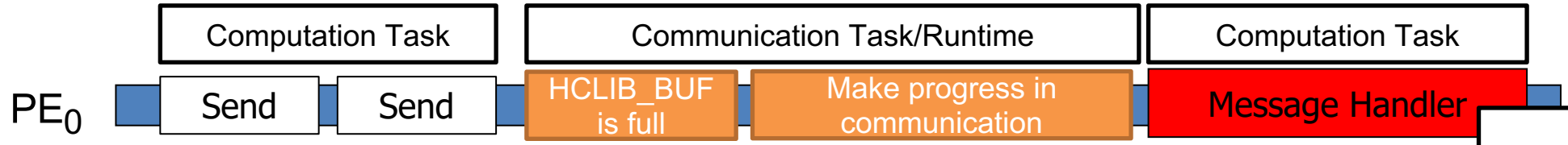
- This pattern requires REQUEST-REPLY sequence



How asynchronous communication works

- Each PE is **single-threaded** and performs different tasks in an interleaved fashion
- Communication flow: user program -> HCLIB_BUF -> Conveyor's buffer -> network

```
// User program
for (i = 0; i < n; i++) { s->send(…); // non-blocking }
```




```
1  // Selector Runtime
2  void send(…) {
3    while (HCLIB_BUF_is_full) {
4      // context-switch
5      hclib::yield();
6    }
7    // push an item
8    // into HCLIB_BUF
9  }
```

```
1  // Selector Runtime
2  void worker_loop(…) {
3    …
4    while(convey_advance()) {
5      // outgoing messages
6      foreach (HCLIB_BUF) convey_push();
7      clear(HCLIB_BUF);
8      // incoming messages
9      while (convey_pull()) { MESSAGE_HANDLER(); }
10     hclib::yield();
11   }
12 }
```

Make progress in communication (flush Conveyor's outgoing buffer and fill incoming buffer)

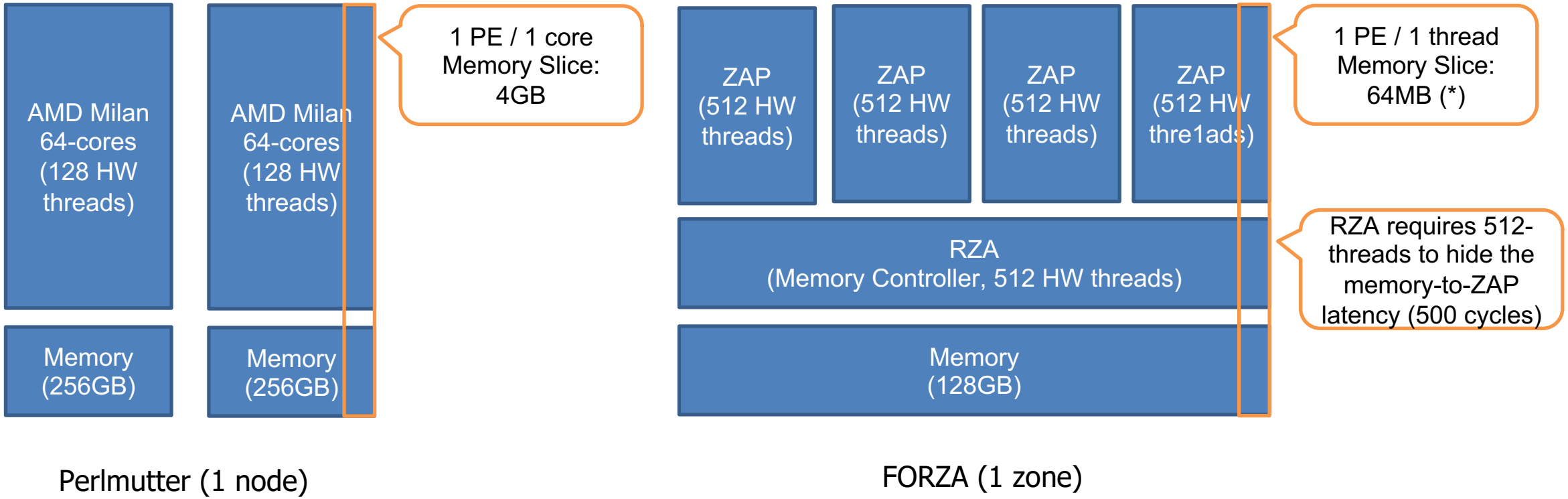
Push HCLIB_BUF's items into Conveyor's outgoing buffer

Pull items from Conveyor's incoming buffer and call the handler

FORZA

Actors on the FORZA HW?

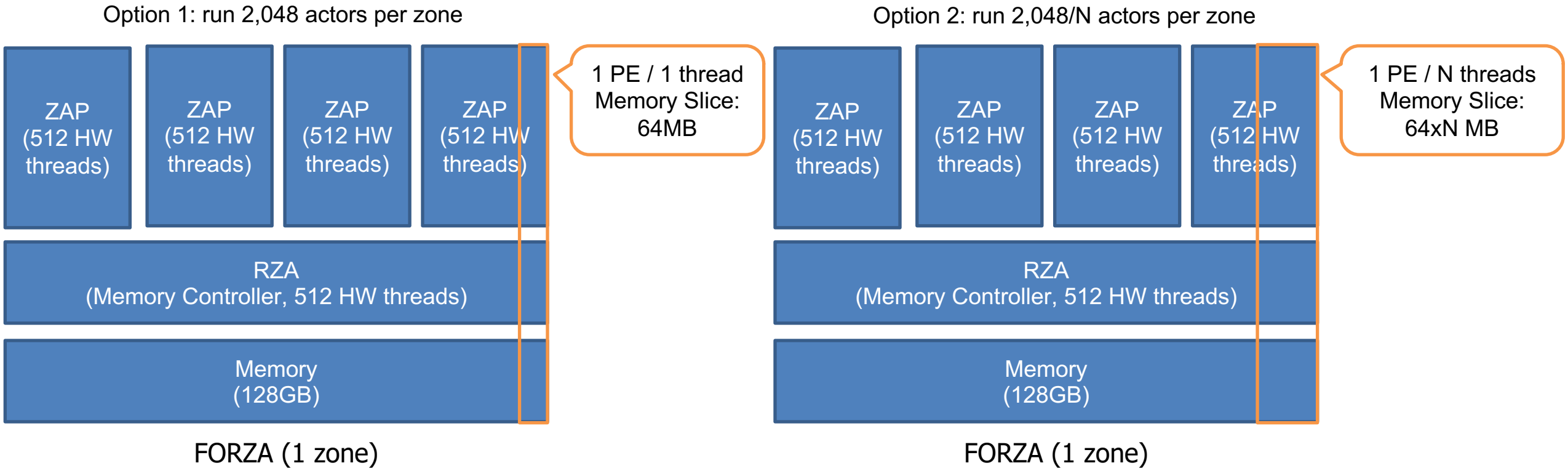
- Pros: Can keep the programming model and runtime implementation as-is
- Cons: Each Actor may only have a 64MB slice of memory (*)
  - The current runtime implementation consumes more memory



(*) Assuming 128GB memory / zone

Actors on the FORZA HW? (Cont'd)

- The goal: come up with a design and implementation of the Actor/Selector that fully utilizes the FORZA HW



Actors on the FORZA HW? (Cont'd)

- Option 1: run 2,048 actors per zone
  - Pros: Can keep the current SW implementation as is
  - Cons: 64MB per actor would be too small to even run the runtime
- Option 2: run 2,048/N actors per zone (N = # of threads / actor)
  - Pros: Each actor can have more memory
  - Cons:
    - Needs to increase parallelism within an actor to saturate ZAP and RZA
    - That is, need to modify/relax the current PGAS Actor runtime / the original Actor model
      - Introducing multi-threading in the process method
      - Parallel processing of multiple messages
        - Violates the one message at a time in the original actor model

More parallelism in a single actor

- Enhancement in the user-facing API
  - Requires the user to specify the parallelism in the process method [1]
    - At least, Triangle counting has enough parallelism inside the process method
    - Need to look into other ISBs and WFs
  - Can we break the one message at a time rule to exploit further parallelism?
- Enhancement in the runtime
  - Needs a light-weight concurrent queue implementation (e.g., ticket-lock [2]) because the conveyor API is not thread safe

```
1 void process(int idx, int sender_rank) {
2    finish {
3       forasync {}
4       async {}
5    }
6    // wait until the completion of
7    // asynchronous tasks to keep
8    // the one message at a time rule
9 }
```

```
1 void process(int idx[N], int sender_rank) {
2    finish {
3      forasync (int i = 0; i < N; i++) {}
4    }
5    // process N messages at a time
6 }
```

[1] Shams M. Imam and Vivek Sarkar. 2012. Integrating task parallelism with actors. (OOPSLA '12)
[2] Jun Shirako, Nick Vrvilo, Eric G. Mercer, Vivek Sarkar. Design, Verification and Applications of a New Read-Write Lock Algorithm. (SPAA'12)

Other important things

- How we provide user-facing API for each type of message?
  - Remote Atomics
  - Migrating threads
  - Active Message
  - Clarification: The current PGAS-Actor model can express all the types with the send API. In FORZA, should we provide a dedicated API routine for each type?