# Discussion Questions for BEAM Message Passing

N. Ramsey and M. Huang, for COMP 250RTS

November 15, 2017

## Recommended reading

All in the BEAM book:

- "Processes" chapter. Focus on the later sections, beginning with "Processes are just memory." Skip these sections:
  - Process listing, programmatic programming, poking processes from the shell
  - HiPE built-in functions (BIFs)
  - Inspecting message handling
  - The process dictionary
  - The "Dig in" section
- In the memory-subsystem chapter, the section on "Process Memory" may be helpful, especially the subsections on "Term sharing" and "Message passing." The subsection on the garbage collector contains too much detail to be useful. The rest of the chapter is not worth reading.

Not recommended: there is a section on message passing in the "Modules" chapter, but it is primarily about how to use the bytecode instructions to encode various forms of receive. Too much detail for us.

### My notes on message passing

When a message won't fit in a machine word, it must be initialized by the sender but collected by the receiver. In BEAM, the message is copied into memory in one of two places:

- If the receiver is suspended, the sender will grab a lock and copy the message directly onto the receiver's heap.
- If the receiver is not suspended, or if another process holds the lock and is concurrently writing into the receiver's heap, the sender will allocate a new *heap fragment* and will copy the message there.

There are two *message queues*: internal and external.

It is not obvious to me whether the "lock free" message passing is truly lock-free or whether there is a short instruction sequence (insert a pointer into a queue) that has to be protected by a lock.

When objects survive a minor collection, they are moved onto an *old heap*.

## Questions I: What's happening?

Most of the questions below are answered in the BEAM book. But not all. Sometimes you will have to imagine what you think the answer has to be (or could be).

(1) When process *sends* a message,

  (a) What memory is allocated, where, and how?
  (b) What data is copied where?
  (c) What locks might be acquired, and what work is done while holding each lock?

(2) When a process evaluates `receive`,

  (a) What memory is allocated, where, and how?
  (b) What data is copied where?
  (c) What memory, if any, is freed?
  (d) What locks might be acquired, and what work is done while holding each lock?

(3) When message is delivered to the mailbox of a process that is blocked evaluating `receive`,

  (a) What memory is allocated, where, and how?
  (b) What data is copied where?
  (c) What memory, if any, is freed?
  (d) What locks might be acquired, and what work is done while holding each lock?

At different times, a single process can be both sender and receiver. And all processes run the same garbage-collection code. But it may be useful to think about garbage collection from two different perspectives:

(4) When the garbage collector runs on a *sender*,

  (a) What memory is allocated, where, and how?
  (b) What data is copied where?
  (c) What memory, if any, is freed?
  (d) What locks might be acquired, and what work is done while holding each lock?

(5) When the garbage collector runs on a *receiver*,

  (a) What memory is allocated, where, and how?
  (b) What data is copied where?
  (c) What memory, if any, is freed?
  (d) What locks might be acquired, and what work is done while holding each lock?

There are more questions on the next page.

# Questions II: Why?

## Heaps and heap fragments

(6) Why not always copy the message into a heap fragment?

(7) Why not always wait to acquire the lock and copy the message directly into the receiver's heap?

## Inboxes

(8) A process's mailbox appears to be represented using two separate queues (or "inboxes").

   (a) Are the two inboxes protected by the same lock? If so, why have two?

   (b) Are the two inboxes protected by two different locks? If so, how can a process evaluating `receive` ever know that it is safe to block?

   (c) Why do you think there are two inboxes?

# Bonus questions: Overall design

Last time, I proposed these constraints:

- Each Erlang process should have not only its own stack but also its own heap and its own garbage collector.
- Processes should have the opportunity to run in parallel.
- Memory should be managed with an eye to performance—controlling both the amount of copying and the amount of contention for shared mutable resources.
- If a process needs to acquire a lock, the amount of work done while holding the lock should be bounded above by a small constant.

(9) How well does BEAM's design respect these constraints?

(10) Can you conjecture how the design might have been influenced by the constraints? Can you identify other constraints that might have influenced the design?