# KILLER features of the BEAM

And what makes the BEAM a unique and powerful tool that really stands out!

# Actor Model

Hundreds, Thousands, Millions of processes…

System limit can be between1,024-134,217,727

They share **NOTHING**

Communicate through **message passing**

# Demo 1

## A hypothetical server

# JUGGLER

## a simplified DEMO SERVER

https://github.com/iguberman/erljuggler_demo

```
accept_loop() ->
  receive                    %% to get a message in my mailbox
    Req -> dispatch(Req, self()) %% self() = my process id
  end,
  accept_loop().
```

```
accept(NumRequests)->
  [dispatch(Req, self()) || Req <- lists:seq(1, NumRequests)],
  collect_responses(0, 0, 0, 0, NumRequests).

dispatch(Req, AcceptorPid) ->
  spawn(
   ?MODULE,                    %% Module
   kick_off_request_handler    %% Function
   [Req, AcceptorPid]).        %% Arguments
```

# JUGGLER

## a simplified DEMO SERVER

https://github.com/iguberman/erljuggler_demo

```
%%% a kind of a supervisor
kick_off_request_handler(Req, AcceptorPid) ->
  RequestHandlerPid =
    spawn(?MODULE, handle_request, [Req, self()]),
  Start = os:system_time(millisecond),
  receive
    {RequestHandlerPid, Resp} ->
        End = os:system_time(millisecond),
        Duration = End - Start,
        io:format("...~p [~b]...", [Req, Duration]),
        AcceptorPid ! {RequestHandlerPid, Resp, Duration};
    Unexpected ->
         AcceptorPid ! {error, Unexpected}
end.
```

# JUGGLER

## a simplified DEMO SERVER

https://github.com/iguberman/erljuggler_demo

```
  handle_request(Req, ParentPid) when is_integer(Req) ->
    Resp = count_to_1000_and_do_other_stuff_too(Req, 0),
    HandlerPid = self(),
    ParentPid ! {HandlerPid, Resp}
  end.
```

```
count_to_1000_and_do_other_stuff_too(_Req, 1000) -> ok;
count_to_1000_and_do_other_stuff_too(Req, C) ->
  case (Req rem 2) of
    0 ->  binary:copy(<<Req/integer>>,300);
    1 ->  binary:copy(<<(Req + 1)/integer>>,200)
  end,
  count_to_1000_and_do_other_stuff_too(Req, C+1).
```

# Demo 2

server with a bug

# JUGGLER

## a simplified DEMO SERVER

https://github.com/iguberman/erljuggler_demo

```
handle_request(Req, ParentPid) when is_integer(Req) ->
 case Req rem 100 of
   0 ->
       io:format("~n**** ~p [INF]*****~n", [Req]),
       ParentPid ! dont_wait_for_me,
         handle_with_inf_loop_bug();
     _Other ->

        Resp = count_to_1000_and_do_other_stuff_too(Req, 0),
        HandlerPid = self(),
        ParentPid ! {HandlerPid, Resp}
    end.
  end.

handle_with_inf_loop_bug()->
    infinite_loop(0).

infinite_loop(C) ->
  _A = binary:copy(<<1>>,200),
  _B = math:sqrt(1235),
  infinite_loop(C+1).
```

# BEAM Scheduler

Cooperative?

**Cooperative at C level**

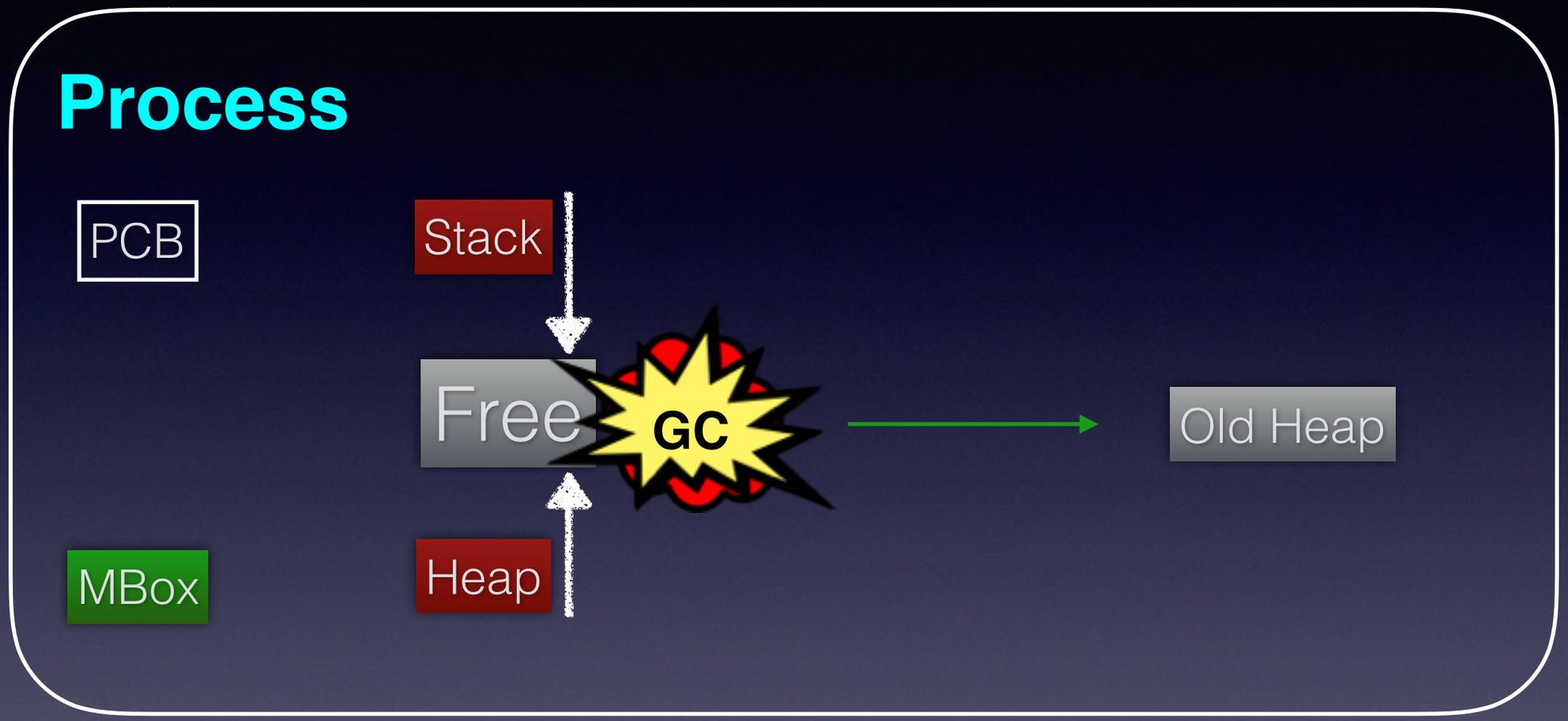
Preemptive?

**Preemptive at Erlang level**
**(by means of reduction counting)**

2000 reductions
Reduction ~= function call

**Word of caution: BIFs and NIFs**

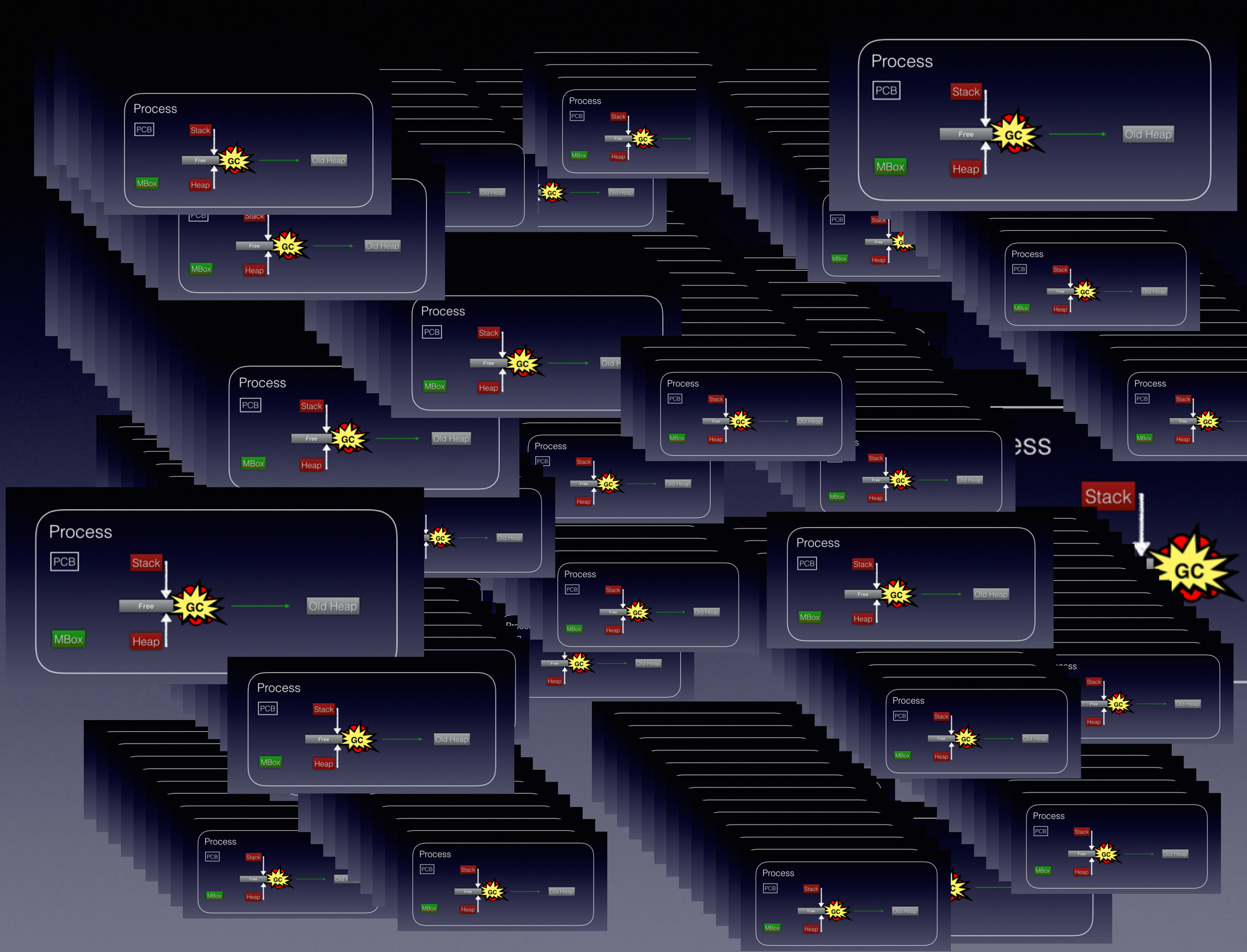
# BEAM Memory model




hipe_bifs:show_heap(Pid).

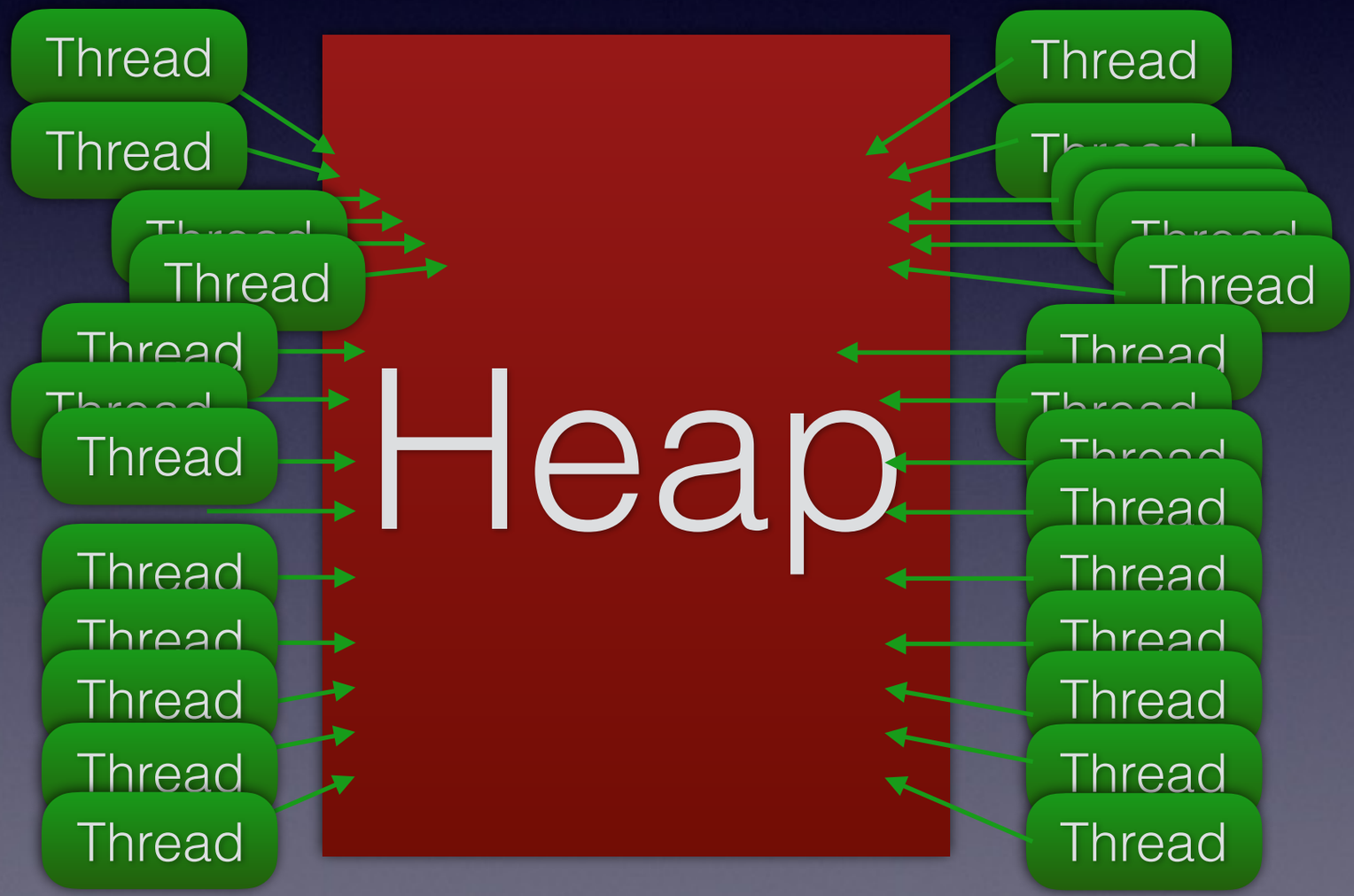
hipe_bifs:show_estack(Pid).

hipe_bifs:show_pcb(Pid). %% Look at **heap_sz** !


Process
PCB
Stack
Free
GC
Old Heap
MBox
Heap

# JVM

(simplified! SORRY! )




Stop the world.
Is that a thing in Erlang?

# DEMO 3

# The KILLER

# KILLER_JUGGLER

## a simplified DEMO SERVER

https://github.com/iguberman/erljuggler_demo

```
%%% a kind of a supervisor
kick_off_request_handler(Req, AcceptorPid) ->
  RequestHandlerPid =
    spawn(?MODULE, handle_request, [Req, self()]),
  Start = os:system_time(millisecond),
  receive
    {RequestHandlerPid, Resp} ->
        End = os:system_time(millisecond),
        Duration = End - Start,
        io:format("...~p [~b]...", [Req, Duration]),
        AcceptorPid ! {RequestHandlerPid, Resp, Duration};
    Other -> AcceptorPid ! {error, Other}
after 5000 ->
    exit(HandlerPid, timedout),
    AcceptorPid ! {HandlerPid, killed}
end.
```

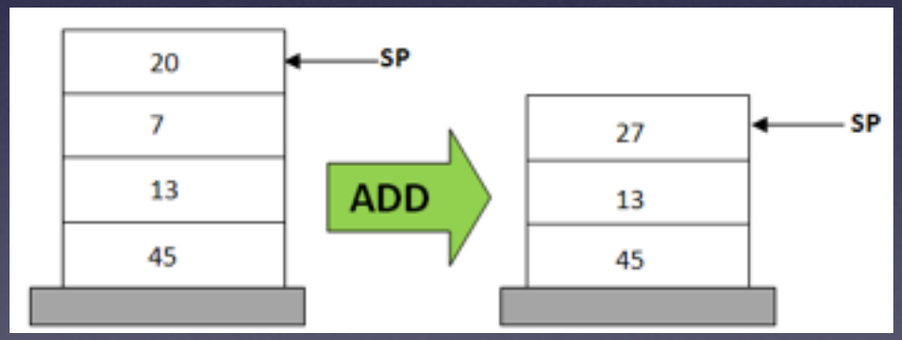
# STACK BASED

# REGISTER BASED

## More Instructions

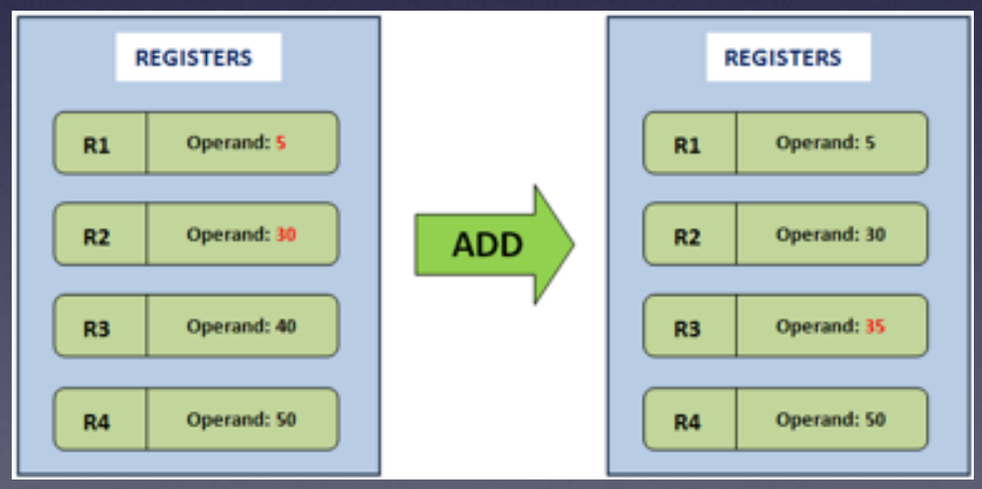
## Fewer instructions

## Instructions are simple




1.POP 20
2.POP 7
3.ADD 20, 7, result
4.PUSH result

## Instructions have more info




**1.** ADD R1, R2, R3 ;
# Add contents of R1 and R2, store result in R3

# Performance Survey on Stack-based and Register-based VirtualMachines

Ruijie Fang, Siqi Liu, 2016

**CONCEPTUM**
(stack-based like JVM)

**INERTIA**
(register-based like BEAM)

VM feature comparisons:

Scientific comparison of REGISTER (Erlang) vs. STACK (JVM) VM

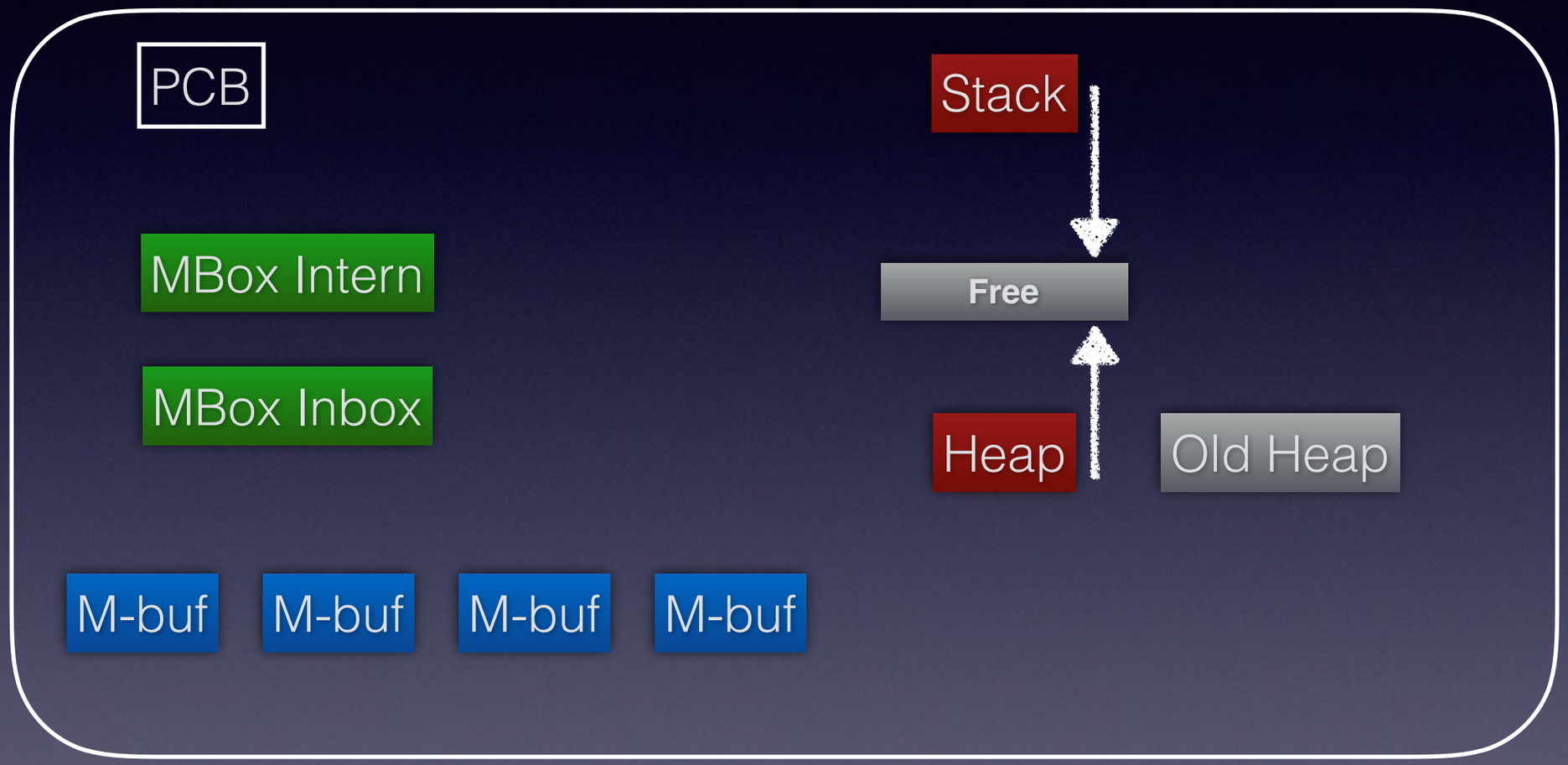
Inertia spends 66.42% less time in instruction dispatch than Conceptum, on average

However, Inertia is still slower
in the overall fetch time, spending 23.5% more time on average in fetching operands than Conceptum does

Based on our test results, stack-based virtual machines
typically perform better on benchmarks featuring a high amount of arithmetic operations.

In contrast to the stack-based virtual machine's
performance, the register-based virtual machine performed much better on recursions and memory operations.

# MBox


PCB
Stack
MBox Intern
Free
MBox Inbox
Heap
Old Heap
M-buf
M-buf
M-buf
M-buf

# DEMO 4

Bring it down by sending messages to the process affected by inf loop!

# TYPE SYSTEM

Strong typed.
So every type has a tag.

Dynamically typed.
Hot code loading anyone?

# TAGGING

In the memory representation of an Erlang term a few bits are reserved for a type tag.

**LEVEL 1 tags:**

00 Header (on heap) CP (on stack)
01 List (cons)
10 **Boxed** <- pointers to the heap
**11 Immediate** <- fit into one word on the stack

**LEVEL 2 tags (Immediate):**

00 **11** Pid
01 **11** Port
**10 11 Immediate 2**
11 **11** Small integer

**LEVEL 3 tags (Immediate 2):**

00 **10 11** Atom
01 **10 11** Catch
10 **10 11** [UNUSED]
11 **10 11** Nil <- for empty list []

**Q. How is cooperative scheduling implemented?**

**A.** If there are untagged  values —
no preempting

# REFERENCES


https://happi.github.io/theBeamBook

https://www.researchgate.net/publication/309631798_A_Performance_Survey_on_Stack-based_and_Register-based_Virtual_Machines (pdf available)

https://llvm.org/devmtg/2014-04/PDFs/Talks/drejhammar.pdf

https://markfaction.wordpress.com/2012/07/15/stack-based-vs-register-based-virtual-machine-architecture-and-the-dalvik-vm/