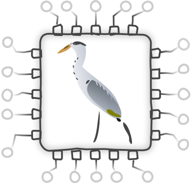
Cloaca

A Concurrent Hardware Garbage Collector for Non-strict Functional Languages

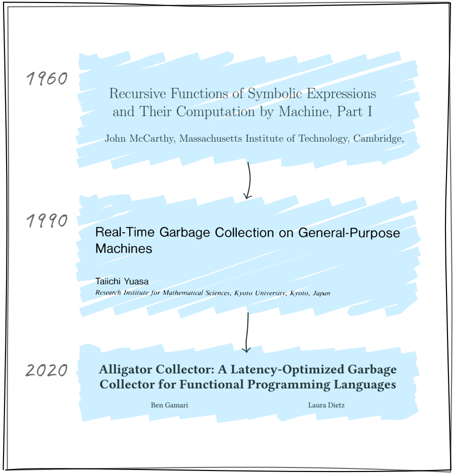
Craig Ramsay & Rob Stewart

September 2024

Heriot-Watt University



Software for Concurrent GC



1960

Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

John McCarthy, Massachusetts Institute of Technology, Cambridge,

1990

Real-Time Garbage Collection on General-Purpose Machines

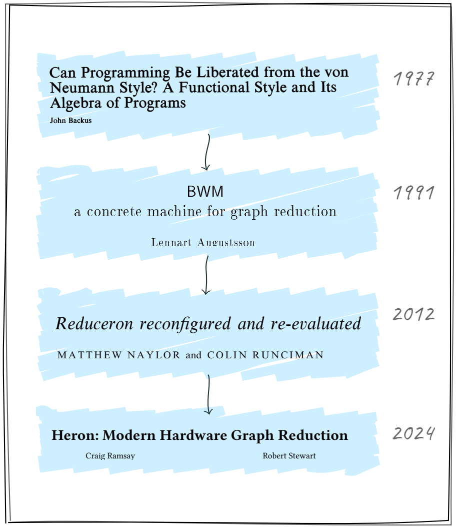
Taiichi Yuasa
Research Institute for Mathematical Sciences, Kyoto University, Kyoto, Japan

2020

Alligator Collector: A Latency-Optimized Garbage Collector for Functional Programming Languages

Ben Gamari Laura Dietz

Hardware for FP



Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus

1977

BWM
a concrete machine for graph reduction

Lennart Augustsson

1991

Reduceron reconfigured and re-evaluated

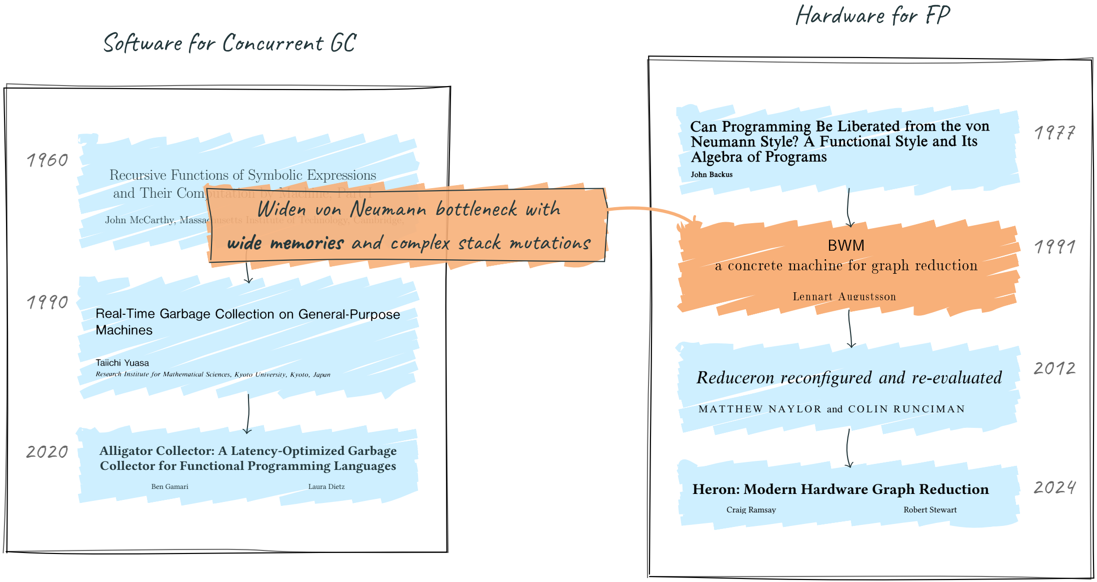
MATTHEW NAYLOR and COLIN RUNCIMAN

2012

Heron: Modern Hardware Graph Reduction

Craig Ramsay Robert Stewart

2024



Software for Concurrent GC

1960

Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

John McCarthy, Massachusetts Institute of Technology, Cambridge,

1990

Real-Time Garbage Collection on General-Purpose Machines

Taiichi Yuasa

Research Institute for Mathematical Sciences, Kyoto University, Kyoto, Japan

2020

Alligator Collector: A Latency-Optimized Garbage Collector for Functional Programming Languages

Ben Gamari — Laura Dietz

Widen von Neumann bottleneck with **wide memories** and complex stack mutations

Hardware for FP

1977

Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus

1991

BWM

a concrete machine for graph reduction

Lennart Augustsson

2012

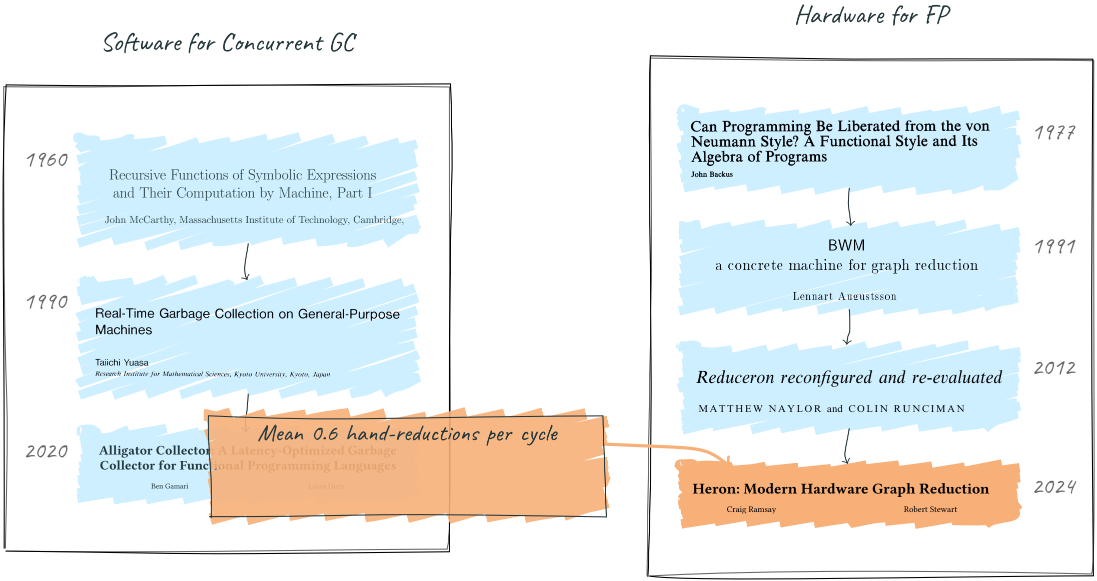
Reduceron reconfigured and re-evaluated

MATTHEW NAYLOR and COLIN RUNCIMAN

2024

Heron: Modern Hardware Graph Reduction

Craig Ramsay — Robert Stewart



Software for Concurrent GC

1960

Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

John McCarthy, Massachusetts Institute of Technology, Cambridge,

1990

Real-Time Garbage Collection on General-Purpose Machines

Taiichi Yuasa

Research Institute for Mathematical Sciences, Kyoto University, Kyoto, Japan

2020

Alligator Collector: A Latency-Optimized Garbage Collector for Functional Programming Languages

Ben Gamari

Laura Dietz

Mean 0.6 hand-reductions per cycle

Hardware for FP

1977

Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus

1991

BWM

a concrete machine for graph reduction

Lennart Augustsson

2012

Reduceron reconfigured and re-evaluated

MATTHEW NAYLOR and COLIN RUNCIMAN

2024

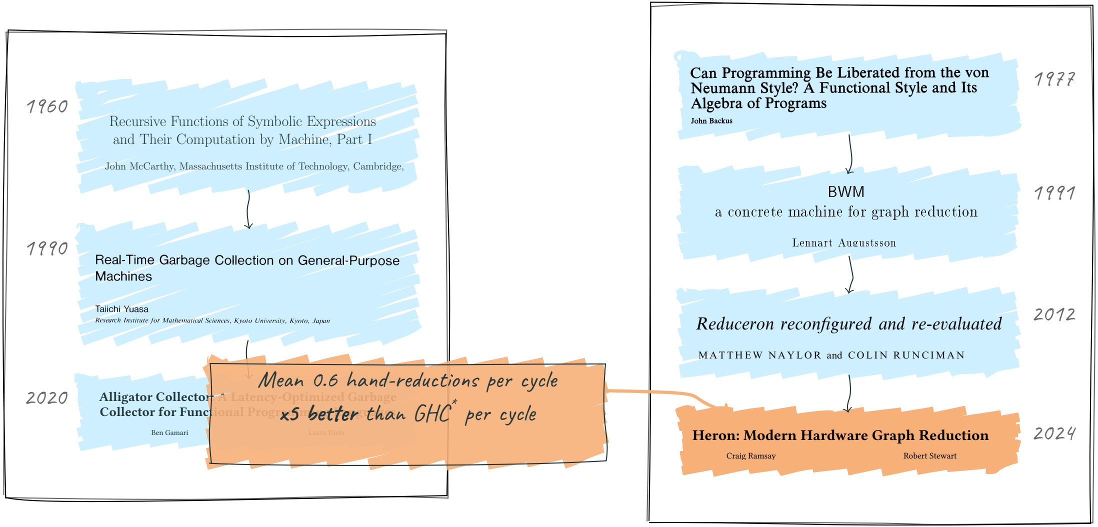
Heron: Modern Hardware Graph Reduction

Craig Ramsay

Robert Stewart

Software for Concurrent GC

Hardware for FP



1960

Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

John McCarthy, Massachusetts Institute of Technology, Cambridge,

1990

Real-Time Garbage Collection on General-Purpose Machines

Taiichi Yuasa
Research Institute for Mathematical Sciences, Kyoto University, Kyoto, Japan

2020

Alligator Collector: A Latency-Optimized Garbage Collector for Functional Programming Languages

Ben Gamari Laura Dietz

Mean 0.6 hand-reductions per cycle

x5 better than GHC* per cycle

1977

Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus

1991

BWM
a concrete machine for graph reduction

Lennart Augustsson

2012

Reduceron reconfigured and re-evaluated

MATTHEW NAYLOR and COLIN RUNCIMAN

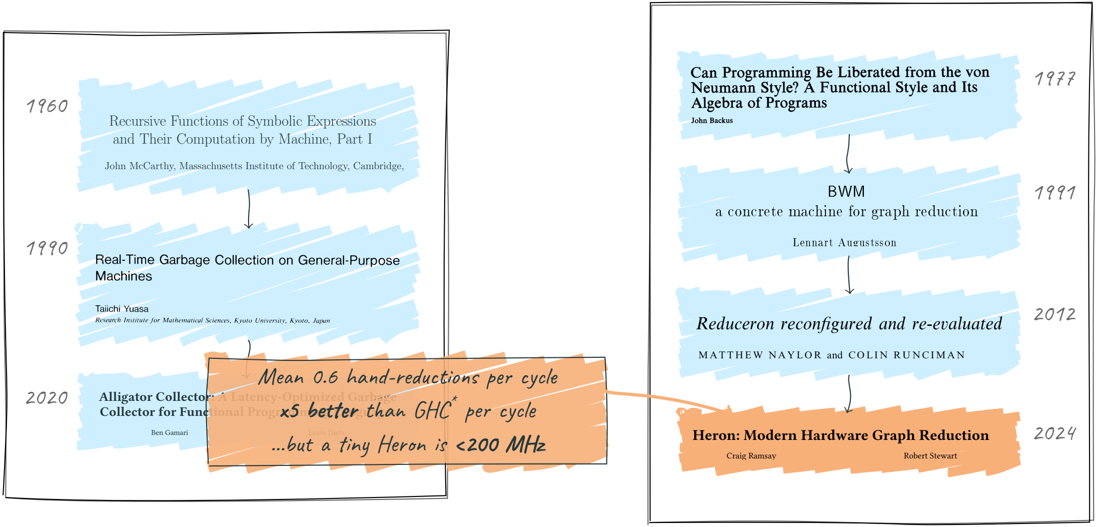
2024

Heron: Modern Hardware Graph Reduction

Craig Ramsay Robert Stewart

Software for Concurrent GC

Hardware for FP



1960

Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

John McCarthy, Massachusetts Institute of Technology, Cambridge,

1990

Real-Time Garbage Collection on General-Purpose Machines

Taiichi Yuasa

Research Institute for Mathematical Sciences, Kyoto University, Kyoto, Japan

2020

Alligator Collector: A Latency-Optimized Garbage Collector for Functional Programming

Ben Gamari Laura Dietz

1977

Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus

1991

BWM

a concrete machine for graph reduction

Lennart Augustsson

2012

Reduceron reconfigured and re-evaluated

MATTHEW NAYLOR and COLIN RUNCIMAN

2024

Heron: Modern Hardware Graph Reduction

Craig Ramsay Robert Stewart

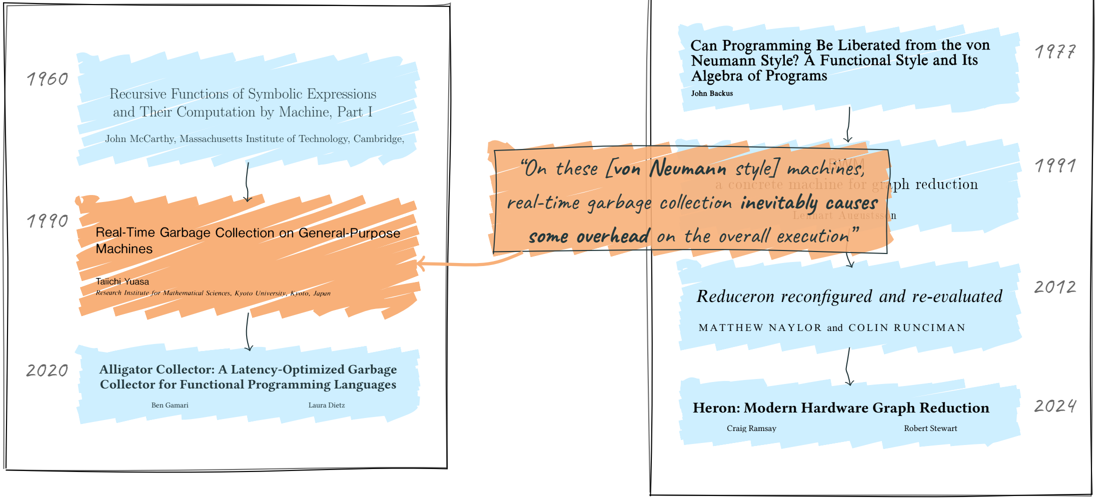
Mean 0.6 hand-reductions per cycle

x5 better than GHC* per cycle

...but a tiny Heron is **<200 MHz**

Software for Concurrent GC

Hardware for FP



1960

Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

John McCarthy, Massachusetts Institute of Technology, Cambridge,

1990

Real-Time Garbage Collection on General-Purpose Machines

Taiichi Yuasa

Research Institute for Mathematical Sciences, Kyoto University, Kyoto, Japan

2020

Alligator Collector: A Latency-Optimized Garbage Collector for Functional Programming Languages

Ben Gamari Laura Dietz

> "On these [von Neumann style] machines, real-time garbage collection **inevitably causes some overhead** on the overall execution"

1977

Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus

1991

RWM

a concrete machine for graph reduction

Lennart Augustsson

2012

Reduceron reconfigured and re-evaluated

MATTHEW NAYLOR and COLIN RUNCIMAN

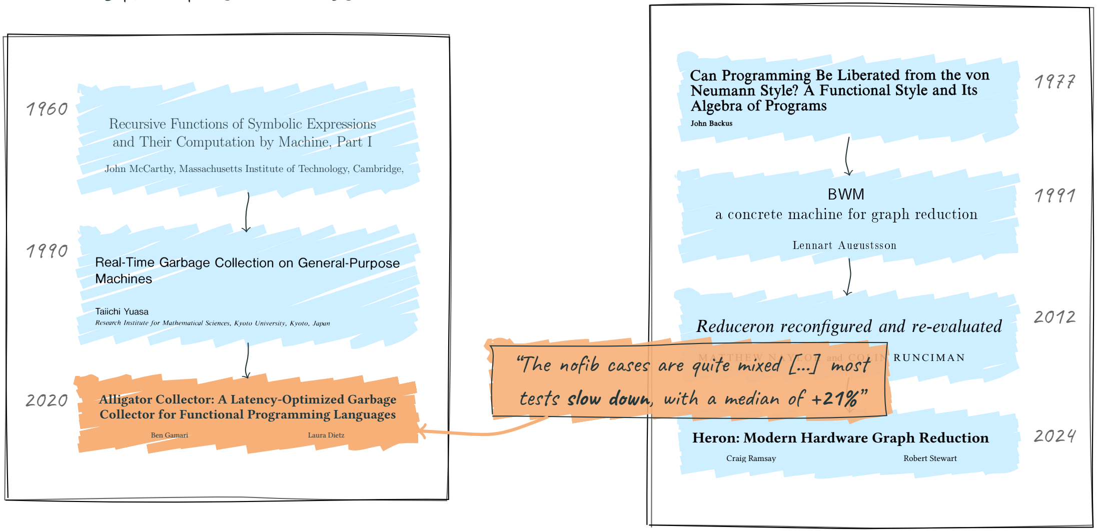
2024

Heron: Modern Hardware Graph Reduction

Craig Ramsay Robert Stewart

Software for Concurrent GC

Hardware for FP



1960

Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

John McCarthy, Massachusetts Institute of Technology, Cambridge,

1990

Real-Time Garbage Collection on General-Purpose Machines

Taiichi Yuasa
Research Institute for Mathematical Sciences, Kyoto University, Kyoto, Japan

2020

Alligator Collector: A Latency-Optimized Garbage Collector for Functional Programming Languages

Ben Gamari Laura Dietz

1977

Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus

1991

BWM
a concrete machine for graph reduction

Lennart Augustsson

2012

Reduceron reconfigured and re-evaluated

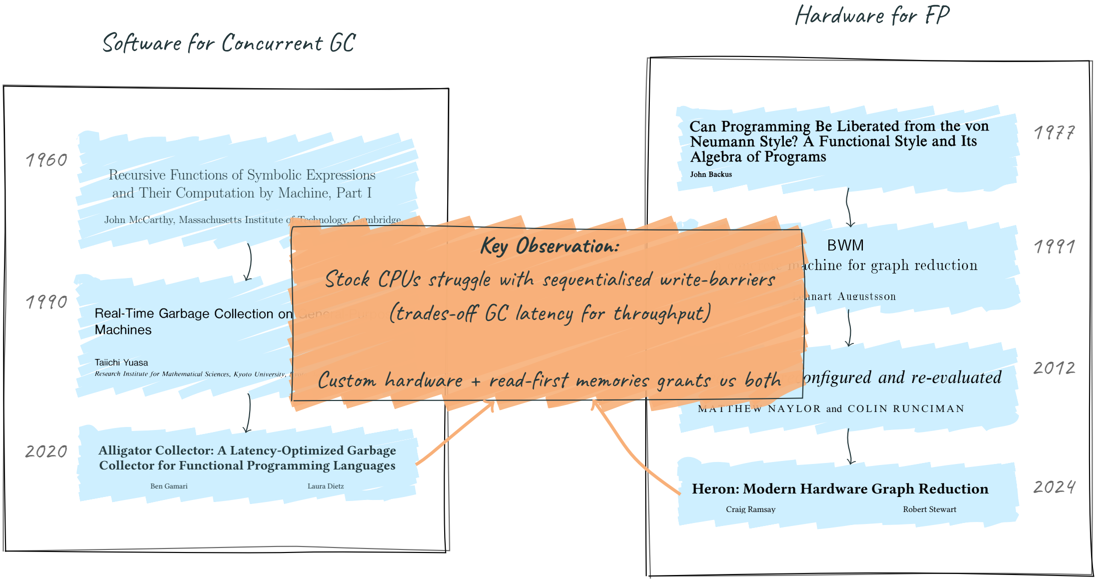
MATTHEW NAYLOR and COLIN RUNCIMAN

2024

Heron: Modern Hardware Graph Reduction

Craig Ramsay Robert Stewart

"The nofib cases are quite mixed [...] most tests **slow down**, with a median of +21%"



Software for Concurrent GC

1960

Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

John McCarthy, Massachusetts Institute of Technology, Cambridge

1990

Real-Time Garbage Collection on Gene

Machines

Taiichi Yuasa

Research Institute for Mathematical Sciences, Kyoto University,

2020

Alligator Collector: A Latency-Optimized Garbage Collector for Functional Programming Languages

Ben Gamari Laura Dietz

Hardware for FP

1977

Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus

1991

BWM

achine for graph reduction

nnart Augustsson

2012

onfigured and re-evaluated

MATTHEW NAYLOR and COLIN RUNCIMAN

2024

Heron: Modern Hardware Graph Reduction

Craig Ramsay Robert Stewart

Key Observation:

Stock CPUs struggle with sequentialised write-barriers (trades-off GC latency for throughput)

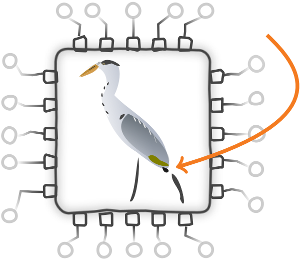
Custom hardware + read-first memories grants us both

Cloaca

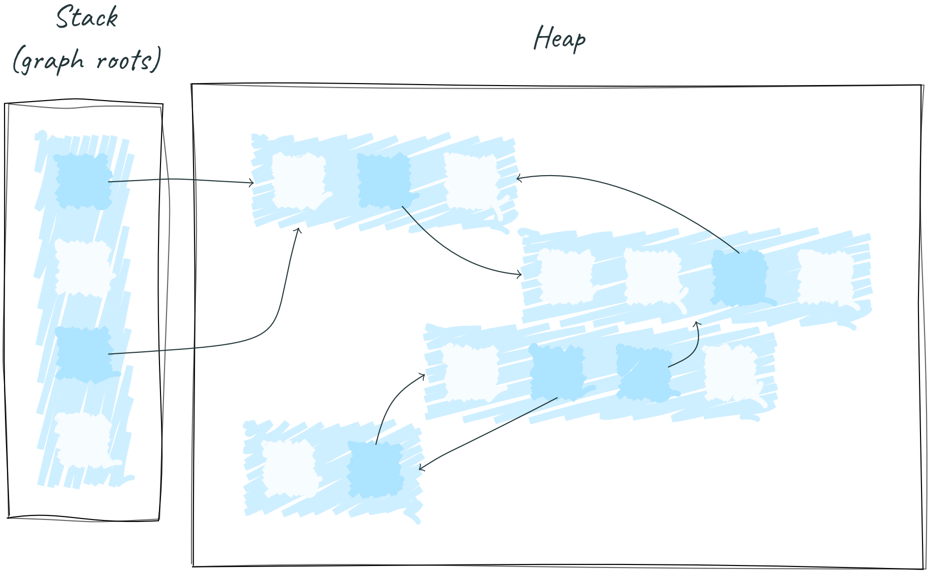
noun [C]

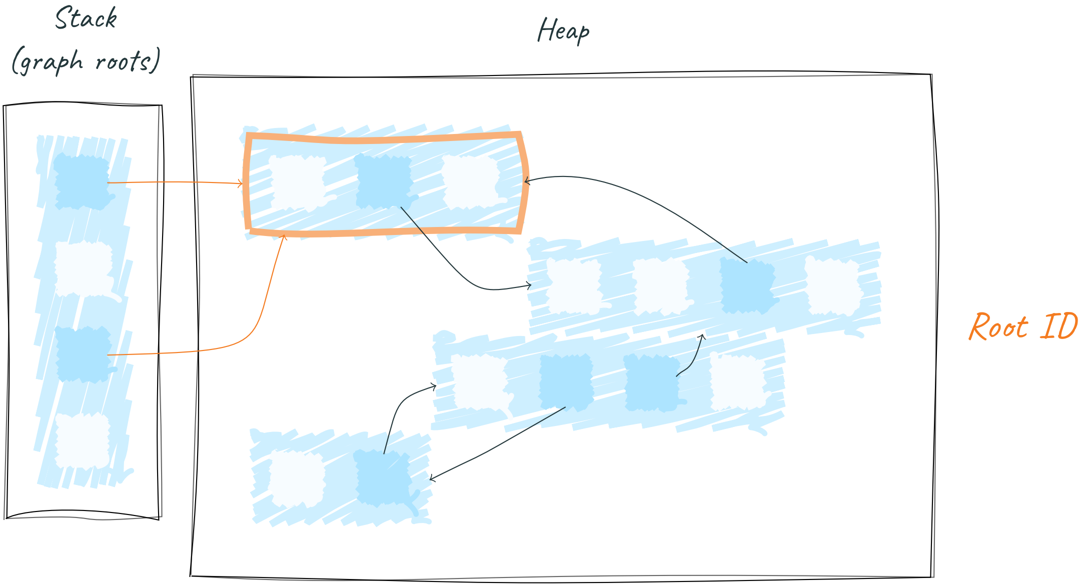
/kloh-ah-kuh/

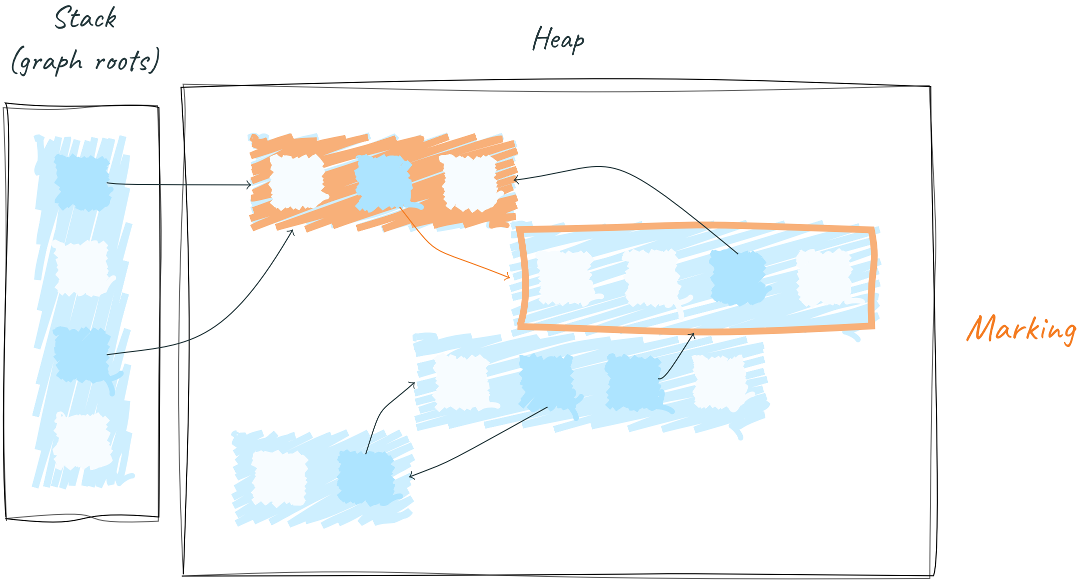
The system responsible for all the waste generated by a heron

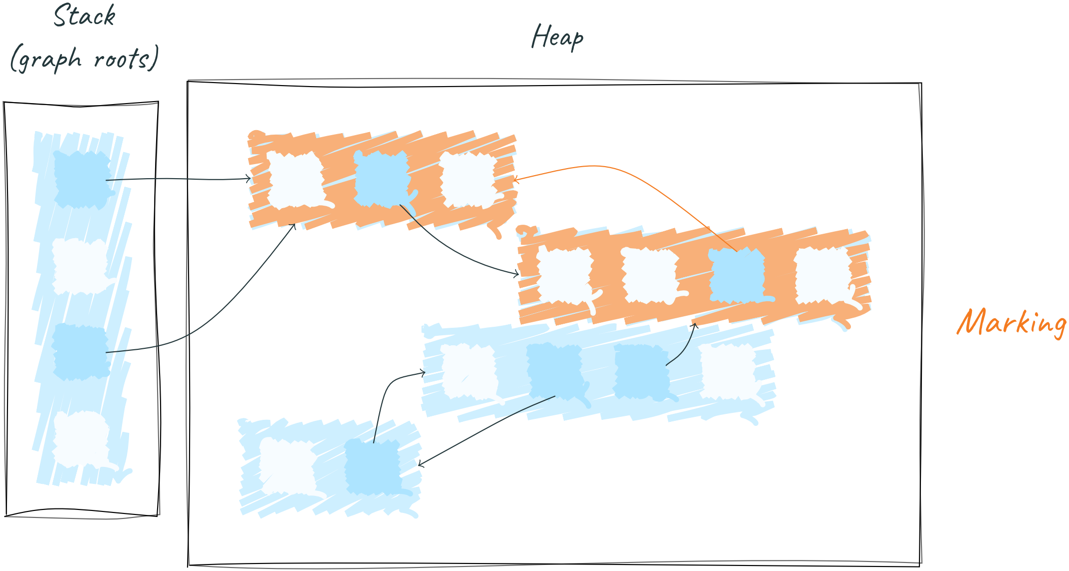


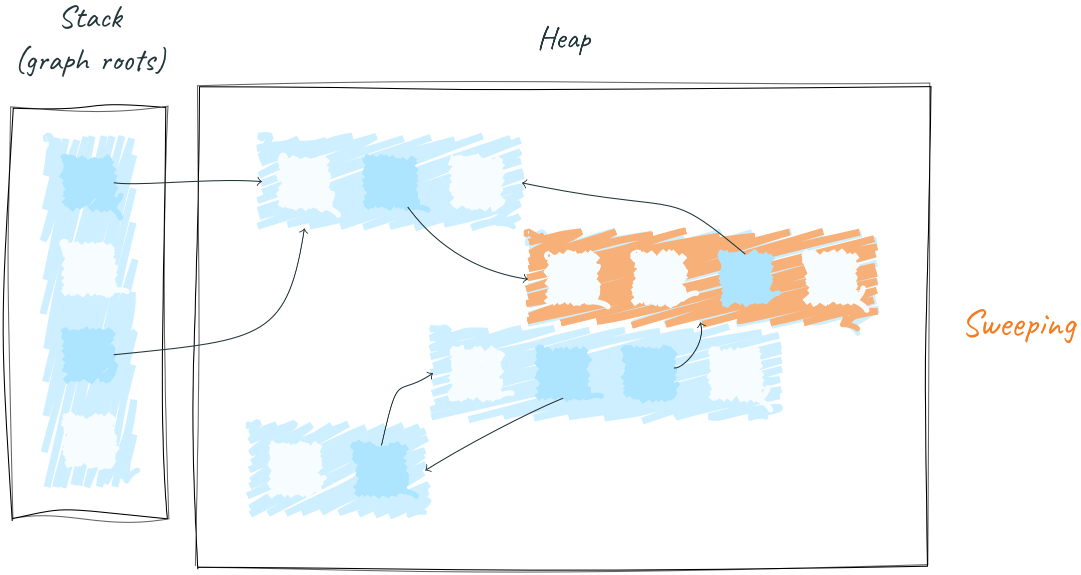
Tracing example

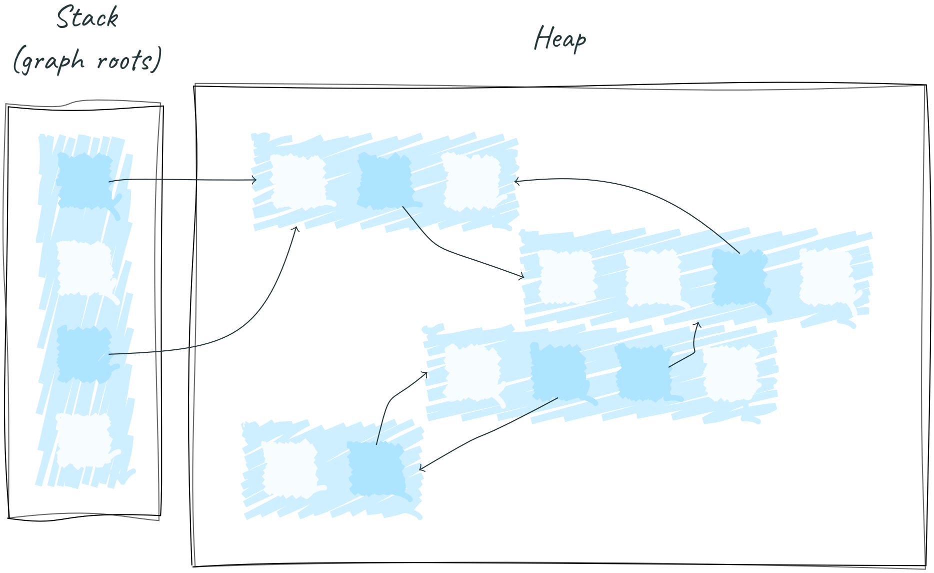

Stack
(graph roots)
Heap


Stack
(graph roots)
Heap
Root ID

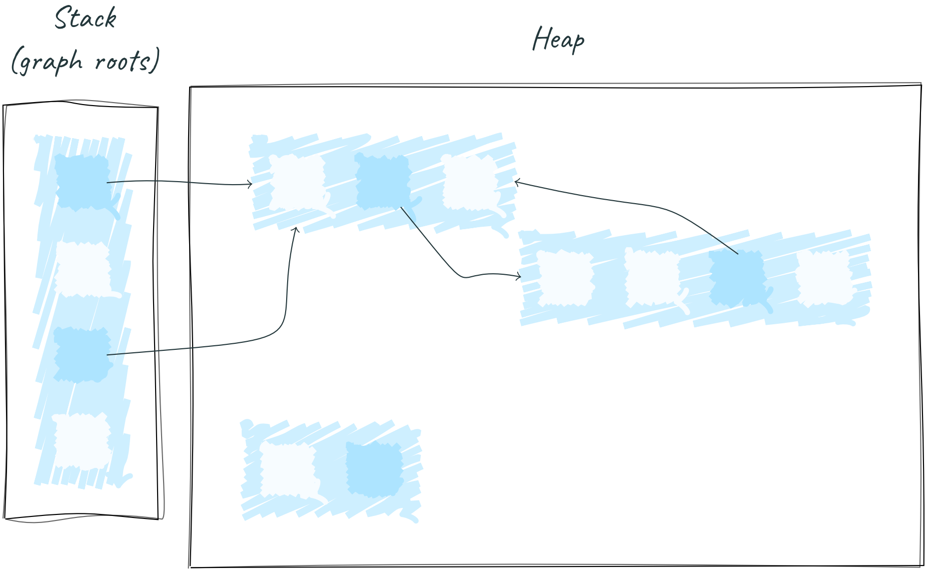

Stack
(graph roots)
Heap
Marking


Stack
(graph roots)
Heap
Marking

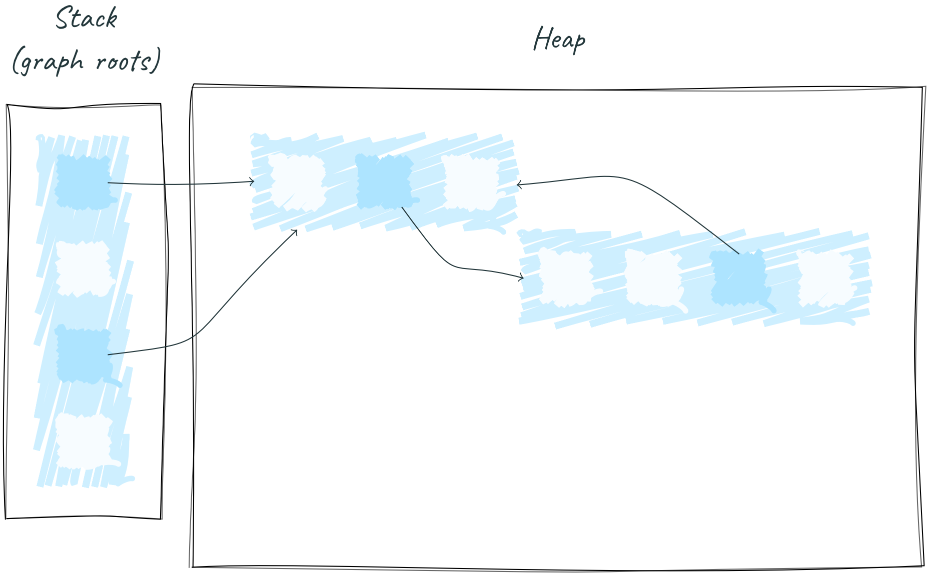

Stack
(graph roots)
Heap
Sweeping



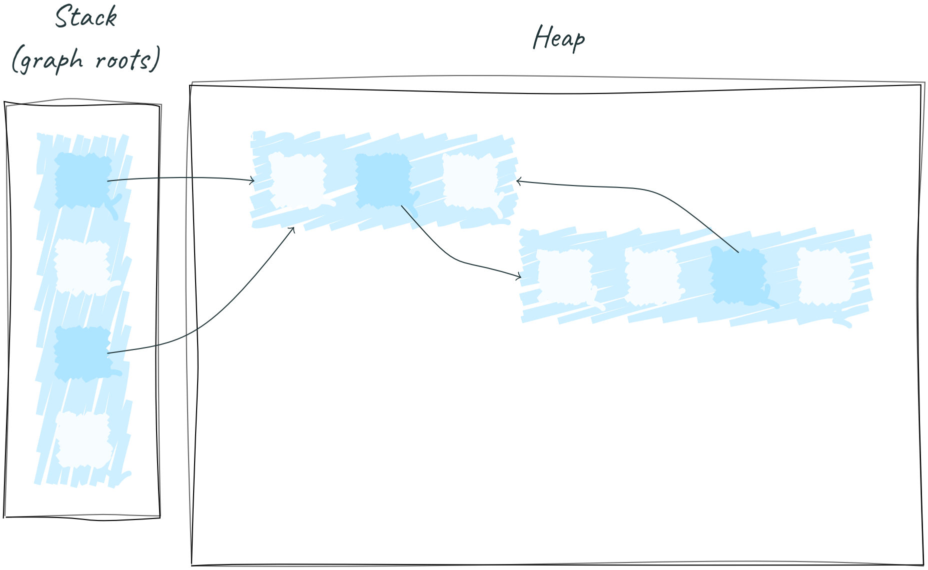
Sweeping



Sweeping



Sweeping



Sweeping

Challenges for concurrent software implementation?

Expect 21% median slowdown for nofib!

1) Allocation depends on GC state

Stop-the-world GC

```
Function alloc (app):
    heap[hp] ⟵ app
    hp++
```

vs

Concurrent GC

```
Function alloc (app):
    if allocBarrier(gcPhase, hp)
    then
        tag hp as Marked
    else
        tag hp as Unmarked
    heap[hp] ⟵ app
    hp++
```

2) Non-moving GC needs complex allocation

Stop-the-world GC vs Concurrent GC

Stop-the-world GC

```
Function alloc (app):
    heap[hp] ⟵ app
    hp++
```

Concurrent GC

```
Function alloc (app):
    a ⟵ pop from freelist
    if allocBarrier(gcPhase, a)
    then
        tag a as Marked
    else
        tag a as Unmarked
    heap[a] ⟵ app
```

3) Prevent graph updates from destroying edges

Stop-the-world GC vs Concurrent GC

```
Function update (nf, a):
    heap[a] ← nf
```

```
Function update (nf, a):
    if updateBarrier(gcPhase) then
        x ← heap[a]
        forall y in x's child pointers do
            remember y for marking
    heap[a] ← nf
```

Additional hardware-enabled optimisations

Heron's existing dynamic update avoidance system...

```
data Atom
  = ...
  | Var Shared Int
  | Arg Shared Int
    ...
```

```
Function unwind (a, shared):
    ...
    ...
    ...
    if shared and not NF then
        push a onto update stack
```

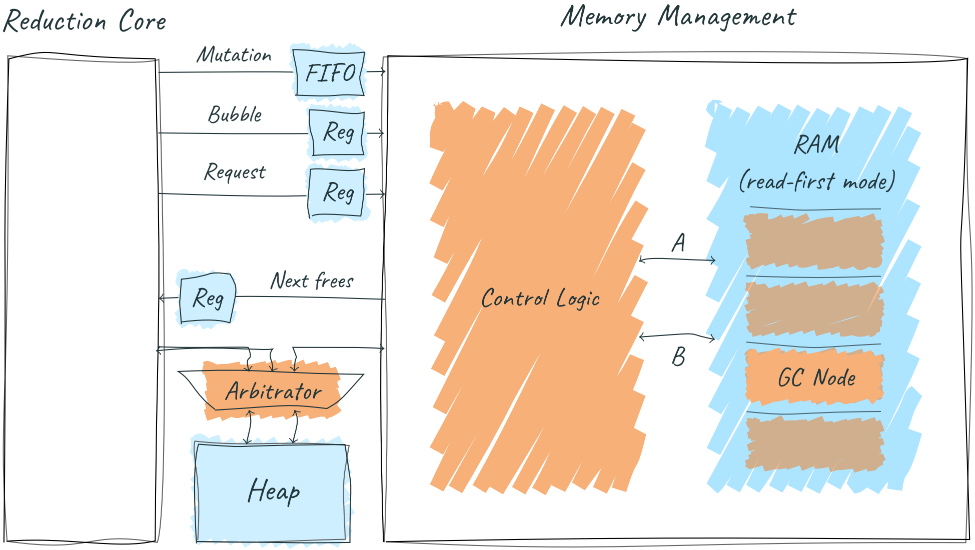
Heron's existing dynamic update avoidance system...

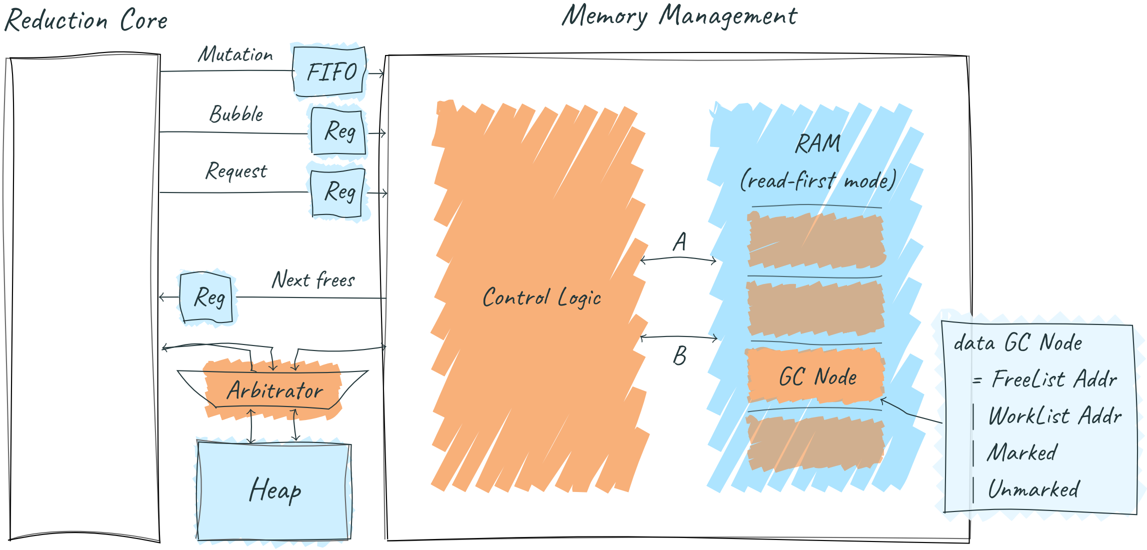
```
data Atom
  = ...
  | Var Shared Int
  | Arg Shared Int
    ...
```

```
Function unwind (a, shared):
    ...
    ...
    ...
    if shared and not NF then
        push a onto update stack
    if not shared then
        dealloc a
```

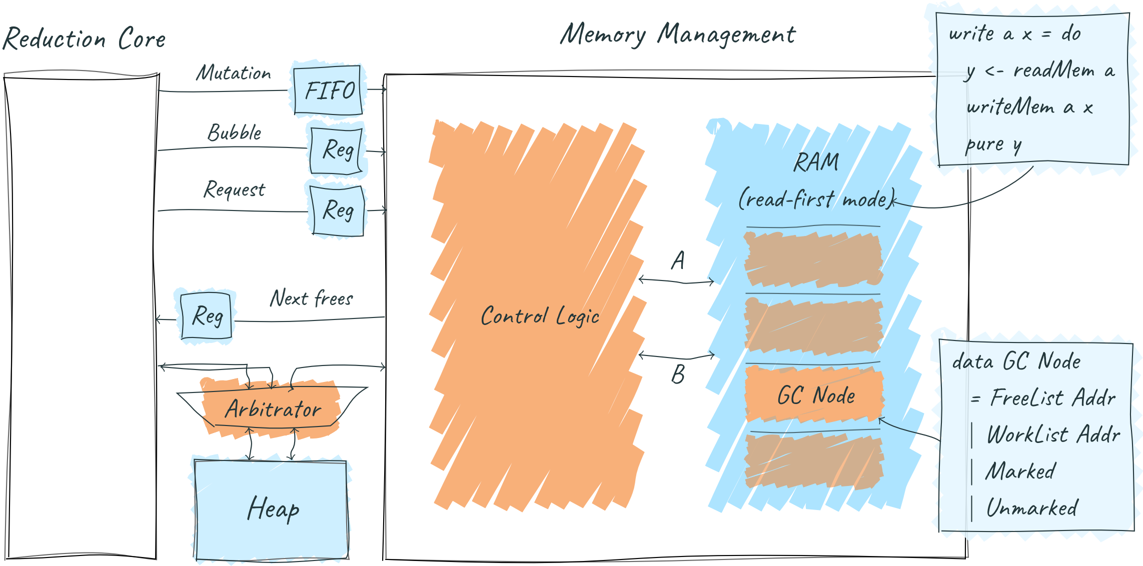
... is just one-bit reference counting with a hat on.

Architecture


Reduction Core
Memory Management
Mutation
FIFO
Bubble
Reg
Request
Reg
Reg
Next frees
Arbitrator
Heap
Control Logic
A
B
RAM
(read-first mode)
GC Node

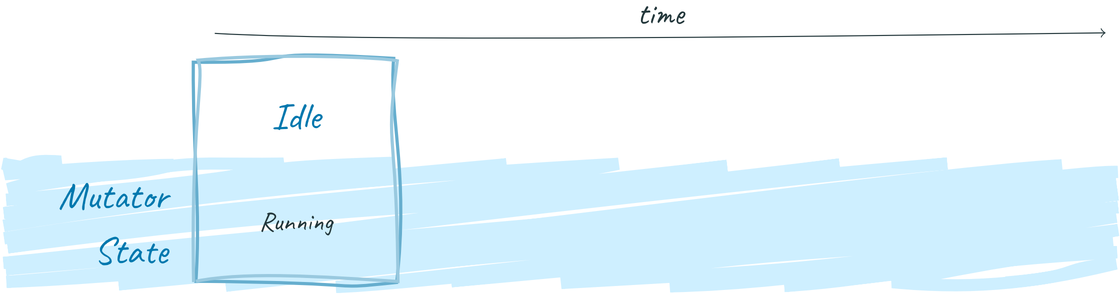


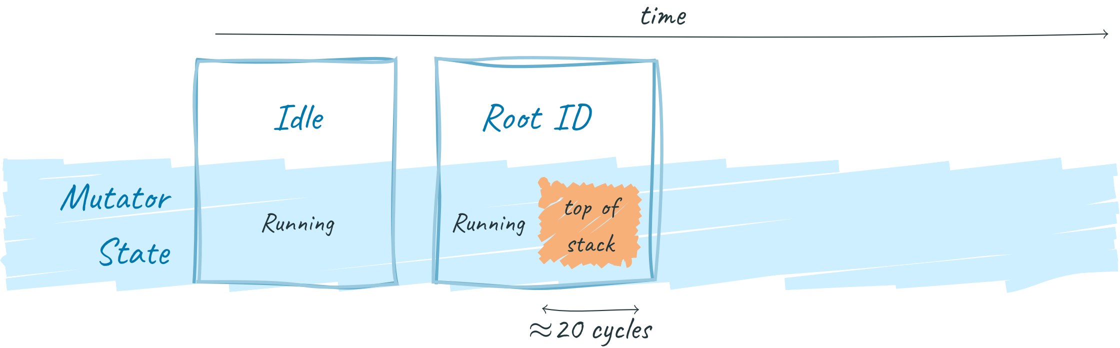
```
data GC Node
  = FreeList Addr
  | WorkList Addr
  | Marked
  | Unmarked
```

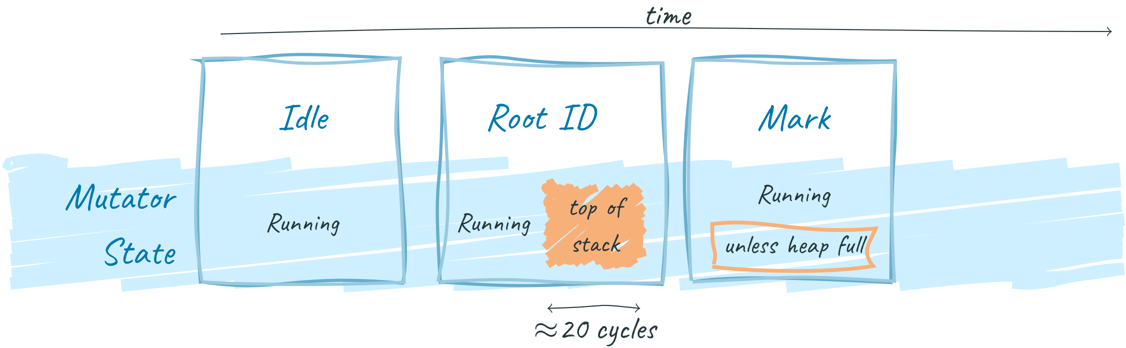


```
write a x = do
  y <- readMem a
  writeMem a x
  pure y
```

```
data GC Node
  = FreeList Addr
  | WorkList Addr
  | Marked
  | Unmarked
```


time
Idle
Mutator
State
Running


time
Mutator State
Idle
Running
Root ID
Running
top of stack
≈20 cycles



time

Idle

Root ID

Mark

Mutator State

Running

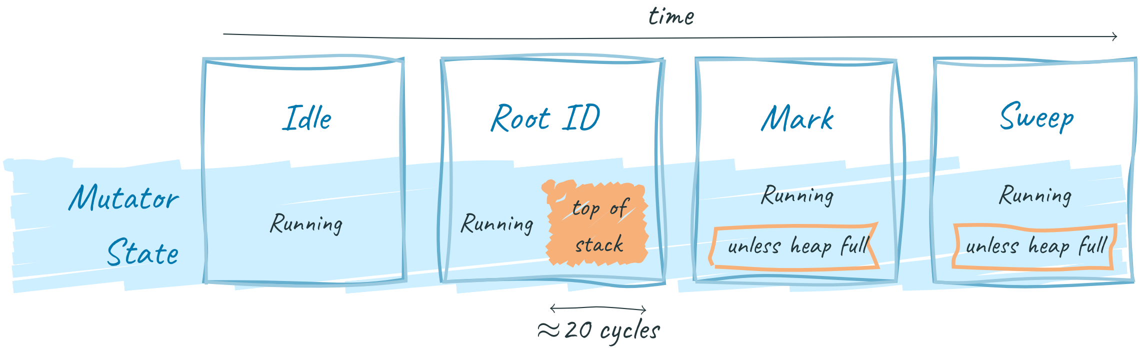
Running

top of stack

Running

unless heap full

≈20 cycles



time

	Idle	Root ID	Mark	Sweep
Mutator State	Running	Running top of stack	Running unless heap full	Running unless heap full

≈ 20 cycles

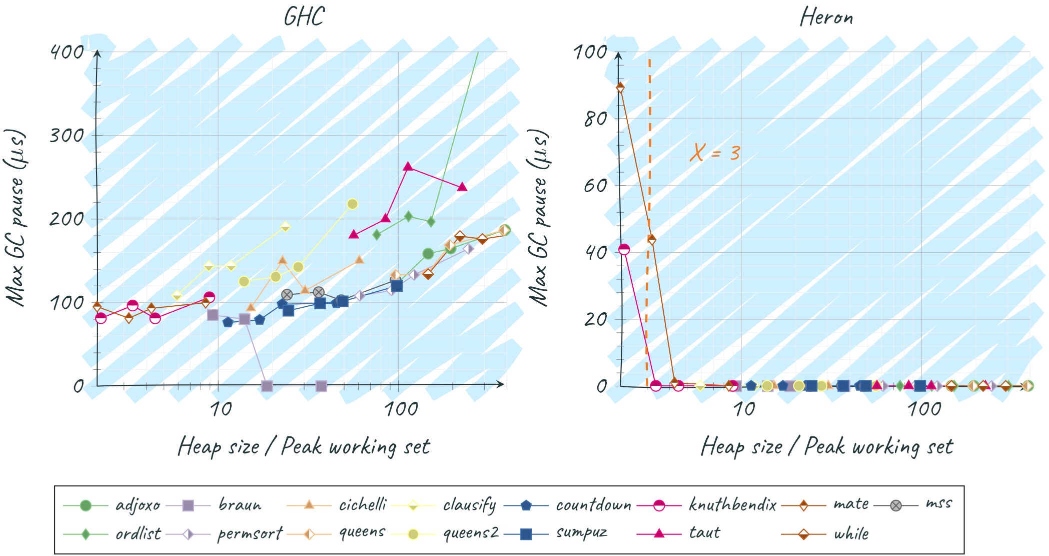
Results

	Peak GHC working set (KB)	GHC Allocations (MB)	LoC
Adjoxo	47	301	72
Braun	46	0	26
Cichelli	52	41	123
Clausify	77	364	69
Countdown	46	54	62
Knuthbendix	105	54	324
Mate	137	430	293
Mss	46	359	13
Ordlist	46	984	28
Permsort	46	2320	10
Queens	55	1038	25
Queens2	47	1084	20
Sumpuz	48	1293	72
Taut	47	236	37
While	46	264	89

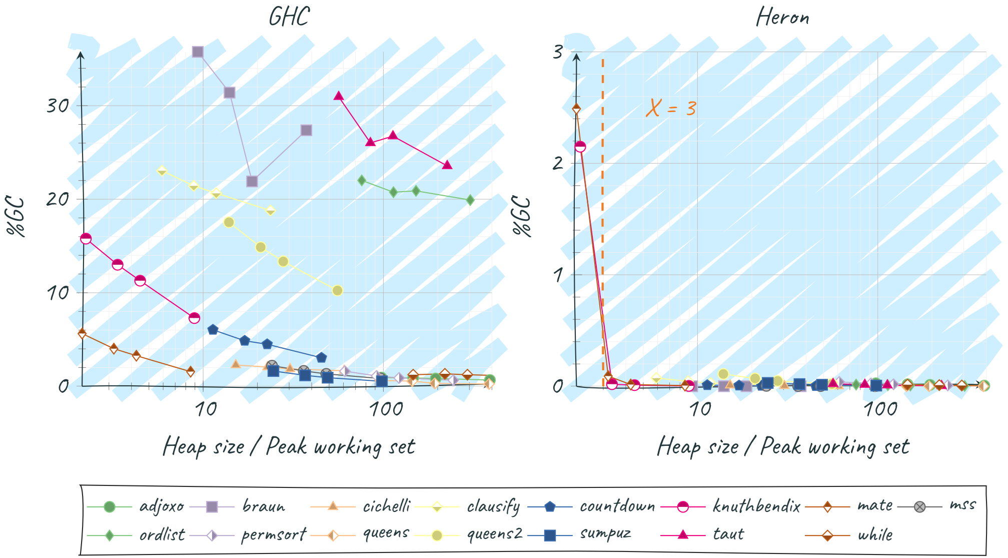
	Peak GHC working set (KB)	GHC Allocations (MB)	LoC
Adjoxo	47	301	72
Braun	46	0	26
Cichelli	52	41	123
Clausify	77	364	69
Countdown	46	54	62
Knuthbendix	105	54	324
Mate	137	430	293
Mss	46	359	13
Ordlist	46	984	28
Permsort	46	2320	10
Queens	55	1038	25
Queens2	47	1084	20
Sumpuz	48	1293	72
Taut	47	236	37
While	46	264	89

	Peak GHC working set (KB)	GHC Allocations (MB)	LoC
Adjoxo	47	301	72
Braun	46	0	26
Cichelli	52	41	123
Clausify	77	364	69
Countdown	46	54	62
Knuthbendix	105	54	324
Mate	137	430	293
Mss	46	359	13
Ordlist	46	984	28
Permsort	46	2320	10
Queens	55	1038	25
Queens2	47	1084	20
Sumpuz	48	1293	72
Taut	47	236	37
While	46	264	89

GC worst-case pause



GC wall-clock overhead



Concurrent tracing GC → high throughput and low latency?

Concurrent tracing GC → high throughput and low latency?

Stock processors + software struggle to maintain throughput.

Write-barriers and friends are hard.

Concurrent tracing GC → high throughput and low latency?

Stock processors + software struggle to maintain throughput.
Write-barriers and friends are hard.

Custom hardware with dual-port read-first memories
can handle them in a single cycle.

Concurrent tracing GC → high throughput and low latency?

Stock processors + software struggle to maintain throughput.
Write-barriers and friends are hard.

Custom hardware with dual-port read-first memories
can handle them in a single cycle.

Cloaca often pauses for only ≈ 20 cycles for a GC pass,

Concurrent tracing GC → high throughput and low latency?

Stock processors + software struggle to maintain throughput.
Write-barriers and friends are hard.

Custom hardware with dual-port read-first memories
can handle them in a single cycle.

Cloaca often pauses for only $\approx$ 20 cycles for a GC pass,
and catches $\approx$ 50% of all our garbage before tracing.

Questions?

Break glass in case of emergency

```
tails  [ ]        = [ ]
tails  (x : xs) = (x : xs)  :  tails  xs

inits  xs =
  case  xs  of
    [ ]        -> [[  ]]
    (y  :  ys) -> xs :  inits  ( init  xs)

segments xs = concatMap tails (inits xs)

mss = maximum . map sum . segments

main = let x = 0 - 50
           y = 150
       in  mss $ enumFromTo x y
```

$e ::=$		Expressions
	$\overline{e}$	(Application)
	$\mid \mathit{case}\ e\ \mathit{of}\ \overline{a}$	(Case expression)
	$\mid \mathit{let}\ \overline{b}\ \mathit{in}\ e$	(Let expression)
	$\mid n$	(Integer)
	$\mid x$	(Variable)
	$\mid \otimes$	(Primitive Op)
	$\mid f$	(Function)
	$\mid K$	(Constructor)
$a ::= K\,\overline{x} \rightarrow e$		Case alternative
$b ::= x \mapsto e$		Let binding
$d ::= f\,\overline{x} = e$		Function definition

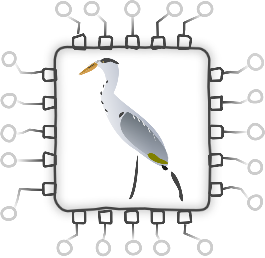
Heron

noun [C]

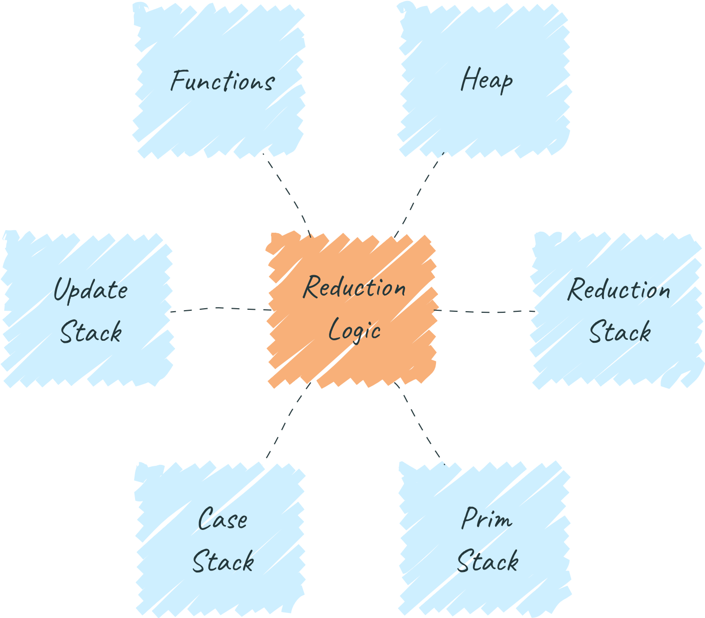
/”herə n/

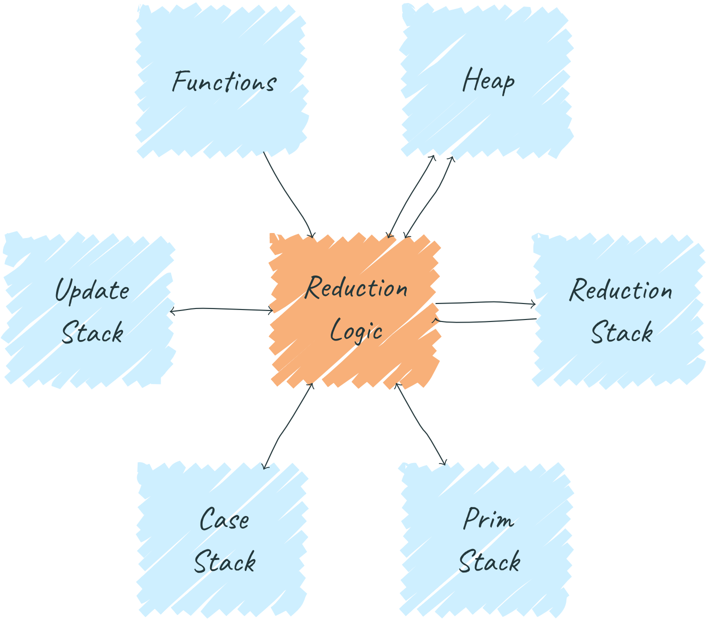
A processor for lazy functional languages.

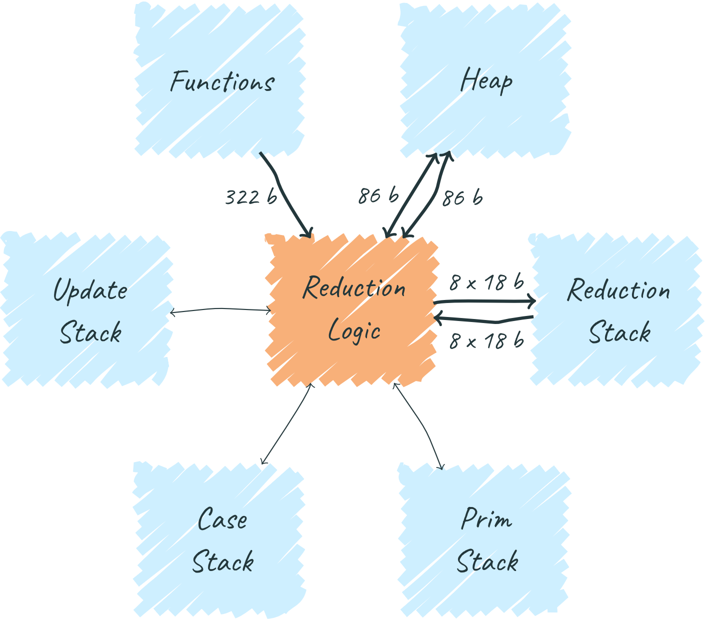
Performs beta reduction in one clock cycle via multiple, wide, multi-ported memories.

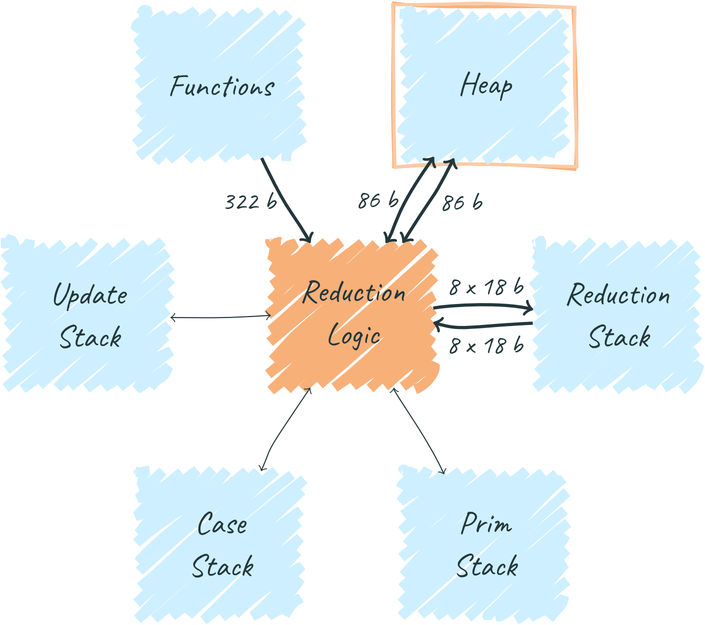


Reduction Logic


Functions
Heap
Update Stack
Reduction Logic
Reduction Stack
Case Stack
Prim Stack


Functions
Heap
Update Stack
Reduction Logic
Reduction Stack
Case Stack
Prim Stack


Functions
Heap
322 b
86 b
86 b
Update
Stack
Reduction
Logic
8 x 18 b
Reduction
Stack
8 x 18 b
Case
Stack
Prim
Stack



Functions

Heap

322 b

86 b

86 b

Update Stack

Reduction Logic

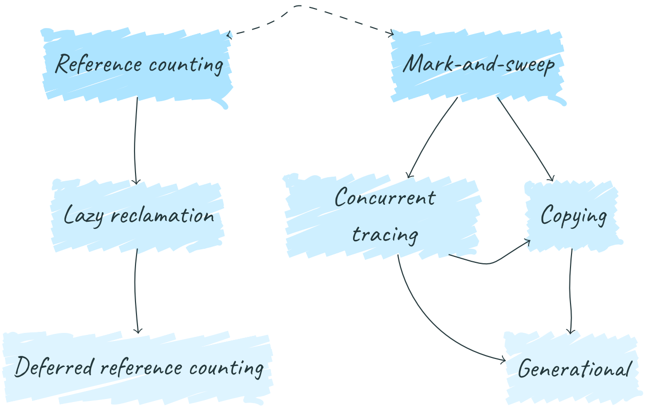
8 x 18 b

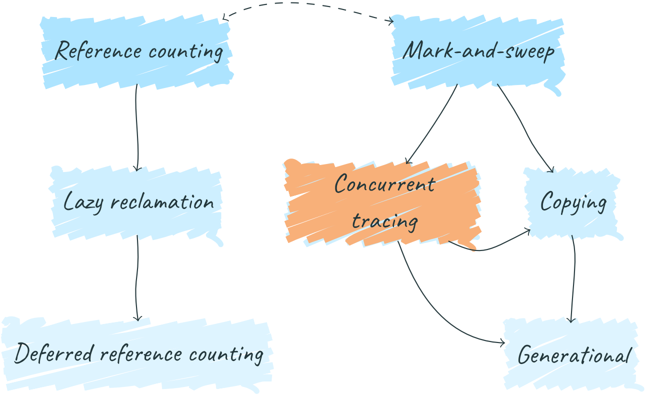
8 x 18 b

Reduction Stack

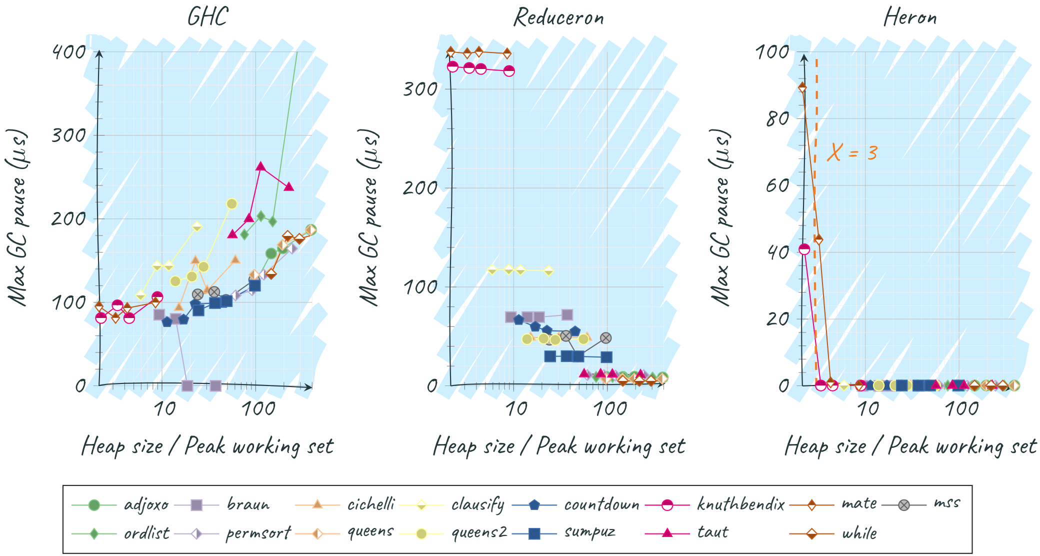
Case Stack

Prim Stack

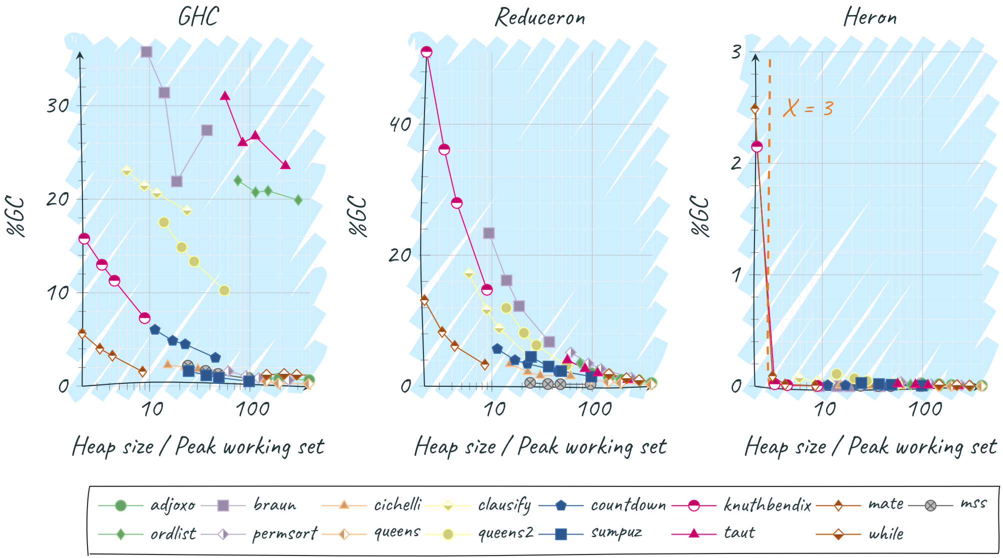

Reference counting
Mark-and-sweep
Lazy reclamation
Concurrent tracing
Copying
Deferred reference counting
Generational


Reference counting
Mark-and-sweep
Lazy reclamation
Concurrent tracing
Copying
Deferred reference counting
Generational

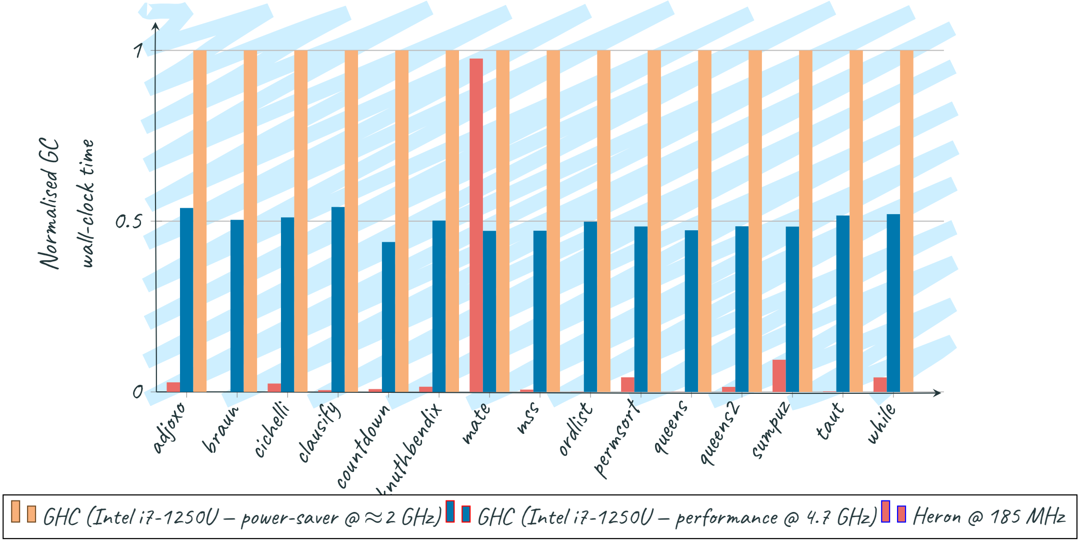
GC worst-case pause



GC wall-clock overhead



Total GC wall-clock pause


Normalised GC
wall-clock time
adjoxo
braun
cichelli
clausify
countdown
knuthbendix
mate
mss
ordlist
permsort
queens
queens2
sumpuz
taut
while
GHC (Intel i7-1250U – power-saver @ ≈ 2 GHz)
GHC (Intel i7-1250U – performance @ 4.7 GHz)
Heron @ 185 MHz