Once bittern, twice shy

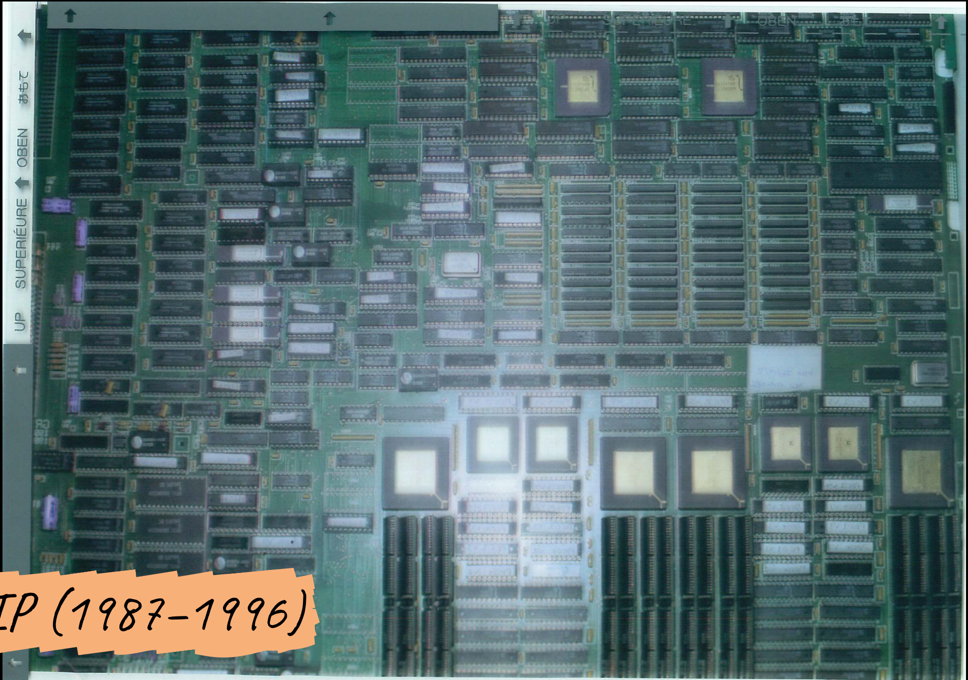
Revisiting hardware architectures for lazy functional languages with Heron

Craig Ramsay & Rob Stewart

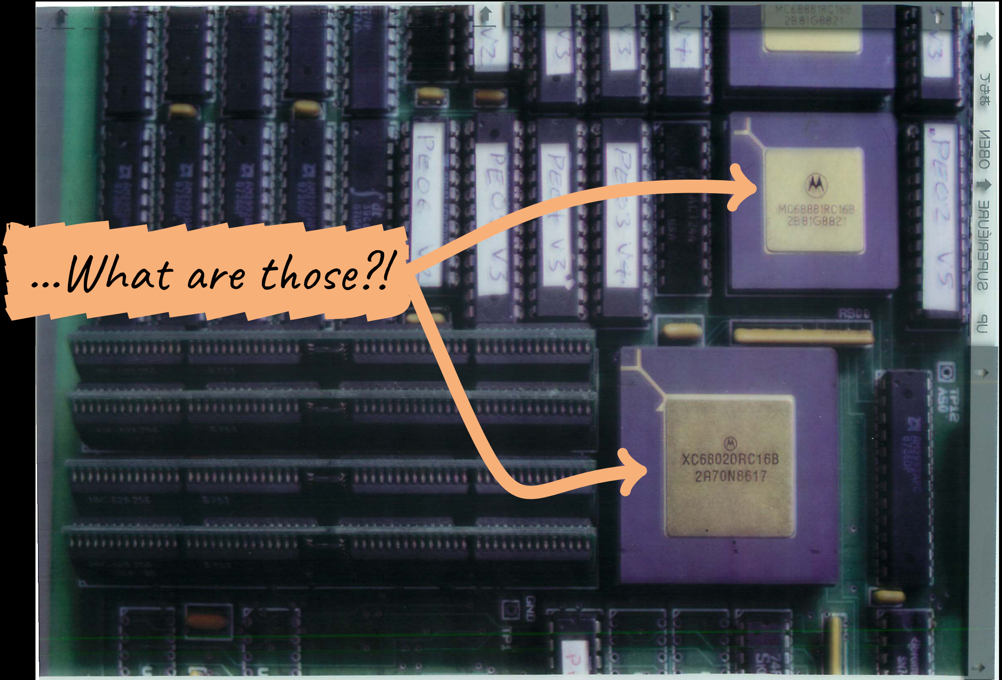
September 2024

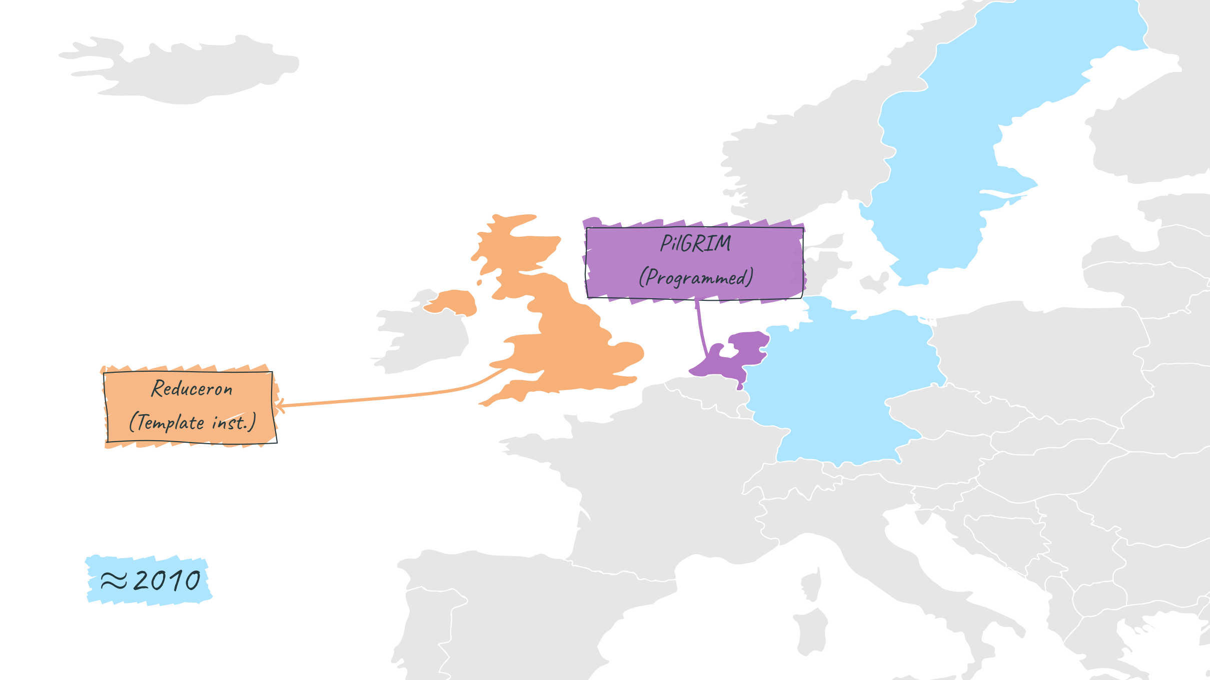
Heriot-Watt University

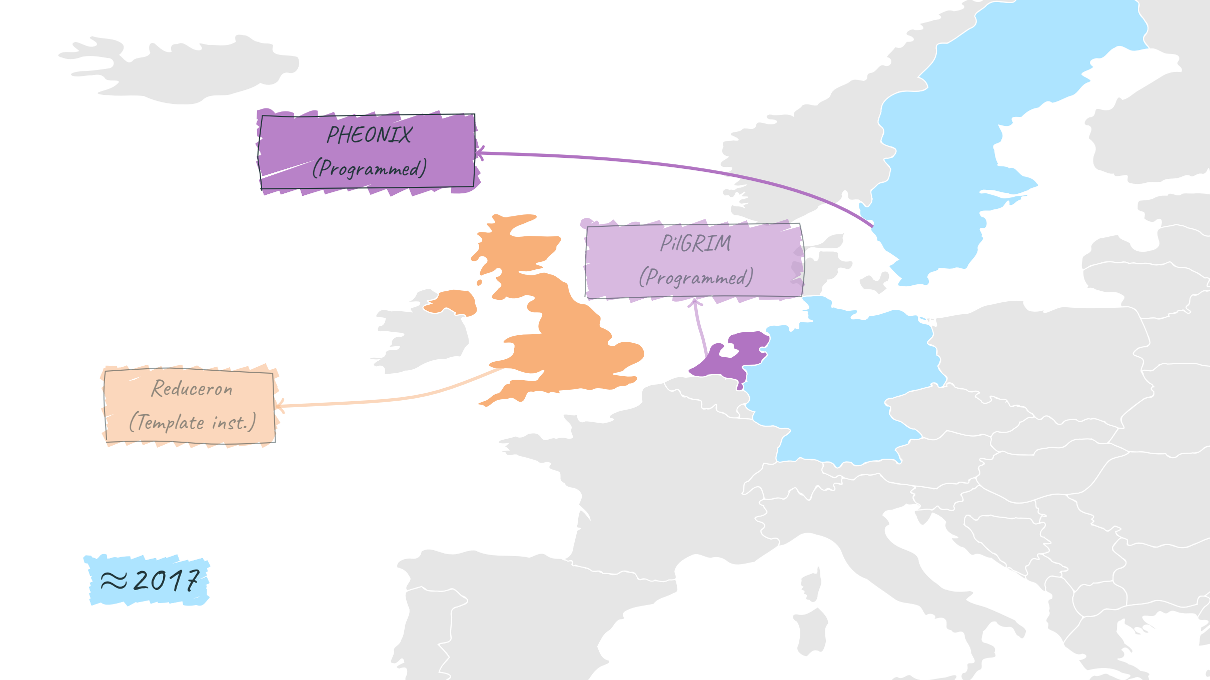

https://haflang.github.io/history

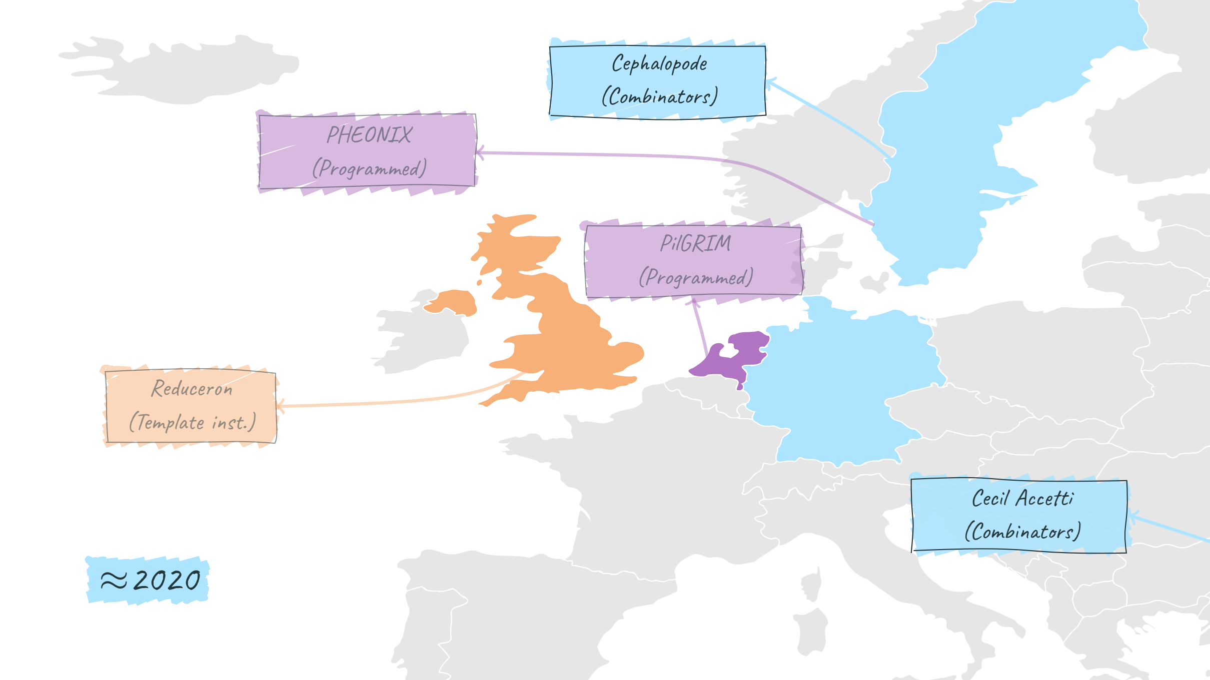


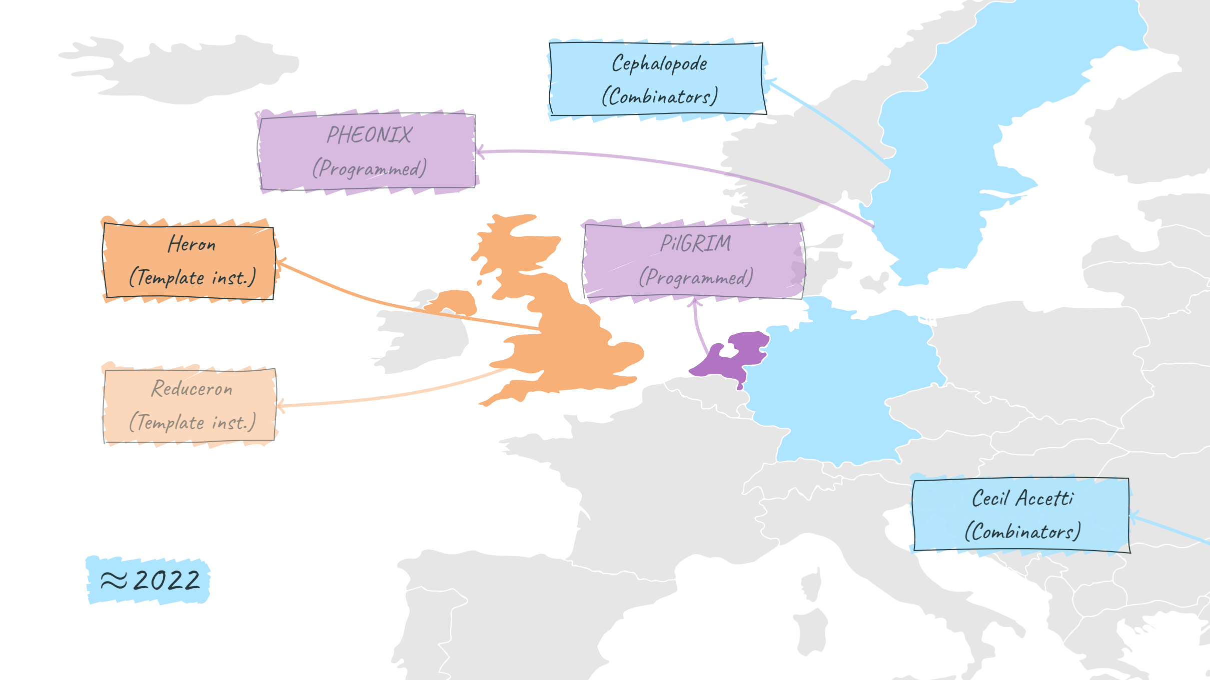
GRIP (1987–1996)


...What are those?!
MC68881RC16B
2B81G8821
XC68020RC16B
2A70N8617

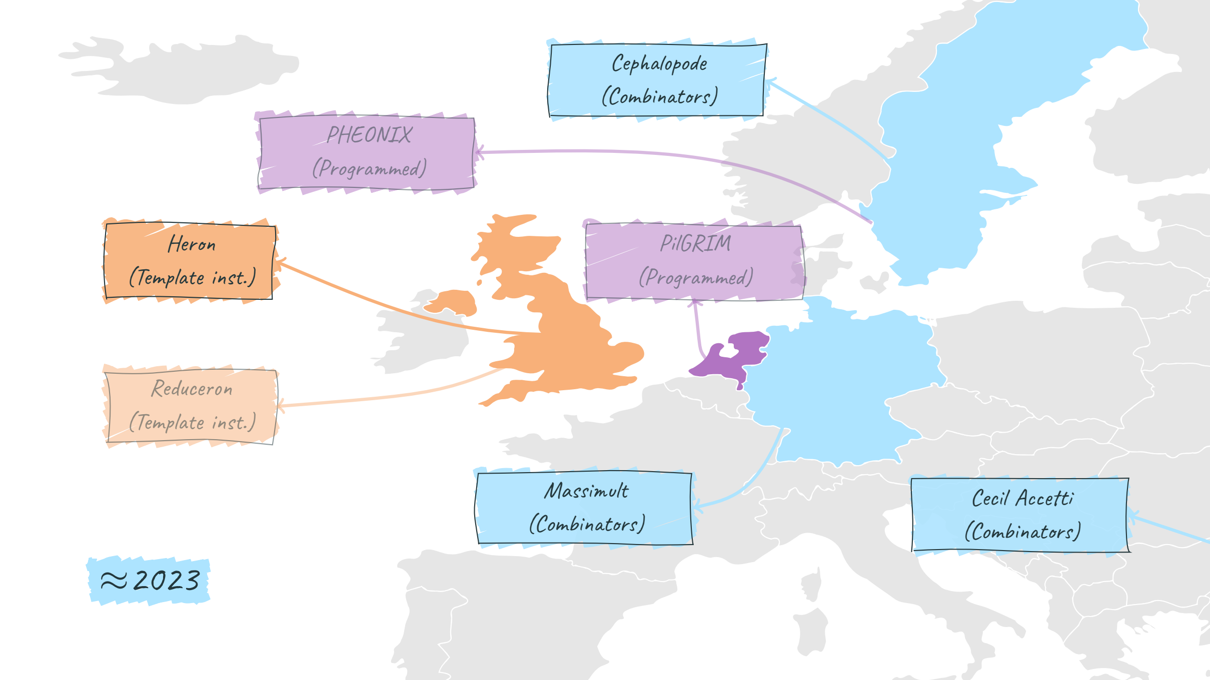

PilGRIM
(Programmed)
Reduceron
(Template inst.)
≈2010


PHEONIX
(Programmed)
PilGRIM
(Programmed)
Reduceron
(Template inst.)
≈2017


Cephalopode
(Combinators)
PHEONIX
(Programmed)
PilGRIM
(Programmed)
Reduceron
(Template inst.)
Cecil Accetti
(Combinators)
≈2020

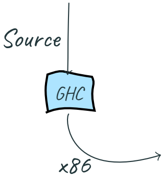


Cephalopode
(Combinators)
PHEONIX
(Programmed)
Heron
(Template inst.)
PilGRIM
(Programmed)
Reduceron
(Template inst.)
Cecil Accetti
(Combinators)
≈2022

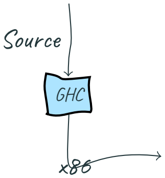

Cephalopode
(Combinators)
PHEONIX
(Programmed)
Heron
(Template inst.)
PilGRIM
(Programmed)
Reduceron
(Template inst.)
Massimult
(Combinators)
Cecil Accetti
(Combinators)
≈2023

HAFLANG Project

```
add x y z =
  x + y + z
```



```haskell
add x y z =
  x + y + z
```




```
s1ba_info: ; Code for helper (\a b -> a+b)
.Lc1bo: ; Check for stack space
  leaq -40(%rbp),%rax
  cmpq %r15,%rax
  jb .Lc1bp ; Jump if stack full
.Lc1bq: ; Reduce helper
  movq $stg_upd_frame_info,-16(%rbp)
  movq %rbx,-8(%rbp)
  movq 16(%rbx),%rax ; Load a & b from heap
  movq 24(%rbx),%rbx
  movl $base_GHCziNum_zdfNumInt_closure,%r14d
  ;; Push `a+b` onto stack
  movq $stg_ap_pp_info,-40(%rbp)
  movq %rax,-32(%rbp)
  movq %rbx,-24(%rbp)
  addq $-40,%rbp
  jmp base_GHCziNum_zp_info ; Enter
.Lc1bp: ; Ask RTS for stack space
  jmp *-16(%r13)
Add_add_info: ; Code for `add`
.Lc1br: ; Check for stack space
  leaq -24(%rbp),%rax
  cmpq %r15,%rax
  jb .Lc1bs ; Jump if stack full
.Lc1bt: ; Check for heap space
  addq $32,%r12
  cmpq 856(%r13),%r12
  ja .Lc1bv ; Jump if heap full
.Lc1bu: ; Reduce `add`
  ;; Build `x+y` thunk on heap
  movq $s1ba_info,-24(%r12)
  movq %r14,-8(%r12)
  movq %rsi,(%r12)
  leaq -24(%r12),%rax
  movl $base_GHCziNum_zdfNumInt_closure,%r14d
  ;; Push `thunk+z` to stack
  movq $stg_ap_pp_info,-24(%rbp)
  movq %rax,-16(%rbp)
  movq %rdi,-8(%rbp)
  addq $-24,%rbp
  jmp base_GHCziNum_zp_info ; Enter
.Lc1bv: ; Ask RTS for heap space
  movq $32,904(%r13)
.Lc1bs: ; Ask RTS for stack space
  movl $Add_add_closure,%ebx
  jmp *-8(%r13)
```

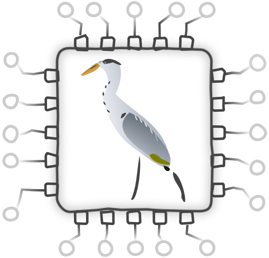
Heron

noun [C]

/ˈherən/

A **graph reduction** processor.

Performs template instantiation in one clock cycle via multiple, wide, multi-ported memories.



[0] Ramsay and Stewart, "Heron: Modern Graph Reduction Hardware".

Heron

noun [C]

/ˈherə n/

A **graph reduction** processor.

Performs template instantiation in one clock cycle via multiple, wide, multi-ported memories.

[0] Ramsay and Stewart, "Heron: Modern Graph Reduction Hardware".

Template example

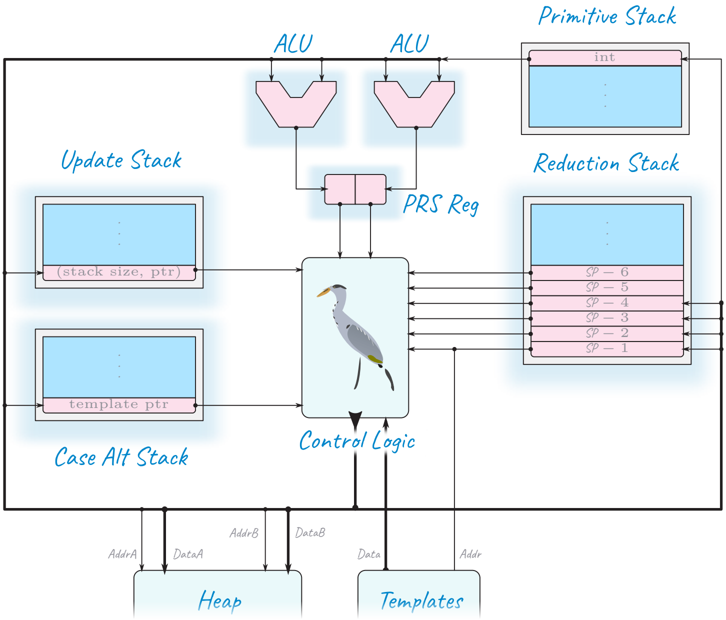
```
enumFrom :: Int -> [Int]
enumFrom n = let m = n + 1
                 ns = enumFrom m
             in Cons n ns
```

Template example

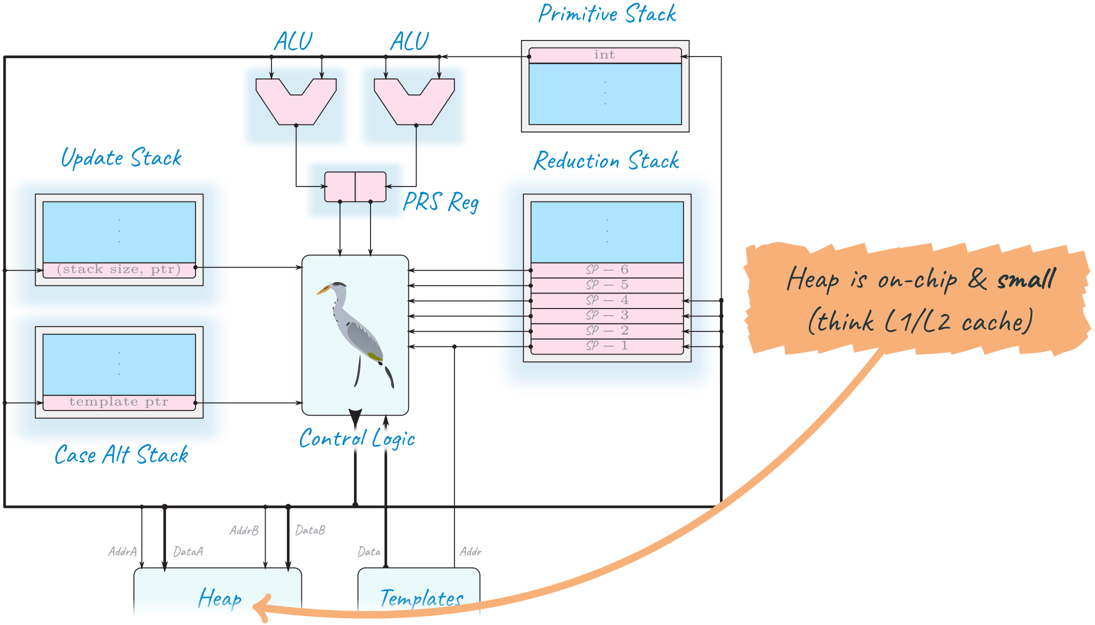
```
enumFrom :: Int -> [Int]
enumFrom n = let m = n + 1
                 ns = enumFrom m
             in Cons n ns
```

↓

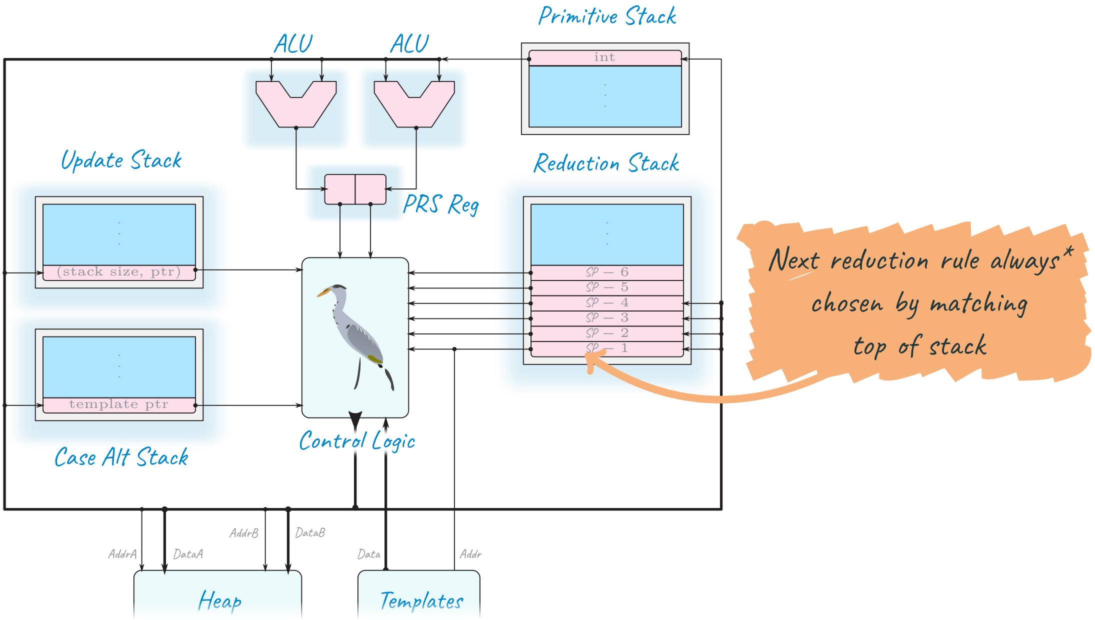
let	APP [	ARG True 0,	PRI 2 +,	INT 1]
	APP [	FUN True 1 0,	VAR False 0,	]
in	APP [	CON 2 0,	ARG True 0,	VAR False 1]

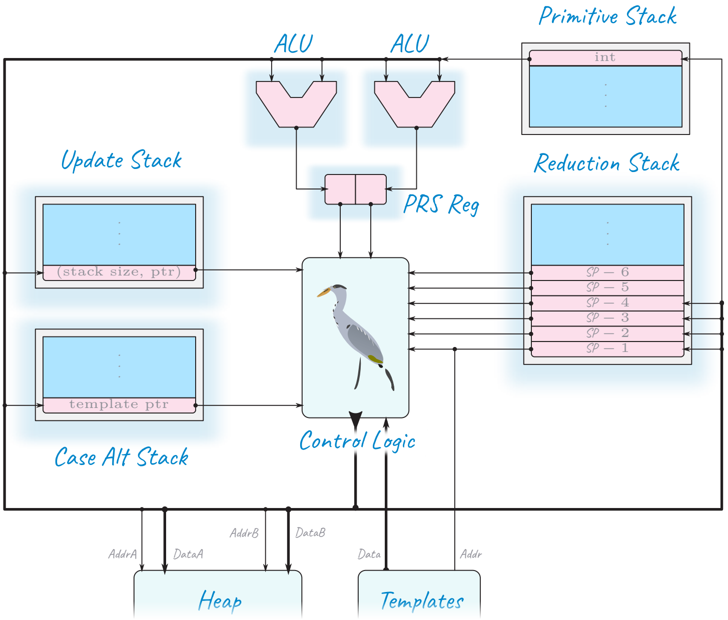


Primitive Stack
int
ALU
ALU
Update Stack
PRS Reg
Reduction Stack
(stack size, ptr)
SP — 6
SP — 5
SP — 4
SP — 3
SP — 2
SP — 1
template ptr
Control Logic
Case Alt Stack
AddrA
DataA
AddrB
DataB
Data
Addr
Heap
Templates



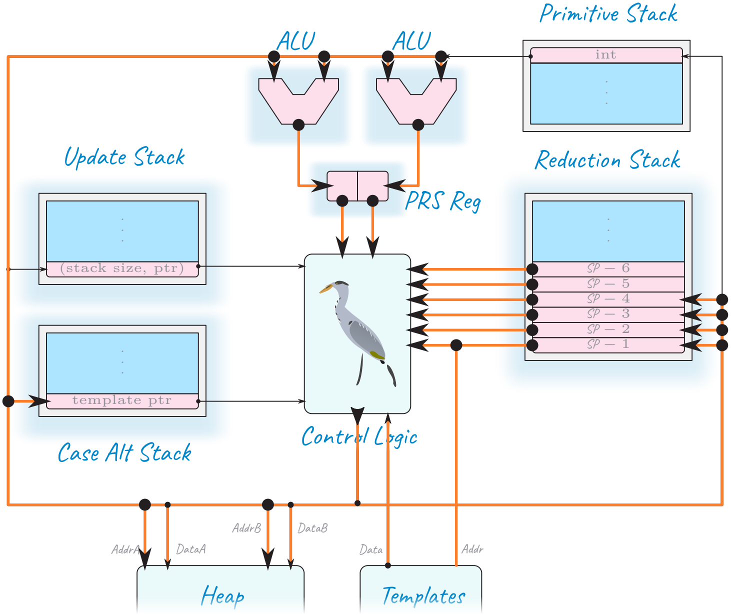
Primitive Stack
ALU
ALU
int
Update Stack
Reduction Stack
PRS Reg
(stack size, ptr)
SP − 6
SP − 5
SP − 4
SP − 3
SP − 2
SP − 1
Heap is on-chip & small
(think L1/L2 cache)
template ptr
Case Alt Stack
Control Logic
AddrA
DataA
AddrB
DataB
Data
Addr
Heap
Templates


Primitive Stack
ALU
ALU
int
Update Stack
Reduction Stack
PRS Reg
(stack size, ptr)
SP − 6
SP − 5
SP − 4
SP − 3
SP − 2
SP − 1
Next reduction rule always*
chosen by matching
top of stack
template ptr
Case Alt Stack
Control Logic
AddrA
DataA
AddrB
DataB
Data
Addr
Heap
Templates



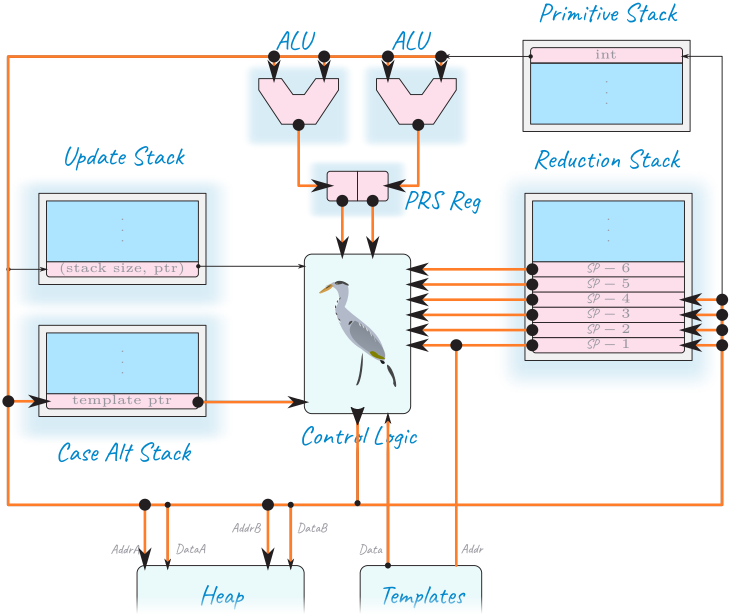
Atoms

= FUN s a n
| CON a n
| VAR s n
| INT n
| PRI a ⊗
| ARG s n
| REG n



Atoms

$=$ FUN s a n
| CON a n
| VAR s n
| INT n
| PRI a $\otimes$
| ARG s n
| REG n



Atoms

= FUN s a n

| CON a n

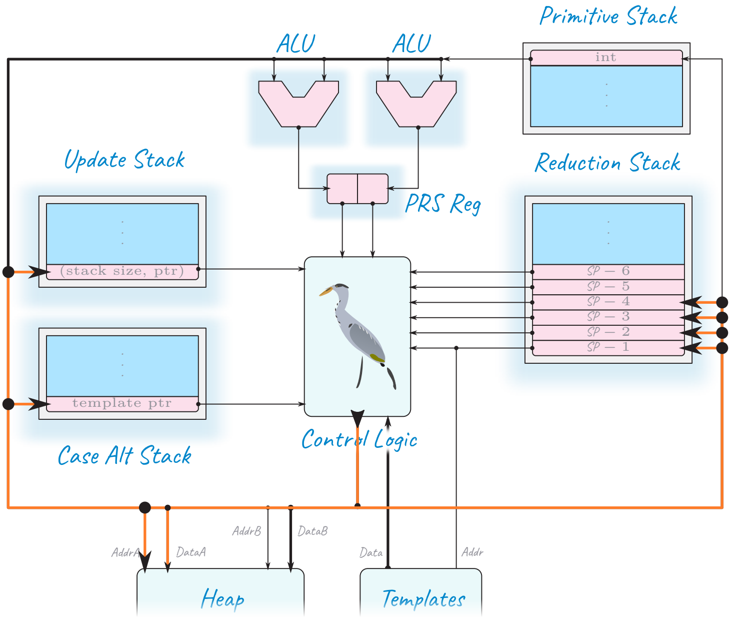
| VAR s n

| INT n

| PRI a $\otimes$

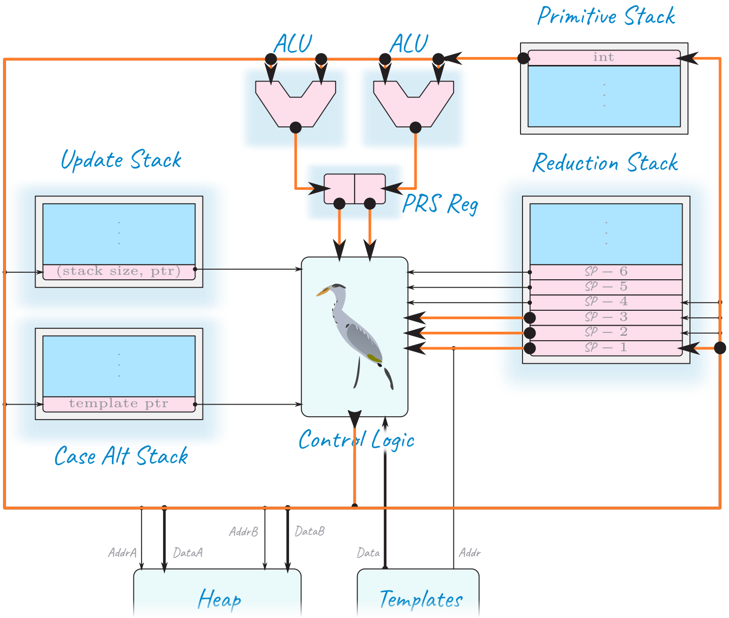
| ARG s n

| REG n



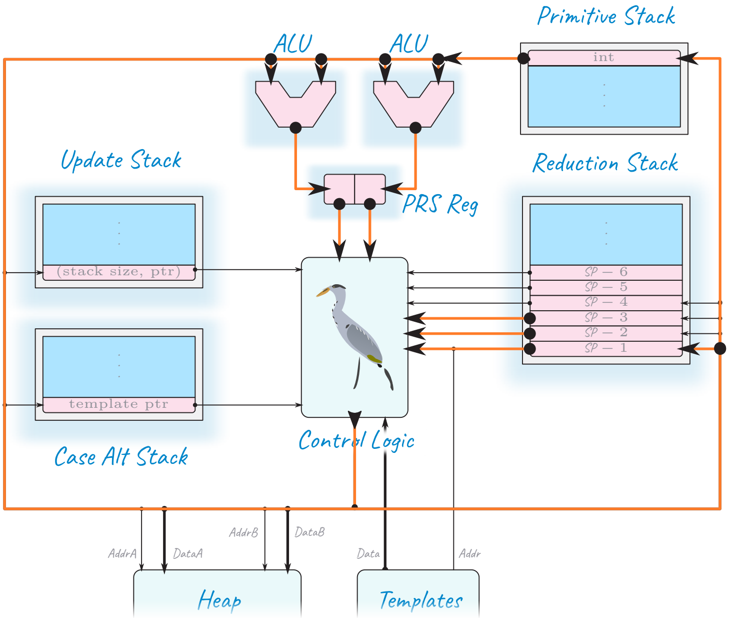
Atoms

= FUN s a n
| CON a n
| VAR s n
| INT n
| PRI a ⊗
| ARG s n
| REG n



Atoms

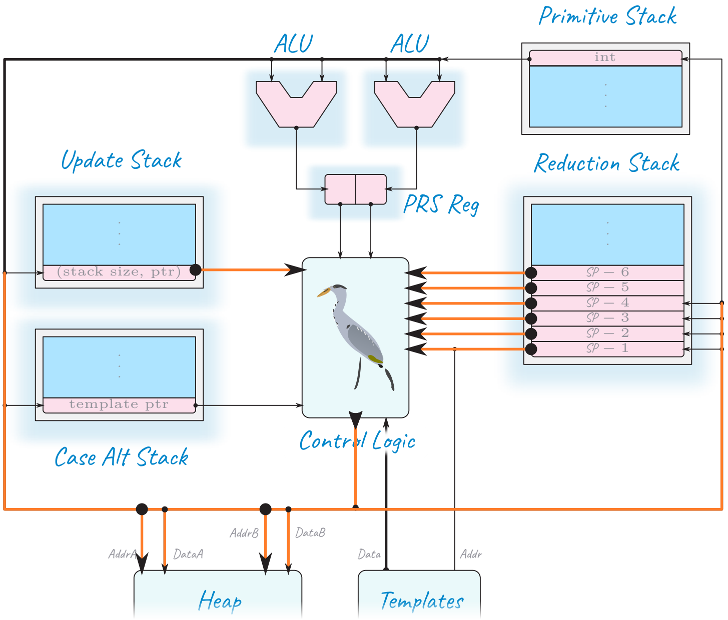
= FUN s a n
| CON a n
| VAR s n
| INT n
| PRI a $\otimes$
| ARG s n
| REG n



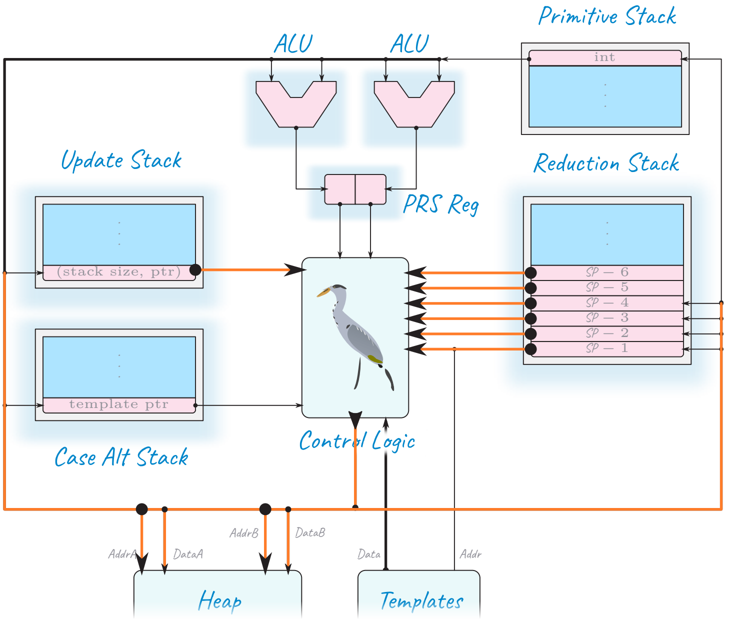
Postfix prims for long spines

$(f\ x\ y) + (g\ z)$

$\Rightarrow f\ x\ y\ g\ z\ +$



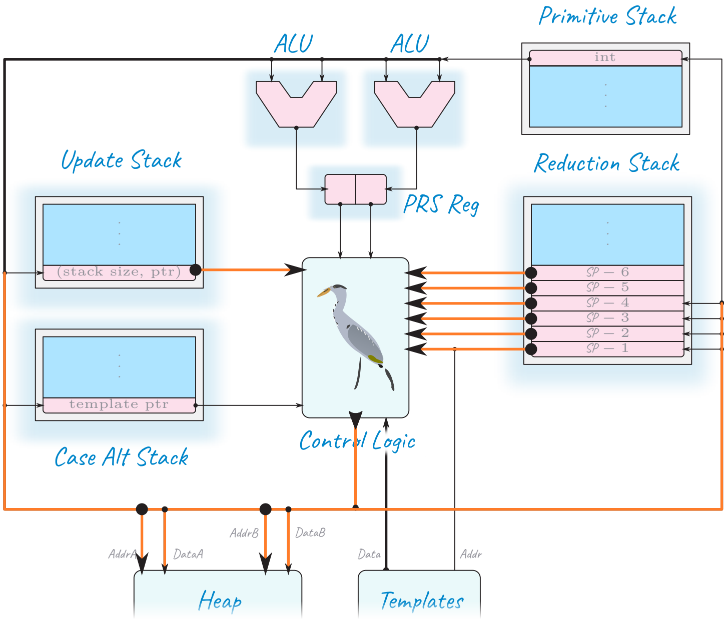
...But what about heap updates?



Avoid most updates via run-time sharing analysis!

Atoms

= FUN s a n
| CON a n
| VAR s n
| INT n
| PRI a ⊗
| ARG s n
| REG n

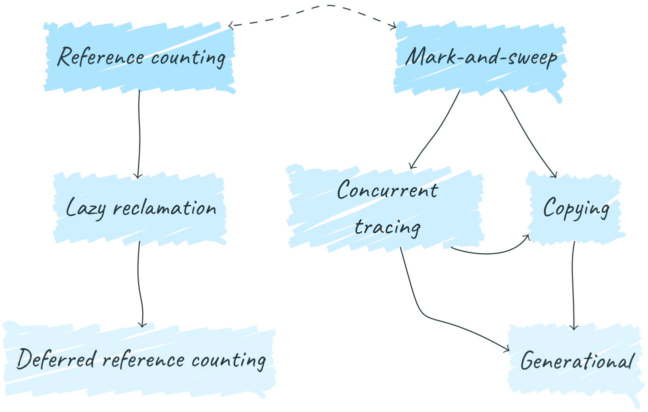


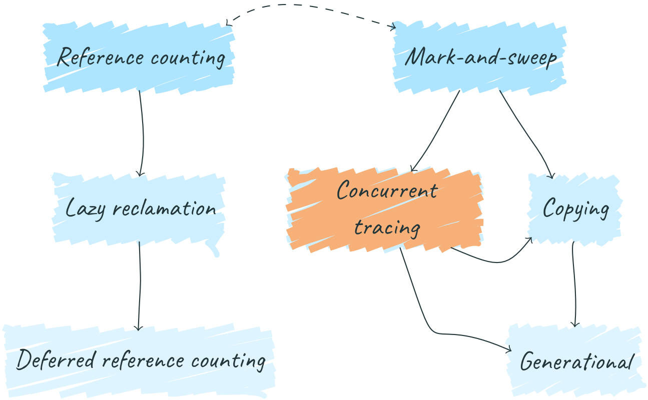
Avoid most updates via run-time sharing analysis!

Atoms

= FUN s a n
| CON a n
| VAR s n
| INT n
| PRI a ⊗
| ARG s n
| REG n

≈ **One-bit reference counting!**


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

Software for Concurrent GC

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

Software for Concurrent GC

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"

Software for Concurrent GC

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

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

Software for Concurrent GC

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

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

Software for Concurrent GC

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

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

Software for Concurrent GC

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

Key Observation:

Stock CPUs sequentialise write-barriers (trades-off GC latency for throughput)

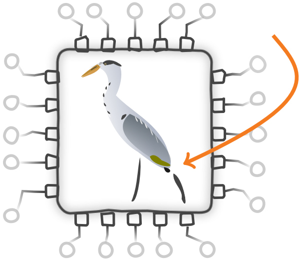
Custom hardware + read-first memories grants us both

Cloaca

noun [C]

/kloh-ah-kuh/

A concurrent hardware **garbage collector** for Heron



[0] Ramsay and Stewart, "Cloaca: A Concurrent Hardware Garbage Collector for Non-strict Functional Languages".

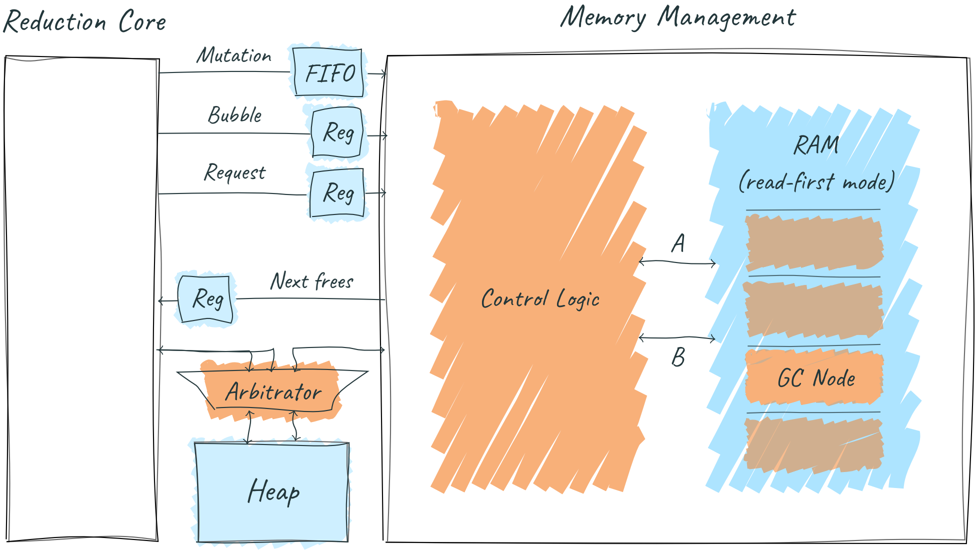
Cloaca

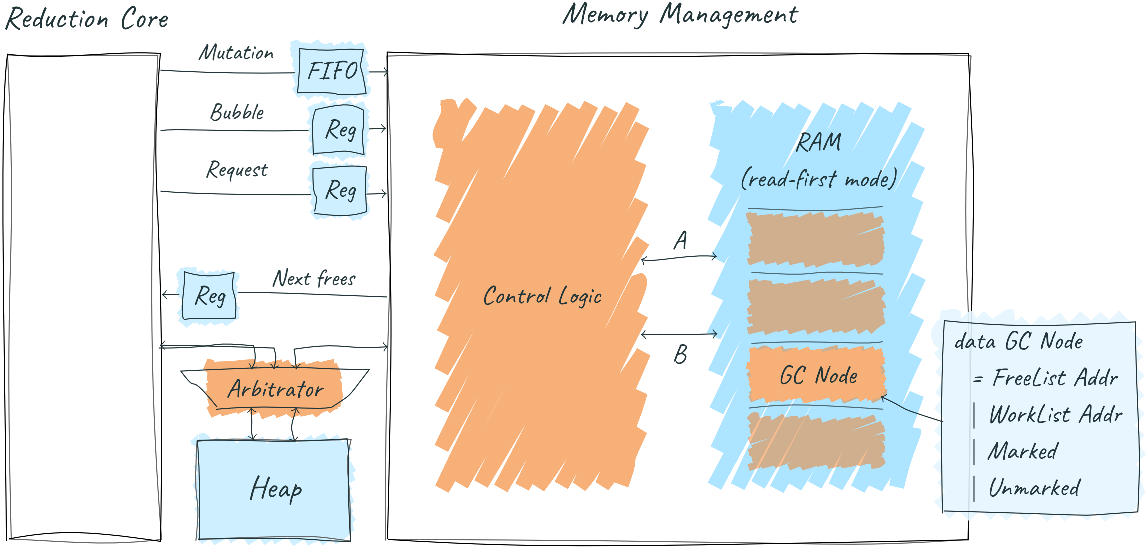
noun [C]

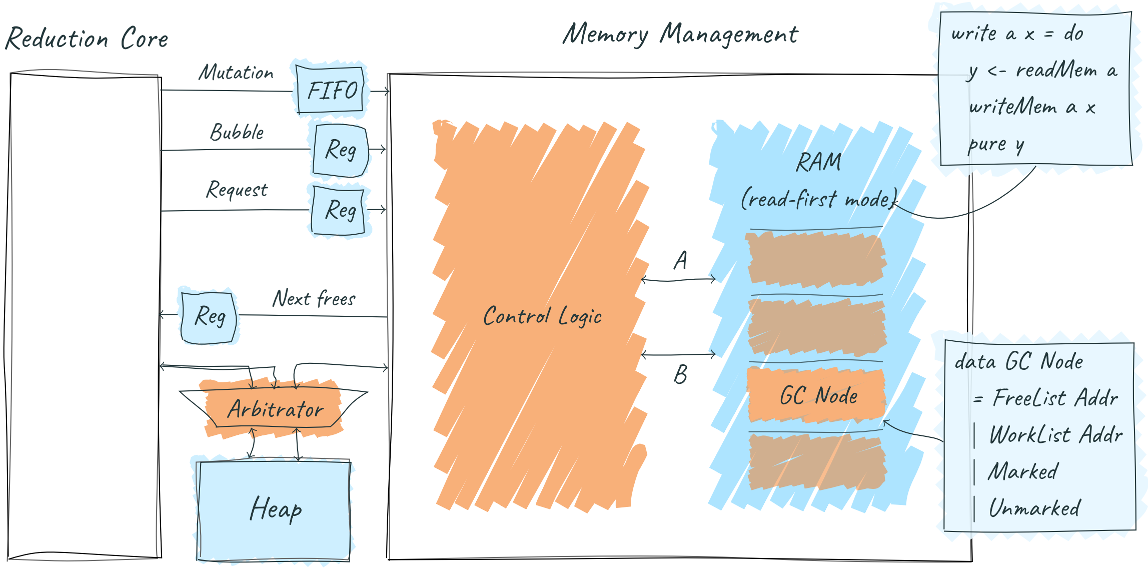
/kloh-ah-kuh/

A concurrent hardware **garbage collector** for Heron

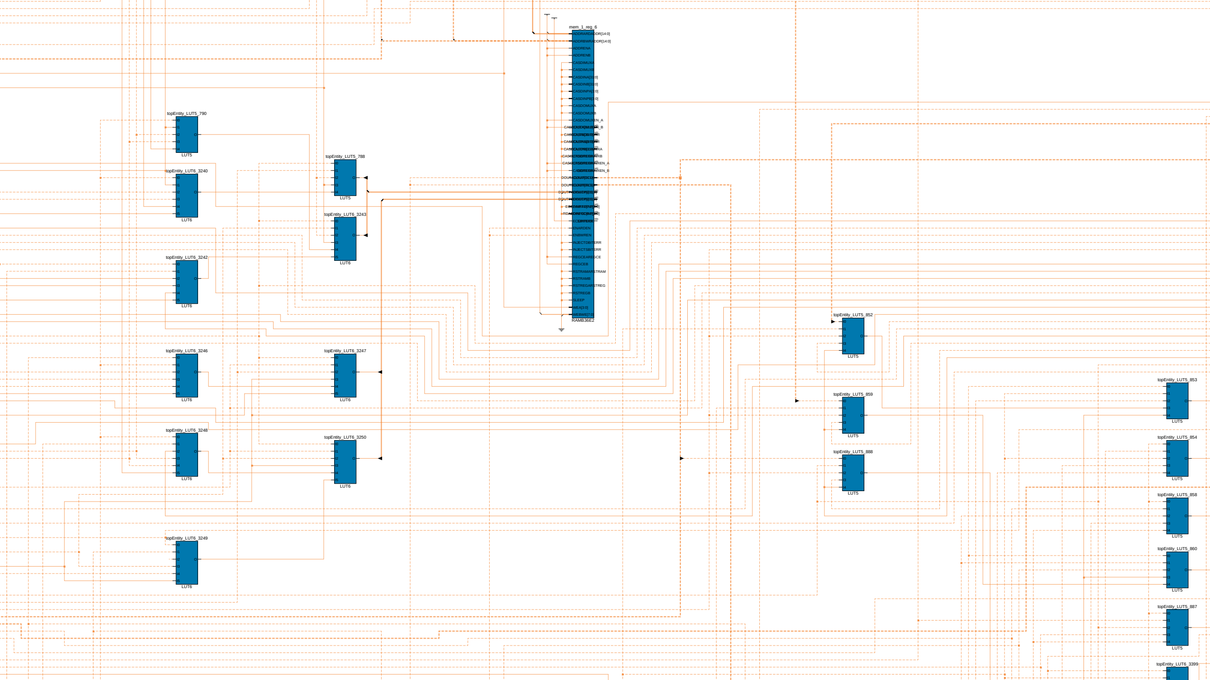
[0] Ramsay and Stewart, "Cloaca: A Concurrent Hardware Garbage Collector for Non-strict Functional Languages".


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


Reduction Core
Memory Management
Mutation
FIFO
Bubble
Reg
Request
Reg
Next frees
Reg
Arbitrator
Heap
Control Logic
A
B
RAM
(read-first mode)
GC Node
data GC Node
= FreeList Addr
| WorkList Addr
| Marked
| Unmarked


Reduction Core
Memory Management
write a x = do
y <- readMem a
writeMem a x
pure y
Mutation
FIFO
Bubble
Reg
Request
Reg
Reg
Next frees
Arbitrator
Heap
Control Logic
RAM
(read-first mode)
A
B
GC Node
data GC Node
= FreeList Addr
| WorkList Addr
| Marked
| Unmarked

Results


mem_1_reg_6
ADDRARDADDR[14:0]
ADDRBWRADDR[14:0]
ADDRENA
ADDRENB
CASDOMUXEN_A
ENARDEN
ENBWREN
INJECTDBITERR
INJECTSBITERR
REGCEAREGCE
REGCEB
RSTRAMARSTRAM
RSTRAMB
RSTREGARSTREG
RSTREGB
SLEEP
WEA[3:0]
WEBWE[7:0]
RAMB36E2
topEntity_LUT5_790
LUT5
topEntity_LUT6_3240
LUT6
topEntity_LUT5_788
LUT5
topEntity_LUT6_3243
LUT6
topEntity_LUT6_3242
LUT6
topEntity_LUT6_3246
LUT6
topEntity_LUT6_3247
LUT6
topEntity_LUT6_3248
LUT6
topEntity_LUT6_3250
LUT6
topEntity_LUT6_3249
LUT6
topEntity_LUT5_852
LUT5
topEntity_LUT5_859
LUT5
topEntity_LUT5_888
LUT5
topEntity_LUT5_853
LUT5
topEntity_LUT5_854
LUT5
topEntity_LUT5_858
LUT5
topEntity_LUT5_860
LUT5
topEntity_LUT5_887
LUT5
topEntity_LUT6_3399

Which Architectures?

	Heron	GHC + Intel i7 1250U	
Platform	Xilinx Alveo U280	Performance	Power-saver

Which Architectures?

	Heron	GHC + Intel i7 1250U	
Platform	Xilinx Alveo U280	Performance	Power-saver
Clock	185 MHz	4.7 GHz	$\approx$ 2 GHz

Which Architectures?

	Heron	GHC + Intel i7 1250U	
Platform	Xilinx Alveo U280	Performance	Power-saver
Clock	185 MHz	4.7 GHz	≈ 2 GHz
Power est.	0.8W dynamic + 3.1 W Static[1]	Cores 15 W or Package 16 W	Cores 2W or Package 6 W

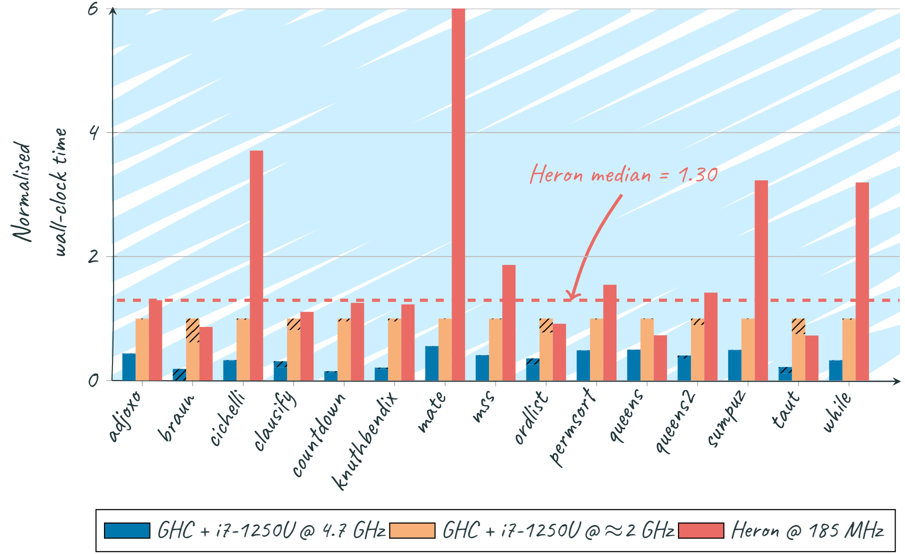
[1] Heron only occupies 1.13% of any resource type though!

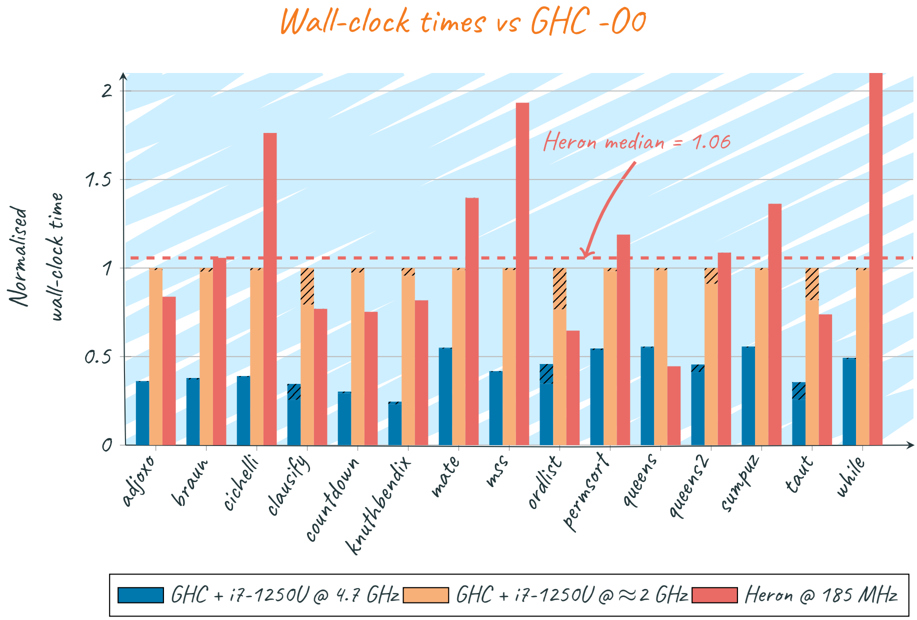
Which Architectures?

	Heron	GHC + Intel i7 1250U	
Platform	Xilinx Alveo U280	Performance	Power-saver
Clock	185 MHz	4.7 GHz	$\approx$ 2 GHz
Power est.	0.8W dynamic + 3.1 W Static[1]	Cores 15 W or Package 16 W	Cores 2W or Package 6 W
Fabrication	16 nm (FPGA!)	10 nm	10 nm

[1] Heron only occupies 1.13% of any resource type though!

Wall-clock times vs GHC -O2





Wall-clock times vs GHC -O0
Normalised wall-clock time
2
1.5
1
0.5
0
Heron median = 1.06
adjoxo
braun
cichelli
clausify
countdown
knuthbendix
mate
mss
ordlist
permsort
queens
queens2
sumpuz
taut
while
GHC + i7-1250U @ 4.7 GHz
GHC + i7-1250U @ ≈2 GHz
Heron @ 185 MHz

Conclusion

A tiny template instantiation core nearly keeps up with a modern CPU running at x10 speed!

Conclusion

A tiny template instantiation core nearly keeps up with a modern CPU running at x10 speed!

Assisted by a fully concurrent GC (modulo $\approx$20 cycles per pass).

Conclusion

A tiny template instantiation core nearly keeps up with a modern CPU running at x10 speed!

Assisted by a fully concurrent GC (modulo $\approx$ 20 cycles per pass).

Lays a path towards a single-chip multi-core architecture.

Conclusion

A tiny template instantiation core nearly keeps up with a modern CPU running at x10 speed!

Assisted by a fully concurrent GC (modulo $\approx$20 cycles per pass).

Lays a path towards a single-chip multi-core architecture.

We want to see more research towards custom FP architectures!

Questions?