Auklet — arithmetic

Bluebird — booleans binary representation

Cardinal — calling conventions: printing, runtime checks

Dove — function declarations / function calls

Eagle

Hawk

Pairs + Dove

Things we don't have yet

Strings
floats
imperative programming (loop)
global vars
classes / structs
modularity
lists/arrays
pointers/references

Structured data & indirection

① Syntax

```
<expr> ::= ... dove exprs
         | (<expr>, <expr>)
         | fst (<expr>)
         | snd (<expr>)
```

| EPair (e1, e2) →

② Semantics

(4, 5) ⟹ (4,5)

(2+3, 7-1) ⟹ (5,6)

fst (2, 3) ⟹ 2

fst (3+7, 1<2) ⟹ 10

snd (1+2, 3+false) ⟹ runtime error

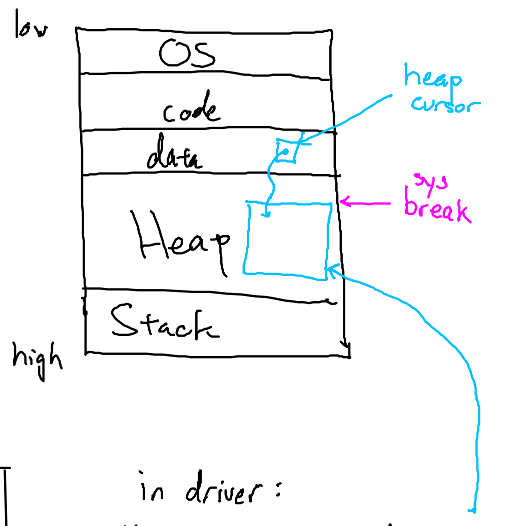
fst (1+2, 3+false) ⟹ runtime error

fst 2 ⟹ runtime error

(1, (2, 3)) ⟹ (1, (2, 3))

let p= (1,2) in (p,p) ⟹ ((1,2),(1,2))

let x= if b then 4 else (5,6) in x ⟹ .... (based on value b)



```
.section data
  align 8
  heap_cursor:
    dq 0

mov rax, 5
mov [heap_cursor], rax
```

db	"byte"	8 bits
dw	"word"	16 bits
dd	"double word"	32 bits
dq	"quad word"	64 bits

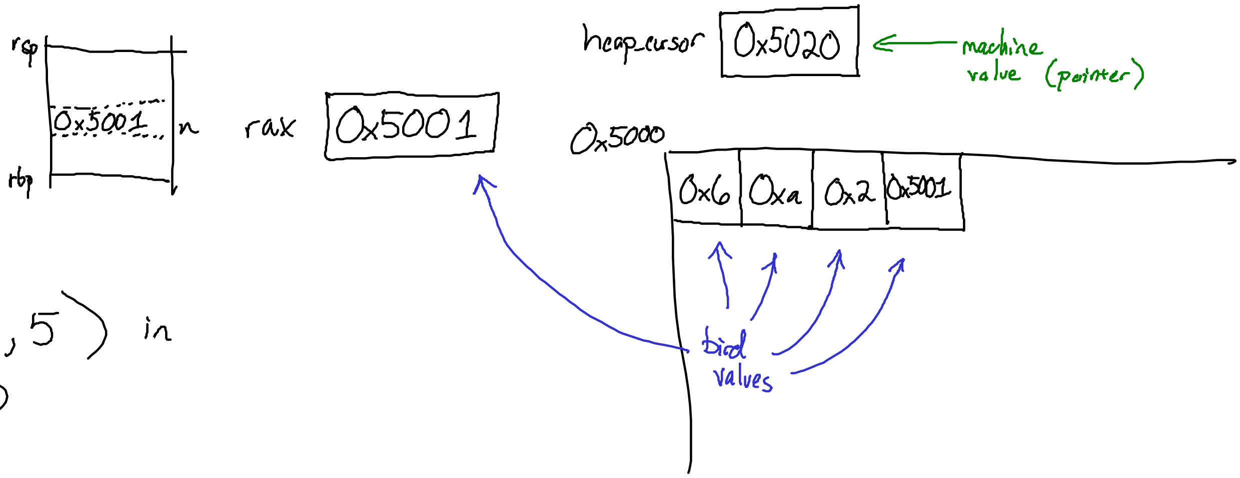
in driver:

malloc ~4MB for "heap"

not doing

- freeing memory
- growing heap

let n = (3, 5) in
(1, n)



Hawk (Eagle) binary representation

```
0xNNNNNNNNNNNNNNN[nnn0] : integer 0xNNNNNNNNNNNNNNNnnn
0x FFFF ------------- FF [1111] : true
0x 7FFF ------------- FF[1111] : false
0x NNNN ------------- NN[n001] : pointer 0x NNNN ------- NN[n000]
```