CMSC 430: Introduction to Compilers

Hustle: heaps and lists

Hustle

- Hustle adds add two **inductively defined data types**, boxes and pairs.

```
(cons v1 v2)

(box v)
```

- So far, all the data we have considered can fit in a single machine word (64-bits).
- Hustle will require us to relax this restriction.

Hustle: new operations and values

- cons
- car
- cdr
- cons?
- box
- unbox
- empty?
- '()

Encoding values in Hustle

		Immediate tag	
60-bits for number	0	0 0 0	Integers
59-bits for code point (only need 21)	0 1	0 0 0	Characters
	0 1 1	0 0 0	`#t`
	1 1 1	0 0 0	`#f`
	1 0 1 1	0 0 0	`eof`
	1 1 1 1	0 0 0	`void`
	1 0 0 1 1	0 0 0	

61-bits for address	0 0 1	Box
61-bits for address	0 1 0	Cons

Bit layout of values

- Values are either:
  - Immediates: end in #b000
  - Pointers: (non-zero in last 3 bits). Now Boxes and pairs
- Immediates are either
  - Integers: end in #b 0 000
  - Characters: end in #b 01 000
  - true: #b 11 000
  - false: #b1 11 000
  - eof: #b10 11 000
  - void: #b11 11 000
  - empty: #b100 11 000

Interp: New Unary Operators

Hustle adds the operators: box unbox car cdr empty? cons?

```
(define (interp-prim1 op v)
  (match (list op v)
    [(list 'box v)              (box v)]
    [(list 'unbox (? box?))     (unbox v)]
    [(list 'car (? pair?))      (car v)]
    [(list 'cdr (? pair?))      (cdr v)]
    [(list 'empty? v)           (empty? v)]
    [(list 'cons? v)            (cons? v)]
    [_ 'err]))
```

Interp: New Binary Operators

Hustle adds the binary operators: cons eq?

```
(define (interp-prim2 op v1 v2)
  (match (list op v)
    [(list 'eq? v1 v2)    (eq? v1 v2)]
    [(list 'cons v1 v2)   (cons v1 v2)]
    [_ 'err]))
```

Various facts about the Hustle compiler

- Registers:
  - `rax` - return value
  - `rsp` - stack pointer
  - `rdi` - first param when calling run-time system
  - `rbx` – heap pointer
- Stack is 8-byte (64-bit) aligned
  - i.e. divisible by 8, ends in `#b000`
  - (Must align to 16-bytes to call)
- Heap is 8-byte (64-bit) aligned
  - i.e. divisible by 8, ends in `#b000`

Runtime

main.c

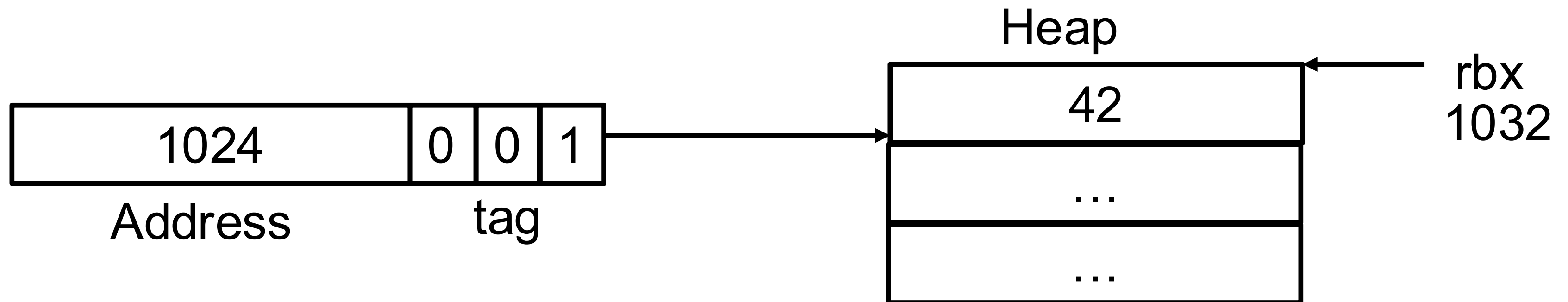
```
heap = malloc(8*heap_size);

val_t result;

result = entry(heap);

print_result(result);
```

Compile.rkt

```
(Push r15)    ;; callee-saved
(Push rbx)
(Mov rbx rdi)    ;; heap pointer
(compile-e e '())
(Pop rbx)    ;; restore
(Pop r15)
(Ret)
(Label 'err)
 pad-stack
(Call 'raise_error)))
```

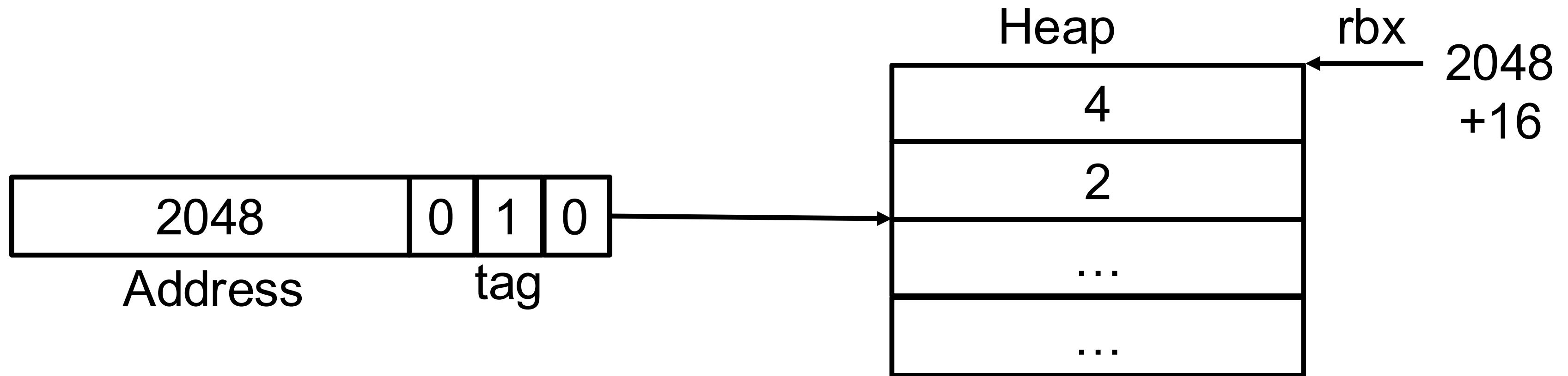
Box

(box 42) is represented as:



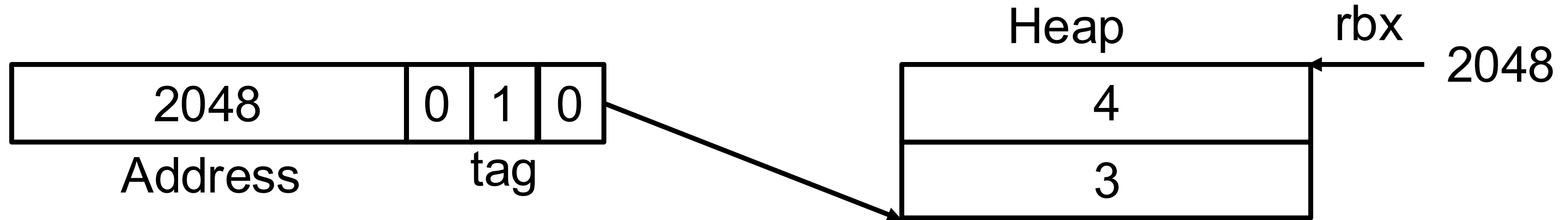
Cons

(Cons 3 4) is represented as:



Cons

(Cons 3 4) is represented as:

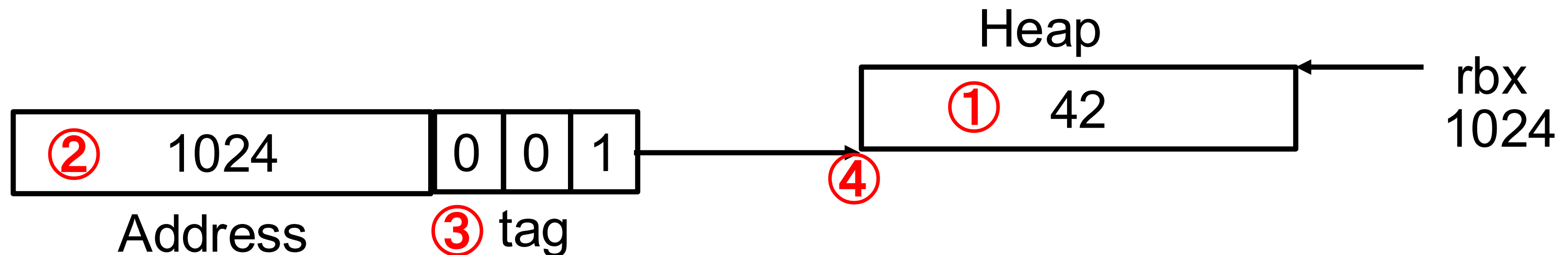


- A pair is allocated as two words in memory. The pair *value* will be represented by the address + type tag in 3 least significant bits
- Pointers always end in #b000 because we allocate in multiples of 8 bytes, so take advantage of this and stash type tag there (no shifting).

Box

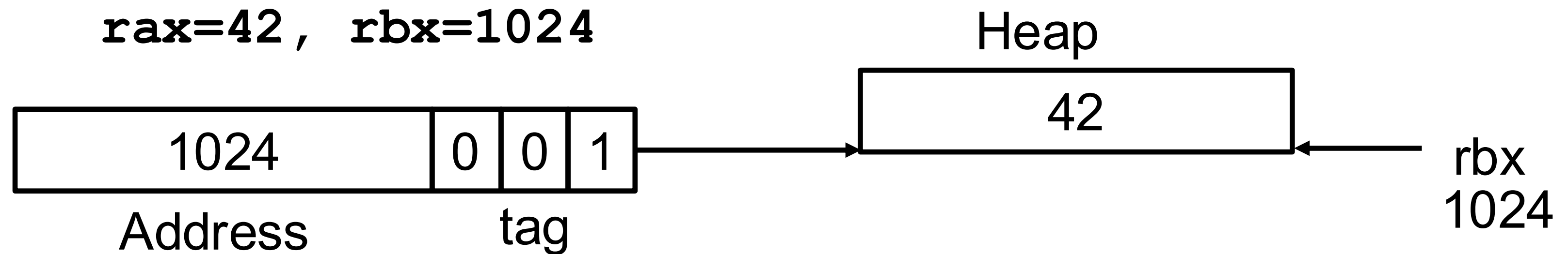
```
['box
 (seq
 ① (Mov (Offset rbx 0) rax) ; memory write
 ② (Mov rax rbx) ; put box in rax
 ③ (Or rax type-box) ; tag as a box
 ④ (Add rbx 8))]
```

`rax=42, rbx=1024`



Unbox

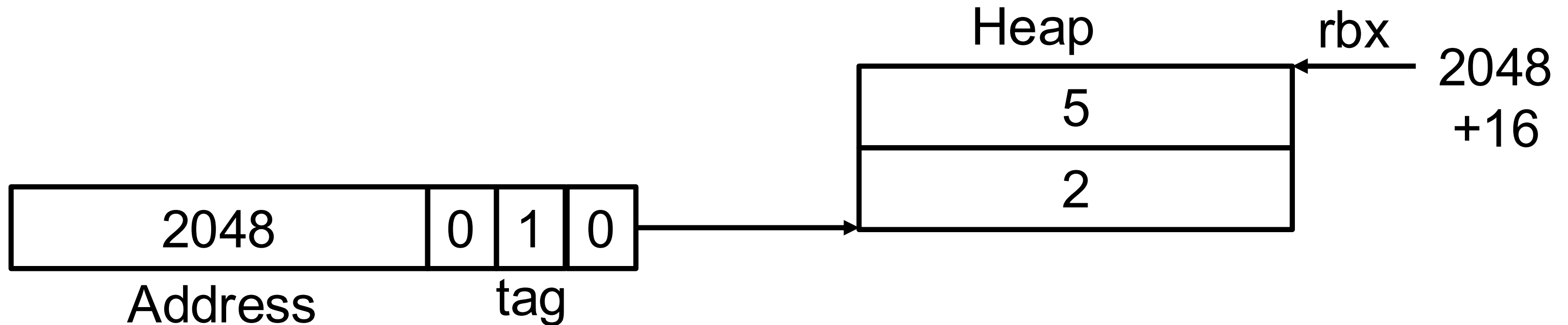
```
['unbox
(seq
  (assert-box rax)
  (Xor rax type-box)
  (Mov rax (Mem rax 0)))]
```



cons

```
['cons (seq
  (Mov (Mem rbx 8) rax);; 5
  (Pop rax);;2
  (Mov (Mem rbx 0) rax);;2
  (Mov rax rbx)
  (Or rax type-cons)
  (Add rbx 16))]
```

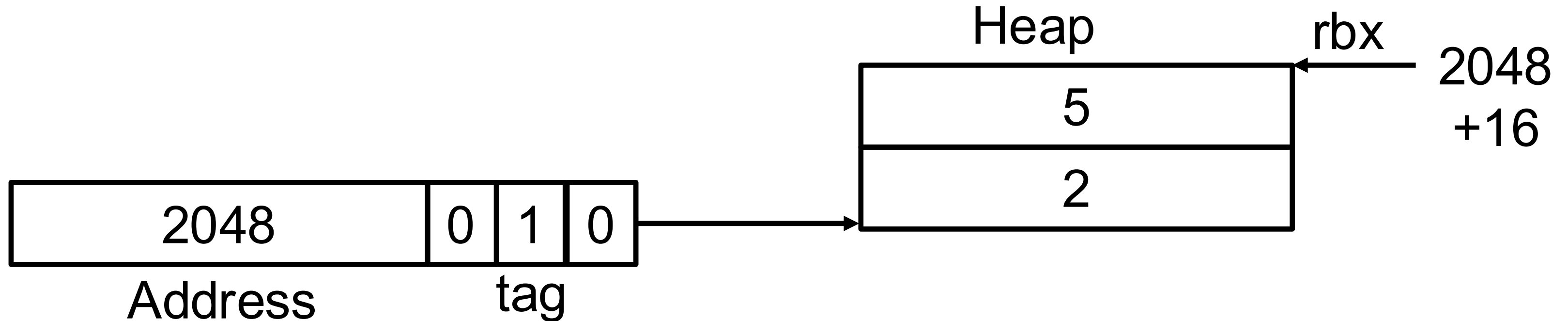
```
(cons 2 5)
rax=5, rbx=2048
Stack: 2
```



car

```
['car
(seq
 (assert-cons rax)
 (Xor rax type-cons)
 (Mov rax (Mem rax 0)))]
```

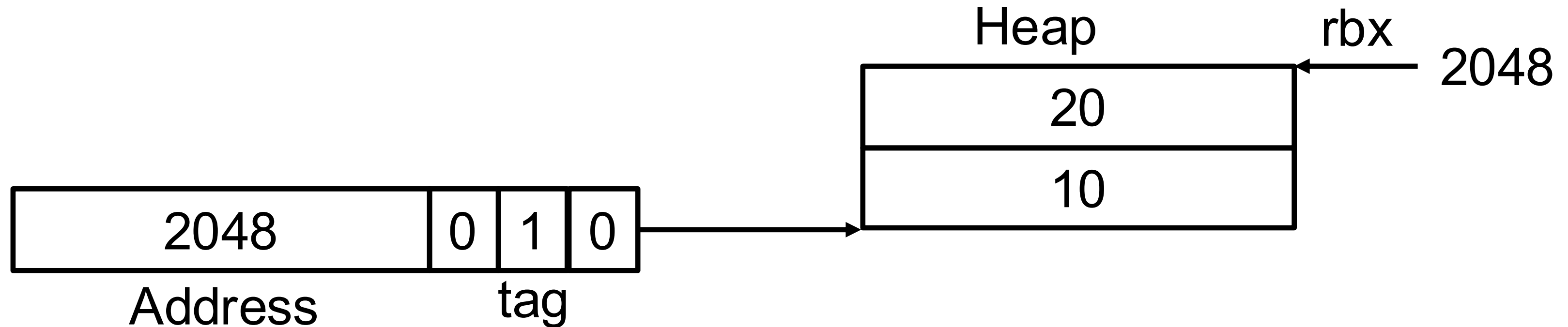
```
(cons 2 5)
rax=2048  010
rbx=2048+16
```



cdr

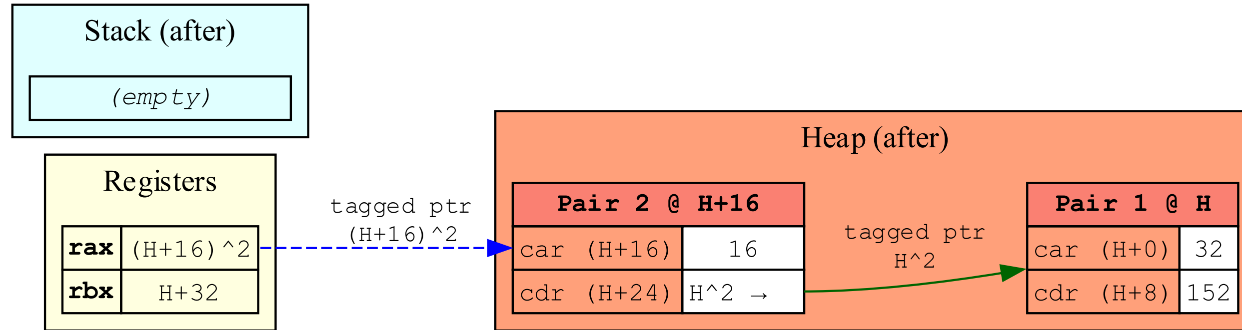
```
['cdr
  (seq
    (assert-cons rax)
    (Xor rax type-cons)
    (Mov rax (Mem rax 8)))]
```

```
(cons 10 20)
 rax=2048  010
 rbx=2048+16
```



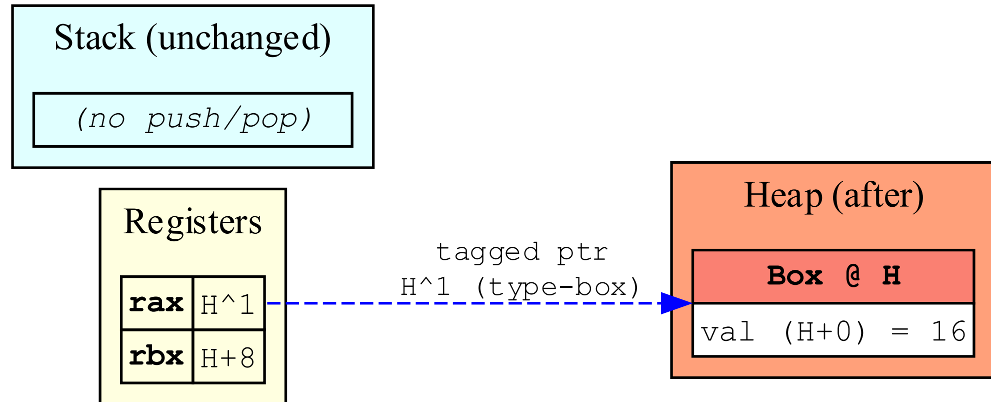
Example 1: (cons 1 (cons 2 '()))

```
(Mov 'rax 16)
(Push 'rax)
(Mov 'rax 32)
(Push 'rax)
(Mov 'rax 152)
(Mov (Mem 'rbx 8) 'rax)
(Pop 'rax)
(Mov (Mem 'rbx 0) 'rax)
(Mov 'rax 'rbx)
(Xor 'rax 2)
(Add 'rbx 16)
(Mov (Mem 'rbx 8) 'rax)
(Pop 'rax)
(Mov (Mem 'rbx 0) 'rax)
(Mov 'rax 'rbx)
(Xor 'rax 2)
(Add 'rbx 16)
```



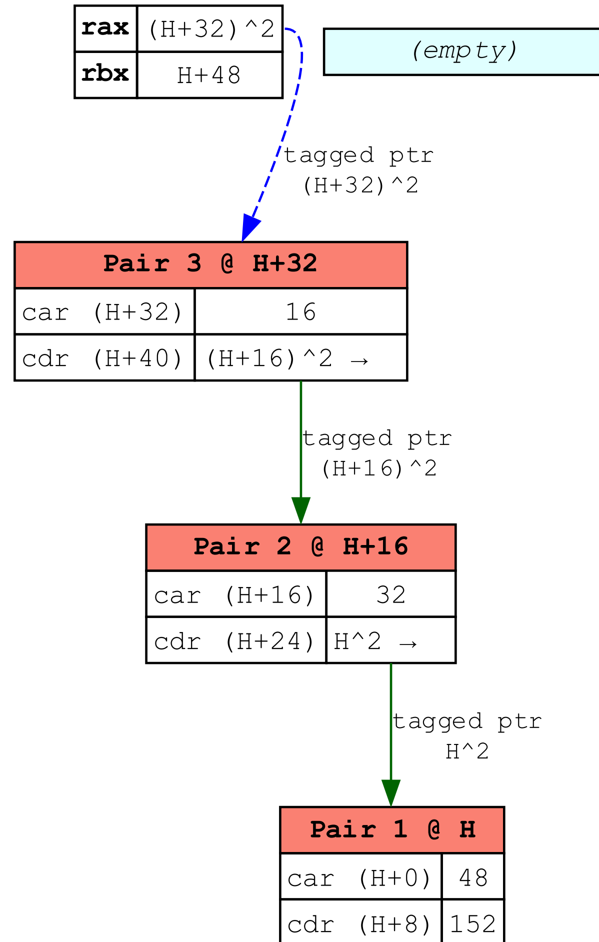
Example 2: (box 1)

```
(Mov 'rax 16)
(Mov (Mem 'rbx) 'rax)
(Mov 'rax 'rbx)
(Xor 'rax 1)
(Add 'rbx 8)
>
```



Example 3: (cons 1 (cons 2 (cons 3 ‘())))

```
(Mov 'rax 16)
(Push 'rax)
(Mov 'rax 32)
(Push 'rax)
(Mov 'rax 48)
(Push 'rax)
(Mov 'rax 152)
(Mov (Mem 'rbx 8) 'rax)
(Pop 'rax)
(Mov (Mem 'rbx 0) 'rax)
(Mov 'rax 'rbx)
(Xor 'rax 2)
(Add 'rbx 16)
(Mov (Mem 'rbx 8) 'rax)
(Pop 'rax)
(Mov (Mem 'rbx 0) 'rax)
(Mov 'rax 'rbx)
(Xor 'rax 2)
(Add 'rbx 16)
(Mov (Mem 'rbx 8) 'rax)
(Pop 'rax)
(Mov (Mem 'rbx 0) 'rax)
(Mov 'rax 'rbx)
(Xor 'rax 2)
(Add 'rbx 16)
```



Example 4: (cons 1 (cons 2 3))

```
(Mov 'rax 16)
(Push 'rax)
(Mov 'rax 32)
(Push 'rax)
(Mov 'rax 48)
(Mov (Mem 'rbx 8)
'rax)
(Pop 'rax)
(Mov (Mem 'rbx 0)
'rax)
(Mov 'rax 'rbx)
(Xor 'rax 2)
(Add 'rbx 16)
(Mov (Mem 'rbx 8)
'rax)
(Pop 'rax)
(Mov (Mem 'rbx 0)
'rax)
(Mov 'rax 'rbx)
(Xor 'rax 2)
(Add 'rbx 16)
```