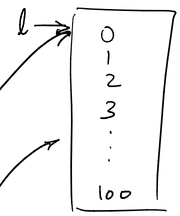
Lecture 18

Thursday, October 22, 2020 3:59 PM

Memory management

```
let f n =
  let l = [0; 1; 2; ...; n] in
  fun i -> List.nth l i        <- Closure

let g = f 100
```

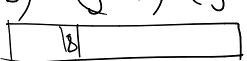
g i = List.nth l i




```
g 3;;    g 4;;
```

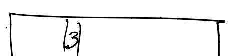
```
let f n =
  let l = [ref 0; ref 1; ... ; ref n] in
  fun i -> nth l i

let g = f 10          let h = f 10
(g 3) (g 4) (g 8)
```

l_g =



g 3

l_h =



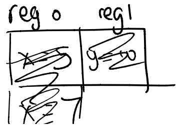
h 3

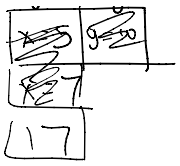
```
(g 3) := 8
```

```
let x = 5 in
  ...
```

```
let x = 5 in        <-
let y = x + 5 in    <-
let x = x + 2 in    <-
x + y
```

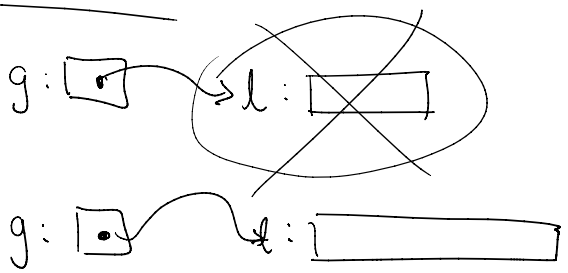
Question: What values are "escaping" this computation?





→ let g = f 10

→ let g = f 100



Stack / Heap

let (x) = 5 in

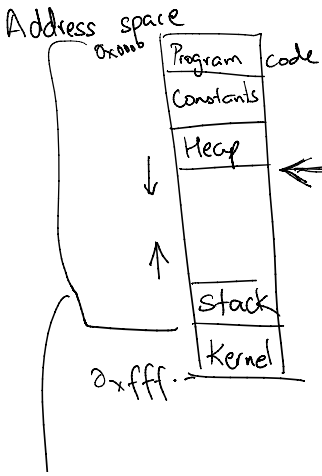
x + 3

- Variable does not escape
- So, it has a limited lifetime
- Allocate it on the (stack).

let g = f 100

- l_g escapes
- Its lifetime is not known "a priori".
- Allocated on the (heap).

Global Interpreter lock.



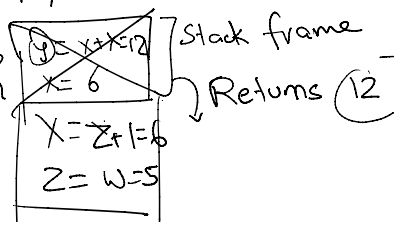
let g x = let y = x+x in y

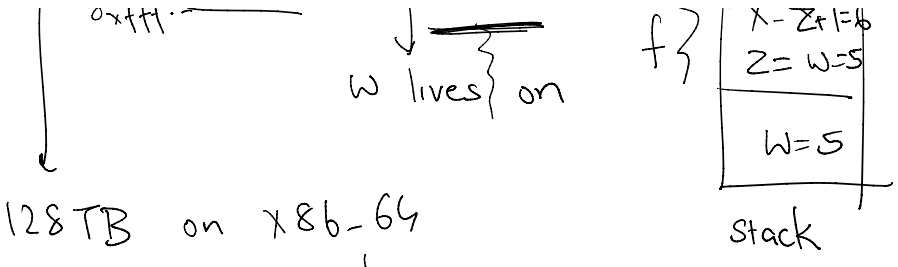
let f z = let x = (z+1) in g(x)+g(x)

let w = 5 in

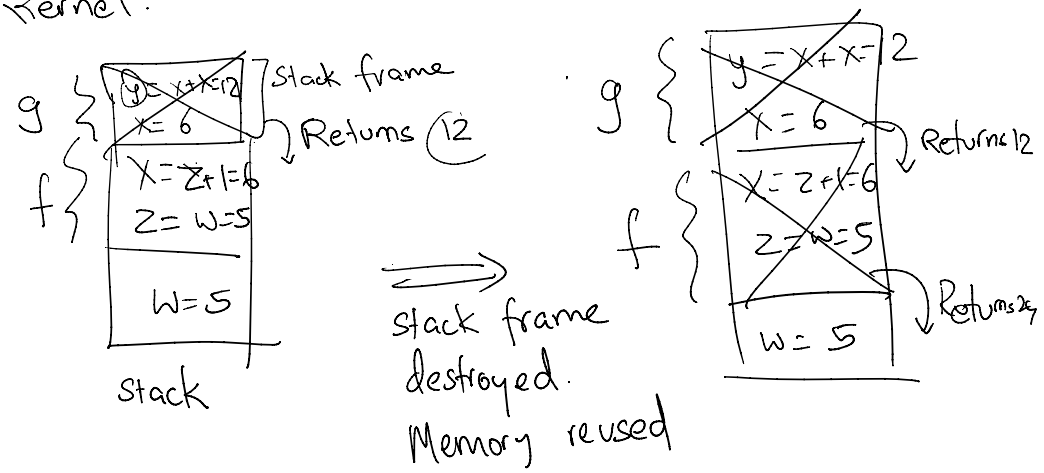
let r = f(w)

w lives on





128TB on x86_64 Linux kernel.



r = 24
w = 5

Memory management in C:

`void *malloc(size_t size)` / OOM

~~calloc~~ ~~realloc~~

Allocates "size" bytes & returns a pointer to them

`free(void *ptr)` ← Frees a pointer previously returned by malloc.

Glibc / jemalloc / dlmalloc / ... / openBSD

↓↑

Calls provided by the kernel

System calls provided by the kernel.

brk

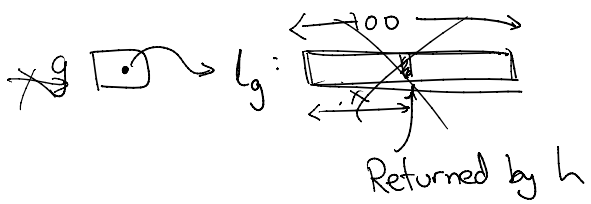
sbrk

mmap

munmap

Garbage Collection

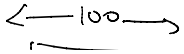
```
let f n =
    let l = [ref 1 ; ref 2 ; ... ; ref n] in
    fun i -> nth l i.

let h x =
    let g = f 100 in
    !(g x)
```



h 5

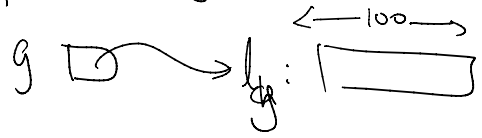
At this point, nobody can access g any more.
Therefore, both g & l_g can be freed.

(h, in)



(h 10)

Therefore, both g & l_g can be freed.



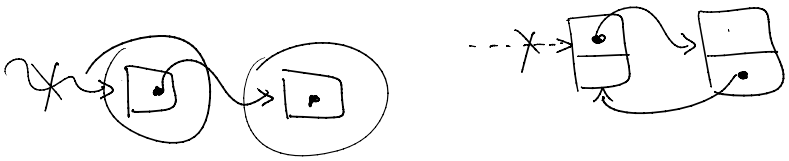
Question: Under what conditions can an object be freed?

Proposal 1: If nothing points to it.

Repeat until exhaustion.

Q1: Is it safe?

Q2: Is it complete?



Reference counting.

Proposal 2: Mark-&-Sweep Garbage Collector.

- Follow all pointers coming out of the stack.
- Do a DFS, mark all heap locations that you visit.
- Make a pass over the heap deallocate all cells which

have not been marked.

GC-pause Garbage collection pause	GC-pressure.

Rust has no GC.

Pauseless GC. vs. stop-the-world.

Epsilon GC in new JVMs.