Louden

5 Basic Semantics (1)

Basic Semantics
- experiment
- Tutorial.

semantics :
- LRM (ISO/ANSI)
- translator (reference impl)
- formal defn

5.1 • Attributes & Bindings

von Neumann.

- name → location → value
- attributes (type, lval, const, ...)
- const may not have location

~~assoc (name, attr) ≜ binding~~

- binding ≜ assoc (name, attrib)

```
ex: OCaml:  let x = 2;;
      let p = (let y = 3 in (x + y))
        x → 2 global
        y → 3 local to let-in.
```

- binding time : static or dynamic

```
const int n = 2;       } static
int n = 2;
int *n = new int(2);   } dynamic.
if local name → loc dynamic (static?)
```

- binding time
  - 3. translation time ← { 1. lang def time; 2. lang impl time }
  - 4. link time
  - 5. exec (load) time
  - 6. run time

1..5 = static
6 = dynamic

- symtab maintains bindings. (static) } compiler.
- ptrs " dyn " } compiler.

Loudon
5 Basic Semantics (2)

~~interpreter~~
~~Environ~~

compiler :
translator : names $\xrightarrow{symtab}$ static attr

interpreter : names $\xrightarrow{environments}$ locations

~~mae~~ locations $\xrightarrow{memory}$ Values.

5.2 • Decls & Scope

decl - estab bindings.

int x; — x → int explicit
x → loc. implicit (static or dyn)
x → value (implicit or undef)
(default value?)

defn vs prototype (partial defn)

struct *x; — incomplete type.

decl : assoc w/ block or struct
local-nested-global
member

block structure

large groups: packages, modules, namespaces
superstructures
explicit visibility.

lexical scope = static
dynamic scope.

incomplete type

scope resolution f::g

Loudon
5 Basic Semantics (3)

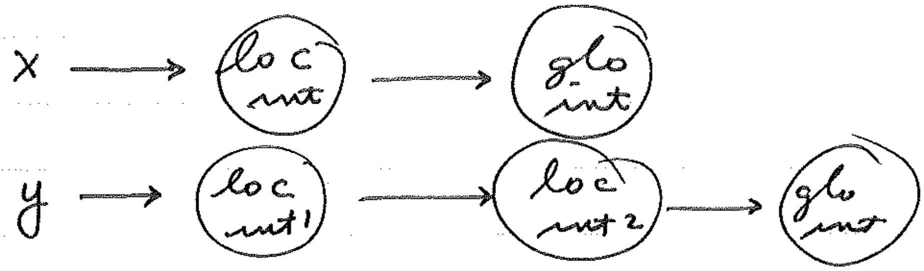
scope - global
nonlocal
local

block structure establishes visibility.

visibility can be hidden
override selection   ::v global

5.3 • Symbol Table
-maintains info about bindings
names → attributes
static - compile time
dynamic - runtime

map



```
global x,y; x=1
            y=2
p() {
  loc x=3
  {
    loc y=4      main() {
  }                loc x=6
}                  Q();
Q() {
  loc y=5   }
  P();
}
```

trace code
static
dynamic.

program

```
glo x = 1
glo y = 2
 P() { Ⓖ
   loc x = 3
       { Ⓗ
         loc y = 4
         Ⓘ
       }
     Ⓙ
 }
 Q() { Ⓓ
     loc y = 5 Ⓔ
     P(); Ⓕ
 }
 main() {
     Ⓐ
     loc x = 6 Ⓑ
     Q() Ⓒ
 }
```

trace prog
symtab

static scope
dynamic scope

x ⟶ symtab
y ⟶ symtab.

struct decl ⟶ attrib local symbol table
must visible while struct viz.

special case lookup
a.f
↑ know the type of a.

multiple nested scopes

5.4 Names & Overloading

overloading - multiple objects same name
- NOT an OO concept! \ override

x
this.x
f::x
::x

3+4 → intadd
3.+.4. → float add.

overload resolution

int max (int, int);
~~real~~ max (real, real); → what about max (3, 4.6)?

overload resol:
- count params, elim wrong count
- match types
- one left = OK
- zero left = error
- many left = ambiguous.
  ⇒ try again with conv? or error.

C++ — 5 levels of retry.

Ada — no conv.

---

but Ada uses context too
f(max (3,4)) — must match 3, 4 but also result

↑↓↑↓

Loudon

5 BasicSemantics

(6)

operator overload
– not different.
in AST no diff between
op & fn .

Ada   `y = m*x + b;`
       `y = "+"("*"(m,x), b);`

Ocaml   `let y = m *. x +. b;;`
         `let y = (+.)((*.) m x) b ;;`
         `let f = (+.) b ;;`
         ↑ but can Curry

no overload
can prefix

Operators:
matchfix   [ ]   [ ]=
c++   ( )   ->

C++   `y = operator + (operator * (m,x), b)`

Fortran   `MOD(a,b)`

Loudon
5 Basic Semantics (7)

5.5 Allocation & Lifetime

environment

maps: names → ~~attrs~~

locations $\xrightarrow{M}$ values

const = no location

string = immutable w/ loc.

block structure

globals — usually static

locals — dyn. alloc

— but static base offset

```
A: { int x
   B: { int y
        int z
      C: { int x
           int z
         }
   }
}
```

— overlaps?

hidden bindings.

— alloc on enter

— delete on exit

static
auto
heap

- activation records
- stack frame

alloc. in env = lifetime of obj

Loud on
5 Basic Semantics (8)

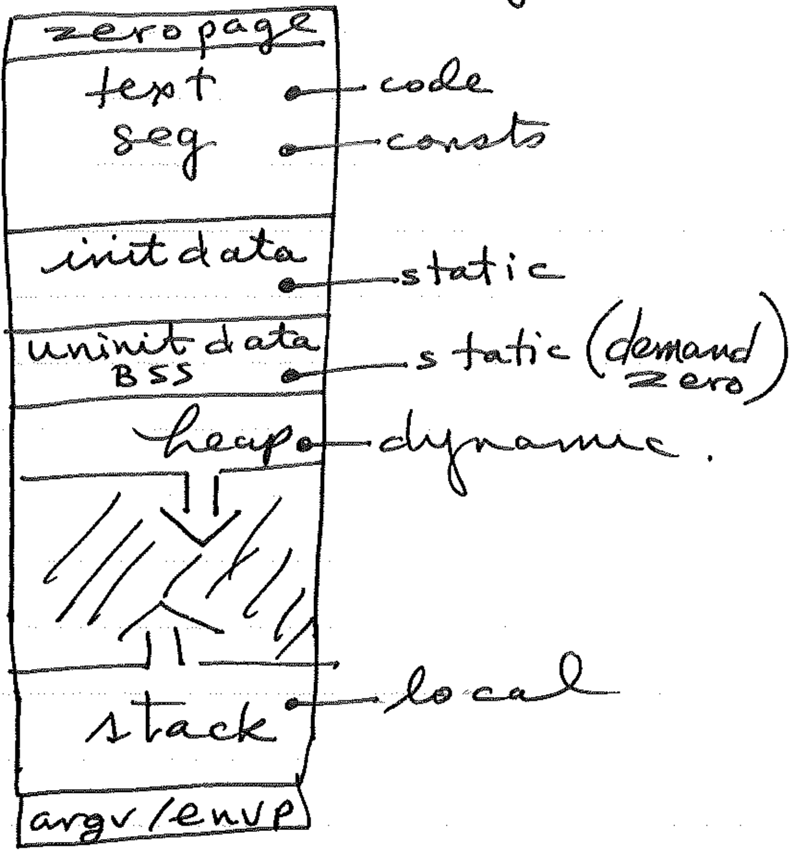
references

→ objects on heap
- indefinite lifetime. → dynamic alloc anonymous

malloc/new
delete/free

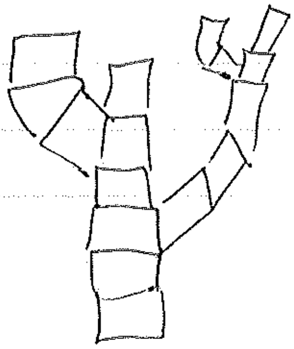
memory leak
dangling pointers — vs — g col

prog structure



auto (local)
static
dynamic (heap)

cactus stack

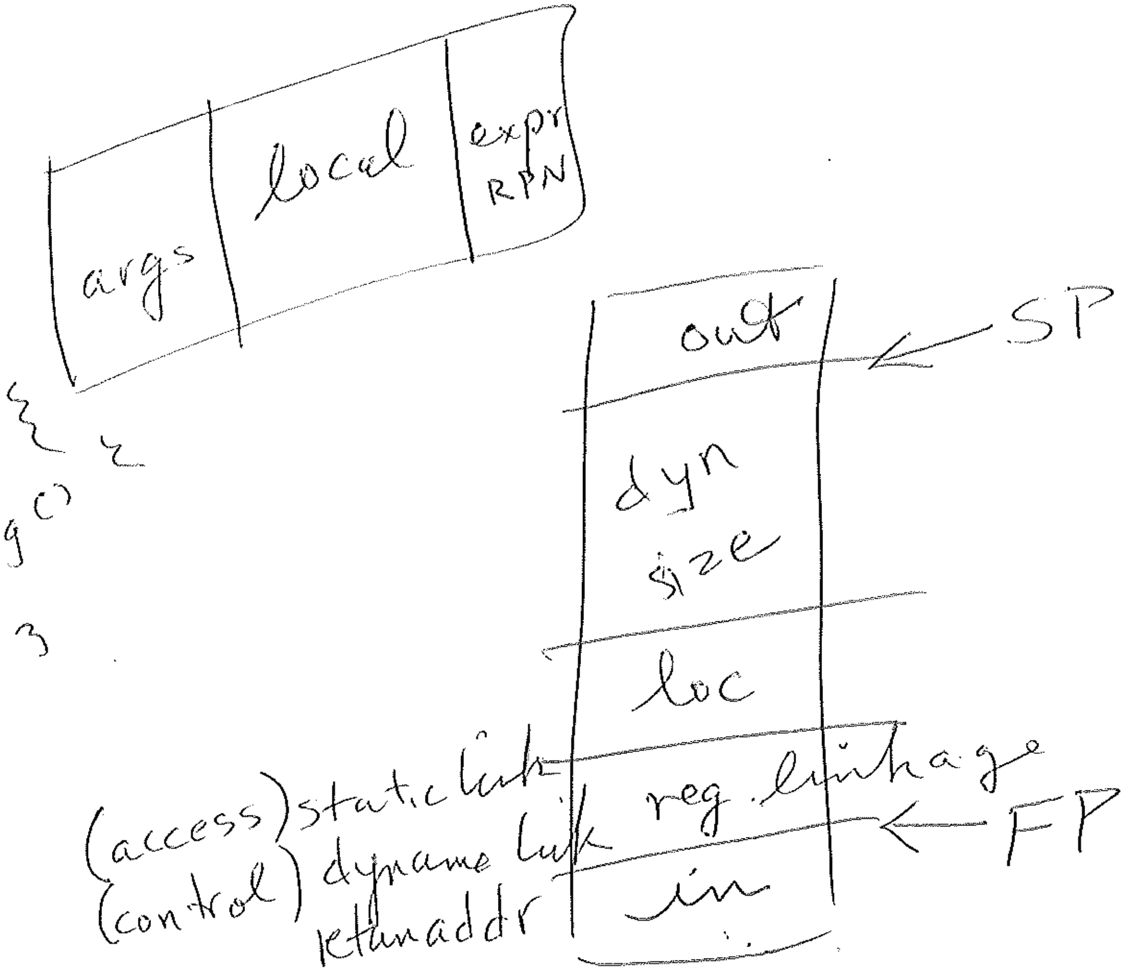


Basic Semantics

(7.1)

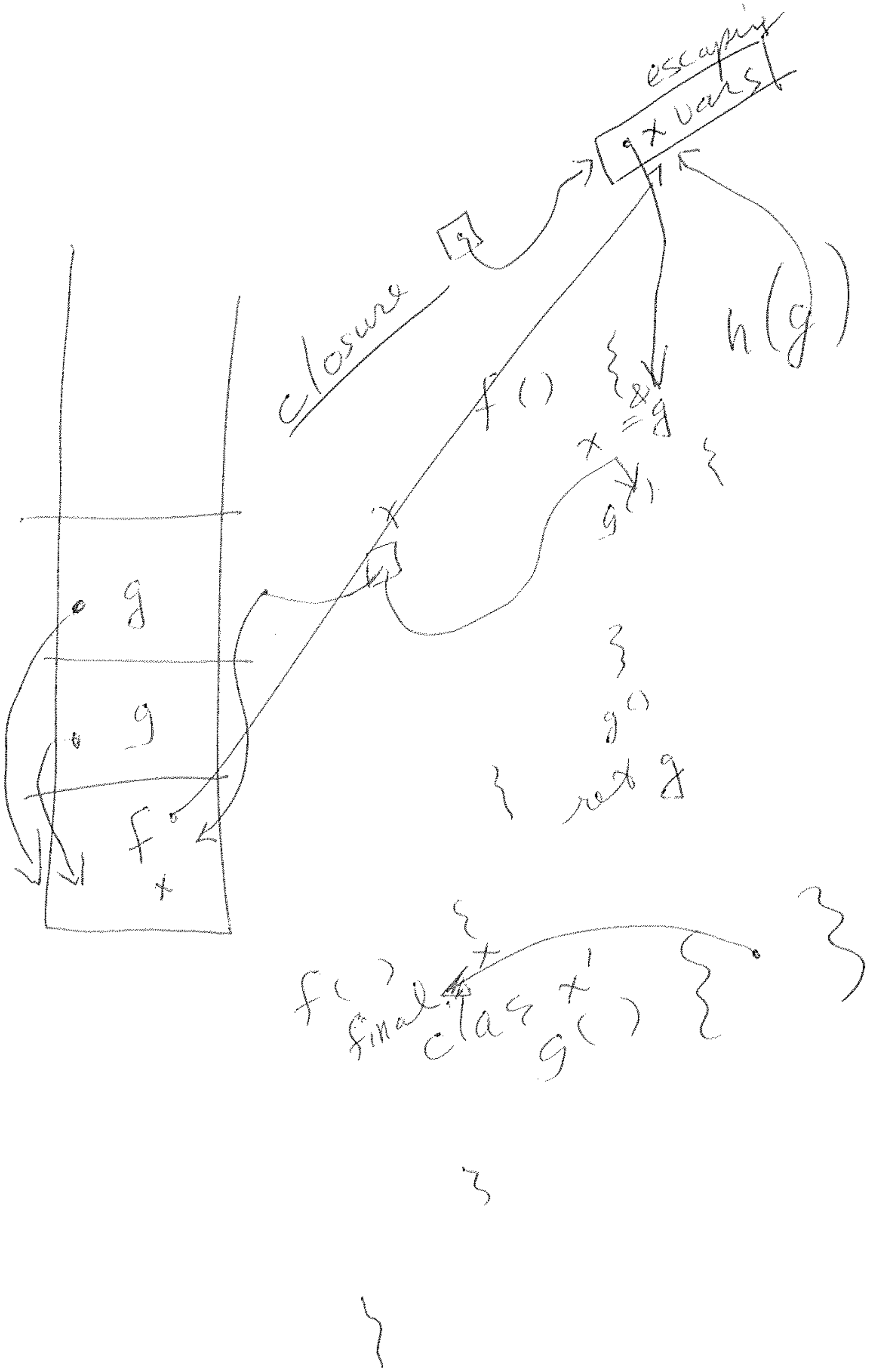
activation records
- static link
- dynamic link
- heap
- closures

cactus stack.



```
f() {
    g() {

    }
}
```


escaping
x vars
closure
h(g)
f() {
&g
x
g() {
}
g()
} ret g
g
g
f
x
{
f() final
x
class x'
g() {
}
}
}

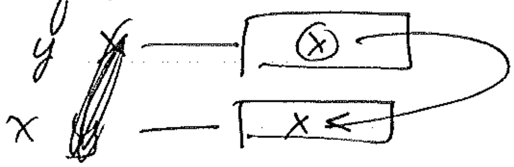
Loudon
5 Basic Semantics (9)

5.6 Variables & Constants

variable := obj whose value can change

name → location $\boxed{\text{value}}$

~~x=y~~ copy value of var
~~y=x~~ x=y



ML: increment x := !x + 1
copy x := !y

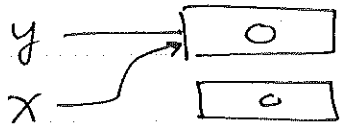
! dereference
:= copy

let x=3
let y=x+2
let x=6

How to Copy

mutable
immutable

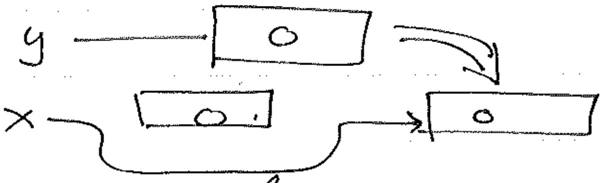
asgt by sharing
x=y



bind x address to what y refers to.

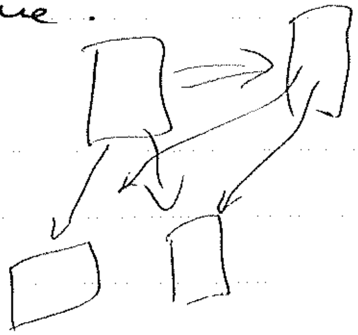
COW
copy on write

asgt by cloning
x=y.clone()



alloc new obj copy value.

shallow copy
deep copy }



Constants

- const = var: no location attrib.
- primitive
- immutable "variable"

has value semantics, not storage semantics
const = name for a value

static const = value computed @ compl time.
dynamic const = computed @ exec time
manifest const = name for a literal
function literal

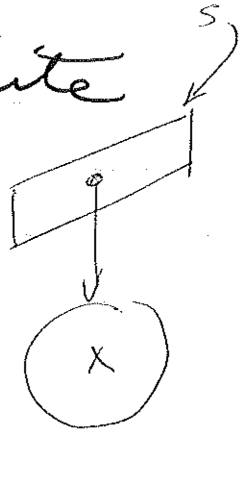
```
let f x = x*x ;;
let f = function x → x*x;;
```

5.7 Aliases Dangles Garbage

alias — two objects bound same value
- pass by reference
- copy pointers
- defeats optimization
  - copy on write

dangling references
- deallocate stg
- ref retained

memory leak



Loudon
5 Basic Semantics (11)

<u>Garbage</u>

- elim dangle ⟶ no free
- garbage wastes memory
- garbage collection
  - mem leak <u>less prob</u> dangle ref

techniques :

- ref count - good C^{++} method
  - can't handle cycles
- stop&copy - mark & sweep
  - copying
  - generational
- concurrent ✓

dangling ptrs
memory leak
aliasing.

live	reachable
dead	unreachable.