Runtime - 1

Data Representation

- size & alignment
- structs - holes.
- SVID — layout of structs unions
  - bit numbering
  - big / little endian
  - host order / network order.

ML

1, 2, 4, 8

Register usage

- free used ~~regs~~ regs.
- minimize bookkeeping regs.
- caller save / callee save.

eax rax
ebx
ecx
edx
esi
edi
esp
ebp
r8
:
r15

SP → top of runtime stack.

(BP) FP → beginning of current frame (may be virtual if frame fixed)

dynamic (control) link

static (access) link

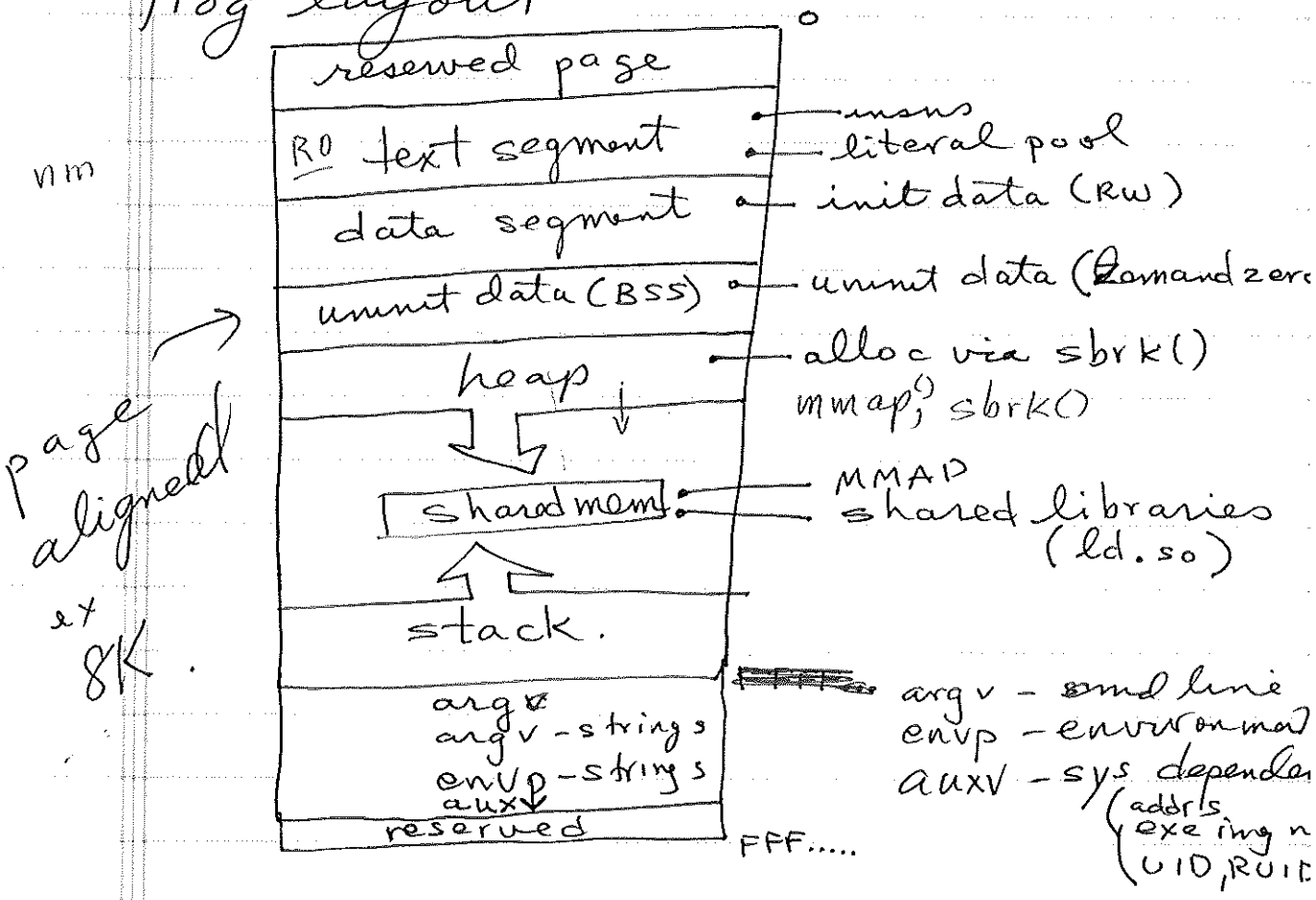
~~contents~~

volatile variables.

Prog layout

Runtime 2

nm



libraries - ~~ld~~ static —
ld dynamic — versioning pr

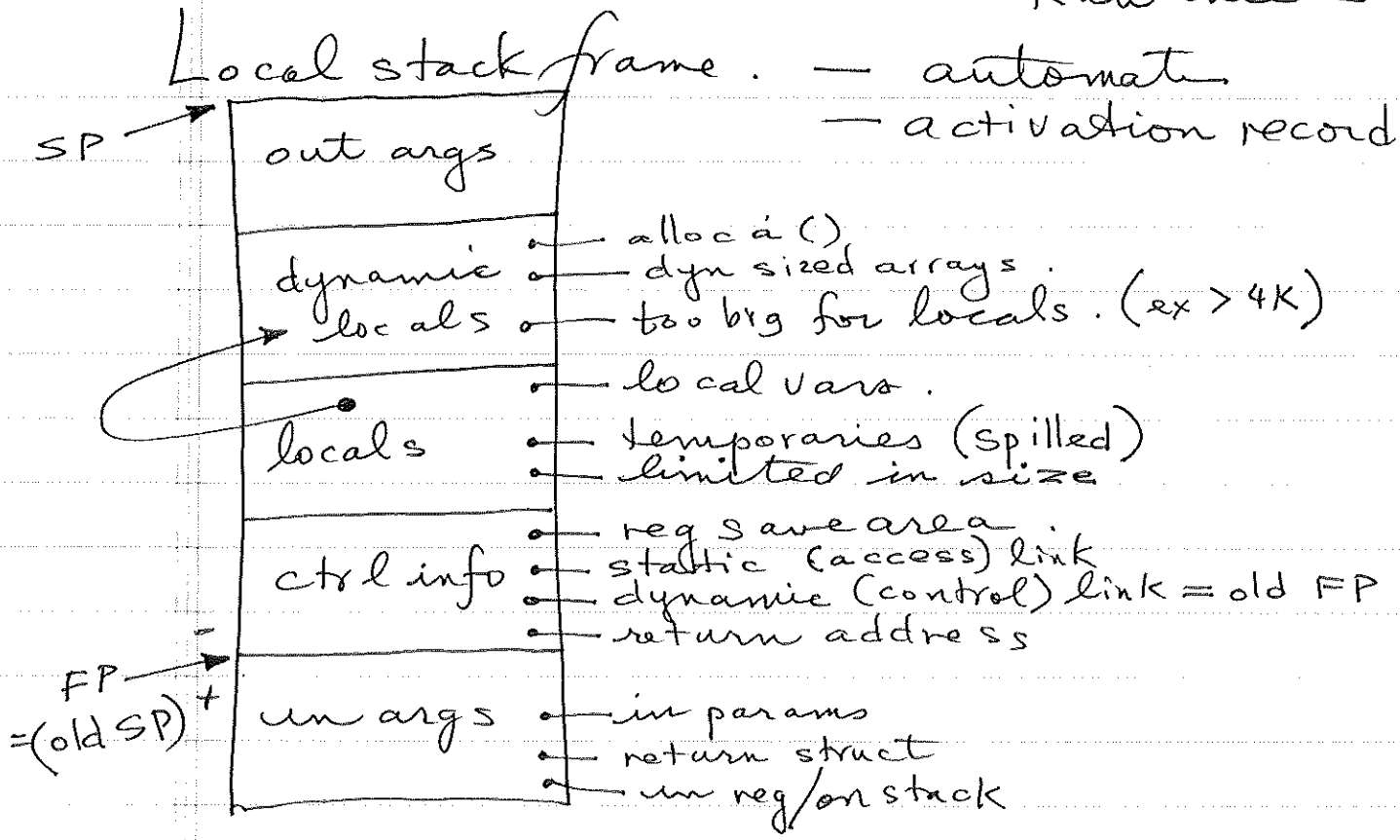
Variables = static | auto | heap.

static — easiest to alloc COMMON BLO
— addr @ link time
— problem with recursive fns
— class - single instance
— class table static.

scope: fn, file scope, external scope.

extern (vs) static
C — all fns static.

Local stack frame. — automatic — activation record.



recursion requires stack.

control

- static link
- dynamic link.
- global offset table ptr (code sharing)

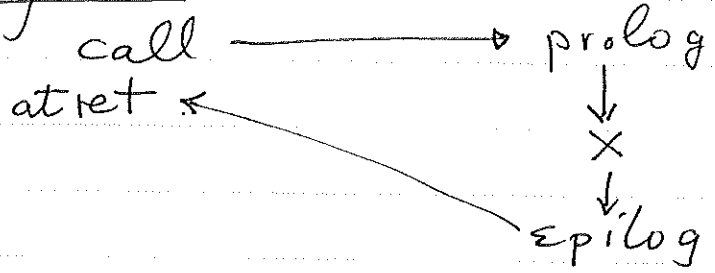
nested procs:

caller - callee next level

- (-1) — static link = caller's frame
- (0) — s.l → copy of caller's s.L.
- (n>0) — n static links back

Runtime 4

Calling Seq



call — assembles args.
- regs o0, o1, ... o5, then stack
- Pentium → reverse push.

gcc -S -O6

— get addr of fn (usu. static)
— ~~save~~ store caller-save regs if in use
— compute static link
— save ret addr & do call.

prolog — save old frame
— alloc new frame
— save callee save regs if used.
— construct display?

| X X X X |

epilog — load callee saved regs.
— recover old SP/FP
— set up ret value
— jump to retaddr.

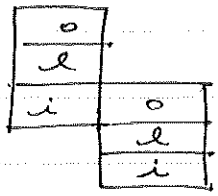
at return — load caller saved regs
— get return value.

Runtime 5

Param Passing

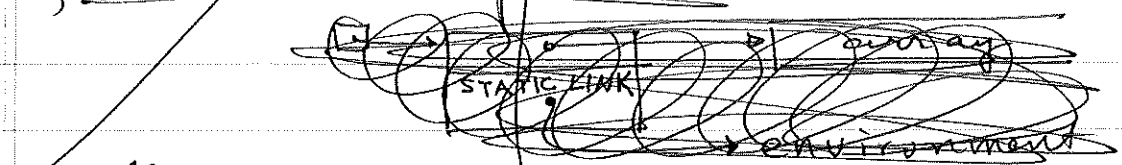
— reentrant procs.

- Runtime stack
  - pushed in reverse order
- passed in flat regs. #MIPS regs = 4
  - p^0, p^1, p^2, p^3 — load regs.
  - rest in some point on stack
  - must be saved
- passed in reg windows
  - #regs on Sparc = 6
  - windows cycle ~~MIPS = 4~~
- typical — 2 params per fn
  - value of

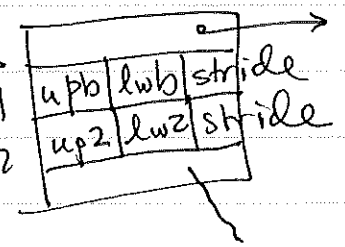


kinds / data
- by value (in) — small
- by result (copy back)
- by value-result
- by reference — large
- by name — thunk.
- by descriptor (ref/ref).

~~Functions as Parameters~~:



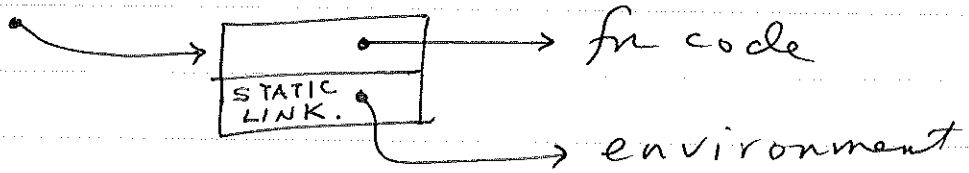
causes aliasing prob unless ~~recursive~~ immutable



slices.

passing fns

Runtime 6



Access to non-local

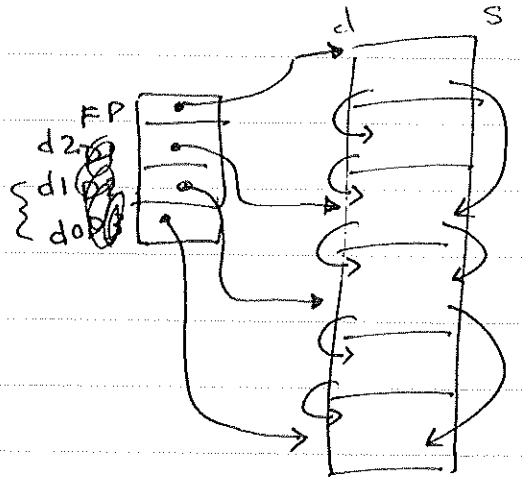
local ⇒ $-N(FP)$
or $+N(SP)$

global ⇒ absolute addr
or: $\pm N(gp)$ global ptr
(per module?)
(per class?)

other: follow n static links

display

```
f {
    g {
        h {
        }
    }
}
```



use d0 - most global
d1 - next
- save/restore @ each call.
- how many regs for display?

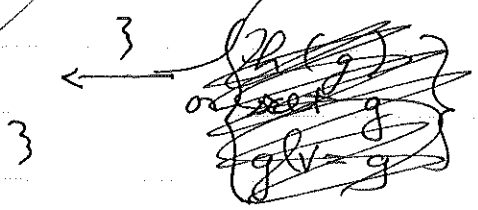
Static Lexical Scope

Runtime 7

- no nested procs
  ⇒ C ⟶ FP & global.

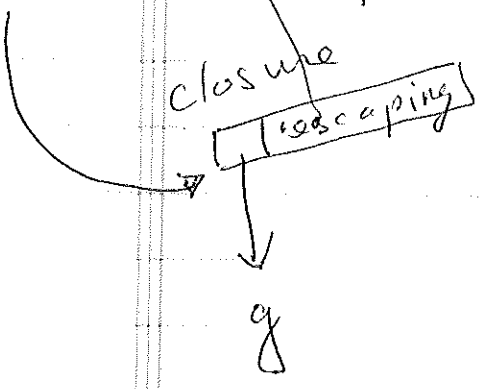
- nested:
  - if no fn params, not a problem
    just use stack
  - however: can't return local by ref
    or: assign & local → global.
  - what about returning a local fn,
    or passing it thru param

```
f() {
  int a
  g() {
    ret a+3;
  }
  call h(g)
  return g
  glovar = g
}
```



g needs f.a
but dangling when f retns.

a is an "escaping var"
- put it on the heap, access by ref
- put entire local frame f on heap
- no stack frame (except required by m/c)
- then gcol is needed or messy.



Dynamic Scope ("local" in Perl) — Runtimes

- deep access — search stack each access
- shallow access ~~— push new value~~
  - each var is a stack
  - local() ⟶ push new var
  - exit scope ⟶ pop it
- PL/1 called these "controlled vars"

PIC - Position Independent code

- shared libs mapped to shmem (RO)
- can't have any abs addrs
- don't need to relink prgs
- need dynamic linker

~~- mass~~ GOT = global offset table.
= offsets of external symbols

ex:
```
gp = GOT_off + 8
call next, r31
next: gp = gp + r31
```
rel to insn that uses it

PLT1: save regs
call dynlinker
function ID
——— linker zaps it to

PLT1: abs
g1 = addr(f)
call (g1)
↖ table in RW mem

trampoline

~~CTP-104-B page 7 week 4~~

Saving & Loading Regs

- preserve across call
  - caller save
  - callee save
- notice which insns change regs too
  - eg. vax movc5 → changes R0..R5
  - sparc call — ch % o7

SPARC save / restore } does not save regs just spins reg win

PREFERENCE	caller reg	callee reg	all reg.
callee save	1ST	3RD	5TH
caller save	2ND	4TH	6TH

↑ fewer regs

→ less code

~~1. callee save~~

1ST ee sv er reg. → need reg mask
- HW support
- w/o more efficient save allregs

2ND easy for retn inst
- global gotos difficult ... must find reload code.
- exceptions

3RD compiler knows regs in use — prolog must be backpatched
- maybe epilog has a backjump

4TH bitmaps only poss.
~~need~~ need reg mask.
VAX call w/ bitmap mask.

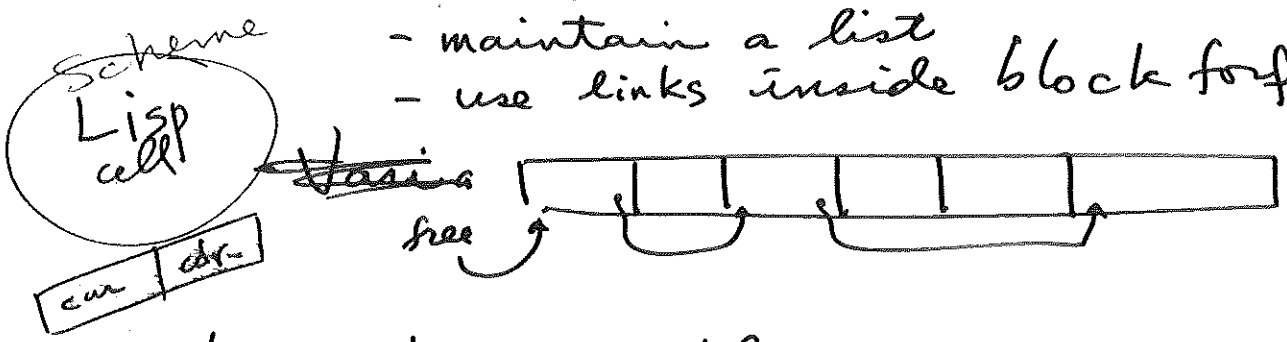
5TH. easy if few regs — unrolled loop if few regs.
STMult.

6TH easy - should use utility routine

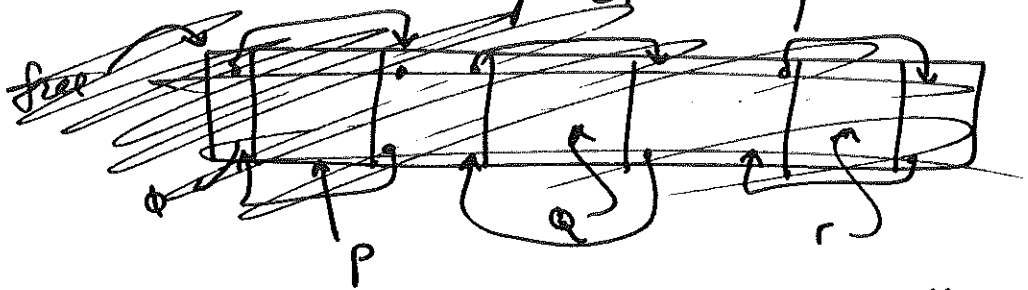
Heap Management

dyn storage allocation
- static + automatic (stack) not sufficient.
  - static: fixed addr; saved call to call
  - stack: unneeded if ∃ heap
    - used for efficiency
    - modern arch req it.
- heap: dyn stg. area (unrelated to heapsort)

- allocation (w/ explicit free)
  - Fixed sized blocks
    - maintain a list
    - use links inside block for free list



- Variable sized blocks
  - first fit or next fit
  - need to keep extra pointers



- boundary tags. & malloc 8 byte overhead

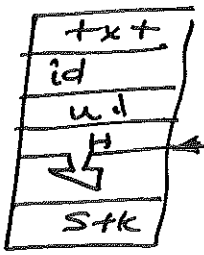
~~free(p)~~ ~~free(int *p)~~

size, link

- becomes messy & bad for virtual mem
- frags the pages.

- split free blocks → alloc
- coalesce when free.

malloc()



sbrk() : always mult of page size.

- compatible: new vs malloc
- portable: only use sbrk() syscall.
  - localize. for other os.
- minimize time
- locality → paging & caching

Fastest free(){}:

~~malloc(){~~
~~p = endheap;~~
~~endheap += s;~~
~~if(endheap > brk)~~

```
malloc(n){
    p = firstfree;
    ~~firstfree += n;~~
    firstfree = (firstfree + n + 7) & ~7;   ← alignment.
    if(firstfree > break){                  ← magic bitpat 0xF? instead
        break = (firstfree + extendamt
                 + pgsiz - 1) & ~(pgsiz - 1);
        sbrk(break)
        assert(ok)
    }
    return p
}
```

$pgsiz = 2^K$ (1K to 16K?)

extendamt = n * pgsiz.

magic #

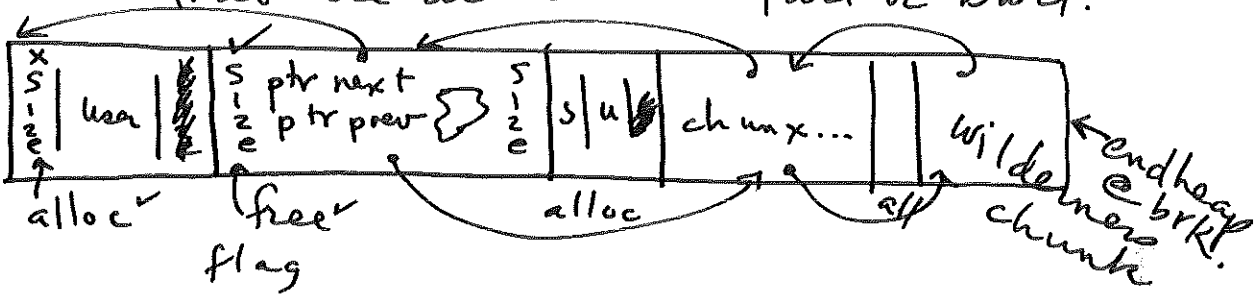
alignment: 8 for double. (strictest hardware req.)

2^K: 8 or 16?

pentium base reg granularity?

actual alg

Boundary tags:
- size fields before and after chunk
- coalesce bordering unused chunks
- traverse all chunks fwd or bwd.



Bins: - maintain chunks in bins, by size
- 128 logarithmic size bins.
- smallest-first, best fit order.

size: 16, 24, 32 ... 512, 576, 640, ..., 2^{31}
(8x64, 9x64, 10x64)

exact bins — sorted bins.

- use wilderness chunk only when all others fail (assume ∞ size)

- sbrk() - extends wilderness.
- mmap() - chunk between heap & stk
  - only for very large req.
  ex ≥ 1MB.
  & not avail in exist.

syscalls are expensive

caching: lazy coalescing
- most pgms alloc/free chunks of a few fixed sizes.

ARENAS:

array of fixed sized chunks
good for structs (classes)
less useful for arrays