Signature: like interface
Structures: the actual type variable implementing interface
Functors: compile time functions from sig —> sig

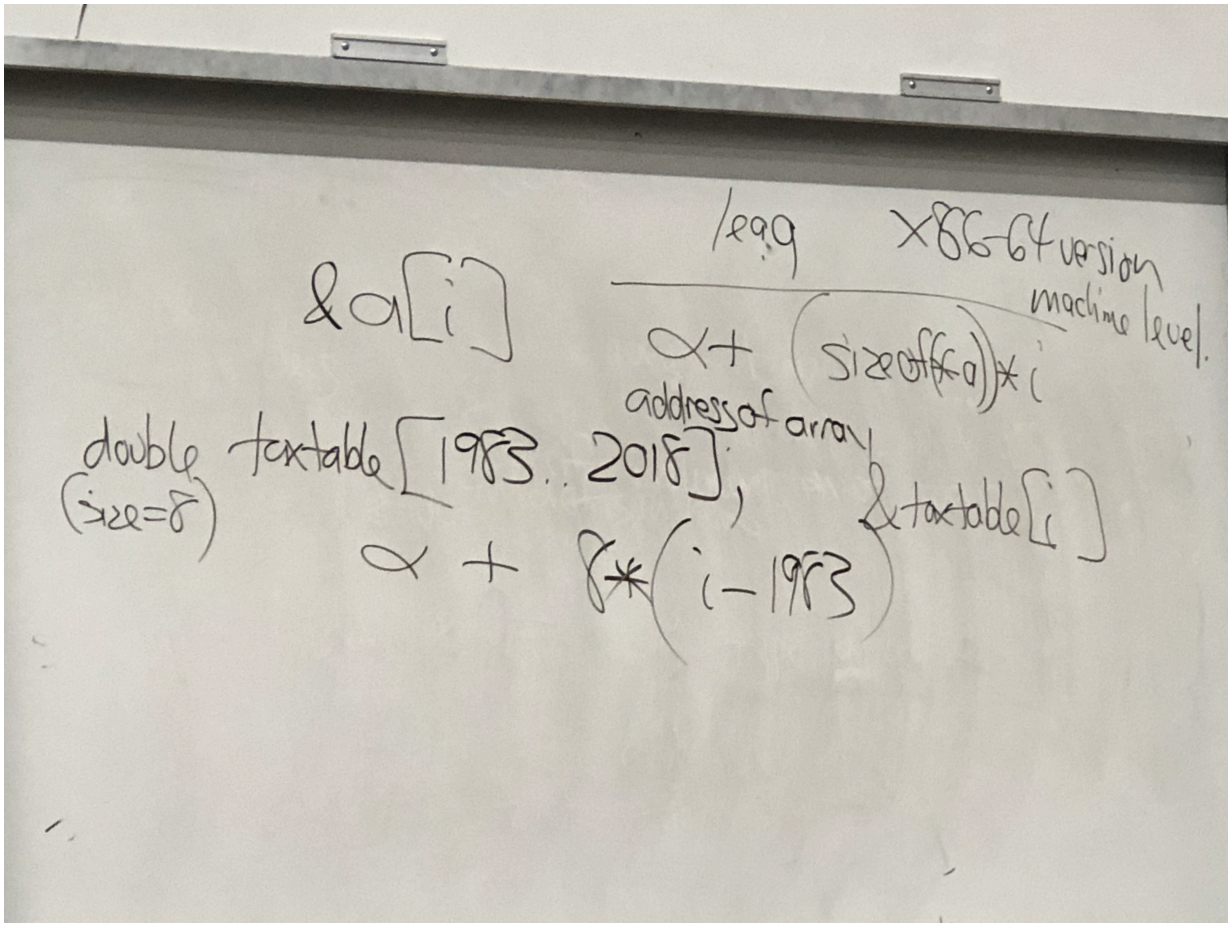
- more powerful than signature
- generates compile time functions from signatures

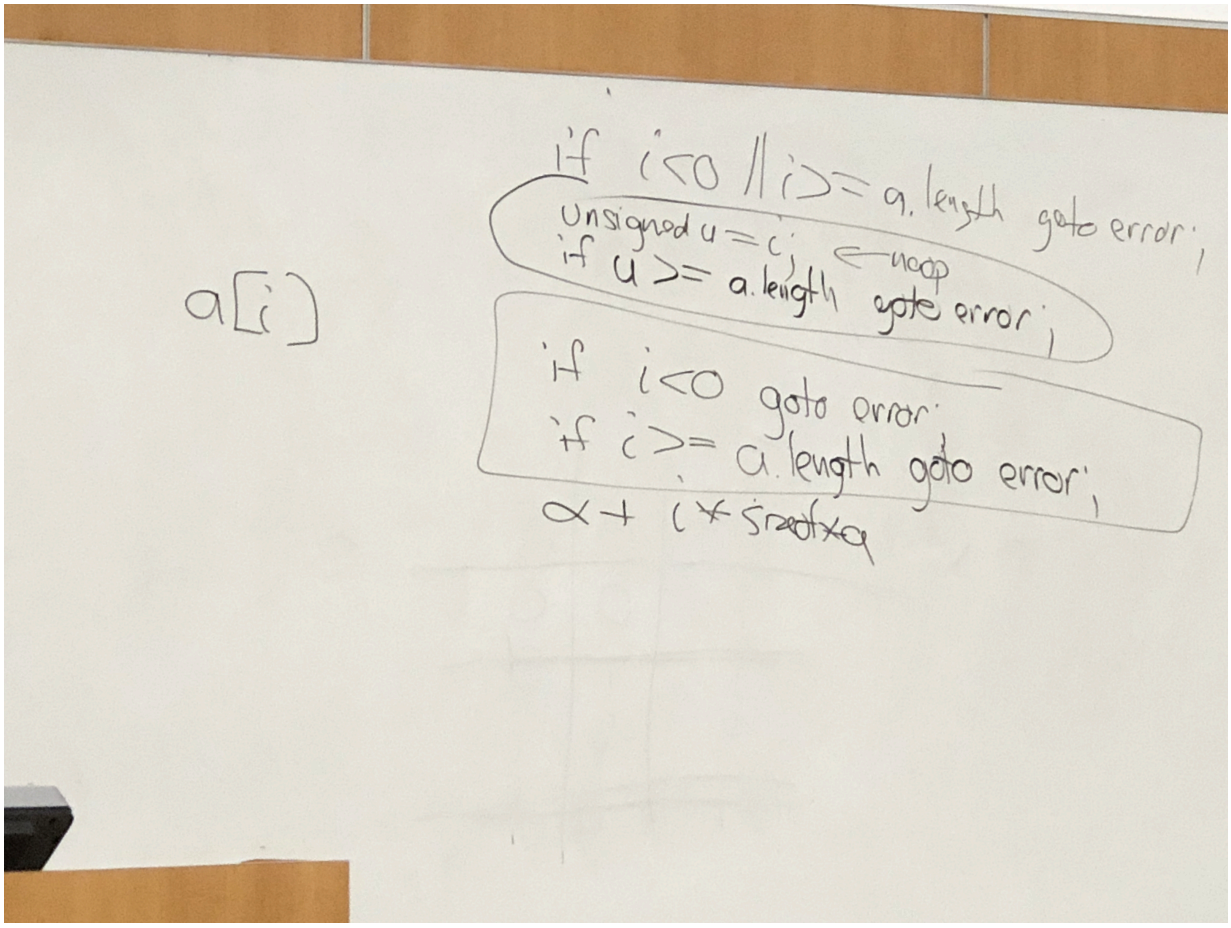
Storage management
warmup. arrays.

All the following need storage:

- string literals
- objects computed dynamically
  - new
- local vars \
- stack for recursion
- pointers to stack locations
- functions (machine code, closures)
- VNCZ flags (etc.)
- function args
- return addresses
- buffers for I/O
- memory manager

Closures
functions packed in other functions that carry information about the function returned
(fun x -> fun y -> x + y) 12
getchar()


&a[i]
e.g. x86-64 version
machine level.
α + (sizeof(*a))*i
address of array
double taxtable[1983..2018];
(size=8)
α + 8*(i-1983)
&taxtable[i]



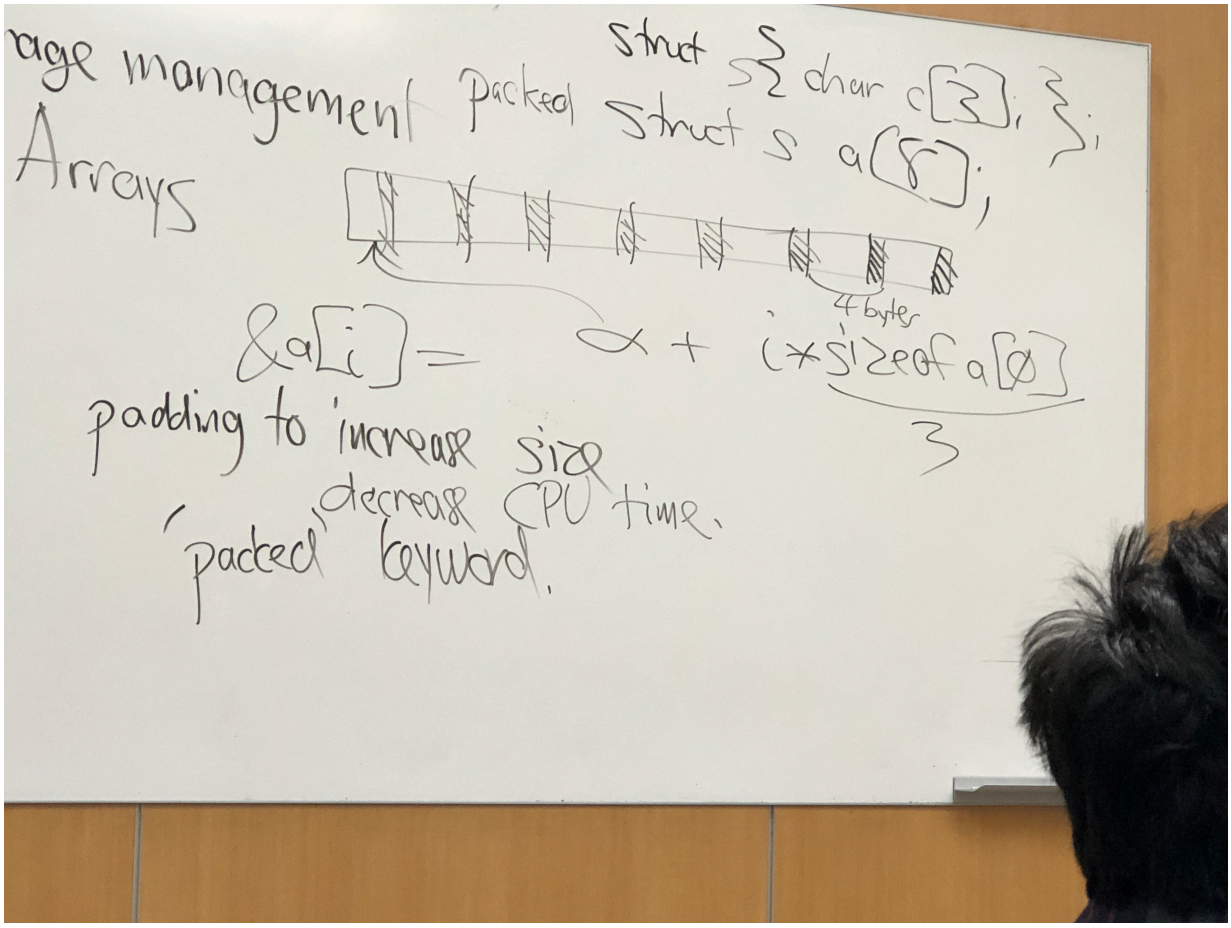
Week 7 Lecture 2

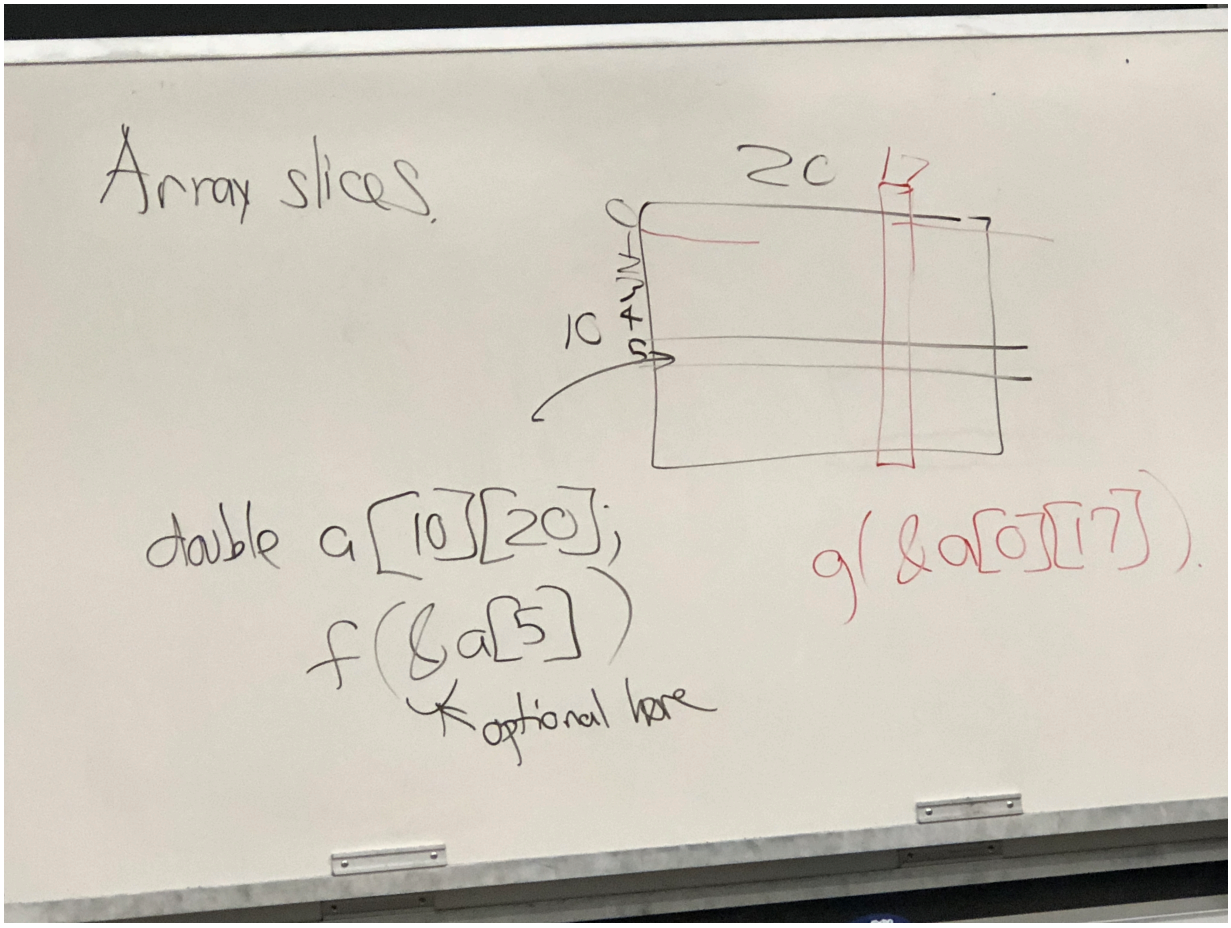
==========

Storage Management

- Arrays

struct s { char c[3]; }
packed struct s a[8];

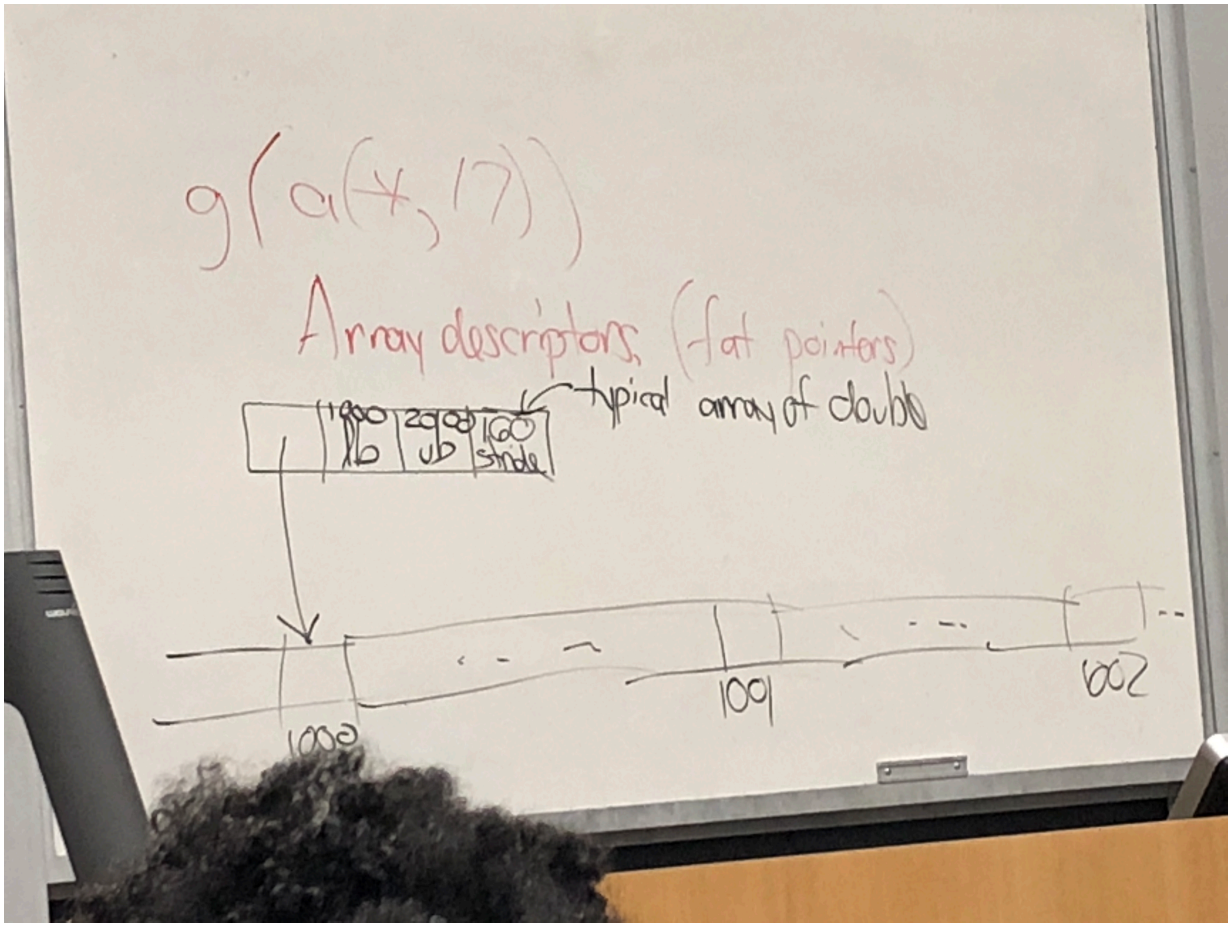

age management packed
struct { char c[3]; };
struct s a[8];
Arrays
4 bytes
&a[i] = α + (i * sizeof a[0])
3
padding to increase size
decrease CPU time.
'packed' keyword.



*** Talking about Fortran

Array Descriptors
Are like “fat pointers”

- Caller’s responsibility to build up array descriptor, so it knows the stride length
- The array descriptor has a lower bound, upper bound, and stride



Fortran has the fastest arrays with their array descriptors, is because it's used for scientific applications that use 2d arrays very often.

C came late to the game so it catered to different audience.

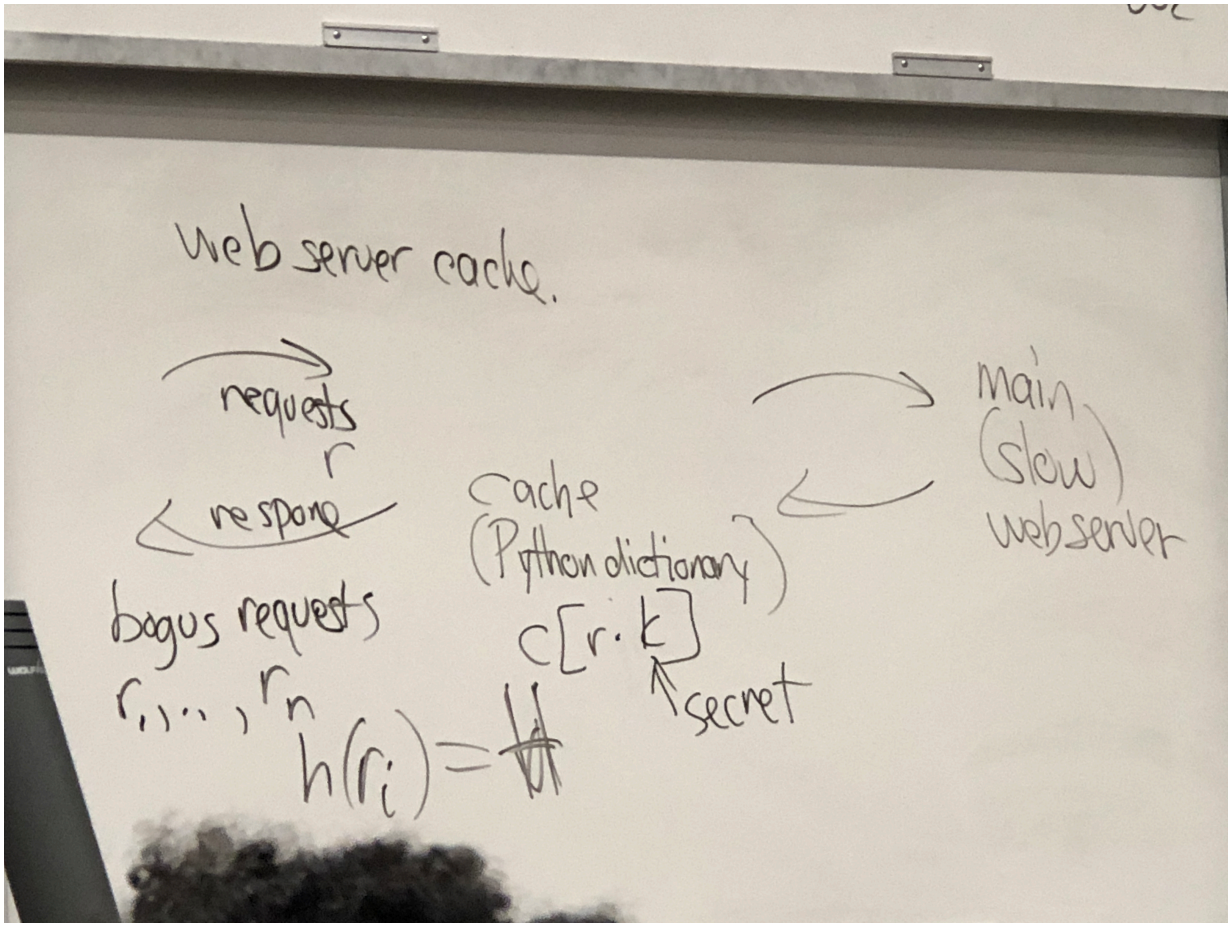
Associative Arrays

- dictionaries in Python
  - d['abc'] = 'def'
  - can build using hash tables

Web server cache example
takes requests ->
maintains cache (python dictionary)
looks up stuff from request in cache

- if finds it, sends a response (fast)
- if doesn't find it, checks with the web server (slow)

To do DDOS attack, we make a bunch of bogus requests r_1, ..., r_n that will all hash to same value h(r_i) = H.



Weak references vs strong references

Weak	Strong
can be removed at any time	“permanent”

Where to put arrays?
Arrays are large
Once we figure out where to allocate array, rest is pretty simple

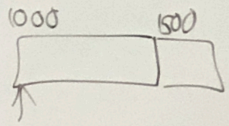
Array allocation strategies

- **static allocations** needs n bytes for each array
- see picture

Where to put arrays?

Array allocation strategies.

- Static allocation
  (before program runs)
  + no run overhead
  + no crashes due storage exhaustion.
  + compiler can optimize more.
  − hassle to change sizes — recompile
  − overallocation is common.
  − hard to have temp arrays



For next time

Dybvig §§1 – 3.4
(Scheme)