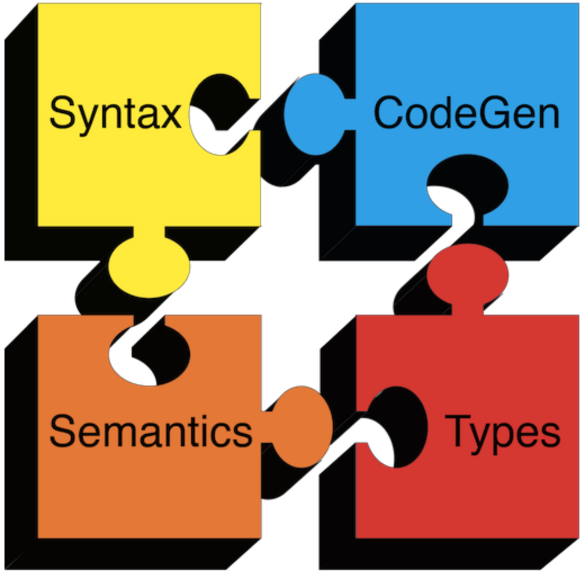


Compilers

Globals & Heap

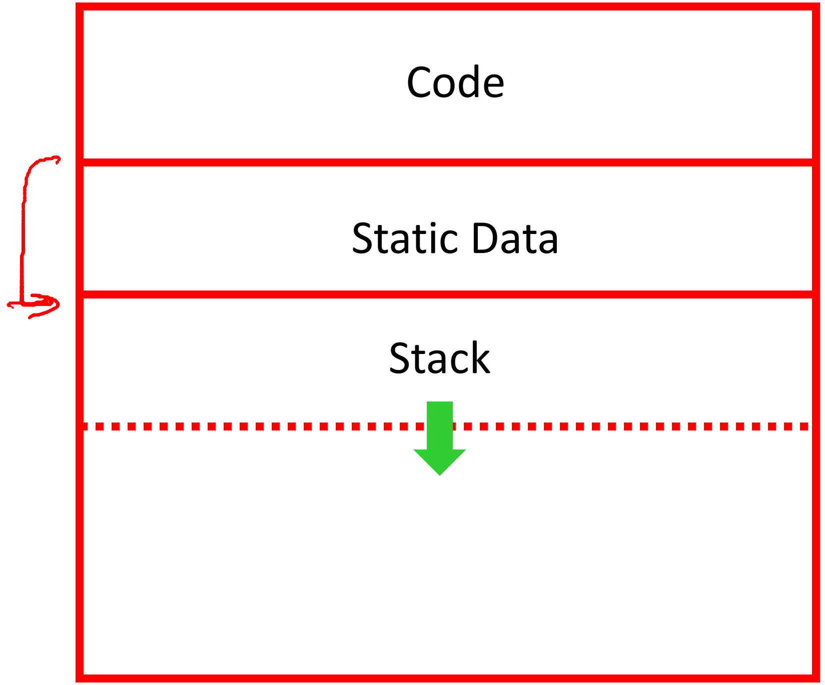
Alex Aiken

Globals & Heap

- All references to a global variable point to the same object
  - Can't store a global in an activation record
- Globals are assigned a fixed address once
  - Variables with fixed address are "statically allocated"
- Depending on the language, there may be other statically allocated values

Globals & Heap

Memory

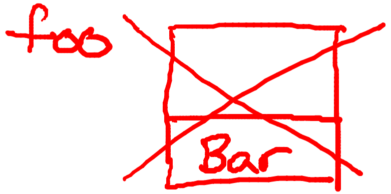


Low Address

High Address

Globals & Heap

- A value that outlives the procedure that creates it cannot be kept in the AR



method foo() { <u>new Bar</u> }

The Bar value must survive deallocation of foo's AR

- Languages with dynamically allocated data use a *heap* to store dynamic data

Globals & Heap

- The code area contains object code
  - For ~~most~~ many languages, fixed size and read only
- The static area contains data (not code) with fixed addresses (e.g., global data)
  - Fixed size, may be readable or writable
- The stack contains an AR for each currently active procedure
  - Each AR usually fixed size, contains locals
- Heap contains all other data
  - In C, heap is managed by *malloc* and *free*

In Java, new

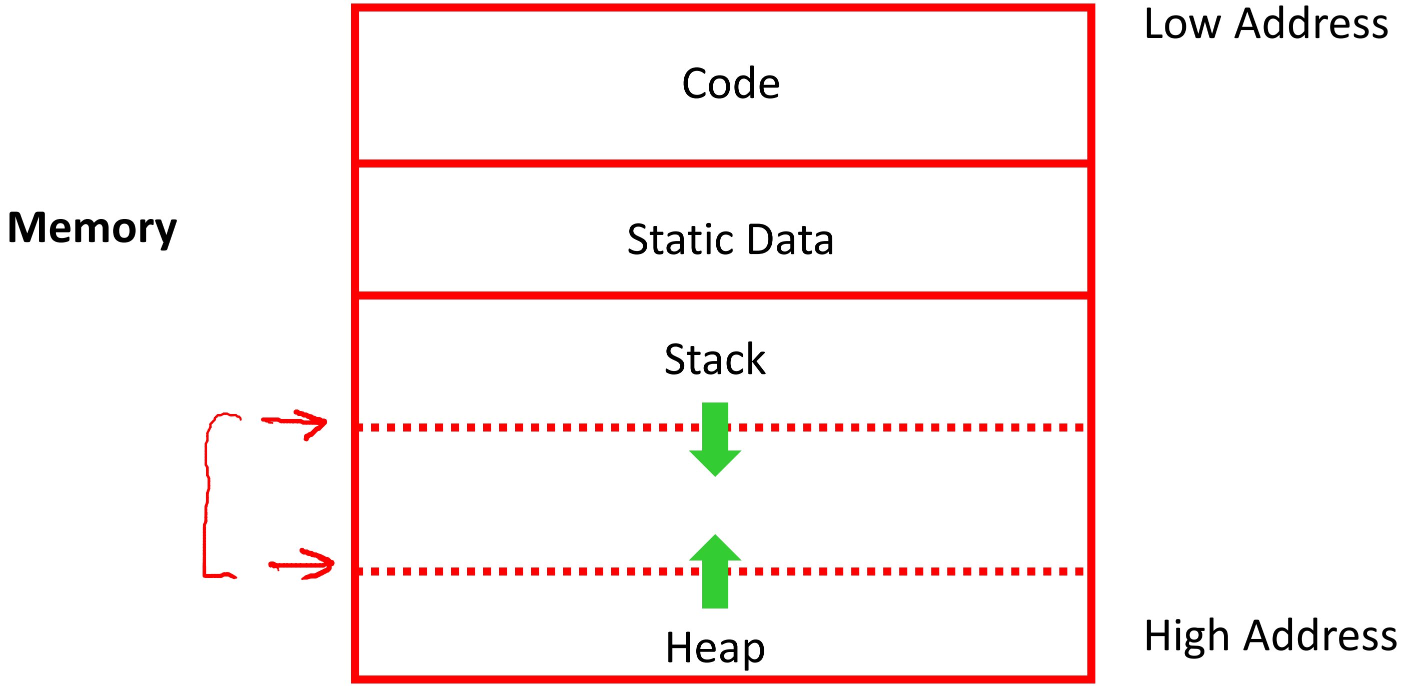
Globals & Heap

- Both the heap and the stack grow

- Must take care that they don't grow into each other

- Solution: start heap and stack at opposite ends of memory and let them grow towards each other

Globals & Heap


Memory
Code
Static Data
Stack
Heap
Low Address
High Address