Compiler Construction

Activation Blocks

Memory Hierarchy

Different kinds of memory in a computer, with different performance:

- **Registers** Small memory units built on the CPU (bytes, 1 cycle)
- **L1 Cache** Last main memory access results (kB, 2-3 cycles)
- **L2 Cache** (MB, 10 cycles)
- **Memory** The usual RAM (GB, 100 cycles)
- **Storage** Disks (100GB, TB, $>$ 1Mcycles)

Use the registers as much as possible.

Register Overflow

What if there are not enough registers?
Use the main memory, but how?
Recursion:

Without
: Each name is bound once. It can be statically allocated a single unit of main memory. (Cobol, Concurrent Pascal, Fortran (unless **recursive**)).

With
: A single name can be part of several concurrent bindings. Memory allocation must be dynamic.

Dynamic Memory Allocation (1/2)

Depending on the persistence, several models:

Global
: Global objects, whose liveness is equal to that of the program, are statically allocated (e.g., **static** variables in C)

Automatic
: Liveness is bound to that of the host function (e.g., **auto** variables in C)

Heap
: Liveness is independent of function liveness:

Dynamic Memory Allocation (2/2)

Heap Liveness is independent of function liveness:

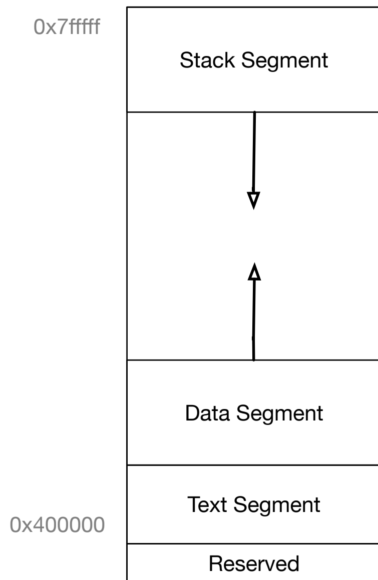
User Controlled

malloc/**free** (C),
new/**dispose** (Pascal),
new/**delete** (C++) etc.

Garbage Collected

With or without **new**
(`lisp`, Smalltalk, ML,
Haskell, Tiger, Perl etc.).

spim Memory Model



Stack Management

Function calls is a last-in first-out process, hence, it is properly represented by a stack.

Or...

"Call tree": the complete history of calls. The execution of the program is its depth first traversal.
Depth-first walk requires a stack.

Activation Blocks

- In recursive languages, a single routine can be “opened” several times concurrently.
- An **activation** designates one single instance of execution.
- Automatic variables are bound to the liveness of the activation.
- Their location is naturally called **activation block**, or **stack frame**.

Activation Blocks Contents

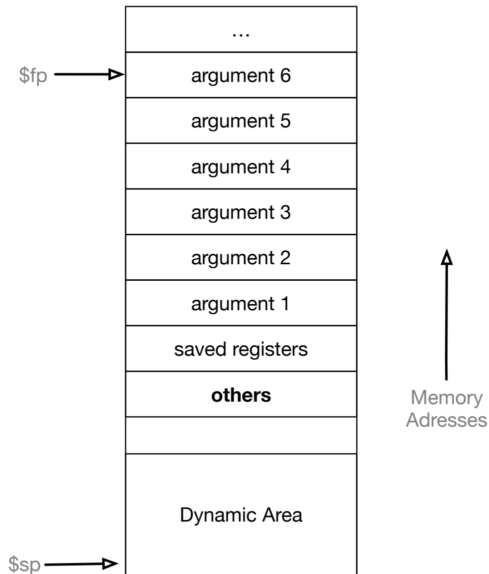
Data to store on the stack:

arguments	incoming
local variables	user automatic variables
return address	where to **return**
saved registers	the caller's environment to restore
temp	compiler automatic variables, spills
static link	when needed

Activation Blocks Layout

The layout is suggested by the constructor.
Usually the layout is from earliest known, to latest.

Activation Blocks Layout on MIPS



Memory
Adresses

Frame & Stack Pointers

The stack of activation blocks is implemented as an array with

frame pointer
: the inner frontier of the activation block

stack pointer
: the outer frontier

Usually the stack is represented growing towards the bottom.

Flexible Automatic Memory

`auto` Static size, automatic memory.

`malloc` Dynamic size, persistent memory.

Automatic memory is extremely convenient...

```
int
open2(char* str1, char* str2, int flags, int mode)
{
  char name[strlen(str1) + strlen(str2) + 1];
  stpcpy(stpcpy(name, str1), str2);
  return open(name, flags, mode);
}
```

Flexible Automatic Memory

malloc is a poor replacement.

```
int
open2(char* str1, char* str2, int flags, int mode)
{
  char* name
    = (char*) malloc(strlen(str1) + strlen(str2) + 1);
  if (name == 0)
    fatal("virtual memory exceeded");
  stpcpy(stpcpy(name, str1), str2);
  int fd = open(name, flags, mode);
  free(name);
  return fd;
}
```

Flexible Automatic Memory

alloca is a good replacement.

```
int
open2(char *str1, char *str2, int flags, int mode)
{
  char *name
    = (char *) alloca(strlen(str1) + strlen(str2) + 1);
  stpcpy(stpcpy(name, str1), str2);
  return open(name, flags, mode);
}
```

Summary