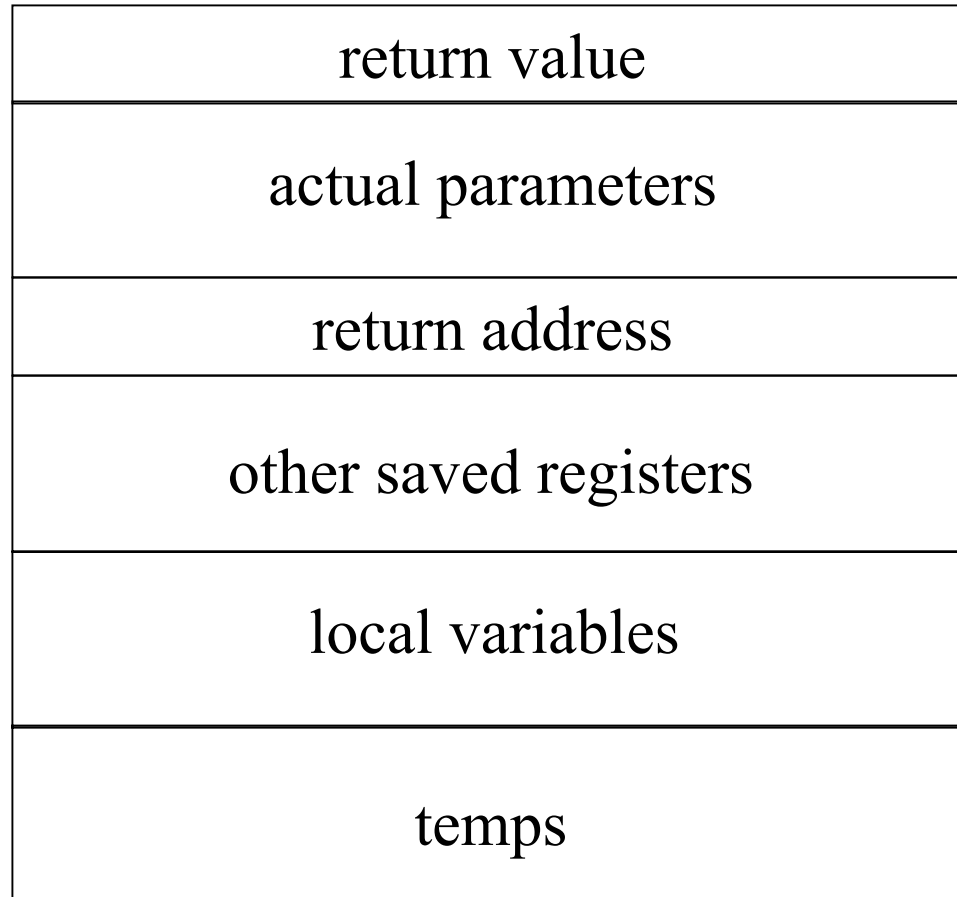


# Runtime Environments II

- Sample activation record layout
- Call and return sequence
- Accessing data in an activation record

# Sample Activation Record Layout



# Call sequence

Call sequence is split between caller and callee:

- code at call-site to start setting up activation record (a.r.) of callee
  - evaluate and store actual parameters (in callee a.r.)
- control jumps to callee
- code at beginning of callee func def. finishes setting up it's a.r.
  - save return address on a.r.
  - save other registers that it may overwrite (old fp? temp regs?)
  - “alloc” space for locals and temps (i.e., advance sp)

# Return sequence

Return sequence is split between callee and caller

- code at end of callee func def. starts return sequence
  - save return value in my a.r.
  - restore old register values from my a.r.  
(i.e., the ones we saved when we were first called)
  - de-allocate space for a.r. (move s.p. back)
- control jumps to caller
- code at call site (after call) finishes return sequence
  - save return value from callee's a.r.  
(now that space may be reused for another a.r.)

# Accessing data in an activation record

- Compiler must statically determine layout of a.r.'s and produce code that accesses the correct locations.
- At compile-time you determine the size and layout of the a.r. for the current routine.
  - each local variable (and param) will have an assigned offset in this a.r.
  - these will be assigned sequentially as you get to each declaration (part of code generation)
  - store the offsets in the symbol table

# Accessing data in an activation record (cont.)

- To generate code for a reference to a variable **x**:
  - look up offset, ***off***, in symbol table entry for **x**
  - gen code such as: ***off* (**reg**)**
    - at run time *reg* contains address of some fixed loc in the a.r.
    - *off* is a constant in the generated code