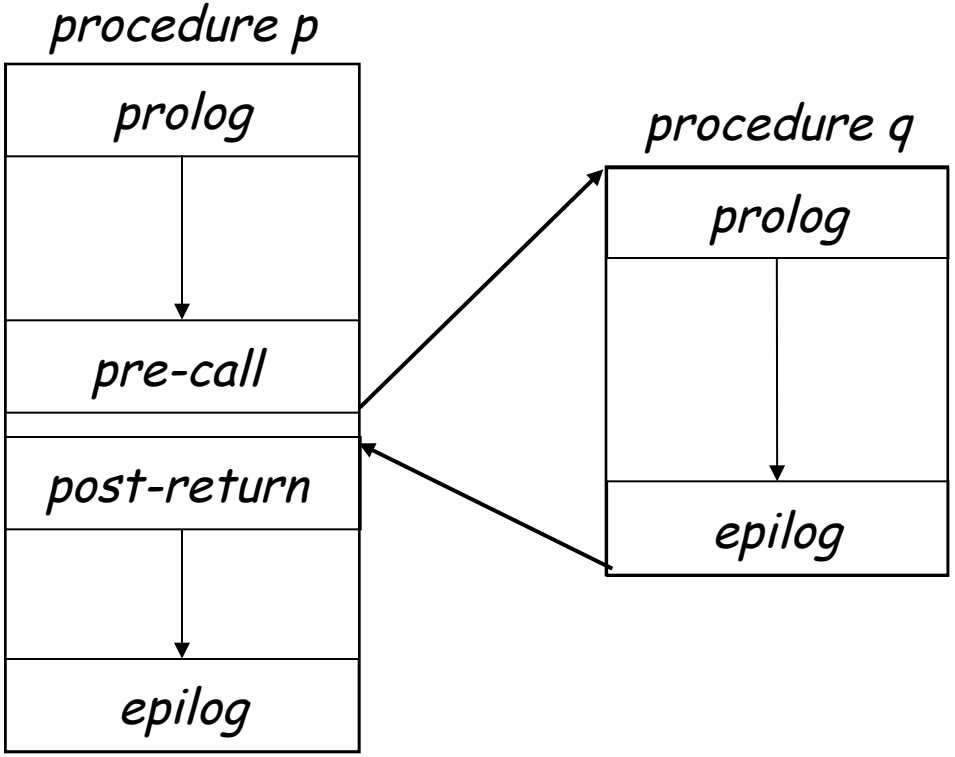
Code Shape I
Procedure Calls, Dispatch,

Procedure Linkages

Standard procedure linkage



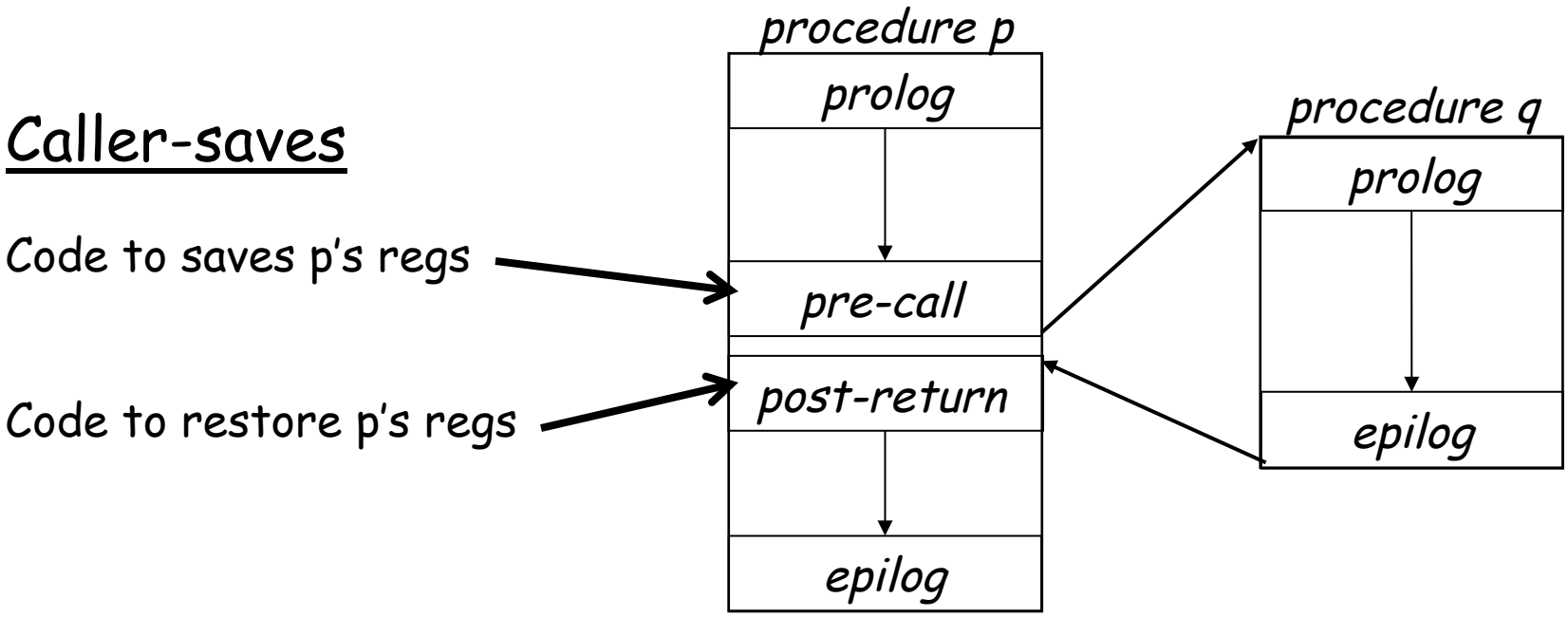
Procedure has
- standard prolog
- standard epilog

Each call site involves a
- pre-call sequence
- post-return sequence

Caller-saves versus callee saves

If *p* calls *q*, one of them must

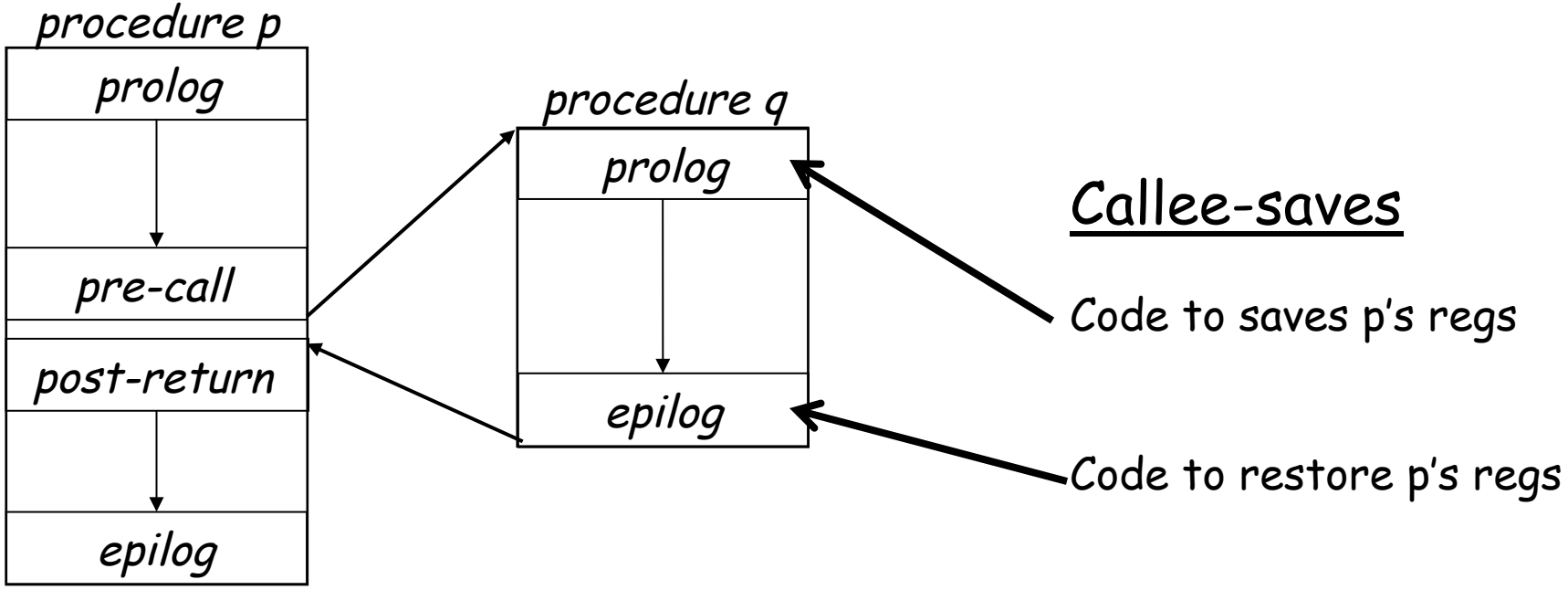
- Preserve register values
  - → Caller-saves registers stored/restored by *p* in *p*'s AR



Caller-saves versus callee saves

If *p* calls *q*, one of them must

- Preserve register values
  - → Callee-saves registers stored/restored by *q* in *q*'s AR



Implementing Procedure Calls

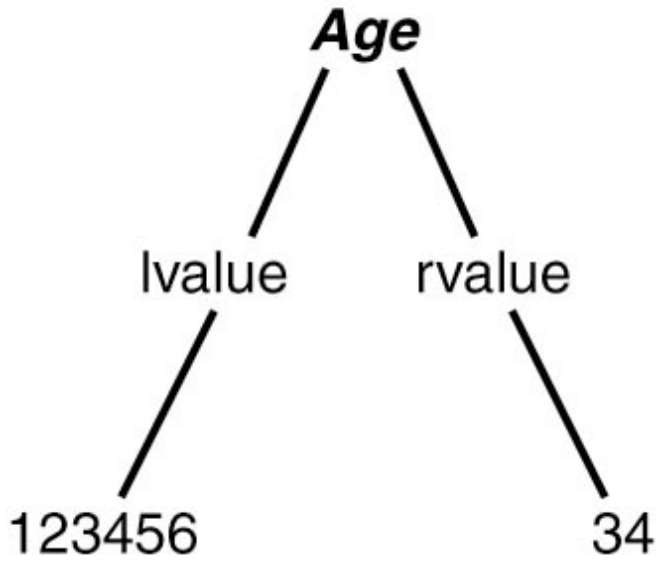
- Allocate the AR
  - → Heap allocation ⇒ callee allocates its own AR
  - → Stack allocation ⇒ caller & callee cooperate to allocate AR

Space tradeoff

- Pre-call & post-return occur on every call
- Prolog & epilog occur once per procedure
- More calls than procedures
  - → Moving operations into prolog/epilog saves space

Evaluating parameters

- Usually language definition issue
- Call by reference ⇒ evaluate parameter to an lvalue
- Call by value ⇒ evaluate parameter to an rvalue & store it



Evaluating parameters

Aggregates (structs), arrays, & strings are usually **call by reference**

- Alternatives
  - → Small structures can be passed in registers
  - → Can pass large c-b-v objects c-b-r and copy on modification

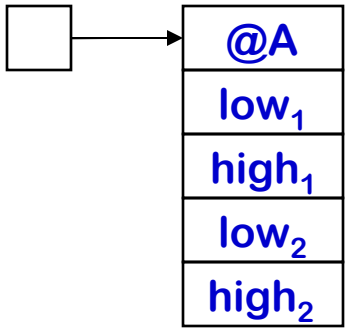
Procedure-valued parameters

- Must pass starting address of procedure

What about arrays as actual parameters?

Whole array as call-by-reference parameter

- Callee needs dimension information
  - → Builds a descriptor called a *dope vector*
- Store the values in the calling sequence
- Pass the address of the dope vector in the parameter slot
- Generate complete address polynomial at each reference



dope vector

What about A[12] as an actual parameter?

If corresponding parameter is a scalar, it's easy

- Pass the address or value, as needed

What if corresponding parameter is an array?

- See previous slide

What about a string-valued argument?

- Call by reference
  - → pass a pointer "descriptor" to start of string
- Call by value ⇒ copy the string & pass it
  - → Can store it in callee's AR
  - → Can pass by reference & have callee copy **it if necessary**

Pointer of string serves as "descriptor" for the string, stored in the appropriate location (register or slot in the AR)

What about a structure-valued parameter?

- Again, pass a descriptor or "handle"
- Call by reference
  - → descriptor (pointer) refers to origin
- Call by value ⇒ create copy & pass descriptor
  - → Can allocate it in callee's AR
  - → Can pass by reference & have callee copy **it if necessary**

If it is actually an array of structures, then use a dope vector

What About Calls in an OOL (Dispatch)?

In an OOL, most calls are indirect calls

- Compiled code does not contain address of callee
  - → Finds it by indirection through class' method table
  - → Required to make subclass calls find right methods
  - → Code compiled in class *C* cannot know of subclass methods that override methods in *C*

What About Calls in an OOL (Dispatch)?

- In the general case, need dynamic dispatch
  - → Map method name to a search key
  - → Perform a run-time search through hierarchy
    - ♦ Start with object's class, search for 1st occurrence of key
    - ♦ This can be expensive
  - → Use a method cache to speed search
    - ♦ Cache holds < ***key, class, method pointer*** >