Runtime System

- First more about symbol tables
  - See AbsractSyntax2 lecture
- Procedure activations
- Activation records
- Runtime stack
- Register save disciplines
- Lexical scoping and static links

Symbol Tables

- **In Tiger compiler, symbol tables are not persistent**
  - **Built, mutated and used during type checking**
  - **Implications**
    - **Not built during parsing**
    - **Types need to be embedded in AST entries or cannot be used by later compiler phases (e.g., block structure and its locals) because of destructive updates to *Table* objects**

Symbol Tables - Alternatives

- **Build one symbol table per scope**
- **Keep list of currently active symbol tables for correct lookup**
- **Keep list of ALL symbol tables and thread them together by the lexical relationships of their corresponding scopes**
- **Can build as parse declarations**
- **Can save for debugging or profiling usage**

Job of Runtime System

- **Names versus data objects**
  - **Same name can refer to different data objects during execution; runtime system provides the mapping**
- **Procedure *activations***
  - **Each time a procedure is called, a new activation of that procedure occurs within an environment (who called it? where it was called from? what declarations are active at call site?)**
  - ***Recursion* - a new activation of same procedure can start before an earlier activation has ended**

Activation Lifetime

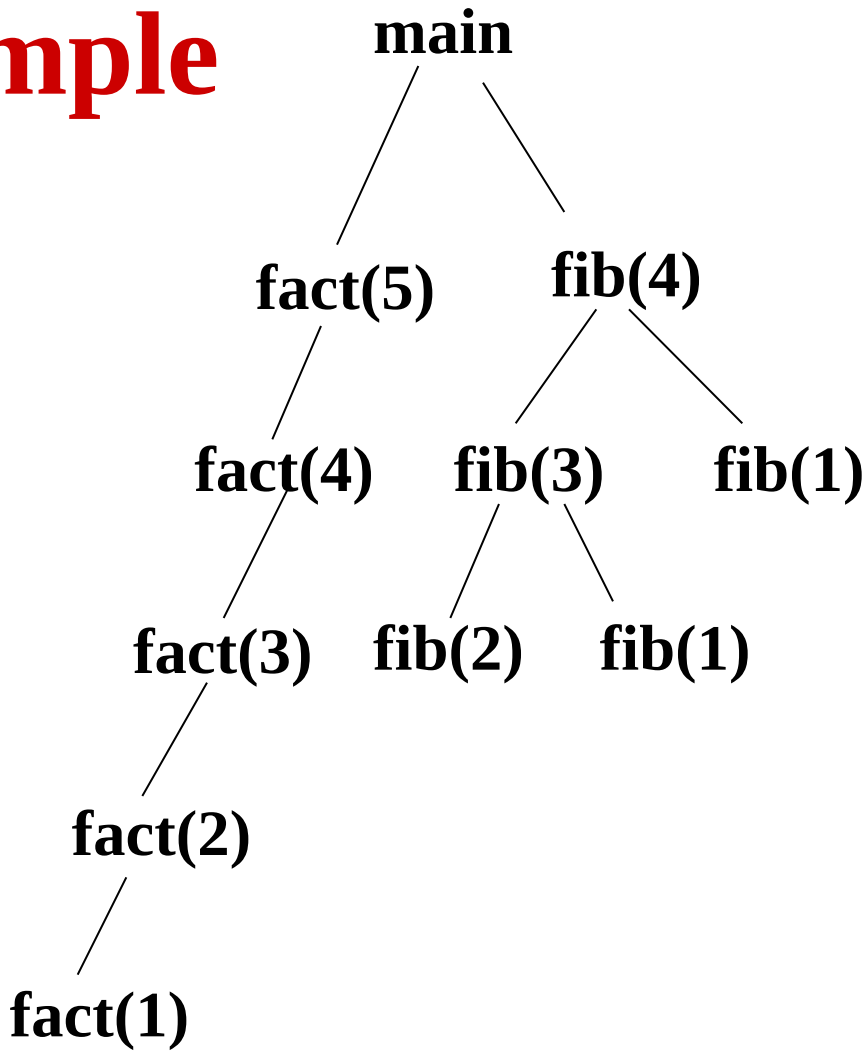
- ***Lifetime of an activation of p:*** **sequence between first and last steps in execution of procedure body, *including* time spent executing any procedures called from p**
- **Block structured languages allow only *nested* procedure lifetimes**
  - **Allows use of stack to define runtime environment**
  - **Can show relations in *procedure activation tree***

Procedure Activation Tree

- **Each node is a procedure activation, each edge represents opening an activation while the parent activation is still open**
- **Flow of control in program is depth-first traversal of activation tree**
- **A node a is to left of node b in tree, if lifetime of a occurs before lifetime of b**
- **Root is main program activation**

Example

```
main
{   fact(x){…fact(x-1)}
    fib(y) { … fib(y-1)+fib(y-2)}
    fact(5)
    fib(4)
}
```



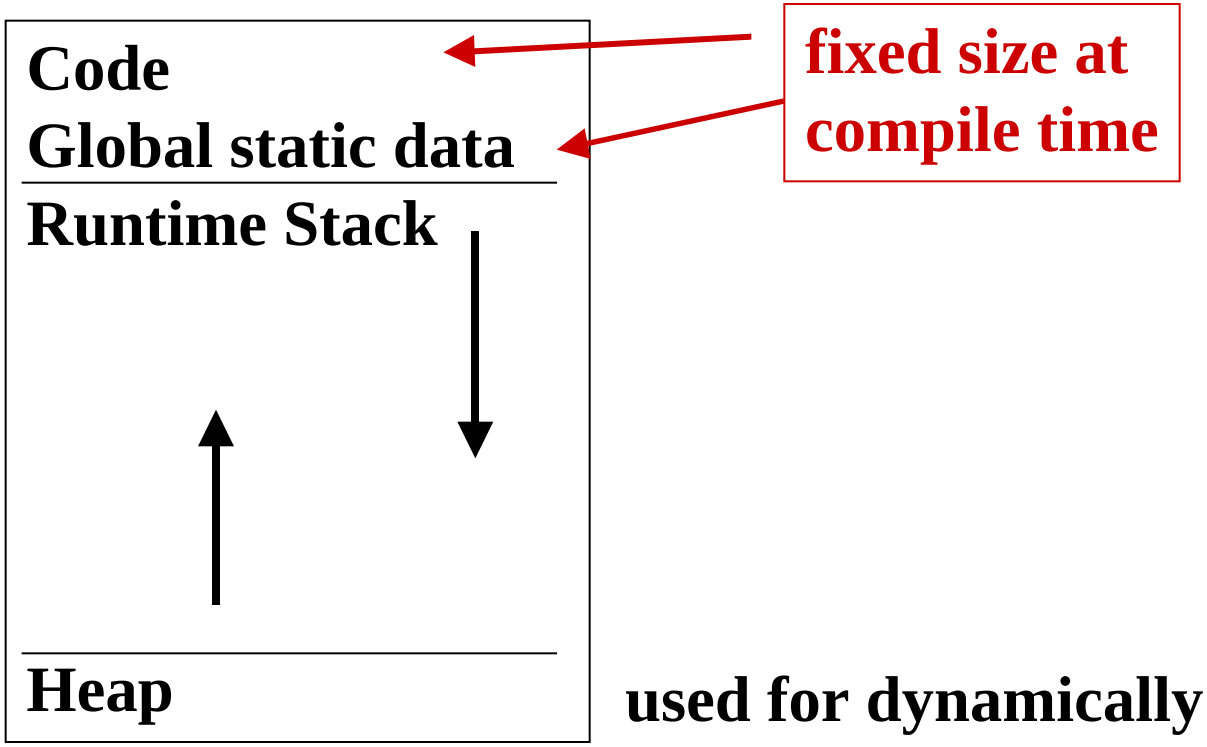
Procedure Activation Tree

- **Depth first traversal of procedure activation tree represents the sequence of procedure activations as they occur during execution**
- **During traversal, stack of procedures on current path represents currently active procedures**
  - **Sometimes called the *control stack***

Q's re:Runtime Support

- **Is recursion allowed?**
- **Can a procedure refer to non-local names?**
- **How are parameters passed?**
- **Are functions/procedures *first class*?**
- **How can storage be dynamically allocated and deallocated?**

Imperative PL Memory Model



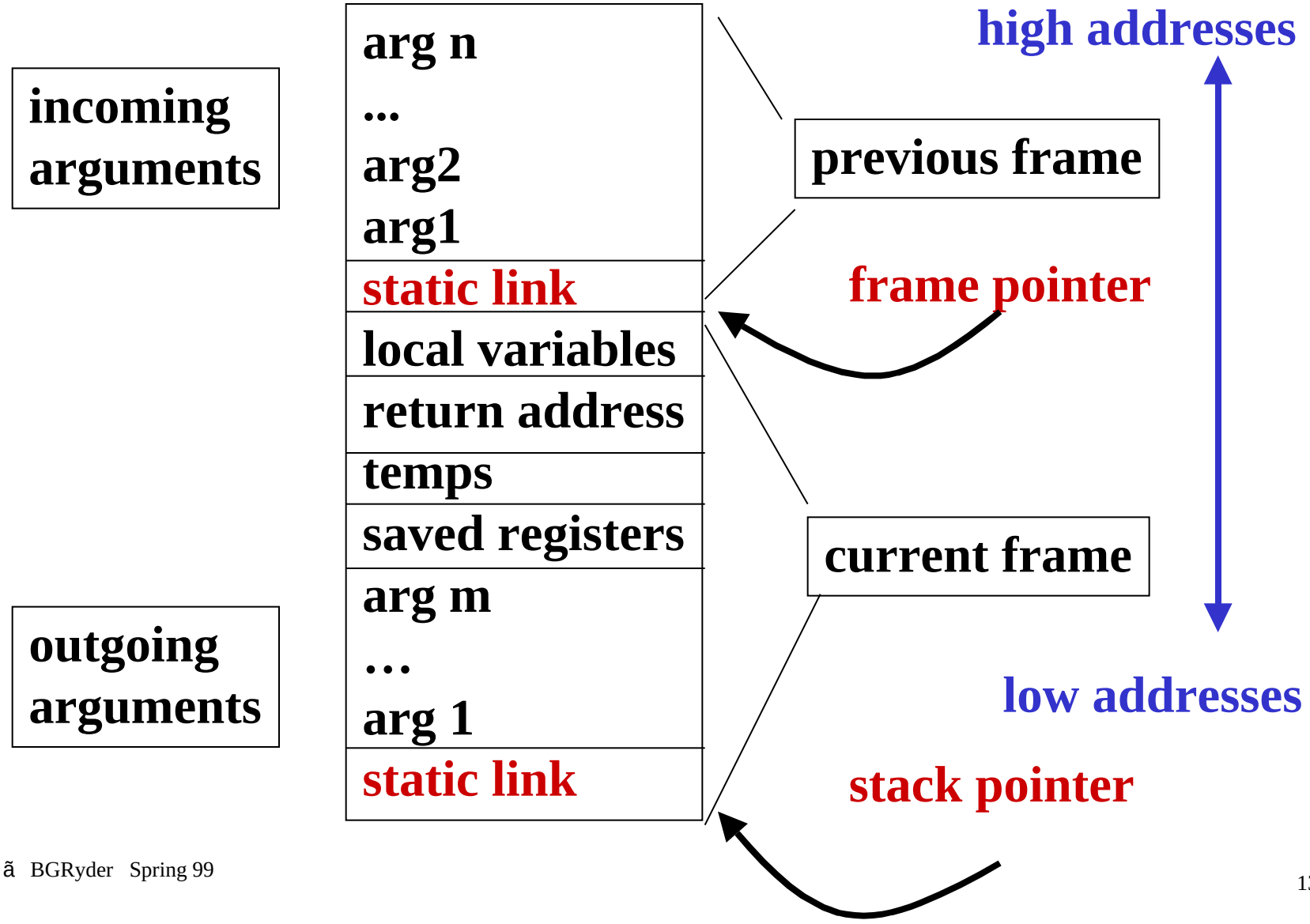
Runtime Stack

- **Frames on stack for each activation which has not yet ended (open function/procedure calls)**
  - **Calls and corresponding returns are LIFO**
  - **When called, push the function's frame onto stack**
  - **On return from the function, remove its frame**
- **Dedicated register always points to *stack_top***
- **Exact frame contents depends on architecture and convention**

Tiger Activation Record

- **Stored in fixed order in frame; *frame pointer* points to frame beginning; fields at offsets**
- **Static link to encompassing scope**
- **Local (non-aggregate) variables**
- **Return address to branch to in code**
- **Temporaries (used in function code)**
- **Saved register contents**
- **Storage for outgoing arguments**

Runtime Stack (Appel, p 133)



Local Non-fixed Size Data

- **In a PL with local dynamic storage allocation, (e.g., non-fixed length parameters A(N))**
  - **Put descriptor for data in fixed size portion of frame**
  - **Later, allocate storage needed at end of frame in variable length portion**

Context Switching - Registers

- **Register contents are saved before context switching into another procedure**
  - ***Callee-save* versus *Caller-save* disciplines**
  - **Contents always saved in frame of saver**
  - **Set by convention of hardware**
  - **Often choose to keep values in registers for efficiency**

Parameter Passing

- *By value*
- By value result (copy in, copy out)
- By result
- *By reference*
- By name (by thunk)

Most common mechanisms in italics. Choice affects how to implement context switching.

Calling Context Switching

- **Conventional to pass first few parameters in specific registers (4-6)**
- **May need to save registers to put the argument values into them; Why practical?**
  - **Most procedures are leaves of calling structure**
  - **Interprocedural register allocation allows parameter passing in different registers**
  - **Needn't ever save dead variables**
  - **Register windows give fresh set of registers to each called function**

Calling Context Switching

- **When a call occurs:**

Callee saves

  - **Caller evaluates actuals and stores them in callee's activation record**
  - **Caller stores code return address and *stack_top* value in callee's activation record**
  - **Caller increments *stack_top* to point within callee's activation record to beginning of local storage**
  - **Callee saves registers into its activation record**
  - **Callee initializes local data and begins execution**

Calling Context Switching

Callee saves

- On return from a call
  - Callee stores its return value in its activation record
  - Callee restores *stack_top* to its former value and restores registers
  - Caller can copy return value into its own activation record

Parameters

- **Some conventions are troublesome**
  - **C requires all parameters be in consecutive storage words**
  - **C allows parameters to have their address taken (dangling pointer problem)**

Return Address

- **Address of code instruction right after the call statement**
- **Put in a designated register by the calling procedure**
- **Return value of a function is also usually returned in a register**

Why ever write to memory?

- Variable is passed by reference
- Variable used in nested procedure
- Value too big to fit in a single register
- Variable is an array
- Register holding variable is needed for another specific purpose
- Too many local+temp variables to fit all in registers