



Principles of Compiler Construction

Prof. Wen-jun LI

School of Computer Science and Engineering

lnslwj@mail.sysu.edu.cn



Lecture 10. Run-Time Environment

1. Storage Management
2. Stack and Activation Record
3. In-Process Communication
4. Heap Management
5. Garbage Collection

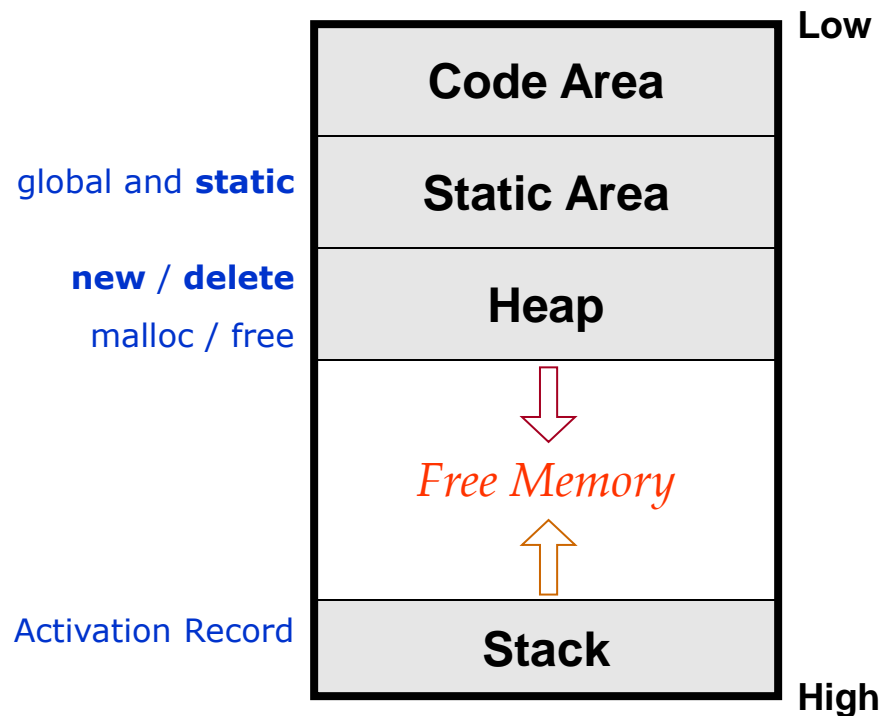
Overview

For example, the subprogram abstraction needs the support by a run-time stack



1. Storage Management

- Typical storage layout and allocation



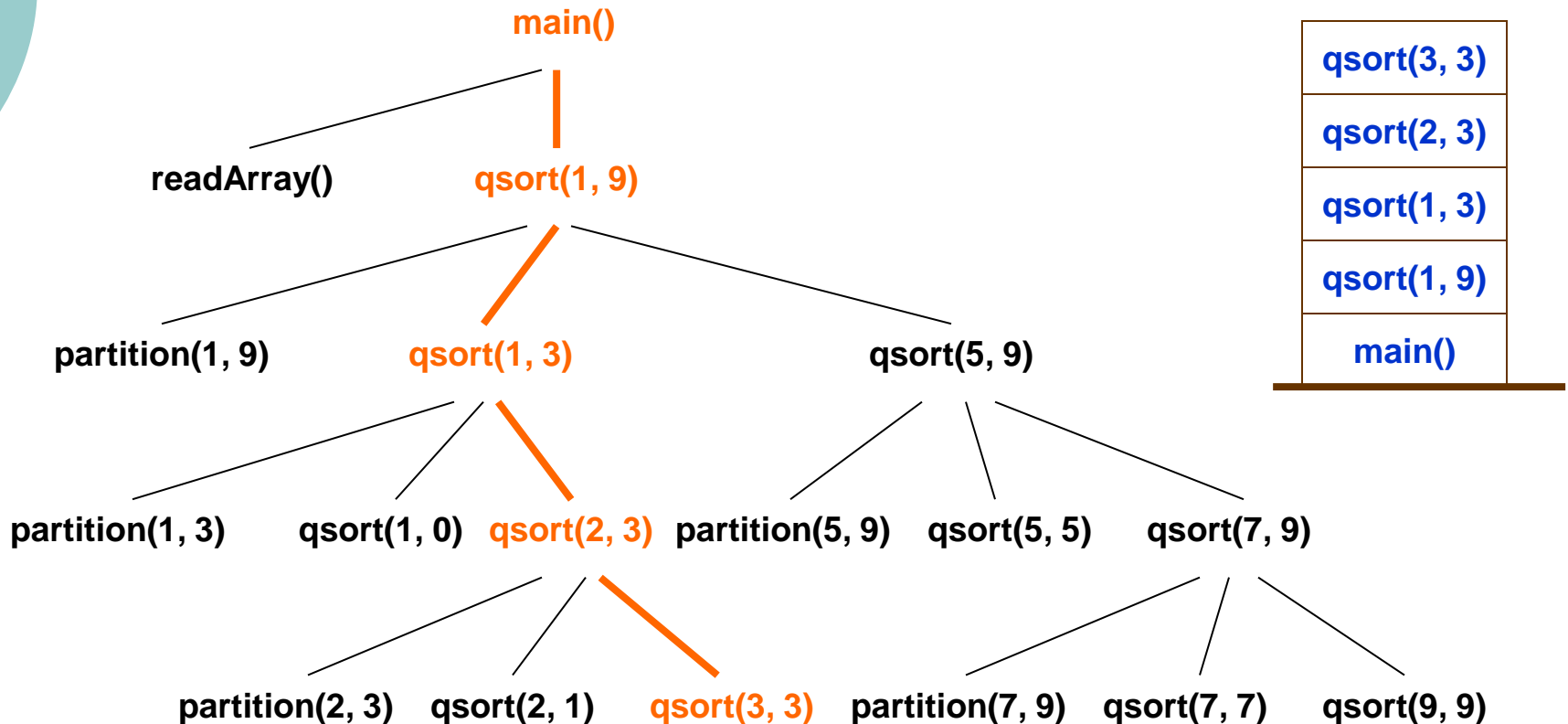


2. Stack and Activation Record

- Support the subprogram abstraction
 - Also called **Control Stack**.
- General design rules
 - **Activation on Demand**

Activation Tree

- Activation tree and run-time stack



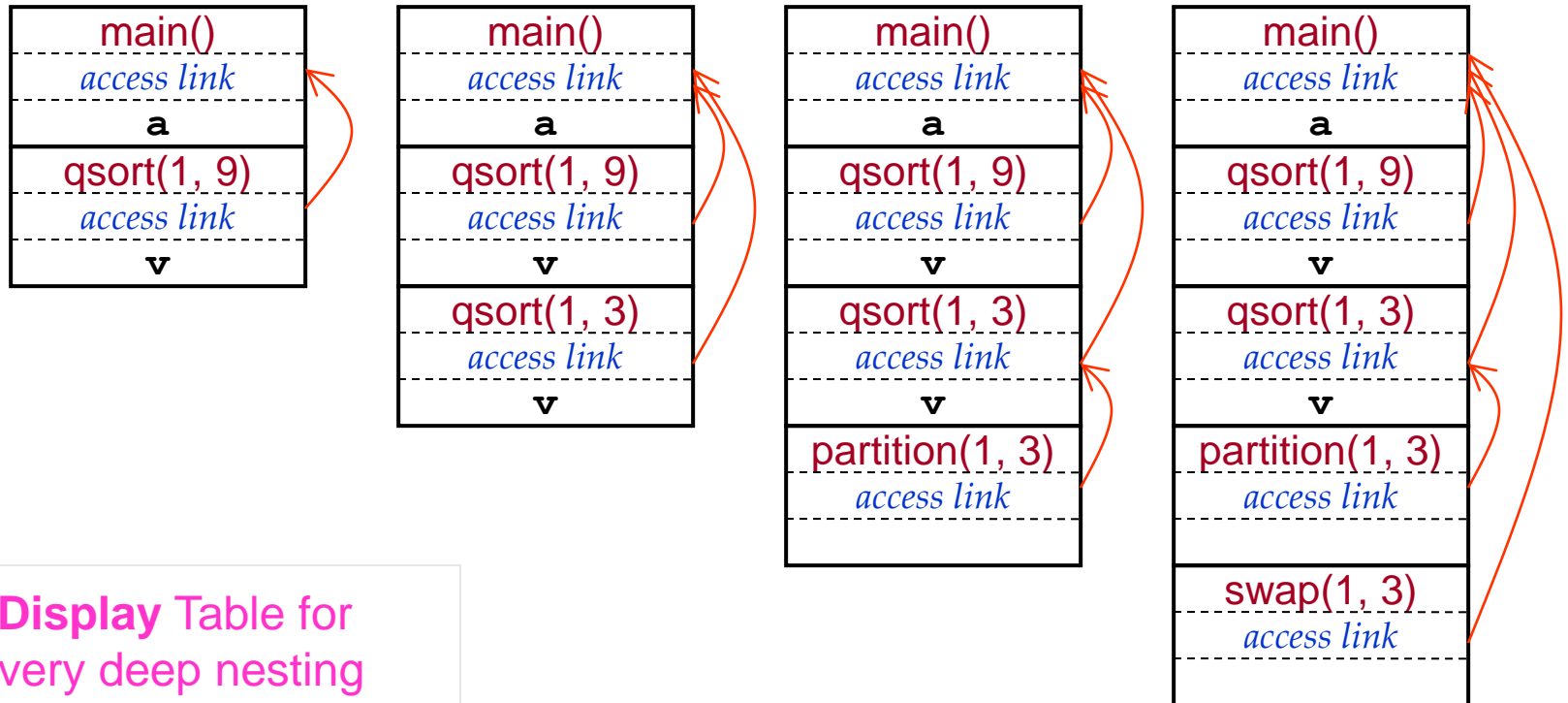
Activation Record (or Frame)

- Typical organization of activation records

Actual Parameters	set by Caller, register preferred
Returned Values	set by Callee, register preferred
Control Link	pointer to Caller's AR
Access Link	pointer to the outer AR
Saved Machine Status	return address & registers
Local Data	user defined
Temporaries	compiler generated

Access Link

- Access link in the quick sort example



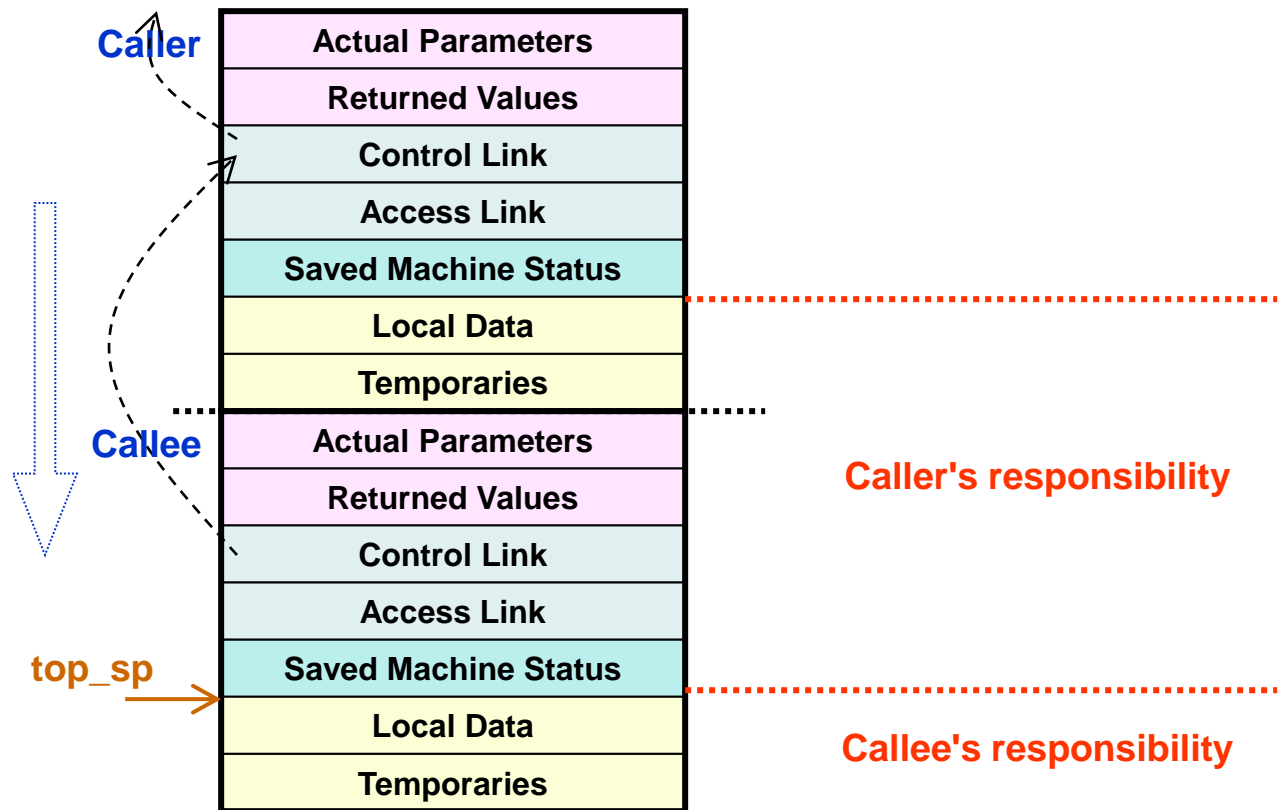
3. In-Process Communication

- Implementation of procedure calls
 - **Calling sequence**
 - Allocate an activation record
 - Enter information into fields
 - ...
 - **Return sequence**
 - Restore the state of the machine
 - Continue the execution of Caller
 - ...

In-Process Communication
vs.
Inter-Process Communication (**IPC**)

Caller vs. Callee

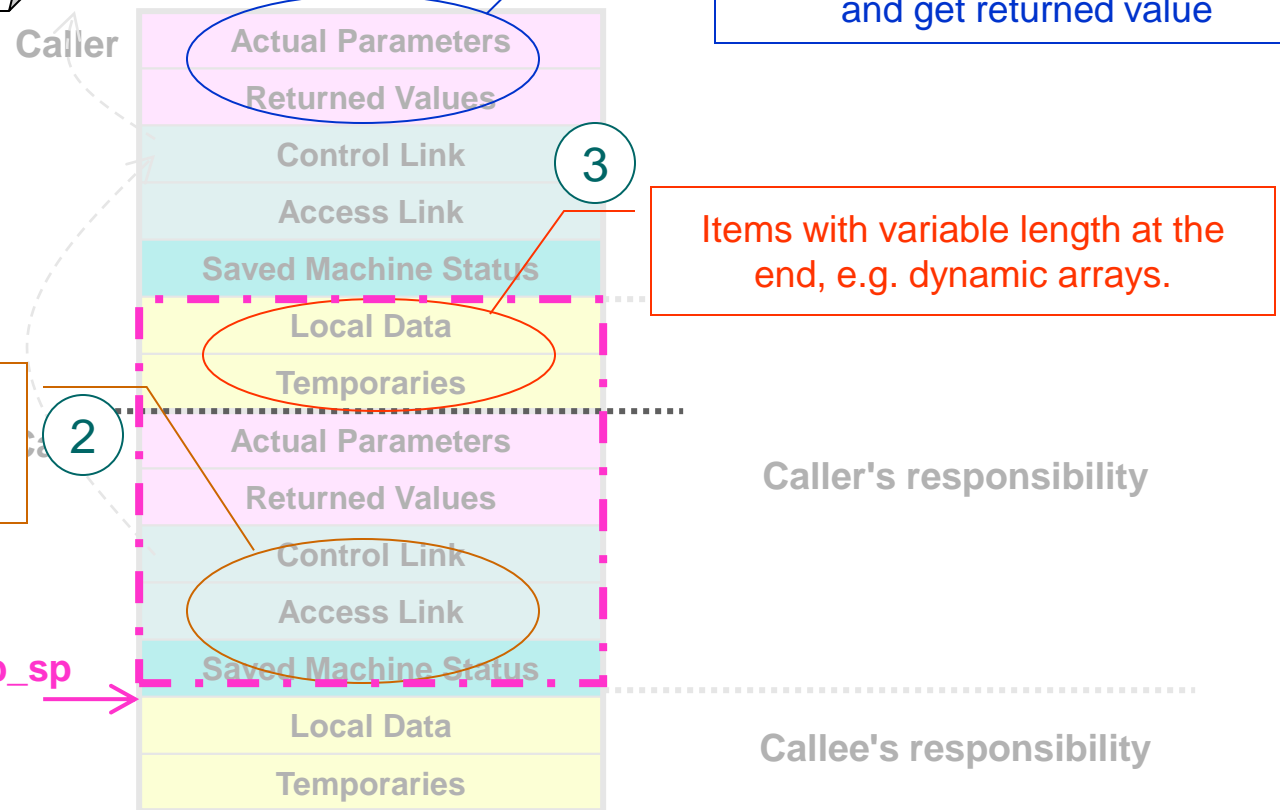
- Implementation of procedure calls



Activation Record Design

Caller does not know local data and temporaries in Callee!

Close to Caller's AR, to facilitate Caller to set actual arguments and get returned value



Calling Sequence

- A typical calling sequence
 1. Caller evaluates the actual arguments.
 2. Caller stores a return address and old **top_sp** to Callee's activation record.
 3. Caller pushes an activation record to the stack (increments **top_sp**).
 4. Callee saves the register values and other status information.
 5. Callee initializes local data and begin the execution.

Return Sequence

- A typical return sequence
 1. Callee sets the return value.
 2. Callee pops the activation record (decrements **top_sp**).
 3. Callee restores the registers.
 4. Callee goes to the return address in the status field.
 5. Caller gets the return value (even **top_sp** has been decremented).



4. Heap Management

- The critical problem of heap management is **fragmentation**
 - Both **time** efficiency and **space** efficiency are considered.
 - How to minimize fragmentation?

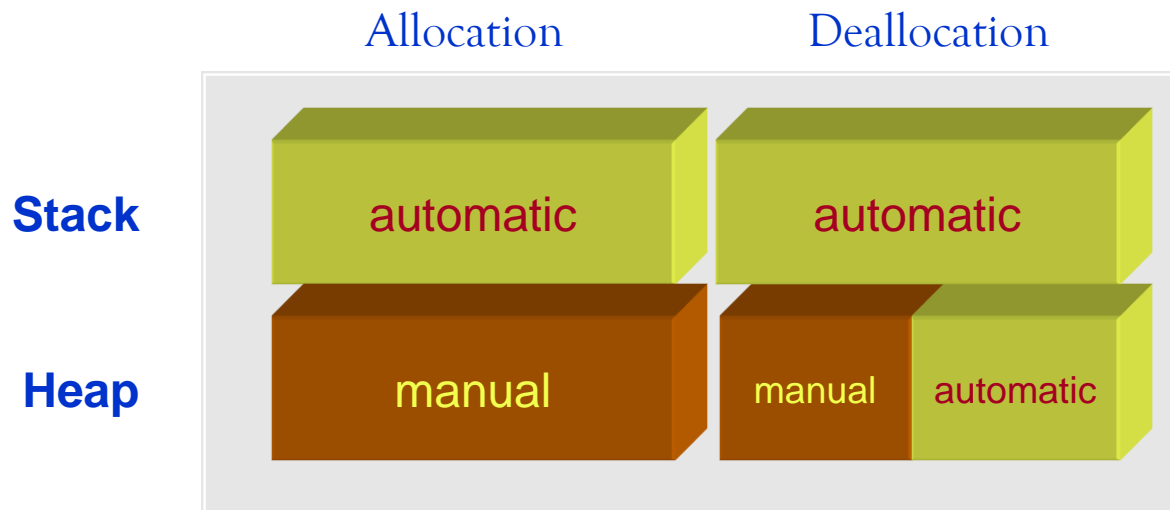


Multithreading

- Multithreading programs share a common heap
 - Of course each program must have its own control stack.
- Communication between procedures in different threads are easier than IPC
 - The shared heap can be utilized in communication.

5. Garbage Collection

- Automatic Memory Management?



Terminology

- Garbage
 - Objects that can not be referenced.
- Memory leaks
 - Maybe the most troublesome bugs.
 - Does Java have memory leaks?
- Profiler (Rational Purify)
 - **JVMPI**: JVM Profiler Interface.
 - Since JDK 1.2 (Java 2).
- Garbage collection
 - Pros and cons?
 - High-Level Abstraction in Languages.

Great expectation to
JVMGCI

Performance Metrics

- Overall Execution Time
- Space Usage
 - Avoid fragmentation
- Pause Time
 - Critical to real-time applications
- Program Locality
 - The program spend most of time executing a relative small code fraction and touching only a small data fraction. (90% time on 10%code)
 - Locality facilitate the utilization of memory hierarchy (register, cache, memory, disk, etc.).
 - Garbage collection can improve both temporal locality and spatial locality.

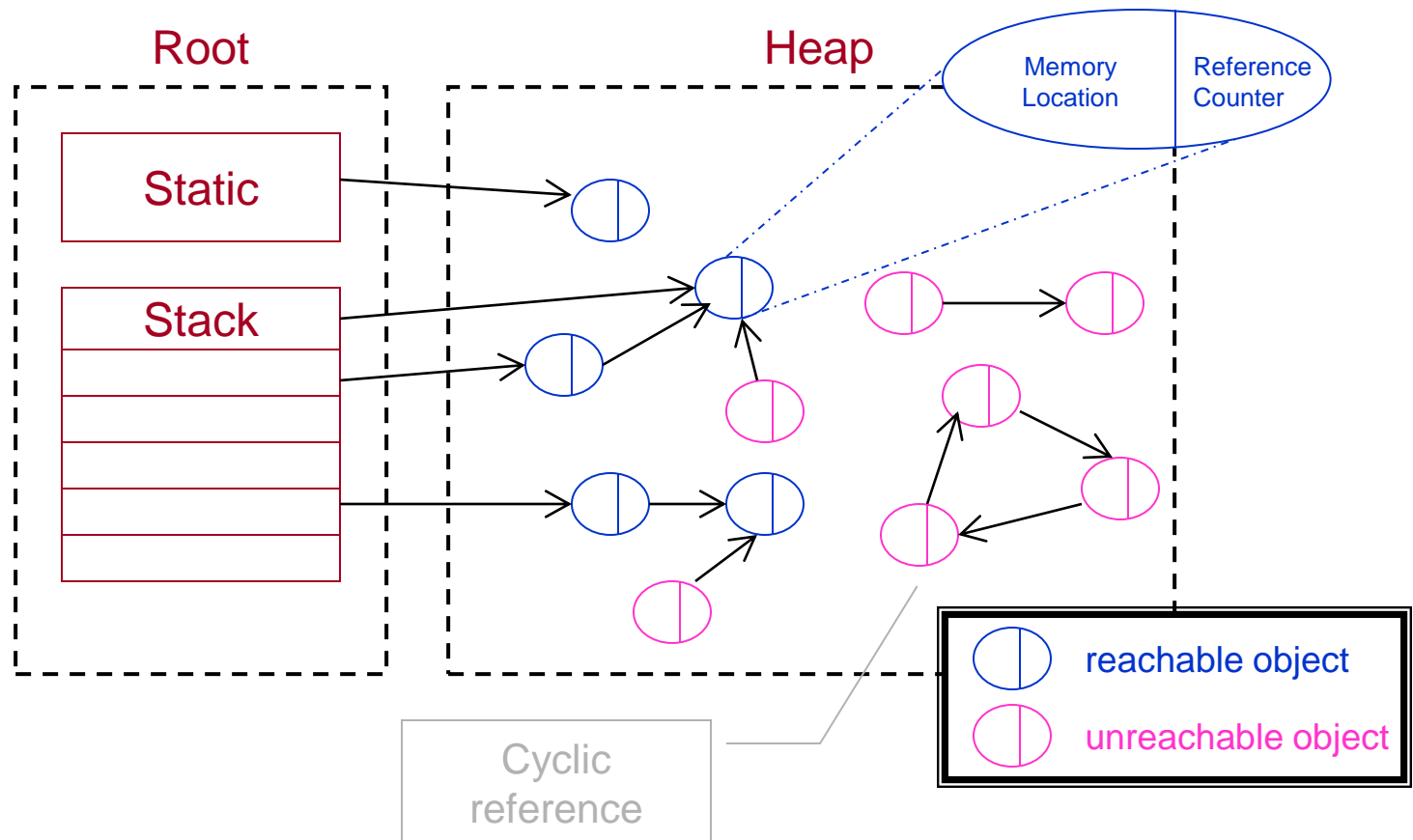


Garbage Collection Algorithms

- Reference counting (immediately)
- Trace-based algorithms (periodically)
 - Basic Mark-and-Sweep
 - Baker's Mark-and-Sweep
 - Basic Mark-and-Compact
 - Cheney's Copying Collector
- Short-pause algorithms
 - Incremental garbage collection
 - Incremental reachability analysis
 - Partial collection
 - Generational garbage collection

Reference Counting

- counter++ vs. counter--



Reference Counting (cont')

- Pros

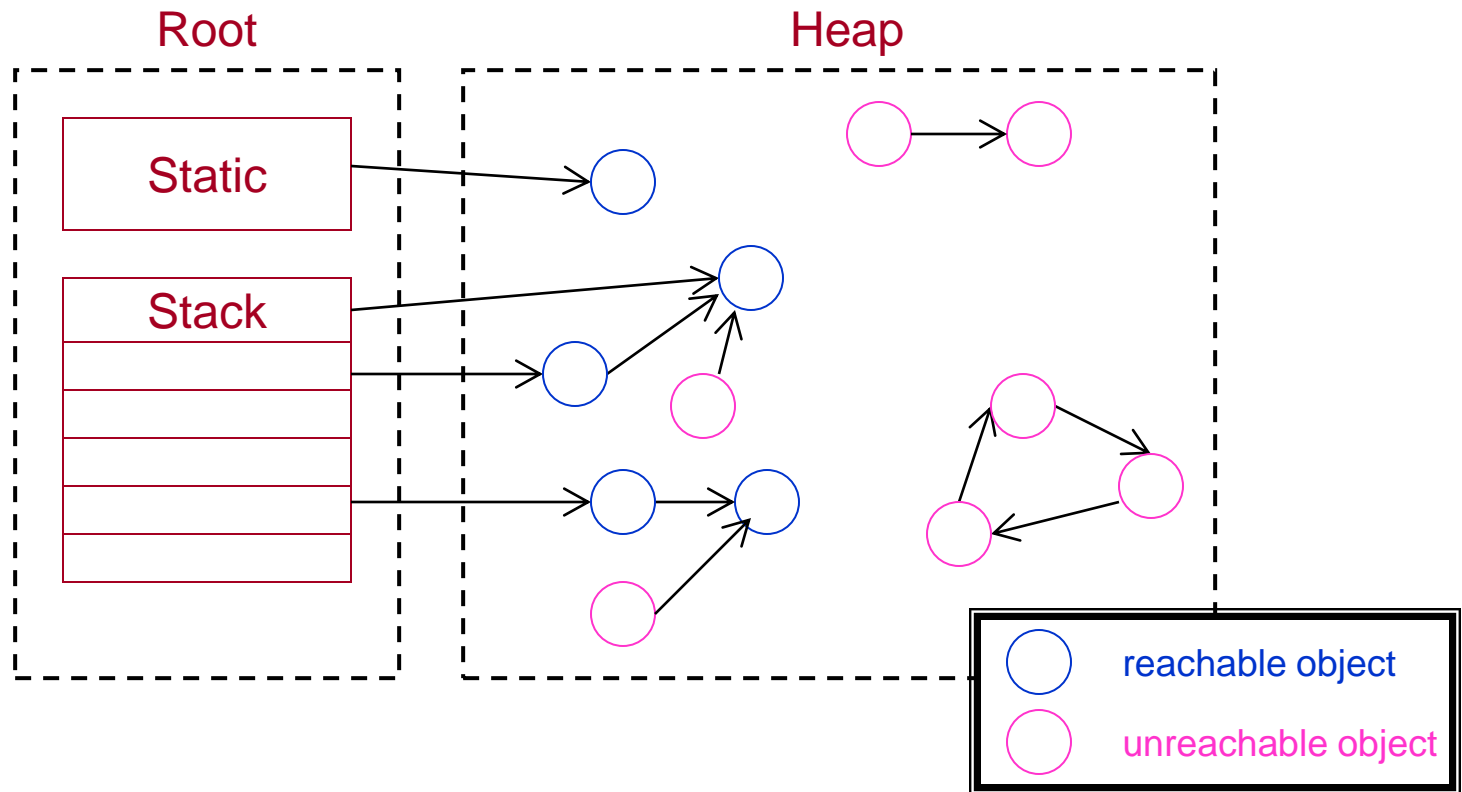
- Simple
- Incremental collection
- Immediate collection

- Cons

- Can not collect unreachable but cyclic data structures.
- Additional operations lead to expensive overhead.
 - Depends on computation, not only on number of objects.

Mark-and-Sweep

- 2 steps: trace (mark), then sweep.



Mark-and-Sweep (cont')

- Pros

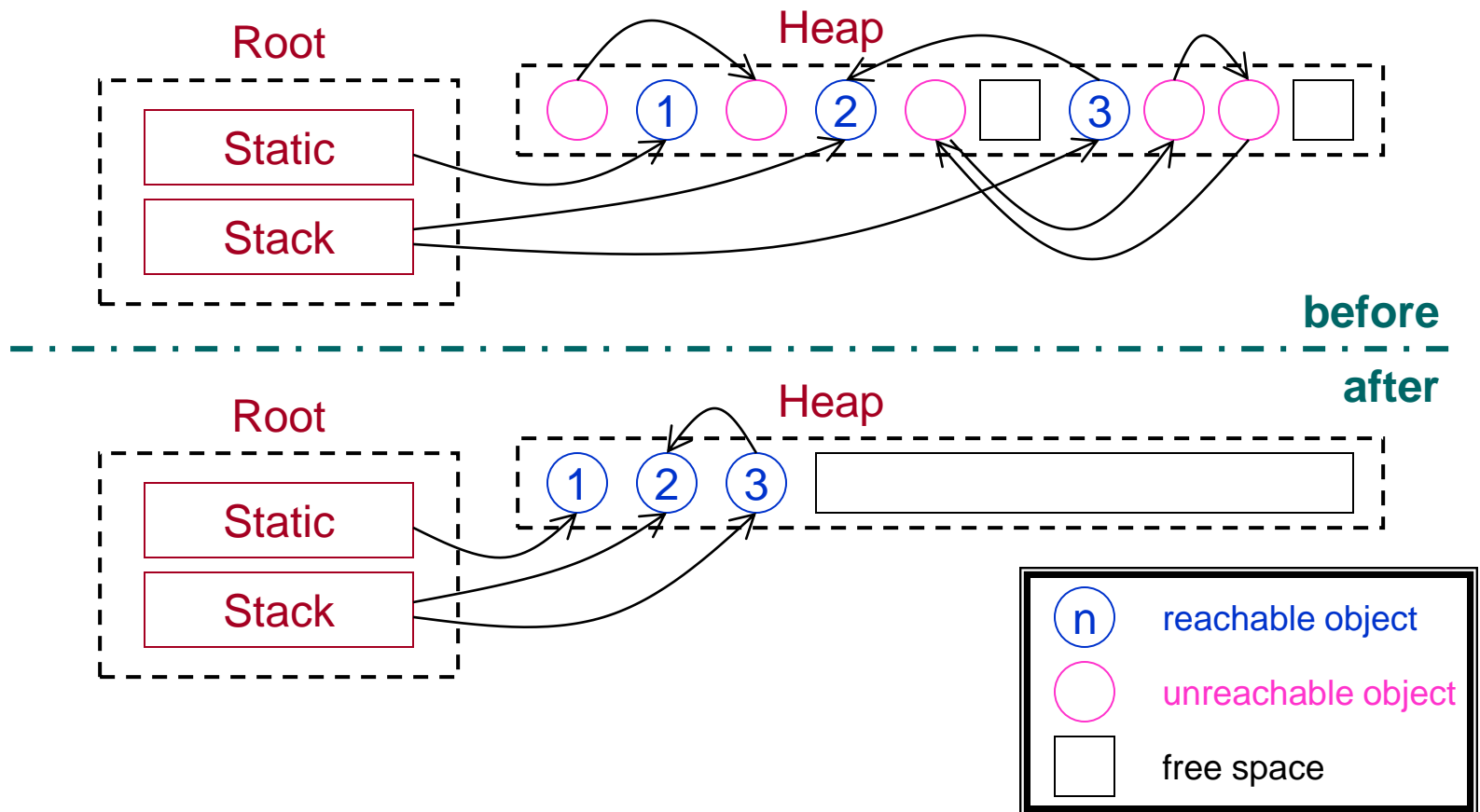
- High efficiency if little garbage exist.
- Be able to collect cyclic references.

- Cons

- Low efficiency with large amount of garbage
 - Improvement by H. Baker (1992)
 - Keep a list of all allocated objects.
- Pause of the mutator (program) make it difficult to apply to real-time applications.
- Leads to fragmentation in the heap.

Mark-and-Compact

- Before vs. after

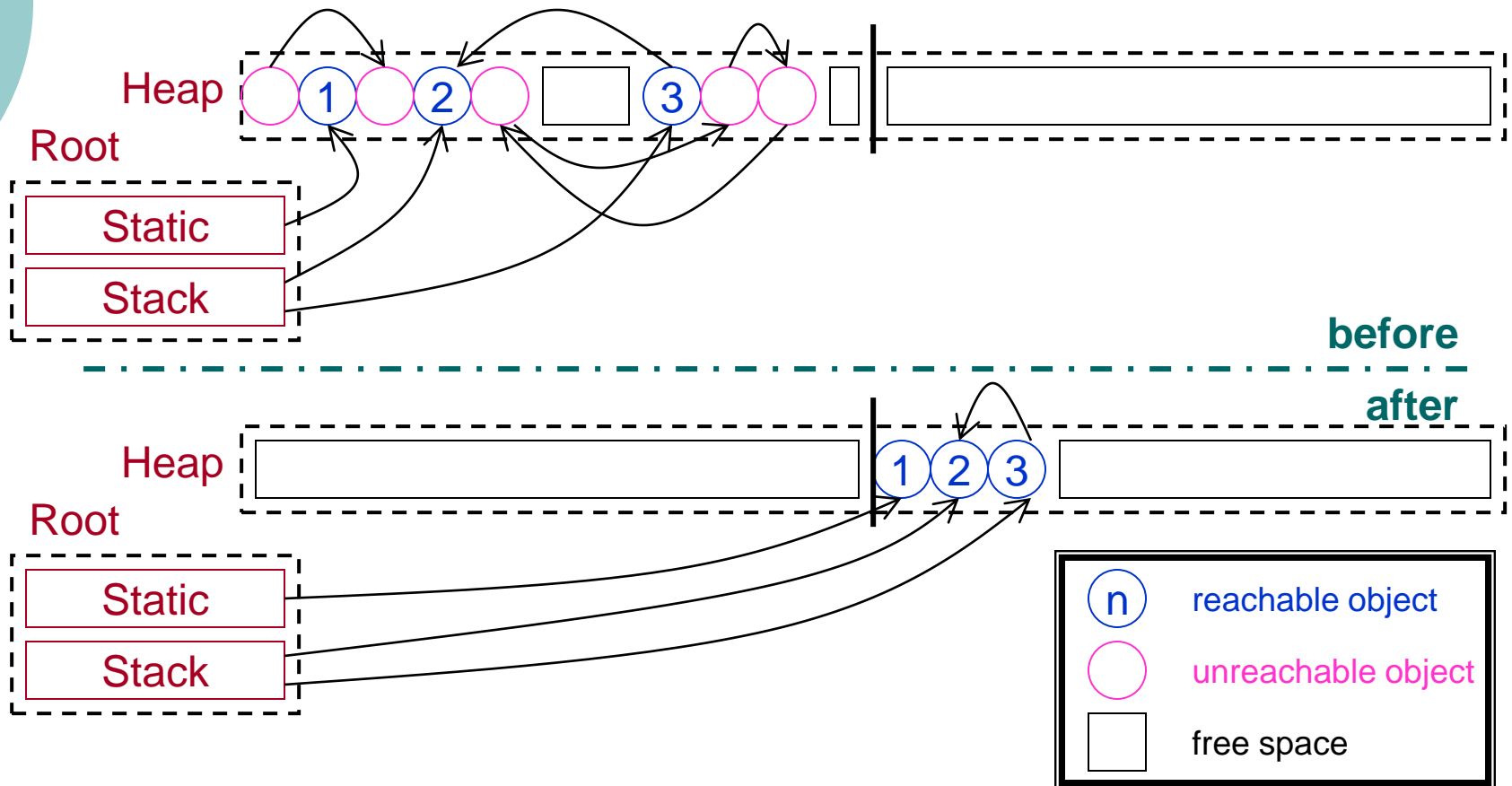


Mark-and-Compact (cont')

- 3 steps
 1. Marking phase: similar to Mark-and-Sweep.
 2. For each reachable object, computes a new address from the low end of the heap.
 3. Copies objects to new locations and updates all references in the root area and objects.
- Pros and cons
 - Pro: avoids fragmentation.
 - So as to improve the space usage and program locality.
 - Con: step 3 (copying in the compacting) is expensive.
 - Updating references is simple and fast.

Copying Collector

○ Before vs. after



Copying Collector (cont')

- Heap is partitioned into 2 semispaces
 - Reachable objects are moved as soon as they are discovered.
 - Does not touch any unreachable objects.
- Pros
 - More efficient and popular.
- Cons
 - Only a half of heap memory can be utilized.

Comparing Costs

- Mark-and-Sweep
 - Number of chunks in the heap
- Baker's Improvement
 - Number of reached objects
- Mark-and-Compact
 - Number of chunks in the heap +
 - **Total size of the reached objects**
- Copying Collector
 - **Total size of the reached objects**



Generational Garbage Collection

- Motivation

- Difference between handling short-lived and long-lived objects.

- Approach

- Objects are divided into multiple generations based on some criteria.
 - Usually related to the age of the objects.
 - GC can be done at different time intervals and even using different techniques, based on the generations.



Implementation in JVM

- Use a two-generation (young and old) approach
- Young generation: copying collector
 - Newly created objects tend to die young.
- Old generation: Mark-and-Compact
 - A good mix of technologies for performance



Exercise 10.1

- Surf the Internet and write a short paper to compare GC on Java platform with GC on Microsoft .Net platform.

Further Reading

- Dragon Book, 2nd Edition (DBv2)
 - Comprehensive Reading:
 - Section 7.2-7.3 on control stack and activation records.
 - Section 7.4 on heap management.
 - Section 7.5-7.7 on garbage collection.
 - Skip Reading:
 - Section 7.1 on introduction to storage management.
 - Section 7.8 on advanced topics in garbage collection.
 - More on garbage collection: R. Jones and R. Lins. **Garbage Collection: Algorithms for Automatic Dynamic Memory Management.** Wiley, 1996, ISBN 0-471-94148-4

Enjoy the Course!

