Principles of Compiler Construction

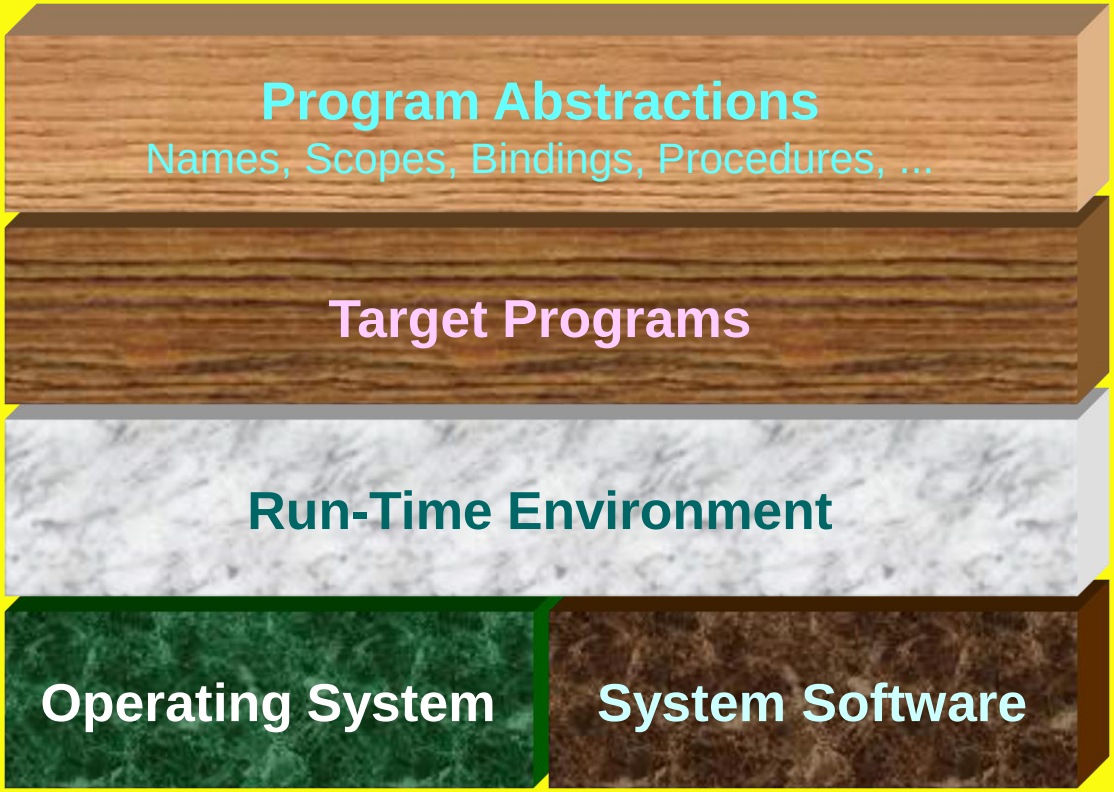
Lecturer: CHANG HUIYOU

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



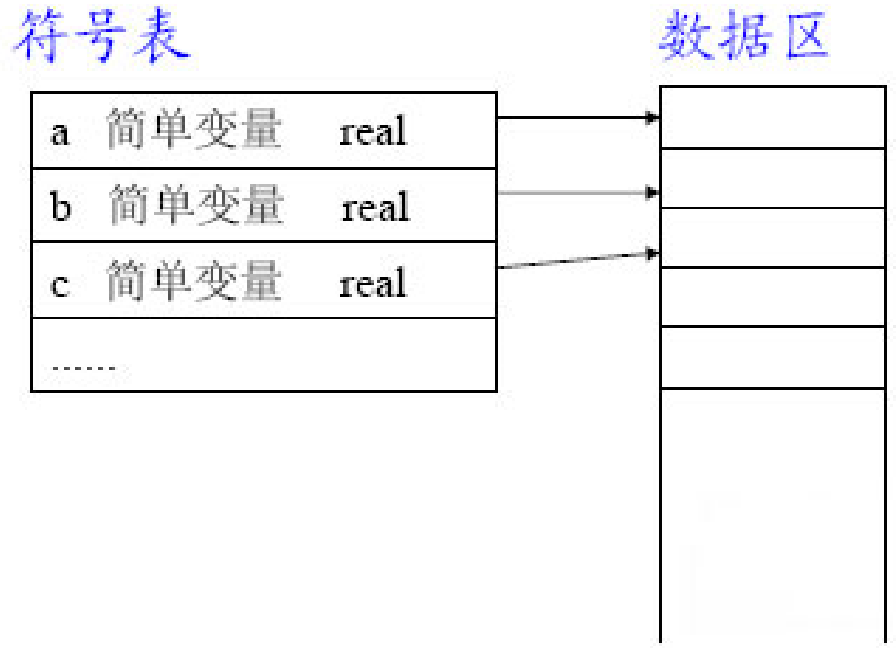
Overview

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

例: real a, b, c ;
...
a := b*c ;

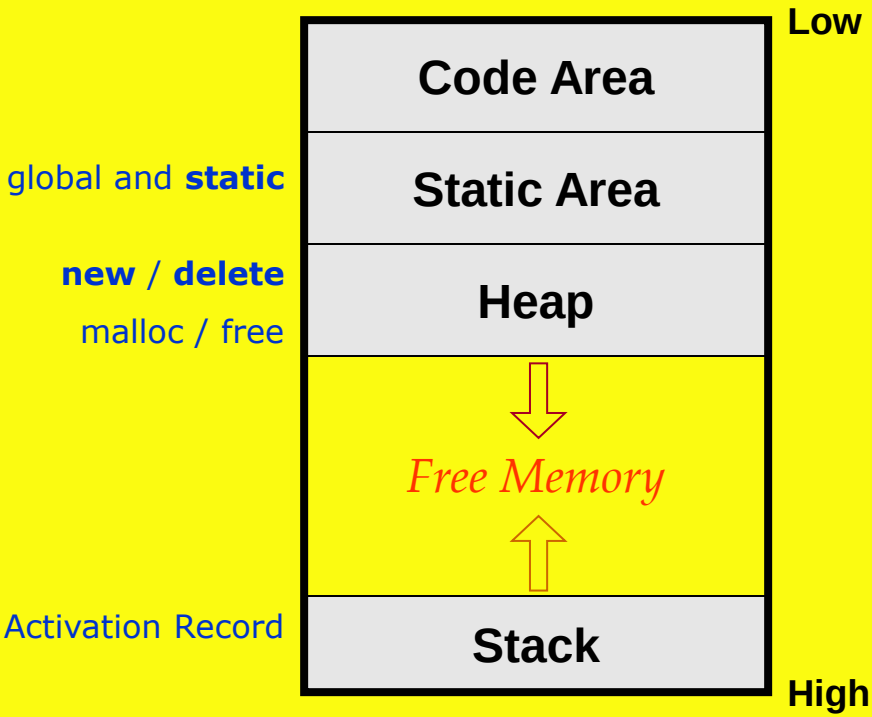
⇩

取b
* c
送a



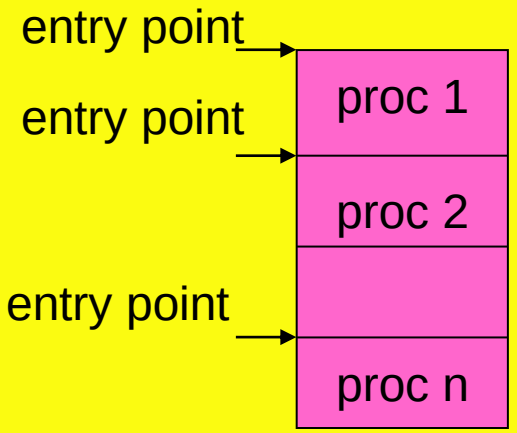
1. Storage Management

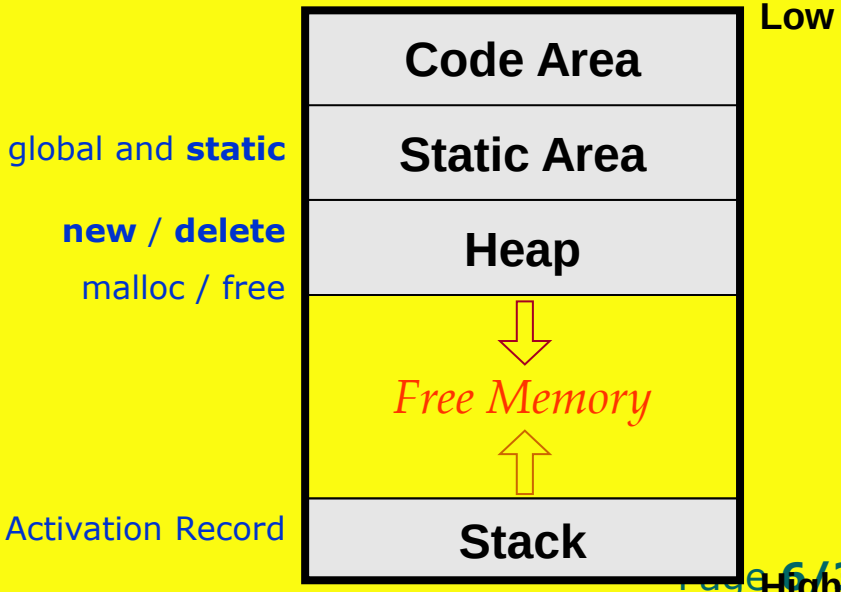
- Typical storage layout and allocation



Code Area

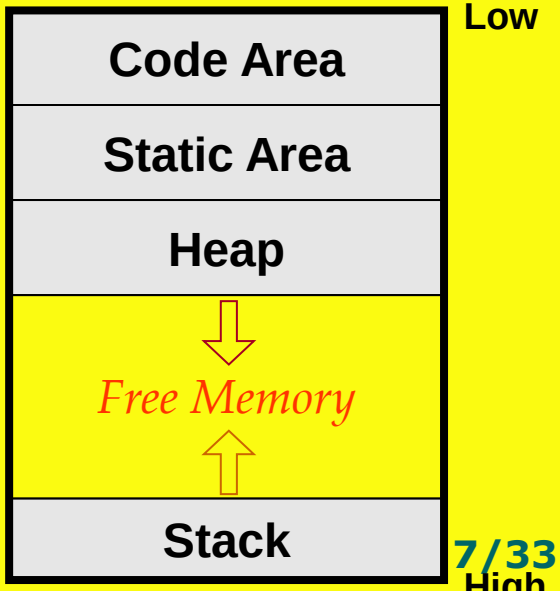
- Addresses in code area are static (i.e. no change during execution) for most programming language.
- Addresses are known at compile time.





Data Area

- Addresses in data area are static for some data and dynamic for others.
  - Static data are located in static area.
  - Dynamic data are located in stack or heap.
    - Stack (LIFO allocation) for procedure activation record, etc.
    - Heap for user allocated memory, etc.



Registers

- General-purpose registers
  - Used for calculation
- Special purpose registers
  - Program counter (pc)
  - Stack pointer (sp)
  - Frame pointer (fp)
  - Argument pointer (ap)

Lecture 10. Run-Time Environment

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

Pascal

```
program sort(input, output);
    var a:array[0..10] of integer;
    procedure readarry;
        var i: integer;
        begin
                for i:=1 to 9 do read(a[i])
        end;
    function partition(y,z:integer):integer;
        var i,j,x,v : integer;
        begin     ……
        end;
    procedure quicksort(m,n:integer)
        var i:integer;
        begin
                if(n>m) then begin
                        i:=partition(m,n);
                        quicksort(m,i-1);
                        quicksort(i+1,n)
                end
        end
    begin
        a[0] := -9999;a[10]:=9999;
        readarray;
        quicksort(1,9)
    end
```

Calling Sequence

- Sequence of operations that must be done for procedure calls
  - Call sequence
    - Sequence of operations performed during procedure calls
      - Find the arguments and pass them to the callee.
      - Save the caller environment, i.e. local variables in activation records, return address.
      - Create the callee environment, i.e. local variables in activation records, callee's entry point.
  - Return sequence
    - Sequence of operations performed when return from procedure calls
      - Find the arguments and pass them back to the caller.
      - Free the callee environment.
      - Restore the caller environment, including PC.

Static runtime environments

- Static data
  - Both local and global variables are allocated once at the beginning and deallocated at program termination
  - Fixed address
- No dynamic allocation
- No recursive call
  - Procedure calls are allowed, but no recursion.
  - One activation record for each procedure, allocated statically
- Example:FORTRAN 77

Memory Organization for Static Runtime Environment

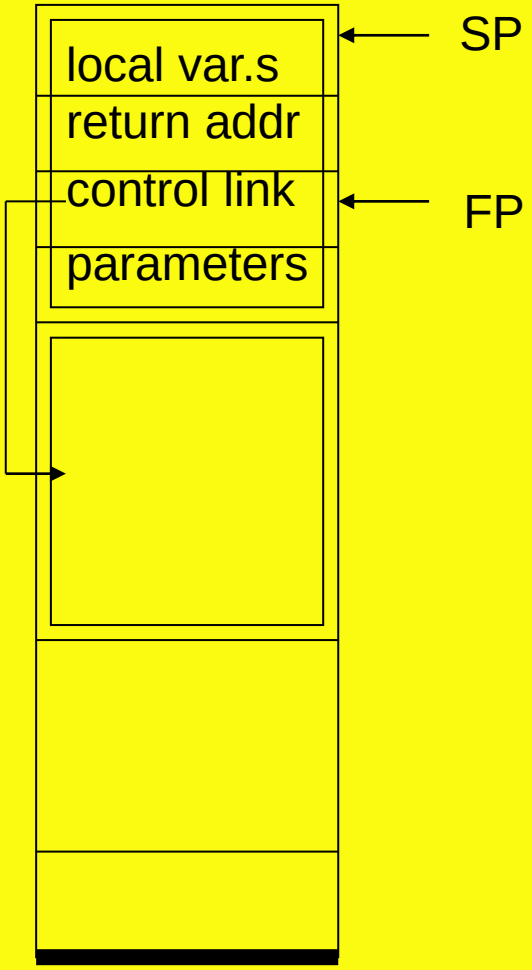
```
PROGRAM TEST
COMMON MAX
INTEGER MAX
REAL TAB(10), TEMP
…
QMEAN(TAB, 3, TEMP)
…
END

SUBROUTINE QMEAN(A, SIZE, MEAN)
COMMON MAX
INTEGER MAX, SIZE
REAL A(SIZE), MEAN, TEMP
INTEGER K
…
END
```

Stack-based runtime environments

- Handle recursive calls
- Activation records are allocated in stack, called rumtime stack or call stack
- One procedure can have more than one activation records in the stack at one time
- Call sequence is more complex than the sequence in static environment

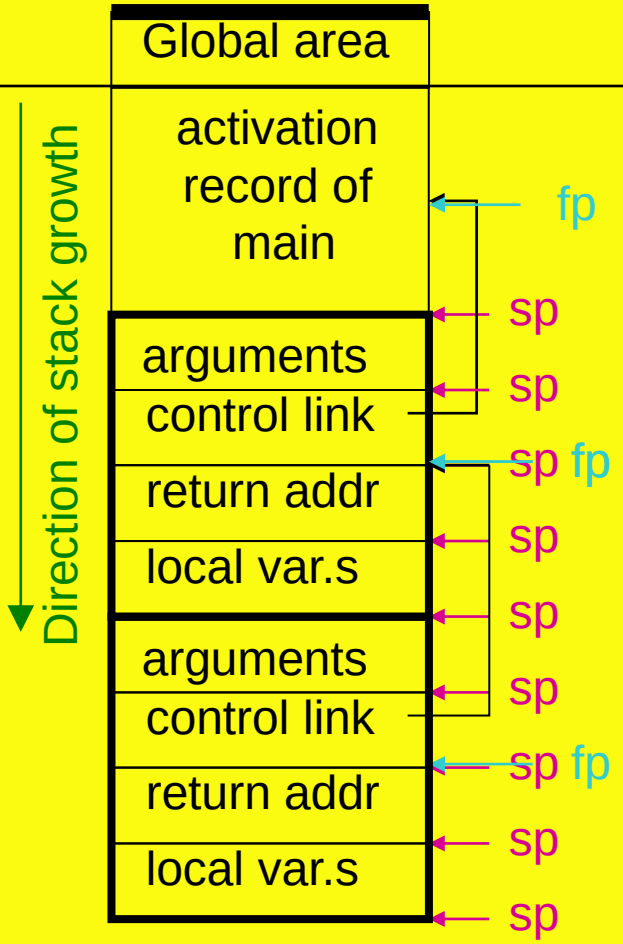
Stack-based Environments Without Local Procedures

- Maintain pointer to the current activation record
  - Store *frame pointer* in an *fp* register
- Record the link from an activation record to the previous record
  - In each activation record, a link from an activation record to the activation record of the caller, called a *dynamic link* or *control link* is stored.
- Sometimes, the area for parameters and local variables need to be identified.
- A stack pointer is maintained in an sp register.



Call Sequence in Stack-based Environments Without Local Procedures

- Calling sequence
  - Push arguments
  - Push fp as control link
  - Copy sp to fp
  - Store return address
  - Jump to callee
  - Reserve space for local variables
- Return sequence
  - Copy fp to sp
  - Load control link into fp
  - Jump to return address
  - Change sp to pop arguments

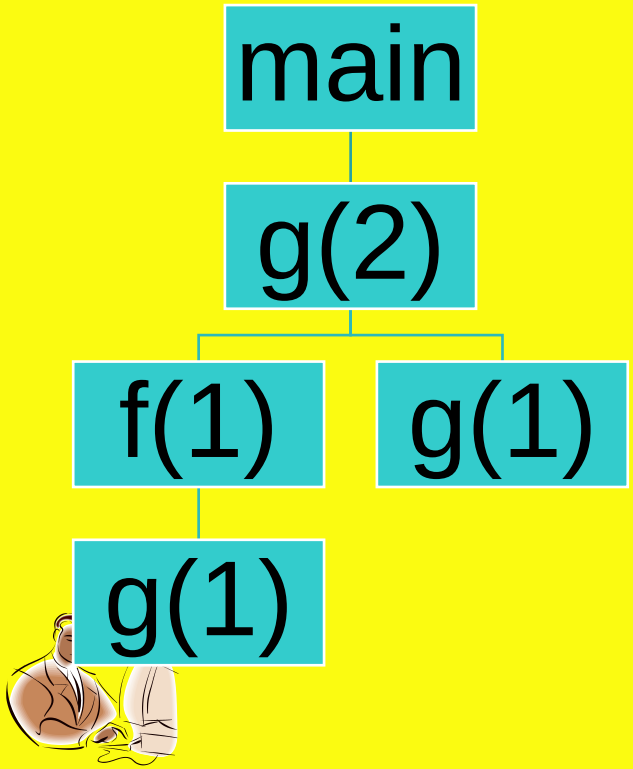


Calling sequence

return addr)

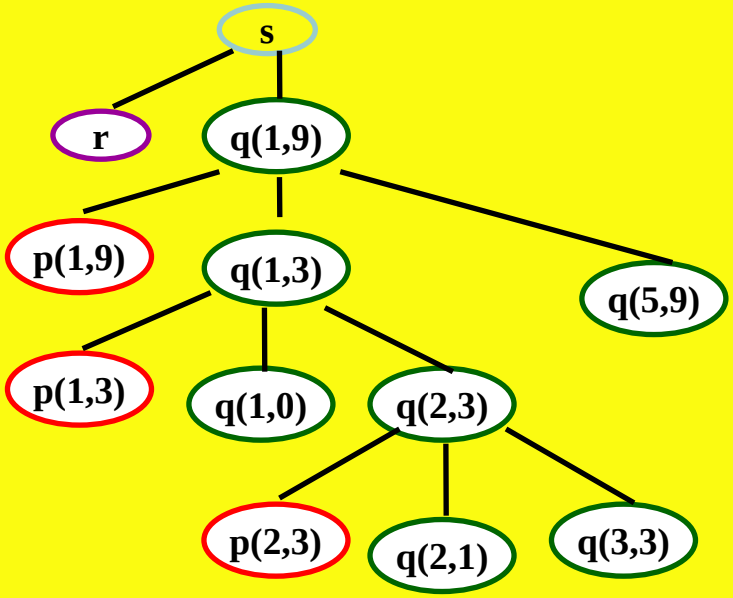
Activation Tree

```
main()
{  …;
   g(x);
   return(0);   }

void g(int m)
{  …
   f(y);…
   g(y);
   …
}

void f(int n)
{  g(n);
   …
}
```

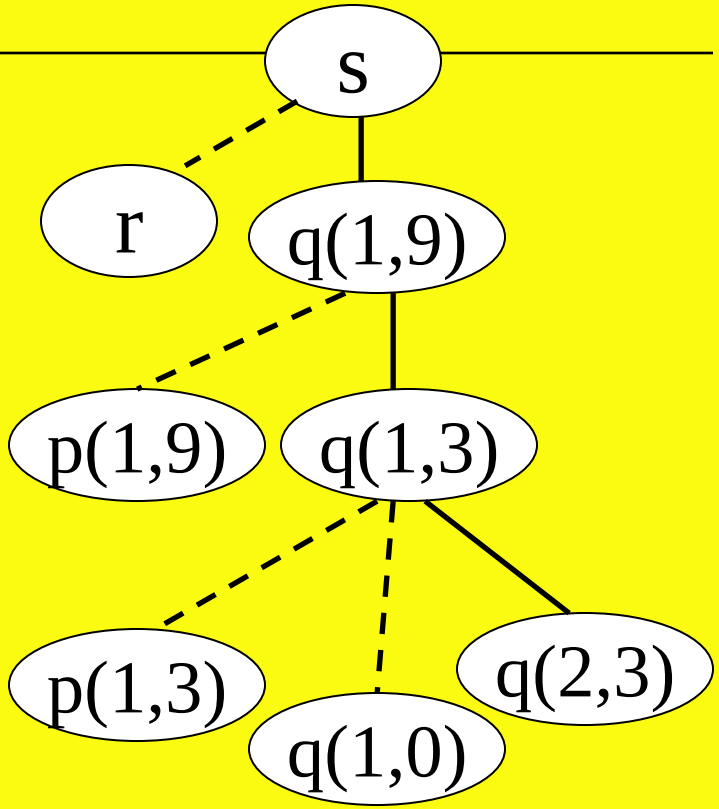


Global area
activation record for main
activation record for g(2)
activation record for f(1)
activation record for g(1)

```
program sort(input, output);
    var a:array[0..10] of integer;
    procedure readarry;
        var i: integer;
        begin
                for i:=1 to 9 do read(a[i])
        end;
    function partition(y,z:integer):integer;
        var i,j,x,v : integer;
        begin     ……
        end;
    procedure quicksort(m,n:integer)
        var i:integer;
        begin
                if(n>m) then begin
                        i:=partition(m,n);
                        quicksort(m,i-1);
                        quicksort(i+1,n)
                end
        end
    begin
        a[0] := -9999;a[10]:=9999;
        readarray;
        quicksort(1,9)
    end
```

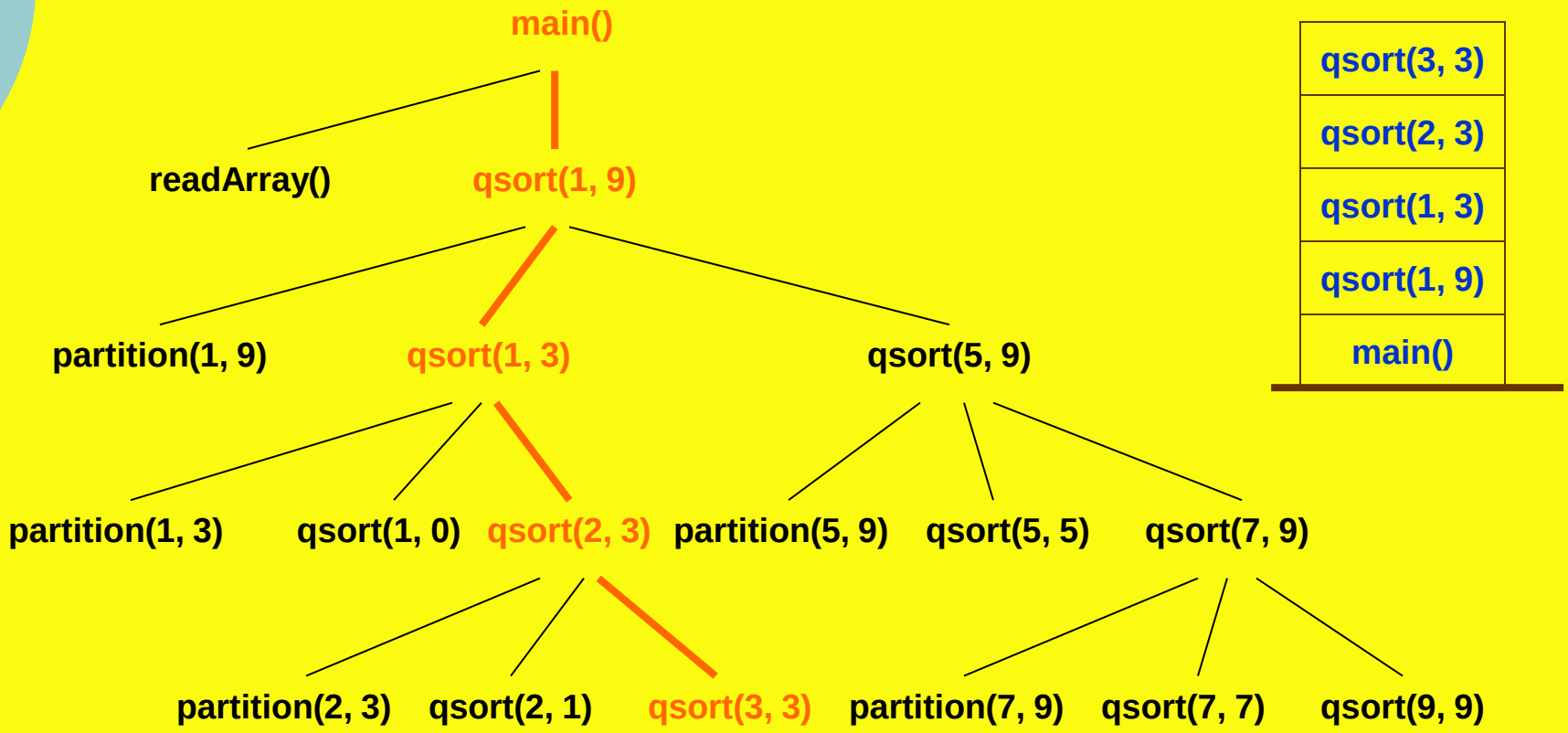


s
S r
S q(1.9)
S q(1.9) p(1,9)
S q(1.9) q(1,3)
S q(1.9) q(1,3) p(1,3)
S q(1.9) q(1,3) q(1,0)
S q(1.9) q(1,3) q(2,3)



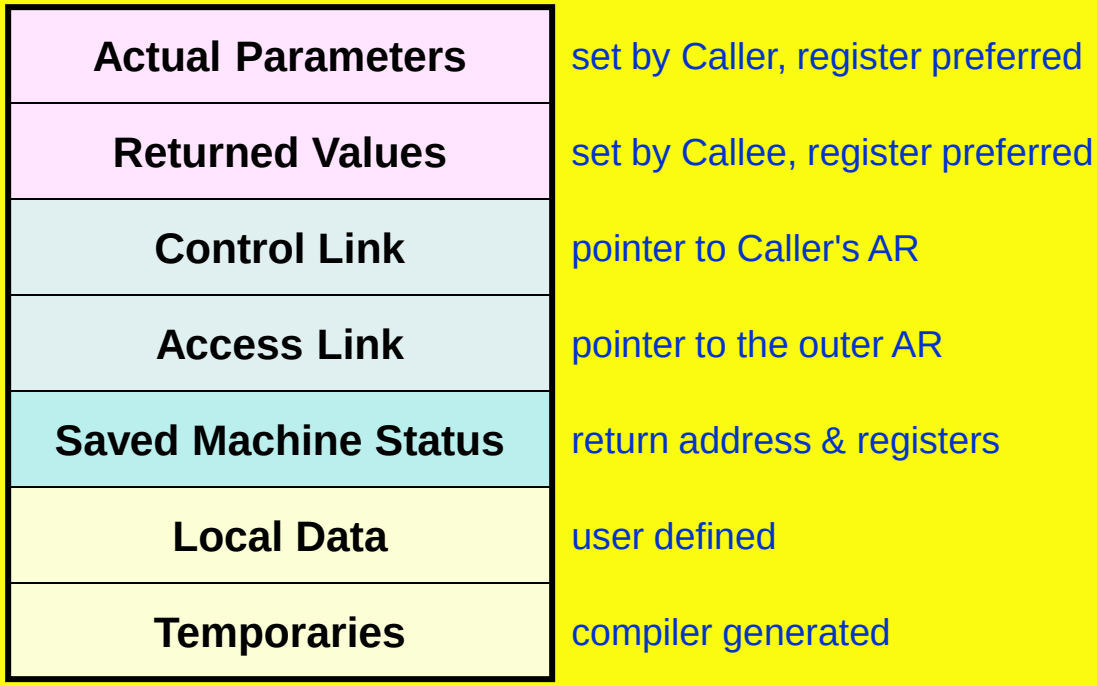
Activation Tree

- Activation tree and run-time stack



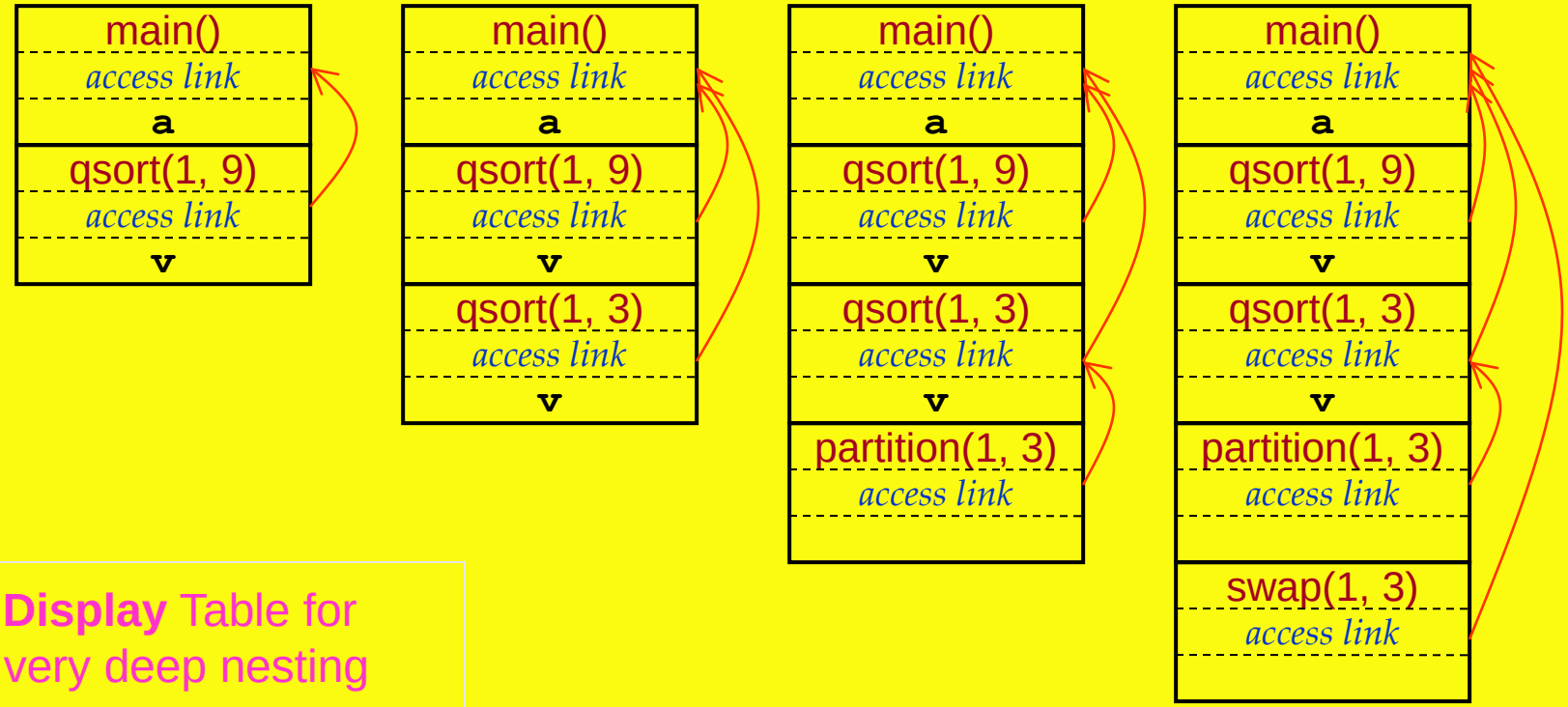
Activation Record (or Frame)

- Typical organization of activation records



Access Link

- Access link in the quick sort example



Display Table for very deep nesting

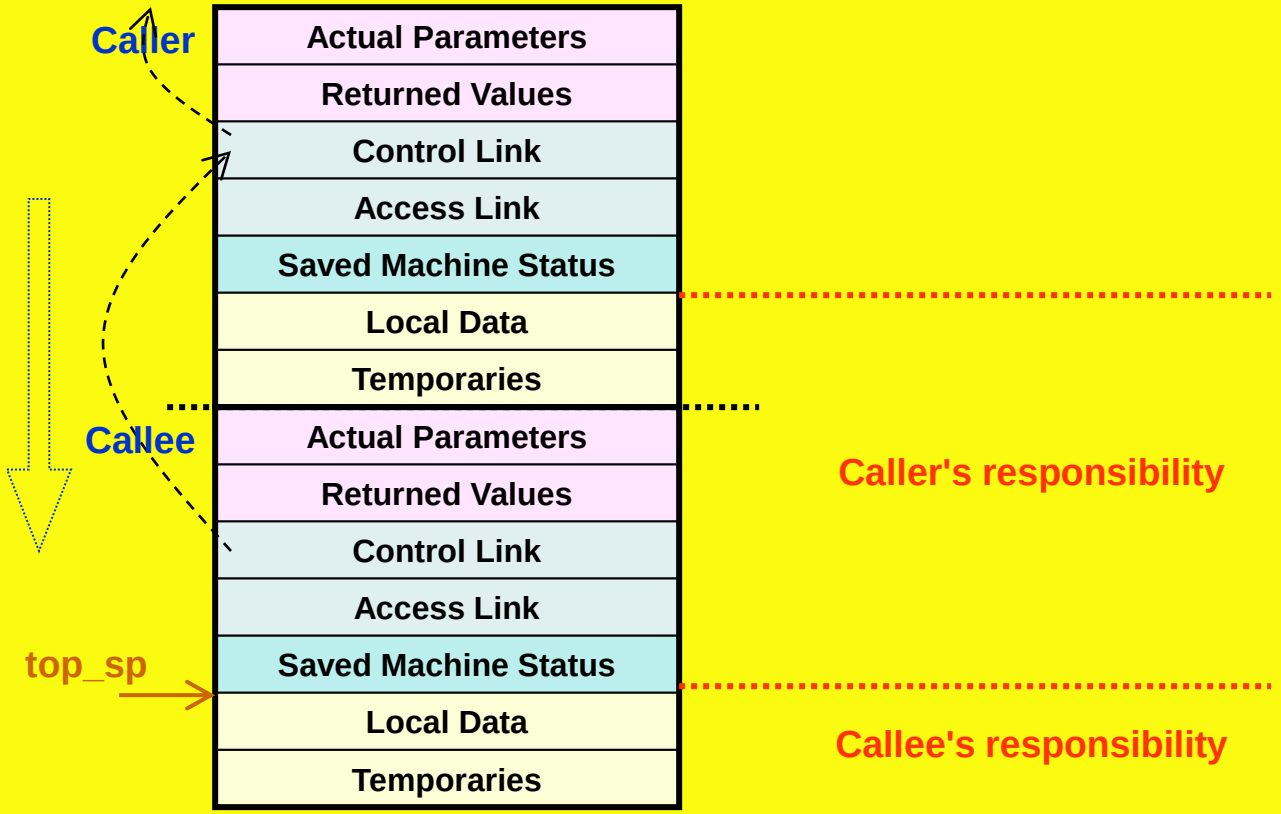
3. In-Process Communication

- 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**)

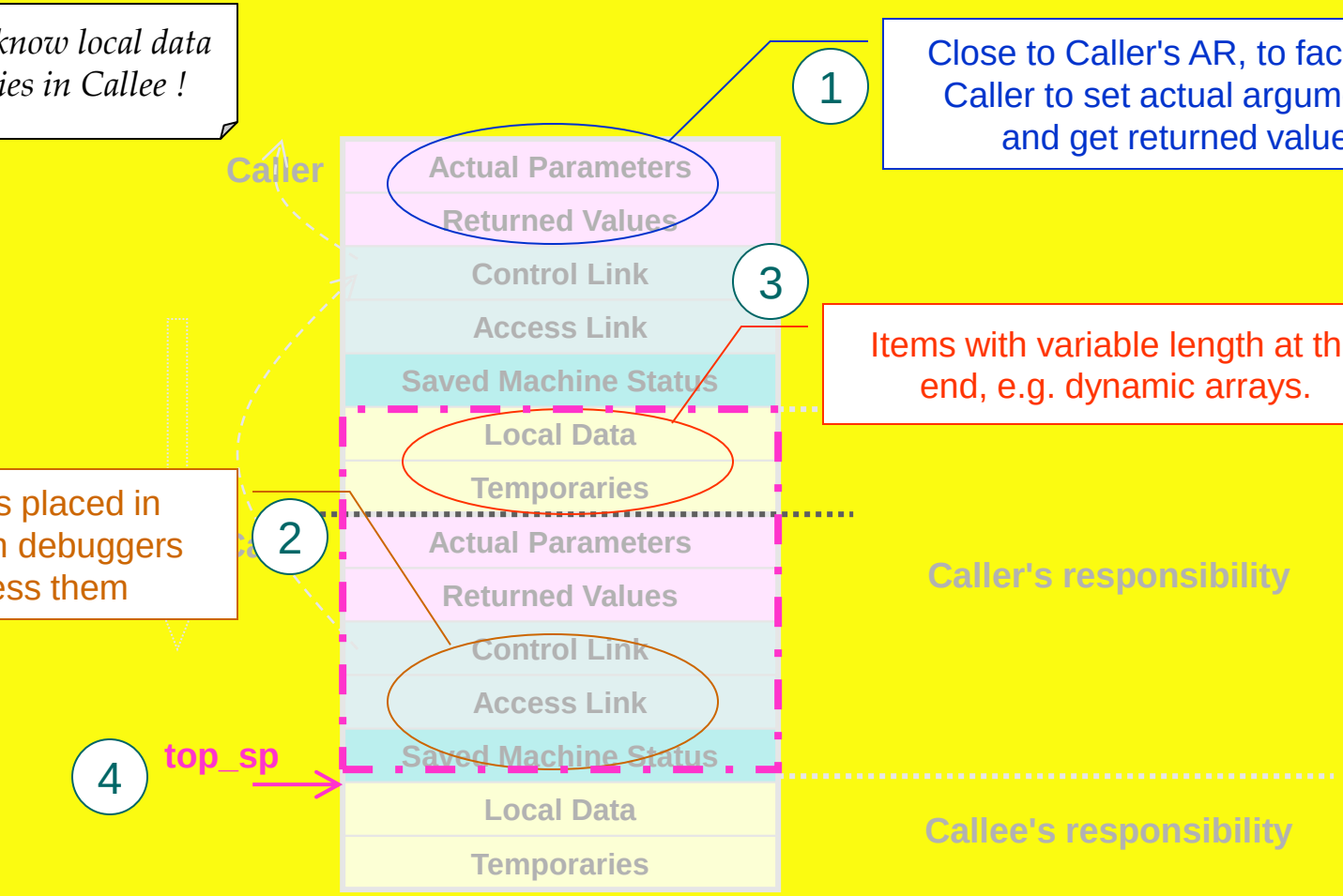
Caller vs. Callee

- Implementation of procedure calls



Activation Record Design

Caller does not know local data and temporaries in Callee !



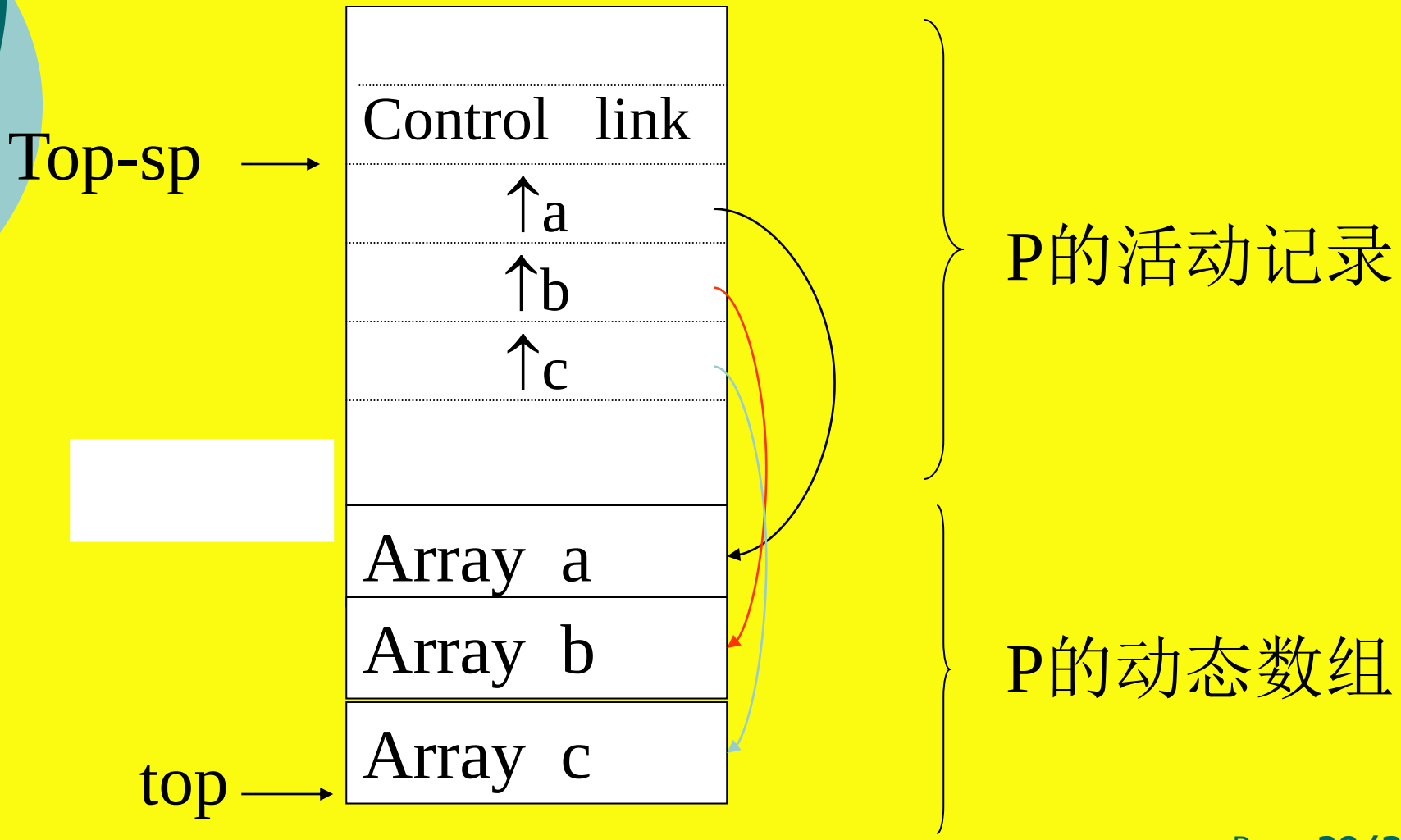
Calling Sequence

- 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).

```
PROCEDURE  exam(l,m,n:integer);
        VAR
              a:array [1..l] of real;
              b:array [1..m] of real;
              c:array [1..n] of real;
         BEGIN
               ……
          END;
```


Top-sp
Control link
↑a
↑b
↑c
Array a
Array b
Array c
top
P的活动记录
P的动态数组

Lecture 10. Run-Time Environment

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

Access links for finding nonlocal data

Sketch of ML program that uses function-parameters

Lecture 10. Run-Time Environment

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

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.

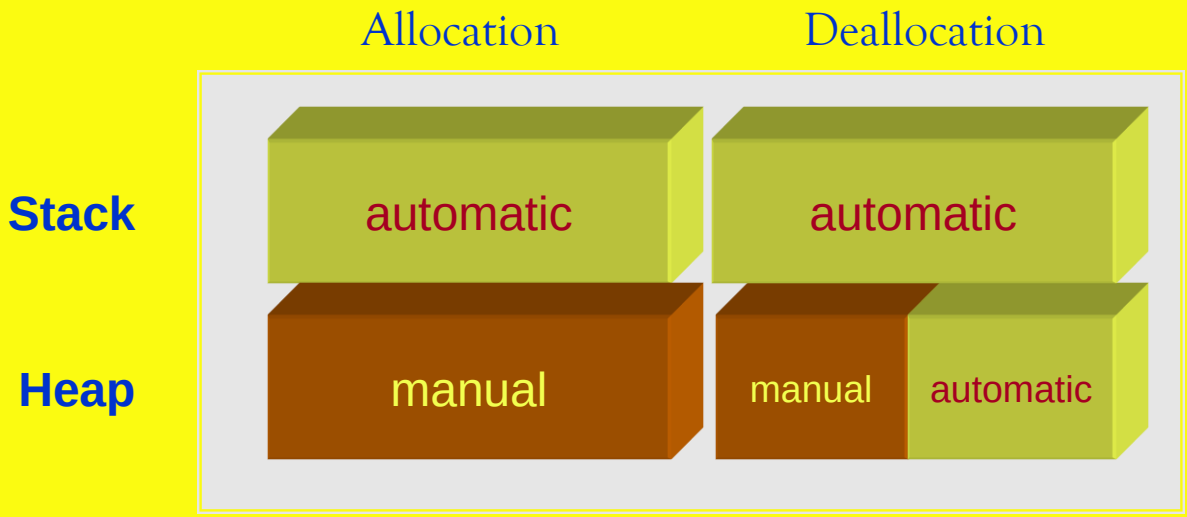
ypical Memory Hierarchy Configurations

Lecture 10. Run-Time Environment

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

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

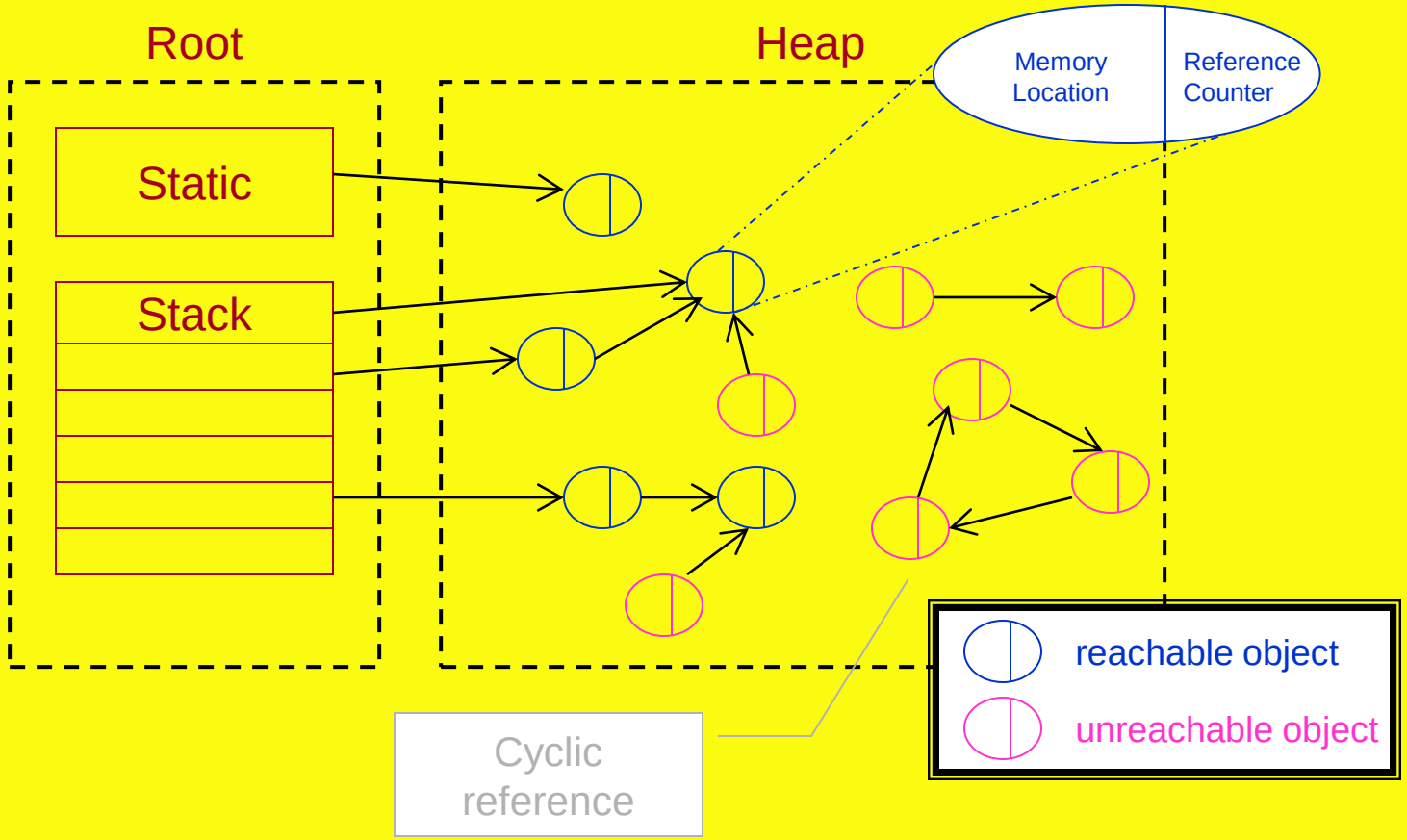
- Overall Execution Time
- Space Usage
  - Avoid fragmentation
- Pause Time
  - Critical to real-time applications
- Program Locality
  - The program spend most of time executing a relative small code fraction and touching only a small data fraction. (90% time on 10%code)
  - Locality facilitate the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

Garbage Collection Algorithms

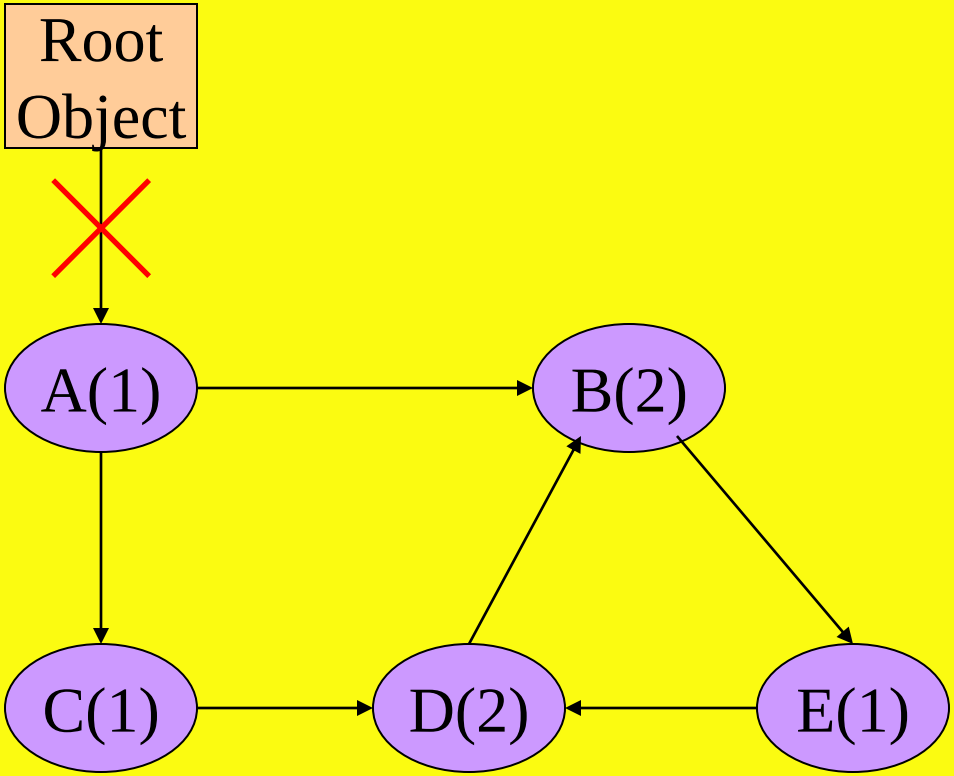
- Reference counting (immediately)
- Trace-based algorithms (periodically)
  - Basic Mark-and-Sweep
  - Baker's Mark-and-Sweep
  - Basic Mark-and-Compact
  - Cheney's Copying Collector
- Short-pause algorithms
  - Incremental garbage collection
  - Incremental reachability analysis
  - Partial collection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

Reference Counting

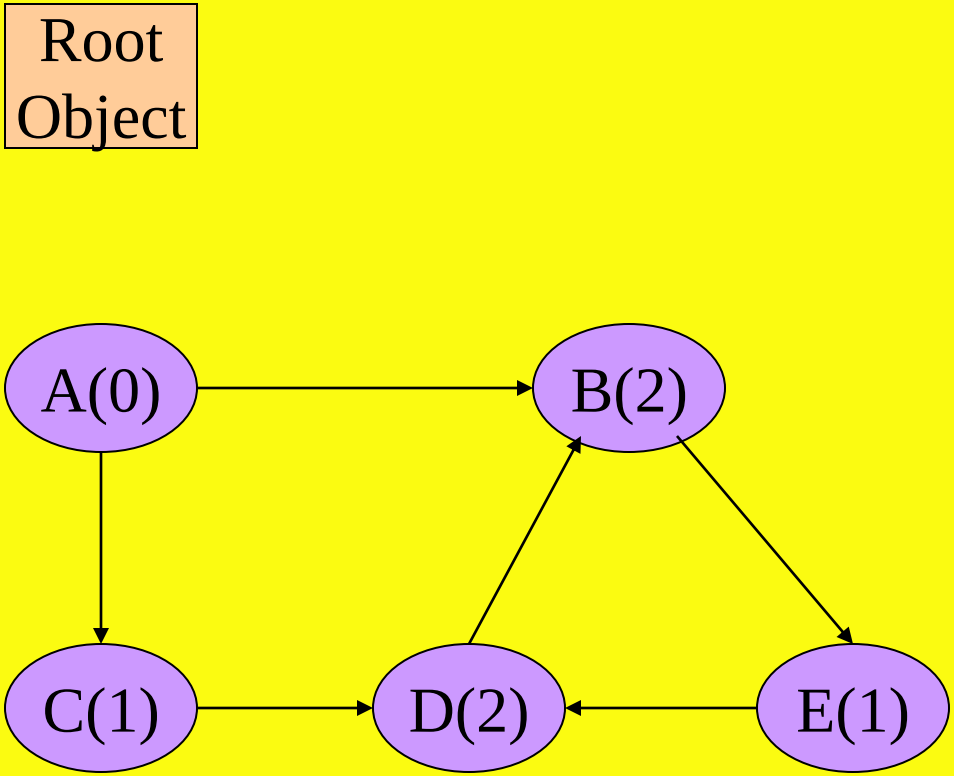
- counter++ vs. counter--



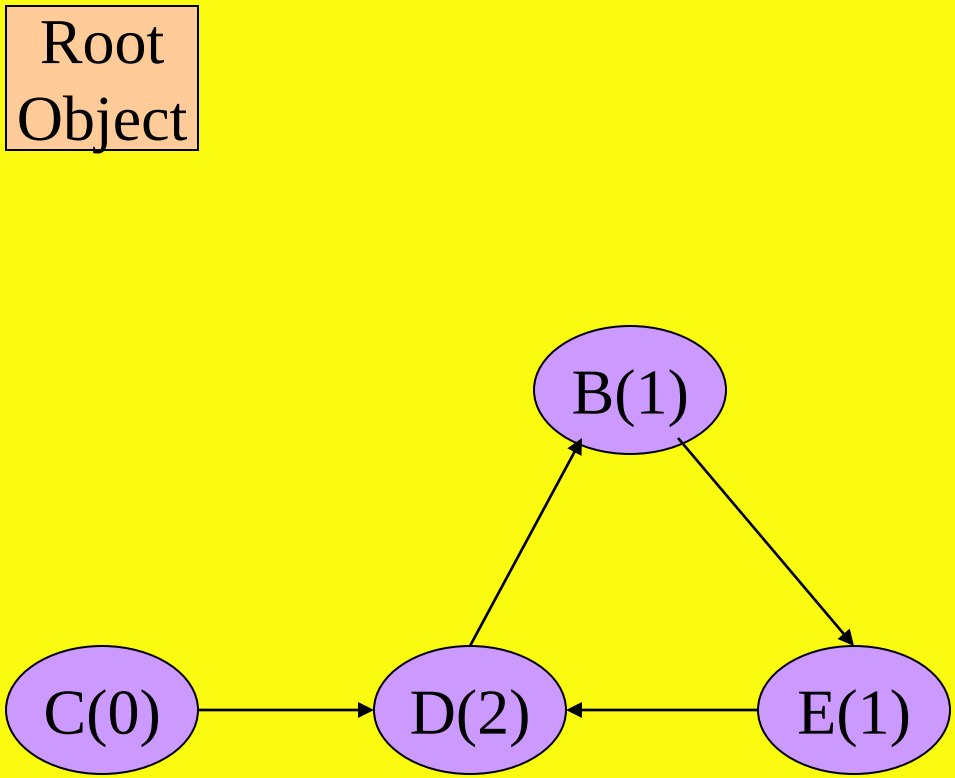
Example: Reference Counting



Example: Reference Counting



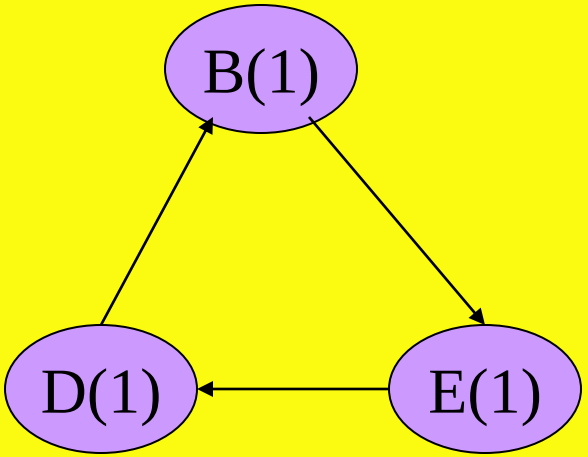
Example: Reference Counting



Example: Reference Counting

Root Object

B, D, and E are garbage, but their reference counts are all > 0. They never get collected.

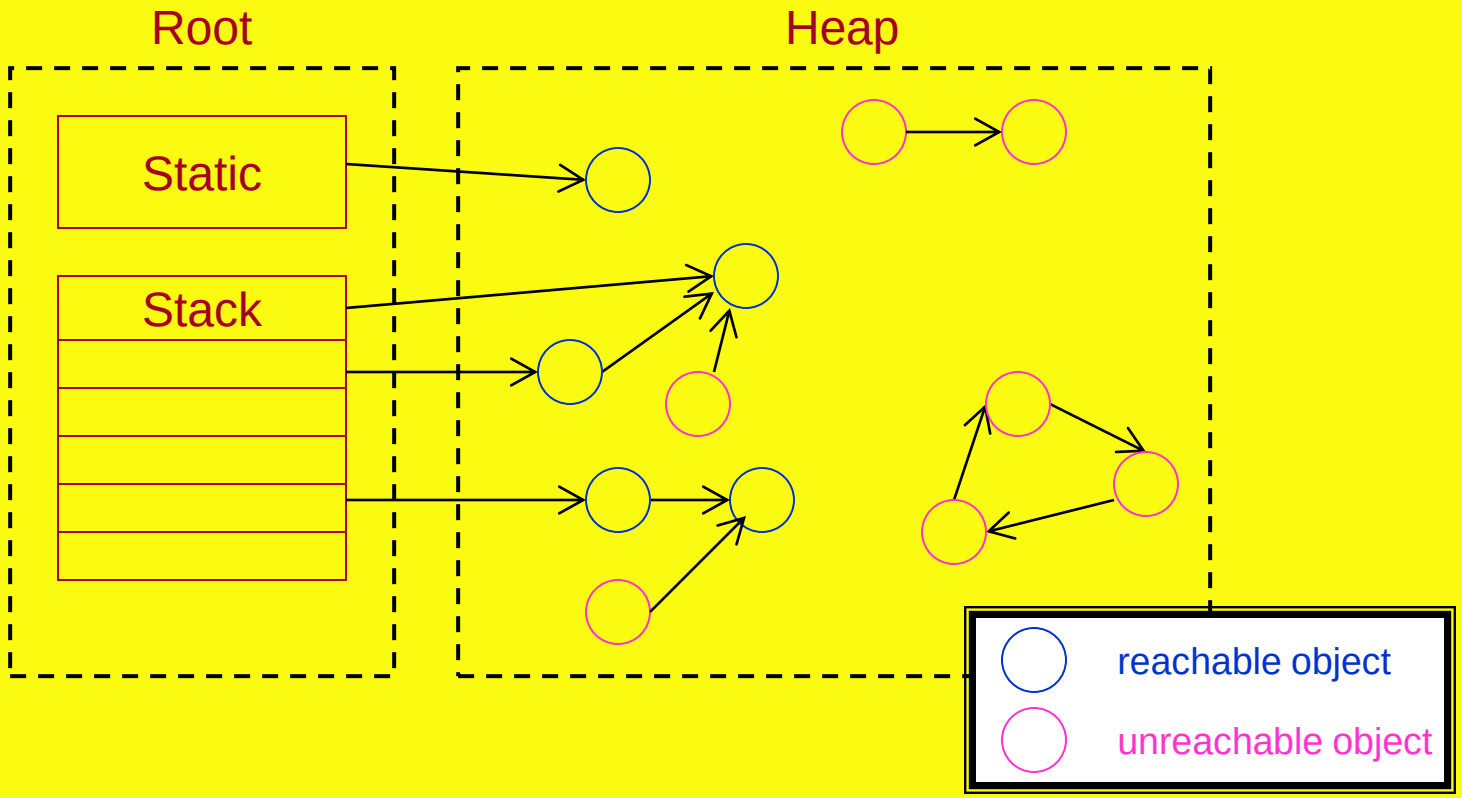


Reference Counting (cont')

- 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.

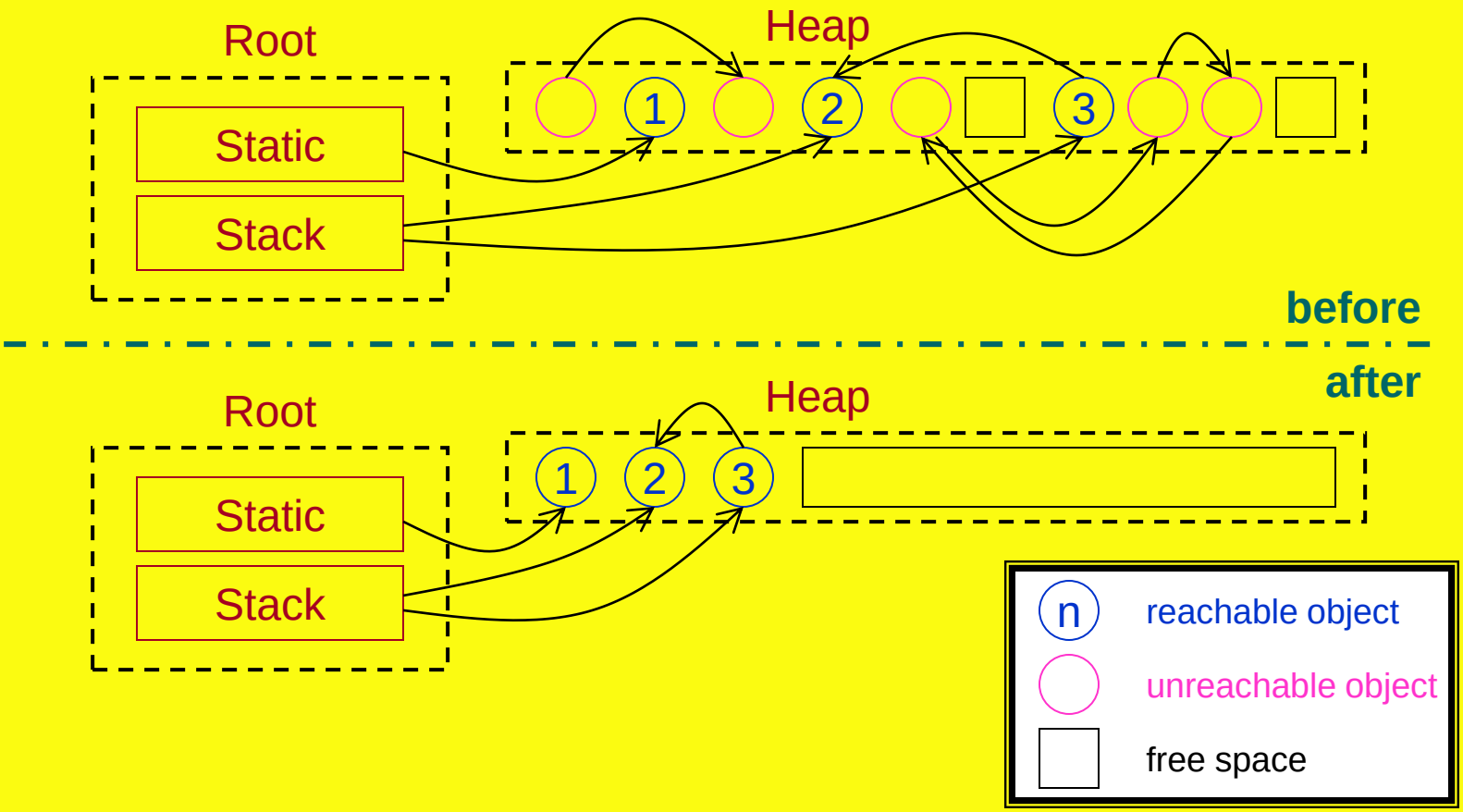


Mark-and-Sweep (cont')

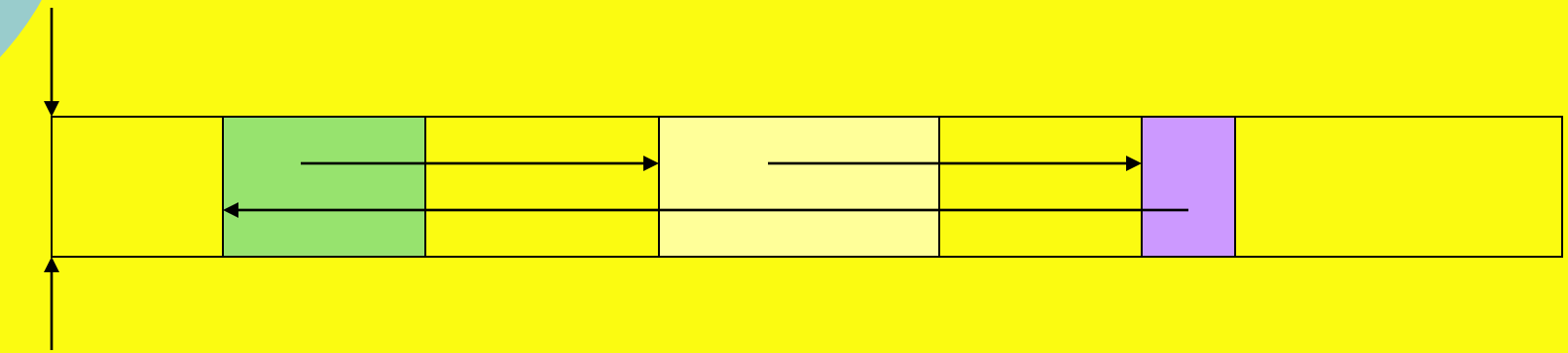
- Pros
  - High efficiency if little garbage exist.
  - Be able to collect cyclic references.
- Cons
  - Low efficiency with large amount of garbage
    - Improvement by H. Baker (1992)
    - Keep a list of all allocated objects.
  - Pause of the mutator (program) make it difficult to apply to real-time applications.
  - Leads to fragmentation in the heap.

Mark-and-Compact

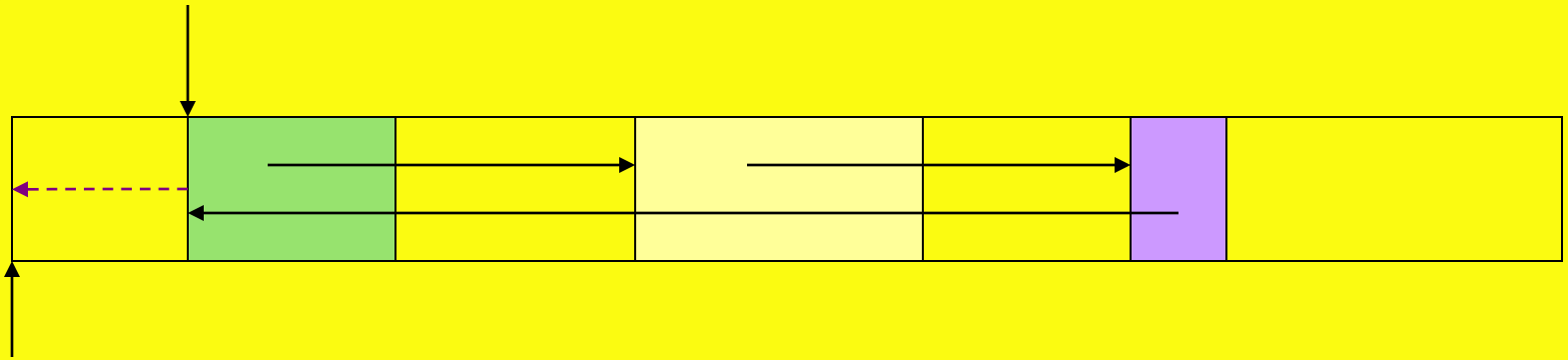
- Before vs. after



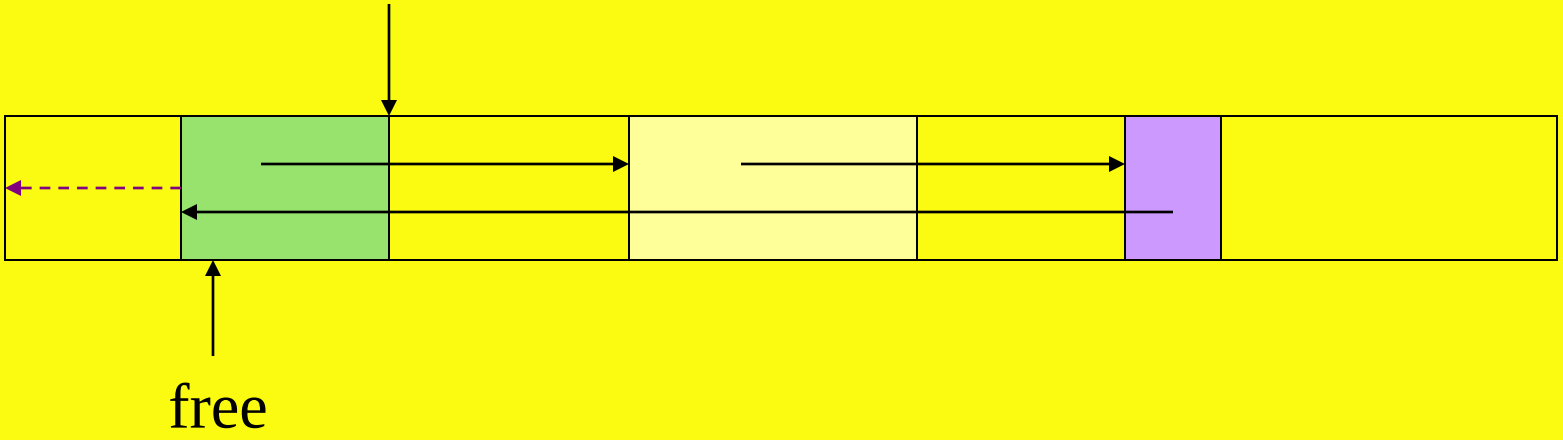
Example: Mark-and-Compact



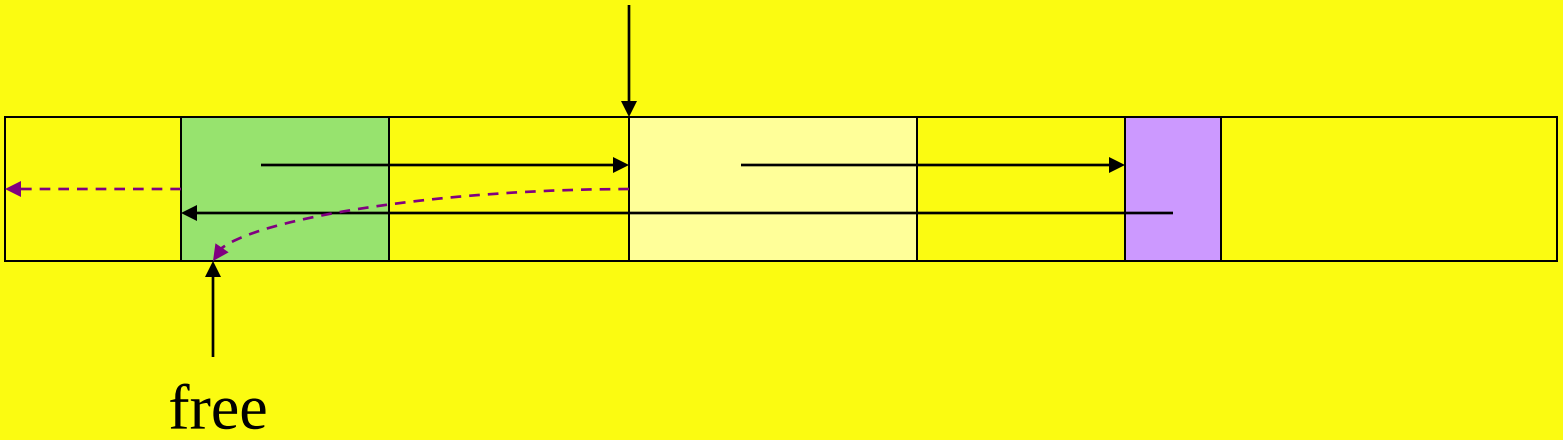
Example: Mark-and-Compact



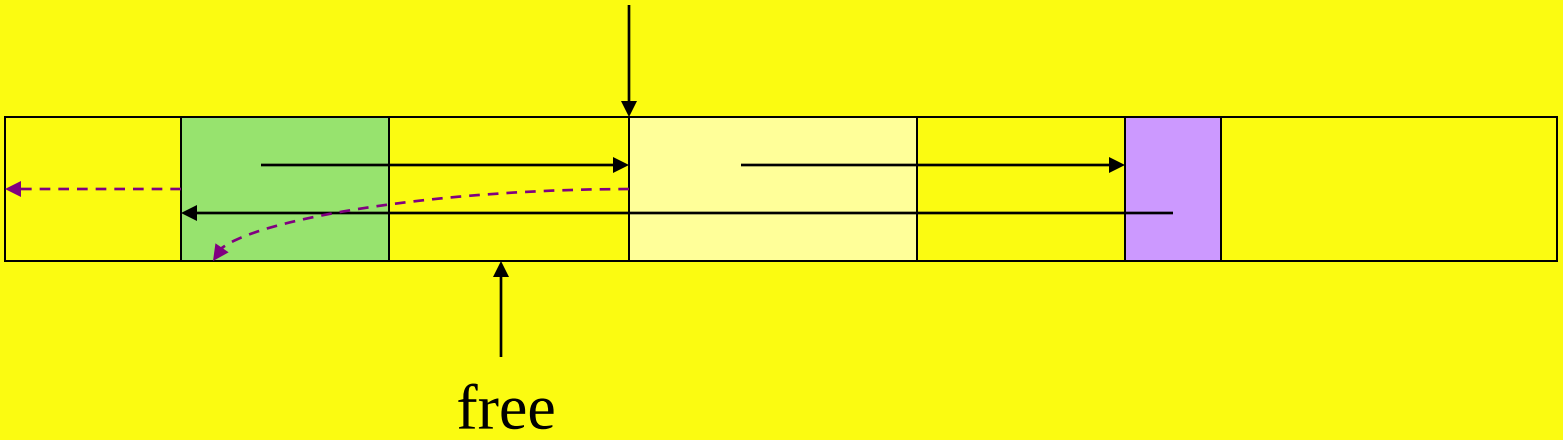
Example: Mark-and-Compact



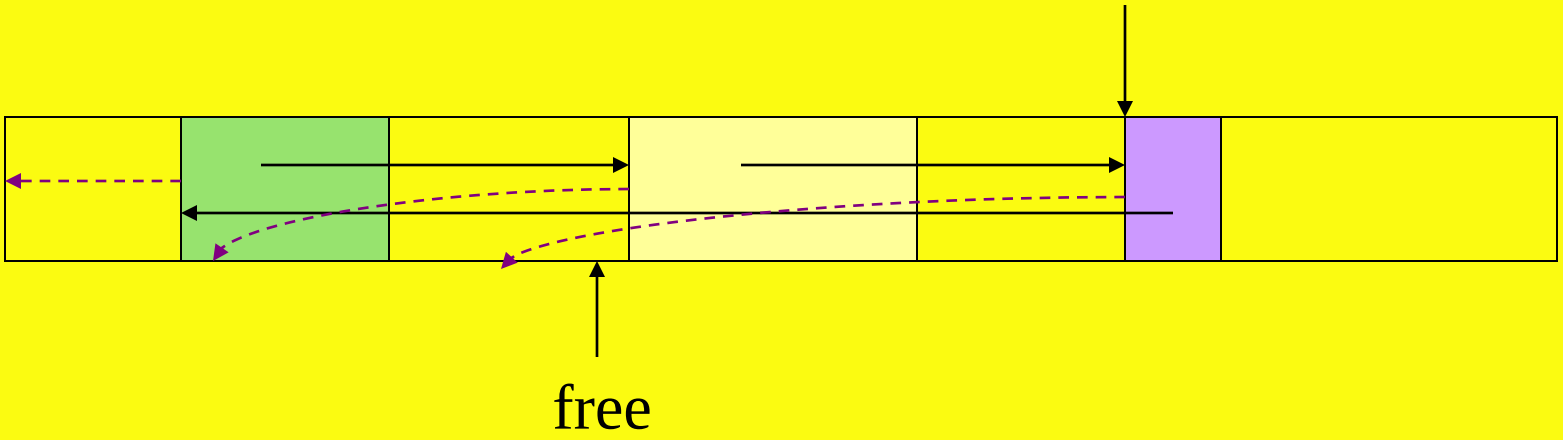
Example: Mark-and-Compact



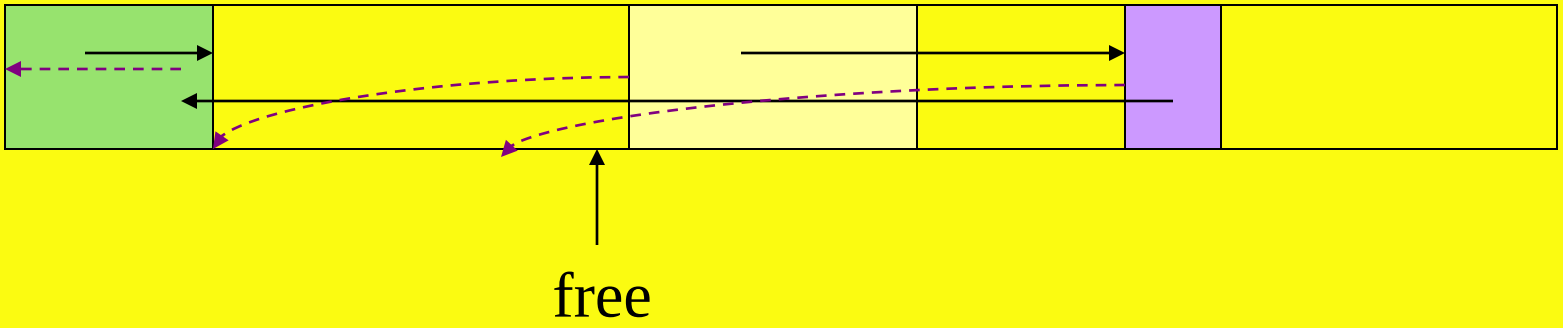
Example: Mark-and-Compact



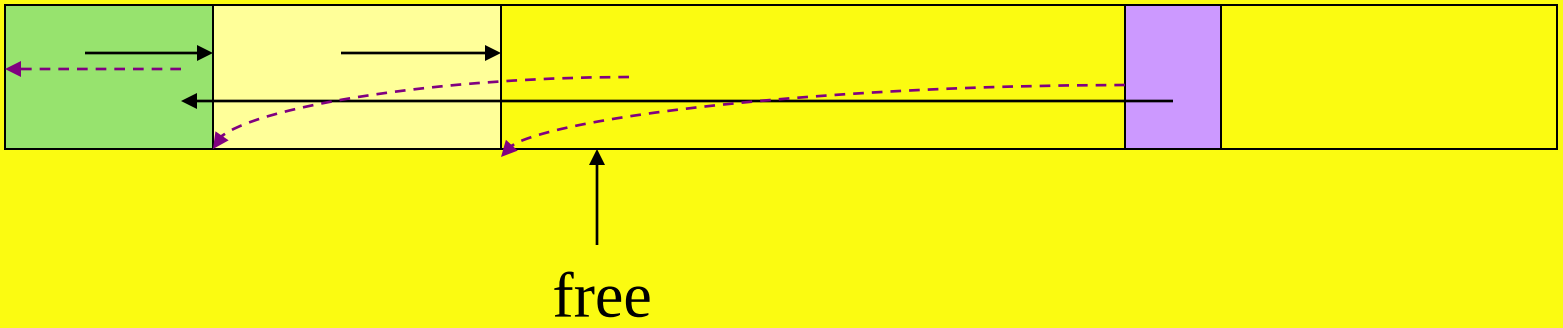
Example: Mark-and-Compact



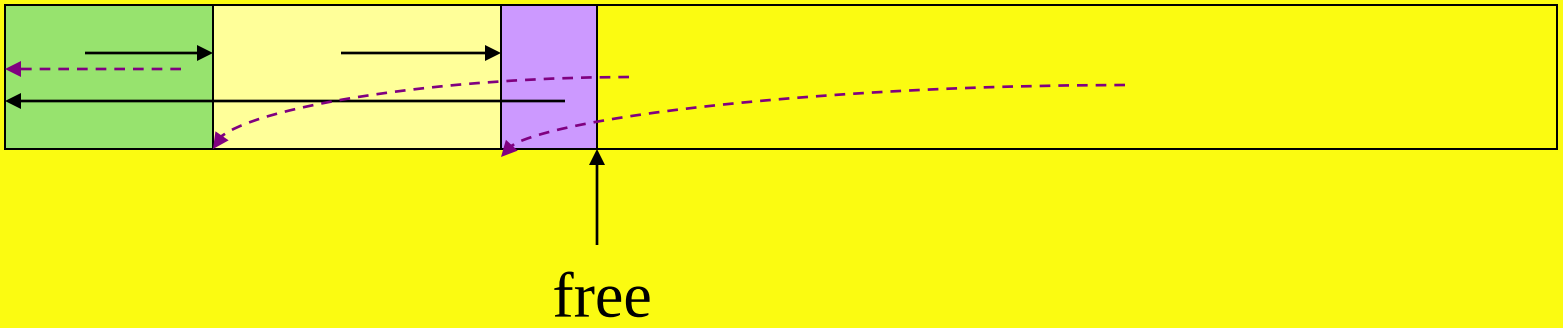
Example: Mark-and-Compact



Example: Mark-and-Compact



Example: Mark-and-Compact

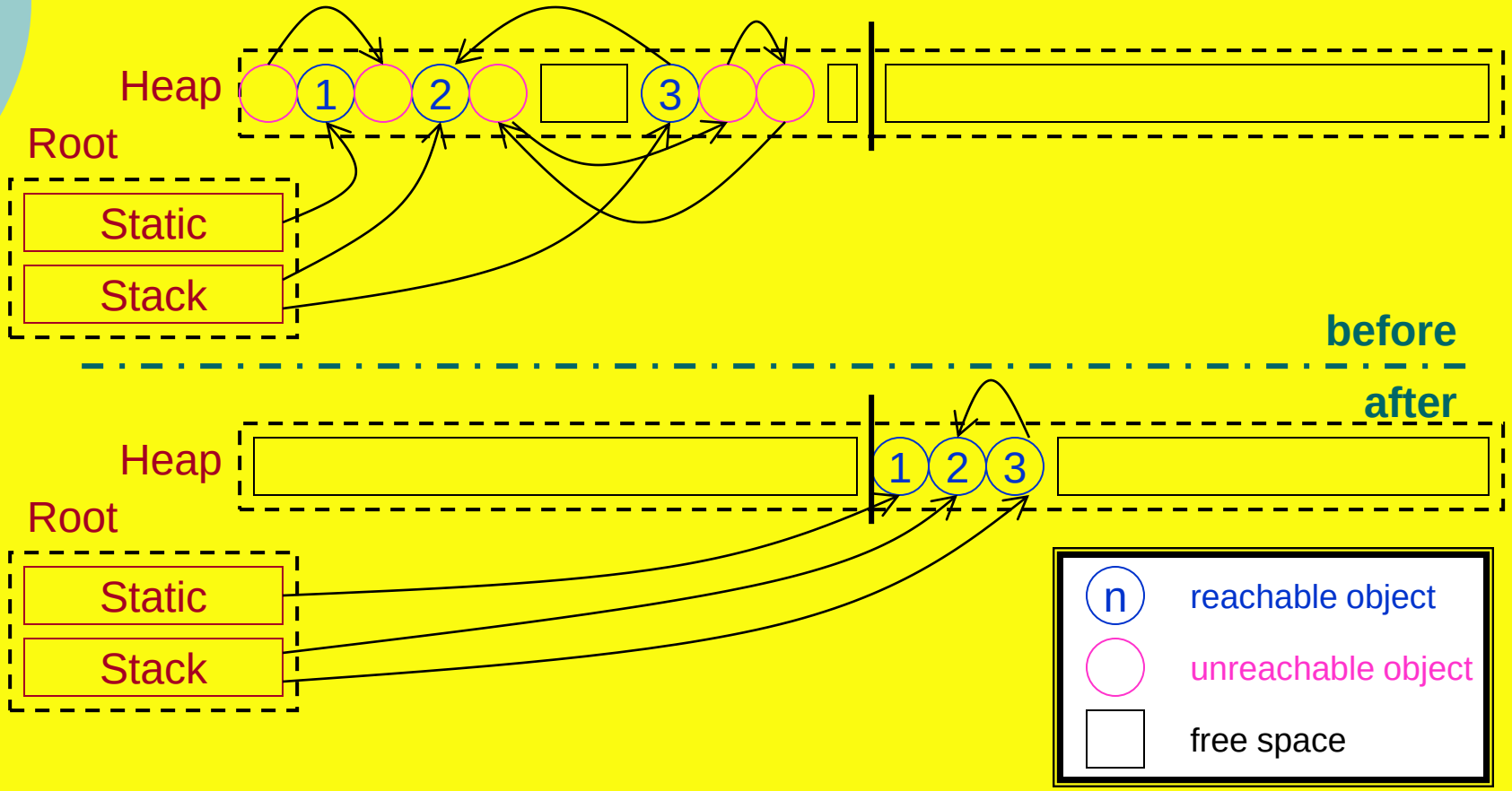


Mark-and-Compact (cont')

- 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

Enjoy the Course!