Software Development for Embedded Systems

Witawas Srisa-an

CSCE 496: Embedded Systems Design and Implementation

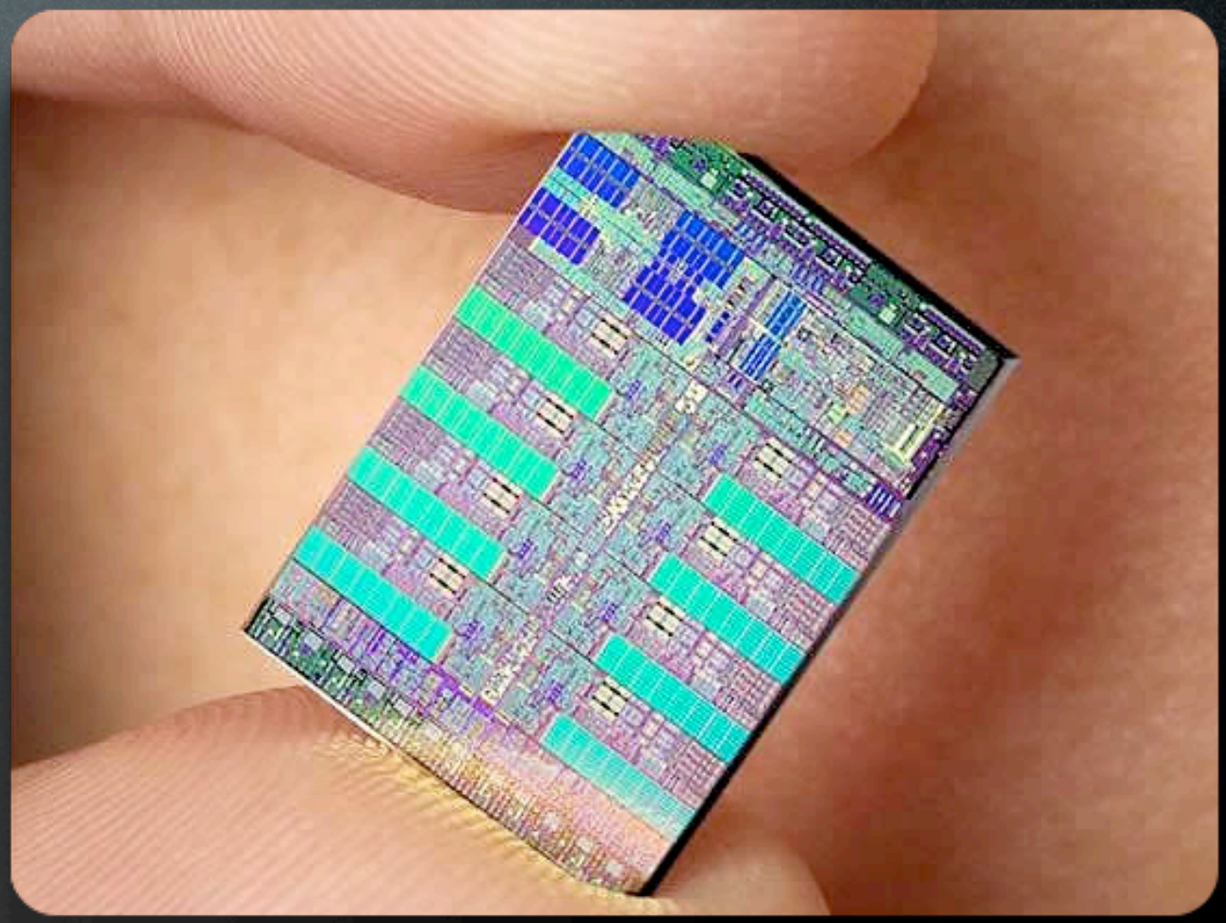
Reminder

- Homework on Gaussian Filter is due on Wednesday before 11:59 pm
  - if you are having difficulties reading the provided image file, be sure to read the help page on fopen in the Stretch IDE
- Any questions about malloc?

2

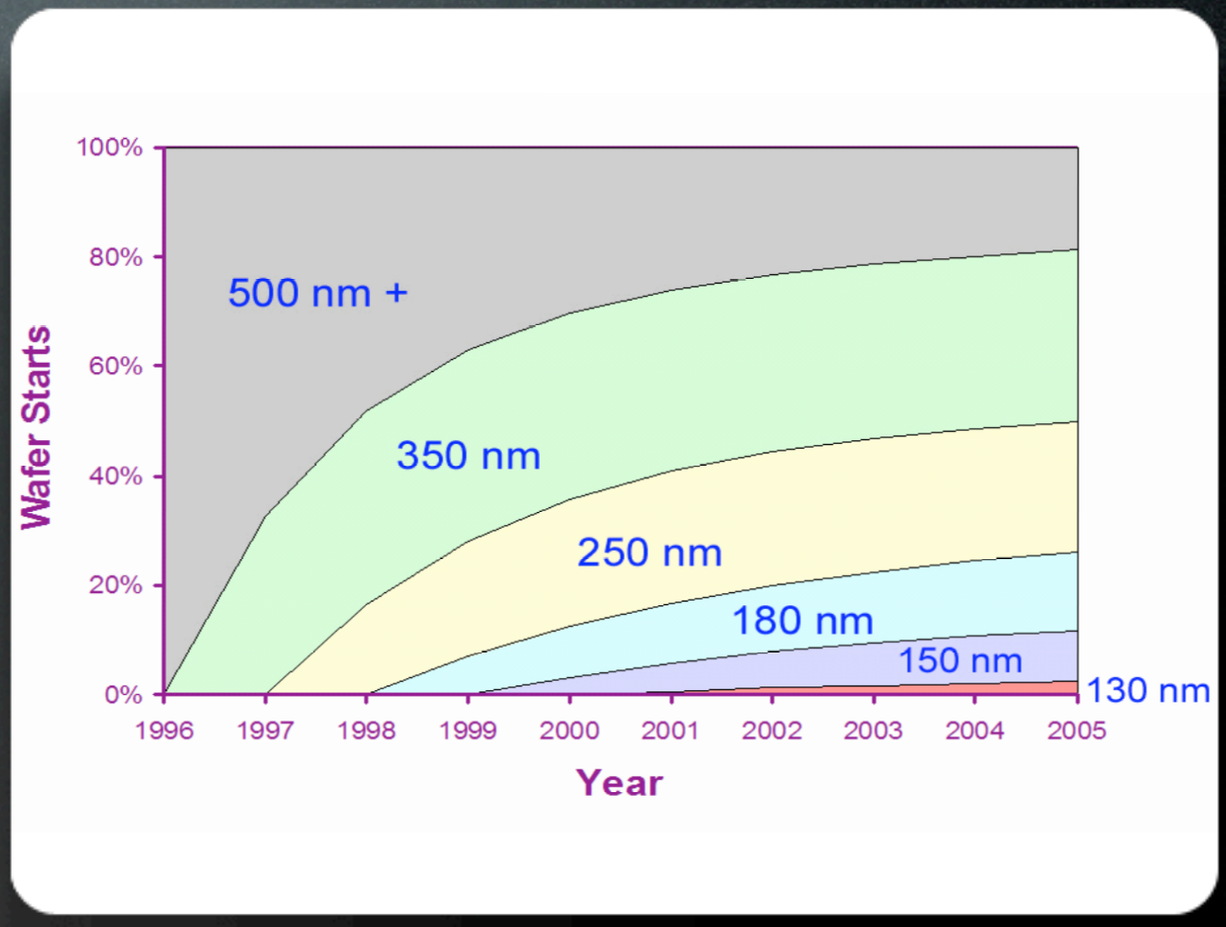
Hardware Trends

- Proliferation of complex embedded systems
  - powerful processors
    - feature-rich (e.g. Cell, Stretch, ARM)
  - advanced runtime support
    - similar features found in processors for desktop and server systems (e.g. MMU, multi-core, fast bus, etc.)



Hardware Trends

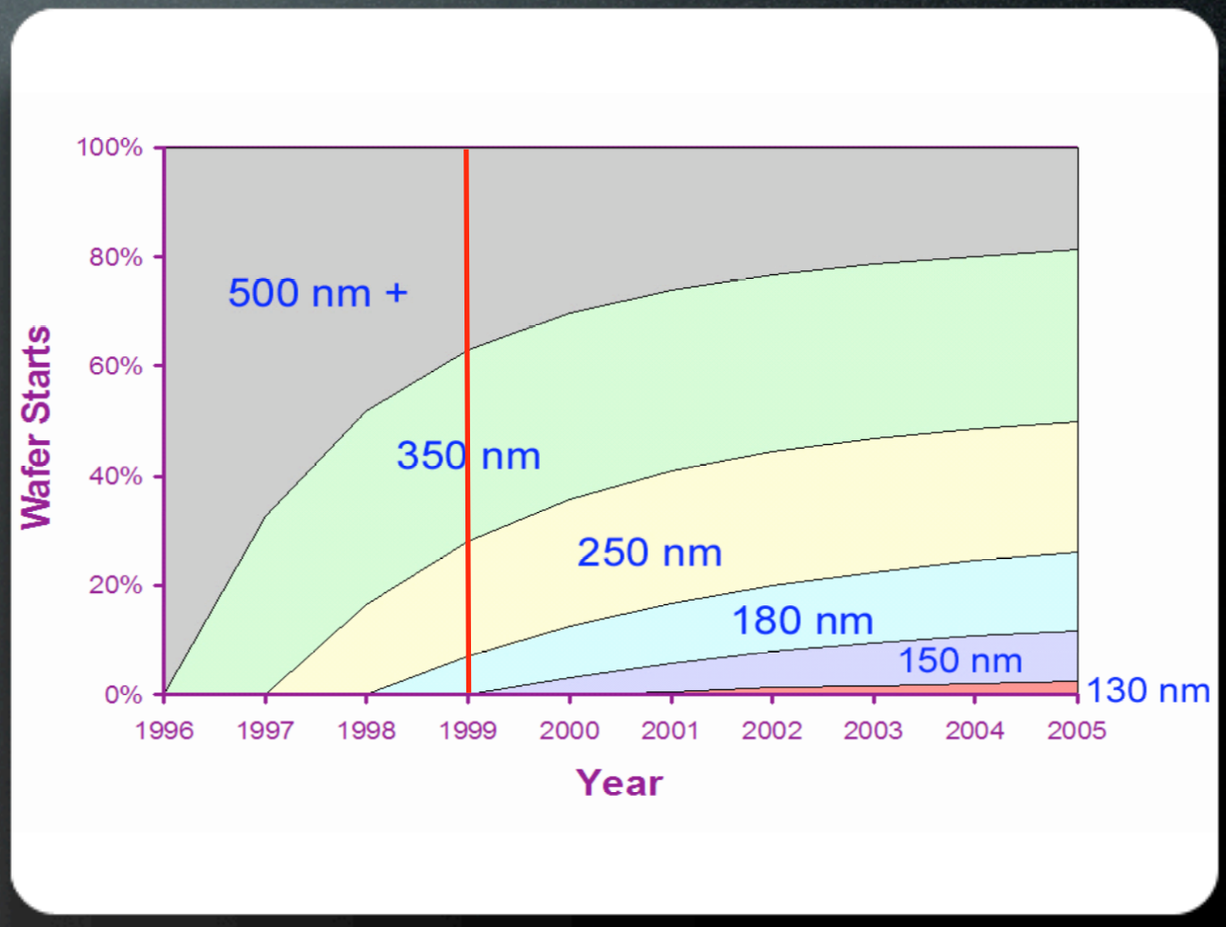
- We are reaping the major benefit of Moore's law
  - old processors don't go away
    - powerful enough for day-to-day applications
    - very low cost



4

Hardware Trends

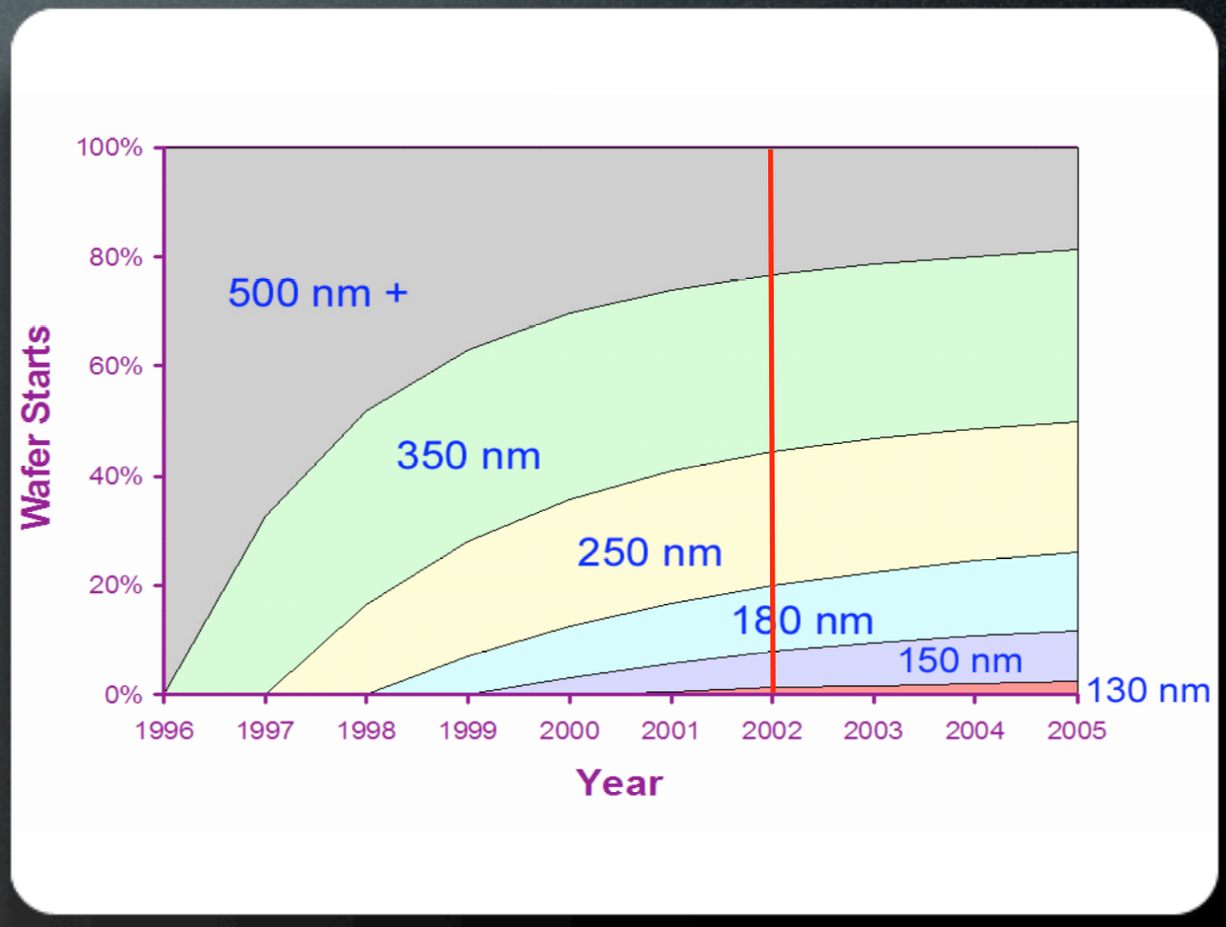
- We are reaping the major benefit of Moore's law
  - old processors don't go away
    - powerful enough for day-to-day applications
    - very low cost



4

Hardware Trends

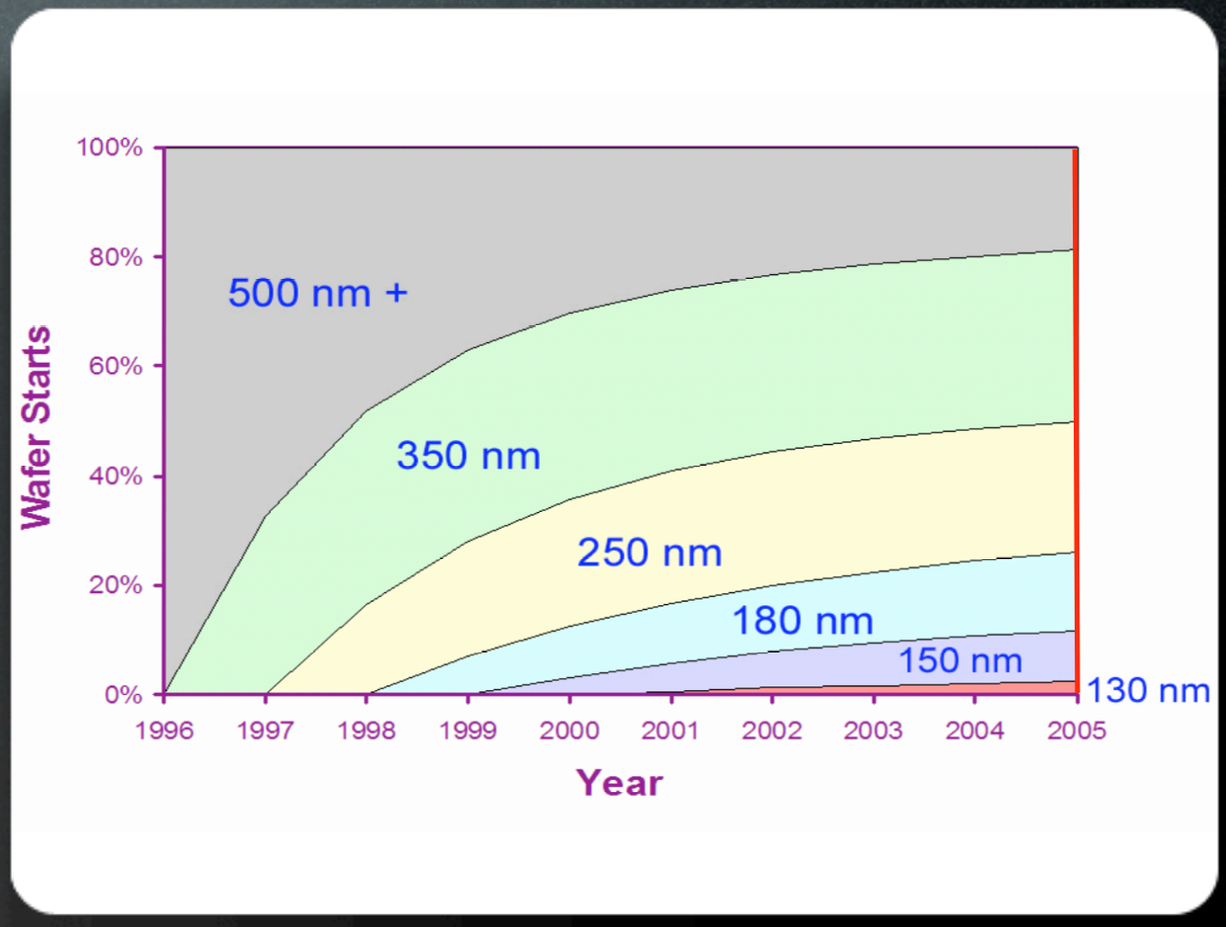
- We are reaping the major benefit of Moore's law
  - old processors don't go away
    - powerful enough for day-to-day applications
    - very low cost



4

Hardware Trends

- We are reaping the major benefit of Moore's law
  - old processors don't go away
    - powerful enough for day-to-day applications
    - very low cost



4

Software Trends

- Complex software systems
  - why not? The hardware can handle it!
- Avionic Software for Boeing ScanEagle UAV > 300,000 line of code

5

Software Trends

- Feature-rich runtime support
  - full-fledged desktop/ server operating systems in embedded devices
  - capability to run more complex software systems on these machines

6

Software Trends

- Leverage mobile/ embedded devices to provide services
  - Telesensing from Lucent
    - prevent Sudden Infant Death Syndrome, detect sleep apnea, etc.
  - Full-fledged web browsers, calendars, file servers, etc.

7

Software Trends

- Assume heterogeneous platforms
  - the billionth handset shipped in 2006
    - build for portability
    - build for generic input/output devices
  - about 20+ operating systems for these devices
    - build to interface with underlying runtime support features

8

Software Trends

- Summary
  - large software systems now and larger in the future
    - more software reuse?
  - assume heterogeneous platforms
    - must be portable
  - providing similar runtime features to much more complex systems

9

Enabling Technologies

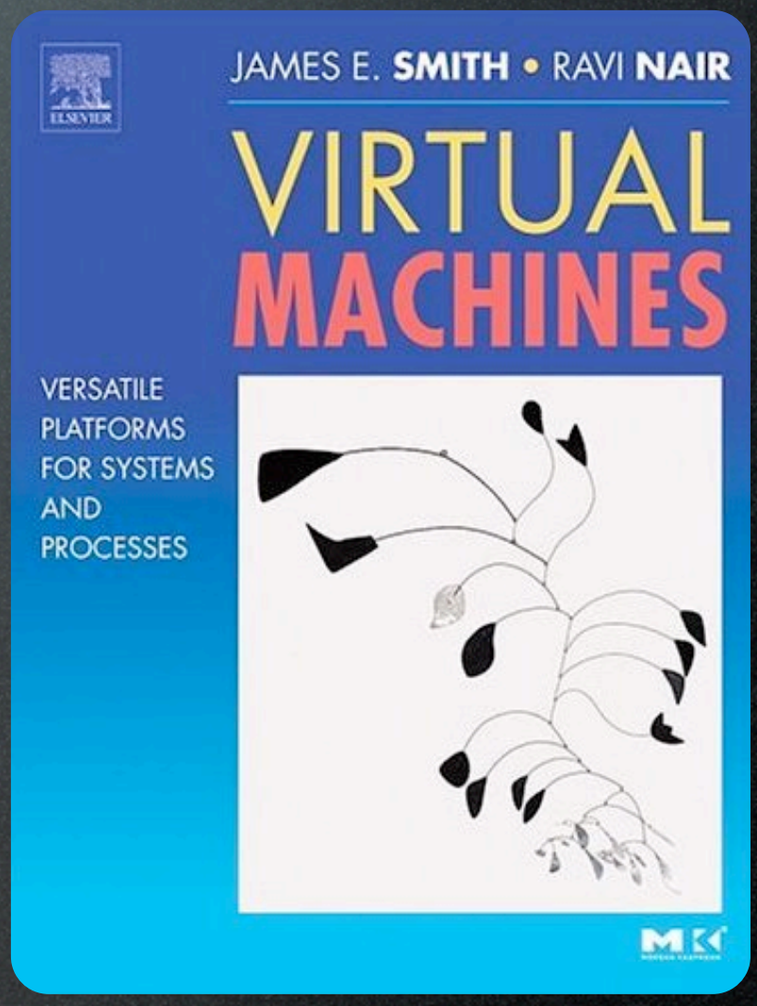
- Modern programming languages (e.g. Java, C#, VB.NET)
  - Object-oriented paradigm
    - promote code reuse
- Virtual Machine (VM) based systems
  - achieve portability but require complex runtime support
    - now available in many embedded devices

10

Virtual Machines

"A **virtual machine** is software that creates a virtualized environment between the computer platform and its operating system, so that the end user can operate software on an abstract machine."

Wikipedia

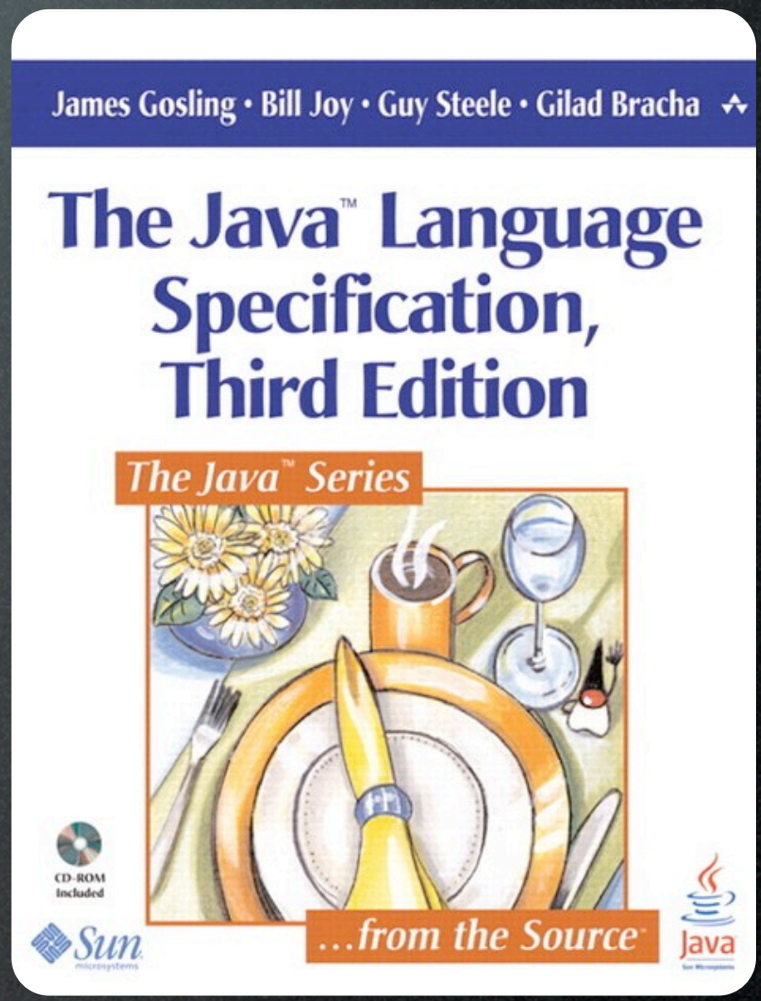


11

Java Virtual Machines

"A **Java Virtual Machine (JVM)** is virtual machine that interprets and executes Java bytecode. This code is most often generated by Java language compilers..."

Wikipedia



12

Java Virtual Machines

- We'll look at the ones developed mainly by Sun Microsystems
  - HotSpot
  - CLDC HotSpot
  - KVM (Kilo Virtual Machine)
    - second most used VM in the world

13

Java Virtual Machines

- We'll look at the ones developed mainly by Sun Microsystems

Trivia: What is the most used VM developed by Sun?

- CLDC HotSpot
- KVM (Kilo Virtual Machine)
  - second most used VM in the world

14

Java Virtual Machines

- KVM vs. HotSpot
  - KVM is interpretation based
    - maximum portability
  - HotSpot combines interpretation and dynamic compilation
    - platform dependent

15

Sidebar: Dynamic Compilation

- Interpreter is a big while loop with many case statements
  - each bytecode is translated to a predefine C/C+ function (e.g. new operator)
- Dynamic compiler takes each method and generates native code
  - can be optimized or non-optimized

16

Sidebar: Dynamic Compilation

- Compilation strategies
  - always compile (e.g. .NET Compact Framework, Jikes RVM)
  - only compile frequently used methods
- Code size
  - a compiled method can be 6 to 8 times larger than its bytecode representation

Sidebar: Dynamic Compilation

- Storage
  - these compiled methods are stored in a dynamic memory region
    - separate code-cache or intermingled with objects in the heap
- Management strategies
  - flush when full, GC, etc.

18

Java Virtual Machines

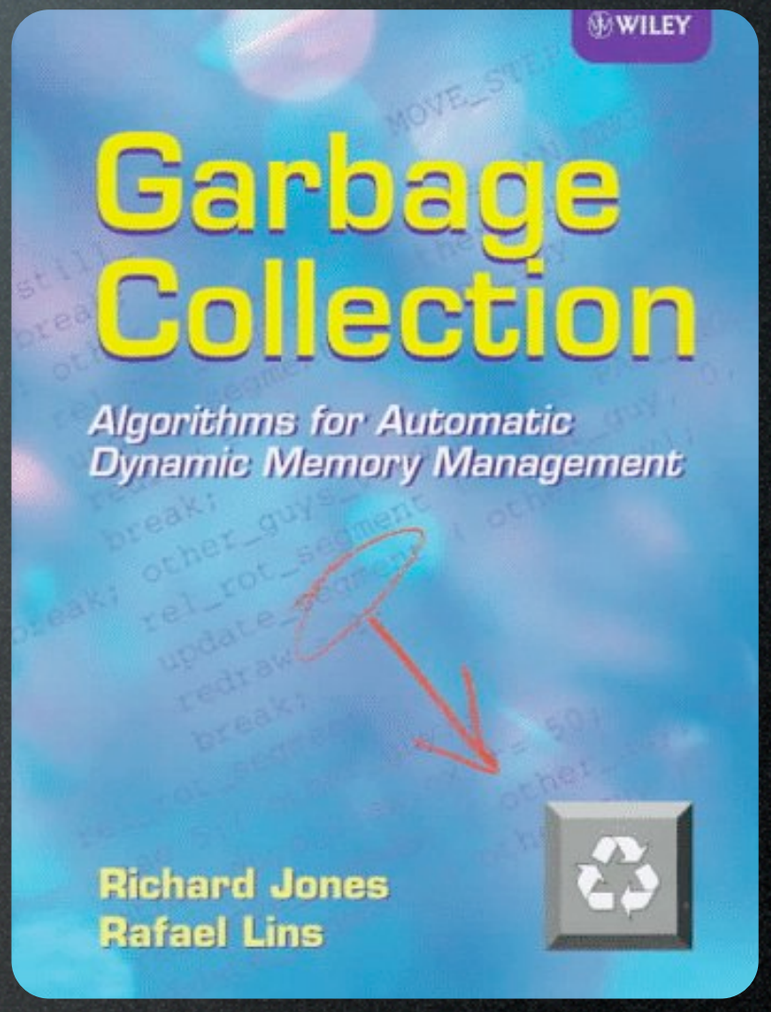
- KVM vs. HotSpot
  - KVM uses simple mark-sweep-compact garbage collection
    - simple but long execution pauses
  - HotSpot uses generational garbage collection
    - more complex with higher runtime overhead, but shorter pauses

19

Sidebar: Garbage Collection

“Garbage Collection (**GC**) is a form of automatic memory management. The garbage collector attempts to reclaim garbage, or memory used by objects that will never again be accessed or mutated by the application.”

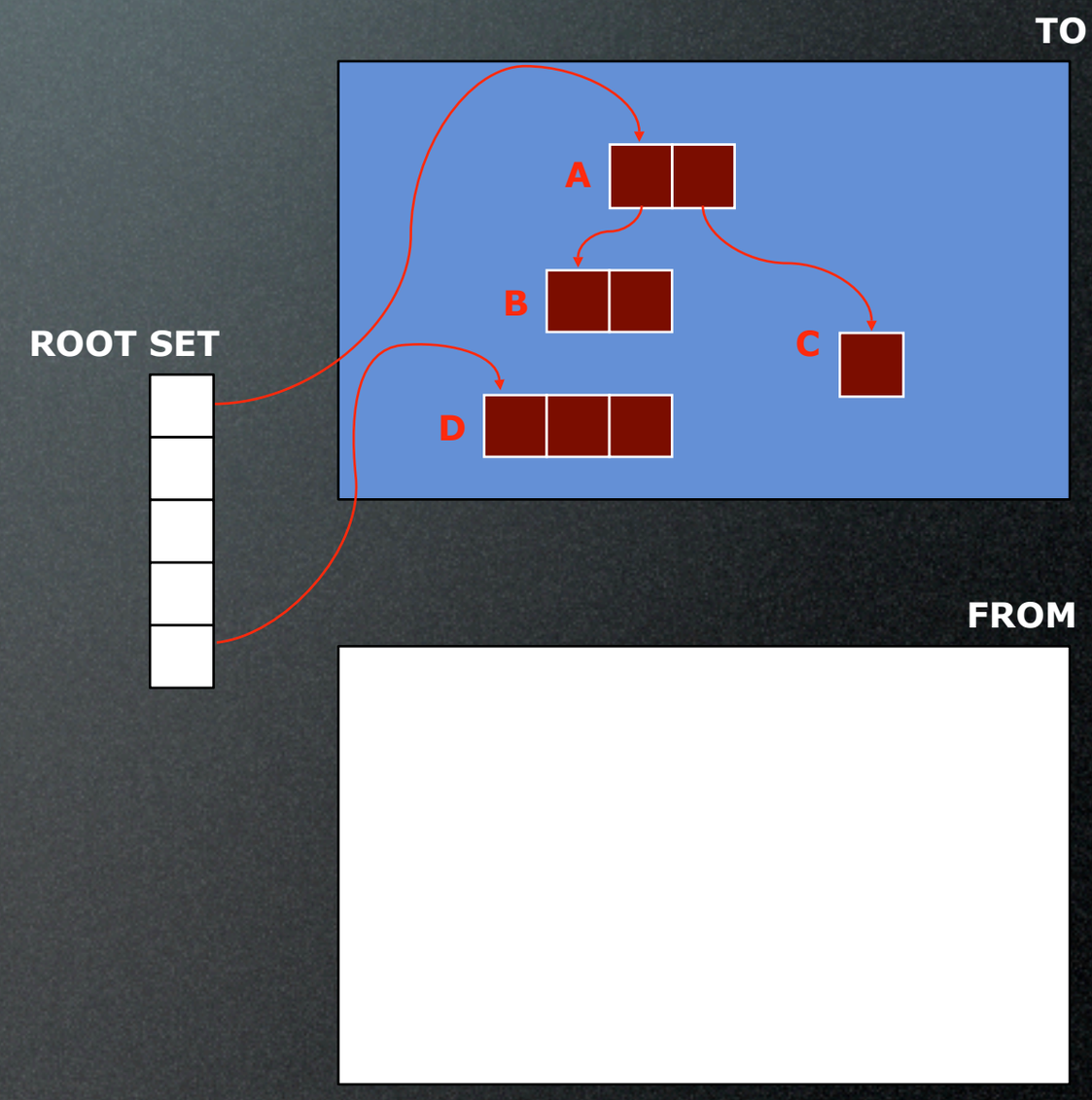
Wikipedia



20

Sidebar: Garbage Collection

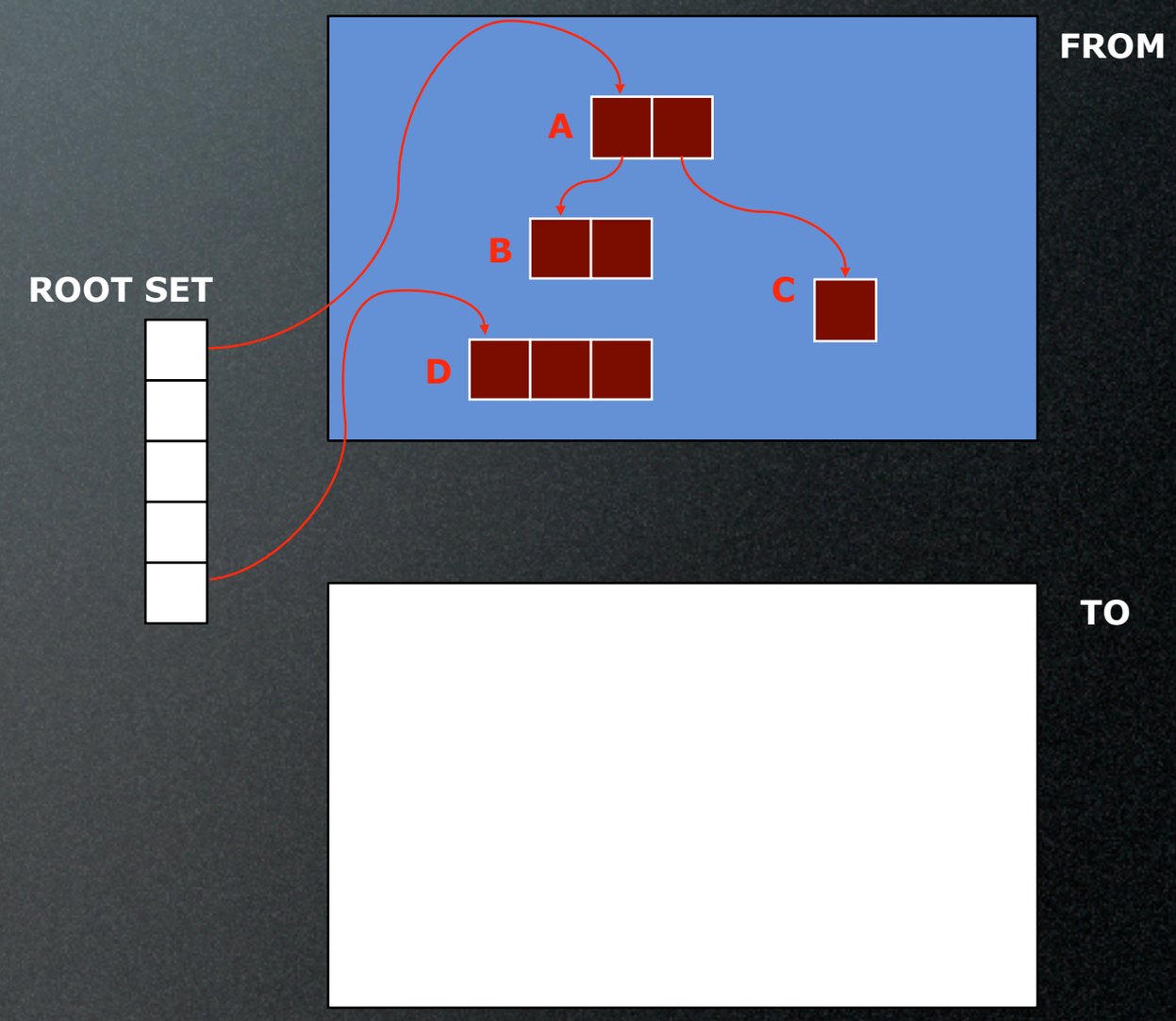
- Copying collector
  - split the heap in half, only one half is used each time
  - when the half is full, migrate surviving objects to the other half then allocate new objects from there



21

Sidebar: Garbage Collection

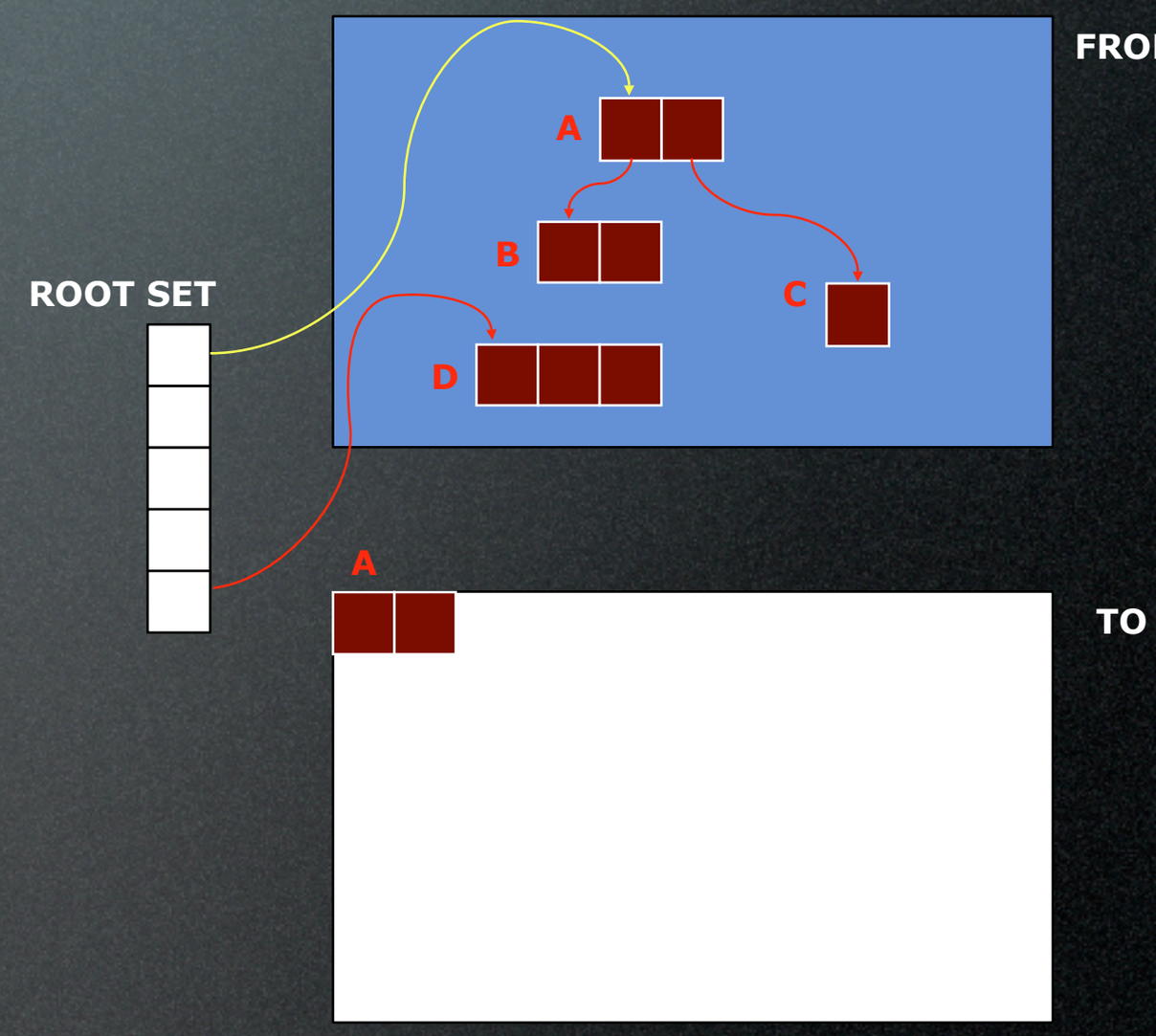
- Copying collector
  - split the heap in half, only one half is used each time
  - when the half is full, migrate surviving objects to the other half then allocate new objects from there



21

Sidebar: Garbage Collection

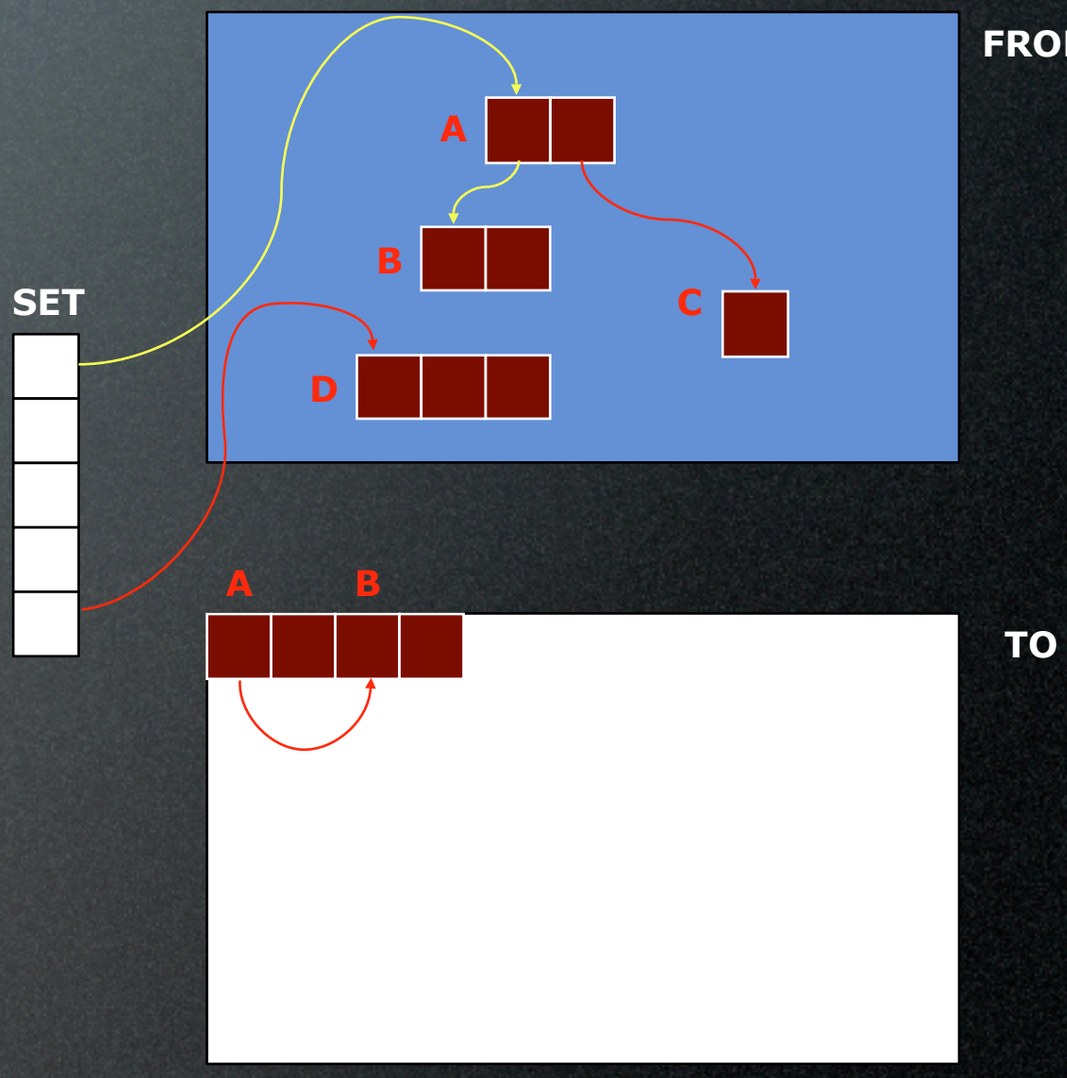
- Copying collector
  - split the heap in half, only one half is used each time
  - when the half is full, migrate surviving objects to the other half then allocate new objects from there



21

Sidebar: Garbage Collection

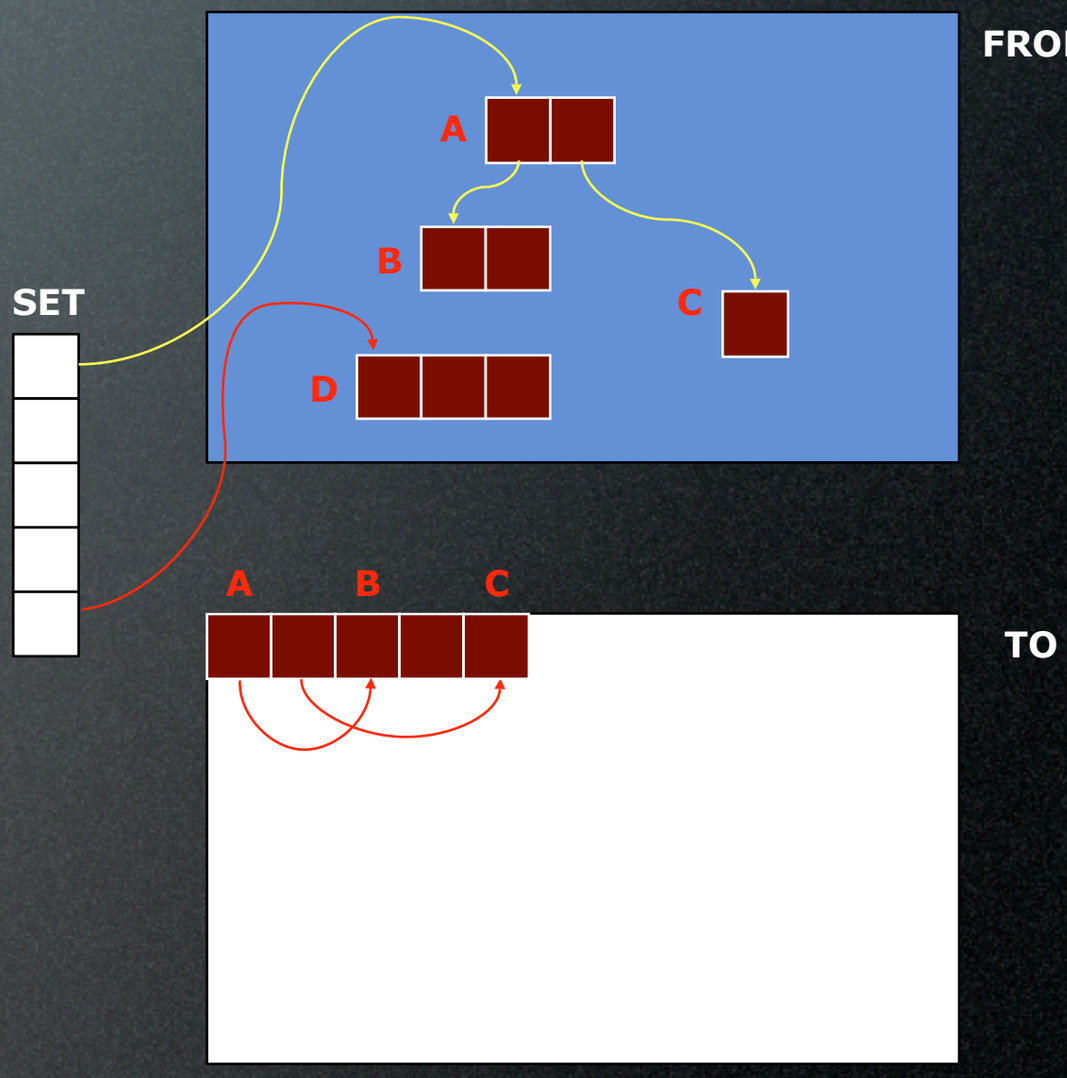
- Copying collector
  - split the heap in half, only one half is used each time
  - when the half is full, migrate surviving objects to the other half then allocate new objects from there



21

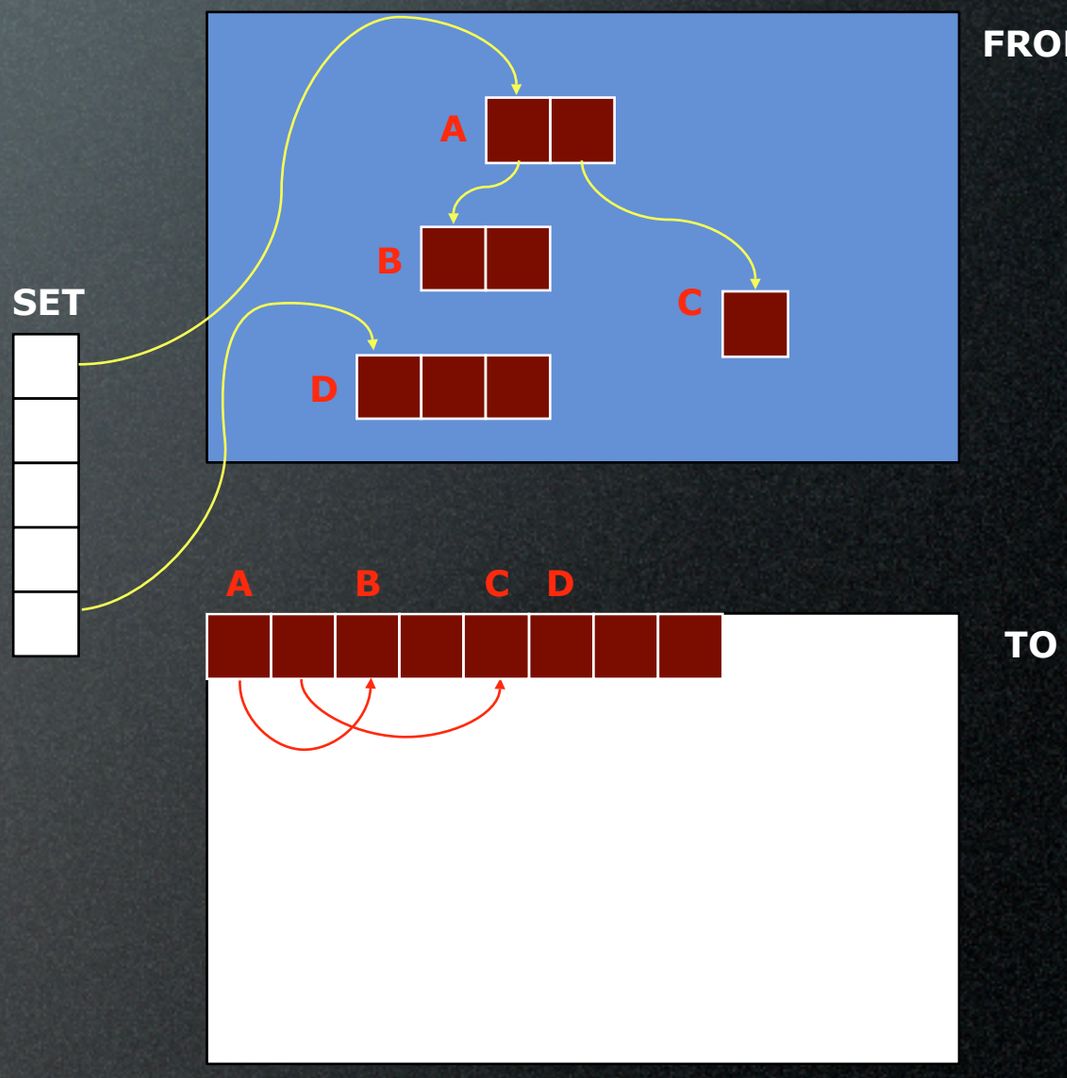
Sidebar: Garbage Collection

- Copying collector
  - split the heap in half, only one half is used each time
  - when the half is full, migrate surviving objects to the other half then allocate new objects from there



Sidebar: Garbage Collection

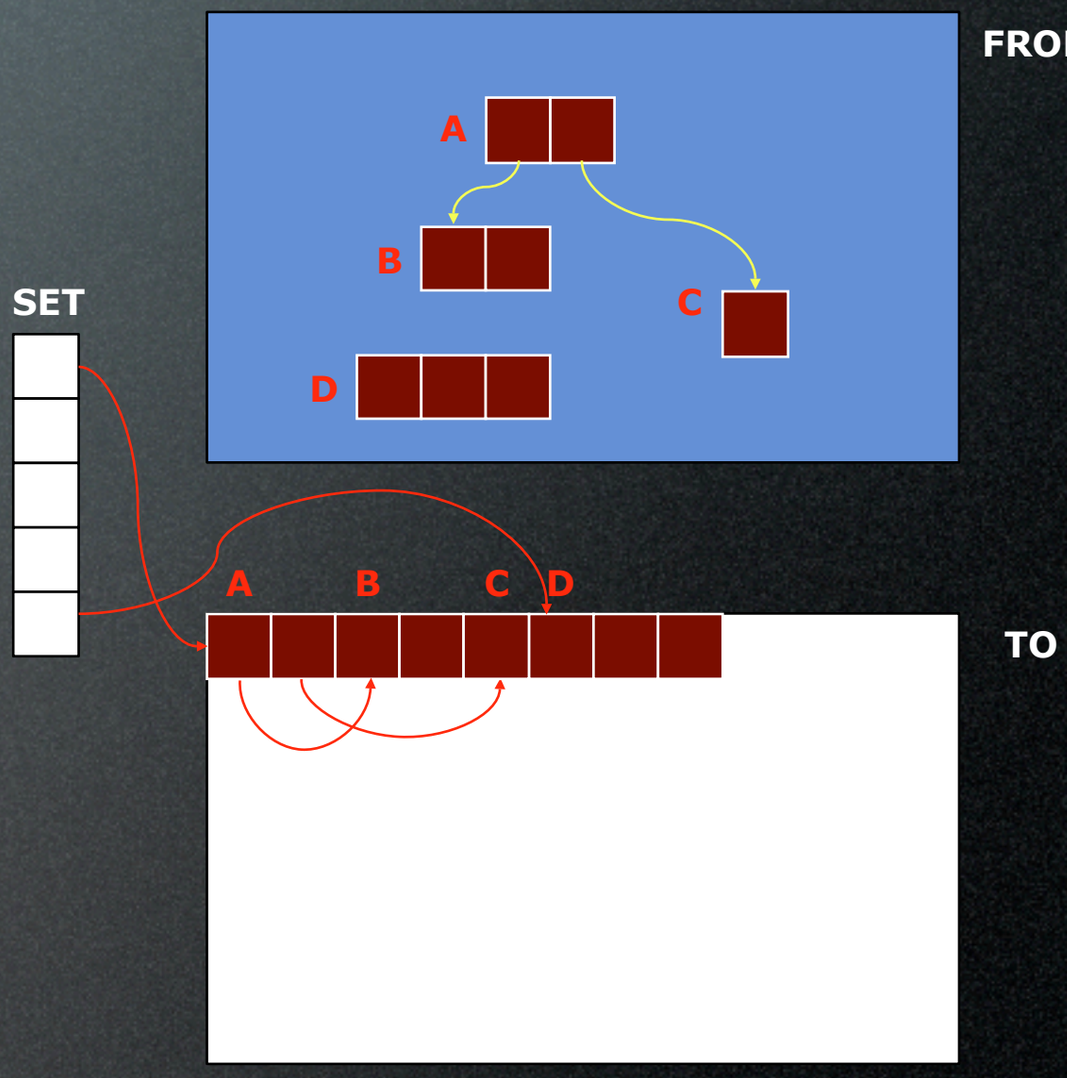
- Copying collector
  - split the heap in half, only one half is used each time
  - when the half is full, migrate surviving objects to the other half then allocate new objects from there



21

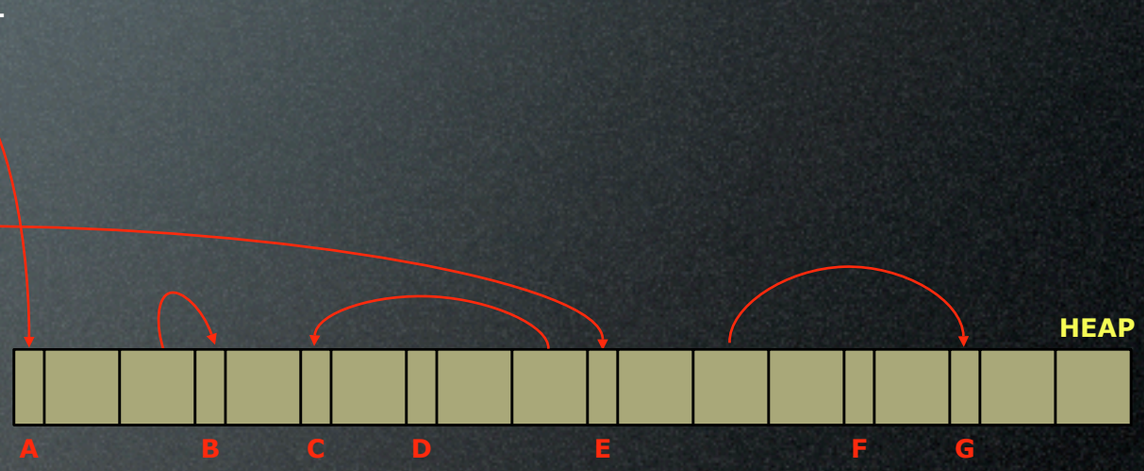
Sidebar: Garbage Collection

- Copying collector
  - split the heap in half, only one half is used each time
  - when the half is full, migrate surviving objects to the other half then allocate new objects from there



Sidebar: Garbage Collection

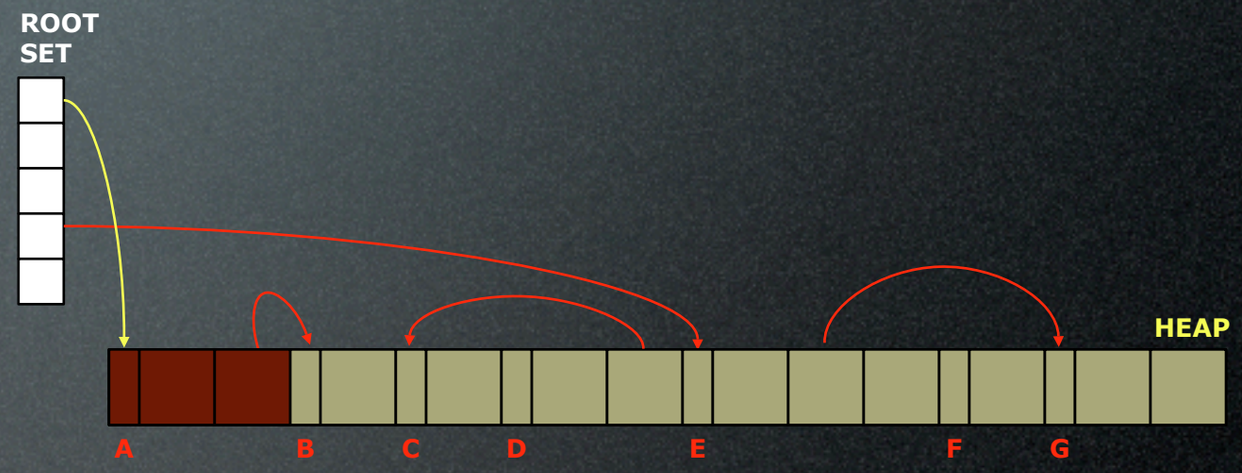
- Mark-sweep collector
  - collect the entire heap each time
  - when the heap is full, identify live objects (marking), then free dead objects (sweeping)



22

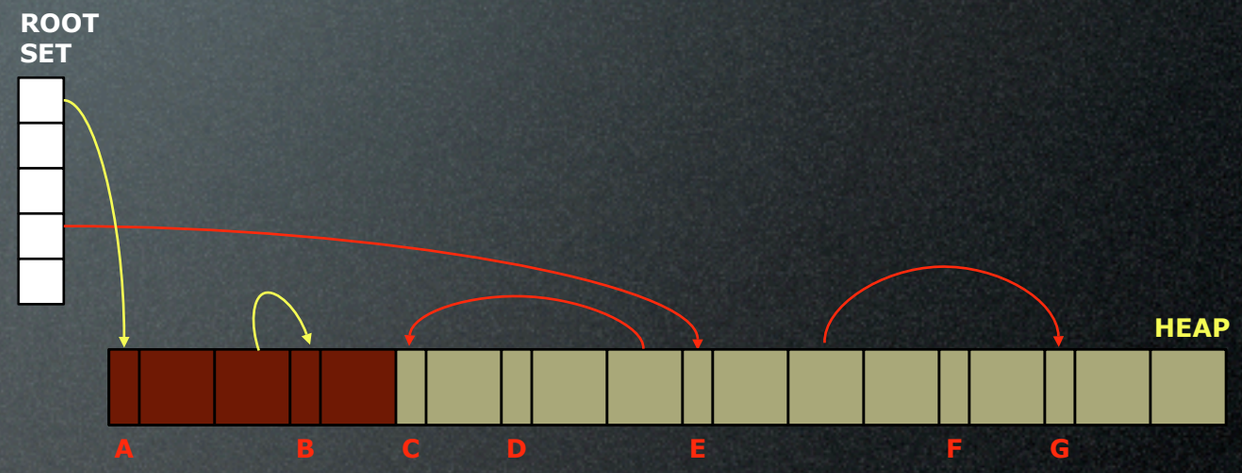
Sidebar: Garbage Collection

- Mark-sweep collector
  - collect the entire heap each time
  - when the heap is full, identify live objects (marking), then free dead objects (sweeping)



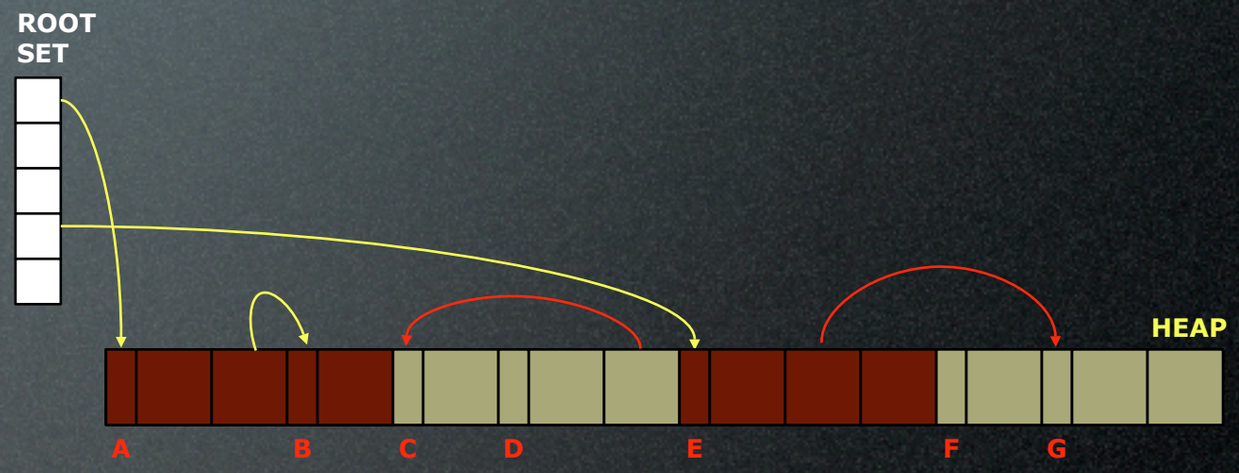
Sidebar: Garbage Collection

- Mark-sweep collector
  - collect the entire heap each time
  - when the heap is full, identify live objects (marking), then free dead objects (sweeping)



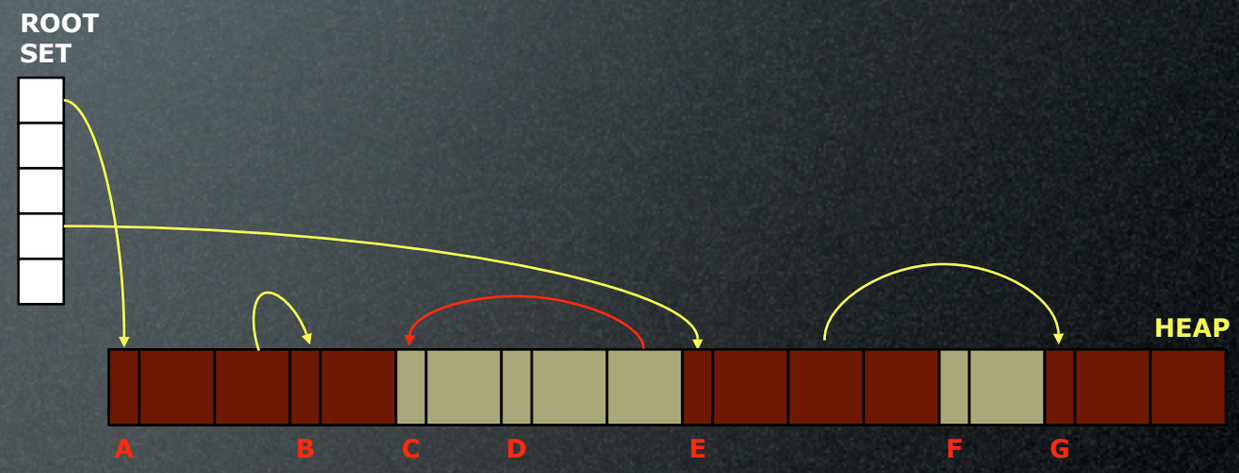
Sidebar: Garbage Collection

- Mark-sweep collector
  - collect the entire heap each time
  - when the heap is full, identify live objects (marking), then free dead objects (sweeping)



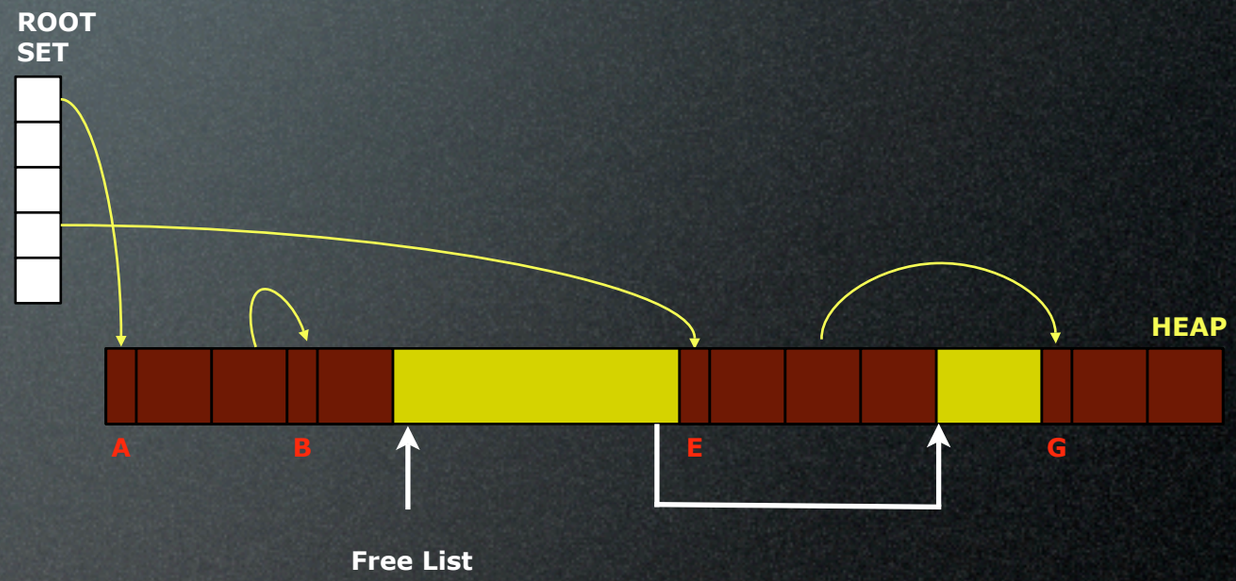
Sidebar: Garbage Collection

- Mark-sweep collector
  - collect the entire heap each time
  - when the heap is full, identify live objects (marking), then free dead objects (sweeping)



Sidebar: Garbage Collection

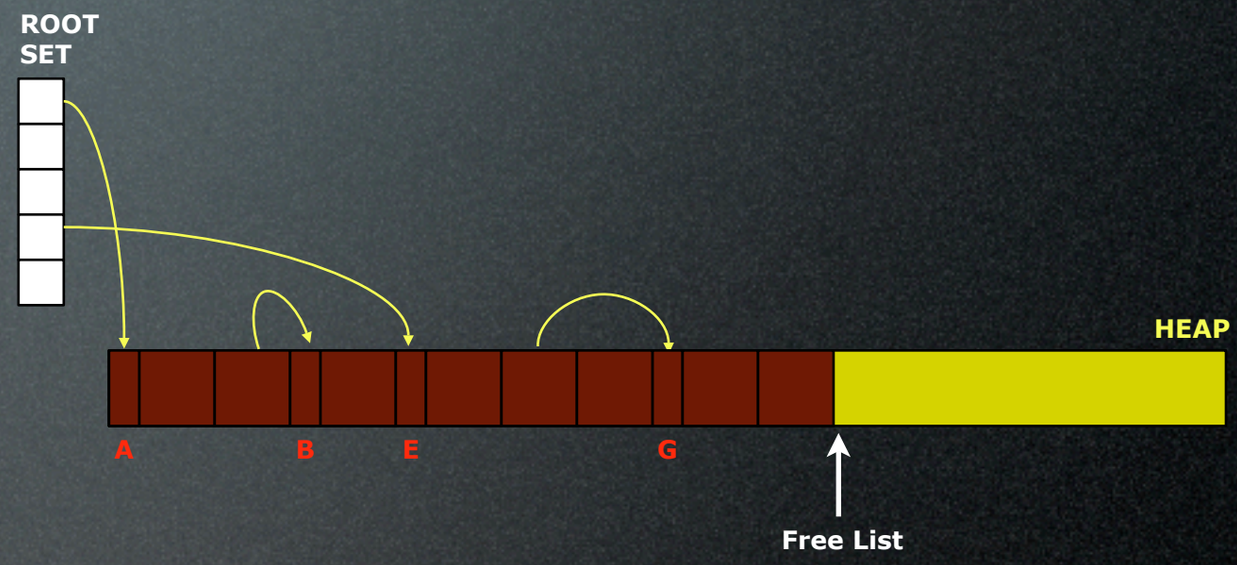
- Mark-sweep collector
  - collect the entire heap each time
  - when the heap is full, identify live objects (marking), then free dead objects (sweeping)



23

Sidebar: Garbage Collection

- Mark-sweep-compact collector
  - collect the entire heap each time
  - similar to mark-sweep collector except that heap is compacted after sweeping



24

Java Virtual Machines

- But typical JVMs are not ready for deployment in realtime embedded systems
  - lacking real-time support
  - unpredictable execution of operations
  - no support for real-time threads
  - no priority inversion avoidance
  - unbounded garbage collection

25

Java Virtual Machines

- But typical JVMs are not ready for deployment in realtime embedded systems

These topics will be the focus of the next few lectures

- unpredictable execution of operations
- no support for real-time threads
- no priority inversion avoidance
- unbounded garbage collection

26