

# Föreläsning 15

---

Tobias Wrigstad

*Imperativ och objekt-  
orienterad programmering*



# Vad är objektorientering?

---

- Ett tankesätt

Allt är "objekt" (vad är ett objekt?)

Ett program består av samverkande objekt

- En av många definitioner

**Inkapsling av tillstånd och beteende**

**Meddelandesändning**

**Polymorfism**

**Dynamisk bindning**

- Många definitioner innefattar även **arv** (vi väntar med det)

# Det objektorienterade tankesättet

---

- Vilka är aktörerna i systemet?
- Vad finns det för sammanhang/beroenden/interaktion?
- Finns strukturella samband?
- Programmet som en modell av verkligheten

# Exempel: personobjekt

---

- En person har ett namn, ett personnummer, en ålder, en partner och ett civilstånd
- En ogift person kan gifta sig med en annan ogift person

```
typedef struct person person_t;
```

```
struct person
{
    // State
    char          *name;
    social_sec_no_t  ssn;
    int           age;
    person_t        *spouse;
    marital_status_t status;

    // Behaviour
    result_t        (*set_spouse)      (person_t *, person_t *);
    person_t*       (*get_spouse)      (person_t *);
    marital_status_t (*marital_status) (person_t *);
};
```

```
#define SendMessage(who,msg,...) who->msg(who,__VA_ARGS__)
```



```
#define SendMessage(r,m,...) r->m(r,__VA_ARGS__)
```

```
SendMessage(kim, set_spouse, robin);
```



*Expanderar till*

```
kim->set_spouse(kim, robin);
```



```
result_t old_school_set_spouse(person_t *current, person_t *spouse)
{
    assert(current);

    if (SendMessage(current, marital_status, MARRIED)) return Already_married;
    if (SendMessage(spouse, marital_status, MARRIED)) return Spouse_already_married;

    current->spouse = spouse;
    spouse->spouse = current;

    SendMessage(current, update_marital_status);
    SendMessage(spouse, update_marital_status);

    return Happily_married;
}
```



```
result_t this_is_the_one(person_t *current, person_t *spouse)
{
    assert(current);

    current->spouse = spouse;

    SendMessage(current, update_marital_status);
    SendMessage(spouse, update_marital_status);

    // Change behaviour!
    current->set_spouse = old_faithful_set_spouse;

    return Happily_married;
}
```

```
result_t old_faithful_set_spouse(person_t *current, person_t *spouse)
{
    // raise hell, some kind of punishment, etc.

    return No_way;
}
```





# Vad har vi just sett? [”Allt” utom arv och polymorfism]

---

## Inkapsling av tillstånd

Strukten i C-filen, ingen direkt åtkomst

## Inkapsling av beteende

Alla funktioner också i strukten — den styr sitt eget beteenden (*jmf. old\_faithful!*)

## Meddelandesändning

SendMessage — ”hörru Kim, slå dina påsar ihop med Robin här, va?”

## Dynamisk bindning

Vilken funktion som anropas av SendMessage(x,y,z) beror på vilken funktion x->y pekar på just nu

# Objektorienterad programmering

---

- Handlar om ett sätt att tänka
- Vi skall träna det, men vi skall också träna oss i att programmera **Java**

*Vi hade inte behövt byta språk, men Java inbjuder till objektorienterad programmering på ett sätt som C inte gör!*

- Det viktigaste är att sätta objekten först!

Vilka objekt har vi i vårt program — hur samverkar de?

# OO i Java

---

- Genealogi: Smalltalk, Objective-C, och i viss mån C och C++
- Pragmatisk OO: inte allt är ett objekt, t.ex. int, float (pga prestanda)
- Högnivå: automatisk minneshantering
- Enkelt implementationsarv
- Multipelt gränssnittsarv
- Java är syntaktiskt mycket likt C och C++

*Det är lätt att råka "programmera C" fast man programmerar i Java*

# Saker ni kommer att älska med Java

---

## **En riktig strängtyp (String)**

`System.console().readLine()` läser in en sträng i programmet

## **Automatisk minneshantering**

Inga malloc/calloc/realloc/free, inga minnesläckage, ingen valgrind

## **Inga segfaults**

Exceptions med stacktraces istället, som om ni redan körde i gdb

## **Det fantastiska standardbiblioteket**

Alla listor, träd, etc., nätverksockets, URL:er, XML, etc. finns redan färdiga

# Inga minnesläckage

---

```
String stupid() {  
    String a = "Hello";  
    String b = " ";  
    String c = "World";  
    String d = a + b;  
    String e = d + c;  
    return e;  
}
```

Alla strängar utom e kommer automatiskt att frigöras!

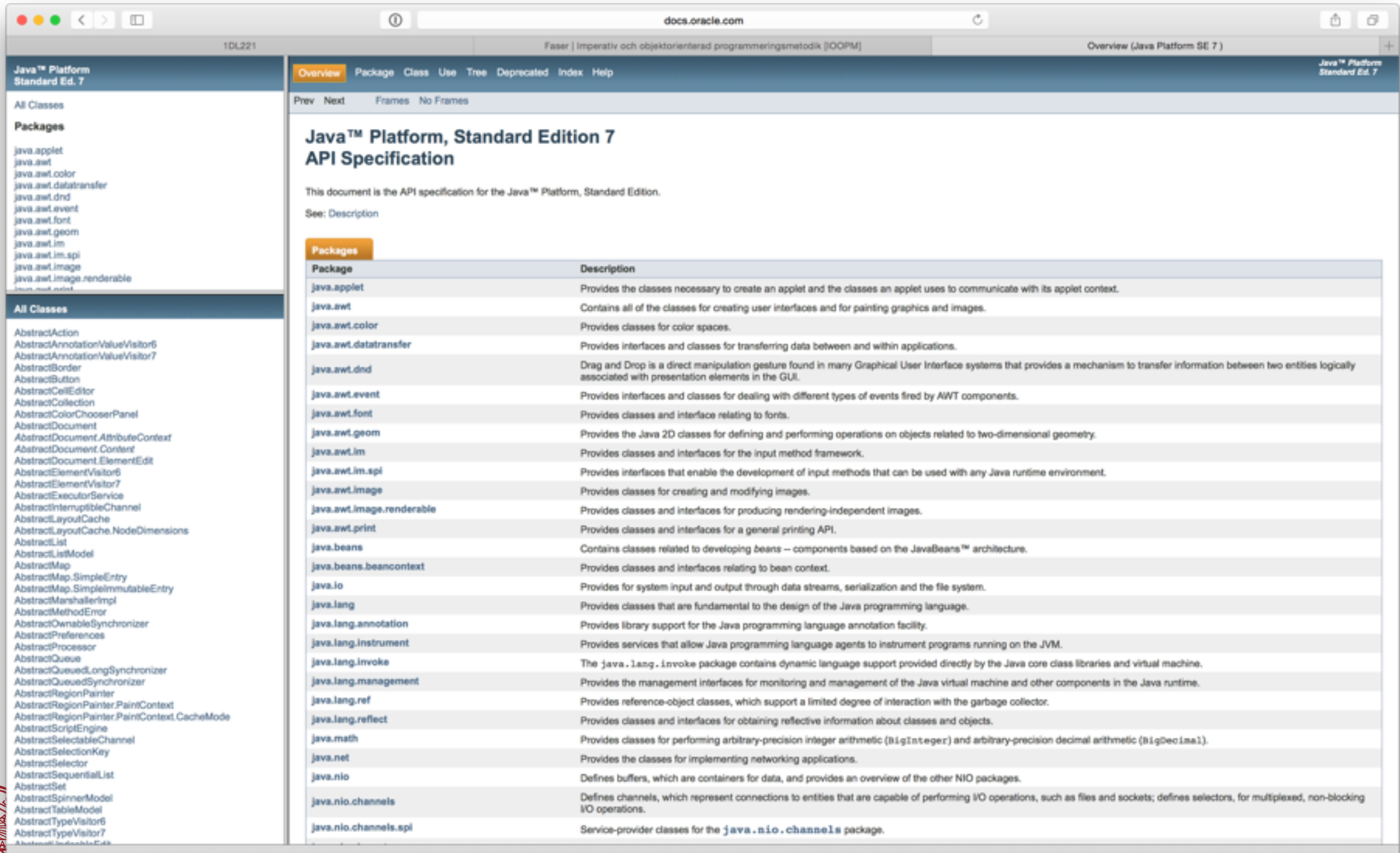
# Avrefererad nullpekare i Java

---

```
Exception in thread "main" java.lang.NullPointerException
    at com.example.myproject.Book.getTitle(Book.java:16)
    at com.example.myproject.Author.getBookTitles(Author.java:25)
    at com.example.myproject.Bootstrap.main(Bootstrap.java:14)
```

Notera att programmet skriver ut vilken rad det kraschade på!

<http://docs.oracle.com/javase/7/docs/api/>

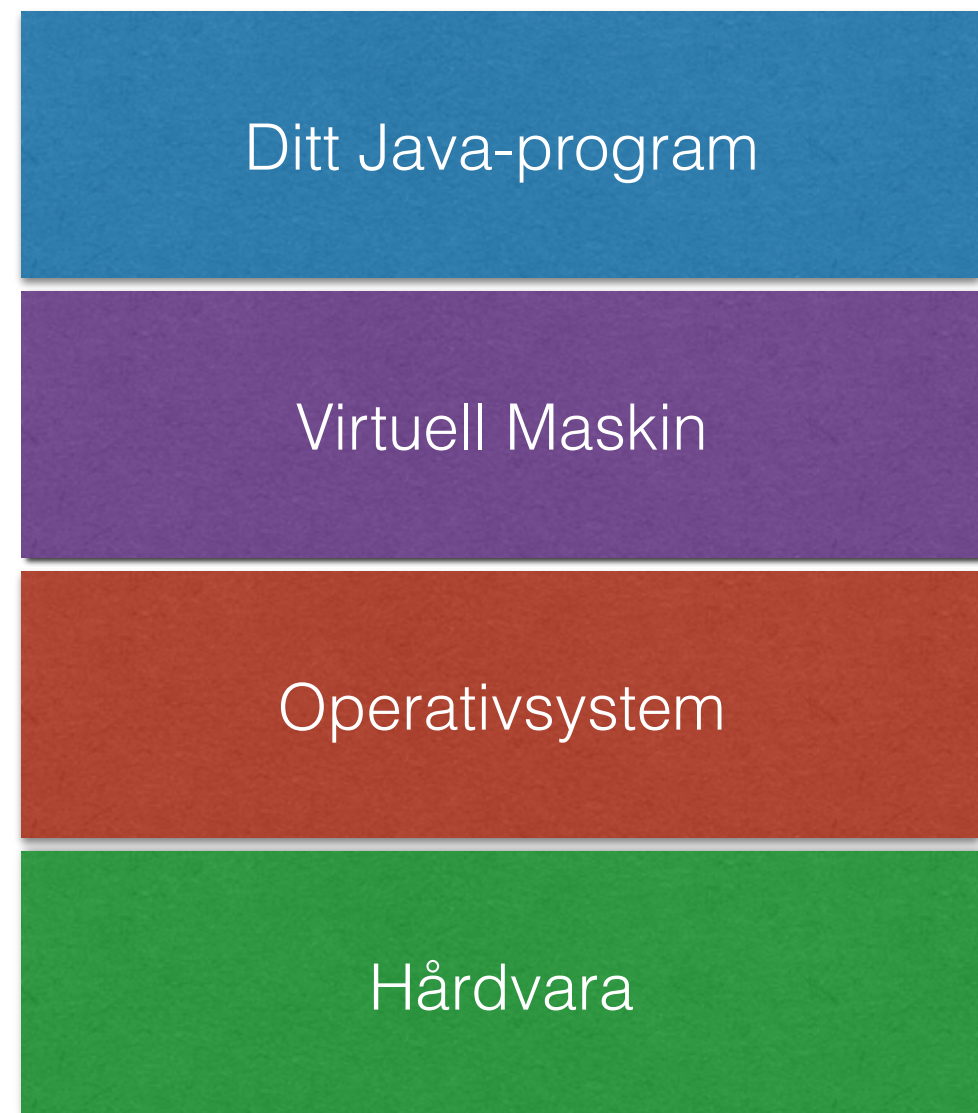
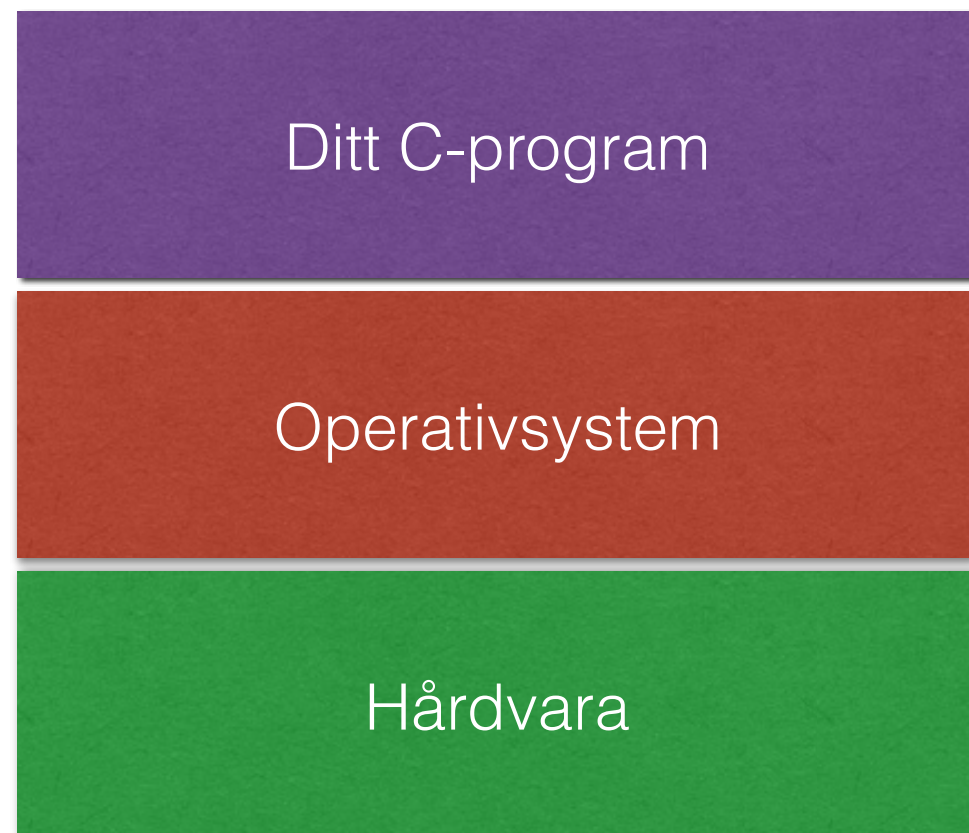


The screenshot displays the Java Platform, Standard Edition 7 API Specification page. The browser address bar shows <http://docs.oracle.com/javase/7/docs/api/>. The page title is "Java™ Platform, Standard Edition 7 API Specification". The sidebar on the left lists various packages under "All Classes". The main content area features a table with the following columns: "Package" and "Description".

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
java.awt.image	Provides classes for creating and modifying images.
java.awt.image.renderable	Provides classes and interfaces for producing rendering-independent images.
java.awt.print	Provides classes and interfaces for a general printing API.
java.beans	Contains classes related to developing beans -- components based on the JavaBeans™ architecture.
java.beans.beancontext	Provides classes and interfaces relating to bean context.
java.io	Provides for system input and output through data streams, serialization and the file system.
java.lang	Provides classes that are fundamental to the design of the Java programming language.
java.lang.annotation	Provides library support for the Java programming language annotation facility.
java.lang.instrument	Provides services that allow Java programming language agents to instrument programs running on the JVM.
java.lang.invoke	The <code>java.lang.invoke</code> package contains dynamic language support provided directly by the Java core class libraries and virtual machine.
java.lang.management	Provides the management interfaces for monitoring and management of the Java virtual machine and other components in the Java runtime.
java.lang.ref	Provides reference-object classes, which support a limited degree of interaction with the garbage collector.
java.lang.reflect	Provides classes and interfaces for obtaining reflective information about classes and objects.
java.math	Provides classes for performing arbitrary-precision integer arithmetic ( <code>BigInteger</code> ) and arbitrary-precision decimal arithmetic ( <code>BigDecimal</code> ).
java.net	Provides the classes for implementing networking applications.
java.nio	Defines buffers, which are containers for data, and provides an overview of the other NIO packages.
java.nio.channels	Defines channels, which represent connections to entities that are capable of performing I/O operations, such as files and sockets; defines selectors, for multiplexed, non-blocking I/O operations.
java.nio.channels.spi	Service-provider classes for the <code>java.nio.channels</code> package.

# Stacken i Java vs. Stacken i C

---





# Kompilera och köra ett Java-program

---

\$ javac MyProg.java

*Skapar en eller flera .class-filer  
varav en heter MyProg.class*

\$ java MyProg

*Startar den virtuella maskinen  
och laddar in MyProg och kör*

Ditt Java-program

Virtuell Maskin

Operativsystem

Hårdvara

# Om vi hininner — ett demo

---

Vi skriver en länkad lista i Java



# Tjuvtitt på nästa uppgifter

---

Kassakösimulering & Twitter-ish



```
tobiasw:twitterish/ (masterX) $ java Twitterish localhost 8080
```

[18:15:26]

```
tobiasw:twitterish/ (masterx) $
```

[18:15:25]

```
tobiasw:twitterish/ (masterX) $ java Server
```

[18:15:31]

```
!! Server listening for connections: 0.0.0.0/0.0.0.0:8080
```

```
!! Server got a connection from: /127.0.0.1:56242
```

```
!! Server listening for connections: 0.0.0.0/0.0.0.0:8080
```

```
>> Received: PostMessage
```

```
!!! Total number of posts at server1
```

```
>> Received: SyncRequest
```

```
Enter your user id (email address): tobiass.wrigstadgit.uu.se
```

Set your password:

```
Enter your user name: tobiass
```

```
Logging in new user tobiias.wrigstad@it.uu.se...
```

Received class Account message

**Actions:**

[P]ost message | [U]pdate feed | [A]dd friend | [R]emove friend |

```
[I]gnore friend | [L]ist friends | [E]dit account | [Q]uit
```

F

Write your message on a single line:

Hello, twitterish!

Message sent

Actions:

[P]ost message | [U]pdate feed | [A]dd friend | [R]emove friend |

```
[I]gnore friend | [L]ist friends | [E]dit account | [Q]uit
```

1

```
Received class SyncResponse message
```

```
{tobias} says:
```

Hello, twitterish!

**Actions:**

[P]ost message | [U]pdate feed | [A]dd friend | [R]emove friend |

```
[I]gnore friend | [L]ist friends | [E]dit account | [Q]uit
```

1

```
bash-3.2$ java Simulator
```

# Skräpsamling 1/2

---



# Resten av denna föreläsning

---

- Hur fungerar malloc?
- Skräpsamlingsstrategier

Referensräkning

Tracing GC

# Vad händer egentligen?

---

```
void *p = malloc(2048);
```

```
free(p);
```



# Hantering av minnet: malloc & free

---

- Två länkade listor: en lista med lediga block (FL), en lista med upptagna block (UL)
- Vid allokering av N bytes, leta upp ett block av storlek  $M \geq N$  i FL, dela upp det i två delar A och B (A=N bytes, B=resten) och flytta A-delen till UL och låt B vara kvar i FL
- Vid frigörande av ett block, flytta det från UL till FL och slå samman eventuellt angränsande block
- Hur FL är sorterad är viktigt:

**Ökande storleksordning:** best fit, minsta möjliga fragmentering

**Minskande storleksordning:** snabb allokering (men mer fragmentering)

**Adressordning:** bättre lokalitet

# Automatisk skräpsamling

---

- Mål: att ge programmeraren en illusion att minnet är oändligt
- Metod: identifiera *skräp-data* och frigör det **automatiskt**
- Definitionen av skräp: data som inte kan nås av programmet
- Mer formellt: objektet O är skräp om det inte finns någon väg i minnesgrafén från något rot (variabeln på stacken, globala variabler, o.dyl.) till O
- Två grundläggande sätt att göra automatisk skräpsamling:

Referensräkning

Tracing

# Referensräkning

---

- Grundläggande idé: varje objekt sparar information om hur många som pekar till det
- När denna räknare når 0 — ta bort objektet
- Varje gång en referens skapas/tas bort, manipulera referensräknaren:

```
void *p = malloc(2048); // refcount 1
void *x = p; // refcount 2
p = NULL; // refcount 1
x = NULL; // refcount 0, free(x)
```

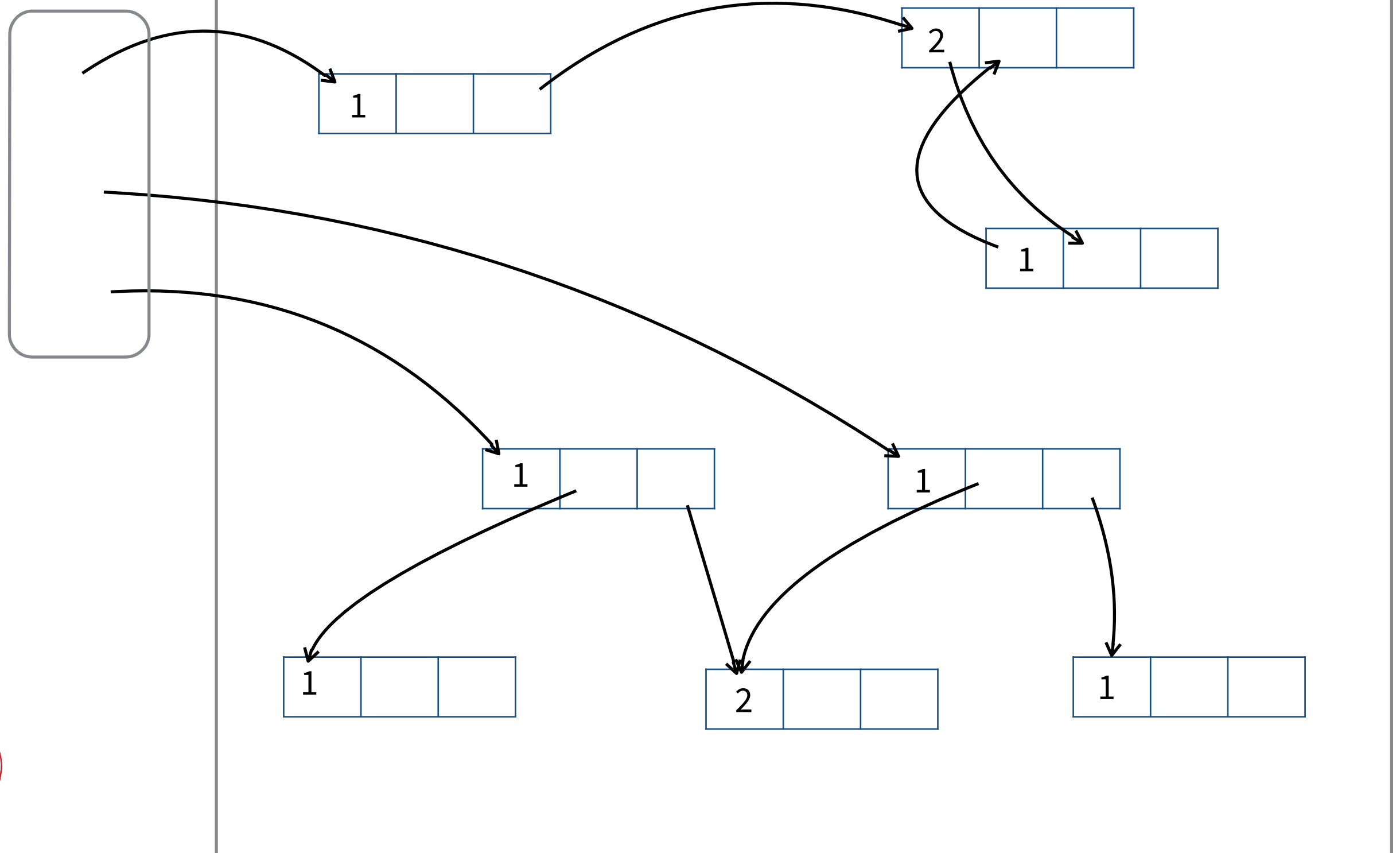
- Problem:

Cykliska strukturer (se nästföljande sidor)

Långlivat minne som manipuleras ofta kostar, fast vi aldrig tar bort det

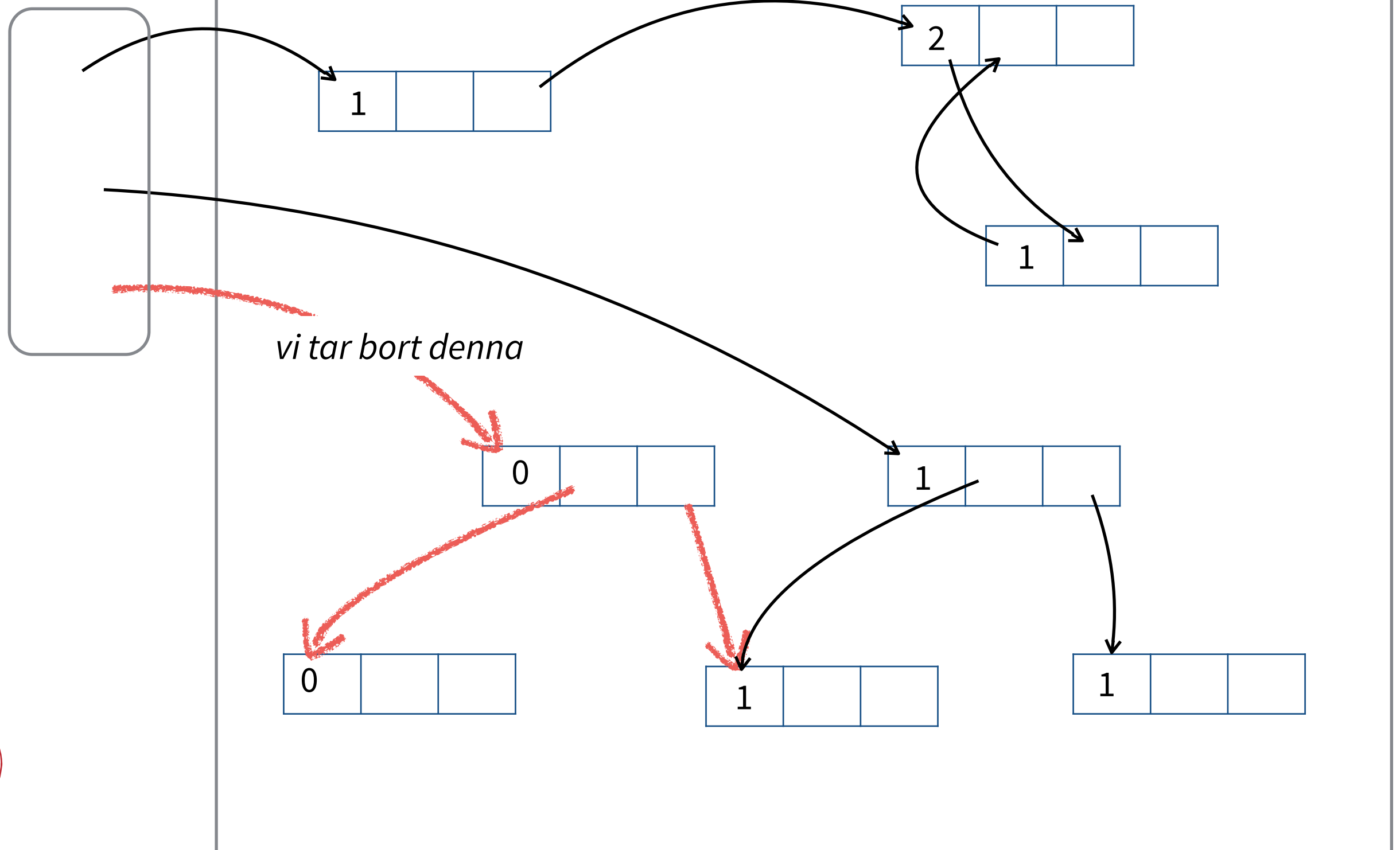
Heap

Root set



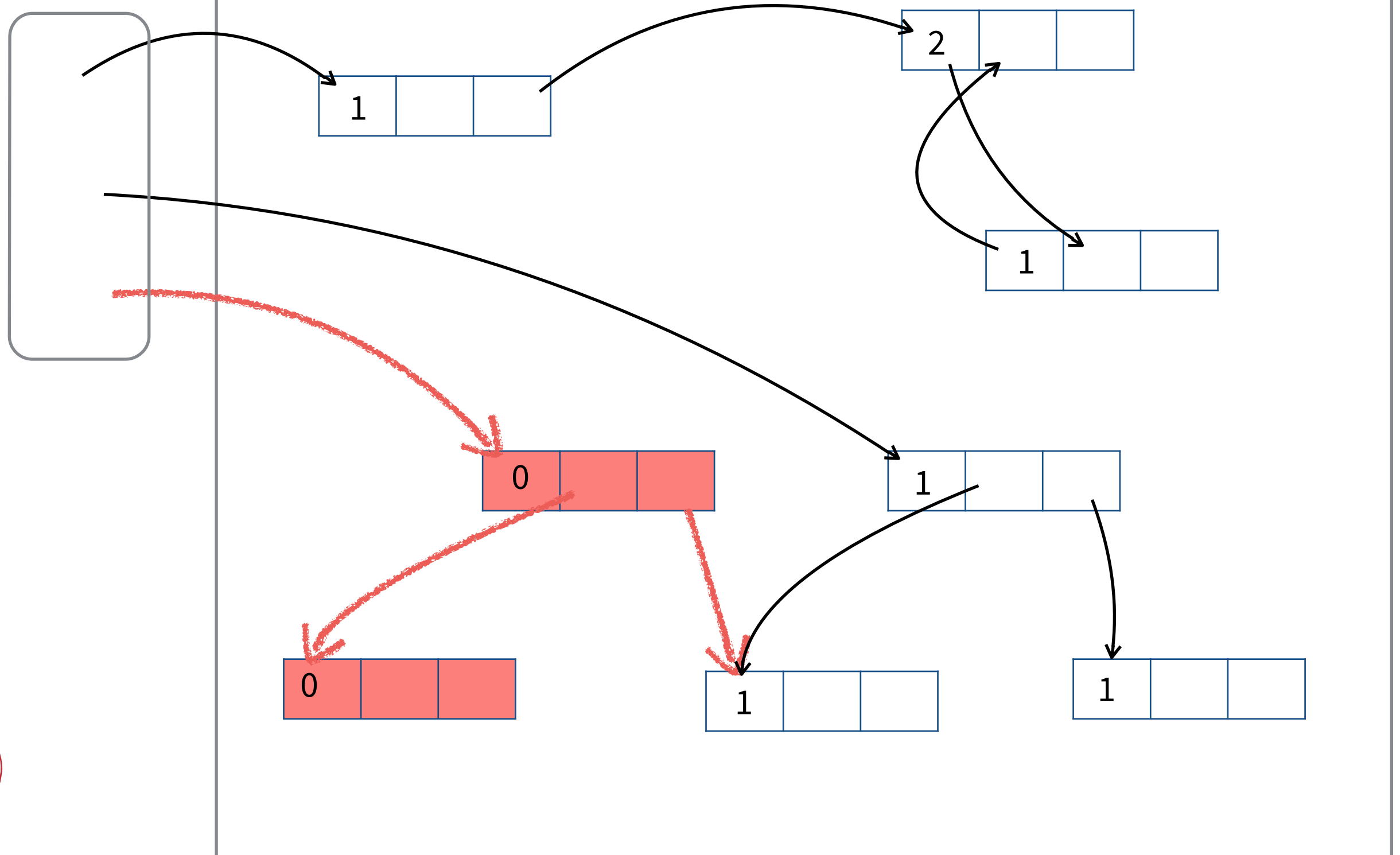
Heap

Root set



Heap

Root set

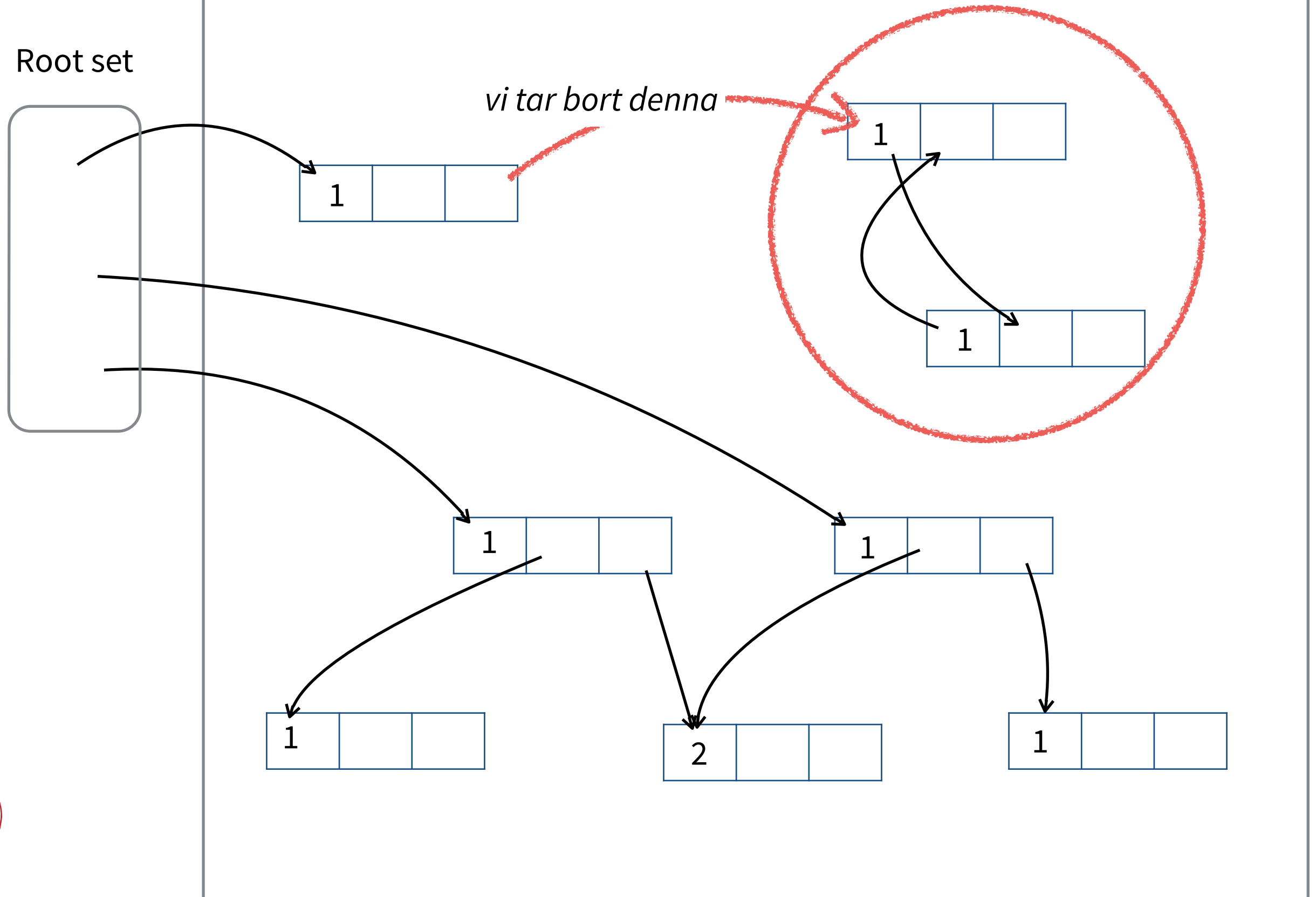


Heap

Läckage!

Root set

*vi tar bort denna*



# Tracing GC: Mark-Sweep

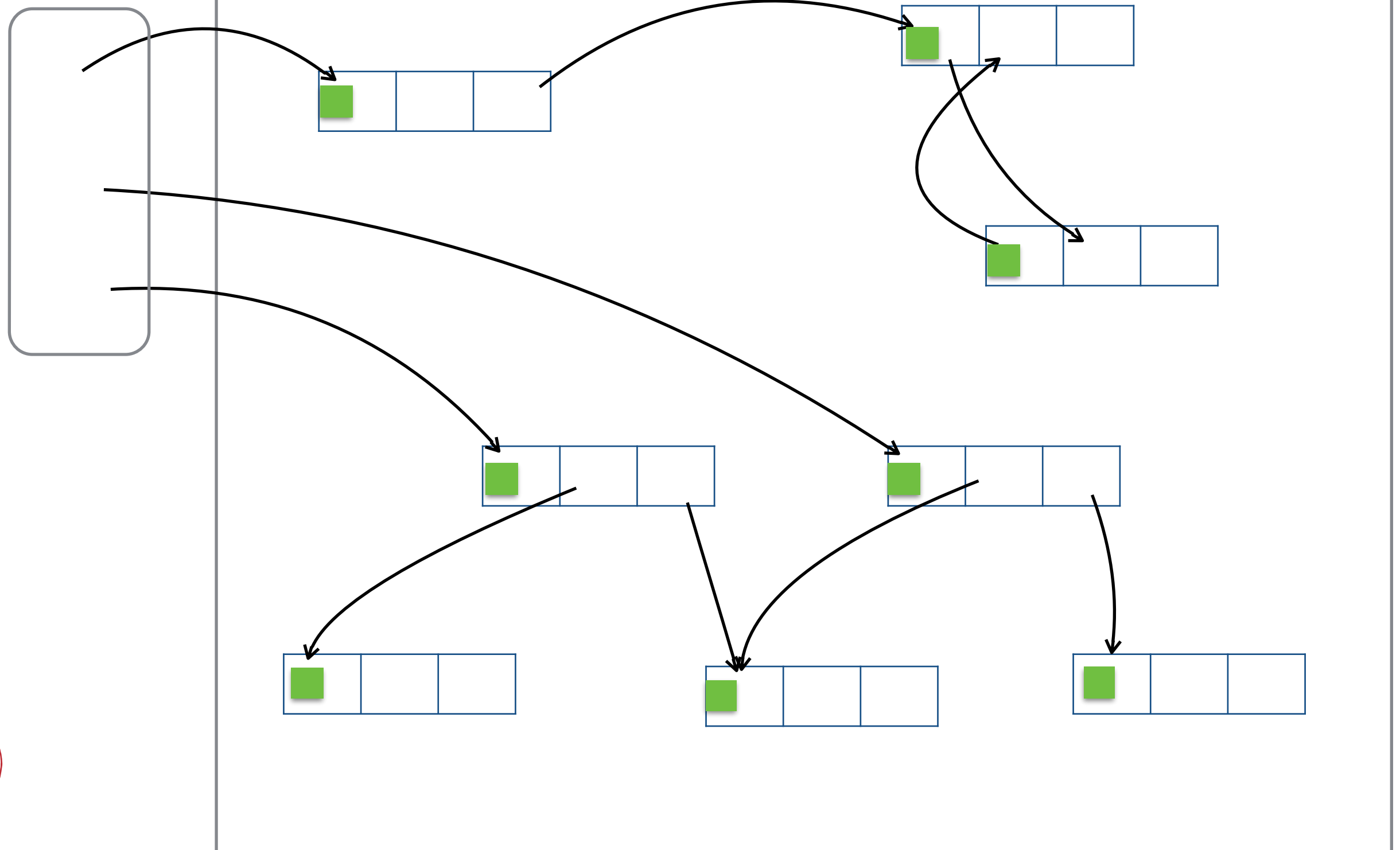
---

- När minnet tar slut:
  1. Följ rötterna och markera alla objekt som kan nå
  2. Iterera över alla objekt och frigör alla som inte markerats i 1.
- Nästa bild visar markering i *mark-fasen* (1.)



Heap

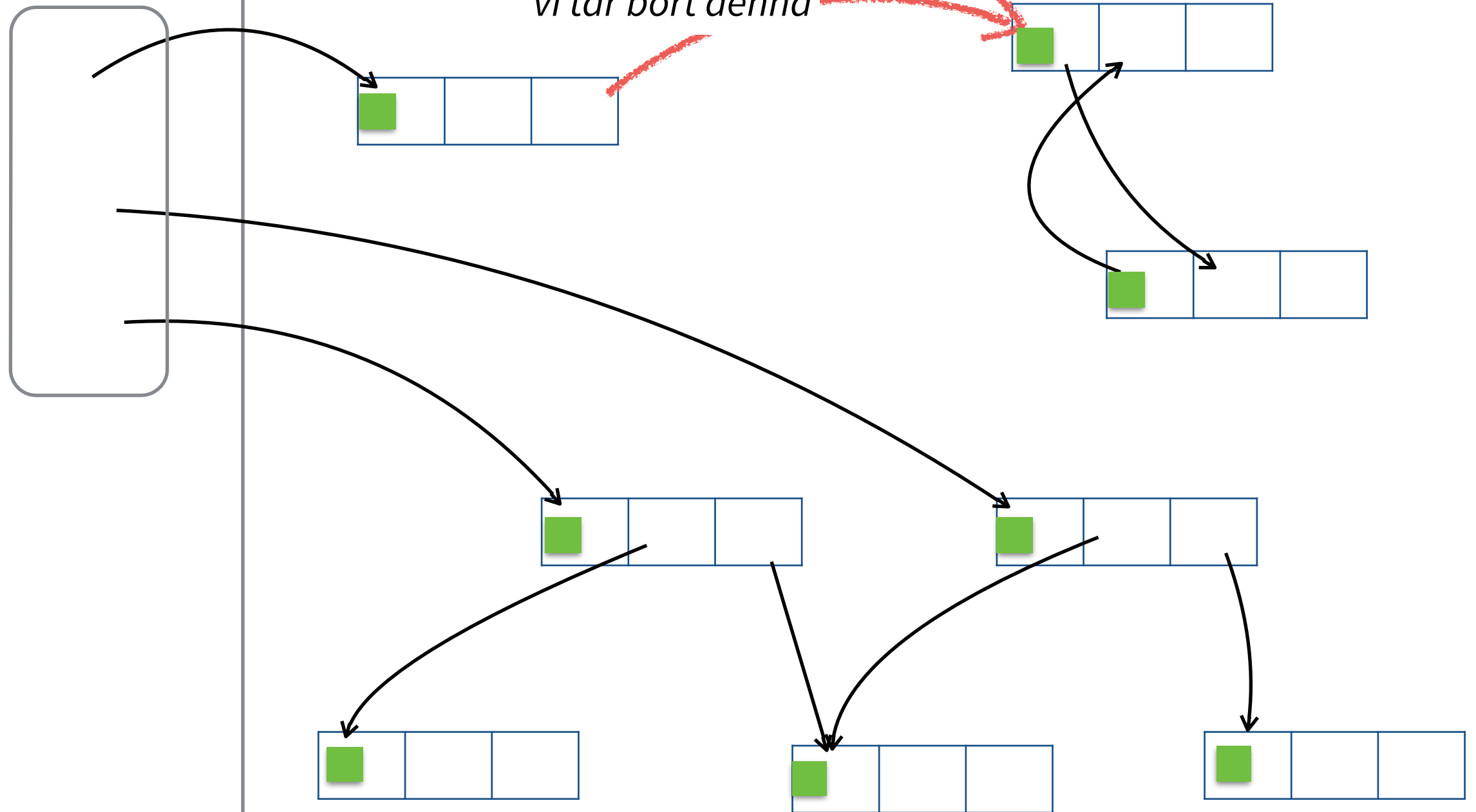
Root set



Heap

Root set

*vi tar bort denna*



# I nästa mark-fas markerar vi med annan färg

---



- Om något är grönt fortfarande efter denna fas är det skräp och skall tas bort

Heap

Root set

*Dessa två kan nu säkert tas bort*

