Lecture 12

Operating System

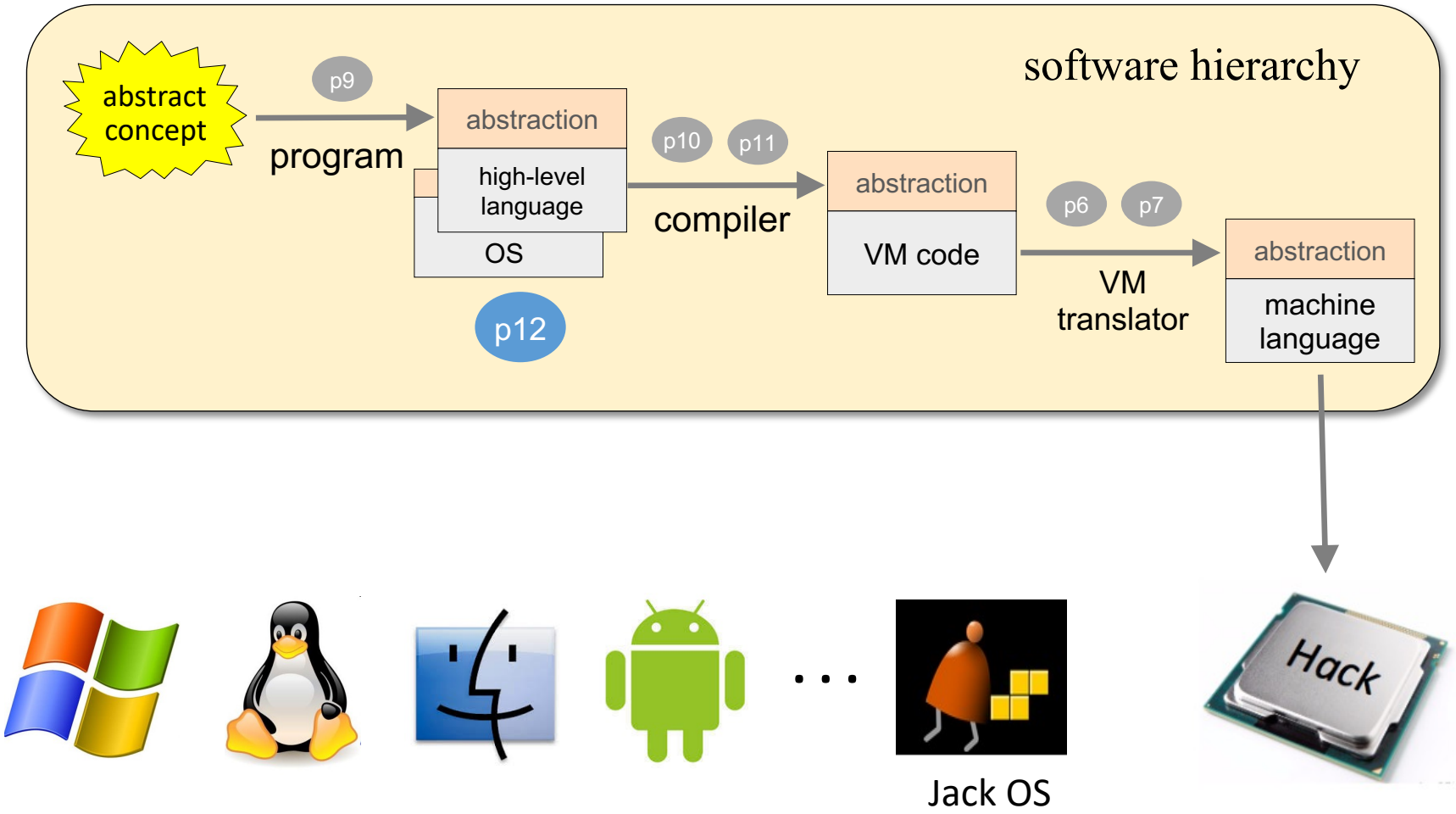
These slides support chapter 12 of the book

The Elements of Computing Systems

By Noam Nisan and Shimon Schocken

MIT Press, 2021

Nand to Tetris Roadmap: Part II



OS services

Typical program

```
/* Computes the length of the hypotenuse in a right triangle */
class Main {
  function void main() {
     var int a;
     var int b;
     var int c;
     let a = Keyboard.readInt("Enter the length of side 1: ");
     let b = Keyboard.readInt("Enter the length of side 2: ");
     do Output.printString("The hypotenuse length is: ");
     do Output.printInt(Main.hypot(a,b));
     return;
  }

  function int hypot(int x, int y) {
     return Math.sqrt((x*x) + (y*y));
  }
}
```

OS services

Typical program

```
/* Computes the length of the hypotenuse in a right triangle */
class Main {
  function void main() {
     var int a;
     var int b;
     var int c;
     let a = Keyboard.readInt("Enter the length of side 1: ");
     let b = Keyboard.readInt("Enter the length of side 2: ");
     do Output.printString("The hypotenuse length is: ");
     do Output.printInt(Main.hypot(a,b));
     return;
  }

  function int hypot(int x, int y) {
     return Math.sqrt((x*x) + (y*y));
  }
}
```

OS services highlighted

OS services

Language extensions

- ✓ Mathematical operations
  (`abs`, `sqrt`, ...)
- ✓ Abstract data types
  (`String`, `Array`, ...)
- ✓ Input functions
  (`readChar`, `readLine` ...)
- ✓ Textual output
  (`printChar`, `printString` ...)
- ✓ Graphics output
  (`drawLine`, `drawCircle`, ...)

. . .

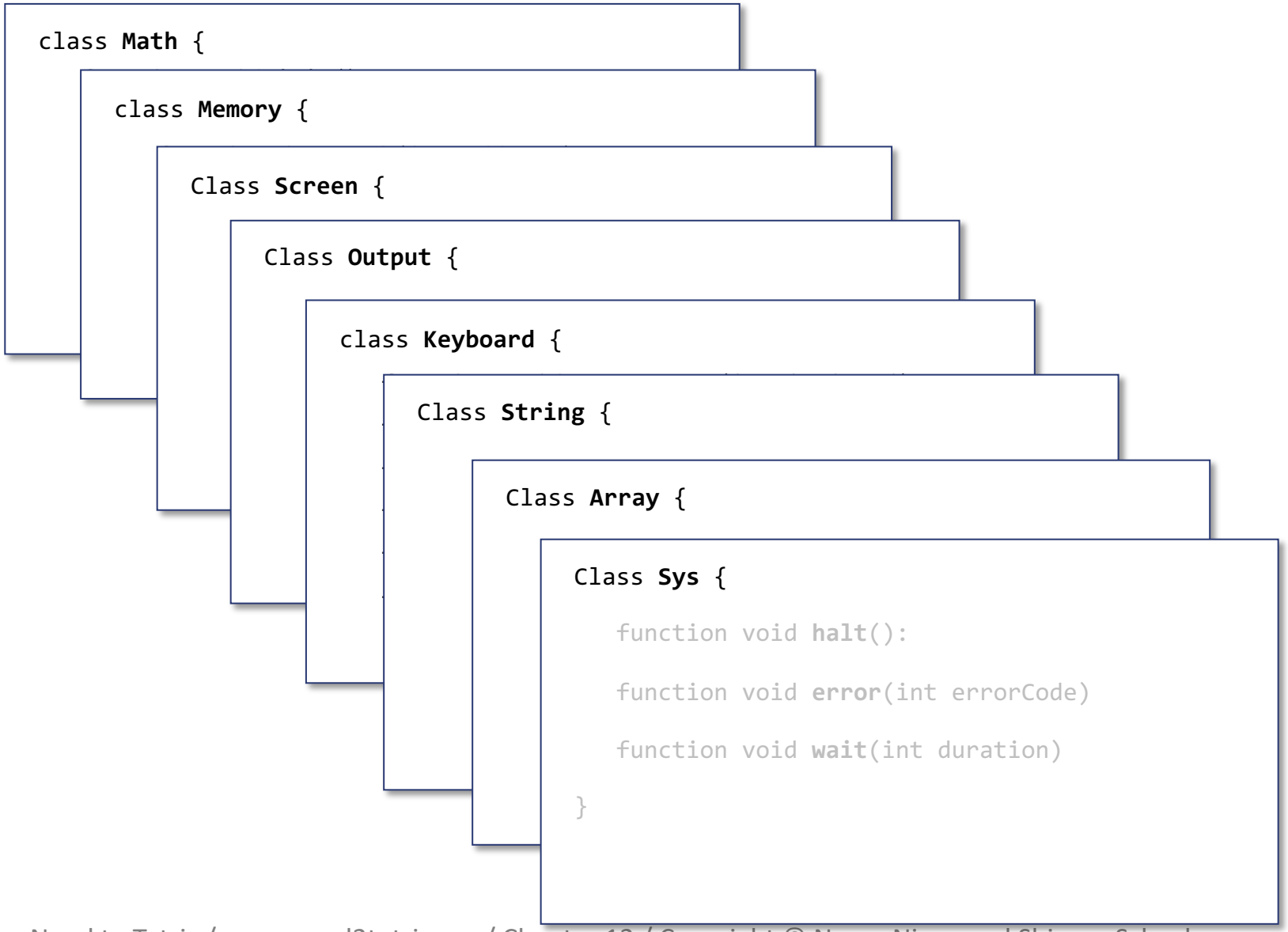
System services

- ✓ Memory management
  (objects, arrays, ...)
- ✓ I/O device drivers
- File system
- UI management
  (shell / windows)
- Multi-tasking
- Networking
- Security

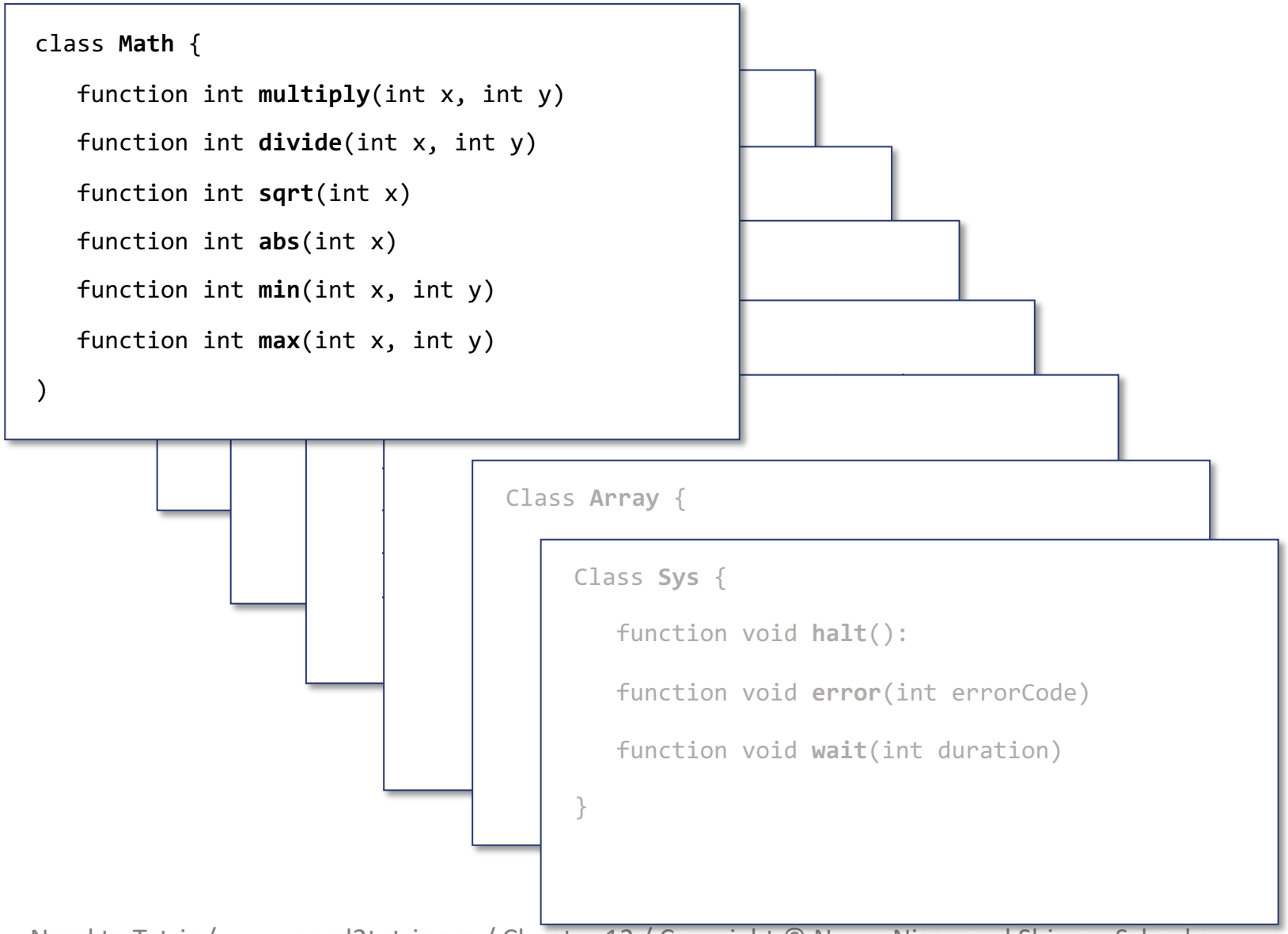
. . .

✓ : implemented in the Jack OS

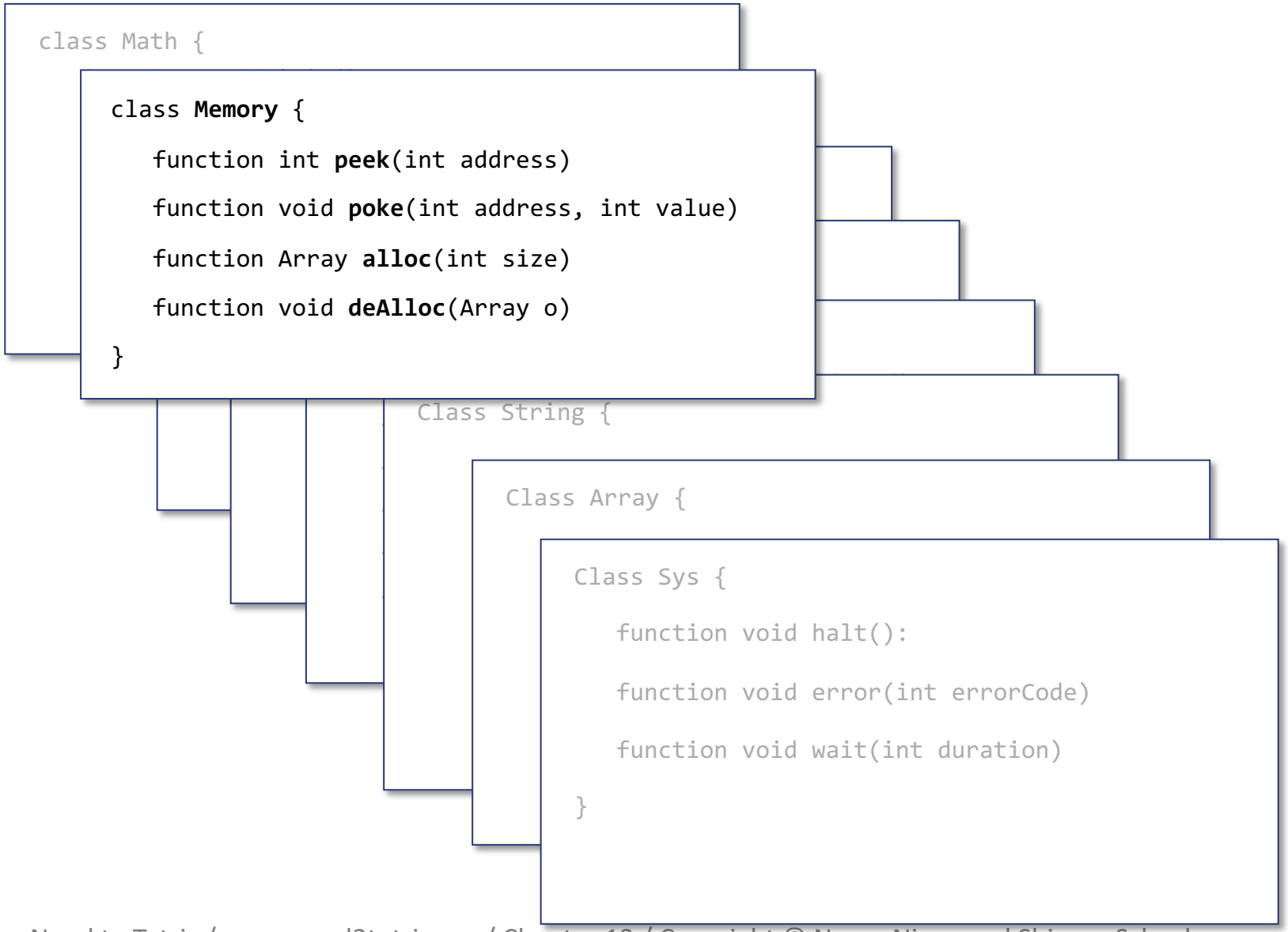
The Jack OS


class Math {
class Memory {
Class Screen {
Class Output {
class Keyboard {
Class String {
Class Array {
Class Sys {
function void halt():
function void error(int errorCode)
function void wait(int duration)
}

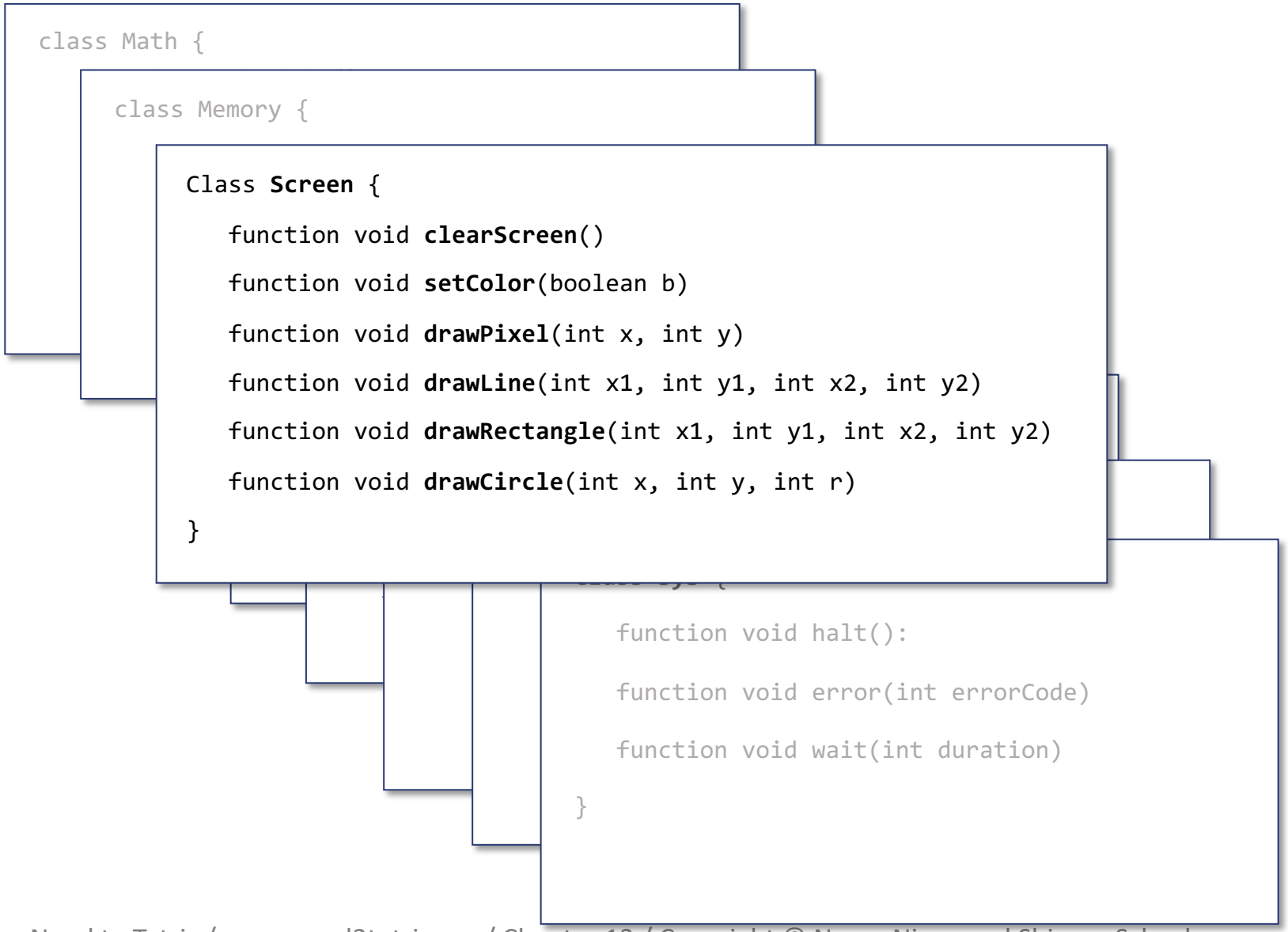
The Jack OS


class Math {
function int multiply(int x, int y)
function int divide(int x, int y)
function int sqrt(int x)
function int abs(int x)
function int min(int x, int y)
function int max(int x, int y)
)
Class Array {
Class Sys {
function void halt():
function void error(int errorCode)
function void wait(int duration)
}

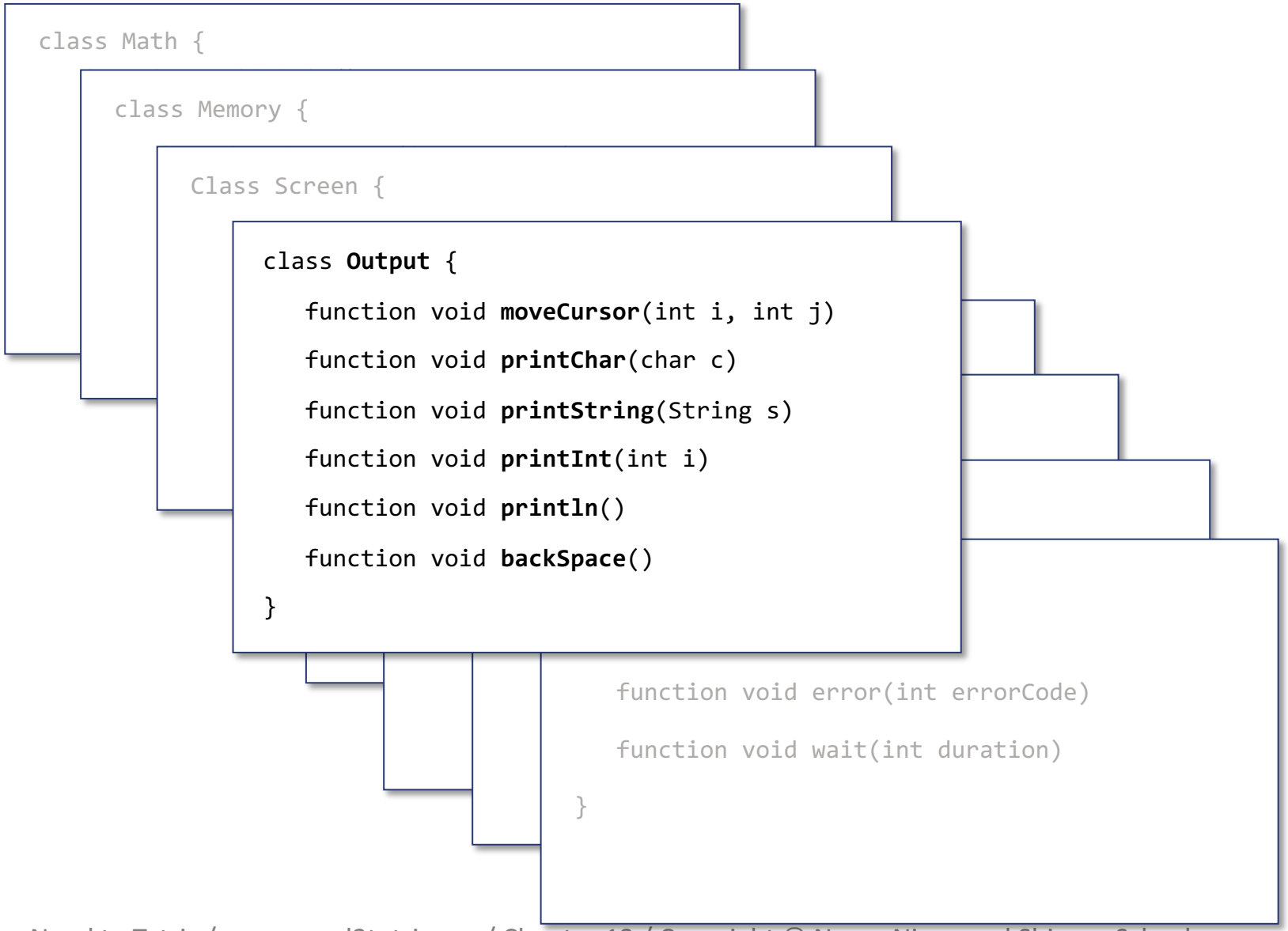
The Jack OS


class Math {
class Memory {
function int peek(int address)
function void poke(int address, int value)
function Array alloc(int size)
function void deAlloc(Array o)
}
Class String {
Class Array {
Class Sys {
function void halt():
function void error(int errorCode)
function void wait(int duration)
}

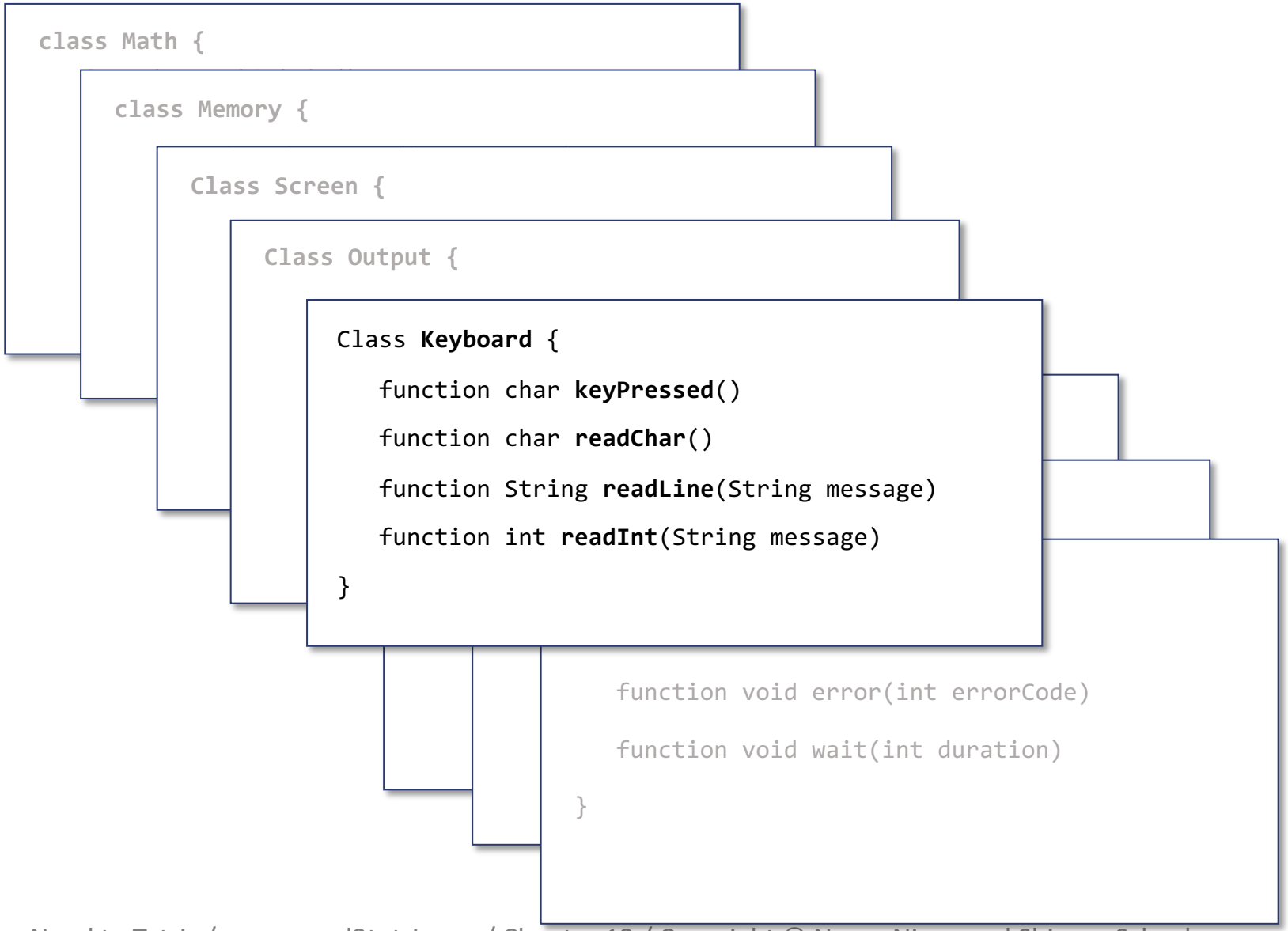
The Jack OS


class Math {
class Memory {
Class Screen {
function void clearScreen()
function void setColor(boolean b)
function void drawPixel(int x, int y)
function void drawLine(int x1, int y1, int x2, int y2)
function void drawRectangle(int x1, int y1, int x2, int y2)
function void drawCircle(int x, int y, int r)
}
function void halt():
function void error(int errorCode)
function void wait(int duration)
}

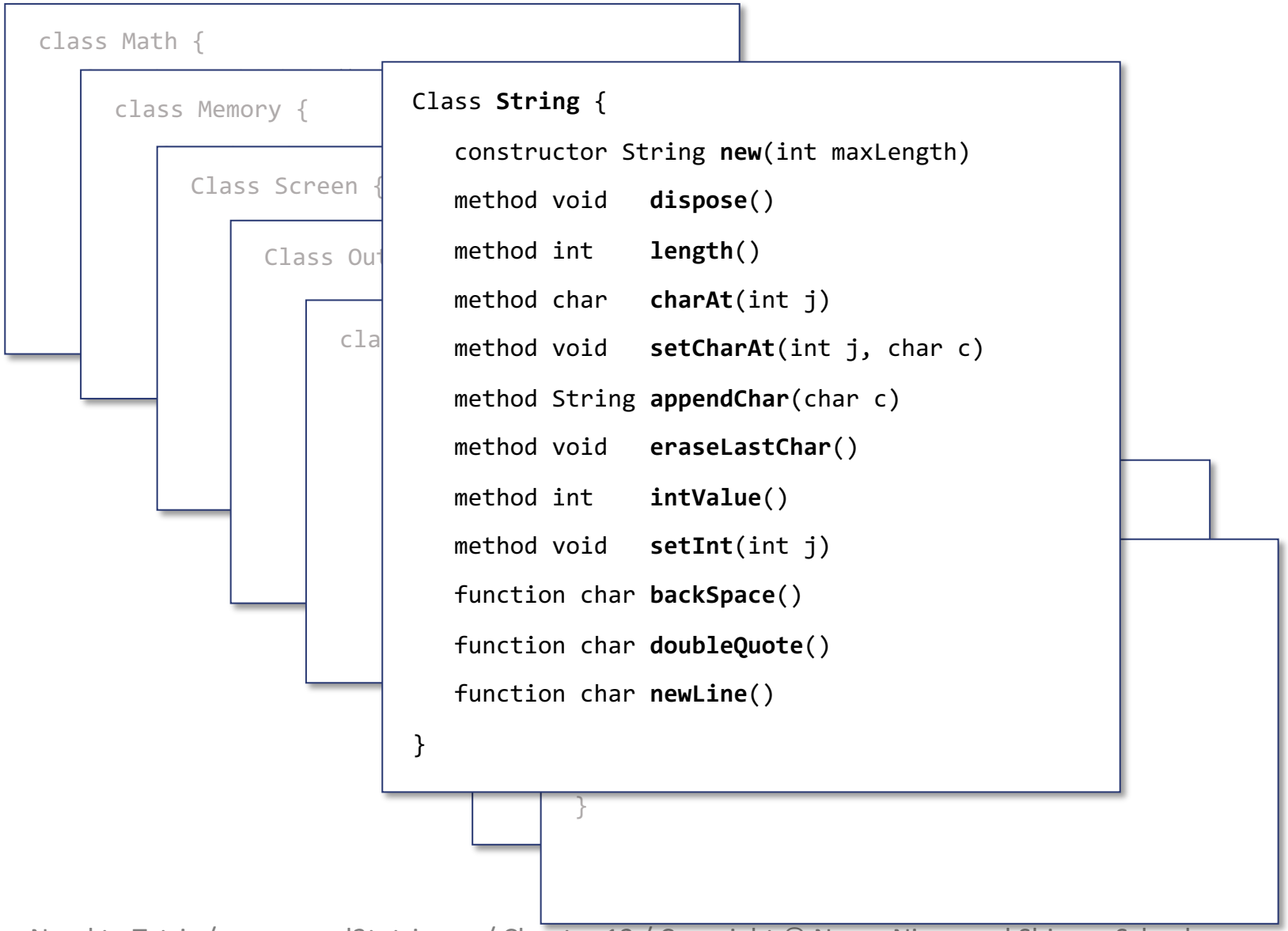
The Jack OS


class Math {
class Memory {
Class Screen {
class Output {
function void moveCursor(int i, int j)
function void printChar(char c)
function void printString(String s)
function void printInt(int i)
function void println()
function void backSpace()
}
function void error(int errorCode)
function void wait(int duration)
}

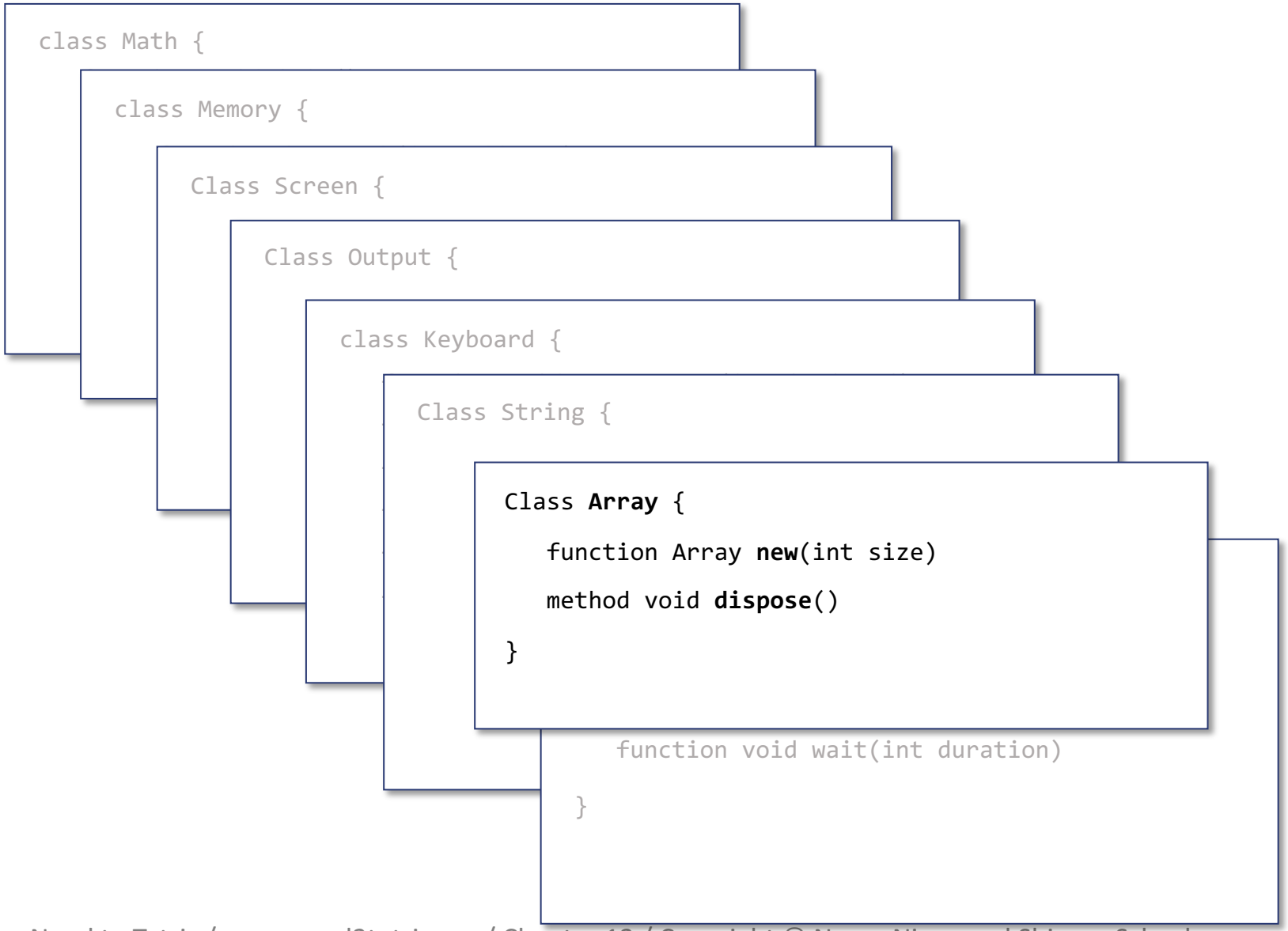
The Jack OS


class Math {
class Memory {
Class Screen {
Class Output {
Class Keyboard {
function char keyPressed()
function char readChar()
function String readLine(String message)
function int readInt(String message)
}
function void error(int errorCode)
function void wait(int duration)
}

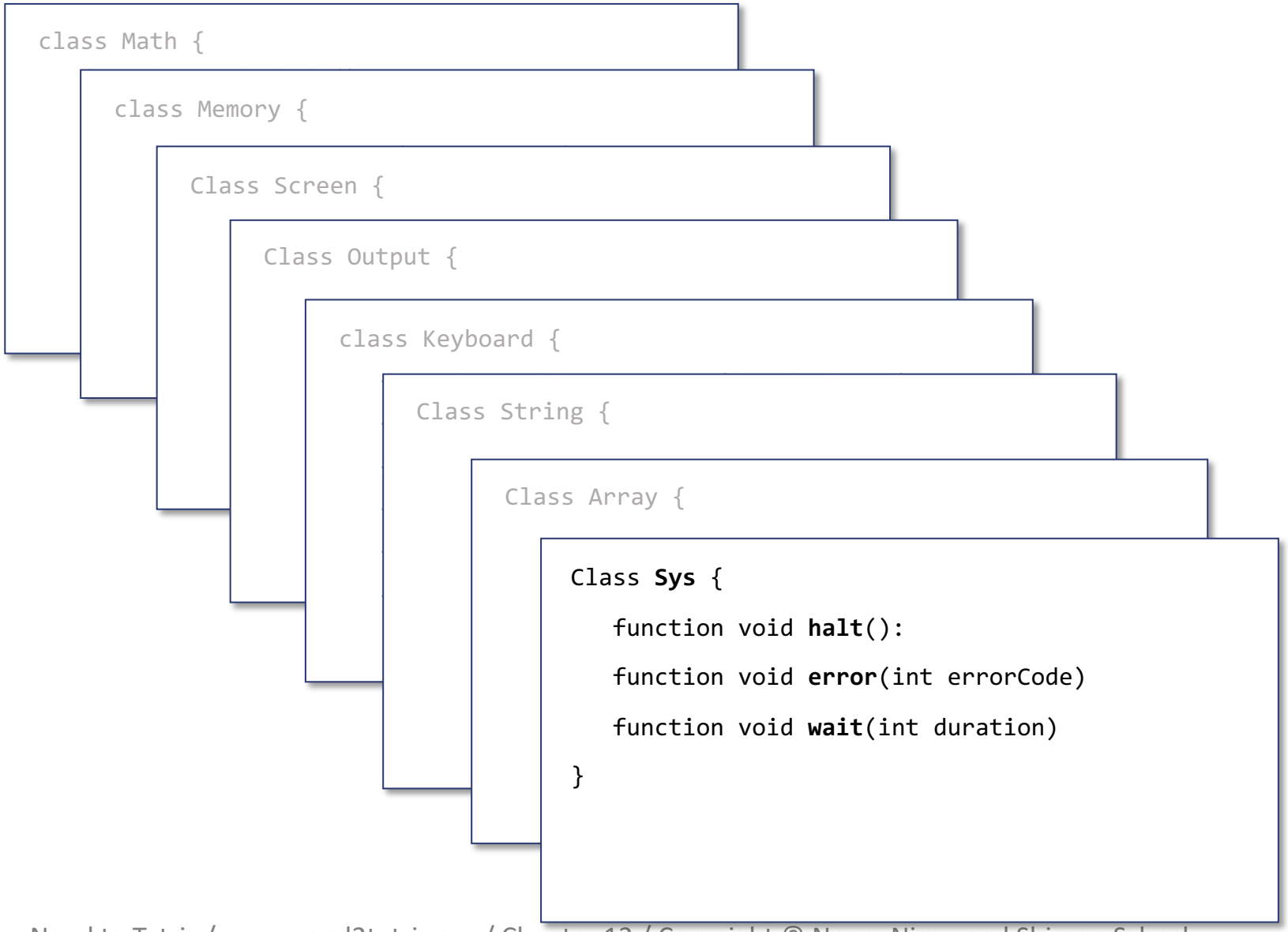
The Jack OS


class Math {
class Memory {
Class Screen {
Class Out
cla
Class String {
constructor String new(int maxLength)
method void dispose()
method int length()
method char charAt(int j)
method void setCharAt(int j, char c)
method String appendChar(char c)
method void eraseLastChar()
method int intValue()
method void setInt(int j)
function char backSpace()
function char doubleQuote()
function char newLine()
}
}

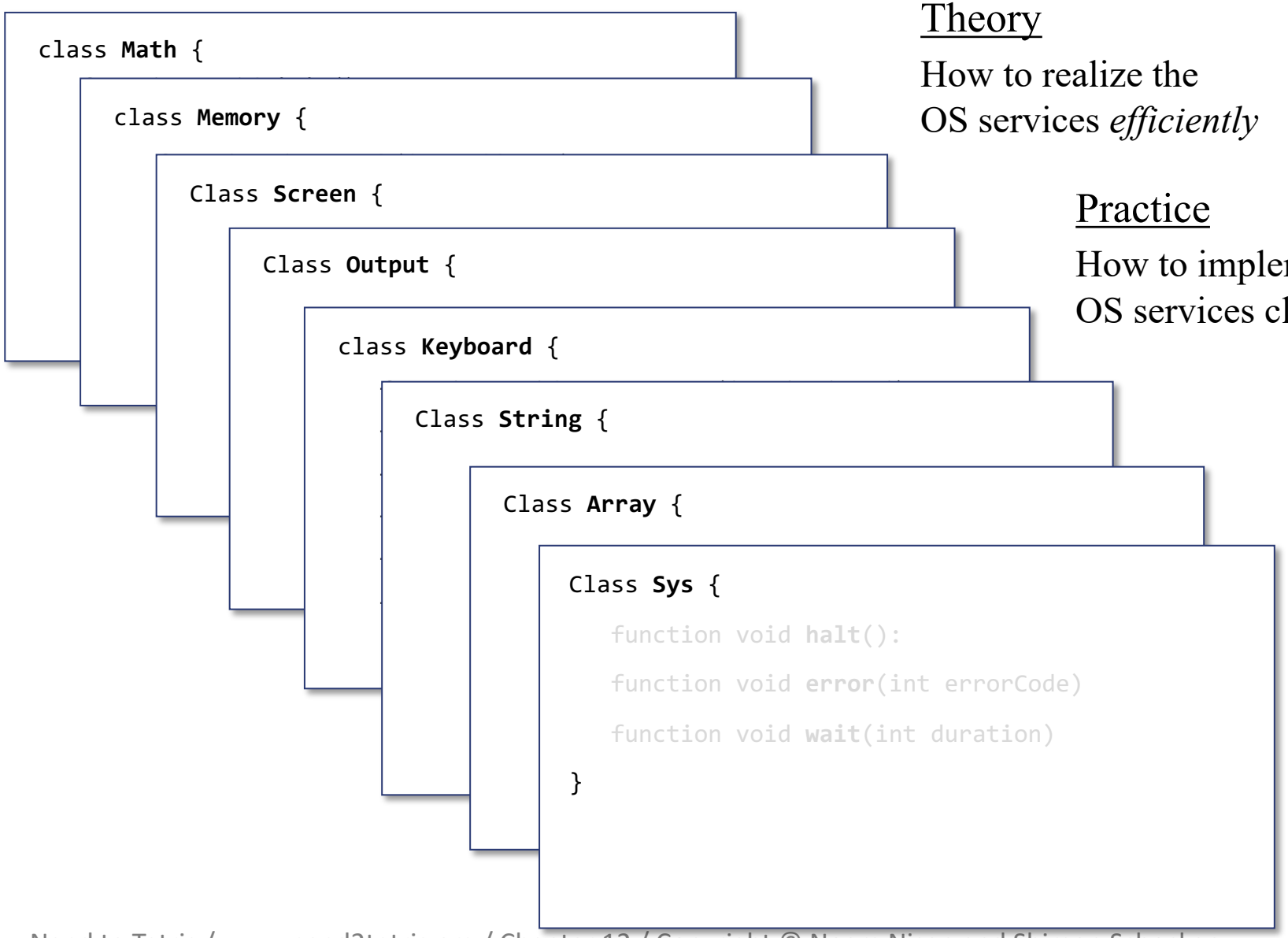
The Jack OS


class Math {
class Memory {
Class Screen {
Class Output {
class Keyboard {
Class String {
Class Array {
function Array new(int size)
method void dispose()
}
function void wait(int duration)
}

The Jack OS


class Math {
class Memory {
Class Screen {
Class Output {
class Keyboard {
Class String {
Class Array {
Class Sys {
function void halt():
function void error(int errorCode)
function void wait(int duration)
}

Take home lessons



Theory

How to realize the
OS services *efficiently*

Practice

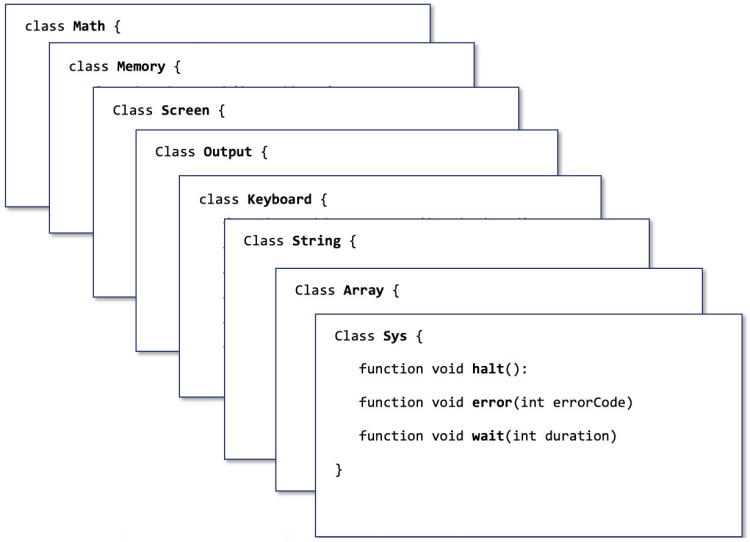
How to implement the
OS services cleverly

Take home lessons

Topics

- Time complexity
- Memory management
- Input handling
- Output handling
  - Characters / fonts
  - Graphics (bitmap / vector)
- String processing
- Type casting

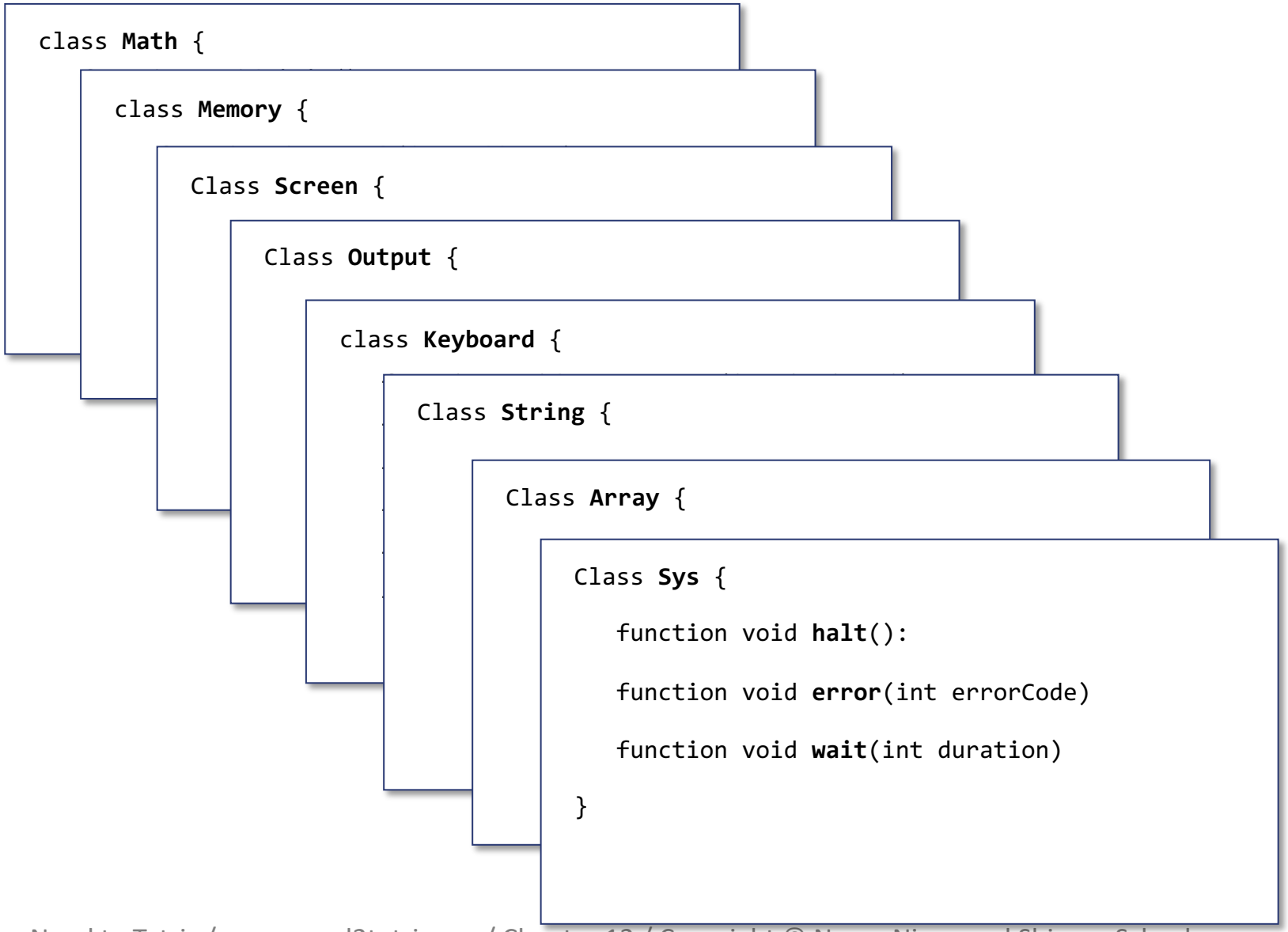
...



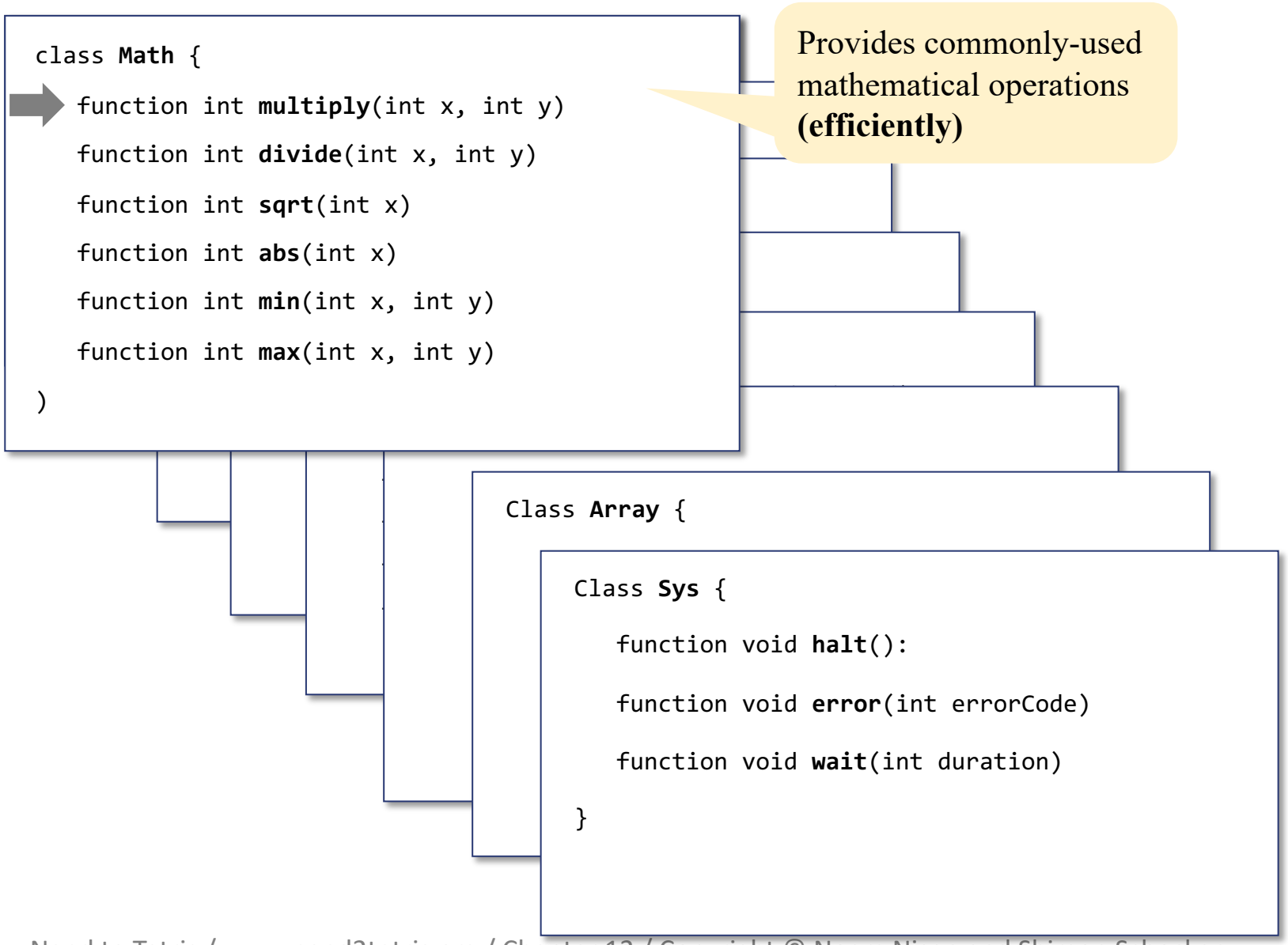
Methodology

- Gem algorithms
- Cool hacks
- Typical trade-offs
- Implementation issues.

The Jack OS


class Math {
class Memory {
Class Screen {
Class Output {
class Keyboard {
Class String {
Class Array {
Class Sys {
function void halt():
function void error(int errorCode)
function void wait(int duration)
}

The Jack OS


class Math {
function int multiply(int x, int y)
function int divide(int x, int y)
function int sqrt(int x)
function int abs(int x)
function int min(int x, int y)
function int max(int x, int y)
)
Provides commonly-used mathematical operations **(efficiently)**
Class Array {
Class Sys {
function void halt():
function void error(int errorCode)
function void wait(int duration)
}

Multiplication

In some class:

```
...
let a = 27 * 9;
// Same as:
let a = Math.multiply(27,9);
...
```

OS `Math` class

```
...
/* Returns x*y, where x, y ≥ 0. */
multiply(x, y):
```

Multiplication

In some class:

```
...
let a = 27 * 9;
// Same as:
let a = Math.multiply(27,9);
...
```

OS `Math` class

```
...
/* Returns x*y, where x, y ≥ 0. */
multiply(x, y):
    // Naïve strategy: repetitive addition
    sum = 0
    for i = 0 ... y – 1   do
        sum = sum + x
    return sum
...
```

In this lecture we present algorithms using block indentation, Python-style

Let's consider another algorithm

Multiplication

```
          1  1  0  1  1   = 27
          1  0  0  1      = 9
       ------------------
          1  1  0  1  1
       0  0  0  0  0
    0  0  0  0  0
 1  1  0  1  1
------------------------
 1  1  1  1  0  0  1  1   = 243
```

```
x :    ... 0  0  0  1  1  0  1  1
y :    ... 0  0  0  0  1  0  0  1   i'th bit of y
       ------------------------------
           0  0  0  1  1  0  1  1      1
           0  0  1  1  0  1  1  0      0
           0  1  1  0  1  1  0  0      0
           1  1  0  1  1  0  0  0      1
       ------------------------------
x*y :  ... 1  1  1  1  0  0  1  1
```

"Long multiplication"
(for *n*-bit numbers):

```
multiply(x, y):
  sum = 0
  xShifted = x
  for  i = 0 ... n – 1   do
      if  ((i'th bit of y) == 1)
          sum = sum + xShifted
      xShifted = xShifted * 2
  return sum
```

Multiplication

Repetitive addition:

```
multiply(x, y):
    sum = 0
    for i = 0 ... y – 1   do
        sum = sum + x
    return sum
```

Example: Multiply 5294868991 * 4294868991

Run-time

Proportional to the *input's size,* ***y***

"Long multiplication"
(for *n*-bit numbers):

```
multiply(x, y):
  sum = 0
  xShifted = x
  for  i = 0 ... n – 1   do
      if  ((i'th bit of y) == 1)
          sum = sum + xShifted
      xShifted = xShifted * 2
  return sum
```

Run-time

Proportional to the *number of digits in y* = $\boldsymbol{\log_2 y}$

Effectively, $O(1)$, since the word's length is fixed.

The algorithm uses only addition operations;

Can be implemented efficiently in either software or hardware.

Math functions

```
class Math {
   function int multiply(int x, int y)
   function int divide(int x, int y)
   function int sqrt(int x)
   function int abs(int x)
   function int min(int x, int y)
   function int max(int x, int y)
}
```

Division

In some class:

```
...
let q = 175 / 3;
// Same as:
let a = Math.divide(175,3);
...
```

OS `Math` class

```
/* Returns the integer part of x / y,
   where x ≥ 0 and y > 0. */
divide (x,y):
```

Division

In some class:

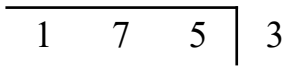
```
...
let q = 175 / 3;
// Same as:
let a = Math.divide(175,3);
...
```

OS `Math` class

```
/* Returns the integer part of x / y,
   where x ≥ 0 and y > 0. */
divide (x,y):
  // Naïve strategy: repetitive subtraction
  count = 0
  rem = x
  while rem ≤ x
      rem = rem – y
      count = count + 1
  return count
```

Let's consider another algorithm

Division



Division

		5	**8**	
	1	7	5	3
	1	5	0	
		2	5	
		2	4	
			1	

"Long division"

Huh?

Division

1	7	5	3

What is the largest number v = (90, 80, 70, 60, 50, 40, 30, 20, 10) so that $3*v \leq 175$?

Division

```
      5   0
    ------------
1   7   5 | 3
1   5   0
    -------
    2   5
```

What is the largest number v = (90, 80, 70, 60, 50, 40, 30, 20, 10) so that $3*v \leq 175$?

quotient = 50,

plus 25 / 3

Division

```
      5   0
    ------------
1   7   5 | 3
1   5   0
-------------
    2   5
```

What is the largest number v = (90, 80, 70, 60, 50, 40, 30, 20, 10) so that $3*v \leq 175$?

quotient = 50,

plus 25 / 3

What is the largest number v = (9, 8, 7, 6, 5, 4, 3, 2, 1) so that $3*v \leq 25$?

Division

```
      5   8
    ------------
1   7   5 | 3
1   5   0
-------------
    2   5
    2   4
    -------
        1
```

What is the largest number v = (90, 80, 70, 60, 50, 40, 30, 20, 10) so that $3*v \leq 175$?

quotient = 50,

plus 25 / 3

What is the largest number v = (9, 8, 7, 6, 5, 4, 3, 2, 1) so that $3*v \leq 25$?

quotient = 58,

plus 1 / 3

Division

```
      5   8
    ------------
1   7   5 | 3
1   5   0
-------------
    2   5
    2   4
    -------
        1
```

What is the largest number v = (90, 80, 70, 60, 50, 40, 30, 20, 10) so that $3*v \leq 175$?

quotient = 50,

plus 25 / 3

What is the largest number v = (9, 8, 7, 6, 5, 4, 3, 2, 1) so that $3*v \leq 25$?

quotient = 58,

plus 1 / 3

1 is less than 3, so the answer is: 58 and a remainder of 1.

Division

Repetitive subtraction:

```
divide (x, y):
  let div = 0
  let rem = x
  while rem ≤ x
      let rem = rem – y
      let div = div + 1
  return div
```

Run-time

Proportional to the *input's size, x*

"Long division":

```
      5   8
    ---------
 1  7   5  | 3
 1  5   0
 ---------
      2   5
      2   4
 ---------
          1
```

Run-time

Proportional to the *number of digits in* $x = log_2 x$

Division

Another efficient division algorithm:

```
/* Returns the integer part of x / y,
   where x ≥ 0 and y > 0  */
divide (x, y):
    // Based on: x/y = 2 · (x/2 / y)
    if  (y > x) return 0
    q = divide (x, 2 * y)
    if ((x – 2 * q * y) < y)
        return 2 * q
    else
        return 2 * q + 1
```

Run-time

Also proportional to $log_2 x$

Effectively, $O(1)$, since the word's length is fixed.

The algorithm uses only addition operations;

Can be implemented efficiently in either software or hardware.

Recap

```
class Math {
   function int multiply(int x, int y)
   function int divide(int x, int y)
   function int sqrt(int x)
   function int abs(int x)
   function int min(int x, int y)
   function int max(int x, int y)
}
```

Square root

Two important properties of the function $\sqrt{x}$

Its inverse function (x^2) can be easily computed;

It's a monotonically increasing function.

Therefore:

Square roots can be computed using binary search

/* Computes the integer part of $y = \sqrt{x}$ for n-bit numbers */

sqrt (x) :

// Finds an integer y such that $y^2 \le x < (y+1)^2$ (for $0 \le x < 2^n$)

// by performing binary search in the range $0 \ldots 2^{n/2} - 1$

$y = 0$

for $j = n/2 - 1 \ldots 0$ do

if $(y + 2^j)^2 \le x$ then $y = y + 2^j$

return y

Run-time

Proportional to $log_2 x$

Implementation notes: Multiplication

```
/* Returns x*y, where x, y ≥ 0. */
multiply(x, y):
  sum = 0
  xShifted = x
  for  i = 0 ... n – 1   do
      if  ((i'th bit of y) == 1)
          sum = sum + xShifted
      xShifted = xShifted * 2
  return sum
```

Handling signed numbers

If the inputs are two's complement values, the algorithm works as-is

Handling overflow

The algorithm always returns the correct answer modulo 2^n

Implementation notes: Multiplication

```
/* Returns x*y, where x, y ≥ 0. */
multiply(x, y):
  sum = 0
  xShifted = x
  for  i = 0 ... n – 1   do
    if  ((i'th bit of y) == 1)
        sum = sum + xShifted
    xShifted = xShifted * 2
  return sum
```

Implementing *i*'th bit of *y*

We suggest using a helper function:

```
// Returns true if the i-th bit of x is 1, false otherwise
function boolean bit(int x, int i)
```

In principle, `bit(x,i)` can be implemented using bit shifting;

Instead, we can use a fixed array that holds the powers of 2:

`twoToThe[i]` = 2^i, $i = 0,...,15$

- Declare `twoToThe` as a static class variable of the OS class `Math`; Construct the array in the function `Math.init`
- Use `twoToThe` to support the implementation of `bit(x,i)`

Implementation notes: Division

```
/* Returns the integer part of x / y,
   where x ≥ 0 and y > 0. */
divide (x, y):
    if  (y > x) return 0
    q = divide (x, 2 * y)
    if ((x – 2 * q * y) < y)
        return 2 * q
    else
        return 2 * q + 1
```

Handling signed numbers

Divide $|x|$ by $|y|$, then set the quotient's sign

Handling overflow of y

The overflow can be detected when y becomes negative;

Change the function's first statement to:

if $(y > x)$ or $(y < 0)$ return 0

Implementation notes: Square root

/* Computes the integer part of $y = \sqrt{x}$ for n-bit numbers */

sqrt (x):

// Finds an integer y such that $y^2 \le x < (y+1)^2$ (for $0 \le x < 2^n$)

// by performing binary search in the range $0 \ldots 2^{n/2} - 1$

$y = 0$

for $j = n/2 - 1 \ldots 0$ do

if $(y + 2^j)^2 \le x$ then $y = y + 2^j$

return y

Handling overflow of $(y + 2^j)^2$

Change the condition $(y + 2^j)^2 \le x$ to:

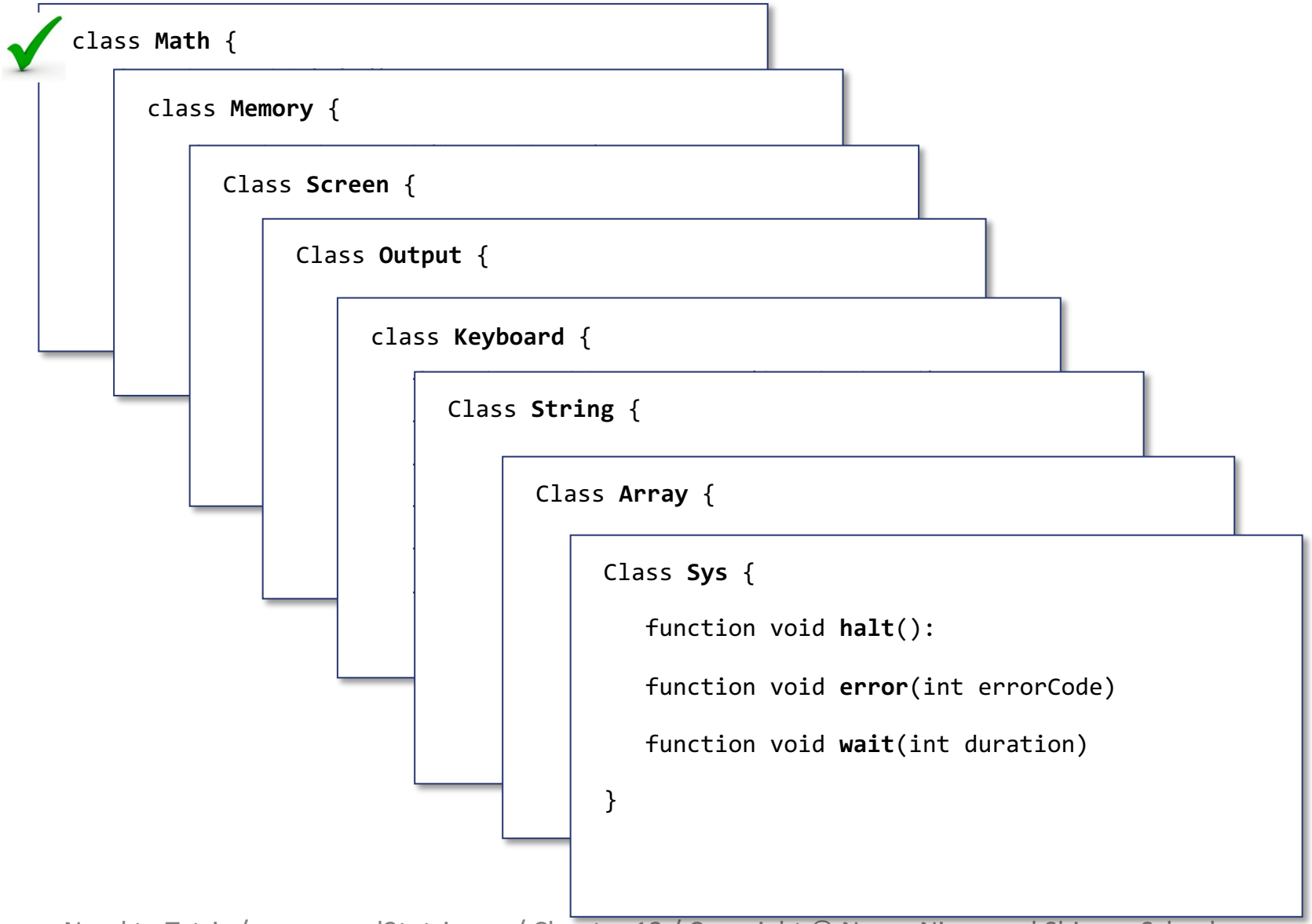
$$(y + 2^j)^2 \le x \text{ and } (y + 2^j)^2 > 0$$

Math functions

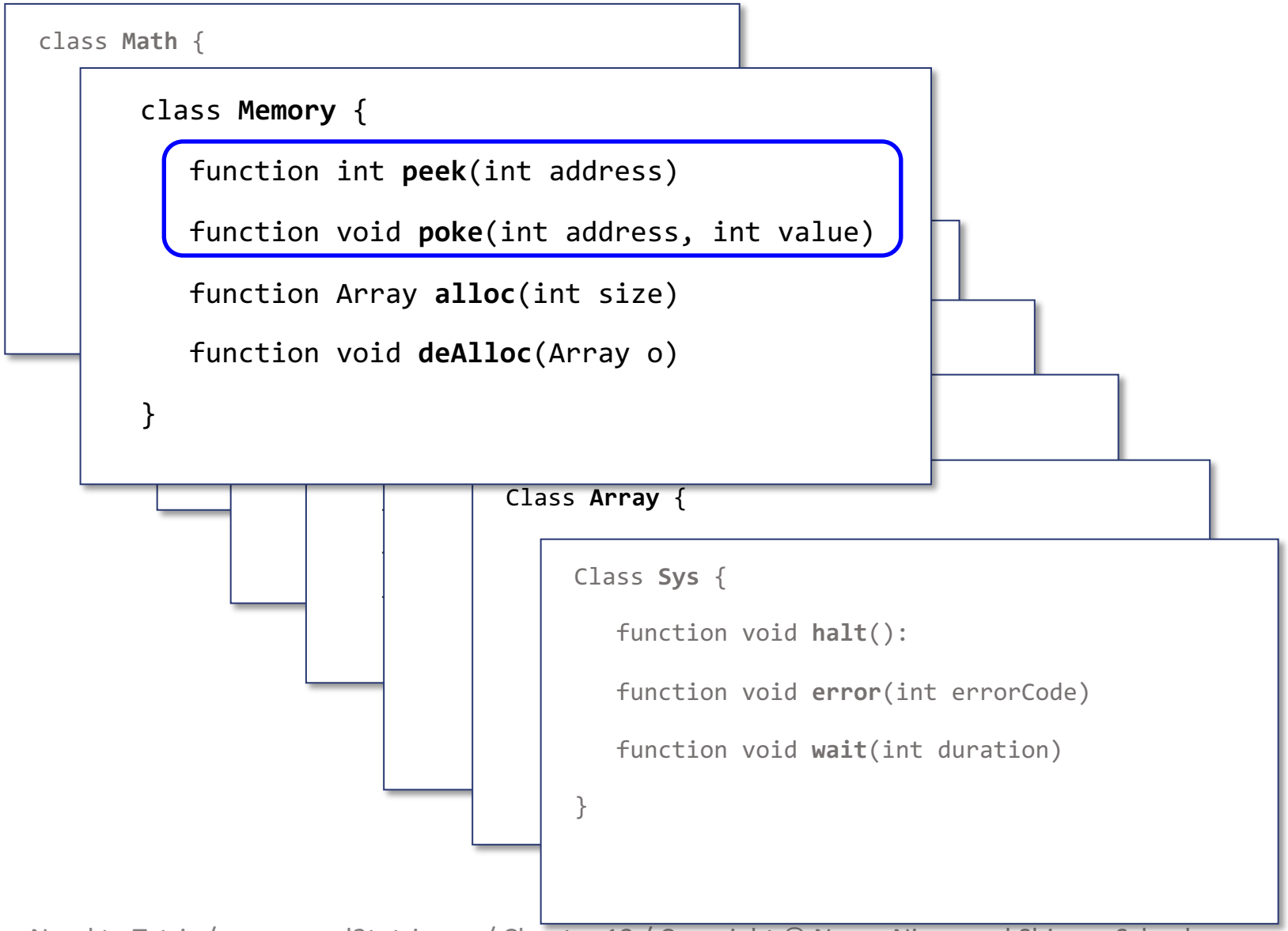
```
class Math {
✓ function int multiply(int x, int y)
✓ function int divide(int x, int y)
✓ function int sqrt(int x)
  function int abs(int x)
  function int min(int x, int y)
  function int max(int x, int y)
}
```

Implementing of `abs`, `min`, `max`: Simple.

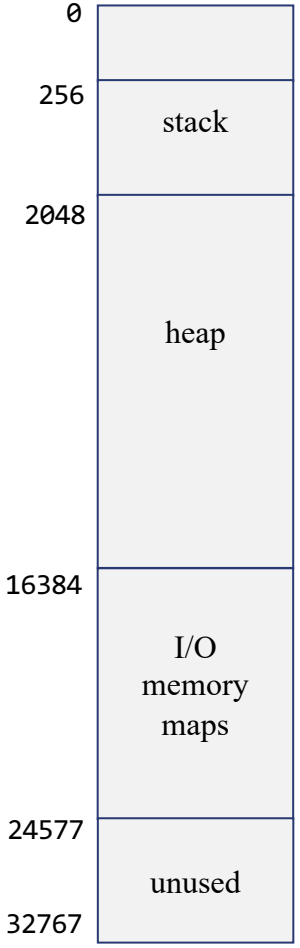
OS services


class Math {
class Memory {
Class Screen {
Class Output {
class Keyboard {
Class String {
Class Array {
Class Sys {
function void halt():
function void error(int errorCode)
function void wait(int duration)
}

OS services


class Math {
class Memory {
function int peek(int address)
function void poke(int address, int value)
function Array alloc(int size)
function void deAlloc(Array o)
}
Class Array {
Class Sys {
function void halt():
function void error(int errorCode)
function void wait(int duration)
}

RAM access: peek / poke



OS class `Memory`

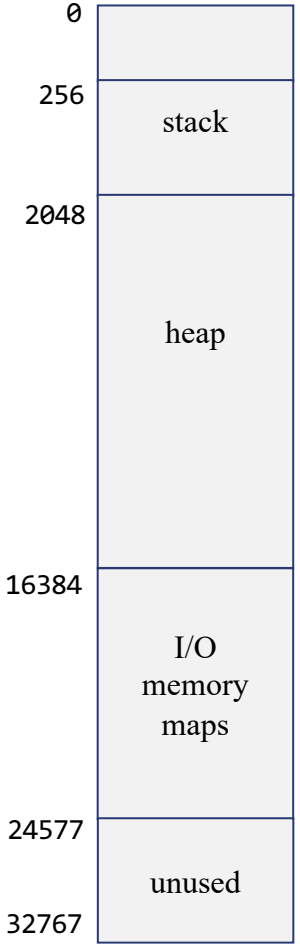
```
/* Memory operations */
class Memory {

    /* Returns the value of the given RAM address */
    function int peek(int address) {}

    /* Sets the value of the given RAM address to the given value */
    function void poke(int address, int value) {}
    ...
}
```

`peek` / `poke`: Designed to encapsulate direct access to *any* word in the RAM

Usage

```
...
let x = Memory.peek(5000)   // x = RAM[5000]
...
do Memory.poke(5000,17)     // RAM[5000] = 17
...
```

RAM access: peek / poke



OS class `Memory`

```
/* Memory operations */
class Memory {
    ...
    static Array ram;
    ...
    // Initializes the Memory class
    function void init() {
        let ram = 0;
        ...
    }
    /* Returns the value of the given RAM address */
    function int peek(int address) {}
    /* Sets the value of the given RAM address to the given value */
    function void poke(int address, int value) {}
    ...
}
```

Usage

```
...
let x = Memory.peek(5000)   // x = RAM[5000]
...
do Memory.poke(5000,17)     // RAM[5000] = 17
...
```

Explanation

Jack is weakly typed; The hack "`let ram = 0`" does not cause the compiler to complain;

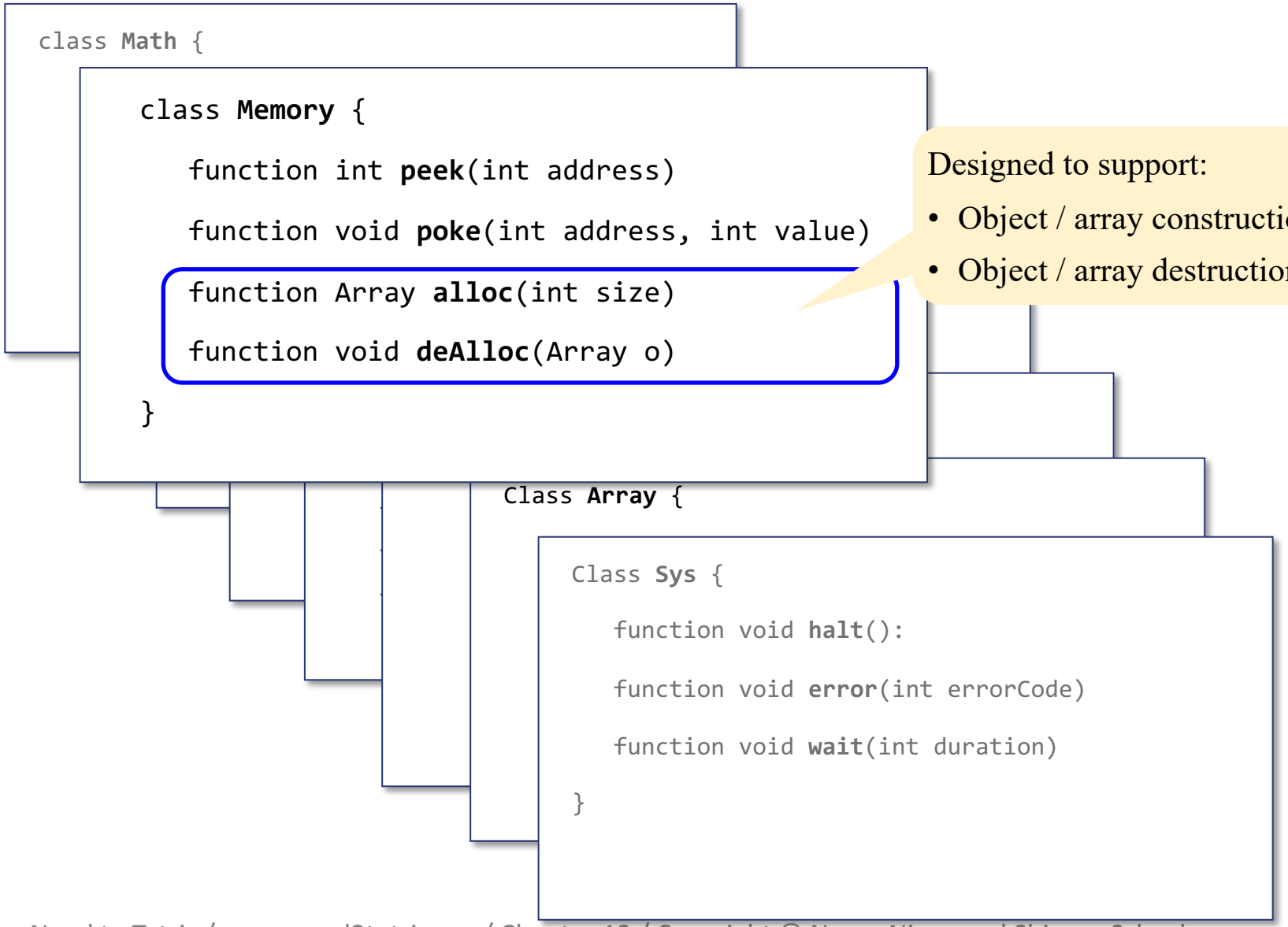
After `init` executes (`Memory.init` will be called by `Sys.init`):

Other `Memory` functions gain full access to the RAM, since accessing array element `ram[`*addr*`]` results in compiled code that accesses `ram[0+`*addr*`]`;

Therefore, `peek` and `poke` can be implemented trivially by accessing the static array `ram`.

`peek` / `poke`: Designed to encapsulate direct access to *any* word in the RAM

OS services



Designed to support:

- Object / array construction
- Object / array destruction

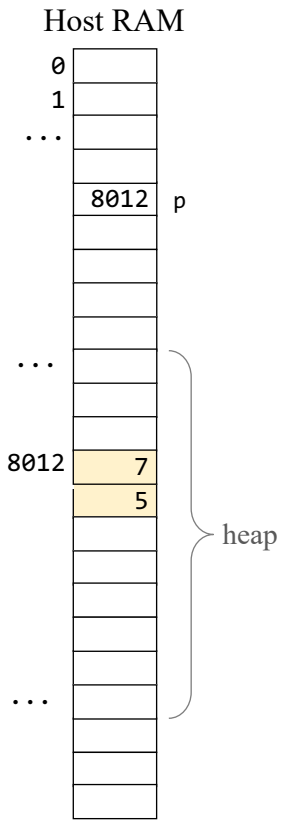
Object / array construction

```
// In some high-level language class:
var Point p;
...
let p = Point.new(7,5);
```

```
class Point {
   field int x,y;
   ...
   constructor Point new(int ax, int ay) {
      let x = ax;
      ley y = ay;
      return this;
   }
   ...
```

When the compiler translates the constructor, it generates VM code affecting:

```
this = Memory.alloc(2)
```



OS `Memory` class API

```
class Memory {
   ...
   /* Finds and allocates from the heap a memory block of the
      specified size and returns its base address */
   function int alloc(int size) {}
   /* De-allocates the given object and frees its space */
   function void deAlloc(int object) {}
}
```

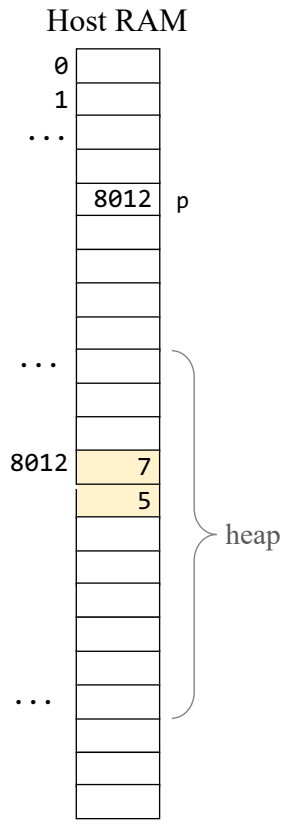
Object / array destruction

```
// In some class:
...
do p.dispose();
```

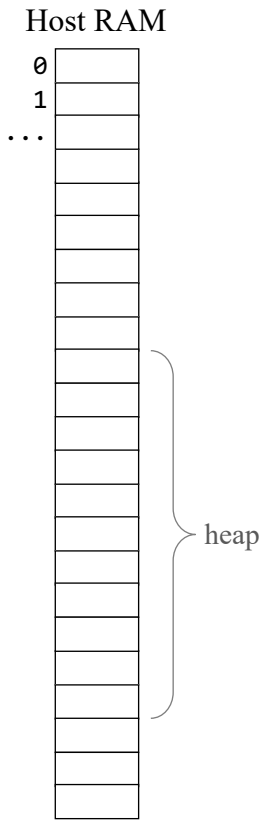
```
class Point {
   ...
   /* Disposes this point */
   method void dispose() {
      do Memory.deAlloc(this);
      return;
   }
...
```

OS `Memory` class API

```
class Memory {
   ...
   /* Finds and allocates from the heap a memory block of the
      specified size and returns its base address */
   function int alloc(int size) {}
   /* De-allocates the given object and frees its space */
   function void deAlloc(int object) {}
}
```



Implementing `alloc` and `deAlloc`



OS `Memory` class API

```
class Memory {
    ...
    /* Finds and allocates from the heap a memory block of the
       specified size and returns its base address */
    function int alloc(int size) {}
    /* De-allocates the given object and frees its space */
    function void deAlloc(int object) {}
}
```

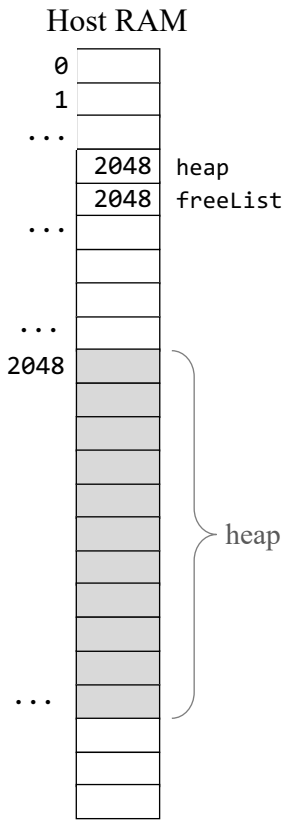
Implementation:

heap management algorithms

Heap management (simple)

Heap management

```
init:
    heap = 2048
    freeList = heap

/* allocates a memory block of size words */
alloc (size):
    block = freeList
    freeList = freeList + size
    return block

/* de-allocates the memory space of the given object */
deAlloc (object):
      do nothing
```

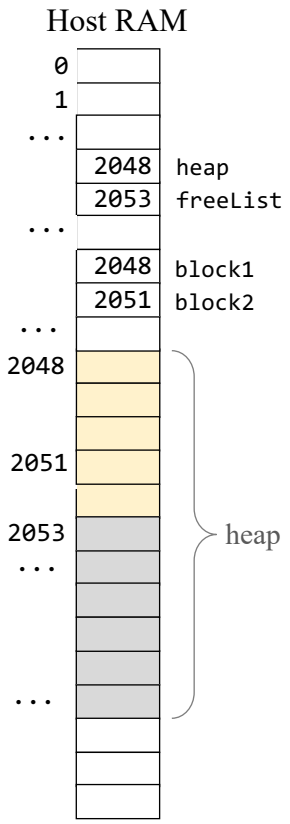


Initial state

Heap management (simple)

Heap management

```
init:
    heap = 2048
    freeList = heap
/* allocates a memory block of size words */
alloc (size):
    block = freeList
    freeList = freeList + size
    return block
/* de-allocates the memory space of the given object */
deAlloc (object):
      do nothing
```



After two objects were allocated

Usage

```
...
block1 = Memory.alloc(3);
...
block2 = Memory.alloc(2);
...
```

Heap management (the real thing)

Data structure:

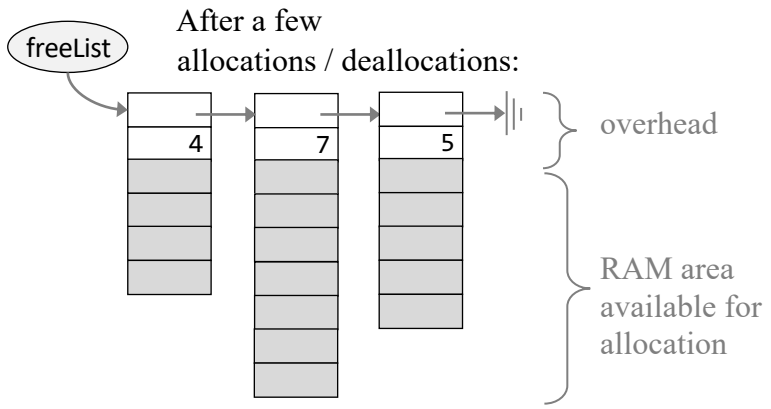
Linked list of available memory segments

alloc(*size*)

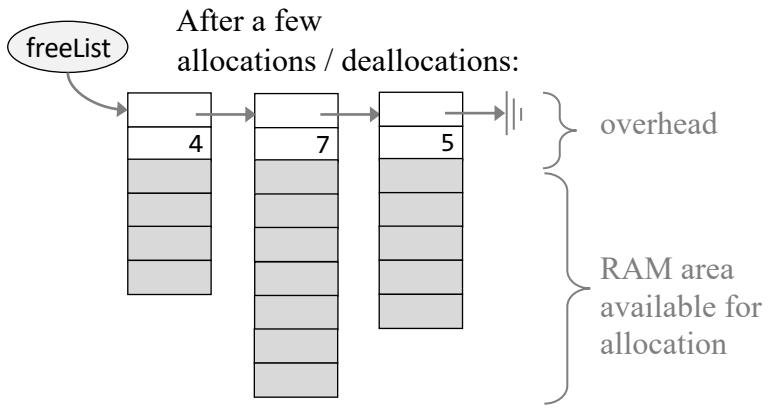
Finds a block of size *size* in one of the segments, removes the block from the segment, and returns the block to the caller

deAlloc(*object*)

Appends the object/block to the *freeList*



Heap management



`alloc(3)`:

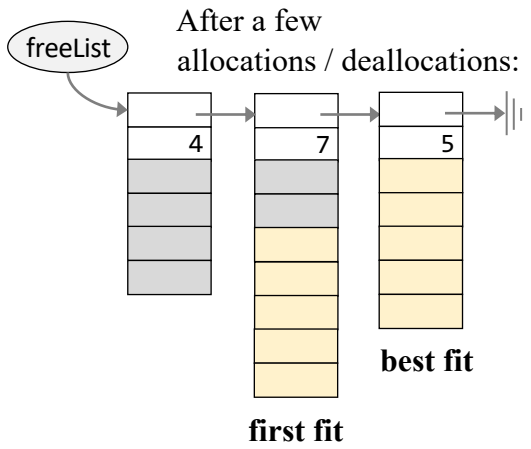
Heap management

alloc(*size*)

Terminology: if *segment.size* ≥ *size* + 2
we say that the segment is *possible*

Search the *freeList* for:

- the first possible segment (*first fit*) , or
- the smallest possible segment (*best fit*)



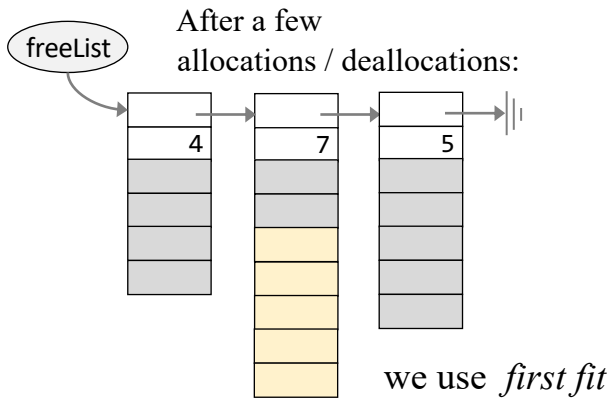
Heap management

alloc(*size*)

Terminology: if *segment.size* ≥ *size* + 2
we say that the segment is *possible*

Search the *freeList* for:

- the first possible segment (*first fit*) , or
- the smallest possible segment (*best fit*)



`alloc(3)`:

Heap management

alloc(*size*)

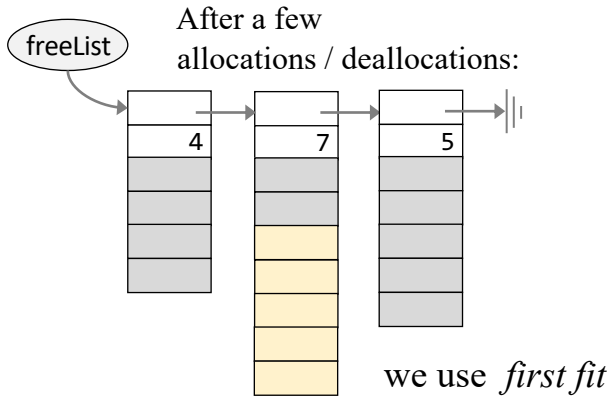
Terminology: if *segment.size* ≥ *size* + 2
we say that the segment is *possible*

Search the *freeList* for:

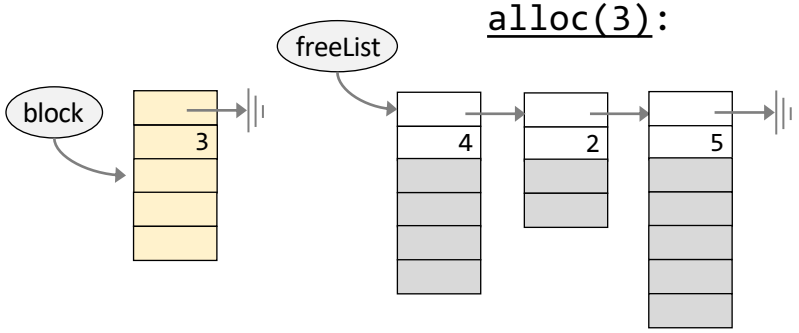
- the first possible segment (*first fit*) , or
- the smallest possible segment (*best fit*)

Carve a block of size *size* + 2 from this segment;

Return the base address of the block's data part.



`alloc(3)`:



`deAlloc(block)`:

Heap management

alloc(*size*)

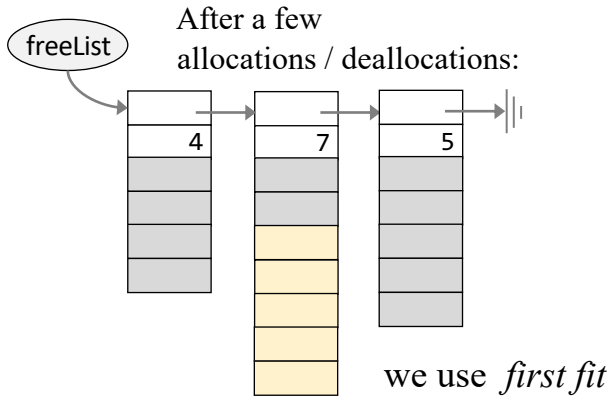
Terminology: if *segment.size* ≥ *size* + 2
we say that the segment is *possible*

Search the *freeList* for:

- the first possible segment (*first fit*) , or
- the smallest possible segment (*best fit*)

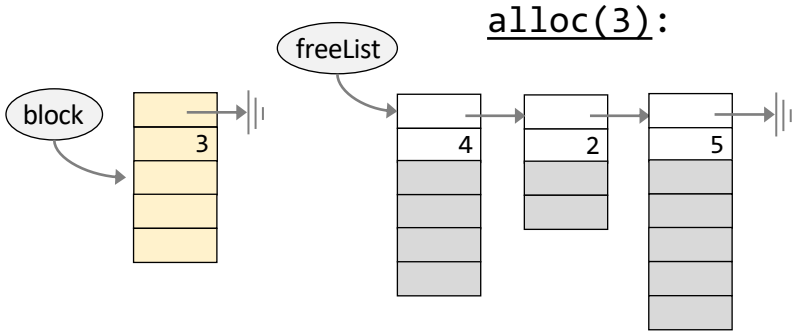
Carve a block of size *size* + 2 from this segment;

Return the base address of the block's data part.



deAlloc(*object*)

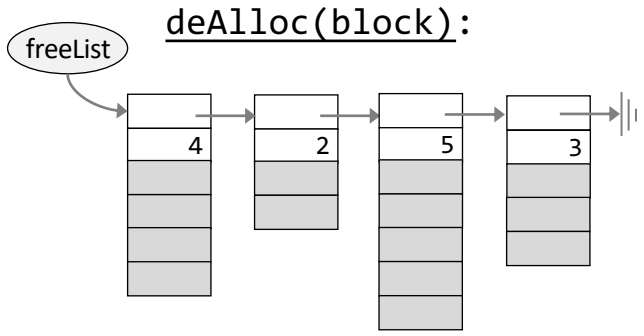
Append *object* to the end of the *freeList*



defrag

The more we recycle (`deAlloc`), the more the `freeeList` becomes fragmented;

Solution: Periodical defragmentation of the `freeList` (nice extension to the Jack OS).

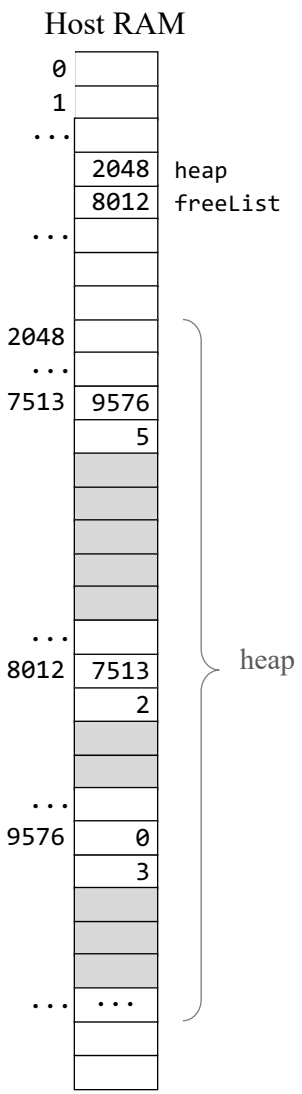


Proposed implementation

init:
freeList = *heap*
freeList.*size* = *heapSize*
freeList.*next* = 0

/* Allocates a memory block of *size* words */
alloc(*size*):
search *freeList* using best-fit or first-fit heuristics
to obtain a segment with *segment.size* ≥ *size* + 2
if no such segment is found, return failure
(or attempt defragmentation)
block = base address of the found space
update the *freeList* and the fields of *block*
to account for the allocation
return *block*

/* de-allocates the memory space of the given object */
deAlloc(*object*):
append *object* to the end of the *freeList*



Implementation tips

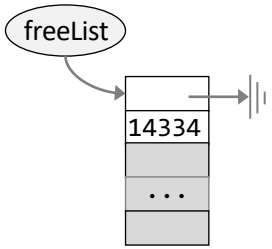
Manage the linked `freeList` directly in the array `heap`

Realize the `next` and `size` properties of the memory segment beginning at address *addr* as `heap[`*addr* – 1`]` and `heap[`*addr* – 2`]`

Implement `alloc`, `deAlloc`, and `deFrag` as operations on the `heap` array.

Proposed implementation

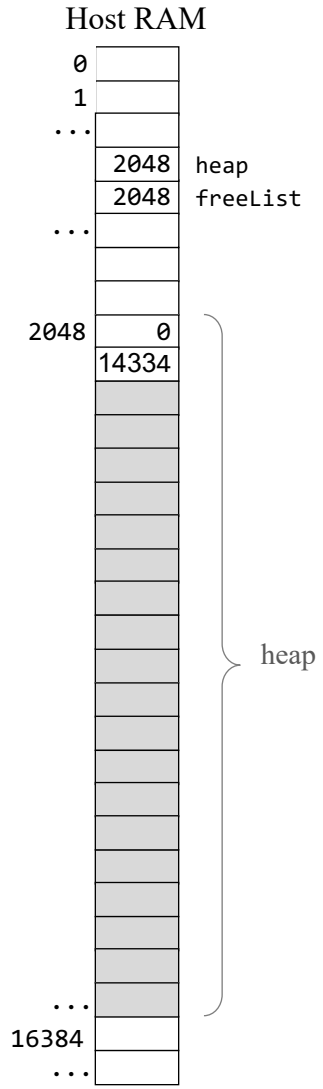
```
class Memory {
  ...
  static Array heap;
  ...
  // Initializes the Memory class
  function void init() {
    ...
    let heap = 2048;
    let freeList = heap;
    let heap[0] = 0;      // next
    let heap[1] = 14334;  // length
    ...
  }
  function int alloc(int size) {}
  function void dealloc(int object) {}
  ...
}
```



Initial state

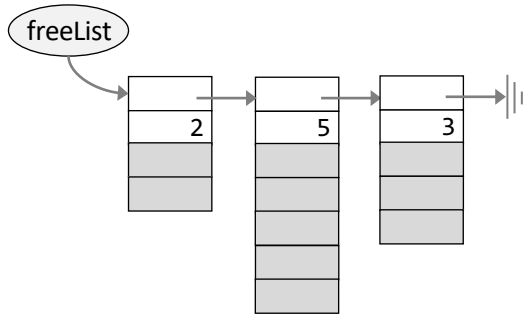
The entire heap is available:

freeList points at one long memory segment

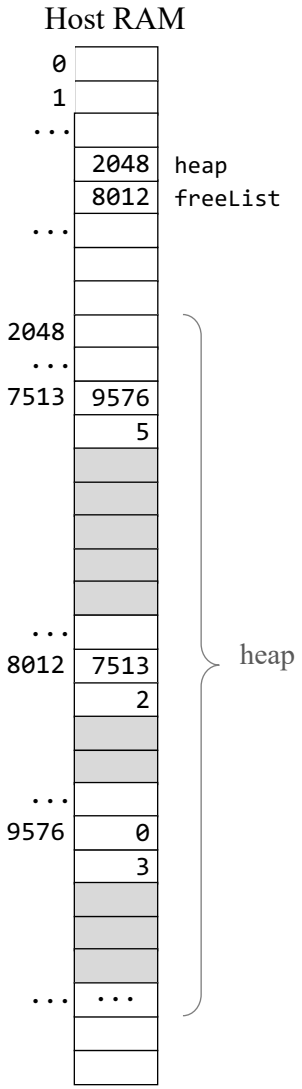


Proposed implementation

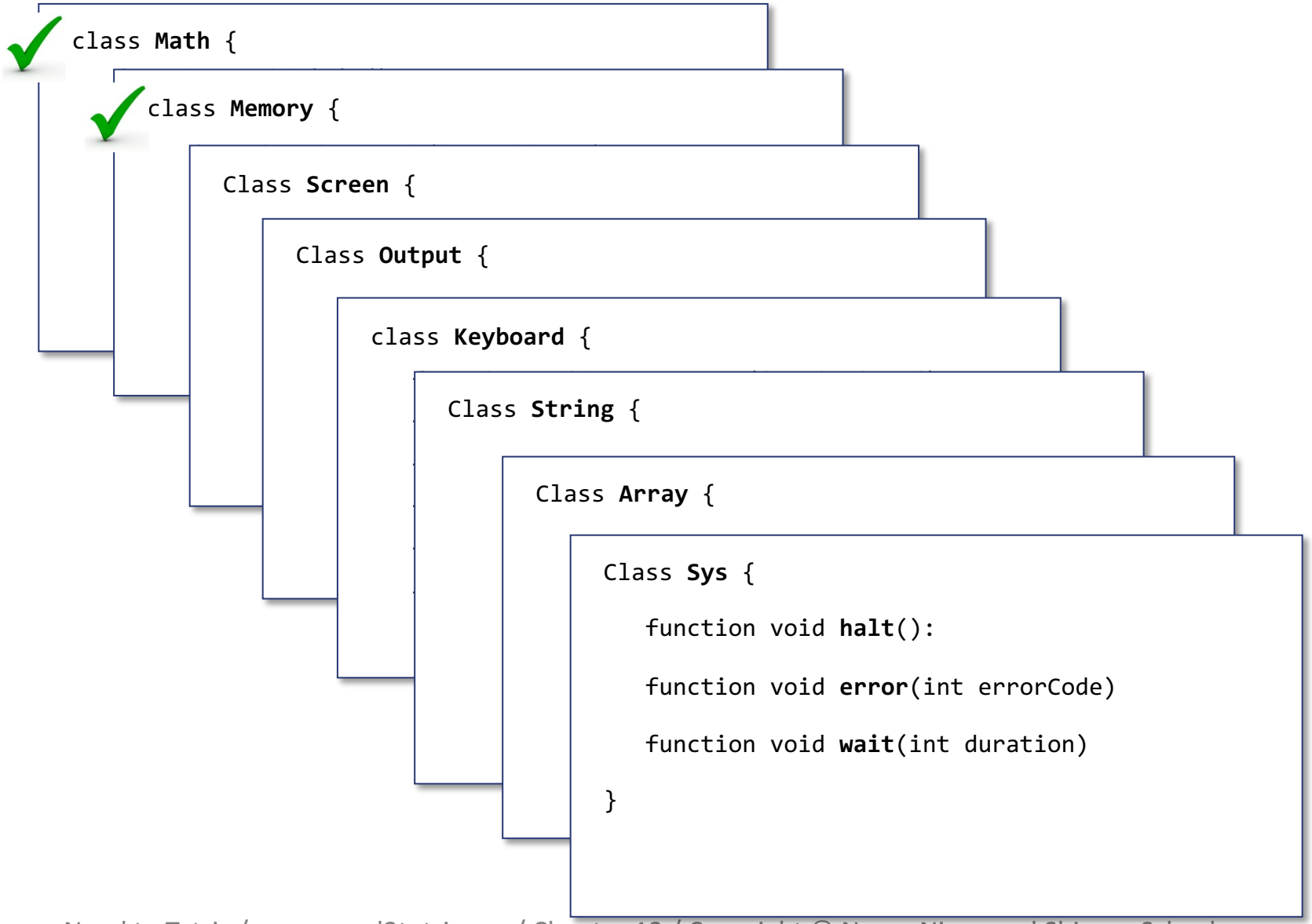
```
class Memory {
  ...
  static Array heap;
  ...
  // Initializes the Memory class
  function void init() {
    ...
    let heap = 2048;
    let freeList = heap;
    let heap[0] = 0;      // next
    let heap[1] = 14334;  // length
    ...
  }
  function int alloc(int size) {}
  function void dealloc(int object) {}
  ...
}
```



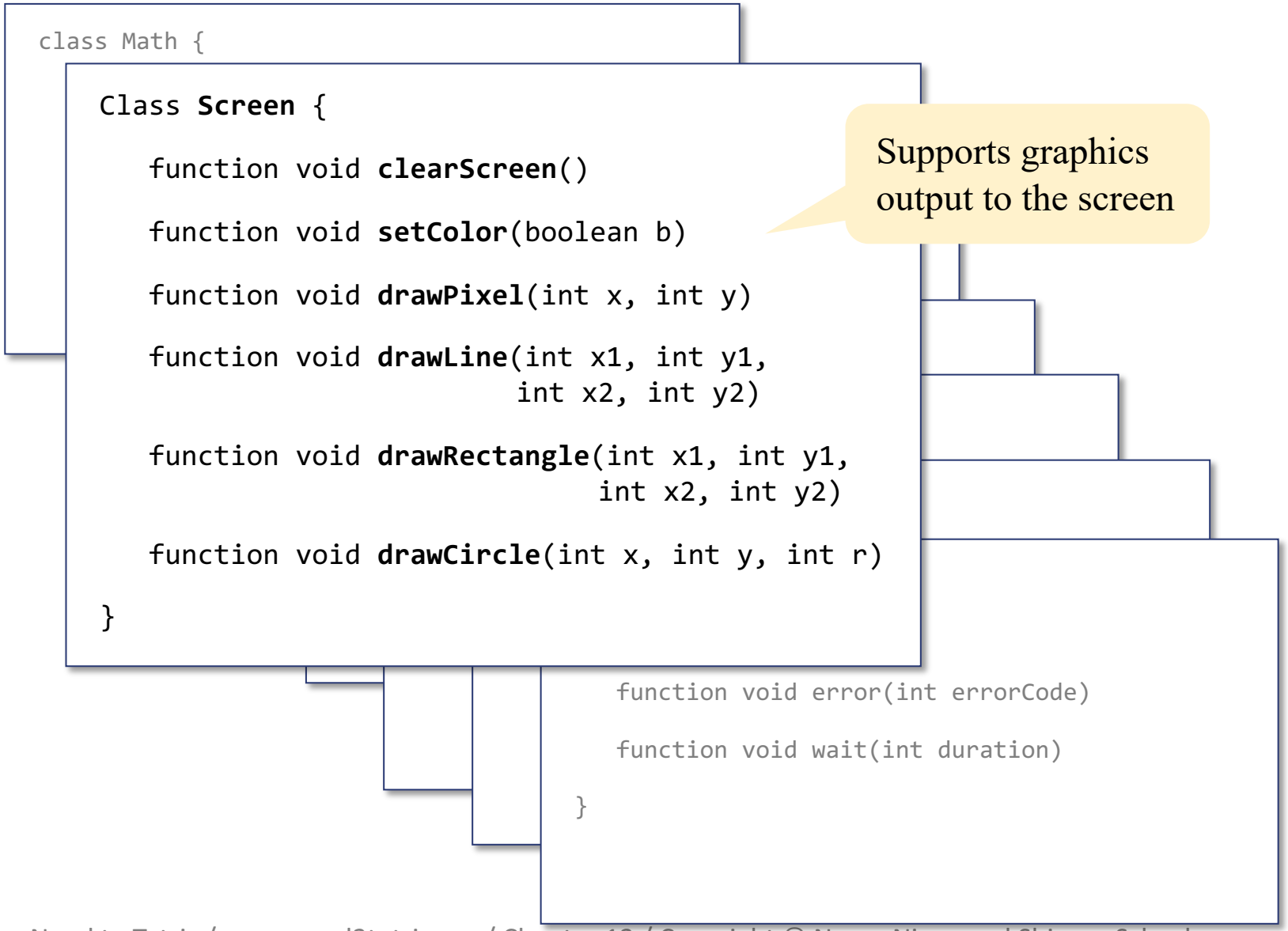
The `freeList` after a few block allocations and deallocations



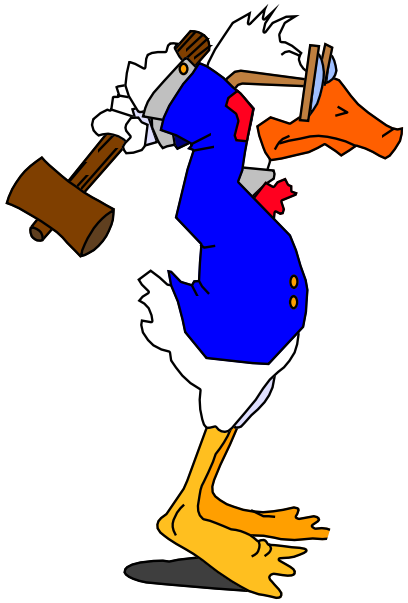
The Jack OS


class Math {
class Memory {
Class Screen {
Class Output {
class Keyboard {
Class String {
Class Array {
Class Sys {
function void halt():
function void error(int errorCode)
function void wait(int duration)
}

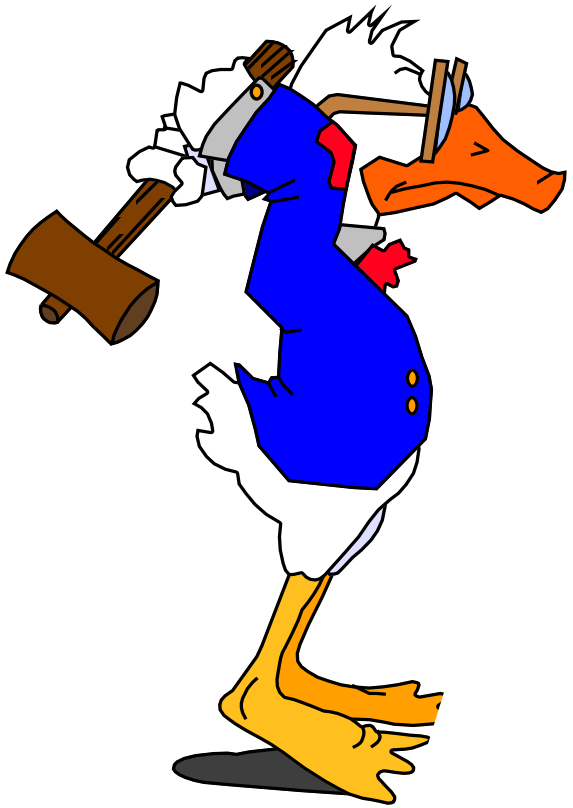
The Jack OS


class Math {
Class Screen {
function void clearScreen()
function void setColor(boolean b)
function void drawPixel(int x, int y)
function void drawLine(int x1, int y1,
int x2, int y2)
function void drawRectangle(int x1, int y1,
int x2, int y2)
function void drawCircle(int x, int y, int r)
}
Supports graphics output to the screen
function void error(int errorCode)
function void wait(int duration)
}

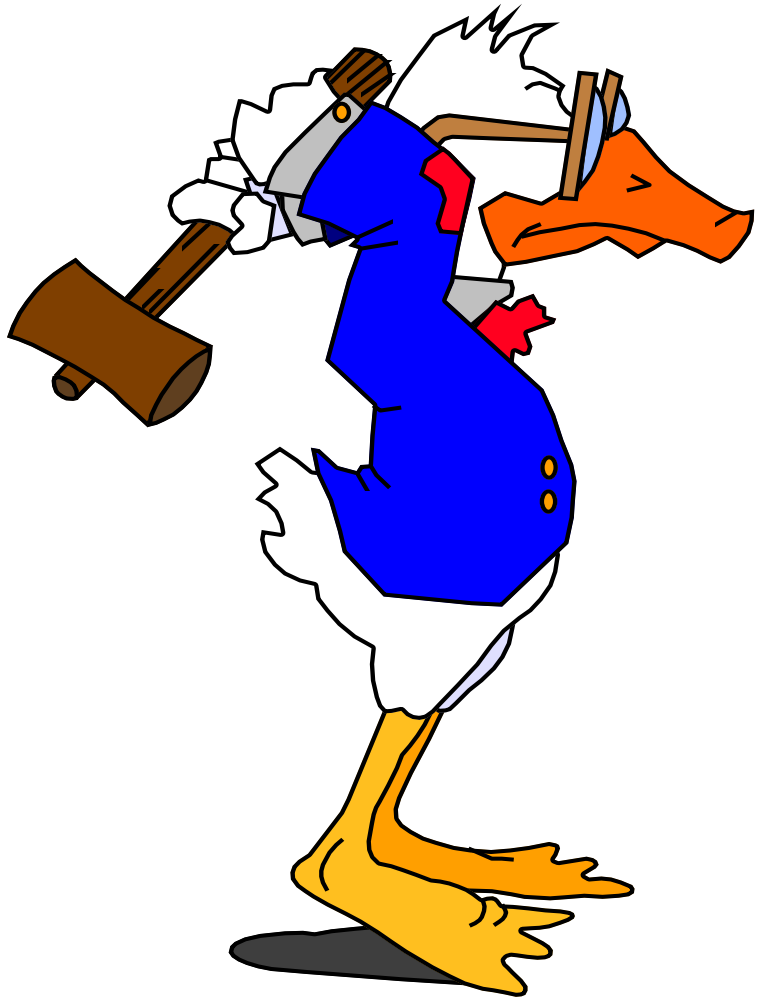
Graphics



Graphics



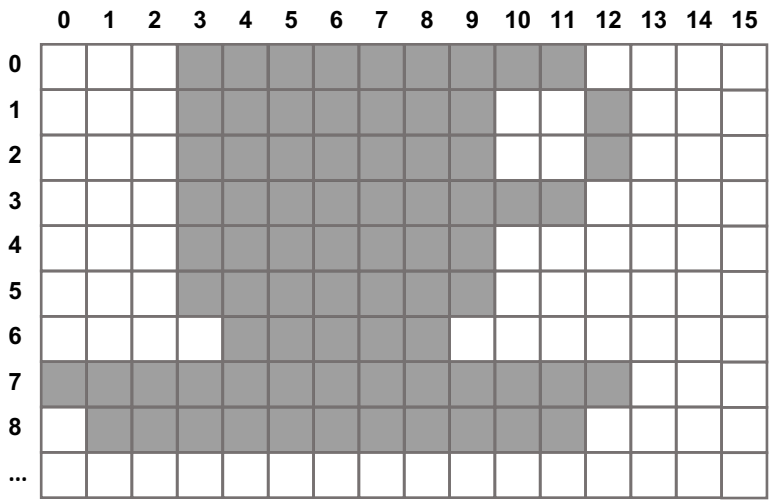
Graphics



vector graphics

bitmap graphics

Graphics



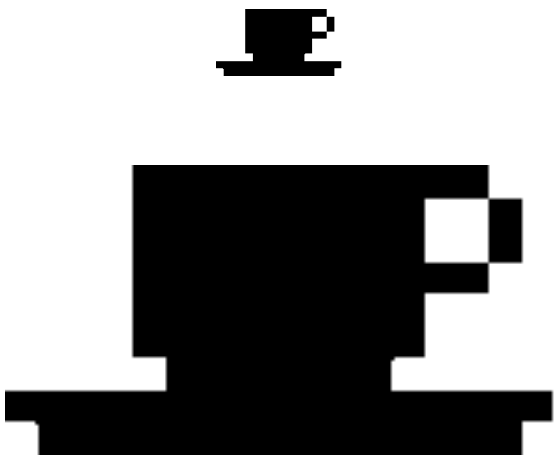
vector graphics file

```
drawLine(3,0,11,0);
drawRectangle(3,1,9,5);
drawLine(12,1,12,2);
drawLine(10,3,11,3);
drawLine(4,6,8,6);
drawLine(0,7,12,7);
drawLine(1,8,11,8);
```

bitmap graphics file

```
0001111111100000
0001111111010000
0001111111010000
0001111111100000
0001111111000000
0001111111000000
...
```

Graphics



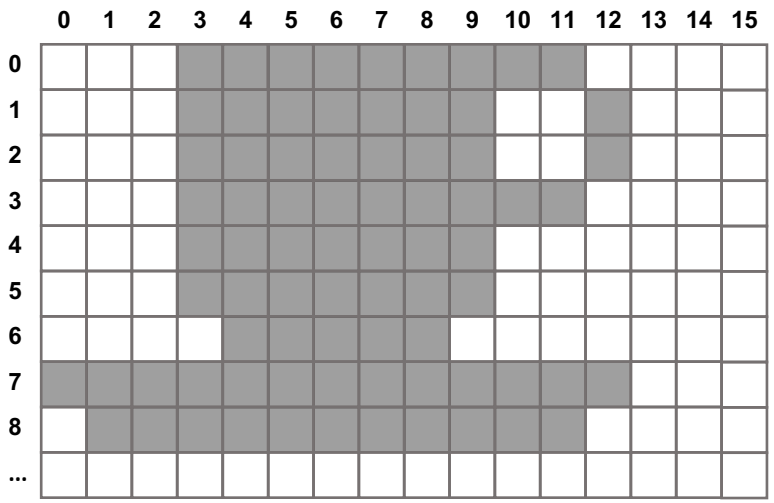
vector graphics file

```
drawLine(3,0,11,0);
drawRectangle(3,1,9,5);
drawLine(12,1,12,2);
drawLine(10,3,11,3);
drawLine(4,6,8,6);
drawLine(0,7,12,7);
drawLine(1,8,11,8);
```

bitmap graphics file

```
0001111111100000
0001111111010000
0001111111010000
0001111111100000
0001111111000000
0001111111000000
...
```

Graphics



vector graphics file

```
drawLine(3,0,11,0);
drawRectangle(3,1,9,5);
drawLine(12,1,12,2);
drawLine(10,3,11,3);
drawLine(4,6,8,6);
drawLine(0,7,12,7);
drawLine(1,8,11,8);
```

Vector graphics images can be easily:

- Stored
- Transmitted
- Scaled
- Turned into bitmap

Graphics

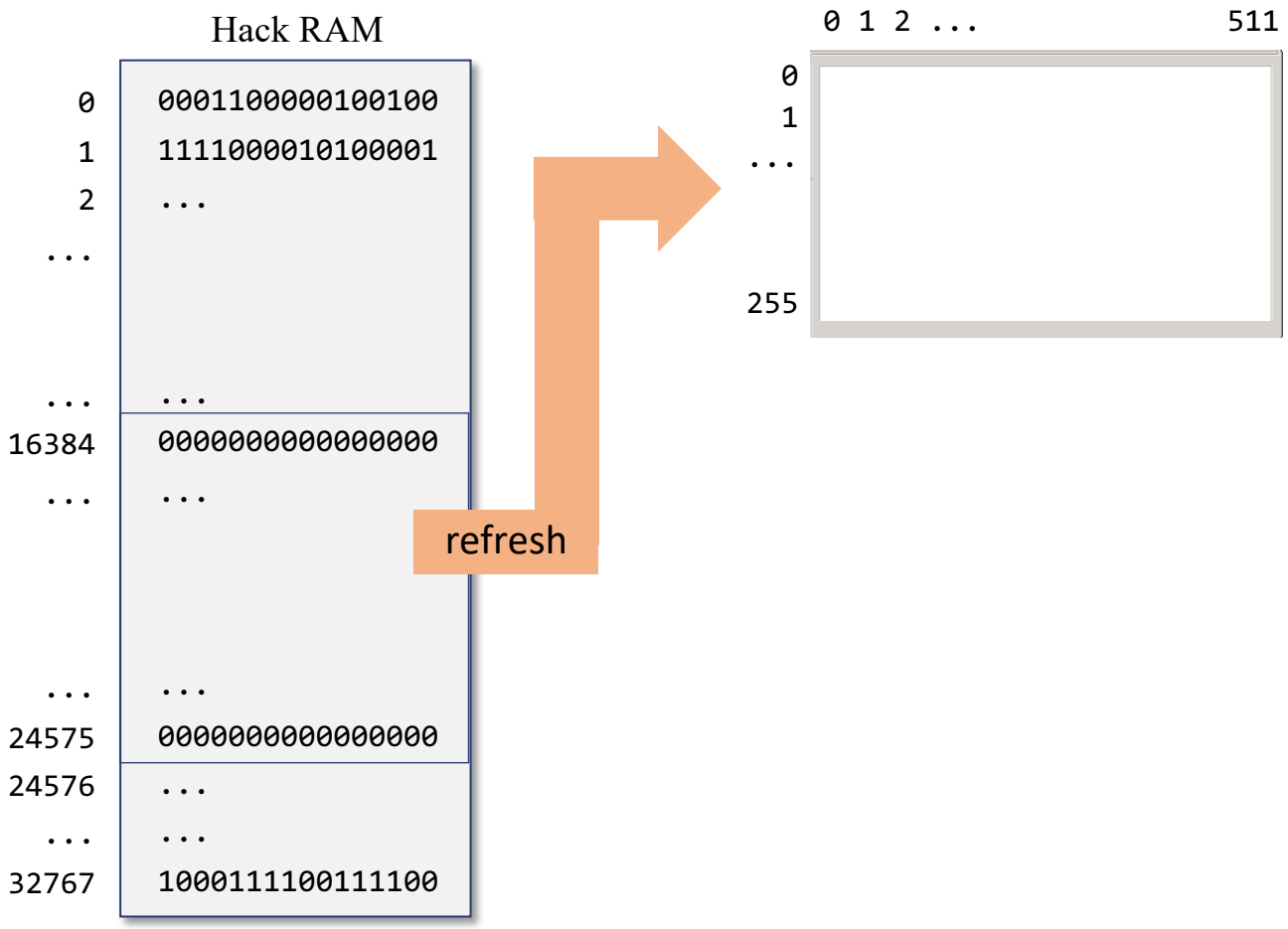
Vector graphics primitives

`drawPixel` (x, y)

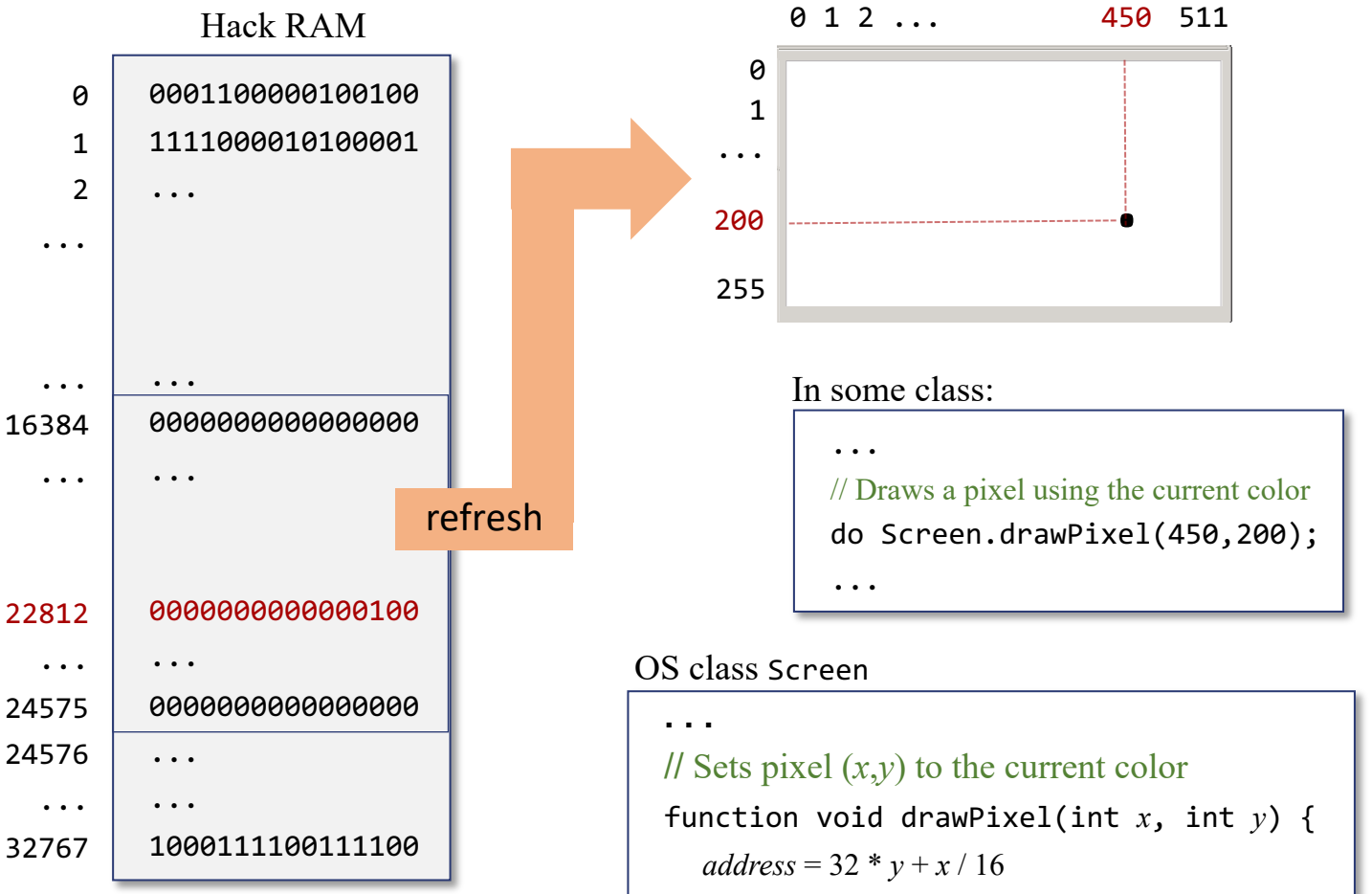
`drawLine` (x_1, y_1, x_2, y_2)

`drawCircle` (x, y, r)

Pixel drawing



Pixel drawing



In some class:

```
...
// Draws a pixel using the current color
do Screen.drawPixel(450,200);
...
```

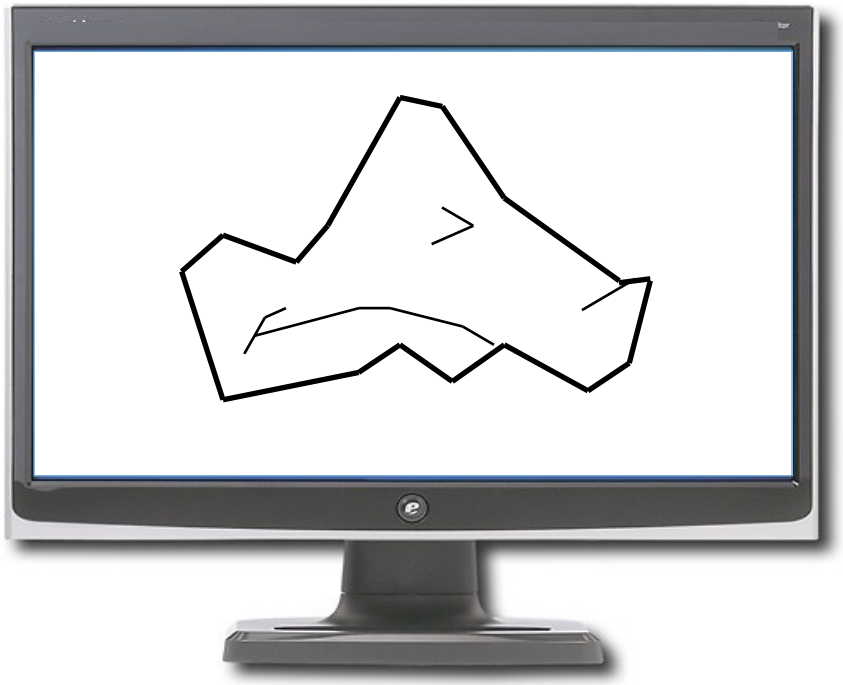
OS class `Screen`

```
...
// Sets pixel (x,y) to the current color
function void drawPixel(int x, int y) {
   address = 32 * y + x / 16
   value = Memory.peek[16384 + address]
   set the (x % 16)th bit of value to the current color
   do Memory.poke(address,value)
}
```

Recap

```
Class Screen {
   function void clearScreen()
   function void setColor(boolean b)
   function void drawPixel(int x, int y)
   function void drawLine(int x1, int y1, int x2, int y2)
   function void drawRectangle(int x1, int y1, int x2, int y2)
   function void drawCircle(int x, int y, int r)
}
```

Line drawing



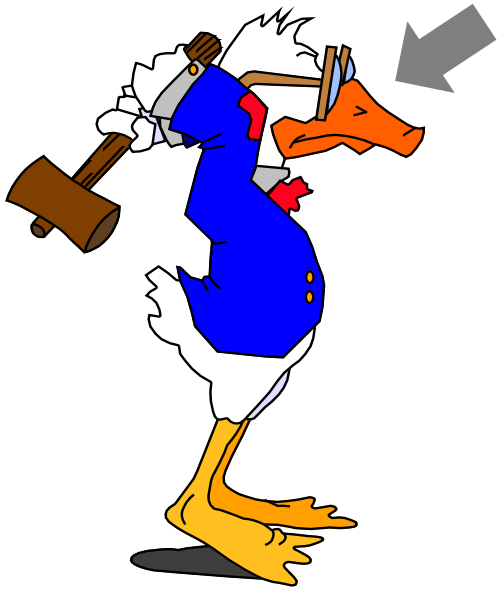
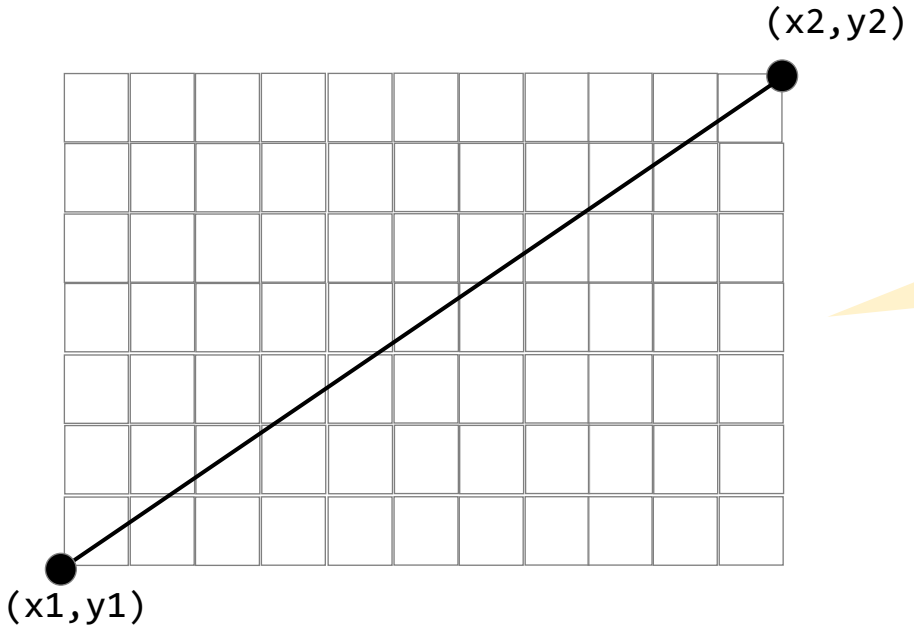


Image drawing can be done by a sequence of `drawLine` operations.

Challenge: Draw the lines *fast*.

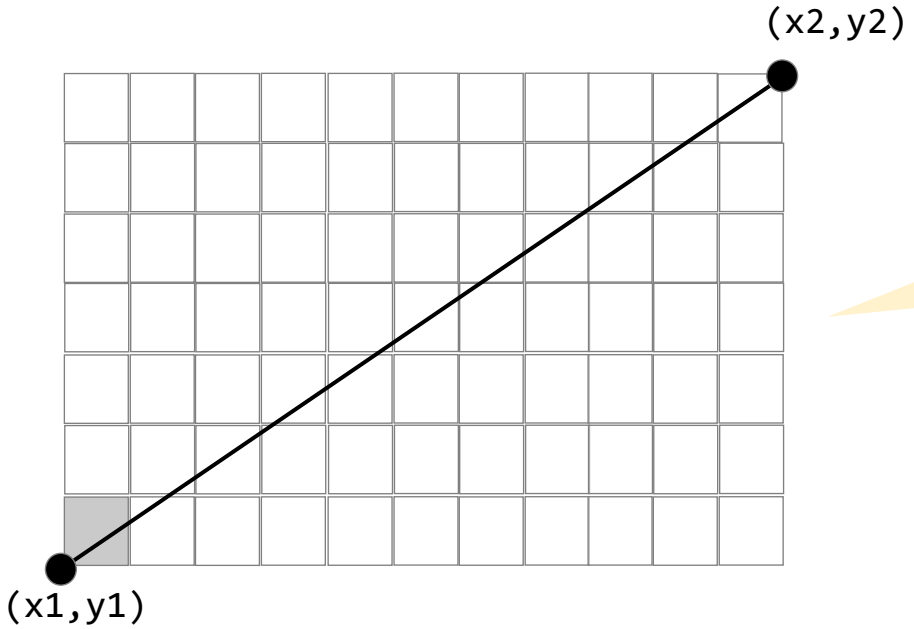
Line drawing



Simplifying assumption:

Focus on lines that go *north-east*

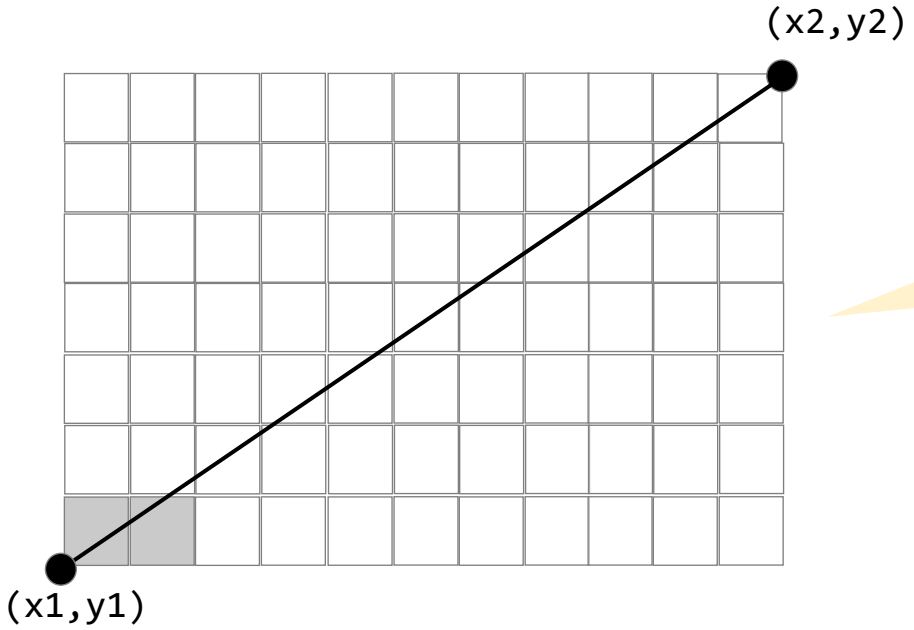
Line drawing



Simplifying assumption:

Focus on lines that go *north-east*

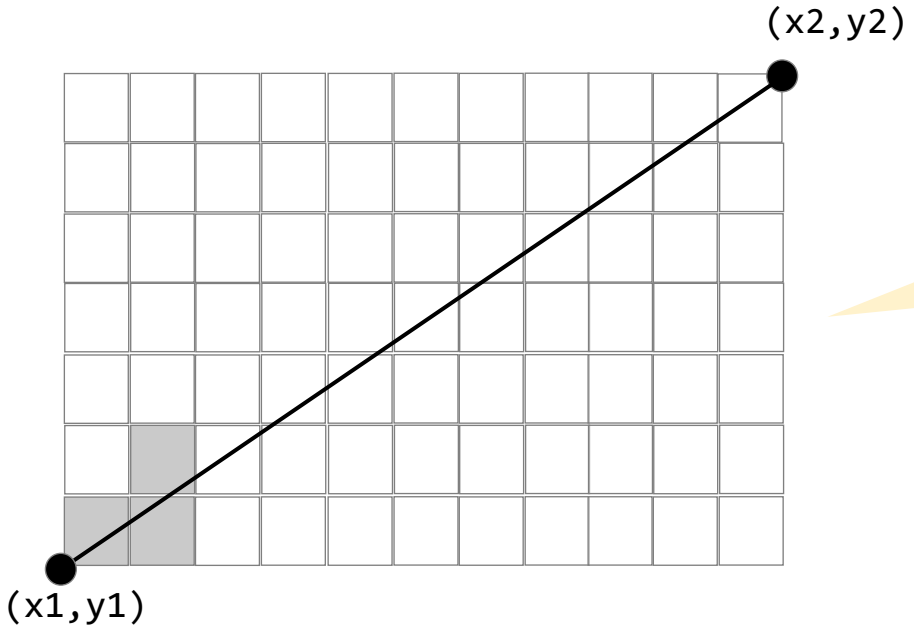
Line drawing



Simplifying assumption:

Focus on lines that go *north-east*

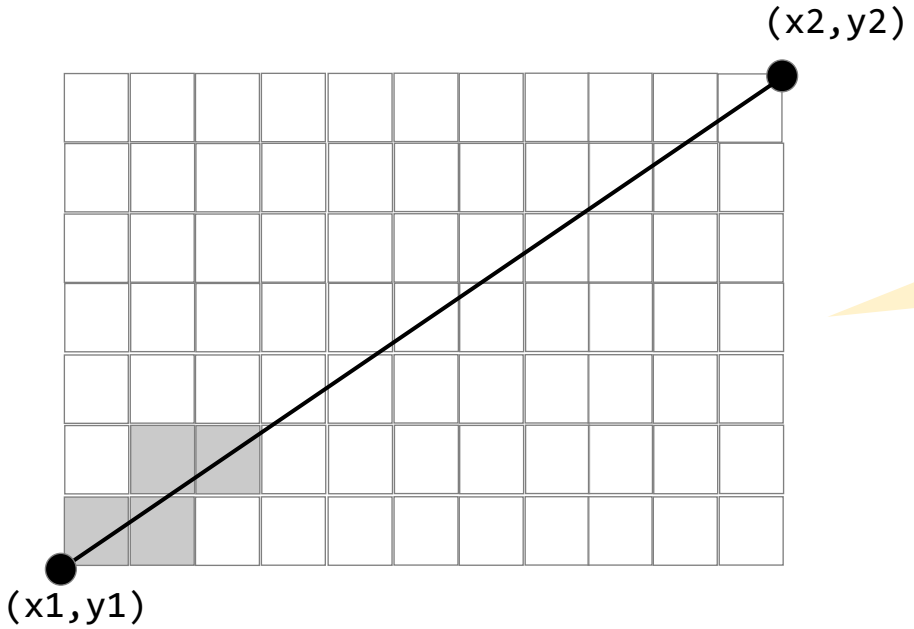
Line drawing



Simplifying assumption:

Focus on lines that go *north-east*

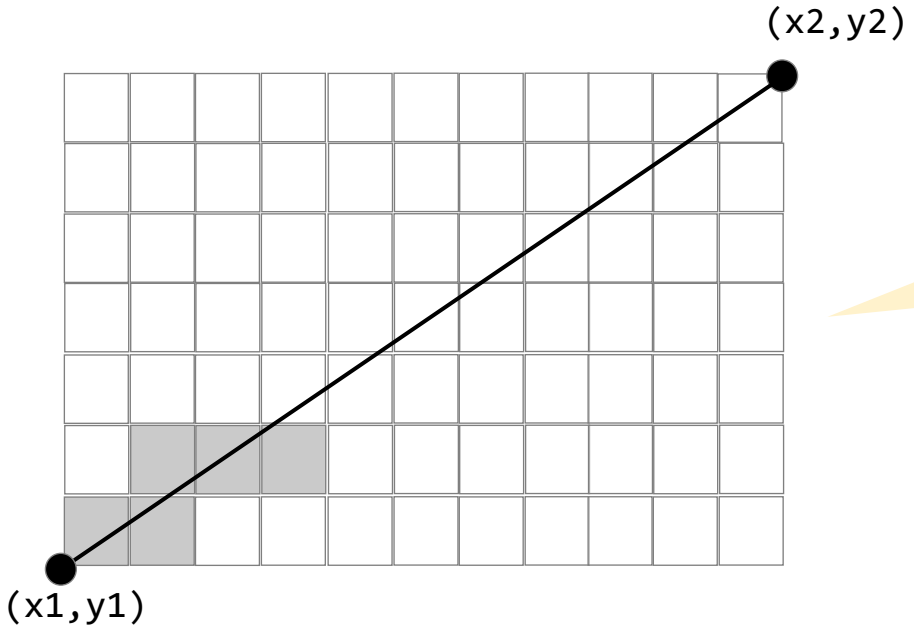
Line drawing



Simplifying assumption:

Focus on lines that go *north-east*

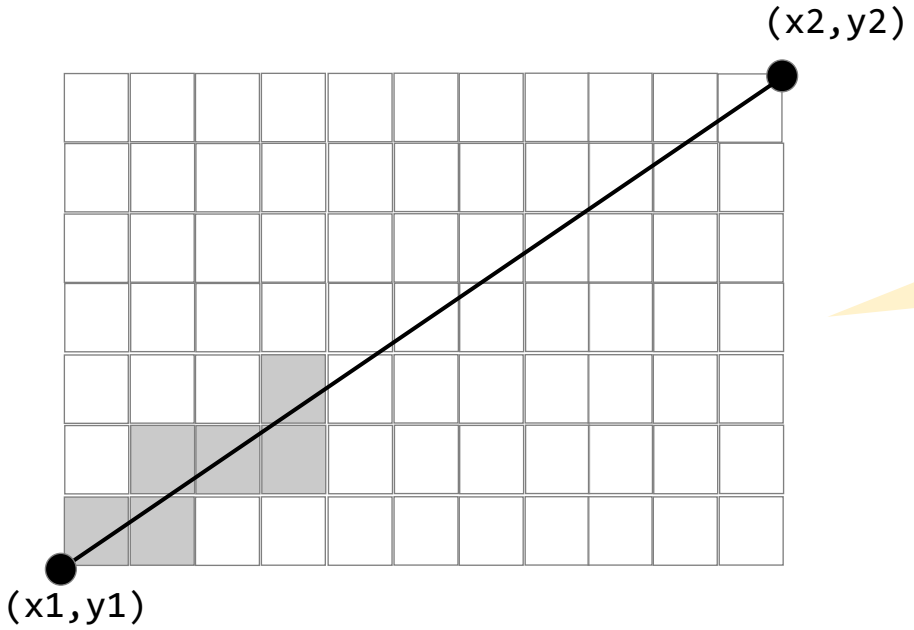
Line drawing



Simplifying assumption:

Focus on lines that go *north-east*

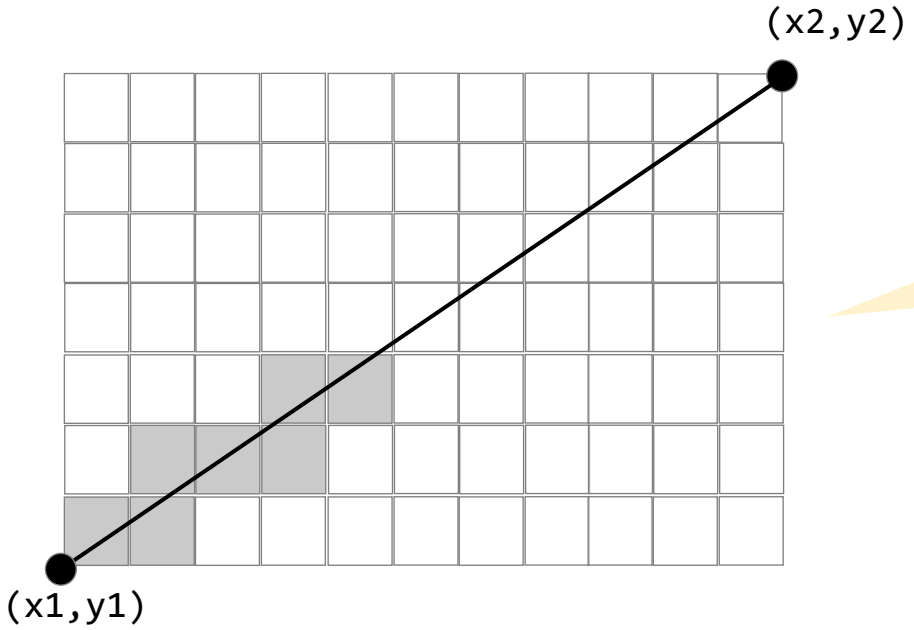
Line drawing



Simplifying assumption:

Focus on lines that go *north-east*

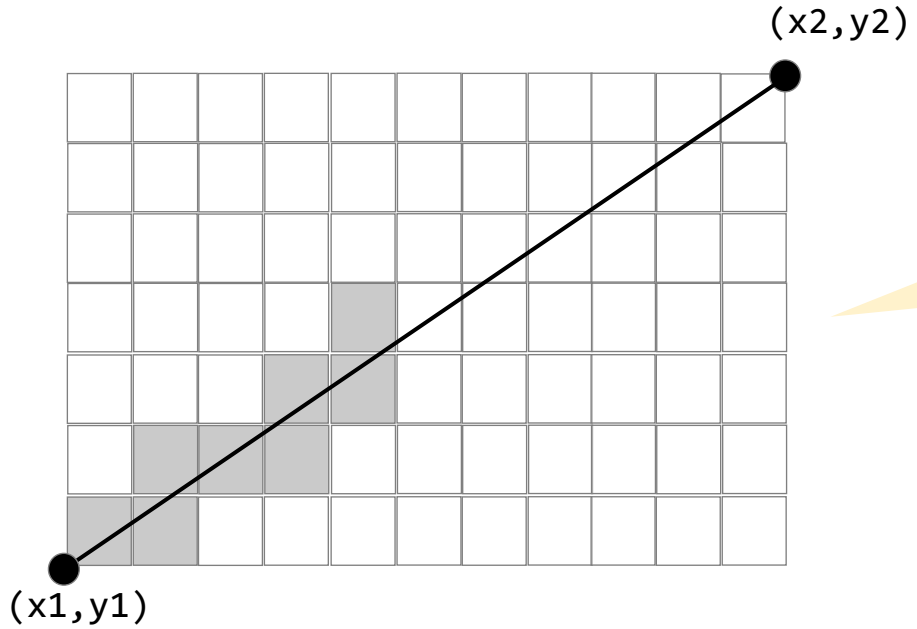
Line drawing



Simplifying assumption:

Focus on lines that go *north-east*

Line drawing



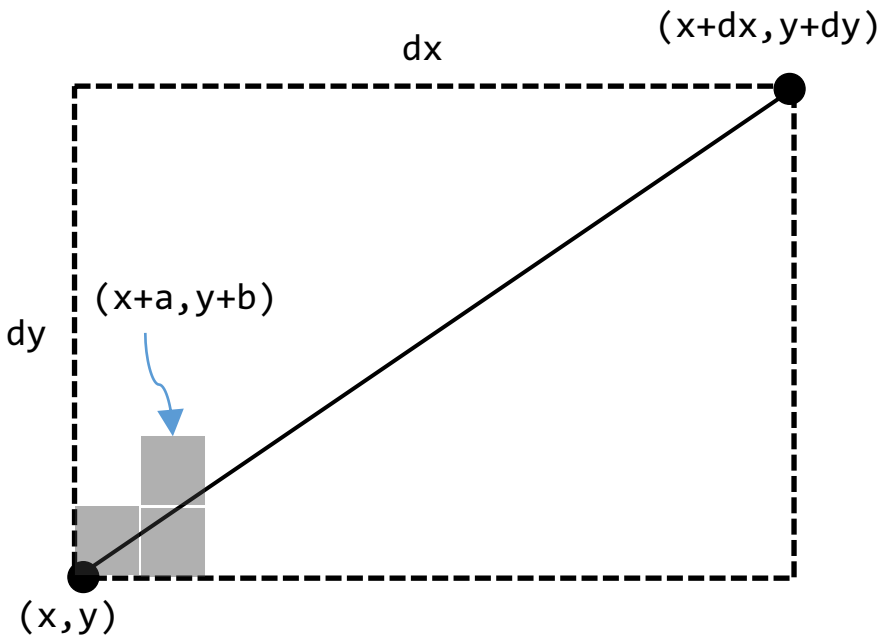
Simplifying assumption:

Focus on lines that go *north-east*

Implemented by a sequence of `drawPixel` operations;

In each stage: Decide *right*, or *up*.

Line drawing



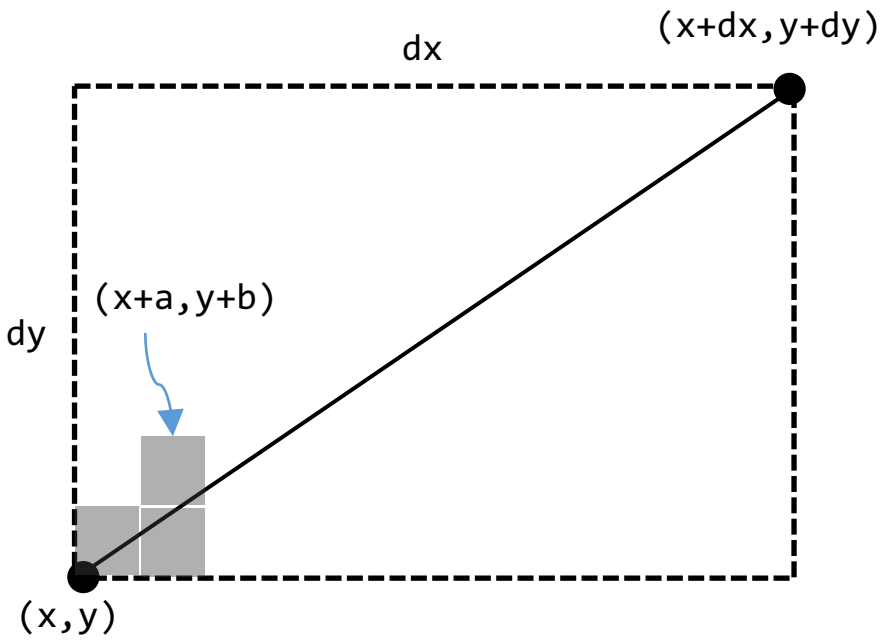
`a` = counts how many pixels we went *right*

`b` = counts how many pixels we went *up*

`drawLine(x1, y1, x2, y2):`

```
x = x1, y = y1, dx = x2-x1,  dy = y2-y1
a = 0,  b = 0
```

Line drawing

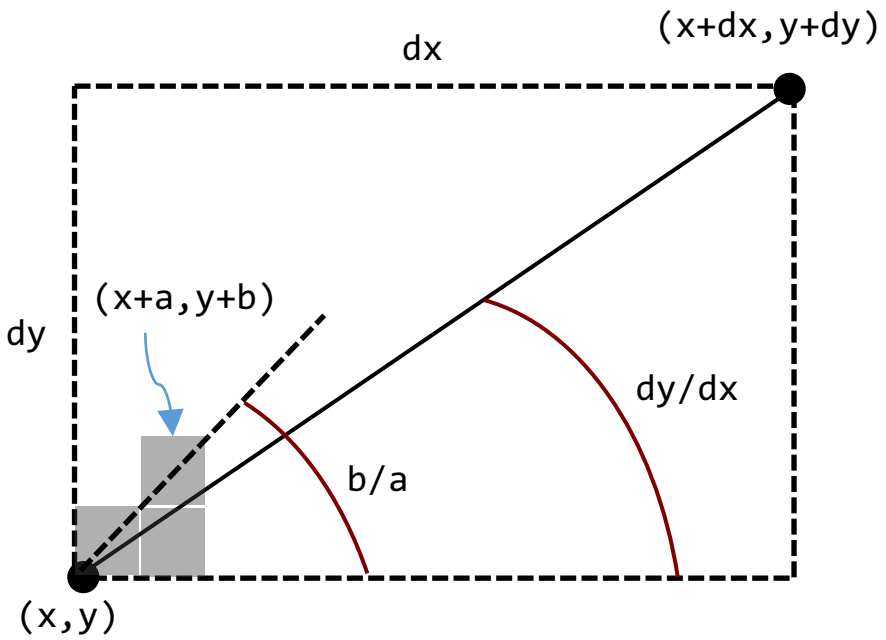


a = counts how many pixels we went right

b = counts how many pixels we went up

```
drawLine(x1, y1, x2, y2):
  x = x1, y = y1, dx = x2-x1, dy = y2-y1
  a = 0, b = 0
  while ((a ≤ dx) and (b ≤ dy))
     drawPixel(x+a, y+b)
     // Computes a and b for the next pixel:
     if (we have to go right): a++
     else:                     b++
```

Line drawing

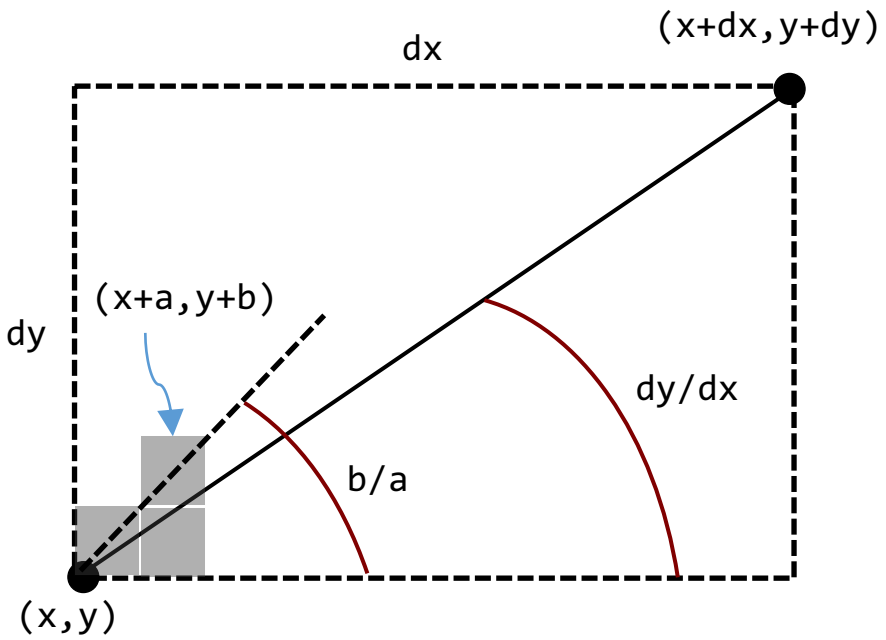


a = counts how many pixels we went right

b = counts how many pixels we went up

```
drawLine(x1, y1, x2, y2):
  x = x1, y = y1, dx = x2-x1, dy = y2-y1
  a = 0, b = 0
  while ((a ≤ dx) and (b ≤ dy))
     drawPixel(x+a, y+b)
     // Computes a and b for the next pixel:
     if (b/a > dy/dx) a++
     else             b++
```

Line drawing



a = counts how many pixels we went right

b = counts how many pixels we went up

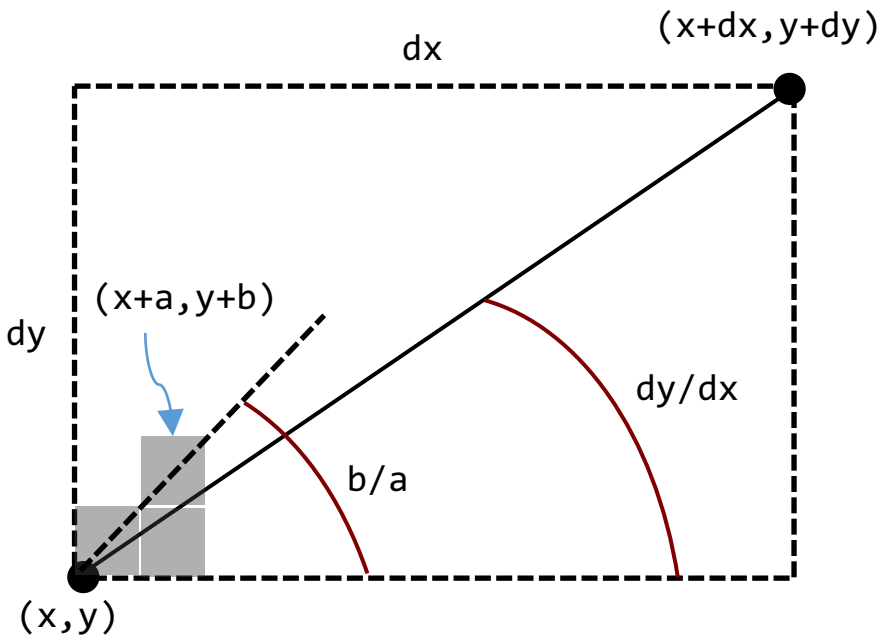
drawLine(x1, y1, x2, y2):

```
x = x1, y = y1, dx = x2-x1,  dy = y2-y1
a = 0,  b = 0
while ((a ≤ dx) and (b ≤ dy))
   drawPixel(x+a, y+b)
   // Computes a and b for the next pixel:
   if (b/a > dy/dx) a++
   else             b++
```

Optimizing the slope condition

1. `(b/a > dy/dx)` has the same value as `(a*dy < b*dx)`
2. let `diff = (a*dy – b*dx)`
3. `(a*dy < b*dx)` has the same value as `diff < 0`

Line drawing



a = counts how many pixels we went right

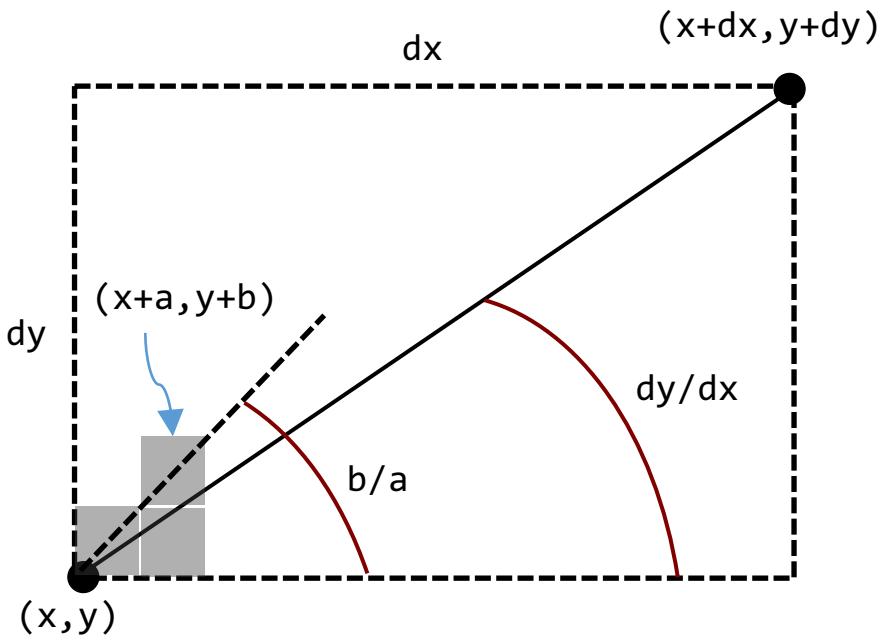
b = counts how many pixels we went up

```
drawLine(x1, y1, x2, y2):
  x = x1, y = y1, dx = x2-x1,  dy = y2-y1
  a = 0,  b = 0
  while ((a ≤ dx) and (b ≤ dy))
     drawPixel(x+a, y+b)
     // Computes a and b for the next pixel:
     if (b/a > dy/dx)  a++
      else             b++
```

Optimizing the slope condition

1. `(b/a > dy/dx)` has the same value as `(a*dy < b*dx)`
2. let `diff` = `(a*dy – b*dx)`
3. `(a*dy < b*dx)` has the same value as `diff < 0`
4. Inspecting `diff`'s definition (2), we see that:
   when `a++`, `diff` goes up by `dy`
   when `b++`, `diff` goes down by `dx`

Line drawing



`a` = counts how many pixels we went right

`b` = counts how many pixels we went up

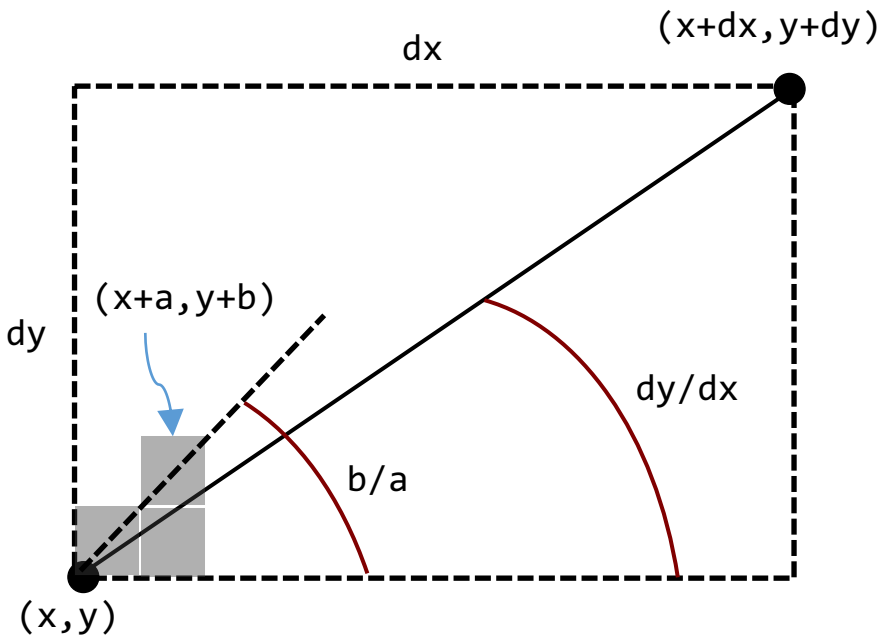
`drawLine(x1, y1, x2, y2):`

```
x = x1, y = y1, dx = x2-x1,  dy = y2-y1
a = 0,  b = 0,  diff = 0
while ((a ≤ dx) and (b ≤ dy))
   drawPixel(x+a, y+b)
   // Computes a and b for the next pixel:
   if (diff < 0) {a++, diff += dy}
    else         {b++, diff -= dx}
```

Optimizing the slope condition

1. `(b/a > dy/dx)` has the same value as `(a*dy < b*dx)`
2. let `diff` = `(a*dy – b*dx)`
3. `(a*dy < b*dx)` has the same value as `diff < 0`
4. Inspecting `diff`'s definition (2), we see that:
   when `a++`, `diff` goes up by `dy`
   when `b++`, `diff` goes down by `dx`

Line drawing



a = counts how many pixels we went right

b = counts how many pixels we went up

drawLine(x1, y1, x2, y2):

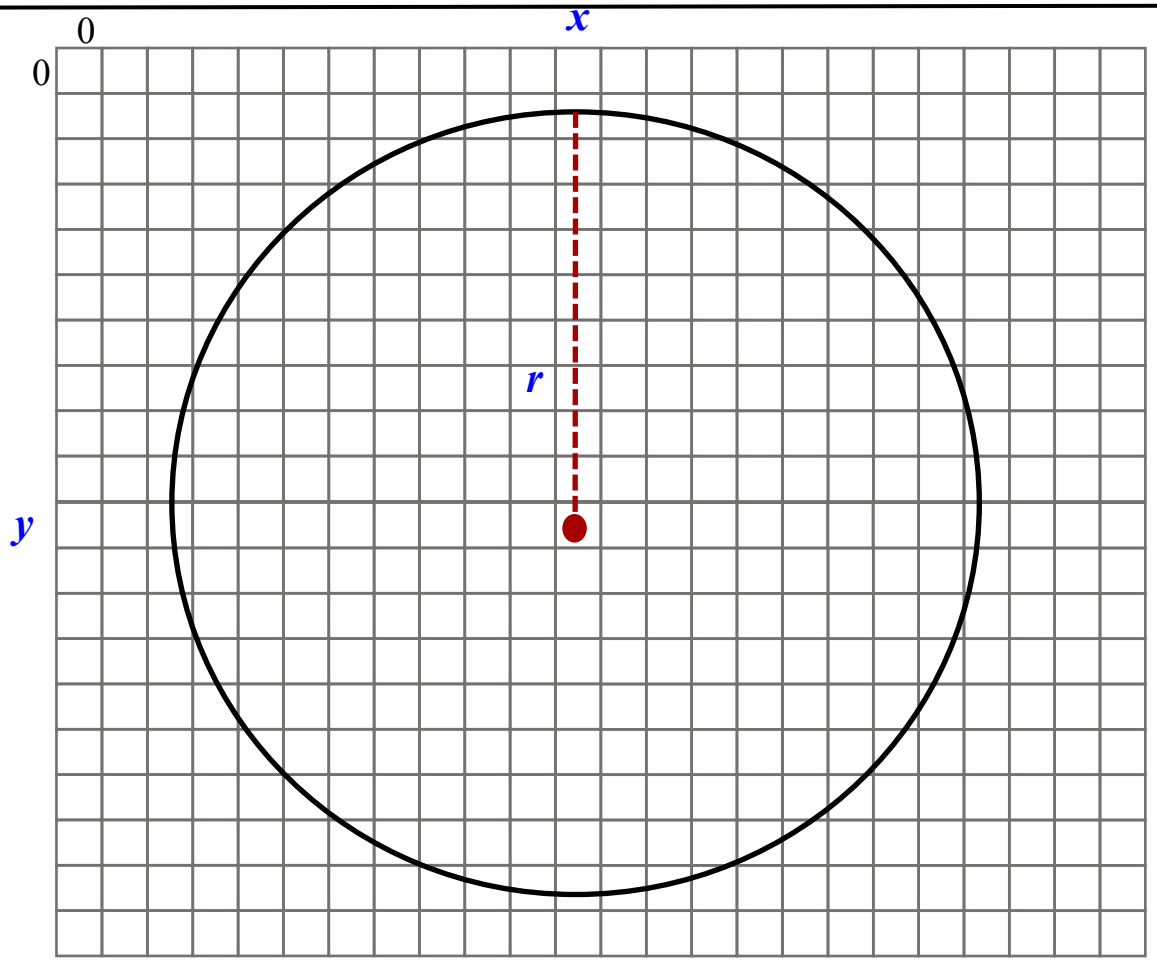
```
x = x1, y = y1, dx = x2-x1, dy = y2-y1
a = 0, b = 0, diff = 0
while ((a ≤ dx) and (b ≤ dy))
   drawPixel(x+a, y+b)
   // Computes a and b for the next pixel:
   if (diff < 0) {a++, diff += dy}
   else          {b++, diff -= dx}
```

Involves only addition / subtraction operations;

Since the screen size is fixed, essentially $O(1)$;

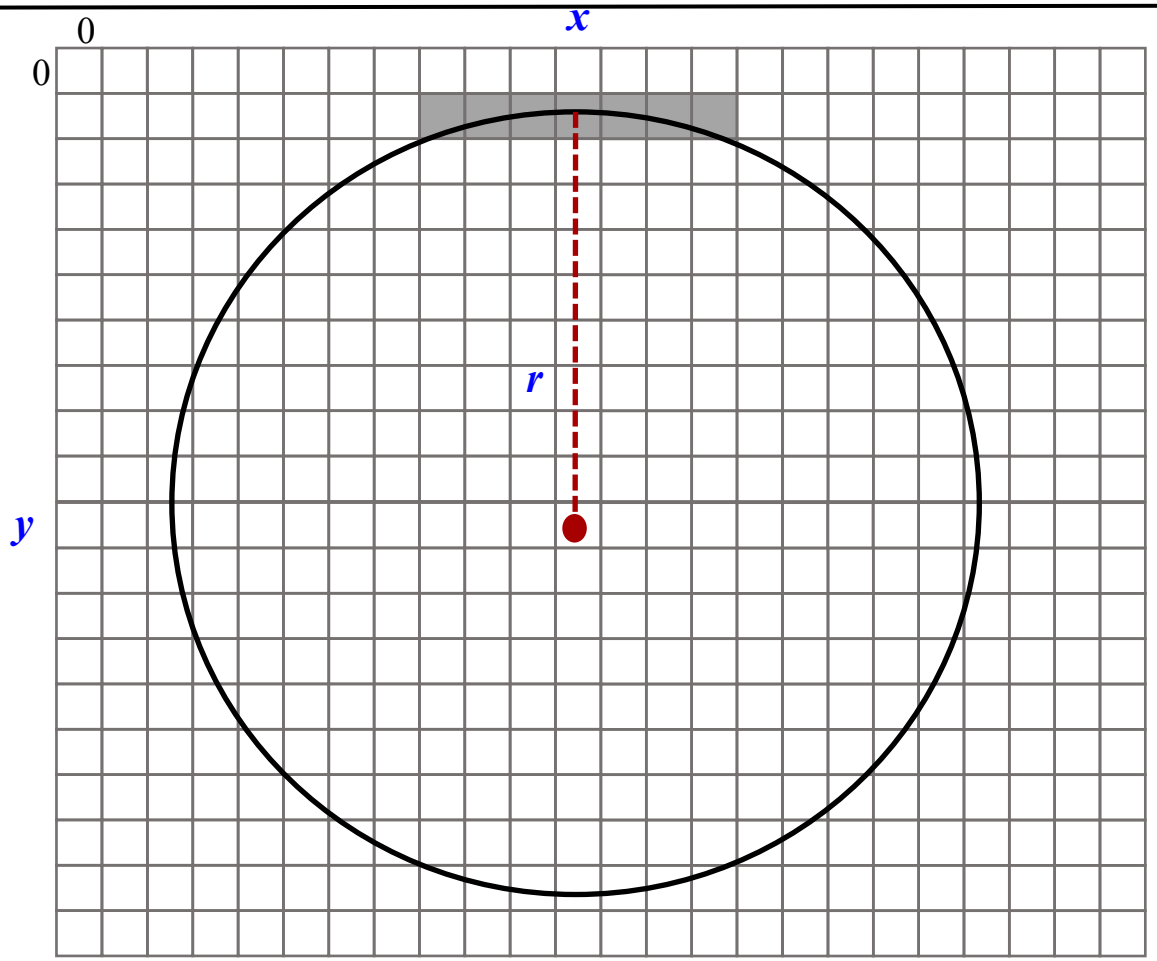
Can be implemented efficiently in either software or hardware.

Circle drawing



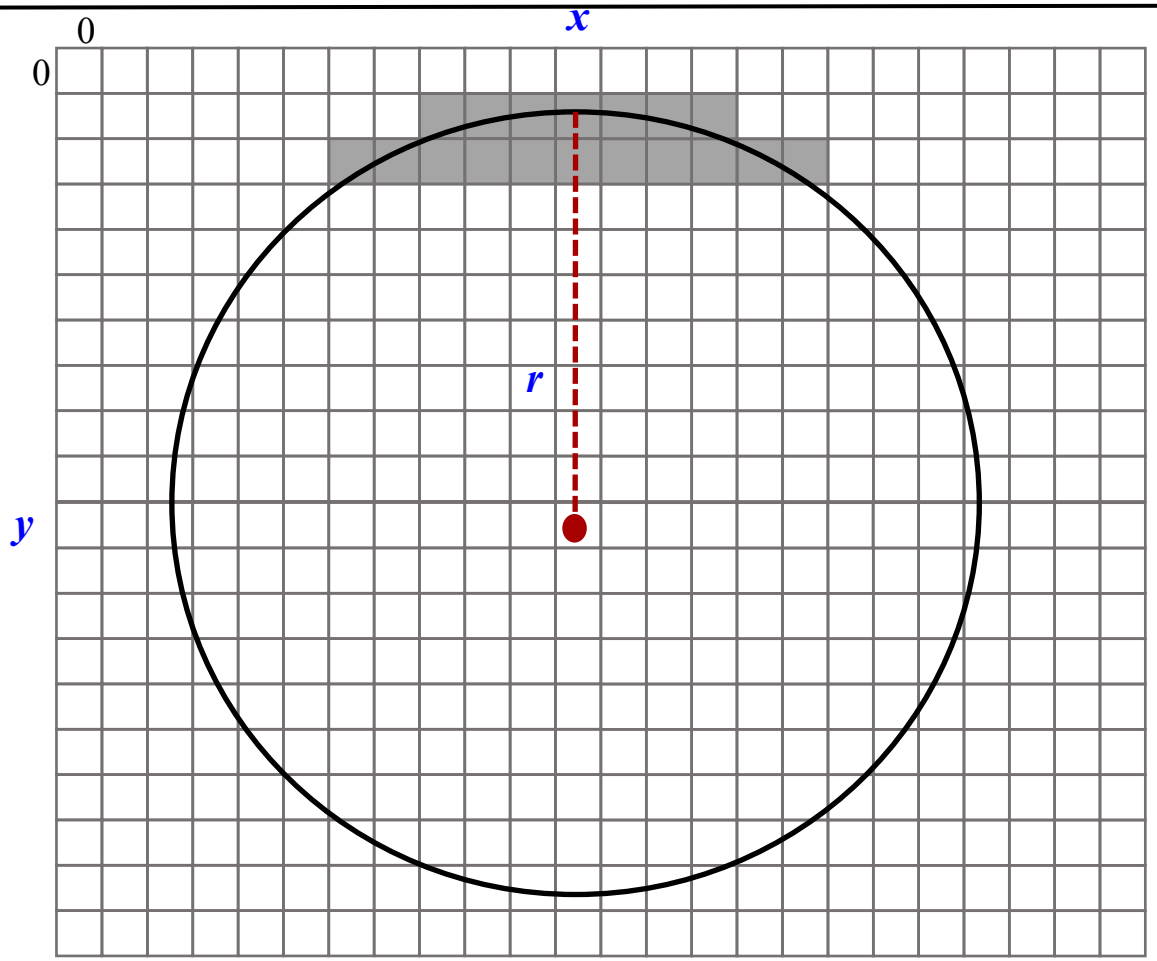
drawCircle (x, y, r):

Circle drawing



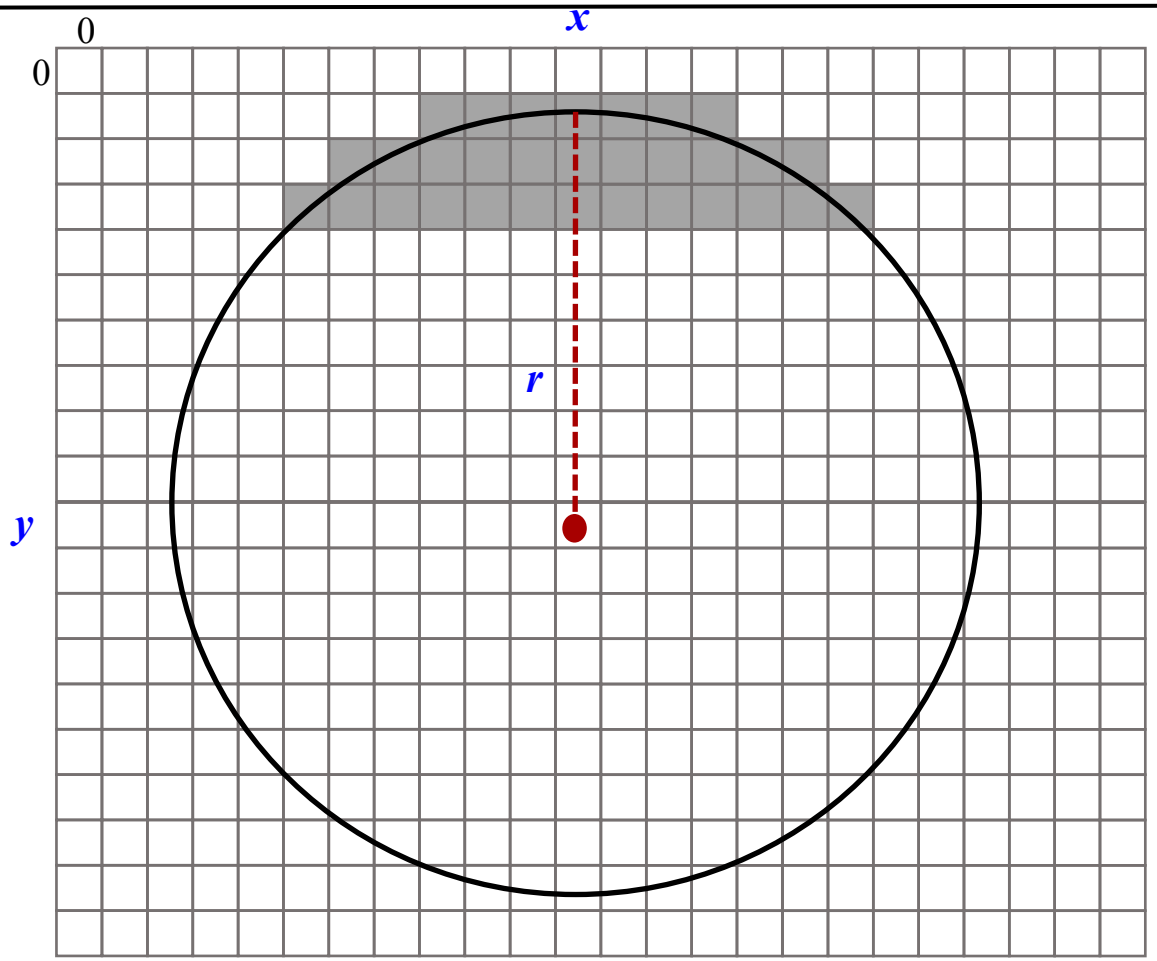
drawCircle (x, y, r):

Circle drawing



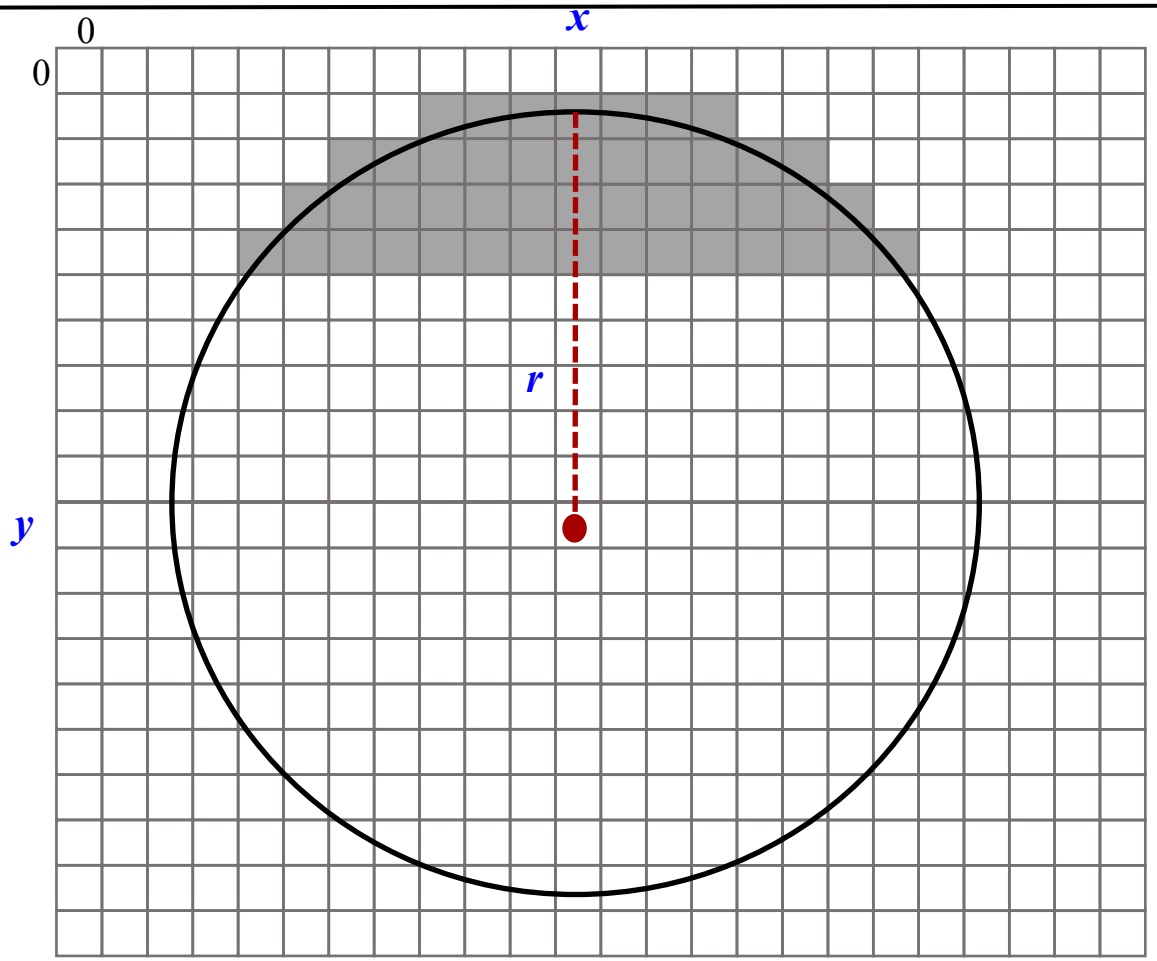
drawCircle (x, y, r):

Circle drawing



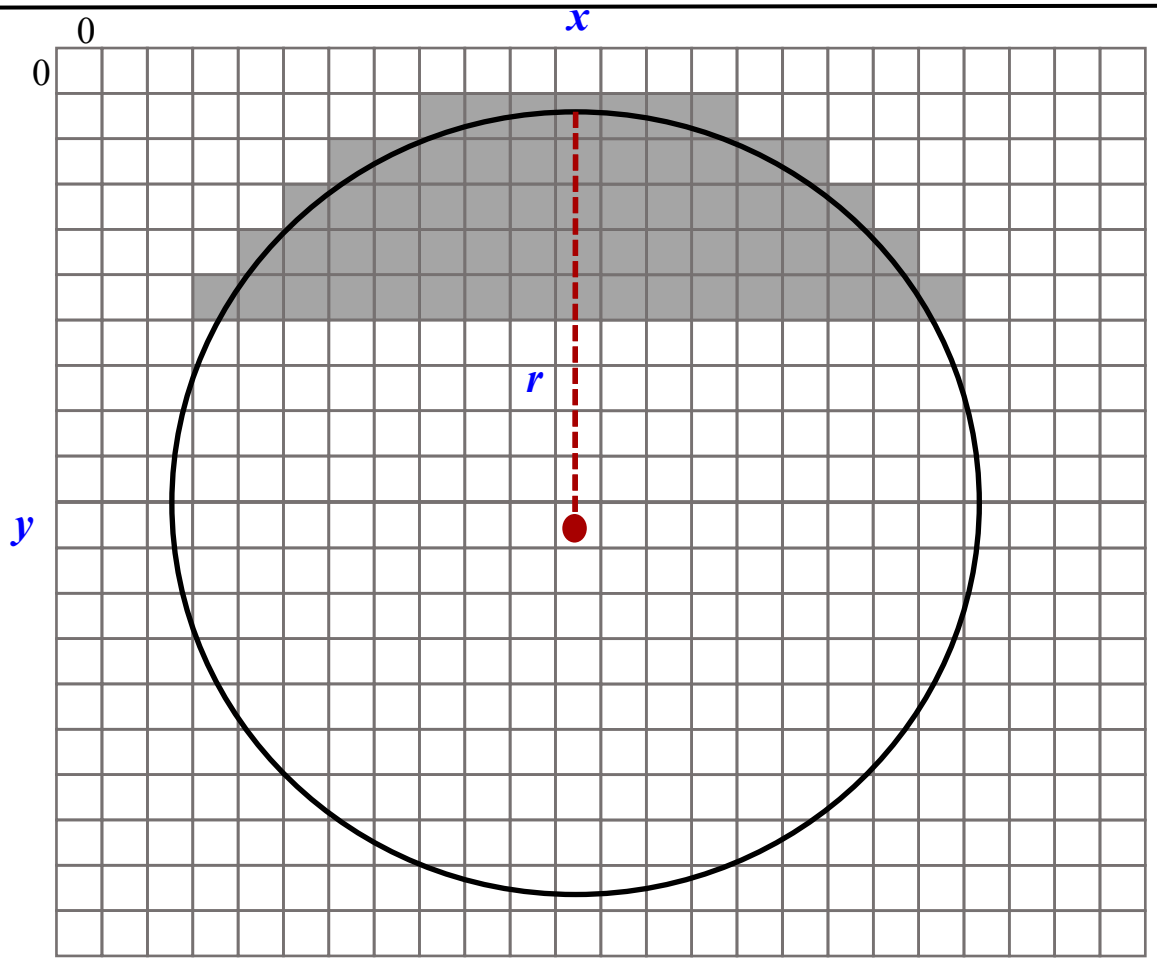
drawCircle (x, y, r):

Circle drawing



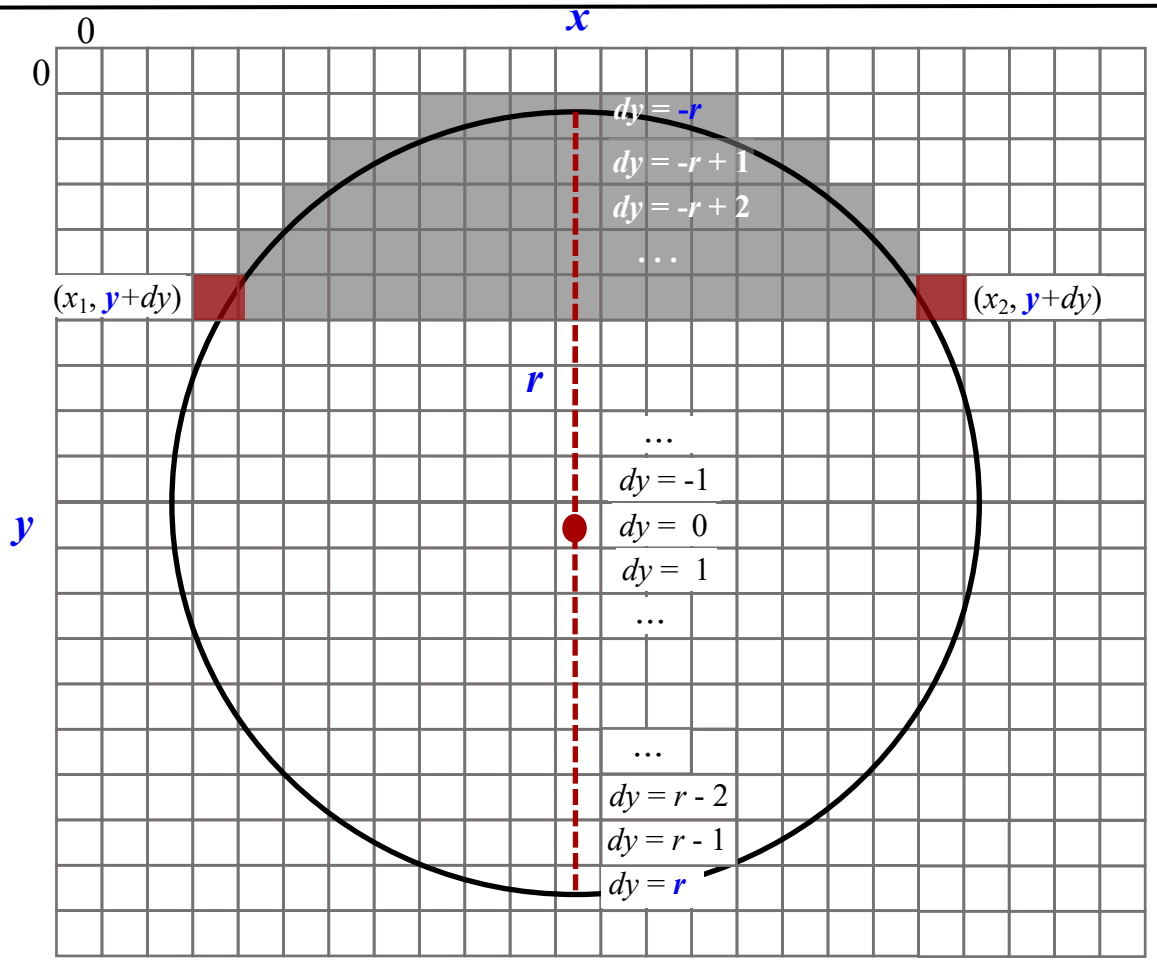
drawCircle (x, y, r):

Circle drawing



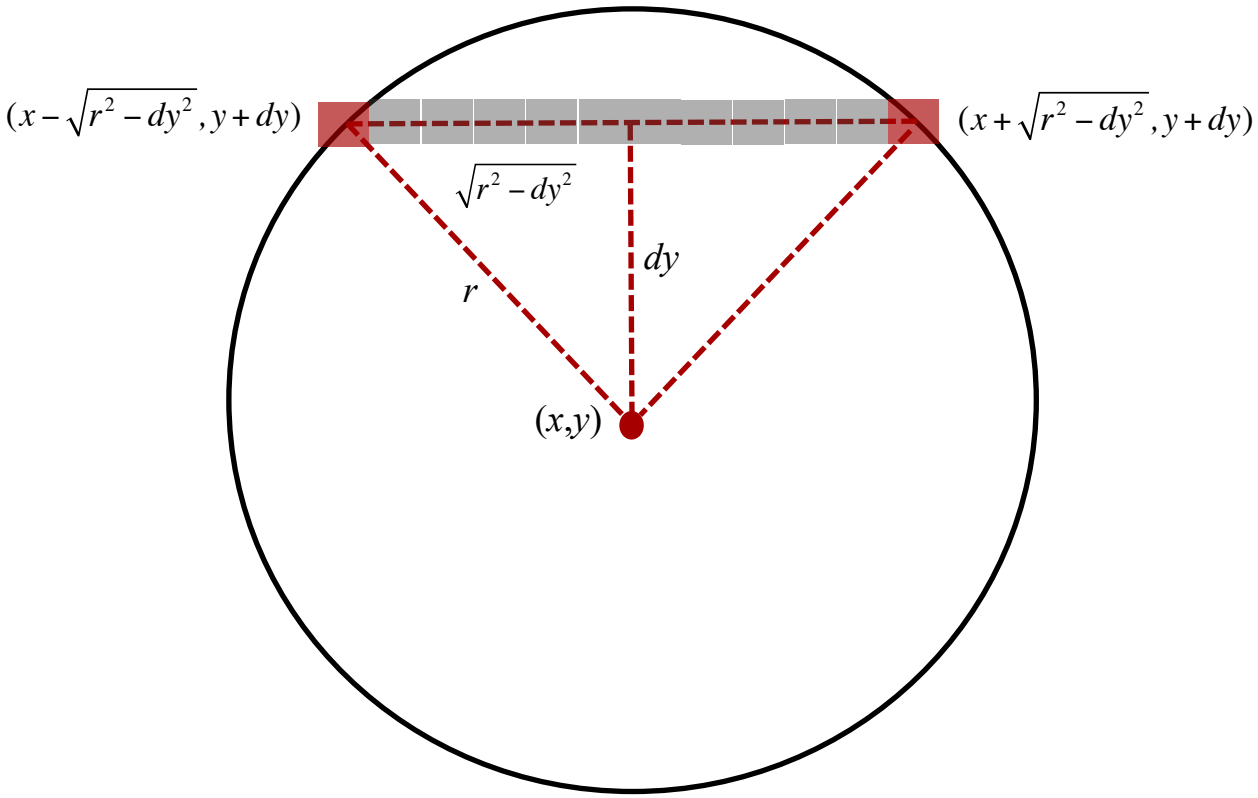
drawCircle (x, y, r):

Circle drawing



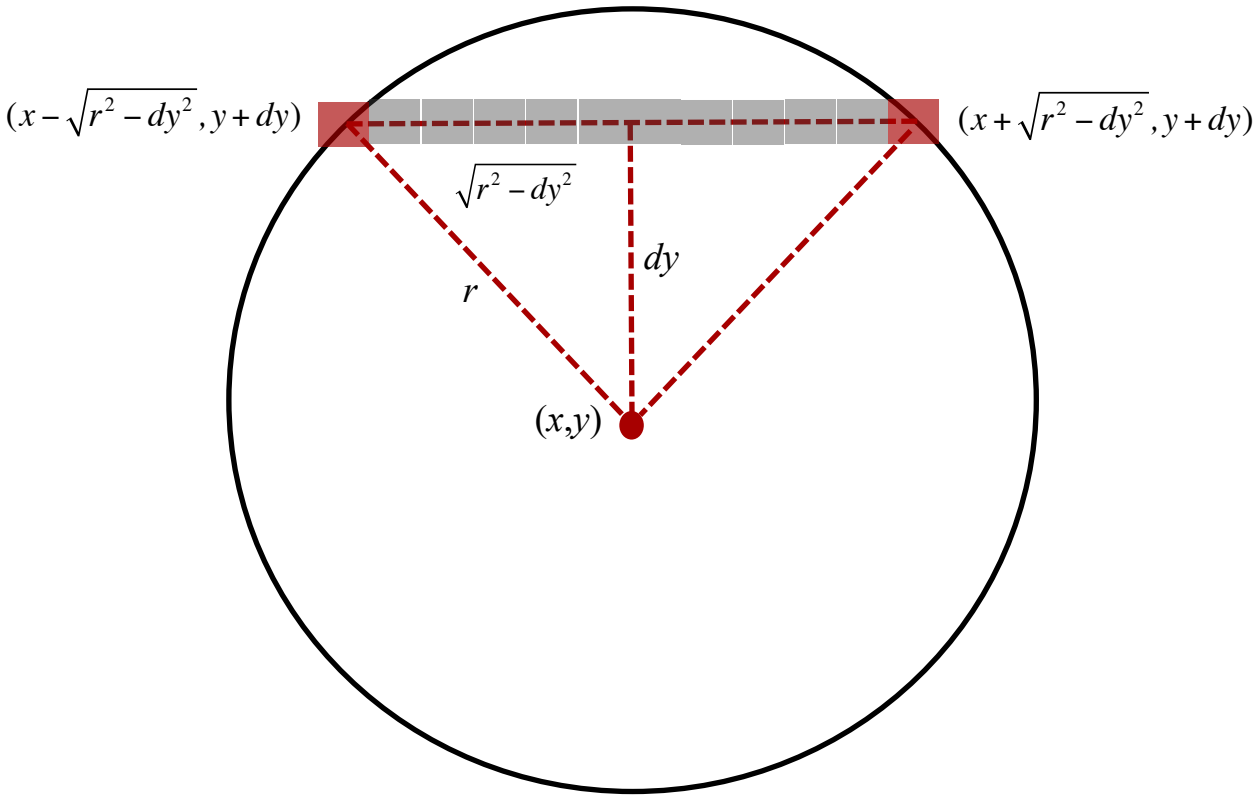

```
drawCircle (x, y, r):
    for each dy = –r  to  r  do:
        drawLine ( x1, y+dy,   x2, y+dy )
```

Circle drawing




```
drawCircle (x, y, r):
    for each dy = –r  to  r  do:
        drawLine ( x1, y+dy,   x2, y+dy )
```

Circle drawing



drawCircle (x, y, r):

for each $dy = -r$ to r do:

drawLine ($x-\sqrt{r^2-dy^2}, y+dy$, $x+\sqrt{r^2-dy^2}, y+dy$)

Implementation notes: `drawPixel`

```
/* Sets pixel (x,y) to the current color */
function void drawPixel(int x, int y) {
    address = 32 * y + x / 16
    value = Memory.peek[16384 + address]
    set the (x % 16)th bit of value to the current color
    do Memory.poke(address,value)
}
```

- Uses the services of `Memory.peek`, `Memory.poke`
- To set a single bit in a 16-bit value, use logical 16-bit operations.

Implementation notes: `drawLine`

```
/* Draws a line from (x1,y1) to (x2,y2)  */
function void drawLine(int x1, int y1, int x2, int y2)
   dx = x2 - x1; dy = y2 – y1;
   a = 0; b = 0; diff = 0;
   while ((a <= dx) and (b <= dy))
      drawPixel(x + a, y + b);
      // decides which way to go (up, or right):
      if (diff < 0) { a++; diff += dy; }
      else          { b++; diff -= dx; }
```

- Reminder: The screen's top-left corner is pixel(0,0)
- Implement the algorithm that draws lines that go north-east
- Extend the algorithm to draw lines in all four directions
- But: Handle the drawing of horizontal and vertical lines as special cases.

Implementation notes: `drawCircle`

drawCircle (x, y, r):

for each $dy = -r$ to r do:

drawLine ($x - \sqrt{r^2 - dy^2}, y + dy$, $x + \sqrt{r^2 - dy^2}, y + dy$)

Can potentially lead to overflow;

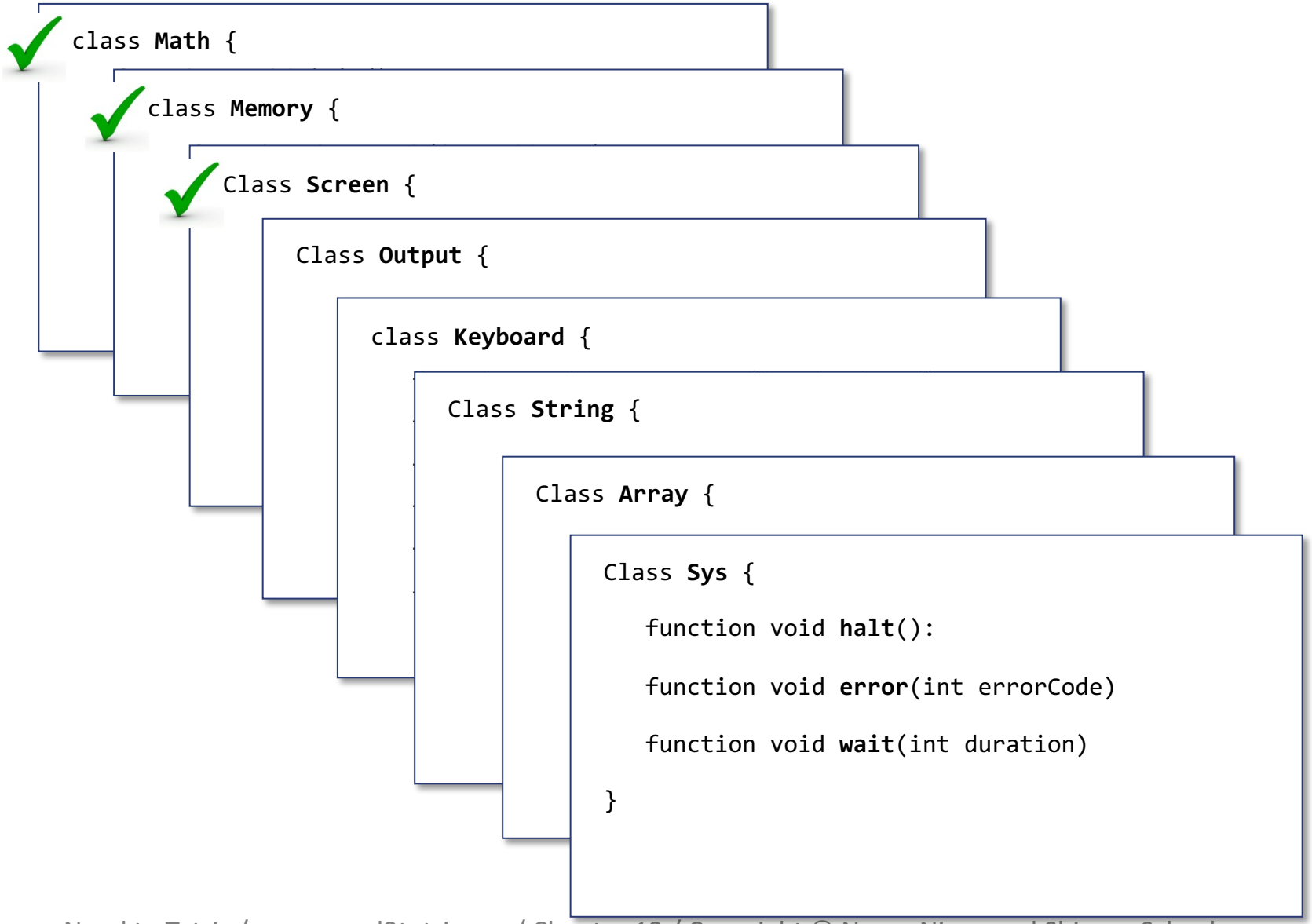
To handle, limit r to be no greater than 181.

Graphics library

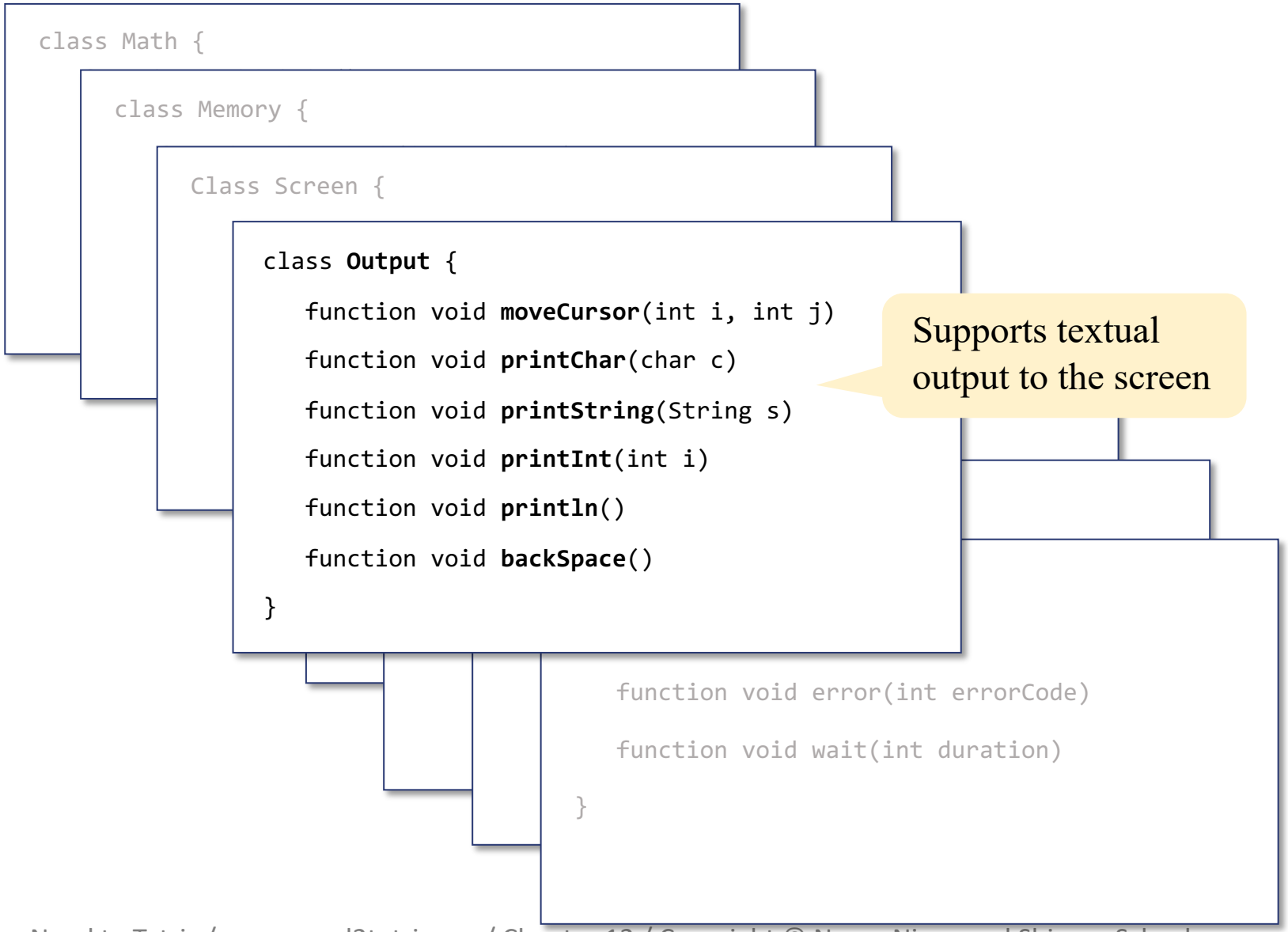
```
Class Screen {
   function void clearScreen()
   function void setColor(boolean b)
✓  function void drawPixel(int x, int y)
✓  function void drawLine(int x1, int y1, int x2, int y2)
   function void drawRectangle(int x1, int y1, int x2, int y2)
✓  function void drawCircle(int x, int y, int r)
}
```

The implementation of the remaining `Screen` functions is simple.

The Jack OS


class Math {
class Memory {
Class Screen {
Class Output {
class Keyboard {
Class String {
Class Array {
Class Sys {
function void halt():
function void error(int errorCode)
function void wait(int duration)
}

The Jack OS

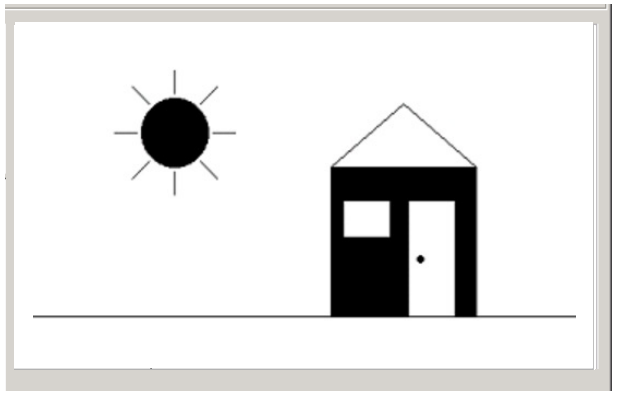

class Math {
class Memory {
Class Screen {
class Output {
function void moveCursor(int i, int j)
function void printChar(char c)
function void printString(String s)
function void printInt(int i)
function void println()
function void backSpace()
}
Supports textual output to the screen
function void error(int errorCode)
function void wait(int duration)
}

Output modes

Graphical output

Screen abstraction:
256 rows of 512 pixels, b&w

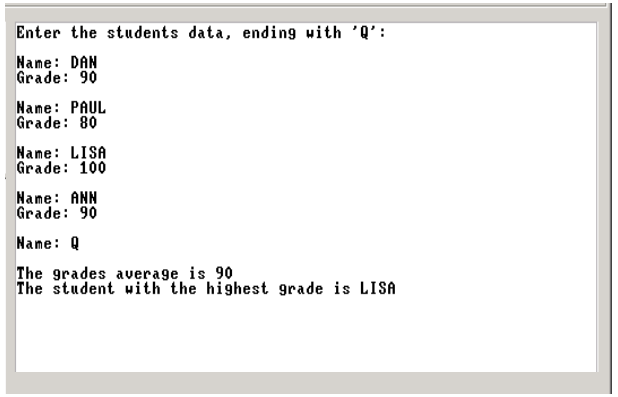
Driven by the Jack OS class `Screen`



Textual output

Screen abstraction:
23 rows of 64 characters, b&w

Driven by the Jack OS class `Output`



The Hack character set

printable characters

key	code
(space)	32
!	33
“	34
#	35
$	36
%	37
&	38
‘	39
(	40
)	41
*	42
+	43
,	44
-	45
.	46
/	47

key	code
0	48
1	49
…	…
9	57

key	code
:	58
;	59
<	60
=	61
>	62
?	63
@	64

key	code
A	65
B	66
C	…
…	…
Z	90

key	code
[	91
/	92
]	93
^	94
_	95
`	96

key	code
a	97
b	98
c	99
…	…
z	122

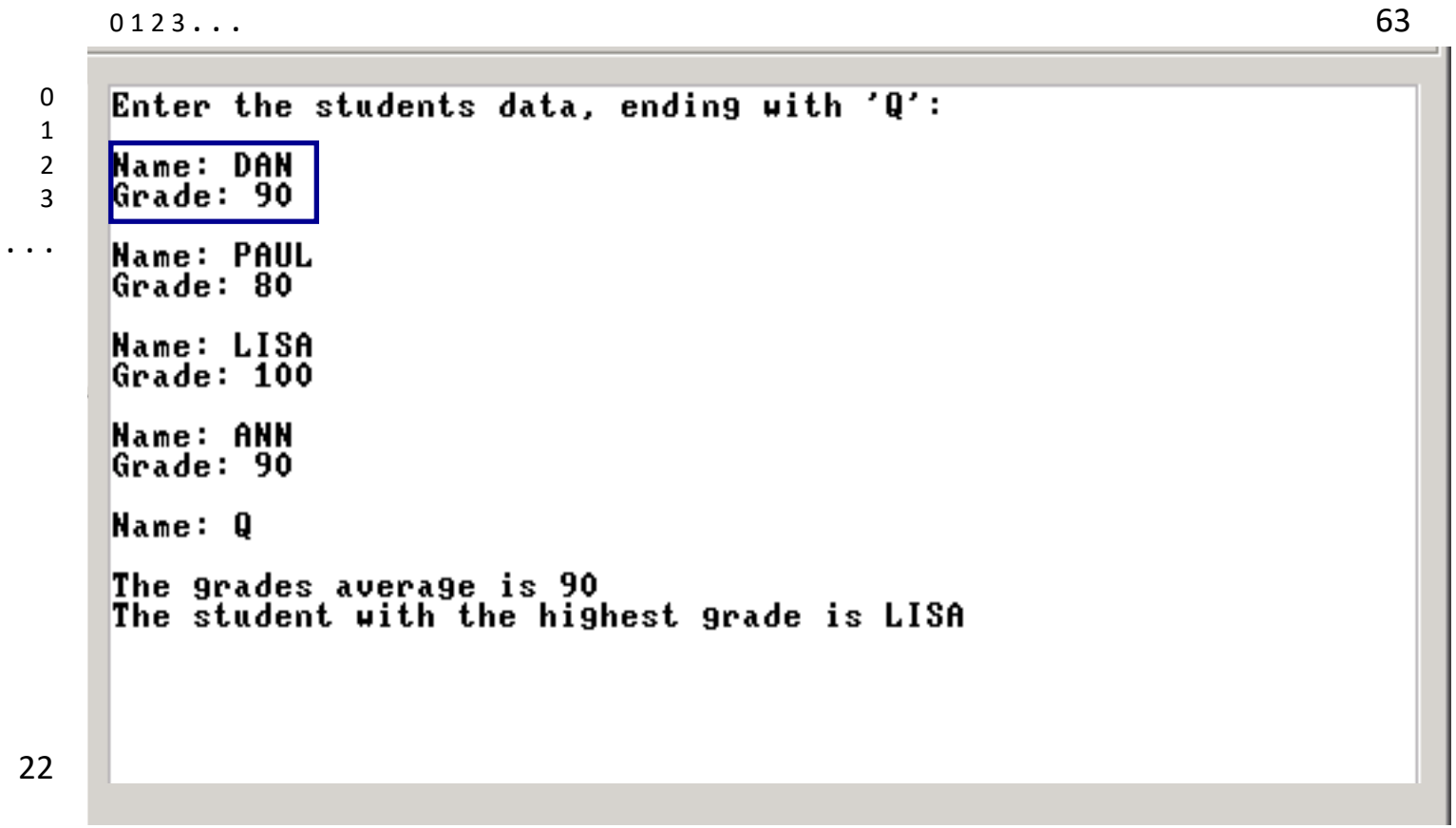
key	code
{	123
|	124
}	125
~	126

(Subset of Unicode)

non-printables

key	code
newline	128
backspace	129
left arrow	130
up arrow	131
right arrow	132
down arrow	133
home	134
end	135
Page up	136
Page down	137
insert	138
delete	139
esc	140
f1	141
. . .	. . .
f12	152

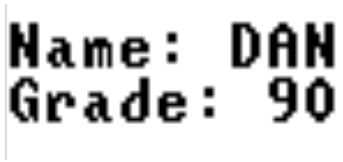
Textual output



Challenges

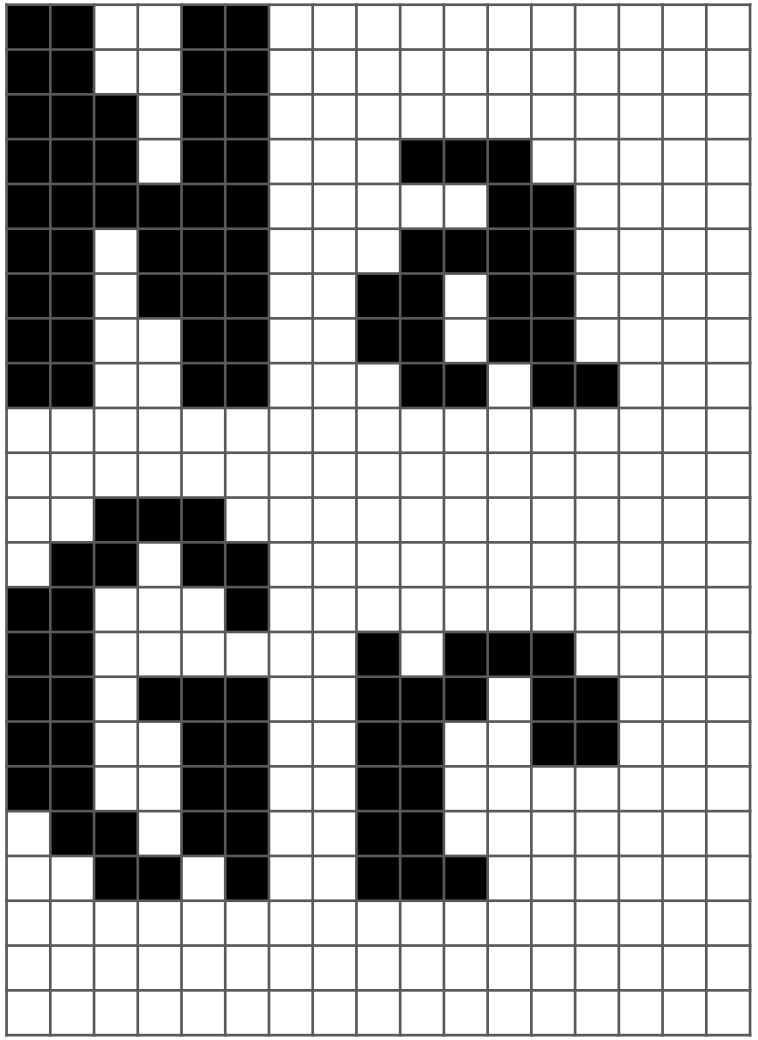
- Use a 256 by 512 pixels grid to realize a 23 by 64 characters grid
- Display and handle characters in a human-friendly way
  (fonts, spacing, cursor, backspace, …)

Font

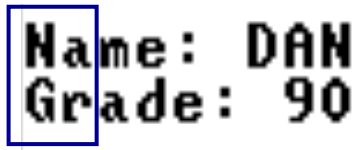


Font

```
Name: DAN
Grade: 90
```



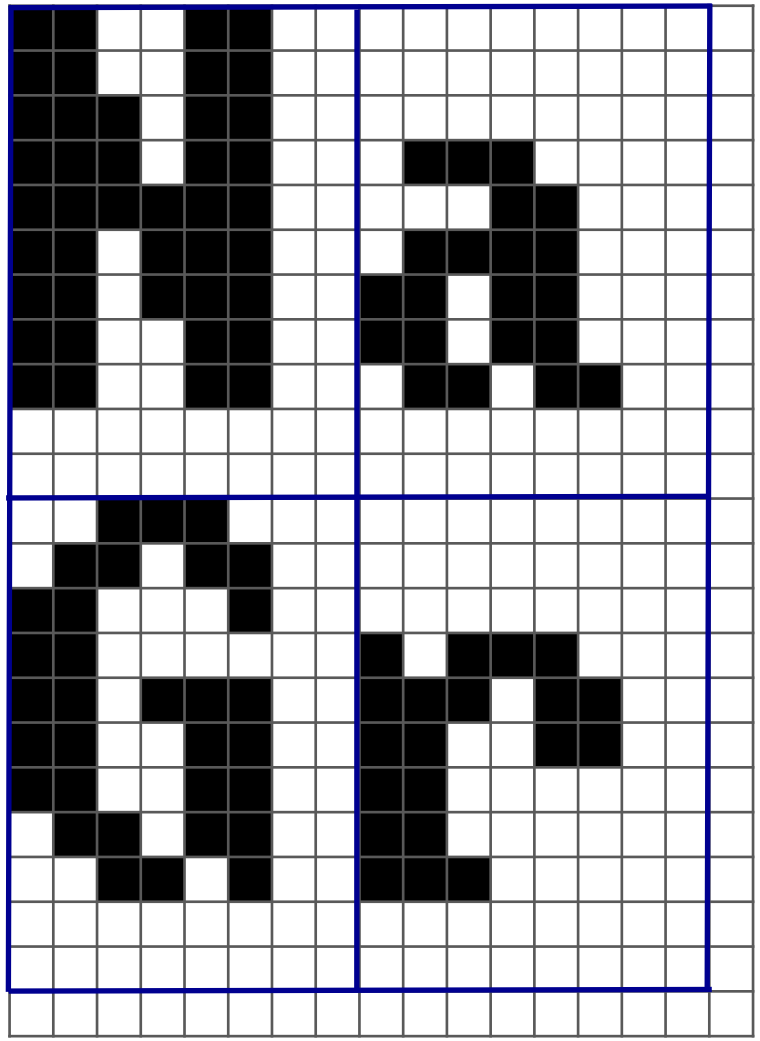
Font



Hack font

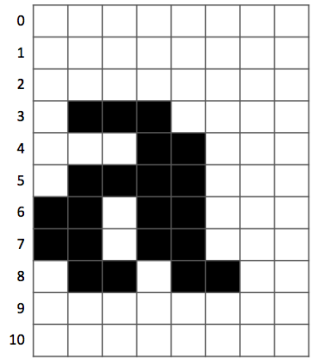
Each character occupies a fixed 11-pixel high and 8-pixel wide “frame”;

The frame includes 2 empty right columns and 1 empty bottom row for character spacing.

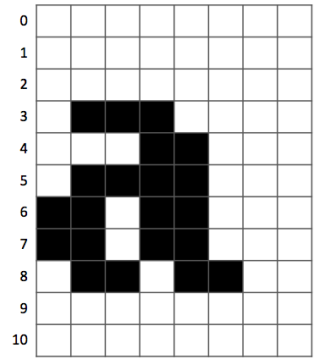


Font

```
class Output {
    static Array charMaps;  // character bitmaps
    ...
```

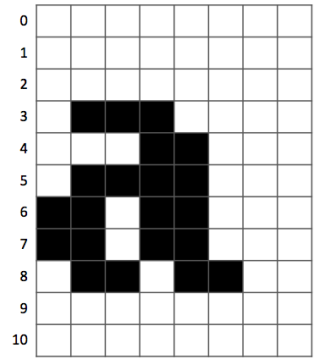


Font




```
class Output {
    static Array charMaps;  // character bitmaps
    ...
    // Builds the character map array
    function void initMap() {
        let charMaps = Array.new(127);

        // Uses a helper function to assign a bitmap to each character in the character set.
        do Output.create(97, 0,0,0,14,24,30,27,27,54,0,0);     // a
        do Output.create(98, 3,3,3,15,27,51,51,51,30,0,0);     // b
        do Output.create(99, 0,0,0,30,51,3,3,51,30,0,0);       // c
        ...
        do Output.create(48, 12,30,51,51,51,51,51,30,12,0,0); // 0
        do Output.create(49, 12,14,15,12,12,12,12,12,63,0,0); // 1
        do Output.create(50, 30,51,48,24,12,6,3,51,63,0,0);    // 2
        do Output.create(51, 30,51,48,48,28,48,48,51,30,0,0); // 3
        ...
        do Output.create(32, 0,0,0,0,0,0,0,0,0,0,0);           // (space)
        do Output.create(33, 12,30,30,30,12,12,0,12,12,0,0);   // !
        do Output.create(34, 54,54,20,0,0,0,0,0,0,0,0);        // "
        do Output.create(35, 0,18,18,63,18,18,63,18,18,0,0);   // #
        ...
        // black square (used for non printable characters)
        do Output.create(0,  63,63,63,63,63,63,63,63,63,0,0);
        return
}
```

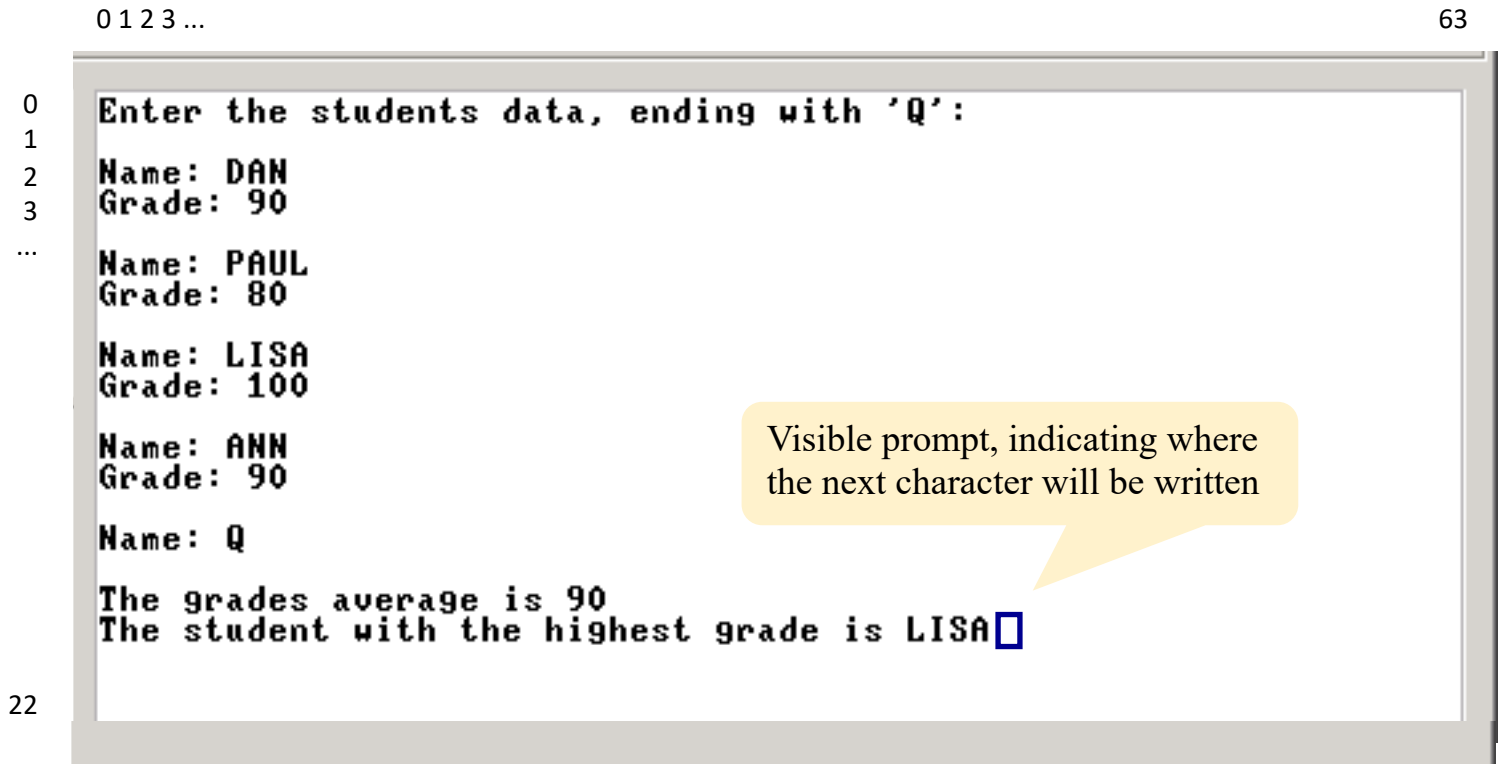
Font



```
class Output {
    static Array charMaps;  // character bitmaps
    ...
    // Builds the character map array
    function void initMap() {
        let charMaps = Array.new(127);

        // Uses a helper function to assign a bitmap to each character in the character set.
        do Output.create(97, 0,0,0,14,24,30,27,27,54,0,0);     // a
        do Output.create(98, 3,3,3,15,27,51,51,51,30,0,0);     // b
        do Output.create(99, 0,0,0,30,51,3,3,51,30,0,0);       // c
        ...
        do Output.create(48, 12,30,51,51,51,51,51,30,12,0,0); // 0
        do Output.create(49, 12,14,15,12,12,12,12,12,63,0,0); // 1
        do Output.create(50, 30,51,48,24,12,6,3,51,63,0,0);    // 2
        do Output.c
        ...
        do Output.c
        do Output.c
        do Output.c
        do Output.c
        ...
        // black squa
        do Output.c
        return
}
```

```
// Creates a bitmap for the given char code, using the given values.
function void create(int index, int a, int b, int c, int d, int e,
                                 int f, int g, int h, int i, int j, int k) {
   var Array map;
   let map = Array.new(11);
   let charMaps[index] = map;
   let map[0] = a;  let map[1] = b;  let map[2] = c;
   let map[3] = d;  let map[4] = e;  let map[5] = f;
   let map[6] = g;  let map[7] = h;  let map[8] = i;
   let map[9] = j;  let map[10] = k;
   return;
}
```

Cursor



Cursor management

If called to display `newLine`: Moves the cursor to the beginning of the next line

If called to display `backspace`: Moves the cursor one column left

If called to display any other character: Displays the character, and moves the cursor one column to the right.

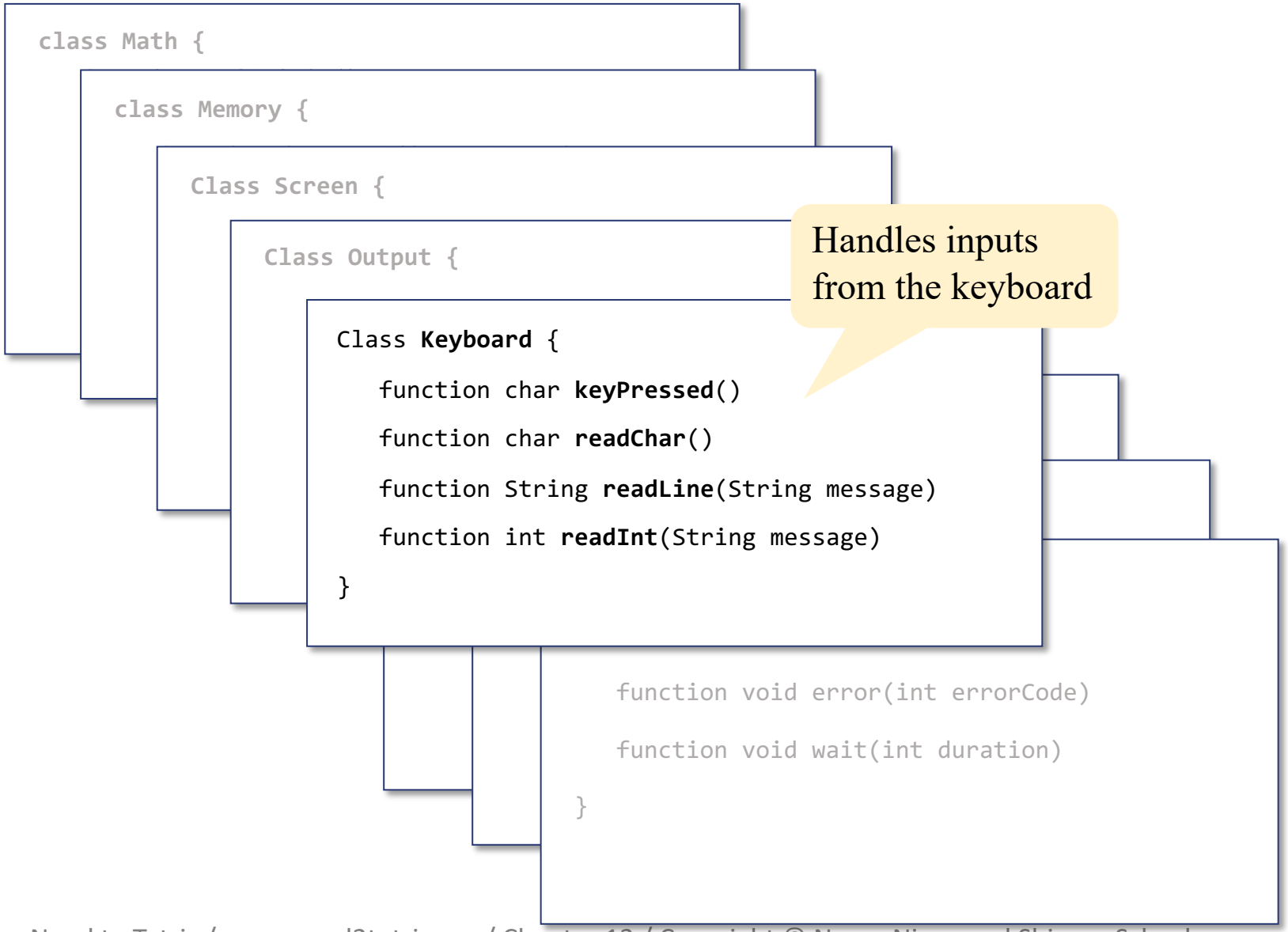
`Output` class API

```
class Output {
  function void init() {}

  /* Moves the cursor to the j'th column of the i'th row, erasing the character that was there. */
  function void moveCursor(int i, int j) {}

  /* Displays c at the cursor location, and advances the cursor one column right. */
  function void printChar(char c) {}

  /* Displays str starting at the cursor location, and advances the cursor appropriately. */
  function void printString(String str) {}

  /* Displays i starting at the cursor location, and advances the cursor appropriately. */
  function void printInt(int i) {}

  /* Advances the cursor to the beginning of the next line. */
  function void println() {}

  /* Erases the character that was last written and moves the cursor one column back. */
  function void backSpace() {}
}
```

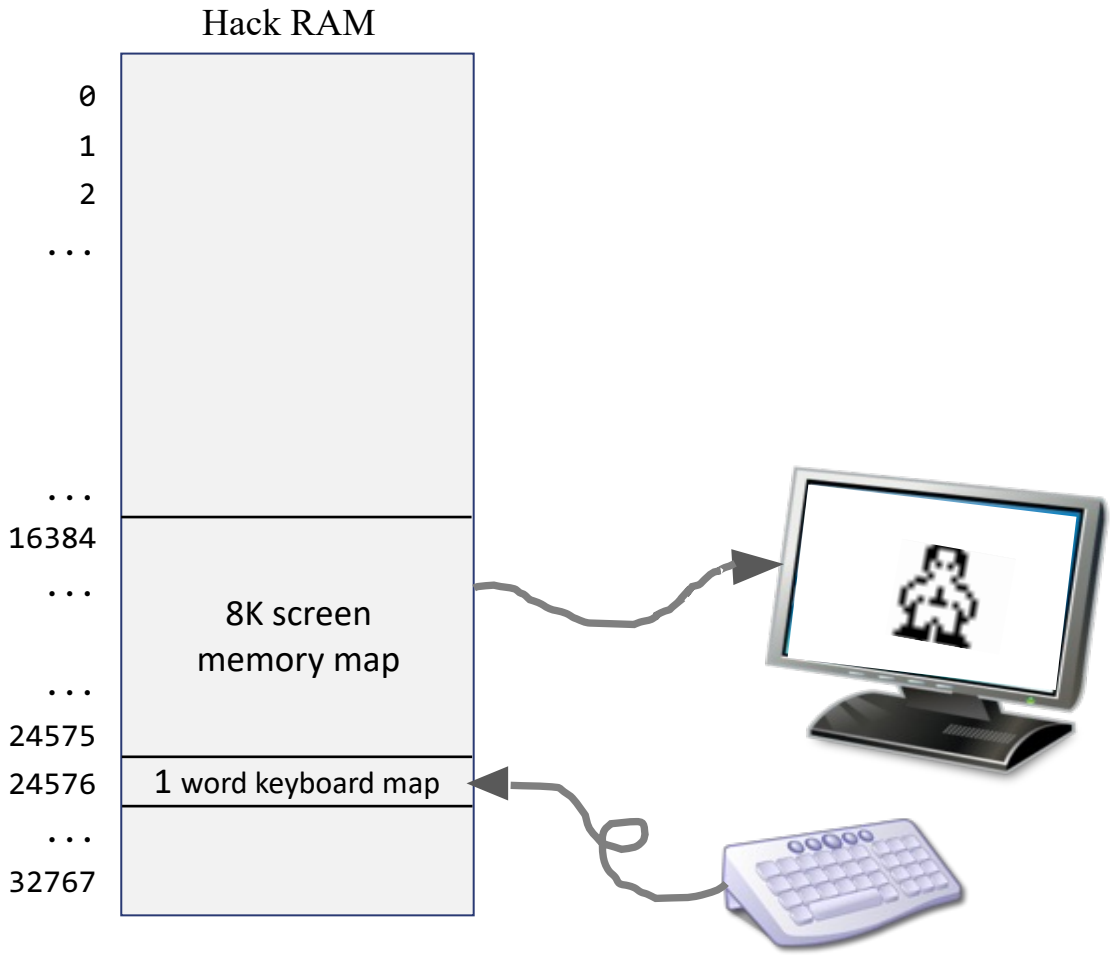
Implementation notes

- Each function prints character bitmaps (elements of the `charMaps` array) and moves the cursor accordingly;
- The Hack font implementation (`charMaps`) is given in the supplied `Output` Jack class.

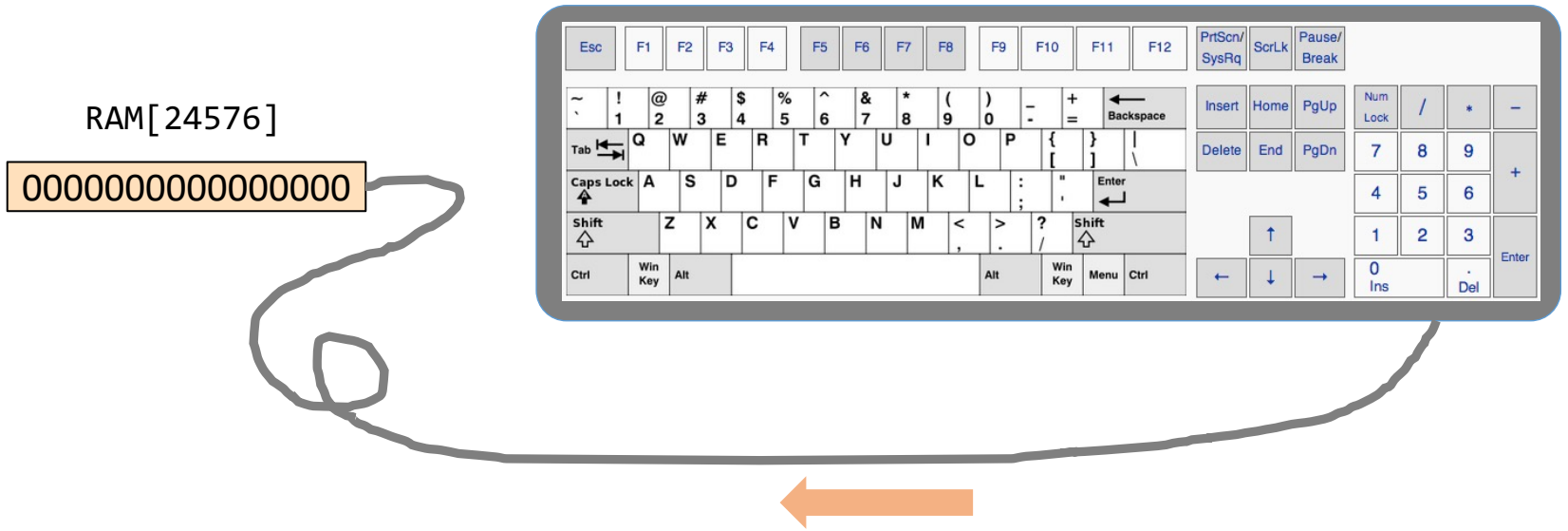
The Jack OS


class Math {
class Memory {
Class Screen {
Class Output {
Class Keyboard {
function char keyPressed()
function char readChar()
function String readLine(String message)
function int readInt(String message)
}
function void error(int errorCode)
function void wait(int duration)
}
Handles inputs from the keyboard

Hack I/O


Hack RAM
0
1
2
...
...
16384
...
...
24575
24576
...
32767
8K screen
memory map
1 word keyboard map

Keyboard memory map

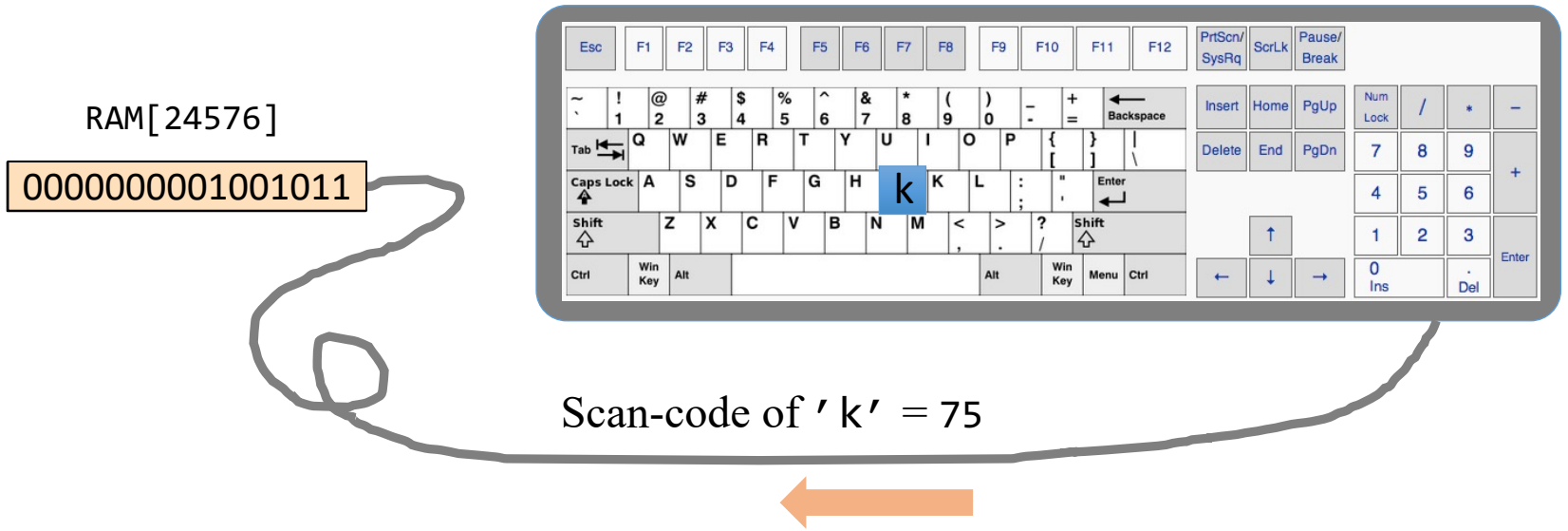


Keyboard memory map

When a key is pressed on the keyboard,
the keyboard register is set to the key's *character code*;

When no key is pressed, the keyboard register is set to `0`.

Keyboard memory map

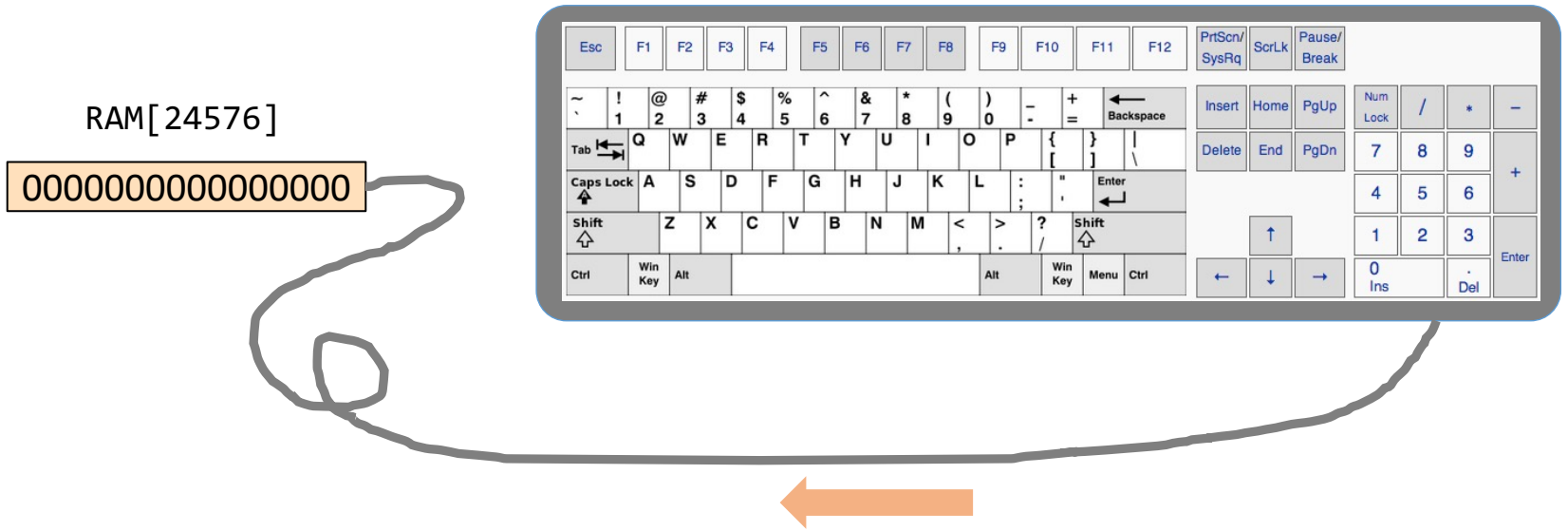


Keyboard memory map

When a key is pressed on the keyboard,
the keyboard register is set to the key's *character code*;

When no key is pressed, the keyboard register is set to `0`.

Keyboard memory map

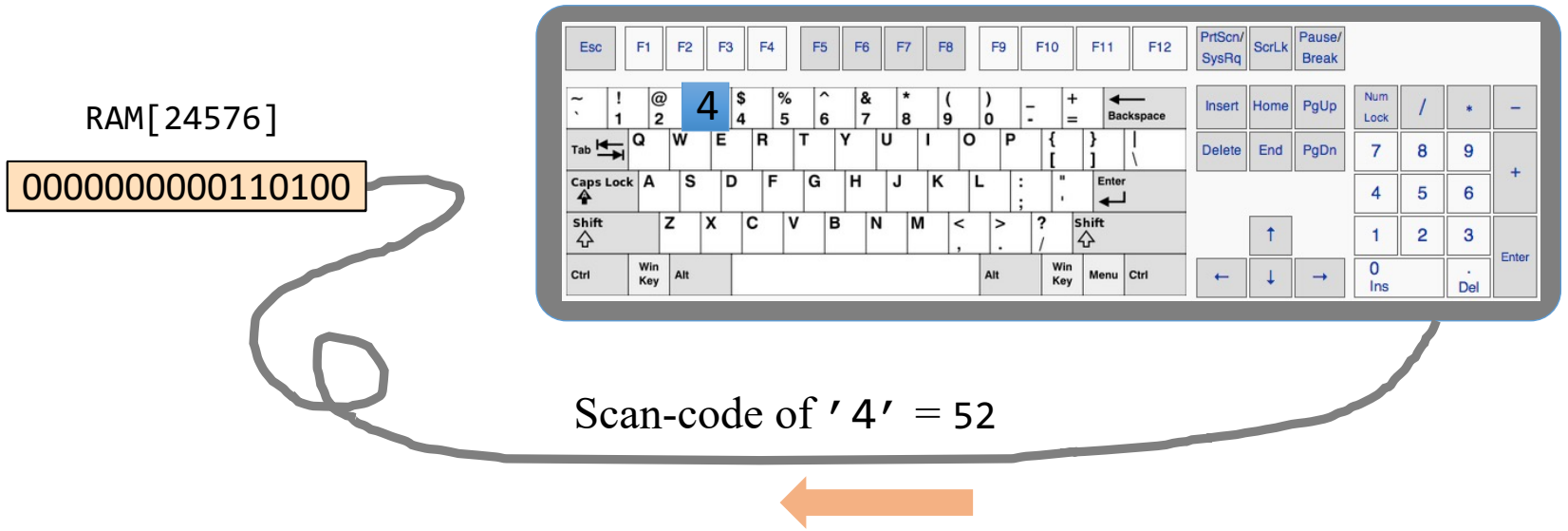


Keyboard memory map

When a key is pressed on the keyboard,
the keyboard register is set to the key's *character code*;

When no key is pressed, the keyboard register is set to `0`.

Keyboard memory map

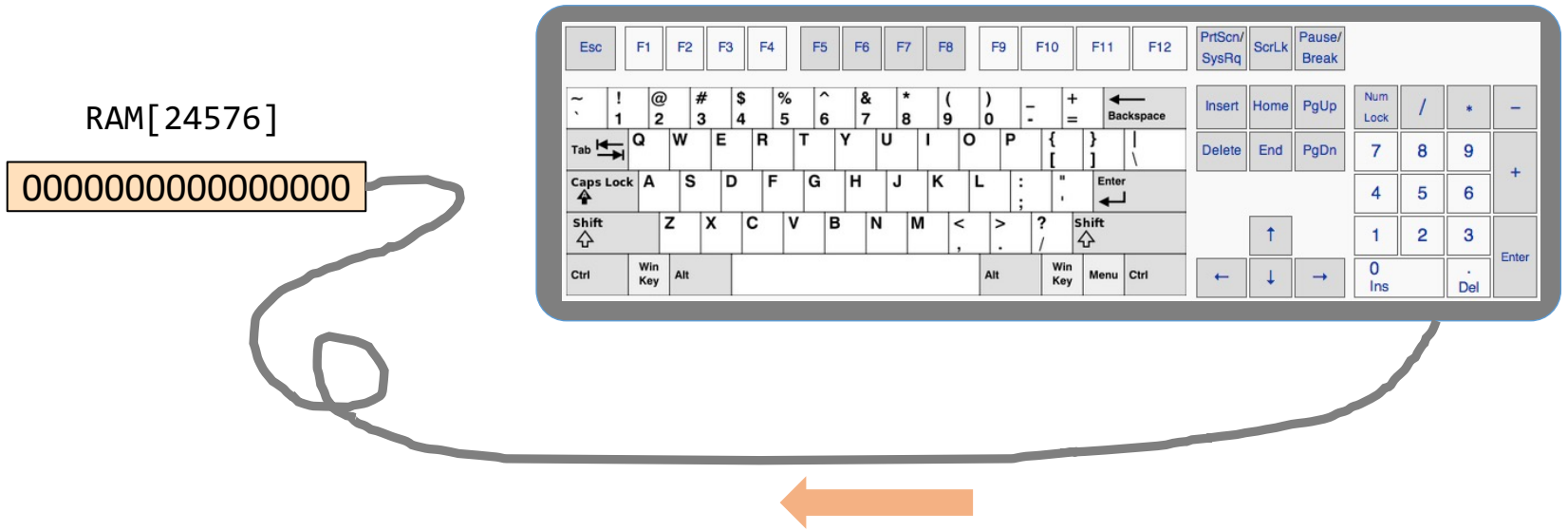


Keyboard memory map

When a key is pressed on the keyboard,
the keyboard register is set to the key's *character code*;

When no key is pressed, the keyboard register is set to `0`.

Keyboard memory map

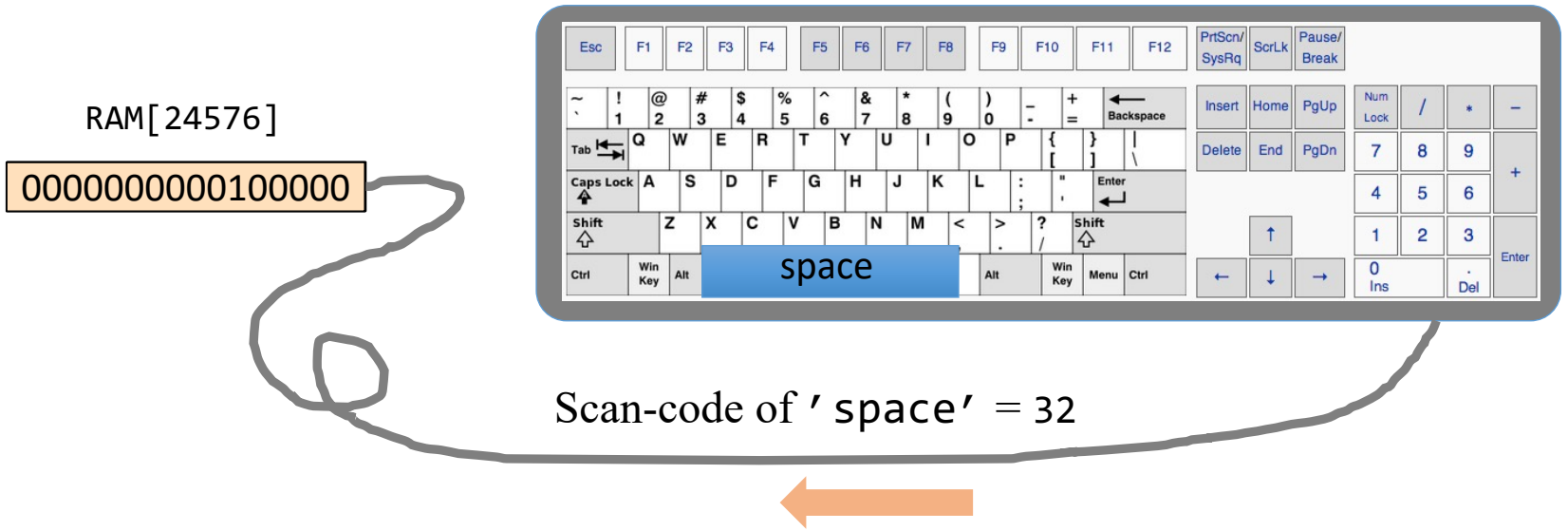


Keyboard memory map

When a key is pressed on the keyboard,
the keyboard register is set to the key's *character code*;

When no key is pressed, the keyboard register is set to `0`.

Keyboard memory map

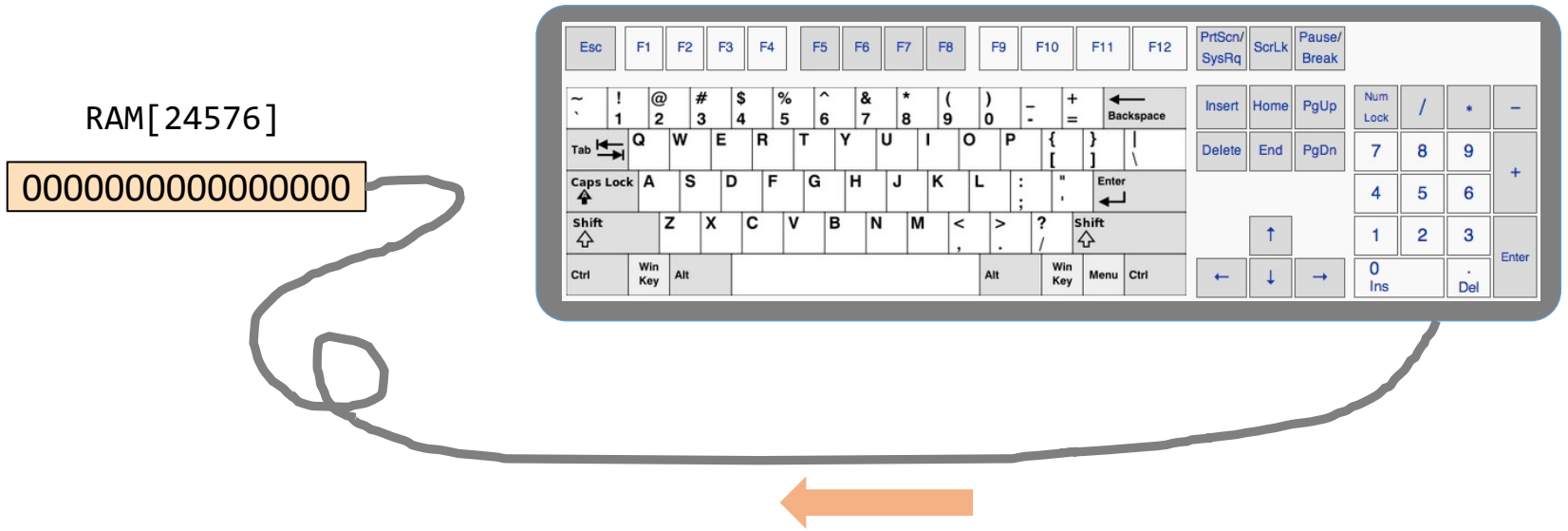


Keyboard memory map

When a key is pressed on the keyboard,
the keyboard register is set to the key's *character code*;

When no key is pressed, the keyboard register is set to `0`.

Keyboard memory map

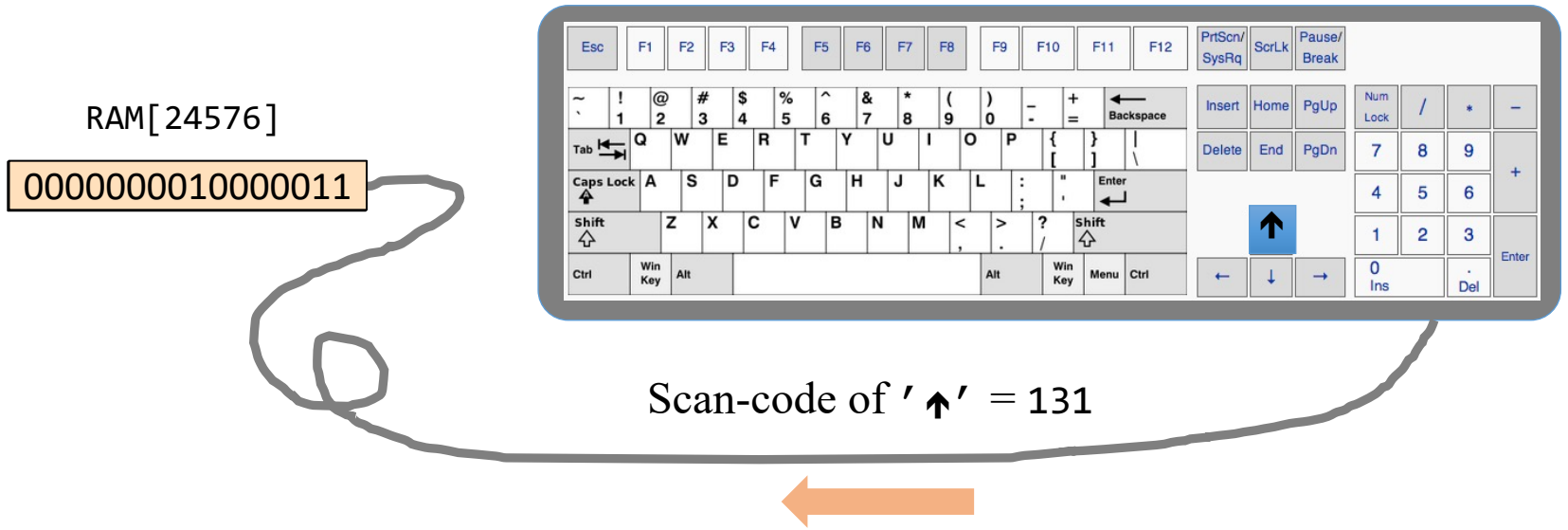


Keyboard memory map

When a key is pressed on the keyboard,
the keyboard register is set to the key's *character code*;

When no key is pressed, the keyboard register is set to `0`.

Keyboard memory map

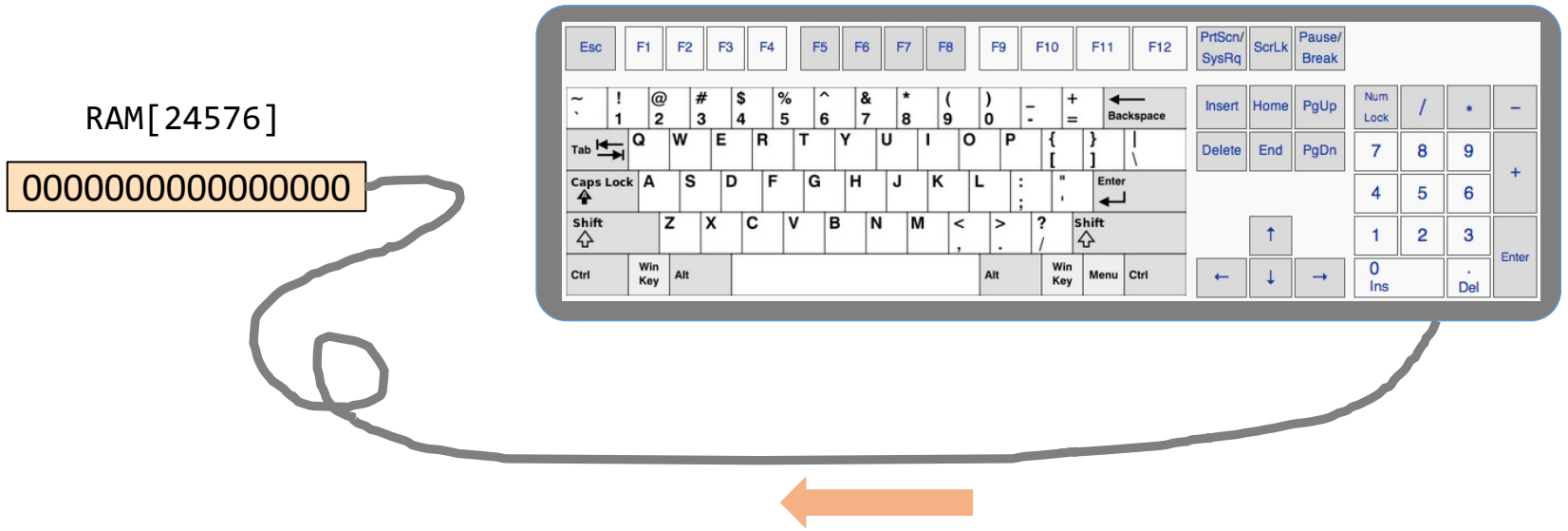


Keyboard memory map

When a key is pressed on the keyboard,
the keyboard register is set to the key's *character code*;

When no key is pressed, the keyboard register is set to `0`.

Keyboard memory map



Keyboard memory map

When a key is pressed on the keyboard,
the keyboard register is set to the key's *character code*;

When no key is pressed, the keyboard register is set to `0`.

Keyboard class API

```
class Keyboard {

    /* Returns the character code of the currently pressed key,
       or 0 if no key is currently pressed.*/
    function char keyPressed() {}

    /* Reads the next character from the keyboard:
       waits until a key is pressed and then released, then echoes
       the key to the screen, and returns the code of the pressed key.*/
    function char readChar() {}

    /* Prints the message on the screen, reads the next line (until a newLine
       character) from the keyboard, and returns its value. */
    function String readLine(String message) {}

    /* Prints the message on the screen, reads the next line
       (until a newline character) from the keyboard, and returns its
       integer value (until the first non numeric character). */
    function int readInt(String message) {}
}
```

keyPressed / readChar / readLine

listens to the keyboard

```
keyPressed():
  if a keyboard key is pressed:
      return the key's character code
  else
      return 0
```

keyPressed / readChar / readLine

listens to the keyboard

```
keyPressed():
  if a keyboard key is pressed:
      return the key's character code
  else
      return 0
```

gets a character

```
/* Waits until a key is pressed and released;
echoes the key on the screen, advances the cursor,
and returns the key's character. */
readChar():
```

keyPressed / readChar / readLine

listens to the keyboard

```
keyPressed():
  if a keyboard key is pressed:
      return the key's character code
  else
      return 0
```

gets a character

```
/* Waits until a key is pressed and released;
echoes the key on the screen, advances the cursor,
and returns the key's character. */
readChar():
  display the cursor
  // waits until a key is pressed
  while (keyPressed() == 0):
      do nothing
  c = code of the currently pressed key
  // waits until the key is released
  while (keyPressed() ≠ 0):
      do nothing
  display c at the current cursor location
  advance the cursor
  return c
```

keyPressed / readChar / readLine

listens to the keyboard

```
keyPressed():
  if a keyboard key is pressed:
      return the key's character code
  else
      return 0
```

gets a character

```
/* Waits until a key is pressed and released;
echoes the key on the screen, advances the cursor,
and returns the key's character. */
readChar():
  display the cursor
  // waits until a key is pressed
  while (keyPressed() == 0):
      do nothing
  c = code of the currently pressed key
  // waits until the key is released
  while (keyPressed() ≠ 0):
      do nothing
  display c at the current cursor location
  advance the cursor
  return c
```

gets a line

```
/* Reads a line (until a newLine character) from the
keyboard, and returns its value, as a string. */
readLine():
```

keyPressed / readChar / readLine

listens to the keyboard

```
keyPressed():
  if a keyboard key is pressed:
      return the key's character code
  else
      return 0
```

gets a character

```
/* Waits until a key is pressed and released;
echoes the key on the screen, advances the cursor,
and returns the key's character. */
readChar():
  display the cursor
  // waits until a key is pressed
  while (keyPressed() == 0):
      do nothing
  c = code of the currently pressed key
  // waits until the key is released
  while (keyPressed() ≠ 0):
      do nothing
  display c at the current cursor location
  advance the cursor
  return c
```

gets a line

```
/* Reads a line (until a newLine character) from the
keyboard, and returns its value, as a string. */
readLine():
  str = empty string
  repeat
      c = readChar()
      if (c == newLine):
          display newLine
          return str
      else if (c = backSpace):
              remove the last character from str
              do Output.backspace()
          else
              str = str.append(c)
  return str
```

Implementation

```
class Keyboard {

    /* Returns the character code of the currently pressed key,
       or 0 if no key is currently pressed.*/
    function char keyPressed() {}

    /* Reads the next character from the keyboard:
       waits until a key is pressed and then released, then echoes
       the key to the screen, and returns the code of the pressed key.*/
    function char readChar() {}

    /* Prints the message on the screen, reads the next line (until a
       newline character) from the keyboard, and returns its value. */
    function String readLine(String message) {}

    /* Prints the message on the screen, reads the next line
       (until a newline character) from the keyboard, and returns its
       integer value (until the first non numeric character). */
    function int readInt(String message) {}
}
```

Implementation notes

`keyPressed:`
Use `Memory.peek`

`readChar:`
Implement the algorithm

`readLine:`
Implement the algorithm

`readInt:`
Read characters (digits) and build the integer value

The Jack OS


class Math {
class Memory {
Class Screen {
Class Out
cla
Class String {
constructor String new(int maxLength)
method void dispose()
method int length()
method char charAt(int j)
method void setCharAt(int j, char c)
method String appendChar(char c)
method void eraseLastChar()
method int intValue()
method void setInt(int j)
function char backSpace()
function char doubleQuote()
function char newLine()
}
}
Realizes the String data type

`String` class API

```
/* Implements the String type. */
class String {

    /* Constructs a new empty string with a maximum length. */
    constructor String new(int maxLength) {}

    /* De-allocates the string and frees its memory space. */
    method void dispose() {}

    /* Returns the length of this string. */
    method int length() {}

    /* Returns the character value at the specified index */
    method char charAt(int j) {}

    /* Sets the j'th character of this string to the given character. */
    method void setCharAt(int j, char c) {}

    /* Appends the given character to the end of this string, and returns the string. */
    method String appendChar(char c) {}

    /* Erases the last character from this string. */
    method void eraseLastChar() {}

    /* Returns the integer value of this string until the first non-numeric character. */
    method int intValue() {}

    /* Sets this String to the representation of the given number. */
    method void setInt(int number) {}

    /* Returns the new line character. */
    function char newLine() {}

    /* Returns the backspace character. */
    function char backSpace() {}

    /* Returns the double quote (") character. */
    function char doubleQuote() {}
}
```

`String` class API

```
/* Implements the String type. */
class String {

    /* Constructs a new empty string with a maximum length. */
    constructor String new(int maxLength) {}

    /* De-allocates the string and frees its memory space. */
    method void dispose() {}

    /* Returns the length of this string. */
    method int length() {}

    /* Returns the character value at the specified index
    method char charAt(int j) {}

    /* Sets the j'th character of this string to the given ch
    method void setCharAt(int j, char c) {}

    /* Appends the given character to the end of this str
    method String appendChar(char c) {}

    /* Erases the last character from this string. */
    method void eraseLastChar() {}

    /* Returns the integer value of this string until the fi
    method int intValue() {}

    /* Sets this String to the representation of the given
    method void setInt(int number) {}

    /* Returns the new line character. */
    function char newLine() {}

    /* Returns the backspace character. */
    function char backSpace() {}

    /* Returns the double quote (") character. */
    function char doubleQuote() {}
}
```

```
// Typical client code (in some Jack class)
...
var String s;
var int x;
...
let s = String.new(6);   // constructs a string with a max
                         // capacity of 6 characters
...
let s = s.appendChar(97);     // s = "a"
let s = s.appendChar(98);     // s = "ab"
let s = s.appendChar(99);     // s = "abc"
let s = s.appendChar(100);    // s = "abcd"
let x = s.length();           // x = 4  (actual length)
...
do s.setInt(314);           // s = "314"
...
let x = 2 * s.intValue();   // x = 628
...
```

int ⟷ string

int ⟷ string algorithms

```
/* Returns the string representation of
   a non-negative integer */
int2String(val):
    lastDigit = val % 10
    c = character representing lastDigit
    if (val < 10)
        return c (as a string)
    else
        return int2String(val / 10).append(c)
```

```
/* Returns the integer value of a string
   of digit characters, assuming that str[0]
   represents the most significant digit. */
string2Int(str):
    val = 0
    for (i = 0 ... str.length) do
        d = integer value of str[i]
        val = val * 10 + d
    return val
```

`String` class implementation

```
/* Implements the String type. */
class String {

    /* Constructs a new empty string with a maximum length. */
    constructor String new(int maxLength) {}

    /* De-allocates the string and frees its memory space. */
    method void dispose() {}

    /* Returns the length of this string. */
    method int length() {}

    /* Returns the character value at the specified index */
    method char charAt(int j) {}

    /* Sets the j'th character of this string to the given character. */
    method void setCharAt(int j, char c) {}

    /* Appends the given character to the end of this string, and returns the string. */
    method String appendChar(char c) {}

    /* Erases the last character from this string. */
    method void eraseLastChar() {}

    /* Returns the integer value of this string until the first non-numeric character. */
    method int intValue() {}

    /* Sets this String to the representation of the given number. */
    method void setInt(int number) {}

    /* Returns the new line character. */
    function char newLine() {}

    /* Returns the backspace character. */
    function char backSpace() {}

    /* Returns the double quote (") character. */
    function char doubleQuote() {}
}
```

Implementation notes

`length`, `charAt`, `setCharAt`, `appendChar`, `eraseLastChar`:

Use array manipulations

`intValue`:

Realize the *string2int* algorithm

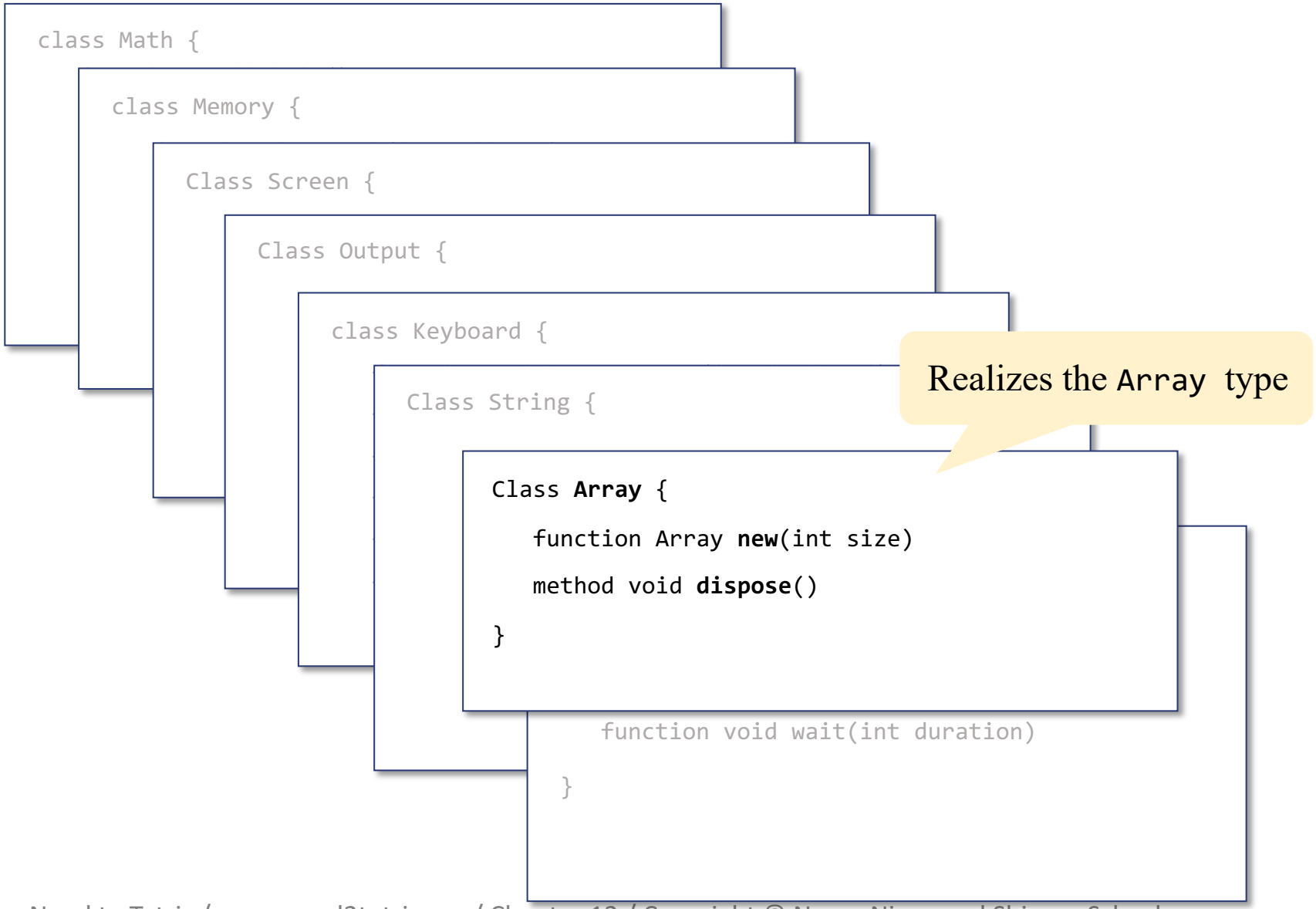
`setInt`:

Realize the *int2String* algorithm

`newLine`, `backSpace`, `doubleQuote`:

Return the `int` values `128`, `129`, `34`.

The Jack OS: `Array`



Array

OS `Array` class API

```
Class Array {

   function Array new(int size)

   method void dispose()

}
```

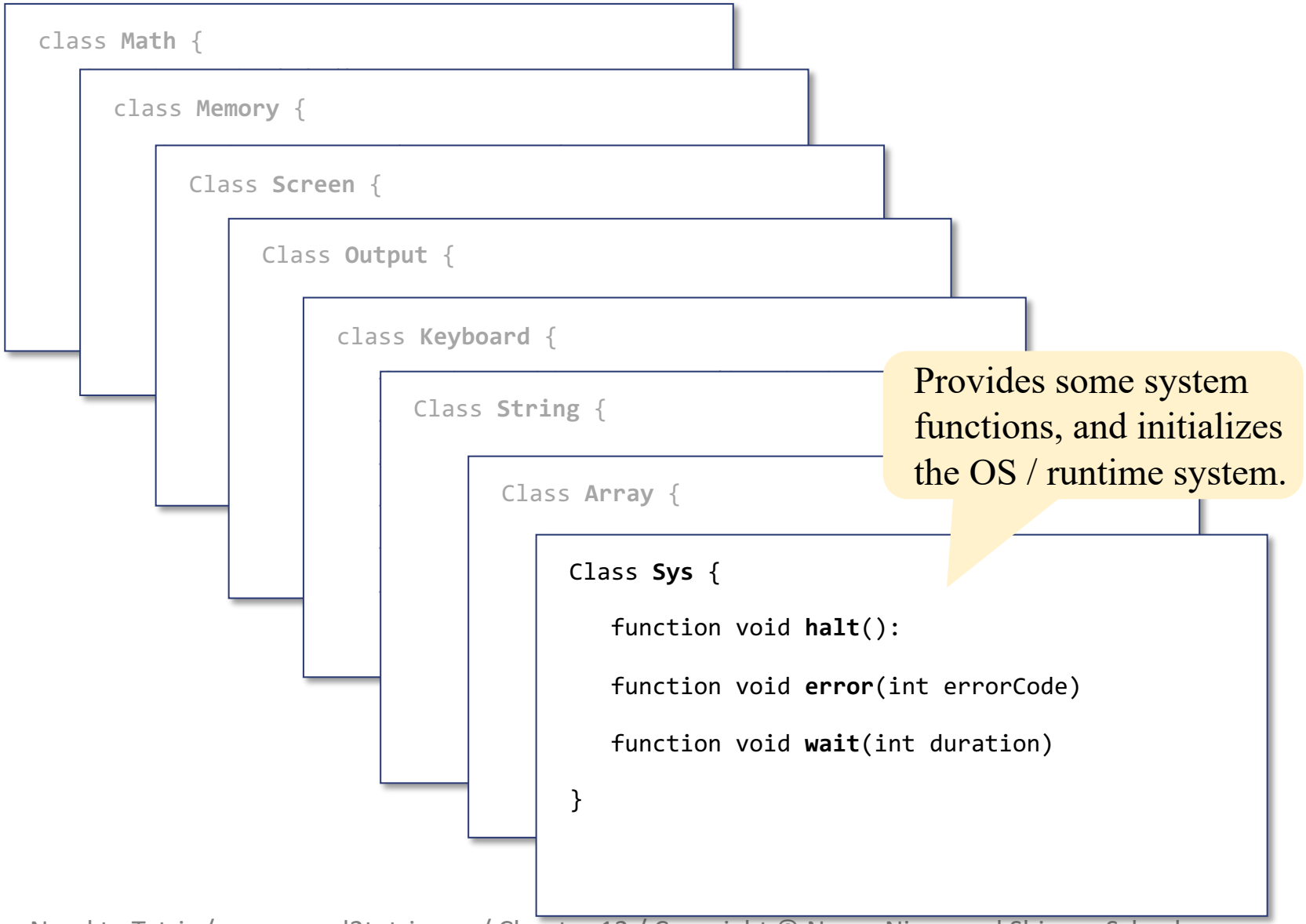
```
// Typical client code (in some Jack class)
...
var Array a, b;
...
let a = Array.new(3);
...
do a.dispose();
...
```

Implementation notes

`new:` Must be Implemented as a *function*, not as a constructor;

`dispose:` Use `Memory.deAlloc`

The Jack OS


class Math {
class Memory {
Class Screen {
Class Output {
class Keyboard {
Class String {
Class Array {
Class Sys {
function void halt():
function void error(int errorCode)
function void wait(int duration)
}
Provides some system functions, and initializes the OS / runtime system.

Sys class

The key function of the `Sys` class is *bootstrapping*

Bootstrapping (booting): the process of loading the basic software into the computer's memory after power-on or general reset, especially the operating system, which will then take care of loading other software as needed (Wikipedia)

Jack contract: Assume that program execution starts with the function `Main.main()`

Hardware contract

When the computer is reset, have it execute the instruction in `ROM[0]`

VM contract

Store the following code (in machine language) beginning at `ROM[0]`:

```
sp = 256
call Sys.init
```

OS contract

Have `Sys.init` initialize the OS, and call `Main.main()`.

Sys class

```
/* A library of basic system services  */
class Sys {

    // (hidden from the Sys API) Initializes the OS classes, and calls ”main”
    function void init() {
      do Memory.init();
      do Math.init();
      do Screen.init();
      do Output.init();
      do Keyboard.init();
      do Main.main();
      do Sys.halt();
    }

    /* Halts execution */
    function void halt() {}

    /* Waits approximately duration milliseconds, and returns */
    function void wait(int duration) {}

    /* Prints the error code in the format "ERR<errorCode>", and halts. */
    function void error(int errorCode) {}
}
```

Implementation notes

`halt`: Use an infinite loop;

`wait`: Use a loop, hardware-specific

The Jack OS

```
class Math {
```

```
class Memory {
```

```
Class Screen {
```

```
Class Output {
```

```
class Keyboard {
```

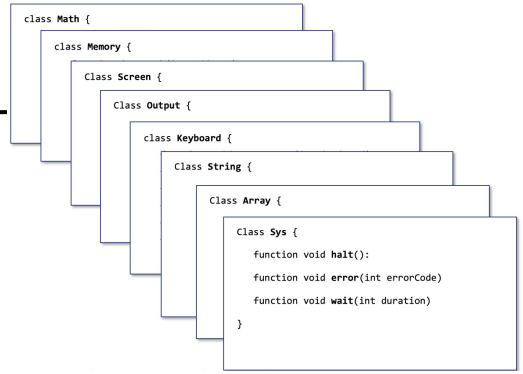
```
Class String {
```

```
Class Array {
```

```
Class Sys {
   function void halt():
   function void error(int errorCode)
   function void wait(int duration)
}
```

Project 12:
Build the OS

The Jack OS



OS abstraction: Specified by the OS classes APIs

OS implementations

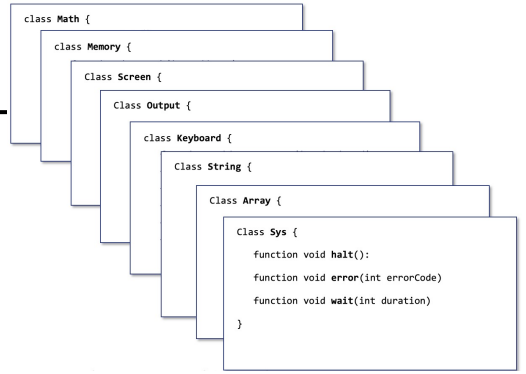
VM emulator: The emulator's software includes a "built-in" implementation of the OS API

`nand2tetris/tools/os`: A set of compiled Jack classes that implement the OS API

`Math.vm`, `Memory.vm`, `Screen.vm`, `Output.vm`, `Keyboard.vm`, `String.vm`, `Array.vm`, `Sys.vm`

Project 12: Write your own OS implementation, using Jack.

Reverse engineering



Suppose you wish to implement an existing OS, consisting of *n* modules, with high inter-dependency.

Suppose that you have access to *executable versions* of the *n* modules

Strategy

Implement each OS module separately, using the remaining *n* –1 executable modules to support it

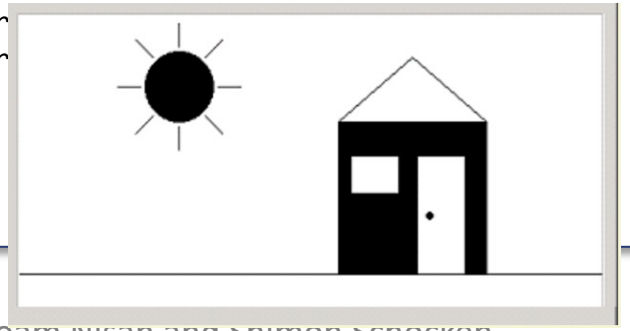
(That's how many versions of Unix were developed).

Example: Implementing the OS class `Screen`

```
/* A library of OS functions for displaying graphics on the screen */
class Screen {

    // Initializes the Screen
    function void init() {}

    /* Erases the entire screen */
    function void clearScreen() {}

    /* Sets the current color, to be used for all su
    Black is represented by true, white by false *
    function void setColor(boolean b) {

    /* Draws the (x,y) pixel, using the current co
    function void drawPixel(int x, int y

  /* Draws a line from pixel (x1,y1) to pixel (x2
    function void drawLine(int x1, int y

    /* Draws a filled rectangle whose top left co
    and bottom right corner is (x2,y2), using the
    function void drawRectangle(int x1,

    /* Draws a filled circle of radius r<=181 aro
    function void drawCircle(int x, int
}
```

The implementation of the `Screen` methods can call any method in any OS class, as needed

```
/* Test program for the OS class Screen (supplied). */
class Main {
    /* Draws an image using lines and circles */
    function void main() {
        do Screen.drawLine(0,220,511,220);          // base line
        do Screen.drawRectangle(280,90,410,220);    // house

        do Screen.setColor(false);
        do Screen.drawRectangle(350,120,390,219); // door
        do Screen.drawRectangle(292,120,332,150); // window

        do Screen.setColor(true);
        do Screen.drawCircle(360,170,3);            // door handle
        do Screen.drawLine(280,90,345,35);          // roof
        do Screen.drawLine(345,35,410,90);          // roof

        do Screen.dr
        do Screen.dr
        ...
        return;
    }
}
```



Example: Implementing the OS class `Screen`

Files: `Screen.jack` (your implementation), and `Main.jack` (test file), in the same folder

If using the VM emulator

1. Compile the folder
2. Load the folder into the VM emulator, and execute the code

   (every call to a `Screen.`*function* will be serviced by your `Screen` implementation; every call to any other OS function will be serviced by the built-in implementations of the remaining 7 OS classes)

If using the CPU emulator

1. Add to the folder the remaining 7 `nand2tetris/tools/os/*.vm` files
2. Compile the folder
3. Use a VM translator to translate the folder, resulting with the file *folderName*`.asm`
4. Load *folderName*`.asm` into the supplied CPU emulator, and execute the code.

Implications

When implementing a `Screen` method, you can call any method from any other OS class, as needed.

Project 12

Step-wise development

`Screen`, `Output`, `String`, `Keyboard`, `Sys`: develop/test separately,
using the supplied `Main.jack` test programs

`Memory`, `Array`, `Math`: develop/test separately,
using the supplied `.jack`, `.tst` and `.cmp` files

(Can be developed in any order)

Final test

1. Make a copy of the given folder `nand2tetris/projects/11/Pong`
2. Copy the 8 compiled `.vm` files of your OS into this folder
3. Load and execute this folder in the VM emulator.

Project 12

Perspective

Missing elements in our OS

- Multi-threading
- Multi-processing
- File system
- Shell / windowing
- Security
- Communications
- Many more missing services.

Perspective

Our OS...

- Closes significant gaps between the underlying hardware and application programs
- Hides many gory low-level details from the application programmer
- Performs its job elegantly and efficiently
- Provides a feeling of what it takes to design and implement OS software.

Perspective

OS code is typically considered *privileged*:

- Accessing OS services requires a permission mechanism that is more elaborate than a simple function call
- OS functions execute in a special protected mode
- OS functions receive more hardware resources
- (In the Hack system, user-level code and OS code are treated the same way).

Perspective

Efficiency: Mathematical operations

- In most computers, core mathematical operations are implemented in hardware
- The hardware and software implementations of these operations are often based on the same algorithmic ideas
- The running time of the multiplication and division algorithms that we presented is $O(N)$ addition operations, N being the number of bits
- Since our addition implementation also requires $O(N)$ operations, the overall running time of our multiplication and division algorithms is $O(N^2)$
- There exist multiplication and division algorithms whose running time is asymptotically much faster than $O(N^2)$
- However, these algorithms are useful only when the number of bits that we have to process is very large.

Perspective

Efficiency: Graphical operations

- The line drawing algorithms that we presented are efficient
- In most systems these graphics primitives are implemented by a combination of software and special graphics-acceleration hardware
- Today, most computers are equipped with a *Graphical Processing Unit*
- The GPU off-loads the CPU from handling high-performance graphics tasks, like drawing 3D images and rendering smooth surfaces
- In the Hack-Jack platform, there is no such separation of responsibilities between the CPU and other dedicated processors.

Chapter 12

Operating System

These slides support chapter 12 of the book
The Elements of Computing Systems
By Noam Nisan and Shimon Schocken
MIT Press