PERCOLATION

Class Meeting #2

COS 226 — Spring 2018

Based on slides by

Jérémie Lumbroso

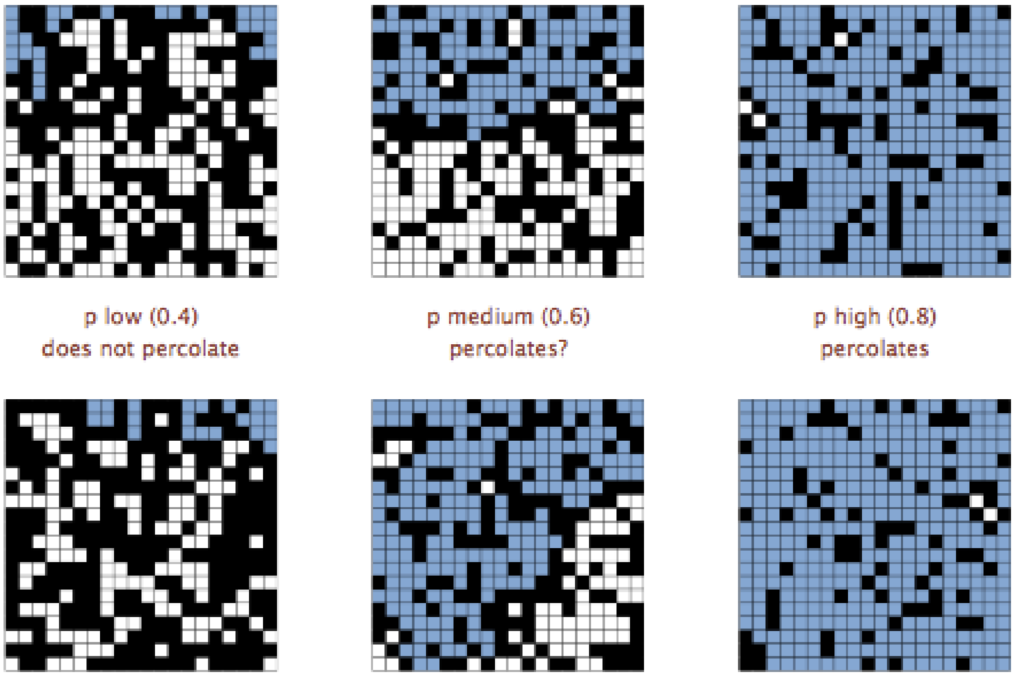
— Motivation
— Problem description
— API
— Backwash
— Empirical Analysis
— Memory Analysis

What does Percolation model?

Likelihood of percolation

Depends on site vacancy probability p.

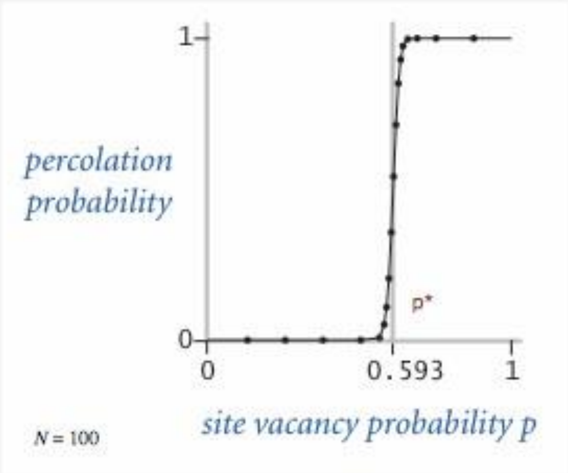


Percolation phase transition

When N is large, theory guarantees a sharp threshold p^*.

- $p > p^*$: almost certainly percolates.
- $p < p^*$: almost certainly does not percolate.

Q. What is the value of p^* ?

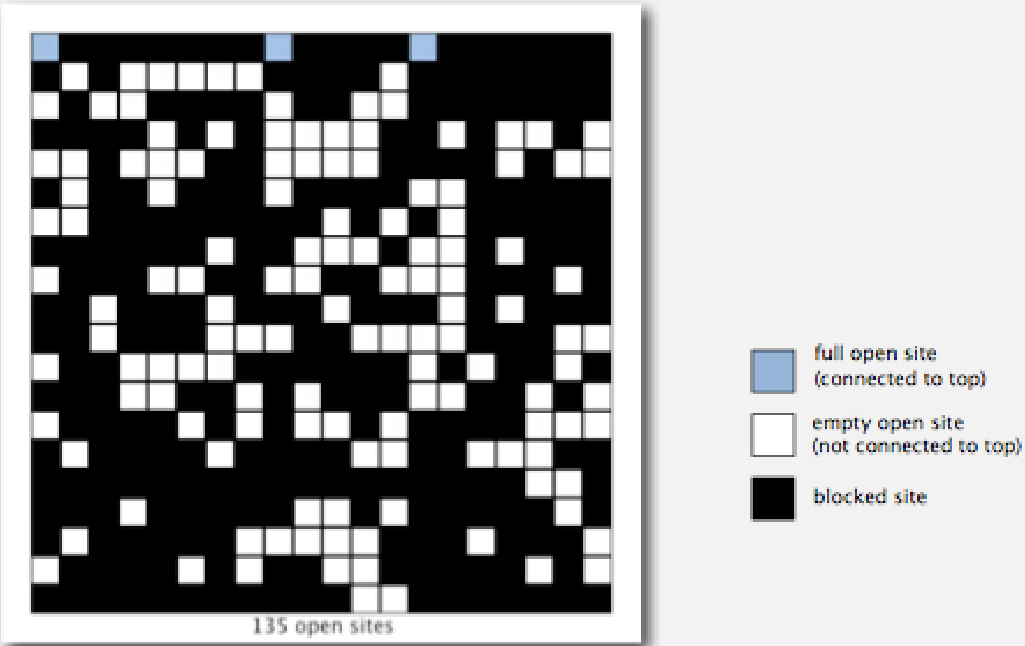


Other examples:

- Water freezing
- Ferromagnetic effects

Monte Carlo simulation

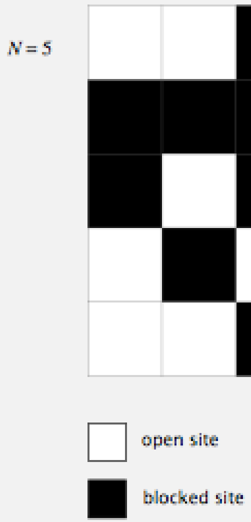
- Initialize N-by-N whole grid to be blocked.
- Declare random sites open until top connected to bottom.
- Vacancy percentage estimates p^*.



$N = 20$

Dynamic connectivity solution to estimate percolation threshold

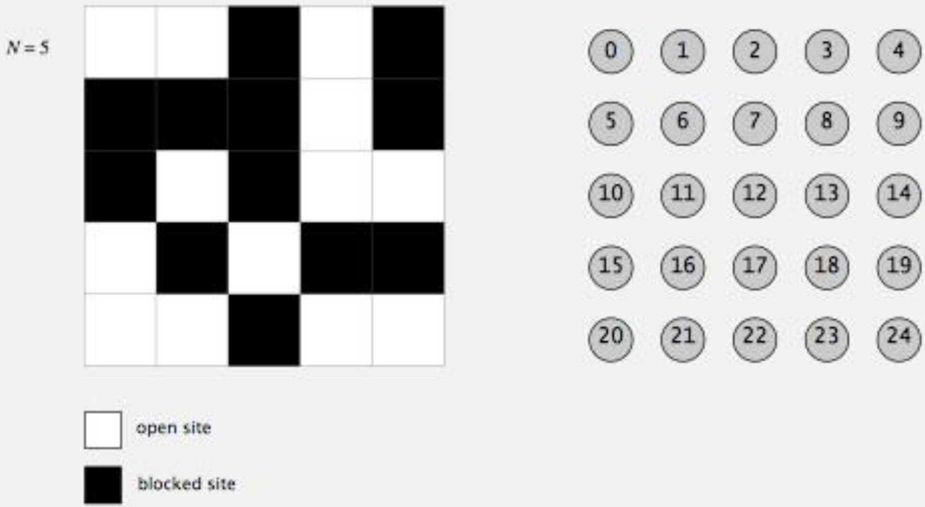
Q. How to check whether an N-by-N system percolates?



Dynamic connectivity solution to estimate percolation threshold

Q. How to check whether an N-by-N system percolates?

- Create an object for each site and name them 0 to $N^2 - 1$.



Create private “helper” funtion

```
private int getIntFromCoord(int row, int col) {
    return N * row + col;
}
```

Or perhaps since this function will be used a lot, should it have a **shorter name**?

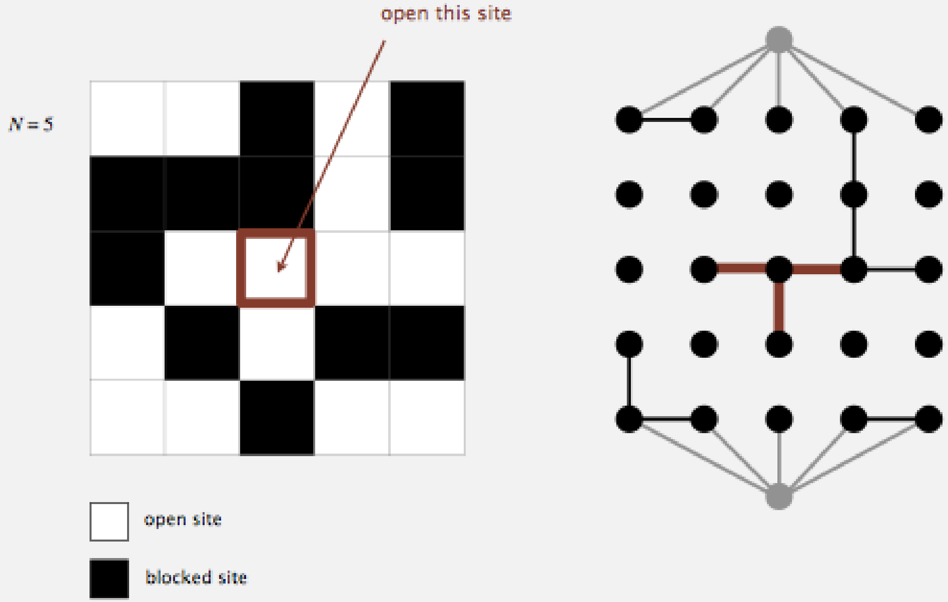
For ex.: `site` or `location` or `cell` or `grid`, etc., …

Dynamic connectivity solution to estimate percolation threshold

Q. How to model opening a new site?

A. Mark new site as open; connect it to all of its adjacent open sites.

up to 4 calls to union()

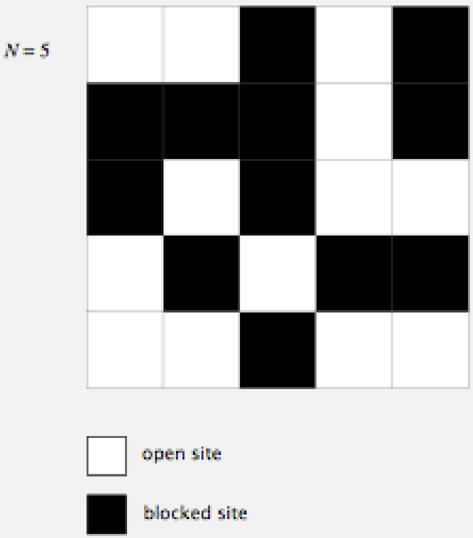


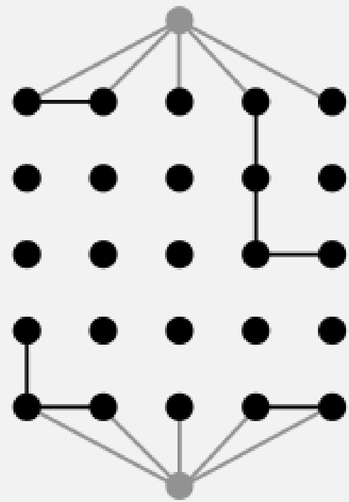
Dynamic connectivity solution to estimate percolation threshold

Clever trick. Introduce 2 virtual sites (and connections to top and bottom).

- Percolates iff virtual top site is connected to virtual bottom site.

efficient algorithm: only 1 call to connected()





```
public class Percolation {
   public Percolation(int N)               // create N-by-N grid, with all sites initially blocked
   public void open(int row, int col)      // open the site (row, col) if it is not open already
   public boolean isOpen(int row, int col) // is the site (row, col) open?
   public boolean isFull(int row, int col) // is the site (row, col) full?
   public int numberOfOpenSites()          // number of open sites
   public boolean percolates()             // does the system percolate?
   public static void main(String[] args)  // unit testing (required)
}

public class PercolationStats {
   public PercolationStats(int N, int T)  // perform T independent experiments on an N-by-N grid
   public double mean()                   // sample mean of percolation threshold
   public double stddev()                 // sample standard deviation of percolation threshold
   public double confidenceLow()          // low  endpoint of 95% confidence interval
   public double confidenceHigh()         // high endpoint of 95% confidence interval
}
```

what **you** must do

both are APIs

what is provided

public class UF

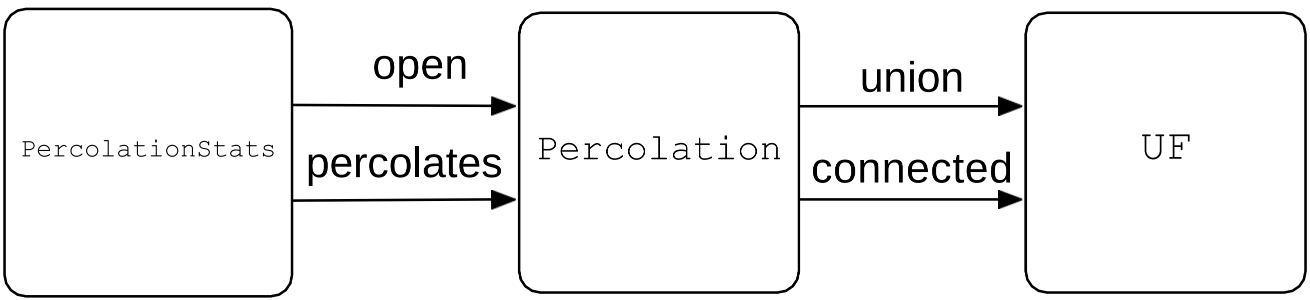
UF(int N)	*initialize union-find data structure with N objects (0 to N – 1)*
void union(int p,int q)	*add connection between p and q*
boolean connected(int p,int q)	*are p and q in the same component?*
int find(int p)	*component identifier for p (0 to N – 1)*
int count()	*number of components*

Why an API?

API = Application Programming Interface

—a contract between a programmers

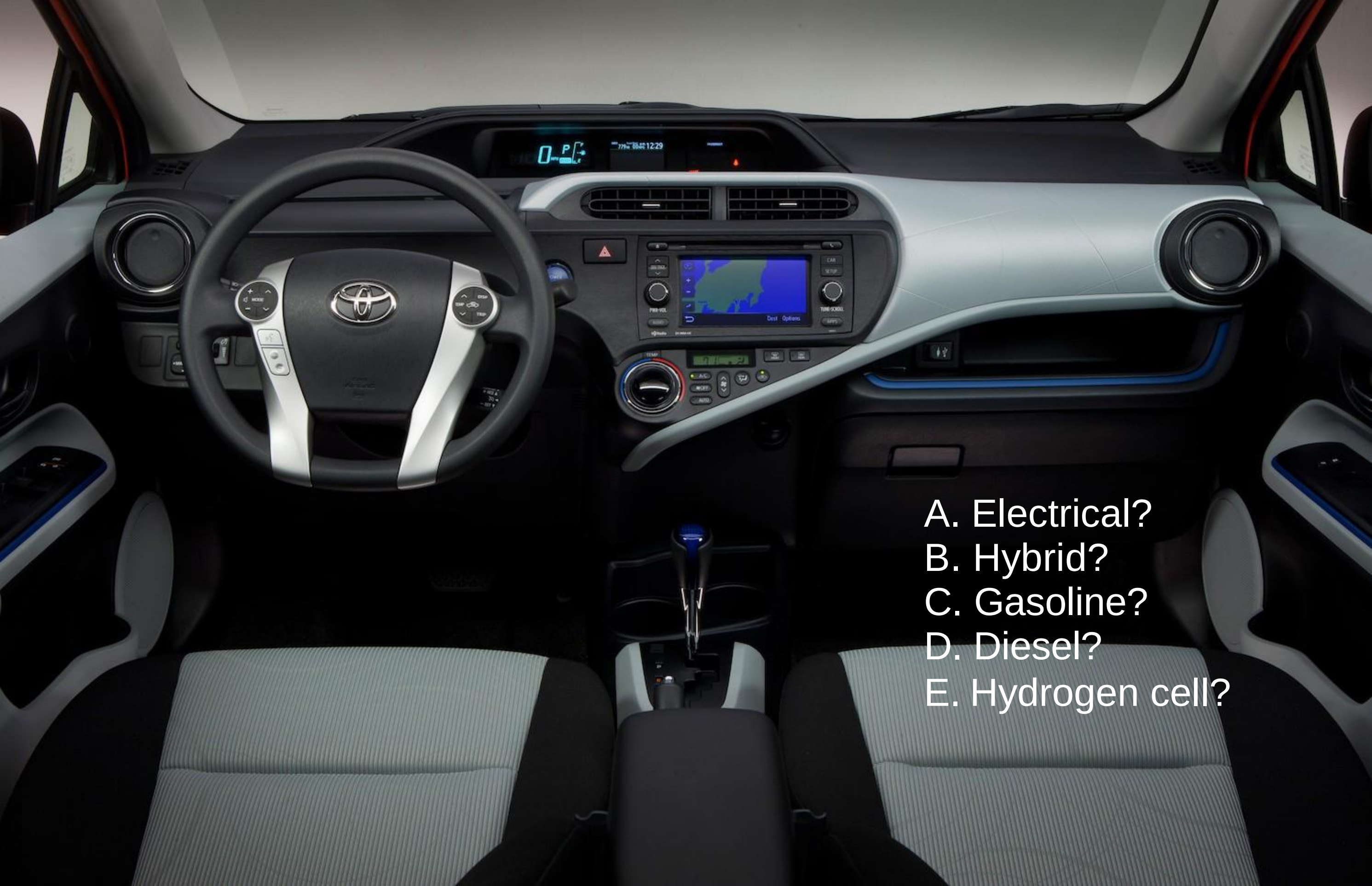
—be able to know about the functionality without details from the implementation



Each of these modules could be programmed by anybody / implemented anyway

Example 1: Car

```
public class Car {
  void turnLeft()
  void turnRight()
  void shift(int gear)
  void break()
}
```


A. Electrical?
B. Hybrid?
C. Gasoline?
D. Diesel?
E. Hydrogen cell?

Example 1: Car

```
public class Car {
   void turnLeft()
   void turnRight()
   void shift(int gear)
   void break()
}
```

Example 2: Electrical Outlets

original API

API with added public members

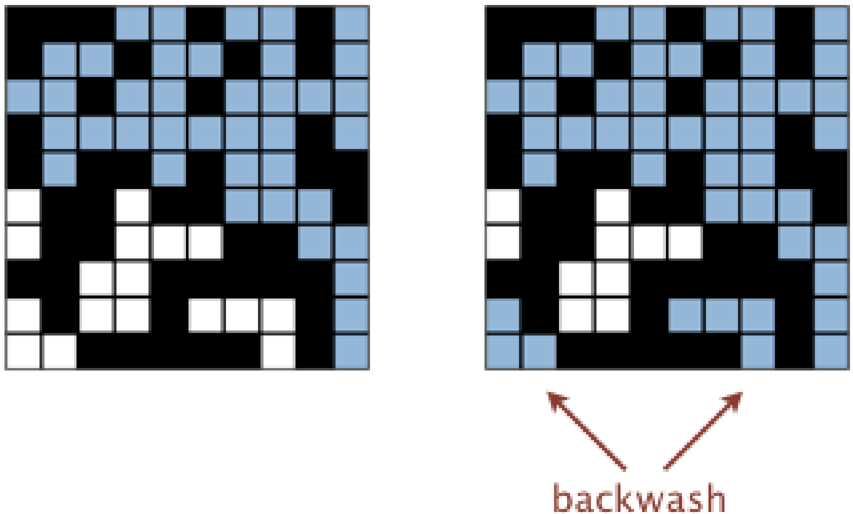
is incompatible with rest of the clients

Why is it so important to implement the prescribed API? Writing to an API is an important skill to master because it is an essential component of modular programming, whether you are developing software by yourself or as part of a group. When you develop a module that properly implements an API, anyone using that module (including yourself, perhaps at some later time) does not need to revisit the details of the code for that module when using it. This approach greatly simplifies writing large programs, developing software as part of a group, or developing software for use by others.

Most important, when you properly implement an API, others can write software to use your module or to test it. We do this regularly when grading your programs. For example, your `PercolationStats` client should work with our `Percolation` data type and vice versa. If you add an extra public method to `Percolation` and call them from `PercolationStats`, then your client won't work with our `Percolation` data type. Conversely, our `PercolationStats` client may not work with your `Percolation` data type if you remove a public method.

Backwash problem

```
% java PercolationVisualizer input10.txt
```



Empirical Analysis

THEORY + PRACTICE

Power Law Running Times

— Typically most running times that are empirically measure are **power laws**

exponent

$$c N^{a}$$

constant factor

parameter (size of the instance)

— Usually when other running times are involved such as N.log N, N.α(n), exp(N), it will be because of a known sub-algorithm

Doubling Hypothesis (1)

Assuming the running time is of the form:

$$t(N) := c \cdot N^a$$

then, to find the exponent a:

$$\frac{t(2N)}{t(N)} = \frac{c \cdot (2N)^a}{c \cdot N^a} = \frac{\cancel{c} \cdot 2^a \cancel{N^a}}{\cancel{c} \cdot \cancel{N^a}} = 2^a$$

Doubling Hypothesis (2)

timing when size of input is 2N (doubled)

exponent!!! *(what we want to find)*

$$\log_2\left(\frac{t(2N)}{t(N)}\right) = a$$

binary logarithm

timing when size of input is N

<u>Recipe:</u>

— timing in N and 2N
— take log base 2 of ratio
— repeat for several points

Doubling Hypothesis (3)

<u>Tips:</u> 1) pick largest points; 2) repeat couple times

N	100	200	400	800	1600	3200	6400
time	0.538473	0.0932774	0.163298	0.744645	2.5858	18.5561	141.455

weird value is an artifact

$$\log_2\left(\frac{t(200)}{t(100)}\right) \approx -2.52927$$

first try with N too small (noise)

$$\log_2\left(\frac{t(6400)}{t(3200)}\right) \approx 2.93038$$

then we get a more realistic value

$$\log_2\left(\frac{t(3200)}{t(1600)}\right) \approx 2.84321$$

and we can try to confirm
(if not, try to get larger point, such as N=12800)

What to do…

— … to determine the constant?

Once exponent(s) is found, obtain by simple division.

— you have **two** variables (such as N and T)

Treat each separately (by making one variable vary, while the other remains constant).

Stopwatch.java

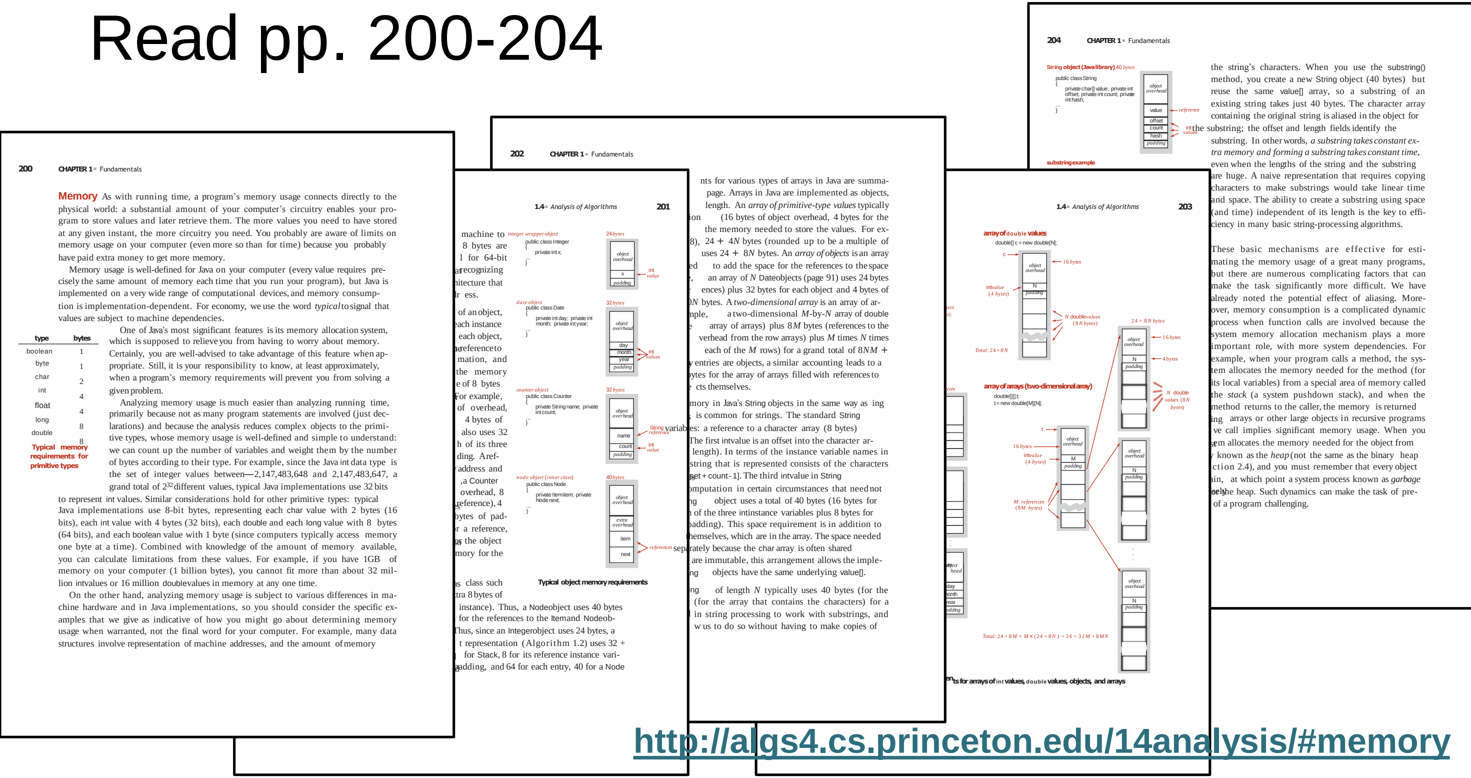
```
sw = new Stopwatch();          // timer starts

ps = new PercolationStats(N, T); // operation we
                                 // want to measure

timing = sw.elapsedTime;     // time in seconds since
                             // the Stopwatch was
                             // created
```

if single observations too fast to measure,
measure several operations at a time and average

Memory Analysis

Memory (1)

Read pp. 200-204



200 CHAPTER 1 ■ Fundamentals

Memory As with running time, a program's memory usage connects directly to the physical world: a substantial amount of your computer's circuitry enables your program to store values and later retrieve them. The more values you need to have stored at any given instant, the more circuitry you need. You probably are aware of limits on memory usage on your computer (even more so than for time) because you probably have paid extra money to get more memory.

Memory usage is well-defined for Java on your computer (every value requires precisely the same amount of memory each time that you run your program), but Java is implemented on a very wide range of computational devices, and memory consumption is implementation-dependent. For economy, we use the word *typical* to signal that values are subject to machine dependencies.

type	bytes
boolean	1
byte	1
char	2
int	4
float	4
long	8
double	8

Typical memory requirements for primitive types

One of Java's most significant features is its memory allocation system, which is supposed to relieve you from having to worry about memory. Certainly, you are well-advised to take advantage of this feature when appropriate. Still, it is your responsibility to know, at least approximately, when a program's memory requirements will prevent you from solving a given problem.

Analyzing memory usage is much easier than analyzing running time, primarily because not as many program statements are involved (just declarations) and because the analysis reduces complex objects to the primitive types, whose memory usage is well-defined and simple to understand: we can count up the number of variables and weight them by the number of bytes according to their type. For example, since the Java int data type is the set of integer values between −2,147,483,648 and 2,147,483,647, a grand total of 2^{32} different values, typical Java implementations use 32 bits to represent int values. Similar considerations hold for other primitive types: typical Java implementations use 8-bit bytes, representing each char value with 2 bytes (16 bits), each int value with 4 bytes (32 bits), each double and each long value with 8 bytes (64 bits), and each boolean value with 1 byte (since computers typically access memory one byte at a time). Combined with knowledge of the amount of memory available, you can calculate limitations from these values. For example, if you have 1GB of memory on your computer (1 billion bytes), you cannot fit more than about 32 million int values or 16 million double values in memory at any one time.

On the other hand, analyzing memory usage is subject to various differences in machine hardware and in Java implementations, so you should consider the specific examples that we give as indicative of how you might go about determining memory usage when warranted, not the final word for your computer. For example, many data structures involve representation of machine addresses, and the amount of memory

http://algs4.cs.princeton.edu/14analysis/#memory

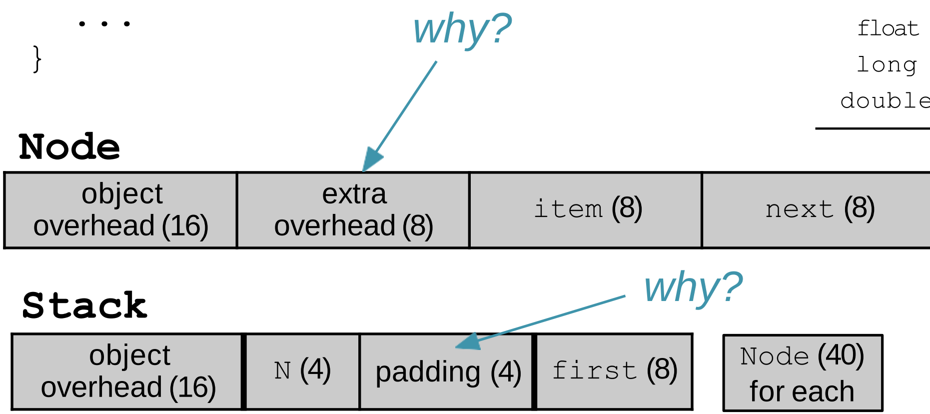
Memory (2)

```
public class Stack {
   private int N;          // size of the stack
   private Node first;     // top of stack

   private class Node {
      private double item;
      private Node next;
   }
   ...
}
```

type	bytes
boolean	1
byte	1
char	2
int	4
float	4
long	8
double	8

type	bytes
boolean[]	$N + 24$
char[]	$2N + 24$
int[]	$4N + 24$
double[]	$8N + 24$

type	bytes
boolean[][]	$\sim MN$
char[][]	$\sim 2MN$
int[][]	$\sim 4MN$
double[][]	$\sim 8MN$



Questions?

More on this in the precept!

Logo by Kathleen Ma (AB ’18)