COS 226, Spring 2013

Algorithms and Data Structures

Josh Hug
Arvind Narayanan

Princeton University

http://www.princeton.edu/~cos226

COS 226 course overview

What is COS 226?

- Intermediate-level survey course.
- Programming and problem solving, with applications.

COS 226 course overview

What is COS 226?

- Intermediate-level survey course.
- Programming and problem solving, with applications.
- Algorithm: method for solving a problem.

COS 226 course overview

What is COS 226?

- Intermediate-level survey course.
- Programming and problem solving, with applications.
- Algorithm: method for solving a problem.
- Data structure: method to store information.

COS 226 course overview

What is COS 226?

- Intermediate-level survey course.
- Programming and problem solving, with applications.
- Algorithm: method for solving a problem.
- Data structure: method to store information.
- Sometimes called: Job Interview 101.

COS 226 course overview

What is COS 226?

- Intermediate-level survey course.
- Programming and problem solving, with applications.
- Algorithm: method for solving a problem.
- Data structure: method to store information.
- Sometimes called: Job Interview 101.

topic	data structures and algorithms
data types	stack, queue, bag, union-find, priority queue
sorting	quicksort, mergesort, heapsort, radix sorts
searching	BST, red-black BST, hash table
graphs	BFS, DFS, Prim, Kruskal, Dijkstra
strings	KMP, regular expressions, tries, data compression
advanced	B-tree, suffix array, maxflow, simplex

Why study algorithms?

Why study algorithms?

Their impact is broad and far-reaching.

Why study algorithms?

Their impact is broad and far-reaching.

Why study algorithms?

Their impact is broad and far-reaching.

Mysterious Algorithm Was 4% of Trading Activity Last Week

CNBC

Published: Monday, 8 Oct 2012 | 4:27 PM ET

By: John Melloy

Text Size − +

Recommend 24 Twitter 2K +1 99 LinkedIn 330 Share

A single mysterious computer program that placed orders — and then subsequently canceled them — made up 4 percent of all quote traffic in the U.S. stock market last week, according to the top tracker of high-frequency trading activity. The motive of the algorithm is still unclear.

The program placed orders in 25-millisecond bursts involving about 500 stocks, according to Nanex, a market data firm. The algorithm never executed a single trade, and it abruptly ended at about 10:30 a.m. ET Friday.

Why study algorithms?

Their impact is broad and far-reaching.

Internet. Web search, packet routing, distributed file sharing, ...

Biology. Human genome project, protein folding, …

Computers. Circuit layout, file system, compilers, …

Computer graphics. Movies, video games, virtual reality, …

Security. Cell phones, e-commerce, voting machines, …

Multimedia. MP3, JPG, HDTV, song recognition, face recognition, …

Social networks. Recommendations, dating, advertisements, …

Physics. N-body simulation, particle collision simulation, …

⋮

Why study algorithms?

Their impact is broad and far-reaching.

Internet. Web search, packet routing, distributed file sharing, ...

Biology. Human genome project, protein folding, …

Computers. Circuit layout, file system, compilers, …

Computer graphics. Movies, video games, virtual reality, …

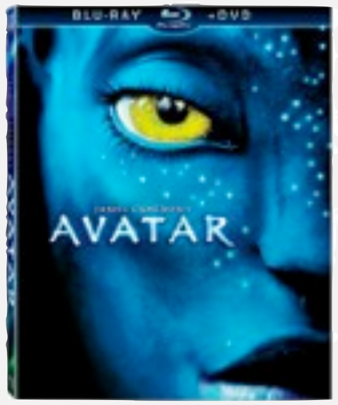
Security. Cell phones, e-commerce, voting machines, …

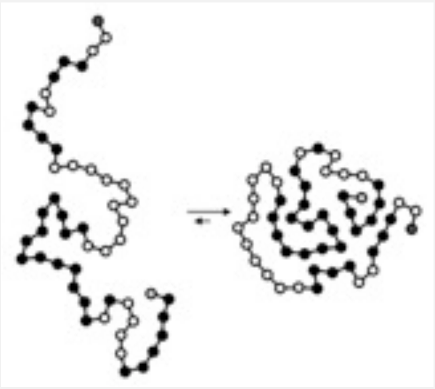
Multimedia. MP3, JPG, HDTV, song recognition, face recognition, …

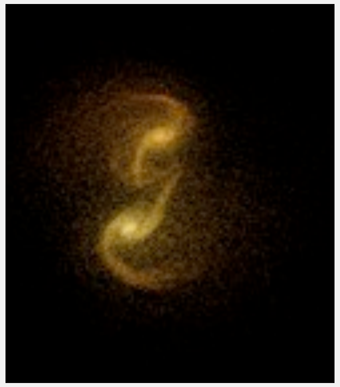
Social networks. Recommendations, dating, advertisements, …

Physics. N-body simulation, particle collision simulation, …

⋮





Why study algorithms?

To become a proficient programmer.

Why study algorithms?

To become a proficient programmer.

"The difference between a bad programmer and a good one is whether [the programmer] considers code or data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships."

— Linus Torvalds (creator of Linux)

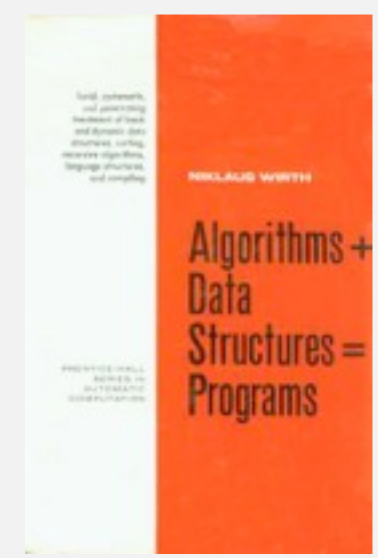
Why study algorithms?

To become a proficient programmer.

"The difference between a bad programmer and a good one is whether [the programmer] considers code or data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships."

— *Linus Torvalds (creator of Linux)*

" Algorithms + Data Structures = Programs. " — *Niklaus Wirth*



Why study algorithms?

For intellectual stimulation.

Frank Nelson Cole

"On the Factorization of Large Numbers"

American Mathematical Society, 1903

$$\mathbf{2^{67}-1 = 193{,}707{,}721 \times 761{,}838{,}257{,}287}$$

Why study algorithms?

They may unlock the secrets of life and of the universe.

Why study algorithms?

They may unlock the secrets of life and of the universe.

Scientists are replacing mathematical models with computational models.

Why study algorithms?

They may unlock the secrets of life and of the universe.

Scientists are replacing mathematical models with computational models.

" Algorithms: a common language for nature, human, and computer. " — *Avi Wigderson*

Why study algorithms?

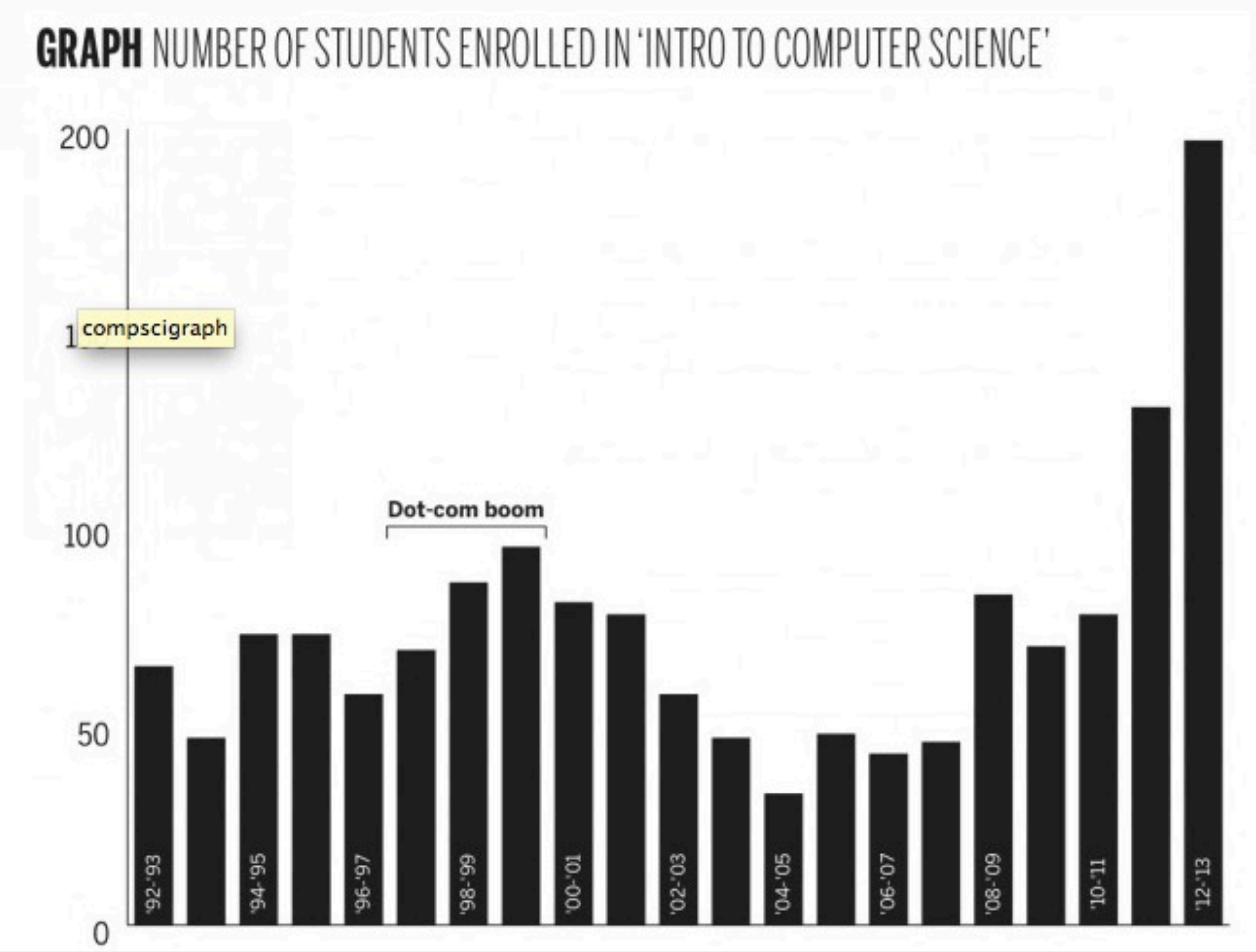
For fun and profit.

Why study algorithms?

For fun and profit.

Why study algorithms?

Everyone else is doing it, so why shouldn't we?



http://yaledailynews.com/blog/2013/01/29/computer-science-dept-overworked-understaffed/

The usual suspects

Lectures. Introduce new material.

Precepts. Discussion, problem-solving, background for assignments.

What	When	Where	Who
L01	MW 11–12:20	McCosh 10	Josh Hug Arvind Narayanan
P01	Th 11:00 - 11:50	Friend 109	Josh Hug
P02	Th 12:30 - 1:20	Babst 105	Maia Ginsburg †
P03	Th 1:30 - 2:20	Babst 105	Arvind Narayanan
P08	F 10:00 - 11:00	Friend 109	Maia Ginsburg †
P05	F 11:00 - 11:50	Friend 109	Nico Pegard
P05A	F 11:00 - 11:50	Friend 108	Stefan Munezel
P06	F 2:30 - 3:20	Friend 109	Diego Perez Botero
P06A	F 2:30 - 3:20	Friend 108	Dushant Arora
P07	F 2:30 - 3:20	CS 102	Jennifer Guo
P04	F 3:30 - 4:20	Friend 109	Diego Perez Botero

† lead preceptor

Where to get help?

Where to get help?

Piazza. Online discussion forum.

- Low latency, low bandwidth.
- Mark solution-revealing questions as private.
- TAs will answer In-lecture questions.
- Course announcements.

piazza

http://www.piazza.com/class#fall2012/cos226

Where to get help?

Piazza. Online discussion forum.

- Low latency, low bandwidth.
- Mark solution-revealing questions as private.
- TAs will answer In-lecture questions.
- Course announcements.

piazza

http://www.piazza.com/class#fall2012/cos226

Office hours.

- High bandwidth, high latency.
- See web for schedule.

http://www.princeton.edu/~cos226

Where to get help?

Piazza. Online discussion forum.

- Low latency, low bandwidth.
- Mark solution-revealing questions as private.
- TAs will answer In-lecture questions.
- Course announcements.

piazza

http://www.piazza.com/class#fall2012/cos226

Office hours.

- High bandwidth, high latency.
- See web for schedule.

http://www.princeton.edu/~cos226

Computing laboratory.

- Undergrad lab TAs in Friend 017.
- For help with debugging.
- See web for schedule.

http://www.princeton.edu/~cos226

Coursework and grading

Coursework and grading

Programming assignments. 45%

- Due on Tuesdays at 11pm via electronic submission.
- See web for collaboration and lateness policy.

Coursework and grading

Programming assignments. 45%

- Due on Tuesdays at 11pm via electronic submission.
- See web for collaboration and lateness policy.

Exercises. 10%

- Due on Sundays at 11pm in Blackboard.

Coursework and grading

Programming assignments. 45%

- Due on Tuesdays at 11pm via electronic submission.
- See web for collaboration and lateness policy.

Exercises. 10%

- Due on Sundays at 11pm in Blackboard.

Exams. 15% + 30%

- Midterm (in class on Monday, March 11).
- Final (to be scheduled by Registrar).

Coursework and grading

Programming assignments. 45%

- Due on Tuesdays at 11pm via electronic submission.
- See web for collaboration and lateness policy.

Exercises. 10%

- Due on Sundays at 11pm in Blackboard.

Exams. 15% + 30%

- Midterm (in class on Monday, March 11).
- Final (to be scheduled by Registrar).

Staff discretion. To adjust borderline cases.

- Report errata.
- Contribute to Piazza discussions.
- Attend and participate in precept/lecture.
- Answering in lecture-questions using a device.

Coursework and grading

Programming assignments. 45%

- Due on Tuesdays at 11pm via electronic submission.
- See web for collaboration and lateness policy.

Exercises. 10%

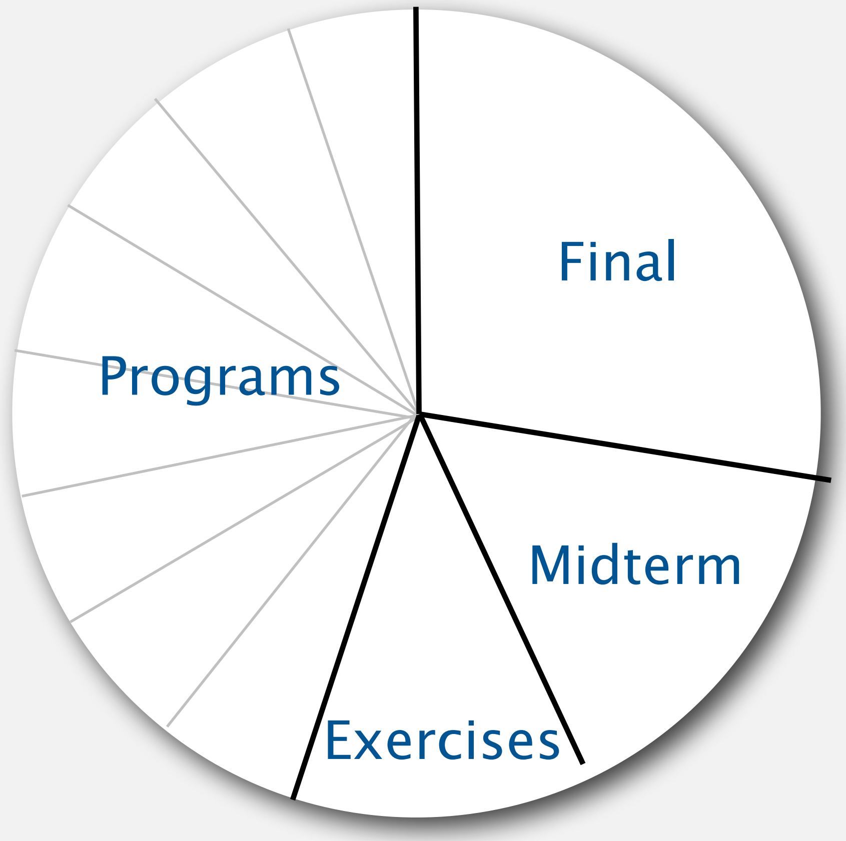
- Due on Sundays at 11pm in Blackboard.

Exams. 15% + 30%

- Midterm (in class on Monday, March 11).
- Final (to be scheduled by Registrar).

Staff discretion. To adjust borderline cases.

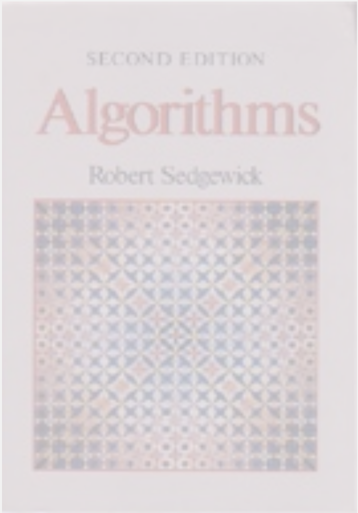
- Report errata.
- Contribute to Piazza discussions.
- Attend and participate in precept/lecture.
- Answering in lecture-questions using a device.



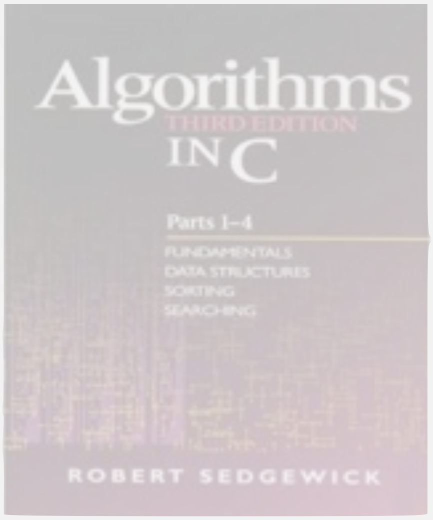
Resources (textbook)

Required reading. Algorithms 4th edition by R. Sedgewick and K. Wayne, Addison-Wesley Professional, 2011, ISBN 0-321-57351-X.

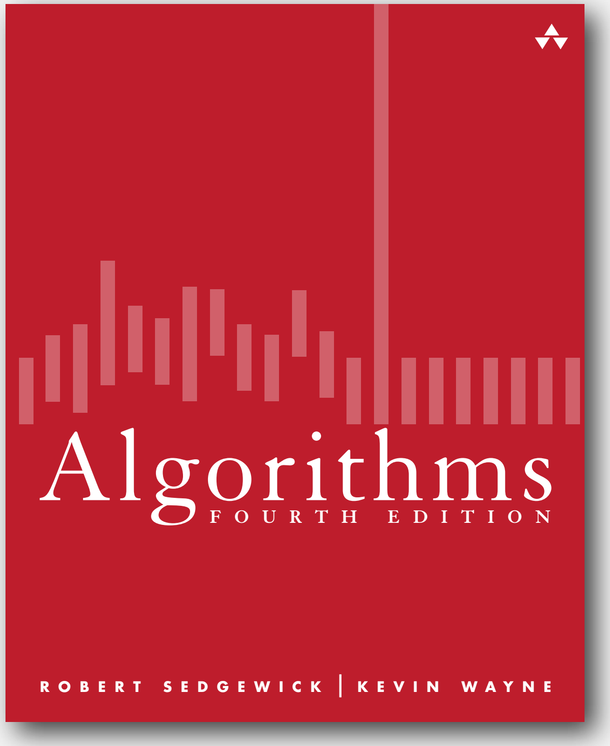
1st edition (1982)



2nd edition (1988)



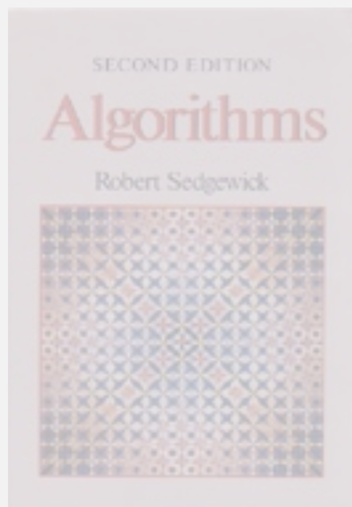
3rd edition (1997)



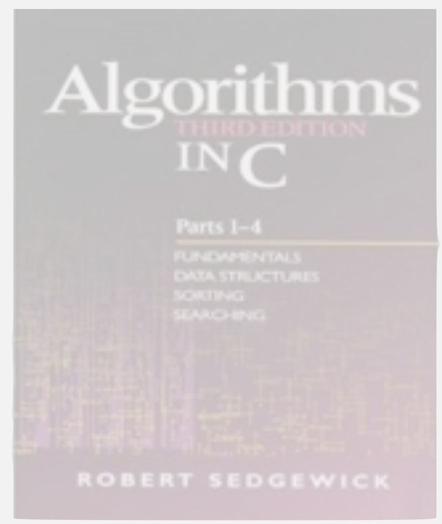
Resources (textbook)

Required reading. Algorithms 4^{th} edition by R. Sedgewick and K. Wayne, Addison-Wesley Professional, 2011, ISBN 0-321-57351-X.

1^{st} edition (1982)



2^{nd} edition (1988)



3^{rd} edition (1997)

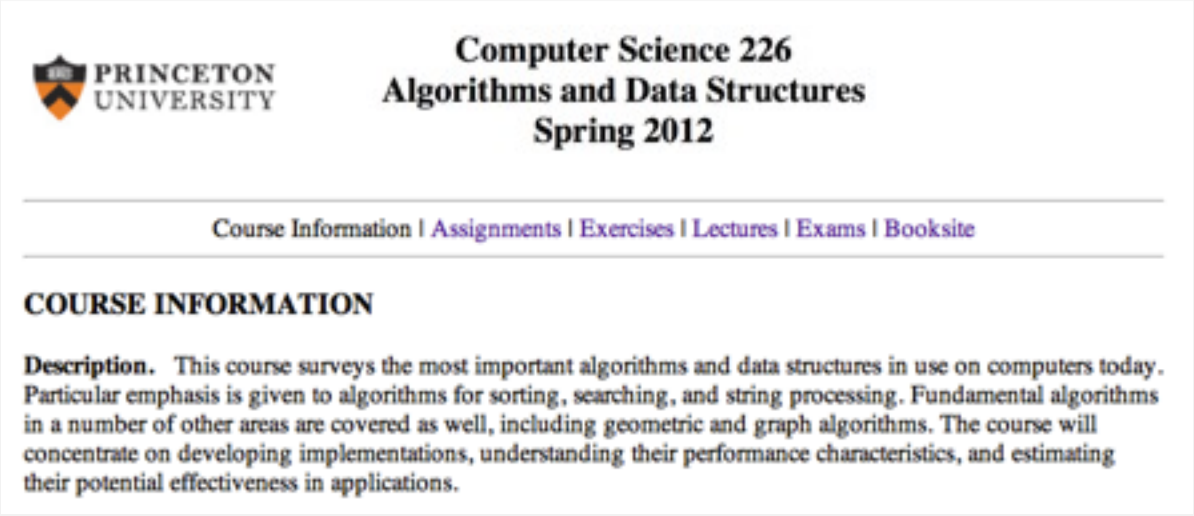
Available in hardcover and Kindle.

- Online: Amazon ($60 to buy), Chegg ($40 to rent), ...
- Brick-and-mortar: Labyrinth Books (122 Nassau St). ← 30% discount with PU student ID
- On reserve: Engineering library.

Resources (web)

Course content.

- Course info.
- Programming assignments.
- Exercises.
- Lecture slides.
- Exam archive.
- Submit assignments.



PRINCETON UNIVERSITY

Computer Science 226
Algorithms and Data Structures
Spring 2012

Course Information | Assignments | Exercises | Lectures | Exams | Booksite

COURSE INFORMATION

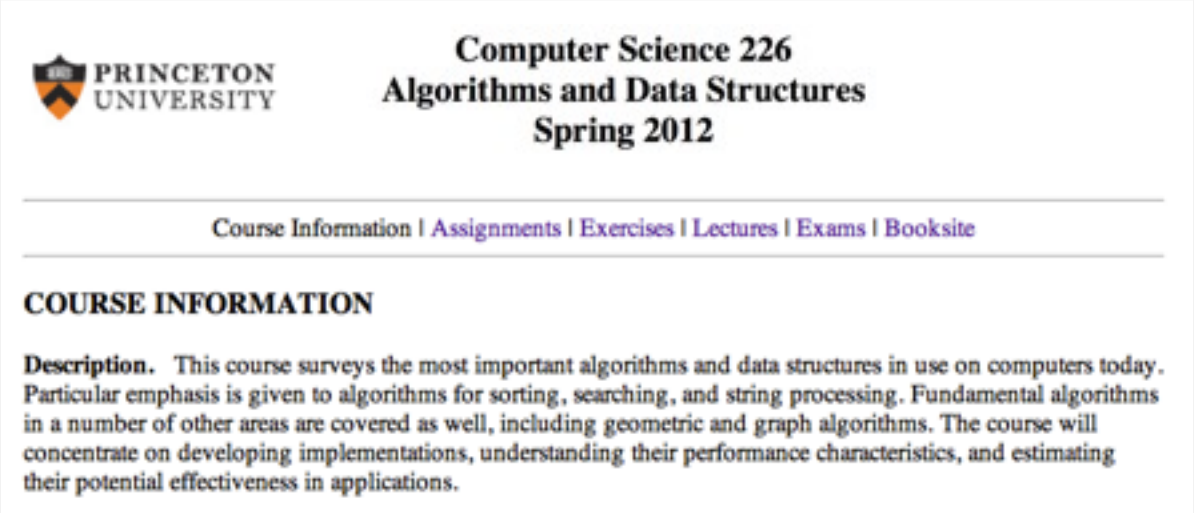
Description. This course surveys the most important algorithms and data structures in use on computers today. Particular emphasis is given to algorithms for sorting, searching, and string processing. Fundamental algorithms in a number of other areas are covered as well, including geometric and graph algorithms. The course will concentrate on developing implementations, understanding their performance characteristics, and estimating their potential effectiveness in applications.

http://www.princeton.edu/~cos226

Resources (web)

Course content.

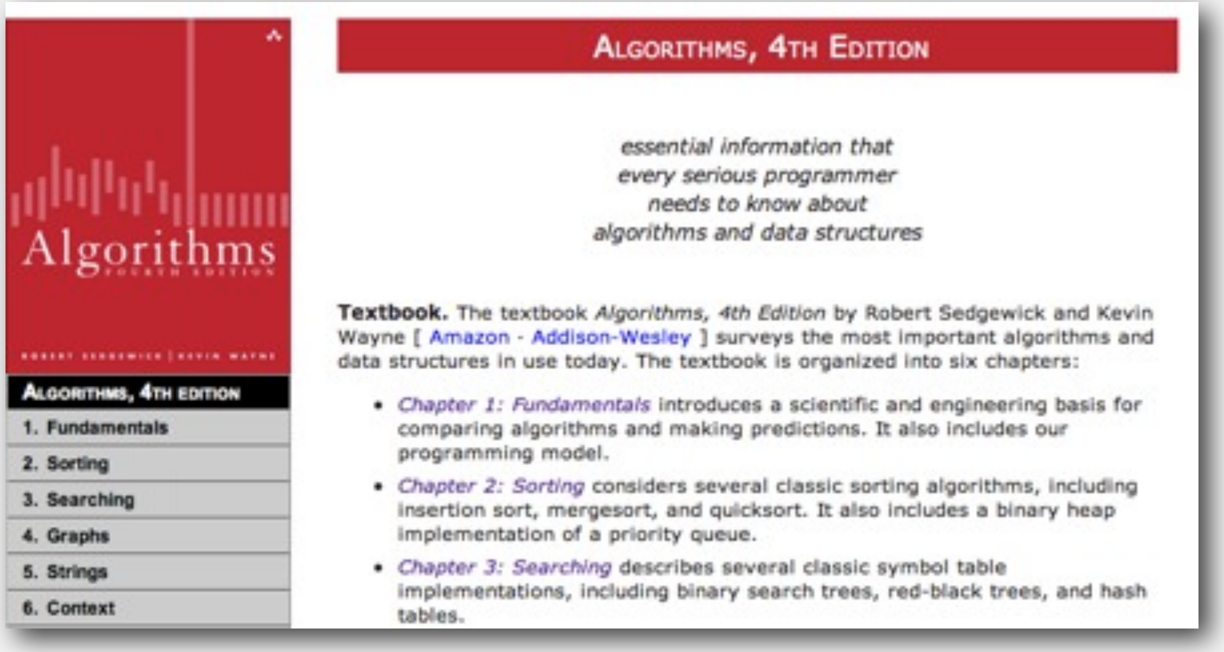
- Course info.
- Programming assignments.
- Exercises.
- Lecture slides.
- Exam archive.
- Submit assignments.



http://www.princeton.edu/~cos226

Booksites.

- Brief summary of content.
- Download code from book.

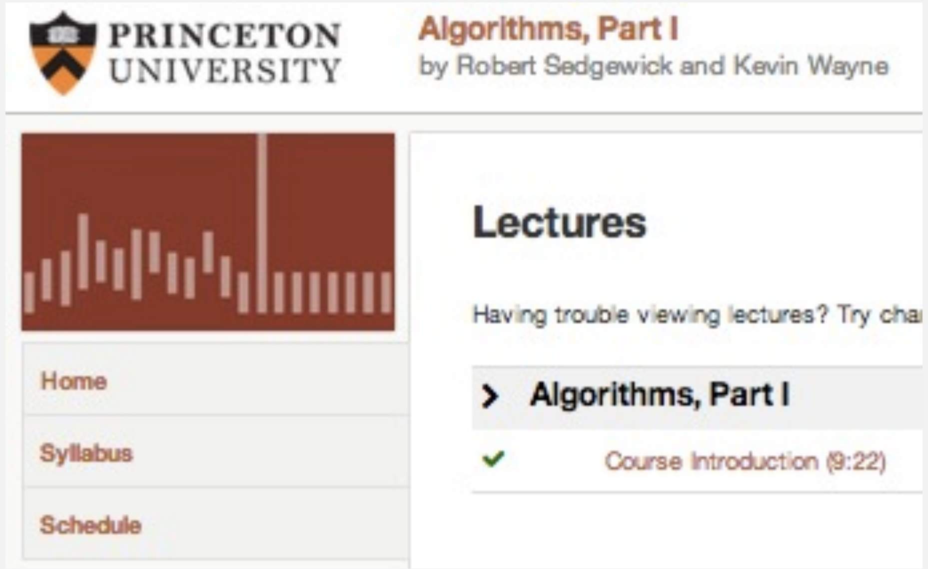


http://www.algs4.princeton.edu

Resources (Coursera) and Flipped Lectures

Coursera Course

- Lectures by Bob Sedgewick.
  - Same content as ours.
- Don't submit assignments!
  - Violates course policy.

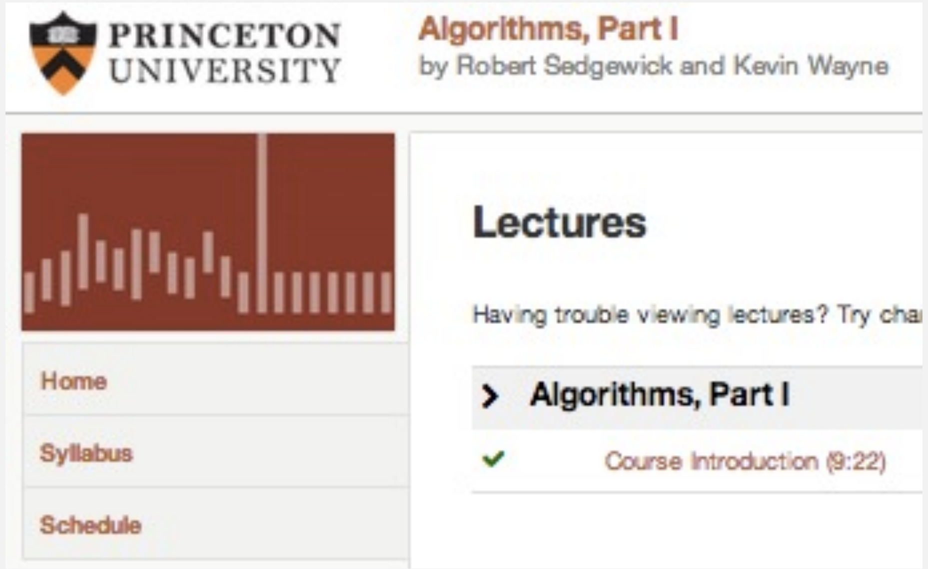


https://class.coursera.org/algs4partI-002/class

Resources (Coursera) and Flipped Lectures

Coursera Course

- Lectures by Bob Sedgewick.
  - Same content as ours.
- Don't submit assignments!
  - Violates course policy.

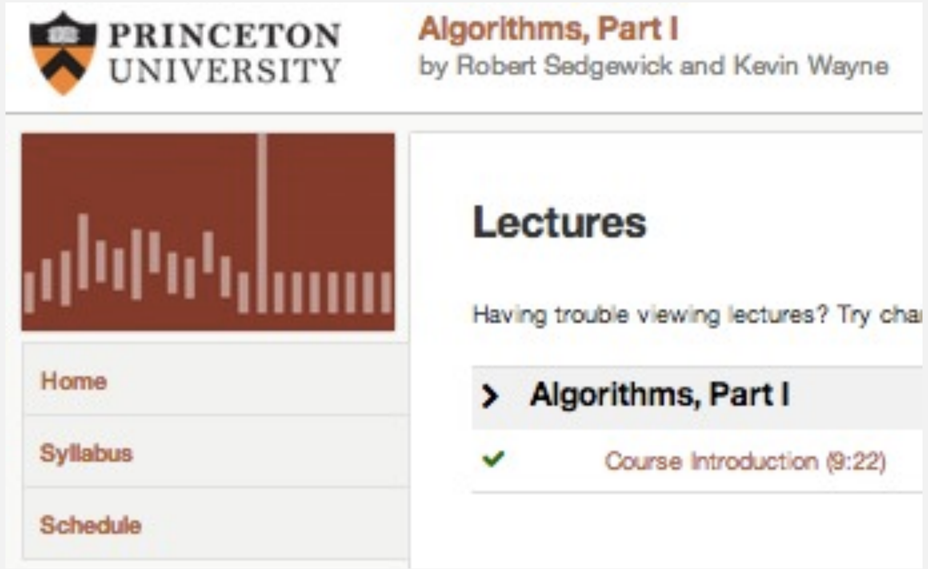


https://class.coursera.org/algs4partI-002/class

Resources (Coursera) and Flipped Lectures

Coursera Course

- Lectures by Bob Sedgewick.
  - Same content as ours.
- Don't submit assignments!
  - Violates course policy.



https://class.coursera.org/algs4partI-002/class

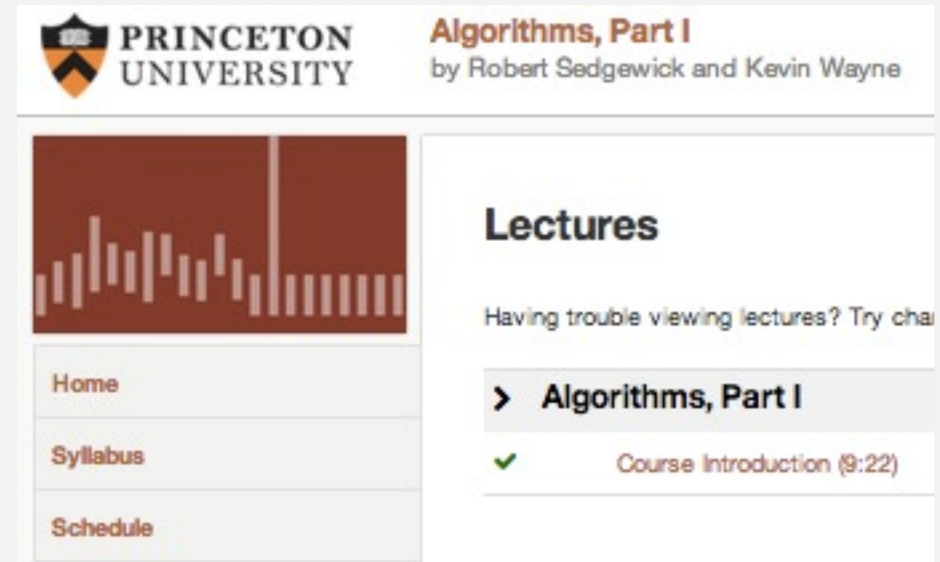
The Flipped Lecture Experiment

- Weeks 4-6 (and more?).
- Watch lectures on Coursera.
- Activities in Lecture.
  - Big picture mini-lectures.
  - Interesting anecdotes.
  - Solo/group work.
  - Old exam problems.
  - Guest speakers.
  - Open Q&A.

Resources (Coursera) and Flipped Lectures

Coursera Course

- Lectures by Bob Sedgewick.
  - Same content as ours.
- Don't submit assignments!
  - Violates course policy.



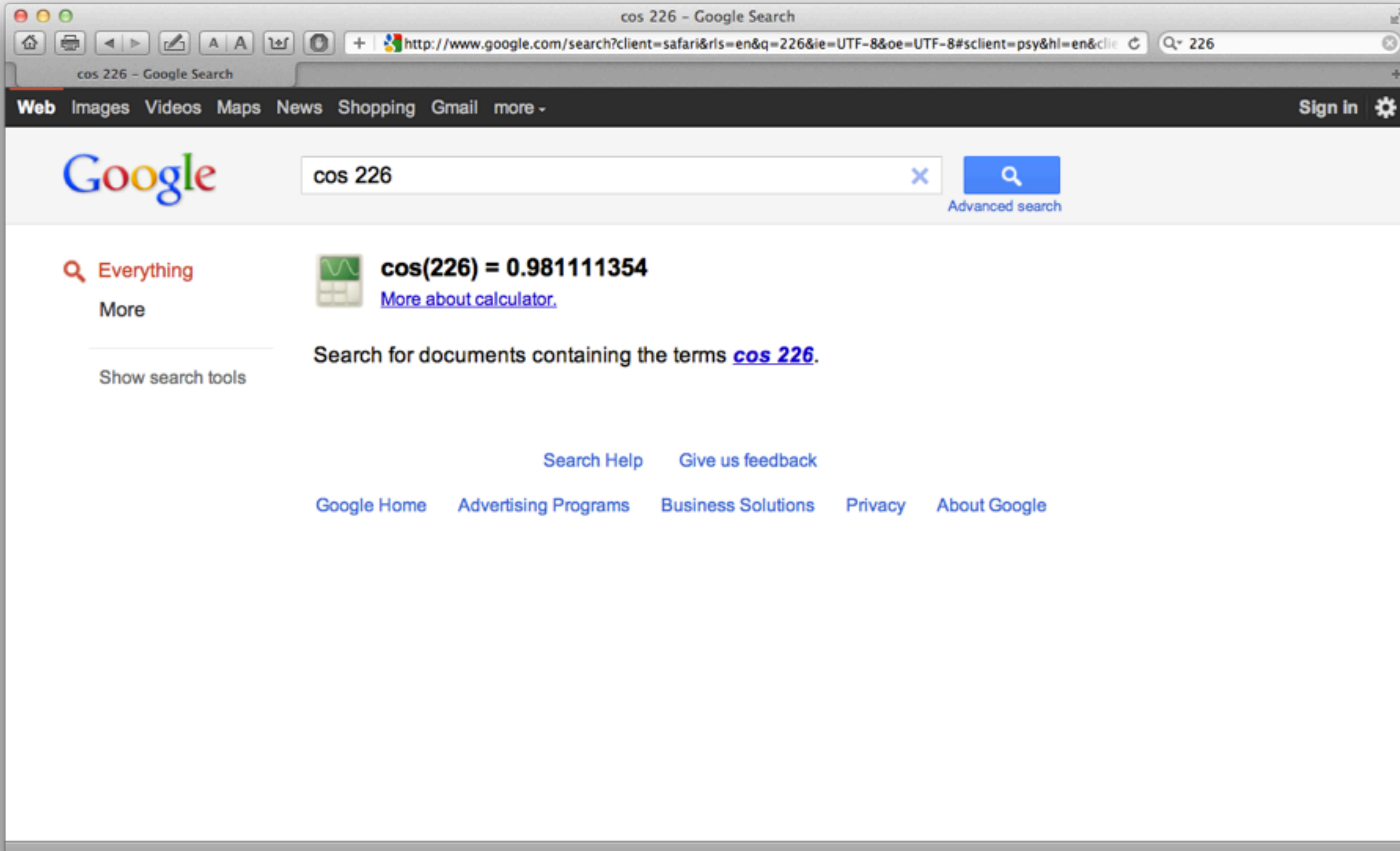
https://class.coursera.org/algs4partI-002/class

The Flipped Lecture Experiment

- Weeks 4-6 (and more?).
- Watch lectures on Coursera.
- Activities in Lecture.
  - Big picture mini-lectures.
  - Interesting anecdotes.
  - Solo/group work.
  - Old exam problems.
  - Guest speakers.
  - Open Q&A.

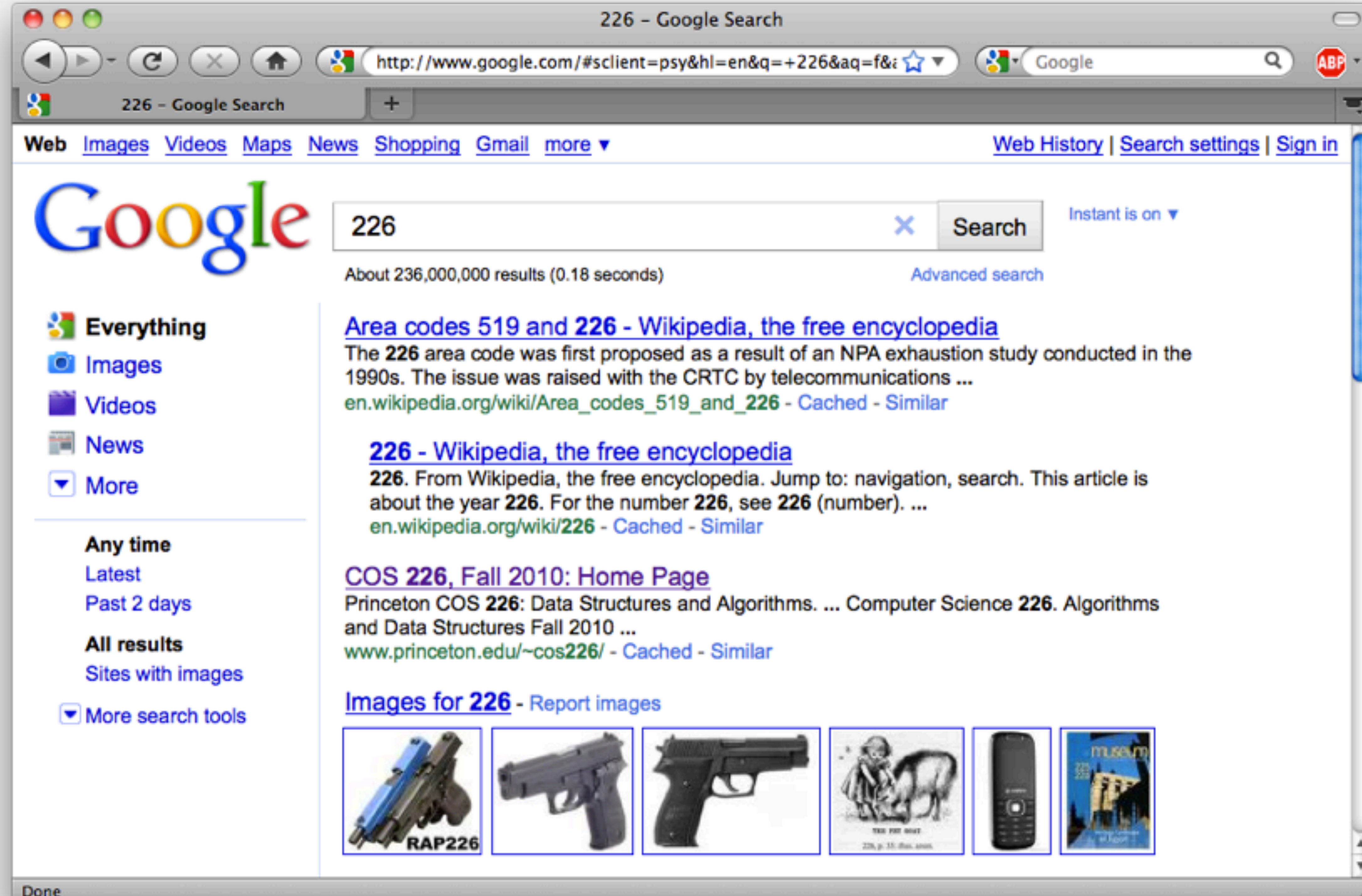
#	DATE	TOPIC	SLIDES	READINGS	DEMOS
		Lectures and dates below are still tentative for Spring 2013			
1	2/4	Intro · Union Find	1up · 4up	1.5	Quick-find · Quick-union
2	2/6	Analysis of Algorithms	1up · 4up	1.4	Binary search
3	2/11	Stacks and Queues	1up · 4up	1.3	Dijkstra 2-stack
4	2/13	Elementary Sorts	1up · 4up	2.1	Selection · Insertion · Shuffle · Graham
5	2/18	Mergesort	1up · 4up	2.2	Merging
6	2/20	Quicksort	1up · 4up	2.3	Partitioning
7	2/25	Priority Queues	1up · 4up	2.4	Heap · Heapsort
8	2/27	Elementary Symbol Tables · BSTs	1up · 4up	3.1–3.2	BST
9	3/4	Balanced Search Trees	1up · 4up	3.3	2-3 tree · Red-black BST
10	3/6	Hash Tables · Searching Applications	1up · 4up	3.4–3.5	linear probing
11	3/11	Midterm Exam		–	–
12	3/13	Geometric Applications of BSTs	1up · 4up	–	Kd tree · Interval search tree

Resources (web)



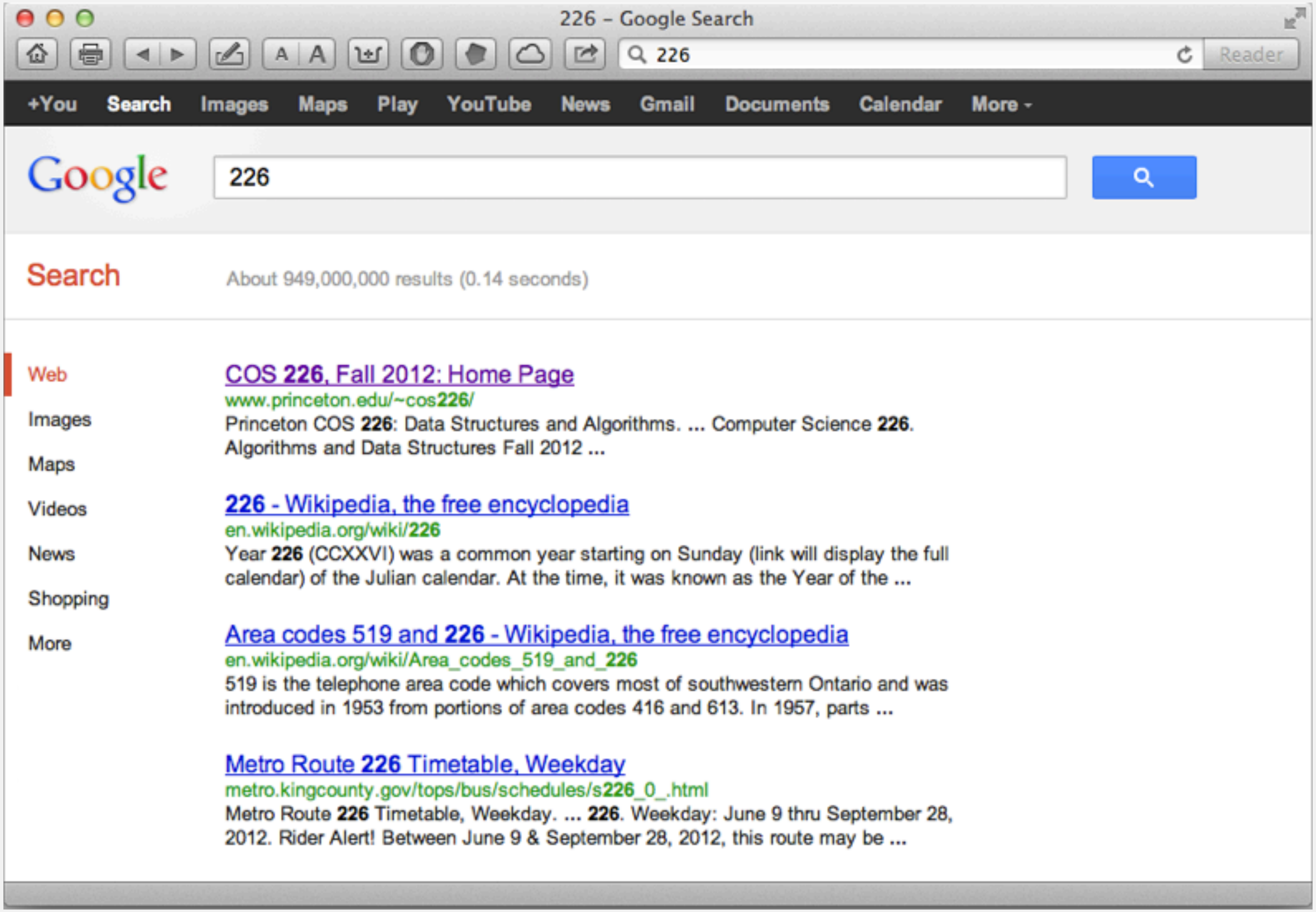
http://www.princeton.edu/~cos226

Resources (web)



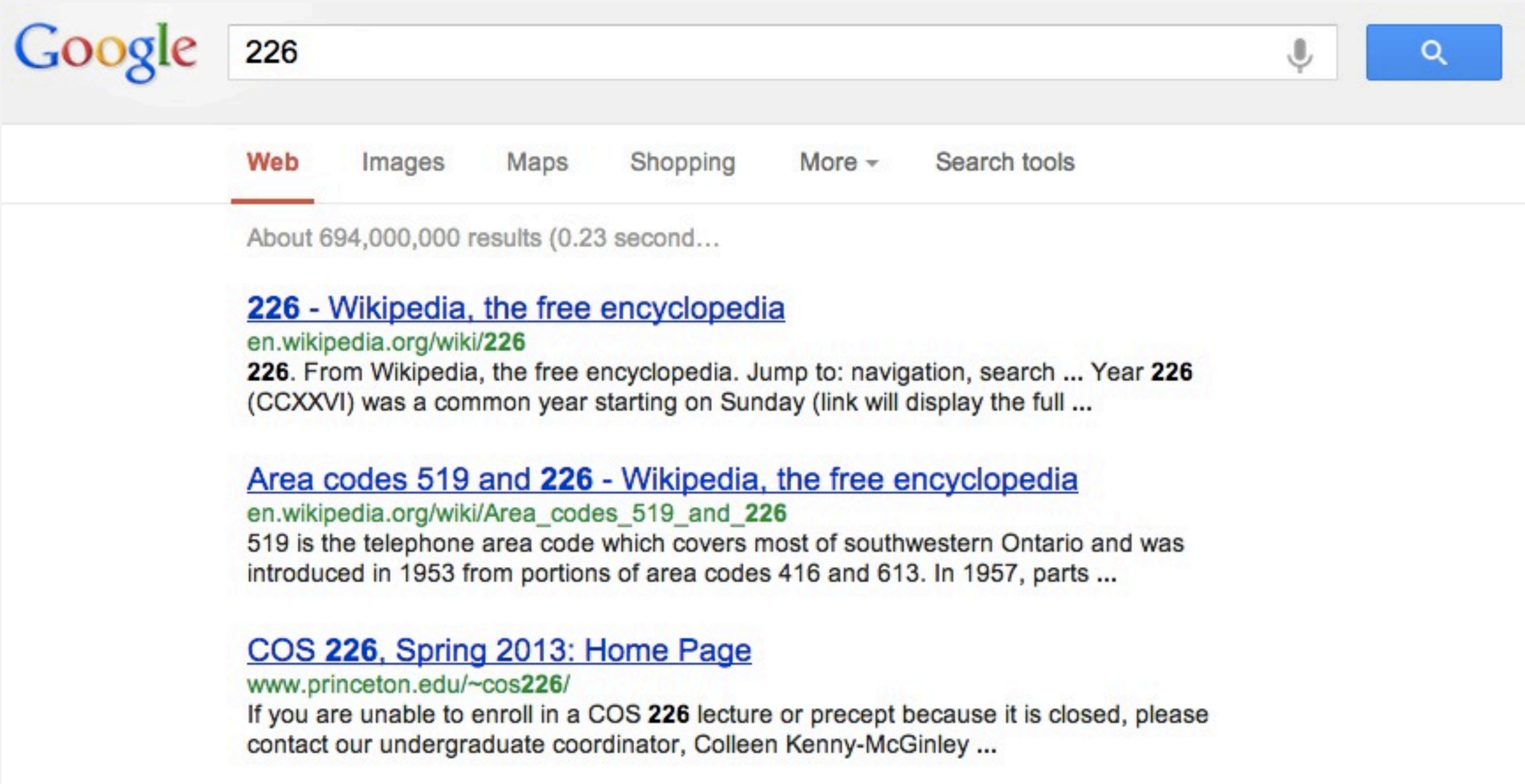
http://www.princeton.edu/~cos226

Resources (web)



http://www.princeton.edu/~cos226

Resources (web)



What's ahead?

Lecture 1. [today] Union find.

Lecture 2. [Wednesday] Analysis of algorithms.

Precept 1. [Thursday/Friday] Meets this week.

What's ahead?

Lecture 1. [today] Union find.

Lecture 2. [Wednesday] Analysis of algorithms.

Precept 1. [Thursday/Friday] Meets this week.

Exercise 1. Due via Bb submission at 11pm on Sunday, February 10th.

Assignment 1. Due via electronic submission at 11pm on Tuesday, February 12th. Pro tip: Start early.

What's ahead?

Lecture 1. [today] Union find.

Lecture 2. [Wednesday] Analysis of algorithms.

Precept 1. [Thursday/Friday] Meets this week.

Exercise 1. Due via Bb submission at 11pm on Sunday, February 10th.

Assignment 1. Due via electronic submission at 11pm on Tuesday, February 12th. Pro tip: Start early.

Right course? See me.

Placed out of COS 126? Review Sections 1.1–1.2 of Algorithms, 4th edition (includes command-line interface and our I/O libraries).

What's ahead?

Lecture 1. [today] Union find.

Lecture 2. [Wednesday] Analysis of algorithms.

Precept 1. [Thursday/Friday] Meets this week.

Exercise 1. Due via Bb submission at 11pm on Sunday, February 10th.

Assignment 1. Due via electronic submission at 11pm on Tuesday, February 12th. Pro tip: Start early.

Right course? See me.

Placed out of COS 126? Review Sections 1.1–1.2 of Algorithms, 4th edition (includes command-line interface and our I/O libraries).

Not registered? Go to any precept this week [only if not registered!].

Change precept? Use SCORE. ← see Colleen Kenny-McGinley in CS 210 if the only precept you can attend is closed

http://algs4.cs.princeton.edu

1.5 Union-Find

- *dynamic connectivity*
- *quick find*
- *quick union*
- *improvements*
- *applications*

Subtext of today's lecture (and this course)

Steps to developing a usable algorithm.

- Model the problem.
- Find an algorithm to solve it.
- Fast enough? Fits in memory?
- If not, figure out why.
- Find a way to address the problem.
- Iterate until satisfied.

The scientific method.

Mathematical analysis.

http://algs4.cs.princeton.edu

1.5 UNION-FIND

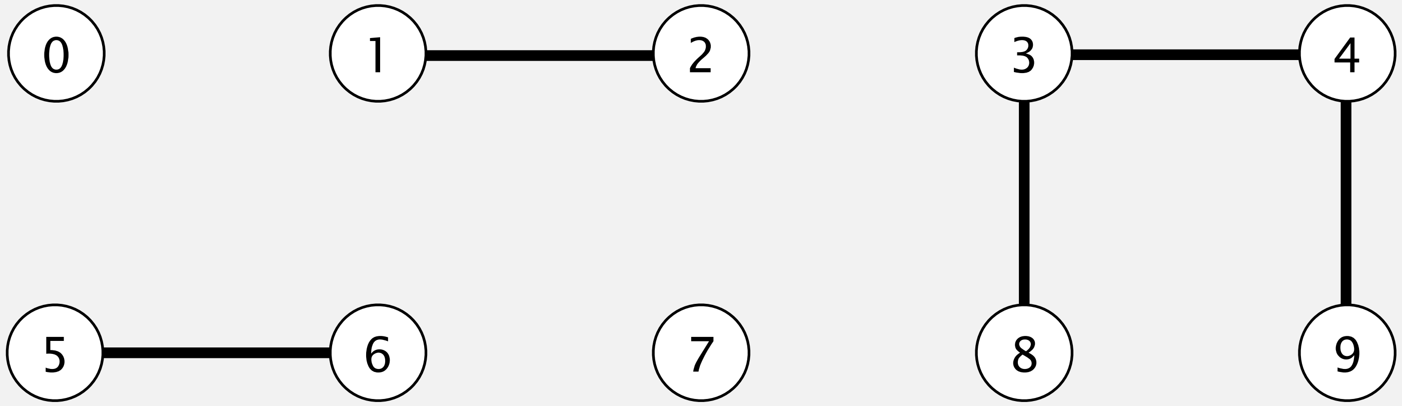
- *dynamic connectivity*
- *quick find*
- *quick union*
- *improvements*
- *applications*

Dynamic connectivity

Given a set of N objects.

- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
```

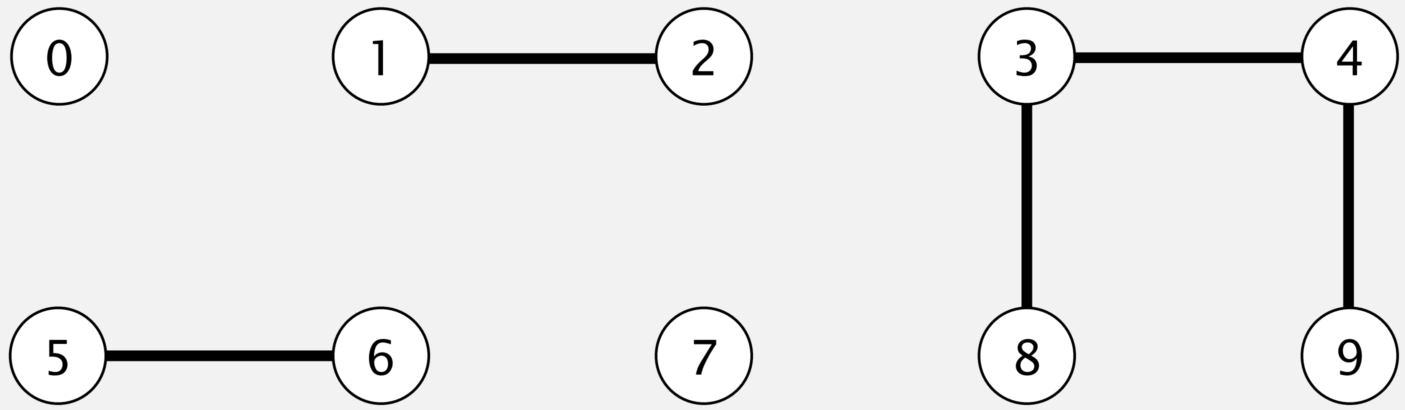


Dynamic connectivity

Given a set of N objects.

- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)
```



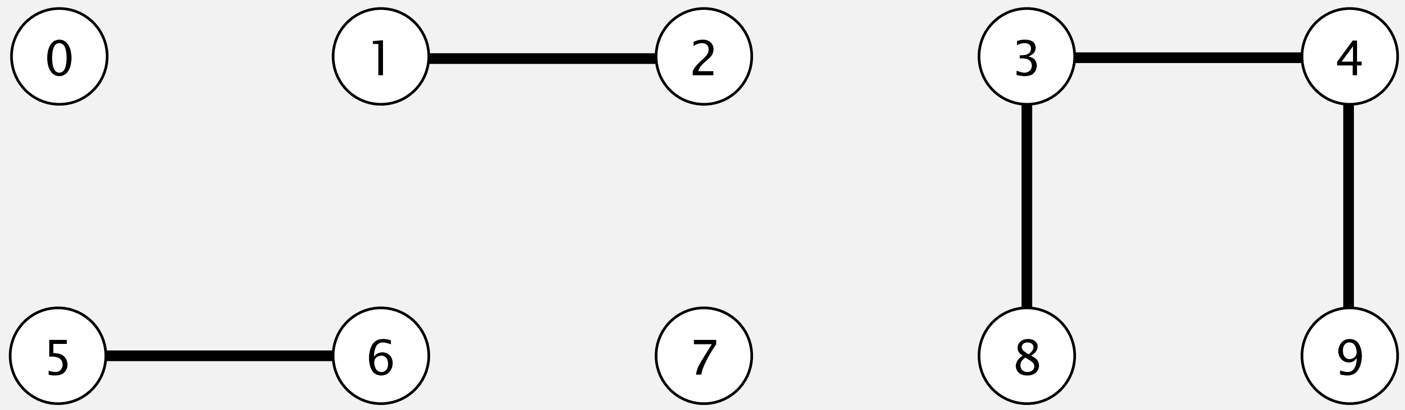
Dynamic connectivity

Given a set of N objects.

- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)
```

✕

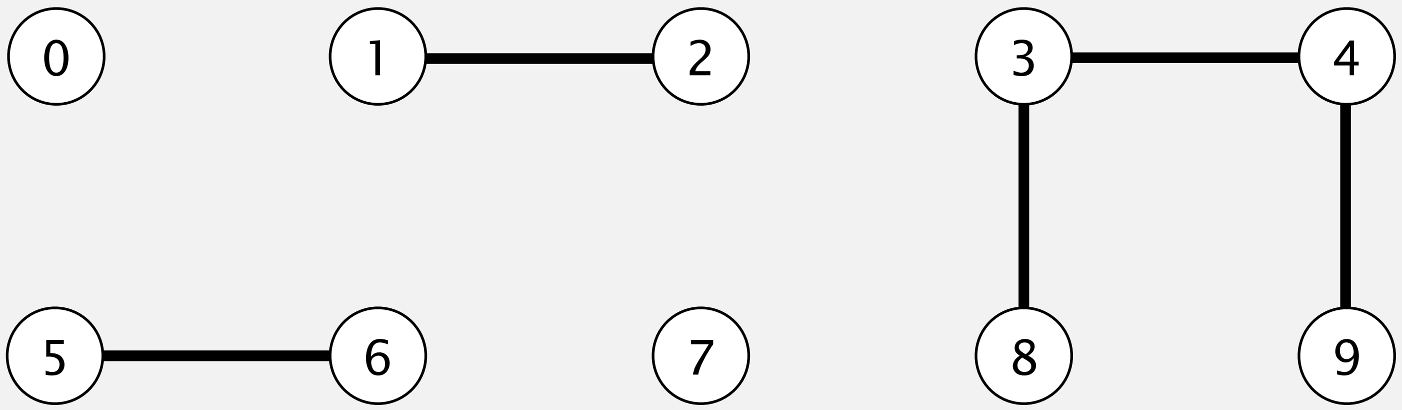


Dynamic connectivity

Given a set of N objects.

- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)   ×
connected(8, 9)
```

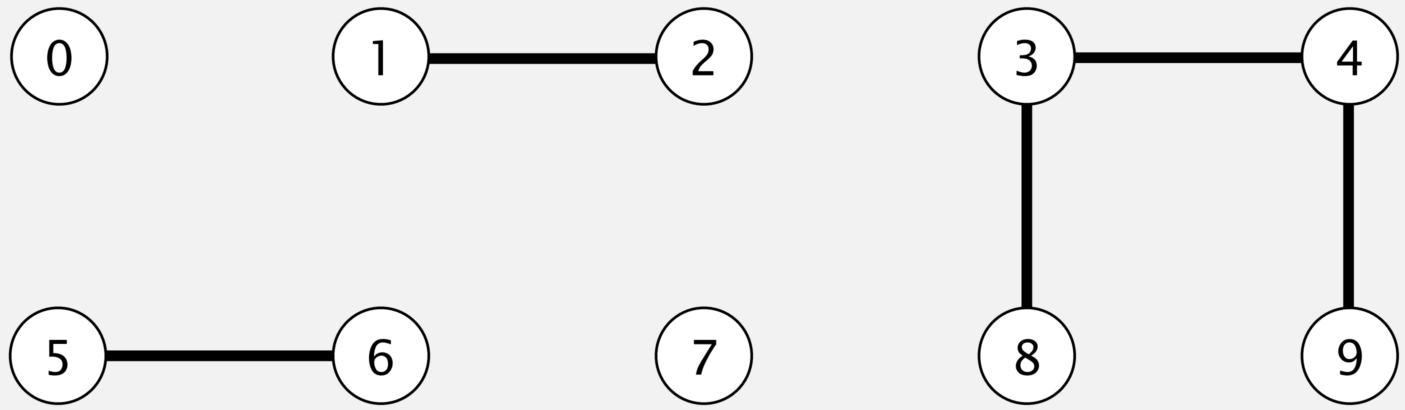


Dynamic connectivity

Given a set of N objects.

- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)   ✗
connected(8, 9)   ✔
```

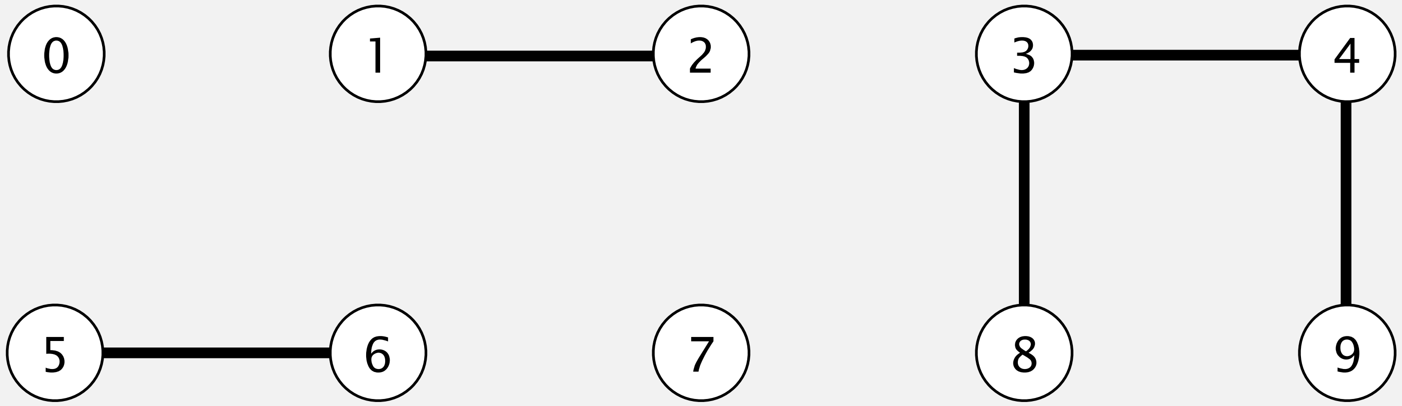


Dynamic connectivity

Given a set of N objects.

- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)    ✗
connected(8, 9)    ✔
union(5, 0)
```

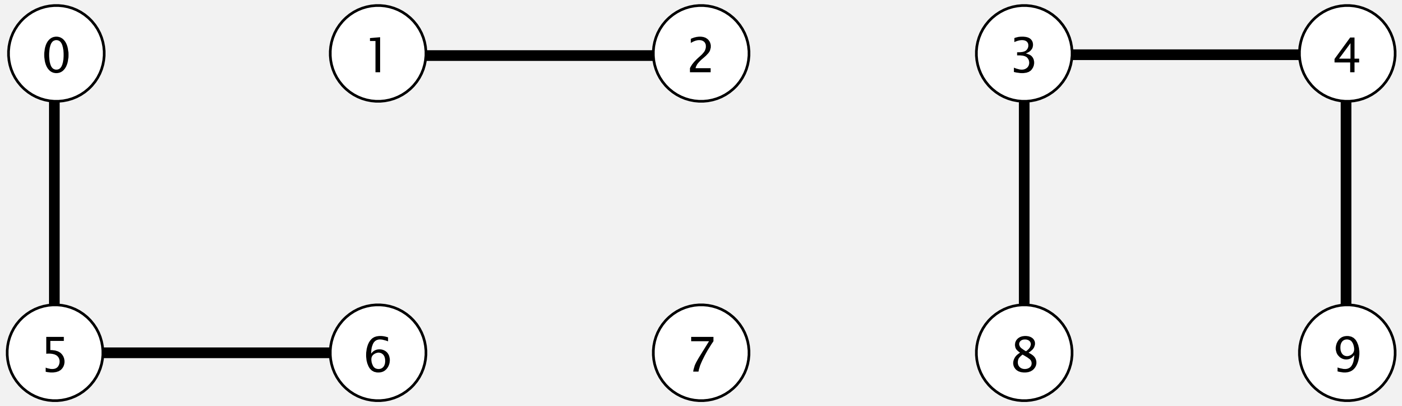


Dynamic connectivity

Given a set of N objects.

- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)    ✗
connected(8, 9)    ✔
union(5, 0)
```

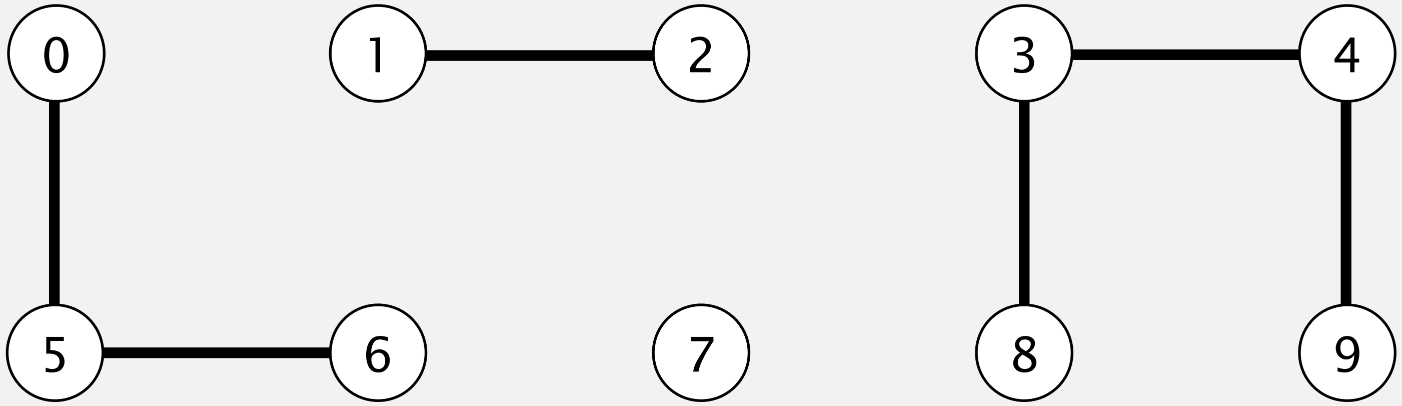


Dynamic connectivity

Given a set of N objects.

- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)    ✗
connected(8, 9)    ✔
union(5, 0)
union(7, 2)
```

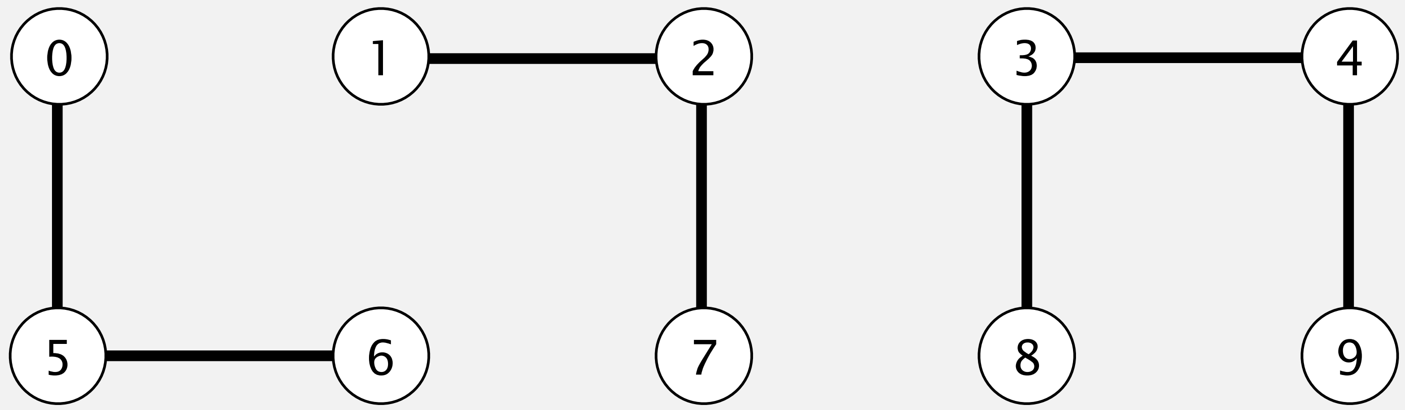


Dynamic connectivity

Given a set of N objects.

- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)   ✗
connected(8, 9)   ✔
union(5, 0)
union(7, 2)
```

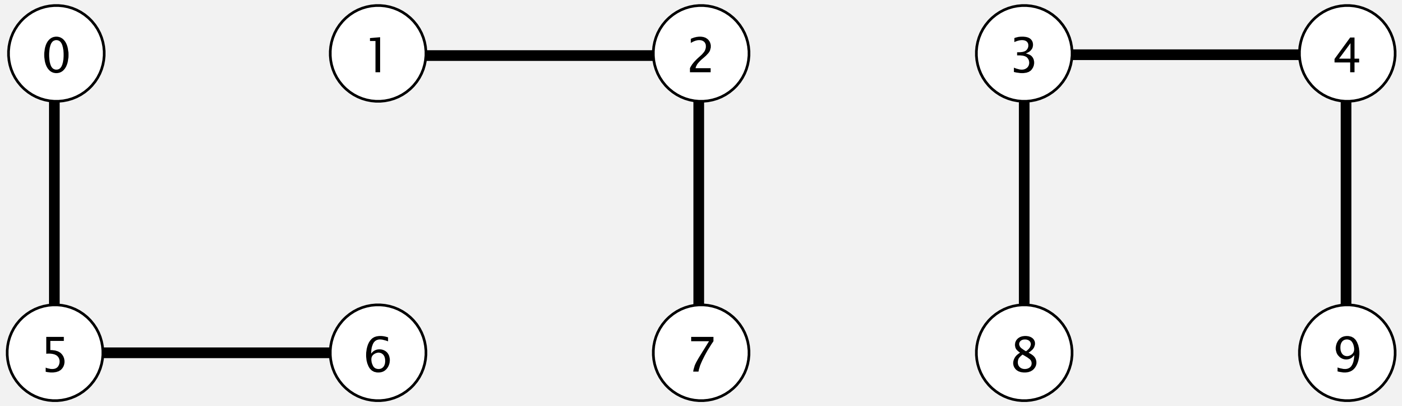


Dynamic connectivity

Given a set of N objects.

- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)    ✗
connected(8, 9)    ✔
union(5, 0)
union(7, 2)
union(6, 1)
```

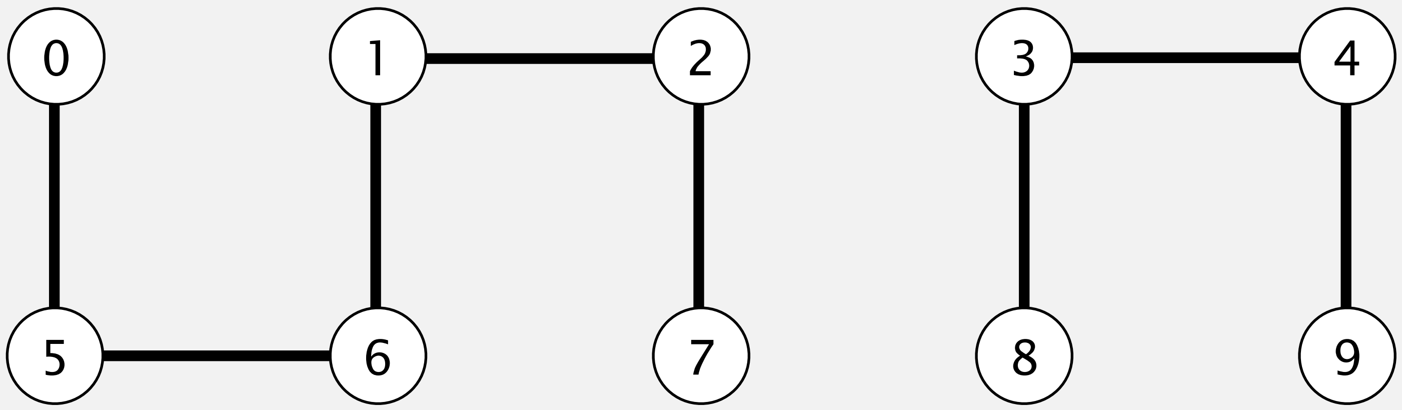


Dynamic connectivity

Given a set of N objects.

- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)    ×
connected(8, 9)    ✔
union(5, 0)
union(7, 2)
union(6, 1)
```

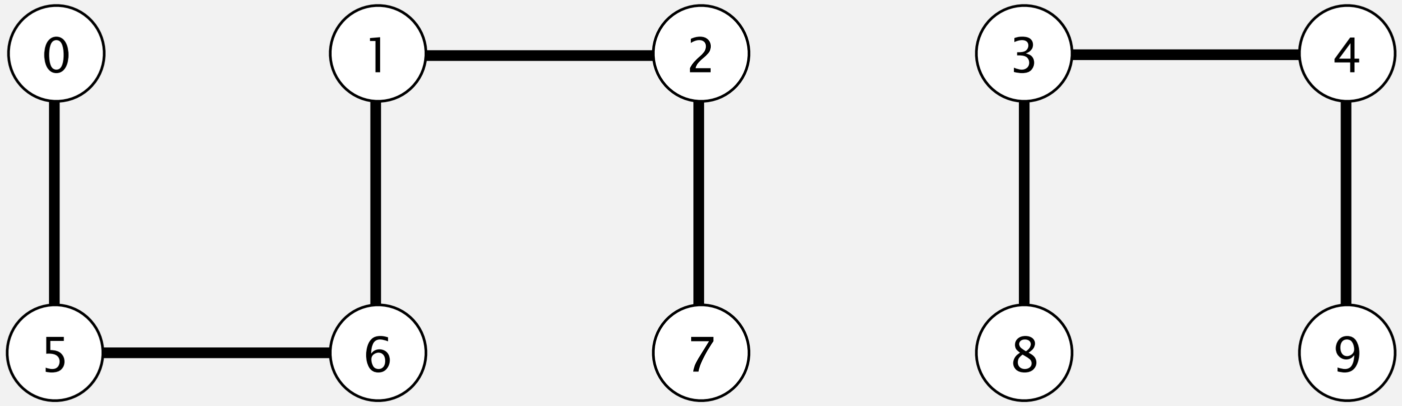


Dynamic connectivity

Given a set of N objects.

- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)   ×
connected(8, 9)   ✔
union(5, 0)
union(7, 2)
union(6, 1)
union(1, 0)
```

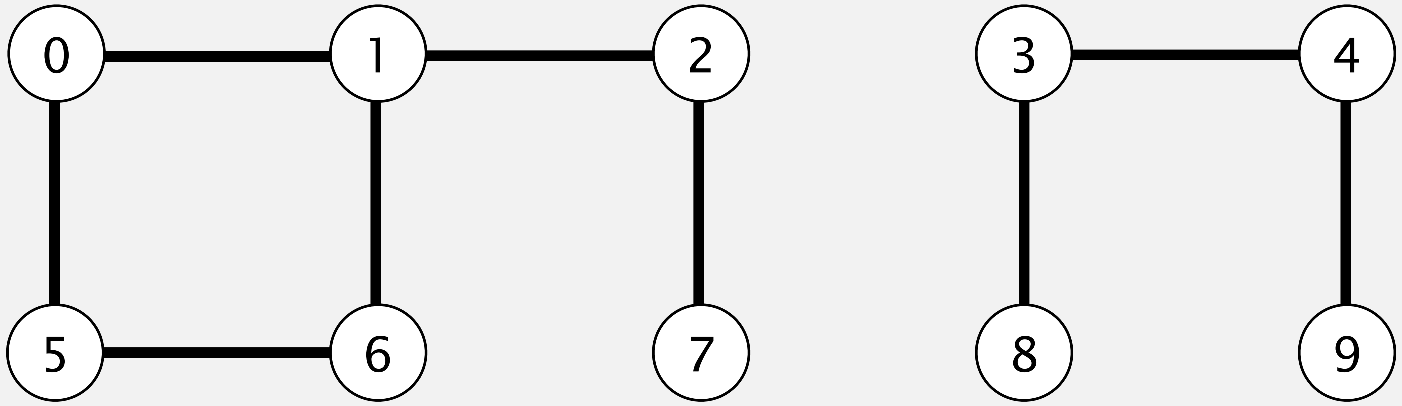


Dynamic connectivity

Given a set of N objects.

- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)    ✗
connected(8, 9)    ✔
union(5, 0)
union(7, 2)
union(6, 1)
union(1, 0)
```

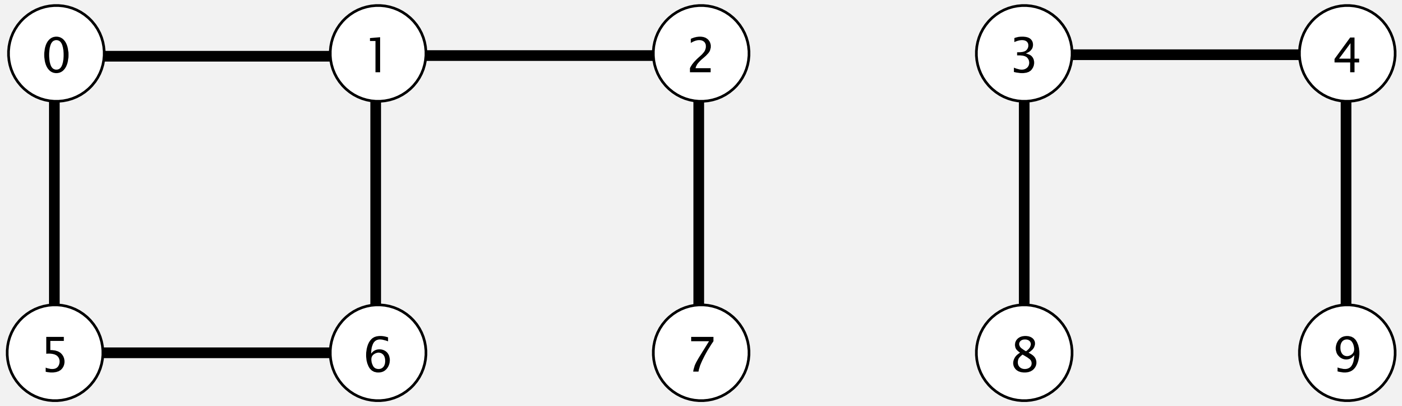


Dynamic connectivity

Given a set of N objects.

- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)    ✗
connected(8, 9)    ✔
union(5, 0)
union(7, 2)
union(6, 1)
union(1, 0)
connected(0, 7)
```

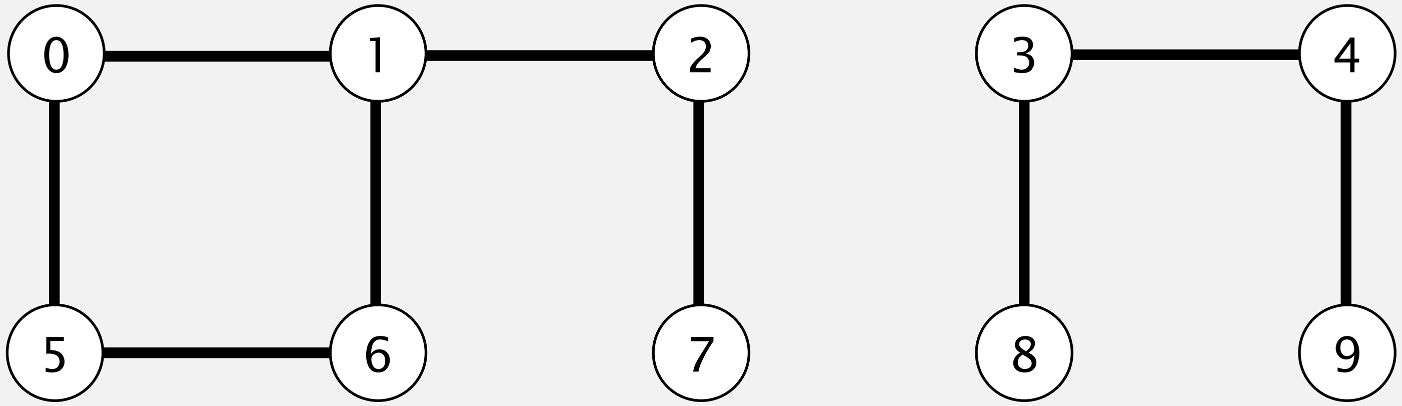


Dynamic connectivity

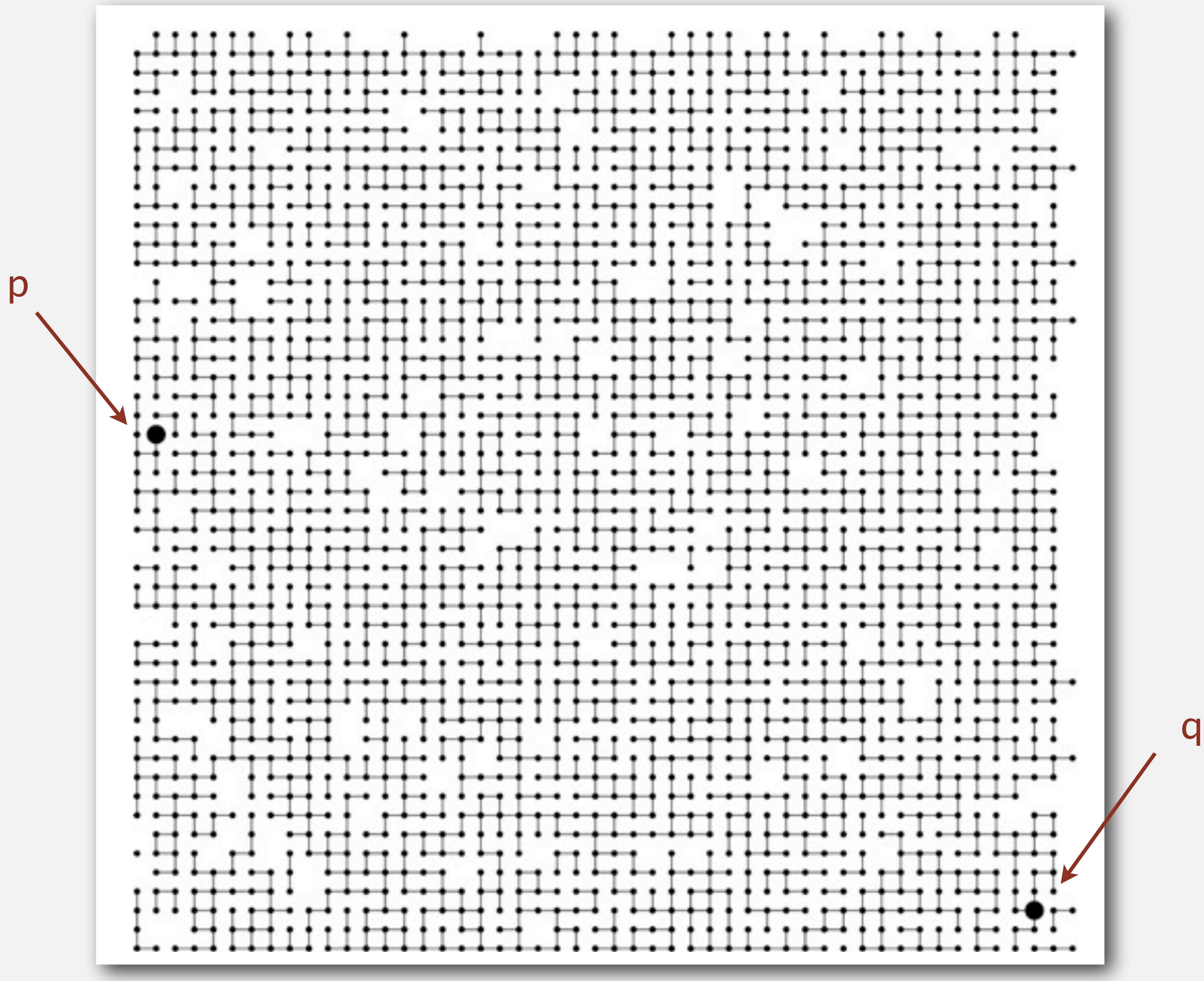
Given a set of N objects.

- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)    ✗
connected(8, 9)    ✔
union(5, 0)
union(7, 2)
union(6, 1)
union(1, 0)
connected(0, 7)    ✔
```

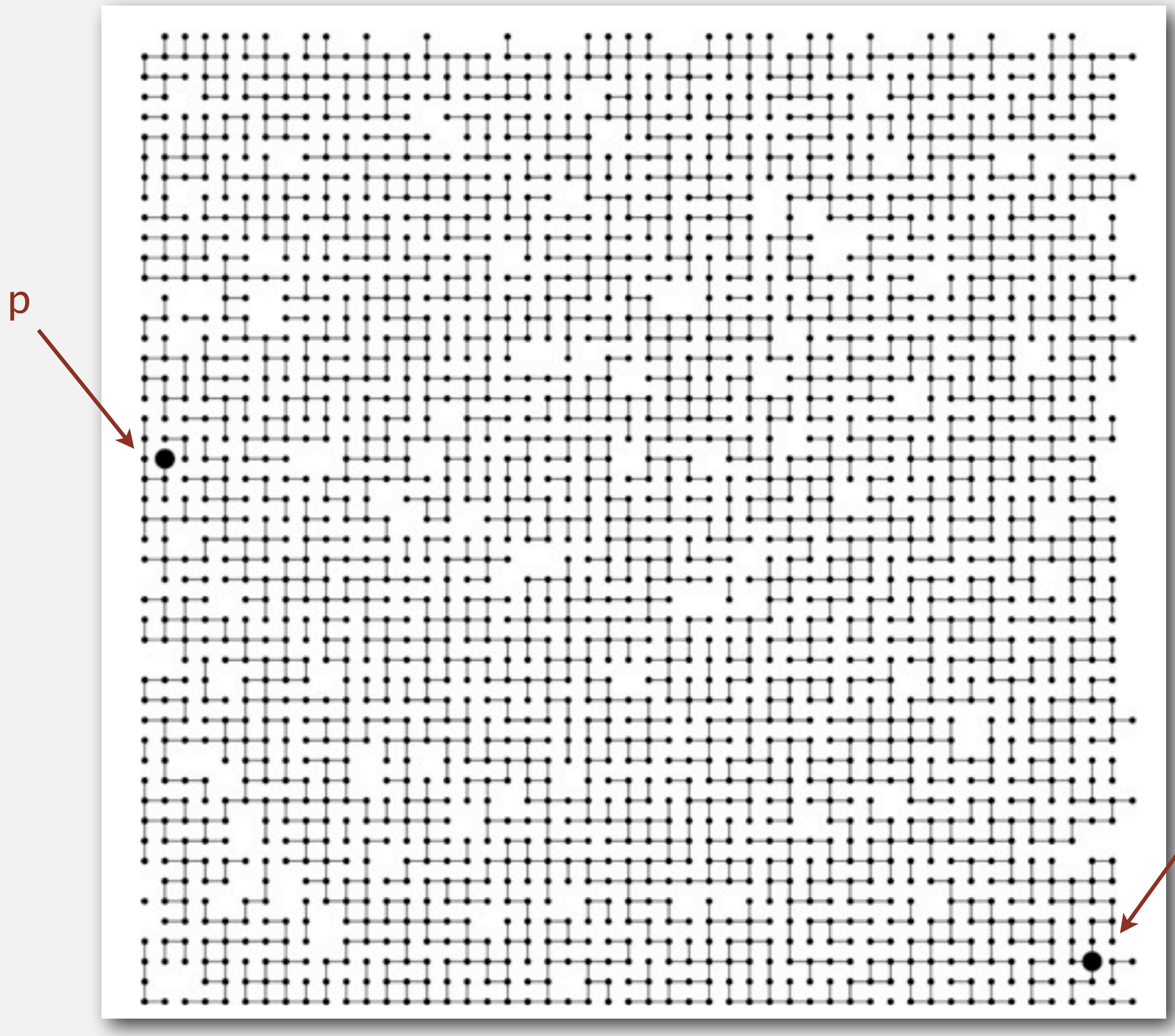


Connectivity example



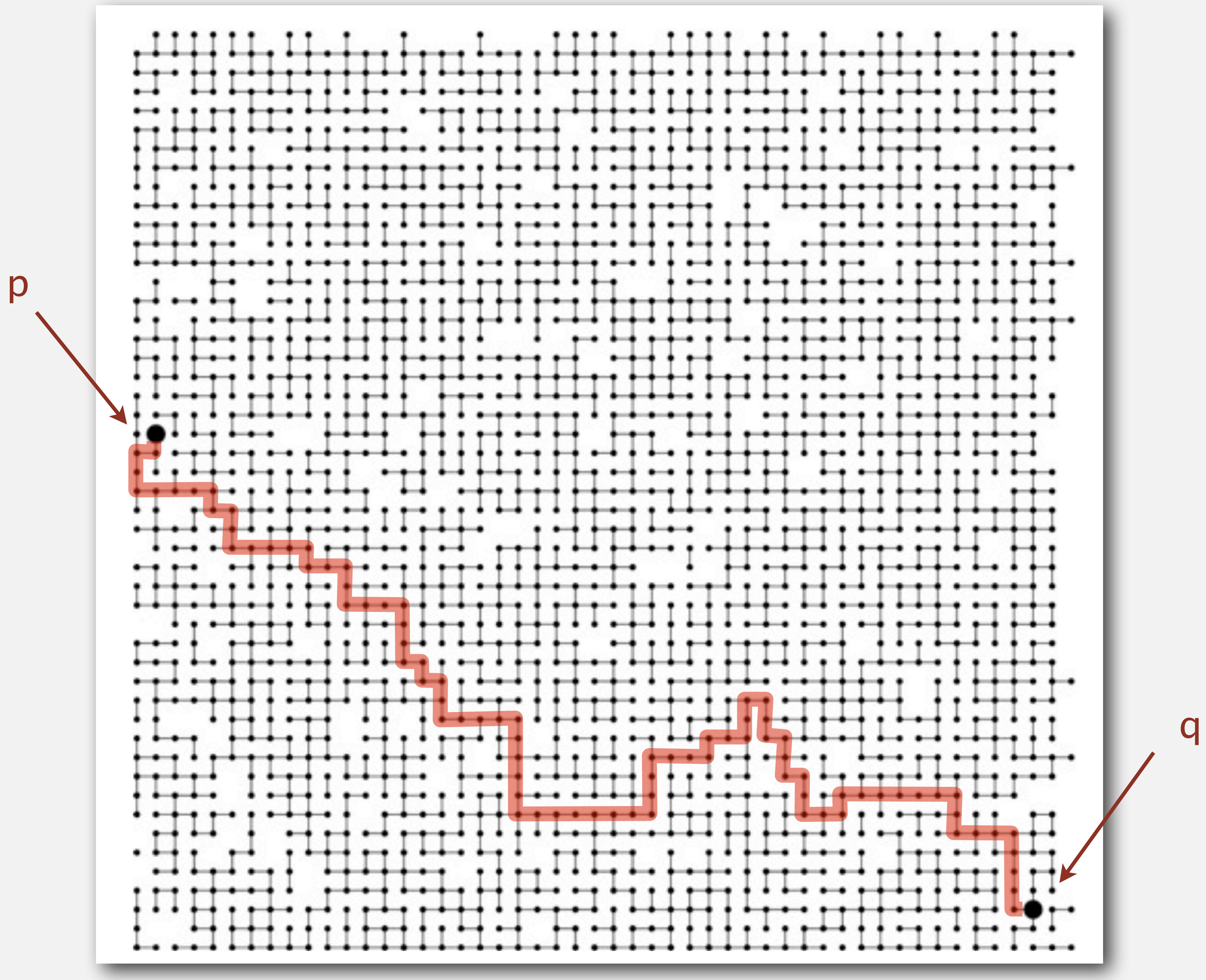
Connectivity example

Q. Is there a path connecting p and q ?



Connectivity example

Q. Is there a path connecting p and q ?



A. Yes.

Modeling the objects

Applications involve manipulating objects of all types.

- Pixels in a digital photo.
- Computers in a network.
- Friends in a social network.
- Transistors in a computer chip.
- Elements in a mathematical set.
- Variable names in Fortran program.
- Metallic sites in a composite system.

Modeling the objects

Applications involve manipulating objects of all types.

- Pixels in a digital photo.
- Computers in a network.
- Friends in a social network.
- Transistors in a computer chip.
- Elements in a mathematical set.
- Variable names in Fortran program.
- Metallic sites in a composite system.

When programming, convenient to name objects 0 to N –1.

- Use integers as array index.
- Suppress details not relevant to union-find.

can use symbol table to translate from site names to integers: stay tuned (Chapter 3)

Modeling the connections

We assume "is connected to" is an equivalence relation:

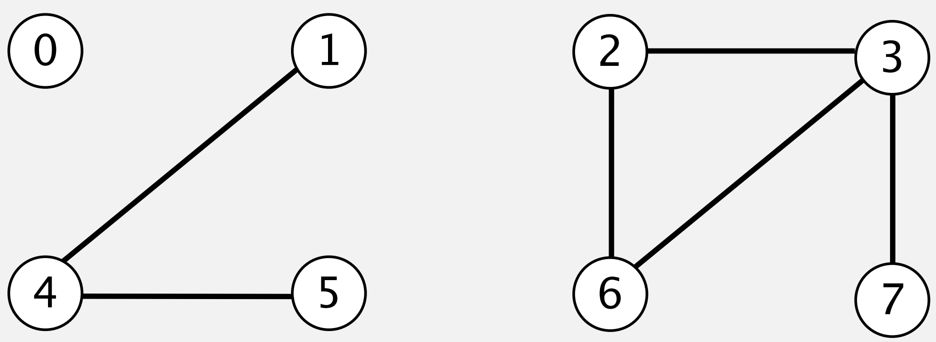
- Reflexive: p is connected to p.
- Symmetric: if p is connected to q, then q is connected to p.
- Transitive: if p is connected to q and q is connected to r, then p is connected to r.

Modeling the connections

We assume "is connected to" is an equivalence relation:

- Reflexive: p is connected to p.
- Symmetric: if p is connected to q, then q is connected to p.
- Transitive: if p is connected to q and q is connected to r, then p is connected to r.

Connected components. Maximal set of objects that are mutually connected.



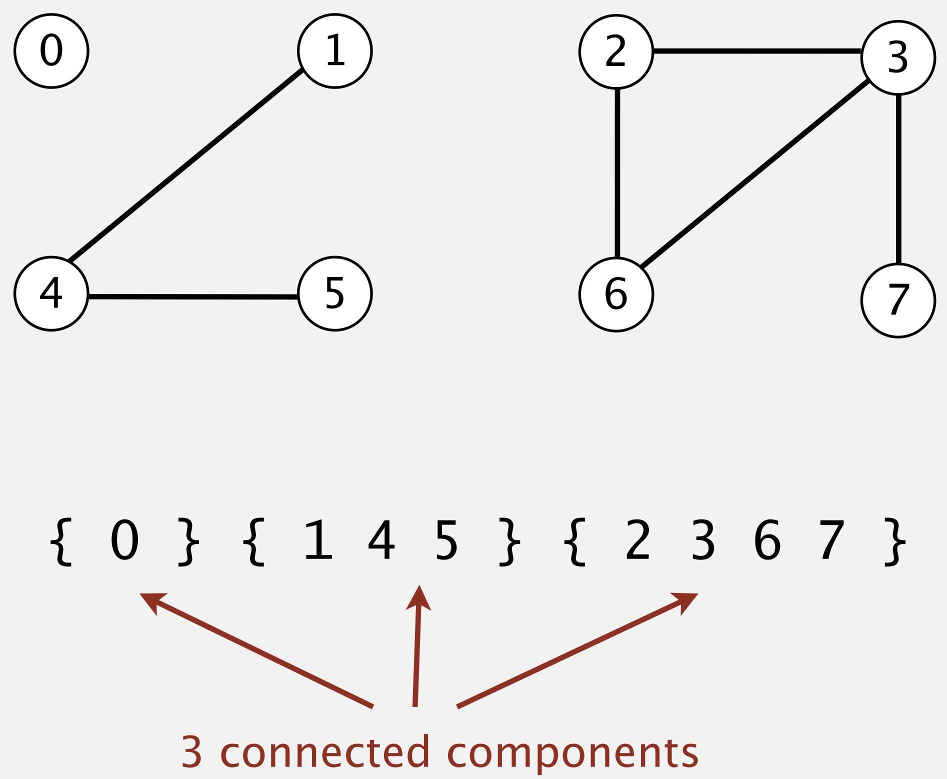
{ 0 } { 1 4 5 } { 2 3 6 7 }

3 connected components

Implementing the operations

Find query. Check if two objects are in the same component.

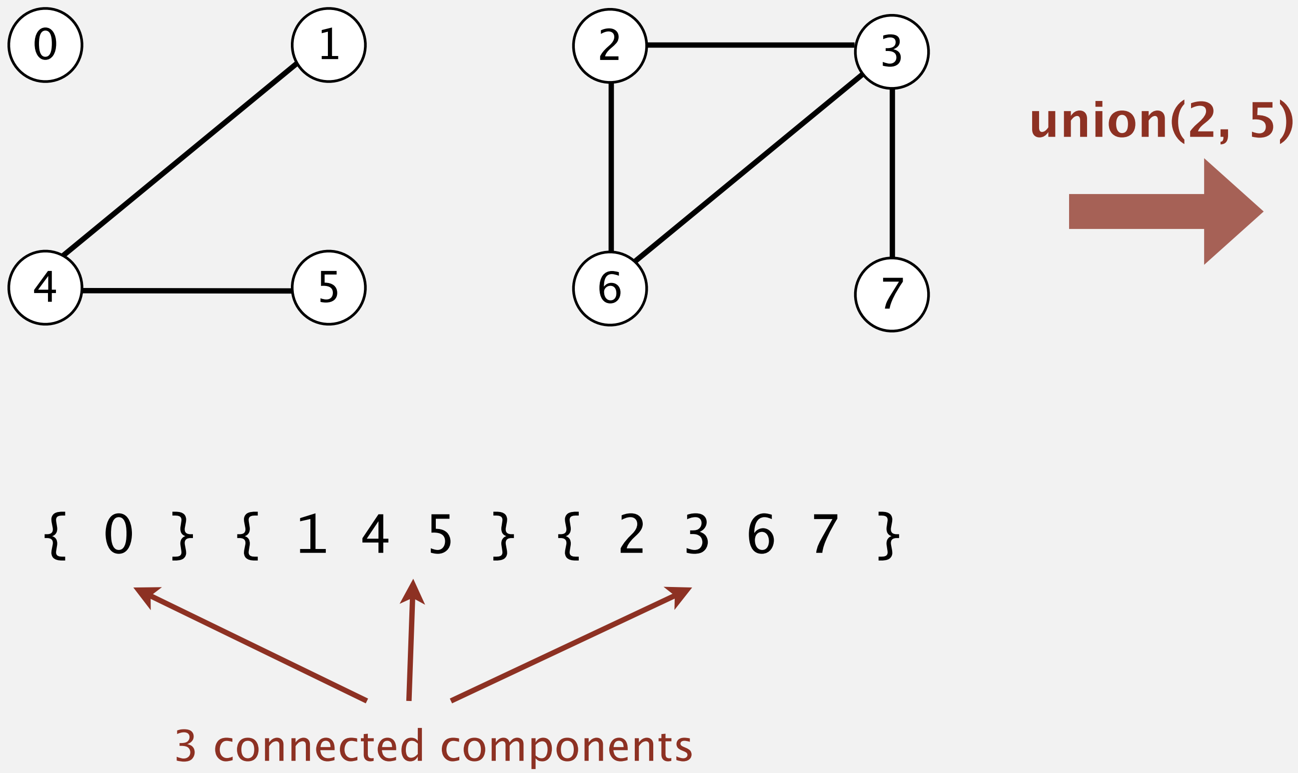
Union command. Replace components containing two objects with their union.



Implementing the operations

Find query. Check if two objects are in the same component.

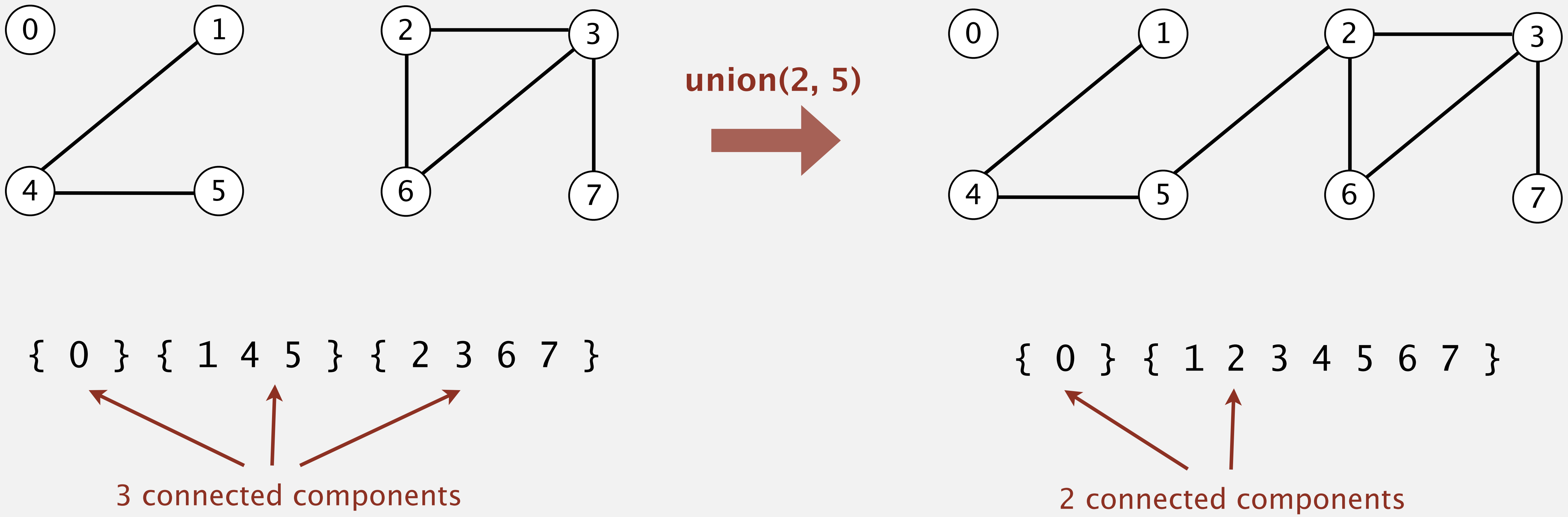
Union command. Replace components containing two objects with their union.



Implementing the operations

Find query. Check if two objects are in the same component.

Union command. Replace components containing two objects with their union.



Union-find data type (API)

Goal. Design efficient data structure for union-find.

- Number of objects N can be huge.
- Number of operations M can be huge.
- Find queries and union commands may be intermixed.

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*

Dynamic-connectivity client

- Read in number of objects N from standard input.
- Repeat:
  - read in pair of integers from standard input
  - if they are not yet connected, connect them and print out pair

```
public static void main(String[] args)
{
   int N = StdIn.readInt();
   UF uf = new UF(N);
   while (!StdIn.isEmpty())
   {
      int p = StdIn.readInt();
      int q = StdIn.readInt();
      if (!uf.connected(p, q))
      {
         uf.union(p, q);
         StdOut.println(p + " " + q);
      }
   }
}
```

```
% more tinyUF.txt
10
4 3
3 8
6 5
9 4
2 1
8 9
5 0
7 2
6 1
1 0
6 7
```

http://algs4.cs.princeton.edu

1.5 Union-Find

- *dynamic connectivity*
- *quick find*
- *quick union*
- *improvements*
- *applications*

Quick-find [eager approach]

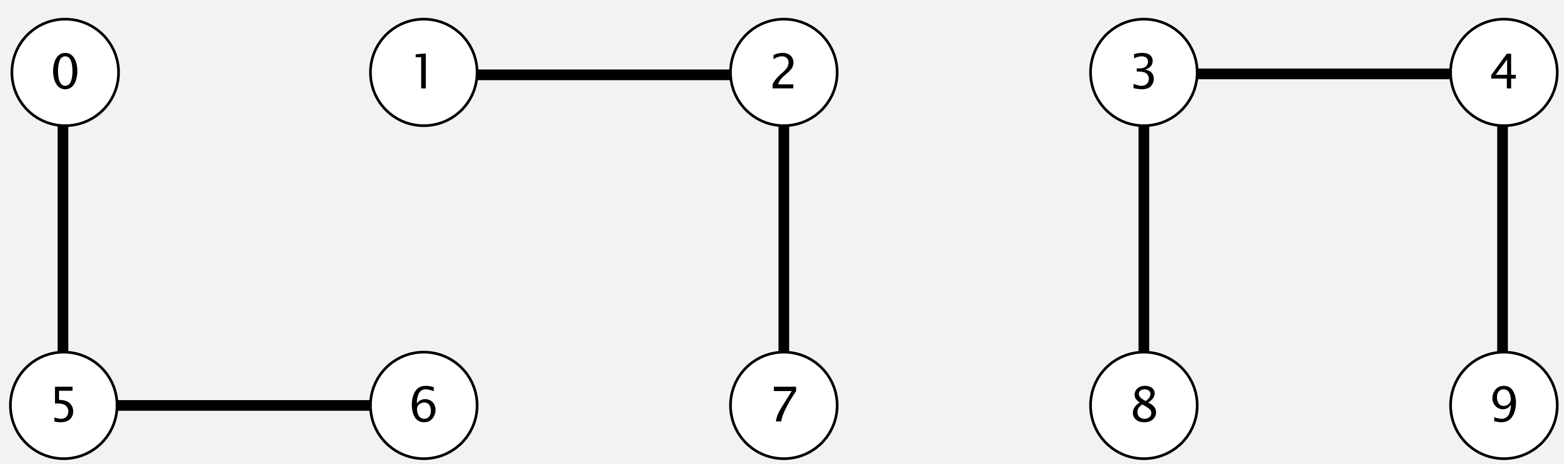
Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `p` and `q` are connected iff they have the same id.

if and only if

	0	1	2	3	4	5	6	7	8	9
`id[]`	0	1	1	8	8	0	0	1	8	8

0, 5 and 6 are connected
1, 2, and 7 are connected
3, 4, 8, and 9 are connected



Quick-find [eager approach]

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `p` and `q` are connected iff they have the same id.

	0	1	2	3	4	5	6	7	8	9
`id[]`	0	1	1	8	8	0	0	1	8	8

Find. id of `p` gives its component.

If `p` and `q` have the same id, they are connected.

`id[6] = 0`; `id[1] = 1`

6 and 1 are not connected

Quick-find [eager approach]

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `p` and `q` are connected iff they have the same id.

	0	1	2	3	4	5	6	7	8	9
`id[]`	0	1	1	8	8	0	0	1	8	8

Find. id of `p` gives its component.

If `p` and `q` have the same id, they are connected.

`id[6] = 0`; `id[1] = 1`

6 and 1 are not connected

Quick-find [eager approach]

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `p` and `q` are connected iff they have the same id.

	0	1	2	3	4	5	6	7	8	9
`id[]`	0	1	1	8	8	0	0	1	8	8

Find. id of `p` gives its component.

If `p` and `q` have the same id, they are connected.

`id[6] = 0`; `id[1] = 1`

6 and 1 are not connected

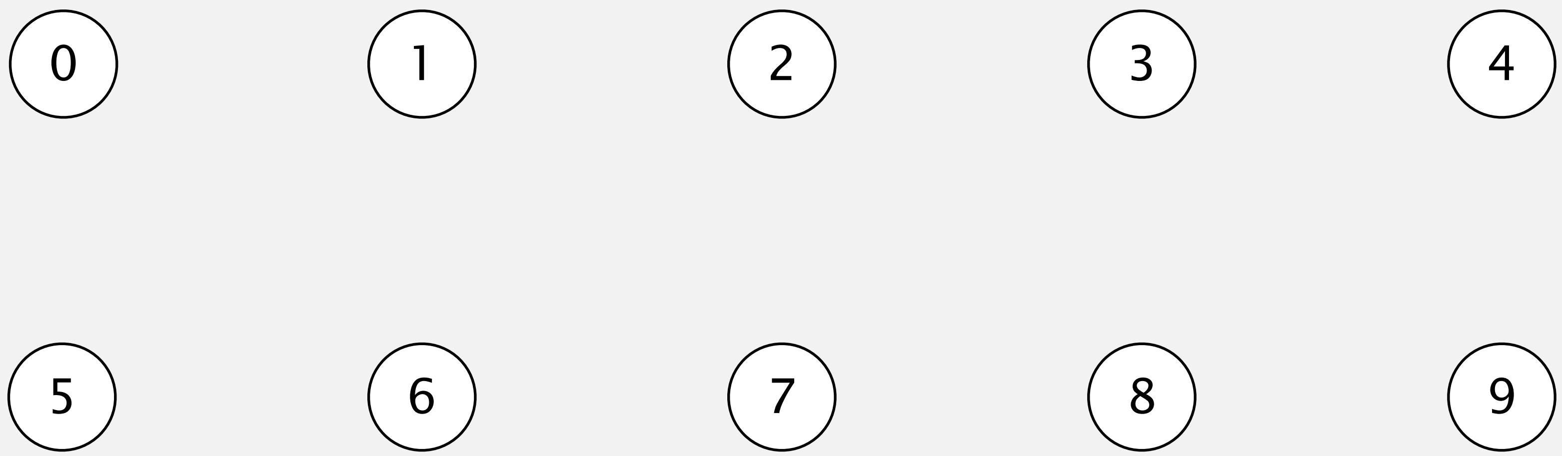
Union. To merge components containing `p` and `q`, change all entries whose id equals `id[p]` to `id[q]`.

	0	1	2	3	4	5	6	7	8	9
`id[]`	1	1	1	8	8	1	1	1	8	8

after union of 6 and 1

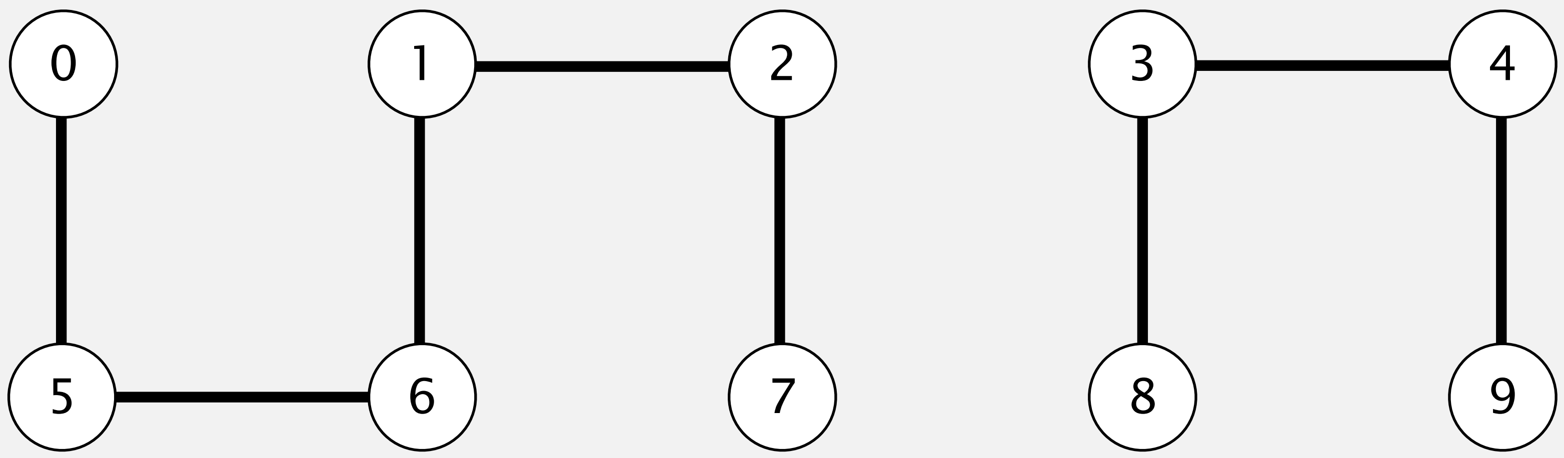
problem: many values can change

Quick-find demo



	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	3	4	5	6	7	8	9

Quick-find demo



	0	1	2	3	4	5	6	7	8	9
id[]	1	1	1	8	8	1	1	1	8	8

Quick-find: Java implementation

```
public class QuickFindUF
{
   private int[] id;

   public QuickFindUF(int N)
   {
      id = new int[N];
      for (int i = 0; i < N; i++)
         id[i] = i;
```

Quick-find: Java implementation

```
public class QuickFindUF
{
   private int[] id;

   public QuickFindUF(int N)
   {
      id = new int[N];
      for (int i = 0; i < N; i++)
         id[i] = i;
   }
```

set id of each object to itself
(N array accesses)

Quick-find: Java implementation

```
public class QuickFindUF
{
   private int[] id;

   public QuickFindUF(int N)
   {
      id = new int[N];
      for (int i = 0; i < N; i++)
         id[i] = i;
   }

   public boolean connected(int p, int q)
   {  return id[p] == id[q];  }
```

set id of each object to itself (N array accesses)

check whether p and q are in the same component (2 array accesses)

Quick-find: Java implementation

```
public class QuickFindUF
{
   private int[] id;

   public QuickFindUF(int N)
   {
      id = new int[N];
      for (int i = 0; i < N; i++)
         id[i] = i;
   }

   public boolean connected(int p, int q)
   {  return id[p] == id[q];  }

   public void union(int p, int q)
   {
      int pid = id[p];
      int qid = id[q];
      for (int i = 0; i < id.length; i++)
         if (id[i] == pid) id[i] = qid;
   }
}
```

set id of each object to itself
(N array accesses)

check whether p and q
are in the same component
(2 array accesses)

change all entries with id[p] to id[q]
(at most 2N + 2 array accesses)

Quick-find is too slow

Cost model. Number of array accesses (for read or write).

algorithm	initialize	union	find
quick-find	N	N	1

order of growth of number of array accesses

Quick-find is too slow

Cost model. Number of array accesses (for read or write).

algorithm	initialize	union	find
quick-find	N	N	1

order of growth of number of array accesses

quadratic

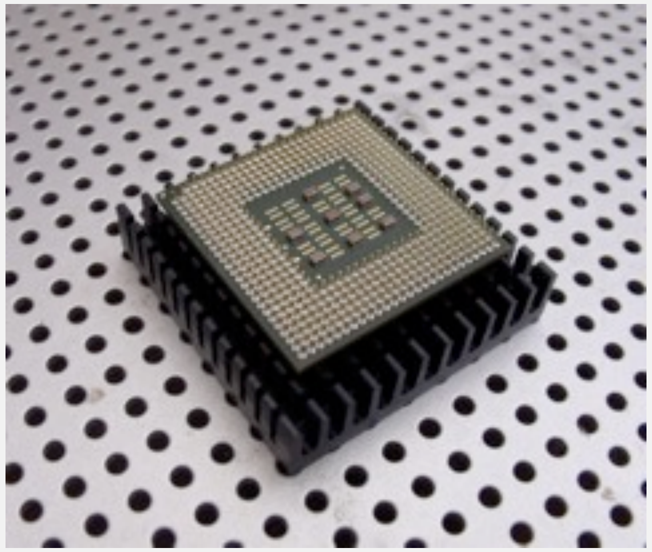
Union is too expensive. It takes N^2 array accesses to process a sequence of N union commands on N objects.

Quadratic algorithms do not scale

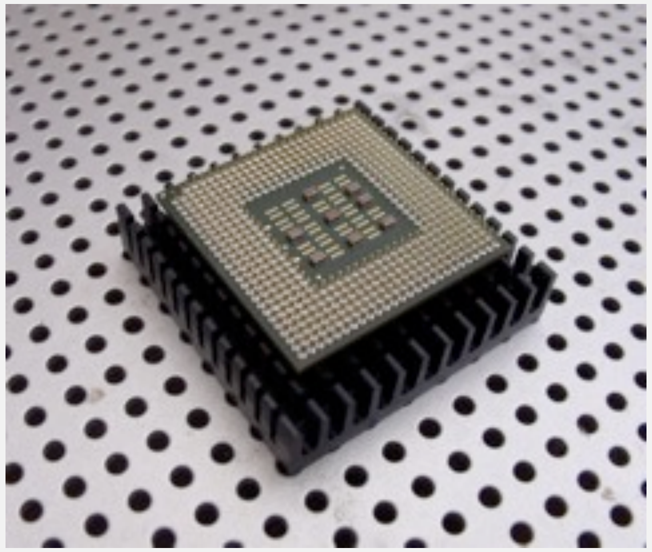
Rough standard (for now).

- 10^9 operations per second.
- 10^9 words of main memory.
- Touch all words in approximately 1 second.

a truism (roughly) since 1950!



Quadratic algorithms do not scale



Rough standard (for now).

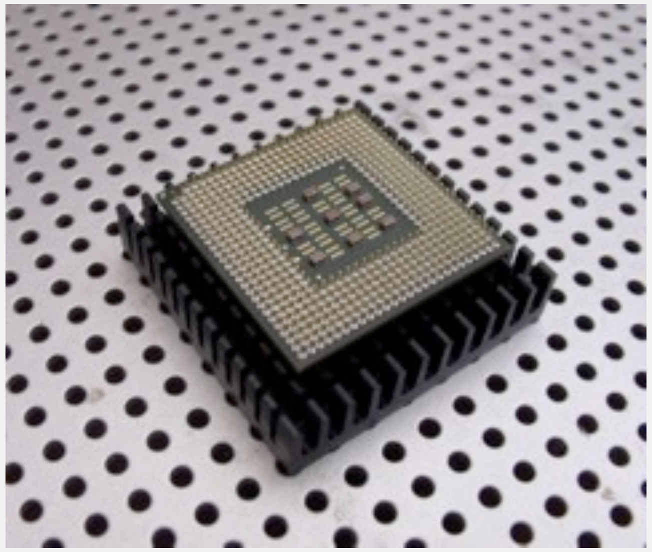
- 10^9 operations per second.
- 10^9 words of main memory.
- Touch all words in approximately 1 second.

a truism (roughly) since 1950!

Ex. Huge problem for quick-find.

- 10^9 union commands on 10^9 objects.
- Quick-find takes more than 10^{18} operations.
- 30+ years of computer time!

Quadratic algorithms do not scale



Rough standard (for now).

- 10^9 operations per second.
- 10^9 words of main memory.
- Touch all words in approximately 1 second.

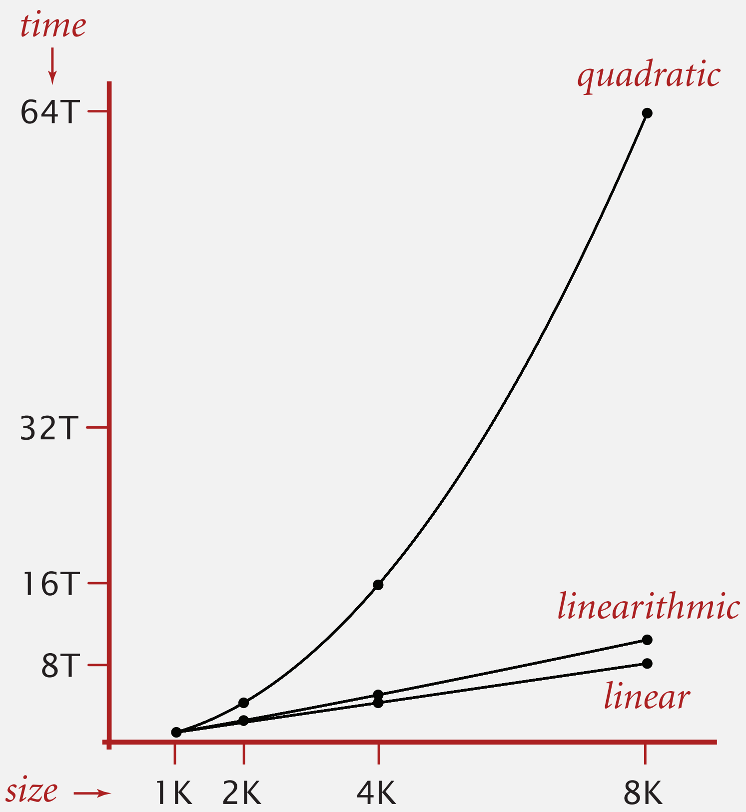
a truism (roughly) since 1950!

Ex. Huge problem for quick-find.

- 10^9 union commands on 10^9 objects.
- Quick-find takes more than 10^{18} operations.
- 30+ years of computer time!

Quadratic algorithms don't scale with technology.

- New computer may be 10x as fast.
- But, has 10x as much memory ⇒ want to solve a problem that is 10x as big.
- With quadratic algorithm, takes 10x as long!



http://algs4.cs.princeton.edu

1.5 UNION-FIND

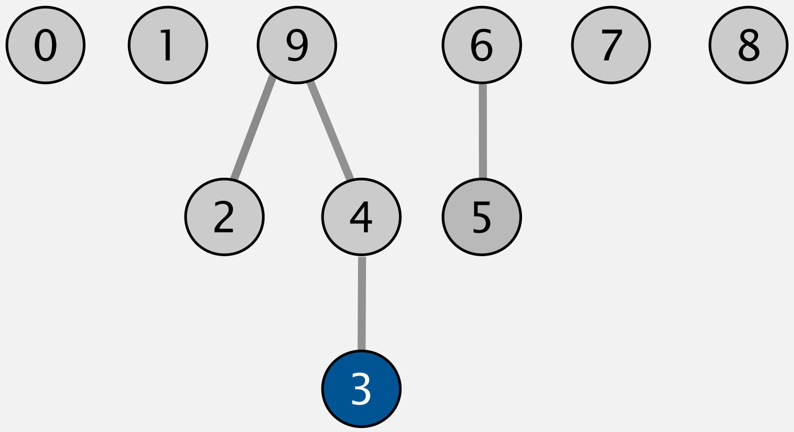
- *dynamic connectivity*
- *quick find*
- *quick union*
- *improvements*
- *applications*

Quick-union [lazy approach]

Data structure.

- Integer array id[] of size N.
- Interpretation: id[i] is parent of i.
- Root of i is id[id[id[...id[i]...]]].

keep going until it doesn't change
(algorithm ensures no cycles)



	0	1	2	3	4	5	6	7	8	9
id[]	0	1	9	4	9	6	6	7	8	9

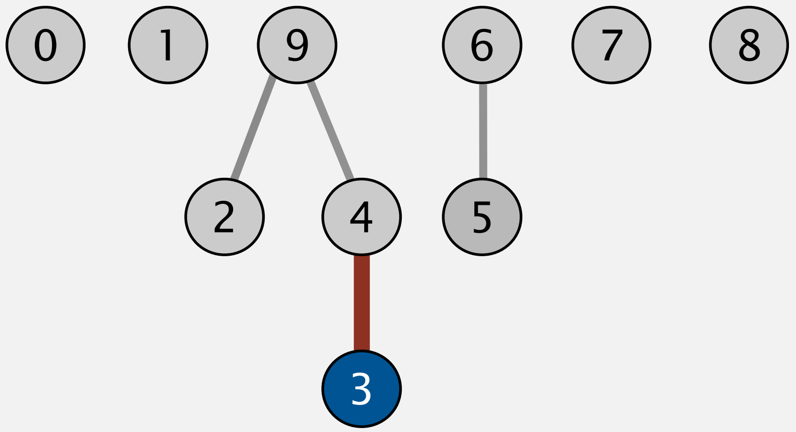
Quick-union [lazy approach]

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `id[i]` is parent of `i`.
- Root of `i` is `id[id[id[...id[i]...]]]`.

keep going until it doesn't change (algorithm ensures no cycles)

	0	1	2	3	4	5	6	7	8	9
id[]	0	1	9	4	9	6	6	7	8	9



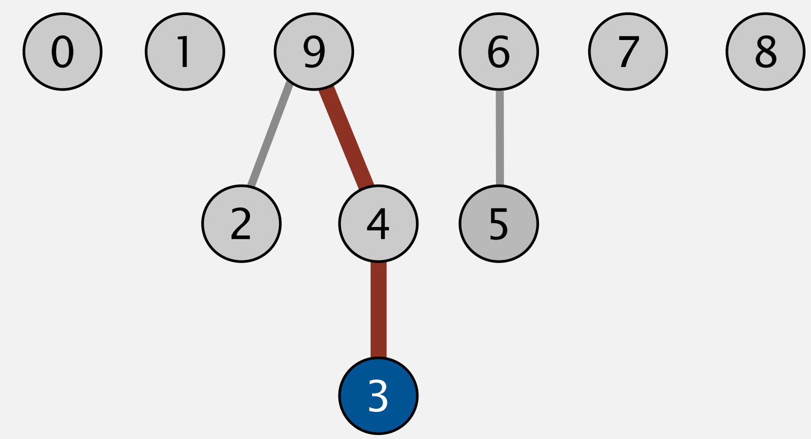
Quick-union [lazy approach]

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `id[i]` is parent of `i`.
- Root of `i` is `id[id[id[...id[i]...]]]`.

keep going until it doesn't change
(algorithm ensures no cycles)

	0	1	2	3	4	5	6	7	8	9
id[]	0	1	9	4	9	6	6	7	8	9



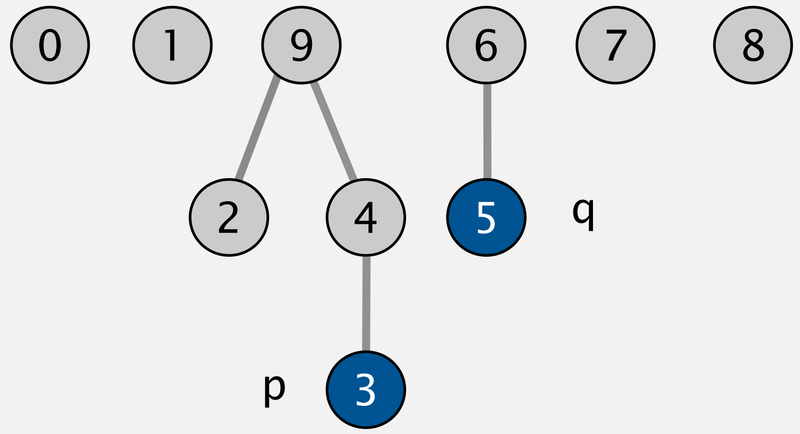
root of 3 is 9

Quick-union [lazy approach]

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `id[i]` is parent of `i`.
- Root of `i` is `id[id[id[...id[i]...]]]`.

	0	1	2	3	4	5	6	7	8	9
id[]	0	1	9	4	9	6	6	7	8	9



root of 3 is 9

root of 5 is 6

3 and 5 are not connected

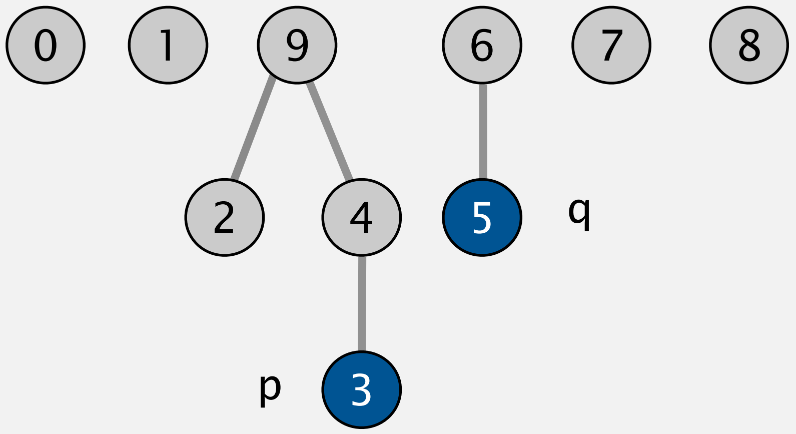
Find. Check if `p` and `q` have the same root.

Quick-union [lazy approach]

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `id[i]` is parent of `i`.
- Root of `i` is `id[id[id[...id[i]...]]]`.

	0	1	2	3	4	5	6	7	8	9
id[]	0	1	9	4	9	6	6	7	8	9



root of 3 is 9

root of 5 is 6

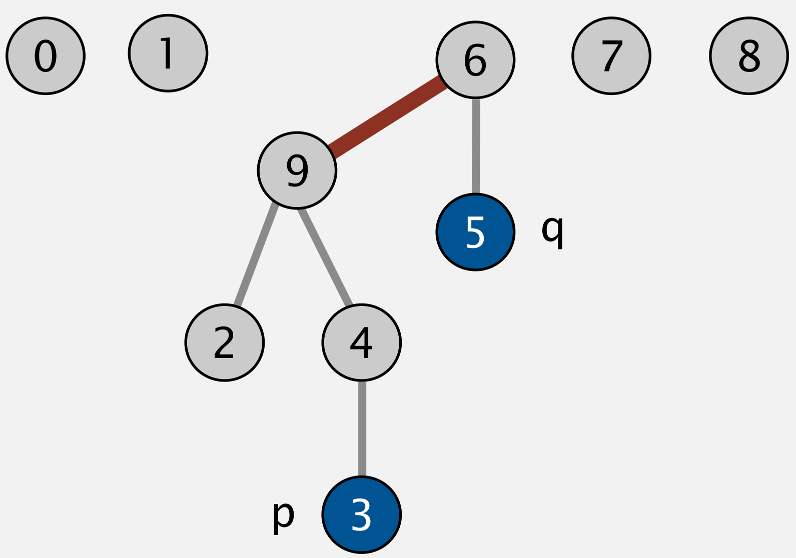
3 and 5 are not connected

Find. Check if `p` and `q` have the same root.

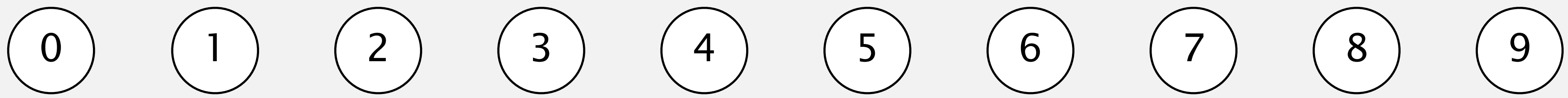
Union. To merge components containing `p` and `q`, set the id of `p`'s root to the id of `q`'s root.

	0	1	2	3	4	5	6	7	8	9
id[]	0	1	9	4	9	6	6	7	8	6

only one value changes



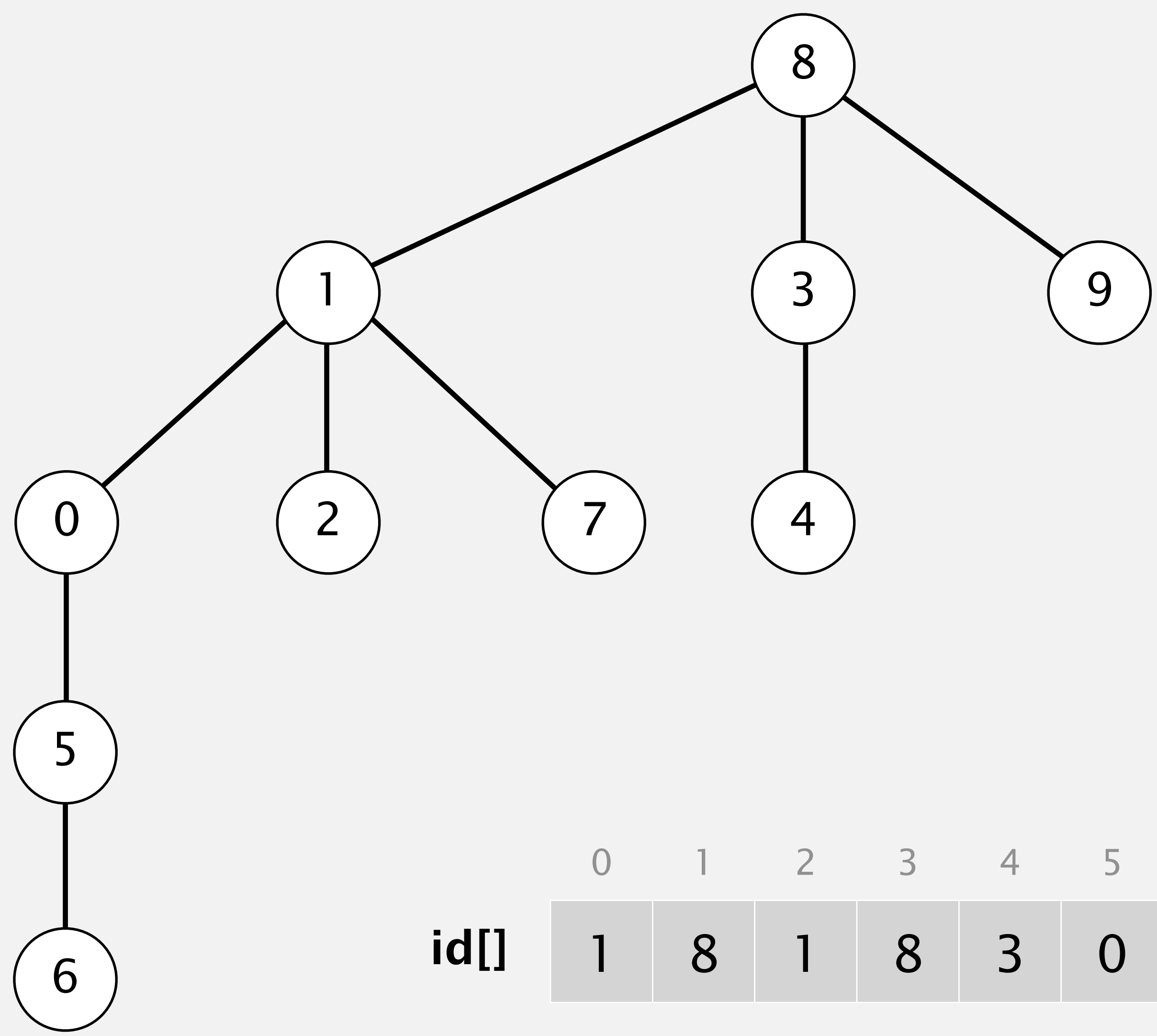
Quick-union demo



	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	3	4	5	6	7	8	9

Quick-union demo

Question: Worst case tree depth? Best Case?



	0	1	2	3	4	5	6	7	8	9
id[]	1	8	1	8	3	0	5	1	8	8

Quick-union: Java implementation

```
public class QuickUnionUF
{
   private int[] id;

   public QuickUnionUF(int N)
   {
      id = new int[N];
      for (int i = 0; i < N; i++) id[i] = i;
   }

   private int root(int i)
   {
      while (i != id[i]) i = id[i];
      return i;
   }

   public boolean connected(int p, int q)
   {
      return root(p) == root(q);
   }

   public void union(int p, int q)
   {
      int i = root(p);
      int j = root(q);
      id[i] = j;
   }
}
```

set id of each object to itself
(N array accesses)

chase parent pointers until reach root
(depth of i array accesses)

check if p and q have same root
(depth of p and q array accesses)

change root of p to point to root of q
(depth of p and q array accesses)

Quick-union is also too slow

algorithm	initialize	union	find
quick-find	N	N	1
quick-union	N	N †	N

← worst case

† includes cost of finding roots

Quick-union is also too slow

Cost model. Number of array accesses (for read or write).

algorithm	initialize	union	find
quick-find	N	N	1
quick-union	N	N †	N

← worst case

† includes cost of finding roots

Quick-union is also too slow

Cost model. Number of array accesses (for read or write).

algorithm	initialize	union	find
quick-find	N	N	1
quick-union	N	N †	N

← worst case

† includes cost of finding roots

Quick-find defect.

- Union too expensive (N array accesses).
- Trees are flat, but too expensive to keep them flat.

Quick-union defect.

- Trees can get tall.
- Find too expensive (could be N array accesses).

http://algs4.cs.princeton.edu

1.5 UNION-FIND

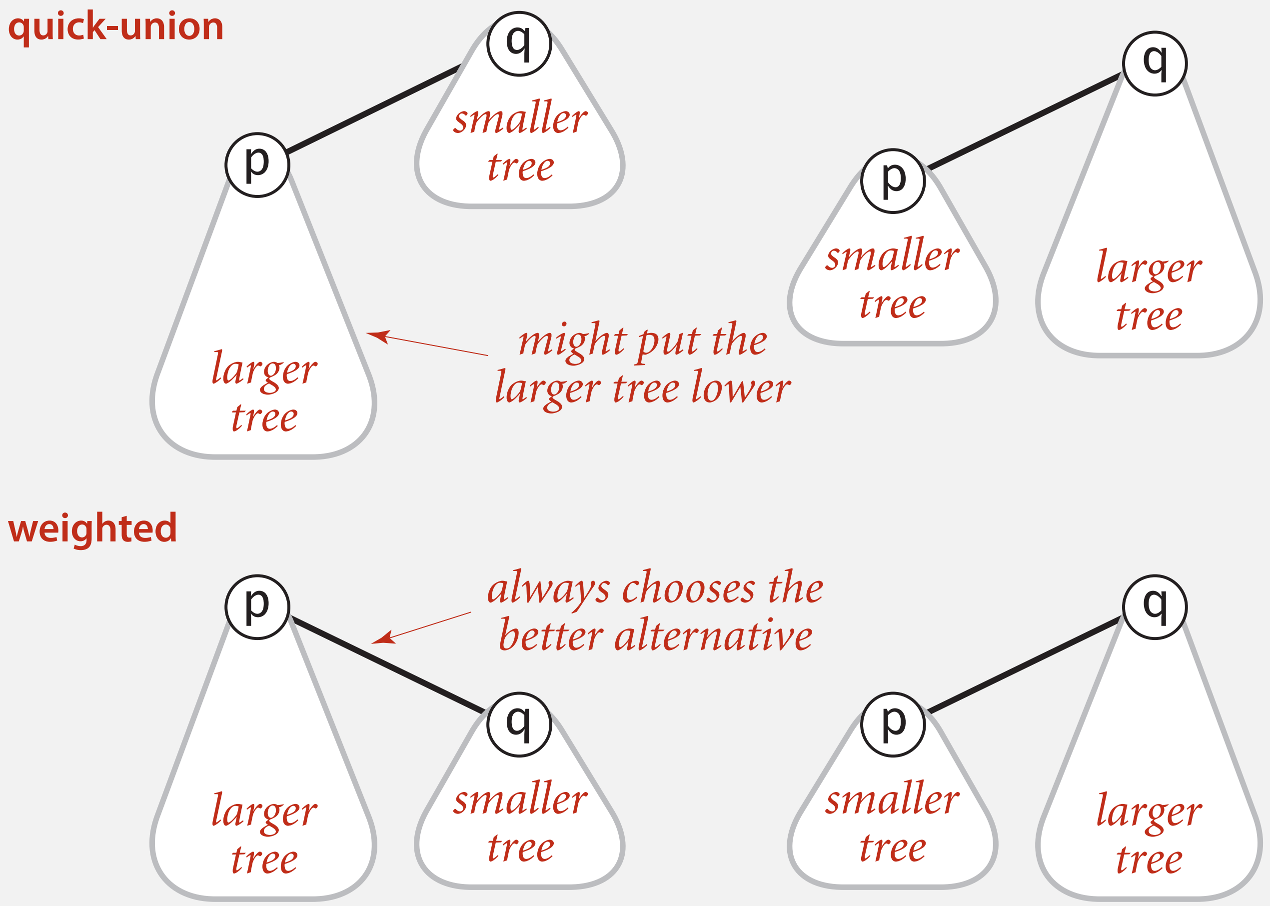
- dynamic connectivity
- quick find
- quick union
- *improvements*
- applications

Improvement 1: weighting

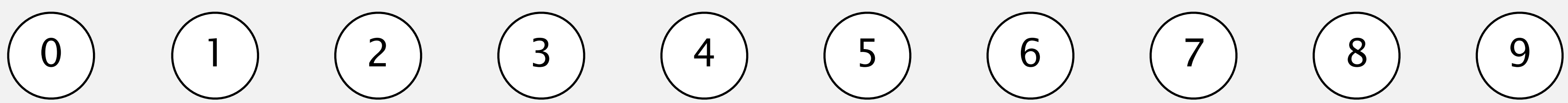
Weighted quick-union.

- Modify quick-union to avoid tall trees.
- Keep track of size of each tree (number of objects).
- Balance by linking root of smaller tree to root of larger tree.

reasonable alternatives:
union by height or "rank"

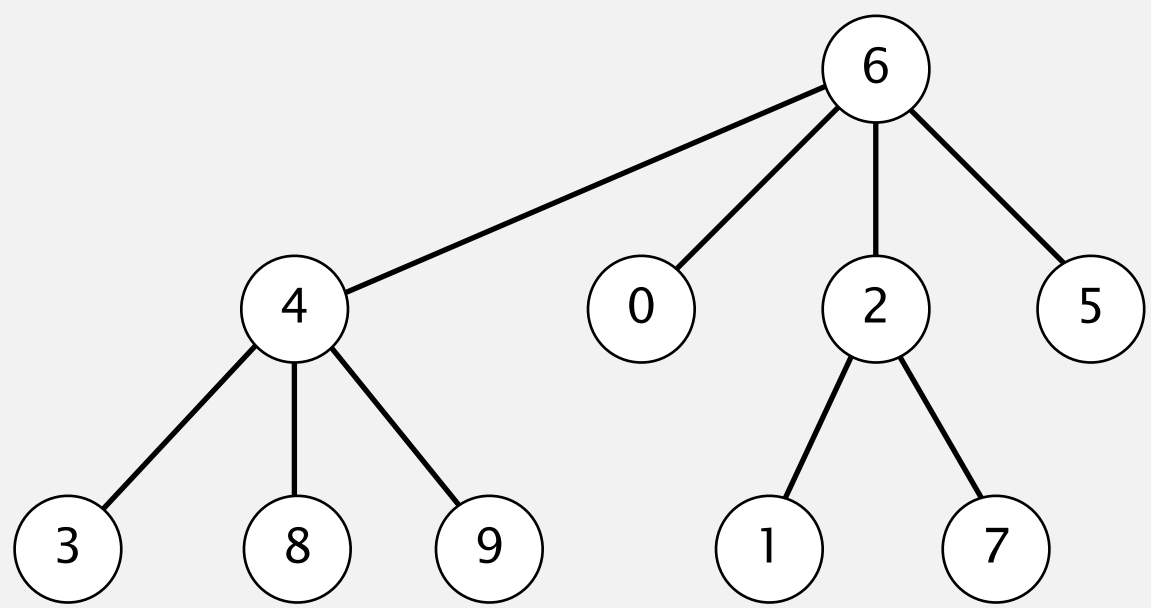


Weighted quick-union demo



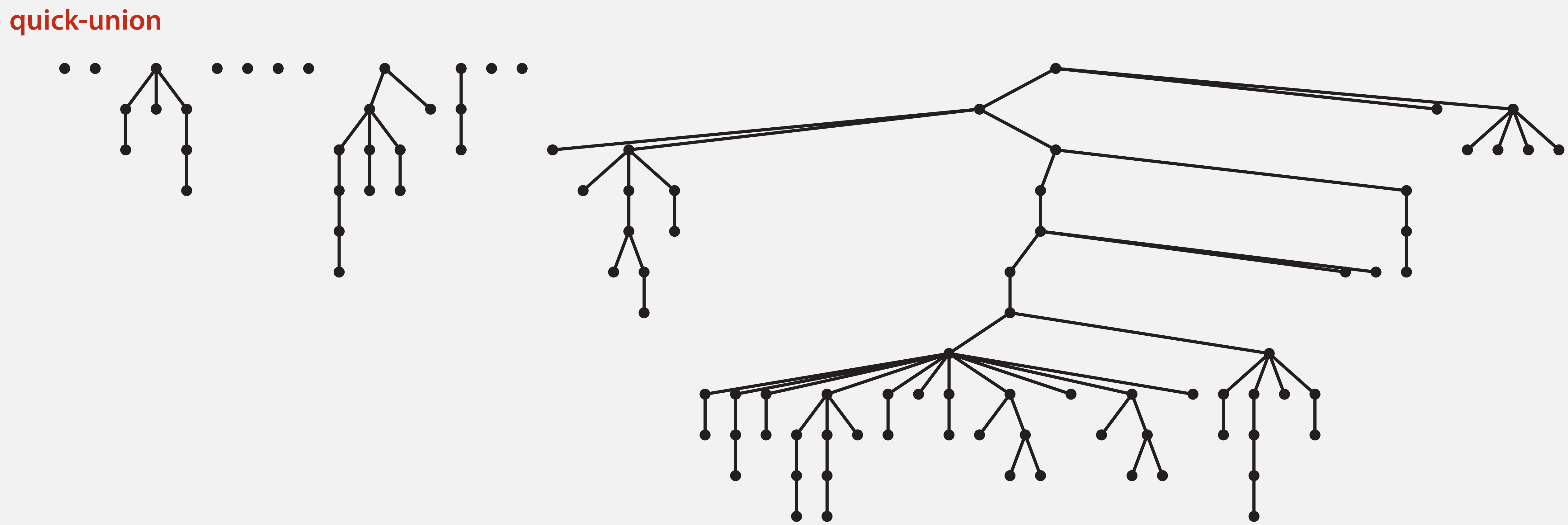
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	3	4	5	6	7	8	9

Weighted quick-union demo

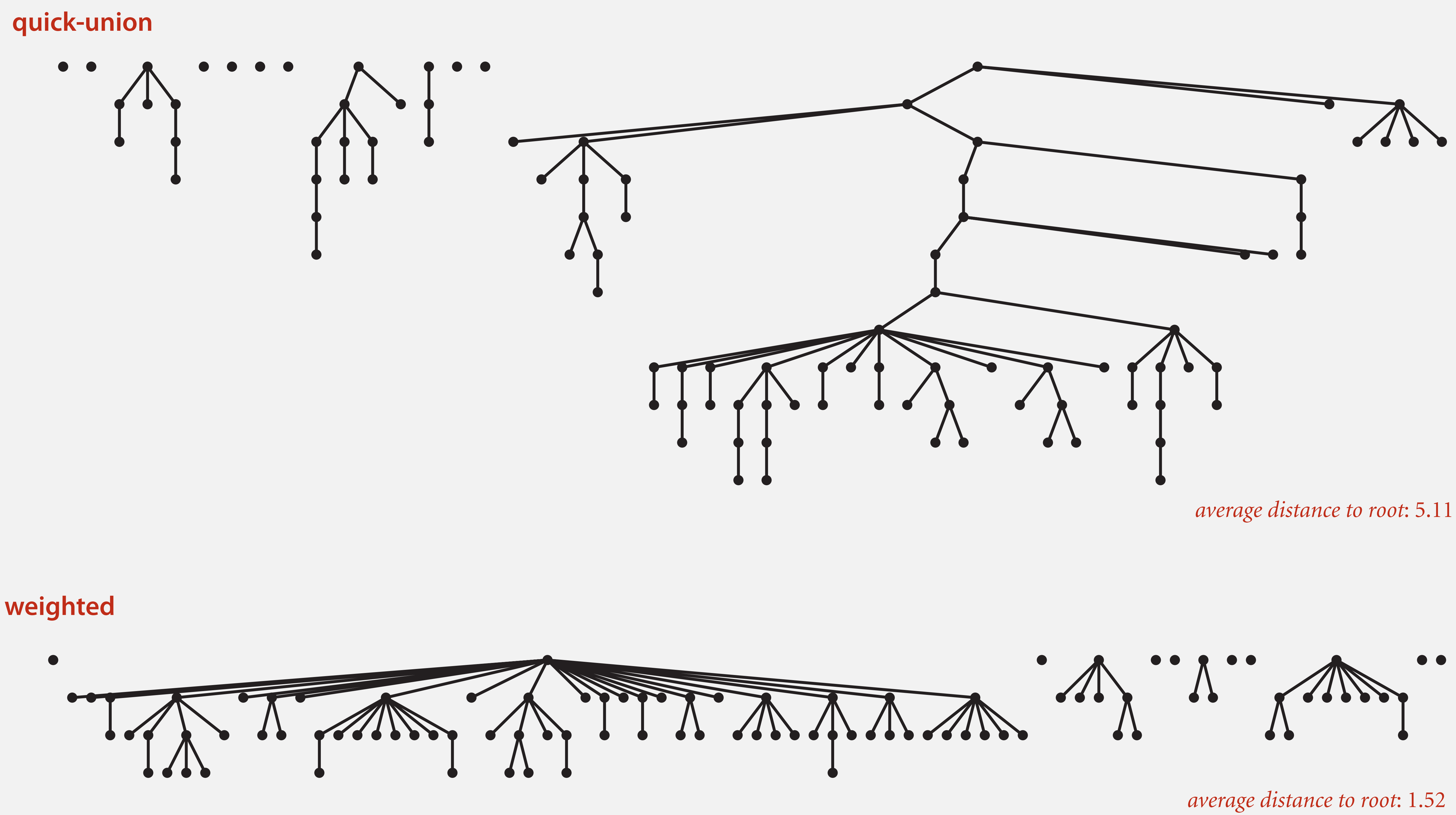


	0	1	2	3	4	5	6	7	8	9
id[]	6	2	6	4	6	6	6	2	4	4

Quick-union and weighted quick-union example



Quick-union and weighted quick-union example



Quick-union and weighted quick-union (100 sites, 88 `union()` operations)

Weighted quick-union: Java implementation

Data structure. Same as quick-union, but maintain extra array `sz[i]` to count number of objects in the tree rooted at `i`.

Weighted quick-union: Java implementation

Data structure. Same as quick-union, but maintain extra array `sz[i]` to count number of objects in the tree rooted at `i`.

Find. Identical to quick-union.

```
return root(p) == root(q);
```

Weighted quick-union: Java implementation

Data structure. Same as quick-union, but maintain extra array `sz[i]` to count number of objects in the tree rooted at `i`.

Find. Identical to quick-union.

```
return root(p) == root(q);
```

Union. Modify quick-union to:

- Link root of smaller tree to root of larger tree.
- Update the `sz[]` array.

```
int i = root(p);
int j = root(q);
if  (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
else                { id[j] = i; sz[i] += sz[j]; }
```

Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p and q.
- Union: takes constant time, given roots.

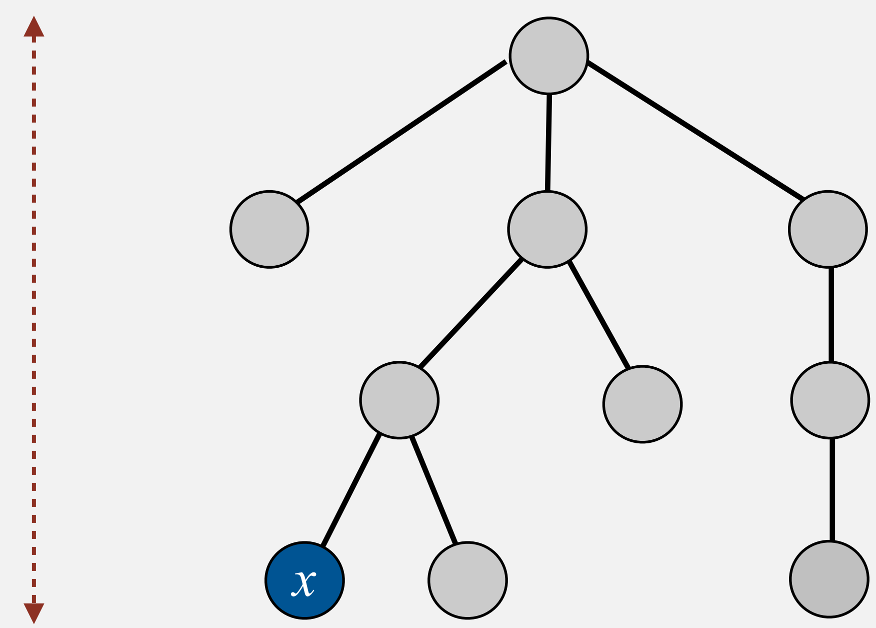
Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p and q.
- Union: takes constant time, given roots.

lg = base-2 logarithm

Proposition. Depth of any node x is at most $\lg N$.



N = 10
depth(x) = 3 ≤ lg N

Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p and q.
- Union: takes constant time, given roots.

lg = base-2 logarithm

Proposition. Depth of any node x is at most $\lg N$.

Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p and q.
- Union: takes constant time, given roots.

lg = base-2 logarithm

Proposition. Depth of any node x is at most $\lg N$.

Pf. When does depth of x increase?

Weighted quick-union analysis

Running time.

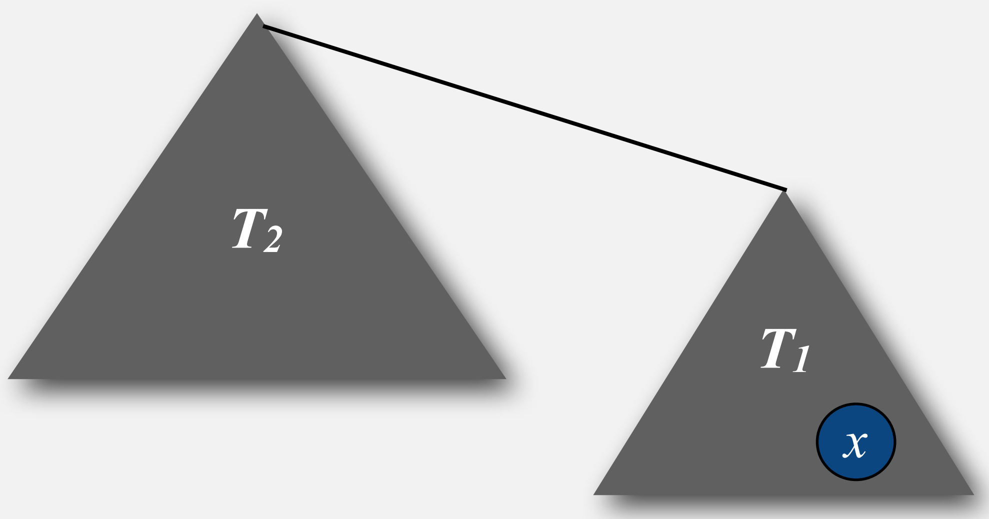
- Find: takes time proportional to depth of p and q.
- Union: takes constant time, given roots.

lg = base-2 logarithm

Proposition. Depth of any node x is at most $\lg N$.

Pf. When does depth of x increase?

Increases by 1 when tree T_1 containing x is merged into another tree T_2.



Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p and q.
- Union: takes constant time, given roots.

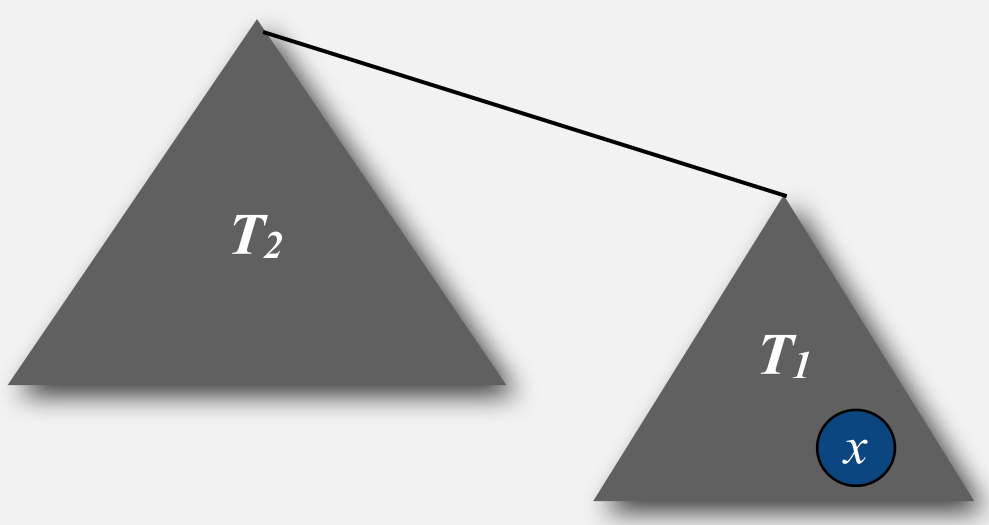
lg = base-2 logarithm

Proposition. Depth of any node x is at most $\lg N$.

Pf. When does depth of x increase?

Increases by 1 when tree T_1 containing x is merged into another tree T_2.

- The size of the tree containing x at least doubles since $|T_2| \geq |T_1|$.



Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p and q.
- Union: takes constant time, given roots.

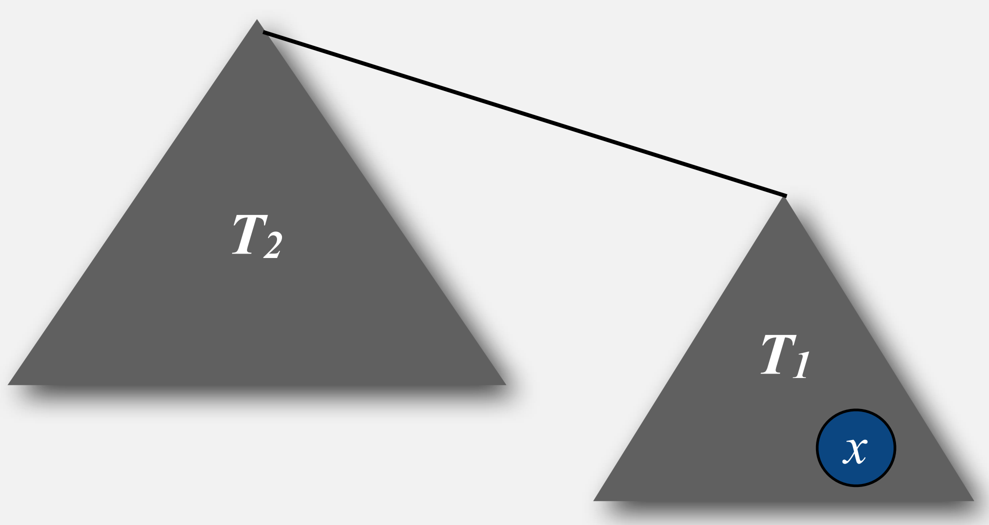
lg = base-2 logarithm

Proposition. Depth of any node x is at most $\lg N$.

Pf. When does depth of x increase?

Increases by 1 when tree T_1 containing x is merged into another tree T_2.

- The size of the tree containing x at least doubles since $|T_2| \geq |T_1|$.
- Size of tree containing x can double at most $\lg N$ times. Why?



Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p and q.
- Union: takes constant time, given roots.

Proposition. Depth of any node x is at most $\lg N$.

algorithm	initialize	union	connected
quick-find	N	N	1
quick-union	N	N †	N
weighted QU	N	lg N †	lg N

† includes cost of finding roots

Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p and q.
- Union: takes constant time, given roots.

Proposition. Depth of any node x is at most $\lg N$.

algorithm	initialize	union	connected
quick-find	N	N	1
quick-union	N	N †	N
weighted QU	N	lg N †	lg N

† includes cost of finding roots

Q. Stop at guaranteed acceptable performance?

Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p and q.
- Union: takes constant time, given roots.

Proposition. Depth of any node x is at most $\lg N$.

algorithm	initialize	union	connected
quick-find	N	N	1
quick-union	N	N †	N
weighted QU	N	lg N †	lg N

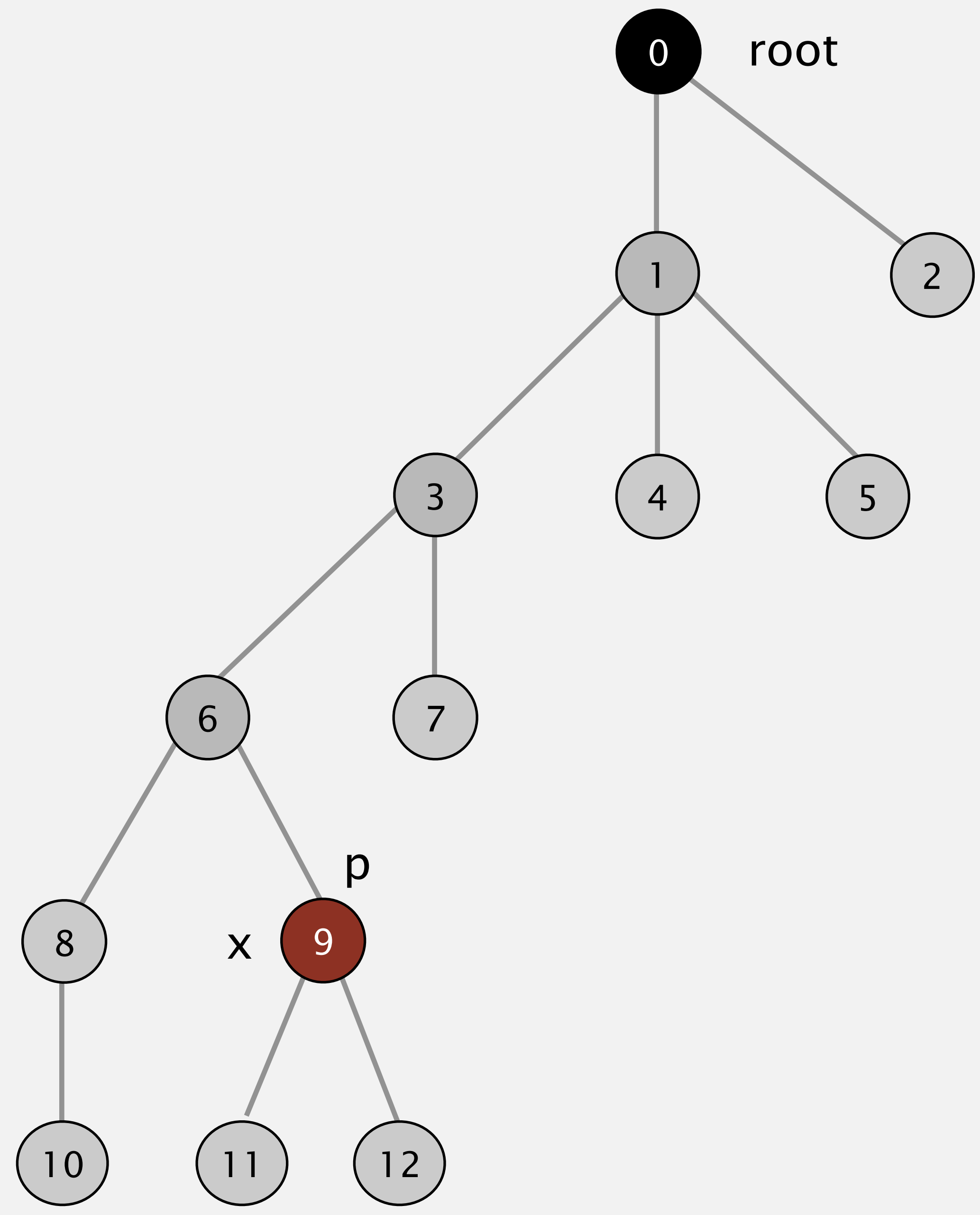
† includes cost of finding roots

Q. Stop at guaranteed acceptable performance?

A. No, easy to improve further.

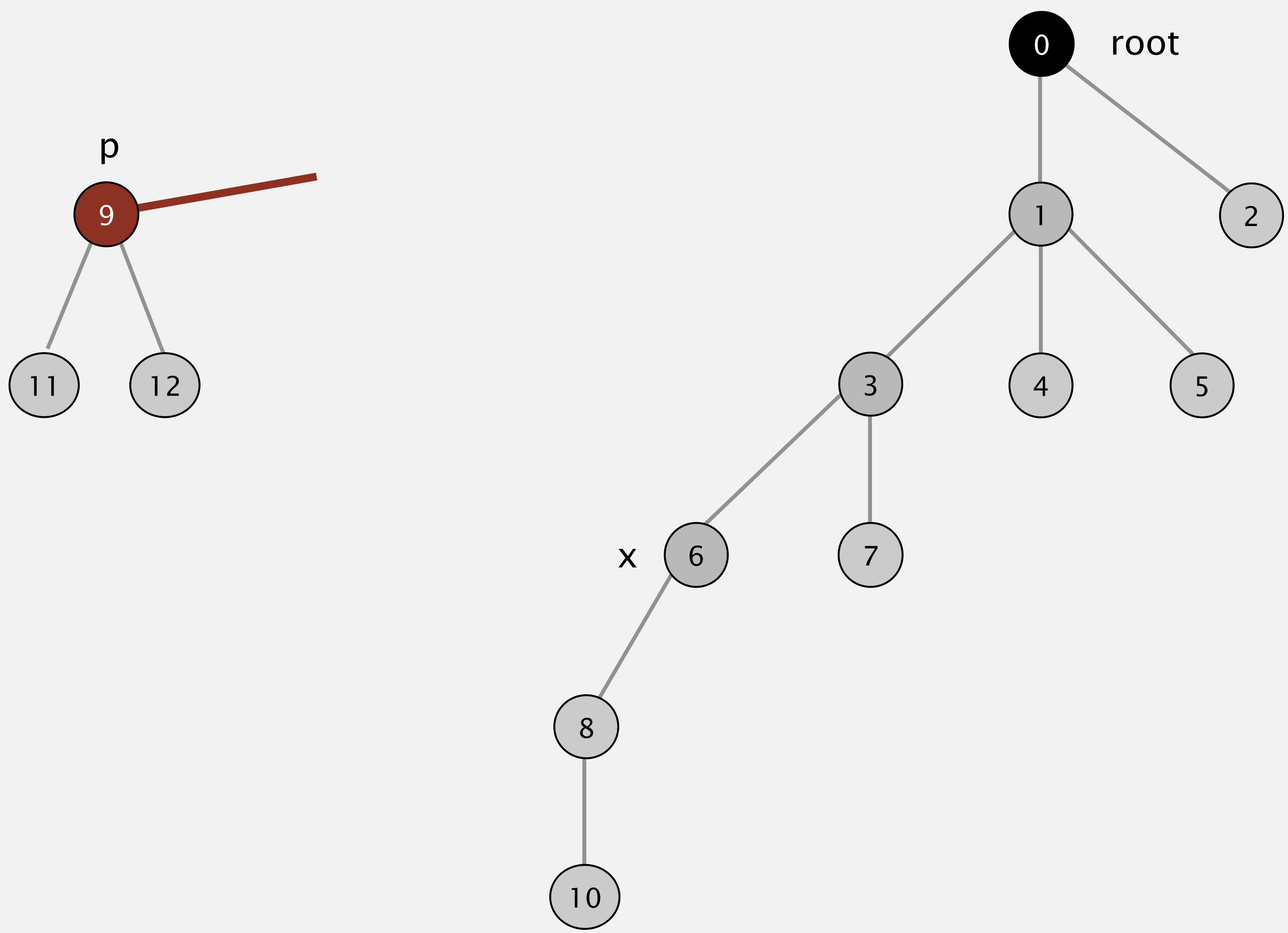
Improvement 2: path compression

Quick union with path compression. Just after computing the root of p, set the id of each examined node to point to that root.



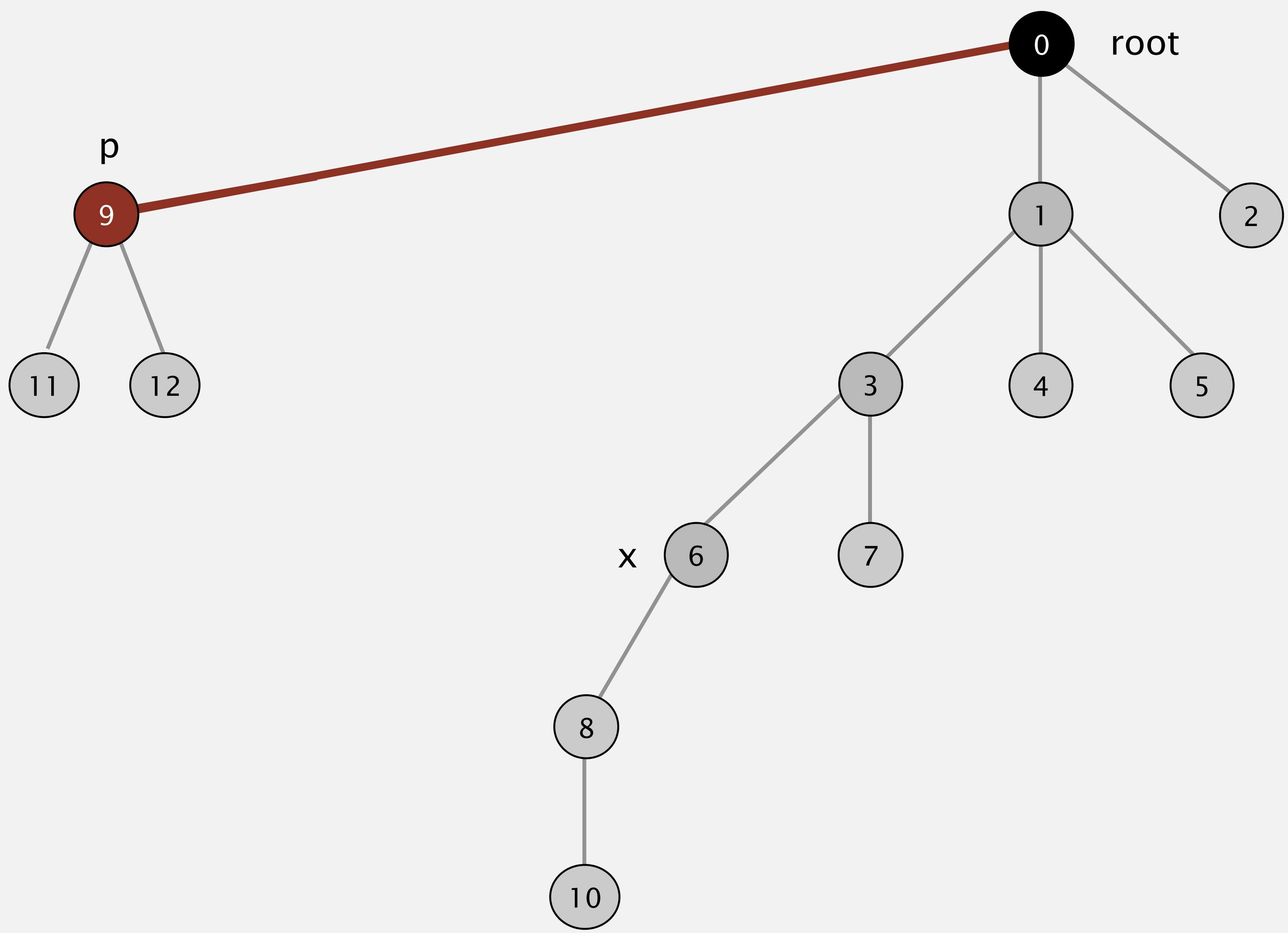
Improvement 2: path compression

Quick union with path compression. Just after computing the root of p, set the id of each examined node to point to that root.



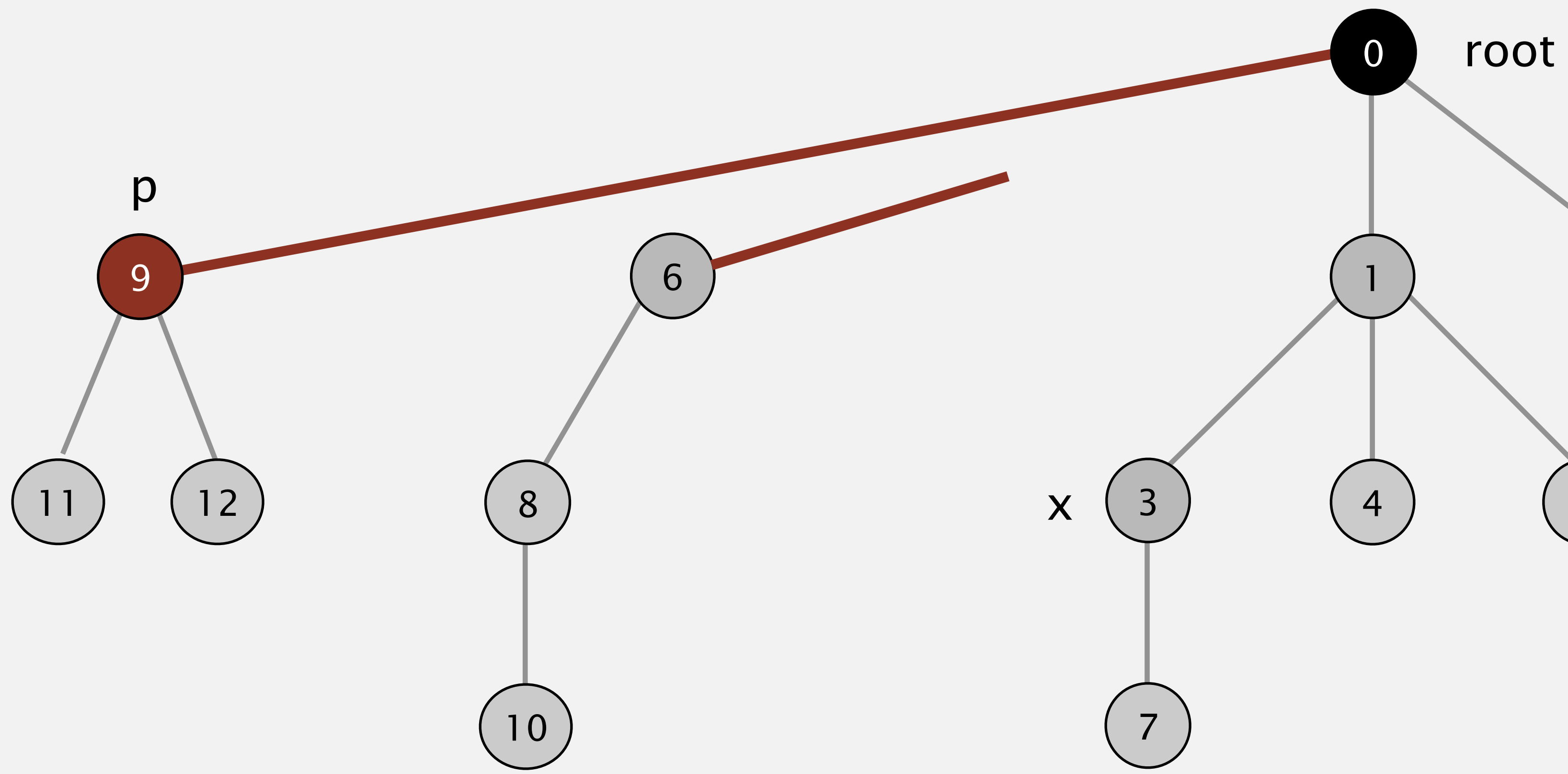
Improvement 2: path compression

Quick union with path compression. Just after computing the root of p, set the id of each examined node to point to that root.



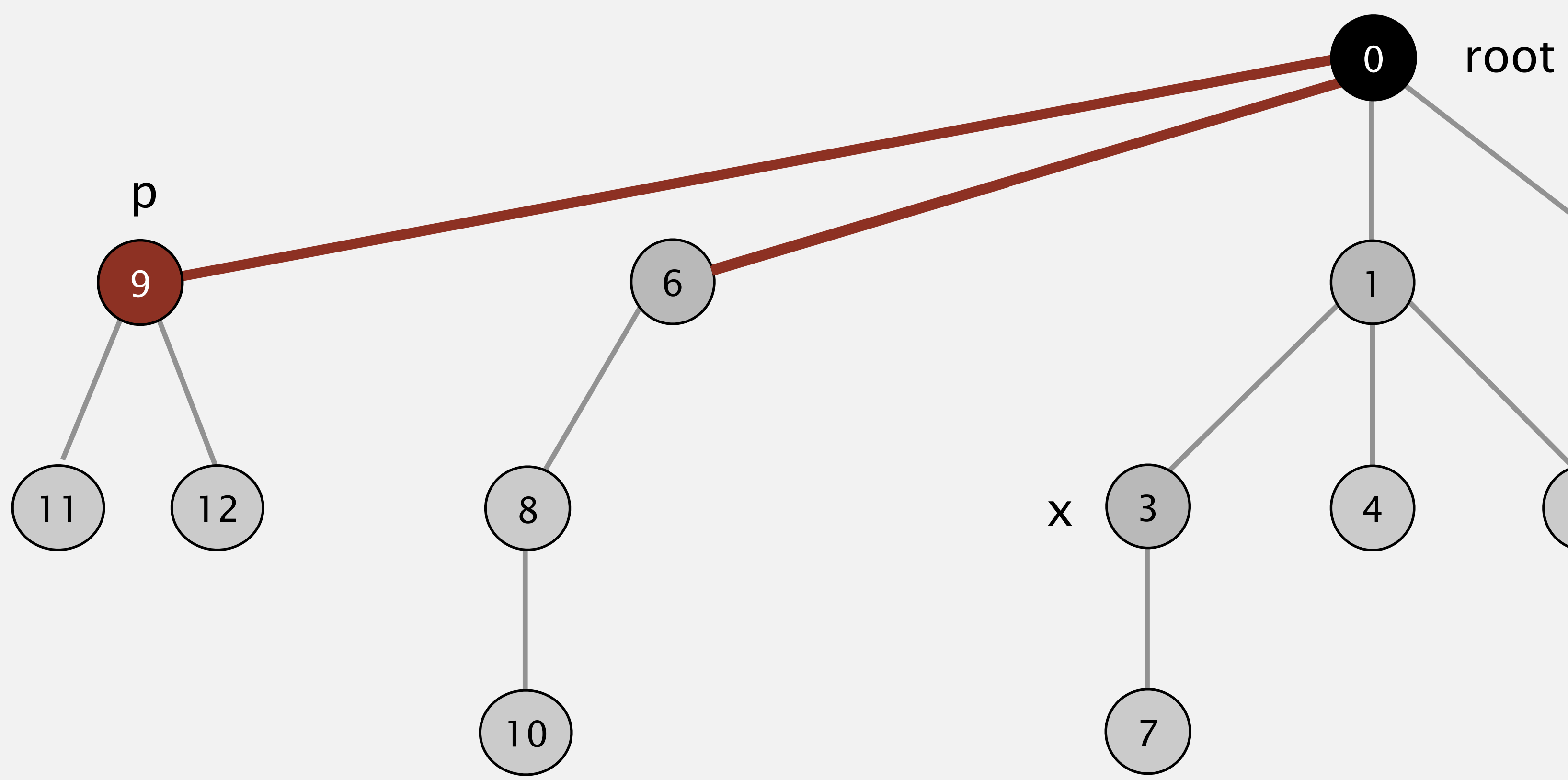
Improvement 2: path compression

Quick union with path compression. Just after computing the root of p, set the id of each examined node to point to that root.



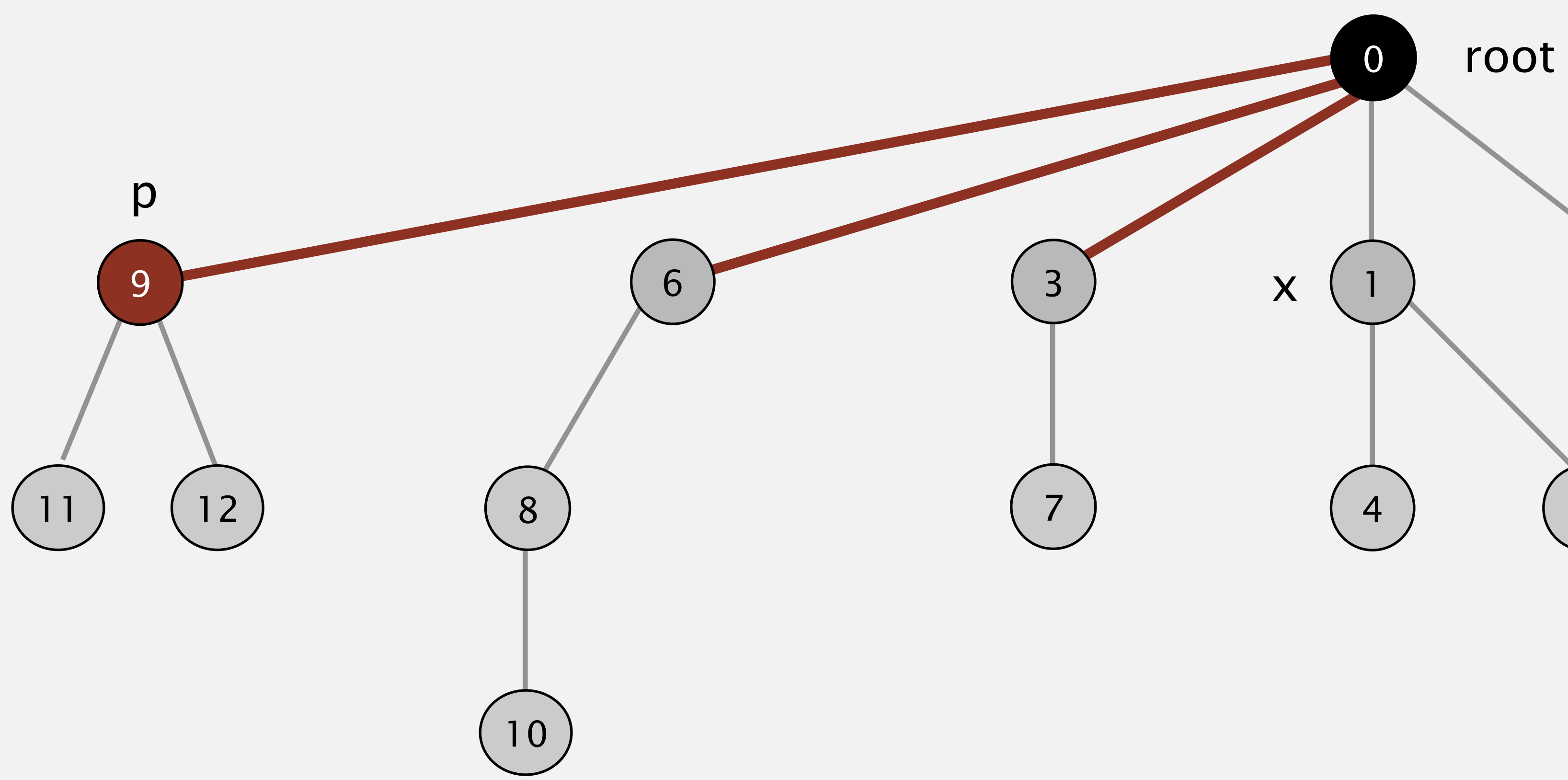
Improvement 2: path compression

Quick union with path compression. Just after computing the root of p, set the id of each examined node to point to that root.



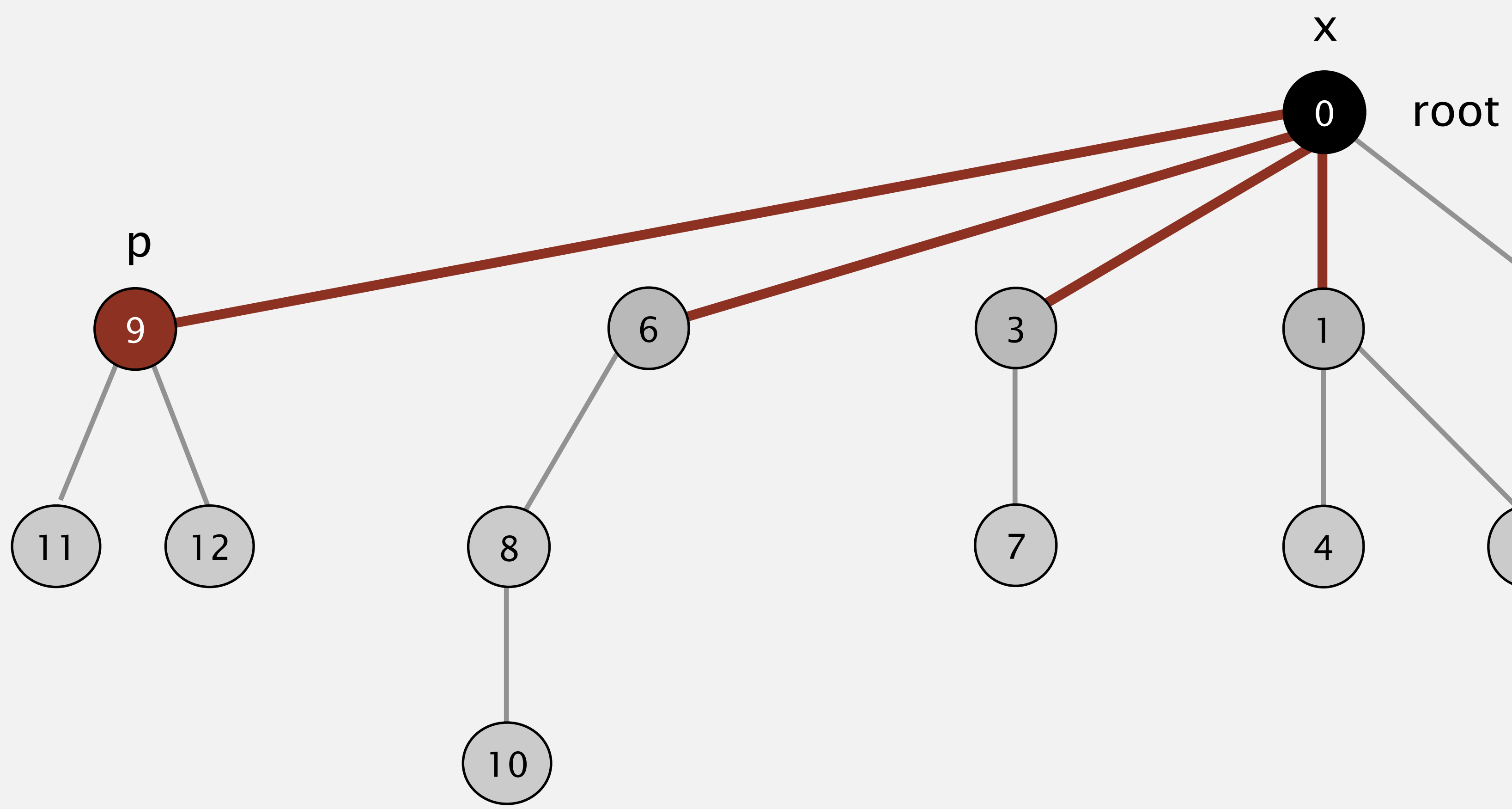
Improvement 2: path compression

Quick union with path compression. Just after computing the root of p, set the id of each examined node to point to that root.



Improvement 2: path compression

Quick union with path compression. Just after computing the root of p, set the id[] of each examined node to point to that root.



Path compression: Java implementation

Two-pass implementation: add second loop to `root()` to set the `id[]` of each examined node to the root.

Path compression: Java implementation

Two-pass implementation: add second loop to `root()` to set the `id[]` of each examined node to the root.

Simpler one-pass variant: Make every other node in path point to its grandparent (thereby halving path length).

```
private int root(int i)
{
   while (i != id[i])
   {
      id[i] = id[id[i]];
      i = id[i];
   }
   return i;
}
```

only one extra line of code !

Path compression: Java implementation

Two-pass implementation: add second loop to `root()` to set the `id[]` of each examined node to the root.

Simpler one-pass variant: Make every other node in path point to its grandparent (thereby halving path length).

```
private int root(int i)
{
   while (i != id[i])
   {
      id[i] = id[id[i]];
      i = id[i];
   }
   return i;
}
```

only one extra line of code !

In practice. No reason not to! Keeps tree almost completely flat.

Weighted quick-union with path compression: amortized analysis

Proposition. [Hopcroft-Ulman, Tarjan] Starting from an empty data structure, any sequence of M union–find ops on N objects makes $\leq c\,(N + M \lg^* N)$ array accesses.

- Analysis can be improved to $N + M\,\alpha(M, N)$.
- Simple algorithm with fascinating mathematics.

N	lg* N
1	0
2	1
4	2
16	3
65536	4
2^{65536}	5

iterate log function

Weighted quick-union with path compression: amortized analysis

Proposition. [Hopcroft-Ulman, Tarjan] Starting from an empty data structure, any sequence of M union–find ops on N objects makes $\leq c\,(N + M \lg^* N)$ array accesses.

- Analysis can be improved to $N + M\,\alpha(M, N)$.
- Simple algorithm with fascinating mathematics.

N	lg* N
1	0
2	1
4	2
16	3
65536	4
2^{65536}	5

iterate log function

Weighted quick-union with path compression: amortized analysis

Proposition. [Hopcroft-Ulman, Tarjan] Starting from an empty data structure, any sequence of M union–find ops on N objects makes $\leq c(N + M \lg^* N)$ array accesses.

- Analysis can be improved to $N + M\,\alpha(M, N)$.
- Simple algorithm with fascinating mathematics.

N	lg* N
1	0
2	1
4	2
16	3
65536	4
2^{65536}	5

iterate log function

Linear-time algorithm for M union-find ops on N objects?

- Cost within constant factor of reading in the data.
- In theory, WQUPC is not quite linear.
- In practice, WQUPC is linear.

Weighted quick-union with path compression: amortized analysis

Proposition. [Hopcroft-Ulman, Tarjan] Starting from an empty data structure, any sequence of M union–find ops on N objects makes ≤ $c\,(N + M \lg^* N)$ array accesses.

- Analysis can be improved to $N + M\,\alpha(M, N)$.
- Simple algorithm with fascinating mathematics.

N	lg* N
1	0
2	1
4	2
16	3
65536	4
2^{65536}	5

iterate log function

Linear-time algorithm for M union-find ops on N objects?

- Cost within constant factor of reading in the data.
- In theory, WQUPC is not quite linear.
- In practice, WQUPC is linear.

Amazing fact. [Fredman-Saks] No linear-time algorithm exists.

in "cell-probe" model of computation

Summary

Key point. Weighted quick union (with path compression) makes it possible to solve problems that could not otherwise be addressed.

algorithm	worst-case time
quick-find	M N
quick-union	M N
weighted QU	N + M log N
QU + path compression	N + M log N
weighted QU + path compression	N + M lg* N

order of growth for M union-find operations on a set of N objects

Summary

Key point. Weighted quick union (with path compression) makes it possible to solve problems that could not otherwise be addressed.

algorithm	worst-case time
quick-find	M N
quick-union	M N
weighted QU	N + M log N
QU + path compression	N + M log N
weighted QU + path compression	N + M lg* N

order of growth for M union-find operations on a set of N objects

Ex. [10^9 unions and finds with 10^9 objects]

- WQUPC reduces time from 30 years to 6 seconds.
- Supercomputer won't help much; good algorithm enables solution.

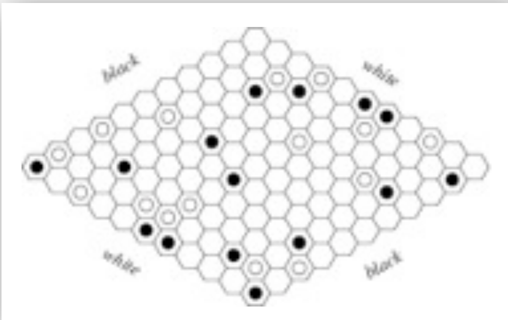
http://algs4.cs.princeton.edu

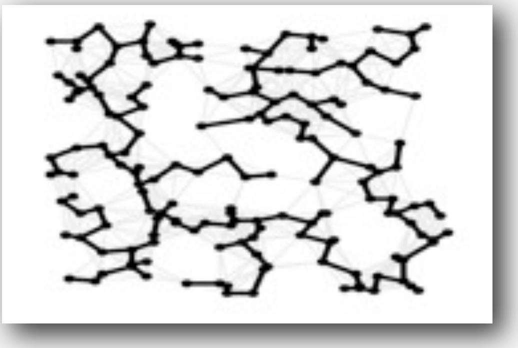
1.5 Union-Find

- *dynamic connectivity*
- *quick find*
- *quick union*
- *improvements*
- *applications*

Union-find applications

- Percolation.
- Games (Go, Hex).
- ✓ Dynamic connectivity.
- Least common ancestor.
- Equivalence of finite state automata.
- Hoshen-Kopelman algorithm in physics.
- Hinley-Milner polymorphic type inference.
- Kruskal's minimum spanning tree algorithm.
- Compiling equivalence statements in Fortran.
- Morphological attribute openings and closings.
- Matlab's `bwlabel()` function in image processing.

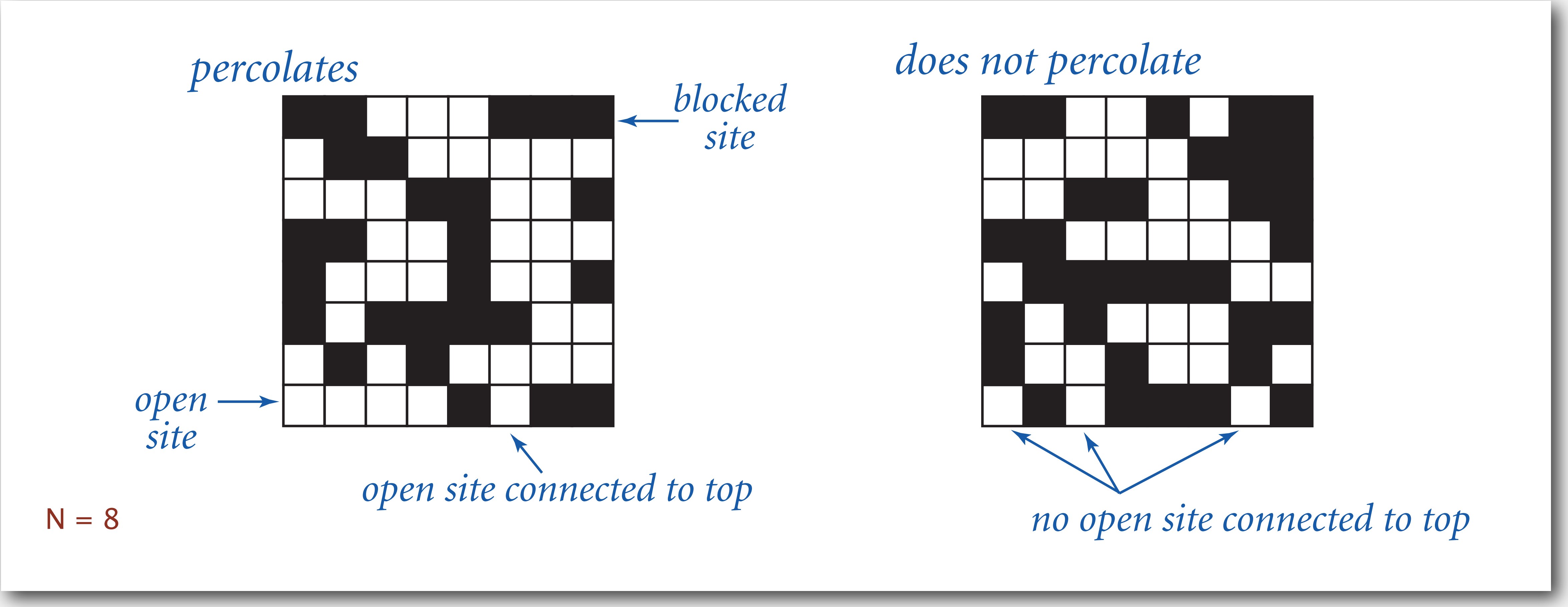




Percolation

An abstract model for many physical systems:

- N-by-N grid of sites.
- Each site is open with probability p (or blocked with probability $1 - p$).
- System percolates iff top and bottom are connected by open sites.



Percolation

An abstract model for many physical systems:

- N-by-N grid of sites.
- Each site is open with probability p (or blocked with probability $1 - p$).
- System **percolates** iff top and bottom are connected by open sites.

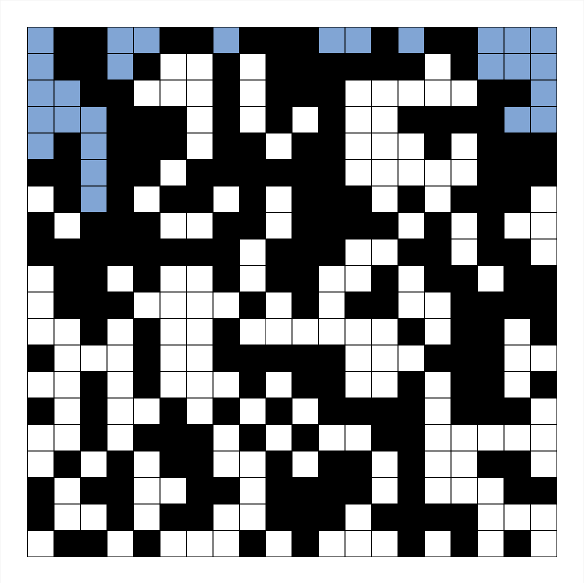
model	system	vacant site	occupied site	percolates
electricity	material	conductor	insulated	conducts
fluid flow	material	empty	blocked	porous
social interaction	population	person	empty	communicates

Likelihood of percolation

Depends on site vacancy probability p.

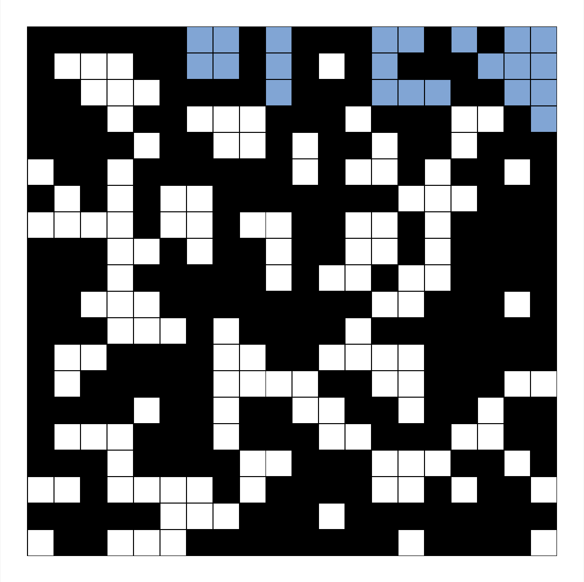
Likelihood of percolation

Depends on site vacancy probability p.



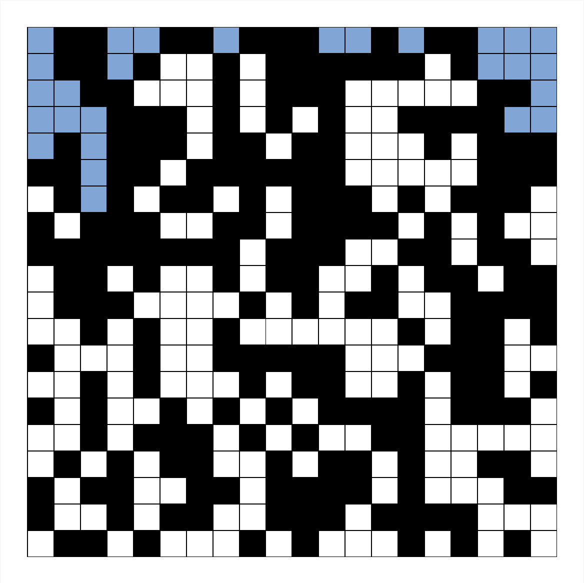
p low (0.4)

does not percolate



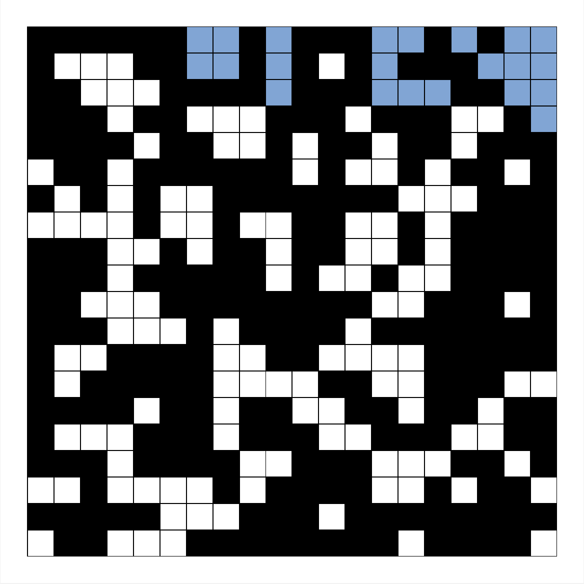
Likelihood of percolation

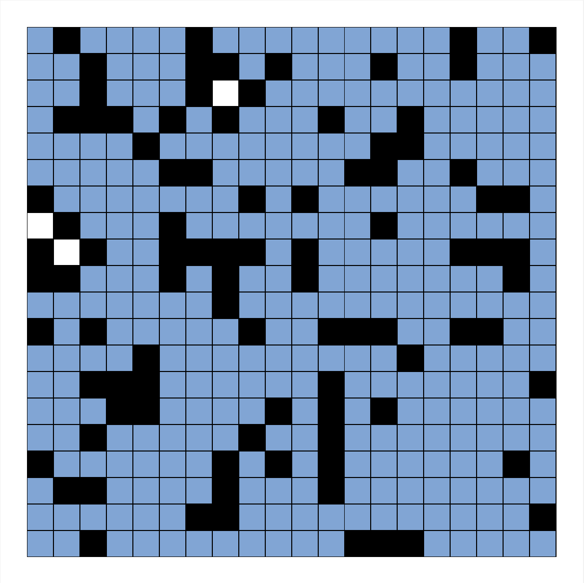
Depends on site vacancy probability p.



p low (0.4)

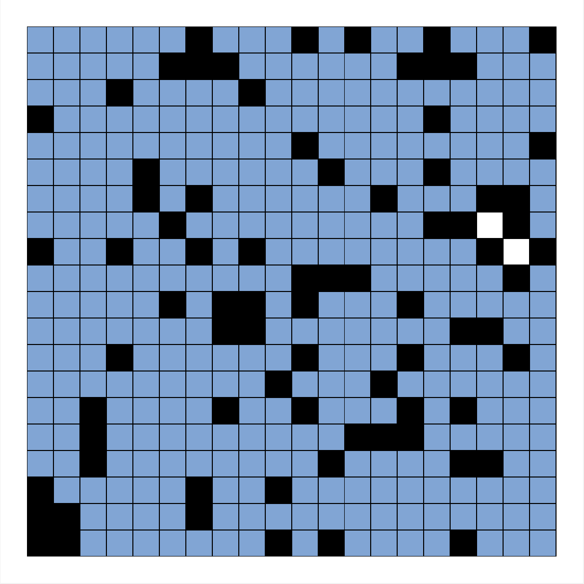
does not percolate





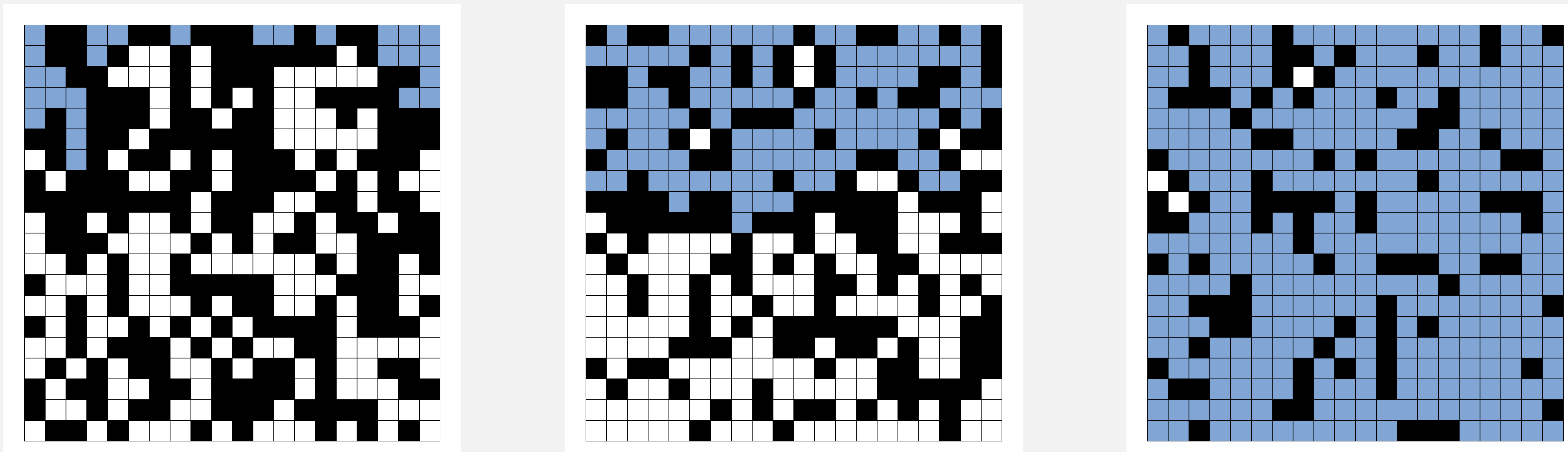
p high (0.8)

percolates



Likelihood of percolation

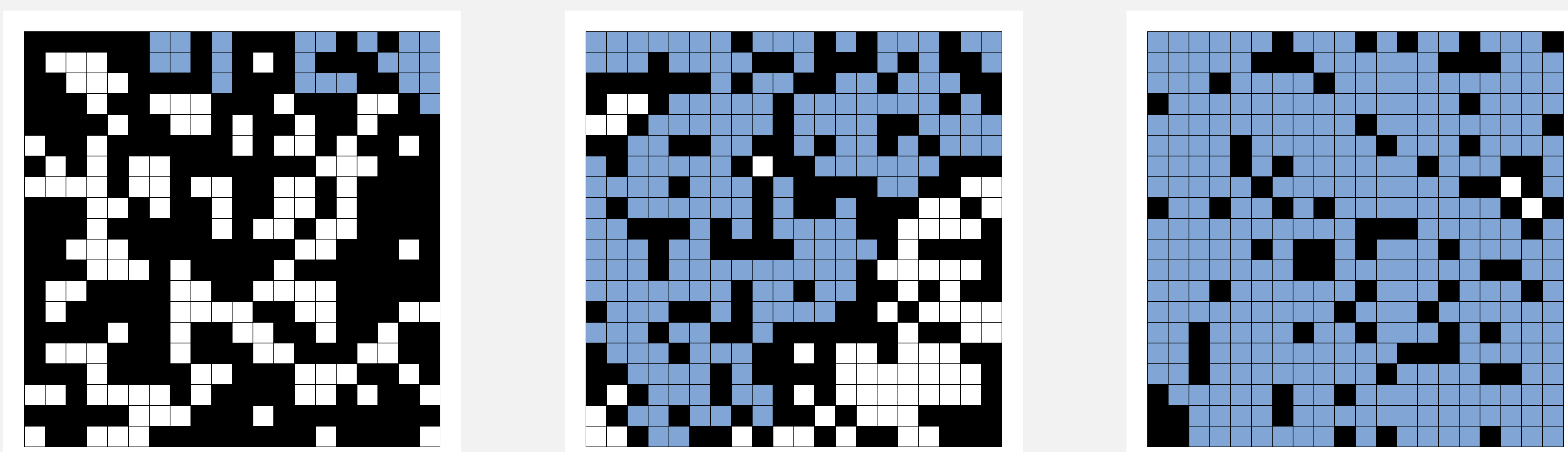
Depends on site vacancy probability p.



p low (0.4)
does not percolate

p medium (0.6)
percolates?

p high (0.8)
percolates



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.

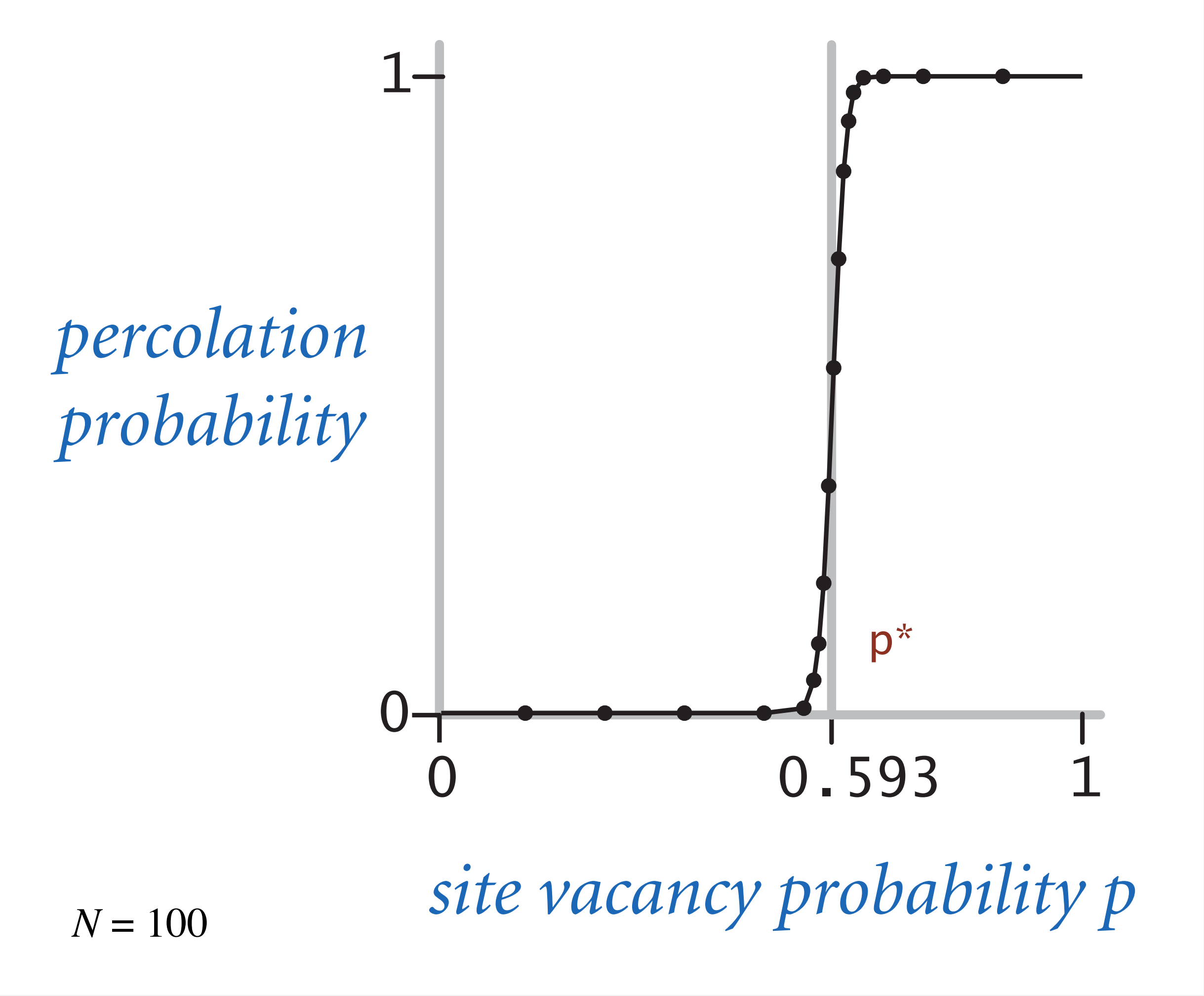
Q. What is the value of p^* ?

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^* ?



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^*.

full open site (connected to top)

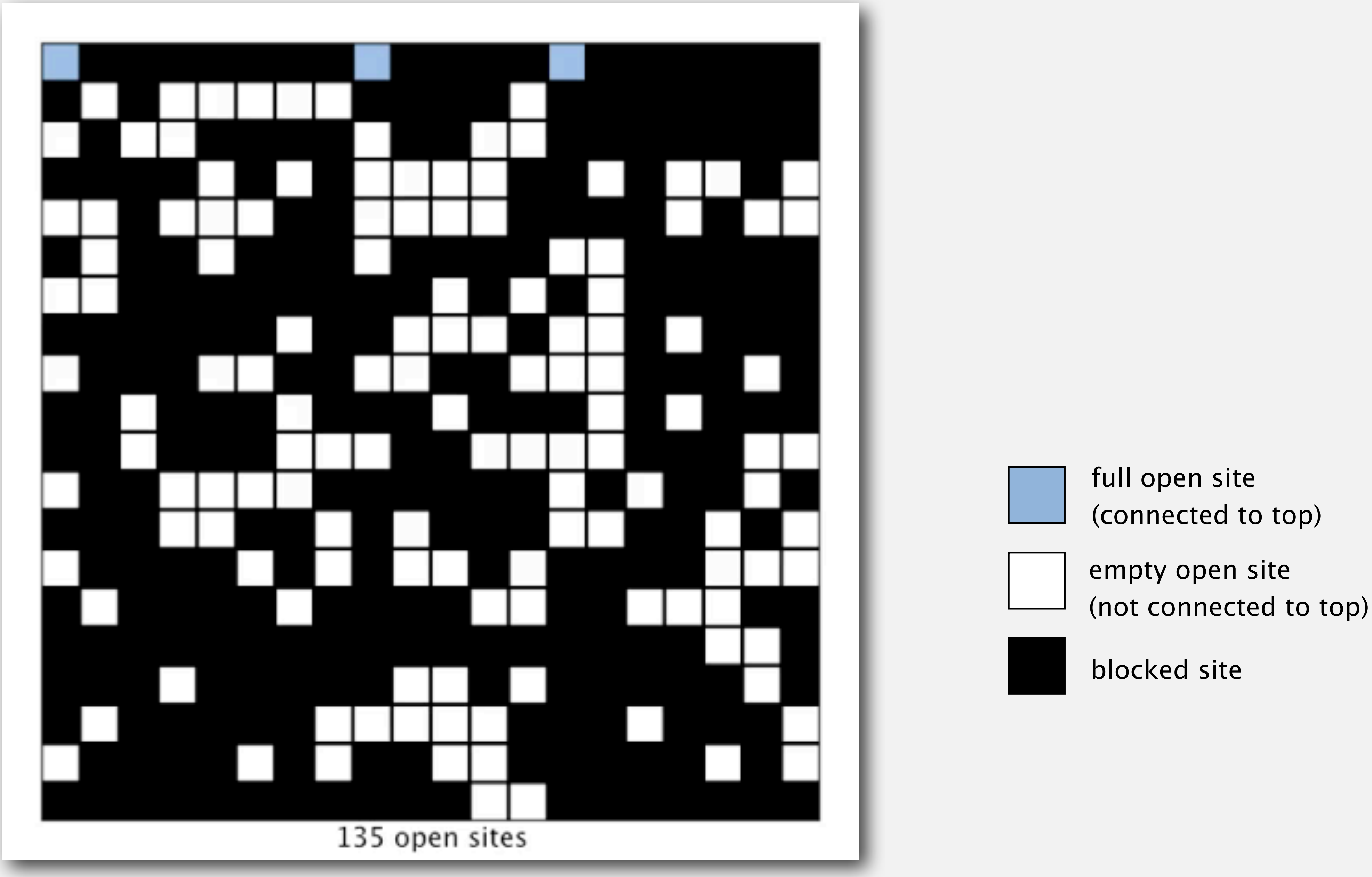
empty open site (not connected to top)

blocked site

$N = 20$

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^*.

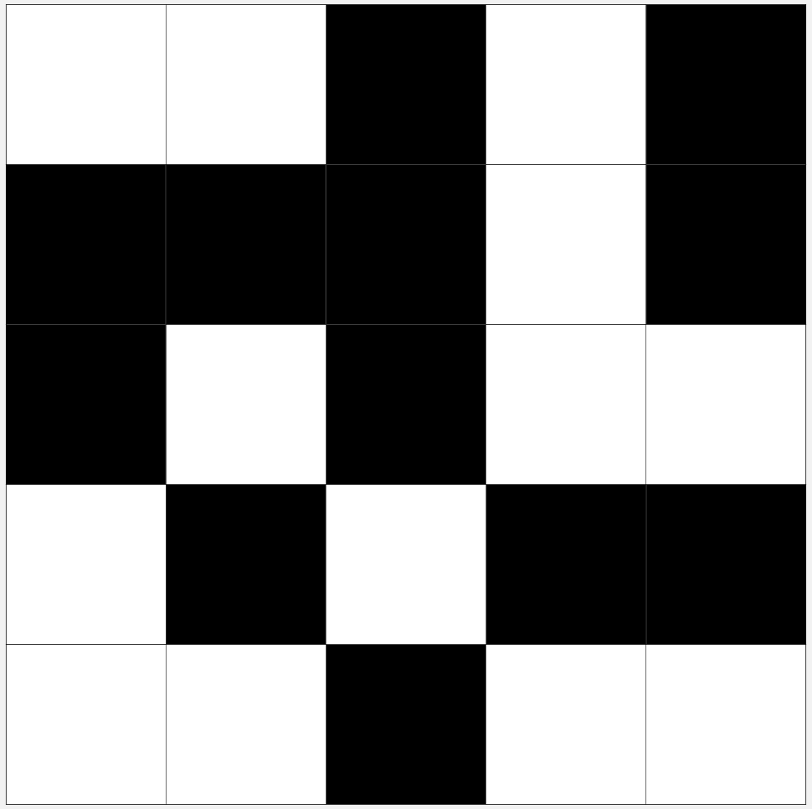


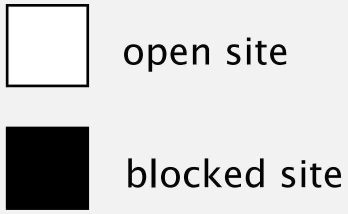
$N = 20$

Dynamic connectivity solution to estimate percolation threshold

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

$N = 5$

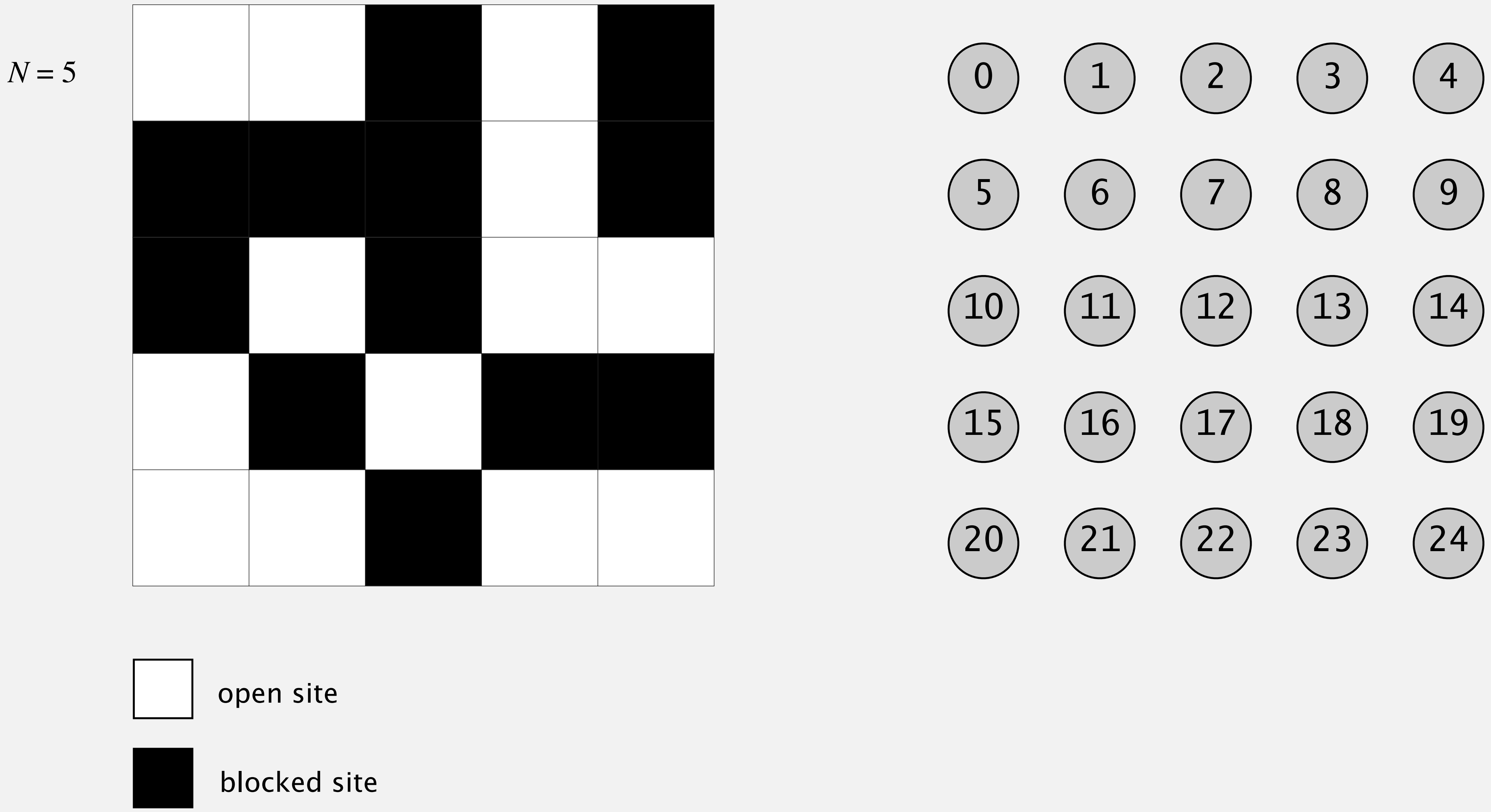




Dynamic connectivity solution to estimate percolation threshold

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

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

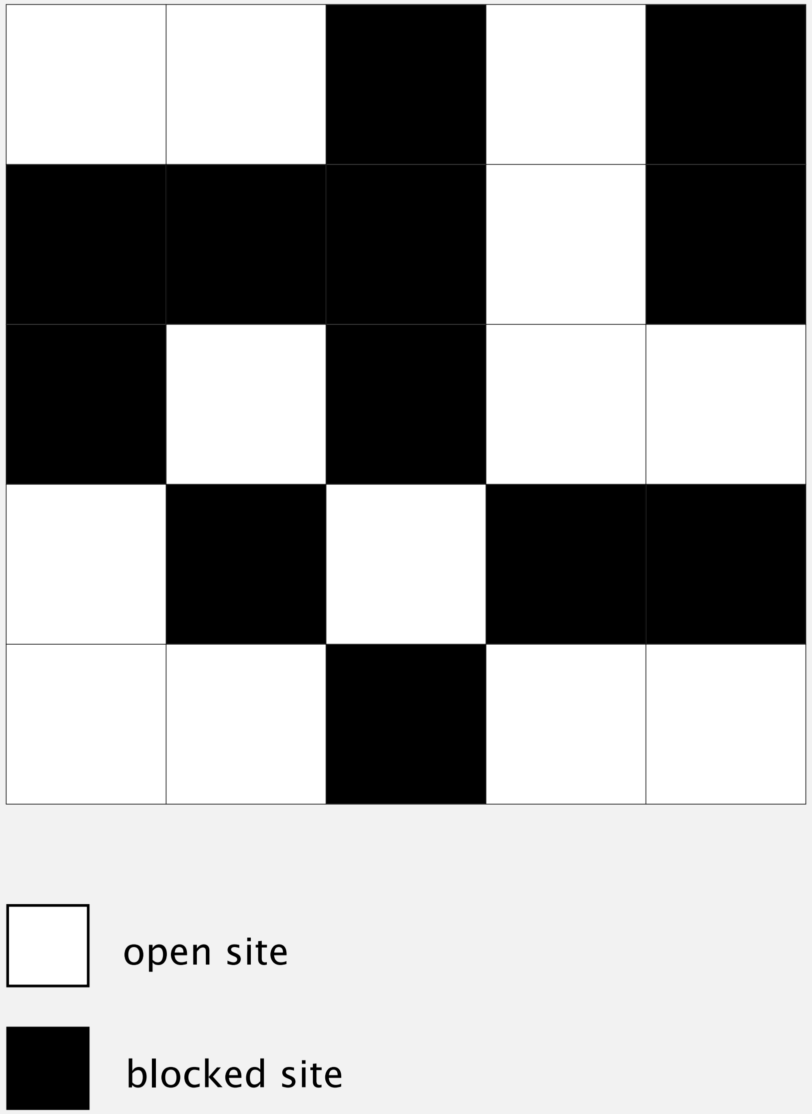


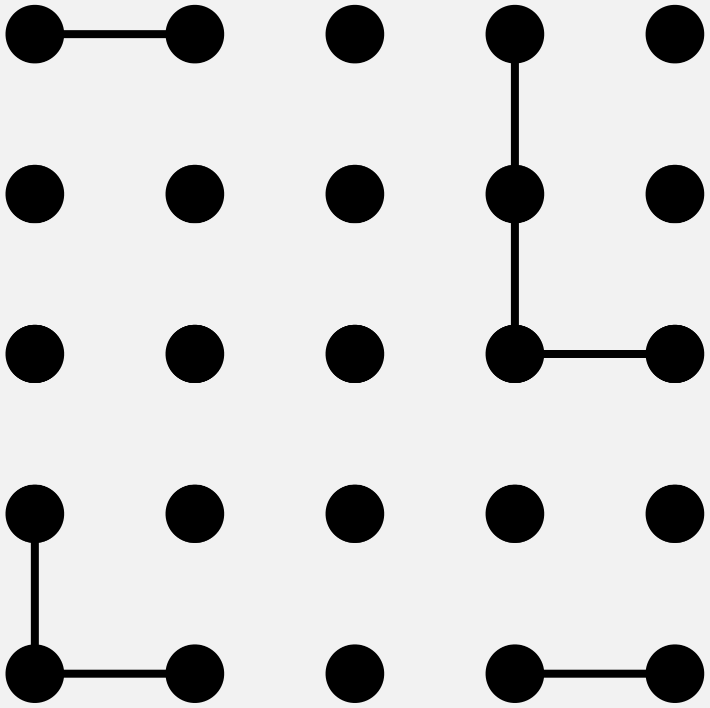
Dynamic connectivity solution to estimate percolation threshold

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

- Create an object for each site and name them 0 to $N^2 - 1$.
- Sites are in same component if connected by open sites.

$N = 5$





open site

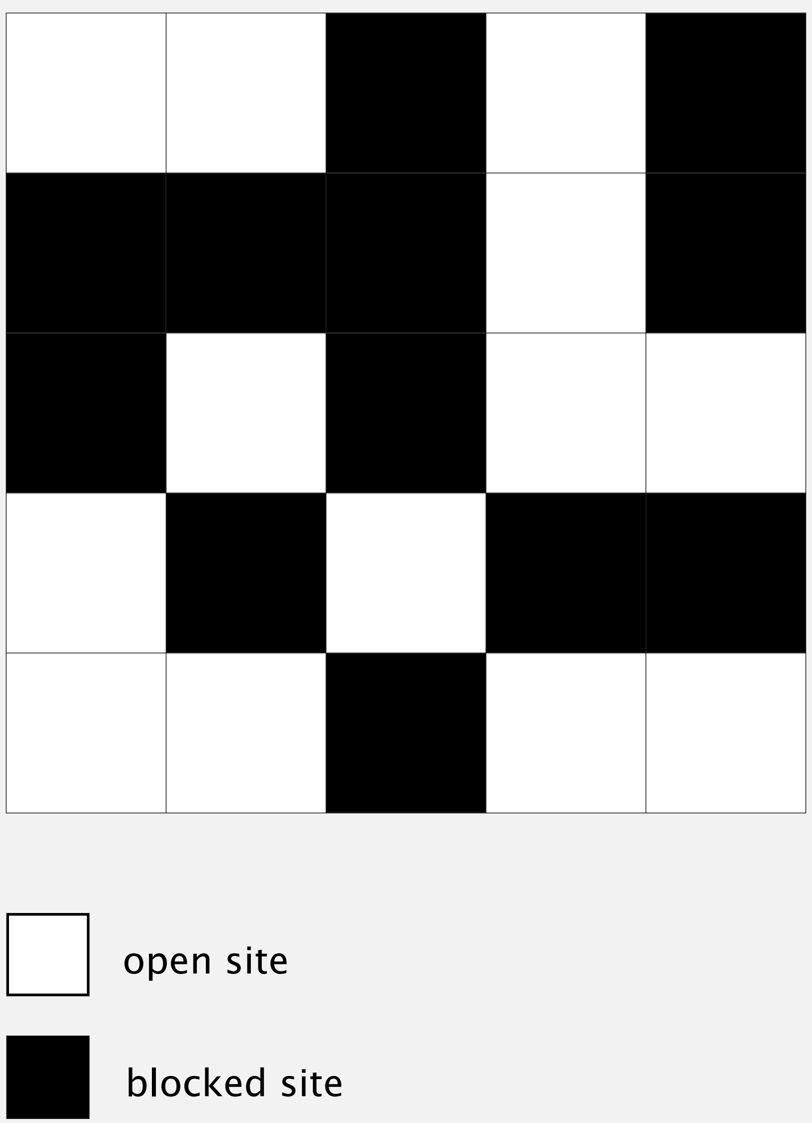
blocked site

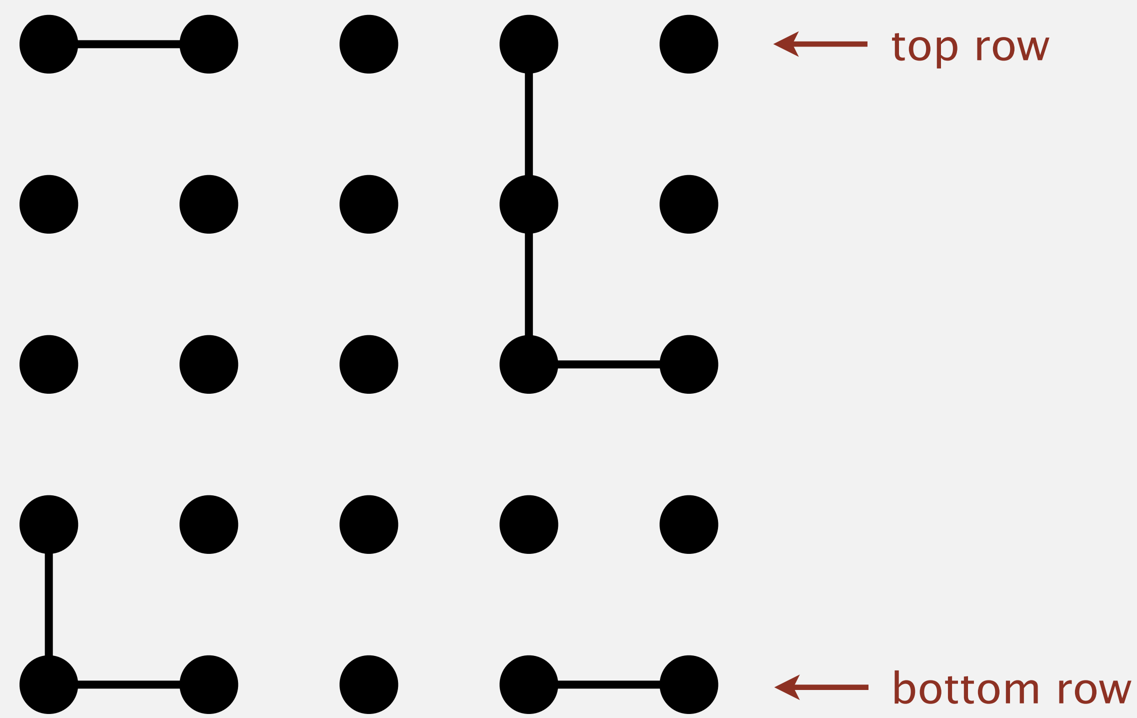
Dynamic connectivity solution to estimate percolation threshold

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

- Create an object for each site and name them 0 to $N^2 - 1$.
- Sites are in same component if connected by open sites.
- Percolates iff any site on bottom row is connected to site on top row.

$N = 5$





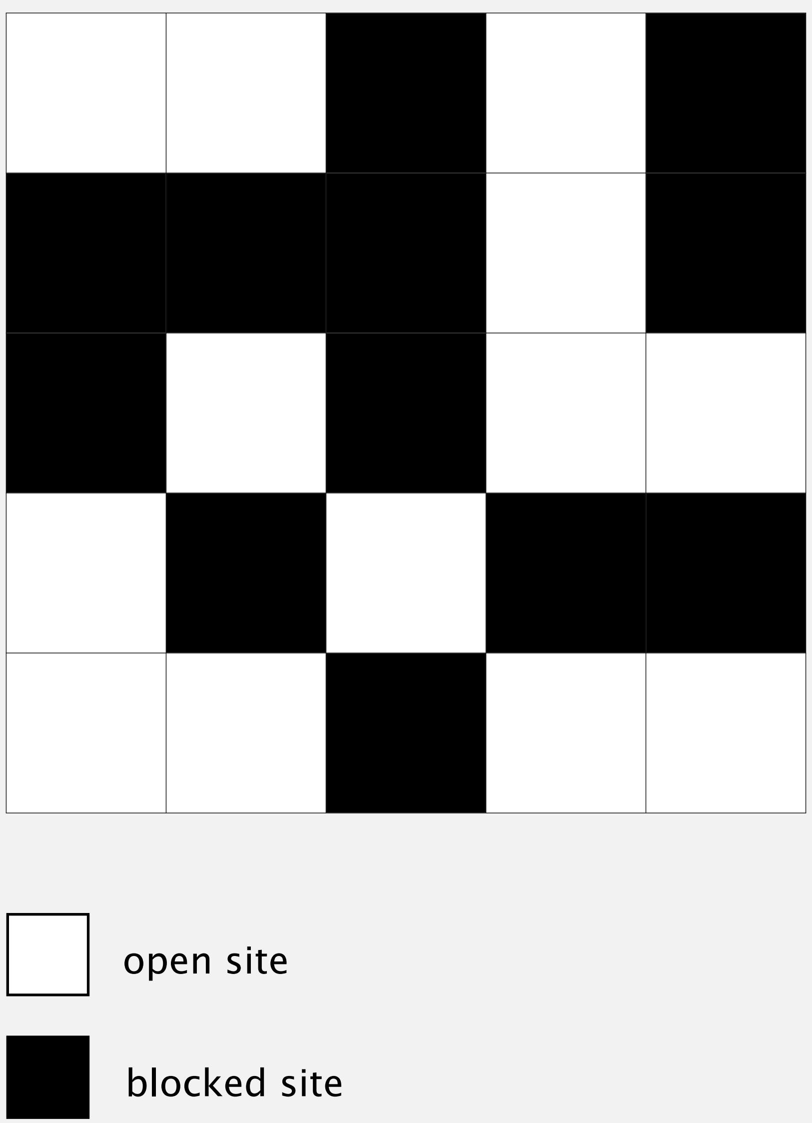
Dynamic connectivity solution to estimate percolation threshold

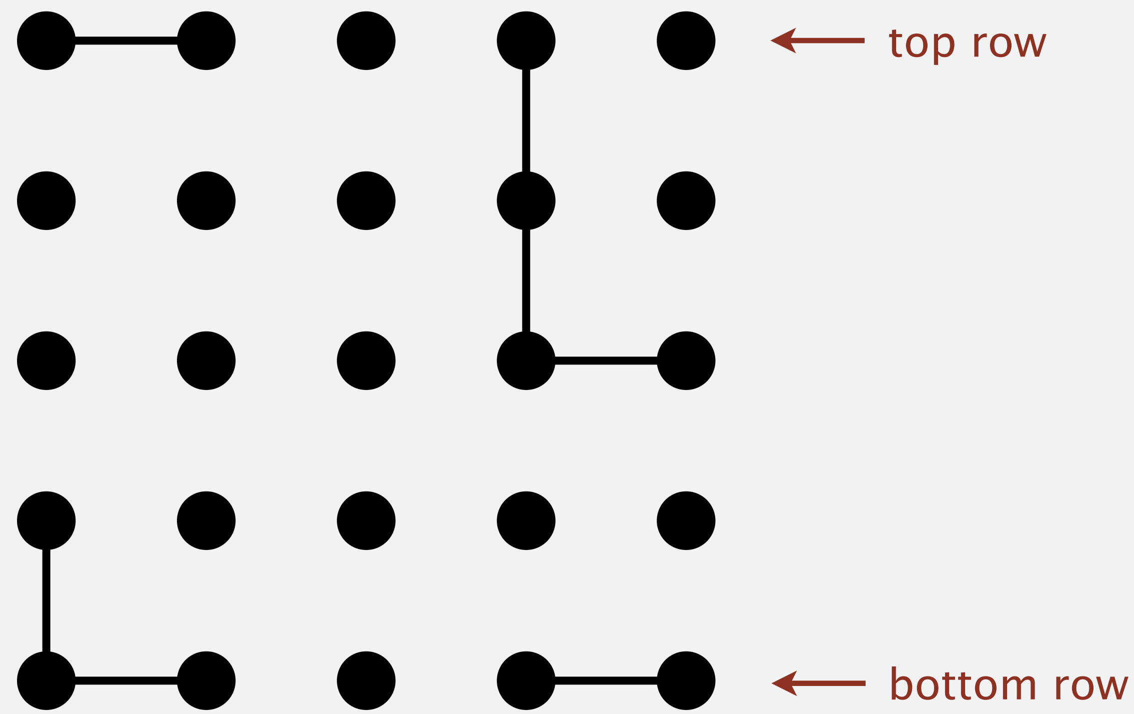
Q. How to check whether an N-by-N system percolates?

- Create an object for each site and name them 0 to $N^2 - 1$.
- Sites are in same component if connected by open sites.
- Percolates iff any site on bottom row is connected to site on top row.

brute-force algorithm: N^2 calls to connected()

$N = 5$



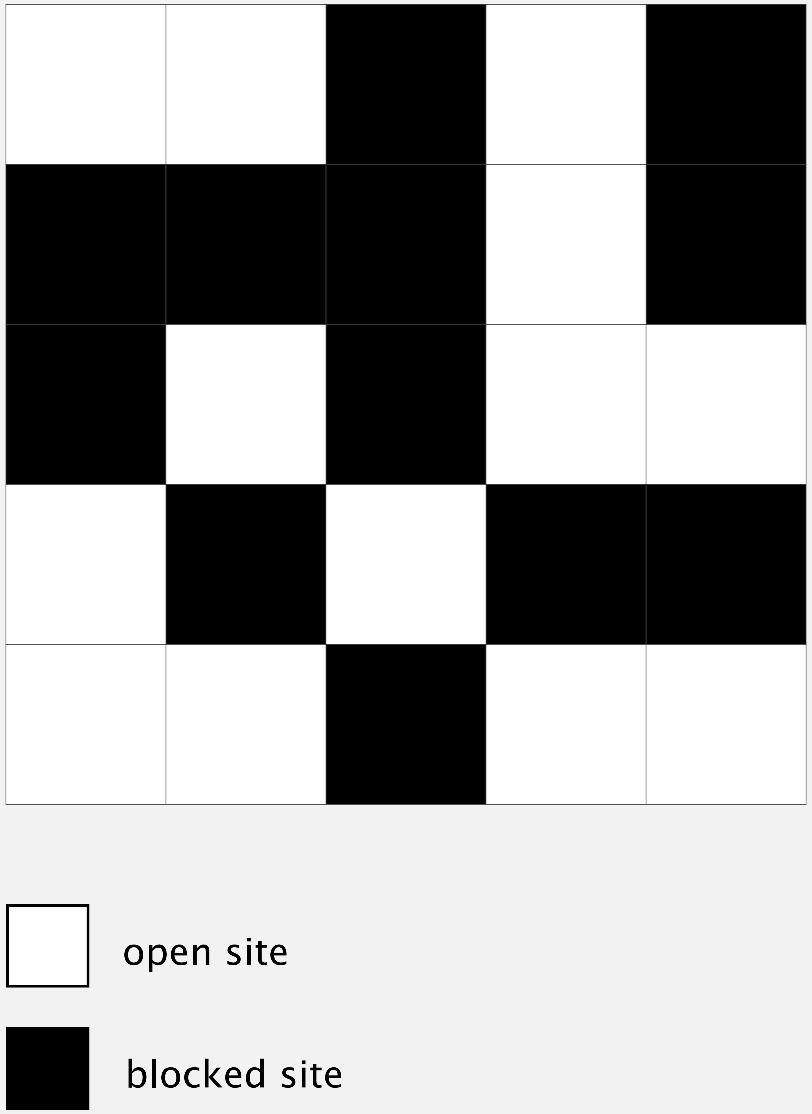


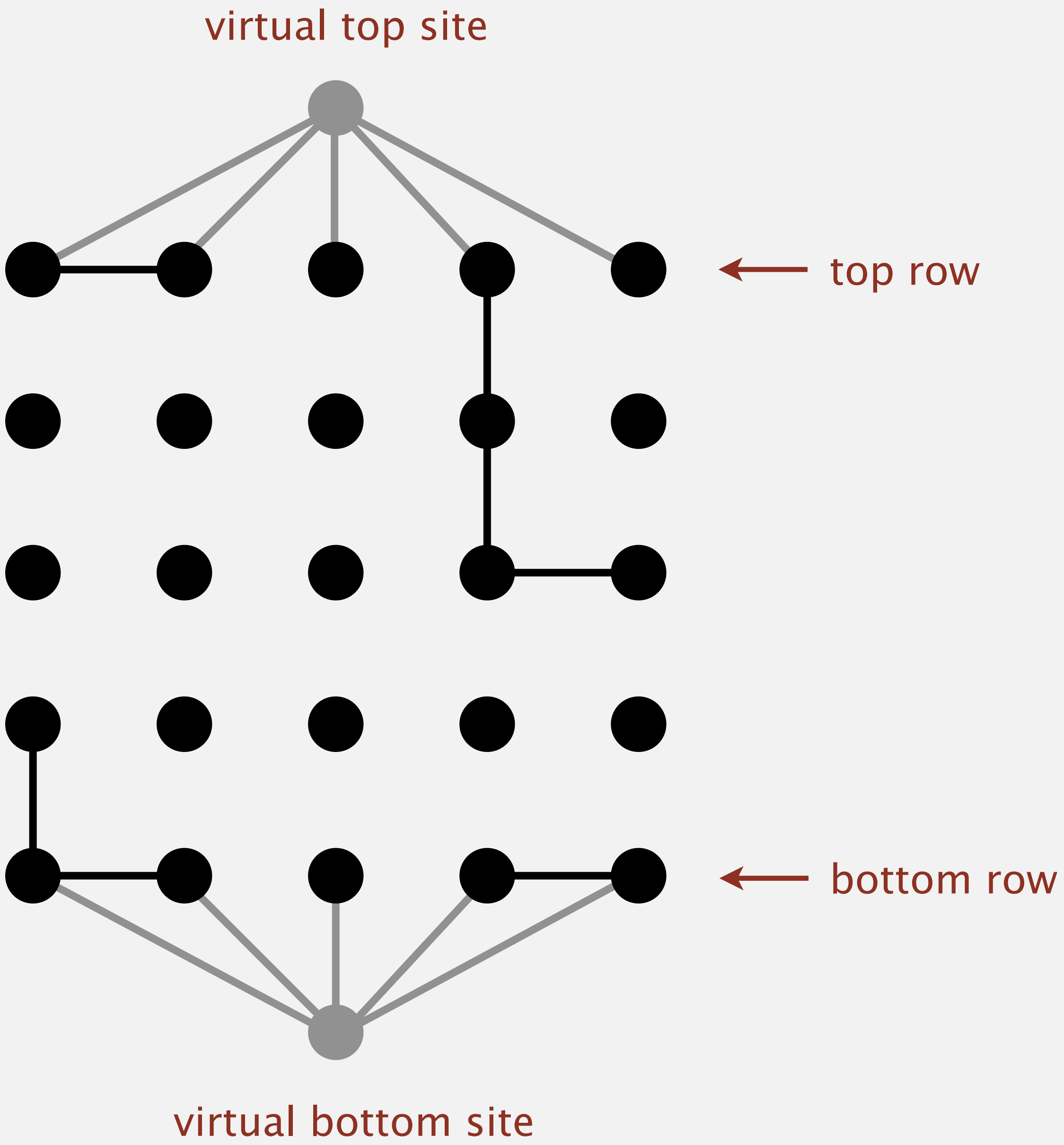
Dynamic connectivity solution to estimate percolation threshold

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

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

$N = 5$





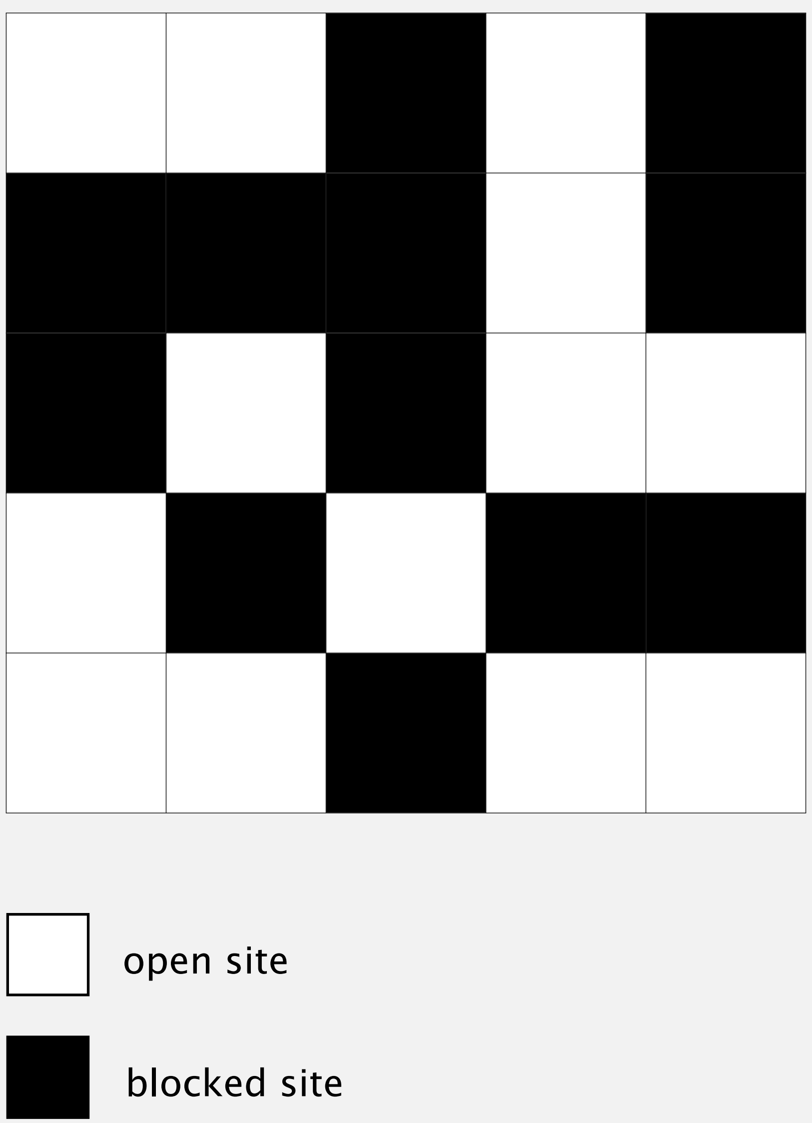
Dynamic connectivity solution to estimate percolation threshold

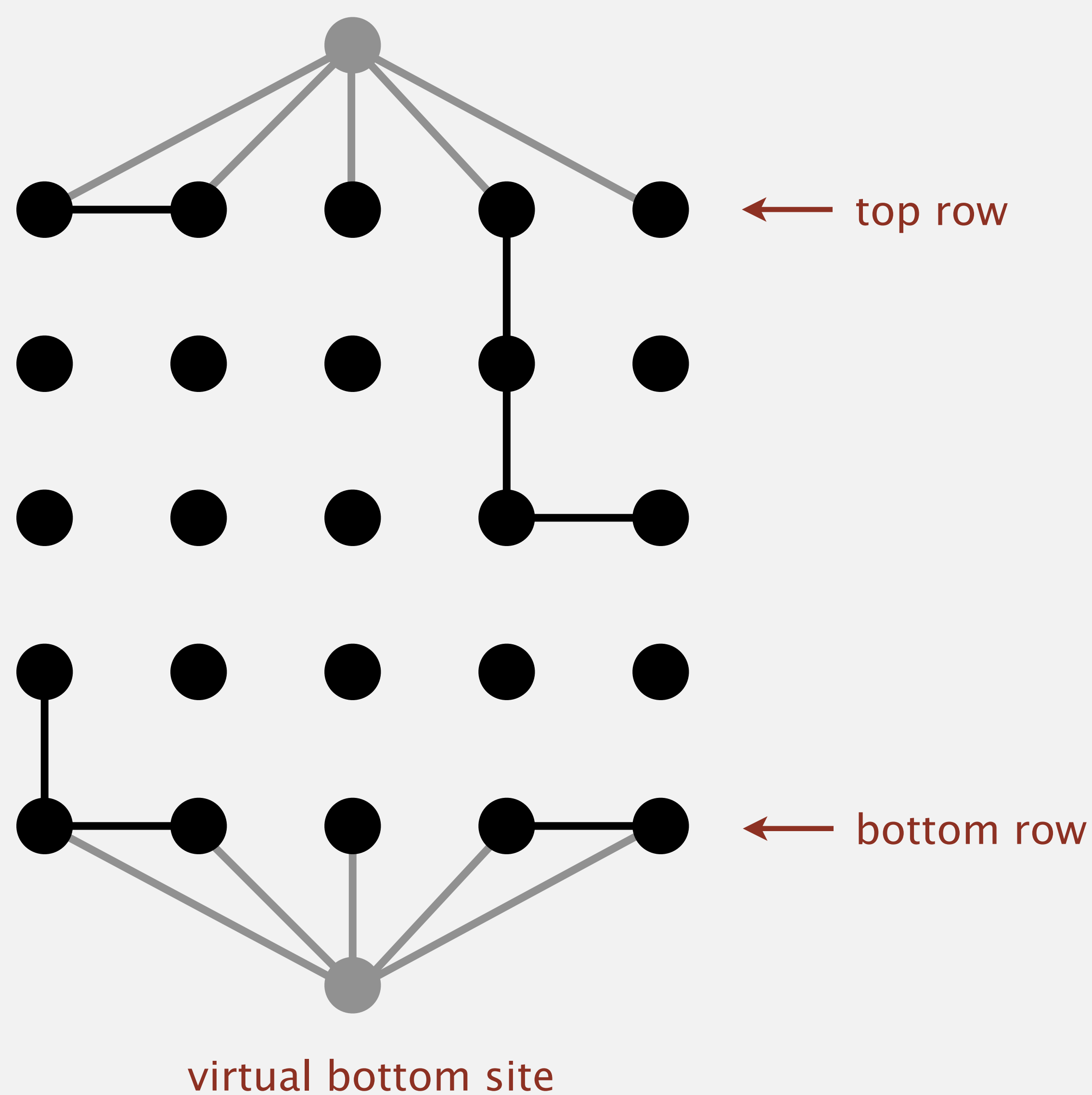
Clever trick. Introduce 2 virtual sites (and connections to top and bottom).

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

efficient algorithm: only 1 call to connected()

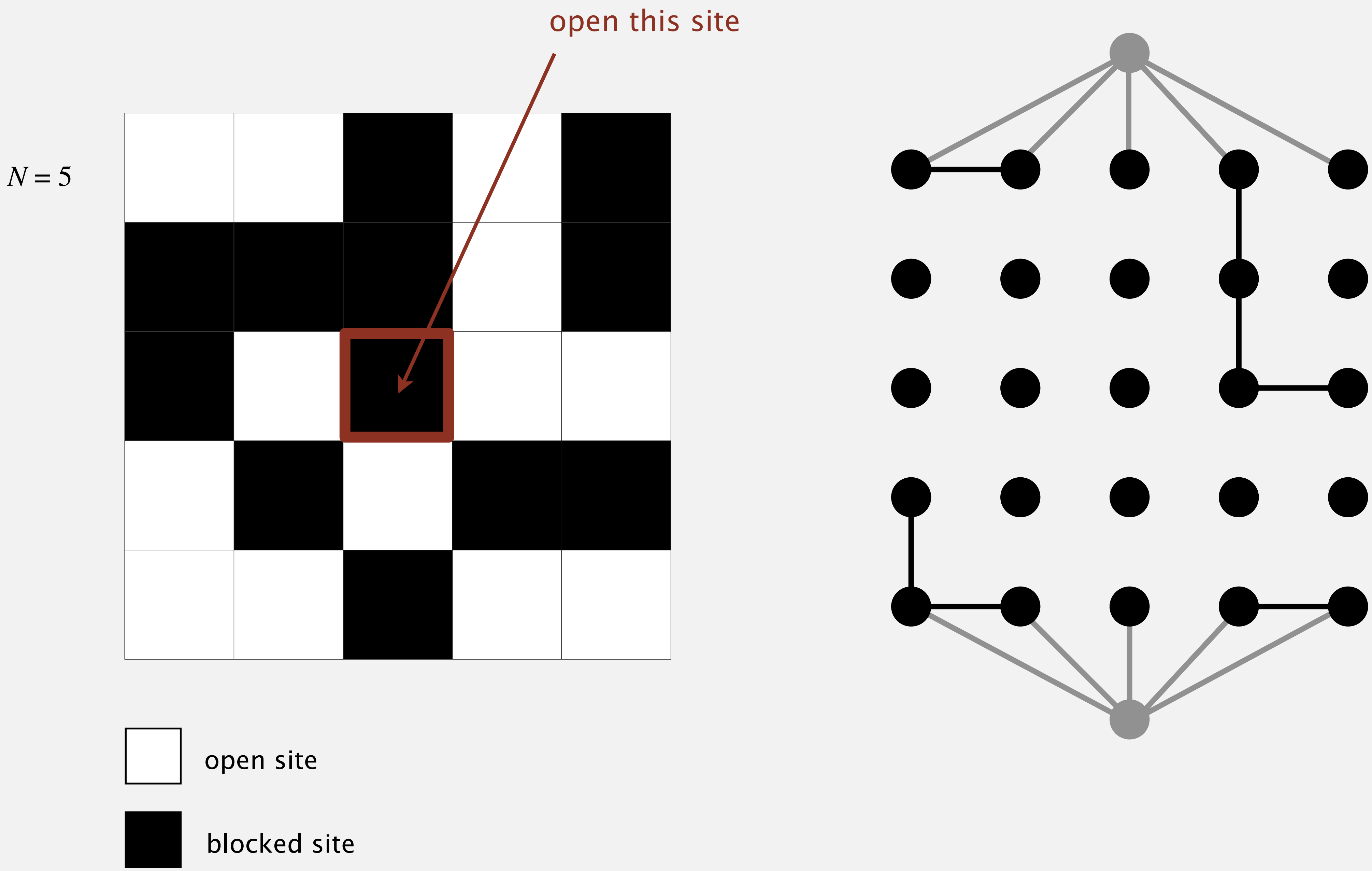
$N = 5$





Dynamic connectivity solution to estimate percolation threshold

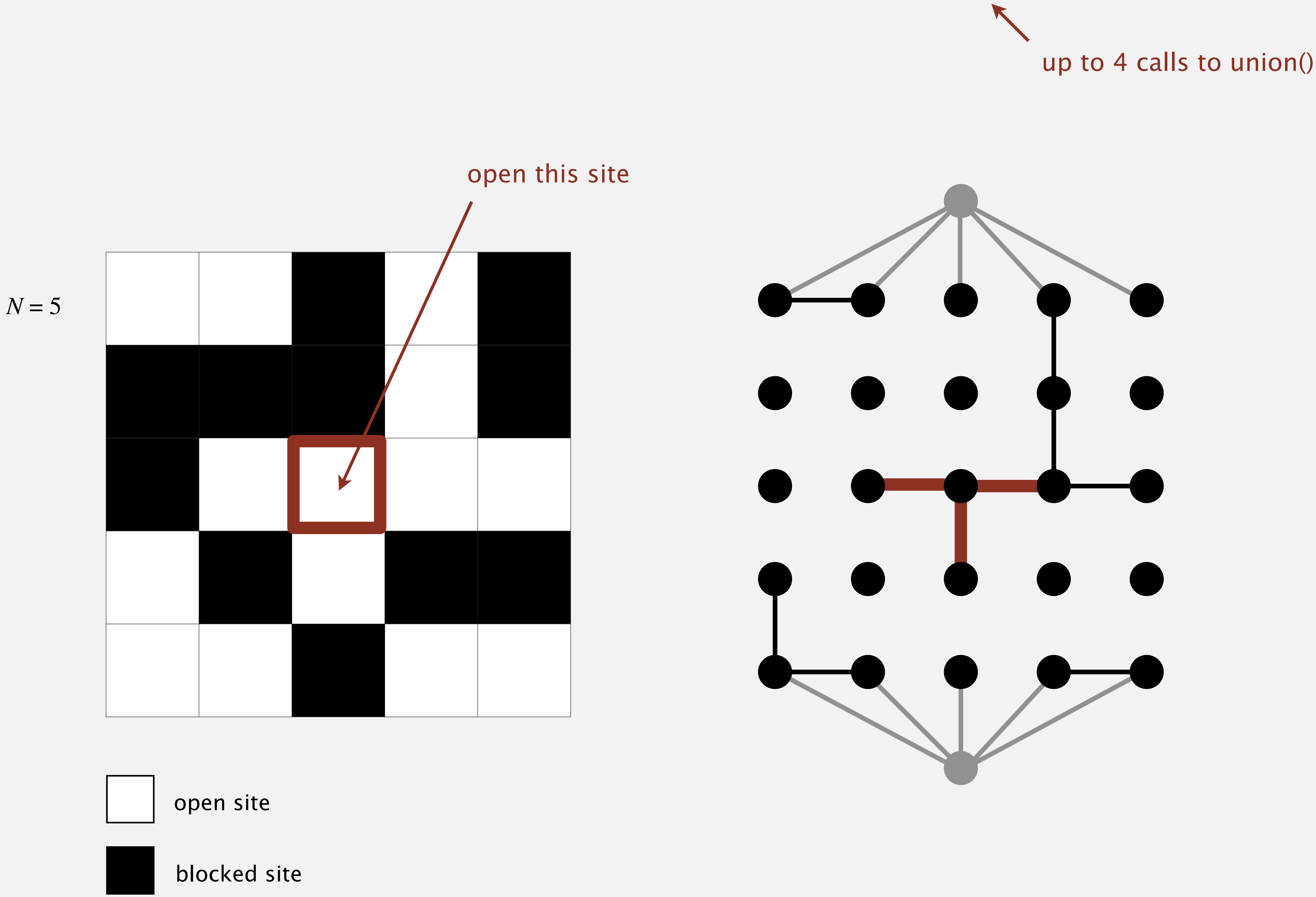
Q. How to model opening a new site?



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.

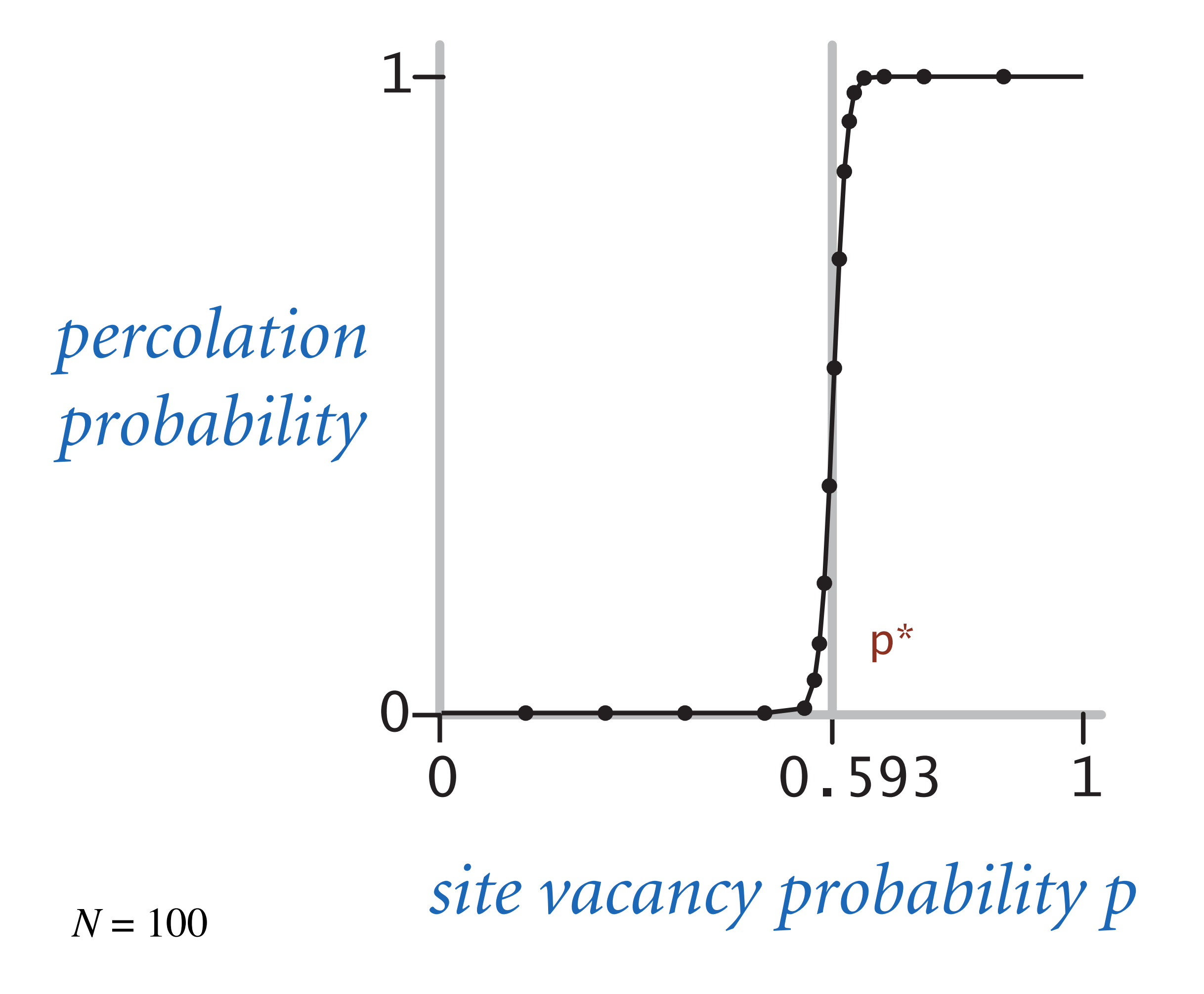


Percolation threshold

Q. What is percolation threshold p^* ?

A. About 0.592746 for large square lattices.

constant known only via simulation



Fast algorithm enables accurate answer to scientific question.

Subtext of today's lecture (and this course)

Steps to developing a usable algorithm.

- Model the problem.
- Find an algorithm to solve it.
- Fast enough? Fits in memory?
- If not, figure out why.
- Find a way to address the problem.
- Iterate until satisfied.

The scientific method.

Mathematical analysis.