CS51: Abstraction and Design in Computation

Introduction

Now playing:
"Jewel"
Flume
Hi This Is Flume

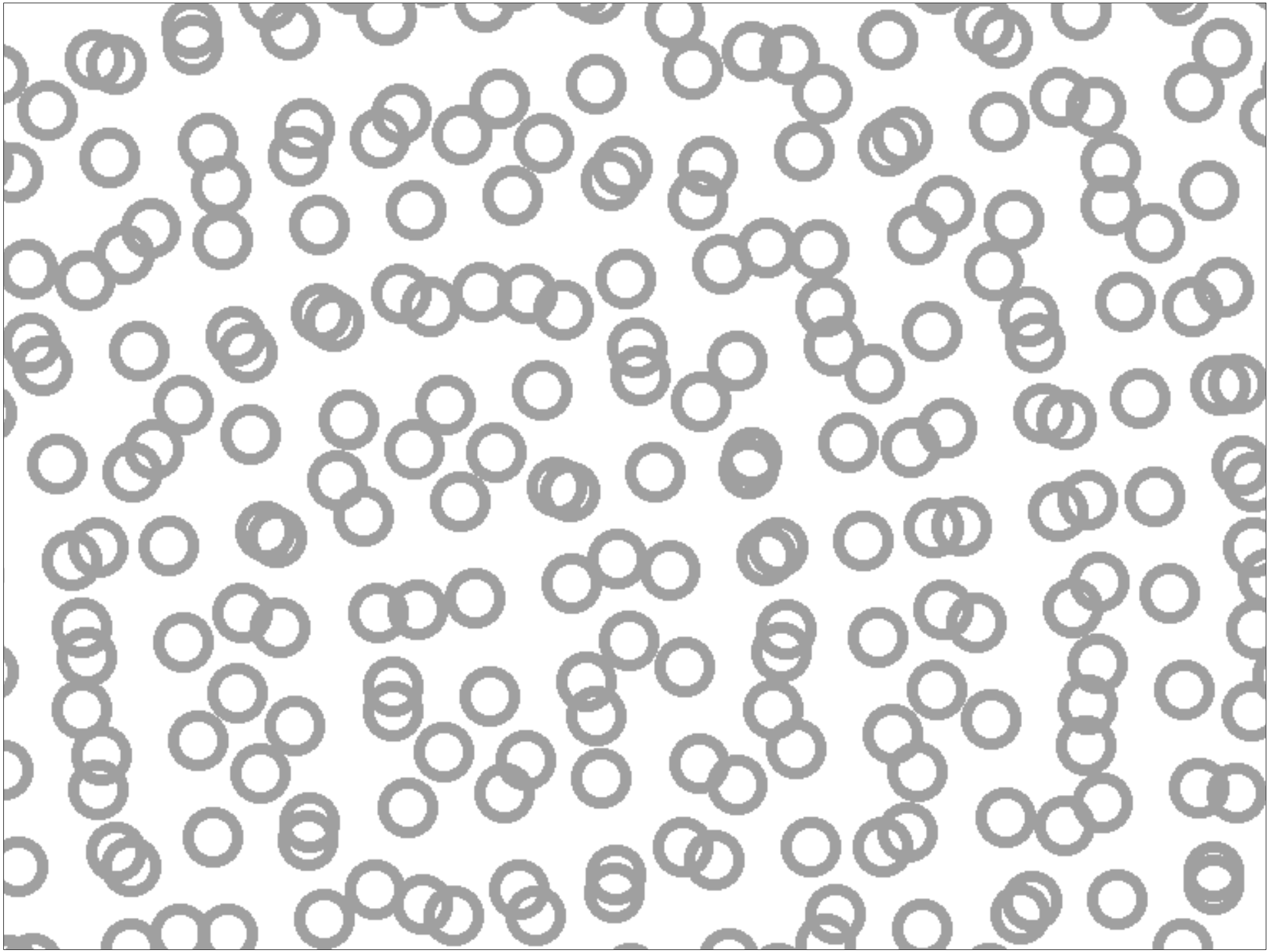
John A. Paulson School of Engineering and Applied Sciences
Harvard University

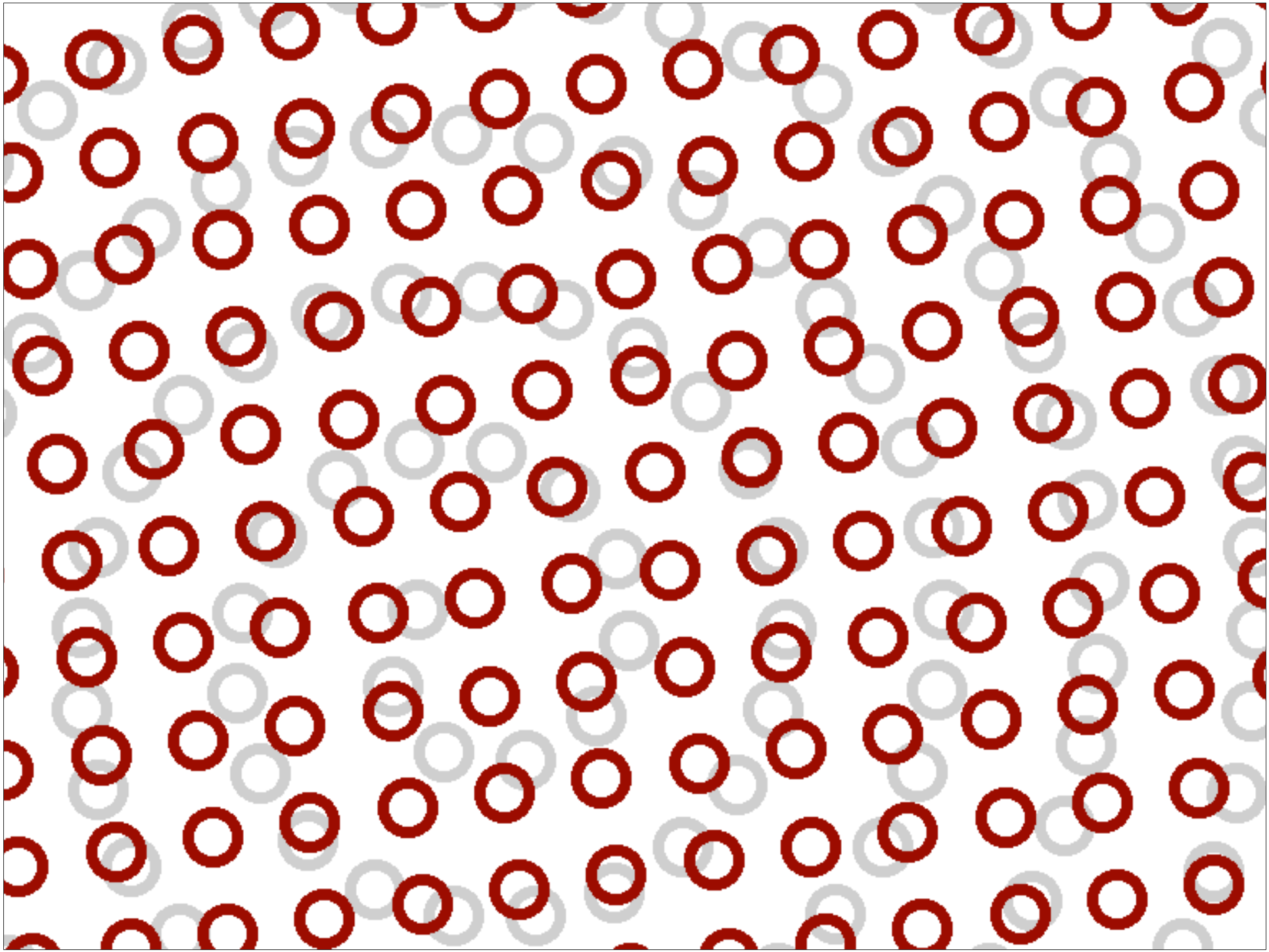
What CS51 is about

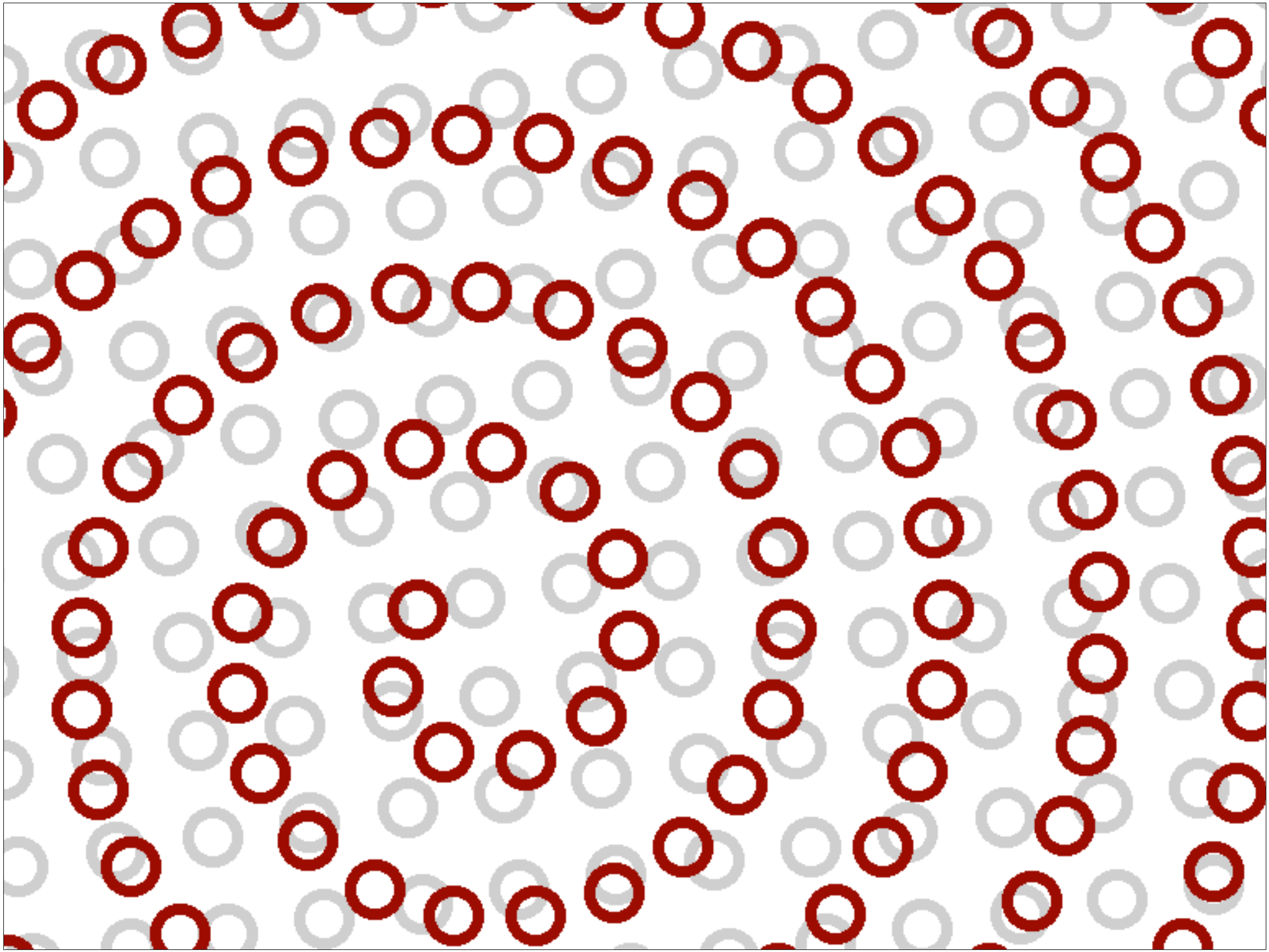
engineering: producing (software) solutions with desirable properties along multiple criteria

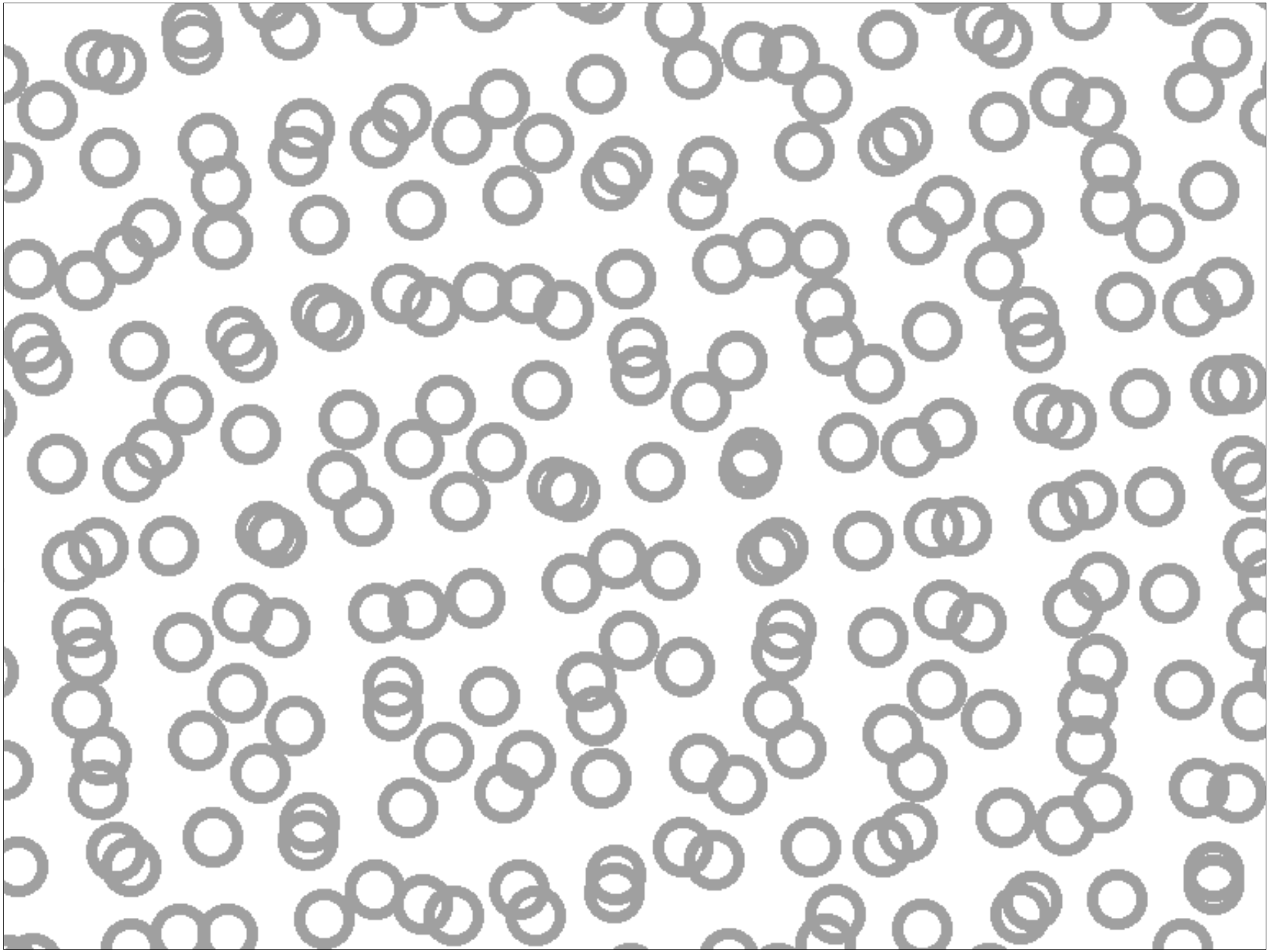
design: selection among alternative approaches to engineering a (software) artifact

abstraction: the process of viewing a set of apparently dissimilar things as instantiating an underlying identity; enabling the alternatives necessary for design







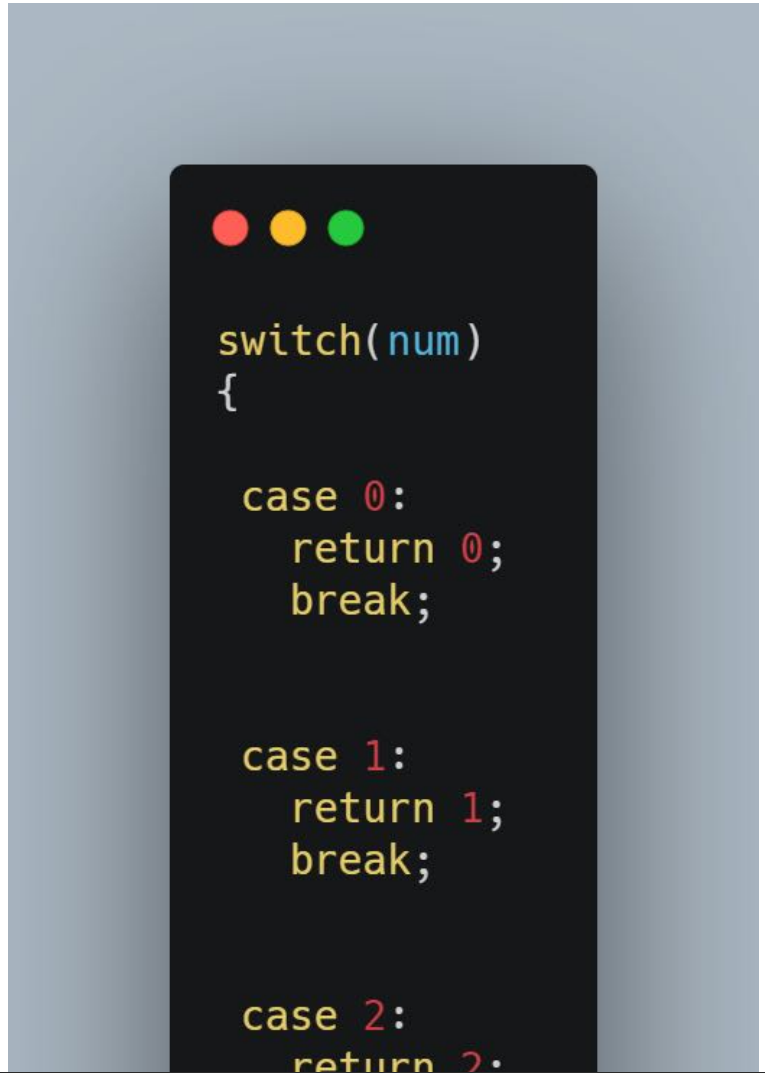


An example abstraction

```
print_int 2; print_newline ();
print_int 3; print_newline ();
print_int 4; print_newline ();
print_int 5; print_newline ();
print_int 6; print_newline ();
print_int 7; print_newline ();
print_int 8; print_newline ()
;;
```

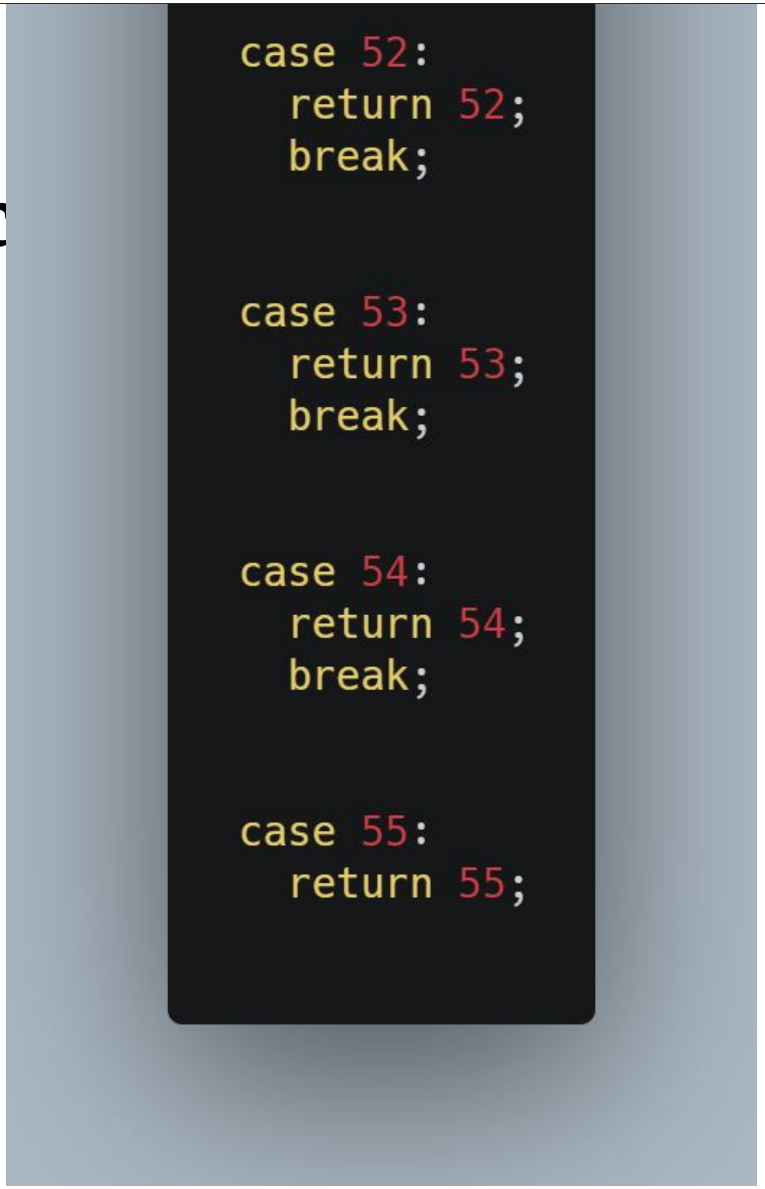
An example abstraction

```
print_int 2; print_newline ();
print_int 3; print_newline ();
print_int 4; print_newline ();
print_int 5; print_newline ();
print_int 6; print_newline ();
print_int 7; print_newline ();
print_int 8; print_newline ()
;;
```



An example abstrac

```
print_int 2; print_newline ();
print_int 3; print_newline ();
print_int 4; print_newline ();
print_int 5; print_newline ();
print_int 6; print_newline ();
print_int 7; print_newline ();
print_int 8; print_newline ()
;;
```



```
case 52:
  return 52;
  break;

case 53:
  return 53;
  break;

case 54:
  return 54;
  break;

case 55:
  return 55;
```

from u/KroutontheSlasher
on r/badcode, 1/22/21

An example abstraction: the state variable

```
print_int 2; print_newline ();
print_int 3; print_newline ();
print_int 4; print_newline ();
print_int 5; print_newline ();
print_int 6; print_newline ();
print_int 7; print_newline ();
print_int 8; print_newline ()
;;
```

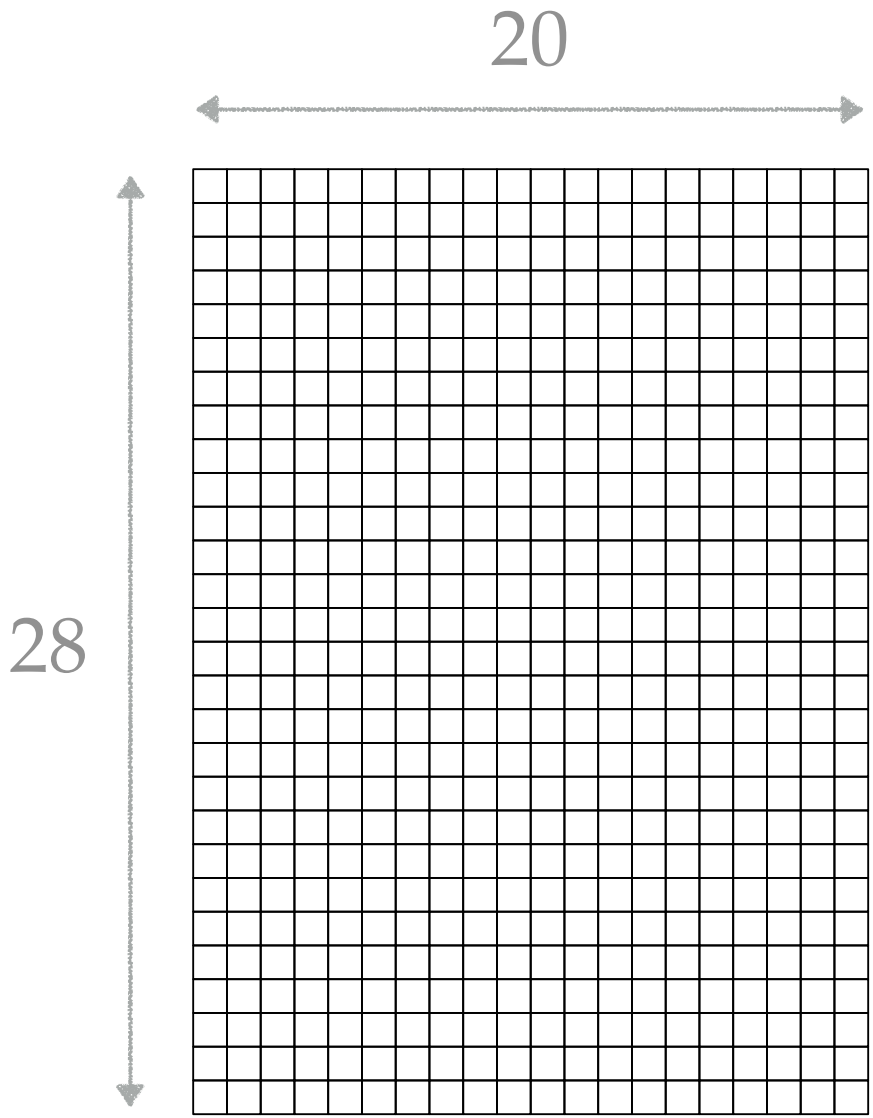
```
print_int x; print_newline ()
```

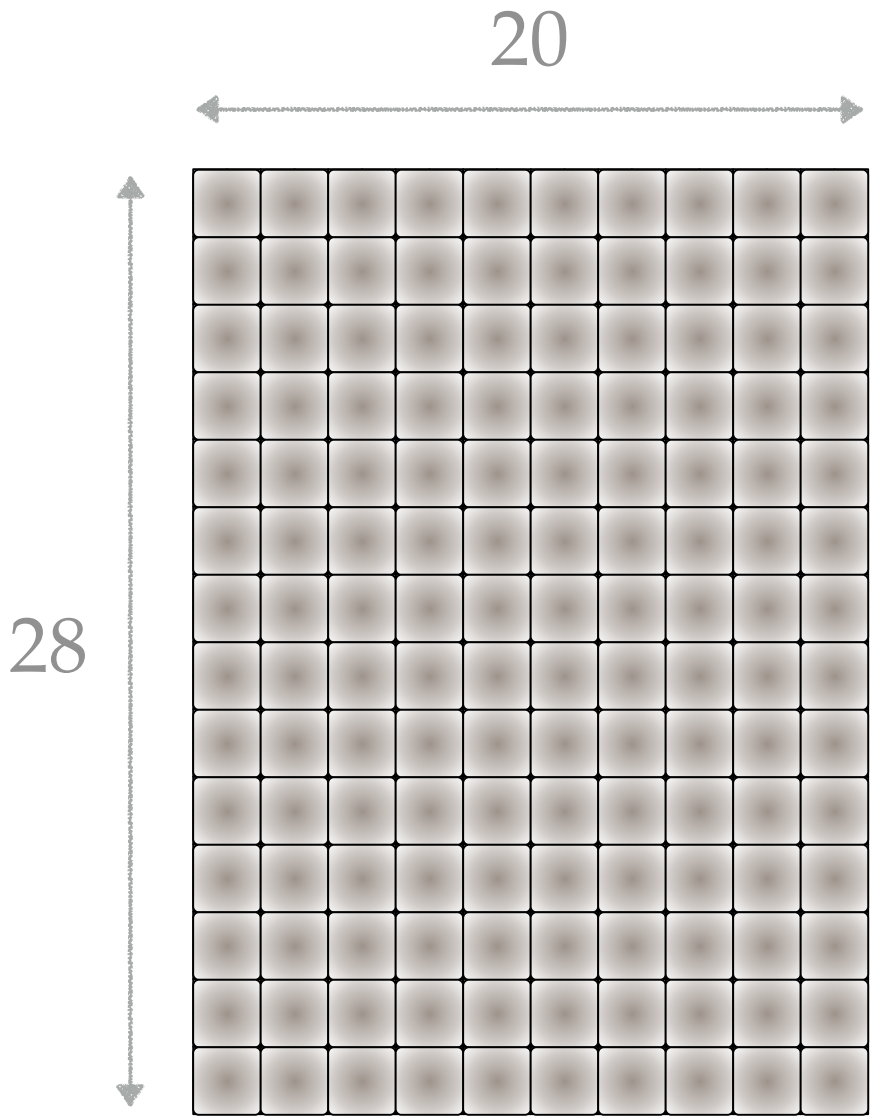
An example abstraction: the state variable and the loop

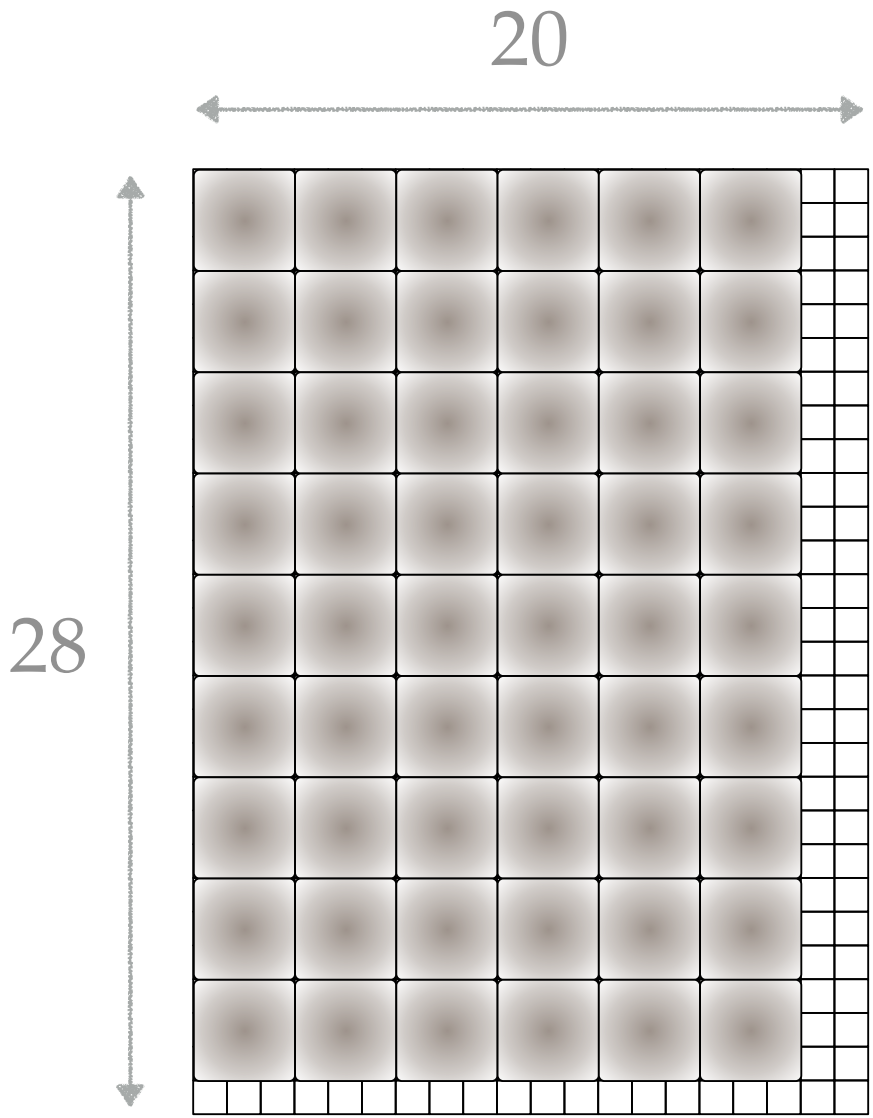
```
print_int 2; print_newline ();
print_int 3; print_newline ();
print_int 4; print_newline ();
print_int 5; print_newline ();
print_int 6; print_newline ();
print_int 7; print_newline ();
print_int 8; print_newline ()
;;
```

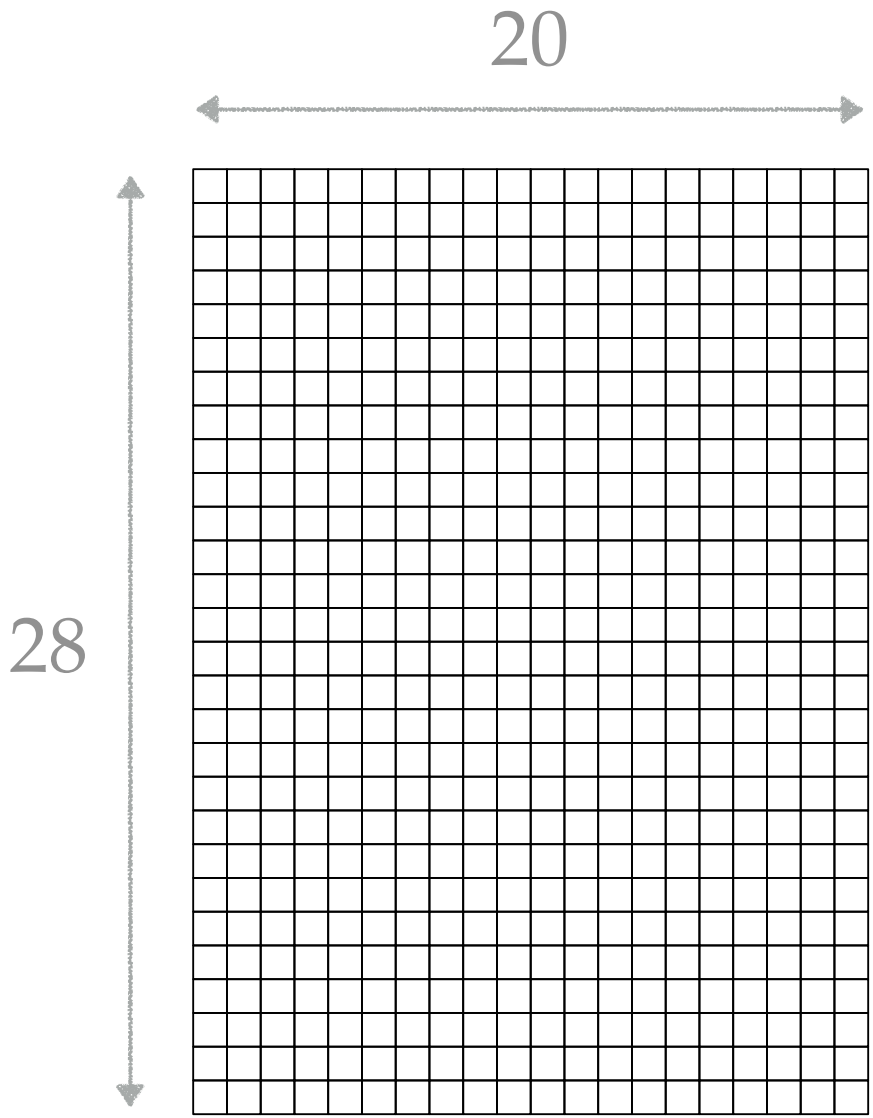
```
for x in 2 to 8 do
  print_int x; print_newline ()
done ;;
```

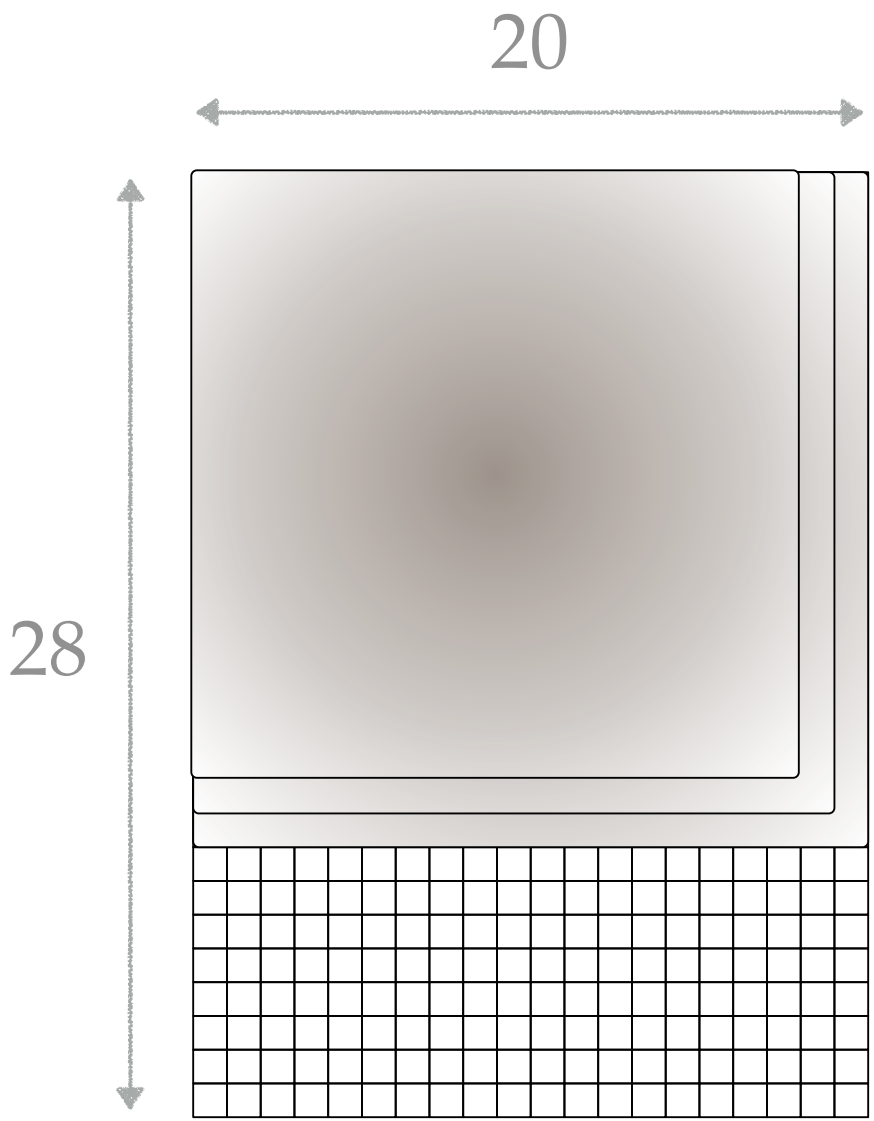
Alan Turing


20
28


20
28


20
28


20
28


20
28

```c
#include <stdio.h>

#define MIN(a, b) ((a) < (b) ? (a) : (b))

unsigned gcd_down(unsigned a, unsigned b)
{
  unsigned guess;
  for (guess=MIN(a, b); guess>1; guess--) {
    if ((a % guess == 0) && (b % guess == 0))
      break;
  }
  return guess;
}

int main()
{
  printf("gcd(10, 15) is %d\n", gcd_down(10,15));
  printf("gcd(5, 19)  is %d\n", gcd_down(5,19));
  printf("gcd(20, 10) is %d\n", gcd_down(20,10));
}
```


find this code in: `gcd.c`

This is not CS50.

```c
#include <stdio.h>

#define MIN(a, b) ((a) < (b) ? (a) : (b))

unsigned gcd_down(unsigned a, unsigned b)
{
  unsigned guess;
  for (guess=MIN(a, b); guess>1; guess--) {
    if ((a % guess == 0) && (b % guess == 0))
      break;
  }
  return guess;
}

int main()
{
  printf("gcd(10, 15) is %d\n", gcd_down(10,15));
  printf("gcd(5, 19)  is %d\n", gcd_down(5,19));
  printf("gcd(20, 10) is %d\n", gcd_down(20,10));
}
```


find this code in: `gcd.c`

```
let gcd_down a b =
  let guess = ref (min a b) in
  while (a mod !guess <> 0) || (b mod !guess <> 0) do
    guess := !guess - 1
  done;
  !guess ;;
```


find this code in: `gcd_versions.ml`

Alonzo Church

```
let gcd_func a b =
  let rec downfrom guess =
    if (a mod guess <> 0) || (b mod guess <> 0) then
      downfrom (guess - 1)
    else guess in
  downfrom (min a b) ;;
```


find this code in: `gcd_versions.ml`

Euclid of Alexandria

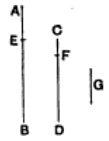
Euclid of Alexandria

PROPOSITION 2.

Given two numbers not prime to one another, to find their greatest common measure.

Let AB, CD be the two given numbers not prime to one another.

Thus it is required to find the greatest common measure of AB, CD.



If now CD measures AB—and it also measures itself—CD is a common measure of CD, AB.

And it is manifest that it is also the greatest; for no greater number than CD will measure CD.

But, if CD does not measure AB, then, the less of the numbers AB, CD being continually subtracted from the greater, some number will be left which will measure the one before it.

For an unit will not be left; otherwise AB, CD will be prime to one another [VII. 1], which is contrary to the hypothesis.

Therefore some number will be left which will measure the one before it.

Now let CD, measuring BE, leave EA less than itself, let EA, measuring DF, leave FC less than itself, and let CF measure AE.

Since then, CF measures AE, and AE measures DF, therefore CF will also measure DF.

But it also measures itself;

therefore it will also measure the whole CD.

But CD measures BE;

therefore CF also measures BE.

But it also measures EA;

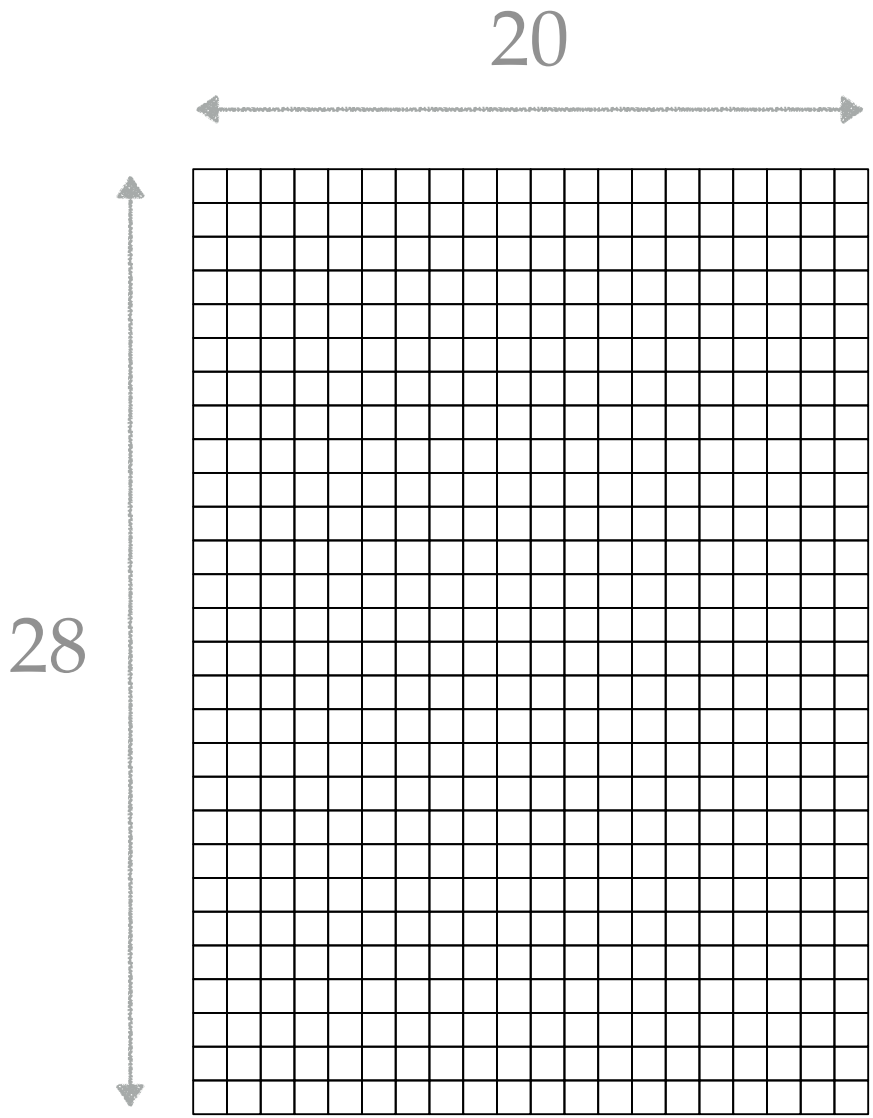
therefore it will also measure the whole BA.

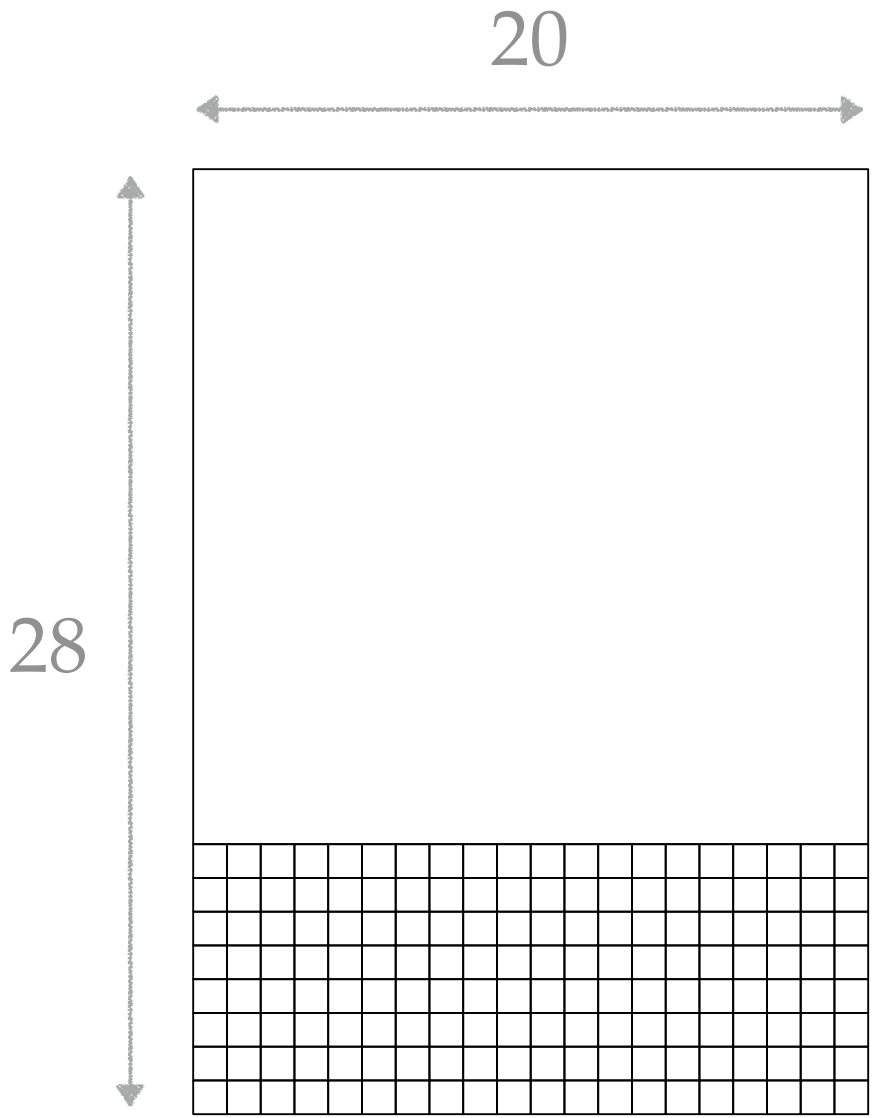
But it also measures CD;

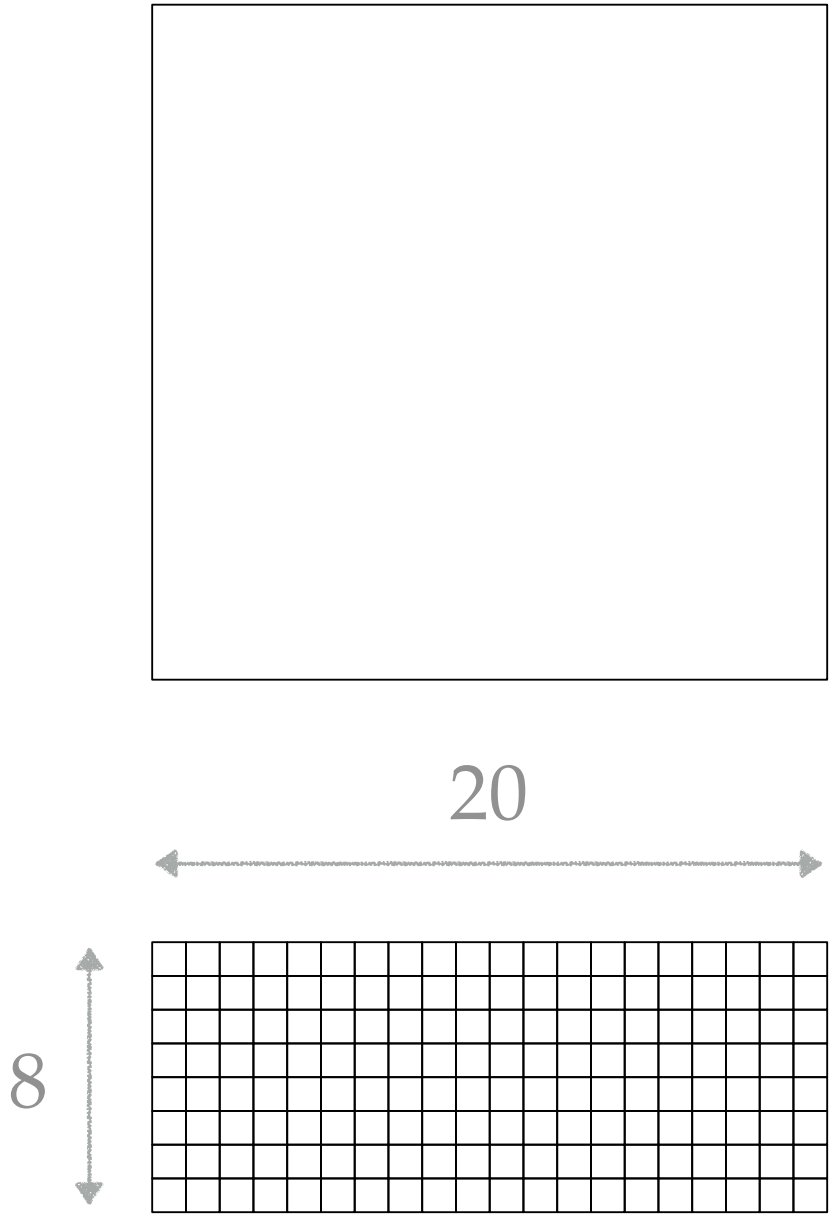
therefore CF measures AB, CD.

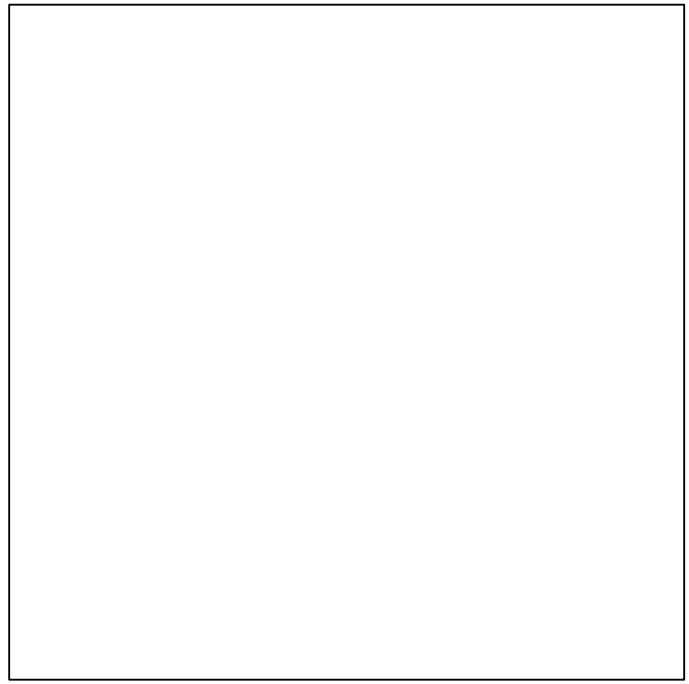
Therefore CF is a common measure of AB, CD.

T. L. Heath, translator. 1908. *The Thirteen books of Euclid's Elements.* Cambridge: Cambridge University Press, page 298.

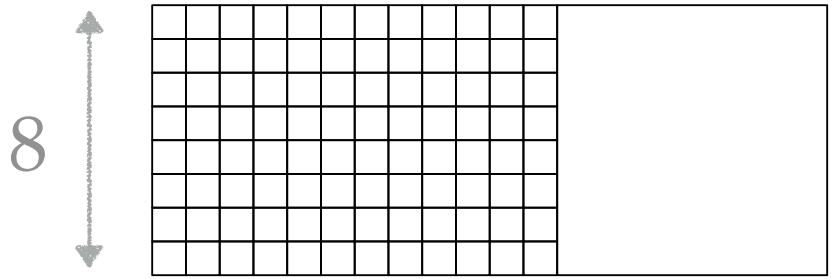

20
28

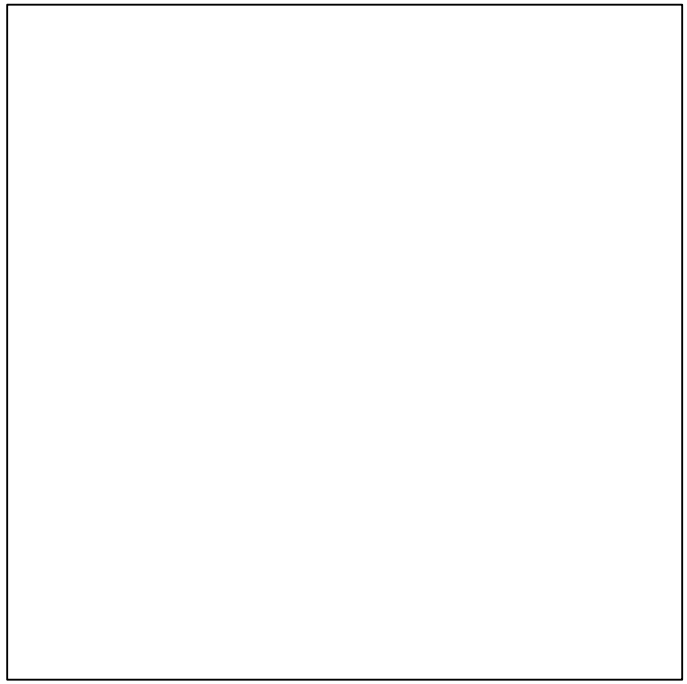

20
28


20
8

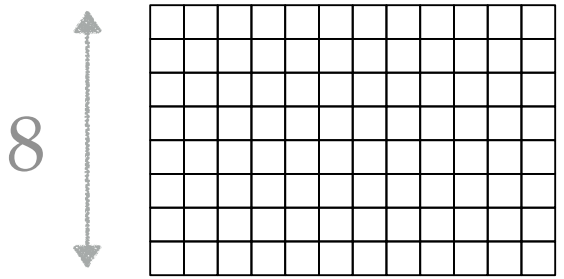


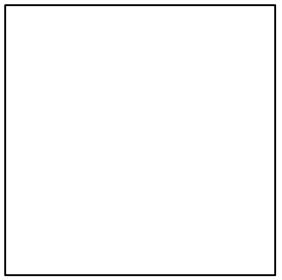

20

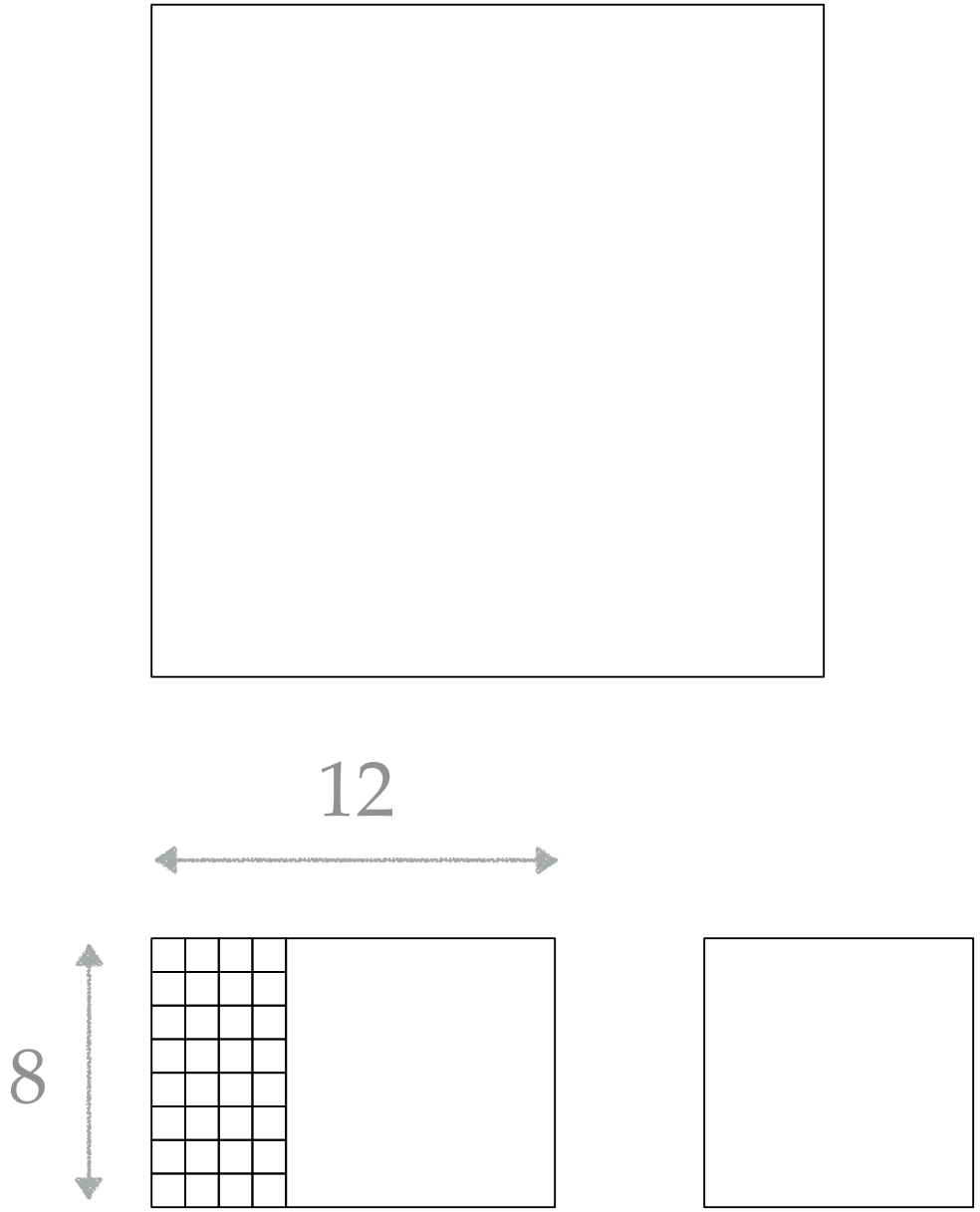

8

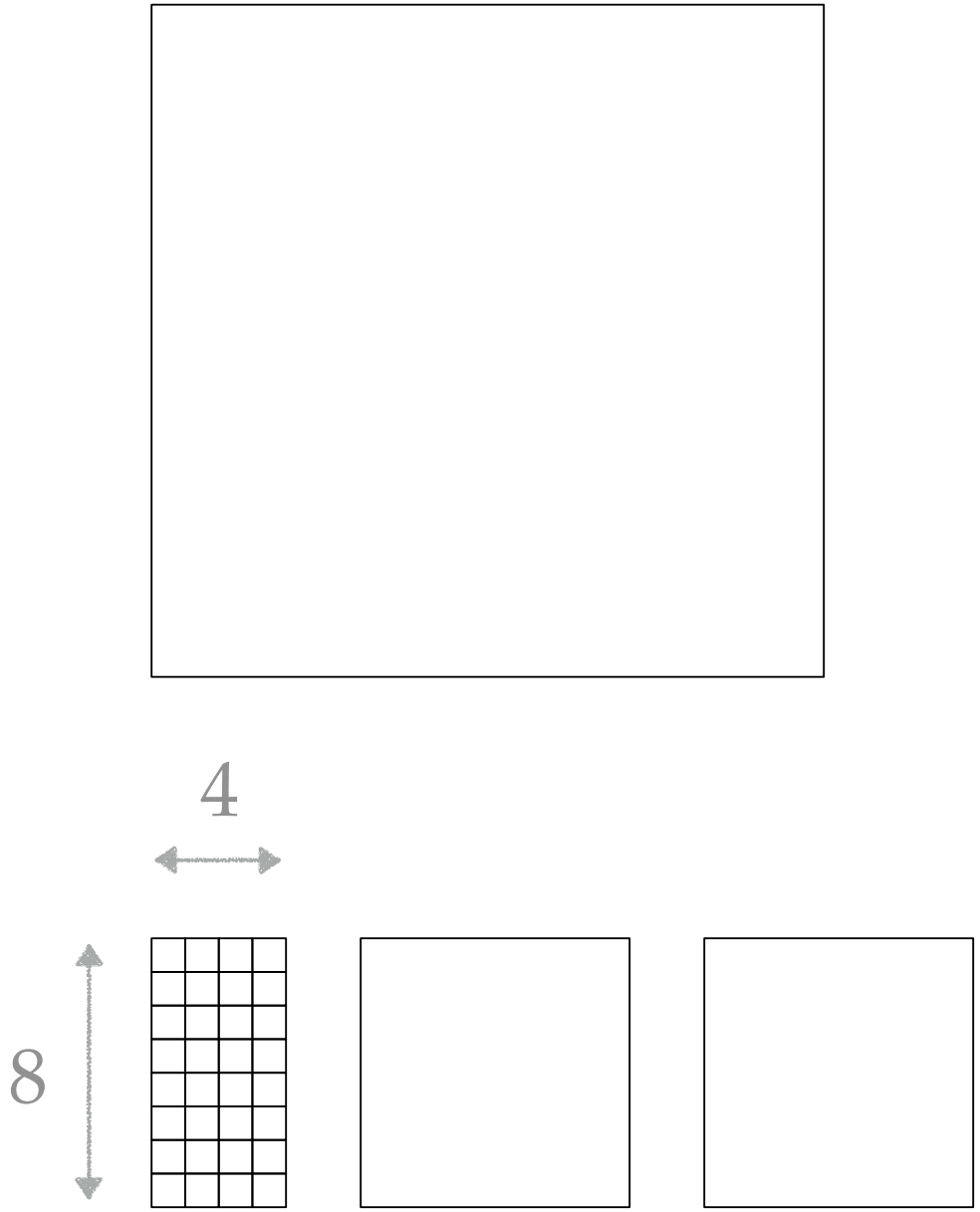


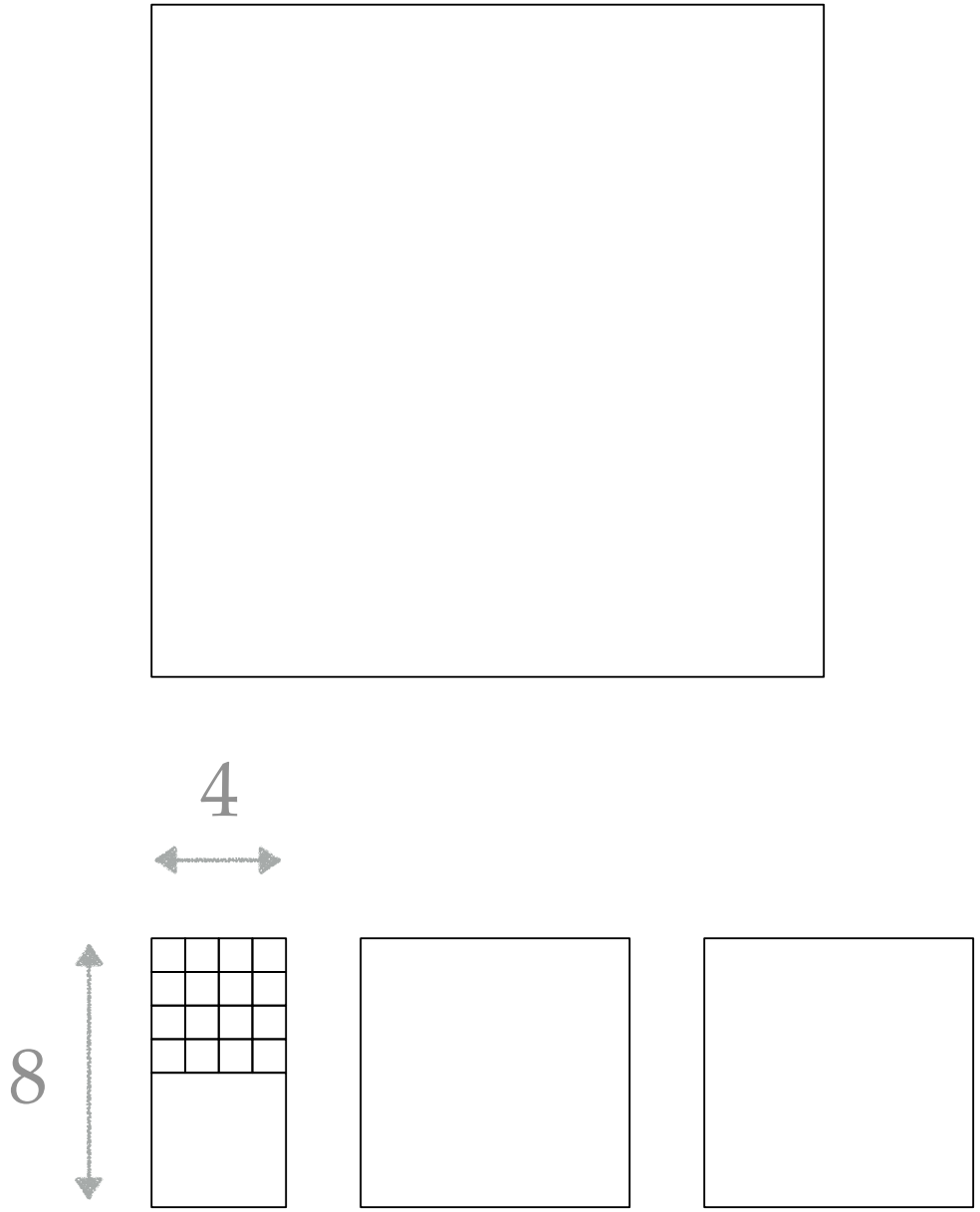

12

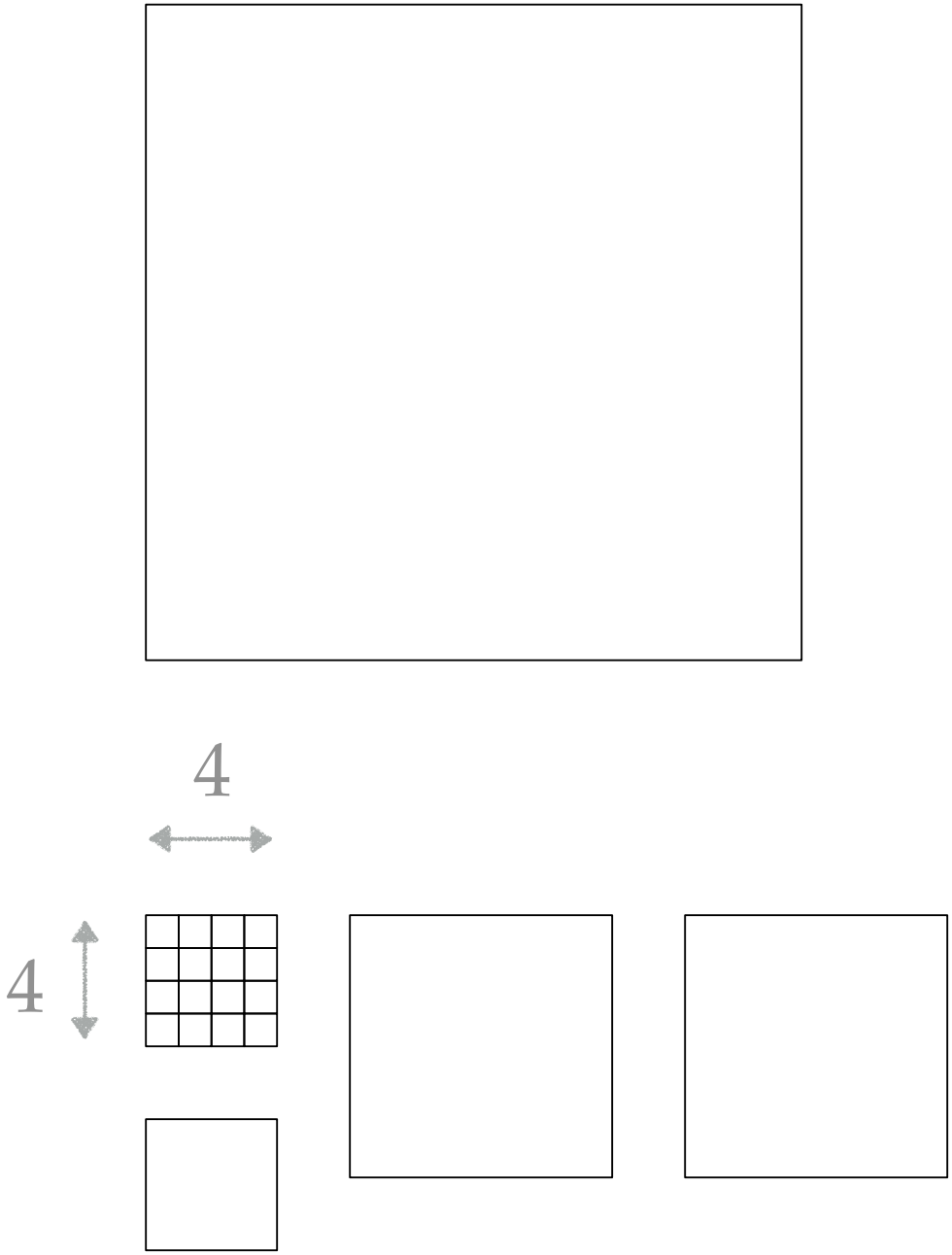

8

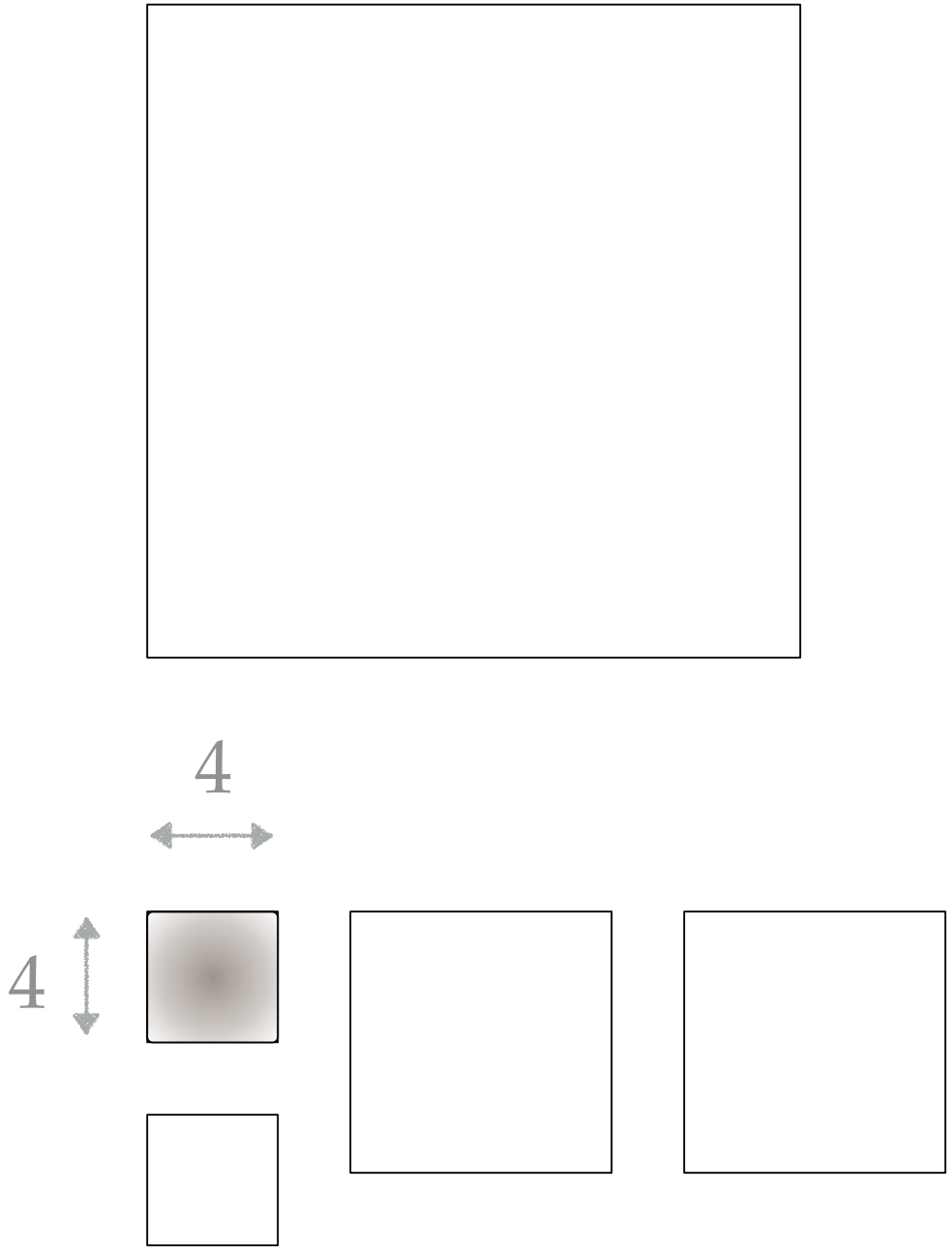


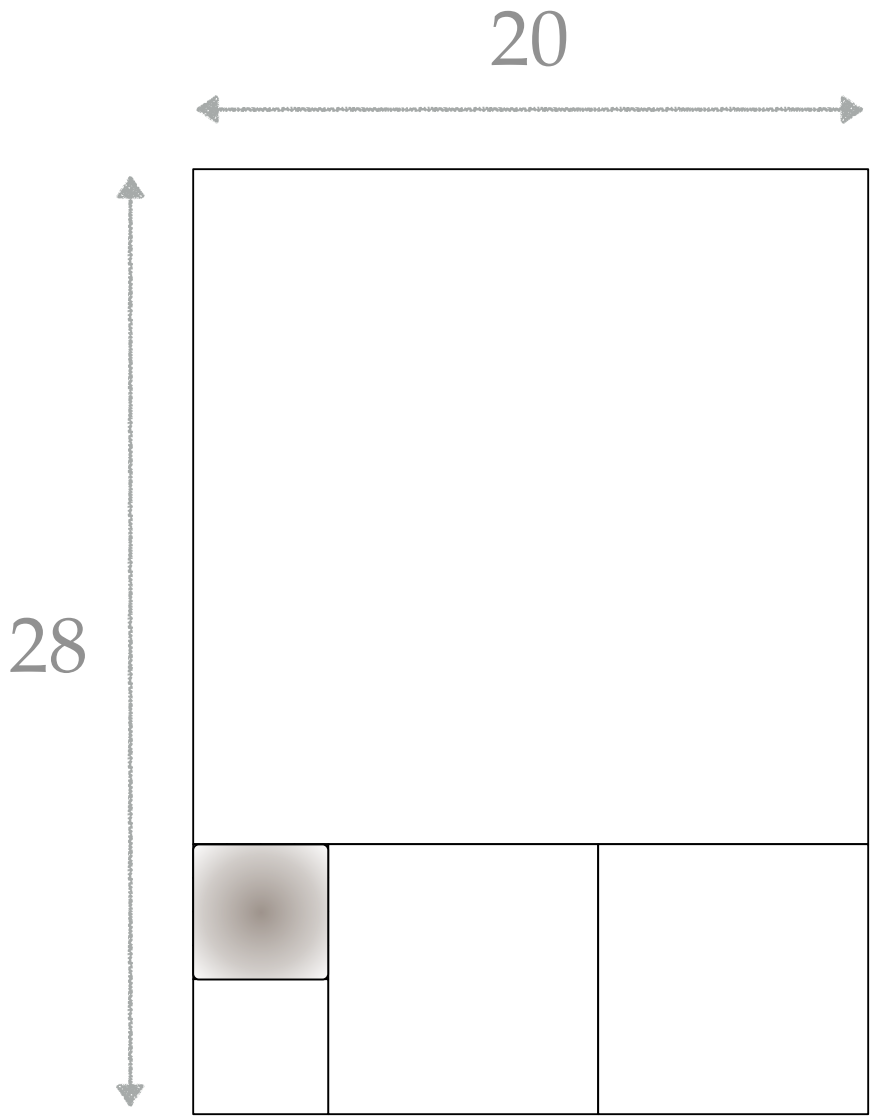

12
8

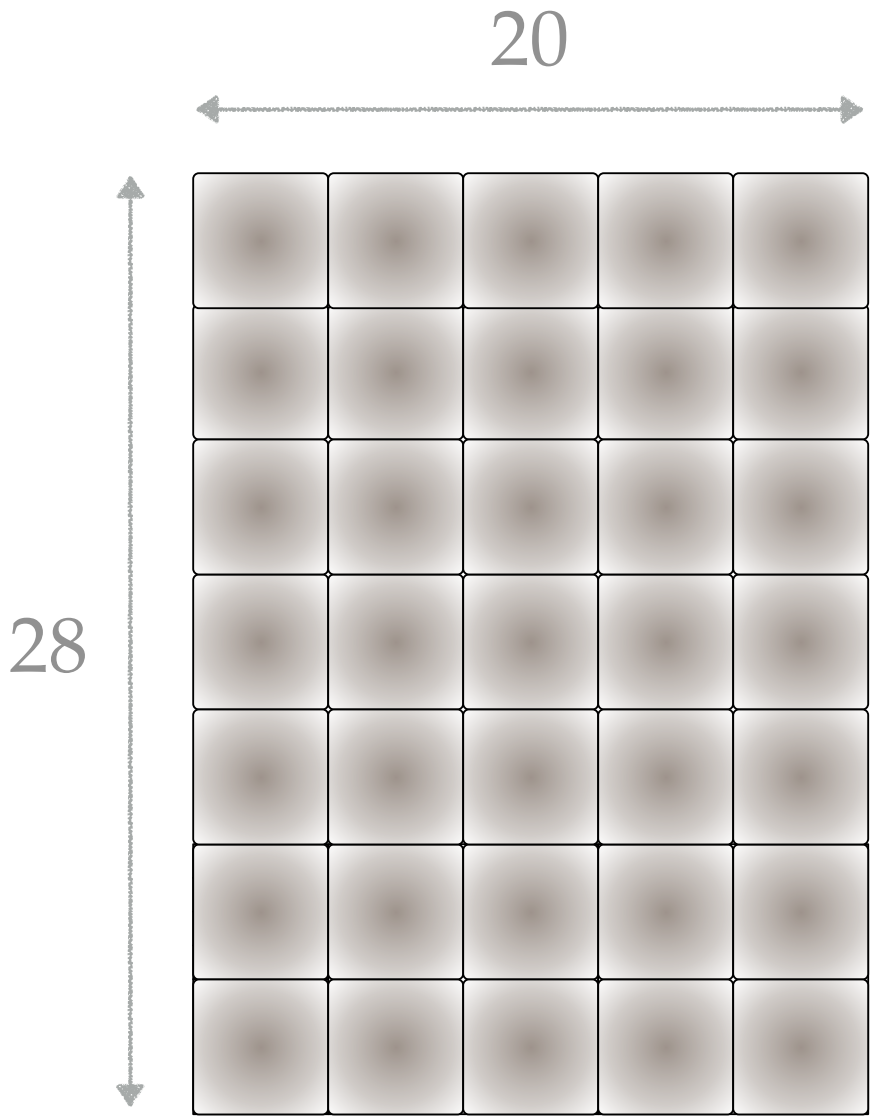

4
8

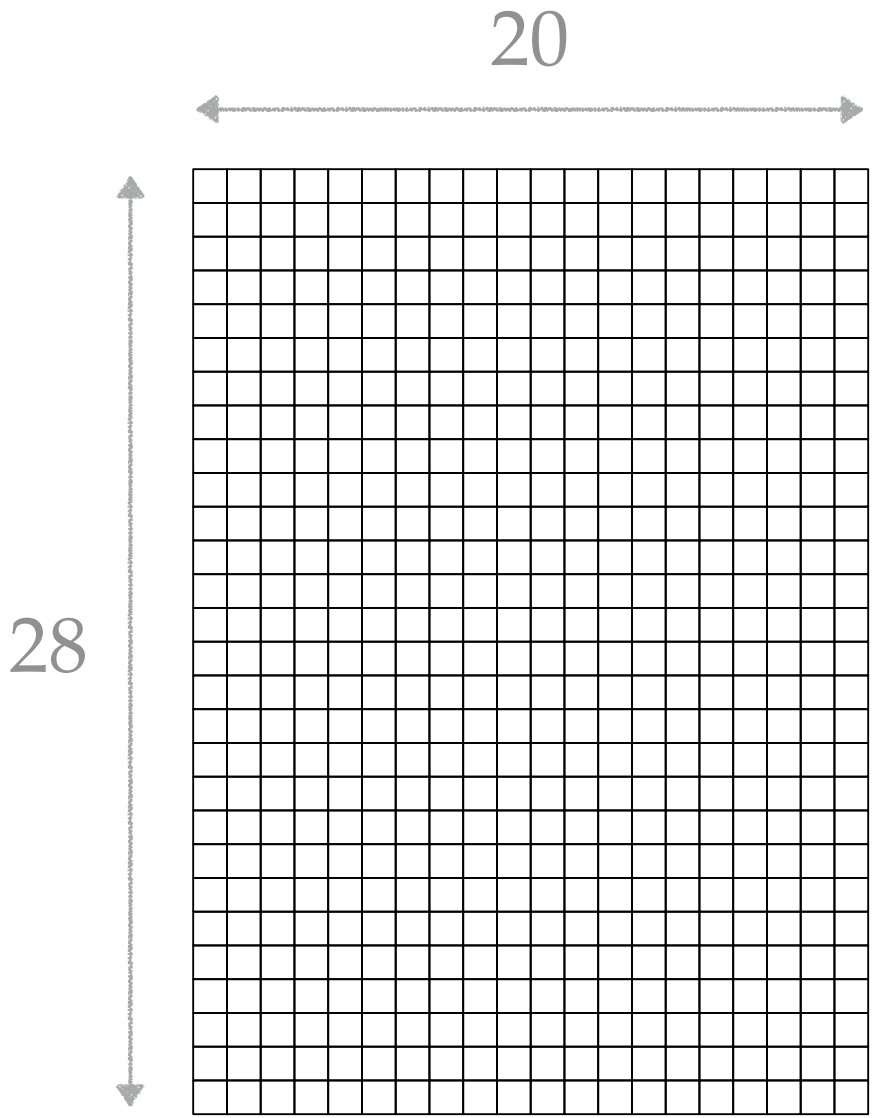

4
8

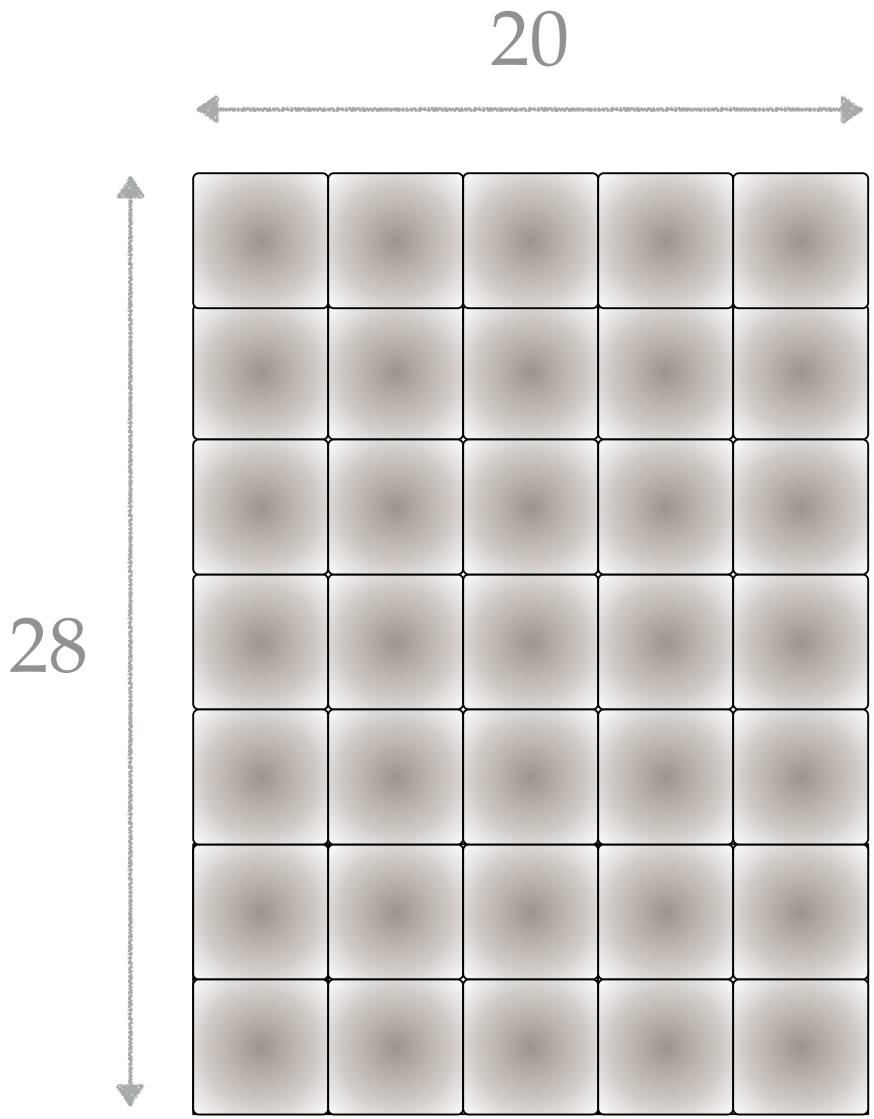

4
4


4
4


20
28


20
28


20
28


20
28

```
let rec gcd_euclid_0 a b =
  if a < b
  then gcd_euclid_0 b a
  else if a = b
       then a
       else gcd_euclid_0 b (a - b) ;;
```


find this code in: `gcd_versions.ml`

```
let rec gcd_euclid_1 a b =
  if a < b
  then gcd_euclid_1 b a
  else if b = 0
       then a
       else gcd_euclid_1 b (a - b) ;;
```


find this code in: `gcd_versions.ml`

```
let rec gcd_euclid_2 a b =
  if a < b
  then gcd_euclid_2 b a
  else if b = 0
       then a
       else gcd_euclid_2 b (a mod b) ;;
```


find this code in: `gcd_versions.ml`

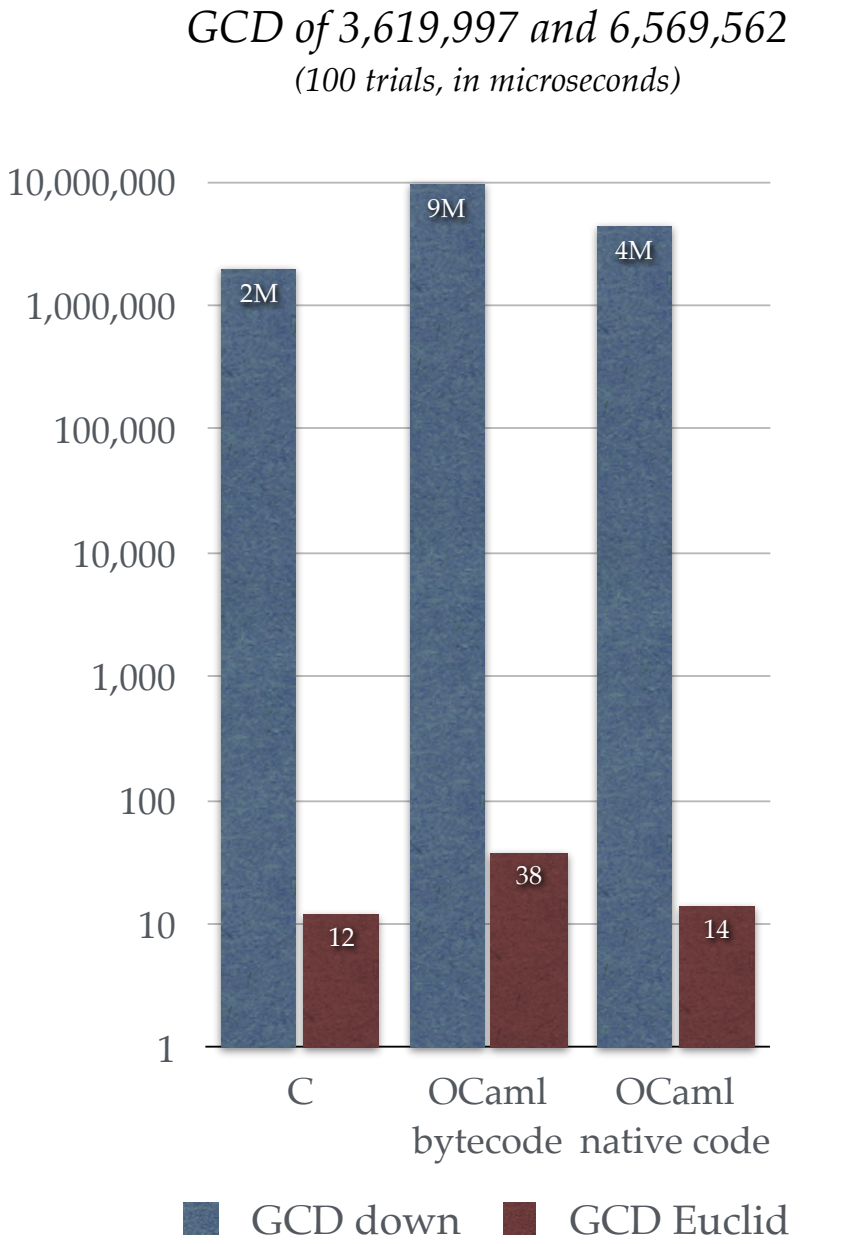
```
let rec gcd_euclid a b =
  if a < b
  then gcd_euclid b a
  else if b = 0
       then a
       else gcd_euclid b (a mod b) ;;
```


find this code in: `gcd_versions.ml`

```
let rec gcd_euclid a b =
  if b = 0
  then a
  else gcd_euclid b (a mod b) ;;
```


find this code in: `gcd_versions.ml`

```
let rec gcd_euclid a b =
  if b = 0
  then a
  else gcd_euclid b (a mod b) ;;
```



find this code in: `gcd_versions.ml`

There is more
than one way to
solve a problem.

There is more
than one way to
solve a problem.

Some ways are
better than others.

There is more than one way to solve a problem.

Some ways are better than others.

succinctness

efficiency

readability

maintainability

provability

testability

There is more than one way to solve a problem.

Some ways are better than others.

succinctness

efficiency

readability

maintainability

provability

testability

beauty

```
let rec gcd_euclid a b =
  if b = 0
  then a
  else gcd_euclid b (a mod b) ;;
```

vs.

```
#include <stdio.h>

#define MIN(a, b) ((a) < (b) ? (a) : (b))

unsigned gcd_down(unsigned a, unsigned b)
{
  unsigned guess;
  for (guess=MIN(a, b); guess>1; guess--) {
    if ((a % guess == 0) && (b % guess == 0))
      break;
  }
  return guess;
}
```


find this code in: `gcd_versions.ml`

What CS51 teaches

1. Software development practice
2. Engineering design principles
3. Fundamental notions of computation
4. Software design concepts

1. Software development practice

Managing a development system

Version control for tracking and collaboration

Compiling complex projects

Unit testing

Invariants

2. Engineering design principles

engineering: producing (software) solutions with desirable properties along multiple criteria

2. Engineering design principles

engineering: producing (software) solutions with desirable properties along multiple criteria

There is more than one way to solve a problem.

Some ways are better than others.

- succinctness
- efficiency
- readability
- maintainability
- provability
- testability
- *beauty*

2. Engineering design principles

Edict of intention:

Express your intentions well.

Edict of irredundancy:

Never write the same code twice.

Edict of decomposition:

Carve software at its joints.

Edict of prevention:

Make the illegal inexpressible.

Edict of compartmentalization:

Limit information to those with a need to know.

3. Fundamental notions of computation

Expressions and the linguistics of programming

Values, types, and type inference

Naming and scope

Semantics

- substitution, environment

Complexity

- order, recurrences

4. Software design concepts

Higher-order functions and *functional* programming

Polymorphism and *generic* programming

Handling anomalous conditions

Algebraic data types

Abstract data types and *modular* programming

Mutable state and *imperative* programming

Loops and *procedural* programming

Infinite data structures and *lazy* programming

Decomposition and *object-oriented* programming

The language: OCaml

Paradigms:

- first-order and higher-order functional programming
- imperative programming
- generic programming
- lazy programming
- object-oriented programming
- concurrent programming

Concepts:

- substitution & environment models of evaluation
- static types, type inference, polymorphism
- abstract data types, interfaces, modules
- encapsulation, classes, subtyping, inheritance
- parallelism, concurrency, synchronization

OCaml impact

F# (Microsoft)

Rust

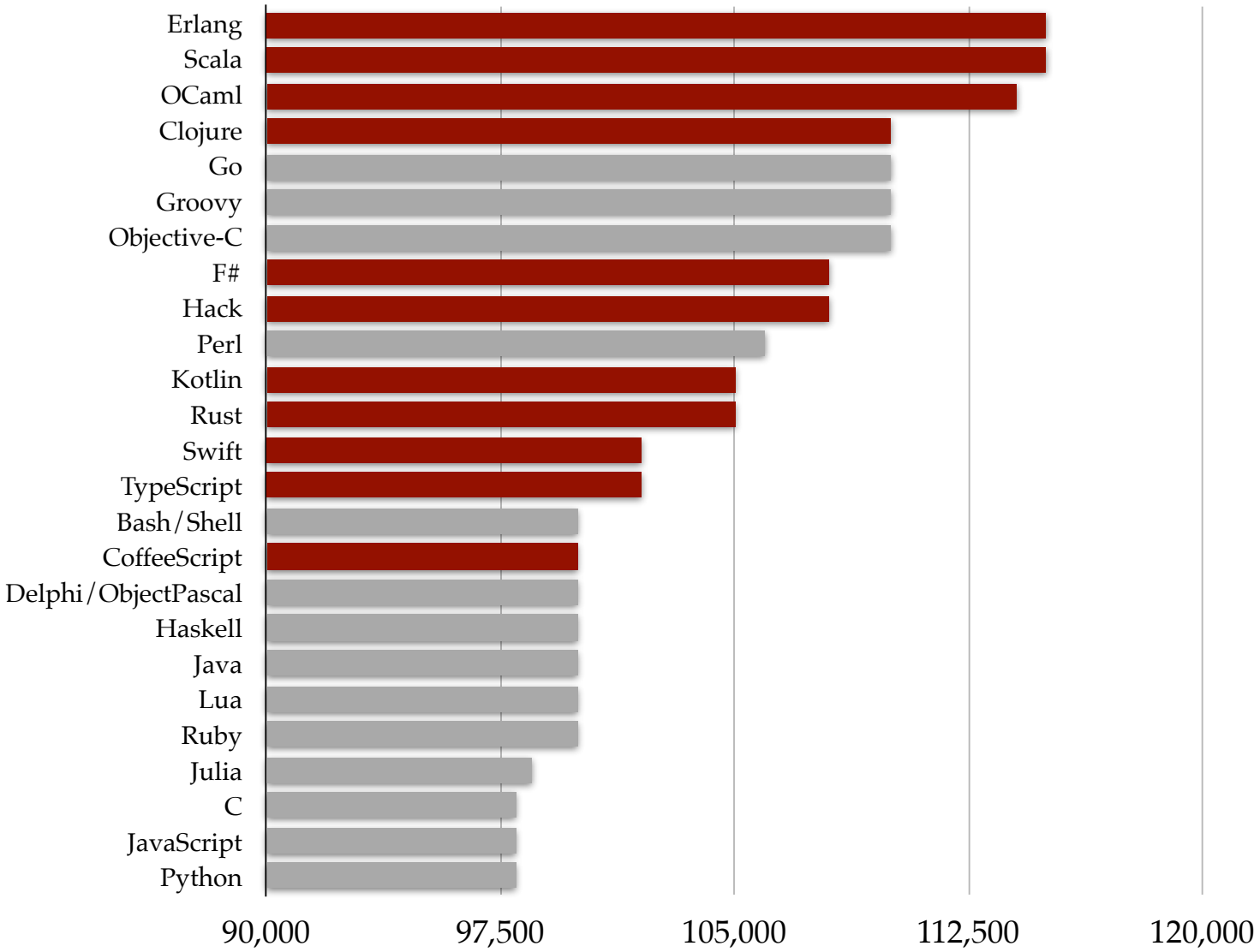
Reason (Facebook)

Elm

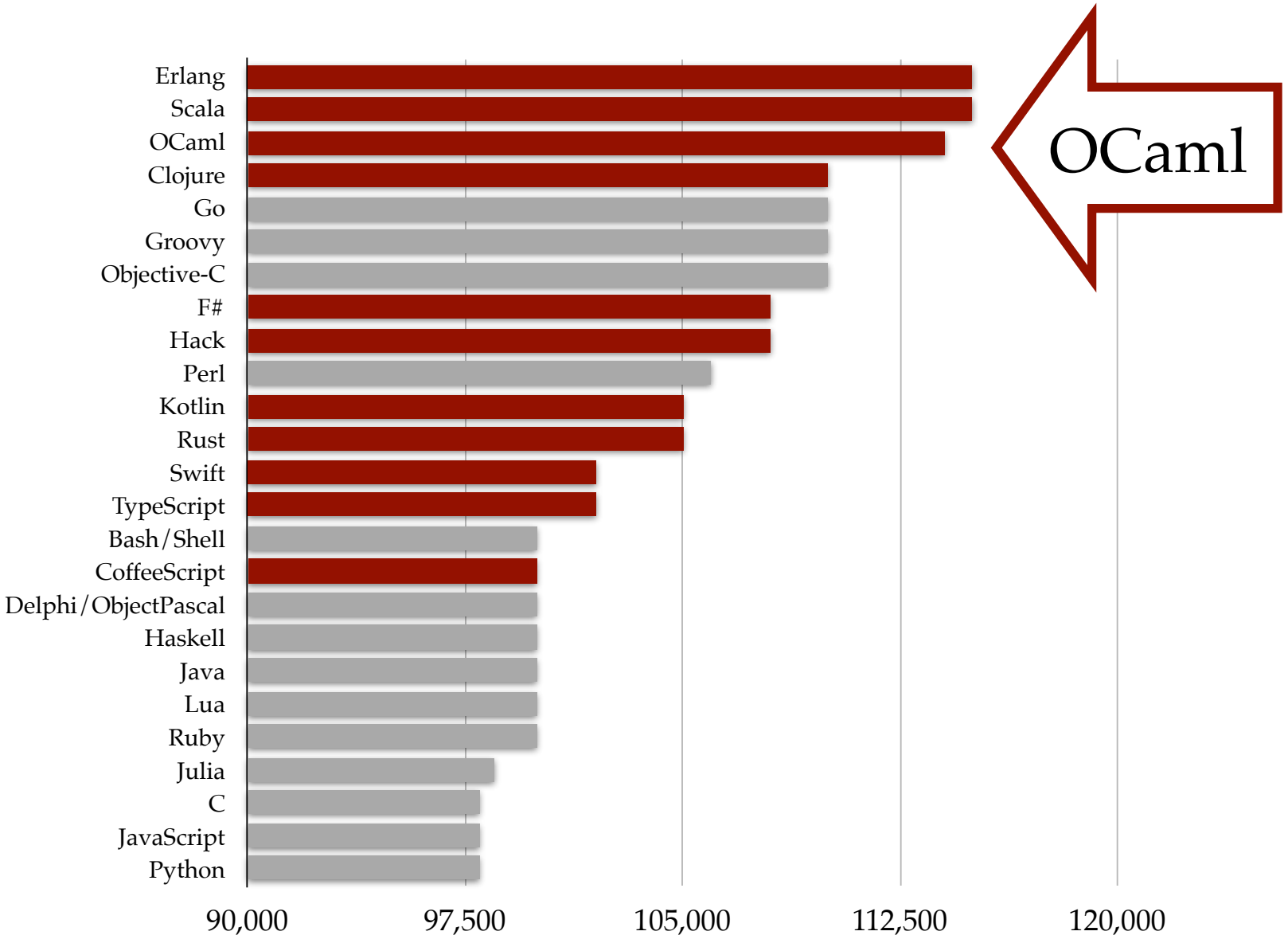
Swift (Apple)

Haskell

...and many others

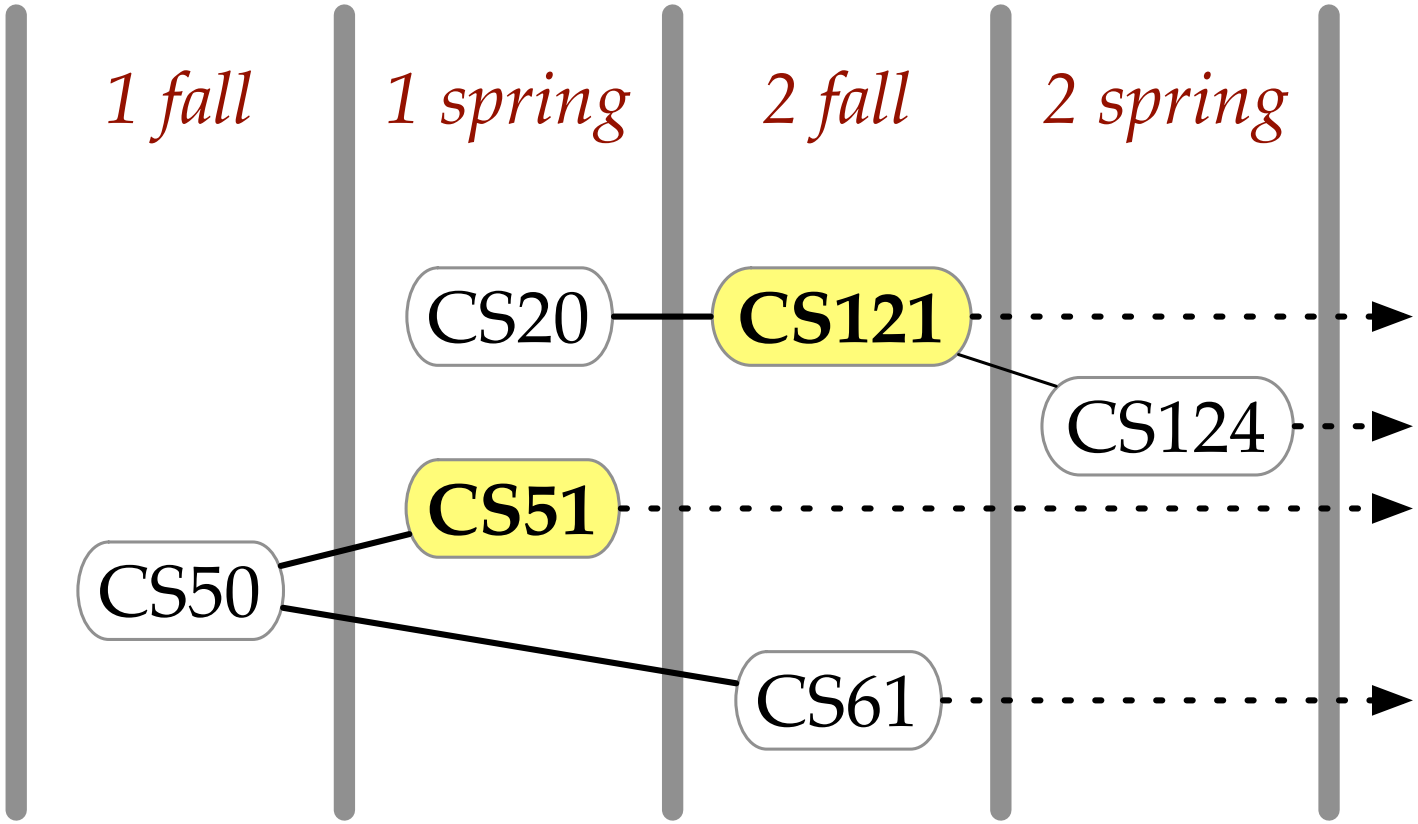


US average salary by technology, StackOverflow Developer Survey 2018



US average salary by technology,
StackOverflow Developer Survey 2018

Other CS courses

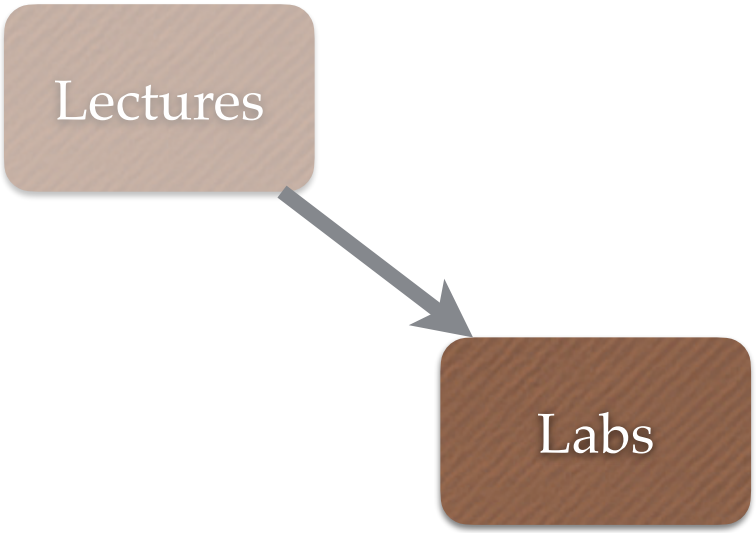


Course structure

Lectures

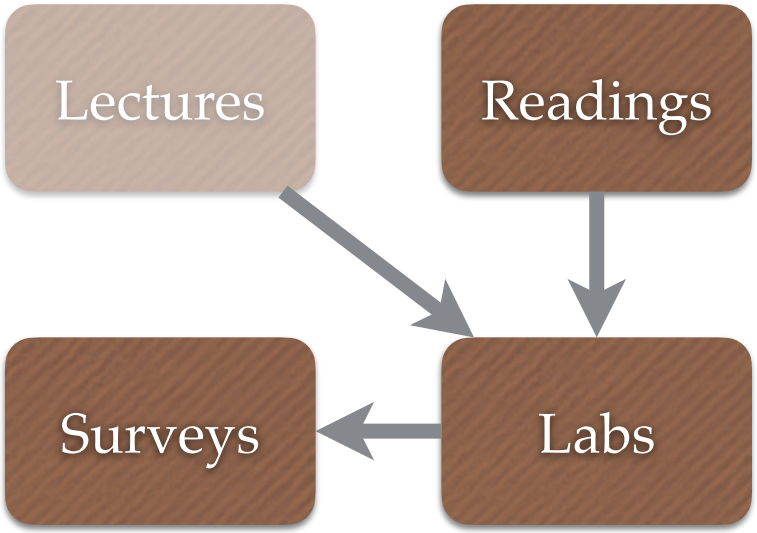
Video: streamed and on web site
Guest lectures

Course structure



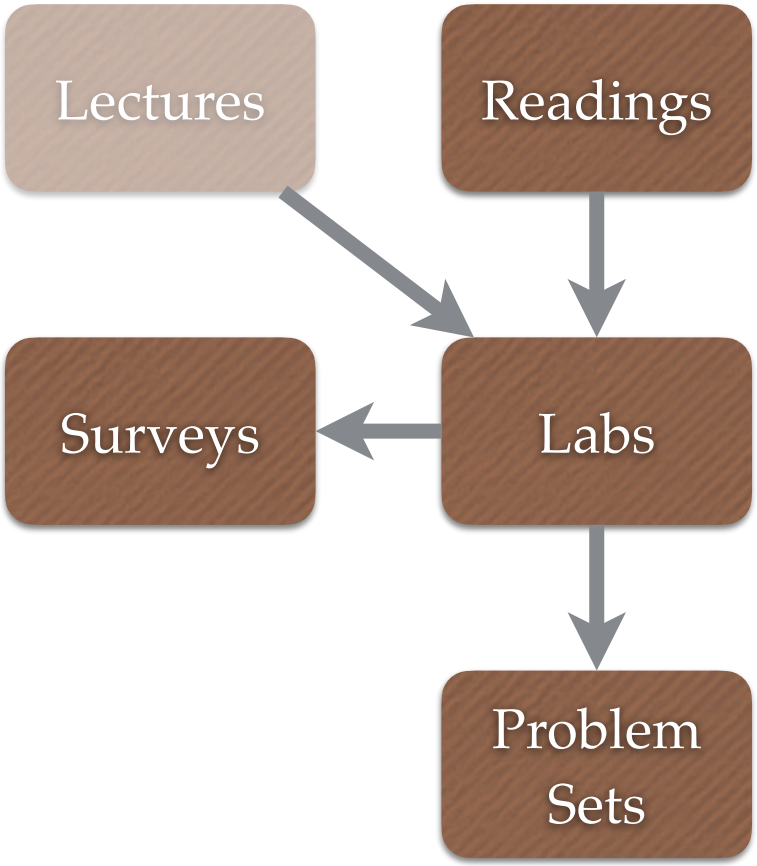
Pair programming labs
Most TTh starting February 2
Northwest Building basement
Two lab slots: 10:30–11:45, 4:30–5:45
Virtual quiz, Sundays

Course structure



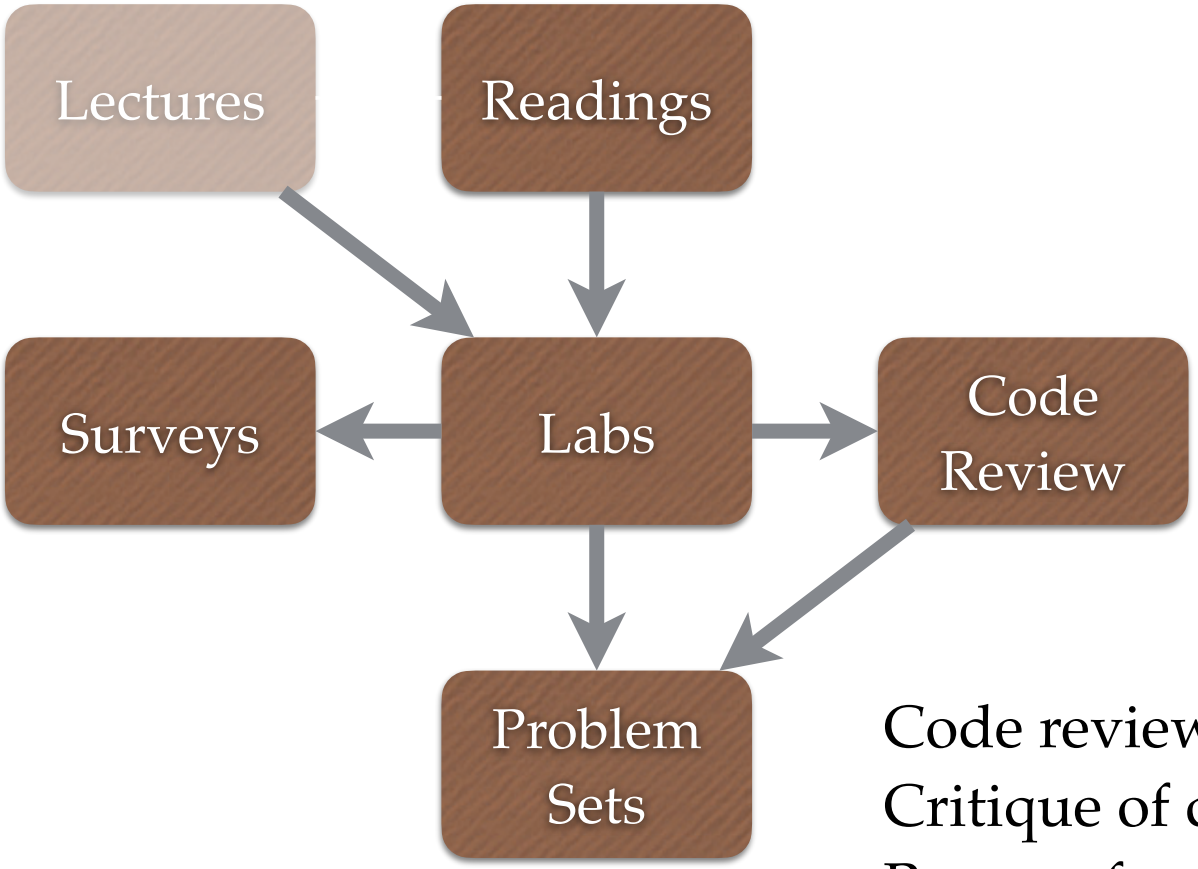
Readings (`book.cs51.io`) to prepare for labs
Post-lab peer- and self-evaluation surveys every few labs

Course structure



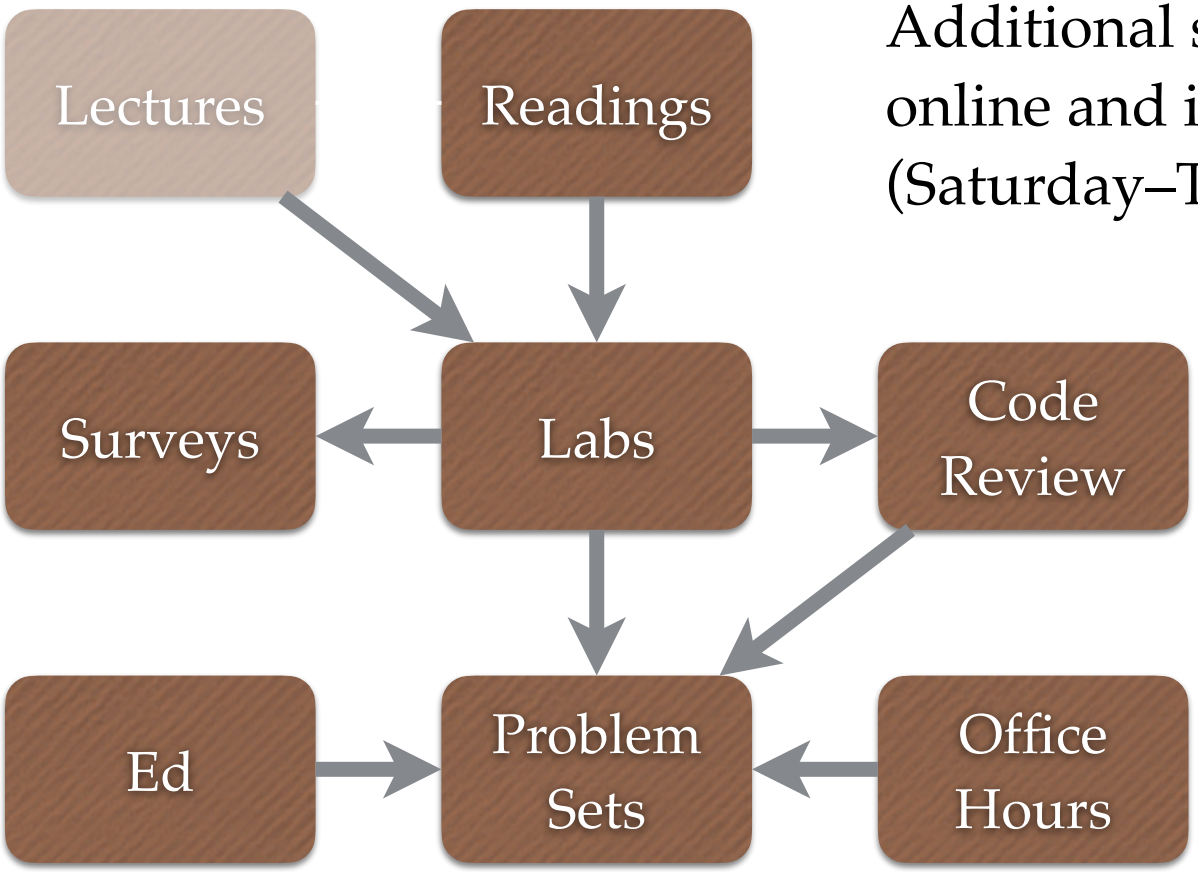
Eight problem sets
Due Wednesdays (typically)

Course structure



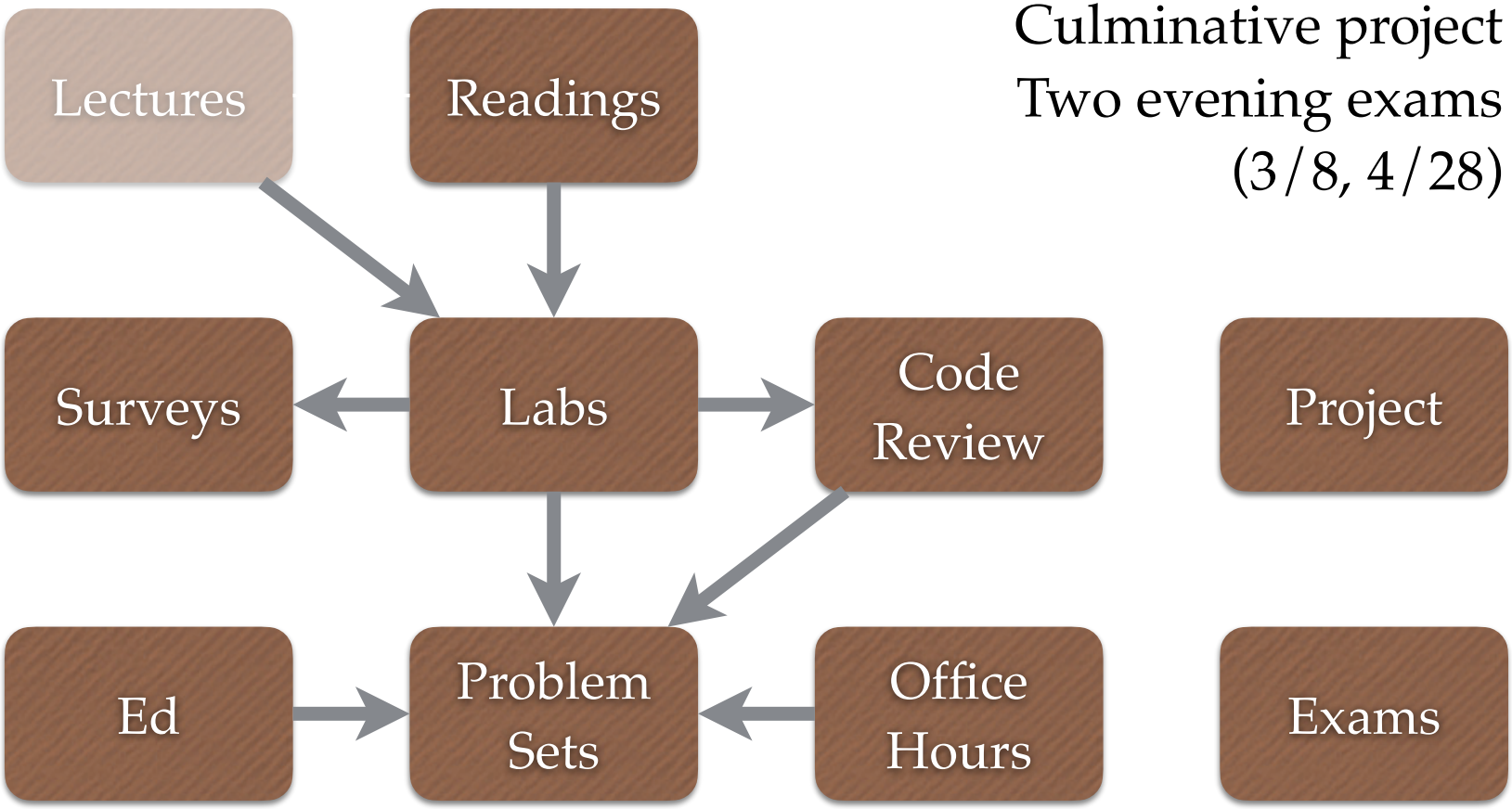
Code reviews Fridays
Critique of code from labs
Prepare for problem sets

Course structure



Additional sources of help online and in office hours (Saturday–Tuesday pm)

Course structure



Culminative project
Two evening exams
(3/8, 4/28)

CS51 Coffee Klatch

Wednesdays 4:30–5:30

Sign up at

`http://url.cs51.io/coffee`

Head staff

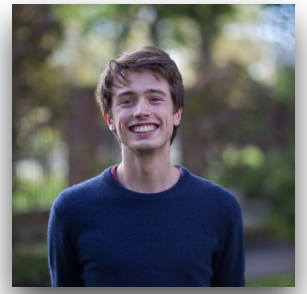
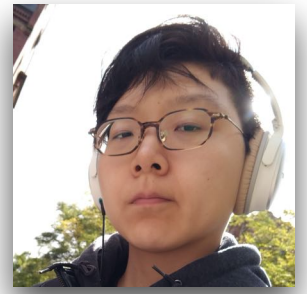
Jordan Barkin
head TF

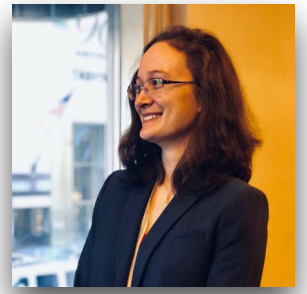
Olivia Graham
head TF

Ahan Malhotra
infrastructure guru










Collaboration policy

Problem sets are done alone or in pairs

Labs are done in pairs and fours

Final project is done alone (with approved exceptions)

Talking together about problem sets and project is *encouraged*: understanding concepts, help finding bugs

Asking for or acquiring solutions from others or revealing solutions to others is expressly *disallowed*

When in doubt, ask me or a TF

Logistics

http://cs51.io

Extra help

Grading

Submitting coursework

Absence policy

Late policy

Laptop policy

Collaboration and academic integrity

Auditing

Simultaneous enrollment

Course climate

Mental health

Accommodations for special requirements

For next time...

Read the syllabus

Read chapters 1-4 (`book.cs51.io`)

Section for labs (Crimson cart) and for code review (sectioning survey at `section.cs51.io`)

Work on Problem Set 0 (installing the required course software), due Monday 11:59pm

- Office hours to help you get things installed will be listed in CS51 Canvas calendar

Euclid of Alexandria

cs51.io