Lecture 11: Foundations of static analysis

John Wickerson

Compilers

Static analysis

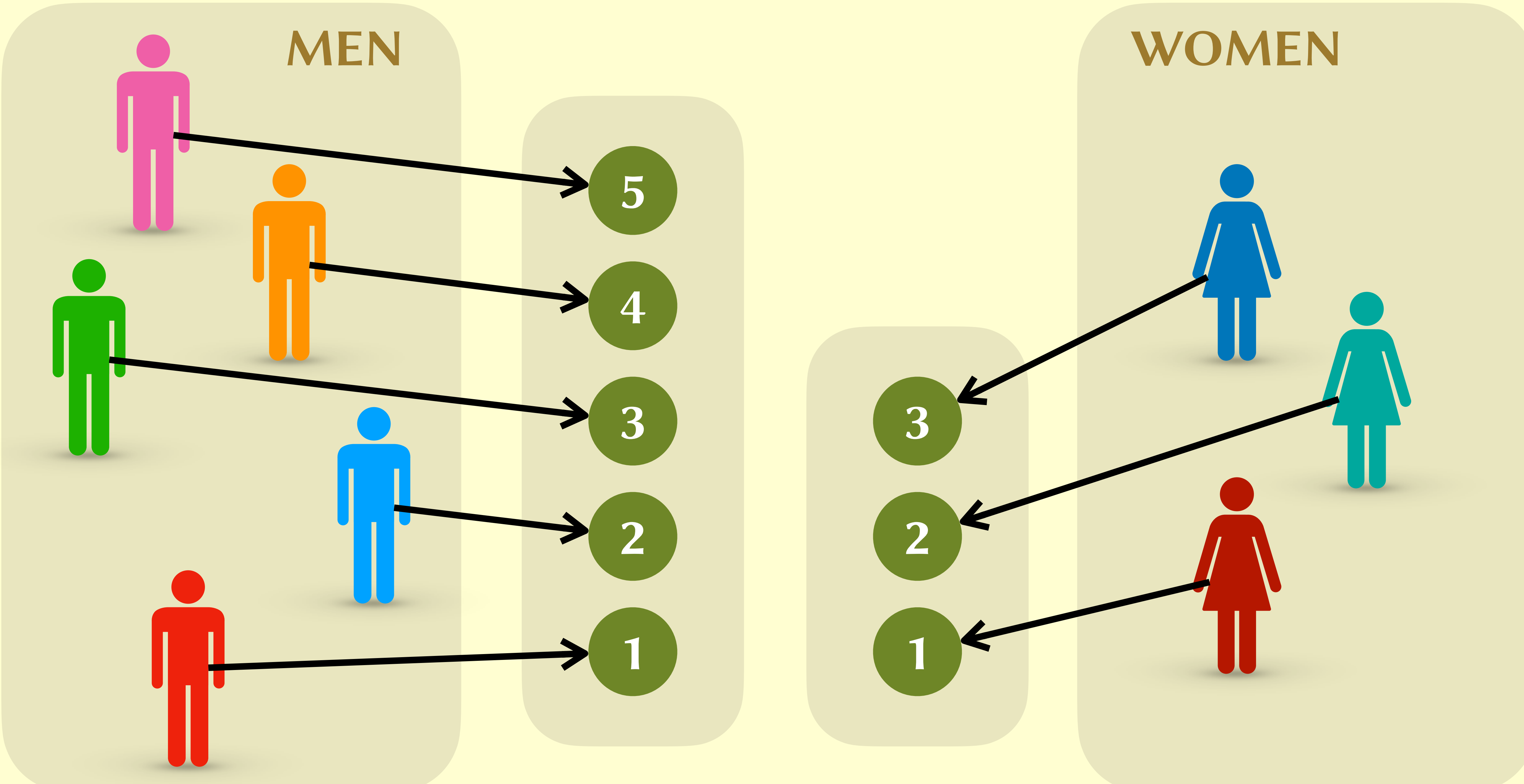
- An **analysis** aims to answer questions about a program, such as:
  - How much memory does it need?
  - How long will it take to run?
  - Which variables can be mapped to the same register?
  - Which instructions can be safely deleted?
- A **static** analysis is done without running the program – it just examines the source code. (A **dynamic** analysis is done while the program is actually running.)

This lecture

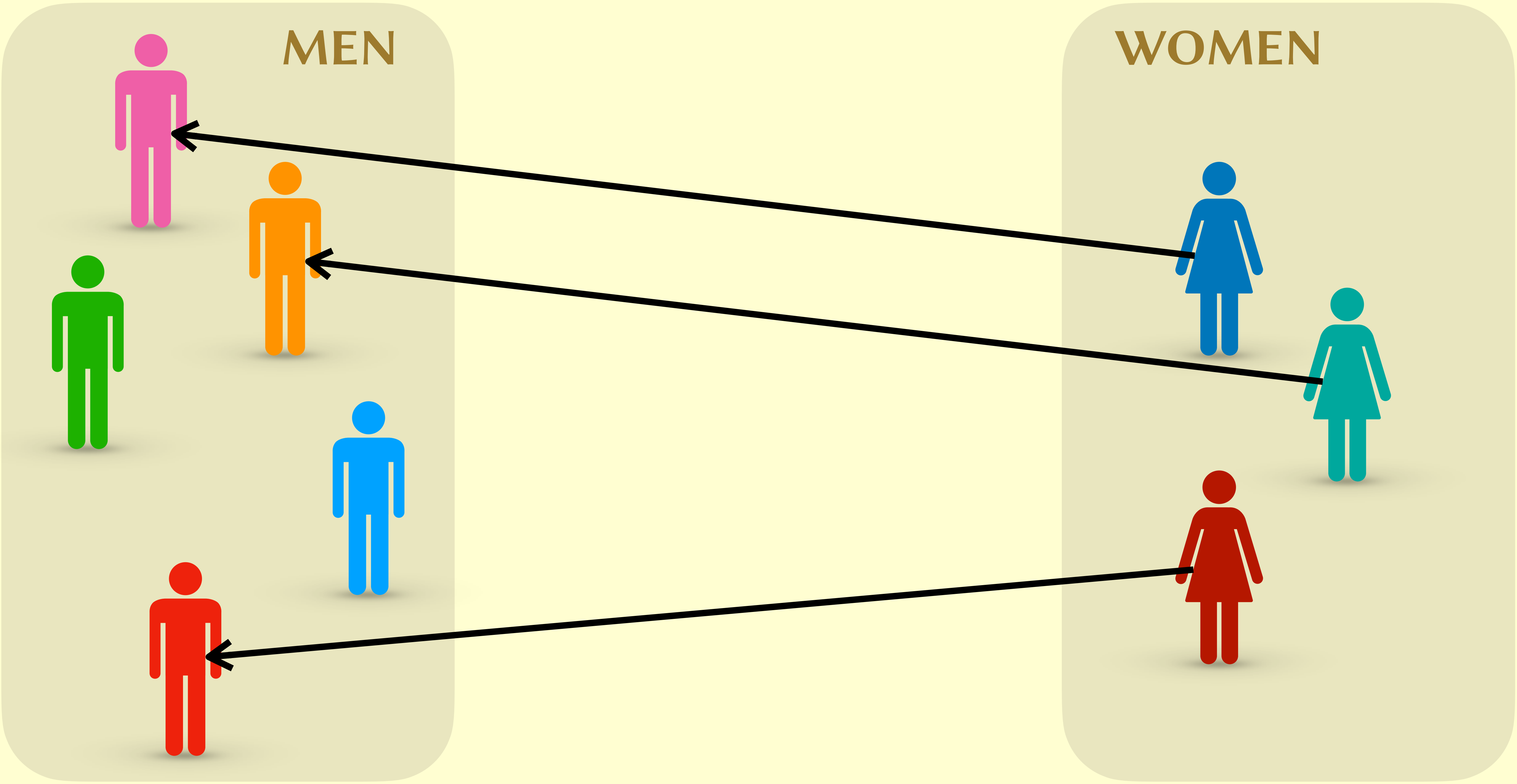
- What are the *fundamental limits* of static analysis?

Class exercise

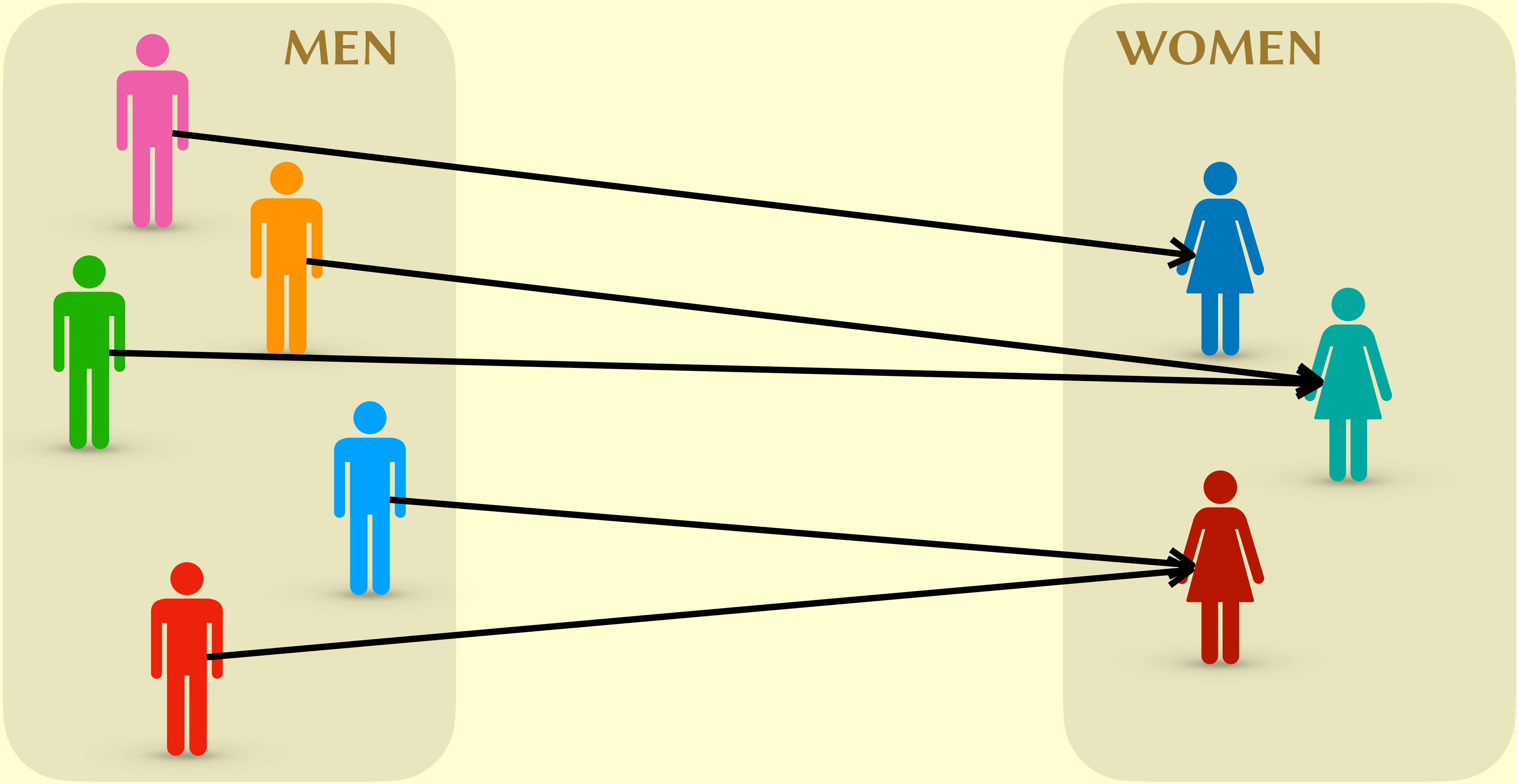
How to count



How to count



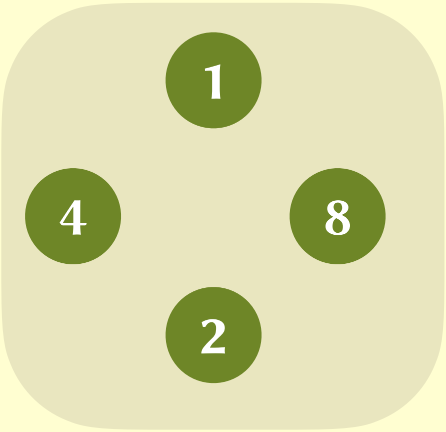
How to count



Terminology

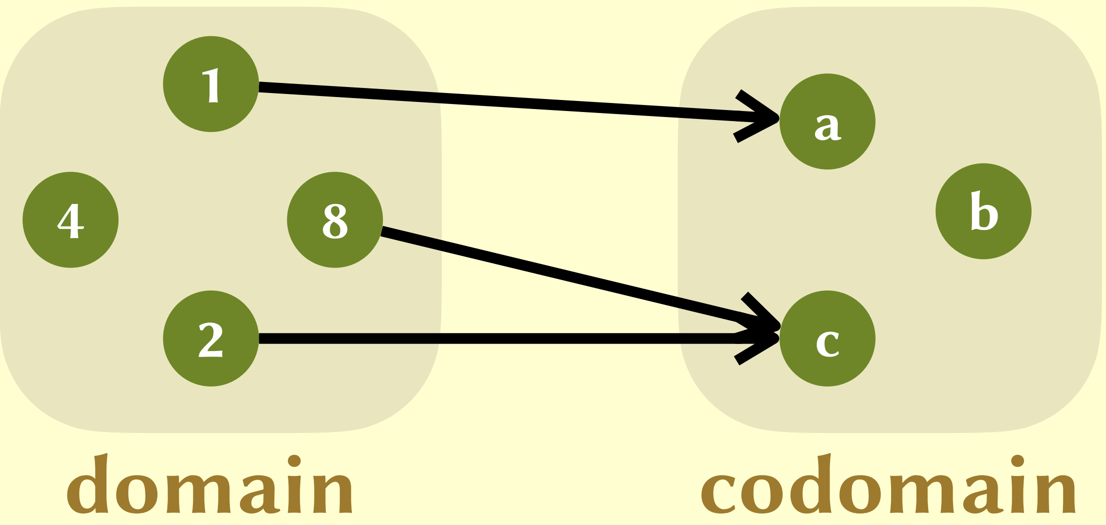
A set

- ... is an *unordered collection of elements.*
- The set {1,2,4,8} can be drawn as:



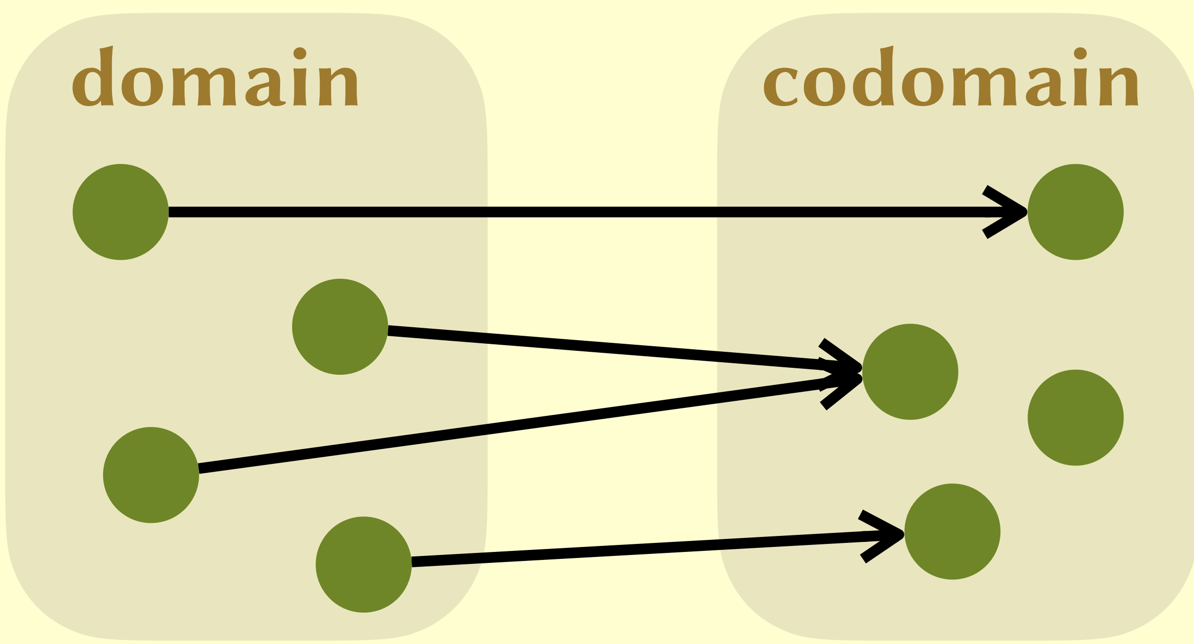
A relation

- … between two sets is a *set of pairs.*
- The first item in each pair is taken from the first set, and the second item in each pair is taken from the second set.
- For instance, one relation between the sets {1,2,4,8} and {a,b,c} is {(1,a), (2,c), (8,c)}.
- It can be drawn as:

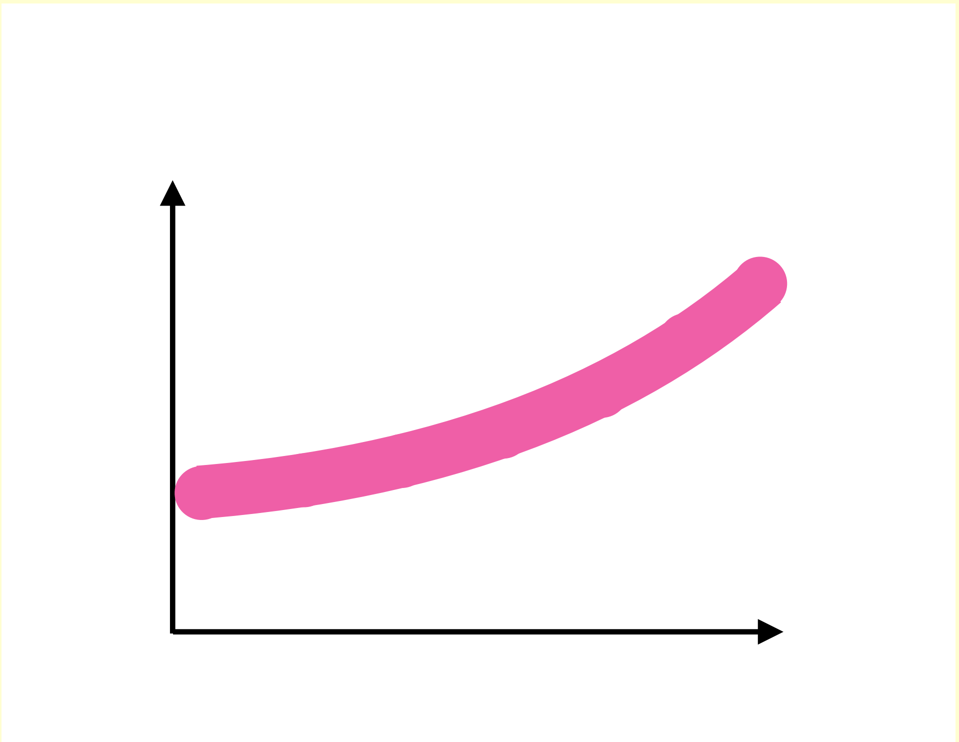


A function

- … is a relation where every element of the **domain** is mapped to one (and only one) element of the **codomain**.

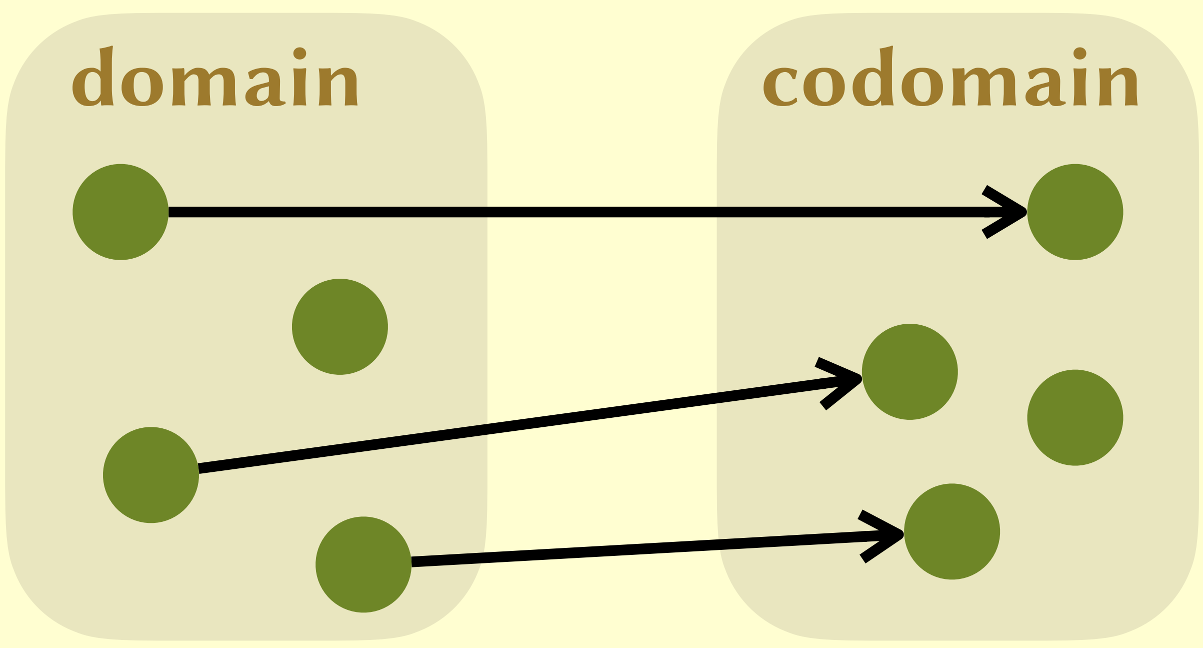


function: ✔

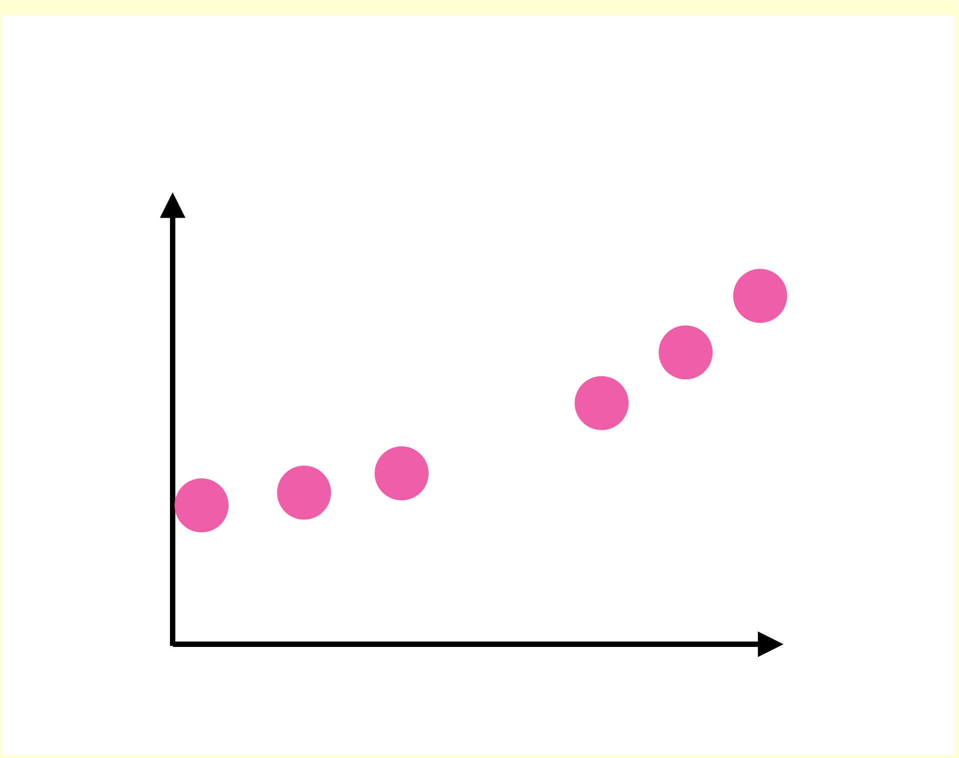


A function

- … is a relation where every element of the **domain** is mapped to one (and only one) element of the **codomain**.

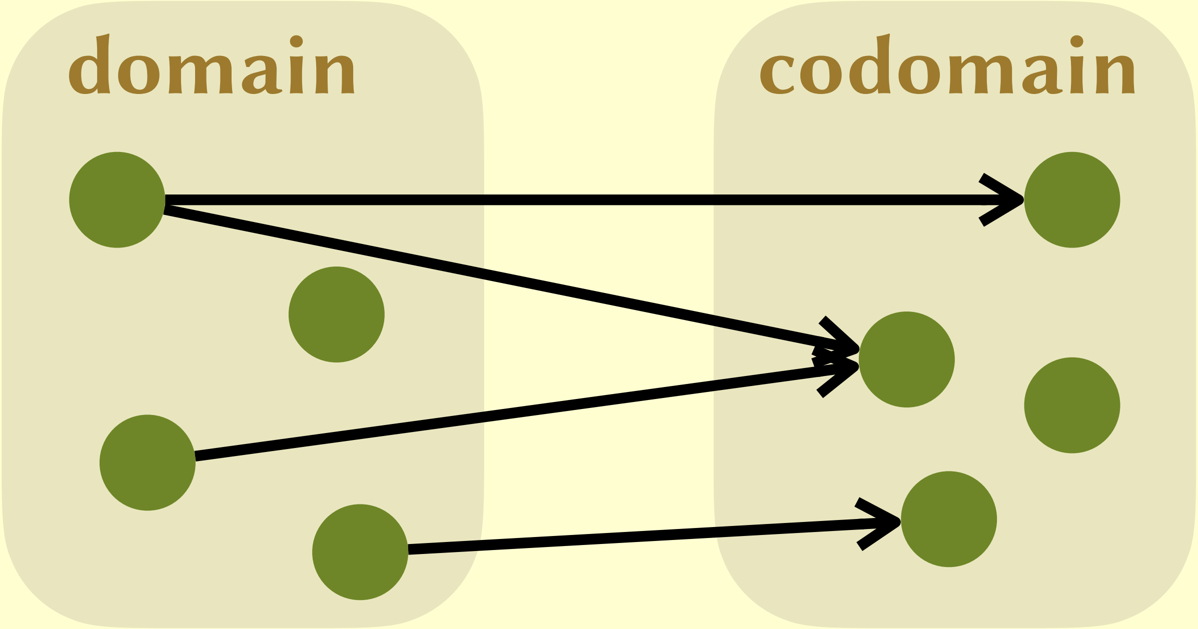


function: ✗

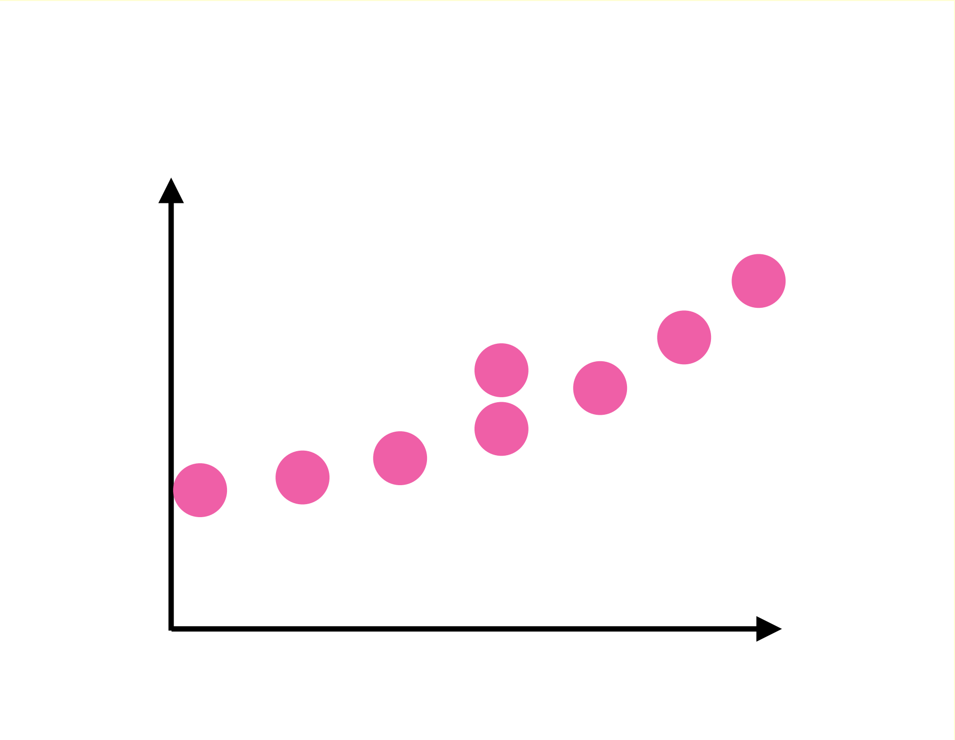


A function

- ... is a relation where every element of the **domain** is mapped to one (and only one) element of the **codomain**.

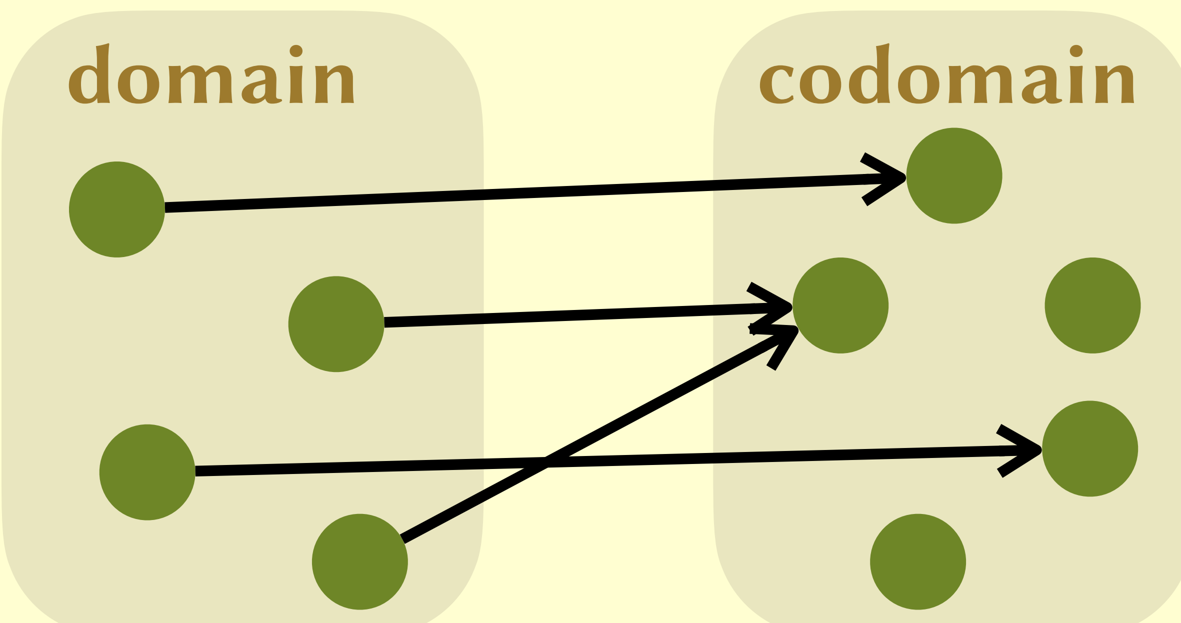


function: ✗



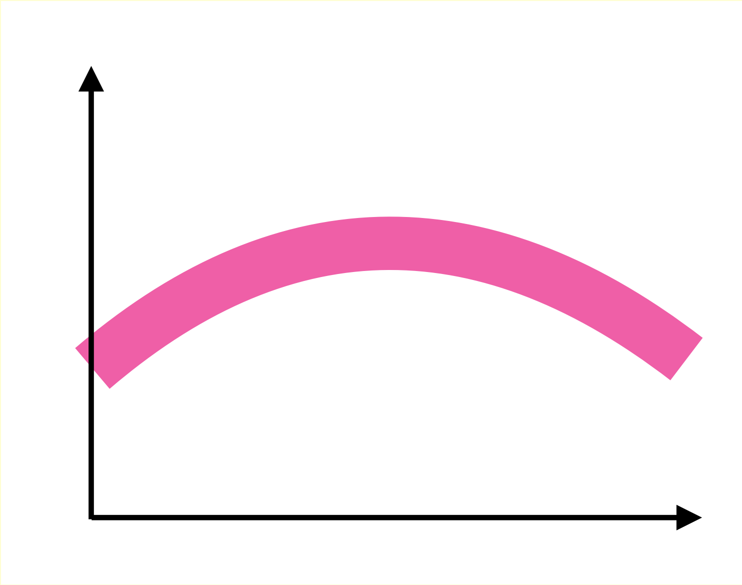
An injective function

- … is a function where every element of the domain is mapped to a different element of the codomain. Also called an **injection**.



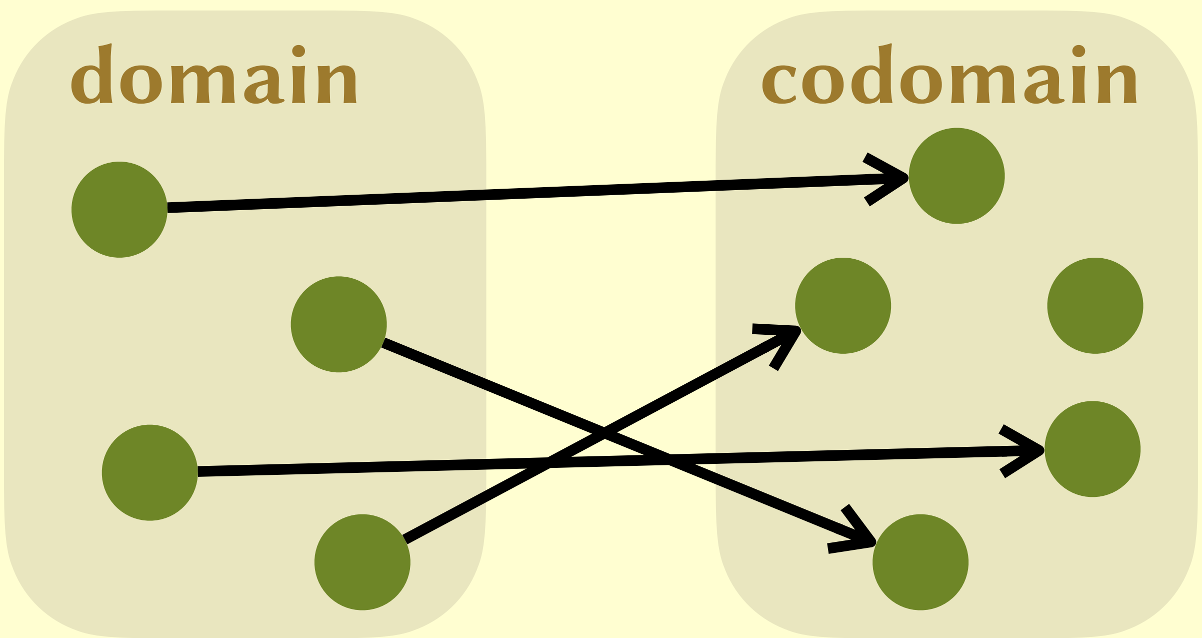
function: ✔

injective: ✘



An injective function

- … is a function where every element of the domain is mapped to a different element of the codomain. Also called an **injection**.

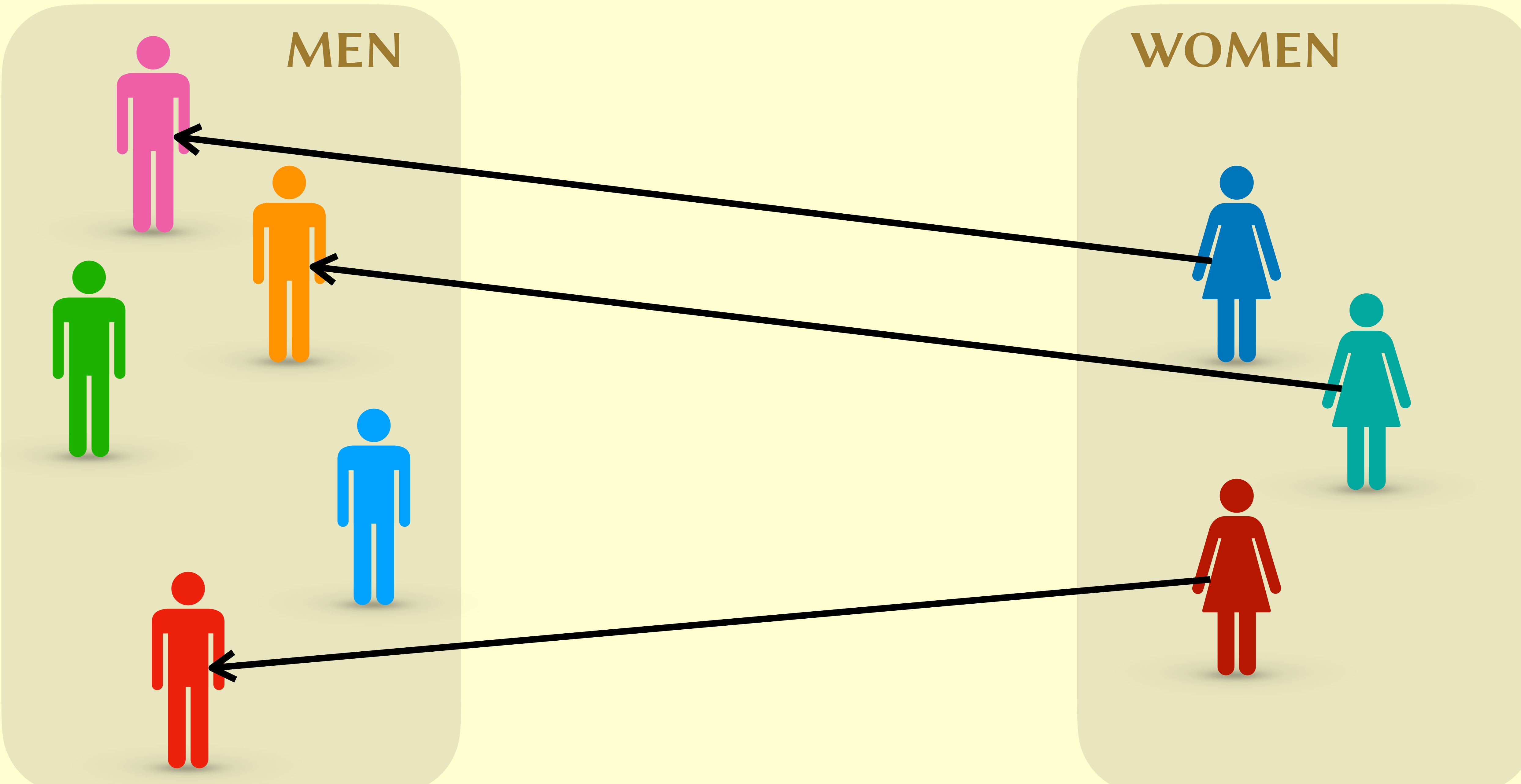


function: ✔

injective: ✔

- **Intuition.** If I can construct an injection, then the codomain is as least as large as the domain.

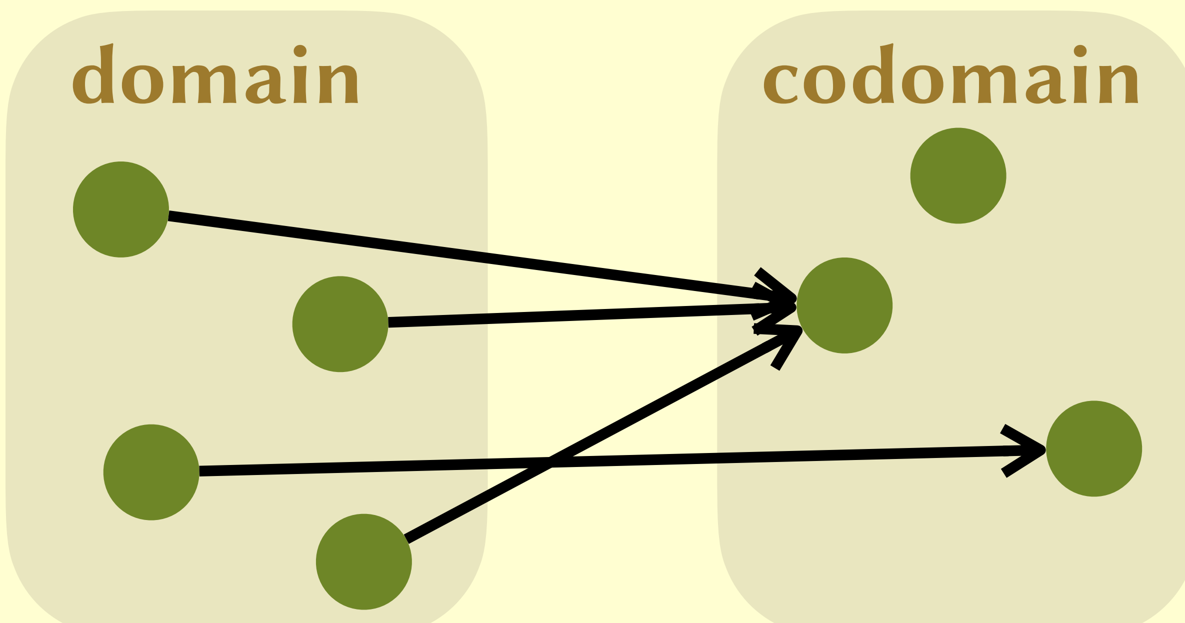
How to count



"There is an **injection** *from the set of women to the set of men, so there are at least as many men as there are women."*

A surjective function

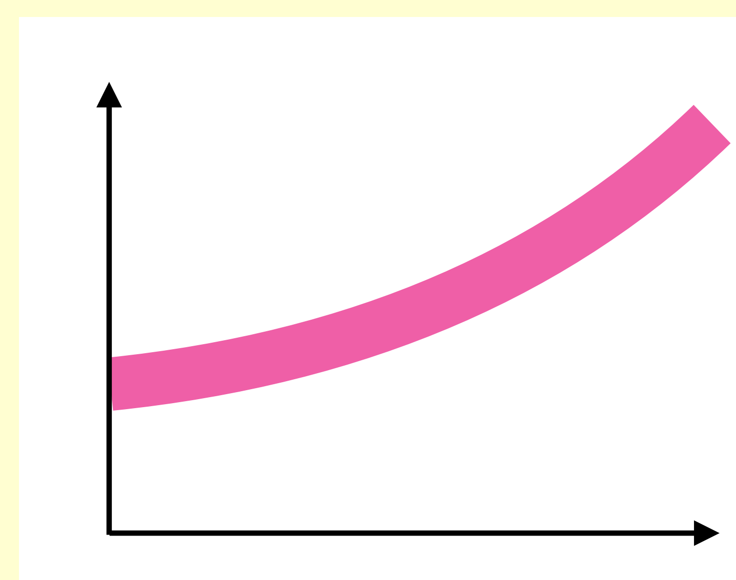
- … is a function where every element of the codomain is mapped to. Also called a **surjection**.



function: ✔

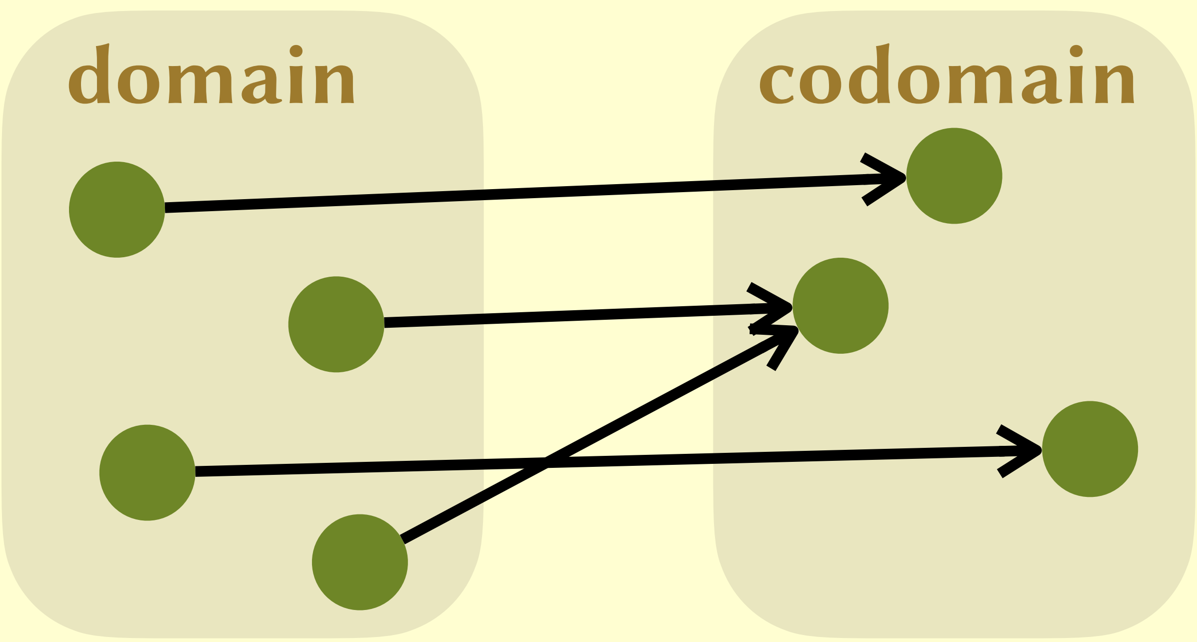
injective: ✘

surjective: ✘



A surjective function

- … is a function where every element of the codomain is mapped to. Also called a **surjection**.



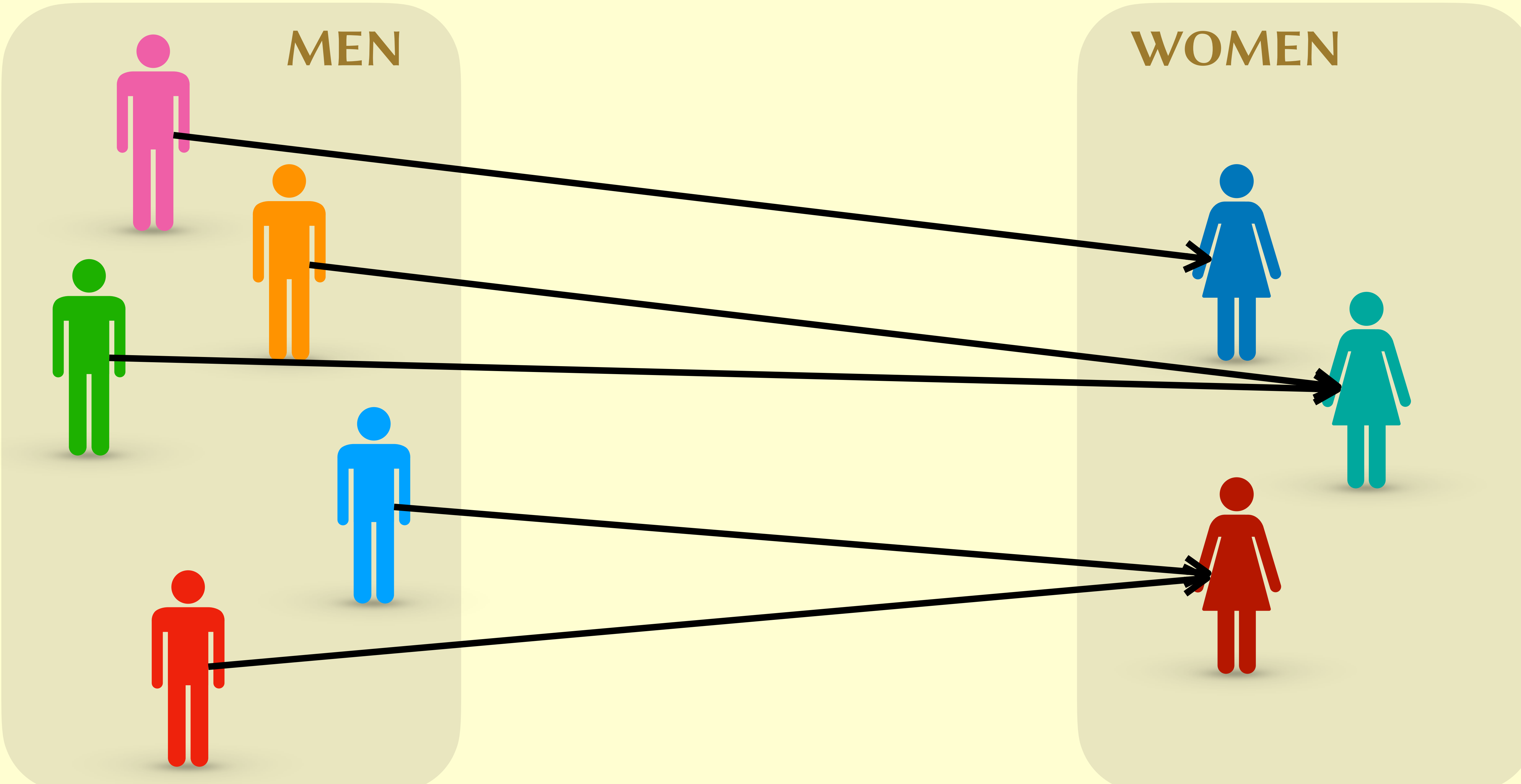
function: ✔

injective: ✘

surjective: ✔

- **Intuition.** If I can construct a surjection, then the domain is as least as large as the codomain.

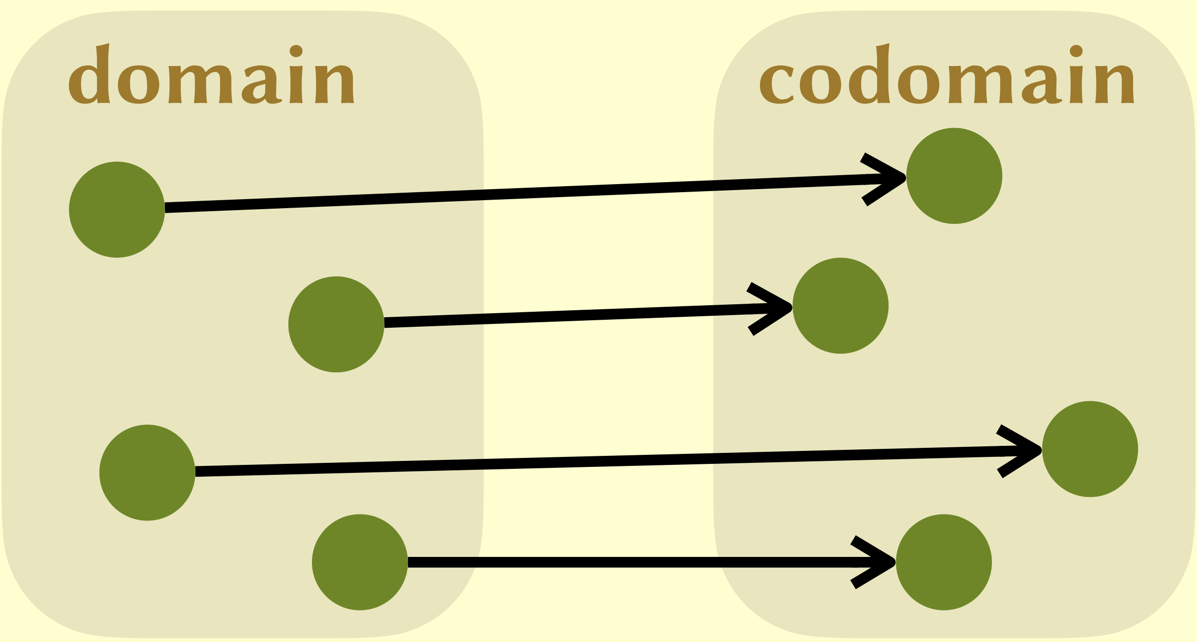
How to count



"There is a **surjection** *from the set of men to the set of women, so there are at least as many men as there are women."*

A bijective function

- … is a function that is both injective and surjective. Also called a **bijection**.



function: ✔

injective: ✔

surjective: ✔

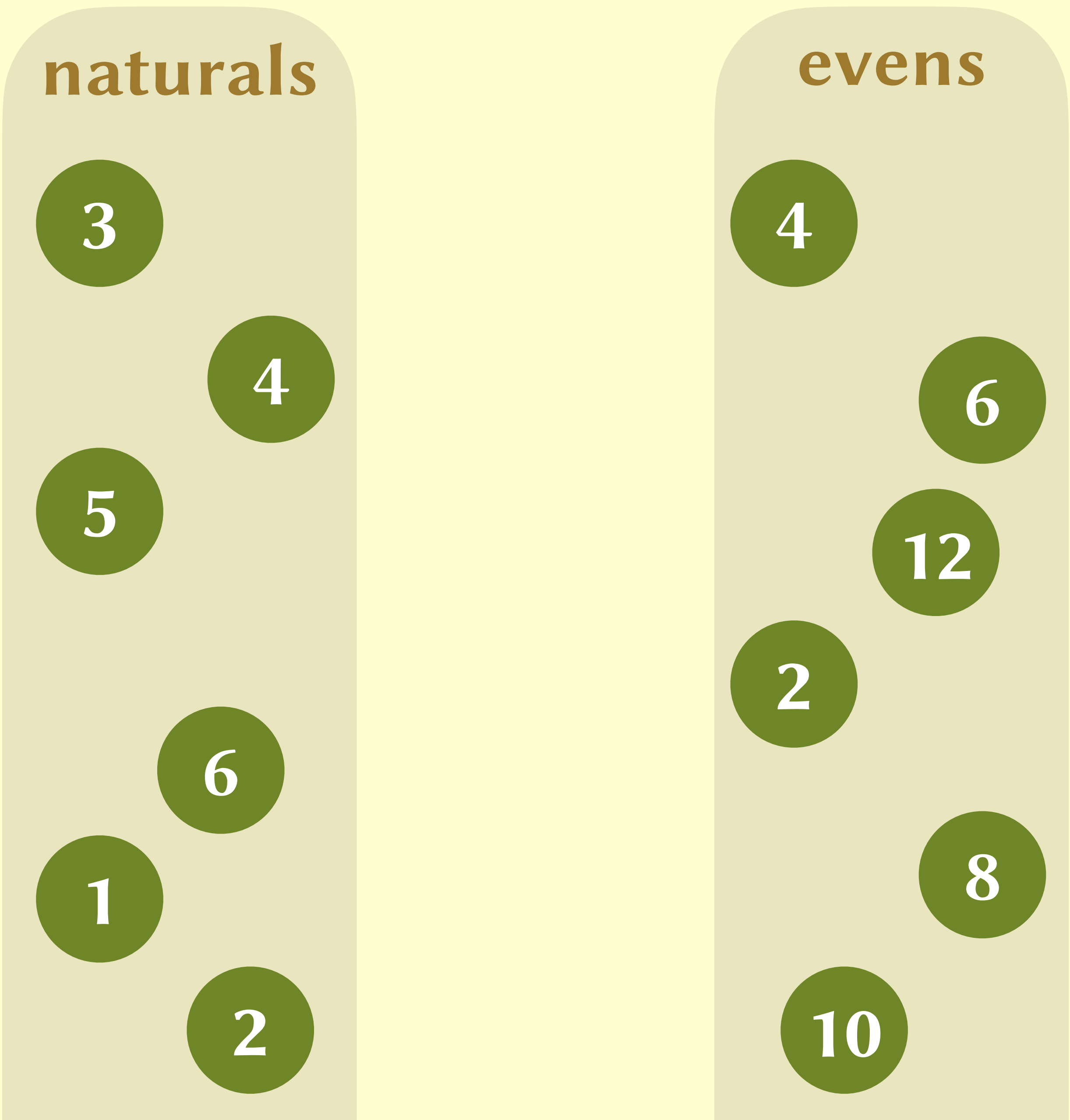
bijective: ✔

- **Intuition.** If I can construct a bijection, then the domain and the codomain are the same size.

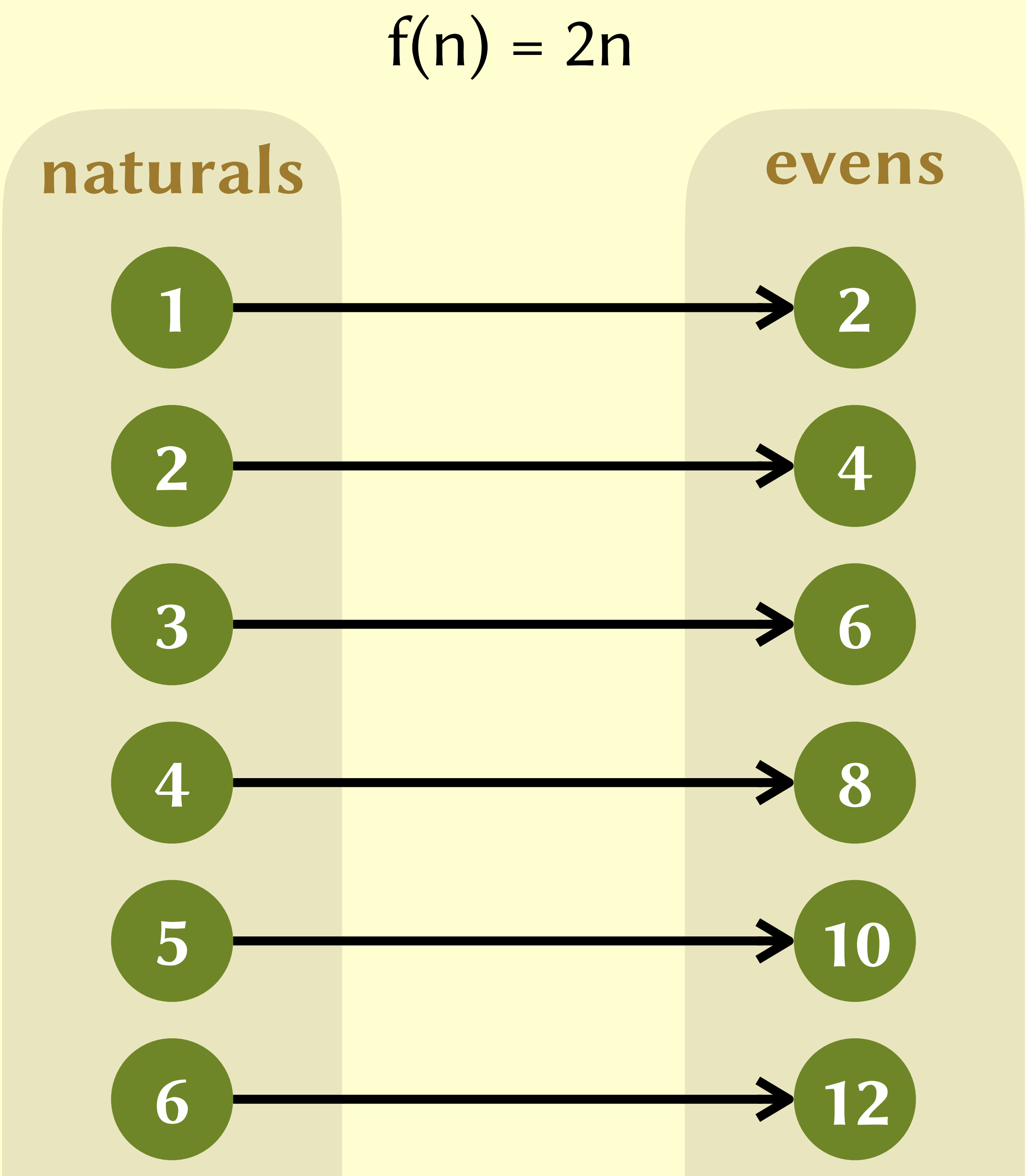
Why is this important?

- Conventional counting only works on **finite** sets.
- Injections and surjections work even with **infinite** sets.

Counting infinite sets



Counting infinite sets



- f is an **injection** because if m and n are different, then 2m and 2n will also be different.
- f is a **surjection** because every even number can be written in the form 2n for some n.
- So f is a **bijection**.
- So there are the same number of **naturals** and **evens**!

How many pairs?

naturals

1

2

3

4

5

6

pairs of naturals

1,1 1,5 4,3 2,4

2,5

3,3

5,1 6,1

1,3

2,2 5,2

3,4 1,4 3,2

4,2 3,1 1,6 4,1

2,3

1,2 2,1

How many pairs?

naturals

1

2

3

4

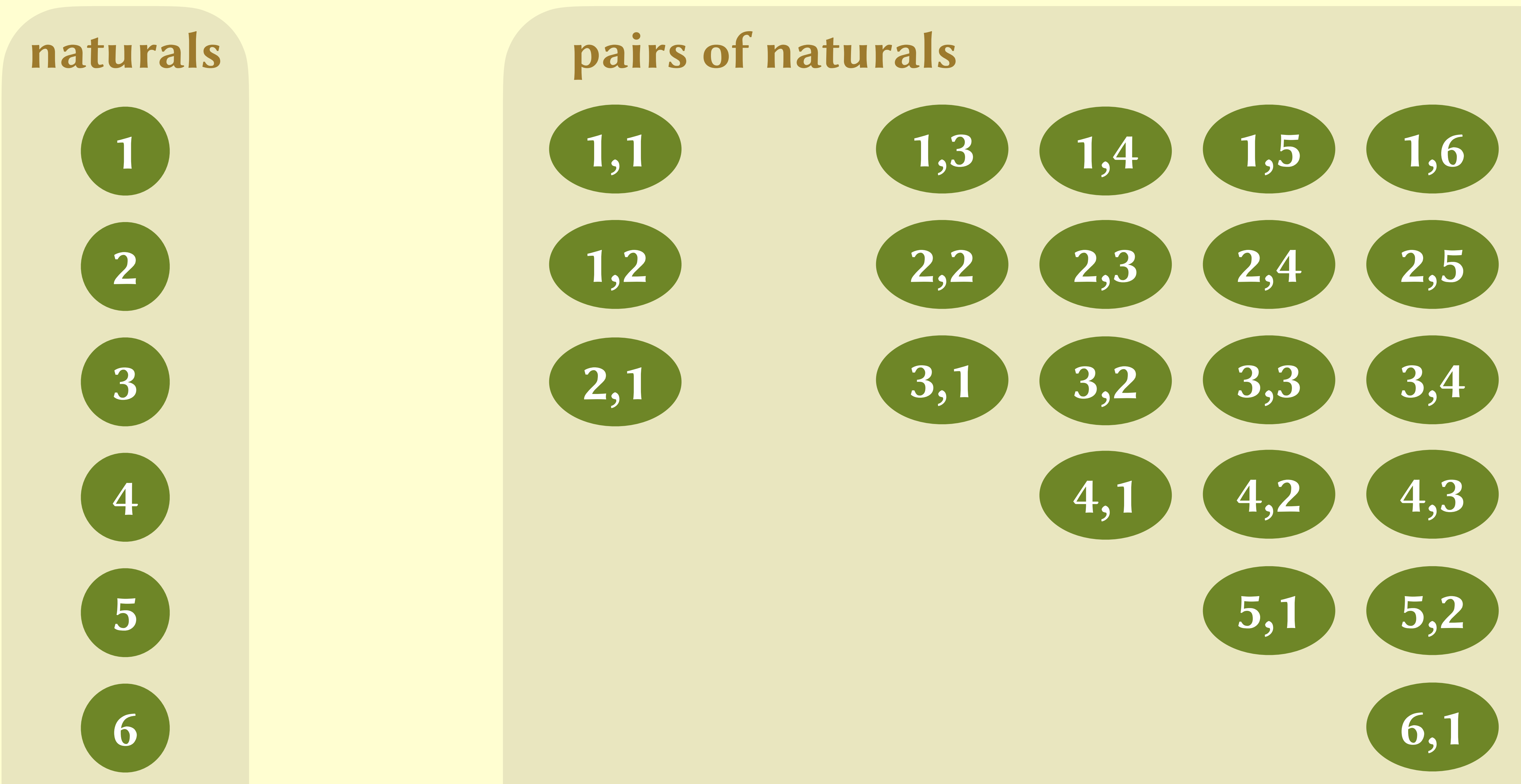
5

6

pairs of naturals

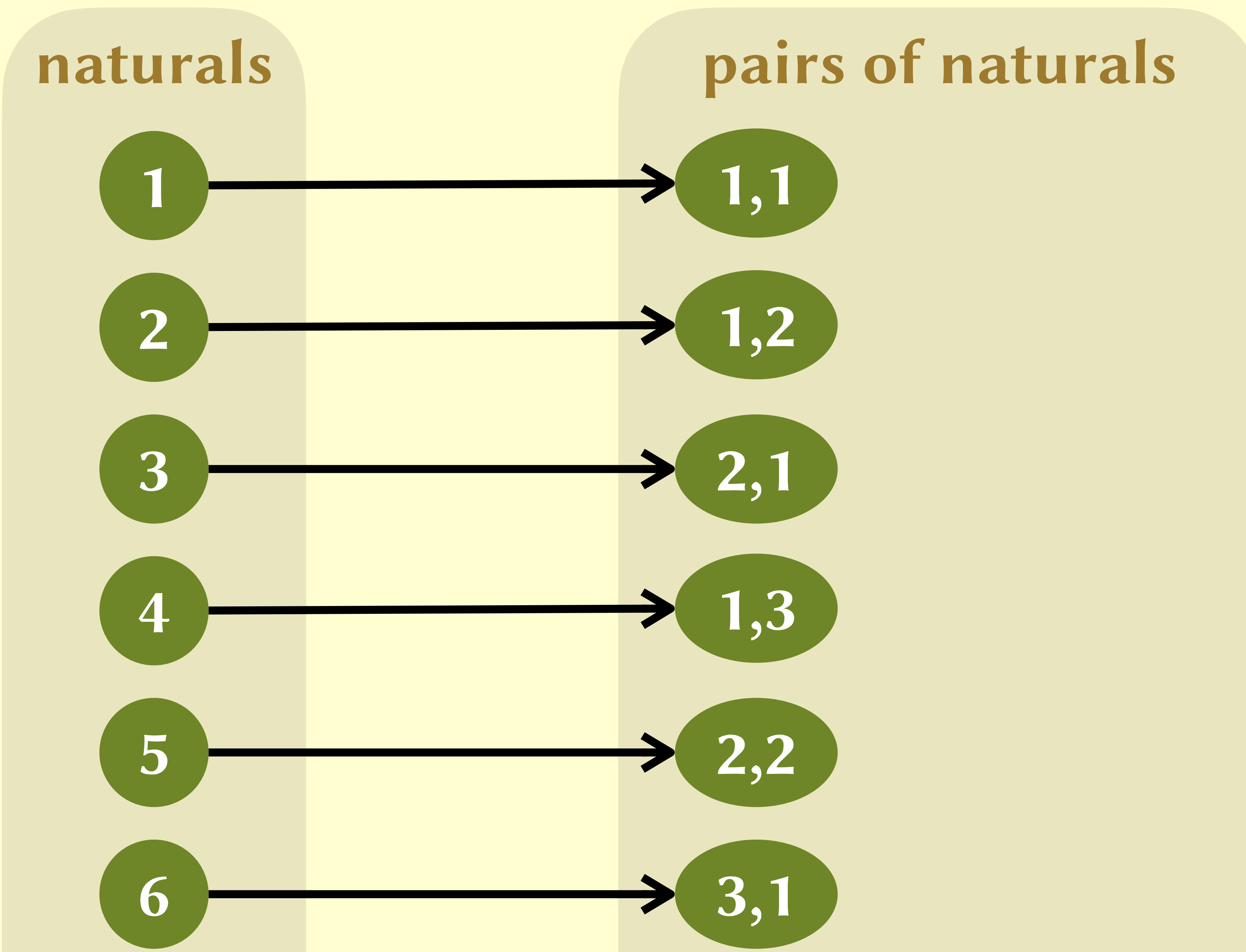
1,1	1,2	1,3	1,4	1,5	1,6
	2,1	2,2	2,3	2,4	2,5
		3,1	3,2	3,3	3,4
			4,1	4,2	4,3
				5,1	5,2
					6,1

How many pairs?



How many pairs?

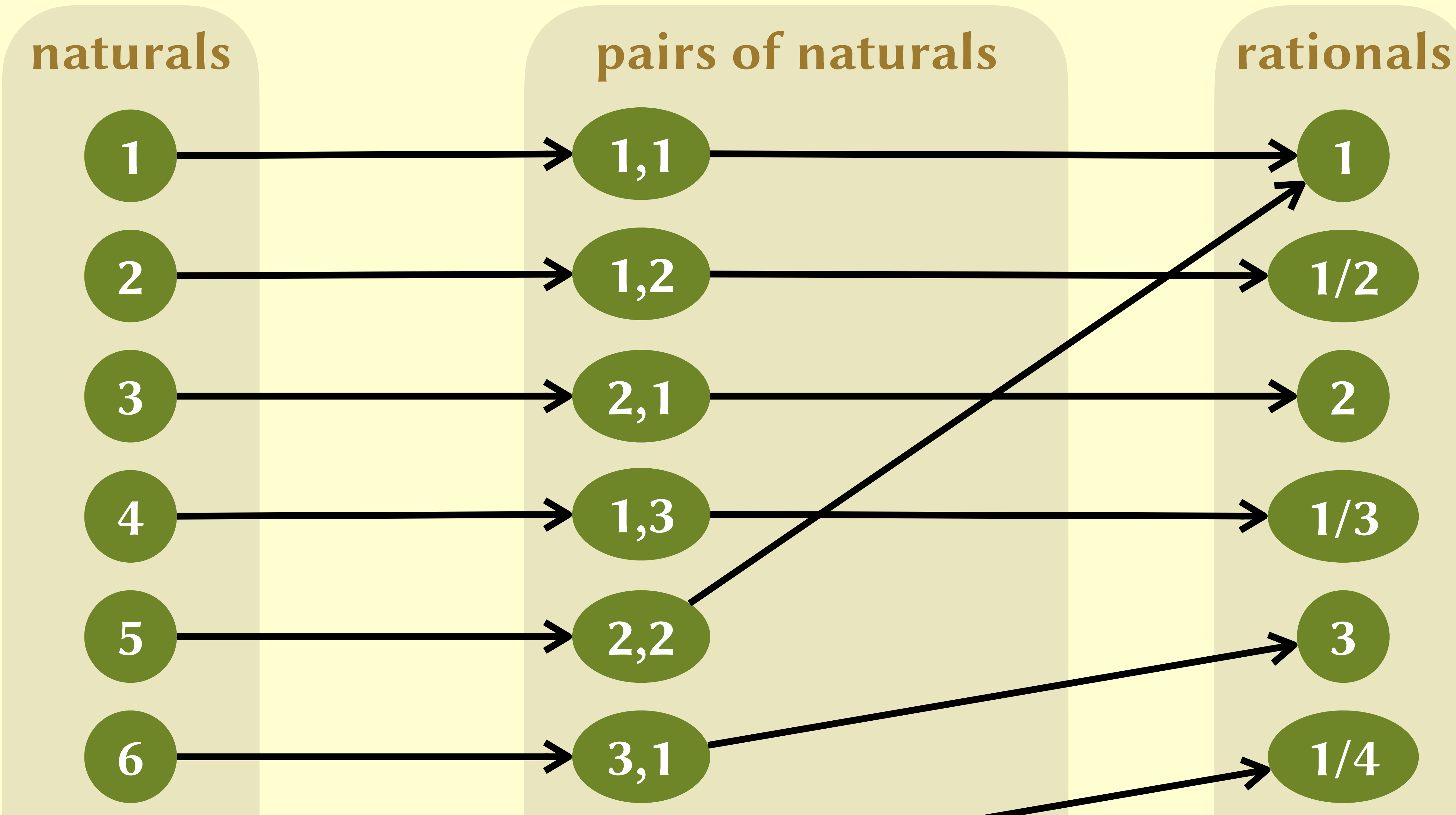
$$\mathsf{f}(n) = \left(n - \frac{d^2 + d}{2}, \frac{d^2 + 3d + 4}{2} - n \right) \quad \text{where } d = \left\lfloor \frac{\sqrt{1 + 7n} - 1}{2} \right\rfloor$$



- f is a **bijection**.
- So there are the same number of **naturals** and **pairs of naturals**!

How many pairs?

$$f(n) = \left(n - \frac{d^2 + d}{2}, \frac{d^2 + 3d + 4}{2} - n \right) \quad \text{where } d = \left\lfloor \frac{\sqrt{1 + 7n} - 1}{2} \right\rfloor$$



Summary so far

- These sets are all the same size:
  - The set of all natural numbers
  - The set of all even numbers
  - The set of all pairs of natural numbers
  - The set of all rational numbers
- We say these sets are **countably infinite** or just **countable**.

How many sets?

naturals

1

2

3

4

5

6

sets of naturals

$\emptyset$

1, 2, 3

6

2, 4, 7, 10

523, 721

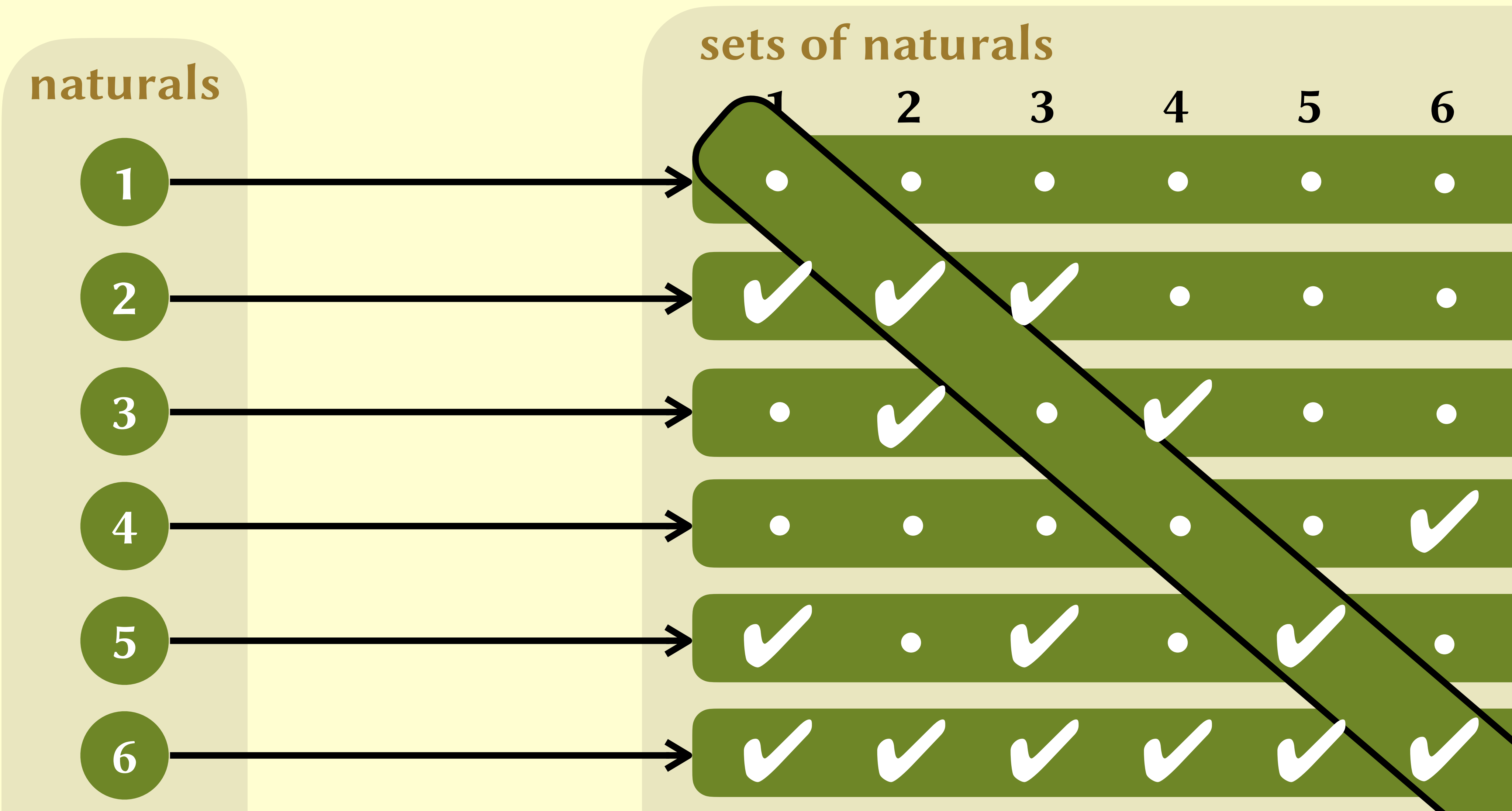
28, 30, 31, 6409, 411393422

all odd numbers

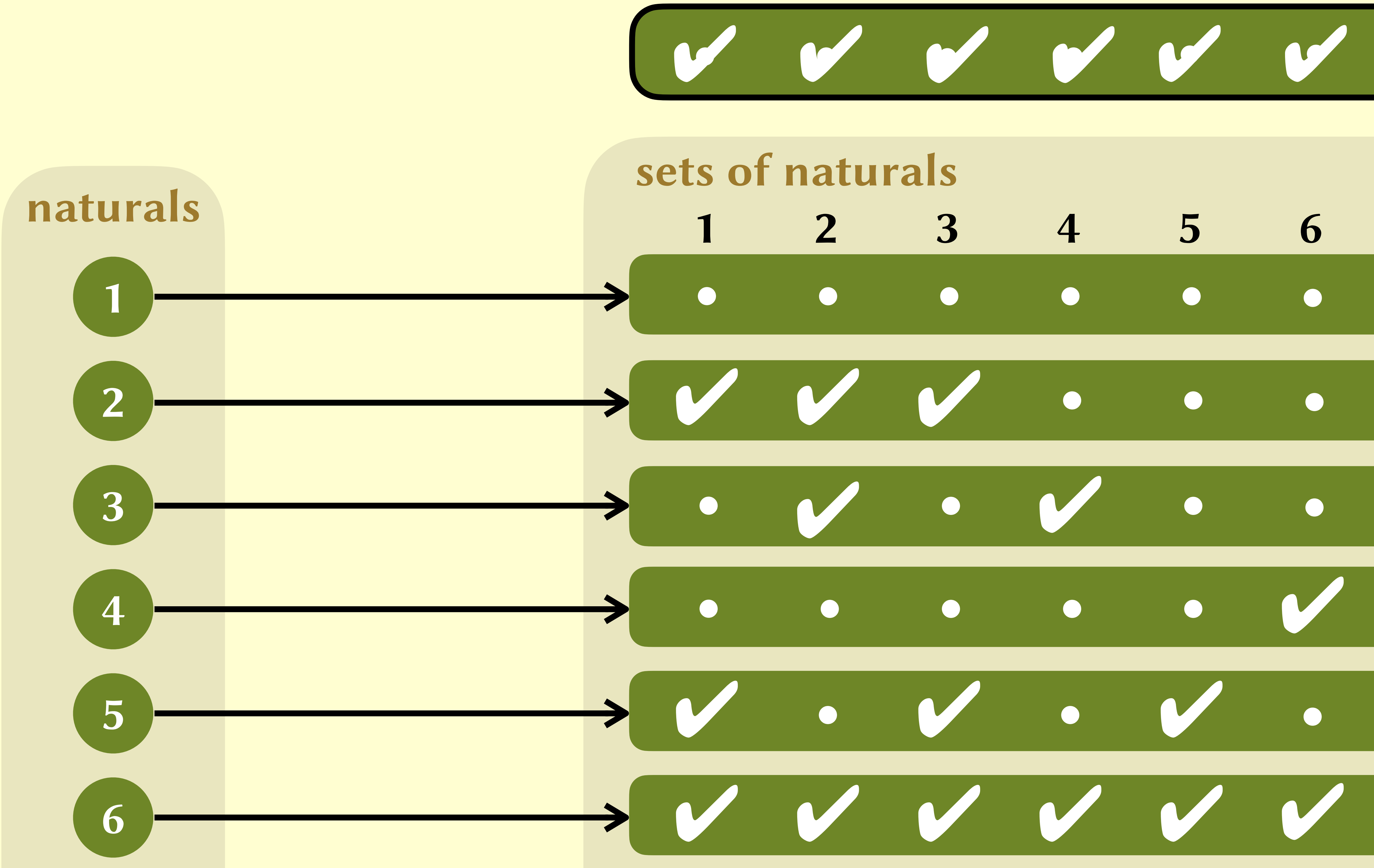
all primes

all naturals

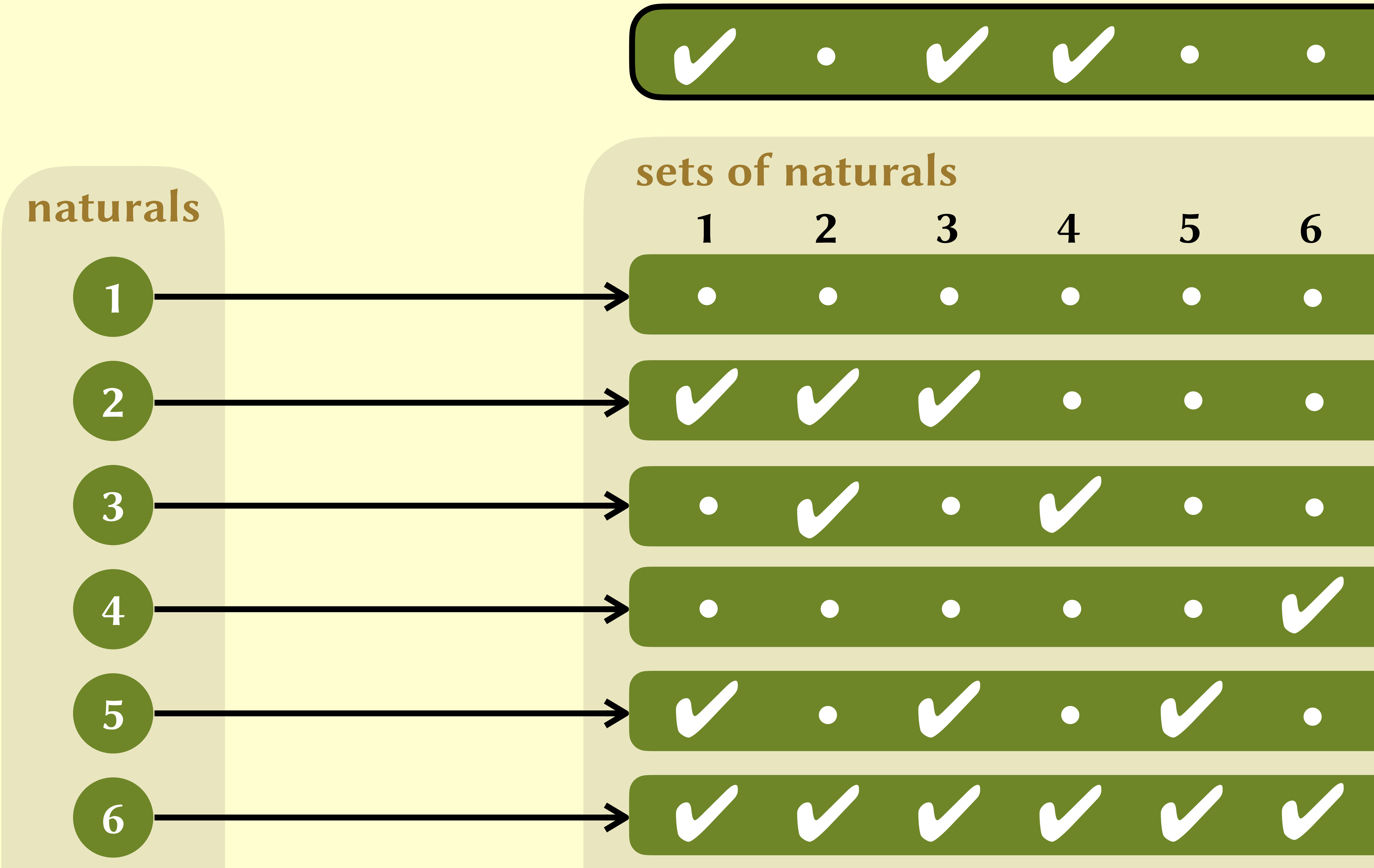
How many sets?



How many sets?



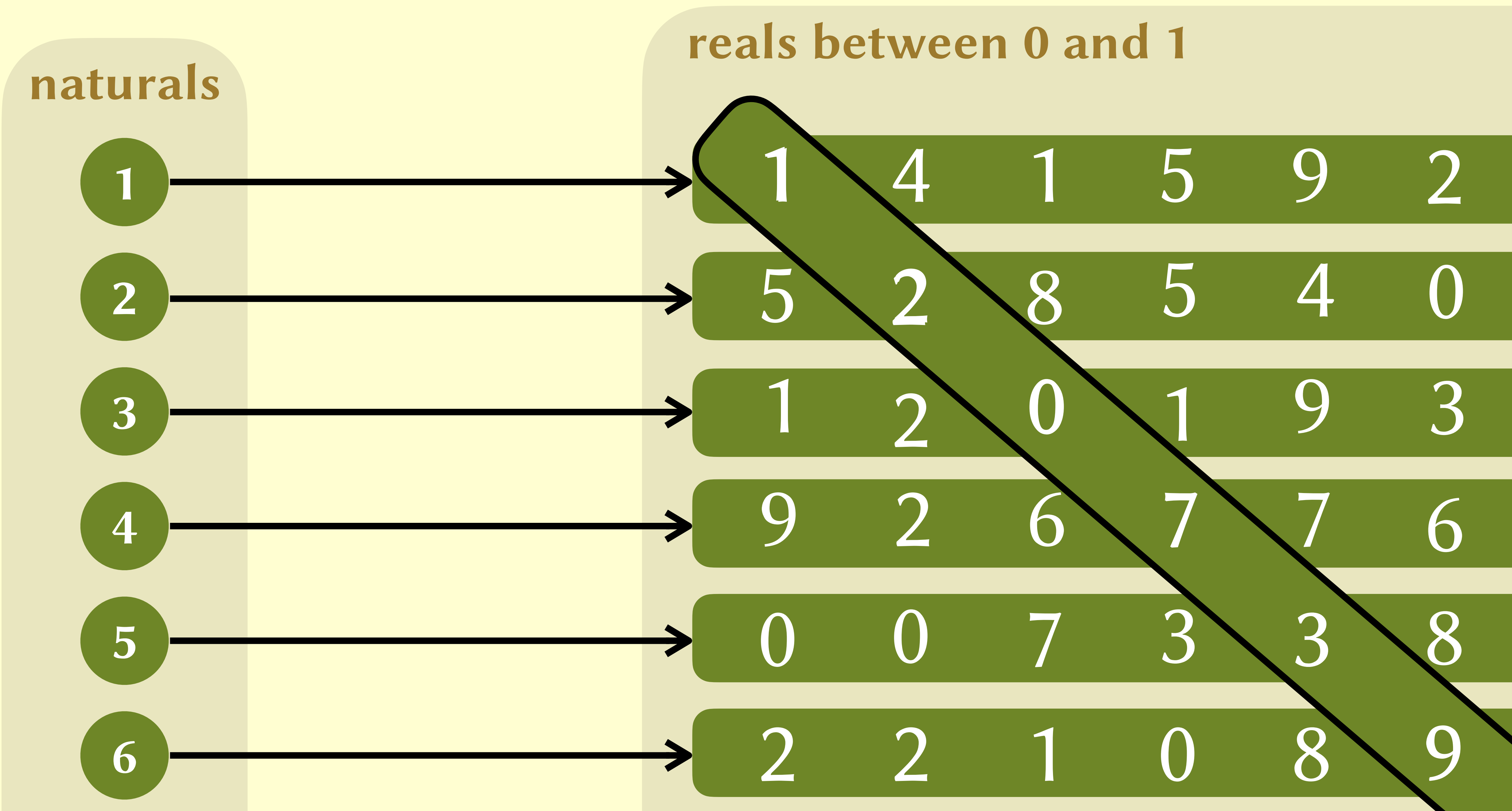
How many sets?



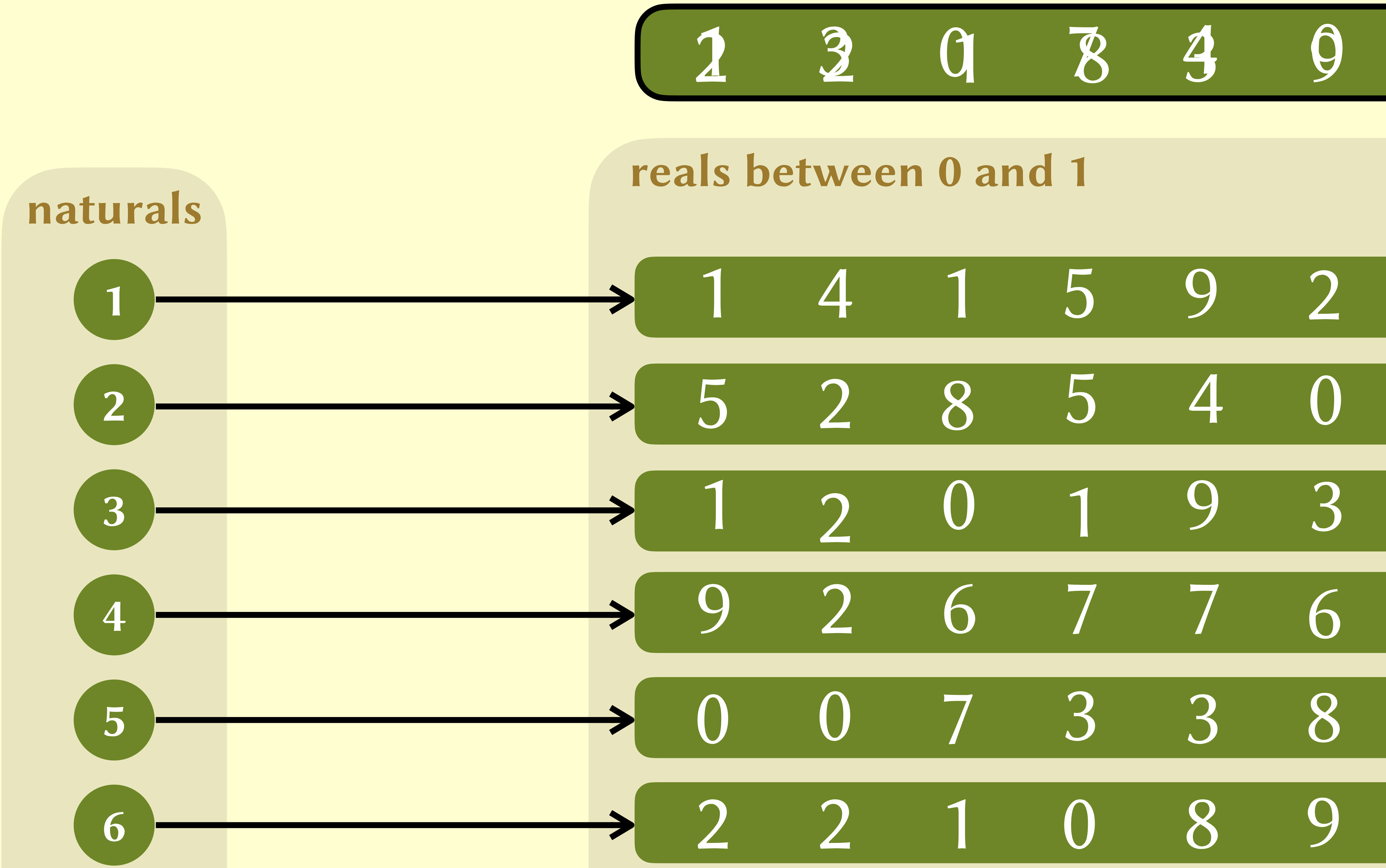
Uncountability

- There are more **sets of natural numbers** than there are natural numbers.
- We say that the set of all sets of natural numbers is **uncountably infinite** or **uncountable**.
- We can do a similar trick with the **real numbers**...

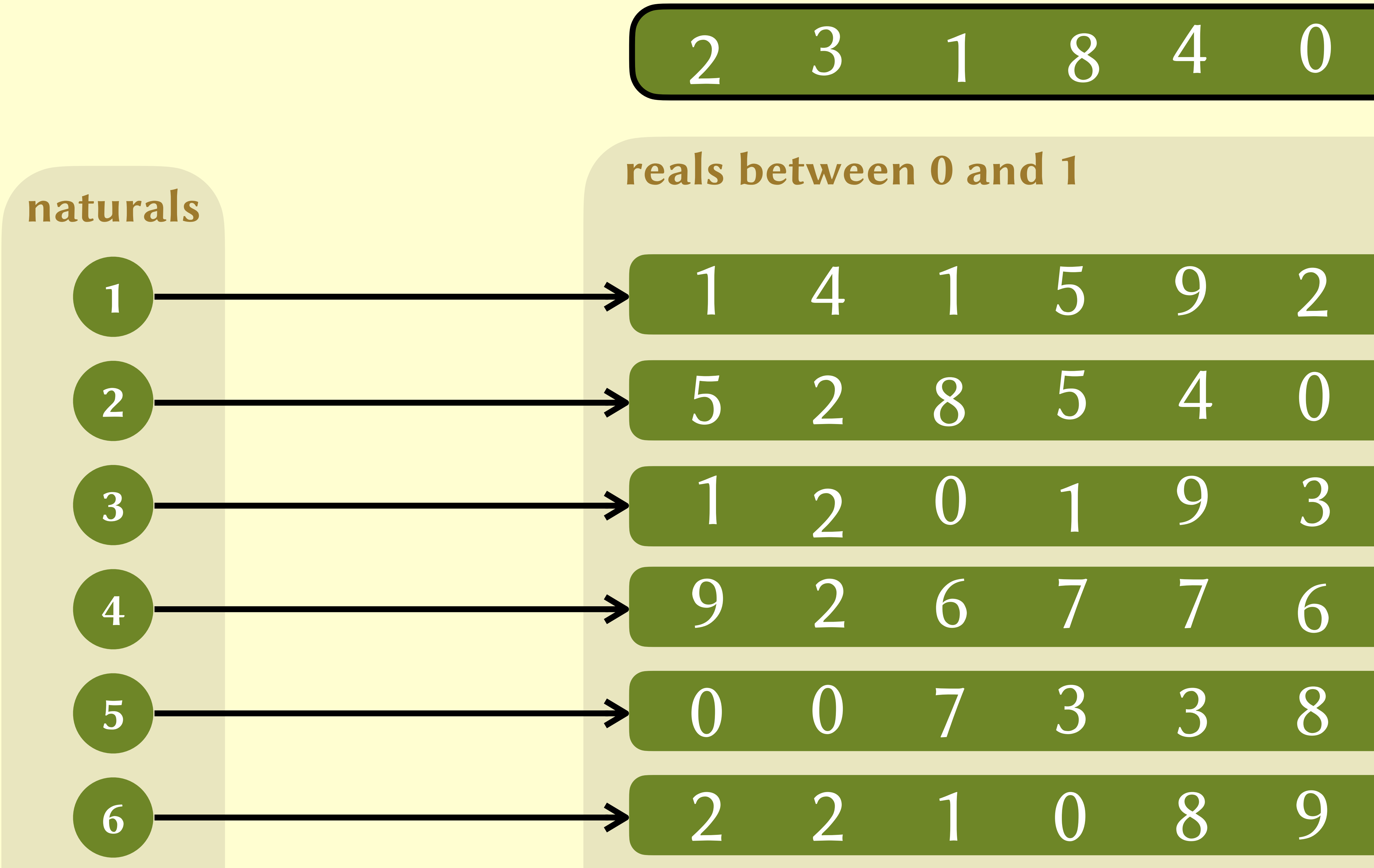
How many reals?



How many reals?

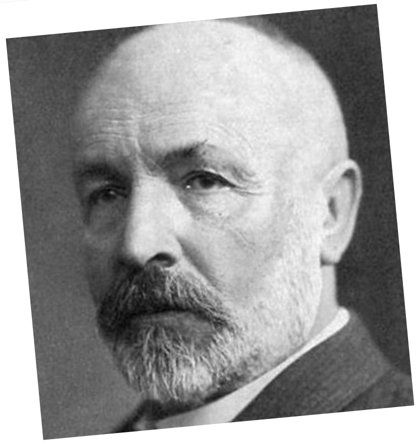


How many reals?



Uncountability

- There are more **sets of natural numbers** than there are natural numbers.
- We say that the set of all sets of natural numbers is **uncountably infinite** or **uncountable**.
- We can do a similar trick with the **real numbers**...



Georg Cantor
1845–1918

The Halting Problem

- **Task.** Write a program `halts` with the following declaration:

  ```
  int halts(char *P, char *D);
  ```

  If the program represented by the string `P` always terminates when run on the input string `D`, then `halts` should return 1. Otherwise it should return 0.

- **Examples:**

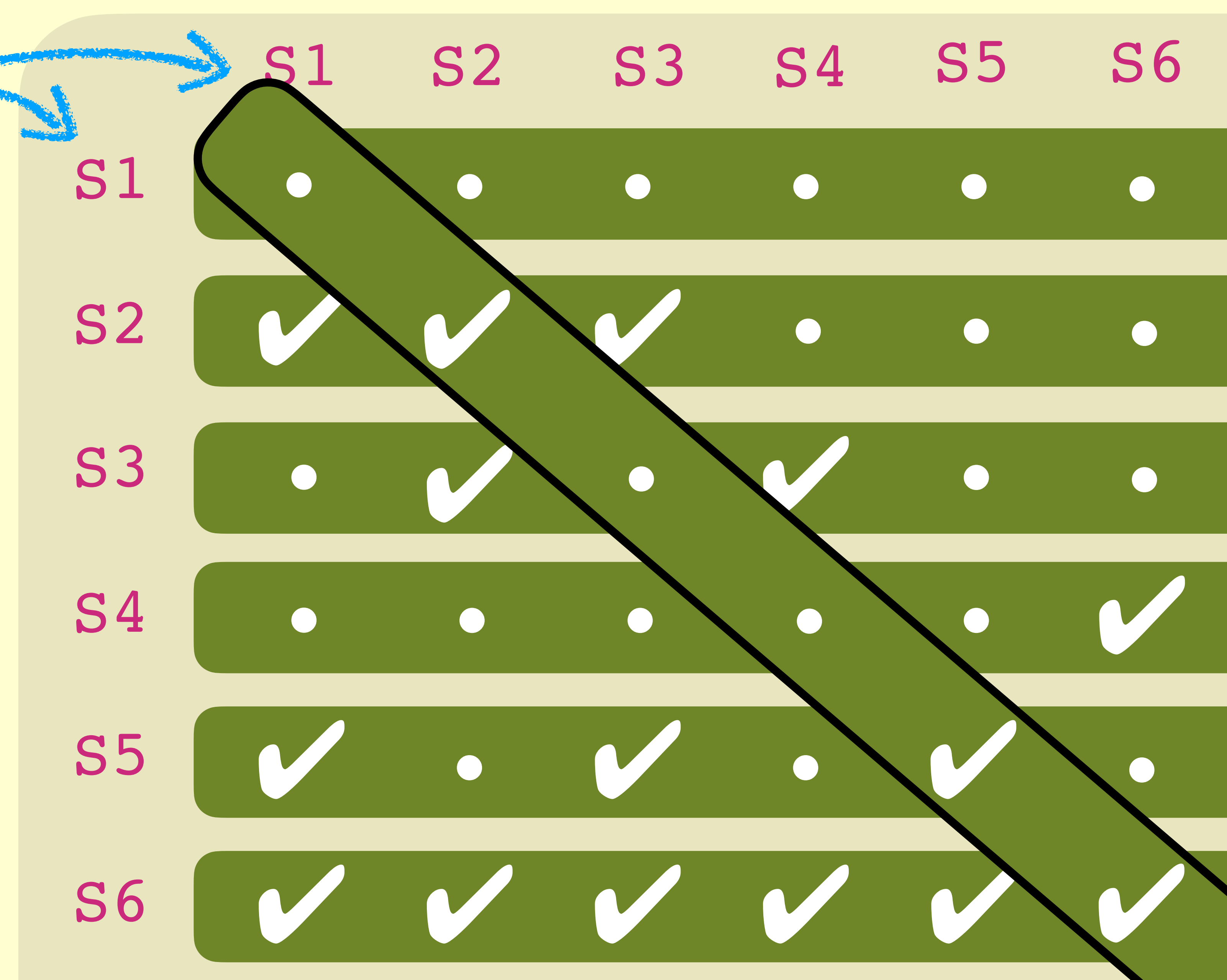
```
S1 = "int s1(char *D) {
        while(1);
     }"
```

`halts(S1, _)` = 0

```
S6 = "int s6(char *D) {
        return 42;
     }"
```

`halts(S6, _)` = 1

The Halting Problem



input strings

strings interpreted as source code

The Halting Problem

✔	✔	✔	✔	✔	✔

input strings

strings interpreted as source code

	S1	S2	S3	S4	S5	S6
S1	•	•	•	•	•	•
S2	✔	✔	✔	•	•	•
S3	•	✔	•	✔	•	•
S4	•	•	•	•	•	✔
S5	✔	•	✔	•	✔	•
S6	✔	✔	✔	✔	✔	✔

The Halting Problem

```
S =
"int s(char *D) {
   if (halts(D, D))
     while(1);
   else
     return 42;
 }"
```

S	✔	•	✔	✔	•	•

	S1	S2	S3	S4	S5	S6
S1	•	•	•	•	•	•
S2	✔	✔	✔	•	•	•
S3	•	✔	•	✔	•	•
S4	•	•	•	•	•	✔
S5	✔	•	✔	•	✔	•
S6	✔	✔	✔	✔	✔	✔

input strings

strings interpreted as source code

The Halting Problem

```
S =
"int s(char *D) {
   if (halts(D, D))
     while(1);
   else
     return 42;
 }"
```

Key question:
What happens if we run `s(S)`?

$$
\begin{tikzcd}
\texttt{s(S)}\text{ halts} \arrow[r, "\text{defn of } \texttt{s}"] & \texttt{halts(S,S)} = 0 \arrow[d, "\text{defn of } \texttt{halts}"] \\
\texttt{halts(S,S)} = 1 \arrow[u, "\text{defn of } \texttt{halts}"] & \texttt{s(S)}\text{ doesn't halt} \arrow[l, "\text{defn of } \texttt{s}"']
\end{tikzcd}
$$

The Halting Problem

- **Task.** Write a p… …owing declaration:

  `int halts(char` … `);`

  If the program repres… …ing `P` always terminates when run on the in… …`alts` should return 1. Otherwise it sho…

- **Examples:**

```
S1 = "int s1(char *D) {
        while(1);
     }"
```

`halts(S1, _)` = 0

```
S6 = "int s6(char *D) {
        return 42;
     }"
```

`halts(S6, _)` = 1

Why is this important?

- Is a live?

```
int main() {
  int a = 42;
  while(1);
  return a;
}
```

- Is a live?

```
int main(int n) {
  int a = 42;
  while(n > 1) {
    if (n % 2 == 0)
      n = n / 2;
    else
      n = 3 * n + 1;
  }
  return a;
}
```

Why is this important?

- **Key message.** There are some questions about programs (e.g. "will it halt?") that compilers cannot always answer.

- **NB.** This doesn't mean that compilers can *never* tell if a program will halt. Rather, they *cannot always* tell if a program will halt.

- For instance, the TERMINATOR tool can "usually" tell whether a program will halt.

Summary

- We can compare the sizes of two sets by constructing **injections**, **surjections**, or **bijections** between them.
- The sets of natural numbers, even numbers, pairs of natural numbers, and rational numbers are all "**countably infinite**".
- But the set of all sets of natural numbers is "**uncountably infinite**". (Proof by **diagonalisation**.)
- The **halting problem** is "**undecidable**". (Proof also by diagonalisation.) So there are some questions that static analysis cannot ^ answer.

  always