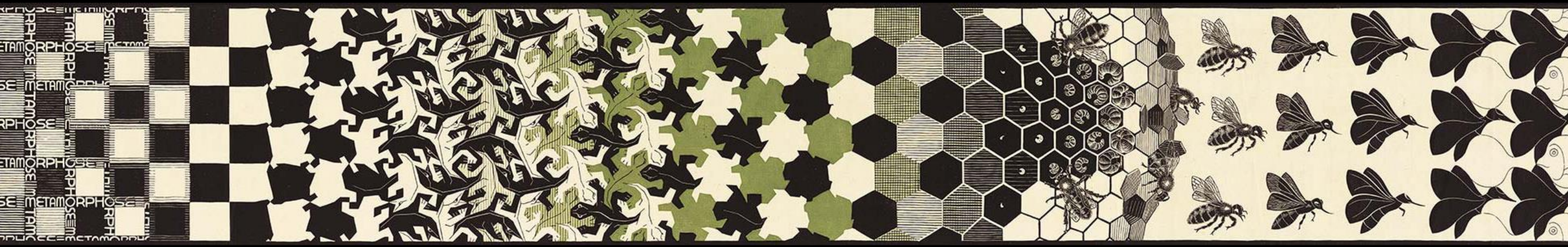
Data, Math and Methods

Week 4, State Machines & Primes



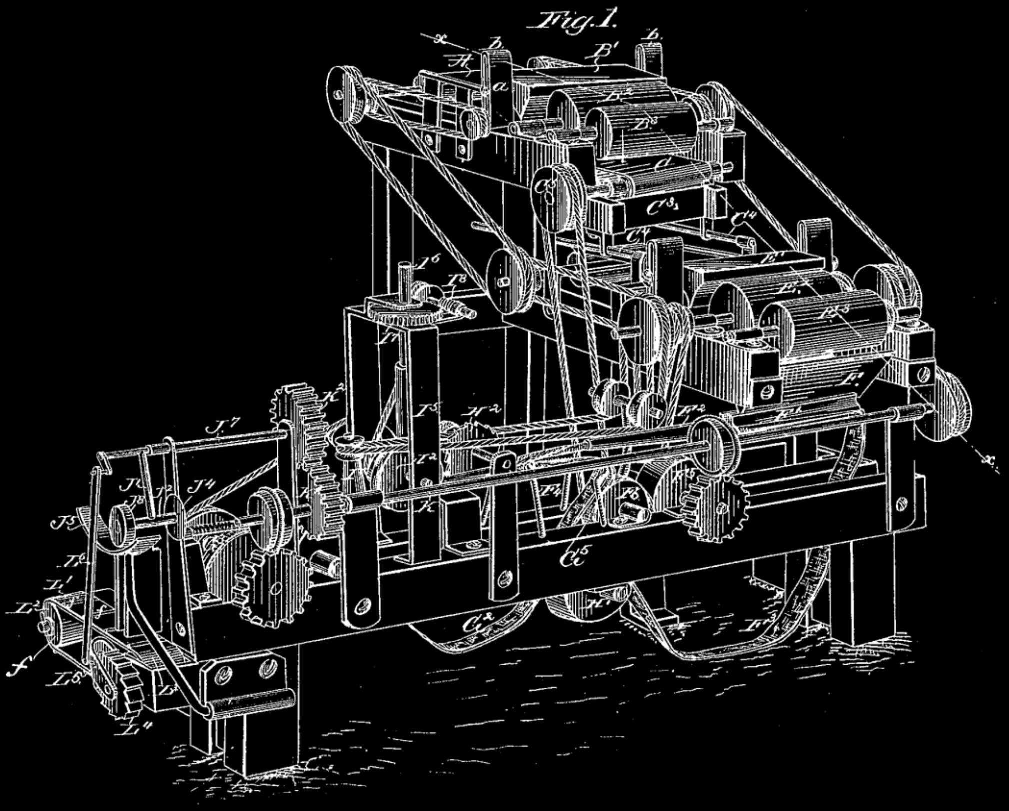
~ Vítek Růžička

Today

- State Machines
  - What they are?
  - Drawing up small state machines on paper
  - Coding part
- Prime numbers
  - Patterns
  - Coding an algorithm to get prime numbers

Machines

- What are state machines? What is a machine?

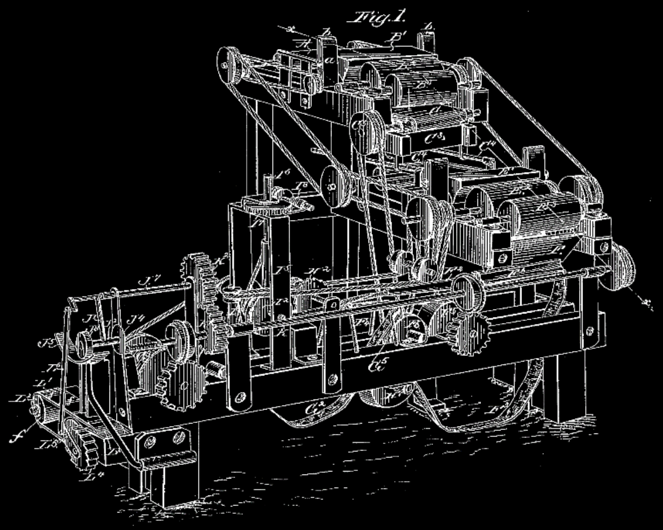


Machines

- What are state machines? What is a machine?

<u>Inputs</u> >

*In some form that the machine will understand = we call that an **<u>alphabet</u>***



\> **<u>Outputs</u>**

Again some which we are expecting

Machine has a **<u>state</u>** (turned on / off / waiting for input / etc ...)

Machines

- We will soon see a more exact example …

Inputs >

- *ID card*
- *person going in*

= ***alphabet***

> **Outputs**

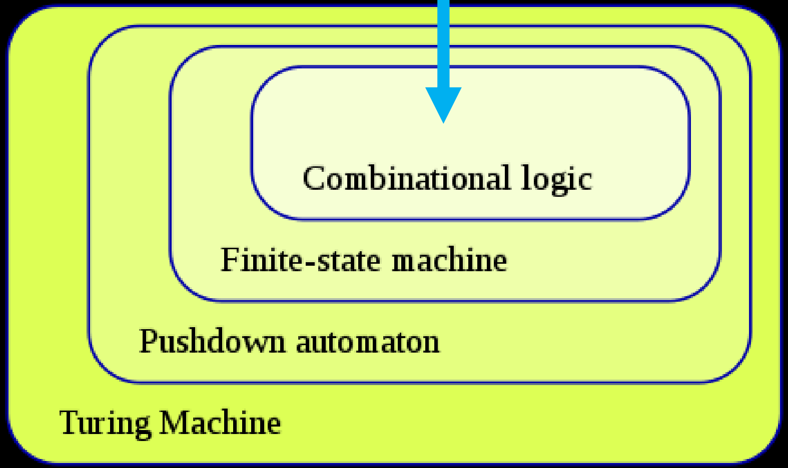
- *Let through*
- *Don't*

State

Ready -> Waiting for ID check -> Let through / Off (other side)

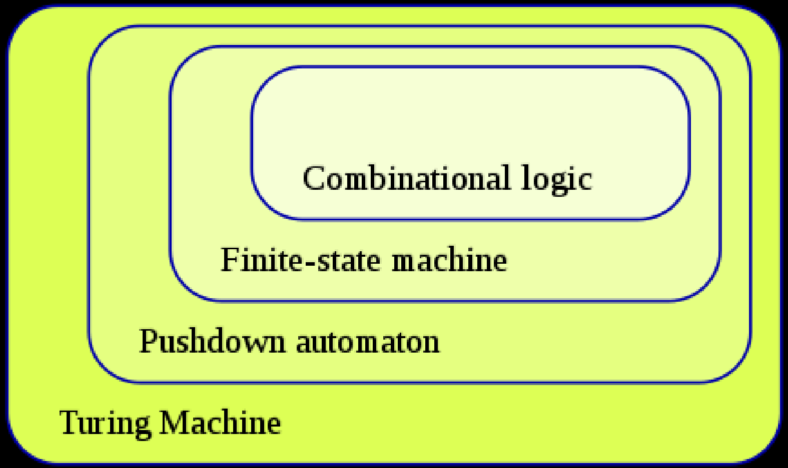
State Machines

$$A \cdot \neg B \cdot \neg C$$



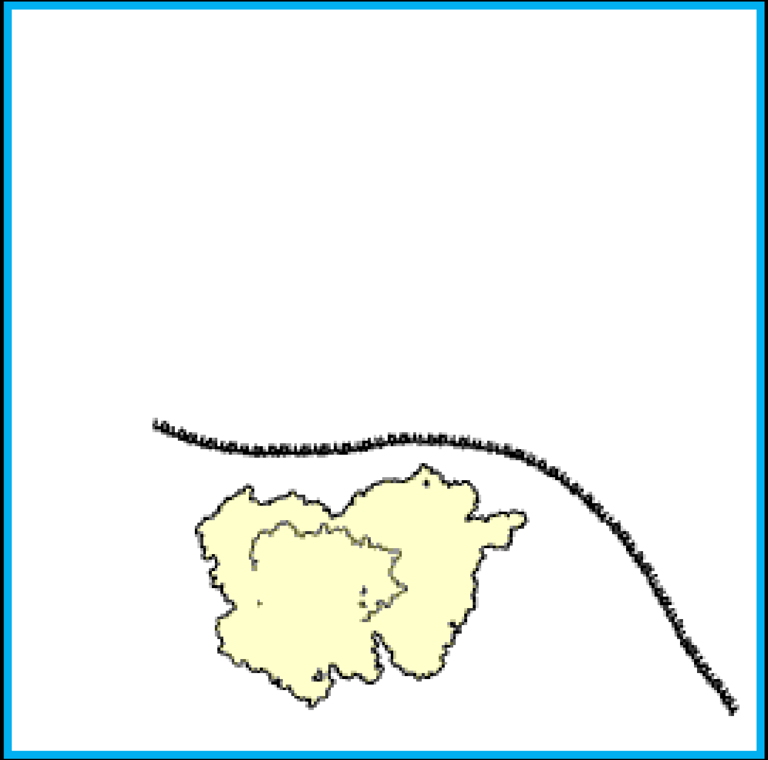
State Machines

- **Finite State Machines** are simple models of machines
  - They can be used to prove things (like if a certain state can be reached at all)
- There is a hierarchy of models …
- … what are these missing?



Machines in nature

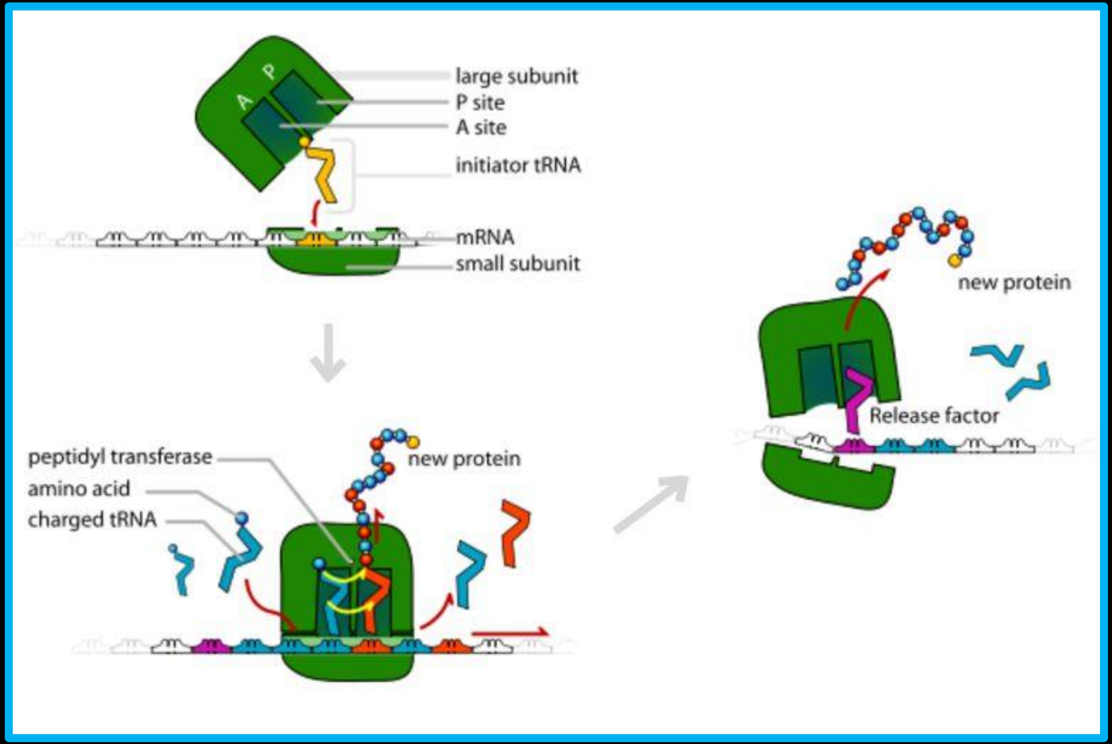
- Anyone recognizes this?



(if the animation is broken)

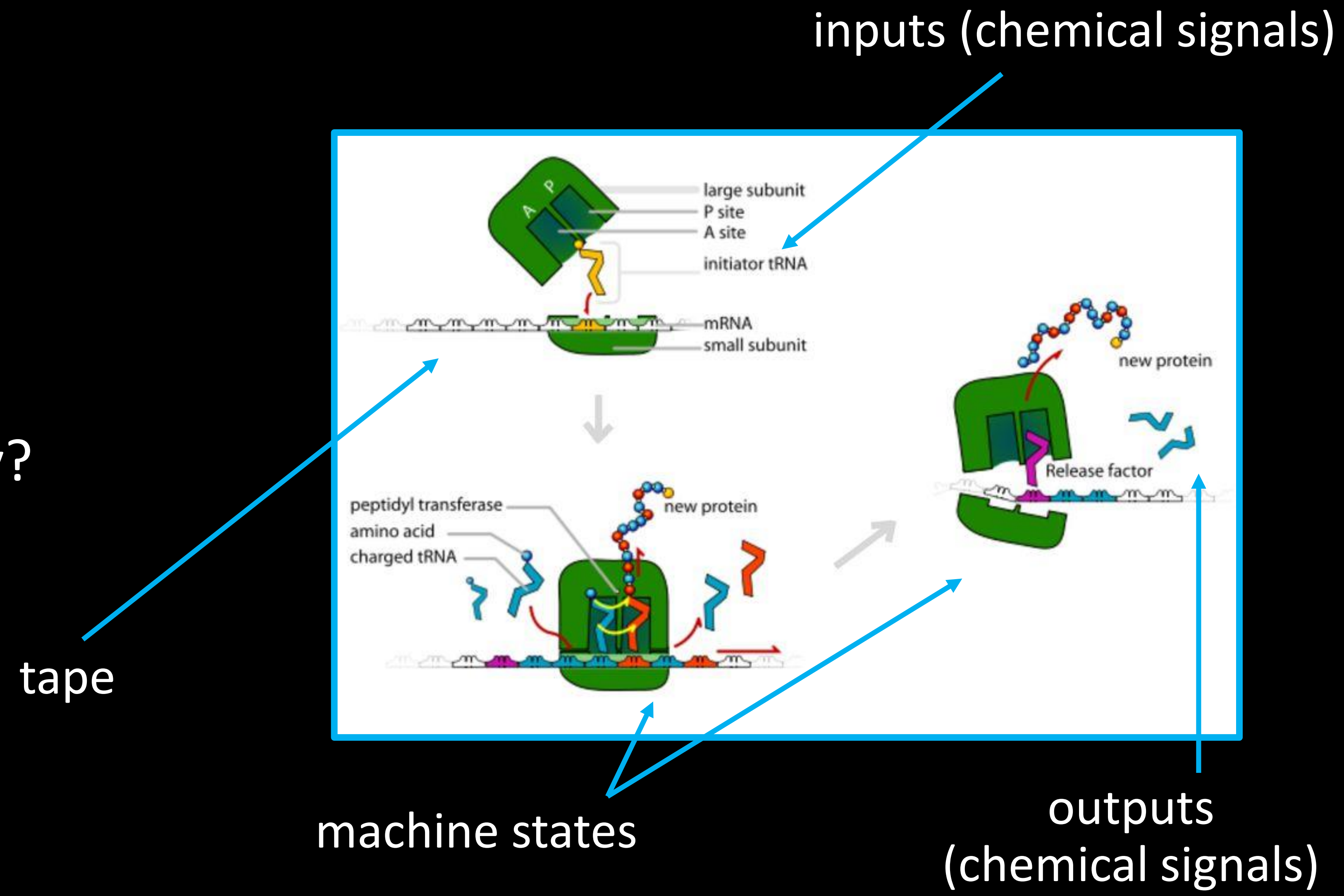
Machines in nature

- Inspiration by biology?



Machines in nature

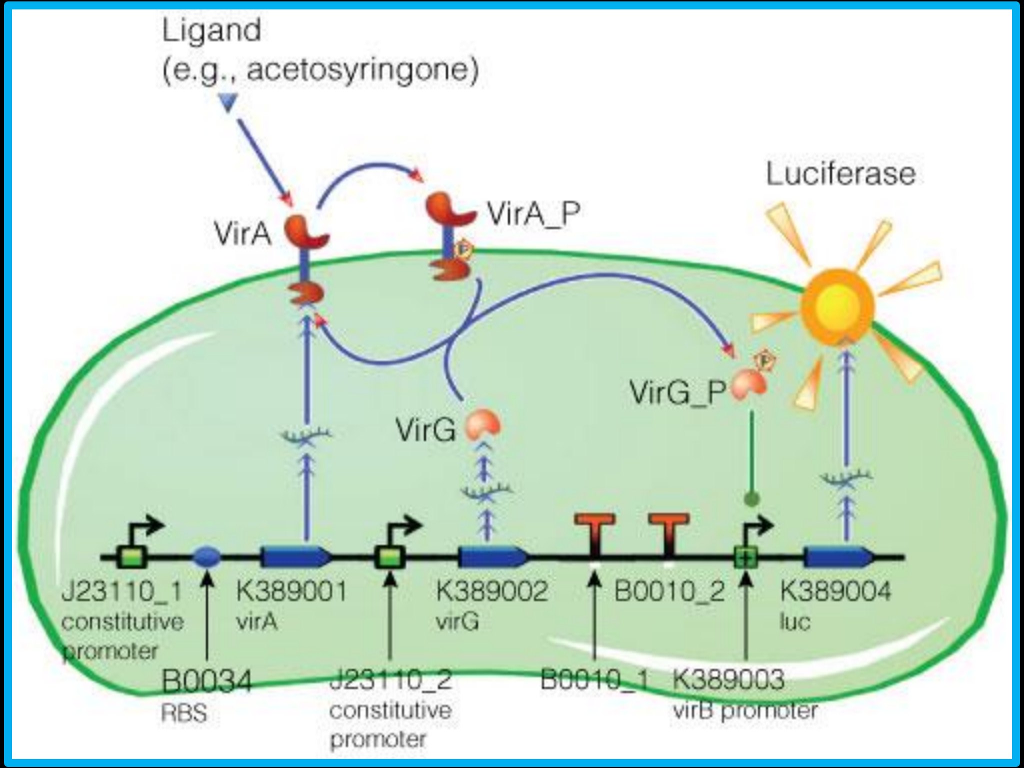
- Inspiration by biology?



Machines and nature

- Paper where they are building an organism (Escherichia Coli) as a machine with desired properties … of a sensor.

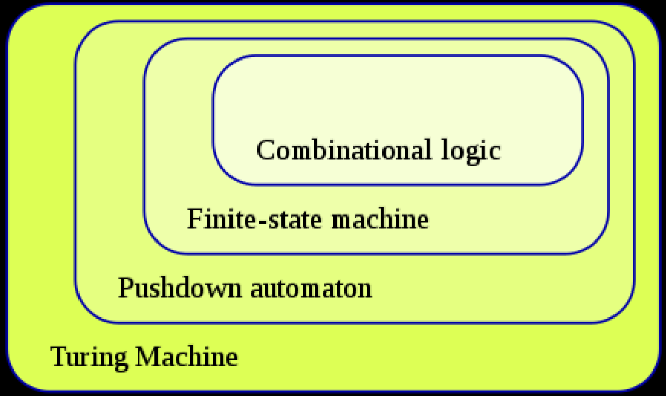
- *“In this project we established an E.coli-based biosensor that is capable of sensing and measuring acetosyringone induction.”*



[Establishing a Luminescent Biosensor in E. coli: The Modulated Acetosyringone Receptor Sensing System]()

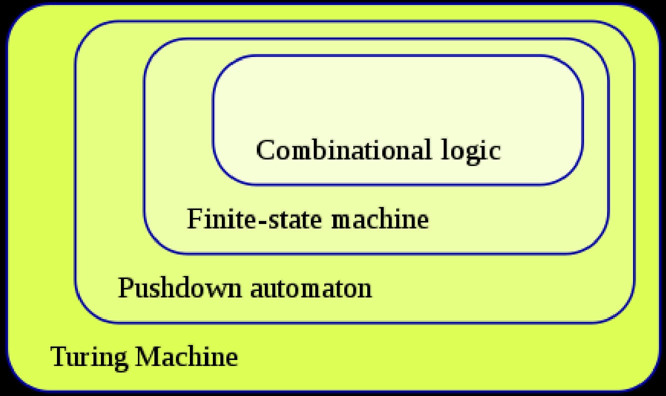
Machines

- **Turing Machines**
  - As an addition have an *infinitely long* **tape** with memory
  - They can read and write into this **tape**
  - Can be used to write any program



Machines

- **Turing Machines**
  - As an addition have an *infinitely long* **tape** with memory
  - They can read and write into this **tape**
  - Can be used to write any program

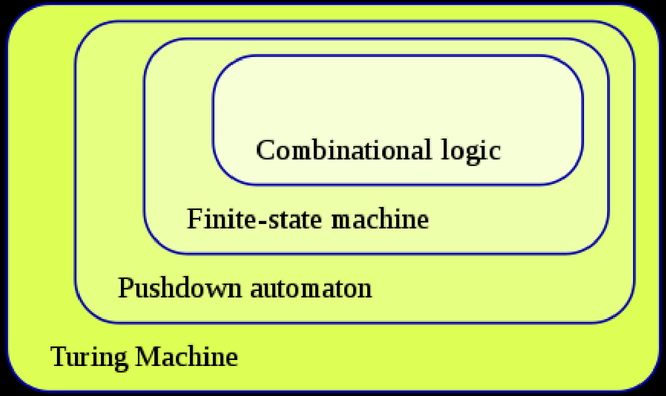


- Some ideas of computing machines existed long time ago … (*abacus, mechanical calculators, …*)

- 1936 model by Alan Turing (+ idea of “calculating machine” 1834 by Charles Babage)

Machines

- **Turing Machines**
  - As an addition have an *infinitely long* **tape** with memory
  - They can read and write into this **tape**
  - Can be used to write any program



- Some ideas of computing machines existed long time ago … (*abacus, mechanical calculators, …*)

- 1936 model by Alan Turing (+ idea of "calculating machine" 1834 by Charles Babage)

*** Birth of Computer Sciences**

Machines

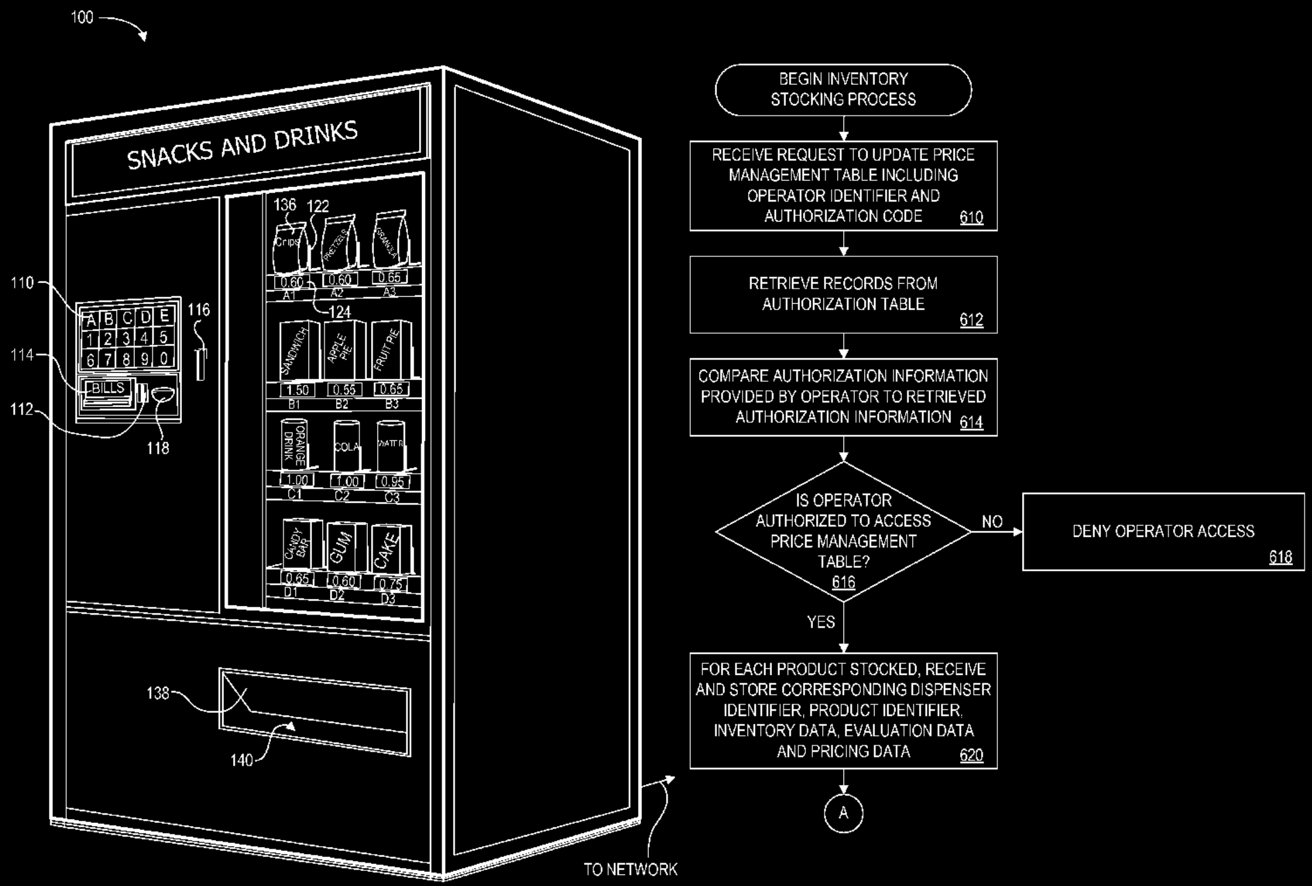
- **Turing Machines**
  - *Turing completeness is the ability for a system of instructions to simulate a Turing machine. A programming language that is Turing complete is* **theoretically capable of expressing all tasks accomplishable by computers**; *nearly all programming languages are Turing complete if the limitations of finite memory are ignored.*

State Machines

- What can we use them for?

State Machines

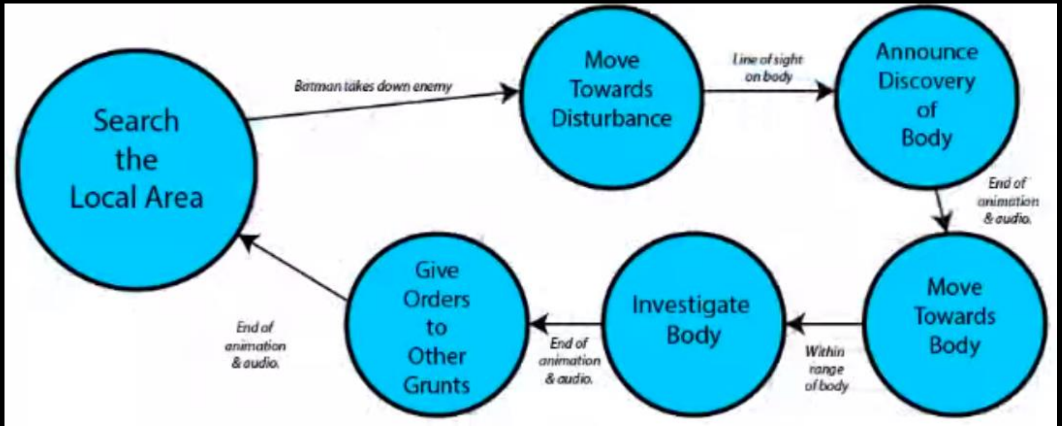
- What can we use them for?
  - Simple machines control logic:



State Machines

- What can we use them for?
  - AI design in games (and other):

    directly as the model to use for NPCs:

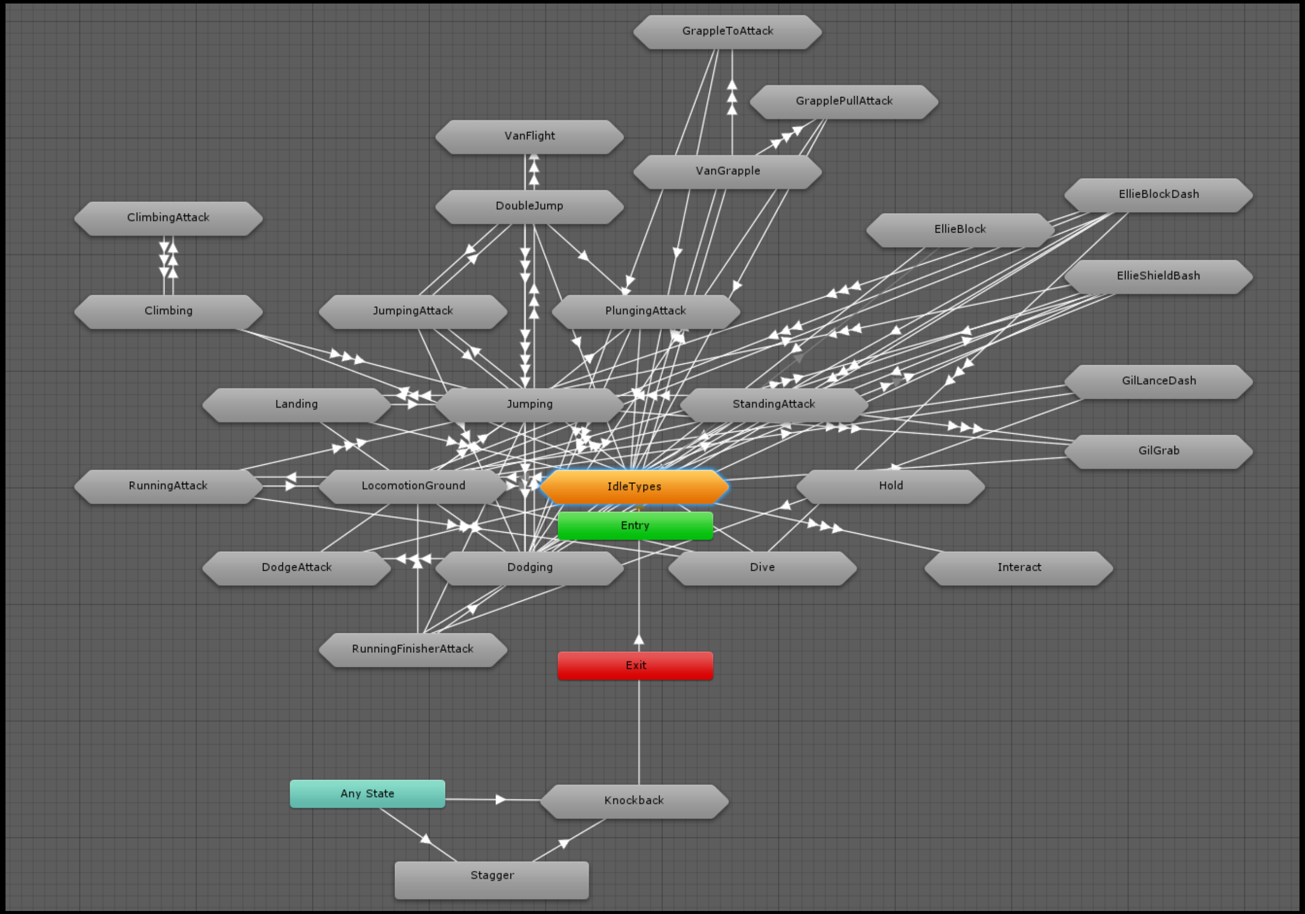


  - AI in Half-Life 1 (uses state machines, then was a huge advancement): https://www.youtube.com/watch?v=JyF0oyarz4U
  - AI in Arkham Assilum (still state machines, more advanced) https://www.youtube.com/watch?v=Oz04rH542I8&t=579s

State Machines

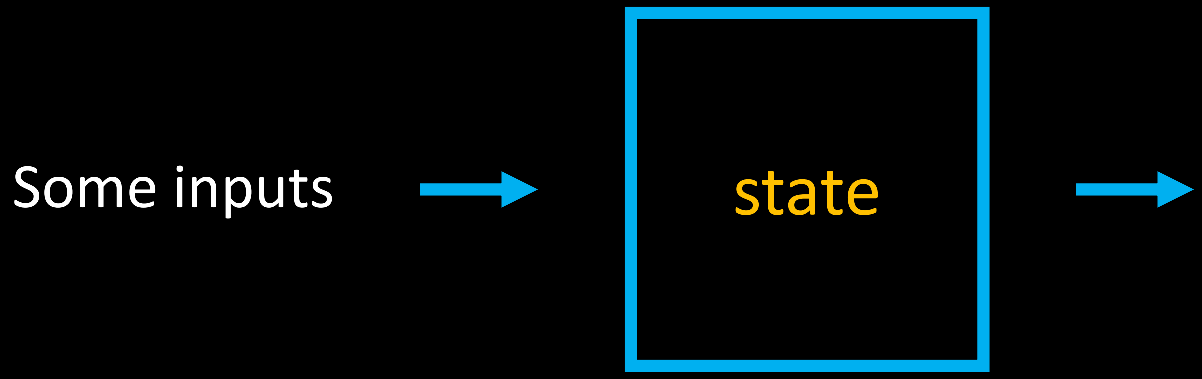
- What can we use them for?
  - AI design in games (and other):

    not directly, as a useful GUI tool to control animations:



State Machines

- What can we use them for?
  - In our case???



- Depending on the state choose:
  - Output color
  - Output image
  - Output looping sample
  - .....

BTW:

Wekinator

Some inputs → ??? → Some output

Look up:
http://www.wekinator.org/example-projects/

State Machines

- Simple example – let’s build a state machine

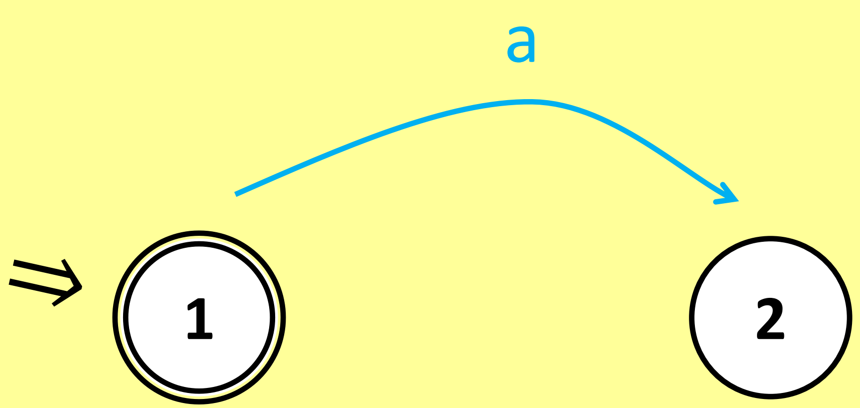
State Machines example

- State machine models some behavior …
- It has some internal states: **1, 2**

State Machines example

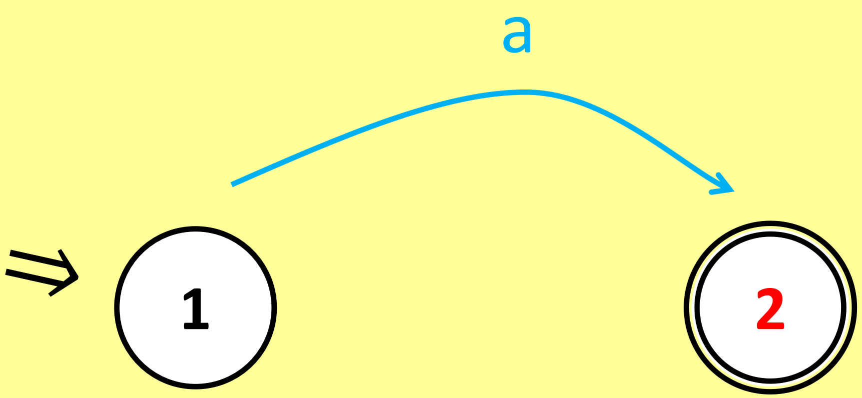
- The machine **starts** in one of these states (marked with the arrow)
- We can consider the machine being in one of the states (marked by the double circle on **1**)

State Machines example



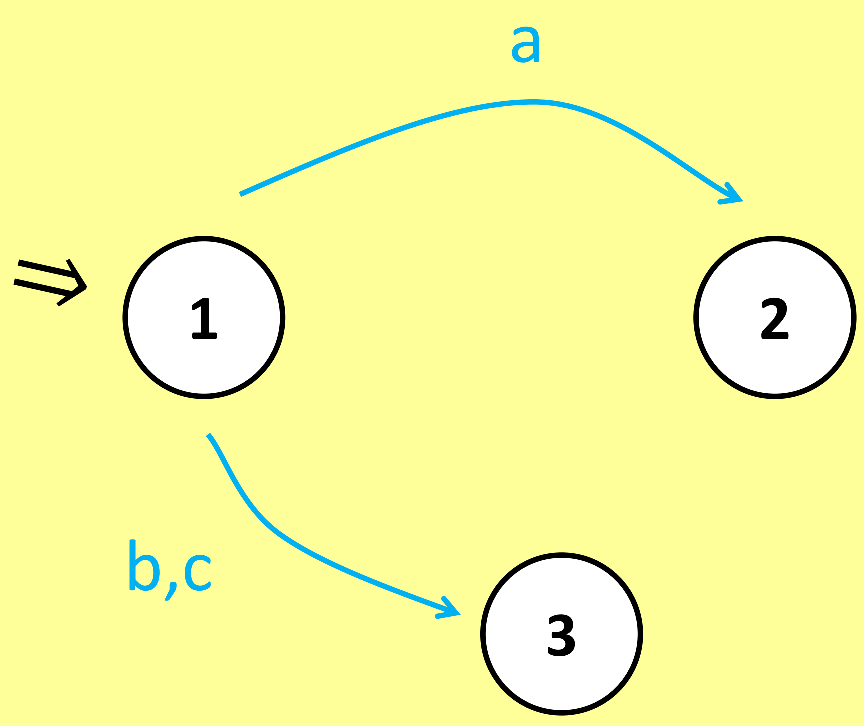
- Arrows show us **transitions** between states

State Machines example



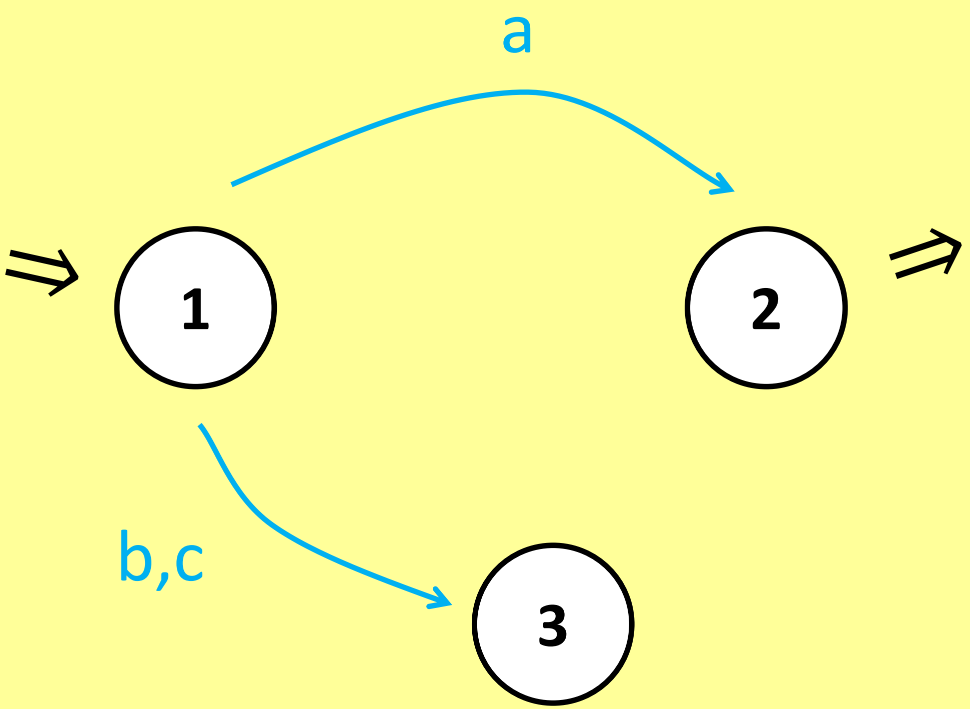
- Arrows show us **transitions** between states
- This one moves the state from **1** to **2** when reading **"a"**

State Machines example



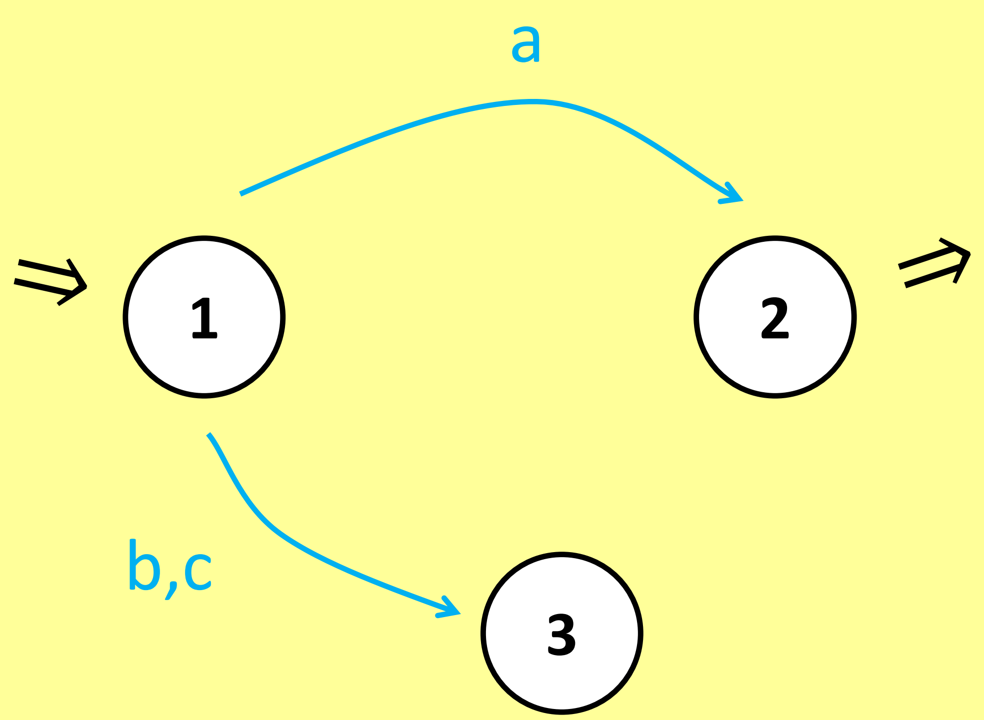
- Let’s add a state **3** where we would get after reading **“b”** or **“c”**

State Machines example

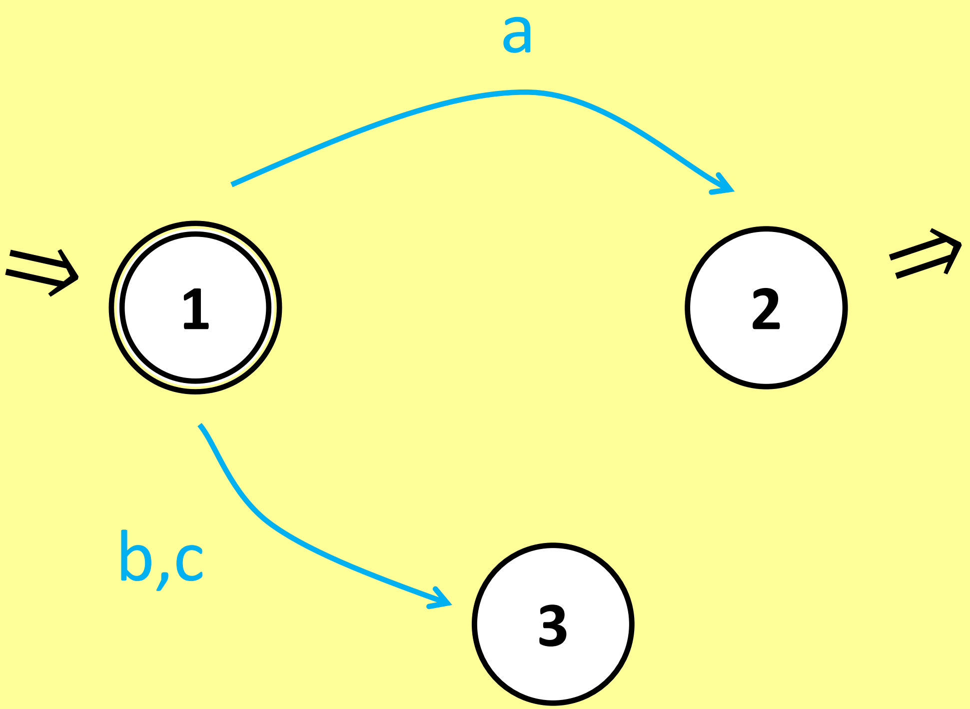


- And finally, we will mark some of the states as **accepting states**, meaning that if we get to them, the machine accepts the given input

State Machines example



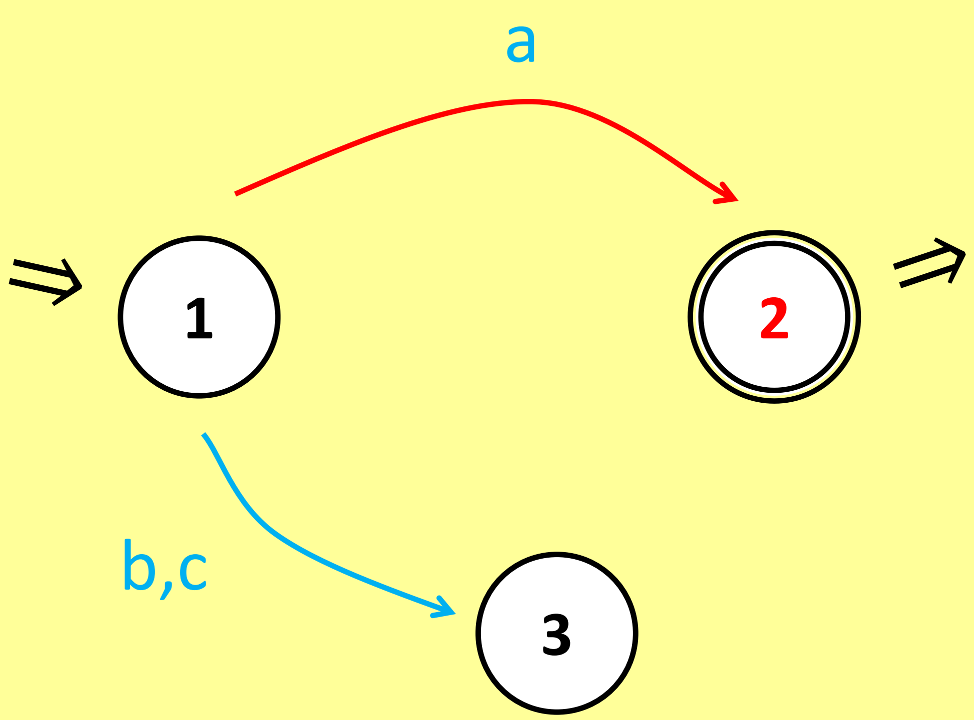
State Machines example



Example 1:

- input is "a"

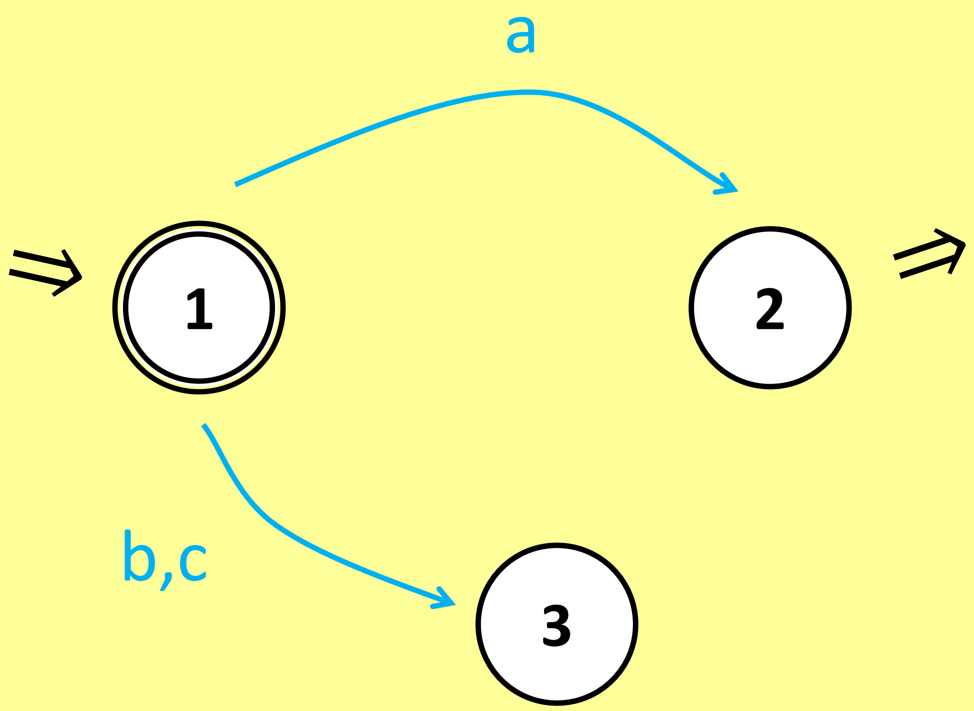
State Machines example



Example 1:

- input is "a"
- We get to the state **2** which is an accepting state – so that means that **this machine accepts the word (or input) "a"**

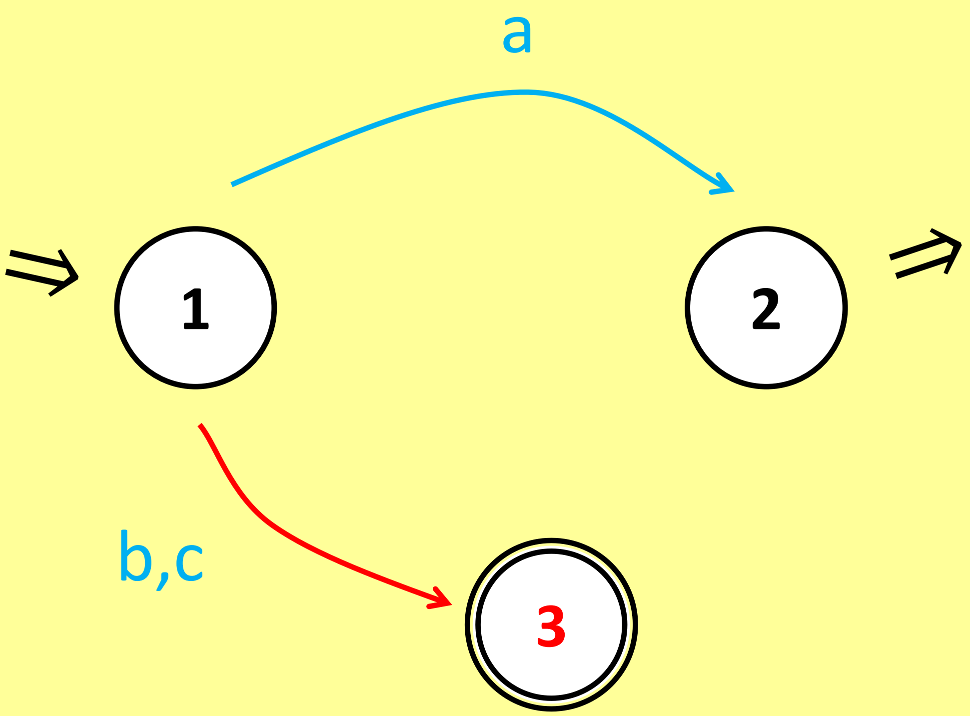
State Machines example



Example 2:

- input is “c”

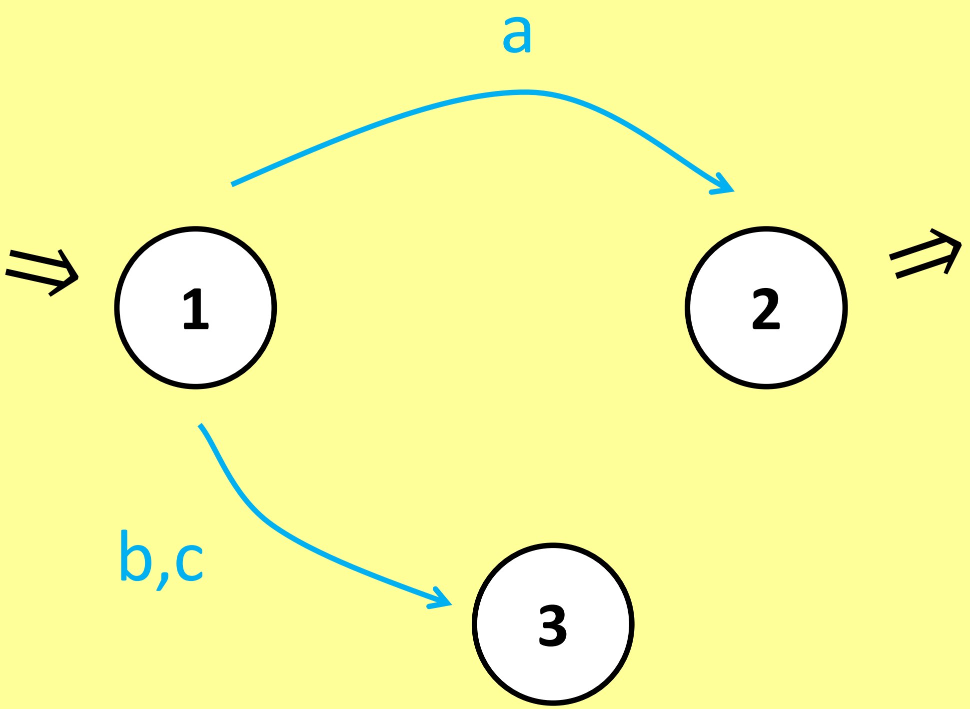
State Machines example



Example 2:

- input is “c”
- We get to the state **3** which is not an accepting state – so that means that **this machine doesn’t accept the word “c”**

State Machines example

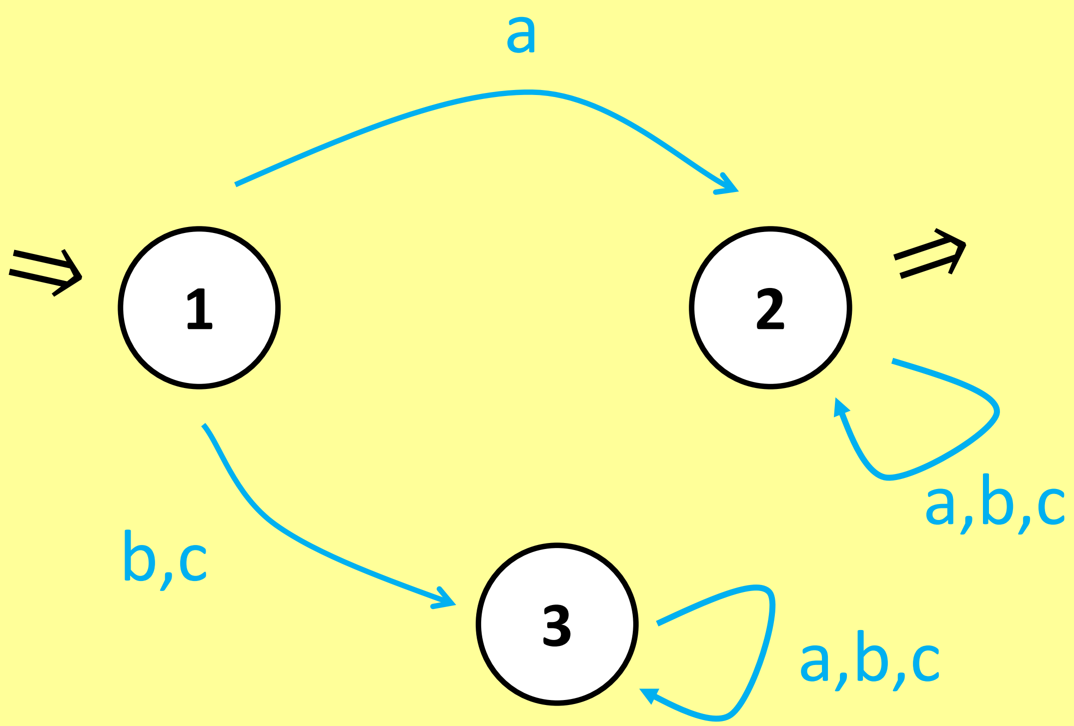


This machine …

… accepts word “a”

… and doesn’t accept “b” or “c”

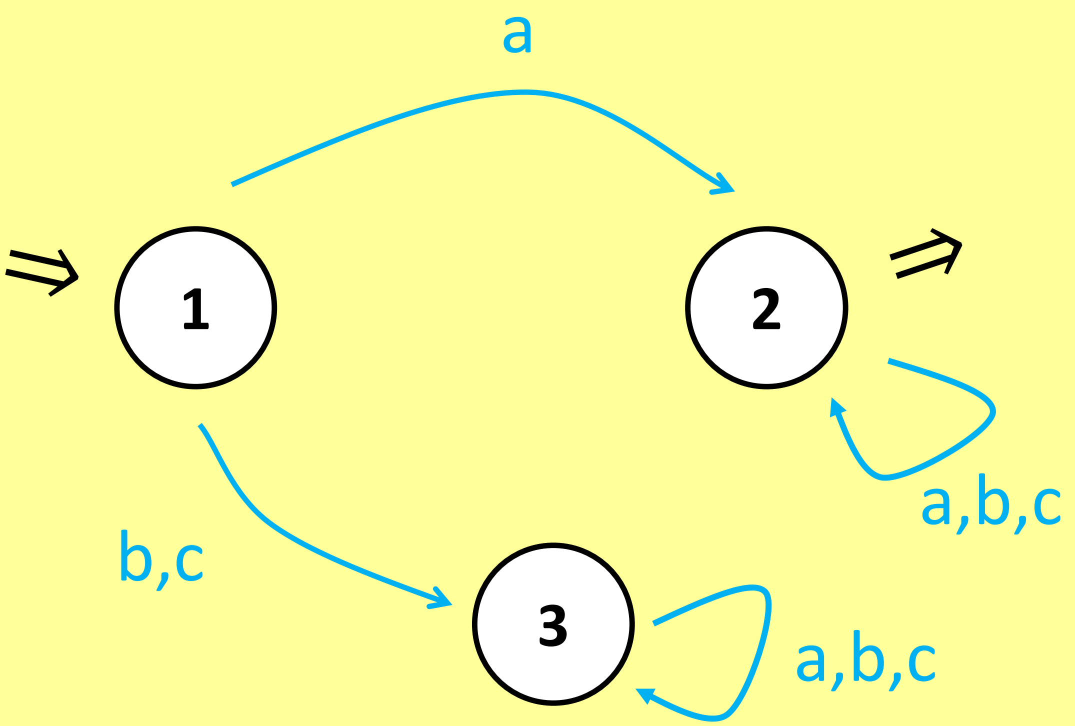
State Machines example



A more complicated machine …

What does it do? (What types of words does it accept?)

State Machines example

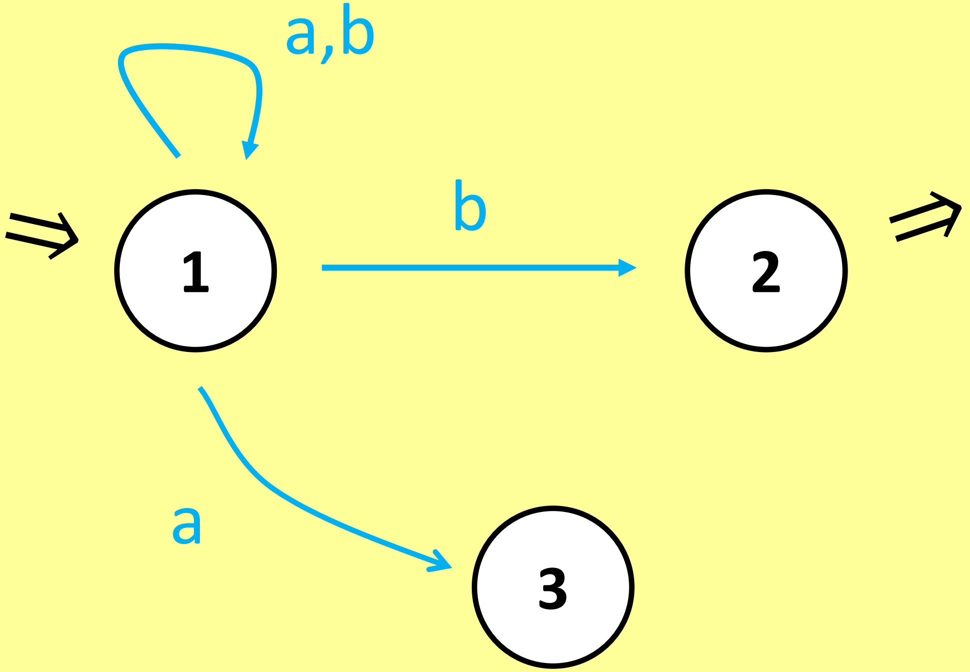


A more complicated machine …

Accepts words starting with “a”.

State Machines example

Until now we always knew where to go, now there are two options:

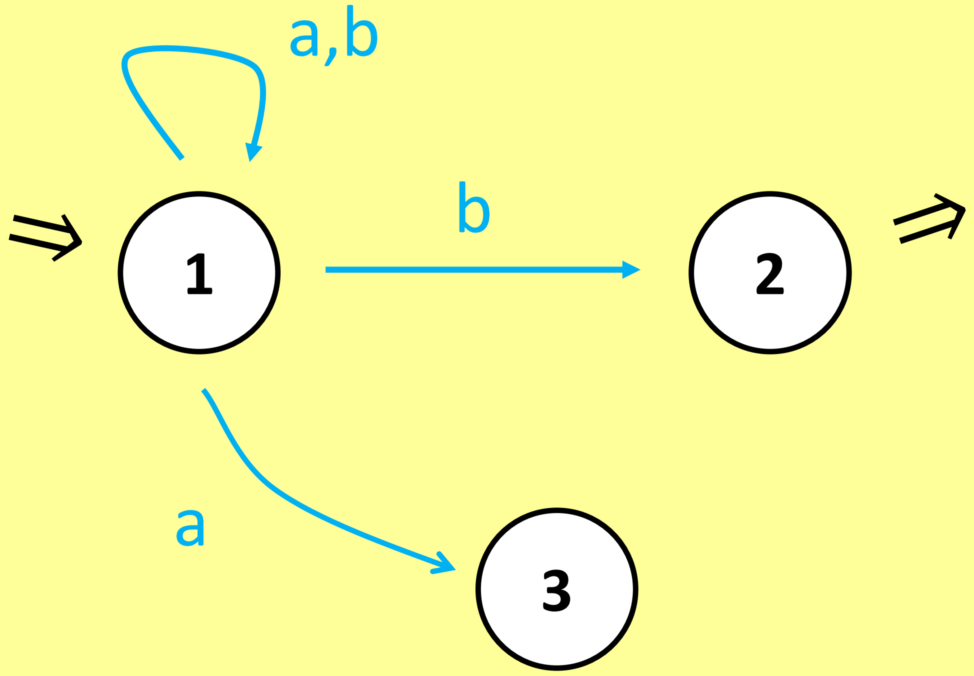


We are in **1** and read "a":

- Stay in **1**
- Go to **3**

State Machines example

Until now we always knew where to go, now there are two options:



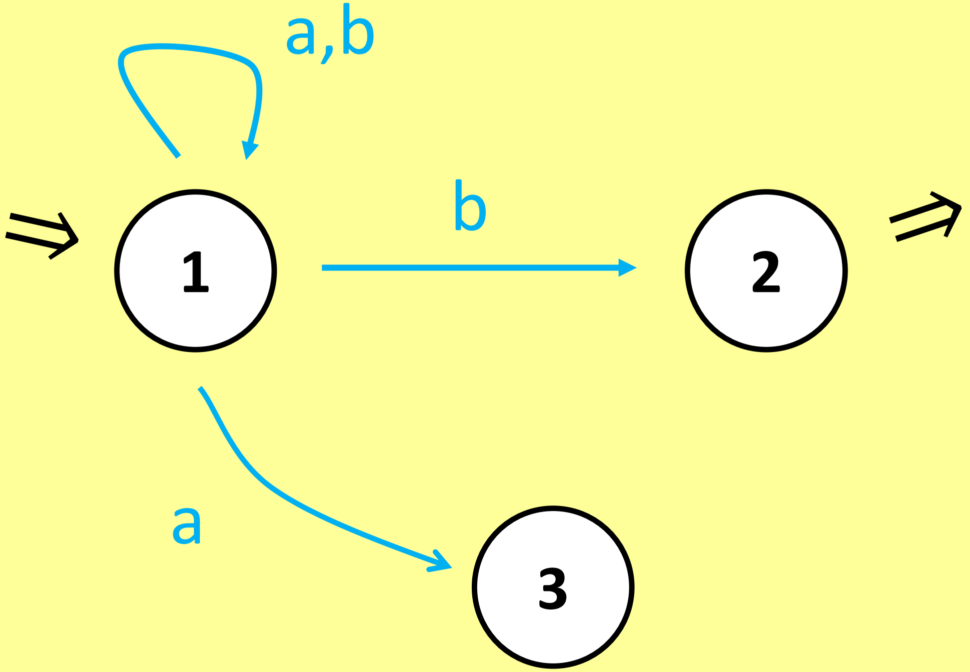
We are in **1** and read "a":

- Stay in **1**
- Go to **3**

We try both of these and wait where we end up after reading the whole input word.

State Machines example

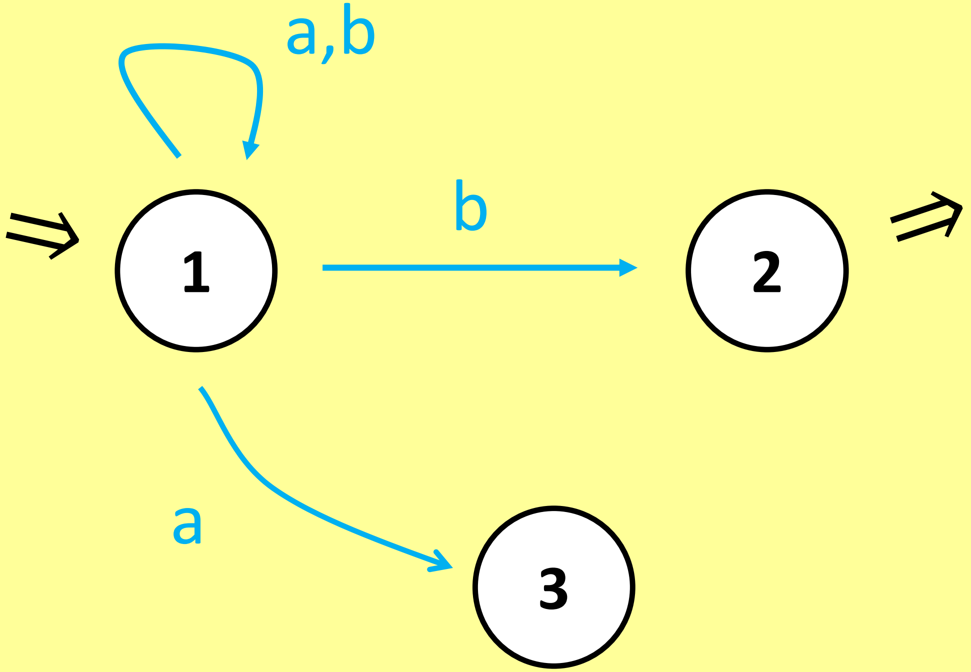
Until now we always knew where to go, now there are two options:



So … **what does this machine do?**

State Machines example

Until now we always knew where to go, now there are two options:



So … **what does this machine do?**

<u>**Accepts any word ending with “b”!**</u>

State Machines

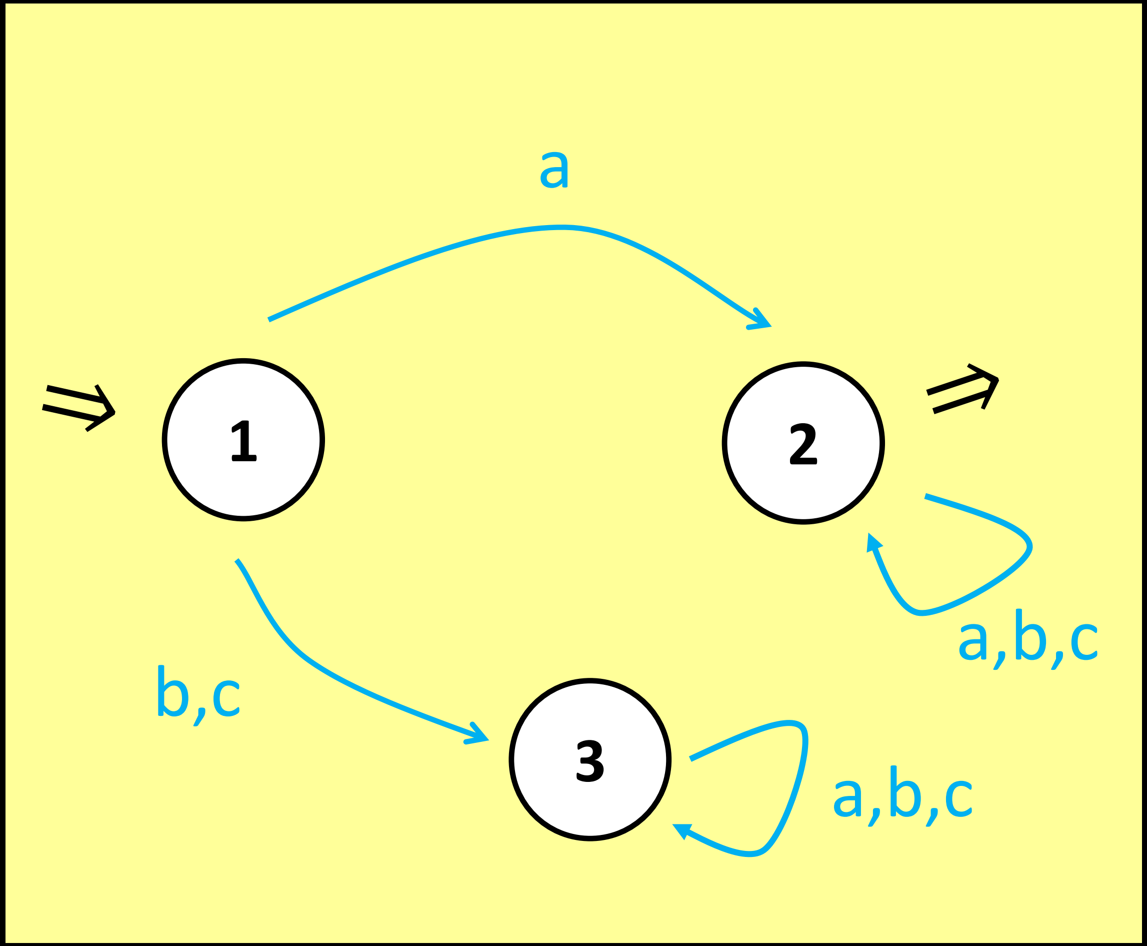
- Finite State Machines:
  - We always know where to go (only one option for a letter from each state)
    => **Deterministic Finite State Machine**

  - We have more options to take (from one of the states)
    => **Non-Deterministic Finite State Machine**

PS: This distinction doesn't matter that much … there even is a proof that we can convert any existing Non-Deterministic FSM into a Deterministic FSM!

State Machines formalized:

- Set of **states** (1,2,3)
  - **Start:** starting state (1)
  - **End:** accepting state (2)
- Accepting **alphabet** (“a”, “b”, “c”)
- **Transitions** between states
  - 1 => 2 with “a”
  - 2 => 2 with “a”, ”b”, ”c”
  - … etc



State Machines - Tasks

- **<u>Let’s design some Finite State Machines:</u>**
  - alphabet = [“a”, “b”]
    => so possible inputs are any words made from these

- **<u>Task 1:</u>** Accepts a word containing only a’s (“a”, “aaa”, …)
- **<u>Task 2:</u>** Accepts “baba” and “abba”
- **<u>Task 3:</u>** Accepts any repetition of “ba” (“baba”, “bababa”, …)
- **<u>Task 4:</u>** Accepts any 3 letter words

State Machines Summary

<u>Why (…are you learning about them)?</u>

- A simple machine design which precedes computers and programming, yet we can *kinda* build programs (creations accepting certain strings) with it.
- It is easy to make it into a real machine (easier than making up an entire computer architecture …)
- There are formal definition, proofs, theory – we can explore some concepts with these designs.

Pause 1

Prime Numbers

- What's a prime number?
  - Def: *A* **prime number** *is a natural number greater than 1 that* ***cannot be formed by multiplying two smaller natural numbers****. A natural number greater than 1 that is not prime is called a* **composite number**.

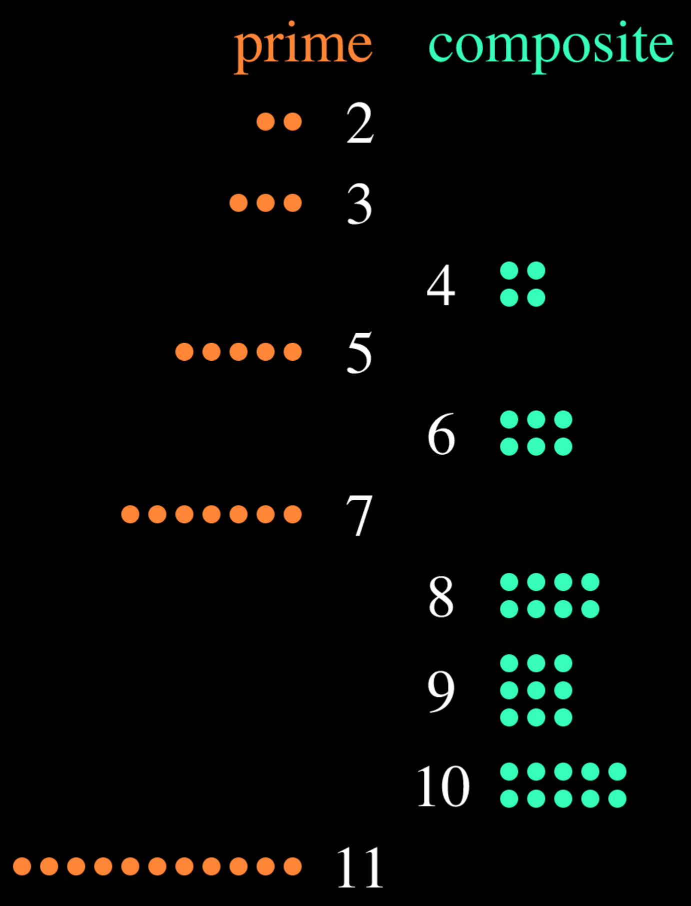
- Divisible without remainder by 1 and itself. (5/5, 5/1)

Prime Numbers

- What troubled mathematicians …
  … are there patterns in prime numbers?
  … what are their properties?

  - “There are infinitely many primes, as demonstrated by Euclid around 300 BC.”

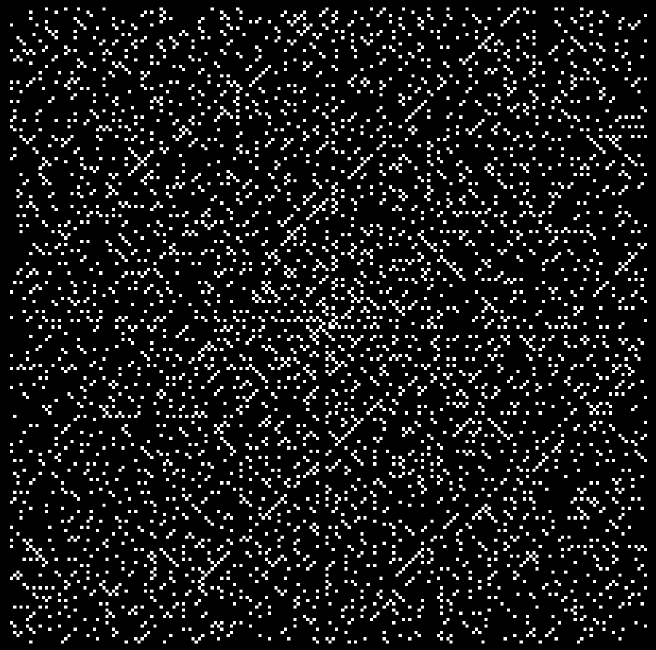
- Prime’s are (computationally) hard to check, which is why they are used in cryptography.



Prime Numbers – Ulam spiral

Ulam spiral of size 200×200. Black dots represent prime numbers. Diagonal, vertical, and horizontal lines with a high density of prime numbers are clearly visible.

Patterns?



Prime Numbers – Ulam spiral

Ulam spiral

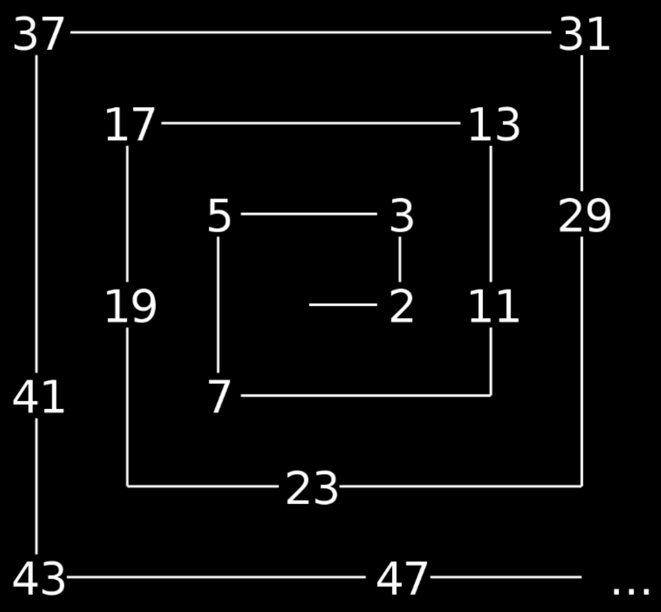
The Ulam spiral is constructed by writing the positive integers in a spiral arrangement on a square lattice:

```
37—36—35—34—33—32—31
|                  |
38  17—16—15—14—13  30
|   |           |   |
39  18   5—4—3  12  29
|   |    |   |  |   |
40  19   6   1—2  11  28
|   |    |      |   |
41  20   7—8—9—10  27
|   |              |
42  21—22—23—24—25—26
|
43—44—45—46—47—48—49...
```

Prime Numbers – Ulam spiral

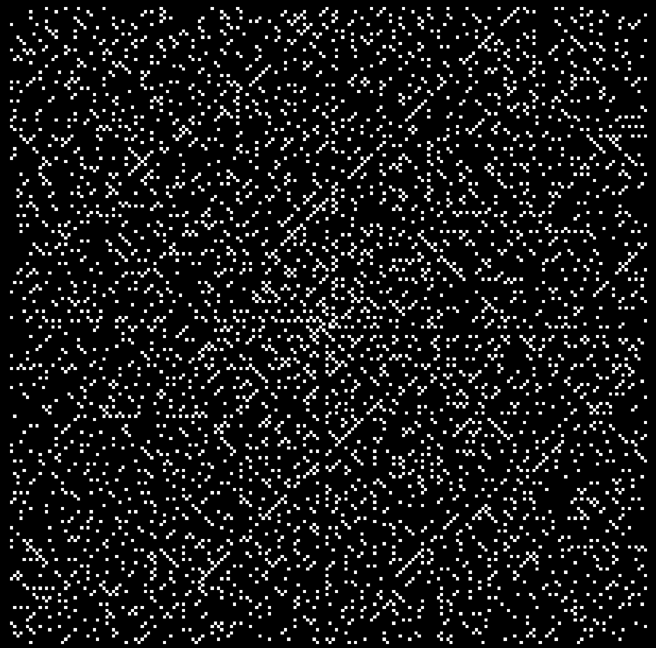
Ulam spiral

… and then marking the prime numbers:



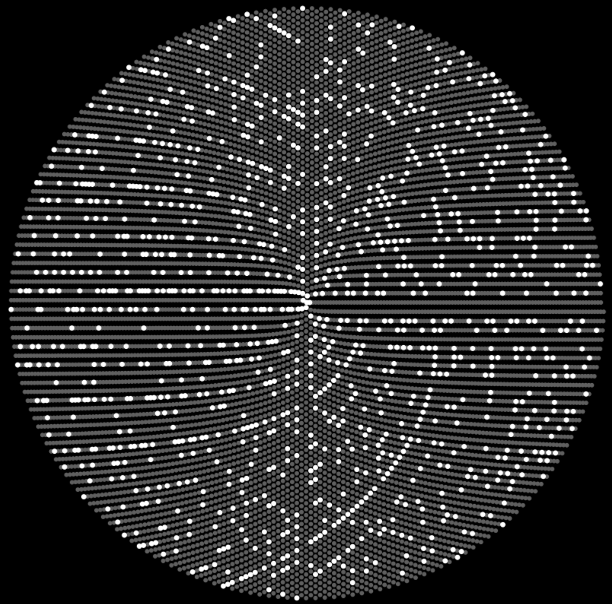
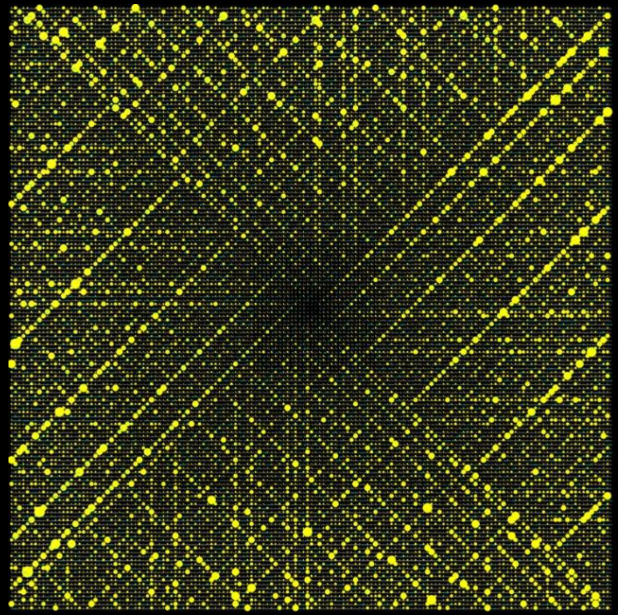
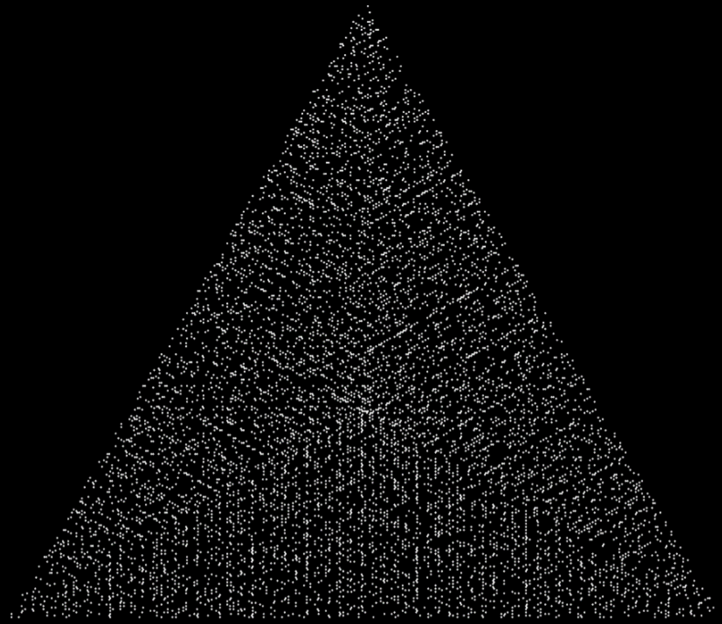
Prime Numbers – Ulam spiral

Visualization:



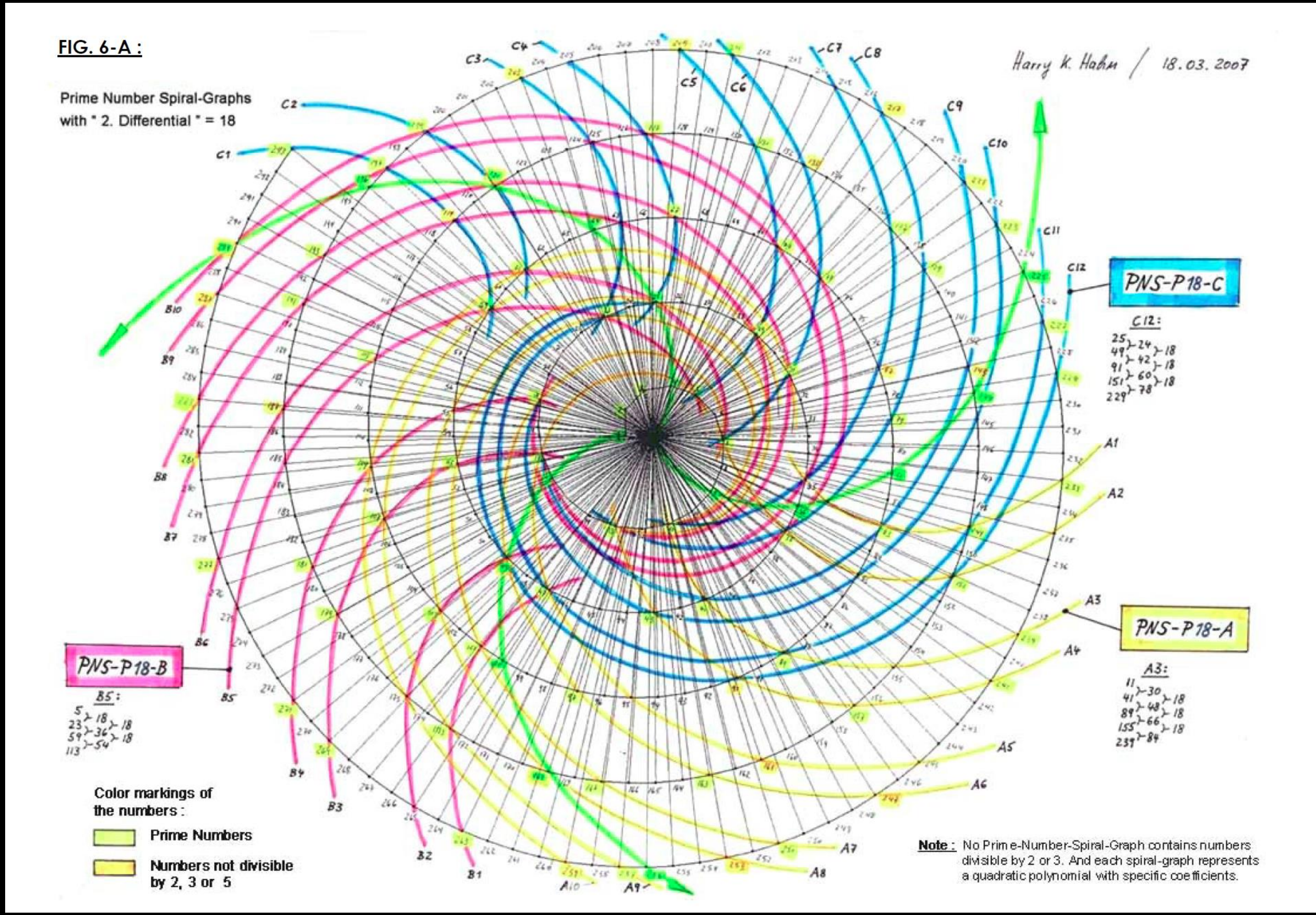
Prime Numbers – Ulam spiral

Same idea, but into different shapes:

Mathematician being excited about them … youtube.com/watch?v=iFuR97YcSLM

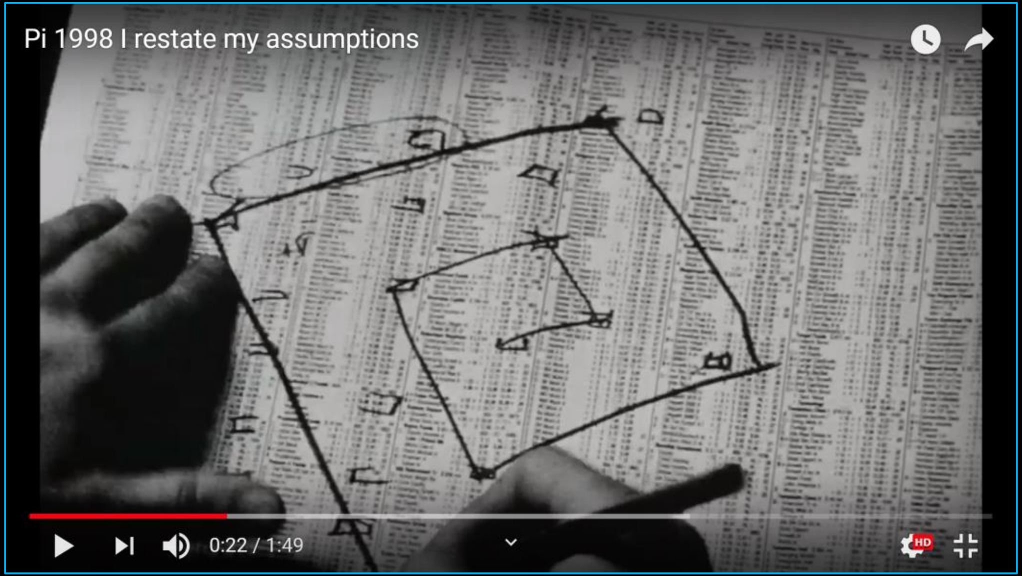
Prime Numbers – Ulam spiral



"This arxiv PDF has to win the insane diagram award for 2008." [blog]

There is a pattern ...

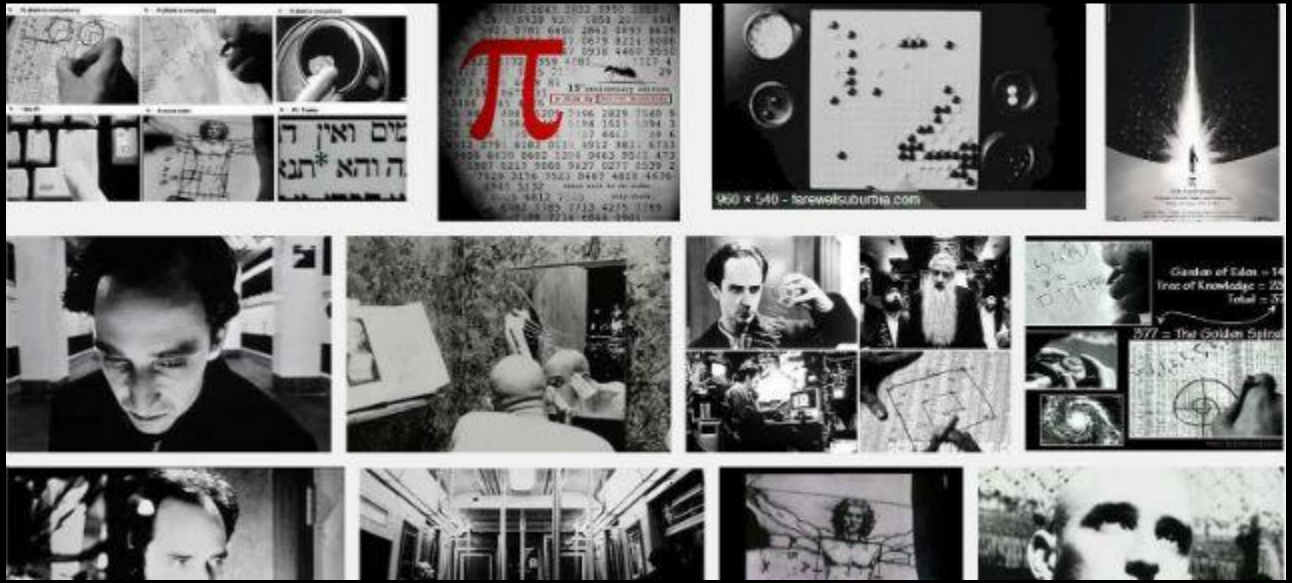
Film: **Pi** (1998) - **Darren Aronofsky**



Watch a clip: youtube.com/watch?v=AdKCLDYHXgM

There is a pattern …

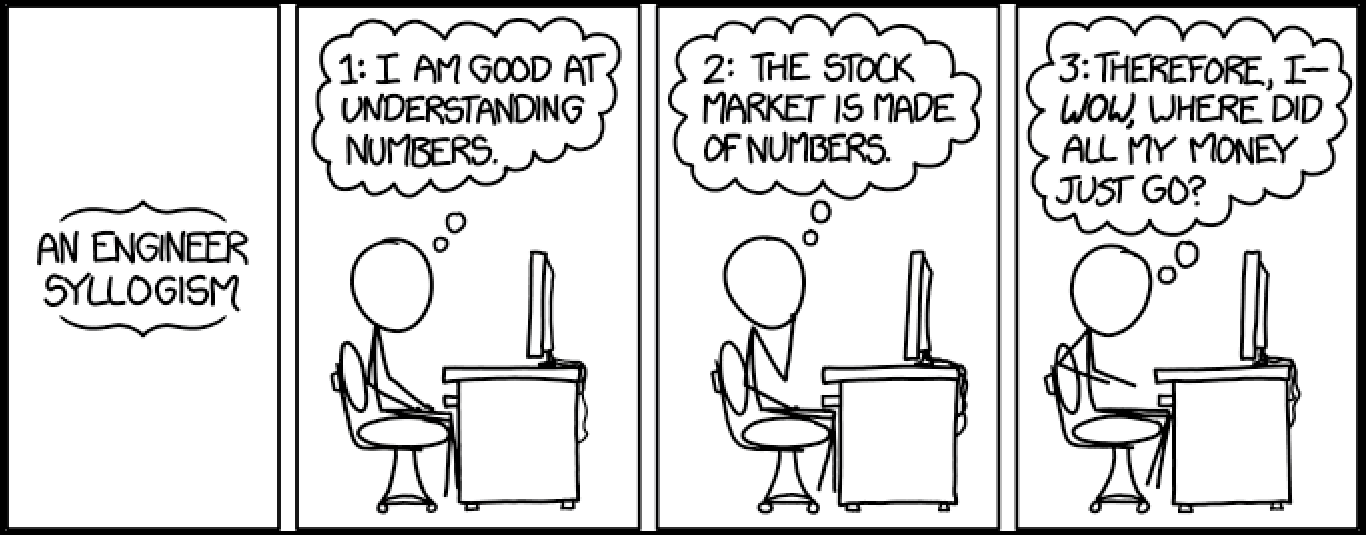
Film: **Pi** (1998) - **Darren Aronofsky**



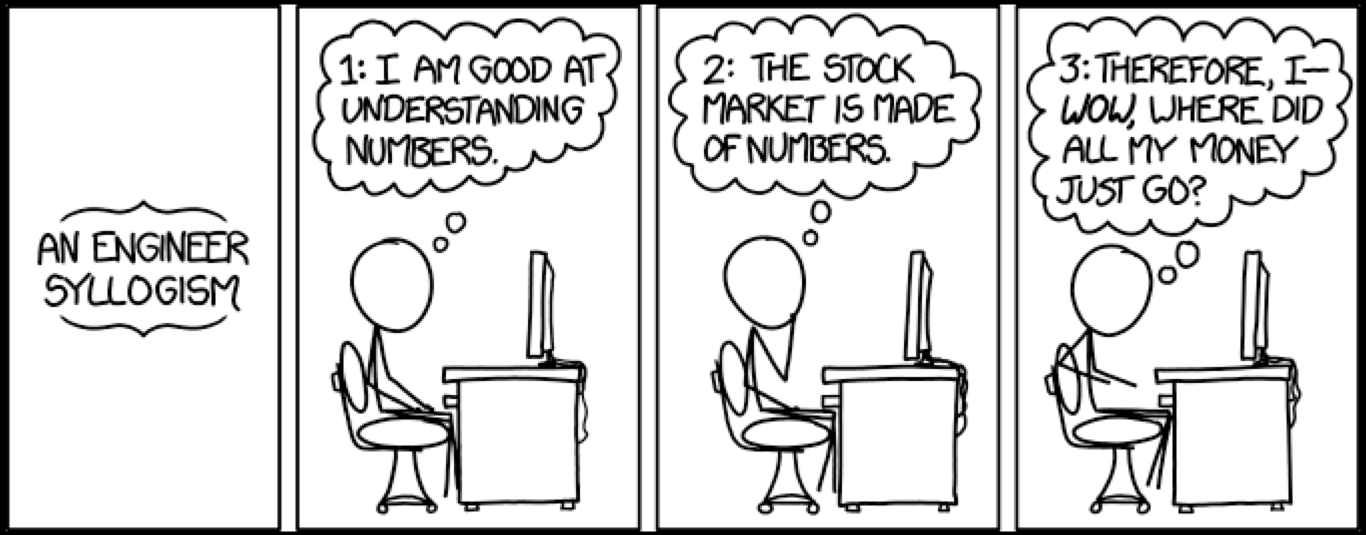
I restate my assumptions: One, Mathematics is the language of nature. Two, Everything around us can be represented and understood through numbers. Three: If you graph the numbers of any system, patterns emerge. Therefore, there are patterns everywhere in nature.

PS: Aronofsky ~ Lynch ~ Cronenberg

There is a pattern …



There is a pattern …



Mouseover: The less common, even worse outcome: "3: [everyone in the financial system] WOW, where did all my money just go?"

Prime Numbers algorithm

- Algorithm “Eratosthenes sieve”

Eratosthenes of Cyrene (276 BC - 195 BC)
Greek polymath

Prime Numbers algorithm

- Algorithm to get all prime numbers between 1 and chosen N:

  We start by having all the numbers marked as potentially prime:

	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

Prime Numbers algorithm

- Algorithm to get all prime numbers between 1 and chosen N:

  One by one we go up in this list …

	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

Prime Numbers algorithm

- Algorithm to get all prime numbers between 1 and chosen N:

  One by one we go up in this list … and when we visit a number (like 2), we will cross out all it's multiples (so 4,6,8, …) as we know that these are composite:

	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

Prime Numbers algorithm

- Algorithm to get all prime numbers between 1 and chosen N:

  Then we go up again until we reach a number which wasn't crossed out:

	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

Prime Numbers algorithm

- Algorithm to get all prime numbers between 1 and chosen N:

  … and we repeat … (*crossing out multiples of 3*)

	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

Prime Numbers algorithm

- Algorithm to get all prime numbers between 1 and chosen N:

  … and we repeat … (*selecting 5*)

	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

Prime Numbers algorithm

- Algorithm to get all prime numbers between 1 and chosen N:

  … and we repeat … (*crossing out multiples of 5*)

	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

Ps: we can start checking with numbers going from 5*5 (any previous one would have already been caught in the sieve)

Prime Numbers algorithm

- Algorithm to get all prime numbers between 1 and chosen N:

  (*7*7 = 49 is outside the range, so all the remaining number are prime*)

	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

Prime Numbers algorithm

- **Task: code Eratosthenes sieve in Python!**

Prime Numbers algorithm

Hints for "Eratosthenes sieve" … roughly:

- Mark all numbers as prime in a supporting data structure
- Go through the list …
  - If we visit a not crossed out number, it's a prime
  - Then we have to cross out it's larger multiples
- In the end only real prime numbers remain

- **Task: code this in Python!**

Next class?

- Probably (tell me folks!) …
  … probably repetition of the topics we did last classes, more examples calculated on paper.

Links?

- Finite State Machine: https://www.youtube.com/watch?v=4rNYAvsSkwk

- Eratosthenes Sieve: https://www.khanacademy.org/computing/computer-science/cryptography/comp-number-theory/v/sieve-of-eratosthenes-prime-adventure-part-4