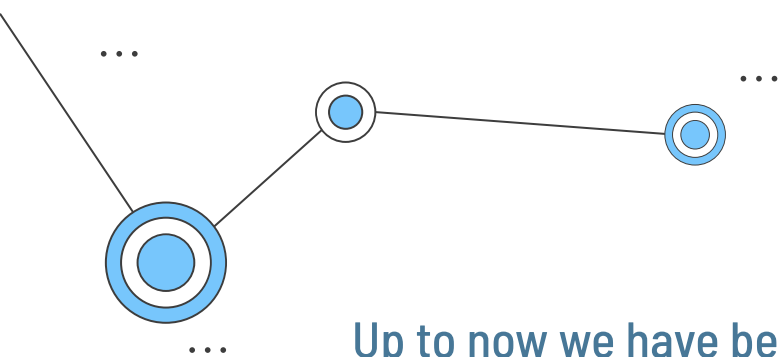
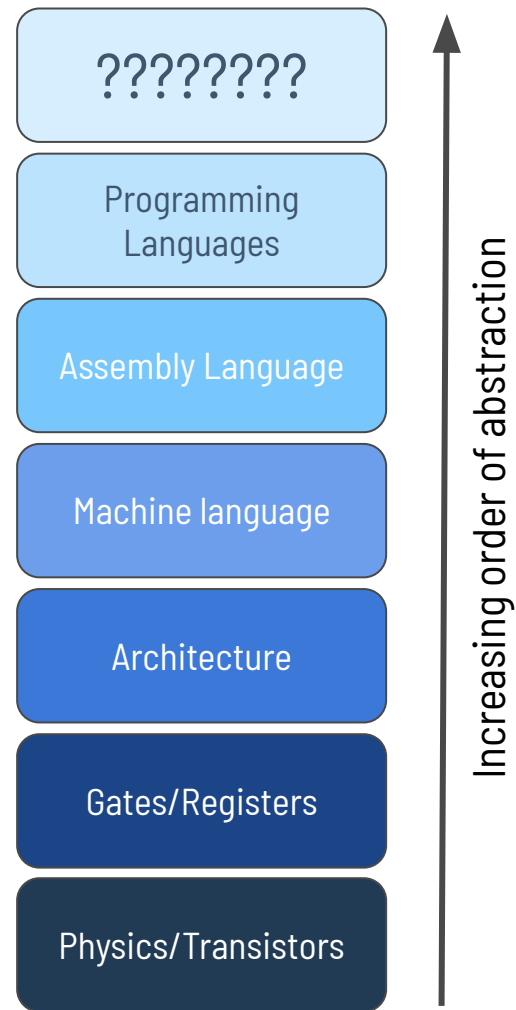
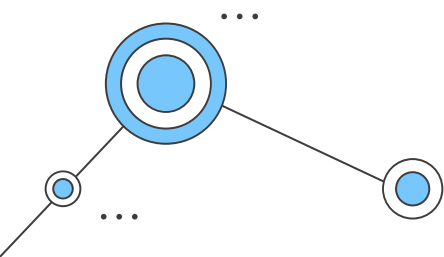


# Algorithms & Problem Solving

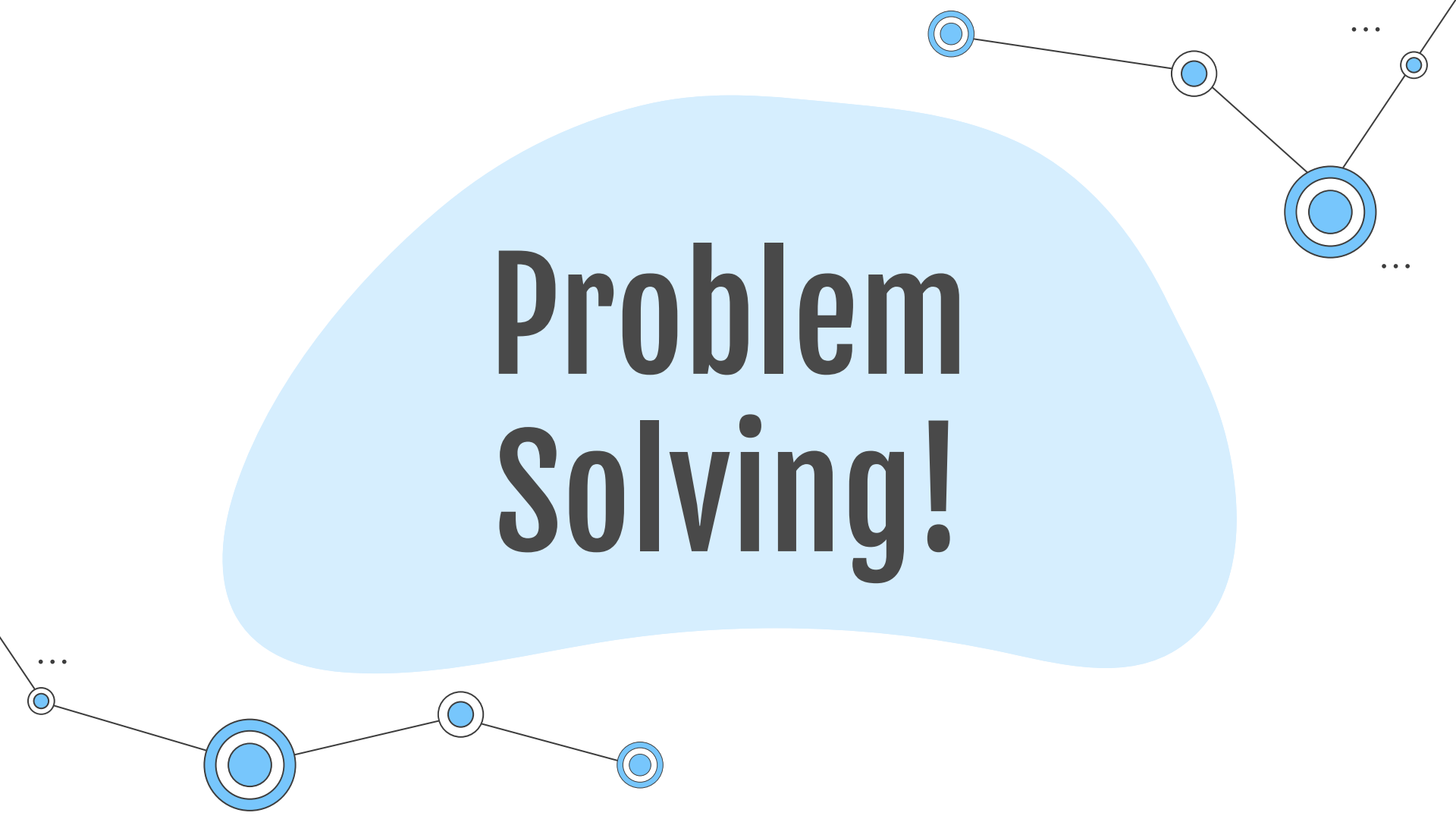


Up to now we have been learning about what is going on inside our computers and how the levels of abstraction work up to the high level programming languages.

Now the question is, what can we use the high level languages and the power of our machines to do? In other words, what is the next level up?



# Problem Solving!



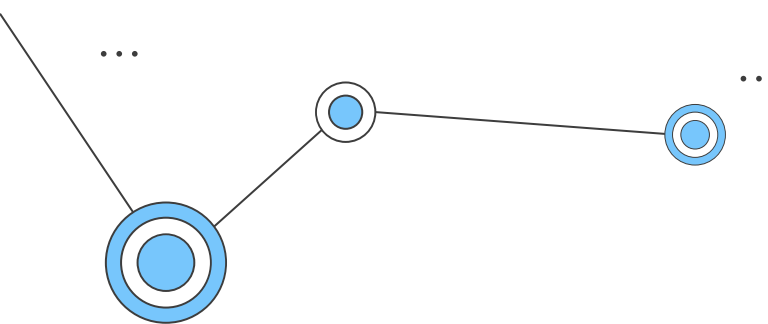
# Computer Science as Problem Solving

We've been talking about computers so far in our discussion of computer science.

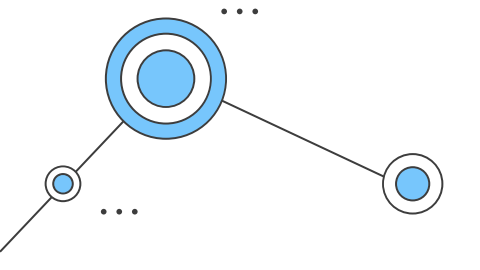
But computers are only a small part of the discipline, they are the means to an end. Computer science is not just the study of computers.

From the reading for today, "computer science is the study of problems, problem-solving, and the solutions that come out of the problem-solving process." Computers are the tools that we use to solve problems.





Computer  
scientists develop  
**algorithms** to solve  
problems



**Algorithms**

Programming  
Languages

Assembly Language

Machine language

Architecture

Gates/Registers

Physics/Transistors

Increasing order of abstraction

# What is an algorithm?

## Definition #1

"A process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer."

- Oxford Dictionary

## Definition #2

In mathematics and computer science, an algorithm is a self contained sequence of actions to be performed. Algorithms can perform calculation, data processing and automated reasoning tasks.

- Wikipedia 2018

## Definition #3

In mathematics and computer science, an algorithm is a finite sequence of well-defined, computer-implementable instructions, typically to solve a class of problems or to perform a computation. Algorithms are unambiguous specifications for performing calculation, data processing, automated reasoning, and other tasks.

- Wikipedia 2020

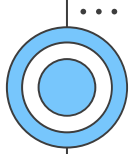
# We use algorithms every day

The classic example is a recipe. A recipe is a high level algorithm for making a particular dish or food.

When using a recipe, it is important that the instructions are clear and unambiguous. Consider this class peanut butter and jelly example:

<https://www.youtube.com/watch?v=Ct-100UqmyY>





# Problem Solving (Algorithm Forming) Techniques

There are a number of ways to go about forming an algorithm which will be discussed more in depth in the next lecture:

1. **Divide and conquer**
  - a. Divide the problem into smaller problems, solve the small problems and then recombine the solutions to get the final solution
2. **Greedy algorithms**
  - a. At each step in the algorithm, a decision is made that choose the optimal choice without thinking about the future
3. **Dynamic programming**
  - a. Storing the solutions to smaller problems so that the same small problems don't have to be solved over and over again, they just get solved once.
4. **Reduction**
  - a. Solving a difficult problem by transforming it into an easier problem that we know how to solve



...



...



# So what makes an algorithm “good”?

## How can we evaluate them?

### Time analysis

How long does the algorithm take to produce the result?  
As the problem gets larger, how does the time scale up?

### Space analysis

How much of the computer's memory does the algorithm use?  
Would we eventually run out of memory if the problem was large enough?

This discussion will take place in the “evaluating algorithms” lecture

# Can we make an algorithm to solve any problem?



No!

There is a notion in computer science of some problems being “computable” and others being “uncomputable”. Computable problems have algorithms that can be used to solve them.

In a later lecture we will discuss how problems can be uncomputable and we will also discuss the notion of some problems being “hard” to compute.