

Computability, complexity, feasibility



Computability

- ◆ The computability problem: is an algorithm computable? It has a solution?
- ◆ The halting problem: Does an algorithm finish?
 - ◆ Video: [The Halting Problem](#)

In order to answer these questions we need a

MACHINE MODEL

The Finite State Machine

- ◆ Its a model to describe a computational automata
- ◆ It has a finite number of possible internal state
- ◆ The transition from a state to another depends to the actual state and the value of the input

The Turing Machine

- ◆ It is a powerful model for describe computational machines
- ◆ It consist of a finite state machine that can read and write to a potentially infinite memory ribbon.
- ◆ Any computational model can be reduced to the Turing machine.

Results on computability

- ◆ There exists some program that cannot be computed i.e. for which it cannot determine the value.
- ◆ There exists some program for which it cannot determine if it finish or not.

Complexity

- ◆ For computable programs there is another problem: how time and resources are required to reach a solution?
- ◆ For "large" problems, it is not so important the exact time spent to resolve the problem but the "trend" respect the "size" of inputs.

O notation

- Given a problem P and an algorithm A that resolve P an input of "size" n for the problem P , the notation $O(n)$ means that for "big" n the time for reach solution is $O(n)$, for ex:
 - $O(n)=k$, constant time
 - $O(n)=n$, linear time
 - $O(n)=n^2$, polinomial time \rightarrow P problem
 - $O(n)=e^n$, exponential time \rightarrow NP Problem

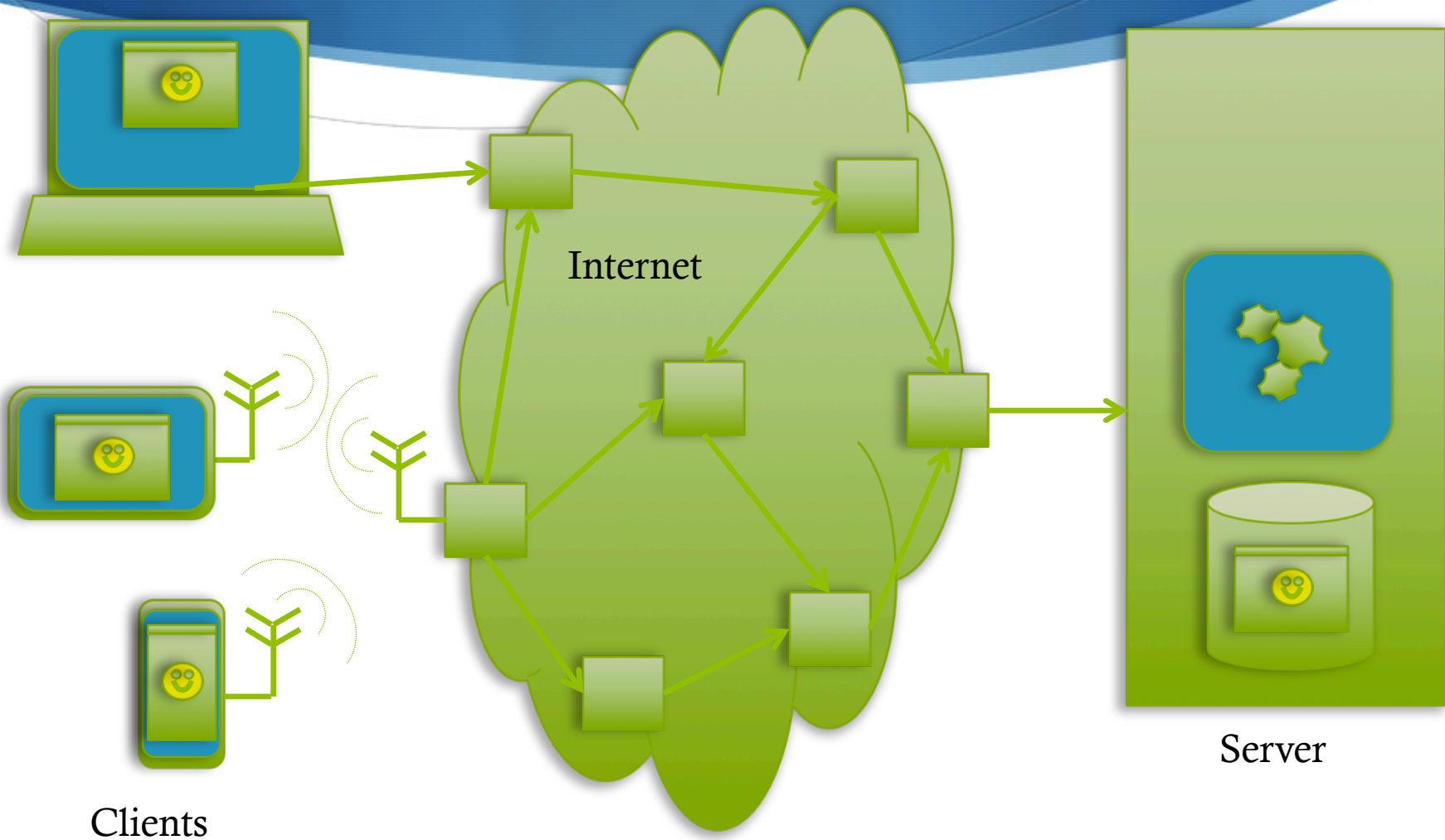
P \leftrightarrow NP Assumption

- ◆ polynomial time algorithm has considered "easy" to compute, though "big" n can be very hard to compute.
- ◆ not polynomial time algorithm, for example exponential time algorithm, has considered "hard" to compute, though can be not so hard to compute for "little" exponents and "little" n .
- ◆ It is widely assumed (though not proved) that exists NP problem that isn't P, i.e. that cannot be resolved "easily".

How to resolve a problem

- ◆ A programming paradigm it is a "way" to describe a solution method for a problem:
 - ◆ imperative languages: C, java, basic
 - ◆ declarative languages: ASP
 - ◆ functional: prolog

Client-Server Architecture



Client constraints

- ◆ bandwidth (mainly smartphone)
- ◆ battery (mainly smartphone)
- ◆ software capability (OS, supported apps)
- ◆ CPU
- ◆ RAM
- ◆ Disk space

Your app/site must be designed in order to match the expected (and acceptable) constraints on client side

Server constraints

- ◆ bandwidth (mainly smartphone)
- ◆ Number of contemporaneous sessions
- ◆ Disk space
- ◆ Cost
- ◆ CPU & RAM (less important for a common website)

The server "size" must be designed in order to balance the tradeoff between user experience and cost