



DESIGN PATTERNS
SOLID PRINCIPLES
DOMAIN DRIVEN DESIGN
SOFTWARE ARCHITECTURE
STANDARD RECOMMENDATIONS

`<p> Here we go! </p>`

HELLO! I'm...

`<p> Luís Paulo Toniette França </p>`

``

` Software Engineering Manager `

` Information Security Consultant `

` Lovely Husband/Son/Brother/Friend `

` Liz's dad `

``



TABLE OF CONTENTS.



01

What is Object
Oriented Programming.



02

What is SOLID
Principles.



03

What is Design
Patterns.



04

What is Domain Driven
Design and Software
Architecture.



05

Where have we arrived
and where should we go?



06

References and
acknowledgments.



01 WHAT IS OBJECT ORIENTED PROGRAMMING

`<p> If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization. </p>`



ENCAPSULATION

```
<p hidden> Let's keep it a secret, OK? </p>
```

A computer lets you make more `mistakes` faster than any other `invention` with the possible exceptions of `handguns` and `Tequila`.



ABSTRACTION

`<p> # $? = ~ | < ! *) @ ; " / + < / p >`

There is no `abstract` art. You must `always` start with something. Afterward you can `remove` all traces of `reality`.



INHERITANCE

`<p> # $? = ~ | < ! *) @ ; " / + < / p >`

To the `brains` of our `predecessors` we owe all of `our` inheritance of civilization and culture.



POLYMORPHISM

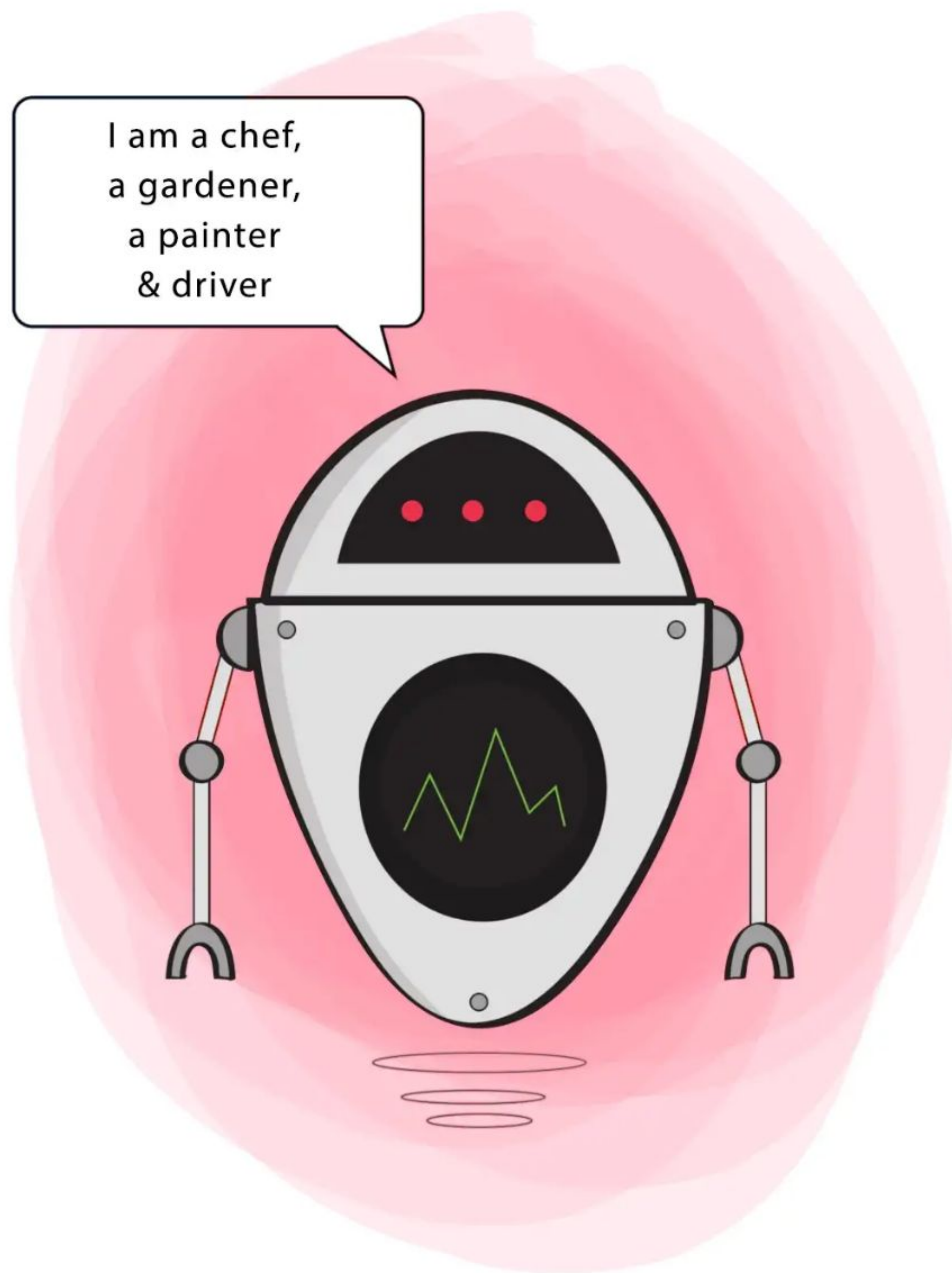
```
<p> |+*)~@#?!$"/<;= </p>
```

To the brains of our predecessors we owe our inheritance of civilization and culture.

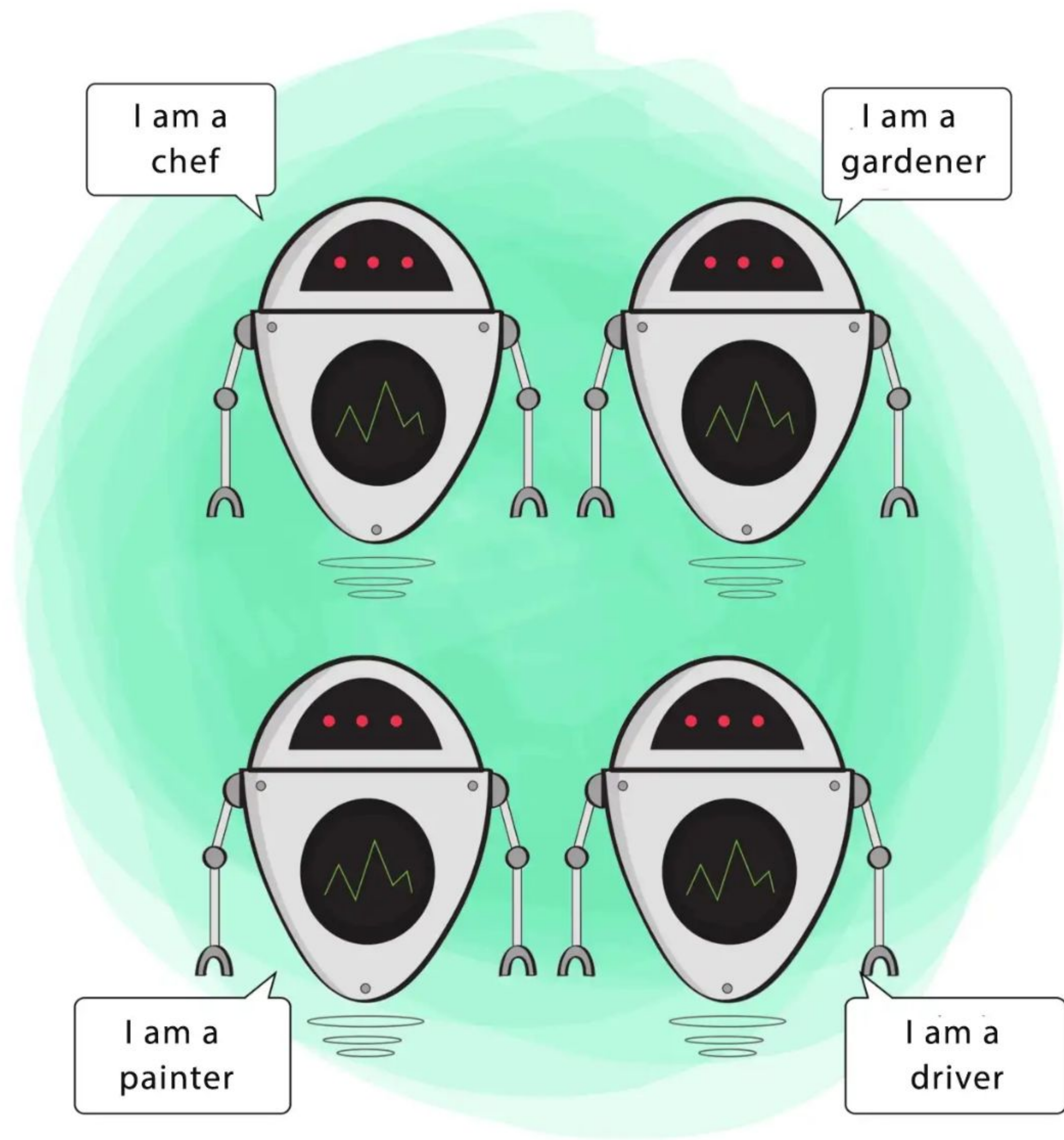


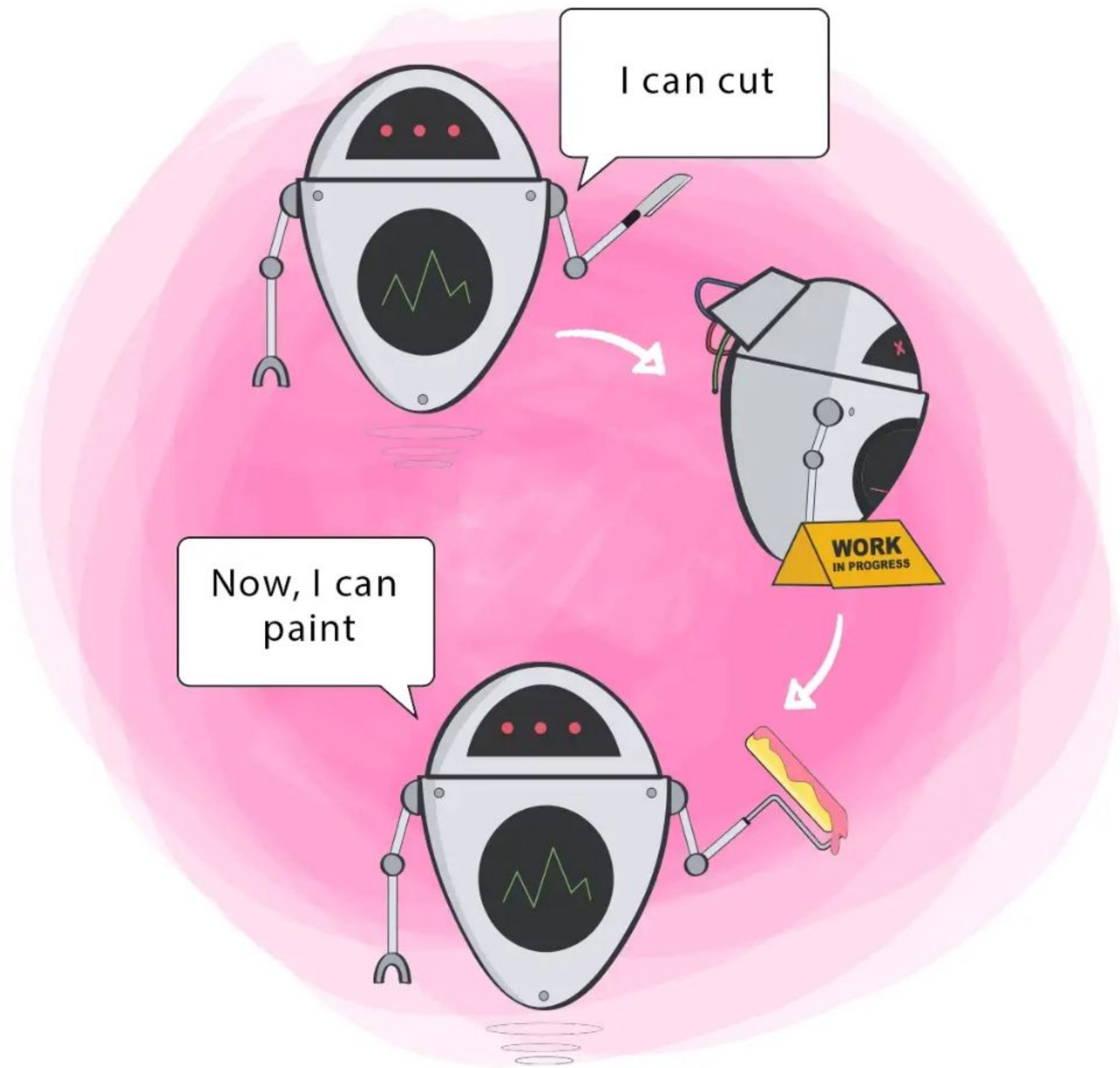
02 WHAT IS SOLID PRINCIPLES

`<p> The first 90% of the code accounts for the first 10% of the development time. The remaining 10% of the code accounts for the other 90% of the development time. </p>`



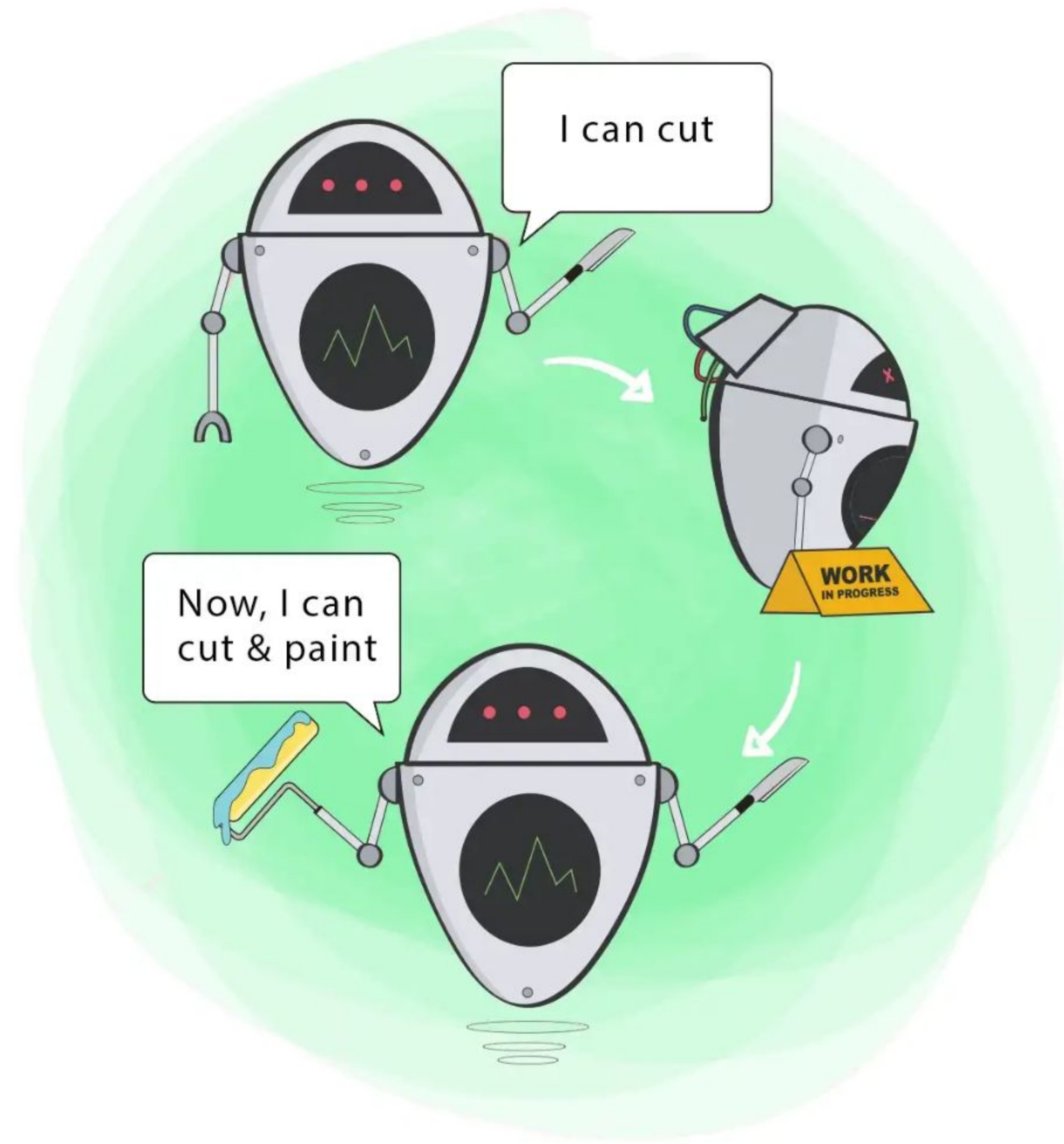
Single Responsibility



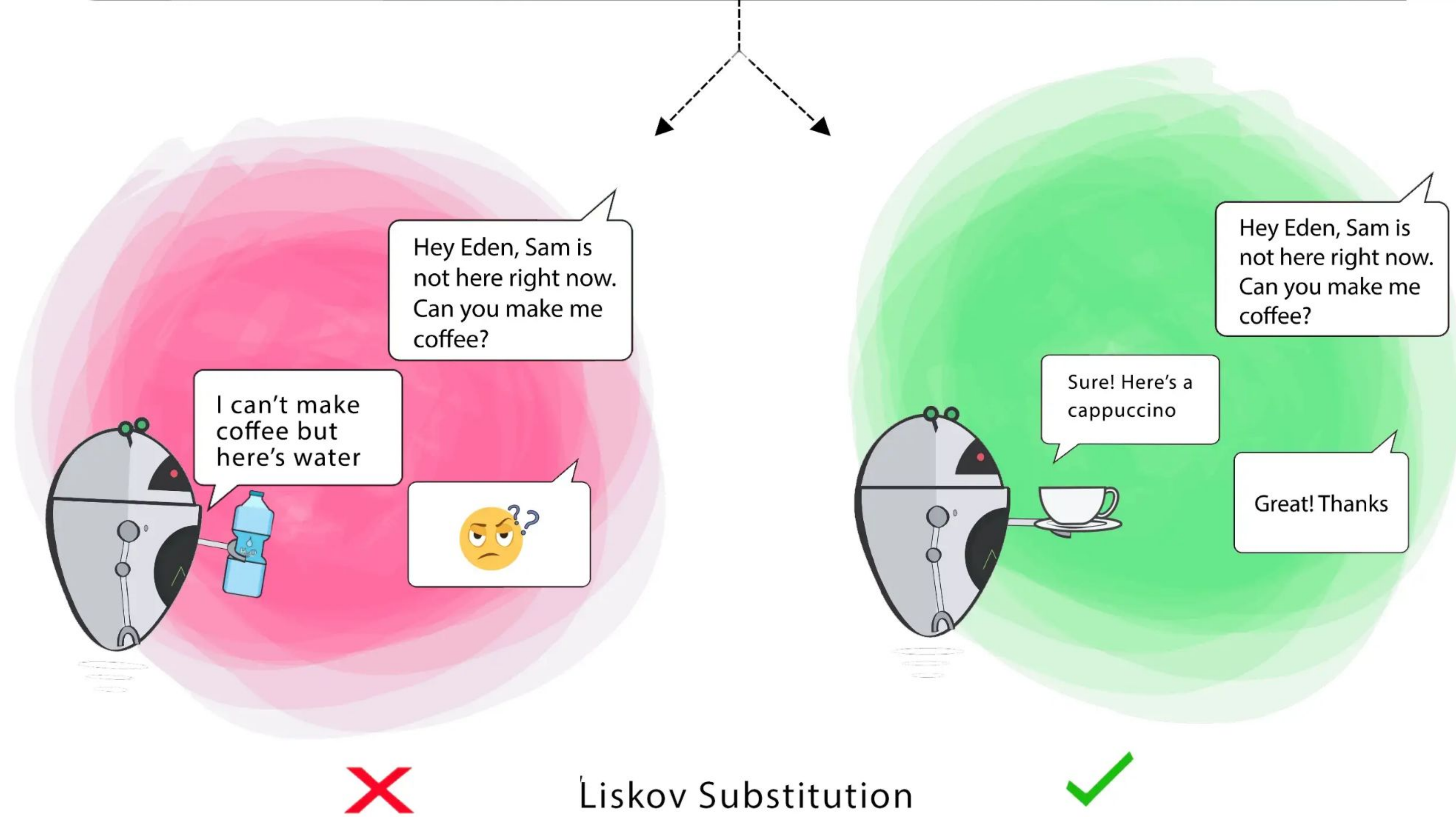
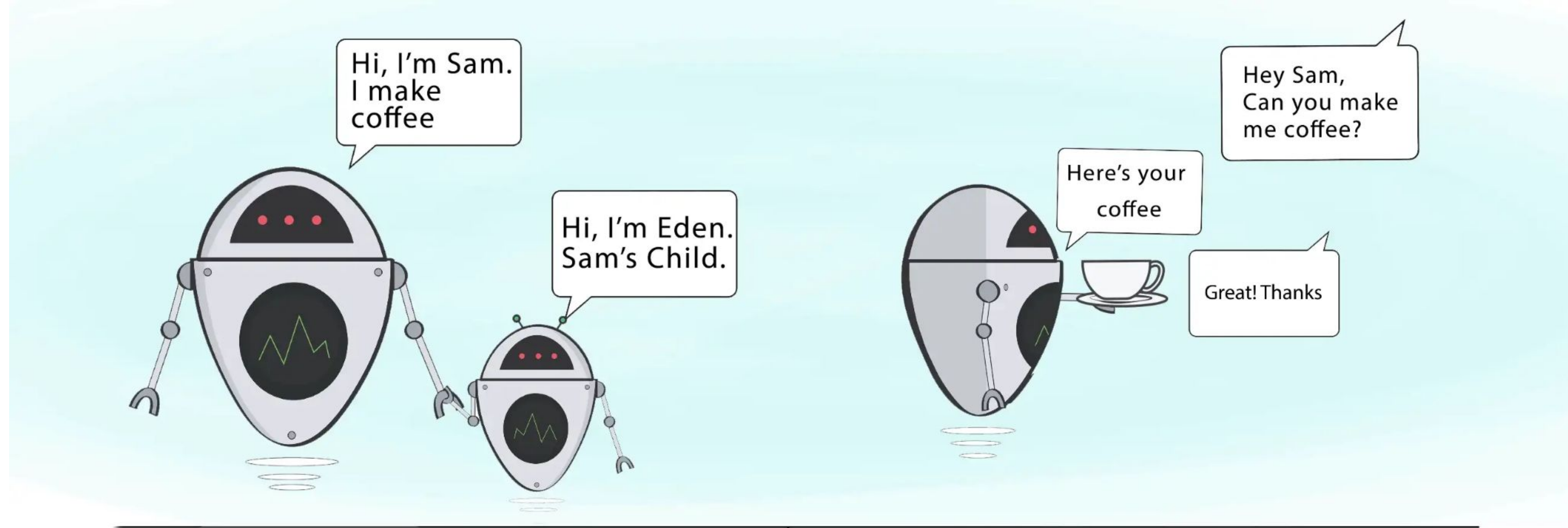


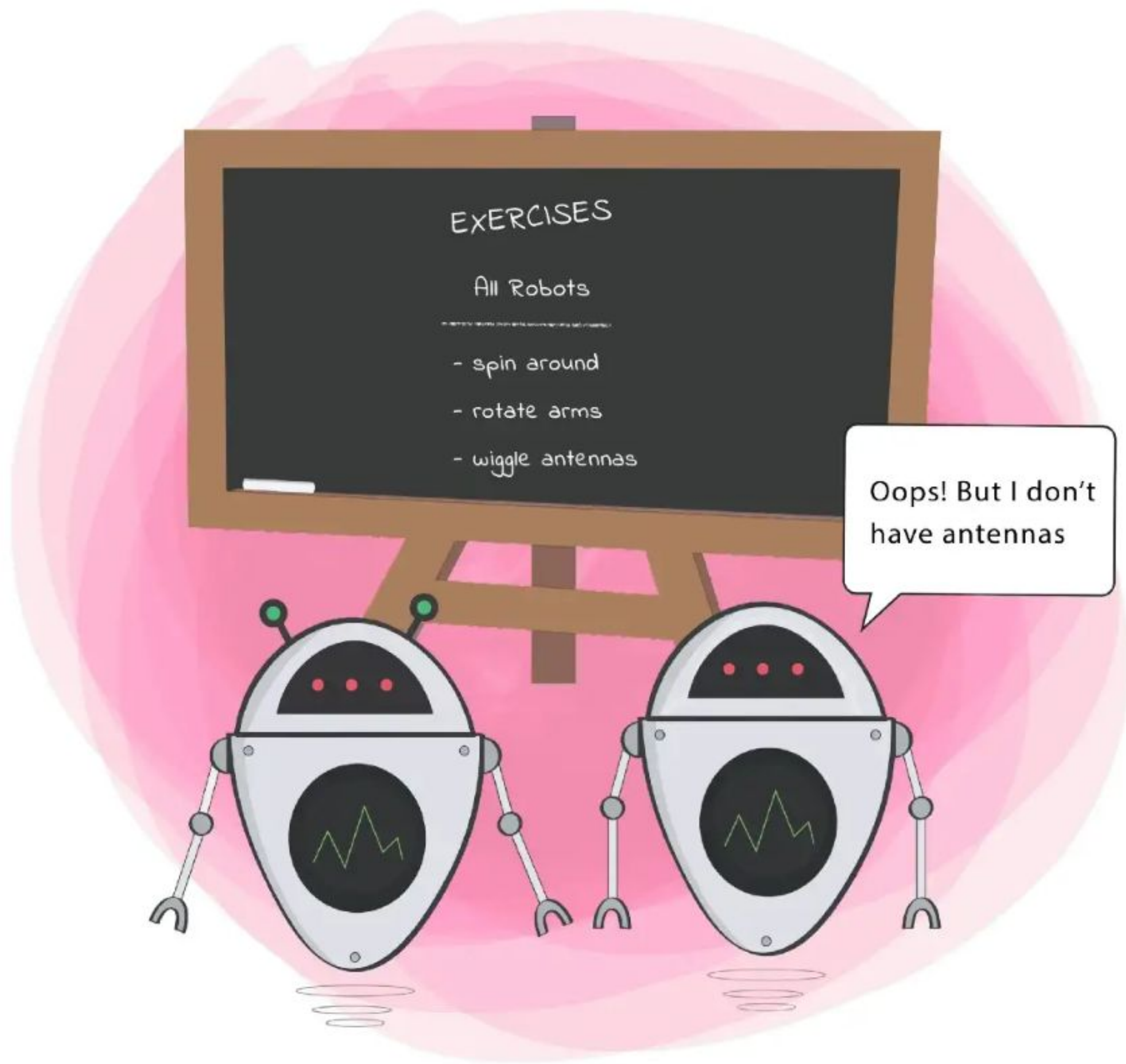
✗

Open-Closed

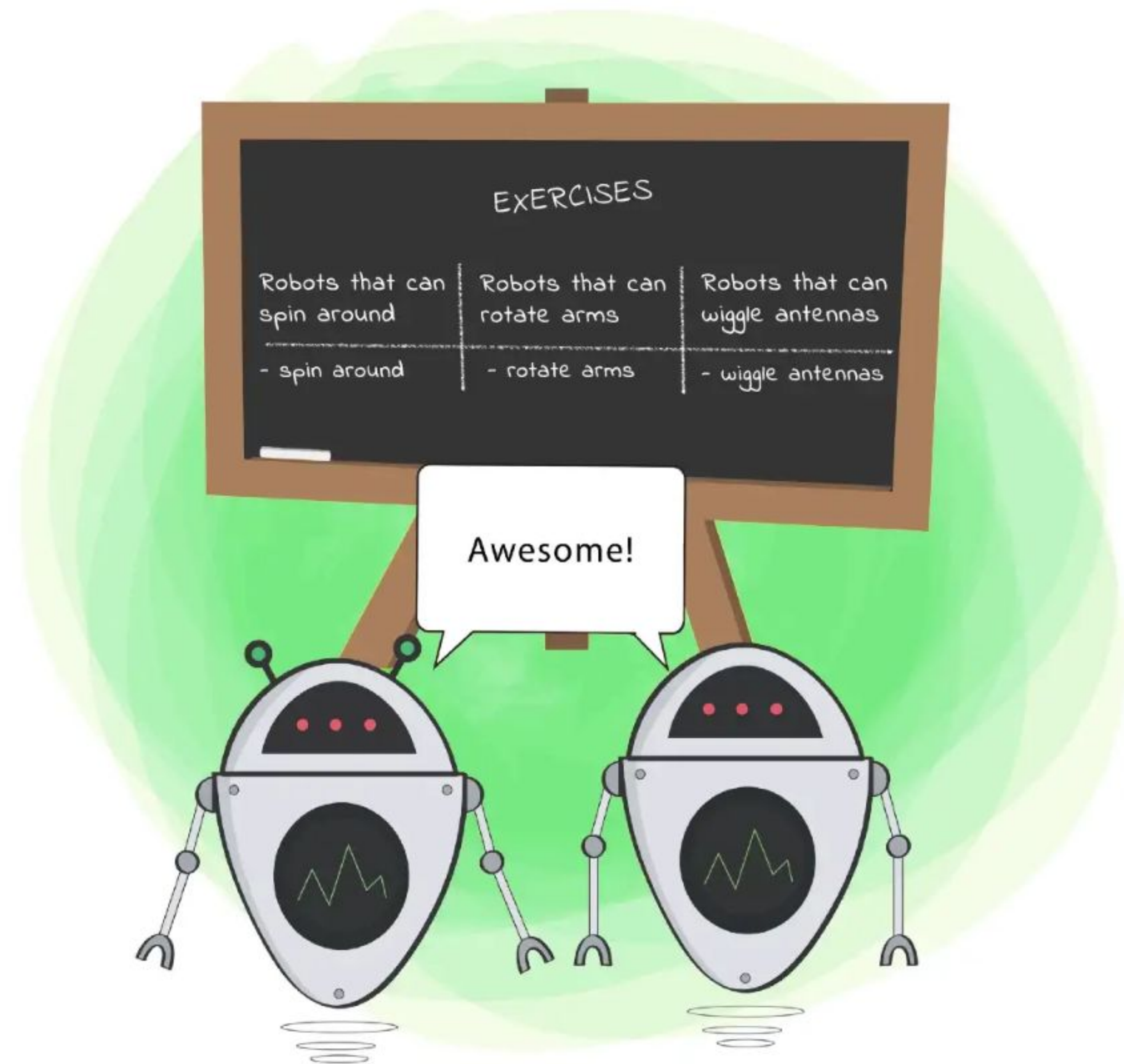


✓



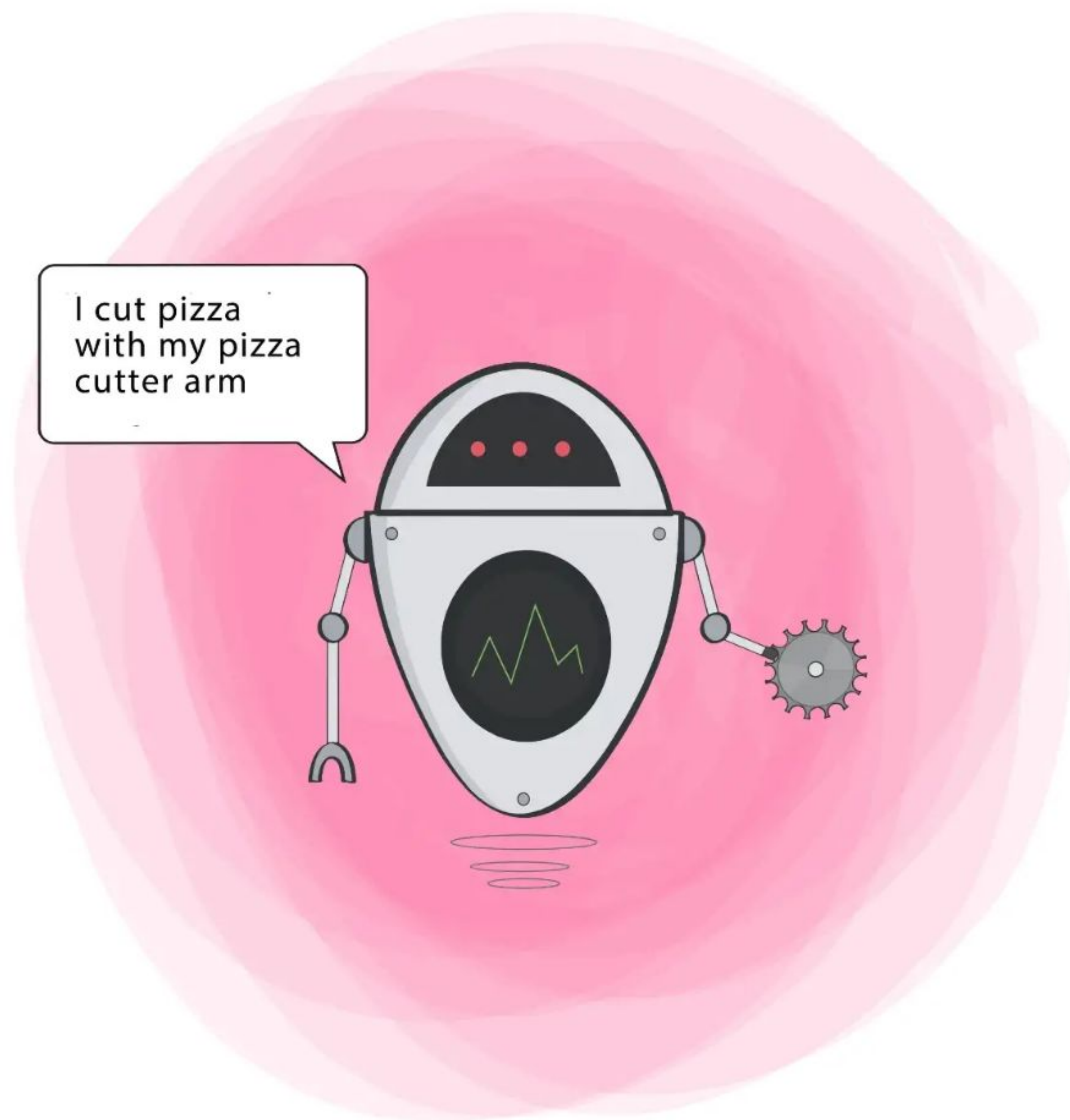


✗

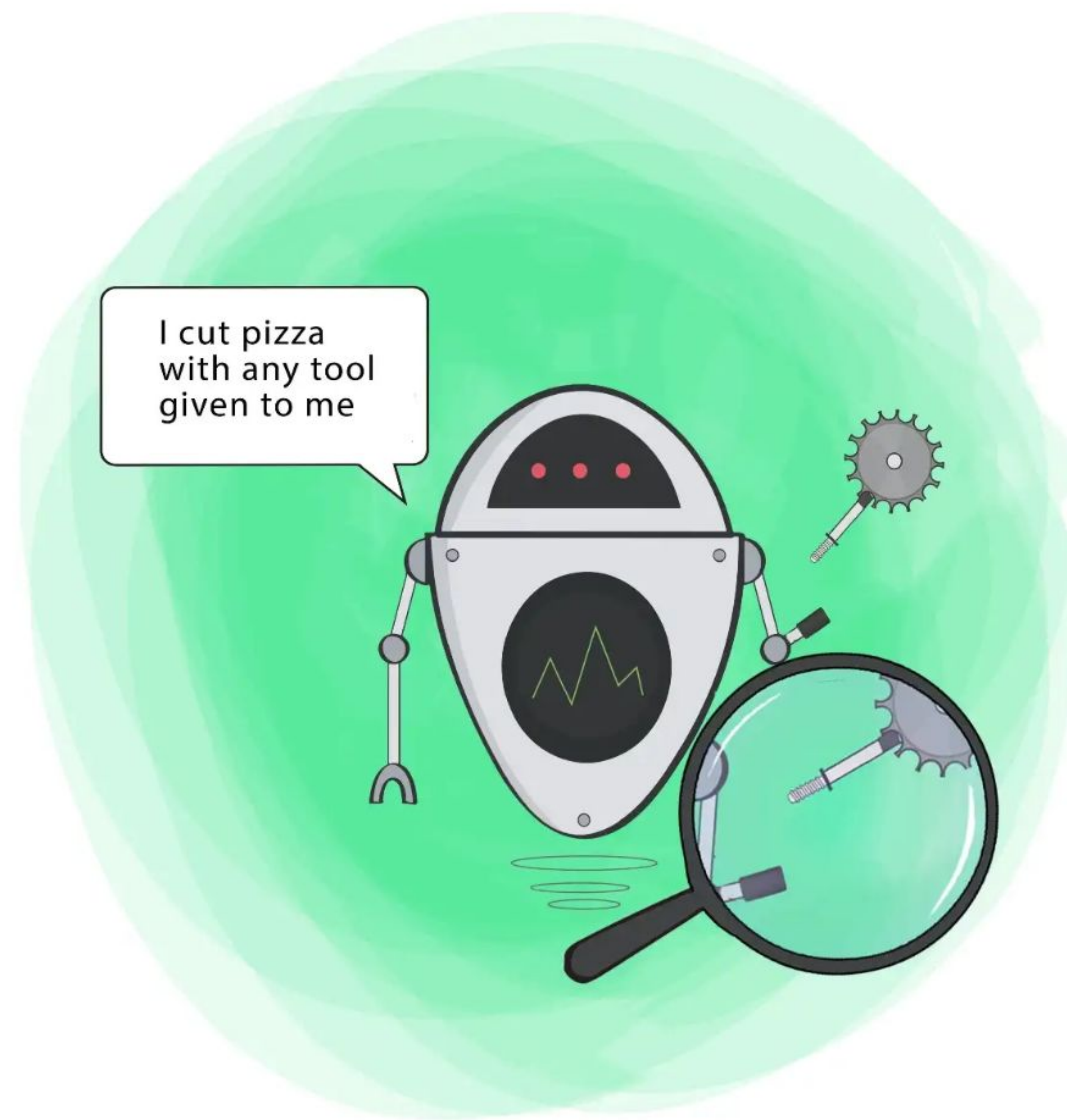


✓

Interface Segregation



✗



✓

Dependency Inversion



03 WHAT IS DESIGN PATTERNS

`<p> There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult. </p>`



CREATIONAL PATTERNS

```
<script> new Factory(); </script>
```

Creational design patterns provide various object creation mechanisms, which increase flexibility and reuse of existing code.



STRUCTURAL PATTERNS

```
<script> new Decorator(); </script>
```

Structural design patterns explain how to **assemble** objects and classes into **larger** structures, while keeping these structures **flexible** and efficient.



BEHAVIORAL PATTERNS

```
<script> new Observer(); </script>
```

Behavioral design patterns are concerned with algorithms and the assignment of responsibilities between objects.



04 WHAT IS DOMAIN DRIVEN DESIGN

<p> To communicate effectively, the code must be based on the same language used to write the requirements - the same language that the developers speak with each other and with domain experts. </p>

“

THE **HEART** OF **SOFTWARE** IS ITS
ABILITY TO SOLVE
DOMAIN-RELATED **PROBLEMS**
FOR ITS **USER**.

– Erick Evans



UBIQUITOUS LANGUAGE

```
<option value="user">Customer</option>
```

Is a methodology that refers to the same language domain experts and developers use when they talk about the domain they are working on.



BOUNDED CONTEXTS

```
<optgroup label="Context">...</optgroup>
```

Is a central pattern in domain-driven design that **contains** the **complexity** of the application. It **handles** large models and teams. This is where you **implement** the code, after you've **defined** the domain and the subdomains.



CONTEXT MAPPING

```
<map name="context">...</map>
```

Is a visual **representation** of the system's bounded contexts and integrations between them. This visual notation gives valuable strategic insight on multiple levels: **high-level** design, **communication** patterns, **organizational** issues

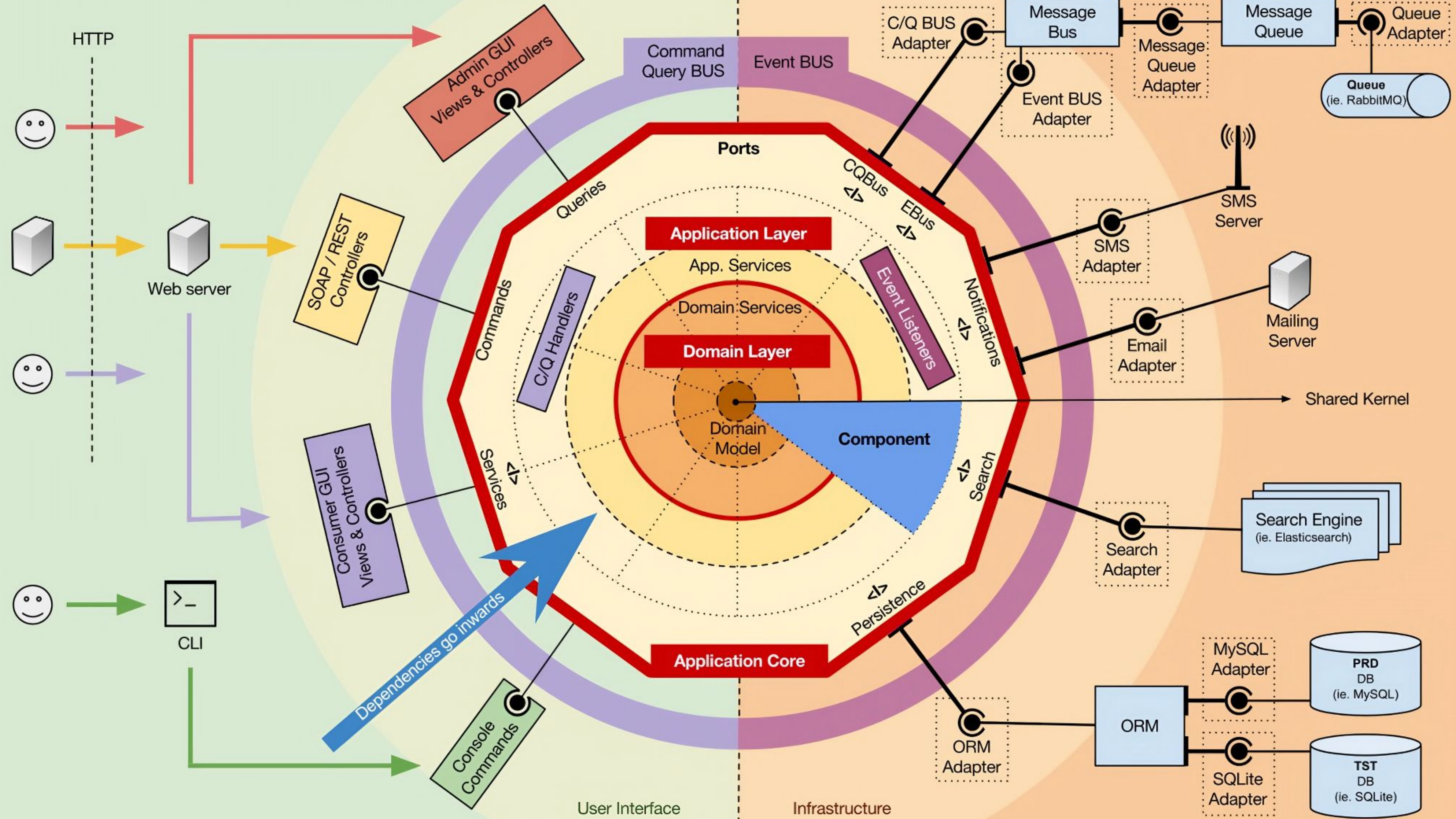


04 WHAT IS SOFTWARE ARCHITECTURE

<p> The component **structure** cannot be designed from the top down. It is not one of the first things about the system that is **designed**, but rather **evolves** as the system **grows** and changes. </p>

Primary/Driving Adapters

Secondary/Driven Adapters

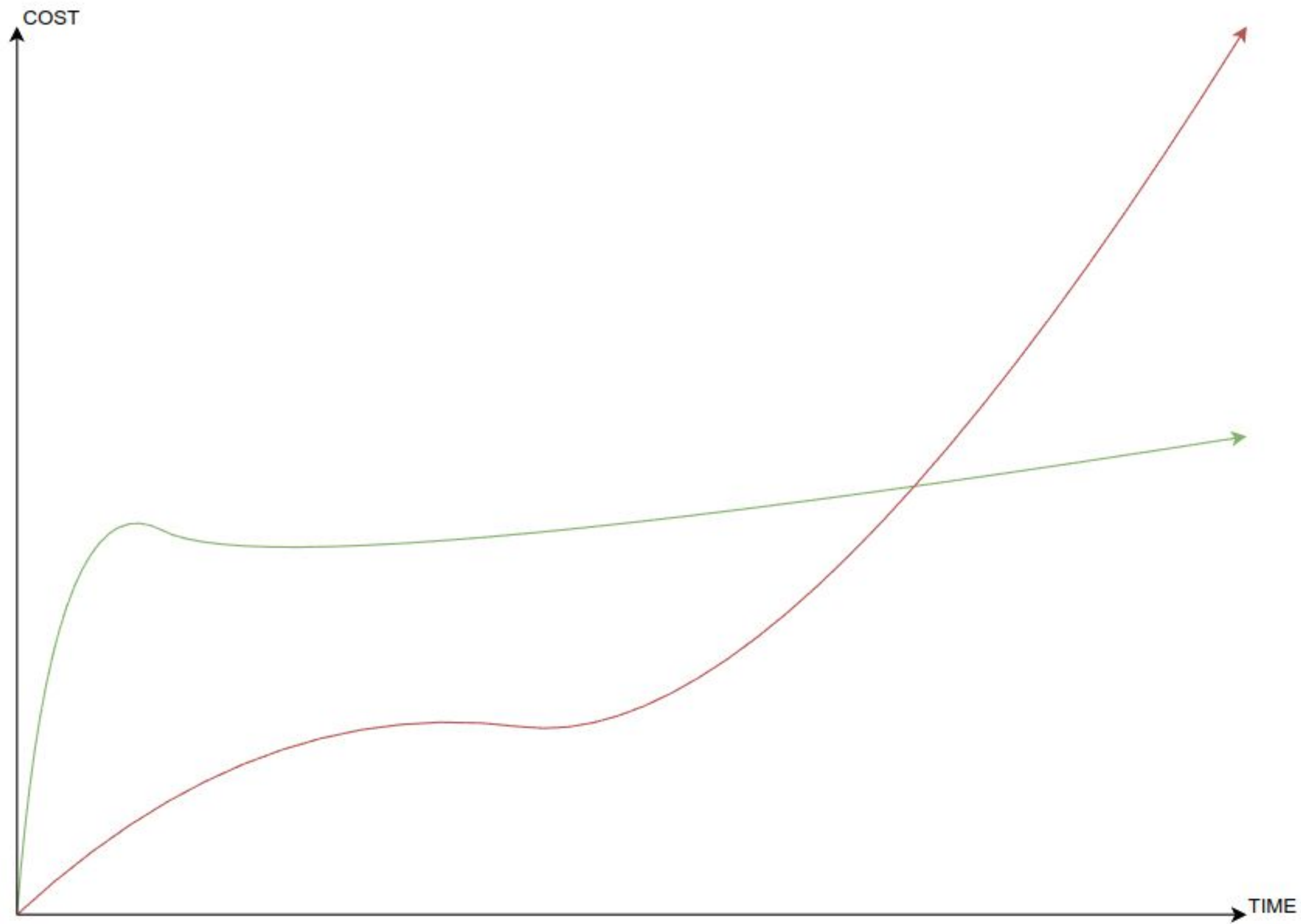


Understand all of this,
but use only what you need



05 LETS SEE SOME CHARTS

`<p> A child asked a programmer why the sun rises in the east,
and sets in the west. His response? It works, don't touch! </p>`






WHAT CAN I READ NOW?

- Object-Oriented Thought Process
- Adaptive Code: Agile coding with design patterns and SOLID principles
- Domain-Driven Design: Tackling Complexity in the Heart of Software
- Clean Code: A Handbook of Agile Software Craftsmanship
- Clean Architecture: A Craftsman's Guide to Software Structure and Design
- Design Patterns: Elements of Reusable Object-Oriented Software



THANK YOU!

Do you have any questions?



```
toniette.github.io  
linkedin.com/in/toniette  
calendly.com/toniette/call
```