Abstraction Barriers!

How do you manage complexity in systems?

- In software engineering, complexity is often the result of *excessive interdependency* and *exposure to underlying implementation details*.

- Fred Brooks' *Mythical Man Month* identifies two kinds of complexity:

1. **Essential Complexity** - inherent in the problem you are solving

2. **Accidental Complexity** - complexity caused by underlying systems

- As software engineers and *designers*, **good, simple abstractions** help you *minimize* accidental complexity and *hide implementation details*.

Abstraction Barriers

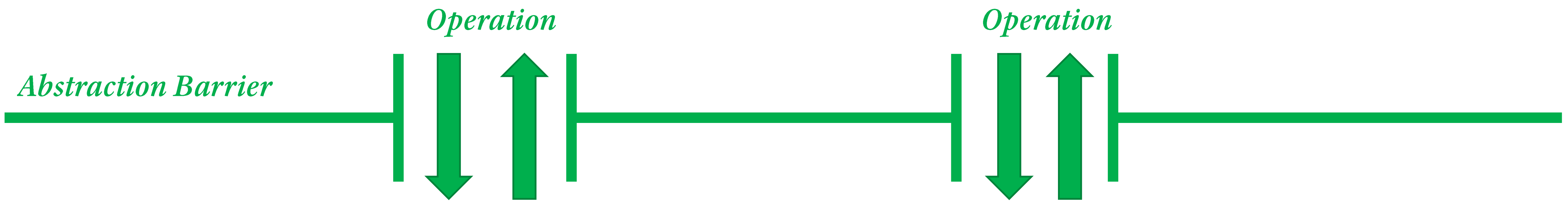
- "**Separates** the **use** of an **abstraction** from the **implementation** of it"
  - *Structure and Interpretation of Computer Programs* by Abelson and Sussman

- This is a design concept you'll see echoed in many forms:
  - Information Hiding
  - Encapsulation
  - Least Privilege
  - Design by Contract

- Good abstraction barriers raise ***essential complexity*** to an **interface** and hide away ***accidental complexity*** beneath it

Usages of the Abstraction

- Expressed and implemented *in terms of* the operations the abstraction provides
- Implementation details beneath the barrier should be irrelevant
  - If a user of the abstraction must "reach across" the barrier to concern itself with details beyond what the operations provide, this is called a "leaky abstraction" and a design problem.



Implementation Details

- Algorithms and data structures of an abstraction.
- The implementation owner can make changes and improvements if the operations exposed at the abstraction barrier maintain their "contracts".

How are barriers of abstraction realized?

- Generally language specific, but a consistent theme:
  - The abstraction is defined in terms of its *operations* and *axioms*.
  - In programming languages, operations are functions/methods.

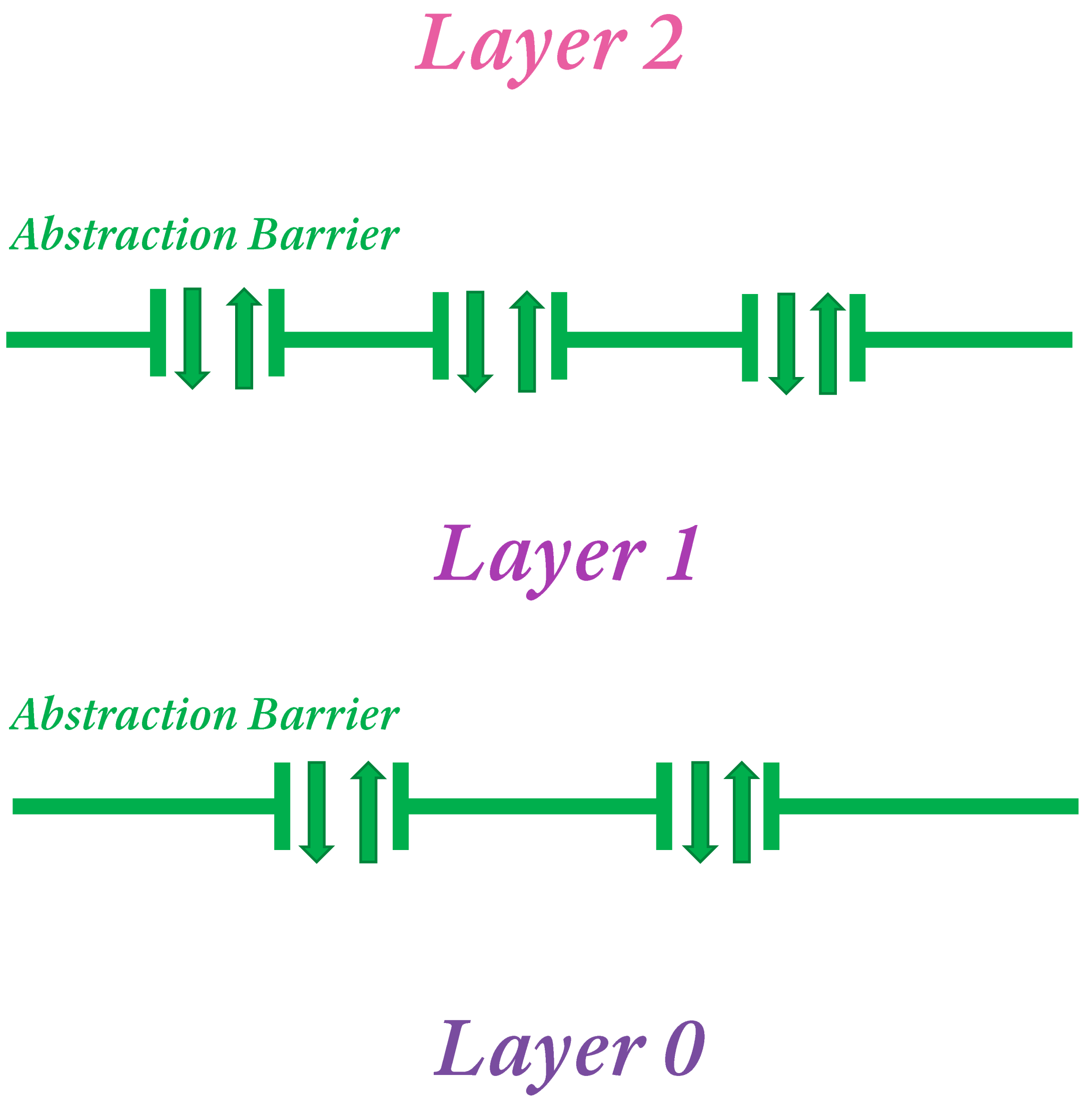
- C: Function Declarations in Header Files
- Java: Interfaces (Public Methods)
- Rust: Traits

- Design Goal: Minimize # operations while maximizing # of use cases
  - These two goals are often at odds with one another. Trade-offs galore!
  - Getting this right in novel abstractions requires iteration and experimentation.

Layered Architectures

- When a barrier of abstraction is *robust,* it becomes a building block
    - Higher level barriers of abstraction are built in terms of lower levels!

- Layered architectures in software engineering enable complex systems to be built with **higher confidence** and **productivity**
    - Lower layers are tested in isolation from higher layers and provide a sturdier foundation than trying to build a complex system ad-hoc.



Layer 2
Abstraction Barrier
Layer 1
Abstraction Barrier
Layer 0