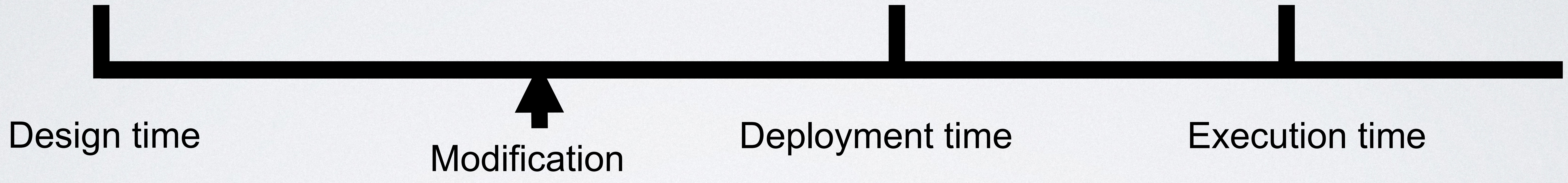
Focus: Modifiability

Goal: identify tactics that can improve modifiability

Source: https://resources.sei.cmu.edu/asset_files/TechnicalReport/2007_005_001_14858.pdf (Bachmann, Bass, Nord)

When Will the Change Occur?



Responsibilities

- A responsibility is an action, knowledge to be maintained, or a decision to be carried out by a software system or an element of that system. [Bachmann, Bass, Nord]
- Responsibilities are assigned to modules
- But what is the cost of modifying a responsibility?
- Responsibilities can be coupled: a modification to one can result in a modification to the other

Coupling

- Cost of modifying module A depends on how tightly-coupled it is to other modules
- Idea: reducing coupling may reduce modification costs
- To reduce coupling:
    - Minimize relationships among elements *not* in the same module
    - Maximize relationships among elements in the same module

Cohesion

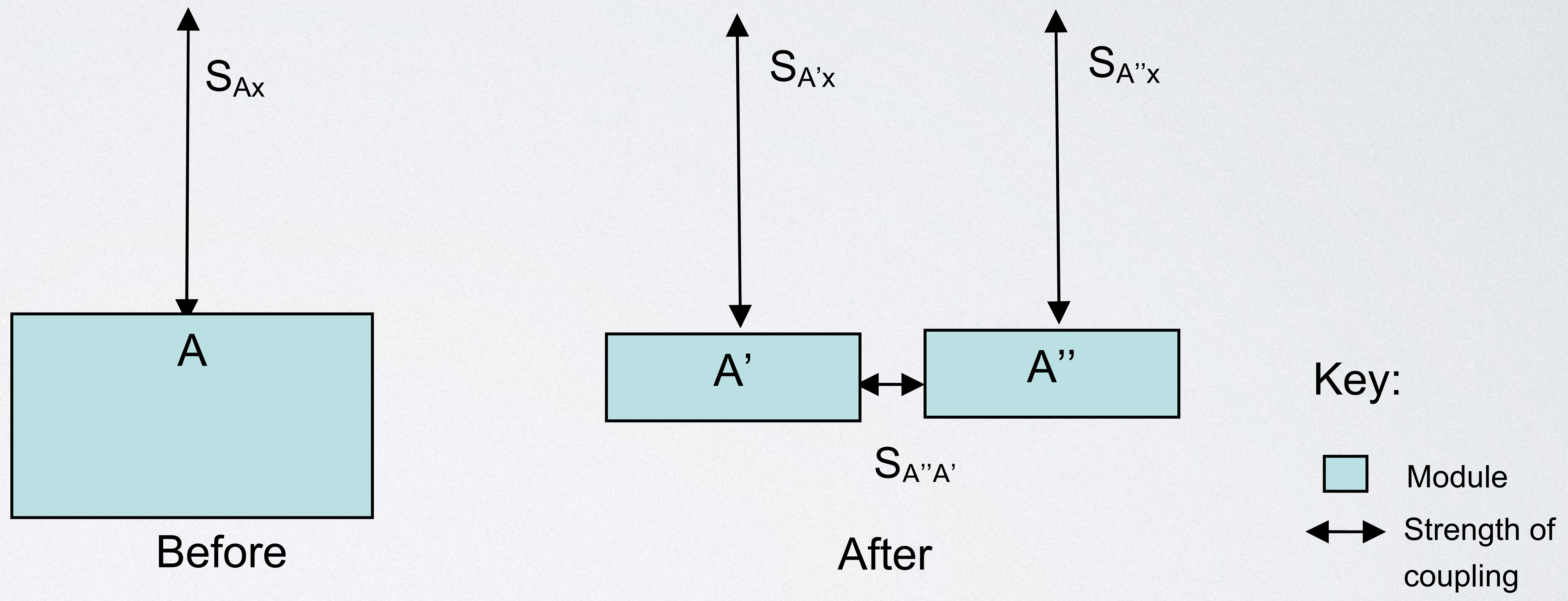
- Put related responsibilities in the same module
- To maximize modifiability, maximize cohesion & minimize coupling

Tactics

- Reducing the cost of modifying a single responsibility
  - Split a Responsibility.
- Increasing cohesion
  - Maintain Semantic Coherence.
  - Abstract Common Services.
- Reducing coupling
  - Use Encapsulation.
  - Use a Wrapper.
  - Raise the Abstraction Level.
  - Use an Intermediary.
  - Restrict Communication Paths.

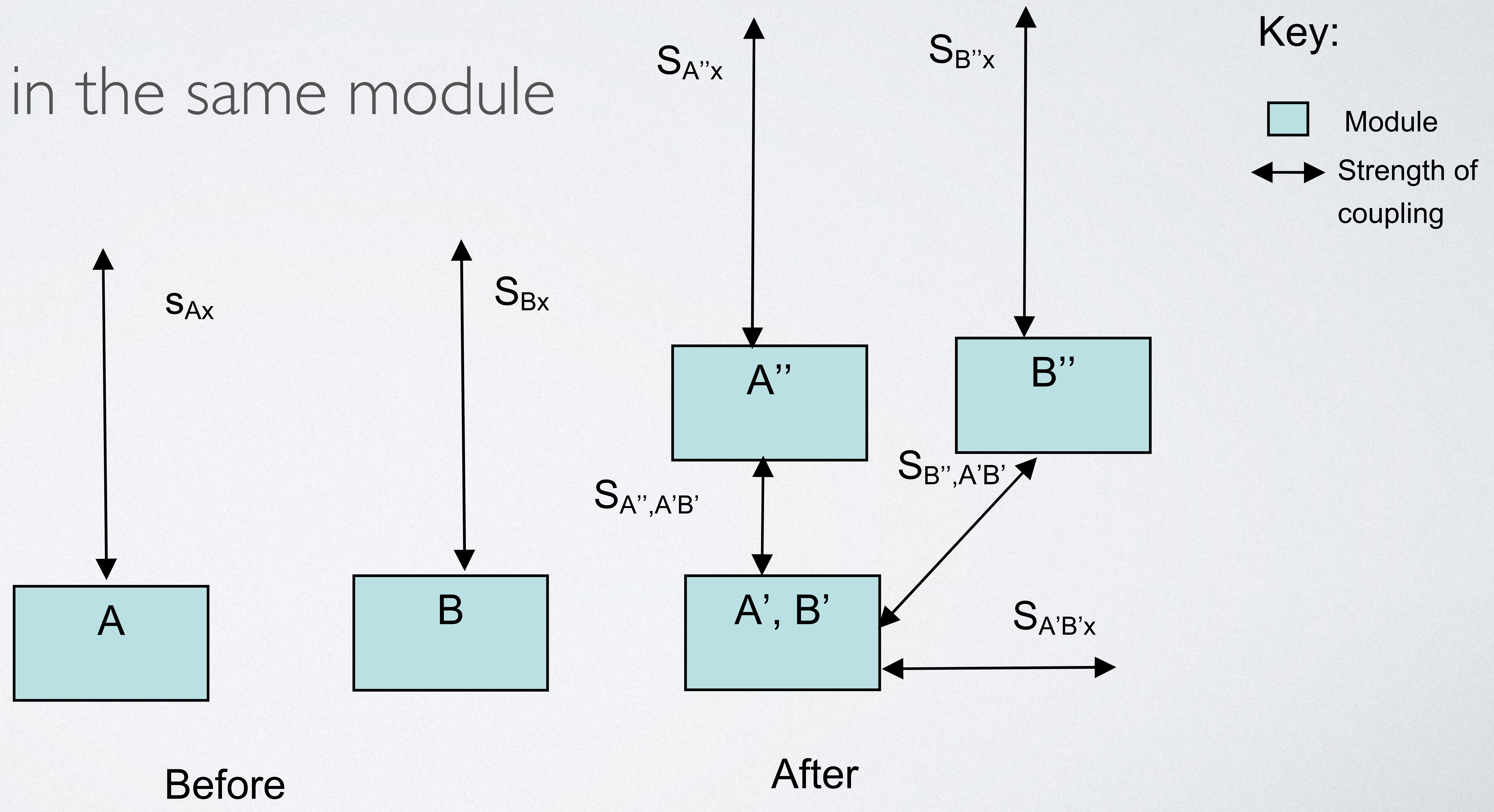
Tactic 1: Split a Responsibility

- Goal: split so the new modules can be modified independently
- Also: enables deferred binding (replace module A'' at runtime)



Tactic 2: Increase Cohesion

- Idea: move responsibilities from one module to another
- Approach: put A' and B' in the same module

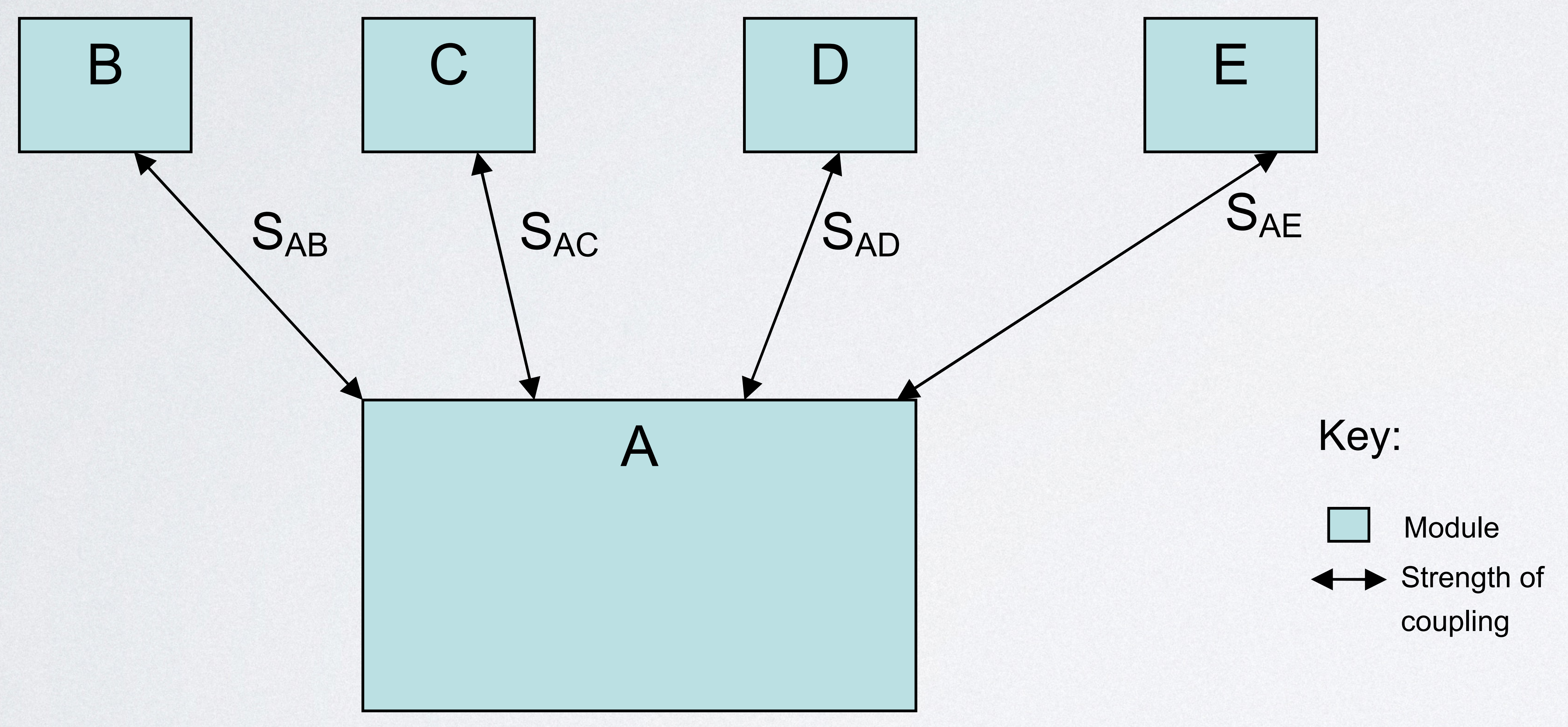


But: How Do We Split a Module?

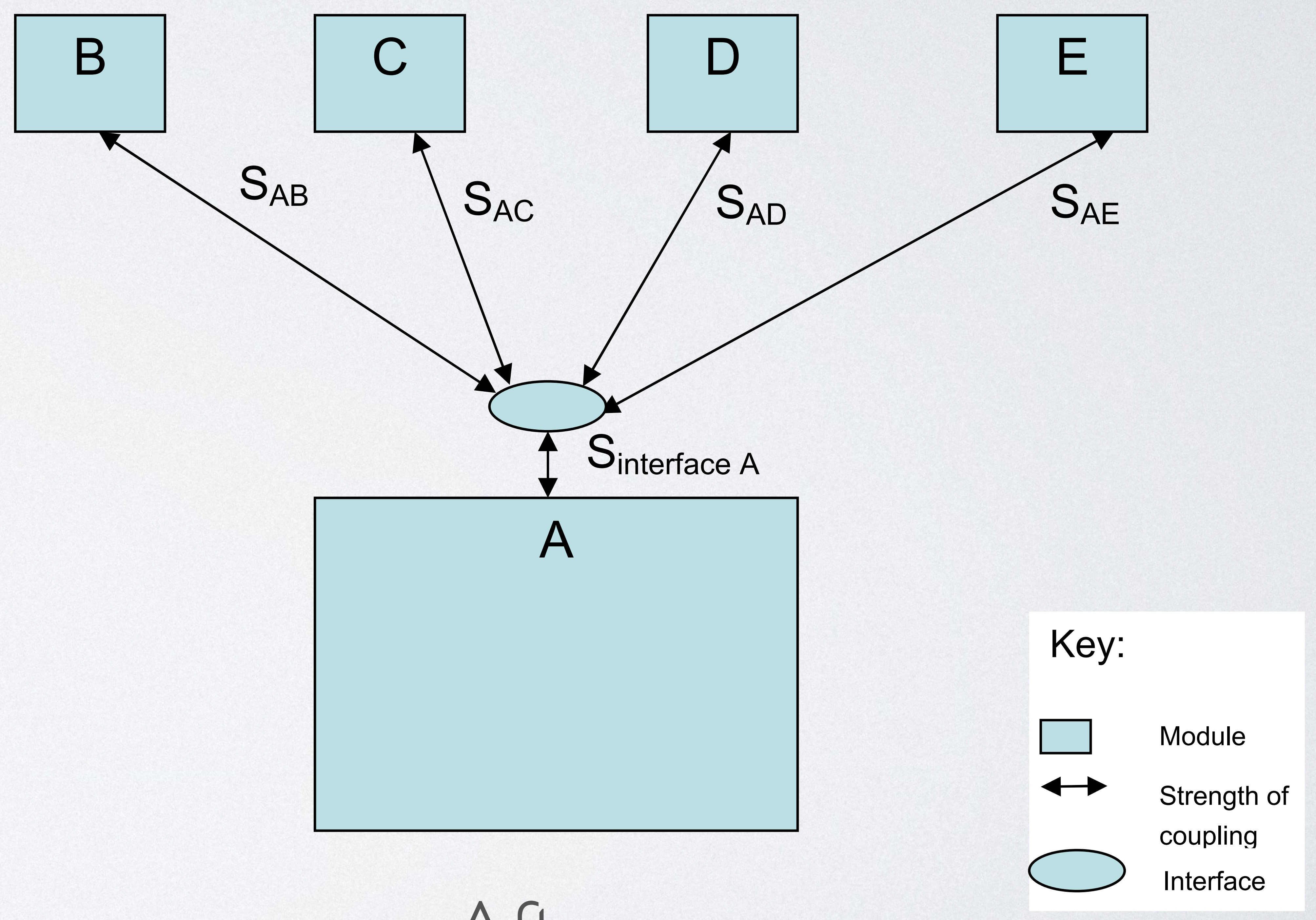
- 2.1: maintain semantic coherence (A', B' may need to change in the future)
- 2.2: abstract common services (A', B' represent similar services)

Tactic 3: Reduce Coupling

- 3.1: Use encapsulation (hide information in A)



Before



After

Add a Wrapper

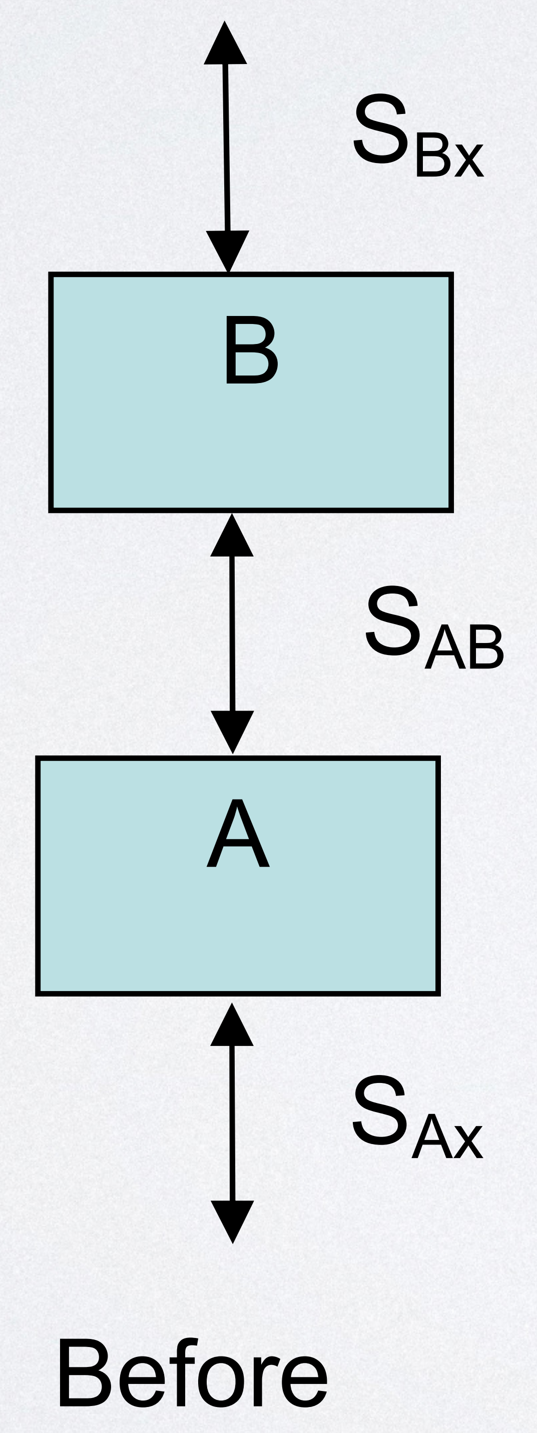
- Encapsulation hides information
- Wrappers transform invocations
  - (yes, the boundary is fuzzy)

Raise the Abstraction Level

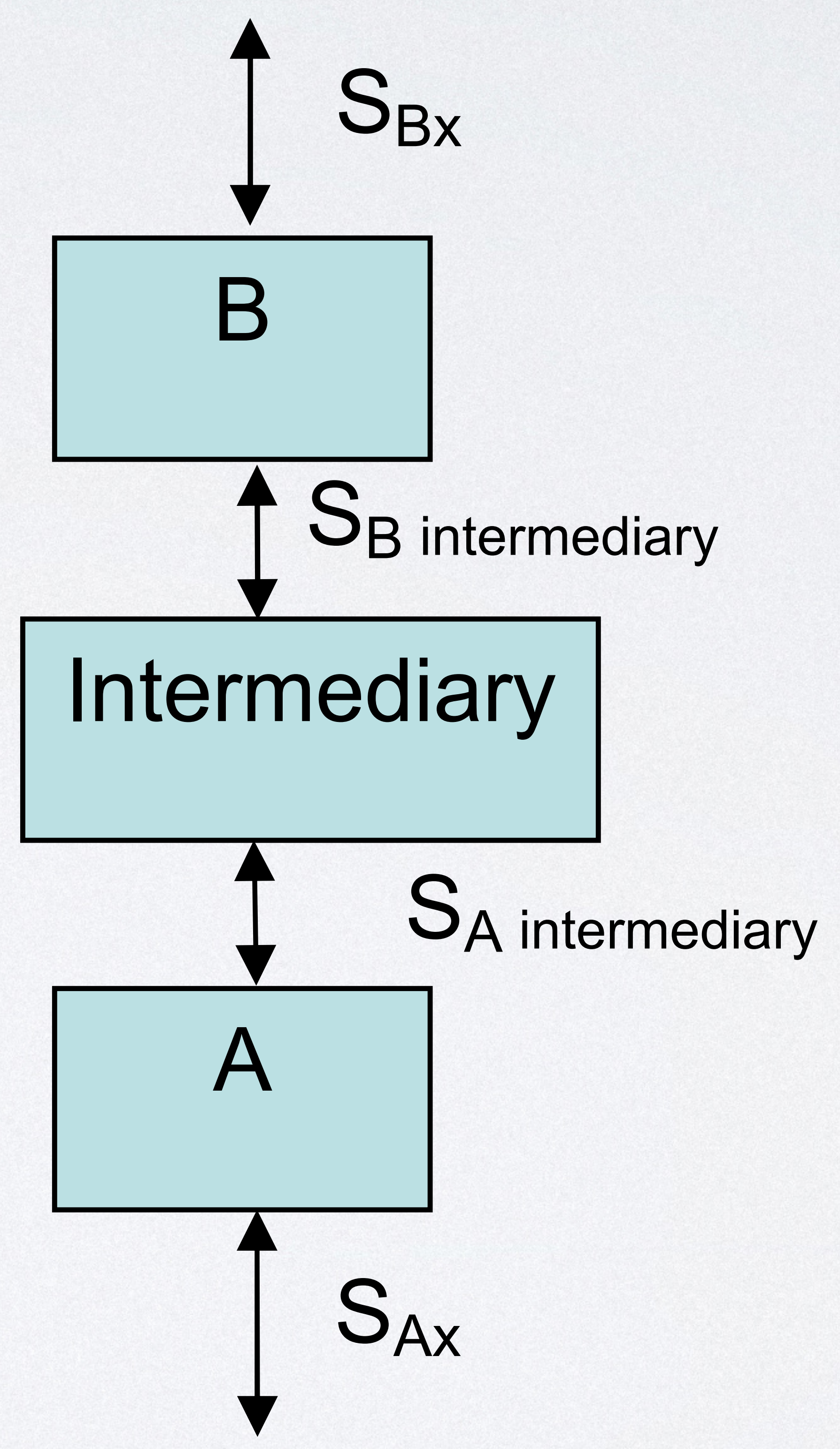
- Usually: add parameters to interface
  - Makes the module more abstract, enables flexibility

Use an Intermediary, Restrict Communication Paths

- Break dependency (but add a new one instead)



Before



After

Key:

▭	Module
↔	Strength of coupling

Architecture Speed Dating!

Or: Instamatic Architecture Micro-Reviews

Focus: Modifiability

Problem: ATMs

- Design an architecture for ATMs. Features:
  - Each bank has remote ATMs and a central server.
  - Users should be able to withdraw cash from their accounts.
- Modification scenario 1: bank merger.
- Modification scenario 2: support two-factor authentication for withdrawals.
- Modification scenario 3: support depositing paper checks.