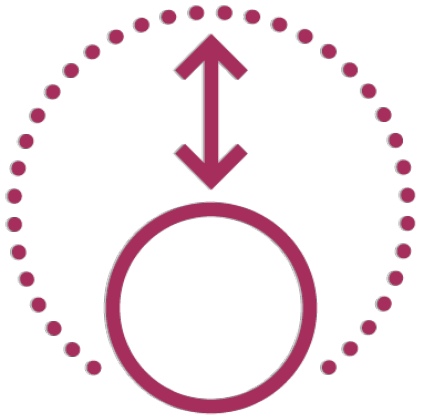
Creating Interfaces to Add Extensibility

Why Interfaces?

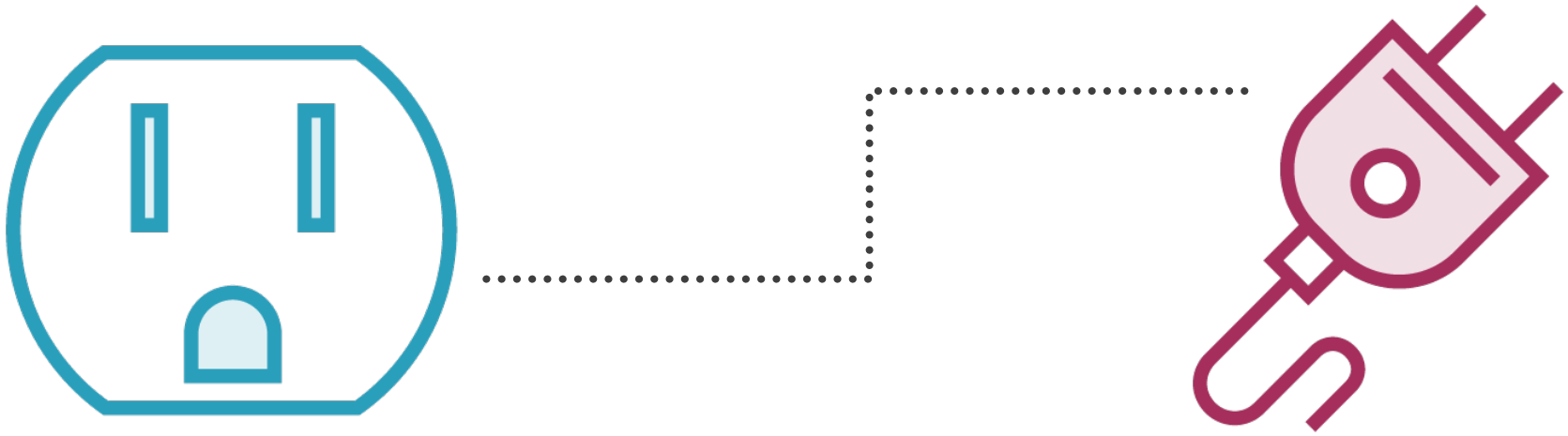
Maintainable



Extensible

Easily testable

Extensible



Different Data Sources

Relational Databases

- Microsoft SQL Server, Oracle, MySQL, etc.

Document / Object Databases (NoSQL)

- MongoDB, Hadoop, RavenDB, etc.

Text Files

- CSV, XML, JSON, etc.

SOAP Services

- WCF, ASMX Web Service, Apache CXF, etc.

REST Services

- WebAPI, WCF, Apache CXF, JAX-RS, etc.

Cloud Storage

- Microsoft Azure, Amazon AWS, Google Cloud SQL

Repository Pattern

Mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects.

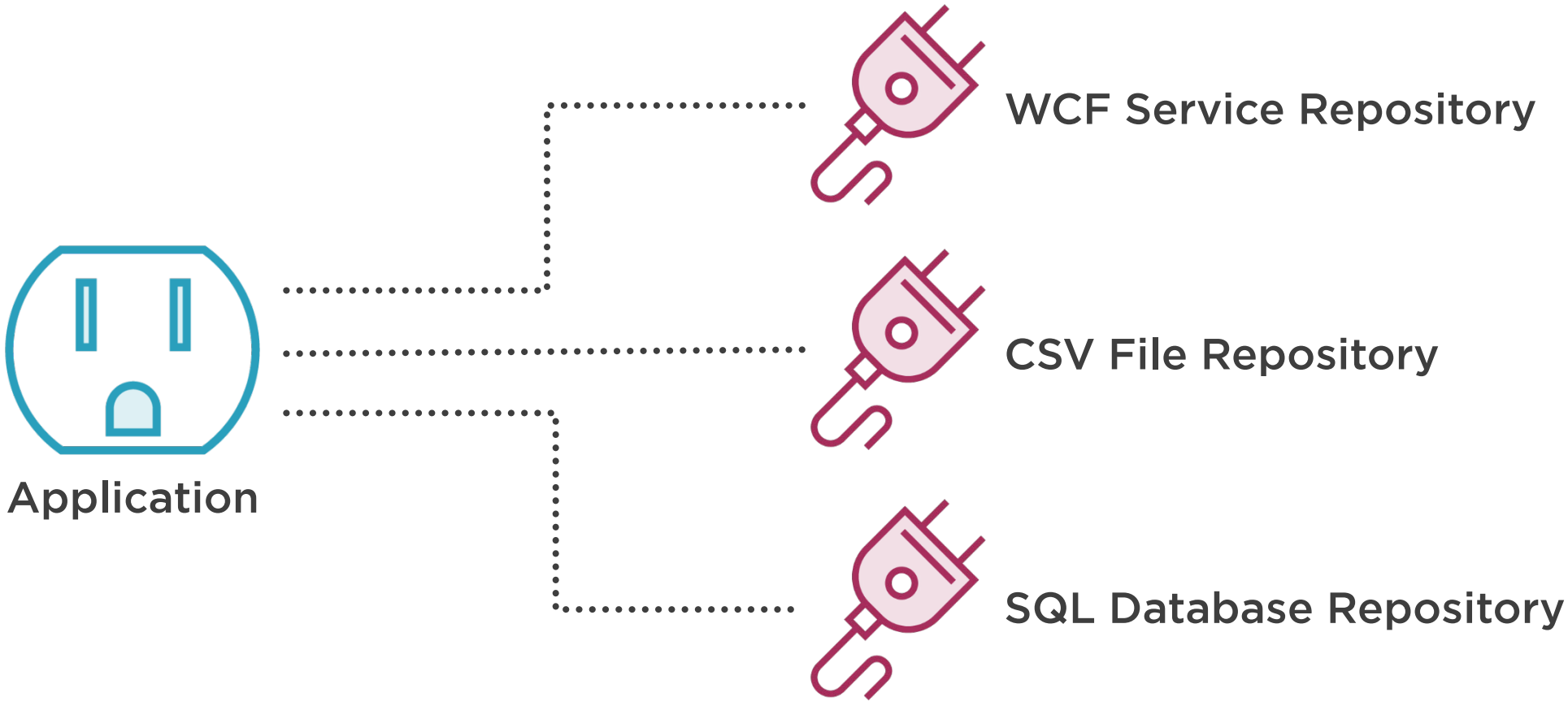
– **Fowler, et al. Patterns of Enterprise Application Architecture. Addison-Wesley, 2003**

Repository Pattern

Layer to separate our application from the data storage technology

- Application
- Repository
- Data Storage

Pluggable Repositories



Simple Repository

Data Access Operations

- **C** Create
- **R** Read
- **U** Update
- **D** Delete

Creating a Repository Interface

```
public interface IPersonRepository
{
    void AddPerson(Person newPerson);

    IEnumerable<Person> GetPeople();

    Person GetPerson(string lastName);

    void UpdatePerson(string lastName,
           Person updatedPerson);

    void UpdatePeople(IEnumerable<Person>
           updatedPeople);

    void DeletePerson(string lastName);
}
```

C

R

U

D

Summary

Repository Pattern

- Create
- Read
- Update
- Delete

How to Create and Implement a Custom Interface

- IPerson Repository

Easy Extensibility

UP NEXT:

Explicit Interface Implementation