الجامعة السورية الخاصة
SYRIAN PRIVATE UNIVERSITY

مقرر بنيان البرمجيات | كلية الهندسة المعلوماتية | المحاضرة الثالثة

General Principles in Software Design and Architecture 2
"System Design Concepts" - "What is Good Design?"

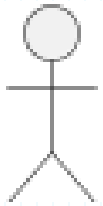
د. رياض سنبل

https://rsonbol.github.io/courses/courses-sa-spu/

Why We Want To Decompose Systems

The system is designed to scrape social media data, perform in-depth analysis, and predict specific product limitations based on user reviews. It then presents the findings through comprehensive reports and charts, effectively illustrating the results. Users can access their accounts, enter their products info (including their related social media pages) and use various functionalities either from a mobile app or a website.

Why We Want To Decompose Systems



Mobile App

Web pages

?

Main Business Features and services

?

social media data scraper

Analyze data and predict limitations

User Mgmt.

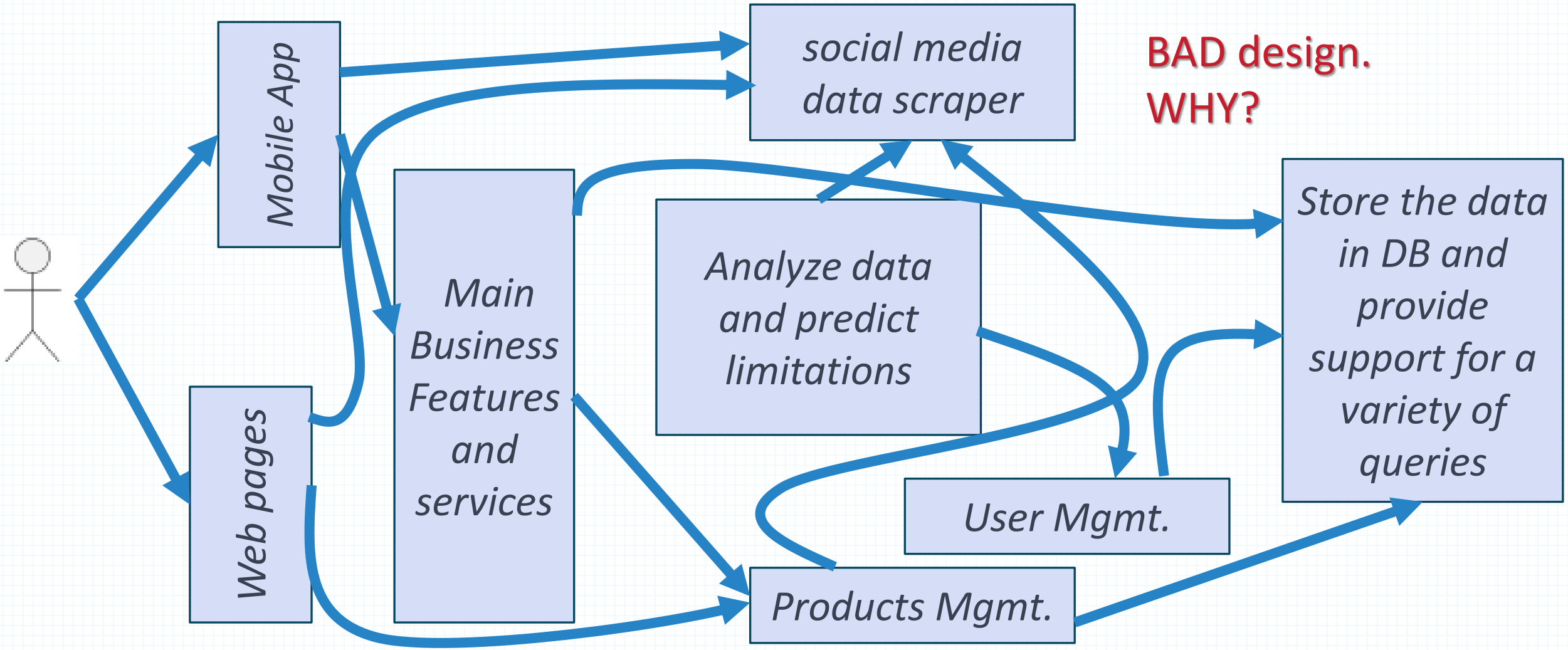
Products Mgmt.

?

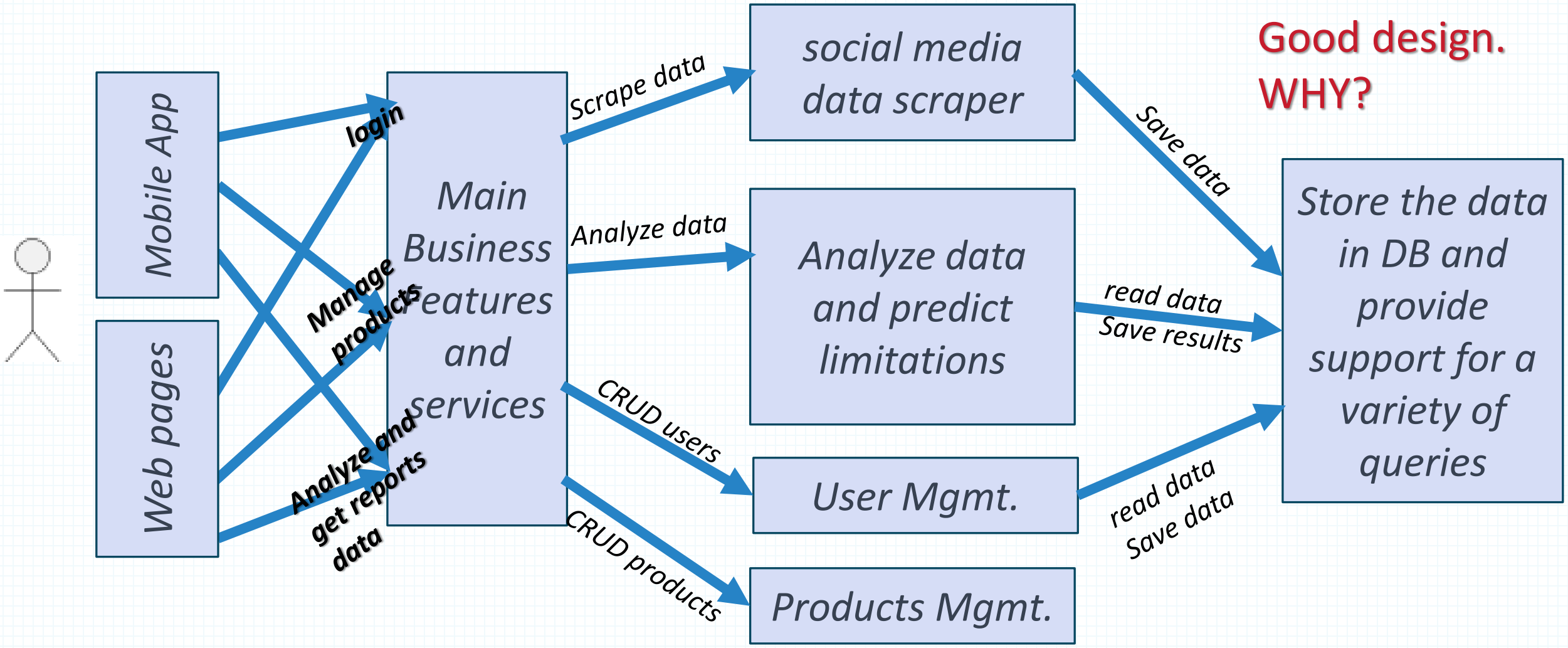
Store the data in DB and provide support for a variety of queries

But How can we connect these elements?

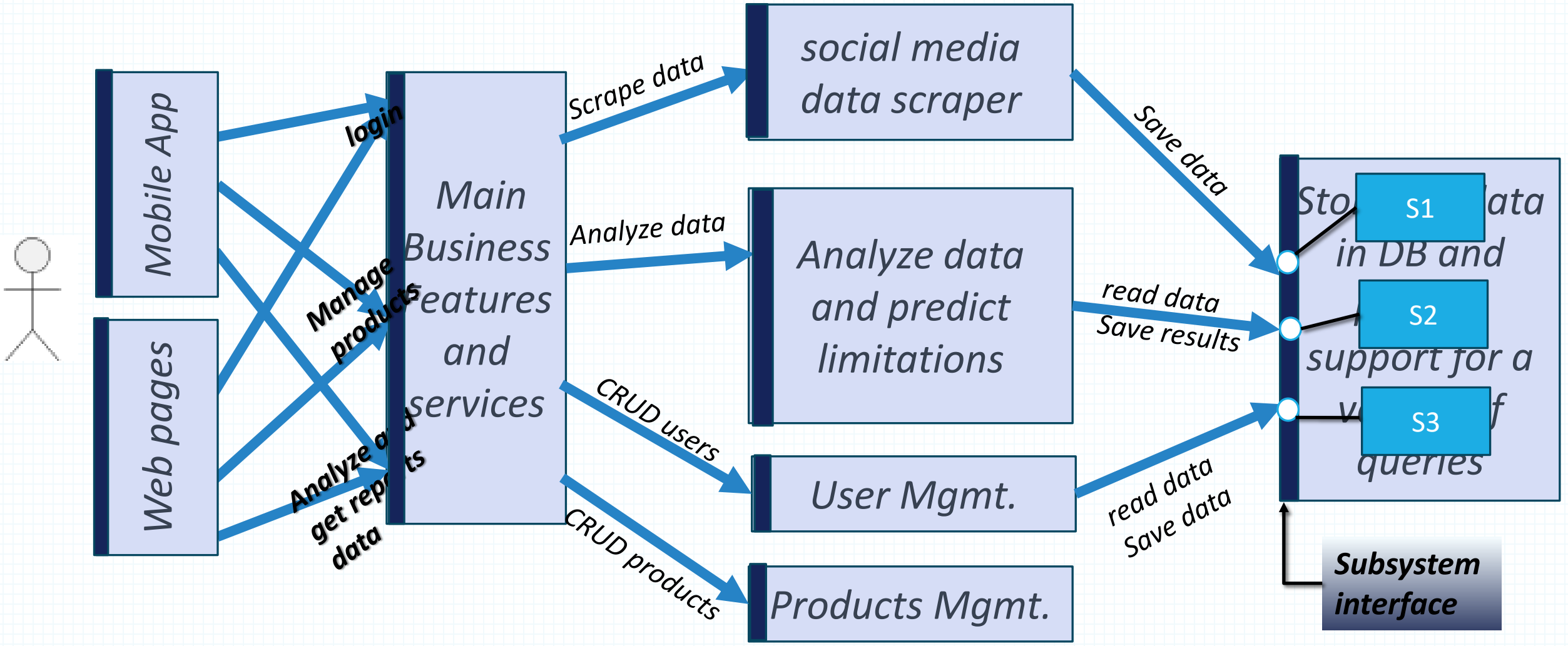
Why We Want To Decompose Systems



Why We Want To Decompose Systems



Why We Want To Decompose Systems



System Design Concepts

- A **subsystem** is a replaceable part of the system with well-defined interfaces that encapsulates the *state* and *behavior* of its contained classes.
- A **service** is a set of named operations that share a common purpose. The "seed" for services are the use cases from the functional model.
- **Subsystem interface** specifies the interaction and information flow from and to subsystem boundaries, but not inside the subsystem.
  - defined during design stage.
- **Application programmer's interface (API)** is the specification of the subsystem interface in a specific programming language
  - defined during implementation stage.

Taxonomy

Product line (or product family)

System or product

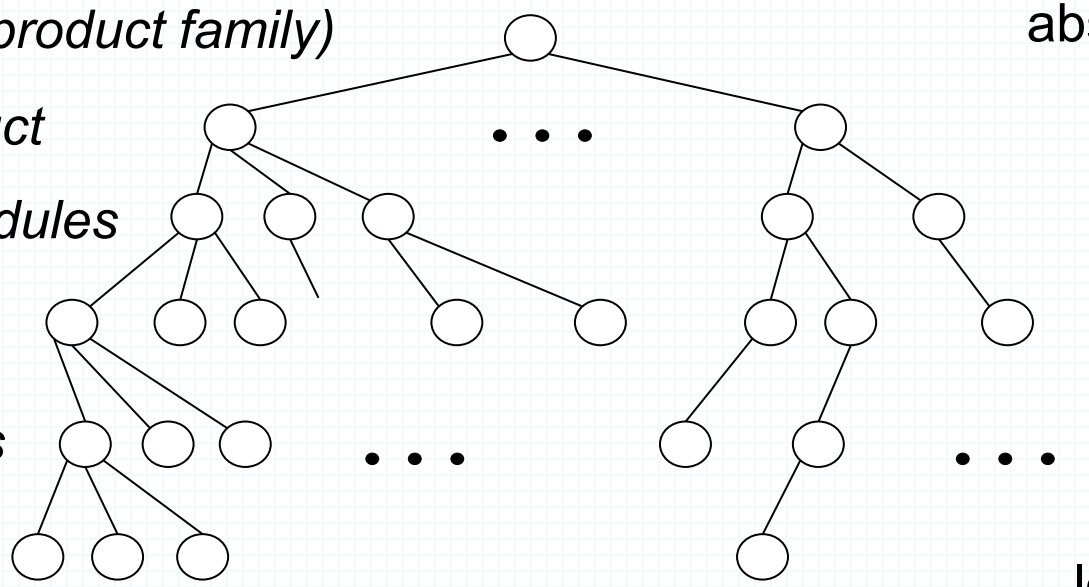
Subsystems/Modules

Packages

Classes/Objects

Methods

Source code



highest abstraction level

lowest level

Layers and Partitions

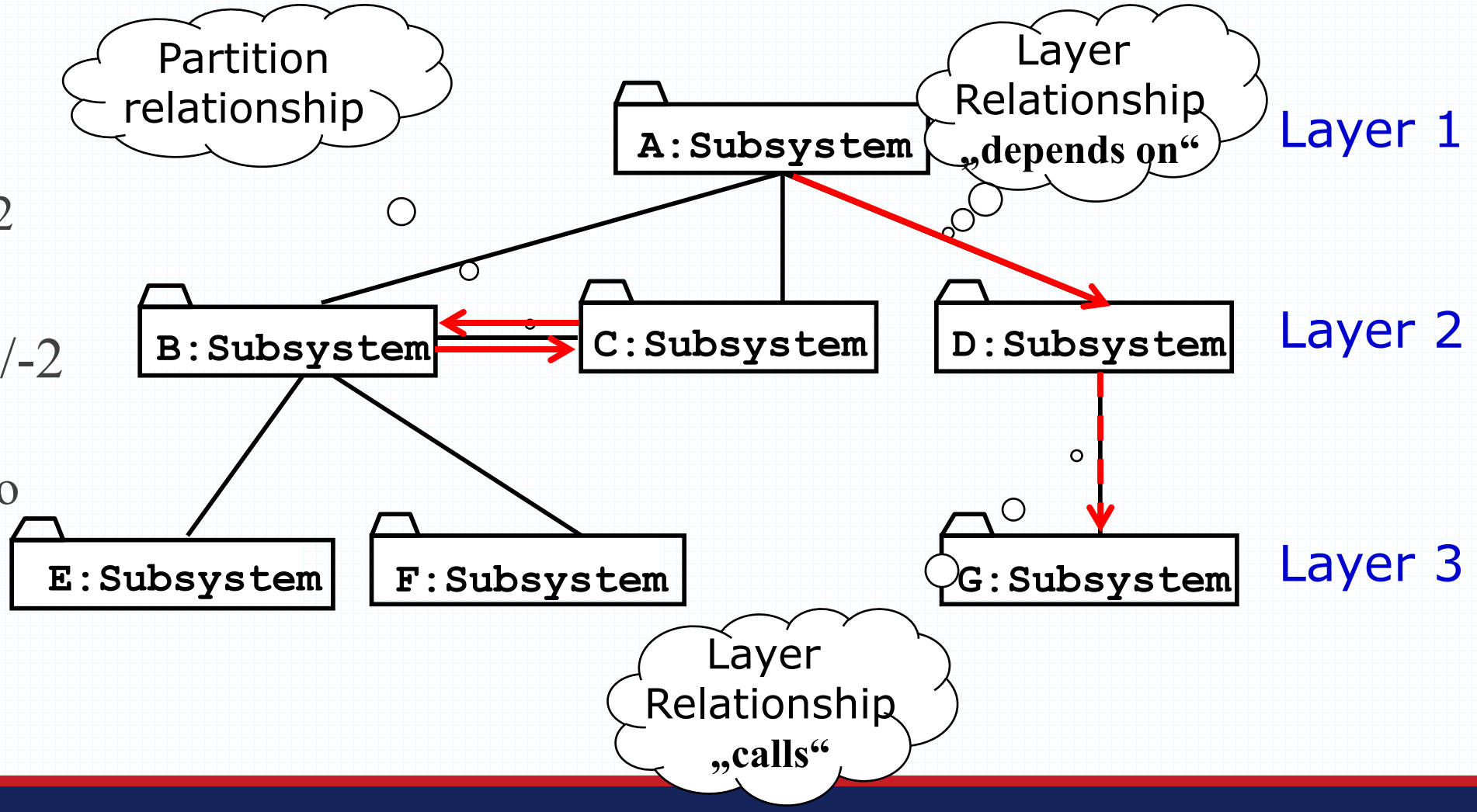
- A layer is a subsystem that provides a service to another subsystem with the following restrictions:
  - A layer only depends on services from lower layers
  - A layer has no knowledge of higher layers
- A layer can be divided horizontally into several independent subsystems called partitions
  - Partitions provide services to other partitions on the same layer
  - Partitions are also called “weakly coupled” subsystems.

Relationships between Subsystems

- Two major types of Layer relationships
  - Layer A "depends on" Layer B (compile time dependency)
    - Example: Build dependencies (make, ant, maven)
  - Layer A "calls" Layer B (runtime dependency)
    - Example: A web browser calls a web server
    - Can the client and server layers run on the same machine?
      - Yes, they are layers, not processor nodes
- Partition relationship
  - The subsystems have mutual knowledge about each other
    - A calls services in B; B calls services in A (Peer-to-Peer)

Example of a Subsystem Decomposition

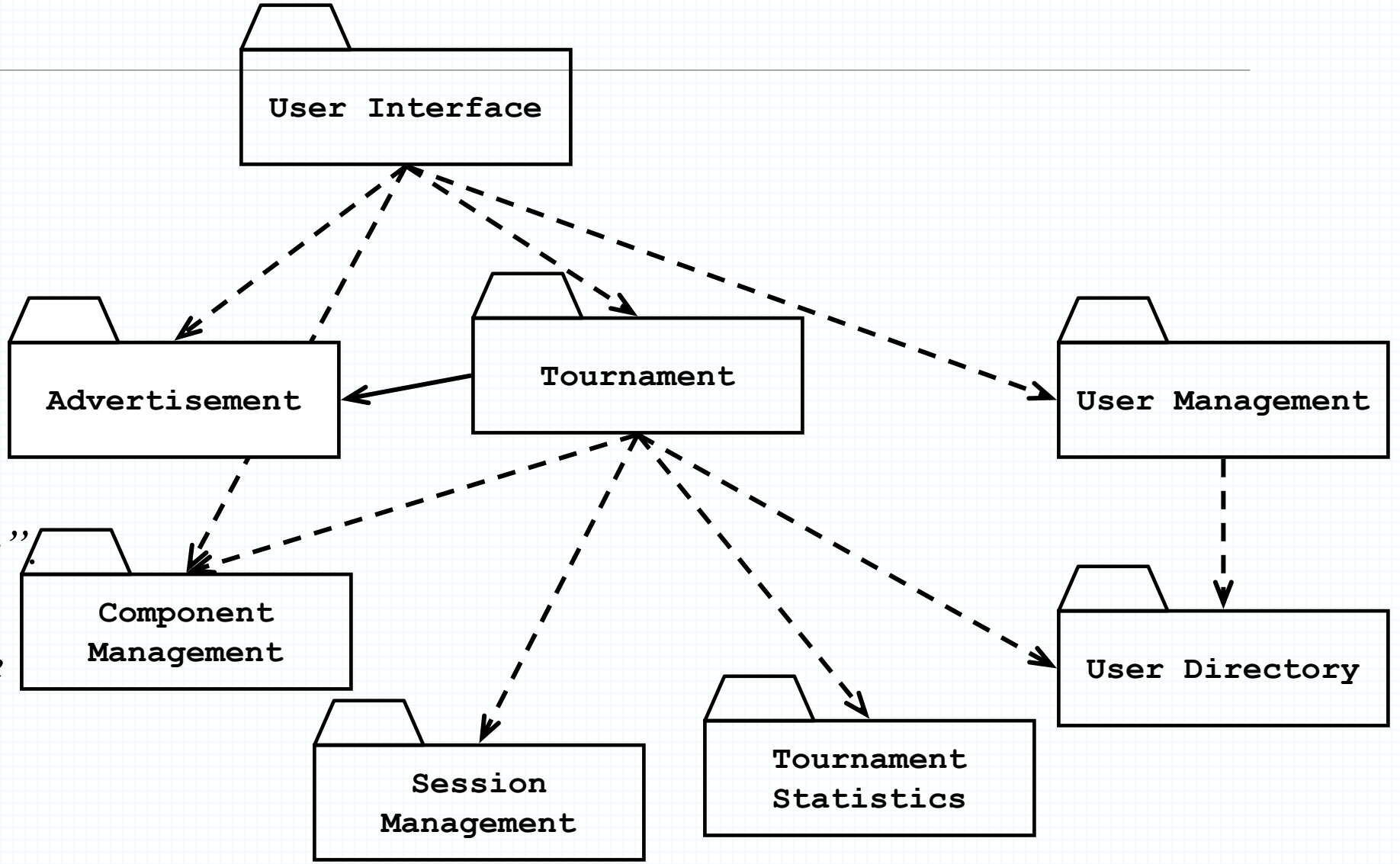
- Subsystem Decomposition Heuristics
  - No more than 7+/-2 subsystems
- No more than 4+/-2 layers
  - Good design: Try to find 3 layers!



Example

New type of association:

- *The dashed line means "depends on at runtime".*
- *The solid line means "depends on at compile time".*



So..
What is Good Design?

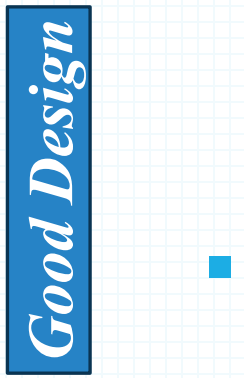
Separation of Concerns (SoC)

- Separation of Concerns (SoC) is a design principle that manages complexity by partitioning the software system so that each partition is responsible for a separate concern.

minimizing the overlap of concerns as much as possible

Coupling and Coherence of Subsystems

- Goal: Reduce system complexity while allowing change
- **Coherence** measures dependency among classes
  - High coherence: The classes in the subsystem perform similar tasks and are related to each other via many associations
  - Low coherence: Lots of miscellaneous and auxiliary classes, almost no associations



- **Coupling** measures dependency among subsystems
  - High coupling: Changes to one subsystem will have high impact on the other subsystem
  - Low coupling: A change in one subsystem does not affect any other subsystem.

Coupling and Coherence of Subsystems

- Goal: Reduce system complexity while allowing change
- **Coherence** measures dependency among classes
  - High coherence can be achieved if most of the interaction is within subsystems, rather than across subsystem boundaries

Good Design

associations

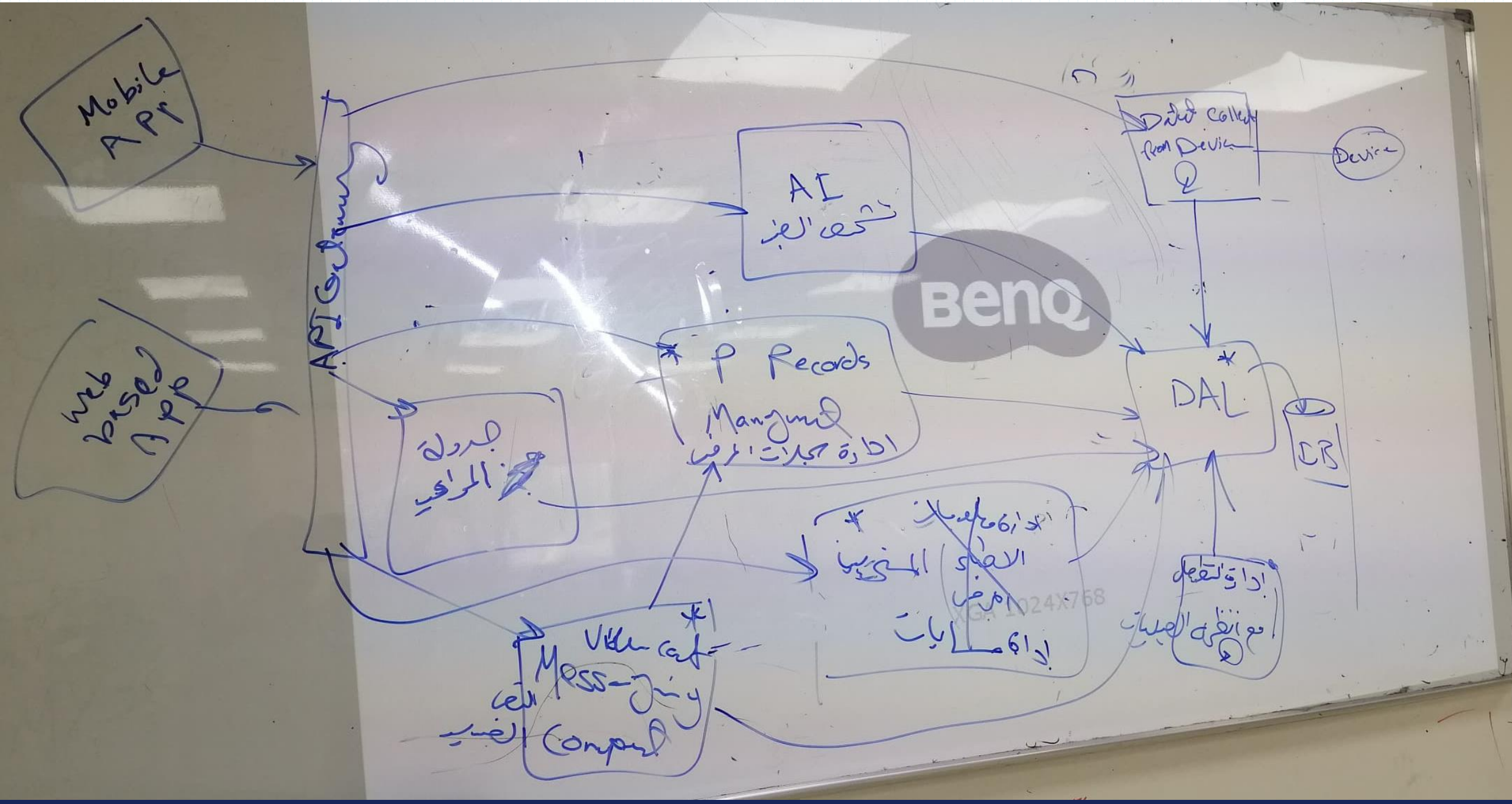
- **Coupling** measures dependency among subsystems
  - Low coupling can be achieved if a calling class does not need to know anything about the internals of the called class (Principle of information hiding)

Case Study

حالة عملية للنقاش

شركة تطوير تعمل على بناء نظام برمجي لإدارة خدمات الرعاية الصحية عن بُعد، ويتميز النظام المراد بالخصائص التالية:

- يتكامل النظام مع أجهزة قياس العلامات الحيوية للمرضى لتحصيل البيانات والقياسات الحيوية مباشرة لتحليلها.
- يوفر واجهة مستخدم تفاعلية للأطباء والممرضين تتيح لهم مراقبة الحالة الصحية للمرضى بشكل لحظي عند الحاجة وتقديم الاستشارات عبر الإنترنت.
- يتيح النظام للمرضى حجز المواعيد والتواصل مع الأطباء عبر المحادثات النصية أو مكالمات الفيديو.
- يقدم النظام توصيات طبية مخصصة بناءً على بيانات المرضى وسجلهم الصحي باستخدام تقنيات الذكاء الصنعي.
- يوفر النظام خدمة إدارة الوصفات الطبية الإلكترونية، حيث يمكن للأطباء إصدار الوصفات وتحويلها مباشرة إلى الصيدليات.
- يدعم النظام تخزين الملفات الطبية بشكل آمن مع إمكانية مشاركتها مع مقدمي الرعاية الصحية بناءً على موافقة المريض.


Mobile APP
Web based APP
API Gateway
AI
Data collect from Device
Device
P Records
Manager
DAL
CB
Messaging
Component

What are some good designs?

Software Architecture Style

Software Architecture Style

Layered Architecture: introduction

Software Architecture Style

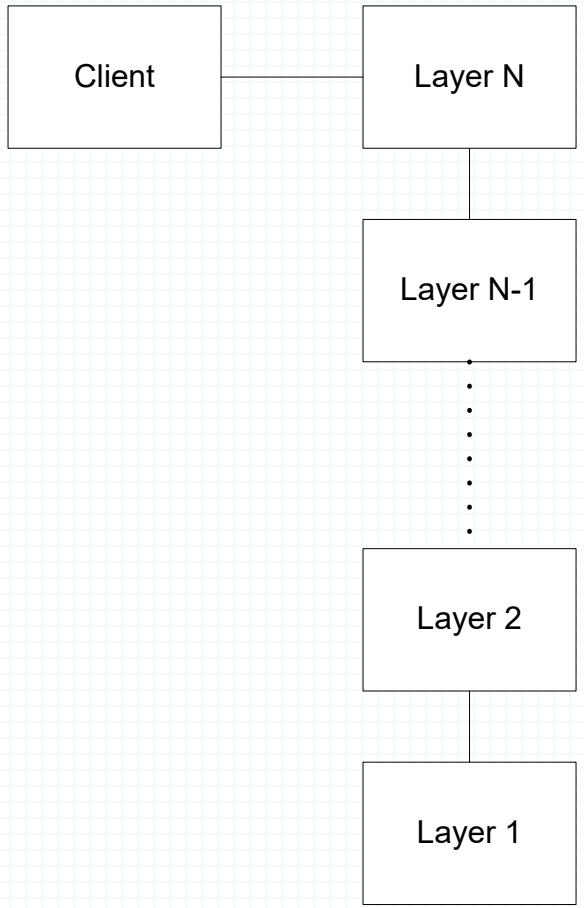
- A software architecture style is a solution to a **recurring problem** that is well understood, in a particular context.
- Each pattern consists of a **context, a problem, and a solution.**
- Using patterns simplifies design and allows us to gain the benefits of **using a solution that is proven to solve** a particular design problem.
- **More than one software architecture pattern** can be used in a single software system. These patterns can be combined to solve problems.

Example: Layered Architecture

- The Problem!
  - You are designing a system that needs to handle **a mix of low-level and high-level issues**
  - High-level operations rely on lower-level ones
  - The system is **large**, and a methodical approach to organizing it is needed to keep it understandable

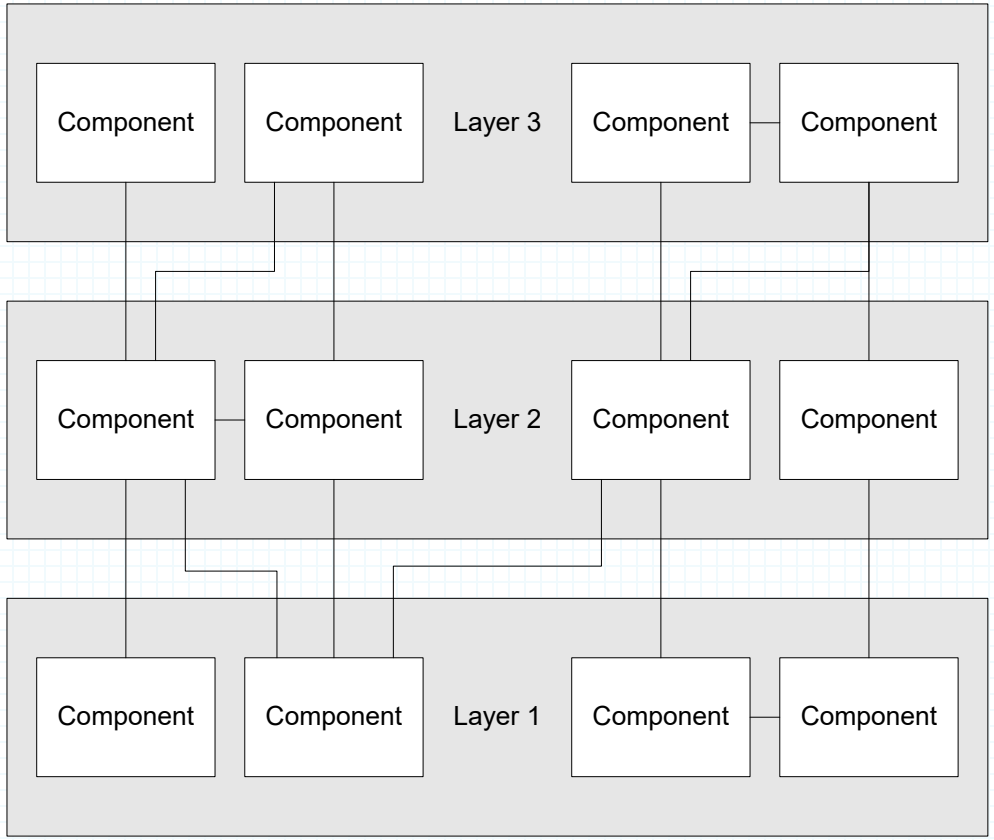
Example: Layered Architecture

- Structure the system into an appropriate **number of layers**, and place them on top of each other
- Each layer represents a **level of abstraction**
- Classes are placed in layers based on their levels of abstraction
- Layer 1 is at the bottom and contains classes **closest** to the hardware/OS
- Layer N is at the top and contains classes that **interact directly** with the system's clients



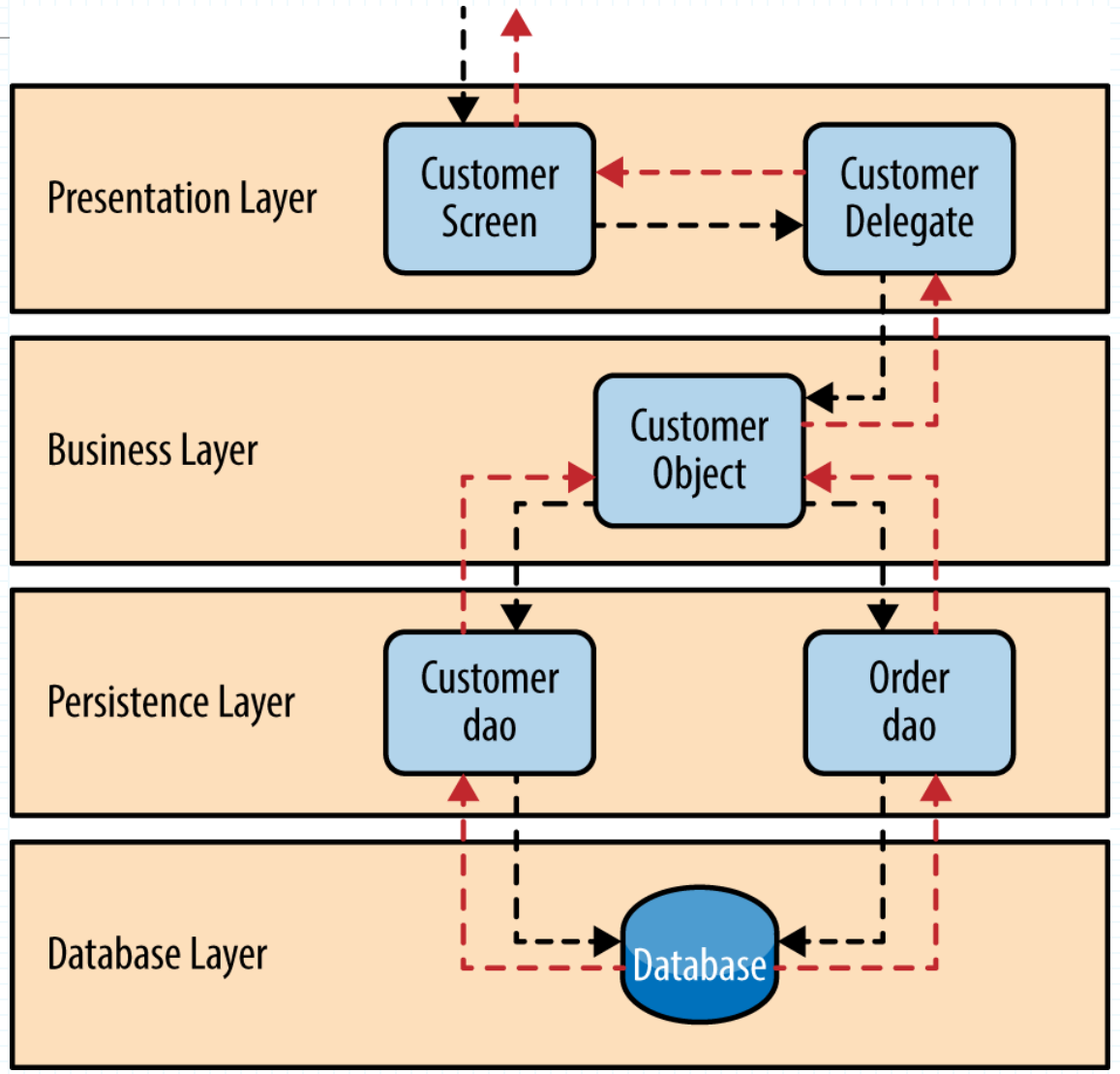
Example: Layered Architecture

- Layer J uses the services of Layer J-1 and provides services to Layer J+1
- Most of Layer J's services are implemented by composing Layer J-1's services in meaningful ways
- Layer J **raises the level of abstraction** by one level
- Classes in Layer J may also use each other



Example: Layered Architecture

- Data Base Layer
- Persistence Later (or Data access layer)
- Business Layer.
- Presentation Layer.



Example

This example has been discussed and solved on the whiteboard during our class session.

- Design a layered architecture for a system that includes the following features:

1. **Course Management**: The ability to manage courses (add, remove, update courses).
2. **Student Management**: The ability to manage students (add, remove, update student information).
3. **Enroll Students in Courses**: Implement business rules for enrollment:
    1. A course can have no more than 50 students.
    2. A student can be enrolled in no more than 5 courses.
4. **Print Course Enrollments**: Display a list of students for any given course.