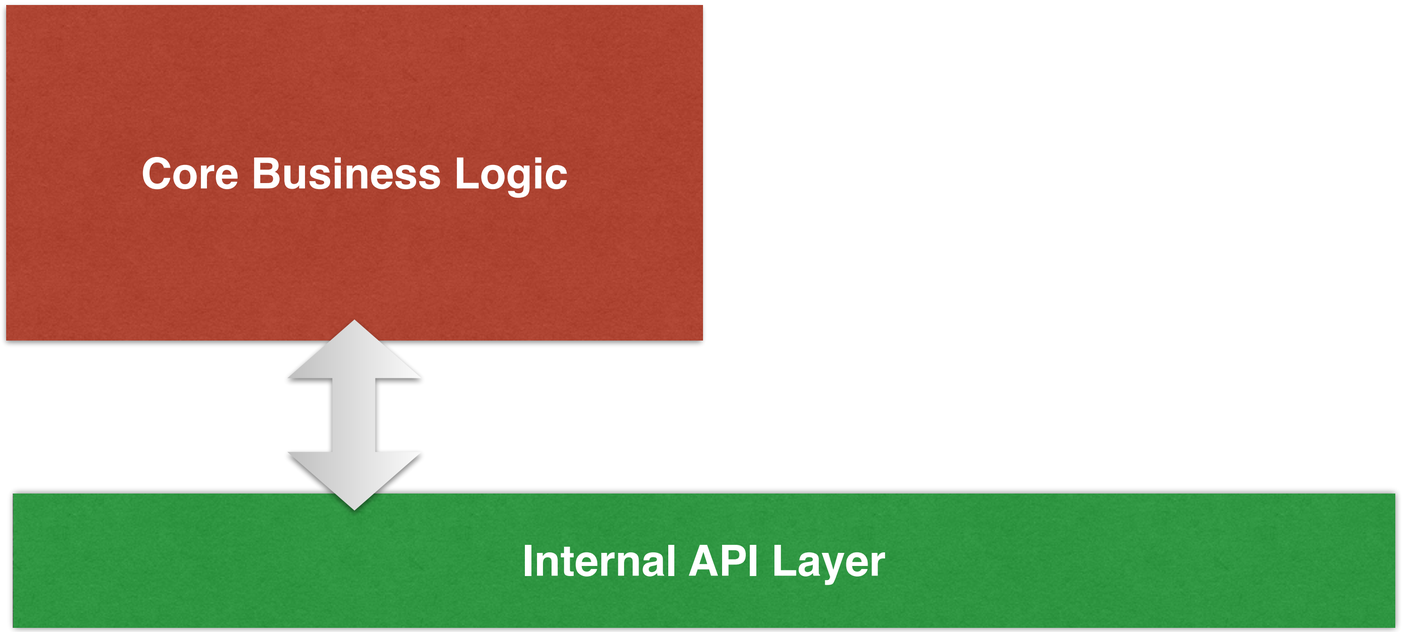
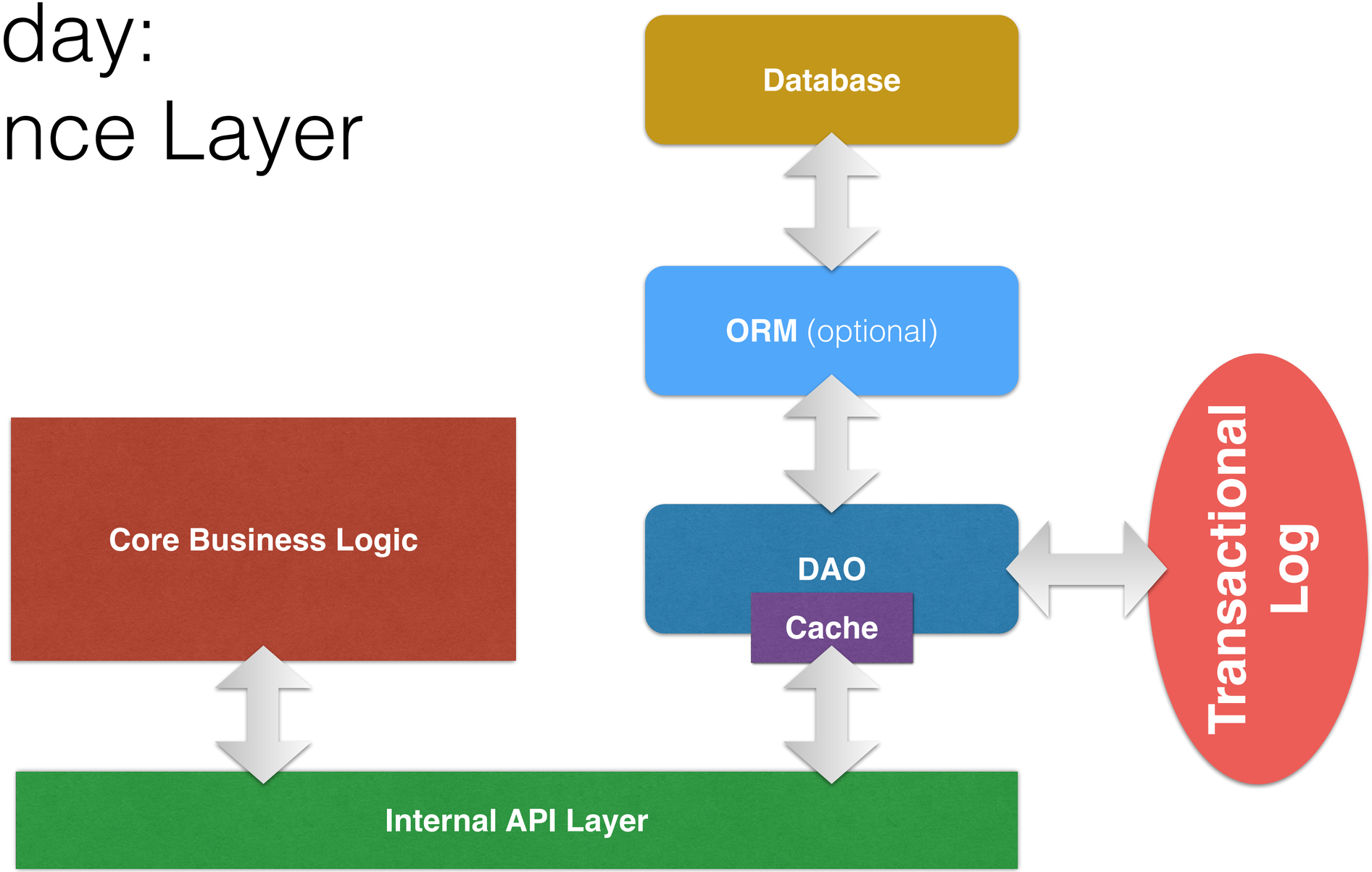
Lecture 4

CSCI 6907.12 - Full Stack Application Engineering

Where we are so far



Today:
Persistence Layer



Persistence Layer

- **Purpose**: To store any data that needs to exist outside your application's single life cycle.
- **Why do we need it**: It is possible to store all your data in memory but hardwares fail, things get corrupted and processes will need to be shut down
- Think of it as temporal storage, not the source of truth.

Parts

- **DAO**: Data Access Object
  - Interface for storing and retrieving your data
- **ORM**: Object Relational Mapper
  - Abstraction over your database interface
- **Database -** The "physical" location of your data
- **Cache -** Ephemeral store for your data, for faster operation
- **Transactional Log -** History of all data related transaction, your source of truth

Database

- Relational vs. Document Oriented

Relational Database

- Based on the relational model of data.
- Tables and rows and Columns
- "Links" two data types (tables), via foreign key
- SQL (Structured Query Language)

Relational Database

- **Pros**:
  - *Type safety* - you are guaranteed that a data is composed of certain attributes (columns) and those attributes are guaranteed to be of certain data type
  - *Normalized Data* - allows compartmentalized view of your data, which saves space and makes complex queries like joins easy
  - *Established History* - the technology has been around since '80s. Proven solution and lots of support
- **Cons**:
  - *Rigid Structure* - once your data organization has been defined, it's hard to change it
  - *Cannot model malleable data* - your application's data can vary depends on usage. Only way to model is in RD is to use null columns
  - *Not suitable for prototyping* - at the design and prototyping stage data organization WILL change

RDMS

- **Relational Database Management Systems**
  - PostgreSQL
  - MySQL (MariaDB)
  - **SQLite**
  - and many many more

Document Oriented Database

- Manages document-oriented information, also known as semi-structured data
- Often called NoSQL databases
- At its core, a key-value store
- Retrieval via metadata
- No predetermined data structure
- Common Document format: XML, YAML, JSON

Document Oriented Database

- **Pros**:
  - *Flexibility* - since there are no predetermined structure to your data, you can easy change the schematics of your data; good for prototyping
  - *Works well with OOP* - because single object maps well into a serialized format, storing and retrieval does not require additional processing
  - *Large Data-set* - when your data-set grows, probability of having uniform data structure decreases
- **Cons**:
  - *No Normalization* - since entire view of an object is stored as a single document, you cannot reuse parts of the data, thus resulting in more computation and more storage space
  - *No Type Safety* - you are not guaranteed that the document that you retrieve will be in certain structure or certain type
  - *Relatively New Technology* - at it's been popular for less than 10 years

Document-Oriented Database

- **MongoDB**
- CouchDB
- PostgreSQL*
- And many more!

Object Relational Mapper

- If you are using a Relational Database, way your data is structured in your code and your database might not be the same.
- ORM abstracts away the database so that you can think about storing objects as objects and not a set of tables, rows and columns
- ORM also tries to create a abstraction layer in a way that you can be agnostic about the actual underlying database

Object Relational Mapper

- **Pros**:
  - *No need to code against a single database*
  - *You can code in terms of Objects not tables*
  - *Adds additional type safety* - since database types and your object types might not match, ORM can help type check the mapping
- **Cons**:
  - *Abstraction is not perfect* - as your data and your queries get complex, you'll run into a situation where abstraction provided by the ORM is not enough
  - *Adds Overhead* - because it's yet another abstraction, you have to think about the performance loss
  - *Yet another technology to learn and maintain*

ORM

- Java - Hibernate
- Python - SQLAlchemy
- Scala - Slick
- Ruby - Sequel

DAO

- **Data Access Object** - Interface into your data
- This abstraction allows you to easily swap out your ORM/Database
- More in the demo

Cache

- Because database retrieval is usually done over the network and require some level of computation (joins, map-reduce), it might be a good idea to cache the result.
- Also, if you know that your data doesn't change much (or know when it will change), it's a good idea to cache the result

Cache

- **Memcached**
- Redis
- Any many more

Transaction Log

- Every data related transaction should be logged
- This should happen at the DAO abstraction so that we can rebuild the database from scratch in other system
- This should be your source of truth not your database

Transaction Log

- Apache Kafka
- RabbitMQ
- ZeroMQ
- Or just plain log

Service Oriented Architecture