COMP6452 Software Architecture for Blockchain Applications
Week 3 Tutorial

David Zhang
d.zhang@unsw.edu.au

Yue Liu
yue.liu13@unsw.edu.au

Term 2, 2021

Hands-on sub-session

Please download and install Node.js with NPM (https://nodejs.org/en/download/) and Truffle Ganache (https://www.trufflesuite.com/ganache), which will be used in next tutorial.
You may then have a look at the example repository (https://gitlab.cse.unsw.edu.au/z5140586/comp6452_oracle_example) beforehand.

Project 1 Troubleshooting

The first part of this sub-session will be about project 1 troubleshooting.

Oracles

Many applications built on blockchain need to interact with other external systems, so that the validation of blockchain transactions might depend on states of those external systems. However, to preserve the deterministic validation of blocks, smart contracts can only access data previously stored on the blockchain and submitted via a transaction; hence, cannot use external data. Blockchain oracles address this limitation by providing the required data from external systems to the blockchain, including for use in smart contracts. The use of oracles makes communication possible from the external world to the blockchain, for example by recording external data on the blockchain in transactions[1]. Alternatively, oracles are also used to push data from blockchain into the external systems.

There are several classification of oracles[2]:

[1]Source: Lo S K, Xu X, Staples M, et al. Reliability analysis for blockchain oracles[J]. Computers & Electrical Engineering, 2020, 83: 106582.

[2]Source: https://blockchainhub.net/blockchain-oracles/

1. **Software Oracles**: handle information that originates from online sources, like temperature, prices of commodities and goods, flight or train delays, etc. The software oracle extracts the needed information and pushes it into the smart contract.

2. **Hardware Oracles**: Some smart contracts need information directly from the physical world, for example, a car crossing a barrier where movement sensors must detect the vehicle and send the data to a smart contract, or RFID sensors in the supply chain industry.

3. **Inbound Oracles**: provide data from the external world. This can be further divided into three modes:

- collect and go (push, sync)
  Data are periodically pushed onto a smart contract, and user smart contract just query synchronously
- subscribe
  Data are periodically pushed onto a smart contract, and user smart contract will be notified for the data
- request and response (pull, async)
  Data are provided on request, and user smart contract will be notified with the response

4. **Outbound Oracles**: provide smart contracts with the ability to send data to the outside world. An example would be a smart lock in the physical world, which receives payment on its blockchain address and needs to unlock automatically.

5. **Consensus-based Oracles**: get their data from human consensus and prediction markets. Using only one source of information could be risky and unreliable. To avoid market manipulation, prediction markets implement a rating system for oracles. For further security, a combination of different oracles may be used, where, for example, three out of five oracles could determine the outcome of an event.

A Pull-mode Oracle

We will build up our own simple oracle shown in figure 1 in the next tutorial. It will pull temperature information into user smart contract.

The user app smart contract will request data through a handler, and the oracle contract receives this request with a request id. It emits an event with request id and address of user app smart contract, with request params. The blockchain listener will catch this event, and query the trusted information provider. Once the data is ready, the blockchain listener will invoke the function of oracle contract with data, request id and user app address attached. The oracle contract then redirects the data to the user app contract.

The system can be described using C4 model[3]. Figure 1 shows the first 3 C's, i.e. Context, Container, Component, of an oracle.

[3]C4 model: https://en.wikipedia.org/wiki/C4_model

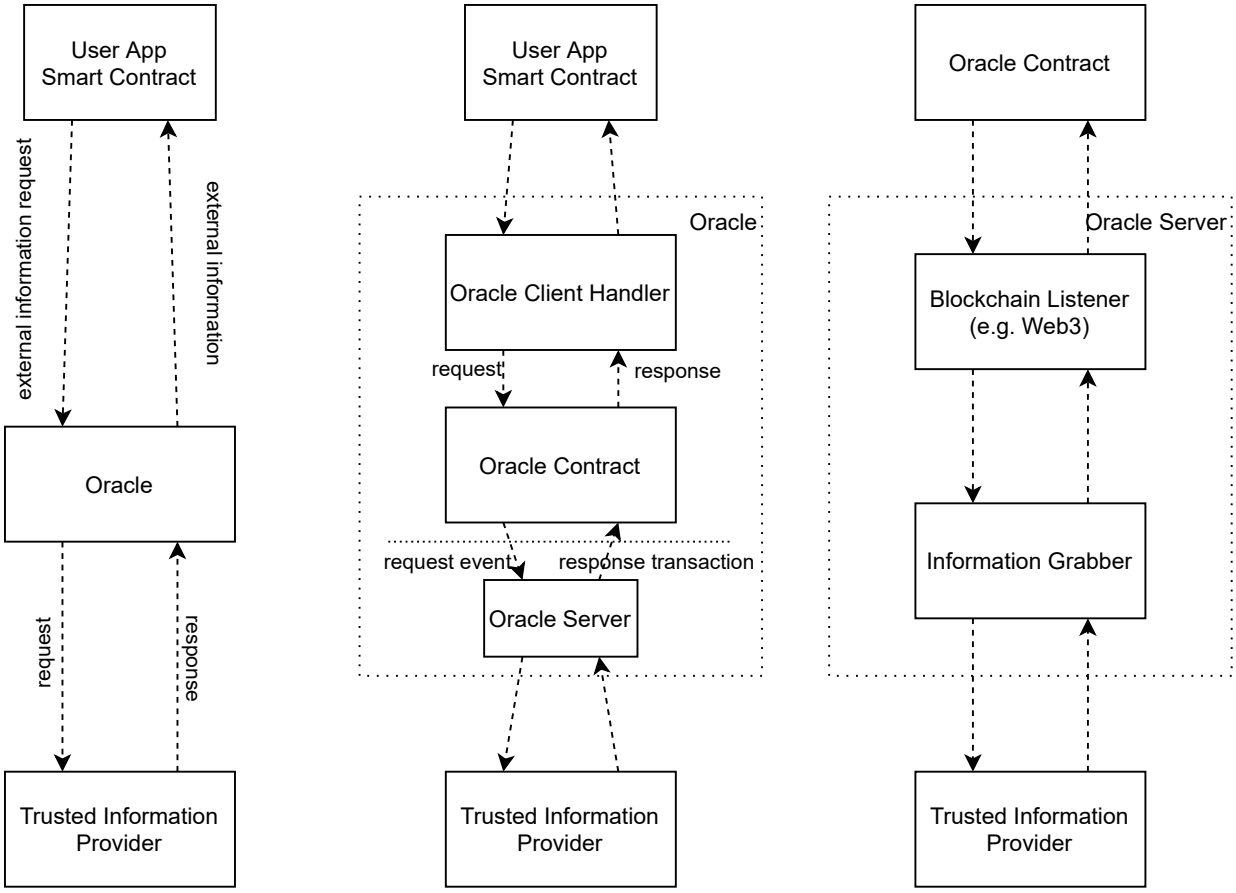

Figure 1: oracle example - context, container, component

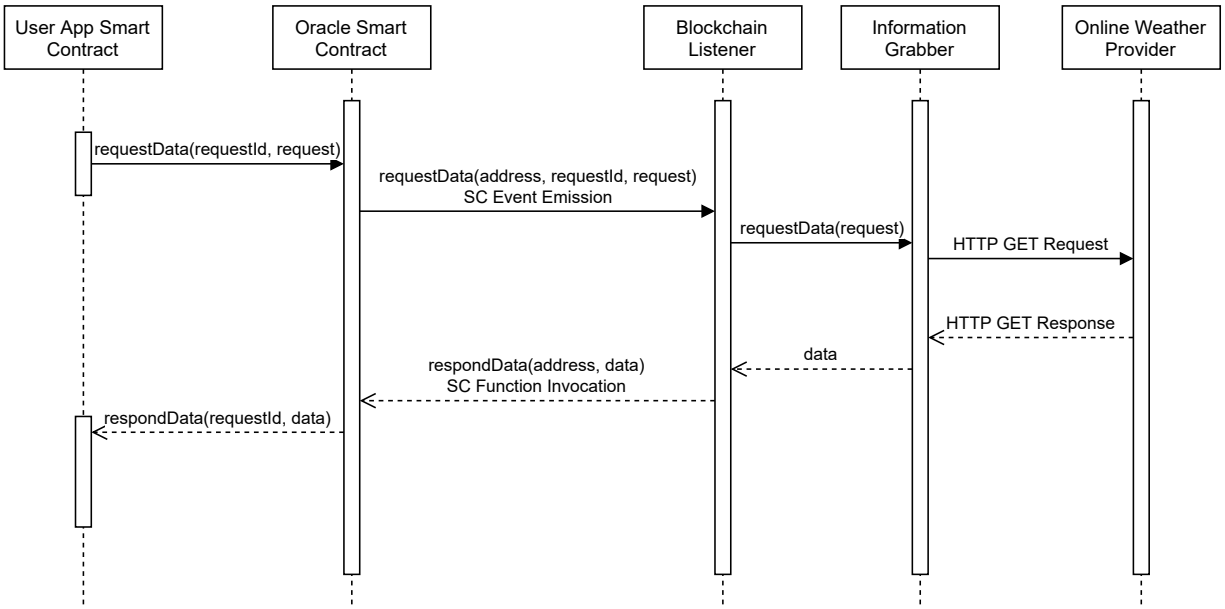

Figure 2: oracle example - sequence diagram

Oracle Services

There are companies providing oracle services. These oracles are more complex and use systems with high security to achieve trustworthiness.

- Provable (formerly Oraclize)
- Chainlink
- Gardener
- Town Crier (acquired by Chainlink)

UML Diagrams

Diagrams are not to complicate you, but to provide a common language to make things clear.

The content of this section is adapted from Wikipedia

In this section, we will briefly introduce UML diagrams to help on your tasks in Project 2.

The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system.

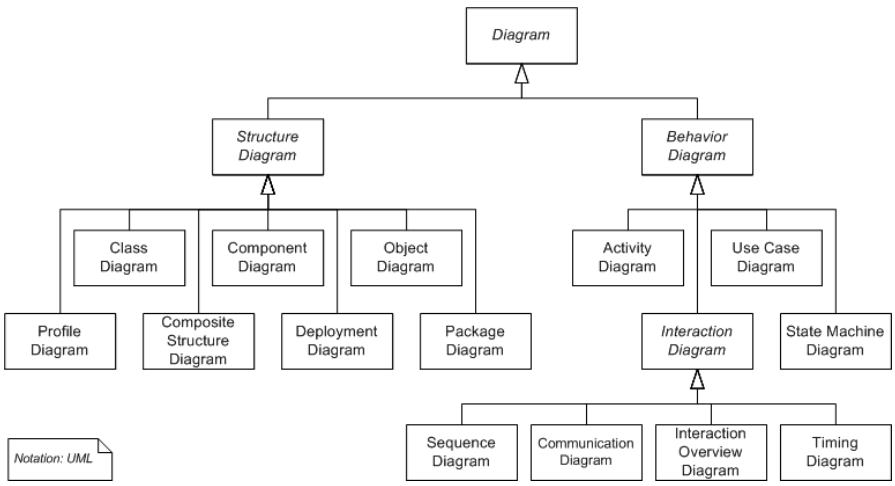

Figure 3: UML diagrams. (Source: https://en.wikipedia.org/wiki/Class_diagram)

First, **structure diagrams** represents the static aspects of the system. It emphasize the things that must be present in the system being modeled (e.g. class diagram). Secondly, **behavior diagrams** represent the dynamic aspect of the system. It emphasizes what must happen in the system being modeled. Since behavior diagrams illustrate the behavior of a system, they are used extensively to describe the functionality of software systems (e.g., use case diagram, state diagram, sequence diagram, activity diagram).

Class Diagram

A class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

In the diagram, classes are represented with boxes that contain three compartments:

- The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.
- The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.
- The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase.

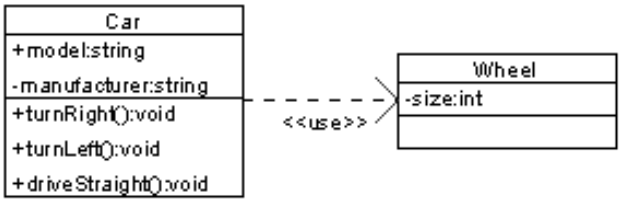

Figure 4: Class diagram example 2. (Source: https://en.wikipedia.org/wiki/Class_diagram)

There are seven relations between different classes:

- **Association**: defines a relationship between classes of objects that allows one object instance to cause another to perform an action on its behalf. ("sending a message", "invoking a method" or "calling a member function")
- **Inheritance**: a class (subclass) is considered to be a specialized form of the other class (superclass).
- **Realization**: one model element (the client) realizes (implements or executes) the behavior that the other model element (the supplier) specifies.
- **Dependency**: exists between two elements if changes to the definition of one element (the server or target) may cause changes to the other. This association is uni-directional.
- **Aggregation**: an association that represents a part-whole or part-of relationship; must be a binary.
- **Composition**: similar to *Aggregation*, but when the container is destroyed, the contents are also destroyed.

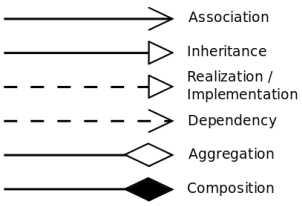

Figure 5: UML relations notation. (Source: https://en.wikipedia.org/wiki/Class_diagram)

Use Case Diagram

A use case diagram is a graphical depiction of a user's possible interactions with a system. The use cases are represented by either circles or ellipses. The actors are often shown as stick figures.

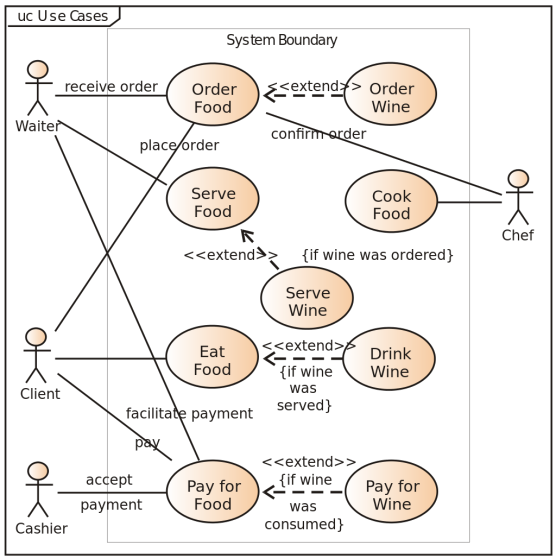

Figure 6: Use case diagram example. (Source: https://en.wikipedia.org/wiki/Use_case_diagram)

State Diagram

States are mainly event-driven, which means that they continuously wait for the occurrence of some external or internal events. The UML state diagrams are directed graphs in which nodes denote states and connectors denote state transitions. The initial transition originates from the solid circle and specifies the default state when the system first begins. Every state diagram should have such a transition, which should not be labeled, since it is not triggered by an event.

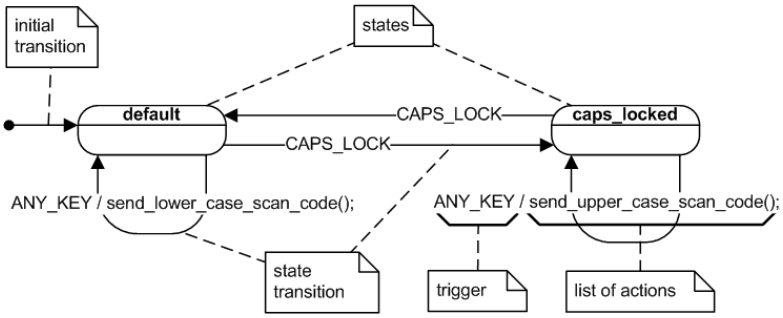

Figure 7: State diagram example. (Source: https://en.wikipedia.org/wiki/UML_state_machine)

Sequence Diagram

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

The lifeline demonstrates a role. Messages, written with horizontal arrows with the message name written above them, display interaction. Solid arrow heads represent synchronous calls, open arrow heads represent asynchronous messages, and dashed lines represent reply messages.

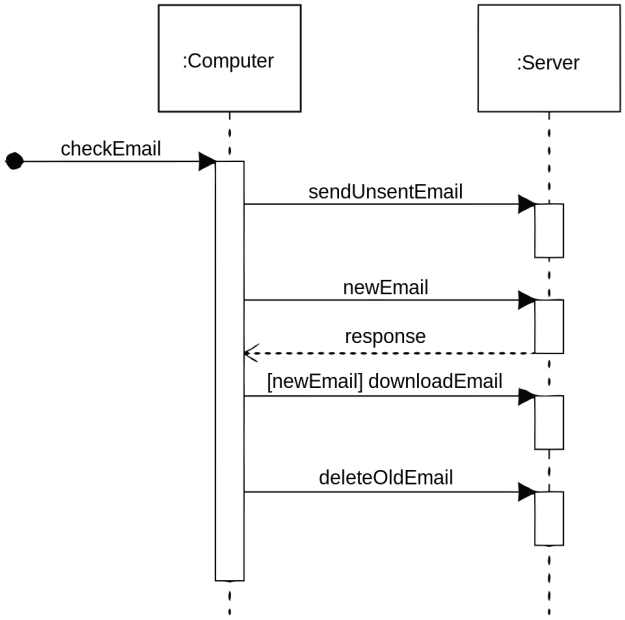

Figure 8: Sequence diagram example. (Source: https://en.wikipedia.org/wiki/Sequence_diagram)

Activity Diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency.

An activity diagram consists of:

- ellipses represent actions;
- diamonds represent decisions;
- bars represent the start (split) or end (join) of concurrent activities;
- a black circle represents the start (initial node) of the workflow;
- an encircled black circle represents the end (final node).

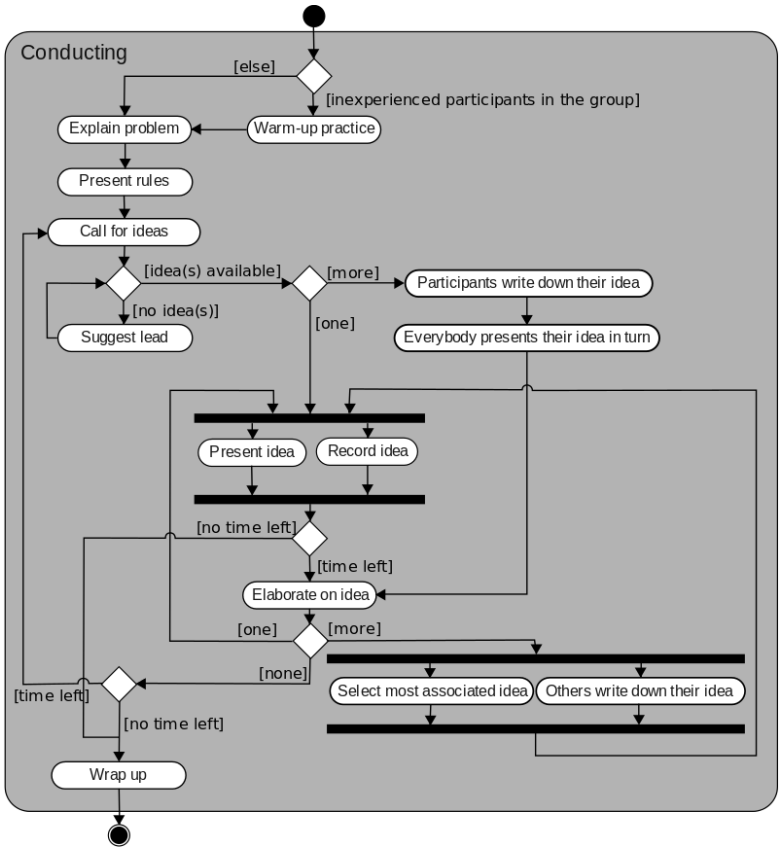

Figure 9: Activity diagram example. (Source: https://en.wikipedia.org/wiki/Activity_diagram)

Blockchain-based Software Architecture Examples

Please see Fig. 10 and 11.

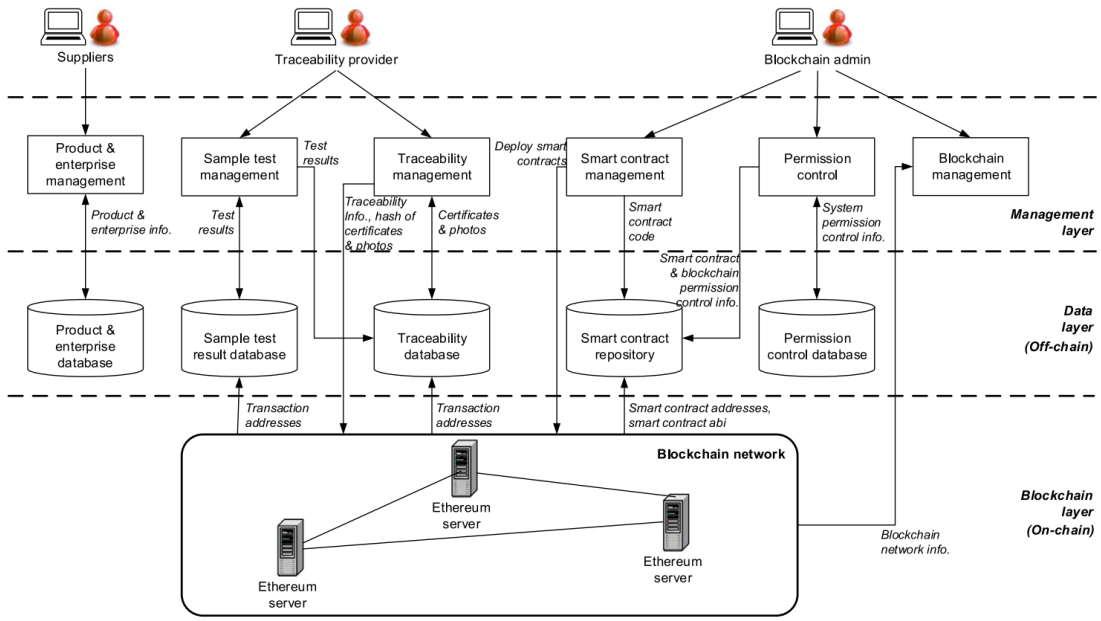

Figure 10: Architecture example 1. (Source: Xu X, Lu Q, Liu Y, et al. Designing blockchain-based applications a case study for imported product traceability[J]. Future Generation Computer Systems, 2019, 92: 399-406.)

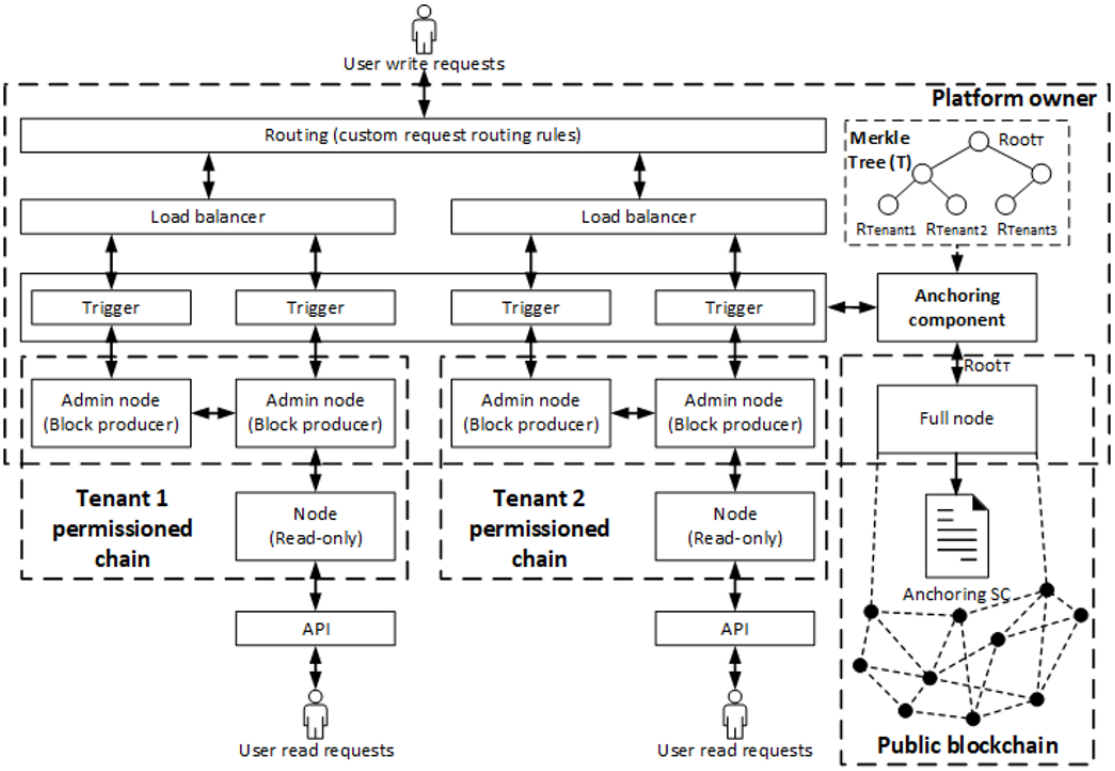

Figure 11: Architecture example 2. (Source: Weber I, Lu Q, Tran A B, et al. A platform architecture for multi-tenant blockchain-based systems[C]//2019 IEEE International Conference on Software Architecture (ICSA). IEEE, 2019: 101-110.) Can also refer to the Guest Lecture of Week 3.

Theory sub-session

Course Discussion

- What are the metrics to classify the types of blockchain?
  Decentralisation level; Deployment; Ledger Structure; Consensus protocol; Block configuration and data structure; Auxiliary blockchain; Anonymity; Incentive.

- What are the main aspects to making design trade-off on when using blockchain technology?
  Design decisions: on-/off-chain data storage; data encryption; choosing different types of blockchain...
  Software quality attributes: security; performance; flexibility; scalability; transparency; privacy...

- What are the metrics to evaluate whether the use of blockchain is suitable?
  multi-party; trusted authority; centralised operation; immutability; performance; transparency...