EP17: Design patterns cheat sheet. Also...

ALEX XU
JUL 30, 2022

For this week's newsletter, we will cover:

- Design patterns cheat sheet
- 6 ways to turn code into beautiful architecture diagrams
- What is a File Descriptor?
- Scan to pay in 2 minutes
- Direct payments

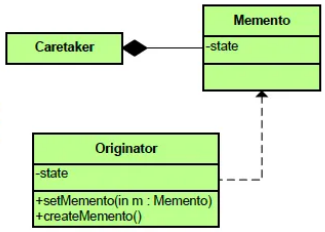
Design patterns cheat sheet

- C Abstract Factory
- S Adapter
- S Bridge
- C Builder
- B Chain of Responsibility
- B Command
- S Composite
- S Decorator
- S Facade
- C Factory Method
- S Flyweight
- B Interpreter
- B Iterator
- B Mediator
- B Memento
- C Prototype
- S Proxy
- B Observer
- C Singleton
- B State
- B Strategy
- B Template Method
- B Visitor

Memento

Type: Behavioral

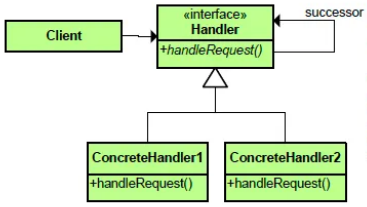
What it is:
Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.



Chain of Responsibility

Type: Behavioral

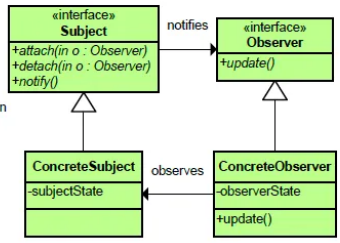
What it is:
Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.



Observer

Type: Behavioral

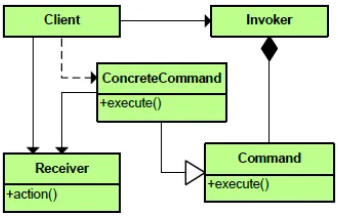
What it is:
Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.



Command

Type: Behavioral

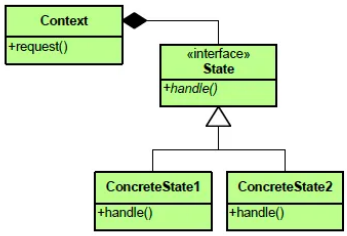
What it is:
Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.



State

Type: Behavioral

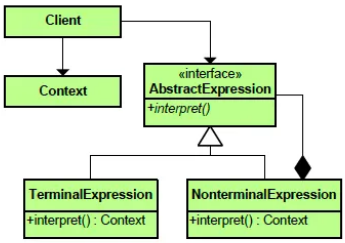
What it is:
Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.



Interpreter

Type: Behavioral

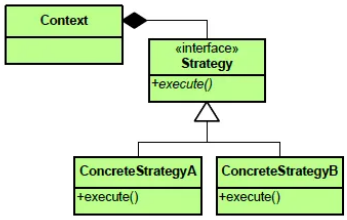
What it is:
Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.



Strategy

Type: Behavioral

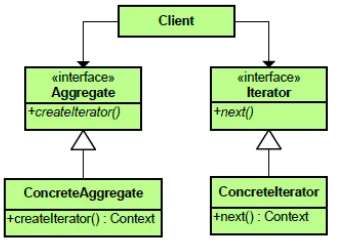
What it is:
Define a family of algorithms, encapsulate each one, and make them interchangeable. Lets the algorithm vary independently from clients that use it.



Iterator

Type: Behavioral

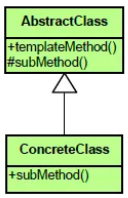
What it is:
Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.



Template Method

Type: Behavioral

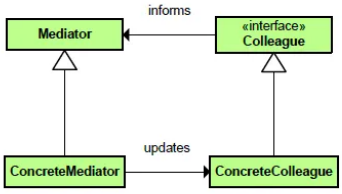
What it is:
Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.



Mediator

Type: Behavioral

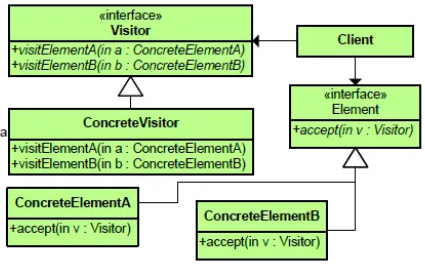
What it is:
Define an object that encapsulates how a set of objects interact. Promotes loose coupling by keeping objects from referring to each other explicitly and it lets you vary their interactions independently.



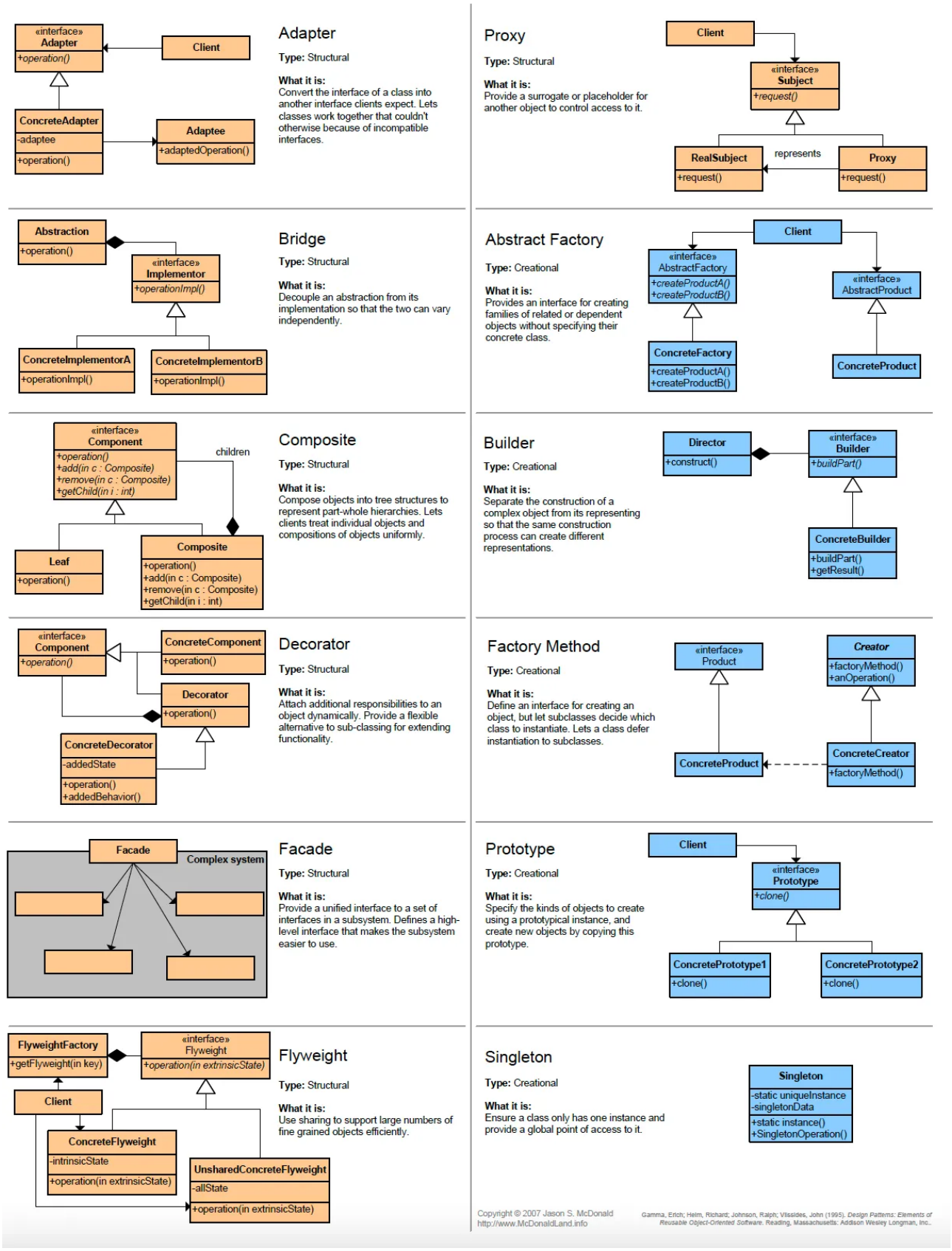
Visitor

Type: Behavioral

What it is:
Represent an operation to be performed on the elements of an object structure. Lets you define a new operation without changing the classes of the elements on which it operates.



Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison Wesley Longman, Inc..



6 ways to turn code into beautiful architecture diagrams

1. Diagrams

Turn python code into cloud system architecture diagrams

Diagram as Code

```
from diagrams import Cluster, Diagram
from diagrams.aws.compute import ECS
from diagrams.aws.database import ElastiCache, RDS
from diagrams.aws.network import ELB
from diagrams.aws.network import Route53

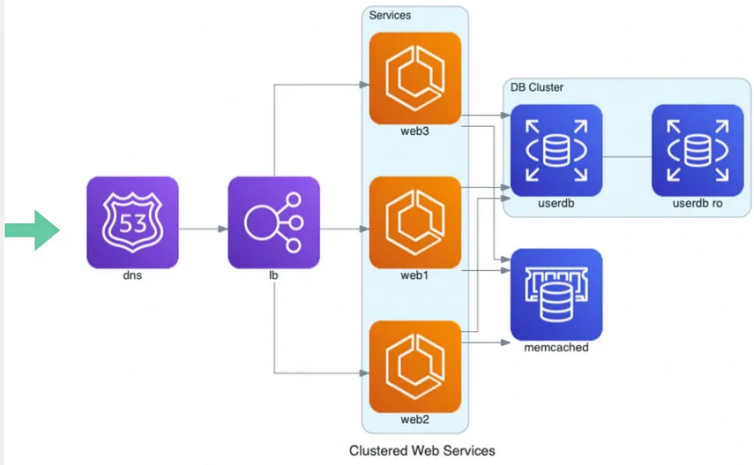
with Diagram("Clustered Web Services", show=False):
    dns = Route53("dns")
    lb = ELB("lb")

    with Cluster("Services"):
        svc_group = [ECS("web1"),
                     ECS("web2"),
                     ECS("web3")]

    with Cluster("DB Cluster"):
        db_primary = RDS("userdb")
        db_primary - [RDS("userdb ro")]

    memcached = ElastiCache("memcached")

    dns >> lb >> svc_group
    svc_group >> db_primary
    svc_group >> memcached
```



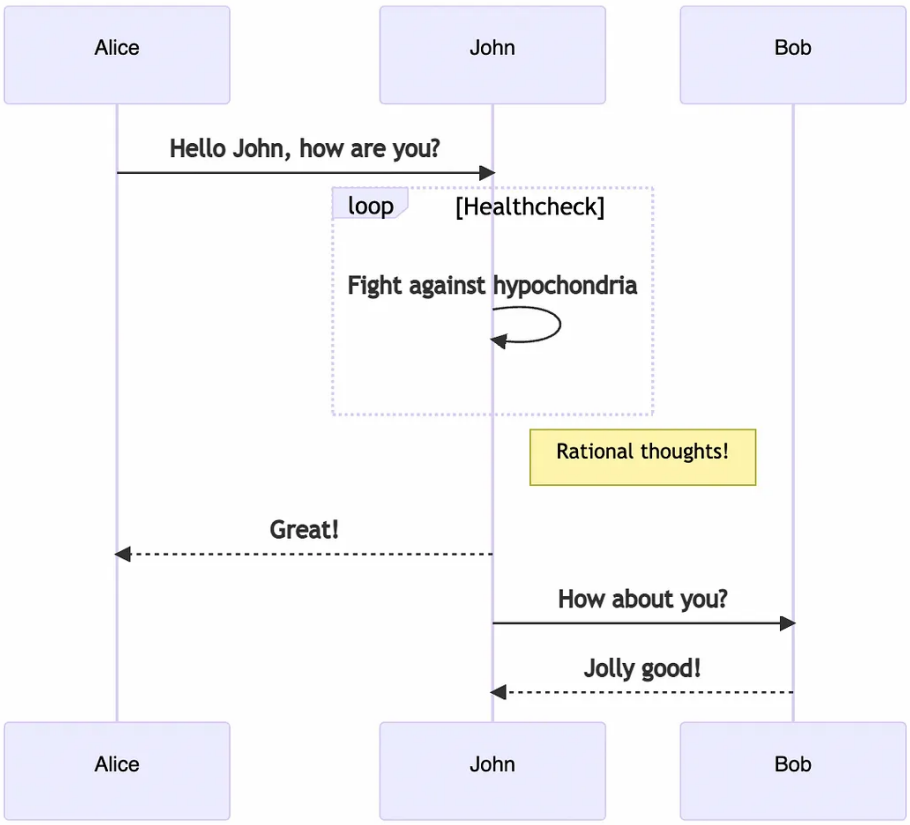
Thanks for reading ByteByteGo Newsletter! Subscribe for free to receive new posts and support my work.

2. Mermaid

Generation of diagram and flowchart from text in a similar manner as markdown

Example:

```
sequenceDiagram
Alice->>John: Hello John, how are you?
loop Healthcheck
    John->>John: Fight against hypochondria
end
Note right of John: Rational thoughts!
John-->>Alice: Great!
John->>Bob: How about you?
Bob-->>John: Jolly good!
```



3. ASCII editor

Free editor: Asciiflow, dot-to-ascii

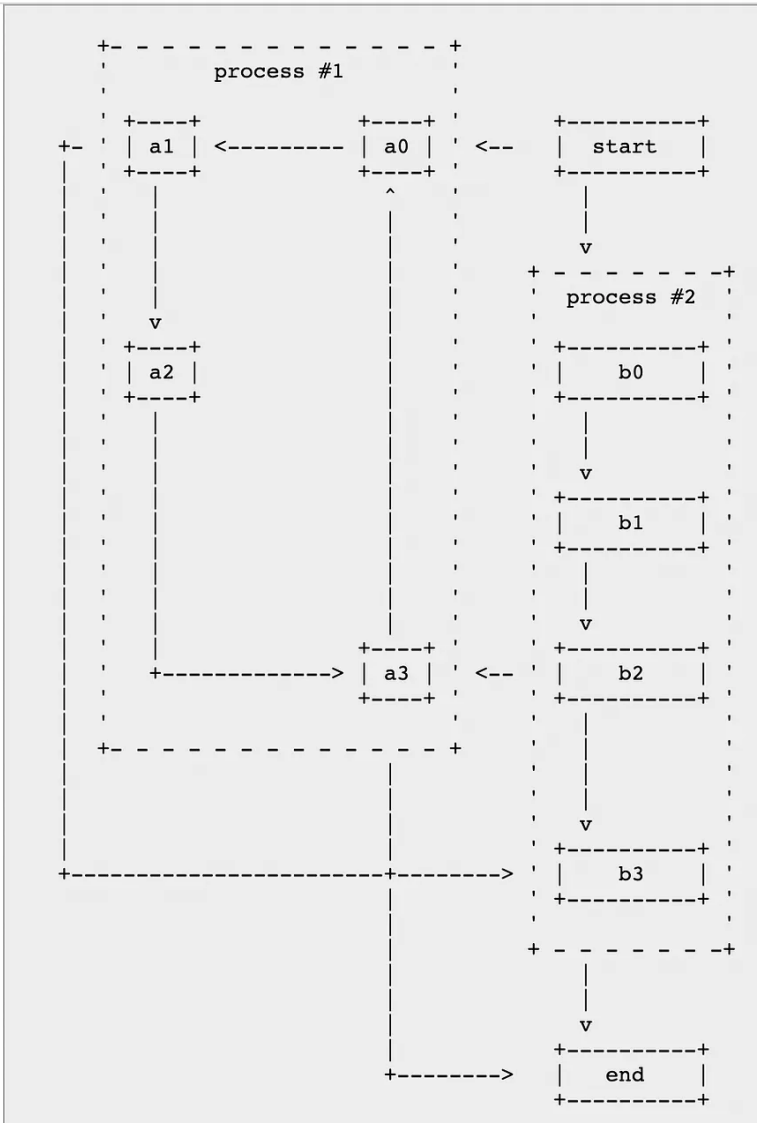
Paid editor: Monodraw

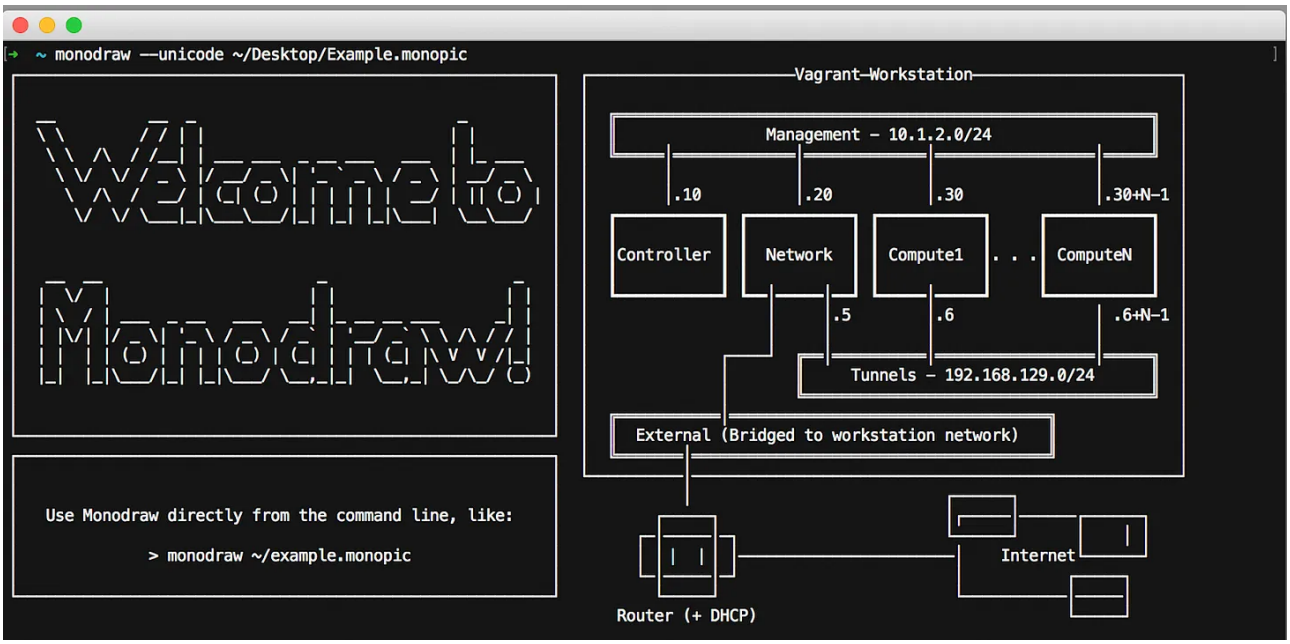
```
# sample 0

digraph {
    subgraph cluster_0 {
        a0 -> a1 -> a2 -> a3;
        label = "process \#1";
    }

    subgraph cluster_1 {
        b0 -> b1 -> b2 -> b3;
        label = "process \#2";
    }

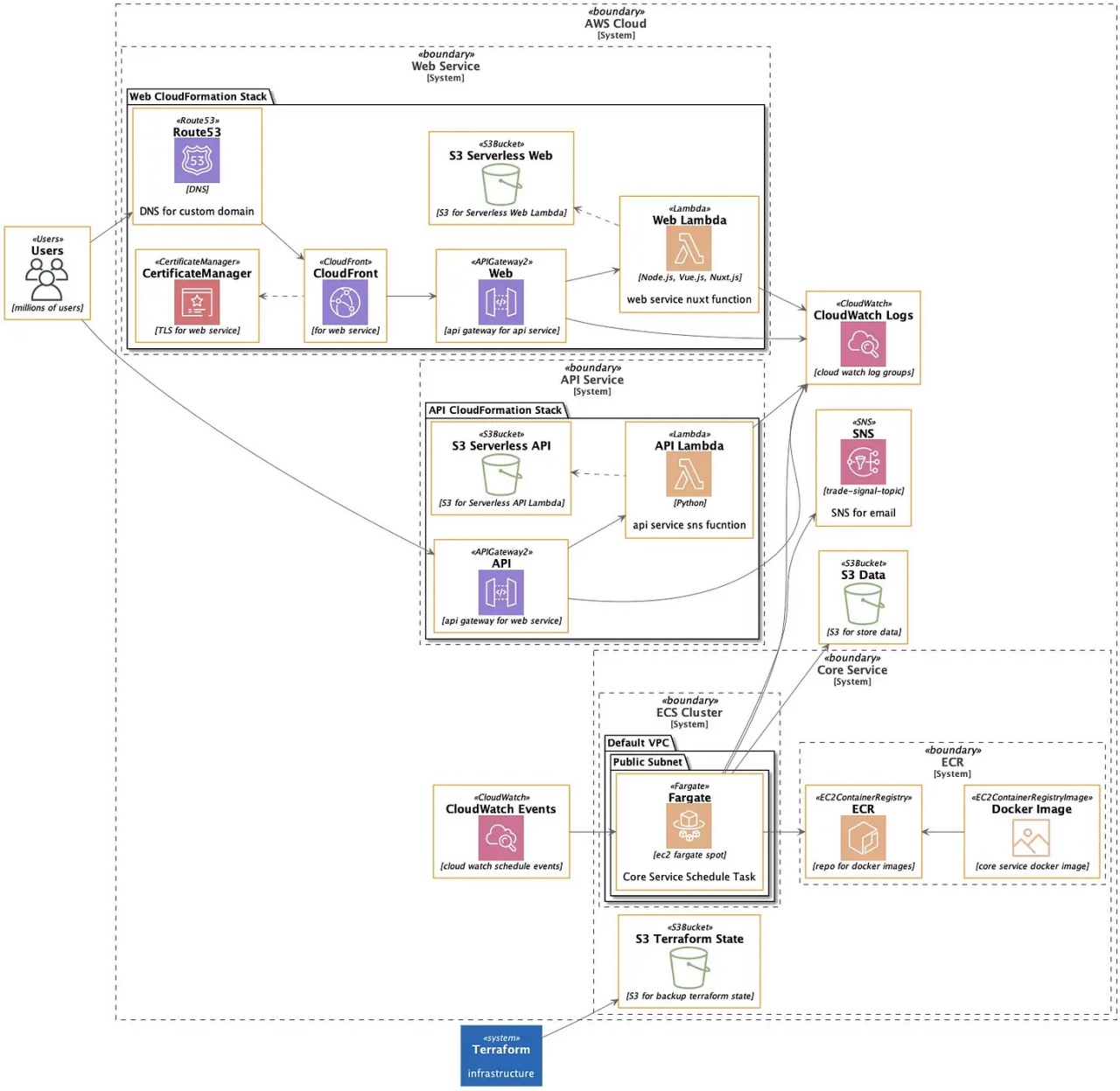
    start -> a0;
    start -> b0;
    a1 -> b3;
    b2 -> a3;
    a3 -> a0;
    a3 -> end;
    b3 -> end;
}
```





4. PlantUML

It is an open-source tool allowing users to create diagrams from plain text language.



5. Markmap

Visualize your Markdown as mindmaps. It supports the VS code plugin.

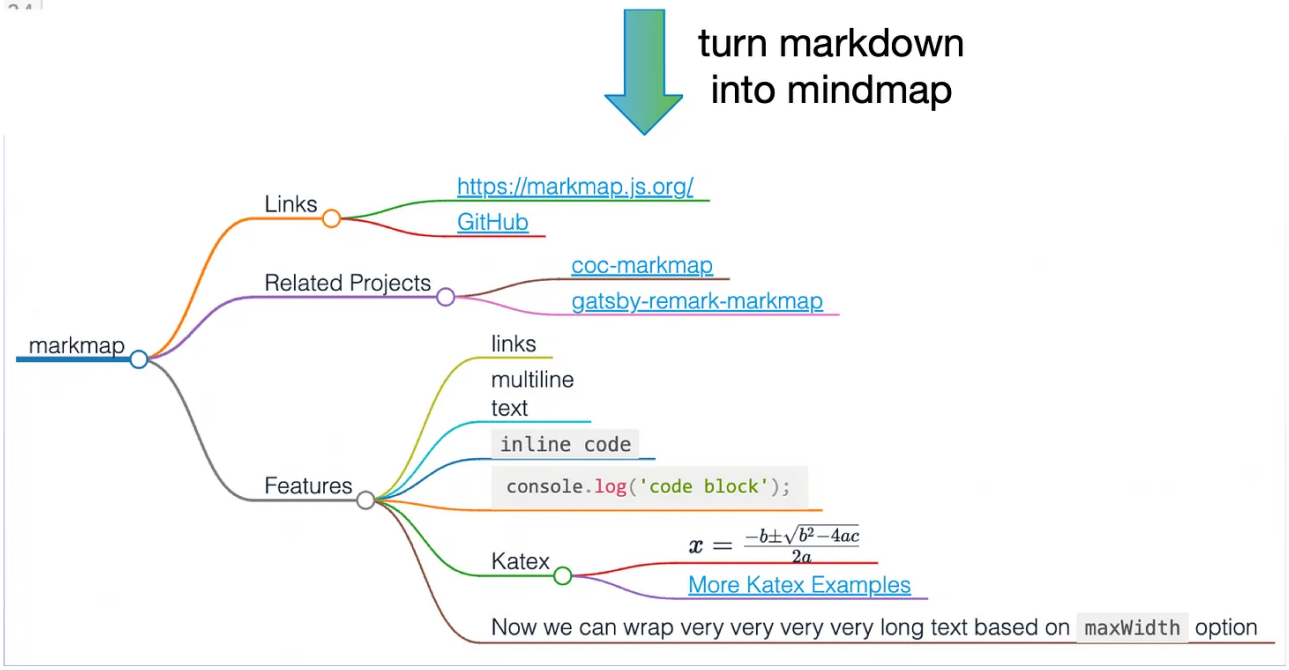
markmap

```
# markmap

## Links
- <https://markmap.js.org/>
- [GitHub](https://github.com/gera2ld/markmap)

## Related Projects
- [coc-markmap](https://github.com/gera2ld/coc-markmap)
- [gatsby-remark-markmap](https://github.com/gera2ld/gatsby-remark-markmap)

## Features
- links
- multiline
  text
- `inline code`
-
    ```js
    console.log('code block');
    ```
- Katex
  - $x = {-b \pm \sqrt{b^2-4ac} \over 2a}$
  - [More Katex Examples](#?d=gist:gera2ld/af76a4c245b302206b16aec503dbe07b:katex.md)
- Now we can wrap very very very very long text based on `maxWidth` option
```



6. Go diagrams

Create beautiful system diagrams with Go

```
d, err := diagram.New(diagram.Filename("app"), diagram.Label("App"), diagram.Direction("LR"))
if err != nil {
    log.Fatal(err)
}

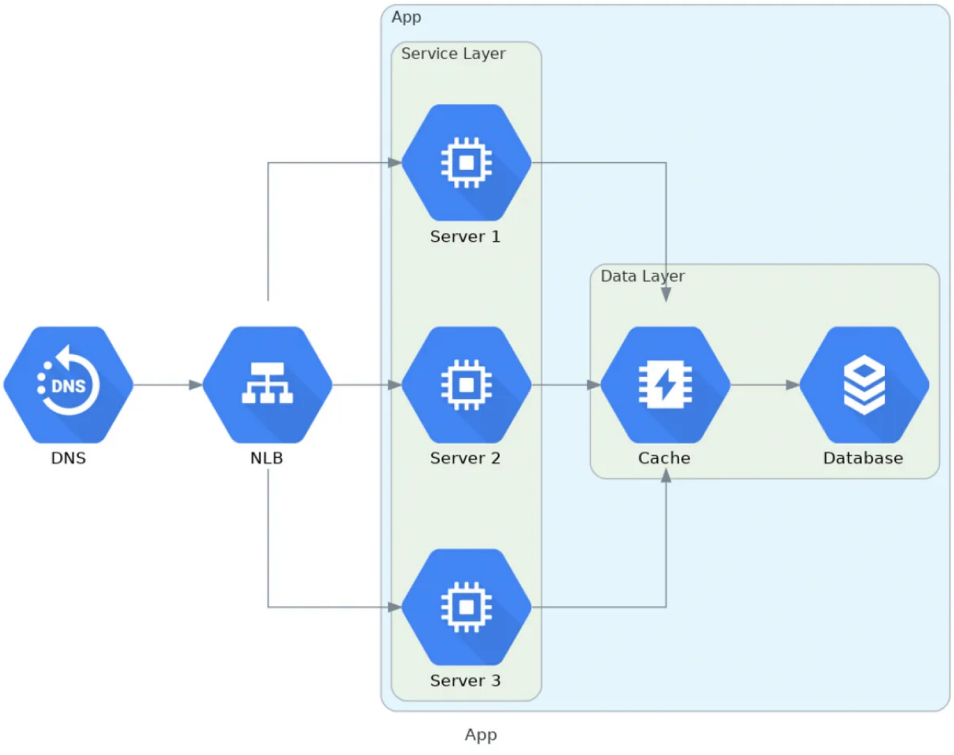
dns := gcp.Network.Dns(diagram.NodeLabel("DNS"))
lb := gcp.Network.LoadBalancing(diagram.NodeLabel("NLB"))
cache := gcp.Database.Memorystore(diagram.NodeLabel("Cache"))
db := gcp.Database.Sql(diagram.NodeLabel("Database"))

dc := diagram.NewGroup("GCP")
dc.NewGroup("services").
    Label("Service Layer").
    Add(
        gcp.Compute.ComputeEngine(diagram.NodeLabel("Server 1")),
        gcp.Compute.ComputeEngine(diagram.NodeLabel("Server 2")),
        gcp.Compute.ComputeEngine(diagram.NodeLabel("Server 3")),
    ).
    ConnectAllFrom(lb.ID(), diagram.Forward()).
    ConnectAllTo(cache.ID(), diagram.Forward())

dc.NewGroup("data").Label("Data Layer").Add(cache, db).Connect(cache, db)

d.Connect(dns, lb, diagram.Forward()).Group(dc)

if err := d.Render(); err != nil {
    log.Fatal(err)
}
```

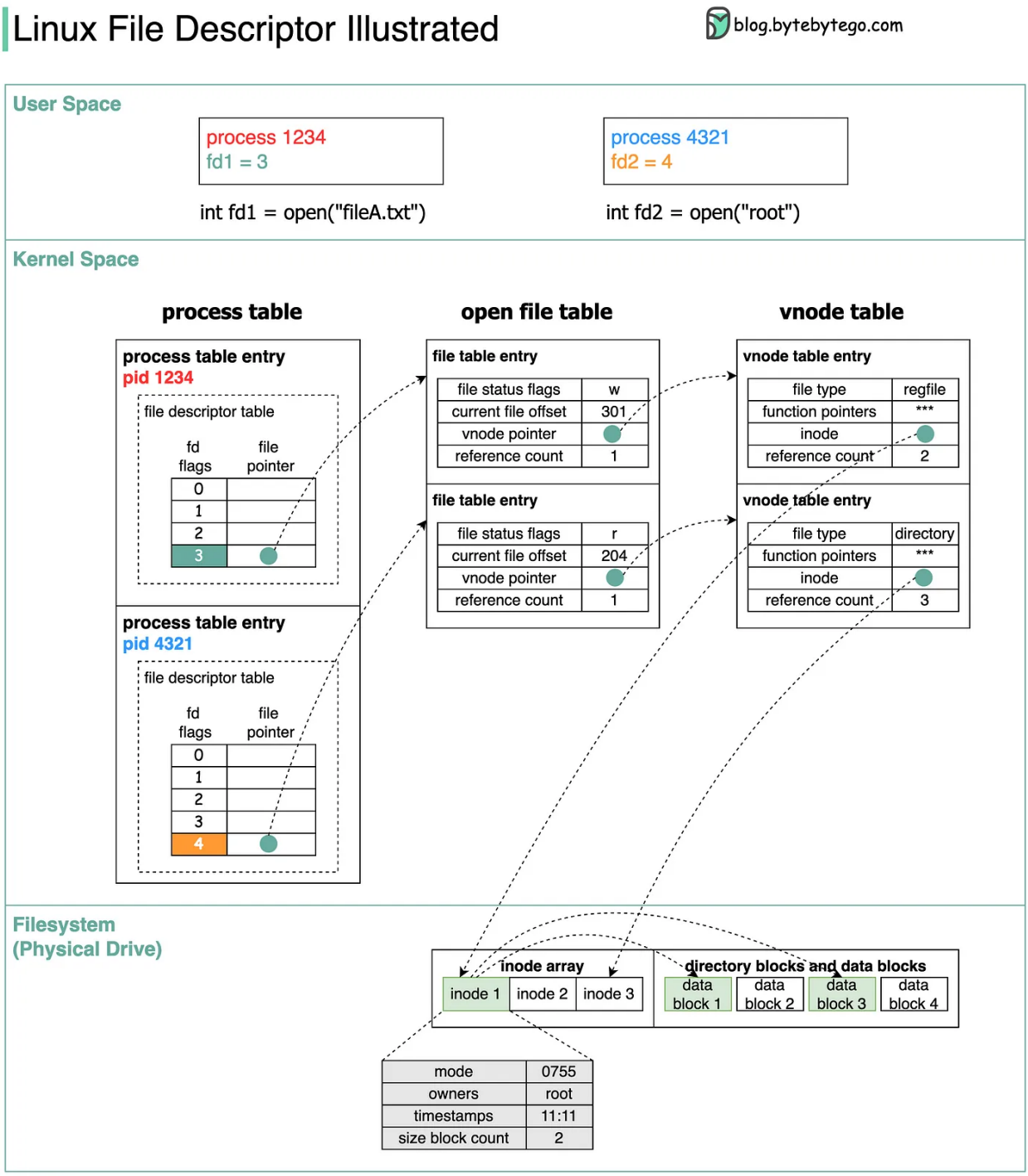


What is a File Descriptor?

How do we interact with Linux Filesystem via file descriptors?

A file descriptor represents an open file. It is a unique number assigned by the operating system to each file. It is an **abstraction** for working with files. We need to use file descriptors to read from or write to files in our program. Each process maintains its own file descriptor table.

The diagram below shows the layered architecture in Linux filesystem. Let's take process 1234 as an example.



- In User Space

When we open a file called “fileA.txt” in Process 1234, we get file descriptor fd1, which is equal to 3. We can then pass the file descriptor to other functions to write data to the file.

- In Kernel Space

In Linux kernel, there is a **process table** to maintain the data for the processes. Each process has an entry in the table. Each process maintains a file descriptor table, with

file descriptors as its indices. Notice that file descriptors 0,1 and 2 are reserved in each file descriptor table to represent stdin, stdout, and stderr.

The file pointer points to an entry in the **open file table**, which has information about open files across all processes. Multiple file descriptors can point to the same file table entry. For example, file descriptor 0, 1 and 2 points to the same open file table entry.

Since different open file table entries can represent the same file, it is a waste of resources to store the file static information so many times. We need another abstraction layer called 'vnode table' to store the static data.

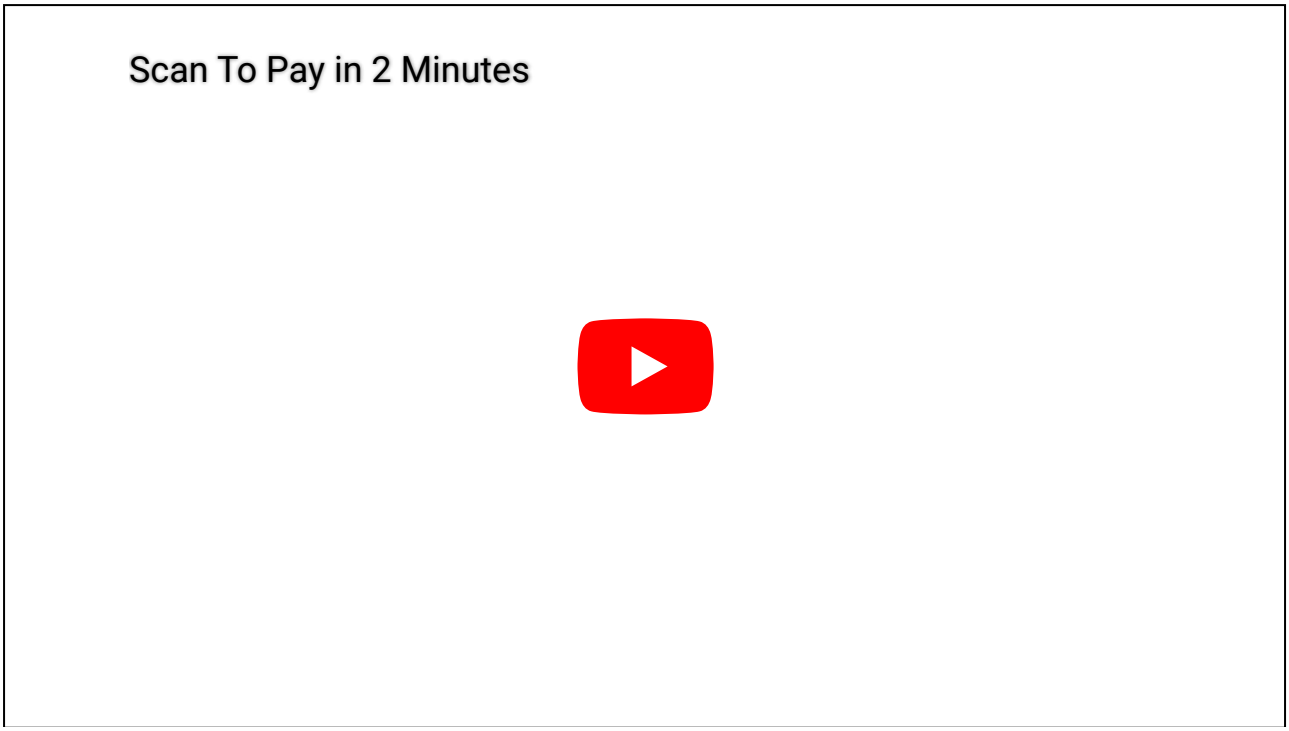
In each file table entry, there is a vnode pointer, which points to an entry in **vnode table**. The static information includes file type, function pointers, reference counts, inode etc. inode describes a physical object in the filesystem.

- In Filesystem

The inode array element stores the actual file information, including permission mode, owners, timestamps, etc. inode also points to the data blocks stored in the filesystem.

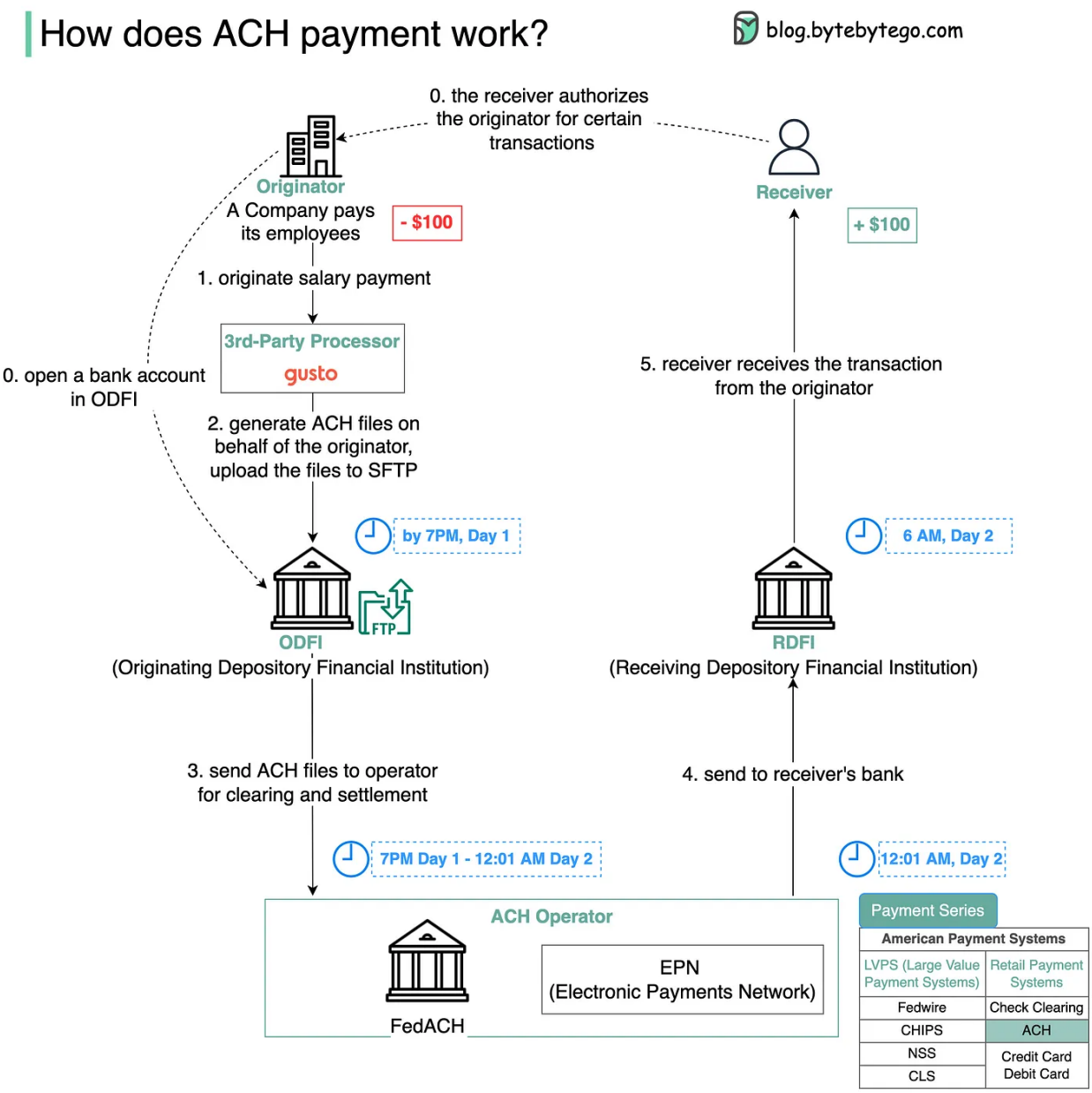
Over to you: When we close a file in a program, do you know which entries are deleted in these data structures?

Scan to pay in 2 minutes (YouTube Video)



Direct payments

Do you know how you get paid at work? In the US, tech companies usually run payrolls via Automatic Clearing House (**ACH**).



ACH handles retail transactions and is part of American retail payment systems. It processes transactions in **batches**, not in real-time. The diagram below shows how ACH direct deposit works with payrolls.

◆ Step 0: Before we can use ACH network, the originator who starts the transactions needs to open an account at a commercial bank because only banks are allowed to initiate ACH transactions directly. The bank is called ODFI (Originating Depository Financial Institution). Then the transaction receiver needs to authorize the originator for certain types of transactions.

🔹 Step 1: The originator company originates salary payment transactions. The transactions are sent to a 3rd-party processor like Gusto. The third-party processor helps with ACH-related services like generating ACH files, etc.

🔹 Step 2: The third-party processor generates ACH files on behalf of the originator. The files are uploaded to an SFTP established by the ODFI. This should be done by the 7 PM cut-off time, as specified by the ODFI bank.

🔹 Step 3: After normal business hours in the evening, the ODFI bank forwards the ACH files to the ACH operator for clearing and settlement. There are two ACH operators, one is the Federal Reserve (FedACH), and the other is EPN (Electronic Payment Network – which is operated by a private company).

🔹 Step 4: The ACH files are processed around midnight and made available to the receiving bank RDFI (Receiving Depository Financial Institution.)

🔹 Step 5: The RDFI operates on the receiver's bank accounts based on the instructions in the ACH files. In our case, the receiver receives $100 from the originator. This is done when the RDFI opens for business at 6 AM the next day.

ACH is a next-day settlement system. It means transactions sent out by 7 PM on one day will arrive the following morning.

Since 2018, it's possible to choose Same Day ACH so funds can be transferred on the same business day.

Over to you: ACH is a US financial network. If you live outside the US, do you know what payment method your employer uses to send payment? What's the difference between ACH and wire transfer?

Thanks for making it this far! 🤗

If you want to learn more about System Design, check out our books:

Paperback edition: https://geni.us/XxCd

Digital edition: https://bit.ly/3lg41jK

Thanks for reading ByteByteGo Newsletter!
Subscribe for free to receive new posts and
support my work.

227 Likes

7 Comments

Write a comment...

Tarek Elias Jul 30, 2022

Great job on the design patterns cheat sheet Mr. Alex, thank you so much!

It would be great if these patterns came with a real-world example, so we can see how each one could be implemented in a real-world application.

LIKE (13) REPLY SHARE

Anos Mhazo Aug 1, 2022

Nice!!!! Thank you for putting so much effort into making this!

LIKE (2) REPLY SHARE

5 more comments...