SE 577
Software Architecture

Web and API Architecture
also
Includes Blockchain Architectures
which are a part of Web 3.0

The Architecture of the Web – How to think about this material

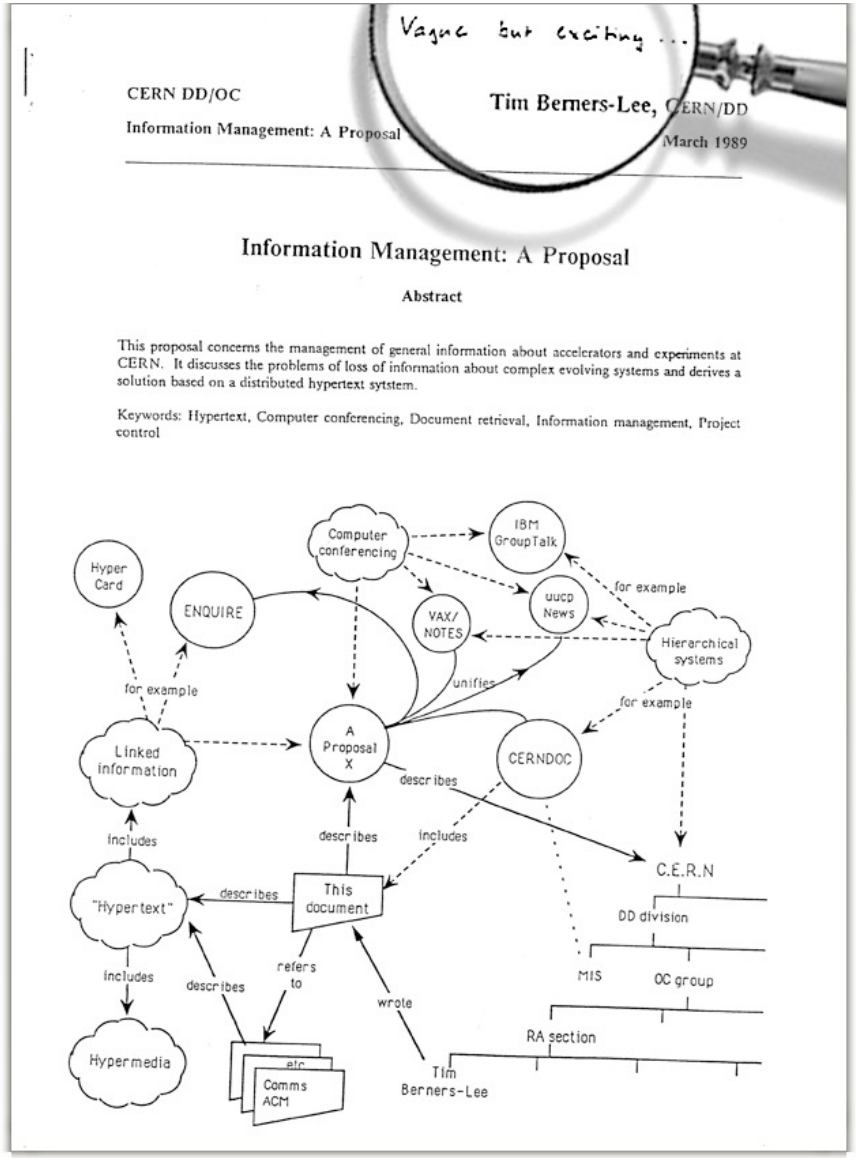
- While the first part of this lecture may appear pretty basic from looking at the slides, its important to understand the "why" we are covering it
- The evolution of web is one of the best examples of architecture done right
- While the web itself is credited to being invented in the early 1990s, the web itself rides on inventions going back to the 1960s with TCP/IP
- Its remarkable about how the web we know today largely evolved from pioneering work that started in the 1960s
- We will examine the Web Architecture across 3 generations – Web1, Web2, and the emerging Web3

The “Father” of the internet – Vint Cerf

My conversation with Vint Cerf over Zoom in Sept 2021

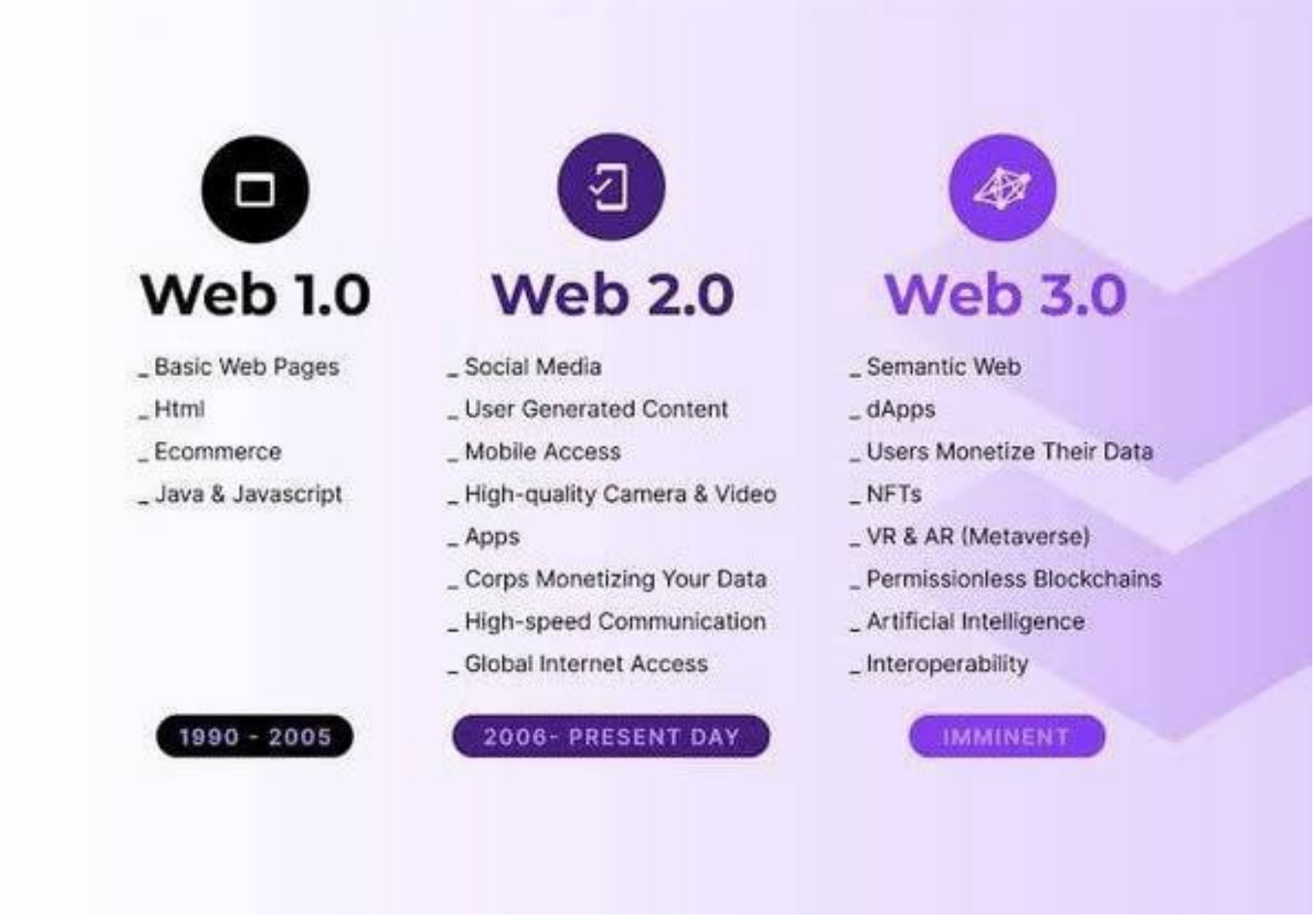
- In September 2021 I had the unbelievable privilege to talk with Vint Cerf for an hour
- Vint is recognized as one of the fathers of the internet for co-inventing TCP/IP and a lot of the foundational technologies that enabled the web going back to the early 1960s
- For those who don’t know the Vint story, he always wears three-piece suits ☺

History of the Web – circa 1989/1990

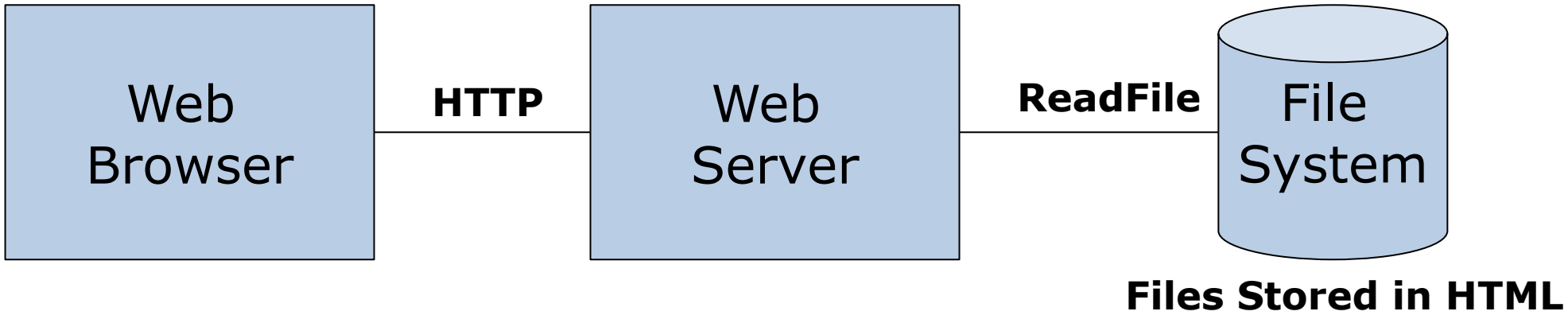


- In March 1989, Tim Berners-Lee wrote a proposal that outlined the architecture of the web
- Initial proposal was considered vague and not well received, it was revised and eventually accepted in October 1990
- Introduced the concepts we know today:
  - HTML: Formatting/Markup
  - URI: Addressing approach (now generally called URL)
  - HTTP: The protocol for encoding and transmitting information over the web

Web 1.0, 2.0, and perhaps soon 3.0



Web 1.0 – The "Read Only Web" Architecture



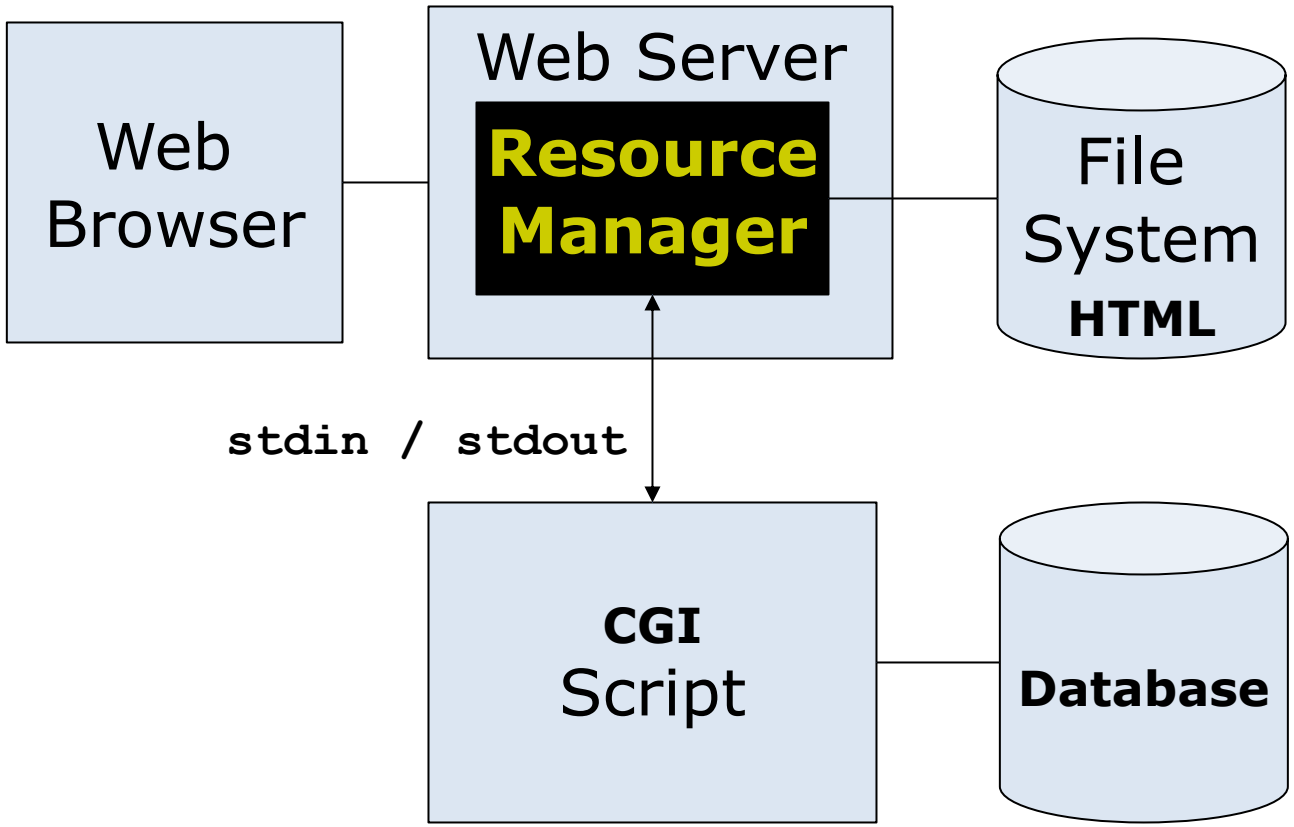
Advantages

- Simple
- Cachable
- Indexable

Disadvantages

- No dynamic content
- No interaction with user
- All requests must go back to server

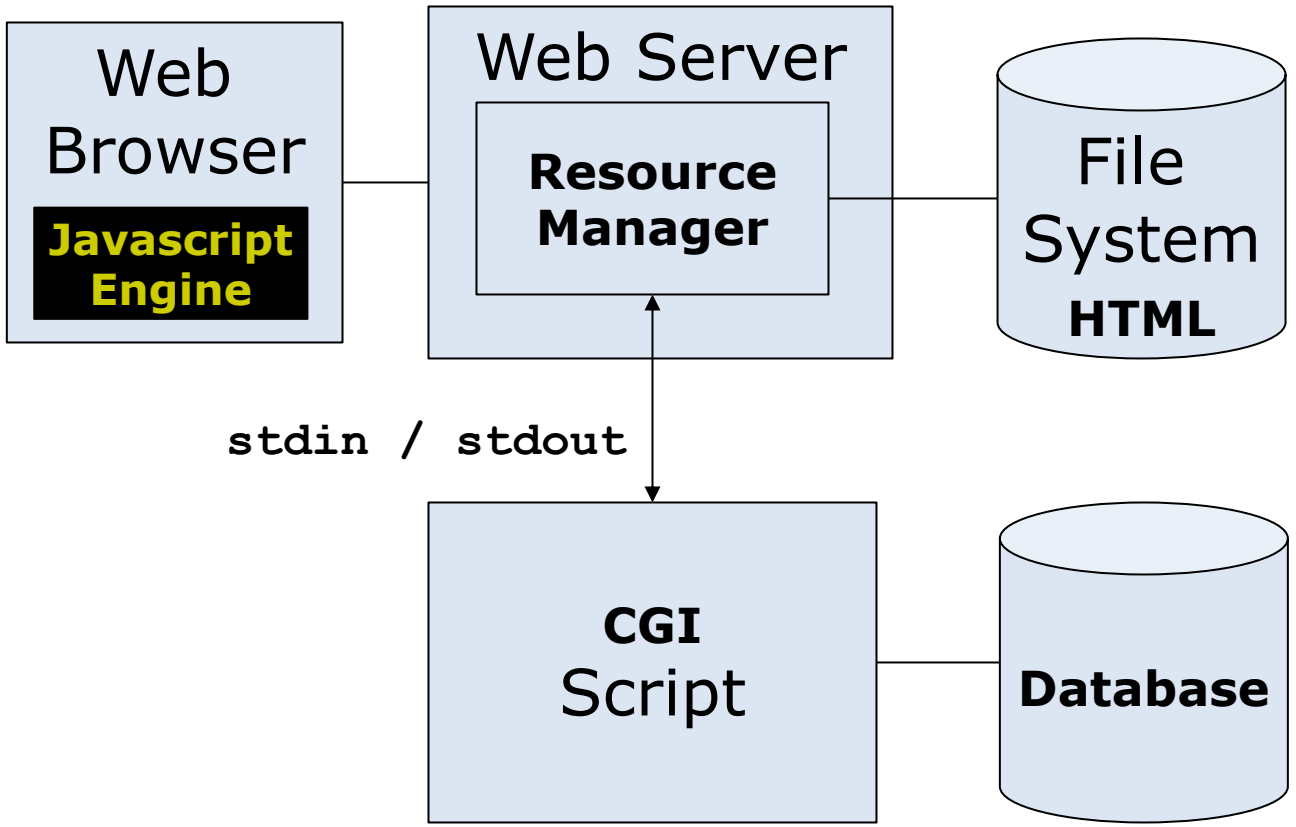
Web 1.0.1 – The Common Gateway Interface (CGI) Circa 1993



Architecture Changes

- Web server acts as router
  - .html files returned from filesystem
  - .cgi extension resulted in web server executing a new process that piped information via stdin and stdout
  - CGI programs generated HTML directly
- Enabled the web to start to support transactional personalized experiences
- Since CGIs are just programs, they were often developed in languages like C and/or Perl
- Architectural challenges
  - Scale – all CGI ran as a process
  - Interactivity, all actions from the browser had to travel over the network to the server

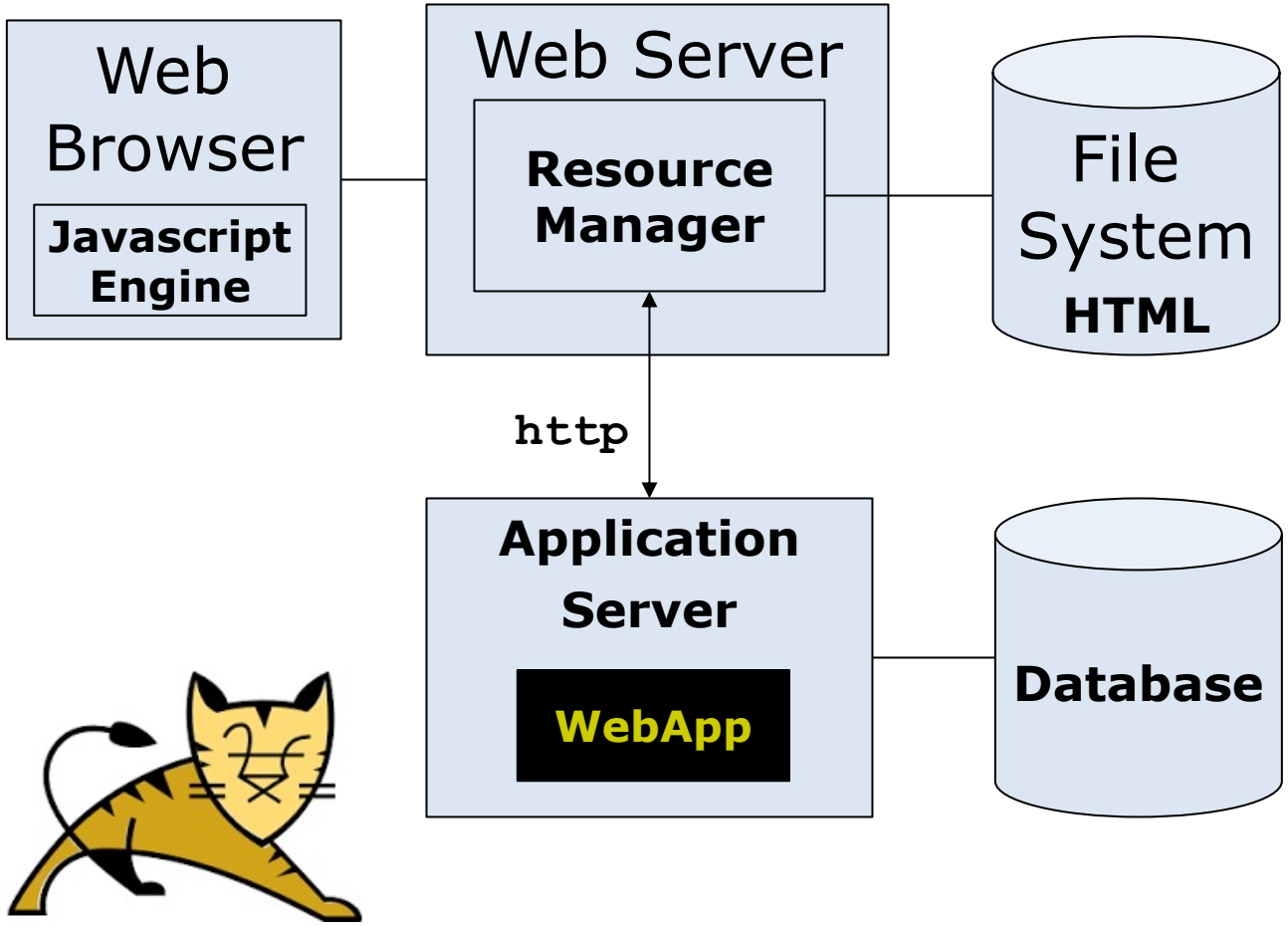
Web 1.0.2 – Javascript Circa 1995



Architecture Changes

- Web Browser gets Smarter
  - HTML can now embed or link to javascript that can be executed on the browser
  - Code running directly in browser improves user experience
- Enabled some user behavior to be processed in browser avoiding trips back to the server
- Architectural challenges
  - Same as 1.0.1, plus,
  - Code redundancy, often checks had to be made both in the client (javascript) and the server (CGI)
  - Javascript was developed in 10 days and has always been regarded as a poor language
  - Javascript compatibility was very poor for the first 20 years of its existence leading to workaround solutions like jQuery becoming popular

Web 1.0.3 – The Application Server - Circa 1999

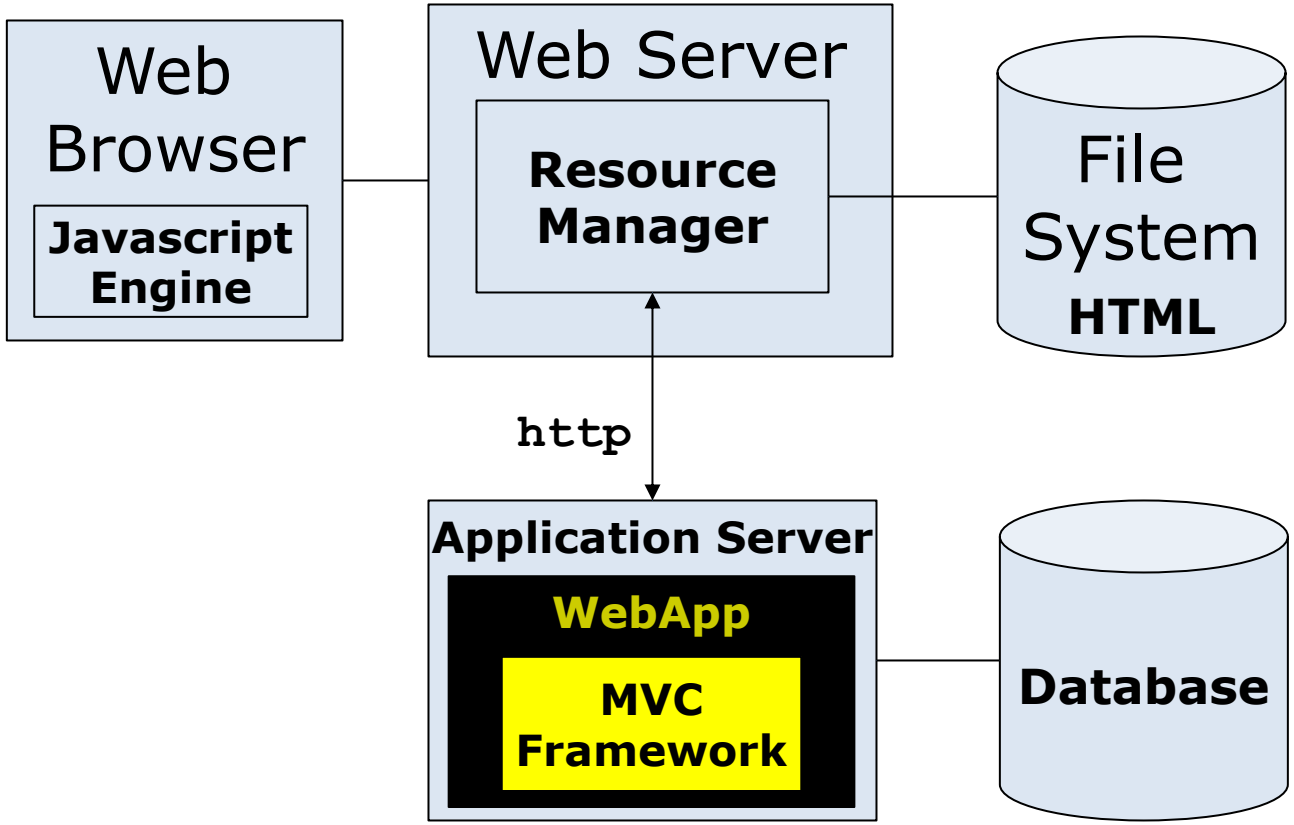


Apache Tomcat 1.0

Architecture Changes

- The CGI handler was replaced with a totally new architecture
  - CGIs were processed based
  - CGIs had to communicate via stdin and stdout to a local file system
- Application servers replaced CGI with interfaces that streamed HTTP over a network protocol (could be localhost, but could also be anywhere)
- Application Server managed the lifecyle of applications – called webapps
- Webapps could be run in threads instead of processes
- Libraries were created that abstracted HTTP and HTML from developers

Web 1.0.4 – The Application Framework - Circa 2000-2003



Architecture Changes

- Previous generation provided code libraries to help translate code data structures to HTML
- Improvements with MVC
  - Server side code could be much better modularized, which supported creating larger applications
  - HTML rendering code replaced with markup files for the view that can be translated into code and pre-compiled for speed
  - Behavior can be altered via configuration
- Challenges
  - Configuration files that controlled code execution became complex and difficult to maintain (not to mention debug)

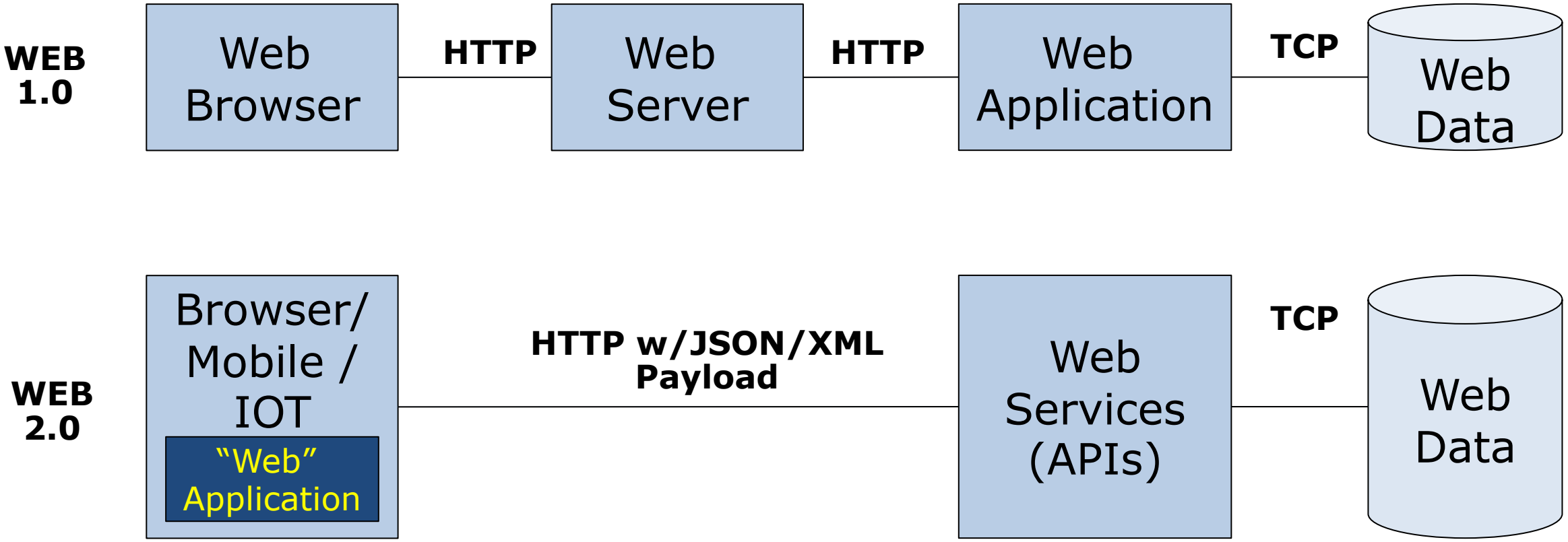
Spring MVC (2003)

Apache Struts (2000)

Web 1.0 Summary (from an architecture perspective)

- Web 1.0 existed for about 15 years between 1990 and 2005
- It started out supporting read-only content
- It evolved to supporting reasonable applications, that could run at reasonable scale over the basic web architecture
- The underlying architecture changed very little over this period, mainly supporting enhancements that allowed content-rich applications to be developed using oriented technologies
- Introduced the MVC pattern as a best practice for developing web-centric applications

Web 2.0 – Circa 2005/2006



The primary architecture shift in web 2.0 is that the application itself moves to running on the client and the client types expand from just a browser to mobile and IoT

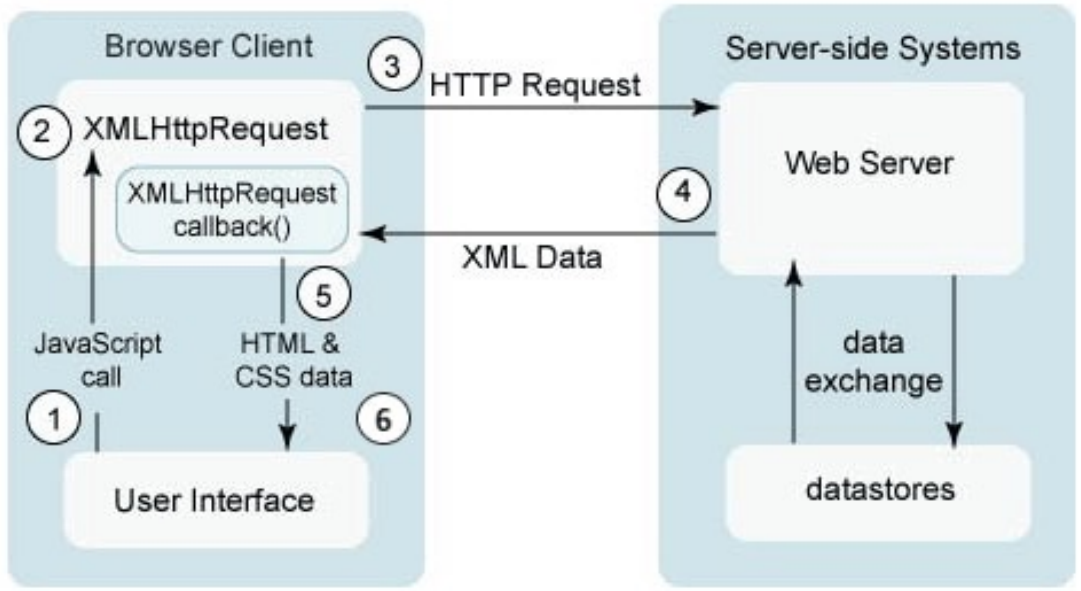
Web 2.0 starts by exploiting a little known feature called XHTR (XML Http Request)

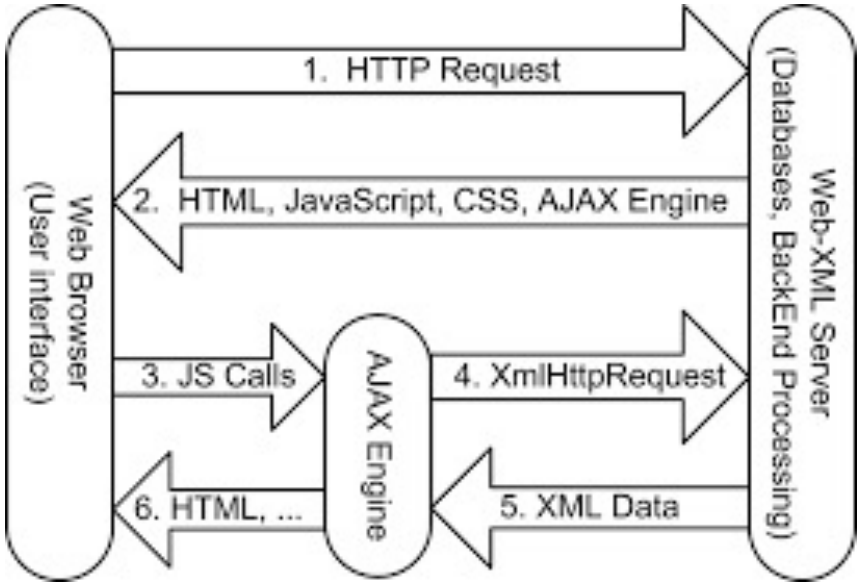
- In 1998, Microsoft developed the concept behind XHTR to support a web-based mail program which was first shipped with IE5 in 1999
- This “feature” was inconsistently implemented in a variety or browsers over the early years
- In 2004, Google saw the promise of XHTR and creaed a browser compatible javascript library that they needed to support a new generation of applications they were creating – Gmail in 2004, and Google Maps in 2005
- In 2006, XHTR was released as a standard by the W3C - https://www.w3.org/TR/2006/WD-XMLHttpRequest-20060405/

Like most interesting things, the enabler for web 2.0 was introduced for a totally different purpose and then later exploited to enable things we use today

What does XHTR (and AJAX) enable along with its architecture?

- Terminology, XHTR is often talked about with something else called AJAX – Ajax stands for asynchronous javascript and XML
- AJAX is the API component that the browser uses, XHTR is the structure of the object that interacts with servers over HTTP



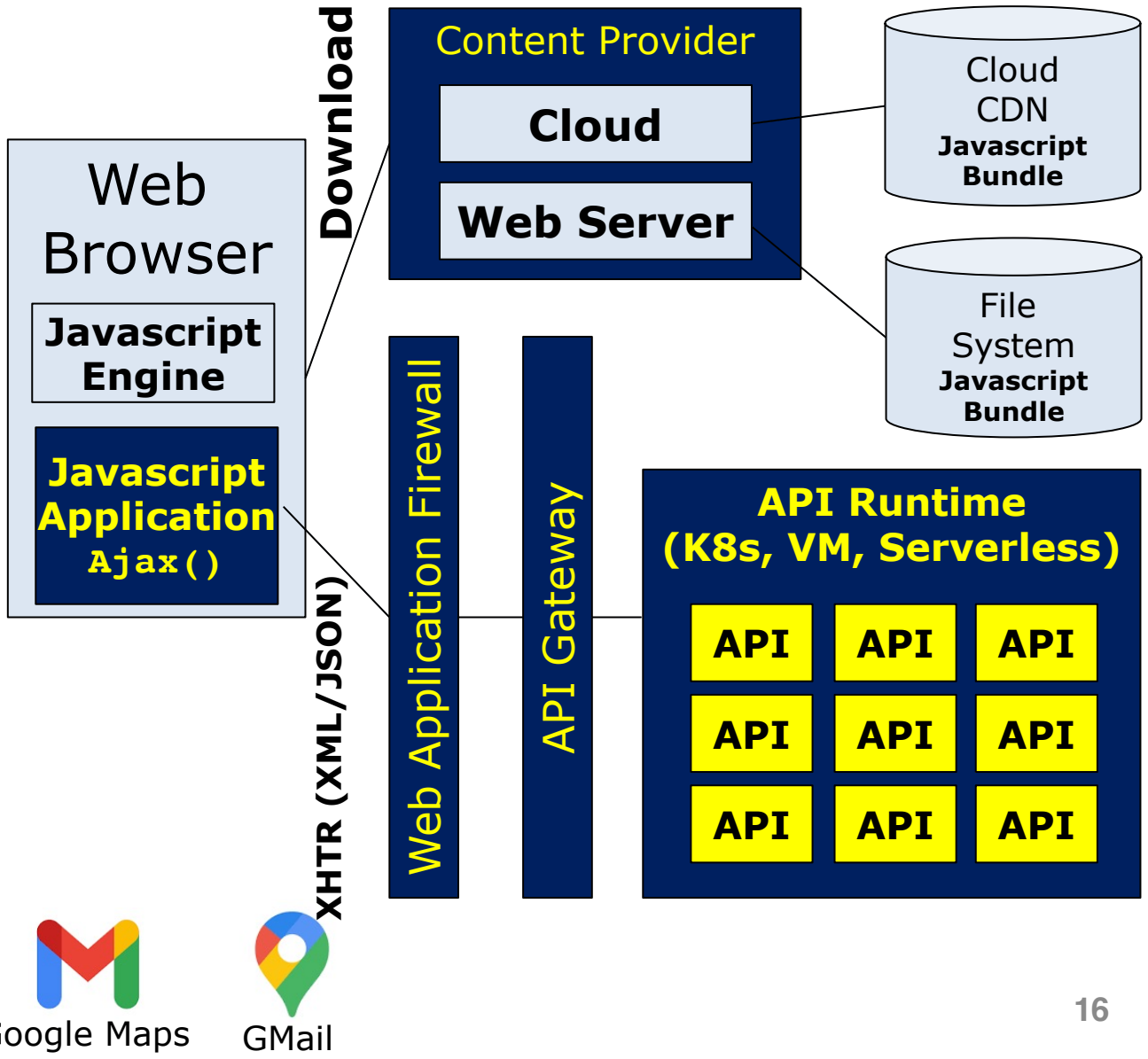


Capabilities enabled by the Web 2.0 Architecture

- Web 1.0 was about the evolution from static content to dynamic content
- Web 2.0 is about running fully featured applications over the web protocols
  - Multi-Client – not just web browsers but mobile and IoT devices
- Web 2.0 also brought with it extensions to the underlying HTTP protocol
  - HTTP/2.0 – Switch from non-persistent to persistent connections; Switch from text to binary payloads
  - MQTT – A queue based lightweight, low-energy protocol suitable to support fleets of IoT devices running embedded web 2.0 applications

Most basically stated –
Web 2.0 is about being able to do useful things over the Web

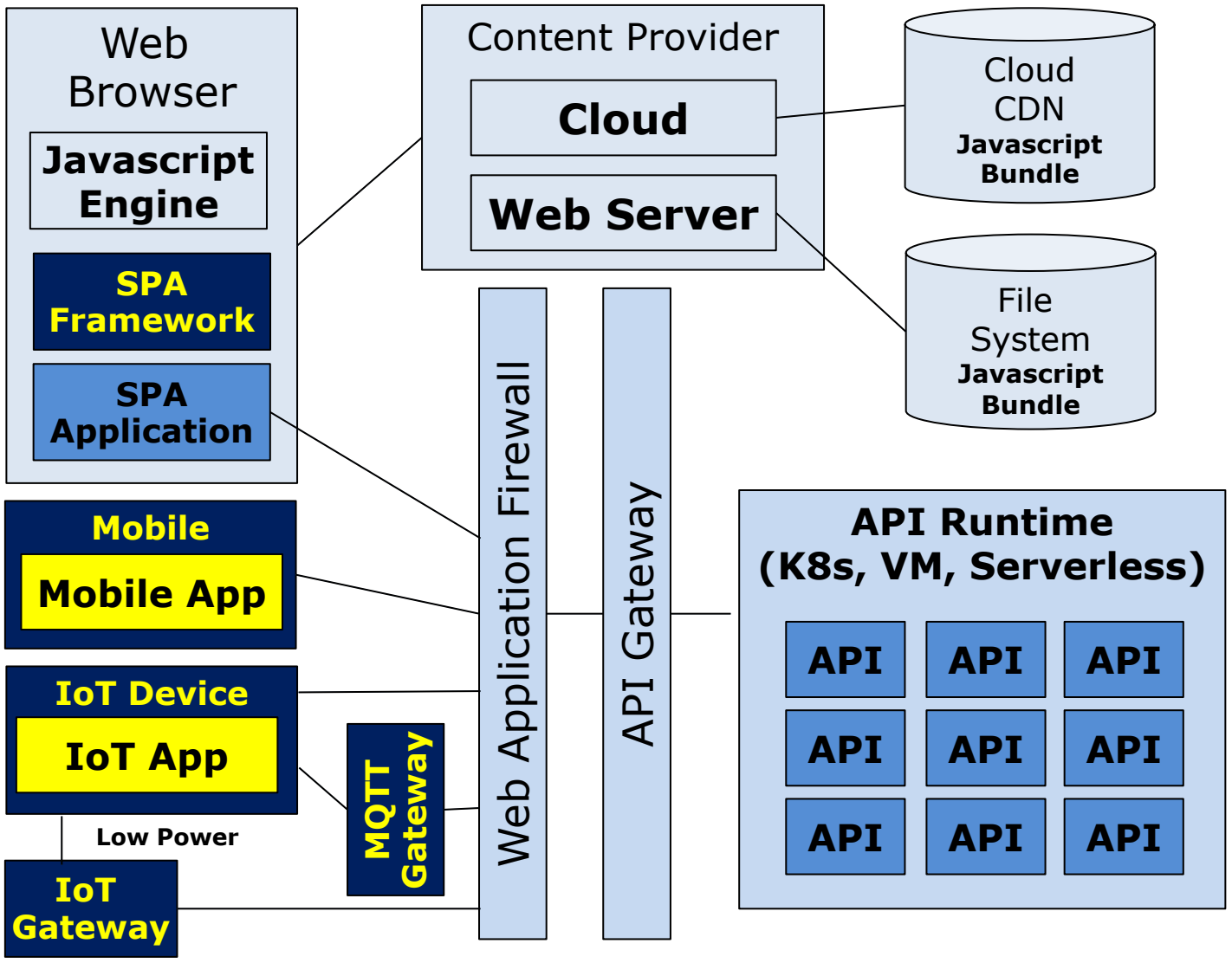
Web 2.0 – Circa 2005



Architecture Changes

- Introduction of multiple new components
  - Content provider: Browser downloads application code either via a web server or a content delivery network
  - Once application is downloaded, it is bootstrapped and executed 100% in browser
  - Application makes requests over HTTP to APIs using Ajax
  - To manage scale and security, purpose built proxies are deployed; a WAF for security, and an API gateway to manage the APIs
- Challenges
  - Javascript is complex to create and maintain these types of applications

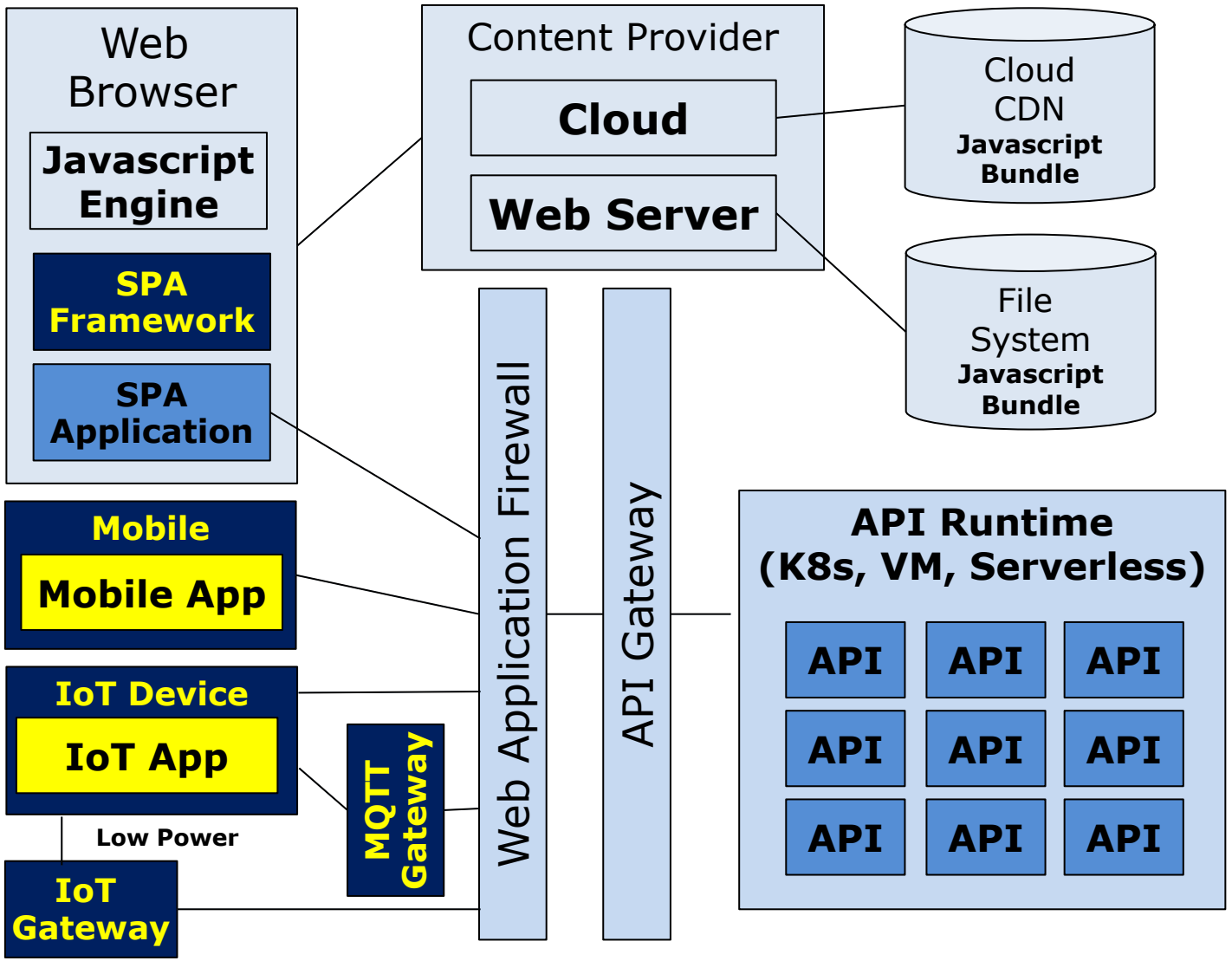
Web 2.x – Circa 2008-today

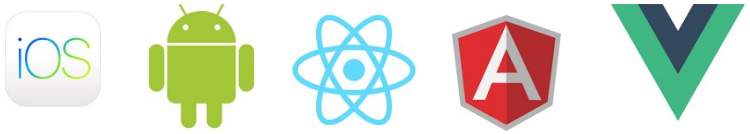


Architecture Changes

- Introduction of multiple new components
  - Single Page Application (SPA) Frameworks mature to make developing modern web applications significantly easier
  - Mobile Apps arise with nice SDKs that reuse the web platform for data exchange
  - IoT devices become very popular – IoT devices have varying capabilities and power thus multiple integration paths are supported – direct HTTP/JSON, MQTT over TCP, or specialized low power protocols such as Z-Wave, ZigBee, Bluetooth, etc

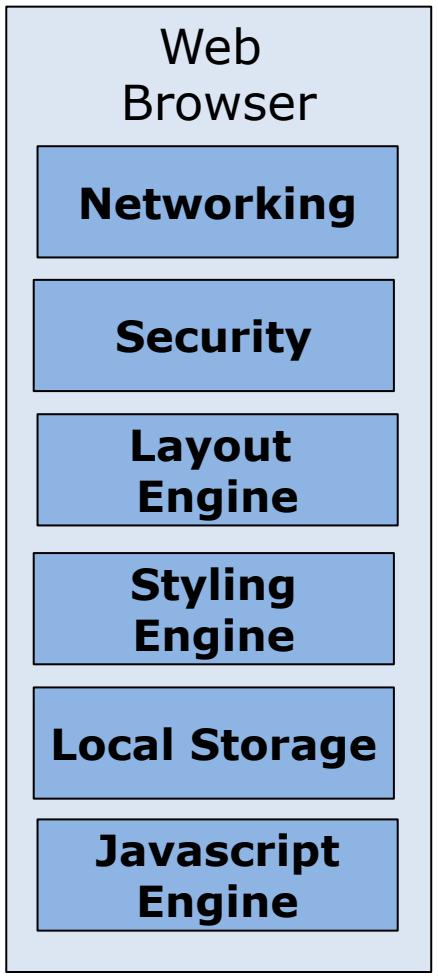
Web 2.x Summary

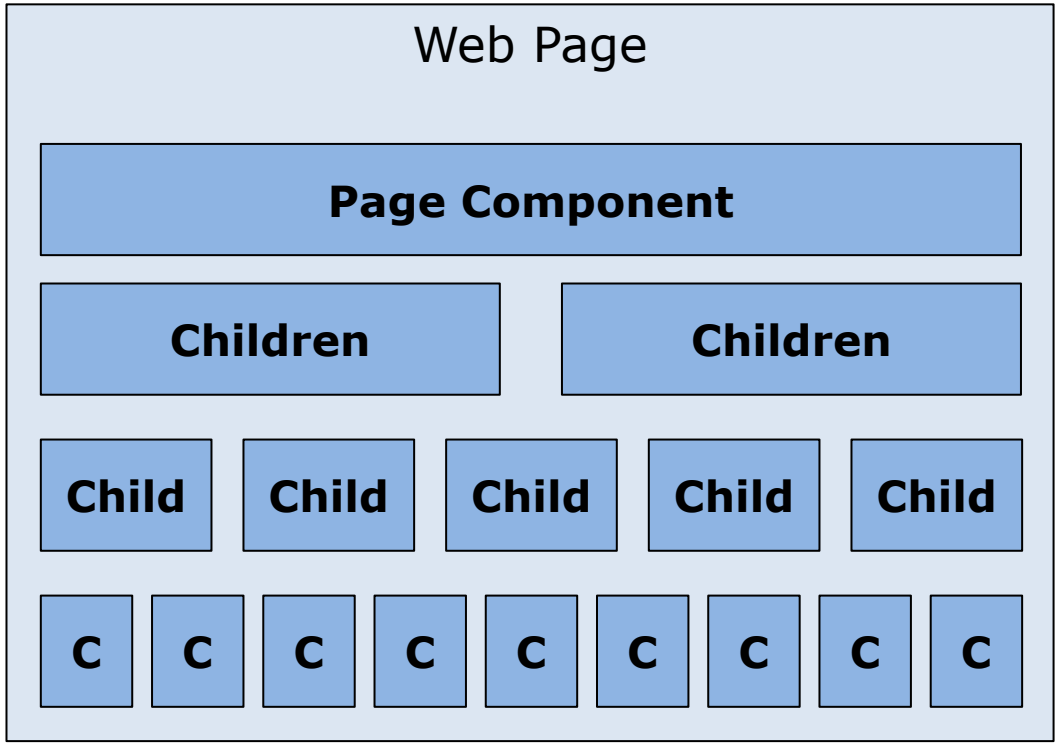




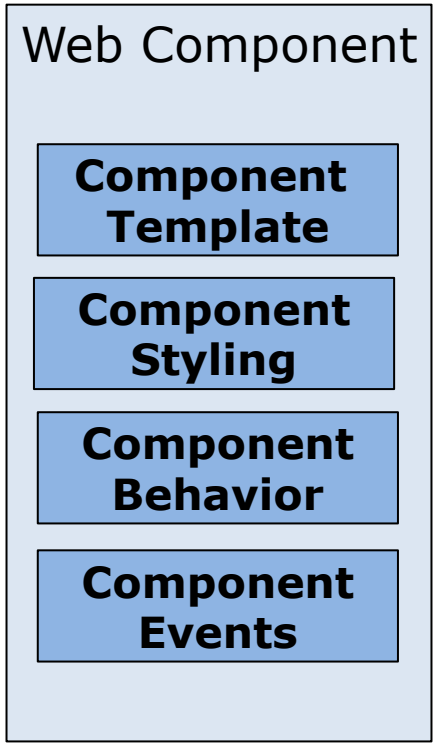
- For all practical purposes this is the current state of the web
  - Web 1.0 was about delivering rich content
  - Web 2.0 is about using the web architecture to do useful things
- Architecture Pivots
  - Application code delivered to client remotely – benefits, easy patching and updates
  - Application runs locally on client – provides limitless scale
  - Data and remote functions provided by APIs – numerous architectural innovations here as well (more to come on this topic)

Single Page Application Architecture - SPA





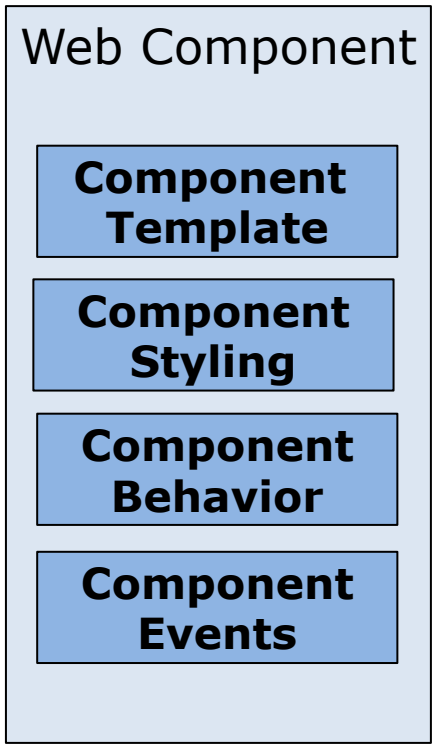
Kind of Trivial but Web Pages can be thought of as a hierarchy of components, where components nest, the leafs of the webpage are native controls such as entry fields, static text, buttons, etc

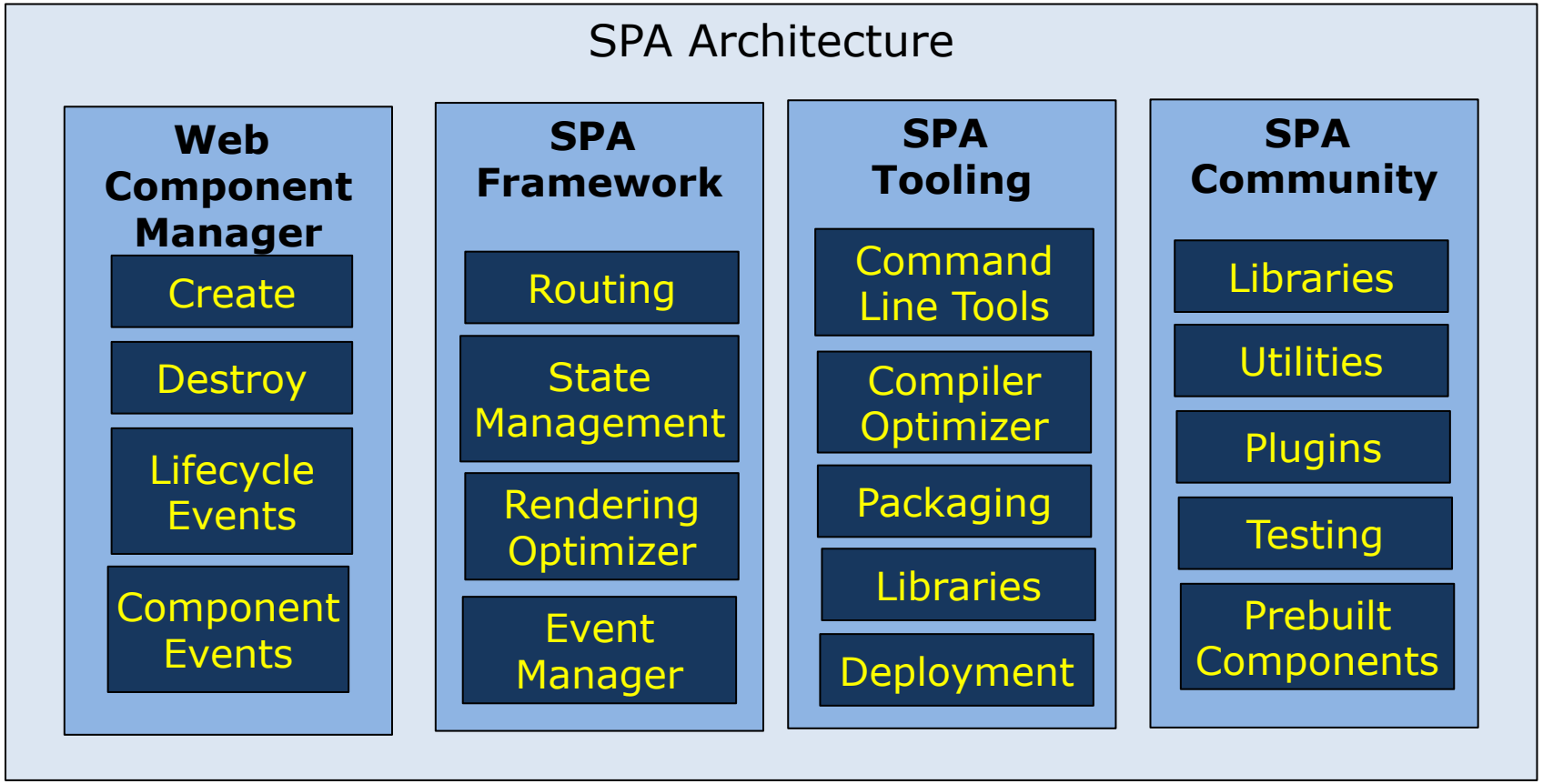


Web Components in an SPA consist of a template for layout, styling for look/feel, Javascript/Typescript code to manage behavior, and an eventing framework to communicate with other web components

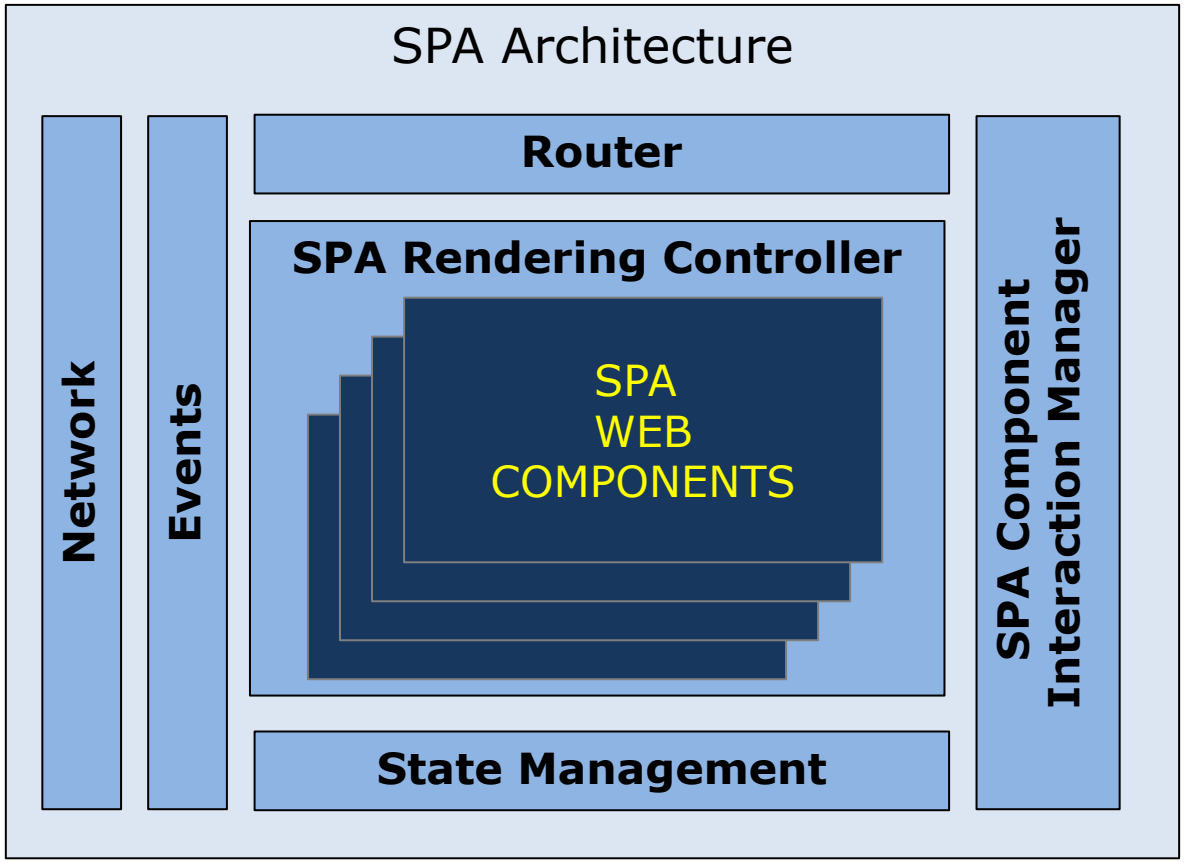
SPA Frameworks

Building SPAs by hand can be very complicated, over the years multiple frameworks and libraries have been introduced to make this easier. Top frameworks are React, Vue, Angular, and most recently Svelte.





SPA Frameworks



SPA Components are **event-driven**, almost all interesting interactions happen over sending and reacting to asynchronous events.

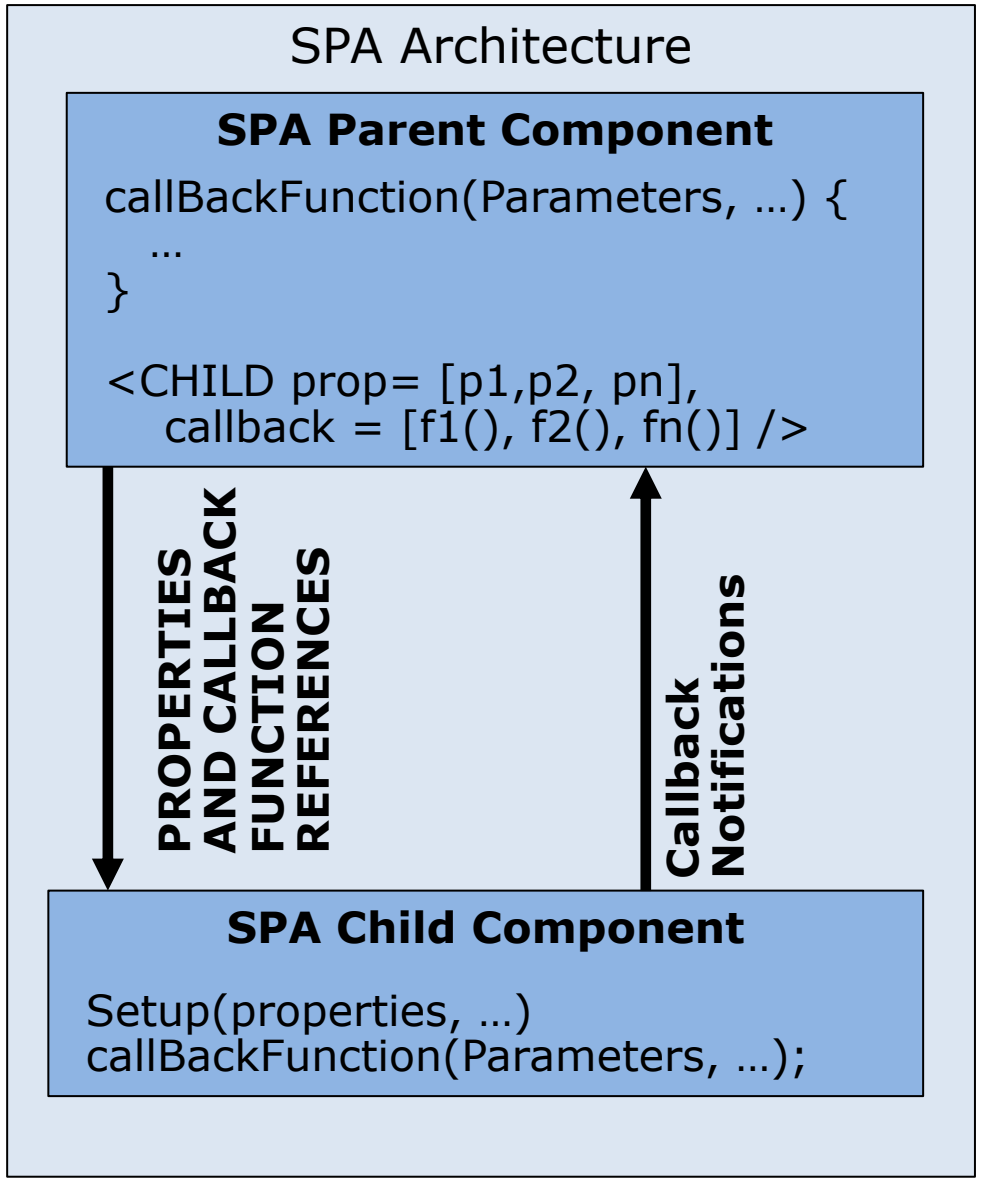
SPA architectures often have abstractions for the **network** interfaces that treat ingress and egress network interactions as events

State management for SPAs (discussed soon) is often facilitated via variations to the flux pattern. This pattern features state being managed in an immutable store, and uses events to query, mutate or react to changes in the store

User interface updates are handled entirely locally, they can be full page swaps, which are driven by the **router**, or they can be incremental page updates based on reacting to user behavior (clicking a button, pressing a key, etc)

SPA Web Components, via an **interaction manager,** can communicate with each other via sending events, and reacting to callbacks.

SPA Component Interaction Best Practices



Remember, SPAs create hierarchies of components

There are strict parent/child relationships

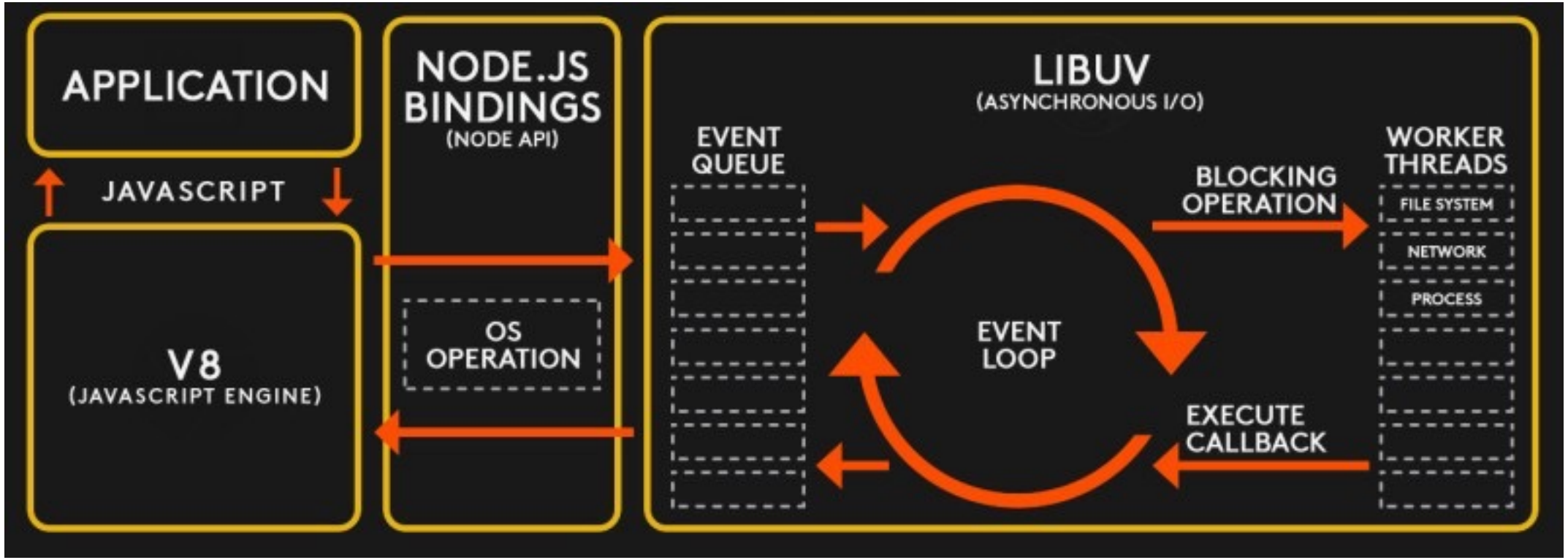
Best practice, when a parent creates a child, it passes all properties that that child needs. These take the form of:

- Properties, or props – think of these as a web component constructor, the parent passes data to the child for initialization
- Callback functions – as the child component runs, sometimes it needs to notify the parent about interesting things, such as state changes or anything else of interest. The parent passes a callback function to the child, that the chid can use to interact with the parent

SPAs are very strict on flow of direction, properties flow down, callbacks / events flow up

The architecture of Javascript, and more importantly its runtime

The javascript runtime is single threaded with an event loop at the center, effective use of asychonous programming techniques is essential to making these applications perform well.



Because most web architectures are I/O bound this model works well, but specific strategies are required for compute bound workloads in order to ensure that work is processed efficiently.

The architecture of Javascript, and more importantly its runtime

The javascript programming model provides multiple models for doing async programming

```
function after(args) {
  //post processing here
}

function do_something(args, cb{
  //do some work
  cb(args)
}

function client() {
  do_something(”hello”, after)
}
```

Callback example – do some work let me know when you are done

```
var promise = new Promise(function(resolve, reject)
{
  // do a thing, possibly async, then…

  if (/* everything turned out fine */) {
    resolve("Stuff worked!");
  }
  else {
    reject(Error("It broke"));
  }
});

//calling it
promise.then(function(result) {
  console.log(result); // "Stuff worked!"
}, function(err) {
  console.log(err); // Error: "It broke"
});
```

Async with Promises

The architecture of Javascript, and more importantly its runtime

The javascript programming model provides multiple models for doing async programming

```
async function f() {
  return 1;
}

//async functions return promises
f().then((v) => console.log(v)
```

Async functions return promises so you can just code them as such

```
async function f() {
  return 1;
}

let v = await f()
console.log(v)
```

You can use await to make async code look synchronous

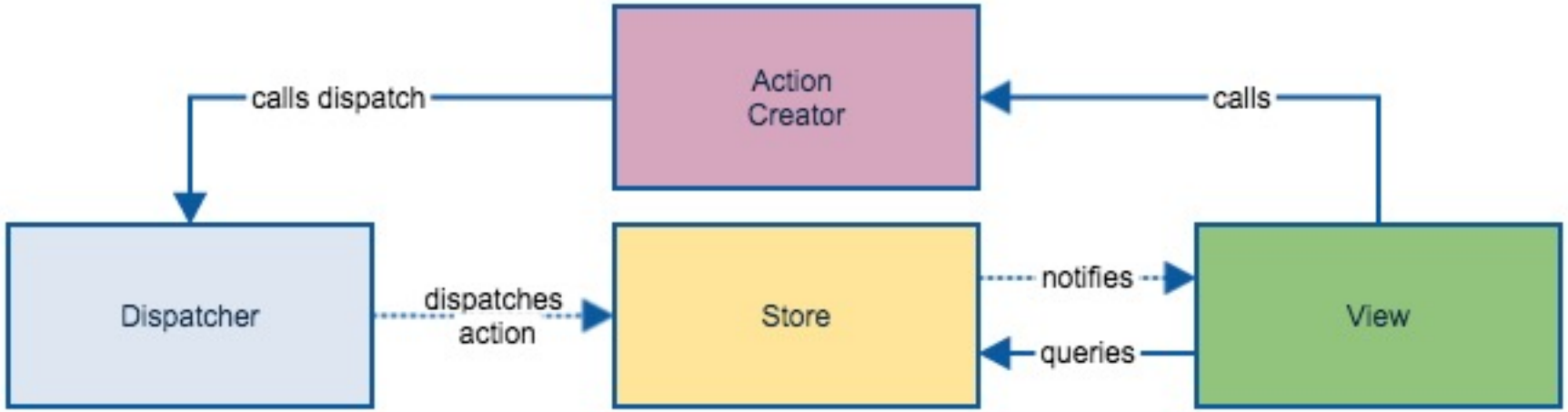
```
async function worker() {
  data = await call_database();
  status = await web_service_call()
  return status
}


//calling it
let s = worker()
console.log(s);
```

Using async/await together, notice how this async code looks sync

Note that javascfipt also has a future called a generator that is used to break up compute intensive work, if you have interest in this please ask a question.

Now we know enough to look at how state management is handled in SPAs

The best architecture pattern for handling state is called flux
https://facebook.github.io/flux/



The key to this pattern is that all information flows in one direction and all interactions between the components are carried over events

Note that flux is a pattern and there are various implementations of this pattern that have been encapsulated in libraries – Redux, Vuex, Pinia

Pinia state management example

```
import { defineStore } from 'pinia'

export const useCounterStore = defineStore({
    id: 'counter',
    state: () => ({ counter: 0 }),
    getters: { doubleCount: (state) => state.counter * 2 },
    actions: {
      increment() { this.counter++ } ,
      async callWebService() { http.get(...)}
    }
})

//using in your components
const counterStore = useCounterStore()
counterStore.inrement()
counterStore.callWebService()
console.log(counterStore.counter)
Console.log(counterStore.doubleCount)
let {counter} = storeToRefs(counterStore) //reactive counters reacts to changes
```

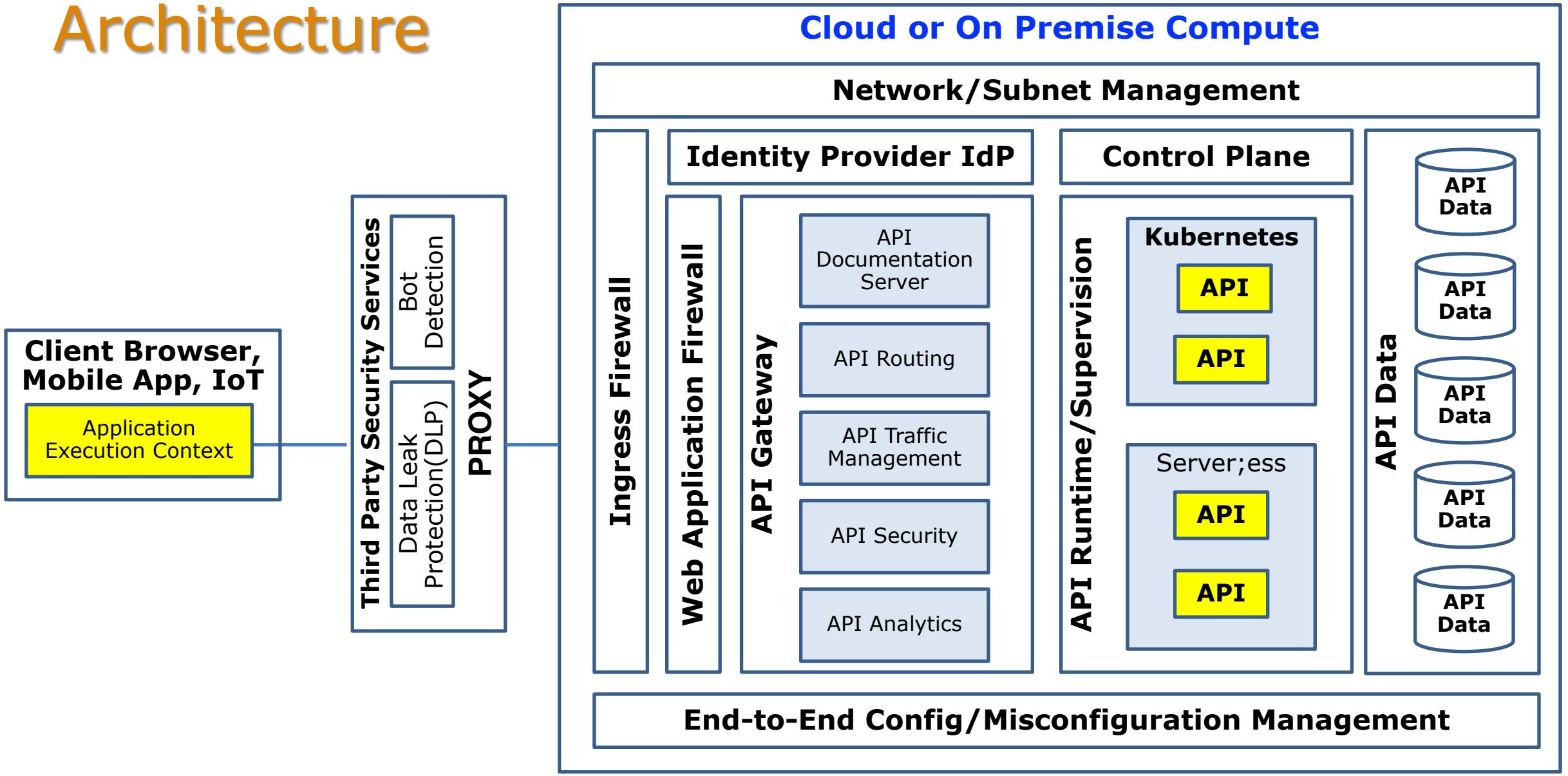
Store setup, define the state properties and initial values

Calculated properties State not mutated

Action that mutates state – notice this keyword

Action can be async as well

Wrapping up Web2 – Overall Reference Architecture



Wrapping up Web2 – Overall Reference Architecture – Component Description

Component	Description
Application Execution (context)	Web 2.0 applications generally execute directly on the client. The client can be a web browser, a mobile device, or an IoT device. The most popular approach is to use single page web application (SPA) technology frameworks such as Angular, React, Vue or Svilte on the client. Emerging technology is web assembly
Data Leak Protection	Often offered by a third party that runs as a hosted service. This component observes all ingress and egress traffic looking for suspicious traffic being sent to destination locations. The objective is to prevent data from leaking.
Bot Detection	With the dark web, and the tendency to reuse passwords, Bots hit various websites using known credentials to try to break into other sites. This is known as credential stuffing. Bot detection solutions apply AI techniques to determine if security/login endpoints are being interacted with by a human or a BOT. Captua technology is a common defense. Shape security is an example provider of such services.
Ingress Firewall	The ingress firewall manages network connections at the edge. They do a varity of things including blocking unwanted TCP/IP ports. Its common for these firewalls to block anything outside of port 443 (HTTPS)

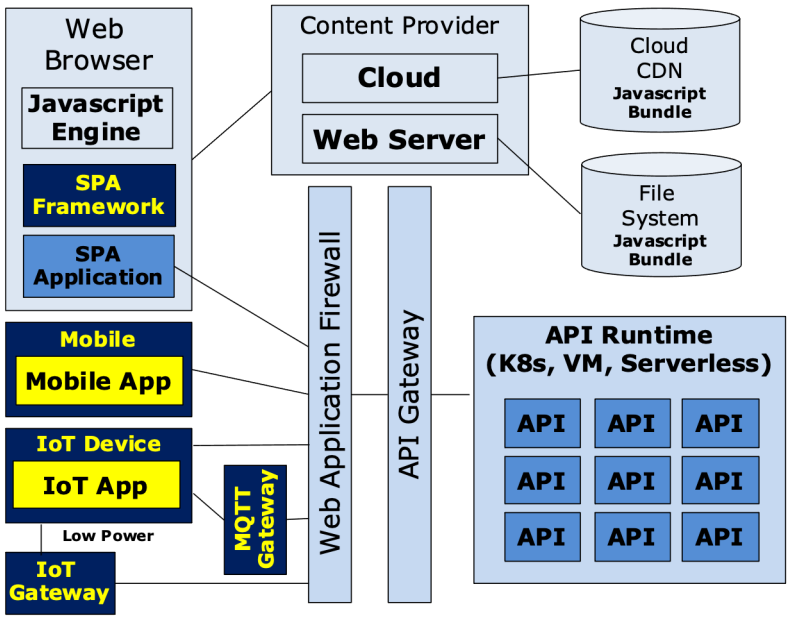
Wrapping up Web2 – Overall Reference Architecture – Component Description

Component	Description
Web Application Gateway (WAF)	The WAF is aware of L7 protocols such as HTTPS and can apply rules to block certain types of attacks. Attacks addressed by WAFs include SQL Injection, Cross Site Scripting, Ddos, Cookie based attacks, etc. Examples: AWS WAF, F5
Data Leak Protection	Often offered by a third party that runs as a hosted service. This component observes all ingress and egress traffic looking for suspicious traffic being sent to destination locations. The objective is to prevent data from leaking.
Identity Provider (IdP)	The identity provider is a security component that handles authentication and authorization services. It can act as a reverse proxy ensuring clients are using secure sessions, or support protocols such as oAuth and issue and validate tokens either in a proxy mode or by providing security signing services to validate tokens. Examples: Okta
Network Subnet Management	These services can be offered via physical network design, or logical network design in the cloud. They can be facilitated by switches, firewalls or cloud based network access controls. The purpose of this component is to ensure that different services are isolated into different subnets and that clear rules are executed that specify what traffic can cross subnet boundaries.

Wrapping up Web2 – Overall Reference Architecture – Component Description

Component	Description
End to End Configuration Management	Ofen deployed in the cloud, these services constantly watch infrastructure and ensure that the deployed configuration manages the desired configuration. This includes what components are running along with their configuration. Example: Cloud Custodian
API Gateway	The API gateway is a critical piece of infrastructure to manage, as it name states, APIs. API gateways include services to manage APIs, to manage access to APIs, to document API interfaces and to manage traffic policies for APIs (e.g., rate limiting, circuit breakers, etc). Examples: Gloo Edge, Envoy, Kong
API Runtime Supervision / API Control Plane	As APIs execute, they benefit by being supervised by a component that manages their availability and scalability – being able to scale up, scale down, detect and restart ill behaving API runtime containers. Examples: Kubernetes, AWS Lambda.
API Data	This layer of the architecture is responsible for providing data to the APIs. Often time it is delivered via special purpose NoSQL databases that are designed for horizontal scale. Examples: MongoDB, DynamoDB, CosmosDB, DocumentDB, etc.

Web 2.x Architecture Summary

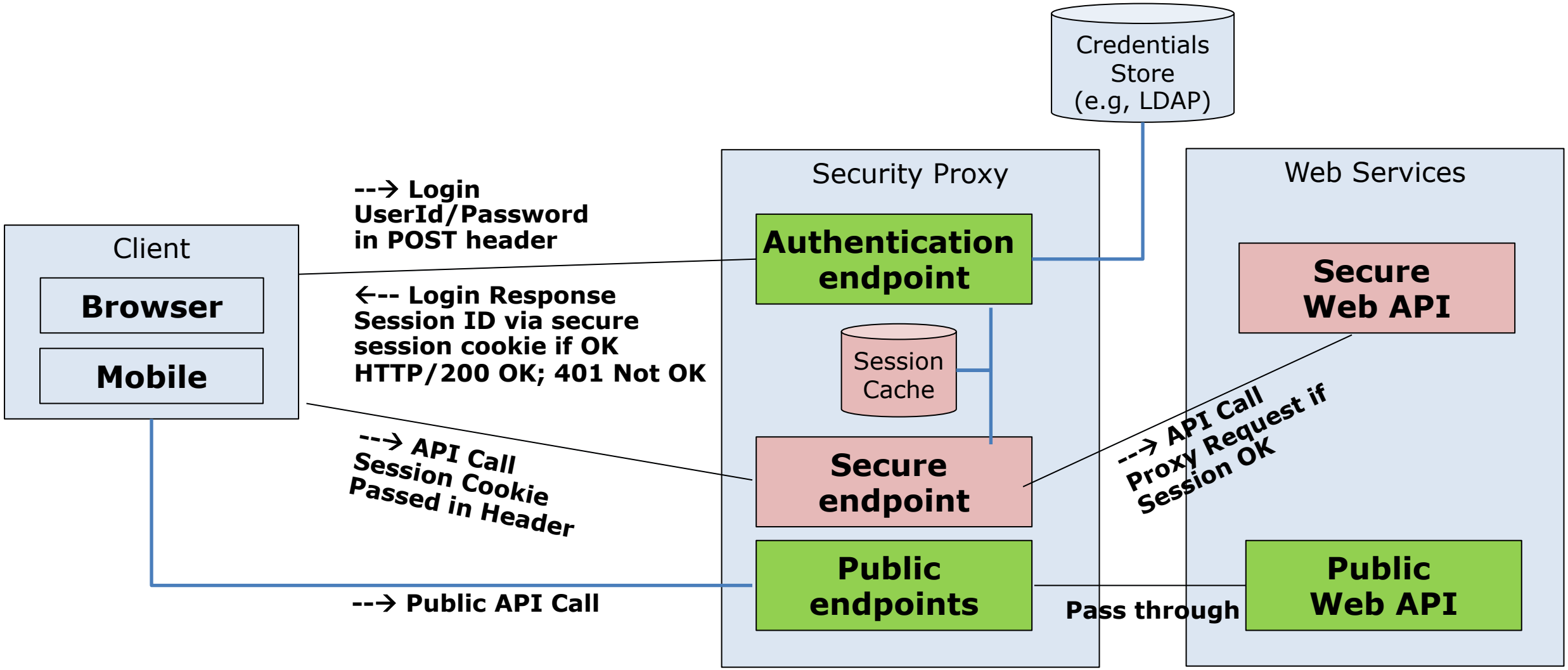


The Web 2.0 architecture has served us very well, but there are some challenges that are calling for an architecture pivot

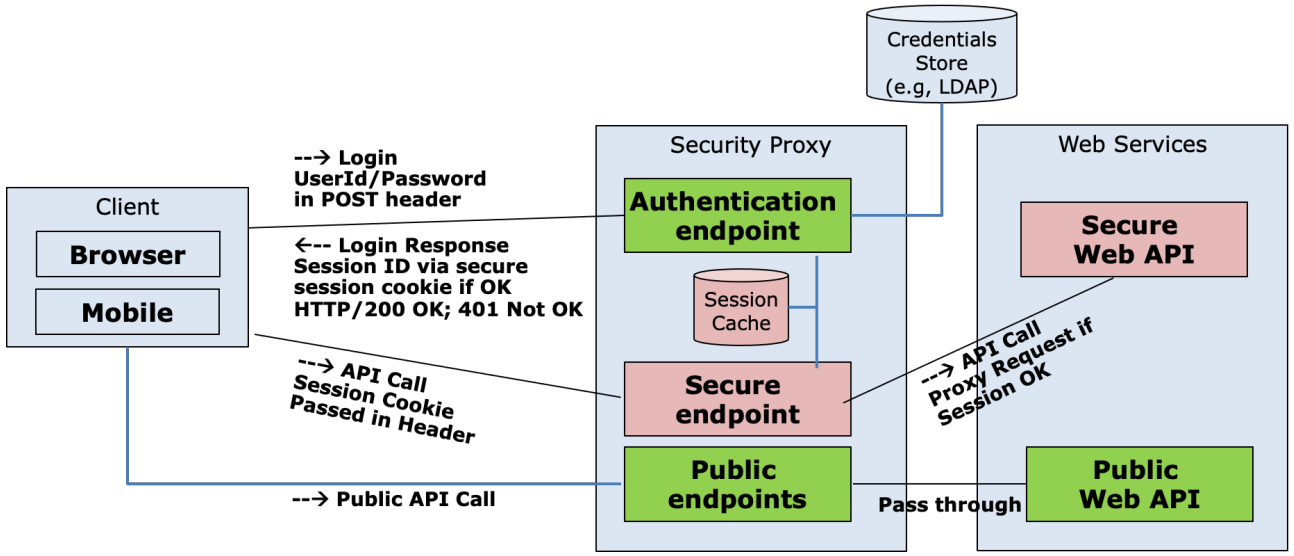
- Although the architecture of the web is highly scalable and distributed, much of the internet is controlled by central parties
  - Google (search), Facebook, Netflix, Amazon, etc
  - Platforms: Amazon AWS/ Microsoft Azure, Google GCP
- Many of the backend services provided by APIs are owned by entities that require knowing who you are
  - Banks, Insurance Companies, Entertainment Providers, Ecommerce Suppliers, etc
  - Think about the number of accounts and passwords you have, each is for a different entity

While its still too early to see where this is going architecturally, we are starting to hear about Web 3.0

Web 2.x Security Architecture – Session Based



Web 2.x Security Architecture – Session Based Architecture Strengths and Weakness



- Works well, time tested
- Very secure assuming payloads, and therefore headers are encrypted over HTTPS
- Issue: Many single points of failure – Proxy, LDAP
- Issue: Very difficult to cluster given session cache has to be distributed, or sticky sessions are required to "stick" a client application to an instance of a proxy
- Issue: Very difficult to federate, for example you want to use a third party to authenticate a user –e.g., Microsoft, google, amazon, twitter
- Issue: Works well for web browsers because cookies are a natural part of the ecosystem, but need to be supported by the client directly for other clients such as mobile applications

Using oAuth for security

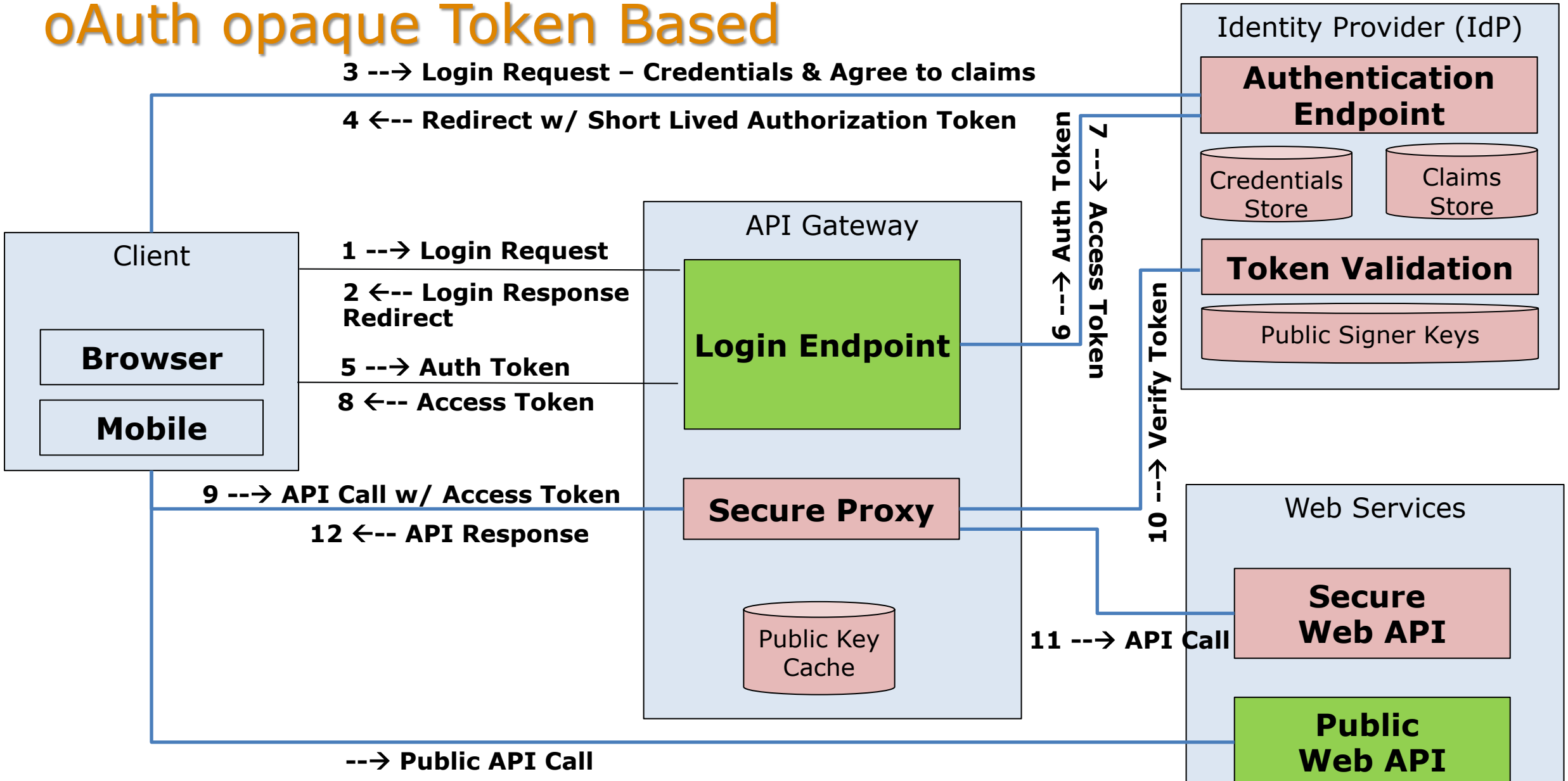
- Most modern security approaches are based on oAuth 2.0
- oAuth uses tokens to ensure security
- Tokens can be opaque or packaged as JWTs

Opaque Tokens Carry No Meaningful Information Normally a Random Value or a UUID
Example: 123e4567-e89b-12d3-a456-556642440000

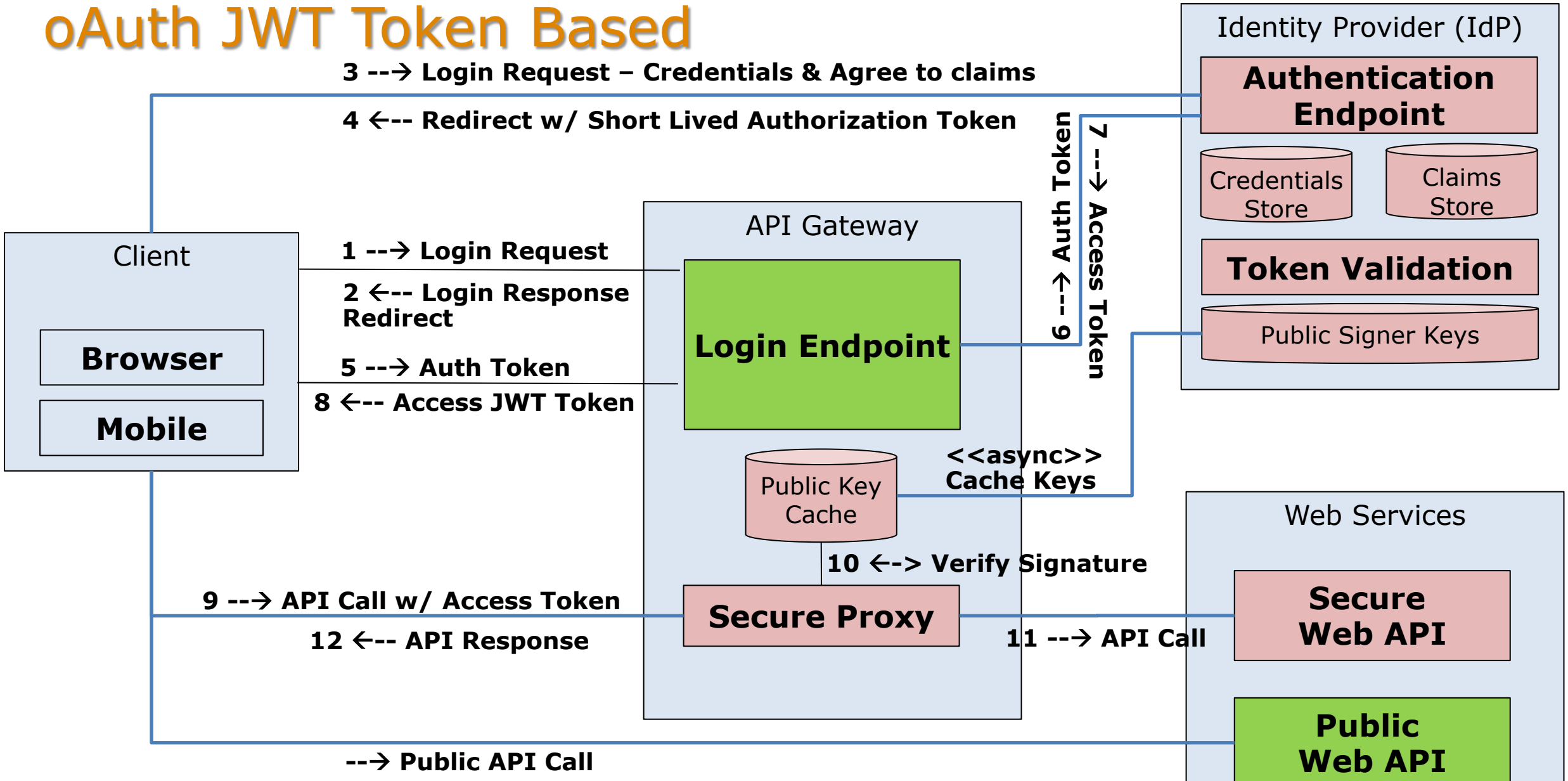
JWTs carry useful information in JSON format and are tamper proof via digital signatures
Example: See JWT.io

Significant Optimizations can be made to the oAuth protocol using JWTs

Web 2.x Security Architecture – oAuth opaque Token Based

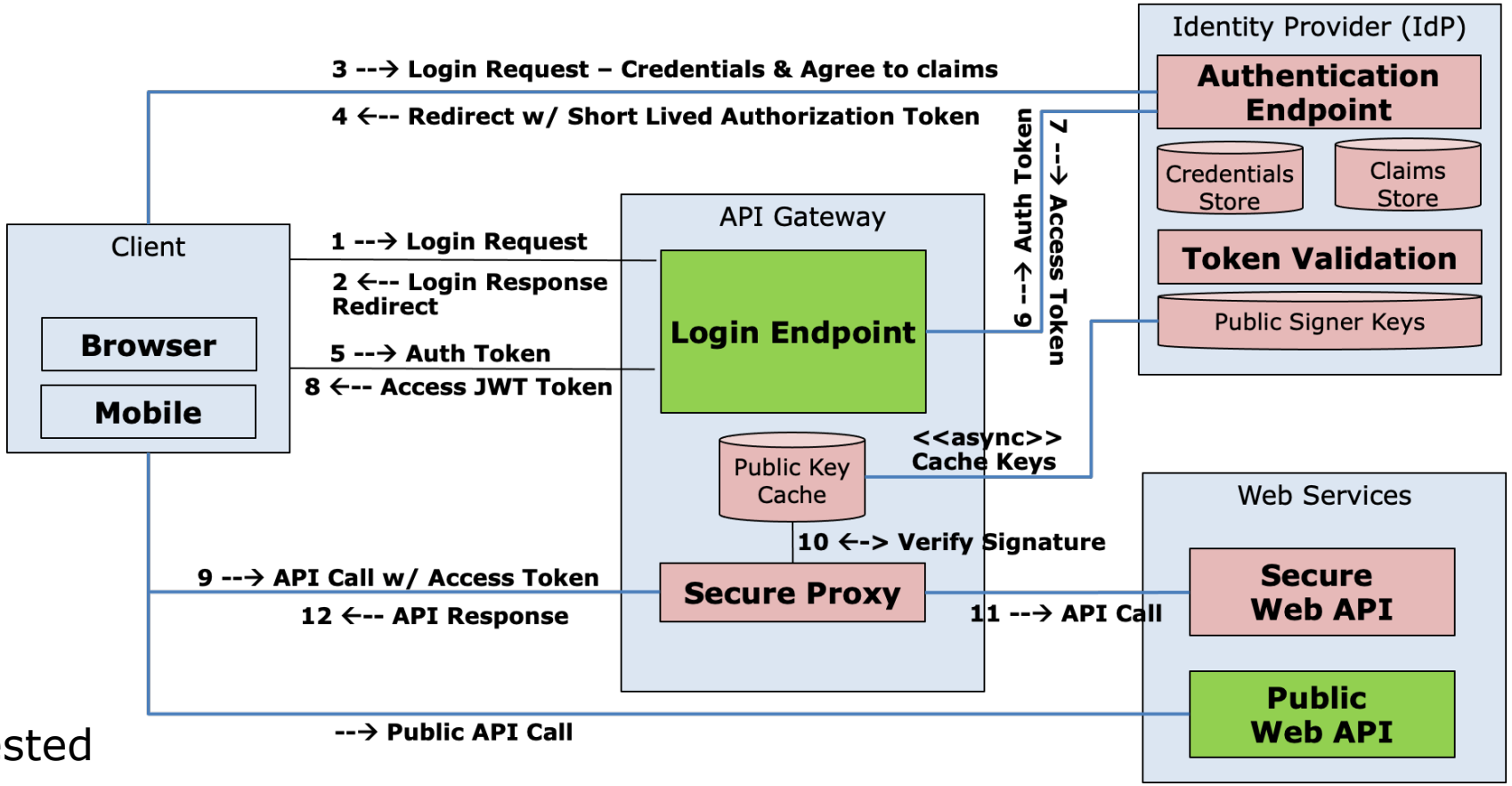


Web 2.x Security Architecture – oAuth JWT Token Based



Can you spot the optimization?

Web 2.x Security Architecture – oAuth Based Security



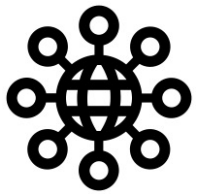
- Works well, time tested
- Very secure and flexible
- Works with a variety of different client devices, not just browsers
- Intrinsically supports federation given the IdP can be internally owned or external
- Has other features that were not covered such as referesh tokens that allow for long lived "remember me" scenarios

Web 3.0 Objectives – Possibly what's next

Web 3.0 Objectives Will Require Significant Architecture Changes to the Web

Decentralized Computing: Edge computing becomes the primary model – in Web 2.0 most transactions happen via APIs owned by centralized parties; Web 3.0 the endpoints directly sell, exchange or barter their data without loosing ownership control. The middleman goes away

Open, Trustless & Permissionless: **Open** in applications built from open source software; **trustless** in that the network itself allows participants to interact publicly or privately without a trusted third party; **permissionless** in that anyone, both users and suppliers can participate without authorization from a governing body



Connectivity and Ubiquity: We are starting to see this emerge in web 2.0; but web 3.0 is assuming that the web will be everywhere – connecting everything without any hardware or software limitations

Semantic Web: AI and ML will replace web 2.0s "query based" model to find information with deep personalization and relevance to the user and their current context

https://www.monocubed.com/blog/what-is-web-3-0/

Web 3.0 Also Introduces New Capabilities that Impact the Architecture

Web 3.0 Objectives Will Require Significant Architecture Changes to the Web

Reimagining Security: Identity moved to edge owned by individual users vs centralized identity providers making large breaches difficult, also the model moves to zero trust where secure connections are setup for every network interaction

Monetization of Users Attention vs Data: Web 2.0 largely monetizes its “free” services via targeted ads, however targeted adds require massive data collection and data processing leading to privacy issues. Web 3.0 focus shifts away from pushing ads to enabling companies to compete for your attention in the open using things like nonfungible tokens (NFTs)

Digital Products and exchange: With web 3.0 tangible digital assets can be created and exchanged using technologies such as blockchain. Examples, digital currency, selling digital content, music, movies, etc. NFTs can also be used to represent and exchange physical assets without intermediary third parties – art, real estate, event tickets, etc.

Before we get into the architecture of Web3, an example is required to understand it.

Scenario: You work for a large company that provides you healthcare coverage, you injure your knee, go to the doctor, the doctor orders an MRI. Possible outcomes:

- You get the MRI, and then a week later an email pointing you to an explanation of benefits showing that the service was fully covered, your cost is **$0, and you are happy**
- You are told by the doctor that you need an MRI, but your insurance requires a pre-authorization approval, and it might take up to a week. The MRI is approved, you get the MRI and then a week later a bill shows up for $50.
- Same as above, but you get a bill for $750 (or any other amount that is calculated)
- Same as above, but you call the insurance company, file an appeal, and then two weeks later a decision is made. The result could be your bill is adjusted to a much smaller amount, even $0, or your appeal can be denied, making you responsible for the full amount
- Your doctor wants you to get an MRI, tells you that a prior authorization is required, but the prior authorization is denied. If you want the MRI you have to pay the full cost, which could be several thousand dollars

How can the same scenario lead to such a diversity of outcomes that impact cost and customer satisfaction?

What is a prior authorization, and why are these things even necessary

The cost of healthcare in the US in 2019 has been reported as $3.8T, which is by far the largest category of spending in the US – about 18% of the total economy and approximately $11.5K for every person in the US

- Most people don't understand healthcare, and fully place trust in doctors. Thus, people think if a doctor says you need something, it must be necessary. However, doctors can be inconsistent, thus there is a need to improve consistency to manage overall healthcare costs
- Doctors are inconsistent, some are very conservative and order tests – "just in case", and some doctors are even fraudulent, for example, they may own a stake in an imaging lab, or get referral fees.
- We see imaging labs all over the place, they are out for profit, and the equipment that they use is expensive. Thus, the cost for the same procedure can vary widely from one imaging center to another
- Most people view processes like prior authorizations as a scheme by healthcare providers to deny coverage, and thus stuff their pockets with more profits. Many don't know that many commercial plans have the employer pay all medical costs, and healthcare companies get paid fees to leverage their networks and expertise. Thus, if a prior authorization is approved or denied, in many cases it does not impact in any way the amount of money a healthcare company spends or profits that they make.

At the end of the day, nobody likes this process, but many agree its necessary

But how can the outcomes for the same use case differ?

- Healthcare is provided to employers, with the terms and conditions outlined in a legal contract. The systems that healthcare companies use to process claims are based on algorithms. Thus interpretation is sometimes needed between a legal contract and an algorithmic decision.
- Healthcare companies deploy multiple strategies to lower the cost of making pre-authorization decisions, many of these things are not visible to patients consuming healthcare services
  - From a scale perspective, healthcare companies negotiate preferred rates with certain imaging providers to drive more volume to their services. What's good about this? There is transparency in terms that clinical best practices are followed – if they are not followed, the lab will not get paid. Patients and employers benefits from lower contracted rates
  - From a customer satisfaction perspective, healthcare companies offer incentives to providers to follow their prior-authorization best practices. Once a provider gets onto this list, they no longer have to get a pre-authorization as its expected that the provider applied best practices and are steering customers to the lowest cost, and highest quality imaging labs.

Why should you care? Would you be willing to drive 2 additional miles to go to an alternative imaging lab that saves you or your employer several hundred dollars?

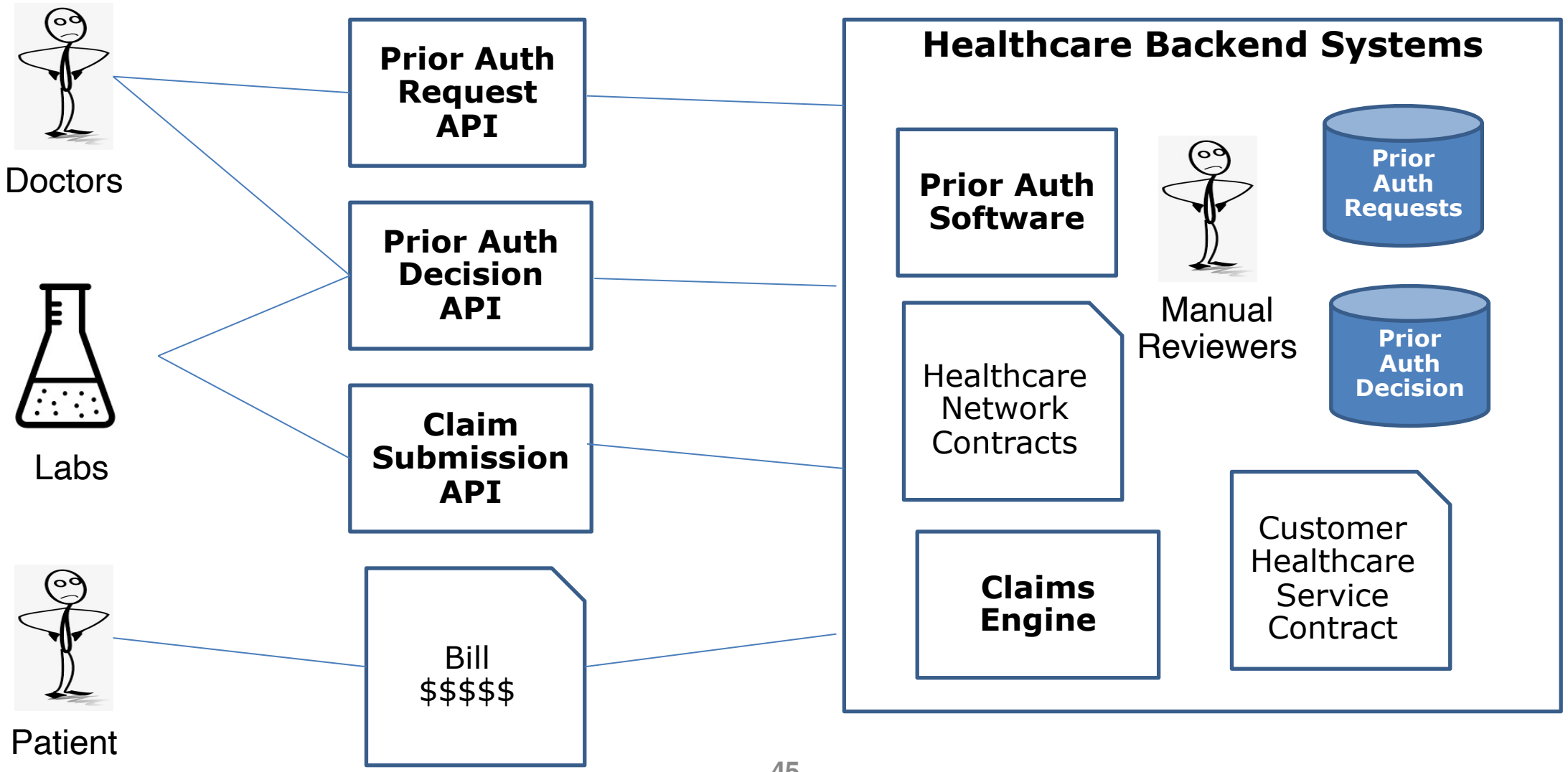
How doe healthcare companies monetize prior authorization services?

All healthcare companies deploy prior authorization as a best practice, how do healthcare companies monetize this service?

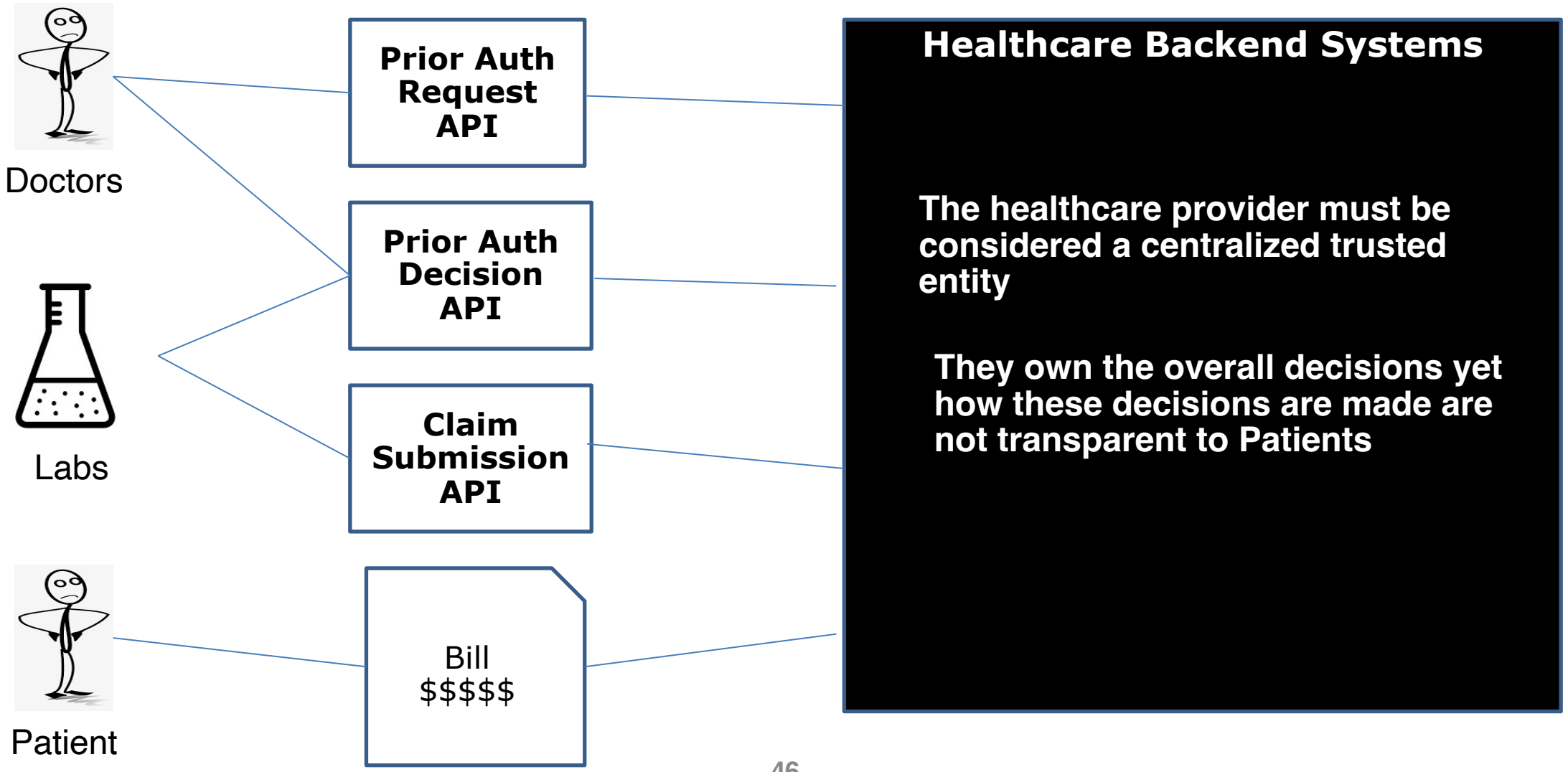
- In many cases healthcare companies cant and do not charge for prior authorization services. The costs to support this process is baked into healthcare premiums. That said, many healthcare companies use either third party experts, or third party software to make prior authorization decisions. Thus prior authorization is an expense and not a profit item.
- Healthcare companies that do prior authorization well, use this as a competitive advantage when selling their overall services to clients, thus doing this well can increase overall sales and profits
- Healthcare companies realize that prior authorization is accepted as necessary by providers and clients, but nobody really likes it because of issues like patient satisfaction, and occasional delays medical services.

Business Objective: Healthcare companies that can apply algorithmic decisions to making prior authorization decisions have a massive opportunity to lower costs, improve clinical quality, and improve patient satisifaction

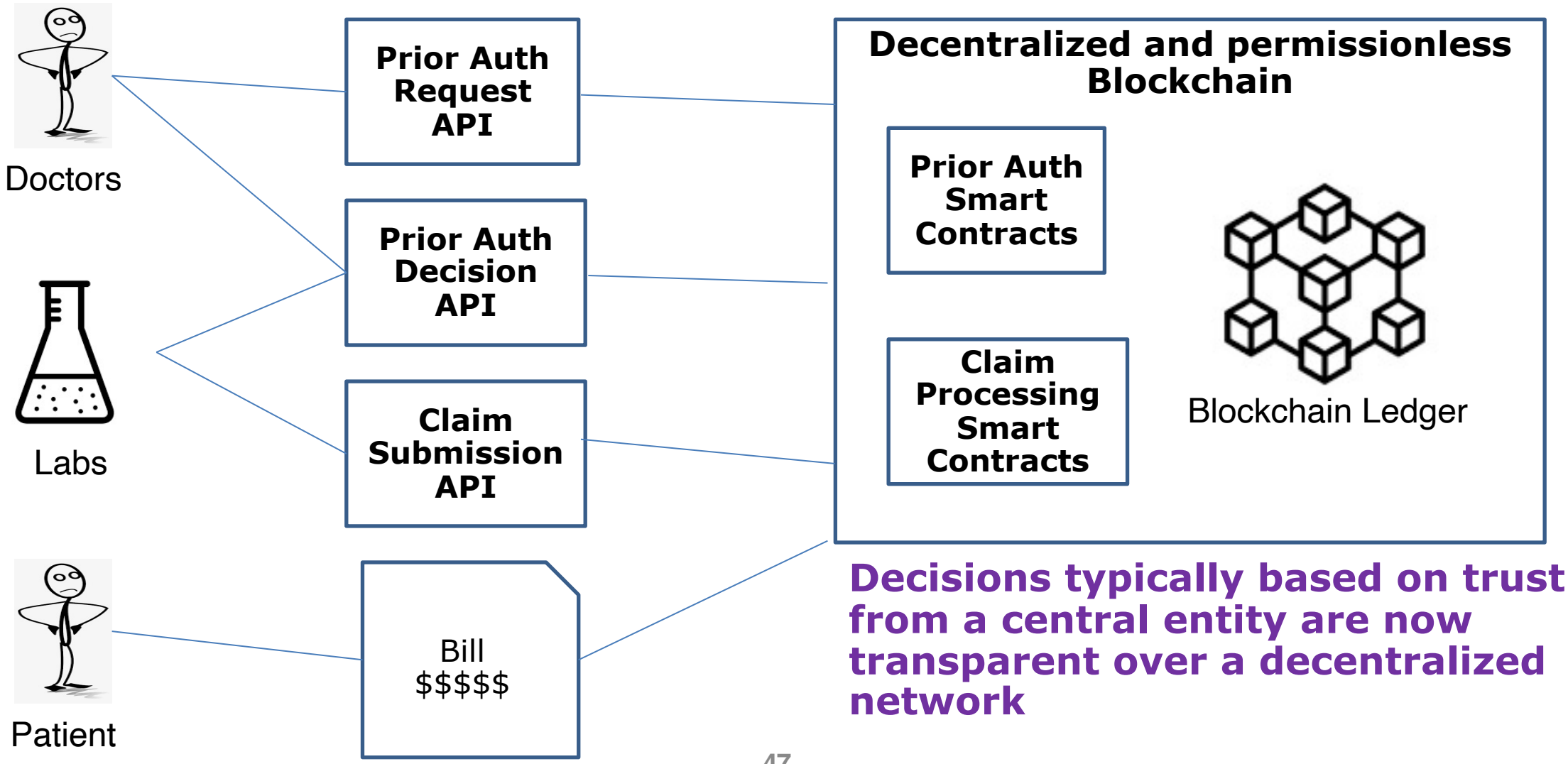
Architecture for Prior-Authorization in Web2



Architecture for Prior-Authorization in Web2

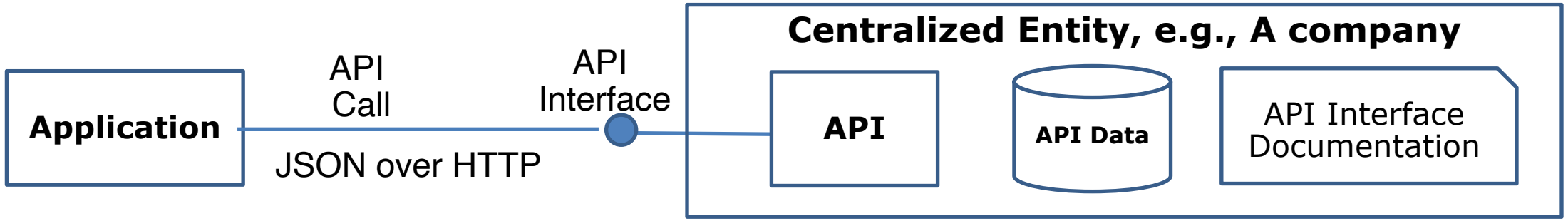


Architecture for Prior-Authorization in Web3



Decisions typically based on trust from a central entity are now transparent over a decentralized network

API in Web 2 vs Smart Contracts in Web3 from an Architecture Perspective

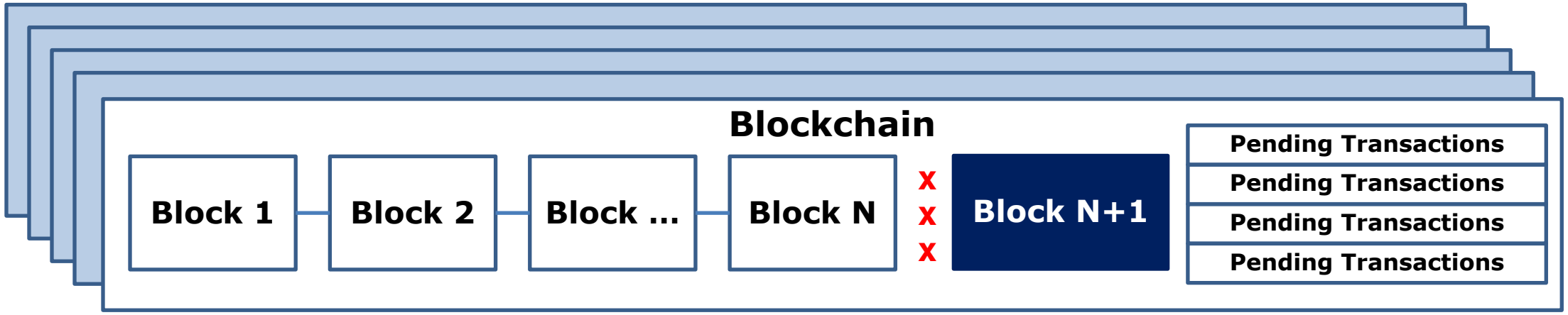


- APIs exposed over public internet
- APIs (usually) provide documentation over interfaces, actions they perform, outcomes and errors
- Entities who provide the API are trusted to fulfil the responsibility of the API

Examples: Send a venmo transaction for $25 to a friend, order a teeshirt over amazon, etc

Examples: Trust must be provided to the entity supplying the API given they do not make available their databases, logs, or API algorithm implementations for reviews. Appeals when there is disagreement or dissatisfaction is not easy – e.g., ”I want a refund, I never received my package”

Blockchain Architecture – The blockchain

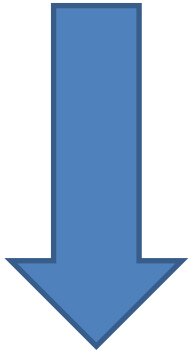


A blockchain is a linked list of blocks, each block containing a validated collection of transactions

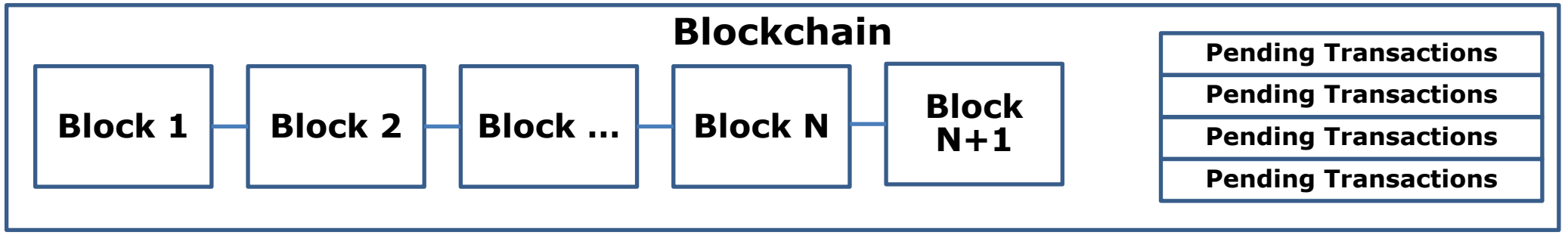
A blockchain is distributed and replicated, all parties can view the same thing

Blockchain miners perform a complex process to mint new blocks from pending transactions, but once minted, the correctness of the block is easily verified by all parties

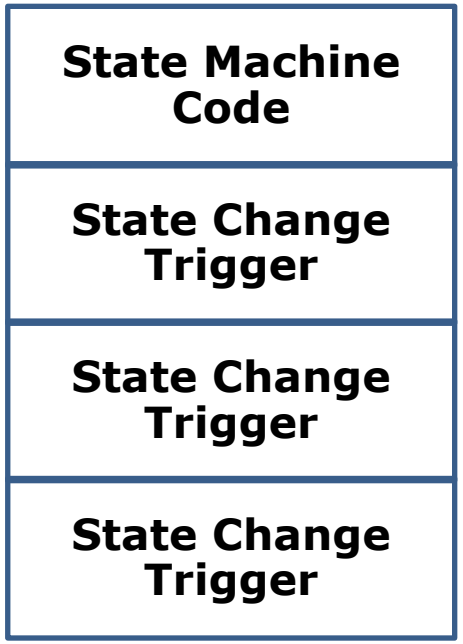
Blockchain miners compete for awards via fees and tips to mint new blocks

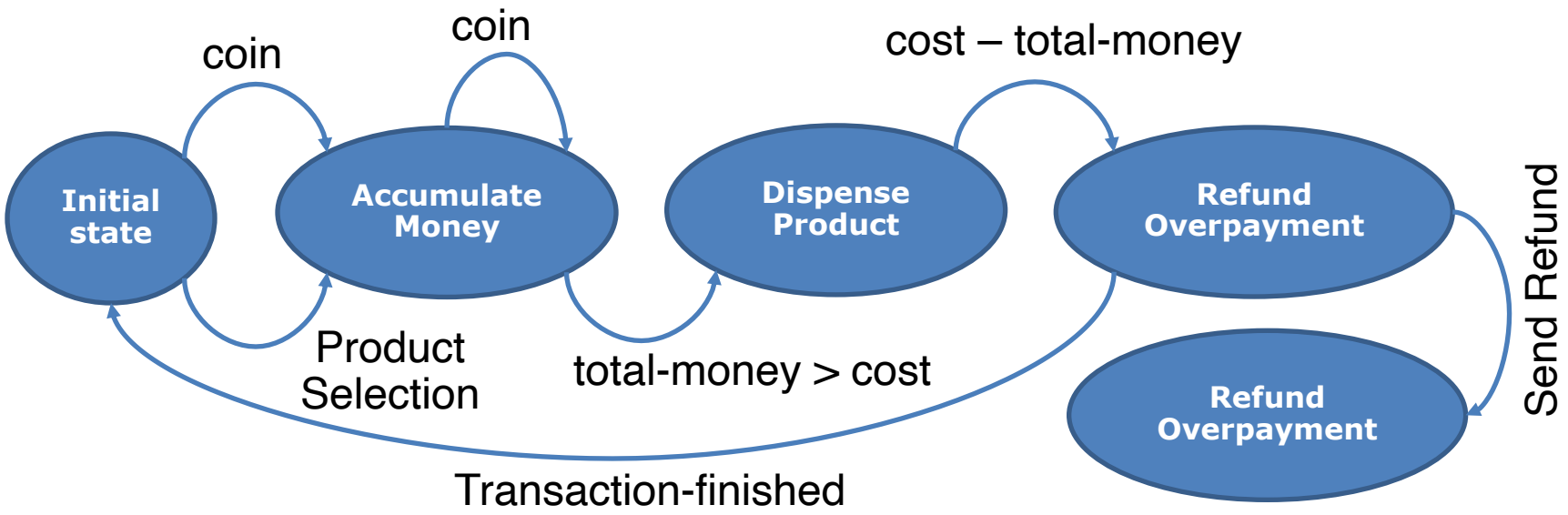


Blockchain Miner(s)



Blockchain Architecture – The smart contract





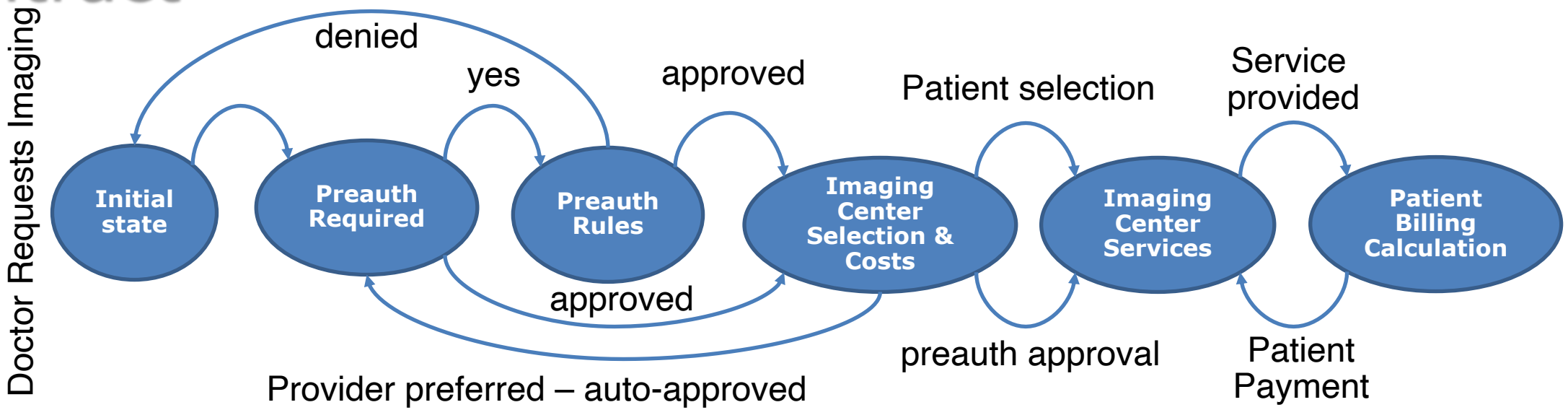
A "smart contract" is basically a state machine. The logic for the state machine is recorded on the blockchain itself

Since on the blockchain, its behavior is open to all to review, is immutable, and cant be changed (but it can be evolved)

All state changes are recorded on the blockchain open to be validated to all – If I paid a coin, all would see it and a merchant could not claim that I did not

Changes in state happen automatically once all of the preconditions are satisfied

Returning to the Pre-Auth example via smart contract



The previous pre-auth experience was not fully transparent to all participants – lends itself to suprises and dissatisfaction

The smart contract makes concepts such as preferred providers, the rules around if a pre-auth is required, how are they approved or denied, cost transparency around location selection and how payment is made transparent to all parties.

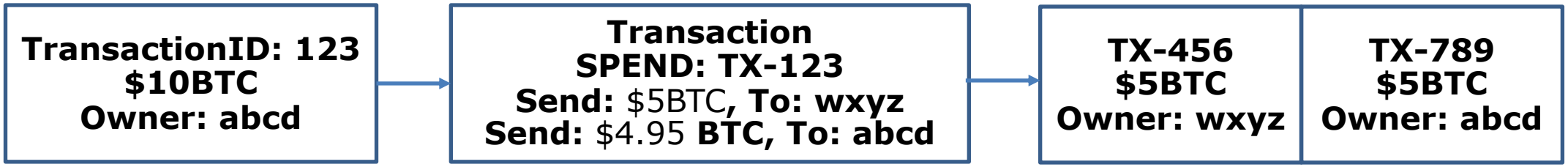
There are no secrets, nor the ability to alter how this works

Identities and patient information is protected via private keys, just like in other blockchains

Fungible Tokens

A fungible token is divisible, interchangeable and non-unique. Crypto-currency is an example. For example, 10 individual bitcoin transactions in a wallet is equivalent to one transaction containing 10 bitcoins. Fungible tokens are often divided when exchanged.

Example, I have a transaction on the blockchain indicating my ownership of 10BTC, and want to send you 6 BTC



Transactions are immutable, to exchange a fungible token, a new transaction is created that “spends” the old one and divides it into multiple parts. One part to new party, the other is change. The reminder is a fee/tip to the miner to incent them to put on a block.

Non Fungible Tokens

The other interesting thing about Web3 is the ability to create and own digital assets

The obvious example is crypto-currency where crypto currency can be used directly in commerce or also be exchanged for fiat money such as US Dollars.

A NFT is just a digitally signed token that is stored on the blockchain – they can be owned or exchanged just like any other crypto-token

Examples:

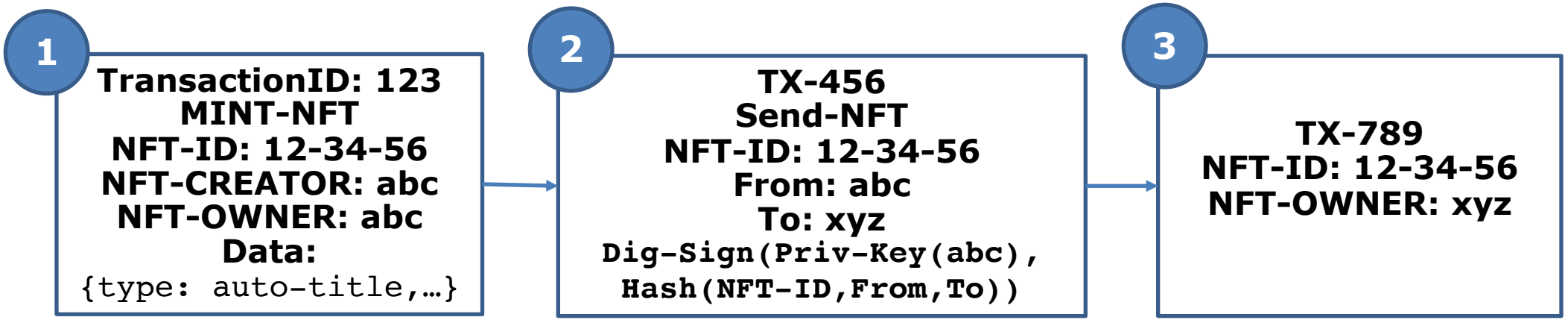
- Car and home titles
- Insurance contracts
- Birth/Marriage/Death certificates
- Electronic art
- Intellectual property

Non-Fungible Tokens

(google ERC-721 for the definition of the standard for NFTs, note the below is simplified to demonstrate the concepts in the architecture)

A non-fungible token is used to assert ownership of an asset.

Example, I mint a NFT asserting my ownership of a car, then I gift it to my son. In this example, my wallet id is "abc", and my son's wallet id is "xyz"



1. A NFT can be created by anybody via self minting. Notice when an NFT is created its given a unique ID and has other attributes such as metadata about the NFT itself
2. When an NFT is going to be exchanged, a new transaction/smart contract is executed, showing ownership transfer using "to" and "from". Notice the private key of the current owner is used to digitally sign the transactions to prove that they are transitioning ownership. The signature can be easiely verified via the public key
3. Once ownership is transferred, notice the NFT-ID does not change, it never will change. This shows linage back to the original creator of the NFT.

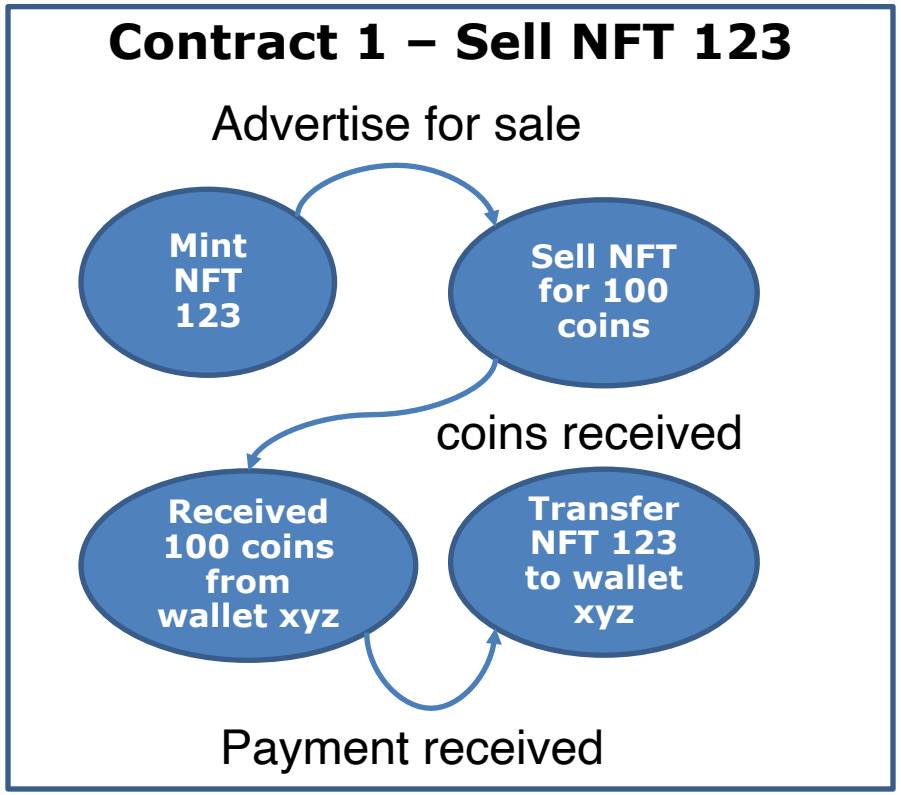
Semi-Fungible Tokens

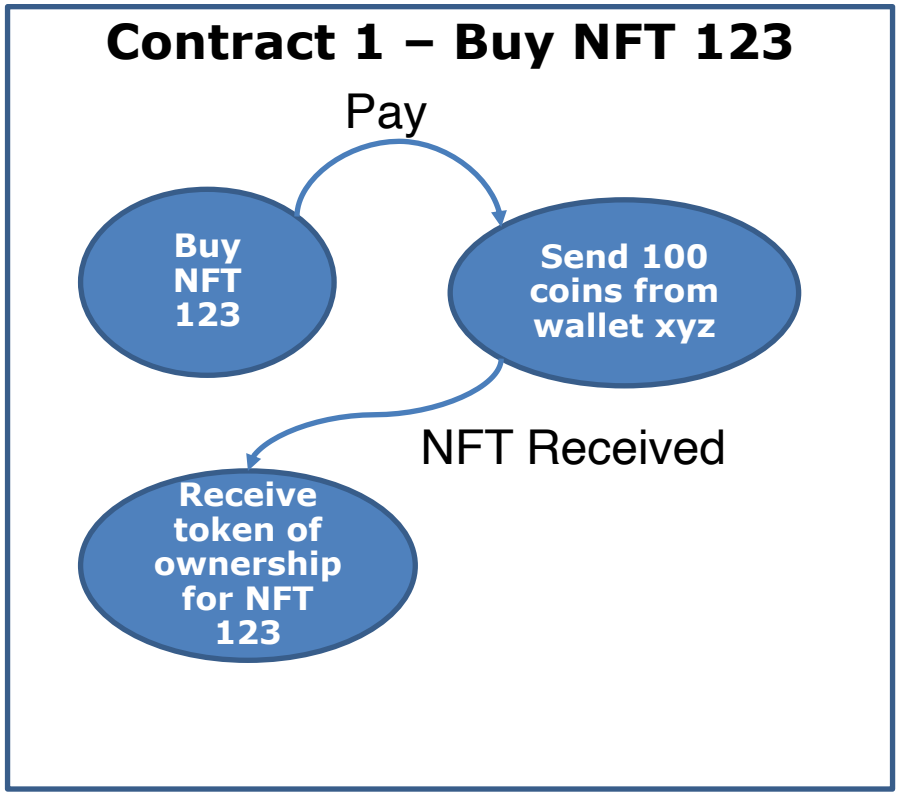
(google ERC-1155 for the definition of the standard for SFTs, note the below is simplified to demonstrate the concepts in the architecture)

- A semi-fungible token is basically the same thing as a NFT, with one minor difference
- NFTs are unique, and when they are minted are tied to exactly one asset
- Semi-Fungible tokens (SFT) can be minted with a quantity. For example mint a “lot” of NFTs. When an SFT is minted, its quantity is fixed and can not be altered up or down.
- Example, I can issue 100 SFTs for a piece of artwork, either digital or physical. Thus each SFT represents 1/100 ownership of the physical asset. You can of course, own more than one SFT for an asset thus increasing your ownership stake
- If you mint a SFT with quantity 1, then its basically a NFT.

Selling Non- or Semi-Fungible Tokens

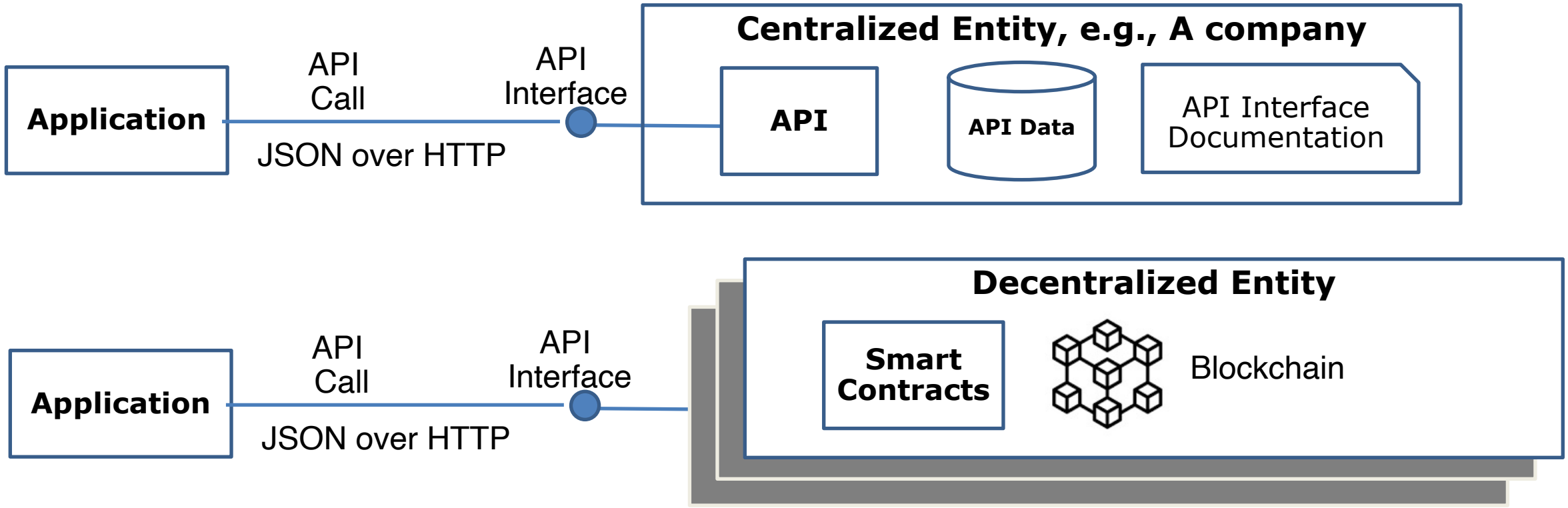
We just covered how to exchange a NFT and SFT, hopefully you can see how we can not only exchange but sell the transfer of rights with a SFT via smart contracts





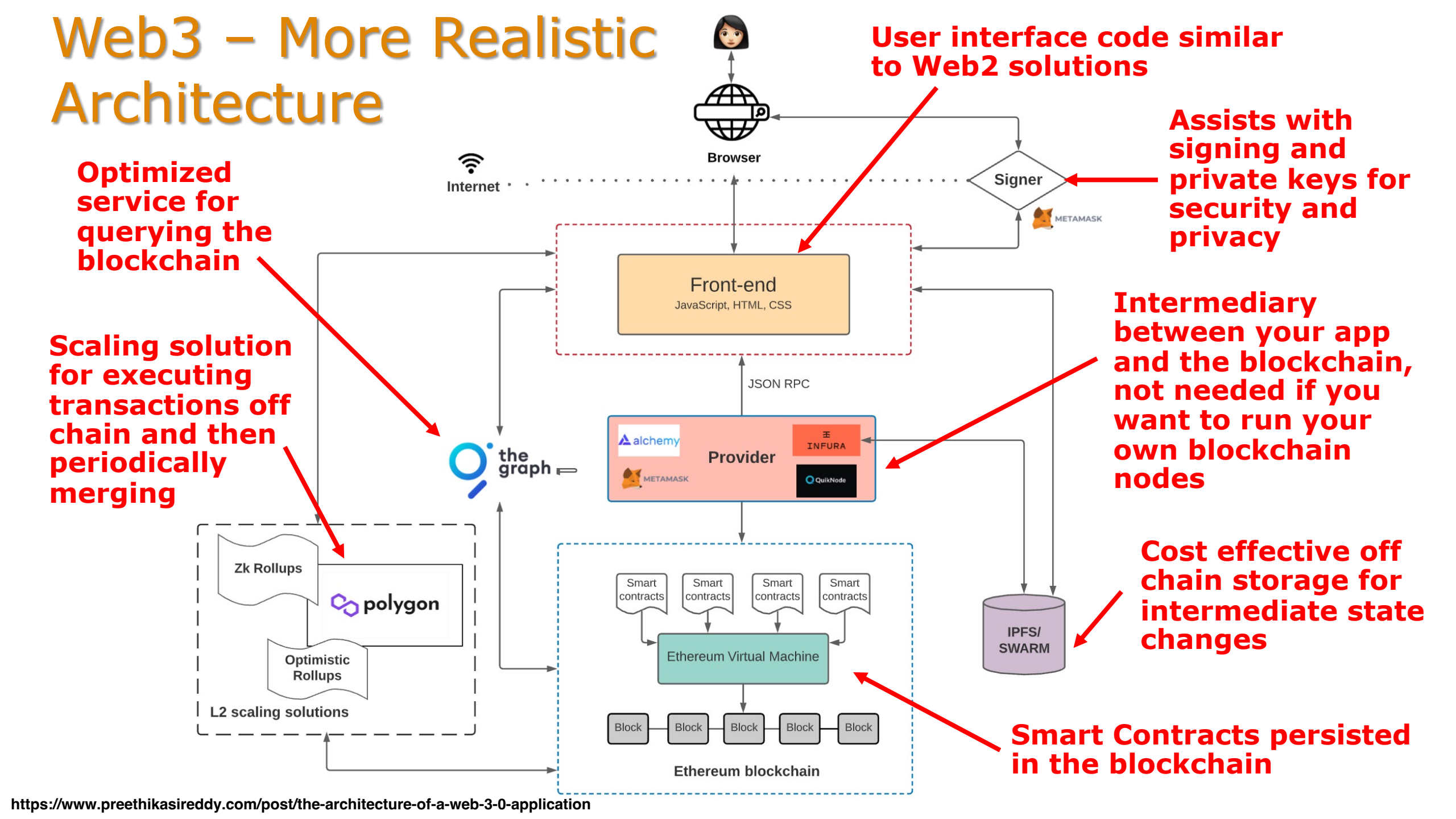
Conceptually a buyer creates a contract offering to sell an NFT, and a buyer creates a contract to buy an NFT. Note that state changes happen automatically when events are triggered via the contract

API in Web 2 vs Smart Contracts in Web3 from an Architecture Perspective



Architecturally, the concepts make sense, but the current-state-of-the-art blockchain technology is hard to scale and introduces other challenges that need to be overcome for this to be a practical solution

Web3 – More Realistic Architecture



https://www.preethikasireddy.com/post/the-architecture-of-a-web-3-0-application

Web 3.0 Summary

Significant Technical Challenges exist for Web3 to overtake Web2, Time will Tell

The more likely outcome is that Web3 will coexist and complement Web2 for some time into the future