

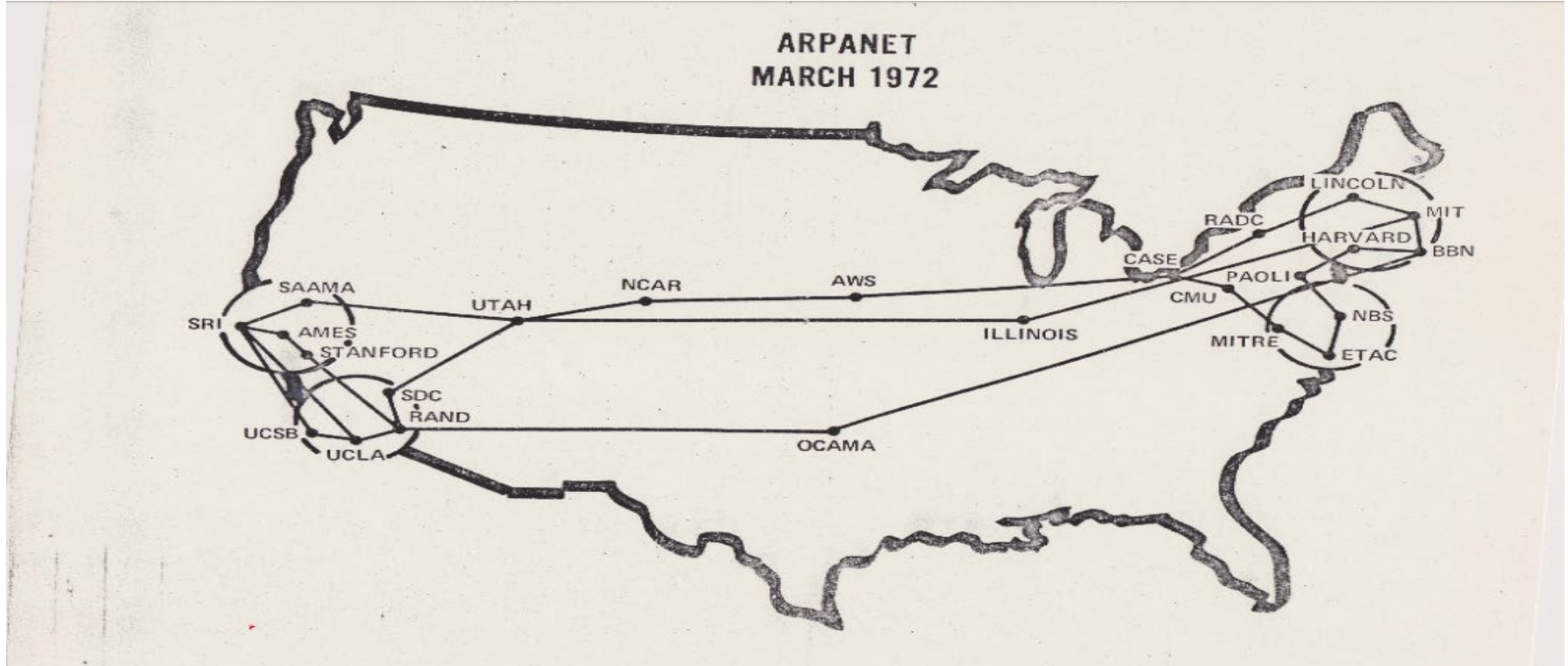
Bilgisayar Mühendisliđi Yüksek Lisans  
Programı  
2026 Bahar

Yazılım Mimarileri  
21.04.2026

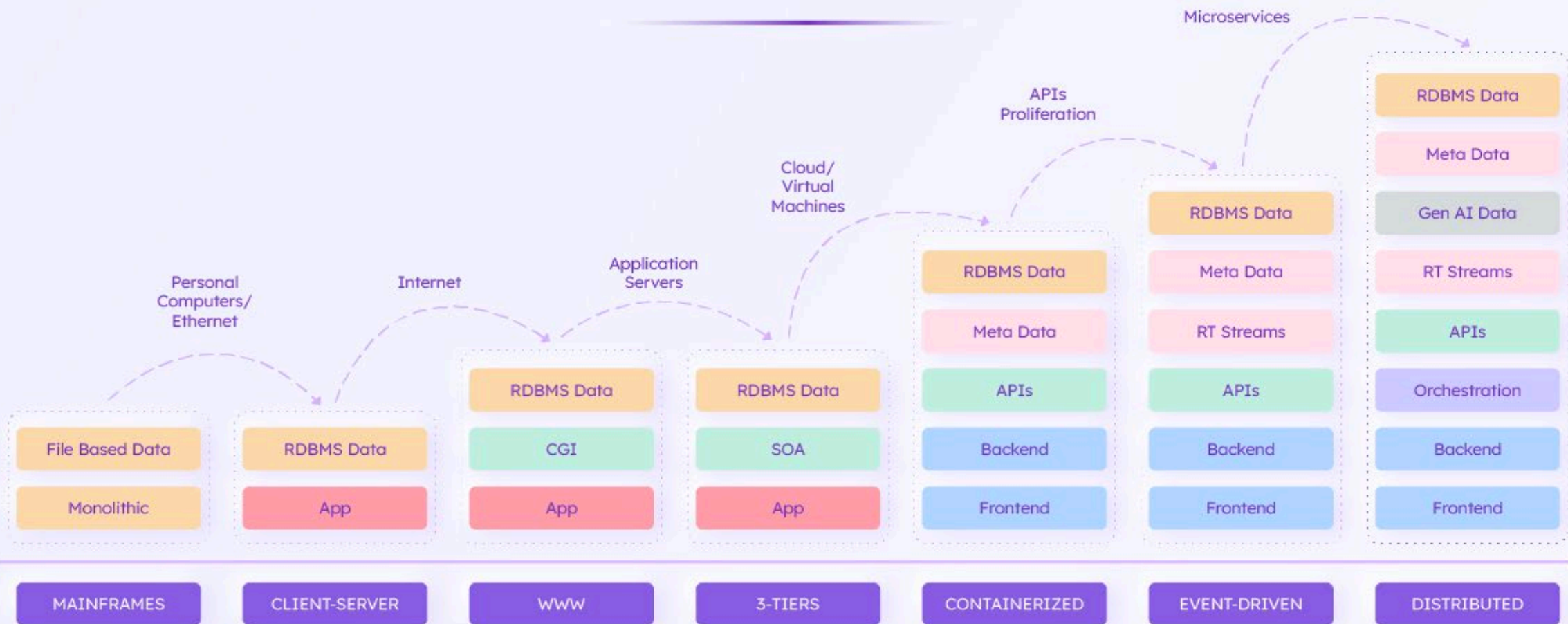
# Klasik Yazılım Mimarileri

İnternetin ilk dönemlerinde kurumsal veri işleme sistemlerinde standart olarak kullanılmıştır.

O dönemlerde bu yapılar tahmin edilebilir veri yükleri için tasarlanmaktaydı.

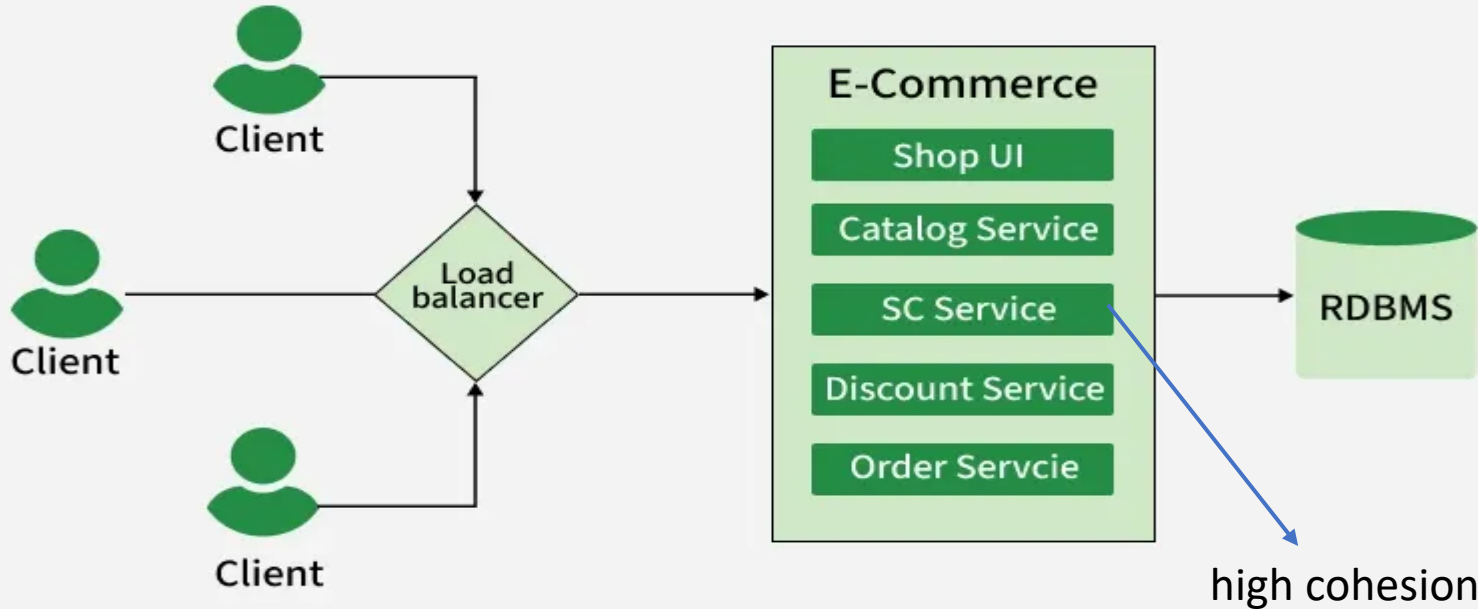


# Tech Stack Evolution



# Monolitik Mimari

- ❑ Tüm uygulama bileşenleri, yani kullanıcı arayüzü, iş mantığı, veri erişimi tek bir yürütülebilir dosya içinde toplanır.
- ❑ Geliştirmenin başlangıçta kolay olmasının, test edilmesinin ve yayına alınmasının basitliğine rağmen, tek bir hatanın tüm sistemi çöktürmesi önemli bir dezavantajdır
- ❑ Teknolojiye bağımlılık çok yüksektir ve ölçekleme zordur.



Ürün kataloğu, kullanıcı hesapları, sepet, siparişler ve ödemeler gibi özelliklerin tek bir uygulama olarak oluşturulup kullanıma sunulduğu bir e-ticaret uygulamasıdır.

# Cohesion & Coupling ilişkisi

Yazılım tasarımının en temel konularından birisi Coupling (Bağlama) ve Cohesion(Yapışma) özellikleridir.

❑ «Cohesion» bir bileşenin elemanlarının işlevsel olarak ilişkili olma derecesinin bir ölçüsüdür.

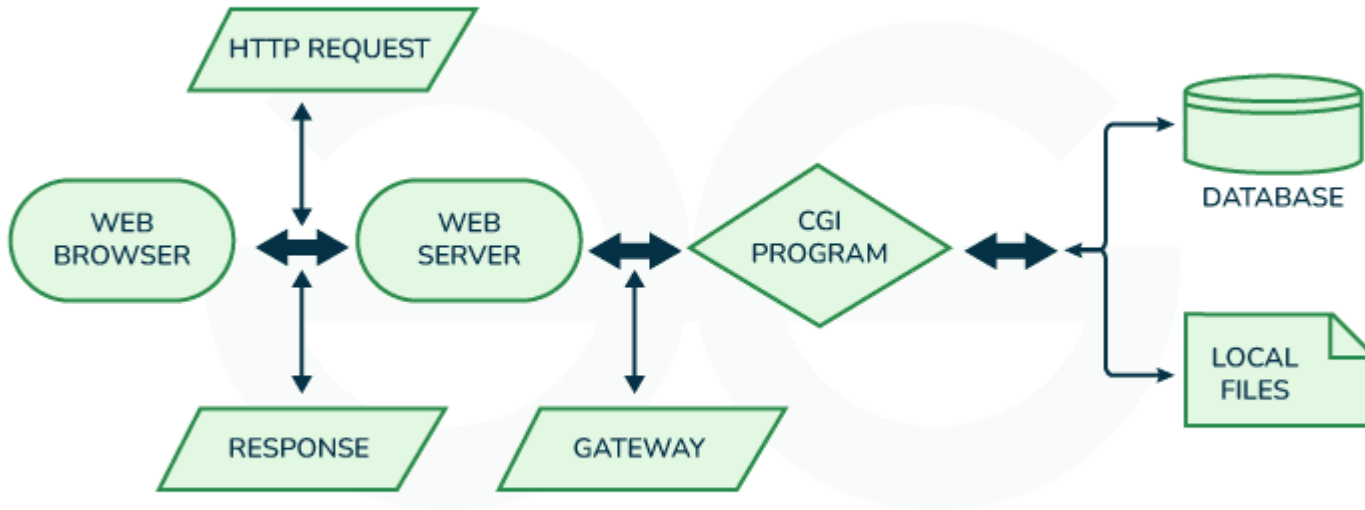
❖ Diğer bir ifade ile «cohesion» bileşenin içini simgeler.

❖ Bu bileşende tek bir görev gerçekleştirilir ve bu göreve ait tüm öğeler tek bir bileşen altında toplanır.

❑ SOLID prensiplerinin S harfini temsil eden (Single Responsibility Principle) cohesion tanımını açıklar.

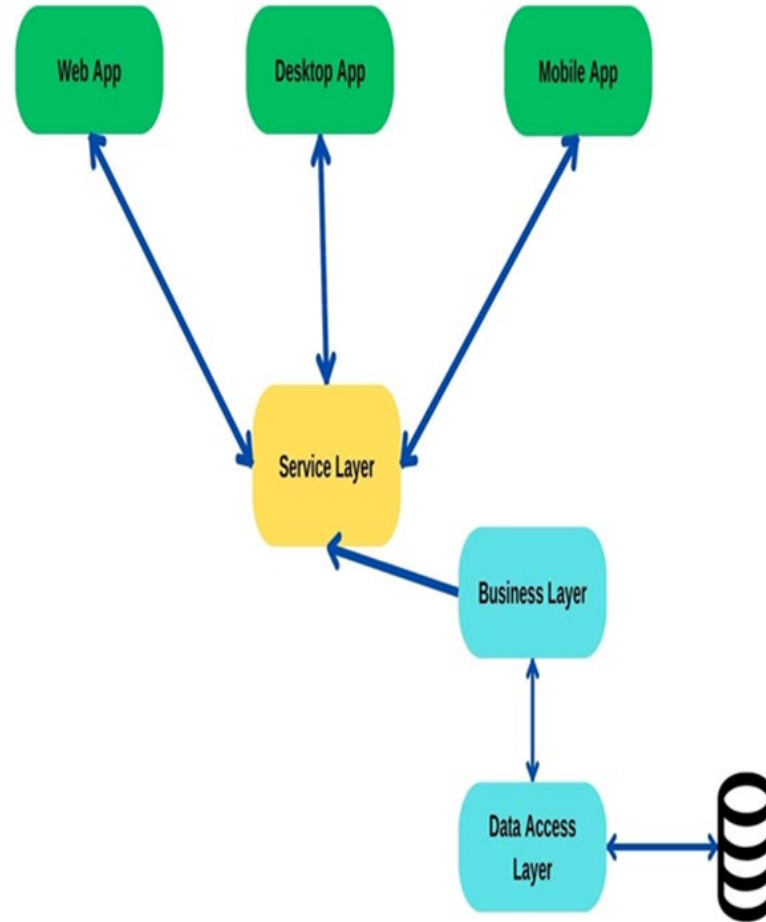
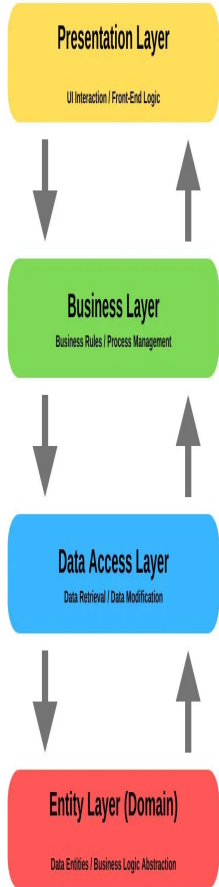
❑ Eğer bir bileşen içerisinde birbirinden bağımsız öğeler (metodlar, veriler) varsa bunlar olabildiğince ayrıştırılmalı ve modülün «**High Cohesion**» özelliğine sahip olması istenir.

# Common Gateway Interface –CGI



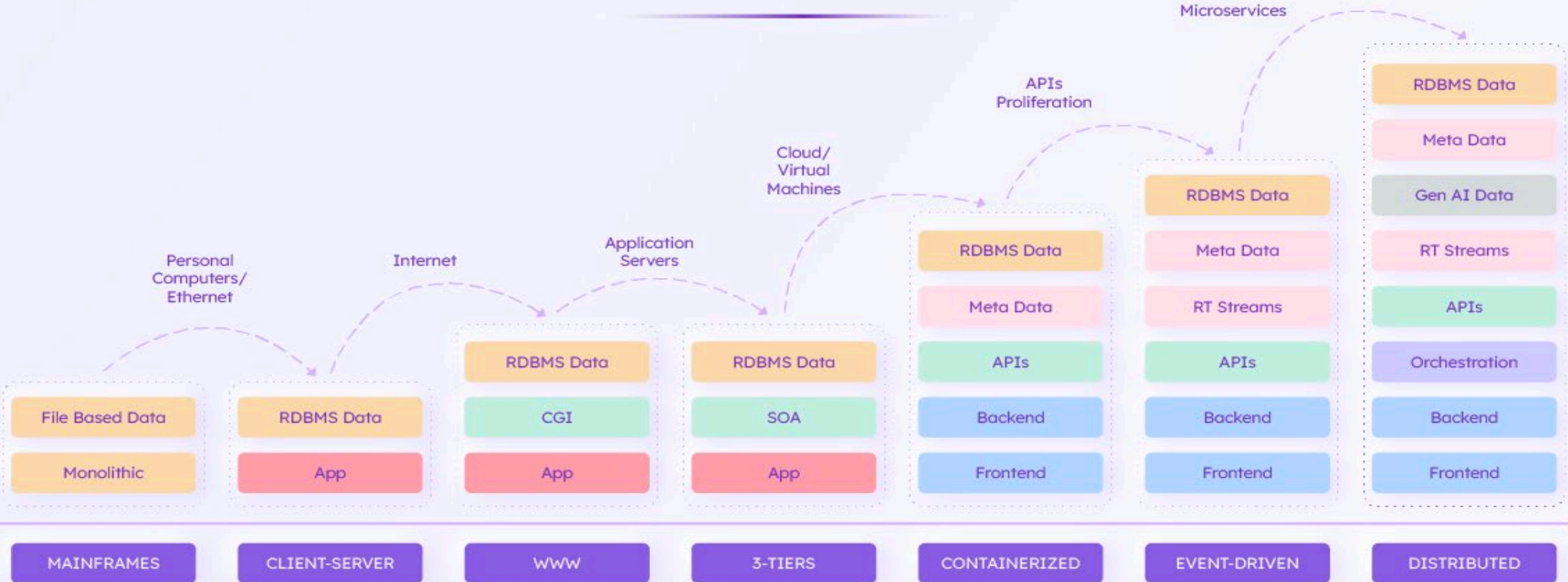
- ❑ Web sunucusu bir kullanıcının HTTPS isteğini bir program çalıştırarak işler.
- ❑ Sunucuda bulunan programı çalıştırarak verileri işler ve ilgili dinamik içeriği üretir.
- ❑ Bir istemci istekte bulunduğunda, bu veriler bir ortam değişkeni olan CGI script'e iletilerek istek işlenir.
  - ❖ Kullanıcının girdisi için dinamik içerik oluşturur.
- ❑ Web sunucusu CGI tarafından oluşturulan çıktıyı aldıktan sonra, yanıtı istemciye gönderir.

# Katmanlı Mimari (Layered / N-Tier Architecture)



- ❑ Uygulama sorumluluklarına göre katmanlara ayrılır.
- ❑ Sunum katmanı, iş mantığı ve verinin ayrıldığı en yaygın klasik yapıdır.
- ❑ Her katman sadece altındaki ve üstündeki katmanla iletişim kurar.
- ❑ Bu mimari, standard kurumsal uygulamalar ve **CRUD** işlemlerinde kullanılır.

# Tech Stack Evolution



# CRUD Nedir?

Bir veri nesnesinin yaşam döngüsünü yönetmede kullanılan standart operasyonların koleksiyonudur.

<b>Kısaltma</b>	<b>İşlem</b>	<b>SQL Komutu</b>	<b>HTTP Metodu (REST)</b>	<b>Açıklama</b>
<b>C</b> - Create	Oluştur	INSERT	POST	Yeni bir kayıt ekler
<b>R</b> - Read	Oku / Listele	SELECT	GET	Mevcut verileri görüntüler
<b>U</b> - Update	Güncelle	UPDATE	PUT / PATCH	Var olan bir kaydı değiştirir
<b>D</b> - Delete	Sil	DELETE	DELETE	Bir kaydı sistemden kaldırır

# CRUD Mimari Tasarımında Neden Önemlidir?

CRUD ile gerçekleştirilen sadece veri girişlerinin işlenmesi değildir; aynı zamanda **sistem tasarımı prensibi** olarak **önemli rol** üstlenir. Bunlar;

**i) Kullanıcı Arayüzü Tasarımı** Sisteme *Kullanıcı Yönetimi* gibi bir özellik eklenirken CRUD işleminin arayüzde nasıl yer alacağı planlamalıdır. Kayıt formu (create), tablo listesi (read), düzenleme ekranı (update) ve onay kutusu (delete) için standart bir akış olmalıdır.

## **ii) Yetkilendirme ve Güvenlik**

Her kullanıcı her CRUD işlemini yapamaz. Örneğin: Ziyaretçi, sadece *Read* yetkisine, Editör, *Create* ve *Update* yetkisine, Admin ise *Delete* dahil tüm yetkilere sahiptir.

## **iii) API Tasarımı (RESTful Services)**

Mikroservisler gibi modern web mimarilerinde servisler birbirleriyle haberleşmelerini CRUD prensiplerini uygulayarak gerçekleştirir. Bir servise GET isteği atılırsa veri istenir; POST isteği atıldığında yeni bir veri oluşturulur.

# Modern Yazılım Mimarileri

- ❑ Her bir uygulama kendi veri tabanına sahiptir; uygulamaların bağımsız olarak gerçekleştirildiği küçük servis parçalarına bölünmüştür.
- ❑ Netflix, Amazon ve Uber gibi büyük ölçekli sistemlerin temelidir.

# Olay Odaklı Mimari (Event-Driven Architecture -EDA)

- ❑ Sistemdeki durum değişikliklerini, yani olayları temel alarak çalışan bir yapıdır.
- ❑ Bir servis bir olay üretir ve bu olayla ilgili diğer servisler tepki verir.
- ❑ Servisler arası bağımlılık, diğer ifade ile *coupling* minimum seviyededir (*loosely coupling*) .

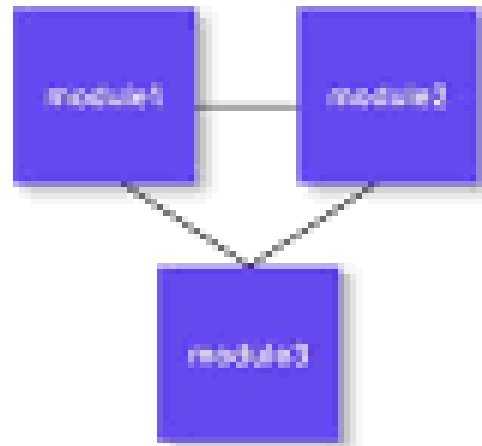
1.Next-Generation Event-Driven Architectures: Performance, Scalability, and Intelligent Orchestration Across Messaging, 2025, Jahidul Arafat,Fariha Tasmin, Sanjaya Poudel.

2.Event-Driven Architecture: Building Responsive and Scalable Systems for Modern Industries, 2021 Ramakrishna Manchana.

# Cohesion & Coupling ilişkisi

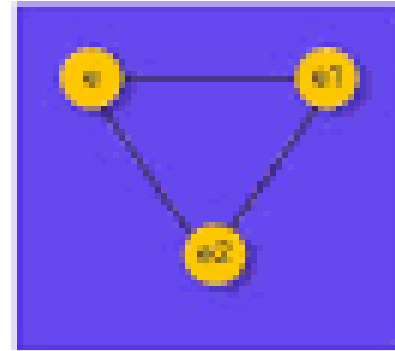
- ❑ Coupling, bileşenler arasındaki karşılıklı bağımlılık derecesinin ölçüsüdür.
- ❑ Tightly Coupled bir yapıda ilişki içerisindeki modüller birbirlerine sıkı sıkıya bağımlıdır.
  - ❖ Bu bağımlılık, bileşenlerin birinde yapılacak değişikliğin diğer bileşenleri de etkiler ve projenin genişletilebilirliği ve maliyeti artar.
- ❑ Bunun önüne geçmek için yazılımın olabildiğince *Loosely Coupled* olması istenir.
  - ❖ Yani bir modül üzerinde yapılan değişikliğin diğer modüllere sıçramaması gerekir.

# Cohesion vs Coupling in Software Engineering



**Coupling**

*Vs*



**Cohesion**

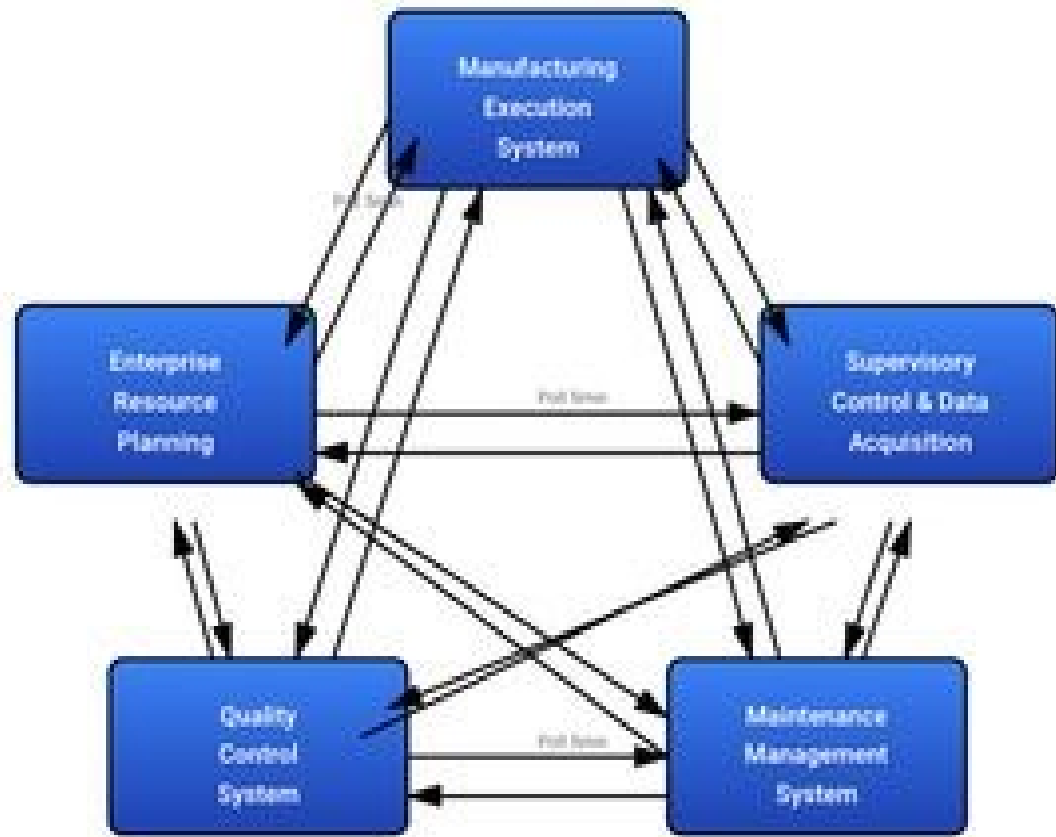
# Geleneksel Olay Odaklı Mimari

- ❑ Tasarım ve implementasyonunun basitliđi,
- ❑ Mimarinin iyi tanımlanmış (well-defined) kalıplar ve uygulamalarını içermesi olması,
- ❑ Güçlü destek ve dokümantasyona sahip olması pozitif özellikleridir.

Diđer taraftan:

- ❑ Deđişen iş yüklerine uyumu oldukça sınırlıdır.
- ❑ Ayarlamaların elle yapılması nedeniyle operasyonel karmaşıklık oldukça yüksektir.
- ❑ Veri miktarının deđişmesi durumunda performansta düşüş görülebilmektedir.

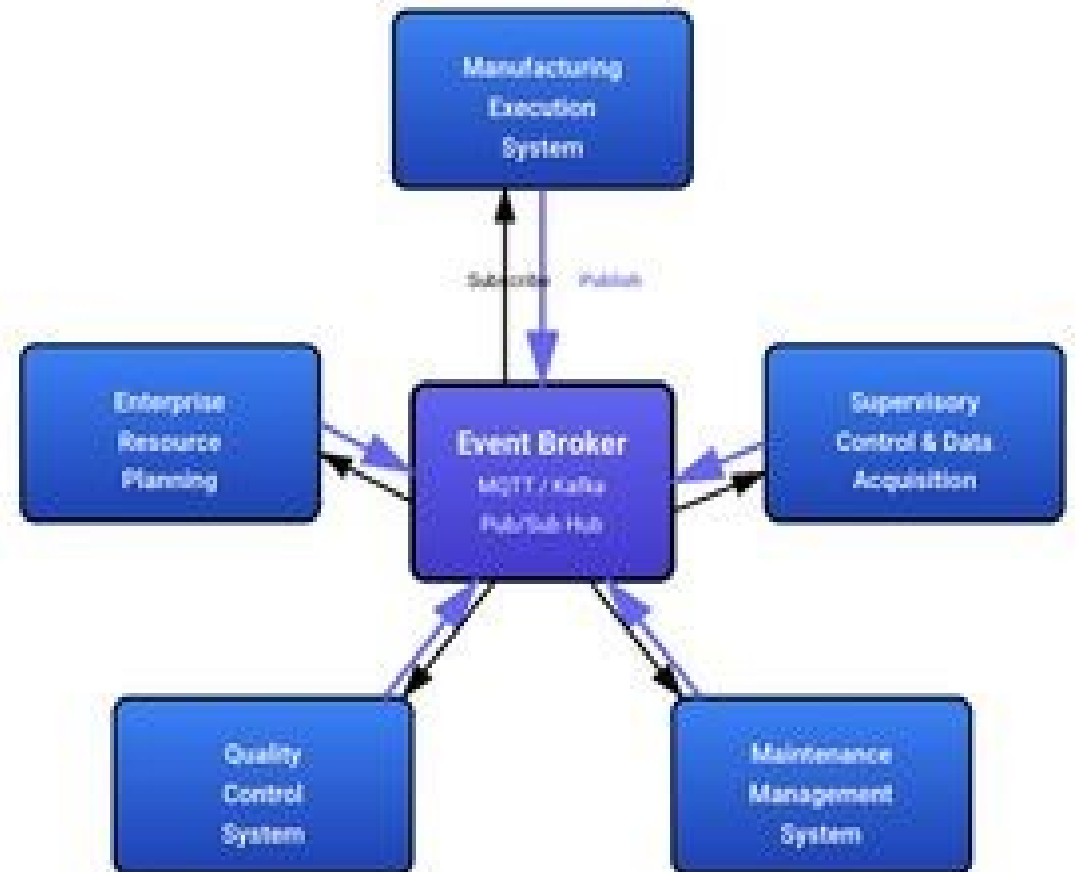
## Traditional Request-Response (Polling)



Timeline: Polling Every 5 Minutes



## Event-Driven Architecture



Timeline: Event Driven Response

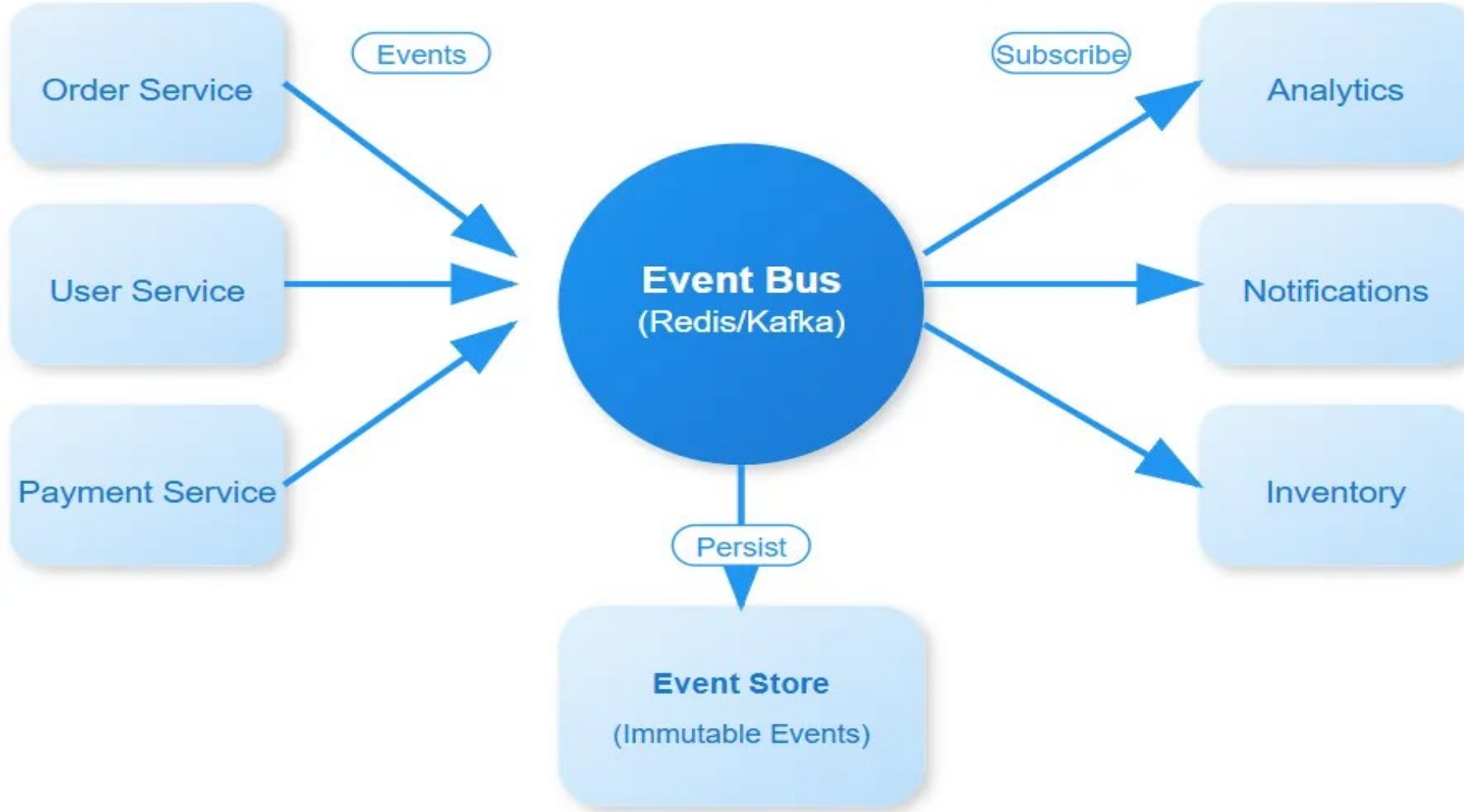


0 seconds delay vs 29-second blind spot

# Polling nedir?

- ❑ Polling tekniđi, basit ve geleneksel olmasına rađmen, ön uç uygulamasını arka uç sunucusuyla uyumlu tutmak üzere kullanılan bir stratejidir.
  - ❖ Polling (yoklama) temelde, sunucudan belirli aralıklarla tekrarlanan veri istekleri gönderilmesidir. Diđer bir ifade ile «herhangi bir deđişiklik oldu mu?» diye sorulmasıdır.
  - ❖ Bu, sunucu durumunda bir deđişiklik olana kadar devam eder.
- ❑ Server-Sent\_Events (SSE) ve WebSockets gibi gerçek zamanlı işlemler sunan daha gelişmiş teknikler ortaya çıkmasına rađmen «polling», güvenilir olması, her düzeye uyumluluđu ve uygulama kolaylığından dolayı birçok uygulamada halen kullanılan bir stratejidir.

## Event-Driven Architecture Components



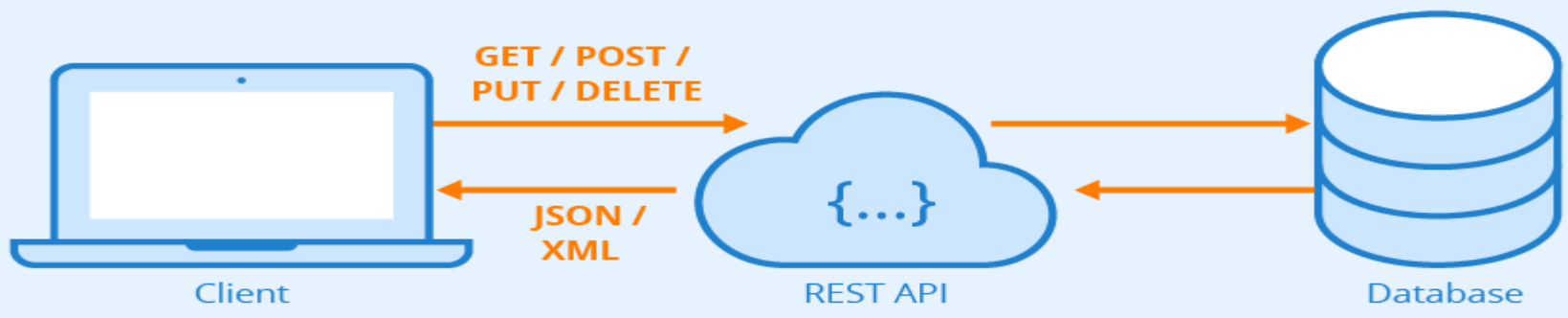
- ❑ Spotify müzik önerileri için günde 4 milyar olay işler.
- ❑ Netflix izleme analizleri için 500 milyar olay işler.
- ❑ Bu süreçlerde sadece veri taşınmayıp, sistemlerin iletişim kurma biçiminde temelden farklı bir yöntem uygulanır.

# Olay Odaklı Mimari ile Geleneksel Mimariler

Özellikler	EDA (Olay Odaklı Mimari)	SOA (servis Odaklı Mimari) - Klasik	MSA (Mikroservis Mimarisisi) - Klasik
Communication (İletişim)	Asynchronous, event-based	Synchronous, often message-based	Synchronous/asynchronous, often REST-based
Coupling (Karşılıklı ilişkiler)	Loosely coupled	Loosely coupled	Loosely coupled
Scalability (Ölçeklenebilirlik)	Asenkron işlemlerde yüksektir.	Servisin tasarımına göre orta düzeydedir	Bağımsız servisin ölçeklenebilirliğine bağlıdır.
Development and Deployment	Zorluğu karmaşıktır; dikkatli olay tasarımı gerektirir.	Standart servis arayüzlerinde orta düzeydedir.	Çoklu servislerin yönetimini gerektirir; karmaşıktır.
Reusability	Orta düzeydedir: Olay işleyicileri (event handlers) yeniden kullanılabilir.	Yüksektir. Servisler yeniden kullanılmak üzere tasarlanır.	Orta düzeydedir. Mikroservisler belli fonksiyonlara özel yazılırlar.
Real-Time Processing	Mükemmeldir. Gerçek zamanlı uygulamalarda idealdir.	Sınırlıdır. Zira talep-cevap (request-response) odaklıdır.	İyidir. Olay odaklı mikro servislerle gerçek zamanlı olarak işlenebilir.
Security	Yüksektir. Güçlü güvenlik ölçütleri gerektirir.	Orta düzeydedir: İmplementasyona göre değişir.	Yüksektir. Güçlü API güvenliği gerektirir.
Observability	Orta düzeydedir. İlave araçlara (tools) gerek vardır.	Orta düzeydedir: External araçlar (tools) ile gerçekleştirilebilir.	Yüksektir. Çoklu servislerle karmaşık olabilir.

# Klasik ve Modern Mimarilerin Karşılaştırması

<b>Özellik</b>	<b>Klasik (Monolitik)</b>	<b>Modern (Mikroservis)</b>
<b>Dağıtım (Deployment)</b>	Tek bir büyük paket	Bağımsız küçük servisler
<b>Ölçeklenebilirlik</b>	Tüm sistemi kopyalayarak	Sadece ihtiyaç duyulan servisi
<b>Teknoloji Yığını</b>	Tek bir dil/framework zorunlu	Her servis farklı dilde yazılabilir
<b>Hata Yönetimi</b>	Tek hata sistemi durdurabilir	Hata sadece ilgili servisi etkiler
<b>Karmaşıklık</b>	Kod seviyesinde karmaşıklık	Operasyonel/Ağ karmaşıklığı



# CGI ve REST (Representational State Transfer)

- ❑ CGI, bir web sunucusunun (Apache) sunucuda çalışan harici bir programla nasıl iletişim kuracağını tanımlayan bir **protokol veya standarttır**.
  - ❖ Kullanıcı, bir sayfa (arama veya form gönderimi gibi) istediğinde, web sunucusu isteği bir script programa (Python, Perl, C++) aktarmak için CGI'yi kullanır.
  - ❖ Script verileri işler ve sunucunun kullanıcıya göndermesi için HTML veya metin geri gönderir.
- ❑ REST, ağ tabanlı uygulamaların tasarımı için bir mimari stil veya bir dizi yönergedir.
  - ❖ Bir yazılım parçası veya belirli bir protokol değildir
  - ❖ HTTP üzerinden verilere nasıl erişileceğinin düzenlenmesinin bir yoludur.
  - ❖ Standart HTTP yöntemleri kullanılarak create, read, update, delete işlemlerinin gerçekleştirileceği bir kaynak olarak ele alınır.

# Derleyici & Yorumlayıcı

- ❑ Derleyici (Compiler) ve yorumlayıcı (Interpreter), programlama dillerinde yazılan kaynak kodun ikili koda (makine diline) dönüştürür.
- ❑ Derleyici (Compiler): Yazılan kodun tamamını ikili koda dönüştürür. Bu sırada hataları kontrol eder ve doğrudan çalıştırılabilir bir dosya (örneğin .exe) oluşturur. Kod bir kez derlendikten sonra, kaynak koda ihtiyaç duymadan defalarca çalıştırılabilir.
  - ❖ C, C++, Rust.
- ❑ Yorumlayıcı (Interpreter): Kodu satır satır veya bloklar halinde okur ve kullanıldığında (at run time) ikili koda dönüştürür.

Çalıştırılabilir bir dosya üretmez; programın çalışması için her seferinde yorumlayıcıya ve kaynak koda ihtiyaç duyulur.

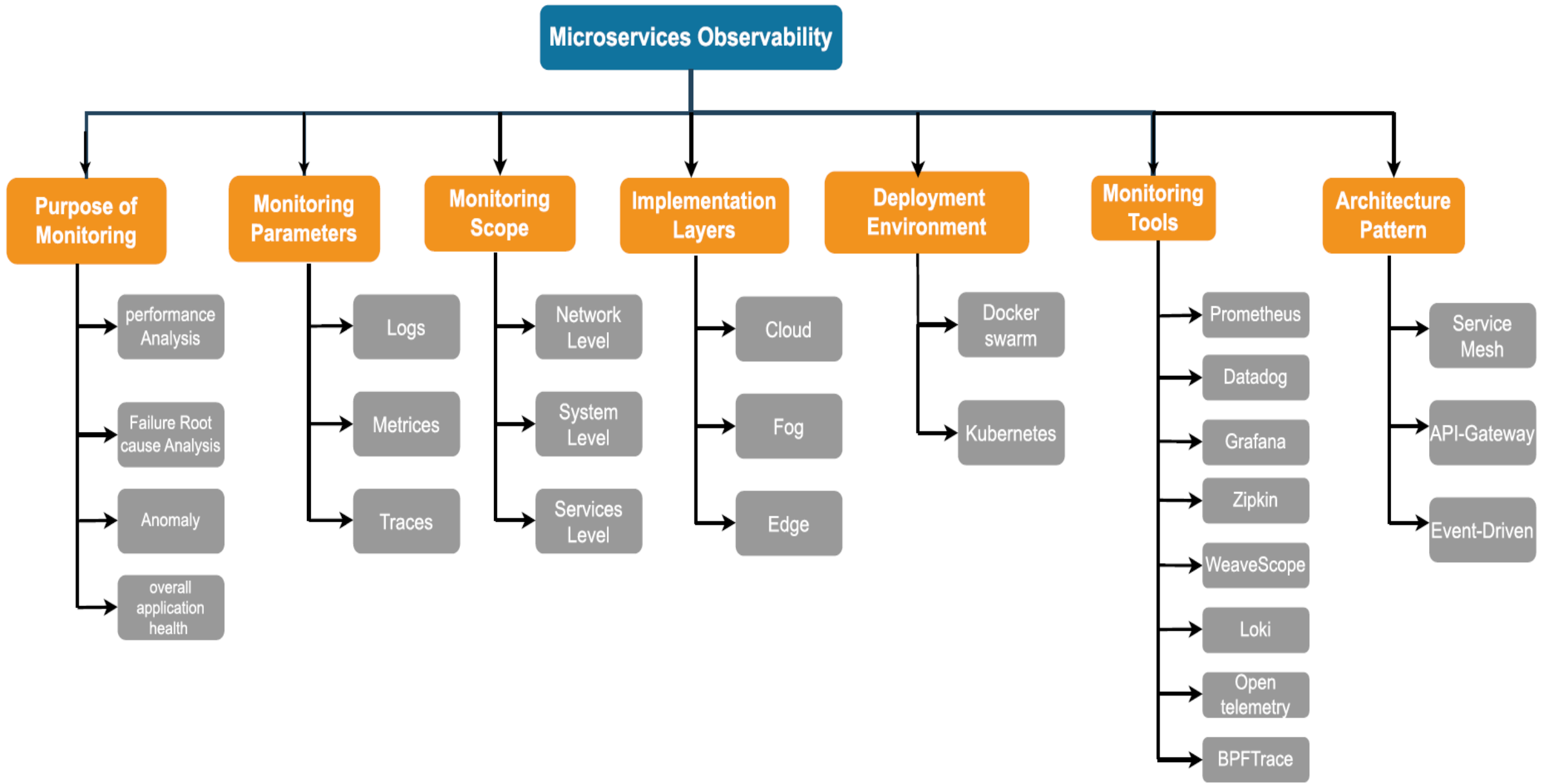
  - ❖ Python, JavaScript, PHP, Ruby.

# Betik (Script) Diller

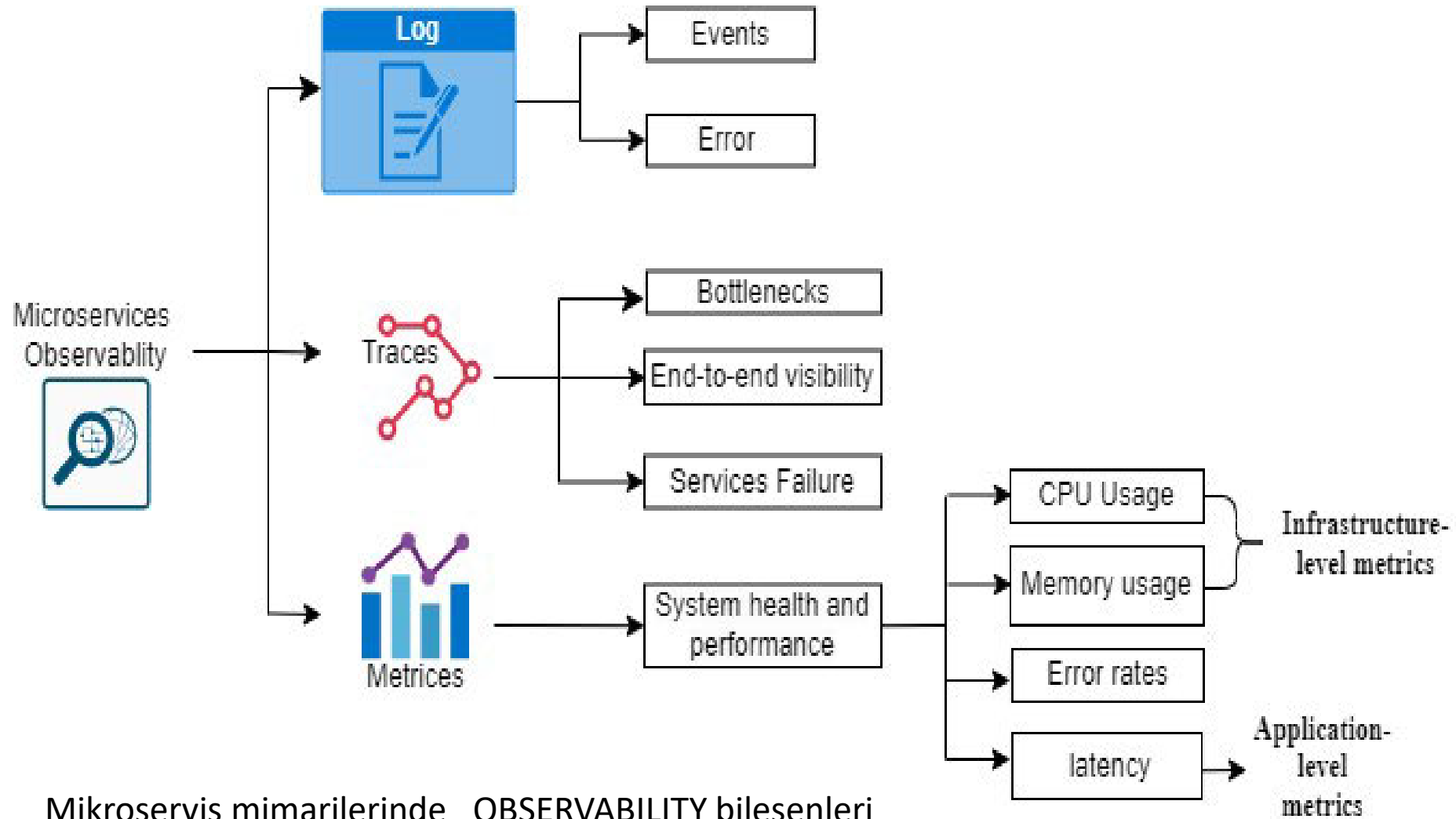
- ❑ Betik (scripting) dilleri, görevleri otomatikleştirmek, sistemleri birleştirmek veya mevcut bir yazılımın içine dinamik işlevler eklemek için kullanılan programlama dilleridir.
- ❑ Bu dillerin en ayırt edici özelliği, kodun bir derleyici (compiler) tarafından makine diline çevrilmek yerine, bir yorumlayıcı (interpreter) tarafından satır satır okunup anlık olarak çalıştırılmasıdır.
  - ❖ Kod yazıldığı anda bir aracı program (interpreter) tarafından doğrudan yürütülür; ayrı bir derleme aşamasına ihtiyaç duymaz.
- ❑ Java Script, Python, PHP, Perl,.....örnekleridir.

# CGI ve REST Karşılaştırması

Feature	CGI (Common Gateway Interface)	REST (Representational State Transfer)
Category	A low-level interface/protocol.	An architectural design style.
Performance	<b>Low.</b> Spawns a new process per request (Resource intensive).	<b>High.</b> Handled by persistent servers or containers.
Data Format	Usually outputs raw HTML for browsers.	Usually exchanges JSON or XML for apps/APIs.
State	Can be stateful (using sessions/cookies).	Strictly <b>Stateless</b> (improves scalability).
Era	Early 1990s (The "Old Web").	2000s to Present (The "Modern Web").



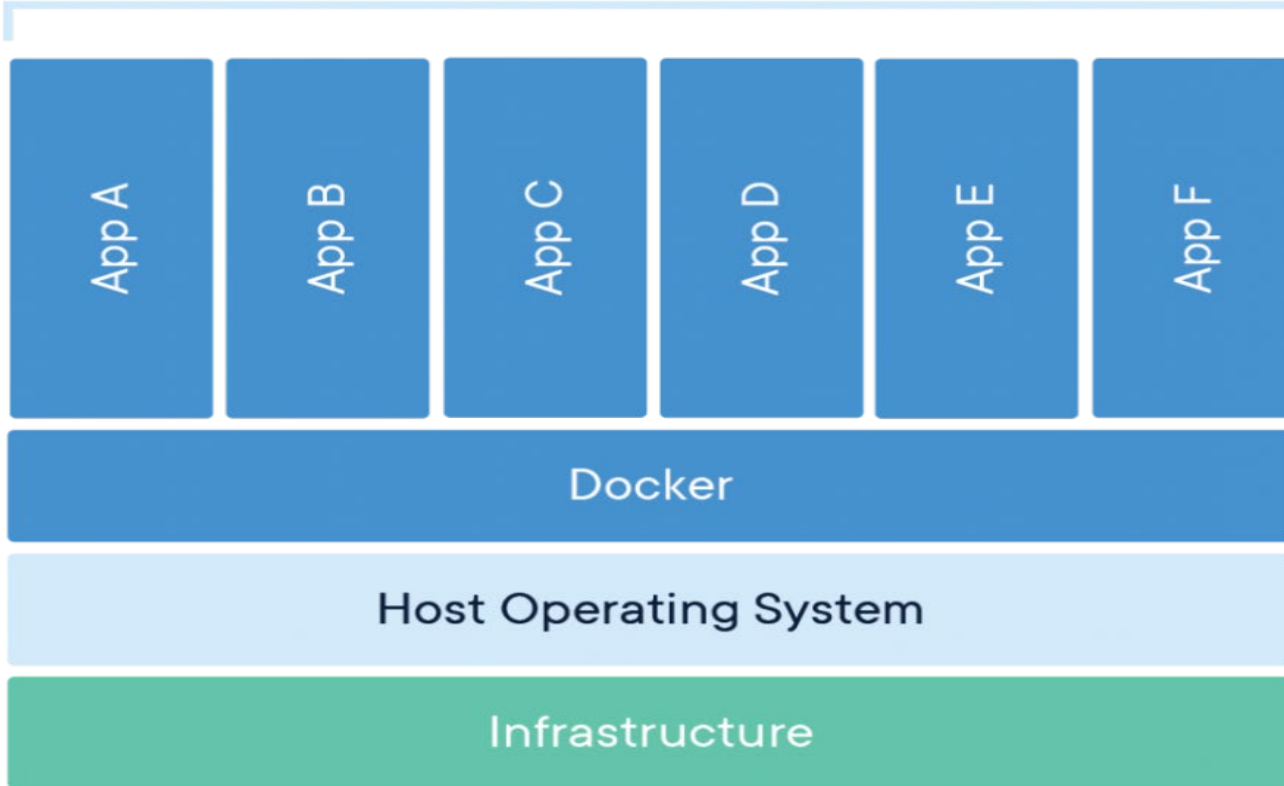
Observability in Microservices: An In-Depth Exploration of Frameworks, Challenges, and Deployment Paradigms, IEEE Access, Digital Object Identifier 10.1109/ACCESS.2025.3562125



# Container nedir?

- ❑ Yazılımın geliştirilmesi, taşınması (shipment) ve dağıtımı (deployment) için standartlaştırılmış birimlere paketlenmesidir.
- ❑ Kodun ve tüm bağımlılıklarının paketlenerek uygulamanın bir bilgi işlem ortamından diğerine hızlı ve güvenilir bir şekilde çalışmasını sağlayan standart bir yazılım birimidir.
- ❑ Docker konteyner, bir uygulamayı çalıştırmak için gerekli her şeyi içeren *lightweight*, bağımsız, çalıştırılabilir bir yazılım paketidir
- ❑ Kod, çalışma zamanını, sistem araçlarını, sistem kütüphanelerini ayarlar.
- ❑ Hem Linux hem de Windows tabanlı uygulamalar için kullanılabilen konteynerleştirilmiş yazılım, altyapıdan bağımsız olarak her zaman aynı şekilde çalışır.
- ❑ Konteynerler, yazılımı ortamından izole eder ve örneğin geliştirme ve test ortamları arasındaki farklılıklara rağmen tekdüze bir şekilde çalışmasını sağlar.

## Containerized Applications



### Lightweight özeliđi:

Konteyner, makinenin iřletim sisteminin ekirdeđini (kernel) paylařır.

Bu nedenle her uygulama iin bir iřletim sistemi gerektirmez. Bylece sunucu verimliliđini artar; sunucu ve lisanslama maliyetleri dřer.

# Gerçek Zamanlı Yanıt Verme

Olay Odaklı Mimaride sistemler programlı bir şekilde soru sormak yerine, olayları dinler ve bir şey olduğu anda tepki verir, böylece olay ve tepki arasındaki boşluğu ortadan kaldırılır.

## Örnekler:

i)Olay, üretimde makinenin bir döngüyü tamamlamasıdır. Örneğin; sensör bir anormallik belirler; parçalardan biri denetimden geçemez; stok bir eşiğe ulaşır.

Bu olaylar, ilgili tüm sistemlere anında bildirilir; otomatik ve paralel olarak yanıt verirler.

ii)Makine bir üretim çalışmasını bitirdiğinde, bir olay tetiklenir; envanter sistemi stok seviyelerini günceller. Bakım, makine saatlerini kaydeder. Kalite kontrol bir denetim planlar. Üretim planlama bir sonraki siparişi verir.

# Geleneksel Üretim Sistemlerinin Yetersizlikleri

- ❑ Geleneksel üretim sistemleri *istek-yanıt* modelleriyle çalışır.
  - ❑ MES (Manufacturing Execution System), envanter sayımları için ERP'yi (Enterprise Resource Planning) sorgular.
  - ❑ SCADA (Supervisory Control and Data Acquisition) sistemlerin sensörlerini sabit aralıklarla sorgular.
  - ❑ Kalite sistemleri, verileri bir araya getirmek için toplu işlemler yürütür. Her sistem, ihtiyaç duyduğu anda ihtiyaç duyduğu veriyi ister.
- Tüm bu sistemler, üretim yavaş ilerlediğinde verimli sonuçlar vermekteydi. Ancak artık çözümlenmeler modern üretim ortamlarında gerçekleştirilmektedir.

# Yetersizlikle ilgili Örnek Vaka

- ❑ Sistemde israf yapısal niteliktedir.
- ❑ Her 100 sorgulama isteğinden yaklaşık 99'u yeni bir bilgi döndürmez.
- ❑ Sorgulama modelleri, aynı veri güncelliğini korumak için olay odaklı alternatiflere göre çok daha fazla sunucu kaynağı tüketir.
- ❑ Bant genişliği ve işlem gücü, hiçbir şeyin değişmediğini doğrulamak için harcanır.

# Yetersizlikle ilgili Örnek Vaka

❑Sorgulama aralıkları arasında belirgin bir risk vardır.

Saat 09:14:01'de üretim hattında bir arıza ortaya çıkar.

Beş dakikada bir sorgulama yapacak şekilde yapılandırılmış izleme sistemi, arızayı 09:15:00'e kadar tespit edemez.

Bu 59 saniye boyunca, kusurlu çıktı sistemde aşağı doğru akarken, örneğin otomotiv tesislerinde maliyetler dakikada 38.000 doları aşan oranlarda artar.

❑Zaman çizelgeleri kayar.

❑Kalite sorunları sistem içerisinde yayılır.

# Yetersizlikle ilgili Örnek Vaka

- ❑ Noktadan noktaya entegrasyonlar sistemin durumunu çok kötüleştirir.
- ❑ Bağlantı sayıları  $n(n-1)/2$  formülünü izler.
  - 5 sistem 10 bağlantı gerektirir. 10 sistem 45 bağlantı gerektirir.
  - Her bağlantı kendi protokolünü, kimlik doğrulama şemasını ve bakım yükünü taşır.
- ❑ Bir ürün yazılımı güncellemesi, düzinelerce uç nokta arasında elle gerçekleştirilmesi gereken yeniden eşleme anlamına gelir.
- ❑ Yeni işlevsellik haftalarca entegrasyon çalışması gerektirir.
- ❑ Tek bir arıza, sistemin bütününde aşağı yönde birbirine sıkıca bağlı her sisteme zincirleme olarak yayılabilir.

# Olay Odaklı Mimari Nasıl Çalışır?

Klasik mimaride her yoklama (polling) sistemi aynı şekilde çalışır.

- ❑ MES(Manufacturing Execution System), bir zamanlayıcıyla uyanır ve bir şey değişip değişmediğini sorar. Genellikle hiçbir şey değişmemiştir. Bu yüzden bekler, sonra tekrar sorar.
- ❑ Event Drivent Architecture (EDA) bu yapıyı tersine çevirir. Her önemli olay, gerçekleştiği anda yayınlanır.

Bunu üretim hattındaki her sistem çiftine uygular ve enerjisinin çoğunu hiçbir şeyin olmadığını doğrulamaya harcayan bir mimari kurulum.