MikroServisler ve Veri Yönetimi

Mikroservisler ve Kurumsal Veri Modeli

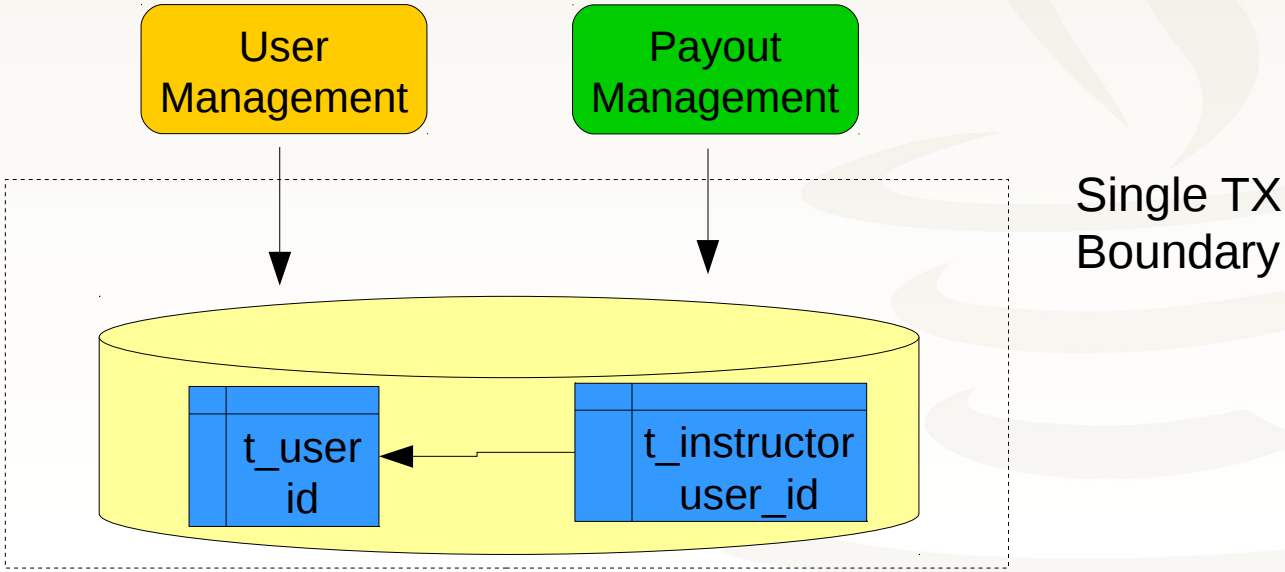
- Mikroservis mimarisi uygulamaların **veritabanı ile etikleşimleri**ni ciddi biçimde etkiler
- Uygulama genelinde tek bir veritabanı veya şema yerine **her bir mikroservisin kendine ait veritabanı** veya şeması olması tercih edilir
- Bu da geleneksel **tek bir kurumsal veri modeli yaklaşımı**na ters düşer

Mikroservisler ve Kurumsal Veri Modeli

- Her bir mikroservis için ayrı veritabanı yaklaşımı çoğunluka **verinin duplikasyonu** ile sonuçlanır
- Senaryoların **veri bütünlüğü**ne yönelik de farklı yaklaşımlar sergilenmesini gerektirir
- Ancak her bir mikroservis için ayrı veritabanı kullanımı mikroservis mimarisel yaklaşımından **maksimum fayda** elde etmek için **kritik önem** arz eder

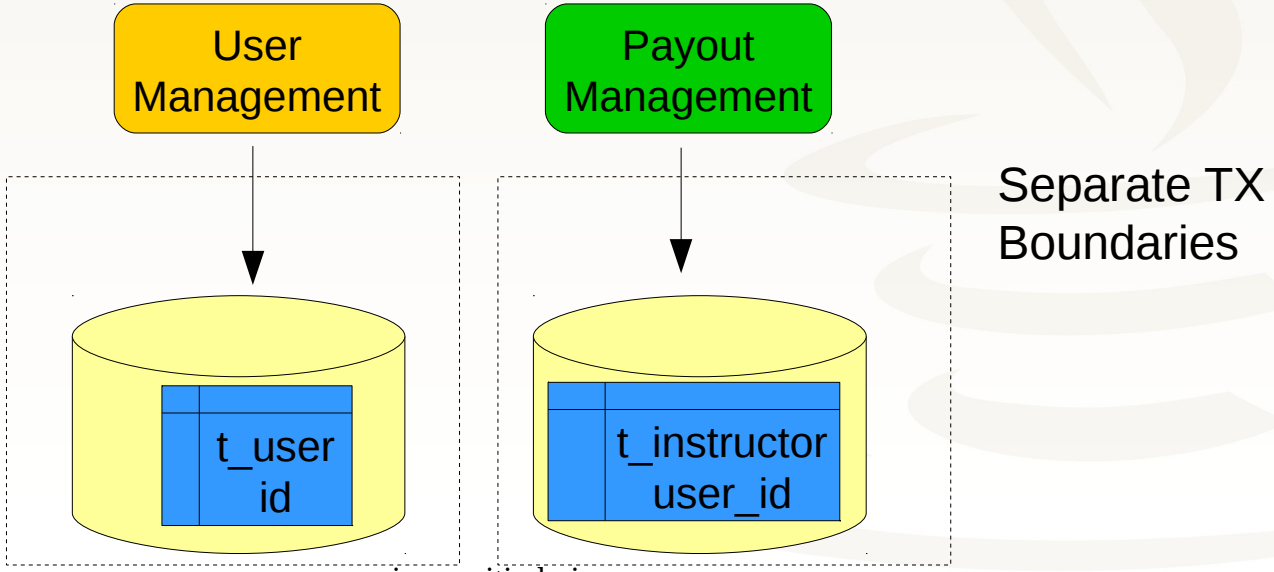
Tek Veritabanı ve Transaction Yönetimi

- Tek veritabanı içinde **TX yönetimi** sayesinde gerçekleştirilen işlemler sistemi belirli **bir consistent state'den, başka bir consistent state'e** taşır
- **Data integrity** kendiliğinden gerçekleşir



Birden Fazla Veritabanı ve Eventual Consistency

- Mikroservislere geçişte veritabanı şemasının ayrılması **transactional integrity'nin sonu** demektir
- Bu durumda karşımıza **eventual consistency** kavramı çıkar



Eventual Consistency

- Data integrity'nin sağlanması, sistemin consistent bir state'de kalabilmesi için **TX boundary kullanılmaz**
- Gerçekleşen işlemler sonucu sistemin ileriki bir zaman diliminde **consistent bir state'e ulaşacağı** varsayılır
- Sistemdeki senaryolar **buna göre** tasarlanır
- Özellikle **uzun süren işlemler** için oldukça uygun bir yaklaşımdır

Compansating Transaction

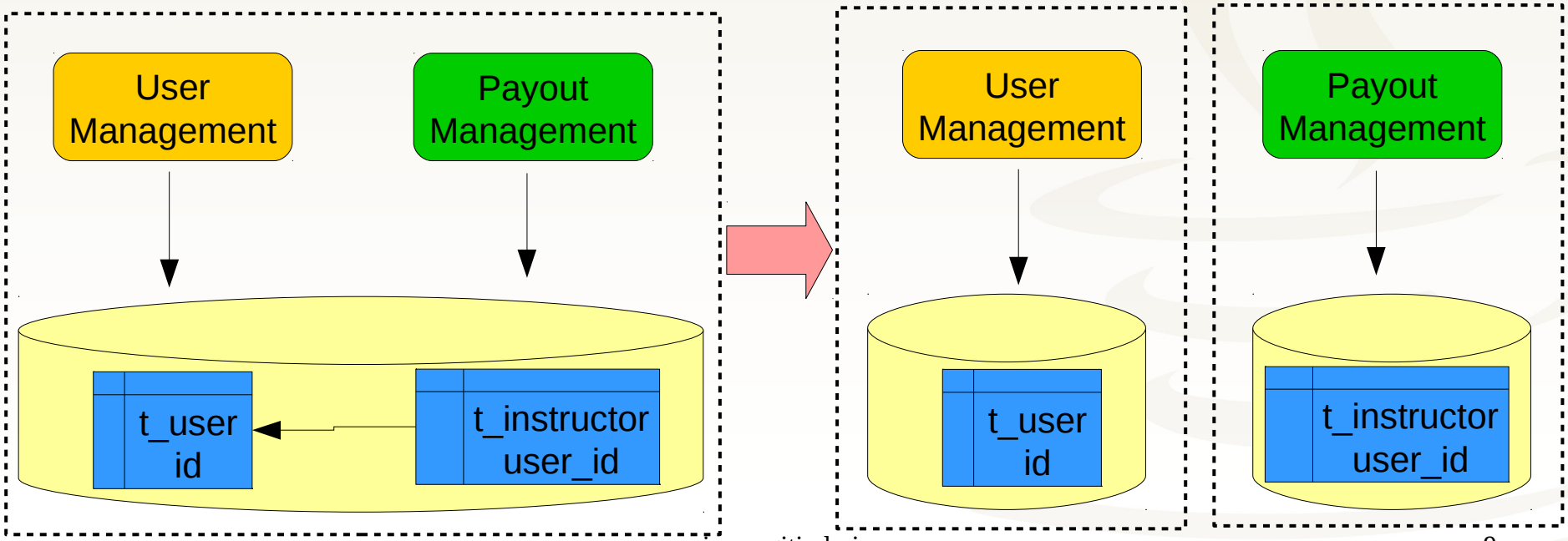
- Eventual consistency yaklaşımında belirli bir takım işlemlerin tam olarak başarısızlığa uğraması sonucu sistemdeki **bir takım işlemleri geri almak** gerekir
- Bunun için **ayrı transactional işlemlerin tetiklenmesi** söz konusudur
- Bu tür transaction'lara **compansating transaction** adı verilir

Distributed Transaction Yönetimi Alternatif Olamaz mı?

- CAP teoremine göre **consistency-availability-partition tolerancy** kabiliyetlerinden aynı anda sadece ikisi sağlanabilir
- Günümüzde kullanılan NoSQL, message broker, dosya sistemi yöneticisi gibi bileşenlerin pek çoğu **distributed TX'i desteklemez**

FK İlişkilerin Yönetimi

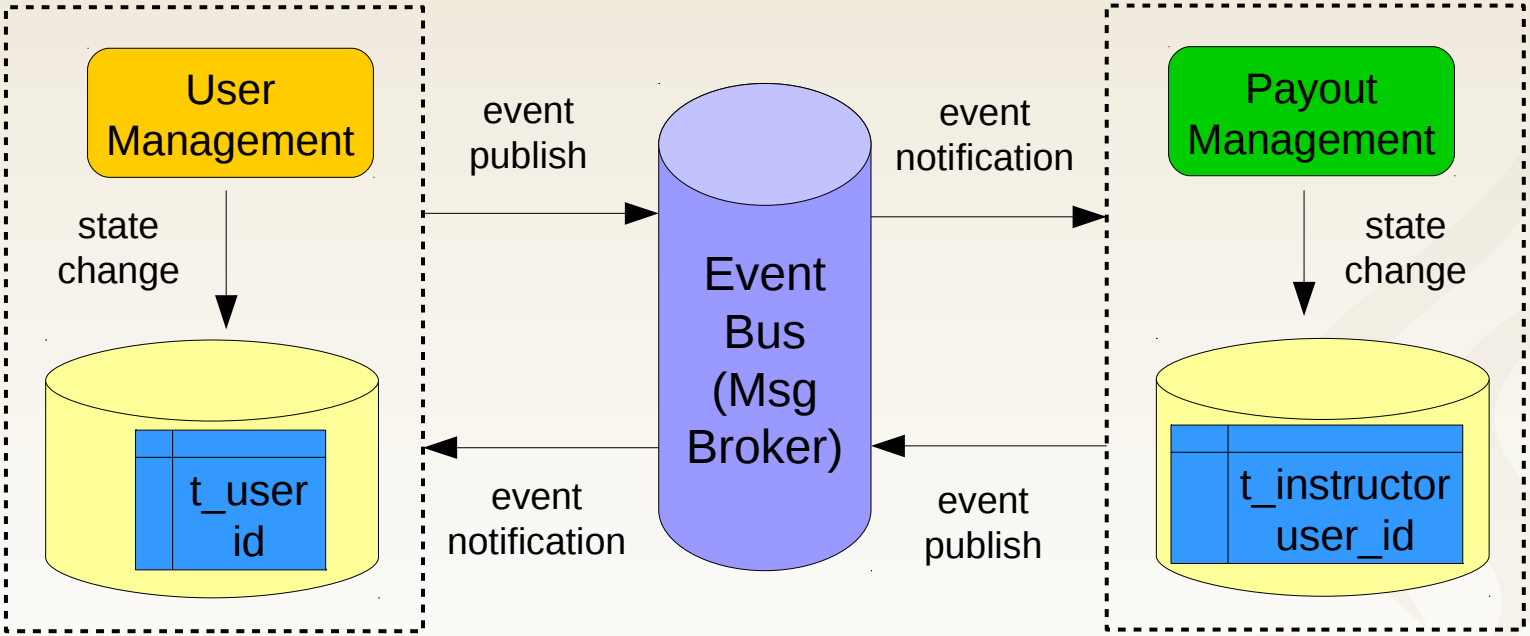
- Bounded Context'ler arası **FK ilişkiler** tamamen **ortadan kalkar**
- Bu tür ilişkiler artık DB düzeyinde değil, **servis düzeyinde yönetilir**



Event Tabanlı Veri Yönetimi

- İş mantığının **birden fazla servis**e yayıldığı senaryoları en efektif gerçekleştirim yöntemi **event tabanlı bir mimari model** kullanmaktır
- Bu mimari modelde servisler herhangi bir **state değişikliği** söz konusu olduğu vakit bunu ilgili diğer **servislere bildiren (notify) event'ler** fırlatırlar (publish)

Event Tabanlı Veri Yönetimi



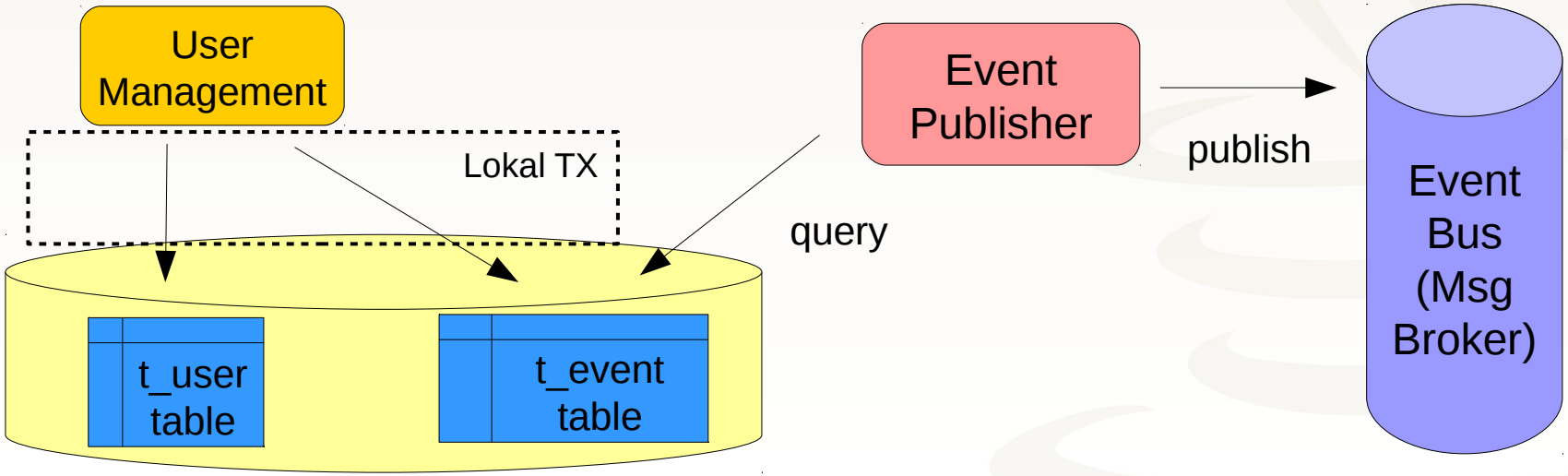
- Burada da state değişikliği ve event publish/notification işlemlerinin **atomik biçimde** yürütülmesi gerekmektedir

Event Tabanlı Veri Yönetimi

- State değişikliği ve event publish/notification işlemlerinin atomik biçimde yürütülmesi için **CAP teoremi ile uyumlu** farklı yöntemler mevcuttur
  - Event publish işleminin **lokal transaction'lar** kullanılarak yapılması
  - DB Transaction **loglarının incelenerek** event publish yapılması
  - **Event sourcing** yaklaşımının kullanılması

Lokal Transaction Yöntemi ile Event Publish

- Bu yöntemde publish edilecek event'ler aynı DB'de **ayrı bir event tablosu**na eklenirler
- Ayrı bir thread, event tablosunu işleyerek **bekleyen event'leri publish** eder

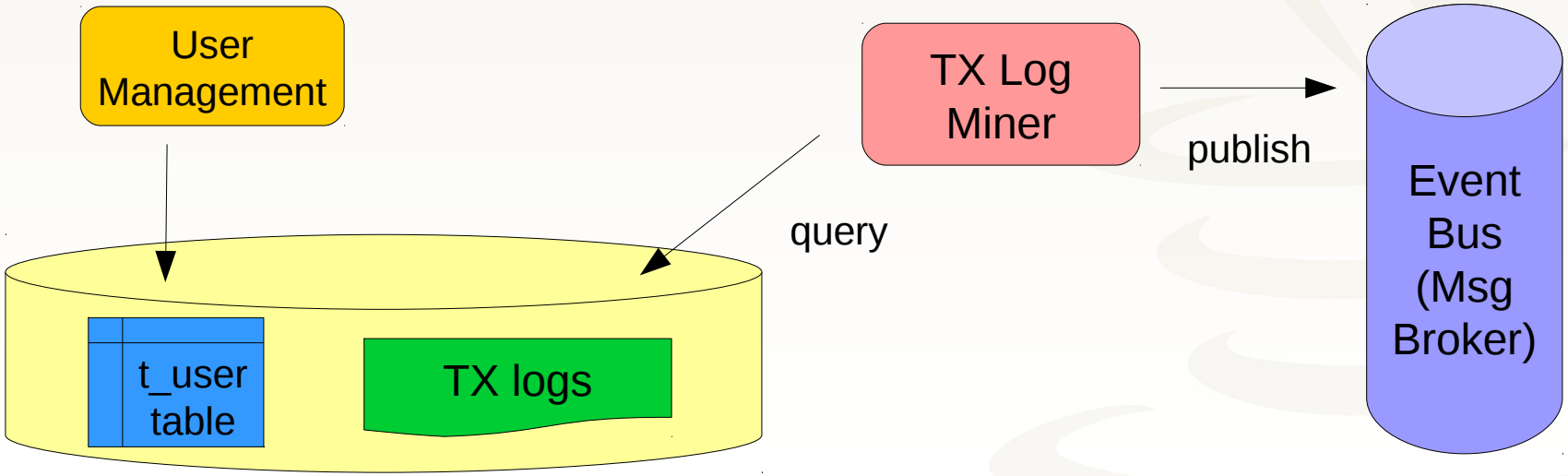


Lokal Transaction Yöntemi ile Event Publish

- Avantajı **distributed TX yönetimine ihtiyaç duymadan** state değişikliği ve event publish işlemlerini gerçekleştirebilmesidir
- Bazı NoSQL veritabanlarında **TX ve query kabiliyetleri sınırlı** olduğundan implement edilmesi zor olabilir
- **Event publish** işleminin geliştiriciler tarafından **explicit yapılması** da diğer bir dezavantajdır

Transaction Log'larını İnceleyerek Event Publish

- Bu yöntemde ayrı bir thread ile veritabanının **TX veya commit logları incelenerek** event publish gerçekleştirilir
- LinkedIn DataBus ve AWS DynamoDB bu yöntemi kullanarak çalışırlar

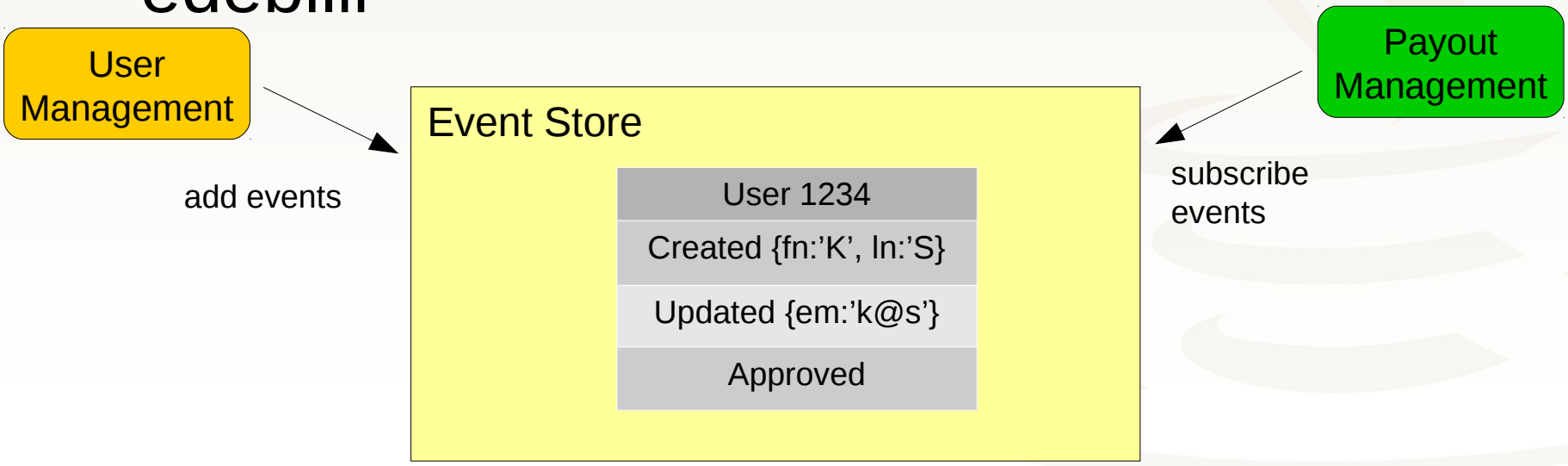


Transaction Log'larını İnceleyerek Event Publish

- 2PC'ye ihtiyaç duymadan **event publish** yapması avantajlarından birisidir
- Uygulama geliştiricinin herhangi bir **explicit işlem yapmaması** da diğer bir avantajıdır
- TX veya commit log'larının yapısının ve formatlarının DB'ler arasında **standart olmaması** en büyük dezavantajıdır
- DB düzeyindeki değişikliklerden **high level business event'leri** türetmek de zor olabilir

Event Sourcing

- Entity state'inin herhangi bir andaki durumu yerine **bu state'e kadar yapılan ve state değişikliğine yol açan event'ler** saklanır
- Uygulama bu event'leri sıra ile işleterek entity'nin **belirli bir andaki state**'ini elde edebilir



Event Sourcing

- Event'lerin persist edilmesi **Event Store** üzerinde tek bir işlem ile gerçekleştiği için bu yöntem **yapısal olarak atomik**tir
- Event Store, **event persist ve event sorgu işlemleri için bir API** sunar
- Event Store ayrıca **message broker** gibi de davranabilir
- Böylece ilgili servisler **event store'a üye olarak** ilgilendikleri event'lerden haberdar olabilirler

Event Sourcing

- **State değişikliklerinde event publish** edilmesi doğal biçimde **atomik** olmaktadır
- Domain nesnelerini persist etmek yerine event'leri persist ettiği için **nesne – veri modeli** (ORM) uyumsuzluk problemi de kökünden hallolmaktadır
- Event'ler domain nesneleri üzerinde gerçekleştirilen işlemlerin **audit kayıtları vazifesini** de görmektedir

Event Sourcing

- Farklı bir programalama modeli olması ve veri yönetimine **farklı bir bakış açısı** getirmesi zorlayıcı noktasıdır
- Faydalı olabilmesi için “command query responsibility segregation” (**CQRS**) yaklaşımı ile birlikte kullanılmalıdır

Ortak Statik Verinin Paylaşımı

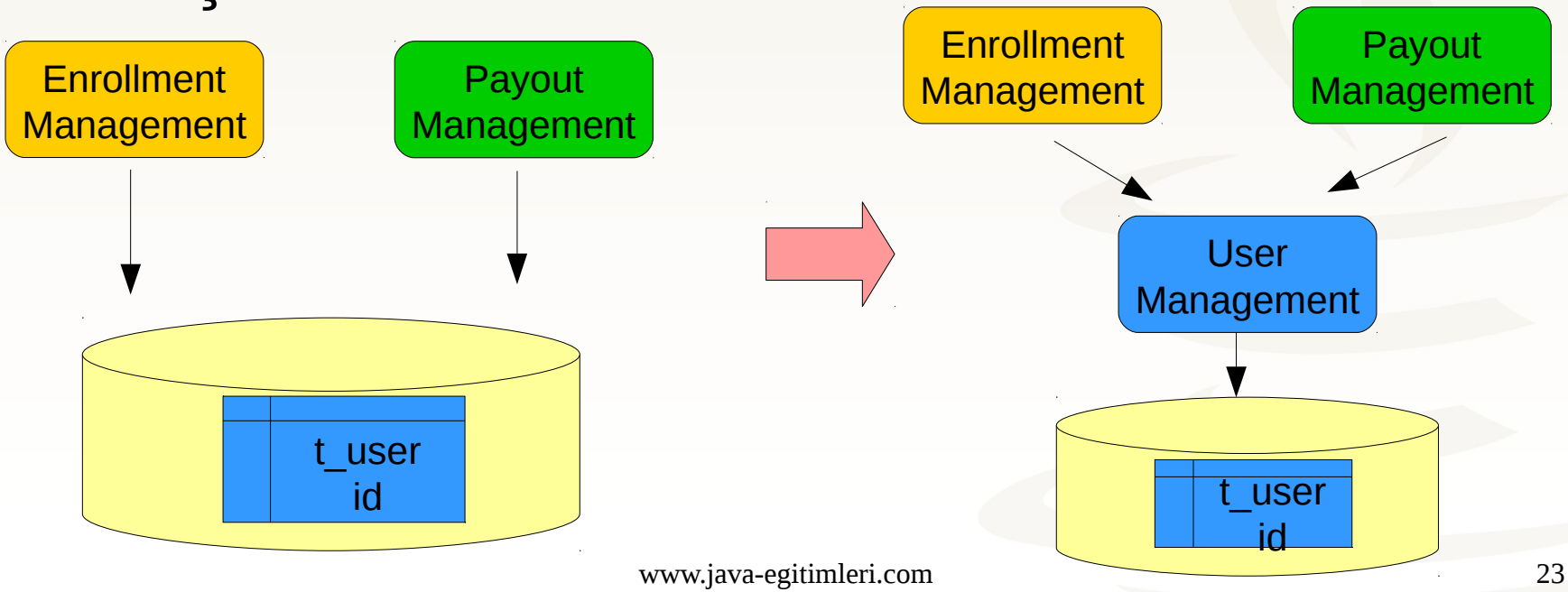
- İl, ülke gibi ortak **değişmeyen/az değişen veri**nin kullanılması için farklı yollar mevcuttur
  - Bu veriler **bütün mikroservis veritabanları**da tekrarlanabilir
    - Dezavantajı duplikasyondur
    - Veri tutarlılığını sağlamak zordur

Ortak Statik Verinin Paylaşımı

- Bu tür veriler **kod içerisinde** veya konfigürasyon property'leri şeklinde yönetilebilir
  - Duplikasyon hala bir sorundur
  - Ancak yönetilebilirlik daha kolaydır
- Diğer bir opsiyon da bu tür verilerin **ayrı bir mikroservis** tarafından yönetilmesidir
  - Çoğu sistem için gereksiz yere çok fazla kompleks bir çözümdür

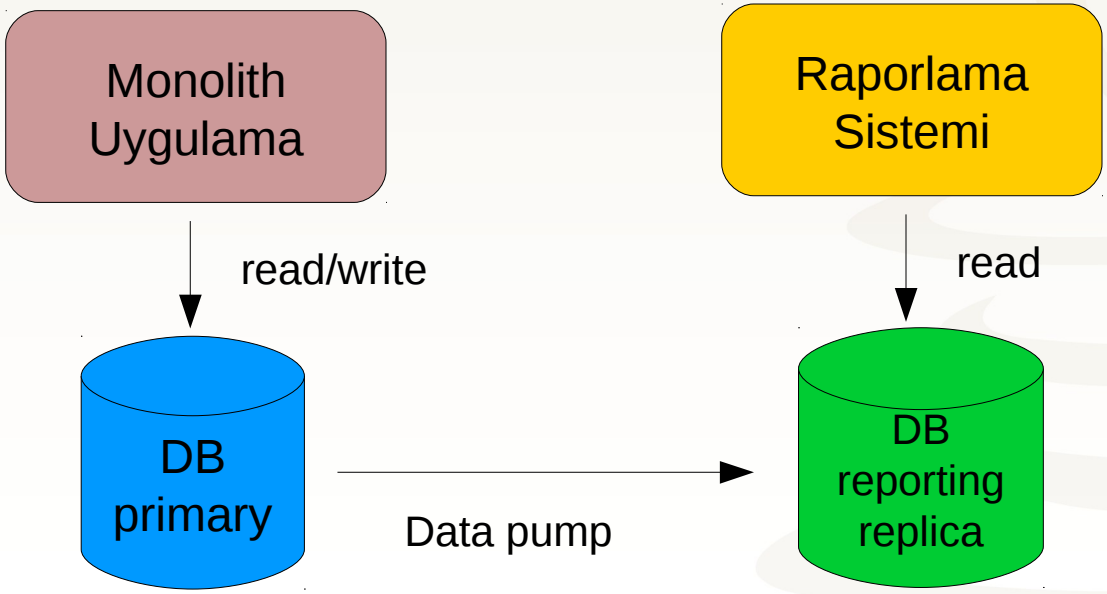
Ortak Dinamik Verinin Paylaşımı

- Birden fazla mikroservis tarafından **ortak** kullanılan ve **üzerinde değişiklikler yapılan dinamik veri**nin yönetimi için en uygun yaklaşım **ayrı bir mikroservis** oluşturmaktır



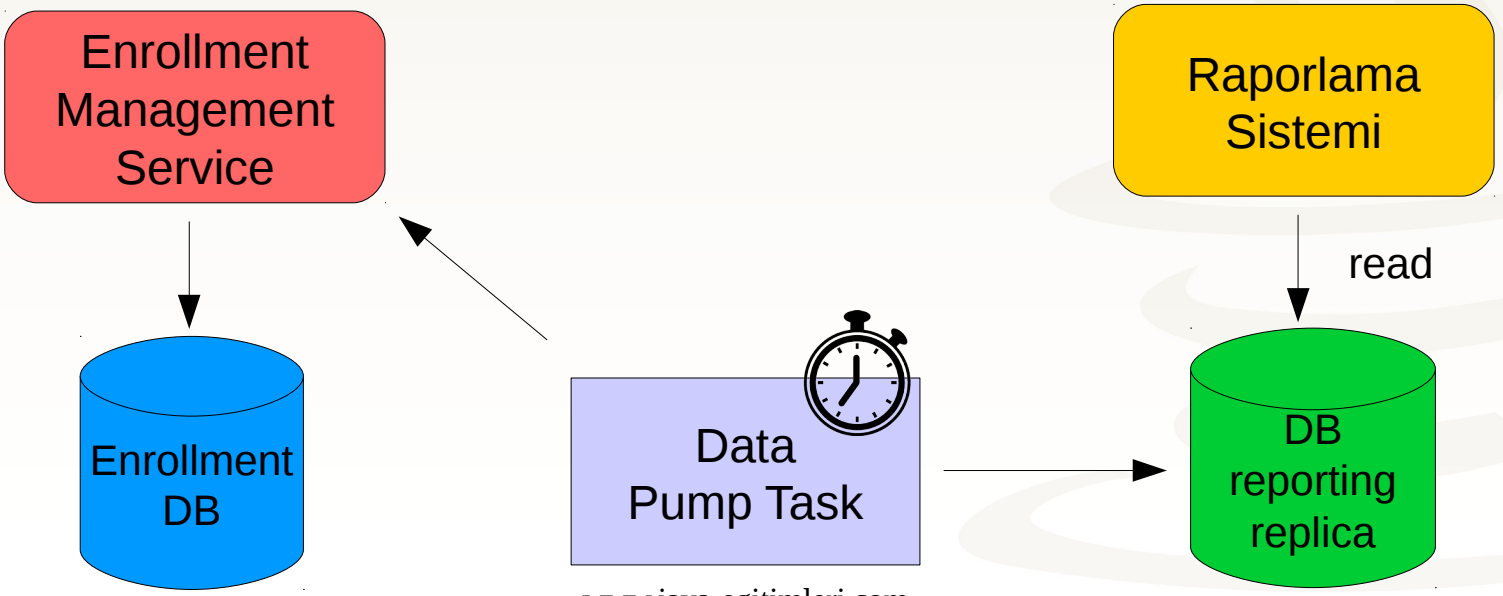
Rapor Verisinin Yönetimi

- **Rapor verisi** genellikle sistemin değişik bölümlerinden bir araya getirilir
- Ayrıca rapor **çıktısına uygun bir yapı**ya dönüştürülür



Rapor Verisinin Yönetimi

- Mikroservislerin ürettiği verinin de “**veri pompa**”sı vasıtası ile raporlama sistemine aktarılması gerekir
- Bunun için ilgili mikroservislerin raporlama için **gerekli veriye de erişim** sunması gerekir



İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com