

AOP Giriş



Aspect Oriented Programlama Nedir?

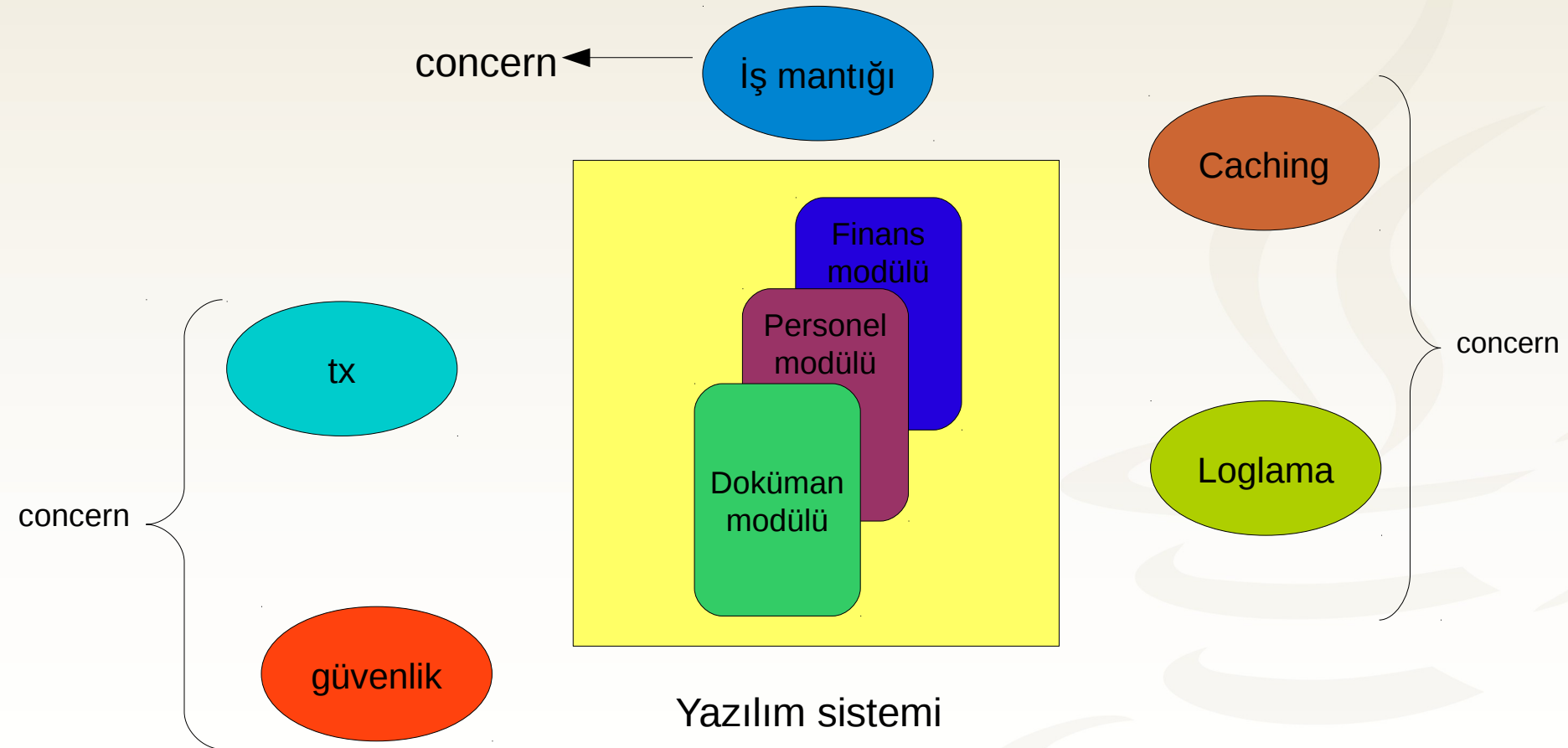
- AOP değişik tip ve nesnelere dağılmış ortak bir **özelliik** veya **davranışın** (**concern**) tek bir yerde ele alınmasını sağlayan programlama modelidir
- OOP'u **tamamlayan** bir yaklaşım sunar, dolayısı ile OOP ile birlikte kullanılır
- Sadece OOP ile çözülemeyen **code scattering** ve **code tangling** gibi problemler en iyi OOP+AOP ile çözülebilir

Concern Nedir?

- Sistem içerisinde sağlanması gereken **bir özellik veya davranıştır**
- İki türüdür
 - Core concern
 - Cross-cutting concern
- Core concern bir modülün temel **/iş mantığı ile ilgili fonksiyonallitesidir**
- Cross-cutting concern ise modüllerin ihtiyaç duyduğu TX, güvenlik, persistence gibi **altyapısal kabiliyetlerdir**

Concern Nedir?

Herhangi bir yazılım sistemi core(business) ve cross-cutting concern'lerin bileşiminden oluşur



OOP'un Temel Problemleri: Code Scattering & Tangling

- Object oriented programlamadaki önlenemeyen iki problem:
 - Code **scattering**: Bir fonksiyonalitenin değişik modüllere dağılması
 - Code **tangling**: İki farklı fonksiyonalitenin iç içe geçmesi
- Sadece OOP yöntemleri, tasarım örüntüleri vb ile bu problemler minimize edilebilir
- Ancak tamamen ortadan kaldırmak mümkün değildir

Code Scattering & Tangling

UserService isimli bir sınıfımız olsun ve createUser isimli bir metodunu implement ediyor olalım. Bu metod içerisinde normal iş mantığının yanında loglama ve tx yönetim işlemleri de yapılıyor olsun.



```
public void createUser(User user) {  
    logger.debug("createUser started");  
  
    TransactionStatus txStatus = transactionManager.getTransaction(  
        new DefaultTransactionDefinition(  
            TransactionDefinition.PROPGATION_REQUIRED));  
  
    try {  
        //user yaratilmasi ile ilgili is mantigi burada yer alir  
  
        transactionManager.commit(txStatus);  
    } catch (Exception ex) {  
        transactionManager.rollback(txStatus);  
  
        throw new RuntimeException(ex);  
    } finally {  
        logger.debug("createUser finished");  
    }  
}
```

Code Scattering & Tangling

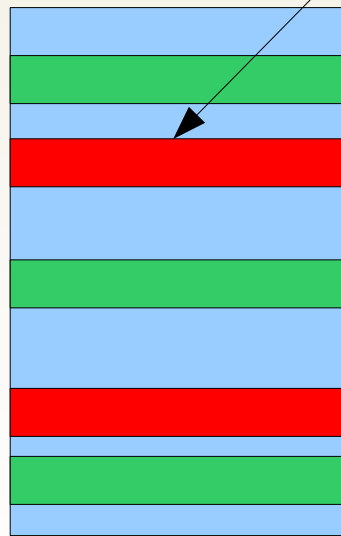
Şimdi de DocumentService isimli bir sınıfımız olsun ve bununda updateDocument isimli bir metodunu implement ediyor olalım. Bu metod içerisinde de normal iş mantığının yanında loglama ve tx yönetim işlemleri de yapılıyor olsun.

```
public void updateDocument(Document doc) {  
    logger.debug("updateDocument started");  
  
    TransactionStatus txStatus = transactionManager.getTransaction(  
        new DefaultTransactionDefinition(  
            TransactionDefinition.PROPROPAGATION_REQUIRES_NEW));  
    try {  
        //doc update ile ilgili is mantigi burada yer alır  
  
        transactionManager.commit(txStatus);  
    } catch (Exception ex) {  
        transactionManager.rollback(txStatus);  
  
        throw new RuntimeException(ex);  
    } finally {  
        logger.debug("updateDocument finished");  
    }  
}
```

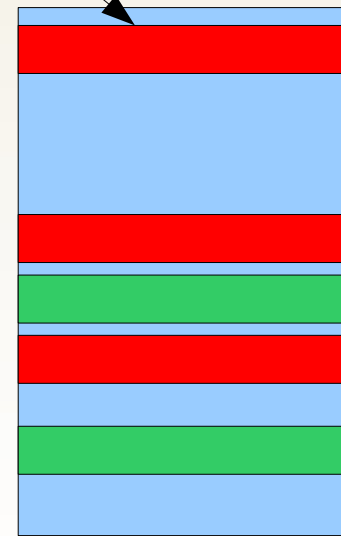
Code Scattering & Tangling

-  : loglama
-  : tx yönetimi

Scattered code : fonksiyonalitenin (concern) değişik modüllere dağılmasıdır



UserService



DocumentService

Tangled code :
Değişik
fonksiyonların
birbiri içine
geçmesidir

Code Scattering & Tangling

```
public void createUser(User user) {
```

```
//user yaratilmasi ile ilgili is mantigi burada yer alır
```

```
}
```

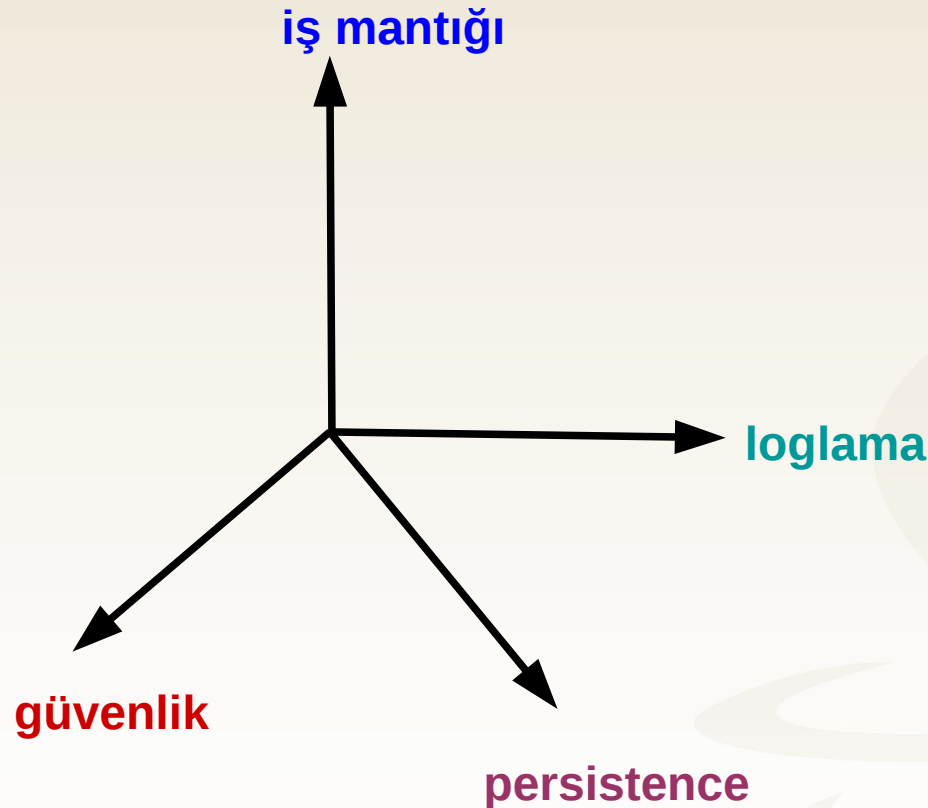
İş mantığı sadece bu kısımdan oluşmaktadır
Diğer kısımlar iş mantığından tamamen bağımsızdır

TX yönetimi, loglama, güvenlik, monitoring, auditing, caching gibi pek çok **altyapısal ihtiyaçlar** en iyi AOP ile çözülür

Code Scattering & Tangling

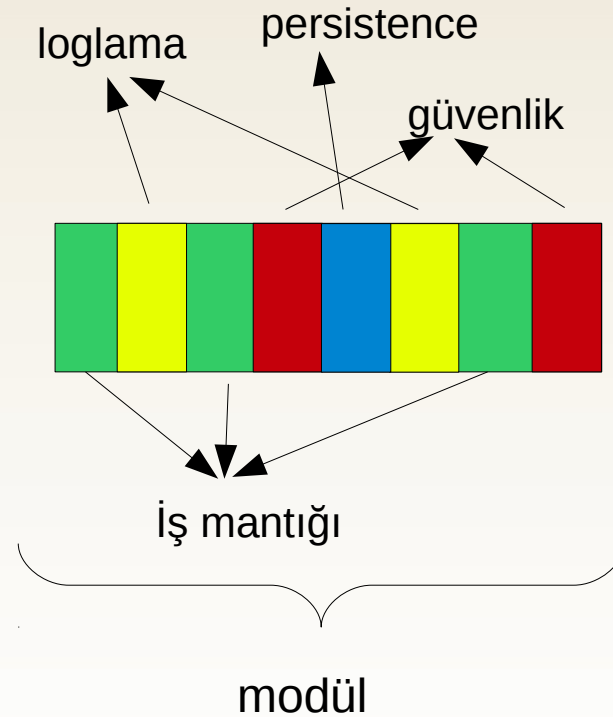
- **Code scattering** iki türüdür
 - Bir fonksiyonalite ile ilgili tekrarlayan (**duplike**) kodların değişik modüllere yayılması
 - Bir fonksiyonaliteyi oluşturan birkaç parçanın sistemin içinde değişik modüllere dağılması
- Doğalarından ötürü **cross-cutting concern**'ler farklı modüllere yayılırlar ve iç içe geçerler

Analizden Gerçekleştirime Geçişte Ortaya Çıkan Problem



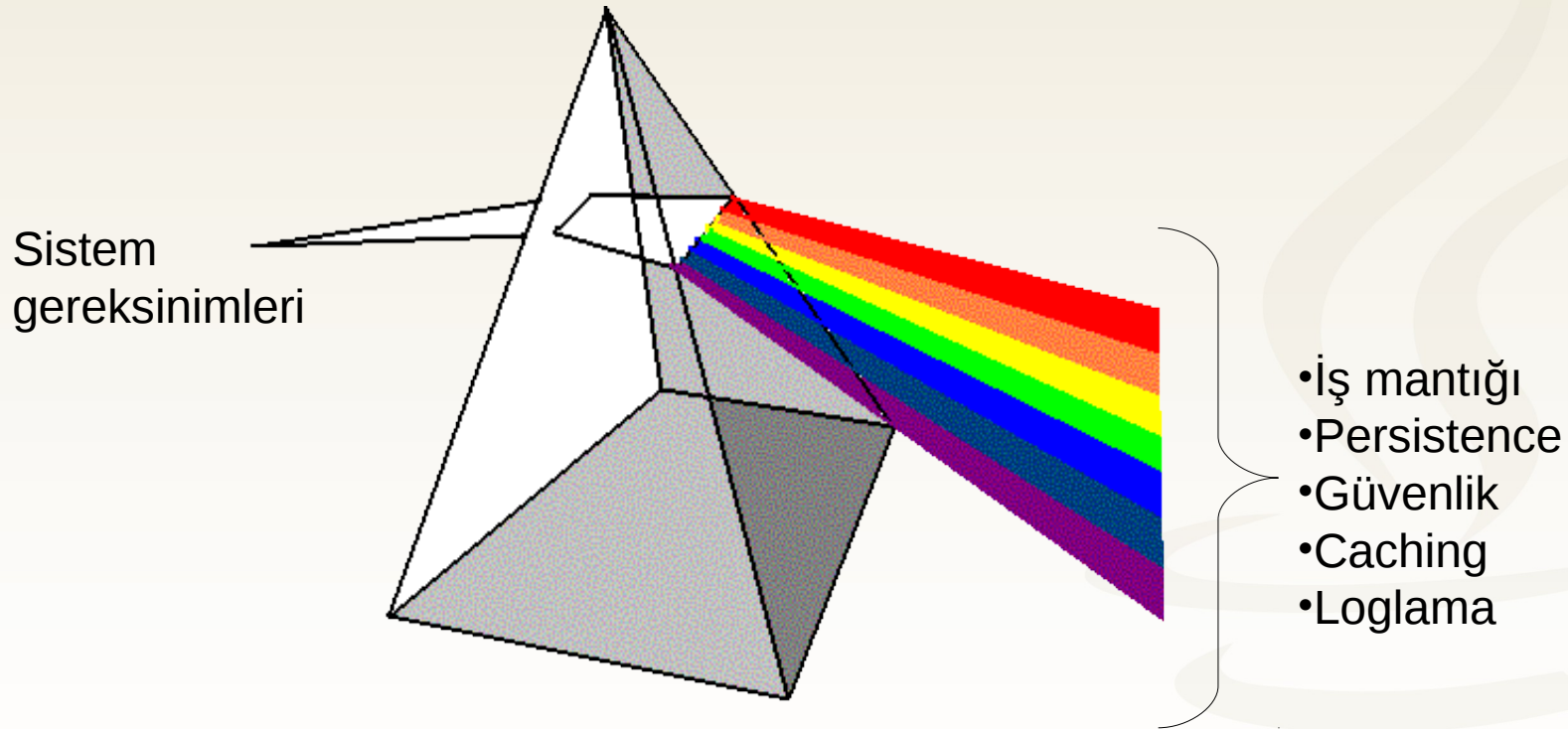
Analiz sürecinde bütün concern'ler birbirinden ayrı modüller biçimde ele alınabilir

Analizden Gerçekleştirime Geçişte Ortaya Çıkan Problem

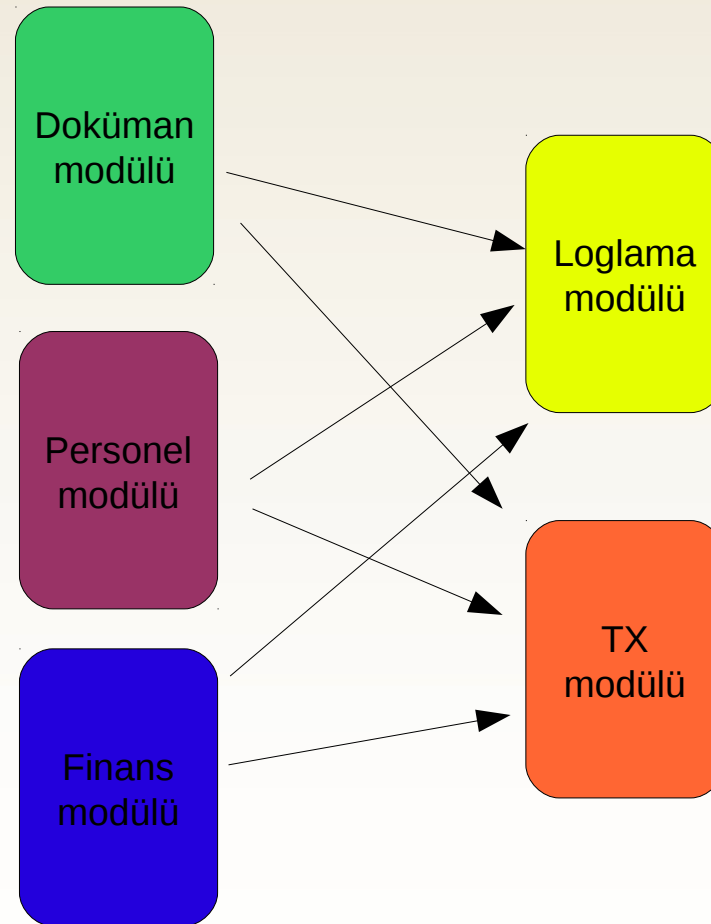


Gerçekleştirim aşamasında ise **code scattering** & **code tangling** problemleri ortaya çıkar

Aspect Bakış Açısı ile Sistem Ayırıştırması



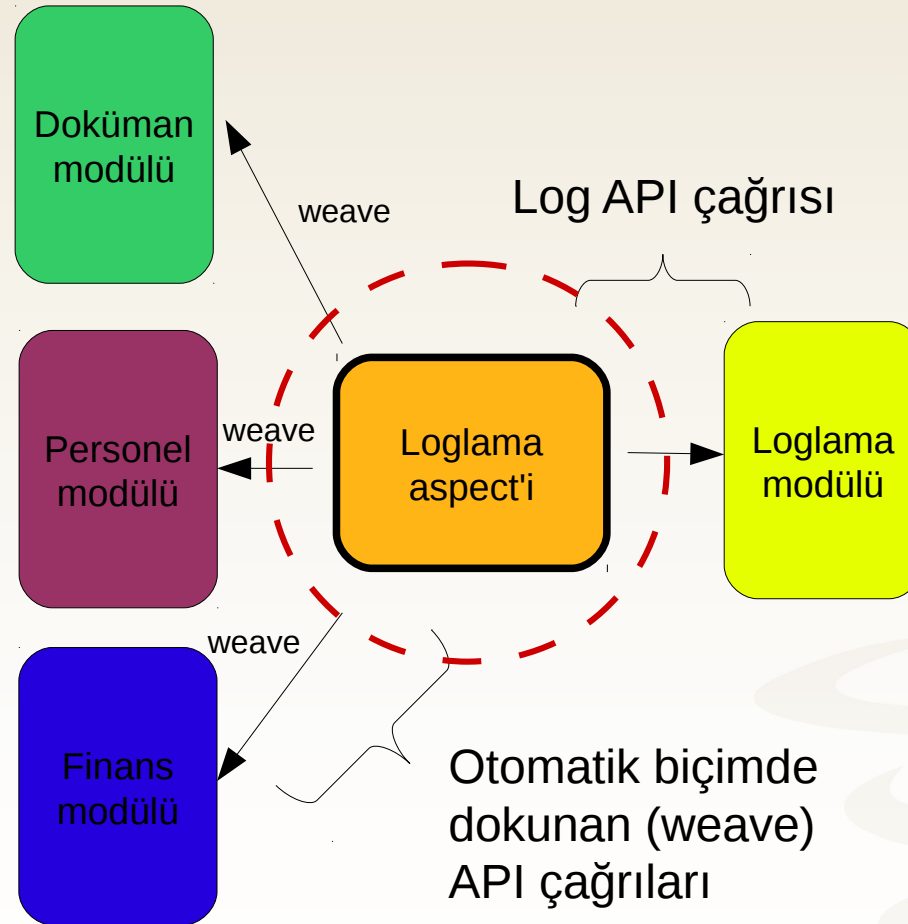
Aspect Bakış Açısı ile Sistem Ayırıştırması



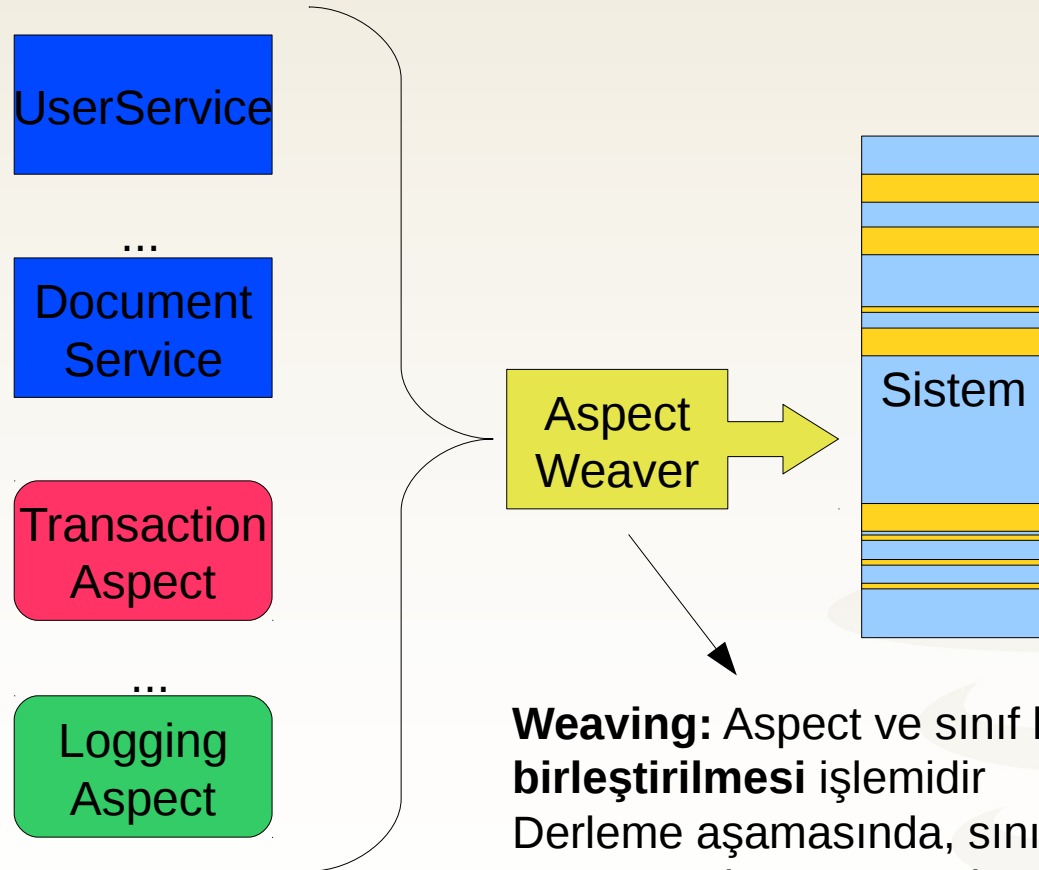
Ayrıştırma/Birleştirme

- Logging, TX gibi altyapısal modüller ile UserService, DocumentService gibi iş modülleri **ayrı ayrı implement** edilirler
- Daha sonraki bir aşamada altyapısal modüllerin fonksiyonallıkları ile iş modüllerinin fonksiyonallıklarının **bir araya getirilmesi** söz konusudur
- Bu işleme **weaving** (dokuma) adı verilir
- Derleme, sınıf yükleme veya çalışma zamanında gerçekleşebilir

Aspectual Ayrıştırma/Birleştirme

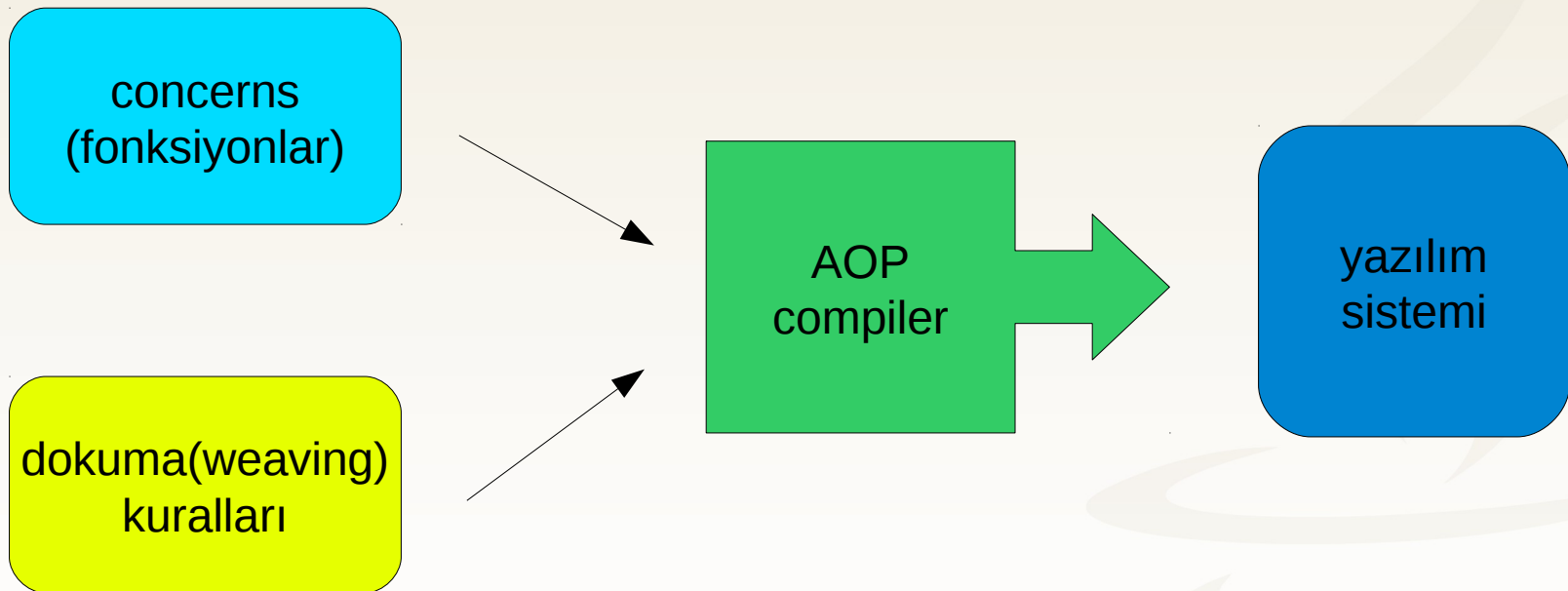


Aspectual Ayrıştırma/Birleştirme

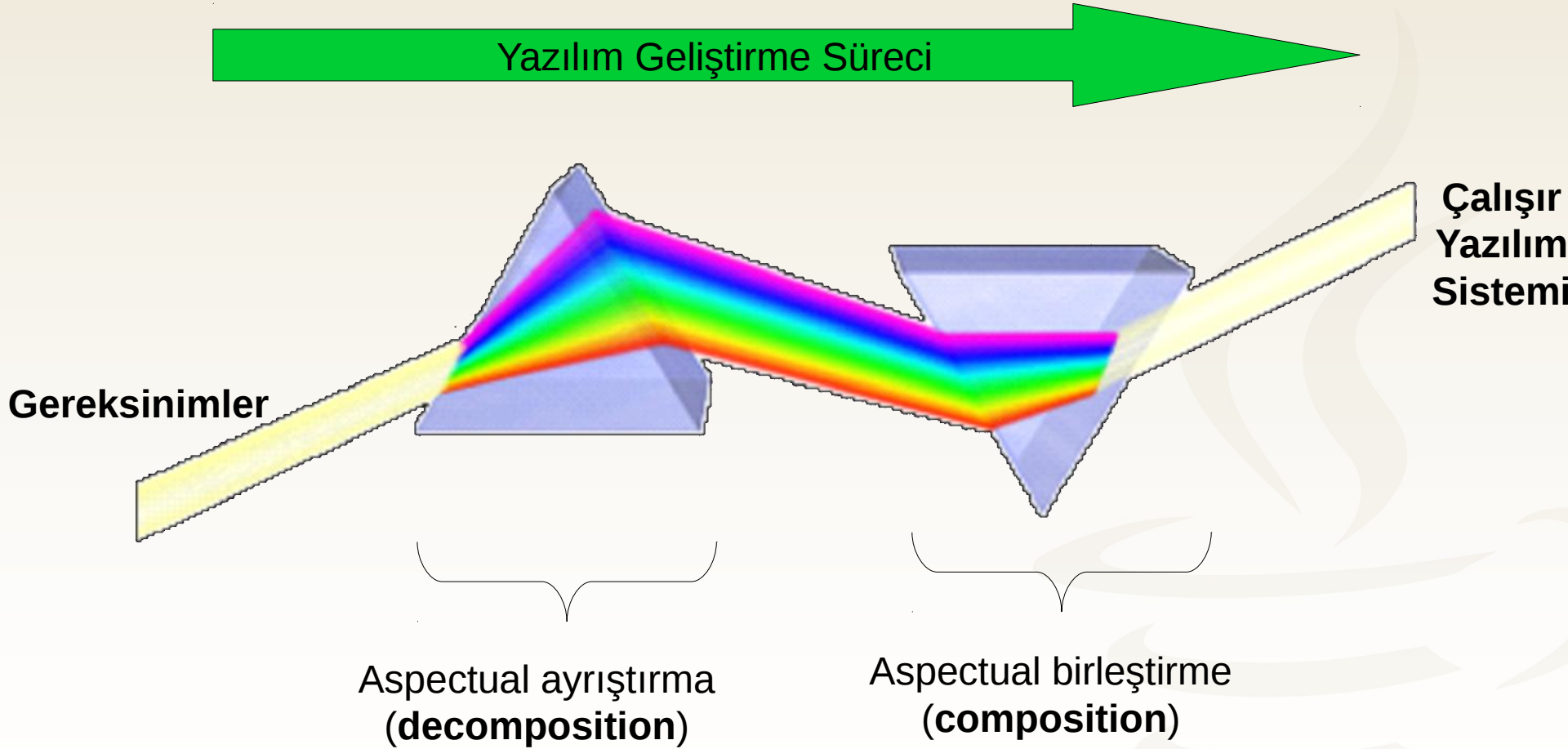


Weaving: Aspect ve sınıf kodlarının tekrardan **birleştirilmesi** işlemidir
Derleme aşamasında, sınıfların yüklenmesi aşamasında, veya runtime'da yapılabilir

Aspectual Ayrıştırma/Birleştirme



Aspectual Ayrıştırma/Birleştirme



AOP'un Faydaları

- En büyük faydası **concern'lerin birbirlerinden ayrılmasıdır**
- Bu sayede sistem **modüler** bir biçimde tasarlanarak implement edilebilir
- Core concern ve cross-cutting concern'ler **ayrı ayrı** geliştirilebilir
- TX, güvenlik, caching, auditing gibi fonksiyonalite'ler farklı sistemlerde **yeniden kullanılabilir**

AOP ile Neler Yapılabilir?

- Sisteme **yeni bir davranış** eklenebilir veya mevcut davranış değiştirilebilir
- Sistemin **statik yapısı** değiştirilebilir
 - Sınıf, interface, attribute, metot vs eklenebilir
- **Derleme zamanında** çalışacak kontroller eklenebilir
 - Derleme hatası veya uyarı verdirilebilir

İletişim

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

