


Web Programlama II

Ders 05: EJB



Slaytlar Hakkında

- Bu slaytlar oldukça yüksek seviyede genel bir bakış açısı sunmaktadır
 - Detaylar Git repository'sindeki yorum satırlarında bulunmaktadır
-



EJB Türleri

- 3 farklı tip bulunmaktadır, sınıflarda @ annotation'ları kullanılır
 - *@Stateless*
 - *@Stateful*
 - *@Singleton*
-

@Stateless

- Kendi state'ine sahip olmaması gereken EJB'ler, ör alanlar
 - ör, *"private int x = 0;"*
 - Hala nesneler inject edilebilir, *EntityManager* gibi
- Teknik olarak, alan değişkenini bildirebilirsiniz ancak Proxy edilmiş EJB'nin her zaman aynı örneği vereceğinin garantisi yoktur
- Belirli bir EJB için Container, bir örnek havuzuna sahip olabilir ve inject edilen bir proxy'yi her kullandığınızda, farklı bir örnekte yöntemi çağırabilir.

@Stateful

- State'i bulunabilir olanlar, ör yerel değişkenler
- *@Stateful* EJB kullanıcıya bağlanır (session gibi)
- Farklı kullanıcılardan çok sayıda isteğiniz (ör. web sayfası ziyaretleri) varsa, her biri için bir EJB örneğiniz olması gerekir.
 - ör, Bir @Stateful EJB kullanan sayfaya istek atan 50.000 farklı kullanıcıya hizmet verebilmesi için 50.000 örneği bellekte tutmanız gerekir
- JEE Container, boş alan bittiğinde EJB örneklerini otomatik olarak diske depolayabilir (ve gerektiğinde devam ettirebilir)
 - Bean'lerin Serializable interface'i kullanması gerekir

@Singleton

- State'i bulunan bir EJB'dir
- Bütün container'da **yalnızca bir örneği** bulunur
- Inject edilmiş bir *@Singleton* her zaman aynı örneği işaret eder
- Aynı singleton farklı thread'ler içerisinde kullanılabilir (ör eş zamanlı web sayfası taleplerini yerine getirmek için), eşzamanlılık sorunlarını önlemek için her yöntem çağrısı proxy sınıfında otomatik olarak synchronized çalıştırılır

Injection

- @EJB ile annotated bir değişken bildirerek başka bir EJB'nin içine bir EJB inject edebilirsiniz.
 - ör, “@EJB private A a;”
 - Hatırlatma: bir EJB’yi «new» ile başlatamazsınız
- Not: Ayrıca @Inject’i de kullanabilirsiniz ancak bu **CDI** (Contexts and Dependency Injection) özelliklerine girmektedir ve yalnızca EJB değil daha genel bir yapıdır
 - Not: CDI özelliklerine bu ders kapsamında bakmayacağız



@PostConstruct

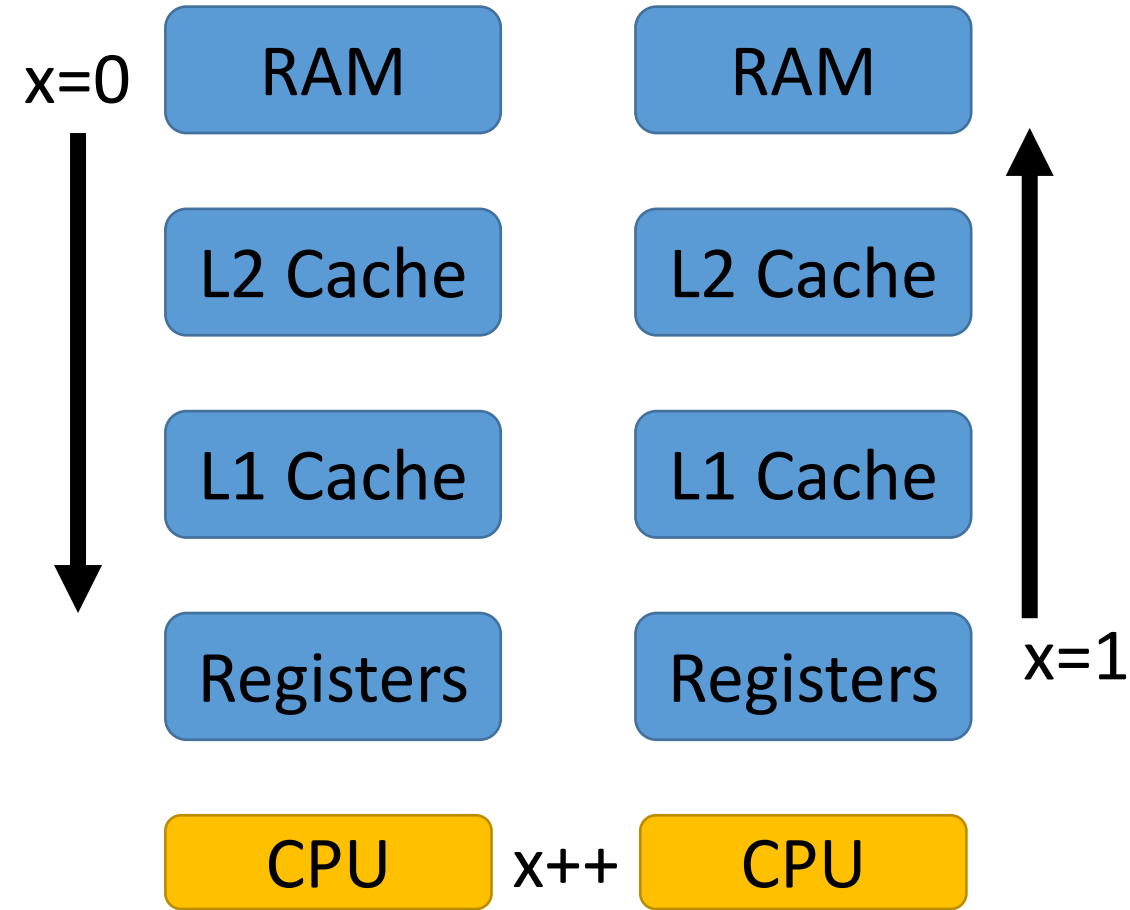
- Container, dependency injection yapmadan önce, "new" ile bir EJB örneği oluşturmalıdır.
 - Bu *constructor* içerisindeki kodların *dependency injection* yapılmadan **ÖNCE** çağırıldığı anlamına gelir
 - Constructor'da inject edilmiş değişkene erişmeye çalışırsanız null pointer exception ile karşılaşabilirsiniz
 - @PostConstruct ile işaretlenmiş metot ise constructor ve DI'dan sonra çağırılacaktır
 - böylelikle, inject edilmiş değişkene bağımlı kodlar çalışabilir
-

Multi-Threading

- *WildFly* gibi sunucular thread havuzuna sahiptir
 - Gelen her HTTP isteği, muhtemelen 2 veya daha fazla CPU'da paralel olarak farklı bir iş parçasığı tarafından işlenebilir.
 - Farklı thread'ler aynı veriler üzerinde çalışırken sorun ortaya çıkmaktadır
 - örnek, *@Singleton*'daki state
 - aynı process'teki thread'lerin aynı heap'i paylaştığını, yani nesnelerin ve state'lerin new anahtar sözcüğüyle bildirildiğini, ancak her thread'in kendi method-call-stack'ine sahip olduğunu hatırlayın
-

CPU ve Cache

- Bir thread'in x değişkenini değiştirdiğini varsayalım, ör, $x++$ yaparak
- x 'in RAM'den CPU kayıtlarına kadar yüklenmesi gerekir
- Kayıtlardaki değişikliklerin RAM'e geri yayılması biraz zaman alabilir.

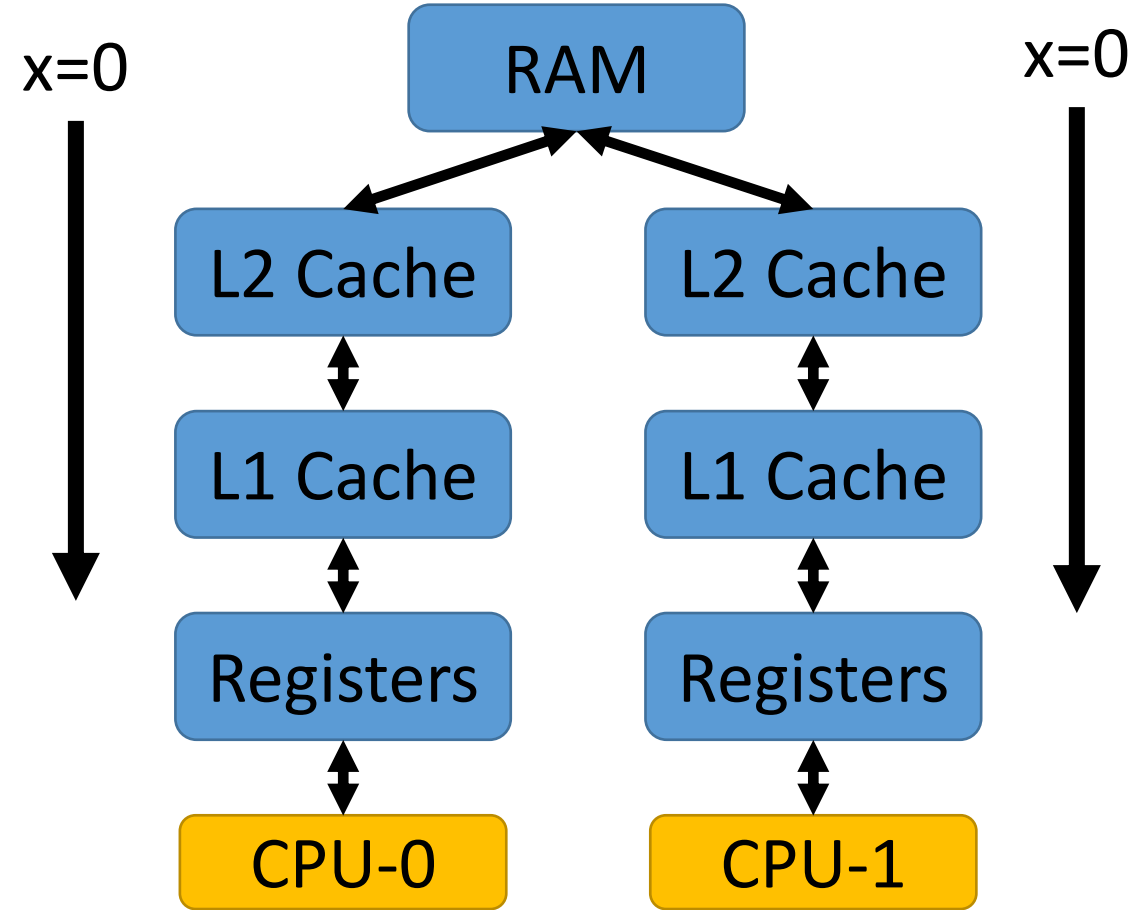


Performans

- *RAM*'ler büyük yapılardır ancak pahalı ve yavaştır
- *Caches* ise çok daha hızlı ancak küçüktür
- Bir hesaplama işlemi verileri kullanır ve bu tür verilerin CPU'ya mümkün olduğunca yakın olmasını isteriz.
- Ancak kayıtlar/önbellekler, hesaplama için gereken tüm verileri tutamaz (ör, thread tarafından yürütülen kod)
- Bu nedenle, gerekli verileri almak için önbellekler arasında sayfa değiş tokuşu yapılır
- Eğer veri hali hazırda cache'deyse tekrar RAM'den yüklenmez

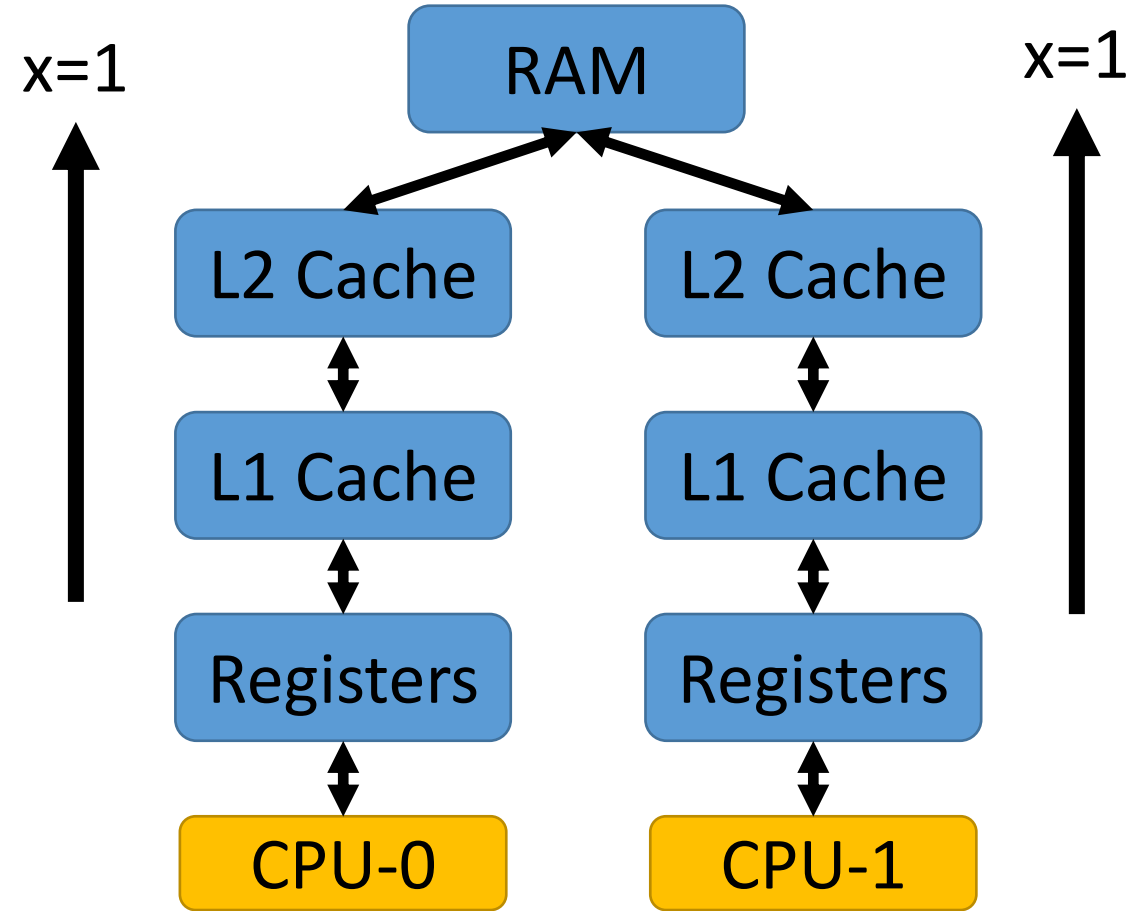
Sorun: 2 CPU'da 2 Thread

- İki thread'in aynı heap'teki **x** değerini okuduğunu var sayalım
- İki CPU'daki register'lar aynı değeri görecektir, ör **x=0**
- Ancak CPU-0 **x++** yaparsa?
 - CPU-1 bu değeri göremeyecektir ve cache'deki **x=0** ile işlem yapacaktır
- Peki her iki CPU da **x++** yaparsa?
 - RAM'i yalnızca biri etkileyecektir



Devam...

- Thread'ler aynı anda çalışacak olursa, her ikisi de x 'i 0'dan 1 yapacaktır
- Cache'de $x=1$ olacaktır ve daha sonra RAM'e yayılacaktır
- Ancak 2 thread tam olarak aynı anda çalışmazsa, sonuç $x=2$ olabilir, çünkü ikinci thread güncellenmiş $x=1$ değeri ile işlem yapar

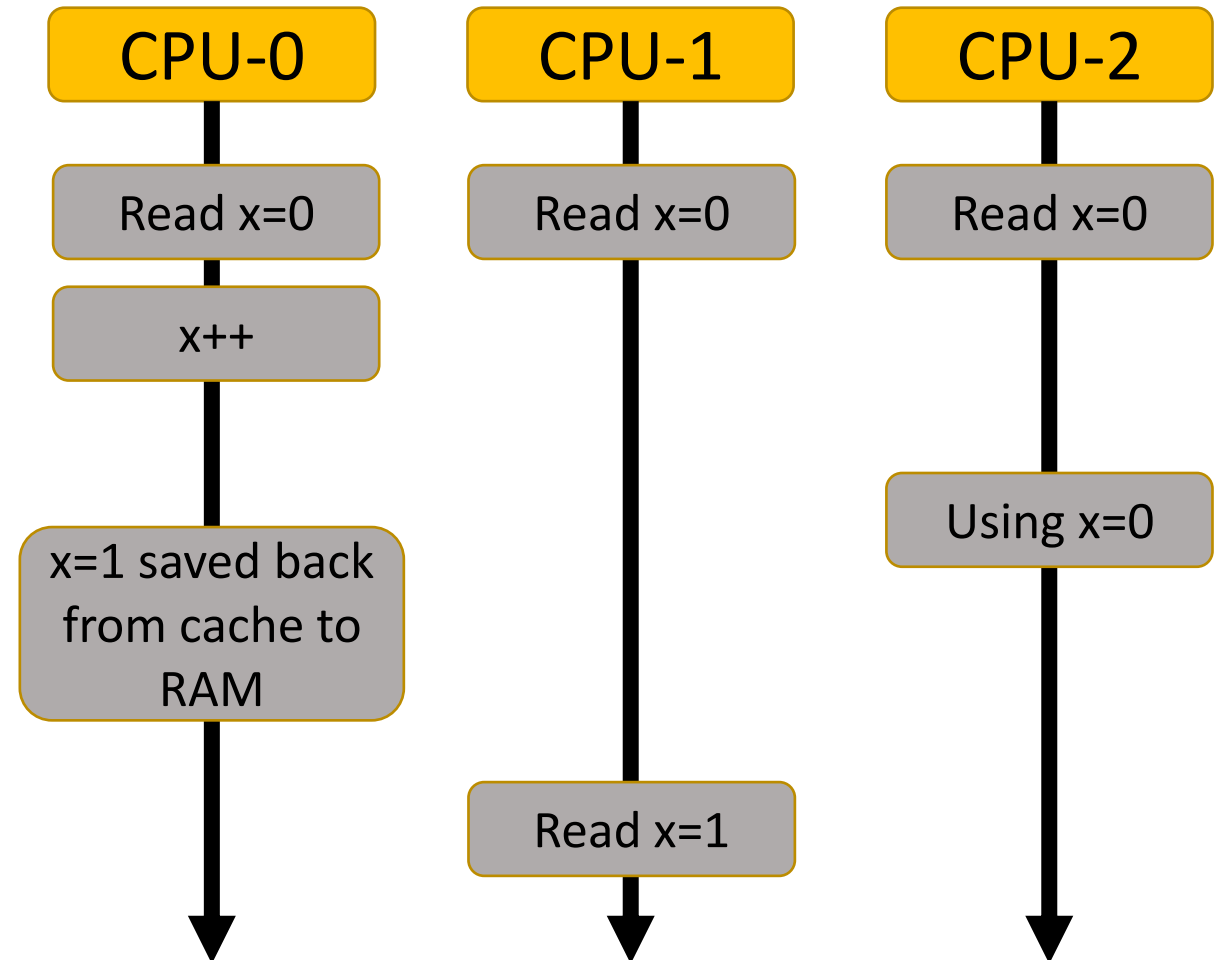


Volatile

- Java'da, değişkenler **volatile** anahtar kelimesi ile oluşturulabilir
 - ör, **volatile int x = 0;**
- A volatile değişkenler her zaman RAM'den okunur ve cache kullanmazlar
- RAM'deki güncel değişimleri alabilmek için faydalıdır
- Okuma/yazma atomik olmadığı için bu tür değerleri yalnızca okumanız ve yazmamanız gerektiğinde iyidir

Volatile Problemleri

- x 'in *volatile* olduğunu varsayalım
- 3 CPU'daki 3 thread aynı anda RAM'den $x=0$ değerini okusun
- CPU-0, $x++$ yapsın
- CPU-1 bu değeri CPU-0 değiştirip RAM'e kaydettikten sonra okusun: $x=1$ değerini okuyacaktır
- CPU-2, $x=0$ değerini kullanacaktır çünkü *volatile* olsa bile CPU-0 did $x++$ işlemini yaptıktan sonra RAM'i güncellemedi



Atomicity

- Farklı thread'ler tarafından paylaşılan bir değişken değerini OKUMAK ve YAZMAK istersek *atomic* olmalıdır
 - ör, Thread **x++** yapacaksa bir başka thread'in bu değeri okuyup kullanmasına izin verilmemelidir
- Java'da kod bloğunu atomic olarak çalıştırmak için **synchronized** anahtar sözcüğü kullanılır. Böylelikle nesne kilitlenir ve işlem tamamlanana kadar başka işlem tarafından kullanılmasına izin verilmez
- Bir başka thread bu kodu kullanmak isterse kilit açılana kadar beklemek zorundadır
- *Problem*: Thread beklemek ve context-switch işlemleri hesaplama maliyeti olarak oldukça yüksektir

Git Repository Modülleri

- *NOT: açıklamaların büyük bir çoğunluğu kod içerisinde yorum satırı olarak bulunmaktadır, burada slaytlarda bulunmamaktadır*
 - **intro/jee/ejb/singleton**
 - **intro/jee/ejb/arquillian**
 - **intro/jee/ejb/multithreading**
 - **intro/jee/ejb/stateful**
 - **intro/jee/ejb/callback**
 - Ders 05 alıştırması (dokümantasyona bakınız)
-