9 Çok İşlemcili / Çok Çekirdekli / Çok Bilgisayarlı Sistemler (*Multiprocessor / Multicore / Multicomputer Systems*)

Sistemlerin performansını iyileştirmek, bazı durumlarda da erişilebilirliğini (*availability*) arttırmak için çok sayıda işlevsel birim paralel olarak çalıştırılır.

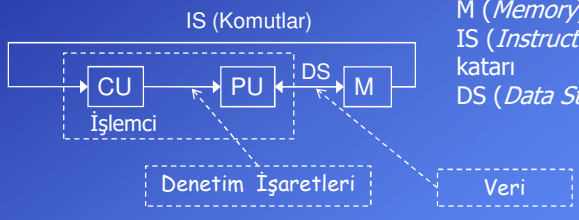
Yazılımda paralellik düzeyleri:

- Komut düzeyinde (*Instruction level*) paralellik: İş hattı (*pipelining*). Farklı komutların alt bölümleri aynı anda (paralel olarak) işlenirler.
- Paralel programlama: Bir programın veya görevin (*task*) bölümleri farklı işlemcilerde paralel olarak çalışırlar.
- İş veya süreç düzeyinde (*Job-level or process-level*) paralellik: Bağımsız uygulamalar (programlar) farklı işlemcilerde paralel olarak çalışırlar.
- Veri düzeyinde (*Data-level*) paralellik: Veri iş hattı veya çok sayıda işlem birimi (*Arithmetic logic unit -ALU*) kullanılır. Örneğin bir dizinin farklı elemanları aynı anda işlenebilir.

9.1 Flynn Sınıflandırması (*Flynn's Taxonomy*) Michael J. Flynn (1934-)

Farklı türlerde paralel yapılar bulunmaktadır.
İlk olarak Flynn tarafından ortaya konan sınıflandırma yöntemi paralel işlem yapma yeteneğine sahip yapıları kategorize etmek için kullanılan en yaygın yöntemdir.

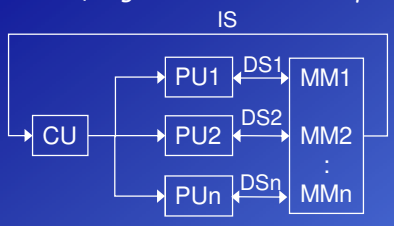
a) SISD (*Single Instruction Single Data*):
Tek komut, tek veri



CU (*Control Unit*): Denetim Birimi
PU (*Processing Unit*): İşlem Birimi
M (*Memory*): Bellek
IS (*Instruction Stream*): Komut katarı
DS (*Data Stream*): Veri katarı

Tek işlemcili sistemdir (*Uniprocessor*).
Tek bir işlemci, belli bir anda tek bir komutu yürütür ve tek bir veri üzerinde işlem yapar.

b) SIMD (*Single Instruction Multiple Data*): Tek komut, çoklu veri



CU (*Control Unit*): Denetim Birimi
PU (*Processing Unit*): İşlem Birimi
MM (*Memory Module*): Bellek Modülü

Her PU kendi bellek modülüne sahiptir.

Tek bir makine dili komut tüm PU'ları yönetir.

Her işlem biriminin kendisine ait ayrı bir veri belleği olduğundan o anda yürütülen komut farklı veriler üzerinde paralel olarak işlem yapar.

Örnek: Vektör ve dizi işlemcileri.

c) MISD (*Multiple Instruction Single Data*): Çok komut, tek veri

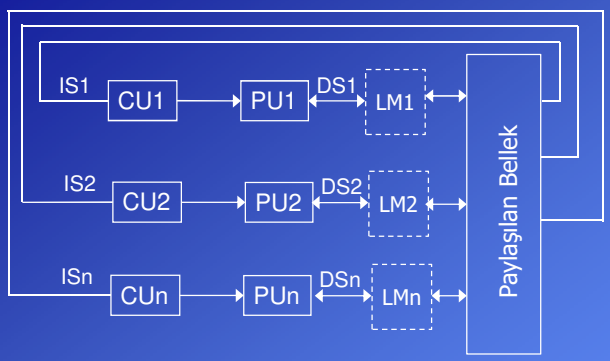
Ticari olarak bu tür sistemler üretilmemektedir.

Farklı komutlar aynı veri üzerinde işlem yaparlar.

Yüksek güvenirlilik gerektiren sistemlerde sınama ve yedekleme amaçlı olarak farklı gruplar tarafından aynı amaçla yazılan programların aynı veriler üzerinde aynı sonuçları üretmesi beklenebilir.

d) MIMD (*Multiple Instruction Multiple Data*): Çok komut, çok veri

i. paylaşılan bellekli (*with shared memory*)

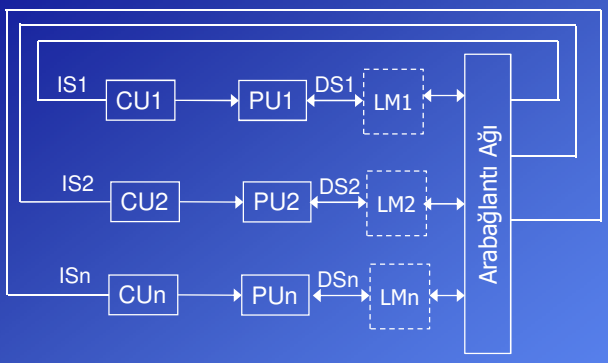


CU (*Control Unit*): Denetim Birimi
PU (*Processing Unit*): İşlem Birimi
LM: (Local Memory): Yerel Bellek
IS (*Instruction Stream*): Komut katarı
DS (*Data Stream*): Veri katarı

Çok sayıda işlemci aynı anda, farklı veriler üzerinde farklı komutları yürütürler.

- **Paylaşılan bellekli** (sıkı bağlı "*tightly coupled*") sistemler: İşlemciler ortak bir belleği paylaşırlar ve bu bellek üzerinden iletişim kurarlar (veri paylaşırlar).
  İşlemcilerin ayrıca kendi yerel bellekleri de (cep bellek) bulunabilir.

d) MIMD (*Multiple Instruction Multiple Data*): Çok komut, çok veri (devamı)
ii. dağıtılmış bellekli (*with distributed memory*)

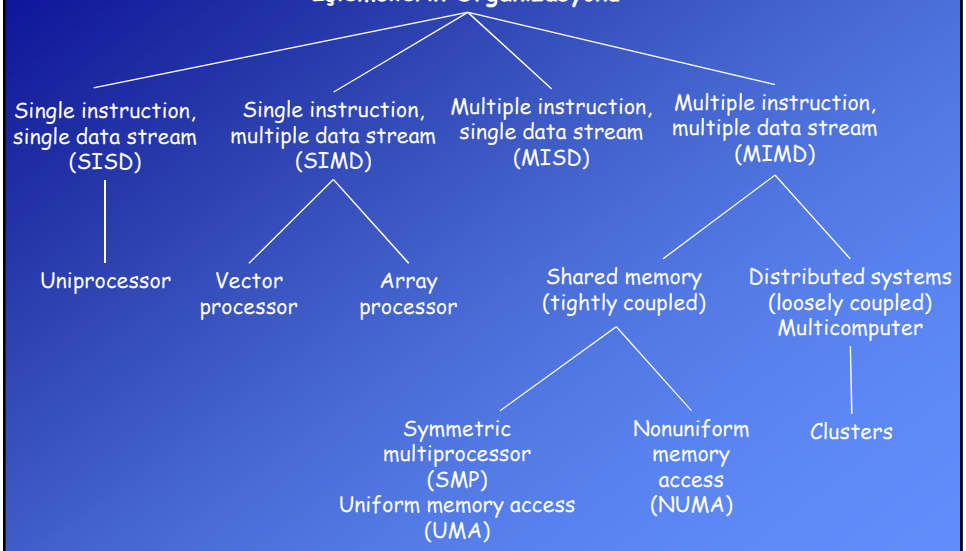


CU (*Control Unit*): Denetim Birimi
PU (*Processing Unit*): İşlem Birimi
LM: (Local Memory): Yerel Bellek
IS (*Instruction Stream*): Komut katarı
DS (*Data Stream*): Veri katarı

- **Dağıtılmış bellekli** (gevşek bağlı "*loosely coupled*") sistemler: Bağımsız işlemciler (bilgisayarlar) bir arabağlantı ağı ile bağlanarak bir küme (*cluster*) oluştururlar.

Bilgisayarlar arasında iletişim ya sabit bağlantılar ya da bir bilgisayar ağı üzerinden sağlanır.

İşlemcilerin Organizasyonu



9.2 Ortak Bellekli (Sıkı Bağlı) Sistemler (*Shared Memory /Tightly Coupled*)

- Tek bir fiziksel adres uzayı vardır (aynı bellek paylaşılır).
- İşlemciler ortak bellekteki paylaşılan değişkenler üzerinden iletişim kurarlar.
- Tüm işlemciler ortak belleğin tüm alanlarına komutlar ile erişebilirler.
- Sistem, tümleşik ortak bir işletim sistemi tarafından kontrol edilir. İşletim sistemi işlemciler arasındaki etkileşimi, süreç (*process*), görev (*task*), dosya (*file*) işlemlerinin yapılmasını sağlar.
- Paylaşılan değişkenler nedeniyle işletim sistemi işlemciler arasında senkronizasyonu da desteklemelidir.
- İki tür paylaşılan bellekli sistem vardır:

a) Simetrik çok işlemcili (*Symmetric multiprocessor* -***SMP***) veya Bir örnek bellek erişimli (*Uniform memory access* - ***UMA***) sistemler:
Bellek erişim süresi, hangi adrese erişilirse erişilsin tüm işlemciler için (hemen hemen) aynıdır.

b) Farklı bellek erişimli (*Nonuniform memory access* - ***NUMA***) sistemler:
İşlemciler mantıksal olarak aynı (tek bir) adres uzayını paylaşırlar ancak bellekler fiziksel olarak sistemin içinde dağıtılmışlardır..
Bir işlemci kendisine fiziksel olarak yakın olan bellek modüllerine uzaktakilere göre daha hızlı erişir.

9.2.1 Simetrik çok işlemcili (**SMP**) veya Bir örnek bellek erişimli (**UMA**) sistemler

Özellikleri:

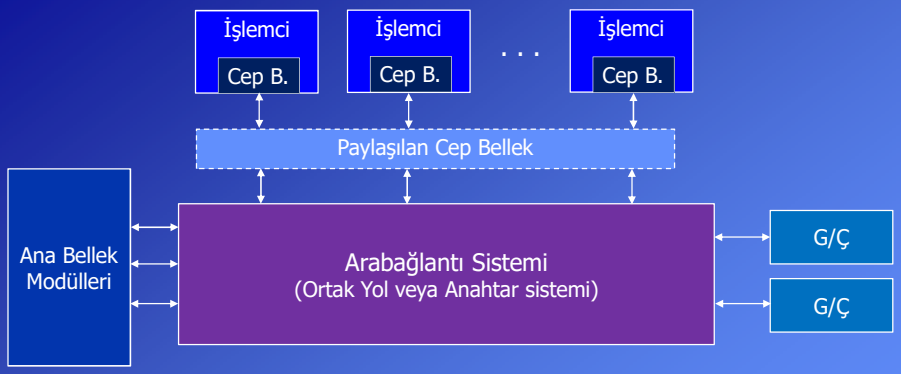
- Tek bir ortak adres uzayı, tek bir işletim sistemi
- Aynı özelliklere sahip iki veya daha fazla sayıda işlemci içerirler.
- Tüm işlemciler aynı işlevleri yerine getirebilirler (simetrik).
- İşlemciler aynı belleği ve G/Ç birimlerini paylaşırlar.
- Sistemdeki elemanlar bir ortak yol veya bir ızgara anahtarlama (*crossbar switch*) yapısı ile birbirlerine bağlanırlar.
- Bellek erişim süresi tüm işlemciler için yaklaşık olarak aynıdır (simetrik) (UMA).

Olası yararları:

- Tek işlemcili sistemlere göre performansları daha yüksektir.
- Erişilebilirlik (*Availability*): Tüm işlemciler aynı işlevleri gerçekleştirebilecek yetenekte olduklarından bir işlemcinin devre dışı kalması sistemin durmasına neden olmaz.
- Büyüme olanağı: Sisteme yeni bir işlemci ekleyerek performansını arttırmak mümkündür.

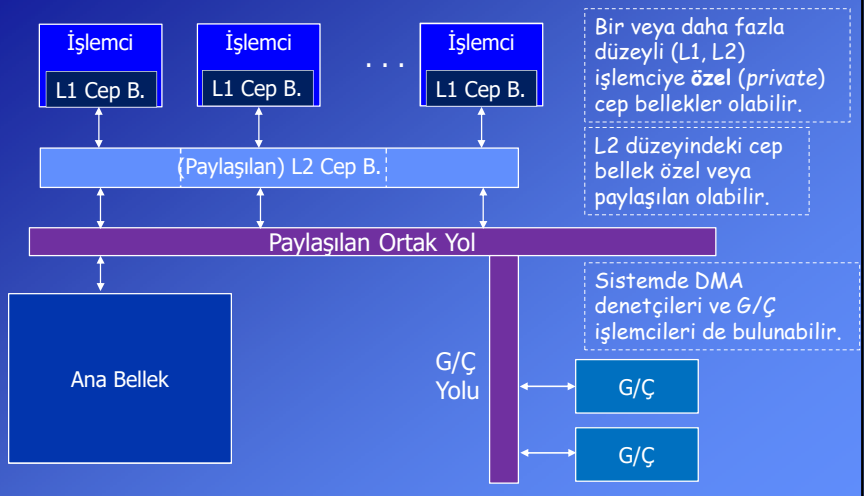
Ancak! Ancak işlemci sayısı arttıkça ortak yolu kullanımdaki çatışmalardan dolayı performans düşmeye başlar (en fazla 64 işlemci).

İçyapı (SMP, UMA):



- Her işlemci kendine ait denetim birimi, ALU, saklayıcılar ve bir veya daha fazla düzeyli cep bellek içerir.
- Ana bellek geçişli (*interleaved*) veya çok kapılı (*multiport*) olabilir; böylece farklı bloklara aynı anda erişmek mümkün olur.
- Arabağlantı sistemi farklı şekillerde tasarlanabilir. Örneğin ortak yol (*shared bus*) veya ızgara anahtar (*crossbar switch*) sistemi olabilir.

Paylaşılan Ortak Yollu Simetrik Çok İşlemcili Sistem İçyapısı:



Zaman paylaşımı (*Time-sharing*): Bir birim ortak yolu kullanırken diğer birimler yola erişemezler ve gerekirse işlemlerini bekletirler.
Bu sistemlerde **yol hakemliği** (*bus arbitration*) gereklidir.

Avantajlar:

- **Basitlik:** İşlemcilerin fiziksel arayüzleri, belleği adreslemeleri, yol hakemliği, zaman paylaşımı yöntemleri tek işlemcili sistemlerdeki gibidir.
- **Esneklik:** Yola yeni işlemciler ekleyerek sistemi genişletmek kolaydır (ancak bunun sınırları vardır).
- **Güvenirlilik:** Ortak yol pasif bir eleman olduğundan yola bağlı herhangi bir elemanın arızalanması tüm sistemin devre dışı kalmasına neden olmaz.

Dezavantajlar:

- **Performans:** Tüm bellek erişimleri ortak yol üzerinden olur.

  Yol çevriminin süresi sistemin hızını sınırlar.

  Ortak yol zaman paylaşımlı olarak kullanıldığından bir işlemci veya DMAC yolu kullanırken diğer elemanlar ana belleğe erişemez.

  Sistemde bulunabilecek MİB'lerin sayısı sınırlıdır (16 - 64).

Çözüm:

- Sık kullanılan veriler işlemcilerin **yerel cep belleklerinde** tutulur.

  Böylece ana belleğe erişim gereksinimi azaltılmış olur.
- **Cep bellek tutarlılığı** sorunu: Eğer yerel bellekteki bir sözcük değiştirilirse bu sözcüğün diğer belleklerdeki kopyaları geçersiz olur. Diğer işlemcilerin uyarılması gerekir (Cep bellek tutarlılığı 9.4 bölümünde açıklanmıştır).

9.2.2 Farklı bellek erişimli (*Nonuniform memory access*- NUMA) çok işlemciler

SMP sistemlerinde, ortak veri yolu, performans açısından bir darboğazıdır.

İşlemci sayısı sınırlıdır.

Gevşek bağlı sistemler (kümeler) bir çözüm olabilir; ancak, bu sistemlerde, uygulamalar, global (paylaşılan) bir belleği göremez.

NUMA sistemleri, paylaşılan belleğin avantajlarını koruyarak, büyük ölçekli çoklu işlemeyi başarmak için tasarlanmıştır.

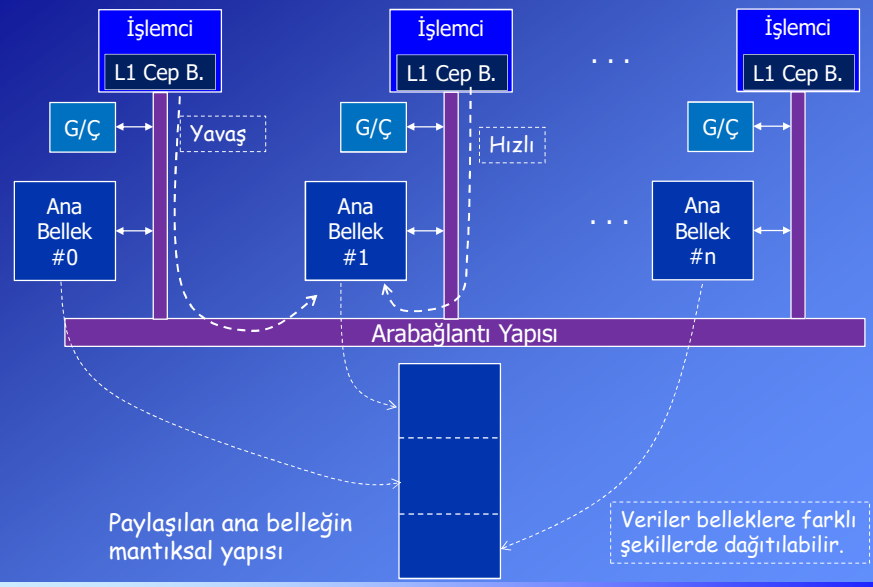
Özellikleri:

- Tek Adres Alanı (paylaşımlı bellek), tek işletim sistemi.
- Paylaşılan bellek, fiziksel olarak, MİB'lere dağıtılır. Bu sistemlere, **dağıtılmış paylaşımlı bellek sistemleri** de denir.
- Bir MİB, kendi bellek modülüne, diğer modüllerden daha hızlı erişebilir.

Performans:

- MİBlerin çoğunlukla kendi ana bellek modüllerine (veya yerel cep belleklerine) ve nadiren uzak bellek modüllerine erişeceği şekilde, işlemler ve veriler sisteme dağıtılabilirse, sistemin performansı artar.
- Programların ve verilerin uzayda ve zamanda yöreselliği, yine önemli bir rol oynamaktadır.

Farklı Bellek Erişimli (NUMA) Çok İşlemcili Sistem Yapısı:



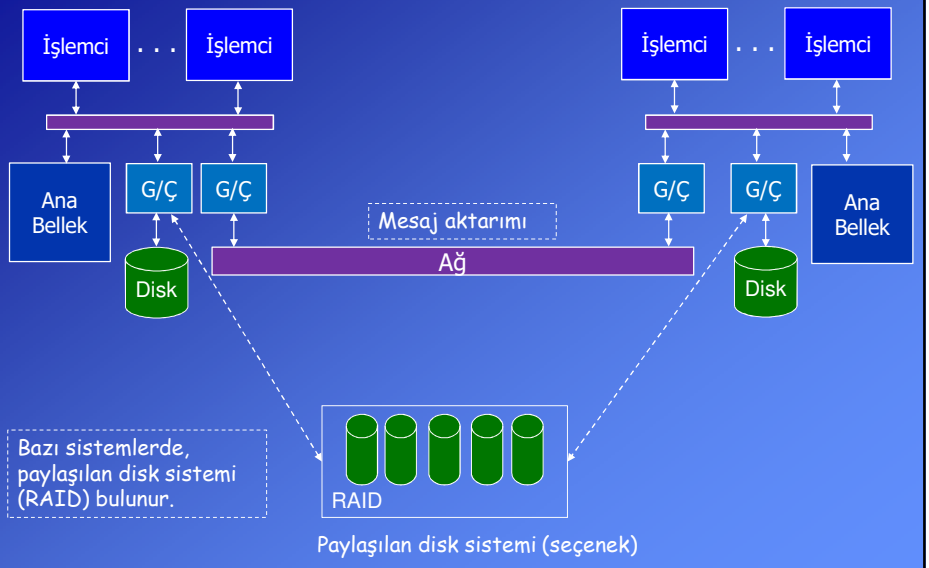
9.3 Dağıtılmış (*distributed*) (gevşek bağlı "*loosely coupled*") sistemler, Çoklu bilgisayarlar (*Multicomputer systems*)

- Her işlemci, kendi fiziksel adres alanına sahiptir.
- Bu işlemciler, mesaj aktarımı (*message passing*) yoluyla haberleşir.
- Mesaj aktarma sisteminin en yaygın örneği, **kümelerdir (*clusters*)**.
- Kümeler, standart ağ donanımları üzerinden birbirlerine bağlı bilgisayar gruplarıdır.
- Bu kümelerin boyutu, on binlerce sunucuya ve daha fazlasına ulaştığında, depo ölçekli (*warehouse-scale*) bilgisayarlar olarak adlandırılırlar ve bulut bilişimde (*cloud computing*) kullanılırlar.

Yararları:

- **Ölçeklenebilirlik (*Scalability*):**
  - Bir küme, her biri çok işlemcili olan onlarca, yüzlerce, hatta binlerce makine içerebilir.
  - Kümeye, küçük artışlarla, yeni sistemler eklemek mümkündür.
- **Yüksek erişilebilirlik (*Availability*):**
  - Bir kümedeki her düğüm, bağımsız bir bilgisayardır. Bu nedenle, bir düğümdeki arıza, servis kaybı anlamına gelmez.
- **Üstün fiyat/performans:**
  - Ucuz, kolayca bulunabilen yapı taşları kullanarak, büyük hesaplama gücü olan bir küme oluşturmak mümkündür.

Gevşek Bağlı (dağıtılmış) Sistem Yapısı:

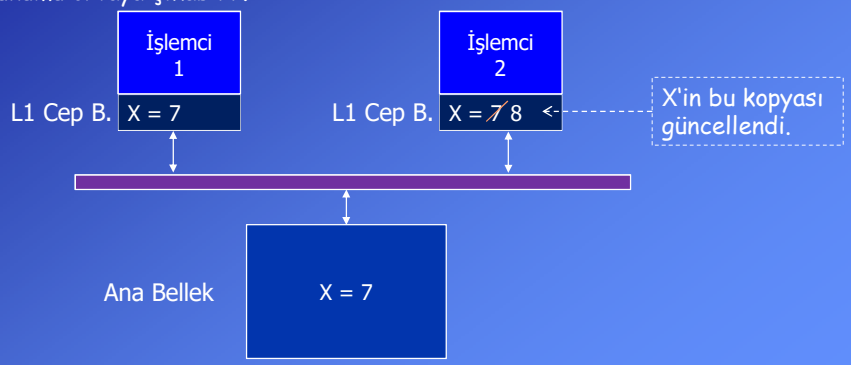


9.4 Cep (ön) bellek tutarlılığı

Ortalama erişim süresini ve gereken (ana) bellek bant genişliğini azaltmak için, cep bellekler kullanılır.

Paylaşılan verileri cep belleğe alma, **cep bellek tutarlılığı sorununu** (**cache coherency problem**) ortaya çıkarır.

Aynı verinin birden fazla kopyası, aynı anda, farklı cep belleklerde bulunabilir ve işlemcilere, kendi kopyalarını serbestçe güncelleme izni verilirse, tutarsız bir bellek görünümü ortaya çıkabilir.



9.4.1 Yazılım çözümleri:

- Ek donanım devresine gerek yoktur.
- Derleyici ve işletim sistemi, derleme anında (*compile time*) sorunu çözerler.
- Ancak, ihtiyatlı (sakıngan) kararlar verirler; bu da cep belleğin verimsiz kullanımına neden olur.
- Derleyici tabanlı mekanizmalar, hangi veri öğelerinin cep belleğe alınmasının tutarsızlığa neden olabileceğini belirlemek için kodu inceler ve bu öğeleri işaretlerler.

  Daha sonra, işletim sistemi veya donanım, bu öğelerin cep belleğe alınmasını önler.

  En basit yaklaşım, paylaşılan veri değişkenlerinin, cep belleğe alınmasını önlemektir (aşırı ihtiyatlı ve verimsizdir).
- Daha etkili bir yaklaşım, paylaşılan değişkenlerin güvenli ve kritik dönemlerini belirlemek için kodu analiz etmek ve cep bellek tutarlılığını sağlamak için, kod içine komutlar eklemektir.

9.4.2 Donanım çözümleri:

a) Dizin protokolleri (*Directory protocols*):

Ana bellekte yer alan bir dizini (*directory*) kullanan bir merkezi denetçi bulunur.

Dizin, hangi işlemcinin kendi özel cep belleğinde, hangi verilerin (çerçevelerin) kopyalarının bulunduğuyla ilgili bilgiler içerir.

- Bir işlemci, bir satırın yerel bir kopyasına yazmak istediğinde, denetçiden, satıra özel erişim (*exclusive access*) talep etmelidir.

  Denetçi, tüm işlemcilere bir mesaj göndererek kendi belleklerindeki veriyi geçersiz kılmaya zorlar ve onay bekler.

  Denetçi, bütün bu işlemcilerden gelen onayları (*acknowledgments*) aldıktan sonra, istekte bulunan işlemciye özel erişim (*exclusive access*) sağlar.
- Bir işlemci, başka bir işlemciye yazma izni verilen (*exclusively granted*) bir satırı okumaya çalıştığında ıska olur (veri artık geçersizdir).

  Doğrudan yazma (*write-through*) yöntemi kullanılıyorsa güncel veri ana bellekte vardır.

  Sonradan yazma (*write-back*) mekanizması kullanılırsa, denetçi, güncel veriyi tutan işlemciye komut göndererek verinin ana belleğe yazılmasını sağlar.

  Veri, artık orijinal işlemci ve istekte bulunan işlemci tarafından okunacak şekilde paylaşılabilir.

a) Dizin protokolleri (Directory protocols) (devamı):

Sakıncaları:

- Merkezi denetçi, bir darboğazdır. Tüm istekler, aynı denetçiye gelir.
- Yerel cep bellek denetçileri ve merkezi denetçi arasındaki iletişim, ek yük getirir (*overhead*).

Avantaj:

- Birden fazla yol veya başka bir karmaşık ara bağlantı düzeni (*interconnection scheme*) içeren, büyük ölçekli sistemlerde etkilidir.

b) Gözetleme protokolleri (*Snoopy protocols*):

- Cep bellek tutarlılığını sürdürme sorumluluğu, çok işlemcili sistemdeki tüm cep bellek denetçileri arasında dağıtılır.
- Paylaşılan bir cep bellek çerçevesi (satır) güncellendiğinde, yerel denetçi bu işlemi tüm diğer cep bellek denetçilerine, bir yayın (*broadcast*) mekanizması aracılığıyla duyurur.
- Her bir cep bellek denetçisi, bu yayınlanmış bildirimleri gözlemlemek için, ağ üzerinde "gözetleme" (*snooping*) yapar ve buna göre tepki verir (örneğin, kendi kopyasını geçersiz kılar).
- Gözetleme protokolleri, ortak yollu çoklu işlemciler için uygundur; çünkü, paylaşılan ortak veri yolu, yayın ve gözetleme için basit bir mekanizma sağlar.
- Hatırlatma: Paylaşılan veri yolundaki trafiği azaltmak için, yerel cep bellekler kullanılır.
- Bu nedenle, paylaşılan veri yolu üzerindeki trafiği, yayın ve gözetleme mesajlarıyla arttırmamaya özen gösterilmelidir.

b) Gözetleme protokolleri (*Snoopy protocols*) (devamı):

İki tür gözetleme protokolü vardır: Yazma geçersiz kılma ve yazma güncelleme

Yazma geçersiz kılma (*write-invalidate*) protokolü:

- İşlemcilerden biri, özel cep belleğindeki çerçeveye yazmak istediğinde "geçersiz kıl" (*invalidate*) mesajı gönderir.
- Tüm cep bellek denetçileri, ilgili cep bellek çerçevesinin kendilerindeki kopyalarını geçersiz kılar.
- Satır özel (paylaşılmamış) olduğunda (*exclusive (not shared)*), işlemci kendi kopyasına yazabilir.
- Doğrudan yazma (*write-through*) yöntemi kullanılırsa, veriler, ana belleğe de yazılır.
- Başka bir MİB, bu verileri okumaya çalışırsa, bir ıska olur ve veri, ana bellekten alınır.

Yazma güncelleme (*write-update*) protokolü:

- İşlemcilerden biri, paylaşılan bir veriyi güncellemek istediğinde, yeni veriyi tüm diğer işlemcilere yayımlar ve böylece onlar da, özel cep belleklerini güncellerler.
- Aynı zamanda, MİB, cep bellekteki kendi kopyasını günceller.

Deneyimler, **geçersiz kılma (*invalidate*)** protokollerinin, **önemli derecede daha az bant genişliği kullandığını** göstermiştir.

MESI (*Modified Exclusive Shared Invalid*) Protokolü

- Bir gözetleme (*snoopy*), yazma geçersiz kılma (*write-invalidate*) cep bellek tutarlılık protokolüdür.
- Sonradan yazma (*write-back*) yönteminin kullanılmasına izin verir. Çerçeveyi değiştirmek gerekinceye kadar, ana bellek güncellenmez.
- Her yerel cep bellekteki, her çerçeve (satır), dört durumdan birinde olabilir:

M (Modified/Değiştirilmiş): Bu cep bellekteki çerçeve değiştirilmiştir. Ana bellektekinden farklıdır.

Bu çerçeve, yalnızca bu cep bellekte geçerlidir.

E (Exclusive/Özel): Cep bellekteki çerçeve, ana bellekteki ile aynıdır ve başka cep belleklerde bulunmamaktadır.

S (Shared/Paylaşılan): Cep bellekteki çerçeve, ana bellekteki ile aynıdır ve başka bir cep bellekte de bulunabilir.

I (Invalid/Geçersiz): Cep bellekteki satır, geçerli veri içermemektedir.

	Modified	Exclusive	Shared	Invalid
Cep bellek çerçevesi geçerli mi?	Evet	Evet	Evet	Hayır
Ana bellekteki veri geçerli mi?	Hayır	Evet	Evet	-
Başka cep belleklerde kopyalar var mı?	Hayır	Hayır	Belki	Belki

Çerçevelerin durum geçişleri:

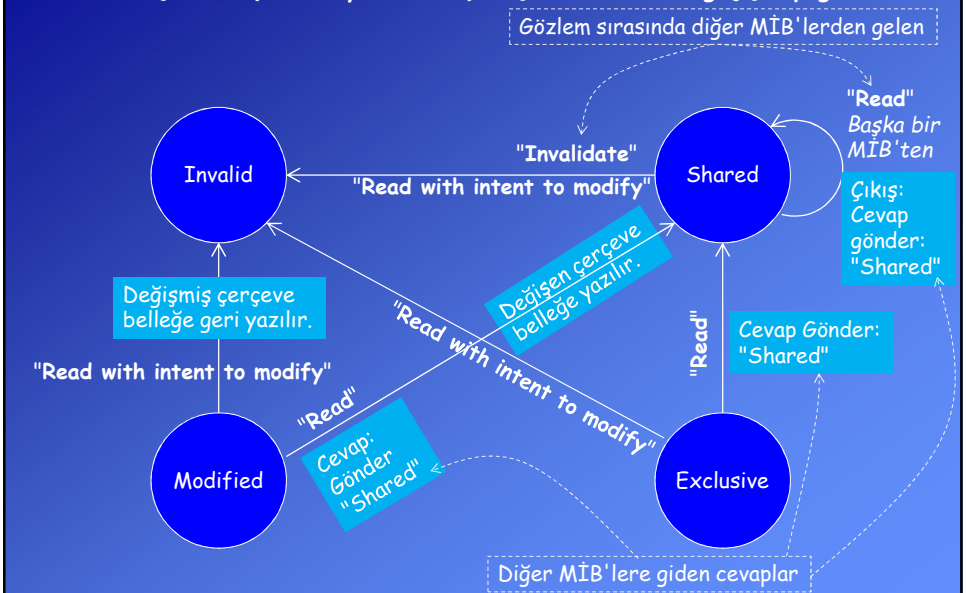
Durum değerini tutmak için, takı belleğindeki her cep bellek çerçevesine, iki bit atanır.

Cep bellek çerçevesinin durumu, iki nedenden dolayı değişebilir:

- Yerel işlemci etkinliği nedeniyle (MİB'in kendi cep belleğine erişimi)
- Gözetleme sonucunda, yol etkinliği nedeniyle (başka bir işlemcinin neden olduğu etkinlik)

Yerel olarak başlatılan (*locally initiated*) erişimlerde durum geçiş diyagramı:

Uzaktan başlatılan (*remotely initiated*) erişimlerde durum geçiş diyagramı:



MESI Protokolü (devamı)

Protokolün çalışması:

Okuma Iska (Geçersiz durumdayken) (*Read Miss (in Invalid state)*):

- İşlemci, ana bellekten çerçeveyi almaya başlar.
- MİB, diğer cep bellek denetçilerine, işlemi gözetleme işareti yollar.
- Dört tane olası karşılık vardır:
  - A. Başka bir cep bellek, değiştirilmemiş (temiz) bir özel kopyaya sahipse (E), bu verileri paylaştığını belirtir.

    Yanıt veren cep bellek çerçevesi, özel (E) durumdan paylaşılan (S) duruma geçer.

    Başlatan MİB, çerçeveyi bellekten okur ve cep bellek çerçevesi, geçersiz durumdan (I), paylaşılan duruma (S) geçer.
  - B. Diğer cep bellekler, değiştirilmemiş (temiz) paylaşılan kopyalara sahiplerse (S), bu verileri paylaştıklarını belirtirler.

    Yanıt veren cep bellek çerçeveleri, paylaşılan durumda kalırlar (S).

    Başlatan MİB, çerçeveyi okur ve çerçeve, geçersiz durumdan (I) paylaşılan duruma (S) geçer.

Okuma Iska (Geçersiz durumda) (*Read Miss (in Invalid state)*) (devamı)

- Dört tane olası karşılık vardır (devamı):
  - C. Başka bir cep bellek, değiştirilmiş bir kopyaya sahipse (M), bellek okuma işlemini engeller ve istenen çerçeveyi sağlar.

    Bu veri, aynı zamanda, ana belleğe yazılır.

    Farklı gerçeklemeler vardır. İstekte bulunan MİB, verileri, yanıt veren MİB'den veya bellek güncellendikten sonra, ana bellekten okuyabilir.

    Yanıt veren MİB'in cep bellek çerçevesi, değiştirilmiş durumdan (M), paylaşılan (S) duruma geçer.

    Başlatan MİB'in cep bellek çerçevesi, geçersiz durumdan (I) paylaşılan duruma (S) geçer.
  - B. İstenen çerçevenin başka hiçbir cep bellekte kopyası yoksa, hiçbir sinyal döndürülmez.

    Başlatan MİB, çerçeveyi, bellekten okur ve cep bellek çerçevesi, geçersiz durumdan (I) özel duruma (E) geçer.

Okuma Vuru (İsabet) (*Read Hit*):

MİB, sadece gerekli veriyi cep bellekten okur.
Cep bellek çerçevesi, aynı (şu anki) durumda kalır: değiştirilmiş, paylaşılmış veya özel.

Yazma Iska (Geçersiz durumdayken) (*Write Miss (in Invalid state)*):

- İşlemci, ana bellekten çerçeveyi almaya başlar.
- MİB, veri yolu üzerinden, *değişiklik yapma niyetiyle okuma* (*read-with-intent-to-modify*) sinyali gönderir.
- İki olası durum vardır:

A. Başka bir cep bellek, çerçevenin değiştirilmiş bir kopyasına sahipse (M), istekte bulunan MİB'e bildirir (bu çerçevedeki bazı sözcükler değiştirildi).

İstekte bulunan MİB, yol işlemini durdurur ve bekler.

Diğer MİB, değiştirilmiş cep bellek çerçevesini, ana belleğe geri yazar ve cep belleğin durumunu, değiştirilmişten (M) geçersize (I) geçirir.

Başlatan MİB, tekrar yol üzerinden *değişiklik yapma niyetiyle okuma* (*read-with-intent-to-modify*) sinyalini gönderir ve çerçeveyi ana bellekten okur.

MİB, çerçevedeki sözcüğü değiştirir ve çerçevenin durumunu, "değiştirilmiş" (M) yapar.

B. İstenen çerçevenin, başka bir cep bellekte değiştirilmiş (M) bir kopyası yoksa, hiçbir sinyal döndürülmez.

Başlatan MİB, çerçeveyi ana bellekten okur ve değiştirir.

Bu çerçevenin başka cep belleklerde (S) ve/veya (E) durumunda kopyaları varsa hepsi geçersiz duruma (I) geçer.

Yazma Vuru (İsabet) (*Write Hit*):

CPU, yazmaya (bir değişkeni değiştirmeye) çalışır ve değişken (çerçeve), yerel cep bellekte bulunmaktadır.

İşlemler, değiştirilen çerçevenin durumuna bağlıdır.

Paylaşılan (*Shared*):

- MİB, paylaşılan veri yolundan, "geçersiz kıl" ("*invalidate*") sinyali gönderir.
- Çerçevenin paylaşılan bir kopyasını, kendi cep belleklerinde bulunduran diğer MİB'ler, bu çerçevenin durumunu, "paylaşılan" dan (S) "geçersiz"e (I) geçirirler.
- Başlatan MİB, değişkeni günceller ve çerçevenin kendi kopyasını, "paylaşılan"dan "değiştirilmiş"e (M) geçirir.

***Özel* (*Exclusive*):**

- MİB, zaten verinin tek (özel) kopyasına sahiptir.
- MİB, değişkeni günceller ve çerçevenin kendi kopyasını, "özel"den (E) "değiştirilmiş"e (M) geçirir.

Değiştirilmiş (*Modified*):

- MİB, zaten verinin tek değiştirilmiş kopyasına sahiptir.
- MİB, değişkeni günceller. Durum, "değiştirilmiş" (M) olarak kalır.

Örnek:

Paylaşılan veri yoluna sahip, bir simetrik çok işlemcili (SMP) sistemde, yerel cep belleğe sahip, iki MİB (MİB1 ve MİB2) vardır.

Sistem, paylaşılan L2 cep bellek içermez.

Cep bellek yönetim birimleri, her kümede iki çerçeve olan (*2-way set associative*) kümeli ve çağrışımlı dönüşüm (*set associative mapping*) yöntemini kullanmaktadır.

Yazma işlemlerinde, Cebe Yükleyerek Yazma (*Write Allocate WA*) ile Bayraklı Sonradan Yazma (*Flagged Write Back FWB*) yöntemleri kullanılmaktadır.

Sistemde, paylaşılan bir X değişkeninin olduğunu varsayalım. Cep bellek tutarlılığını sağlamak için, gözetlemeli (*snoopy*) MESI protokolü kullanılır.

Aşağıdaki sorular, birbirlerinden bağımsız olarak cevaplanabilir.

a) Her iki MİB'in cep belleklerinin, X değişkeninin geçerli kopyalarını içerdiğini varsayalım. X'in kopyası, MİB1'in cep belleğinde küme:1, çerçeve:0'da ise, MİB2'nin cep belleğindeki yerini bilebilir miyiz? Neden?

Çözüm:

Simetrik bir çok işlemcili (SMP) sistemde, MİB'ler, aynı bellek alanını kullanırlar. Dolayısıyla, X değişkeni, MİB1'in ve MİB2'nin belleklerinde aynı adrese sahiptir.

Eğer MİB1'in cep belleğinde küme: 1, çerçeve:0 'da ise, o zaman, MİB2'nin cep belleğinde de, küme 1'de olmalıdır. Ancak, küme 1'de hangi çerçevede yer aldığını bilemeyiz.

Örnek (devamı):

b) MİB1 cep belleğindeki X değişkenini içeren çerçevenin durumu "özel" (E) ise MİB2'nin cep belleğinde, buna karşılık gelen çerçevenin durumu nedir?

Çözüm:

Bu durumda, X değişkeninin geçerli kopyaları, ana bellekte ve MİB1 cep belleğinde bulunur. Bu nedenle, MİB2'nin cep belleğinde buna karşılık gelen çerçeve, "**geçersiz**" (*invalid*) durumunda olmalıdır.

c) MİB1 cep belleğindeki X değişkenini içeren çerçevenin "değiştirilmiş" (M) durumda olduğunu ve MİB2'nin X değişkenine yazmak istediğini varsayalım. MESI protokolü tarafından gerçekleştirilen işlemleri listeleyin.

Çözüm: MİB1'de değiştirilmişse, MİB2'de bulunmamaktadır (geçersizdir) (I).

- MİB2 (Yazma ıska), *değişiklik yapma niyetiyle okuma* (*read-with-intent-to-modify*) sinyali gönderir.
- MİB1, istekte bulunan MİB2'ye "*Ana bellek geçerli değil*" sinyali gönderir.
- MİB1, değiştirilmiş cep bellek çerçevesini, ana belleğe yazar ve cep bellek durumunu, "değiştirilmiş"ten (M) "geçersiz"e (I) değiştirir.
- MİB2 tekrar "*read-with-intent-to-modify*" sinyali gönderir ve çerçeveyi ana bellekten okur.
- MİB2, çerçevedeki sözcüğü değiştirir ve çerçevenin durumunu, "değiştirilmiş" (M) yapar.

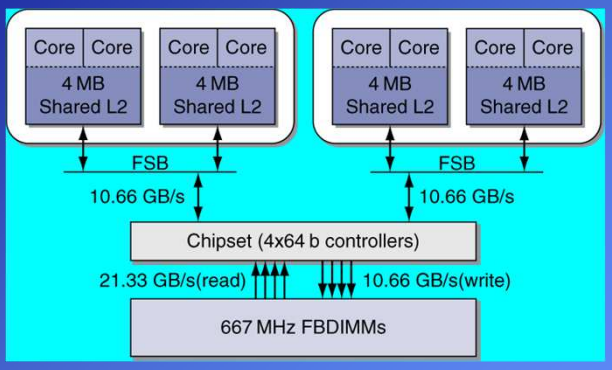
9.5 Paralel işlemenin (*parallel processing*) zorlukları

- Programlarda sınırlı paralellik. Bazı işlemciler kullanılamaz (yük dağılımı).
- İletişimin getirdiği ek yük. İşlemciler arasında iletişimin yüksek maliyeti.
- Paralel program yazmak zordur.
- Benzer yükleri olan bağımsız bölümlere ayırma: Zamanlama/planlama (*scheduling*) ve yük dengeleme (*load balancing*) problemi.
- Senkronizasyon problemleri: Bağımlılıklar, kritik bölümler (*Dependencies, critical sections*)

9.6 Örnek Çok İşlemcili Sistemler [1]

Tek bir yongaya birden fazla işlemci yerleştirildiğinde, sisteme **çok çekirdekli yonga işlemci** (multicore chip processor) denir.

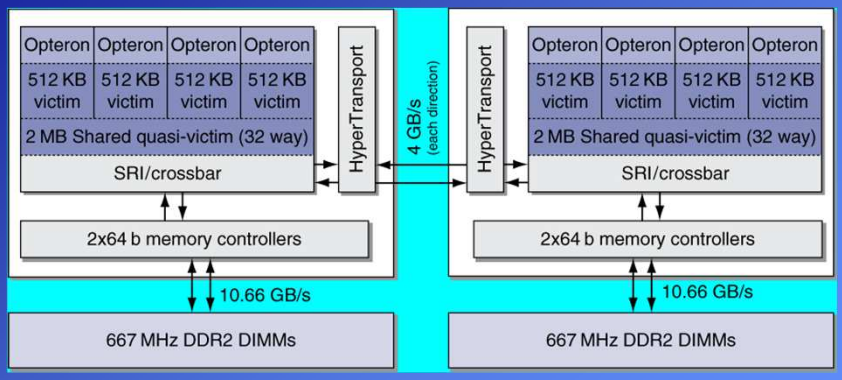
2 × dört-çekirdek Intel Xeon e5345 (Clovertown) içeren bir SMP sistemi:



L1 Cep B.:
128 KB (kod)
128 KB (veri)

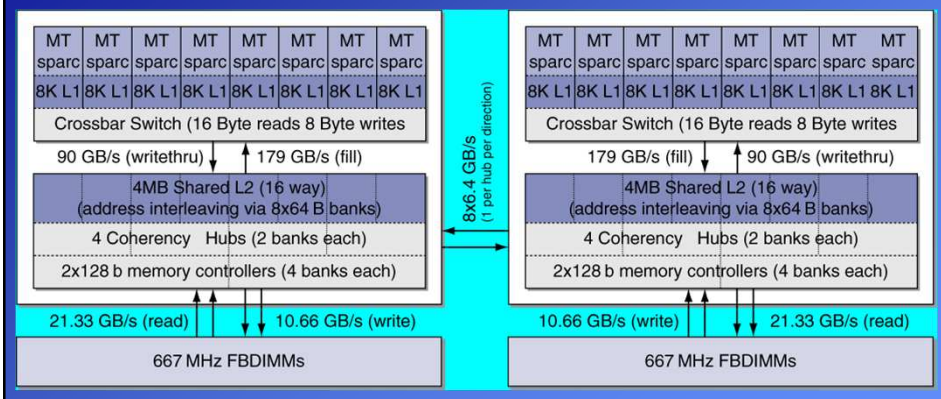
1 Kaynak: Patterson & Hennessy, *Computer Organization and Design, Morgan Kaufmann, Elsevier, 2009.*

2 × dört-çekirdek AMD Opteron X4 2356 (Barcelona)



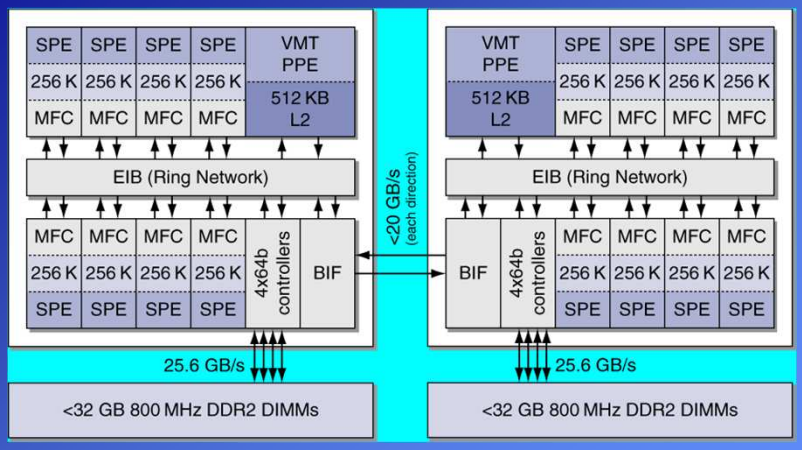
Kaynak: Patterson & Hennessy, *Computer Organization and Design, Morgan Kaufmann, Elsevier, 2009.*

2 × sekiz-çekirdek Sun UltraSPARC T2 5140 (Niagara 2)



Source: Patterson & Hennessy, *Computer Organization and Design, Morgan Kaufmann, Elsevier, 2009.*

2 × sekiz-çekirdek IBM Cell QS20:



Source: Patterson & Hennessy, *Computer Organization and Design, Morgan Kaufmann, Elsevier, 2009.*

Performans engeli (performance wall) ve yeni çözüm arayışları *

Bilgisayarların performansları, yarı iletken elamanlar (transistörler) ve bilgisayar mimarisindeki gelişmeler (cep bellek, iş hattı) sayesinde artmıştır.

Ancak, bu gelişmeler (özellikle, Moore yasası) sona ermektedir.

Tasarımcılar, bilgisayarların işlem gücünü arttırmak için, saat işaretini ve/veya tümleşik devredeki transistör sayısını arttırmaktadır. Ancak, bu ısınma/soğutma problemine neden olmaktadır (power wall).

İş hattı ve çok çekirdekli sistemler gibi mimari yapıların da, kendilerine özgü sorunları bulunmaktadır.

Bununla birlikte, büyük veri, makine öğrenmesi ve güvenlik gibi uygulama alanları için geliştirilen yeni yazılım sistemlerinin performans gereksinimleri, 1 saniyede 1 milyon trilyon kayan noktalı sayı işlemini (1 exaflop) geçmektedir.

Endüstrideki birçok uzman, 2020 yılına gelindiğinde, bilgisayarların uzun süre önce öngörülen **performans engeline** (*performance wall*) ulaşacağına inanmaktadır.

Gelecekteki bilgisayar sistemleri için öngörülen elemanlar, devreler ve bilgisayar mimarisi konularında bilgi almak için "IEEE Rebooting Computing Initiative" girişiminin web sayfasını ziyaret ediniz. http://rebootingcomputing.ieee.org/

*Kaynak: T. M. Conte, E. P. DeBenedictis, P. A. Gargini, and E. Track, "Rebooting Computing: The Road Ahead," Computer, vol. 50, no. 1, pp. 20–29, Jan. 2017.