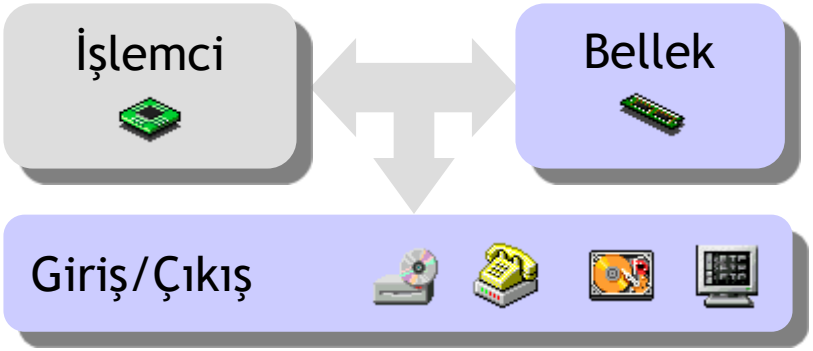
Ön bellekler (Cache'ler)

Hatırlatma: Bellek Sistemleri ve I/O

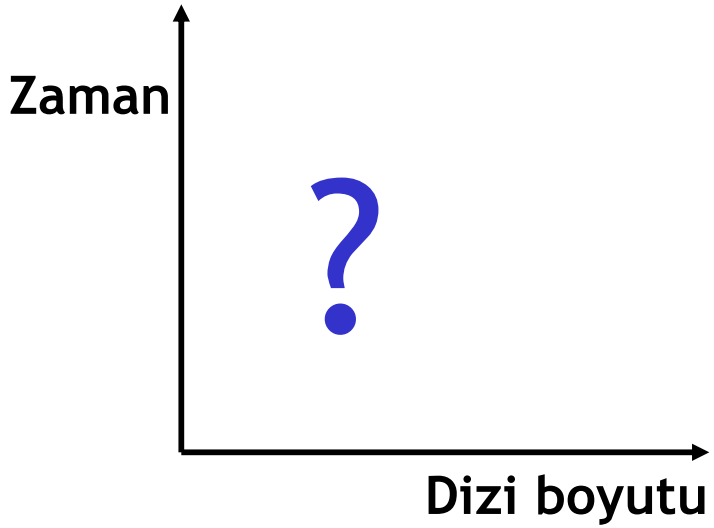
- İşlemcilerimiz zaten hızlı ama onu yeterince meşgul tutacak kadar veriyi nasıl sağlayacağız?
- Bellek ve I/O bir sistemin performansını etkileyen en önemli bileşenleri olduğunu görmüştük.
- Önümüzdeki birkaç hafta bellek sistemleri üzerine konuşacağız.
  - Bir cache belleğin genel bellek erişimi hızına olan derin etkisi.
  - Sanal bellek kullanımı ile işlerin kolaylaşması, güvenlik kaygıları.
  - İşlemci, bellek. cache ve çevre birimler birbirlerine nasıl bağlılar?



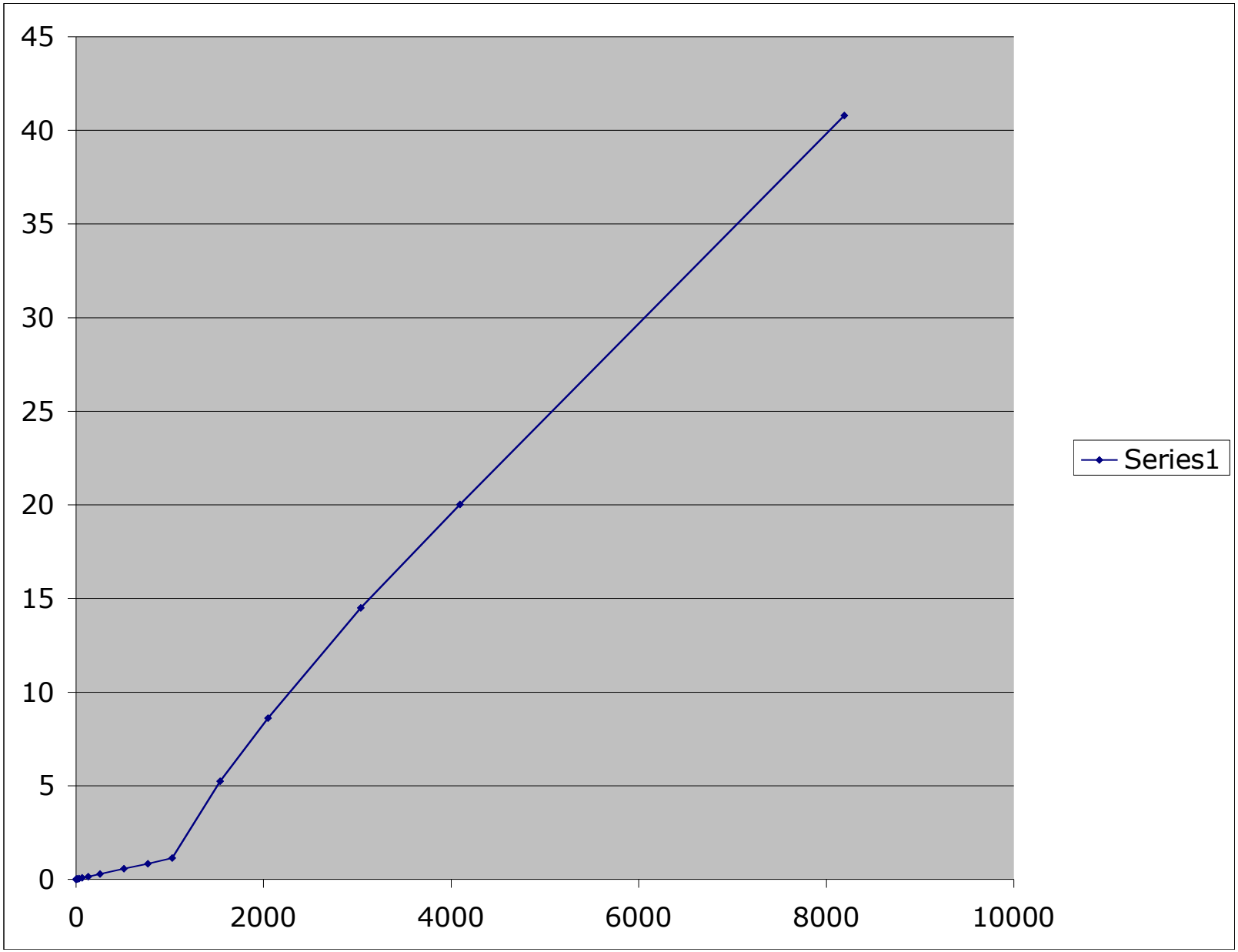
Veri Boyutu Çalışma Süresini Nasıl Etkiler?

```
int array[SIZE];
int A = 0;

for (int i = 0 ; i < 200000 ; ++ i) {
   for (int j = 0 ; j < SIZE ; ++ j) {
        A += array[j];
   }
}
```

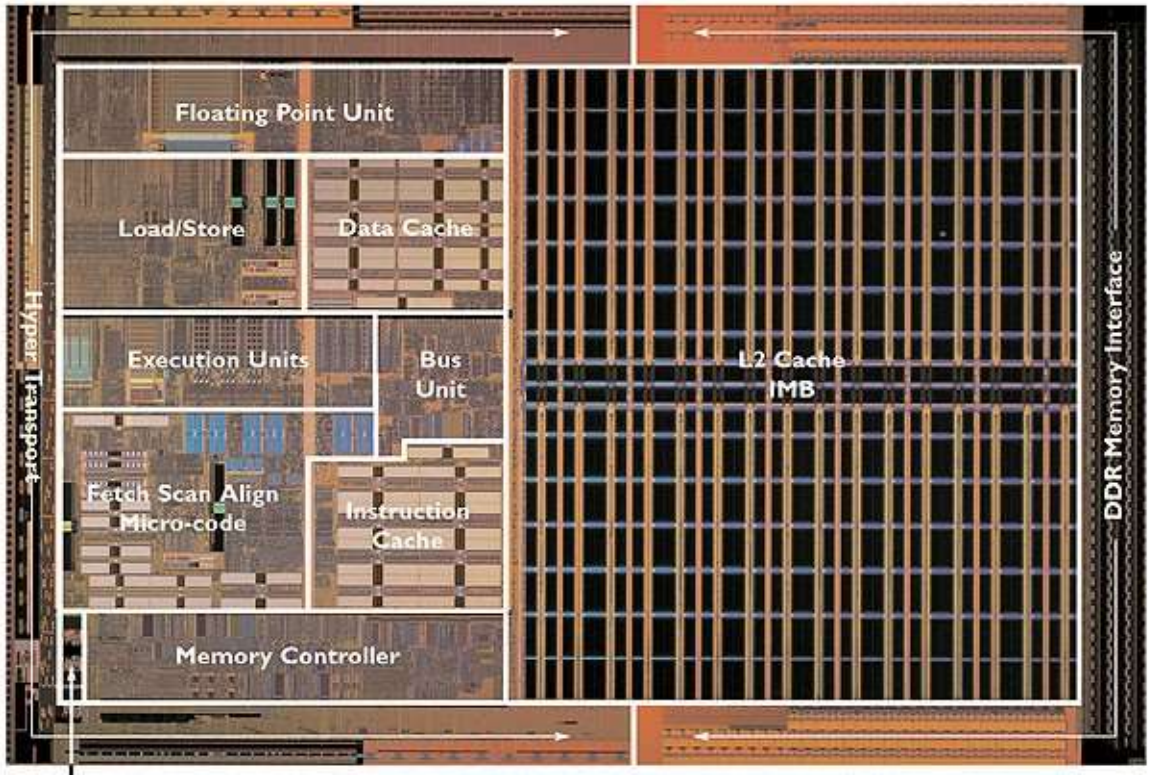


Küçük-Büyük Boyutlarda Dizi İşlemek



Cache Belleklere Giriş

- Bugün aşağıdaki sorulara cevaplar vereceğiz.
  - Neden çok büyük ve hızlı bellek sistemleri üretemiyoruz, engel nedir?
  - Ön bellek (ing:cache) nedir?
  - Cache'in etkisi nereden geliyor? (cevap: yerellik / locality)
  - Cache'ler nasıl organize edilir?
    - Bir veriyi cache içine nasıl koyar ve onu orada nasıl buluruz?



Hızlı ve Geniş

- Günümüz bilgisayar sistemlerinin en büyük ihtiyacı hızlı ve geniş depolama alanlarıdır.
  - Veri tabanı ve big data uygulamaları, büyük veri setli bilimsel hesaplamalar, video ve müzik dosyaları gibi büyük yer gerektiren uygulamalar vardır.
  - Hızla çalışan bir işlemcide bir saat sinyali gecikme bile hoş karşılanmaz, etrafındakilerin hızı onun iş hattını meşgul tutmak için çok önemlidir. Çok işlemcili sistemlerde bu konu daha da önemli.

Yavaş veya Küçük?

- Maalesef hız, maliyet ve kapasite arasında bir geçimsizlik var.

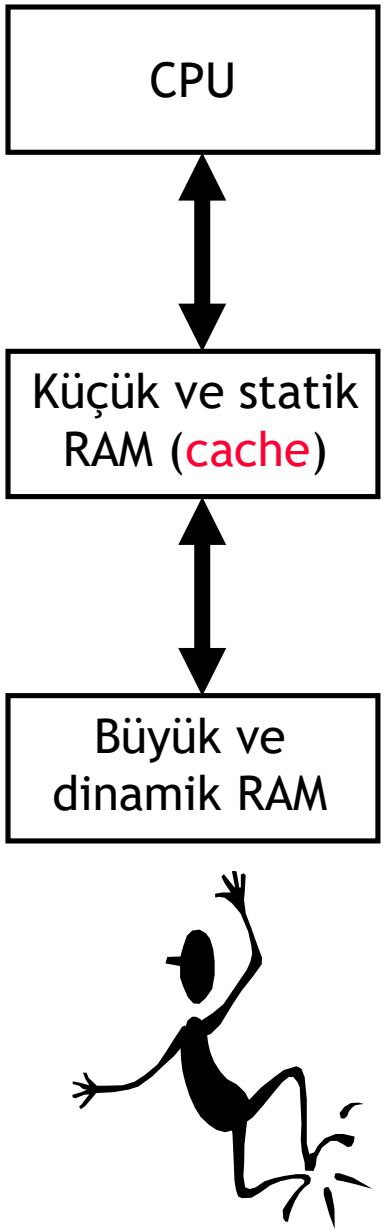
Ortam	Hız	Maliyet	Kapasite
Statik RAM	En hızlı	Pahalı	En küçük
Dinamik RAM	Yavaş	Ucuz	Büyük
Hard diskler	En yavaş	En ucuz	En büyük

- Hızlı cache bellekler işlemci fiyatını arttırır, piyasa fiyatı pahalıdır.
- Oysa dinamik bellek (RAM)'lerde, aynı hattakilere göre daha uzun beklemeler oluyor. Eğer her okuma/yazma direk bu bellekten yapılsa, o zaman CPU saat sinyalini veya boşta geçen zamanları arttırabiliriz.
- Aşağıda kaba bir karşılaştırma tablosu veriyoruz. Rakamlar hatalı olabilir.

Ortam	Gecikme	Maliyet/MB	Kapasite
Statik RAM	1-10 saat sinyali	~$5	128KB-2MB
Dinamik RAM	20-50 saat sinyali	~$0.10	1GB-64GB
Hard diskler	100,000 saat siny.	~$0.0005	500GB-4TB

Cache'lere Giriş

- Hızlı ve ucuz bellek arasında bir denge kursak hoş olmaz mıydı?
- Biz bunu küçücük, çok hızlı ama pahalı cache'leri tanıyarak yapacağız.
  - Cache işlemci ile yavaş kalan dinamik bellek arasına girer.
  - Cache'in ana amacı ana bellekte sıklıkla kullanılan verileri üzerinde tutarak tüm operasyona hız katmaktır.
- Cache ile bellek erişimi bir anda hızlanır, çünkü en çok kullanılan alanları hızlandırdık.
  - Bu noktadan sonra sıklıkla kullanılan bellek adreslerine okuma ve yazma isteği geldiğinde cache cevap verecektir.
  - Ve yine bu noktadan sonra cache'de bulunmayan adreslere ulaşmak gerektiğinde ana bellek cevap verecektir.



Yerellik (Locality) Prensibi

- Bir program daha çalışmadan hangi veriler sıklıkla kullanılacak bilmek zor, hatta imkansızdır. Dolayısıyla küçücük bir cache içine neyi yazarız bilmek zor.
- Fakat pratikte, cache'in kendi avantajına kullanacağı, çoğu programın yarattığı bir veri yerelliği/mekanı (ing:locality) vardır.
  - **Zamanda yerellik (ing:temporal locality)** der ki programınız bellekte bir adrese bir kez ulaşırsa, o adrese tekrar ulaşma ihtimali çok yüksektir.
  - **Alanda yerellik (ing:spatial locality)** der ki programınız bellekte bir adrese bir kez ulaşırsa, o adresin devamındaki adreslere de ulaşma ihtimali çok yüksektir.

Programlardaki Zamansal (Geçici) Yerellik

- **Zamansal yerellik (ing:temporal locality)** der ki programınız bellekte bir adrese bir kez ulaşırsa, o adrese tekrar ulaşma ihtimali çok yüksektir.
- Programlarımızda kullanılan döngüler bu yerelliğe en iyi örneklerdir.
  - Döngü gövdesi defalarca işletilecektir.
  - Bu esnada bilgisayar bazı bellek pozisyonlarına (kod veya veri) defalarca erişecektir.
- RISC komutlarından oluşan bir örnek verirsek:

```
Loop:  lw    t0, 0(a0)
       add   t0, t0, t1
       sw    t0, 0(a0)
       addi  a0, a0, 4
       blt   a0, a1, Loop
```

  - Her döngüde birer kez olacak şekilde döngü içi komutlar defalarca işletilecekler.

Verilerdeki Zamanda Yerellik

- Programlar özelikle döngülerde aynı değişkenlere defalarca ulaşırlar. Aşağıda **sum** ve **i** tekrar tekrar okunup/yazılıyor.

```
sum = 0;
for (i = 0; i < MAX; i++)
    sum = sum + f(i);
```

- Bazen çok sık kullanılan değişkenler CPU'nun kendi register'larında tutulur. Register CPU içi bellek demek, en hızlı kayıt budur aslında. Bunu derleyici halleder. Ancak bu her zaman mümkün olmayabiliyor.
  - Limitli sayıda register var.
    - Intel'de bize verilen 3-4. RISC'te 26 kadar.
  - Bazen değişken paylaşımlı ise veya dinamik atanmış belleklerde olunca mecburen ana bellekte veya cache bellekte tutulur.

Programlardaki Alanda (Uzaysal) Yerellik

- **Alanda yerellik (ing:spatial locality)** der ki programınız bellekte bir adrese bir kez ulaşırsa, o adresin devamındaki adreslere de ulaşma ihtimali çok yüksektir.

```
sub  sp, sp, 16
sw   ra, 0(sp)
sw   s0, 4(sp)
sw   a0, 8(sp)
sw   a1, 12(sp)
```

- Neredeyse tüm programlar (koşullu/koşulsuz sıçramalar hariç) Alanda yerelliğe uyar. Çünkü programlarda komutlar peş peşe işlenir.
  - Yani *i* adresindeki bir komut çalıştıysa, peşinden yüksek olasılıkla ***i+1*** adresindeki komut çalıştırılır.
- Döngü gibi kod parçaları hem zamanda, hem de alanda yerellik içerir.

Verilerdeki Alanda Yerellik

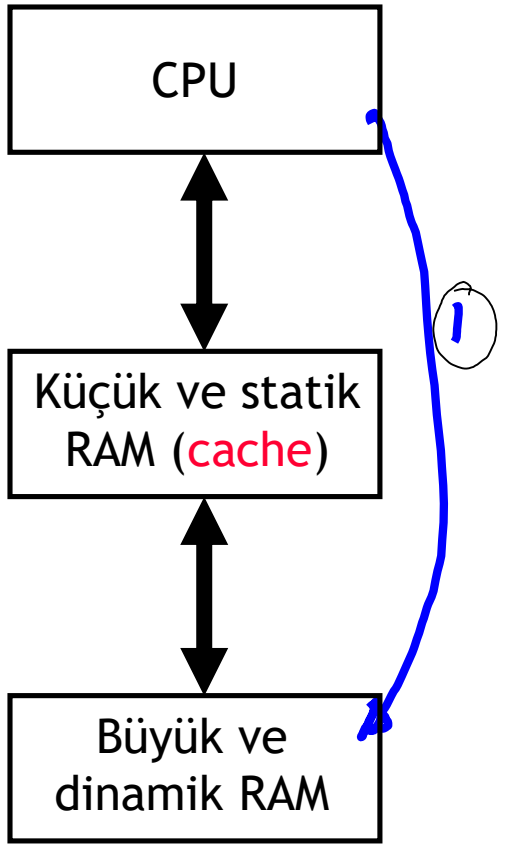
- Programlar sık sık bellekte peş peşe duran adreslere ulaşır.
  - Yandaki kodda görülen **a** isimli dizi gibi veriler bellekte ardışık adreslerde kayıtlıdırlar.
  - 32 bit, 64 bit gibi veriler ile string verilerin byte'ları da bellekte peş peşe durur.
  - Bağımsız alanlardan oluşan kayıt (ing:record) tipi veya yandaki **employee** gibi obje tipi veriler bellekte peş peşe byte'larda durur.
- Verilerde zamanda ve alanda yerellik birlikte olur mu?
  - Zor ama olabilir

```
sum = 0;
for (i = 0; i < MAX; i++)
    sum = sum + a[i];
```

```
employee.name = "Homer Simpson";
employee.boss = "Mr. Burns";
employee.age = 45;
```

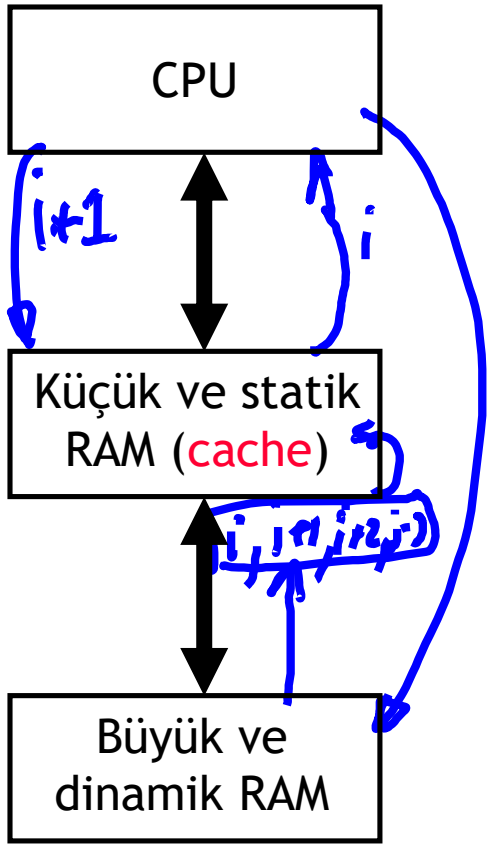
Cache'ler Zamanda Yerellik Avantajından Nasıl Yararlanır?

- İşlemci ana bellekten bir adrese ilk kez ihtiyaç duyduğunda bir kopyasını da cache belleğe yazar.
  - Bu adrese bir kez daha ihtiyacımız olduğunda, yavaş kalan dinamik belleğe başvurmaktansa, cache'deki kopyasını kullanırız.
  - Bu adrese tekrar ihtiyacımız olmazsa boşuna cache'e yazmış oluruz, olsun...
  - Demek ki her ilk okuma normal çalışmamızdan az biraz daha yavaş olur, çünkü hem ana belleğe, hem de cache beleğe ulaşmış oluyoruz.
  - Ama aynı bellek alanına tekrar ihtiyaç olunca bilgiler hızlı geliyor.
- Zamanda yerellik avantajı budur—sıklıkla kullanılan alanlar hızlı olan bellekte saklanır.



Cache'ler Alanda Yerellik Avantajından Nasıl Yararlanır?

- İşlemci ana bellekten ***i*** konumundaki bir adrese ilk kez ihtiyaç duyduğunda bir kopyasını da cache belleğe yazar.
- Ama sadece ***i*** konumuyla yetinmeyip bazı ek adresleri de cache yazarız, mesela ***i*** 'den ***i*** **+ 3** 'e kadar tüm alanları.
  - Eğer CPU sonradan ***i*** **+ 1,** ***i*** **+ 2 veya** ***i*** **+ 3** adreslerinden birisine ihtiyaç duyarsa, yavaş kalan dinamik belleğe başvurmaktansa, cache'deki kopyalarını kullanır.
  - Örneğin bir tamsayı dizi işlenirken cache'de 16 byte'lık satırlar varsa, hep dörder eleman yazılır.
  - Dört diyoruz da (4*4 = 16 byte) ne kadar fazla yan yana byte olursa o kadar iyi.
- Yine her ilk okuma normal çalışmamızdan az biraz daha yavaş olur (ing:performance penalty), ama komşu bellek alanlarına ihtiyaç olunca bilgiler hızlı geliyor.



Cache'ler Her Yerdeler

- Ön belleğe atma mimariye özel bir şey değil.
  - Cache'ler diğer tüm diğer alanlarda da var.
  - Hard diskler (64MB gibi)
  - Web tarayıcılar (cache klasörleri)
  - Dosya sistemleri
  - Google'un ön derlemesi yapılmış sorguları
  - TLB (?)

Diğer tip Cache'ler

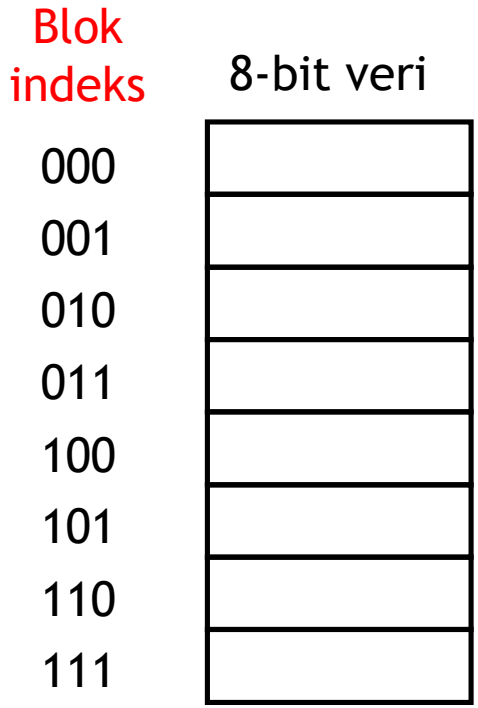
- Cache'lerin ardındaki mantık diğer tüm diğer alanlarda da var..
- Ağlar buna güzel bir örnektir.
  - Ağlarda oldukça yüksek gecikmeler yaşanıyor (ing;latency) ve çok düşük bant genişilkleri oluyor, bu yüzden tekrar tekrar aynı verilerin indirilmesi en istenmeyen şeydir.
  - Chrome ve Internet Explorer gibi araçlar çok sık bakılan web sayfalarını (en azından resimleri) sizin hard diskinizde tutabiliyor.
  - Ağ yöneticiniz bir ağ üzerinden cache açabildiği gibi, Akamai gibi firmalar da cache yapan servisler sunuyor.
- Diğer örnekler:
  - İşlemcilerde TLB "translation lookaside buffer" isimli bir cache sanal bellek işlemlerini desteklemek için kullanılıyor, bunu göreceğiz.
  - İşletim sistemleri sıklıkla ulaşılan disk bloklarını ana belleğe yazabiliyor. Sonra da bunlar CPU cache'ine geçiyorlar!

Vurduk ve Kaçırdık Durumları (Hits and Misses)

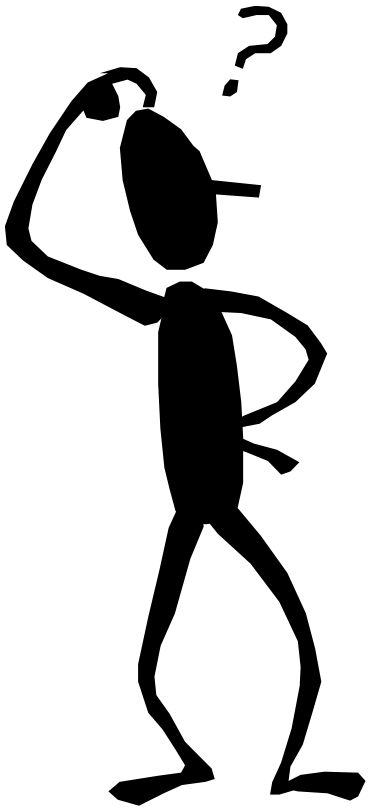
- Bir "**cache hit**" aranan verinin cache içinde olması demektir.
  - Hit almak iyidir, çünkü cache veriyi ana bellekten hızlı döndürür.
- Bir "**cache miss**" ise aranan verinin cache içinde olmaması demektir.
  - Miss almak kötüdür, bu durumda CPU yavaş olan ana belleğe başvurur ve veriyi bekler.
- Cache performansı için iki temel ölçüt vardır.
  - "**hit rate**" bellek erişimlerinin yüzde kaçı cache tarafından karşılanmış demektir. (0.0 →1.0 aralığında bir değerdir)
  - "**miss rate**" (= 1 – hit rate) bellek erişimlerinin yüzde kaçı yavaş olan ana bellek tarafından karşılanmış demektir.
- Tipik bir "cache hit rate" **%95** veya daha fazladır, bu da demektir ki bellek erişimlerinin çoğunluğu cache tarafından karşılanmış, sistem aşırı hızlanmıştır.
- Performansa gene döneceğiz, şimdi cache nasıl yapılıyor ona bakalım.

Basit Bir Cache ile Başlayalım

- Bu cache'ler bloklara bölünmüştür, blok sayıları farklı olabilir.
  - Ama genelde blok sayıları 2'nin katı olmalıdır.
  - Basitlik için şimdilik her blok bir byte tutsun diyelim.
    - Bunda alanda yerellik avantajı olamıyor, sonra düzelteceğiz.
- Aşağıda 8 adetlik, her alanında bir byte tutulan bir cache verilmiştir.



Dört Önemli Soru

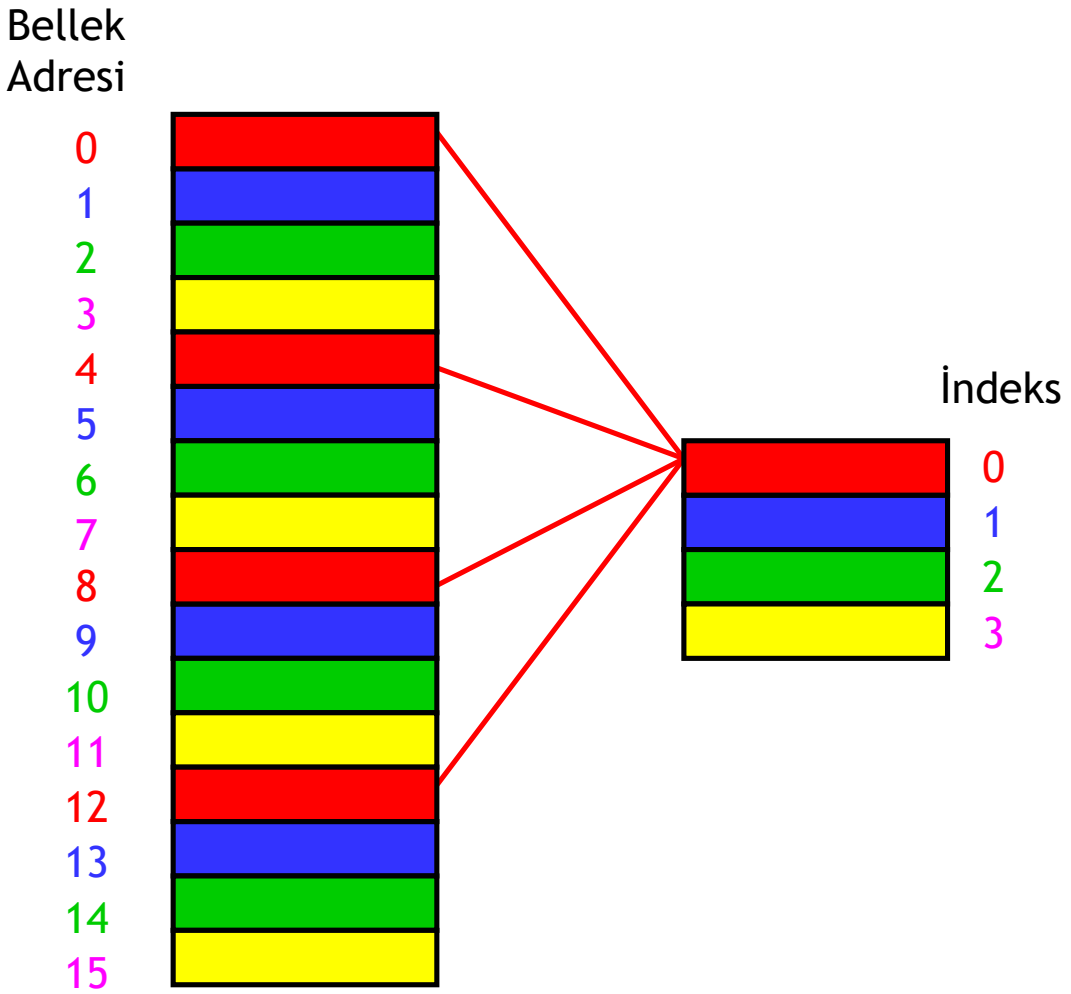


1. Ana bellekten cache içine bir veriyi alırken tam olarak nereye yazacağız?
2. Bir veri cache içinde var mı, yok mu nasıl söyleriz? Orada yoksa ana bellekten çekmek lazım da...
3. Eninde sonunda bu küçücük bellek dolacak ve ana bellekten gelen bir veri diğeri üzerine yazılacaktır. Bu alan hangisi olacak?
4. Cache ve ana belleğe yazma stratejimiz ne olacak? Veriyi geri yazma gerekirse nereye yazacağız?

- Soru 1 ve 2 birbirleriyle alakalıdır. Veriyi nereye yazdığımıza dair bir kural varsa o veriyi orada ararız!

Veriyi Cache İçinde Nereye Koyalım?

- Bir **Direkt Adresli (ing:Direct-Mapped)** cache basit bir yaklaşımla çözüm bulur: Her ana bellek adresi cache üzerinde bir bloğa atanmıştır.
- Örneğin, sağda 16-byte ana bellek ve 4-byte cache verilmiştir (4x1-byte blok).
- Kırmızı bellek pozisyonları **0**, **4**, **8** **ve** **12** hepsi cache üzerinde kırmızı blok **0**'a yazabilirler.
- Adres 1, 5, 9 ve 13 cache blok 1'i kullanır vb.
- Bu atamaları nasıl belirledik?



Atamalar MOD işlemi ile Bulunur…

- Ana bellek adresinin hangi cache bloğuna düşeceğini bulmanın bir yolu mod ya da kalan işlemi uygulamaktır.
- Cache 2^k blok içersin bellek adresi de *i* olsun bu adresin cache blok indeksi:

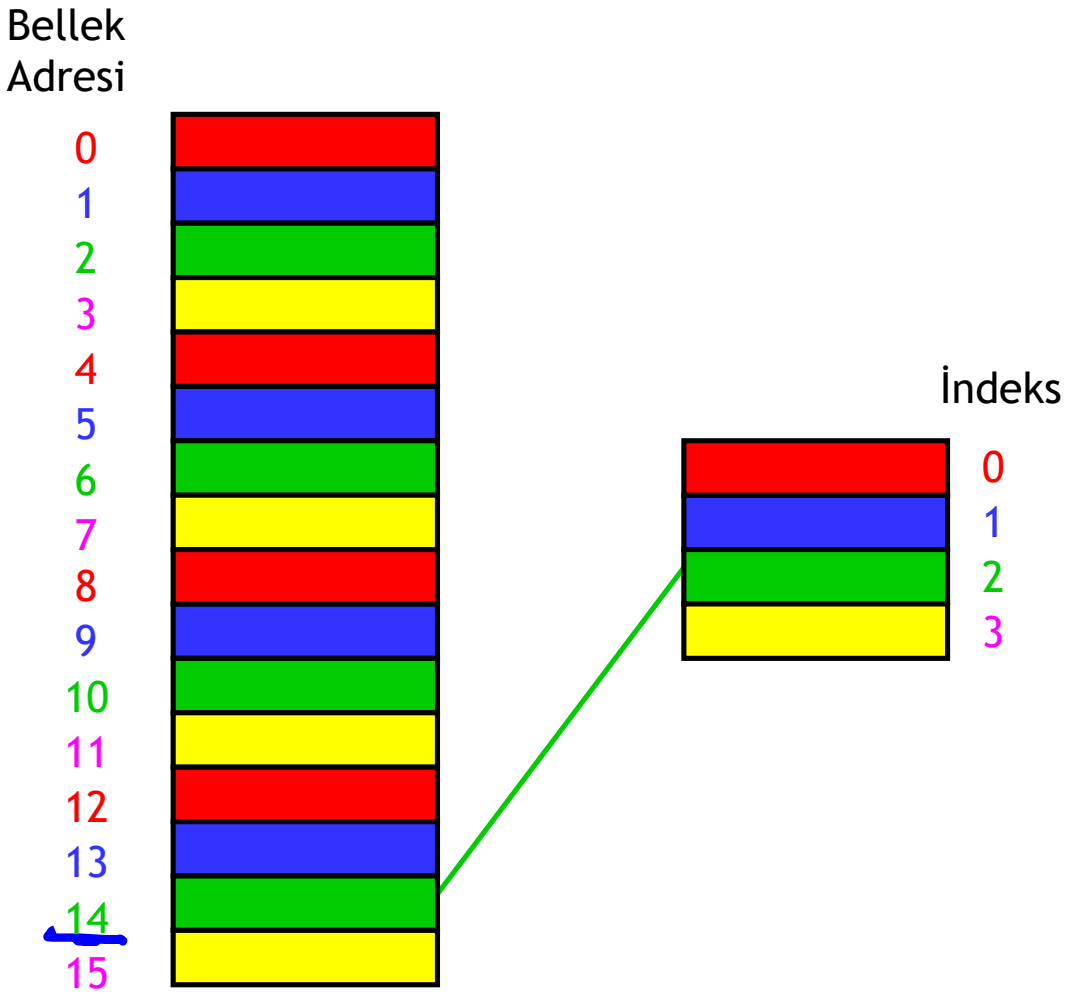
  $i \bmod 2^k$

- Mesela adres 14 cache blok 2'ye düşer

  14 mod 4 = 2

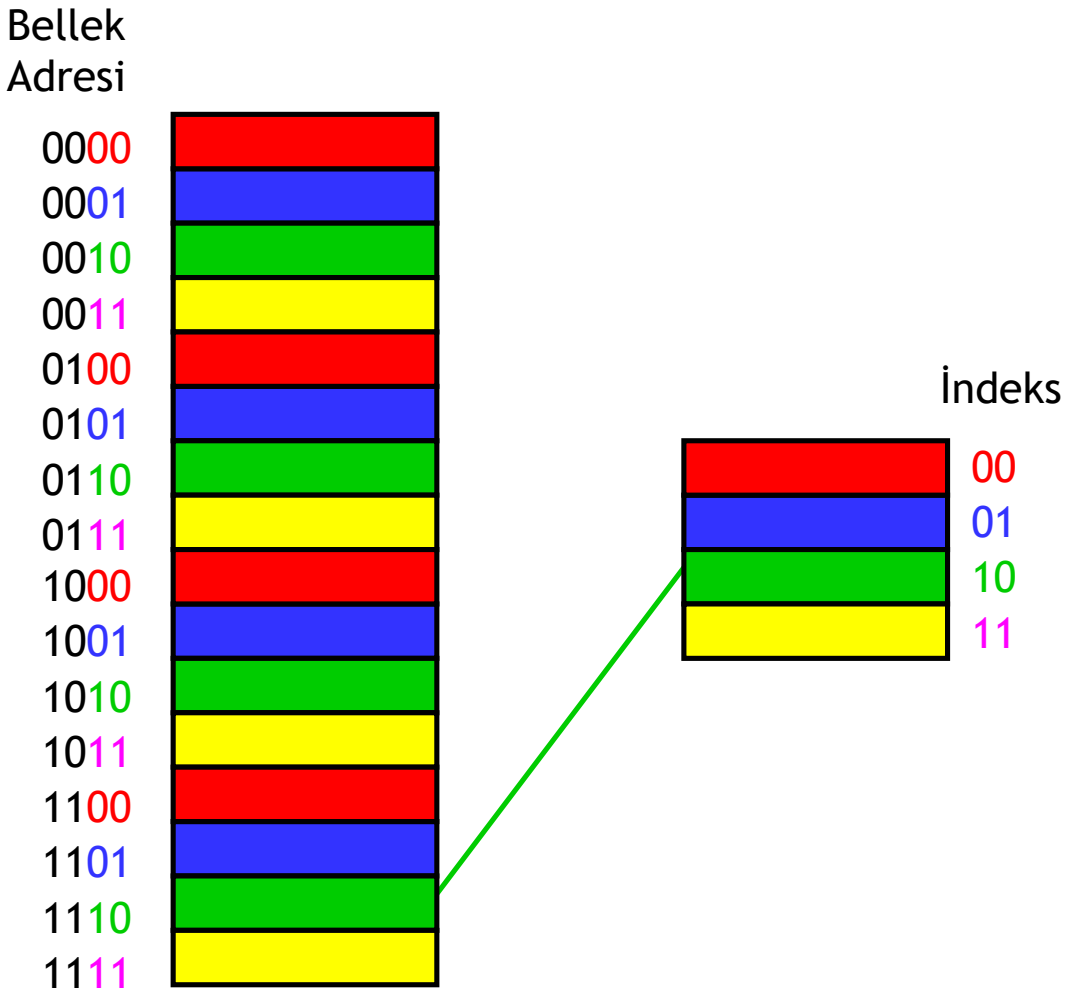
  Bölme işleminden kalan 2'dir.

  14 | 4
  12 | 3
  2



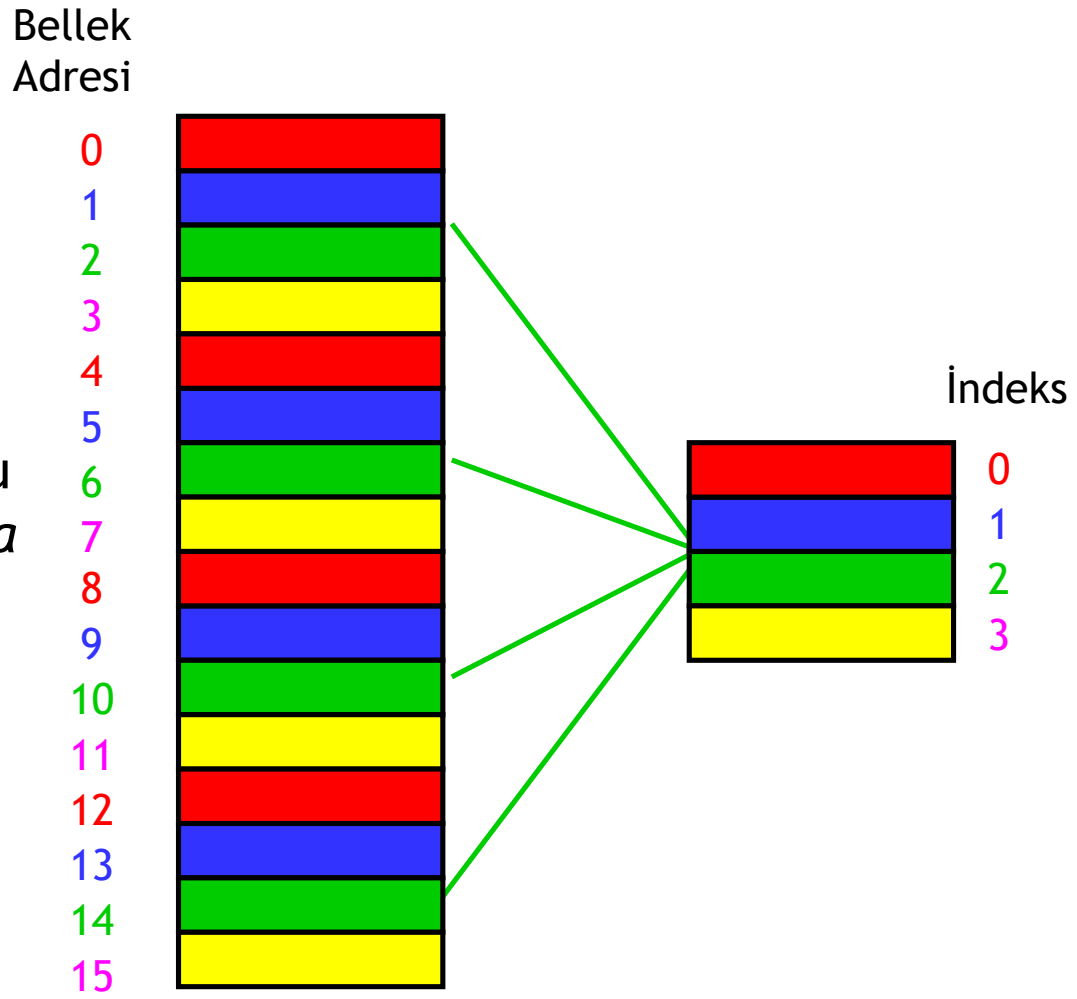
…veya En Az Önemli Bitlerle Bulunur

- Ana bellek adresinin hangi cache bloğuna düşeceğini bulmanın alternatif yolu, adresin sağda kalan bitlerini (ing:least significant *k* bits) kullanmaktır.
- 4 cache bloğumuz olduğundan adresin sağdan iki bitini kullanmalıyız.
- Yine 14 adresine bakarsak (binary : 1110) sağdaki iki bitten blok 2 (binary : 10) elde edilir.
- Sağdan *k* biti almakla mod 2^k işlemi aslında aynı şeydir.



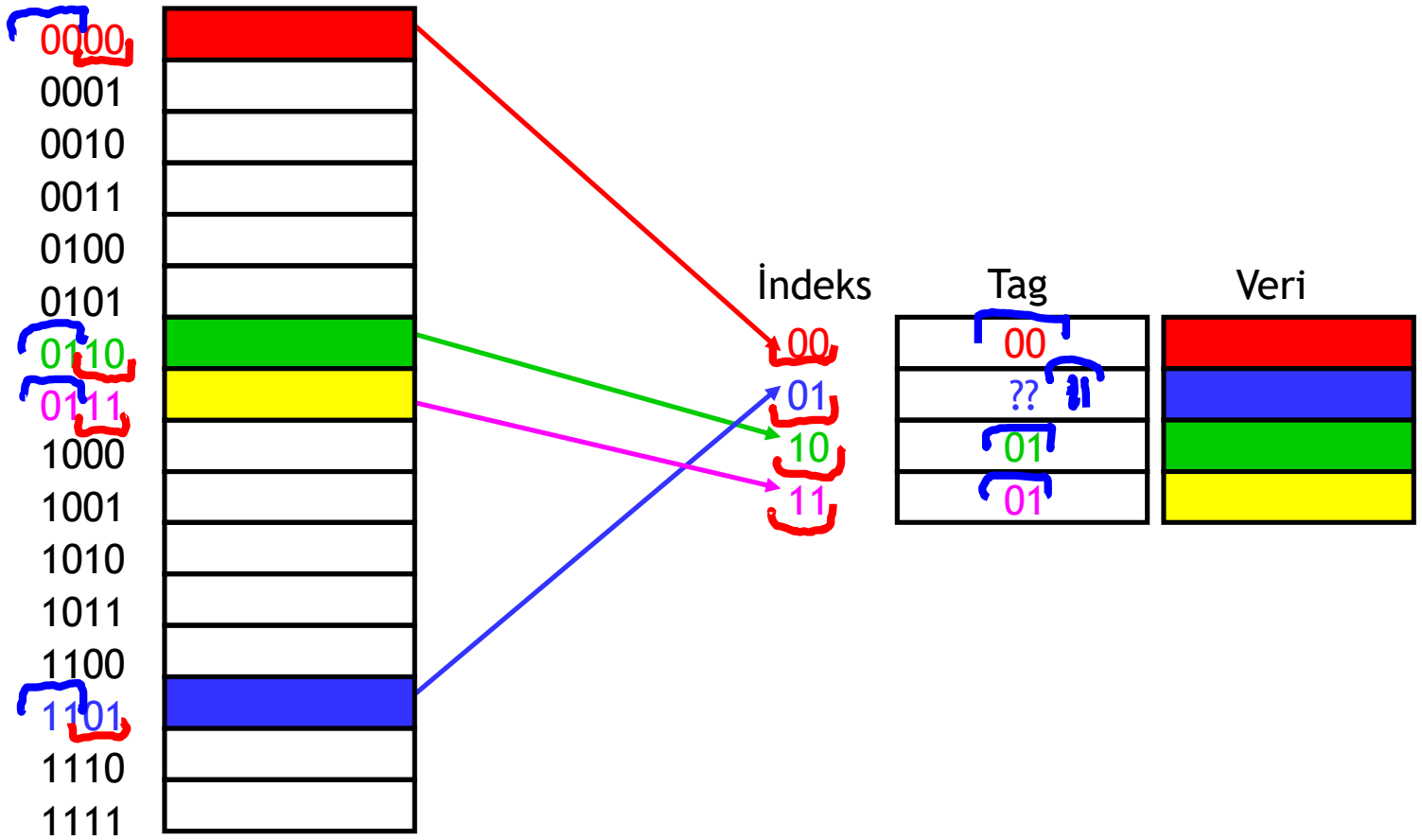
Cache İçinde Veriyi Nasıl Bulacağız?

- 4 önemli sorudan ikincisi neydi? İlgilendiğimiz veri acaba cache içinde tutuluyor mu nasıl belirleyeceğiz?
- İlgilendiğimiz yeşil *i* adresinin cache bloğunu mod işlemi ile buluruz.
- Onu buluruz da diğer yeşil adresler de mod işleminden sonra buraya yazılmış olabilir. Bunları nasıl ayıracağız?
- Bu örnekteki yeşil cache bloğu (2 numara) adres 2, 6, 10 *veya* 14 olabilir...



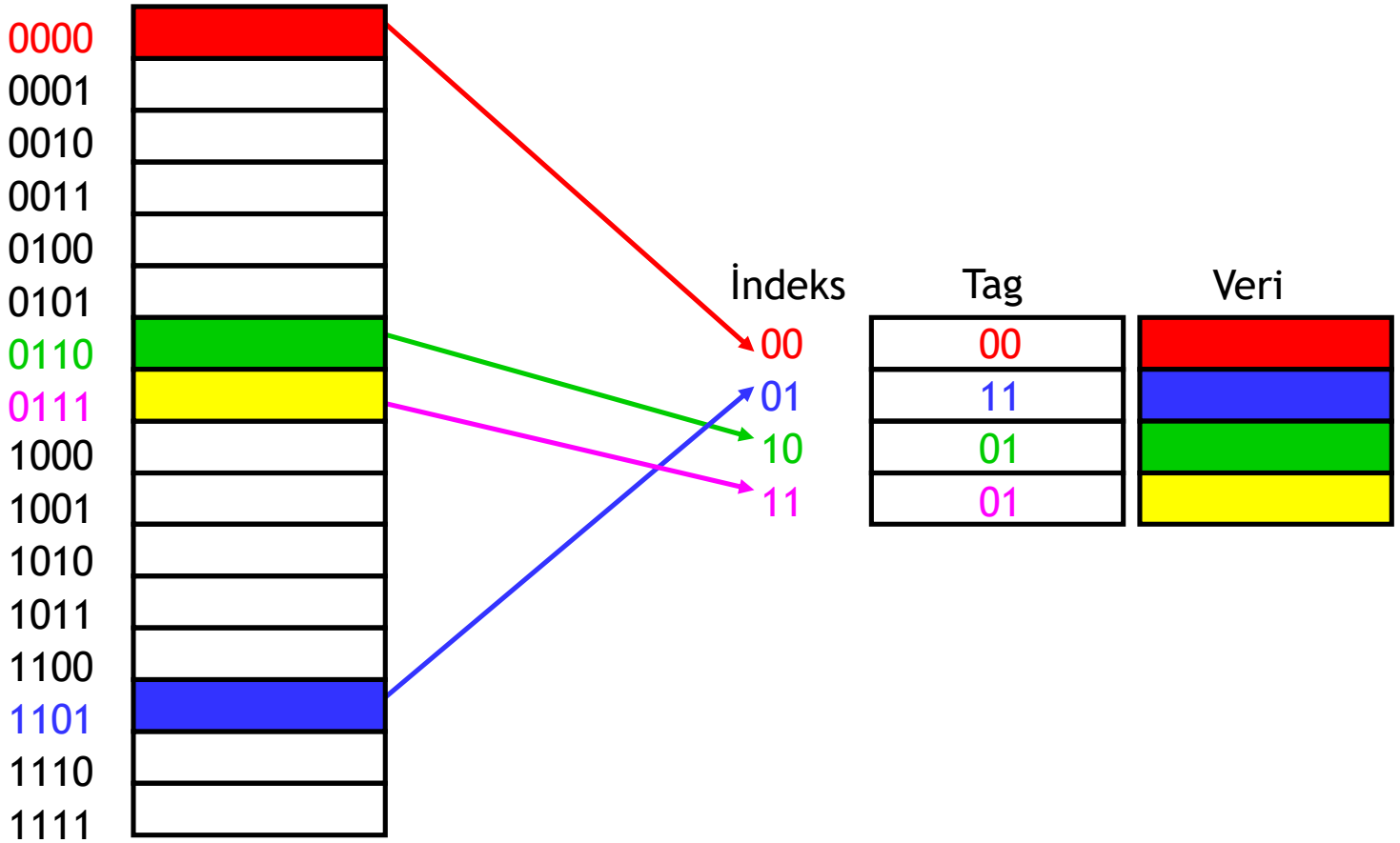
Etiket (Tag) Eklemek

- Bu durumda cache içine **etiketler (ing: tags)** eklemek zorunda kalırız.
- Bu etiketler şu an o blokta hangi adresli verinin durduğunu açıklar.
- Adreste solda kalan kısmı, bloğun yanına yazacağız.



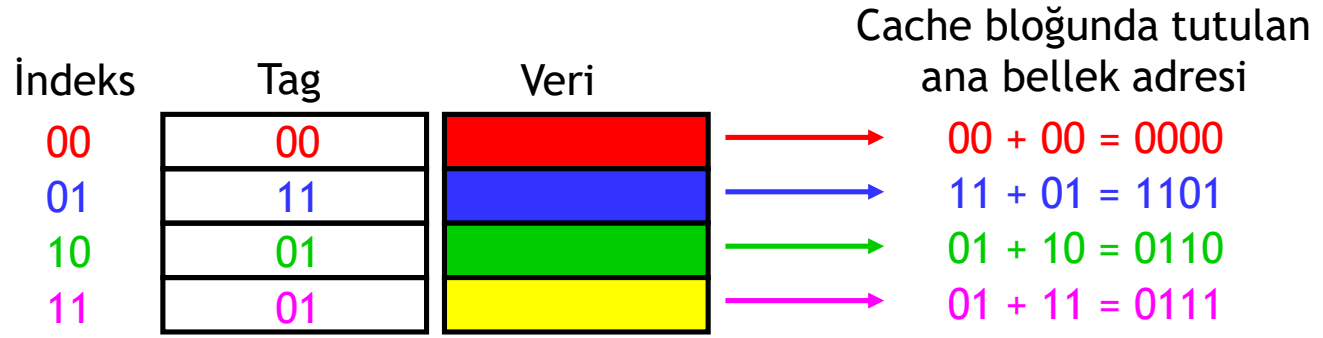
Etiket (Tag) Eklemek

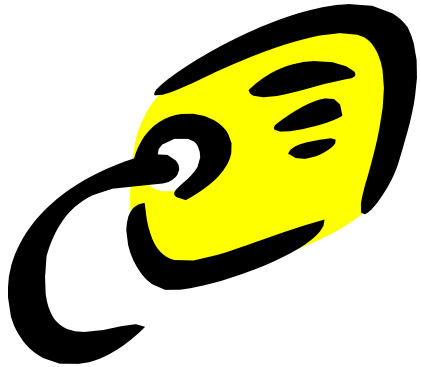
- Bu durumda cache içine **etiketler (ing: tags)** eklemek zorunda kalırız.
- Bu etiketler şu an o blokta hangi adresli verinin durduğunu açıklar.
- Adreste solda kalan kısmı, bloğun yanına yazacağız.
- İndeksi yazalım mı? Gerek yok onlar sabit...



Cache'mizde Neler Var Bakalım

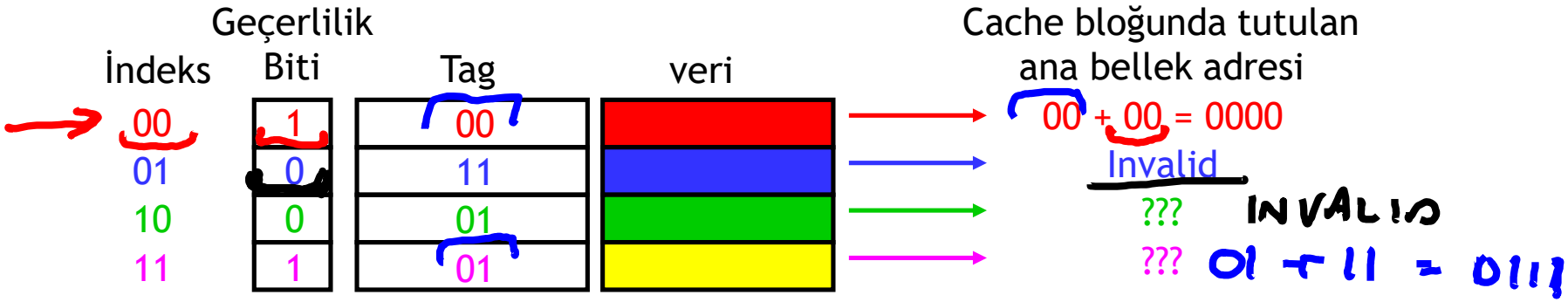
- Artık bir cache bloğunda hangi adresli veri tutuluyor bulabiliriz.
- Bunun için cache bloğunda yazılı tag değerine satır indeksini eklememiz gerekiyor ki tam adresi bulalım.





Bir Ek Daha Gerekli: Geçerlilik Biti (Valid Bit)

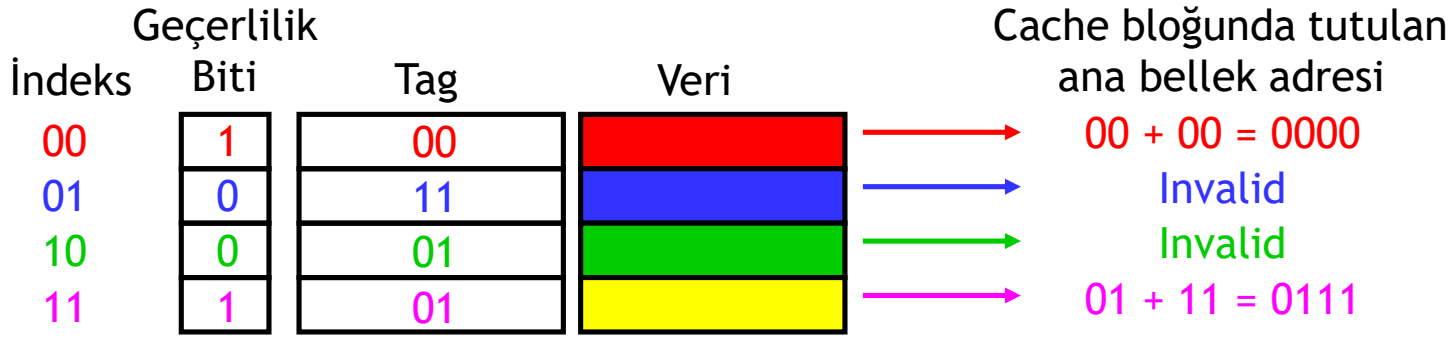
- Sistem açılınca cache boş olmalı veya içindeki verilere güvenemeyiz.
- Her cache bloğunun başına o blokta yazanın geçerli olduğunu veren bir **geçerlilik biti (ing:valid bit)** eklemeliyiz.
  - Sistem açılırken tüm bu bitler sıfır yapılır.
  - Veriler geldikçe önce bu bit bir yapılır.



- Yani cache içinde veriler olabilir ama geçerlilik biti o verinin kullanılıp, kullanılamayacağını bize söyler.

Bir Ek Daha Gerekli: Geçerlilik Biti (Valid Bit)

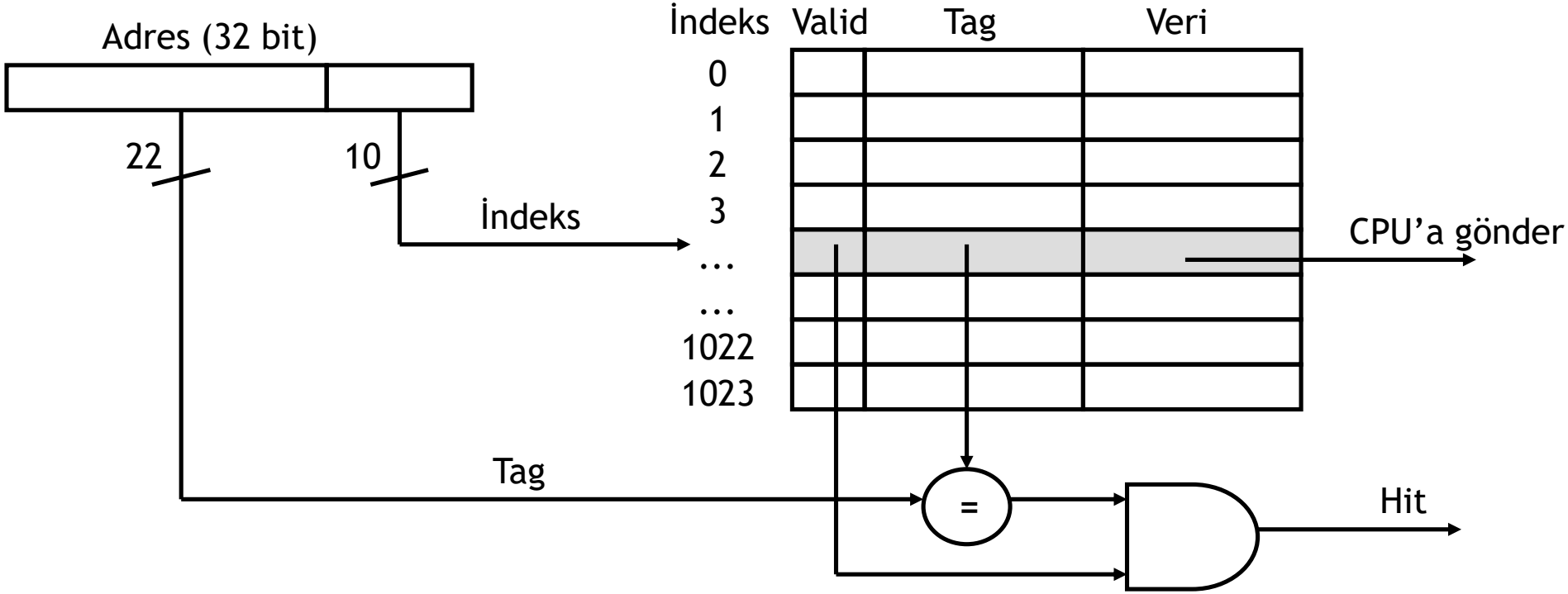
- Sistem açılınca cache boş olmalı veya içindeki verilere güvenemeyiz.
- Her cache bloğunun başına o blokta yazanın geçerli olduğunu veren bir **geçerlilik biti (ing:valid bit)** eklemeliyiz.
  - Sistem açılırken tüm bu bitler sıfır yapılır.
  - Veriler geldikçe önce bu bit bir yapılır.



- Yani cache içinde veriler olabilir ama geçerlilik biti o verinin kullanılıp, kullanılamayacağını bize söyler.

Cache Hit Durumda Neler Oluyor?

- CPU'nun bellekten bir veriye ihtiyacı olduğunda, verinin adresi **cache kontrolcüsüne** yollanır.
  - Adresin sağdan *k* biti cache'deki bir adresin indeksini verir.
  - Eğer blok geçerli veri tutuyorsa ve o satırda yazılı tag dediğimiz etiket ile *m*-bitlik adresin soldan (*m* – *k*) biti aynıysa, veri cache'dedir, CPU'ya bu veri yollanır.
- Aşağıdaki çizimde 32-bit bellek adresleri ve bir 2^{10}-byte cache kullanılmış.



Cache Miss Durumda Neler Oluyor?

- Cache hit alınca verimiz çok kısa bir gecikmeyle işlemciye gelir (1 saat sinyalinde, örneğin 4GHz makinede 0.25ns).
  - Eğer CPU ana belleği direkt kullansaydı, CPU ve bellek 50nsn'de çalışırsa anlaşırlar. Yani CPU 200MHz'i geçemez.
  - Biz çoğu bellek işleminin cache hit aldığını ve cache'in sayesinde CPU'nun hızlı çalıştığını söylüyoruz.
  - İşlemci hızı ne olursa olsun cache o hızla çalışır.
- Ancak cache miss yaşadıkça çok daha yavaş olan ana bellek erişimleri söz konusu oluyor.
- Sorunu aşmanın en basit yolu ana bellekten veri gelene dek CPU'yu uyutmak (ing:stall) , tabi bu işlem bitince veri cache'e de yazılmış olacak.
- %95 işlem 1 saat sinyalinde, %5 işlem (1+50) saat sinyalinde gerçekleşir.
  - Niye 1+50, çünkü cache'e sorduk yok (1 saat sinyali), ana bellekten getirdik (50 saat sinyali).

Cache hit ve miss genel performans formülü

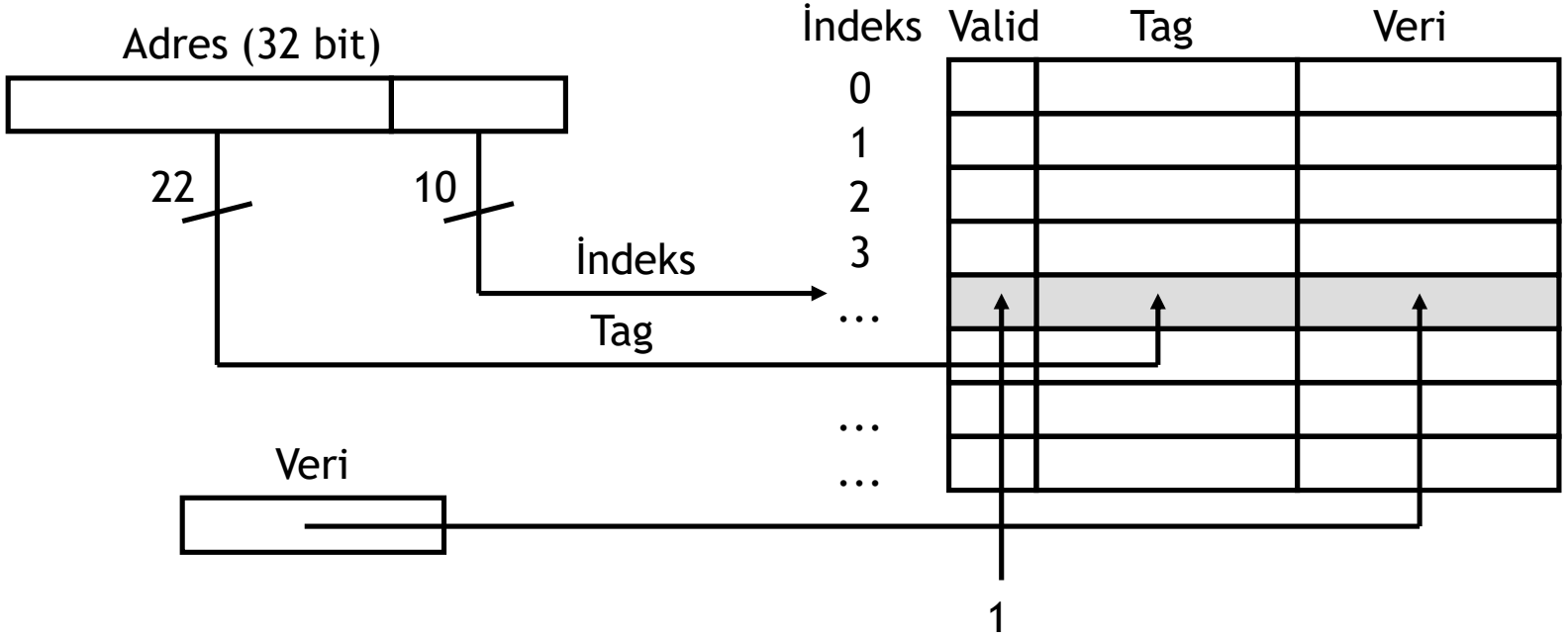
- Bir cache bellek yüzde olarak (0.0→1.0) hit oranında bellek işleminin C saat sinyalinde,
- Geri kalan işlerini de miss yada (1-hit) işlemi de bir üst bellekten C+M saat sinyalinde gerçekletirsin.
  - Niye C+M, çünkü cache'e sorduk yok (C saat sinyali), üst bellekten getirdik (M saat sinyali).
- Bu durumda:
  - Süre = hit x C + miss x (C+M)
  - Süre = (hit+miss) x C + miss x M # Not: (hit+miss) = 1
  - Süre = C + miss x M

Cache Miss Durumda Neler Oluyor? (devamı)

- L1 cache'imiz 1 saat sinyalinde cevap versin, %95 hit oranı olsun, ana bellek 50 saat sinyalinde cevap versin.
- %95 işlem 1 saat sinyalinde, %5 işlem (1+50) saat sinyalinde gerçekleşir.
  - 0.95x1 + 0.05x(1+50) = 3.5 saat sinyali
  - Ya da 1+0.05x50 = 3.5 saat sinyali
  - L1 cache olmasa 50 saat sinyaliydi !
- Araya bir L2 cache ekleyelim. L2 cache'imiz 10 saat sinyalinde cevap versin, %90 hit oranı olsun, sonuç ne olur?
  - 0.95x1 + 0.05x(L2+RAM gecikmesi) haline dönüşür, parantez içini hesaplayalım:
    - L2+RAM gecikmesi = 0.90x10 + 0.10x(10+50) = 15 saat sinyali
    - Ya da L2+RAM gecikmesi = 10 + 0.10x50 = 15 saat sinyali
    - Burası bile 50 yerine 15 saat sinyali.
  - Bulunan değeri parantezin içine yazarsak:
    - 1 + 0.05x(15) = 1.75 saat sinyali
    - L1+L2 cache daha güzel ! Bir saat sinyaline yakın.

Cache'e Bir Blok Yüklemek

- Mecbur kaldık, ana bellekten okuduk, bunu cache'e de yerleştiririz.
  - Adresin sağdan *k* biti cache'deki bir adresin indeksini verir.
  - *m*-bitlik adresin soldan (*m* – *k*) biti o satırdaki tag dediğimiz etiket alanına yazılır.
  - Veri (sadece bir byte) veri alanına yazılır.
  - Blok geçerlilik biti bir yapılır.



Cache'de Yazmak İstediğimiz Alan Doluysa?

- Dört önemli sorudan üçüncüsü, yazmak istediğimiz alanda geçerlilik biti 1 olan başka bir veri var.
- Bunu detaylı cevaplayacağız!
  - Cache miss almışsak otomatik o alana veri olsa da yazacağız.
  - Bu işlem "en az kullanılanın üzerine yaz politikasıdır" (ing: least recently used replacement policy) ve iddiası o en eski veriye sistemin artık ihtiyacı olmaz şeklindedir.
- Gelecek konularda tekrar değiniriz.

HIT Oranına Örnek

- A[i] cache'de varsa A[i+1]'de vardır, A[i+2]'de … Daha ne kadar mümkün?
- Cache'de 64 byte yan yana tutulabilsin.
- 64/(tamsayı boyutu) = 64/4 = 16 tamsayı tutulur.
- A(i) cache miss yaşatır sadece…

A[i+1} → A[i+15] cache hit yaşatır.

- Yani 1/16 miss, 15/16 = 0.9375 hit.
- En basit dizi de bile **%93,75** hit alıyorsunuz.
- Fakat bir **sum** değişkeni, 10binlik döngüde 20bin kez kullanılırsa:
- 1/20,000 miss, 19,999/20,000 = 0.99995 hit.
- Tüm program ortalamasının hit oranının **>%95** olması çok doğaldır.

Özet

- Bu derste **cache** mantığını kısaca açıkladık.
  - **Alanda ve zamanda yerelliğin** sağladığı avantajlarla küçücük ama hızlı ve pahalı bir belleğin sistemin ortalama bellek erişimi performansına nasıl derin etkisi oluyormuş gördük.
  - Bir cache yaptık onu **bloklara** böldük, her bloğa bir **valid bit**, o alanda hangi adresli veri var gösteren bir **tag** alanı ve veri alanı koyduk.
- Gelecek konumuzda gelişmiş cache yapılarını ve sistem bellek performansı nasıl hesaplanır göreceğiz.