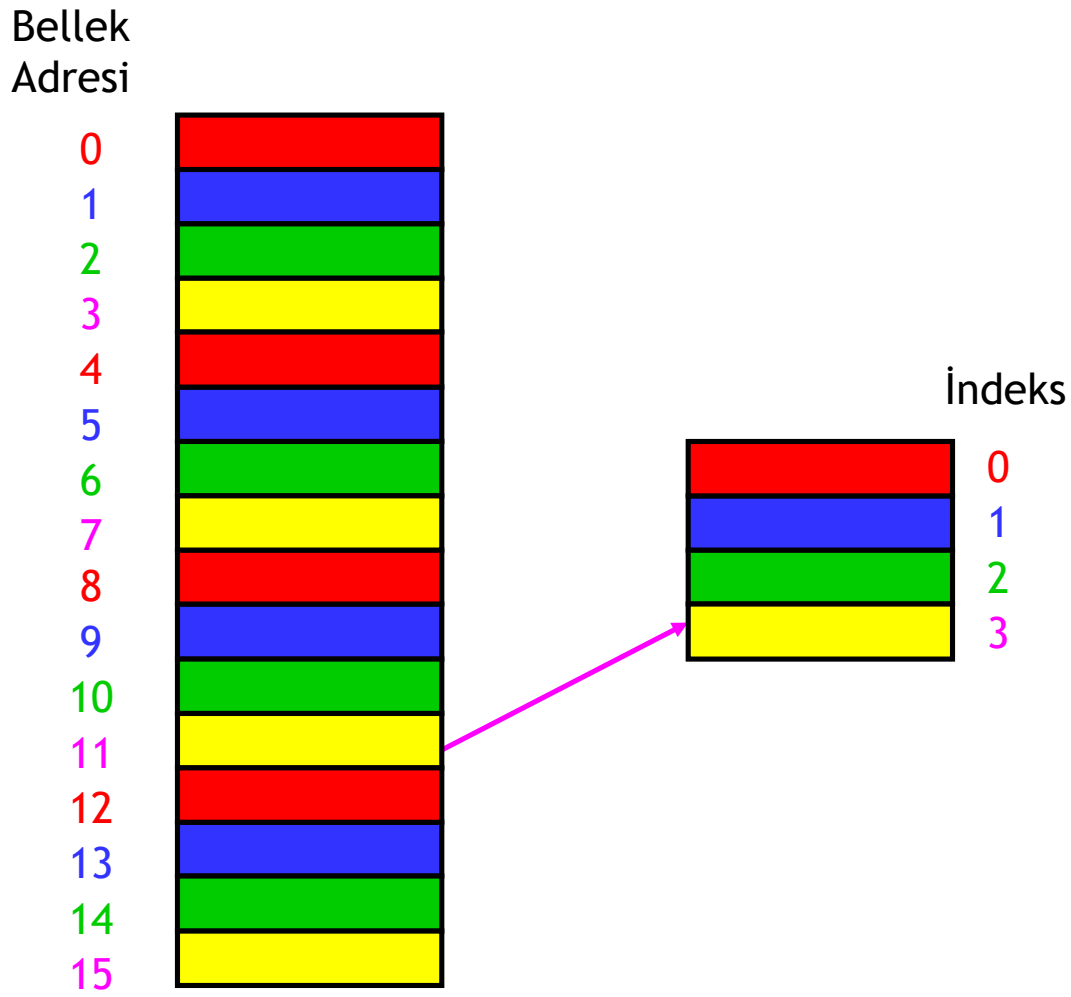
Önbellekler - 2

Daha gelişmiş cache bellekler

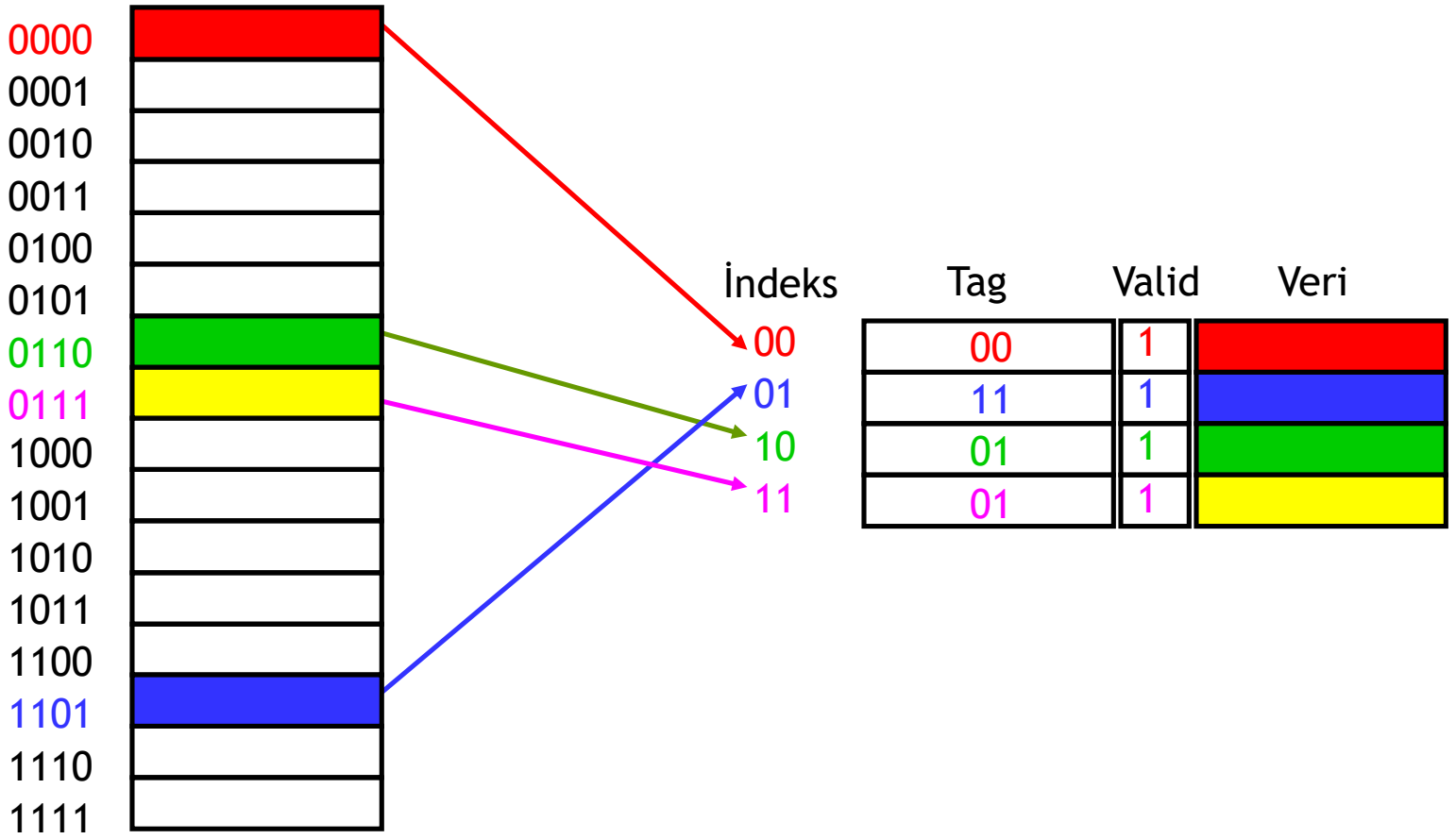
Direkt Atamalı Cache'ler

- Eğer cache 2^k blok tutuyorsa, sağdan k bit (ing: least significant bit (LSBs)) indeks için kullanılır.
  - adresi *i* olan bir veri *i* mod 2^k loğunda tutulur.
- Öreğin, bellek adresi 11 lan veri 11 mod 4 = 3 bloğunda tutulur. Zaten 1011 adresinin son iki biti de 11'dir.



Tag ve Valid Bitleri

- 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...



Bir Cache Ne Kadar Büyür?

- Cache kullanılan 32-bit adresli ve byte-byte adreslenebilen bir makinede şu karakteristik özellikler vardır :
  - Direkt atamalıdır
  - Her blok bir byte tutar
  - Cache indeks değeri adresin sağda kalan son 8 bitidir.

- Burada iki soru sorulabilir:
  - Cache kaç blok tutar?

  - Bu cache yapısı için kaç bit kullanılır (*veri*, tag, valid bitleri toplamı)?

... Devam slayta bakın

Bir Cache Ne Kadar Büyür?

- Cache kullanılan 32-bit adresli ve byte-byte adreslenebilen bir makinede şu karakteristik özellikler vardır :
  - Direkt atamalıdır (önceki derste gördük)
  - Her blok bir byte tutar
  - Cache indeks değeri adresin sağda kalan son 8 bitidir.

- Burada iki soru sorulabilir:
  - Cache kaç blok tutar?

    8-bit indeks → 2^8 = 256 blok

  - Bu cache yapısı için kaç bit kullanılır (*veri*, tag, valid bitleri toplamı)?

    tag boyutu = 24 bit (32 bit adres - 8 bit indeks)
    (24 tag biti + 1 valid biti + 8 veri biti) x 256 blok
    = 33 bit x 256 = 8448 bit

Diğer Cache Yapıları

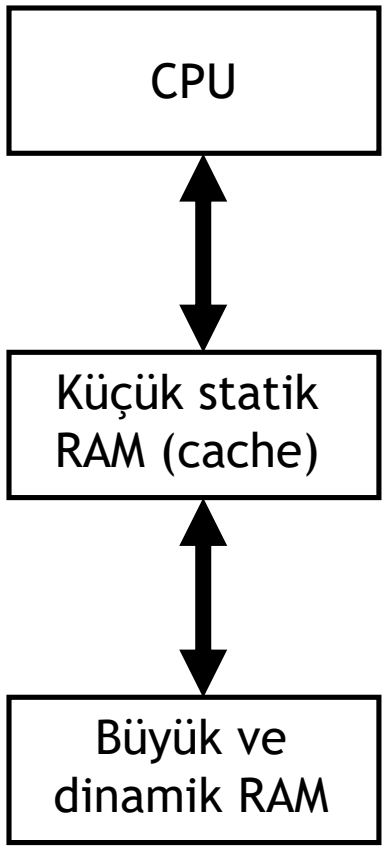
- Bugün, daha üstün, alternatif cache bellek yapılarından bahsedeceğiz.
  - Uzaysal yerellik (ing: spatial locality) avantajından da yararlanacağız ama nasıl olacak?
  - Bir de sürekli bir bloğun üzerine yazma yapılan potansiyel bir sorun çıkacak, onu nasıl aşacağız?

- Önce cache performansı üzerine konuşarak konuya girelim...

Bellek Sistemi Performansı

- Bir bellek sisteminin performansıdan bahsetmek için, birkaç önemli faktöre odaklamamız lazım.
  - Cache'den CPU'ya veri yollamak ne kadar sürer?
  - Ana bellekten Cache'e veri yazmak ne kadar sürer?
  - Ana belleğe ulaşım ne kadar sıklıkla oluyor?

- Bu soruların hepsinin birer "değişken ismi" var.
  1. **Hit time** Cache'den CPU'ya veri yollama süresidir. Çok ta hızlıdır, 1-3 saat sinyali.
  2. **Miss penalty** Ana bellekten Cache'e veri yazma süresidir. Onlarca saat sinyaline malolur.
  3. **Miss rate** istenen verinin cache'de olmaması oranıdır.



Ortalama Bellek Erişim Zamanı

- **Ortalama Bellek Erişim Zamanı (ing: average memory access time)** veya **AMAT** şöyle bulunur.

$$\text{AMAT} = \text{Hit time} + (\text{Miss rate} \times \text{Miss penalty})$$

Parantez içi aranan verinin cache olmayıp getirilmesi zamanı olduğundan, bu hesap hit+mis gibi bir ortalama değer verir.

- Neden Hit time x Hit rate dememiş?
  - Çarparsak 0.95, çarpmazsak 1 olur.
  - Yani küçük bir değer, artıdan sonrası önemli.
- Bir sistemin Ortalama Bellek Erişim Zamanı nasıl arttırılır?
  - Açıktır ki küçük AMAT iyidir.
  - Miss penalty değeri hit time değerinden büyük o halde düşük AMAT için
    - Ya miss penalty düşecek.
    - Ya da miss rate düşecek.
- Sadece AMAT sistem performansı değildir. Programlarınızın çalışma zamanı (ing: execution time) en geçerli performans ölçütüdür.

Performans Örneği

- 1000 satırlık bir programda %33 oranında bellek kullanımı var. Cache'in hit oranı %97 ve aranan bilgi cache'de ise (hit time) bir saat sinyalinde CPU'ya geliyor, ama değilse (miss penalty) 54+1 saat sinyali sürüyor.

AMAT = Hit time + (Miss rate x Miss penalty)

=

=

- Miss rate nasıl düşürülür?

... Sonraki slayta bakın

Performans Örneği

- 1000 satırlık bir programda %33 oranında bellek kullanımı var. Cache'in hit oranı %97 ve aranan bilgi cache'de ise (hit time) bir saat sinyalinde CPU'ya geliyor, ama değilse (miss penalty) 54+1 saat sinyali sürüyor.

AMAT = Hit time + (Miss rate x Miss penalty)
= 1 saat sinyali + (0,03 x 55 saat sinyali)
= 2,65 saat sinyali

- Programınızda %33 oranında bellek kullanımı var demiştik. Başka bilgi yok, demek ki geri kalan tüm komutlar 1 saat sinyalinde bitiyor olarak varsayılırsa, toplam çalışma zamanı nedir?

T = 1000 x (0,33 x 2,65 + 0,67 x 1)
= 1000 x 1,5445 = 1545 saat sinyali

Not: Virgülden sonrası her zaman yukarı yuvarlanır.

Miss Rate Nasıl Düşürülür?

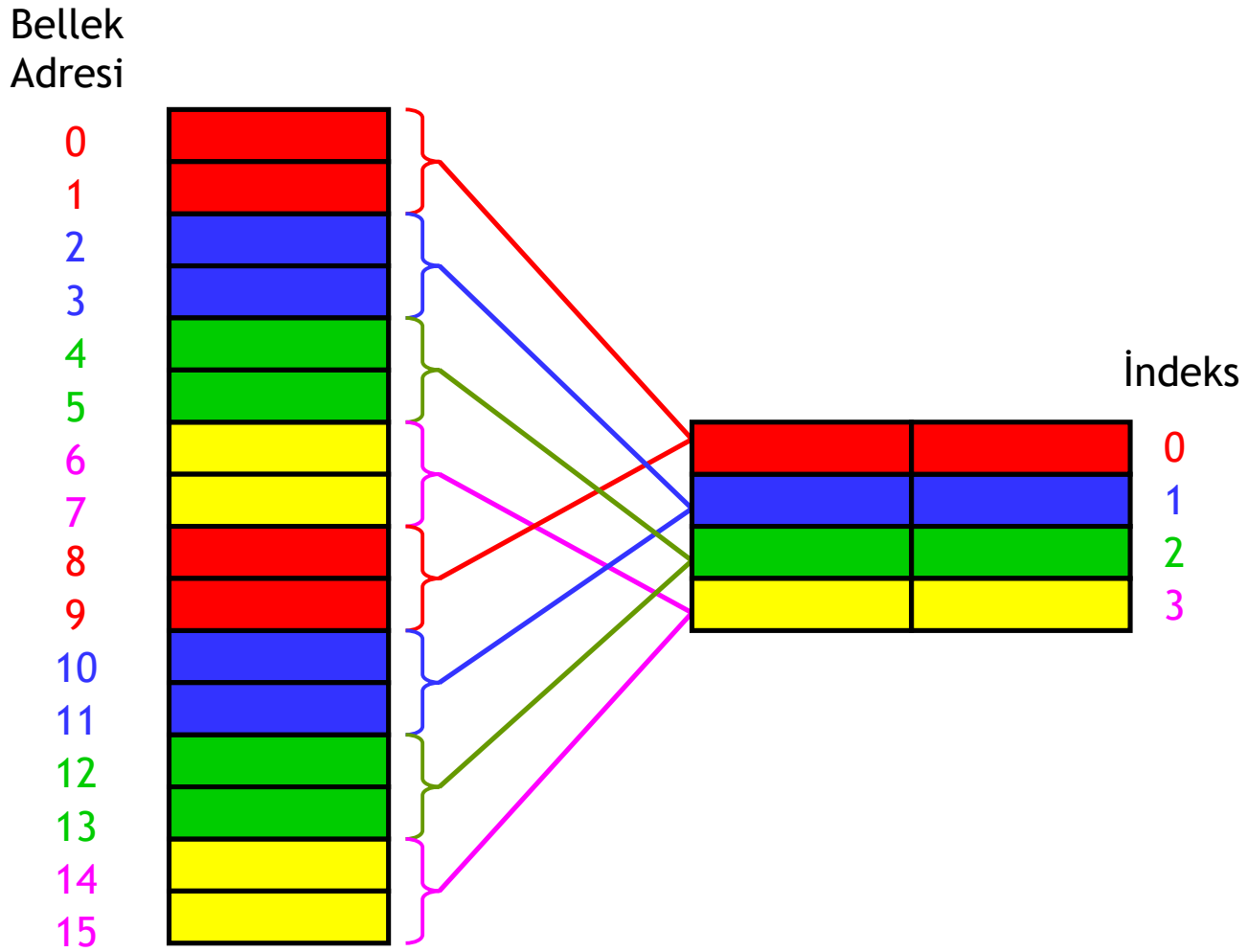
- Cache mükemmel olsa hiç kaçırmaz, AMAT=1 our.
- Ama sadece %3 kaçakla (miss rate) bile, AMAT değeri %165 artıyor!
- Acaba ikinci bir seviye cache mi koysak?
  - Evet kesinlikle... Onun hit oranı %95 bile olsa, süresi 10 saat sinyali olsa herşey değişir.
  - L1 konuşuyorduk, L2 cache gerekti.

- Her bloğunda bir byte bilgi tutan cache uzaysal yerellikten (ing: Spatial locality) yararlanamaz,
  - Hani bellekte bir adrese ihtiyacınız varsa, peşindeki verilere de ihtiyacınız olacaktır varsayımımız vardı ya..
- O biraz çare olur muydu?
- Ne yapabiliriz?

Uzaysal Yerellik

- Cache bloğu bir byte'tan daha çok veri tutsun diye ne yapılabilir?

- Yanda her blokta iki byte tutan bir cache yaptık, yani artık her yüklemede iki byte getireceğiz.
- Adres 12 bellekten okunduğunda komşu adresler 12 *ve* 13 birlikte 2 nolu cache bloğuna alınmalıdır.

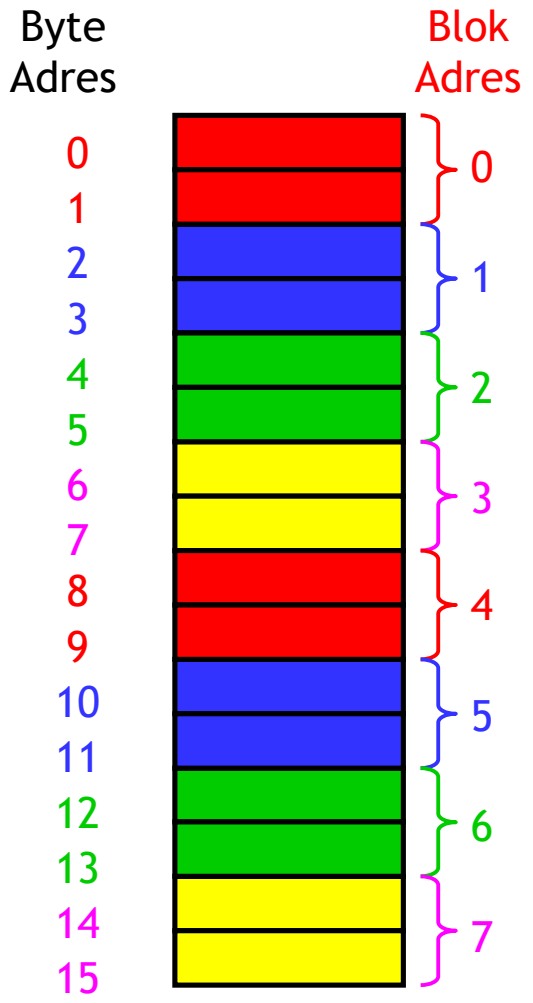


Blok Adres

- Bu cache içinde verilerin nasıl yerleşmiş olacağını nasıl gösterebiliriz?
- Artık **blok adres** zamanıdır! Eğer cache blok boyutu 2^n byte ise, ana belleği de 2^n-byte parçaya bölebiliriz.
- Adres *i*'den blok adresini hesaplamak için, tamsayı bölme işlemi yapılabilir

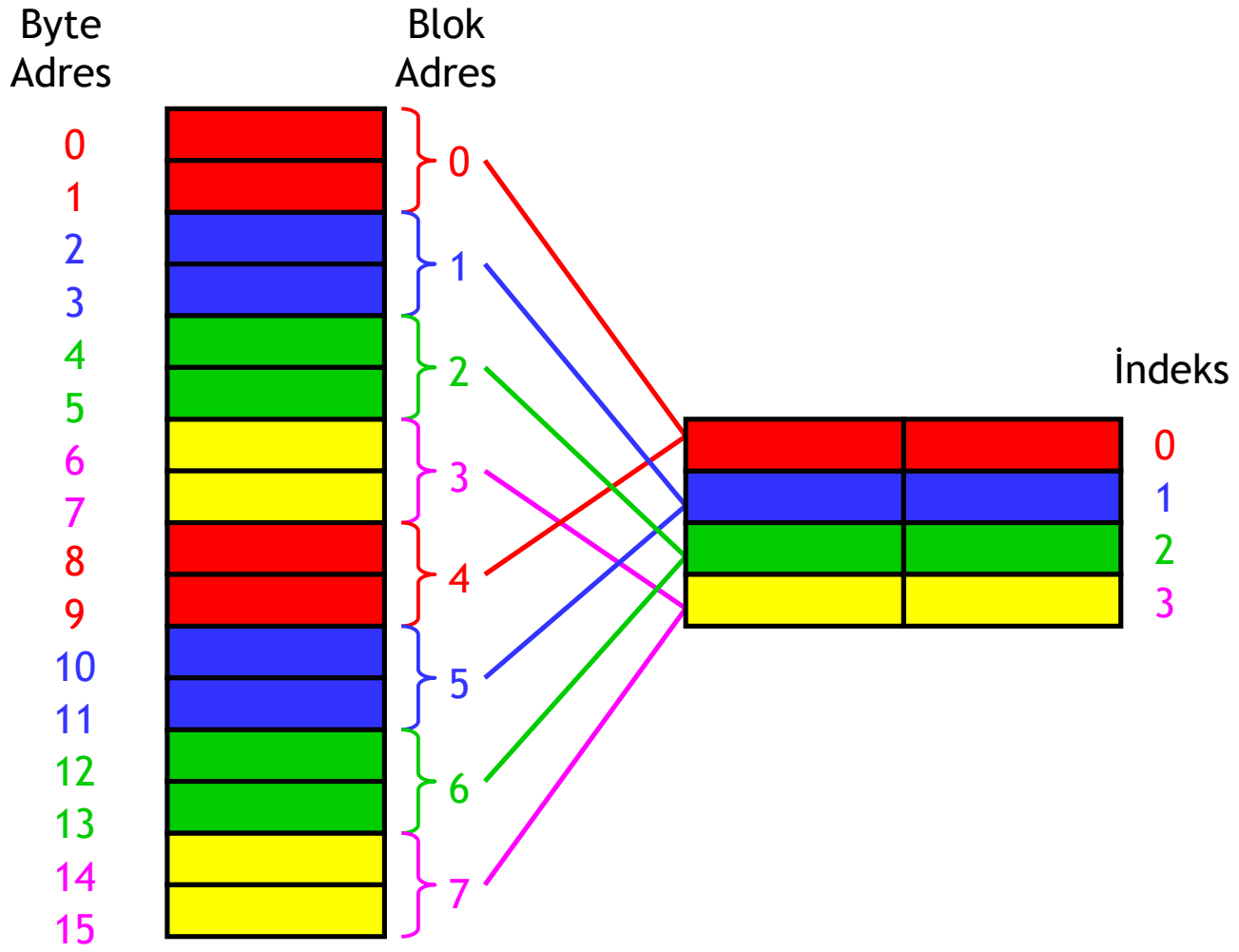
$$i \;/\; 2^n$$

- Örneğimiz iki-byte cache blokları içeriyor, yani 16-byte'lık ana belleği "8-blok" ana bellek olarak düşünebiliriz.
- Örneğin, bellek adresi **12** ve **13 beraber** blok adresi **6**'ya denk gelirler çünkü **12 / 2 = 6** ve **13 / 2 = 6** olmaktadır.



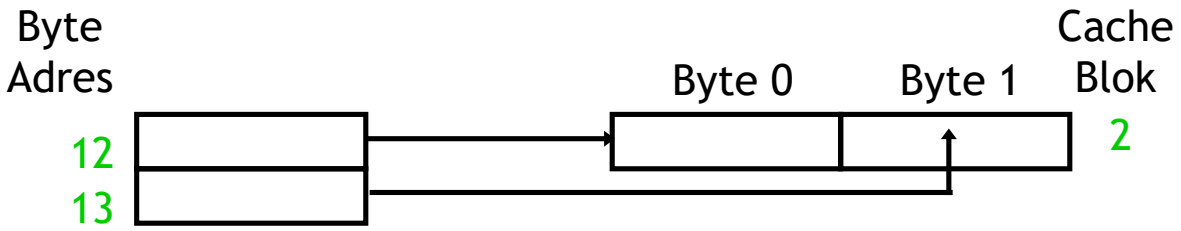
Cache Yerleşimi

- Bir kez blok adresini elde edince , blok adresini cache satır sayısına böldüğümüzde çıkan sayı verinin cache'de hangi satıra yerleşeceğini ifade eder.
- Bizim örneğimizde, bellek bloğu **6** cache bloğu **2**'ye yerleşir çünkü **6 mod 4 = 2**.
- Yani bu durumda **12** ve **13** ana bellek adresleri cache blok **2**'ye yazılırlar.
- Daha kolay bir yolu olmalı değil mi?



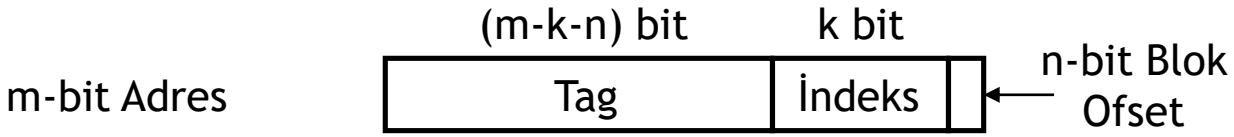
Blok Numarasına Göre Veri Yerleşimi

- Artık biz bellekten bir adrese ulaştığımızda, uzaysal yerellikten yararlanmayı umarak onun ait olduğu tüm bloğu cache'e yazarız.
- Örneğimizde, program adres 12'deki byte veriye ulaşınca onun ait olduğu tüm blok 6 (yani adres 12 ve 13) cache blok 2'ye yazılır.
- Önemli not: Eğer adres 13 gerekli olurda önce okunursa, onun da blok adresi 6 olduğundan, yine adres 12 ve 13 cache blok 2'ye yazılır.
  - Bu bir dizi ise ve başlangıcı 13 adresi ise, belki adres 12 boşuna yüklenecek, ama katı kurallarımız var, kuralları bozamayız.
- Basitleştirelim; adres *i* hangi bellek bloğuna düşüyorsa o blok cache'e tümüyle yazılacak.

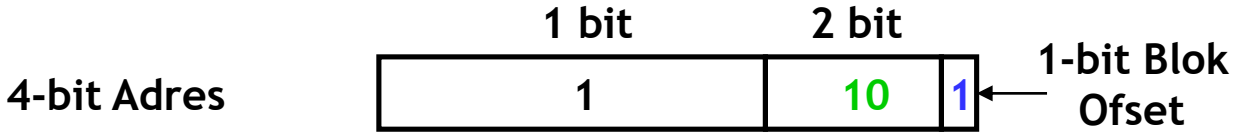


Kolay Yoldan Verinin Cache İçindeki Pozisyonu

- Şimdi elimizde 2^k bloklu, her biri 2^n byte içeren bir cache var.
- Ana bellek adreslerinin cache içindeki yerleşimini kolay yoldan şöyle bulabiliriz:
  - Adresin *k* biti 2^k cache blok içinden birini seçer.
  - Ama adresin sağdan *n* biti de verinin cache bloğu içinde kaçıncı kolona veya ofsete yazılacağını belirtir (2^n kolon var).

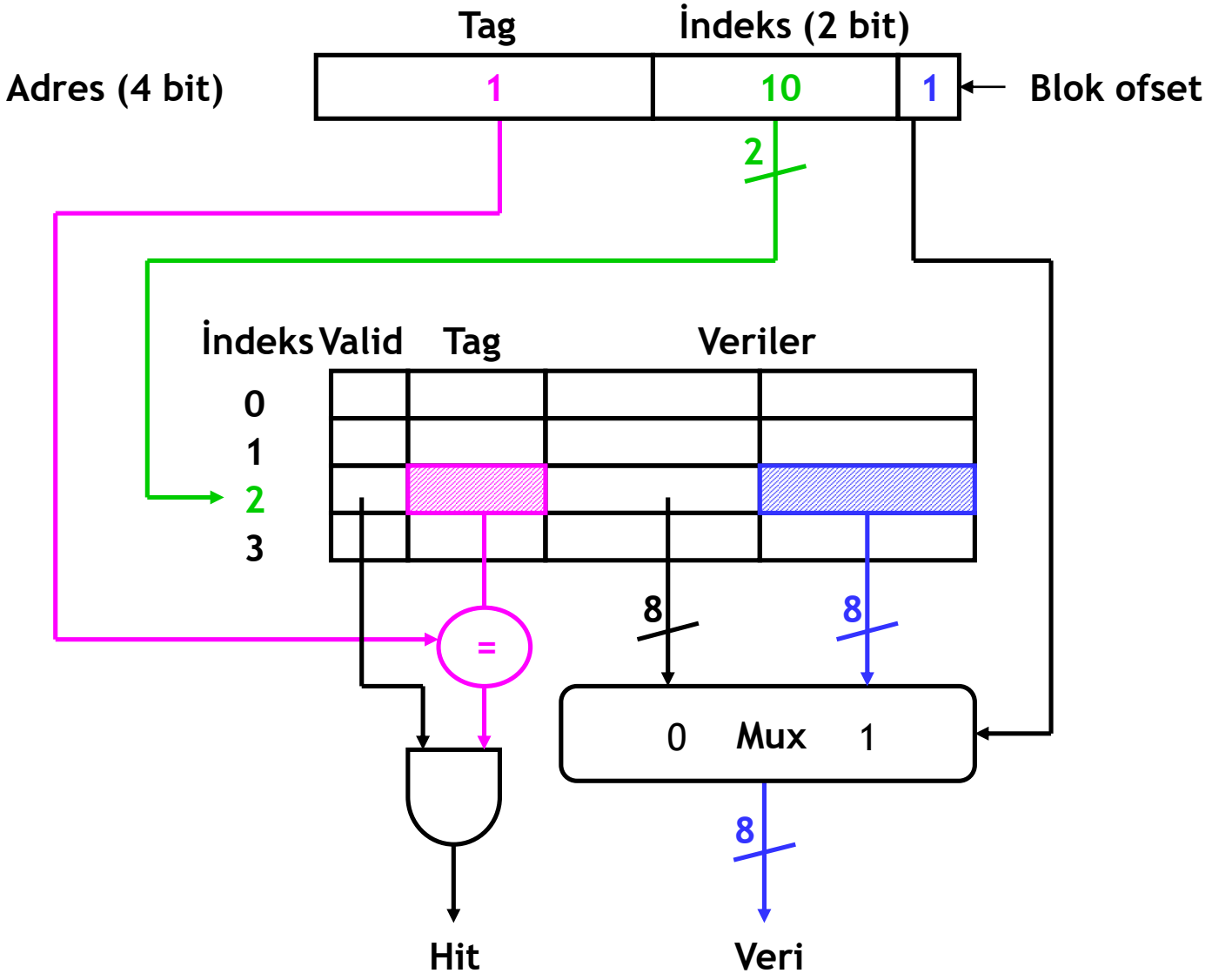


- Bizim örnekte 2^2-blok cache, her blokta 2^1 byte tutuyor. O halde, bellek adresi 13 (1101), cache blok 2 içinde (1**10**1), ofset 1'e (110**1**) yerleşir.

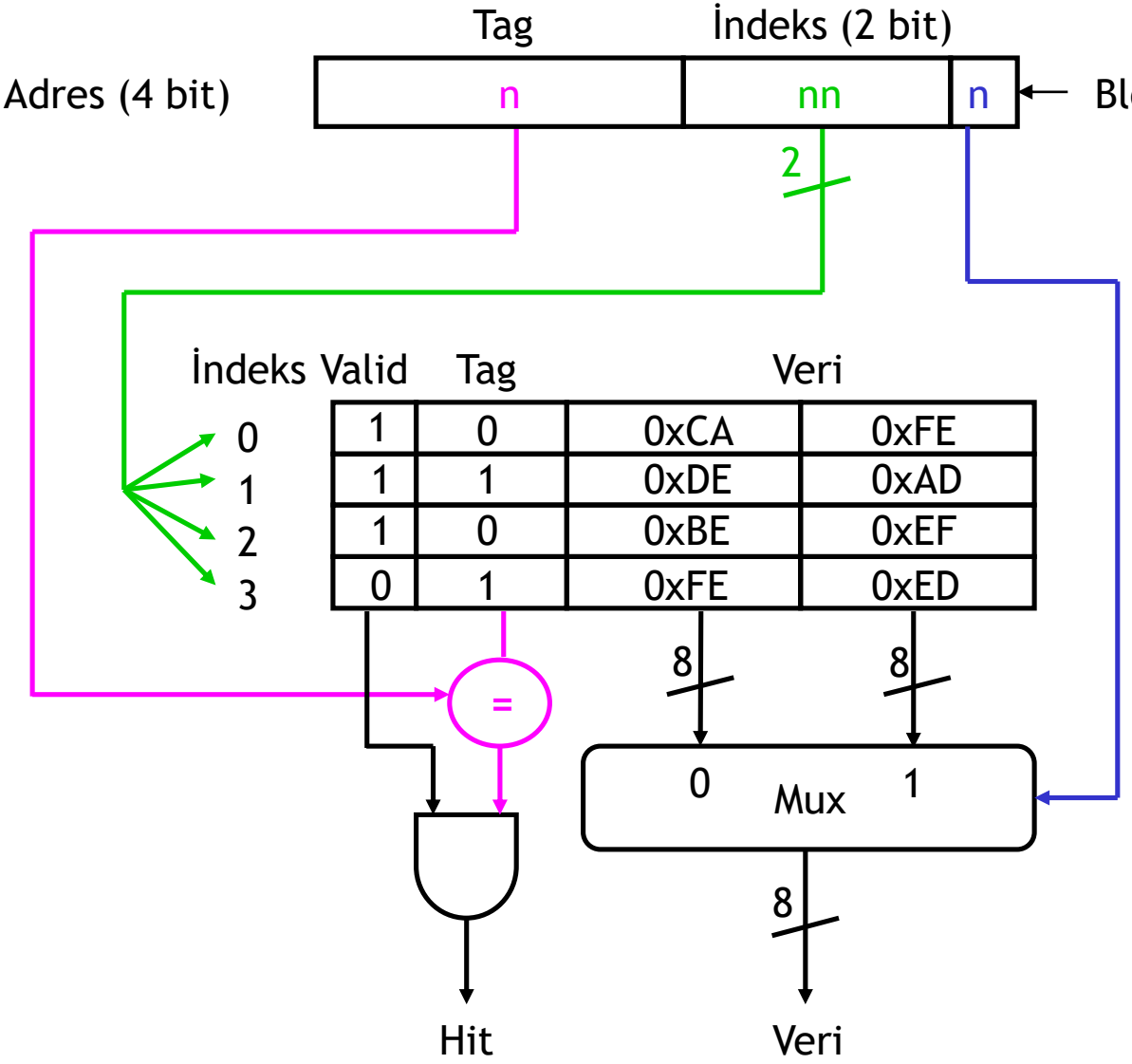


  - Adresin blok numarası 13 / 2 = 6, cache indeksi 6 mod 4 = 2.
  - Blok ofseti direkt adresten bulunabilir 13 mod 2 = 1.

13 Adresinin Yerleşimi ve Cache'den Okunma Resmi



Bir Örnek

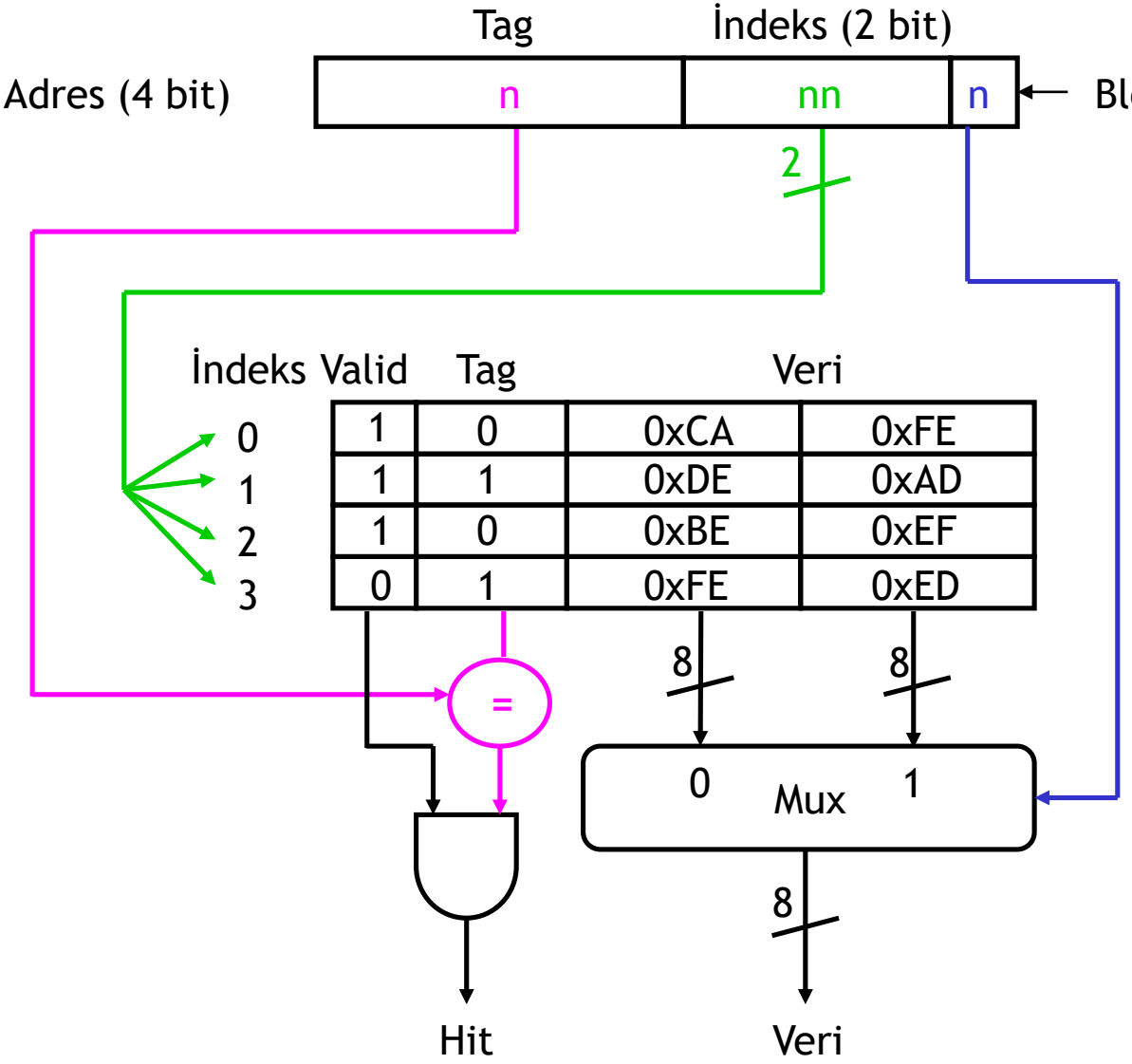


Aşağıdaki adreslere bakıp, cache'den çıkan byte'ı söyleyin (veya miss)?

- 1010
- 1110
- 0001
- 1101

... Sonraki slayta bakın

Bir Örnek

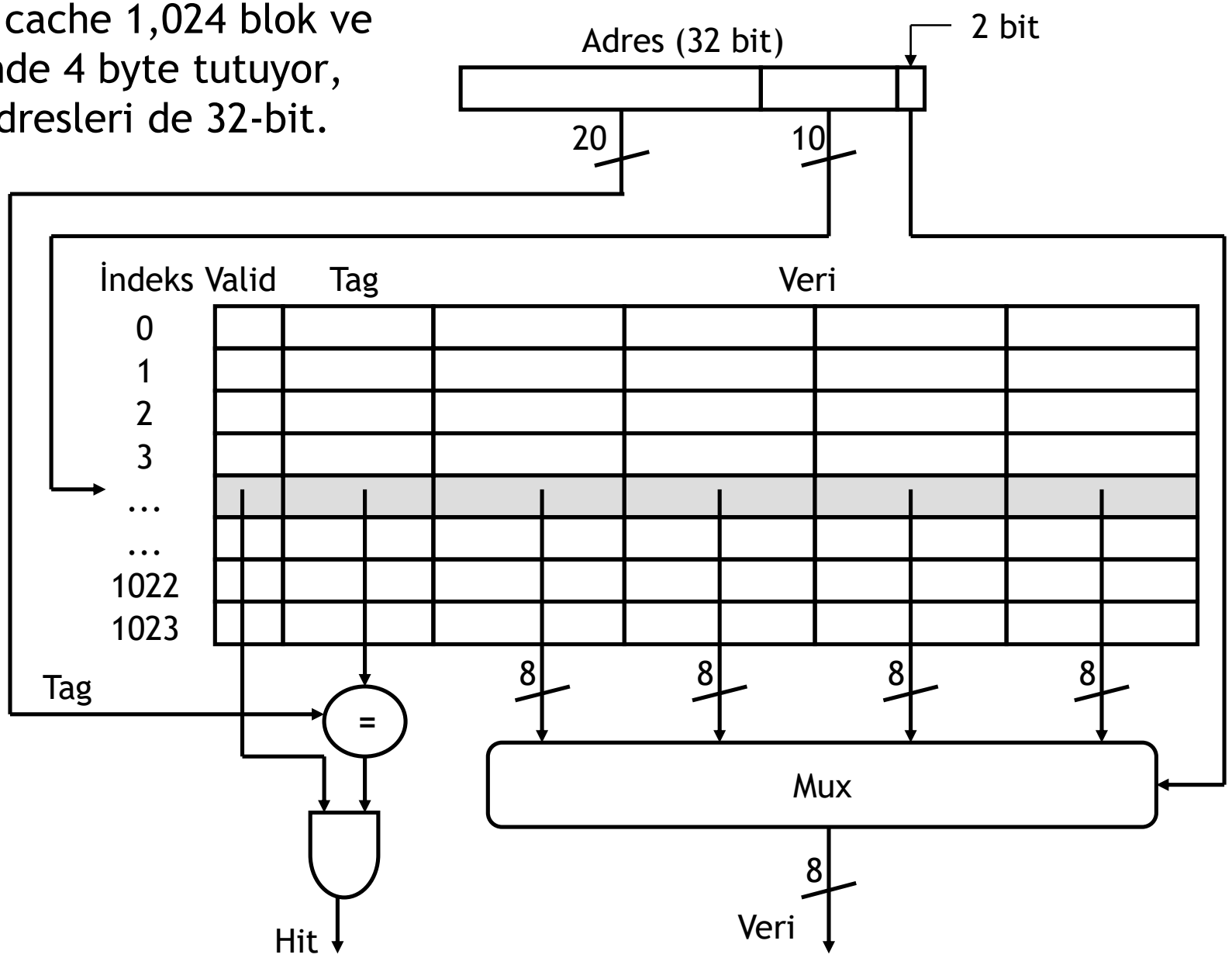


Aşağıdaki adreslere bakıp, cache'den çıkan byte'ı söyleyin (veya miss)?

- 1010 (0xDE)
- 1110 (miss, valid=0)
- 0001 (0xFE)
- 1101 (miss, tag hatalı)

Daha Büyük Bir Cache'in Örnek Diyagramı

- Yandaki cache 1,024 blok ve her birinde 4 byte tutuyor, bellek adresleri de 32-bit.

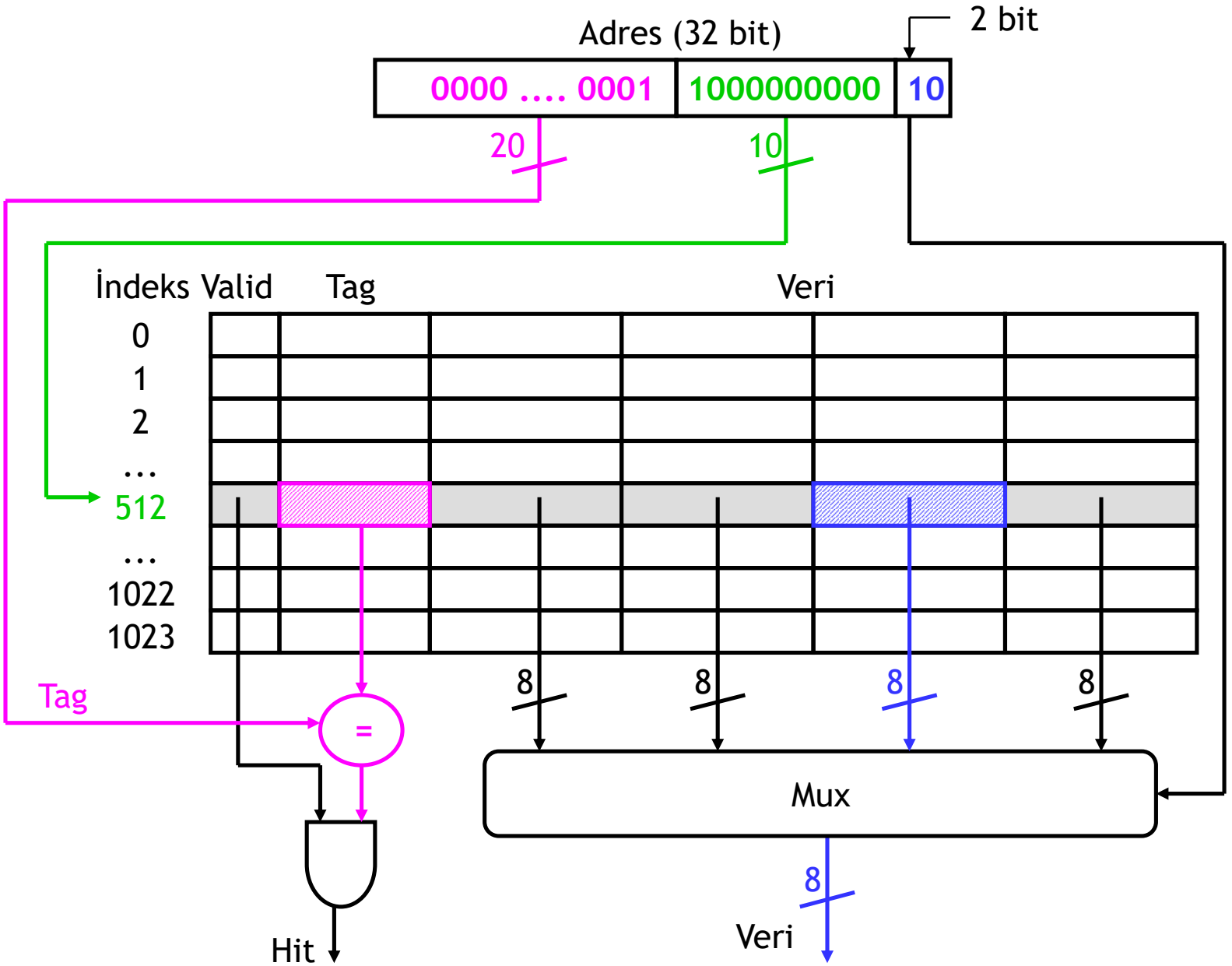


Büyük Cache'in Örnek Yerleşimleri

- Bellek adresi 6146 olan byte bize lazım, 2^2-byte uzunlukta 2^{10}-blok içeren yapısı direkt atamalı (ing: direct-mapped) cache'e veriler nasıl yerleşir?
- Hemen binary adres kolaylığından yararlanalım.
  - 6146 binary olarak **00...01 1000 0000 00 10**.
  - En sağdan 2 bit, **10**, blok içinde 2 nolu kolona yazılacak demek (sıfırdan başlandığı için üçüncü kolon oluyor).
  - Solundaki 10 bit, **1000000000**, blok numarasıdır (**512**).

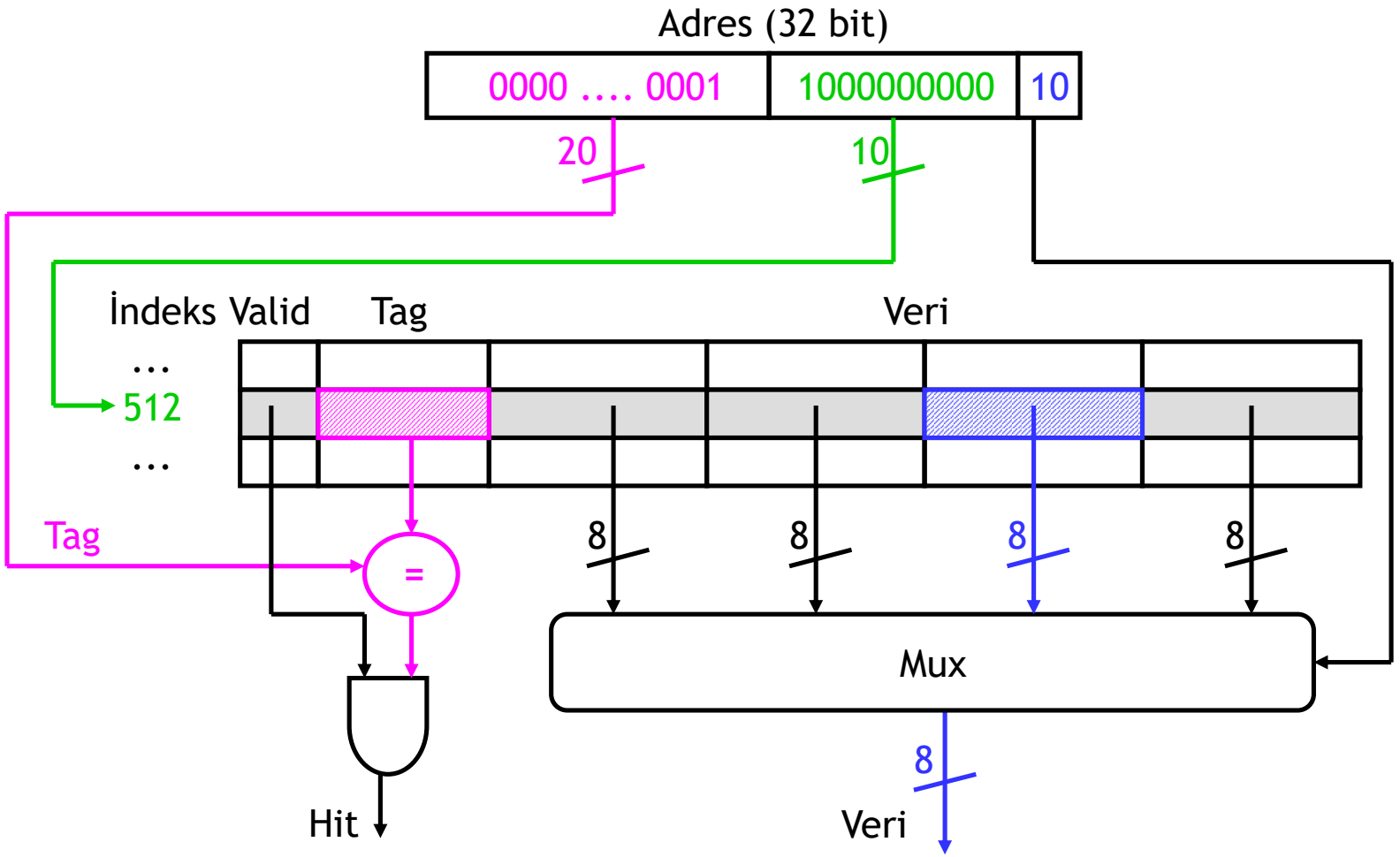
- Modulo aritmetik ile bulacağım derseniz o da olur.
  - Blok ofset = 6146 mod 4 = 2.
  - Blok adres = 6146/4 = 1536.
  - Cache indeks = 1536 mod 1024 = 512.

Büyük Cache'in Örnek Yerleşimleri (Diyagram)



Bu Cache Bloğunun Diğer Kolonlarında Neler Var?

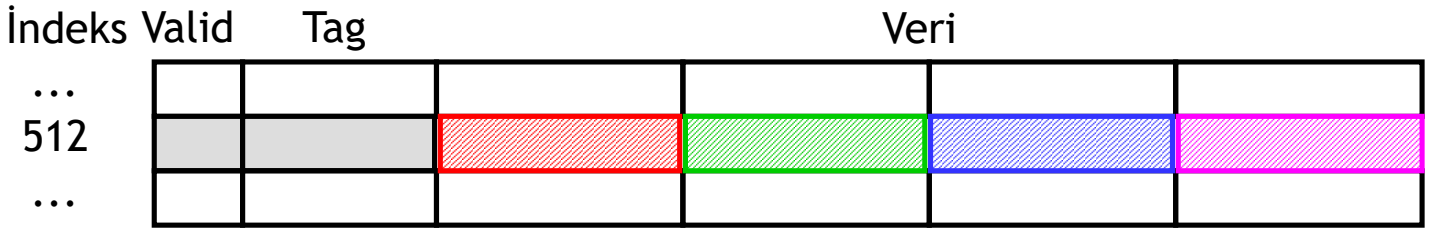
- Cache bloğunun diğer üç byte'ı aynı bellek bloğunda olanlardır
- Bahsedilen bellek bloğu; adresleri aynı indeksli (1000000000) ve aynı tag'li (0000 .... 0001) olan ofseti (00...11) arasında değişen 4 byte'tır.



Bu Cache Bloğunun Diğer Kolonlarında Neler Var?

- Cache bloğumuzun içine bir bakalım.
  - 1536 nolu bellek bloğu adres 6144 .. 6147 arasını tutar.
  - Cache bloğundaki 0-3 arasında numaralandırılan kolonlardaki veriler adres 6144, 6145, 6146 ve 6147 olacaktır.

Blok ofset	Bellek adresi	Ondalık adres
00	00..01 1000000000 00	6144
01	00..01 1000000000 01	6145
10	00..01 1000000000 10	6146
11	00..01 1000000000 11	6147



Bir Cache Ne Kadar Büyür Yeni Örnek?

- Cache kullanılan 32-bit adresli ve byte-byte adreslenebilen bir makinede şu karakteristik özellikler vardır :
  - Direkt atamalıdır
  - Her blok bir 32 byte tutar
  - Cache satırları 1024 adettir.

- Bu cache yapısı için kaç byte kullanılır (*veri*, tag, valid bitleri toplamı)?

tag boyutu = 17 bit (32 bit adres - 10 bit indeks - 5 bit ofset)

(17 tag biti + 1 valid biti + 8x32 veri biti) x 1024 blok

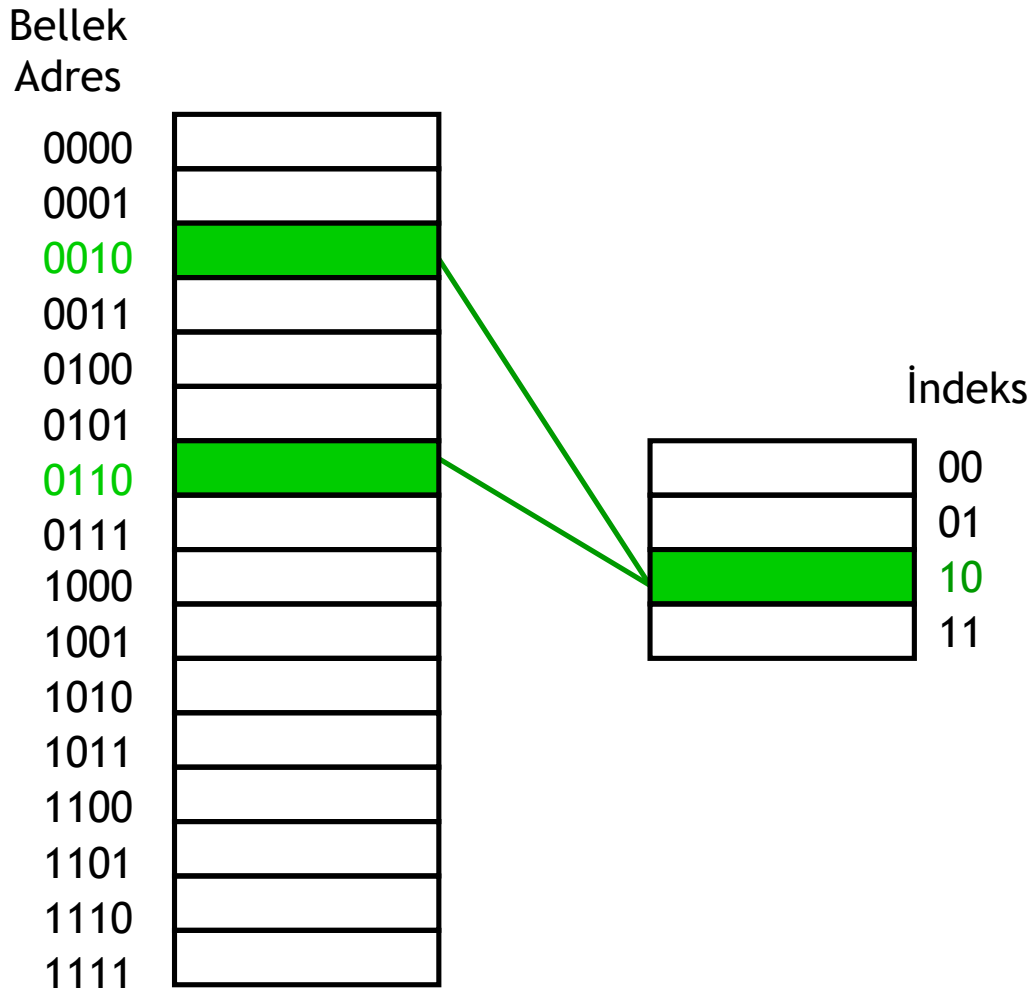
= 274 bit x 1K = 274 Kilobit = 34,25 KB

Direkt Atamanın Dezavantajları

- Bir direkt atamalı cache yapmak çok kolay: indeksler ve ofsetler adresin bitlerinden hesaplanabiliyor, çünkü her bellek adresi bir bloğa ait.
- Fakat aslında çok sıkıntılı bir yapıdır.
- İlk örnek cache'de adresler şöyle olsa

  2, 6, 2, 6, 2, …?

- Her adres cache miss üretir ve sürekli ana bellekten veri alınır.
- Demek ki tahminimizden daha kötü senaryolar var.
- Bir satırda birden fazla veri tutmak da kurtarmayabilir.



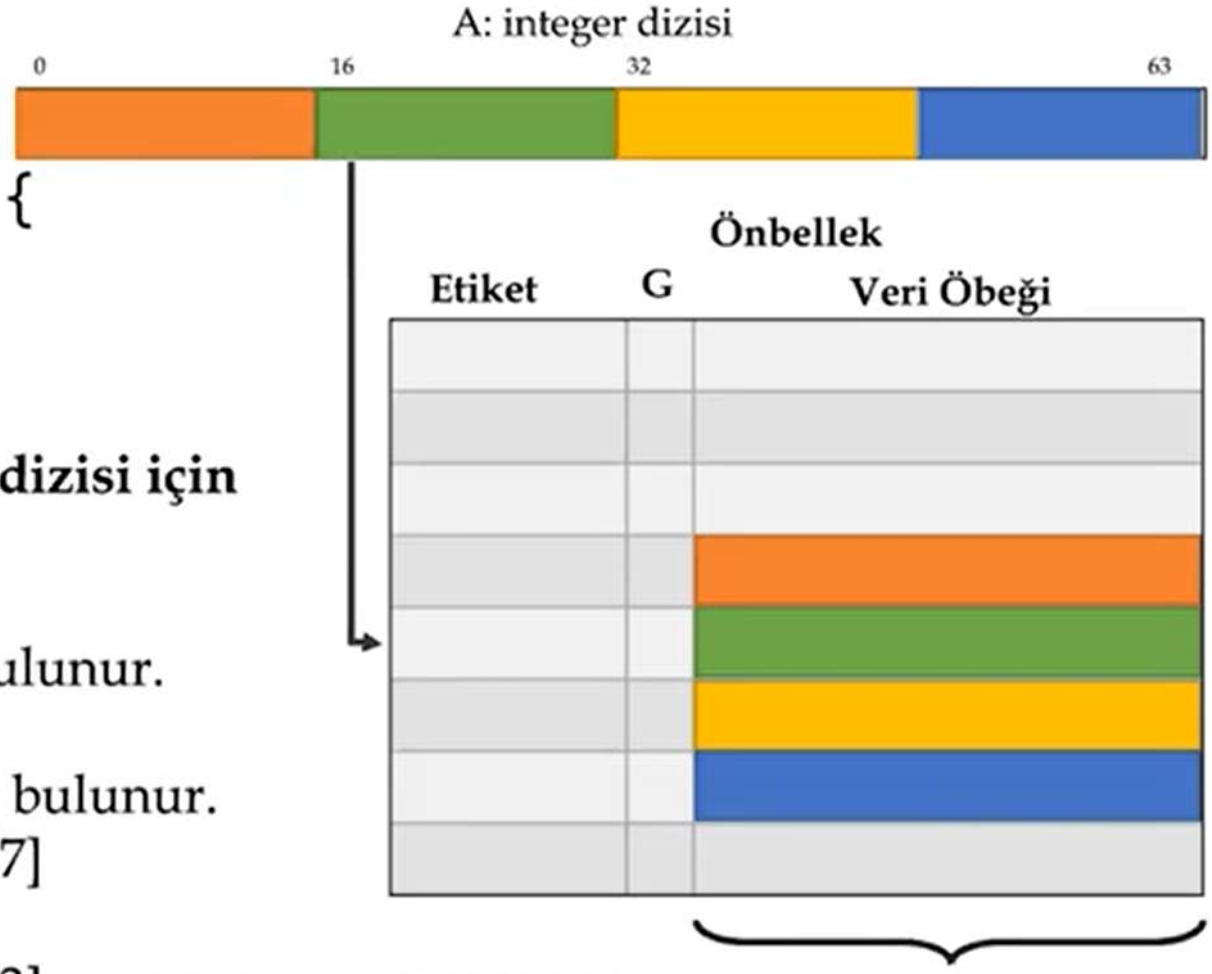
Ödev/Sınav Sorularından Bir Örnek

Alanda Yerellik

```
for (int i =0;i<64;i++) {
      B[i] = A[i] * 2;
}
```

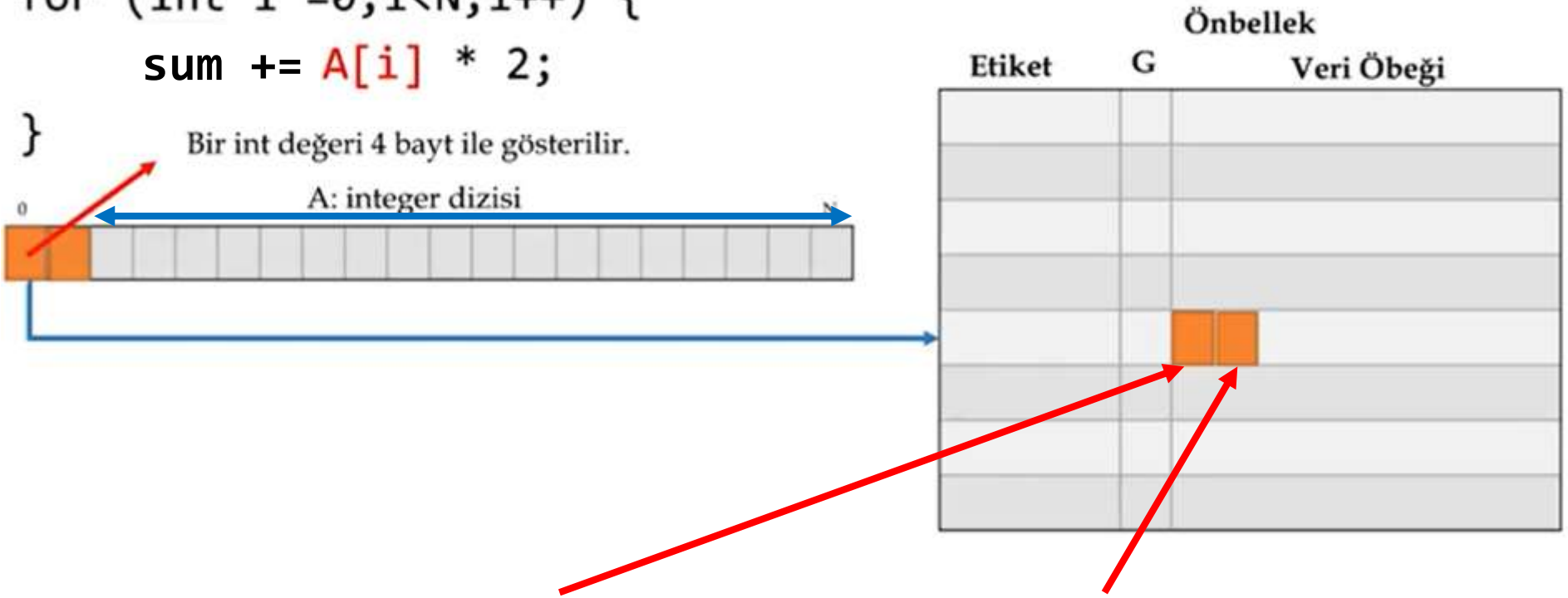
Veri öbeği 64 byte ve N = 64 ise A dizisi için önbellekte bulma oranı kaçtır?

A[0] bulunamaz, getirilir.
A[0…15] getirildiği için A[1…15] bulunur.
A[16] bulunamaz, getirilir.
A[16…31] getirildiği için A[17…31] bulunur.
A[32] bulunamaz, getirilir. A[32…47] getirildiği için A[33,47] bulunur.
A[48] bulunamaz, getirilir. A[48…63] getirildiği için A[49…63] bulunur.



Zamanda Yerellik Örneği

```
int sum = 0
for (int i =0;i<N;i++) {
      sum += A[i] * 2;
}
```



- sum = 0 dendiğinde, **sum** ilk boyalı kısma yerleşir, **i** de yanına yerleşir.
- Gerisi A dizisinin 13 elemanı veya başka veriler olabilir, kesin bir şey yok.
- sum ve i için cache içinde bulunma (hit) oranı nedir?
  - sum bir kez bulunamadığı için getirilir, for'da hem okunup, hem güncellenir, ama hep hit alır: 2N/(2N+1)
  - i ise, sum'ın yanında geldi, i için hiç miss olmuyor hit oranı %100 !

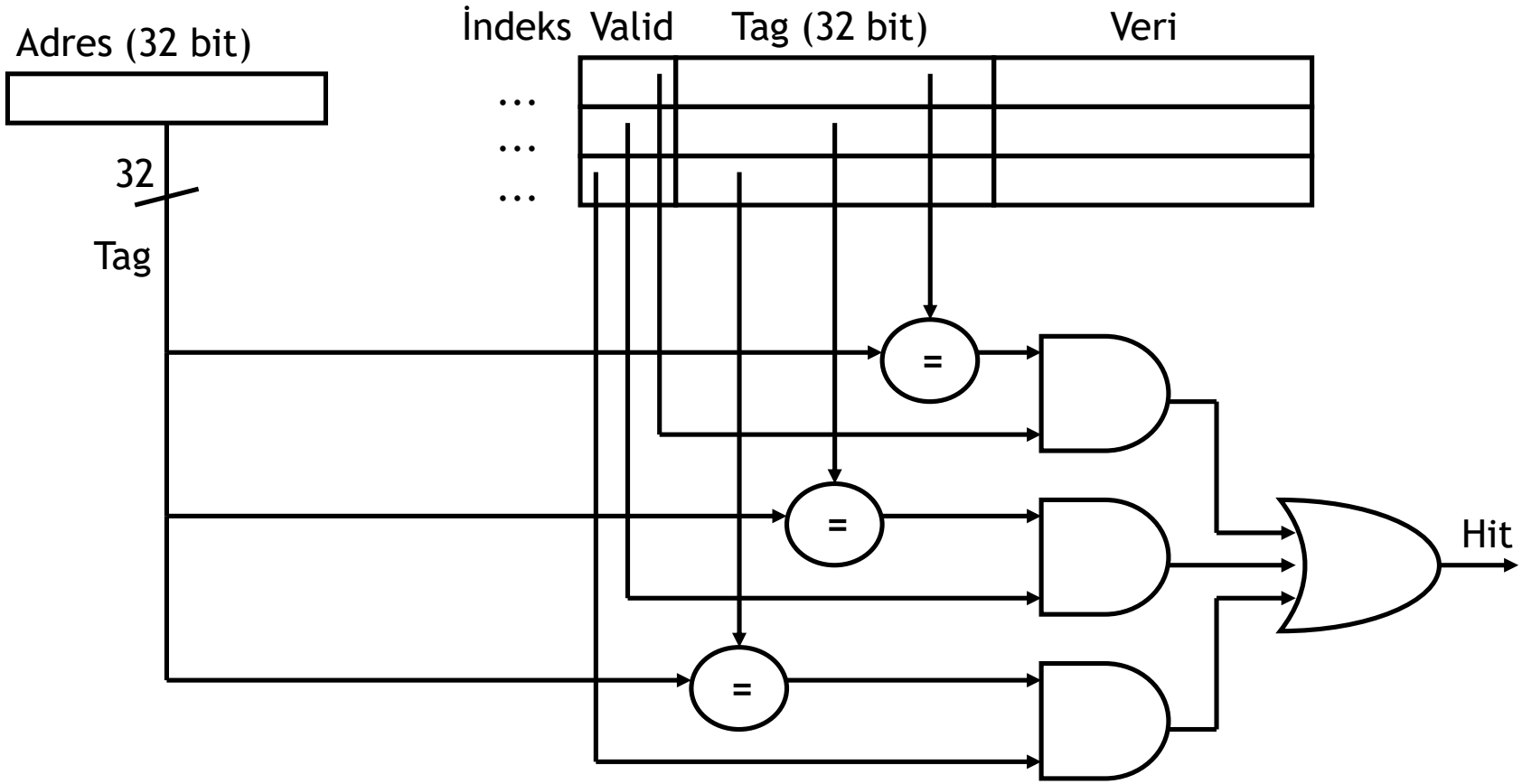
BİRLİKSEL (ASSOCIATIVE) CACHE'LER

Bir Tam Birliksel Cache

- **Bir tam birliksel cache (ing: fully associative cache)** verinin cache içinde herhangi bir bloğa yazılabilmesi demektir.
- Bu cache bellek adreslerini belli alanlara zorlamaz.
  - Bellekten veri alındığında, cache'de kullanılmayan blok varsa oraya yazılır.
  - Eğer tüm alanlar dolu ise, uzun süredir erişilmeyen (ing: least recently used) bir bloğun üzerine yazılır.
  - Aynı cache bloğuna atanan veriler diye bir şey kalmadığından veri çakışması hiç yaşanmaz.
- Önceki örnekte bellek adres 2, cache blok 2'ye yerleşirse, adres 6 artık onun üzerine gelmiyor ve belki cache blok 3'e yazılıyor. Artık tüm 2 ve 6 bellek adresi erişimleri cache hit veriyor, cache miss değil.

Bu Cache'lerin Maliyeti?

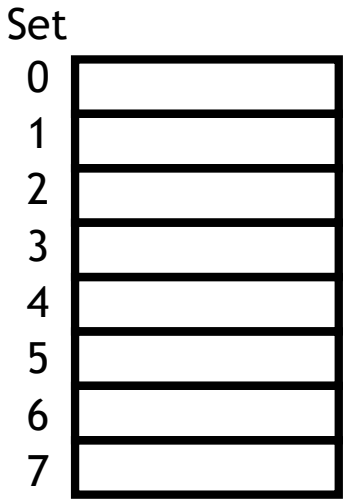
- Maalesef çok küçük satır içeren TLB gibi cache'ler haricinde bu cache'ler kullanmak için çok pahalılar.
  - Çünkü burada indeks artık kullanılmıyor ve bu durumda adresin tamamı tag alanı oluyor. Bu da cache'in toplam boyutunu arttırıyor.
  - Bu cache içinde veri herhangi bir blokta olabilir, o halde her cache bloğu gelen tag değerini kontrol etmeli yani çok sayıda karşılaştırma devresi gereklidir.



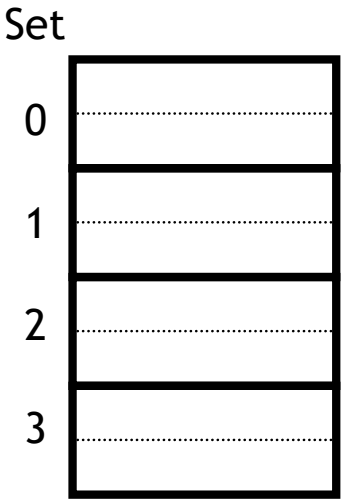
Çözüm: Set Birliktelikleri

- Ara bir çözüm olarak **set-associative cache**'ler vardır.
  - Cache **set** ismi verilen bloklara ayrılmıştır.
  - Her bellek adresi bu sefer cache içinde belli setlere yönlendirilir, ama veri bu set içinde herhangi bir satırda/blokta olabilir.
- Eğer her set 2^x blok olursa, cache'imizin ismi **2^x-way associative cache** olur.
- İşte 8 bloklu cache için bazı olası dizaynlar.

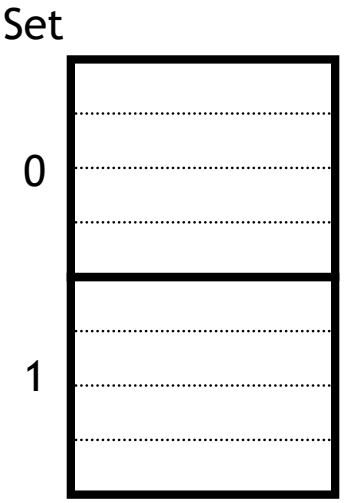
1-way associativity
8 set, her biri 1 blok



2-way associativity
4 set, her biri 2 blok

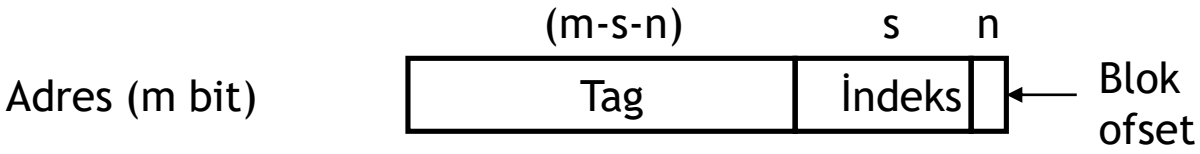


4-way associativity
2 set, her biri 4 blok



Setlerde Yerleşimler

- Bellek adresinin ait olduğu seti hesaplarken daha önceki hesaplama yöntemlerinden pek uzaklaşmıyoruz.
- Eğer cache 2^s set ise ve her blok 2^n byte tutuyorsa, bellek adresi şöyle bölünebilir.



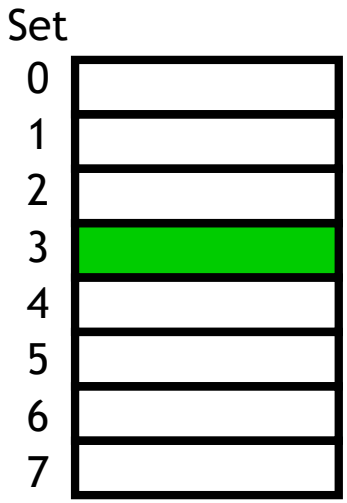
- Artık tek tek blokların indeksini değil, burada set indeksini buluyoruz. Tek satırlı blok yerine çok satırlı set var.

Blok Ofset = Bellek Adres mod 2^n
Blok Adres = Bellek Adres / 2^n
Set İndeks = Blok Adres mod 2^s

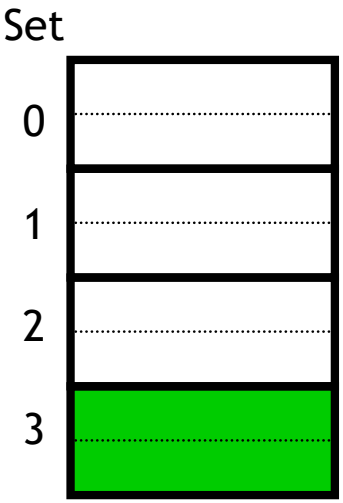
Bir Set-Associative Cache'de Örnek Yerleşim

- Yine bellek adresi 6195'den veri gelsin, 8-bloklu cache dizaynlarımız, her blokta 16 byte tutuyorlar, yerleşim nasıl olur?
- 6195 binary olarak 00...0110000 011 0011.
- Her blok 16 byte ise, sağdan 4 bit blok ofsetini verir.
- 1-way cache için, sola doğru 3 bit (011) set indeksidir.
  2-way cache için, sola doğru 2 bit (11) set indeksidir.
  4-way cache için, sola doğru 1 bit (1) set indeksidir.
- Verimiz yeşille gösterilen set içinde herhangi bir blokta.

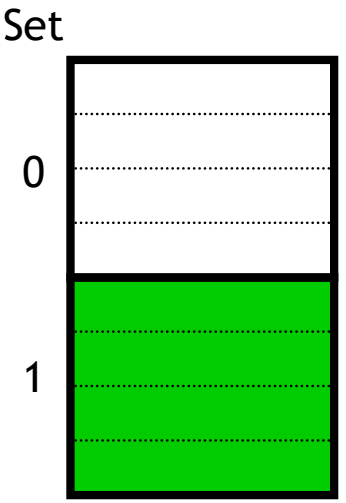
1-way associativity
8 set, her biri 1 blok



2-way associativity
4 set, her biri 2 blok

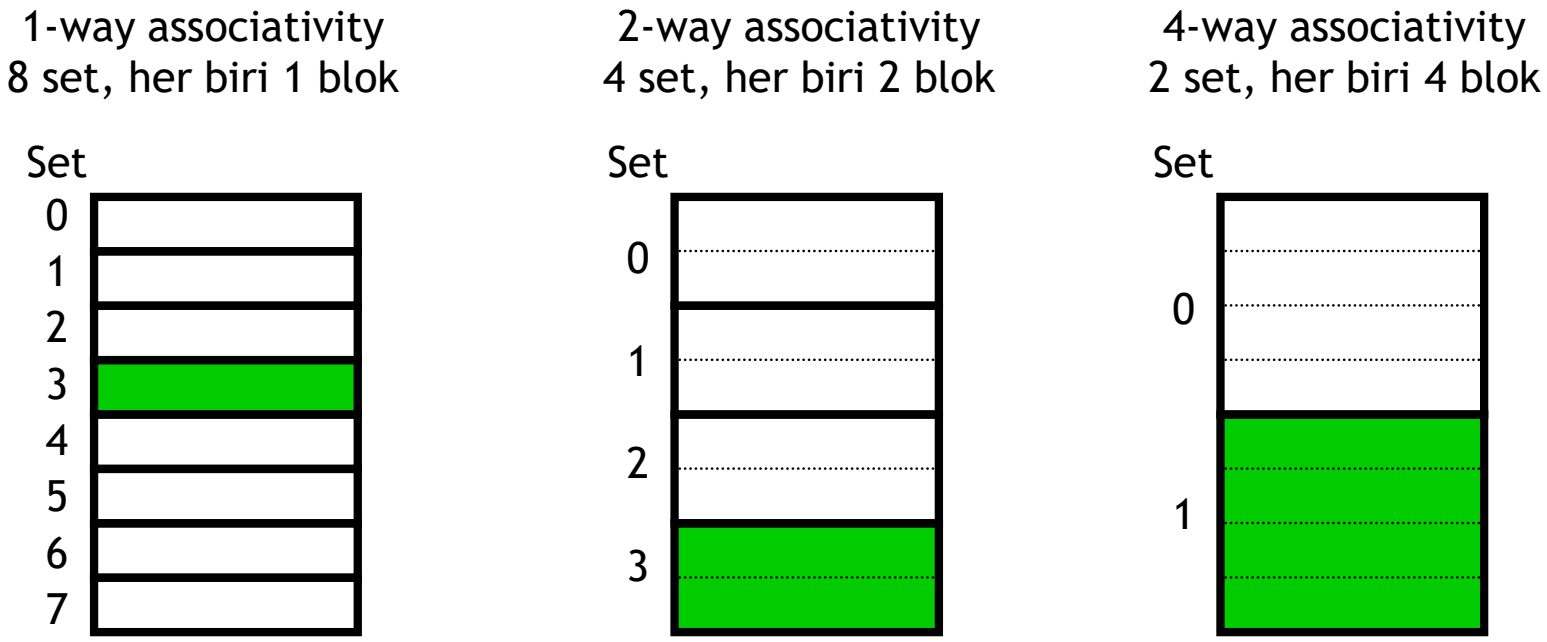


4-way associativity
2 set, her biri 4 blok



Blok Yerleşimi

- Doğru sette boş bir blok verimizi yazmak için kullanılır.
- Hiç boş blok yoksa, cache kontrolcüsü en az kullanılan (ing: least recently used) bloğun üzerine yazar.
- Set satır/blok sayısı yüksek olan associative cache'lerde, kim en az kullanılmış bulmak oldukça pahalı bir iştir, bazı yaklaşık sonuç veren çözümler var ama detaya girmeyeceğiz.

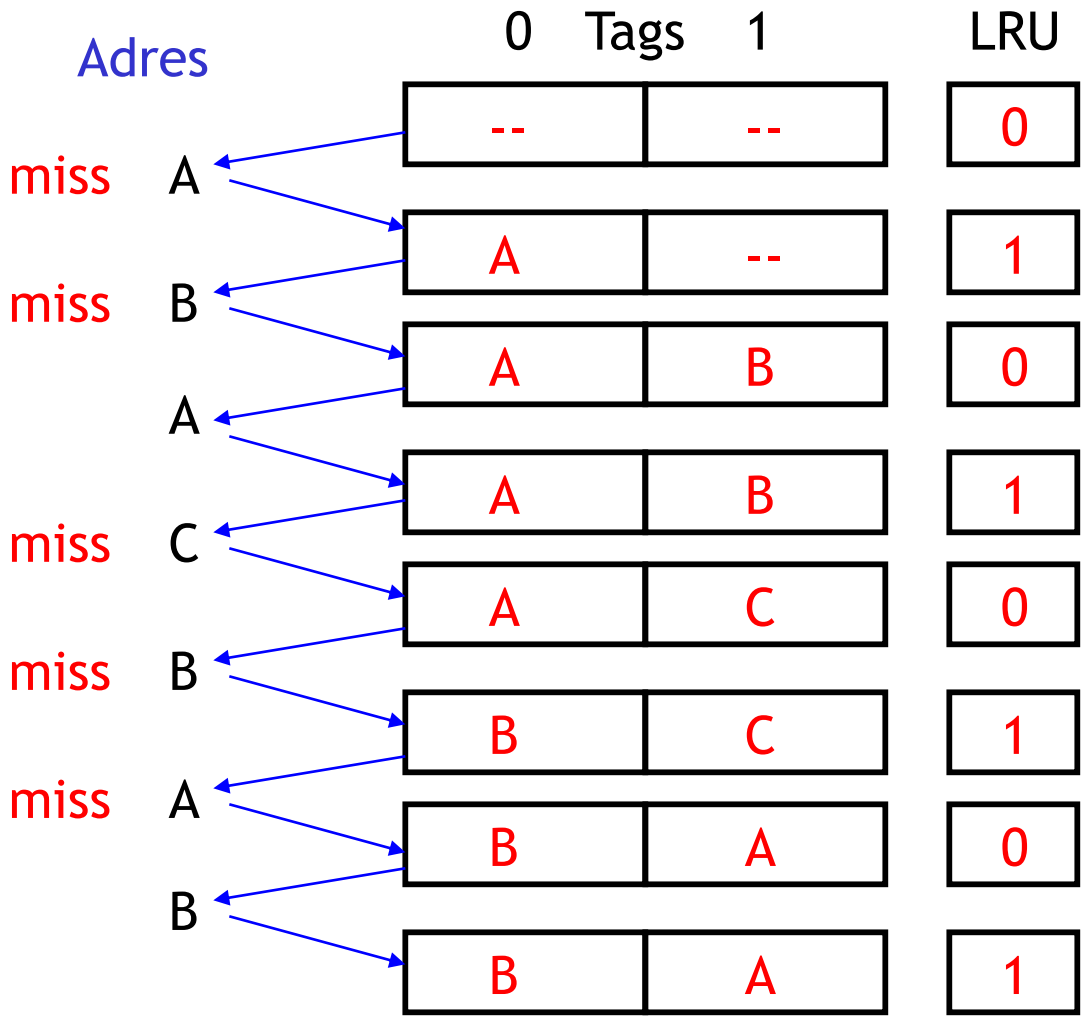


En Az Kullanılan (LRU) Örneği

- Bir ‘fully-associative cache’ iki blok tutsun, aşağıdaki bellek adresleri cache miss döndürür.
  - Bloklar yana doğru dizilidir.

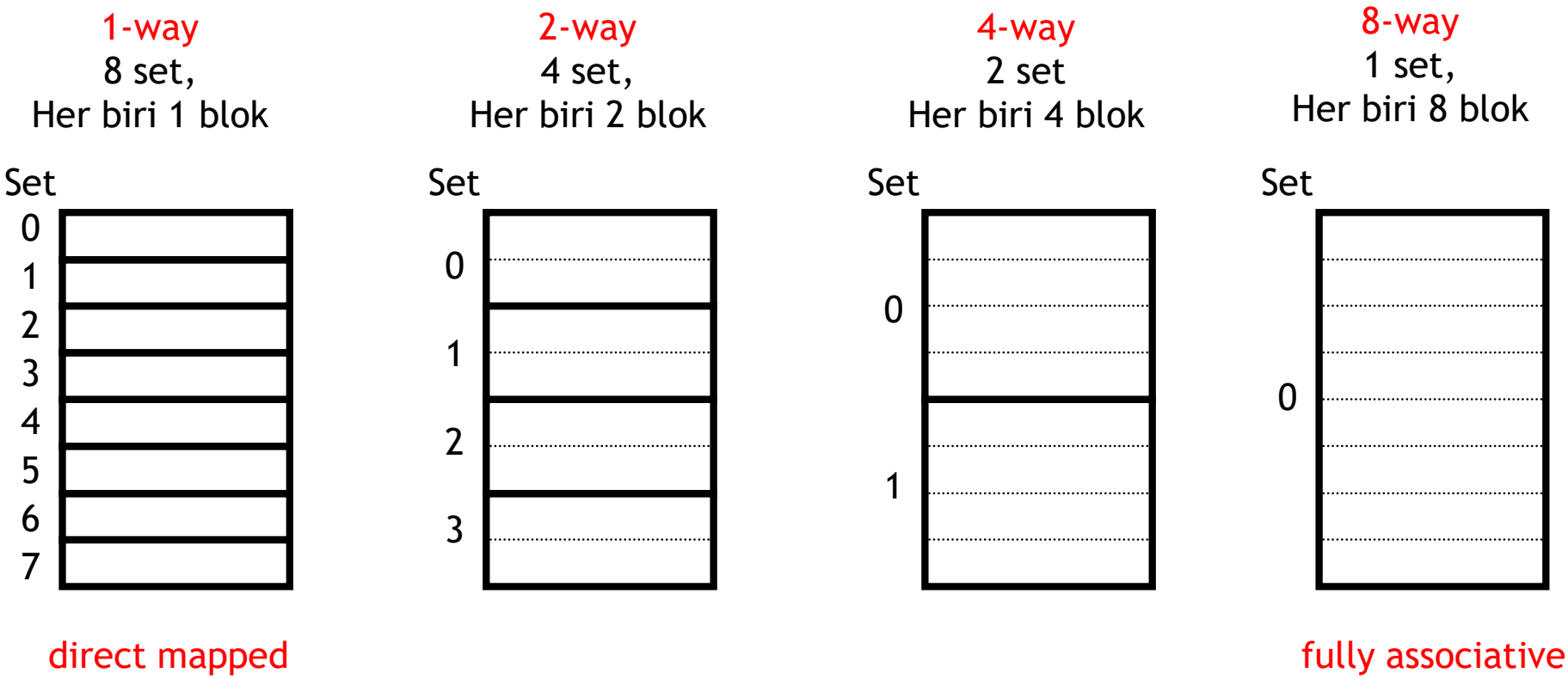
Cache miss varsa, LRU ile yerleşim,

cache hit varsa, LRU düzeltme yapılır.



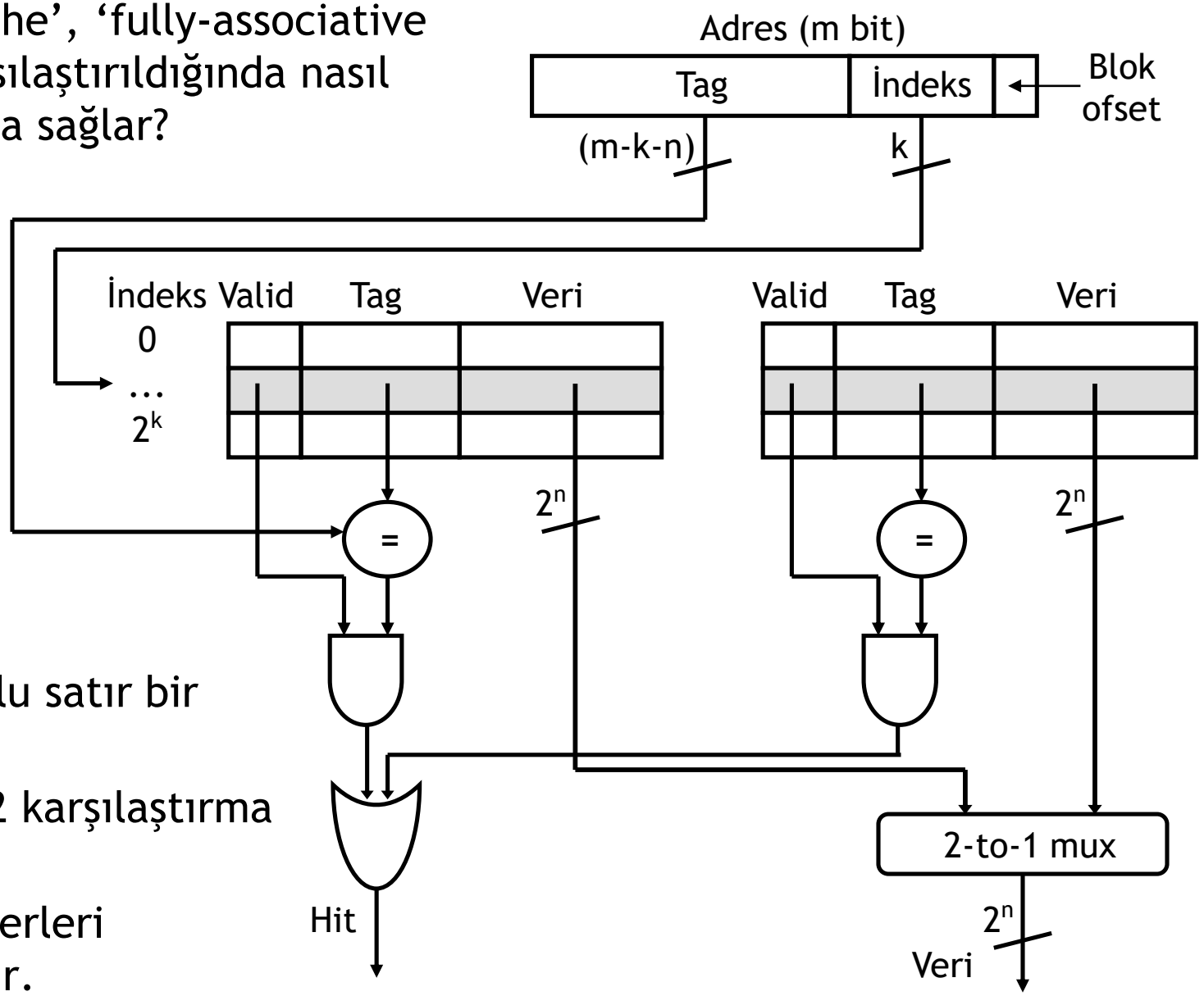
Set Associative Cache'ler Her Amaca Uygundur

- 1-way set associative cache = direct-mapped cache aslında.
- Benzer şekilde, eğer bir cache 2^k blok içeriyorsa, 2^k-way set associative cache = fully-associative cache aslında.



2-way Set Associative Cache Uygulaması

- Bir ‘2-way cache’, ‘fully-associative cache’ ile karşılaştırıldığında nasıl kolay uygulama sağlar?



- Boyalı 2 kolonlu satır bir settir.
- Sette sadece 2 karşılaştırma yeterlidir.
- Cache tag değerleri de daha kısadır.

Özet

- Büyük blok boyutları uzaysal yerellik (ing: spatial locality) sağlar. Burada bir adresten veri almak yerine, cache içine komşu adresler de getirilir.
- Associative cache'ler her bellek adresini cache içinde bir sete atar, fakat set içinde hangi bloğa yazılacak belli değildir.
  - Set boyutları 1'den (direct-mapped), 2^k 'ya (fully associative) değişir.
  - Büyük setler ve yüksek satır sayıları (ing: associativity) daha az cache çakışması ve düşük cache miss oranları demek, fakat donanımsal maliyet çok yüksek.
  - Pratikte, 2-way'den 16-way set-associative cache'lere kadar uygulamalar kullanılırken düşük miss oranları yüksek maliyet arasında bir denge kurulur.
- Devam derste, cache performansı üzerine daha çok konuşacağız ve cache'e veri yazma yöntemlerini tartışacağız.