Kamer Kaya

Sabancı Üniversitesi

Yüksek Başarımlı (Paralel) **Hesaplama**

Veri Bilim - Yapay Öğrenme Yaz Okulu, 2017
"Matematiksel Temeller ve Vaka Çalışmaları"

Yüksek **başarım**?

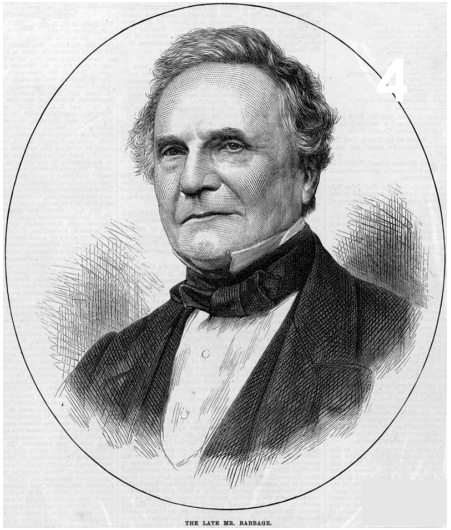
- 1. (isim) Elde edilen bir başarı
- 2. Herhangi bir olayı veya durumu başarma isteği ve gücü
- 3. Kişinin yapabileceği **en iyi** derece, performans
- 4. Herhangi bir eseri, oyunu, işi vb.ni ortaya koyarken gösterilen başarı, performans

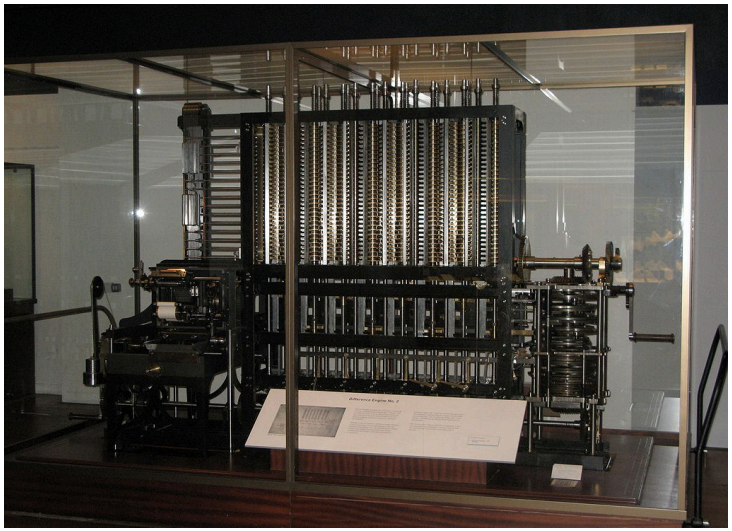
1800ler

"Computer" kelimesinin ilk defa New York Times gazetesinde gözüktüğü tarih: May 2, 1892; US Civil Service Commission tarafından verilen bir ilan:

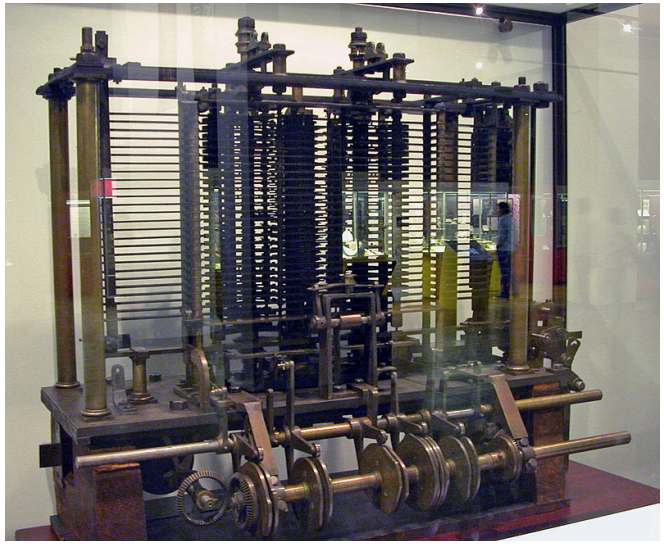
"A Computer Wanted. [...] The examination will include the subjects of algebra, geometry, trigonometry, and astronomy."

1800ler
Charles Babbage





The Science Museum Difference Engine No. 2, Babbage'ın birebir tasarımı



Analytical Engine prototipi, The Science Museum

1930lar
Alan Turing

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

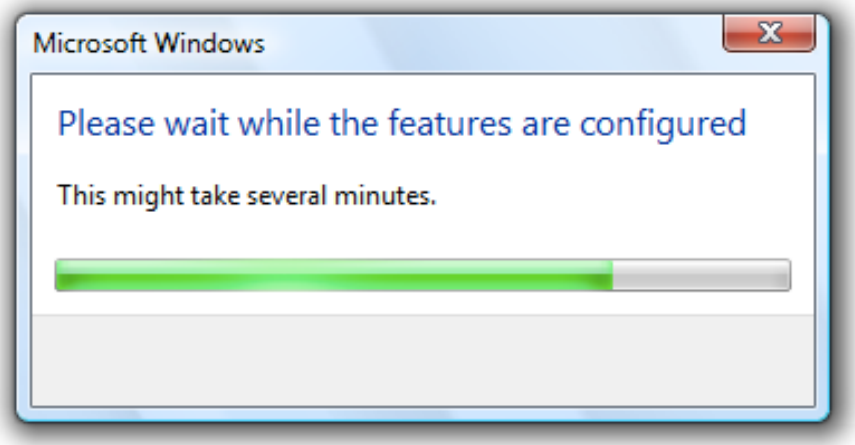
By A. M. Turing.

[Received 28 May, 1936.—Read 12 November, 1936.]

1936

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

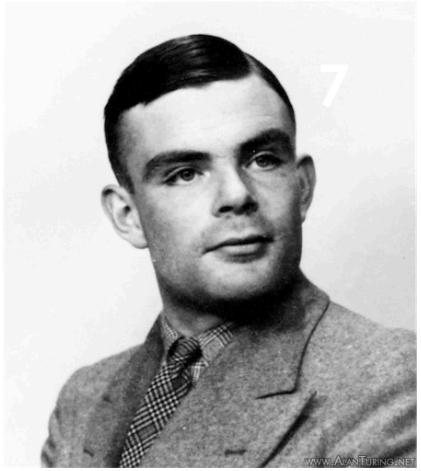
1930lar
Alan Turing

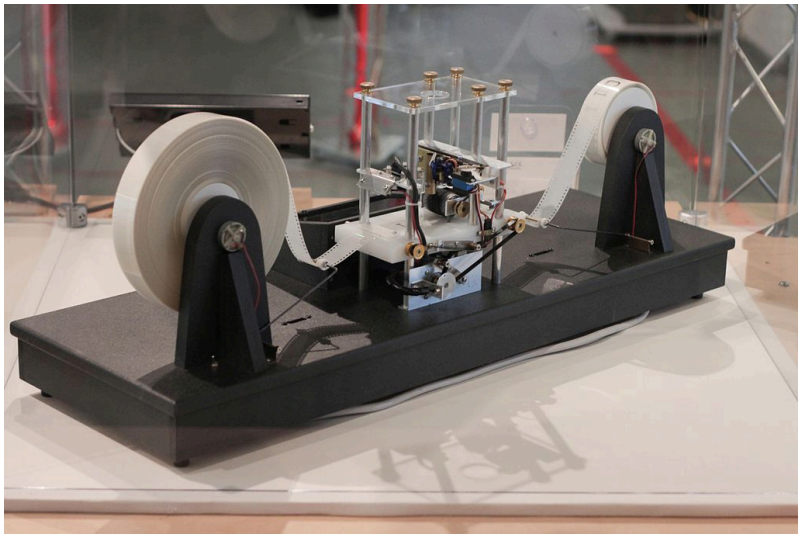


Hesaplanabilirlik teorisinde, Durma problemi, keyfi bir bilgisayar programının ve girdinin açıklamasından, programın koşmayı bitirip bitirmeyeceğine ya da sonsuza dek koşmaya devam edip etmeyeceğine karar verme problemidir.

Büyük olasılıkla durmayacak bir program

1930lar
Alan Turing

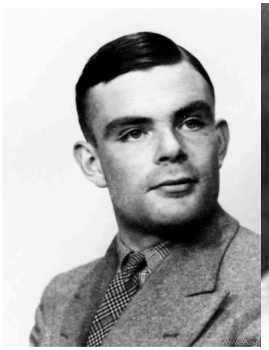




Bir Turing makinesi

Hesaplanabilirlik teorisinde, bir veri-manipülasyon kuralları sistemi (bir bilgisayarın komut kümesi, bir programlama dili veya hücresel bir otomat gibi) tek bir bantlı Turing makinesini simüle etmek için kullanılabiliyorsa, Turing tam veya hesaplamalı evrensel olduğu söylenir .

1939-1945
II. Dünya Savaşı


Turing

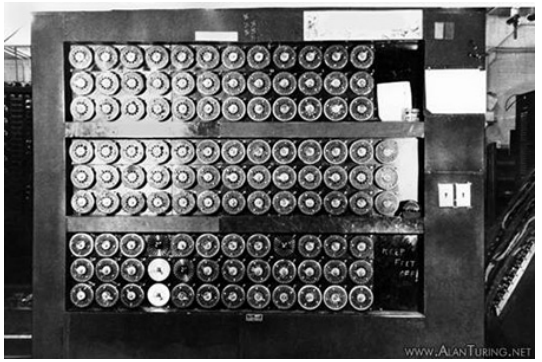

Welchman


Flowers

İYİ ADAMLAR

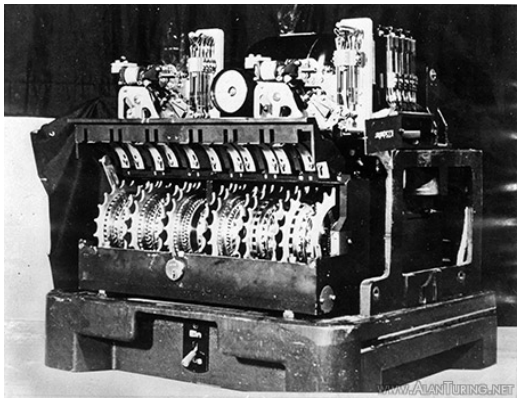
Enigma

vs.



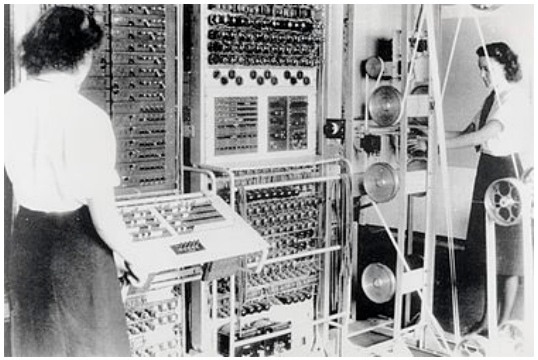
Bombe

KÖTÜ ADAMLAR



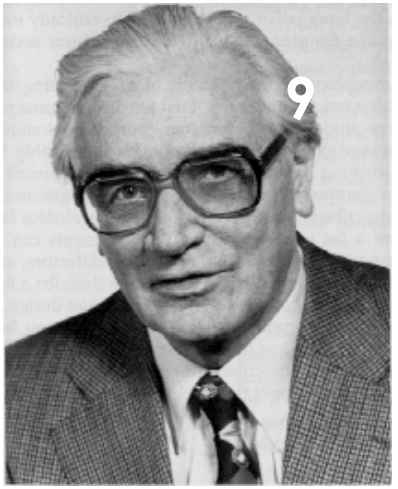
A Lorenz Şifreleme Makinası

vs.

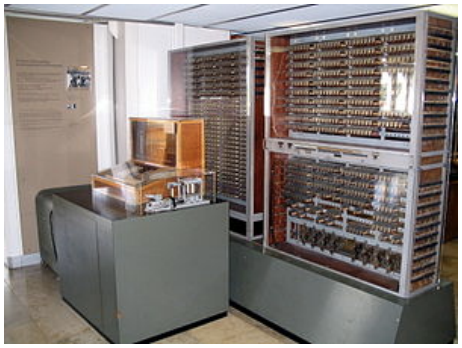

Colossus

1936-38 Z1 1939-40 Z2 1941... Z3...

(TURING TAM)



Konrad Zuse



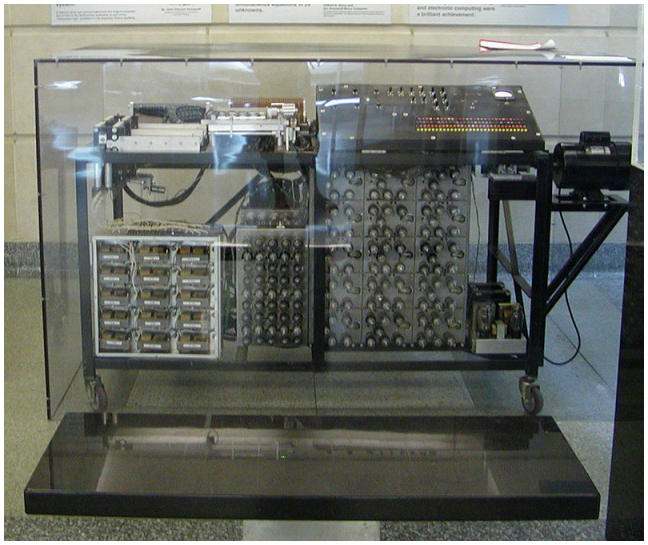
Zuse Z3 replikası
Deutsches Museum, Munich

Ortalama hesaplama hızı: ek - 0.8 saniye, çarpma - 3 saniye
Aritmetik birim: İkili kayan noktalı, 22 bit, ekle, çıkart, çarp, böl, karekök
Veri belleği: 22 bit uzunluğa sahip 64 sözcük
Program hafızası: Delinmiş selüloit bant
Girdi: Ondalık kayan nokta sayıları
Frekans: 5.3 Hertz
Güç tüketimi: Yaklaşık 4.000 watt
Ağırlık: Yaklaşık 1 ton (2,200 lb)

1939-42 Atanasoff-Berry Bilgisayarı(ABC)

Atanasoff

Berry



Atanasoff–Berry bilgisayarı Durham Center, Iowa State University

- **Üç önemli fikir:**
- Tüm sayıları ve verileri temsil etmek için ikili basamakları kullanma
- Tüm hesaplamaları, tekerlekler, mandallar veya mekanik anahtarlar yerine elektronik kullanarak gerçekleştirmek
- Hesaplama ve hafızanın ayrıldığı bir sistemi kullanmak.

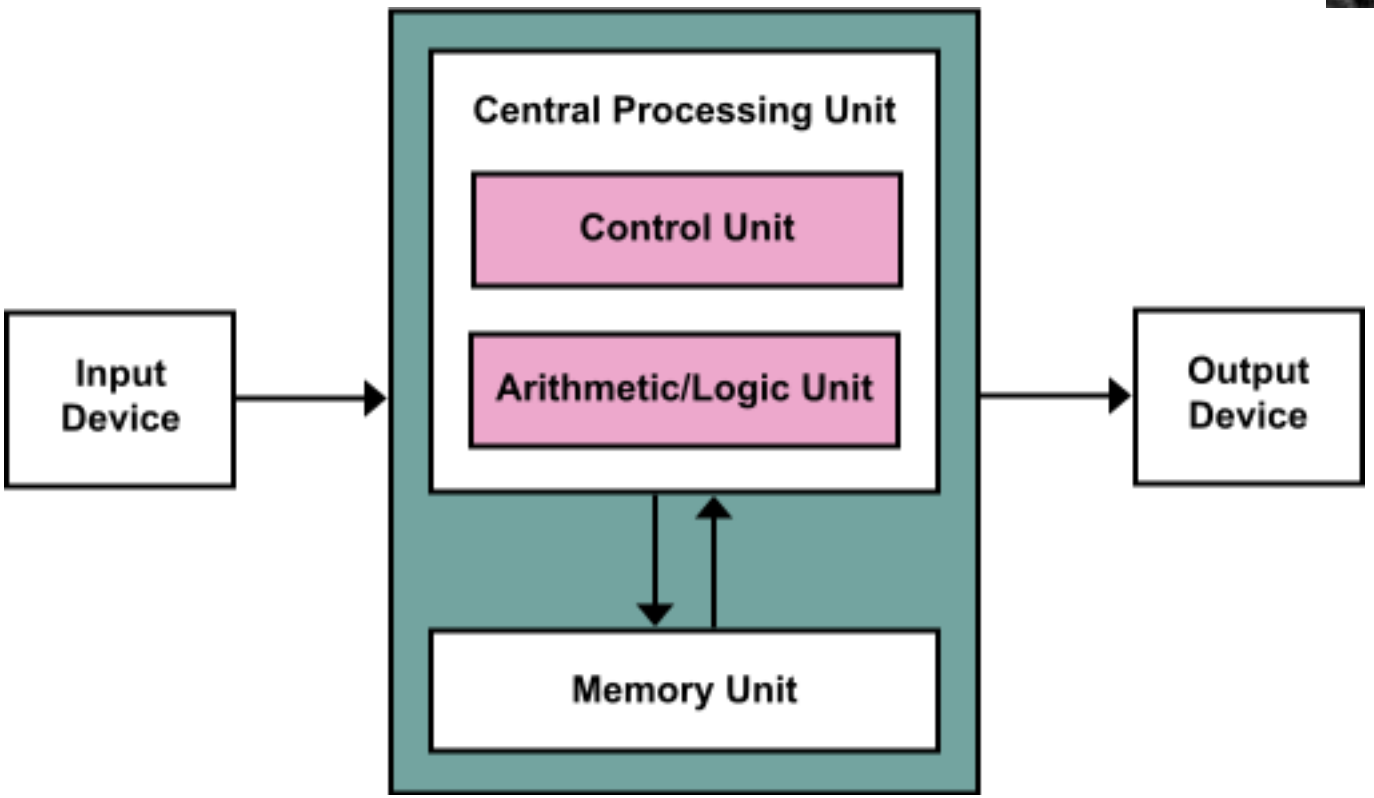
1945-1960s

ENIAC (1945)

IBM 5Mb HD (1956)

1945
Von Neumann Mimarisi

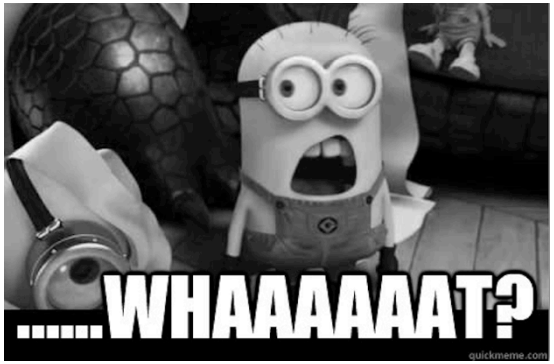
John von Neumann



o zamanlar geleceği tahmin etmek oldukça güçtü

"Sanırım bütün dünyada sadece 5 bilgisayarlık bir pazar olacak''

IBM başkanı, 1943



ama bazıları dğour tahmin edebilmişti

"Gelecekte bilgisayarların ağırlığı en fazla

1.5 ton

olacak."

Popular Mechanics, 1949

1965
Moore's Law

Gordon Moore

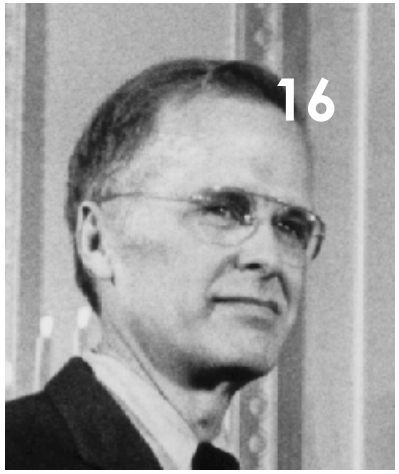
Cramming More Components onto Integrated Circuits

GORDON E. MOORE, LIFE FELLOW, IEEE

With unit cost falling as the number of components per circuit rises, by 1975 economics may dictate squeezing as many as 65 000 components on a single silicon chip.

1975: Yarı iletken karmaşıklığı, yaklaşık 1980 yılına kadar yıllık iki katına çıkmaya devam edecek ve bundan yaklaşık iki yılda bir iki katına çıkacaktı

1977
von Neumann darboğazı



John Backus

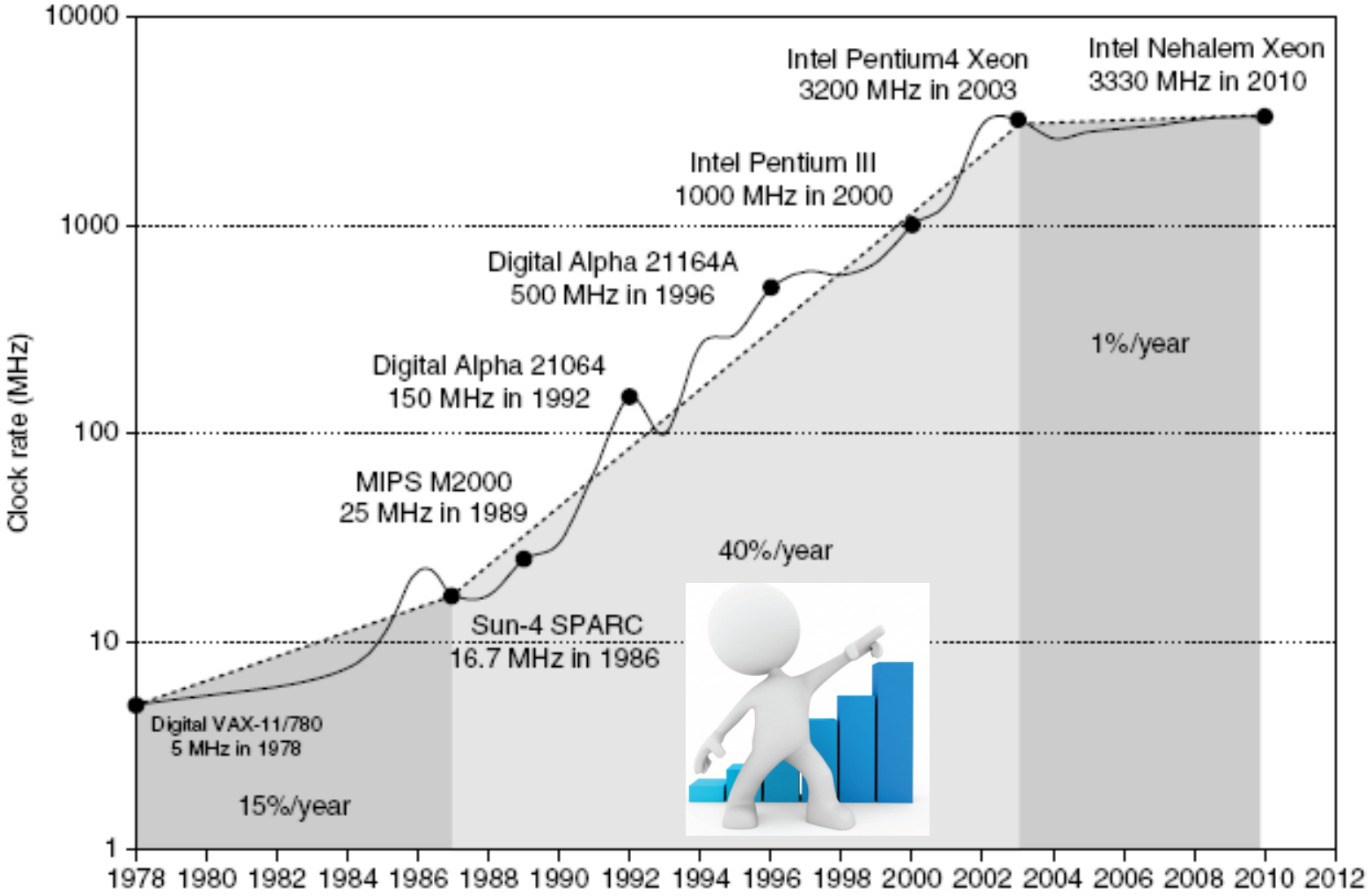
- Bir sistem darboğazı: Merkezi İşlemci Birimleri (CPU'lar) ve Rasgele Erişim Belleği arasındaki bant genişliği tipik bir CPU'nun dahili olarak veri işleyebileceği hızdan çok daha düşüktür.

- Bir 'entelektüel darboğaz': programcılar CPU ve RAM arasında gidip gelen "bir sürü gerekli/gereksiz kelimeyi" azaltmak için gerekli kod optimizasyonlarını düşünmek için çok zaman harcıyorlar.

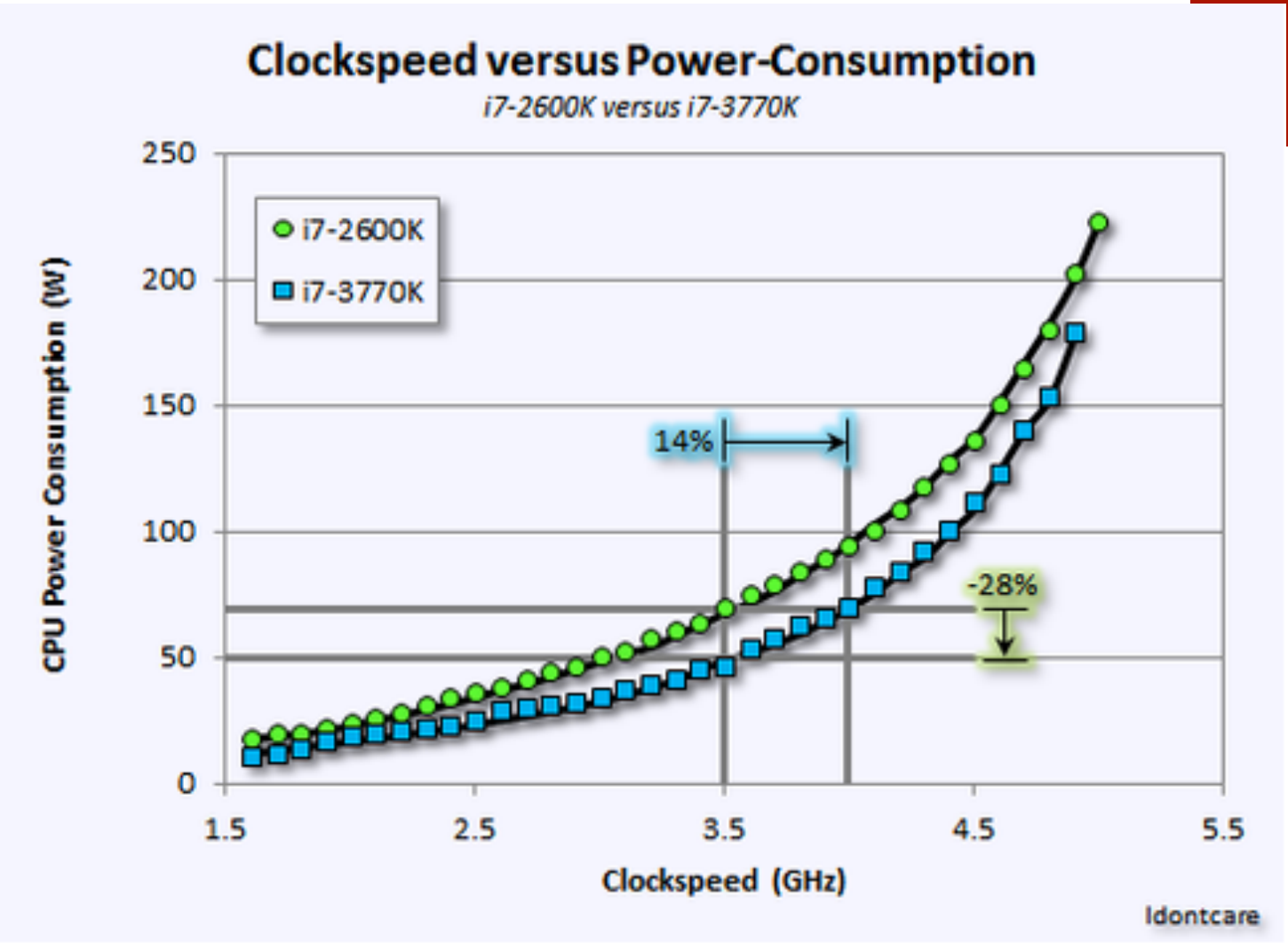
"… programming is basically planning and detailing the enormous traffic of words through the von Neumann bottleneck, and much of that traffic concerns not significant data itself, but where to find it." [1977 Turing Award Lecture]

Bir işlemci

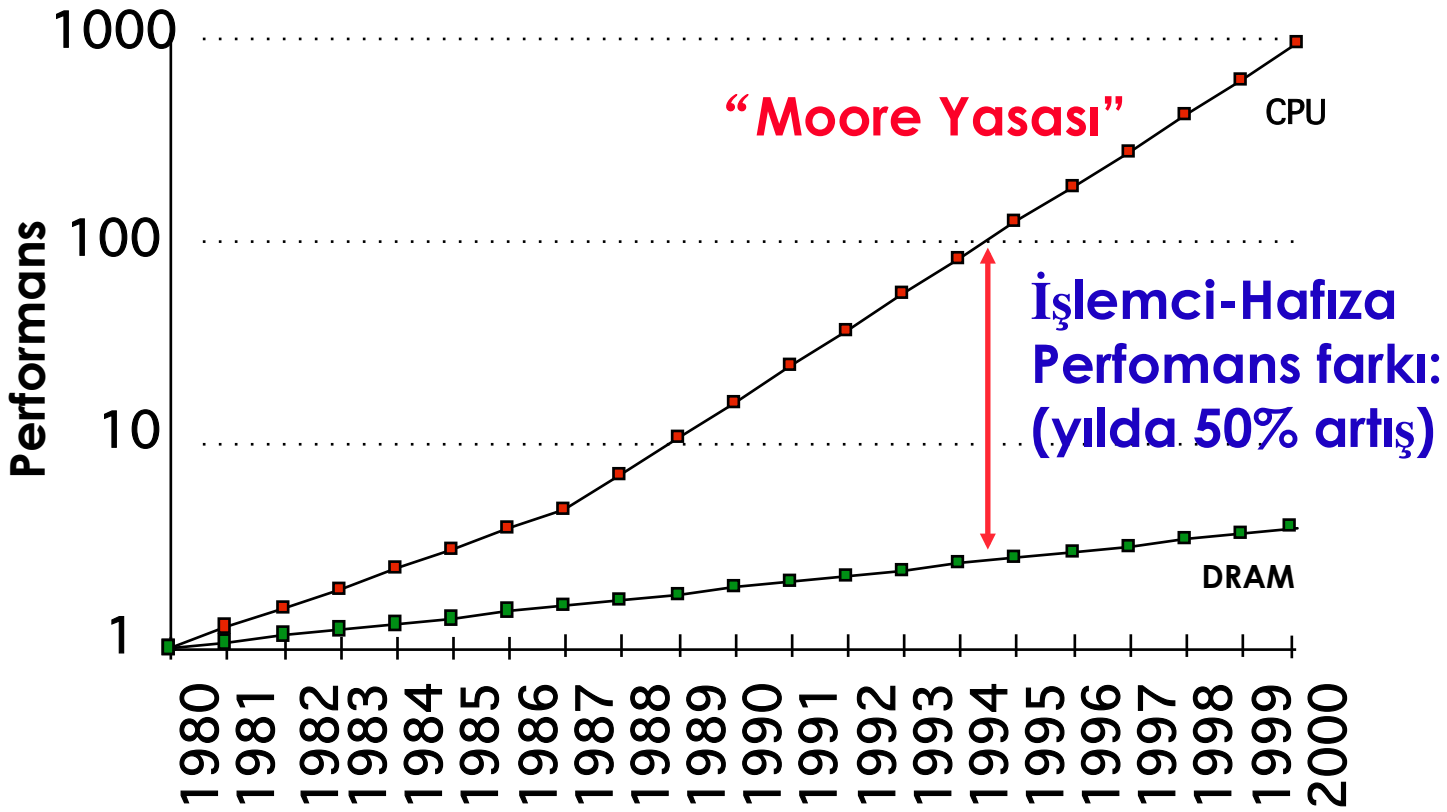
Bir zamanlar…



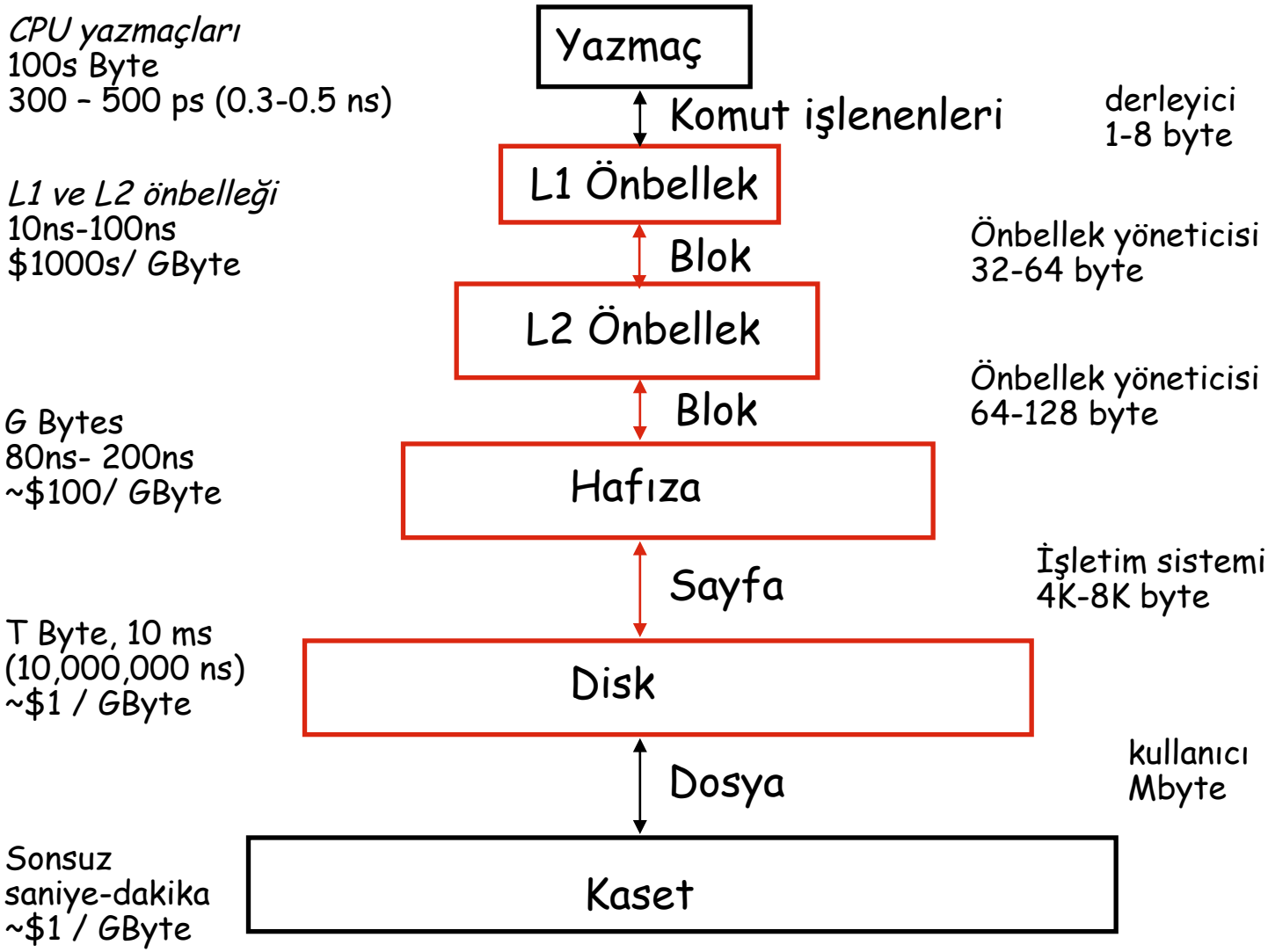
Güç duvarı


Clockspeed versus Power-Consumption
i7-2600K versus i7-3770K
CPU Power Consumption (W)
250
200
150
100
50
0
i7-2600K
i7-3770K
14%
-28%
1.5
2.5
3.5
4.5
5.5
Clockspeed (GHz)
Idontcare

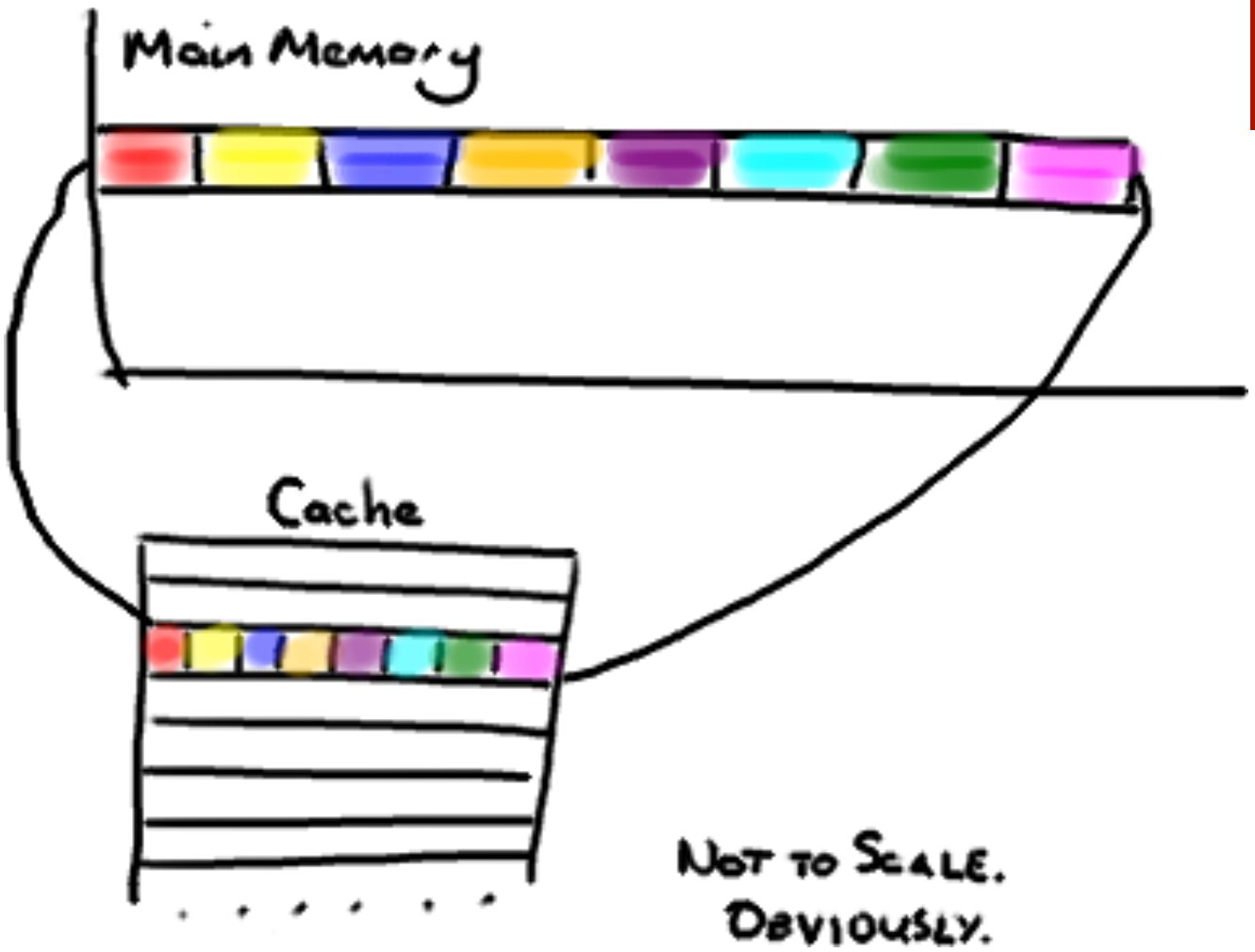
Hafıza duvarı



Hafıza duvarını aşmak



Önbellek: en iyi dostunuz



Trisha Gee

Matrislerin sütunlarının toplamı

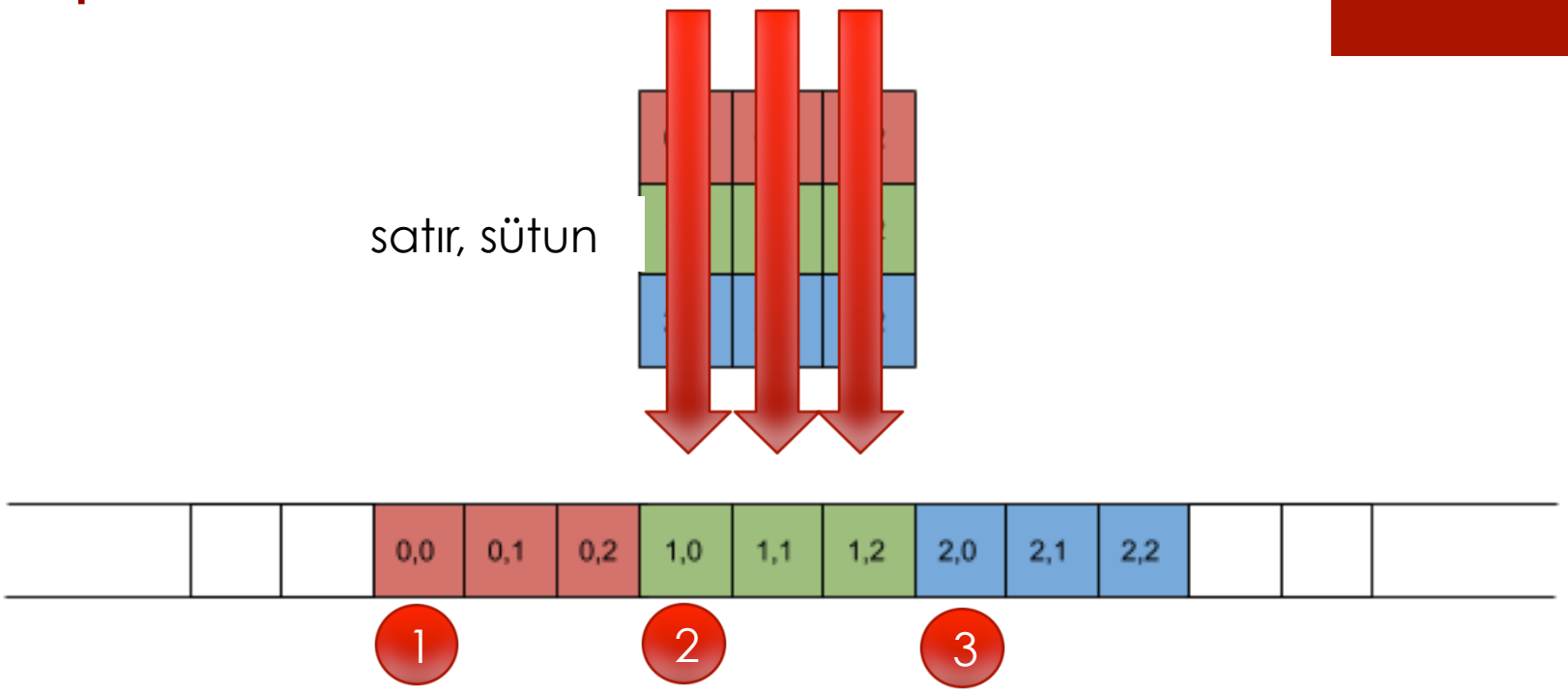
satır, sütun

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

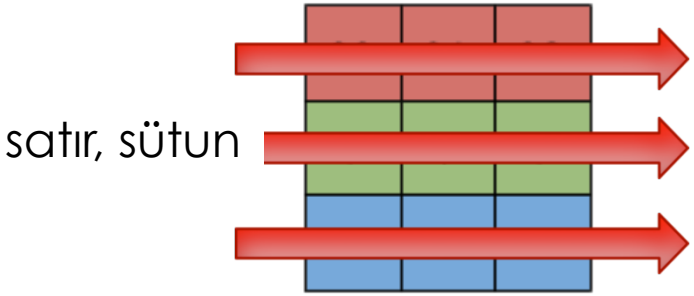
2d verilerin hafıza yapısı

Matrislerin sütunlarının toplamı

satır, sütun



Matrislerin sütunlarının toplamı



Matrislerin sütunlarının toplamı

```
for (i = 0; i < N; i++) {
  for (j = 0; j < N; j++) {
    column_sum[i] += b[j][i];
  }
}
```

2 saniye

```
for (j = 0; j < N; j++) {
  for (i = 0; i < N; i++) {
    column_sum[i] += b[j][i];
  }
}
```

0.1 saniye

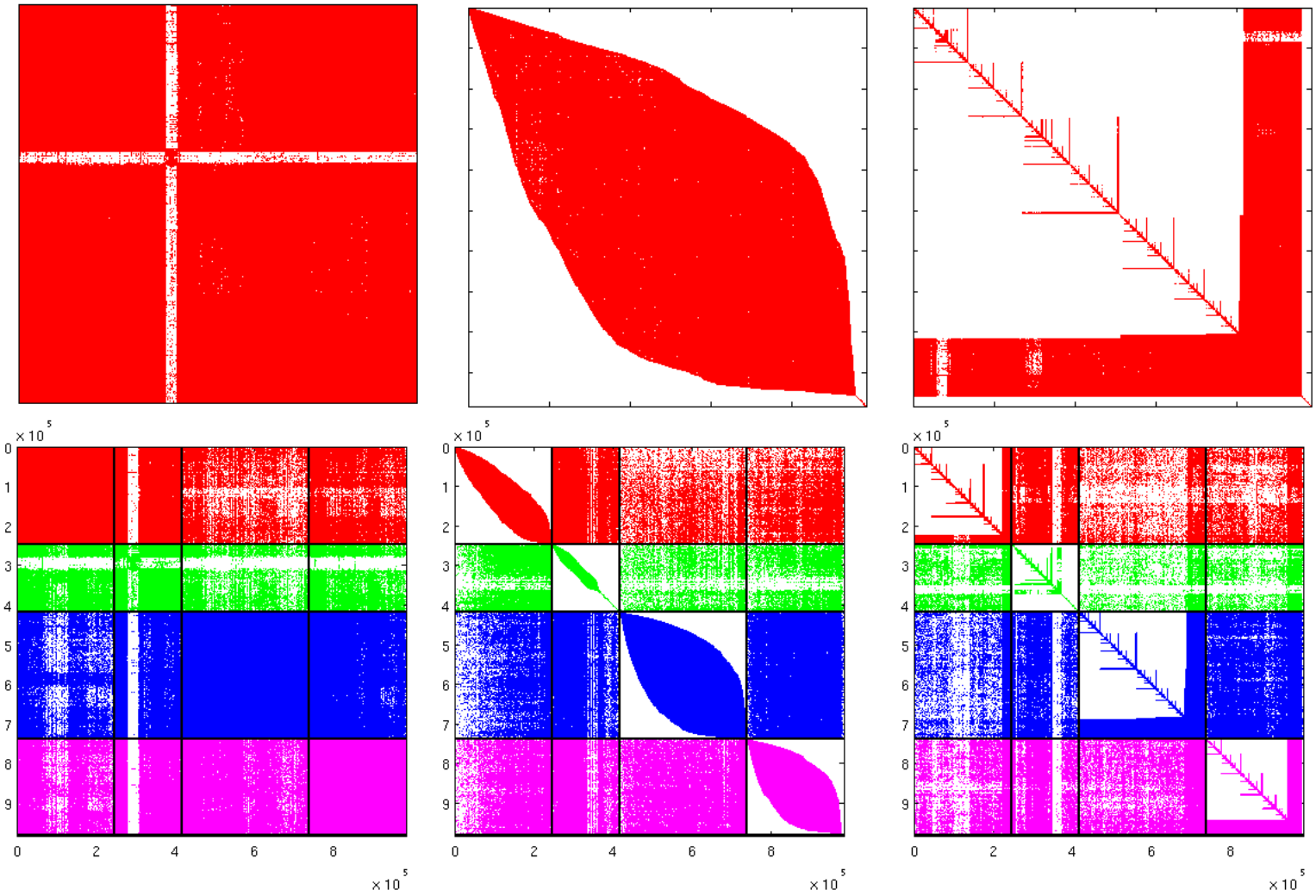
20x daha hızlı

Matris Çarpımı

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 2 & 0 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 4 & 4 \\ 10 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 0 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 7 & 10 \end{bmatrix}$$

Matris sıralama



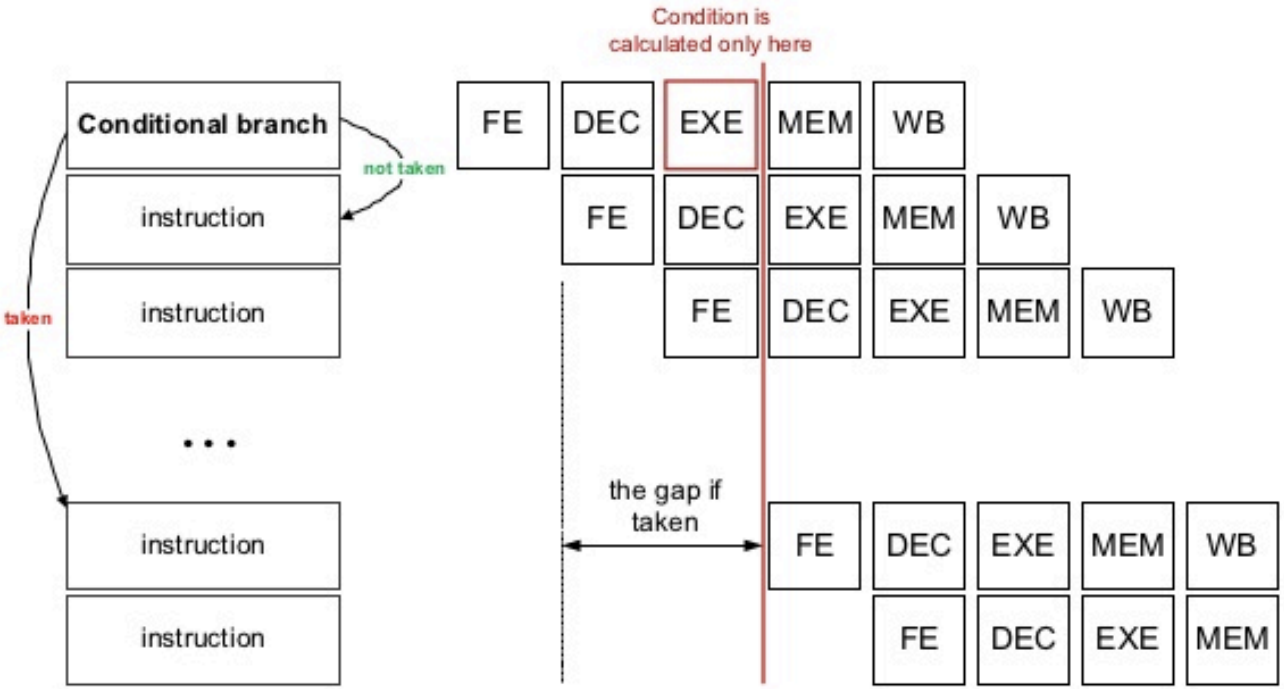
Komut seviyesinde koşutluk

Program to evaluate $X = (A + B) * (C + D)$:

```
ADD   R1, A, B        /*  R1 □ M[A] + M[B]    */
ADD   R2, C, D        /*  R2 □ M[C] + M[D]    */
MUL   X, R1, R2       /*  M[X] □ R1 * R2      */
```

Komutlar: Düzenleme, yeniden sıralama vb.

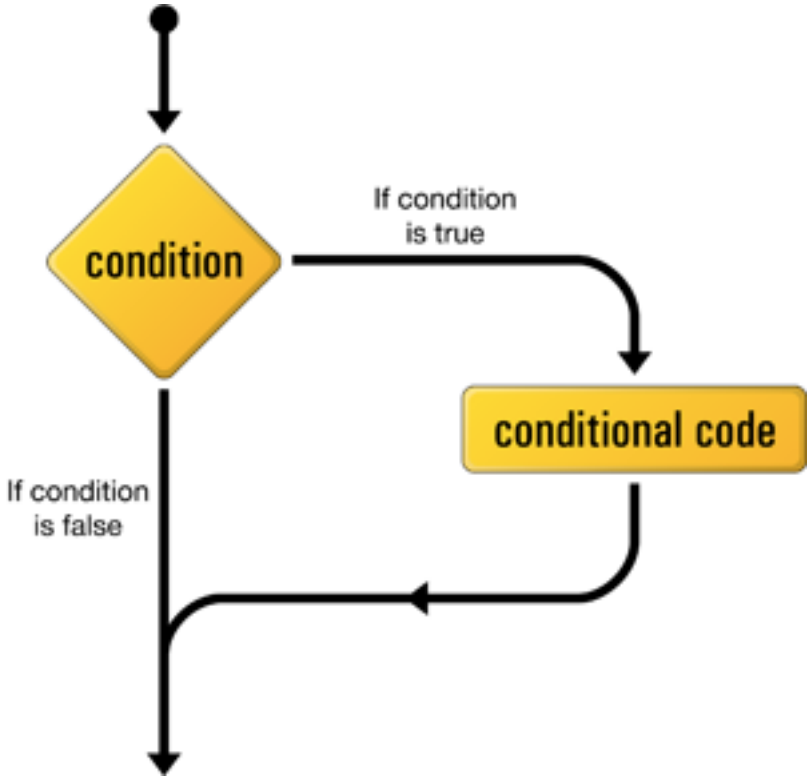
- Her zaman kodladığınız gibi çalışmıyor



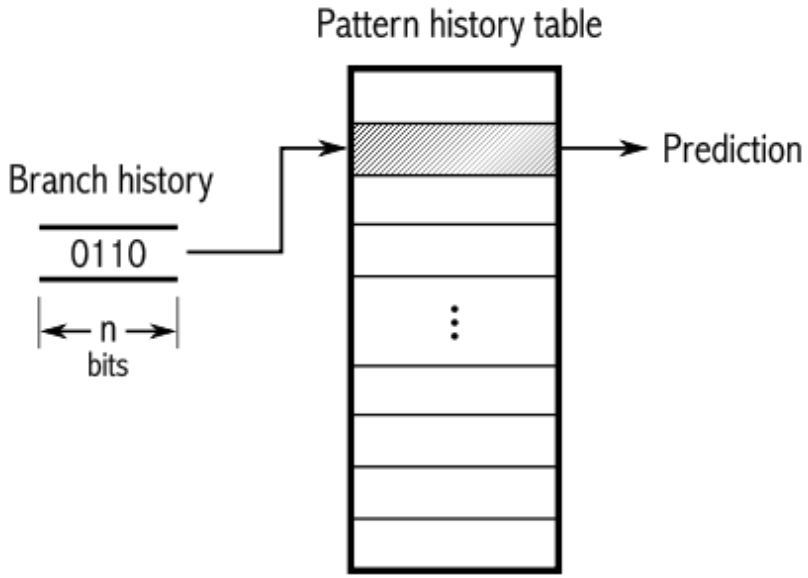
Pipelining

Komutlar: Düzenleme, yeniden sıralama vb.

- Ne çalıştırmak istediğimizi bilmiyorsak



Komutlar: Düzenleme, yeniden sıralama vb.



Dal tahmincisi

Komutlar: Düzenleme, yeniden sıralama vb.


```
for (unsigned c = 0; c < arraySize; ++c) {
  data[c] = std::rand() % 256;
}
```

```
long long sum = 0;
for (unsigned i = 0; i < 100000; ++i) {
  for (unsigned c = 0; c < arraySize; ++c) {
    if (data[c] >= 128)
      sum += data[c];
  }
}
```

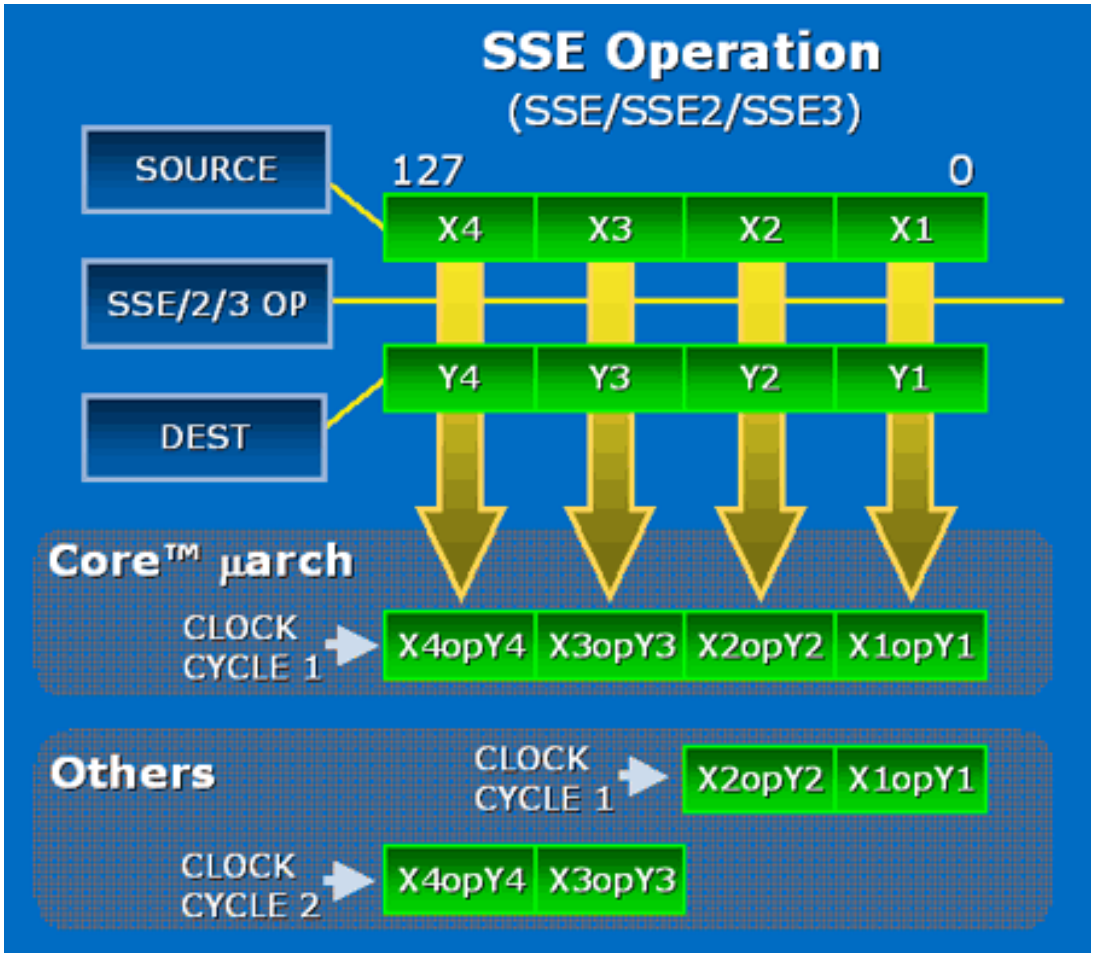
13.2 saniye, 1.5 B yanlış tahmin

Komutlar: Düzenleme, yeniden sıralama vb.


```
for (unsigned c = 0; c < arraySize; ++c) {
  data[c] = std::rand() % 256;
}

std::sort(data.begin(), data.end());

long long sum = 0;
for (unsigned i = 0; i < 100000; ++i) {
  for (unsigned c = 0; c < arraySize; ++c) {
    if (data[c] >= 128)
      sum += data[c];
  }
}
```

2.1 saniye, 364 K yanlış tahmin

Komut seviyesinde koşutluk



Komut seviyesinde koşutluk: matris-vektör çarpımı örneği

Matris-vektör çarpımı

Hafıza duvarını aşmak

- Bu değil

```
for (i = 0; i < N; i++) {
    sum += a[i]*b[i];
}
```

- Belki de bu

```
for (i = 0; i < N; i++){
    fetch (&a[i+1]);
    fetch (&b[i+1]);
    sum += a[i]*b[i];
}
```

Hafıza duvarını aşmak

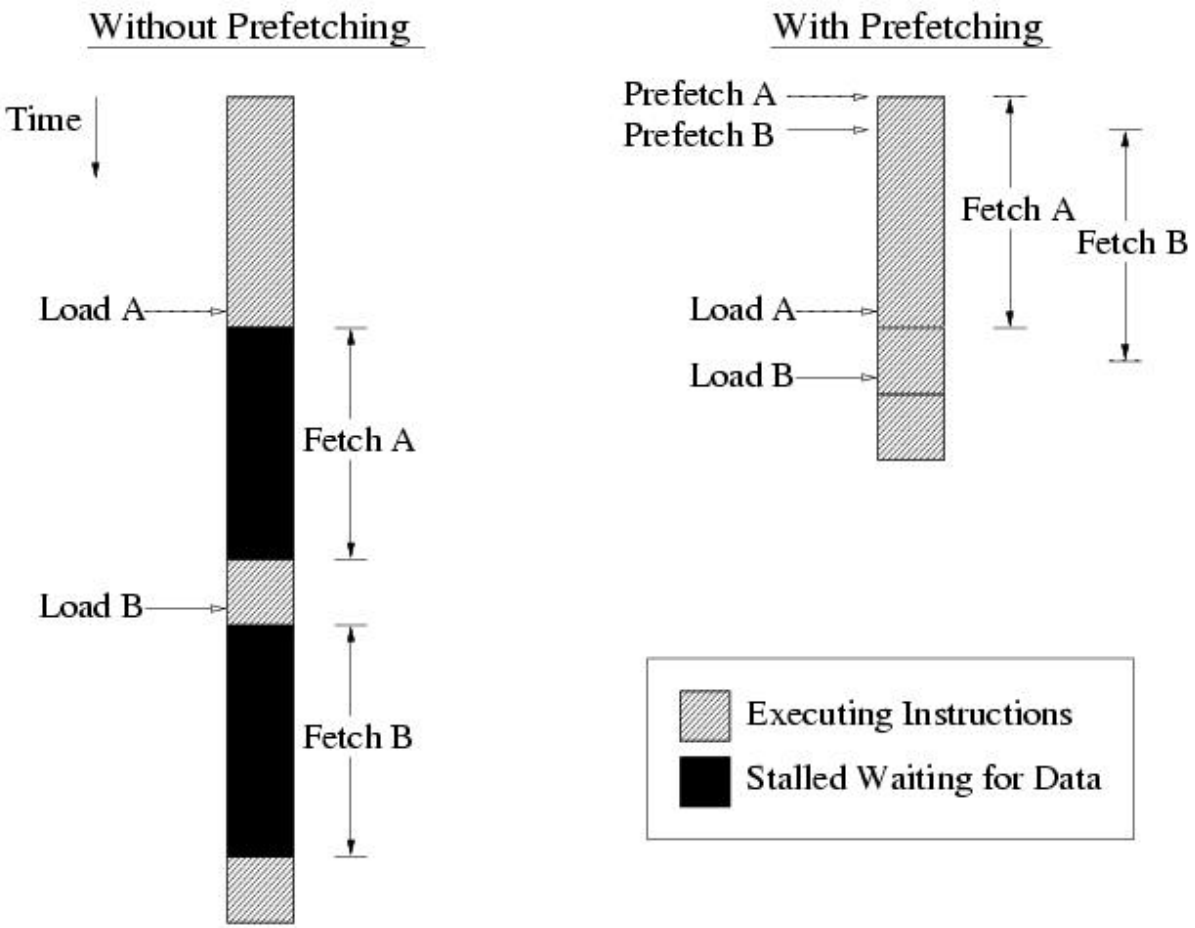
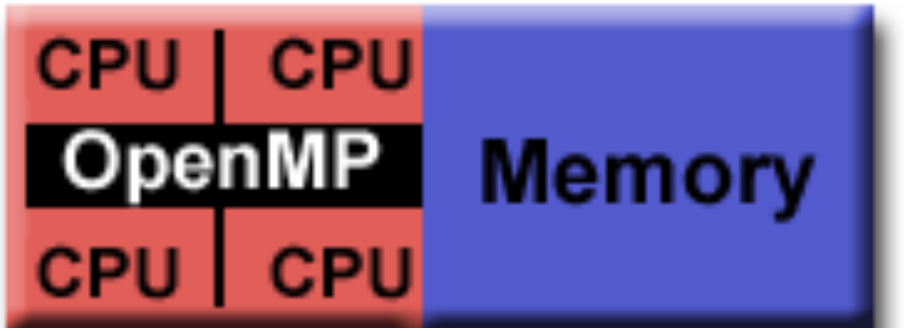


Figure 1.4: Illustration of how prefetching tolerates memory latency.

Bir işlemciden daha fazlası

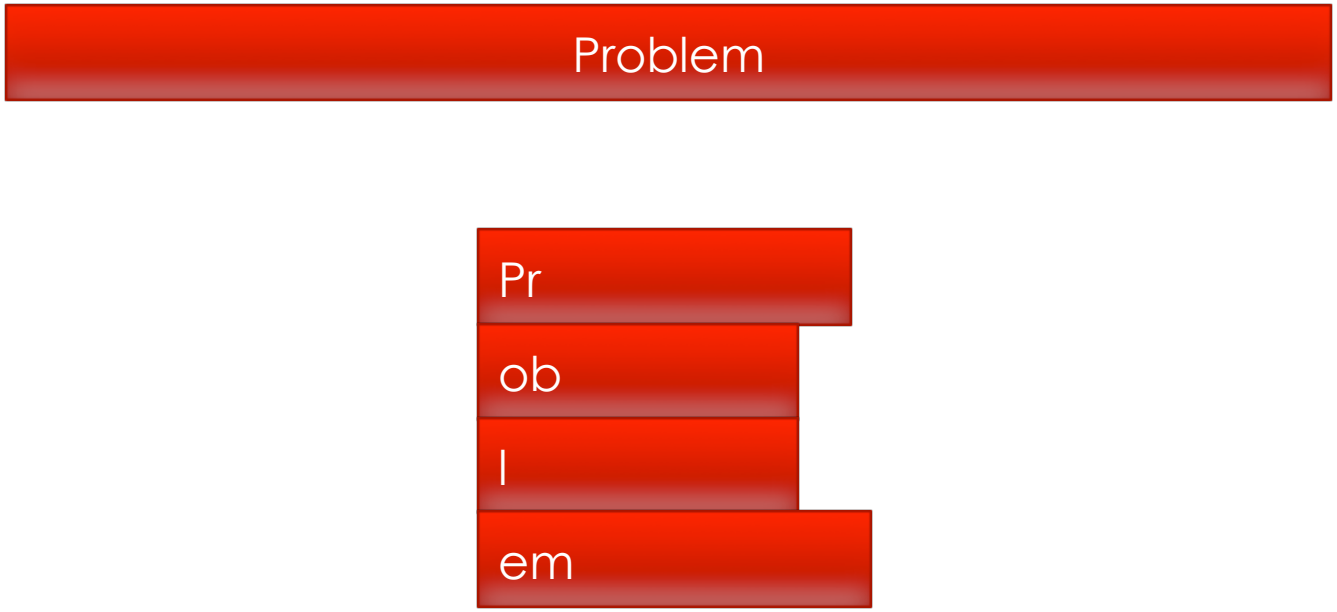


Bugünün işlemcileri

- Intel E7-8890v3
  - Frekans: 2.2 GHz
  - 14nm
  - Çekirdek: 24
  - Hafıza: 3.07TB / soket
  - Hafıza veriyolu: 85 GBs
  - 60 MB önbellek

Paralel (Koşut) İşlemler

- Aynı anda birden fazla işlemin işlemci tarafından gerçekleştirilmesi.



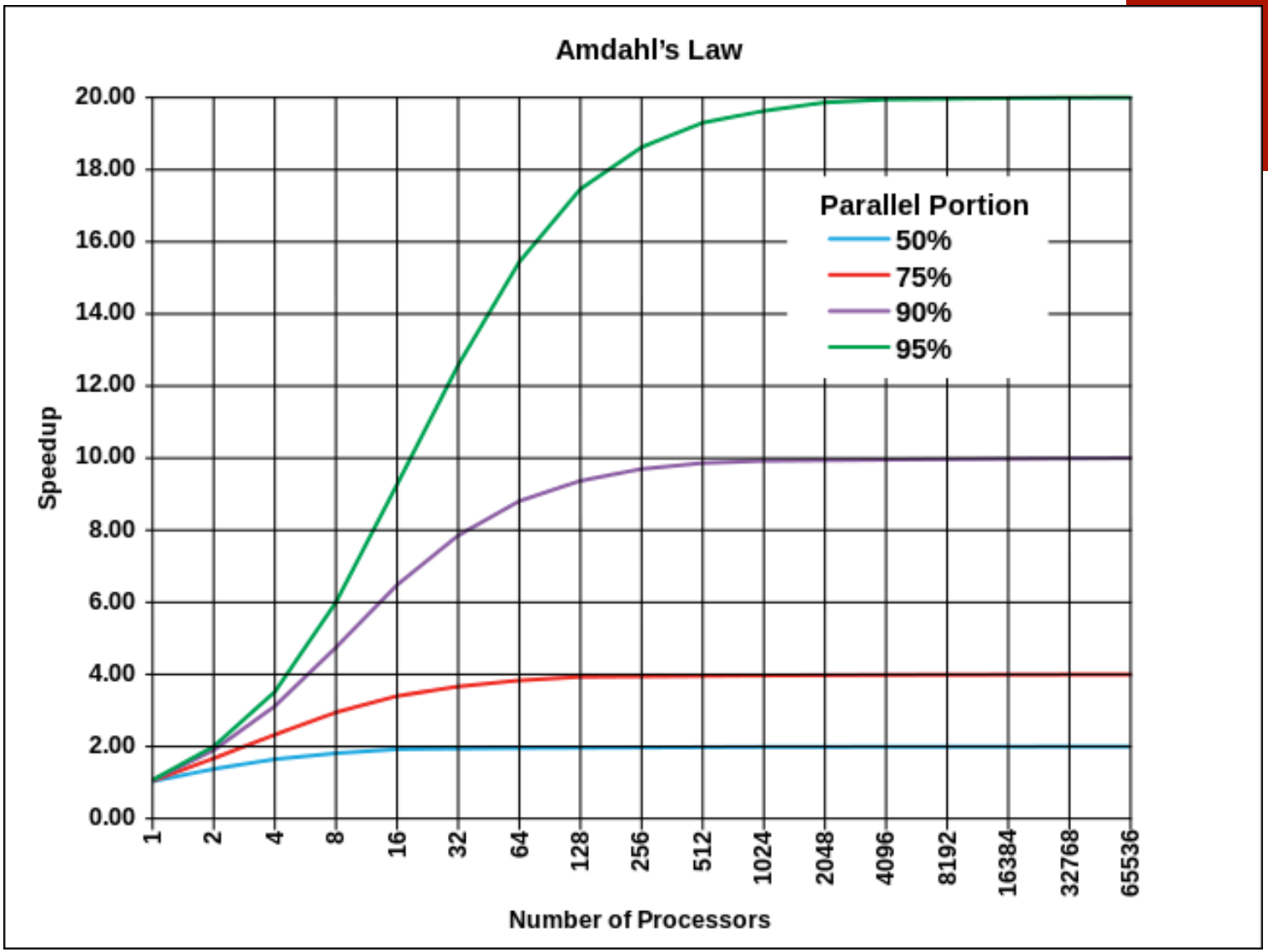
Birden fazla işlemci ve hızlanma

Bir problemi tek işlemci ile **T**(1) zamanda çözelim.

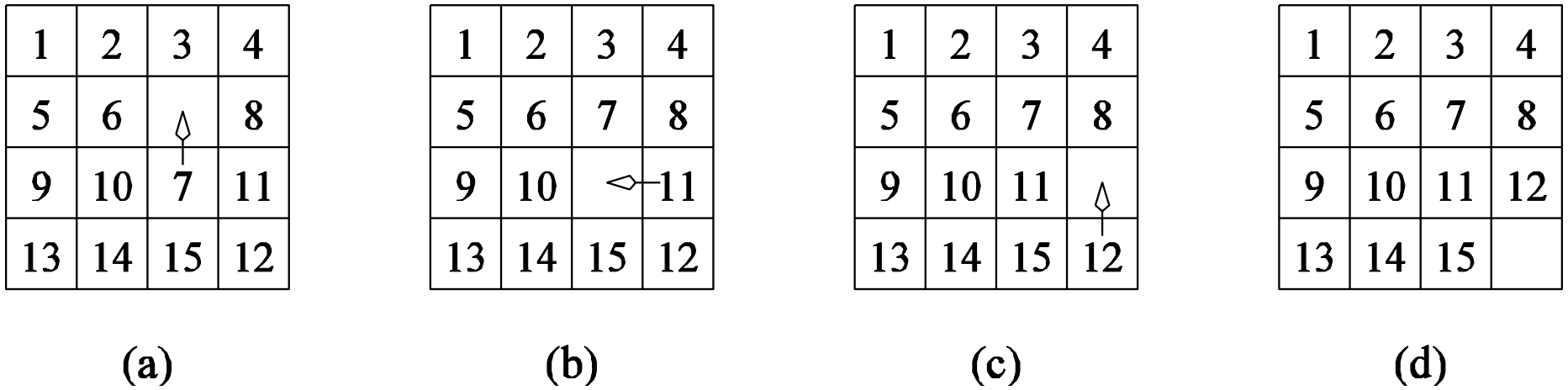
Aynı problem **p** işlemci ile **T**(**p**) zamanında çözülsün.

Hızlanma oranı **T**(1) / **T**(**p**)'dir.

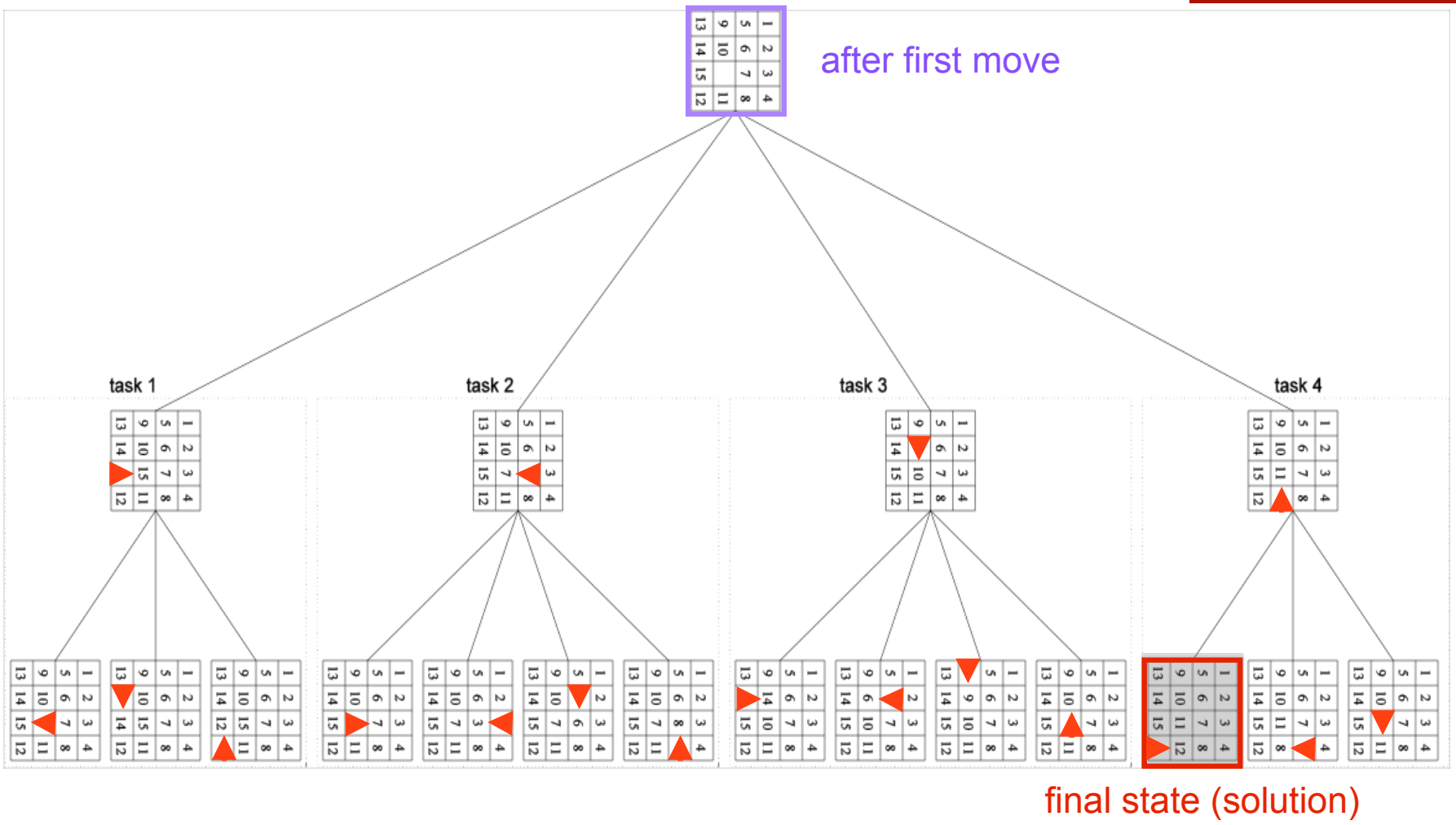
Amdahl Kuralı



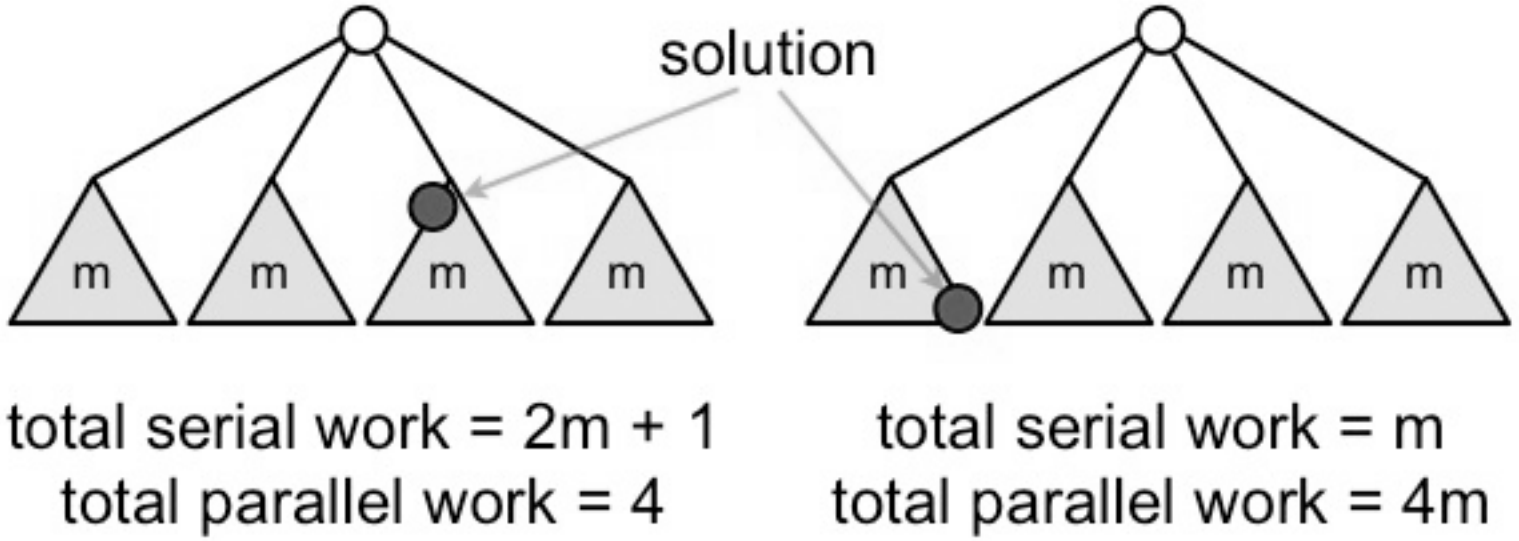
Aykırı bir problem



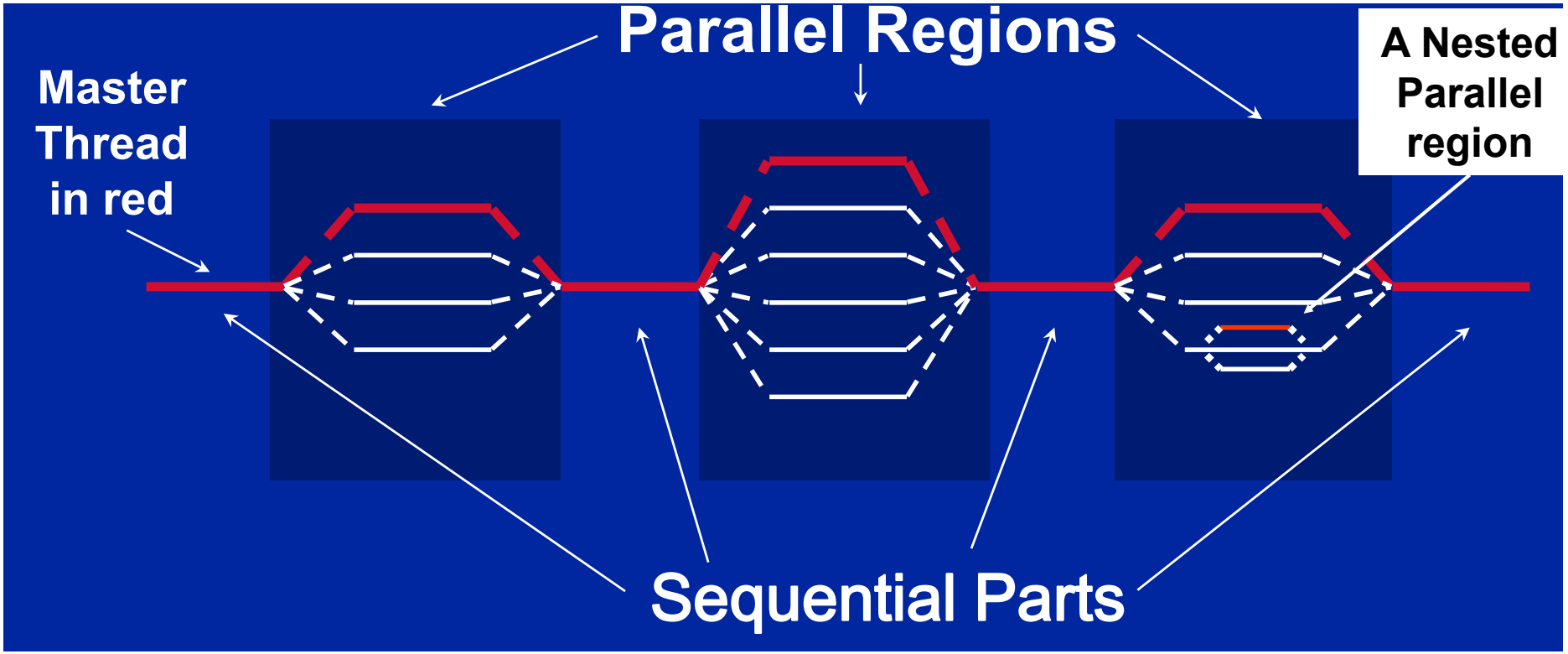
Aykırı bir problem



Aykırı bir problem



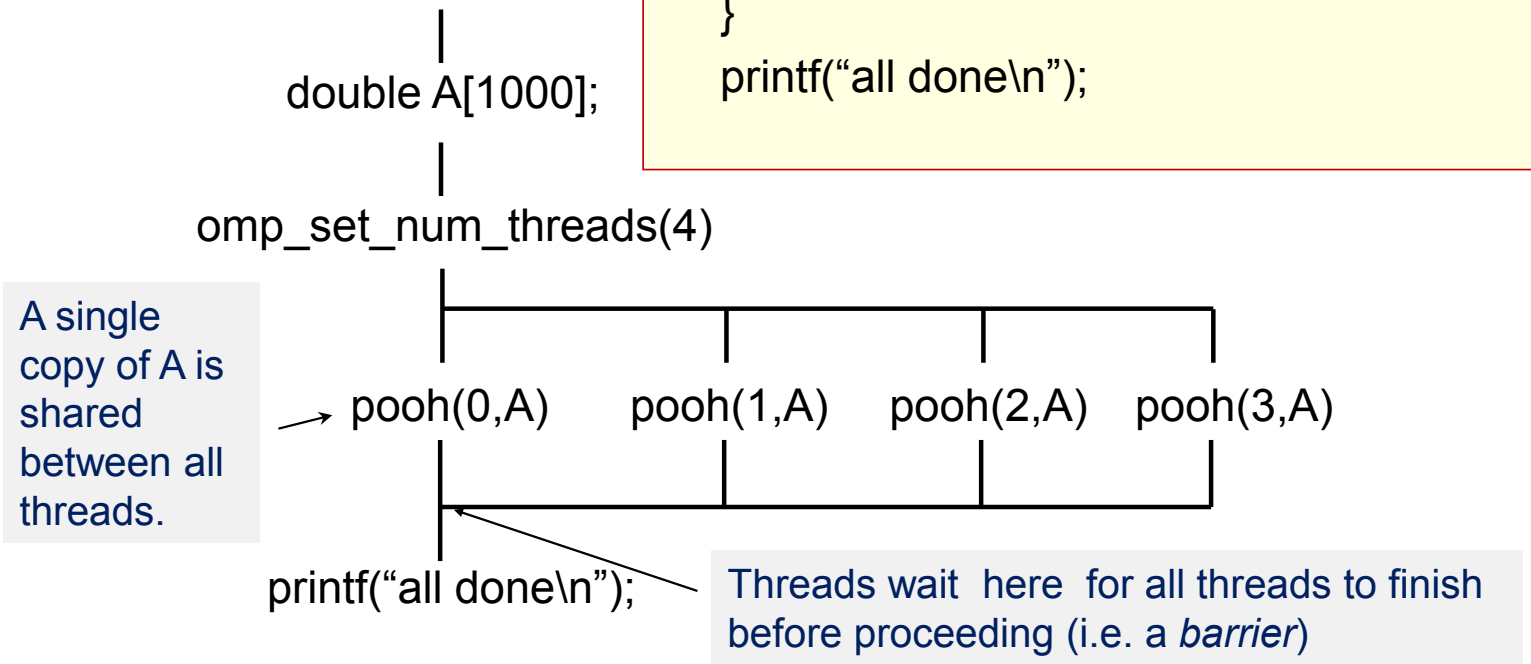
Çok çekirdekli işlemcileri programlama: OpenMP



Çok çekirdekli işlemcileri programlama: OpenMP

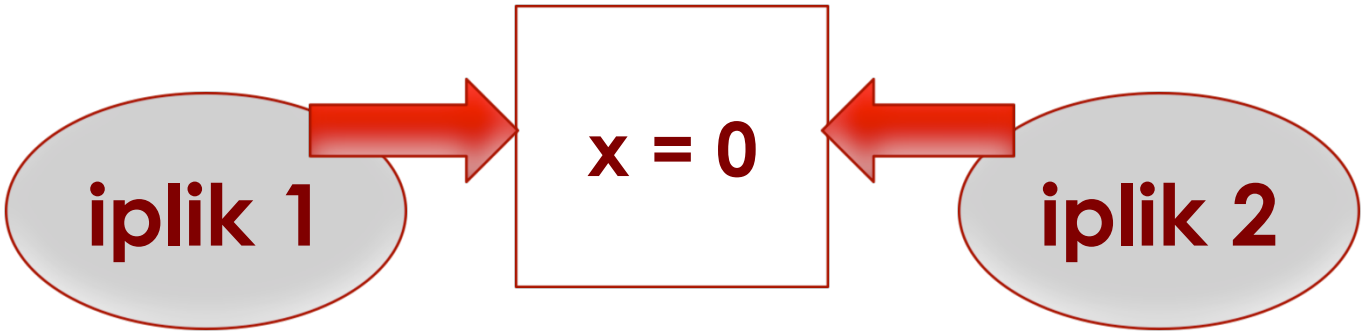
- Each thread executes the same code redundantly.

```
double A[1000];
#pragma omp parallel num_threads(4)
{
    int ID = omp_get_thread_num();
    pooh(ID, A);
}
printf("all done\n");
```



Çok çekirdekli işlemcileri programlama: OpenMP

PARALEL PROGRAMLAMA 101

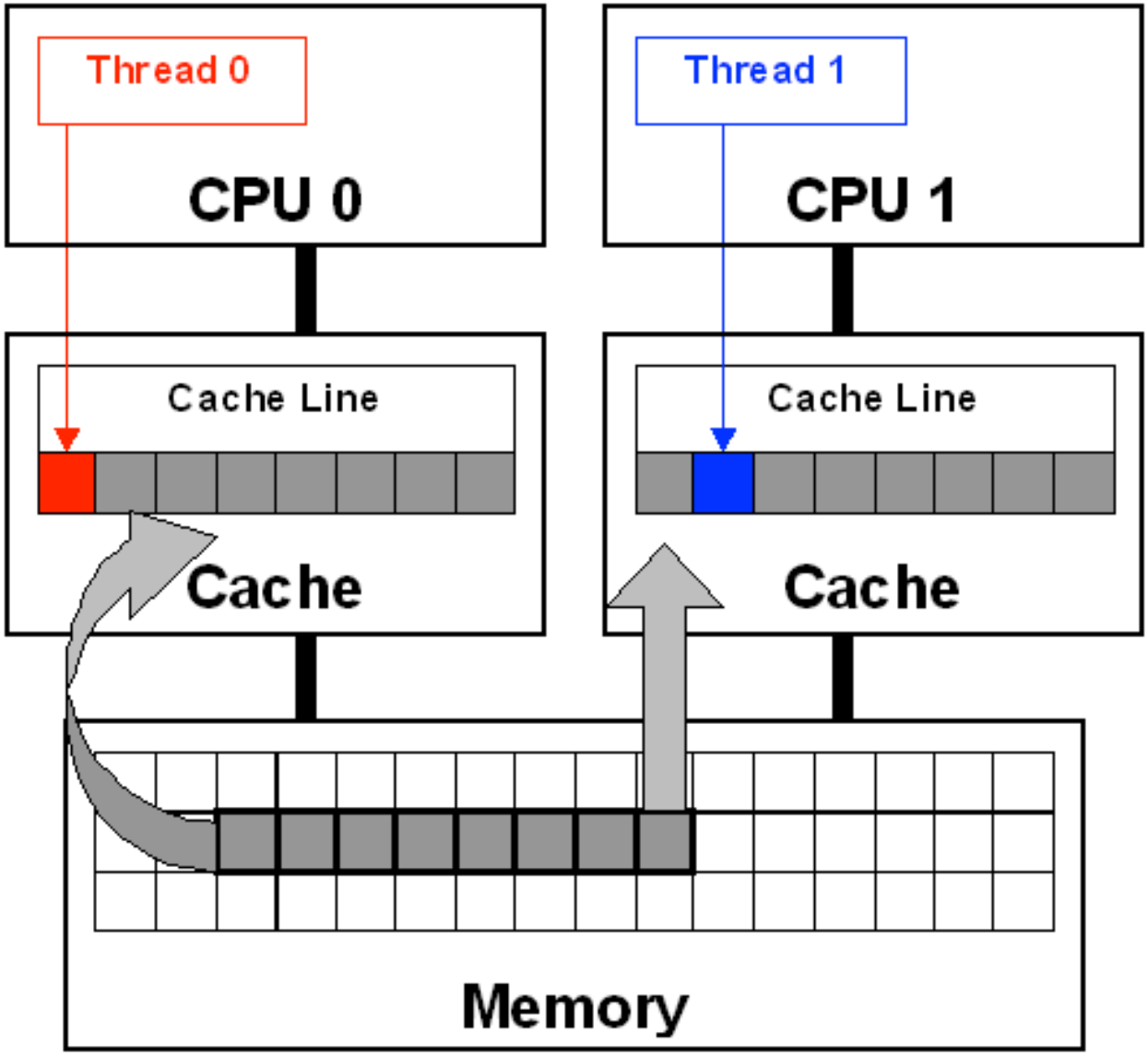


x = x + 1 **x = x + 1**

OpenMP: vektör çarpımı


```
double sum = 0.0;
double sum_local[NUM_THREADS];
#pragma omp parallel num_threads(NUM_THREADS)
{
  int me = omp_get_thread_num();
  sum_local[me] = 0.0;
  #pragma omp for
  for (i = 0; i < N; i++)
     sum_local[me] += x[i] * y[i];
  #pragma omp atomic
  sum += sum_local[me];
}
```

OpenMP


Thread 0
Thread 1
CPU 0
CPU 1
Cache Line
Cache Line
Cache
Cache
Memory

OpenMP: vektör çarpımı

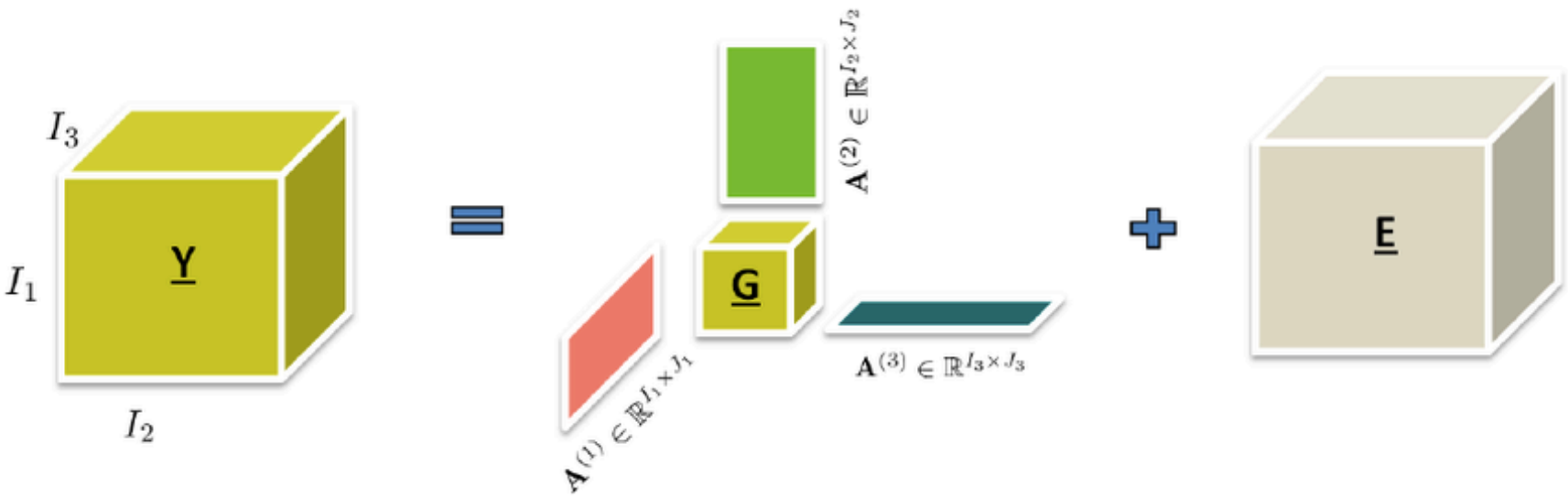

```
double sum = 0.0;
double sum_local[NUM_THREADS][16];
#pragma omp parallel num_threads(NUM_THREADS)
{
  int me = omp_get_thread_num();
  sum_local[me][0] = 0.0;
  #pragma omp for
  for (i = 0; i < N; i++)
     sum_local[me] += x[i] * y[i];
  #pragma omp atomic
  sum += sum_local[me][0];
}
```

OpenMP

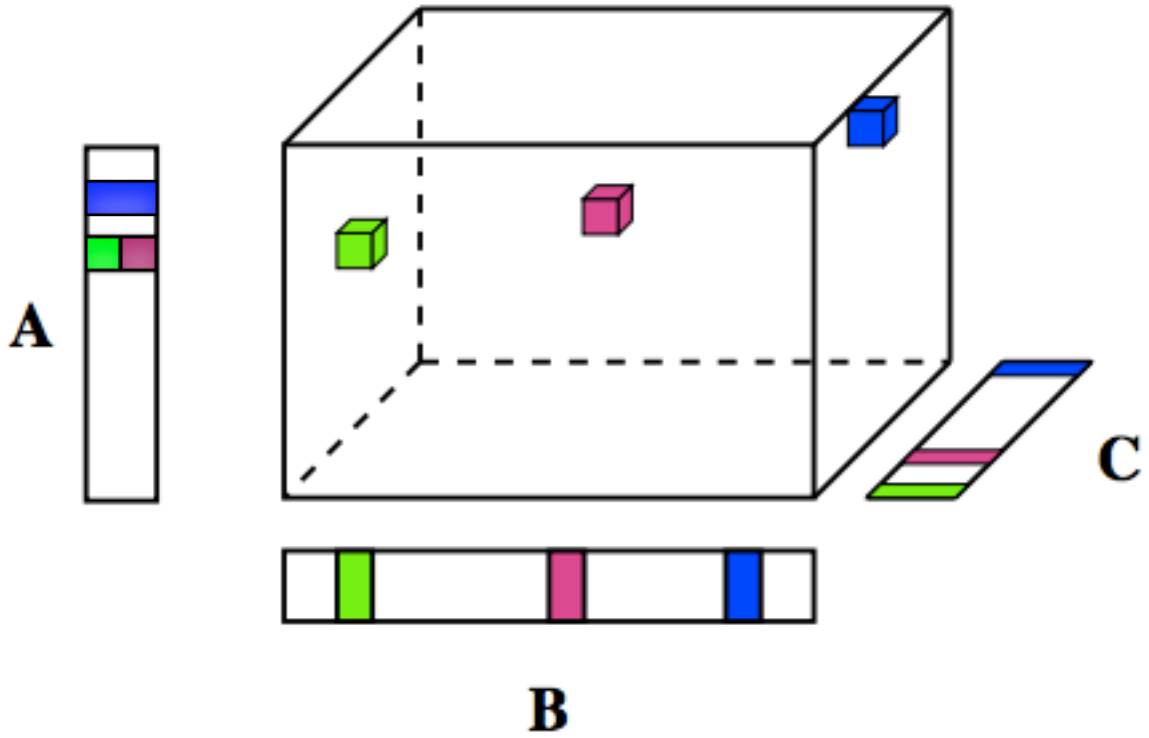
Threads wait their turn – only one at a time calls consume()

```
float  res;
#pragma omp parallel
{    float B;   int i, id, nthrds;
     id = omp_get_thread_num();
     nthrds = omp_get_num_threads();
      for(i=id;i<niters;i+=nthrds){
           B =  big_job(i);
#pragma omp critical
           res += consume (B);
     }
}
```

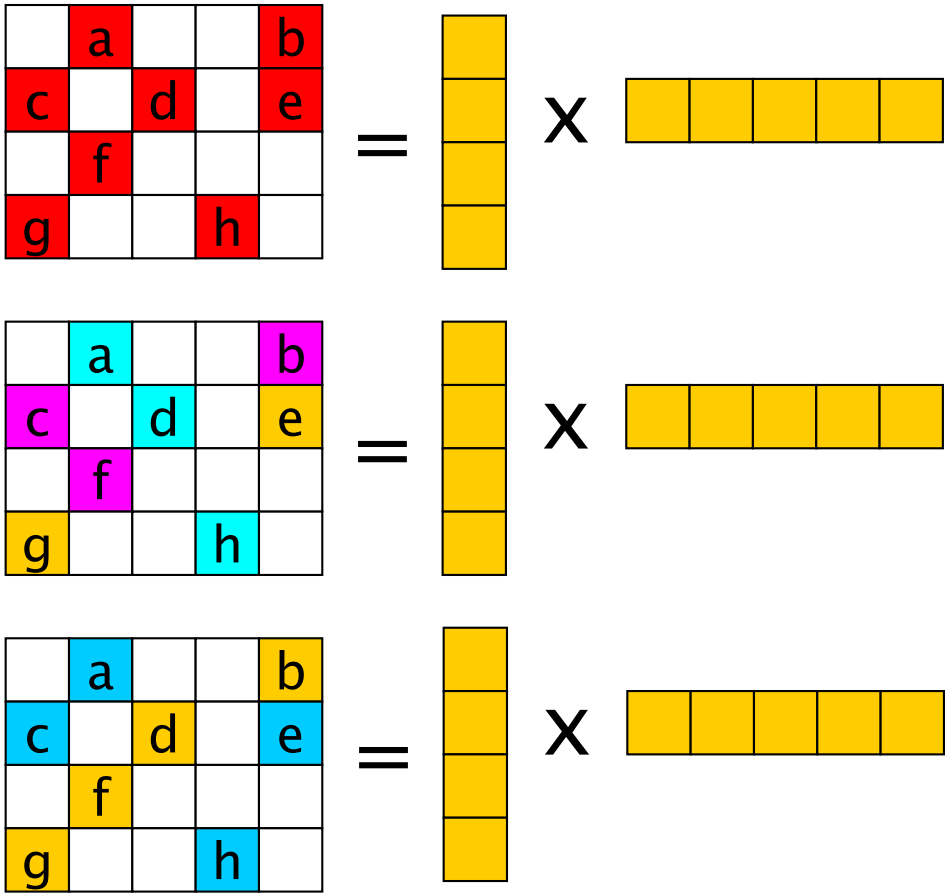
Örnek problem: Tensör ayrıştırma



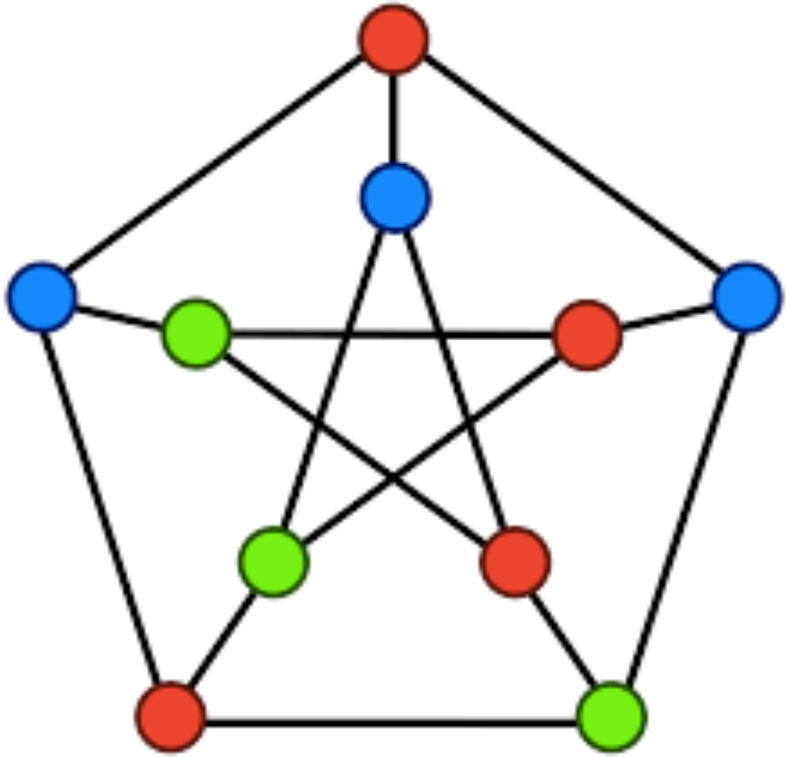
Örnek problem: Koşut tensör ayrıştırma



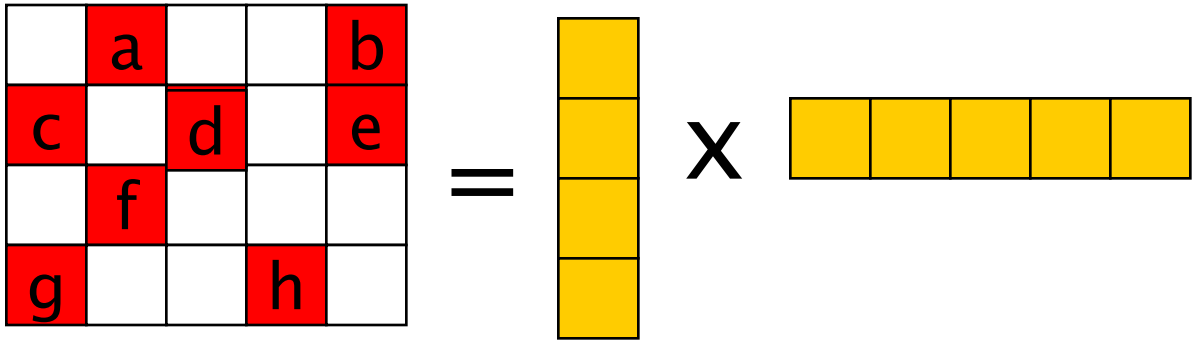
Örnek problem: Koşut tensör ayrıştırma

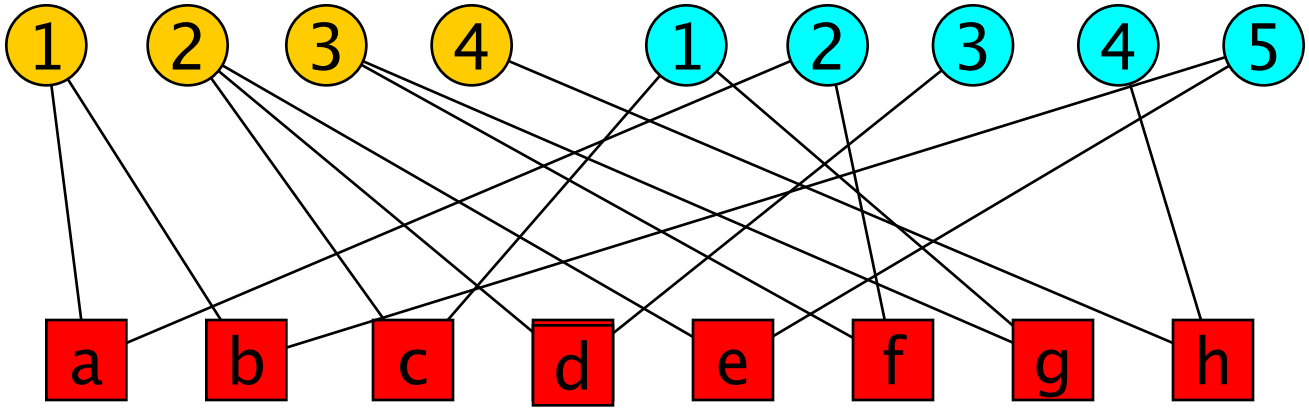


Çizge boyama problemi

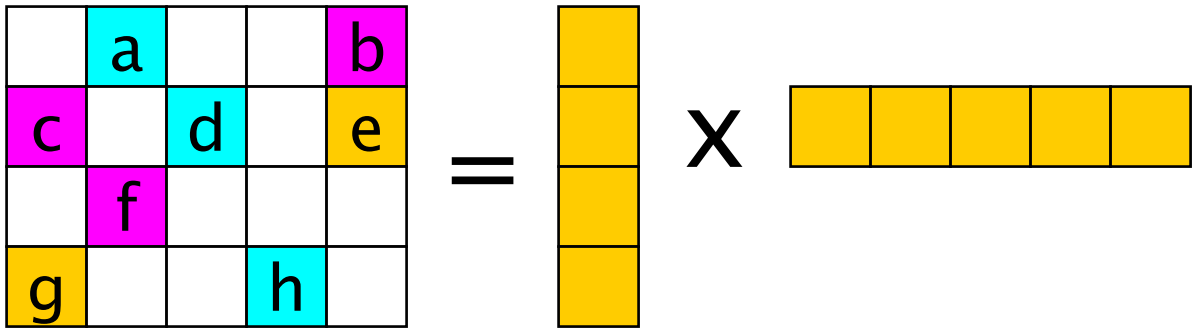


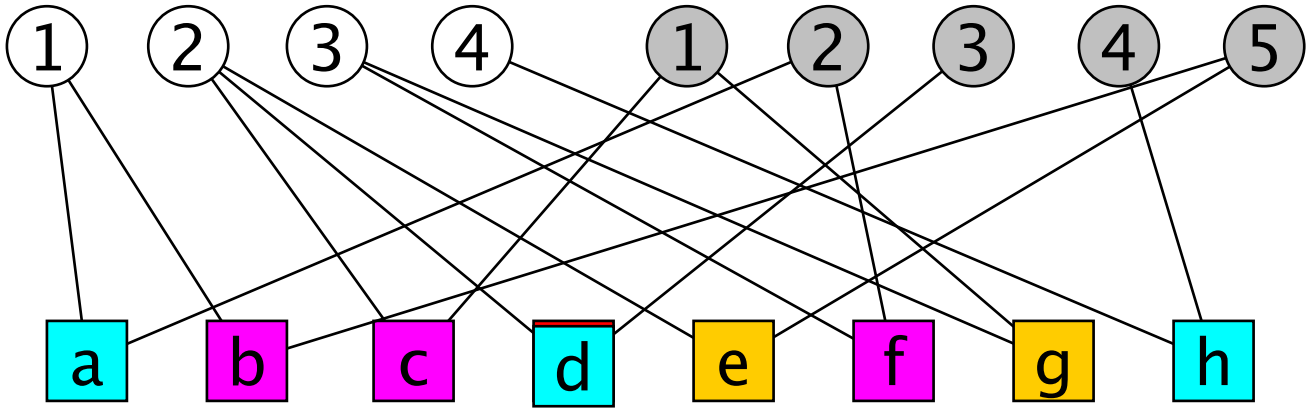
Hiperçizge ile modelleme



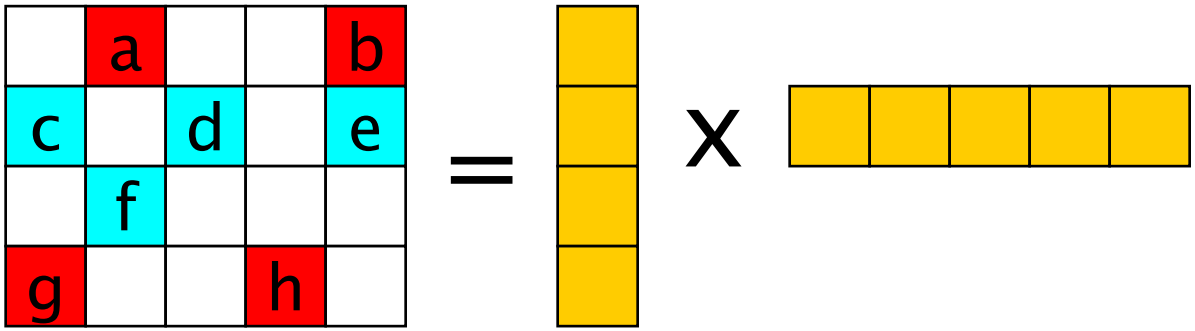


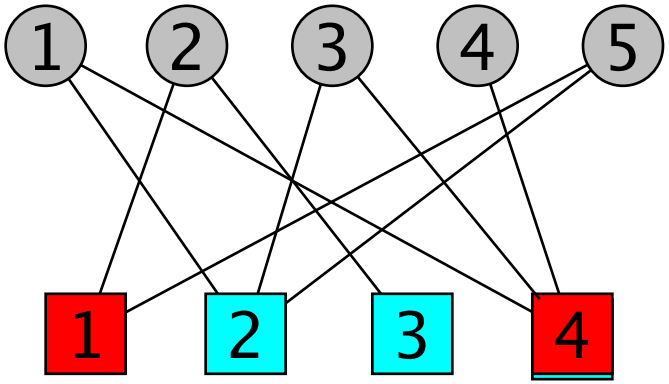
Hiperçizge boyama



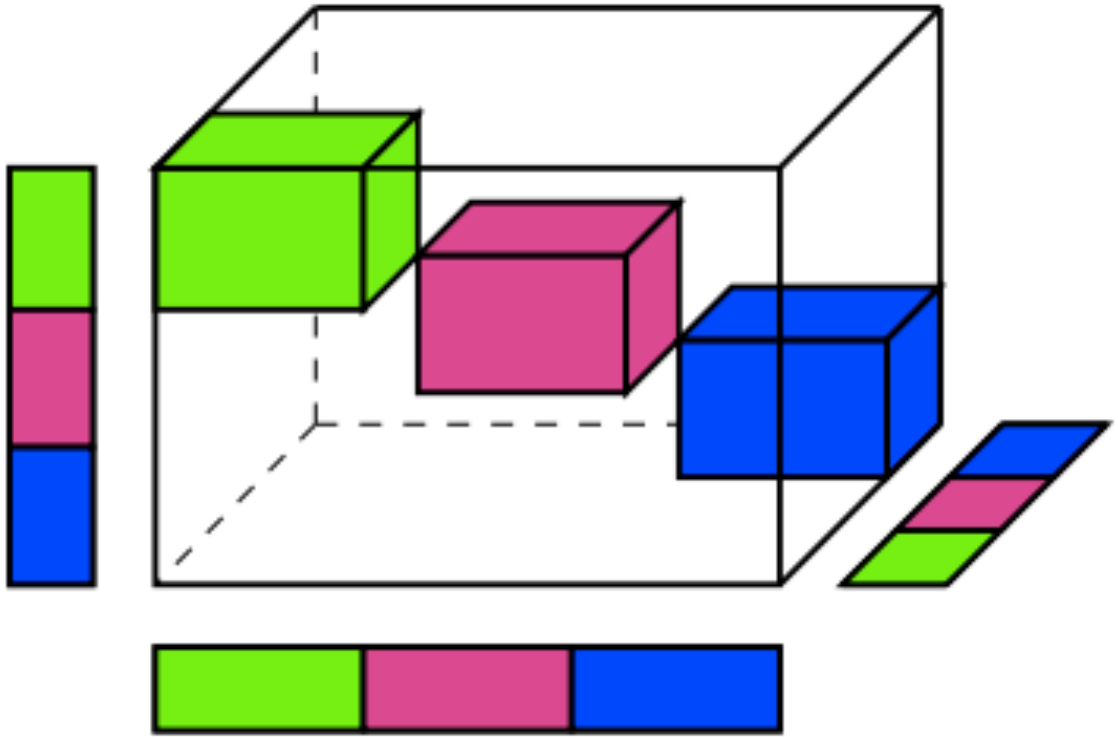


Hiperçizge modelleme/ boyama





Koşut tensör ayrıştırma

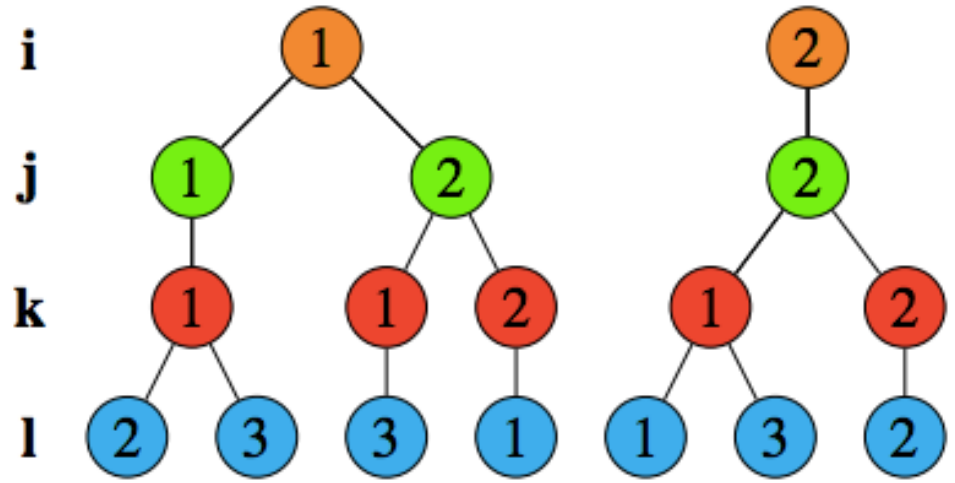


Koşut tensör ayrıştırma

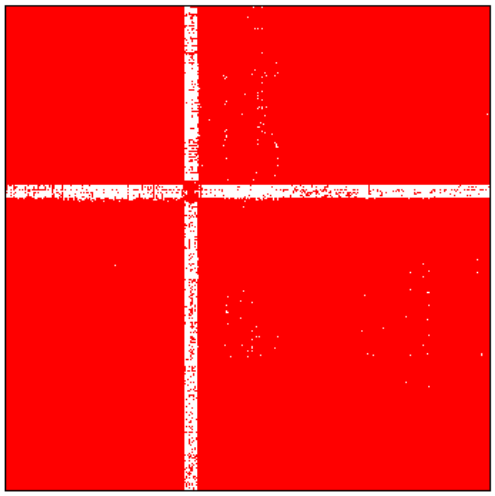
```
155758                                39848
113027                                37751
75902                                 38389
69064                                 37083
47450                                 41190
39880                                 39587
20634                                 41029
22481                                 39657
15741                                 37098
14061                                 39542
16174                                 37720
8107                                  39300
6846                                  40733
12103                                 39121
11866                                 40461
3680                                  38434
INT 15 0.518887 5.99912               INT 15 0.200038 5.99988
INT 31 1.05262 5.74374                INT 31 0.415625 5.74477
INT 47 1.58465 5.57151                INT 47 0.629195 5.57245
INT 63 2.11765 5.43935                INT 63 0.843154 5.44008
```

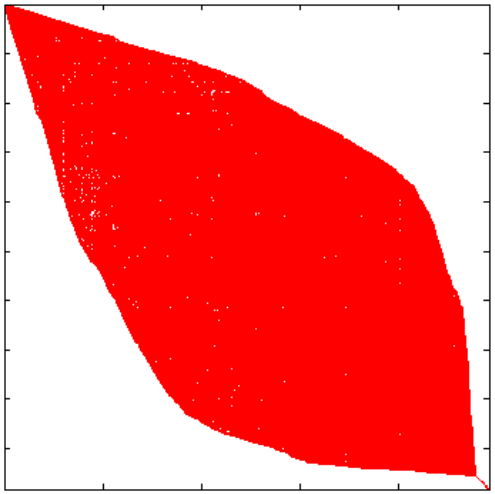
Tensör depolama

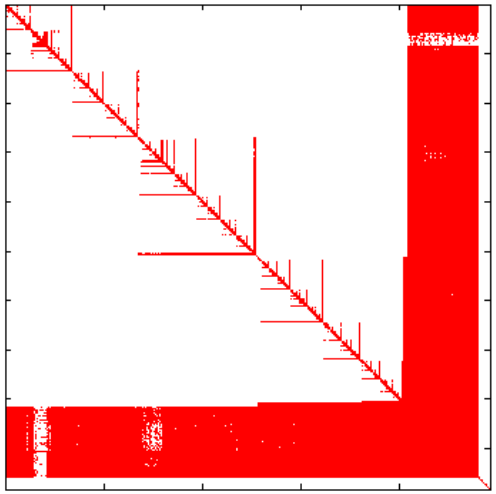
i	j	k	l
1	1	1	2
1	1	1	3
1	2	1	3
1	2	2	1
2	2	1	1
2	2	1	3
2	2	2	2



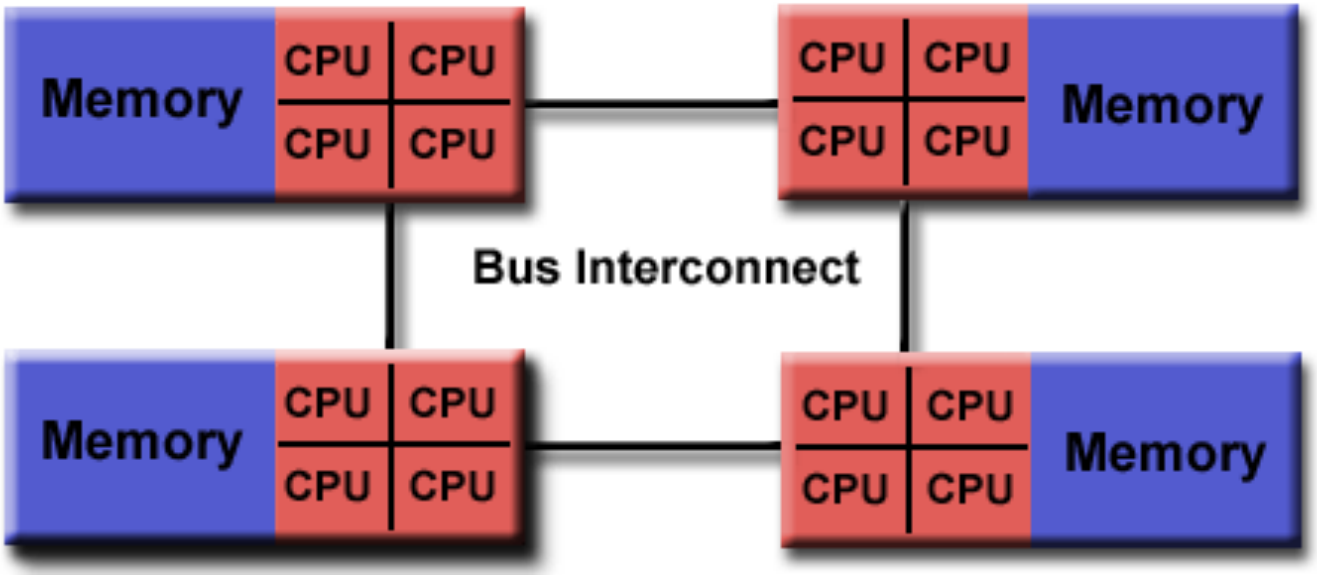
Tensör sıralama?



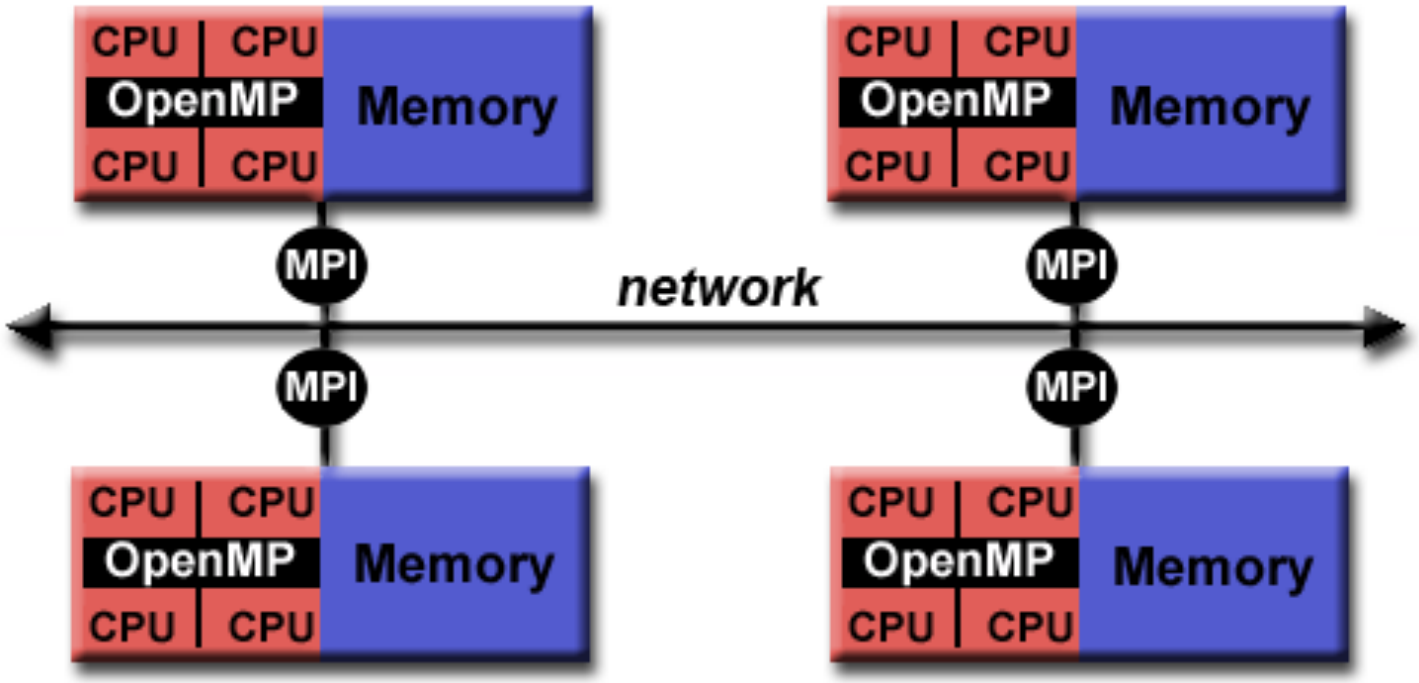




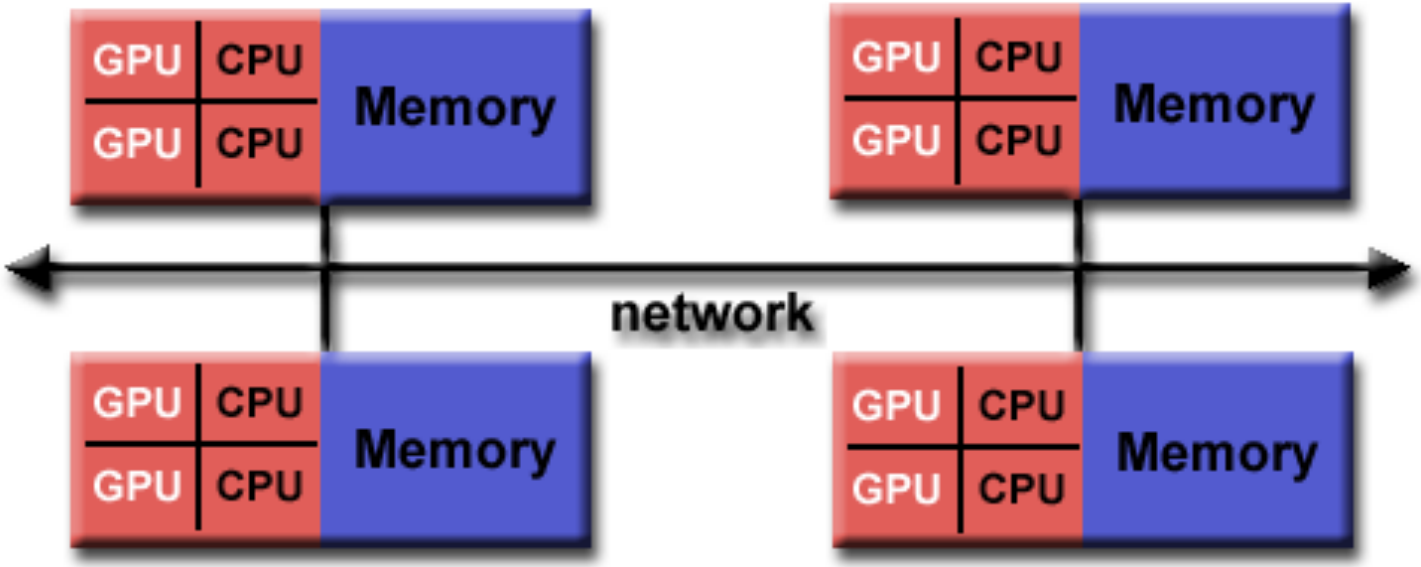
Bir bilgisayar



Bir bilgisayardan daha fazlası



Bir bilgisayardan daha fazlası



Bir bilgisayardan çok daha fazlası

1,024 işlemci
59.7 gigaflop/s

M-5: LOS ALAMOS NATIONAL LAB
No 1. süperbilgisayar, Haziran1993

Bir bilgisayardan çoooook daha fazlası

478.2 Teraflop/s

BLUEGENE/L: LAWRENCE LIVERMORE NATIONAL LABORATORY, No 1. süperbilgisayar, 2004-2007

Neredeyse en süper bilgisayar

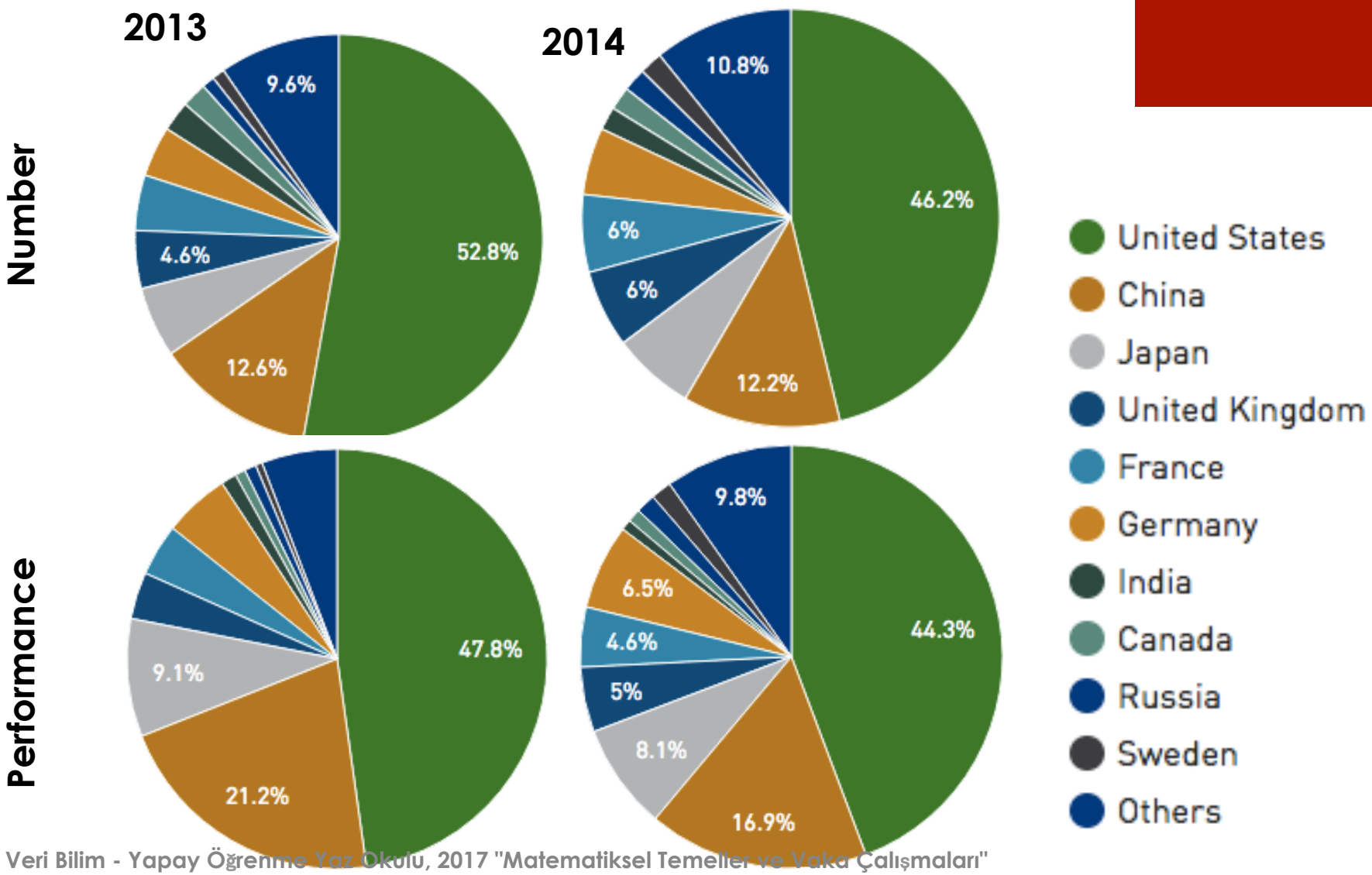
33.86 Petaflop/s (san. kuadrilyon işlem)
3,120,000 çekirdek
Intel Xeon Phi

TIANHE-2 (MILKYWAY-2) : NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY

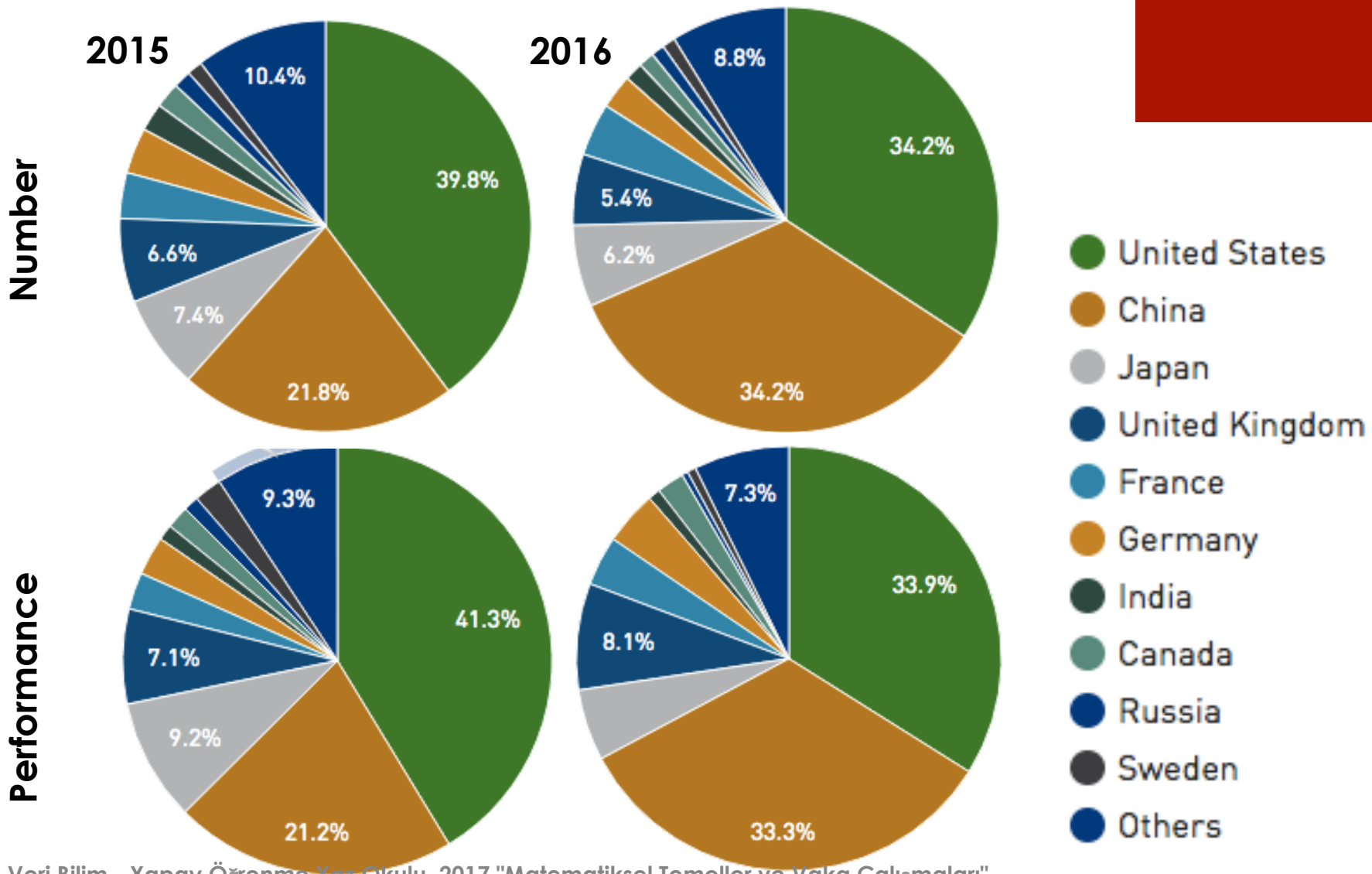
En süper bilgisayar

- **SUNWAY TAIHULIGHT – 125 Petaflop/s**
  - NATIONAL SUPERCOMPUTING CENTER IN WUXI
- Araştırma geliştirme amaçlı
  - Meteoroloji,
  - Derin öğrenme,
  - Imalat ve malzeme bilimleri,
  - Veril analizi

En süper 500 bilgisayar: Ülkeler



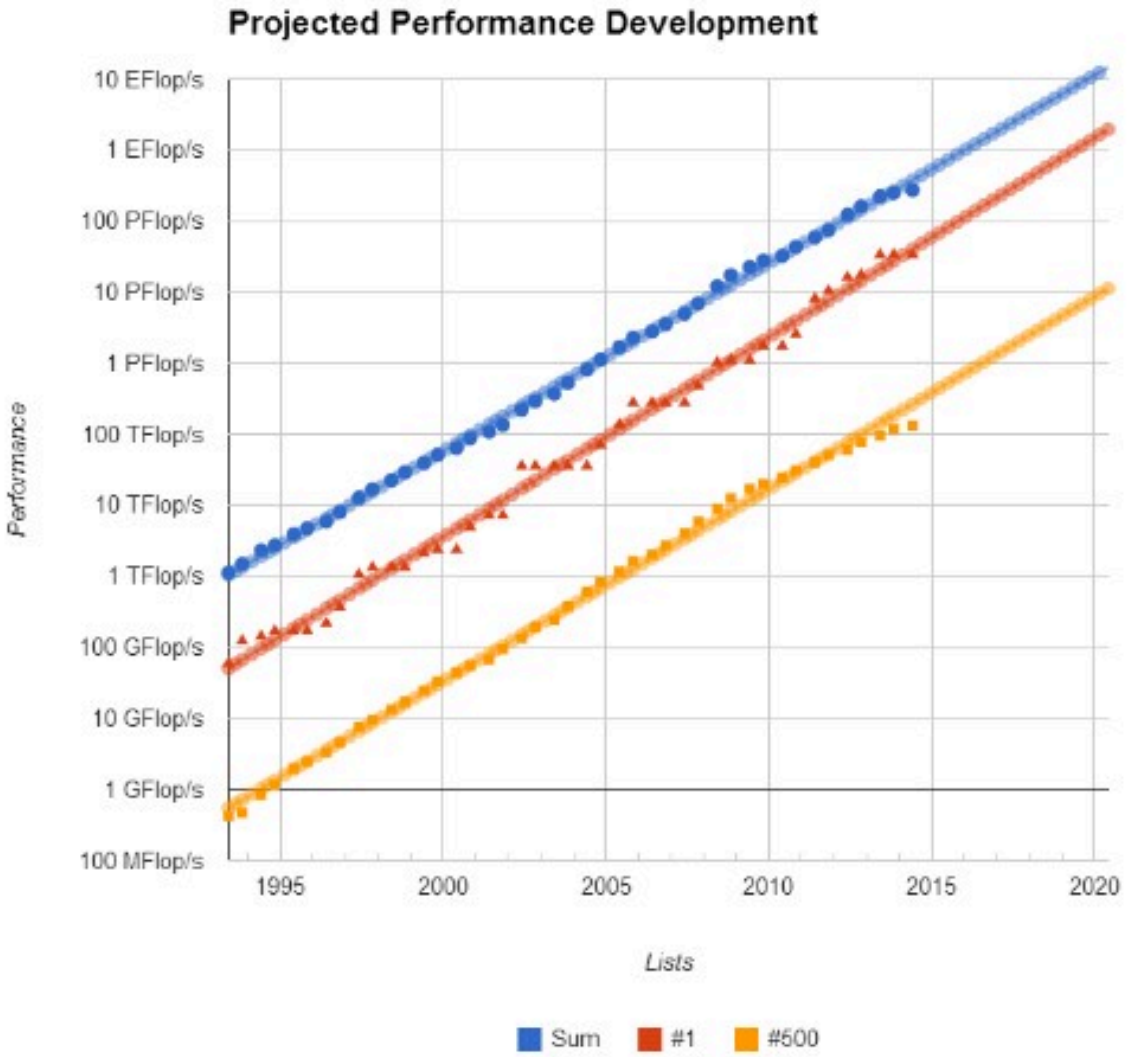
En süper 500 bilgisayar: Ülkeler



HPC Yatırımları

Country	HPC Strategy/Program and Description	Investment Level
United States	National Strategic Computing Initiative (NSCI)	@$320 million/year
China	13th Five-Year Development Plan (Develop Multiple Exascale Systems)	$200 million/year (for next five years)
Japan	Flagship2020 Program	@$200 million/year (for next five years)
European Union	ExaNeSt; PRACE; ETP4HPC	@$1.1 in billion total allocated through 2020 (annual allocations N/A)
India	National Supercomputing Mission	$140 million/year (for five years from 2016-2020)
South Korea	National Supercomputing Act	$20 million/year (for five years from 2016-2020)
Russia	HPC Focus of Medvedev Modernisation Programme	N/A

Süperbilgisayar trendleri

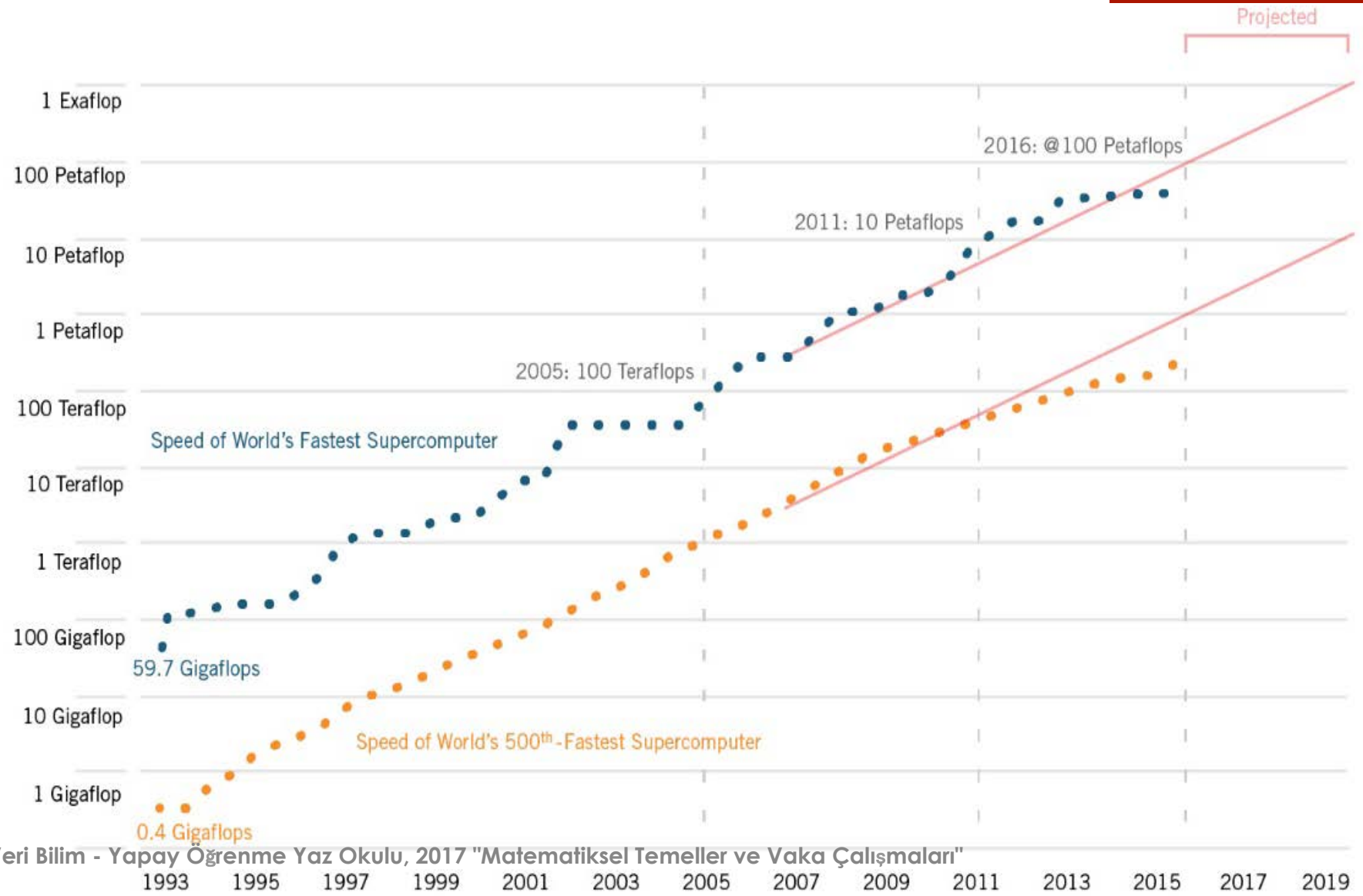


S. Anthony

Exascale: Kilo, Mega, Giga, Tera, Peta, Exa.

- Saniye başına işlem sayısı (FLOPS, TEPS, …)
  - Terascale: 10^{12}, 1,000,000,000,000
  - Petascale: 10^{15}, 1,000,000,000,000,000
  - Exascale: 10^{18}, 1,000,000,000,000,000,000

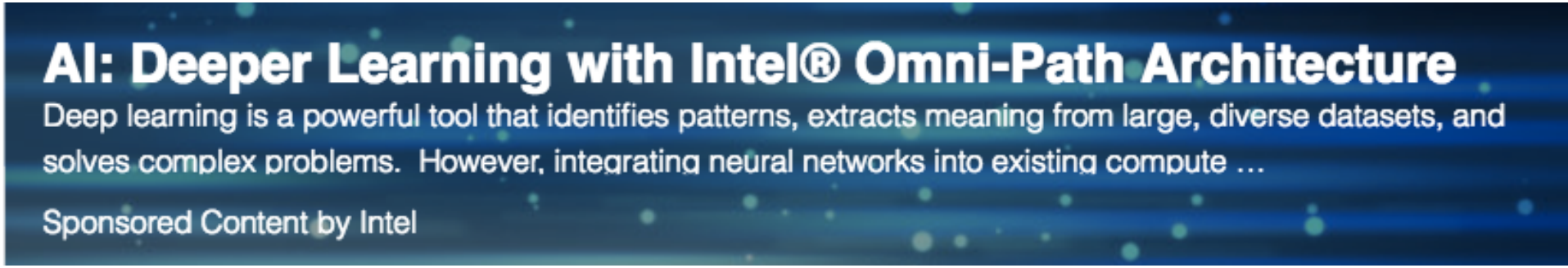
Exascale: Kilo, Mega, Giga, Tera, Peta, Exa.



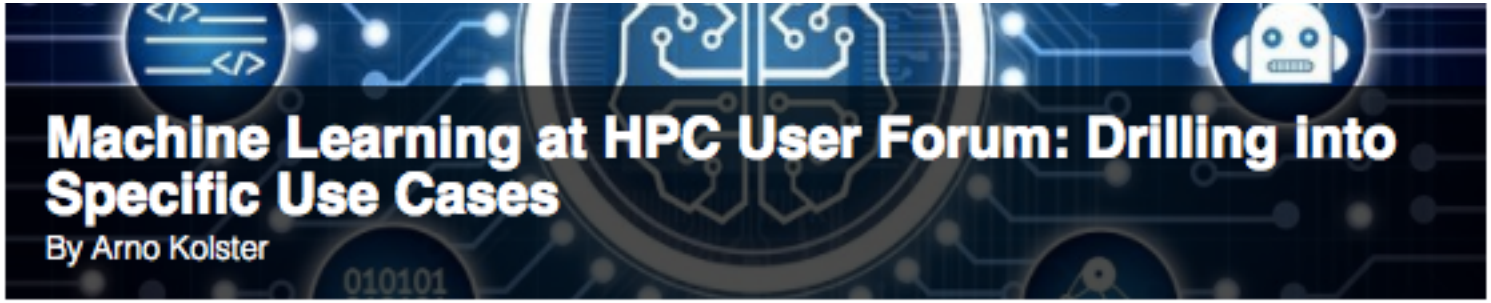
Exascale: Kilo, Mega, Giga, Tera, Peta, Exa.

- Exascale hesaplama Yüksek Başarımlı Hesaplama alanının şu andaki en önemli hedeflerinden birisidir
- Başlangıç: 2008/USA, 2011/Europe
- Hedef: America, 2021-2023, China önümüzdeki yıl (gerçek?)

HPC'de geçen ay



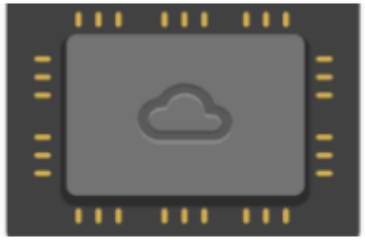
HPC'de geçen ay



September 22, 2017

The 66th HPC User Forum held September 5-7, in Milwaukee, Wisconsin, at the elegant and historic Pfister Hotel, highlighting the 1893 Victorian décor and art of "The Grand Hotel Of The West," contrasted nicely with presentations on the latest trends in modern computing – deep learning, machine learning and AI.

HPC'de geçen ay



Google Cloud Makes Good on Promise to Add Nvidia P100 GPUs

September 21, 2017

Google has taken down the notice on its cloud platform website that says Nvidia Tesla P100s are "coming soon." That's because the search giant has announced the beta launch of the high-end P100 Nvidia Tesla GPUs on t Read more…

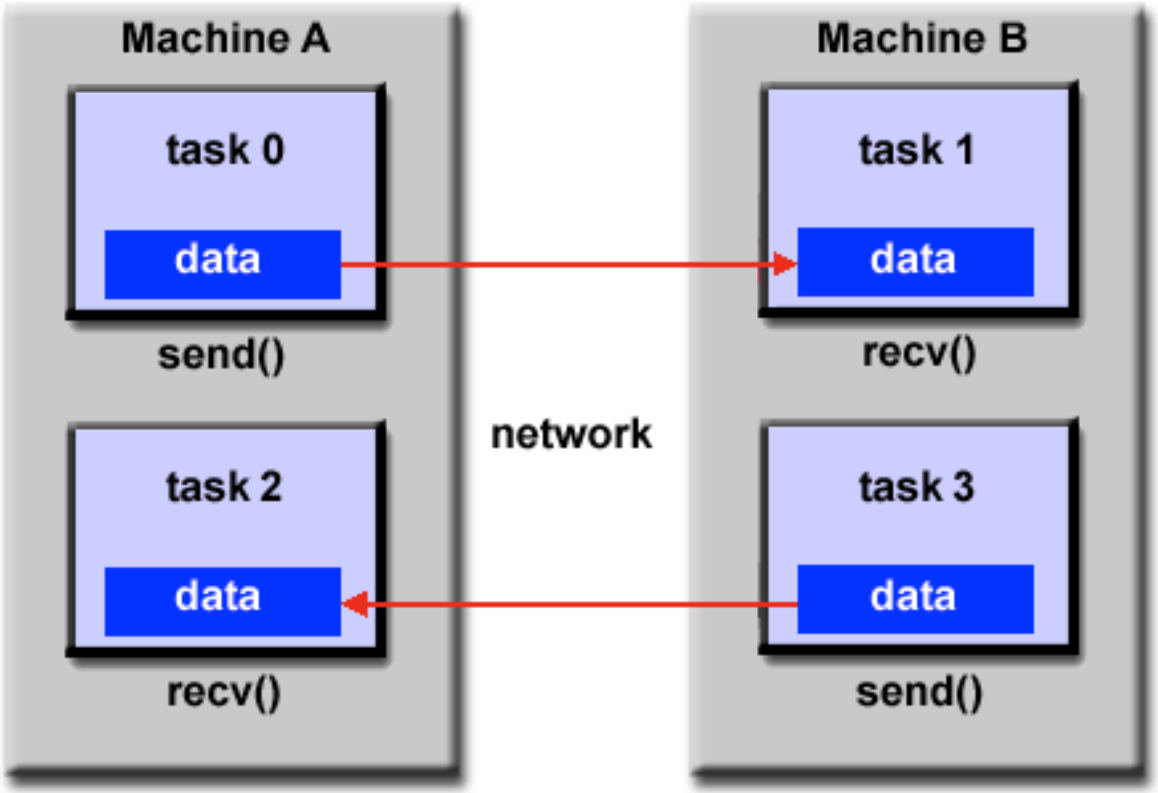
By George Leopold

HPC'de geçen ay

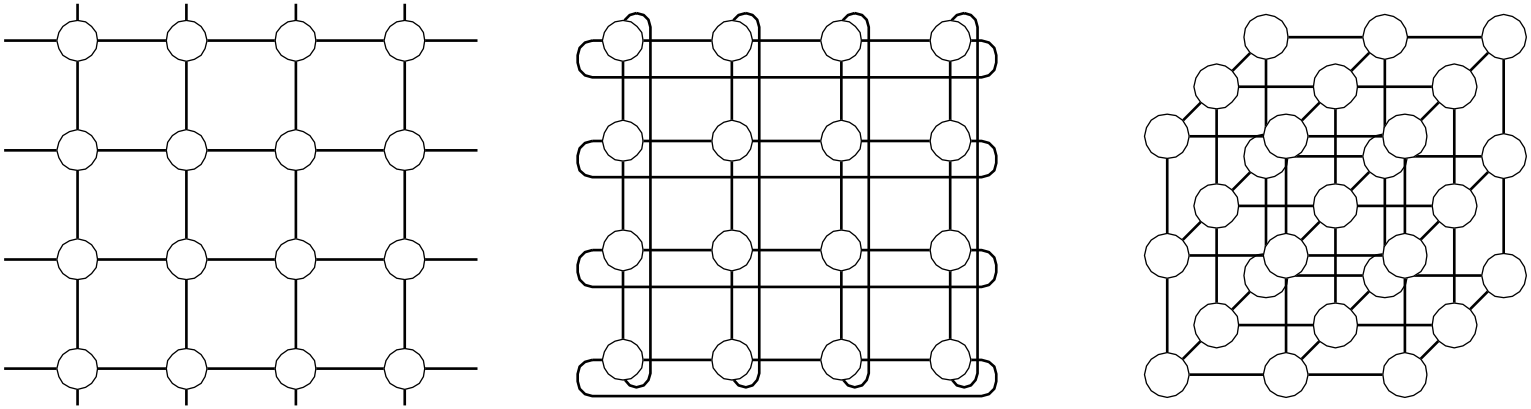
DeepL Anchors Neural Machine Translator at Verne Global's HPC-Optimised Data Center

September 20, 2017

Veri alışverişi

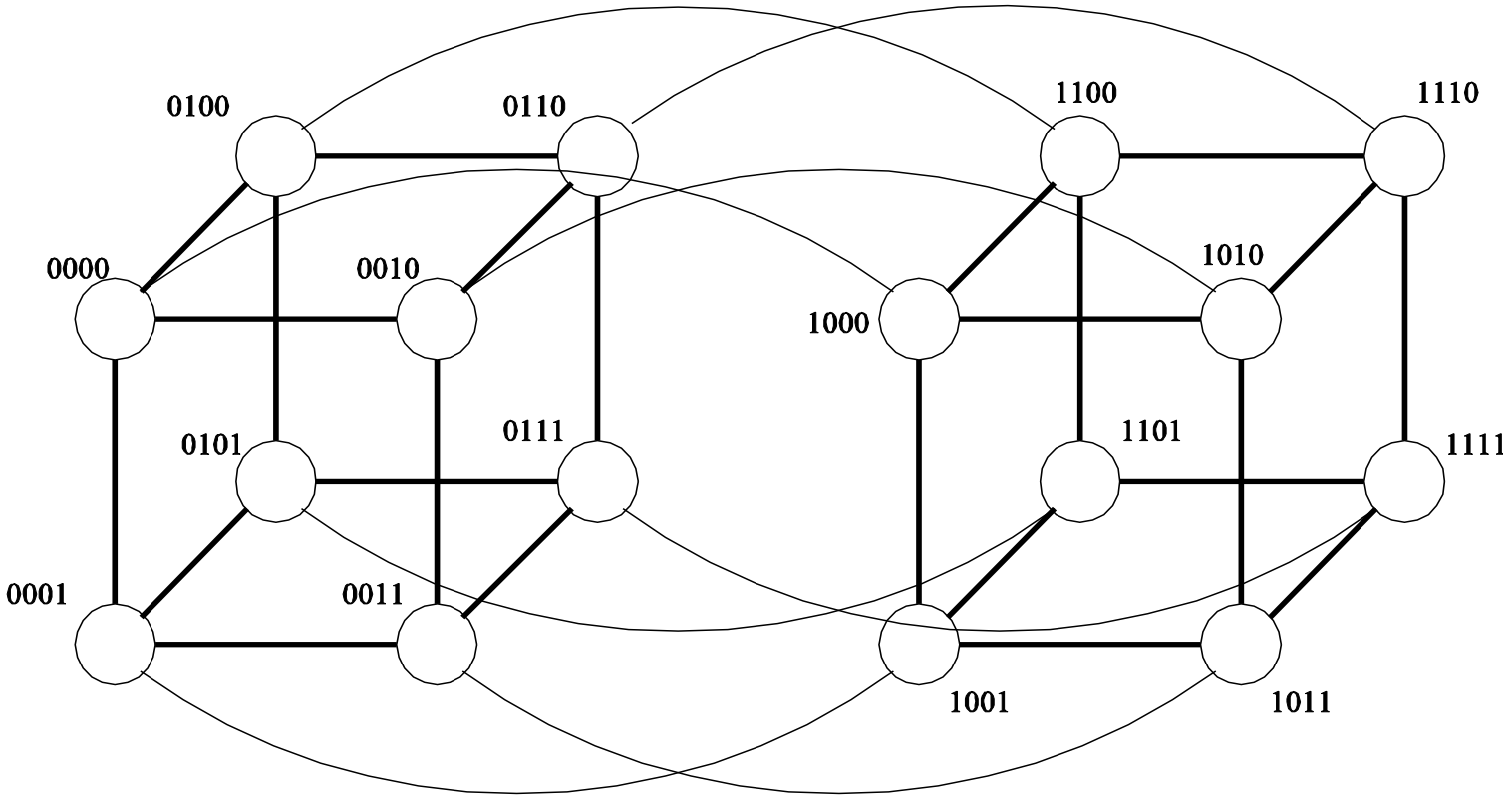


Ağ topolojileri



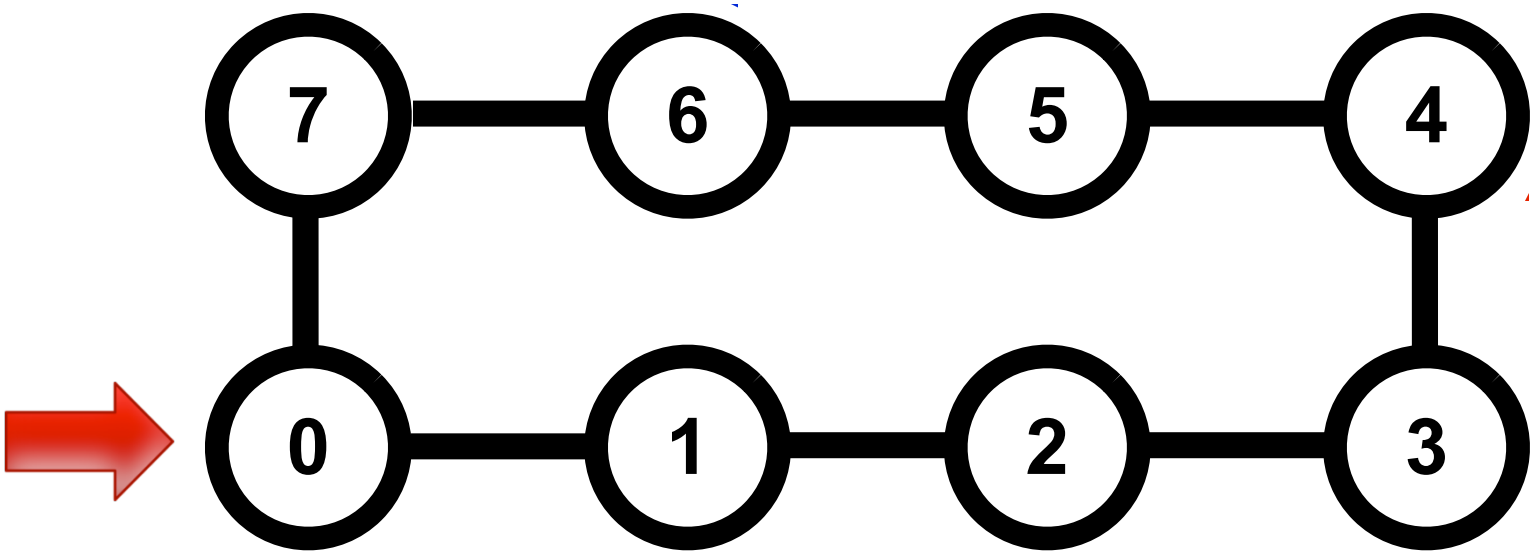
(a) (b) (c)

Ağ topolojileri

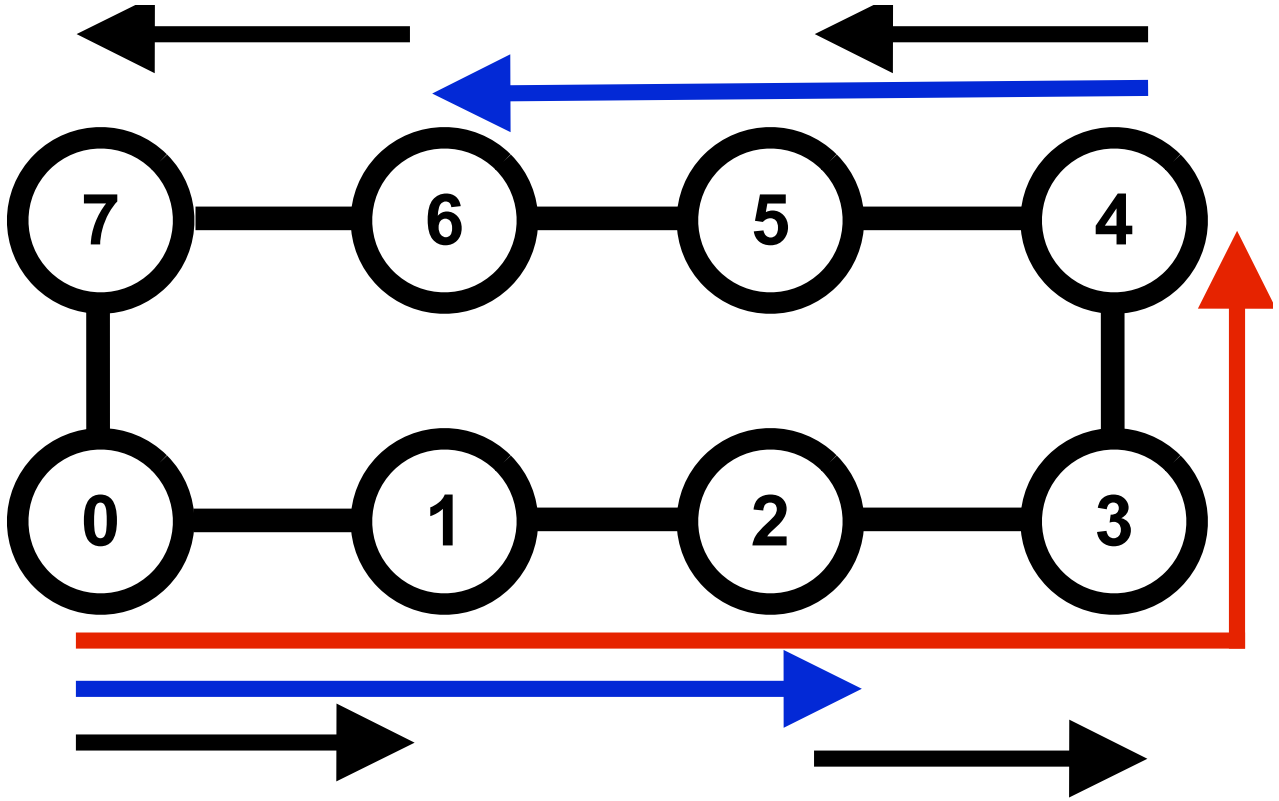


4-D hypercube

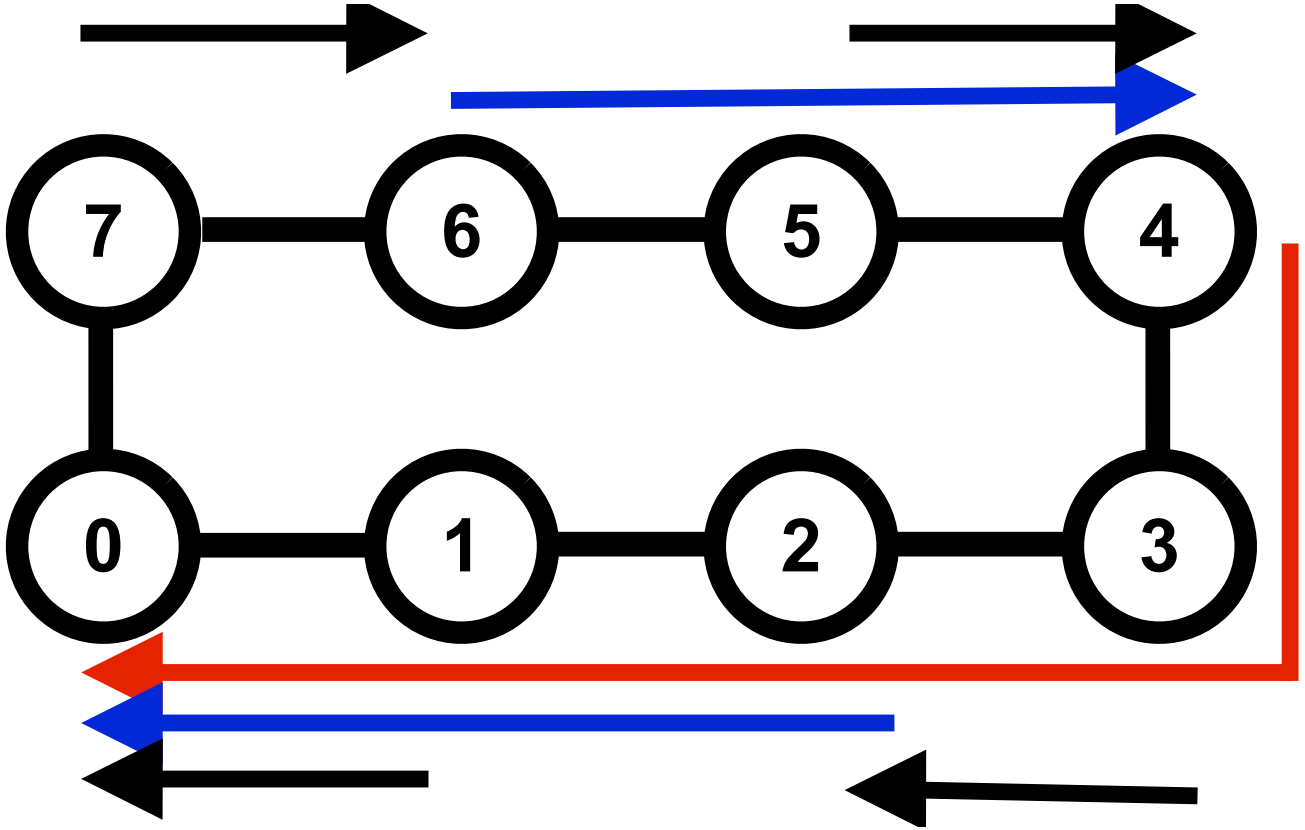
Örnek problem: bir mesajı her yere gönderme



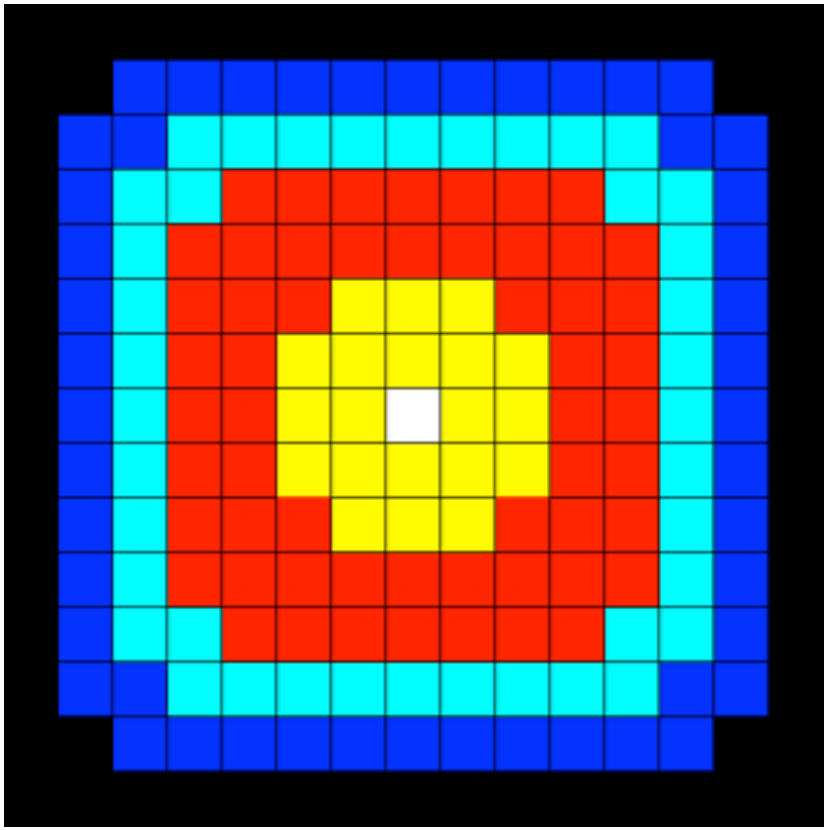
Örnek problem: bir mesajı her yere gönderme

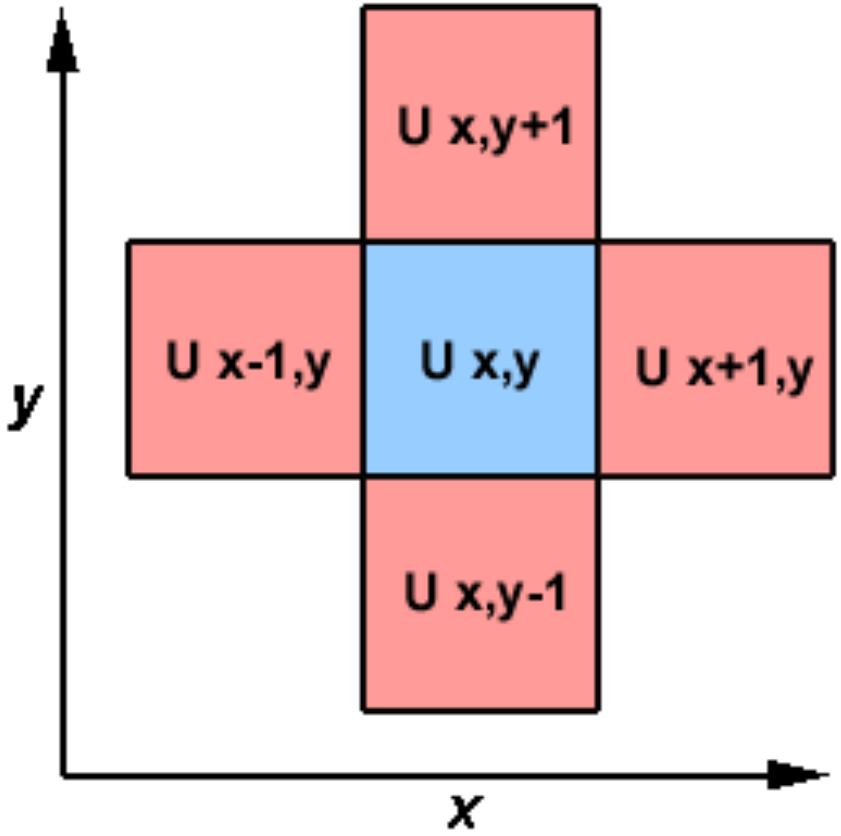


Örnek problem: mesajları bir yerde toplama

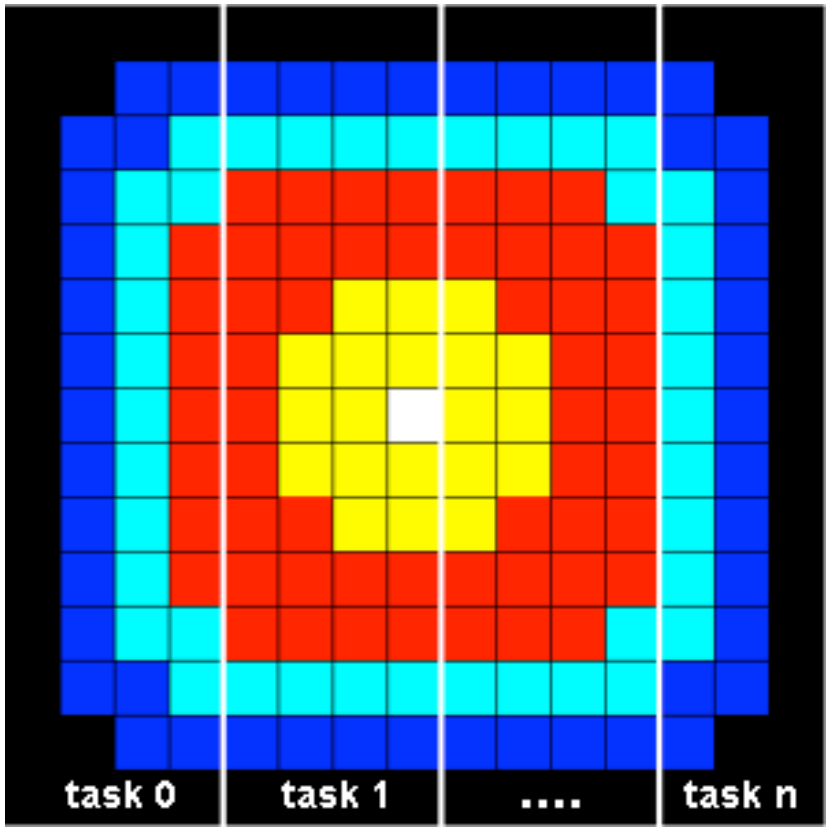


Neden veri gönderiyoruz?

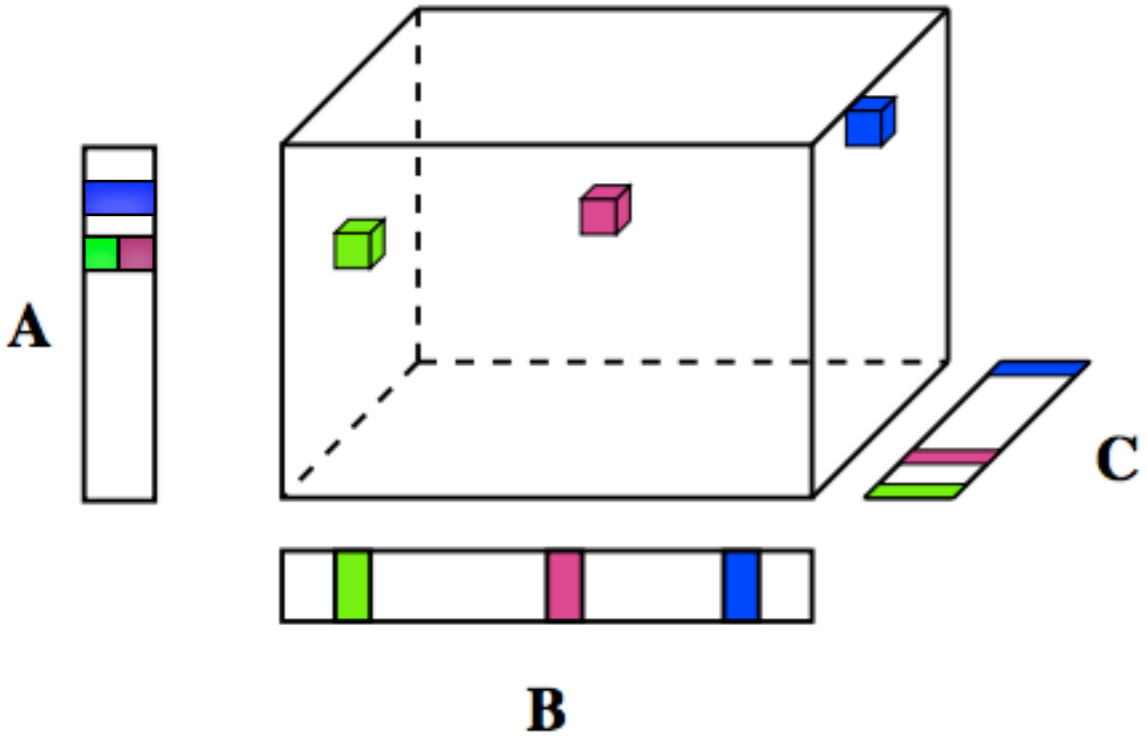




Neden veri gönderiyoruz?



Neden veri gönderiyoruz?



MPI

```
#include <mpi.h>
#include <stdio.h>


int main(int argc, char *argv[])
{
        int npes, myrank;
        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &npes);
        MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
        printf("From process %d out of %d, Hello World!\n",
                myrank, npes);
        MPI_Finalize();
        return 0;
}
```

MPI

```
int MPI_Send(void *buf, int count, MPI_Datatype datatype,
             int dest_pe, int tag, MPI_Comm comm)

int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
             int source_pe, int tag, MPI_Comm comm,
             MPI_Status *status)
```

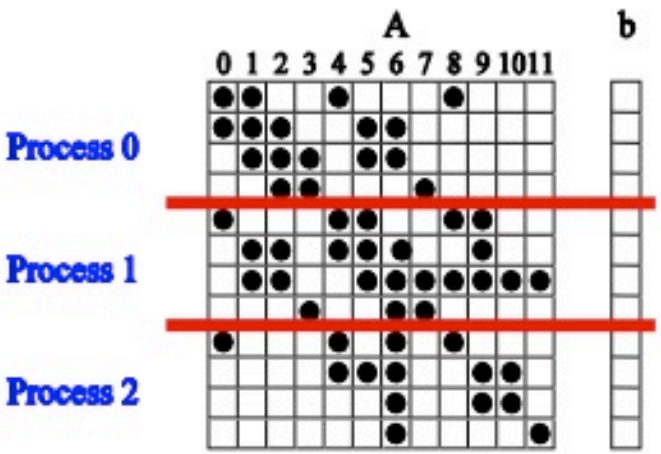
MPI: herkes bir sonraki işlemciye mesaj gönderirse


```
int a[10], b[10], npes, myrank;
MPI_Status status;
...
MPI_Comm_size(MPI_COMM_WORLD, &npes);

MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

MPI_Send(a, 10, MPI_INT, (myrank+1)%npes, 1,
         MPI_COMM_WORLD);

MPI_Recv(b, 10, MPI_INT, (myrank-1+npes)%npes, 1,
          MPI_COMM_WORLD, &status);
...
```

MPI: herkes bir sonraki işlemciye mesaj gönderirse


```
if (myrank%2 == 1) {   // odd processes send first, receive second
        MPI_Send(a, 10, MPI_INT, (myrank+1)%npes, 1,
                MPI_COMM_WORLD);
        MPI_Recv(b, 10, MPI_INT, (myrank-1+npes)%npes, 1,
                MPI_COMM_WORLD, &status);
}
else {                  // even processes receive first, send second
        MPI_Recv(b, 10, MPI_INT, (myrank-1+npes)%npes, 1,
                MPI_COMM_WORLD, &status);
        MPI_Send(a, 10, MPI_INT, (myrank+1)%npes, 1,
                MPI_COMM_WORLD);
}
```

Matrisler ve (Hiper)çizgeler



sparse matrix structure

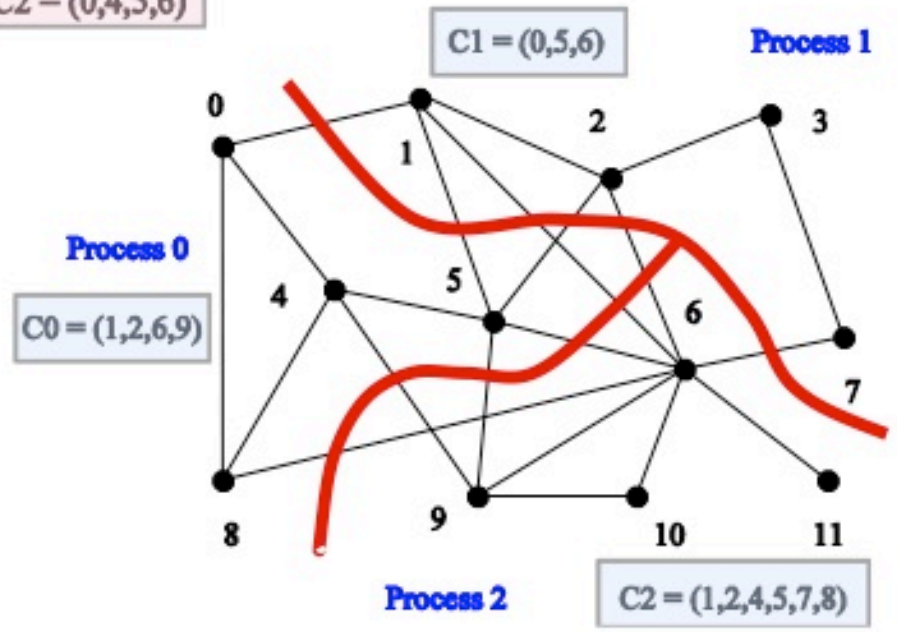
17 items to communicate

C0 = (4,5,6,7,8)

C1 = (0,1,2,3,8,9,10,11)

C2 = (0,4,5,6)

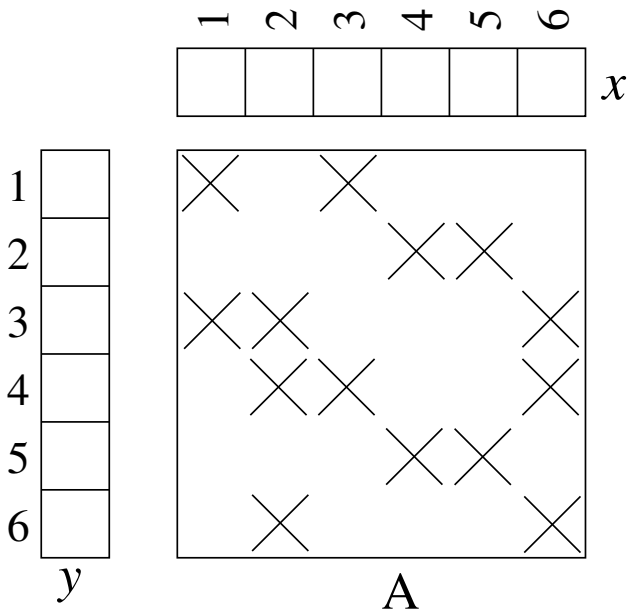
13 items to communicate



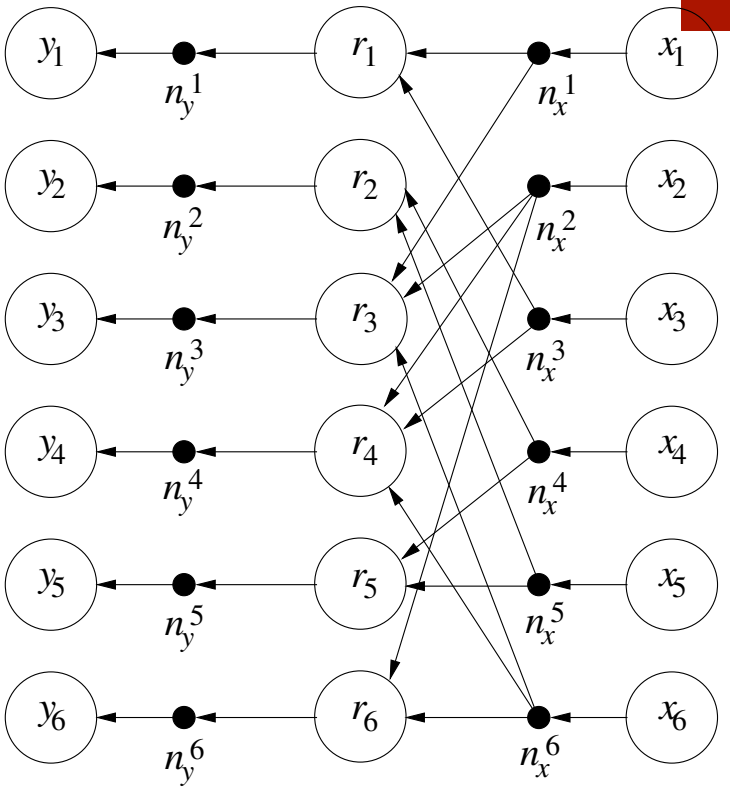
mapping

partitioning

Örnek problem: SpMV

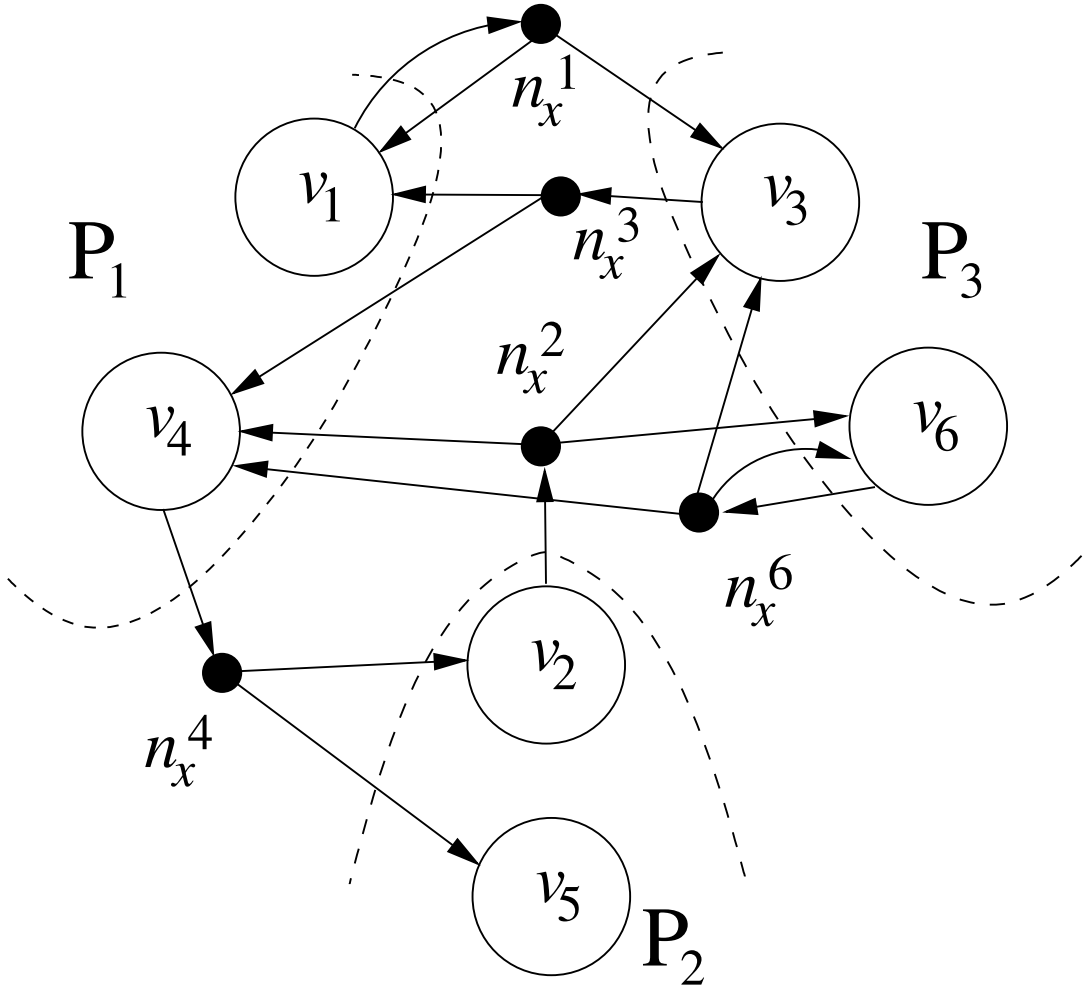


(a) Sample matrix-vector multiplication



(b) Directed hypergraph model

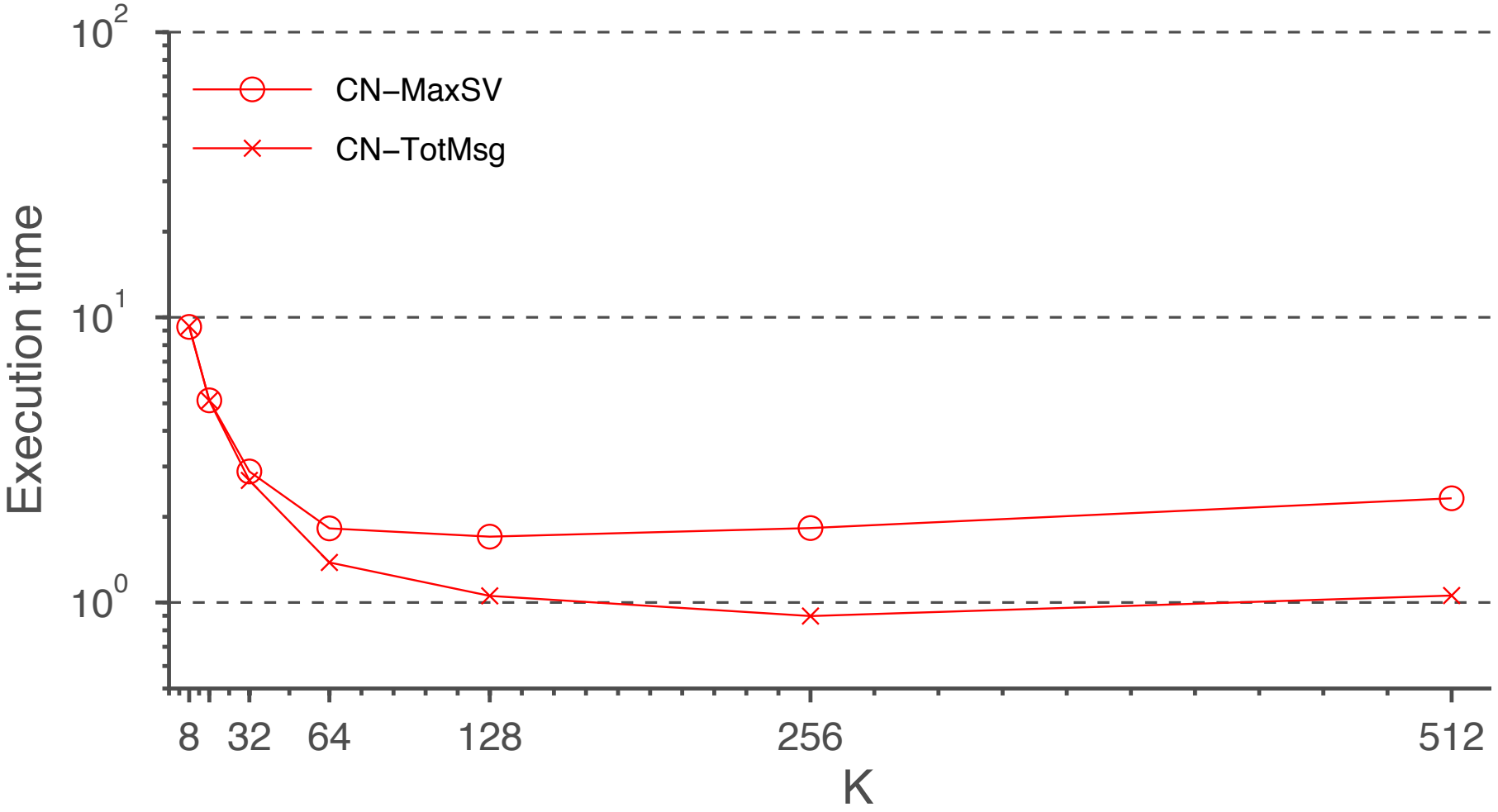
Örnek problem: SpMV



Kapasite metrikleri
- Gönderilen
- Alınan **Byte**
- Toplam

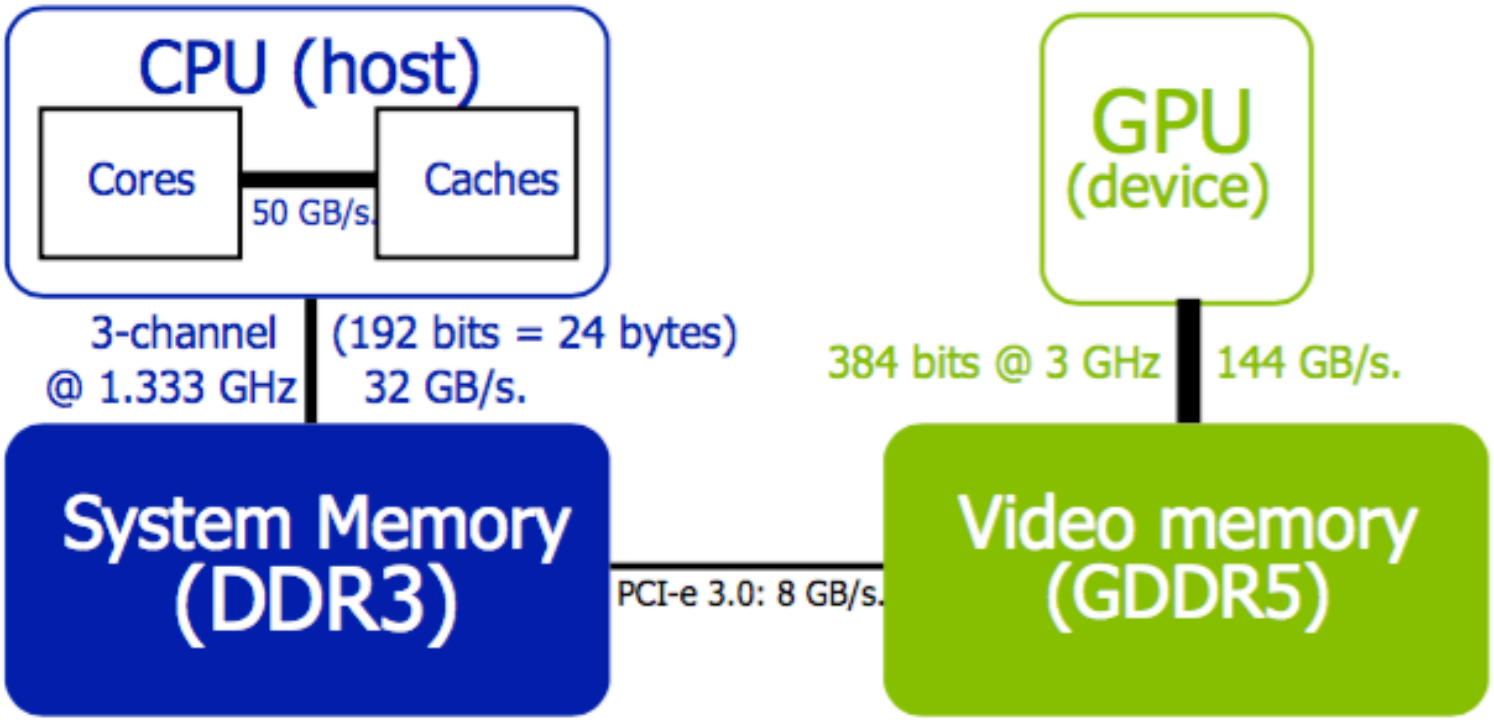
Hız metrikleri
- Gönderilen
- Alınan **Mesaj**
- Toplam

Örnek problem: SpMV

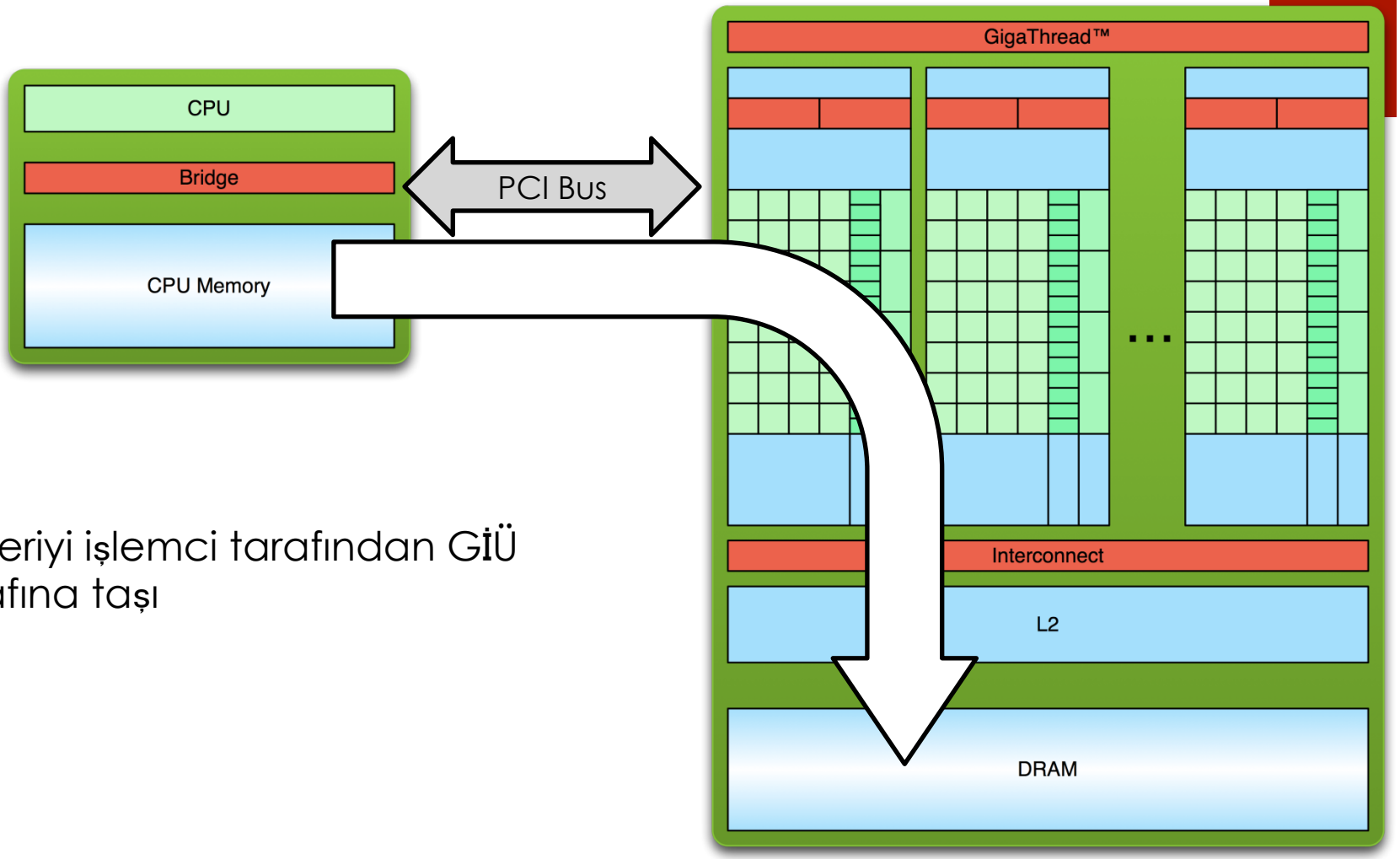


Grafik İşleme Üniteleri

Grafik İşleme Üniteleri

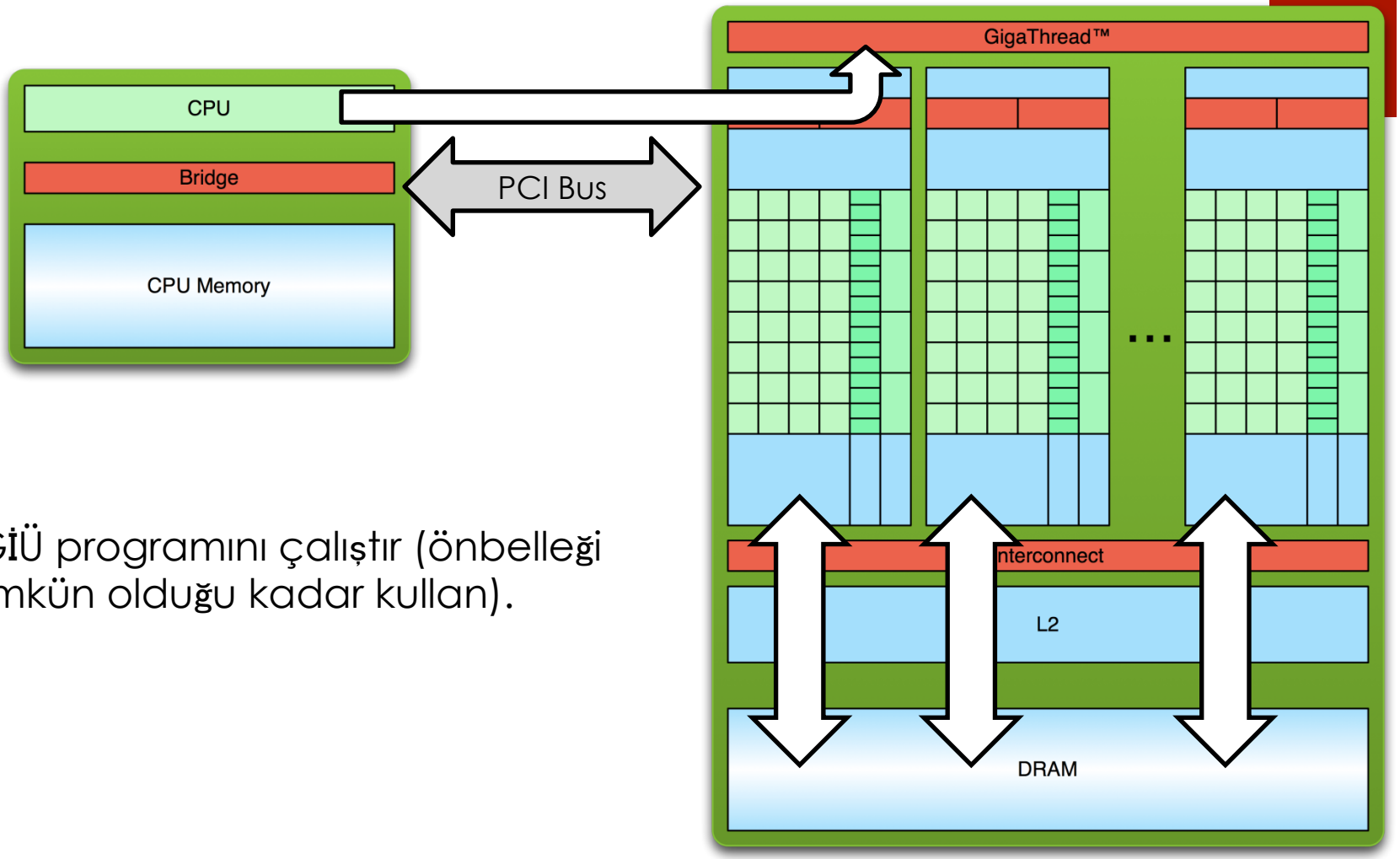


(Basitçe) GİÜ kullanımı



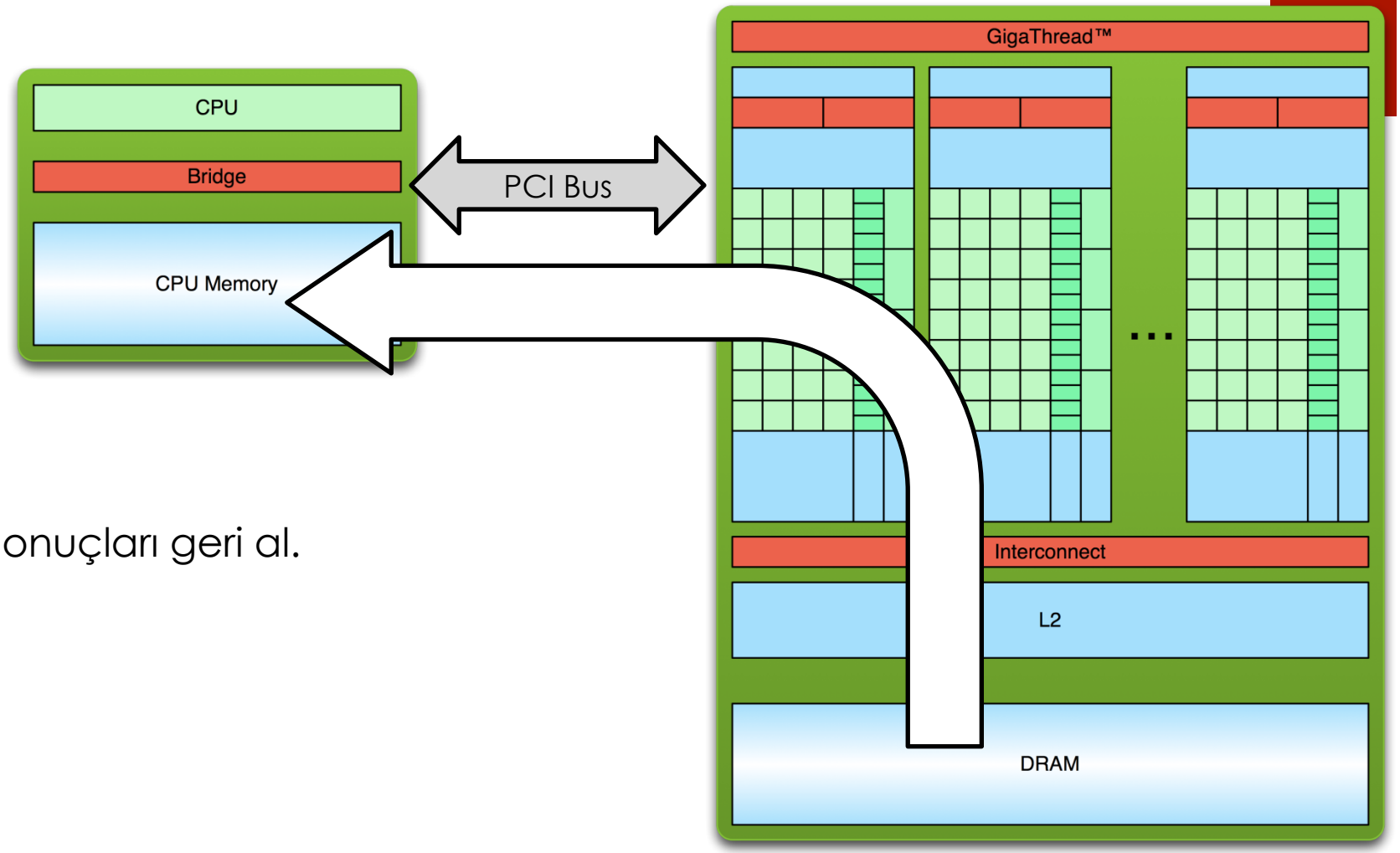
1- Veriyi işlemci tarafından GİÜ tarafına taşı

(Basitçe) GİÜ kullanımı



2- GİÜ programını çalıştır (önbelleği mümkün olduğu kadar kullan).

(Basitçe) GİÜ kullanımı



3- Sonuçları geri al.

GİÜ: Özellikler

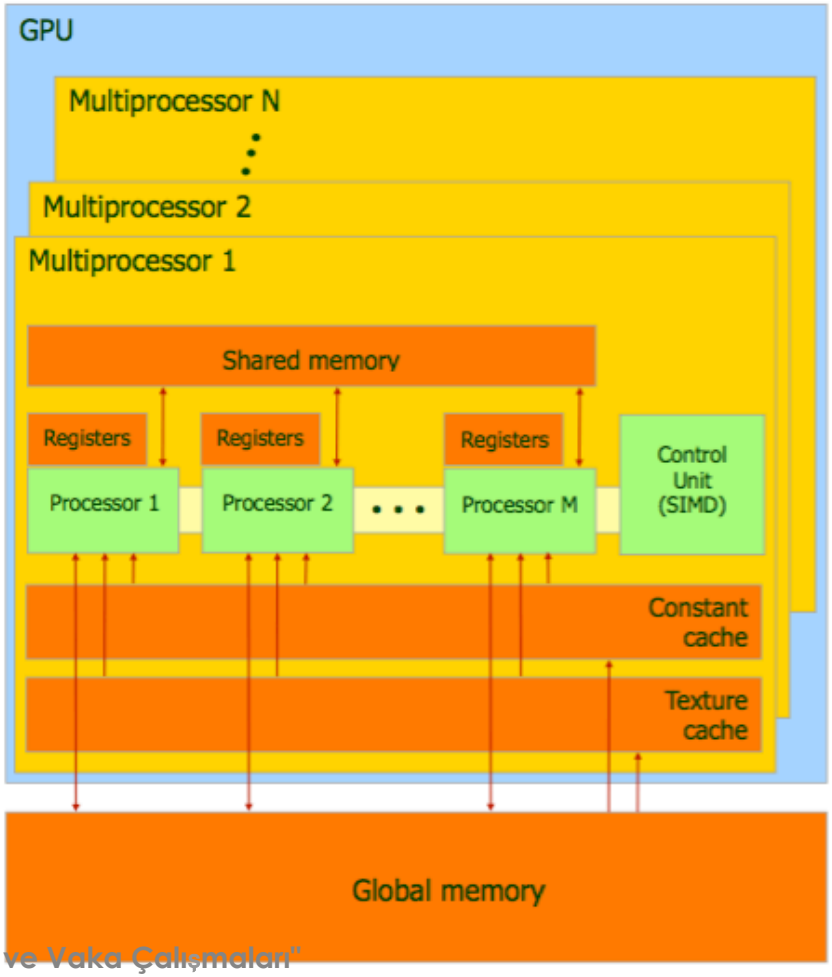
Öznitelikler	Tesla K80	Tesla K40
GPU	2x Kepler GK210	1 Kepler GK110B
Peak double precision float. point performance	2.91 Tflops (GPU Boost Clocks) 1.87 Tflops (Base Clocks)	1.66 Tflops (GPU Boost Clocks) 1.43 Tflops (Base Clocks)
Peak single precision float. point performance	8.74 Tflops (GPU Boost Clocks) 5.6 Tflops (Base Clocks)	5 Tflops (GPU Boost Clocks) 4.29 Tflops (Base Clocks)
Memory bwidth (ECC off)[2]	480 GB/sec (240 GB/sec per GPU)	288 GB/sec
Memory size (GDDR5)	24 GB (12GB per GPU)	12 GB
CUDA cores	4992 (2496 per GPU)	2880

Grafik İşleme Üniteleri

SPECIFICATIONS	
GPU Architecture	NVIDIA Pascal
NVIDIA CUDA® Cores	3584
Double-Precision Performance	5.3 TeraFLOPS
Single-Precision Performance	10.6 TeraFLOPS
Half-Precision Performance	21.2 TeraFLOPS
GPU Memory	16 GB CoWoS HBM2
Memory Bandwidth	732 GB/s
Interconnect	NVIDIA NVLink
Max Power Consumption	300 W

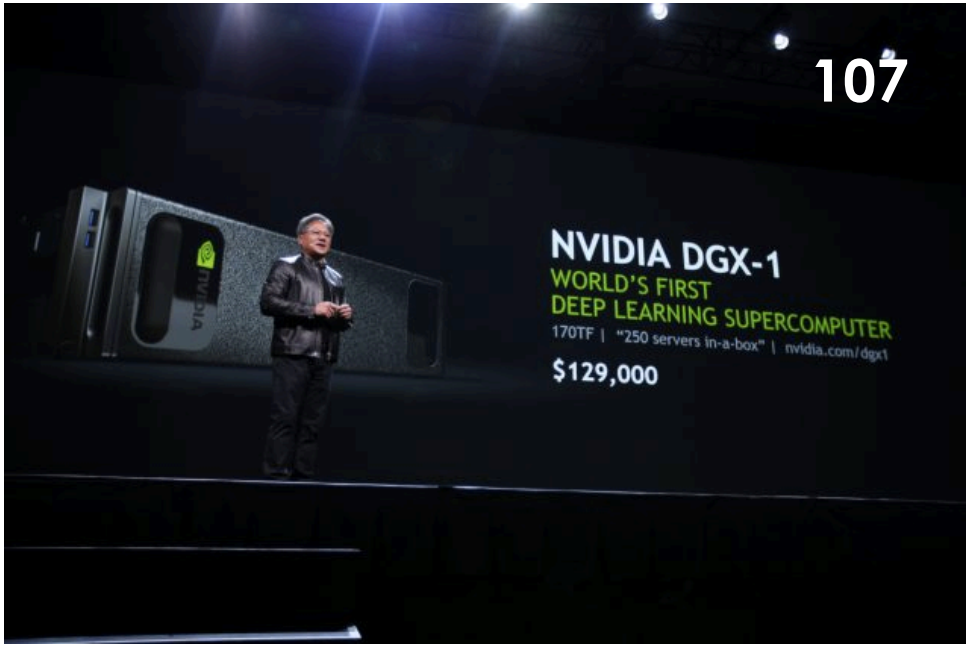
NVIDIA TESLA P100

7000$



Grafik İşleme Üniteleri

SYSTEM SPECIFICATIONS	
GPUs	8x Tesla P100
TFLOPS (GPU FP16 / CPU FP32)	170/3
GPU Memory	16 GB per GPU
CPU	Dual 20-core Intel® Xeon® E5-2698 v4 2.2 GHz
NVIDIA CUDA® Cores	28672
System Memory	512 GB 2133 MHz DDR4
Storage	4x 1.92 TB SSD RAID 0
Network	Dual 10 GbE, 4 IB EDR
Software	Ubuntu Server Linux OS DGX-1 Recommended GPU Driver
System Weight	134 lbs
System Dimensions	866 D x 444 W x 131 H (mm)
Packing Dimensions	1180 D x 730 W x 284 H (mm)
Maximum Power Requirements	3200W
Operating Temperature Range	10 - 30°C



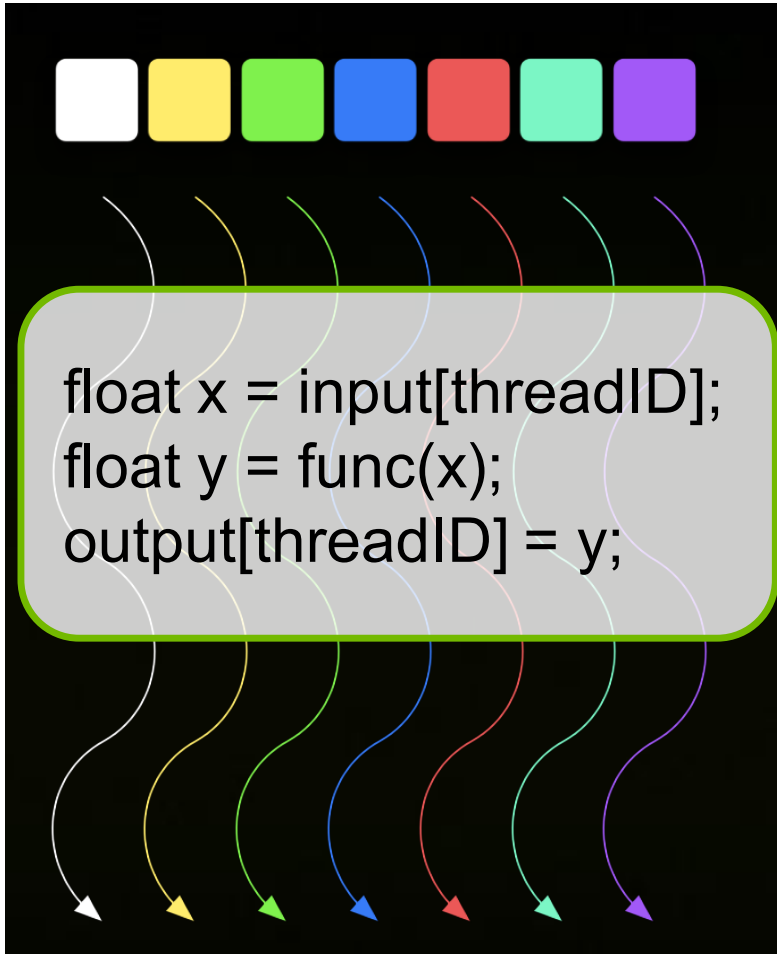
GİÜ: Programlama - CUDA

```
__global__
void hello(char *a, int *b) {
        a[threadIdx.x] += b[threadIdx.x];
}
```

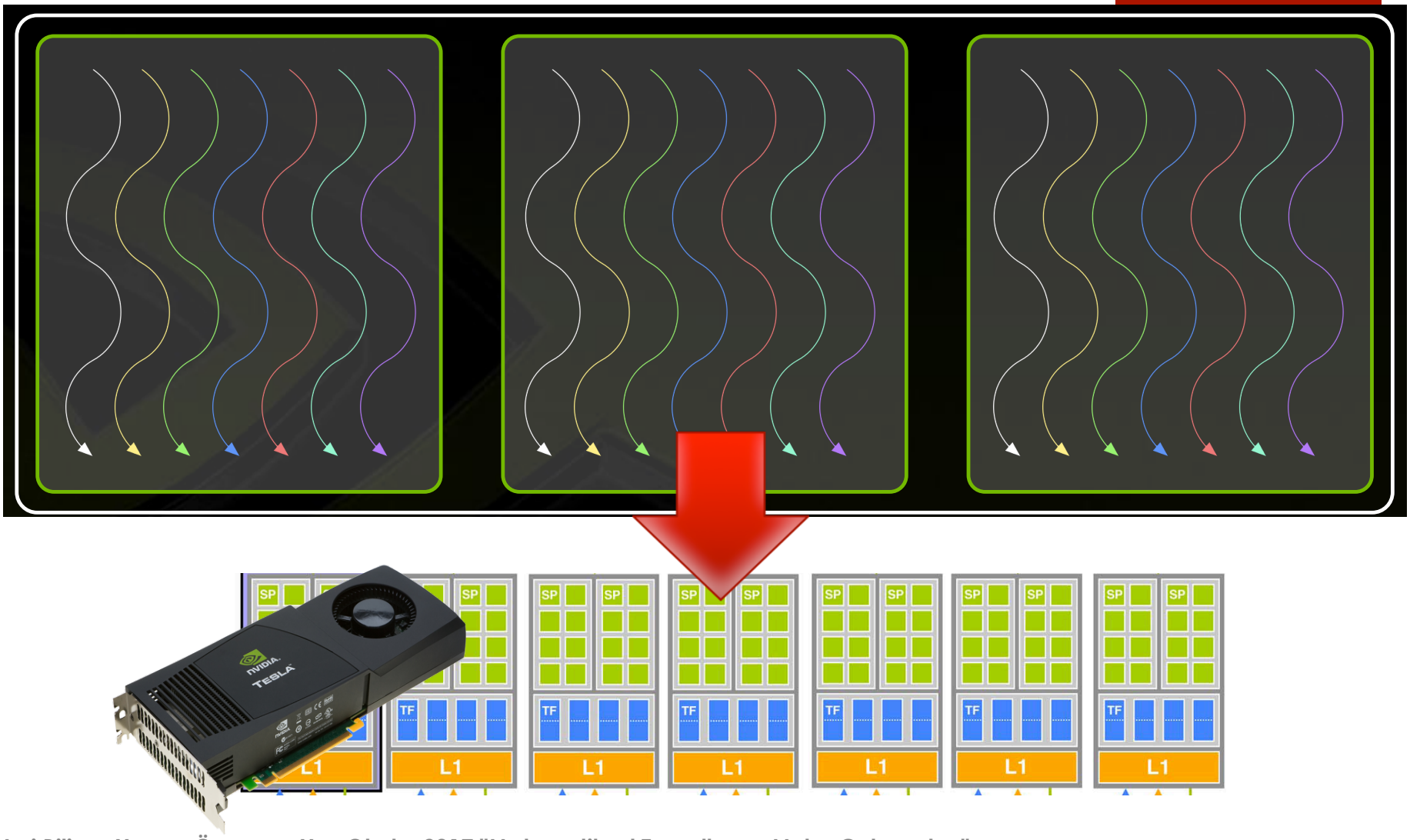
GİÜ: Programlama - CUDA

```
cudaMalloc( (void**)&bd, isize );
cudaMemcpy( ad, a, csize,cudaMemcpyHostToDevice );
cudaMemcpy( bd, b, isize, cudaMemcpyHostToDevice );

dim3 dimBlock( blocksize, 1 );
dim3 dimGrid( 1, 1 );
hello<<<dimGrid, dimBlock>>>(ad, bd);
cudaMemcpy(a, ad, csize, cudaMemcpyDeviceToHost );
cudaFree( ad );
cudaFree( bd );
```

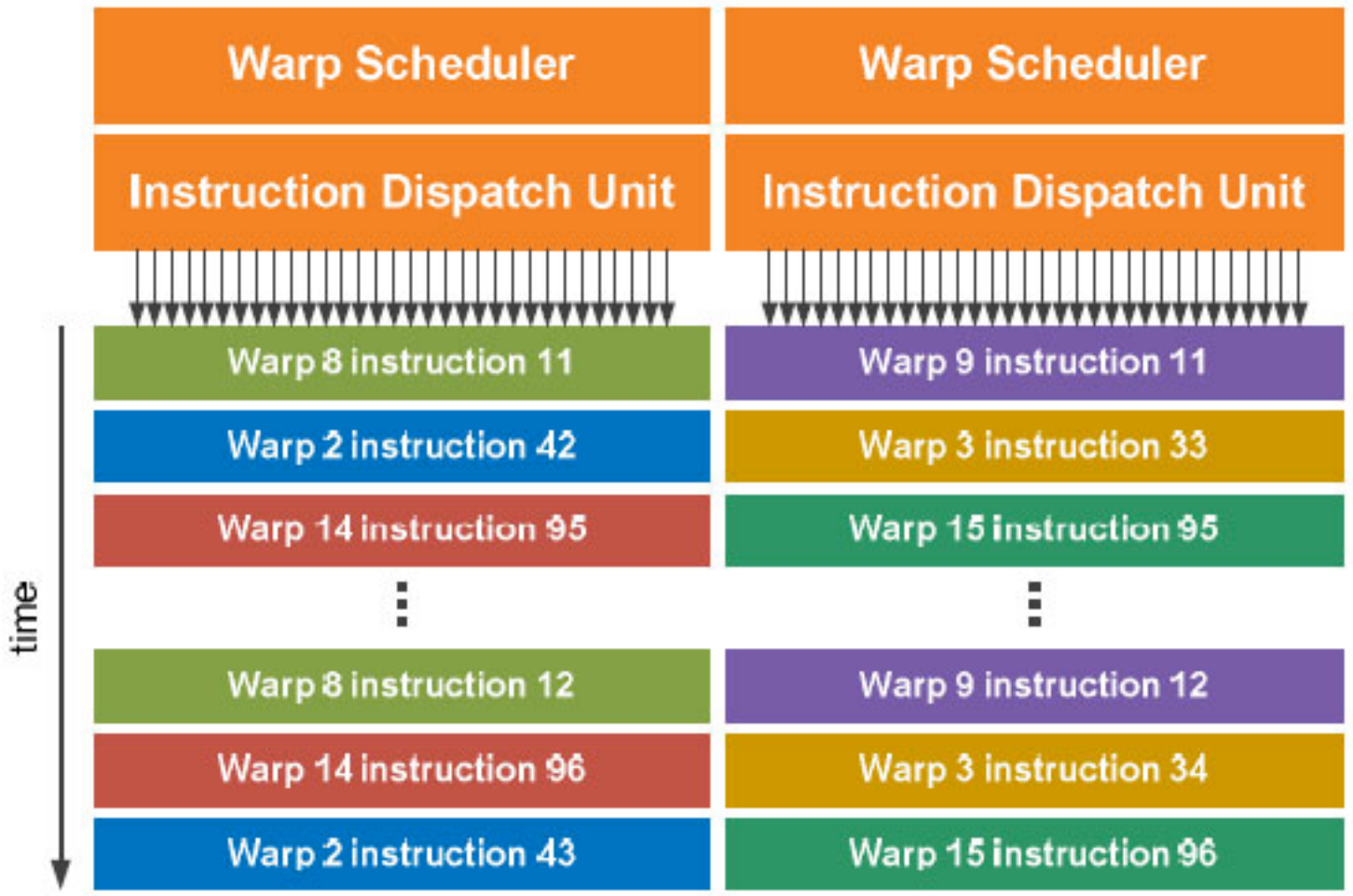
GİÜ: Programlama – CUDA Kernel



GİÜ: Programlama – CUDA Kernelden Bloklara



GİÜ: Programlama – CUDA Bloklardan Warplara



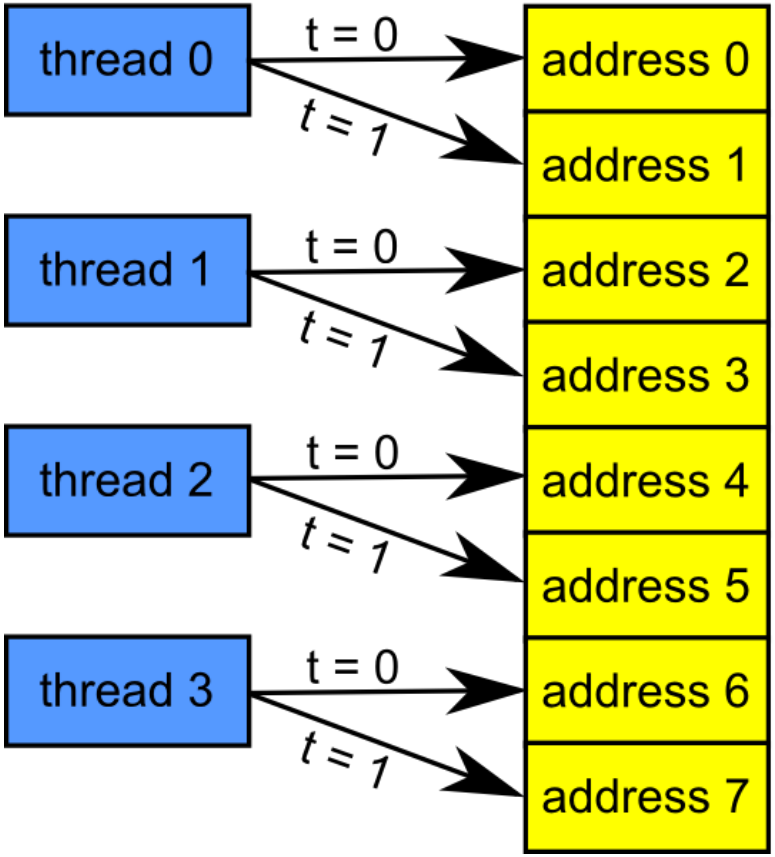
GİÜ: Programlama – CUDA Bloklardan Warplara

En önemli üç nokta

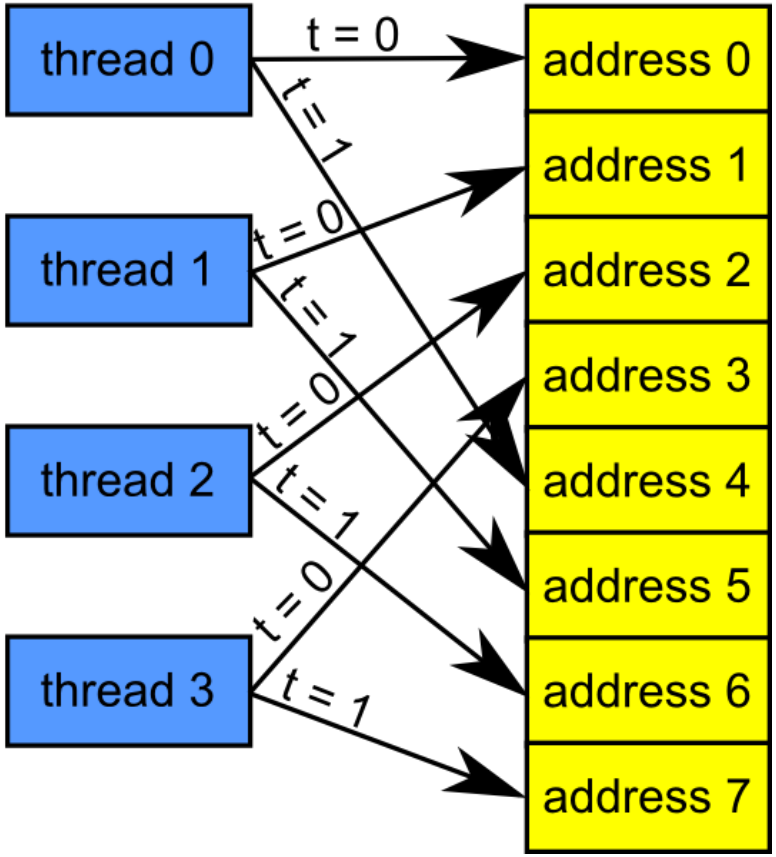
- Warp içinde dallanan kodlar olmamalı
- Warp içindeki ipliklere eşit yük paylaşımı
- Warp içindeki ipliklerin kompakt hafıza erişimi

GİÜ: Programlama – CUDA Bloklardan Warplara

traditional multi-core
optimal memory access pattern



many-core GPU
optimal memory access pattern



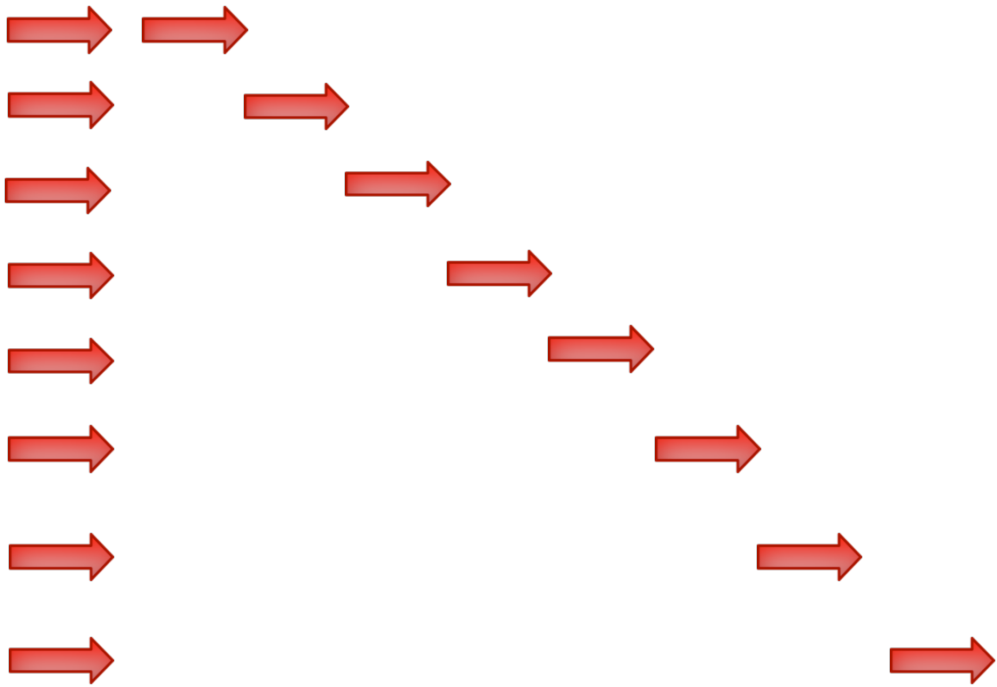
GİÜ: Programlama – CUDA Bloklardan Warplara

Hangisi bantgenişliğini daha verimli kullanır?

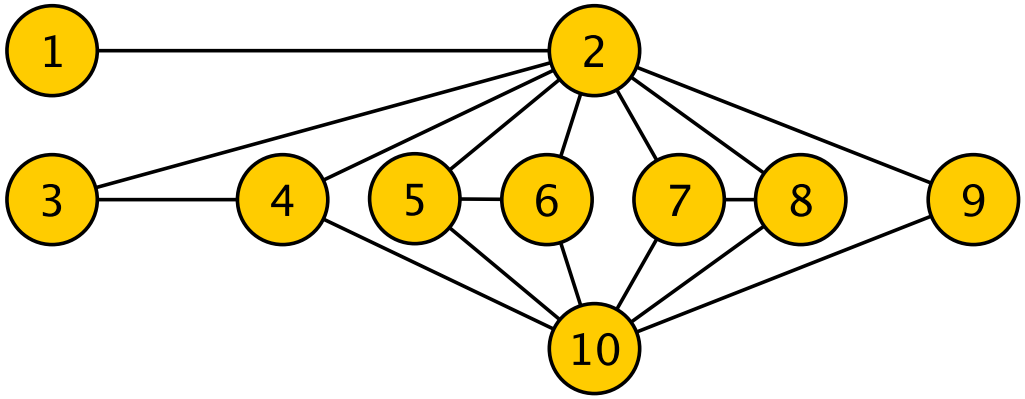
```
int a = input[i];

int b = input[2*i];
```

Örnek program: switch

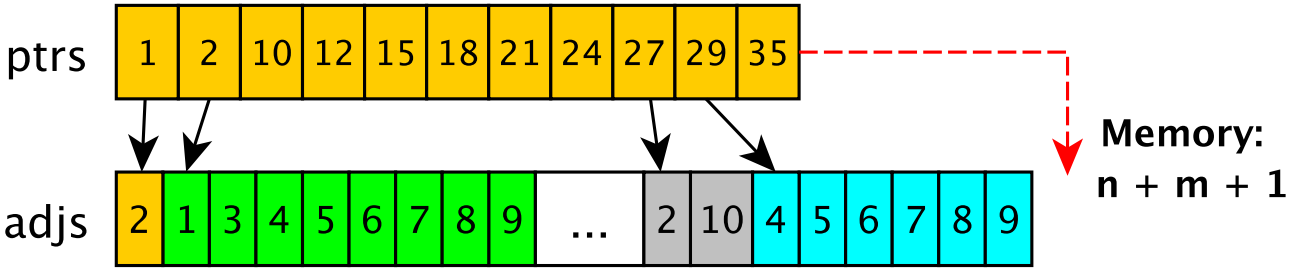
```
int a = threadID.x;
switch(a) {
        case(1): doA();
        case(2): doB();
        case(3): doC();
        case(4): doD();
        case(5): doE();
        case(6): doF();
        case(7): doG();
        case(8): doH();
}
```



Örnek program: PageRank, BFS vs.

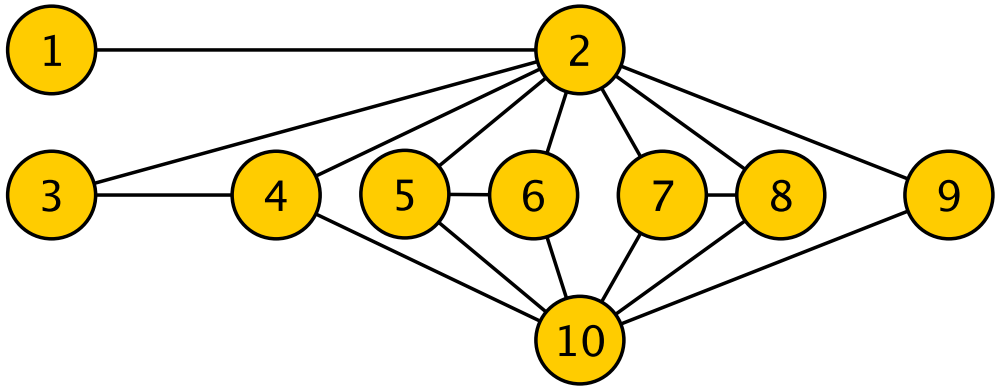


(a) A toy graph G

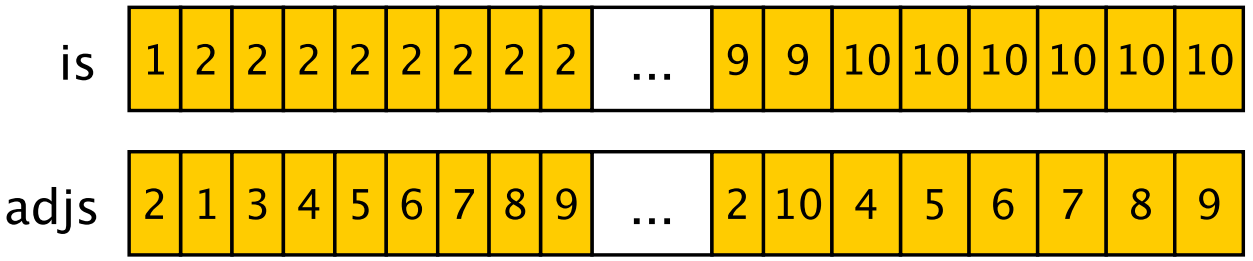


(b) CSR representation of G

Örnek program: PageRank, BFS vs.

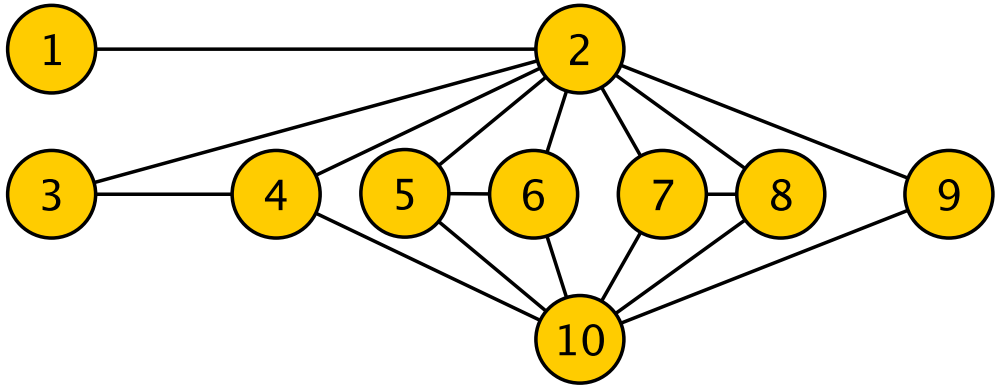


(a) A toy graph G

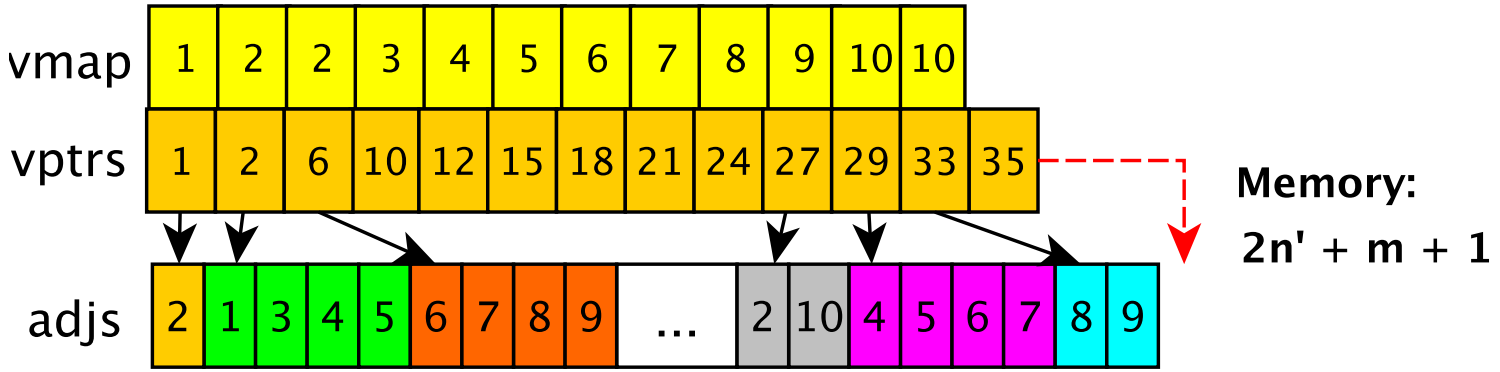


(c) COO representation of G

Örnek program: PageRank, BFS vs.

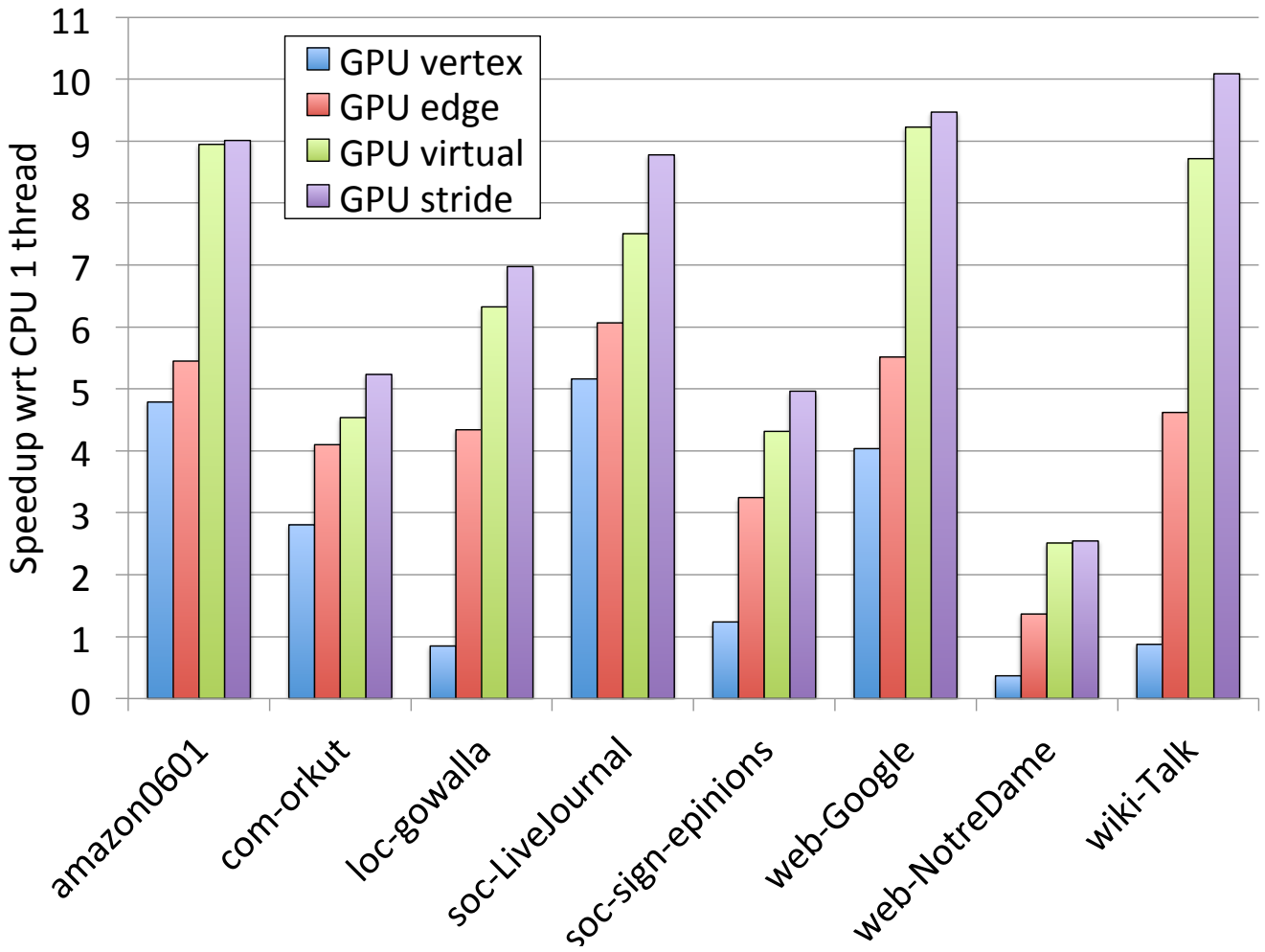


(a) A toy graph G

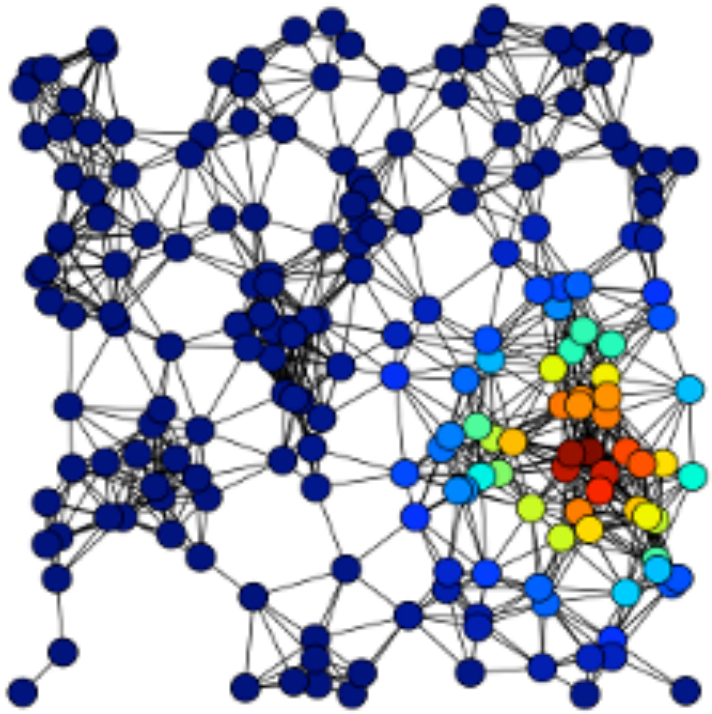


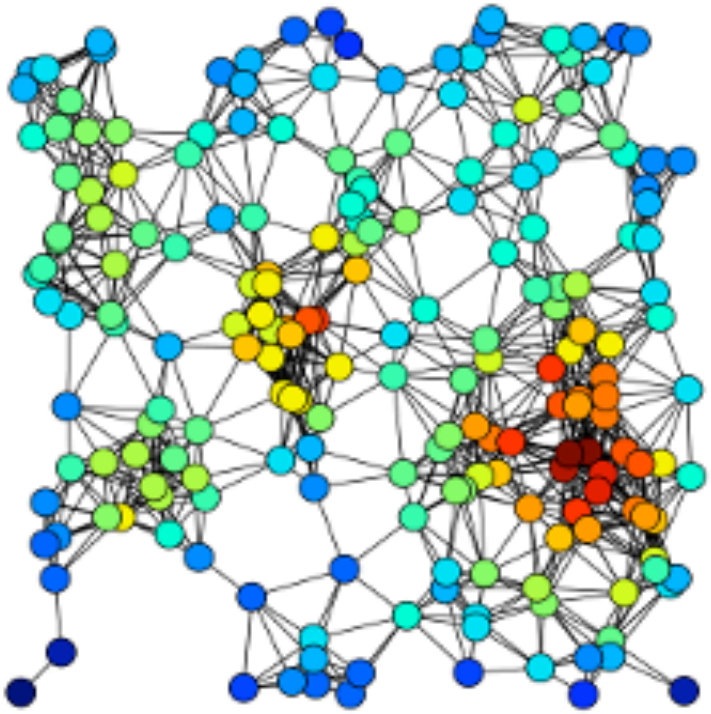
(d) Virtual-CSR representation of G

Örnek program: PageRank, BFS vs.

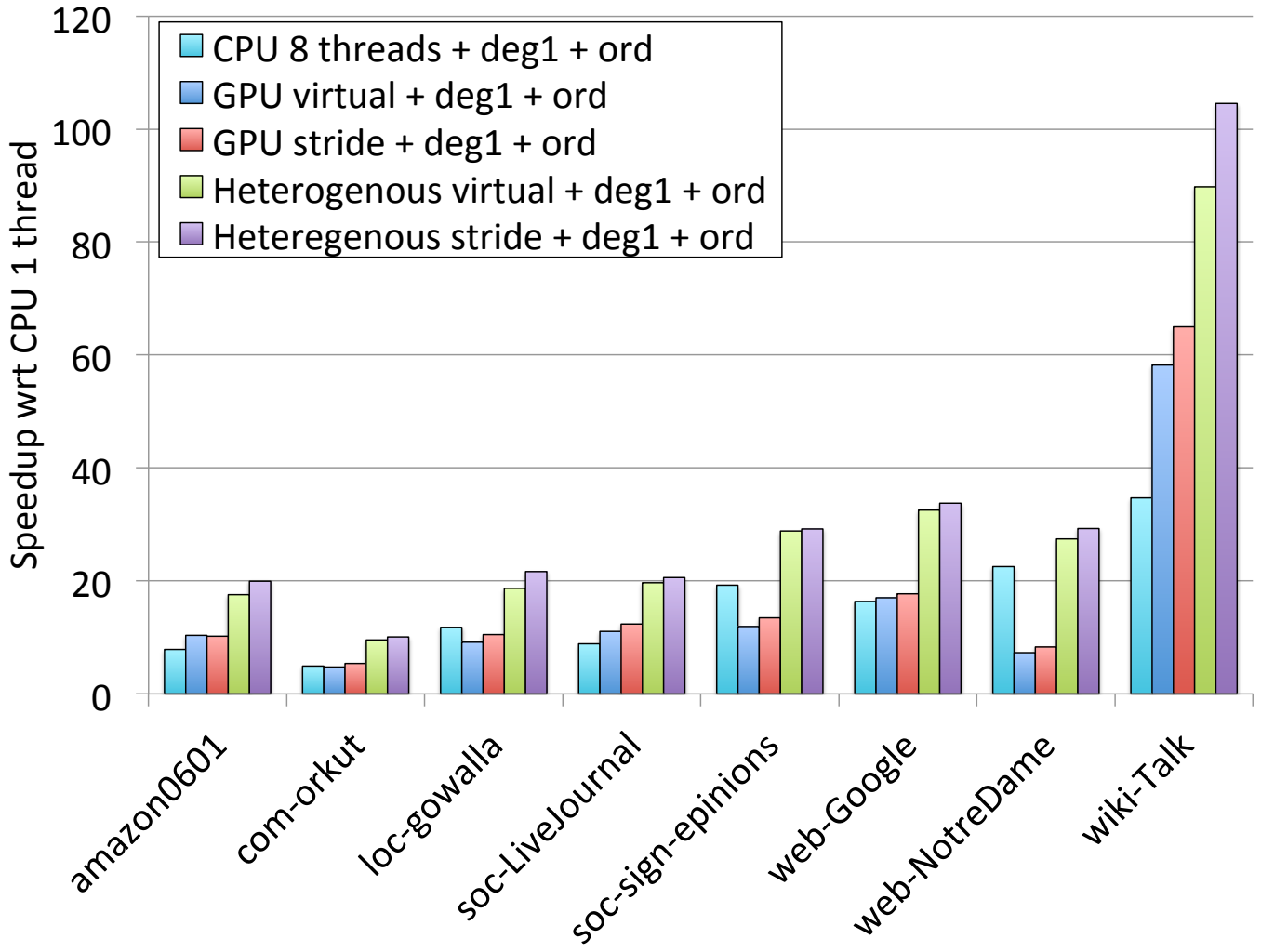


Örnek problem: Merkezi noktaları bulmak

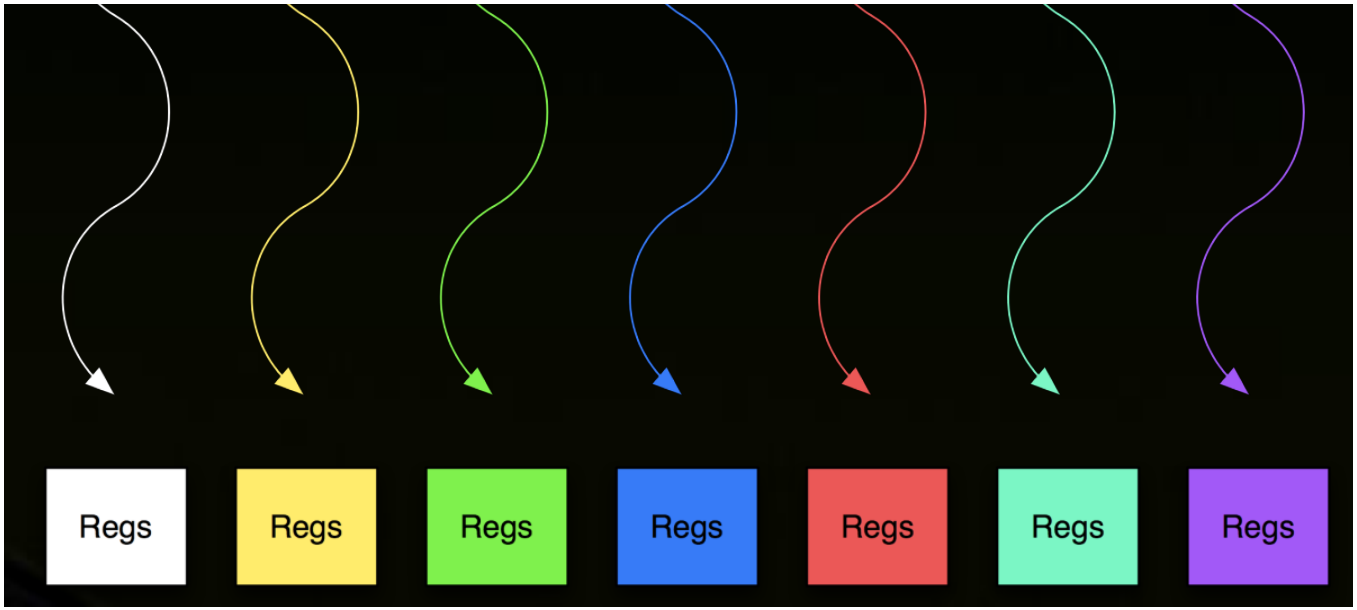




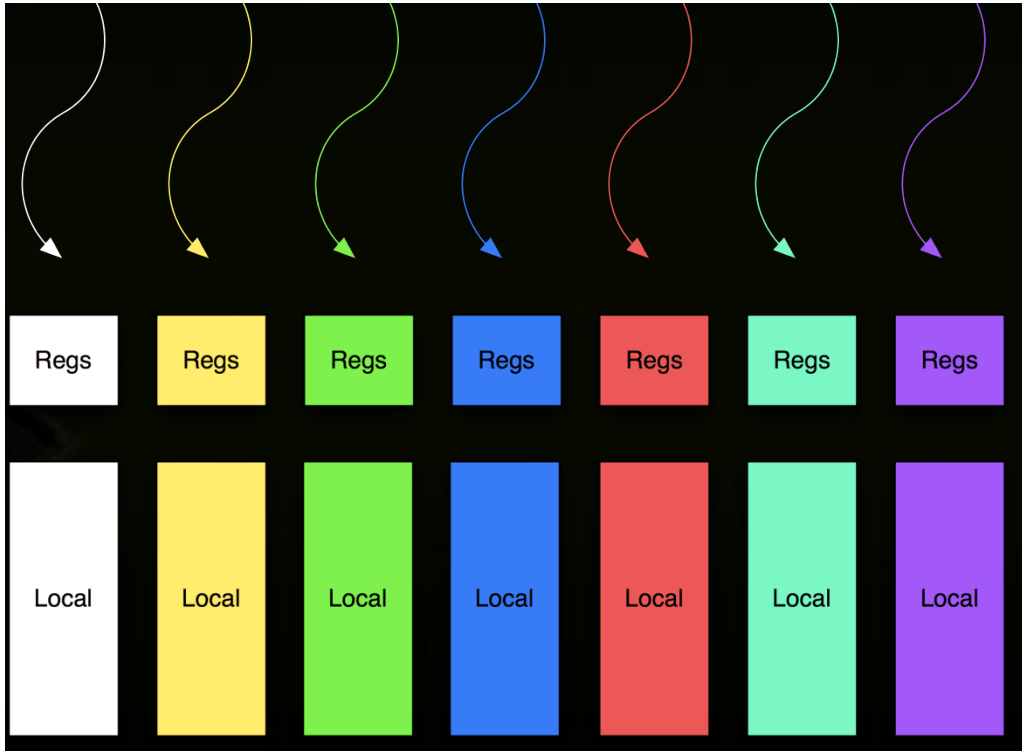
Örnek problem: Merkezi noktaları bulmak



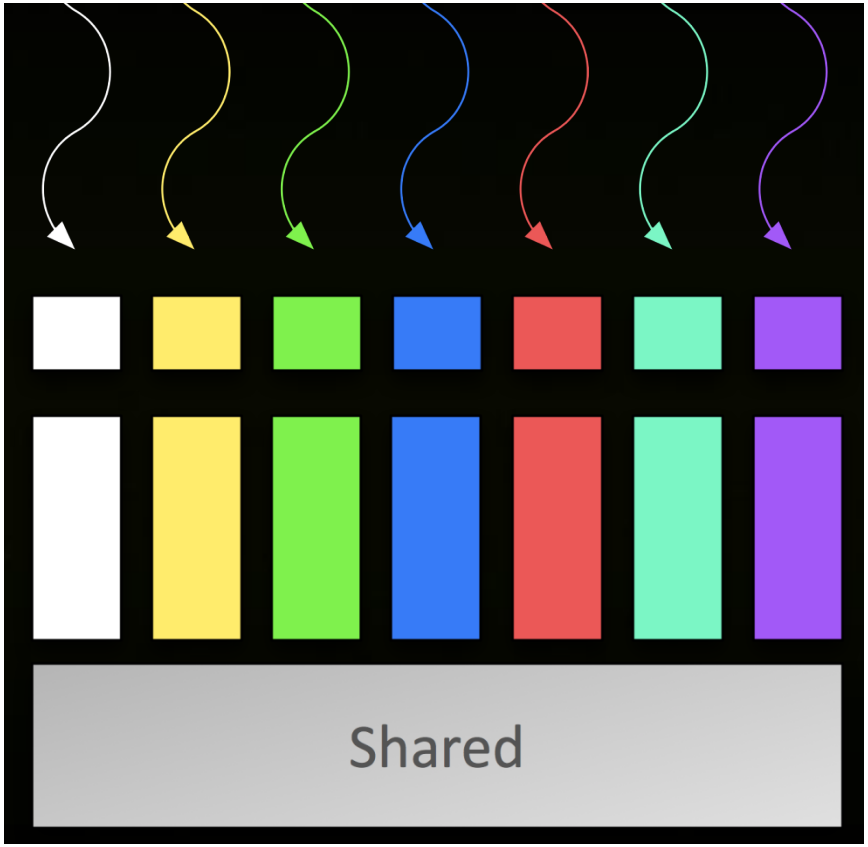
GİÜ: Programlama – CUDA Hafıza yapısı



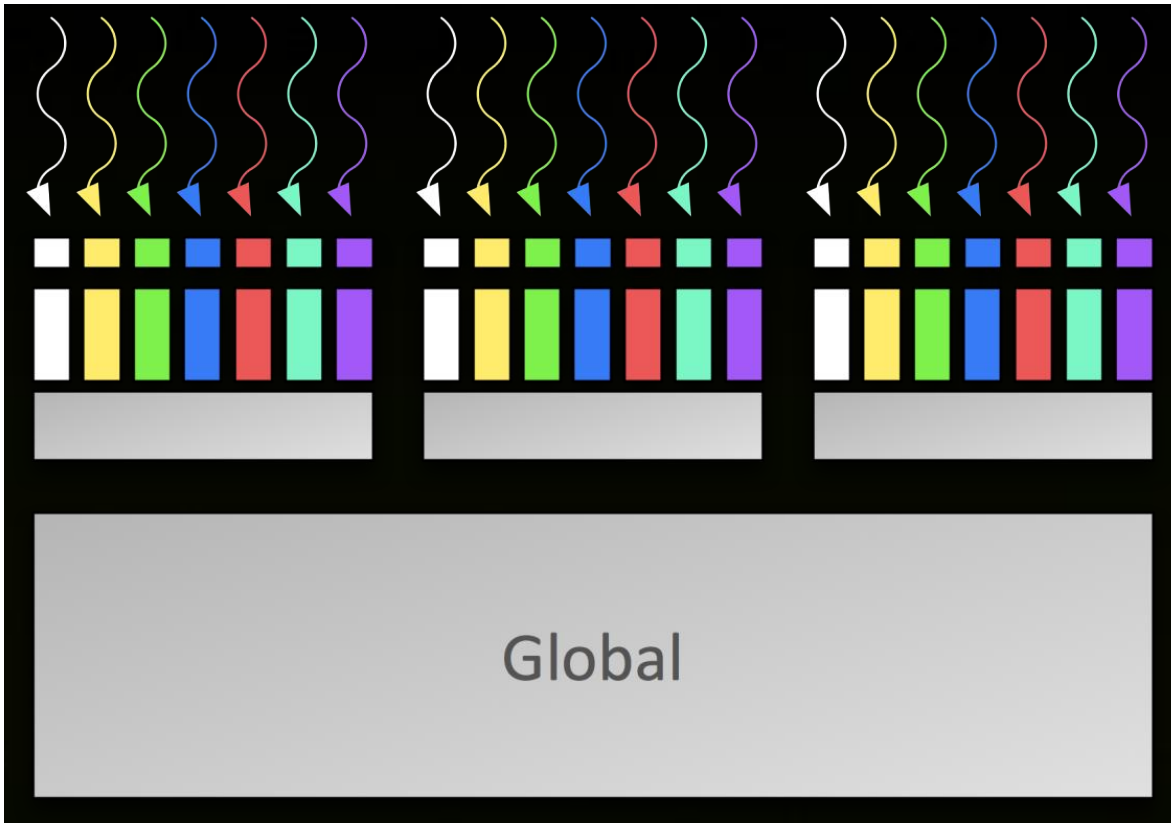
GİÜ: Programlama – CUDA Hafıza yapısı



GİÜ: Programlama – CUDA Hafıza yapısı



GİÜ: Programlama – CUDA Hafıza yapısı



Atomics

- Guarantee that only a single thread has access to a piece of memory during an operation
- No dropped data, but ordering is still arbitrary
- Different types of atomic instructions
- Atomic Add, Sub, Exch, Min, Max, Inc, Dec, CAS, And, Or, Xor
- Can be done on device memory and shared memory
- Much more expensive than load + operation + store

Example: Histogram

13

```
// Determine frequency of colors in a picture.
// Colors have already been converted into integers
// between 0 and 255.
// Each thread looks at one pixel,
// and increments a counter

__global__ void histogram(int* colors, int* buckets)
{
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    int c = colors[i];
    buckets[c] += 1;
}
```

Example: Histogram

14

```
// Determine frequency of colors in a picture.
// Colors have already been converted into integers
// between 0 and 255.
// Each thread looks at one pixel,
// and increments a counter

__global__ void histogram(int* colors, int* buckets)
{
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    int c = colors[i];
    buckets[c] += 1;
}
```

Why is this incorrect?

Example: Histogram

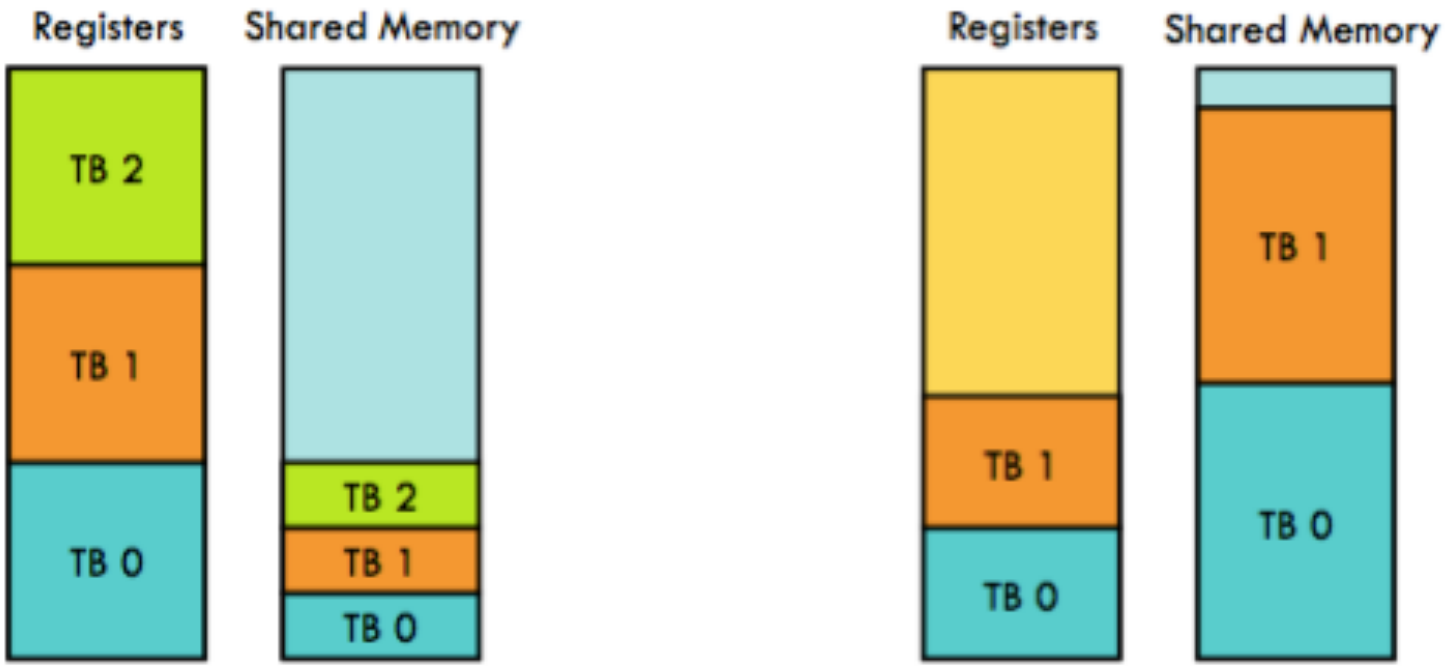
15

```
// Determine frequency of colors in a picture.
// Colors have already been converted into integers
// between 0 and 255.
// Each thread looks at one pixel,
// and increments a counter atomically

__global__ void histogram(int* colors, int* buckets)
{
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    int c = colors[i];
    atomicAdd(&buckets[c], 1);
}
```

Occupancy

- What determines occupancy?
- Limited resources!
  - Register usage per thread
  - Shared memory per thread block

Resource Limits (1)



- Pool of registers and shared memory per SM
  - Each thread block grabs registers & shared memory
  - If one or the other is fully utilized ➡ no more thread blocks

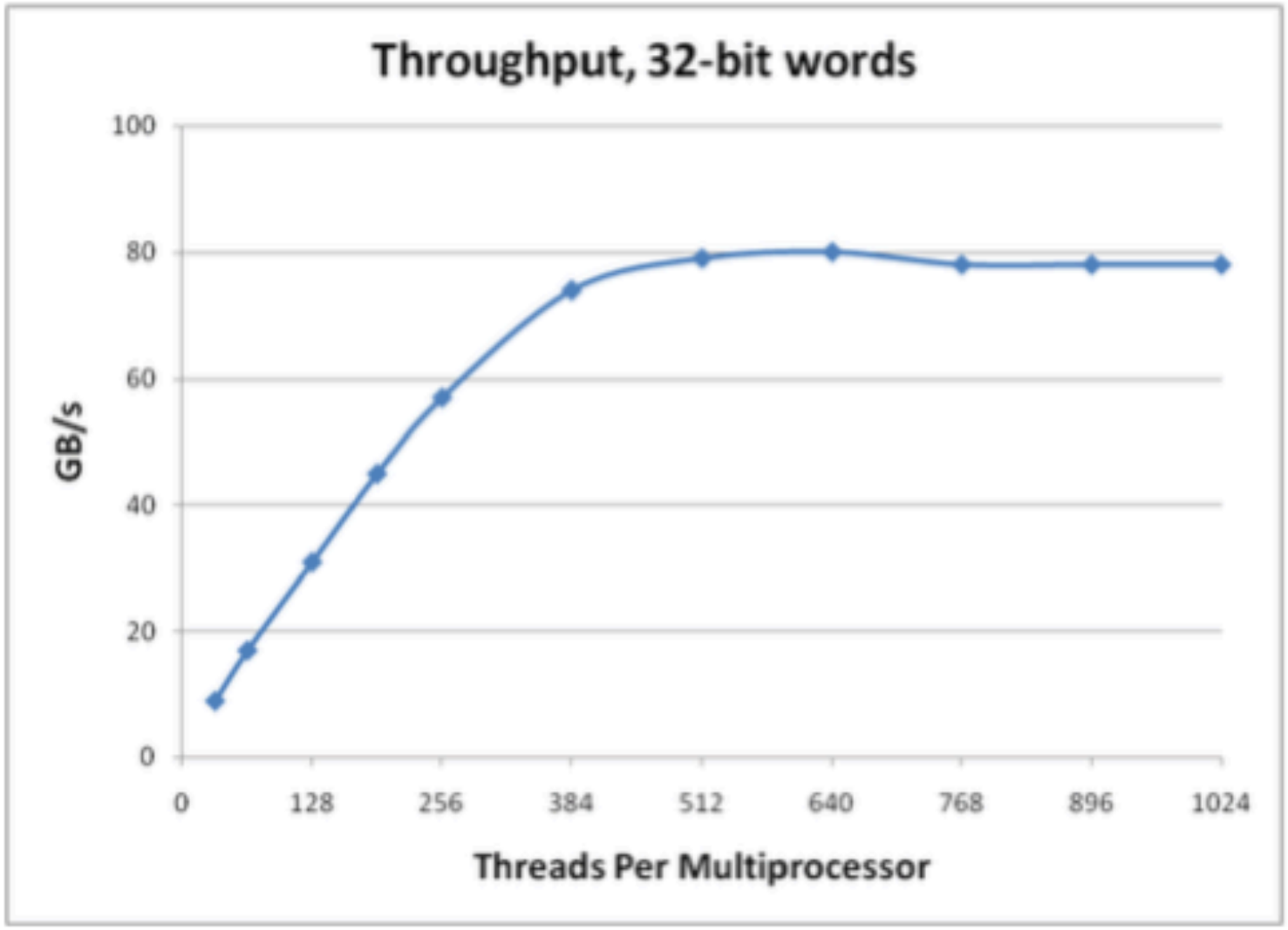
Resource Limits (2)

- Can only have 8 thread blocks per SM (16 for Kepler)
  - If they're too small, can't fill up the SM
  - Need 128 threads / block on gt200 (4 cycles/instruction)
  - Need 192 threads / block on Fermi (6 cycles/instruction)

- Higher occupancy has diminishing returns for hiding latency

Hiding Latency with more threads

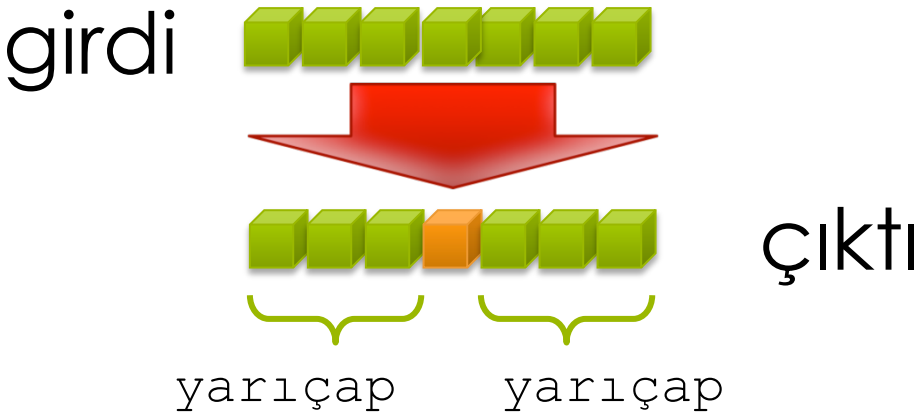
41

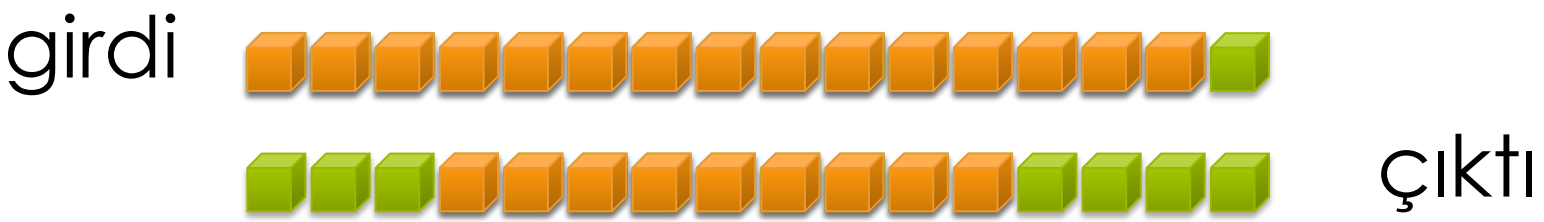


Örnek problem: 1D-stencil

Tek boyutlu dizinde

- Her çıktı elemanı belli bir yarıçaptaki girdi elemanlarının toplamına eşit





Örnek problem: 1D-stencil

```
__global__ void stencil_1d(int *in, int *out) {
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    for (int offset = -R; offset <= R; offset++)
      out[i] += in[i + offset];
}
```

Örnek problem: 1D-stencil

```
__global__ void stencil_1d(int *in, int *out) {
    int i = threadIdx.x + blockIdx.x * blockDim.x;
    int result = 0;
    for (int offset = -R; offset <= R; offset++)
      result += in[i + offset];
    out[i] = result;
}
```

Örnek problem: 1D-stencil


```
__global__ void stencil_1d(int *in, int *out) {
  __shared__ int temp[BLOCK_SIZE + 2 * RADIUS];
  int gindex = threadIdx.x + blockIdx.x * blockDim.x;
  int lindex = threadIdx.x + RADIUS;

  // Read input elements into shared memory
  temp[lindex] = in[gindex];
  if (threadIdx.x < RADIUS) {
    temp[lindex - RADIUS] = in[gindex - RADIUS];
    temp[lindex + BLOCK_SIZE] =
      in[gindex + BLOCK_SIZE];
  }
```

Örnek problem: 1D-stencil


```
  // Apply the stencil
  int result = 0;
  for (int o = -RADIUS ; o <= RADIUS ; o++)
    result += temp[lindex + o];

  // Store the result
  out[gindex] = result;
}
```

İsviçre Çakıları…

Tek kod. Bütün işlemciler ve hızlandırıcılar

OpenMP 5.0, OpenACC, OpenCL

İsviçre Çakıları…

OpenMP to Intel Xeon Phi

```
#pragma omp target device(0)  map(tofrom:B)
#pragma omp parallel for
for (i=0; i<N; i++)
        B[i] += sin(B[i]);
```

İsviçre Çakıları…

OpenMP to GPGPU

```
#pragma omp target device(0) map(tofrom:B)
#pragma omp teams num_teams(num_blocks)
                num_threads(bsize)
#pragma omp distribute
for (i=0; i<N; i += num_blocks)
      #pragma omp parallel for
      for (b = i; b < i+num_blocks; b++)
            B[b] += sin(B[b]);
```

İsviçre Çakıları…

OpenACC to GPGPU

```
#pragma acc parallel copy(B[0:N]) \
num_gangs(numblocks) \
vector_length(bsize)
#pragma acc loop gang vector
for (i=0; i<N; ++i) {
        B[i] += sin(B[i]);
}
```

İsviçre Çakıları

CUDA

```
__global__ void
vectorAdd(float* a, float* b, float* c) {
  int index = blockIdx.x * blockDim.x + threadIdx.x;
  c[index] = a[index] + b[index];
}
```

OpenCL

```
__kernel void
vectorAdd(__global float* a, __global float* b,
          __global float* c) {
    int index = get_global_id(0);
    c[index] = a[index] + b[index];
}
```

İsviçre Çakıları

Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: programming productivity, performance, and energy consumption

Suejb Memeti[1] Lu Li[2] Sabri Pllana[3] Joanna Kołodziej[4] Christoph Kessler[5]

Our major observations include: (1) programming with OpenCL requires significantly more effort than programming with OpenACC for SPEC Accel benchmark suite; (2) programming with OpenCL on average requires about two times more effort than programming with CUDA for Rodinia benchmark suite; (3) the human factor can impact the fraction of code lines to parallelize the code; (4) OpenMP, OpenCL, and CUDA have comparable performance and energy consumption on Ida; and (5) OpenCL performs better than OpenACC for SPEC Accel benchmark suite on Ida; (6) programming with OpenMP requires less effort than programming with OpenCL and CUDA.

Python ile basitçe paralel programlama

- Çok çekirdekli işlemciler…
- Birden fazla iplik kullanımı (OpenMP, CUDA vb.)
- Fakay python'da bu çok verimli olmuyor
- Global Interpreter Lock
  - Güvenli çok çekirdekli işleme

Python ile basitçe paralel programlama

- Python'un multiprocessing kütüphanesinin bir altkümesini inceleyeceğiz.
  - İlk örneklerimizde Process sınıfını kullanacağız.
  - Daha sonra Pool sınıfı ile basit bir paralleleştirme elde edeceğiz.
    - apply, map, apply_async, map_async

Bibliyografi ve ilgili bağlantılar

- **Where HPC & Big Data Intersect**, Bruce Hendrickson, Sandia National Labs
- **A Tale of Two Data-Intensive Paradigms: Applications, Abstractions, and Architectures**, Jha, S., Qiu, J., Luckow, A., Mantha, P., 2014 IEEE International Congress on Big Data (BigData Congress)
- **How Big Data Is Redefining High Performance Computing**, Earl Joseph, Steve Conway, Chirag Dekate, Bob Sorensen
- http://www.extremetech.com/extreme/185057-supercomputer-stagnation-new-list-of-the-worlds-fastest-computers-makes-exascale-by-2020-seem-unlikely
- **Accelerating Iterative Big Data Computing Through MPI**, Fan Liang and Xiaoyi Lu
- **A Tale of Two Data-Intensive Paradigms: Applications, Abstractions, and Architectures** Shantenu Jha, Judy Qiu, Andre Luckow, Pradeep Mantha, Geoffrey C.Fox
- http://www.theplatform.net/2015/02/22/flink-sparks-next-wave-of-distributed-data-processing/
- http://mechanitis.blogspot.com.tr/2011_07_01_archive.html
- top500.org/featured/top-systems/
- http://sebastianraschka.com/Articles/2014_multiprocessing.html
- https://docs.python.org/3/library/multiprocessing.html