

Paralel Programlama ve MPI



Konu başlıkları

- Paralel hesaplama
 - Hızlanma
- Paralel bilgisayarlar
 - Ortak bellekli sistemler ve programlama
 - Dağıtık bellekli sistemler ve programlama
 - Dağıtık ortak bellekli sistemler ve programlama
 - Kümeler (Clusters)
- İleti temelli (message-passing) programlama
- MPI

Daha Fazla Hız

- Mevcut hesaplamayı daha hızlı yapmak her zaman en doğal istek
- Özellikle modellemeler, mühendislik uygulamaları ve bilimsel uygulamalarda daha yüksek hıza ihtiyaç duyulmakta (need for speed :)
- DNA modellenmesi, hava tahmini* vs. ağır problemlere örnek
- Her hesaplama için bir “tahammül” sınırı var

Süper Kahraman: Paralel Hesaplama

- Paralel hesaplama: Bir problemi, birden fazla bilgisayar ve/veya işlemci kullanarak çözmek

Yani ?

- Daha hızlı hesaplama: Basit, 5 bilgisayar, 1 bilgisayarın yaptığı bir işi 5 kat daha kısa sürede yapar. (Yalan*)
- Paralel hesaplamanın temel taşları:
 - İletişim
 - Senkronizasyon

* 1 kadın, 9 ayda 1 çocuk doğurabilir ama 9 kadın 1 ayda 1 çocuk doğuramaz

Hızlanma

$$S(p) = \frac{\text{Tek işlemcide çalışma zamanı (en iyi seri algoritma ile)}}{p \text{ adet işlemcide çalışma zamanı}} = \frac{t_s}{t_p}$$

- $S(p)$ hızlanmayı (speedup) ifade eder.
- Tek işlemcide bilinen en iyi seri algoritma kullanılır. (adil kıyaslama için) Paralelde kullanılan algoritma ise (muhtemelen) farklıdır.

Hızlanma - 2

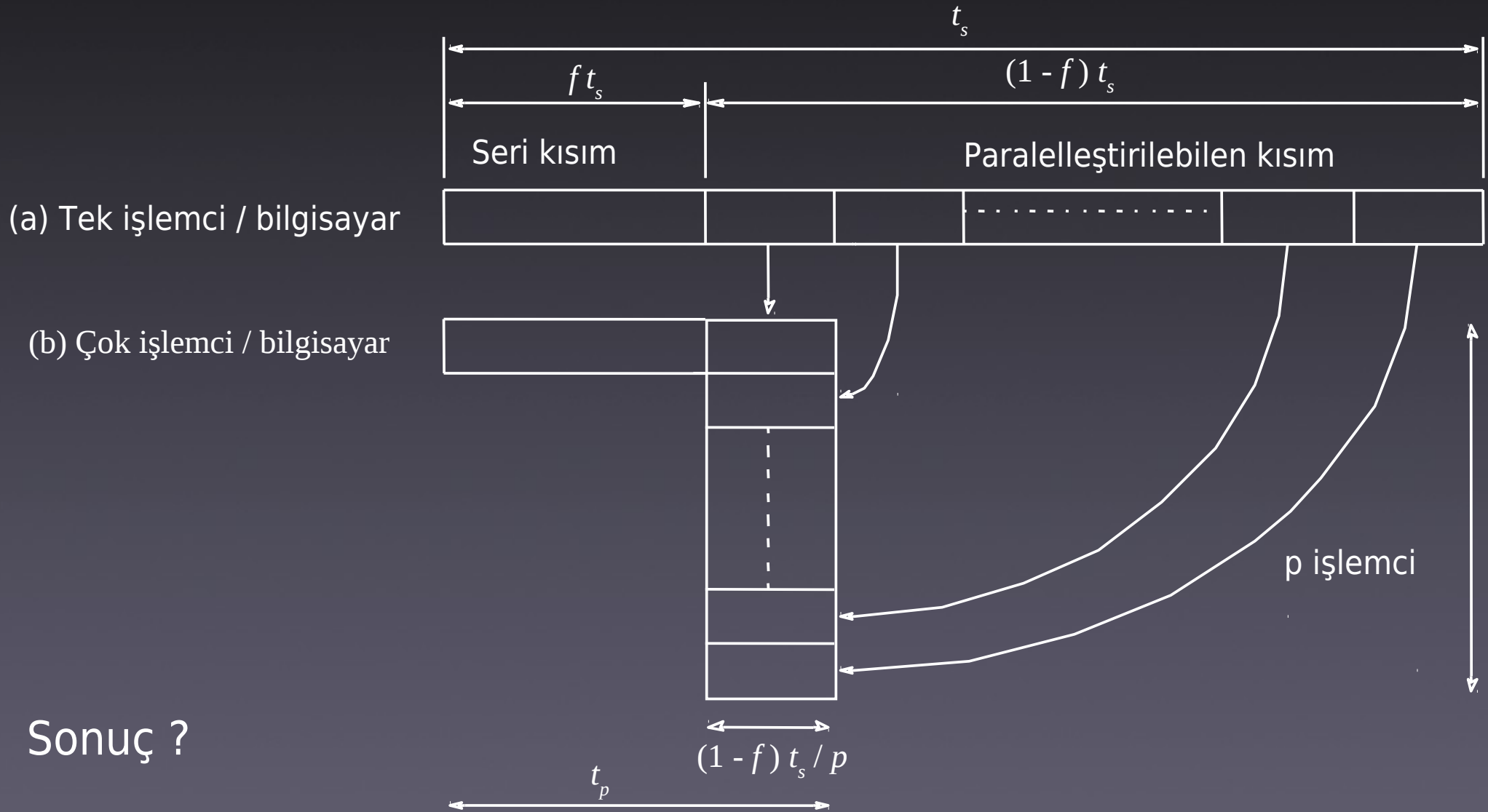
- p adet işlemciyle p kat hızlanılabilir. (doğrusal hızlanma) Daha fazlası mümkün mü?

Evet! Nasıl?!

- Daha fazla hafıza
- Gerekçisiz (nondeterministic) algoritmalar
 - ✓ Doğrusal arama örneği ilerde

Hızlanma - 3

Amdahl Yasası



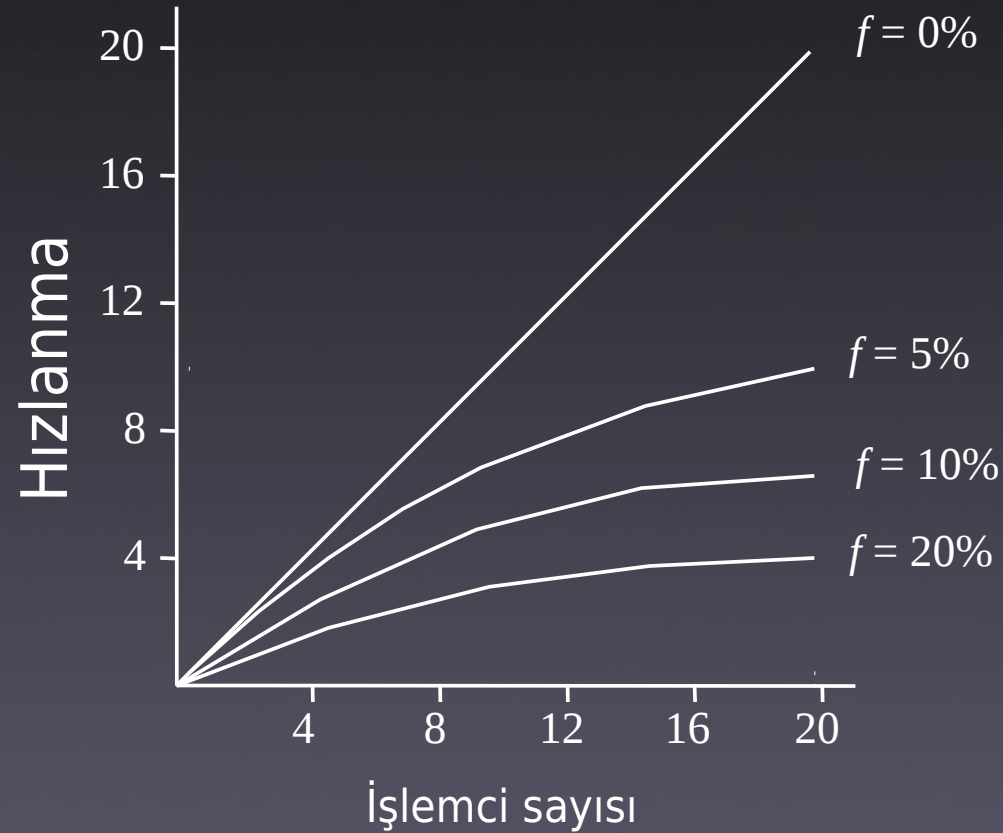
Sonuç ?

Hızlanma:

$$S(p) = \frac{t_s}{ft_s + (1-f)t_s/p} = \frac{p}{1 + (p-1)f}$$

Bu eşitlik Amdahl Yasası olarak bilinir.

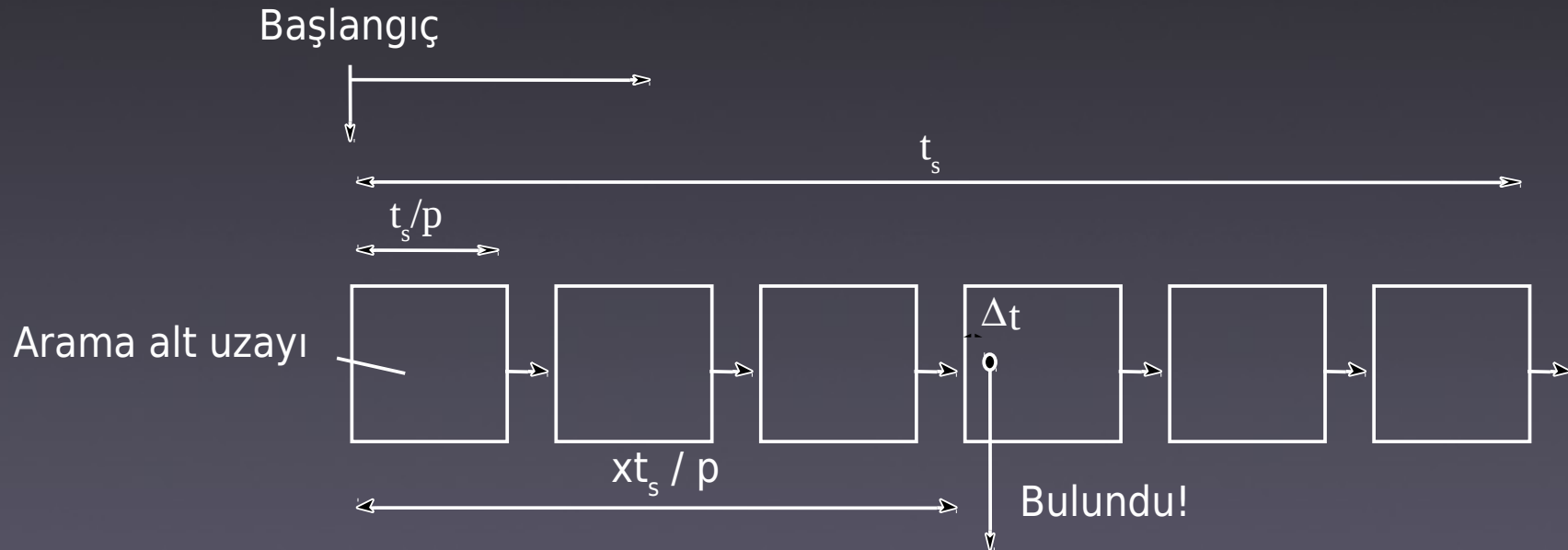
İşlemci sayısına göre hızlanma



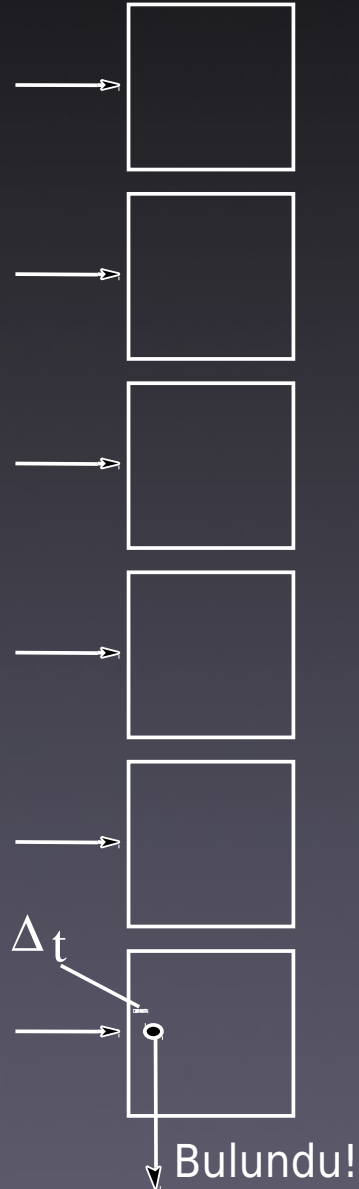
Neden? İletişim masrafı

Doğruötesi (SuperLinear) hızlanma örneđi - Arama

(a) Seri - Her alt uzay, sırayla aranır.



(b) Paralel - Her alt uzayda aynı anda arama yapılır



Bu durumda hızlanma:

$$S(p) = \frac{\left[x \quad x \quad \frac{t_s}{p} \right] + \Delta t}{\Delta t}$$

Olur, ve $\Delta t \rightarrow 0$, $S(p) \rightarrow \infty$

Paralel Bilgisayarlar

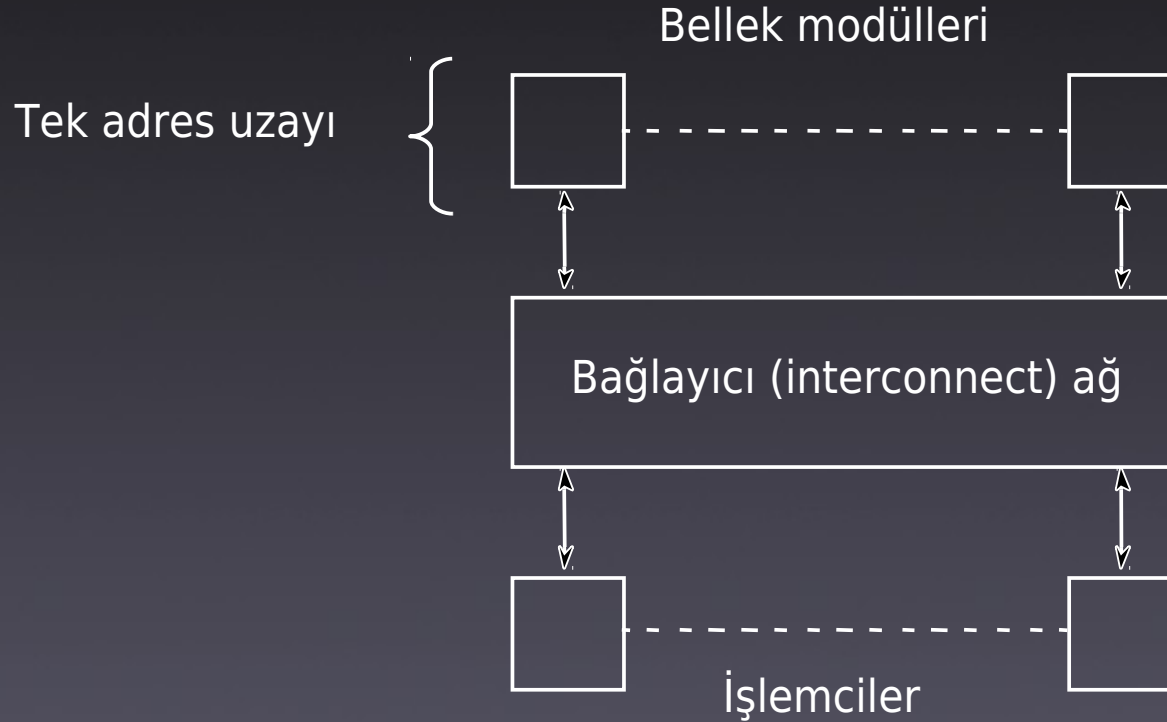
İki temel tür:

- Ortak bellekli (shared memory) sistemler
 - SMP (symmetric multiprocessing) — Intel Core serisi
 - NUMA (Non-Uniform memory access)
- Dağıtık* bellekli (distributed memory) sistemler
 - Kümeler
 - MPP (massively parallel processors)

* *Bildiğimiz dağıtık kavramı değil! Ayrı ayrı bellekleri kullandıklarını belirtir.*

Ortak Bellekli Sistemler

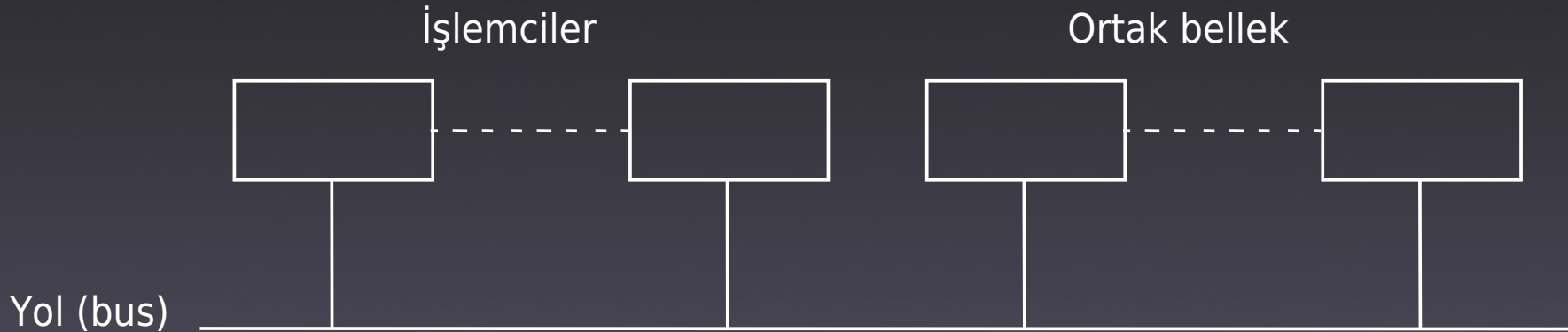
Çok işlemcili bir sistem yaratmanın akla gelen ilk yolu: Tüm bellek ve işlemcileri birbirine bağlamak :)



- Ölçeklenme sorunu! $< \sim 100$
- > 100 (massively parallel) için dağıtık bellek şart

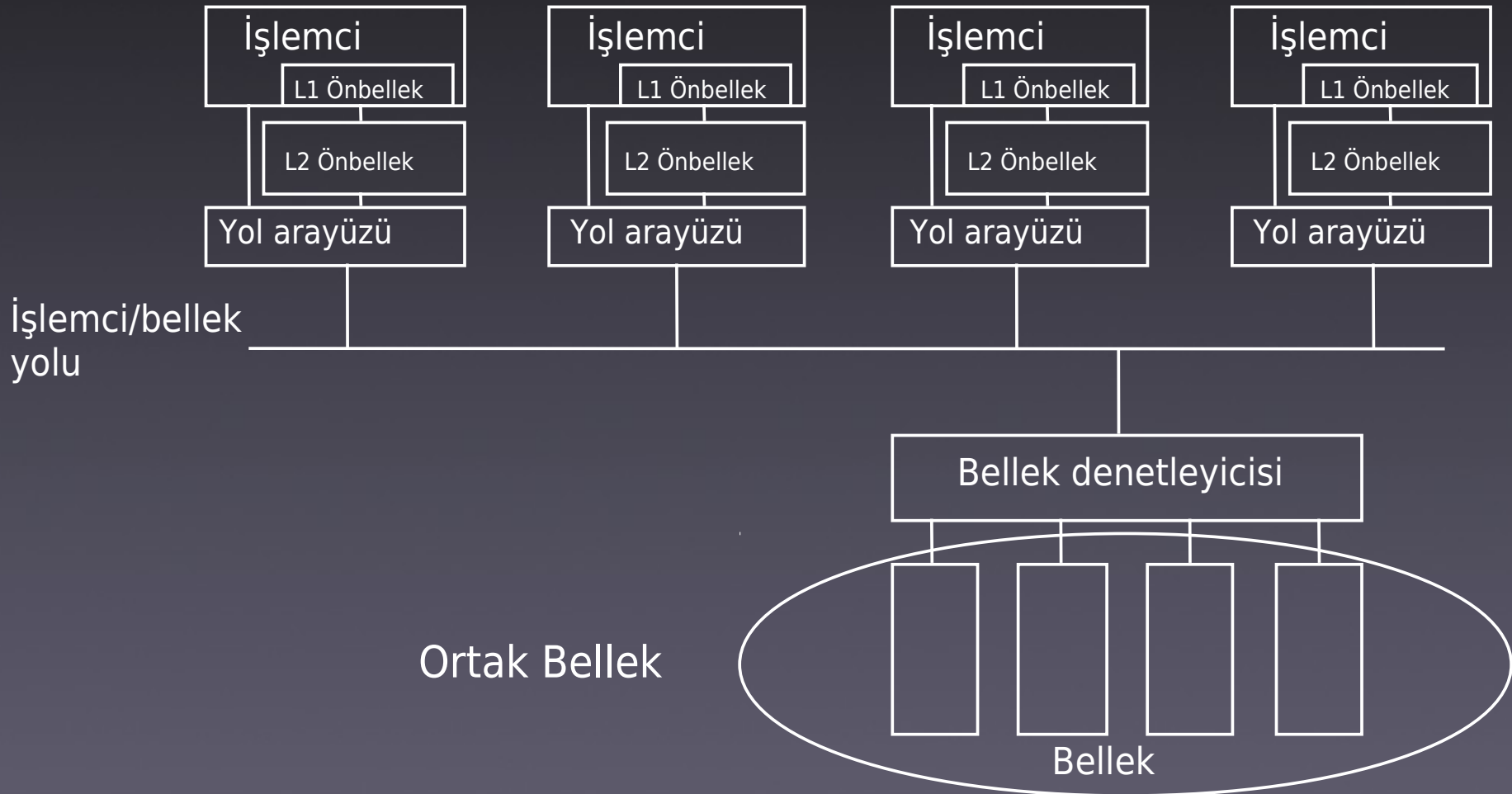
Ortak Bellekli Sistemler - 2

Örnek sistem:



- Pentium Core Duo/Quad

Pentium Core 2 Quad



Nasıl Programlamalı?

Bir sürü yolu mevcut:

- **Thread'ler** – programcı, programı eşzamanlı çalışacak parçalara ayırır. Bu parçalar (threadler) aynı belleği kullandığı için dışarda(heap, global) tanımlanan herşeye rahatça erişebilirler. Kontrol sende!

Misal, Pthreads. Hemen her uygulamanın paralellik ihtiyacı karşılanabilir.

- **Seri programlama dilleriyle yazılan programları, derleyici direktifleri yoluyla paralelleştirerek, bu direktiflerden anlayan bir derleyici kullanmak** – kodu paralelleştirmek programcı ve derleyicinin ortak çalışmasıyla gerçekleştirilir

Misal OpenMP - OpenMP derleyicisiyle kullanılır, pilleri içindedir (özünde thread'leri kullanır)

```
#pragma omp parallel shared(a,b,c,nthreads,chunk) private(i,tid)
{
    tid = omp_get_thread_num();
    if (tid == 0)
    {
        nthreads = omp_get_num_threads();
        printf("Number of threads = %d\n", nthreads);
    }
    printf("Thread %d starting...\n",tid);
    #pragma omp for schedule(dynamic,chunk)
    for (i=0; i<N; i++)
    {
        c[i] = a[i] + b[i];
        printf("Thread %d: c[%d]= %f\n",tid,i,c[i]);
    }
} /* end of parallel section */
```

OpenMP example

Nasıl Programlamalı? - 2

- **Seri programlama dillerini, paralelden anlayacak şekilde genişleterek [yazımına (syntax) müdahale ederek] bir dil türevi oluşturmak**
 - Misal UPC (Unified Parallel C) - UPC derleyicisi gerektirir, kısaca paralel C türevi. Paralelliği yazım (syntax) seviyesinde desteklemekte
- **Tamamıyla paralel yeni bir programlama dili geliştirmek -** derleyici her işlemci için ayrı program üretir.
 - Misal (rahmetli) Occam - 80'lerde çıktı, artık pek sık kullanılmıyor

UPC Örneği:

```
// vect_mat_mult.c
#include <upc_relaxed.h>
shared [THREADS] int a[THREADS][THREADS];
shared int b[THREADS], c[THREADS];

void main (void) {
    int i, j;
    upc_forall( i = 0 ; i < THREADS ; i++ ; i) {
        c[i] = 0;
        for ( j= 0 ; j< THREADS ; j++ )
            c[i] += a[i][j]*b[j];
    }
}
```

occam Örneği

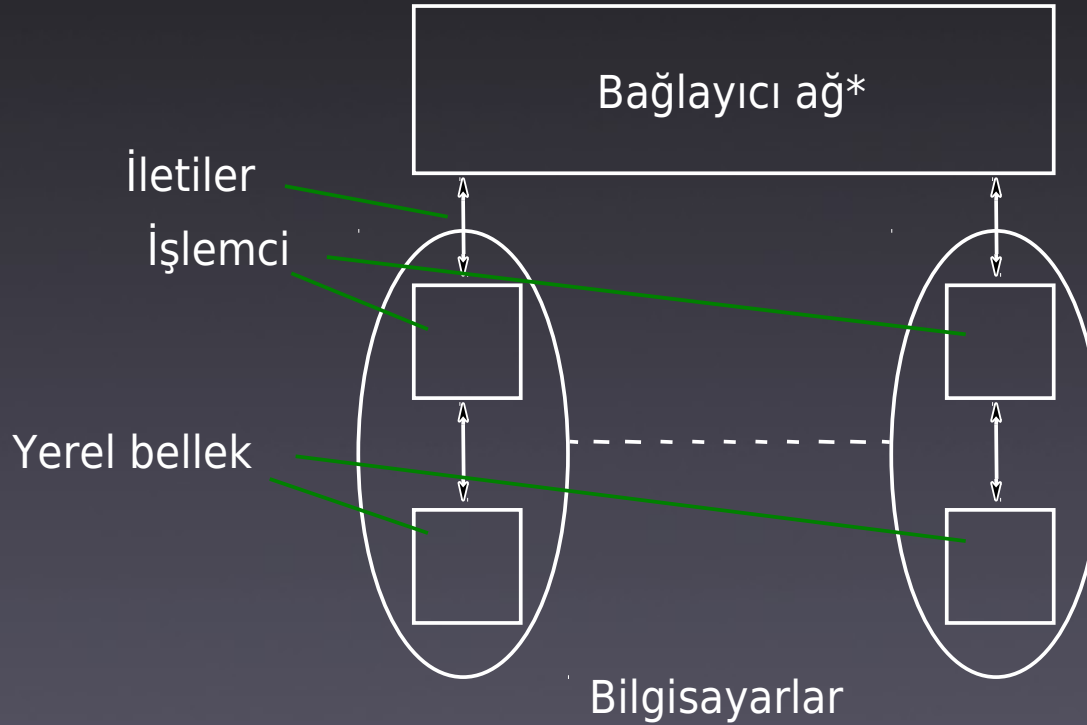
```
PROC producer (CHAN INT out!)
    INT x:
    SEQ
    x := 0
    WHILE TRUE
        SEQ
        out ! x
        x := x + 1
    :

PROC consumer (CHAN INT in?)
    WHILE TRUE
    INT v:
    SEQ
    in ? v
    .. do something with `v'
    :

PROC network ()
    CHAN INT c:
    PAR
    producer (c!)
    consumer (c?)
    :
```

Dağıtık Bellekli Sistemler

- Tüm işlemci/bilgisayarlar bir bağlayıcı ağ (interconnect network) ile bağlı:



- Her işlemci, yerel belleğine doğrudan, diğer belleklere ise oradaki işlemcilere ileti yollayarak erişir

* Mesh, hypercube, crossbar, tree vs.

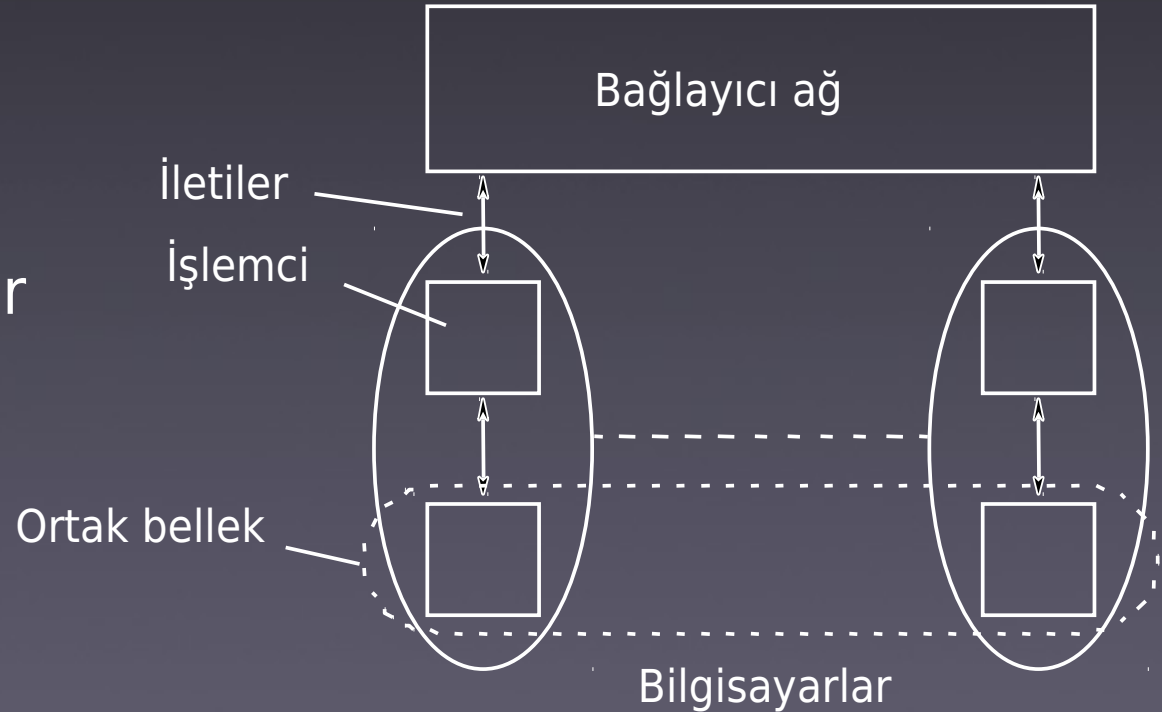
Dağıtık Bellekli Sistemler - 2

- İleti-temelli çoklubilgisayarlar (message-passing multicomputers), yerel bellekli çoklubilgisayarlar diye de bilinirler
- Nasıl programlamalı?
 - İleti temelli (message-passing) iletişim kullanılır.
 - MPI ve eski dost PVM
 - ~1000 işlemcili (massively) paralel sistemlerde kullanılır
 - Ortak bellekte de kullanılabilir

Dağıtık Ortak Bellekli Sistemler

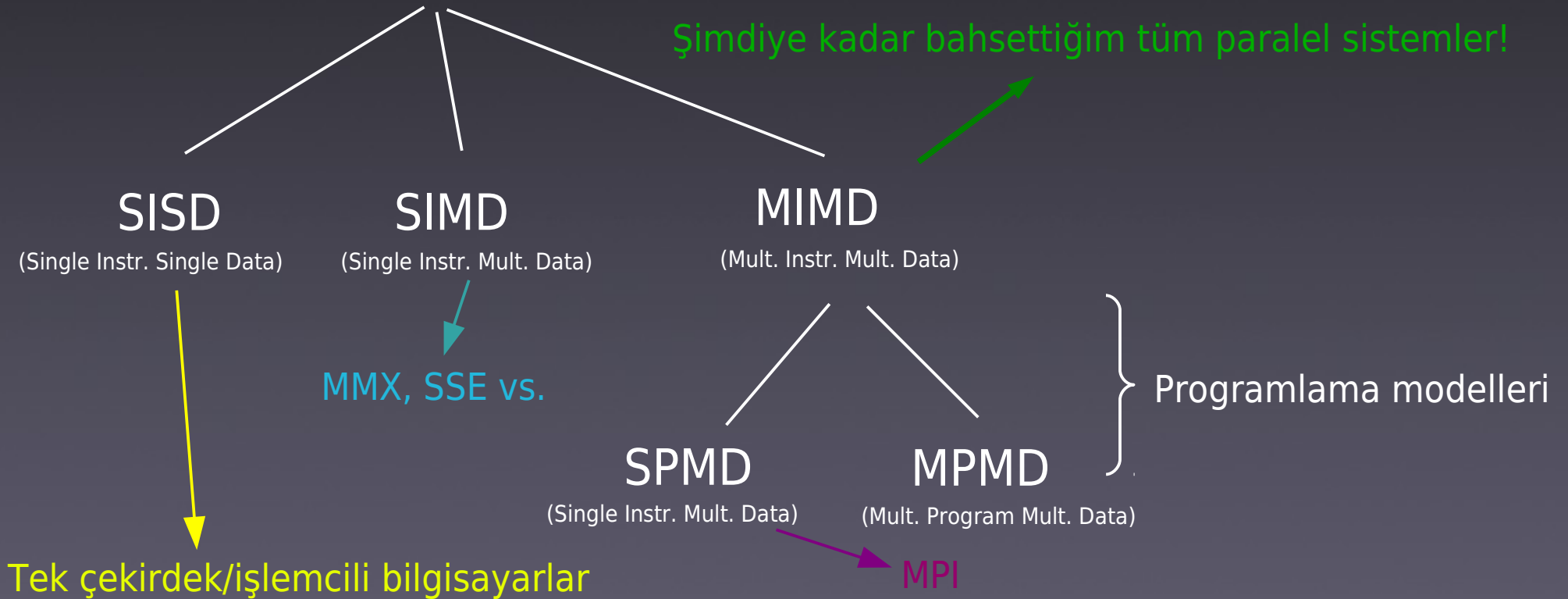
- Bağlı bir grup bilgisayarı (ya da aynı makinedeki işlemci/bellek çiftlerini), yerel belleklerin tümünü, tek bir ortak bellekmiş gibi tek adres uzayı içinde kullanacak şekilde biraraya getirilmesiyle oluşturulur. Tüm ortak bellek programlama yöntemleri caizdir.

- Kolaylık, çoğu zaman az performansla cezalandırılır



Flynn'in Sınıflandırması

Flynn (1966) komut ve veri akışını temel alarak bilgisayarlar için bir sınıflandırma yarattı



Kümeler (Clusters)

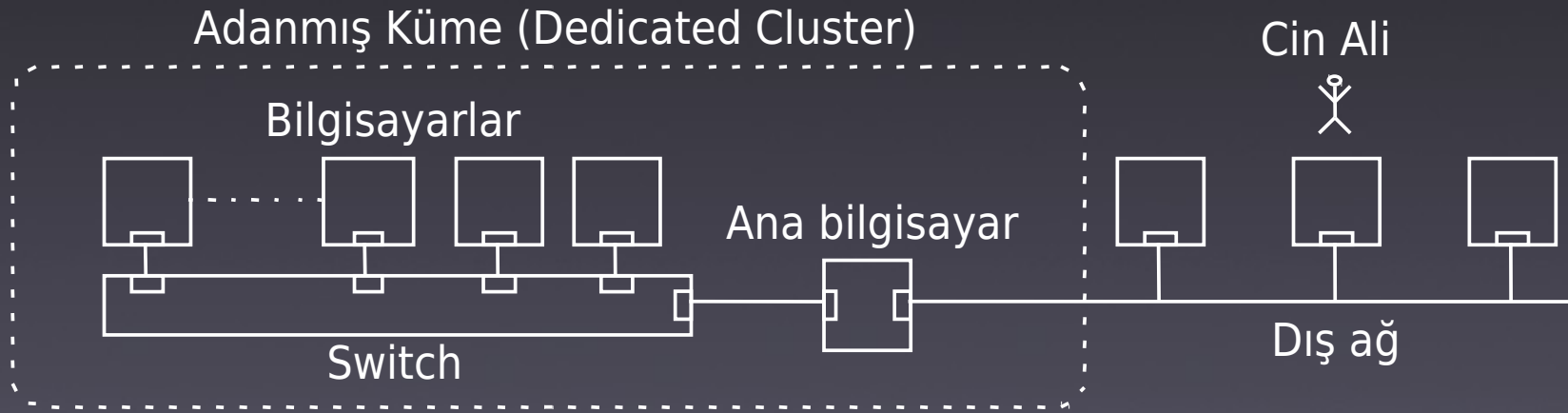
- Birbirine bağlı bilgisayarlar, 90'ların başından itibaren yüksek başarımli hesaplamada (high performance computing) pahalı süper bilgisayarlara sıkı rakip olmaya başladılar.
Misal NASA Beowulf* procesi, Berkeley NOW procesi vs.
- Neden?
 - Gayet ucuz!
 - Yeni çıkan işlemcilerden direk faydalanılabilir
 - Ucuz ağ teknolojileri

* Bu ad, artık piyasada bulunan bilgisayarlarla kurulan kümelerle özdeşleşmiş durumda: Beowulf kümeleri. Bu tip kümelerde, sıklıkla PC+ Linux + Ethernet üçlüsü kullanılır.

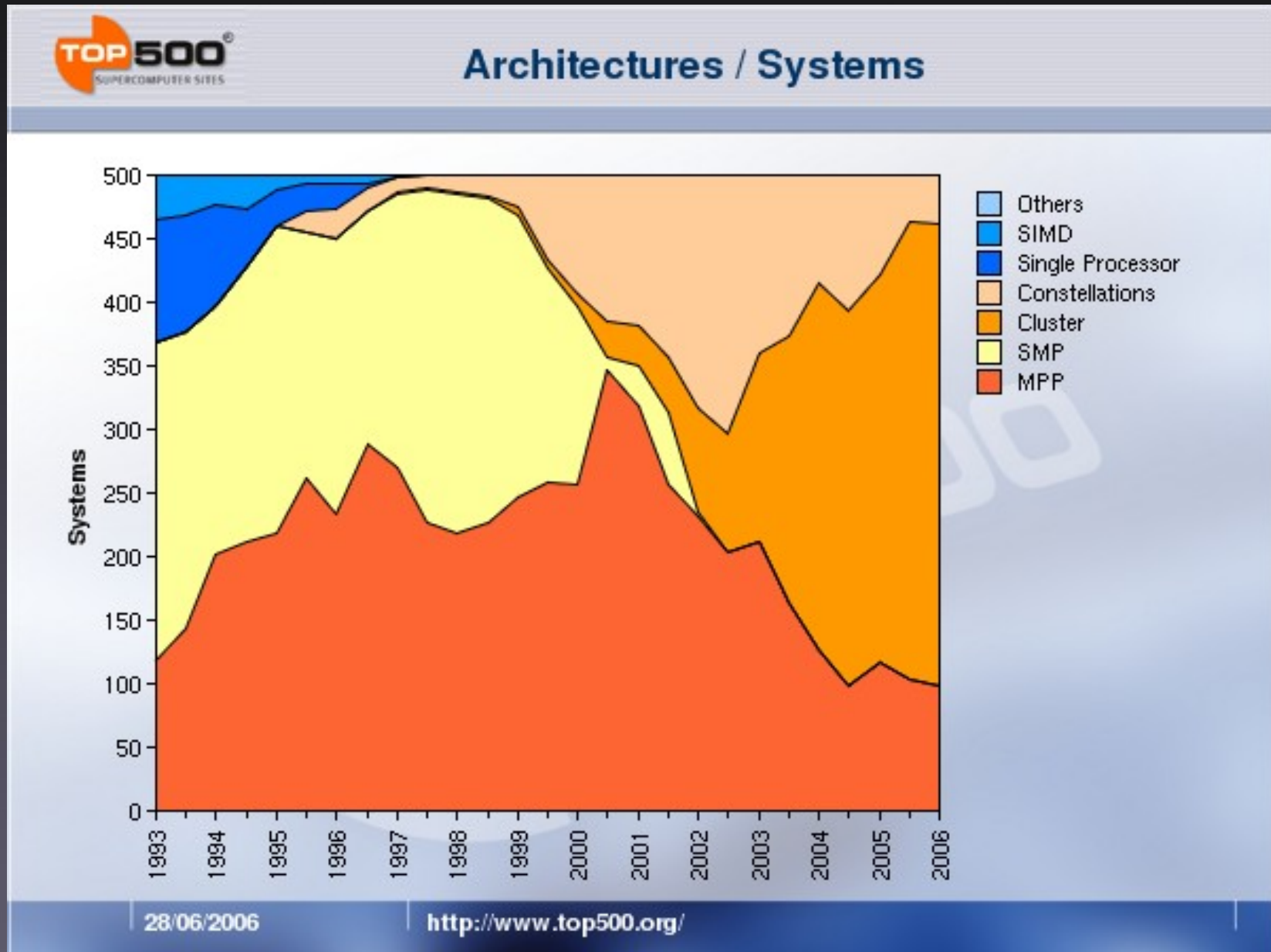
Kümeler için geliştirme araçları

- Evet yine onlar :)
 - Parallel Virtual Machine (PVM) – 80'lerde geliştirildi ve çok meşhurladı.
 - Message-Passing Interface (MPI) – 90'larda geliştirildi, çoğu alanda PVM'den üstün, misal:
 - Birçok kaliteli implementasyonu var
 - Taşınabilir
 - Yüksek başarımlı iletişim
- Özetle ikisi de, kullanıcı seviyesinde (user space) ileti-temelli programlama sağlamakta. C, Fortan gibi dillerde çalışabilmekte.
- Kümelerde ayrıca OpenMP + MPI hibrit çözümü de sık kullanılır

Adanmış Küme Modeli



TOP500



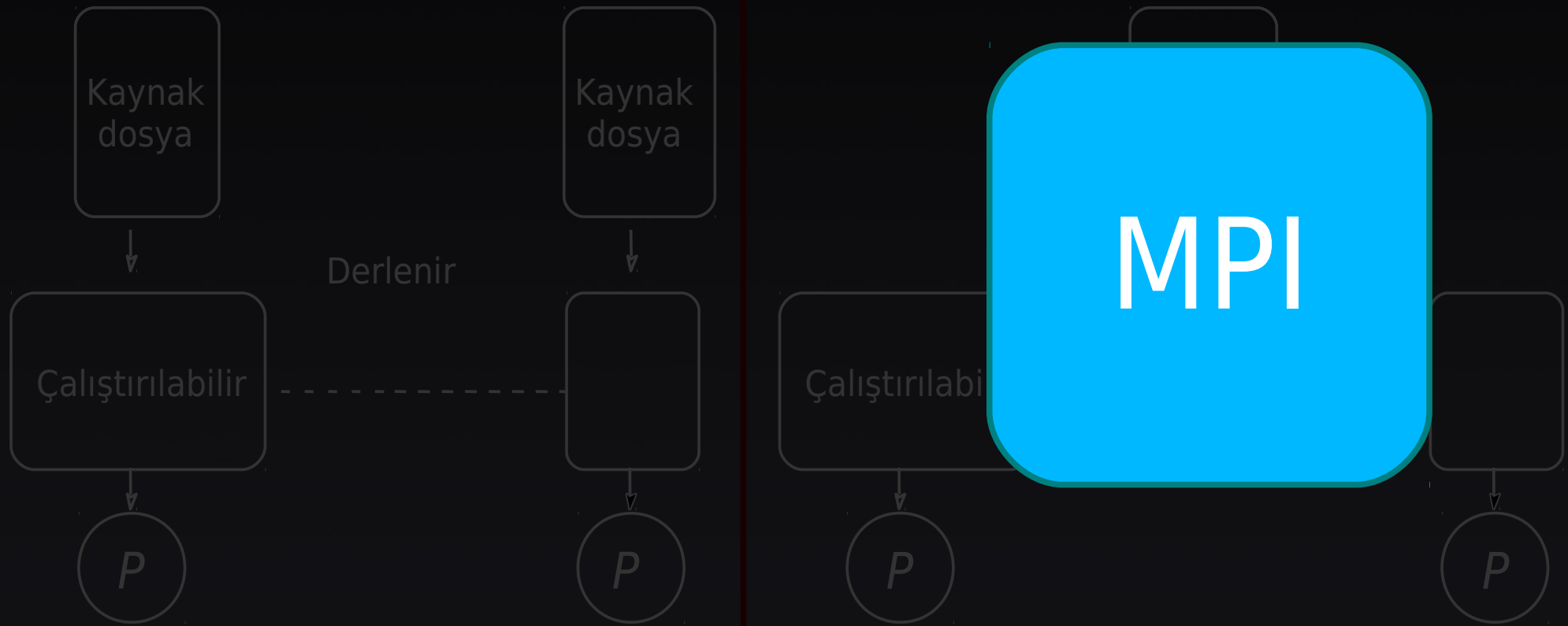
İleti Temelli Hesaplama

- Programlama modelleri:

MPMD

ya da

SPMD



İleti Temelli Hesaplama - 2

Önemli rutinler:

- İletişim rutinleri
 - Noktadan-noktaya gönder/al rutinleri
 - Senkron (synchronous)
 - Asenkron (asynchronous)
 - Bloke eden (blocking)
 - Bloke etmeyen (nonblocking)
 - Grup içi gönder/al rutinleri
 - Yayın (broadcast)
 - Bölerek gönderme (scatter)
 - Toplayarak alma (gather)
- Senkronizasyon rutinleri
 - Barrier

MPI

(Message Passing Interface)

- Bir grup akademisyen ve çeşitli şirketlerden katılımcılar tarafından
 - Yaygın kullanım ve
 - Taşınabilirlik

amaçlanarak oluşturulan bir kütüphane standardıdır.

- Sadece rutinleri belirler, implementasyonla alakası yoktur.
- Zibille bedava ve de özgür implementasyonu mevcut
- Sürüm 1'de sadece statik süreç oluşturma vardı
- Sürüm 2'de dinamik olarak da kullanılabilen

MPI - Örnek

```
int main (int argc, char *argv[])
{
    MPI_Init(&argc, &argv);
    .
    .
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /* kimim ben? */
    if (myrank == 0)
        master();
    else
        slave();
    .
    .
    MPI_Finalize();
}
```

MPI – Diğer Rutinler

- `MPI_Send(buf, count, datatype, dest, tag, comm)`
 - Bloke eden gönder rutini
- `MPI_Recv(buf, count, datatype, src, tag, comm, status)`
 - Bloke eden alma rutini
- `MPI_Isend (...)` / `MPI_Irecv(...)` - bloke etmeyen gönder/al
 - `MPI_Wait()` ve `MPI_Test()` ile denetlenebilirler
- `MPI_Bcast()` / `MPI_Gather()` / `MPI_Scatter()`
- `MPI_Reduce()` - `MPI_Gather` gibi ama bir işlem yaparak alır
- `MPI_Barrier()` - Grup senkronizasyonu *
- `MPI_SendRecv()` - İkili senkronizasyon

MPI - İmplementasyonlar

- MPICH
 - www-unix.mcs.anl.gov/mpi/mpich
- LAM-MPI
 - www.lam-mpi.org
- OpenMPI
 - www.open-mpi.org
 - Optimum MPI projesi
 - Pardus pakedi mevcut
- FT-MPI, LA-MPI, PACX-MPI...
- Fortran, C++, Python, Java vs. bindingleri mevcut

MPI – Örnek program

```
#include "mpi.h"

#include <stdio.h>

#include <math.h>

#define MAXSIZE 1000

void main(int argc, char **argv)

{

    int myid, numprocs;

    int data[MAXSIZE], i, x, low, high, myresult, result;

    char fn[255];

    char *fp;

    MPI_Init(&argc,&argv);

    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);

    MPI_Comm_rank(MPI_COMM_WORLD,&myid);

    if (myid == 0)

    { /* Open input file and initialize data */

        strcpy(fn,getenv("HOME"));

        strcat(fn,"/MPI/rand_data.txt");

        if ((fp = fopen(fn,"r")) == NULL) {

            printf("Can't open the input file: %s\n\n", fn);

            exit(1);

        }

        for(i = 0; i < MAXSIZE; i++)

            fscanf(fp,"%d", &data[i]);

    }

    MPI_Bcast(data, MAXSIZE, MPI_INT, 0, MPI_COMM_WORLD);

    /* broadcast data */

    x = MAXSIZE/numprocs; /* Add my portion Of data */

    low = myid * x;

    high = low + x;

    myresult = 0;

    for(i = low; i < high; i++)

        myresult += data[i];

    printf("I got %d from %d\n", myresult, myid);

    /* Compute global sum */

    MPI_Reduce(&myresult, &result, 1, MPI_INT, MPI_SUM, 0,

              MPI_COMM_WORLD);

    if (myid == 0)

        printf("The sum is %d.\n", result);

    MPI_Finalize();

}
```

MPI – Derleme & Çalıştırma

- ssh / rsh hazırlanır
- Hesaplama da kullanılacak bilgisayarların listesi (hostfile) hazırlanır
- mpicc ile programlar derlenir
- mpirun ile çalıştırılır

Bağlantılar

- Barry Wilkinson, Michael Allen - Parallel Programming kitabının sitesi
 - http://www.cs.uncc.edu/~abw/parallel/par_prog/
- Wikipedia
 - http://en.wikipedia.org/wiki/Parallel_Computing
- MPI: Complete Reference
 - http://www.csd.uoc.gr/~hy555/mpi/mpi_complete_reference.pdf

Sorular ?

Teşekkürler

Korsan değil, özgür yazılım!