Bölüm 2: Yapılar

İşletim Sistemleri

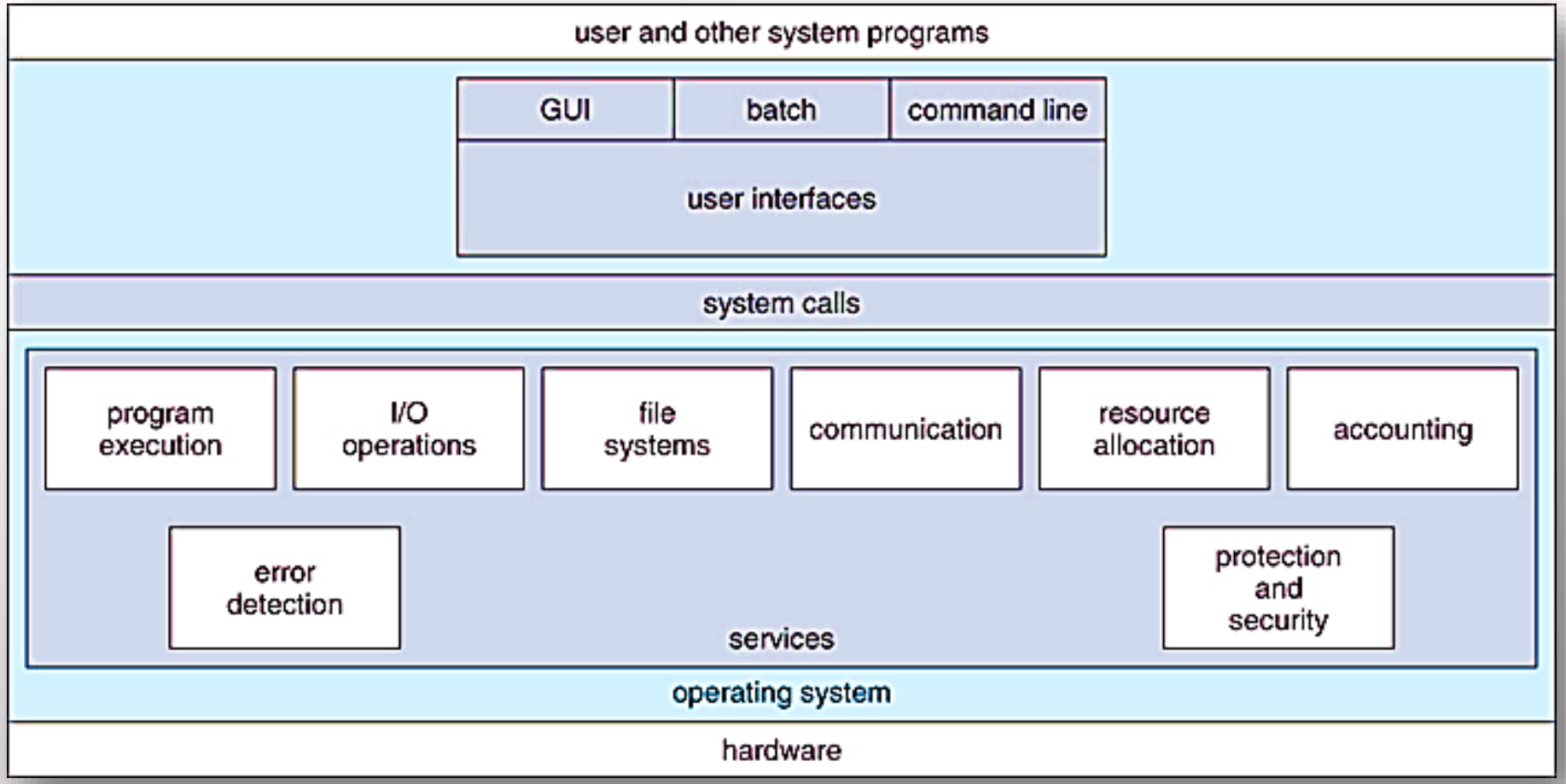
İşletim Sistemi Yapıları

- Farklı işletim sistemi bileşenleri,
  - çekirdek içinde nasıl entegre edilecek?
  - bu strateji, işletim sistemi yapısı olarak adlandırılır.
- Bu yapı,
  - işletim sisteminin nasıl çalıştığını ve
  - bileşenlerinin birbiriyle nasıl etkileşime girdiğini belirler.

Neden Yapılandırma Önemli?

- İşletim sistemi, uygulamaların donanım ile iletişim kurmasını sağlar.
  - Karmaşık tasarım,
  - Kullanım kolaylığı,
  - Değiştirilebilirlik ihtiyacı,
  - İşletim sistemini özenle inşa etmeyi gerektirir.
- Her bir yapı,
  - Bazı avantajlar sunar ve
  - Belirli kullanım senaryolarına uygun çözümler sağlar.

İşletim Sistemi Servisleri



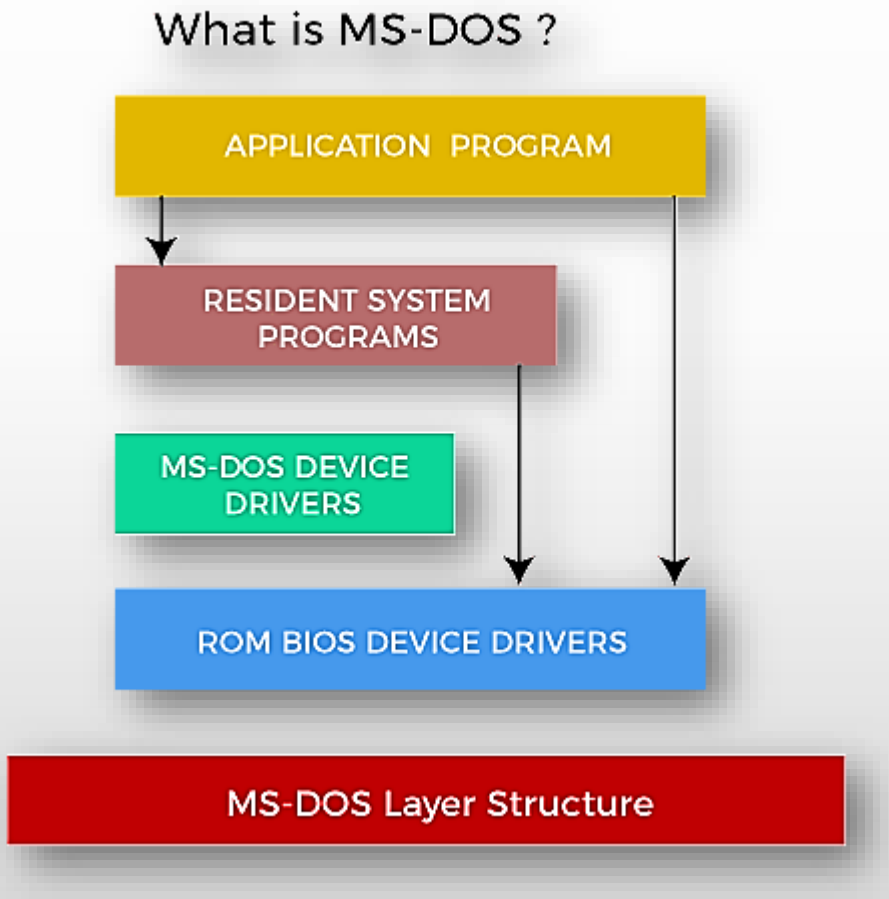
İşletim Sistemi Yapıları

- Basit/Monolitik Yapı (*Simple/Monolithic Structure*)
- Mikroçekirdek Yapı (*Micro-Kernel Structure*)
- Hibrit-Çekirdek Yapı (*Hybrid-Kernel Structure*)
- Exo-Çekirdek Yapı (*Exo-Kernel Structure*)
- Katmanlı Yapı (*Layered Structure*)
- Modüler Yapı (*Modular Structure*)
- Sanal Makineler (*Virtual Machines*)

Basit/Monolitik Yapı

- Belirgin bir yapı yok, küçük, basit ve sınırlı.
- Arayüzler ve işlevsellik seviyeleri iyi bir şekilde ayrılmamış.
- MS-DOS, bu tip bir işletim sistemine örnektir.
- MS-DOS'ta, uygulama programları temel giriş/çıkış rutinlerine erişebilir.
- Kullanıcı programlarından biri tüm sistemi çökertebilir.
- Modüller arasında belirgin sınırlar olmadığı için yapı karmaşık.
- İşletim sistemi içinde veri gizlemeyi (*data hiding*) zorunlu kılmaz.

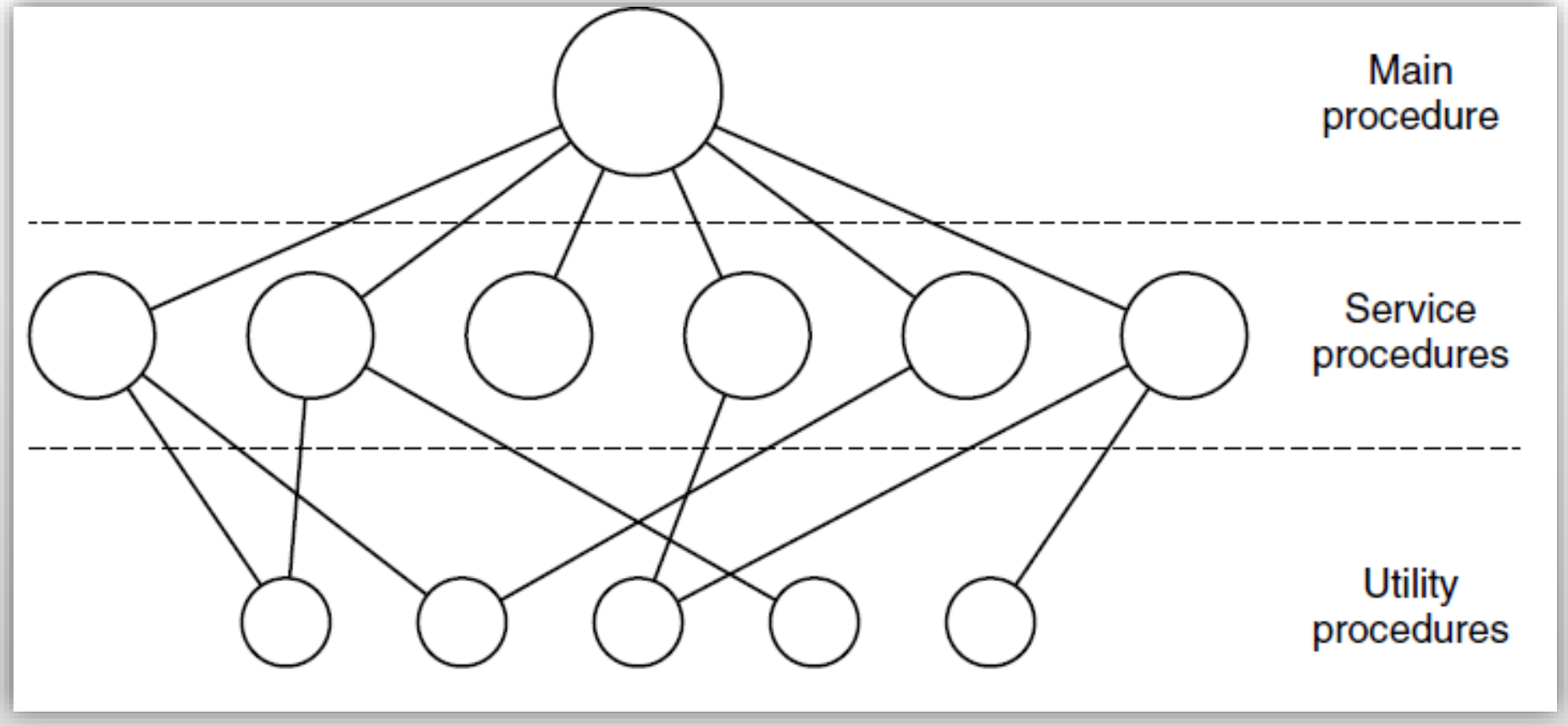
Basit/Monolitik Yapı



Monolitik Sistem

- İşletim sisteminin temel yapısı,
  - İstenen hizmet prosedürünü başlatan *ana program*.
  - Sistem çağrılarını gerçekleştiren bir dizi *hizmet prosedürü*.
  - Hizmet prosedürlerine yardımcı olan bir dizi *yardımcı prosedür*.

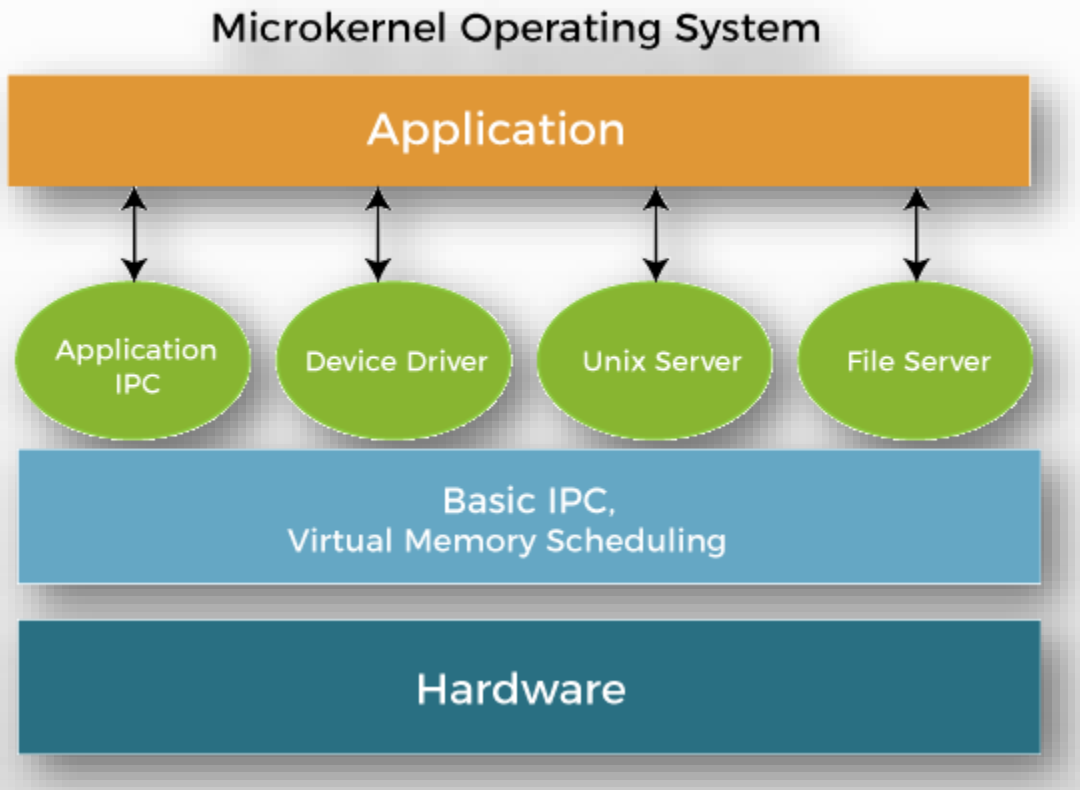
Monolitik Sistem Yapısı



Mikroçekirdek Yapı

- Gereksiz bileşenler çekirdekten çıkarılarak tasarlanır.
- Çıkan bileşenler sistem ve kullanıcı programları olarak uygulanır.
- Daha küçük bir çekirdek (*micro-kernel*) elde edilir.
- Tüm yeni hizmetler kullanıcı alanına eklenir, çekirdek değişmeden kalır.
- Daha güvenli ve güvenilir bir yapı sunar.
- Mac OS, bu tür bir işletim sistemine örnektir.
- İşletim sistemini çeşitli platformlara taşınabilir hale getirir.
- Microkernel'lar küçük olduğu için etkili bir şekilde test edilebilir.
- Modül arası iletişimin artması sistem performansını düşürebilir.

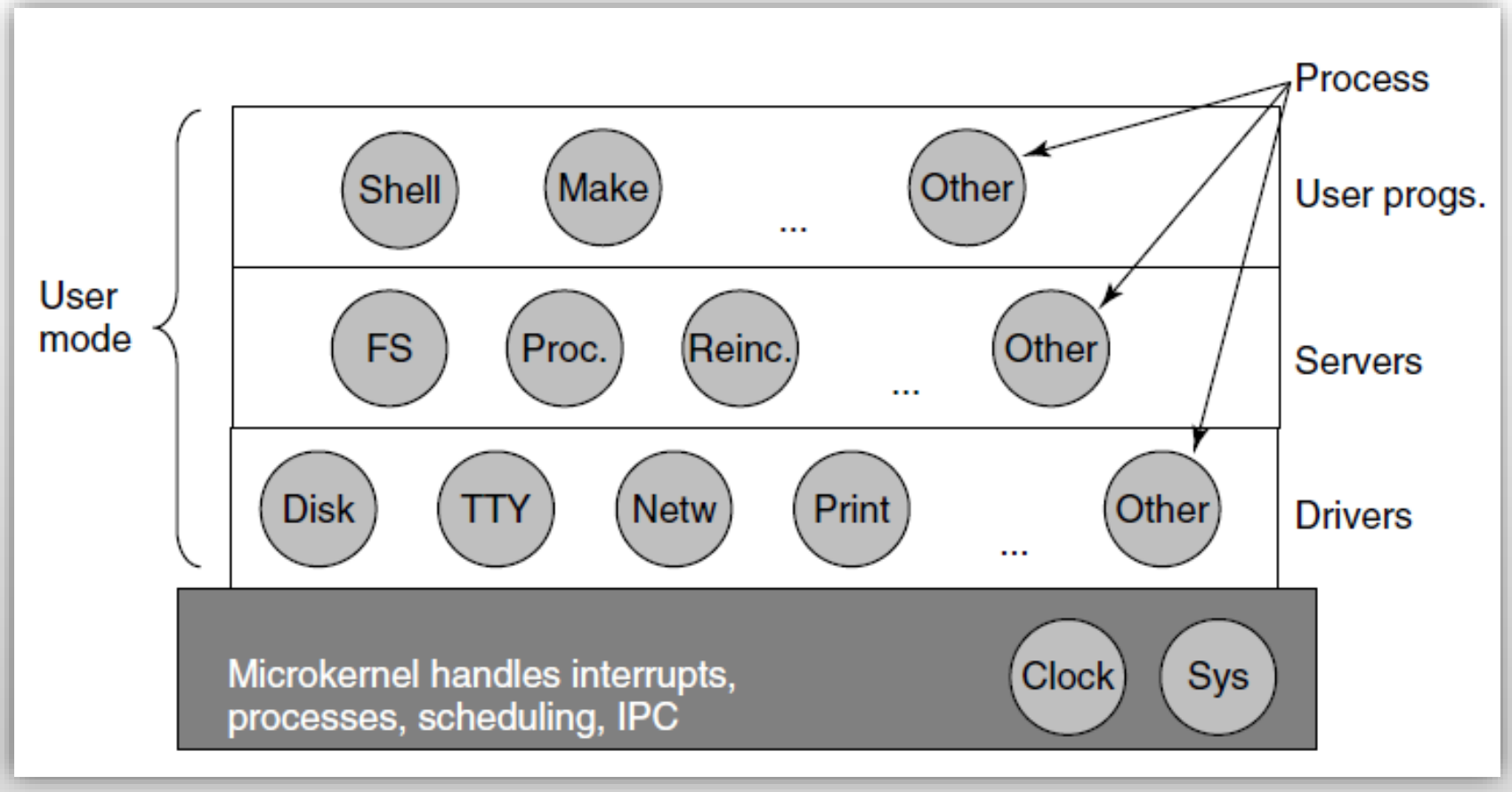
Mikroçekirdek Yapı



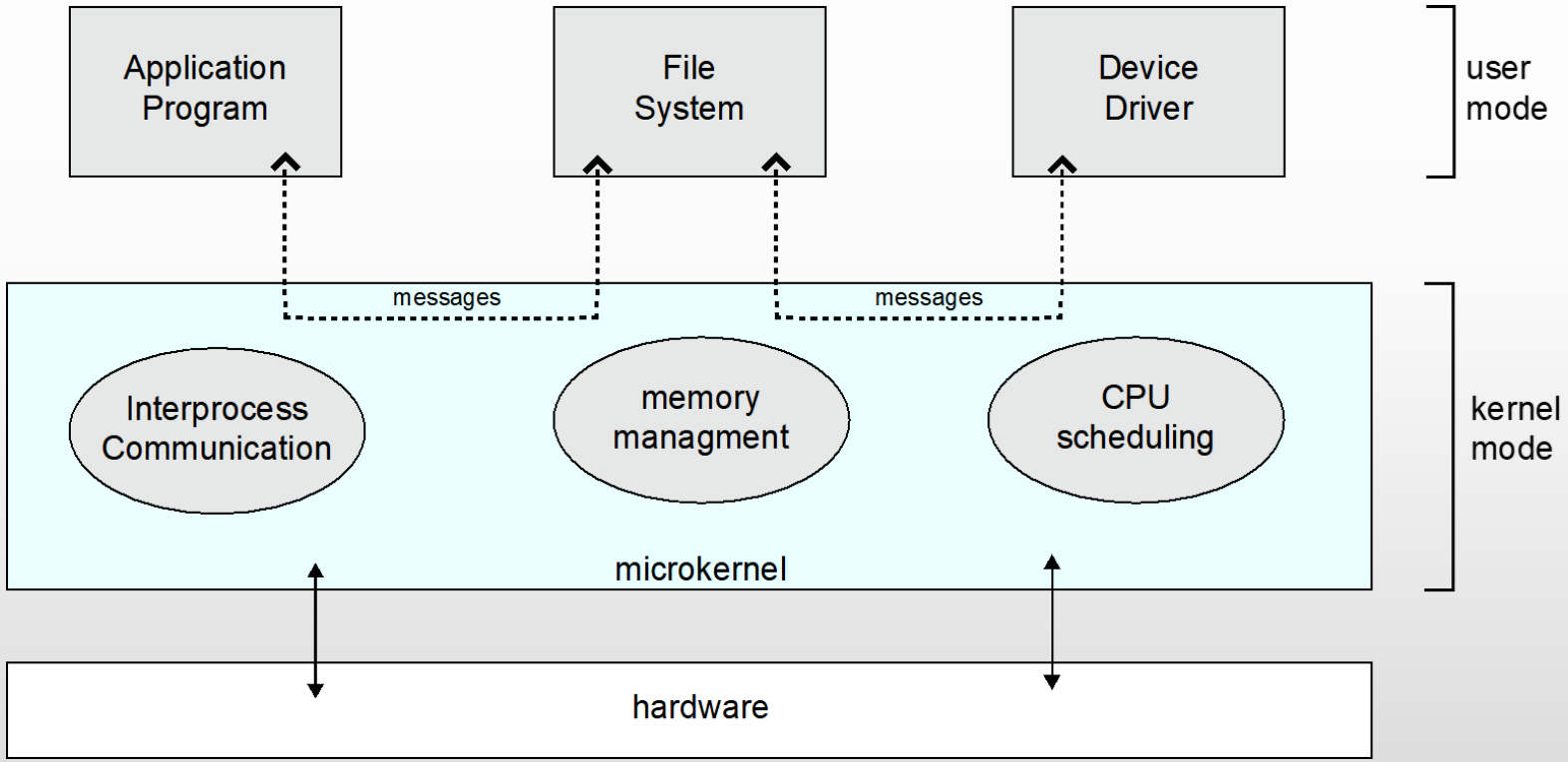
Mikro Çekirdek

- Çekirdekte az sayıda sürecin yürütülmesine izin verilir.
- Hataların etkilerini en aza indirir.
  - Sürücüdeki bir hatanın sistemi çökertmesi istenmez.
- Mekanizma çekirdekte, ilke (policy) çekirdeğin dışındadır.
  - Mekanizma, süreçler önceliklerine göre çizelgelenir (*kernel*).
  - İlke, süreç öncelikleri kullanıcı modunda tanımlanır (*user*).

Mikro Çekirdek Sistem Yapısı



Mikro Çekirdek Sistem Yapısı



Andrew Tanenbaum (Monolitik çekirdek)

- Monolitik çekirdeklerin tasarımı ve uygulanması daha basittir.
- Daha bütünleşik bir sistem sağlar.
- Doğrudan işlev çağrıları daha iyi performans sağlar.
- Süreçler arası iletişim gerektirmez.
- Çekirdekteki hatalar tüm sistemi çökertebileceğinden risklidir.!
- Her şeyin tek modülde olması hataları bulmayı ve düzeltmeyi zorlaştırır.!

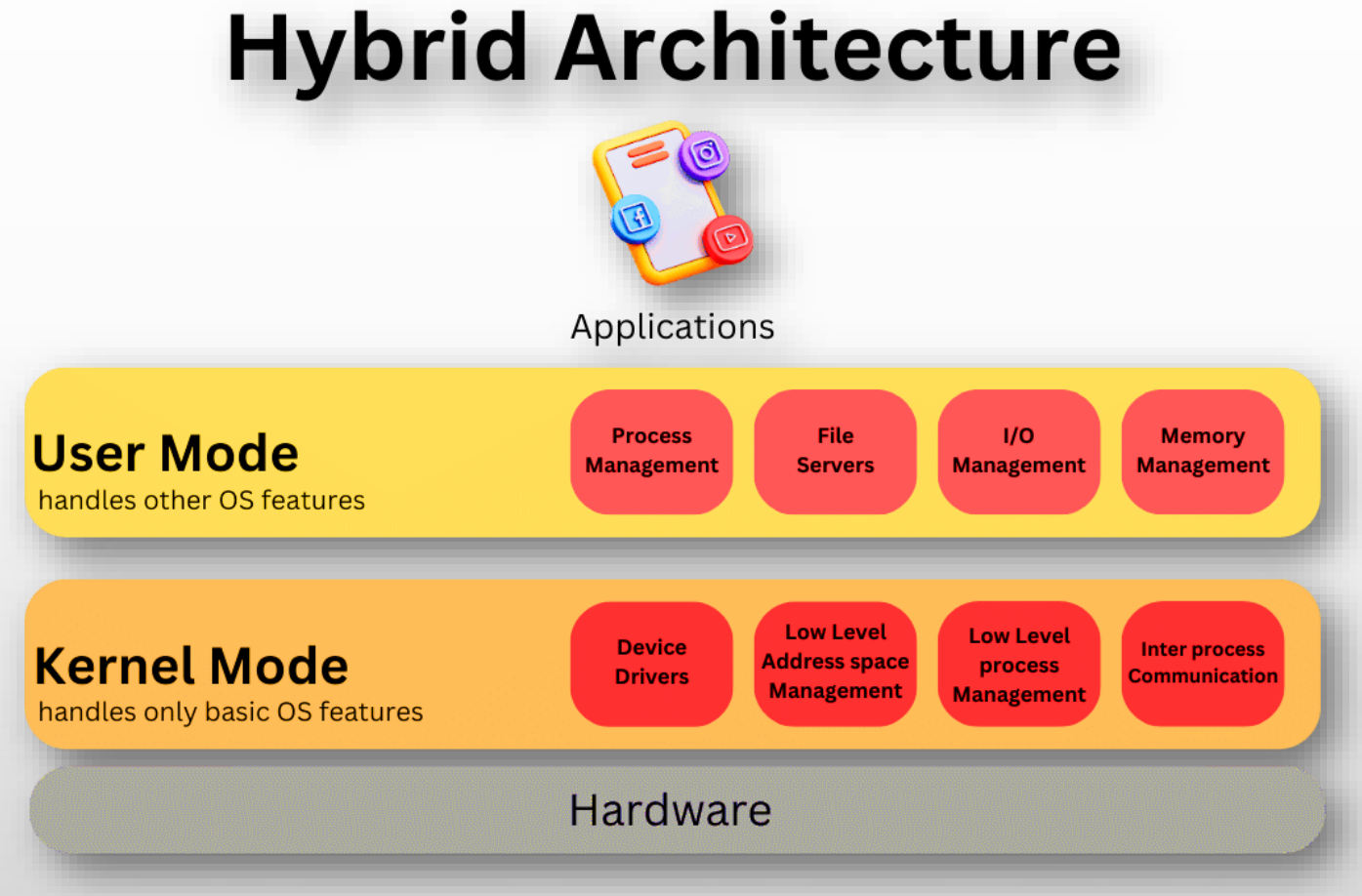
Linus Torvalds (Mikro Çekirdek)

- Mikro çekirdekler daha modüler, esnek ve ölçeklenebilirdir.
- Daha kolay korunabilir ve geliştirilebilir.
- Herhangi bir bileşendeki hatanın verebileceği hasarı sınırlar.
- Daha kararlı ve güvenilirdir.
- Modern bilgisayar mimarileri, mikro çekirdeklerin performans sorunlarının üstesinden gelebilir.!
- Uygun şekilde tasarlanırsa daha iyi performans sunabilir.
- Linux'un açık kaynak geliştirme modeli, hızla gelişmesine ve yaygın olarak kullanılmasına yardımcı oldu.

Hibrit-Çekirdek Yapı

- Monolitik yapının hız ve tasarımı ile Mikro-çekirdek yapının modülerliği ve kararlılığını birleştirir.
- Kritik işlevleri micro-çekirdek içinden ayırarak hata ayıklama ve bakım için genel sistem güvenilirliğini artırır.
- Micro-çekirdek yaklaşımını benimseyerek daha iyi izolasyon ve güvenlik sağlar.

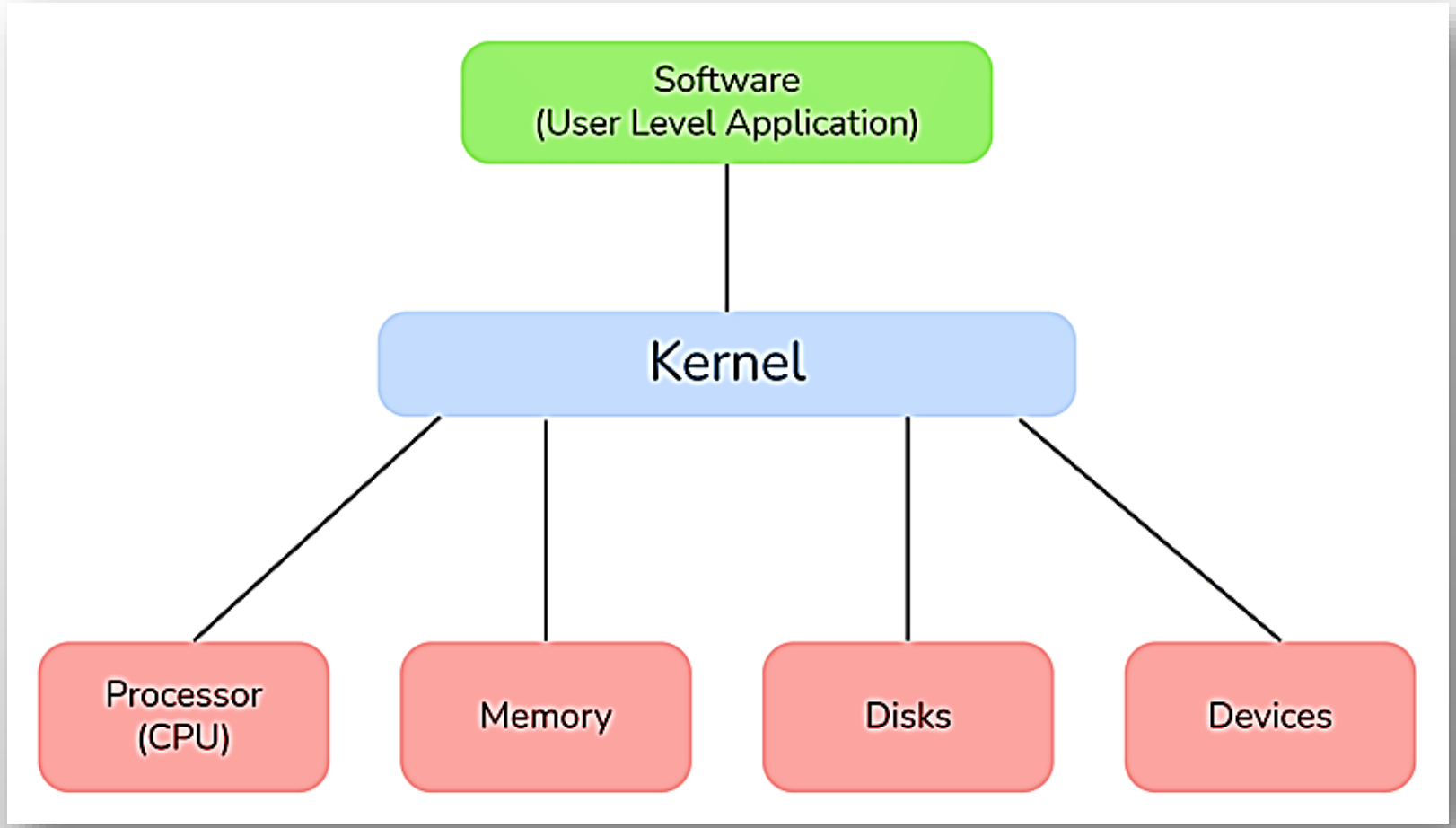
Hibrit-Çekirdek Yapı



Exo-Çekirdek Yapı

- MIT tarafından geliştirilen bir işletim sistemidir.
- Donanım kaynakları uygulama düzeyinde yönetilir.
- İyileştirilmiş uygulama kontrolü sağlar.
- Yönetimi güvenlikten ayırır.
- Uygulamanın performansını artırır.
- Uygulamaların özel bellek yönetim sistemi kullanmasına olanak tanır.

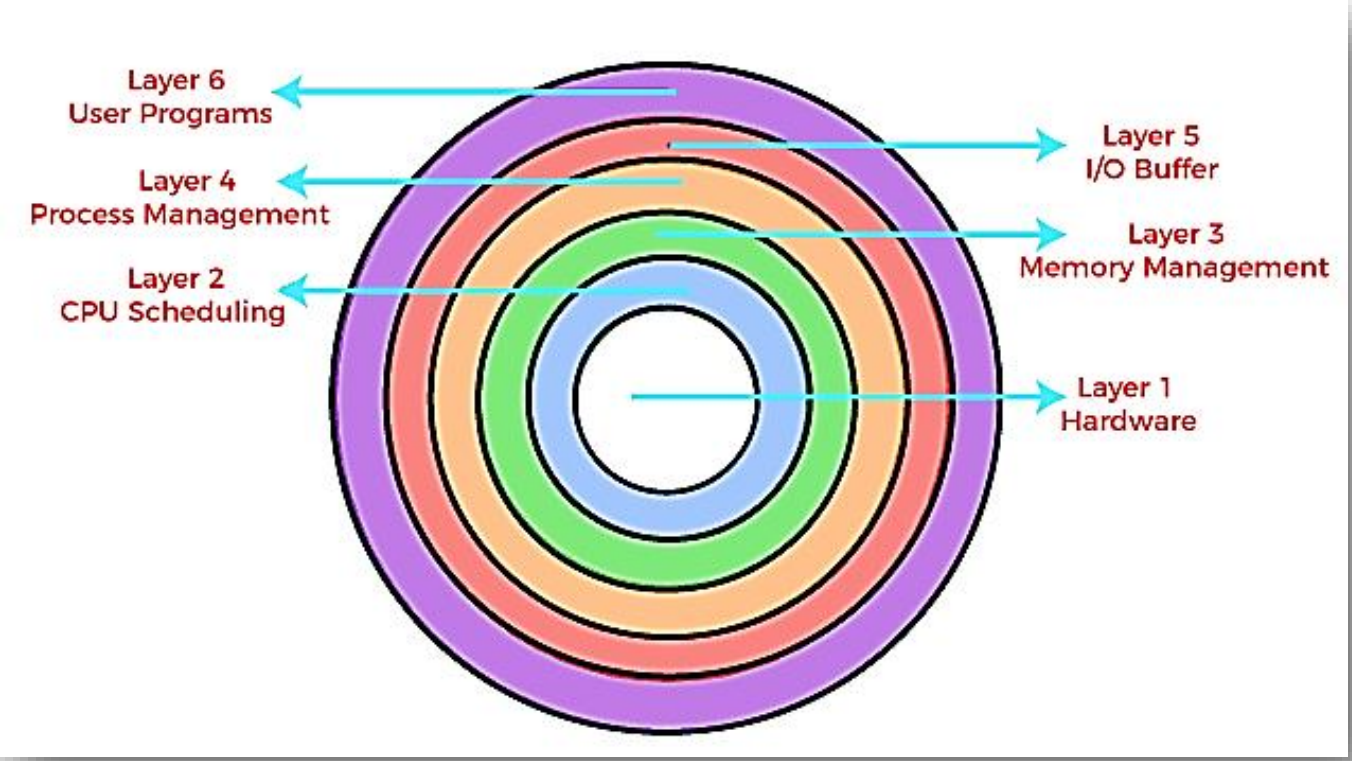
Exo-Çekirdek Yapı



Katmanlı Yapı

- İşletim sistemi bir dizi katmana (seviye) bölünür.
- En alt katman donanım, en üst katman kullanıcı arayüzüdür.
- Her katman, alt seviyedeki katmanların işlevlerini kullanır.
- Hata ayıklama ve sistem doğrulama kolaydır.
- Hata sadece ilgili katmanda aranır.
- Her katmanda veri değiştirilerek diğer katmana iletilir.

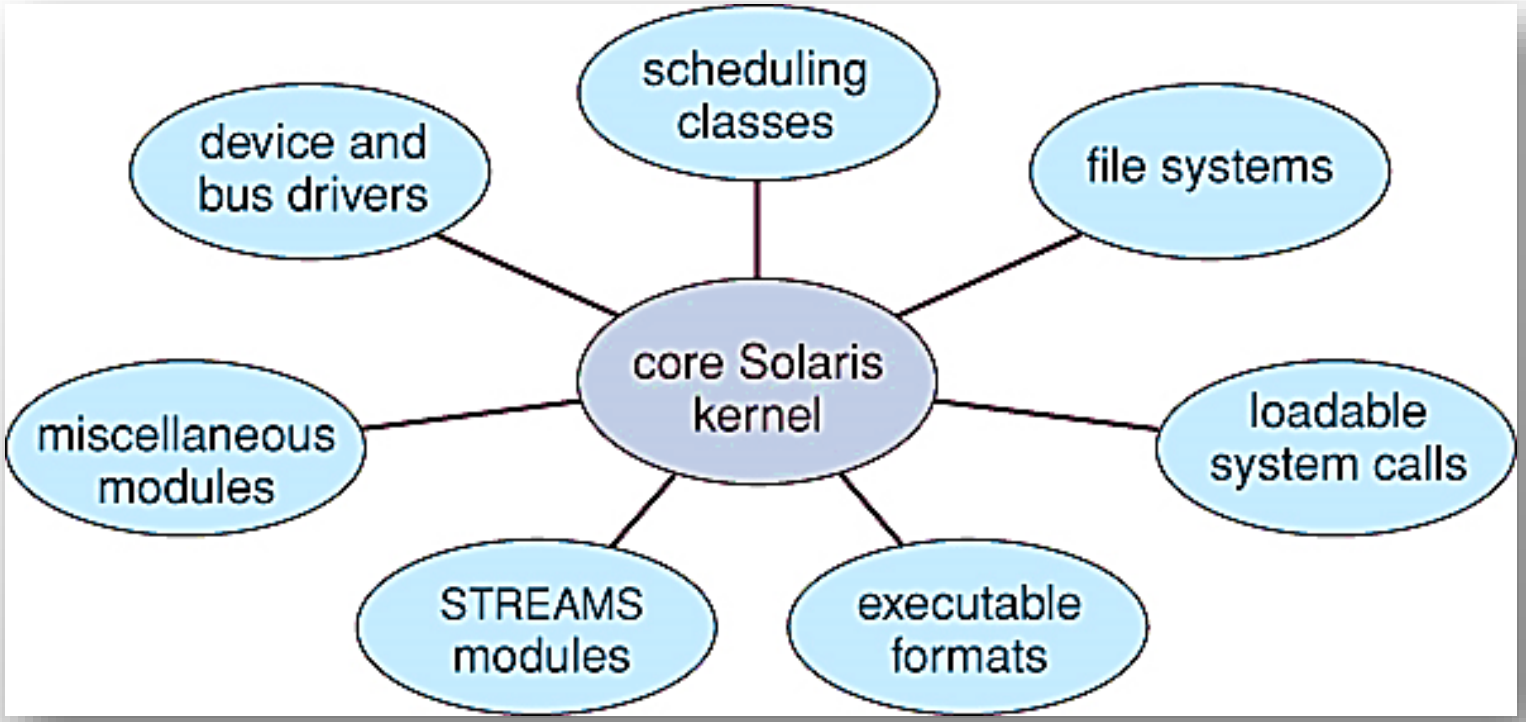
Katmanlı Yapı



Modüler Yapı

- İşletim sistemi için en iyi yaklaşım olarak kabul edilir.
- Çekirdek, yalnızca belirli temel bileşenlere sahiptir.
- Hizmetler, çekirdeğe çalışma zamanında veya önyükleme sırasında modül olarak yüklenir.
- Katmanlı yapıya benzer, çekirdek tanımlanmış arayüzlere sahip.
- Bir modül başka bir modülü çağırabilir, katmanlı yapıdan daha esnek.
- Örneğin, Solaris OS bu yapıya sahip.
- Her modülün birbiriyle uyumlu olması *zor*!

Modüler Yapı



Zamanlayıcı (timer)

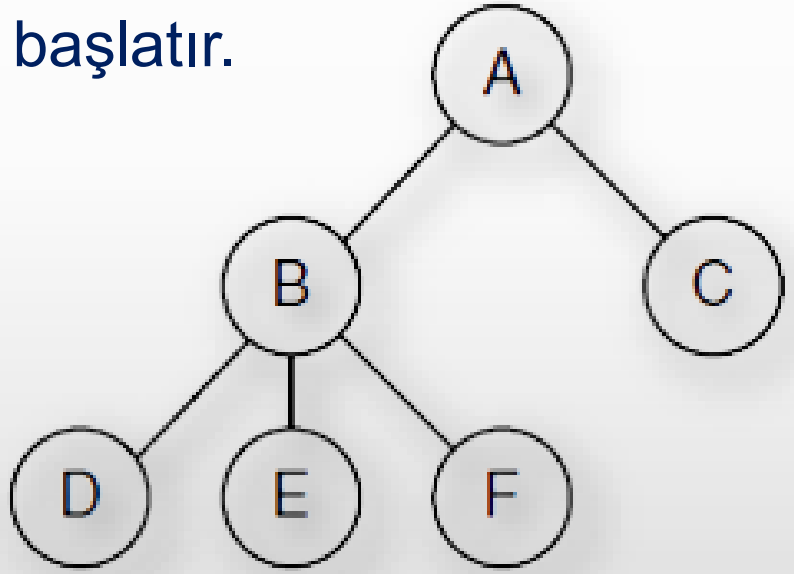
- Sonsuz döngü hatasını önlemek için kullanılır.
- Belirli bir süre dolduğunda kesme yaratır.
  - Fiziksel saat (*clock*) tarafından azaltılan bir sayaç tutulur.
  - İşletim sistemi sayaca değer atar (*ayrıcalıklı (privileged) talimat*).
  - Sayaç sıfır olduğunda kesme üretilir.
  - İşletim sistemi kontrolü tekrar kazanır.
  - Kendisine ayrılan süreyi aşan görevleri sonlandırır.

Süreçler

- İşletim sistemi tarafından yürütülen programlardır.
- Bir programın çalıştırılabilmesi için gerekli tüm bilgiyi tutar.
- Süreç tablosunda tutulurlar.
- İşletim sistemi tarafından atanmış kaynaklar (bellek, CPU) ile ilişkilidir.
- Diğer süreçler ile konuşabilirler (*IPC*).
- Bellekte saklanır ve yürütülürler.
- Adres uzayı ile ilişkilidir.
- **Adres uzayı**: yürütülebilir program, programın verileri ve yığını
- Yazmaçlar, dosyalar, alarmlar, ilgili süreçler…

Süreçler

- Süreç ağacı.
- A süreci, B ve C olmak üzere 2 çocuk süreç başlatır.
- B süreci D, E ve F olmak üzere 3 çocuk süreç başlatır.



Adres Uzayı

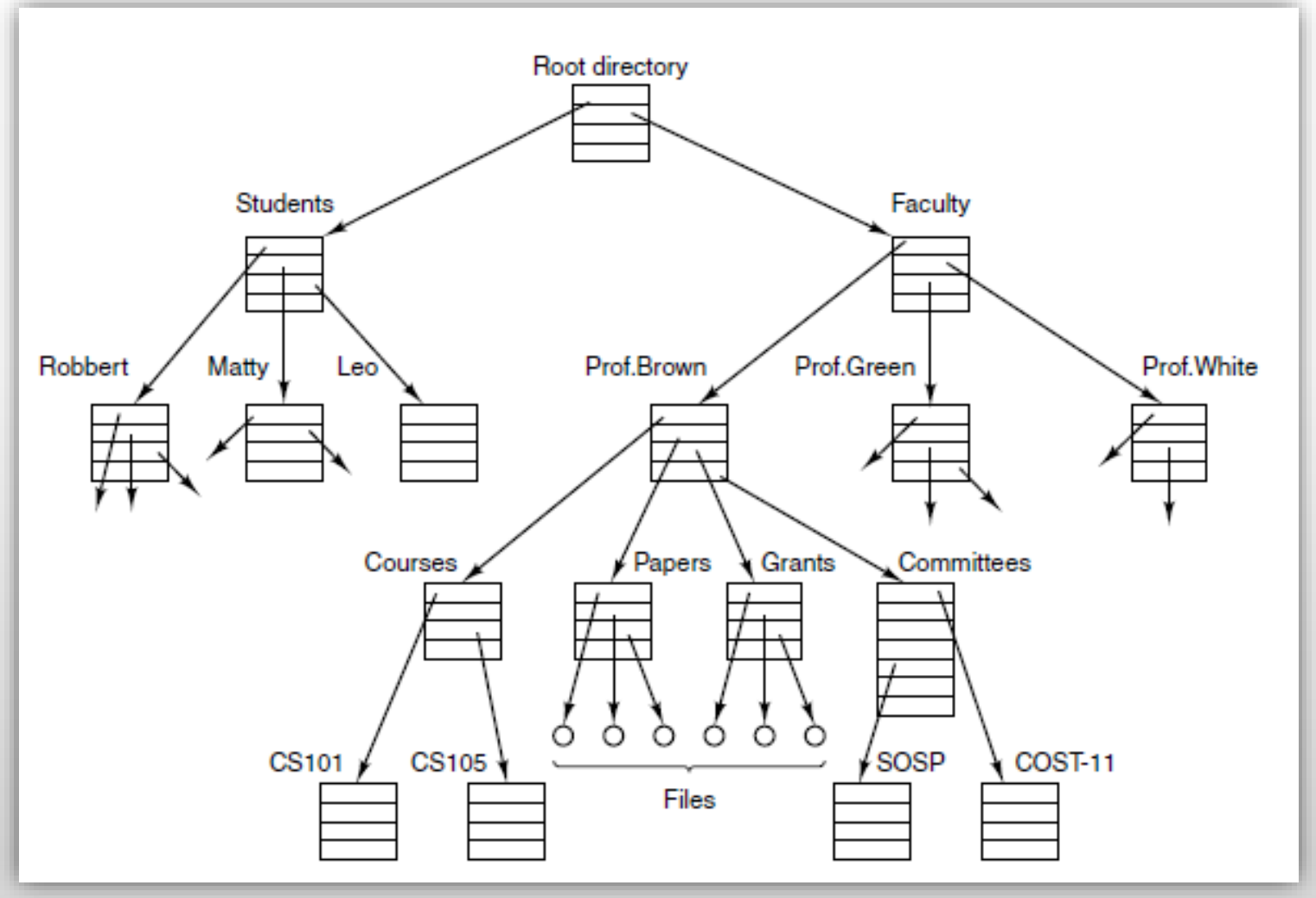
- Bir süreç tarafından kullanılan bellek alanı.
- Bellekte aynı anda birden çok süreç bulunur.
- Bazı süreçler, fiziksel olarak mevcut olandan fazla belleğe ihtiyaç duyar.
- Bu durumda, sanal bellek (*virtual memory*) kavramı devreye girer.

Dosyalar

- Verileri kalıcı olarak depolayan birimler.
- Dosyalar, veri, metin, resim, video, ses ve diğer türlerde olabilir.
- İşletim sistemi tarafından yönetilir.
- İşletim sistemi tarafından belirlenen dizin yapısına göre saklanır.
- Kullanıcılar tarafından erişilebilir, okunabilir, yazılabilir veya silinebilir.
- Blok tabanlı (disk), karakter tabanlı (yazıcı, modem) olabilir.

Dosyalar

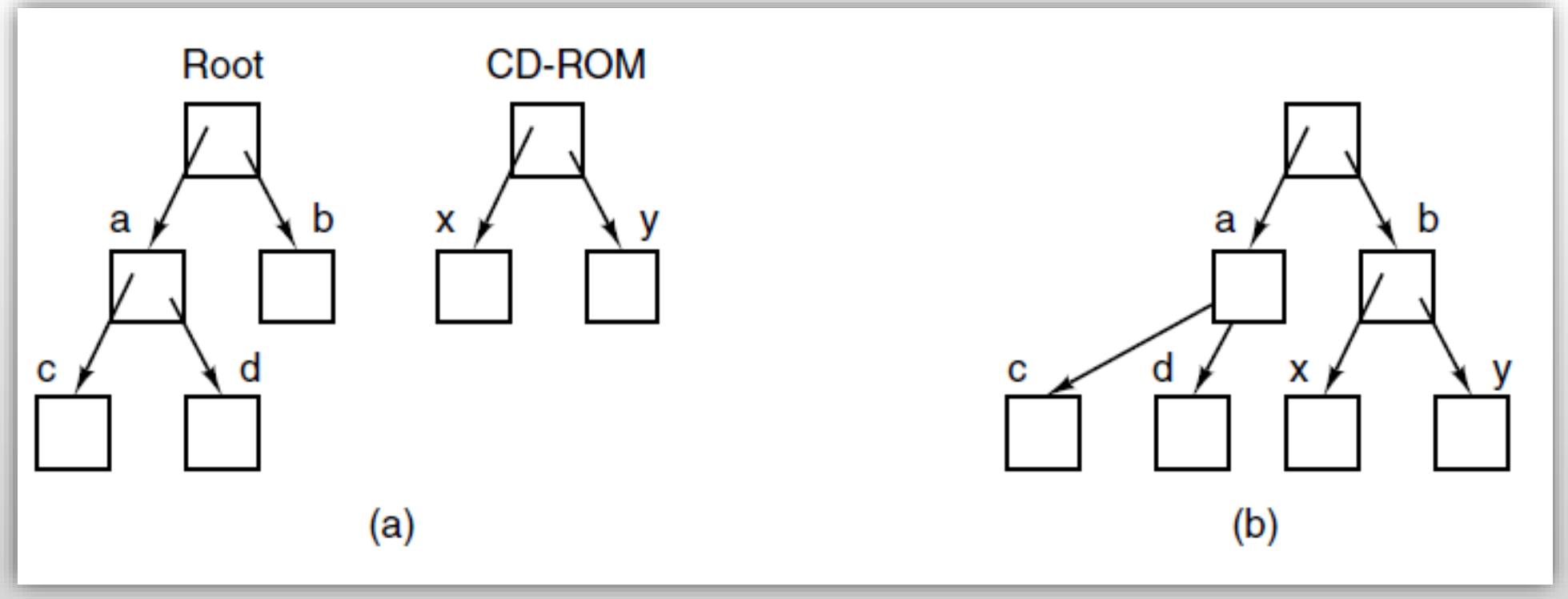
- Örnek dosya sistemi.



Dosyalar - Bağlama İşlemi

(a) Bağlamadan (*mount*) önce, CD-ROM'daki dosyalara erişilemez.

(b) Bağlandıktan sonra, dosya hiyerarşisinin bir parçasıdırlar.

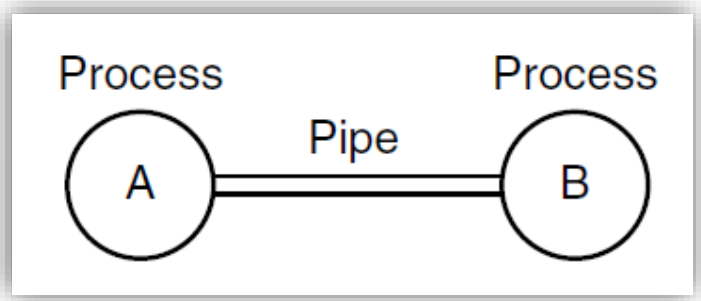


Dosya Bağlama (mount)

- *mount*, dosya sistemlerini bağlamak ve yönetmek için kullanılan komut.
- Yeni bir depolama birimini sistem içine entegre eder.
- `mount /dev/sdb1 /mnt` - /dev/sdb1'i /mnt dizinine bağlar.
- **ro (read-only)**: Salt okunur bağlama.
- **rw (read-write)**: Okuma ve yazma izni ile bağlama.

Dosyalar

- 2 süreç boru hattı (*pipe*) ile bağlanmış.



- Boru hattı, iki süreci birbirine bağlar.
- Veri gönderip alarak iletişim kurmalarını sağlayan tek yönlü veri kanalıdır.
- **Anonim (*anonymous*)**, ata ve çocuk süreçler arasında *IPC* için kullanılır.
- **Adlandırılmış (*named*)**, iki ayrı süreç arasında *IPC* için kullanılır.

Boru Hattı Örneği (pipe)

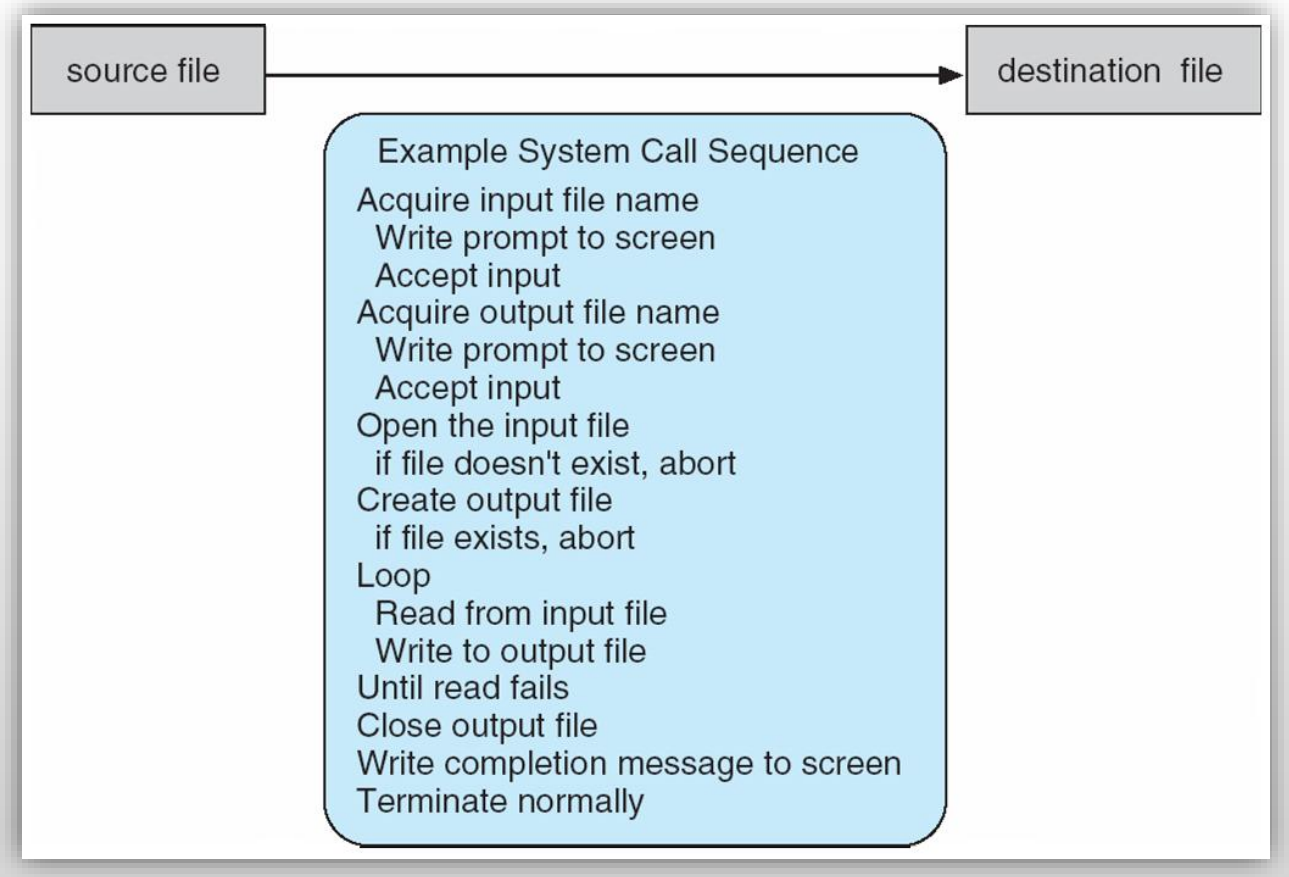
- `ls -l | grep "anahtar_kelime"`
- ls komutu ile listelenen dosya/dizinlerin çıktısı,
- grep komutu ile filtrelenir,
- anahtar kelimeyi içeren satırlar ekrana yazdırılır.
- *ls -l*, çıktıyı bir *pipe* (|) aracılığıyla *grep* komutuna iletir.

Sistem Çağrıları

- İşletim sistemi tarafından sağlanan hizmetlere erişmek için kullanılır.
  - Örneğin, dosya işlemleri, bellek yönetimi, zaman hizmetleri, vb.
- İşletim sistemi tarafından tanımlanmış bir arayüze göre gerçekleştirilir.
- Uygulamalar tarafından kullanılır, işletim sistemi tarafından yürütülür.
- Sistemden sisteme farklılık gösterir, ancak temel kavramlar benzerdir.

Örnek Sistem Çağrısı

- Bir dosyanın içeriğinin başka bir dosyaya aktarılması

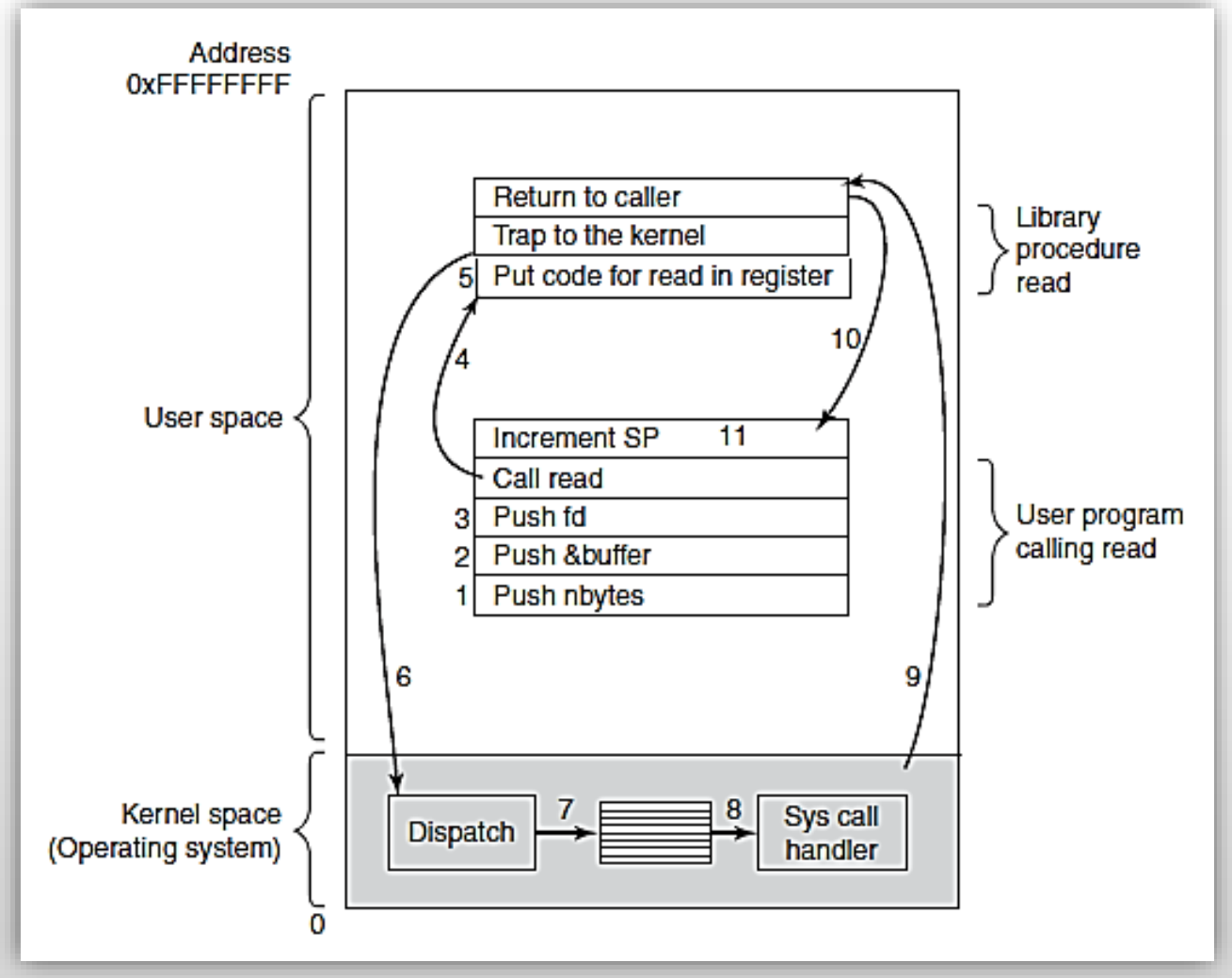


Örnek Sistem Çağrısı

```c
#include <unistd.h> // Header, read ve write fonksiyonlarını içerir
#include <stdio.h>  // Standard I/O fonksiyonlarını içerir
int main() {
    char buffer[100]; // Kullanıcı girdisini saklayan tampon
    // 0 parametresi standart girişi (stdin) temsil eder
    ssize_t bytesRead = read(0, buffer, sizeof(buffer));
    if (bytesRead == -1) {
        perror("okuma hatası");
        return 1; // Hata kodu döndür
    }
    return 0; // Program başarıyla tamamlandı
}
```

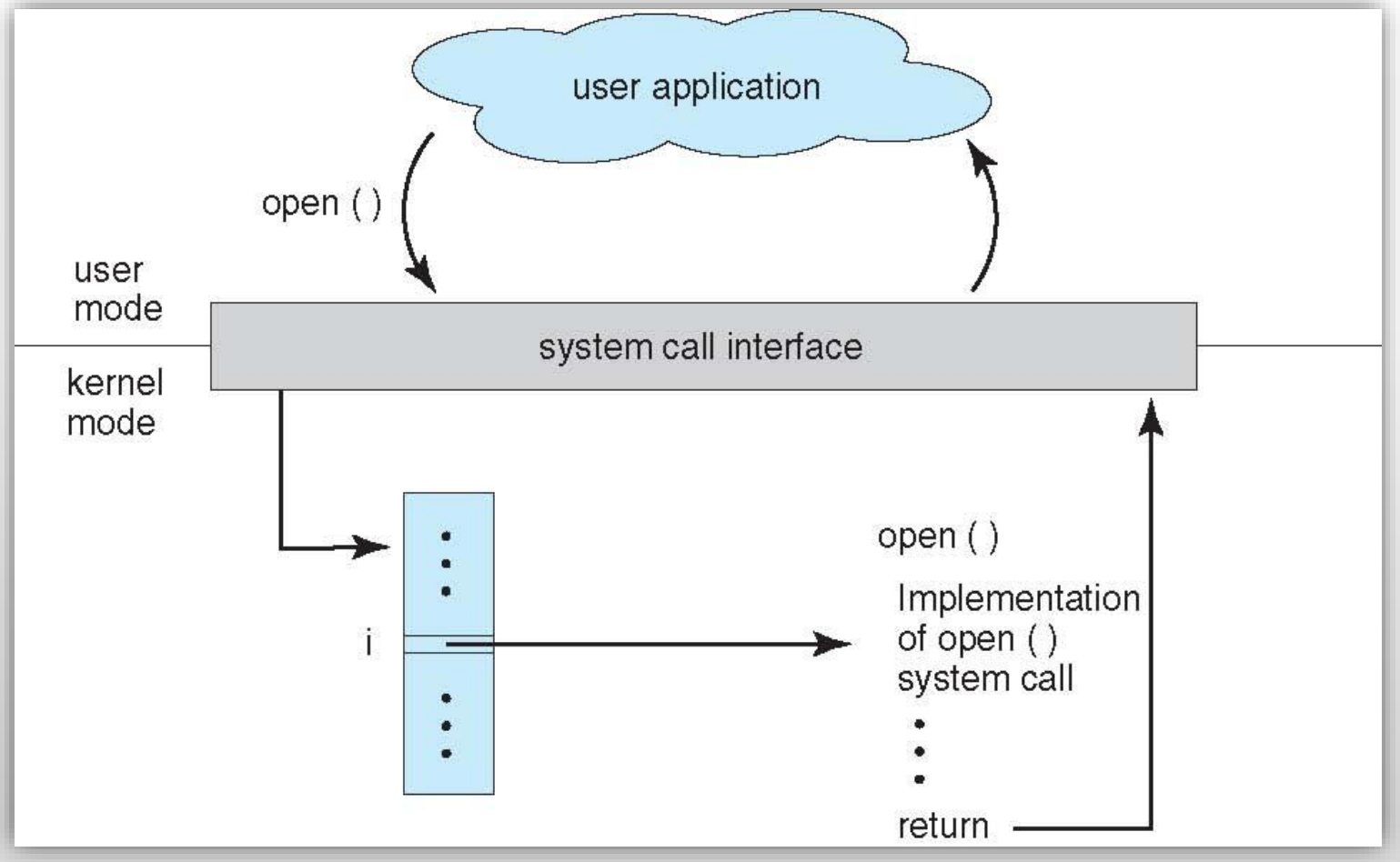
Sistem Çağrıları

- **read(fd, buffer, nbytes)** sistem çağrısının adım adım gösterimi.

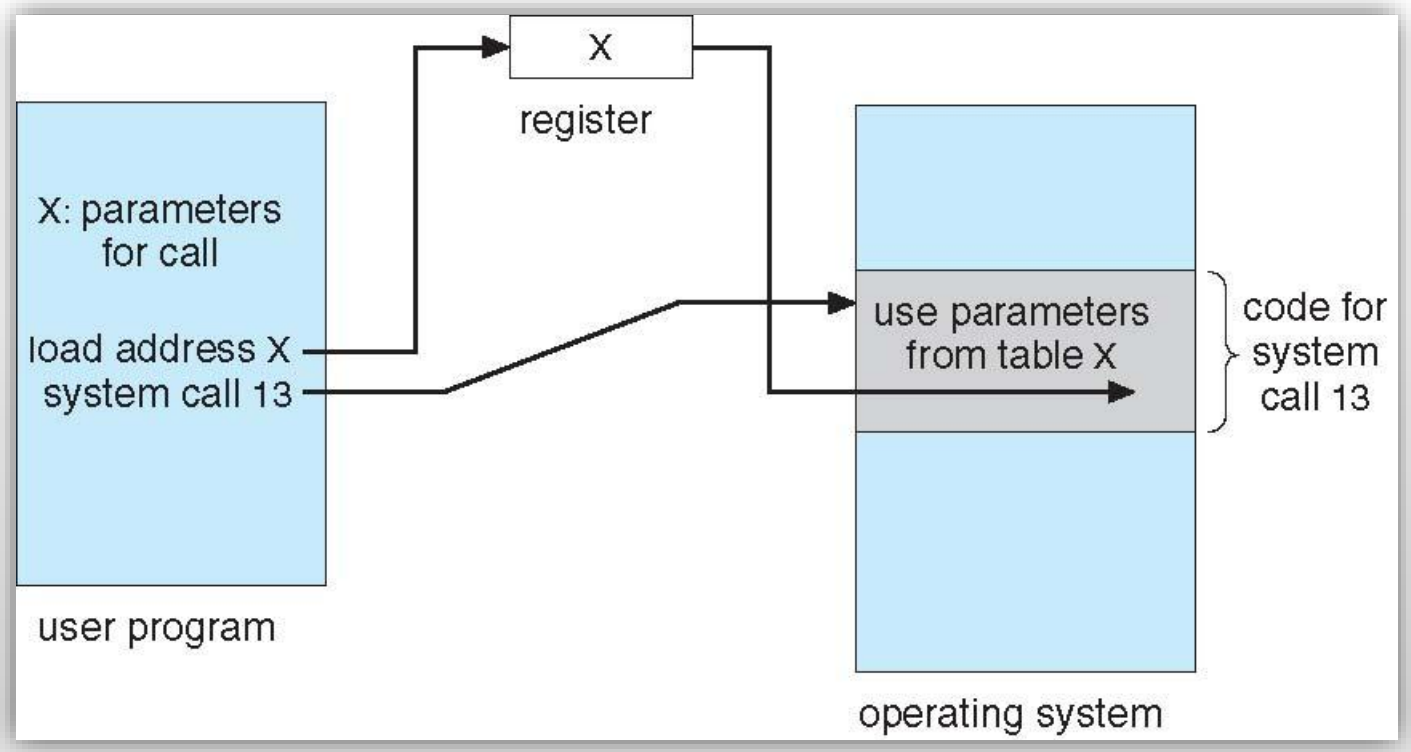


Sistem Çağrıları

- .



Sistem Çağrıları – Parametre Aktarımı



Örnek Sistem Çağrısı

```c
#include <unistd.h> // Header, write fonksiyonunu içerir

int main() {
    // yazdırma işlemi için write sistem çağrısı
    // 1 parametresi standart çıkışı (stdout) temsil eder
    write(1, "Merhaba, Dünya!\n", 15);
    return 0; // Program başarıyla tamamlandı
}
```

Sistem Çağrıları – Süreç Yönetimi

Başlıca POSIX sistem çağrıları. Hata durumunda -1 döner.

pid: işlem kimliği (*process identifier*).

s: geri dönüş kodu (*return code*).

Çağrı	**Tanım**
pid = fork()	Ata süreç ile aynı bir alt süreç oluşturur.
pid = waitpid(pid, &statloc, options)	Çocuk sürecin sonlanmasını bekler.
s = execve(name, argv, environp)	Bir sürecin ana görüntüsünü değiştirir.
exit(status)	Süreci yürütmeyi durdurur ve durumu geri döndürür.

Sistem Çağrıları – Dosya Yönetimi

- **fd**: dosya tanıtıcı (*file descriptor*),
- **n**: bayt sayısı,
- **position**: dosya içinde görece konum (*offset*).

Çağrı	Tanım
fd = open(file, how,...)	Okuma, yazma veya her ikisi için bir dosya açar.
s = close(fd)	Açık bir dosyayı kapatır.
n = read(fd, buffer, nbytes)	Bir dosyadan arabelleğe veri okur.
n = write(fd, buffer, nbytes)	Arabellekten bir dosyaya veri yazar.
position = lseek(fd, offset, whence)	Dosya işaretçisini hareket ettirir.
s = stat(name, &buf)	Bir dosyanın durum bilgisini alır.

Sistem Çağrıları – Dizin Yönetimi

- .

Çağrı	Tanım
s = mkdir(name, mode)	Yeni bir dizin oluşturur.
s = rmdir(name)	Boş bir dizini kaldırır.
s = link(name1, name2)	name1'i işaret eden name2 adında girdi oluşturur.
s = unlink(name)	Bir dizin girdisini kaldırır.
s = mount(special, name, flag)	Bir dosya sistemini bağlar.
s = umount(special)	Bir dosya sisteminin bağlantısını kaldırır.

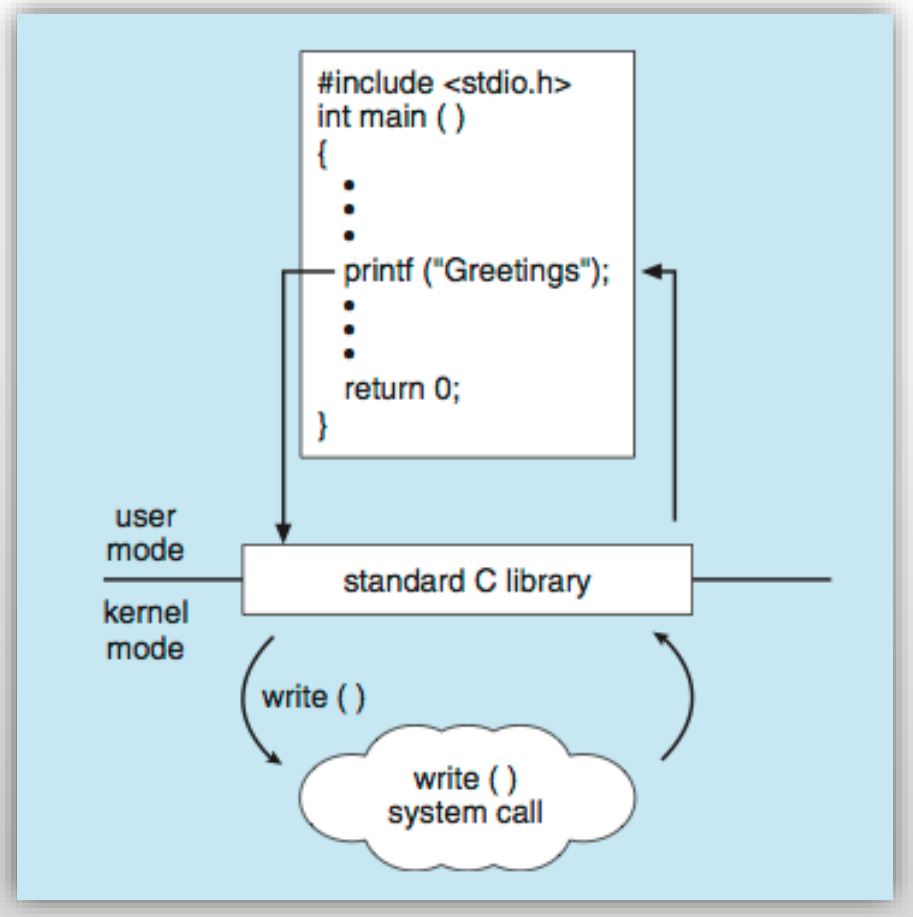
Sistem Çağrıları

seconds: geçen süre

Çağrı	Tanım
s = chdir(dirname)	Çalışma dizinini (*working directory*) değiştirir.
s = chmod(name, mode)	Bir dosyanın koruma (*mode*) bitlerini değiştirir
s = kill(pid, signal)	Bir sürece sinyal gönderir.
seconds = time(&seconds)	1 Ocak 1970'ten bu yana geçen süreyi döndürür.

Standart C Kütüphanesi

- .



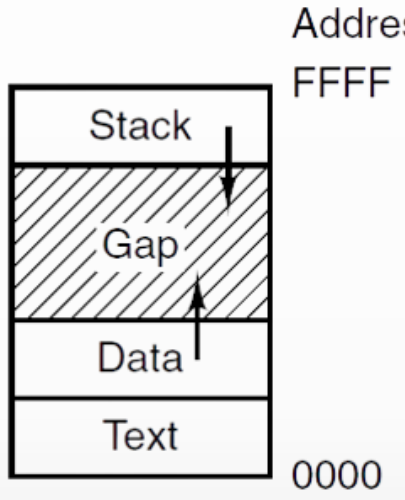
Süreç Yönetimi

```
#define TRUE 1

while (TRUE) {                                  /* repeat forever */
        type_prompt( );                         /* display prompt on the screen */
        read_command(command, parameters);      /* read input from terminal */

        if (fork( ) != 0) {                     /* fork off child process */
             /* Parent code. */
             waitpid(–1, &status, 0);           /* wait for child to exit */
        } else {
             /* Child code. */
             execve(command, parameters, 0);    /* execute command */
        }
}
```

fork Çağrısı

- Paralel yürütme ve süreç oluşturma için kullanılır.
- Ata ve çocuk süreçler,
  - aynı program kodunu paylaşırlar,
  - farklı bellek adres uzaylarına sahiptirler.
- *fork* çağrısı ile bir süreç, bir çocuk süreci oluşturur.
- Ata süreç, oluşturulan çocuk sürecin *PID*'sini (*process id*) alır.

Süreçlerin Bellek Yönetimi

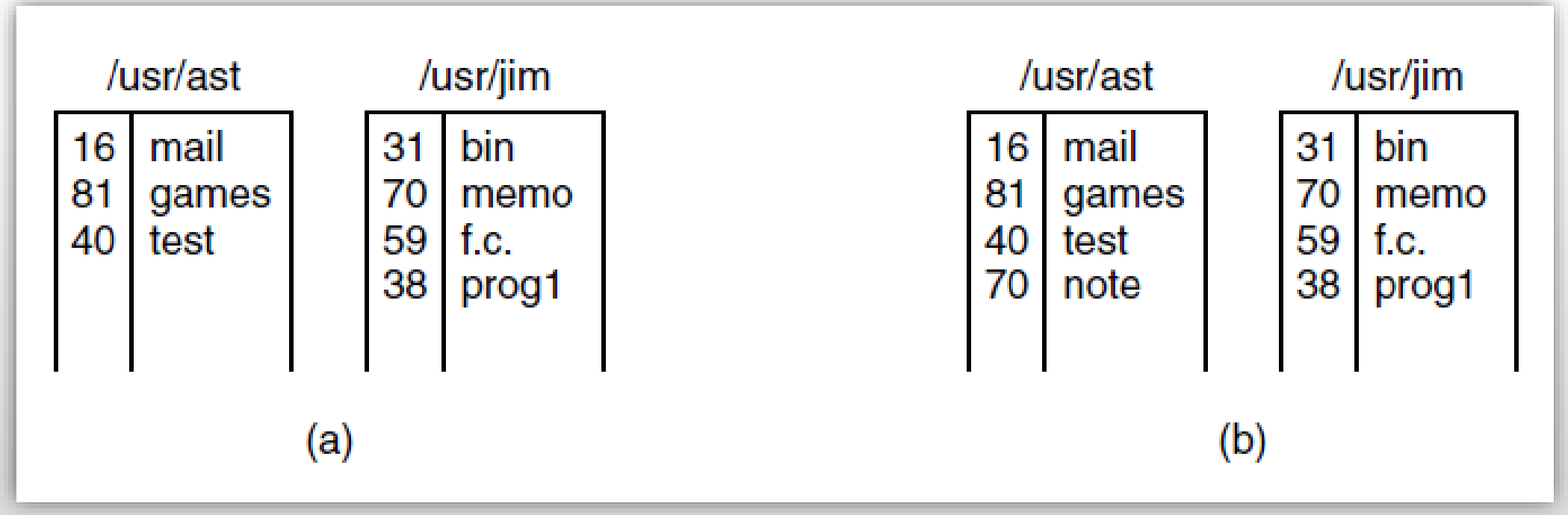


- Süreçler 3 kesime sahiptir. metin, veri, yığın.
- **Yığın**, süreç için gerekli olan parametreler, yerel değişkenler ve fonksiyon dönüş adresleri gibi geçici verileri saklar.
- **Veri**, genel ve statik değişkenlerin yanı sıra dinamik olarak ayrılmış belleği depolar. Tüm iş parçacıkları arasında paylaşılır.
- **Metin**, sürecin talimatlarını uygulayan makine kodunu tutar. Salt okunurdur ve sürecin tüm iş parçacıkları arasında paylaşılır. Talimatları yürütmek için CPU tarafından kullanılır.

Dizin Yönetimi

(a) *usr/jim/memo*, */usr/ast* dizinine bağlanmadan önce.

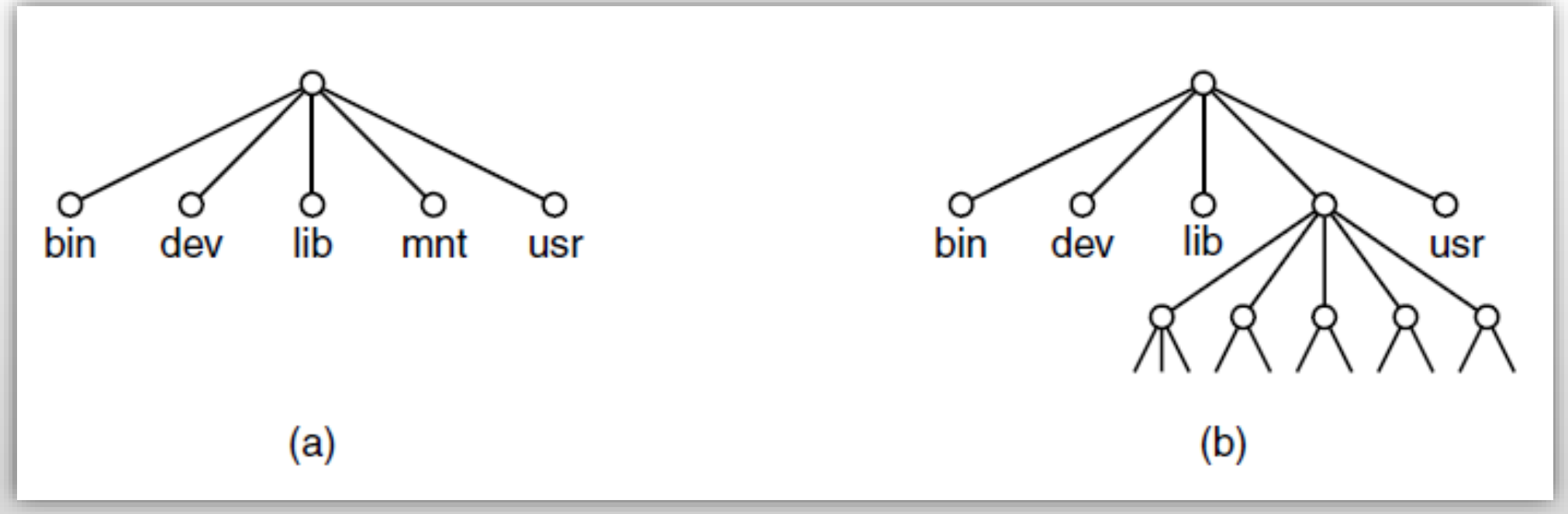
(b) */usr/ast/note* adı ile bağlandıktan sonra.



Dizin Yönetimi

(a) Bağlamadan (*mount*) önce

(b) /mnt dizinine bağladıktan sonra



Kilitlenme (Deadlock)

(a) Potansiyel kilitlenme

(b) Gerçekleşmiş kilitlenme

The Windows Win32 API

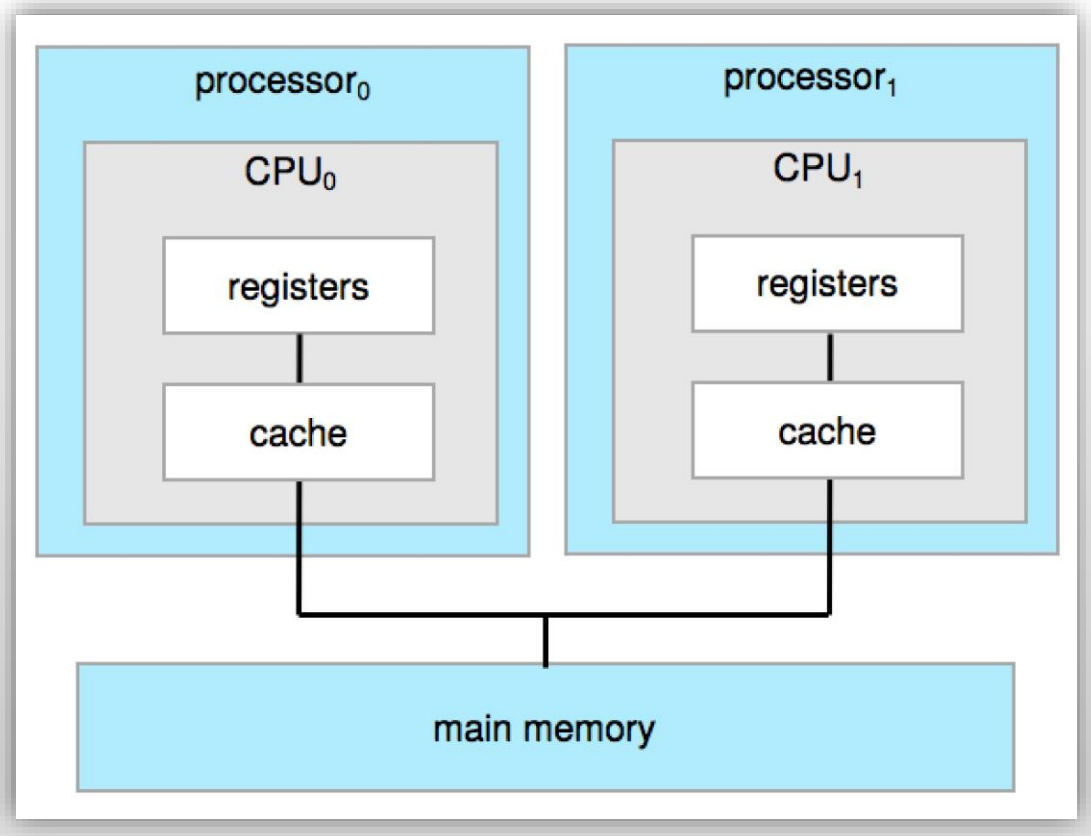
UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory

The Windows Win32 API

lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

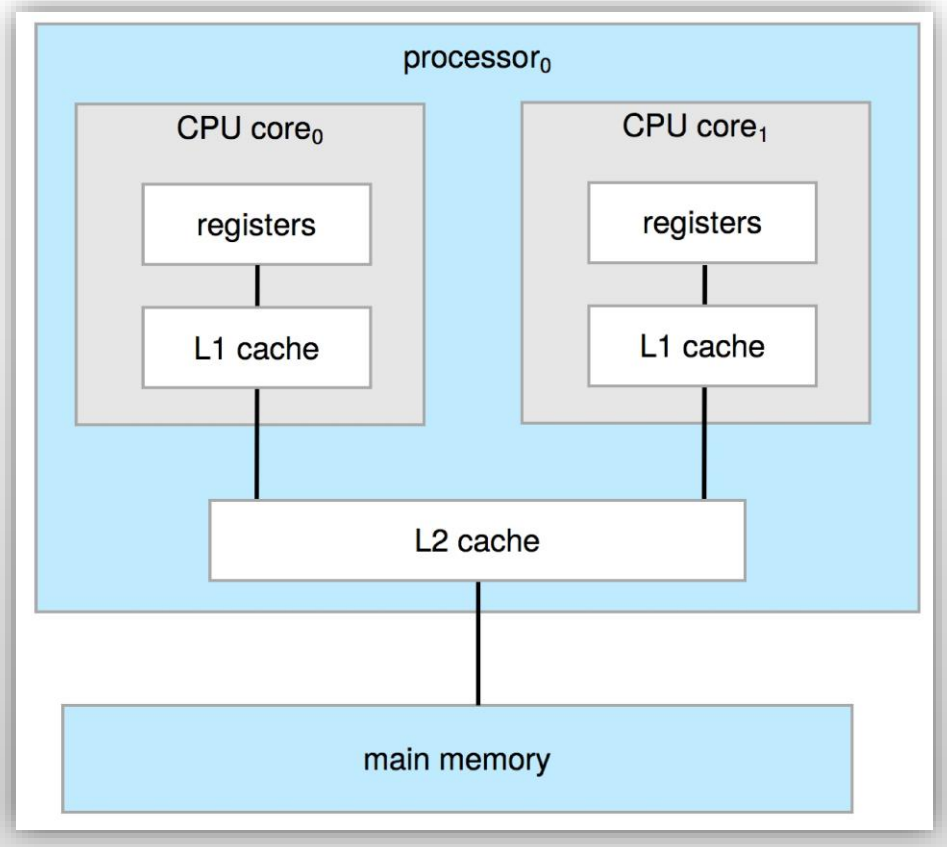
Simetrik Çoklu İşlemci Mimarisi

- .

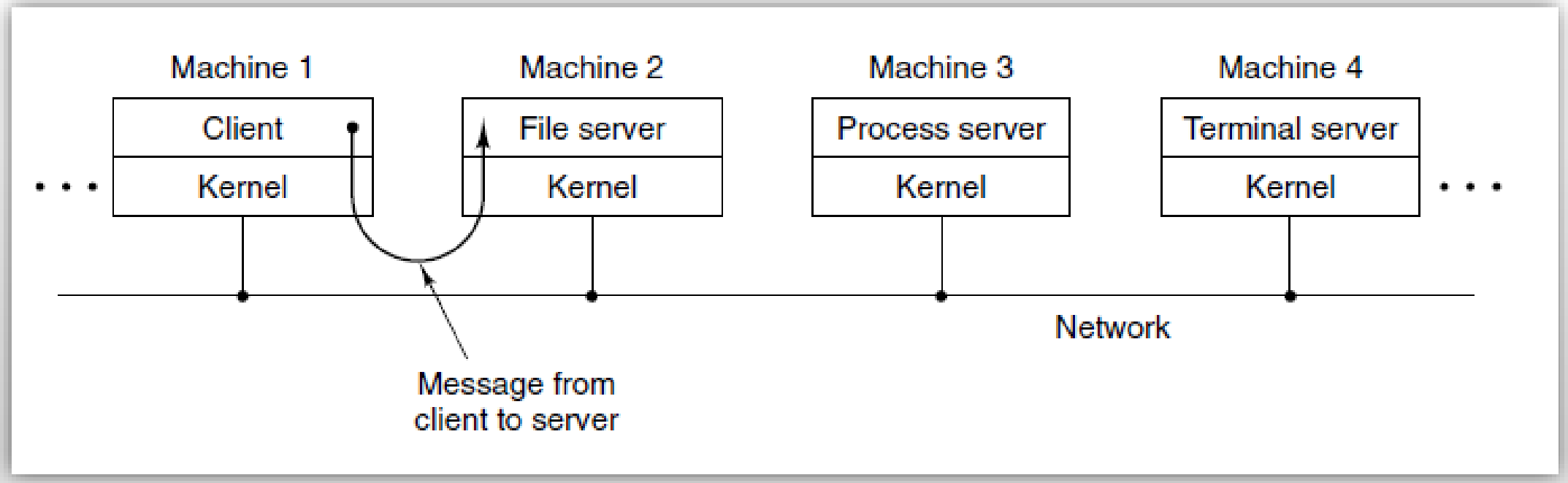


Çift Çekirdekli İşlemci Mimarisi

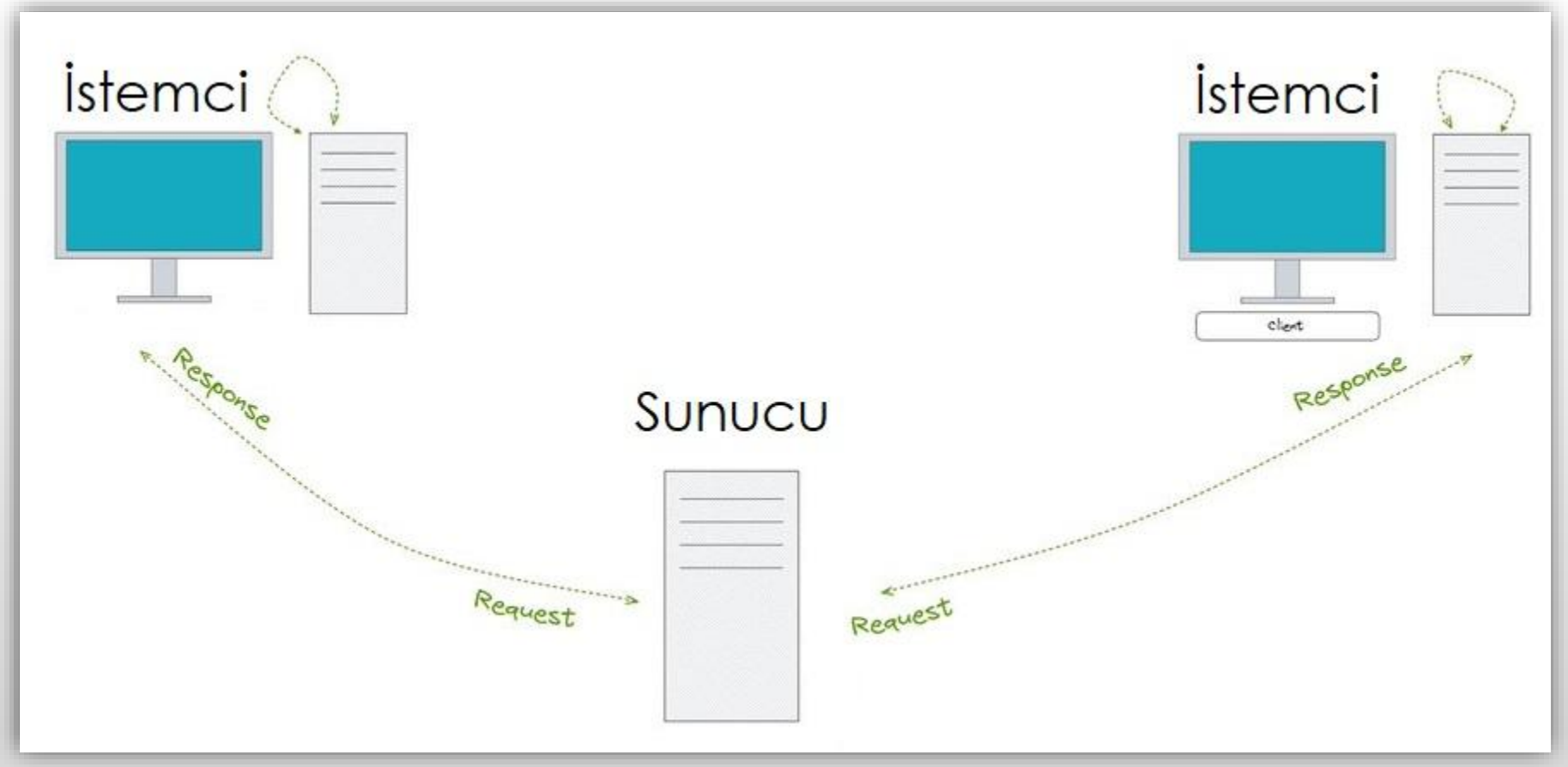
- .



İstemci Sunucu Modeli



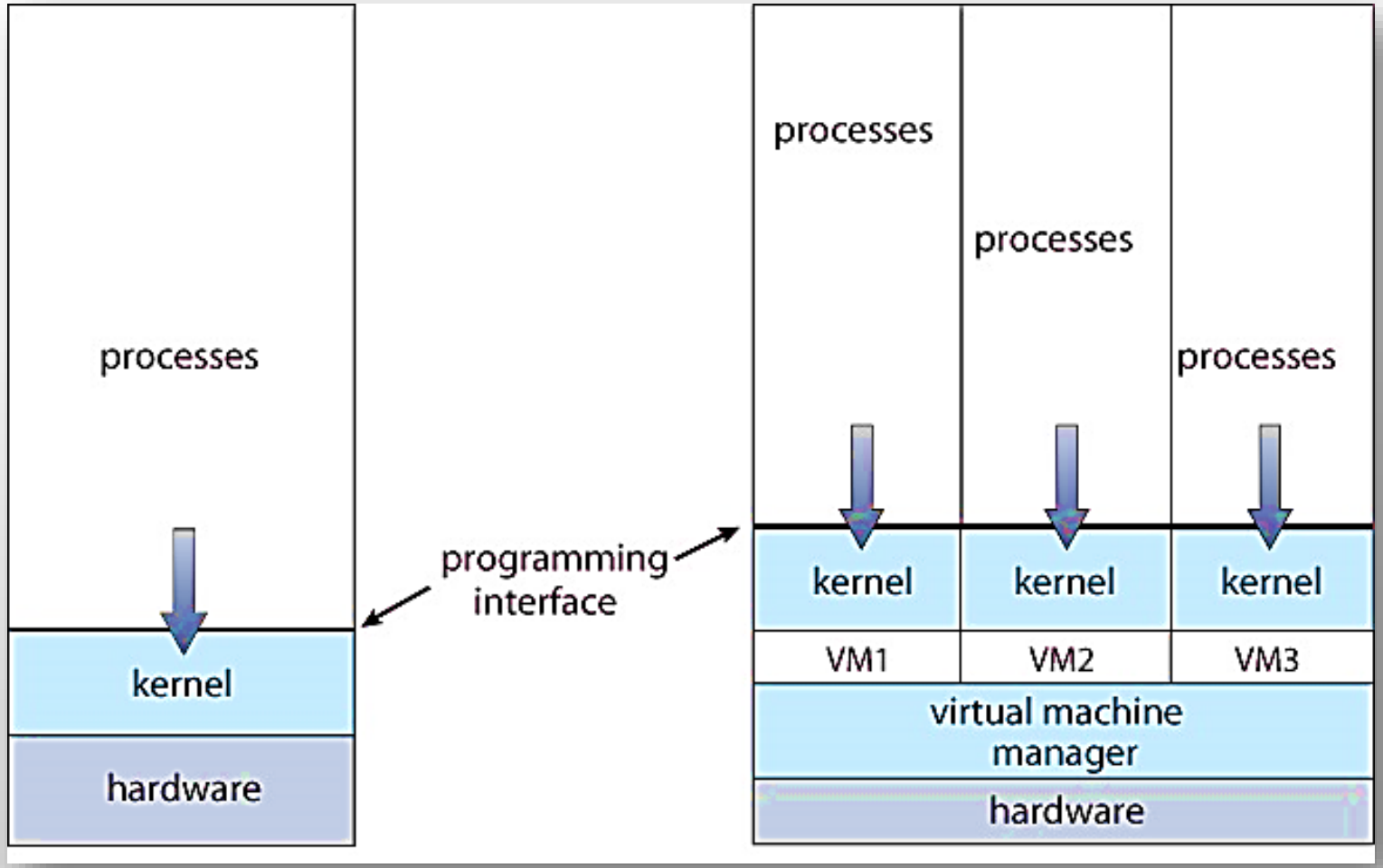
İstemci Sunucu Modeli



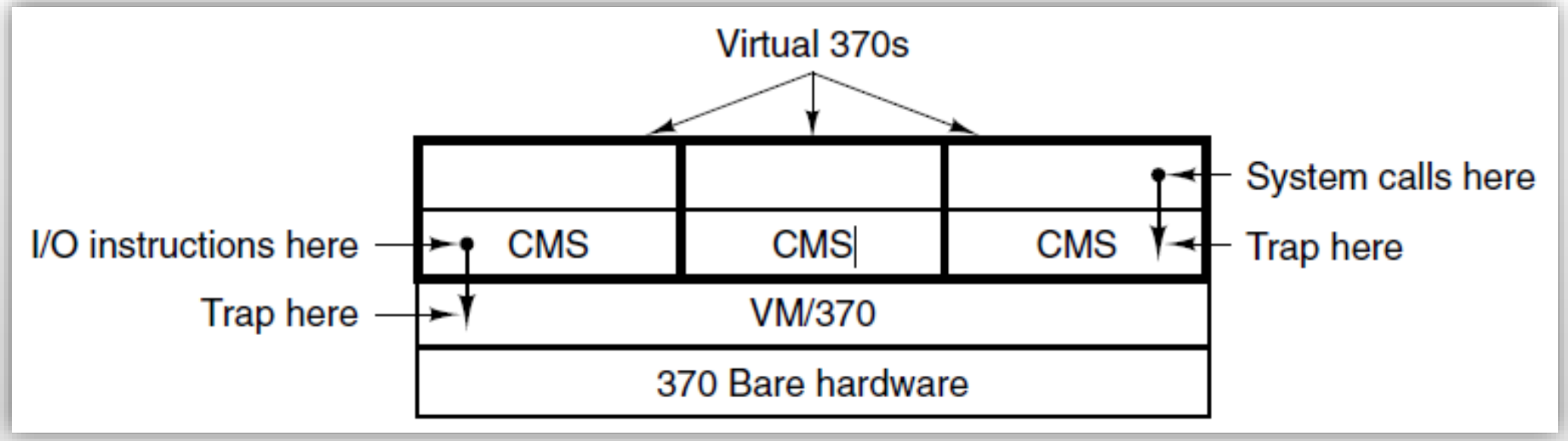
Sanal Makineler

- Aynı fiziksel donanım üzerinde eş zamanlı olarak çalışan birden çok farklı işletim sistemine, bağımsız donanım gibi görünen bir arayüz sağlar.
- Her işletim sistemi, kendi CPU, RAM, I/O aygıtları, sabit diskler gibi kaynaklara erişimi ve kontrolü olduğuna inanır.
- Farklı işletim sistemlerine sahip sanal makineler, aynı donanım üzerinde çalışabilir.

Sanal Makineler

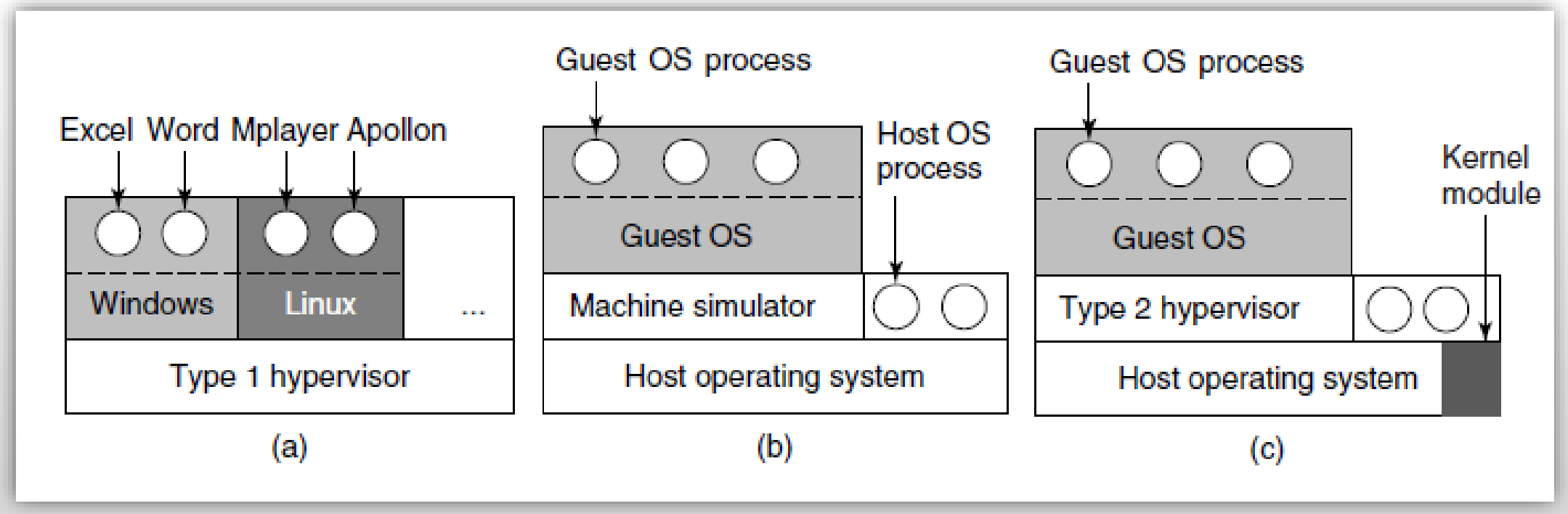


Sanal Makine Yapısı



Sanal Makine Yapısı

(a) Tip 1 hipervizör (sanal makine yöneticisi) (b) Yalın tip 2 hipervizör. (c) Pratik tip 2 hipervizör.



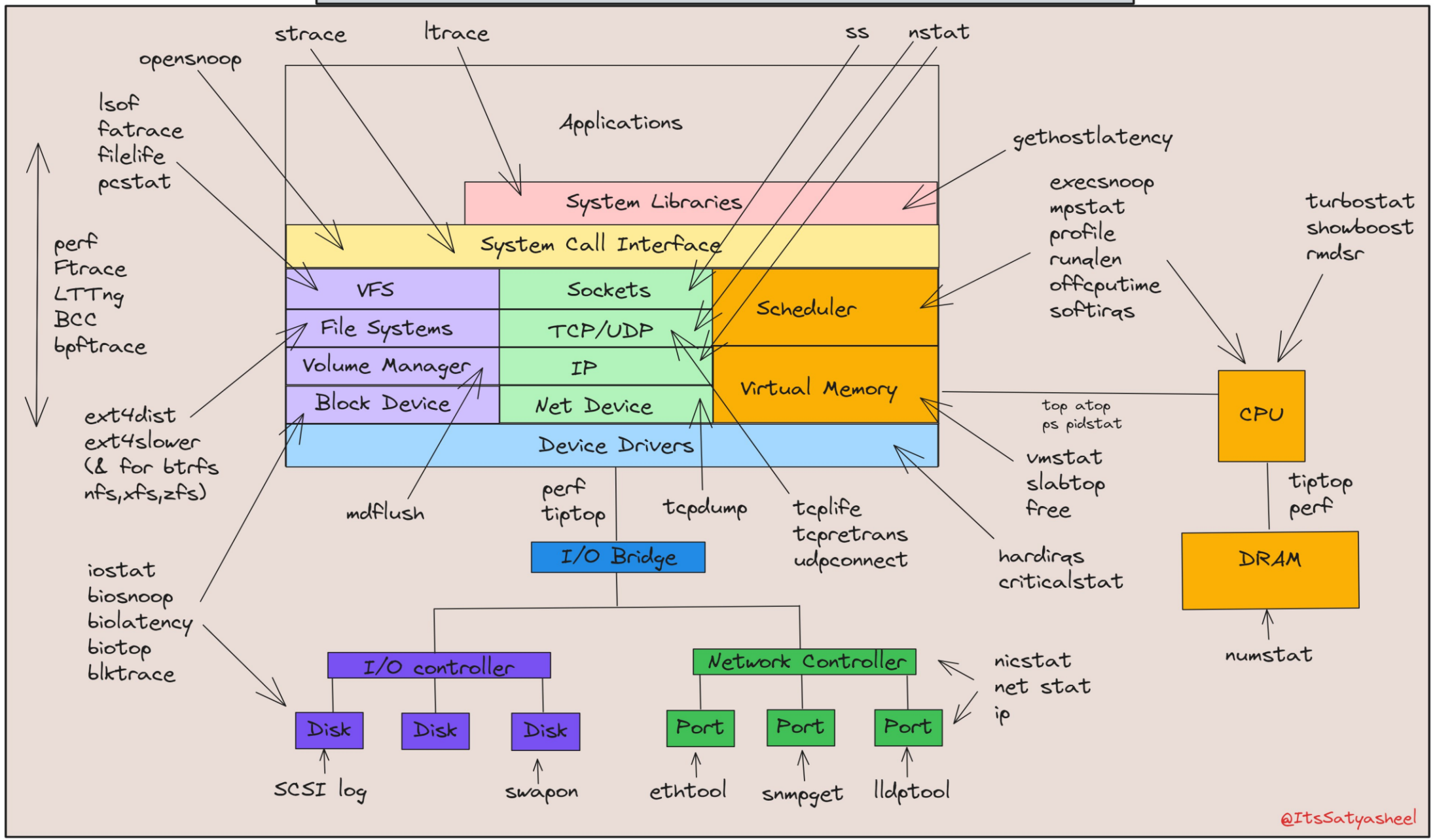
Tip 1 Hipervizör (Type 1 Hypervisor)

- İşletim sisteminden bağımsız doğrudan donanım üzerinde çalışır.
- Sanal makineleri yöneten bir sanallaştırma yazılımıdır.
- Fiziksel sunucuları sanal makinelerle bölme ve yönetme ihtiyacı için.
- Kaynakların daha etkin bir şekilde kullanılması ve izlenmesi amacıyla.
- Genellikle "*bare-metal*" hipervizör olarak adlandırılır.
- Veri merkezlerinde fiziksel sunucuları sanal makinelere bölme ve yönetme.
- **VMware ESXi**: Endüstri lideri, geniş özelliklere sahip
- **Microsoft Hyper-V Server**: Microsoft'un ücretsiz çözümü
- **Xen**: Açık kaynaklı ve ölçeklenebilir

Tip 2 Hipervizör (Type 2 Hypervisor)

- Bir işletim sistemi üzerinde çalışır.
- Sanal makineleri yöneten bir sanallaştırma yazılımıdır.
- Kişisel bilgisayarlar veya geliştirme ortamları için sanallaştırma ihtiyacı.
- Genellikle "*hosted*" hipervizör olarak adlandırılır.
- Tip 1 hipervizörlere göre daha düşük performans sunar.
- İşletim sistemi bağımsızlığı sağlar.
- **Oracle VirtualBox**: Ücretsiz ve açık kaynaklı, geniş platform desteği.
- **VMware Workstation**: Profesyonel kullanım, gelişmiş özelliklere sahip.
- **Parallels Desktop**: macOS üzerinde kullanılır.

Linux Performance Observability Tools



SON