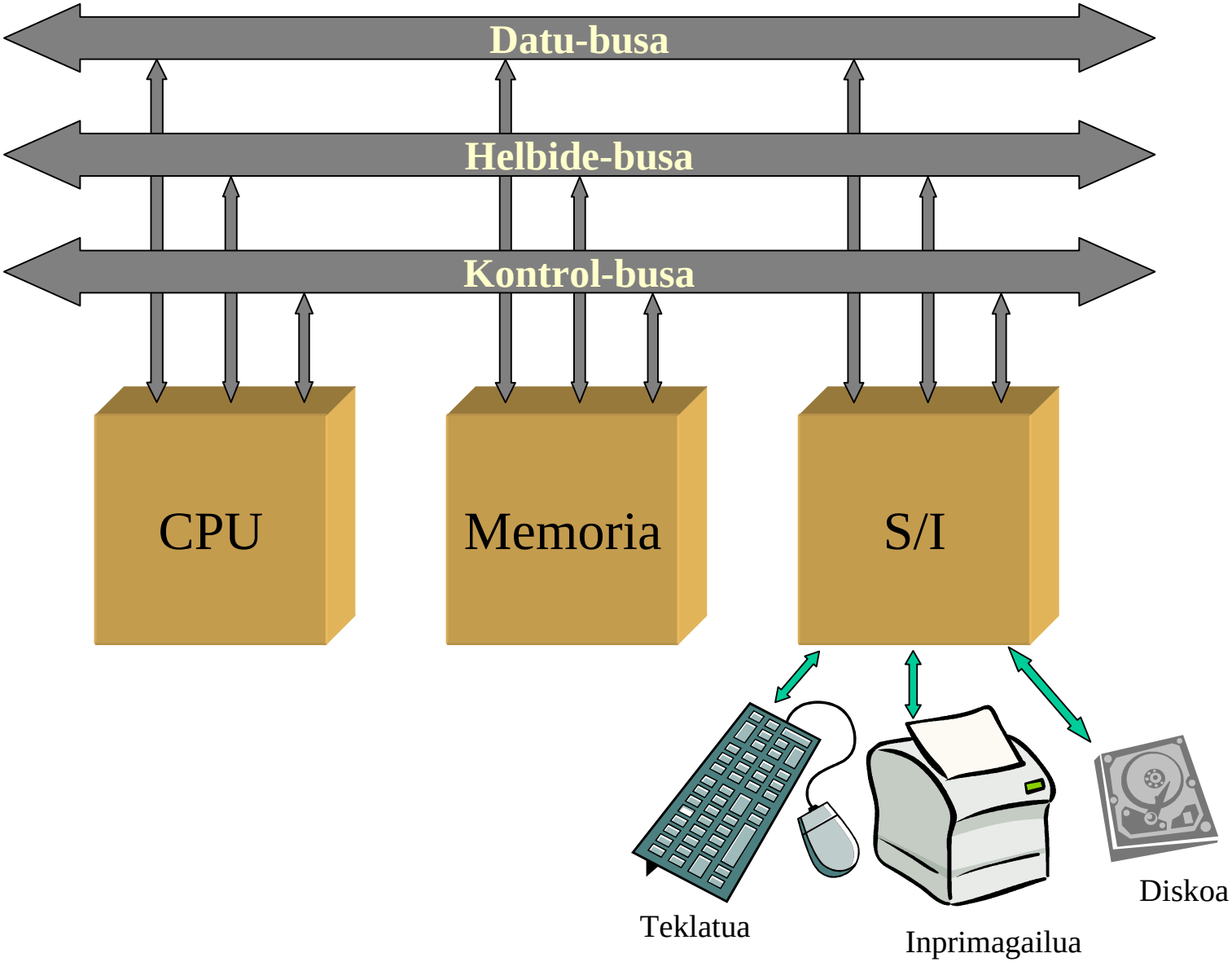
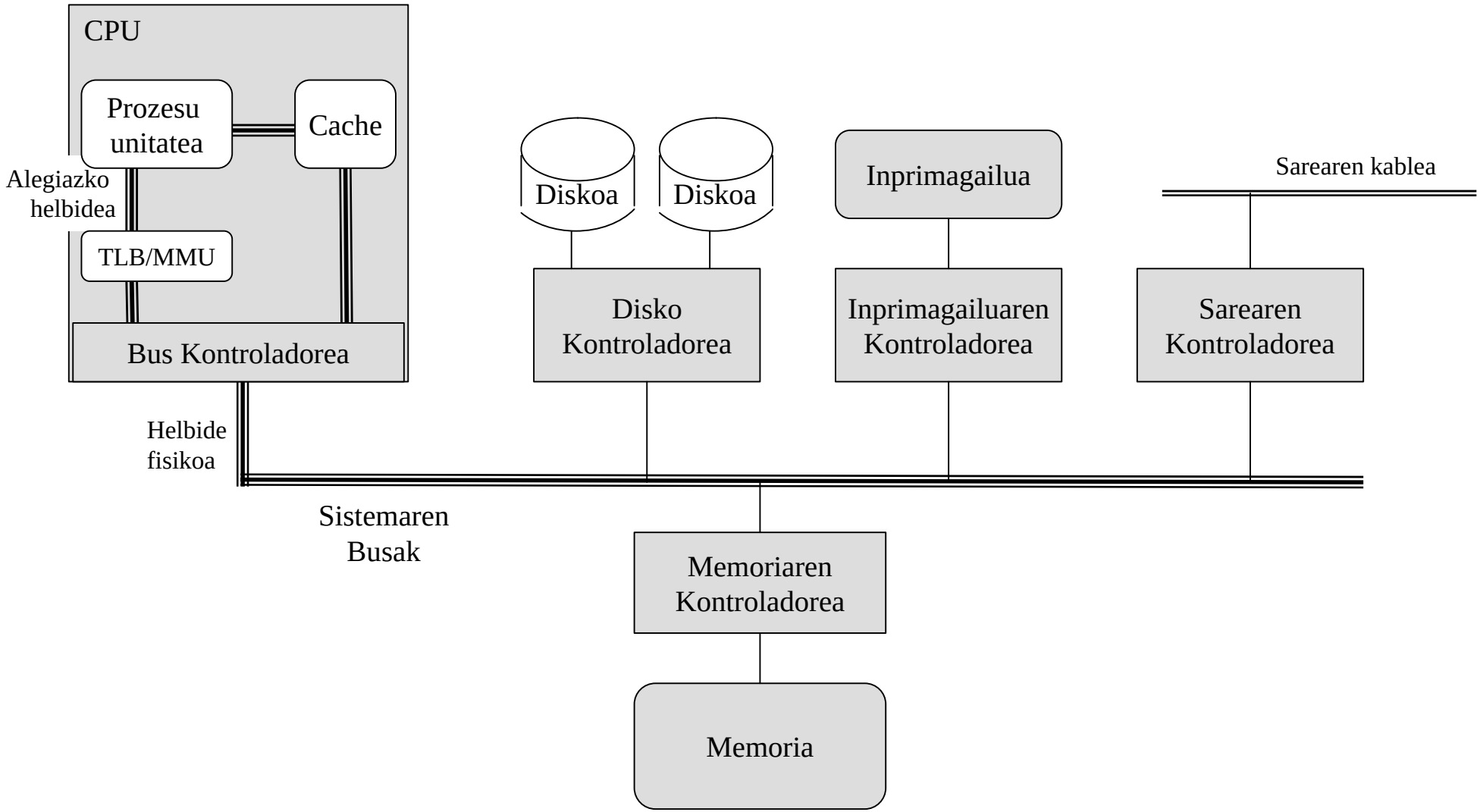
2. Gaia: Sistema-deiak

1. Oinarrizko arkitektura
2. S/I errutinen mekanismoa
3. Errutina egoiliarrak
4. Sistema Eragilea atzitzeko modua: sistema-deiak

Von Neumann motako makina



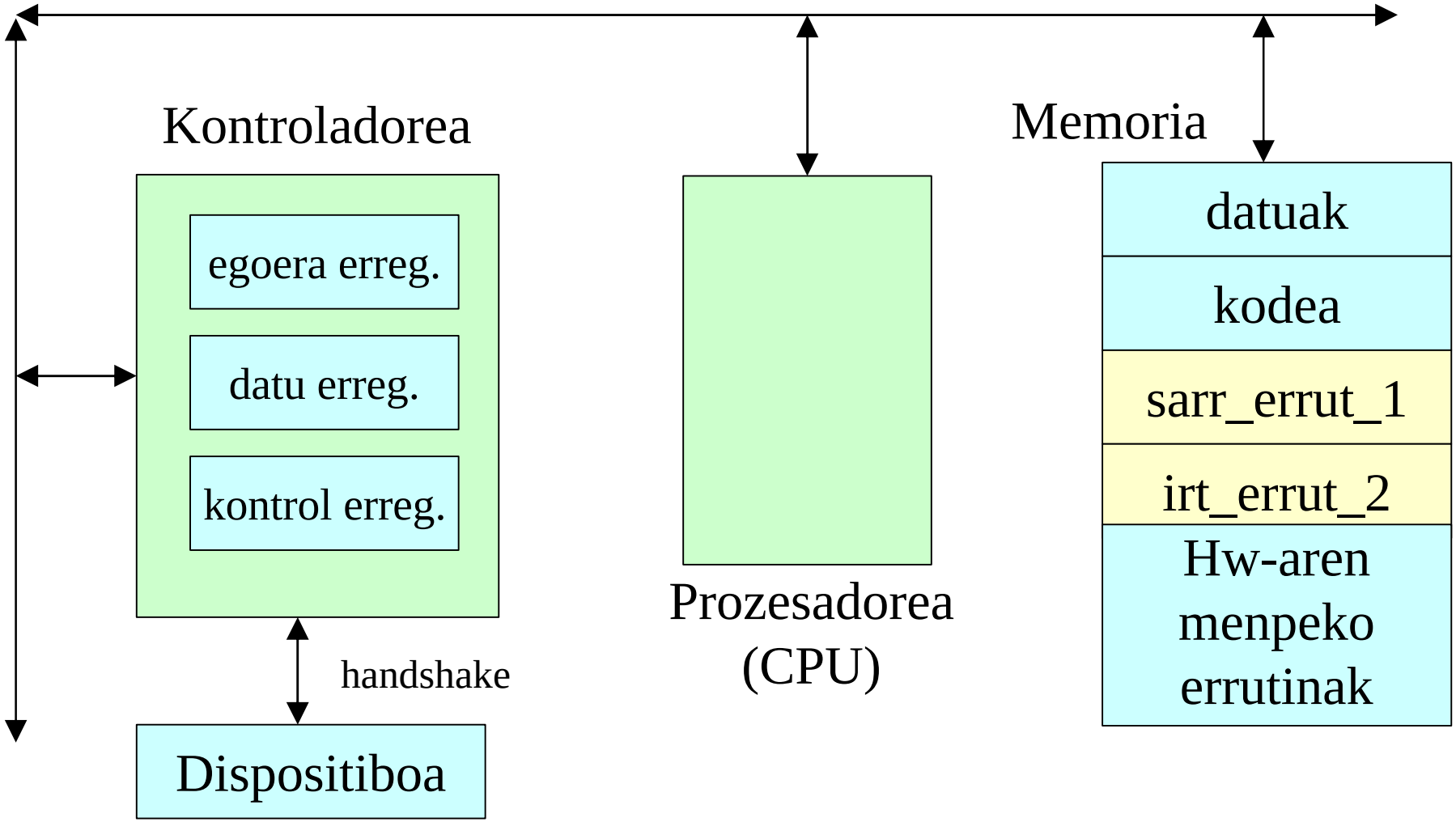
Oinarrizko arkitektura: kontroladoreak



Oinarrizko arkitektura

- Osagai bakoitza (prozesadorea, memoria, busa, periferikoak) eraginkorragoa bada, konputagailu osoa ere eraginkorragoa gerta daiteke, baina ez derrigorrez
  - Baliabide guztien **kudeaketa orekatua** izango da eraginkortasunaren arrakasta edo porrotaren erantzulea
- **Sistema Eragile** izenez ezagutzen den programa-multzoaren funtzioetariko bat konputagailuaren baliabide guztiak **era eraginkorrean** kudeatzea da
  - Kudeaketa hau egiteko moduak eragingo du konputagailuaren errendimenduan

Inkesta bidezko Sarrera/Irteera



Inkesta bidezko Sarrera/Irteera (2)

- Dispositiboa erabiltzeko prozedura:
  1. Irakur/idatz ote daitekeen egiaztatu egoera-erregistroan
  2. Prest dagoenean, datua irakurri/idatzi
  3. Kontroladoreari adierazi datua irakurria/idatzia izan dela
- Kontroladorearen egoera egiaztatzeko, egoera-erregistroa irakurriko da inkesta edo itxoite aktiboa *(busy waiting)* delakoaren bidez
- Erabiltzailearen programaren ardura da egoera-erregistroa aztertu eta beharrezkoa denean erroreen tratamendu-errutina deitzea

Inkesta bidezko Sarrera/Irteera (adibidea)

Adibidea: *DISP1* dispositibotik 80 karaktere irakurri ondoren beste batean, *DISP2n*, idatzi

Errutina motak:

Hardwarearen menpeko errutinak:
kontroladoreen erregistroak atzitzeko (kodetutzat jotzen dira)

Sarrera/Irteerako errutinak:
sarrerak eta irteerak burutzen dituzte

erroreak errutina:
errorerik badago, programaren exekuzioa amaitzen du, mezu bat atereaz (kodetutzat jotzen da)

⇨ S/I landua/gordina: kontrol-karaktereak

Lerro-saltoa, ezabatzeko tekla, *EOF*…

Inkesta bidezko Sarrera/Irteera (adibidea)

1 dispositiborako sarrera errutina

```
sarr_errut_1 (char *bektorea, int kopurua)
{
   int j, egoera;

   for (j = 0; j < kopurua; j++)
   {
      do {
         egoera = irakurri_egoera_erreg(DISP1);
      } while (egoera == LANEAN);
      erroreak(DISP1, egoera);
      bektorea[j] = irakurri_datu_erreg(DISP1);
      idatzi_kontrol_erreg(DISP1, IRAKURRITA);
   }
}
```

2 dispositiborako irteera errutina

```
irt_errut_2 (char *bektorea, int kopurua)
{
   int j, egoera;

   for (j = 0; j < kopurua; j++)
   {
      do {
         egoera = irakurri_egoera_erreg(DISP2);
      } while (egoera == LANEAN);
      erroreak(DISP2, egoera);
      idatzi_datu_erreg(DISP2, bektorea[j]);
      idatzi_kontrol_erreg(DISP2, IDATZITA);
   }
}
```

erabiltzailearen programa

(sinkronoa)

```
main ()
{
   char buff[80];

   while (TRUE) {
      sarr_errut_1(buff, 80);
      irt_errut_2(buff, 80);
   }
}
```

Inkesta bidezko Sarrera/Irteera

- Arazoa:
  - Sarrera/Irteera gauzatzen ari den bitartean, CPU-a denbora gehiena beste ezer egin gabe itxaroten ari da
  - Itxarote-denborak oso luzeak izan daitezke
    - Sarrera/Irteerako dispositiboak motelak dira (CPU-arekin alderatuz)

- Irtenbidea:
  - Etenen bidezko Sarrera/Irteera

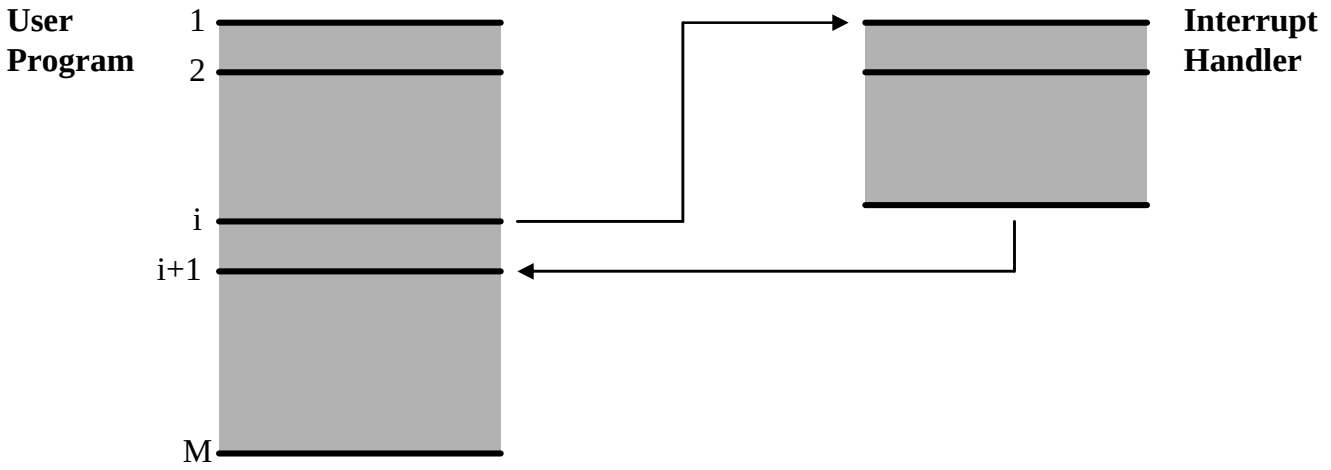
Memoria hierarkia

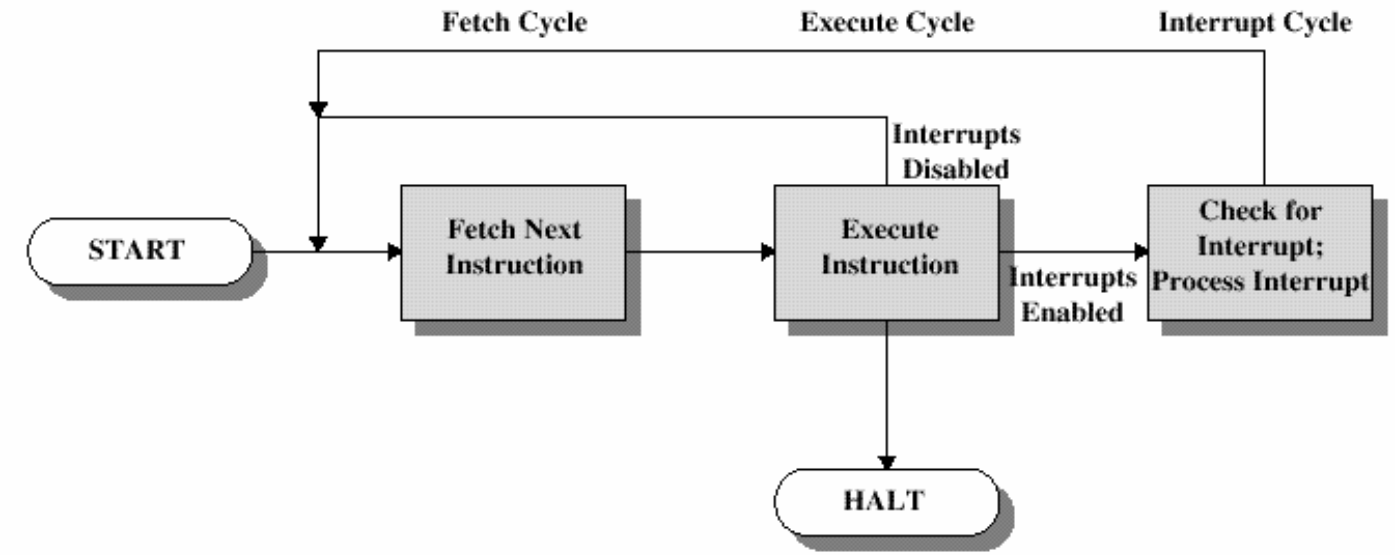
	Atzipen denbora	Tamaina
Erregistroa	0,5 ns	512 Byte
Cache On-chip (L1)	2 ns	32 KB - 64 KB
Cache Off-chip (L2)	5 - 10 ns	512 KB - 2 MB
Memoria nagusia	20 - 50 ns	2 GB
Disko magnetikoa	10 ms	100 GB
CD_ROM	300 ms	700 MB
Zinta magnetikoa	1 s - 1 min	GB / TB

Etenak

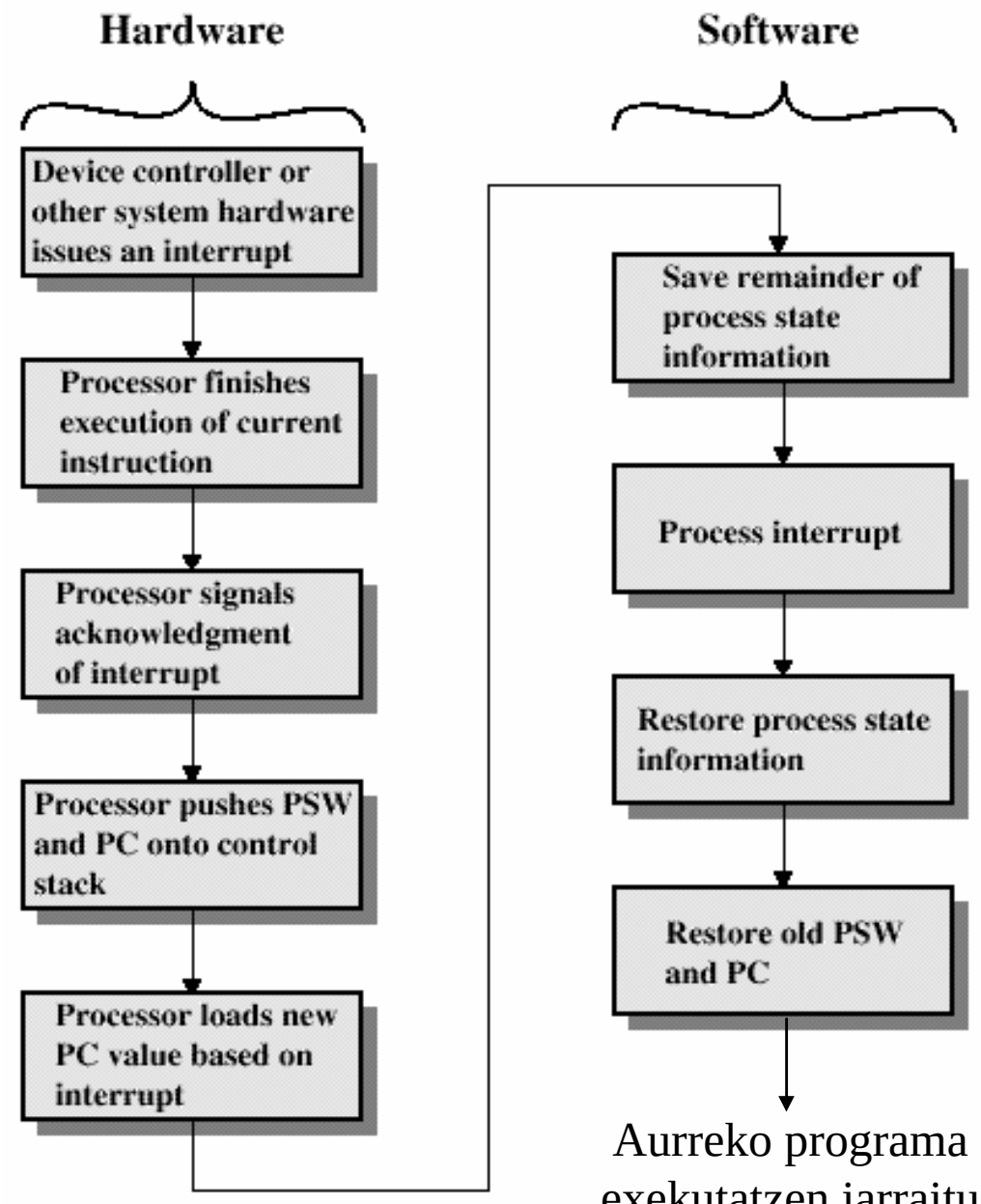
- Etena – konputagailu sistema batean sortutako gertaera, zeinak exekuzio fluxuan eragiten duen
- Eten motak:
  - Hardware etenak
    - Erlojuaren etena (periodikoa)
    - S/I dispositibo batek sortua (asinkronoa)
    - Software errorea (overflow aritmetikoa, agindu ezezaguna, memoria erreferentzi okerra…)
    - Hardware errorea (bus errorea)
  - Software etena edo *Trap* deiturikoa
    - Agindu berezi baten bidez sortua
    - Sinkronoa aginduen zerrendari dagokionez

Instruction Control Flow and Instruction Cycle with Interrupts

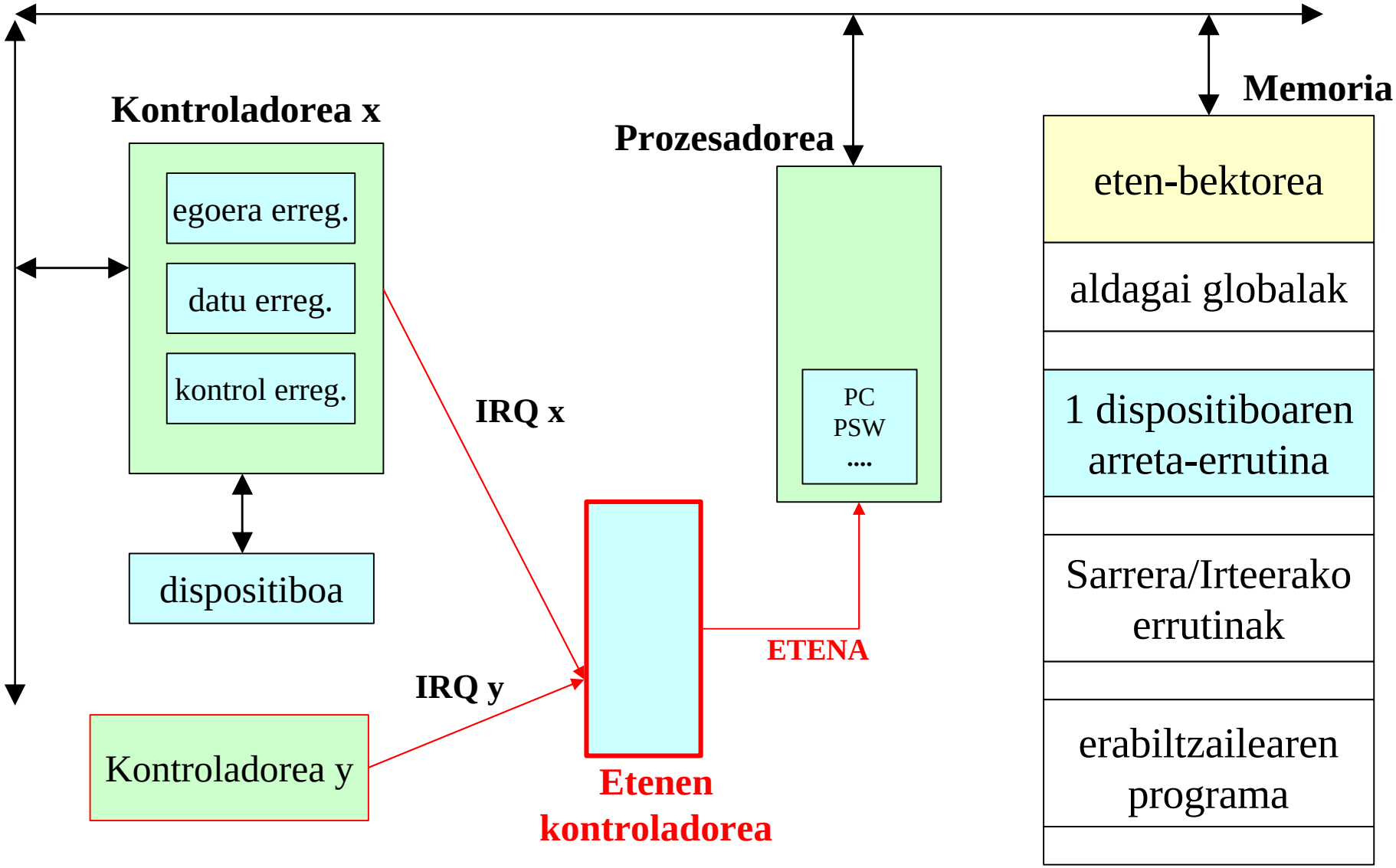




Etenen kudeaketa



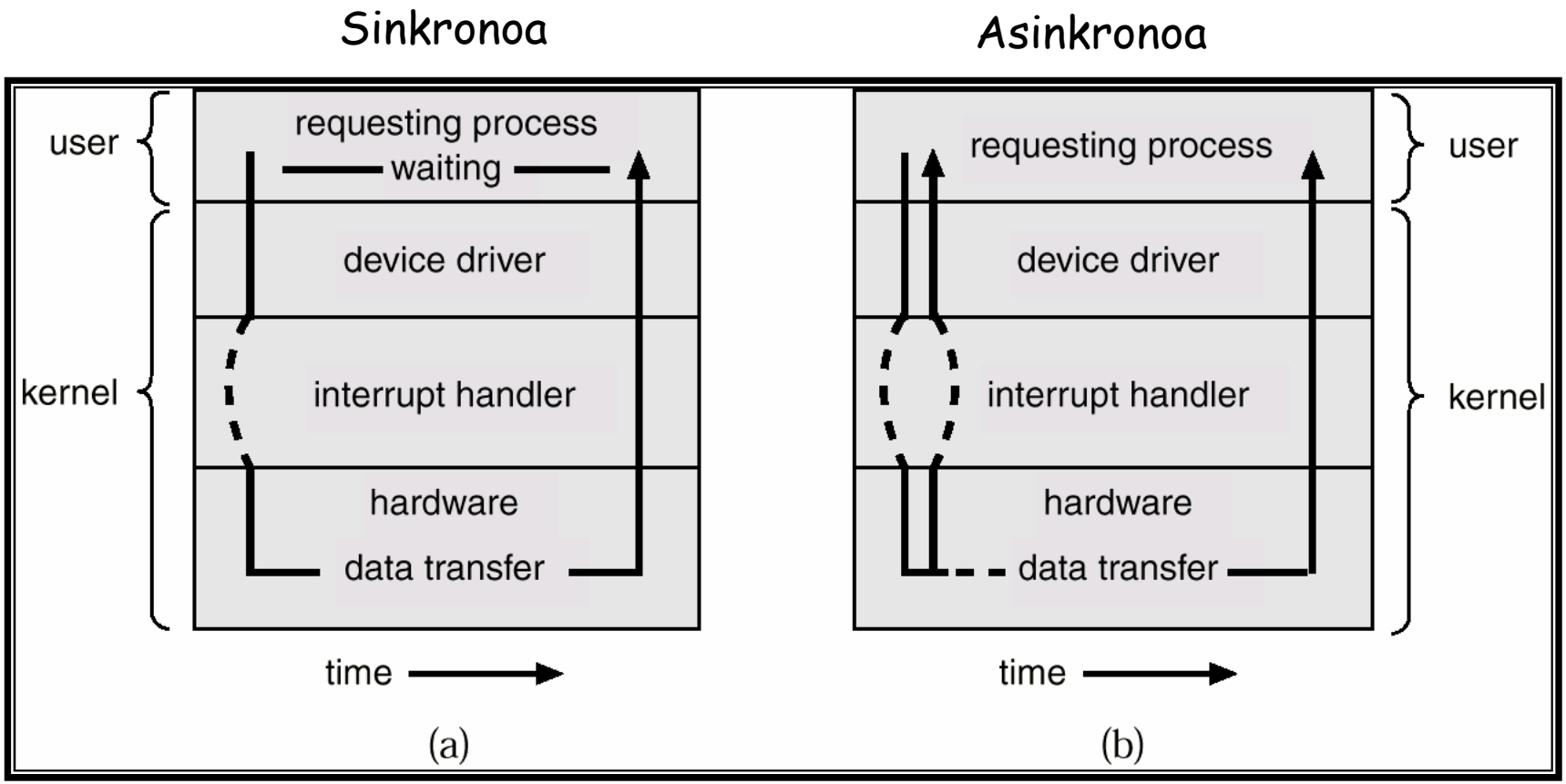
Etenen bidezko Sarrera/Irteera



Etenen bidezko Sarrera/Irteera (2)

- Paralelotasuna eskaintzen du: eragiketak sinkronoak ala asinkronoak izan daitezke
- Dispositiboak karaktere/bloke bat irakurri/idaztean, eten bat sortzen du
- Sarrera eta irteera errutinez gain, programatzaileak arreta-errutina programatu behar du, etena gertatzean exekutatuko dena
- Eten-bektorea aldatu behar da ere, etena heltzerakoan programatzailearen arreta errutina exekuta dadin

S/I eragiketa motak



Etenen bidezko Sarrera/Irteera (adibidea)

Erabiltz. programa (sinkronoa)

```
main ()
{
   char buff[80];

   aldatu_bektore_sarrera(DISP1, arreta_errutina_1);
   aldatu_bektore_sarrera(DISP2, arreta_errutina_2);
   while (TRUE)
   {
      sarr_errut_1(buff, 80, SINKRO);
      irt_errut_2(buff, 80, SINKRO);
   }
}
```

Sinkronizazio-errutina

```
void sinkronizatu (int *bukaera)
{
   while ((*bukaera) == FALSE) NOP;
}
```

1 dispositiboaren sarrera errutina

```
sarr_errut_1 (char *bektorea, int kop, int asink_sink)
{
   amaiera1 = FALSE; buff1 = bektorea; indize1 = 0;
   konta1 = kop;
   if (asink_sink == SINKRO)
      sinkronizatu(&amaiera1);
}
```

Aldagai globalak

```
int amaiera1 = TRUE, amaiera2 = TRUE;
char *buff1, *buff2;
int konta1 = 0, konta2 = 0;
int indize1, indize2;
```

2 dispositiboaren irteera errutina

```
irt_errut_2 (char *bektorea, int kop, int asink_sink)
{
   int egoera;
   amaiera2 = FALSE;
   /* lehenengo idazketa inkesta bidez */
   do {
      egoera=irakurri_egoera_erreg(DISP2);
   } while (egoera == LANEAN);
   erroreak(DISP2, egoera);
   buff2 = bektorea;
   indize2 = 1;
   konta2 = kop;
   idatzi_datu_erreg(DISP2, buff2[0]);
   idatzi_kontrol_erreg(DISP2, IDATZITA);
   if (asink_sink == SINKRO)
      sinkronizatu(&amaiera2);
}
```

Etenen bidezko Sarrera/Irteera (adibidea)

1 dispositiboaren arreta-errutina

```
arreta_errutina_1 ()
{
   int egoera;

   if (konta1 != 0)
   {
      egoera = irakurri_egoera_erreg(DISP1);
      erroreak(DISP1, egoera);
      buff1[indize1++] =
                  irakurri_datu_erreg(DISP1);
      konta1--;
      if (konta1 == 0) amaiera1 = TRUE;
   } /* else irakurketa aurreratua */
   idatzi_kontrol_erreg(DISP1, IRAKURRITA);
   amaiera_eten_errut();
}
```

2 dispositiboaren arreta-errutina

```
arreta_errutina_2 ()
{
   int egoera;

   egoera = irakurri_egoera_erreg(DISP2);
   erroreak(DISP2, egoera);
   konta2--;
   if (konta2 > 0)
   {
      idatzi_datu_erreg(DISP2,
                              buff2[indize2++]);
      idatzi_kontrol_erreg(DISP2, IDATZITA);
   }
   else amaiera2 = TRUE;
   amaiera_eten_errut();   /* eoi(); iret */
}
```

Etenen bidezko Sarrera/Irteera (adibidea)

Erab. programa (sinkronoa)

```
main ()
{
   char buff[80];

   aldatu_bektore_sarrera(DISP1,
                          arreta_errutina_1);
   aldatu_bektore_sarrera(DISP2,
                          arreta_errutina_2);
   while (TRUE)
   {
      sarr_errut_1(buff, 80, SINKRO);
      irt_errut_2(buff, 80, SINKRO);
   }
}
```

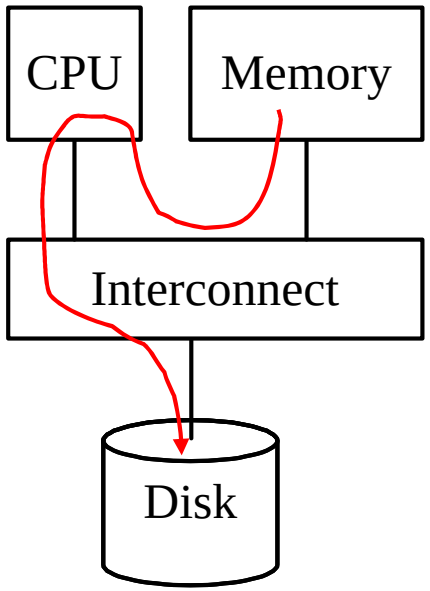
Erab. programa (asinkronoa)

```
main ()
{
   char bek1[80], bek2[80];

   aldatu_bektore_sarrera(DISP1,
                          arreta_errutina_1);
   aldatu_bektore_sarrera(DISP2,
                          arreta_errutina_2);
   while (TRUE)
   {
      sarr_errut_1(bek1, 80, SINKRO);
      sinkronizatu(&amaiera2);
      irt_errut_2(bek1, 80, ASINK);
      sarr_errut_1(bek2, 80, SINKRO);
      sinkronizatu(&amaiera2);
      irt_errut_2(bek2, 80, ASINK);
   }
}
```

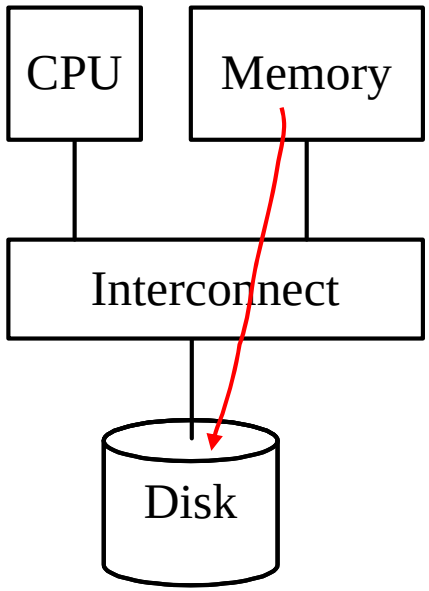
Direct Memory Access (DMA)

- Used to avoid programmed I/O for large data movement
- Requires DMA controller
- Bypasses CPU to transfer data directly between I/O device and memory

Programmed I/O vs DMA

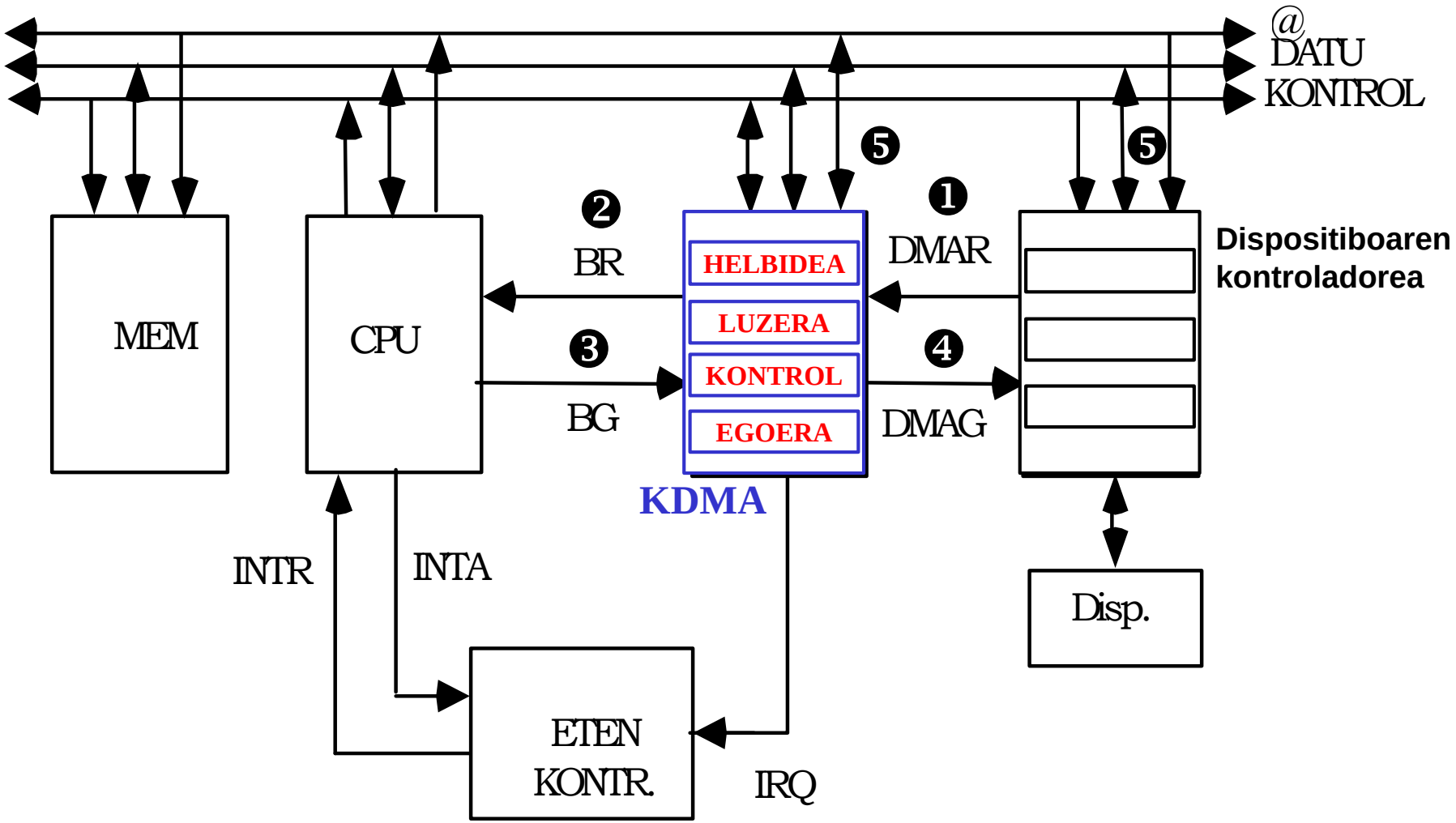


Programmed I/O



DMA

Memoriaren atzipen zuzena (DMA)
Arkitektura



DMA bidezko Sarrera/Irteera

- Paralelotasun handigoa eskaintzen du: dispositiboa eta hardwarea arduratzen dira transferentzi osoaz modu paraleloan

- Bloke bat transferitu behar denean, dispositiboa eta memoria artean protokolo bat antolatzen du DMA-k

- DMA-k eten bat sortzen du transferentzi osoa amaitu denean. Programatzaileak DMA-ren arreta-errutina programatu behar du

DMA bidezko Sarrera/Irteera (adibidea)

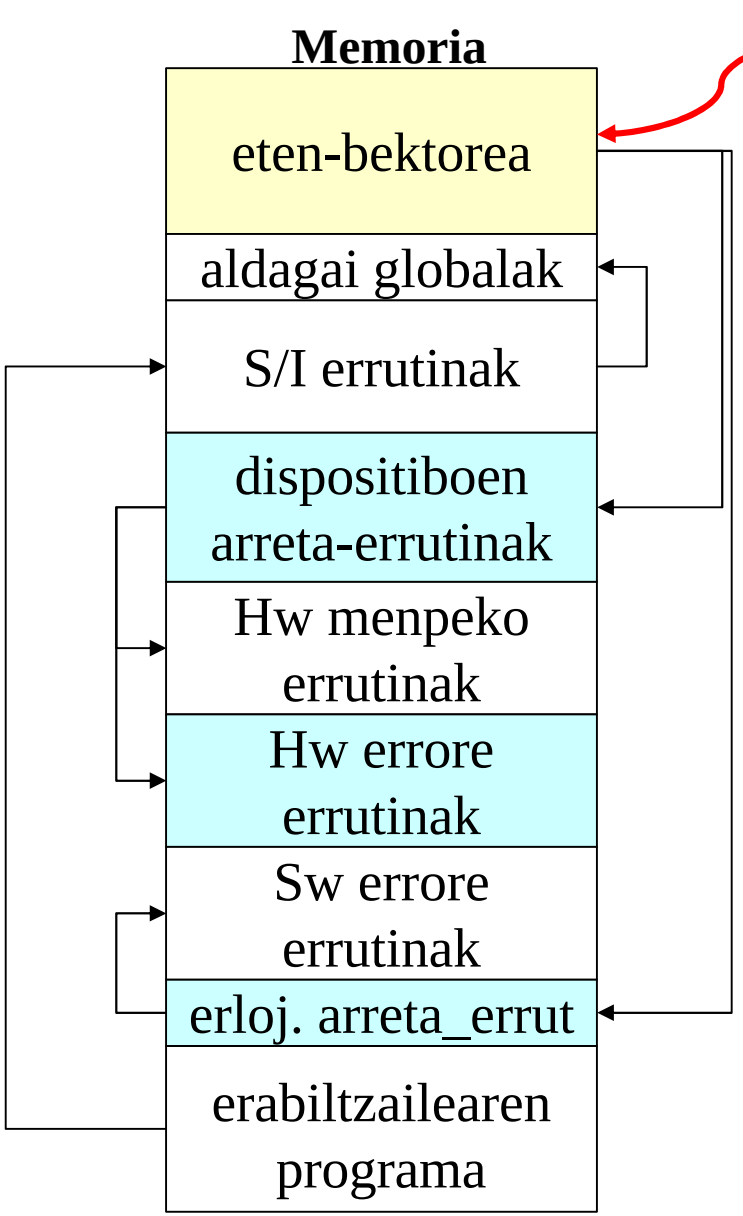
2 dispositiboaren arreta-errutina

```
arreta_errutina_2 ()
{
   int egoera;

   egoera=irakurri_egoera_erreg(KDMA);
   erroreak(KDMA, egoera);
   amaiera2 = TRUE;
   amaiera_eten_errut();   /* iret */
}
```

2 dispositiboaren irteera errutina

```
irt_errut_2 (char *bektorea, int kop, int asink_sink)
{
   int egoera;

   amaiera2 = FALSE;
   do {
      egoera=irakurri_egoera_erreg(KDMA);
   } while (egoera == LANEAN);
   erroreak(KDMA, egoera);
   idatzi_helbide_erreg(KDMA, &(bektorea[1]));
   idatzi_luzera_erreg(KDMA, kop-1);
   idatzi_kontrol_erreg(KDMA, HASI_IDAZKETA);
   programatu_kontroladorea(DISP2, bektorea[0]);
   if (asink_sink == SINKRO)
      sinkronizatu(&amaiera2);
}
```

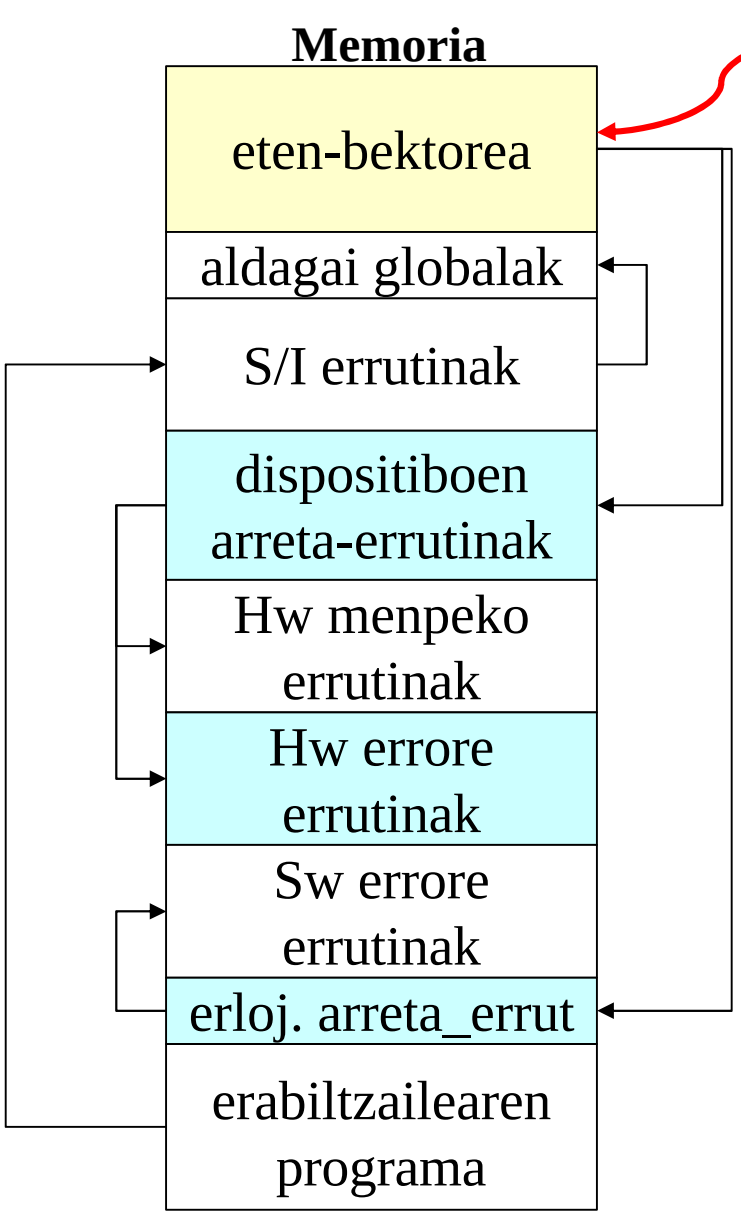
Denbora eta erroreen kontrola



Erlojuaren arreta errutina

```
arreta_errutina_erlojua ()
{
   static int konta_seg = 0, tik_kop = 0;

   if (max_segundo != 0) {
      tik_kop++;
      if (tik_kop == SEGUNDO_BAT) {
         konta_seg++;
         if (konta_seg == max_segundo)
            erroreak(ERLOJUA, PASA_DA);
         tik_kop = 0;
      }
   }
   amaiera_eten_errut();   /* iret */
}
```

Denbora eta erroreen kontrola



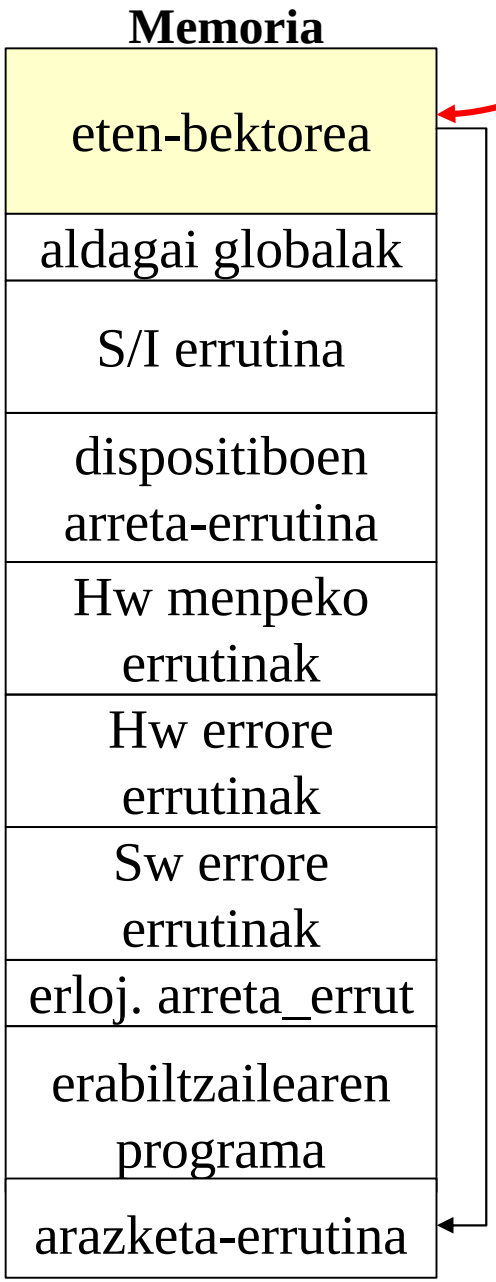
Aldagai globalak

```
...
int max_segundo = 0;
```

Erabiltzailearen programa

```
main ()
{
  char buff[80];

  aldatu_bektore_sarrera(ERLOJUA,
                         arreta_errutina_erlojua);
  max_segundo = DENBORA_MAXIMOA;
  /* Programaren gainontzeko kodea */
  ...
}
```

Arazketa



etenen bidezko mekanismoa

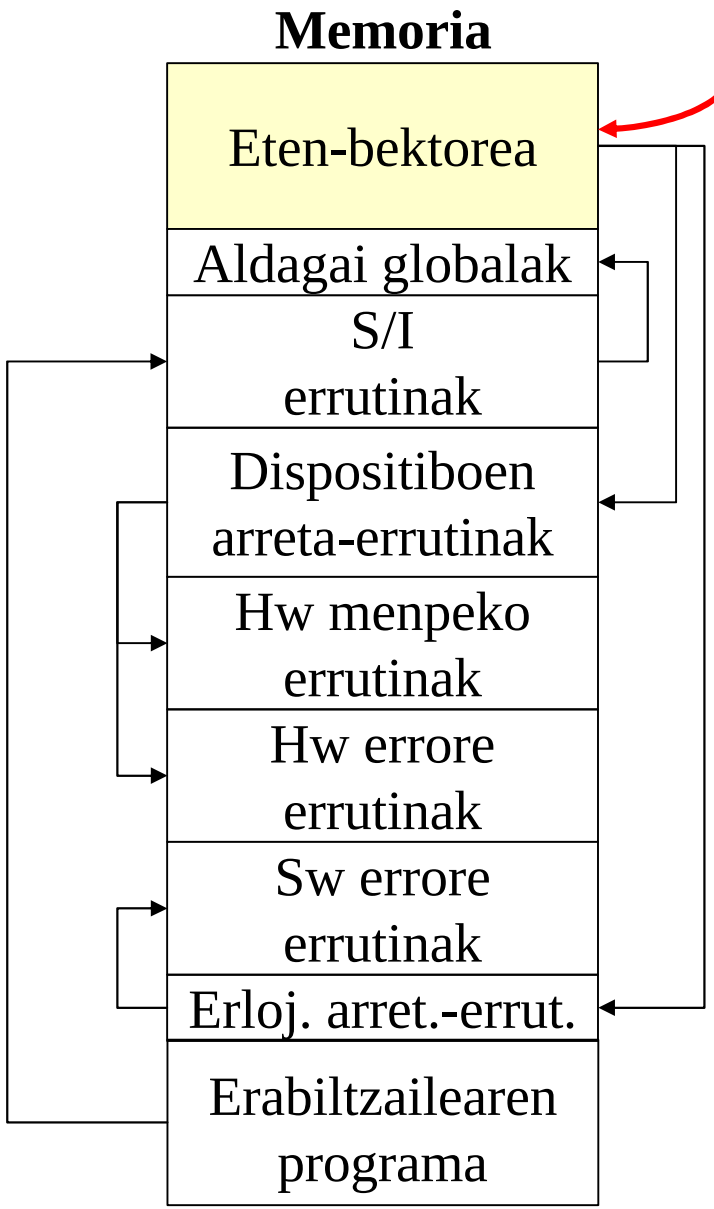
arazketa-errutina

```
arazketa_errut ()
{
   desaktibatu_aztarna_bita();

   /* araztatzailearen kodea */

   aktibatu_aztarna_bita();
}
```

aldaketak erabiltzailearen programan

```
main ()
{
   aldatu_bektore sarrera(ARAZKETA, arazketa_errut);
   ...
   aktibatu_aztarna_bita();

   /* programaren kodea */

   desaktibatu_aztarna_bita();
   ...
}
```

S/I-ren eskema orokorra (etenen bidezkoa)



Etenen bidezko mekanismoa

Mekanismoaren desabantailak:

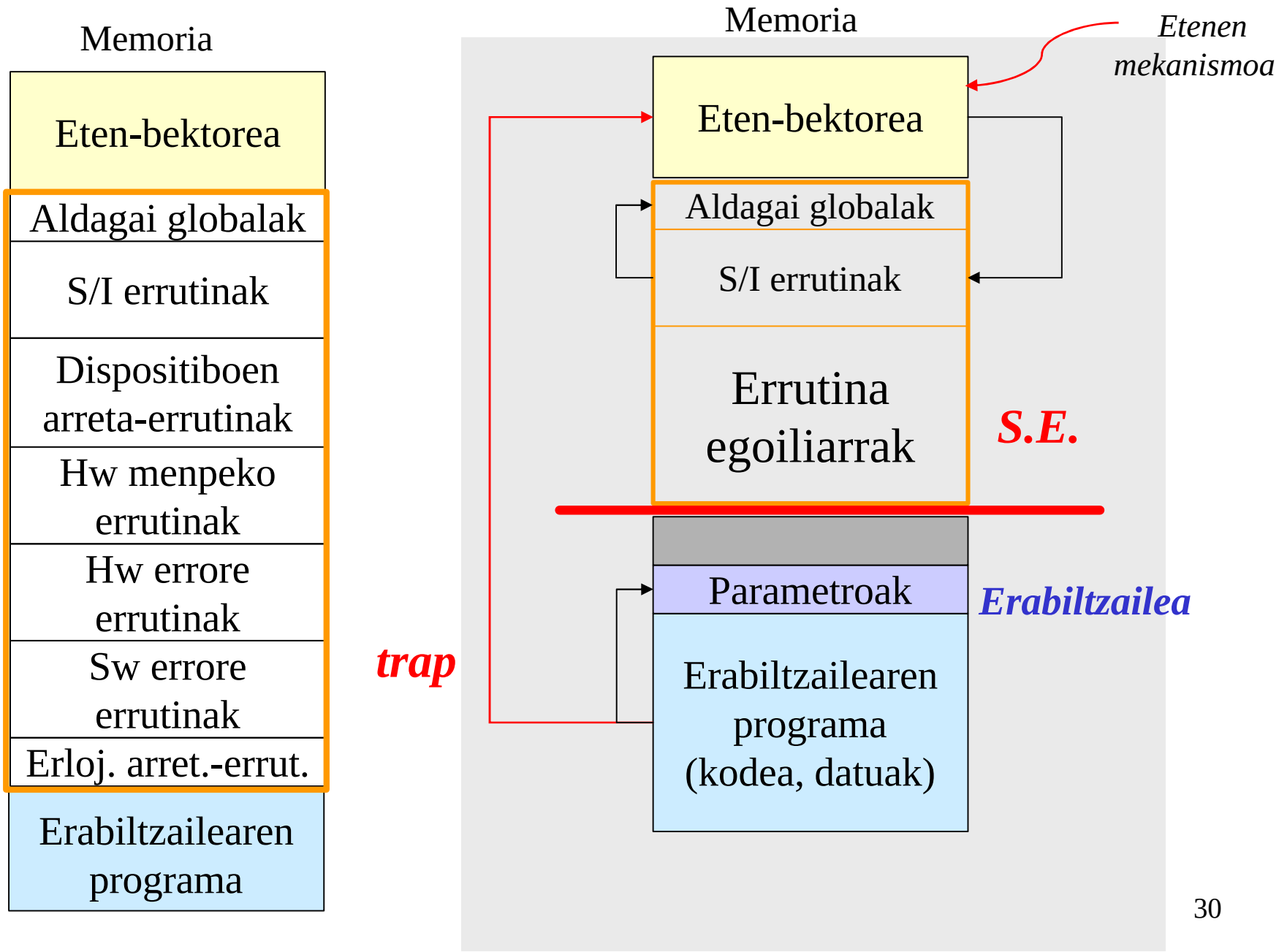
- Programatzaileen aldetik makinaren ezagutza
- Hw dependentziak, aldaketak?
- S/I errutinen eguneraketak
- Erabiltzailearen programen mantenketa, eta bere bizi-zikloa

Sistema-deiak

- Erabiltzaileen programen eta sistema eragilearen arteko interfazea (tartekaria)
- Eten mota berezi bat bezala prozesatzen dira (*TRAP x*)
- Sistema-dei bakoitzak zenbaki konkretu bat du. Prozesadoreetan agindu berezi bat erabili ohi da sistema-deiak inplementatzeko:

INT xx, int86()	(Intel x86)
SVC	(IBM 360/370) – supervisory call
trap	(PDP 11)
tw	(PowerPC) - trap word
tcc	(Sparc)
break	(MIPS)

Eten-bektorearen bidezko sistema-deia



Sistema-deiak

- Gaur egun, sistema-deiak goi-mailako programazio-lengoaietatik egin daitezke (adib., C)

- Programatzea errazagoa da:

  SVC 15 vs read(file-d, buffer, n-bytes)

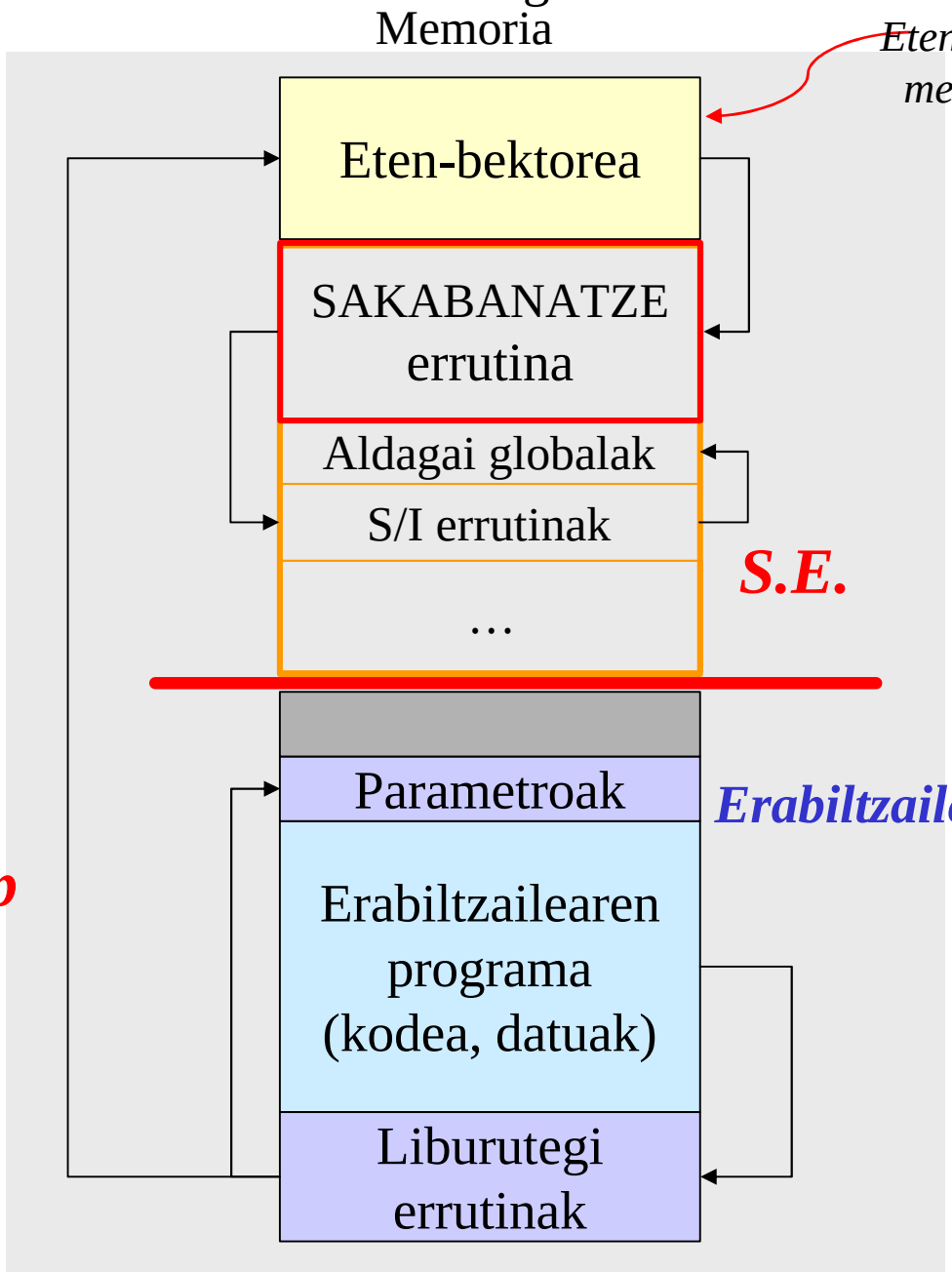
- Sistema-deiek makinaren baliabideak atzitzen duten aginduak exekutatzen dituzte, adibidez S/I aginduak dispositiboak atzitzeko

- Agindu hauek era kontrolatu batean exekuta daitezela komeni zaigu, Sistema Eragilearen kontrolpean!

Eten-bektorearen bidezko sistema-deia

Liburutegia + eten-bektorearen tamaina mugatzeko teknika

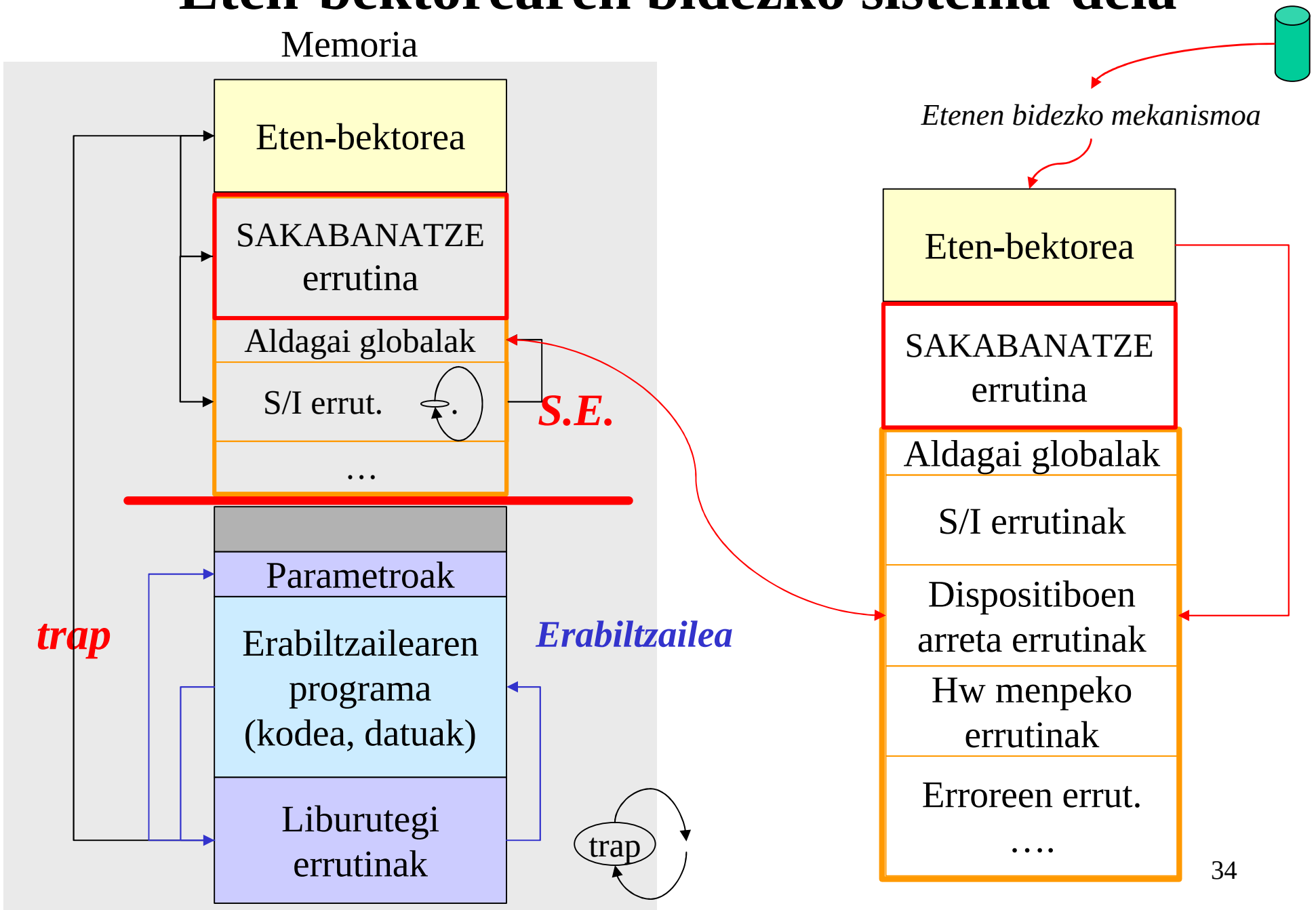
Aldagai globalak
S/I errutinak
Dispositiboen arreta errutinak
Hw menpeko errutinak
Hw errore errutinak
Sw errore errutinak
Erloj. arret.-errut.



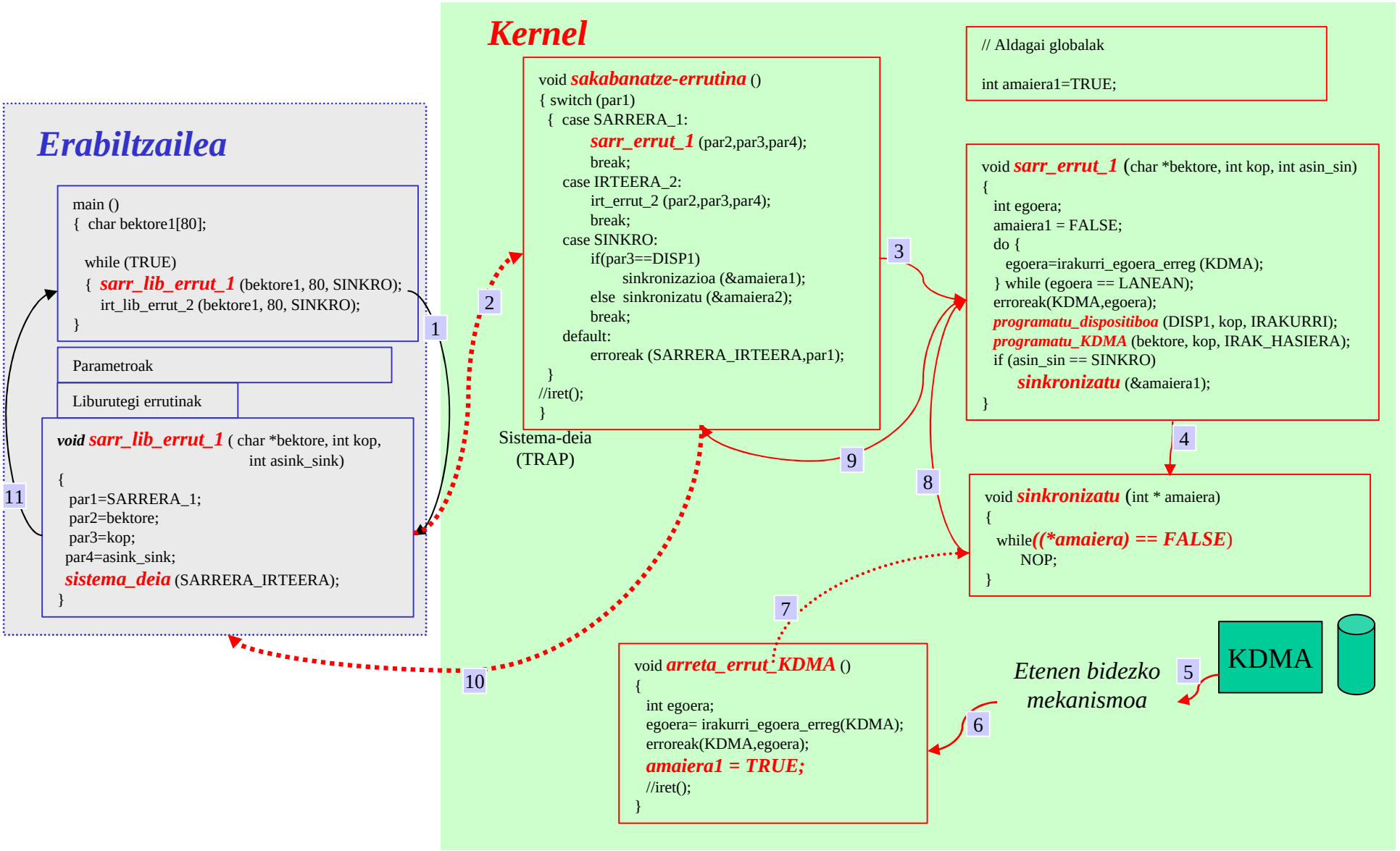
Abstrakzio teknikak

- Sistema-dei batzuk liburutegietako aginduen bidez estaltzen dira
- Adibidea: fopen() / fclose() – C abstrakzioak
  - Windows-en:
    - fopen() ⇨ CreateFile()
    - fclose() ⇨ CloseHandle()
  - Unix-en:
    - fopen() ⇨ open()
    - fclose() ⇨ close()

Eten-bektorearen bidezko sistema-deia



Eten-bektorearen bidezko sistema-deia



Liburutegiko errutinak

```
sar_lib_errut_1(char *bektore, int kop,
                                    int asink_sink)
{
   par1 = SARRERA_1;   par2 = bektore;
   par3 = kop;   par4 = asink_sink;
   sistema_deia(SARRERA_IRTEERA);
}
```

```
irt_lib_errut_2(char *bektore, int kop,
                                    int asink_sink)
{
   par1 = IRTEERA_2;   par2 = bektore;
   par3 = kop;   par4 = asink_sink;
   sistema_deia(SARRERA_IRTEERA);
}
```

```
sinkronizatu_lib_errut(int dispositiboa)
{  par1 = SINKRO;   par3 = dispositiboa;
   sistema_deia(SARRERA_IRTEERA);
}
```

parametroak pasatzeko aldagaiak

```
int par1, par3, par4; char *par2;
```

sakabanatze-errutina

```
sakabanatze_errut()
{  switch (par1) {
      case SARRERA_1:
         sar_errut_1(par2, par3, par4); break;
      case IRTEERA_2:
         irt_errut_2(par2, par3, par4); break;
      case SINKRO:
         if (par3 == DISP1)
            sinkronizatu(&amaiera1);
         else sinkronizatu(&amaiera2);
         break;
      default:
         erroreak(SARRERA_IRTEERA, par1);
   }
}
```

erabiltzailearen programa sinkronoa

```
main()
{  char bekt[80];
   while (TRUE) {
      sar_lib_errut_1(bekt, 80, SINKRO);
      irt_lib_errut_2(bekt, 80, SINKRO);
   }
}
```

erabiltzailearen programa sinkronoa

```
main()
{
   char bekt[80];

   while (TRUE)
   {
      sar_lib_errut_1(bekt, 80, SINKRO);
      irt_lib_errut_2(bekt, 80, SINKRO);
   }
}
```

erabiltzailearen programa asinkronoa

```
main()
{
   char bekt1[80], bekt2[80];

   while (TRUE)
   {
      sar_lib_errut_1(bekt1, 80, SINKRO);
      sinkronizatu_lib_errut(DISP2);
      irt_lib_errut_2(bekt1, 80, ASINK);
      sar_lib_errut_1(bekt2, 80, SINKRO);
      sinkronizatu_lib_errut(DISP2);
      irt_lib_errut_2(bekt2, 80, ASINK);
   }
}
```

Sistema-deiak (1)

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
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

Sistema-deiak (2)

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

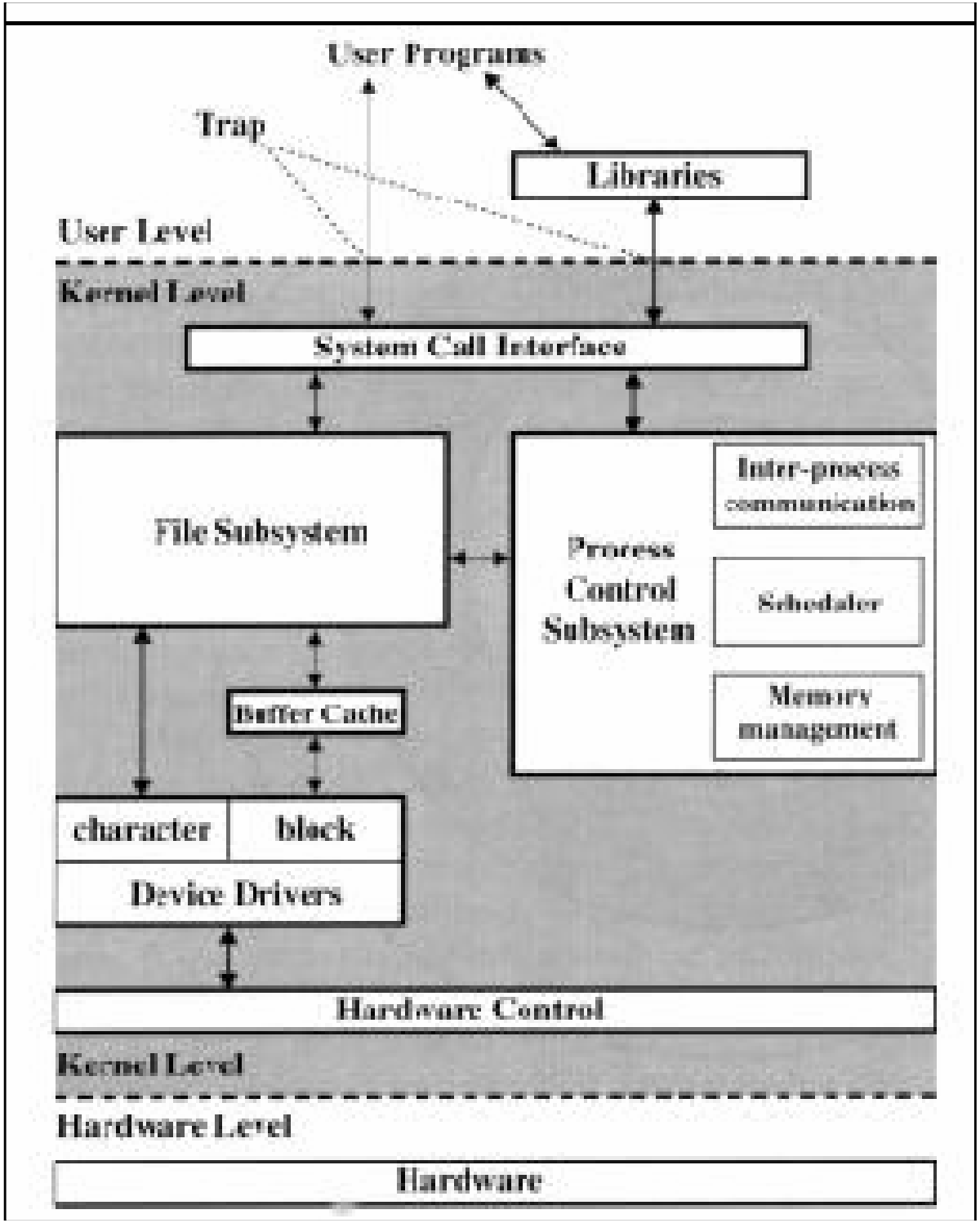
Sistema-deiak (3)

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970


User Programs
Trap
Libraries
User Level
Kernel Level
System Call Interface
File Subsystem
Inter-process communication
Process Control Subsystem
Scheduler
Memory management
Buffer Cache
character
black
Device Drivers
Hardware Control
Kernel Level
Hardware Level
Hardware

Windows NT/2000/XP

- Zer da sistema-dei bat Windows munduan?
  - Azpi-sistema batek eskainitako APIa?

  ala

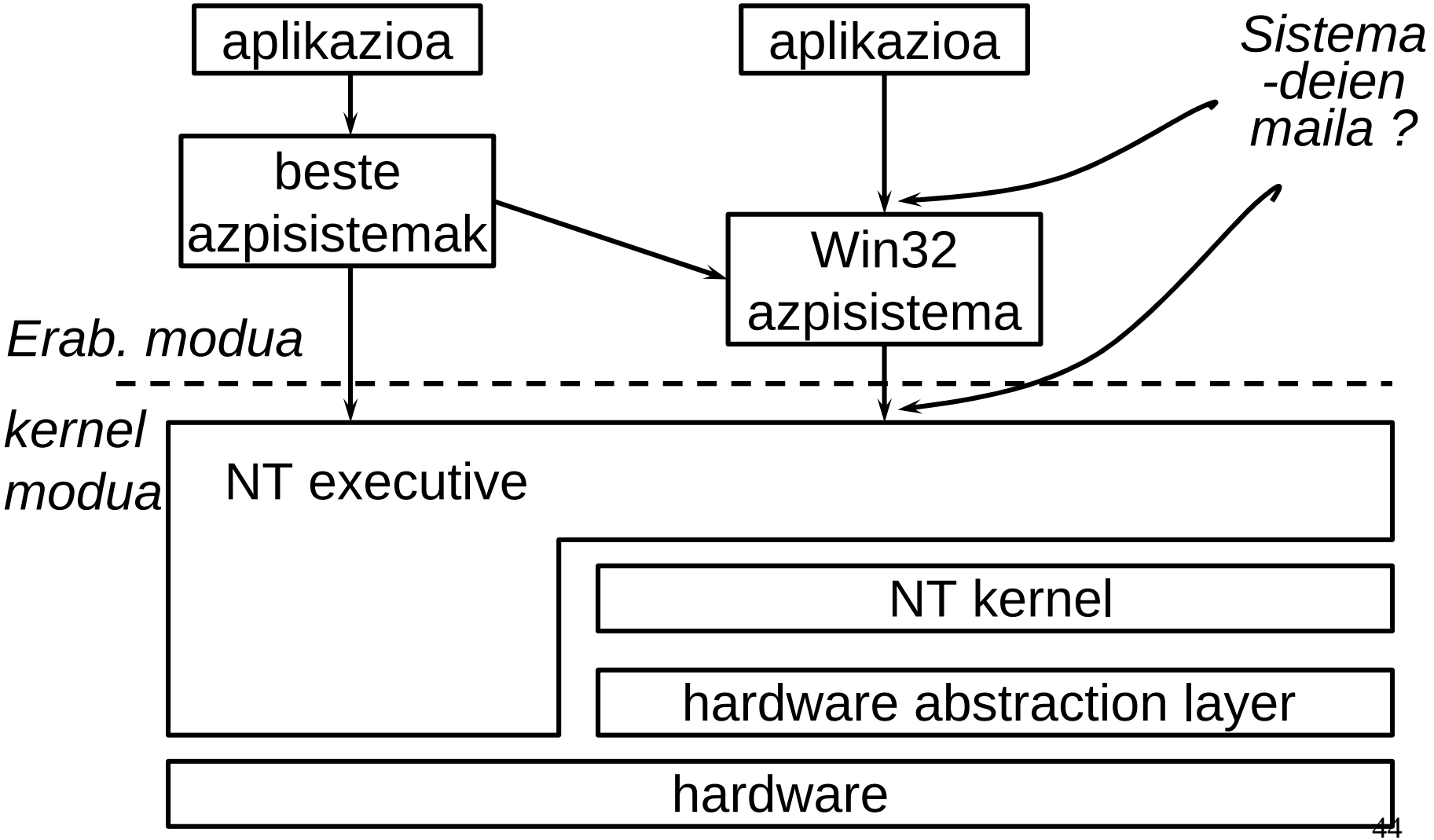
  - Oinarrizko APIa (*Native API*)?

Unix vs Windows NT/2000/XP

"UNIX applications can call kernel functions, or *system calls*, directly. In Windows NT, applications call APIs that the OS environment to which they are coded (Win32, DOS, Windows 3.x, OS/2, POSIX) exports. The NT system-call interface, called the Native API, is hidden from programmers and largely undocumented (>1000 system calls)."

"The number of UNIX system calls is around 200 to 300. The API that UNIX applications write to is the UNIX system-call interface, whereas the API that the majority of NT applications write to is the Win32 API, which translates Win32 APIs to Native APIs."

Windows 2000/NT/XP



API-ak

- Zenbait SEk (adib., Windows) ez dute publiko egiten kernel deien zerrenda osoa
  - API kontzeptua liburutegi mailako funtzien mailan
- Besteak (adib., UNIX & Linux) publiko egiten dituzte liburutegi eta kernel funtzioak
  - Bi *mailek* alegiazko makina osoa definitzen dute
- Eztabaidatu kernel funtzioak publiko egitearen abantaila eta desabantailak

- Sistema-dei mota arruntak:
  - Prozesuen kudeaketa
    - Create/Destroy
    - supend/activate
    - Prozesuen arteko komunikazioa
  - Memoria kudeketa
    - Allocate/deallocate memory
    - Increase/decrease memory
  - Fitxategien kudeaketa
    - create/destroy, open/close
    - Move, rename
    - aldatu fitxategien ezaugarriak
  - S/I errutinak
  - Beste Funtzioak
    - get/set data eta ordua, alarmak
    - generate diagnostics
    - GUI

  …

Adibidea

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

int main() {
   unsigned long t;

   while (1) {
      t = time(0);
      printf("time(0) = %ld -> %s\n", t, ctime(&t));
      sleep(1);
   }
}
```

Adibidea

```
[acaf0266@acpt00 acaf0266]$ ls -l
total 16
-rwxr-xr-x    1 acaf0266 acaf        11908 feb 27 17:25 ordua
-rw-r--r--    1 acaf0266 acaf          183 feb 27 15:17 ordua.c

[acaf0266@acpt00 acaf0266]$ ordua
time(0) = 1172596555 -> Tue Feb 27 18:15:55 2007
time(0) = 1172596556 -> Tue Feb 27 18:15:56 2007
time(0) = 1172596557 -> Tue Feb 27 18:15:57 2007
time(0) = 1172596558 -> Tue Feb 27 18:15:58 2007
time(0) = 1172596559 -> Tue Feb 27 18:15:59 2007
time(0) = 1172596560 -> Tue Feb 27 18:16:00 2007
time(0) = 1172596561 -> Tue Feb 27 18:16:01 2007
```