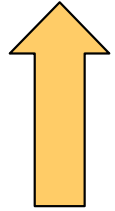


6. Gaia. Multiprogramazioa eta *multithreading*: *Programa independenteak*

1. Motibazioa.
2. Exekuzio-fluxuak: prozesuak eta hariak.
3. Exekuzio-testuingurua.
4. Egoerak eta trantsizioak.
5. Sistema-deiak.
6. Sistema Eragile baten lan egiteko modu orokorra.

Motibazioa



*Prozesadorearen eta S/Iko dispositiboen arteko abiadura aldea.
Prozesadorearen geldiuneak aprobetxatzeko mekanismoak.*

a) S/I Asinkronoak (S/I bitartean, beste zerbait egin)

- ✓ Aginduen berrordenatzea.
- ✓ Ez da teknika gardena: programazioa zaildu egiten du.
- ✓ Sinkronizazio esplizitua behar du, transferentzia bukatu den jakiteko.

b) Buffering (S/Iak paraleloan egin programarekin)

S/Iko eragiketak buffer izeneko tamaina finkoko memoria-blokeetan burutzen ditu SEak. Programak bloke osoa kontsumitu arte, ez dago transferentzia berririk egin beharrik.

Buffer bikoitza: irakurketa aurreratua beste buffer batean.

- S/Ietara zuzendutako programak: bufferrak hutsik/beteta.
- Bufferrek memoria okupatzen dute.

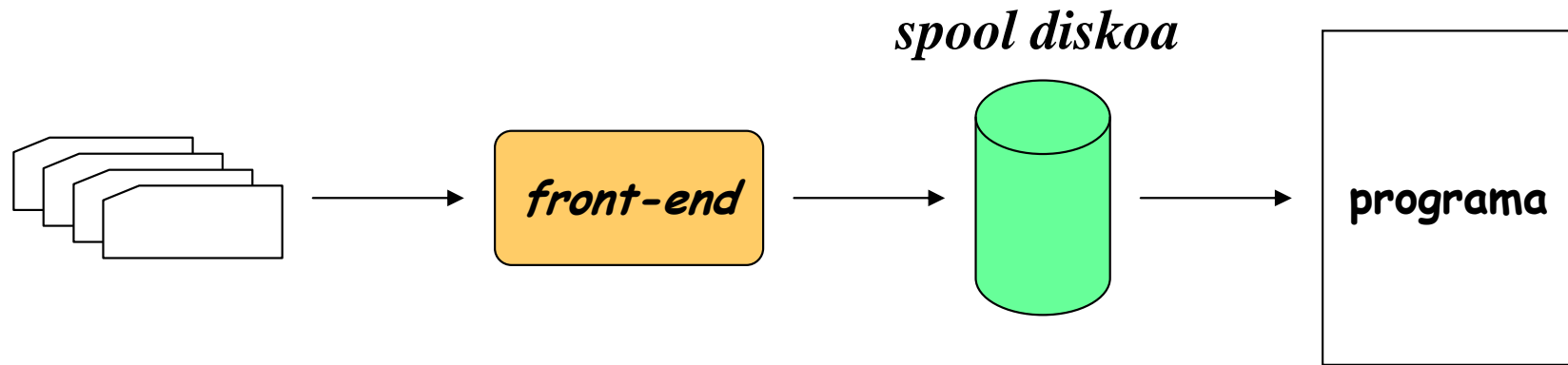
c) Spooling (hardware berezia erabili S/Iak azkartzeko)

S/Iko dispositibo motelak era ez elkarreragilean erabiltzen direnean, tarteko dispositibo azkarrago bat erabil daiteke.

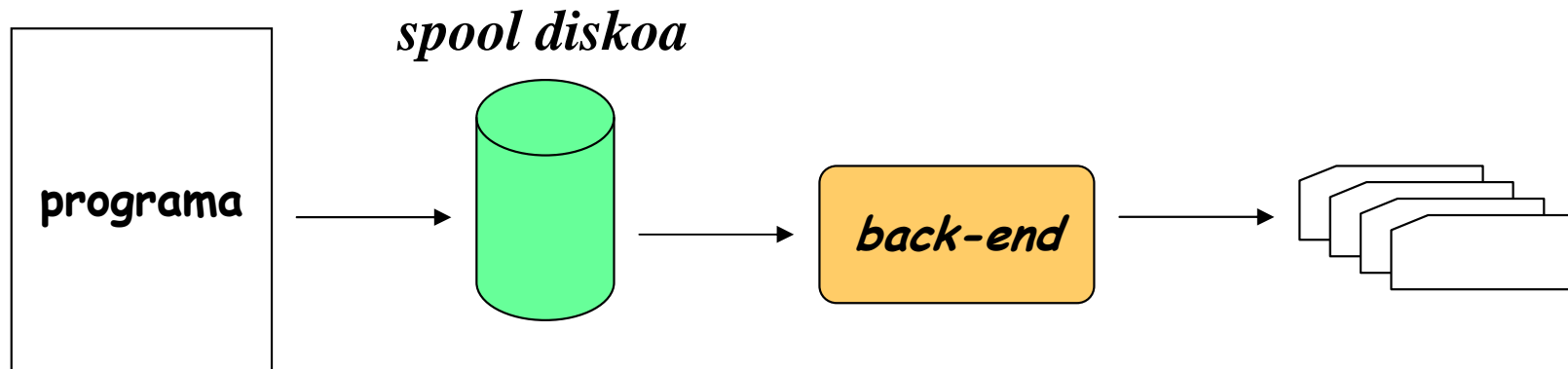
Irakurketetan: hardware berezia (aurreko-prozesadorea edo *front-end*).

Idazketetan: hardware berezia (*back-end*) edota SEaren laguntza.

Spooling (simultaneous peripheral operations on-line)

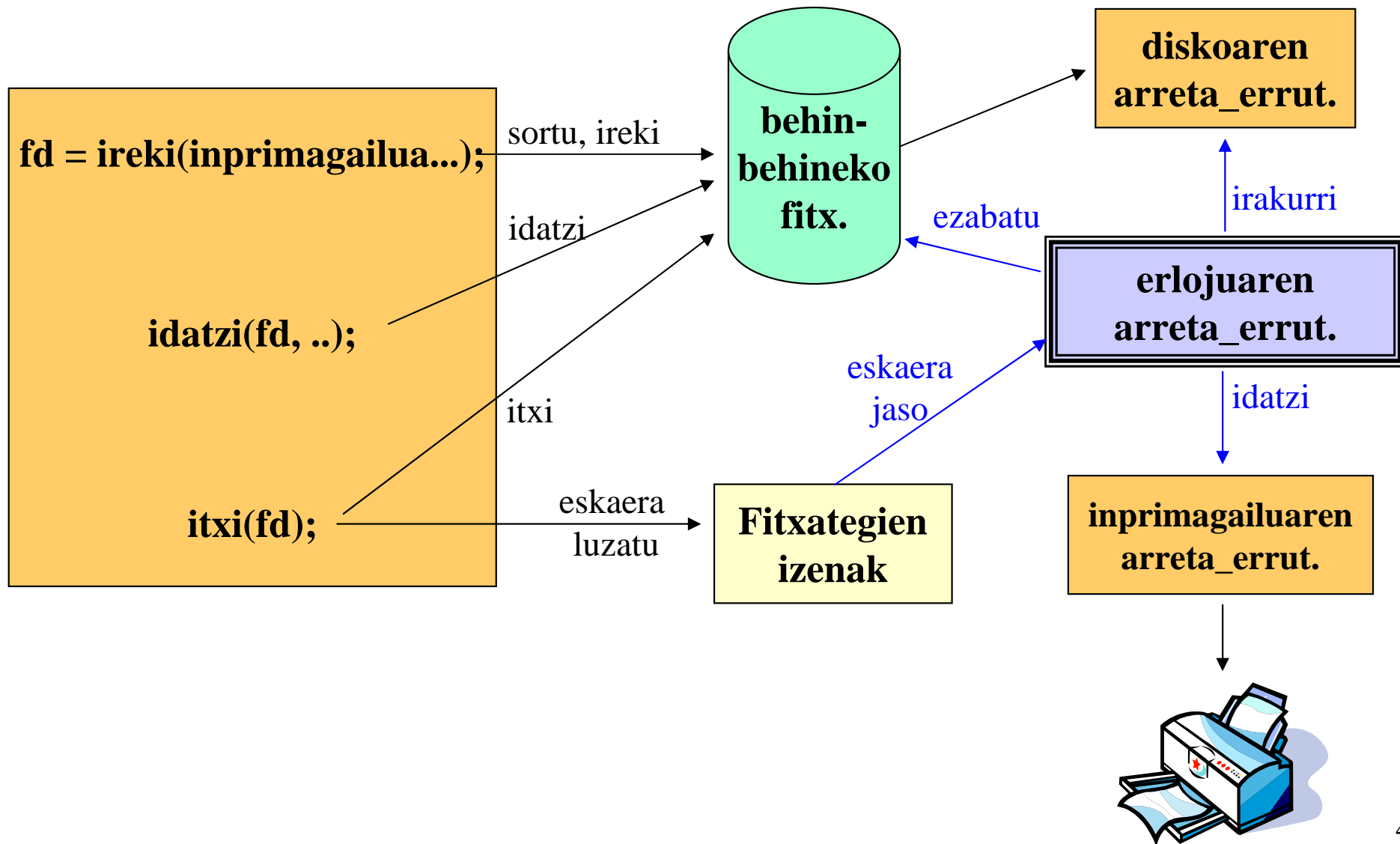


Sarrerarako dispositibo motelak



Irteerarako dispositibo motelak

Inprimagailuaren *spooling*



Abantailak - Eragozpenak

↳ Abantailak

- ✓ *Buffering*: S/Ia programa beraren aginduekin paraleloan.
- ✓ *Spooling*: S/Ia beste programa batekin paraleloan.
- ✓ *Spooling*: erraza lanaren kopia bat baino gehiago lortzea.

↳ Eragozpenak

- ✓ Diskoan espazioa okupatzen da (behin-behineko fitx.).
- ✓ Tratamendu-denbora luzeagoa da.

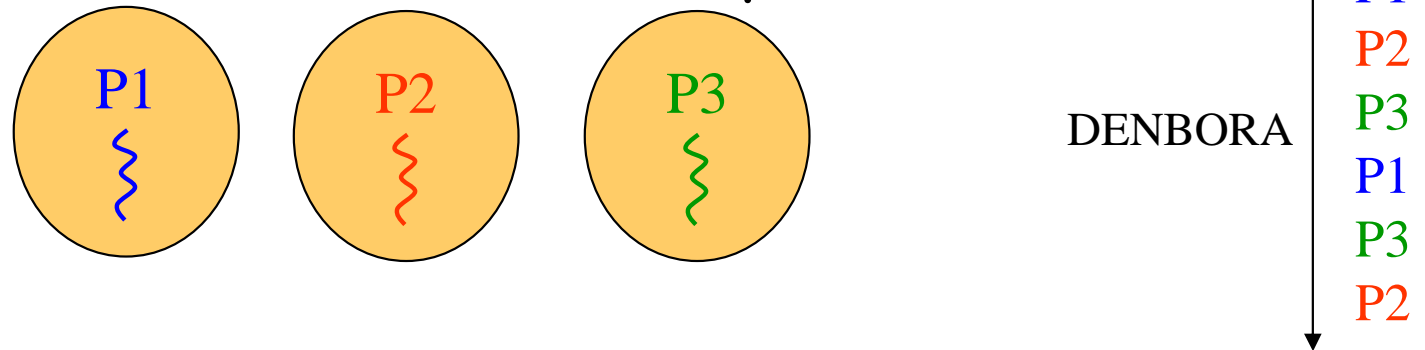
Hala ere, teknika hauek ez dute arazoa guztiz konpontzen:
Erabiltzaile batek ez du CPUa denbora osoan okupaturik edukitzen.

□ Soluzioa: konkurrentzia programen artean

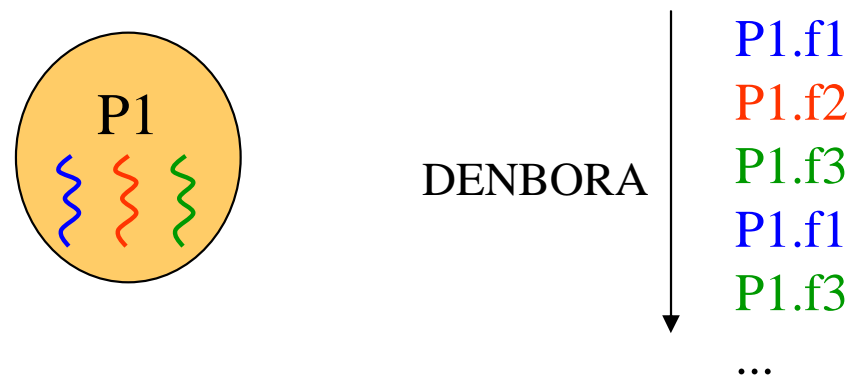
- Memoria nahikoa badago *programa bat baino gehiago kargatu*.
- Exekutatzen ari den programa S/I baten zain gelditzen denean, beste programaren bat prest badago, exekutatu dadila.

Exekuzio-fluxuak: prozesuak, hariak

MULTIPROGRAMAZIOA: *Programa bat baino gehiago* memorian *bere kodearen exekuzioa denboran txandakatuz*, denak SEaren kontrolpean



MULTIFLUXUA: *Exekuzio-fluxu bat baino gehiago* duen programa bat, *fluxu bakoitzaren exekuzioa denboran txandakatuz*, denak SEaren kontrolpean



Exekuzio-testuingurua

SE multiprogramatu batek programek, bakarrik (*monoprogramazioa*) nahiz beste programa batzuekin konkurrentzian exekutatzeko (*multiprogramazioa*), bere eginkizuna ondo betetzen dutela ziurtatu behar du.

SEak programa bakoitzari buruzko informazioa eduki behar du, CPUaren kontrola programa batetik beste batera aldatu ahal izateko: **testuinguru-aldaketa**.

PROZESU = *Testuinguru-informazioa* + programa (agind. eta datuak).

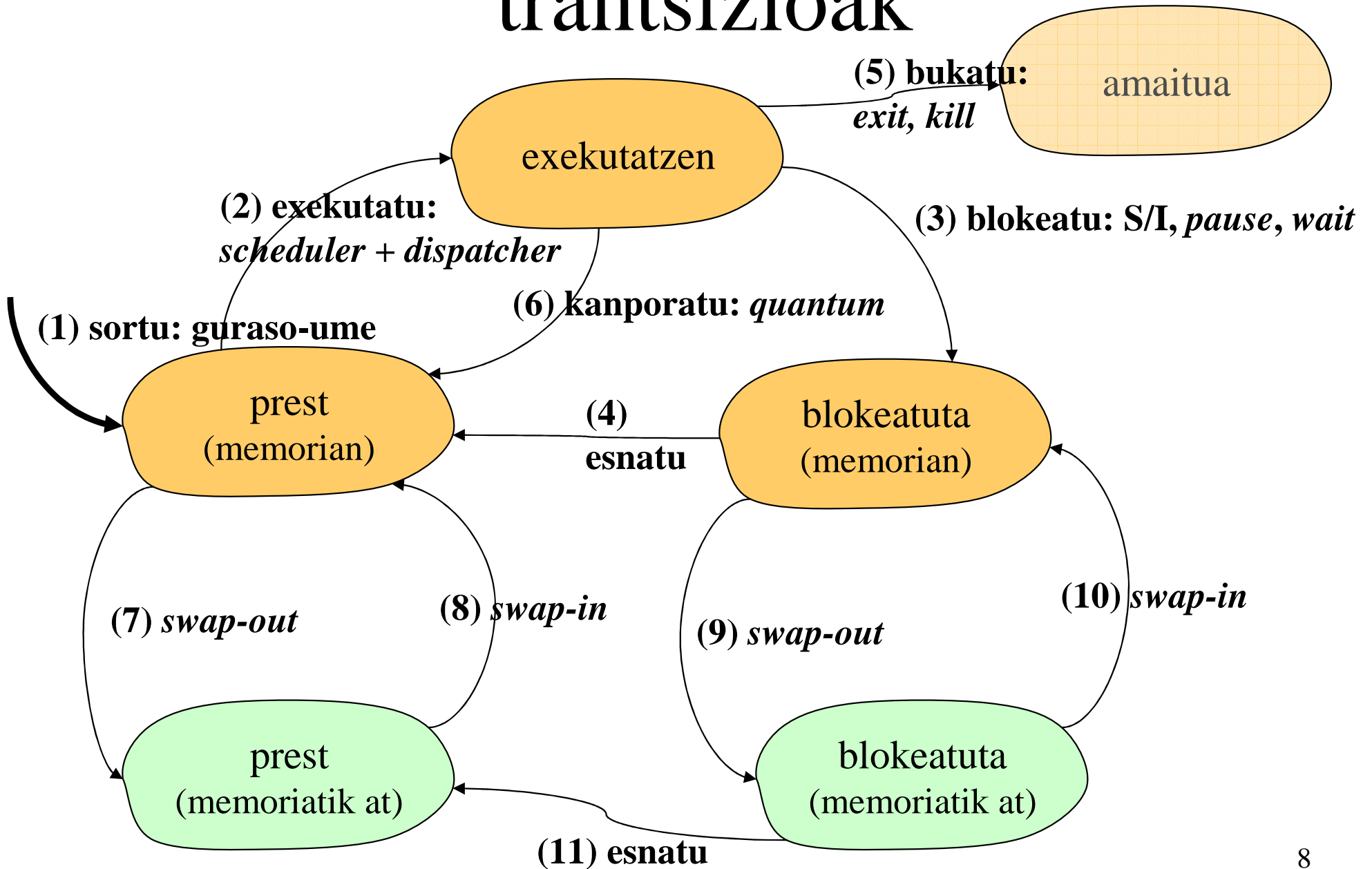
Testuinguru-informazioa:

- Identifikadorea
- Kanal-taula
- CPUaren erregistroen balioak
- ...

PROZESU: exekutatzeko ari den programa baten instantzia.

Multiprogramazio-maila: SEak duen prozesu-kopurua.

Prozesuen egoerak eta trantsizioak



Prozesuen egoerak eta trantsizioak

(1) Sortu

NOIZ: Sistema-dei berezi baten exekuzioa (fork)

EGINBEHARRA:

Identifikadore bat esleitu.

Programa memorian kargatu (beharrezkoa bada).

Testuinguru informazioa hasieratu (kanal-taula, ...).

Guraso prozesua - Ume prozesua.

Huts egin dezake, adibidez programa ez bada aurkitzen, behar beste memoriarik ez badago, ...

Prozesuen egoerak eta trantsizioak

(2) Exekutatu

NOIZ: Exekutatzen ari den prozesua blokeatzen denean, bukatzen duenean edota **quantum**-a bukatzen zaionean.

EGINBEHARRA:

SCHEDULER: SEaren funtzioa, exekutatuko den prozesua aukeratzen duena. **Lehentasun** estatikoak ala dinamikoak.

DISPATCHER: aukeratutako prozesua martxan jartzen arduratzen den SEaren funtzioa (testuinguru informazioa ezartzen du).

Prozesu NULUA.

(3) Blokeatu

NOIZ: Sistema-deia blokeagarri bat egitea:

- S/I
- Denbora itxoite bat
- Beste prozesu batekin sinkronizatzea

...

EGINBEHARRA:

SEak prozesuaren testuinguru informazioa gorde behar du. Beste prozesu bat exekutatzen jarri behar da (2).

Prozesuen egoerak eta trantsizioak

(4) Esnatu

NOIZ: SEaren errutina **asinkrono** baten exekuzioa (adibidez, dispositibo baten arreta errutina).
Sinkronizazio baten bukaera.

...

EGINBEHARRA:

Esnatzen den prozesua prest egoeran ipini.

Esnatzen den prozesua exekutatzera pasatzeko aukera eman daiteke. Horretarako exekutatzen ari den prozesua prest egoeran ipintzen da (CPU mailako kanporatzea) eta scheduler eta dispatcher funtzioak berriro exekutatzen dira.

Prozesuen egoerak eta trantsizioak

(5) Bukatu

NOIZ: Bukatzeko sistema-deia (exit).

Beste prozesu bat bukarazteko sistema-deia (kill).

EGINBEHARRA:

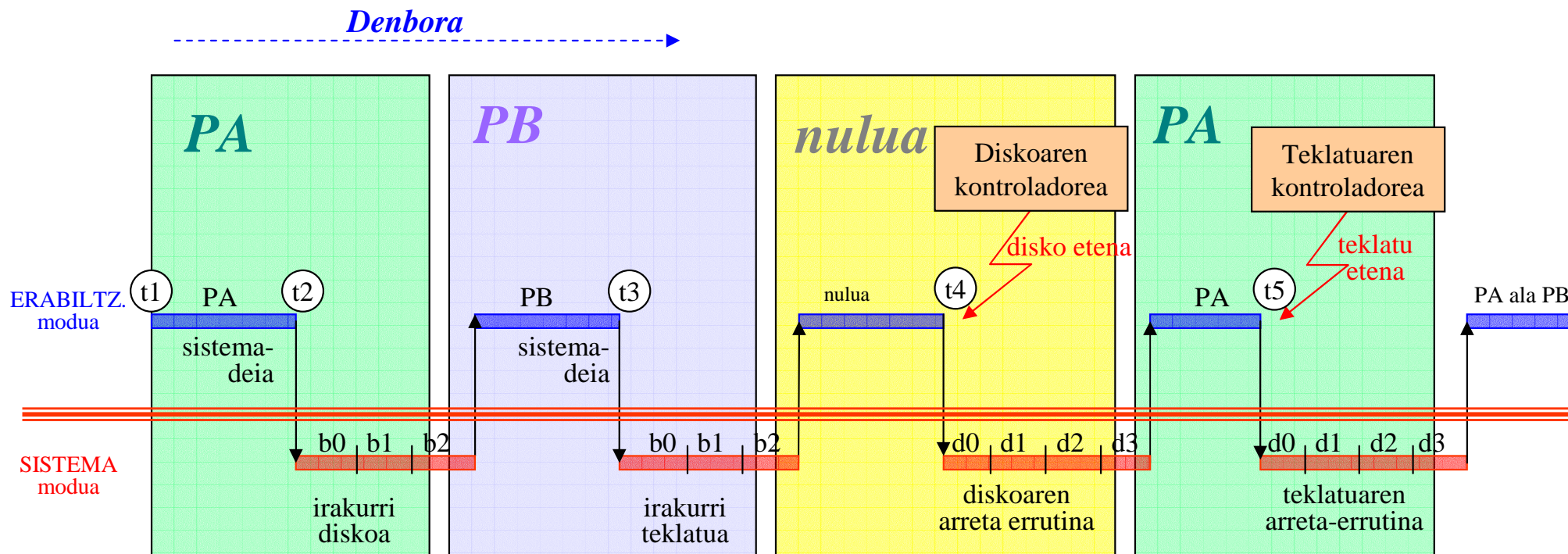
SEak prozesua erabiltzen ari zen baliabideak askatu behar ditu.

Amaitua egoerara pasatzen da prozesua.

Bukaera kodea gorde egiten da gurasoarentzat.

Gurasoak umeren baten zain gelditzen denean (wait) ume baten bukaera kodea jasoko du, eta SEak ume horren informazioa ezabatuko du. Umeak baditu eta inork ez badu bukatu, orduan guraso blokeatu egingo da ume batek bukatu arte.

Egoera-aldaketen adibidea



Ⓣ1 Hasierako egoera:
PA: exekutatzen
PB, nulua: prest

Ⓣ2 PA-k sistema-deia exekutatzen du
b0: Programatu kontroladorea
b1: Blokeatu PA prozesua
b2: *scheduler* eta *dispatcher*

Ⓣ3 PB-k sistema-deia exekutatzen du
b0: Programatu kontroladorea
b1: Blokeatu PB prozesua
b2: *scheduler* eta *dispatcher*



Ⓣ4

Disko-kontroladorearen etena
Arreta-errutina:
d0: Emaitzak egiaztatu
d1: Desblokeatu eragiketa
agindutako prozesua

KANPORAKETA politikaren kasuan:
d2: Kanporatu exekutatzen dagoen prozesua
d3: *scheduler* eta *dispatcher*

Ⓣ5
Teklatu-kontroladorearen
etena
Arreta-errutina:
(t4 bezala)

Egoera-aldaketen adibidea

Gertaera	Exekututzen	Prest	Blokeatuta	Amaituta
... Irakurri(Disk)	PA	PB Nulua		
... Irakurri(Tekl)	PB	Nulua	PA(Disk)	
...  Disk_Arret_Errut()	Nulua		PA(Disk) PB(Tekl)	
...  Tekl_Arret_Errut()	PA	Nulua	PB(Tekl)	
...	PA ala PB	PA ala PB Nulua		

Sistema monoprogramatuekiko desberdintasunak

➤ Prozesuen kudeaketa:

- Behar berriak: sinkronizazioa, komunikazioa, ...

➤ Memoriaren kudeaketa:

- Programa anitz memorian edukitzea oinarrizko bihurtzen da.
- Programak ez osoan memorian eta estekatze dinamikokoak garrantzia hartzen dute programak konkurrentzian exekutatzekoan.
 - Programen kodea konpartitzeko aukera.

➤ S/I-en kudeaketa:

- Baliabideak:
 - Konpartigarriak (diskoa).
 - Serialki berrerabilgarriak (inprimagailua).

Sistema monoprogramatuekiko desberdintasunak

➤ Sistema-deiak:

- Prozesuen eta denboren kontrola.
- Sinkronizazioa eta komunikazioa.

➤ Komando-Interpretatzailea:

- Programak atzeko planoan exekutatzeko aukera (*spawn, background, paraleloan*): &
 - Postubakarreko sistemetan multiprog. modu bakarra.
- Prozesuen arteko komunikazioetan informazioa gordetzeko tokiak (*pipe, buzoi, ...*): `ls | more`
- Beste KI bat paraleloan exekutatzeko aukera, komando fitxategiak (*script*) exekutatzeko.

Prozesuen kontrolerako sistema-deiak Unix-en

- Prozesuen identifikazioa
 - getpid, getppid, getuid
- Prozesuen sorrera
 - fork, exec??
- Prozesuen bukaera
 - exit, wait, kill
- Denboraren kontrola eta seinaleak
 - alarm, pause, signal, time, ctime

Prozesuen identifikazioa

int getpid();

- Prozesuaren identifikadorea (pid) bueltatzen du

int getppid();

- Gurasoaren identifikadorea (pid) bueltatzen du

int getuid();

- Prozesuaren “jabea” den erabiltzailearen identifikadorea (uid) bueltatzen du

Prozesuen identifikazioa

```
main()    /* adibide1 */
{
    int id_prozesua, id_gurasoa, id_erabiltzailea;

    id_prozesua = getpid();
    id_gurasoa = getppid();
    id_erabiltzailea = getuid();

    printf("Prozesua: %d\n", id_prozesua);
    printf("Gurasoa: %d\n", id_gurasoa);
    printf("Erabiltzailea: %d\n", id_erabiltzailea);
    exit(0);
}
```

Prozesuen identifikazioa

Prozesua: 3456

Gurasoa: 2121

Erabiltzailea: 500

Prozesuen sorrera

int fork();

- Sistema-dei honek prozesu berri bat sortzen du, prozesu deitzailearen “klona” dena. Prozesu deitzaileari gurasoa deituko diogu, eta sortutako prozesuari umea
- Umeak bere exekuzioa **fork**-aren hurrengo agindutik hasten du (gurasoak ere bai)
- Umeak gurasotik dena heredatzen du...

Prozesuen sorrera



Prozesuen sorrera

int fork();

- Umeak gurasotik dena heredatzen du...
- ... baina umeak **pid** desberdina du!
- **fork** deiak honakoa bueltatzen du:
 - Umeari: 0 (zero)
 - Gurasoari: umearen **pid**-a

Prozesuen sorrera

```
main()    /* adibide2 */
{
    int pid;

    pid = fork();

    if (pid == 0)    /* umea */
        printf("%d umea naiz, %d gurasoarena\n",
                getpid(), getppid());
    else    /* gurasoa */
        printf("%d gurasoa naiz, %d umearena\n",
                getpid(), pid);

    printf("Bukatzera noa %d\n", getpid());    /* biak */
    exit(0);
}
```

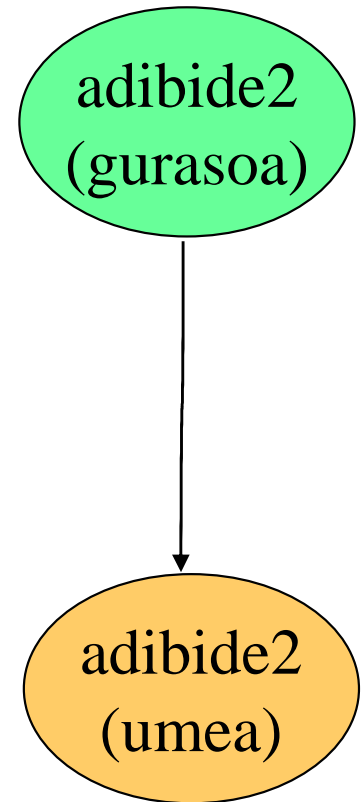
Prozesuen sorrera

```
main()    /* adibide2 */
{
    int pid;

    pid = fork();

    if (pid == 0)    /* umea */
        printf("%d umea naiz, %d gurasoarena\n",
                getpid(), getppid());
    else    /* gurasoa */
        printf("%d gurasoa naiz, %d umearena\n",
                getpid(), pid);

    printf("Bukatzera noa %d\n", getpid());    /* biak */
    exit(0);
}
```



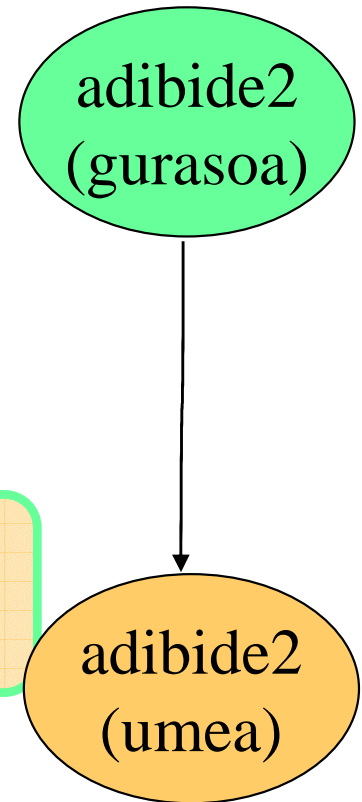
Prozesuen sorrera

```
main()    /* adibide2 */
{
    int pid;

    pid = fork();

    if (pid == 0)    /* umea */
        printf("%d umea naiz, %d gurasoarena\n",
                getpid(), getppid());
    else    /* gurasoa */
        printf("%d gurasoa naiz, %d umearena\n",
                getpid(), pid);

    printf("Bukatzera noa %d\n", getpid());    /* biak */
    exit(0);
}
```



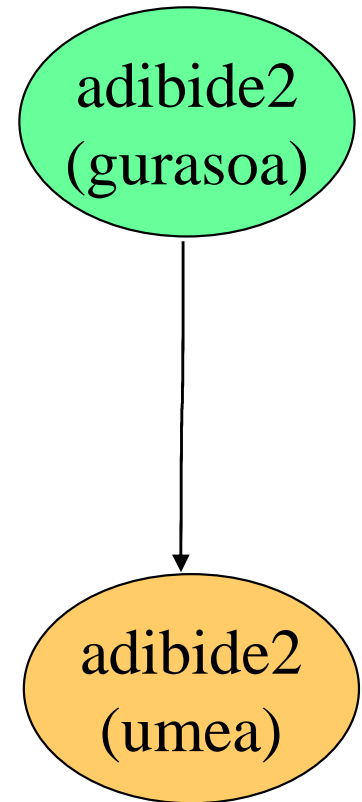
Prozesuen sorrera

```
main()    /* adibide2 */
{
    int pid;

    pid = fork();

    if (pid == 0)    /* umea */
        printf("%d umea naiz, %d gurasoarena\n",
                getpid(), getppid());
    else    /* guraso */
        printf("%d guraso naiz, %d umearena\n",
                getpid(), pid);

    printf("Bukatzera noa %d\n", getpid());    /* biak */
    exit(0);
}
```



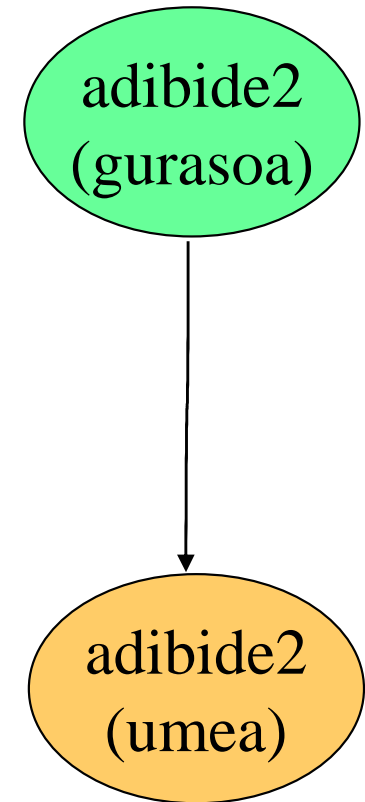
Prozesuen sorrera

```
main()    /* adibide2 */
{
    int pid;

    pid = fork();

    if (pid == 0)    /* umea */
        printf("%d umea naiz, %d gurasoarena\n",
                getpid(), getppid());
    else    /* gurasoa */
        printf("%d gurasoa naiz, %d umearena\n",
                getpid(), pid);

    printf("Bukatzera noa %d\n", getpid());    /* biak */
    exit(0);
}
```



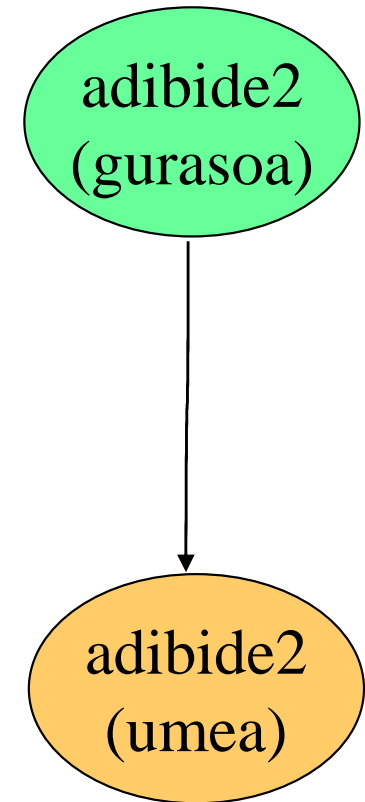
Prozesuen sorrera

```
main()    /* adibide2 */
{
    int pid;

    pid = fork();

    if (pid == 0)    /* umea */
        printf("%d umea naiz, %d gurasoarena\n",
                getpid(), getppid());
    else    /* gurasoa */
        printf("%d gurasoa naiz, %d umearena\n",
                getpid(), pid);

    printf("Bukatzera noa %d\n", getpid());    /* biak */
    exit(0);
}
```



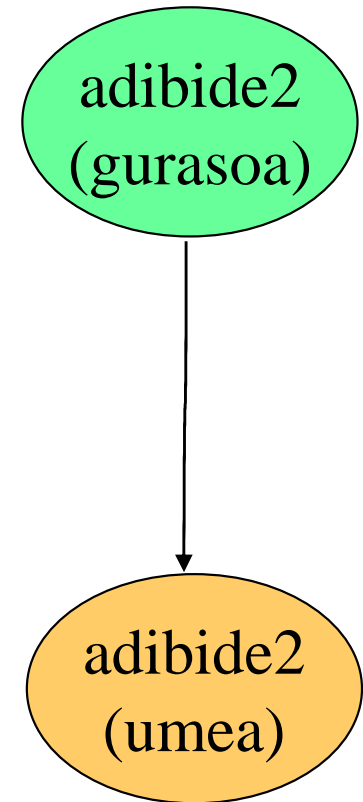
Prozesuen sorrera

```
main()    /* adibide2 */
{
    int pid;

    pid = fork();

    if (pid == 0)    /* umea */
        printf("%d umea naiz, %d gurasoarena\n",
                getpid(), getppid());
    else    /* guraso */
        printf("%d guraso naiz, %d umearena\n",
                getpid(), pid);

    printf("Bukatzera noa %d\n", getpid());    /* biak */
    exit(0);
}
```



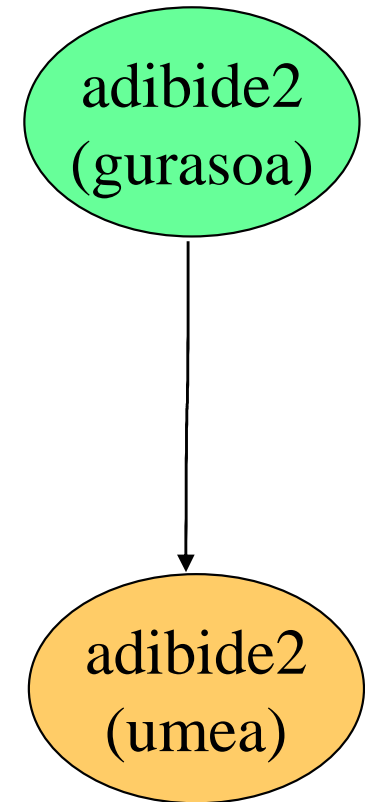
Prozesuen sorrera

```
main()    /* adibide2 */
{
    int pid;

    pid = fork();

    if (pid == 0)    /* umea */
        printf("%d umea naiz, %d gurasoarena\n",
                getpid(), getppid());
    else    /* gurasoa */
        printf("%d gurasoa naiz, %d umearena\n",
                getpid(), pid);

    printf("Bukatzera noa %d\n", getpid());    /* biak */
    exit(0);
}
```



Prozesuen sorrera

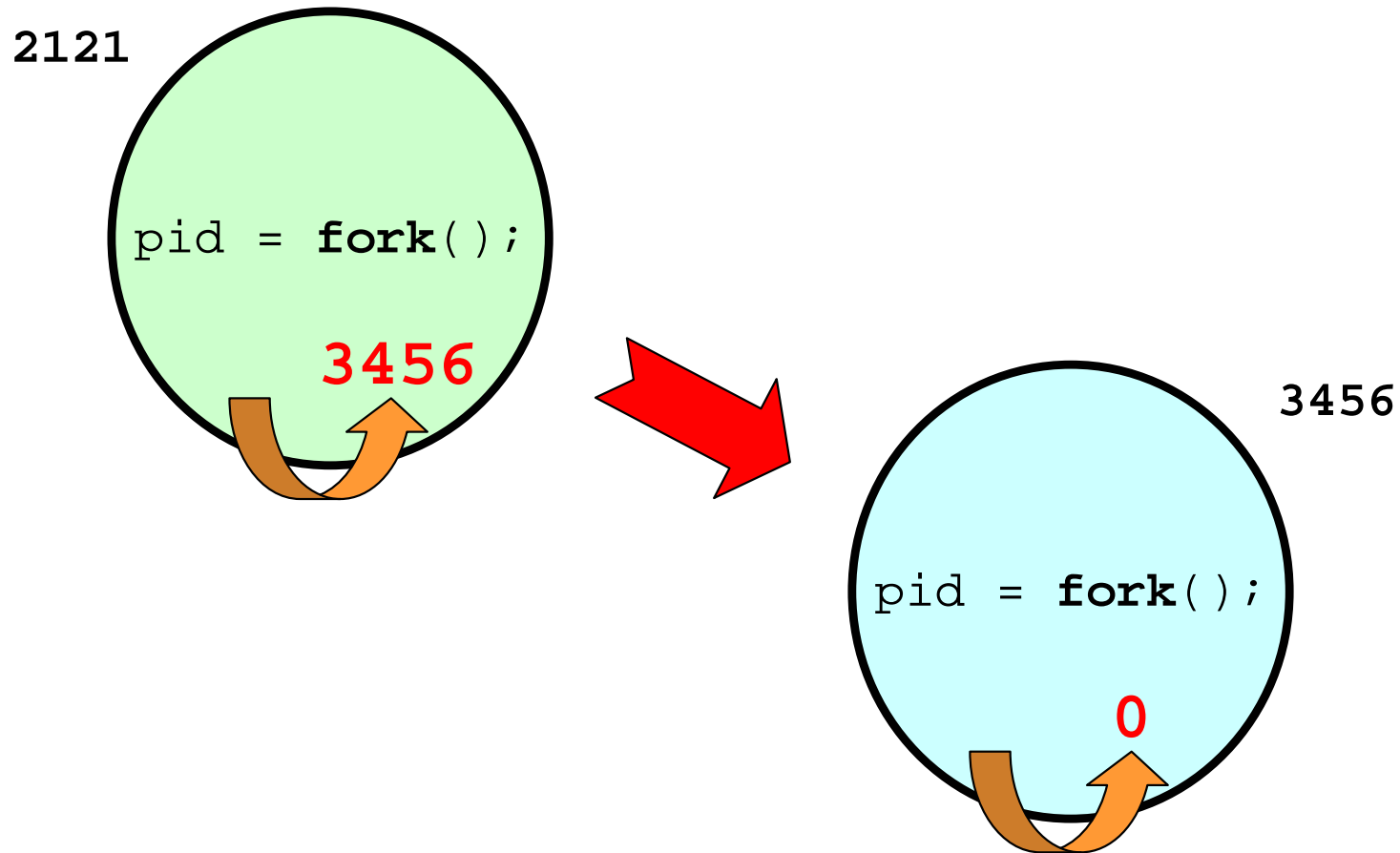
```
2121 gurasoa naiz, 3456 umearena  
3456 umea naiz, 2121 gurasoarena  
Bukatzera noa 2121  
Bukatzera noa 3456
```

```
3456 umea naiz, 2121 gurasoarena  
2121 gurasoa naiz, 3456 umearena  
Bukatzera noa 2121  
Bukatzera noa 3456
```

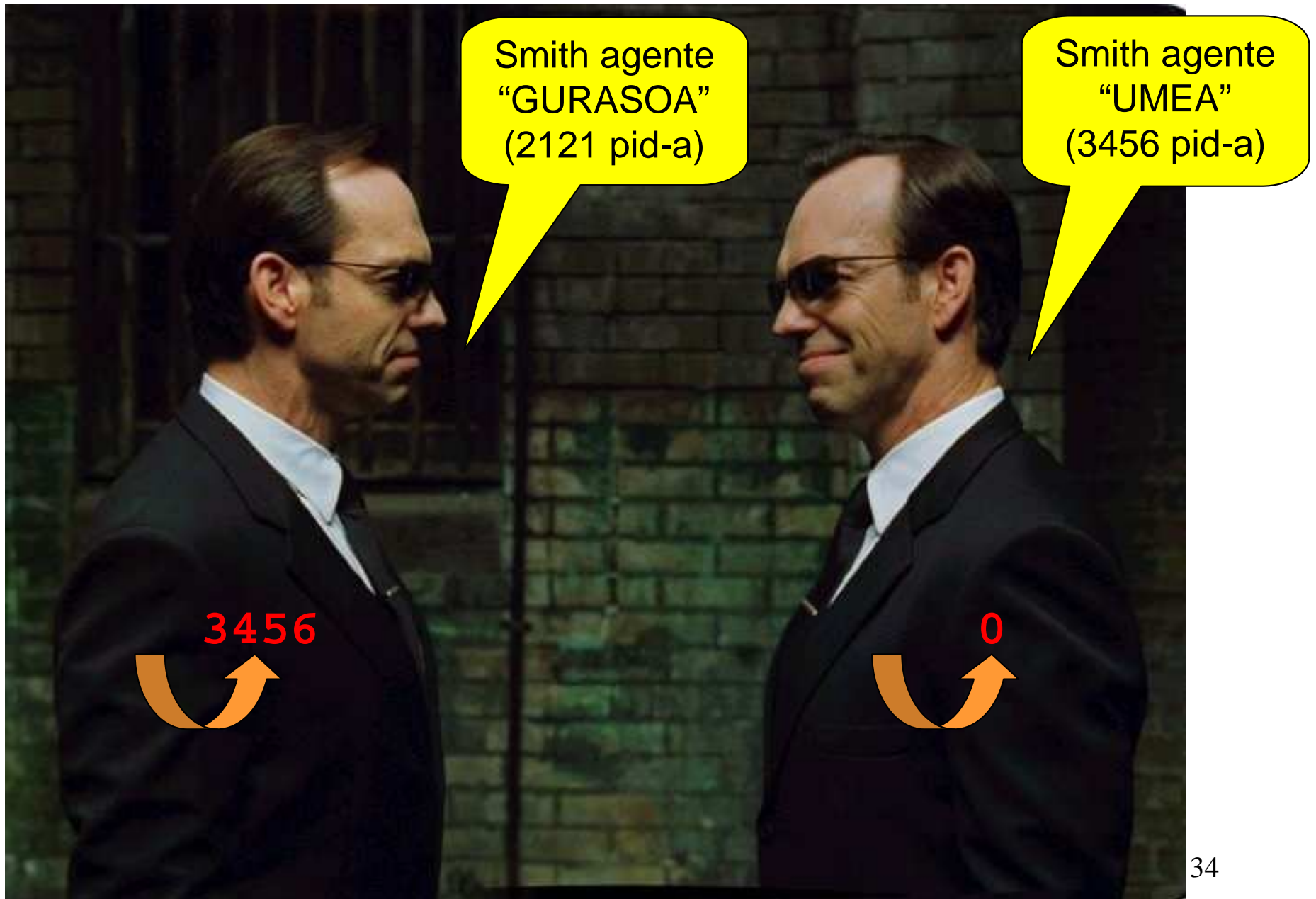
```
2121 gurasoa naiz, 3456 umearena  
Bukatzera noa 2121  
3456 umea naiz, 1 gurasoarena  
Bukatzera noa 3456
```

... "1" da bere gurasoa
bukatu duelako, eta
Unix-en umezurtz
prozesuak "init"-ek
adoptatzen dituelako

Prozesuen sorrera



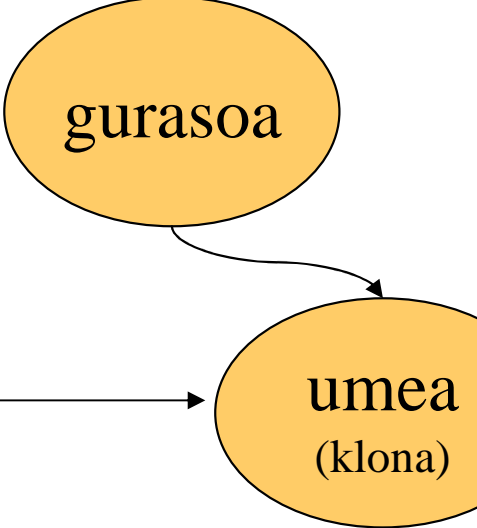
Prozesuen sorrera



fork sistema-deiaren proba

```
main()
{
    int pid;

    printf("fork deiaren proba\n");
    pid = fork();
    printf("kontrola itzuli zaio ");
    if (pid == 0) printf("umeari\n");
    else printf("gurasoari, %d izanik ume berriaren pid-a\n", pid);
}
```



gurasoaren irteera

fork deiaren proba

kontrola itzuli zaio gurasoari, 999 izanik ume berriaren pid-a

umearen irteera

kontrola itzuli zaio **umeari**

•gurasoa

•**umea** 35

•gurasoa + umea

fork sistema-deiaren proba

Irteeraren kasu erreal posible batzuk

fork deiaren proba

kontrola itzuli zaio gurasoari, 999 izanik umearen pid-a

kontrola itzuli zaio umeari

fork deiaren proba

kontrola itzuli zaio umeari

kontrola itzuli zaio gurasoari, 999 izanik umearen pid-a

fork deiaren proba

kontrola itzuli zaio kontrola itzuli zaio umeari

gurasoari, 999 izanik umearen pid-a

gurasoa

umea

gurasoa + umea



Prozesuen sorrera

`int exec??(...);`

- Prozesua exekutatzen ari den programa **aldatzen** du
 1. Prozesuaren edukia husten du, testuingurua mantenduz
 2. Programa berria kargatzen du
- `exec??` deia ongi burutzen bada, ez du ezer bueltatzen... **programa aldatu egin delako!**
- Erroreren bat gertatzen bada, **-1** bueltatzen du

Prozesuen sorrera

exec funtzio familia:


```
int execl(char *path, char *arg0, ..., NULL);
```

```
int execv(char *path, char *arg[]);
```

```
int execlp(char *path, char *arg0, ..., char *envp[]);
```

```
int execve(char *path, char *arg[], char *envp[]);
```

```
 int execlp(char *file, char *arg0, ..., NULL);
```

```
 int execvp(char *file, char *arg[]);
```

Prozesuen sorrera

exec??(...): programa berriak eredu hau jarraitzen du:

```
int main(int argc, char *argv[]);
```

- **argc**: argumentu kopurua (programaren izena barne)
- **argv**: argumentu bektorea (**argv[0]**: programaren izena), **NULL**-ekin bukatu behar da
- **path**: programa berria gordetzen duen fitxategi exekutagarriara apuntatzen du (*path* osoa)
- **file**: fitxategi exekutagarriaren izena (*path* erlatiboa)
- **envp**: ingurunea, kanpoko **environ** aldagaitik lortua

Prozesuen sorrera

```
main()    /* adibide3 */
{
    int pid;

    pid = fork();
    switch (pid){
        case -1:    /* errorea */
            exit(-1);
        case 0:    /* umea */
            execlp("ls", "ls", "-al", NULL);
            erre("execlp");
            break;
        default:    /* gurasoa */
            printf("%d gurasoa, %d umearena\n", getpid(), pid);
    }
}
```

Prozesuen sorrera

```
acpt00% adibide3
```

```
2121 gurasoa, 3456 umearena
```

```
total 24
```

```
drwxr-xr-x    4 root      root          4096 feb  3 09:32 .  
drwxr-xr-x  671 root      root        12288 mar 29 16:05 ..  
drwxr-xr-x  254 root      root          4096 feb  5 15:43 acaf  
drwxr-xr-x    3 acaf0000 acaf          4096 feb  3 09:32 soft
```

```
acpt00%
```

Prozesuen sorrera

2121

```
main() /* adibide3 */
{
    int pid;

    pid = fork();
    switch (pid){
        case -1: /* errorea */
            exit(-1);
        case 0: /* umea */
            execlp("ls", "ls", "-al", NULL);
            erre("execlp");
            break;
        default: /* gurasoa */
            printf("%d gurasoa, %d umearena\n",
                getpid(), pid);
    }
}
```



3456

```
main() /* ls */
{

    /* programa berria */

}
```

Prozesuen sorrera

```
main(int argc, char *argv[])    /* adibide4 */
{
    int pid;

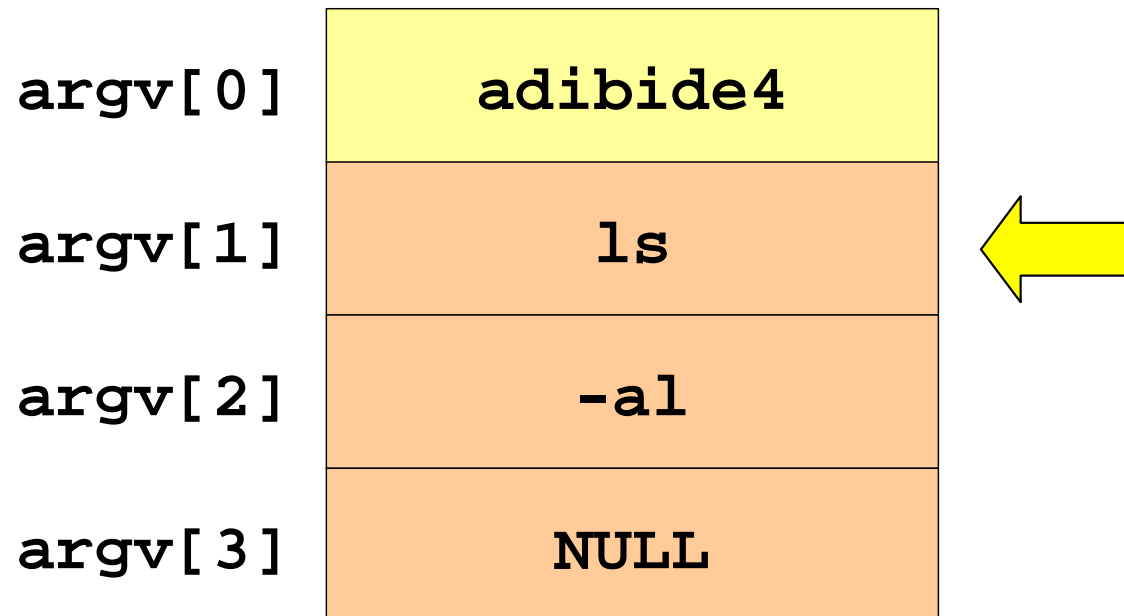
    pid = fork();
    switch (pid){
        case -1:    /* errorea */
            exit(-1);
        case 0:    /* umea */
            execvp(argv[1], &(argv[1]));
            erre("execvp");
            break;
        default:    /* gurasoa */
            printf("%d gurasoa, %d umearena\n", getpid(), pid);
    }
}
```

Prozesuen sorrera

```
acpt00% adibide4 ls -al
2121 gurasoa, 3456 umearena
total 24
drwxr-xr-x    4 root      root          4096 feb  3 09:32 .
drwxr-xr-x  671 root      root         12288 mar 29 16:05 ..
drwxr-xr-x  254 root      root          4096 feb  5 15:43 acaf
drwxr-xr-x    3 acaf0000 acaf          4096 feb  3 09:32 soft
acpt00%
```

Prozesuen sorrera

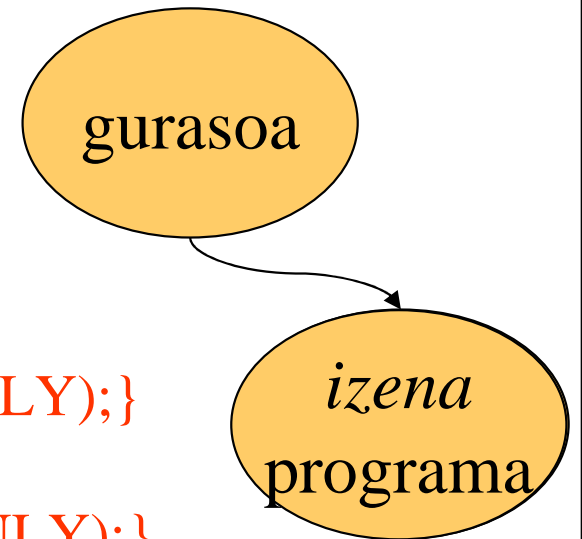
- Zergatik `execvp(argv[1], &(argv[1]));`?



fork eta *exec* deien konbinaketa

```
int exekutatu_programa(char *izena, char *kanal0, char *kanal1, char *kanal2,
                        char *argv[])
{
    int pid;

    pid = fork();
    if (pid == 0) { /* umea */
        if (kanal0 != NULL)
            {close(0); open(kanal0, O_RDONLY);}
        if (kanal1 != NULL)
            {close(1); open(kanal1, O_WRONLY);}
        if (kanal2 != NULL)
            {close(2); open(kanal2, O_WRONLY);}
        execvp(izena, argv);
        errore("exec");
    }
    else return(pid);
}
```



gurasoa
umea
gurasoa + umea

Prozesuen bukaera

void **exit**(int egoera);

- Prozesuaren bukaera “kontrolatua”
- Unix-ek bukaera-kodea (**egoera**) gorde egiten du gurasoak **wait()** exekutatu arte

int **wait**(int *egoera);

- Deitzailea bere umeren baten bukaera arte geldiarazten du
- Umerik ez badu, -1 bueltatzen du, deitzailea blokeatu gabe
- Bukatzen duen ume-prozesuaren identifikadorea bueltatzen du
- **egoera** umeak bueltatutako bukaera-kodea da

int **kill**(int pid, int SIGKILL);

- **SIGKILL** seinalea **pid** prozesuari bidaltzen dio, bukaraziz

Denboraren kontrola

unsigned int alarm(unsigned int seg);

- Unix-ek deitzaileari **SIGALARM** seinalea bidaltzen dio **seg** segundo igaro ondoren

void pause();

- Unix-ek deitzailea seinale bat jaso arte blokeatzen du
- Edozein seinale jasotzerakoan esnatzen da prozesua

int signal(int seinale, void funtz());

- **seinale** seinaleari **funtz** funtzioa lotzen dio

Denboraren kontrola

unsigned long `time(0)`;

- Unix-eko denbora bueltatzen du (1970eko urtarrilaren 1etik igarotako segundo kopurua)

char *`ctime(unsigned long t_unix)`;

- string batean denbora itzultzen duen liburutegi-errutina
- Adibidez: **Thu Apr 1 21:01.04 2004**

Denboraren kontrola

(*denbora.h*)

```
void itxaron_denbora(unsigned segundo_kopurua)
{
    signal(SIGALARM, fnulua);
    alarm(segundo_kopurua);
    pause();
}

void fnulua()
{
    return;
}
```

Beste sistema-deiak multiprogramaziorako

- **int nice(int balioa);**
 - Prozesuaren lehentasuna aldatzeko balio du
 - Lehentasuna jaitea beti da posible, igotzea aldiz soilik *root*-ek egin dezake

Adibide-programa (gurasoa.c)

Guraso-prozesu batek programa baten **exekuzioa abiarazten** du, honen **izena eta argumentuak bigarren parametrotik** pasatzen zaiolarik. **Lehen argumentuak programaren exekuzio-denbora maximoa** adieraziko du; denbora hori pasa eta programak bukatu ez badu, guraso prozesuak **umearen exekuzioa amaiaraziko** du. Guraso-prozesuak **bere identifikadorea eta abiarazten duen programarena ere idazten** ditu. Programa berriak amaitzean, gurasoak bere iraupena idatzi eta itzulera-kodea itzultzen du, edo -1 balioa programa amaiarazi badu.

Beharrezkoa da, oraingoz, **erloju-prozesu** bat abiaraztea programaren exekuzio-denbora kontrolatzeko, ezin baita zuzenean *itxaron_denbora* **funtzioa** erabili, honek, sinkronoa izatean, guraso-prozesua blokeatuta utziko bailuke umea bukatu ondoren ere.

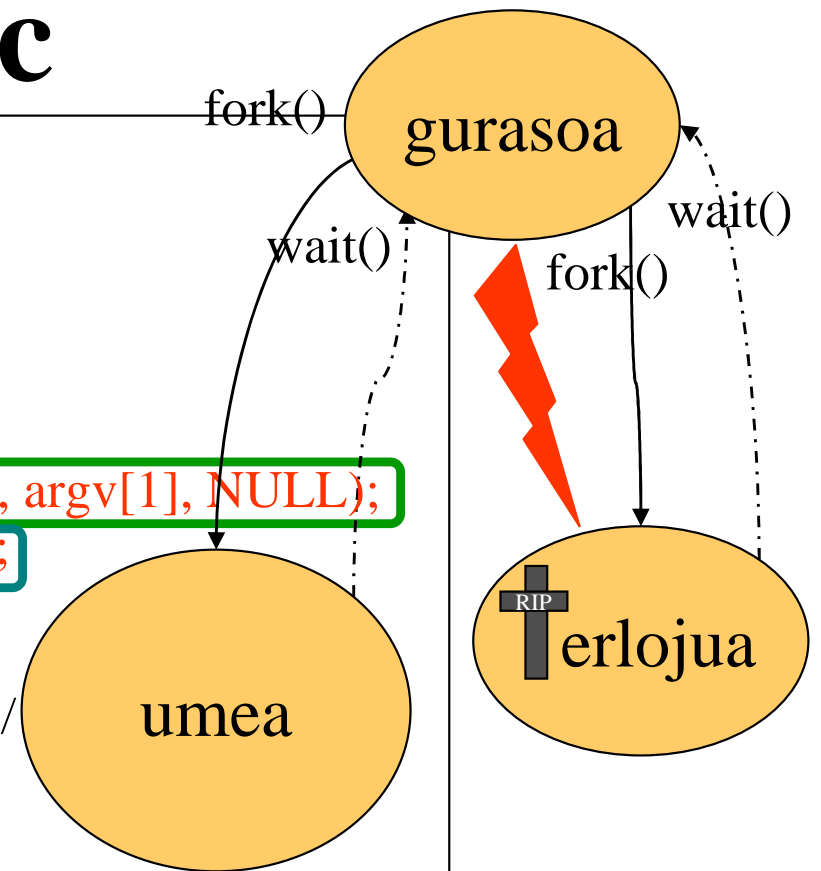
Exekuzio-adibidea:

> **gurasoa 60 nire_programa**

gurasoa.c

```
main(int argc, char *argv[])
{
    int umeid, erlojuid, id, t1, lag, lag2;

    id = getpid();
    printf("—guraso-prozesua: %d\n", id);
    if ((erlojuid = fork()) == 0) execlp("erlojua", "erlojua", argv[1], NULL);
    if ((umeid = fork()) == 0) execvp(argv[2], &(argv[2]));
    printf("—ume-prozesua: %d\n", umeid);
    t1 = time(0);
    if ((id = wait(&lag)) == umeid) { /* amaiera arrunta */
        kill(erlojuid, SIGKILL);
        wait(&lag2);
    }
    else { /* akatu ume-prozesua */
        kill(umeid, SIGKILL);
        wait(&lag2);
        printf("—denbora pasata\n");
        lag = -1;
    }
    t1 = time(0) - t1;
    printf("—ume-prozesuaren denbora: %d\n", t1);
    exit(lag);
}
```

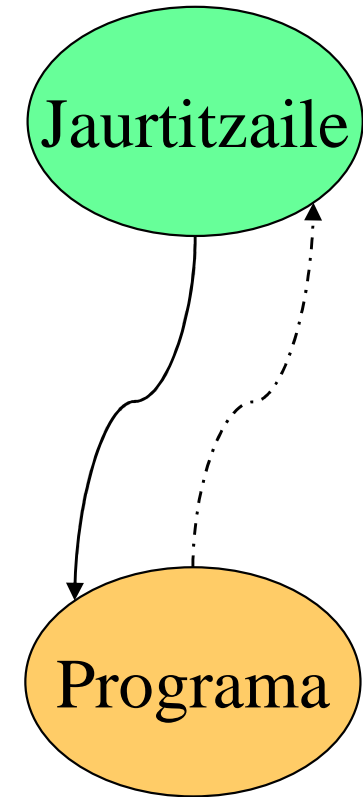


```
/* erlojua.c */
#include <stdlib.h>
#include "denbora.h"

main(int argc, char *argv[])
{
    itxaron_denbora(atoi(argv[1]));
}
```

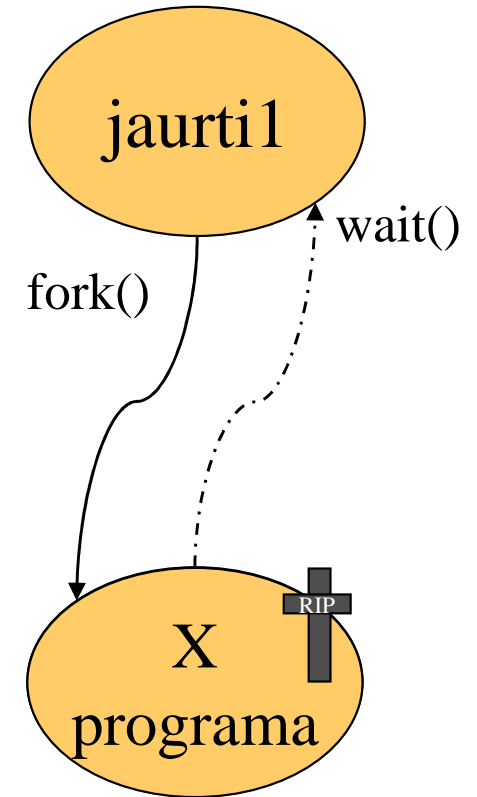
Komando-Interpretatzaile simple bat: Jaurtitzailerak

```
main(int argc, char *argv[])
{
    ...
    while (! Bukaera)
    {
        Idatzi_Prompt();
        Irakurri_Komando_Lerroa();
        Aztertu_Komando_Lerroa();
        Exekutatu_Programa();
        Itxaron_Umea();
    }
}
```



jaurti.c

```
/* jaurti.c */
.....
main(int argc, char *argv[]) /* jaurtizailea */
{
    ...
    write(1, "Jaurti1> ", 9);
    while ((n = read(0, buf, BUFSIZE)) > 0) {
        buf[n] = '\n'; n++;
        err = lortu_argumentuak(buf, n, args, MAXARGS);
        switch (pid = fork()) {
            case -1: errore("fork");
                break;
            case 0: /* ume-prozesua hemendik aurrera */
                execvp(args[0], args); /* komandoa jaurti */
                errore("exec");
                break;
            default:
                printf("%d (%s ..) prozesua sortua\n", pid, args[0]);
                if (wait(NULL) != pid) errore("wait");
                for (n = 0; n < BUFSIZE; n++) buf[n] = '\0';
                write(1, "Jaurti1> ", 9);
                break;
        }
    }
}
```

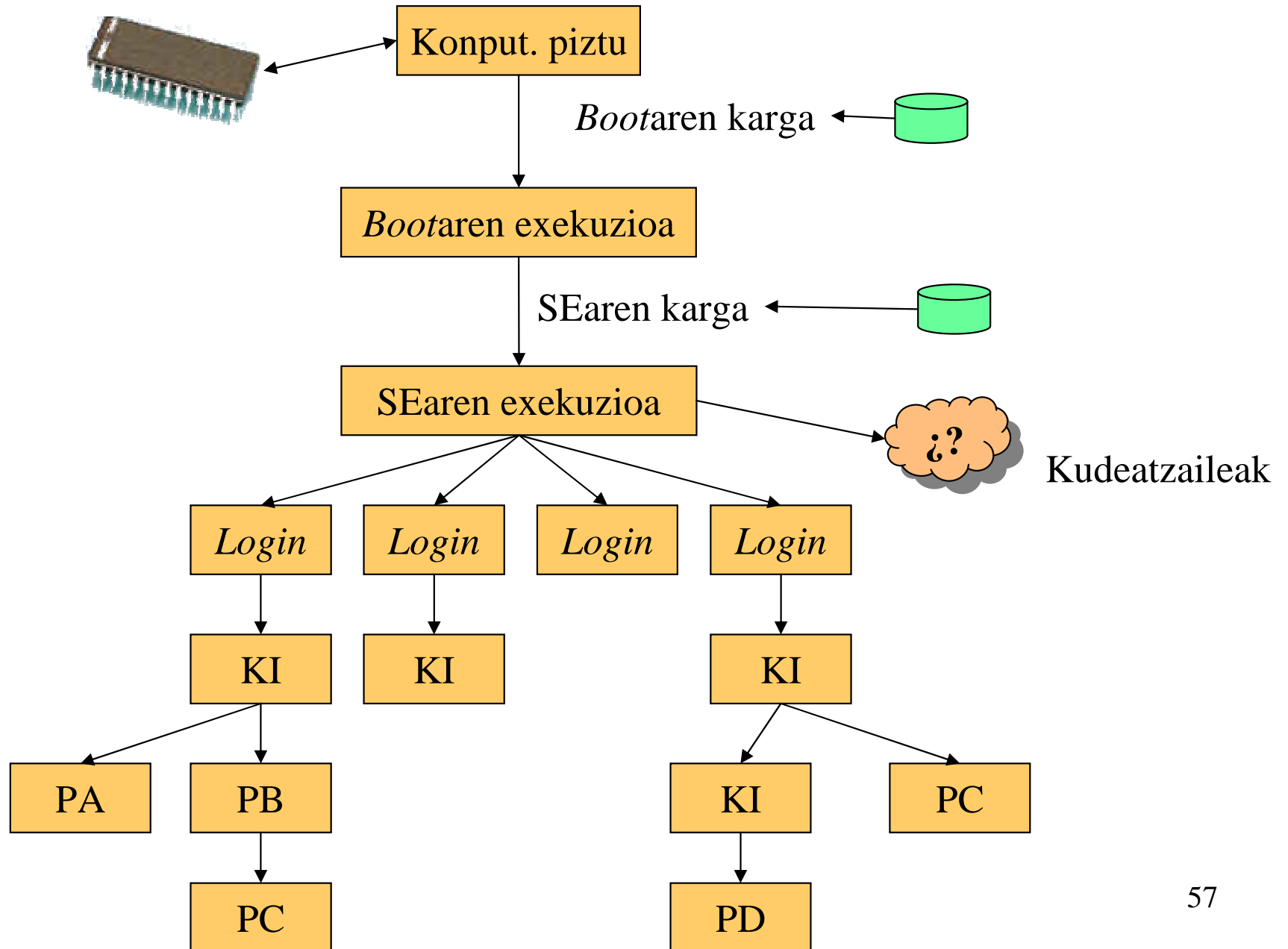


jaurti1.c (jarraipena)

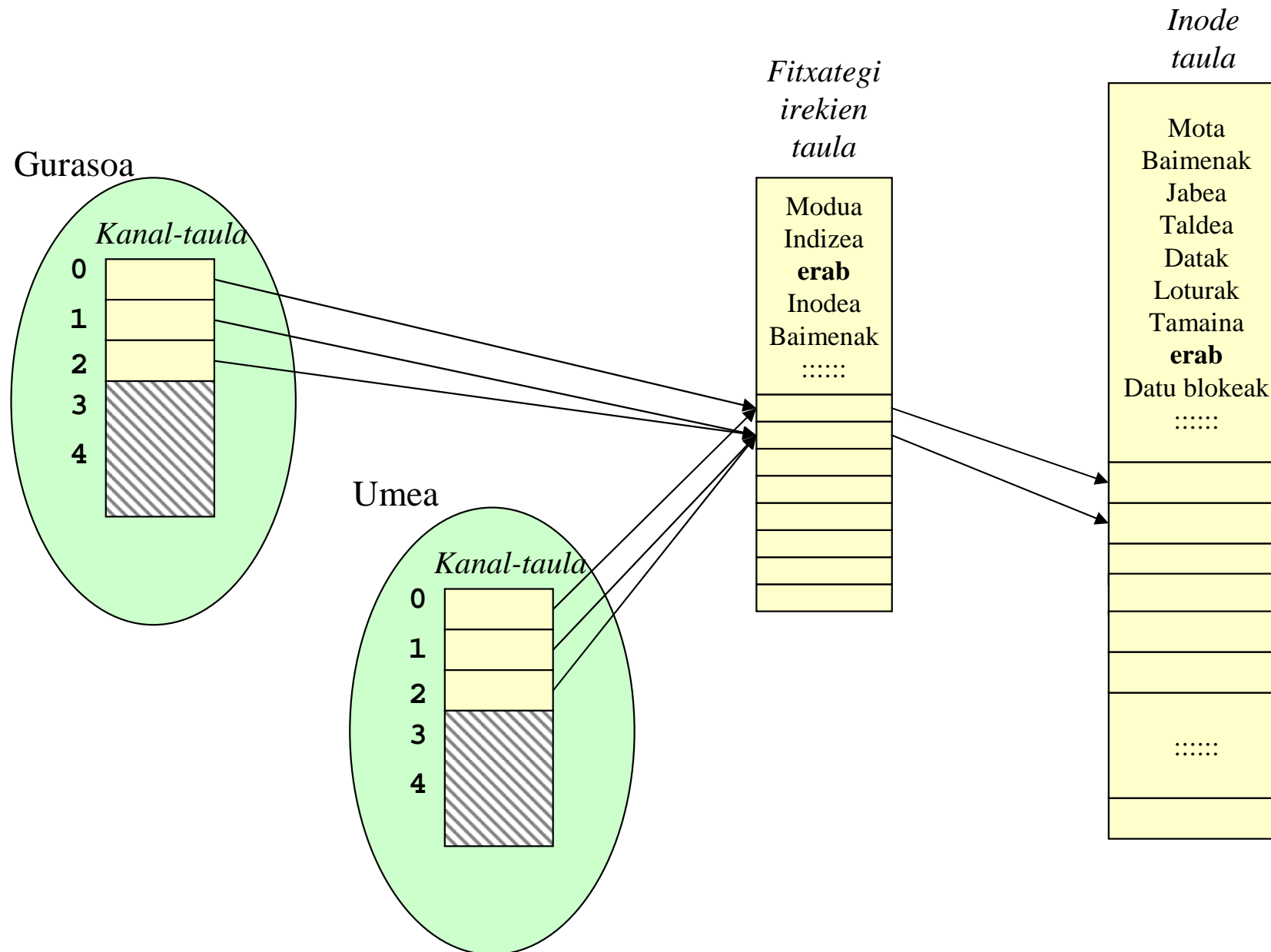
```
int lortu_argumentuak(char *buf, int n, char *args[], int m)
{
    int i, j;

    for (i = 0, j = 0; (i < n) && (j < m); j++) {
        /* zuriuneak pasa */
        while (((buf[i] == ' ') || (buf[i] == '\n')) && (i < n)) i++;
        if (i == n) break;
        args[j] = &buf[i];
        /* bilatu zuriune karakterea */
        while ((buf[i] != ' ') && (buf[i] != '\n')) i++;
        buf[i++] = '\0';
    }
    args[j] = NULL;
}
```

SE multiprogramatu baten funtzionamendua



Kanal-taularen herentzia *fork()* egiterakoan

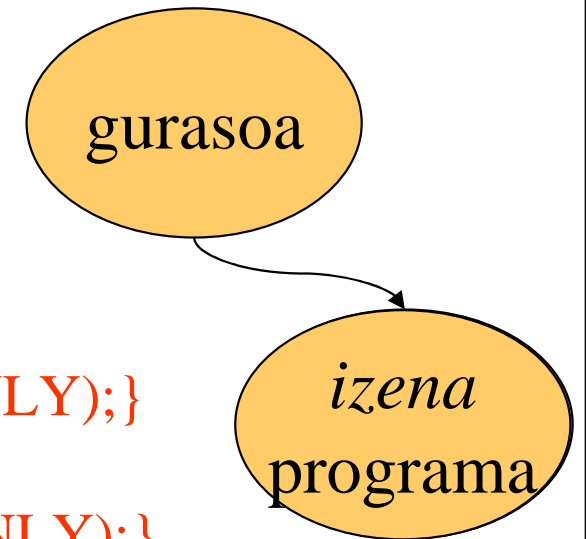


Sarrera/Irteeraren berbideraketa

(fork + exec)

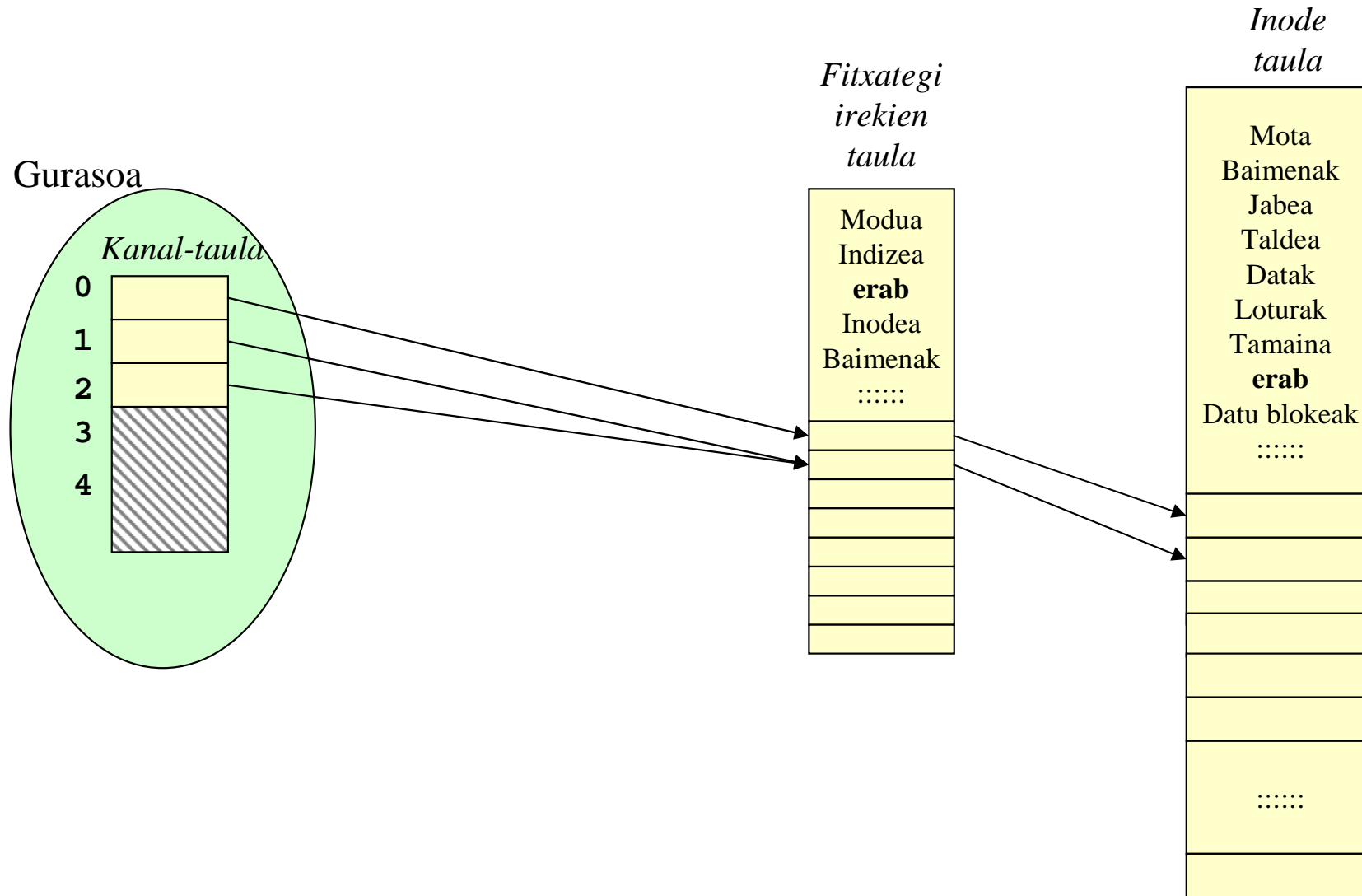
```
int exekutatu_programa(char *izena, char *kanal0, char *kanal1, char *kanal2,
                        char *argv[])
{
    int pid;

    pid = fork();
    if (pid == 0) { /* umea */
        if (kanal0 != NULL)
            {close(0); open(kanal0, O_RDONLY);}
        if (kanal1 != NULL)
            {close(1); open(kanal1, O_WRONLY);}
        if (kanal2 != NULL)
            {close(2); open(kanal2, O_WRONLY);}
        execvp(izena, argv);
        errore("exec");
    }
    else return(pid);
}
```



gurasoa
umea
gurasoa + umea

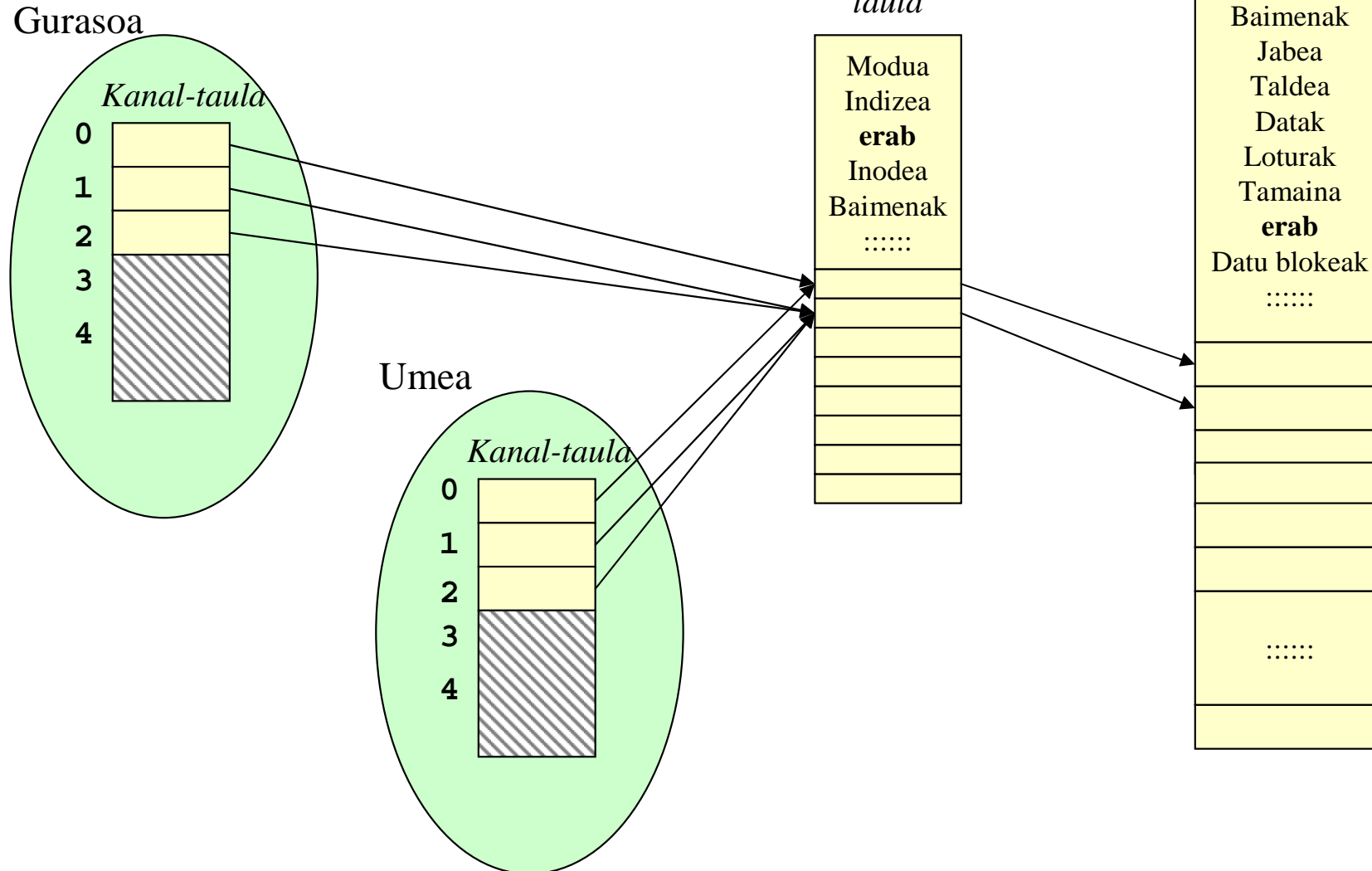
fork() + Sarrera/Itteera berbideraketa + *exec()*



fork() + Sarrera/Itxera berbideraketa + *exec()*

Umearen sorrera

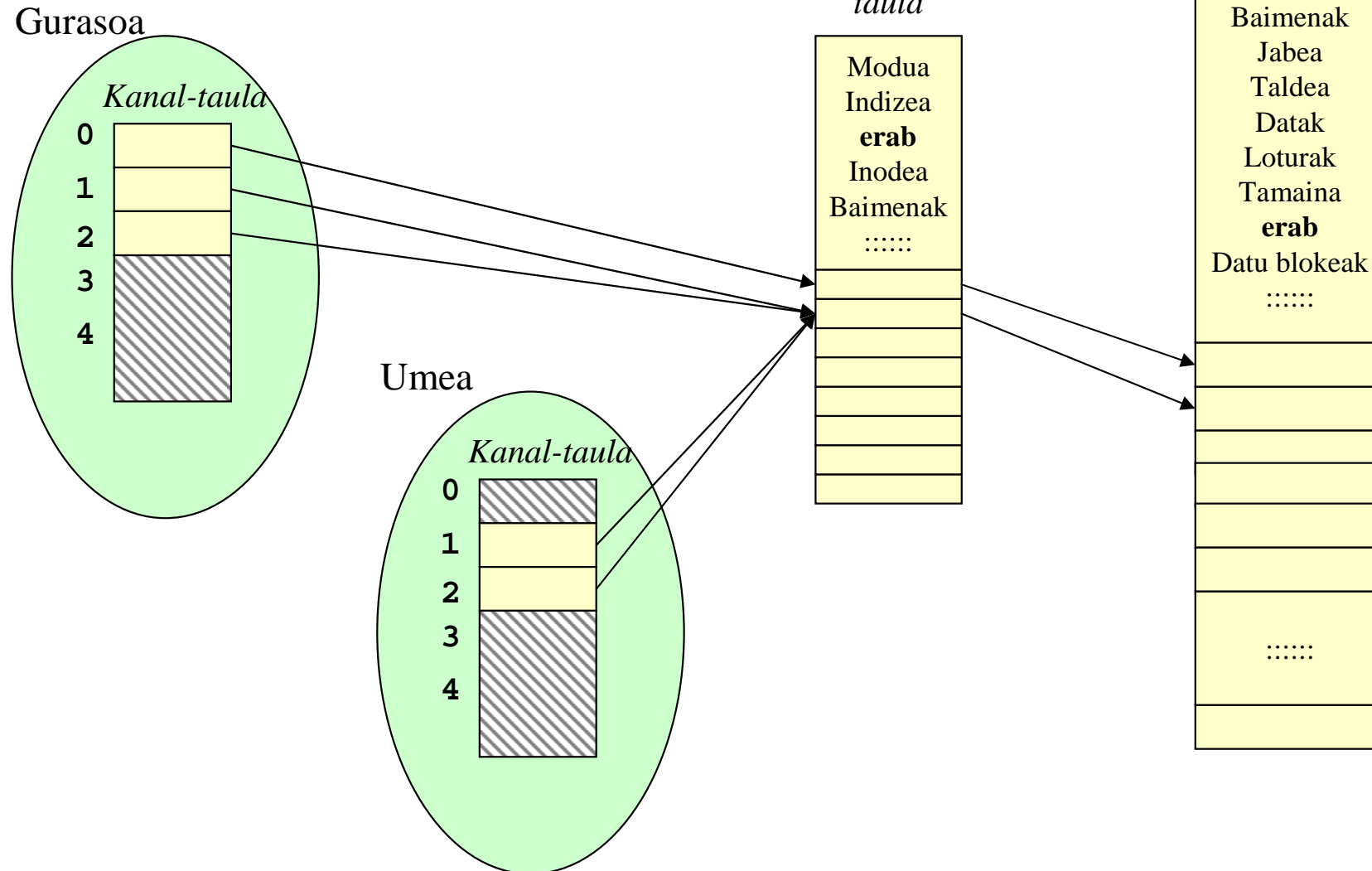
Gurasoa: *fork()*;



fork() + Sarrera/Irteera berbideraketa + *exec()*

Sarrera estandarraren berbideraketa

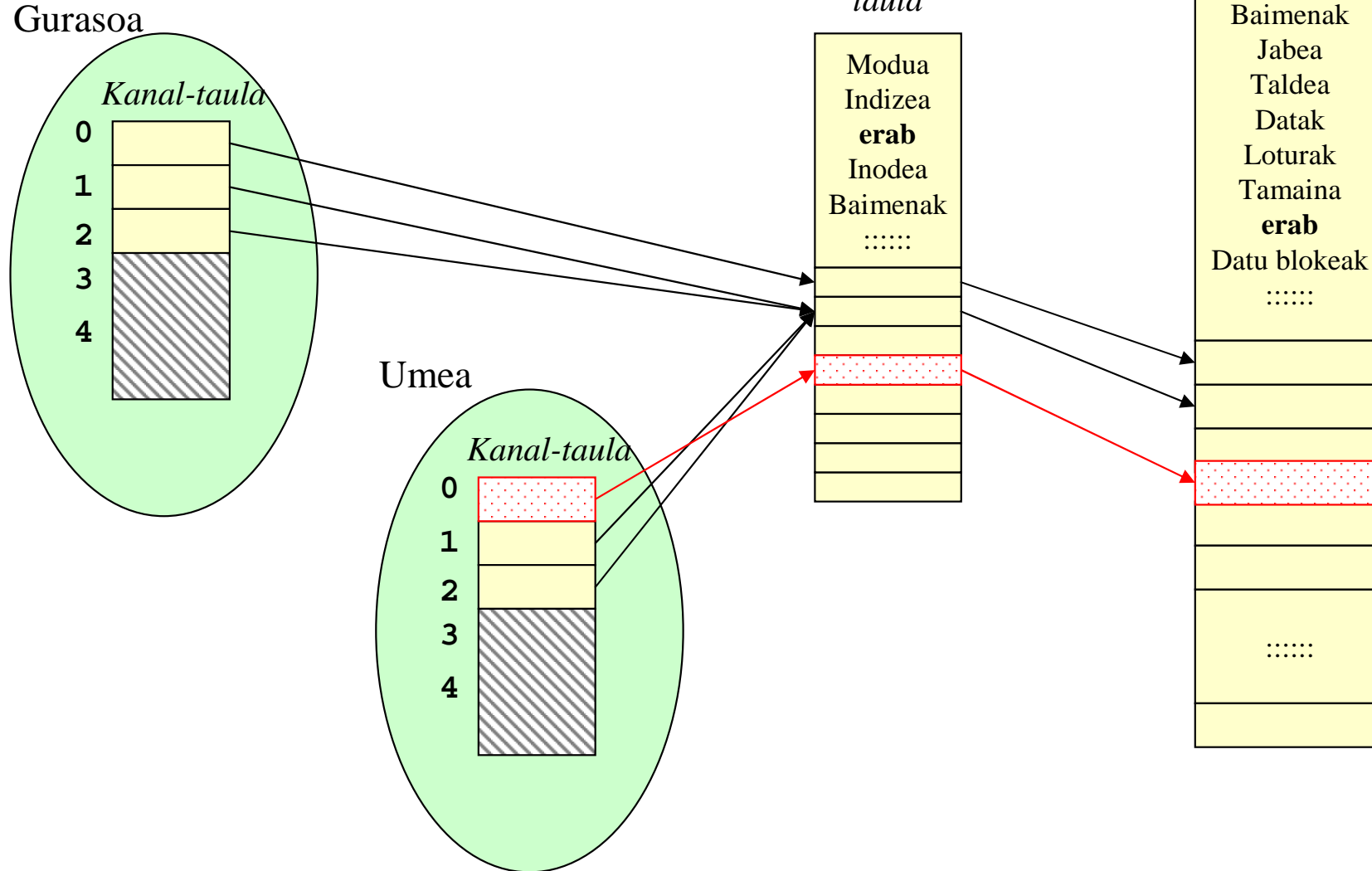
Umea: *close(0);*



fork() + Sarrera/Irteera berbideraketa + *exec()*

Sarrera estandarraren berbideraketa

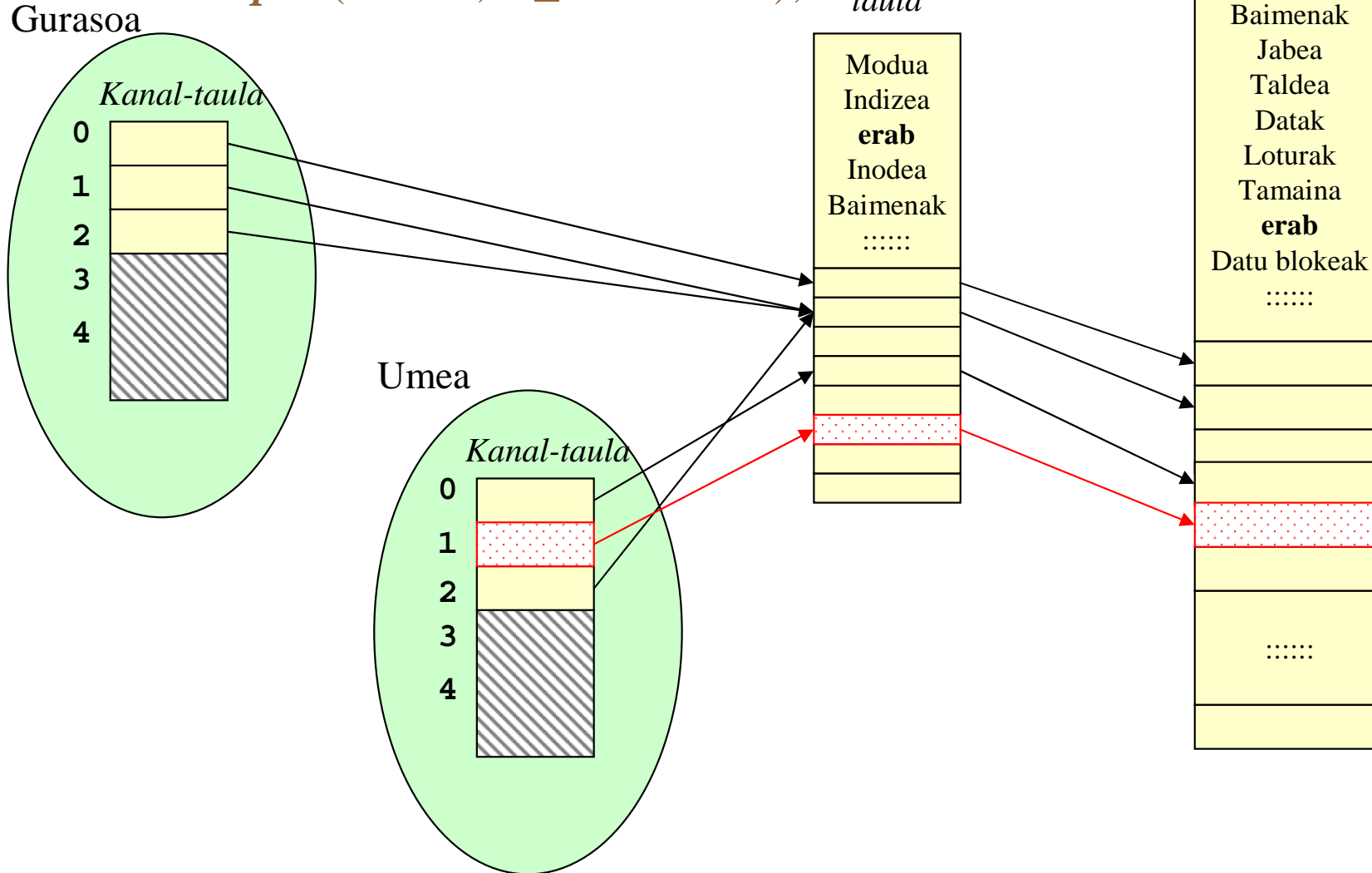
Umea: *open(kanal0, O_RDONLY);* *Fitxategi irekien taula*



fork() + Sarrera/Itzera berbideraketa + *exec()*

Itzera estandarren berbideraketa

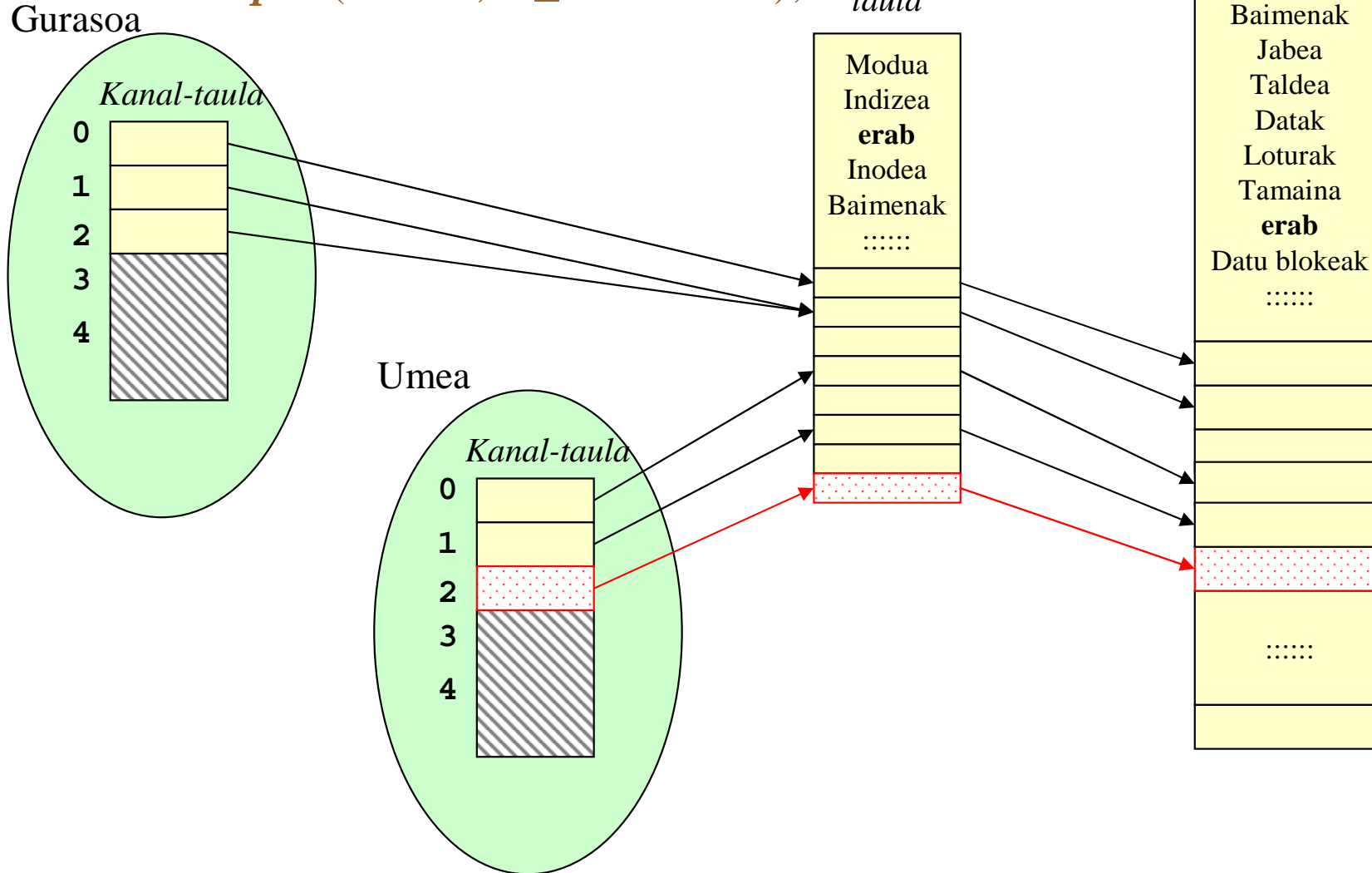
```
Umea: close(1);  
open(kanal1, O_WRONLY);
```



fork() + Sarrera/Irteera berbideraketa + *exec()*

Errore-irteeraren berbideraketa

```
Umea: close(2);  
open(kanal2, O_WRONLY);
```



fork() + Sarrera/Itteera berbideraketa + *exec()*

