

7. Gaia. Multiprogramazioa: *prozesu konkurrenteen arteko komunikazioa*

1. Motibazioa
2. Komunikazioa eta sinkronizazioa
3. Komunikazio eta sinkronizazio metodoak
4. Buzoien bidezko komunikazioa
5. Sistema-deiak eta adibideak
6. Bezero-Zerbitzari eredua eta adibideak

Motibazioa

Sistema Eragile multiprogramatua: prozesu bat baino gehiago era konkurrentean exekututzen

- Prozesu independenteak / ez independenteak:
- Prozesu independenteak:
 - Exekuzio determinista: prozesu baten exekuzioak, datu berdinekin, beti emaitza berdina ematen du
 - Prozesu baten exekuzioak ez du besteen exekuzioetan eraginik. Baliabideak erabiltzeko orduan (CPUa ezik), ez dago prozesuen arteko lehiarik edota koordinaziorik / sinkronizaziorik

Motibazioa

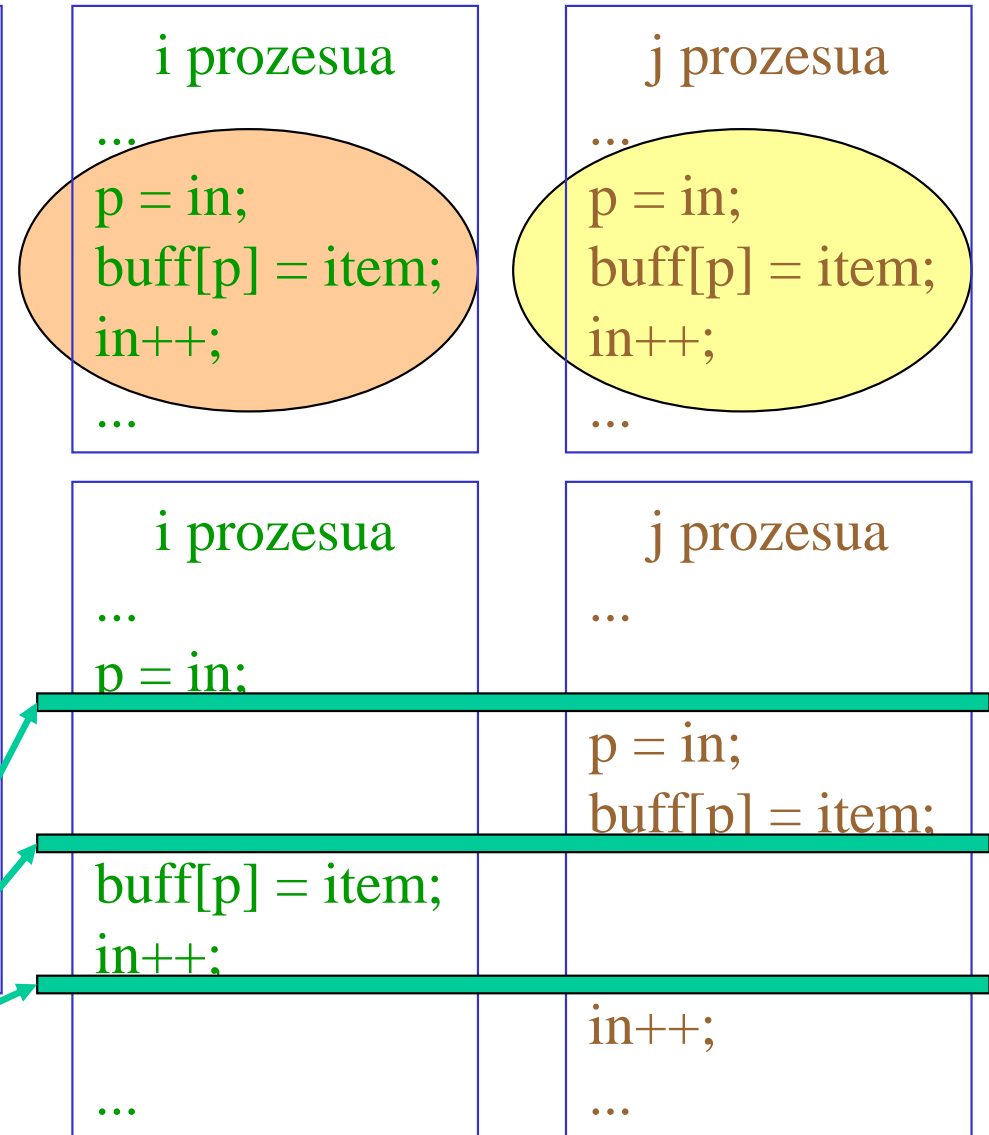
- **Prozesu ez independenteak:**
 - Exekuzio ez determinista: prozesu baten exekuzioak, datu berdinekin, ez du beti emaitza berdina emango
 - Azken emaitza prozesuen exekuzioen ordenaren menpe egongo da. Baliabideren bat erabiltzerakoan prozesuek kooperatu edota lehiatu egiten dute
 - *Ekoizle-Kontsumitzaile* ereduan, bi prozesuk tamaina finkoko buffer bat konpartitzen dute. Ekoizleak elementuak sortu eta bufferrean uzten ditu. Kontsumitzaileak aldiz, bufferretik hartzen ditu elementuak

Motibazioa

- Prozesu ez independenteak:
 - *Bezero-Zerbitzari* eredua
 - Adibidez, inprimagailuaren *spooling*-a. Prozesuek inprimatu nahi dutena karpeta baten uzten dute, eta beste prozesu batek (*daemon*) periodikoki konprobatzen du ea badauden fitxategirik inprimatzeko
 - Prozesuen arteko komunikaziorako eta sinkronizaziorako mekanismoak

Komunikazioa eta sinkronizazioa

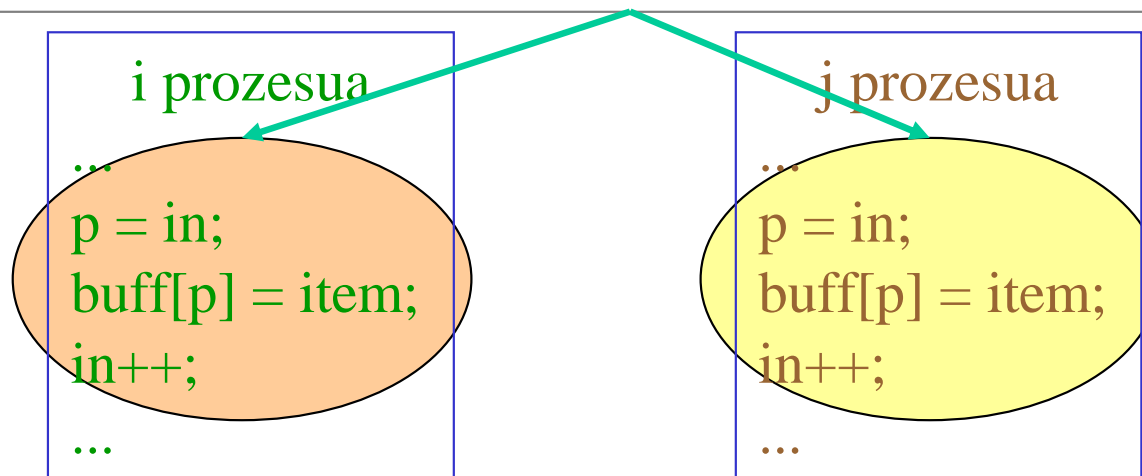
- **Komunikazioa:**
 - Prozesuen arteko informazio trukaketa
- **Sinkronizazioa:**
 - Implizitua (komunikazio mekanismoren baten)
 - Esplizitua. Sinkronizaziorako funtzio espezifikoak
- **Sekzio Kritikoaren arazoa:**
 - Lasterketa baldintzak



Komunikazioa eta sinkronizazioa

Sekzio Kritikoa:

Zenbait prozesuk konpartitutako baliabideak atzitzen duen kodea



- Arazoa: atzipen eskusiboa ziurtatzea sekzio kritikoetan
- “*P_i prozesu batek SK exekutatzen ari denean, beste edozein prozesu ezin izango da bertan sartu, P_i prozesua atera arte*”

Komunikazioa eta sinkronizazioa

- Atzipen eksklusiboa sekzio kritikoetan
- $\{P_0, P_1, \dots, P_{n-1}\}$ n prozesu, R baliabidea konpartitzen dute. Bedi SK baliabidea atzitzeko prozesu bakoitzak exekutatzeko duen kode zatia. Honakoa bete behar da:
 - *“ P_i prozesu batek SK exekutatzeko ari denean, beste edozein prozesu ezin izango da bertan sartu, P_i prozesua atera arte”*

Sekzio kritikoa atzitzeko protokoloa:

```
...  
Sartu_SK(); /* SKan ez bada inor, jarraitu; itxaron bestela */  
...  
Irten_SK();  
...
```

Komunikazioa eta sinkronizazioa

SKen atzipen eksklusiborako baldintzak:

1) **Elkarrekiko esklusioa**

Prozesu bat baino gehiago ezin da SKan egon aldi berean

2) **Elkar-blokeaketarik ez**

SKaren kanpo geldituriko prozesu batek ezin du galarazi beste prozesu bat SKan sar dadin

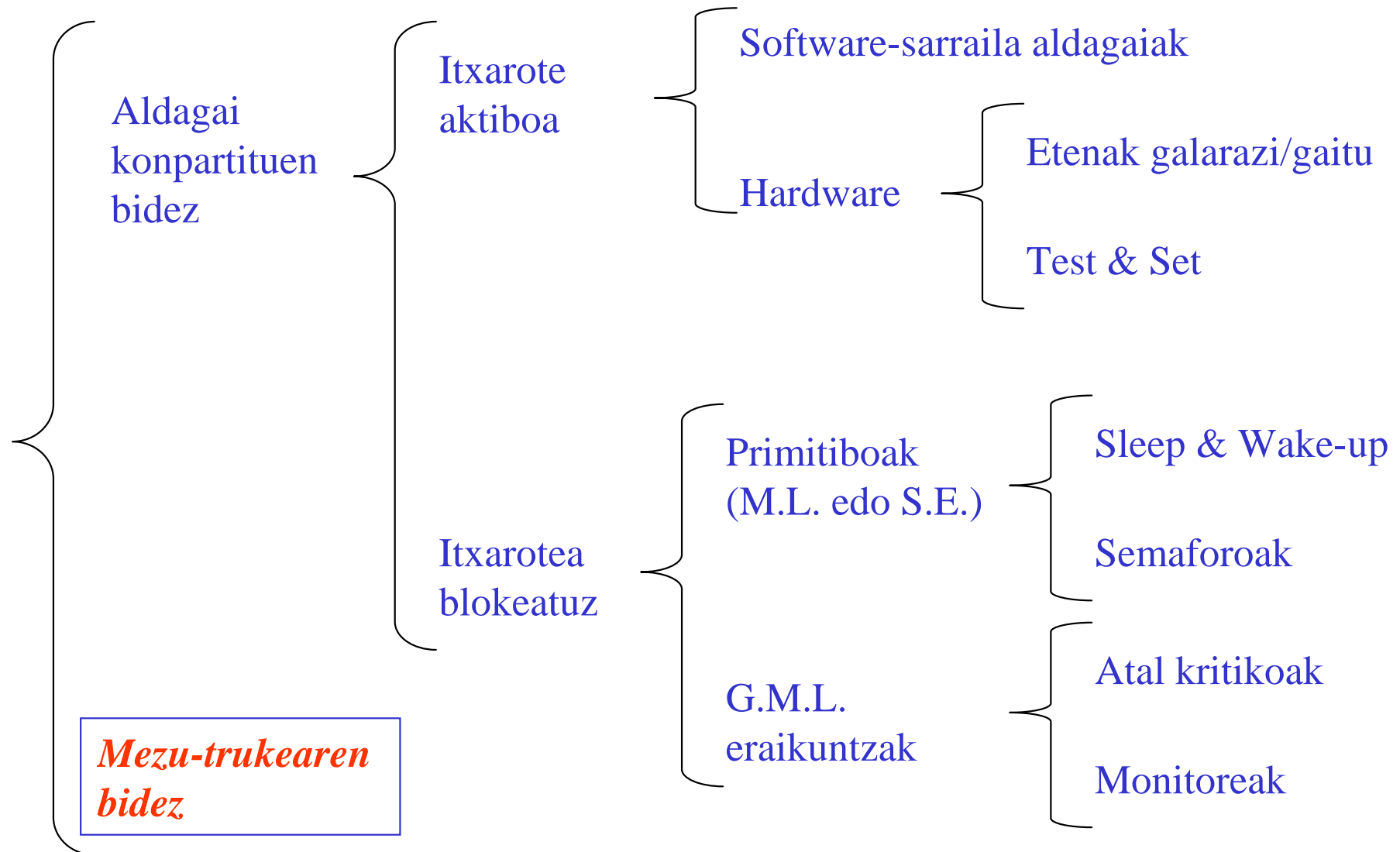
3) **Itxarote mugatua ("*bounded waiting*")**

Prozesu bat ezin da etengabe SKan sartzeko zain egon

4) **Hardwareekiko independentzia**

Ezin da prozesadore kopuru edo prozesuen abiadurari buruzko suposaketarik egin. Horretaz dakigun bakarra zera da: Makina-Lengoiako aginduak *zatiezinak* edo *atomikoak* dira eta sekuentzialki exekututzen dira

Komunikazio eta sinkronizazio metodoak



Buzoiak (FIFO ilarak)

Izendun buzoiak:

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
int mknod(char *path, int modua, int disp);
```

```
    fd = mknod(str, S_IFIFO|0666, 0);
```

```
int mkfifo(char *path, int modua);
```

```
    fd = mkfifo(str, 0666);
```

Buzoiak (FIFO ilarak) - jarraipena

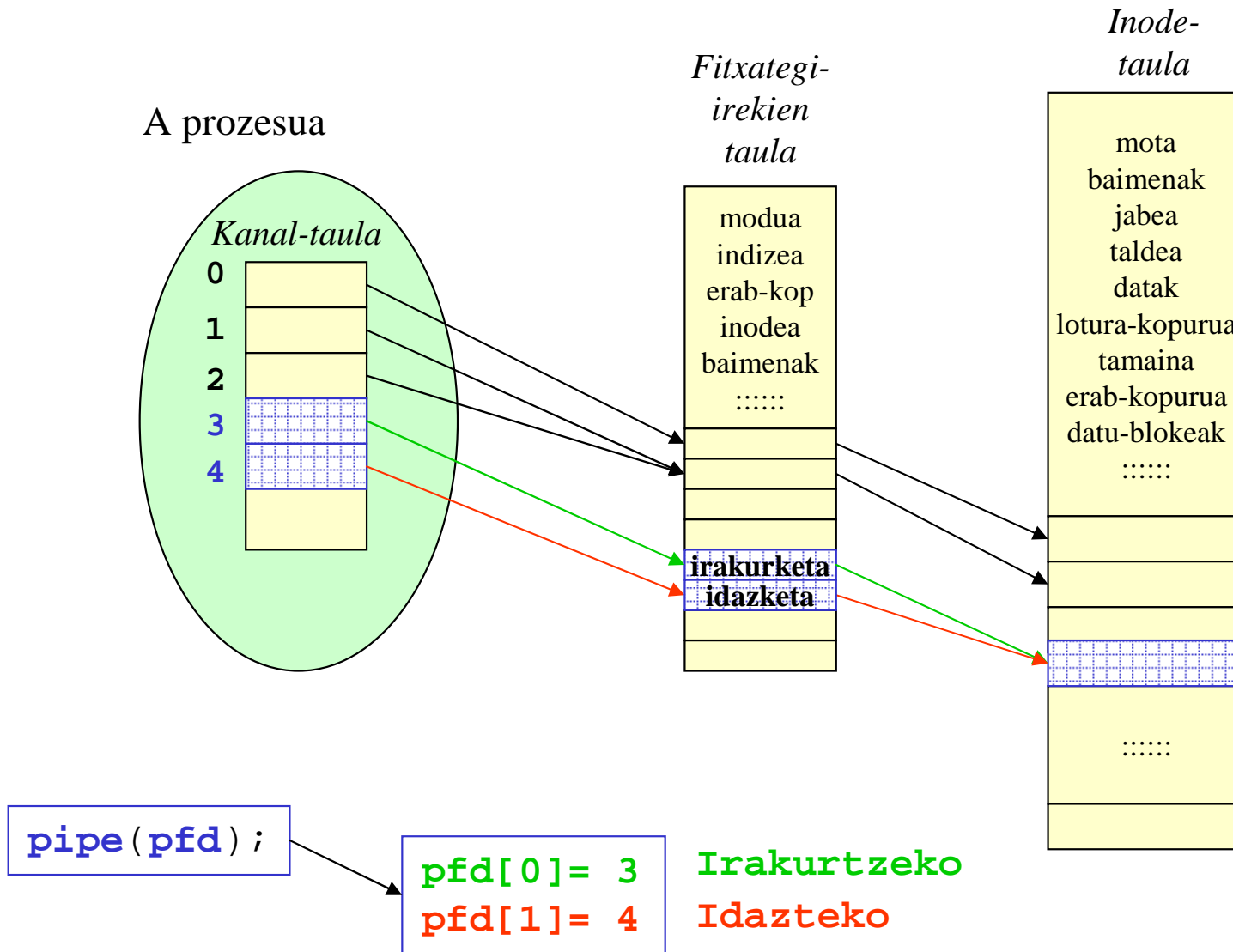
Izenik gabeko buzoiak (pipeak):

```
int pfd[2];  
  
int pipe(int *pfd);  
  
    int pfd[2];  
  
    pipe(pfd);  
    ...
```

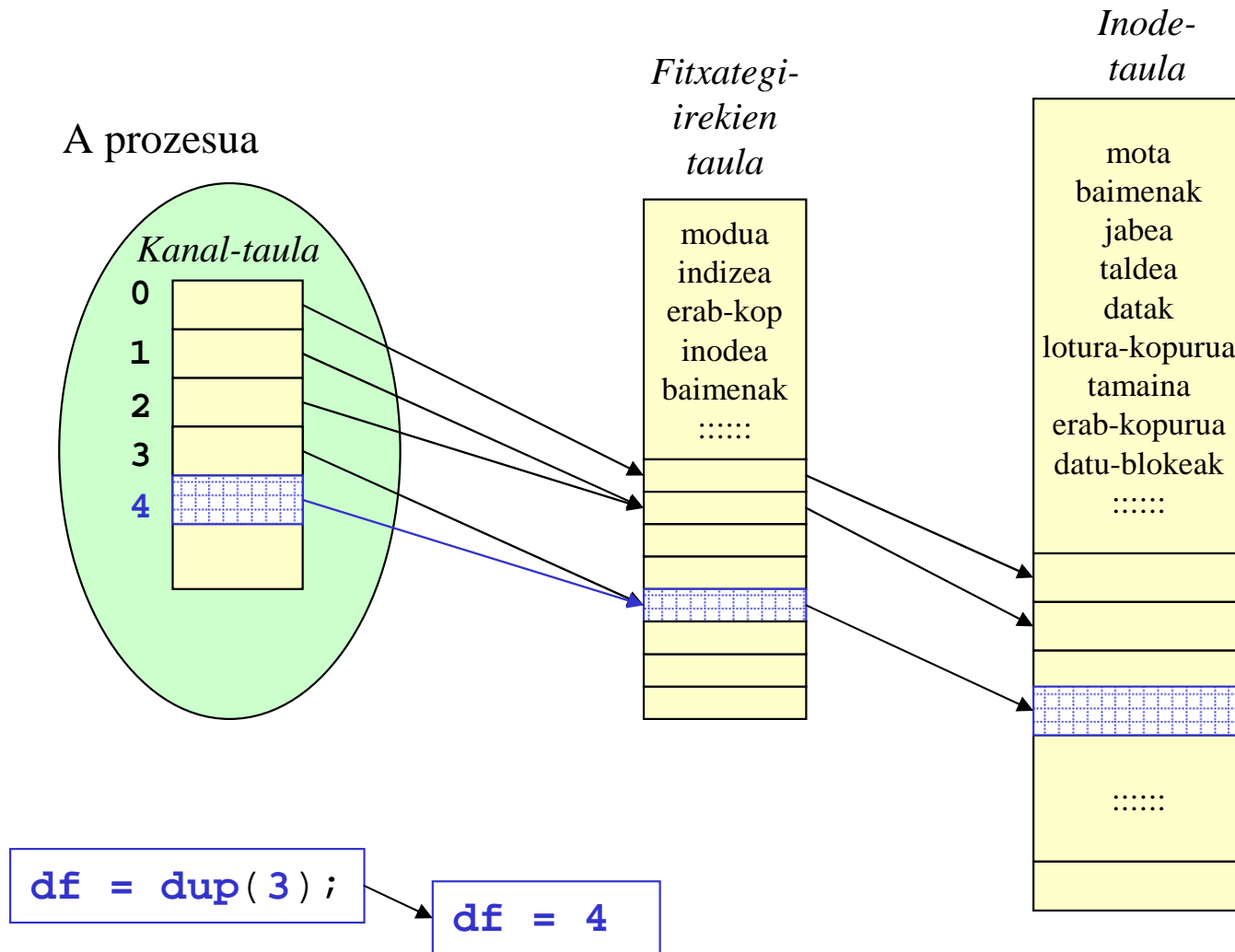
Besteak:

```
int dup(int fd);
```

pipe sistema-deia



dup sistema-deia



Izendun buzoien adibidea (*mknod*)

```
main() /* who | wc */
```

```
{
```

```
    char str[256];
```

```
    sprintf(str, "buzoiaren_izena");
```

```
    if (mknod(str, S_IFIFO|0666, 0) == -1)
```

```
        erre("mknod");
```

```
    switch (fork())
```

```
    {
```

```
        case -1: erre("fork");
```

```
        case 0: /* 1. umea: who */
```

```
            if (close(1) == -1) erre("close");
```

```
            if (open(str, O_WRONLY) != 1)
```

```
                erre("open");
```

```
            execlp("who", "who", NULL);
```

```
            erre("execlp");
```

```
    }
```

```
    switch (fork())
```

```
    {
```

```
        case -1: erre("fork");
```

```
        case 0: /* 2. umea: wc */
```

```
            if (close(0) == -1) erre("close");
```

```
            if (open(str, O_RDONLY) != 0)
```

```
                erre("open");
```

```
            execlp("wc", "wc", NULL);
```

```
            erre("execlp");
```

```
    }
```

```
    while (wait(NULL) != -1);
```

```
    unlink(str);
```

```
}
```

Izenik gabeko buzoien adibidea (*pipe* + *dup*)

```
main() /* who | wc */
```

```
{
```

```
    int pfd[2];
```

```
    if (pipe(pfd) == -1) errore("pipe");
```

```
    switch (fork())
```

```
    {
```

```
        case -1: errore("fork");
```

```
        case 0: /* 1. umea: who */
```

```
            if (close(1) == -1) errore("close");
```

```
            if (dup(pfd[1]) != 1) errore("dup");
```

```
            if ((close(pfd[0]) == -1) ||  
                (close(pfd[1]) == -1)) errore("close");
```

```
            execlp("who", "who", NULL);
```

```
            errore("execlp");
```

```
    }
```

```
    switch (fork())
```

```
    {
```

```
        case -1: errore("fork");
```

```
        case 0: /* 2. umea: wc */
```

```
            if (close(0) == -1) errore("close");
```

```
            if (dup(pfd[0]) != 0) errore("dup");
```

```
            if ((close(pfd[0]) == -1) ||  
                (close(pfd[1]) == -1)) errore("close");
```

```
            execlp("wc", "wc", NULL);
```

```
            errore("execlp");
```

```
    }
```

```
    if ((close(pfd[0]) == -1) || (close(pfd[1]) == -1))  
        errore("close");
```

```
    while (wait(NULL) != -1) ;
```

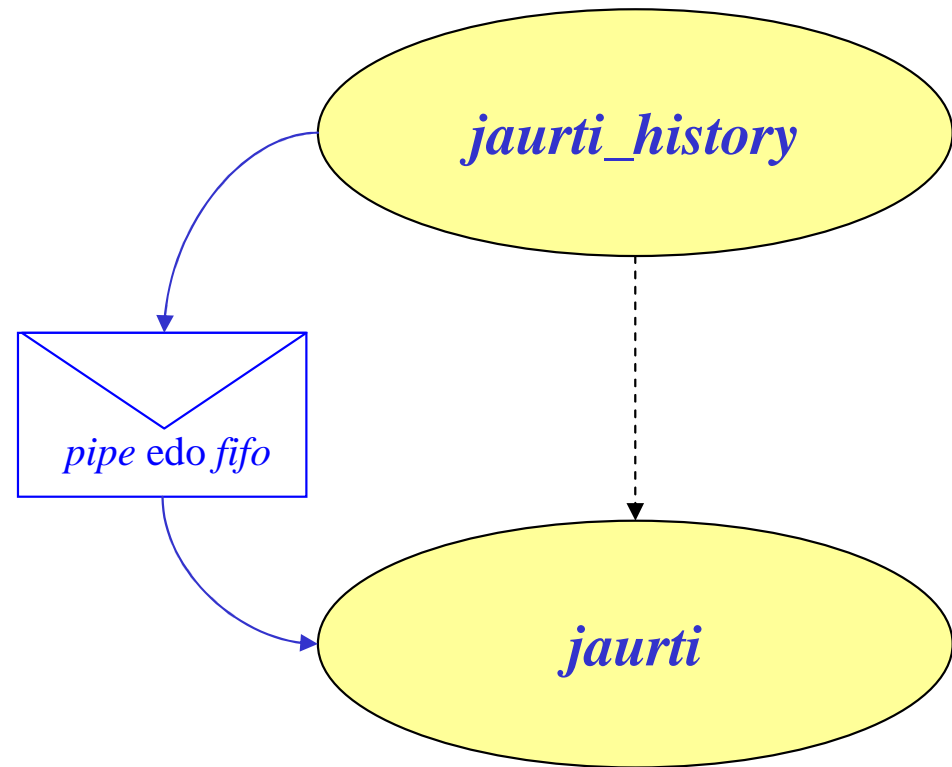
```
}
```

```

main(int argc, char *argv[])
{
    ...
    /* Hasieratzea */
    pipea_edo_fifo_sortu();
    jaurti_abiatu_sarrera_berbideratuz();
    while (TRUE)
    {
        irakurri_komando_lerroa();
        if (!_karakterez_hasten_da) {
            komando_lerroa_bilatu();
            idatzi_komando_lerroa_pipean();
        }
        if (Exit_da) Amaitu();
        else if (History_da) Idatzi_Historia();
        else idatzi_komando_lerroa_pipean();
    }
    ....
}

```

jaurti_history



jaurti_history

```
main(int argc, char *argv[])  
{  
    char zerrenda[24][80];  
    char komandoa[80];  
    char *komandoa_hist;  
    int pfd[2];
```

```
    pipe(pfd);  
    if (fork() == 0) { /* umea (jaurti) */  
        close(0);  
        dup(pfd[0]);  
        close(pfd[0]); close(pfd[1]);  
        execlp("jaurti", "jaurti", NULL);
```

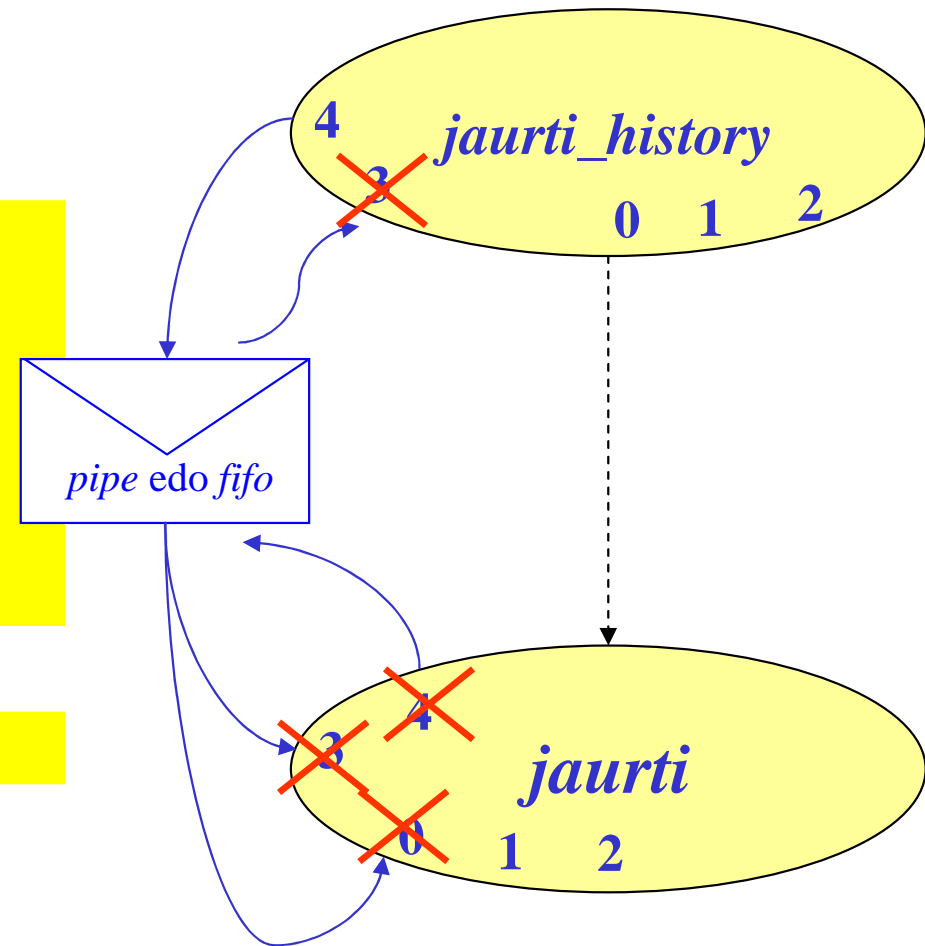
```
    }  
    close(pfd[0]); /* gurasoak ez du erabiliko */
```

```
    while (TRUE) {
```

```
        ...
```

```
    }
```

```
}
```

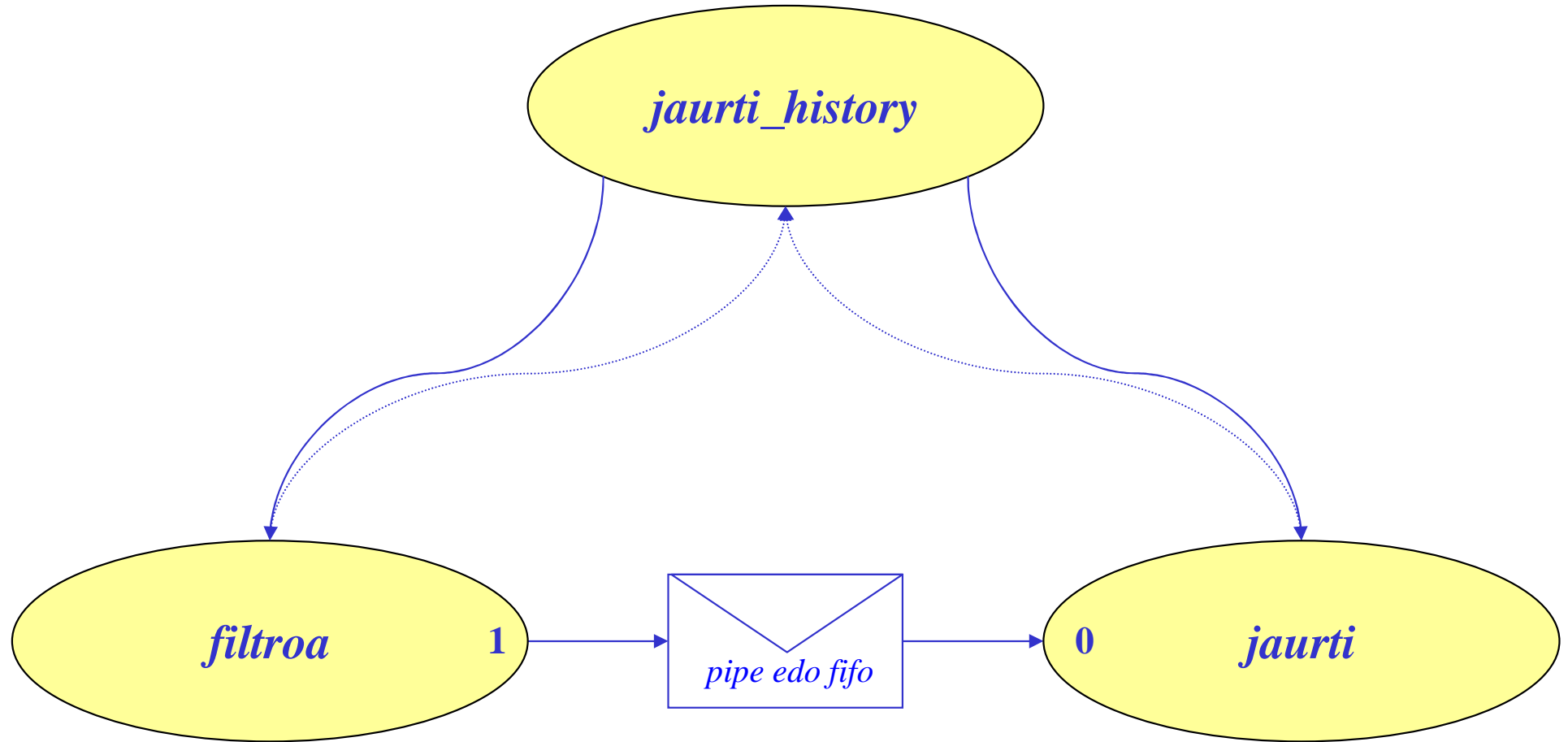


jaurti_history (jarraipena)

```
while (TRUE) {
    if ((n = read(0, komandoa, 80)) == 0) break;    /* Irteera EOF-gatik */
    komandoa[n] = '\0';
    if (hasten_da(komandoa, '!')) {
        if ((komandoa_hist = bilatu_historian(zerrenda, komandoa)) == NULL) {
            write(1, "Komando okerra\n", 15);
            continue;    /* berriro while-ra pasatzen da */
        }
        strcpy(komandoa, komandoa_hist);
    }
    if (! strcmp(komandoa, "exit")) break;    /* Irteera EXIT-gatik */
    if (! strcmp(komandoa, "history")) idatzi_historia(zerrenda);
    else {
        write(pfd[1], komandoa, strlen(komandoa));
        gehitu_historiari(zerrenda, komandoa);
    }
}
close(pfd[1]);    /* Honek EOF baldintza suposatzen du pipe-an, jaurti desblokeatuz */
wait(NULL);    /* gurasoa jaurti umearen amaieraren zain gelditzen da */
```



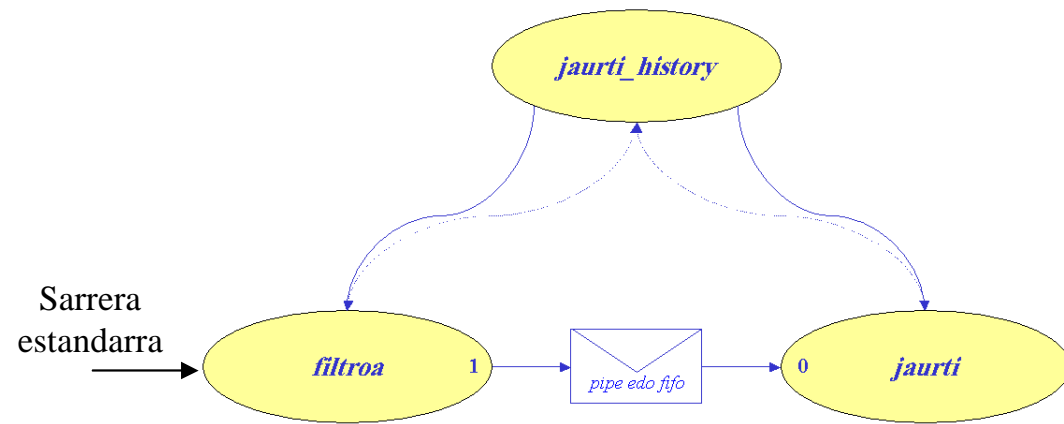
jaurti_history (hobekuntza)



jaurti_history (*hobekuntza*)

```
int main(int argc, char *argv[])
{
    ...
    /* Hasieratzea */
    pipea_edo_fifo_sortu();
    filtroa_abiatu_irteera_berbideratuz();
    jaurti_abiatu_sarrera_berbideratuz();
    ume_baten_zain_geratu(wait);
    if (filtroa_amaitu_da)
        jaurti_amaitu(kill);
    else /* jaurti amaitu da*/
        filtroa_amaitu(kill);
    ....
}
```

filtroa prozesuak sarrera estandarretik komandoak jaso eta tratatu ondoren pipean idatzi.

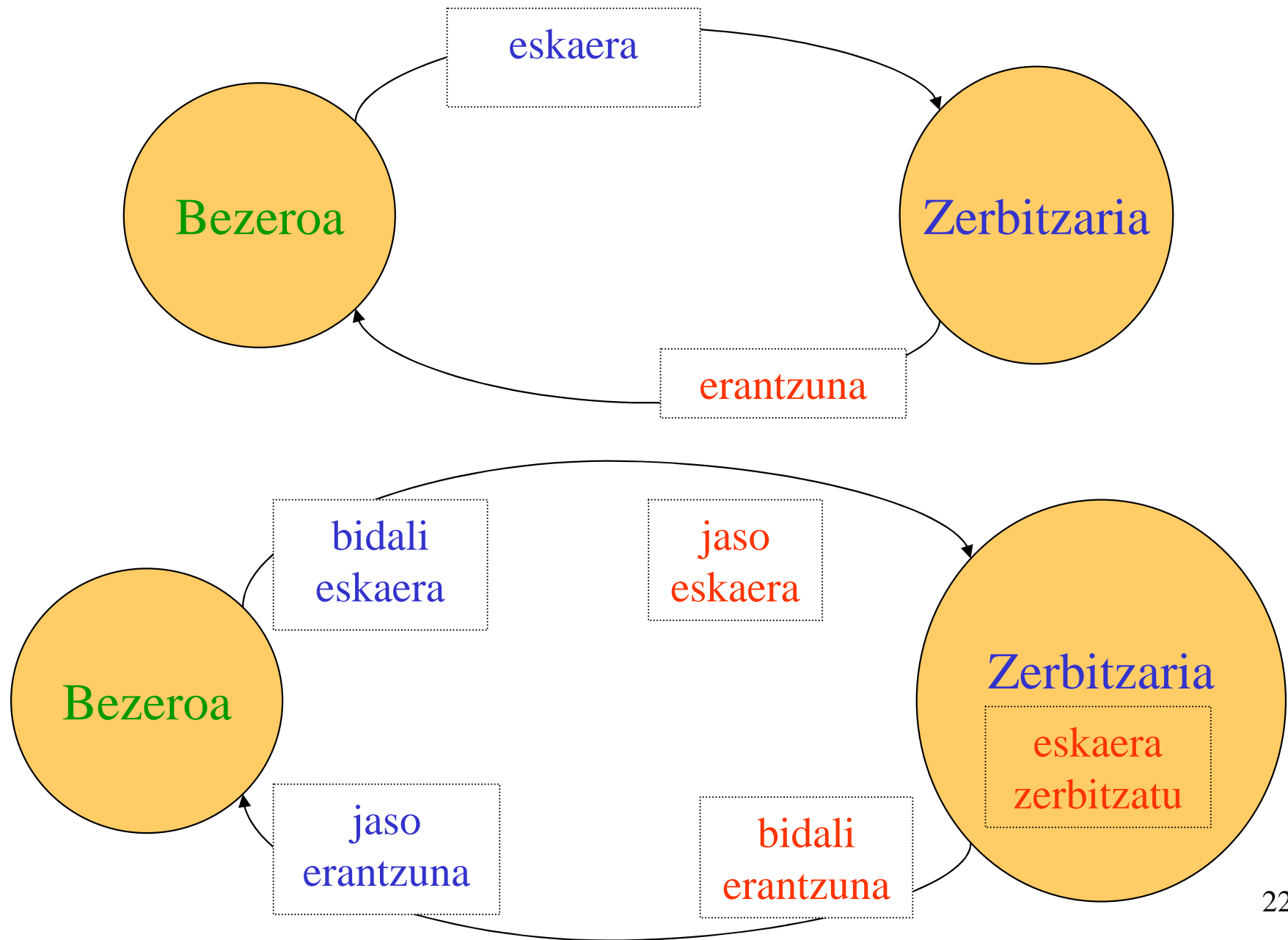


- Jaurti_history-k **kontrol lana** burutu. (umeak sortu eta zain geratu - *wait*).

filtrao

```
main(int argc, char *argv[]) {
    while (TRUE) {
        if ((n = read(0, komandoa, 80)) == 0) break; /* Irteera EOF-gatik */
        komandoa[n] = '\0';
        if (hasten_da(komandoa, '!')) {
            if ((komandoa_hist = bilatu_historian(zerrenda, komandoa)) == NULL) {
                write(2, "Komando okerra\n", 15);
                continue; /* berriro while-ra pasatzen da */
            }
            strcpy(komandoa, komandoa_hist);
        }
        if (! strcmp(komandoa, "exit")) break; /* Irteera EXIT-gatik */
        if (! strcmp(komandoa, "history")) idatzi_historia(zerrenda);
        else {
            write(1, komandoa, strlen(komandoa));
            gehitu_historiari(zerrenda, komandoa);
        }
    }
    exit(0);
}
```

Bezero-Zerbitzari eredua



Bezero-Zerbitzari eredua

```
bezeroaren_errutina()  
{  
    bidali_eskaera();  
    jaso_erantzuna();  
}
```

```
X_zerbitzaria()  
{  
    hasieratzea();  
    while (TRUE)  
    {  
        jaso_eskaera();  
        eskaera_zerbitzatu();  
        bidali_erantzuna();  
    }  
}
```

Bezero-Zerbitzari eredua

- Bezeroa (eskaerak):

- *Sinkronoak*

- Erantzuna itxaroten du (datuak) zerbitzaritik
 - a) Eskatera jasotzen denean
 - b) Eskatera zerbitzatzaren hasten denean
 - c) Eskatera zerbitzatu (amaitu) denean

```
bezeroaren_errutina()  
{  
    bidali_eskaera();  
    jaso_erantzuna();  
}
```

- *Asinkronoak*

- Ez du erantzunik itxaroten

```
bezeroaren_errutina()  
{  
    bidali_eskaera();  
}
```

- Zerbitzaria

- *Elkarreragilea*

- Zerbitzariak aldiko eskaera bat zerbitzatzaren du

- *Konkurrentea*

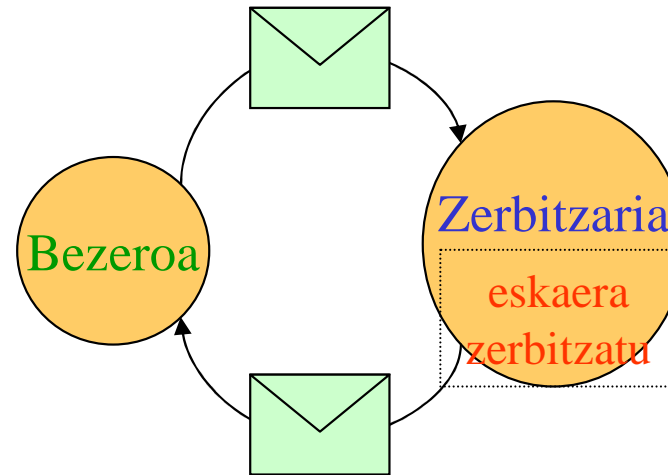
- Zerbitzariak aldi berean eskaera bat baino gehiago zerbitzatu ditzake (eskaera bakoitza beste prozesu bati pasatuz)

Bezero-Zerbitzari ereduak

- Zerbitzaria

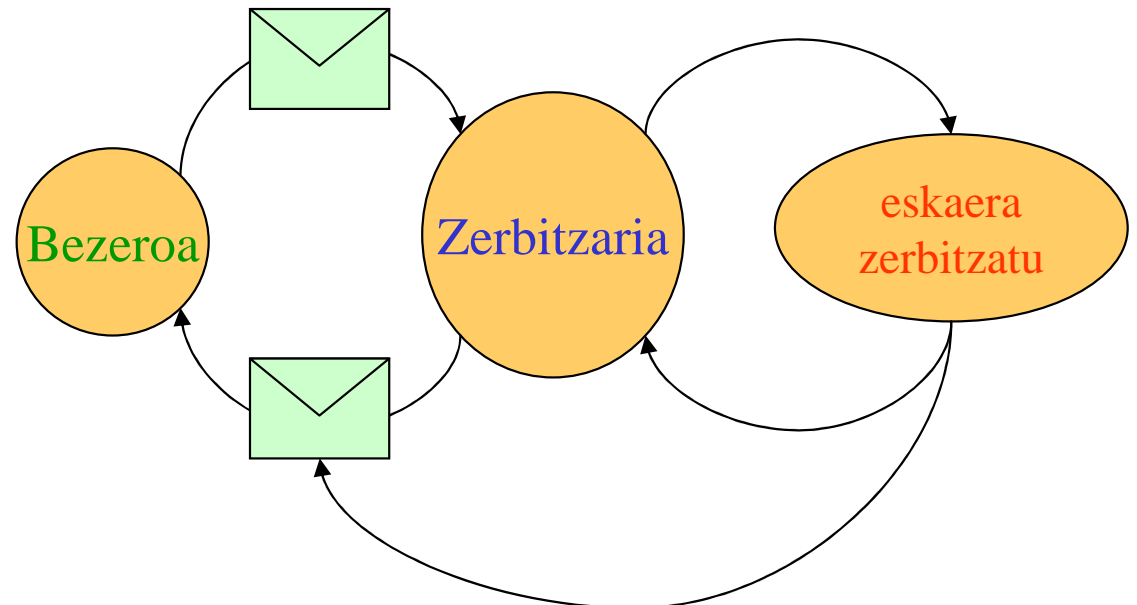
- *Elkarreragilea*

- Zerbitzaria berak burutzen du eskaera



- *Konkurrentea*

- Zerbitzariak beste prozesu bati (umea ala ez) agintzen dio eskaera burutzea



Bezeroaren errutinaren adibidea (sinkronoa/asinkronoa)

```
bezeroaren_errutina(int asin_sin, int buzoi_espl)
{
    if (asin_sin == SINKRO) { /* SINKRONOA */
        eraiki_eskaera(param, nire_buzoia, &eskaera);
        bidali_eskaera bidali(X_Zerbitzari_buzoia, &eskaera);
        jaso_erantzuna jaso(nire_buzoia, &erantzuna);
    } else { /* ASINKRONOA */
        eraiki_eskaera(param, buzoi_espl, &eskaera);
        bidali_eskaera bidali(X_Zerbitzari_buzoia, &eskaera);
    }
}
```

Zerbitzariaren adibidea

```
X_Zerbitzaria()  
{  
    hasieratu_egiturak(X_Zerbitzaria);  
    hasieratu_buzoiak(X_Zerbitzaria);  
    while (TRUE)  
    {  
        jaso  
        eskaera    jaso(X_Zerbitzari_buzoia, &esk);  
                  esk_param_lortu(esk, &param, &bez_buzoia);  
  
        eskaera  
        zerbitzatu zerbitzatu_eskatutako_eragiketa(param);  
  
        bidali  
        erantzuna  bidali(bez_buzoia, "ESK_AMAIERA");  
    }  
}
```