

3. gaia

Exekuzio konkurrentea

3.1 Konposizio paraleloa – ekintzen tartekatzea

Bitez P eta Q prozesuak, orduan

(P||Q) –k adierazten du

P eta Q-ren exekuzio konkurrentea.

|| : konposizio paraleloaren eragilea.

MUXU = (muxukatu->STOP) .

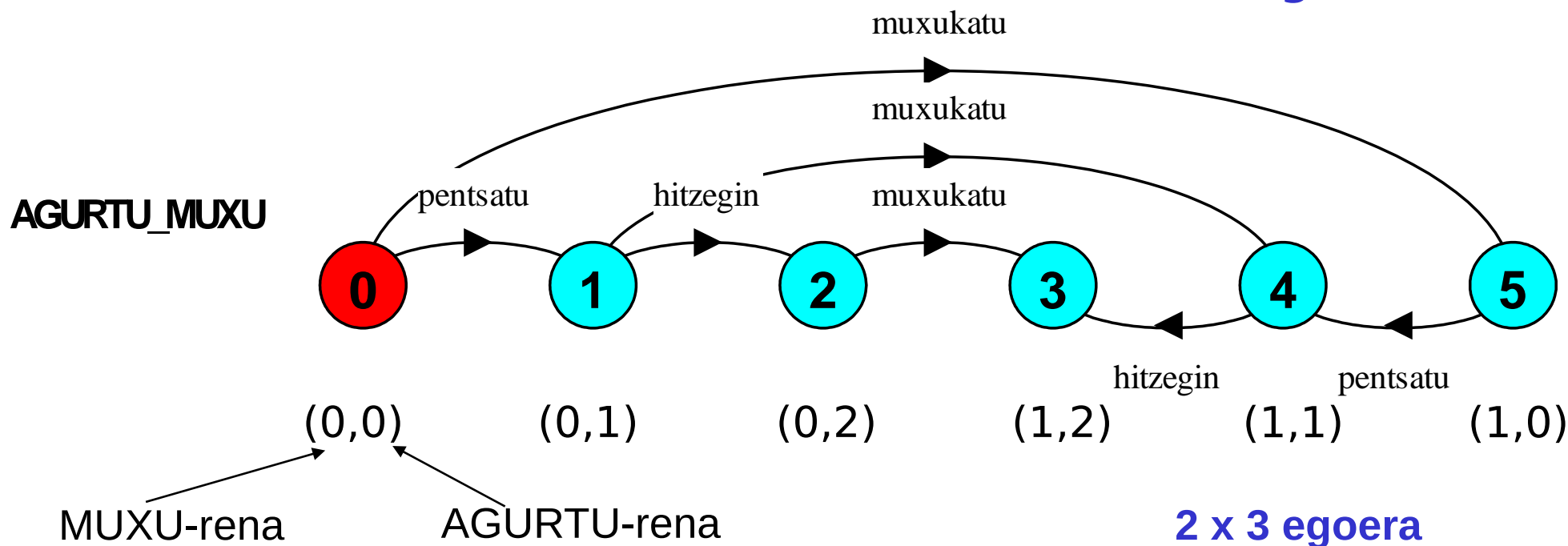
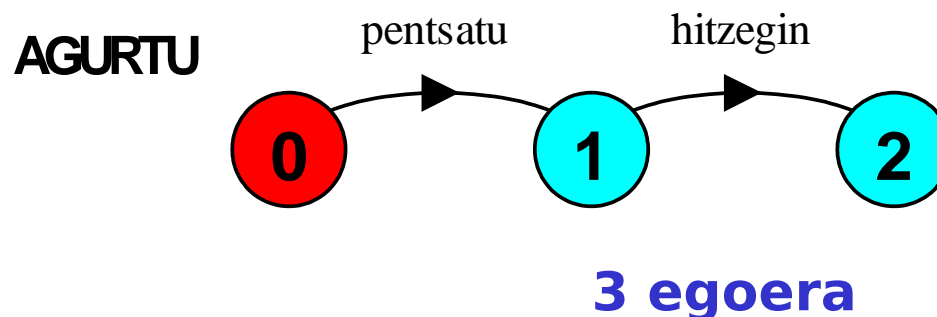
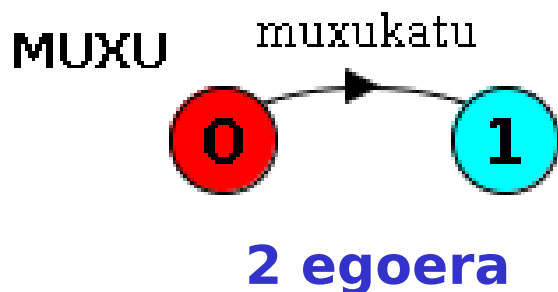
AGURTU = (pentsatu->hitzegin->STOP) .

||AGURTU_MUXU = (MUXU || AGURTU) .

pentsatu→hitzegin→muxukatu
pentsatu→muxukatu→hitzegin
muxukatu→pentsatu→hitzegin

} Ekintzak tartekatzean traza posibleak.

Konposizio paraleloa – ekintzen tartekatzea



Konposizio paraleloa – ekintzen tartekatzea

Irrati-erlojuaren adibidea:

```
ERLOJUA = (tick->ERLOJUA).
```

```
IRRATIA = (on->off->IRRATIA).
```

```
||ERLOJU_IRRATIA = (ERLOJUA || IRRATIA).
```

Margotu LTSa eskuz (eta egiaztatu gero LTSArekin).

Trazak?

Egoera kopurua?

Konposizio paraleloa – lege algebraikoak

Konmutatiboa: $(P||Q) = (Q||P)$

Asoziatiboa: $(P||(Q||R)) = ((P||Q)||R)$
 $= (P||Q||R).$

Interakzioa modelatzen – konpartituriko ekintzak

Konpartituriko ekintzak (shared actions):

- Konposizio bateko prozesuek amankomunean dituzten ekintzak.
- Prozesuen interakzioa modelatzen dute

Konpartitua ez den ekintza bat modu arbitrarioan izan daiteke tartekatua.

Konpartituriko ekintza bat, ekintza hori konpartitzen duten prozesu guztiek aldi berean exekutatu dute.

```
EGIL = (egin->prest->EGIL) .  
ERAB = (prest->erabili->ERAB) .  
|| EGIL_ERAB=(EGIL || ERAB) .
```

EGILea eta **ERAB**iltzailea
prest daudenean
sinkronizatzen dira

*Margotu LTSa eskuz (eta egiaztatu gero LTSArekin).
Trazak?
Egoera kopurua?*

Interakzioa modelatzen (*handshake*)

EGIL2 = (egin->prest->erabilita->EGIL2).

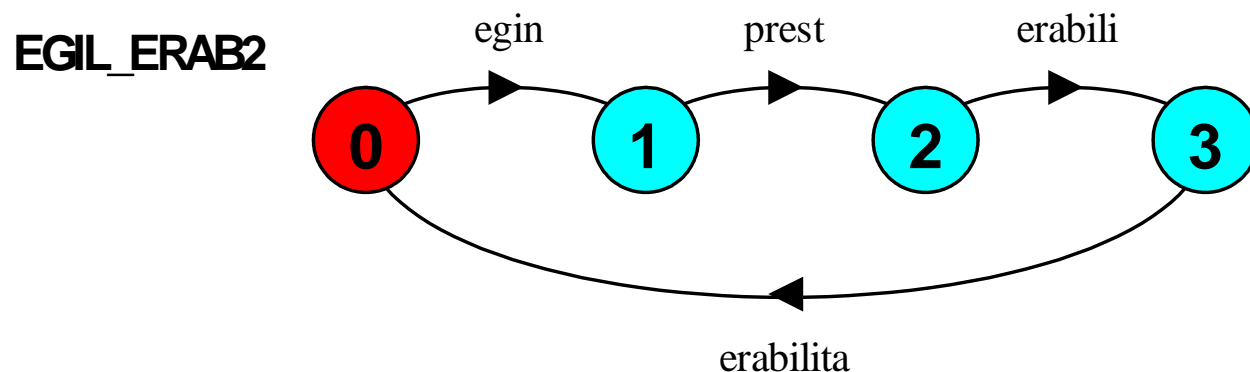
3 egoera

ERAB2 = (prest->erabili->erabilita->ERAB2).

3 egoera

|| EGIL_ERAB2 = (EGIL2 || ERAB2).

3 x 3 egoera?



4 egoera!!!

Interakzioak
portaera osoa
murrizten du.

Interakzioa modelatzen – konpartituriko baliabideak

Guk bi prozesuren arteko interakzioa modelatzeko

- ez ditugu erabiliko konpartituriko ekintzak,
- **konpartituriko baliabide** bat adieraziko duen beste prozesu “**pasibo**” bat baizik

```
range Bool = 0..1
EGIL3 = (egin->EGIL3).
ERAB3 = (erabili->ERAB3).
SINK          = SINK [0],
SINK[i:Bool] = (  when (i==0) egin      -> SINK[1]
                  |  when (i==1) erabili -> SINK[0]
                  ).
||EGIL_ERAB3 = (EGIL3 || ERAB3 || SINK).
```

*LTSA ulertu ezazu LTSA erabiliz.
Trazak? Egoera kopurua?*

Ariketak

- 1) EGIL_ERAB3 adibidean, gehitu BOTATZAILE prozesu bat, “erabili” ekintza egin ondoren "bota" egiten duena.

Traza:

egin->erabili->bota->egin->erabili->bota->...

- 2) Oraingoan, EGIL_ERAB3 adibidean, modelatu BOTATZAILE2 prozesu bat, "bota" egiten duena “erabili” ondoren edota “erabili” egin gabe (hori bai, “egin” ekintza egin ondoren).

Traza posiblea:

egin->erabili->bota->egin->bota->egin->bota->egin->erabili->bota->...

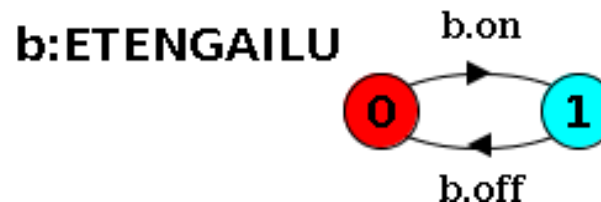
Prozesuak etiketatzea

a:P

P-ren alfabetoko ekintza-etiketa bakoitzari a aurrizkia jartzen dio.

Etengailu prozesu baten bi **instantzia**:

```
ETENGAILU = (on->off->ETENGAILU).
|| BI_ETENGAILU = (a:ETENGAILU || b:ETENGAILU).
```



Etengailu prozesuaren **instantzien** array bat:

```
const N=3
range R=1..N
```

```
|| ETENGAILUAK = (forall[i:R] s[i]:ETENGAILU).
```

edo

```
|| ETENGAILUAK = (s[i:R]:ETENGAILU).
```

Prozesuak etiketatzea aurrizki-etiketazko multzo batekin

$\{a1, \dots, ax\} :: P$

P-ren alfabetoko n etiketa guztiak $a1.n, \dots, ax.n$ etiketekin ordezkatzeko du. Gainera P-ren definizioan dauden $(n \rightarrow X)$ trantsizio guztiak $(\{a1.n, \dots, ax.n\} \rightarrow X)$ trantsizioekin ordezkatzeko dira.

Prozesuei aurrizkiak jartzea erabilgarria da **konpartituriko** baliabideak modelatzeko:

range Bool = 0..1

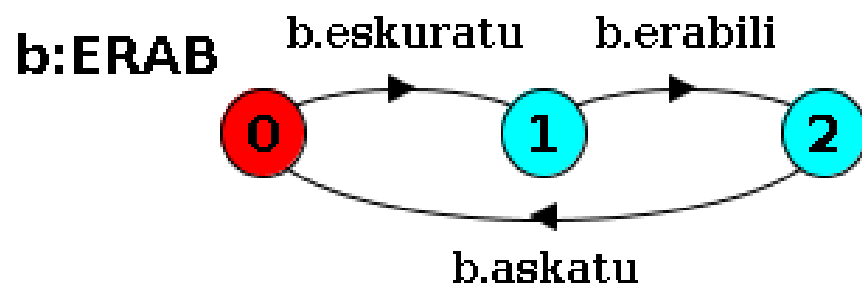
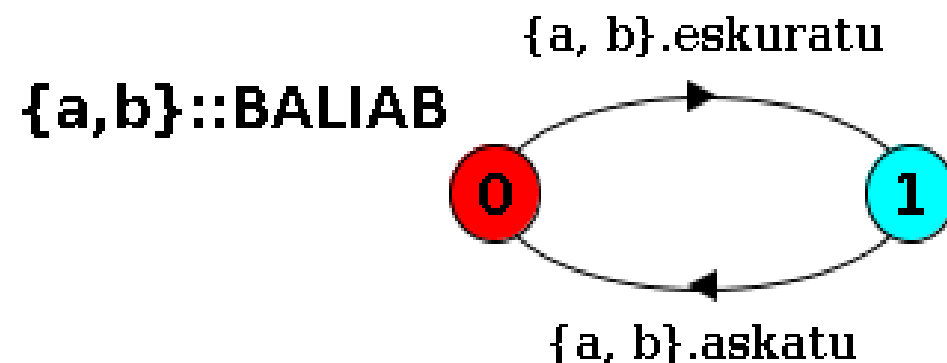
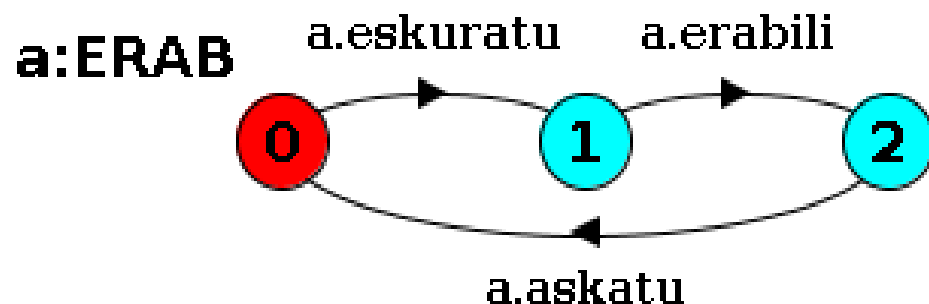
ERAB = (eskuratu -> erabili -> askatu -> ERAB) .

BALIAB = BALIAB[0],

**BALIAB[i:Bool] = (when (i==0) eskuratu -> BALIAB[1]
| when (i==1) askatu -> BALIAB[0]) .**

|| ERAB_BALIAB = (a:ERAB || b:ERAB || {a,b} :: BALIAB) .

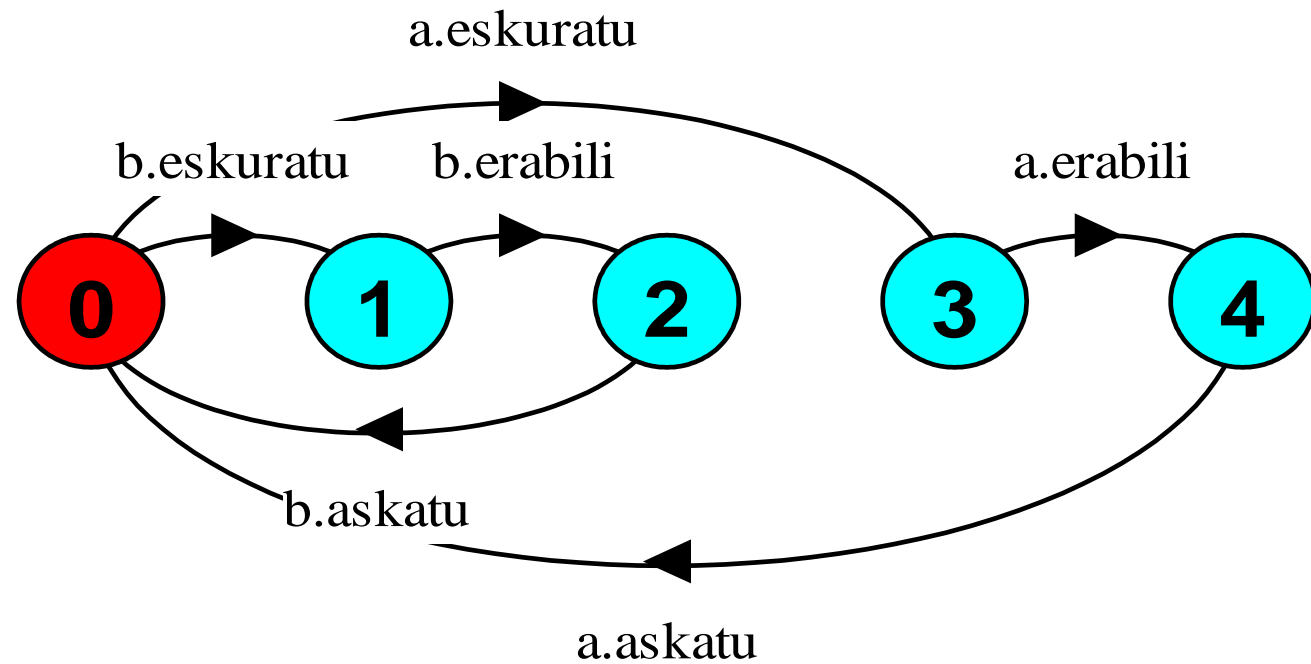
Prozesuen aurrizki-etiketak konpartituriko baliabidetarako.



Nola ziurtatzen du ereduak baliabidea eskuratzen duen erabiltzailea dela askatuko duena?

Prozesuen aurrizki-etiketak konpartituriko baliabidetak.

ERAB_BALIAB



Ariketak

- 3) ERAB_BALIAB adibidean, gehitu EGILE prozesu bat, "egin" egiten duena, a edo b ERABiltzaileek "erabili" aurretik.

Traza posiblea:

```
egin->b.eskuratu->b.erabili->b.askatu->egin->b.eskuratu->b.erabili->  
b.askatu->egin->a.eskuratu->a.erabili->a.askatu->egin->b.eskuratu->  
b.erabili->...
```

- 4) Oraingoan 2 EGILE egongo dira, x eta y. Bi egileek "egin" egin beharko dute (berdin da zein ordenean) a edo b ERABiltzaileek erabili aurretik.

Traza posiblea:

```
y.egin->x.egin->b.eskuratu->b.erabili->b.askatu->x.egin->y.egin->  
b.eskuratu->b.erabili->b.askatu->x.egin->y.egin->a.eskuratu->  
a.erabili-> a.askatu->y.egin->...
```

Ekintzak berretiketatzea

Prozesuei berretiketatzeko-funtzioak aplikatuko dizkiegu, ekintzen etiketen izenak aldatzeko.

Berretiketatzeko-funtzioen itxura orokorra hau da:

$l\{etikberria_1etikzaharra_1, \dots etikberria_n etikzaharra_n\}$

```
range Bool = 0..1
EGIL3 = (egin->EGIL3).
ERAB3 = (erabili->ERAB3).
SINK      = SINK [0],
SINK[i:Bool] = ( when (i==0) utzi  -> SINK[1]
                  | when (i==1) hartu -> SINK[0] ).
```

Horrela berretiketatzeko gendake ziurtatzeko prozesu konposatuak sinkronizatzen direla ekintza zehatzetan:

```
||EGIL_ERAB3 = (EGIL3 || ERAB3 || SINK)
               /{utzi/egin, hartu/erabili}.
```

Ekintzak ezkutatzeara (*hiding*)

konplexutasuna gutxiagotzeko abstrakzioa.

$\backslash\{a1..ax\}$ ezkutatzearagilea

P prozesuari aplikatzen diogunean, P alfabetoko $a1..ax$ ekintzen izenak ezabatzen dira, ekintza hauek “isilak” bihurtuz.

- Ekintza isil hauek **tau** etiketatzen dira.
- Prozesu ezberdinetako ekintza isilak ez dira konpartitzen.

Batzuetan egokiagoa da erakutsiko diren etiketen multzoa adieraztea:

$@\{a1..ax\}$ interfaze-eragilea

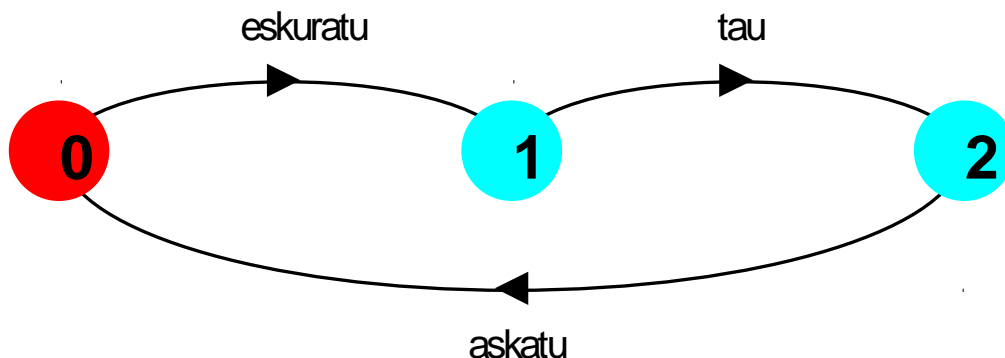
P prozesuari aplikatzen diogunean, $a1..ax$ multzoan ez dauden P alfabetoko ekintzak ezkututzen dira.

Ekintzak ezkutatzea

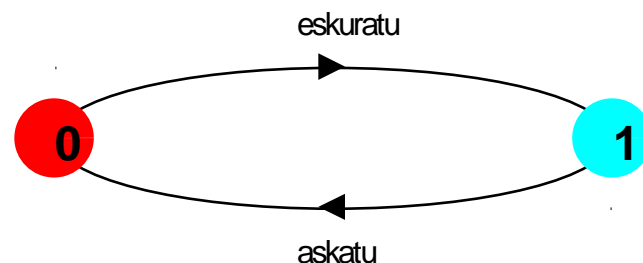
Ondoko definizioak baliokideak dira:

ERABILTZAILEA =
(eskuratu->erabili->askatu->ERABILTZAILEA)
\{erabili\}.

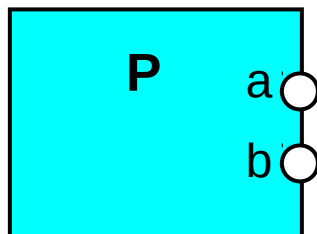
ERABILTZAILEA =
(eskuratu->erabili->askatu->ERABILTZAILEA)
@{eskuratu,askatu}.



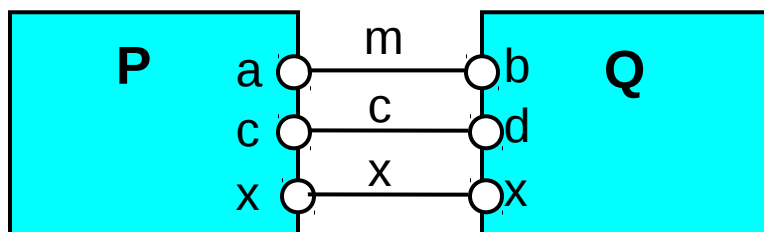
Minimizazioak **tau** ekintza ezkutatuak ezabatzen ditu portaera baliokidea duen LTS bat sortuz.



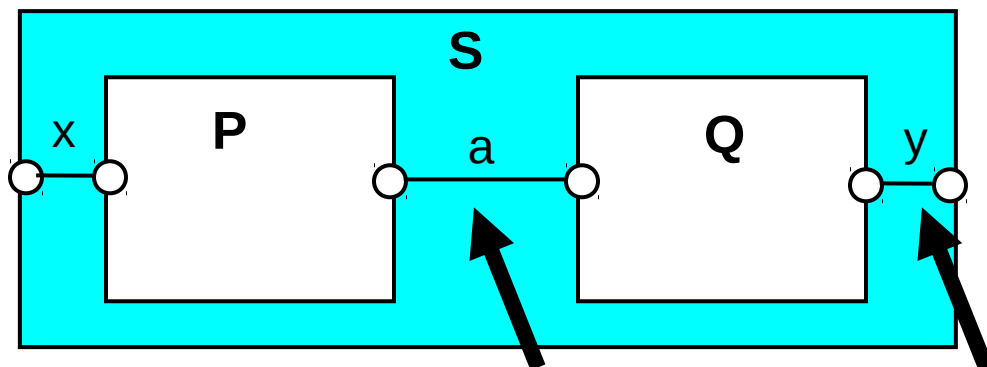
Egitura-diagramak



{a,b} alfabetoa
duen P prozesua.



Konposizio paraleloa
 $(P||Q) / \{m/a, m/b, c/d\}$



Prozesu konposatua
 $||S = (P||Q) @ \{x,y\}$

Ekintza
konpartitua

Ekintza
ikuskorra

Egitura-diagramak

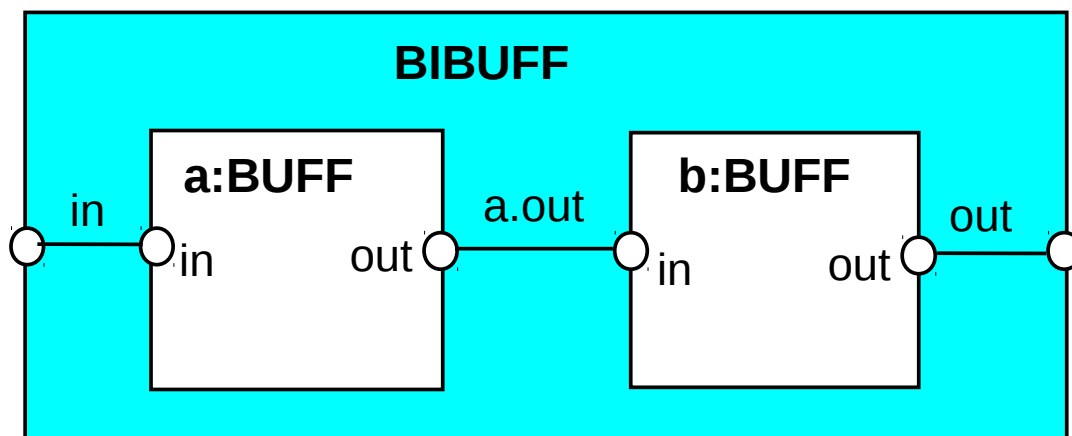
Egitura-diagramak eredu baten egitura estatikoa adierazteko erabiltzen dira.

- Egoera-makinek ez baitute egitura estatikoa harrapatzen.

Egitura estatikoa ondoko eragileek deskribatzen dute: konposizio paraleloa, berretiketatzea eta ezkutatzeta.

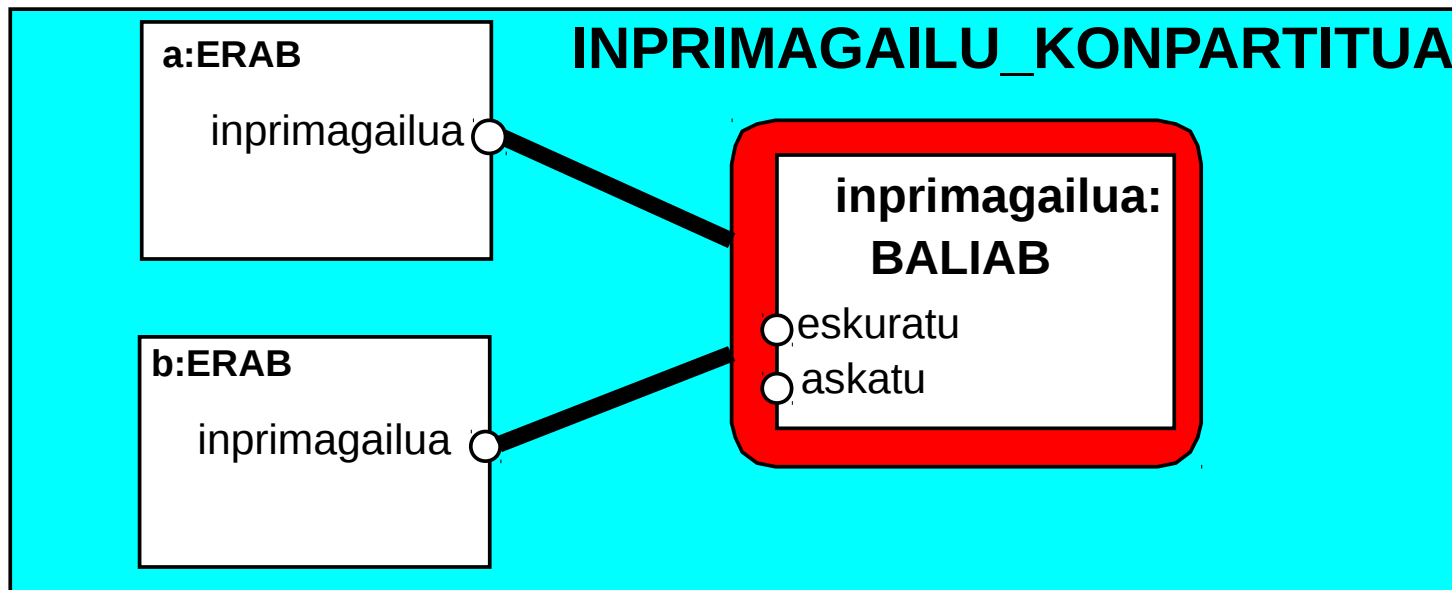
range $T = 0..3$

$\text{BUFF} = (\text{in}[i:T] \rightarrow \text{out}[i] \rightarrow \text{BUFF}).$



**$|| \text{BIBUFF} = (\text{a:BUFF} || \text{b:BUFF})$
 $/\{\text{in} \quad / \text{a.in},$
 $\quad \text{a.out} / \text{b.in},$
 $\quad \text{out} \quad / \text{b.out}\}$
 $@\{\text{in}, \text{out}\}.$**

Egitura-diagramak – baliabideak konpartitzea



```

ERAB = (inprimagailua.eskuratu->erabili
        ->inprimagailua.askatu->ERAB) .

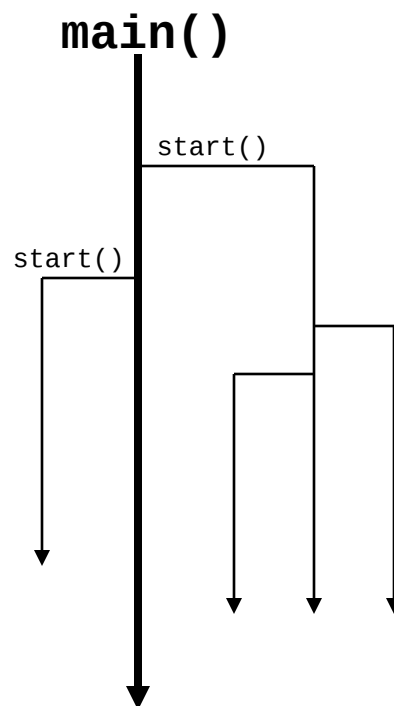
BALIAB      = BALIAB[0],
BALIAB[i:0..1] = ( when (i==0) eskuratu -> BALIAB[1]
                  | when (i==1) askatu   -> BALIAB[0] ) .

|| INPRIMAGAILU_KONPARTITUA = ( a:ERAB
                                || b:ERAB
                                || {a,b}::inprimagailua:BALIAB) .

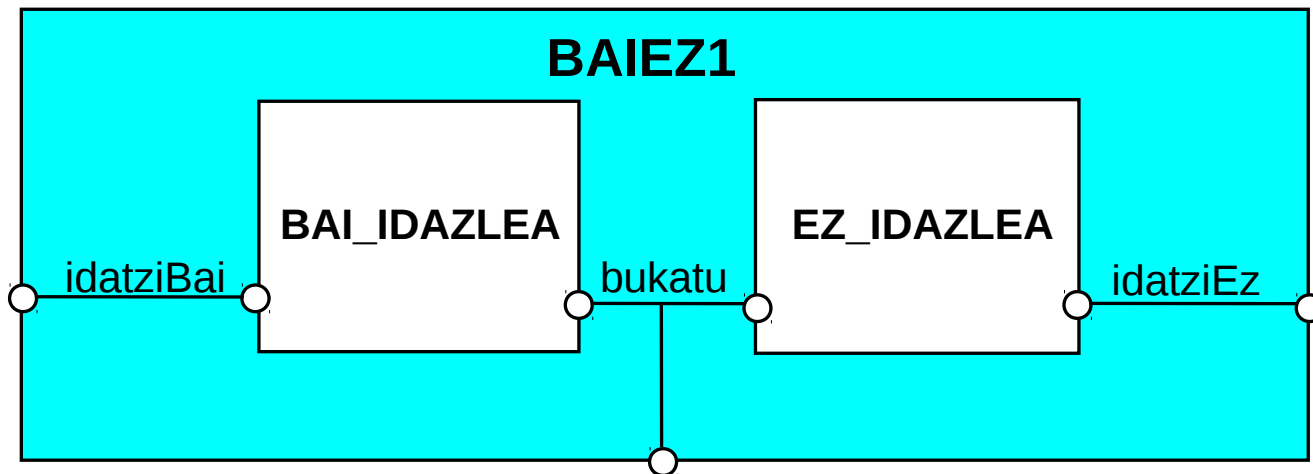
```

3.2 Hari-anitzetako (multi-threaded) programak Java-n

Java-n konkurrentzia gertatzen da hari bat baino gehiago bizirik dagoenean.



BAIEZ1 eredua



BAI_IDAZLEA = (**idatziBai** -> **BAI_IDAZLEA**
| **bukatu** -> **STOP**).

EZ_IDAZLEA = (**idatziEz** -> **EZ_IDAZLEA**
| **bukatu** -> **STOP**).

||BAIEZ1 = (**BAI_IDAZLEA** || **EZ_IDAZLEA**).

Honela interpretatu:

input: **run**

outputs: **idatziBai** eta
idatziEz

Baiez1 implementazioa

```
class BaiIdazlea extends Thread {  
    public void run() {  
        while(true){  
            System.out.println("bai");  
            try {sleep(1000);}  
            catch (InterruptedException e){}  
        }  
    }  
}  
  
class EzIdazlea extends Thread {  
    public void run() {  
        while(true){  
            System.out.println("ez");  
            try {sleep(400);}  
            catch (InterruptedException e) {}  
        }  
    }  
}
```

run() bukatuko da
Thread.interrupt()-ek
etenaldi bat goratzen badu.

Baiez1 implementazioa

```
class BaiEz1App{
    public static void main (String args[]) {
        BaiIdazlea bai = new BaiIdazlea();
        EzIdazlea ez = new EzIdazlea();
        bai.start();
        ez.start();
    }
}
```

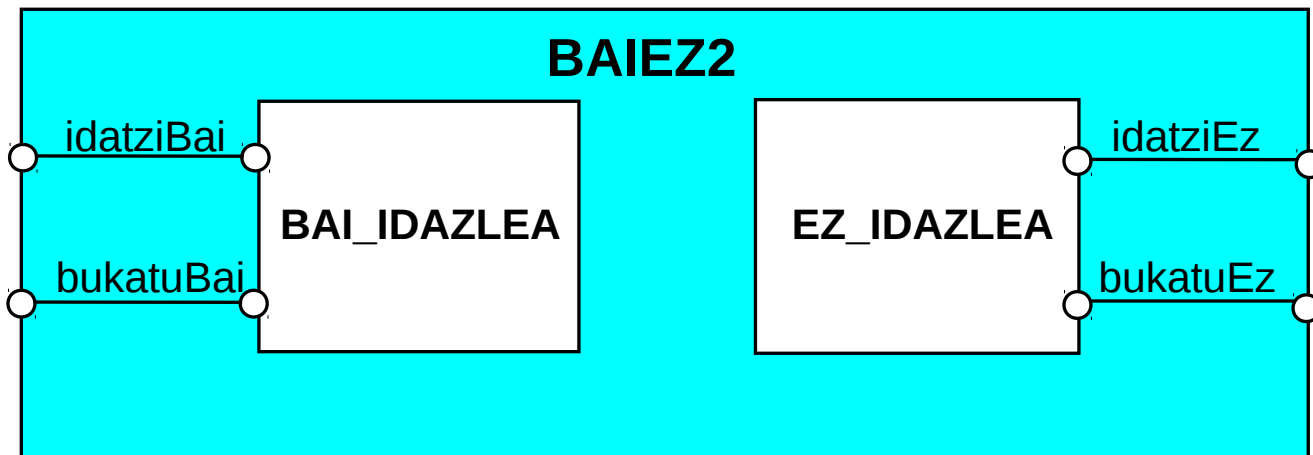
← Hari berriak sortu

← Harien exekuzioa hasieratu

Klase nagusiak (main) sortzen dituen hariak konkurrenteki exekutatzen dira, denbora elkarbanatuz.

Programa bukatuko da hariak bukatzean (kasu honetan etenaldi bat goratzean).

BAIEZ2 eredua



**BAI_IDAZLEA = (idatziBai -> BAI_IDAZLEA
| bukatuBai -> STOP).**

**EZ_IDAZLEA = (idatziEz -> EZ_IDAZLEA
| bukatuEz -> STOP).**

|| BAIEZ2 = (BAI_IDAZLEA || EZ_IDAZLEA).

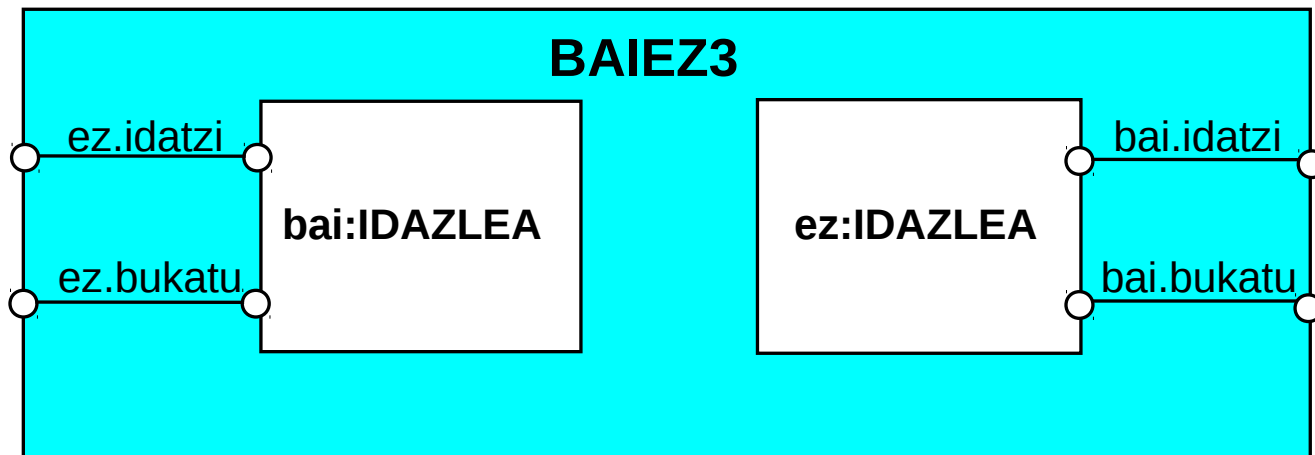
Baiez2 implementazioa

```
class BaiIdazlea extends Thread {
    public void run() {
        int i;
        for (i=0;i<4;i++) {
            System.out.println("bai");
            try {sleep(1000);}
            catch (InterruptedException e) {}
        }
    }
}

class EzIdazlea extends Thread {
    public void run() {
        int i;
        for (i=0;i<20;i++) {
            System.out.println("ez");
            try {sleep(400);}
            catch (InterruptedException e) {}
        }
    }
}
```

Programa bukatuko da bi hariak bukatzean.

BAIEZ3 eredua



**IDAZLEA = (idatzi -> IDAZLEA
|bukatu -> STOP).**

||BAIEZ3 = (a:IDAZLEA||b:IDAZLEA) .

Baiez3 implementazioa

```
class Idazlea extends Thread {  
    private String baiEz;  
    private int denbora, kopurua;  
    public Idazlea(String s, int i, int j){  
        baiEz=s;  
        denbora=i;  
        kopurua=j;  
    }  
    public void run() {  
        int i;  
        for (i=0;i<kopurua;i++) {  
            System.out.println(baiEz);  
            try {sleep(denbora);}  
            catch (InterruptedException e) {}  
        }  
        System.out.println(" Bukatuta (" +baiEz+" )");  
    }  
}
```

Baiez3 implementazioa

```
class BaiEz3App{  
    public static void main (String args[]) {  
        Idazlea bai = new Idazlea("bai",1000,4);  
        Idazlea ez  = new Idazlea("ez",400,20);  
        bai.start();  
        ez.start();  
    }  
}
```

Klase bereko bi instantzia definitu eta konkurrenteki exekutatzeko dira.

Instantzia bakoitzak bere atributuak ditu: **String s, int i, int j**

Ariketak

5. BaiEz inplementazioa egokitu, bi hari sortu beharrean, hiru sortzeko: “bai”, “ez” eta “agian”
6. BaiEz inplementazioa egokitu, exekuzioan zehar harien abiadura aldatzeko.
Adibidez hasieran “bai” azkar idazten du eta bukaeran poliki,
eta “ez” hasieran poliki eta gero azkar.
7. Bi kotxe (bi hari) izango ditugu. Kotxeak “+” eta “*” ikurrekin adieraziko dira eta ezkerretik eskuinera mugituko dira, bakoitza abiadura desberdinean.
Aplikazioa JAVA erabiliaz inplementatu.
8. Kotxeen ariketa egokitu exekuzioan zehar kotxeen abiadura aldatzen joateko.
9. Irakasleak inplementatutako kotxeen ariketaren bertsioa (irudiekin) egokitu,
kotxeen abiadura aldatzen joateko exekuzioan zehar.
10. Lorategi batean bi ate daude. Pertsona bat sartzen den bakoitzean ate horrek kontagailu bat inkrementatzen du jakiteko zenbat sartu diren guztira.
Pertsonak sartuko dira lorategira beteta egon arte. Pertsonak ez dira aterako, soilik sartu.
FSP eredua eta Java programa egin.