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.

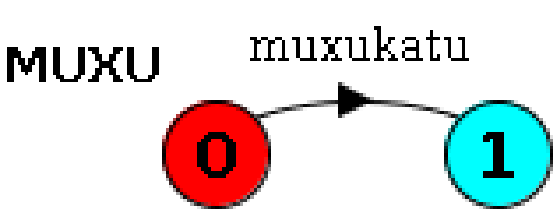
|| : konposizio paraleloaren eragilea.

```
MUXU      = (muxukatu->STOP).
AGURTU    = (pentsatu->hitzegin->STOP).

||AGURTU_MUXU = (MUXU || AGURTU).
```

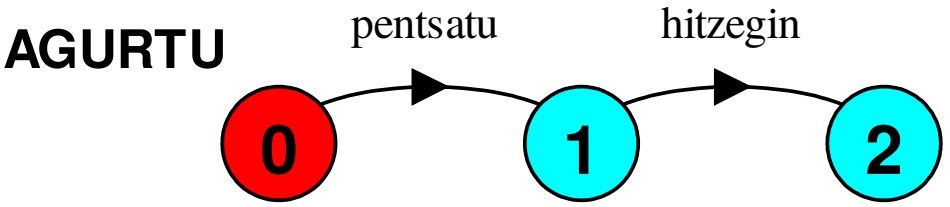
```
pentsatu→hitzegin→muxukatu
pentsatu→muxukatu→hitzegin
muxukatu→pentsatu→hitzegin
```

Ekintzak tartekatzean traza posibleak.

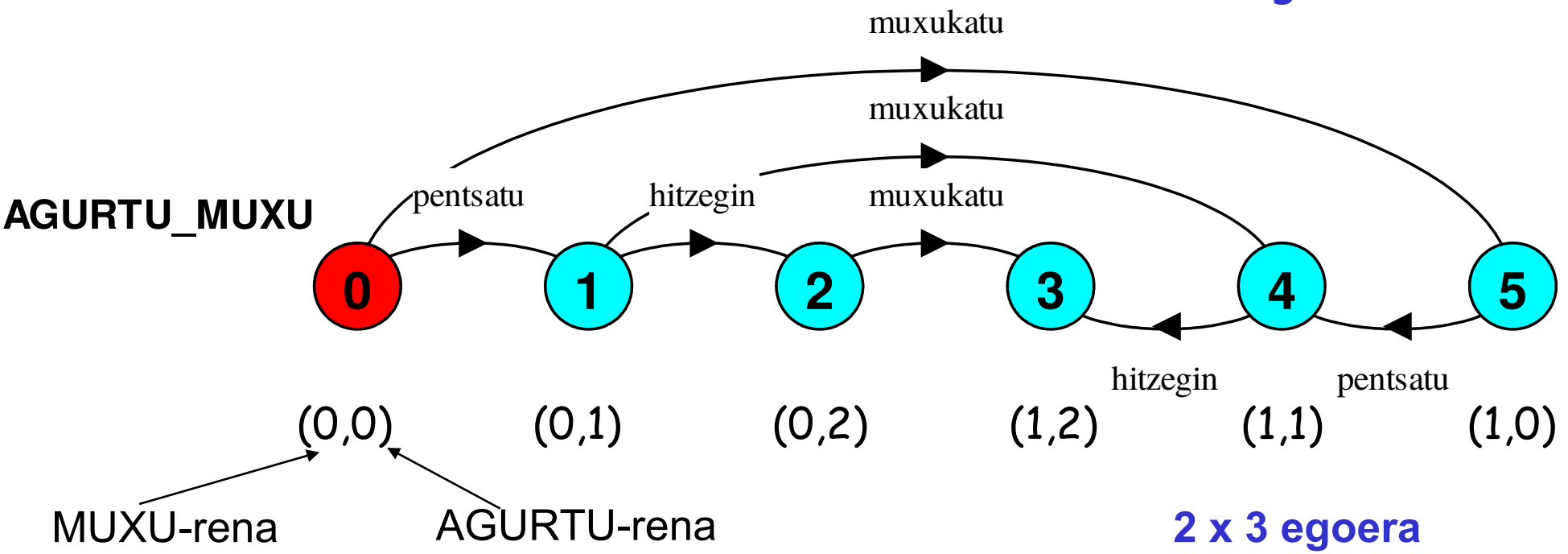
Konposizio paraleloa – ekintzen tartekatzea



2 egoera



3 egoera



2 x 3 egoera

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 arbitarioan izan daiteke tartekatua.

Konpartituriko ekintza bat, ekintza hori konpartitzen duten prozesu guztiek aldiberean exekutatuko dute.

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

||EGIL_ERAB=(EGIL || ERAB).
```

EGILea eta **ERABiltzailea** **prest** daudenean sinkronizatzen dira

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

Interakzioa modelatzen (*handshake*)

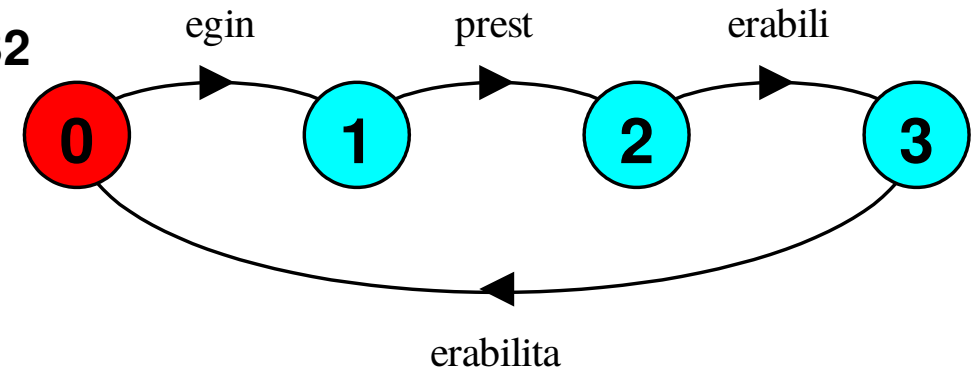
```
EGIL2 = (egin->prest->erabilita->EGIL2).

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

||EGIL_ERAB2 = (EGIL2 || ERAB2).
```

3 egoera

3 egoera

3 x 3 egoera?

EGIL_ERAB2



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:

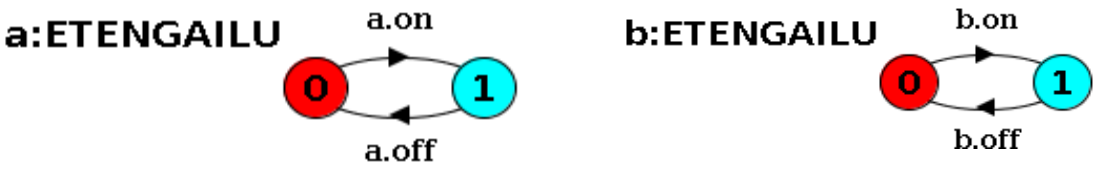
```
egin->erabili->bota->egin->bota->egin->bota->egin->erabili->
bota->...
```

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,..,ax}::P
P-ren alfabetoko n etiketa guztiak a1.n,…,ax.n etiketekin ordezkatzen du. Gainera P-ren definizioan dauden (n->X) trantsizio guztiak ({a1.n,…,ax.n} ->X) trantsizioekin ordezkatzen dira.

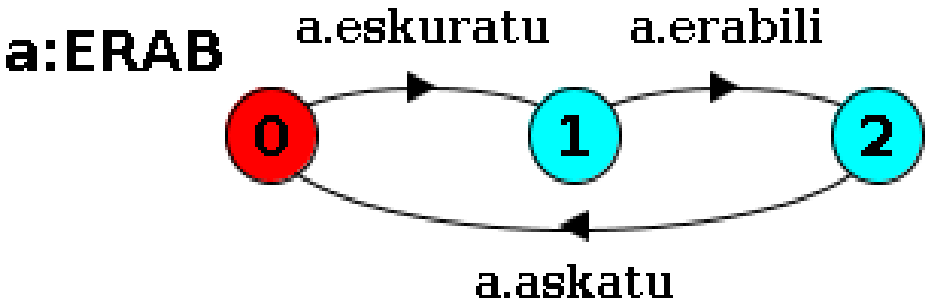
Prozesuei aurrizkiak jartzea erabilgarria da **konpartituriko** baliabideak modelatzeko:

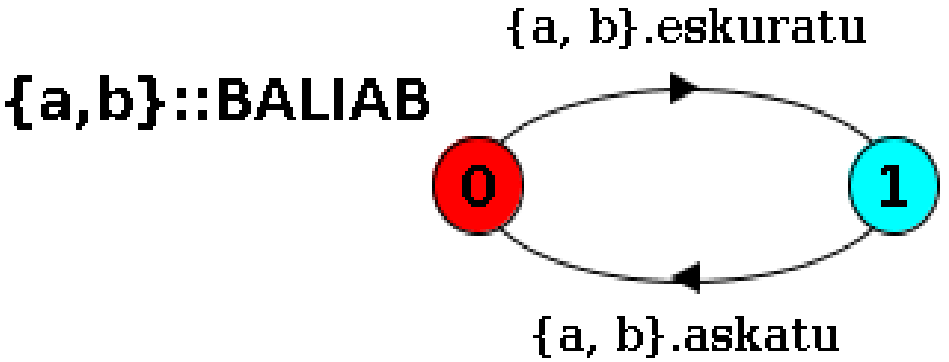
```
range Bool = 0..1

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).
```

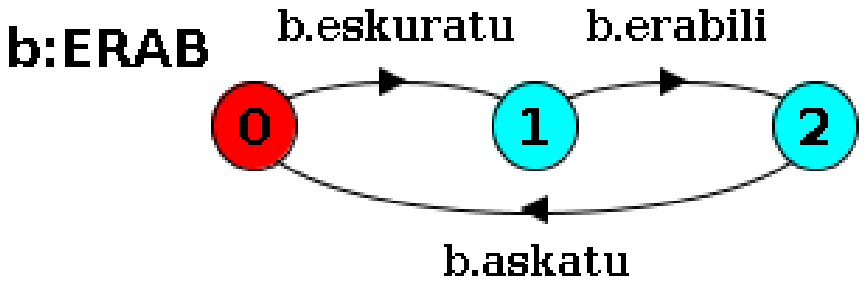
Edo baita modu honetan ere:

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

Prozesuen aurrizki-etiketak konpartituriko baliabidetarako.

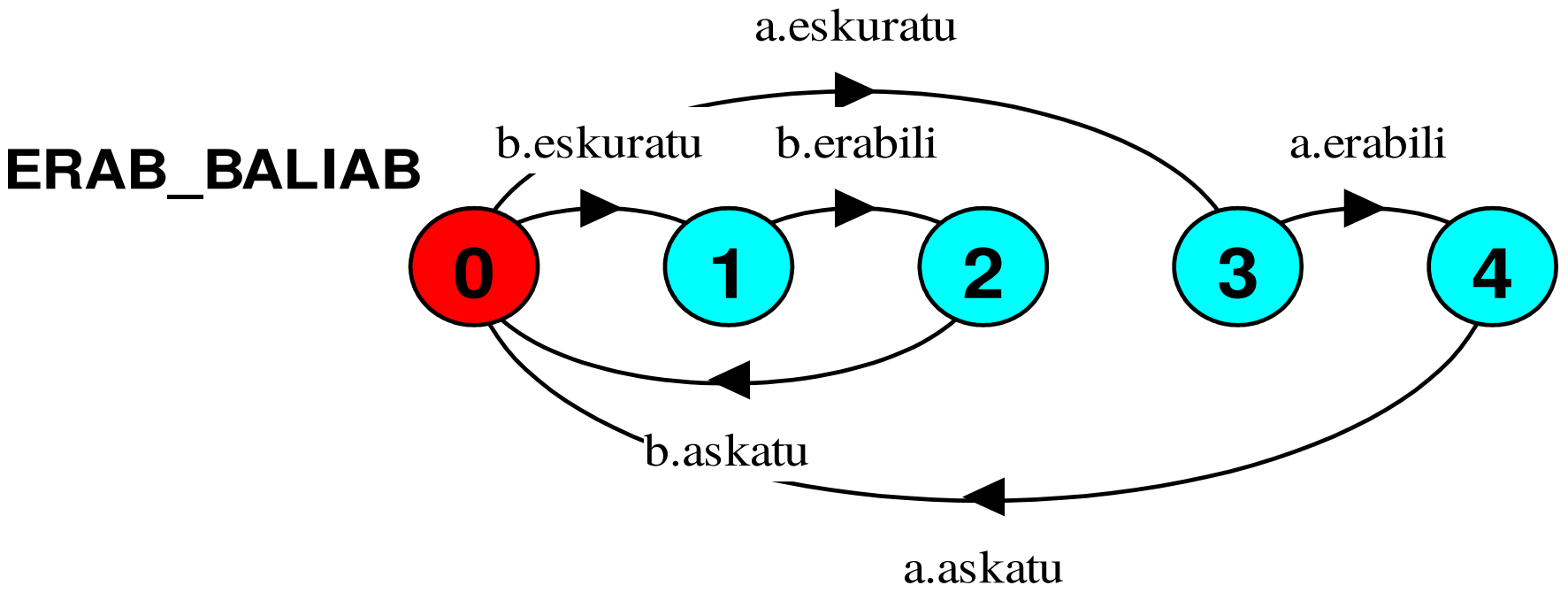






Nola ziurtatzen du ereduak baliabidea eskuratzen duen erabiltzailea dela askatuko duena?

Prozesuen aurrizki-etiketak konpartituriko baliabidetarako.



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 berretiketatze-funtzioak aplikatuko dizkiegu, ekintzen etiketen izenak aldatzeko.
Berretiketatze-funtzioen itxura orokorra hau da:

/{*etikberria*_1/*etikzaharra*_1,… *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 berretiketatuko genuke ziurtatzeko prozesu konposatuak sinkronizatzen direla ekintza zehatzetan:

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

Ekintzak ezkutatzea (*hiding*)

konplexutasuna gutxiagotzeko abstrakzioa.

\{a1..ax} ezkutatze-eragilea
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.

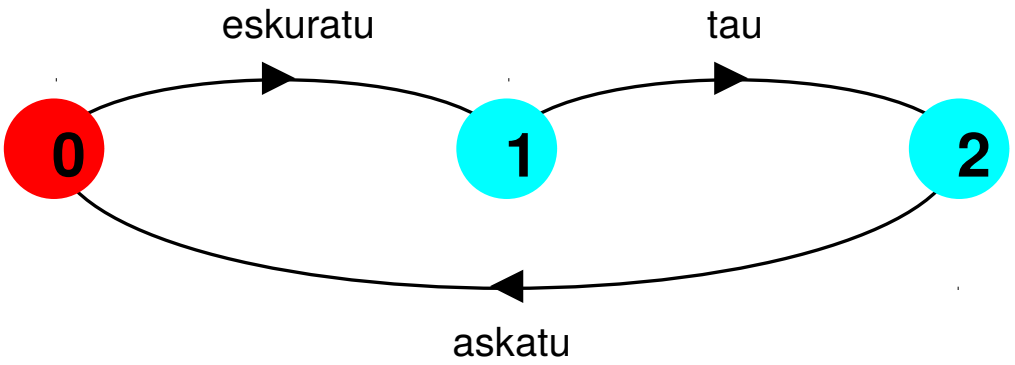
Batzuetan egokiagoa da erakutsiko diren etiketen multzoa adieraztea:

@{a1..ax} interfaze-eragilea
P prozesuari aplikatzen diogunean, a1..ax multzoan ez dauden P alfabetoko ekintzak ezkutatzen dira.

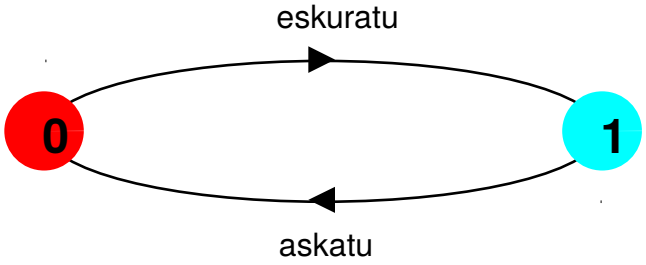
Ekintzak ezkutatzea

Ondoko definizioak baliokideak dira:

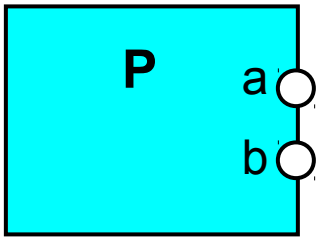
```
ERABILTZAILEA =
   (eskuratu->erabili->askatu->ERABILTZAILEA)
   \{erabili}.

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



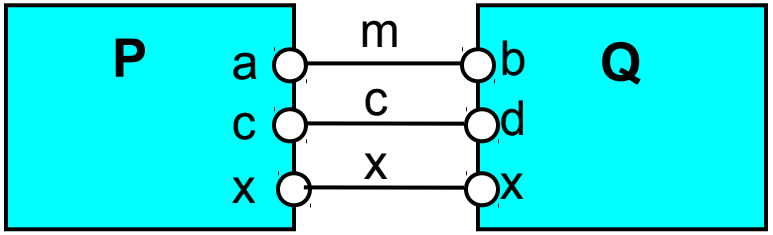
Minimizazioak **tau** ekintza ezkutatuak ezabatzen ditu portaera baliokidea duen LTS bat sortuz.



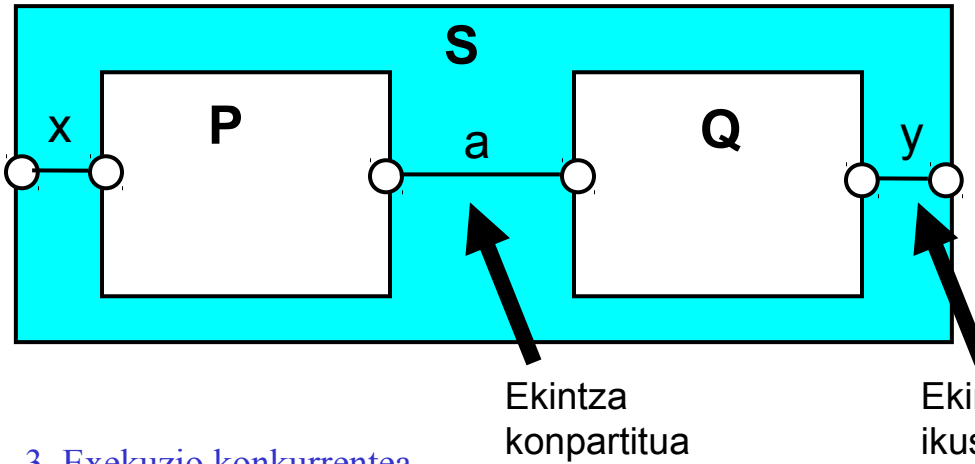
Egitura-diagramak



{a,b} alfabetoa
duen P prozesua.



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



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

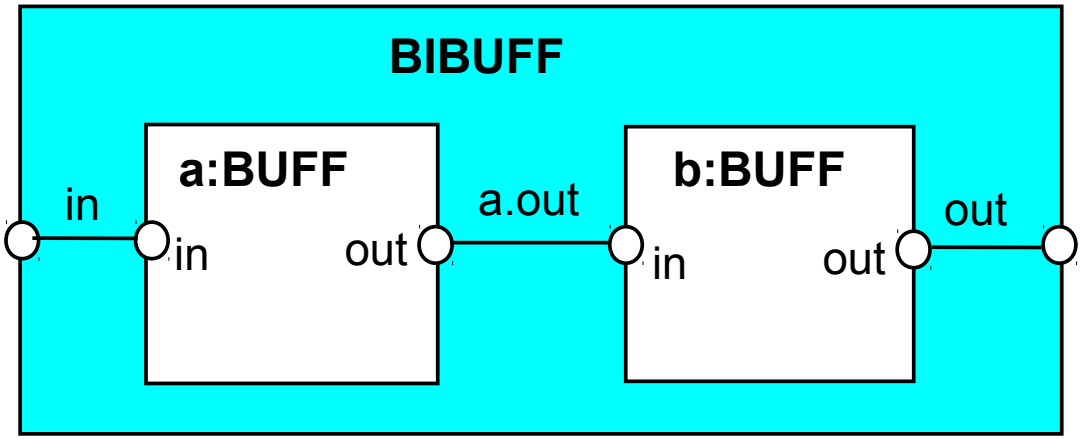
Egitura-diagramak

Egitura-diagramak eredu baten egitura estatikoa adierazteko erabiltzen dira.

- Egoera-makinek ez baitute egitura estatikoa harrapatzen.

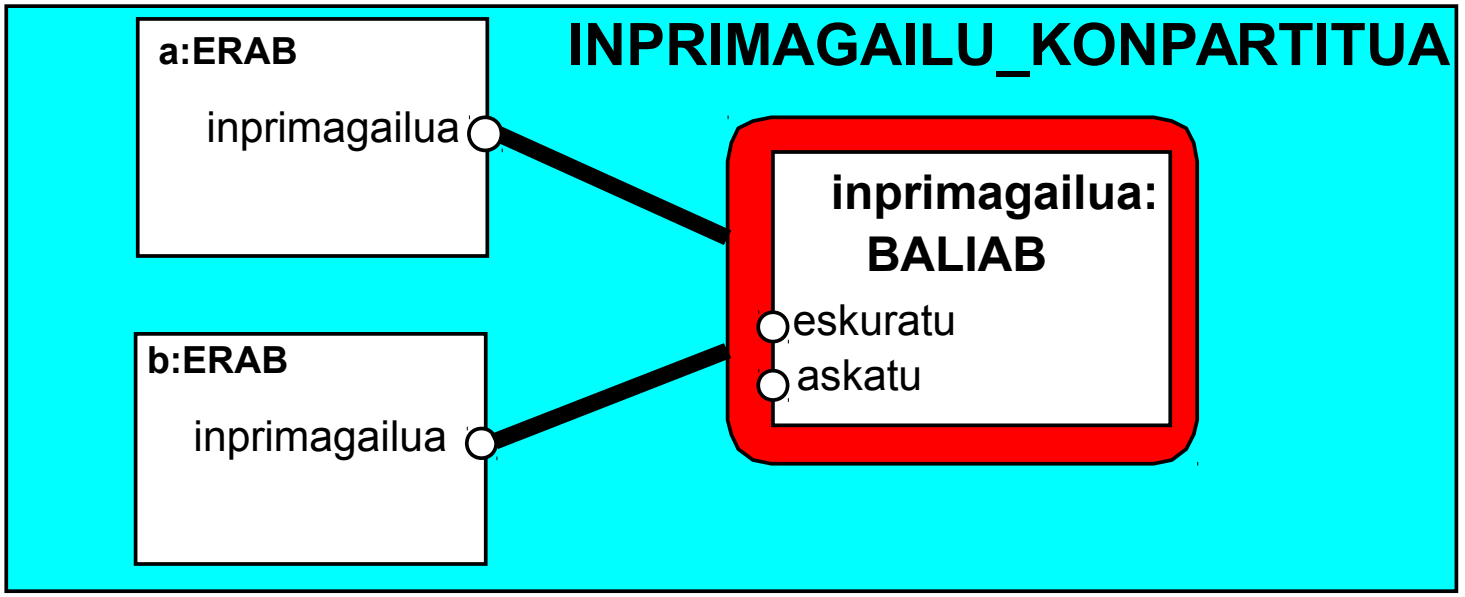
Egitura estatikoa ondoko eragileek deskribatzen dute: konposizio paraleloa, berretiketatzea eta ezkutatzea.

```
range T = 0..3
BUFF = (in[i:T]->out[i]->BUFF).
```




```
||BIBUFF=(a:BUFF||b:BUFF)
          /{in   /a.in,
            a.out/b.in,
            out  /b.out}
          @{in,out}.
```

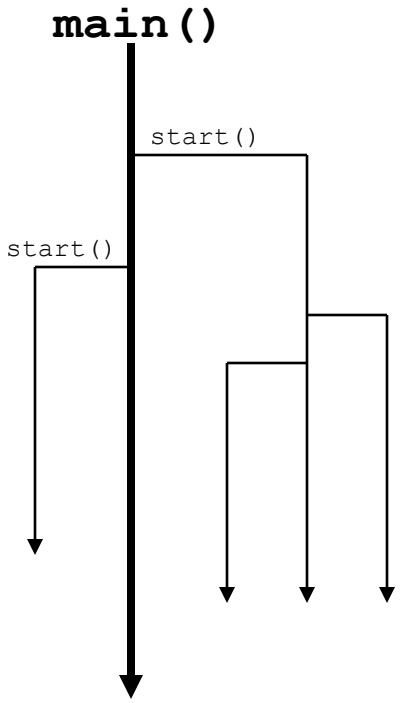
Egitura-diagramak – baliabideak konpartitzea



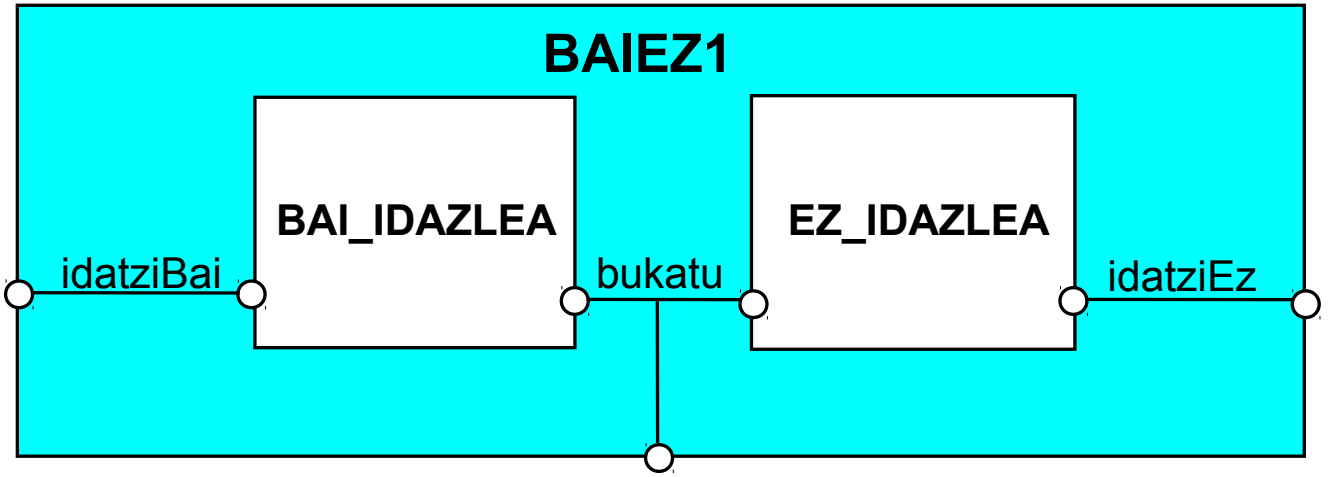

```
ERAB = (inprimagailua.eskuratu->erabili
                              ->inprimagailua.askatu->ERAB).
BALIAB           = BALIAB[0],
BALIAB[i:0..1]   =( when (i==0) eskuratu -> BALIAB[1]
                  | when (i==1) askatu   -> BALIAB[0] ).

||INPRIMAGAILU_KONPARTITUA = ( {a,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 inplementazioa

```
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 inplementazioa

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

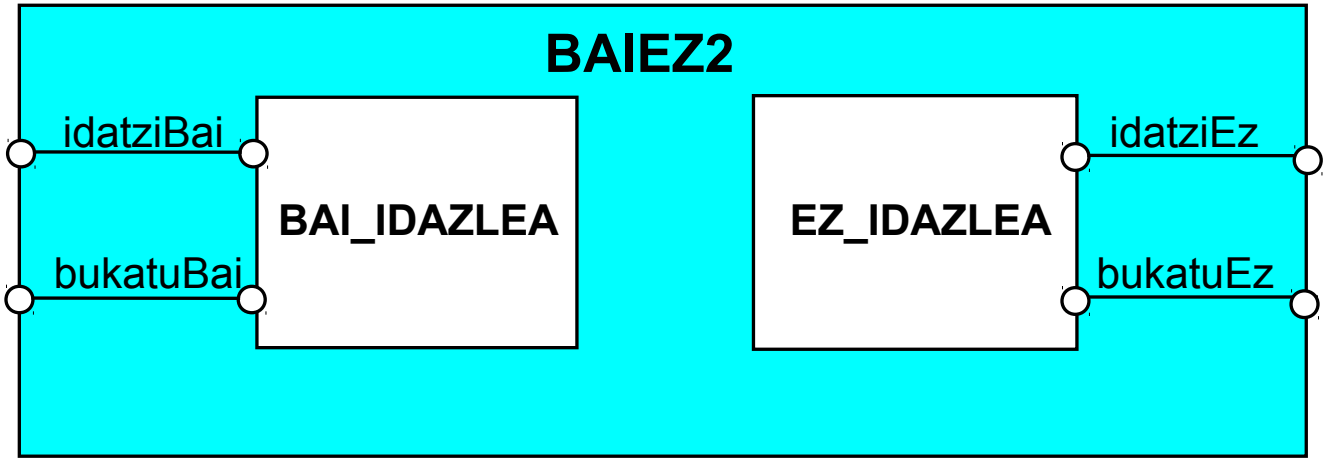
Hari berriak sortu

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



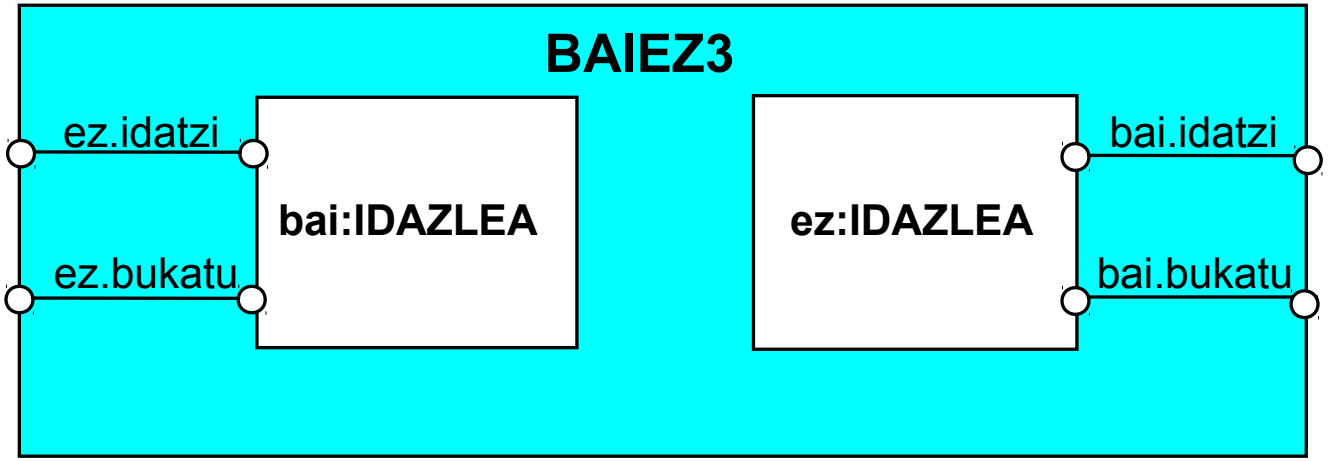

```
BAI_IDAZLEA = (idatziBai -> BAI_IDAZLEA
              |bukatuBai -> STOP).

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

||BAIEZ2 = (BAI_IDAZLEA||EZ_IDAZLEA).
```

Baiez2 inplementazioa

```
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 inplementazioa (*Idazlea* klase bakar batekin)

```
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 inplementazioa

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

Klase bereko bi instantzia definitu eta konkurrenteki exekutatzen dira.

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

Baiez4 inplementazioa (idazketak *Pantaila* klasean)

```
class Idazlea extends Thread {
        private String baiEz;
        private int denbora, kopurua;
        private Pantaila p;
        public Idazlea(String s, int i, int j, Pantaila pant){
                baiEz=s; denbora=i; kopurua=j;
                p=pant;
        }
        public void run()  {
            int i;
            for (i=0;i<kopurua;i++) {
                p.margotu(baiEz);
                try {sleep(denbora);}
                catch (InterruptedException e) {}
            }
            p.margotu(baiEz+" bukatu da");
        }
}

class Pantaila {
        public void margotu(String s){
                System.out.println(s);
        }
}
```

Baiez4 inplementazioa

```
class BaiEz4App{
        public static void main (String args[]) {
                Pantaila p = new Pantaila();
                Idazlea bai = new Idazlea("bai",1000,4,p);
                Idazlea ez  = new Idazlea("\tez",400,15,p);
                bai.start();
                ez.start();
        }
}
```

Klase bereko bi instantzia definitu eta konkurrenteki exekutatzen dira.

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

Ariketak

5. **BaiEz4** adibidearen Java inplementazioa egokitu,
   bi hari sortu beharrean, hiru sortzeko:
   "bai", "ez" eta "agian"

```
bai
        ez
                agian
        ez
        ez
```

```
bai
bai
bai
        ez
bai
bai
        ez
bai
        ez
bai
        ez
bai
        ez
        ez
bai
        ez
        ez
        ez
```

6. **BaiEz4** adibidearen Java inplementazioa egokitu,
   exekuzioan zehar harien abiadura aldatzeko.
   Hasieran "bai" azkar idazten du eta bukaeran poliki,
   eta "ez" hasieran poliki eta gero azkar.

7. Ondoko programa inplementatu JAVA erabiliz:
   Bi kotxe (bi hari) izango ditugu.
   Kotxeak "+" eta "*" ikurrekin adieraziko dira
   eta ezkerretik eskuinera mugituko dira, bakoitza abiadura desberdinean.

   Pantailaren "garbiketa" egiteko,ondokoa erabil daiteke:

```
for(int i=0; i<24; i++){System.out.println();}
```

8. Aurreko ariketa egokitu, exekuzioan zehar kotxeen abiadura aldatzen joateko.

Ariketak

9. Irakasleak inplementatutako kotxeen ariketaren bertsioa (irudiekin) egelatik jeitsi eta 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.
Prozesu aktiboak (eta hariak) ateak izango dira. Suposatu ahal dugu ate bakoitzean dagoen sensoreak jakinarazten duela norbait sartzean.
FSP eredua eta Java programa egin.

11*. Nahi dituzun hobekuntzak egin 9. ariketan:
- fondo bat jarriz, kotxe gehiago ipiniz, kotxeek aurrera eta atzera eginez, kotxeez gain beste aktore batzuk agertuz, mugimenduak ez soilik horizontalean izanik...
- hasieran interfazeak eskeintzen du aukera, erabiltzaileak sartzeko kotxe kopurua, mugimendu motak, azelerazio parametroak...
- botoi batzuen edo teklen bidez, kotxeak azeleratzeko eta desazeleratzeko, edota aurrera joateaz gain, kotxeak gora eta behera egiteko...

12**. Inplementatu zuk asmatutako programa bat hariak eta irudiak erabiliz.