10. előadás

Párhuzamos programozási kérdések.
Az Ada taszkok (1).

Párhuzamos programozás

- Program: több tevékenység egyidőben
- Önálló folyamatok
- Együttműködnek
- Többféle lehetőség
  - Elosztott rendszer (distributed system)
  - Többszálú program (multithreaded program)
  - …
- Célok
  - Hatékonyság növelése
  - A feladat egyszerűbb megoldása

Elosztott rendszer

- Több számítógép hálózatba szervezve
- A folyamatok elosztva a hálózaton
- Kliens-szerver programok
- Elosztott objektumrendszer (CORBA, DCOM)
- PVM, MPI, messaging rendszerek
- Ada 95: Partíciók
- Kommunikáció:
  - Távoli eljáráshívás (Remote Procedure Call)
  - Üzenetküldés

Megosztott memóriájú rendszer

- Shared memory system
- Szemben az elosztott (memóriájú) rendszerrel
- Egy gépen belül több processzor
- Egyszerűbb kommunikáció

Többszálú programok

- Egy processzen belül több végrehajtási szál
- Pehelysúlyú folyamatok (light-weight processzek) a heavy-weight processzben
- Operációs rendszerek is támogathatják
- Programozási nyelvi eszközök (Ada, Java)
- Egy tárterületen belül futnak
  - A processzhez rendelt memóriában
  - Megosztott memóriás modell
  - Saját végrehajtási verem

Mire jó a több szál

- Ha több tevékenységet kell végezni egyszerre
- Egymástól függetlenül fogalmazhatjuk meg ezeket
- Összehangolásra szükség van:
  - Szinkronizáció
  - Kommunikáció
- Példák
  - Több kliens kiszolgálása a szerverben
  - Webböngésző: a felhasználói felület kezelése + letöltés
  - Szimulációk

Problémák

- Elméleti nehézség (programhelyesség)
- Gyakorlati nehézség (nemdeterminisztikusság) tesztelés, nyomkövetés

- Ütemezés
- Interferencia
- Szinkronizáció
- Kommunikáció
- Holtpont és kiéheztetés

Ada taszkok

- A többszálú programozáshoz nyelvi eszköz
- Gazdag lehetőségek

```
task A;
task body A is
begin
        for I in 1..10000 loop
                Put_Line("Szia!");
        end loop;
end A;
```

Taszk programegység

- Specifikáció és törzs
  - Különválik (mint a csomag és a sablon)

    task A; task body A is … end A;
  - Specifikáció: hogyan lehet vele kommunikálni
  - Törzs: hogyan működik
- Nem lehet könyvtári egység
  (be kell ágyazni más programegységbe)

Kétszálú program

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Kétszálú is
        task Egyik;
        task body Egyik is
        begin
                loop
                        Put_Line("Szia!");
                end loop;
        end Egyik;
begin
        loop
                Put_Line("Viszlát!");
        end loop;
end Kétszálú;
```

Lehetséges kimenetek

Szia!	Szia!	Szia!	Szia!	SziViszla!
Viszlát!	Szia!	Szia!	Szia!	át!
Szia!	Szia!	Szia!	Szia!	Szia!
Viszlát!	Szia!	Viszlát!	Szia!	Szia!
Szia!	Viszlát!	Szia!	Szia!	Viszlát!
Viszlát!	Viszlát!	Szia!	Szia!	ViszSzia!
Szia!	Viszlát!	Viszlát!	Szia!	láSzit!a!
Viszlát!	Viszlát!	Viszlát!	Szia!	
Szia!	Szia!	Viszlát!	Szia!	Szia!
Viszlát!	Szia!	Viszlát!	Szia!	Szia!

Jellemző kimenet

- Sokszor az egyik (például Viszlát!)
- Utána sokszor a másik
- Nem ugyanannyiszor
- Változik futás közben, hogy melyikből mennyi
- Futásról futásra is változik

Ütemezés

- A futtató rendszer ütemezi a szálakat
  - Vagy az oprendszer, vagy az Ada rendszer
  - Co-rutinok: programozó ütemez (Simula 67)
- Az Ada nyelv nem tesz megkötést arra, hogy egy adott Ada implementációban az ütemezés milyen algoritmussal történik
- Jó program: bármely ütemezésre jó

Időosztásos ütemezés

- Nem valódi párhuzamosság
- Egy processzor: egyszerre csak egyet
- Időszeletek (time slicing)
- Gyorsan váltogatunk, párhuzamosság látszata
- Egyenlő időszeletek: pártatlanság
- Kooperatív viselkedés

Kompetitív viselkedés

- „Fusson, amíg van mit csinálnia”
- Amíg számol, addig övé a vezérlés
- Vezérlés elvétele: csak speciális pontokon (blokkolódás, szinkronizáció)
- Hatékonyabb, mint az időszeletes ütemezés
  - Kevesebb kontextusváltás (context switch)
- Nem pártatlan:
  egy taszk kisajátíthatja az erőforrásokat

Prioritás

- A magasabb prioritású szálak előnyben
- Ada: Real-Time Systems Annex
- Set_Priority, Get_Priority alprogramok

Pártatlanság, kooperáció

- Egy taszk lemondhat a vezérlésről
- delay utasítás

```
loop
        Put_Line("Szia!");
        delay 0.0;
end loop;
```

- Jellemző kimenet: felváltva írnak ki

Várakozás

- Egy időre felfüggeszthető a taszk
- A delay utasítással: **delay 2.4;**
- A delay után Duration típusú érték van
  - Predefinit valós fixpontos típus
  - A másodperc megfelelője
- Legalább ennyi ideig nem fut a taszk
- Szimulációkban gyakran használjuk
- Használjuk szinkronizációs utasításokban

Taszk elindulása és megállása

```
procedure P is
        task T;
        task body T is … begin  … end T;
begin
        …
end P;
```

itt indul a T

P itt bevárja T-t

Szülő egység

- Az a taszk / alprogram / könyvtári csomag / blokk, amelyben deklaráltuk
  - (!)
- Elindulás: a szülő deklarációs részének kiértékelése után, a szülő első utasítása előtt
- A szülő nem ér véget, amíg a gyerek véget nem ér
- Függőségi kapcsolatok

Taszk befejeződése

- Bonyolult szabályok
- Fogalmak: komplett, abortált, terminált
- Például:
  - elfogytak az utasításai (akár kivétel miatt)
  - és a tőle függő taszkok már terminálták
- T'Callable
- T'Terminated

Ha több egyforma szál kell

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Háromszálú is
        task Egyik;
        task body Egyik is
        begin
                loop    Put_Line("Szia!");      end loop;
        end Egyik;
        task Másik;
        task body Másik is … begin … end Másik;
begin
        loop    Put_Line("Viszlát!");   end loop;
end Háromszálú;
```

Taszk típussal

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Háromszálú is
        task type Üdvözlő;
        task body Üdvözlő is
        begin
                loop    Put_Line("Szia!");      end loop;
        end Üdvözlő;
        Egyik, Másik: Üdvözlő;
begin
        loop    Put_Line("Viszlát!");   end loop;
end Háromszálú;
```

Taszk típus

- Taszkok létrehozásához
  - Ugyanazt csinálják, ugyanolyan utasítássorozatot hajtanak végre
  - Saját példánnyal rendelkeznek a lokális változókból
- Korlátozott (limited) típusok
- Cím szerint átadandó, ha alprogram-paraméter
- Programegységek
  - Mint a taszk programegységek
  - Specifikáció és törzs különválik
  - Nem lehet könyvtári egység (be kell ágyazni más programegységbe)

Taszk típus diszkriminánsai

- Mint a rekordok esetében
- Lehet a diszkrimináns mutató típusú is
- Taszkok létrehozásakor aktuális

```
task type T ( Id: Integer );
task body T is … begin … Put(Id); … end T;
X: T(13);
```

Mutató típusú diszkrimináns

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Háromszálú is
        type PString is access String;
        task type Üdvözlő ( Szöveg: PString );
        task body Üdvözlő is
        begin
                loop    Put_Line(Szöveg.all);    end loop;
        end Üdvözlő;
        Egyik: Üdvözlő(new String'("Szia!"));
        Másik: Üdvözlő(new String'("Hello!"));
begin
        loop    Put_Line("Viszlát!");    end loop;
end Háromszálú;
```

Taszkok létrehozása allokátorral

- Taszk típusra mutató típus:

  task type Üdvözlő (Szöveg: PString);
  type Üdvözlő_Access is access Üdvözlő;
  P: Üdvözlő_Access;

- A mutató típus gyűjtőtípusa egy taszk típus
- Az utasítások között:

  P := new Üdvözlő(new String'("Szia!"));

Taszk elindulása és megállása

```
procedure Fő is
    task type Üdvözlő ( Szöveg: PString );
    type Üdvözlő_Access is access Üdvözlő;
    P: Üdvözlő_Access;
    task body Üdvözlő is … begin  … end;
begin
    …
    P := new Üdvözlő( new String’("Szia!") );
    …
end Fő;
```

itt indul

Fő itt bevárja

Szülő egység

- Az a taszk / alprogram / könyvtári csomag / blokk,
  - amelyben deklaráltuk
  - amely a mutató típust deklarálta
- Elindulás:
  - a szülő deklarációs részének kiértékelése után, a szülő első utasítása előtt
  - az allokátor kiértékelésekor
- A szülő nem ér véget, amíg a gyerek véget nem ér
- Függőségi kapcsolatok, befejeződés

Komplett taszk

- Ha véget ért a törzs utasításainak végrehajtása
  - normálisan
  - kivételes eseménnyel
    - és nincs kivételkezelő rész
    - vagy van, de nincs megfelelő ág benne
    - vagy van, és a kivételkezelő ág lefutott
- Ha a taszk elindítása során kivétel lépett fel

Abnormális állapotú taszk

- Az abort utasítással „kilőhető” egy taszk
  - akár magát is kilőheti: öngyilkos
- Nem túl szép módja egy taszk leállításának

```
task type T( Id: Integer );
task body T is ... end T;
X: T(1); Y: T(2);
begin
…
abort X, Y;
```

Egy taszk terminál, ha

- komplett, és az összes tőle függő taszk terminált már
- abnormális állapotba jutott és törlődött a várakozási sorokból
- **terminate** utasításhoz ért, és a taszk olyan programegységtől függ, amely már komplett, és a leszármazottai terminálták már, komplettek, vagy szintén **terminate** utasításnál várakoznak

Szülő terminálása

- Egy szülő addig nem terminálhat, amíg az általa létrehozott taszkok nem terminálnak
- Miután befejezte saját utasításainak végrehajtását (azaz kompletté vált), megvárja, hogy a tőle függő taszkok is terminálják (az **end**-nél várakozik)

Információcsere taszkok között

- Nonlokális (globális) változókon keresztül
  - Nem szeretjük...
  - Biztonságosabbá tehető különféle pragmák segítségével (Atomic és Volatile)
- **Randevúval**
  - Ezt fogjuk sokat gyakorolni
  - Aszimmetrikus, szinkron, pont-pont, kétirányú kommunikációt tesz lehetővé
- Védett egységek használatával

Nonlokális változón keresztül

```
procedure P is
        N: Natural := 100;
        task type T;
        task body T is
        begin
                … if  N > 0 then N := N-1; … end if; …
        end T;
        A, B: T;
begin … end P;
```

legfeljebb N-szer szabadna

Interferencia

- Ha a taszkok önmagukban jók
- De együttes viselkedésük rossz

- Szinkronizáció segíthet

Randevú

- Szinkronizációhoz és kommunikációhoz
- Egyik taszk:
  belépési pont (**entry**) és **accept** utasítás
- Másik taszk: meghívjuk a belépési pontot

- Aszimmetrikus
- Szinkron
- Pont-pont
- Kétirányú

Fiú és lány taszkok

```
task Lány is
      entry Randi;
end Lány;

task body Lány is
begin
      …
      accept Randi;
      …
end Lány;
```

```
task Fiú;

task body Fiú is
begin
      …
      Lány.Randi;
      …
end Fiú;
```

Fiú és lány taszkok

```
task Lány is
      entry Randi;
end Lány;

task body Lány is
begin
      …
      accept Randi;
      …
end Lány;
```

```
task Fiú is end Fiú;

task body Fiú is
begin
      …
      Lány.Randi;
      …
end Fiú;
```

Pont-pont kapcsolat

- Egy randevúban mindig két taszk vesz részt
- Egy fiú és egy lány
- Szerepek
  - Egy taszk lehet egyszer fiú, másszor lány
- Nincs "broad-cast" jellegű randevú

Aszimmetrikus

- Megkülönböztetjük a hívót és a hívottat (fiú és lány)
- Egész másként működik a két fél
- A szintaxisban is kimutatható a különbség
- A hívó ismeri a hívottat, de fordítva nem

Az **accept** törzse

- A randevú az accept utasítás végrehajtásából áll
- Ez alatt a két taszk „együtt van”
- A fogadó taszk hajtja végre az **accept**-et

- Az **accept**-nek törzse is lehet, egy utasítássorozat

Huzamosabb randevú

```
task Lány is
      entry Randi;
end Lány;

task body Lány is
begin
      …
      accept Randi do
            …
      end;
      …
end Lány;
```

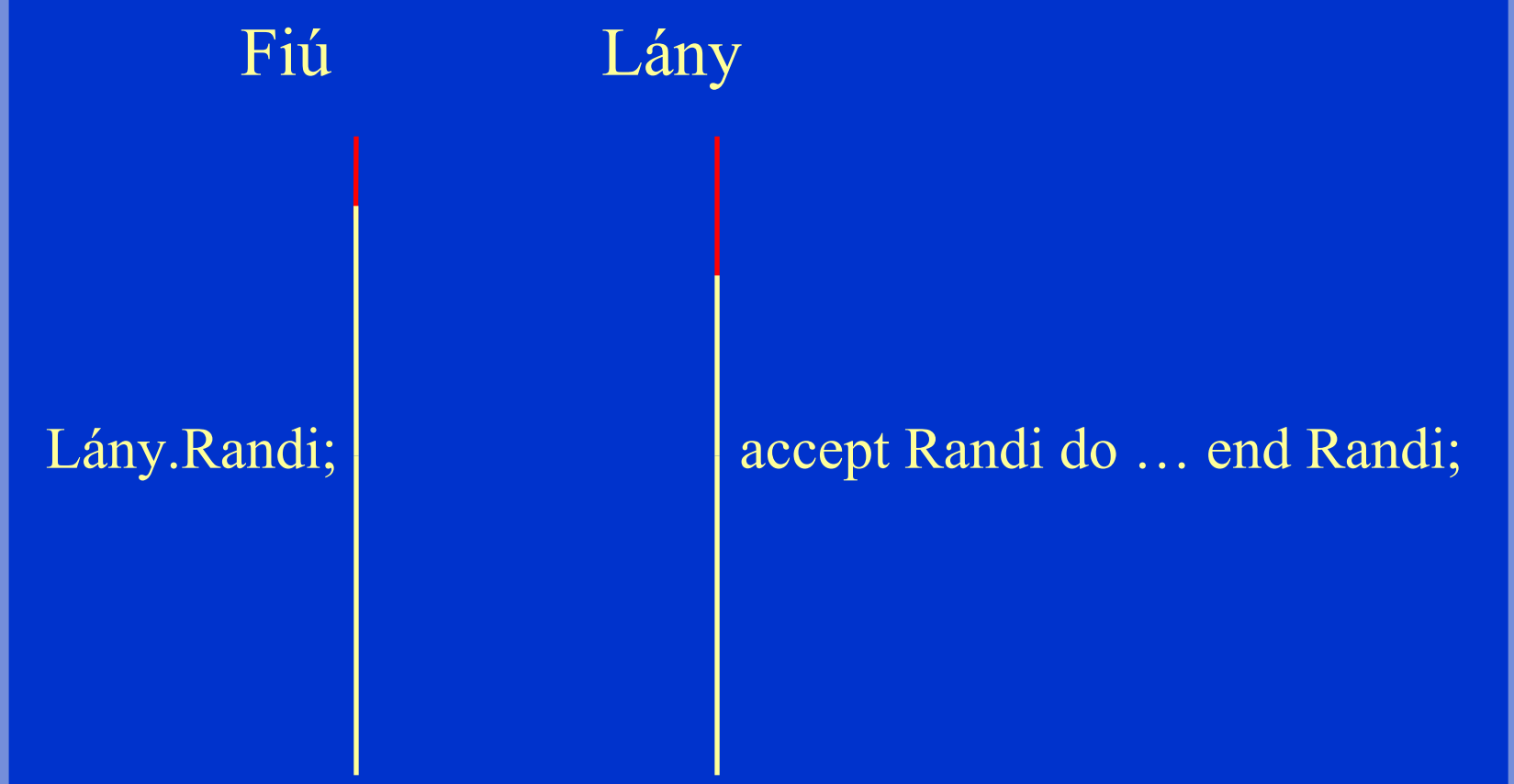
```
task Fiú;

task body Fiú is
begin
      …
      Lány.Randi;
      …
end Fiú;
```

Szinkronitás

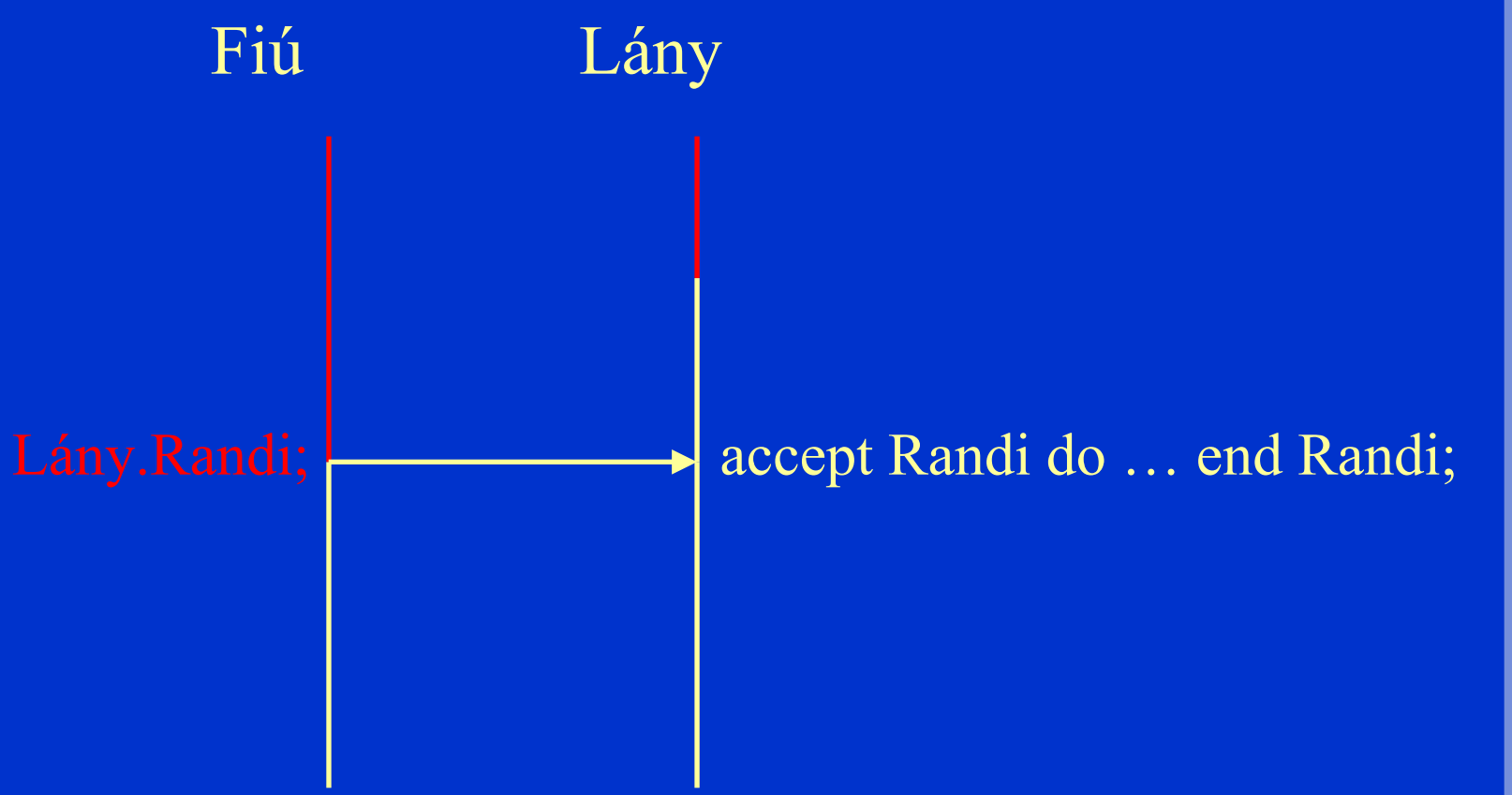
- A randevú akkor jön létre, amikor mindketten akarják
- Bevárják egymást a folyamatok az információcseréhez
- Az aszinkron kommunikációt le kell programozni, ha szükség van rá

A fiú bevárja a lányt (1)

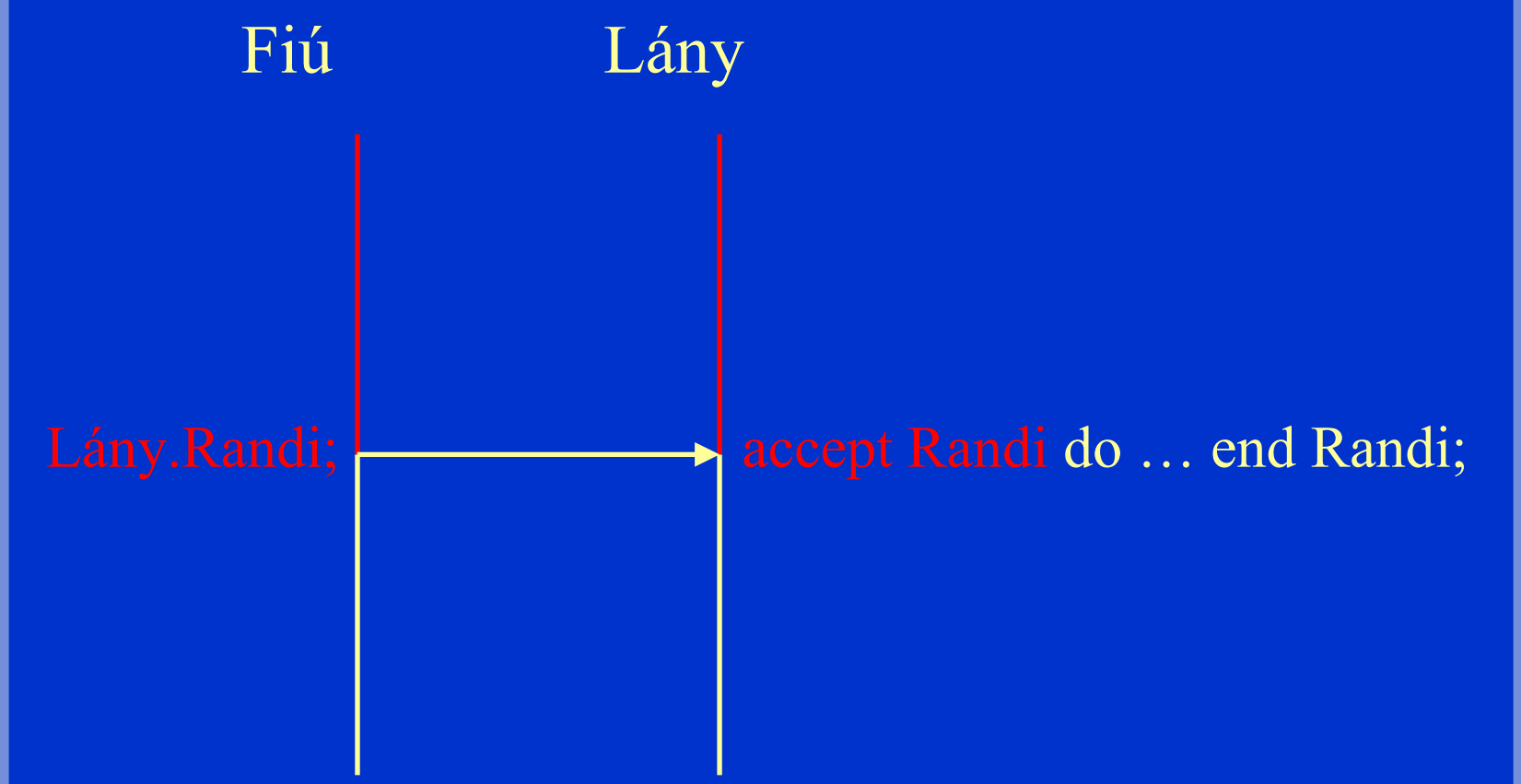


A fiú bevárja a lányt (2)

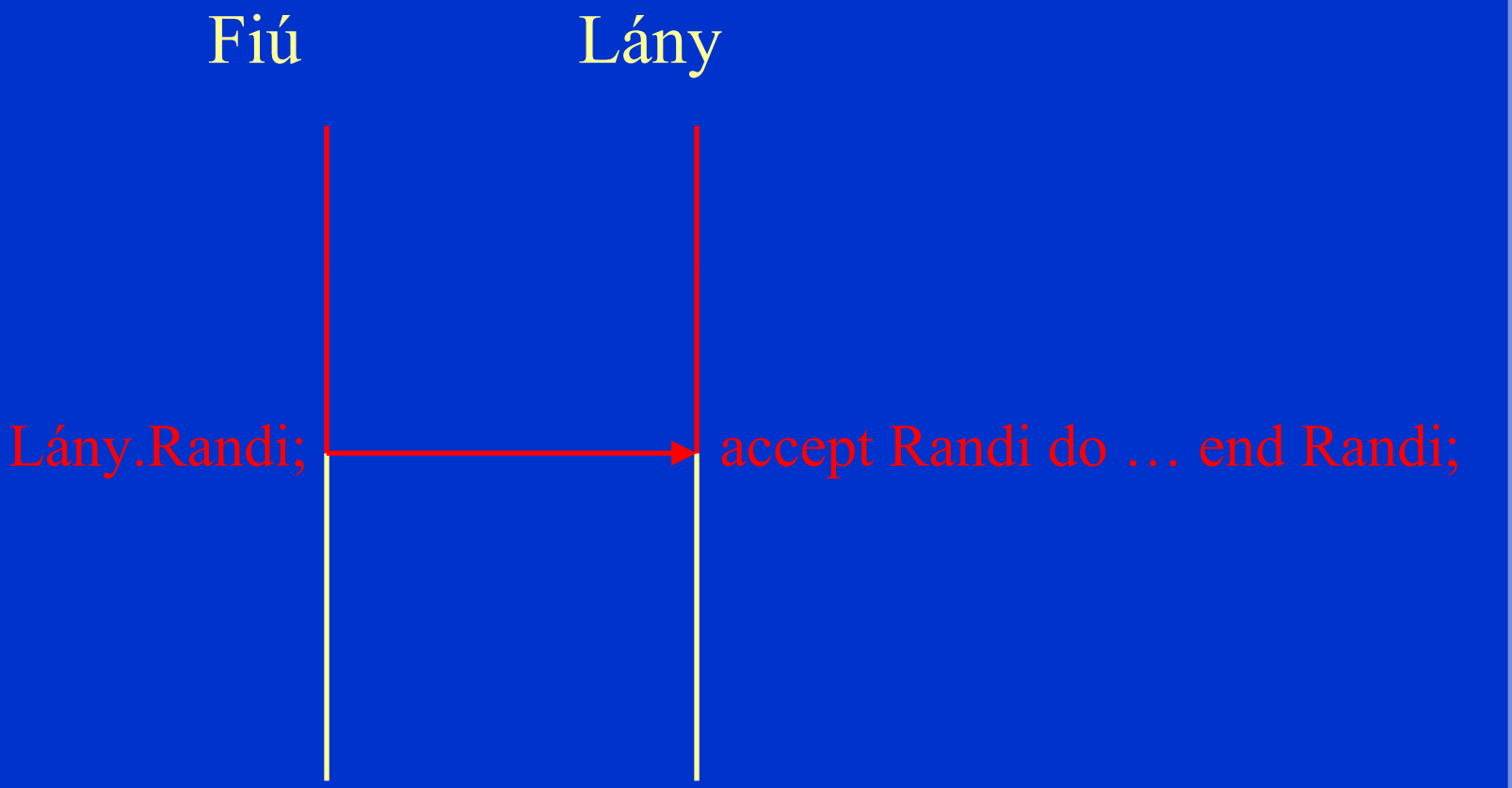
A fiú bevárja a lányt (3)



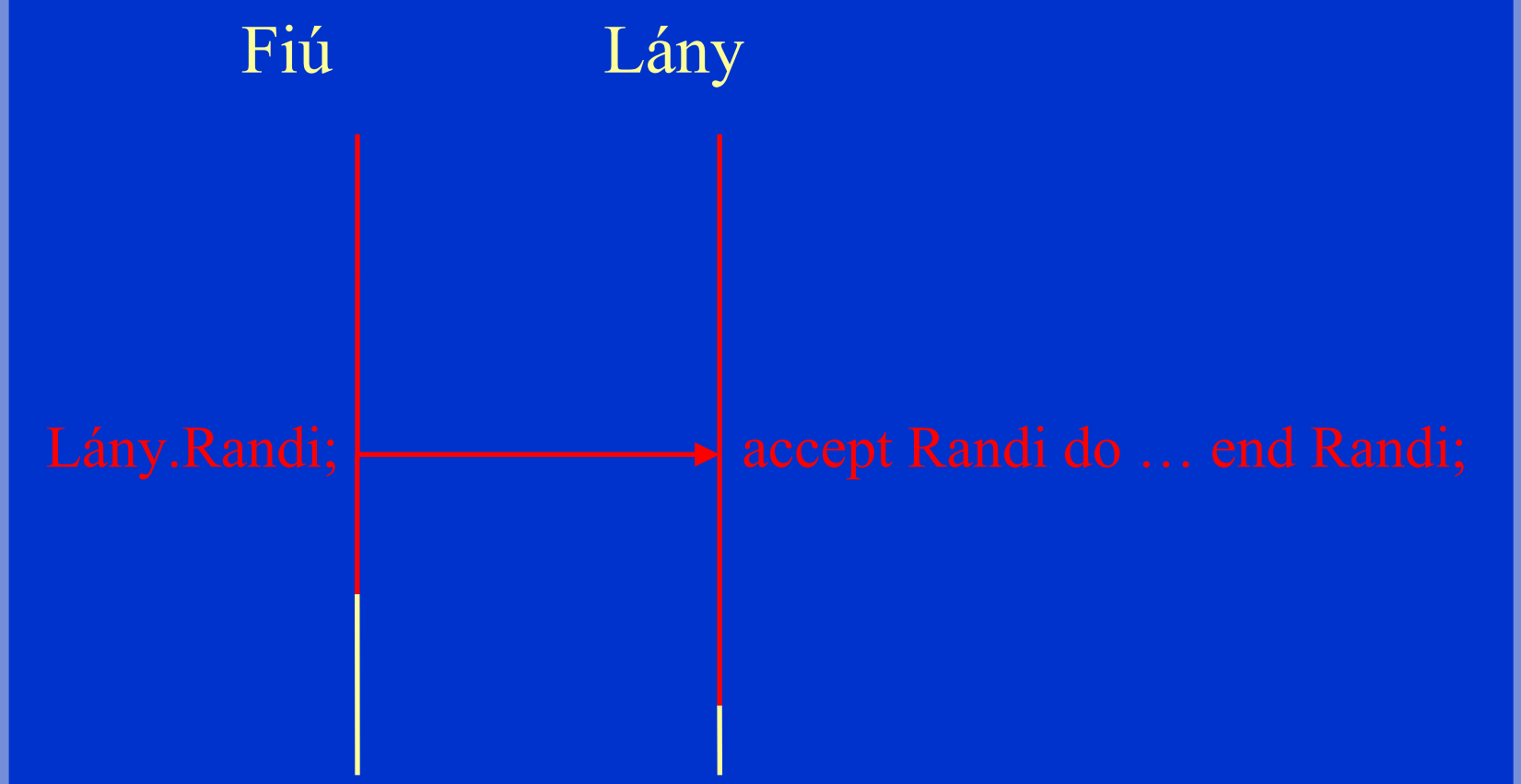
A fiú bevárja a lányt (4)



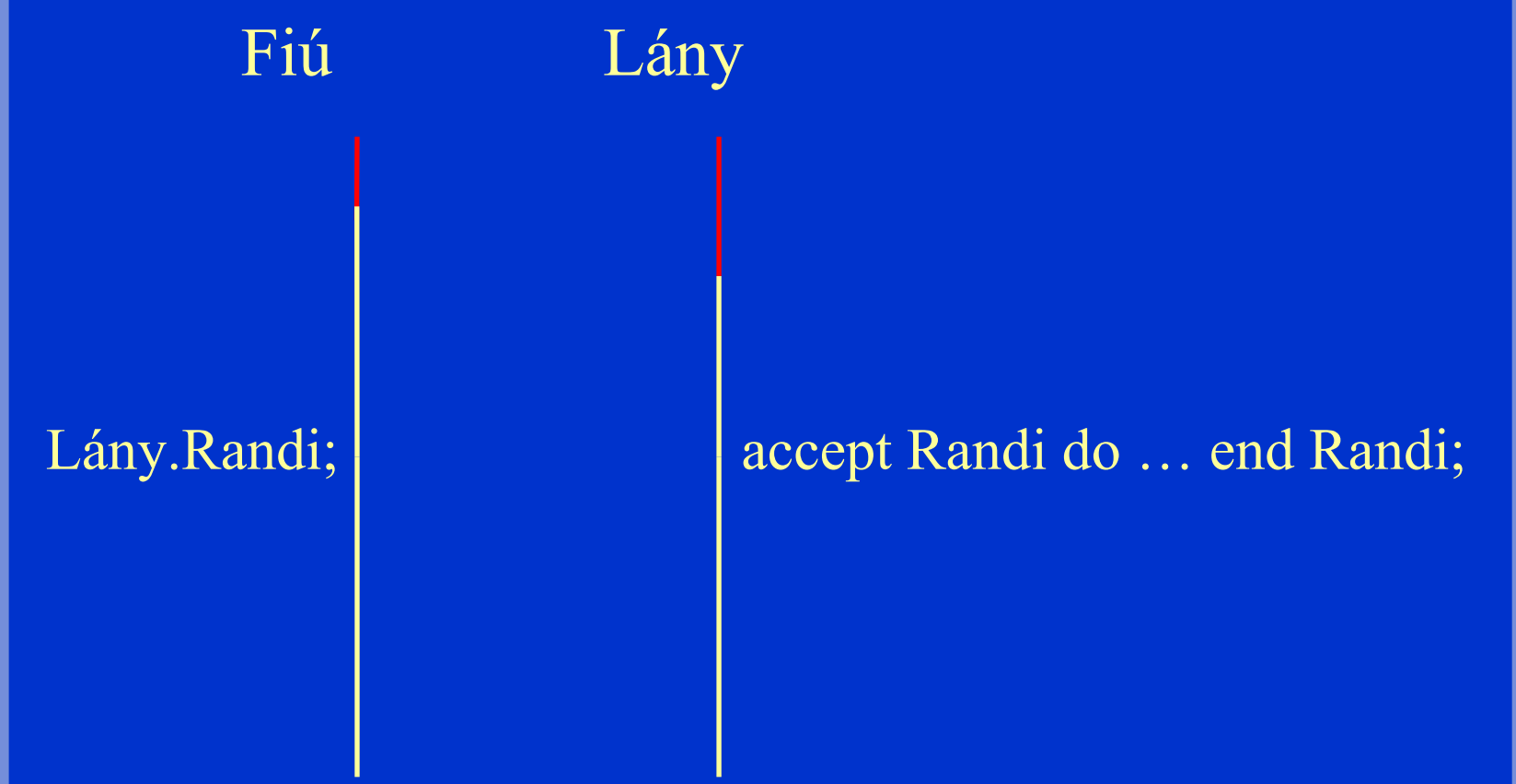
A fiú bevárja a lányt (5)



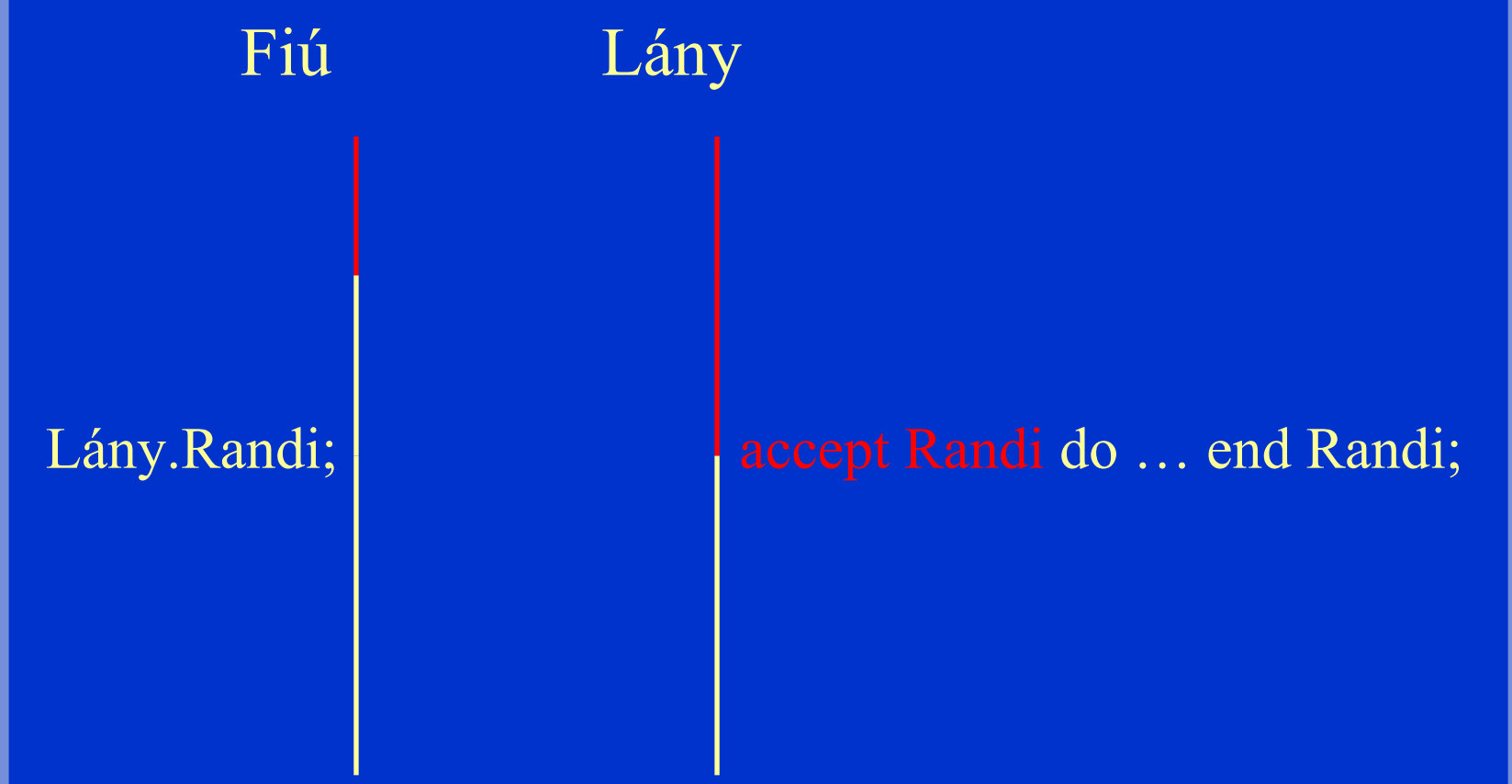
A fiú bevárja a lányt (6)



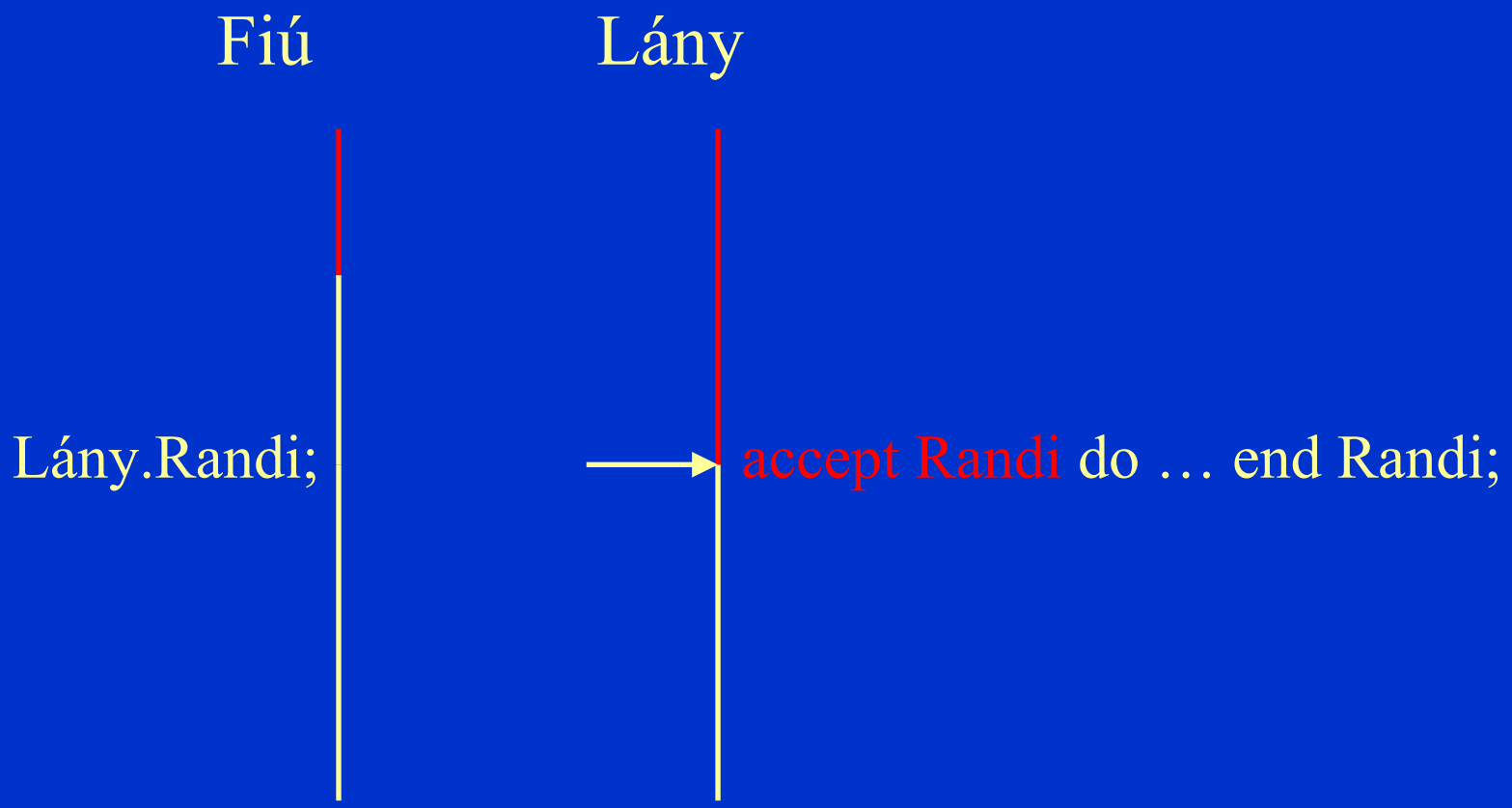
A lány bevárja a fiút (1)



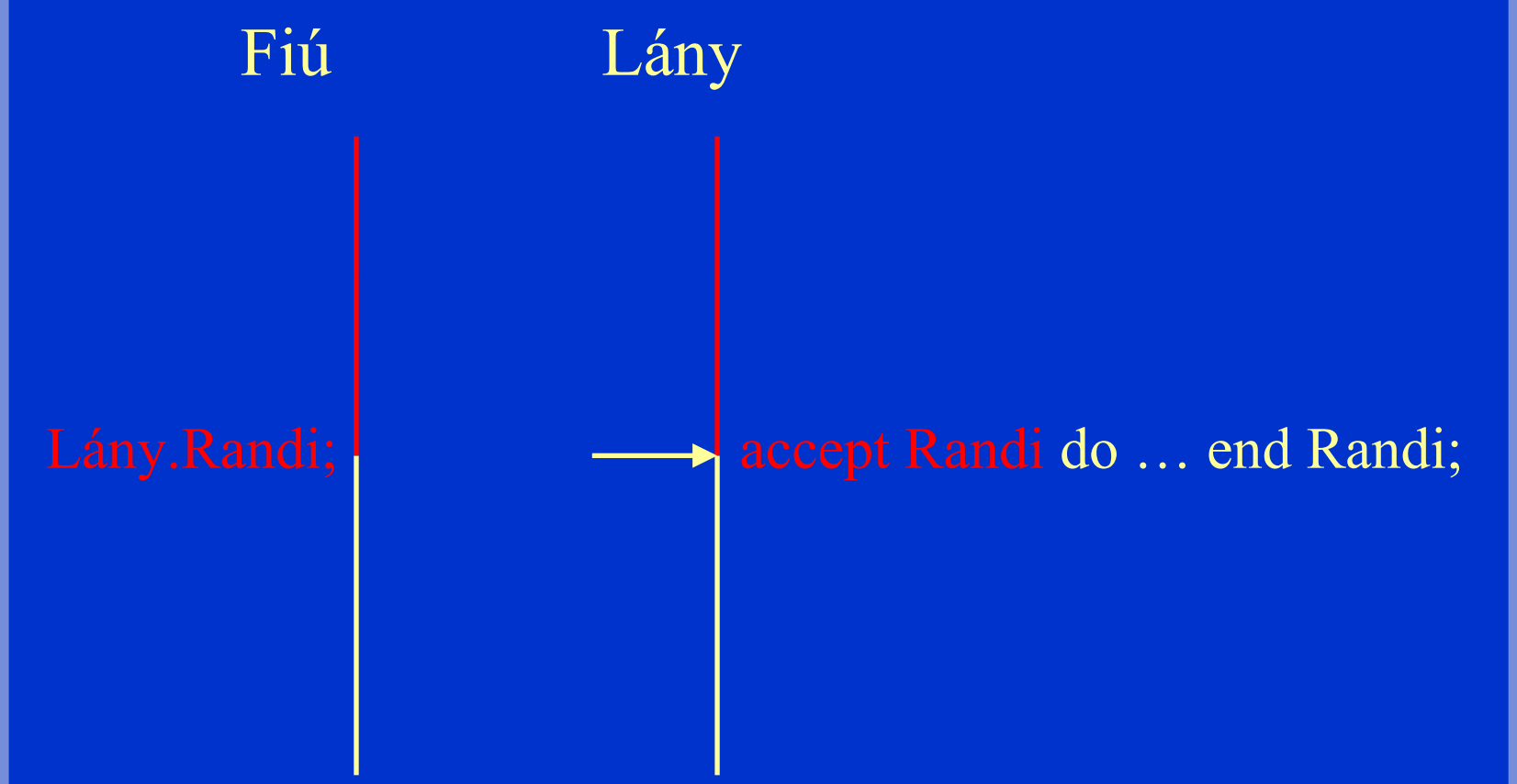
A lány bevárja a fiút (2)



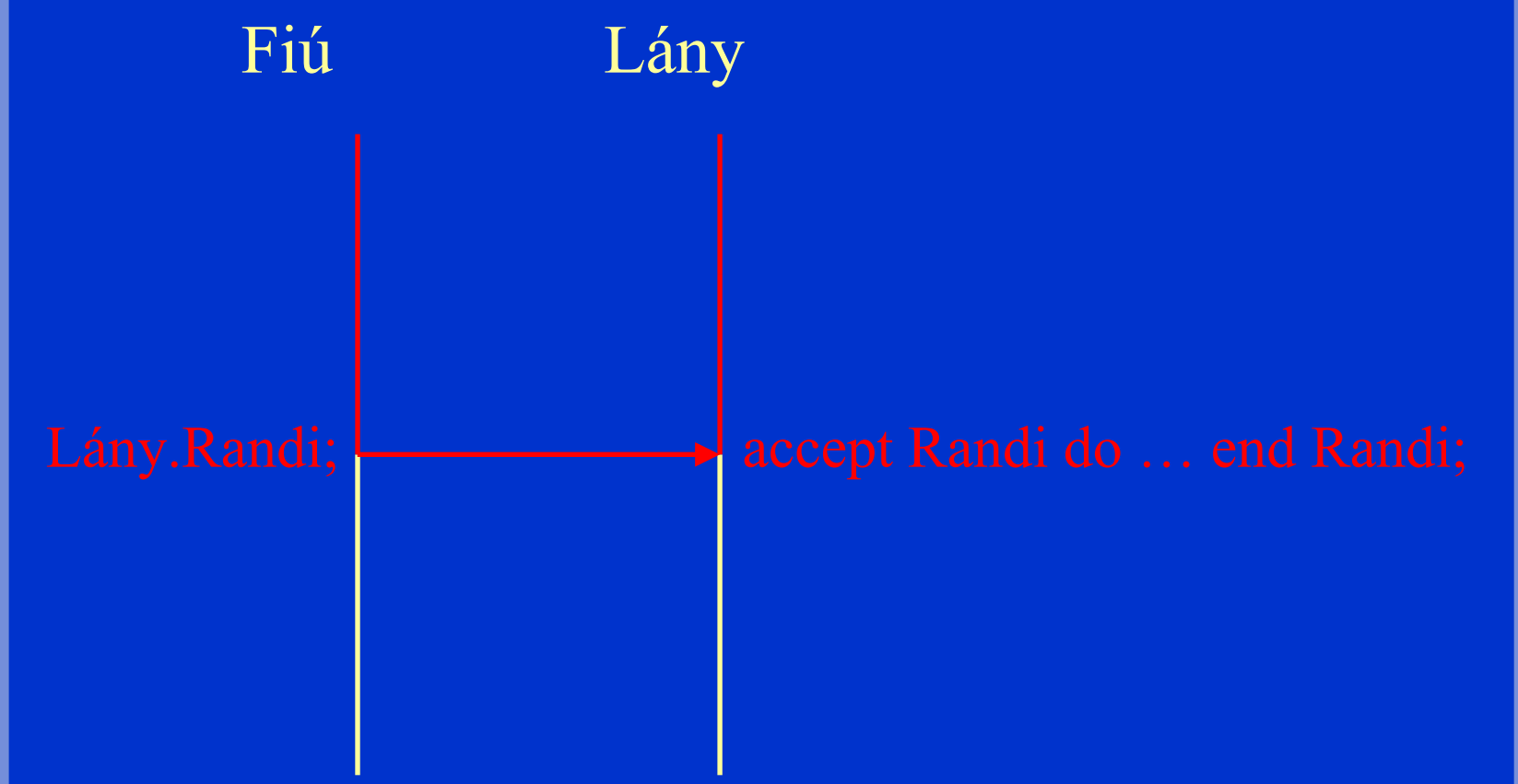
A lány bevárja a fiút (3)



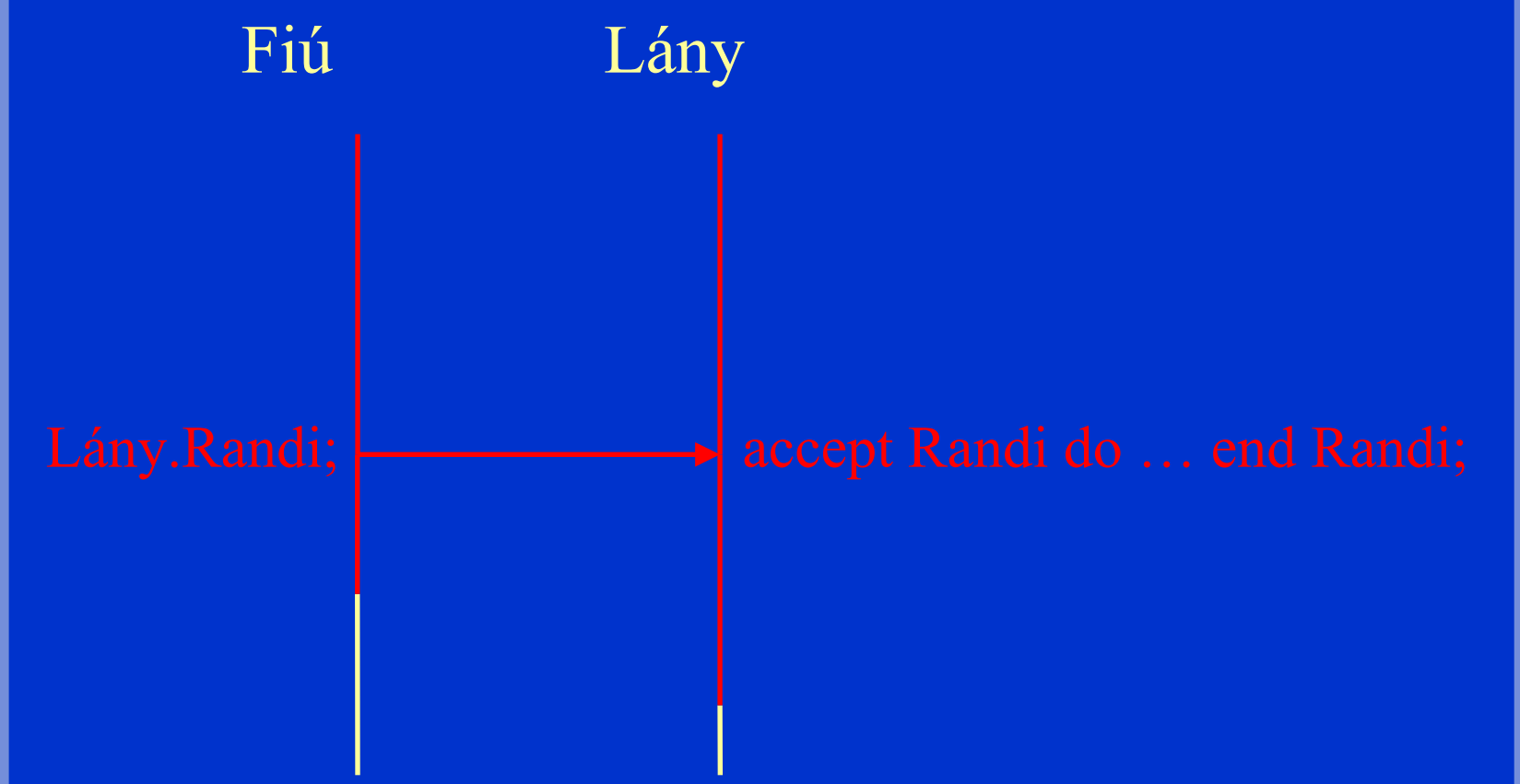
A lány bevárja a fiút (4)



A lány bevárja a fiút (5)



A lány bevárja a fiút (6)



Kommunikáció

- A randevúban információt cserélhetnek a taszkok
- Paramétereken keresztül
  - Entry: formális paraméterek
  - Hívás: aktuális paraméterek
- A kommunikáció kétirányú lehet, a paraméterek módja szerint (in, out, in out)
- Az alprogramok hívására hasonlít a mechanizmus (fontos különbségek!)

Várakozási sor

- Egy entry-re több fiú is várhat egyszerre
- De a lány egyszerre csak eggyel randevúzik
- Egy entry nem olyan, mint egy alprogram, mert nem „re-entráns”
- A hívó fiúk bekerülnek egy várakozási sorba
- Minden entry-nek saját várakozási sora van
- Ha a hívott lány egy accept utasításhoz ér, a legrégebben várakozóval randevúzik (vagy, ha nincs várakozó, akkor maga kezd várni)

A belépési pontok

- Minden belépési ponthoz tartoznia kell legalább egy accept utasításnak a törzsben
- Akár több accept utasítás is lehet a törzs különböző pontjain elhelyezve
- (Egy hívóban több entry-hívás is lehet)
- Az **entry** várakozási sorának aktuális hossza: ‘**Count** attribútum

```
Randi'Count
```

Több belépési pont is lehet

```
task Tároló is
        entry Betesz( C: in Character );
        entry Kivesz( C: out Character );
end Tároló;
```

Egyelemű buffer

```
task body Tároló is
        Ch: Character;
begin
        loop
                accept Betesz( C: in Character ) do
                        Ch := C;
                end Betesz;
                accept Kivesz( C: out Character ) do
                        C := Ch;
                end Kivesz;
        end loop;
end Tároló;
```

Így használhatjuk

```
task body T is
        Ch: Character;
begin
        ...
        Get(Ch);
        Tároló.Betesz(Ch);
        ...
        Tároló.Kivesz(Ch);
        Put(Ch);
        ...
end T;
```

Aszinkron kommunikáció

```
task body A is
        Ch: Character;
begin
        ...
   Get(Ch);
   Tároló.Betesz(Ch);
        ...
end A;
```

```
task body B is
        Ch: Character;
begin
        ...
   Tároló.Kivesz(Ch);
   Put(Ch);
        ...
end B;
```