

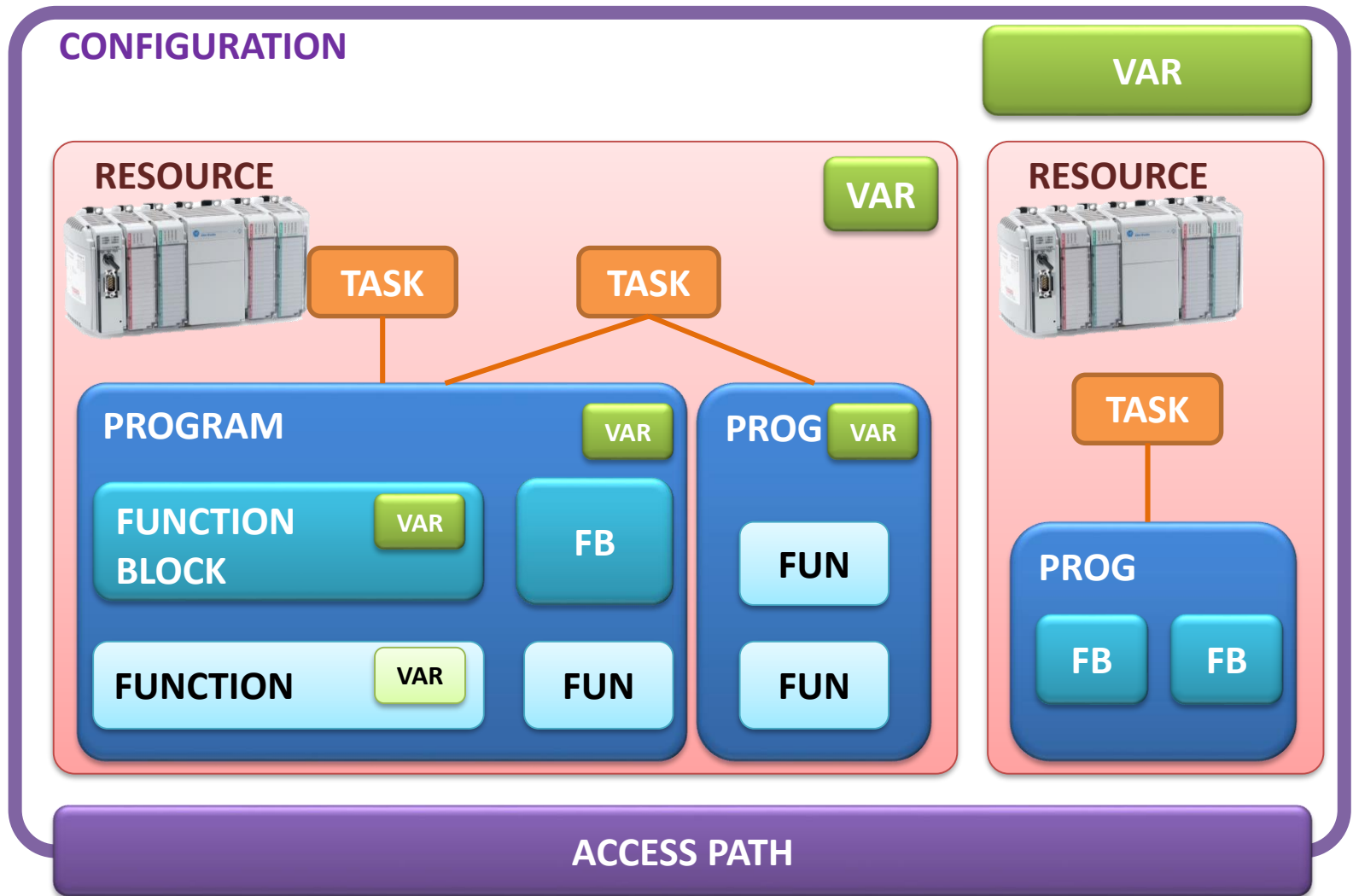
# Az IEC 61131-3 szabvány programozási nyelvei

## Strukturált szöveg

Programozható irányítóberendezések  
és szenzorrendszerek

KOVÁCS Gábor  
gkovacs@iit.bme.hu

# Áttekintés



# Programszervezési egységek

## POU típus és név

### Deklarációs rész:

- Interfész változók
- Lokális változók
- Globális változók

### POU törzs: programkód

- Ladder Diagram (LD)
- Instruction List (IL)
- Function Block Diagram (FBD)
- Structured Text (ST)
- Sequential Function Chart (SFC)

**PROGRAM** prog\_name

PROGRAM ConveyorControl

**FUNCTION\_BLOCK** fb\_name

FUNCTION\_BLOCK Pusher

**FUNCTION** fun\_name : DataType

FUNCTION IsReady : BOOL

# Strukturált szöveg (Structured Text, ST)

- Magas szintű szöveges nyelv
- Világos felépítés
- Hatékony programszervezési módok
- A gépi kódra fordítás nem tartható kézben közvetlenül
- A magas absztrakciós szint szuboptimális implementációhoz vezethet



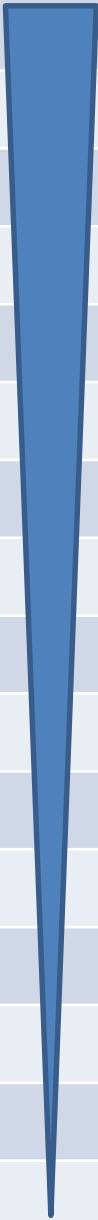
**A teljes szöveges kód végrehajtható ciklusonként**

# Kifejezések (expression)

- A műveletek a kifejezések eredményét dolgozzák fel
- Egy kifejezés elemei
  - Operandusok – akár más kifejezések
  - Operátorok

# Operandusok

- Literálisok
  - 17, 'my string', T#3s
- Változók (elemiek vagy származtatottak)
  - Var1, VarArray[12]
- Függvények visszatérési értékei
  - Add(2,3), sin(1.76)
- Más kifejezések
  - 10+20 (=Add(10,20))

Operátor	Leírás	Példa → Eredmény	Prioritás
( )	Zárójel: végrehajtási sorrendre hat	$(3+2)*(4+1) \rightarrow 25$	
<fcn name>	Függvényhívás	<code>CONCAT('PL','C') → 'PLC'</code>	
-	Ellentett (aritmetikai)	$-10 \rightarrow -10$ ( $-1 \times 10$ )	
NOT	Komplement (logikai negálás)	<code>NOT TRUE → FALSE</code>	
**	Hatványozás	$2^{**}7 \rightarrow 128$	
*	Szorzás	$2*7 \rightarrow 14$	
/	Osztás	$30/6 \rightarrow 5$	
MOD	Maradékképzés (modulo)	$32 \text{ MOD } 6 \rightarrow 2$	
+	Összeadás	$32+6 \rightarrow 38$	
-	Kivonás	$32-6 \rightarrow 26$	
<, <=, >, >=	Összehasonlítás	$32<6 \rightarrow \text{FALSE}$	
=	Egyenlőség	<code>T#24h = T#1d → TRUE</code>	
<>	Egyenlőtlenség	$2<>5 \rightarrow \text{TRUE}$	
&, AND	Logikai ÉS	<code>TRUE AND FALSE → FALSE</code>	
XOR	Logikai kizáró vagy (XOR)	<code>TRUE XOR FALSE → TRUE</code>	
OR	Logikai VAGY	<code>TRUE OR FALSE → TRUE</code>	

# Függvényhívások

- Kifejezésekben: a kifejezés a függvény visszatérési értékét használja fel
- Formális hívás
  - Zárójelek között, paraméter-azonosítókkal
  - A paraméterek sorrendje tetszőleges
  - Elhagyott paraméter esetén a függvény annak kezdeti értékét használja
  - `LIMIT (MN:=0, MX:=10, IN:=7, Q=>VarOut)`
- Informális hívás
  - Közvetlen értékek meghatározott sorrendben
  - `LIMIT (0, 7, 10)`



# Műveletek (statement)

Kulcsszó	Művelettípus
<code>: =</code>	Értékadás
<code>&lt;FB name&gt; (parameters)</code>	FB hívás
<code>RETURN</code>	Visszatérés a hívó POU-ba
<code>IF</code>	Kiválasztás
<code>CASE</code>	Kiválasztás
<code>FOR</code>	Iteráció
<code>WHILE</code>	Iteráció
<code>REPEAT</code>	Iteráció
<code>EXIT</code>	Iteráció befejezése

# Értékadás

VAR

d: INT;

e: ARRAY [0..9] OF INT;

END\_VAR

- := operátor

- Értékadás

- Egyelemű változónak

d:=4;

- Tömb elemének

e[3] := d\*\*2;

- Adattípusok

- A bal és jobb oldal adattípusa kompatibilis

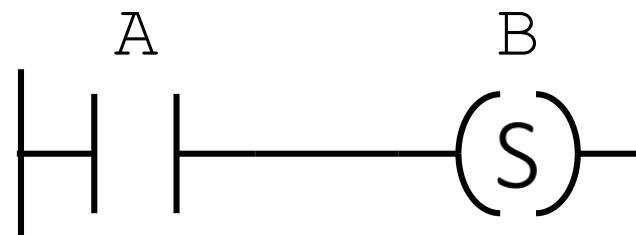
- A típuskonverziós függvények kifejezésként használhatók

d:=REAL\_TO\_INT(SIN(2))

# Bináris értékadás

- Nem a létradiagramban megszokott logikai függvény (kifejezés), hanem művelet
- Ha az értékadás nem hajtódik végre (pl. IF), akkor a változó megőrzi addigi értékét, nem íródik felül

```
IF A  
THEN B:=1;  
END_IF;
```



# FB hívás

- Az FB-hívás művelet, kifejezésben nem megengedett
- Formális hívás
  - Paraméter-azonosítókkal
  - A paraméterek sorrendje tetszőleges
  - Kihagyott paraméterek helyettesítése
    - Előző híváskori értékükkel
    - Első hívás esetén kezdeti értékükkel
- Informális hívás
  - Közvetlen értékek megfelelő sorrendben

# FB hívás - példa

```
PROGRAM MyProg
VAR
    MyTimer:    TON;
    A:          BOOL;
    MyInt :     INT;
END_VAR
(*...*)
MyTimer (PT:=T#1s, IN:=(MyInt=7), Q=>A);
MyTimer ((MyInt=7), T#1s);
A:=MyTimer.Q;
(*...*)
END_PROGRAM;
```

# Kiválasztás

- Kiválasztás logikai (Boolean) értékű kifejezés alapján
- Minden ágba tetszőleges számú műveletből álló blokk állhat
- ELSIF és ELSE ágak elhagyhatók
- END\_IF; használata kötelező

```
IF <BOOL expression> THEN
    <statement block>
ELSIF <BOOL expression> THEN
    <statement block>
ELSIF <BOOL expression> THEN
    <statement block>
ELSE
    <statement block>
END_IF;
```

# Kiválasztás - példa

```
IF (Pusher_Move=1) THEN
    MotorOut:=1;
    DirOut:=1;
ELSIF (Pusher_Move=2) THEN
    MotorOut:=1;
    DirOut:=0;
ELSE
    MotorOut:=0;
    DirOut:=0;
END_IF;
```

# Eset-kiválasztás

- Kiválasztás egész-értékű kifejezés alapján
- Az esetekhez több érték is megadható
- Alapértelmezett eset: ELSE (elhagyható)
- END\_CASE; nem hagyható el

```
CASE <INT expression> OF  
<value1>:           <statement block>  
<value2>,<value3>:  <statement block>  
ELSE   <statement block>  
END_CASE;
```



# Eset-kiválasztás - példa

```
CASE Pusher_Move OF
  1:  MotorOut:=1;
      DirOut:=1;
  2:  MotorOut:=1;
      DirOut:=0;
ELSE
      MotorOut:=0;
      DirOut:=0;
END_CASE;
```

# Iteráció

- Az iteráció egyetlen PLC-cikluson belül hajtódik végre
- Ha óvatlanul használjuk, akkor rontja a determinizmust és watchdog-hibát is okozhat
- Ne használjuk eseményre való várakozásra!
- Használhatjuk
  - Tömb vagy adatmező elemeinek vizsgálata
  - Egy művelet megadott számú ismétlésére

# While hurok

- A feltételes kifejezést a műveletek végrehajtása előtt vizsgálja
- Akkor hajtja végre a műveleteket, ha a feltételes kifejezés értéke TRUE

```
WHILE <BOOL expression> DO  
    <statement block>  
END_WHILE;
```

# While hurok - példa

VAR

MyArray: 1..10 OF INT;

i: INT;

MaxVal: INT:=0;

END\_VAR

(\* ... \*)

i:=1;

WHILE (i<=10) DO

IF (MyArray[i]>MaxVal)

THEN MaxVal:=MyArray[i];

END\_IF;

i:=i+1;

END\_WHILE;

(\* ... \*)

# Repeat – Until hurok

- A feltételes kifejezést a műveletek végrehajtása után vizsgálja (a műveleti blokk legalább egyszer végrehajtódik)
- Az iterációt a feltételes kifejezés TRUE értéke esetén fejezi be

```
REPEAT
    <statement block>
UNTIL <BOOL expression>
END_REPEAT;
```

# Repeat hurok - példa

VAR

MyArray: 1..10 OF INT;

i: INT;

MaxVal: INT:=0;

END\_VAR

(\* ... \*)

i:=0;

REPEAT

i:=i+1;

IF (MyArray[i]>MaxVal)

THEN MaxVal:=MyArray[i];

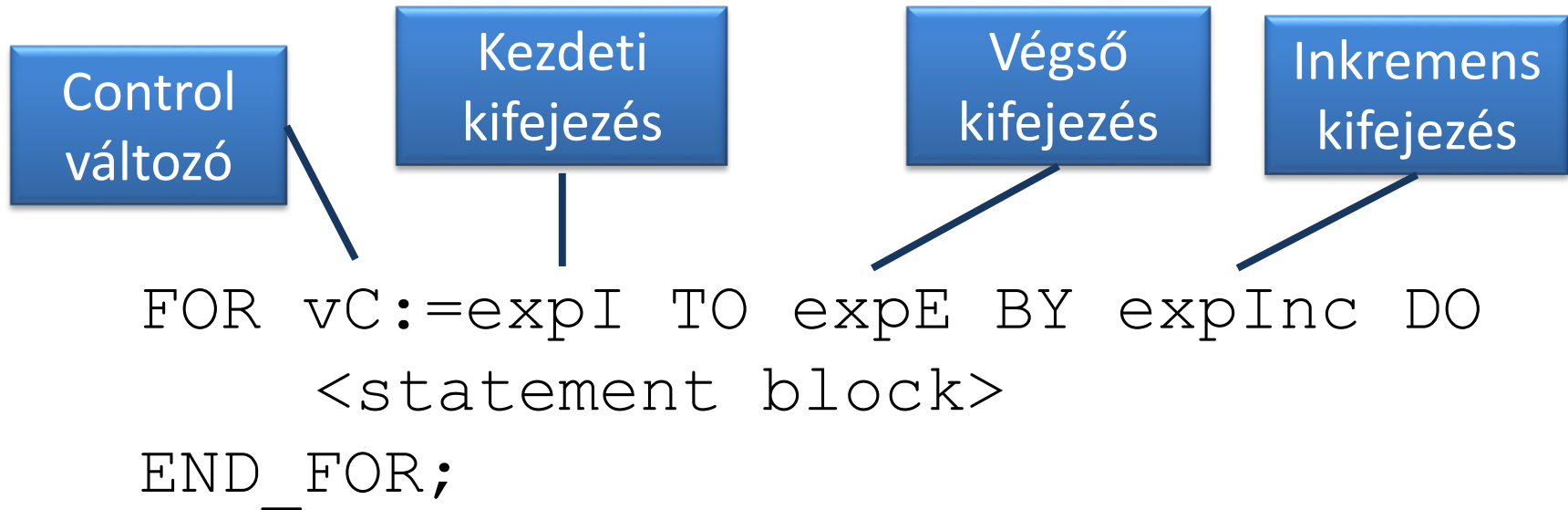
END\_IF;

UNTIL (i=10)

END\_REPEAT;

(\*...\*)

# For hurok



- A négy változó/kifejezés azonos adattípusú kell, hogy legyen (SINT, INT, DINT)
- A Control változónak, valamint a kezdeti és végkifejezésben szereplő változóknak nem adható érték a hurkon belül
- Az inkremens kifejezésben szereplő változónak a hurkon belül is adható érték (bár nem ajánlott)

# For hurok - példa

```
VAR
```

```
    MyArray:    1..10 OF INT;
```

```
    i:          INT;
```

```
    MaxVal:     INT:=0;
```

```
END_VAR
```

```
(* ... *)
```

```
FOR i:=10 TO 1 BY -1 DO
```

```
    IF (MyArray[i]>MaxVal)
```

```
        THEN MaxVal:=MyArray[i];
```

```
    END_IF;
```

```
END_FOR;
```

```
(* ... *)
```



# Kilépés hurkokból

- A hurkokból az EXIT művelettel lehet kilépni
- Csak abból a hurokból lép ki, amelyben végrehajtjuk, külsőbb szinten nem hat

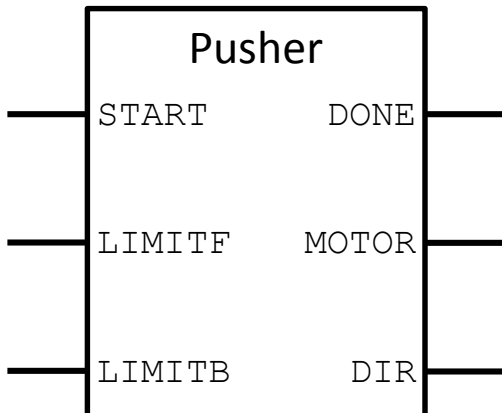
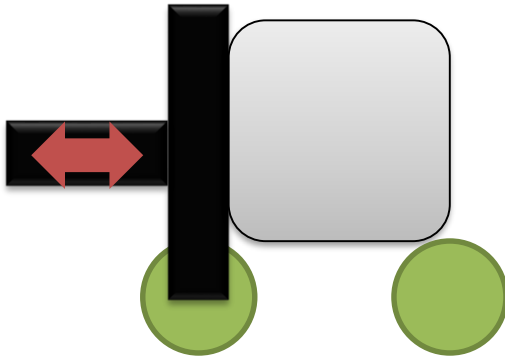
```
j := 0;  
WHILE (j < 10) DO  
    i := 0;  
    WHILE (i < 10) DO  
        IF (i = j) THEN EXIT;  
        ELSE i := i + 1;  
        END_IF;  
    END_WHILE;  
    j := j + 1;  
END_WHILE
```

i	j
0	0
1	0
1	1
2	0
2	1
2	2
3	0
...	

# Visszatérés a hívó POU-ba

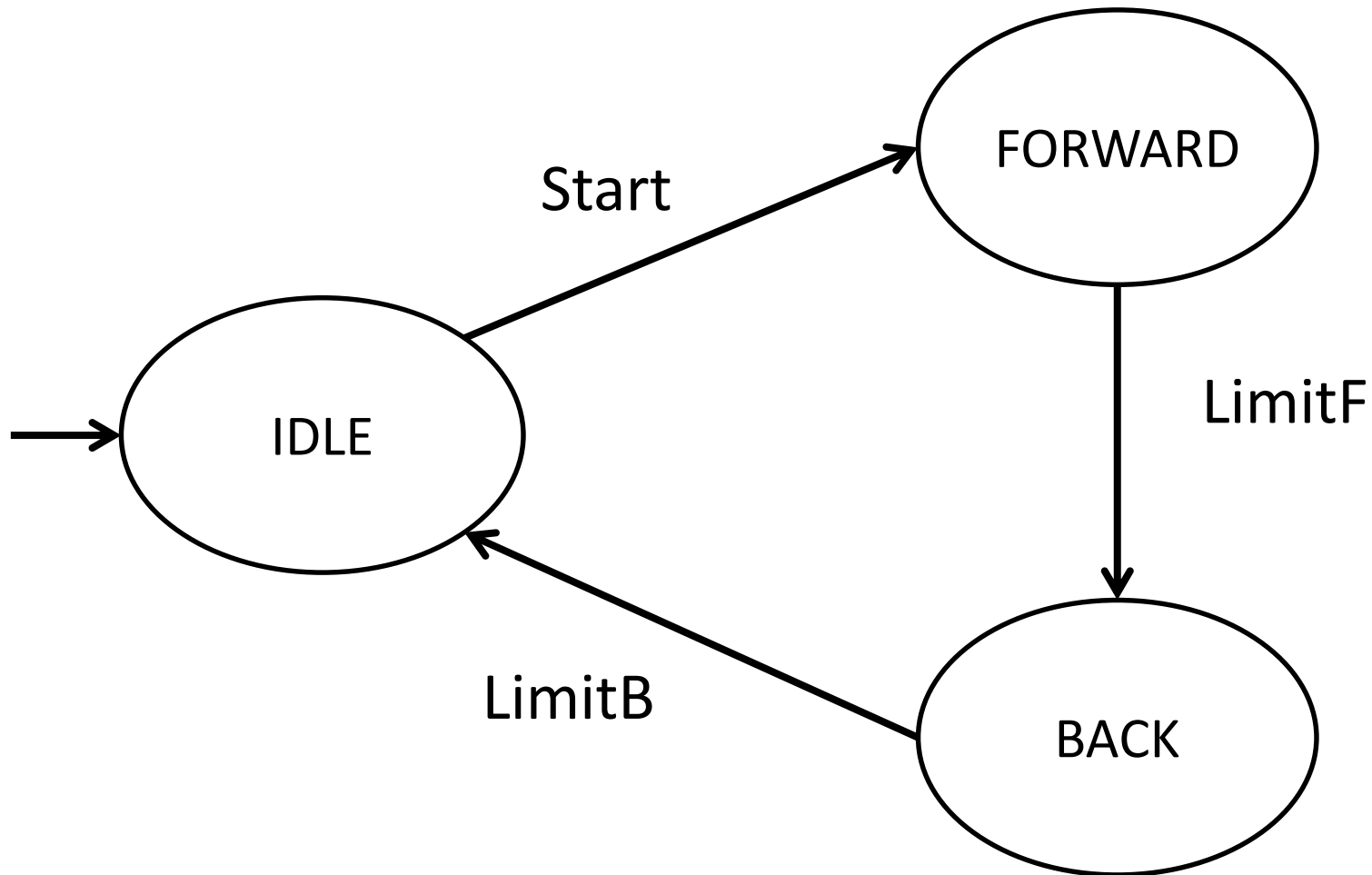
- RETURN kulcsszó
- Kiválasztás művelet tetszőleges ágában használható feltételes visszatérésre
- Függvények esetén a visszatérési értéket előbb be kell állítani (függvénynévvel egyező nevű változó)
- Ha hiányzik, az utolsó sor végrehajtása után történik meg a visszatérés

# Példa: Tologató



- A tologató hátsó vég helyzetéből a START bemenet felfutó élére indul
- Addig mozog előre, amíg az első végálláskapcsoló nem jelez
- Ekkor irányt vált és a hátsó végálláskapcsoló jelzéséig mozog hátra
- Amikor visszatért a kiindulási helyzetbe, a DONE kimenetet egyetlen ciklus idejére igazra állítja

# A tologató állapotgépe



# Tologató - deklarációs rész

```
FUNCTION_BLOCK FBPusher
VAR RETAIN
    StateEnum: (Idle, Fwd, Bwd);
    R_Start:    R_EDGE;
END_VAR
VAR_OUTPUT
    Motor: BOOL :=0;
    Dir:    BOOL;
    Done:   BOOL;
END_VAR
VAR_INPUT
    Start : BOOL;
    LimitF: BOOL;
    LimitB : BOOL;
END_VAR
```

# Tologató - programkód

```
R_Start (CLK:=Start);
```

```
CASE StateEnum OF
```

```
  Idle:      IF (R_Start.Q) THEN StateEnum:=Fwd; END_IF;  
             Done:=FALSE;
```

```
  Fwd:      IF (LimitF) THEN StateEnum:=Bwd; END_IF;
```

```
  Bwd:      IF (LimitB) THEN  
             StateEnum:=Idle;
```

```
             Done := TRUE;
```

```
          END_IF;
```

```
END_CASE;
```

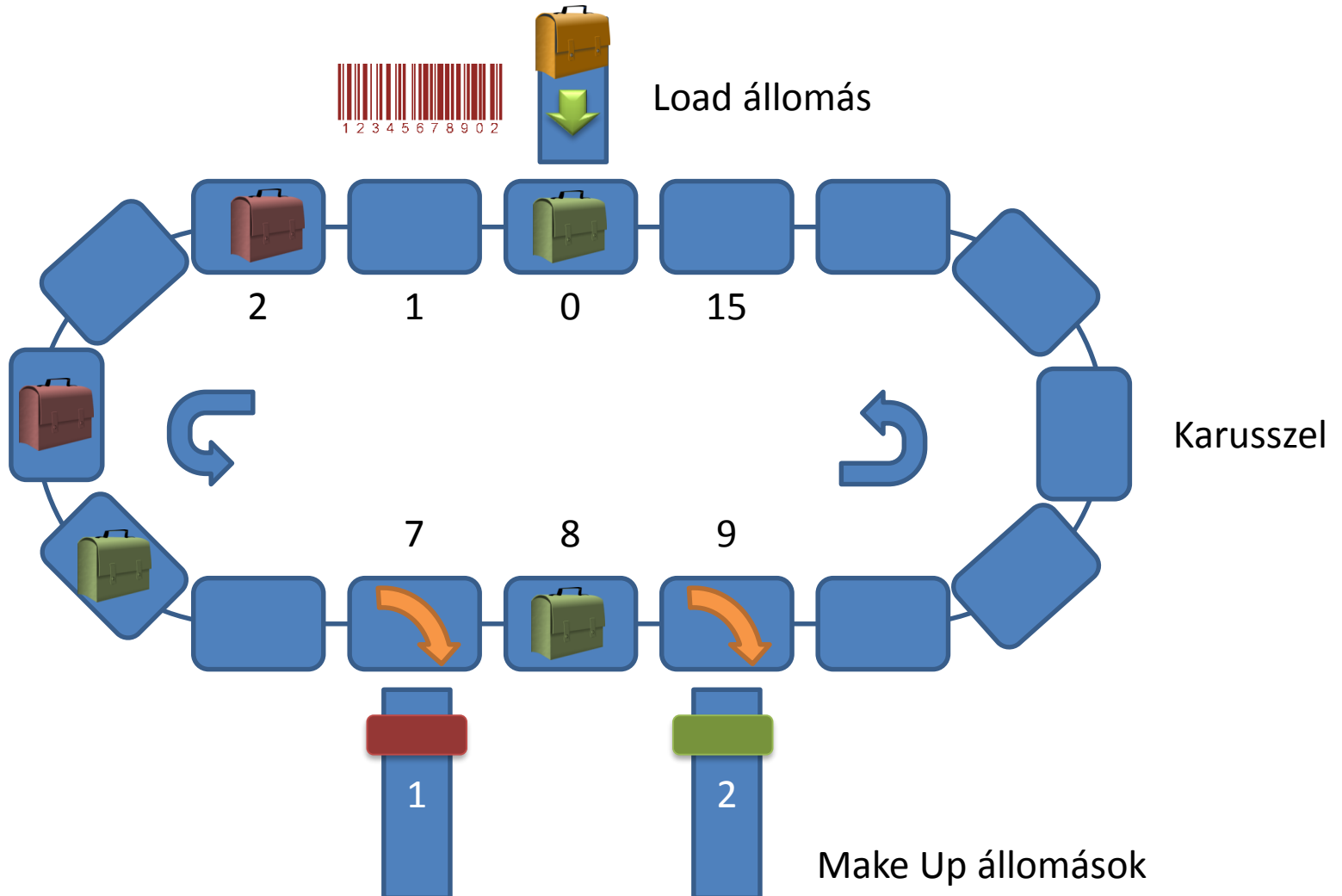
A Done változót itt TRUE értékre állítjuk, a következő ciklusban azonban nullázzuk is az Idle állapothoz tartozó esetben, így valóban egyetlen ciklus idejére lesz aktív

```
Motor:=NOT (StateEnum=Idle);
```

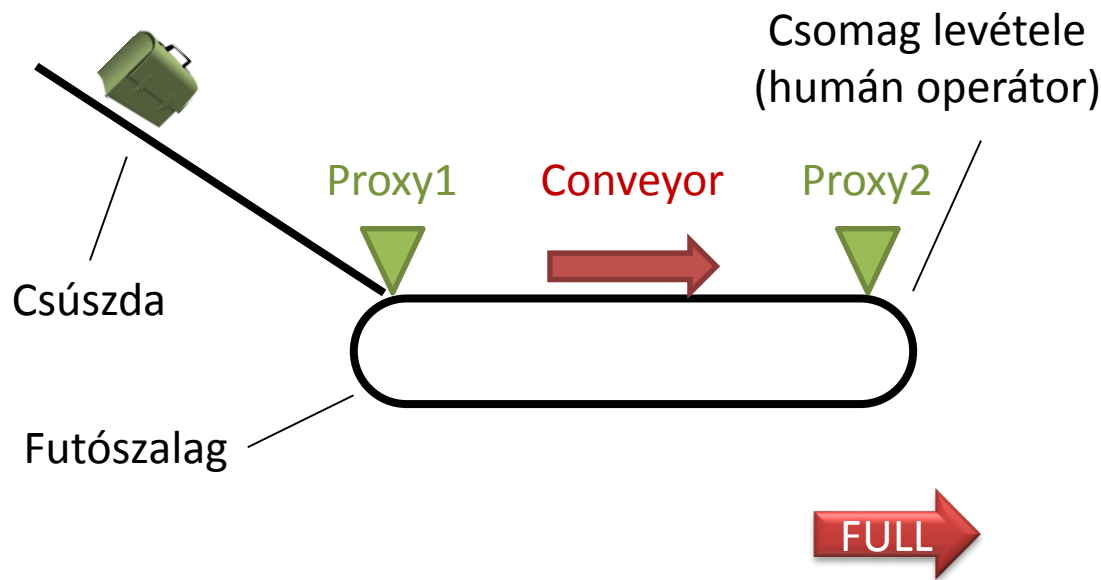
```
Dir:=(StateEnum=Fwd);
```

```
END_FUNCTION_BLOCK;
```

# Csomagszállító rendszer



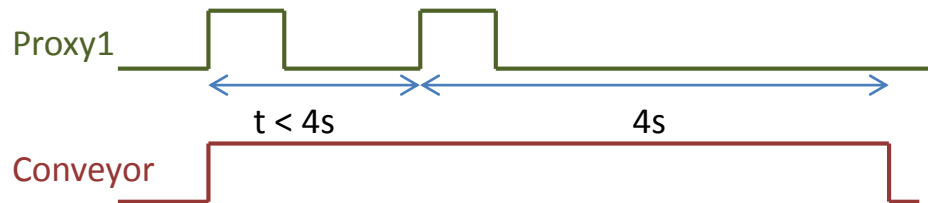
# Make Up állomás



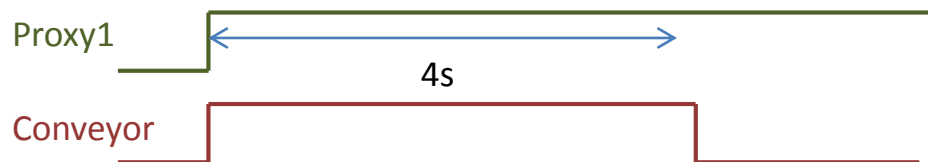
- A Proxy1 közelítésérzékelő jelzésére működtessük a szalagot 4 másodpercig
- A Proxy2 közelítésérzékelő lefutó élére működtessük a szalagot 1 másodpercig
- Ha tele van a szalag, jelezzük a Full kimeneten



# Időzítések a bemeneti oldalon



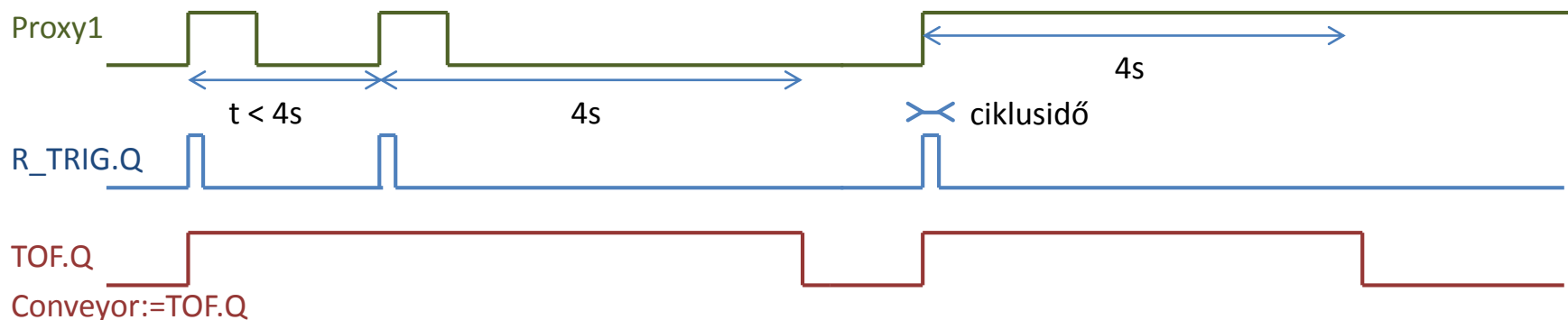
Ha az első csomag érkezése után elindított mozgás közben új csomag, érkezik, akkor a mozgás annak érkezése után 4 másodperccel álljon le



Ha a szalag megtelt, az érkezett csomag nem jut tovább az érzékelőtől. A mozgás ekkor is csak 4 másodpercig tartson.

Megoldás: R\_TRIG + TOF

- Az R\_TRIG funkcióblokk a bemenetére kötött közelítésérzékelő felfutó élére 1 PLC-ciklus idejéig aktív kimenetet (R\_TRIG.Q) ad
- R\_TRIG.Q lefutó élét egy TOF időzítővel 4 másodperccel késleltetjük

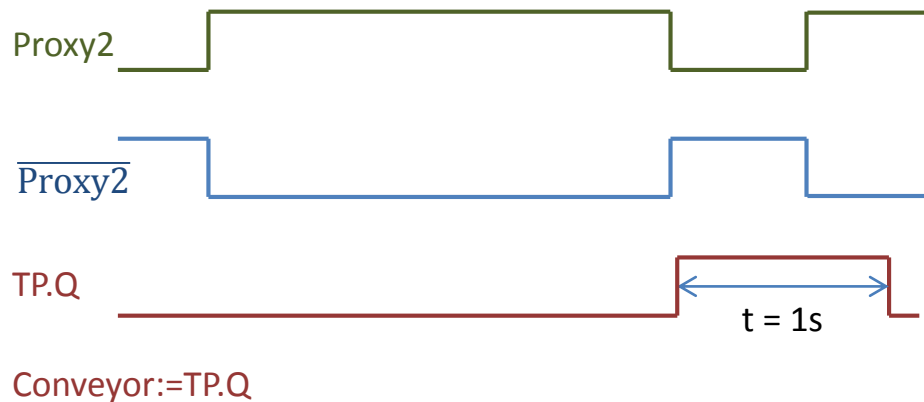


# Időzítések a kimeneti oldalon



Ha elvesznek egy csomagot a kimeneti oldalról (lefutó él), akkor 1 másodpercig működtetni kell a szalagot. Ezalatt az esetleges további jelváltásokat nem kell figyelni.

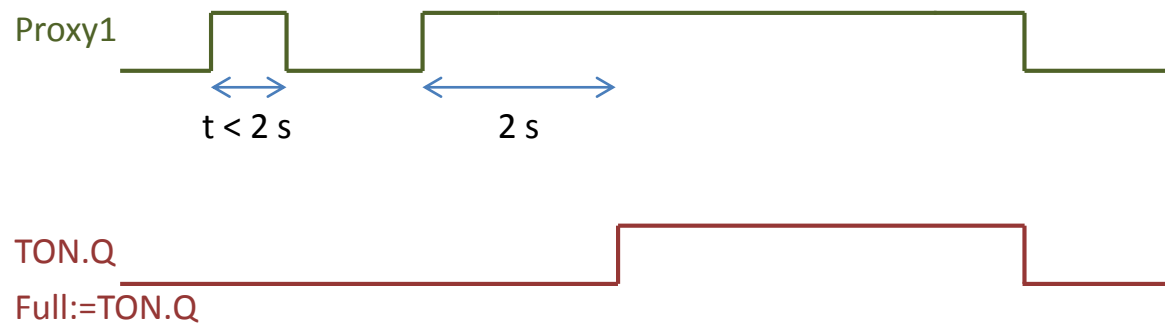
Megoldás: TP időzítő  $\overline{\text{Proxy2}}$  bemenettel



A szalagot mozgatni kell, ha akár a be-, akár a kimeneti időzítő kimenete aktív!

# Teli szalag jelzése

- Mikor van tele a szalag?
- Ha egy csomag érkezik, elindítjuk a szalagot, de a csomag 2 másodpercen belül nem távozik a közelítésérzékelő elől
- Ha a csomag később távozik, akkor már nincs tele a szalag
- Megoldás: TON időzítő



```
FUNCTION_BLOCK FBMakeUp
```

```
VAR_INPUT
```

```
    Proxy1: BOOL;
```

```
    Proxy2: BOOL;
```

```
END_VAR
```

```
VAR_OUTPUT
```

```
    Conveyor: BOOL;
```

```
    Full: BOOL;
```

```
END_VAR
```

```
VAR
```

```
    rProxy: R_TRIG;
```

```
    TimerIn: TOF;
```

```
    TimerOut: TP;
```

```
    TimerFull: TON;
```

```
END_VAR
```

```
rProxy(CLK:=Proxy1);
```

```
TimerIn(IN:=rProxy.Q, PT:=T#4s);
```

```
TimerOut(IN:=NOT Proxy2, PT:=T#1s);
```

```
TimerFull(IN:=Proxy1, PT:=T#2s, Q=>Full);
```

```
Conveyor:=TimerIn.Q OR TimerOut.Q;
```

Bemeneti változók: a két közelítésérzékelő

Kimeneti változók: futószalag működtetés és tele szalag jelzés

Bemeneti érzékelő felfutó élét érzékelő FB

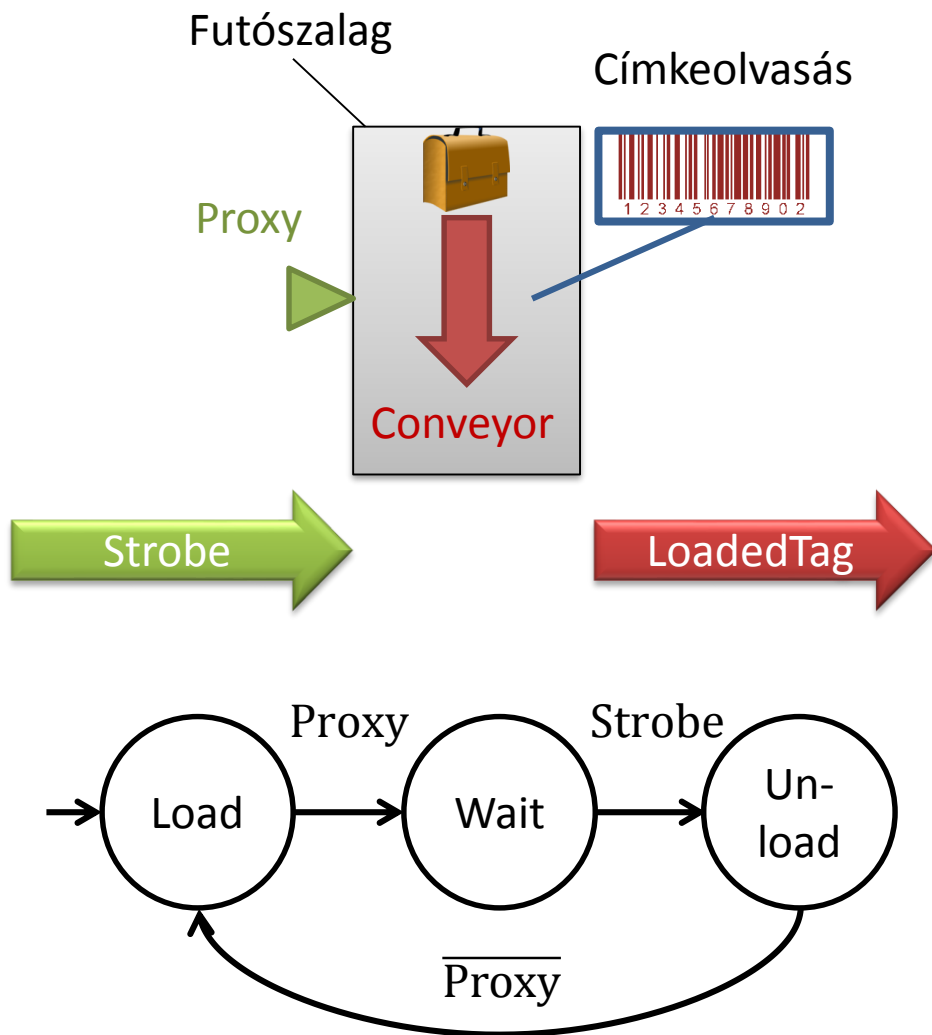
Időzítők példányosítása

Időzítések bejövő és kimenő csomag esetén

Teli szalag jelzése

Szalag-kimenet beállítása

# Load állomás



- Indításkor járjon a futószalag
- Ha csomag érkezik, akkor állítsuk meg és olvassuk be a címkét (TagRead() függvény)
- Strobe jel (hívás Strobe=TRUE értékkel): az állomáson tartózkodó csomag továbbítható
  - Ha van csomag, akkor indítsuk el a szalagot és a címkét adjuk ki a LoadedTag kimeneten
  - Ha nincs csomag, akkor a LoadedTag kimenet értéke legyen 0
  - Strobe jel nélküli hívás esetén is 0-ba állítsuk a LoadedTag kimenetet

```
FUNCTION_BLOCK FBLoad
```

```
VAR_INPUT
```

```
    Strobe: BOOL;
```

```
    Proxy: BOOL;
```

```
END_VAR
```

```
VAR_OUTPUT
```

```
    Conveyor: BOOL;
```

```
    LoadedTag: UINT;
```

```
END_VAR
```

```
VAR
```

```
    Tag: UINT;
```

```
    State: (Load, Wait,  
            Unload);
```

```
END_VAR
```

Felsorolásként származtatott  
állapotváltozó (adattípusa INT)

Szalag-kimenet  
beállítása

```
CASE State OF  
Load:
```

```
    Tag:=0;
```

```
    IF Proxy THEN
```

```
        State:=Wait;
```

```
        Tag:=TagRead();
```

```
    END_IF
```

```
Wait:
```

```
    IF Strobe THEN
```

```
        State:=Unload;
```

```
    END_IF
```

```
Unload:
```

```
    IF NOT Proxy THEN
```

```
        State:=Load;
```

```
    END_IF
```

```
END_CASE
```

```
IF Strobe THEN
```

```
    LoadedTag:=Tag;
```

```
ELSE
```

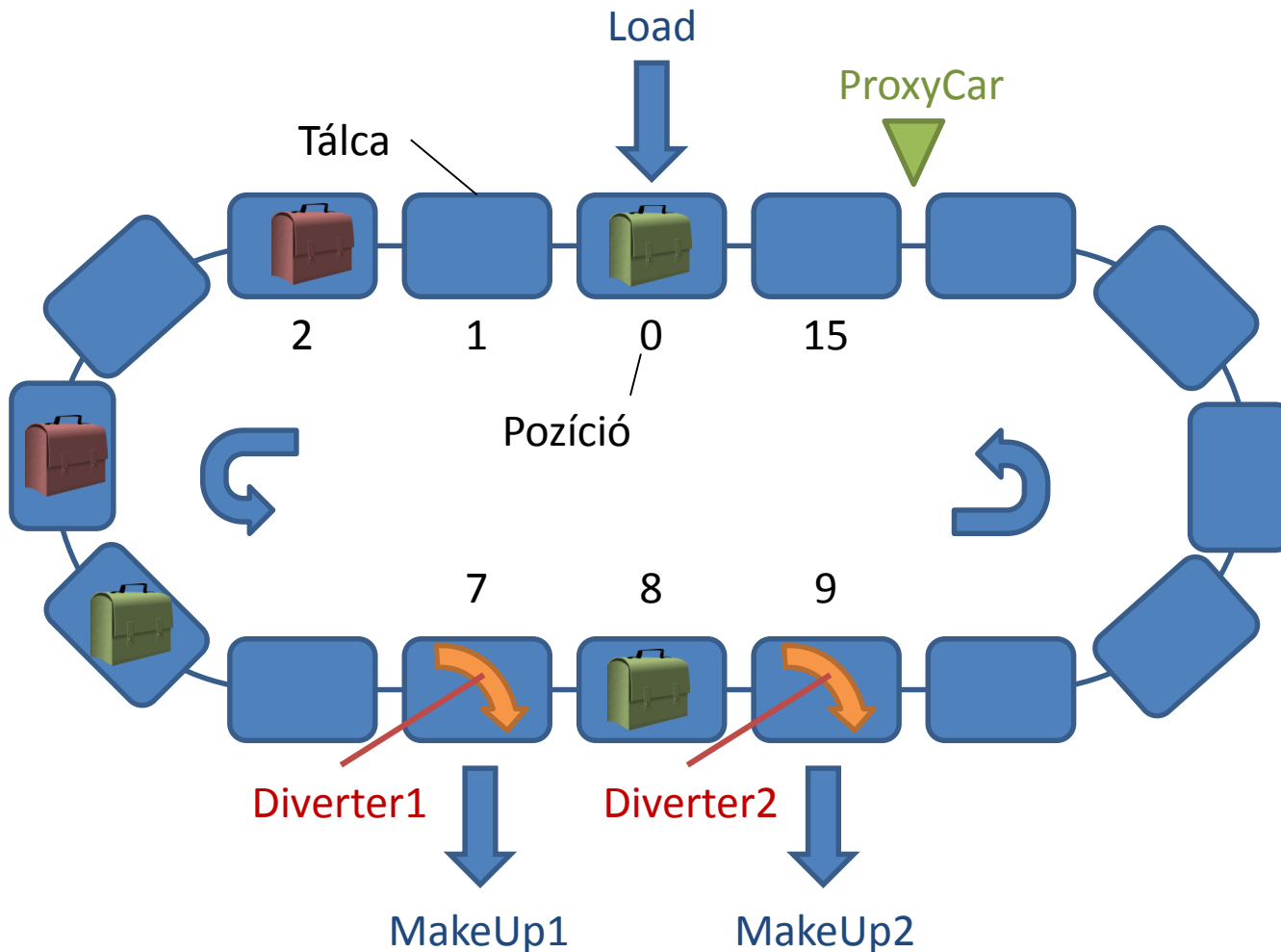
```
    LoadedTag:=0;
```

```
END_IF
```

```
Conveyor:=NOT(State=Wait);
```

Állapotgép CASE  
struktúrával

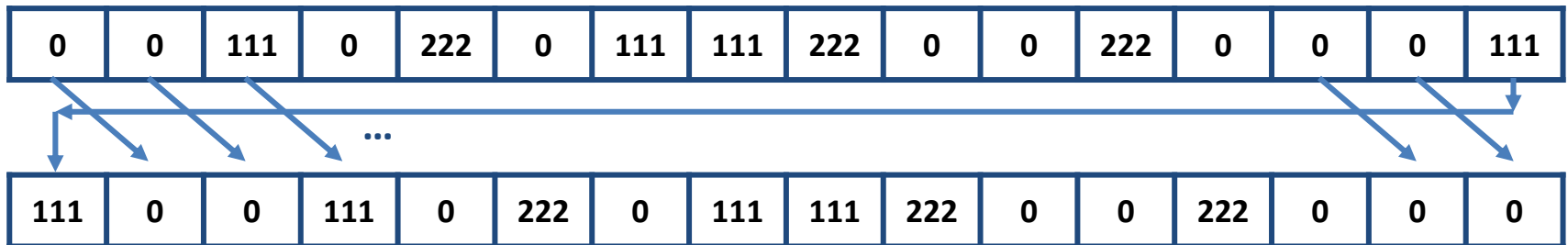
# Főprogram



- A karusszel folyamatosan mozog, tálcáiban a csomagokkal
- A csomagok a Load állomás felől érkeznek
- Csomag csak akkor érkezhets, ha a Load állomásnál lévő (0 pozíció) tálca üres
- A ProxyCar érzékelő felfutó éle jelzi egy tálca elhaladását
- A csomagok a diverterek segítségével boríthatók a Make Up állomások csúszdáira
- Teli Make Up állomásra tilos csomagot továbbítani
- A csomagokat a címke által jelzett állomásra kell eljuttatni

# Csomagok követése

- Az egyes pozíciókban lévő csomagok címkeit tároljuk egy 16 elemű tömbben
  - 111: 1. MakeUp állomásra továbbítandó
  - 222: 2. MakeUp állomásra továbbítandó
  - 0: üres tálca
- Proxy felfutó élére léptessük a vektort (cirkulárisan): forgatás (ROR)





# Tömb forgatását végző függvény

```
FUNCTION Rotate : ARRAY[0..15]
OF UINT
VAR_INPUT
    ArrayIn: ARRAY [0..15] OF
UINT;
END_VAR
VAR
    i: INT;
END_VAR

Rotate[0]:=ArrayIn[15];
FOR i:=15 TO 1 BY -1 DO
    Rotate[i]:=ArrayIn[i-1];
END_FOR
```

# Főprogram

- Proxy felfutó élére
  - Tömb léptetése
  - Ha a 0 pozícióban a tálca üres, akkor Strobe jelzés küldése FBLoad-nak
  - Ha a 7. pozícióban 111 címkéjű csomag van és az A Make Up állomás nincs tele, akkor a tálca billentése és 0 beírása a 7. pozícióba
  - Ha a 9. pozícióban 222 címkéjű csomag van és a B Make Up állomás nincs tele, akkor a tálca billentése és 0 beírása a 9. pozícióba
- Minden ciklusban: állomás-funkcióblokkok hívása

```
PROGRAM PLC_PRG
```

```
VAR
```

```
    MakeUpA: FBMakeUp;
```

```
    MakeUpB: FBMakeUp;
```

```
    Load: FBLoad;
```

```
    ProxyA1 AT %IX0.0: BOOL;
```

```
    ProxyA2 AT %IX0.1: BOOL;
```

```
    ConvA AT %QX0.0: BOOL;
```

```
    ProxyB1 AT %IX0.2: BOOL;
```

```
    ProxyB2 AT %IX0.3: BOOL;
```

```
    ConvB AT %QX0.1: BOOL;
```

```
    ProxyLoad AT %IX0.4 : BOOL;
```

```
    ConvLoad AT %QX0.2: BOOL;
```

```
    ProxyCar AT %IX0.5: BOOL;
```

```
    Diverter1 AT %QX0.3: BOOL ;
```

```
    Diverter2 AT %QX0.4: BOOL ;
```

```
    Carrousel: ARRAY [0..15] OF UINT;
```

```
    rCar: R_TRIG;
```

```
END_VAR
```

Állomás-funkcióblokkok példányosítása

Fizikai ki- és bemenetek

Karusszel-tömb

```
rCar (CLK:=ProxyCar) ;
```

Proxi felfutó élének figyelése

```
IF rCar.Q THEN
```

Tömb léptetése

```
  Carrousel:=Rotate (Carrousel) ;
```

```
  IF (Carrousel[7]=111 AND NOT MakeUpA.Full) THEN
```

```
    Diverter1:=TRUE:
```

```
    Carrousel[7]:=0;
```

Csomag továbbítása a Make Up állomás felé

```
  ELSE
```

```
    Diverter1:=FALSE;
```

```
  END_IF;
```

```
  IF (Carrousel[9]=222 AND NOT MakeUpB.Full) THEN
```

```
    Diverter2:=TRUE:
```

```
    Carrousel[9]:=0;
```

```
  ELSE
```

```
    Diverter2:=FALSE;
```

```
  END_IF;
```

Új csomag betöltésének engedélyezése (Strobe=TRUE)

```
  IF Carrousel[0]=0 THEN
```

```
    Load (Proxy:=ProxyLoad, Strobe:=TRUE,
```

```
    LoadedTag=>Carrousel[0], Conveyor=>ConvLoad) ;
```

```
  END_IF
```

FB hívások minden ciklusban

```
END_IF;
```

```
MakeUpA (Proxy1:=ProxyA1, Proxy2:=ProxyA2, Conveyor=>ConvA) ;
```

```
MakeUpB (Proxy1:=ProxyB1, Proxy2:=ProxyB2, Conveyor=>ConvB) ;
```

```
Load (Proxy:=ProxyLoad, Strobe:=FALSE, Conveyor=>ConvLoad) ;
```

# Alternatív megoldás

```
rCar (CLK:=ProxyCar) ;  
IF rCar.Q THEN  
    Carrousel:=Rotate (Carrousel) ;  
    Diverter1:=(Carrousel[7]=111) AND (NOT MakeUpA.Full) ;  
    Diverter2:=(Carrousel[9]=222) AND (NOT MakeUpB.Full) ;  
    Carrousel[7]:=BOOL_TO_INT (NOT Diverter1) *Carrousel[7] ;  
    Carrousel[9]:=BOOL_TO_INT (NOT Diverter2) *Carrousel[9] ;
```

Csomagtovábbítás másképp:  
logikai függvény használata

Tömb elemeinek nullázása: ha a diverter-kimenet 1,  
akkor 0-t (INT), egyébként az eredeti értéket írjuk be

```
IF Carrousel[0]=0 THEN  
    Load (Proxy:=ProxyLoad, Strobe:=TRUE,  
    LoadedTag=>Carrousel[0], Conveyor=>ConvLoad) ;  
END_IF  
END_IF ;  
MakeUpA (Proxy1:=ProxyA1, Proxy2:=ProxyA2, Conveyor=>ConvA) ;  
MakeUpB (Proxy1:=ProxyB1, Proxy2:=ProxyB2, Conveyor=>ConvB) ;  
Load (Proxy:=ProxyLoad, Strobe:=FALSE, Conveyor=>ConvLoad) ;
```