A TinyOS operációs rendszer

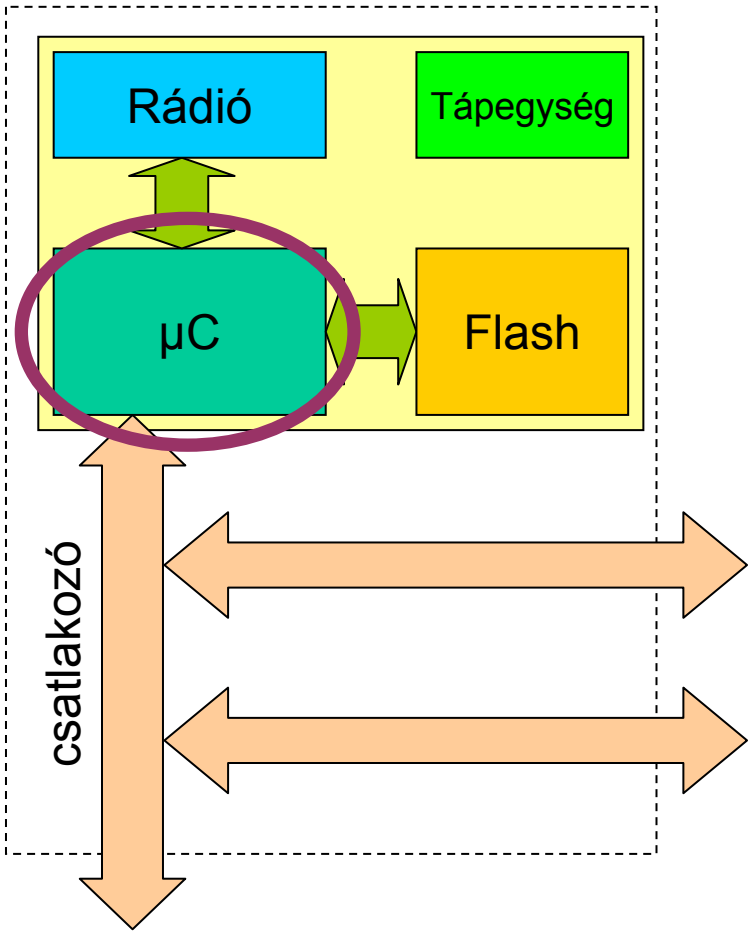
(bevezető)

http://www.tinyos.net/

Beágyazott Információs Rendszerek

Budapesti Műszaki és Gazdaságtudományi Egyetem

Ismétlés: Mica Proc
Atmel ATmega128L (103L)



128kB programmemória (flash)
4kB EEPROM
4kB SRAM
32 általános I/O vonal
8 bemeneti-, 8 kimeneti vonal
32 általános célú regiszter
RTC (real time counter)
4 timer/counter/PWM
UART
WDT (watchdog timer)
SPI port
3 energiatakarékos üzemmód

TinyOS operációs rendszer

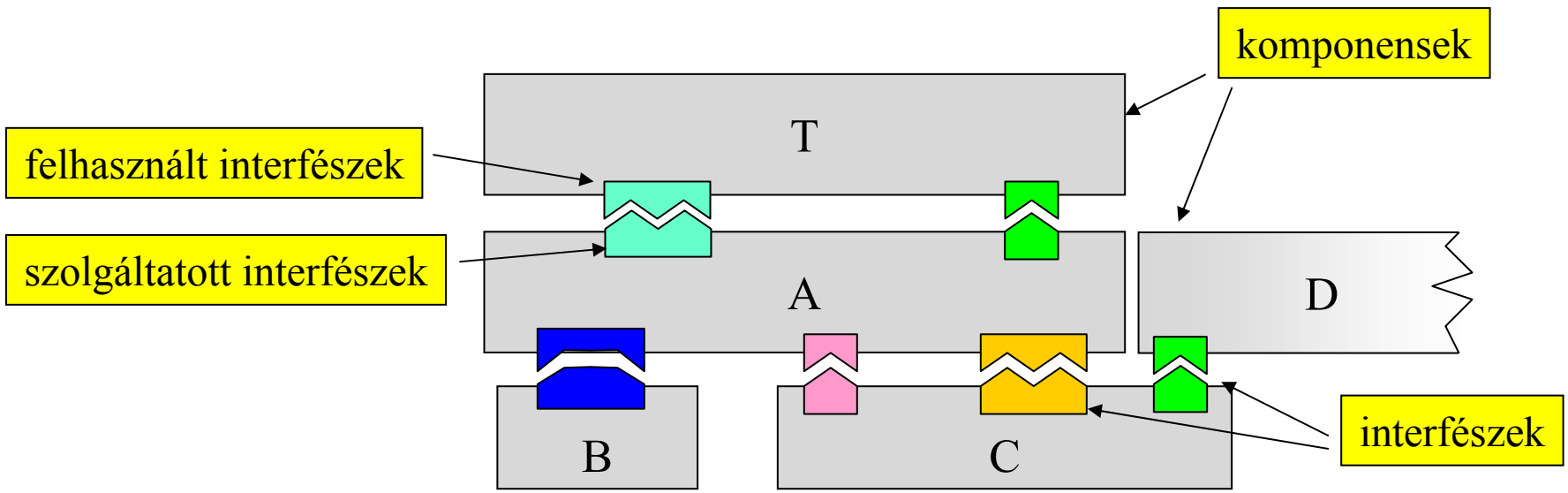
- vezeték nélküli szenzorhálózatokhoz kifejlesztett
- nyílt kódú (*open source*)
- ingyenes
- komponens alapú
- eseményvezérelt (*event triggered*)
- beágyazott operációs rendszer
- kapcsolódó programnyelv: NesC

NesC programnyelv

- Beágyazott rendszerekhez fejlesztették
- Szenzorhálózatokhoz különösen jól illeszkedik
- C-szerű szintaxis
- A TinyOS filozófiájához illeszkedik
  - konkurencia-modell
  - komponensek kezelése
- Fordítás-idejű konkurencia-ellenőrzés

Komponensek

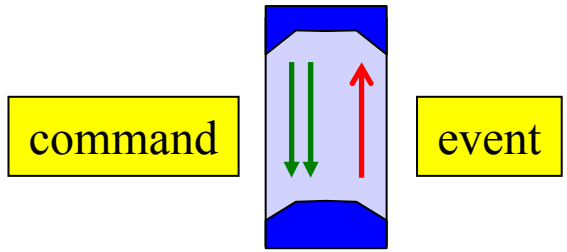
- Az applikáció összehuzalozott komponensekből épül fel
- A komponensekhez *kétirányú* interfészeken keresztül lehet kapcsolódni
- A komponens szolgáltatást nyújt (*provides*) és használ (*uses*)



Komponensek típusai

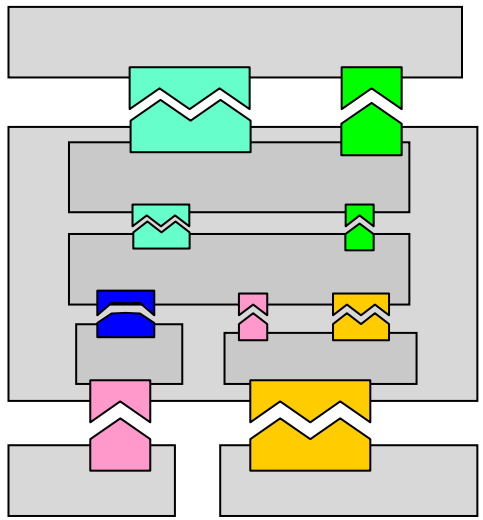
Modul (module): interfészek *implementációi*

- Interfészek tartalmaznak:
  - parancs (command)
  - esemény (event)



Konfiguráció (configuration): *huzalozás*

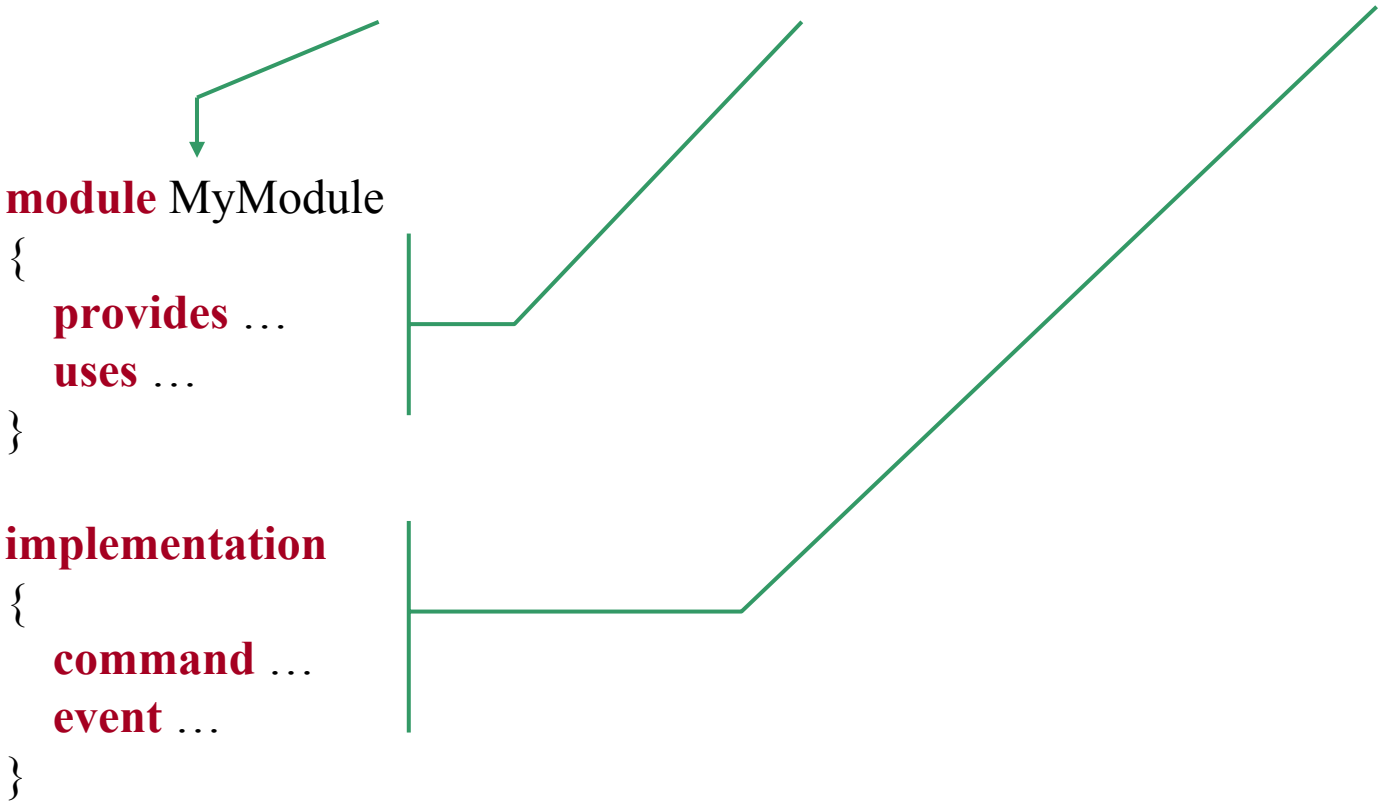
- Hierarchikus
- Tartalmazhat:
  - modulokat
  - konfigurációkat



Komponensek:

Modulok

module <azonosító> <interfész definíció> <implementáció>



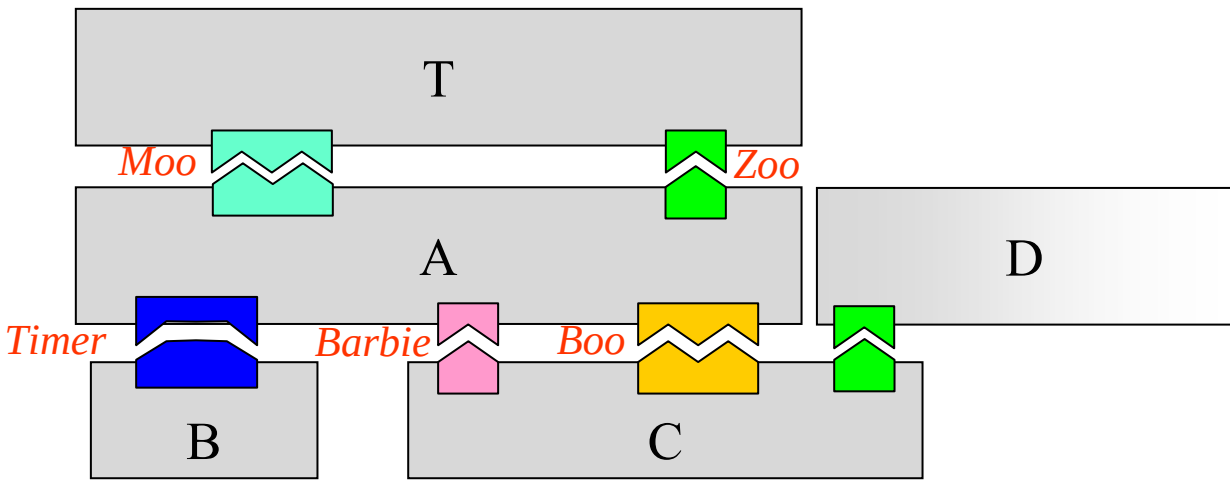
```
module MyModule
{
   provides …
   uses …
}

implementation
{
   command …
   event …
}
```

Komponensek:

Modulok: interfész definíció

- **Definiálja a használt és szolgáltatott interfészeket**
- Implementálja
  - a szolgáltatott command-okat
  - a felhasznált eseményeket

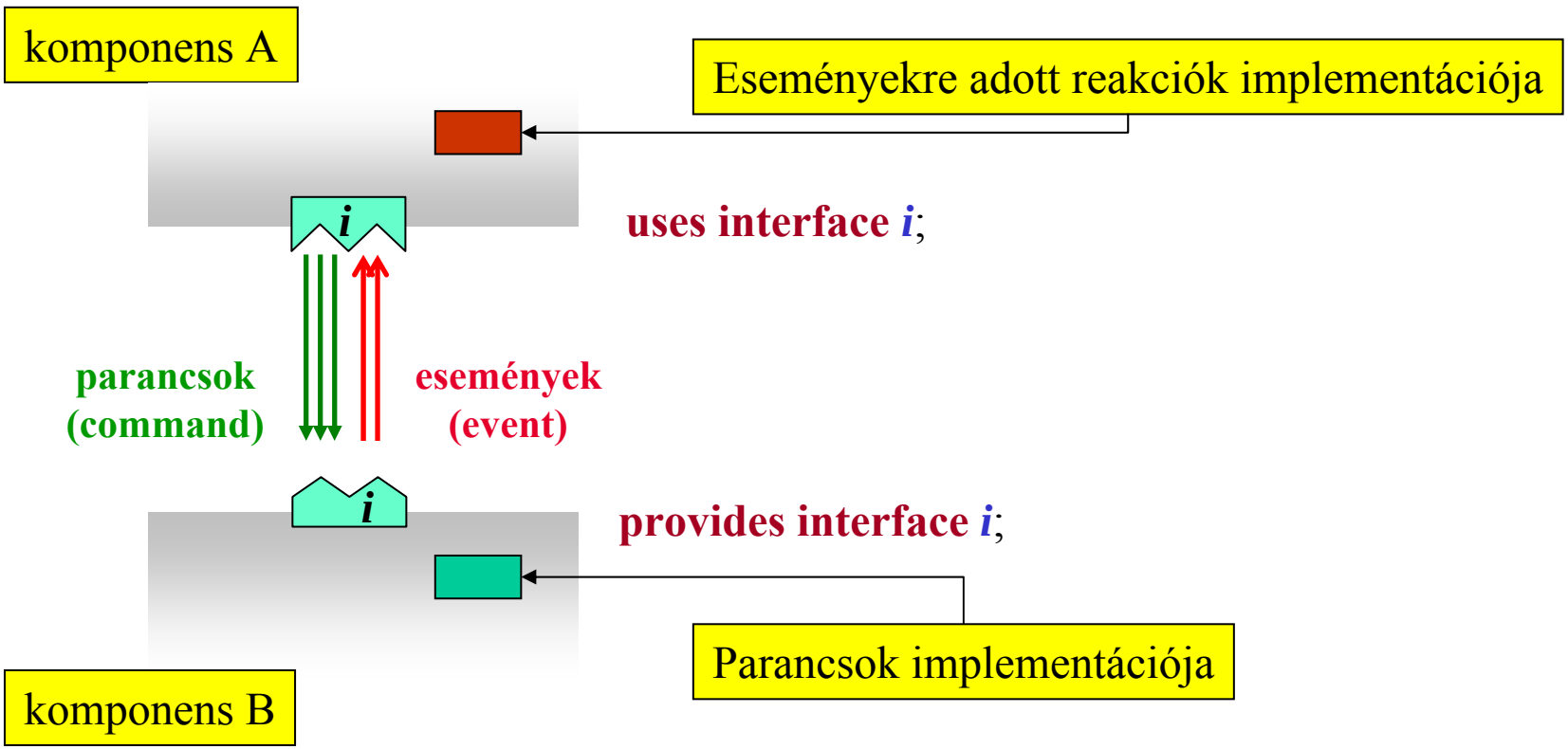
```
module A {
  provides {
    interface Moo;
    interface Zoo;
  }
  uses {
    interface Timer;
    interface Barbie;
    interface Boo;
  }
}
...
```



Komponensek:

Parancsok és események

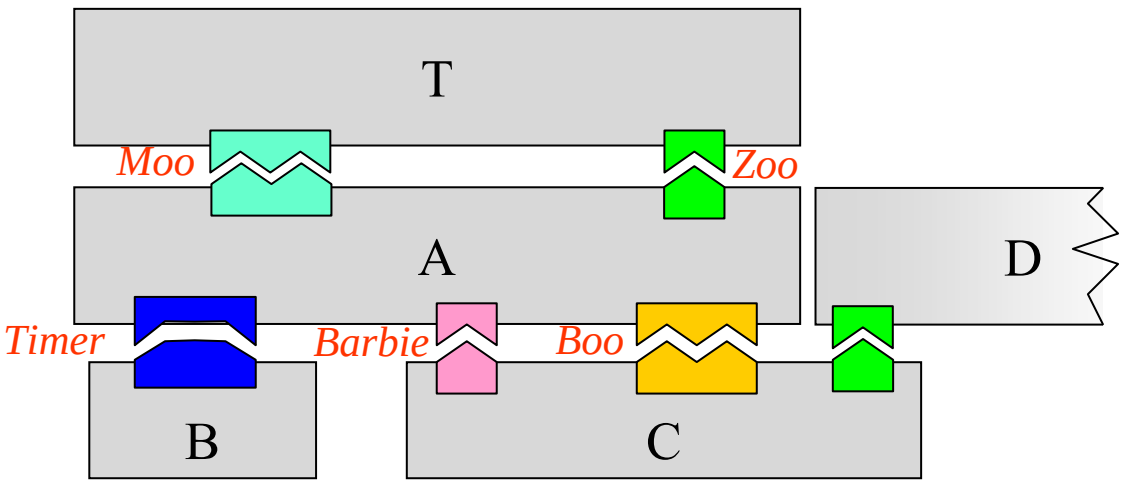
Parancsok és események iránya, implementációja



Komponensek:

Modulok: implementáció

- Definiálja a használt és szolgáltatott interfészeket
- **Implementálja**
  - **a szolgáltatott command-okat**
  - **a felhasznált eseményeket**



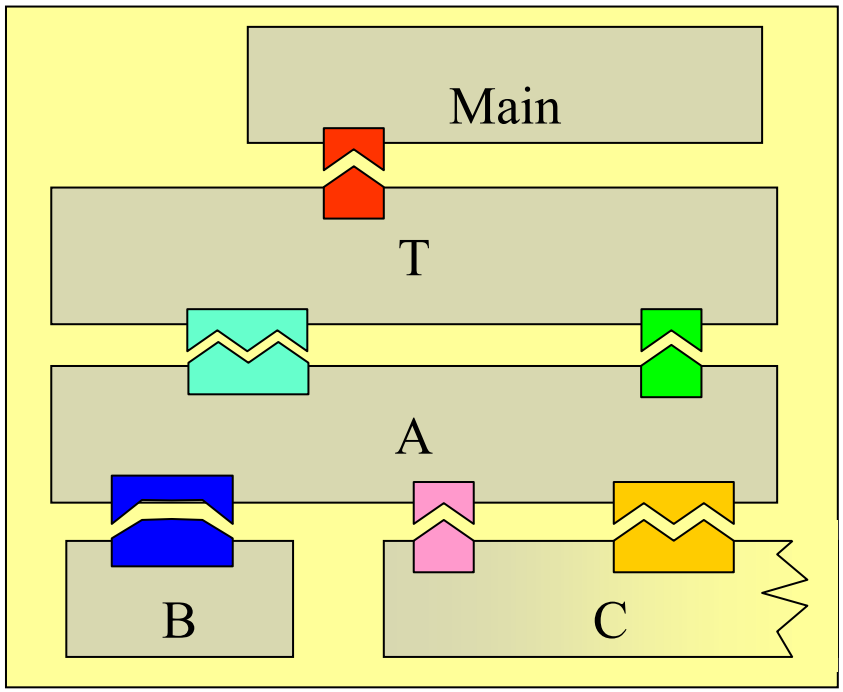

```
implementation {

command result_t Moo.start()
 {   ...
 }
command result_t Moo.stop()
 {   ...
 }
command result_t Zoo.init()
 {   ...
 }
command result_t Zoo.close()
 {   ...
 }
event result_t Timer.fired()
 {   ...;
 }
event result_t Barbie.Cries()
 {   ...;
 }
}
```

Komponensek:

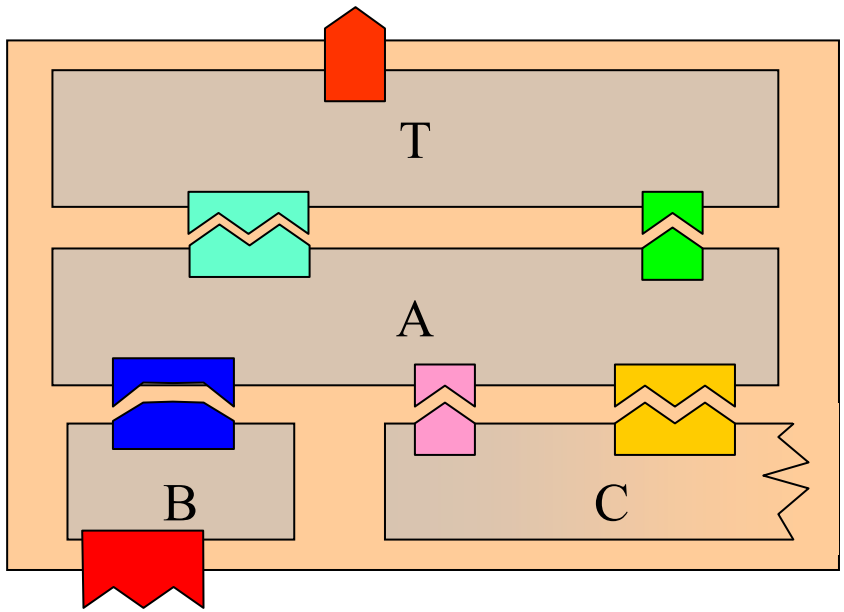
Konfigurációk

A konfigurációk komponensek **huzalozására** (**->**) használatosak

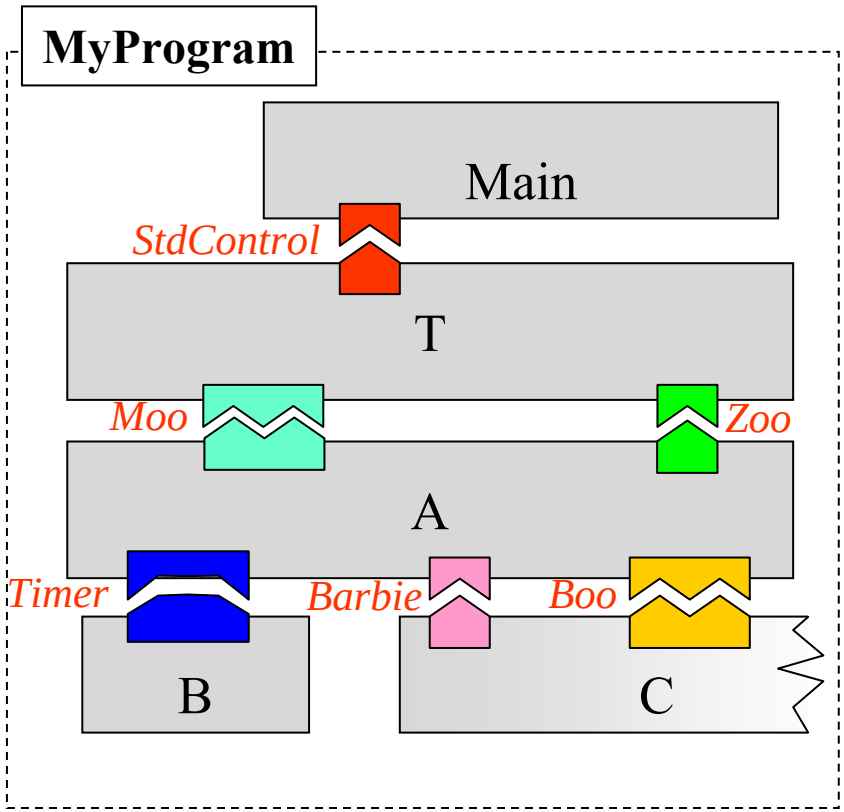
I. Top-level konfiguráció (applikáció)



II. Újrafelhasználható komponens

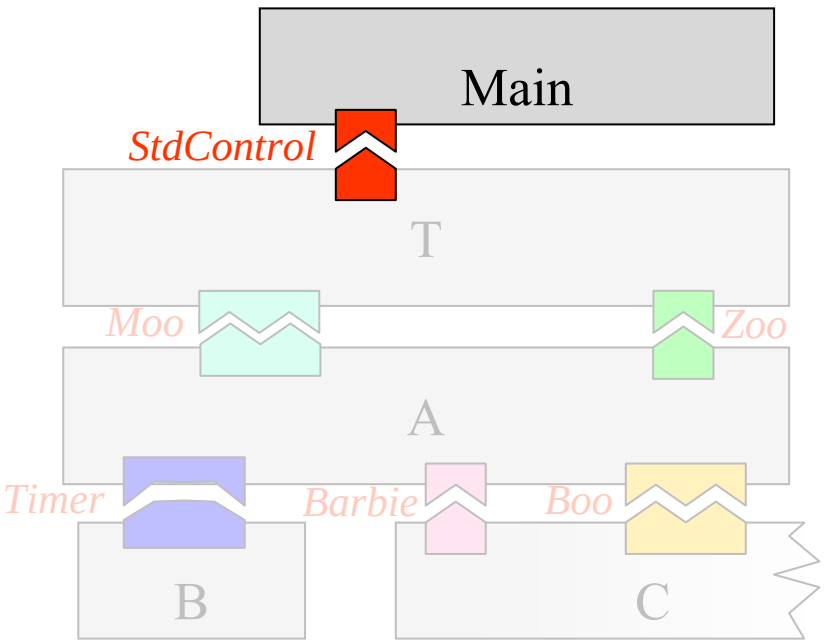


Top-level konfigurációk I.




```
configuration MyProgram {
}

implementation {
  components Main,T, A, B, C,...;
    ...
  Main.StdControl -> T.StdControl;
  T.Moo -> A.Moo;
  T.Zoo -> A.Zoo;
  A.Timer -> B.Timer;
  A.Barbie -> C.Barbie;
  A.Boo -> C.Boo;
    ...
}
```

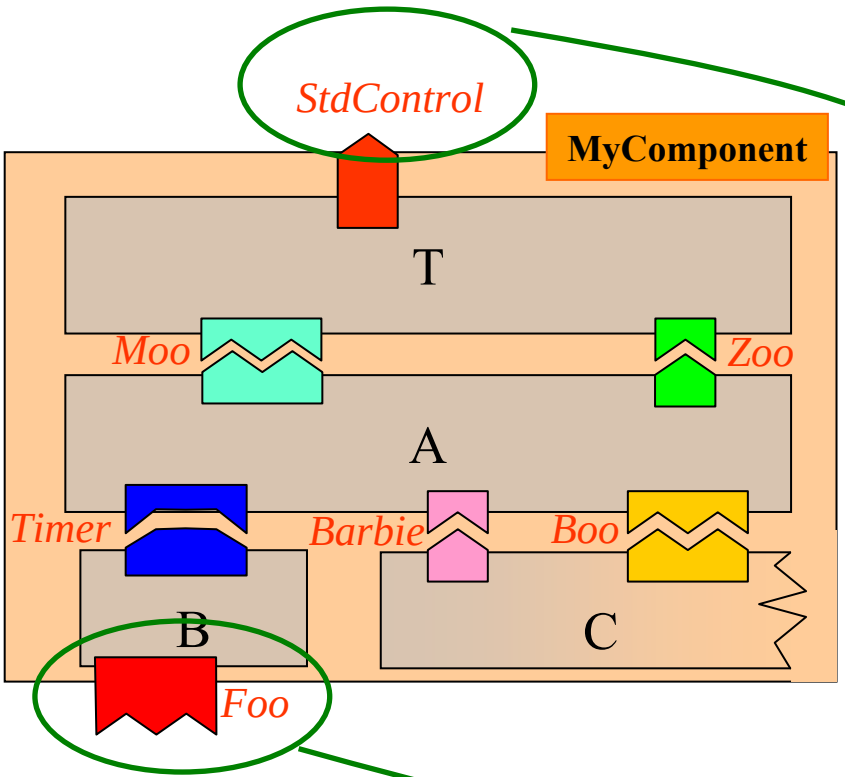
Top-level konfigurációk II.



- Minden programnak tartalmaznia kell egy **Main** komponenst
- A **Main** komponens indul el *reset* után először
- Meghívja a **StdControl** interfész ***init*** utasítását
- Meghívja a **StdControl** interfész ***start*** utasítását
- Elindítja az ütemezőt
- A ***stop*** utasítás az erőforrások kímélését szolgálja (standby)

Komponensek:Konfigurációk:

Újrafelhasználható komponensek




```
configuration MyComponent {
  provides interface StdControl;
  uses interface Foo;
}

implementation {
  components T, A, B, C,...;
   ...
  StdControl = T;
  T.Moo -> A.Moo;
  T.Zoo -> A.Zoo;
  A.Timer -> B.Timer;
  A.Barbie -> C.Barbie;
  A.Boo -> C.Boo;
  Foo = B;
   ...
}
```

Komponensek:Konfigurációk:

Komponens egyedek

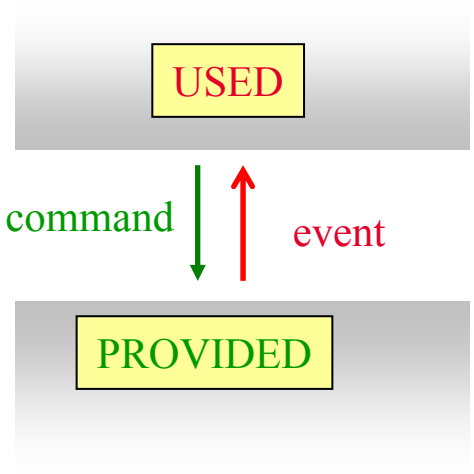
Komponens egyedek átnevezhetők. Könnyebb a komponensek későbbi cseréje.

components T = components T as T

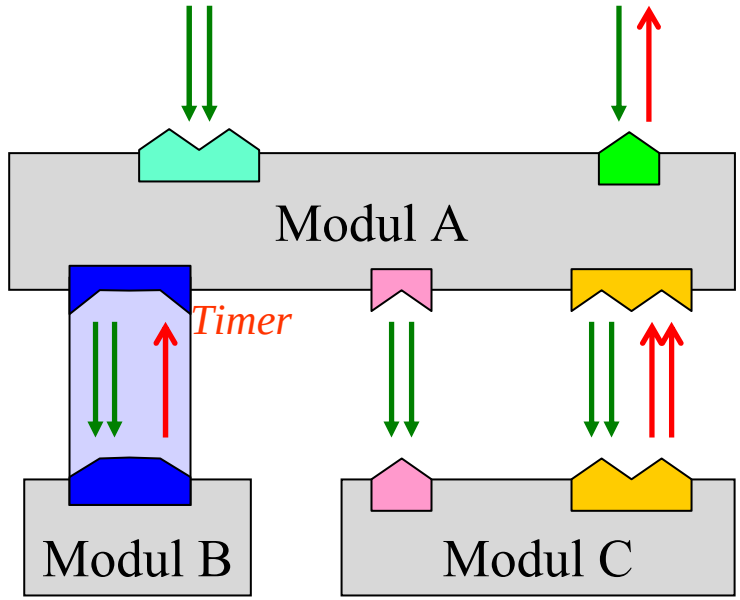
```
configuration MyComponent {
  provides interface StdControl;
  uses interface Foo;
}

implementation {
  components T, A, B, C,...;
    ...
  StdControl = T;
  T.Moo -> A.Moo;
  T.Zoo -> A.Zoo;
  A.Timer -> B.Timer;
  A.Barbie -> C.Barbie;
  A.Boo -> C.Boo;
  Foo = B;
    ...
}
```

```
configuration MyComponent {
  provides interface StdControl;
  uses interface Foo;
}

implementation {
  components T as Q, A, B, C,...;
    ...
  StdControl = Q;
  Q.Moo -> A.Moo;
  Q.Zoo -> A.Zoo;
  A.Timer -> B.Timer;
  A.Barbie -> C.Barbie;
  A.Boo -> C.Boo;
  Foo = B;
    ...
}
```

Interfészek




```
module A {
  provides {
    interface Moo;
    interface Zoo;
  }
  uses {
    interface Timer;
    interface Barbie;
    interface Boo;
  }
}
```




```
interface Timer {
    command result_t start(char type, uint32_t interval);
    command result_t stop();
    event   result_t fired();
}
```

Interfészek: StdControl

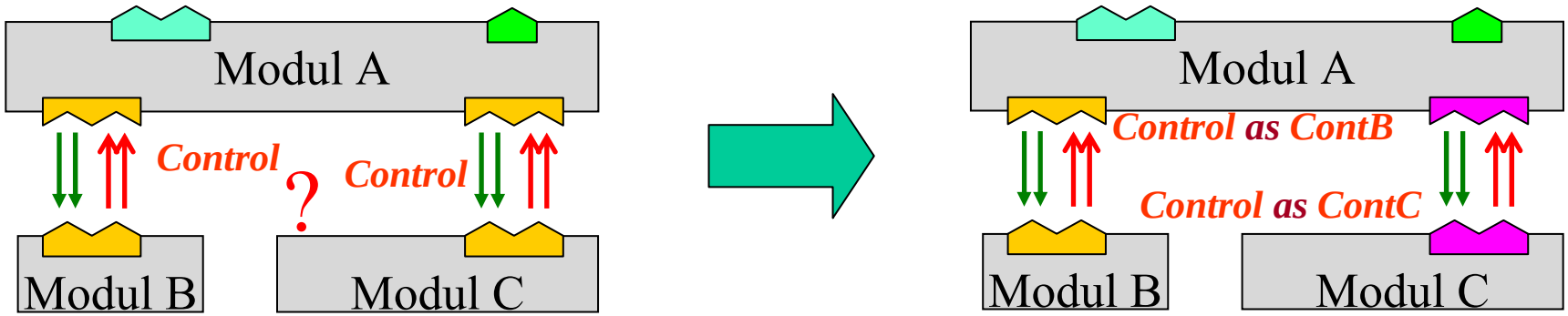
```
interface StdControl {
    command result_t init();
    command result_t start();
    command result_t stop();
}
```

A StdControl interfész felhasználása: `init* (start|stop)*`

„Mély” szemantikájú: minden komponensnek *init*, *start*, vagy *stop* hatására meg kell hívnia saját alkomponenseinek *init*, *start*, vagy *stop* utasitásait!

Interfész egyedek

Interfészek elnevezése, példányosítása



típus

név

```
module A {
uses {
  interface Control as ContB;
  interface Control as ContC;
  }
```

```
implementation {
  components A, B, C;
  A.ContB -> B.Control;
  A.ContC -> C.Control;
}
```

Megjegyzés
Rövid forma:
interface Control as Control; = interface Control;

Paraméterezett interfészek

Modul definíció

```
Modul A {
provides {
    ...
    interface Boo[uint8_t id];
    ...
  } ...
```

256 interfész egyedet definiál

Konfiguráció

```
implementation {
  components A, B, …;
    ...
    B.Moo -> A.Boo[12];
    ...
}
```

Egy egyedet felhasznál

Taszkok és HW eseményt kezelő függvények

- **Taszk**: függvényhívás késleltetett végrehajtással
- Taszk indítását kezdeményezik, majd az valamikor elindul és lefut
- Nincs végtelen ciklus és blokkoló utasítás
- Taszk indítását kezdeményezheti:
  - másik taszk
  - esemény (event)
  - utasítás (command)
- Taszk nem szakíthat meg taskot
- Taszkot megszakíthat HW eseményt kezelő függvény
- Hosszabb végrehajtási idejű is lehet

- **HW eseményt kezelő függvény**: HW IT-hez rendelve
- Bármikor futhat
- Más kódot (taszkot vagy másik HW kezelőt) megszakíthat
- Rövid legyen

Taskok kezelése

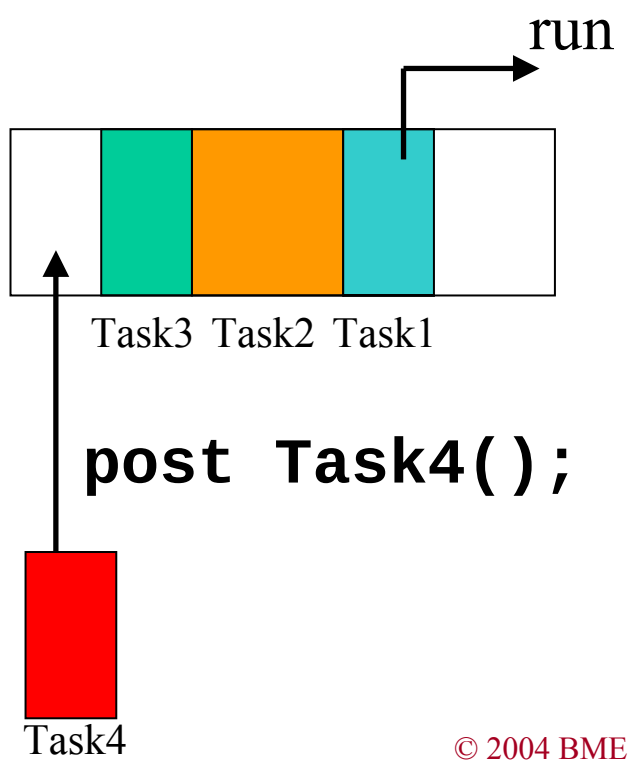
Taszkok deklarálása:

```
task void taskname() {
   ...
}
```

Taszkok indításának kérése:

```
{
 ...
 post taskname();
 ...
}
```

Task-Queue (FIFO):



Konkurencia modell

1. Taszkok között nincs kiürítés.

2. HW eseményeket (IT) kezelő függvények viszont megszakítást okoznak!

Szinkron kód (SC):

A kódnak azon része (függvények, event-ek, command-ok, taszkok), amely csak taszkokból érhető el.

Aszinkron kód (AC):

A kódnak azon része, amely elérhető legalább egy IT kezelőből.

Versenyhelyzet:

SC – SC ✓

SC – AC !

AC – AC !

Kölcsönös kizárás

Közös erőforrások védelme:

1. **Elérés csak szinkron kódból.**
2. **Atomikus utasítások használata.**

Atomikus utasitások:

```
atomic {
// védett szakasz
   ...
}
```

Késleltetik az interrupt végrehajtást:
- válaszidőt növelik, jittert okozhatnak
- command hivás és event küldés kerülendő (válaszidő a használt komponenstől függ)

Aszinkron kód jelzése:

```
async event ...

async command ...
```

Fordító figyelmeztet a potenciális versenyhelyzetre

Kétfázisú műveletek

Egyszerű taszk filozófia előnye:

Nincsenek blokkolódó hívások

→ Nem kell kontextus váltás

→ Nem kellenek szinkronizációs primitívek

Hátránya:

Nincsenek blokkolódó hívások

→ alkalmazás logikája bonyolultabbá válik

→ kétfázisú (split-phase) műveletek kellenek

1. **Kérés – command**
2. **Válasz/eredmény – event**

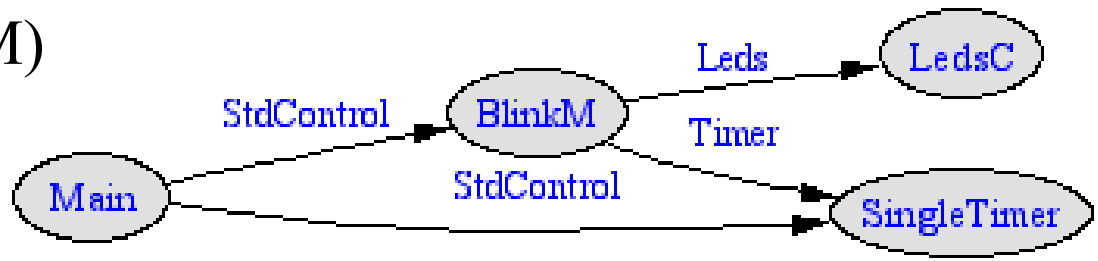
Példaprogram: Blink

Cél:

- LED-ek villogtatása
- Gyakoriságot óra vezérli

Felhasznált modulok:

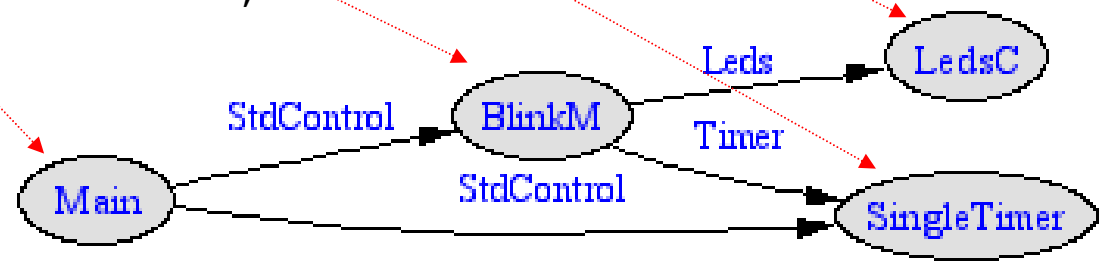
- Main
- LED-vezérlő (LedsC)
- Óra (SingleTimer)
- Villogtató (BlinkM)



Blink: Konfigurációs file

Blink.nc

```
configuration Blink {
}

implementation {
  components Main, BlinkM, SingleTimer, LedsC;

  Main.StdControl -> SingleTimer.StdControl;
  Main.StdControl -> BlinkM.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```



Blink: Modul file

BlinkM.nc

Szolgáltatott command-ok

```
module BlinkM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
  }
}
```

Felhasznált event-ek

```
implementation {

command result_t StdControl.init() {
    call Leds.init();
    return SUCCESS;
  }
command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 1000);
  }

command result_t StdControl.stop() {
    return call Timer.stop();
  }

event result_t Timer.fired()
  {
    call Leds.redToggle();
    return SUCCESS;
  }
}
```

Blink: Interfészek

StdControl

```
interface StdControl {
  command result_t init();
  command result_t start();
  command result_t stop();
}
```

Leds

```
interface Leds {
    command result_t init();
    command result_t redOn();
    command result_t redOff();
    command result_t redToggle();
    ...
    command uint8_t  get();
    command result_t set(uint8_t value);
}
```

TIMER_REPEAT, TIMER_ONE_SHOT

1/1024 sec egységekben

```
interface Timer {
    command result_t start(char type, uint32_t interval);
    command result_t stop();
    event   result_t fired();
}
```

Timer

Blink: SingleTimer

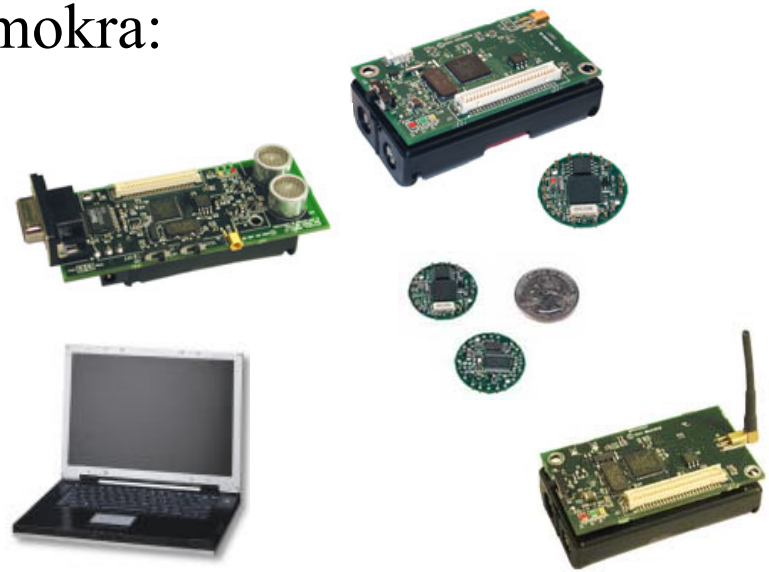
SingleTimer.nc

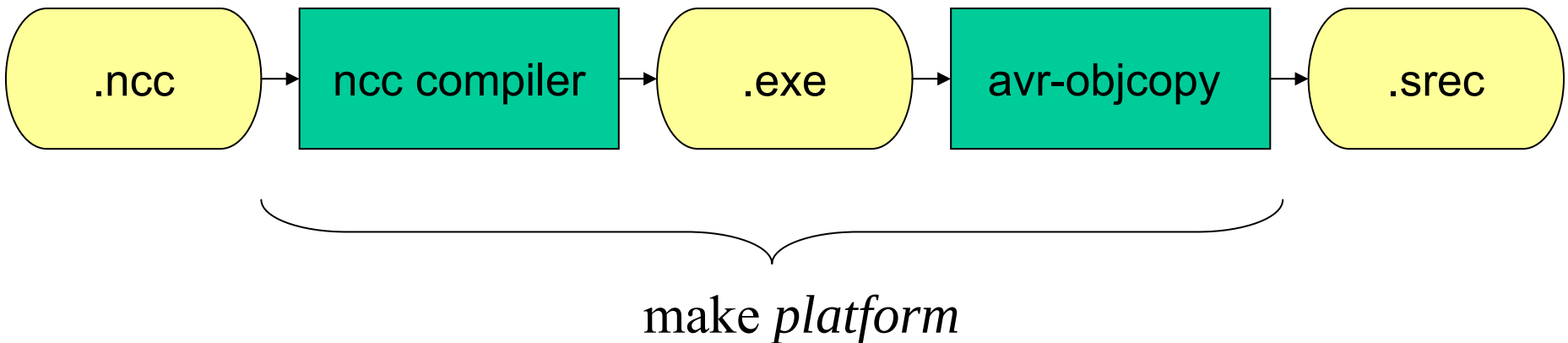
```
configuration SingleTimer {
  provides interface Timer;
  provides interface StdControl;
}

implementation {
  components TimerC;

  Timer = TimerC.Timer[unique("Timer")];
  StdControl = TimerC;
}
```

Fordítás

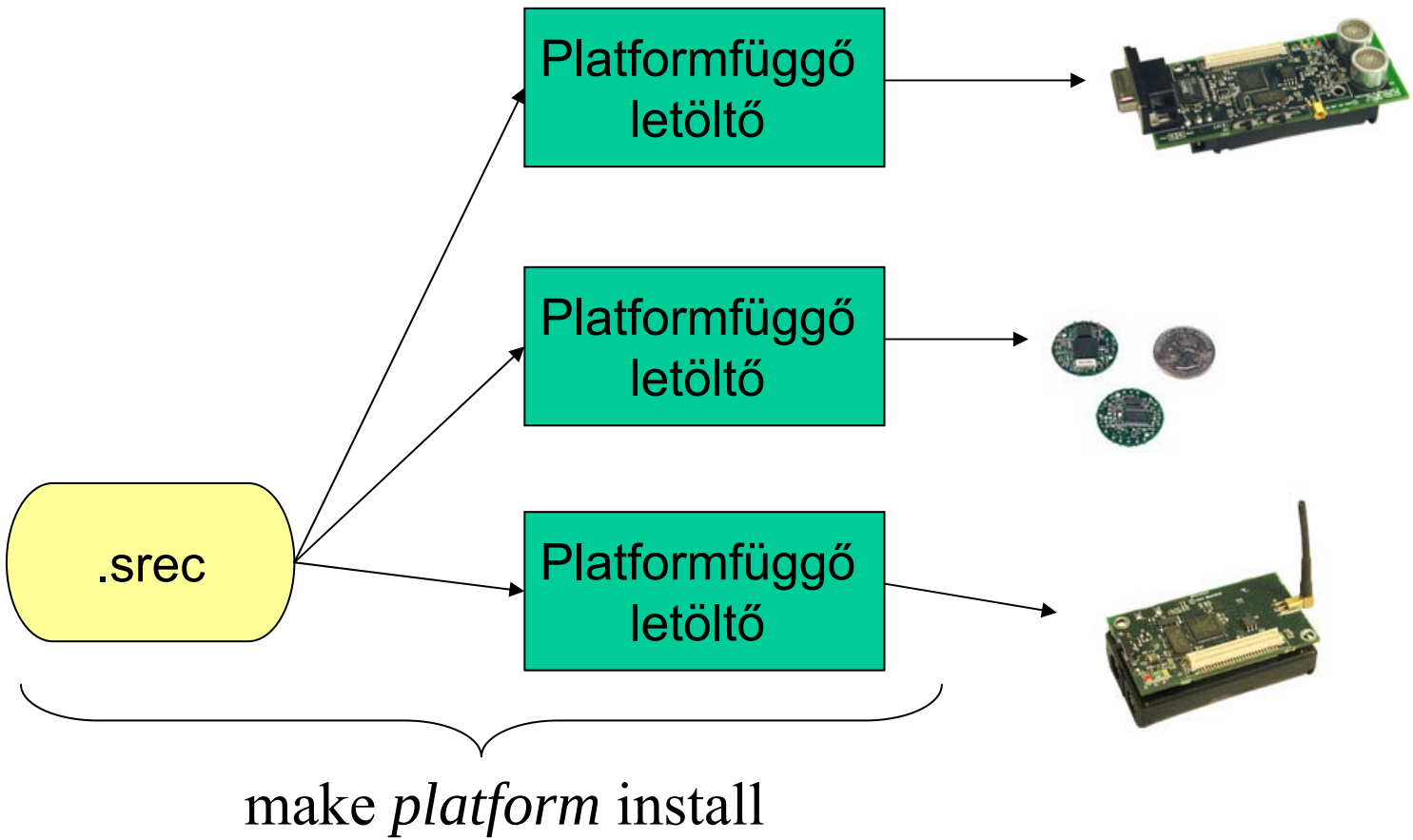
Fordítás történhet különböző platformokra:

- ✓avrmote
- ✓mica
- ✓mica2
- ✓mica2dot
- ✓mica128
- ✓pc





Futtatás



TOSSIM szimulátor

- Kód változtatás nélkül futtatható PC-n
- Tetszőleges számú egyed (szimulált mote) létrehozható
- Minden mote ugyanazt a programot futtatja

- Fordítás: `make pc`
- Futtatás: `build/pc/main.exe [options] 10`

10 példányban fut

- Kimenet: szöveges debug-üzenetek
- Kijelzett események köre változtatható
- A kódba debug-üzenetek külön is beépíthetők
- Rádió kapcsolati mátrix beállítható

TOSSIM példa

Rádió üzenetek, ledek állapota

```
$ make pc

$ export DBG=am,led
$ build/pc/main.exe -rf=my-lossy-matrix3.nss 3
```

3 mote

Veszteséges rádió modell

Beépített debug-üzenetek

```
$ export DBG=usr1
$ build/pc/main.exe -rf=my-lossy-matrix6.nss 6
```

6 mote