Modellek ellenőrzése

Rendszermodellezés

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Ariane 5 hordozórakéta

- A legerősebb európai hordozórakéta

Ariane 5 hordozórakéta

- 1996. június 4-én 37 másodperccel a kilövés után megsemmisítette magát (F-501)
  - A szállított négy műhold is megsemmisült
  - $370 milliós kár

Ariane 5 hordozórakéta

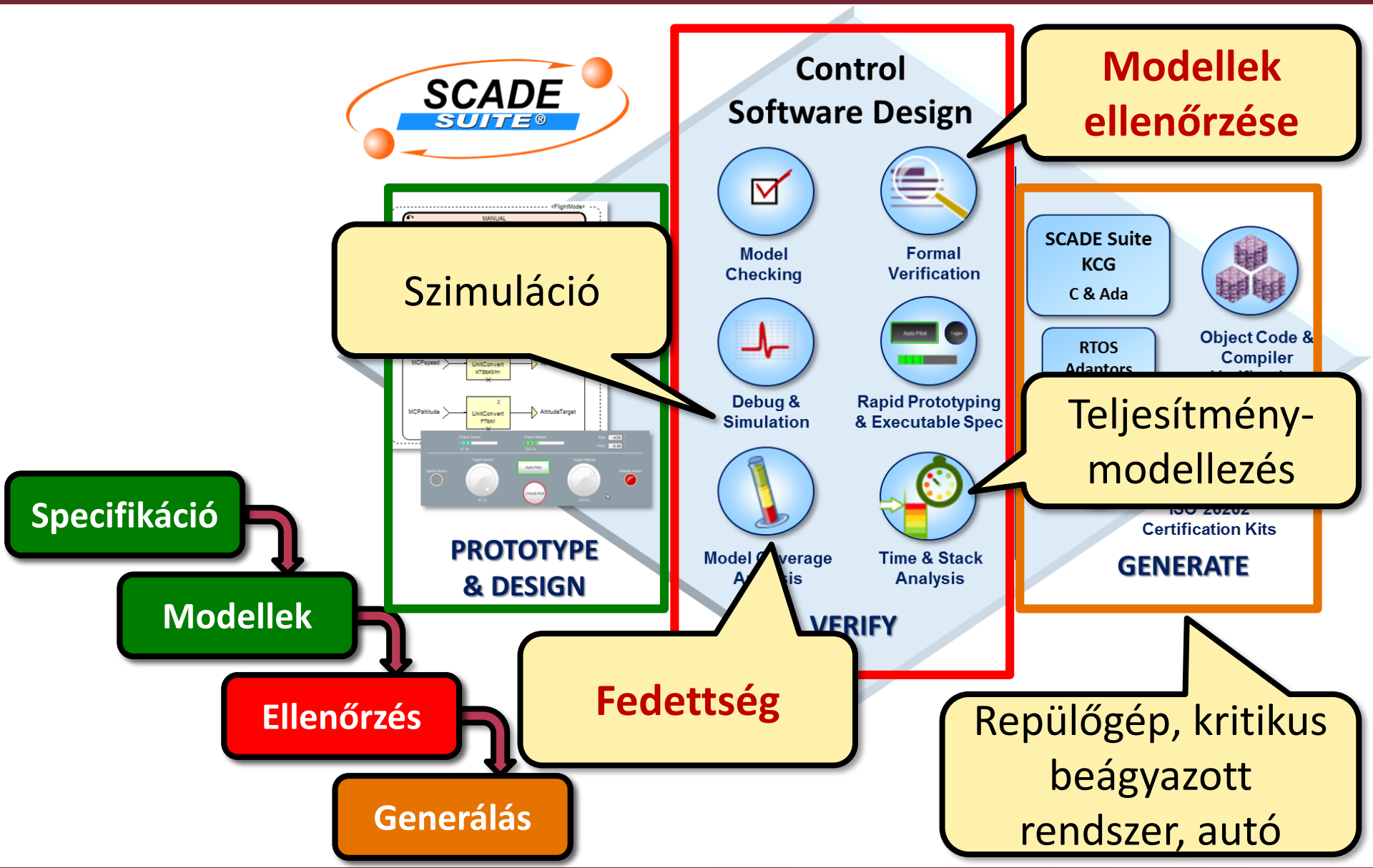
- 1996. június 4-én 37 másodperccel a kilövés után megsemmisítette magát (F-501)
  - A szállított négy műhold is megsemmisült
  - $370 milliós kár
- A világ (egyik) legdrágább szoftverhibája
  - Elsődleges ok:

    64 bites szám sikertelen konverziója 16 bitesre
  - Másodlagos ok:

    Soha nem tesztelték együtt a modulokat

    (+előző verzió tervezői döntései, követelmények rossz értelmezése)

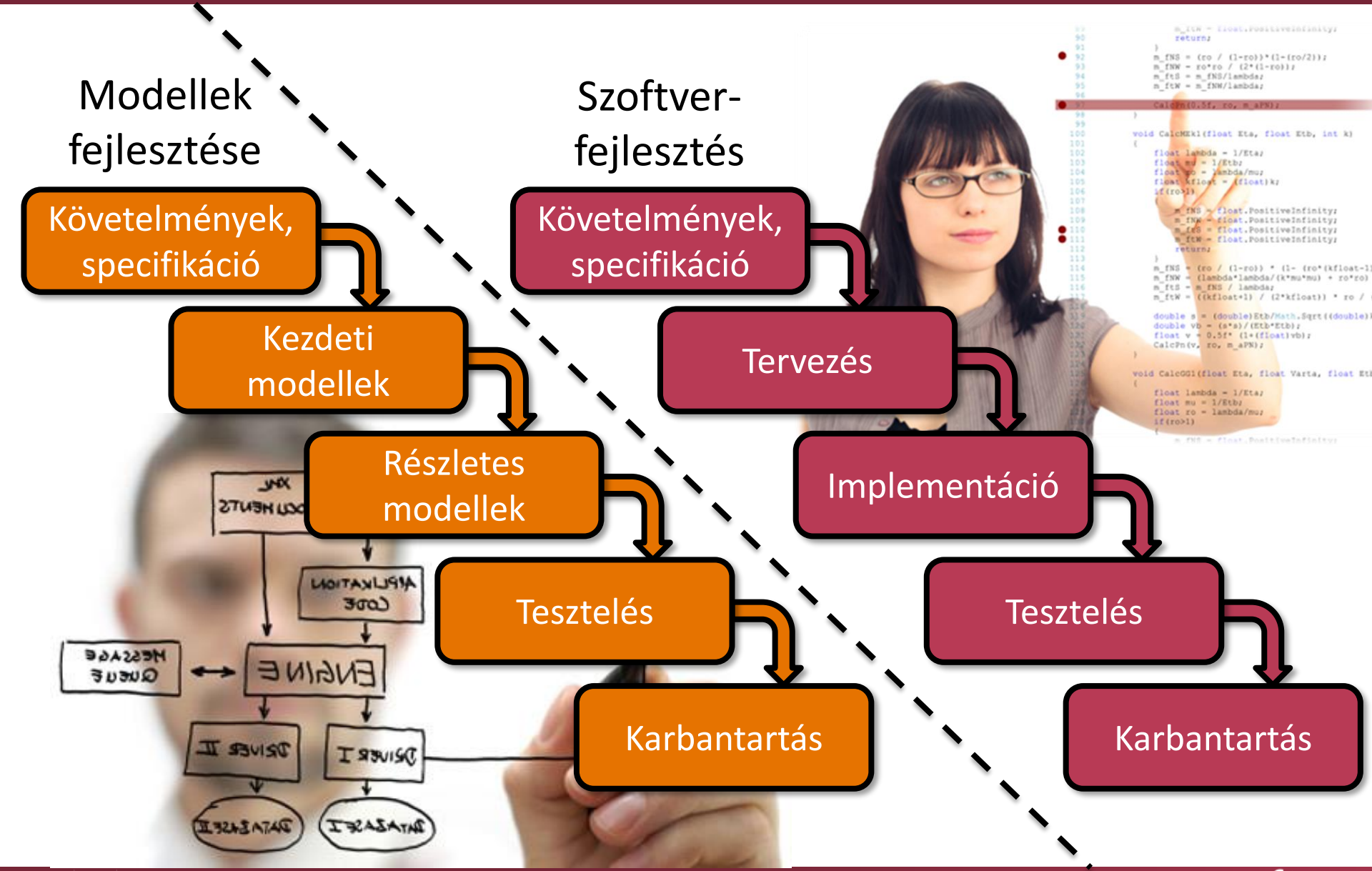
Példa: Esterel SCADE



Tartalom

- Alapfogalmak
- Statikus ellenőrzés
- Tesztelés
- Formális verifikáció

Motiváció: modellek életciklusa



Kódgenerálás esetén

Modellek fejlesztése

Szoftver-fejlesztés

Követelmények, specifikáció

Kezdeti modellek

Követelmények, specifikáció

Implementáció

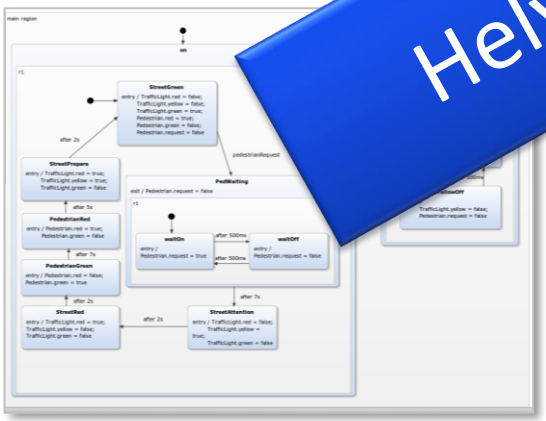
Tesztelés

Tesztelés

Karbantartás

Karbantartás

Helyes modell, helyesebb kód



ALAPFOGALMAK

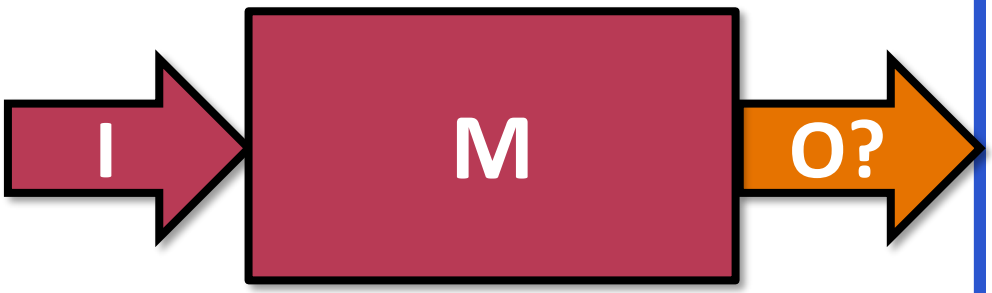
Modellek és feladatok

- Szintézis:

  Specifikációnak megfelelő modell?

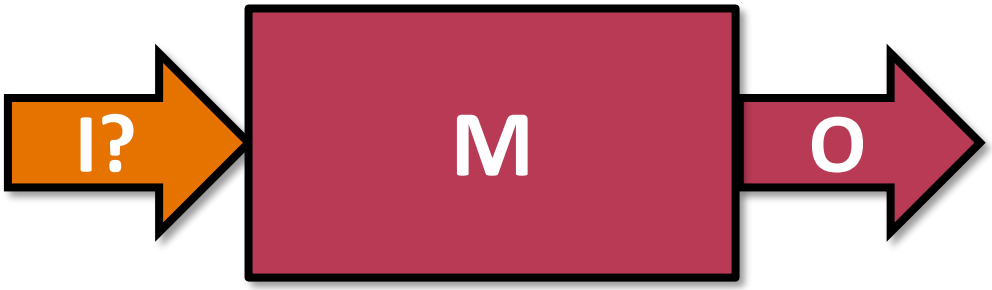
- Analízis:

  Modell viselkedése?



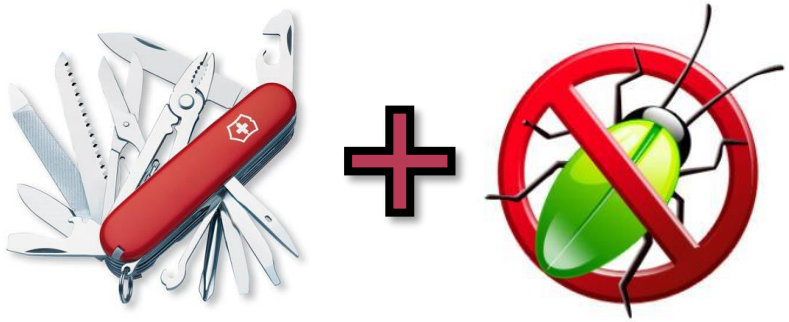
- Vezérlés:

  Kívánt állapot hogy érhető el?



Helyesség

- **Helyesség:**
  modell vagy kód megfelelése a követelményeknek.
  - **Funkcionális helyesség:**
    megfelelés a funkcionális követelményeknek
  - Nemfunkcionális követelmények ellenőrzése:
    lásd teljesítménymodellezés előadás
- **Szempontok:**
  - Mindig képes teljesíteni a feladatot
  - Hibamentes
  - Nincs tiltott viselkedés



Funkcionális követelmények csoportosítása

- **Megengedett** viselkedés:
  - Milyen állapotokban lehet/nem lehet a rendszer
  - Milyen viselkedés tilos
  - Univerzális követelmények
    - Mindig igaznak kell lenniük
- **Elvárt** viselkedés:
  - Milyen állapotokba kell tudni eljutni
  - Milyen funkciókat kell tudnia a rendszernek
  - Egzisztenciális követelmények
    - Lehessen lehetőség a teljesülésükre

Funkcionális követelmények csoportosítása

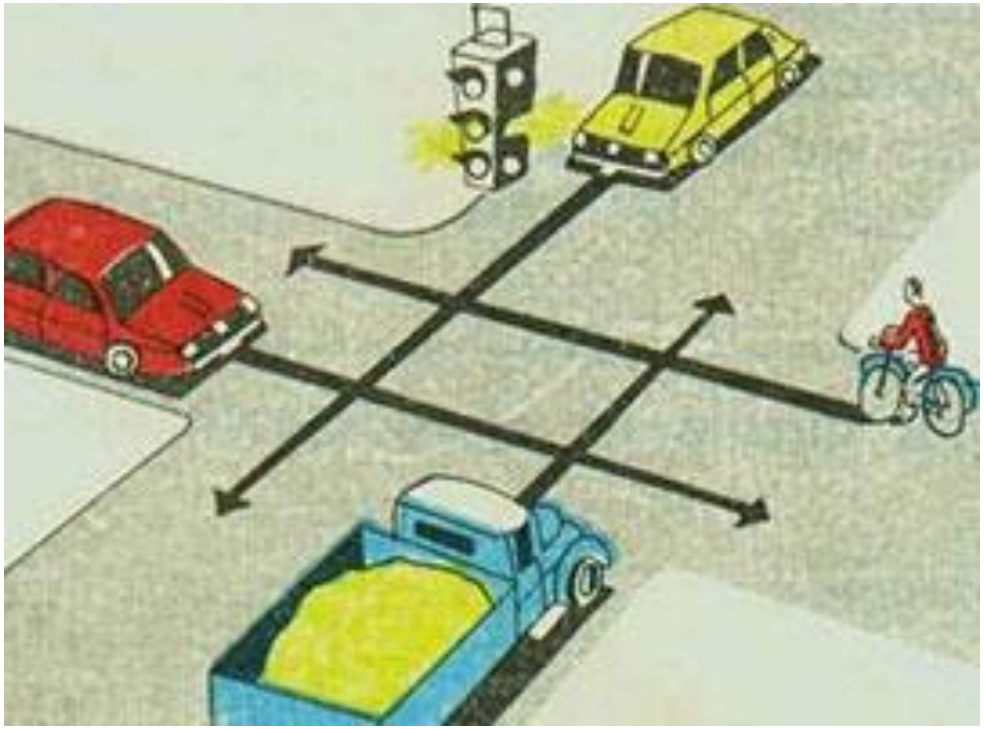
- **Megengedett** viselkedés:
  - Milyen állapotokban
  - Milyen viselkedés tilos
  - Univerzális követelmények
    - Mindig igaznak kell lenniük
- **Elvárt** viselkedés:
  - Milyen állapotokba
  - Milyen funkciókat kell tud
  - Egzisztenciális követelménye
    - Lehessen lehetőség a teljesül

„Egy kereszteződés lámpái **soha nem lehetnek** egyszerre zöldek.”

„A lámpa **legyen képes** zöldre váltani.”

Holtpont

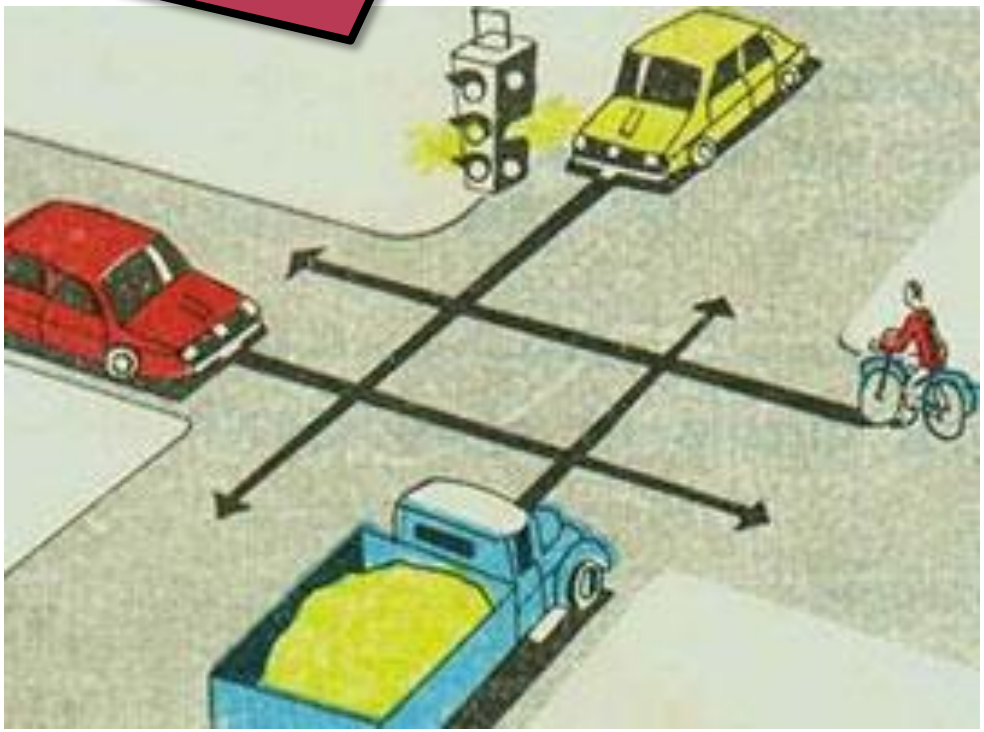
- Beragadt **állapot**:
  a rendszer csak külső segítséggel képes kilépni.
  - Pl. egymásra várakozó folyamatok miatt



Holtpont (deadlock)

- „Útkereszteződésben - ha közúti jelzésből vagy forgalmi szabályból más nem következik - a jobbról érkező járműnek van elsőbbsége.” (1988. évi I. törvény a közúti közlekedésről)

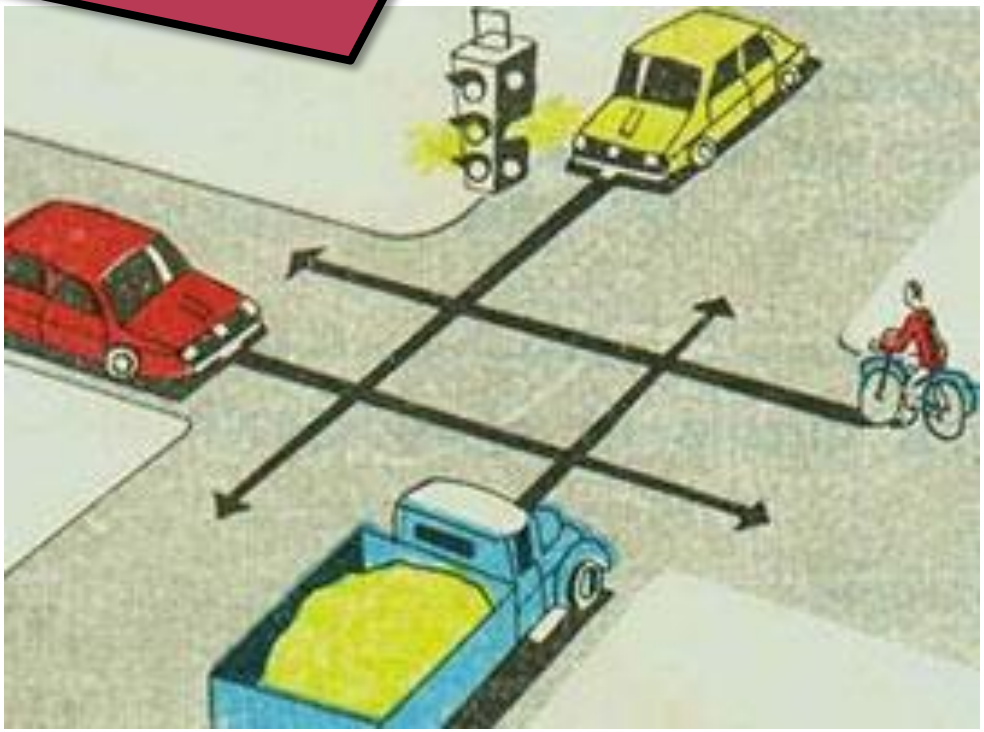
?



?

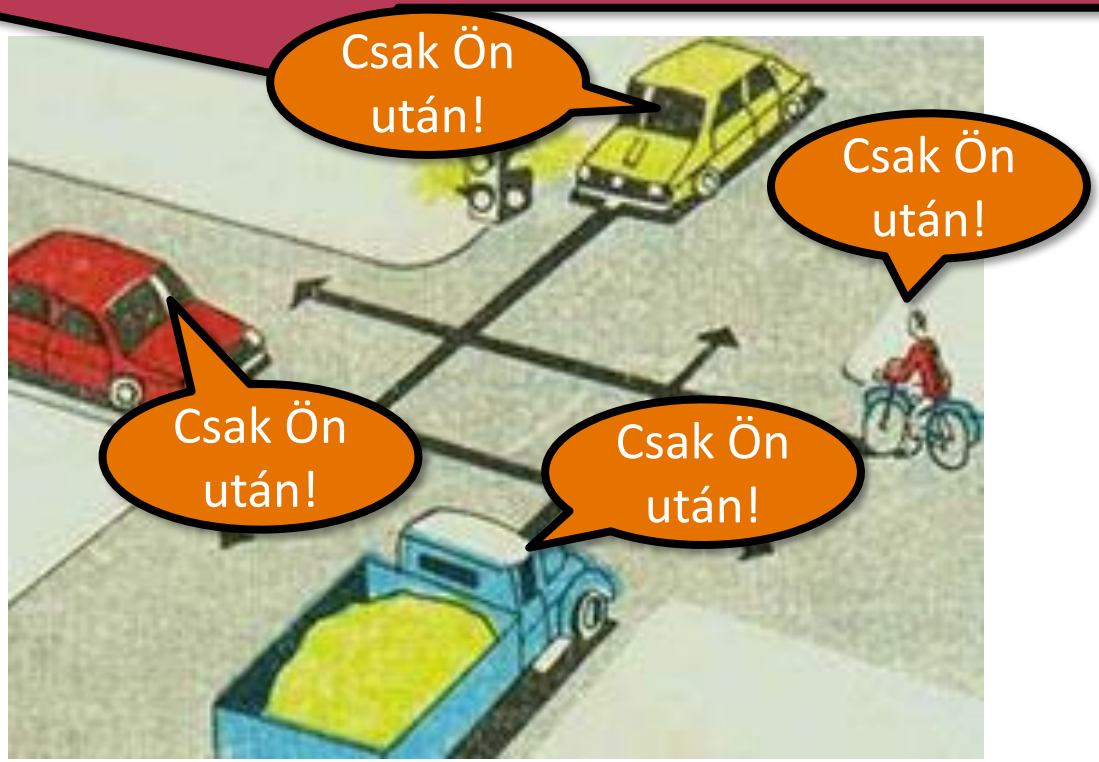
Holtpont feloldása

- „Ha 4 autó egyszerre ér a jobbkezes kereszteződésbe, akkor valamelyiknek le kell mondania az elsőbbségről és elengedni a másikat. Ha nem teszi, akkor a KRESZ szerint ott fognak állni örökké.” (gyakorikerdesek.hu)



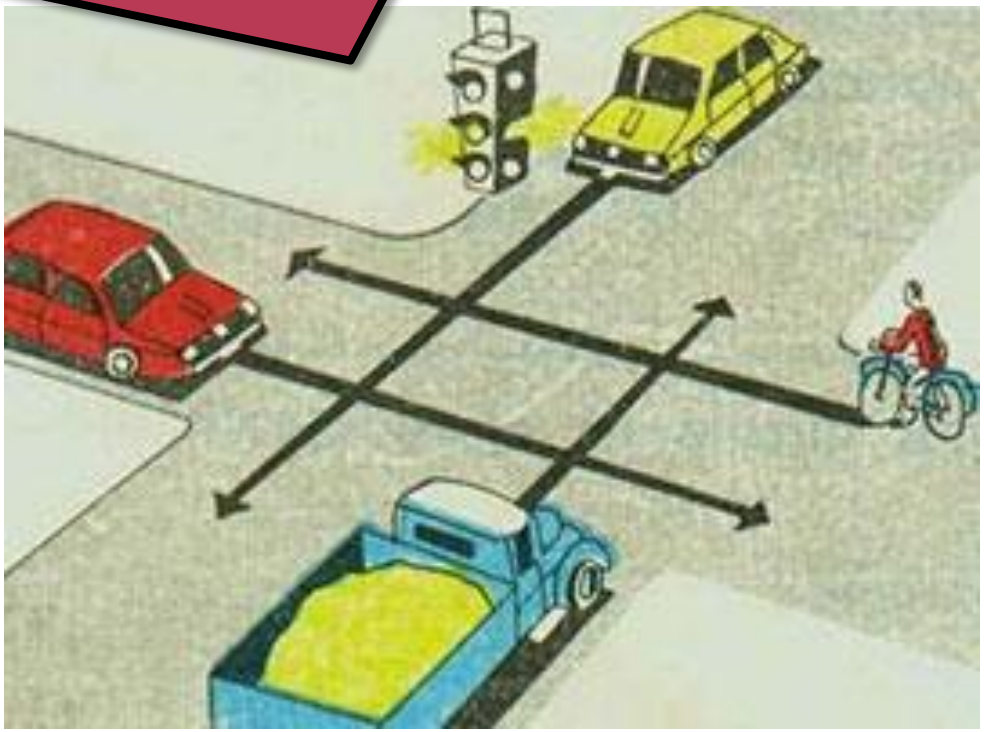
Holtpont feloldása

- „Ha 4 autó egyszerre ér a jobbkezes kereszteződésbe, akkor valamelyiknek le kell mondania az elsőbbségről és elengedni a másikat. Ha nem teszi, akkor a KRESZ szerint ott fognak állni örökké.” (gyakorikerdesek.hu)



Újabb holtpont

- „Ha 4 autó egyszerre ér a jobbkezes kereszteződésbe, akkor valamelyiknek le kell mondania az elsőbbségről és elengedni a másikat. Ha nem teszi, akkor a KRESZ szerint ott fognak állni örökké.” (gyakorikerdesek.hu)

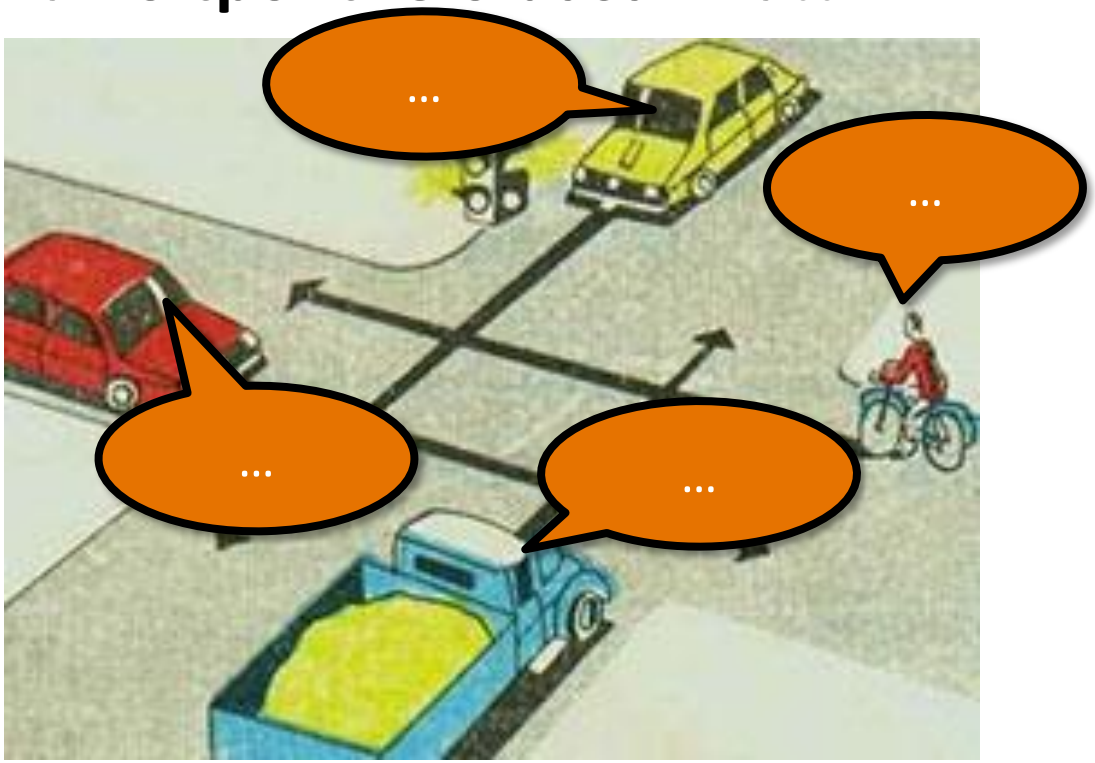


Holtpont feloldása

- „Útkereszteződésben - ha közúti jelzésből vagy forgalmi szabályból más nem következik - a jobbról érkező járműnek van elsőbbsége.” (1988. évi I. törvény a közúti közlekedésről)

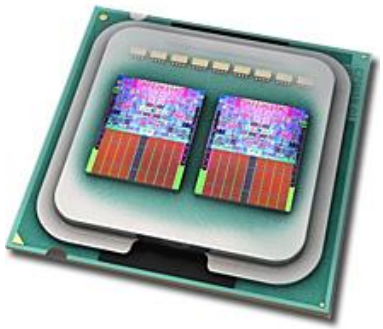
Végtelen ciklus (livelock)

- Beragadt **állapothalmaz**:
  a rendszer csak külső segítséggel/a modelltől való eltéréssel képes kilépni.
  - Pl. éppen a **holtpont feloldása** miatt



Holtpont (deadlock)

- Beragadt **állapot**:
  a rendszer csak külső segítséggel képes kilépni.
  - Pl. egymásra várakozó folyamatok miatt
- Gyakori tervezési hiba párhuzamos rendszereknél
  - Sokszor nehéz elkerülni, feloldani
    - A jónak hitt megoldás is problémát okozhat
  - Nehéz tesztelni, látszólag véletlenszerű is lehet
  - „Többmagos CPU krízis”

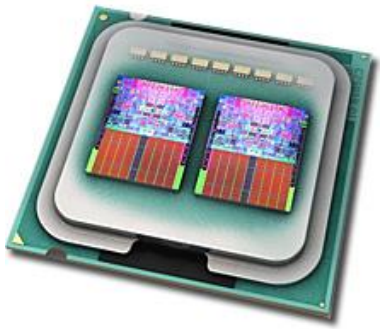


Holtpont (deadlock)

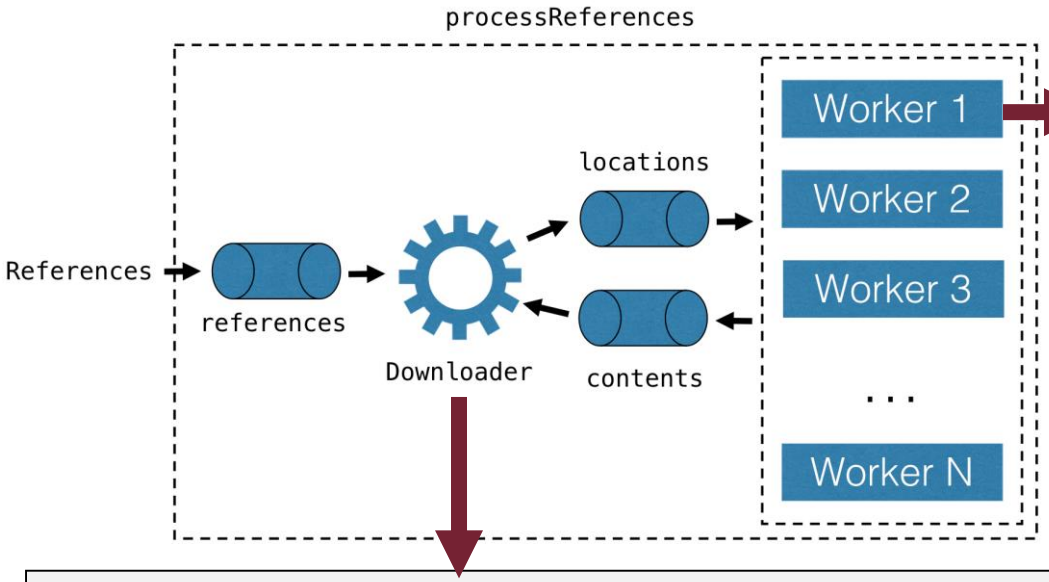
- Beragadt **állapot**:
  a rendszer csak külső s
  - Pl. egymásra várakozó fo
- Gyakori tervezési hiba párhuzamos rendszereknél
  - Sokszor nehéz elkerülni oldani
    okozhat
  - N nszerű is lehet
  - „

„Két folyamatnak üzenetet kell cserélnie, de mind a kettő a másik üzenetére vár.”

„Két folyamat mindegyikének kétféle erőforrás kell a továbblépéshez, de mindegyikük egyet-egyet lefoglalt.”



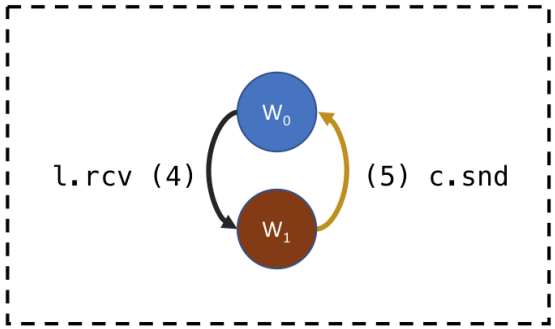
Példa: Kommunikáló SW komponensek



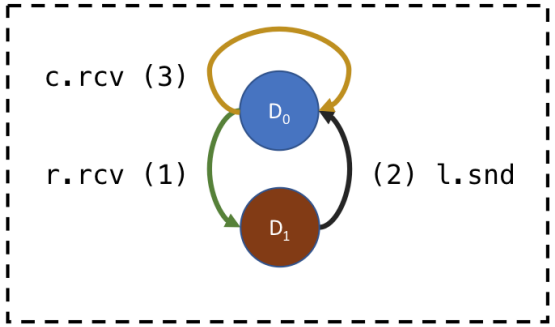

```
for (loc in locations) {          // (4)
  val content = downloadContent(loc)
  contents.send(                  // (5)
    LocContent(loc, content)
  )
}
```

```
while (true) {
  select<Unit> {
    references.onReceive {                    // (1)
      ref ->
        val loc = ref.resolveLocation()
        //...
        locations.send(loc)                   // (2)
      }
    contents.onReceive {                      // (3)
      (loc, content) -> //...
    }
  }
}
```

Worker

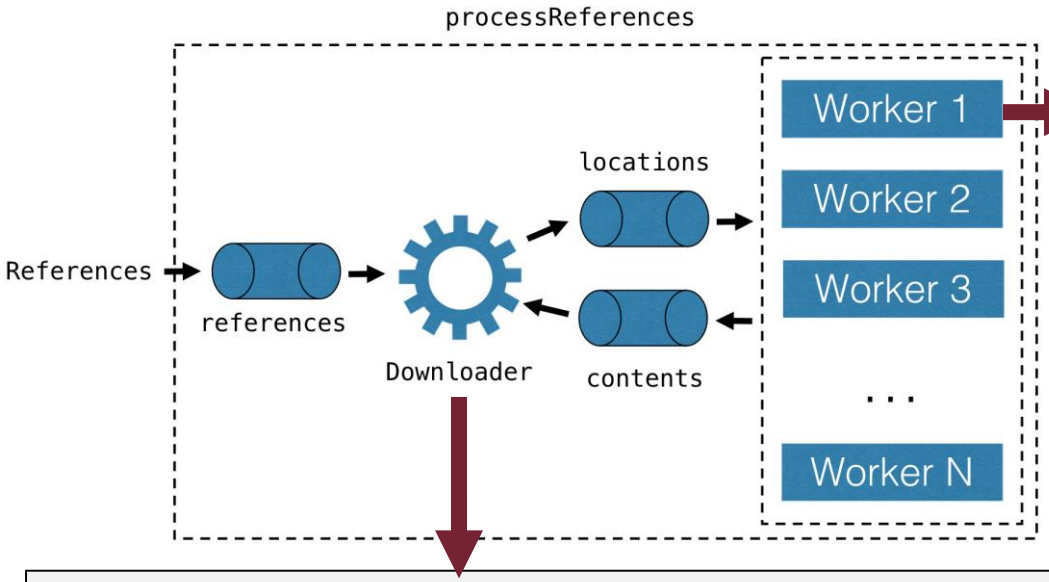


Downloader



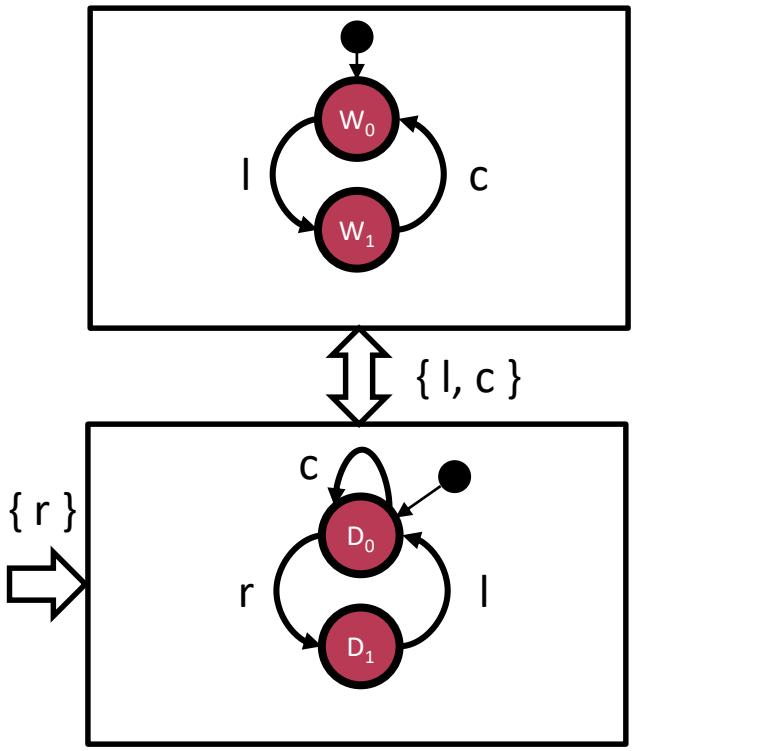
Forrás: https://medium.com/@elizarov/deadlocks-in-non-hierarchical-csp-e5910d137cc

Példa: Kommunikáló SW komponensek



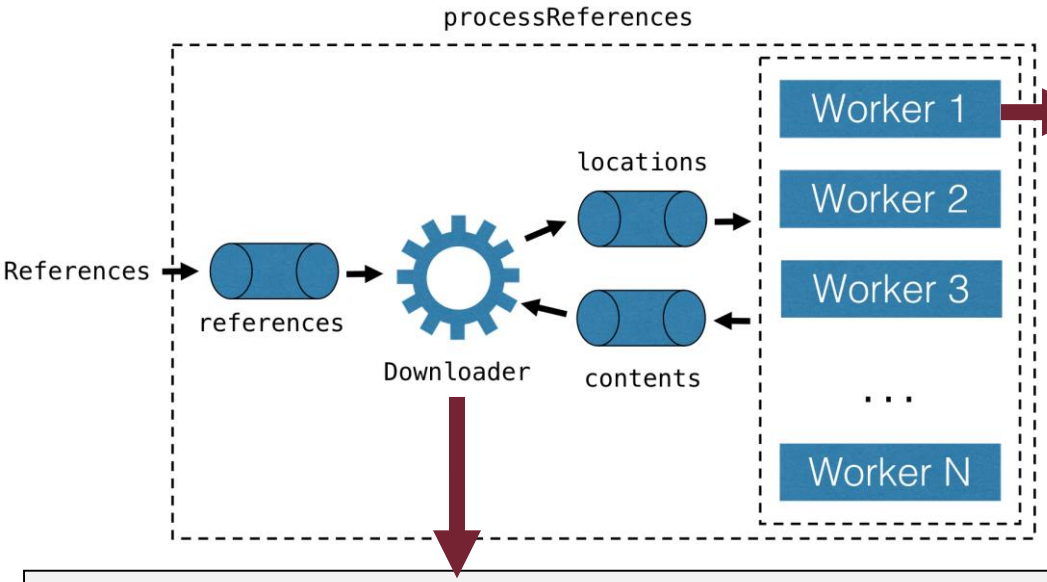

```
for (loc in locations) {          // (4)
  val content = downloadContent(loc)
  contents.send(                  // (5)
    LocContent(loc, content)
  )
}
```

```
while (true) {
  select<Unit> {
    references.onReceive {                    // (1)
      ref ->
        val loc = ref.resolveLocation()
        //...
        locations.send(loc)                   // (2)
      }
    contents.onReceive {                      // (3)
      (loc, content) -> //...
    }
  }
}
```



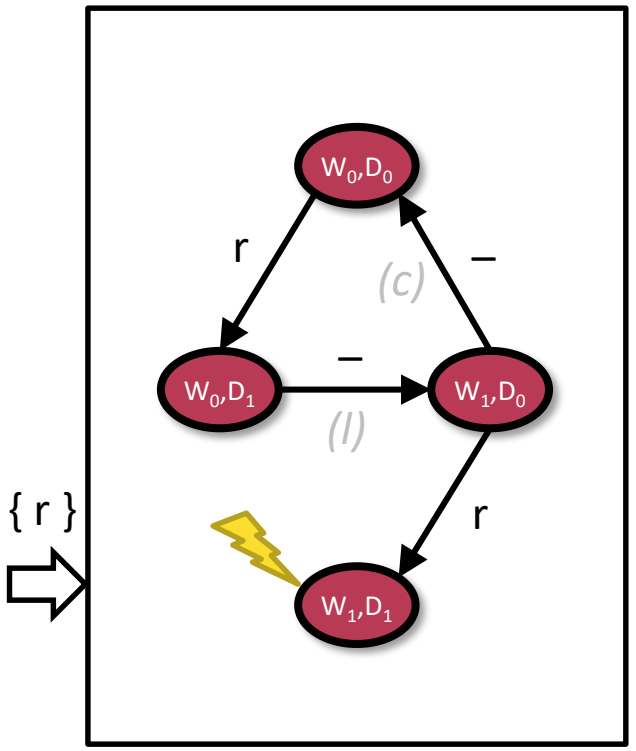
Forrás: https://medium.com/@elizarov/deadlocks-in-non-hierarchical-csp-e5910d137cc

Példa: Kommunikáló SW komponensek



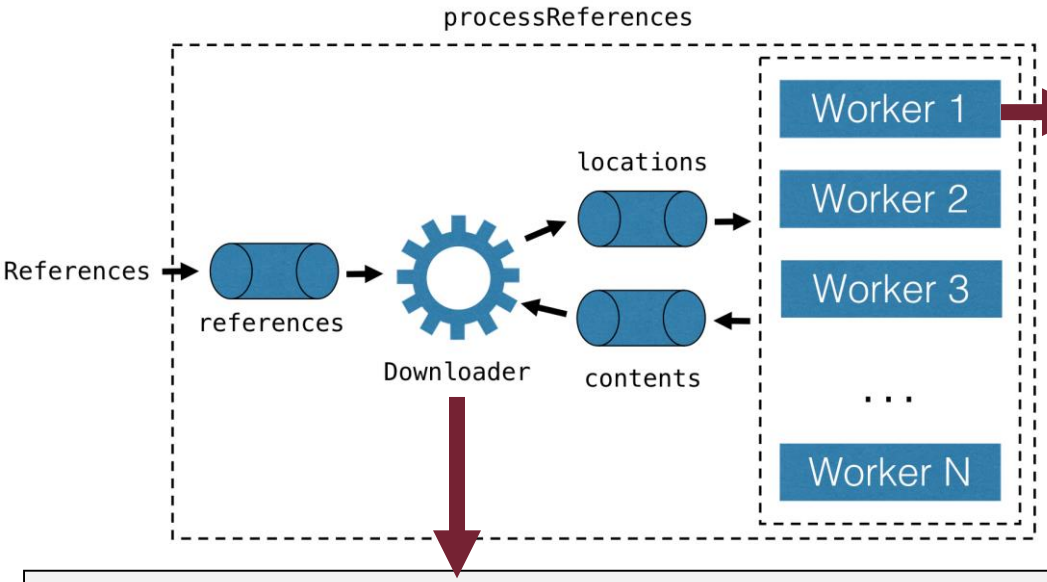

```
for (loc in locations) {          // (4)
  val content = downloadContent(loc)
  contents.send(                  // (5)
    LocContent(loc, content)
  )
}
```

```
while (true) {
  select<Unit> {
    references.onReceive {                    // (1)
      ref ->
        val loc = ref.resolveLocation()
        //...
        locations.send(loc)                   // (2)
      }
    contents.onReceive {                      // (3)
      (loc, content) -> //...
    }
  }
}
```



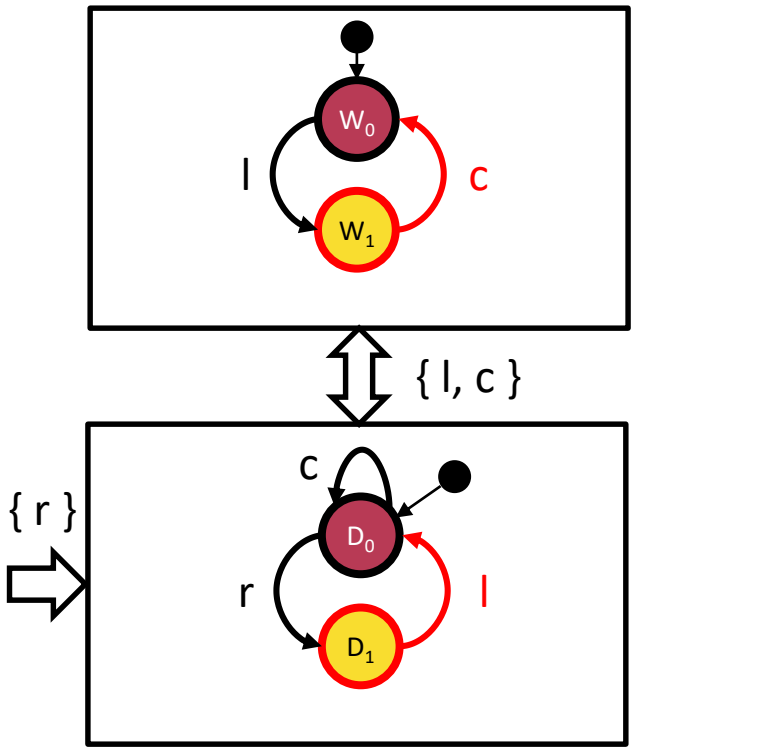
Forrás: https://medium.com/@elizarov/deadlocks-in-non-hierarchical-csp-e5910d137cc

Példa: Kommunikáló SW komponensek



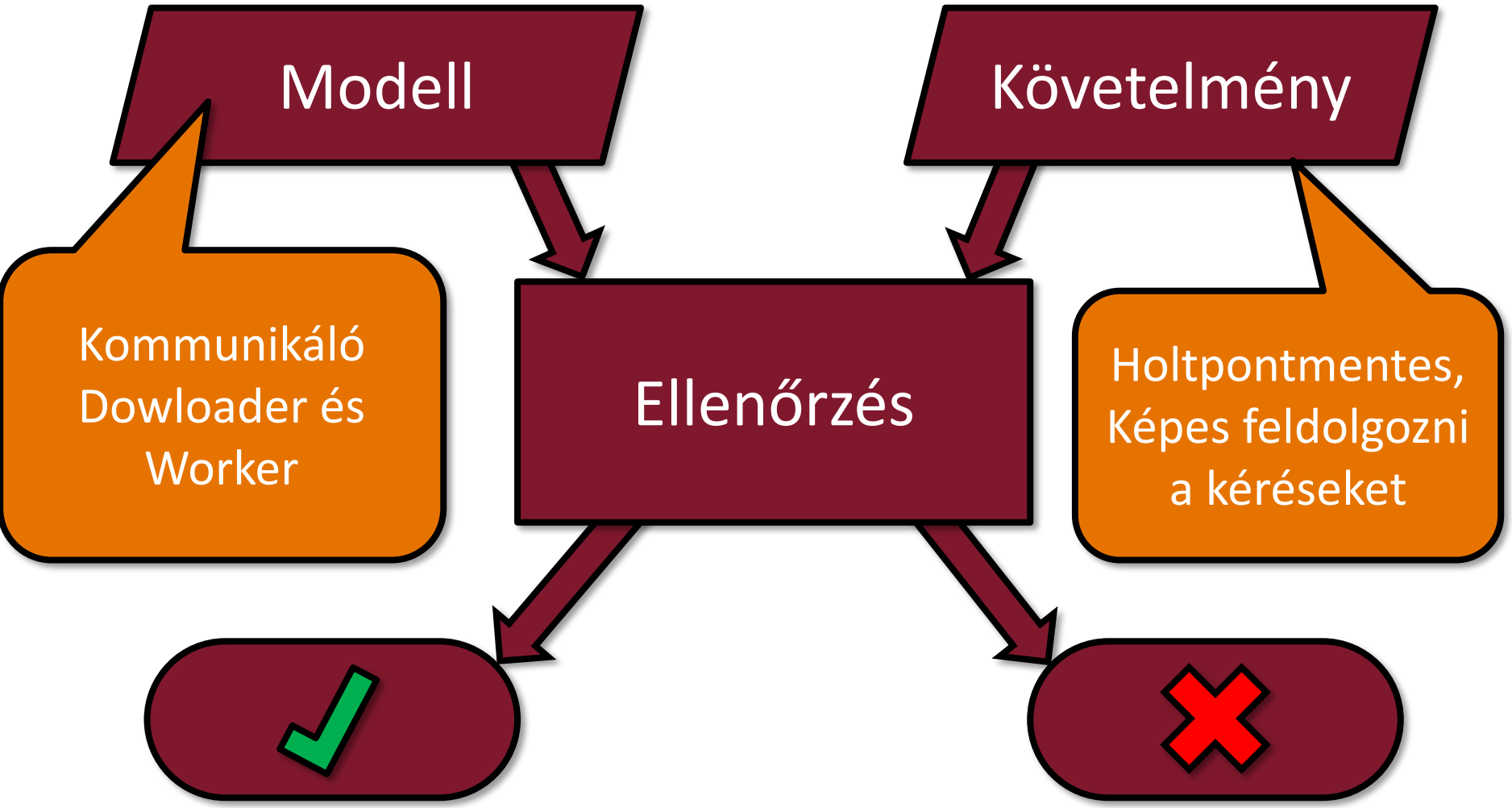

```
for (loc in locations) {          // (4)
  val content = downloadContent(loc)
  contents.send(                  // (5)
    LocContent(loc, content)
  )
}
```

```
while (true) {
  select<Unit> {
    references.onReceive {                    // (1)
      ref ->
        val loc = ref.resolveLocation()
        //...
        locations.send(loc)                   // (2)
      }
    contents.onReceive {                      // (3)
      (loc, content) -> //...
    }
  }
}
```



Forrás: https://medium.com/@elizarov/deadlocks-in-non-hierarchical-csp-e5910d137cc

Modellek ellenőrzése



Vizsgálatok fajtái

- Cél szerint:
  - **Verifikáció:**
    jól csinálom a rendszert?
    - Az implementáció megfelel a specifikációnak?
  - **Validáció:**
    jó rendszert csinálok?
    - A rendszer teljesíti a felhasználói követelményeket?
- Módszer szerint:
  - Statikus ellenőrzés
  - Dinamikus ellenőrzés
    - Szúrópróbaszerű (tesztelés, szimuláció)
    - Kimerítő/teljes (modellellenőrzés)

Alapfogalmak

Statikus ellenőrzés

Tesztelés

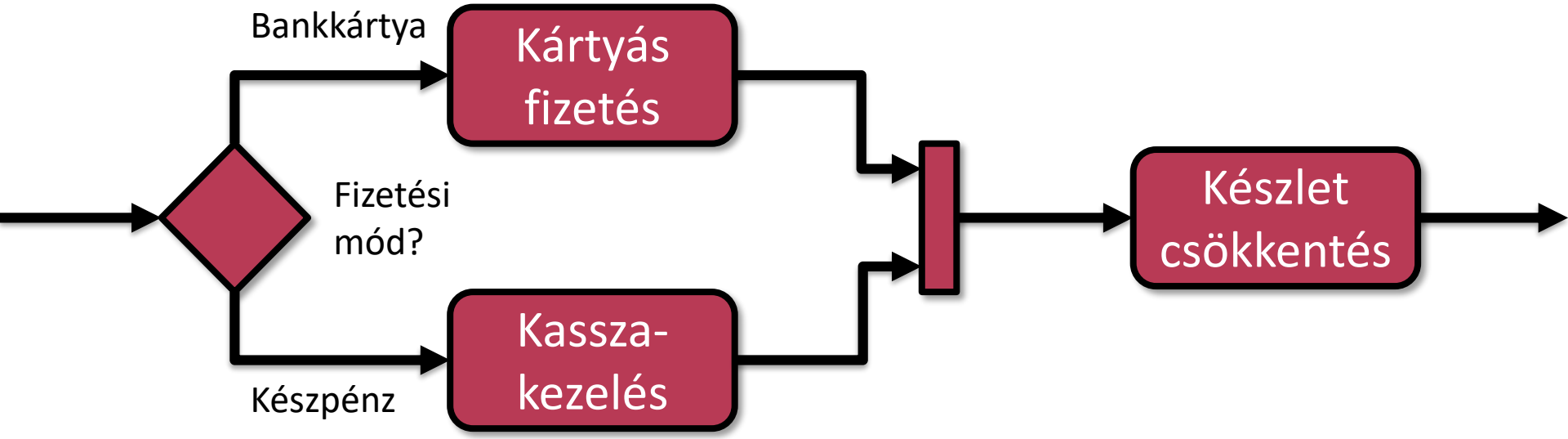
Formális verifikáció

STATIKUS ELLENŐRZÉS

Hibaminták

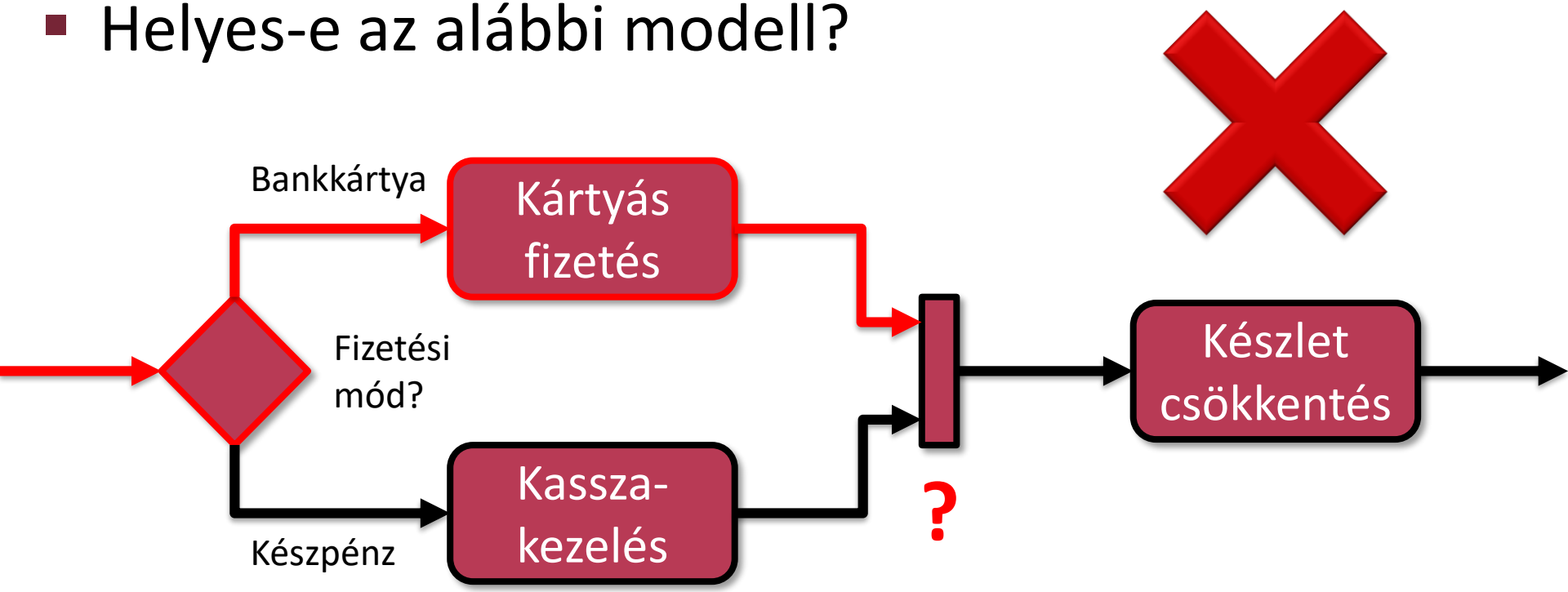
Decision és Join

- Helyes-e az alábbi modell?



Decision és Join

- Helyes-e az alábbi modell?

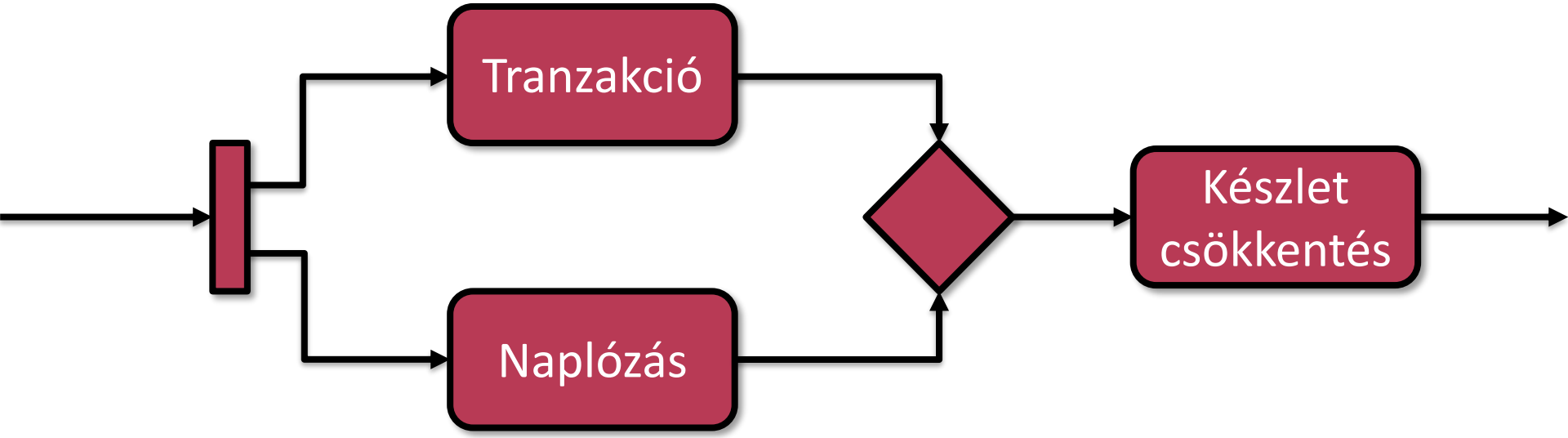


- Join: csak akkor léphet tovább, ha mindegyik bemeneten érkezett token

→ **DEADLOCK**

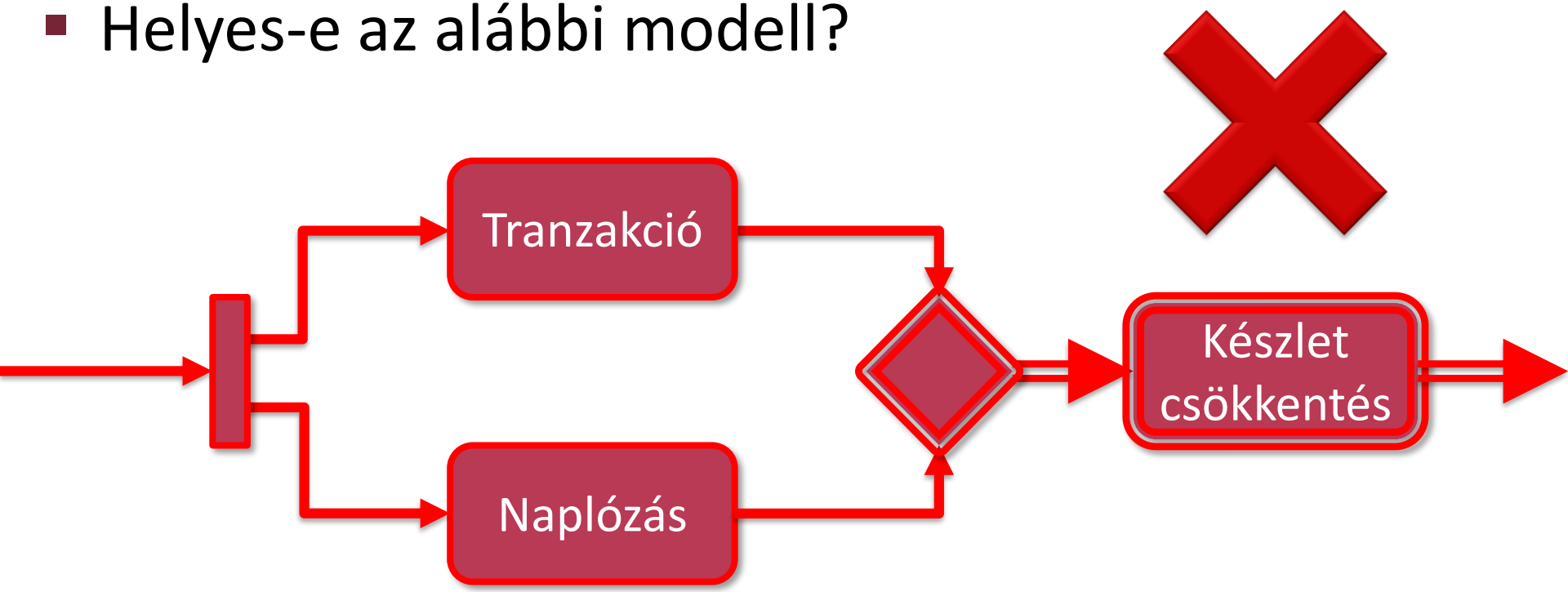
Fork és Merge

- Helyes-e az alábbi modell?



Fork és Merge

- Helyes-e az alábbi modell?

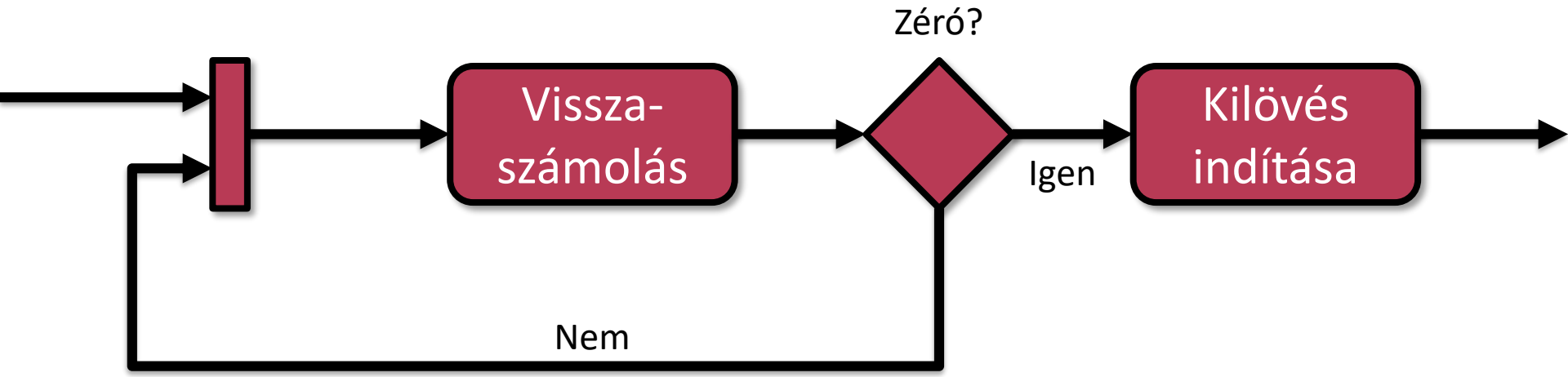


- Merge: bármelyik ágon érkező tokent átengedi
  - Nem szinkronizálja a szálakat

→ „Készlet csökkentés” kétszer fut le

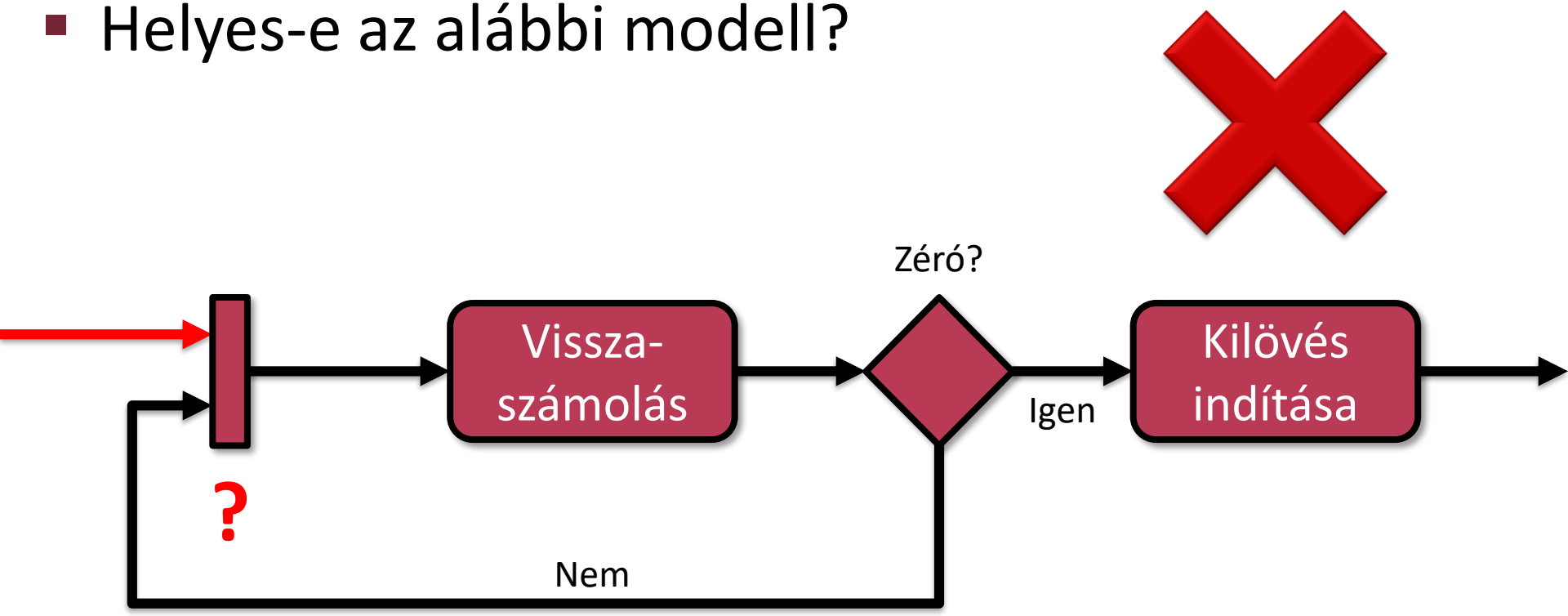
Ciklus 1.

- Helyes-e az alábbi modell?



Ciklus 1.

- Helyes-e az alábbi modell?

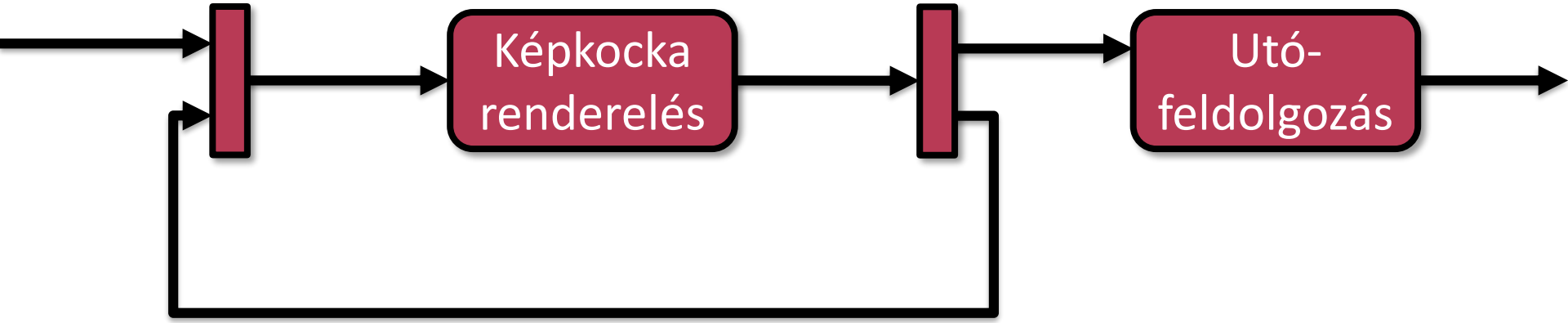


- Join: csak akkor léphet tovább, ha mindegyik bemeneten érkezett token

→ **DEADLOCK**

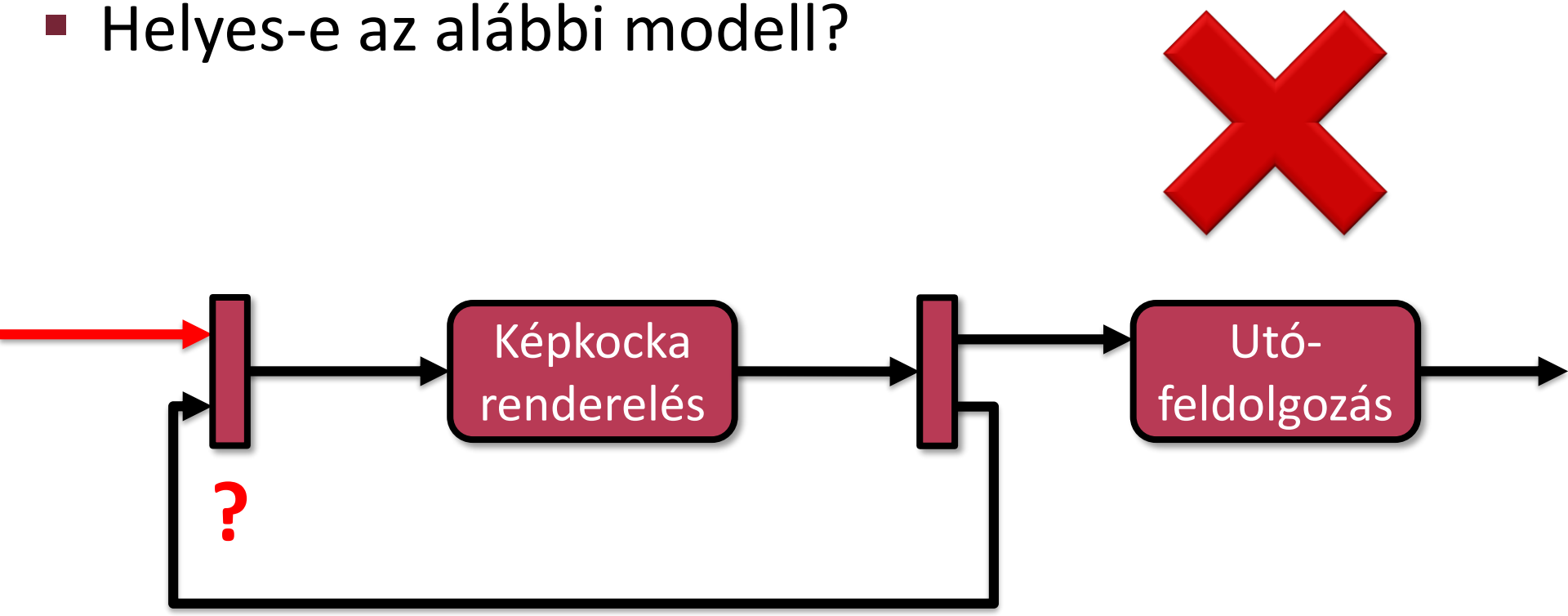
Ciklus 2.

- Helyes-e az alábbi modell?



Ciklus 2.

- Helyes-e az alábbi modell?

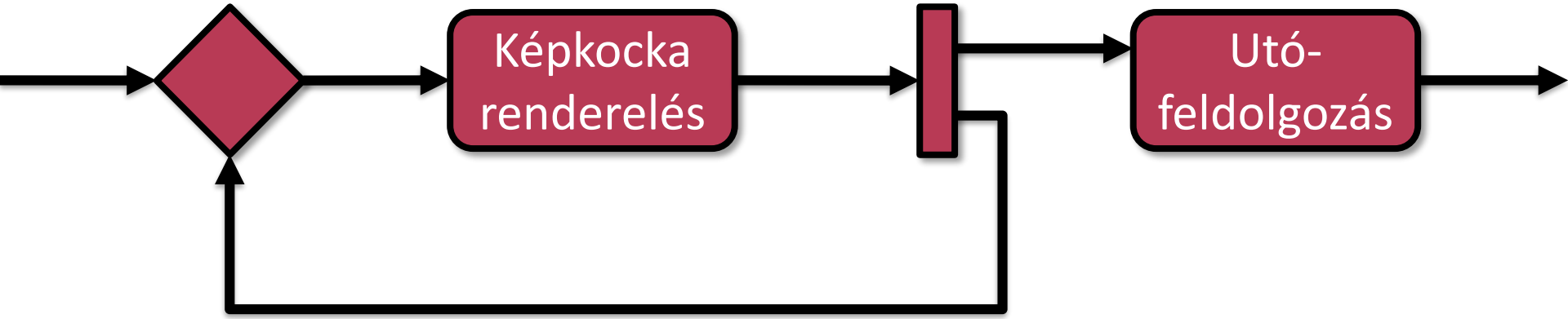


- Join: csak akkor léphet tovább, ha mindegyik bemeneten érkezett token

→ **DEADLOCK**

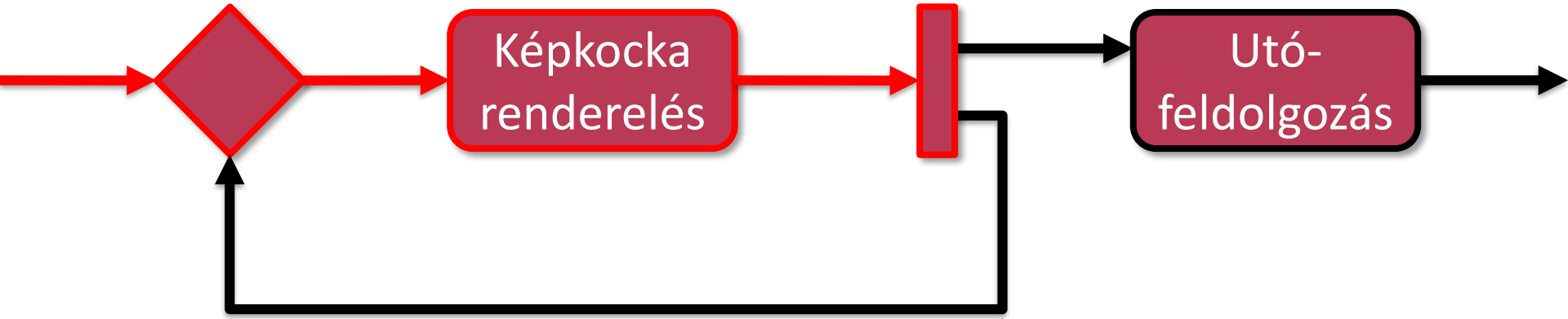
Ciklus 3.

- Helyes-e az alábbi modell?



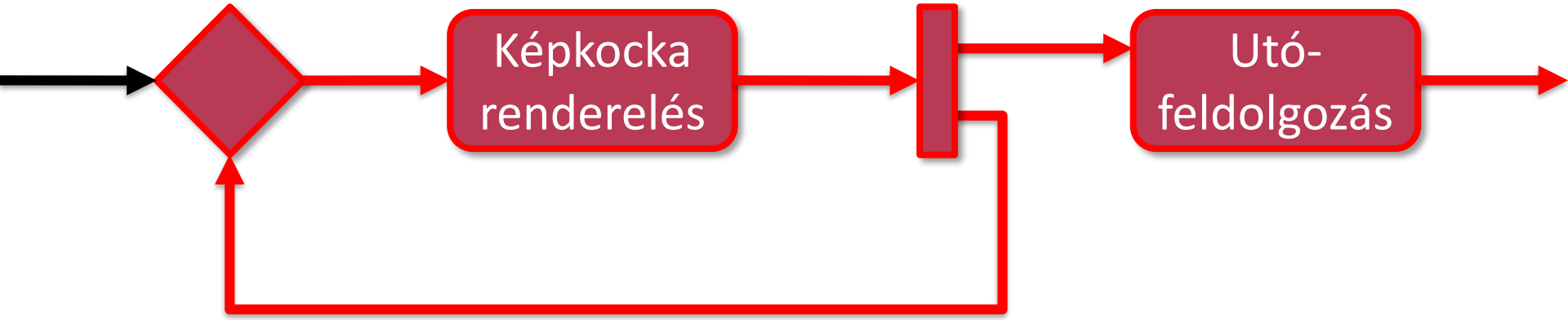
Ciklus 3.

- Helyes-e az alábbi modell?



Ciklus 3.

- Helyes-e az alábbi modell?

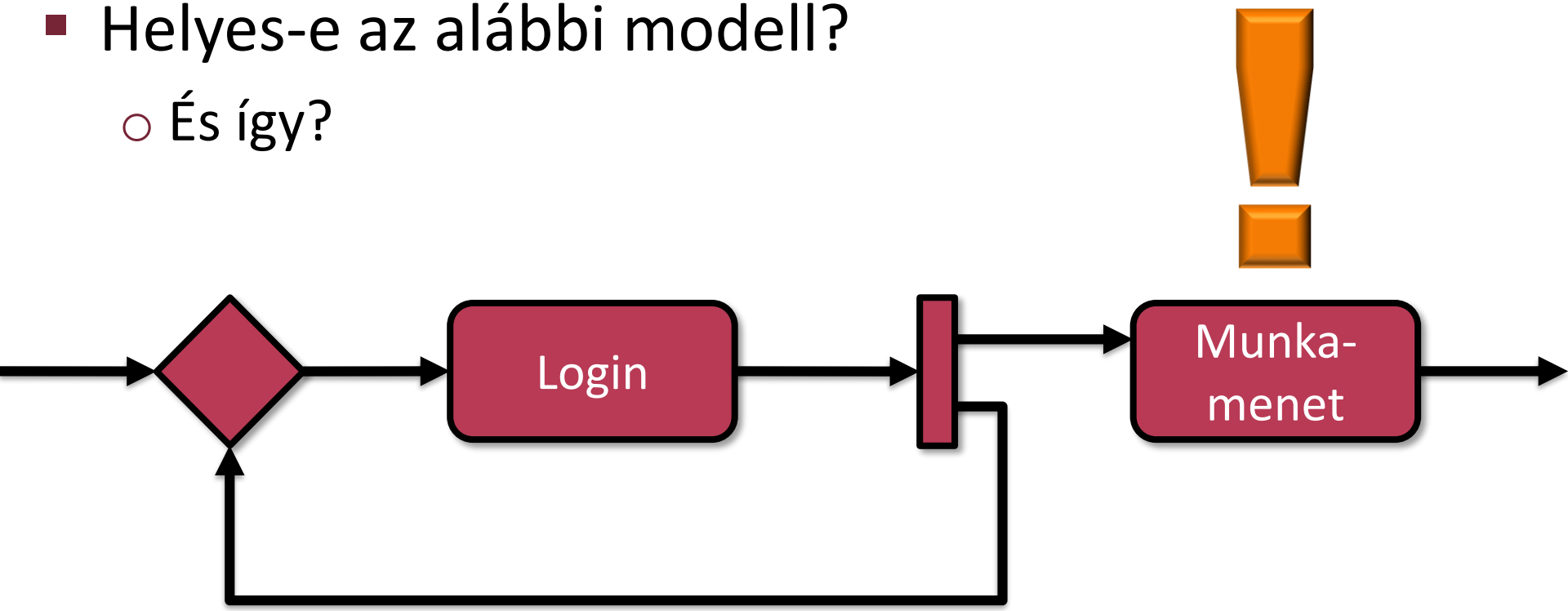


- Minden iterációban egy új képkocka
  - Mindegyikre utófeldolgozás (sokszor – hányszor?)

Határeset…

Ciklus 3.

- Helyes-e az alábbi modell?
  - És így?

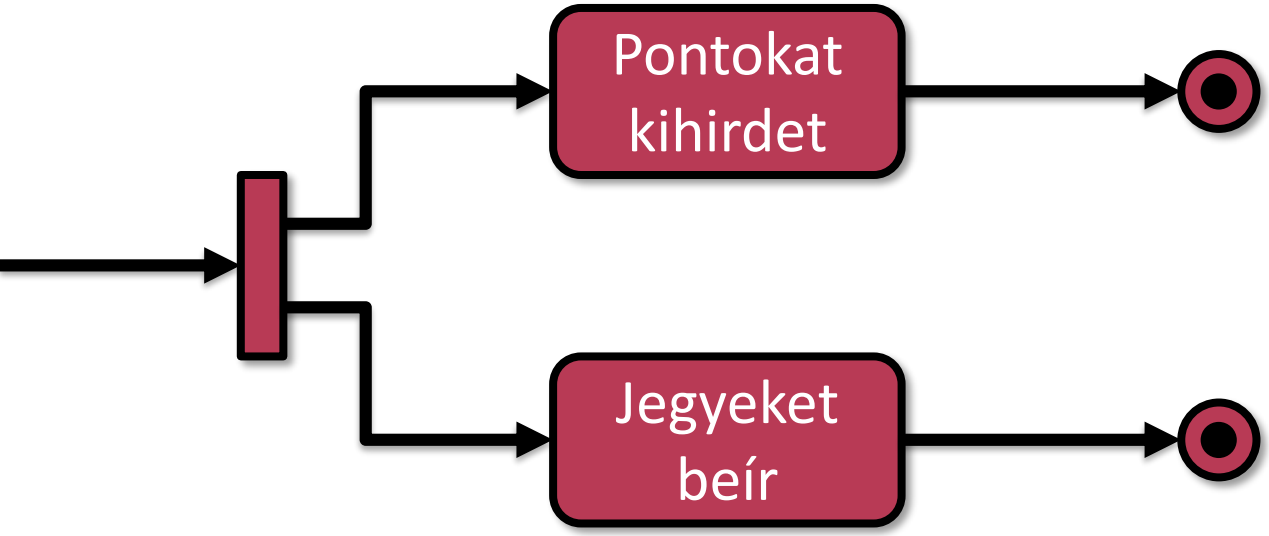


- Minden login után újabb login…
  - …és egy munkamenet…?

→ Szálakat „termel” a hibás implementáció

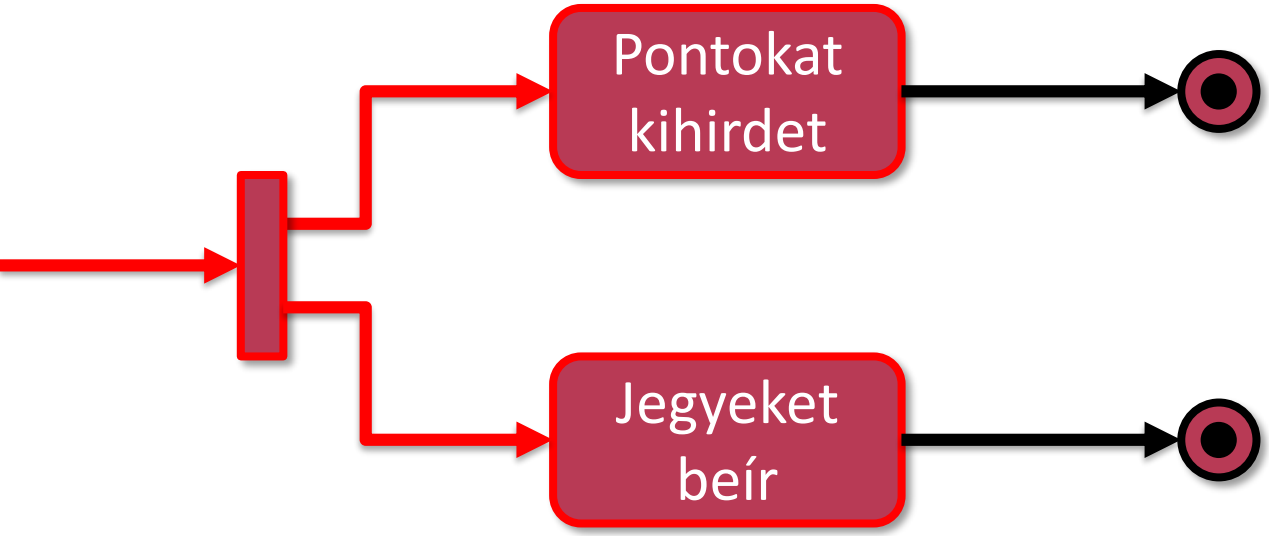
Terminál csomópont

- Helyes-e az alábbi modell?



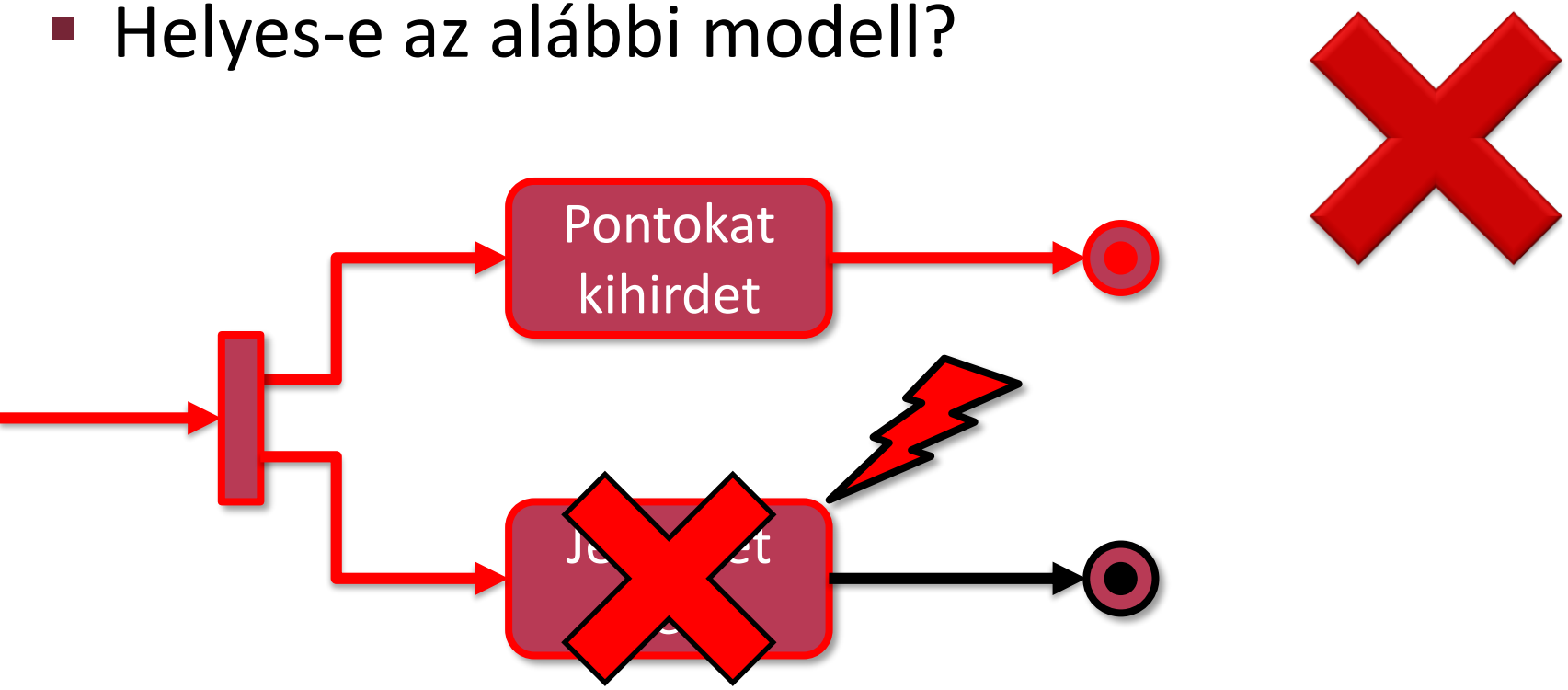
Terminál csomópont

- Helyes-e az alábbi modell?



Terminál csomópont

- Helyes-e az alábbi modell?

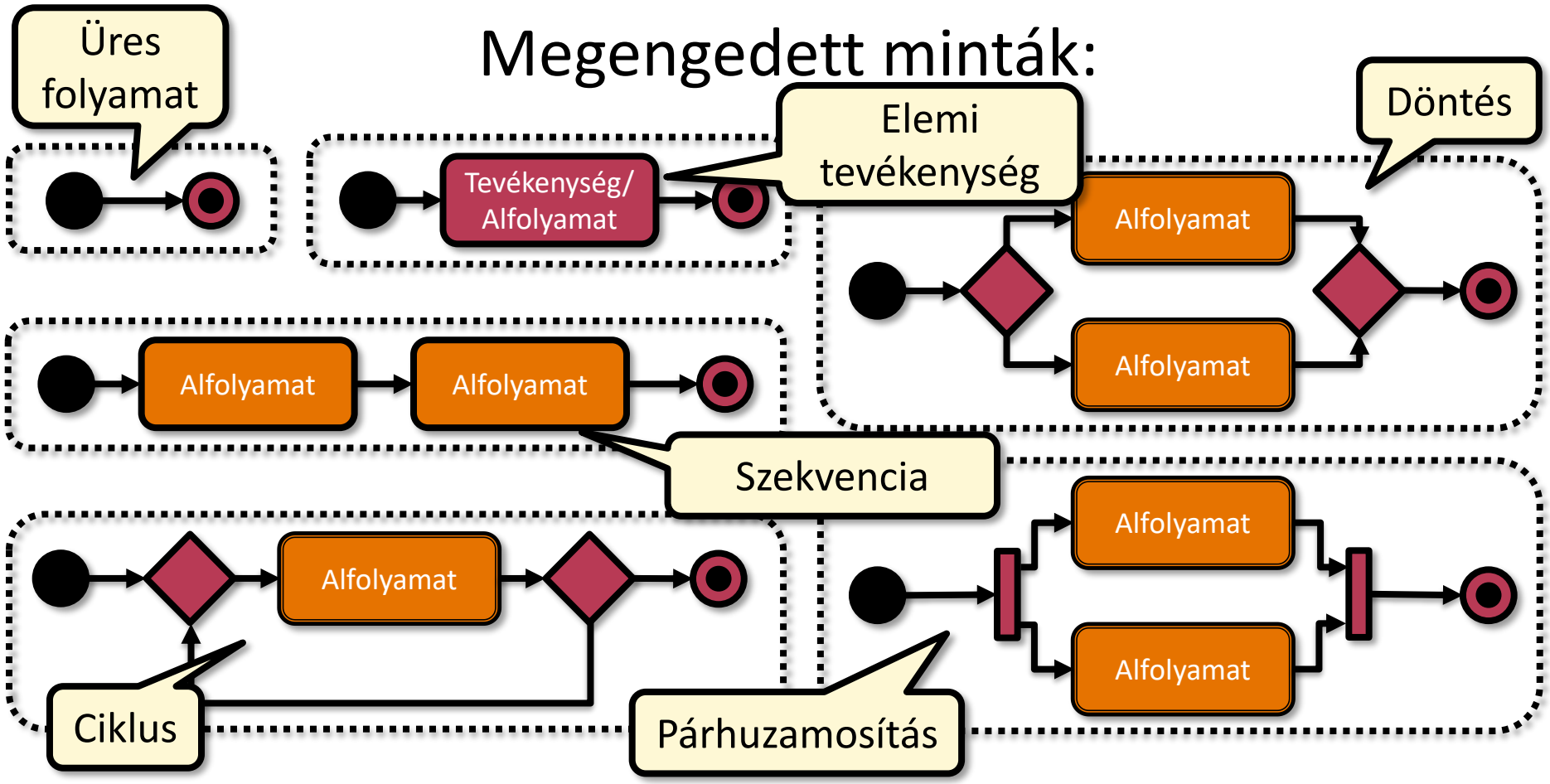


- Terminál csomópont: azonnal leállítja az **egész** folyamatot

→ **A másik tevékenység nem fog lefutni**

Jólstrukturált folyamatmodellek

- **Tanulság:** Jólstrukturált folyamatmodellekkel ezek a hibák elkerülhetők



Adatkezelés statikus ellenőrzése

- Egy eljárás két egész számot szoroz
  - Származtatott követelmény:
    - „Ha legalább az egyik páros, az eredmény is az”
  - Végigkövethető a kódon
    - „Fejben végrehajtás”

```
int mul(int a, int b) {
    return a * b;
}
```

- **Szimbolikus végrehajtás**
  - Konkrét értékek helyett értékhalmazokkal hajtjuk végre a programot
  - Érdekes bemenetek meghatározhatóak
    - Pl. ahol belső elágazások vannak

→Milyen bemenettel érhetőek el az ágak?

Statikus ellenőrzés: szintaxisellenőrzés

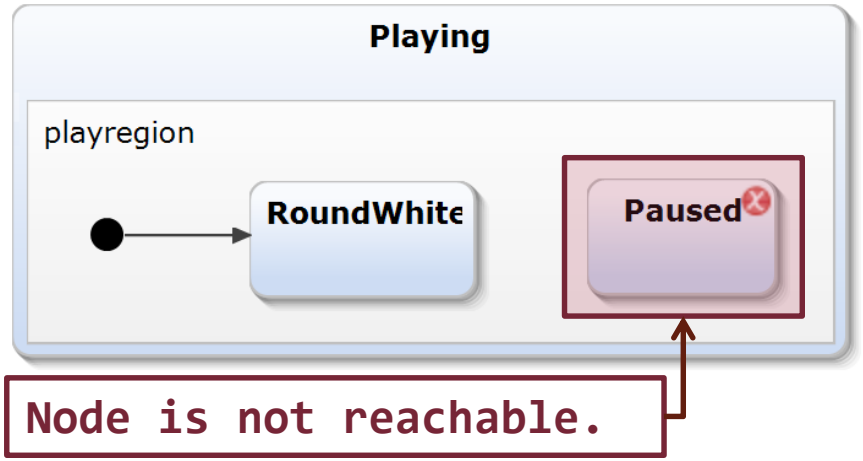
- Szintaktikai ellenőrzés: modellező eszközök összekötik a logikailag egymásra épülő modellelemeket

- Szintaxisvezérelt szerkesztő
  - Szerkesztés közben hiba→ `Couldn't resolve reference`
  - Fejlett szerkesztőeszköz (például lehetőségek felkínálása)
- Kód és diagram együtt

- Programozás: szerkesztés közben **hibás**
  Modellezés: szerkesztés közben **helyes**

Statikus ellenőrzés: strukturális helyesség

- Strukturális ellenőrzés: modell gráf vizsgálata
- Hibaminták keresése szerkesztés közben
- Például elérhetetlen állapot:



- További ellenőrzések: hiányzó kezdőállapot, holtpont, változó értékadások, stb.

Statikus ellenőrzés: tervezési szabályok

- **Példa 3**. Tervezési irányelvek támogatása:
  További szabályok adhatóak a modellhez.
  - *Always* és *Oncycle*: Órajelre tüzelő esemény
  - Tetszőleges frekvencia → Tipikusan hibás működés

A házi feladatban tilos az *Always* és *Oncycle* események használata.

Alapfogalmak

Statikus ellenőrzés

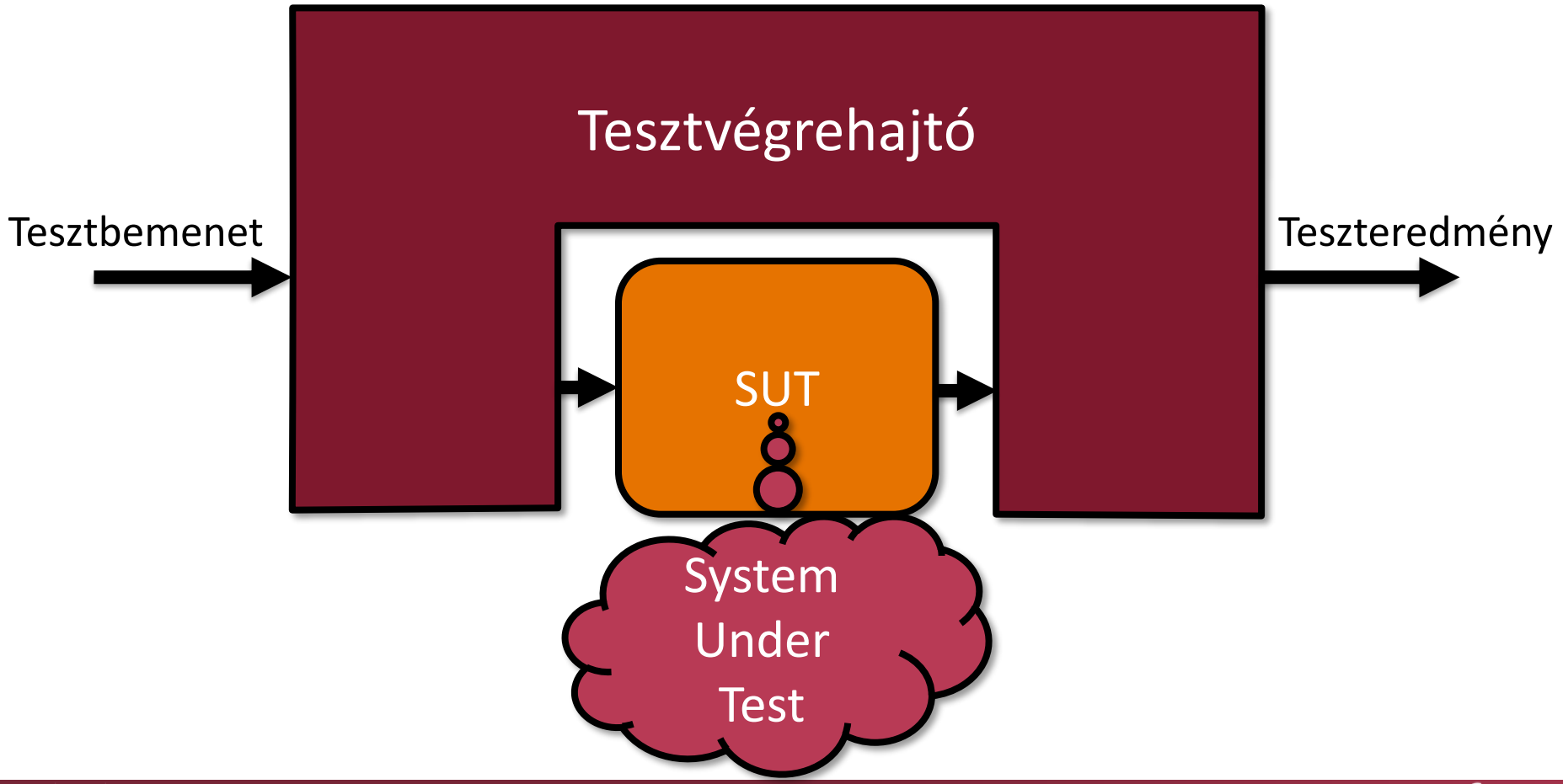
Tesztelés

Formális verifikáció

TESZTELÉS

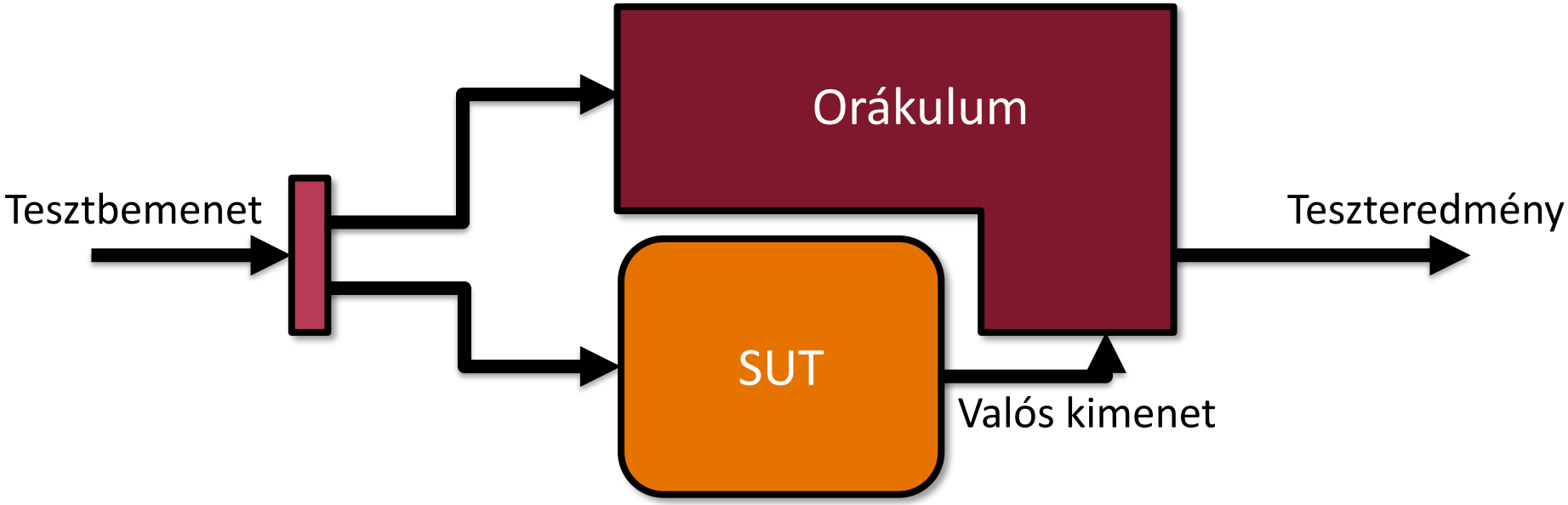
Csupán „próbálgatás”?

Modellek tesztelése



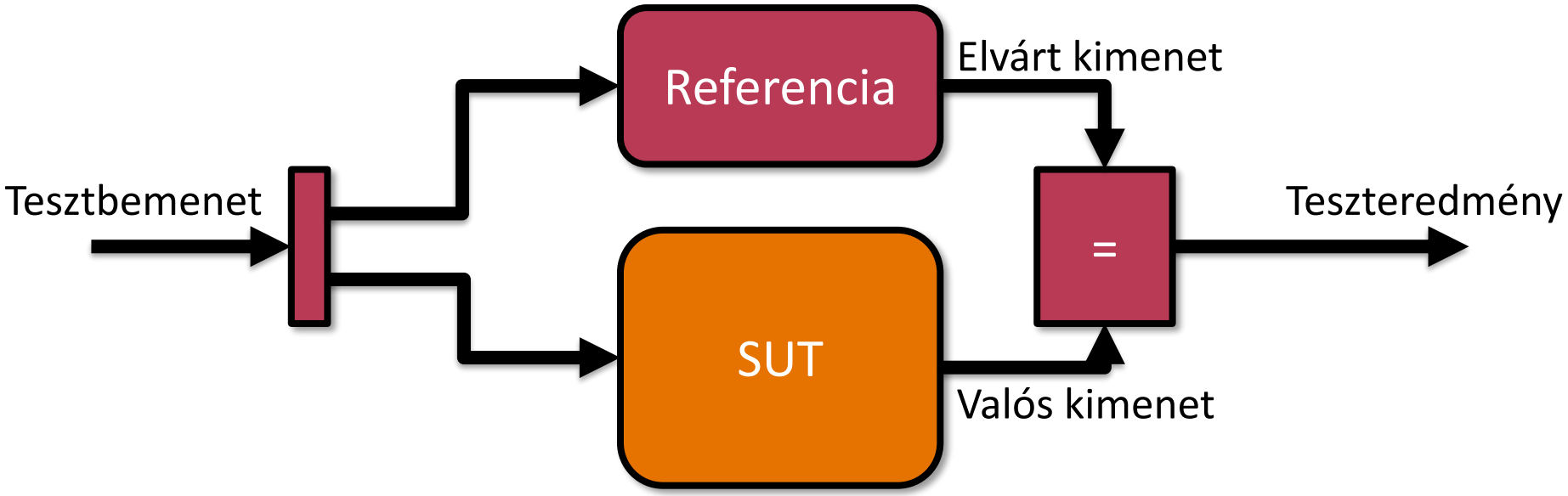
Modellek tesztelése

- **Orákulum:**
  elvárt eredmények előállítása és összehasonlítása

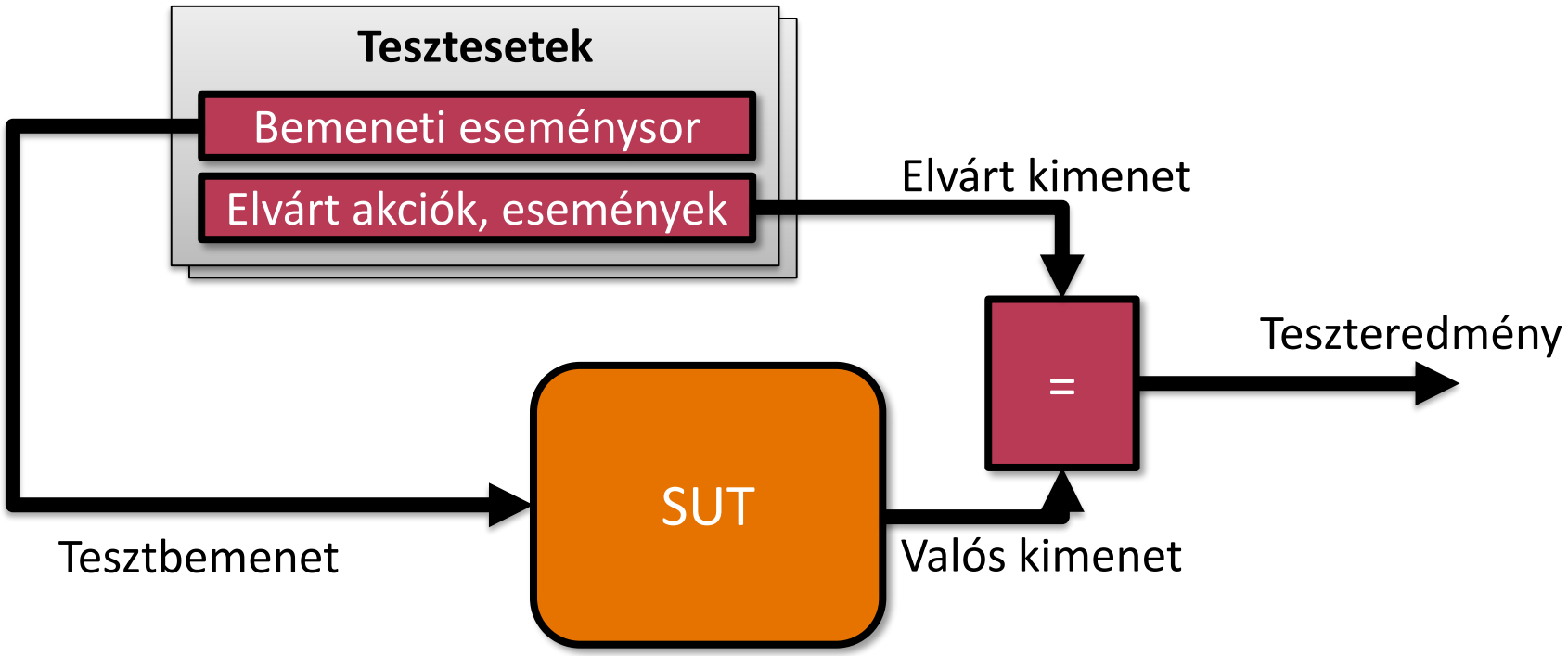


Modellek tesztelése

- **Referencia:**
  tesztbemenet alapján elvárt kimenet



Modellek tesztelése példa: Yakindu állapotgép



Modellek tesztelése példa: Yakindu állapotgép

Példa teszteset: Beállítások menüben 1 és 3 perc között lehet 5 másodperc léptekkel állítani a kezdőidőt.

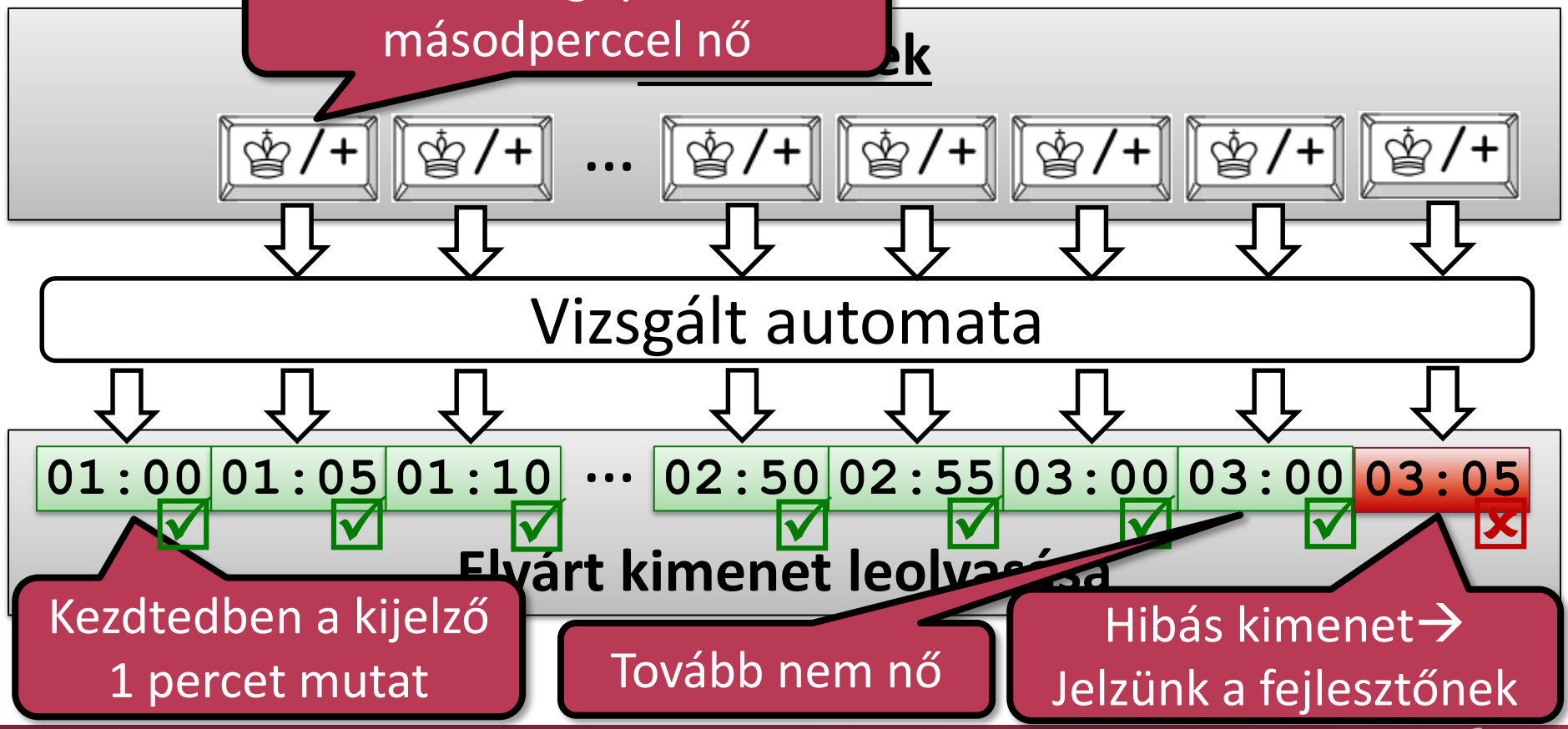
Bemenetek

Vizsgált automata

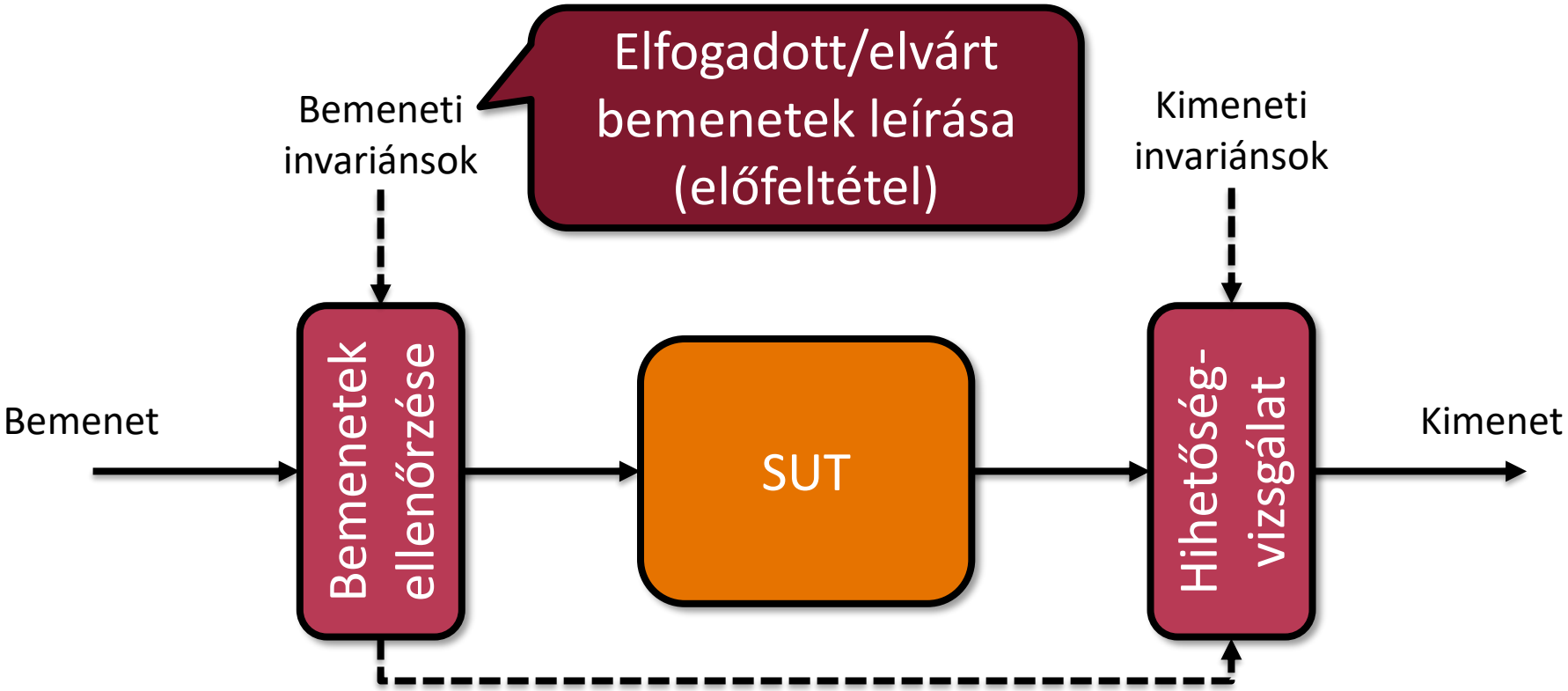
Elvárt kimenet leolvasása

Modellek tesztelése példa: Yakindu állapotgép

Példa teszteset: Beállítások menüben 1 és 3 perc között lehet 5 másodperc léptekkel állítani a kezdőid

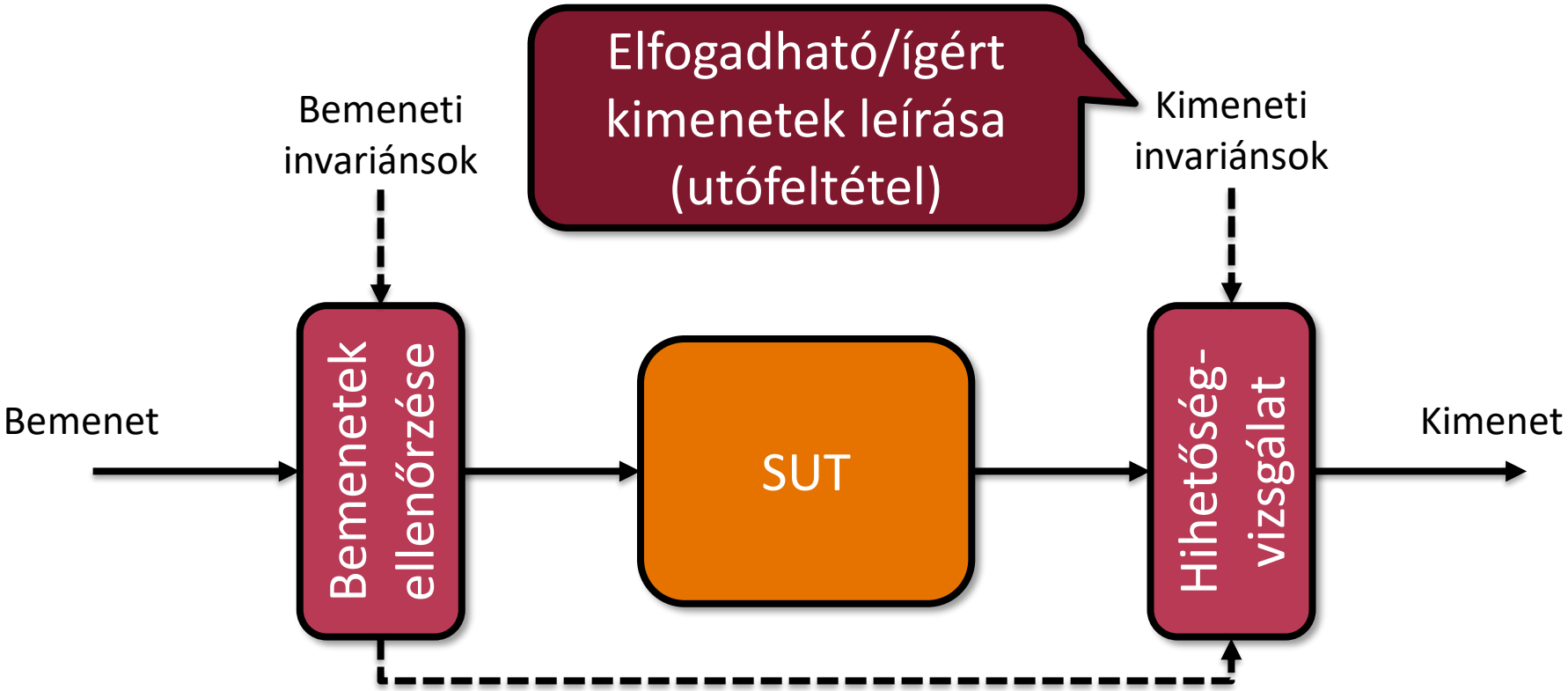


Öntesztelés (monitor)



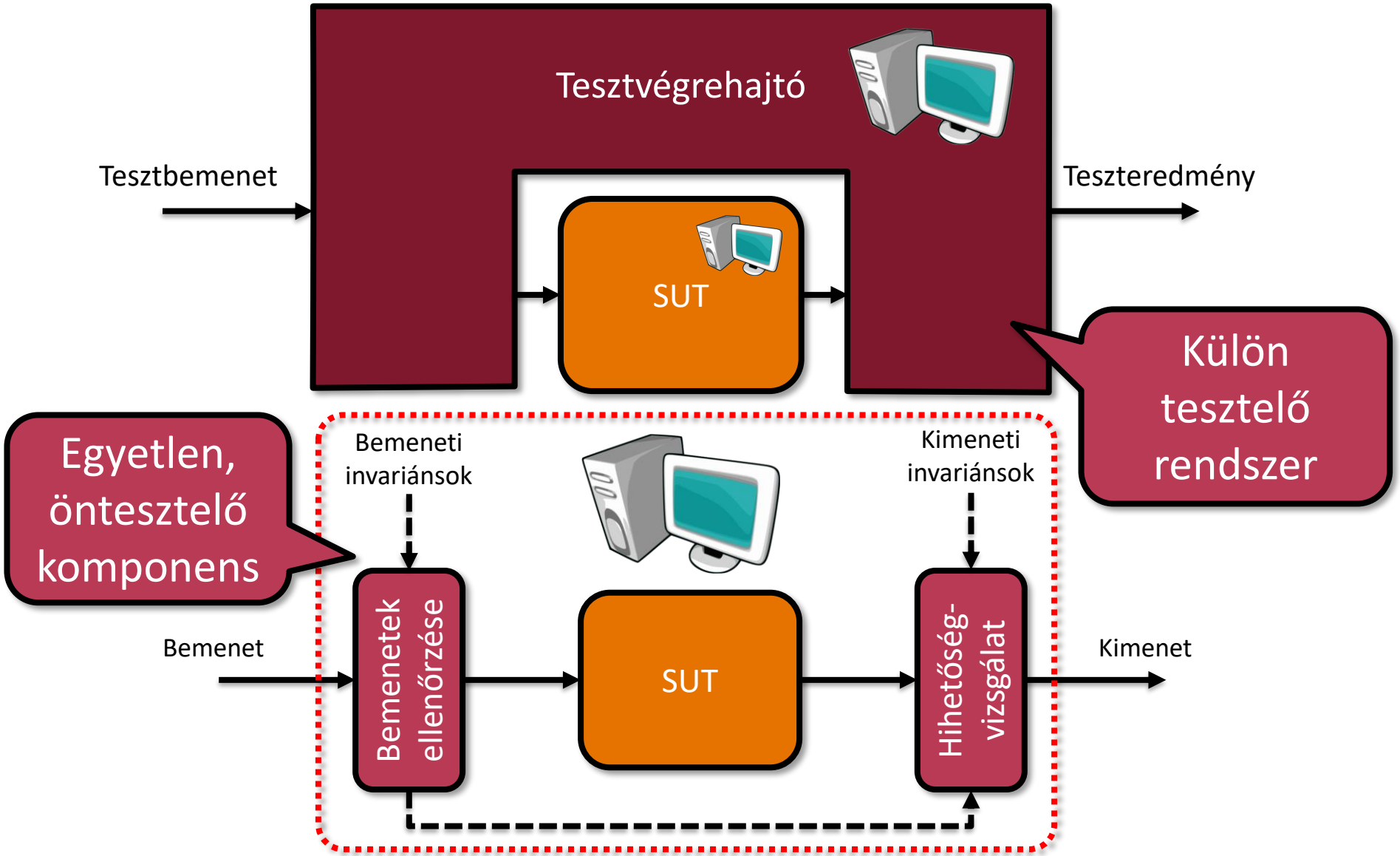
- **Invariáns tulajdonság:**
  folyamatosan igaznak kell lennie

Öntesztelés (monitor)



- **Invariáns tulajdonság:**
  folyamatosan igaznak kell lennie

Öntesztelés vs. Külső tesztelés



Öntesztelő program

Előfeltétel: a diszkrimináns nemnegatív

```
void Roots(float a, b, c,
           float &x1, &x2)
{
    float d = sqrt(b*b-4*a*c);

    x1 = (-b + d)/(2*a);
    x2 = (-b - d)/(2*a);
}
```

Utófeltétel: mindkét megoldás zérushely

```
void RootsMonitor(float a, b, c,
                  float &x1, &x2)
{
    // előfeltétel
    float D = b²-4·a·c;
    if (D < 0)
        throw "Invalid input!";

    // végrehajtás
    Roots(a, b, c, x1, x2);

    // utófeltétel
    assert(a·x1²+b·x1+c == 0 &&
           a·x2²+b·x2+c == 0);
}
```

Öntesztelő program

Exception (kivétel):
A normálistól eltérő, váratlan helyzet, aminek **kezelését máshol valósítjuk meg**.
Oka: hibás használat.

Assert (feltételezés):
Hibás állapot, aminek kezelésére a kód **nincs felkészítve**.
Oka: hibás implementáció vagy futásidejű hiba.

```
void RootsMonitor(float a, b, c,
                  float &x1, &x2)
{
    // előfeltétel
    float D = b²-4·a·c;
    if (D < 0)
        throw "Invalid input!";

    // végrehajtás
    Roots(a, b, c, x1, x2);

    // utófeltétel
    assert(a·x1²+b·x1+c == 0 &&
           a·x2²+b·x2+c == 0);
}
```

Modellek tesztelése

- A modell futtatása: szimuláció
  - Adott bemenetekre vizsgált viselkedés
- Teszteset:
  1. Tesztbemenet
     - pl. értéktartomány közepe és két széle
  2. Elvárt kimenet

Milyen bemenetekkel teszteljünk?

Fedettség

- Adott tesztkészlet esetén a **fedettség** a tesztek futtatása alatt érintett részek aránya a modellben.
  - Állapot lefedettség (állapotgépben):

$$\frac{\textbf{érintett állapotok}}{\textbf{összes állapot}}$$

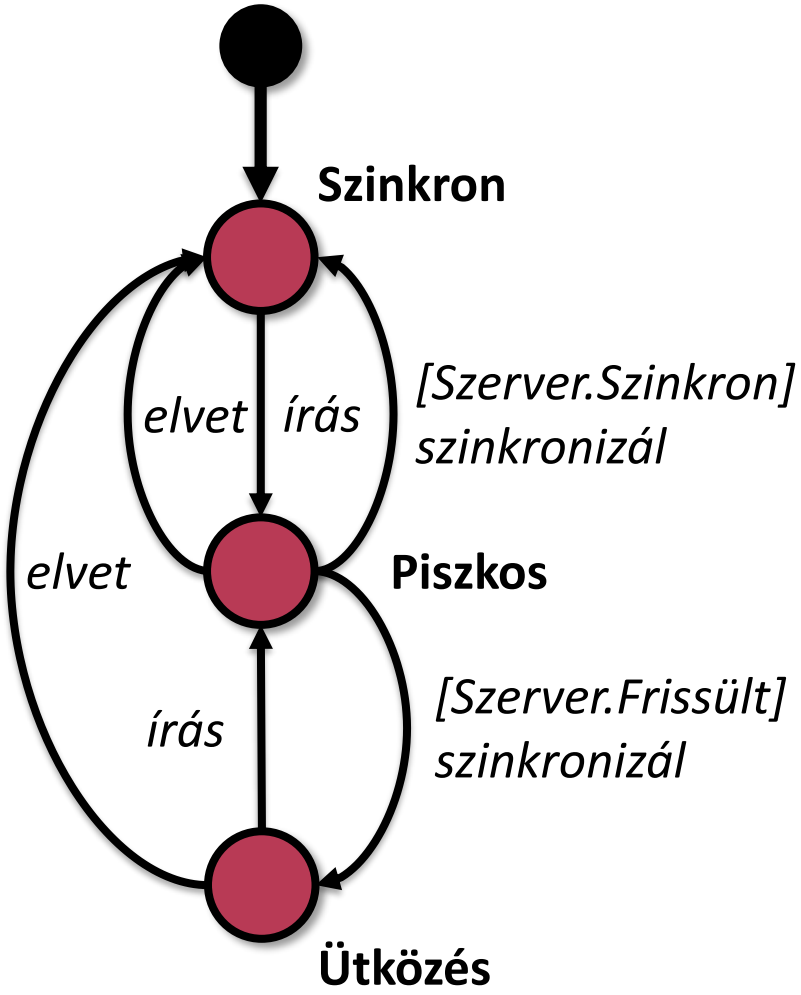
  - Átmenet lefedettség (állapotgépben):

$$\frac{\textbf{érintett átmenetek}}{\textbf{összes átmenet}}$$

  - Utasítás lefedettség (vezérlési folyamban):

$$\frac{\textbf{érintett tevékenységek}}{\textbf{összes tevékenység}}$$

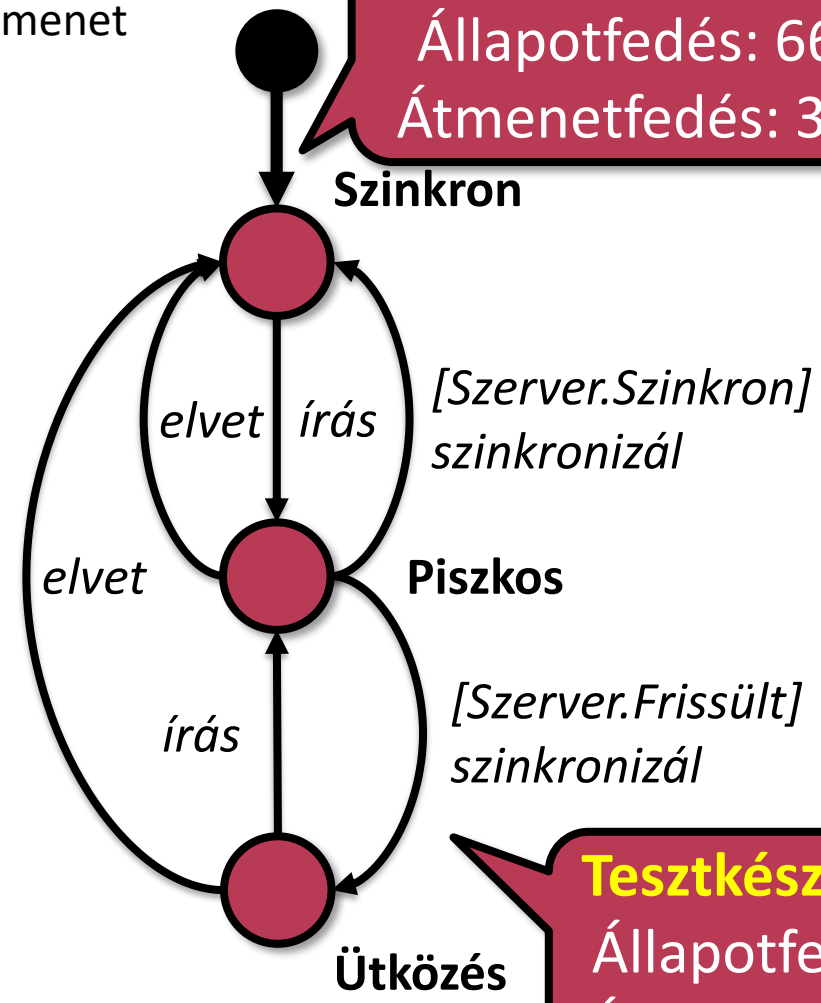
Példa: Felhőalapú adattárolás



„Felhő alapú adattárolást modellezünk, egyetlen állományra szorítkozva. A kliens írhatja az állományt, szinkronizálhat a szerverrel és elvetheti a helyi módosításokat. A szerveren tárolt változat verziójától függően a szinkronizálás ütközést is okozhat, ha más is módosította az állományt.”

Példa: Felhőalapú adattárolás

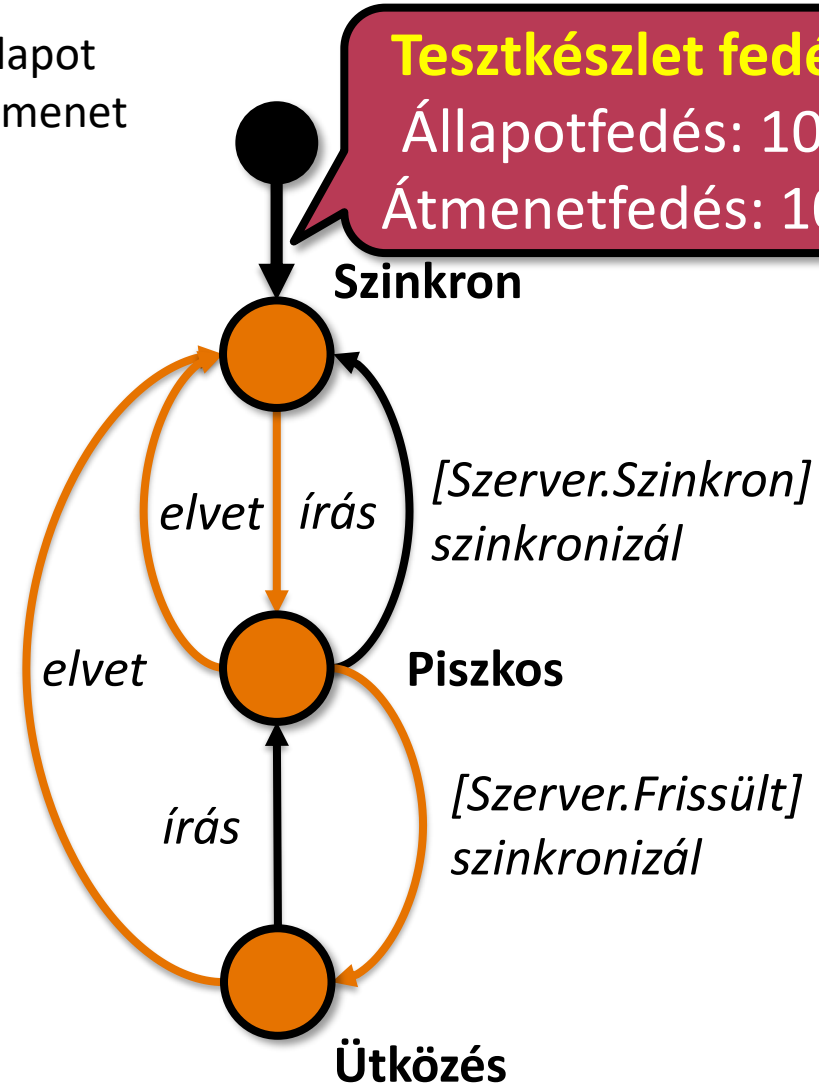
3 állapot
6 átmenet



1. Teszteset:
   a) írás
   b) elvet
2. Teszteset:
   a) írás
   b) Szerver = Frissült
   c) szinkronizál
   d) elvet

Példa: Felhőalapú adattárolás

3 állapot
6 átmenet



3. Teszteset:

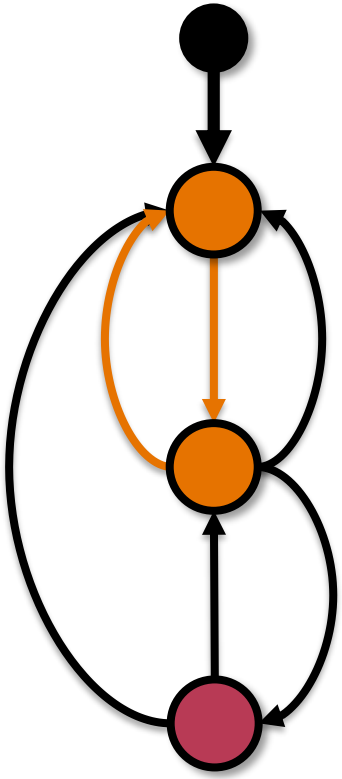
a) írás
b) Szerver = Frissült
c) szinkronizál
d) írás
e) Szerver = Szinkron
f) szinkronizál

Fedettség

1. Teszteset után:

Állapotfedés: 2/3=66%

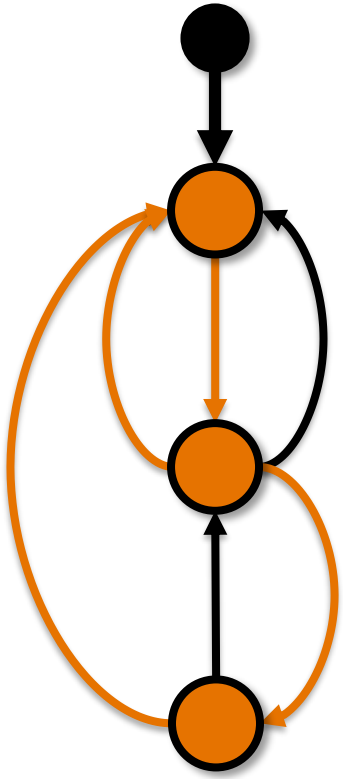
Átmenetfedés: 2/6=33%



2. Teszteset után:

Állapotfedés: 3/3=100%

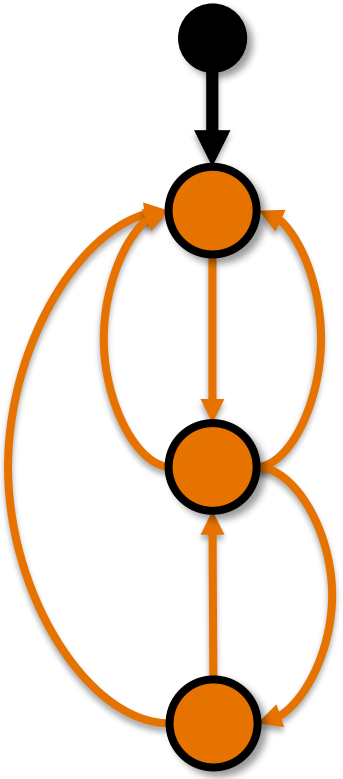
Átmenetfedés: 4/6=66%



3. Teszteset után:

Állapotfedés: 3/3=100%

Átmenetfedés: 6/6=100%



Tesztelt modellek használata

- **Szoftvertesztelés:**
  - (100%-osan fedő) tesztkészlet újrafelhasználása
  - Fedő tesztbemenetek (bemenet)
  - Modell által adott kimenetek (elvárt kimenet)
- **Monitorozás:** a modell szimulálása a szoftver futtatása mellett
  - Azonos bemenetek a modellnek és a programnak
  - Kimenetek összevetése → **hibadetektálás**
- **Logelemzés:**
  - Monitor futtatása naplózott bemenetek/kimeneten

Tesztelt modellek használata

- **Szoftvertesztelés:**
  - (100%-osan fedő) tesztkészlet újrafel
  - Fedő tesztbemenetek (bemenet)
  - Modell által adott kimenetek (elvárt kimenet)
- **Monitorozás:** a modell szimulálása a szoftver futtatása mellett
  - Azonos bemenetek a modellnek és a
  - Kimenetek összevetése → **hibadetektálás**
- **Logelemzés:**
  - Monitor futtatása naplózott bemene

Futtatás **előtt**

Futtatás **közben**

Futtatás **után**

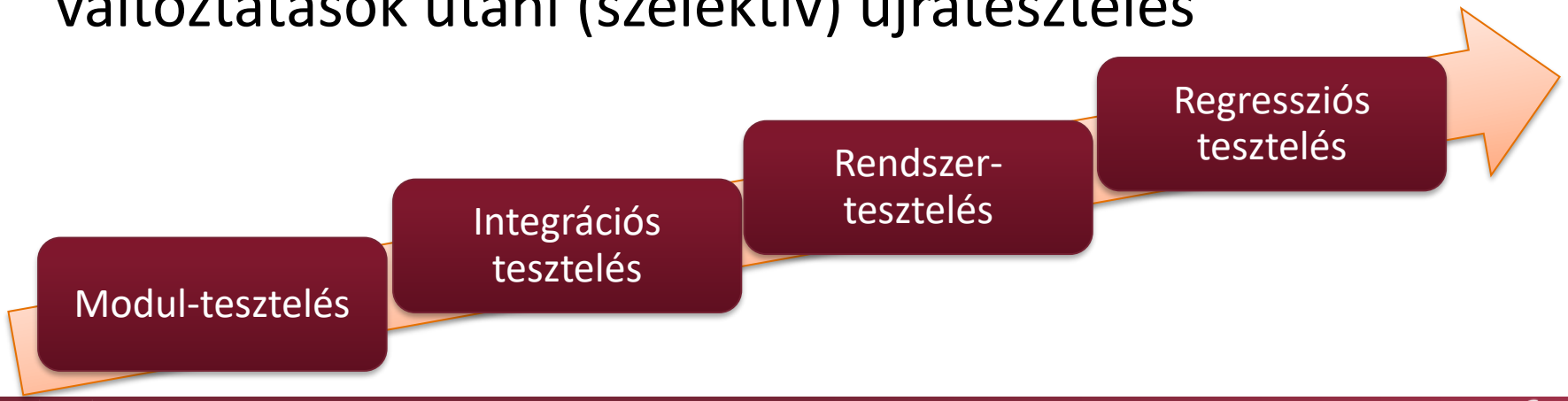
Tesztdokumentáció

- A teszteseteket és teszteredményeket is dokumentálni kell!
  - Mit vizsgál?
  - Milyen követelmény alapján?
  - Milyen bemenettel?
  - Futott-e már?
  - Ha igen, sikeres volt?

- Nyomonkövethetőség:
  - Teszteletlen kódrészletek és ellenőrizetlen követelmények felderítése
  - Teszteredmények nyilvántartása

Tesztek fajtái, szakaszai

- **Modulltesztelés:**
  egy komponens leválasztása és tesztelése
- **Integrációs tesztelés:**
  több komponens együttes tesztelése
- **Rendszertesztelés:**
  a teljes rendszer együttes tesztelése
- **Regressziós tesztelés:**
  változtatások utáni (szelektív) újratesztelés



Alapfogalmak

Statikus ellenőrzés

Tesztelés

Formális verifikáció

FORMÁLIS VERIFIKÁCIÓ

Modellellenőrzés

Formális verifikáció

- **Formális verifikáció:** modellek/programok helyességének bizonyítása matematikai eszközök segítségével
  - Bővebben lásd: Formális Módszerek MSc tárgy
- Eszközök:
  - **Modellellenőrzés**
    - Lehetséges viselkedések kimerítő vizsgálata
  - Automatikus helyességbizonyítás
    - Axiómarendszerek alapján automatikus tételbizonyítás
  - Konformancia vizsgálatok
    - Megfelelőség ellenőrzése modellek között

Modellellenőrzés

- **Modellellenőrzés:** a modell lehetséges viselkedéseinek kimerítő (teljes) vizsgálata adott követelmények alapján
  - Hibás működések keresése

    → **ellenpélda**

Tesztelés	Modellellenőrzés
Szúrópróbaszerű	Kimerítő/teljes
Elvárt kimeneteket ellenőriz	Állapotok sorozatát ellenőrzi
Kisebb számítási igényű	Nagy számítási igényű
Nem bizonyítóerejű	Formálisan bizonyít

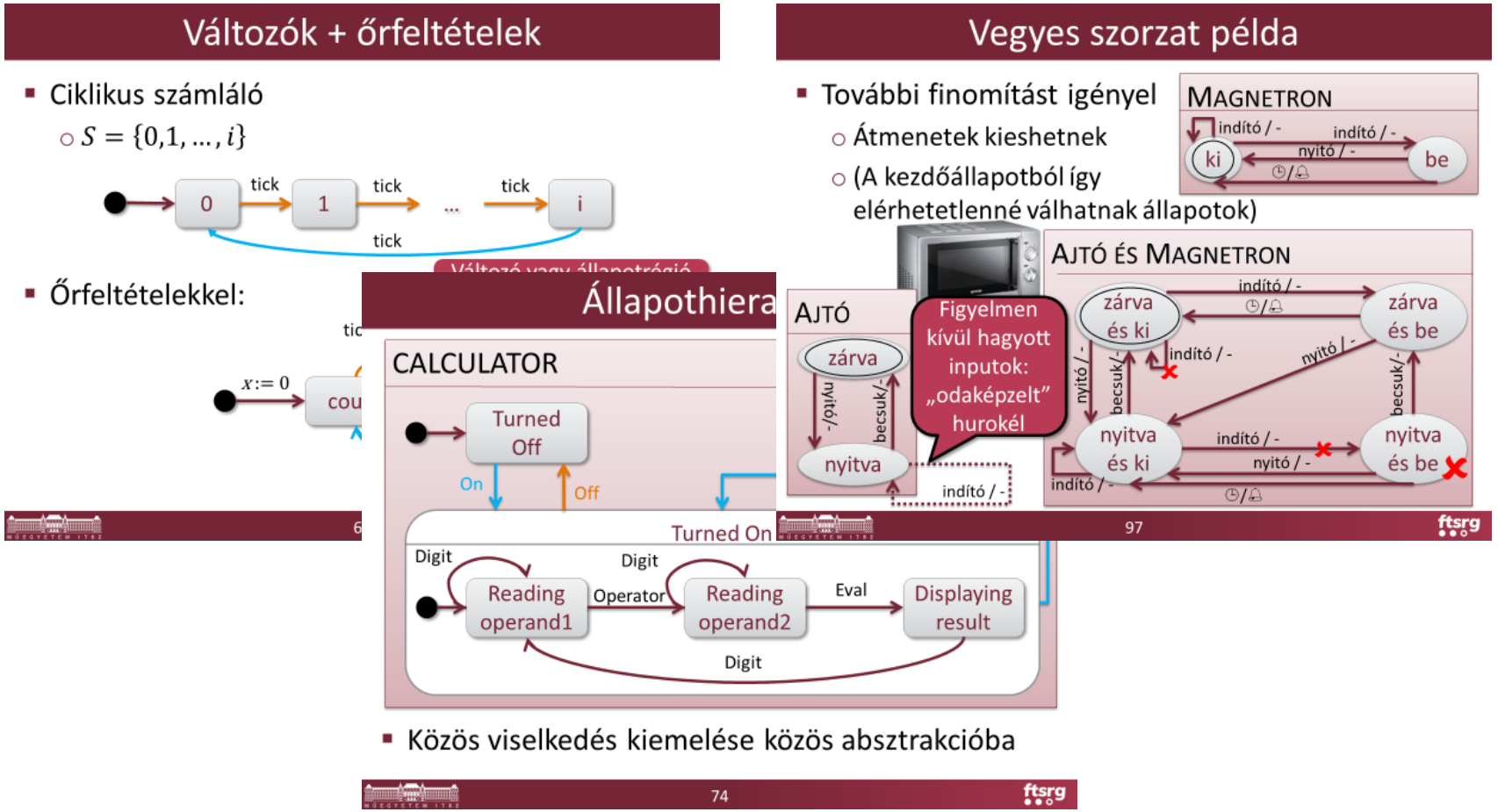
Modellellenőrzés

- **Modellellenőrzés:** a modell lehetséges viselkedéseinek kimerítő (teljes) vizsgálata adott követelmények alapján
  - Hibás működések keresése

    → **ellenpélda**
- **Állapottér-generálás:** a lehetséges viselkedések felderítése
  - Gyakran: állapotgépek vegyes szorzatának kiszámítása
- **Követelmények:** az állapotgráf elvárt tulajdonsága
  - Gyakran: temporális logikai kifejezések

Állapottér-generálás

- Egyszerű állapotgép előállítása az összetett formalizmusokból

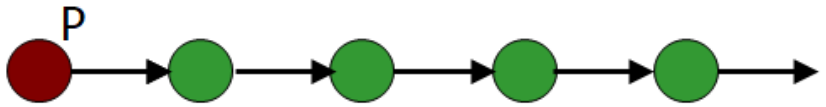


Követelmények: Temporális logika

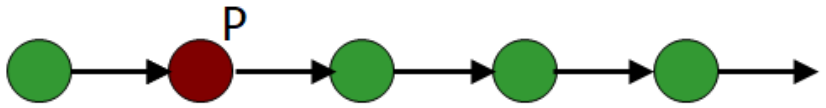
- Állapotok **halmaza** leírható állításlogikával
  - Pl: „$\Delta T < 0$” vagy „hőmérséklet > 5” (P, Q állítások)
- Állapotok **szekvenciája**: (lineáris) temporális logika

Mikor legyen igaz P (és Q)?

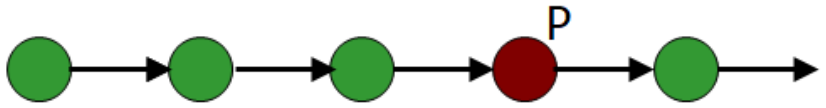
- Aktuális állapotban — P



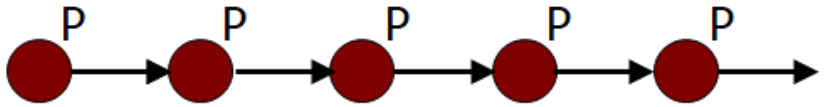
- Következő állapotban — X P



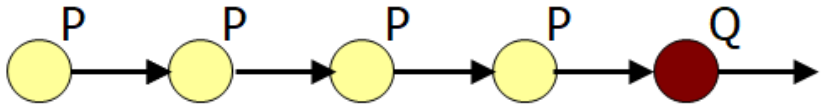
- Jövőbeli állapotban — F P



- Minden állapotban — G P



- P teljesül amíg Q nem — P U Q



Alapfogalmak

Statikus ellenőrzés

Tesztelés

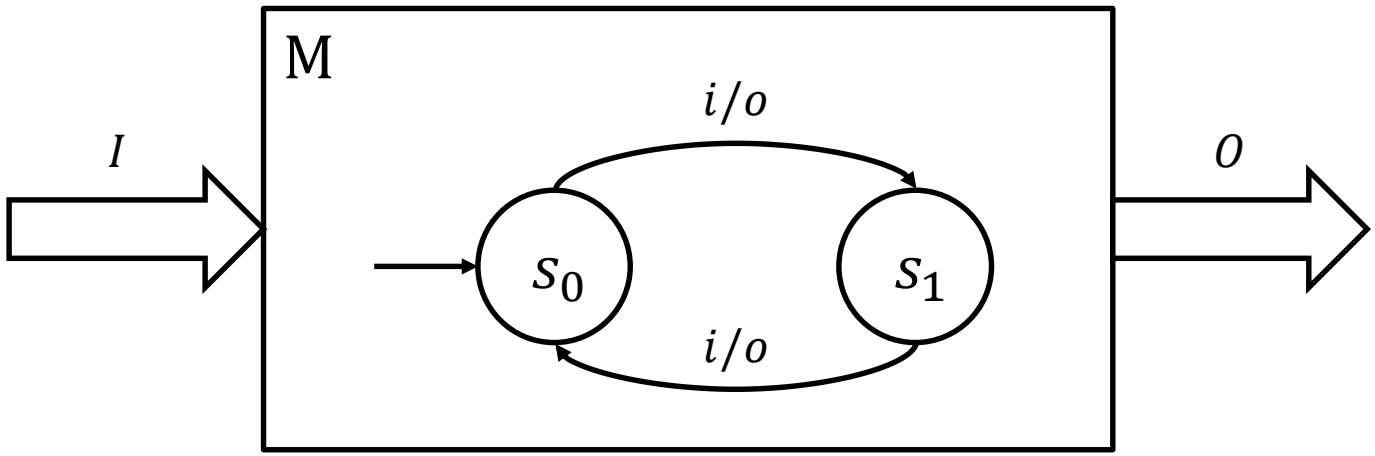
Formális verifikáció

FORMÁLIS SZEMANTIKA

Pontosan mit jelentenek az eddigi fogalmak?

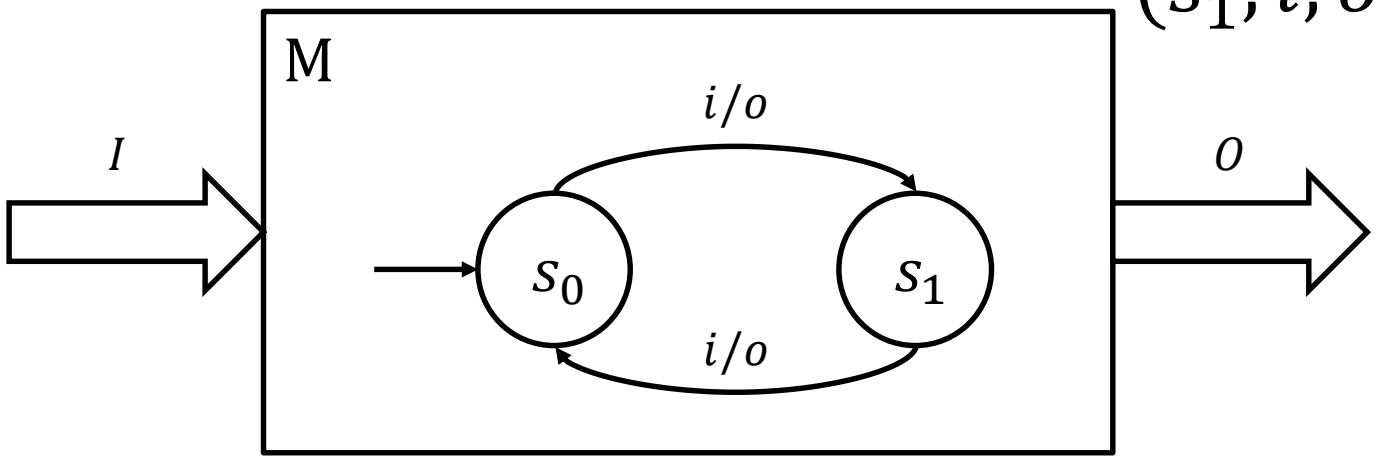
Egyszerű állapotgépek

- Bemeneti események: I
- Kimeneti események: O
- Állapotok halmaza: S
  - Kezdőállapot: $s_0 \in S$
- Állapotátmeneti szabályok: $R \subseteq S \times I \times O \times S$



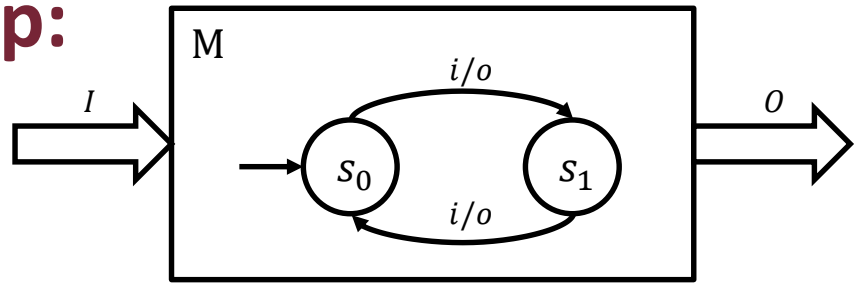
Egyszerű állapotgépek

- Bemeneti események: $I = \{i\}$
- Kimeneti események: $O = \{o\}$
- Állapotok halmaza: $S = \{s_0, s_1\}$
  - Kezdőállapot: $s_0 \in S$
- Állapotátmeneti szabályok: $R = \{(s_0, i, o, s_1), (s_1, i, o, s_0)\}$



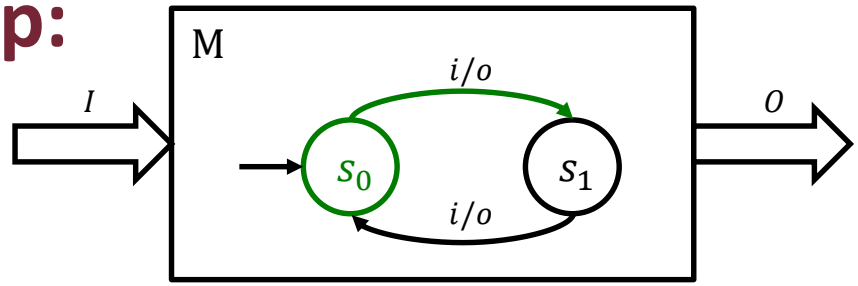
Egyszerű állapotgépek

- Bemeneti események: $I = \{i\}$
- Kimeneti események: $O = \{o\}$
- Állapotok halmaza: $S = \{s_0, s_1\}$
  - Kezdőállapot: $s_0 \in S$
- Állapotátmeneti szabályok: $R = \{(s_0, i, o, s_1), (s_1, i, o, s_0)\}$
- **Holtpontmentes állapotgép:**
  - Minden $s \in S$ állapotra
  - Létezik $(s, \iota, \omega, s') \in R$
  - …vagyis minden csomópontból indul ki él



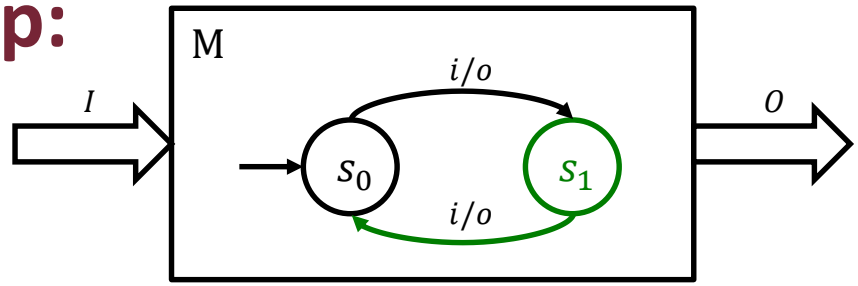
Egyszerű állapotgépek

- Bemeneti események: $I = \{i\}$
- Kimeneti események: $O = \{o\}$
- Állapotok halmaza: $S = \{s_0, s_1\}$
  - Kezdőállapot: $s_0 \in S$
- Állapotátmeneti szabályok: $R = \{(s_0, i, o, s_1), (s_1, i, o, s_0)\}$
- **Holtpontmentes állapotgép:**
  - Minden $s \in S$ állapotra
  - Létezik $(s, \iota, \omega, s') \in R$
  - …vagyis minden csomópontból indul ki él



Egyszerű állapotgépek

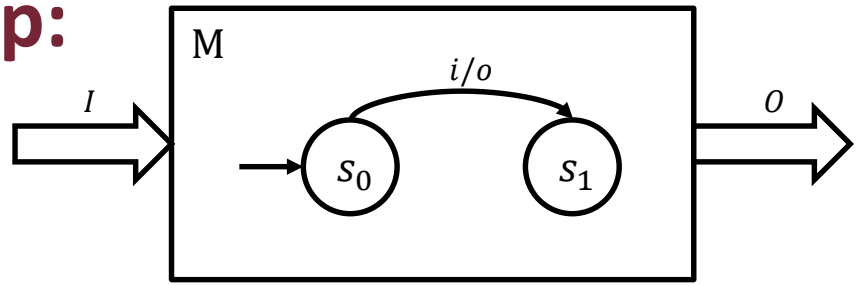
- Bemeneti események: $I = \{i\}$
- Kimeneti események: $O = \{o\}$
- Állapotok halmaza: $S = \{s_0, s_1\}$
  - Kezdőállapot: $s_0 \in S$
- Állapotátmeneti szabályok: $R = \{(s_0, i, o, s_1), (s_1, i, o, s_0)\}$
- **Holtpontmentes állapotgép:**
  - Minden $s \in S$ állapotra
  - Létezik $(s, \iota, \omega, s') \in R$
  - …vagyis minden csomópontból indul ki él



Egyszerű állapotgépek

- Bemeneti események: $I = \{i\}$
- Kimeneti események: $O = \{o\}$
- Állapotok halmaza: $S = \{s_0, s_1\}$
  - Kezdőállapot: $s_0 \in S$
- Állapotátmeneti szabályok: $R = \{(s_0, i, o, s_1)\}$

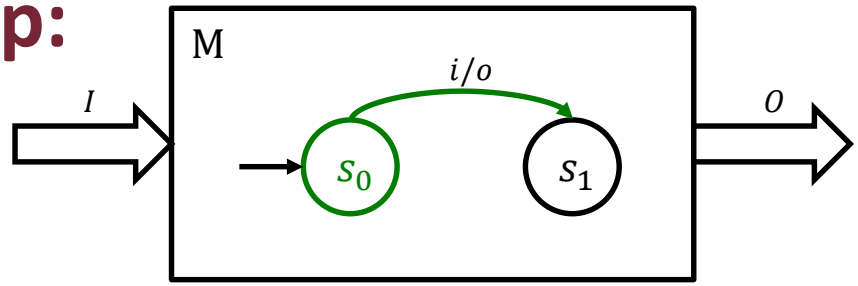
- **Holtpontmentes állapotgép:**
  - Minden $s \in S$ állapotra
  - Létezik $(s, \iota, \omega, s') \in R$
  - …vagyis minden csomópontból indul ki él



Egyszerű állapotgépek

- Bemeneti események: $I = \{i\}$
- Kimeneti események: $O = \{o\}$
- Állapotok halmaza: $S = \{s_0, s_1\}$
  - Kezdőállapot: $s_0 \in S$
- Állapotátmeneti szabályok: $R = \{(s_0, i, o, s_1)\}$

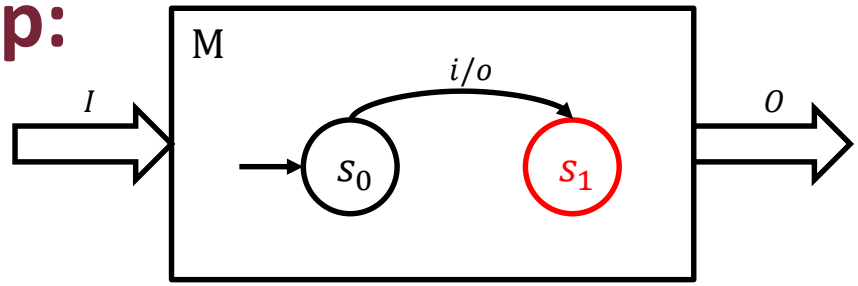
- **Holtpontmentes állapotgép:**
  - Minden $s \in S$ állapotra
  - Létezik $(s, \iota, \omega, s') \in R$
  - …vagyis minden csomópontból indul ki él



Egyszerű állapotgépek

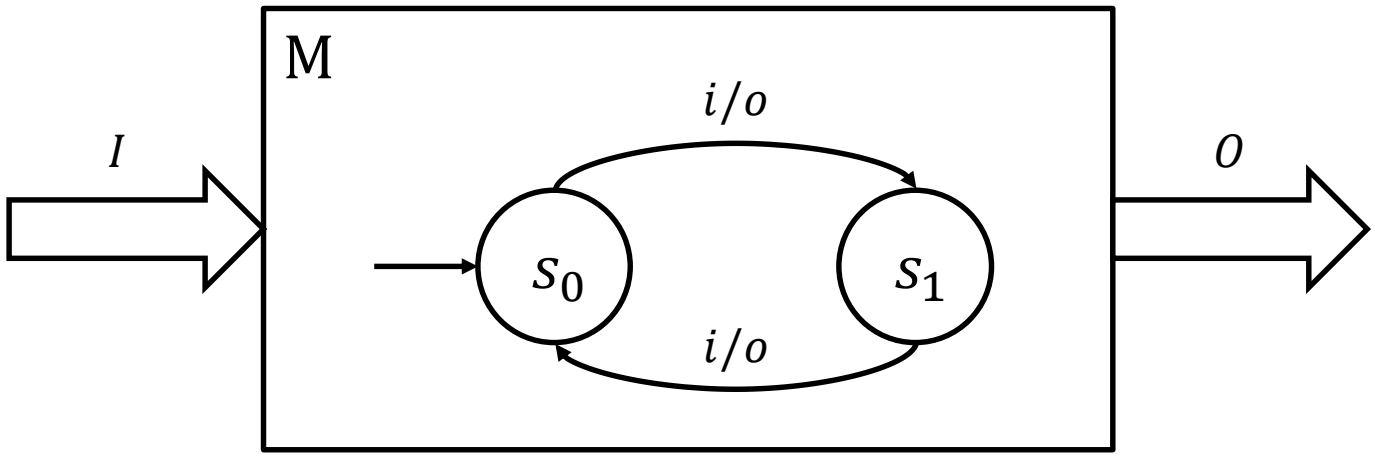
- Bemeneti események: $I = \{i\}$
- Kimeneti események: $O = \{o\}$
- Állapotok halmaza: $S = \{s_0, s_1\}$
  - Kezdőállapot: $s_0 \in S$
- Állapotátmeneti szabályok: $R = \{(s_0, i, o, s_1)\}$

- **Holtpontmentes állapotgép:**
  - Minden $s \in S$ állapotra
  - Létezik $(s, \iota, \omega, s') \in R$
  - …vagyis minden csomópontból indul ki él



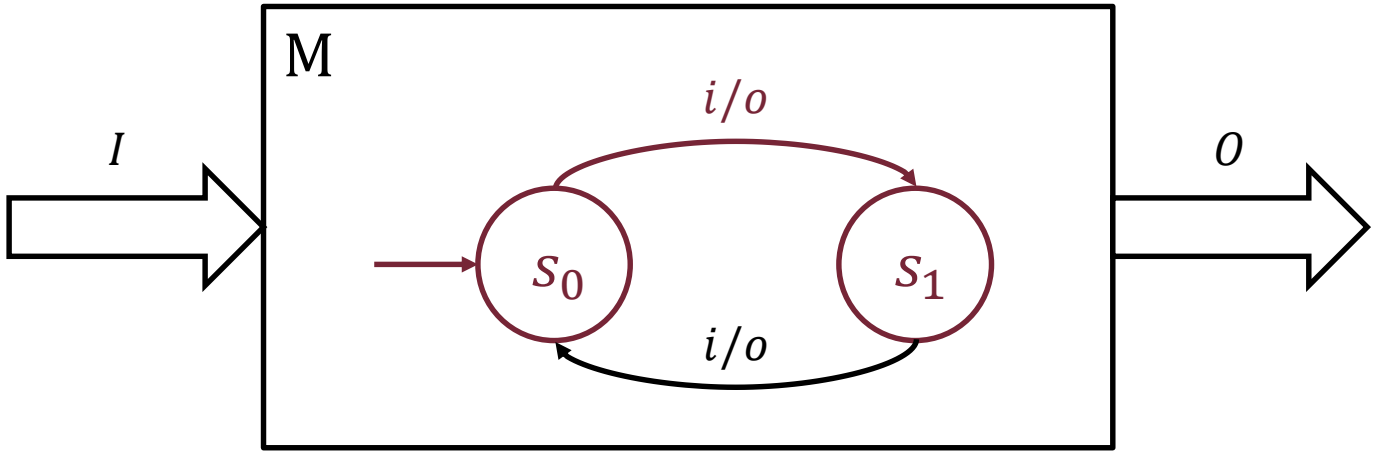
Egyszerű állapotgépek

- Egyszerű állapotgép: $\mathrm{M} = (I, O, S, s_0, R)$
- Végrehajtási út: $\pi = \langle s_0, i_0, o_0, s_1, i_1, o_1, s_2, \dots \rangle$
  - $(s_k, i_k, o_k, s_{k+1}) \in R$
  - $\Pi_M = \{\pi_1, \pi_2, \dots\}$ az M állapotgép végrehajtási útjai



Egyszerű állapotgépek

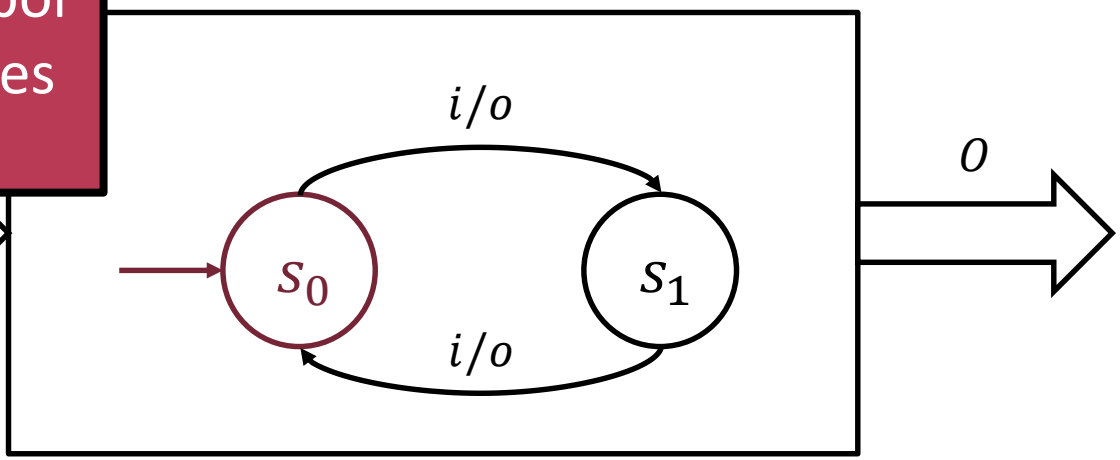
- Egyszerű állapotgép: $\mathrm{M} = (I, O, S, s_0, R)$
- Végrehajtási út: $\pi = \langle s_0, i_0, o_0, s_1 \rangle$
  - $(s_0, i_0, o_0, s_1) \in R$
  - Π_M ebben az esetben végtelen nagy



Egyszerű állapotgépek

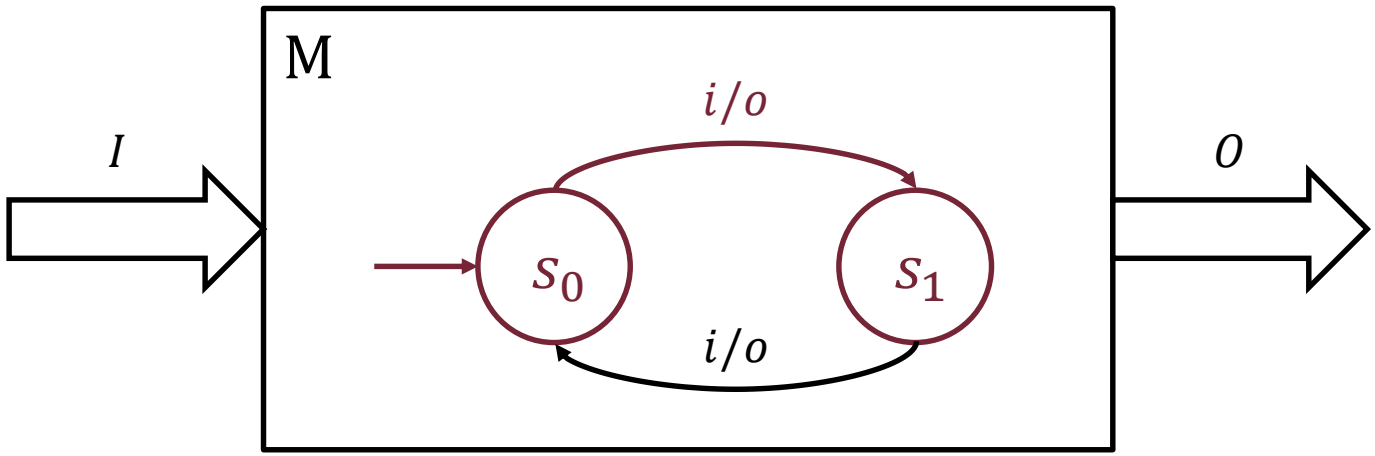
- Egyszerű állapotgép: $\mathrm{M} = (I, O, S, s_0, R)$
- Végrehajtási út: $\pi = \langle s_0, i_0, o_0, s_1, i_1, o_1, s_2, \dots \rangle$
- **Megfigyelhető viselkedés:**
  - *(bemenet szekvencia, kimenet szekvencia)* párok
  - $\sigma_M \subseteq I^* \times O^*$ $\quad (\langle\rangle, \langle\rangle) \in \sigma_M$

X^* az X elemeiből képezhető összes szekvencia



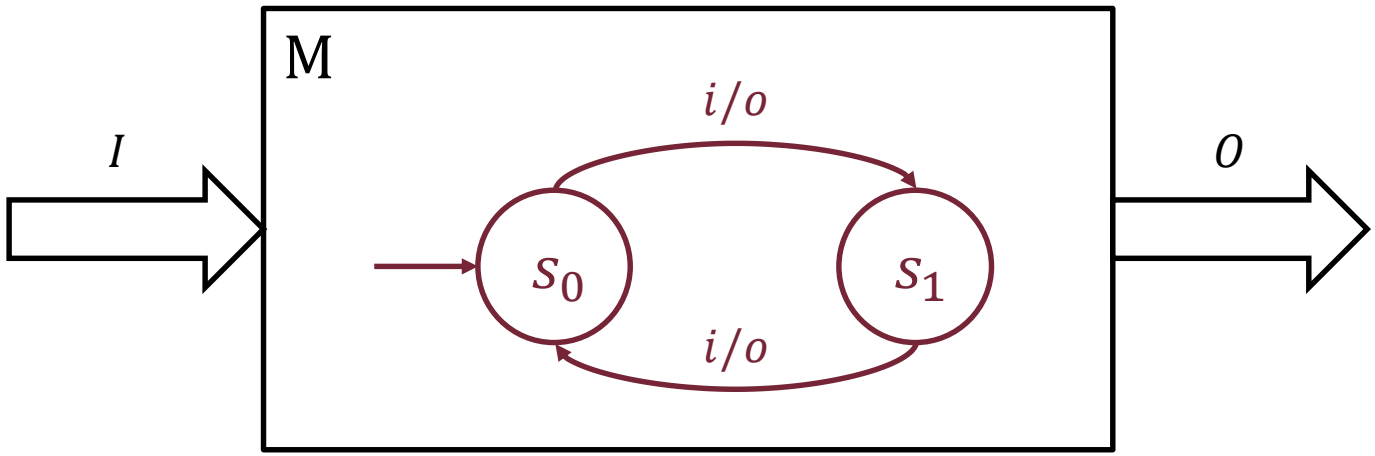
Egyszerű állapotgépek

- Egyszerű állapotgép: $\mathrm{M} = (I, O, S, s_0, R)$
- Végrehajtási út: $\pi = \langle s_0, i_0, o_0, s_1, i_1, o_1, s_2, \dots \rangle$
- **Megfigyelhető viselkedés:**
  - *(bemenet szekvencia, kimenet szekvencia)* párok
  - $\sigma_M \subseteq I^* \times O^*$ $\quad (\langle i \rangle, \langle o \rangle) \in \sigma_M$



Egyszerű állapotgépek

- Egyszerű állapotgép: $\mathrm{M} = (I, O, S, s_0, R)$
- Végrehajtási út: $\pi = \langle s_0, i_0, o_0, s_1, i_1, o_1, s_2, \dots \rangle$
- **Megfigyelhető viselkedés:**
  - *(bemenet szekvencia, kimenet szekvencia)* párok
  - $\sigma_M \subseteq I^* \times O^*$ $\quad (\langle i, i \rangle, \langle o, o \rangle) \in \sigma_M$



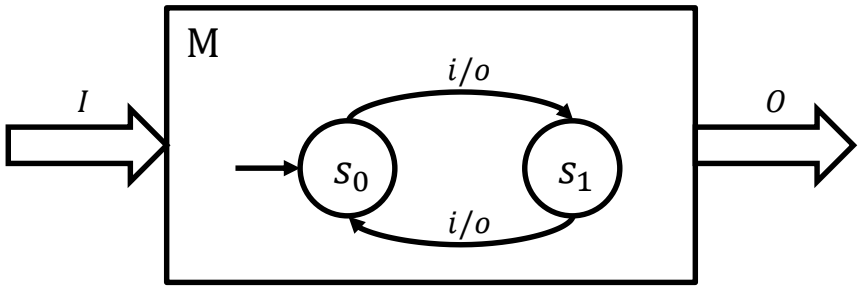
Egyszerű állapotgépek

- Egyszerű állapotgép: $\mathrm{M} = (I, O, S, s_0, R)$
- Végrehajtási út: $\pi = \langle s_0, i_0, o_0, s_1, i_1, o_1, s_2, \dots \rangle$
- **Megfigyelhető viselkedés:**
  - *(bemenet szekvencia, kimenet szekvencia)* párok
  - $\sigma_M \subseteq I^* \times O^* \qquad (\langle i, i \rangle, \langle o, o \rangle) \in \sigma_M$
  - $(\langle i_0, i_1, \dots \rangle, \langle o_0, o_1, \dots \rangle) \in \sigma_M$ akkor és csak akkor, ha:
    - Létezik olyan $\pi \in \Pi_M$ végrehajtási út, hogy
    - $\pi = \langle s_0, i_0, o_0, s_1, i_1, o_1, s_2, \dots \rangle$
  - σ_M mint függvény: $\sigma_M : I^* \to 2^{O^*}$

O^* hatványhalmaza:
Részhalmazok halmaza

Egyszerű állapotgépek

- Egyszerű állapotgép: $\mathrm{M} = (I, O, S, s_0, R)$
- Végrehajtási út: $\pi = \langle s_0, i_0, o_0, s_1, i_1, o_1, s_2, \dots \rangle$
- **Megfigyelhető viselkedés:**
  - *(bemenet szekvencia, kimenet szekvencia)* párok
  - $\sigma_M \subseteq I^* \times O^* \qquad (\langle i, i \rangle, \langle o, o \rangle) \in \sigma_M$

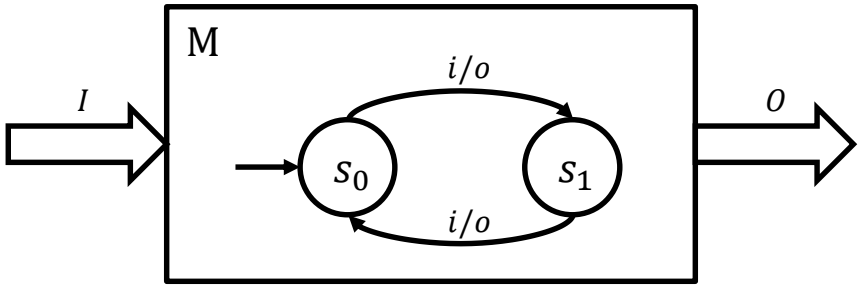


$$\sigma_M(\langle i_0, i_1, \dots \rangle) = \begin{cases} \{o^n\} \text{ ha } \langle i_0, i_1, \dots \rangle = i^n \\ \emptyset \text{ egyébként} \end{cases}$$

  - σ_M mint függvény: $\sigma_M : I^* \to 2^{O^*}$

Egyszerű állapotgépek

- Egyszerű állapotgép: $\mathrm{M} = (I, O, S, s_0, R)$
- Végrehajtási út: $\pi = \langle s_0, i_0, o_0, s_1, i_1, o_1, s_2, \dots \rangle$
- Megfigyelhető viselkedés: $\sigma_M : I^* \to 2^{O^*}$
- **Determinisztikus állapotgép:**
  - Minden $\langle i_0, i_1, \dots \rangle \in I^*$ bemeneti szekvenciára
  - $|\sigma_M(\langle i_0, i_1, \dots \rangle)| \leq 1$
    - Vagyis legfeljebb egyféle viselkedés figyelhető meg

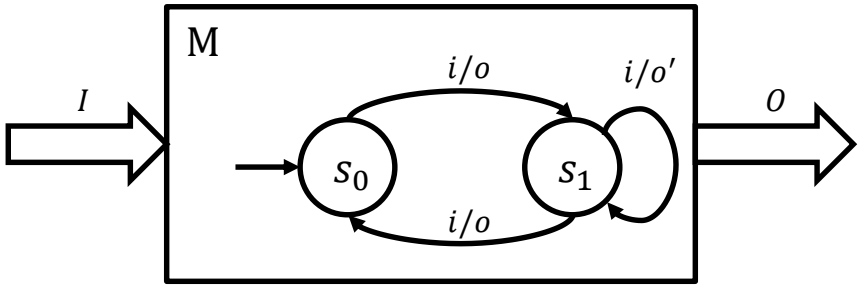


$$\sigma_M(\langle i_0, i_1, \dots \rangle) = \begin{cases} \{o^n\} \text{ ha } \langle i_0, i_1, \dots \rangle = i^n \\ \emptyset \text{ egyébként} \end{cases}$$

$$\Rightarrow |\sigma_M\ (\langle i_0, i_1, \dots \rangle)| \leq 1$$

Egyszerű állapotgépek

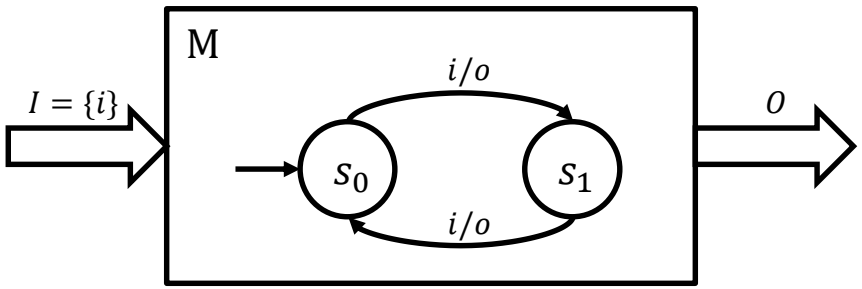
- Egyszerű állapotgép: $\mathrm{M} = (I, O, S, s_0, R)$
- Végrehajtási út: $\pi = \langle s_0, i_0, o_0, s_1, i_1, o_1, s_2, \dots \rangle$
- Megfigyelhető viselkedés: $\sigma_M : I^* \to 2^{O^*}$
- **Determinisztikus állapotgép:**
  - Minden $\langle i_0, i_1, \dots \rangle \in I^*$ bemeneti szekvenciára
  - $|\sigma_M(\langle i_0, i_1, \dots \rangle)| \leq 1$
    - Vagyis legfeljebb egyféle viselkedés figyelhető meg



$$|\sigma_M(\langle i, i \rangle)| = |\{\langle o, o \rangle, \langle o, o' \rangle\}| = 2 > 1$$

Egyszerű állapotgépek

- Egyszerű állapotgép: $\mathrm{M} = (I, O, S, s_0, R)$
- Végrehajtási út: $\pi = \langle s_0, i_0, o_0, s_1, i_1, o_1, s_2, \dots \rangle$
- Megfigyelhető viselkedés: $\sigma_M : I^* \to 2^{O^*}$
- **Teljesen specifikált állapotgép:**
  - Minden $\langle i_0, i_1, \dots \rangle \in I^*$ bemeneti szekvenciára
  - $|\sigma_M(\langle i_0, i_1, \dots \rangle)| \geq 1$
    - Vagyis legalább egyféle viselkedés megfigyelhető



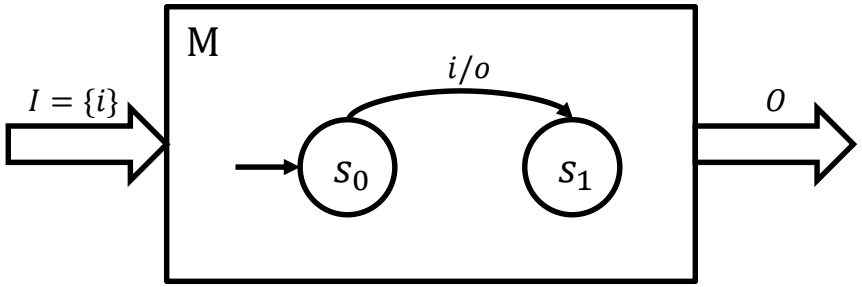
$$\sigma_M(\langle i_0, i_1, \dots \rangle) = \begin{cases} \{o^n\} \text{ ha } \langle i_0, i_1, \dots \rangle = i^n \\ \emptyset \text{ egyébként} \end{cases}$$

de $I^* = \{i^n \mid n \geq 0\}$

$\Rightarrow |\sigma_M(\langle i_0, i_1, \dots \rangle)| = 1 \geq 1$

Egyszerű állapotgépek

- Egyszerű állapotgép: $\mathrm{M} = (I, O, S, s_0, R)$
- Végrehajtási út: $\pi = \langle s_0, i_0, o_0, s_1, i_1, o_1, s_2, \dots \rangle$
- Megfigyelhető viselkedés: $\sigma_M : I^* \to 2^{O^*}$
- **Teljesen specifikált állapotgép:**
  - Minden $\langle i_0, i_1, \dots \rangle \in I^*$ bemeneti szekvenciára
  - $|\sigma_M(\langle i_0, i_1, \dots \rangle)| \geq 1$
    - Vagyis legalább egyféle viselkedés megfigyelhető



$|\sigma_M(\langle i, i \rangle)| = |\emptyset| = 0 < 1$

Egyszerű állapotgépek – Összefoglalás

- Egyszerű állapotgép: $\mathrm{M} = (I, O, S, s_0, R)$
- Végrehajtási út: $\pi = \langle s_0, i_0, o_0, s_1, i_1, o_1, s_2, \dots \rangle$
- Megfigyelhető viselkedés: $\sigma_M : I^* \to 2^{O^*}$
- Determinisztikus állapotgép:
  - Minden $\langle i_0, i_1, \dots \rangle \in I^*$ bemeneti szekvenciára
  - $|\sigma_M(\langle i_0, i_1, \dots \rangle)| \leq 1$
- Teljesen specifikált állapotgép:
  - Minden $\langle i_0, i_1, \dots \rangle \in I^*$ bemeneti szekvenciára
  - $|\sigma_M(\langle i_0, i_1, \dots \rangle)| \geq 1$
- Holtpontmentes állapotgép: *(létezik kimenő él)*

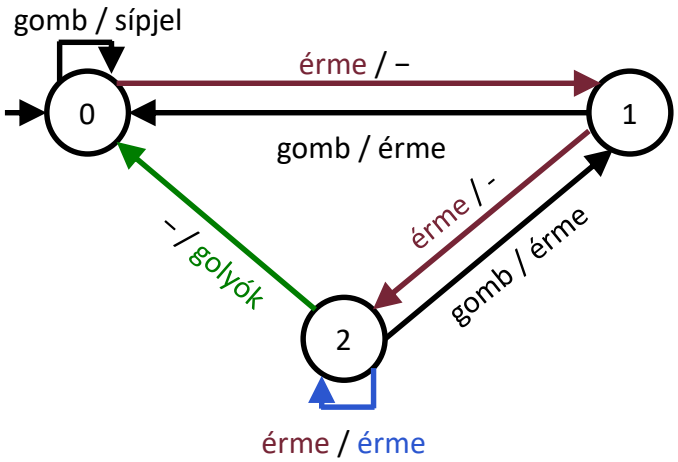
Egyszerű állapotgépek – „Üres esemény”

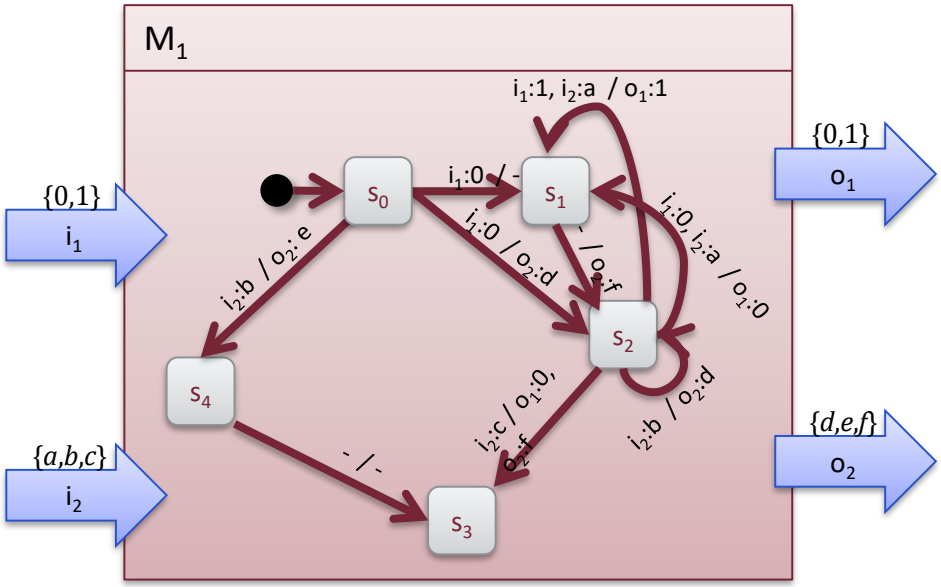
Spontán átmenet, kimenet hiánya:

- $R \subseteq S \times (I \cup \{-\}) \times (O \cup \{-\}) \times S$
- Végrehajtási utakban lehet „–” („üres esemény”)
- Bementi és kimeneti szekvenciákban viszont nem
  - Továbbra is: $\sigma_M \subseteq I^* \times O^*$
  - $(\langle i_0, i_1, \dots \rangle, \langle o_0, o_1, \dots \rangle) \in \sigma_M$ akkor és csak akkor, ha
    - Létezik olyan π útvonal, amiből
    - csak a nem „–” bemenetekre szűrve $\langle i_0, i_1, \dots \rangle$-t kapunk, és
    - csak a nem „–” kimenetekre szűrve$\langle o_0, o_1, \dots \rangle$-t kapunk
- Spontán átmenet általában **nemdeterminisztikus**

Determinisztikusság statikus ellenőrzése

- Determinisztikusság **elégséges** feltételei:
  - Legfeljebb egy kezdőállapot
  - Minden állapotban, bármely bemeneti esemény hatására legfeljebb egy tranzíció tüzelhet
  - Nincs spontán állapotátmenet ?

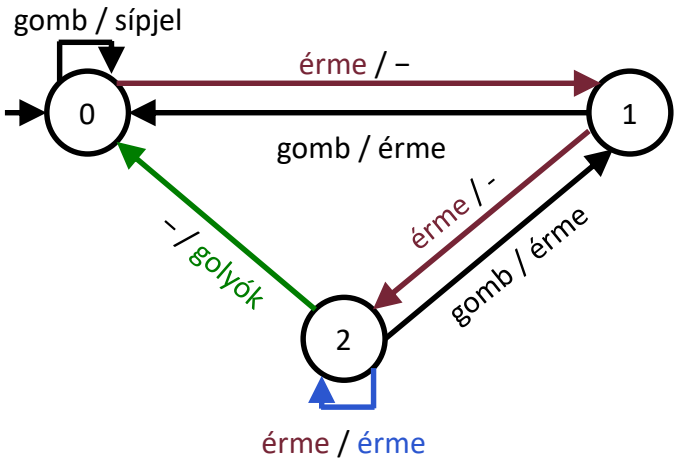


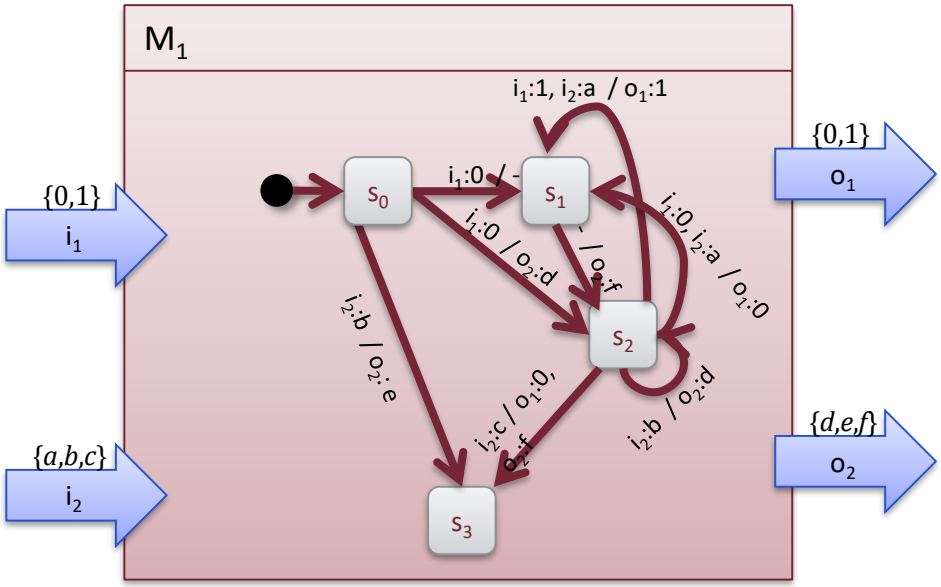


$$\sigma_M(\langle \text{érme}, \text{érme}, \text{érme} \rangle) = \{\langle \text{érme} \rangle, \langle \text{golyók} \rangle\}$$

Determinisztikusság statikus ellenőrzése

- Determinisztikusság **elégséges** feltételei:
  - Legfeljebb egy kezdőállapot
  - Minden állapotban, bármely bemeneti esemény hatására legfeljebb egy tranzíció tüzelhet
  - Nincs spontán állapotátmenet ?





$\sigma_M(\langle \text{érme}, \text{érme}, \text{érme} \rangle) = \{\langle \text{érme} \rangle, \langle \text{golyók} \rangle\}$

Folyamatmodellek

- Döntési csomópontok kimenő ágain lévő őrfeltételek vizsgálata: $g_1, g_2, \dots, g_n$
- **Teljesen specifikált döntés:**
  - Az őrfeltételek **teljes** feltételrendszert alkotnak
  - $g_1 \vee g_2 \vee \cdots \vee g_n$ minden esetben **igaz**
- **Determinisztikus döntés:**
  - Az őrfeltételek **kizárólagos** feltételrendszert alkotnak
  - $\bigvee_{i \neq j}(g_i \wedge g_j)$ minden esetben **hamis** (max. egy g_i igaz)
- **Feltételesen kizárólagos/teljes**
  - Adott feltételek esetén: $\varphi \rightarrow \psi(g_1, \dots, g_n)$

Folyamatmodellek

- Döntési csomópontok kimenő ágain lévő őrfeltételek vizsgálata: g_1, g_2
- **Teljesen specifikált döntés:**
  - Az őrfeltételek **teljes** feltételrendszert alkotnak
  - $g_1 \vee g_2$ minden esetben **igaz**
- **Determinisztikus döntés:**
  - Az őrfeltételek **kizárólagos** feltételrendszert alkotnak
  - $g_1 \wedge g_2$ minden esetben **hamis**
- **Feltételesen kizárólagos/teljes**
  - Adott feltételek esetén: $\varphi \rightarrow (g_1 \sim g_2)$

Folyamatmodellek

- Döntési csomópontok kimenő ágain lévő őrfeltételek vizsgálata: $g_1, g_2, \ldots, g_n$
- **Teljesen specifikált döntés:**

$x < 0$
 $x > 0$ X $x < 0$
 $x \geq 0$ ✓

- **Determinisztikus döntés:**

$x \leq 0$
 $x \geq 0$ X $x < 0$
 $x \geq 0$ ✓

- **Feltételesen kizárólagos/teljes**

$x \neq 0$ feltételezés mellett mindkét rossz példa megjavul

Alapfogalmak

Statikus ellenőrzés

Tesztelés

Formális verifikáció

SZOFTVER VERIFIKÁCIÓ

Control-flow automata
Ellenpélda-vezérelt absztrakció finomítás

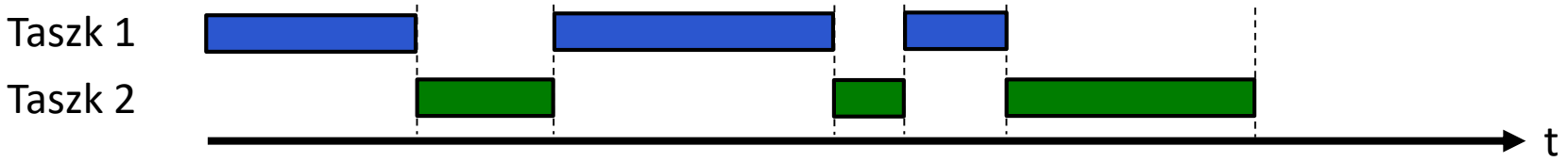
Cél

- **Programkód modellezése állapotgéppel**

→ Szoftverek formális ellenőrzése

- Követelmények:
  - Utasítás végrehajtása – atomi tevékenység
  - (Egyszerű) feltétel kiértékelése – atomi tevékenység
  - **Minden más megszakítható**
    - Végrehajtás során az operációs rendszer bárhol megállhat és átválthat egy másik taszkra, szálra

Nem megszakítható



Programkód modellezése: CFA

- **Control Flow Automata**
- Állapotgép változókkal és kód-specifikus elemekkel
- **Vezérlési helyek** (≈állapot az állapotgépekben)
  - Vezérlési helyek mindig két utasítás között
    - Mint a *breakpoint* debuggolásnál (*„mi a következő sor?”*)
  - Speciális *„end”* hely a végrehajtás végéhez
  - Speciális *„error”* hely az *assert* utasításokhoz
- **Tranzíciók:**
  - Feltételek (*if*, *while*) modellezése őrfeltételekben
  - Utasítás végrehajtása tranzíciókon (atomiság)
  - Itt: egy tranzíción **vagy** őrfeltétel **vagy** egyetlen utasítás

CFA előállítása programkódból - példa

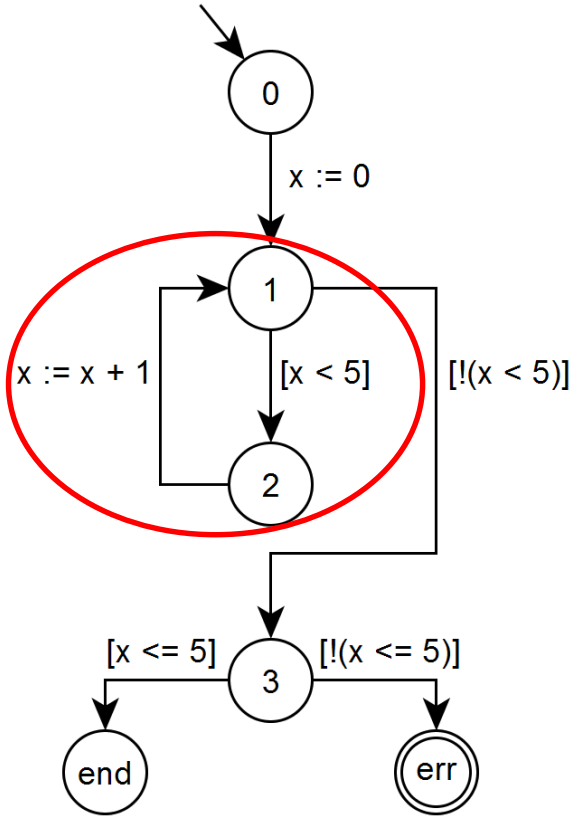
1. Program kiegészítése *assert*–tel
2. Kezdő hely – első kódsor előtt
3. Első utasítás
   - Utasítás végrehajtása utáni hely
     - itt: következő sor eleje
   - Tranzíció utasítással
4. While – feltételkiértékelés
   - Ha teljesül, belemegyünk
   - Ha nem, a következő utasításra ugrunk
5. Ciklusmag - végrehajtás után ugrás az elejére
6. Assertion
   - Ha nem teljesül, hiba – speciális hibahely
   - Ha teljesül, következő utasítás
7. Ha nincs több utasítás, vége – végső hely

```
0: int x=0;
1: while (x<5) {
2:   x=x+1;
   }
3: assert (x<=5)
```

Feltétel és utasítás elválasztása

- Miért nem lehet egy tranzíción egyszerre **őrfeltétel** és **utasítás**?
  - A végrehajtás megállhat a feltétel kiértékelése és az utasítás végrehajtása között.
  - Közben egy másik szál módosíthatja x értékét
    - (példa az előadás végén)

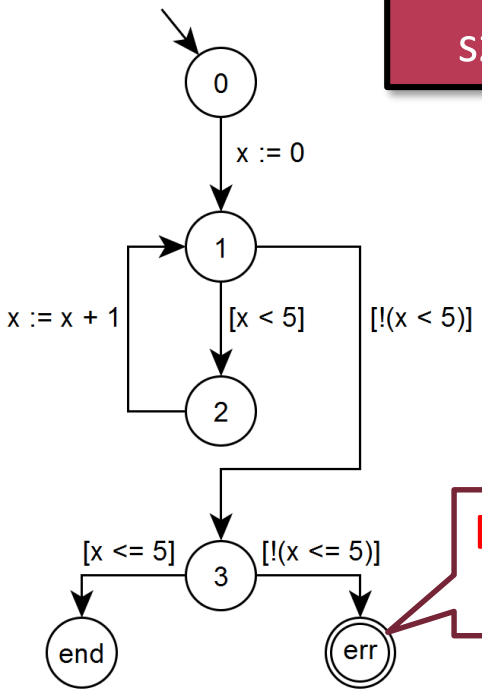
```
0: int x=0;
1: while (x<5) {
2:    x=x+1;
   }
3: assert (x<=5)
```



Szoftver verifikáció – Állapottér-generálás

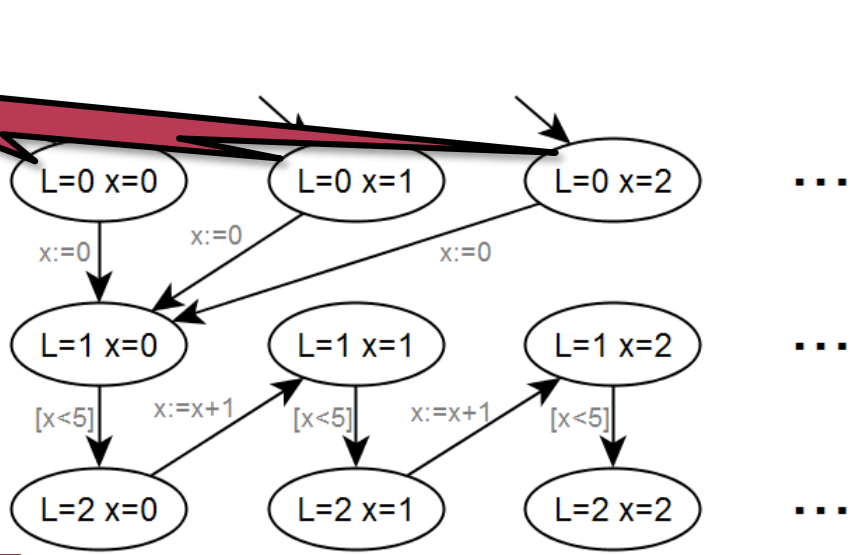
- **Állapot:** vezérlési hely + változók értékei (L, x_1, x_2, …, x_n)
- **Átmenet:** műveletek

CFA



Generált állapottér

(egyszerű állapotgép spontán átmenetekkel)



Memória-szemét

Eljuthatunk ide?

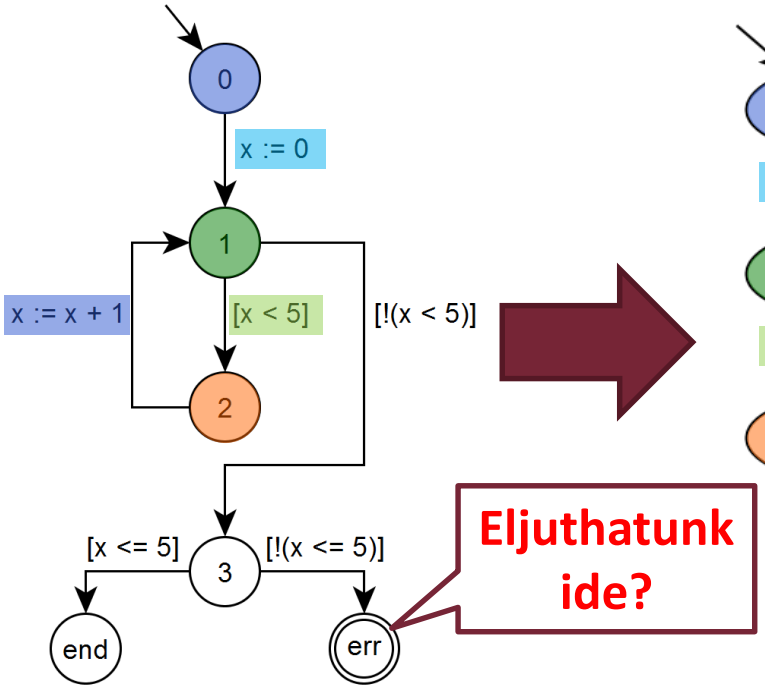
Hibaállapot a potenciális állapottérben – elérhető?

L=err x=6

Szoftver verifikáció – Állapottér-generálás

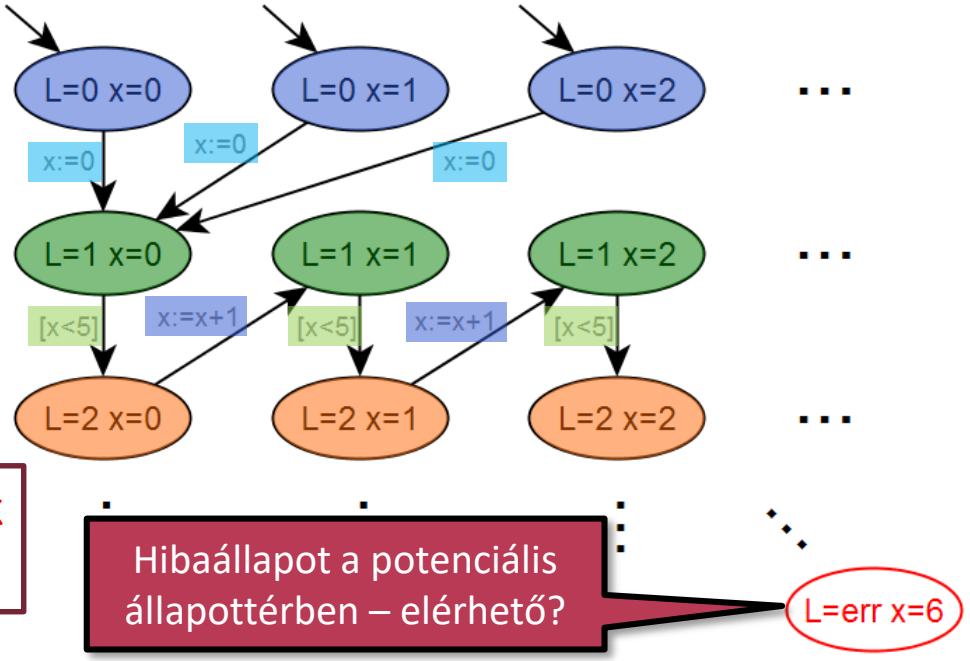
- **Állapot:** vezérlési hely + változók értékei (L, x_1, x_2, …, x_n)
- **Átmenet:** műveletek

CFA



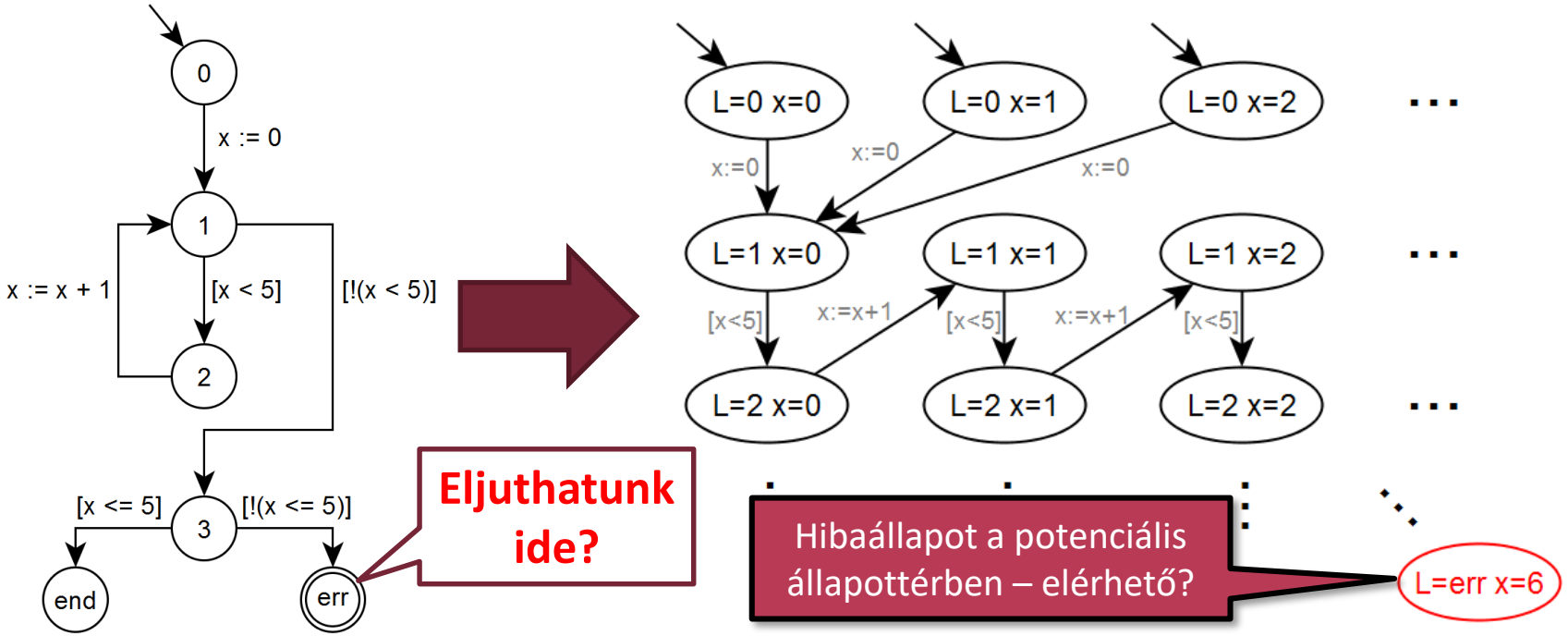
Generált állapottér

(egyszerű állapotgép spontán átmenetekkel)



Szoftver verifikáció – Állapottér-generálás

- **Állapot:** vezérlési hely + változók értékei (L, x_1, x_2, …, x_n)
- **Átmenet:** műveletek
- Probléma: **állapottér robbanás** az adatváltozók miatt
  - Pl.: 10 vezérlési hely, 2 db 32 bites int → $10 \cdot 2^{32} \cdot 2^{32}$ lehetséges állapot
- **Cél:** állapottér reprezentáció méretének **csökkentése absztrakcióval**



Szoftver verifikáció – Állapottér-generálás

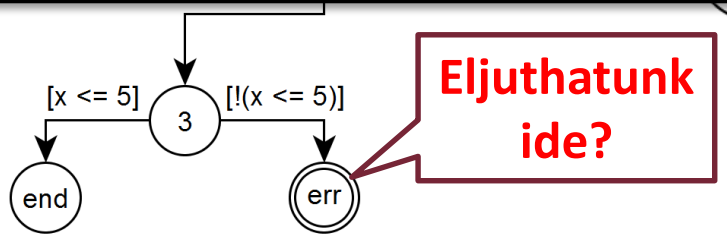
- **Állapot:** vezérlési hely + változók értékei (L, x_1, x_2, …, x_n)
- **Átmenet:** műveletek
- Probléma: **állapottér robbanás** az adatváltozók miatt
  - Pl.: 10 vezérlési hely, 2 db 32 bites int → $10 \cdot 2^{32} \cdot 2^{32}$ lehetséges állapot
- **Cél:** állapottér reprezentáció méretének **csökkentése absztrakcióval**

Egy általános megoldás:

Ellenpélda-vezérelt absztrakció finomítás

(CounterExample-Guided Abstraction Refinement)

CEGAR

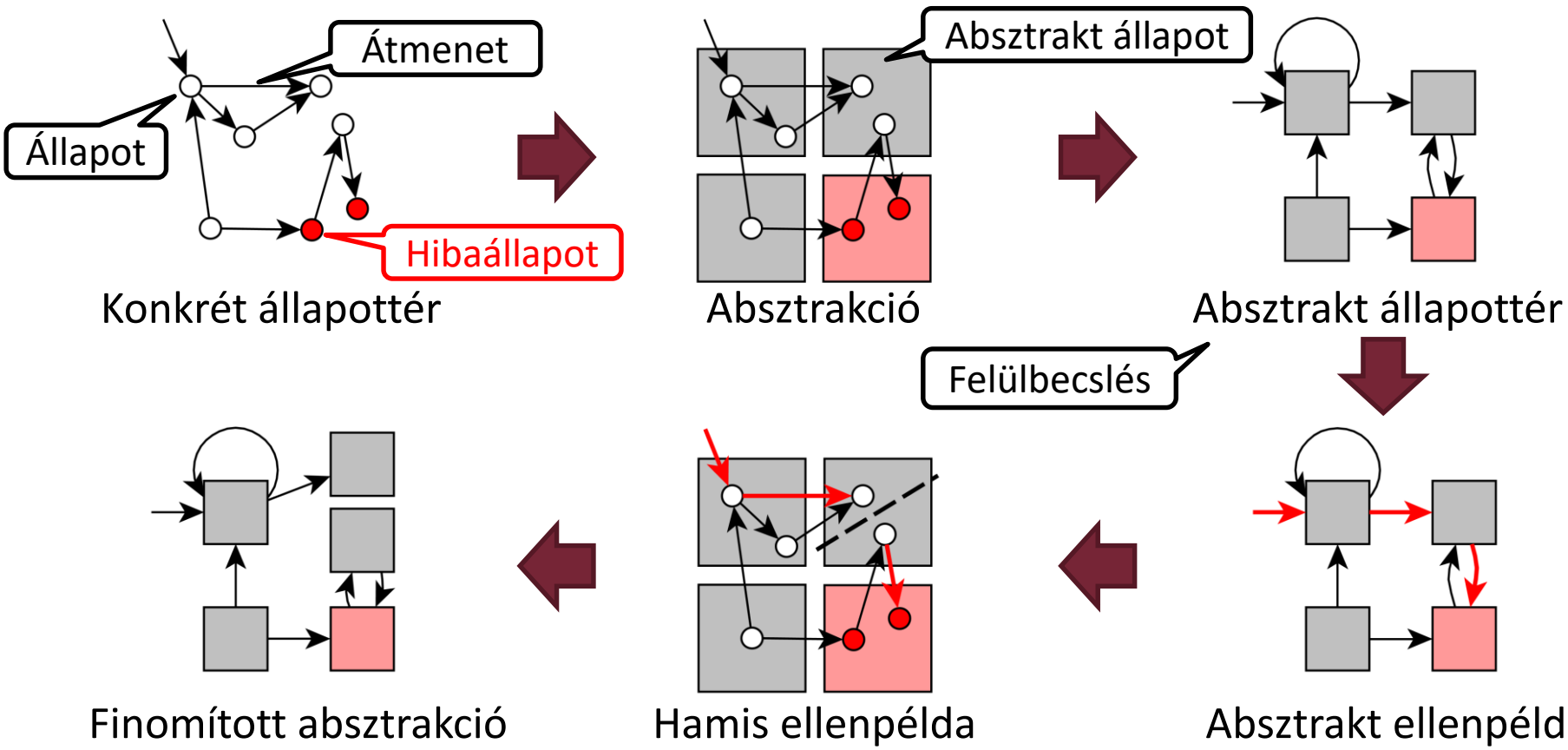


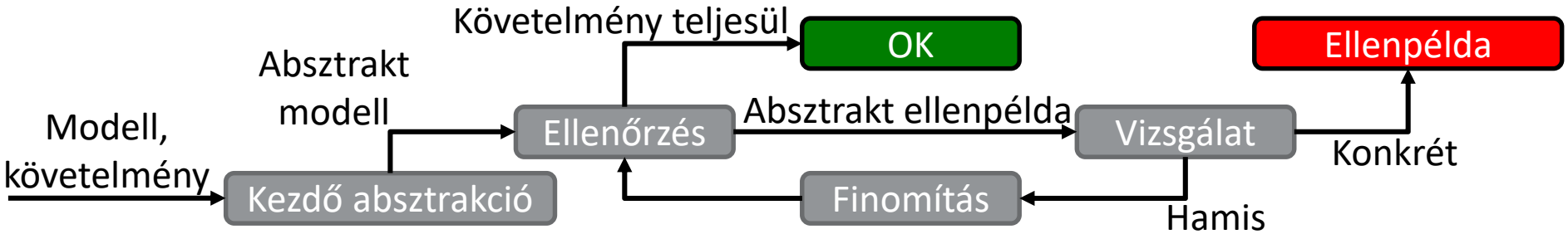
Eljuthatunk ide?

Hibaállapot a potenciális állapottérben – elérhető?

L=err x=6

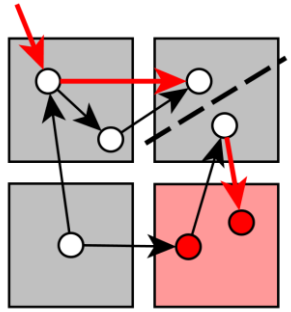
CEGAR – Áttekintés





CEGAR – Absztrakció fajtái

- **Predikátumabsztrakció**
  - **Kezdetben:** csak vezérlési hely
  - **Finomítás:** az elakadt állapot két felében legyen más az új predikátum igazságértéke
  - Jól használható nagy értékkészletek esetén



- **Változó láthatóság** (vetítés)
  - **Kezdetben:** csak vezérlési hely
  - **Finomítás:** az elakadt állapot két felében legyen más az új változó értéke
  - Jól használható, ha sok (irreleváns) változó van
- **Kombinálható**
  - Egyes változókat látunk, másokról predikátumaink vannak

CEGAR – Összefoglalás

- **Ellenpélda-vezérelt absztrakció finomítás**
  - Counterexample-guided abstraction refinement (CEGAR)
- Kezdetben **durva absztrakción** keresünk hibát
  - Ha nem találunk, nem is lehet
- A talált hibát **szimuláljuk** a konkrét rendszeren
  - Ha szimulálható, akkor valódi a hiba
- Ha **hamis hibát** találtunk, **finomítunk**
  - Úgy, hogy ezt a hibát már ne találhassuk meg

Absztrakt modell = Kisebb modell = Hatékonyság++

SZOFTVER VERIFIKÁCIÓ PÉLDA

Kölcsönös kizárási algoritmus modellellenőrzése

Kölcsönös kizárás algoritmus (pszeudo-kód)

- 2 résztvevőre, 3 megosztott változóval (H. Hyman, 1966)
  - **blocked0**: Első résztvevő (P0) be akar lépni
  - **blocked1**: Második résztvevő (P1) be akar lépni
  - **turn**: Ki következik belépni (0 esetén P0, 1 esetén P1)

P0

```
while (true) {
   blocked0 = true;
   while (turn!=0) {
    while (blocked1==true) {
        skip;
    }
    turn=0;
   }
   // Critical section (cs)
   blocked0 = false;
   // Do other things
}
```

P1

```
while (true) {
   blocked1 = true;
   while (turn!=1) {
    while (blocked0==true) {
        skip;
    }
    turn=1;
   }
   // Critical section (cs)
   blocked1 = false;
   // Do other things
}
```

Helyes-e ez az algoritmus?

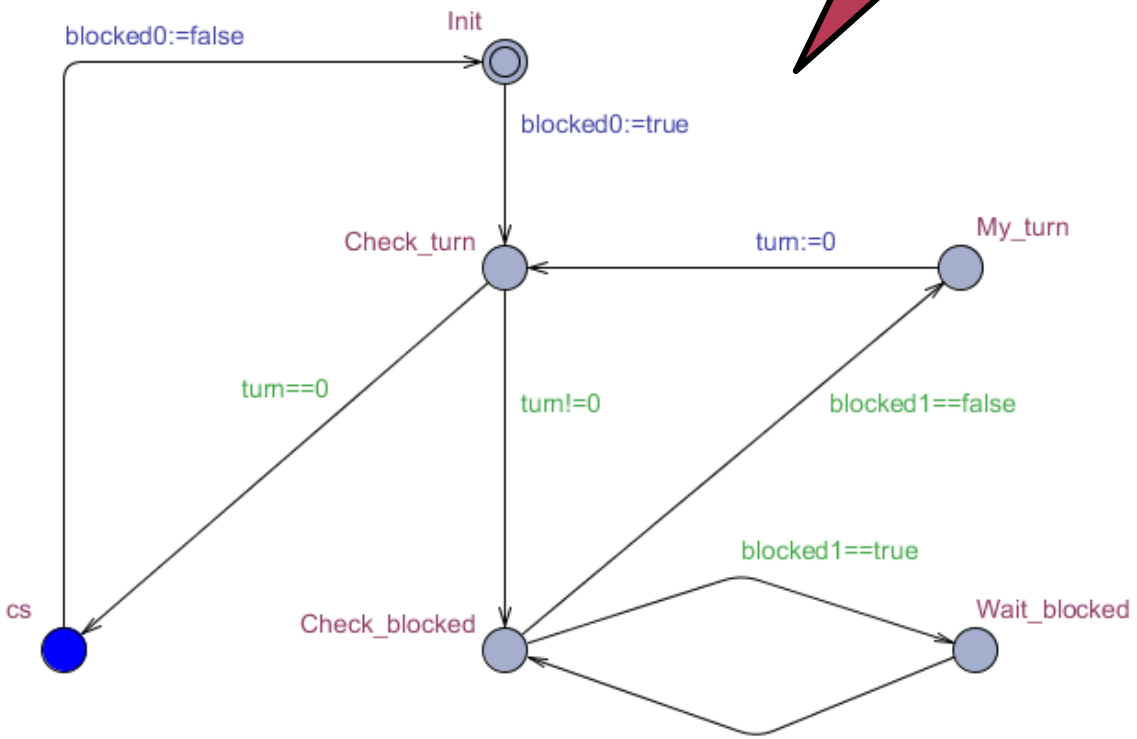
A P0 processz modellje

Deklarációk:

```
bool blocked0;
bool blocked1;
int[0,1] turn=0;
system P0, P1;
```

UPPAAL modellellenőrző eszköz

A P0 automata:



P0

```
while (true) {
    blocked0 = true;
    while (turn!=0) {
        while (blocked1==true) {
            skip;
        }
        turn=0;
    }
    // Critical section
    blocked0 = false;
    // Do other things
}
```

Hibás viselkedés

- **blocked0**: false
- **blocked1**: false
- **turn**: 0

P0

```
while (true) {
   blocked0 = true;
   while (turn!=0) {
    while (blocked1==true) {
        skip;
    }
    turn=0;
   }
   // Critical section (cs)
   blocked0 = false;
   // Do other things
}
```

P1

```
while (true) {
   blocked1 = true;
   while (turn!=1) {
    while (blocked0==true) {
        skip;
    }
    turn=1;
   }
   // Critical section (cs)
   blocked1 = false;
   // Do other things
}
```

Hibás viselkedés

- **blocked0**: false
- **blocked1**: **true**
- **turn**: 0

P0

```
while (true) {
   blocked0 = true;
   while (turn!=0) {
    while (blocked1==true) {
        skip;
    }
    turn=0;
   }
   // Critical section (cs)
   blocked0 = false;
   // Do other things
}
```

P1

```
while (true) {
   blocked1 = true;
   while (turn!=1) {
    while (blocked0==true) {
        skip;
    }
    turn=1;
   }
   // Critical section (cs)
   blocked1 = false;
   // Do other things
}
```

Hibás viselkedés

- **blocked0**: false
- **blocked1**: true
- **turn**: 0

P0

```
while (true) {
   blocked0 = true;
   while (turn!=0) {
    while (blocked1==true) {
        skip;
    }
    turn=0;
   }
   // Critical section (cs)
   blocked0 = false;
   // Do other things
}
```

P1

```
while (true) {
   blocked1 = true;
   while (turn!=1) {
    while (blocked0==true) {
        skip;
    }
    turn=1;
   }
   // Critical section (cs)
   blocked1 = false;
   // Do other things
}
```

Hibás viselkedés

- **blocked0**: false
- **blocked1**: true
- **turn**: 0

P0

```
while (true) {
   blocked0 = true;
   while (turn!=0) {
    while (blocked1==true) {
        skip;
    }
    turn=0;
   }
   // Critical section (cs)
   blocked0 = false;
   // Do other things
}
```

P1

```
while (true) {
   blocked1 = true;
   while (0!=1) {
    while (blocked0==true) {
        skip;
    }
    turn=1;
   }
   // Critical section (cs)
   blocked1 = false;
   // Do other things
}
```

Hibás viselkedés

- **blocked0**: false
- **blocked1**: true
- **turn**: 0

P0

```
while (true) {
   blocked0 = true;
   while (turn!=0) {
    while (blocked1==true) {
        skip;
    }
    turn=0;
   }
   // Critical section (cs)
   blocked0 = false;
   // Do other things
}
```

P1

```
while (true) {
   blocked1 = true;
   while (turn!=1) {
    while (blocked0==true) {
        skip;
    }
    turn=1;
   }
   // Critical section (cs)
   blocked1 = false;
   // Do other things
}
```

Hibás viselkedés

- **blocked0**: false
- **blocked1**: true
- **turn**: 0

P0

```
while (true) {
   blocked0 = true;
   while (turn!=0) {
    while (blocked1==true) {
        skip;
    }
    turn=0;
   }
   // Critical section (cs)
   blocked0 = false;
   // Do other things
}
```

P1

```
while (true) {
   blocked1 = true;
   while (turn!=1) {
    while (false==true) {
        skip;
    }
    turn=1;
   }
   // Critical section (cs)
   blocked1 = false;
   // Do other things
}
```

Hibás viselkedés

- **blocked0**: **true**
- **blocked1**: true
- **turn**: 0

P0

```
while (true) {
   blocked0 = true;
   while (turn!=0) {
    while (blocked1==true) {
        skip;
    }
    turn=0;
   }
   // Critical section (cs)
   blocked0 = false;
   // Do other things
}
```

P1

```
while (true) {
   blocked1 = true;
   while (turn!=1) {
    while (false==true) {
        skip;
    }
    turn=1;
   }
   // Critical section (cs)
   blocked1 = false;
   // Do other things
}
```

Hibás viselkedés

- **blocked0**: true
- **blocked1**: true
- **turn**: 0

P0

```
while (true) {
   blocked0 = true;
   while (turn!=0) {
    while (blocked1==true) {
        skip;
    }
    turn=0;
   }
   // Critical section (cs)
   blocked0 = false;
   // Do other things
}
```

P1

```
while (true) {
   blocked1 = true;
   while (turn!=1) {
    while (false==true) {
        skip;
    }
    turn=1;
   }
   // Critical section (cs)
   blocked1 = false;
   // Do other things
}
```

Hibás viselkedés

- **blocked0**: true
- **blocked1**: true
- **turn**: 0

P0

```
while (true) {
   blocked0 = true;
   while (0!=0) {
    while (blocked1==true) {
        skip;
    }
    turn=0;
   }
   // Critical section (cs)
   blocked0 = false;
   // Do other things
}
```

P1

```
while (true) {
   blocked1 = true;
   while (turn!=1) {
    while (false==true) {
        skip;
    }
    turn=1;
   }
   // Critical section (cs)
   blocked1 = false;
   // Do other things
}
```

Hibás viselkedés

- **blocked0**: true
- **blocked1**: true
- **turn**: 0

P0

```
while (true) {
   blocked0 = true;
   while (turn!=0) {
    while (blocked1==true) {
        skip;
    }
    turn=0;
   }
   // Critical section (cs)
   blocked0 = false;
   // Do other things
}
```

P1

```
while (true) {
   blocked1 = true;
   while (turn!=1) {
    while (false==true) {
        skip;
    }
    turn=1;
   }
   // Critical section (cs)
   blocked1 = false;
   // Do other things
}
```

Hibás viselkedés

- **blocked0**: true
- **blocked1**: true
- **turn**: **1**

P0

```
while (true) {
   blocked0 = true;
   while (turn!=0) {
    while (blocked1==true) {
        skip;
    }
    turn=0;
   }
   // Critical section (cs)
   blocked0 = false;
   // Do other things
}
```

P1

```
while (true) {
   blocked1 = true;
   while (turn!=1) {
    while (blocked0==true) {
        skip;
    }
    turn=1;
   }
   // Critical section (cs)
   blocked1 = false;
   // Do other things
}
```

Hibás viselkedés

- **blocked0**: true
- **blocked1**: true
- **turn**: 1

P0

```
while (true) {
   blocked0 = true;
   while (turn!=0) {
    while (blocked1==true) {
        skip;
    }
    turn=0;
   }
   // Critical section (cs)
   blocked0 = false;
   // Do other things
}
```

P1

```
while (true) {
   blocked1 = true;
   while (turn!=1) {
    while (blocked0==true) {
        skip;
    }
    turn=1;
   }
   // Critical section (cs)
   blocked1 = false;
   // Do other things
}
```

Hibás viselkedés

- **blocked0**: true
- **blocked1**: true
- **turn**: 1

P0

```
while (true) {
   blocked0 = true;
   while (turn!=0) {
    while (blocked1==true) {
        skip;
    }
    turn=0;
   }
   // Critical section (cs)
   blocked0 = false;
   // Do other things
}
```

P1

```
while (true) {
   blocked1 = true;
   while (1!=1) {
    while (blocked0==true) {
        skip;
    }
    turn=1;
   }
   // Critical section (cs)
   blocked1 = false;
   // Do other things
}
```

Hibás viselkedés

- **blocked0**: true
- **blocked1**: true
- **turn**: 1

P0

```
while (true) {
   blocked0 = true;
   while (turn!=0) {
    while (blocked1==true) {
        skip;
    }
    turn=0;
   }
   // Critical section (cs)
   blocked0 = false;
   // Do other things
}
```

P1

```
while (true) {
   blocked1 = true;
   while (turn!=1) {
    while (blocked0==true) {
        skip;
    }
    turn=1;
   }
   // Critical section (cs)
   blocked1 = false;
   // Do other things
}
```

Hibás viselkedés