Modellalapú szoftverfejlesztés

X. előadás

Gráfmintaillesztés,
Gráftranszformáció

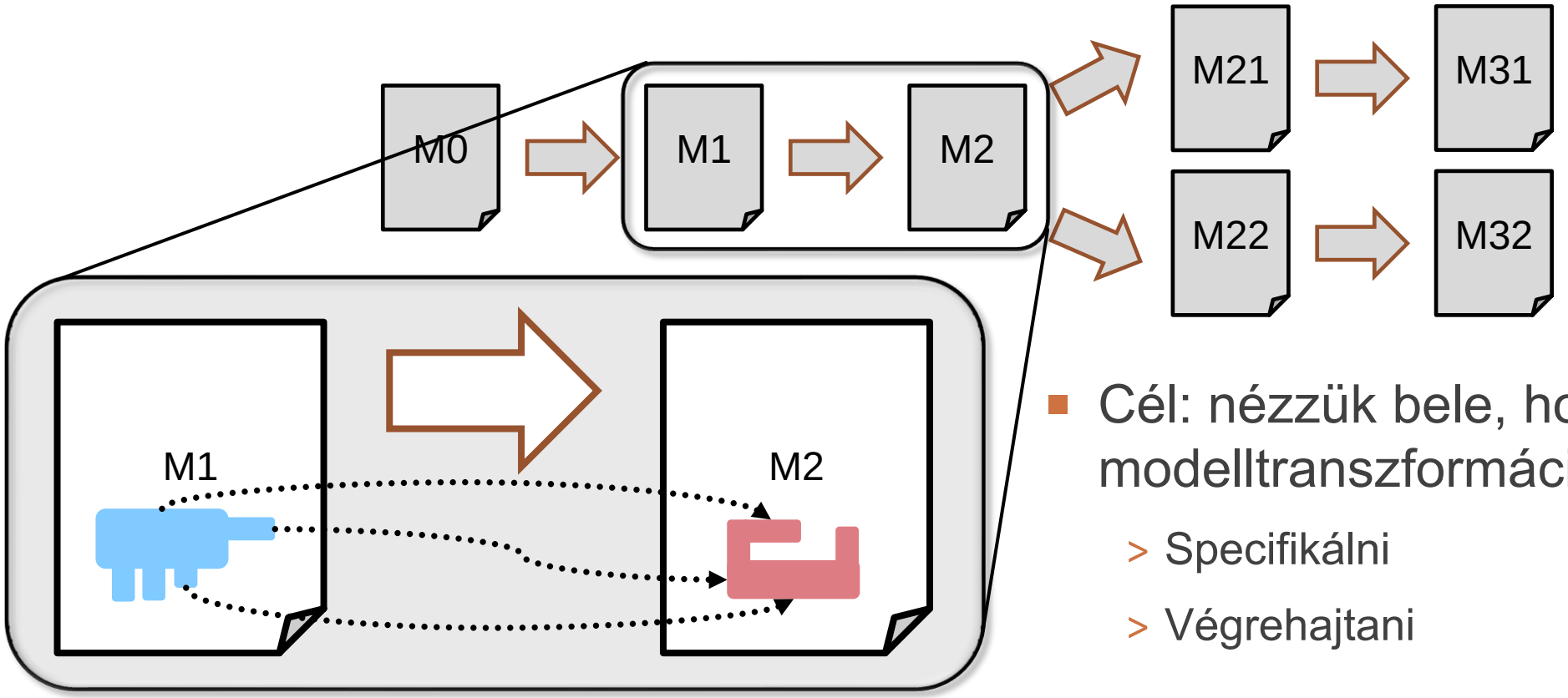
Ficsor Attila

Gráfmintaillesztés, Gráftranszformáció

- **Alapfogalmak**
- **Gráfmintaillesztés**
- **Modelltranszformációk**
- **Inkrementális transzformációk**
- **Tervezésitér bejárás**

Motiváció: Modellek transzformációja

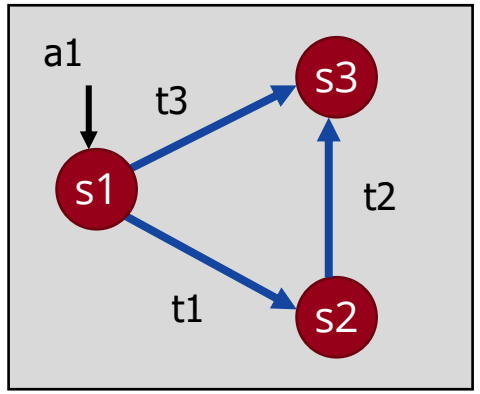
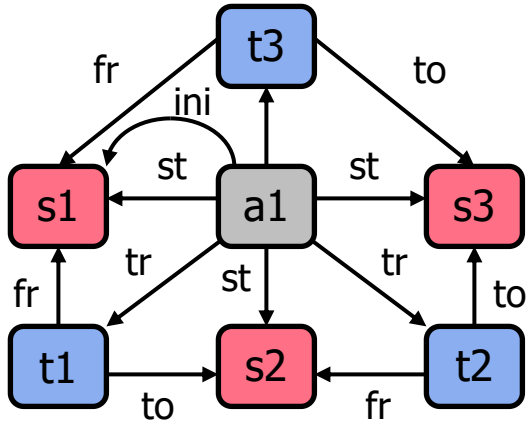
- **Modellalapú fejlesztés:** Modellek az elsődleges dokumentumok
- Modelleket fejlesztünk, automatizáljuk a modellfeldolgozást

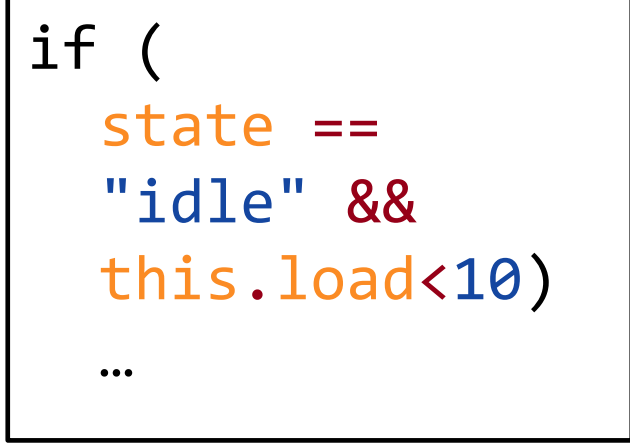
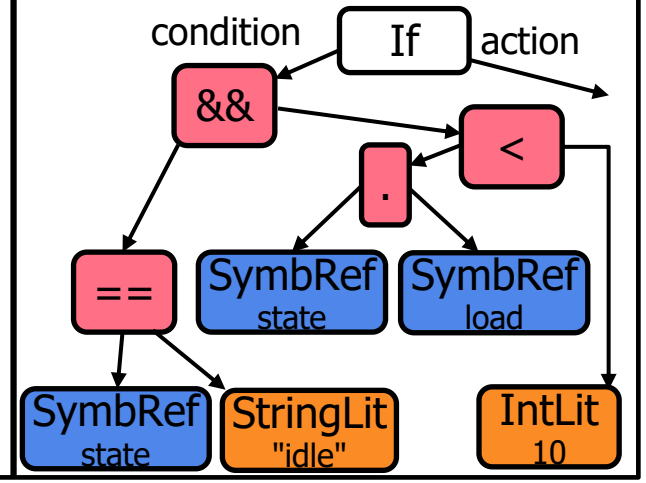


- Cél: nézzük bele, hogyan lehet modelltranszformációkat
  - > Specifikálni
  - > Végrehajtani

Absztrakt szintaxis

- **Hogyan módosítsuk a modelleket?**
- **Ötlet:** módosítsuk modellek reprezentációját közvetlenül → **Absztrakt szintaxis**

Konkrét	Absztrakt
	

Konkrét	Absztrakt
	

```
if (
   state ==
   "idle" &&
   this.load<10)
   …
```

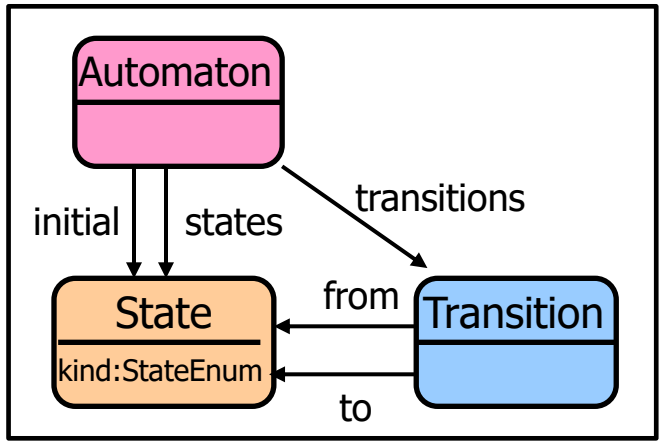

- **Feladat:** gráfok módosítására módszer!

Gráfmintaillesztés, Gráftranszformáció

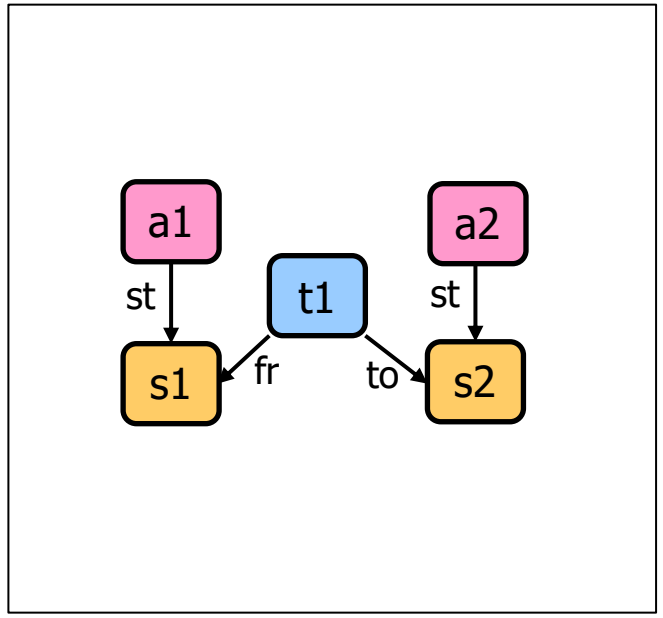
- Alapfogalmak
- **Gráfmintaillesztés**
- Modelltranszformációk
- Inkrementális transzformációk
- Tervezésitér bejárás

Egy egyszerű példa

Metamodell



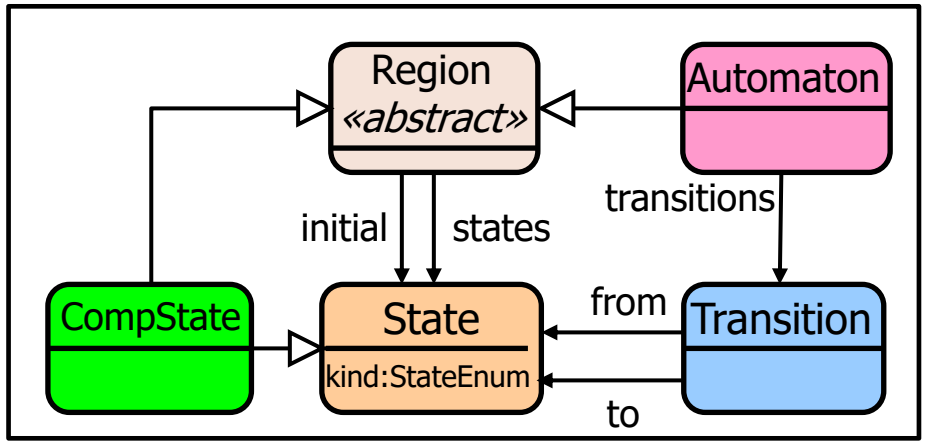
Szabálysértés példa



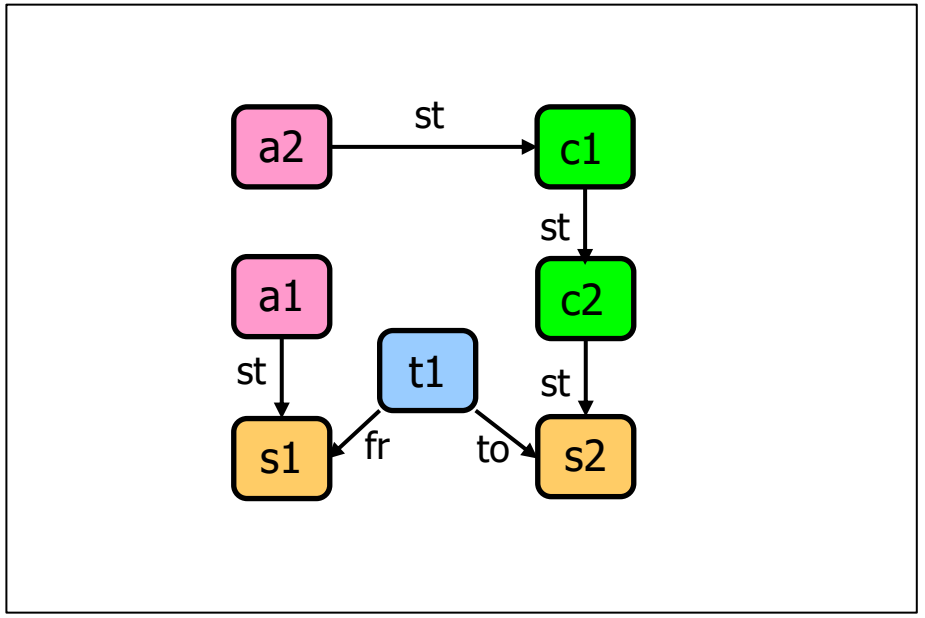
- Jólformáltsági kényszer:
  - Tranzíció forrás- és célállapotának ugyanabban az állapotgépben kell lennie
- Cél: szabálysértések megtalálása...
  - A szabályszegés egy *Tranzíció*, aminek „*from*” éle egy *s1 State*-re mutat, és „*to*” éle egy *s2 State*-re mutat, ahol *s1 állapotgépe* nem *s2* állapotgépe

Egy összetettebb példa

Metamodell



Szabálysértés példa



- Jólformáltsági kényszer:
  - Tranzíció forrás- és célállapotának ugyanabban az állapotgépben kell lennie
- Cél: szabálysértések megtalálása...
  - A szabályszegés egy *Tranzíció*, aminek „*from*” éle egy *s1 State*-re mutat, és „*to*” éle egy *s2 State*-re mutat, ahol *s1* állapotgépe nem *s2* állapotgépe

Programozott bejárás vs. lekérdezések

- **Cél:** kényszer megsértéseinek megtalálása a modellben

A modell bejárása általános célú nyelven

```
for (Automaton automaton : automatons) {
  for (Transition transition : automaton.getTransitions()) {
    State sourceState = transition.from;
    // melyik automaton definiálja ezt az állapotot?
    Automaton sourceAutomaton = null;
    for (Automaton candidate : automatons) {
      if (candidate.getStates().contains(sourceState)) {
        sourceAutomaton = candidate;
        break;
      }
    }
    // ... ugyanezt a targetState esetében, majd
    if (sourceAutomaton != targetAutomaton)
      // szabálysértés jelentése
  }
}
```

„egyszerű példa”

Programozott bejárás vs. lekérdezések

- Cél: kényszer megsértéseinek megtalálása a modellben
  - A modell bejárása általános célú nyelven
  - Használjunk egy **lekérdezési DSL-t**
    - Tömörebb
    - A lekérdezés **deklaratív** funkcionális specifikációja
    - Szabadon értelmezhető a **lekérdezőmotor** (query engine) által (pl. optimalizálás)
    - Platformfüggetlen lehet
- A validálás csak egy felhasználási módja a **modell-lekérdezéseknek**
  - Származtatott tulajdonságok
  - M2M/M2T transzformáció, Szimuláció
  - …

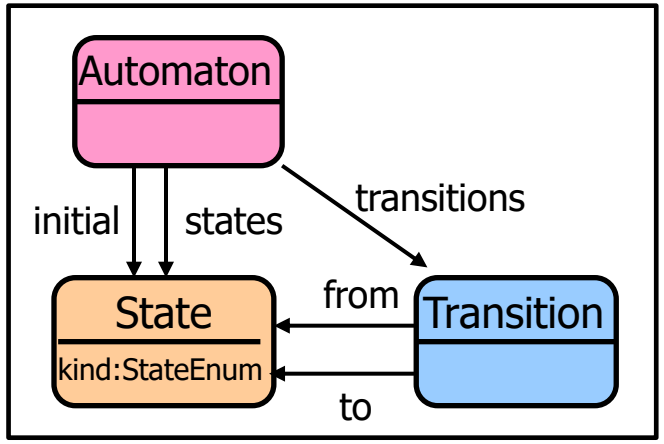
Lekérdezési nyelvi stílusok

- SQL-szerű (relációs algebra)
  - > Példa: EMF Query
  - > ☺ Jó az attribútum korlátozásokhoz
  - > ☹ Nem túl tömör a kapcsolatokra (sok join)
- Funkcionális stílus
  - > Példa: OCL
  - > Valamelyest deklaratív

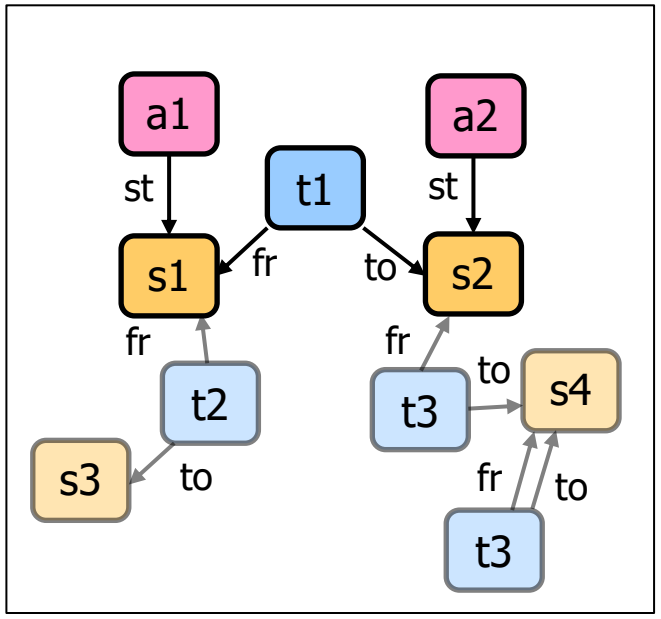
```
context Transition inv:
  Automaton.allInstances()->forAll(a |
    a.states->includes(self.from) =
    a.states->includes(self.to)
  );
```

- Logikai stílus
  - > Domain relációs kalkulus / gráfminták / Datalog
  - > Még inkább deklaratív

Metamodell



Szabálysértés példa



A szabályszegés egy Tranzíció, aminek „from” éle egy s1 State-re mutat, és „to” éle egy s2 State-re mutat, ahol s1 állapotgépe nem s2 állapotgépe

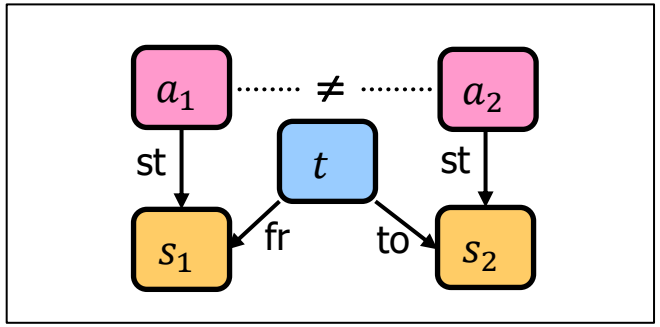
Formális logikával (Domain Relációs Kalkulus)

$$\{t \mid \exists s_1, s_2, a_1, a_2 : Transition(t) \wedge from(t, s_1) \wedge to(t, s_2) \wedge states(a_1, s_1) \wedge states(a_2, s_2) \wedge a_1 \neq a_2\}$$

```
violates(t) <->
Transition(t),from(t,s1),to(t,s2),
states(a1,s1),states(a2,s2),a1!=a2
```

Datalog-szerű lekérdezési nyelvek

Lekérdezőmotor



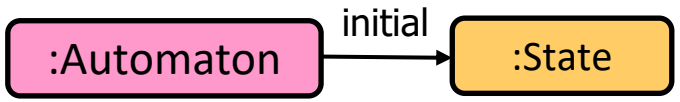
Minta

Minta anatómiája

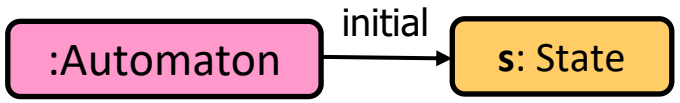
- A mintában szereplő típusok és referenciák egy metamodellben vannak definiálva
- Egy minta bevezet néhány csomópont változót

Pl. kettő csomópont: 

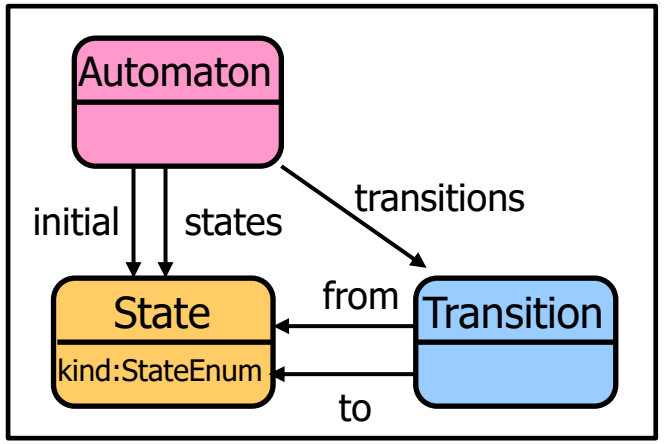
- És definiál köztük viszonyokat és állításokat



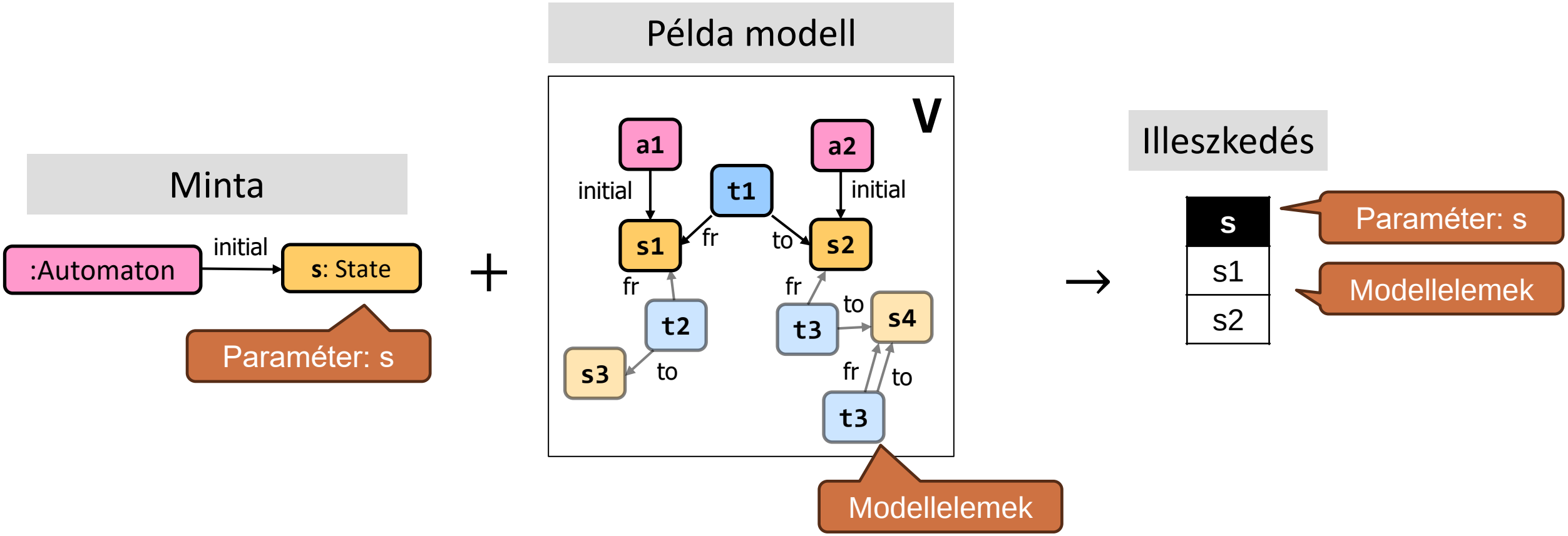
- Némely változót megnevezünk



Metamodell

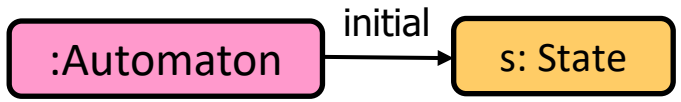


Minta illesztése

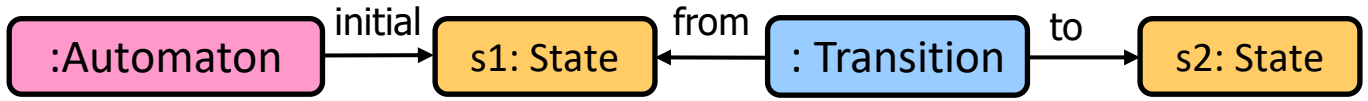


Példák

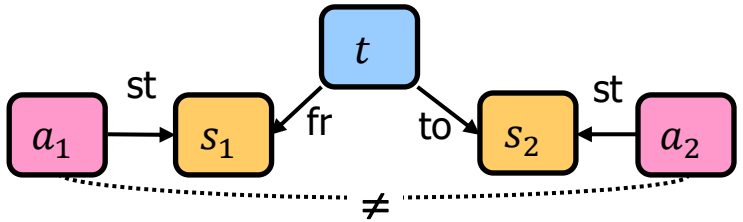
- **Egyszerű példa:** kezdőállapotok a modellben



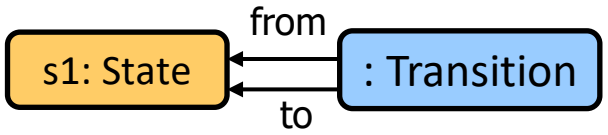
- **Láncolat (∧):** Második állapotok a modellben



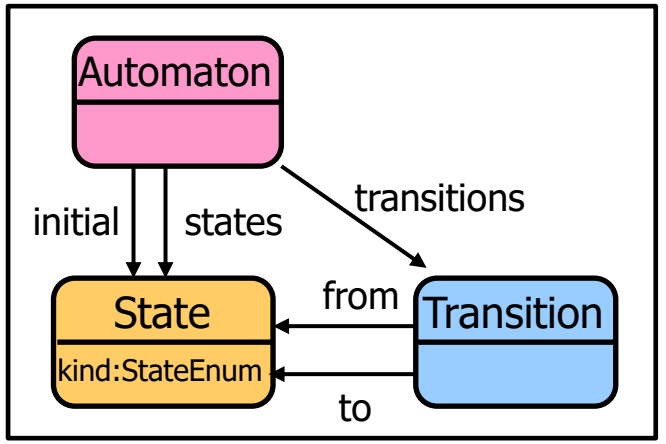
- **≠ :** Automatákon átívelő tranzíció



- =: Hurokél

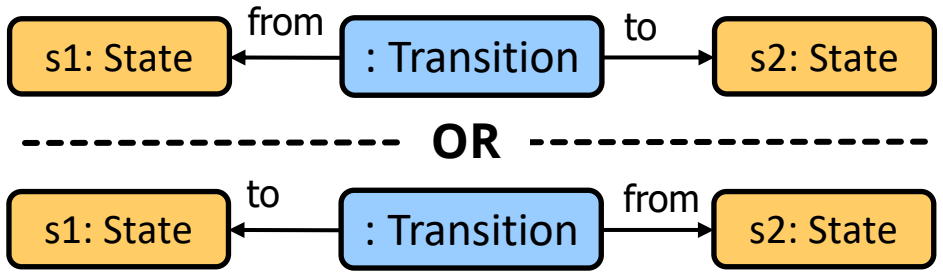


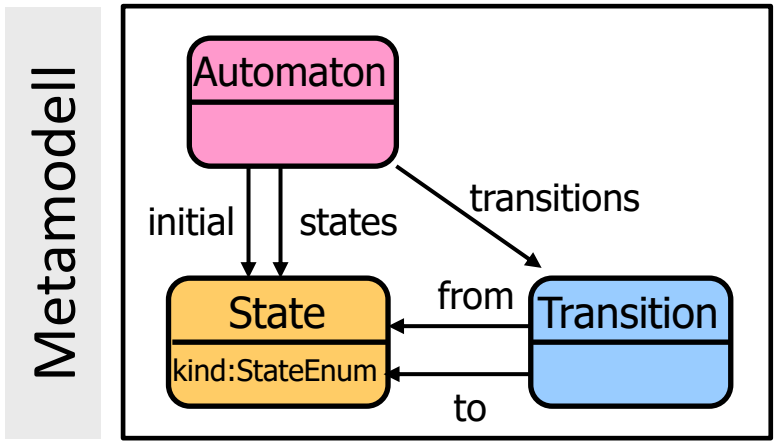
Metamodell



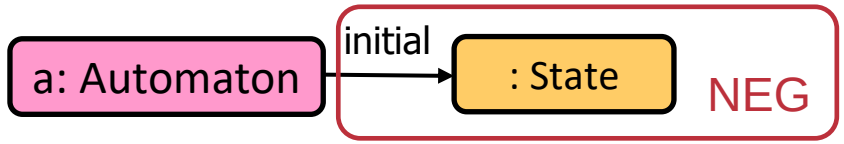
Példák 2

- (∨): Két állapot össze van kötve

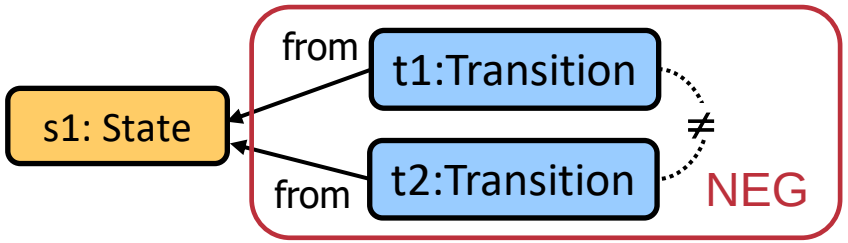




- (¬, Negative Application Condition):
  - > kezdőállapot nélküli automata



  - > olyan állapot, aminek a kezdőállapotából nem megy ki két tranzakció (determinisztikus)

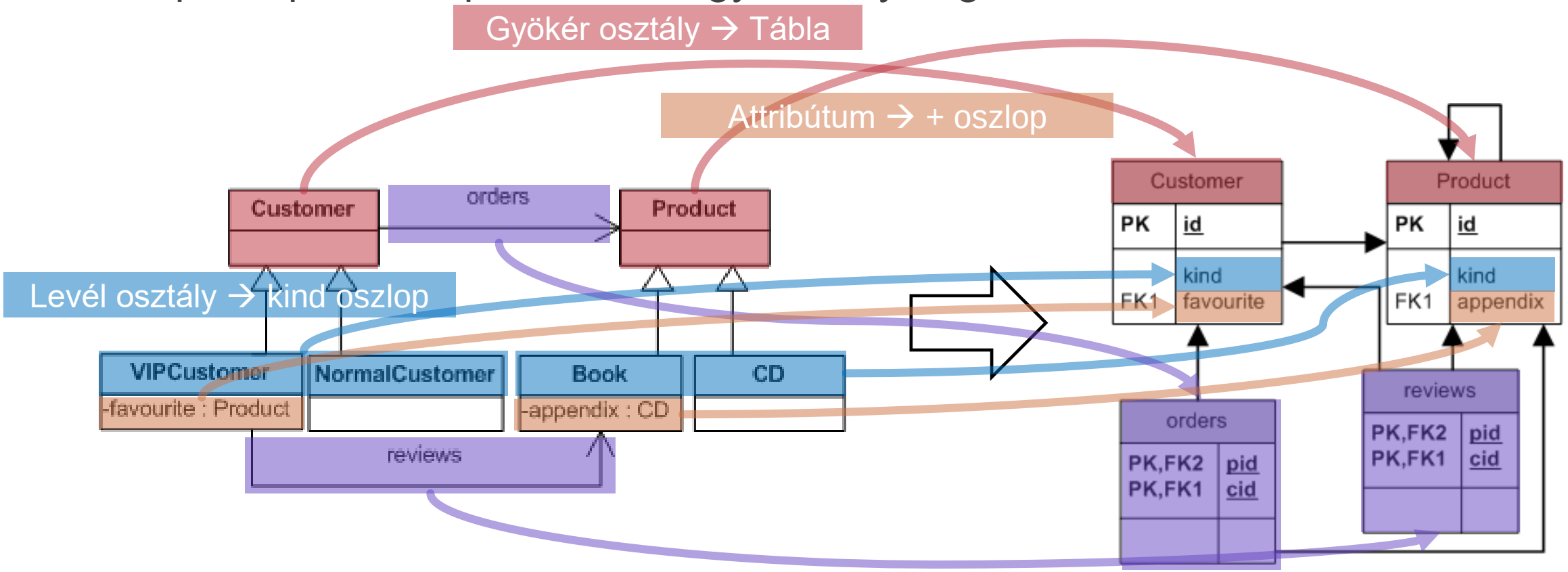


Gráfmintaillesztés, Gráftranszformáció

- Alapfogalmak
- Gráfmintaillesztés
- **Modelltranszformációk**
- Inkrementális transzformációk
- Tervezésitér bejárás

Példa Transzformáció

- Tipikus példa: képezzünk le egy osztálydiagramot adatbázis táblákra!

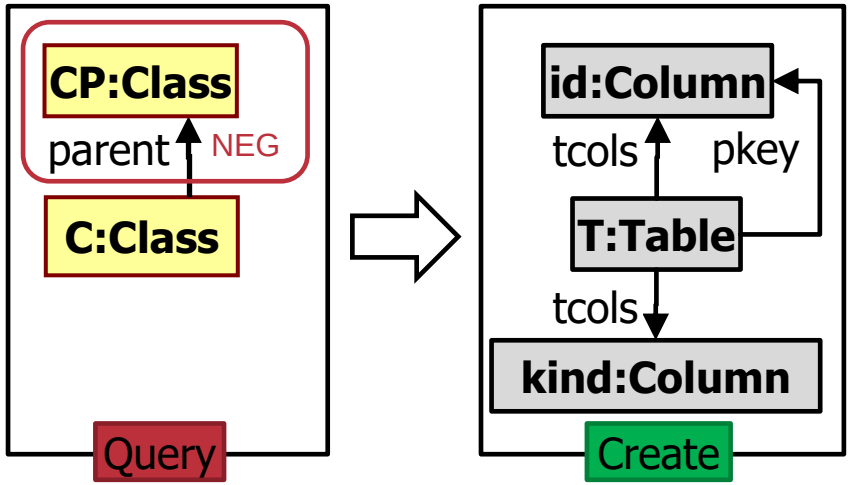


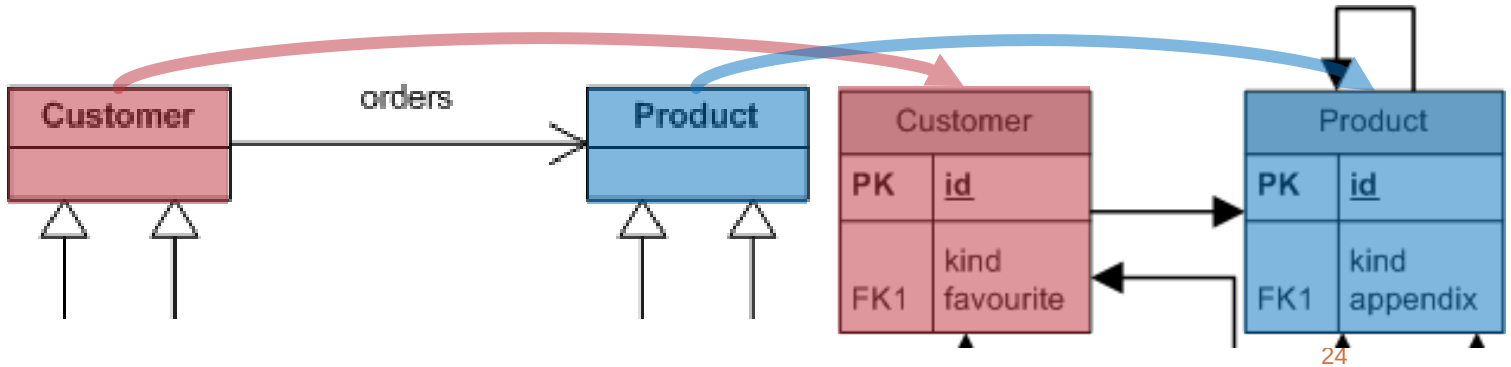
Példa Transzformáció

- Hogyan oldanánk a gyökér osztályokat reprezentáló táblák létrehozását?

1. Lekérdeznénk a gyökér osztályokat (osztály, aminek nincs őse)
2. Létrehoznánk a táblákat, és velük a szükséges oszlopokat
3. Ismételnénk amíg tudjuk

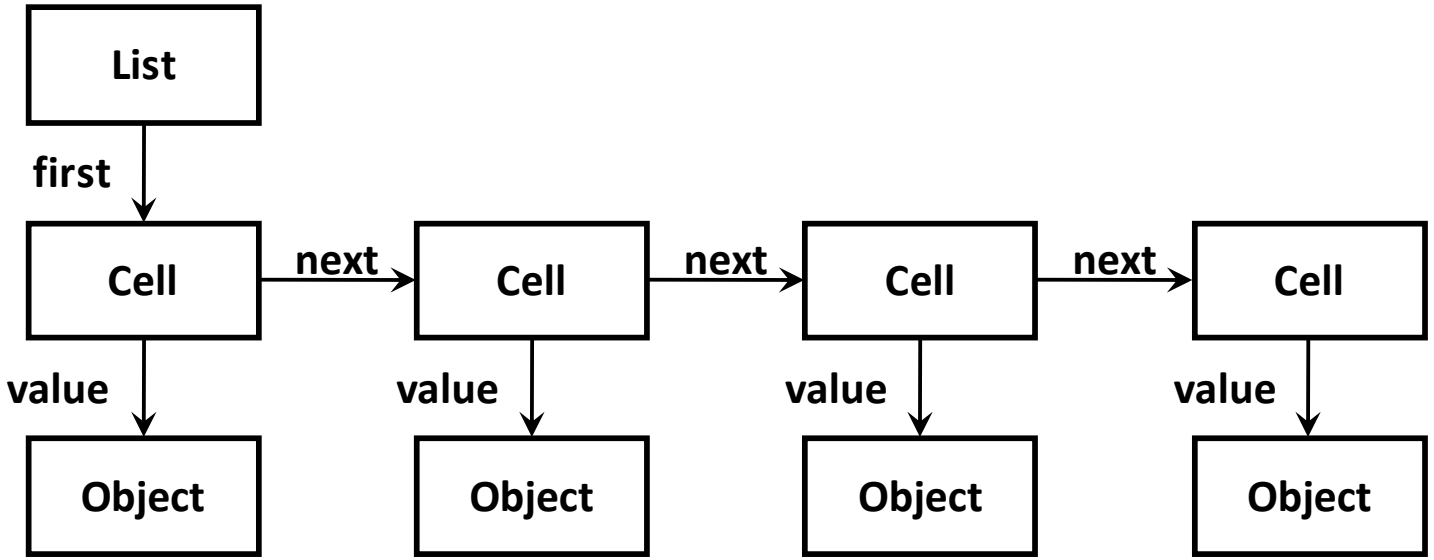
- Cél: Hasonló szabályokkal megfogalmazni az egész transzformációt





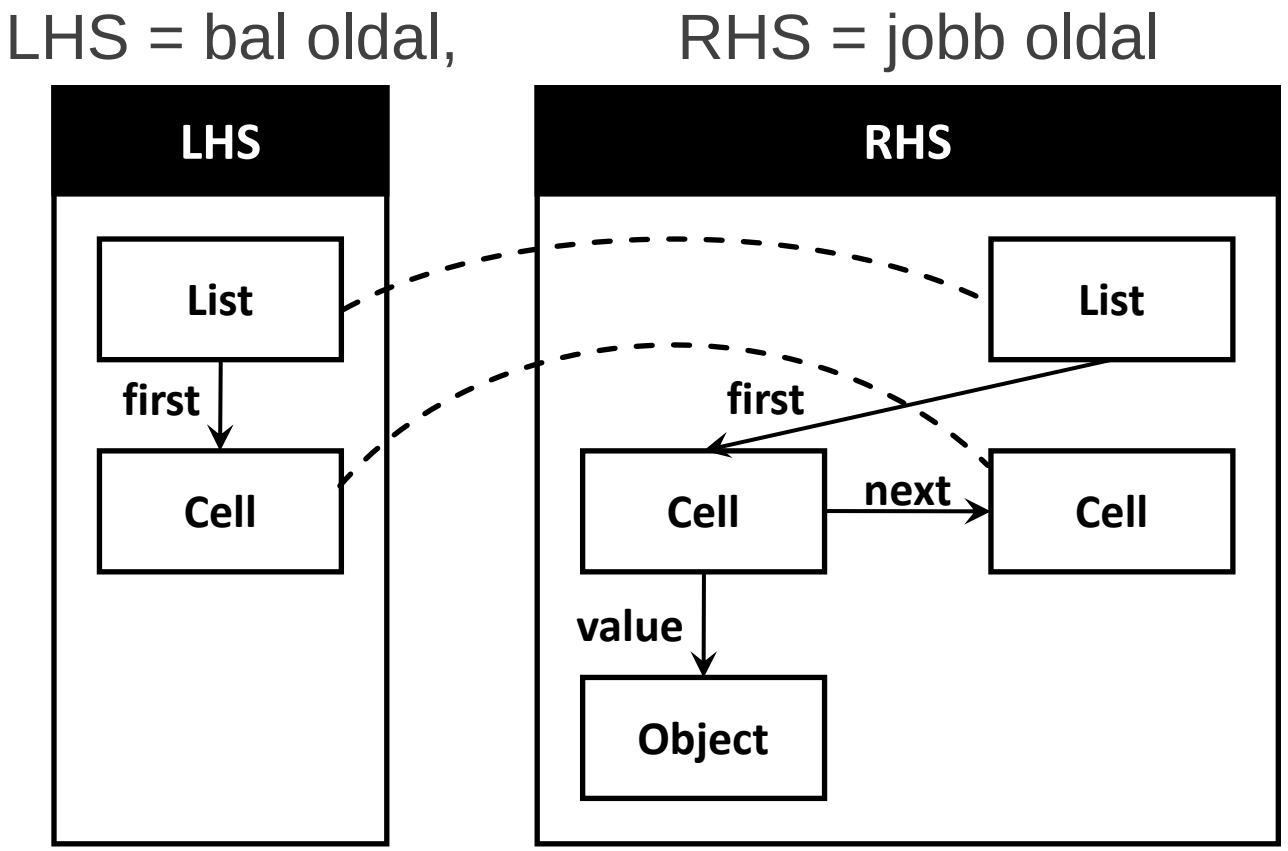
Gráftranszformáció

- Modell = Címkézett gráf



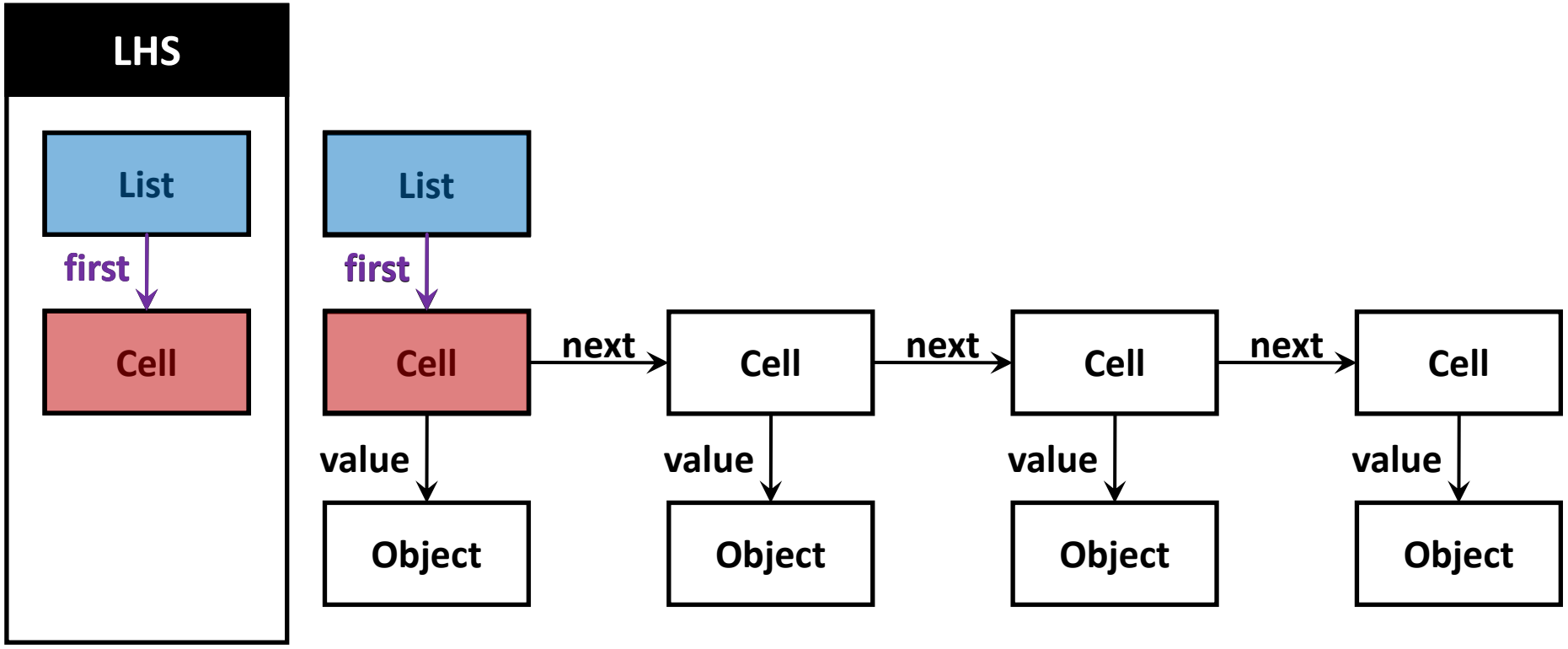
Gráftranszformációs szabály

- Gráf átírási szabály, két gráffal van megfogalmazva



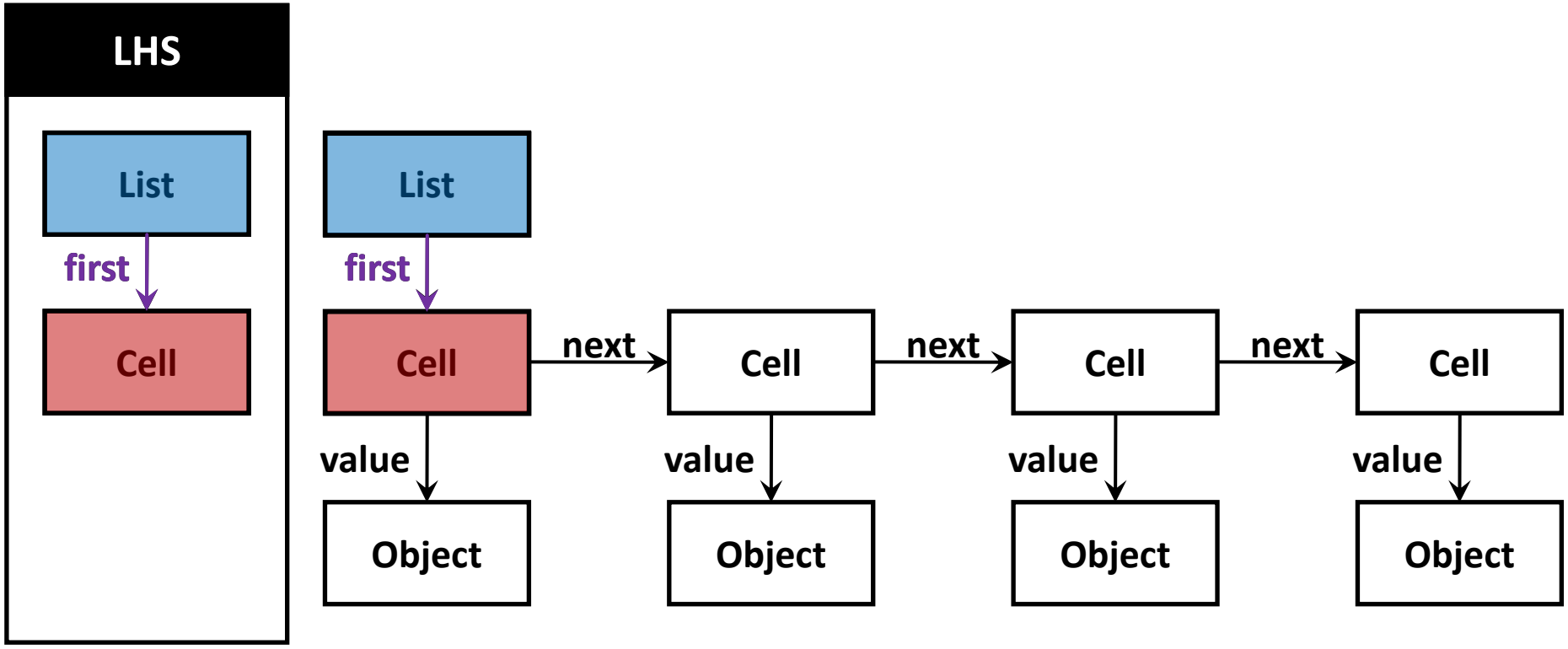
Gráftranszformáció: Mintaillesztés

- **Illesztés:** megkeressük a LHS-t tartalmazó részgráfokat a forrás gráfban



Gráftranszformáció: Mintaillesztés

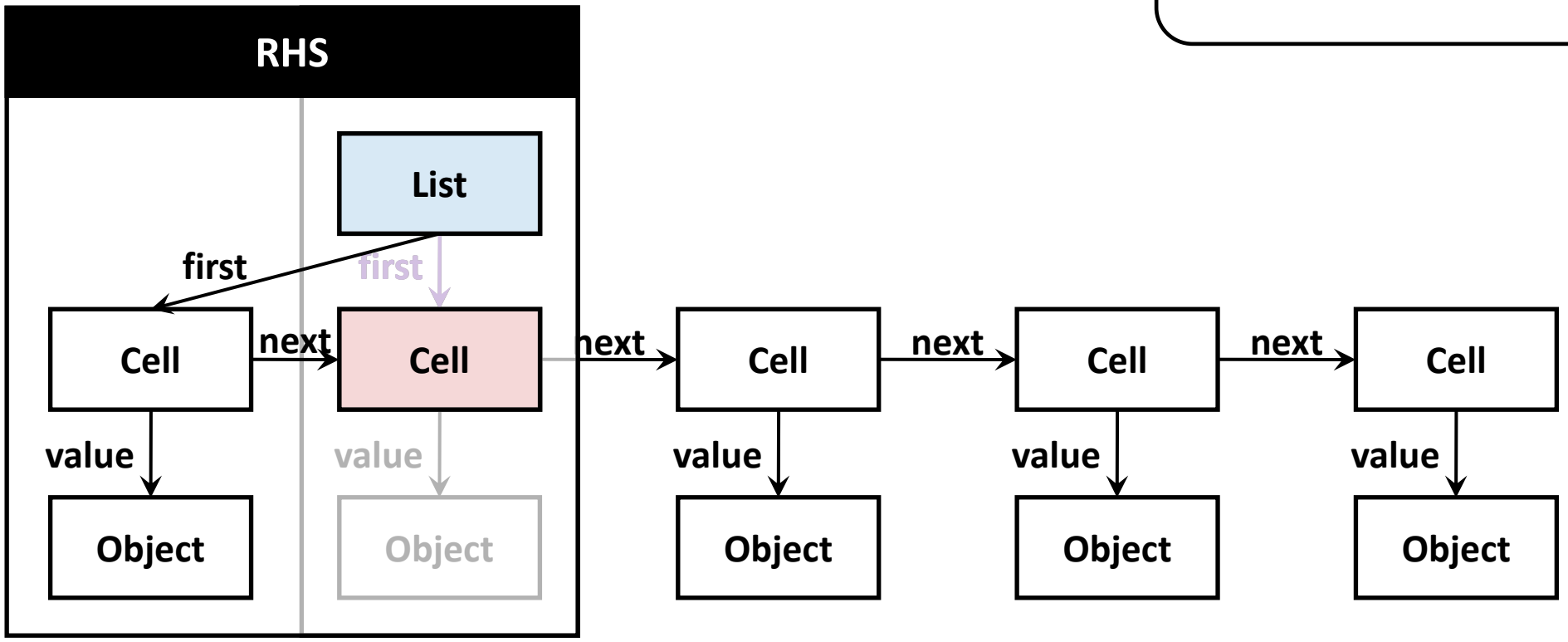
- **Illesztés:** megkeressük a LHS-t tartalmazó részgráfokat a forrás gráfban



Gráftranszformáció: Átírás végrehajtása

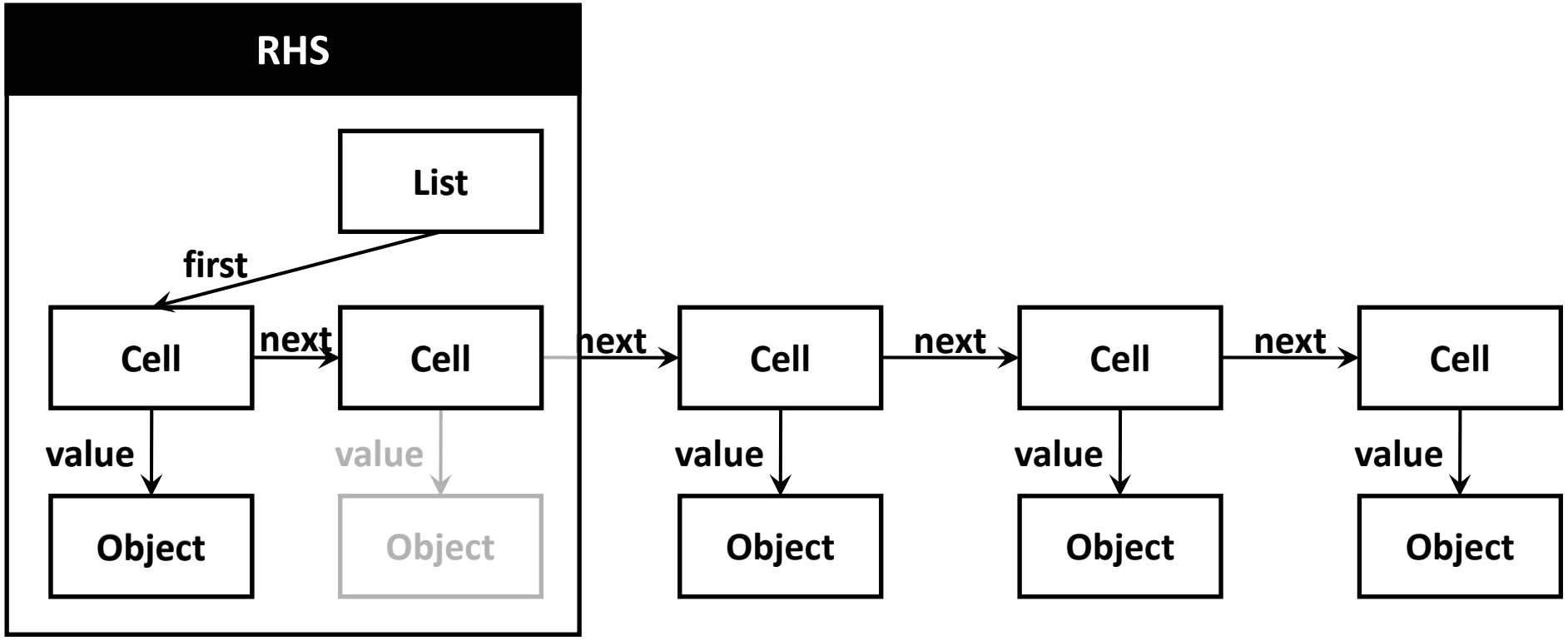
- Illesztés mentén lecseréljük az LHS-t RHS-re.

LHS\RHS → Töröl
RHS\LHS → Beszúr
RHS∩LHS → Békénhagy



Gráftranszformáció: Átírás végrehajtása

- Új gráfot kapunk



Gráftranszformációk értelmezése

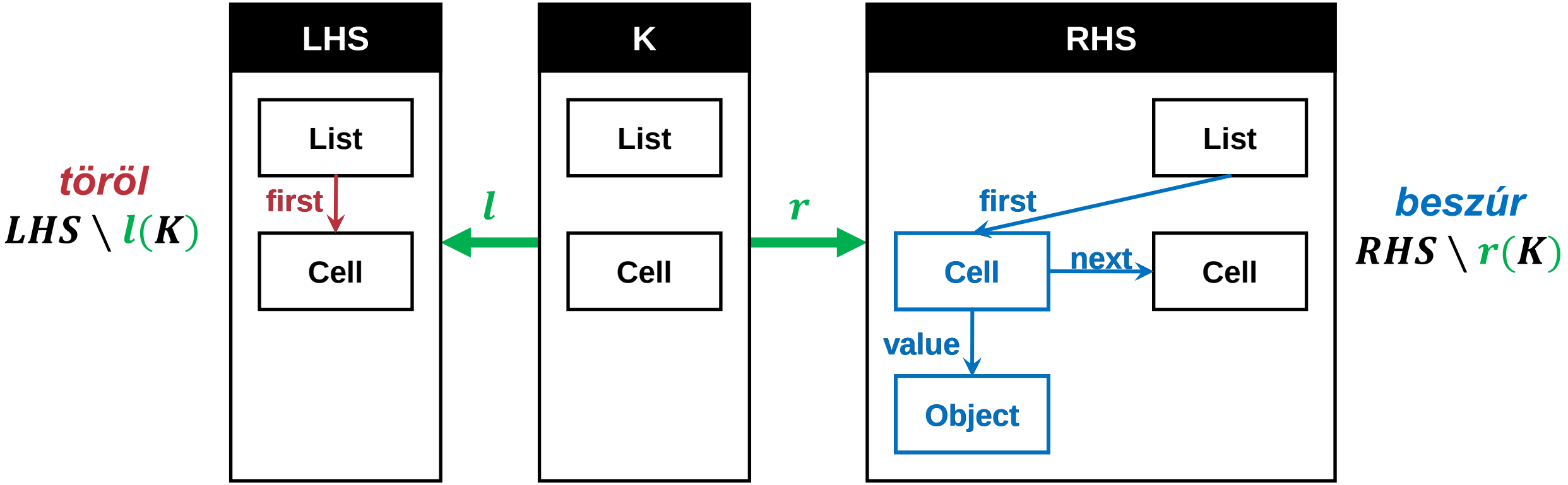
Kétféle értelmezést szoktak használni:

- Operacionalizált
  - > Modellelemek törlése és beszúrása
  - > Könnyen végrehajtható és implementálható
- Matematikai viszonyok vizsgálata
  - > Leírja hogy a dokumentumoknak milyen viszonyokban kell lenniük egymással
  - > Hasznos a bizonyításokhoz

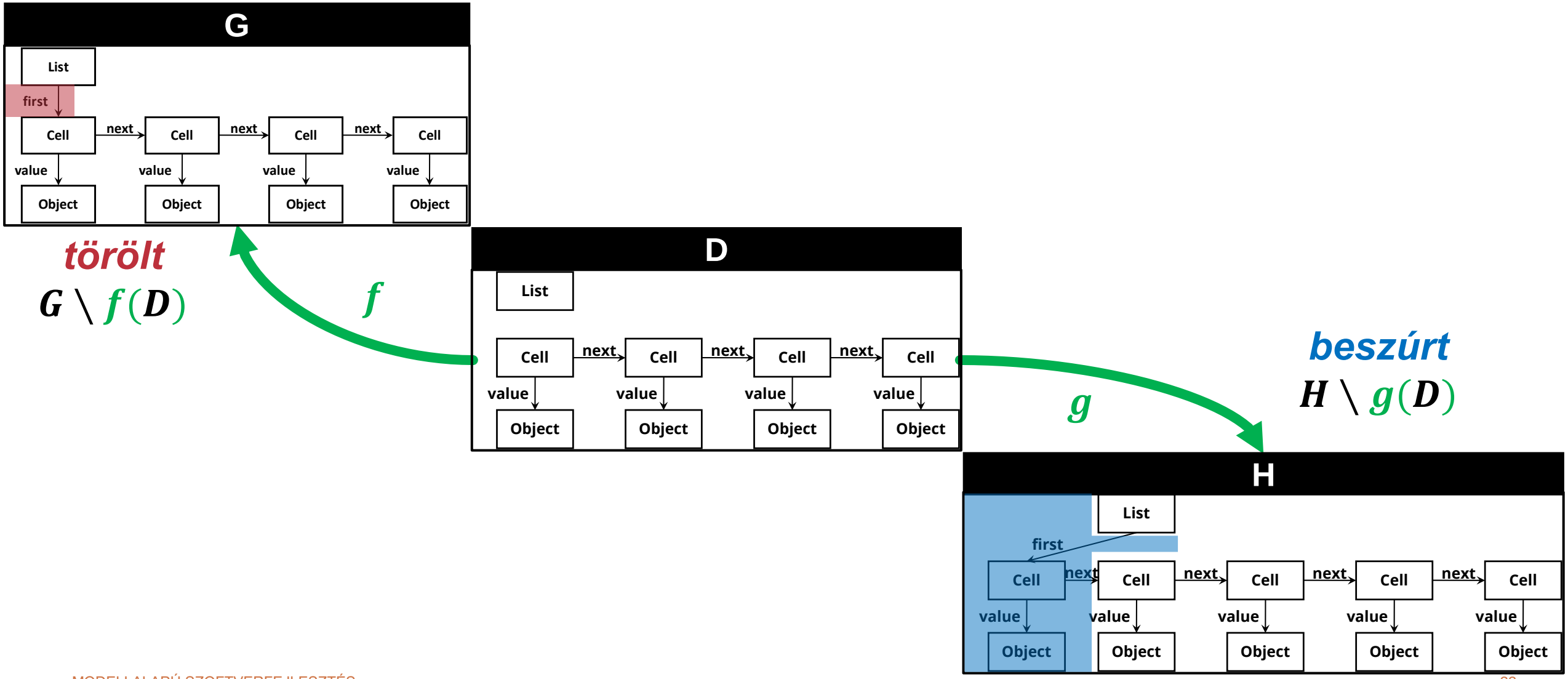
→ Matematikailag precíz és hatékonyan végrehajtható formalizmus

Gráftranszformációk anatómiája

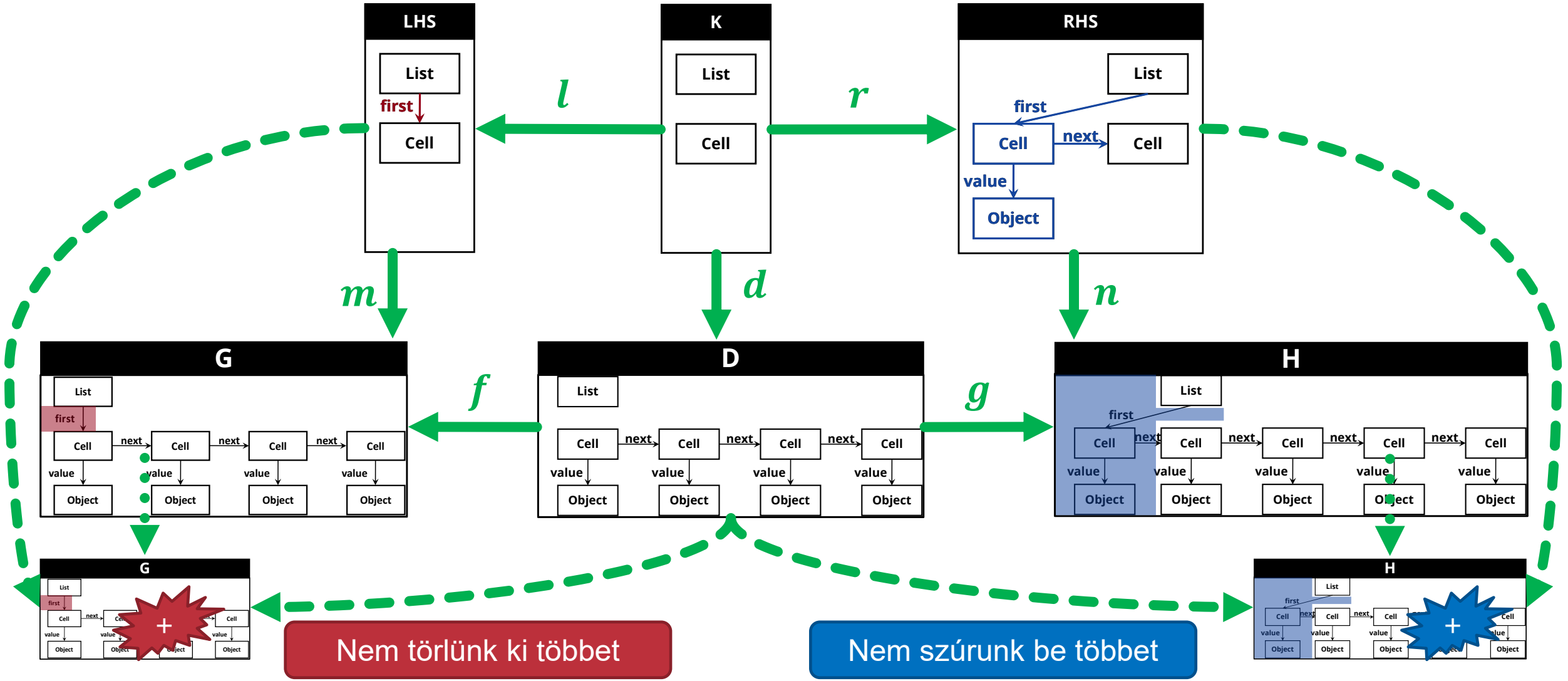
- Vizsgáljuk meg, hogy a transzformáció során mely gráf illeszthető melyikbe!



Gráftranszformációk anatómiája



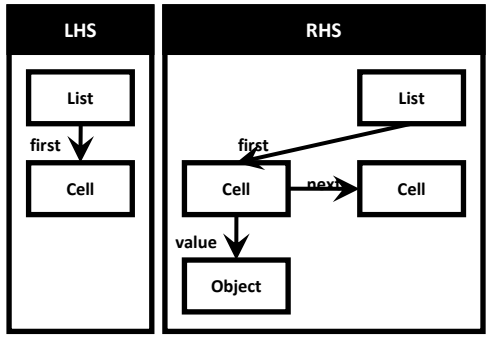
Teljes anatómia



Gráftranszformáció

- Szabályok megfogalmazása modellek átírására
- Nyelvtani szabályok kiterjesztése

List → List,Cell **vs**



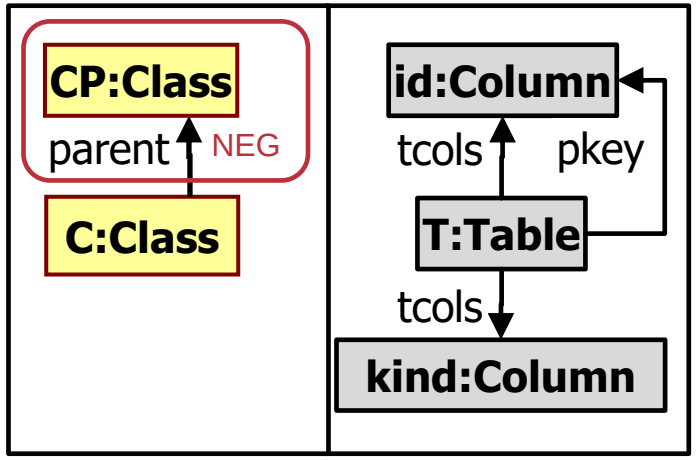
- Szemléletes, de matematikailag precíz formalizmus

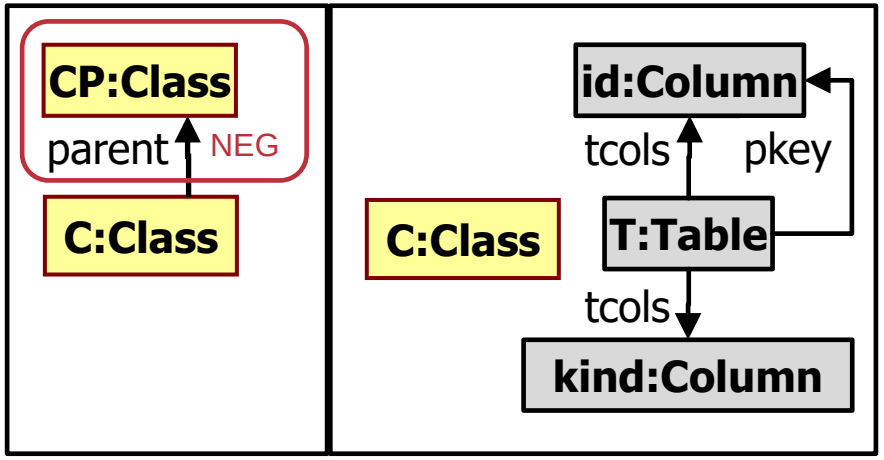
  (Terminálódás, Sorrendezés, Konfluencia, …)

- Eszköztámogatás (lásd előző gyakorlat)

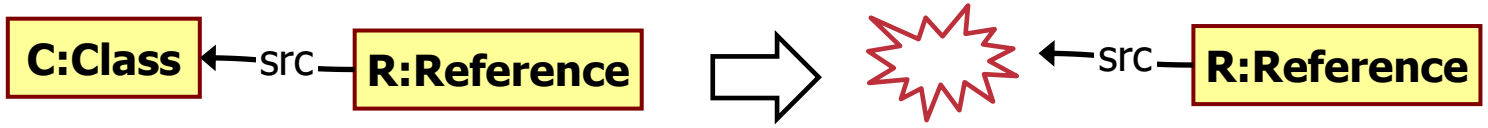
Példák

- Ősosztályok leképezése (törléssel és törlés nélkül)





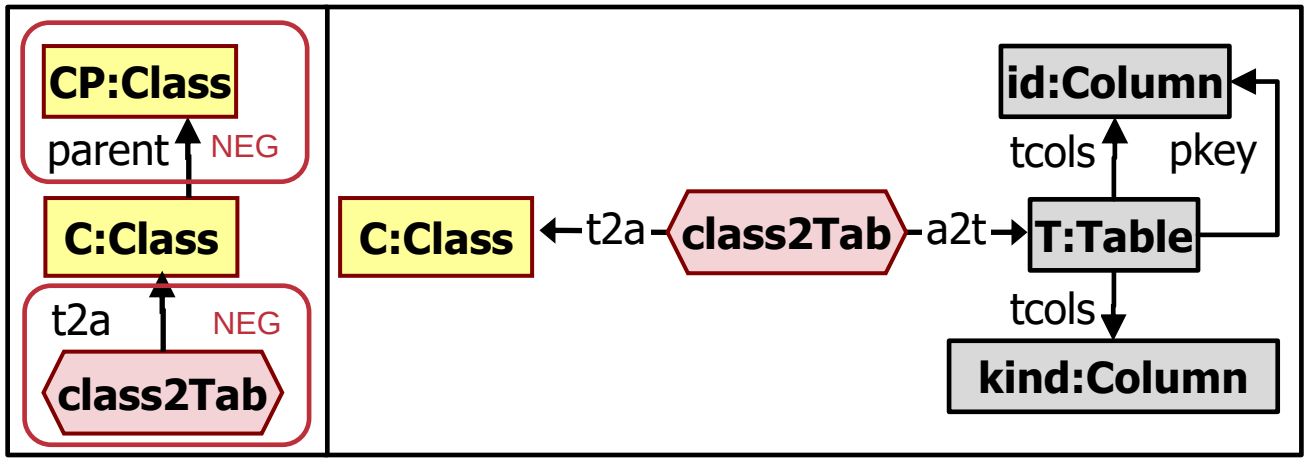
- Mi történik, ha kitörlünk egy elemet, amire mutat még él?



- „Lógó élek” megoldása: Töröljük az éleket / Visszavonjuk a transzformációt

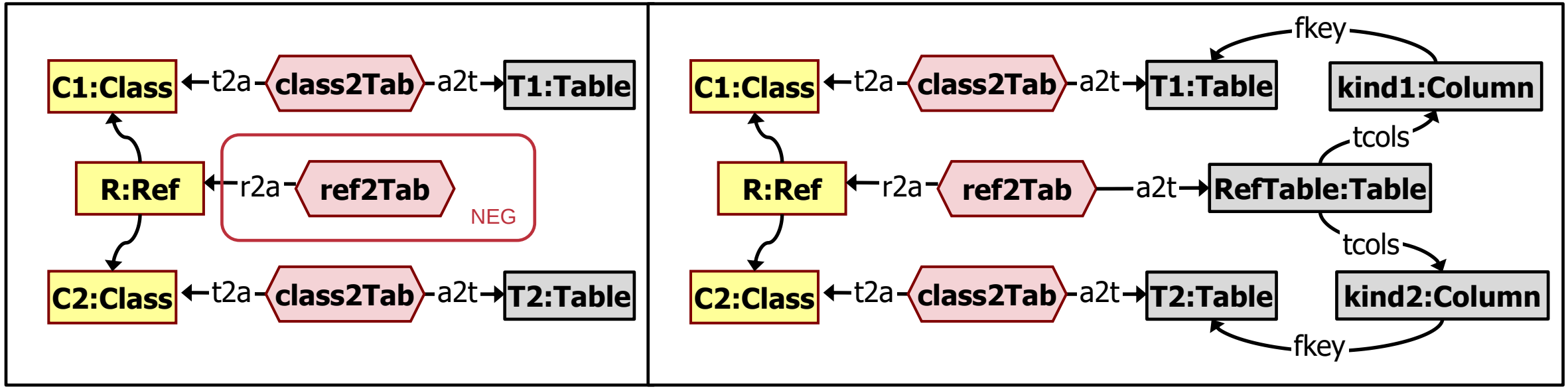
Példák

- Ősosztályok leképezése nyomonkövethetőséggel:
  - *Keressünk olyan ősosztályt,*
  - *amely még nem lett leképezve,*
  - *majd képezzük le.*



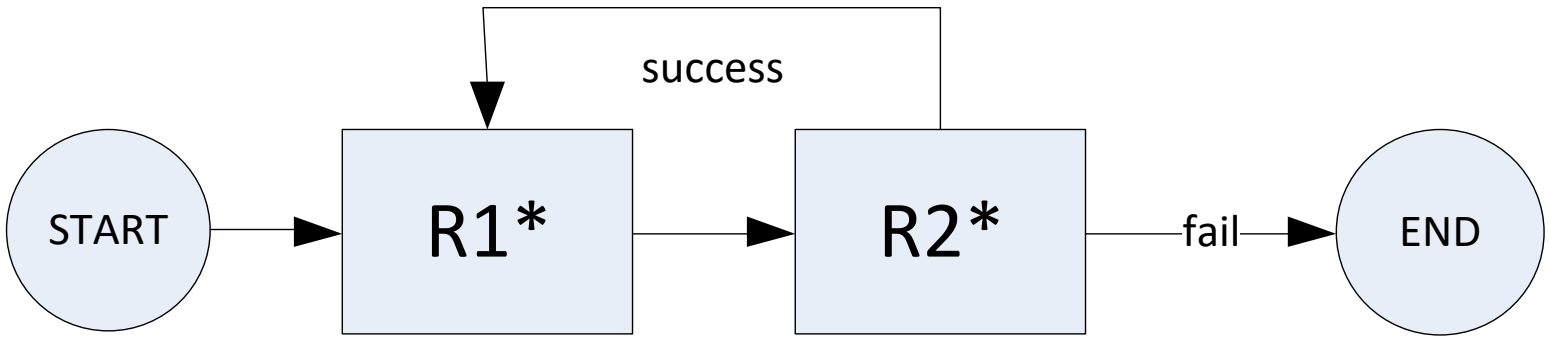
Példák

- Referenciák leképezése

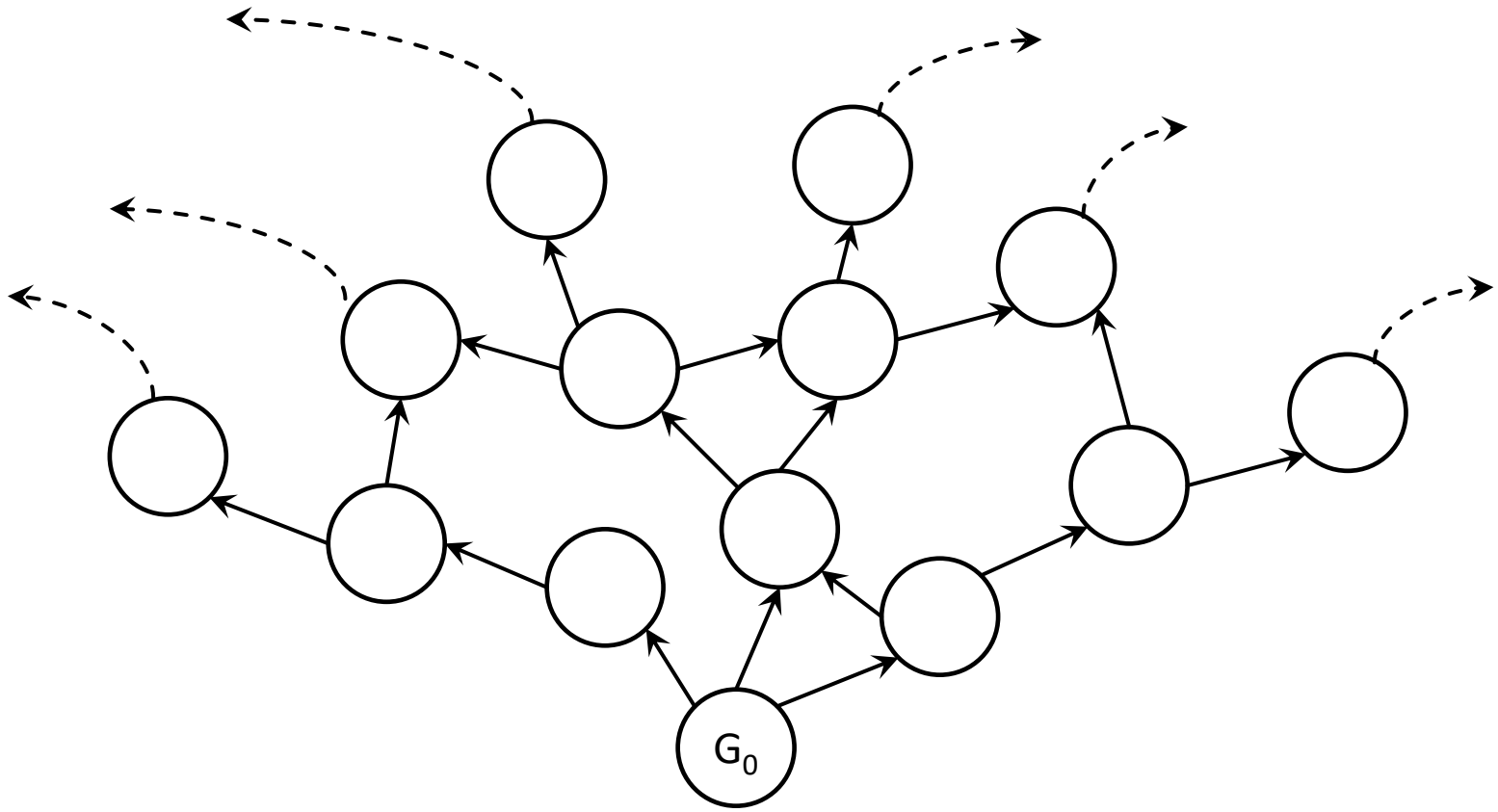


Vezérlési szerkezetek

- Milyen sorrendben hajtsuk végre a szabályokat?
- Több lehetőség, lásd előző előadás.
- De például:
  - > Tüzelj szabadon választott transzformációkat, amíg ez lehetséges (~ alapértelmezett)
  - > Tüzeld el az összes szabad transzformációt egyszer
  - > vezérlési gráf (explicit vezérlés)

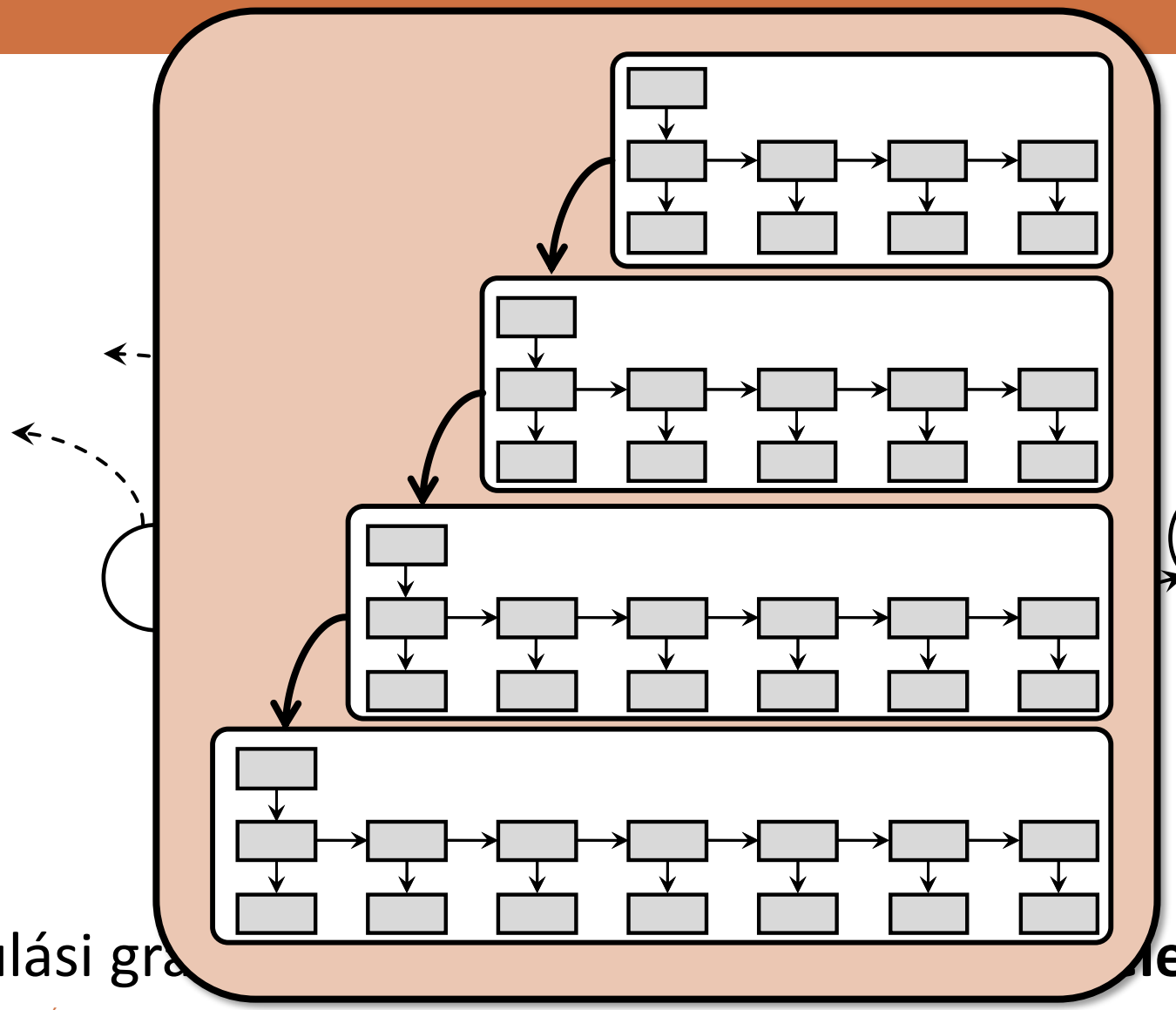


Állapottér



Kiindulási gráf **+** GT szabályok → **(tipikusan végtelen)** Állapottér

Állapottér



Még egyszerű példa esetén is végtelen

Olyan rendszereket jellemezhetünk, amelyek:
- Nemdeterminisztikusak
- Végtelen állapottérrel rendelkeznek

Kiindulási gra **len)** Állapottér

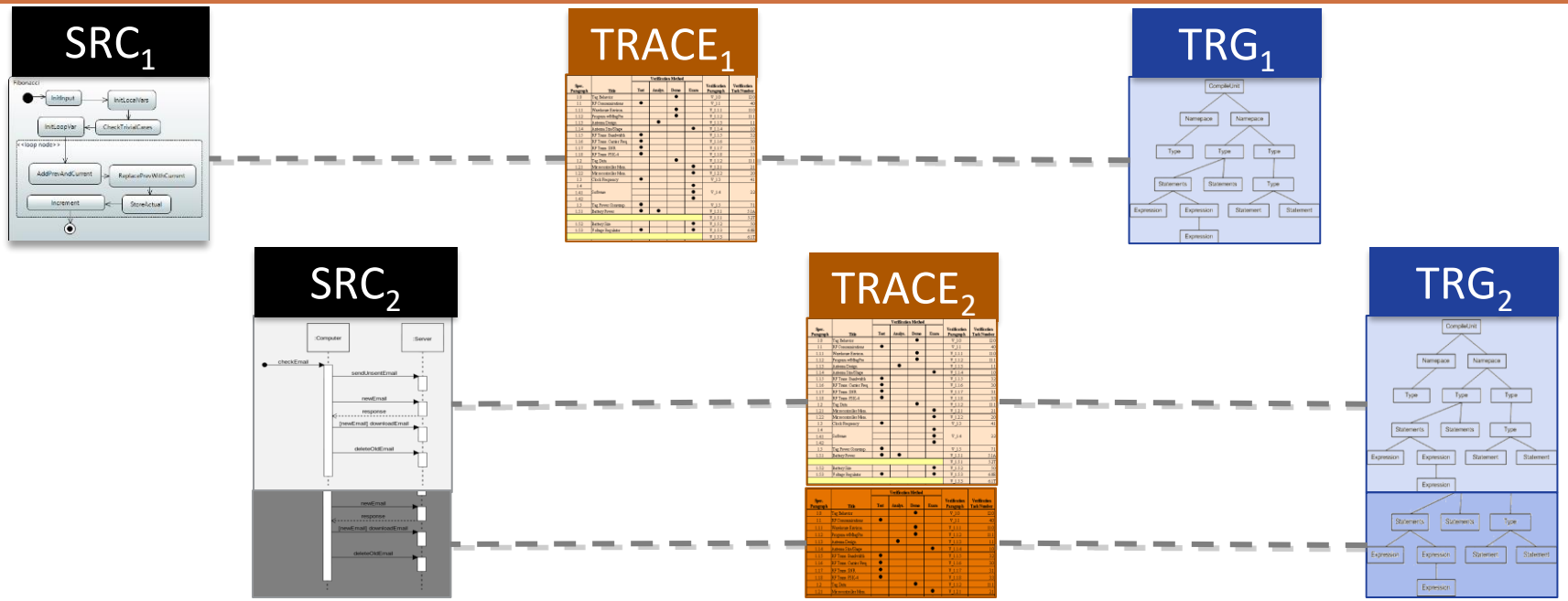
Modeltranszformációk típusai

- A bemenetek és kimentek száma
  - In-place: ugyanaz a bemenet és a kimenet, azaz a modellt felülírjuk (pl szimuláció léptetése, quick fix)
  - Out-place: A kimenet egy másik modell (pl: ORM esetén osztálydiagram → Tábla)
- A nyelv szerint
  - Endogén: Ugyanaz a metamodell (pl control-flow egyszerűsítése, eredeti megtartása)
  - Exogén: Különböző a metamodell
- Az irány szerint
  - Egyirányú: Akkor van értelme, ha egy forrás modellből egy célmodellt készítünk (ORM példa)
  - Kétirányú: Mindkét irányból végrehajtható (Pl ha adatbázis sémából is tudnánk osztályokat készíteni)

Gráfmintaillesztés, Gráftranszformáció

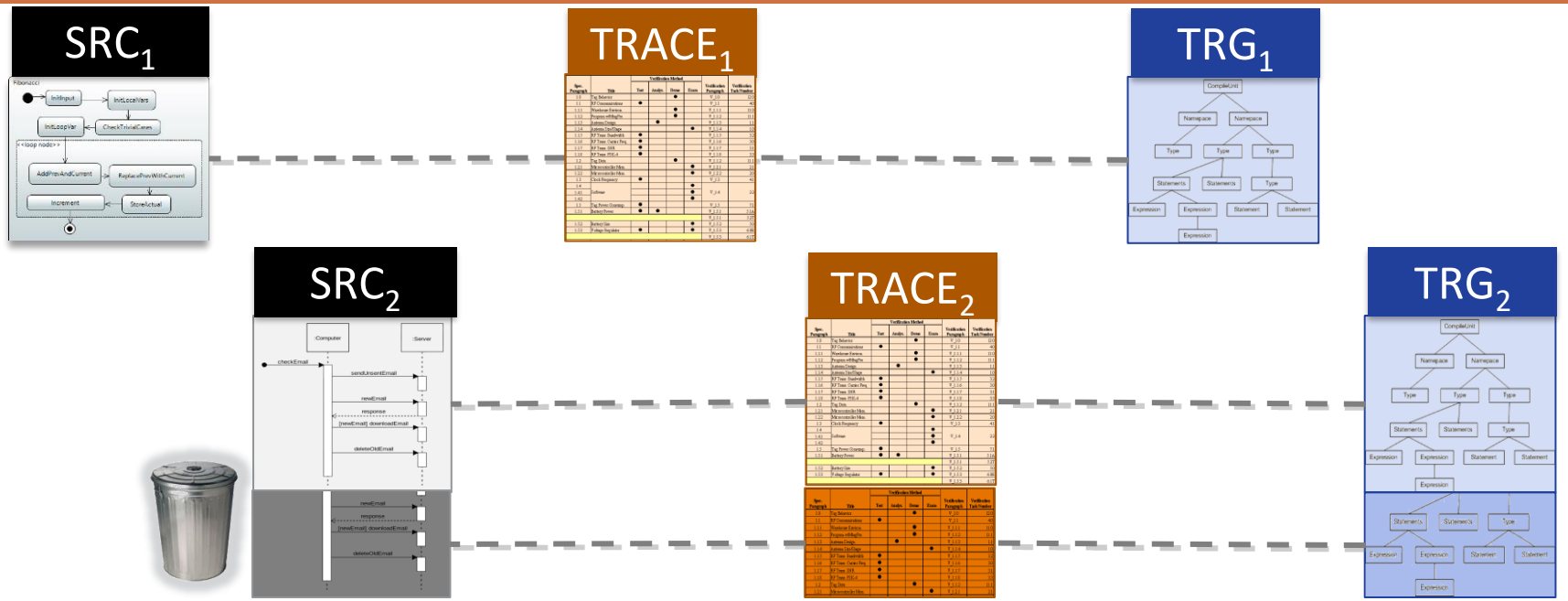
- Alapfogalmak
- Gráfmintaillesztés
- Modelltranszformációk
- **Inkrementális transzformációk**
- Tervezésitér bejárás

Inkrementális végrehajtás: Batch transzformáció



1. Első transzformáció
2. Forrásmodell megváltozik
3. Újrafuttajuk az elejétől az összes forrásmodellre

Piszkos Inkrementalitás



Előnyök:
- Nagy lépésenkénti inkrementalitás
- Kerüli a folyamatos végrehajtást

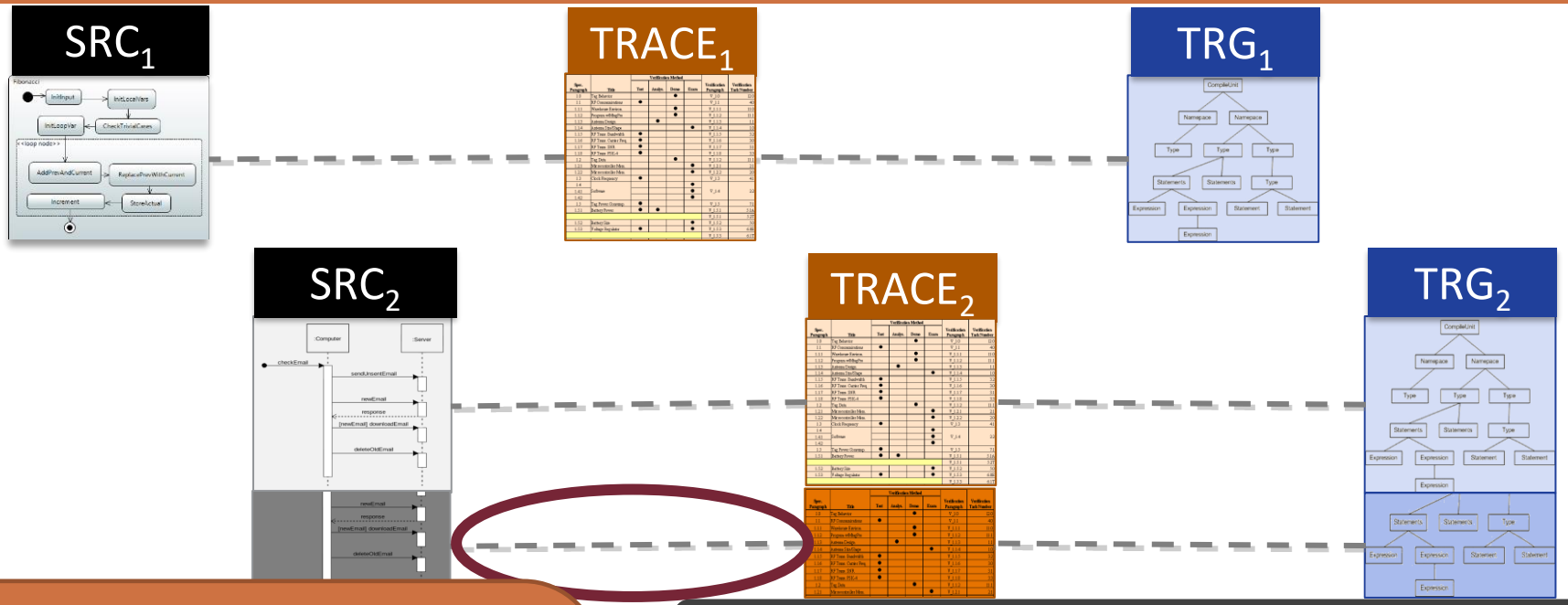
Hátrányok:
- Összetett MT lassú lehet
- Tisztítás (hiba után)?
- Láncolás?

1. Első transzformáció

2. Forrásmodell megváltozik

3. Újrafuttajuk az elejétől csak a módosított modellekre

Inkrementalitás nyomonkövethetőséggel

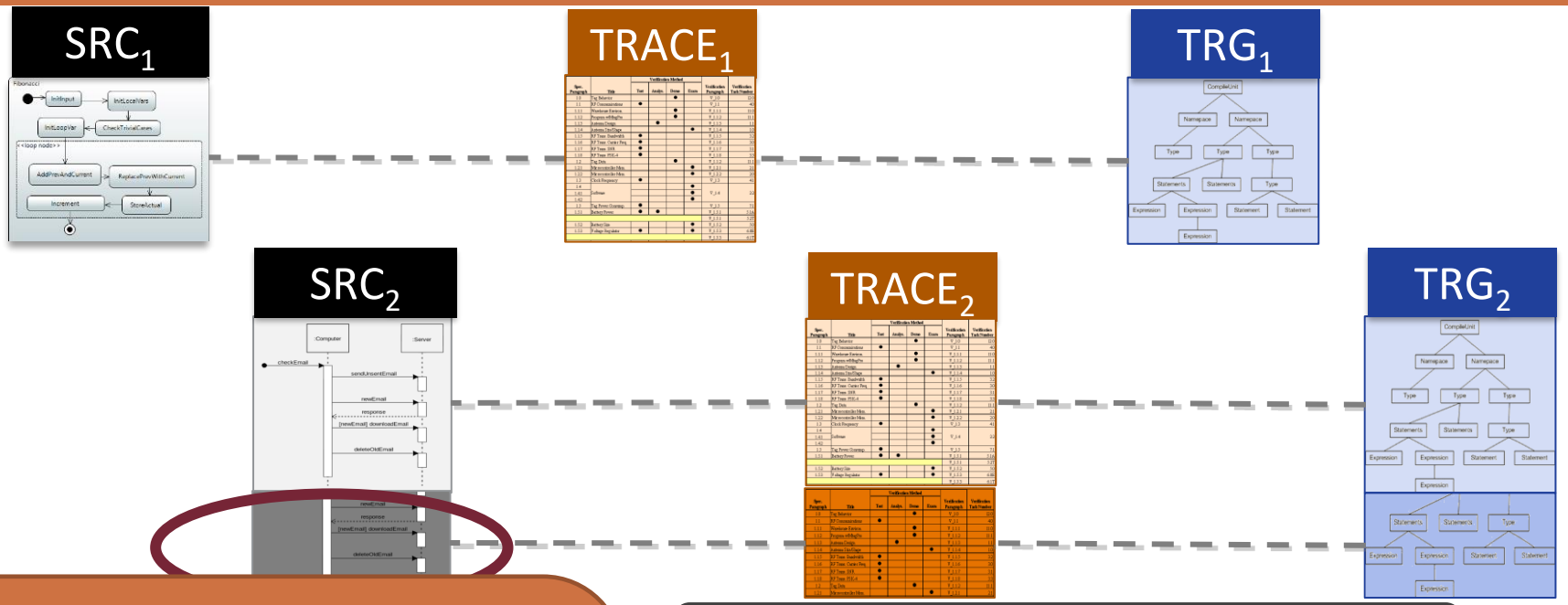


Előnyök:
- Kis lépésenkénti inkrementalitás
- Jobb teljesítmény

Hátrányok:
- Nagymértékben függ a nyomonkövethetőségi kapcsolatoktól
- Intelligens illesztőre van szükség

1. Első transzformáció
2. Forrásmodell megváltozik
3. Hiányzó nyomkövetési kapcsolatok felderítése
4. MT újbóli végrehajtása csak a nem nyomon követhető elemek esetében

Eseményvezérelt Transzformációk



Előnyök:
- Finomított kontextus: a lekérdezés eredményének változásai vezérlik
- Láncolás
- Elkerüli a folyamatos számolást

Hátrányok:
- Nyelvi szintű korlátozások
- Élőben kell "figyelni"

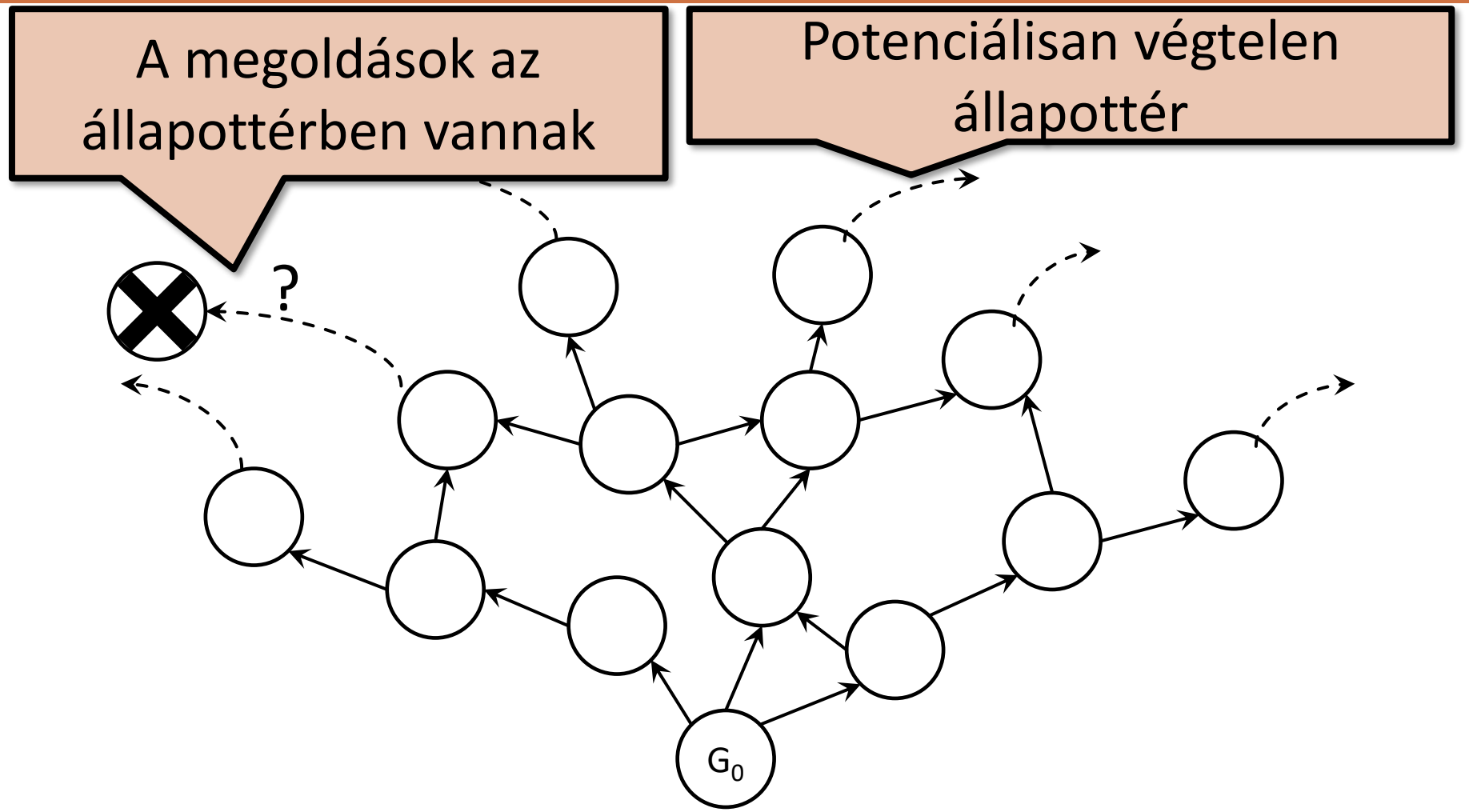
1. Első transzformáció
2. Forrásmodell megváltozik
3. Change notification feldolgozása
4. Változás terjesztése

VIATRA

Gráfmintaillesztés, Gráftranszformáció

- Alapfogalmak
- Gráfmintaillesztés
- Modelltranszformációk
- Inkrementális transzformációk
- Tervezésitér bejárás

Visszatekintés: GT rendszer állapottere



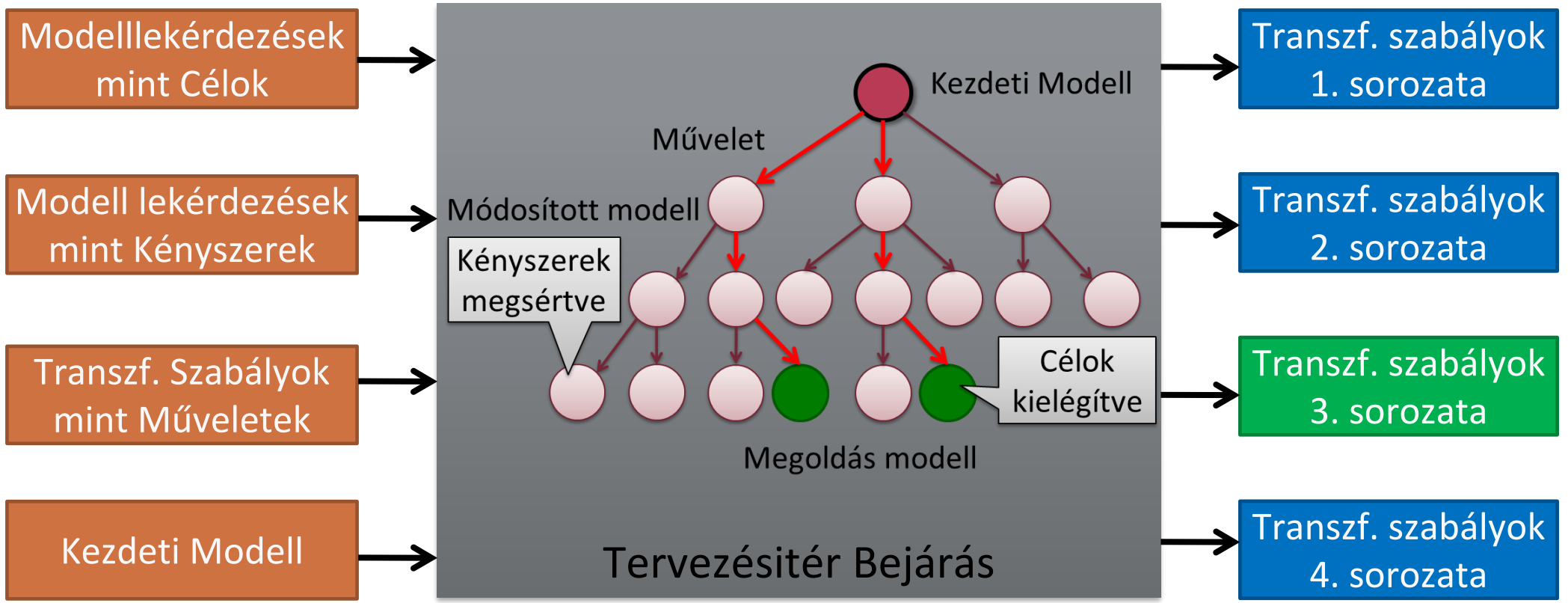
Kiindulási gráf + GT szabályok → Állapottér

Tervezésitér bejárás

Speciális állapottér bejárás

- potenciálisan végtelen állapottér
- „sűrű” megoldási tér

Modellvezérelt Irányított Tervezésitér Bejárás

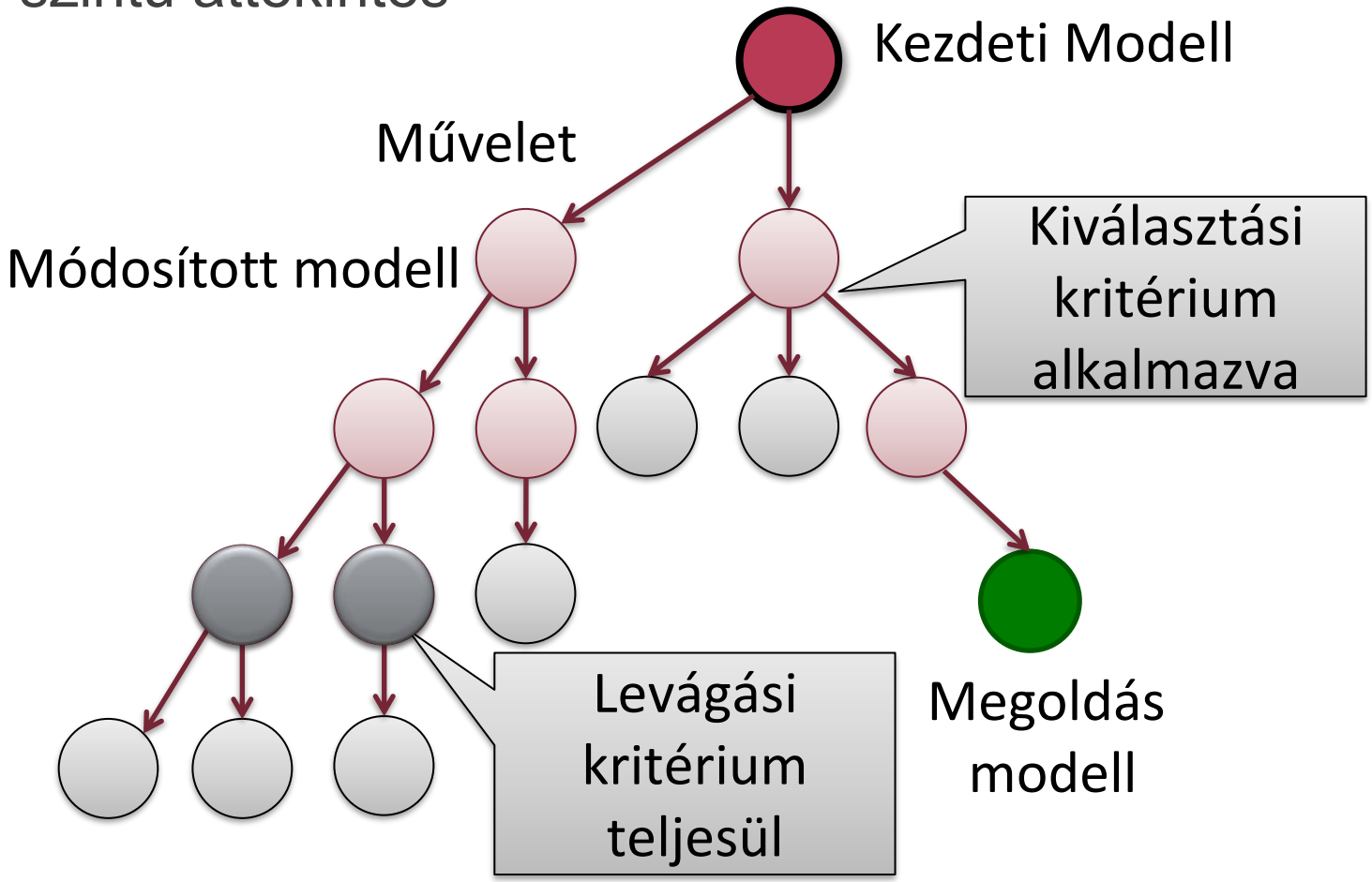


Útmutatás a bejáráshoz: Tippek

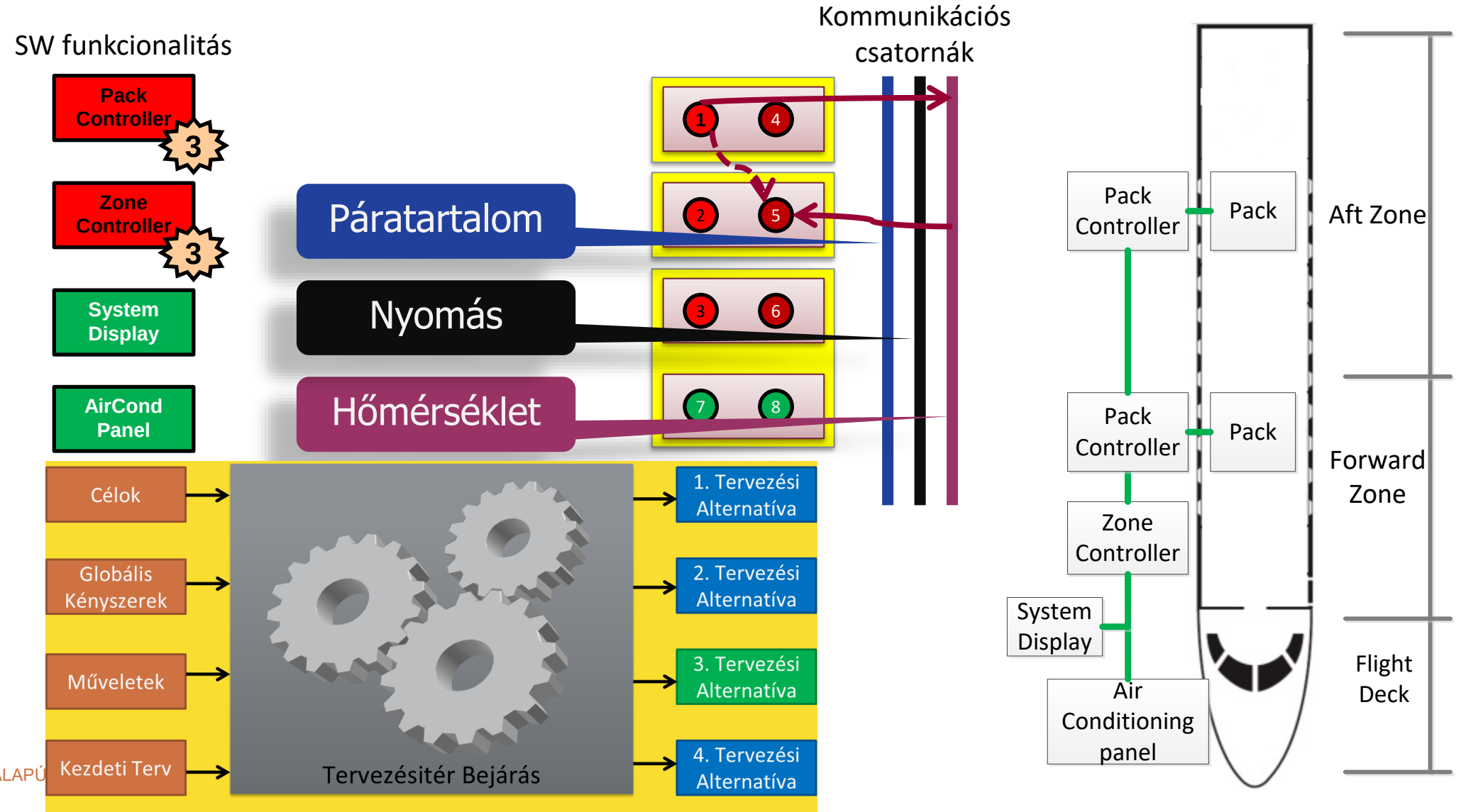
- tervező / végfelhasználó
- formális elemzés

Irányított Tervezésitér Bejárás

- Magas szintű áttekintés

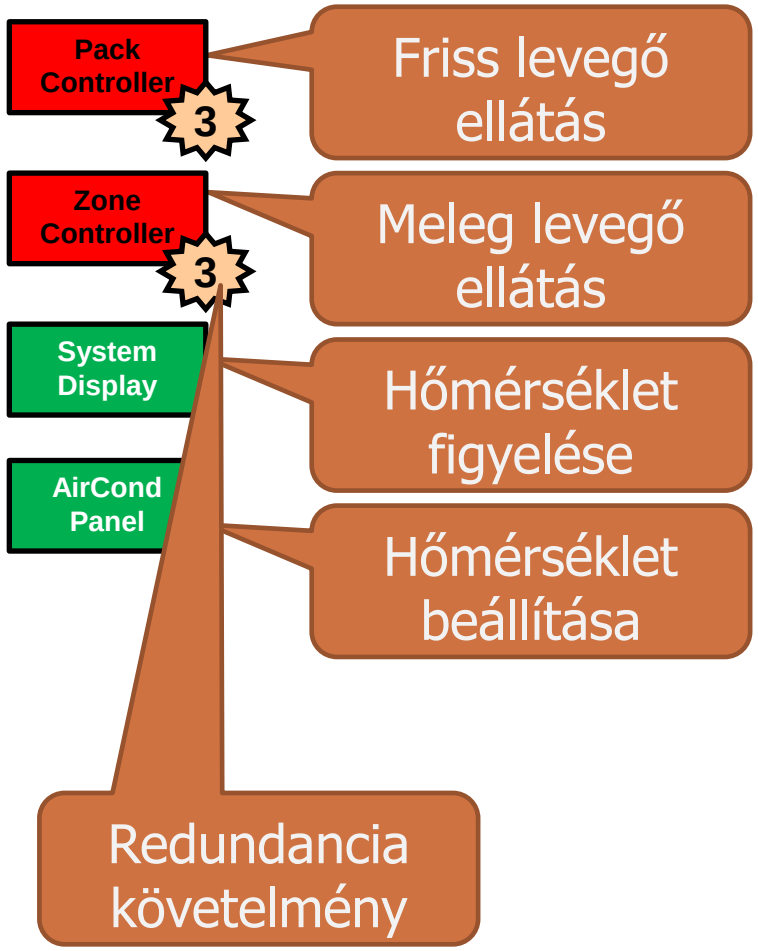


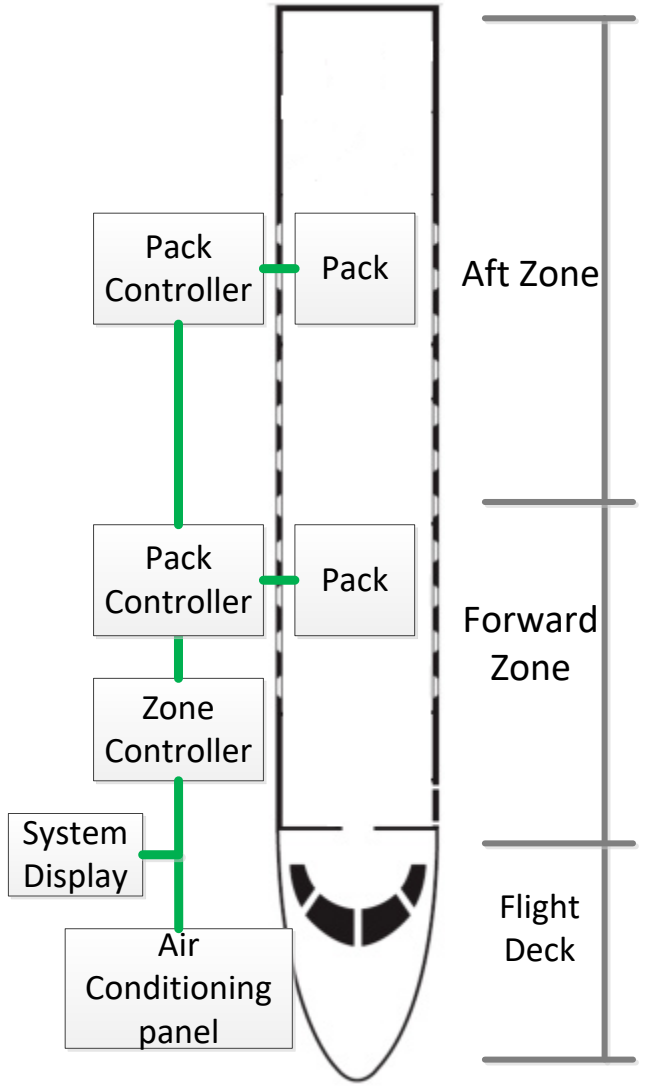
Tervezésitér Bejárás IMA Konfiguráció Tervezéséhez



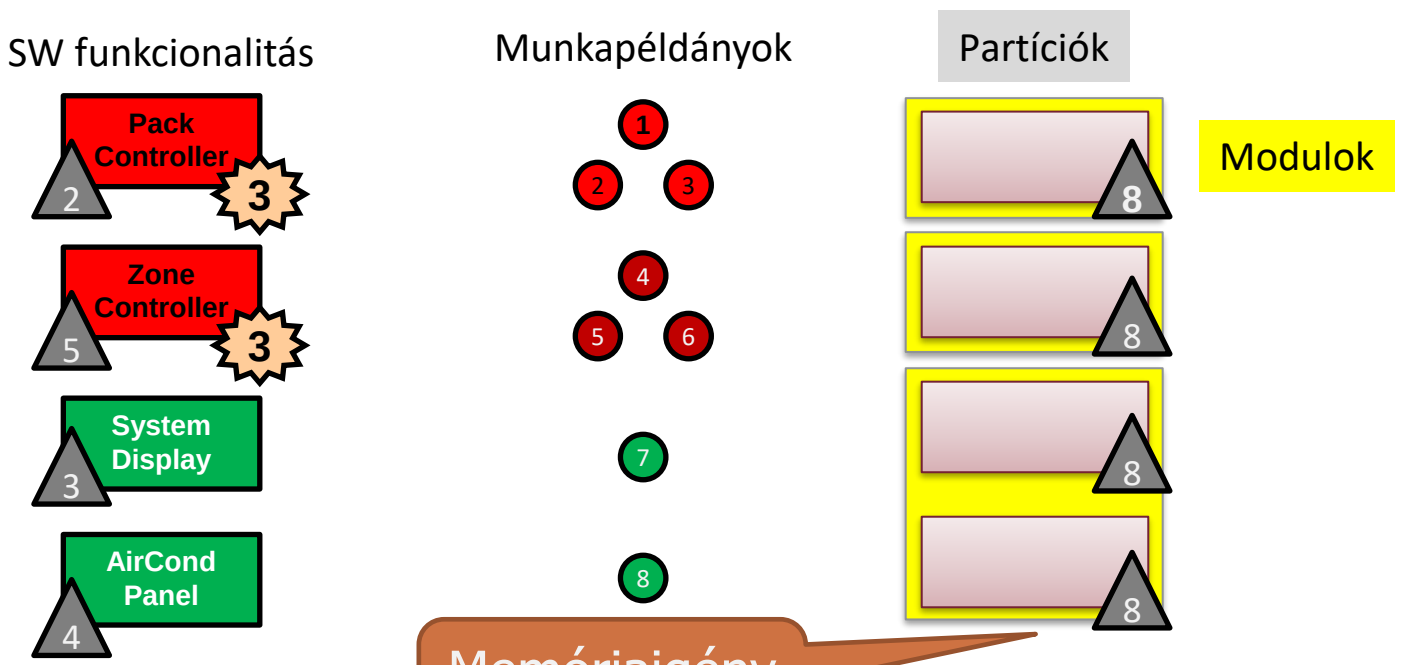
ARINC653 konfigurációk tervezése

SW funkcionalitás
(kritikus + nem kritikus)

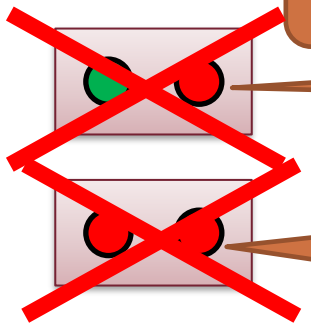




Munkapéldányok, Partíciók, Modulok



Kényszerek

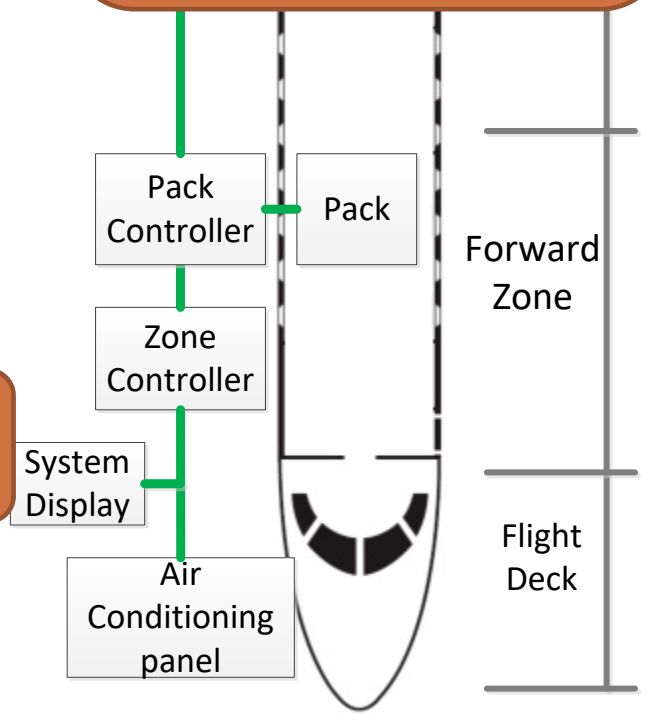


Memóriaigény + kényszerek

Ne keverjen kritikus és nem kritikus feladatokat

Ne keverje ugyanazon kritikus feladat példányait

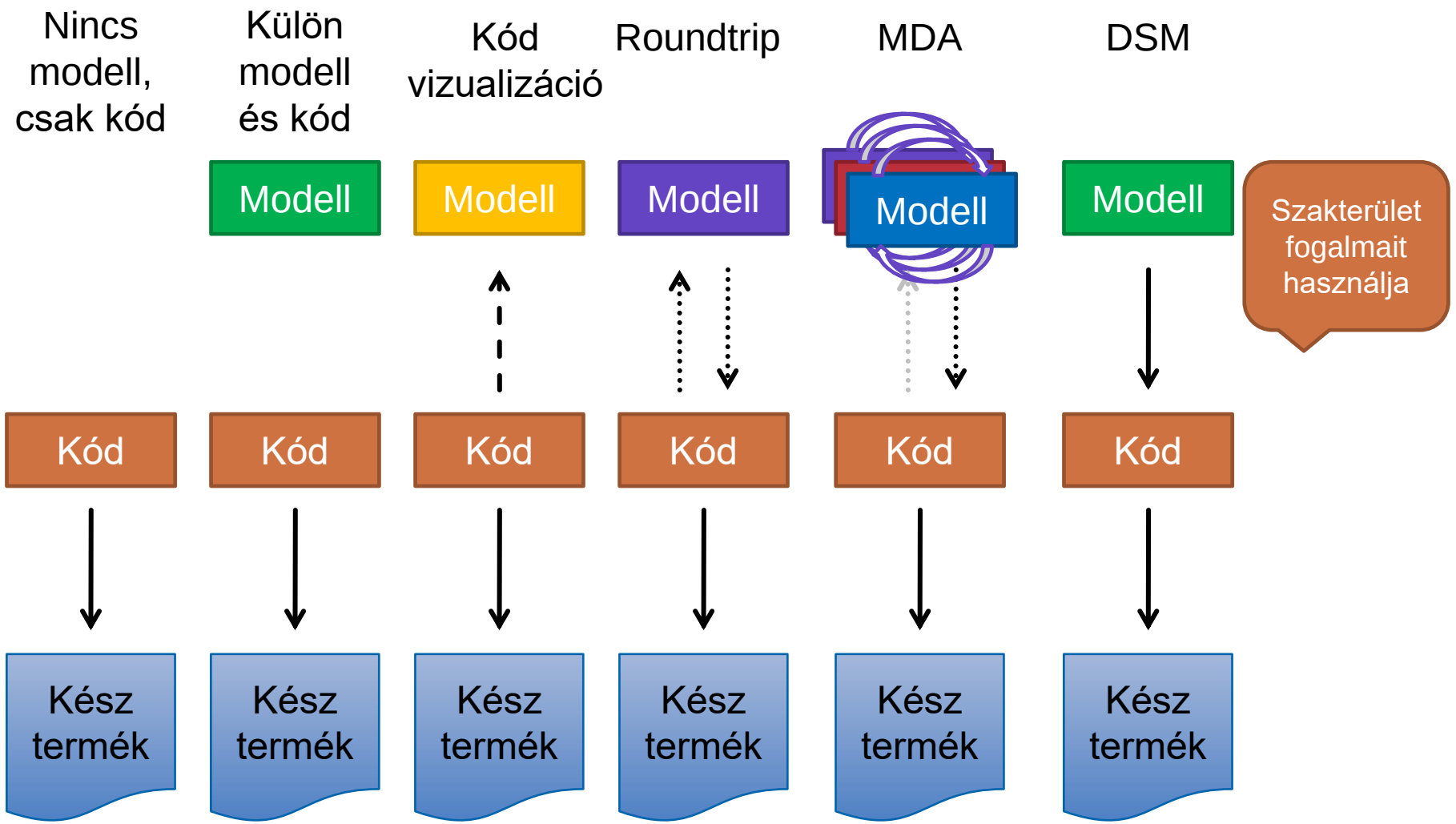
További kényszerek
- WCET,
- ütemezés stb.
- interfészek
- adattípusok



Gráfmintaillesztés, Gráftranszformáció

- **Alapfogalmak**
- **Gráfmintaillesztés**
- **Modelltranszformációk**
- **Inkrementális transzformációk**
- **Tervezésitér bejárás**

Hogyan használunk modelleket?



Köszönöm a figyelmet!