Modellalapú szoftverfejlesztés

VI. gyakorlat

Gráfmintaillesztés, Modelltranszformáció

Ficsor Attila

Gráfmintaillesztés, Modelltranszformáció

I. Gráftranszformáció

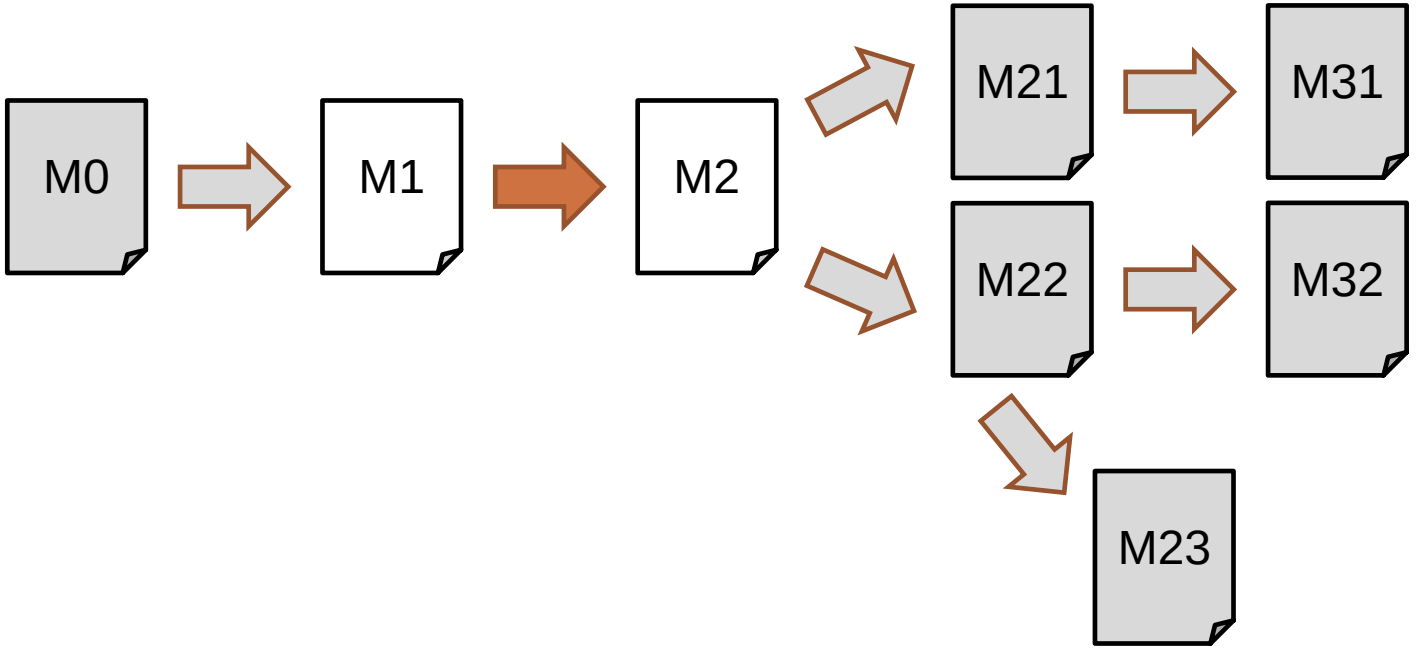
II. Viatra Query Language

III. Viatra Validation Framework

IV. Benchmark

Motiváció: Modellek transzformációja

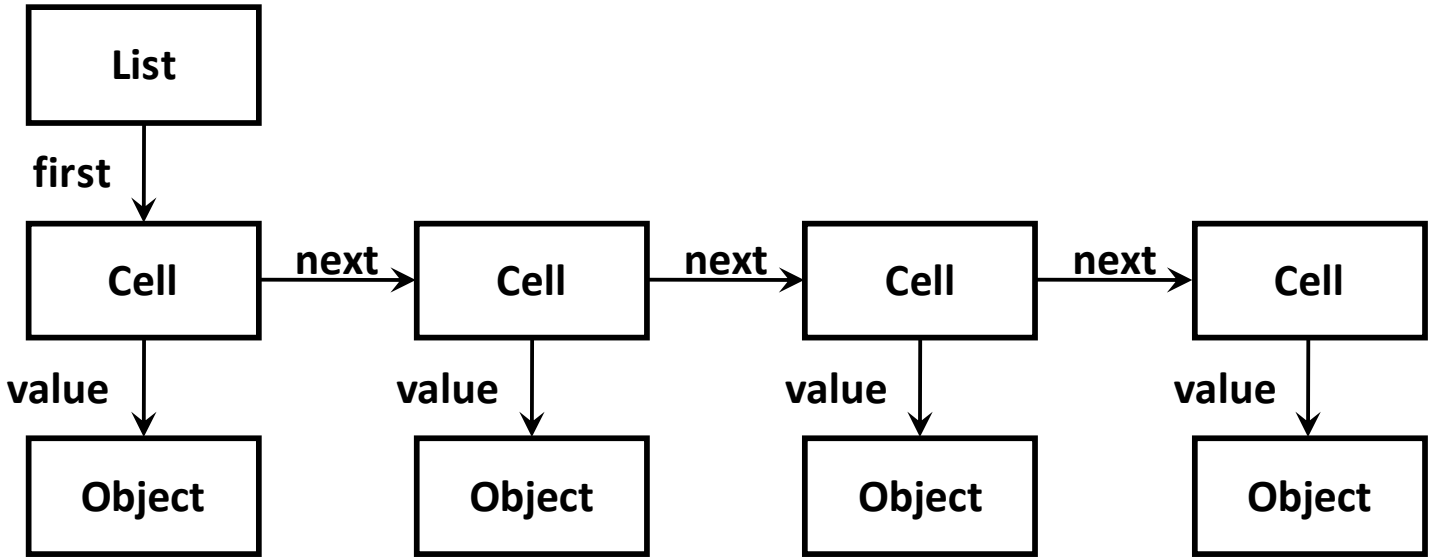
- **Modellalapú fejlesztés:** Modellek az elsődleges dokumentumok
- Modelleket fejlesztünk, automatizáljuk a modellfeldolgozást



- **Cél:** modelltranszformációk hatékony megfogalmazása és végrehajtása

Gráftranszformáció

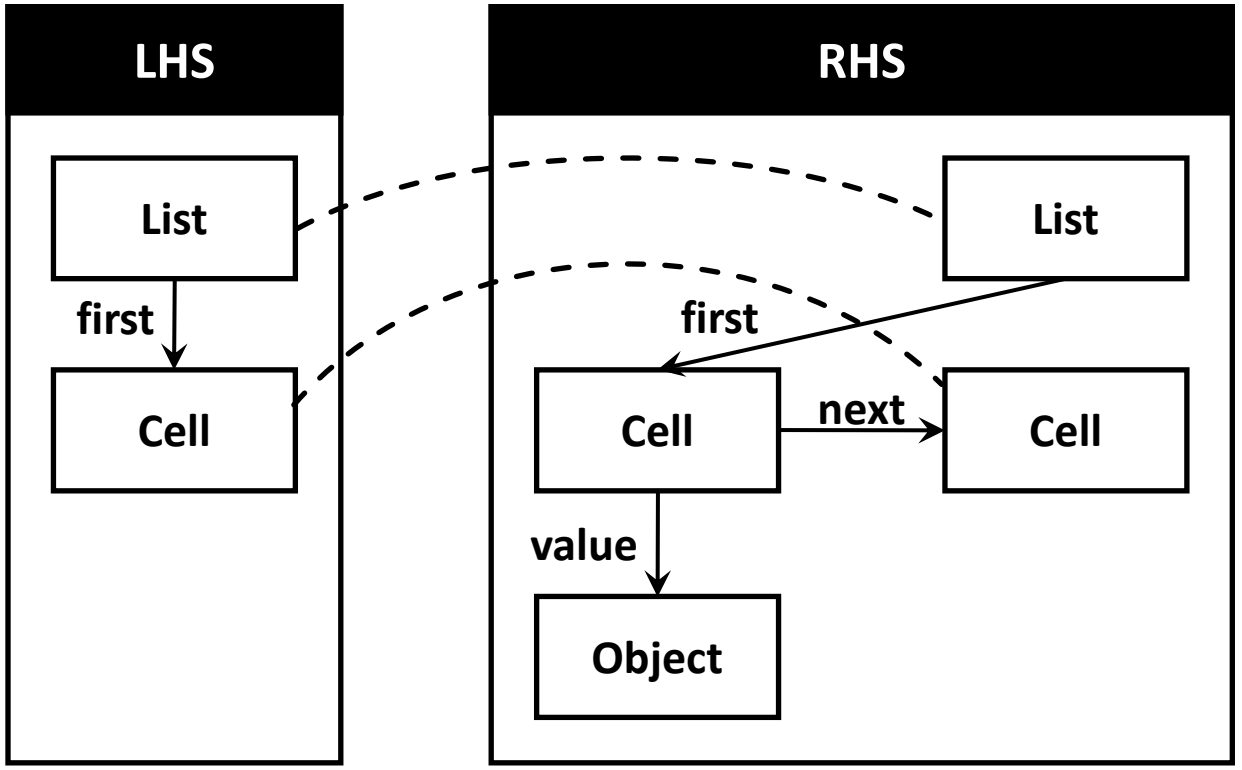
- Modell = Címkézett gráf



Gráftranszformációs szabály

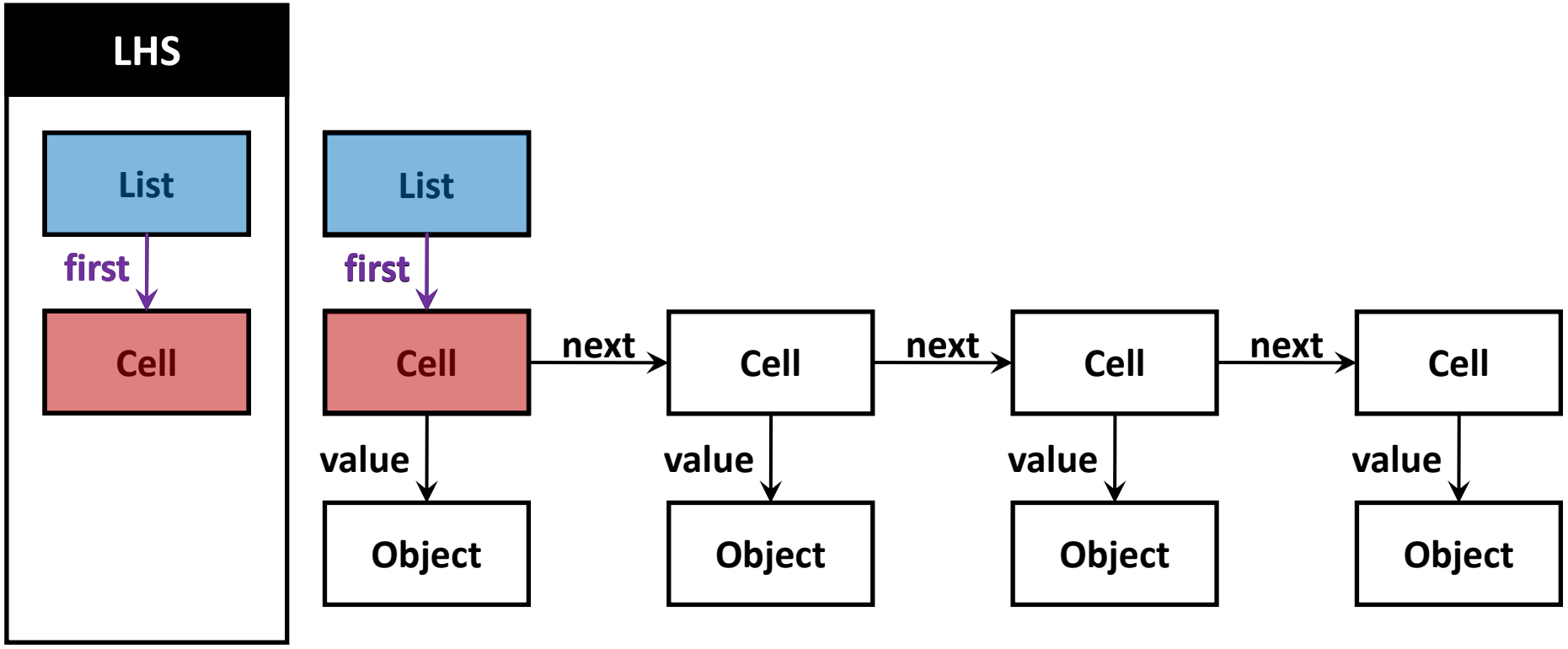
- Gráf átírási szabály, két gráffal van megfogalmazva

LHS = bal oldal, RHS = jobb oldal



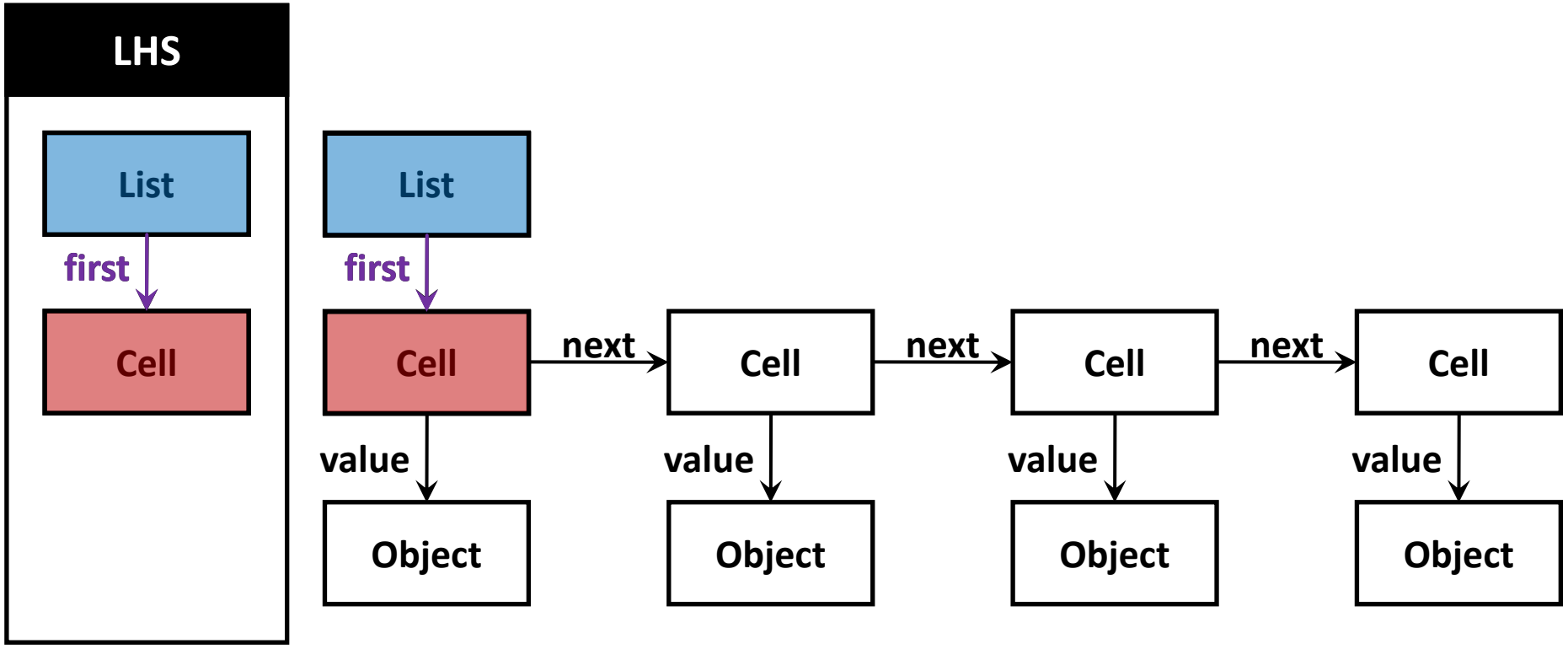
Mintaillesztés

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



Mintaillesztés

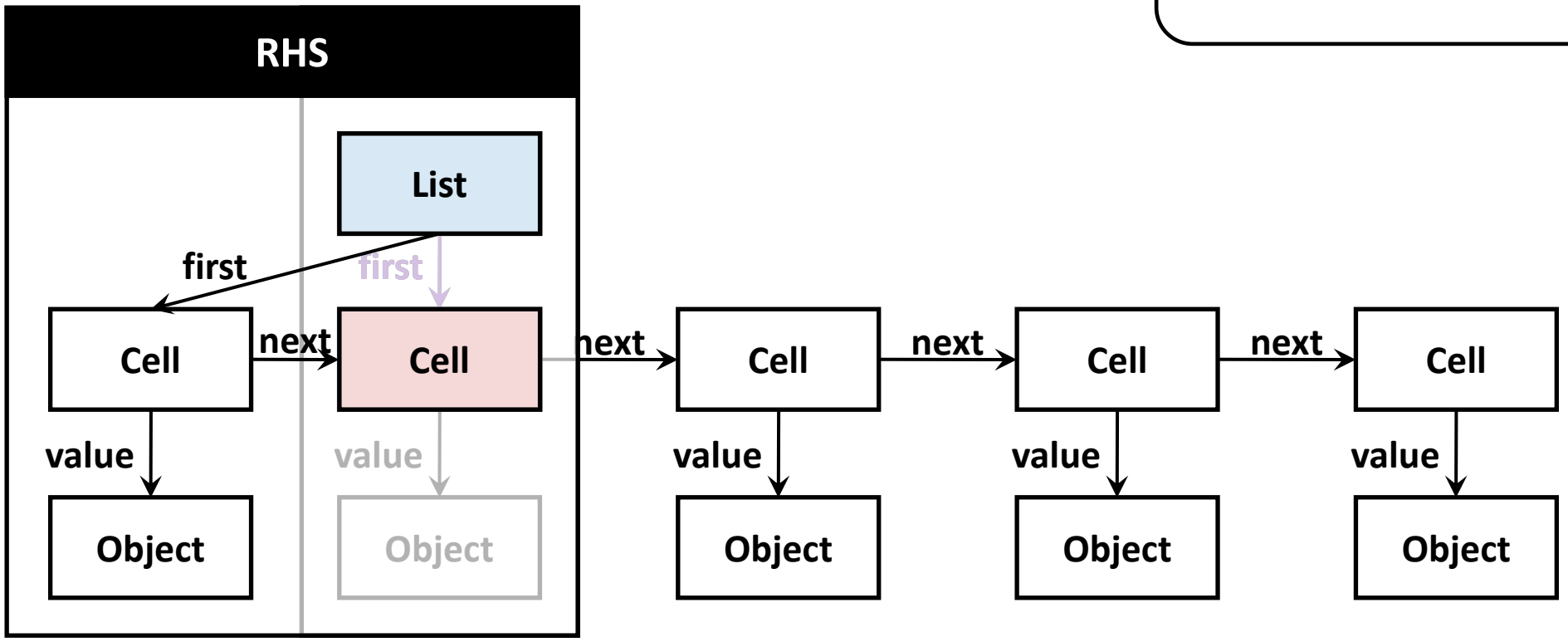
- **Illesztés:** megkeressük a LHS-t tartalmazó részgráfokat a forrás gráfban



Transzformáció végrehajtása

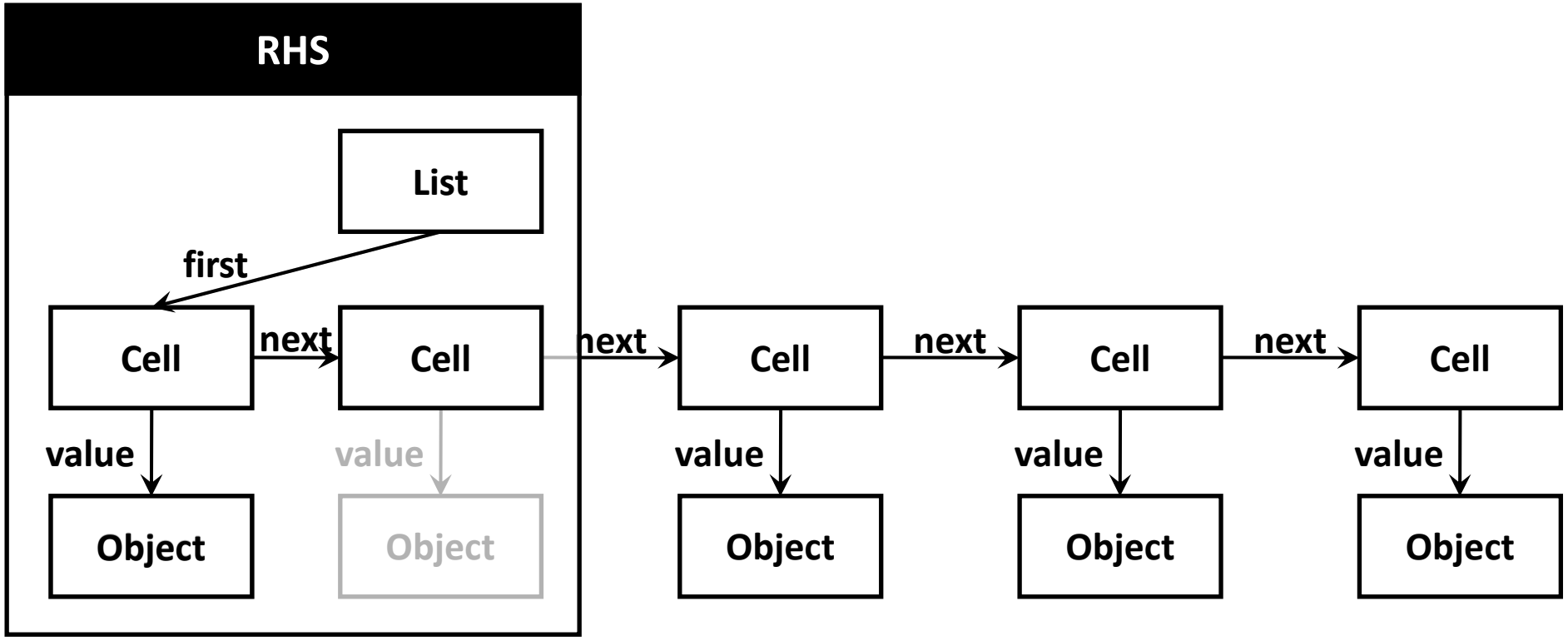
- Illesztés mentén lecseréljük az LHS-t RHS-re.

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



Transzformáció végrehajtása

- Új gráfot kapunk



Gráfmintaillesztés, Modelltranszformáció

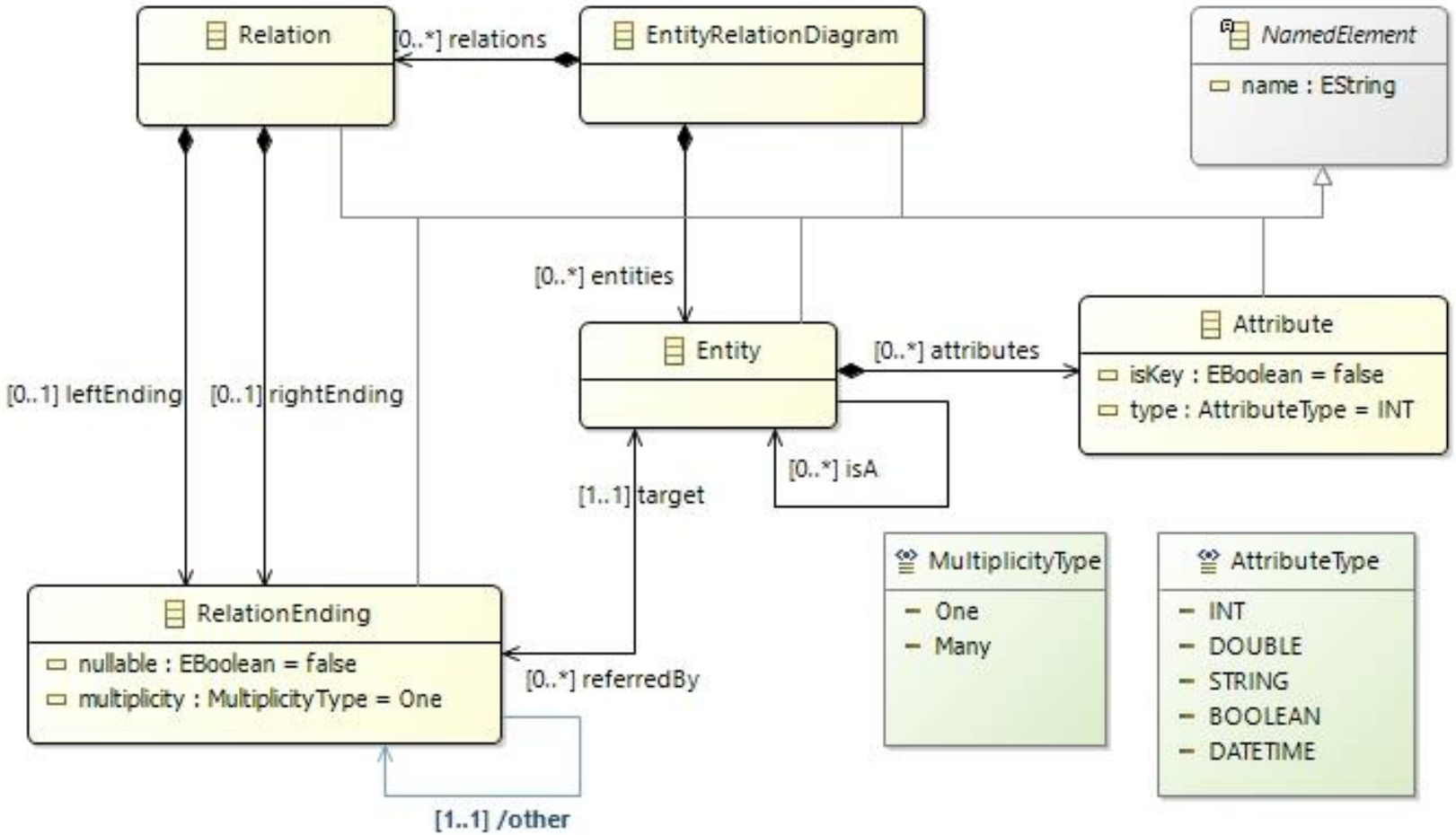
I. Gráftranszformáció

II. Viatra Query Language

III. Viatra Validation Framework

IV. Benchmark

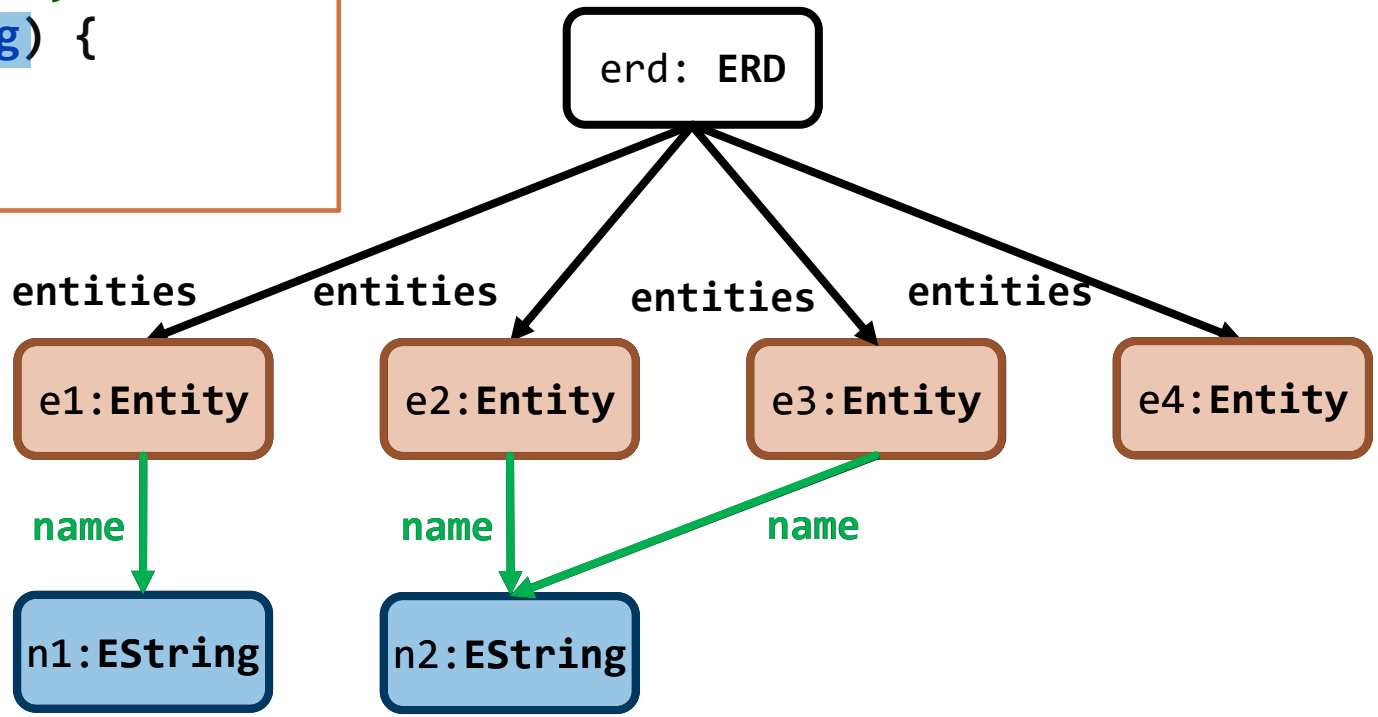
PÉLDA Egyed-Kapcsolat Diagram metamodell



PÉLDA Mintaillesztés

```
// e egy Egyed-Kapcsolat diagram entitása, n névvel
pattern entity(e: Entity, n: java String) {
    Entity.name(e,n);
}
```

entity	
e: Entity	n: EString
e1	n1
e2	n2
e3	n2



A mintaillesztő modellmérettől függetlenül konstans idő alatt végzi el!

VQL Egyszerű lekérdezések

```
// e egy Egyed-Kapcsolat diagram entitása, n névvel
pattern entity(e: Entity, n: java String) {
    Entity.name(e,n);
}
// Típus kényszerként
pattern entity(e,n) {
    Entity(e);
    NamedElement.name(e,n);
}
// Intelligens típuskövetkeztetés
pattern entity(e,n) {
    Entity.name(e,n);
}
// Reláció bal végének multiplicitása 1..1
pattern multiplicityOne(r: Relation) {
    Relation.leftEnding.nullable(r, false);
    Relation.leftEnding.multiplicity(r, MultiplicityType::One);
}
```

VQL – Egyszerű lekérdezések

```
// Egy Egyed-Kapcsolat diagram entitása, n névvel
pattern entity(e: Entity, n: java String) {
    Entity.name(e,n);
}
// Típus kényszerként
pattern entity(e,n) {
    Entity(e);
    NamedElement.name(e,n);
}
// Intelligens típusköve
pattern entity(e,n) {
    Entity.name(e,n);
}
// Reláció bal végének multiplicitása 1..1
pattern multiplicityOne(r: Relation) {
    Relation.leftEnding.nullable(r, false);
    Relation.leftEnding.multiplicity(r, MultiplicityType::One);
}
```

Lekérdezés definíció

Opcionális paraméter típus

Típus kényszer

EMF típusok és Java adattípusok támogatása

Lekérdezés paraméterek

Attribútum navigáció

Kényszerek konjunkciója

Útvonal kifejezés

VQL — Minta kompozíció és negáció

```
// trace egy nyomkövetési objektum
// egy erdiagram elem és egy rdb elem között
pattern traceOfElement(traceRoot,trace,erElement,rdbElement) {
    TraceRoot.traces(traceRoot, trace);
    Trace.erdiagramElement(trace, erElement);
    Trace.rdbElement(trace, rdbElement);
}
```

```
// Egy trace objektum egy erdiagram elemet több rdb elemhez köt
pattern nondeterministicTrace(trace) {
    find traceOfElement(_,trace,from1,to);
    find traceOfElement(_,trace,from2,to);
    from1 != from2;
}
// Tábla, ami nem köthető erdiagram elemhez
pattern tableToRemove(table: Table, trace: Trace) {
    neg find traceOfElement(_,_,_,table);
    Trace.rdbElement(trace, table); // A trace objektumot is töröljük
}
```

Minta kompozíció / hívás

Negáció „nincs olyan”

Anonim változók „bármely” (ld. Prolog/Datalog)

Tranzitív lezárt és diszjunkció

```
pattern relation(from,to) {
    Relation.leftEnding.target(r,from);
    Relation.rightEnding.target(r,to);
}

pattern reachable(from:Entity,to:Entity) {
    from == to;
} or {
    find relation+(from,to);
}

pattern unreachableEntity(e1:Entity, e1:Entity) {
    EntityRelationDiagram.entities(erd,e1);
    EntityRelationDiagram.entities(erd,e2);
    neg find reachable(e1,e2);
}
```

Megjegyzendő:
- negatív hívások nem kötik a fejléc paraméter változókat
- a mintákat élek kössék össze (kerüljük a Descartes-szorzatot)

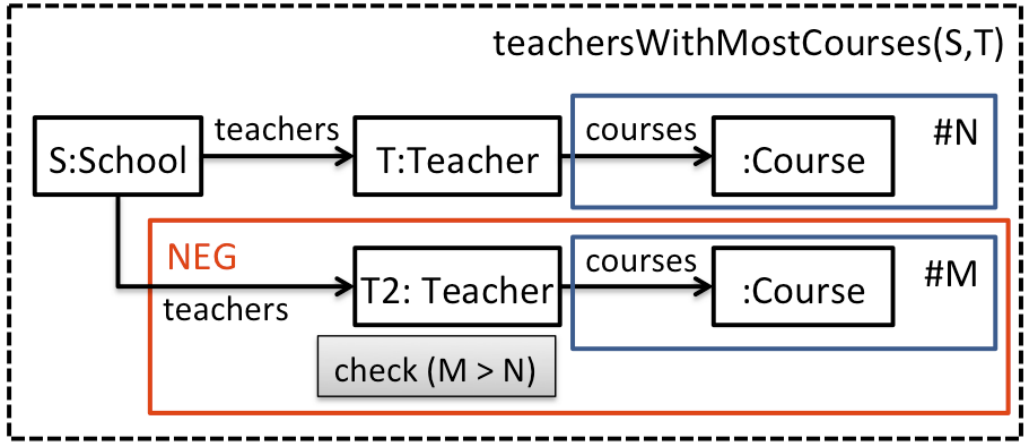
Diszjunkció (minta szintjén)

Tranzitív lezárt bináris (2-param) minták fölött

Vagy ekvivalens módon:

```
neg find relation*(e1,e2);
```

VQL Check kifejezés & Aggregáció



Feladat 1:
Legtöbb tárgy egy iskolában vagy összesen?

Feladat 2:
Ez még hatékonyabb lenne **max find**…
használatával. Hogyan és miért?

```
pattern teachersWithMostCourses(school : School, teacher : Teacher) {
    School.teachers(school,teacher);
    neg find moreCourses(teacher);
}
pattern moreCourses(teacher : Teacher) = {
    n == count find coursesOfTeacher(teacher,_course);
    m == count find coursesOfTeacher(teacher2,_course2);
    Teacher(teacher2); // Ugyanabban az iskolában?
    teacher != teacher2;
    check(n < m);
}
```

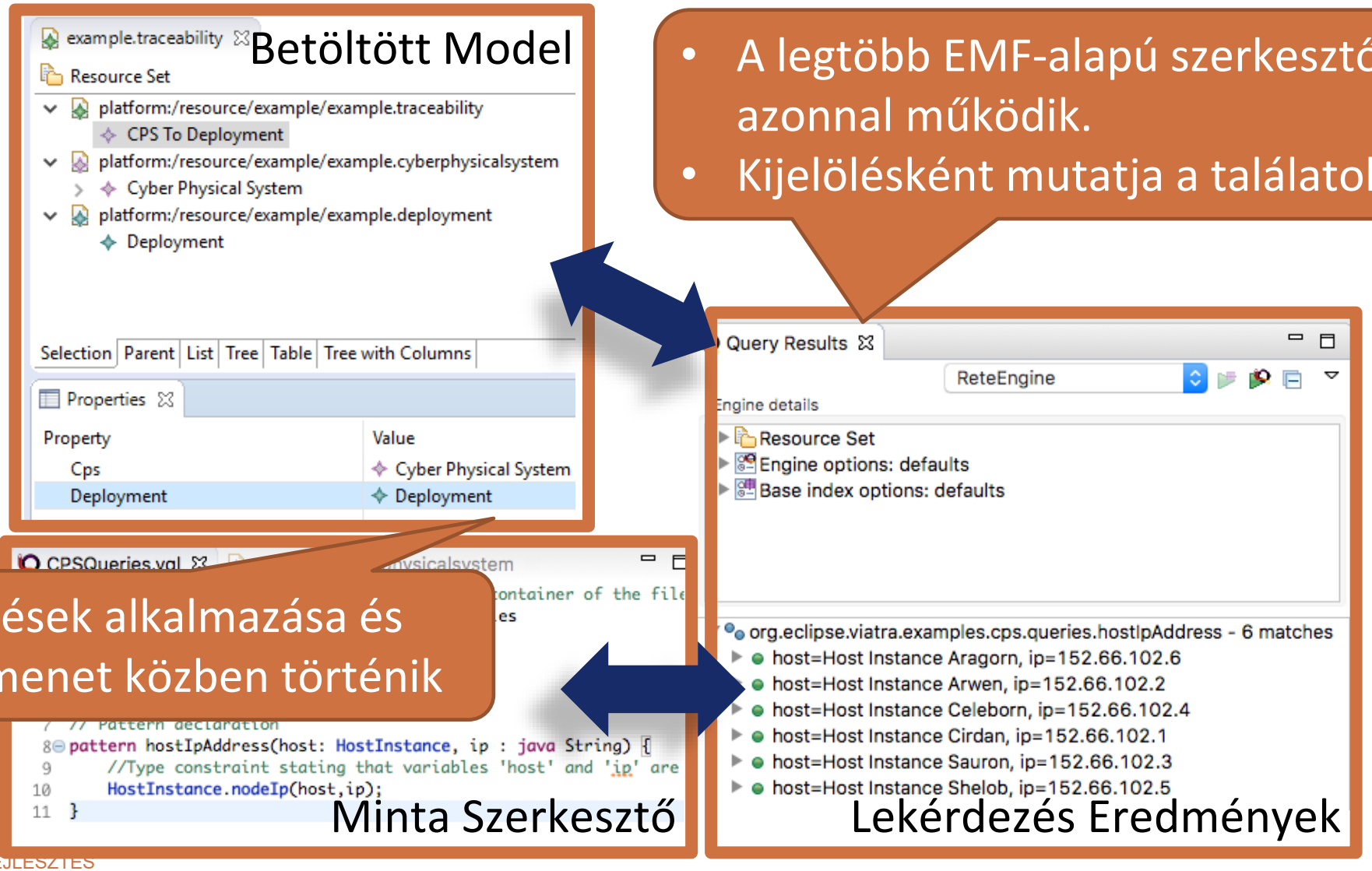
Match számolás

Check kifejezés attribútum értékekre (pure!)

VIATRA QUERY Language áttekintése

- A mintaleíró nyelv jellemzői
  - Bármilyen (tiszta) EMF-alapú DSL-lel és alkalmazással együttműködik
  - Újrafelhasználhatóság minták kompozíciójával
  - Rekurzió, negáció
  - Generikus és paraméterezett modell-lekérdezések
  - Élek / hivatkozások kétirányú navigálhatósága
  - Azonnali hozzáférés egy típus összes példányához
  - Komplex változás-felismerés
- Előnyök
  - Teljesen deklaratív + skálázódó teljesítmény

VIATRA QUERY Fejlesztői Eszközök



- A legtöbb EMF-alapú szerkesztővel azonnal működik.
- Kijelölésként mutatja a találatokat

VQL — A szemantika tisztázása

```
pattern reachableRec(from:State,to:State) {
    from == to;
} or {
    find transition(from, intermediate);
    find reachableRec(intermediate, to);
}
```

Query

from	to
s1	s1
s1	s2
s2	s2
…	…

Eredmények

- **Halmaz** szemantika → lekérdezés eredmények egy **relációt** alkotnak (tuple-ök halmaza)
  - A visszaküldött tuple-ök sorrendje nemdeterminisztikus
  - Nincsenek duplikált tuple-ök (szuper fontos az aggregáláshoz!)
    - Még akkor sem, ha egy rejtett belső változóban különböznek (pl. **`intermediate`** / köztes)
    - Még akkor sem, ha különböző **`or`**-ral összekapcsolt minta törzsekből származnak (pl. <s1,s1> cikluson keresztül)
- (Parciális) paraméterkötés/helyettesítés
  - Az összes s1-ből elérhető állapot megkeresése ⇔ s1 helyettesítés a **`from`**-ba, reláció szűrése
- Rekurziós szemantika: legkisebb fix pont
  - (Futásidejű opcióváltás szükséges)

Nem kell előre meghatározni, melyik paraméter bemenet/kimenet

Rekurzió nehéz, ezért támogatja a tranzitív lezártat

Gráfmintaillesztés, Modelltranszformáció

I. Gráftranszformáció

II. Viatra Query Language

III. Viatra Validation Framework

IV. Benchmark

VIATRA QUERY Validation Framework

- Egyszerű validációs motor
  - > Támogatja a menet közbeni validációt inkrementális mintaillesztés és problémamarker-kezelés révén
  - > VIATRA QUERY gráfmintákat használ a kényszerek megadásához
- Szimulálja az EMF Validációs Markereket
  - > A kompatibilitás és a meglévő szerkesztőkkel való egyszerű integráció biztosítása érdekében
  - > Nem használja közvetlenül az EMF Validation-t
  - > A végrehajtási modell eltérő

Jólformáltsági szabály specifikációja gráfmintákkal

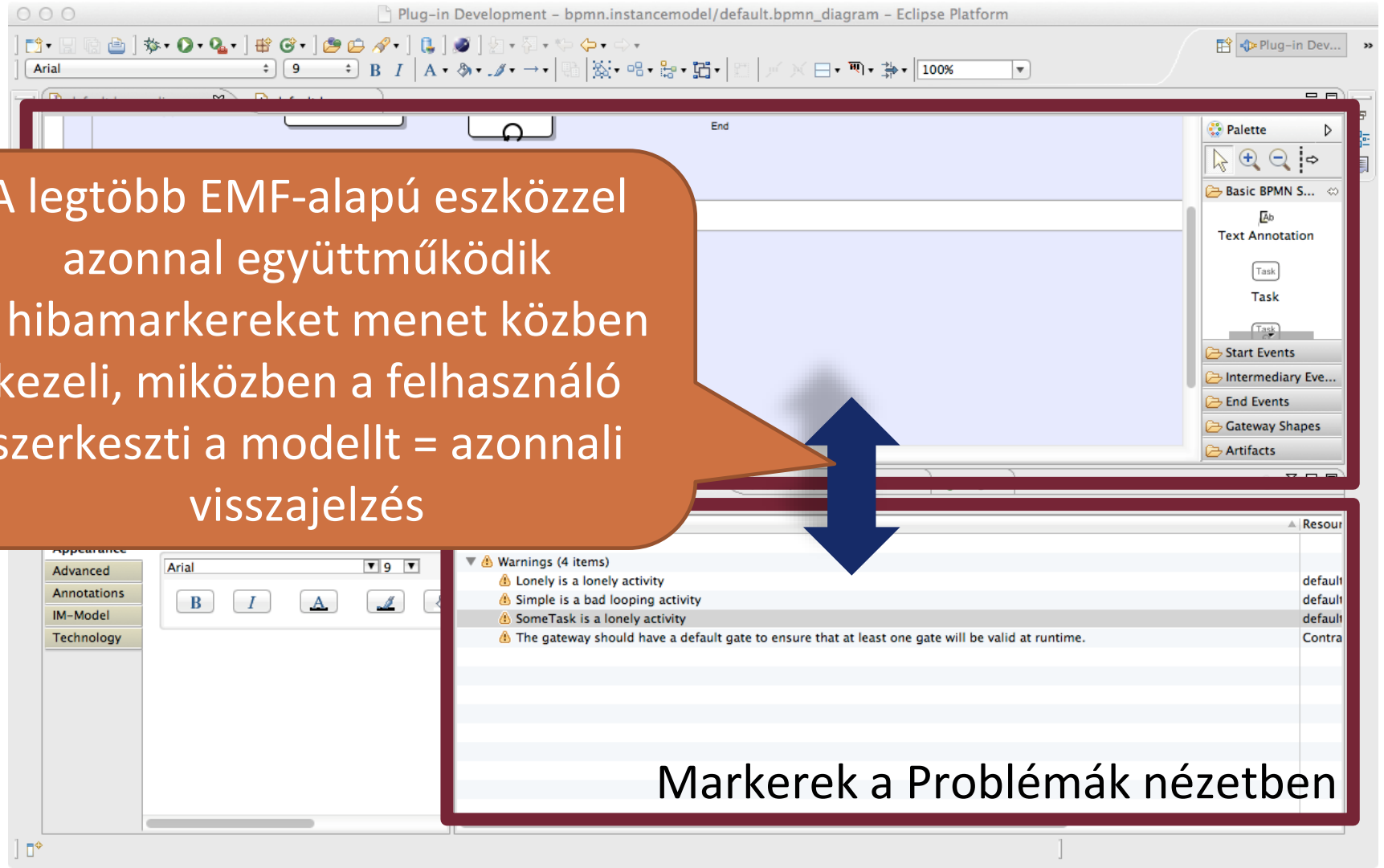
- Jólformáltsági szabály (Well-formedness rule / WFR): *Invariánsok* amiknek mindig teljesülniük kell
- Specifikáció = elemi kényszerek halmaza + kontextus
  - > Alapvető kényszerek: Query (Lekérdezés / minta)
  - > Hely/kontextus/kulcs: egy modellelem, amelyre a problémamarkert helyezzük.
- A gráfminták által meghatározott kényszerek
  - > A "rossz eset" mintájának meghatározása
    - – Vagy közvetlenül
    - – Vagy a "jó eset" definíciójának negálásával.
  - > Az egyik változót jelöljük ki helyként/kontextusként.

Match (Illeszkedés): Az invariáns megsértése

EXAMPLE Validation constraint

```
@Constraint(key = {e}, message = „Entity name is missing", severity =
"warning" )
pattern entity(e) {
    Entity.name(e,"");
}
```

PÉLDA GUI – VIATRA Model Validation



Validálás életciklusa

- Kényszerek megsértése
  - Problémamarkerekkel (Problem Marker) ábrázolva (Problems nézet)
  - A Marker szövege frissül, ha az érintett elemek megváltoznak a modellben.
  - A marker törlődik, ha a szabályszegés már nem áll fenn
- Életciklus
  - Szerkesztőhöz kötött validáció (a markerek eltávolítása a szerkesztő bezárásakor)
  - Inkrementális karbantartás futó szerkesztőn kívül nem praktikus

Gráfmintaillesztés, Modelltranszformáció

I. Gráftranszformáció

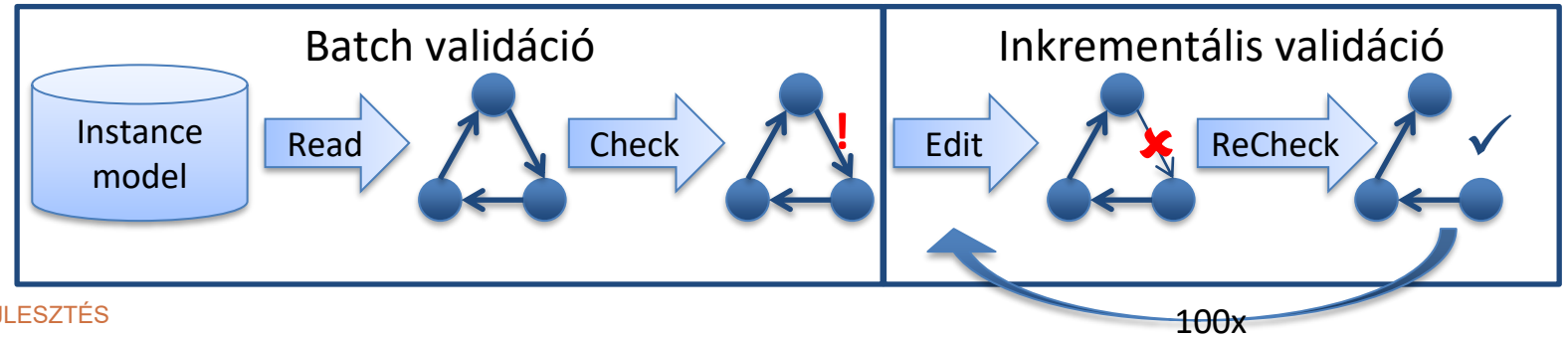
II. Viatra Query Language

III. Viatra Validation Framework

IV. Benchmark

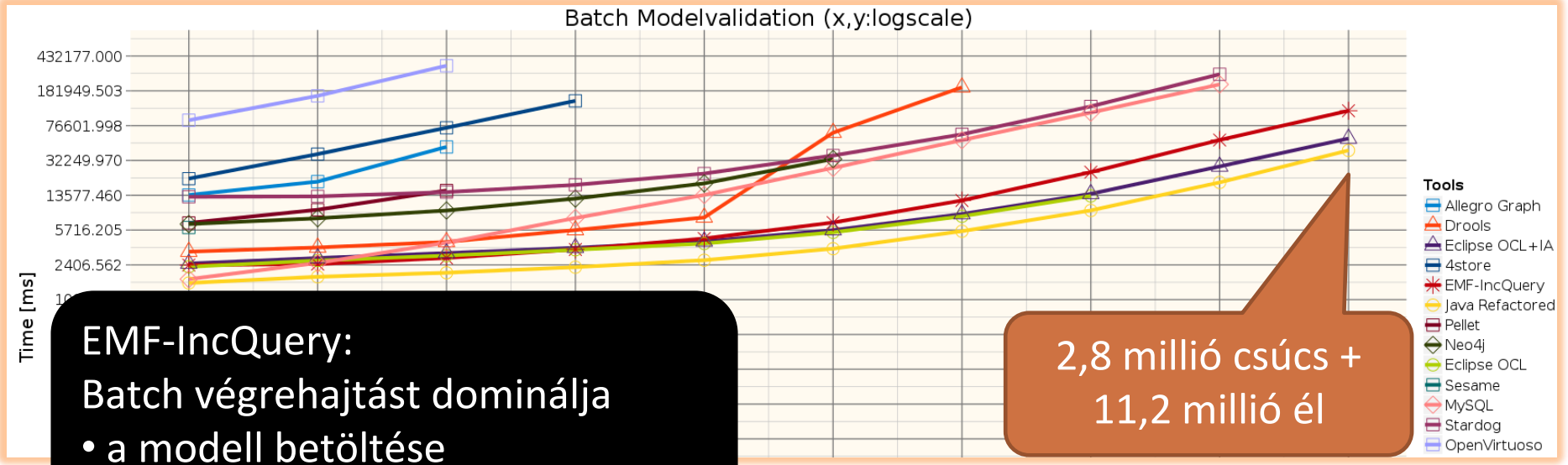
A Train Benchmark

- Modellvalidációs munka:
  - A felhasználó szerkeszti a modellt
  - A jólformáltsági korlátozások azonnali validálása
  - A modellt ennek megfelelően javítjuk
- Scenario:
  - Load
  - Check
  - Edit
  - Re-Check
- Modellek:
  - Véletlenszerűen generált
  - Közel áll a valós esetekhez
  - Különböző metrikákat követve
  - Testreszabott eloszlások
  - Alacsony számú szabálysegés
- Lekérdezések :
  - Két egyszerű lekérdezés (<2 objektum, attribútum)
  - Két összetett lekérdezés (4-7 join, negáció stb.)
  - Validált találati halmazok



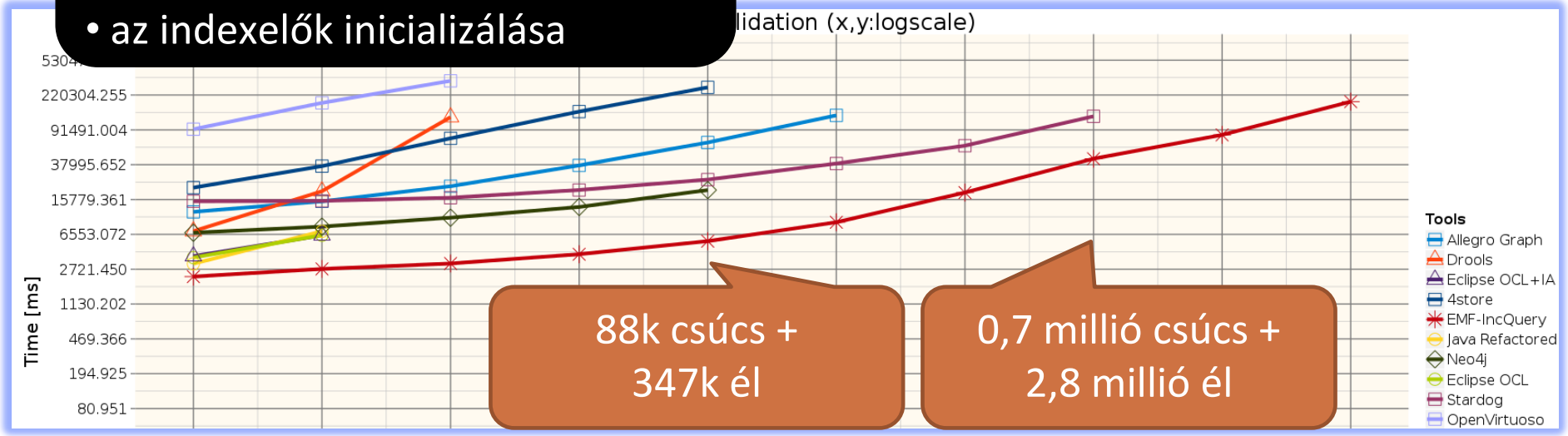
Milyen eszközöket hasonlítunk össze?

Batch validáció futásideje (összetett lekérdezések)



EMF-IncQuery:
Batch végrehajtást dominálja
- a modell betöltése
- az indexelők inicializálása

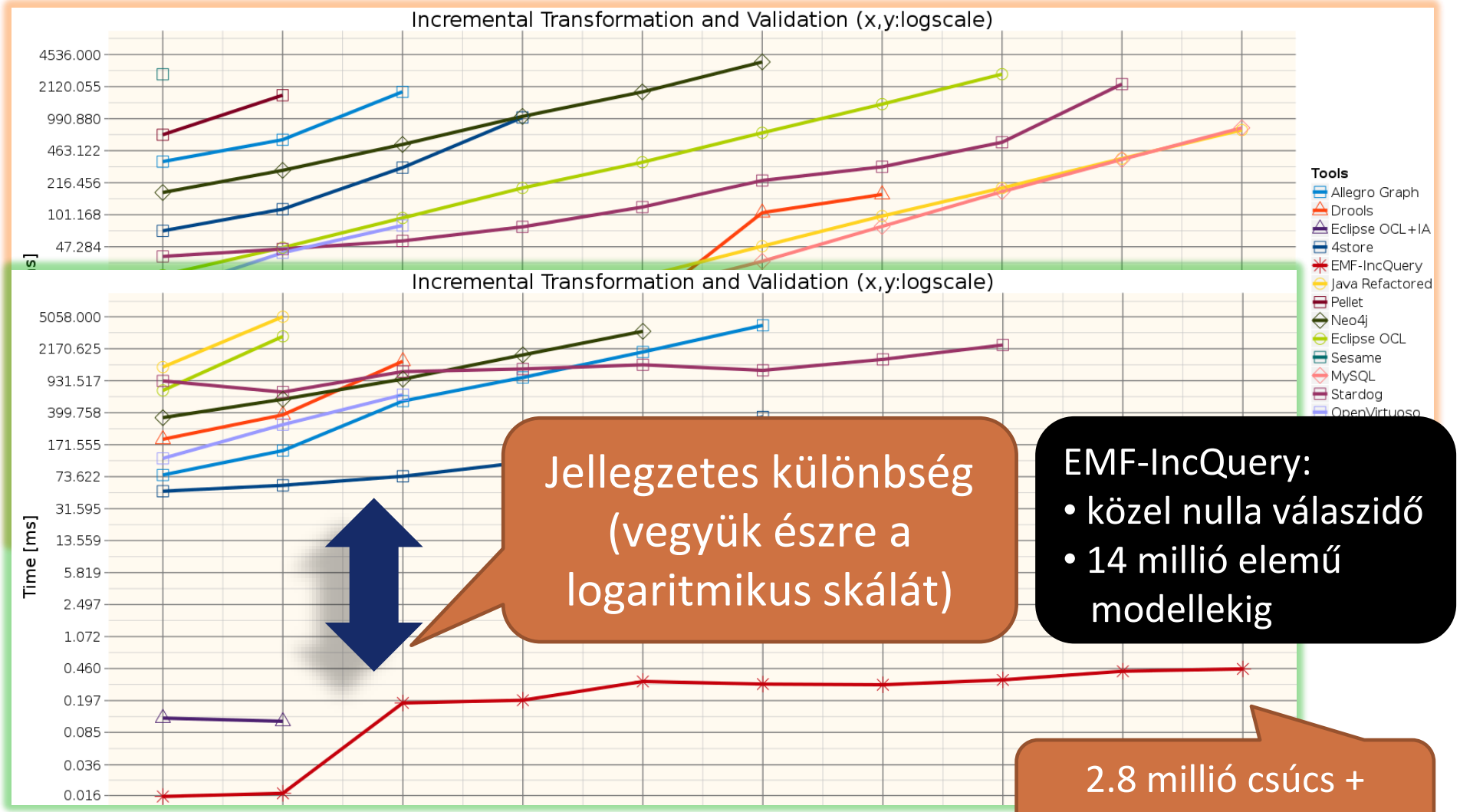
2,8 millió csúcs +
11,2 millió él



88k csúcs +
347k él

0,7 millió csúcs +
2,8 millió él

Újravalidálási idő (összetett lekérdezések)



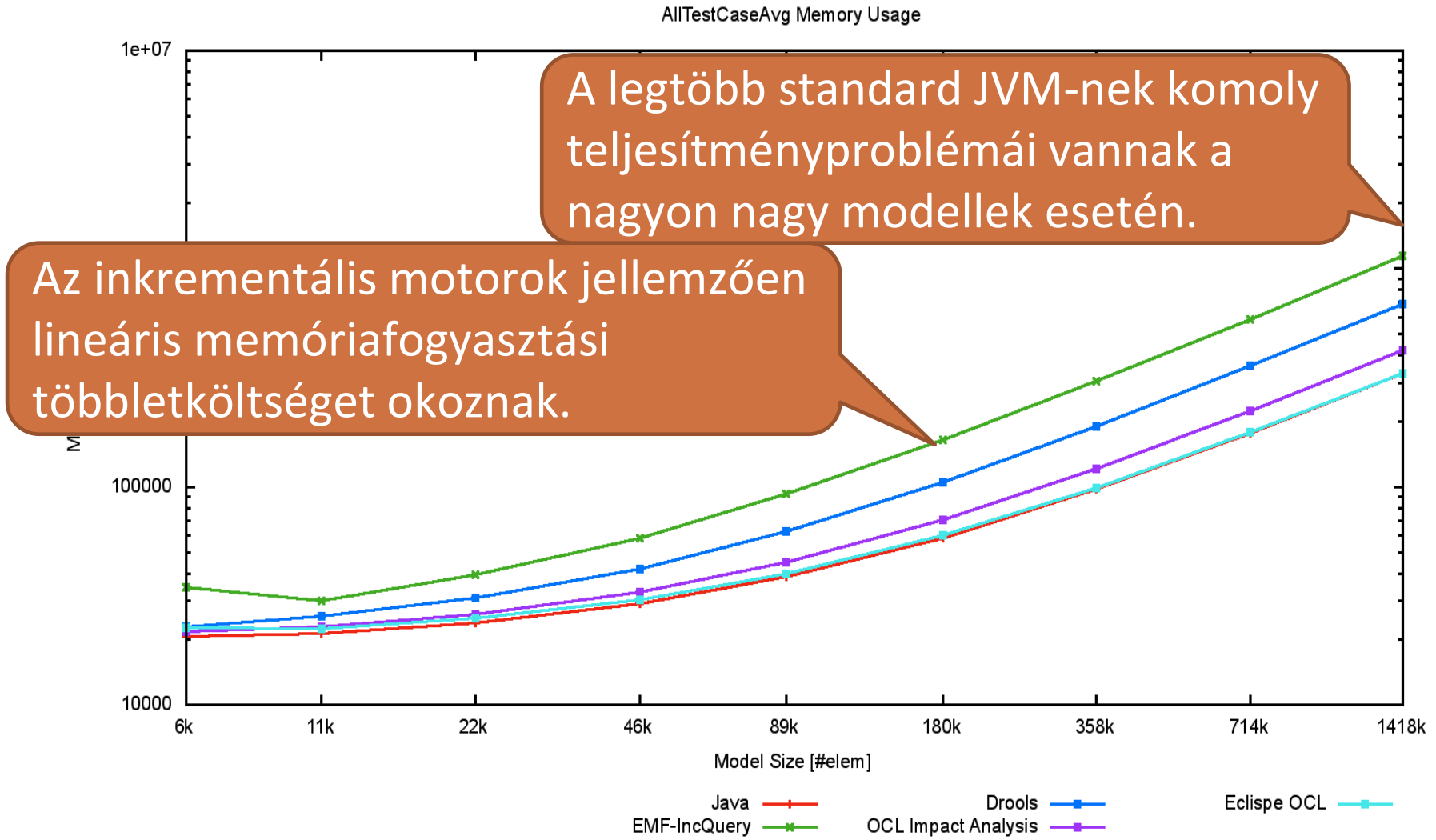
Jellegzetes különbség (vegyük észre a logaritmikus skálát)

EMF-IncQuery:
- közel nulla válaszidő
- 14 millió elemű modellekig

2.8 millió csúcs + 11.2 millió él

http://incquery.net/publications/trainbenchmark további részletekért

Memória használat



A legtöbb standard JVM-nek komoly teljesítményproblémái vannak a nagyon nagy modellek esetén.

Az inkrementális motorok jellemzően lineáris memóriafogyasztási többletköltséget okoznak.

VIATRA QUERY Néhány Alkalmazása

- Komplex nyomon követhetőség
- Lekérdezésvezérelt nézetek
- Absztrakt modellek származtatott objektumokkal

Eszköztár az IMA konfigurációkhoz

- Csatlakozás Matlab Simulink modellhez
- Export: Matlab2EMF
- Modell módosítása EMF-ben
- Re-import: EMF2Matlab

MATLAB-EMF Bridge

- Élő modellek (25 képkocka/s frissítés)
- Komplex eseményfeldolgozás

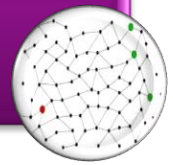
Gesztusfelismerés

- Kísérletek nyílt forráskódú Java projekteken
- Local search vs. Inkrementális vs. Native Java kód

Code smell felismerése

- Műveletek szabályai
- Komplex szerkezeti kényszerek (GP-ként)
- Hintek és útmutatás
- Potenciálisan végtelen állapottér

Tervezési tér feltárása



- Itemis (fejlesztő)
- Embraer
- Thales
- ThyssenKrupp
- CERN

Ismert felhasználók