

# Modellező eszközök, kódgenerálás

## Rendszermodellezés 2020.

**Budapesti Műszaki és Gazdaságtudományi Egyetem  
Méréstechnika és Információs Rendszerek Tanszék**



# MOTIVÁCIÓ: MI A KÓD SZEMANTIKÁJA?

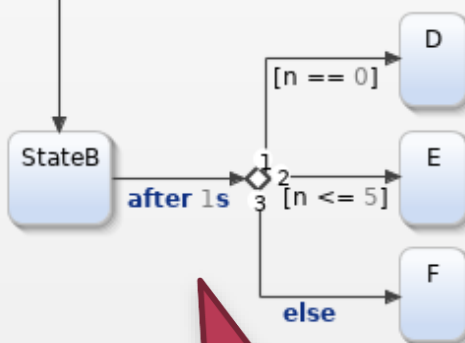
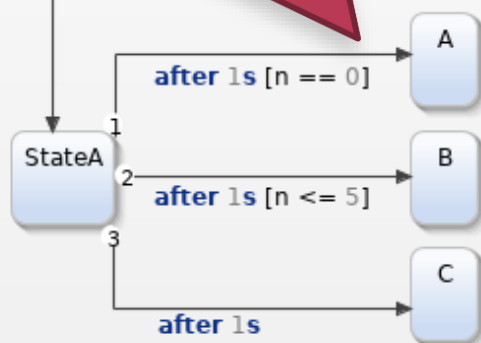
# Példa: Házi feladat 2019.

- „egy programhiba miatt egyes esetekben hibásan működik, így a kiértékelő tévesen hibát fog jelezni....”
- Feladat
  - Azonos időzítésű...
  - Egymással akár átfedő őrfeltételű ( $x=1$ ,  $x<5$ )...
  - Különböző tranzíciók kezelése

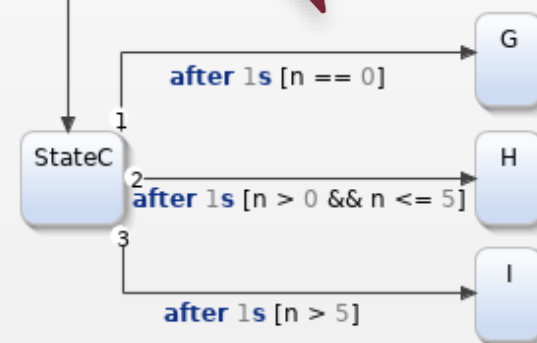
# Lehetséges megoldások

main region

Prioritás használata  
→ Események kiértékelése  
befolyásolta!



Egymást explicit  
kizáró őrfeltételek

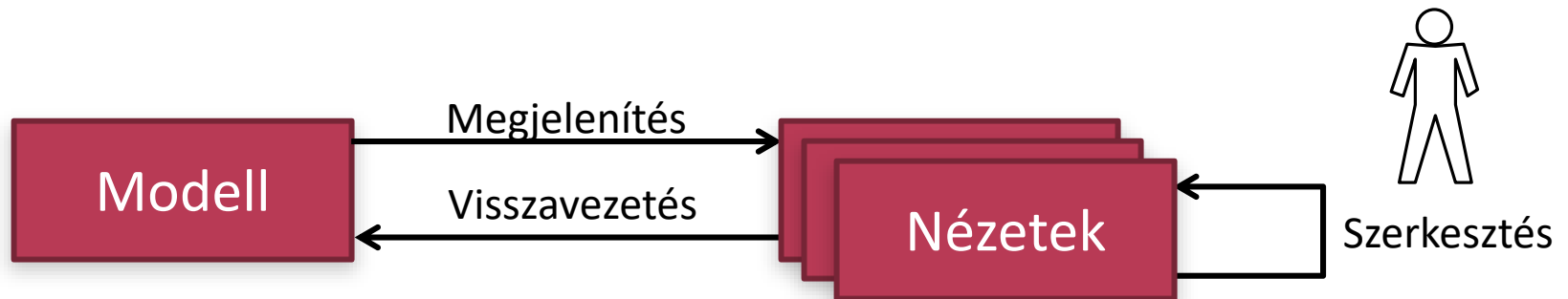


Choice + default ág

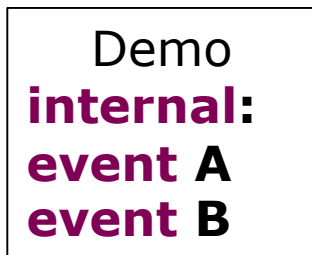
# Miről lesz szó?

- Modellező eszközök funkciói
- Kódgenerálás
- Esettanulmány

# Modellező eszközök funkciói



## Szöveges



## Grafikus



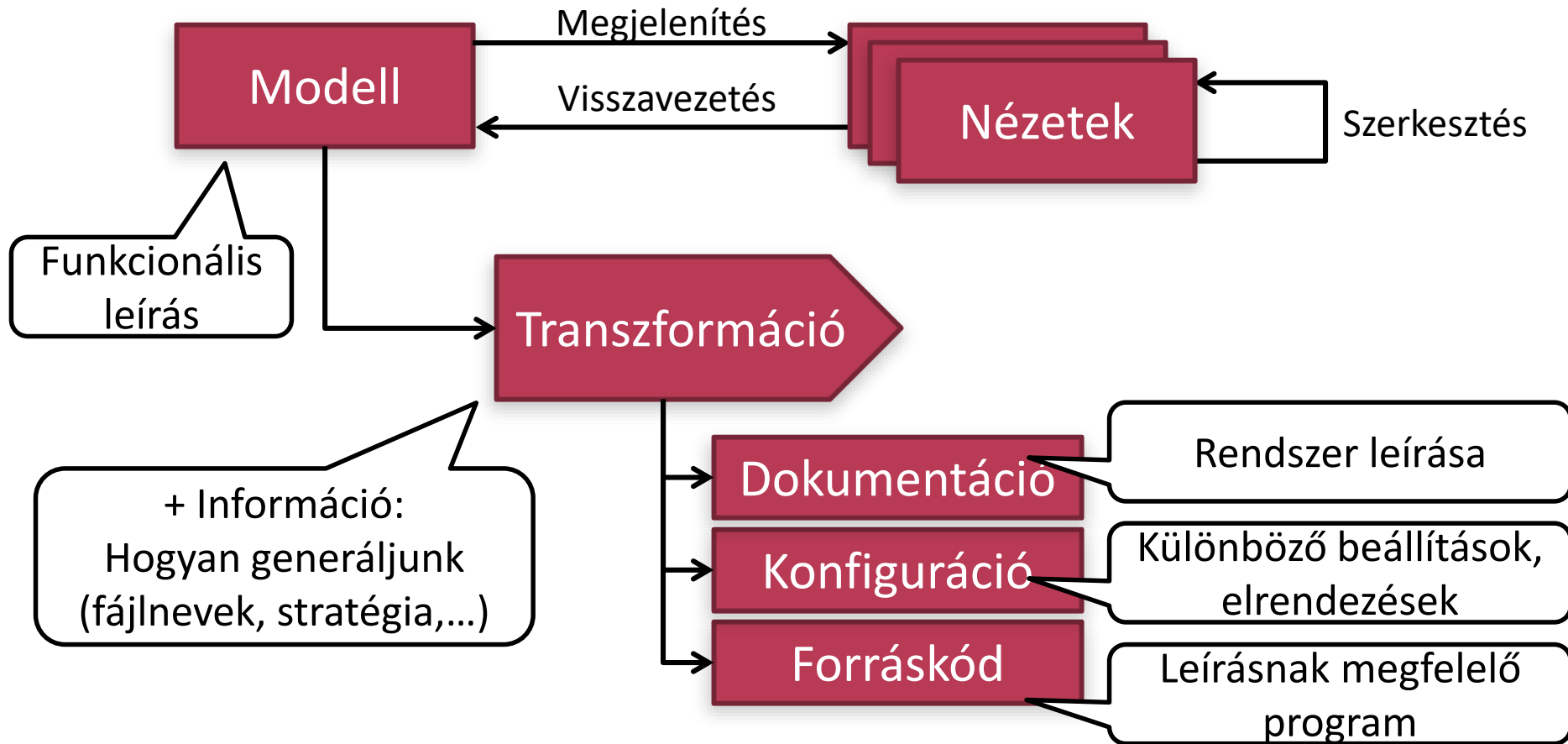
## Egyéb szerkesztőfelületek

Properties

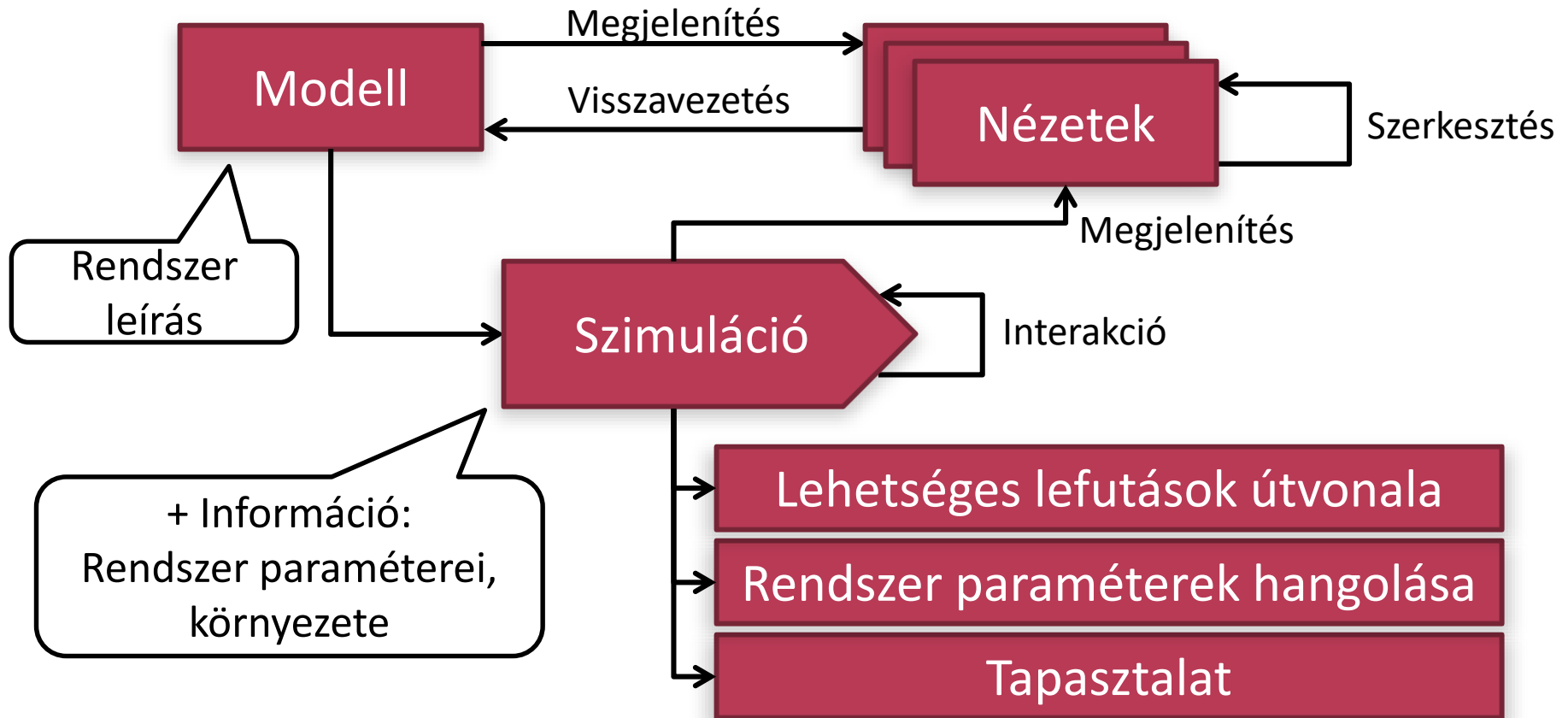
**S State READY**

Model	State Name:
Appearance	READY
	entry/ Display.text = "Ready to play"
	entry/ Display.blackDisplay = -1
	entry/ Display.whiteDisplay = -1

# Modellező eszközök funkciói



# Modellező eszközök funkciói



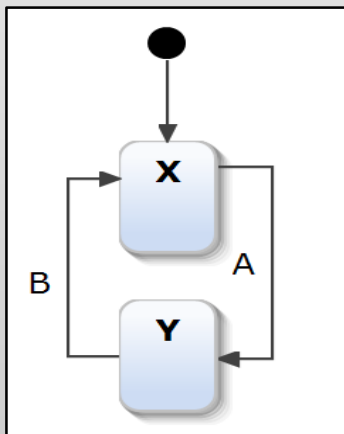


# MODELLEZŐ ESZKÖZÖK

# Yakindu modellezési funkciói

Konkrét szintaxis  
(Szerkesztők)

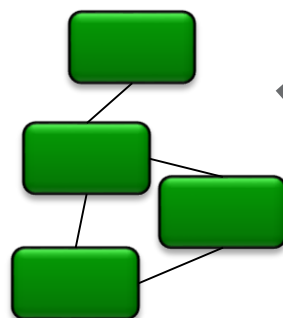
Grafikus



Szöveges

Demo  
**internal:**  
**event A**  
**event B**

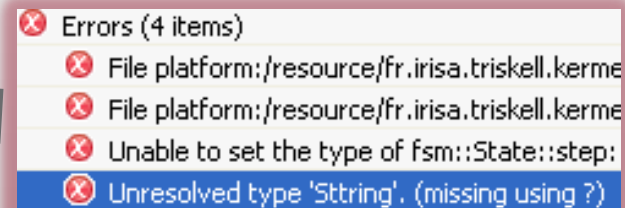
Szintaxis → Szemantika



Modell  
(Absztrakt szintaxis)

Modellező funkciók

Ellenőrzés



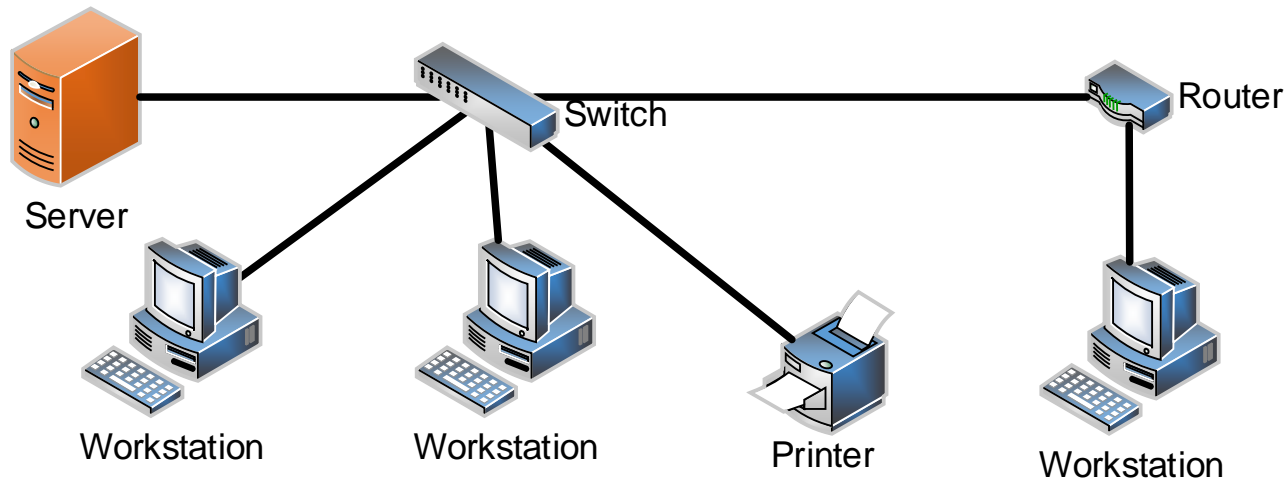
Kódgenerálás

```
</membership>  
<profile defaultProvider="Sitefinity">  
<providers>  
<clear/>  
<add name="Sitefinity" connectionS  
</providers>  
<properties>  
<add name="FirstName"/>  
<add name="LastName"/>  
<!-- SNP specific properties -->  
<add name="NickName" />  
<add name="Gender" />
```

(Forráskód, dokumentáció,  
konfiguráció)

# Absztrakt szintaxis

- **Definíció:** a szerkesztés alatt álló rendszermodell strukturális modellje.
  - A modell felépítésének modellje???
- Modellező program kezeli
- Emlékeztető: strukturális modell = **gráf**
  - **csomópontok, élek és tulajdonságok gráfja**

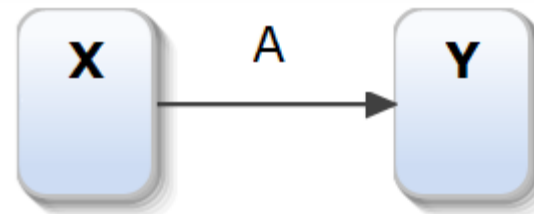


# Absztrakt szintaxis példa: Yakindu

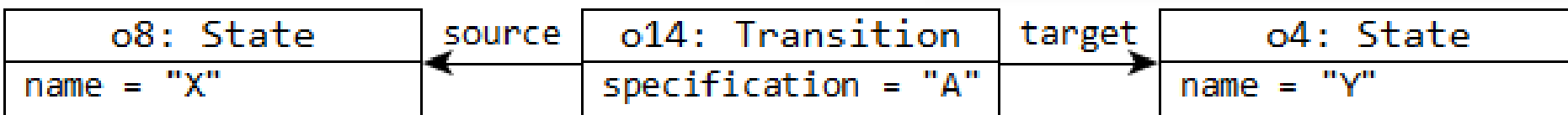
## Kérdés:

Hogyan készítenénk modellező programot?

## Példa: Yakindu modell



## Absztrakt szintaxis

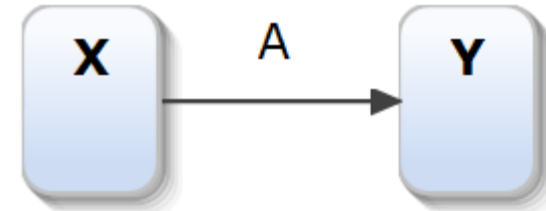


# Absztrakt szintaxis példa: Yakindu

## Kérdés:

Hogyan készítenénk modellező programot?

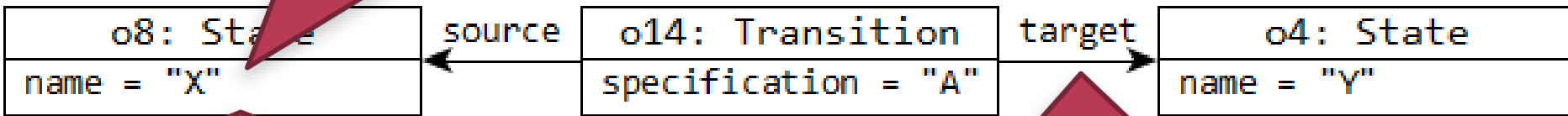
## Példa: Yakindu modell



Neveket String-ként tároljuk

```
name = "X"
```

## Absztrakt szintaxis



Modellelemeket objektumként

Kapcsolatokat "pointerekkel"

Válasz: objektum-orientált program + Extra funkciók

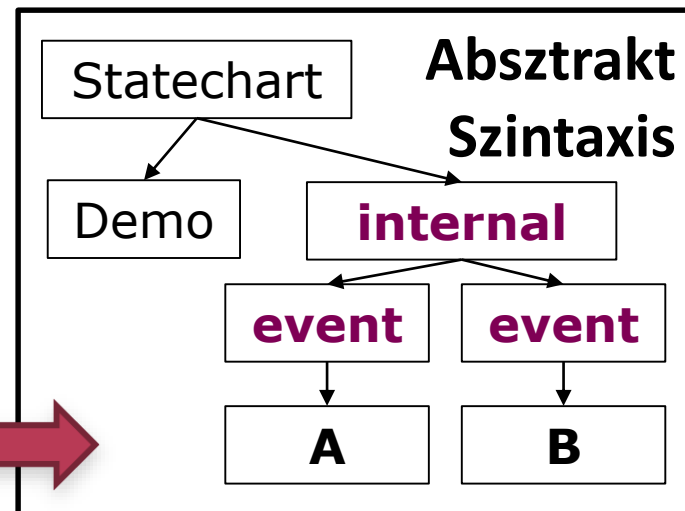
# Konkrét szintaxis: Szöveges szintaxis

- **Cél:** konkrét megjelenítés  $\Leftrightarrow$  mögöttes modell
- Szöveges szintaxis (programozási nyelv)
  - Feladat: Szöveg  $\Leftrightarrow$  Modell
  - Nyelvtani szabályok alapján

Demo  
**internal:**  
**event A**  
**event B**

**Nyelvtan**

```
<Statechart> ::= <Name> <Interface>*  
<Interface> ::= ("internal" | <Name>) ":"  
                <Event>*  
<Event> ::= "event" <Name>  
<Name> ::= ...
```

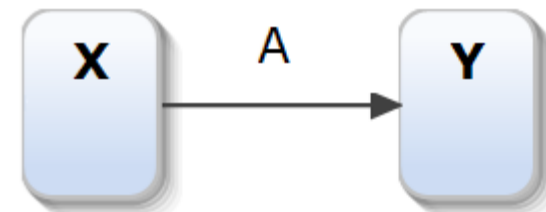


Megfelelő technológiákkal (pl. Xtext) könnyű saját modellező / programozási nyelvet csinálni!

# Konkrét szintaxis: Grafikus szintaxis

## ■ Grafikus szintaxis (Diagram)

- Feladat: Diagram  $\leftrightarrow$  Modell
- Könnyebben átlátható, nehezebben írható
- **Nézeti modell szabályok**



### Feltétel a modellen

Id\*:

Domain Class\*:

Semantic Candidates Expression:

### Diagram elem létrehozása

Label Alignment:  Left  Center  Right

Label Expression:

Label Position:

Color\*:

Label Color\*:

Border Color\*:

**Feltétel a modellben teljesül  $\rightarrow$  Diagram elem létrejön**  
**Diagram változik  $\rightarrow$  Modell változik**

# Konkrét szintaxis: Grafikus szintaxis

Eredmény:

The screenshot shows a graphical modeling tool interface. At the top is a toolbar with various icons for editing and navigation. Below the toolbar, two states are visible: 'X' (a solid blue square) and 'Y' (a dashed blue square). Below the diagram is a 'Properties' panel for 'State Y'. The panel has tabs for 'Properties' and 'Problems', and a sub-tab for 'State Y'. The 'State Y' sub-tab is active, showing a table of properties and their values.

Property	Value
State Y	
Composite	<input checked="" type="checkbox"/> false
Documentation	<input type="checkbox"/>
Incoming Transitions	→ X -> Y (A)
Leaf	<input checked="" type="checkbox"/> true
Name	<input type="checkbox"/> Y

**Megfelelő technológiákkal (pl. Sirius) könnyű saját modellező / programozási nyelvet csinálni!**



# Modellek validálása: szintaxisellenőrzés

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

**Deklarálás interfészen:**

```
var clock: integer = 60
```

**Használat modellben:**

```
after 1 s [clock>0]/ clock -= 1
```

- Szintaxisvezérelt szerkesztő
  - Szerkesztés közben hiba → **Couldn't resolve reference**
  - Fejlett szerkesztőeszköz (például lehetőségek felkínálása)

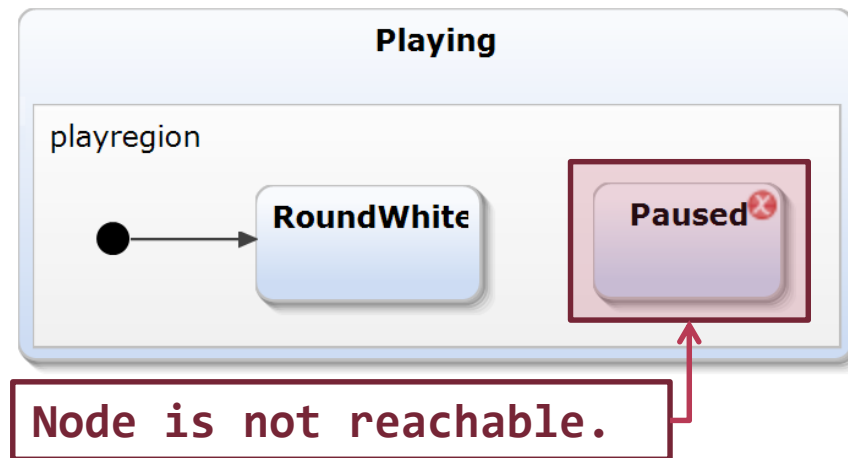
```
after 1 s [clock>0]/ clock -= 1
```



- Kód és diagram együtt
- Programozás: szerkesztés közben **hibás**  
Modellezés: szerkesztés közben **helyes**  
(legalább az alapvető struktúrája)

# Modellek validálása: strukturális helyesség

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



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

# KÓDGENERÁLÁS

# Motiváció

- Fejlesztési idő lerövidítése:  
modellek/követelmények/tervek alapján
  - dokumentáció
  - forráskód
  - konfigurációautomatikus generálása
- Példa: ReMo HF
  - Feladat: paraméterek → feladatkiírás (különböző nyelveken), tesztesetek
  - Yakindu: modell → kód

# Feladatkiírás generálás példa

- Példa: Játékosok kezdeti gondolkodási ideje
- Paraméterek:
  - Gondolkodási idő kiinduló értéke
  - Gondolkodási idő változtatása
    - Nem lehetséges
    - Mindkét játékosnak egyszerre
    - Játékosokra külön-külön
- LaTeX kód:

A játék indításakor mindkét játékos gondolkodási időt mérő órája beáll a kezdeti gondolkodási időre. A kezdeti gondolkodási idő mértéke `\ifbool{setupInitialTimeSettable}{a sakkóra bekapcsolásakor}\setupInitialTimeDefault{}` másodperc mindkét játékos esetén `\ifbool{setupInitialTimeSettable}{}`, de `\ifbool{setupInitialTimeIndividual}{játékosonként külön-külön}` megváltoztatható ez az érték (ld. `\refstruc{sec:settings-details}`).

# Feladatkiírás generálás példa

- LaTeX kód:

A játék indításakor mindkét játékos gondolkodási időt mérő órája beáll a kezdeti gondolkodási időre. A kezdeti gondolkodási idő mértéke `\ifbool{setupInitialTimeSettable}{a sakkóra bekapcsolásakor}\setupInitialTimeDefault{}` másodperc mindkét játékos esetén `\ifbool{setupInitialTimeSettable}{, de \ifbool{setupInitialTimeIndividual}{játékosonként külön-külön}` megváltoztatható ez az érték (ld. `\refstruc{sec:settings-details}`).

- Generált feladatkiírás

A játék indításakor mindkét játékos gondolkodási időt mérő órája beáll a kezdeti gondolkodási időre. A kezdeti gondolkodási idő mértéke a sakkóra bekapcsolásakor 150 másodperc mindkét játékos esetén, de megváltoztatható ez az érték (ld. 2.4. alszakasz).

A játék indításakor mindkét játékos gondolkodási időt mérő órája beáll a kezdeti gondolkodási időre. A kezdeti gondolkodási idő mértéke a sakkóra bekapcsolásakor 180 másodperc mindkét játékos esetén, de játékosonként külön-külön megváltoztatható ez az érték (ld. 2.4. alszakasz).

# Programozó vs kódgenerátor

- **Jól megírt** kódgenerátor előnyei
  - Helyes kódot generál → Tanúsítvány
  - Gombnyomásra generálódik a kód
    - Minden változtatás után azonnal
  - Optimalizálható átláthatóságra, hatékonyságra, stb.
- Nehéz jól megírni egy kódgenerátort
  - Éppen a fentiek miatt

# Kódgenerálás feladatai

- **Feladat:** modellnek megfelelő viselkedésű program automatikus előállítás
- Több megoldás létezik → tervezői döntések
  - **Interpretált:** modellt beolvassuk és végrehajtjuk
  - **Generatív:** forráskód szintézise
  - **Programozási nyelvek:** Java, C, C++
  - **Optimalizálás:** memória vs CPU  
Megfigyelhetőség vs Teljesítmény
  - Hogyan kapcsoljunk saját kódot a generálthoz?
- **Kódgenerátor:** paraméterezhető + kiterjeszhető



# Generálás vs Interpretálás

## ■ Dinamikus interpreter

- Gyors indulás, eredmények azonnal
- Általában idő/memória overhead
- Futásidejű függőség
- Tudja támogatni a modell változtatását végrehajtás közben
- Viselkedés mindig megfelel a modellnek

## ■ Kódgenerálás

- Indulás generálás és fordítás után
- Hatékony futtatásra optimalizálható
- Változtatás után újra kell generálni
- Kézzel változtatható a kód  
→ *Ez biztos jó?*

# Kódgenerátor

- Mi is az a kódgenerátor?

- Tévhit:

*Olyan program, ami programokat ír...???*

- Valóság:

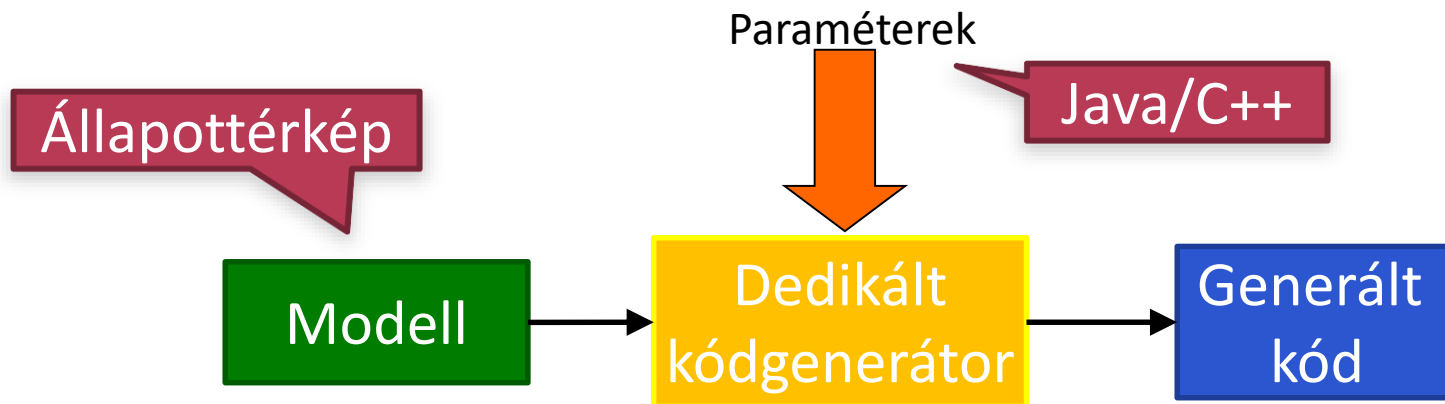
Egyszerű program sok-sok kiíratással

```
sourceFile.write("    temp = ((AIDA_PARTITION_TYPE*) selfModule.partitions.elements);\n" )
i = 0
for partition in partitions:
    numPorts = getNumberOfAllCommPorts_Partition(currModuleComm, interPartitionComm, partition.partitionName)
    sourceFile.write("    temp[" + str(i) + "].partition_id = " + str(partition.partitionID) + ";\n" )
    sourceFile.write("    strcpy( &temp[" + str(i) + "].partition_name[0], \"" + str(partition.partitionName) + "\");\n" )
    sourceFile.write("    temp[" + str(i) + "].ports.type = CONST_AIDA_PORTS_TYPE;\n" )
    sourceFile.write("    temp[" + str(i) + "].ports.elements = &mem_ports_" + str(partition.partitionName) + "[0];\n" )
    sourceFile.write("    temp[" + str(i) + "].ports.numOfElements = " + str(numPorts) + ";\n" )
    sourceFile.write("\n" )
    i = i + 1
## end for
sourceFile.write("\n")
```

- Kitekintés: *Quine* – program, ami kiírja a saját kódját

# Kódgenerátor típusok I.

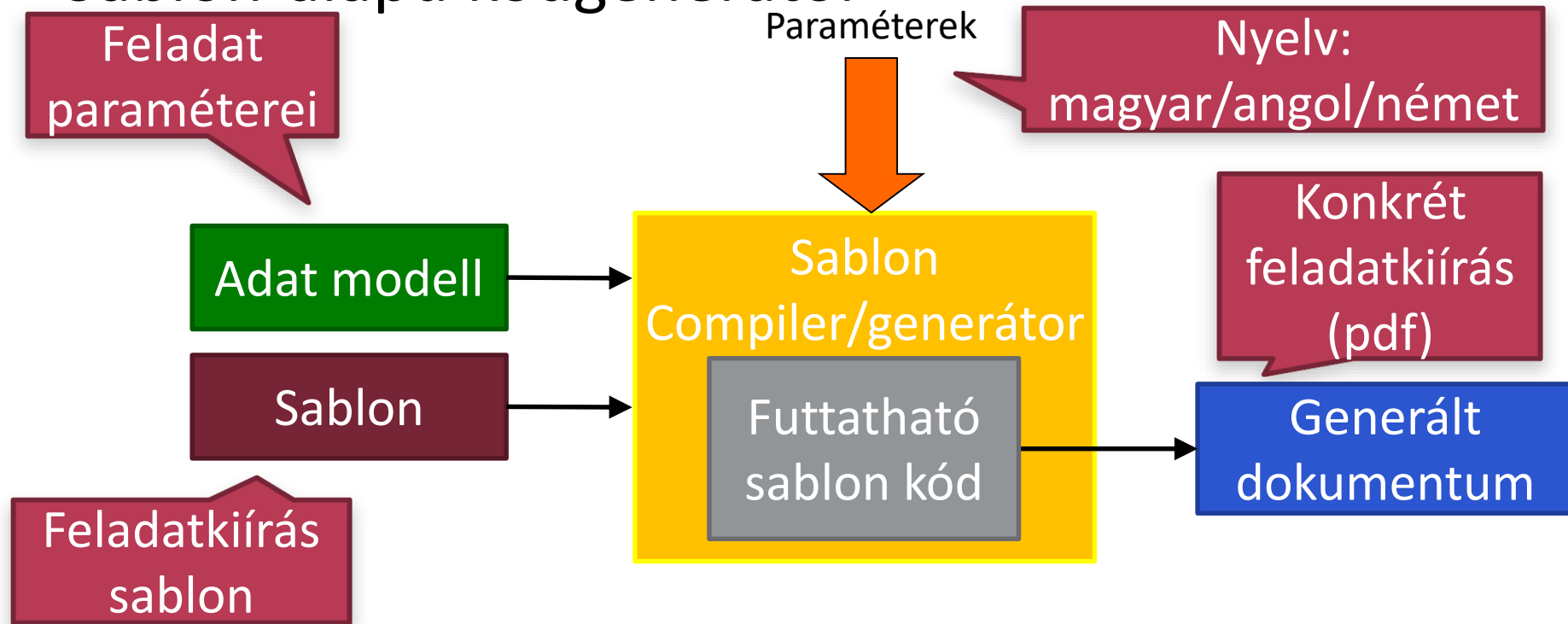
- Dedikált kódgenerátor (Id. Yakindu)



- Modellező eszköz része
- Tanúsítványozhatóság
- Alacsony testreszabhatóság

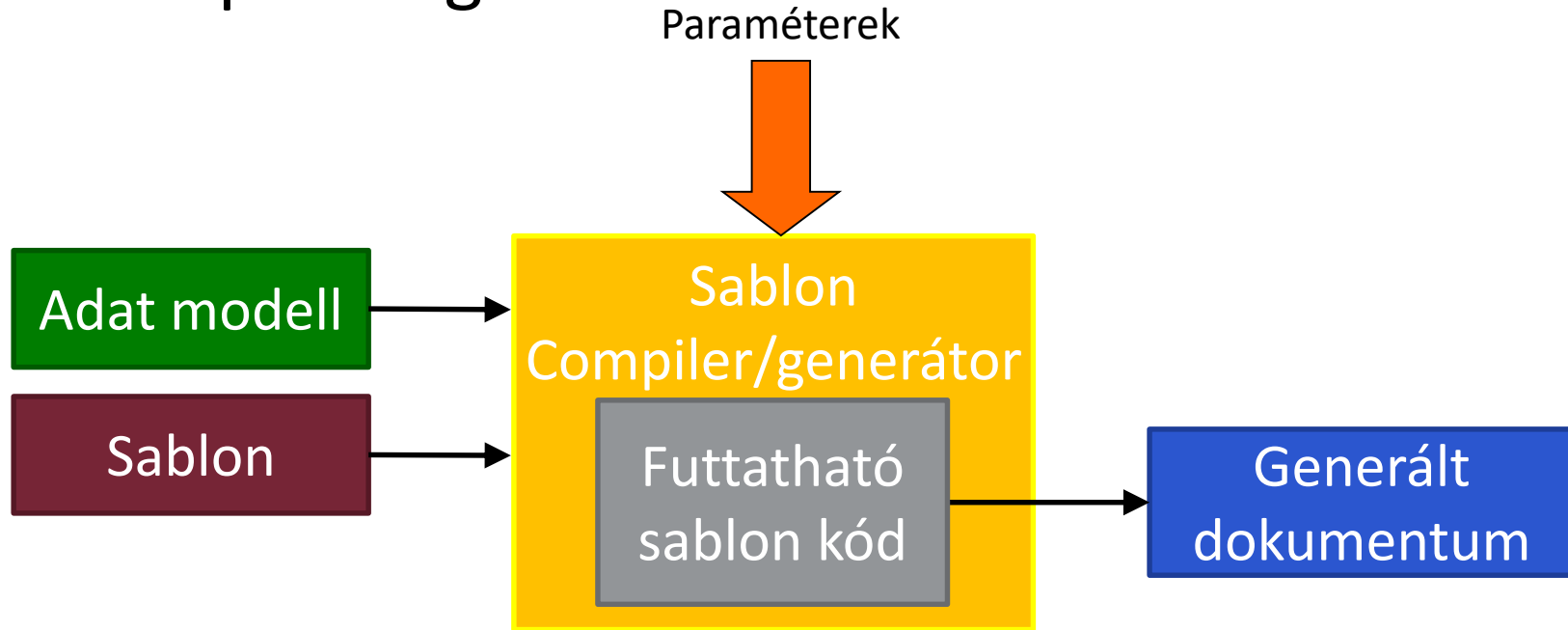
# Kódgenerátor típusok II.

## ■ Sablon-alapú kódgenerátor



# Kódgenerátor típusok II.

## ■ Sablon-alapú kódgenerátor



- Gyorsabb fejlesztés
- Komplex változások az életciklus során
  - A sablon és a modell függetlenül változik

# Funkciók

- Hatékonyság vs. újrafelhasználhatóság, érhetőség
  - Nyomonkövethetőség
    - Adott kódrészlet mely modellelem miatt van ott?
    - *Inkrementalitás* támogatása: csak ott változzon a kód, ahol a modell
- Hatékony újragenerálás

# Generált kód illesztése

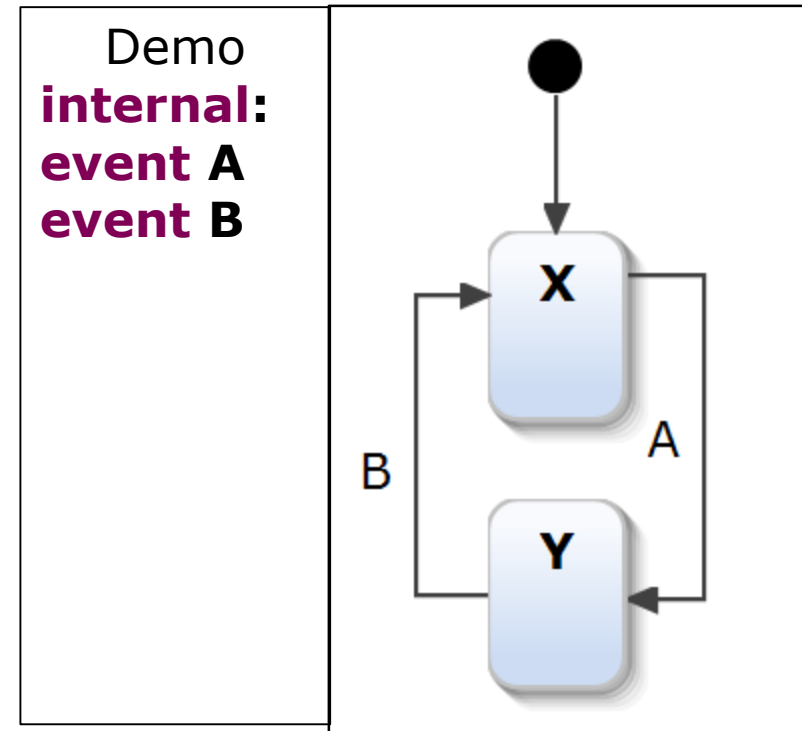
- A modellből generált kód csak egy része az alkalmazásnak. *Hogyan illesszük a többihez?*
  - Generált kódot kézzel nem módosítunk  
→ bármikor újragenerálódhat!
  - Modellben/sablonban növeli a komplexitást  
*Nem oda való...*
- Generált és kézzel írt kód külön fájlban/mappában
- Illesztés: *ragasztó kód* (glue code)
  - Generált kód hívása
  - Leszármazás generált kódból

# KÓDGENERÁLÁS PÉLDA



# Kódgenerátor példán keresztül:

- **Feladat:**  
Generáljunk C kódot  
Yakindu állapotgépekből
- Írjunk olyan függvényt:  
→ kap egy Modell objektumot  
← visszaad egy szöveget
- A szöveg egy „Demo.c” fájlba kerül
- Fordító lefordítja



# Sablon alapú kódgenerátor (Xtend)

- Cél: Állapotok → Enum

Összevágunk egy char\*-ba a kimenetet,  
%s helyére írjuk az X,Y neveket

- Megoldás: C program

```
sprintf(result,  
"enum states {\n\tState%s, \n\tState%s\n};",  
state1->name,  
state2->name);
```

```
enum states {  
    StateX,  
    StateY  
};
```

😊 Ez egyből működik!  
☹ nehezen átlátható.

- Sablon (Xtend):

```
'''  
enum states {  
    State«state1.name»,  
    State«state2.name»  
}'''
```

Kiemeljük (*escape*) a változások helyét

Megírjuk „sablonosan”

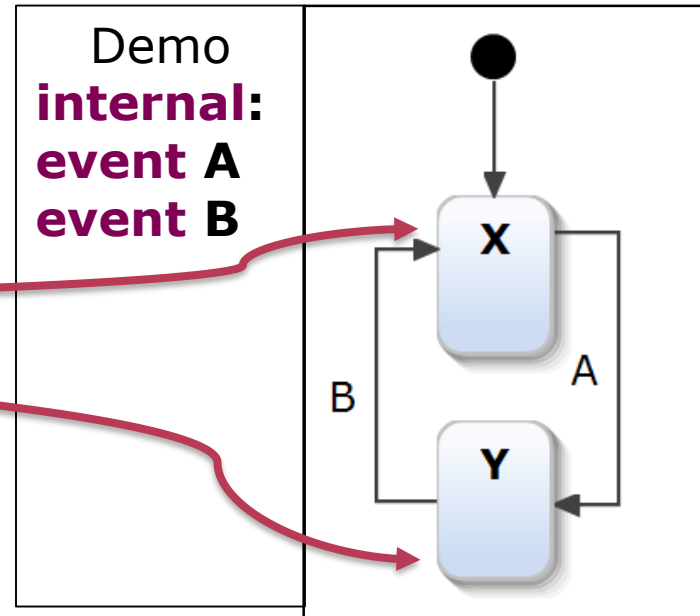
😊 Könnyebb írni  
😊 Átláthatóbb  
😊 Könnyű módosítani  
☹ +1 technológia

# Kódgenerátor példa: Állapotok

## ■ Várt C kód:

```
//States of the statemachine  
enum states {  
    StateX,  
    StateY  
};
```

Lehetséges állapotok:  
Enumként felsorolva



## ■ Sablon:

```
//States of the statemachine  
enum states {  
«FOR state : states  
    SEPARATOR ', '»  
    State«state.name»  
«ENDFOR»  
};
```

1.Összes állapoton végigmegyünk  
2.Vesszővel elválasztva írjuk ki:

State«név»  
Pl: StateX, StateY

# Kódgenerátor példa: kezdőállapot

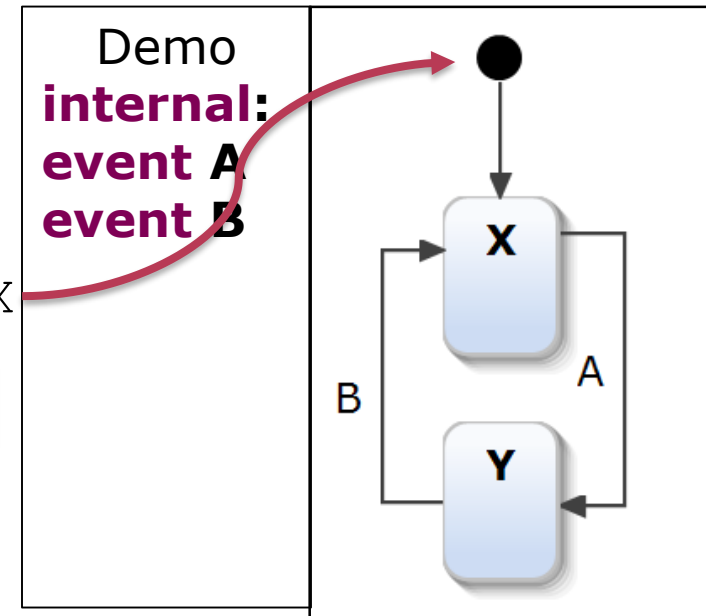
## ■ Várt C kód:

```
// The actual state
// First = initial state.
enum states currentState = StateX
```

Aktuális állapot = kezdőállapot

## ■ Sablon:

```
// The current state.
// Initial value = the entry state.
enum states currentState = State«findEntry(states).name»
```



1. Megkeressük a kezdőelemet
2. Kiírjuk a nevét

# Kódgenerátor példa: Állapotátmenet

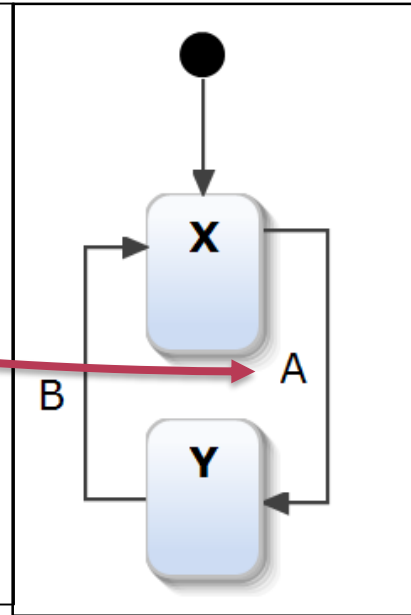
## ■ Várt C kód:

```
// Execute "A" event  
void doA () {  
    switch (currentState) {  
        case StateX:  
            currentState = StateY;  
            break;  
        case StateY:  
            break;  
    }  
}
```

A / X → Y

A / -

Demo  
**internal:**  
event A  
event B



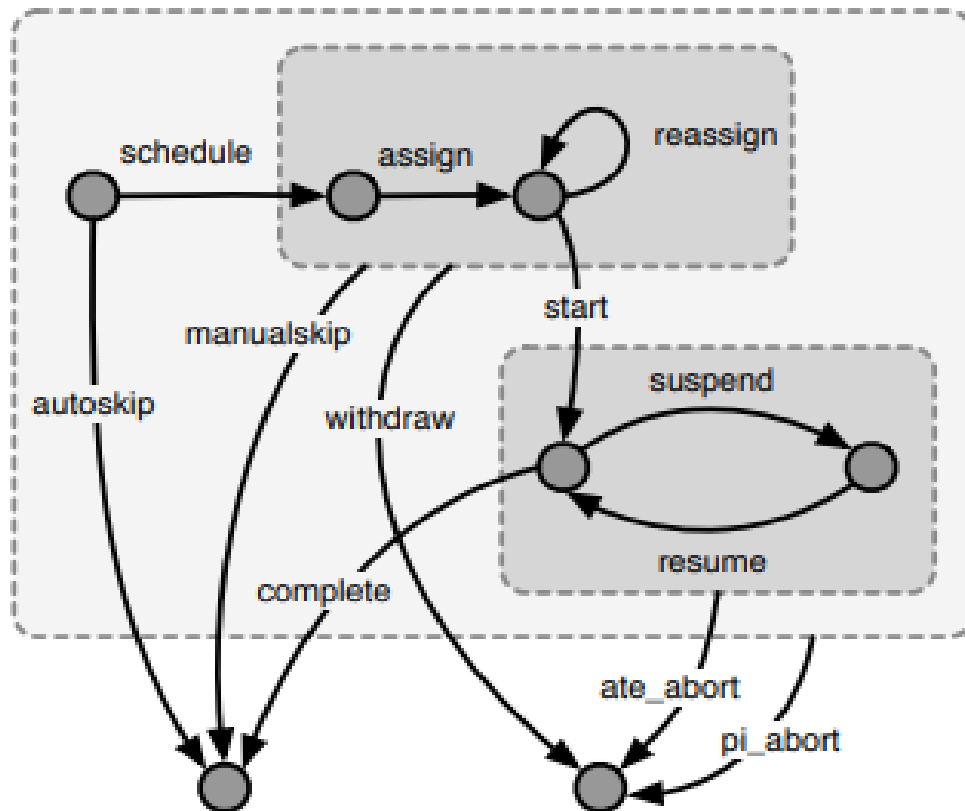
## ■ Sablon (vázlatosan):

1. Minden eseményhez egy `do«Esemény neve»` függvény
2. A tranzíciók alapján mást csinál a függvény

Egy (egyszerű) állapotgéphez ennyi a kódgenerátor!

# Szimuláció példa

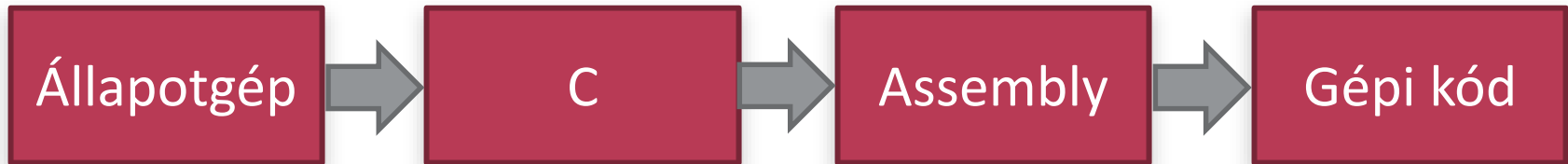
- Példa: szimulációs események előállítás (BIMP, MXML)



IEEE XES szabvány események tárolására

# Kódgenerátor Összefoglalás

- Kódgenerálás = Fordító
- Ugyanaz a lépés:

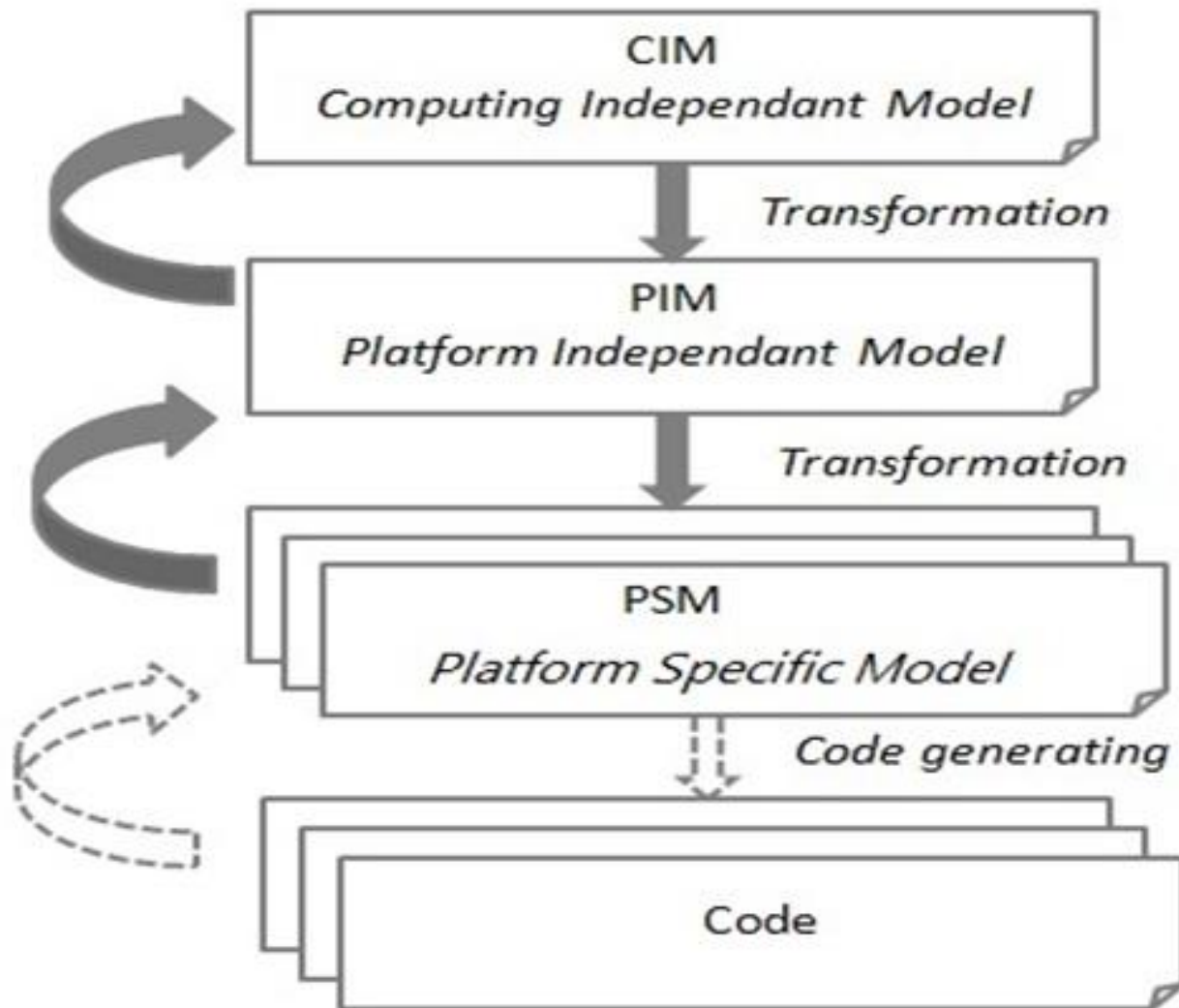


Jóslat:

tegnap tervezési minta →  
ma kódgenerátor →  
holnap nyelvi elem

- A problémát a saját nyelvén: Produktivitás ++
- Sok unalmas, bonyolult kód automatikusan Teljesítmény ++
- Ellenőrizzük a saját nyelvén: Megbízhatóság ++
- Tanszékünkön fejlesztett projektek: akár **95%** generált kód

# Model Driven Architecture





# ÜZLETI FOLYAMATOK VÉGREHAJTÁSA BLOCKCHAIN ALAPON

Contract-Based Business Process Execution. Attila Klenik, Hyperledger Internship of Linux Foundation. (Mentor: László Gönczy)

<https://hgf18.sched.com/event/G8sC/business-process-support-for-consortial-blockchains-laszlo-gonczy-budapest-university-of-technology-and-economics-bme>

<https://inf.mit.bme.hu/news/2017/03/linux-foundation-t%C3%A1mogatja-tansz%C3%A9ki-kutat%C3%B3munk%C3%A1t>

# (Privát) Blockchain technológia

- Röviden:
  - Elosztott
  - Letagadhatatlan
  - Többrésztvevős
  - Tranzakciók
- Felhasználási területek
  - (kripto)valuták
  - Árucseré
  - Szállítmányok követése



# Motiváció

- Üzleti folyamatok elemei
  - Együttműködési minták magas szintű leírása
  - Több résztvevővel
  - Központilag végrehajtva ☹️
- (Nagyvállalati) Blockchain célja
  - Tranzakciók kezelése
  - „Okos szerződések” alapján
  - Több résztvevővel
  - Robosztus, elosztott módon 😊

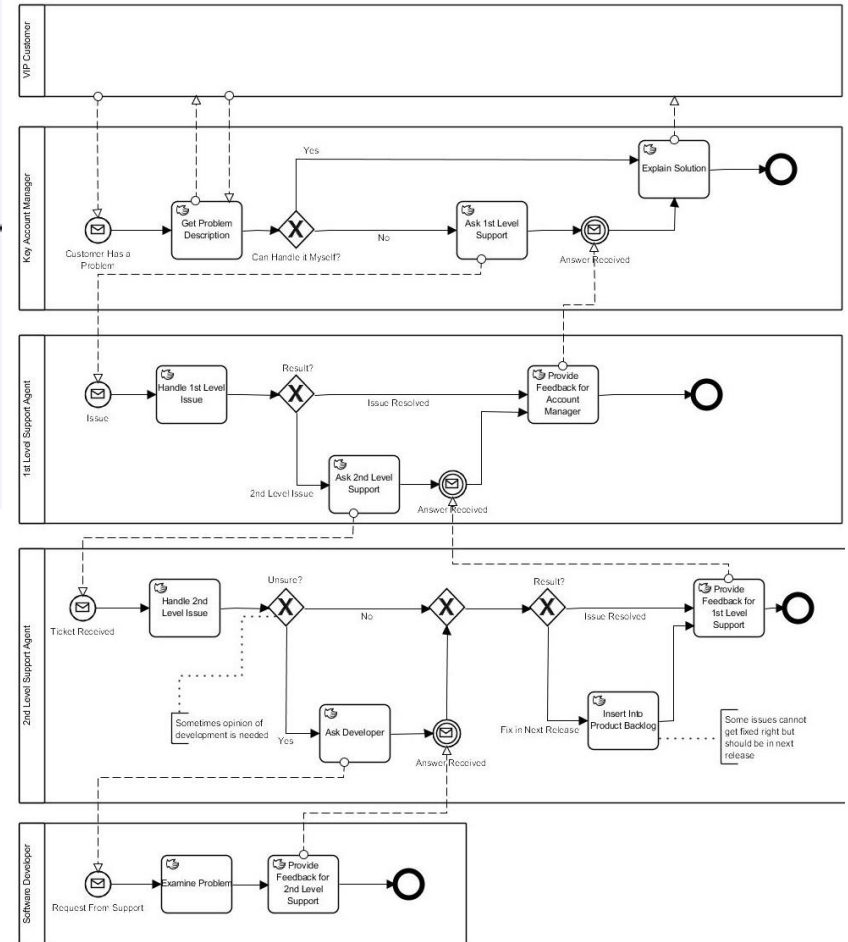
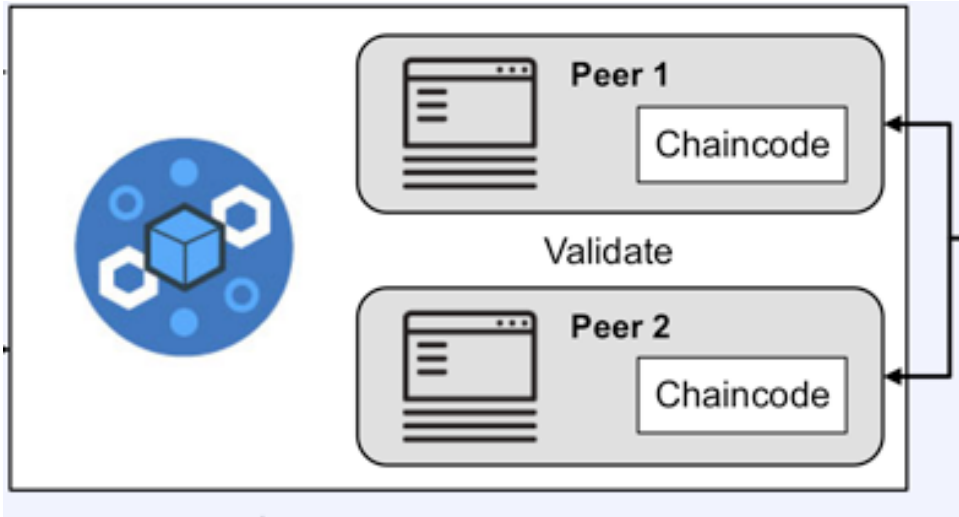
# Motiváció

- Modell alapú fejlesztés
  - Magasabb absztrakciós szint
  - Produktivitás
    - Modellek (szerződés minták) újrahasznosítása
    - Tervezés egyszerűsítése
  - Automatizált fejlesztés
  - Minőségbiztosítás
    - Modell validáció, modellellenőrzés, modell alapú tesztelés
- „Blockchainesítés”
  - Meglévő megoldások (részben) blockchainre vitele

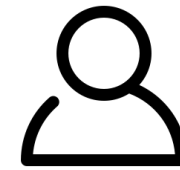
# Kitekintés

- Mi történik ellenőrizetlen szerződések esetén?
- Pl. „végtelen” token hozható létre
  - <https://medium.com/coinmonks/not-smart-contracts-ethereum-erc-20-tokens-flaw-f1d82545b4b>
- Rengeteg hibás/sebezhető szerződés „él”
  - <https://arxiv.org/pdf/1802.06038.pdf>

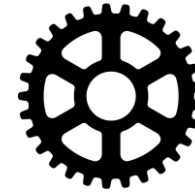
# Okos szerződések okosan



# BPMN elemek



Résztevők



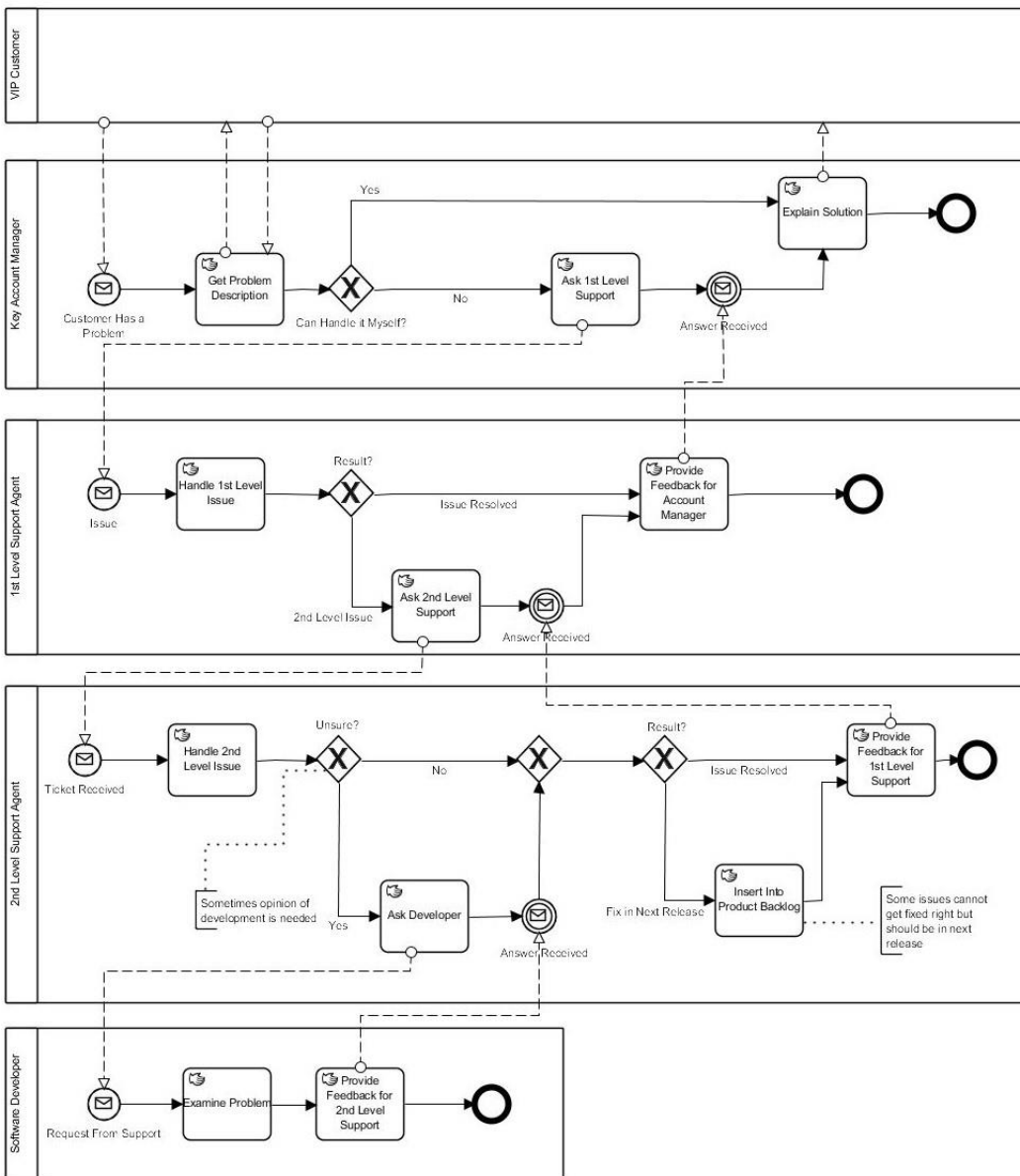
Tevékenységek



Adat



Vezérlés



# BPMN Elements in Blockchain Platforms



Résztevők



„Szervezetek”



Tevékenységek



Tranzakciók  
blockchain felett



Adat



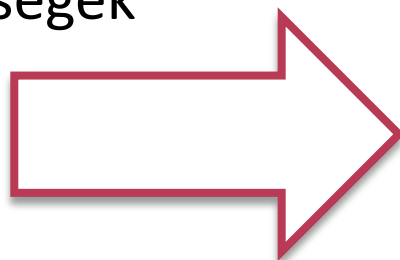
Adatmodell az okos  
szerződéshez



Vezérlés



Feltételek (korlátok)  
az okos szerződésben





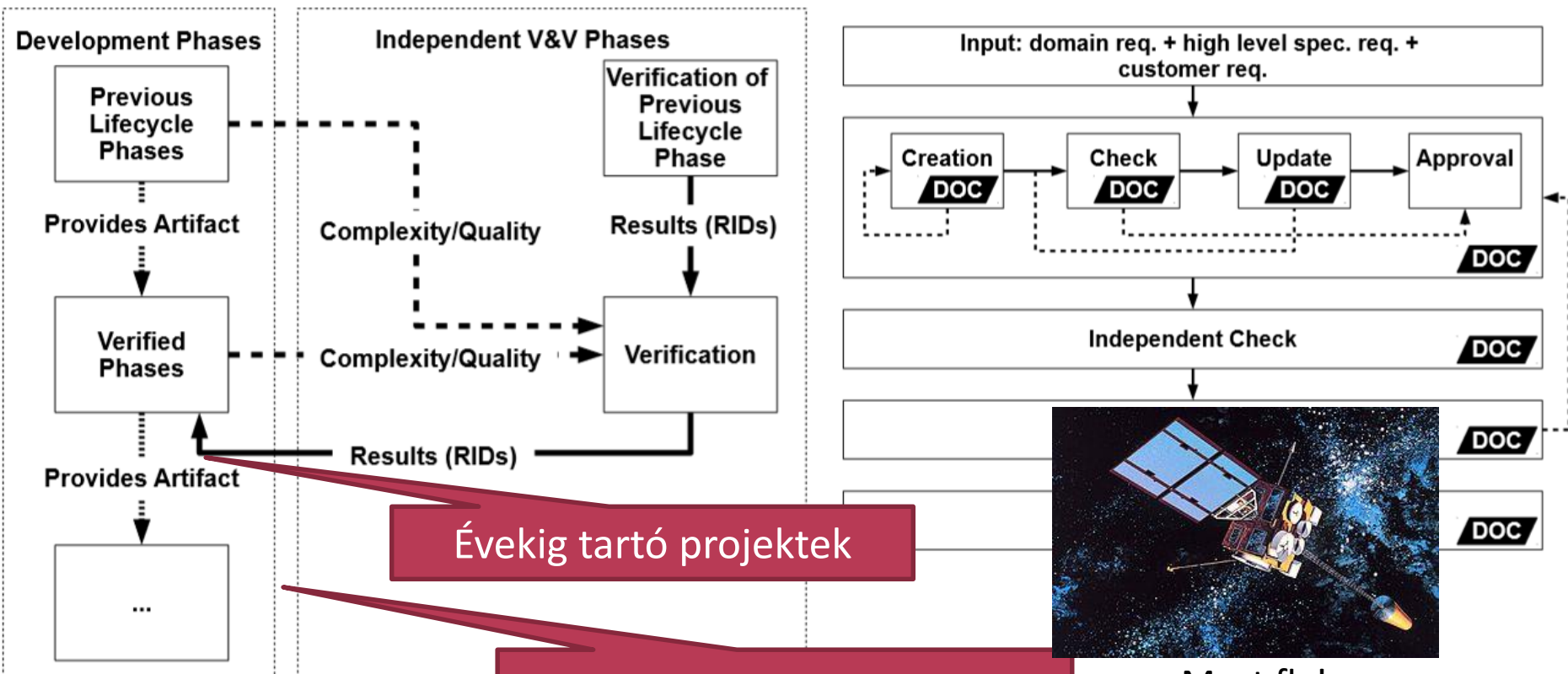
# Célok

- Üzleti folyamatok alapján okos szerződések generálása
  - Nem feltétlen a végrehajtó környezet szintjén
  - Nehezen ellenőrizhető
  - A kód és a specifikáció összerendelése
- A fejlesztés részleges automatizálása
  - „boilerplate” kód generálása
  - Hibalehetőségek számának csökkentése
  - A vezérlési logika előállítás
    - Üzleti logika váza
    - Meglévő lépések becsatolása?

# Használati esetek

- Biztosítás, pénzügyek, ...
- Közművek, energiakereskedelem/átvitel
- Logisztika/közlekedés
- **Beszállítói láncok**
- Egészségügy
- **Tanúsítványok**
- ...szabványok határozzák meg

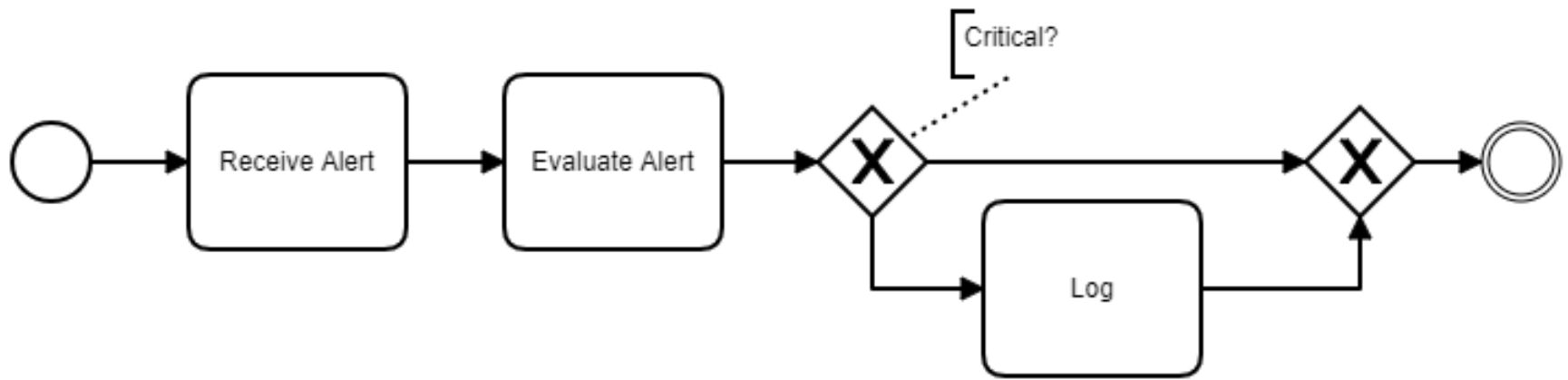
# Kritikus rendszer tanúsítása



Évekig tartó projektek

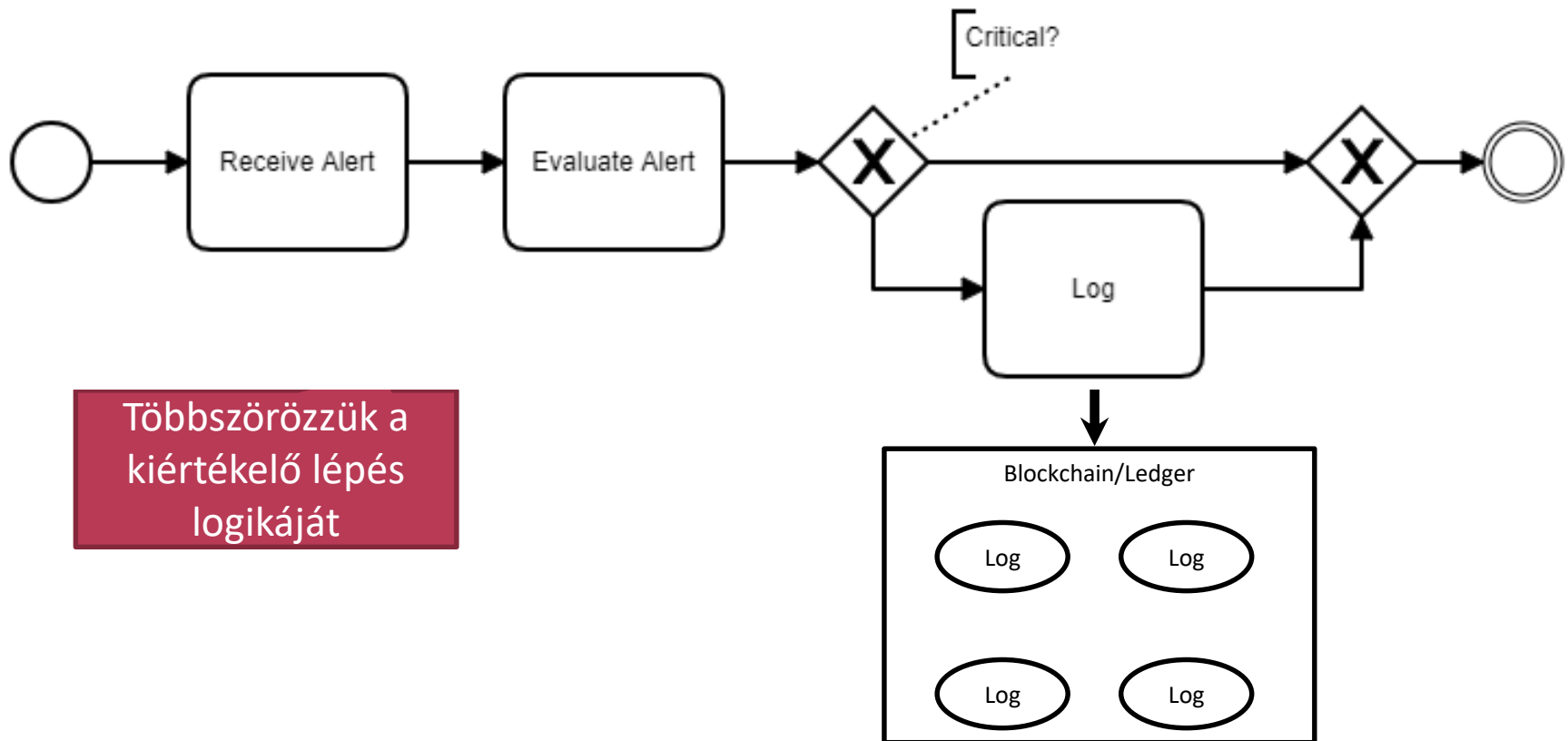
Dokumentumok, szereplők,  
stb változnak

# 0. szint: eredeti folyamat



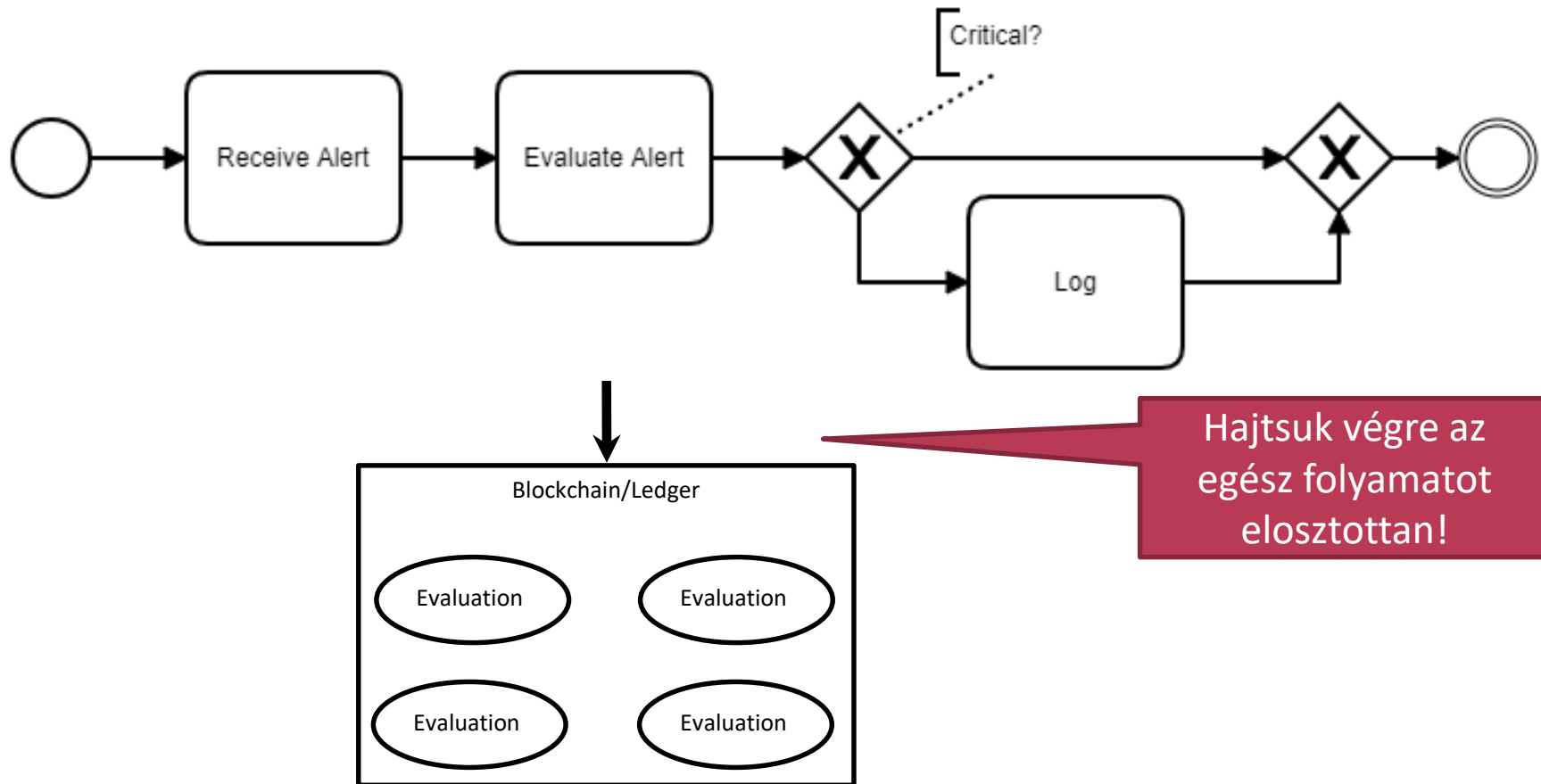
... többszörözzük a logokat

# 1. szint: adat blokkláncon



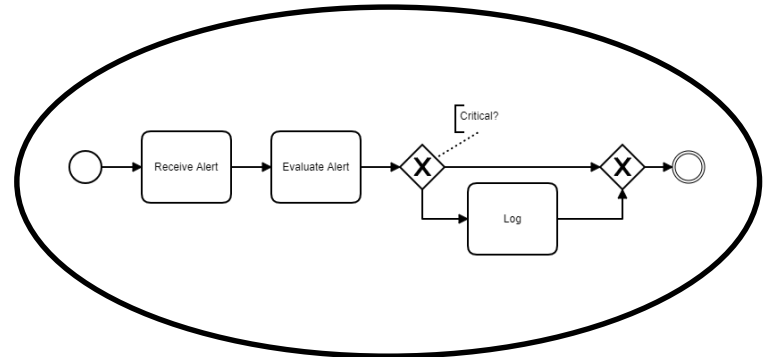
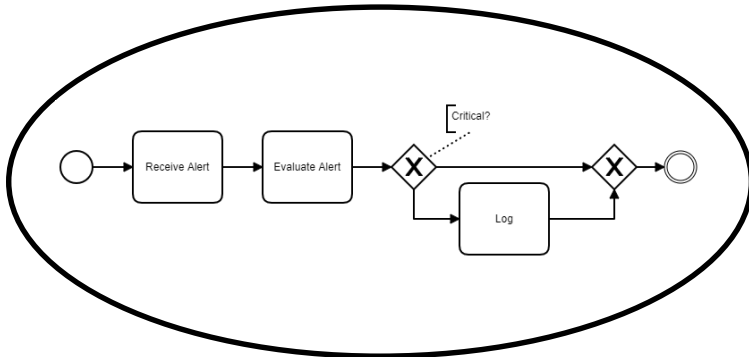
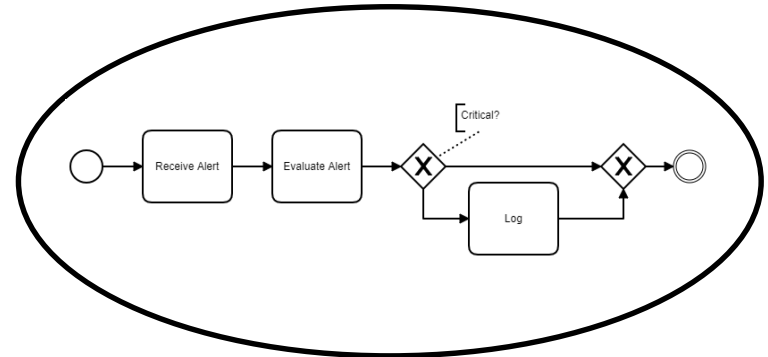
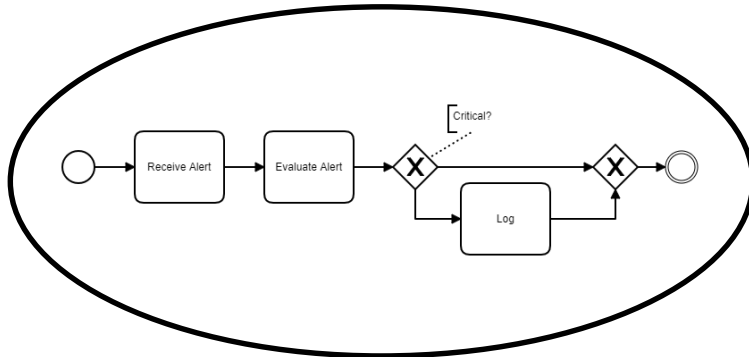
Többszörözzük a kiértékelő lépés logikáját

# 2. szint: döntési logika blokkáncon



# 3. szint: Üzleti folyamat blokkláncon

Blockchain/Ledger



# BPMN2FABRIC

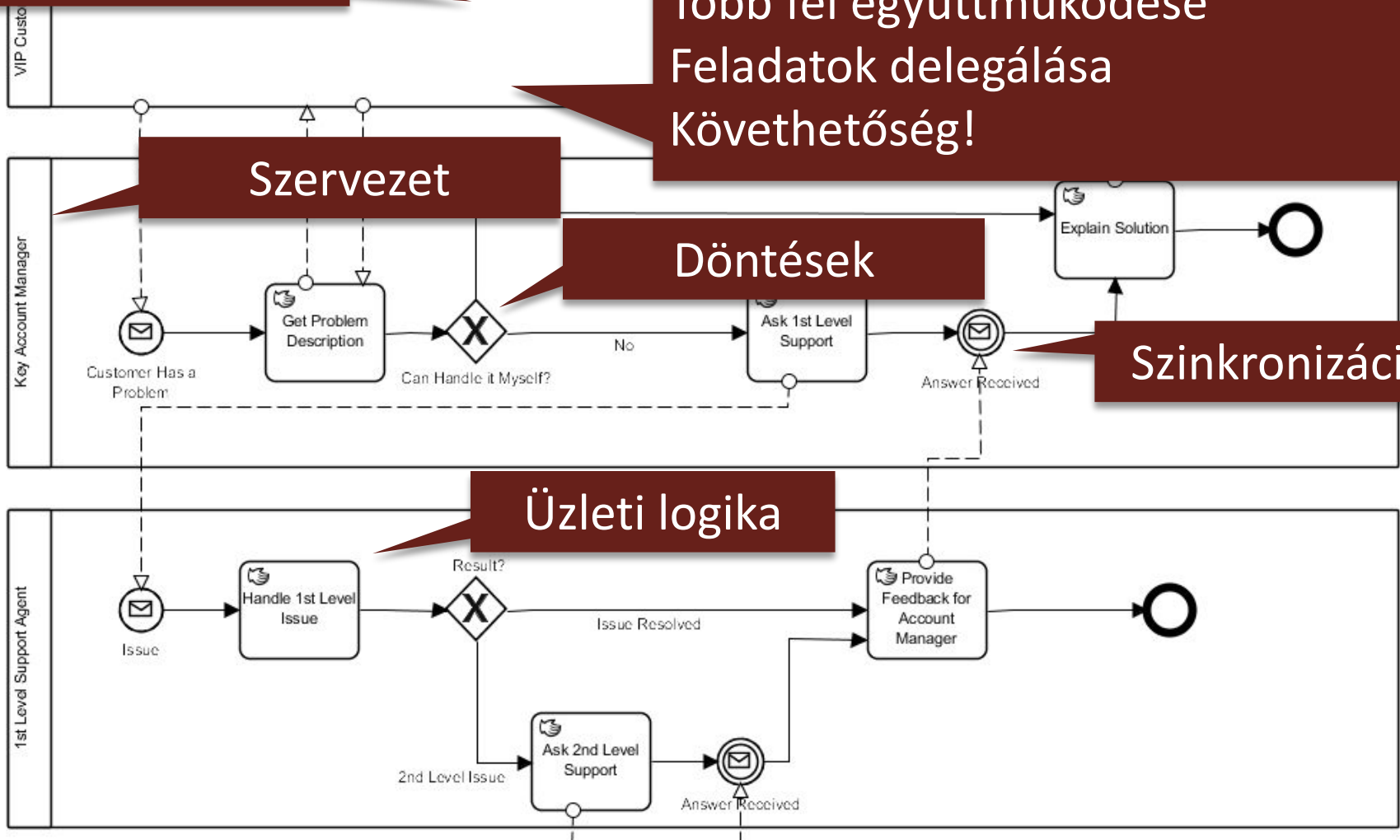
*Collaboration Orchestration in Hyperledger Fabric with BPMN Processes.*  
Attila Klenik, László Gönczy, András Pataricza, Imre Kocsis@ICACON2018



# Incidens menedzsment példa

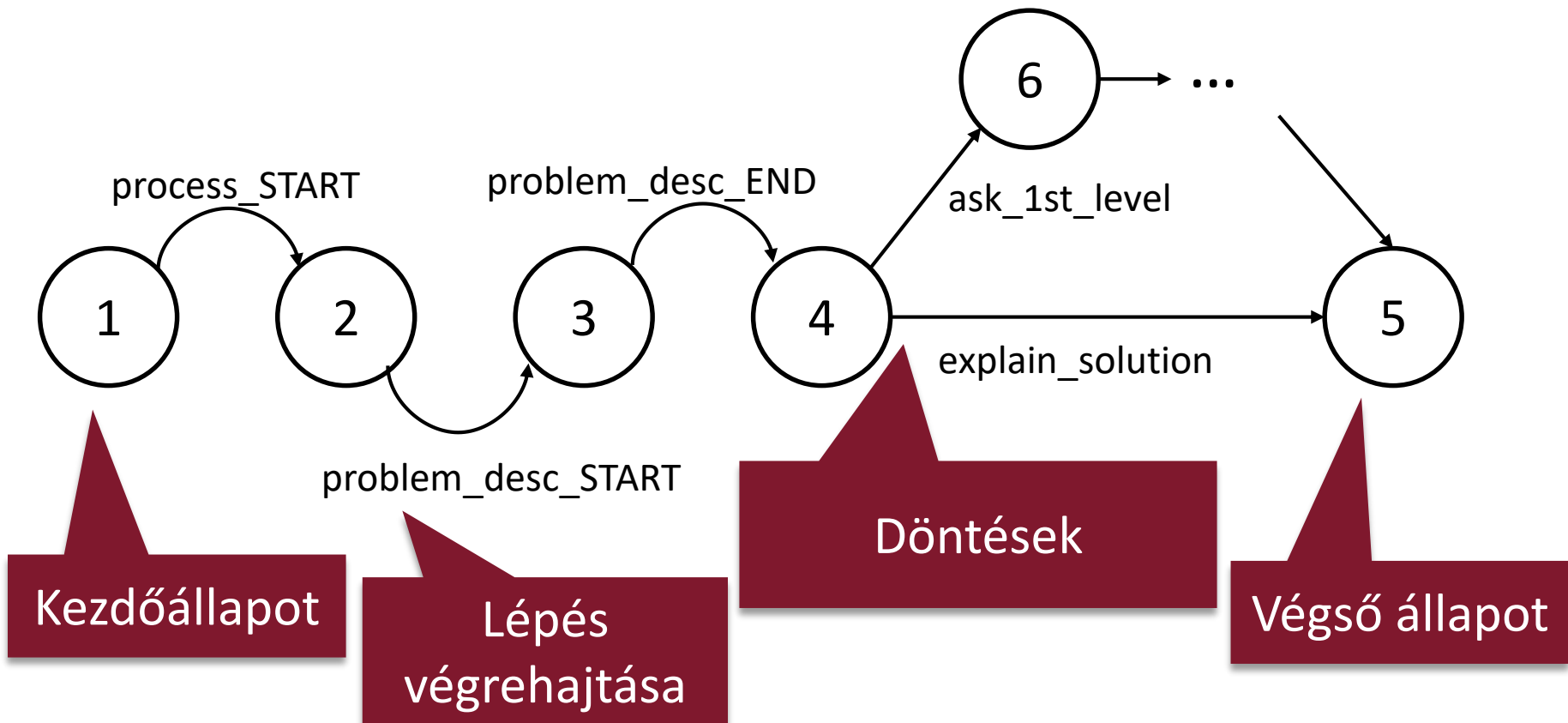
Hálózaton kívül

Több fél együttműködése  
Feladatok delegálása  
Követhetőség!



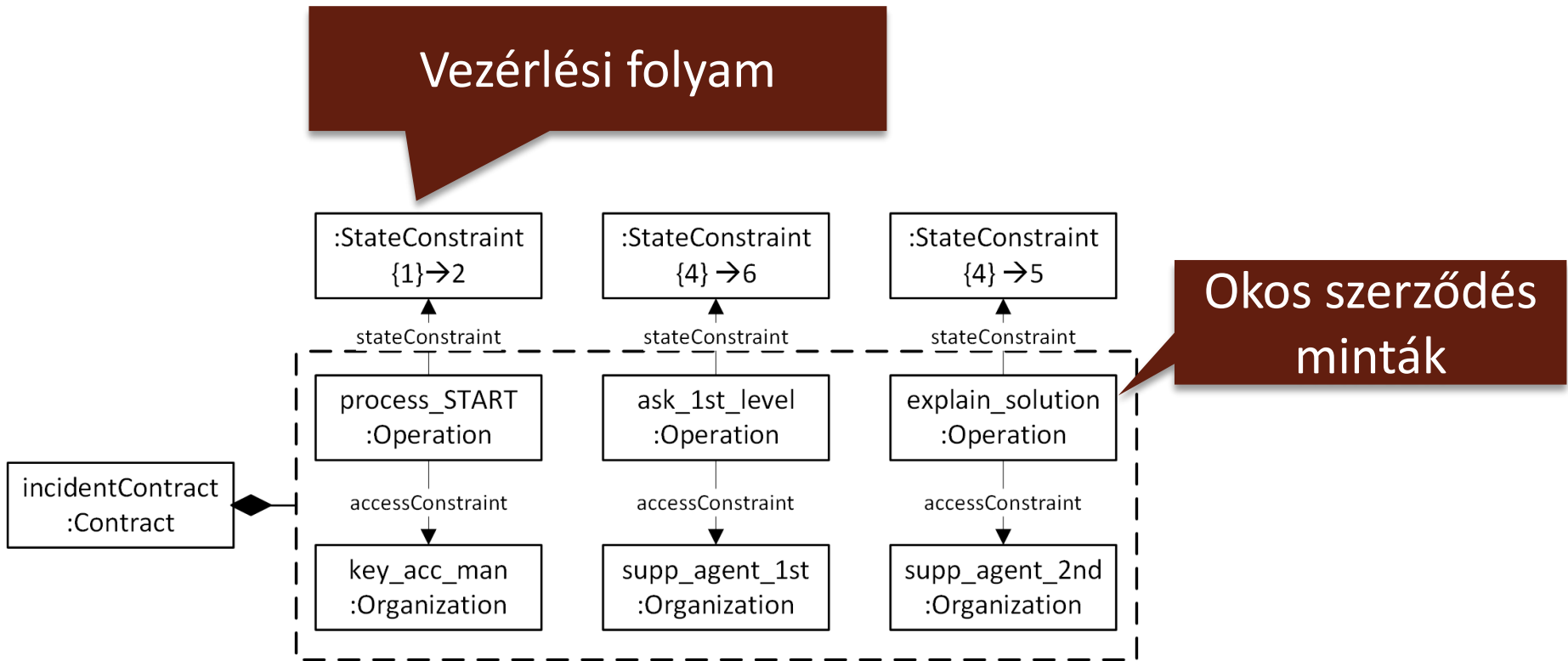
<https://www.businessprocessincubator.com/content/incident-management-as-detailed-collaboration/>

# Dinamikus viselkedés: állapotmodell



# Platformfüggetlen példánymodell

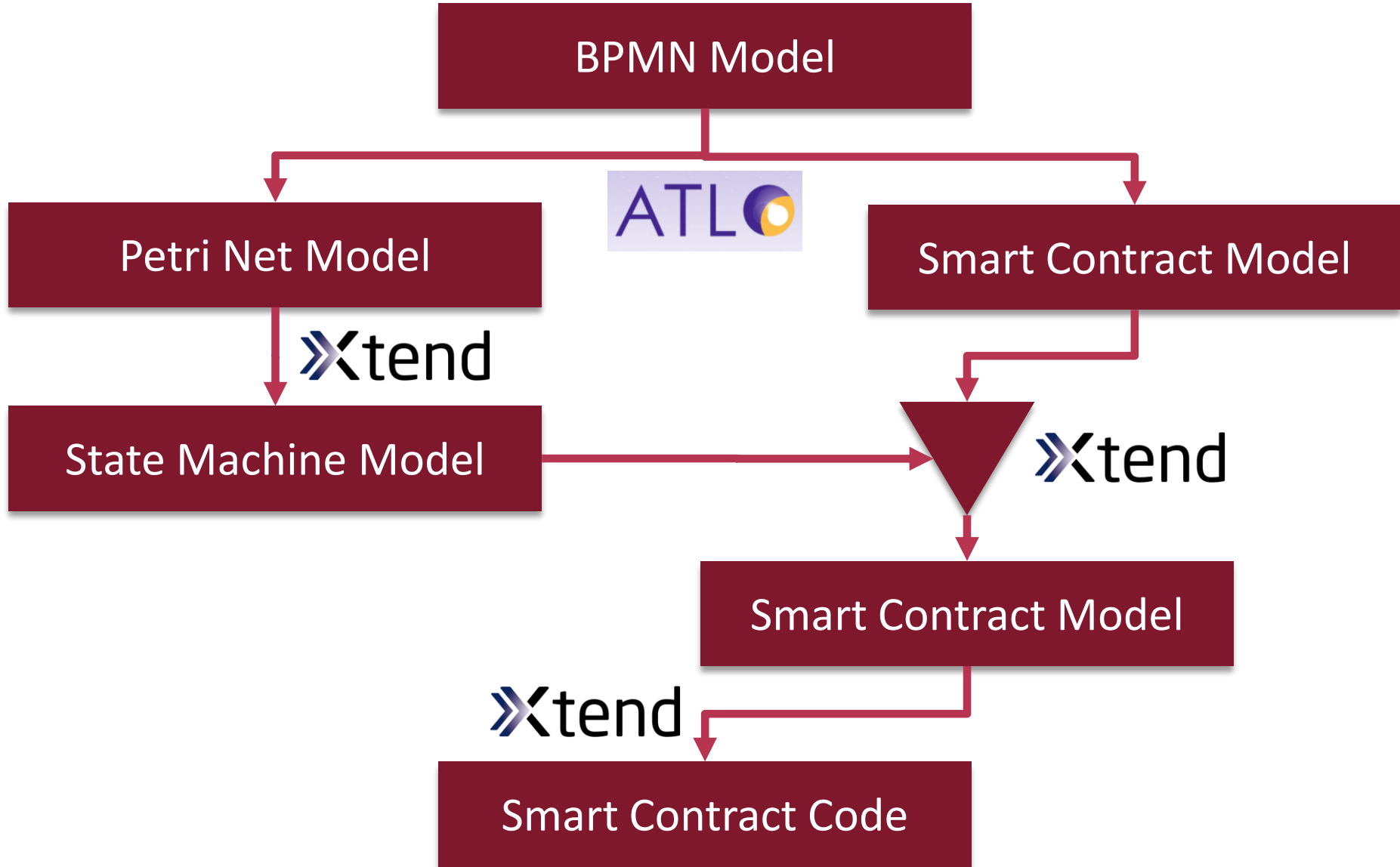
Vezérlési folyamat



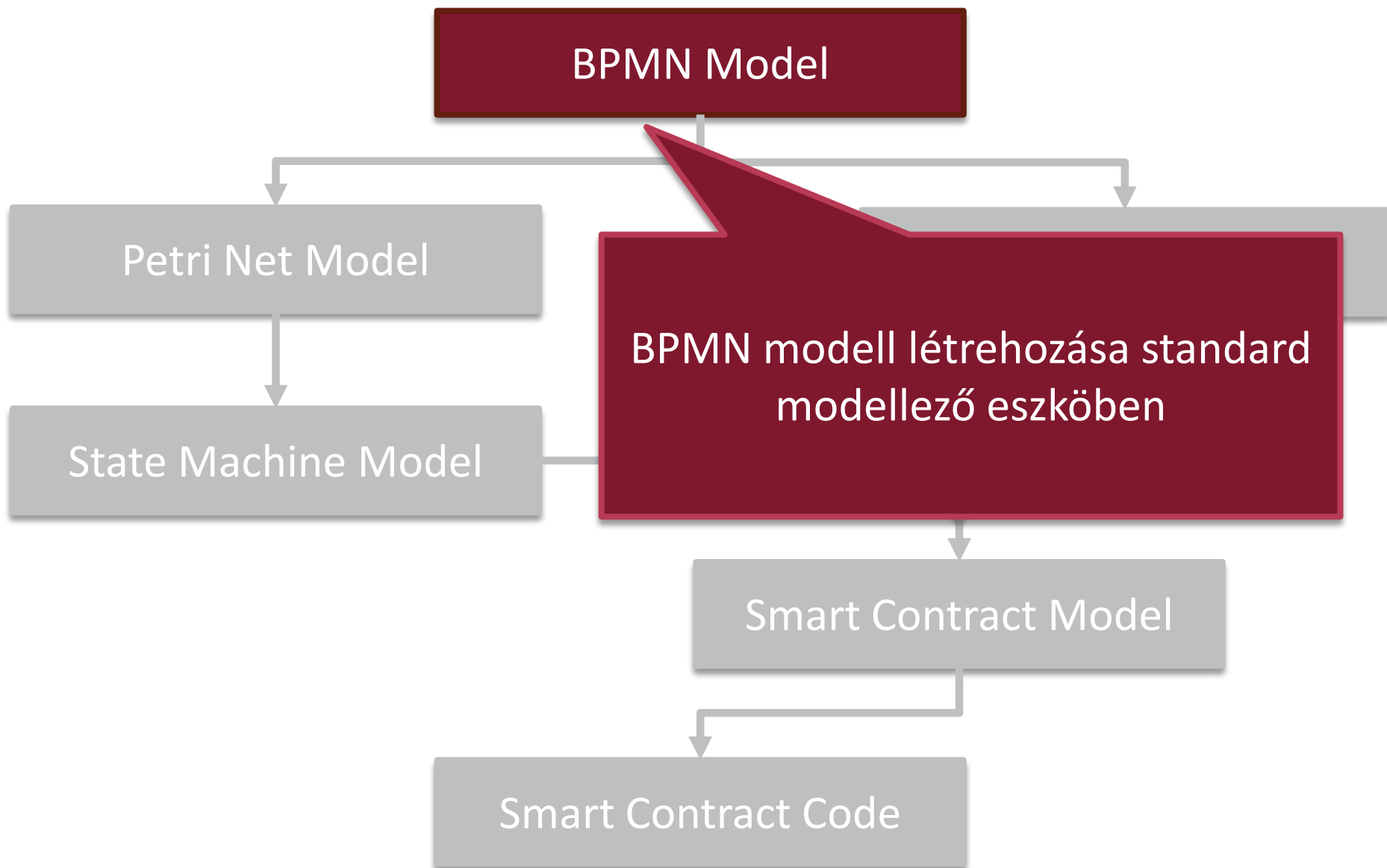
Okos szerződés minták

Okos szerződés szintű hozzáférésvédelem

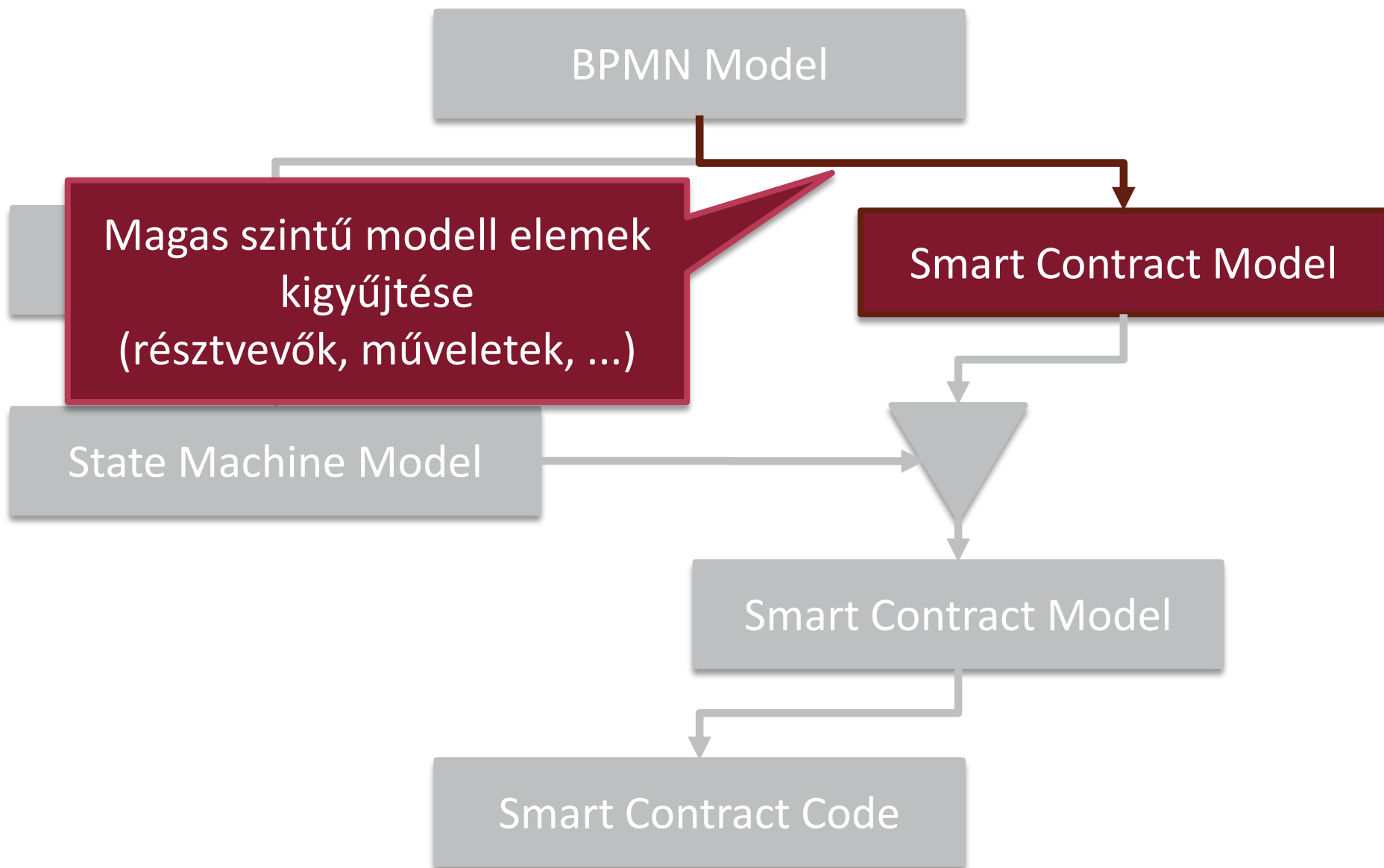
# Transzformációs folyamat



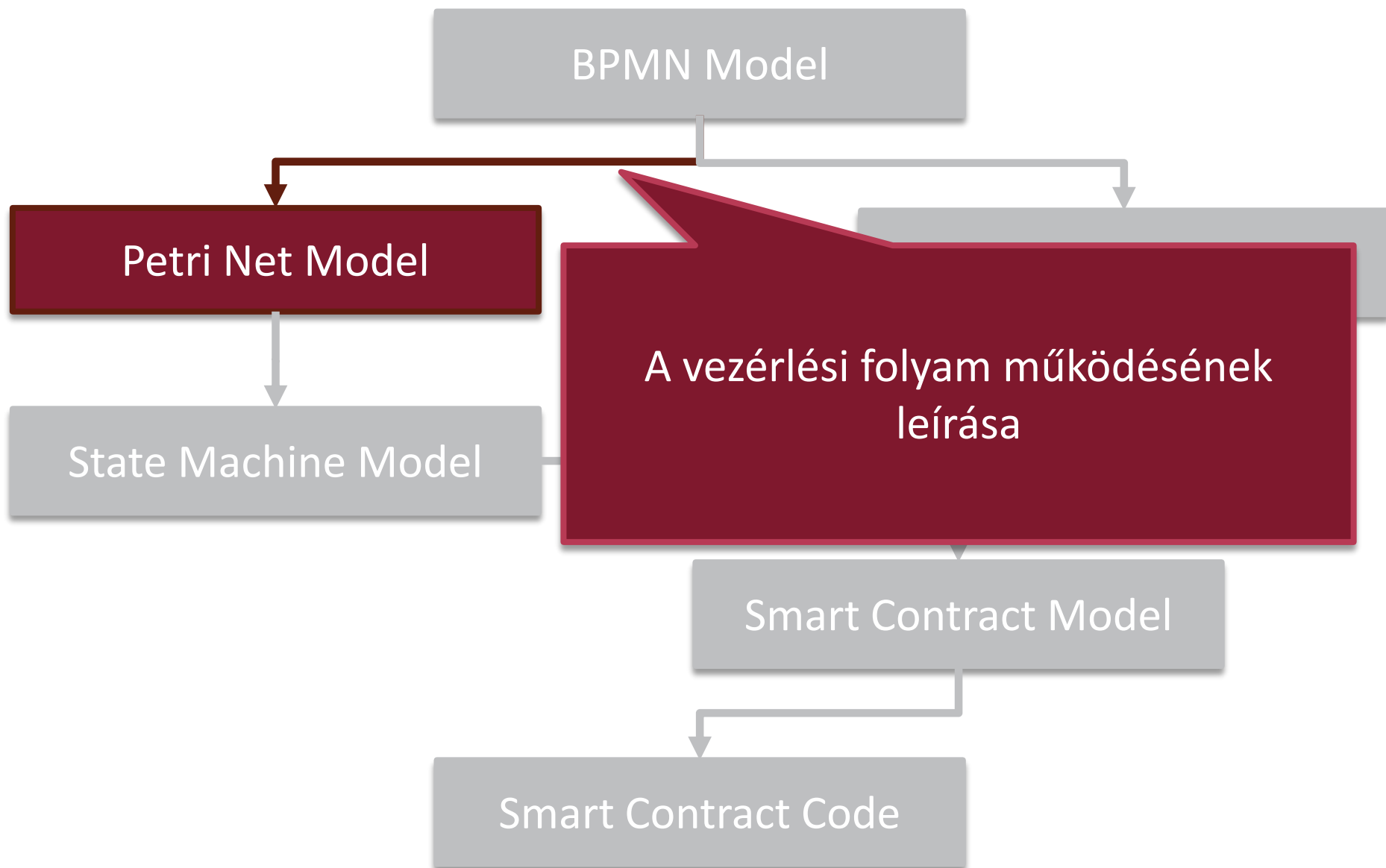
# Transzformációs folyamat



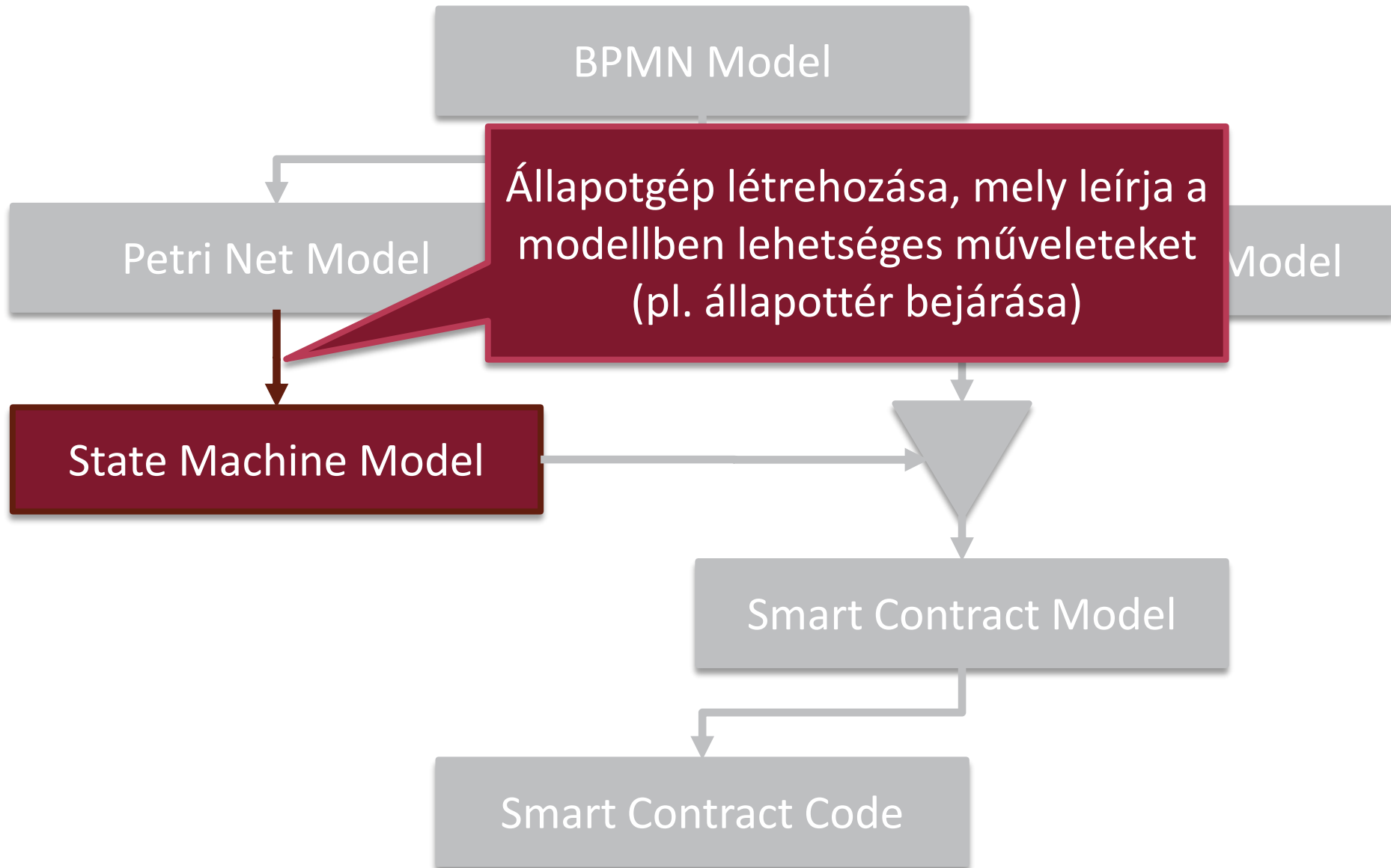
# Transzformációs folyamat



# Transzformációs folyamat

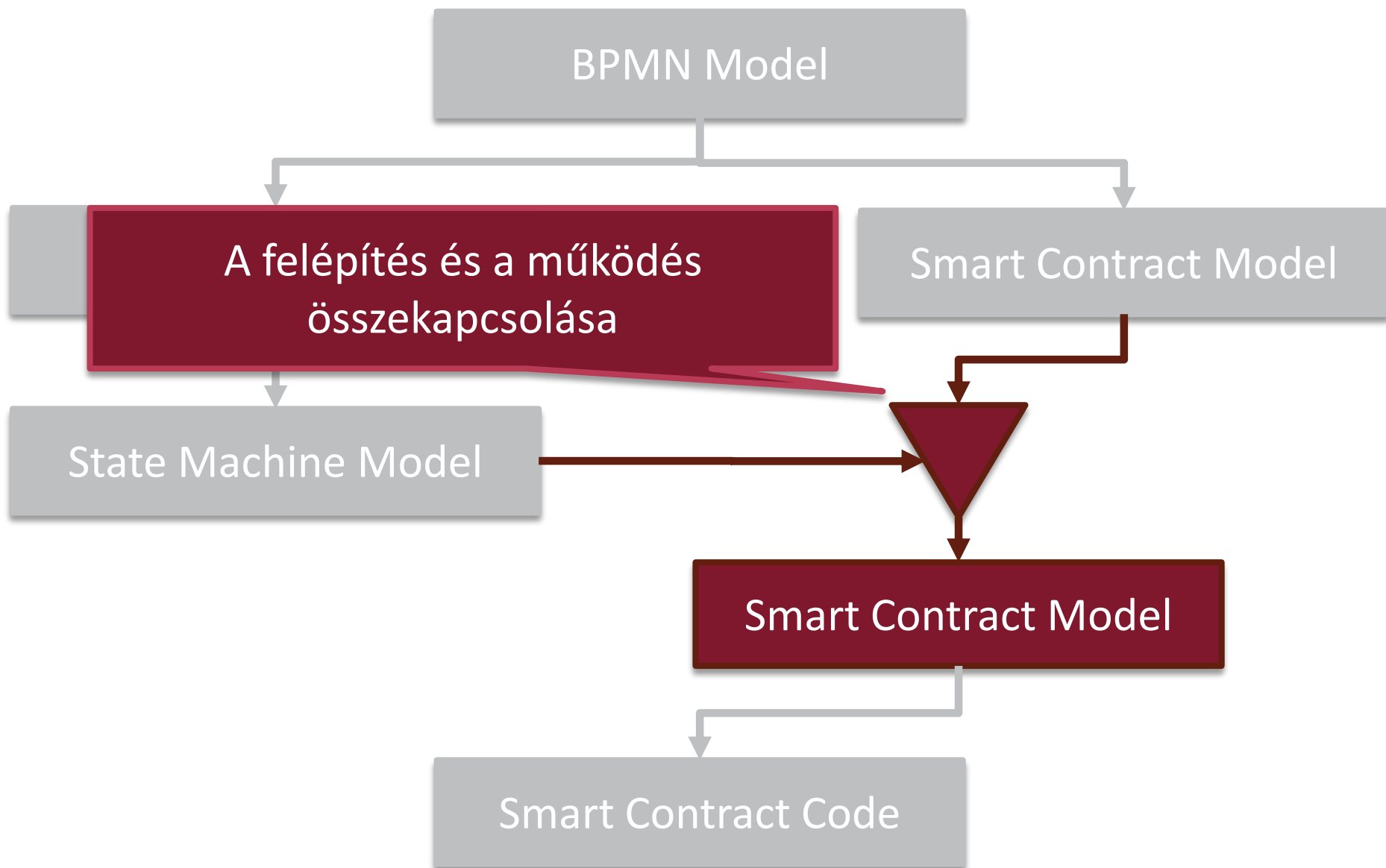


# Transzformációs folyamat





# Transzformációs folyamat



# Transzformációs folyamat

