

Magasabb szintű formalizmus: Állapottérképek (statecharts)

dr. Majzik István

BME Méréstechnika és Információs
Rendszerek Tanszék

Modellek a formális ellenőrzéshez

Mivel nyújt
többet egy
magasabb szintű
formalizmus?

**Mérnöki
modellek**

**Magasabb szintű
formalizmusok**
SC, PN, CPN, DFN

**Alapszintű matematikai
formalizmusok**
KS, LTS, KTS

**Modell-
transzformációk**



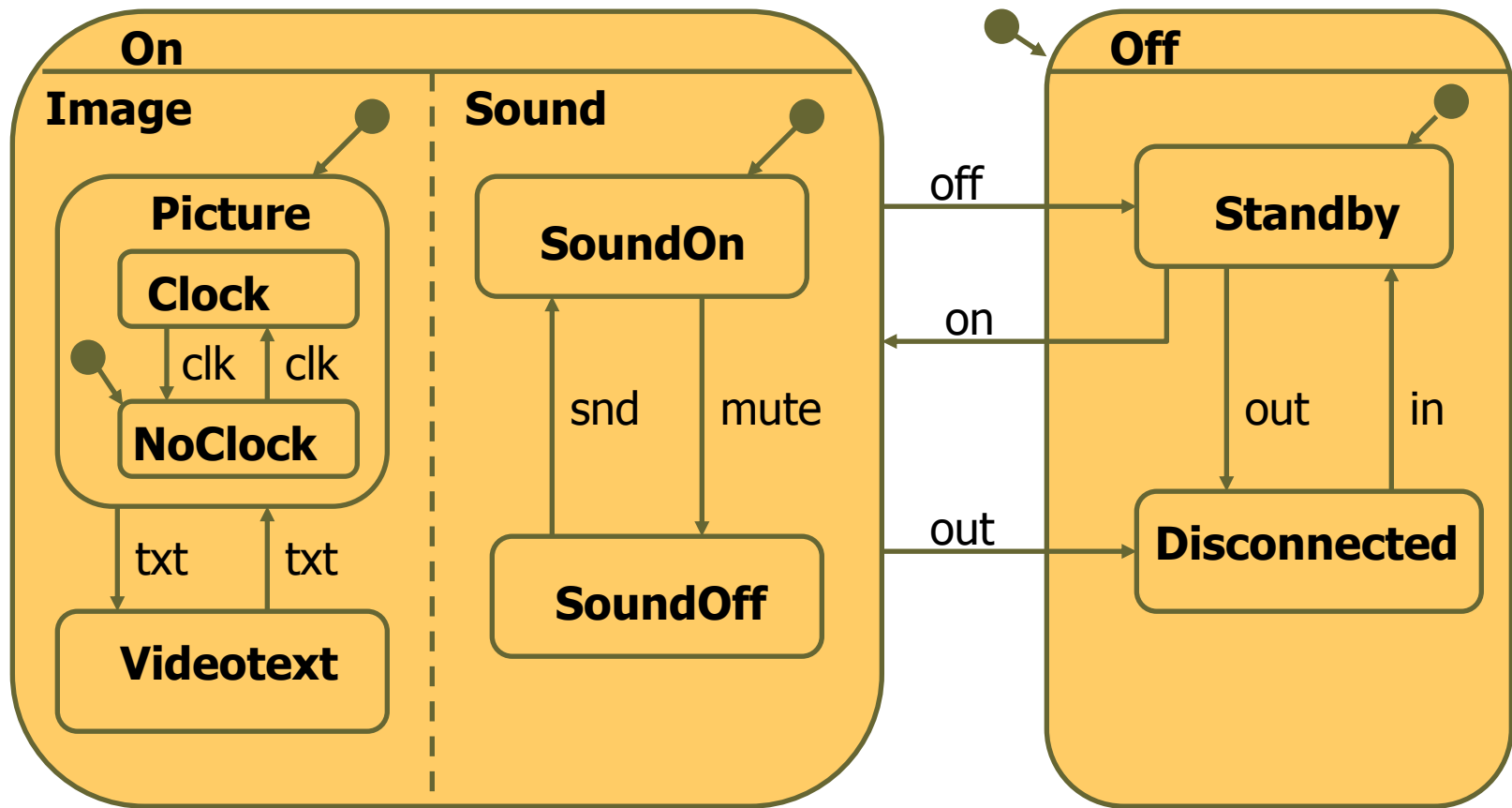
Mi az állapottérképek célja?

- **Állapot alapú, eseményvezérelt rendszerek viselkedésének megadására alkalmasak**
 - Egy állapotgép viselkedésének leírása
 - **Reaktív viselkedés**
 - Külső események (jel, üzenet) hatására történő állapotváltások
 - Akciók: az állapotátmenetekhez rendelt tevékenységek
 - Szokásos használat: beágyazott vezérlők, protokollok
- **Alapelemek**
 - **Állapot, aktív állapot:** Adott feltételek teljesülése
 - Pl. üzemmód, állapotváltozók értékei
 - **Állapotátmenet:** Állapot változása
 - Trigger esemény, őrfeltétel, akció tartozhat hozzá
 - **Esemény:** Aszinkron történés, paramétere is lehetnek
 - Önálló elem (eseményosztály példánya)

Bővítések a könnyebb használat érdekében

- **Állapotok finomítása: Állapothierarchia**
 - Szuperállapot: alállapotok közös tulajdonságaihoz
- **Konkurens viselkedés leírása**
 - Nem akarunk sorrendi kötöttséget megadni (egyidejűleg, vagy tetszőleges sorrendben végzett feldolgozás)
 - Többszálú / elosztott / párhuzamos végrehajtás esetén
- **Összetett állapotátmenetek**
 - Szétváló, egyesülő, feltételtől függő elágazó átmenet
- **Emlékezés: Visszatérés egy korábbi aktív állapotkonfigurációra**
 - Visszatérés adott feldolgozáshoz közbenső esemény után
 - Egy állapotfinomítási szinten vagy mélyebben is

Gyorsítósáv 1: Állapotfinomítás

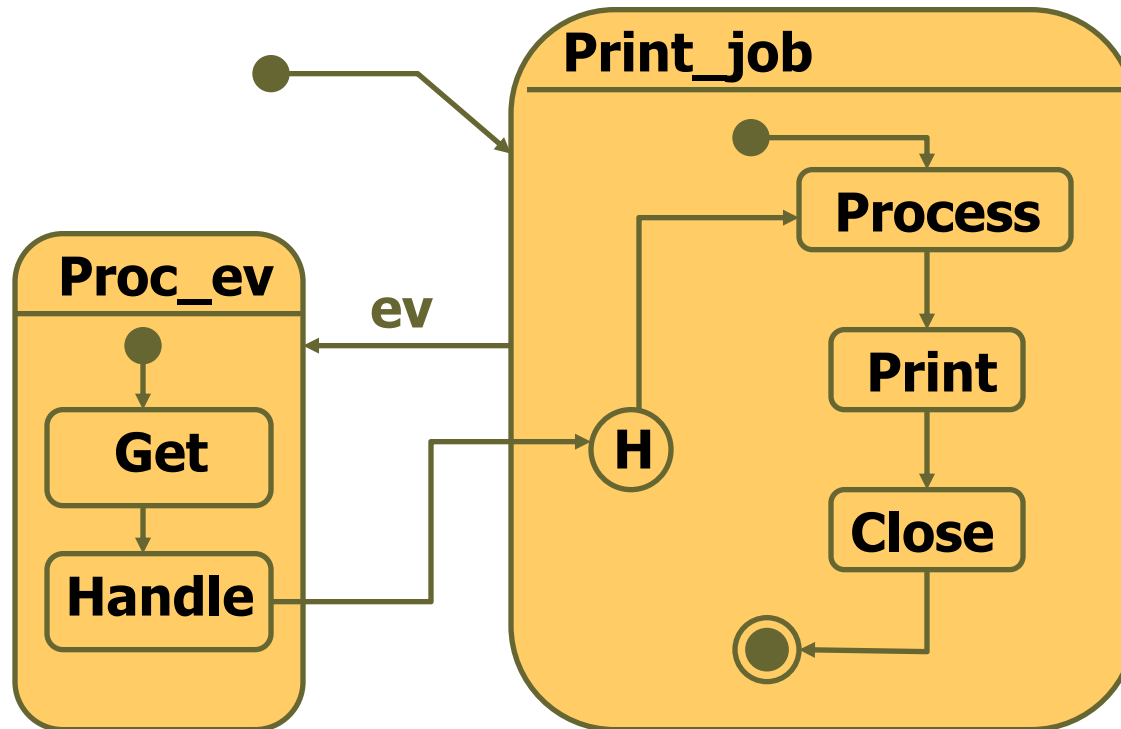


AND jellegű finomítás

OR jellegű finomítás

Belépési akció, kilépési akció és belső aktivitás is lehet

Gyorsítósáv 2: Emlékező állapot



Emlékező állapotok (history state):

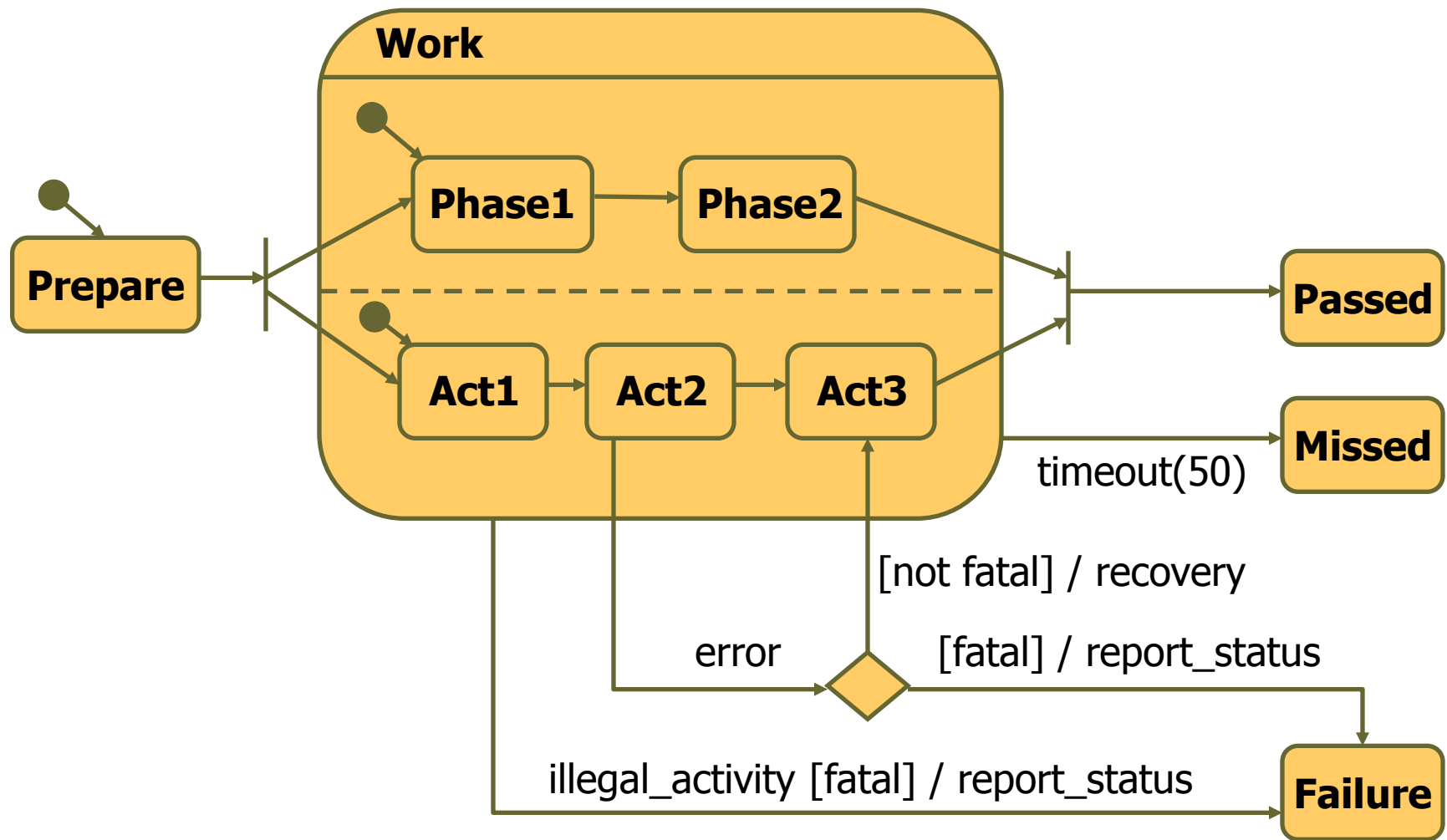
Az utolsó aktív állapotkonfigurációt „tárolja” a tartalmazó állapotban

- Egyszerű emlékező állapot: csak az adott finomítási szintet
- Mélyen emlékező állapot: a mélyebb finomítási szinteket is
- AND finomítás egy régiójában: Csak a régióra vonatkozik

H

H*

Gyorsítósáv 3: Állapotátmenetek

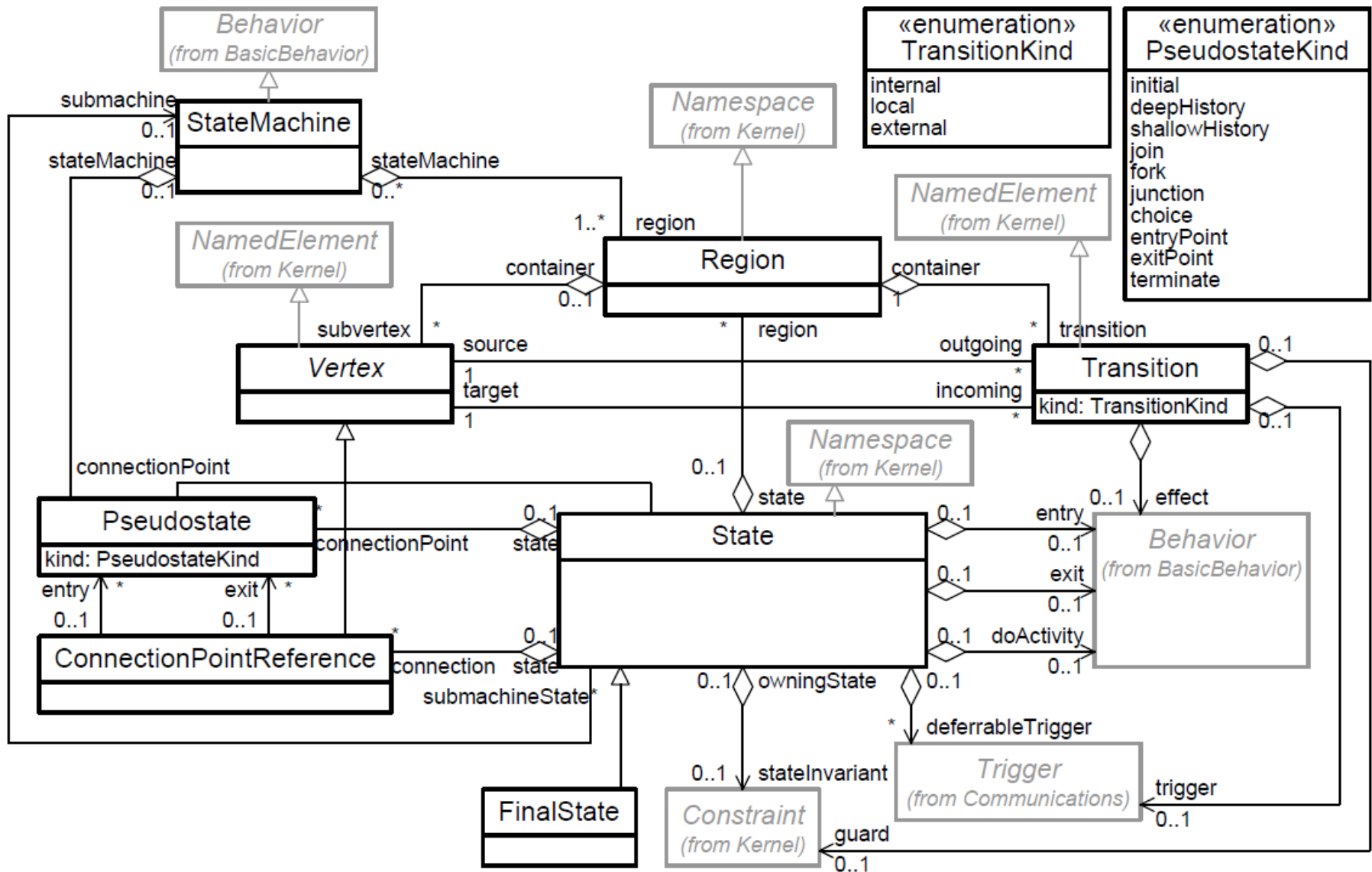


Szintaxis: **trigger** [guard] / action

Szétváló, egyesülő átmenetek; elágazás

Az állapottérképek szintaxisa (UML szerinti szintaxis)

UML State Machine metamodel



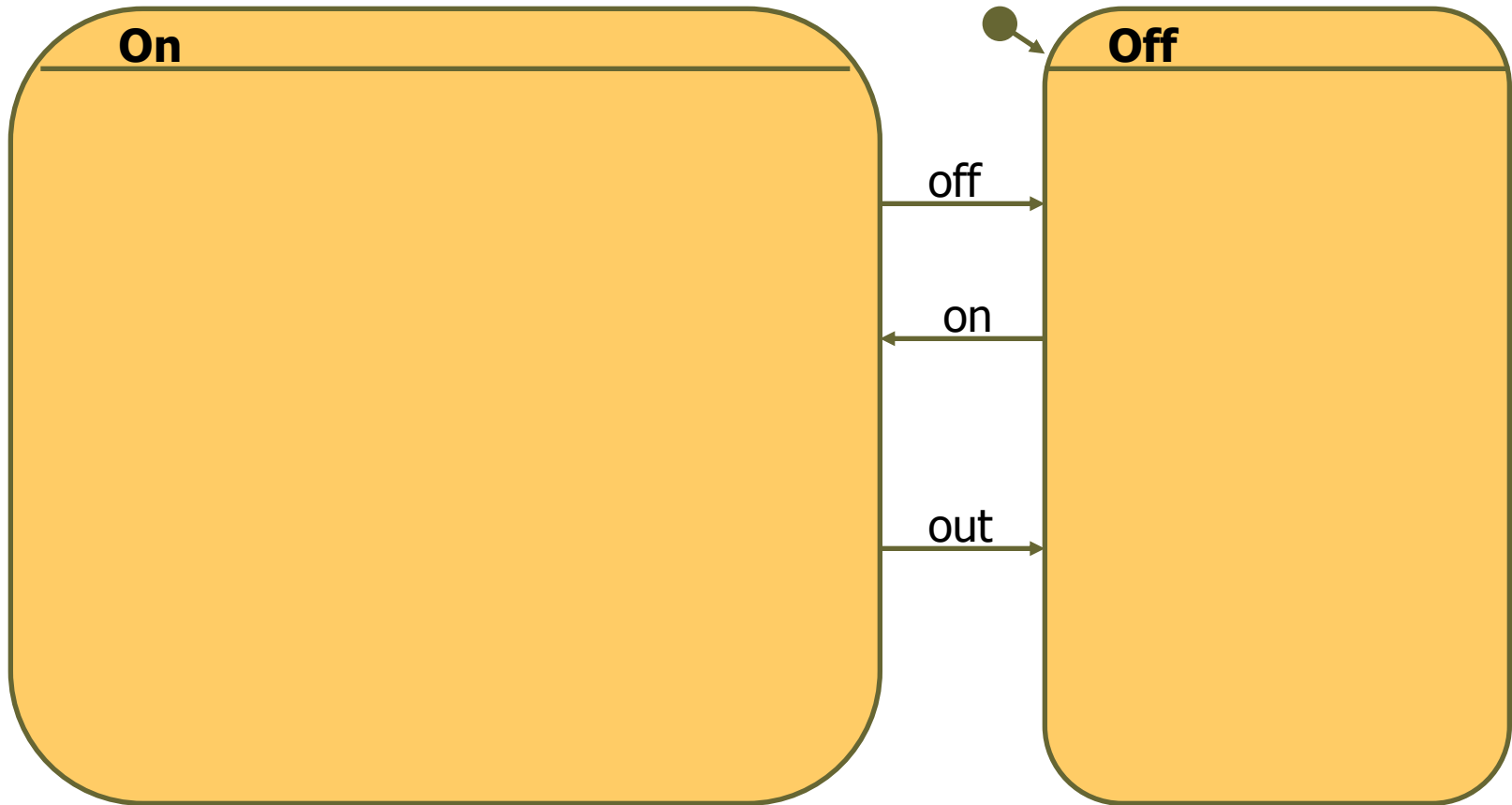
Állapotok: Akciók és állapotfinomítás

- Állapotok: Alapszintű modellelem
- Állapotokhoz kötődő akciók:
 - Belépési akció (**entry** / ...)
 - Kilépési akció (**exit** / ...)
 - Belső aktivitások (**do** / ..., **<event>** / ...)
- Állapotfinomítás
 - Egyszerű állapot: nincs finomítása
 - OR jellegű finomítás: alárendelt alállapotok
 - Ezek közül egyszerre egy állapot lehet aktív
 - AND jellegű finomítás: konkurens régiók
 - Egyidejűleg minden egyes régióban kell legyen aktív állapot!

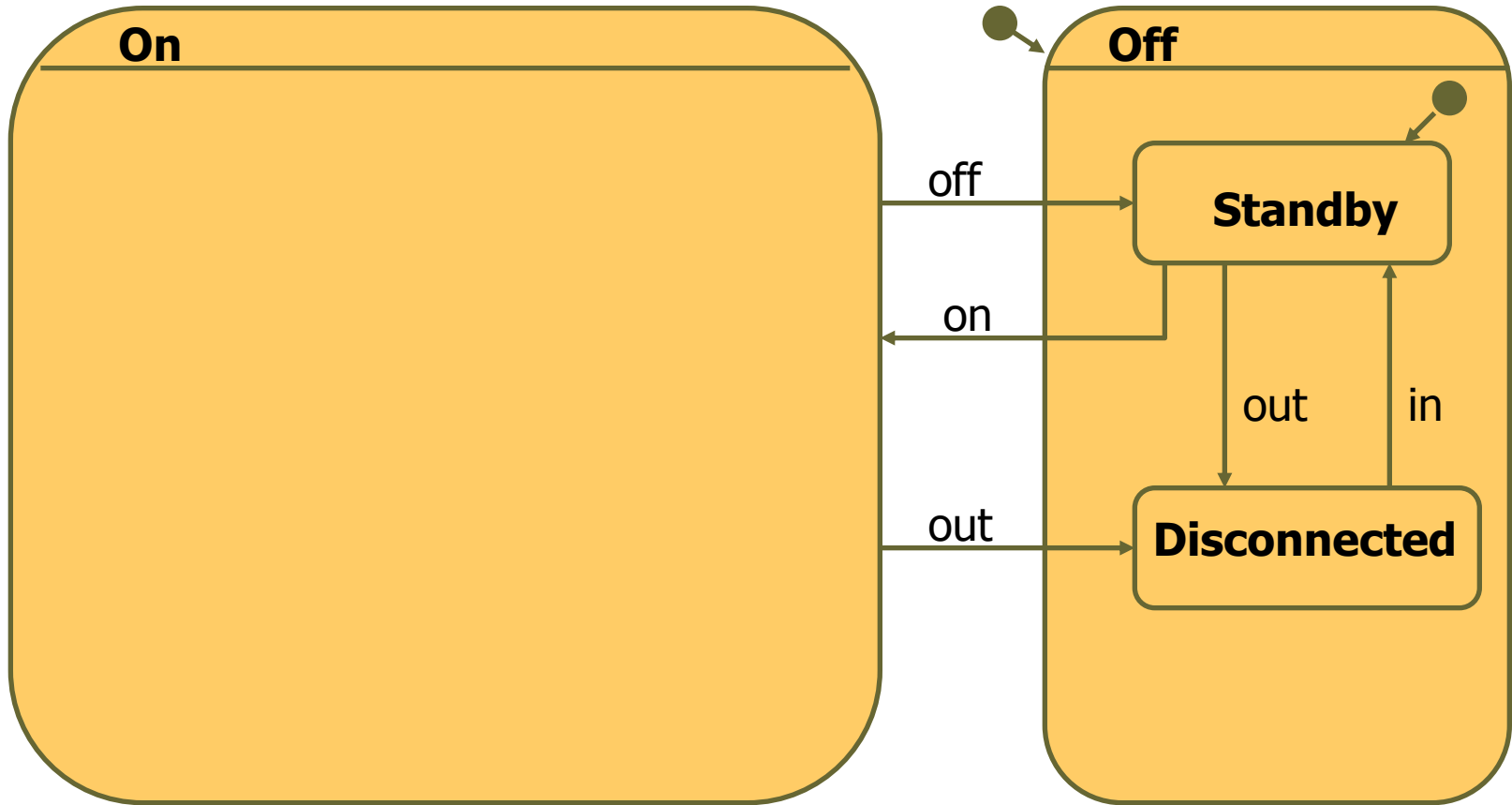
print_job

entry / init()
exit / reset()
do / poll()
job / print()

Példa: Állapotfinomítás

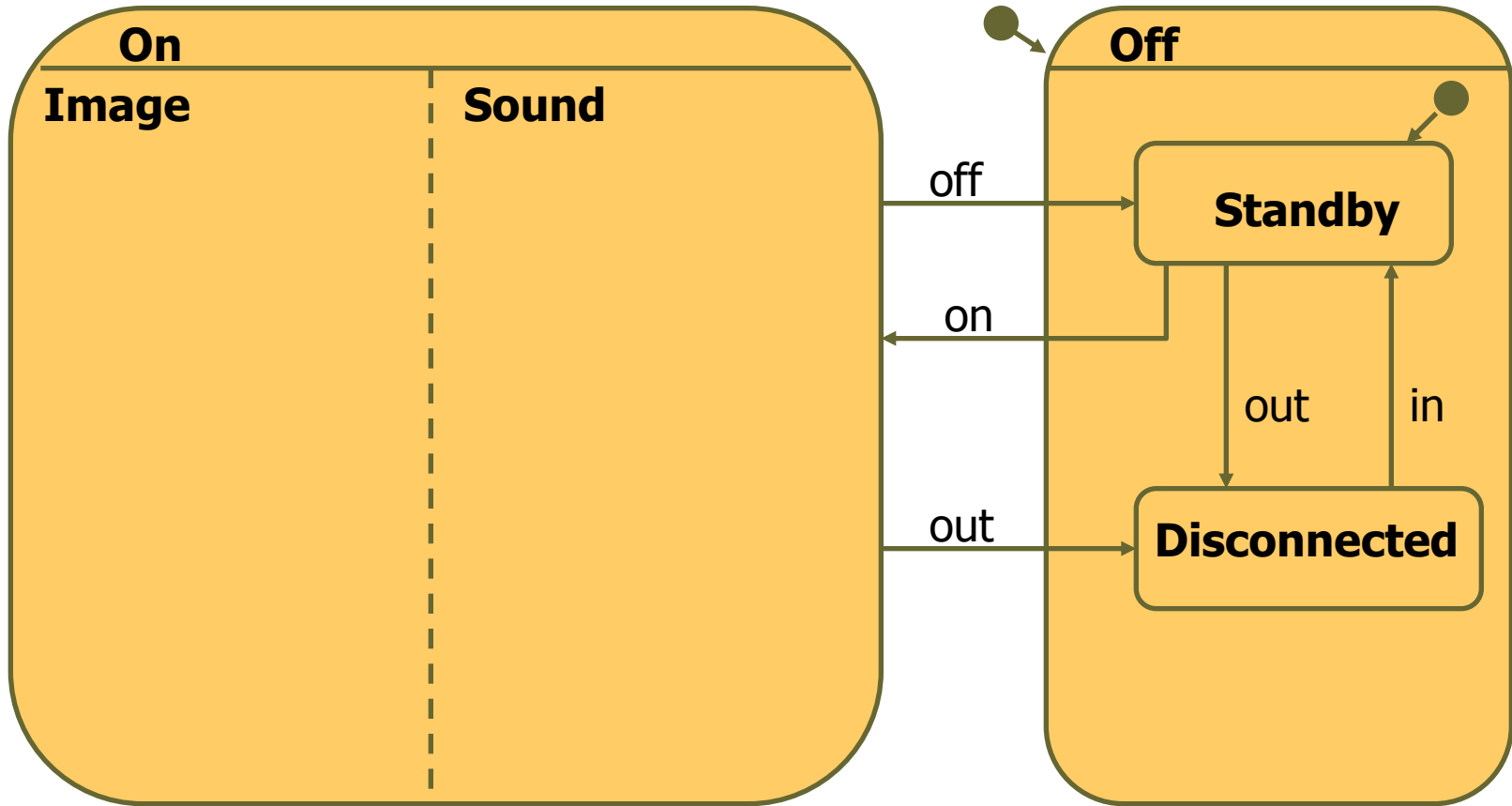


Példa: Állapotfinomítás



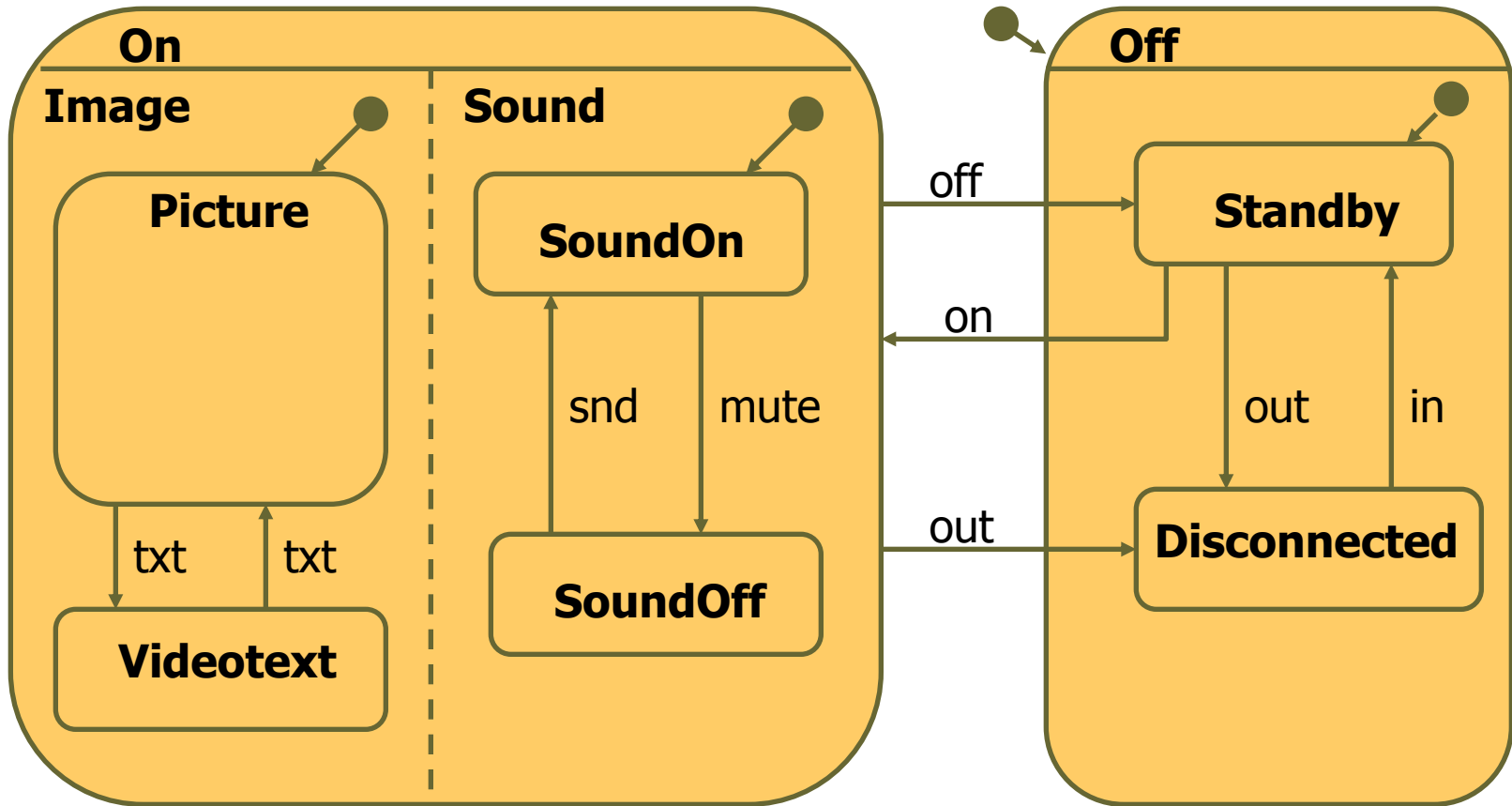
OR jellegű finomítás

Példa: Állapotfinomítás



AND jellegű finomítás

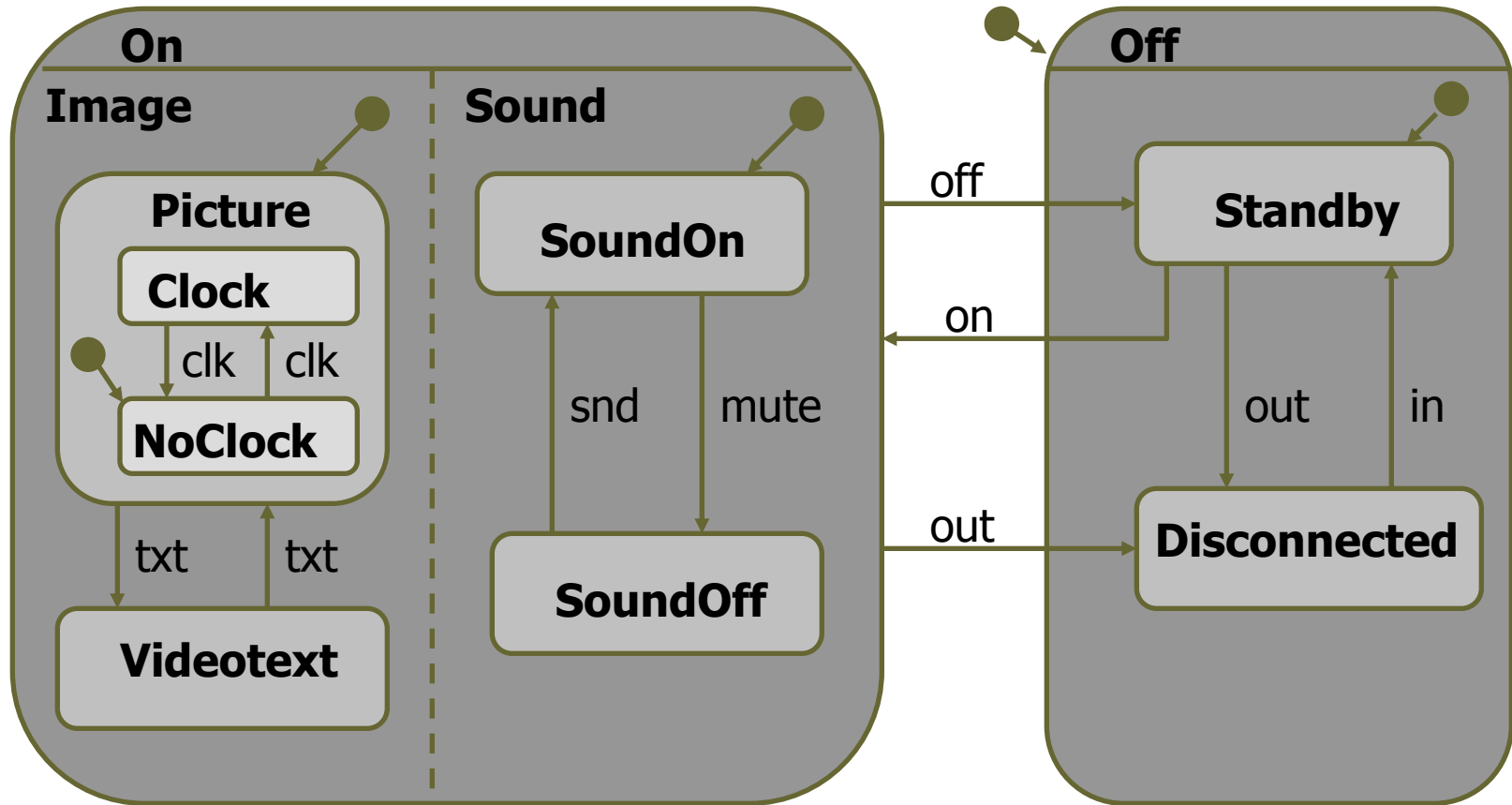
Példa: Állapotfinomítás



AND jellegű finomítás

OR jellegű finomítás

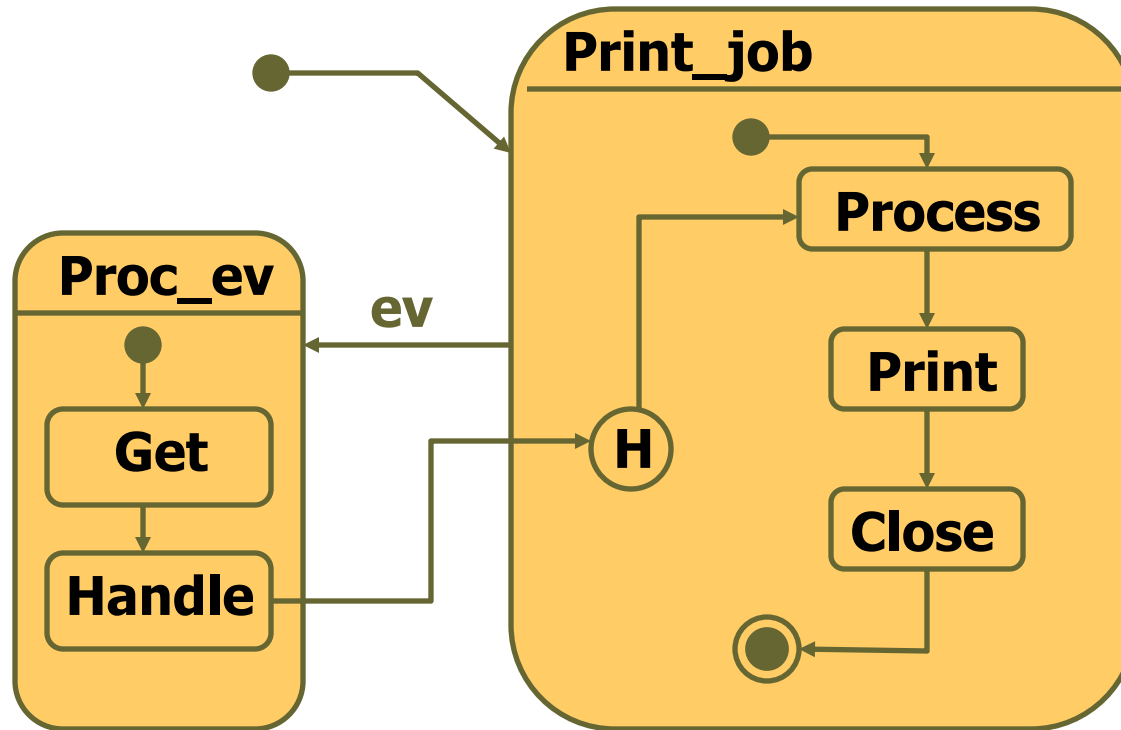
Példa: Állapotfinomítás



Pszeudo-állapotok

- **Kezdőállapot** jelzés: régióba való belépéskor lesz aktív 
 - Minden OR finomításban egy
 - Minden AND régióban egy
- **Végállapot** jelzés: állapotgép terminálás 
- **Emlékező** állapotok (history state)
 - Az utolsó aktív állapotkonfigurációt „tárolja” a tartalmazó állapotban
 - Egyszerű emlékező állapot: csak az adott finomítási szintet 
 - Mélyen emlékező állapot: a mélyebb finomítási szinteket is 
 - AND finomítás egy régiójában: Csak a régióra vonatkozik
 - Mit jelent az emlékező állapothoz húzott **bemenő** átmenet?
 - Tüzelésekor a „tárolt” állapotkonfigurációba kerül az objektum
 - Az emlékező állapot egy „hivatkozás” a tárolt állapotkonfigurációra
 - Mit jelent az emlékező állapotból húzott **kimenő** átmenet?
 - Alapértelmezett állapotot jelöl ki arra az esetre, ha előzőleg még nem volt aktív állapot

Példa: Emlékező állapot



(Állapot)átmenetek

- Állapotátmenetek megadása
- Szintaxis:

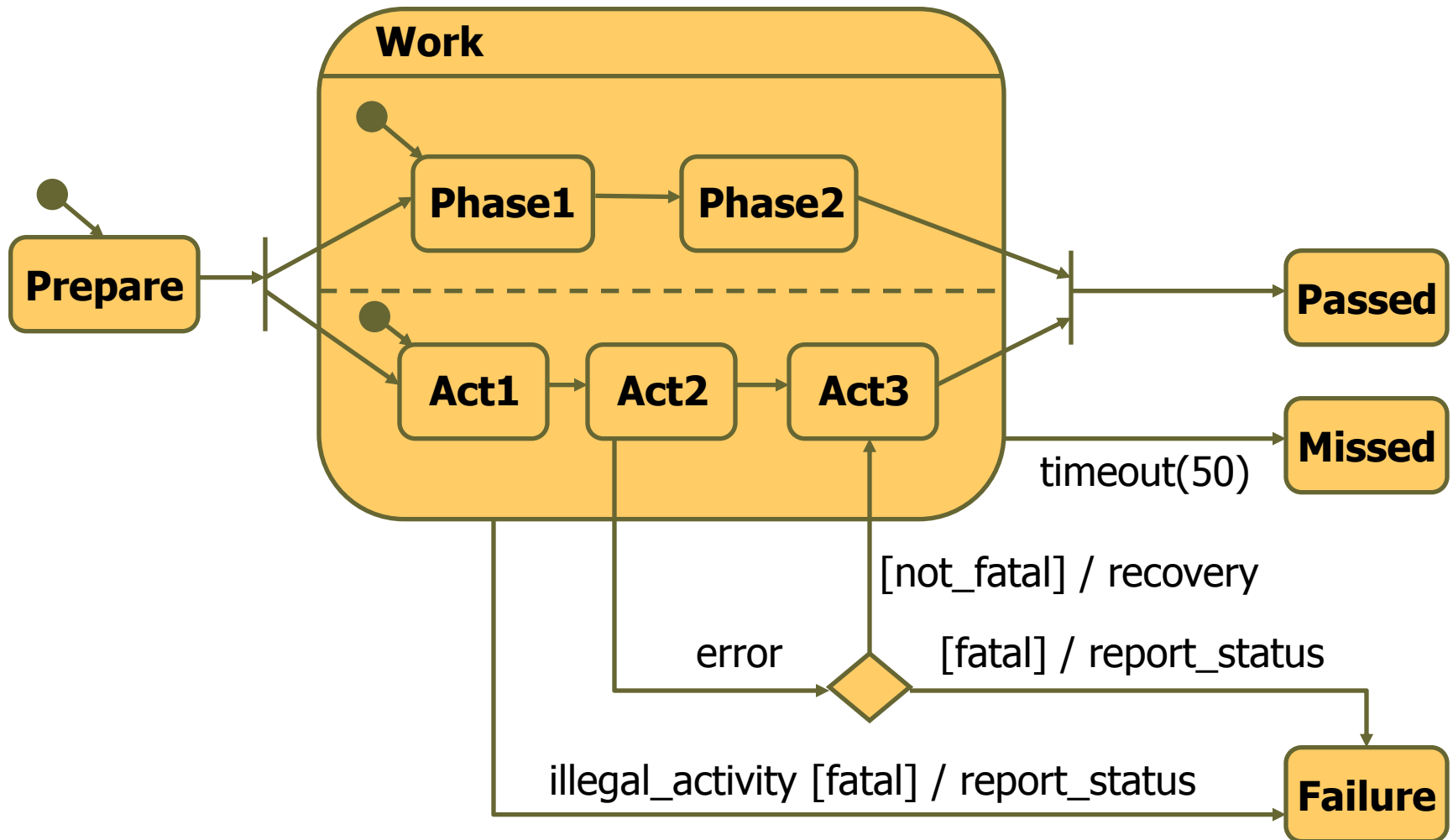
```
trigger [guard] / action
```

- **trigger**: kiváltó esemény
- **guard**: az átmenet őrfeltétele
 - Predikátum az állapotváltozók és esemény paraméterek felhasználásával
 - Állapotra való hivatkozás is lehet: `is_in(state)`
- **action**: akció (művelet)
 - akció szemantika: művelet részletezése

Átmenetek specialitásai

- Összetett átmenetek
 - Szétváló (fork): konkurens régiókban lévő állapotokba való együttes belépés
 - Egyesülő (join): konkurens régiókban lévő állapotokból való együttes kilépés
 - Elágazó (condition): több, őrfeltételtől függő átmenet egyszerűsített jelölése (szegmensek)
- Állapothierarchián átívelő átmenetek
 - Megengedett (bár nem elegáns)
- Time-out mint trigger
 - Fennáll, ha az állapotgép az átmenethez tartozó kiindulási állapotban tartózkodik az adott időintervallumban

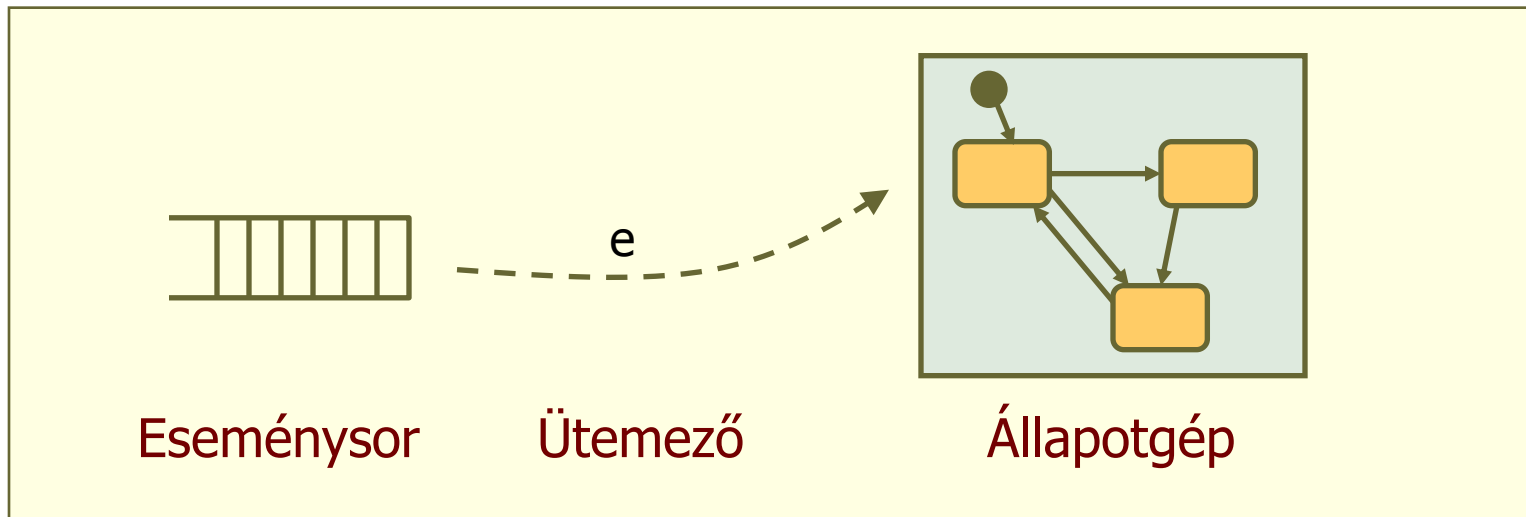
Példa: Állapotátmenetek



Az állapottérképek informális szemantikája (UML szerinti szemantika)

Szemantika: Hogyan működik?

- Alapelemek:
 - **Állapotgép:** Az állapottérkép írja le a viselkedését
 - **Eseménysor + Ütemező:** „Futtató rendszer”
(az állapotgép szempontjából külső elemek)



Mit ad meg a szemantika?

Mit tesz az állapotgép egy esemény hatására

→ állapotgép egy **lépése**

- Állapotátmenetek „**tüzelnek**”
 - Újdonság: egy esemény hatására több konkurens állapotátmenet tüzelhet (aktív régiókban)
- **Állapotkonfiguráció változik**
 - Állapotkonfiguráció: **Több aktív állapot** lehet
 - Aktív állapot minden AND régiójában kell legyen egy aktív állapot
 - Aktív állapot OR finomításában kell legyen egy aktív állapot
 - Egy OR finomításban illetve egy régióban csak egy aktív állapot lehet
 - Rekurzívan érvényes

A szemantika alaptulajdonságai

- Egyenként feldolgozott események
 - Az ütemező akkor küld újabb eseményt, ha az előző feldolgozása teljes egészében megtörtént
 - Stabil állapotkonfiguráció kialakult: nincs trigger nélküli átmenet
- Események teljes feldolgozása (run to completion)
 - Átmenetek maximális halmaza tüzel
 - Minden engedélyezett átmenet tüzel, kivéve ha ezt konfliktus megakadályozza
 - Ezek tüzelése után dolgozza fel a következő eseményt
- Az eseményfeldolgozás a szemantika lényege
 - Ez alapján lehet programkód alakjában megvalósítani az állapotgépet (forráskód generálás)

Az eseményfeldolgozás lépései 1/4

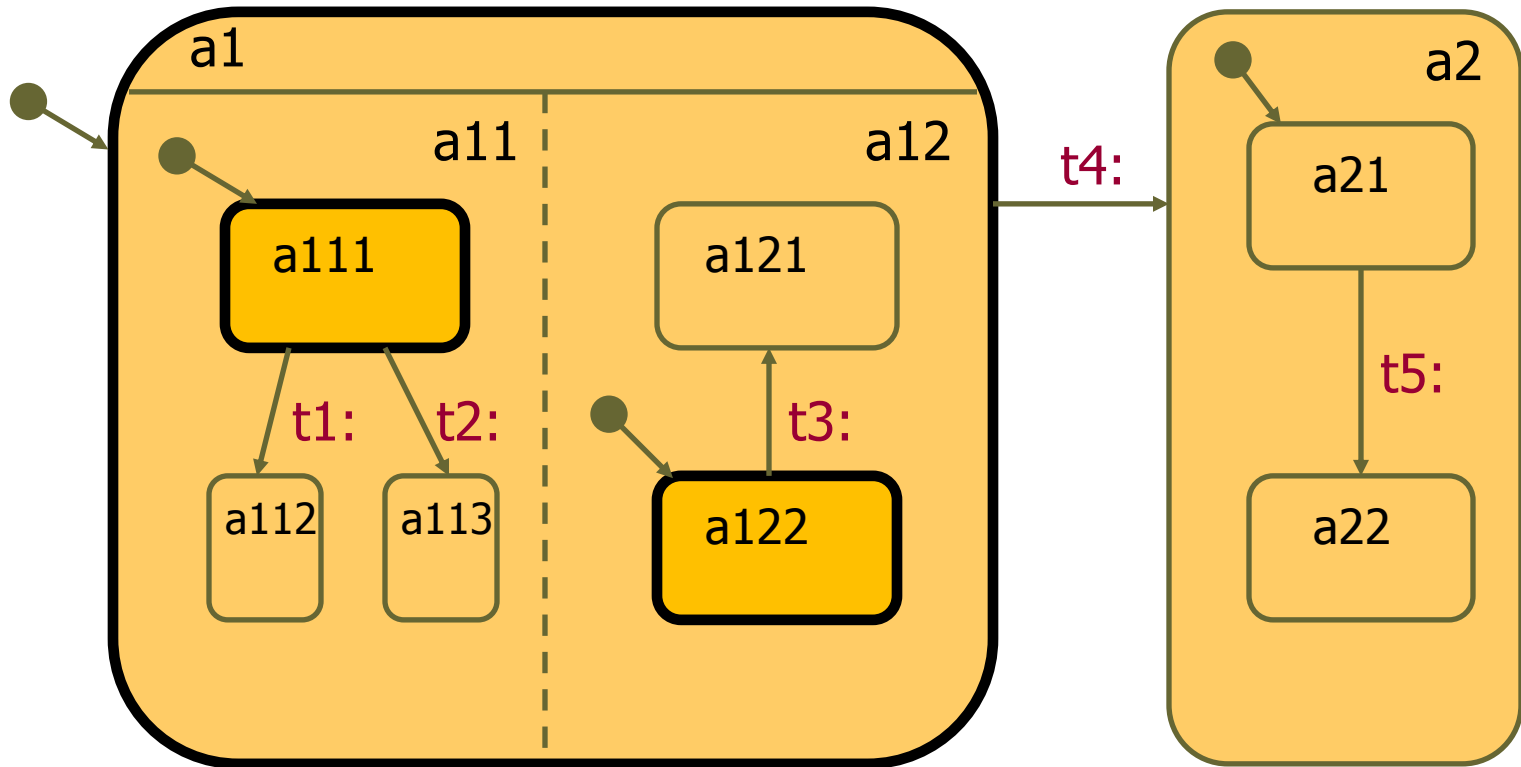
- Külső feltétel: Ütemező a stabil konfigurációban lévő állapotgépnek egy eseményt juttat
- Engedélyezett átmenetek:
 - Kiindulási (forrás-) állapot aktív
 - A kiválasztott esemény az átmenet triggere
 - Az őrfeltételek teljesülnek

Esetek az engedélyezett átmenetek száma alapján:

- Ha csak egy van: Tüzelhet!
- Ha nincs: Halasztott-e az esemény?
 - Igen: tárolás, új esemény kérése az ütemezőtől
 - Nem: esemény eldobható (hatás nélküli)
- Ha több van: Tüzelő átmenetek kiválasztása szükséges
 - Befolyásol: A konfliktus
 - Konfliktusban lévő átmenetek együtt nem tüzelhetnek

Példa: Konfliktus

A példában a **t1, ..., t5** nevű átmenetek **ugyanazon e eseményre triggereltek**. Melyek vannak konfliktusban? (Aktívak a vastagon bekeretezett állapotok.)



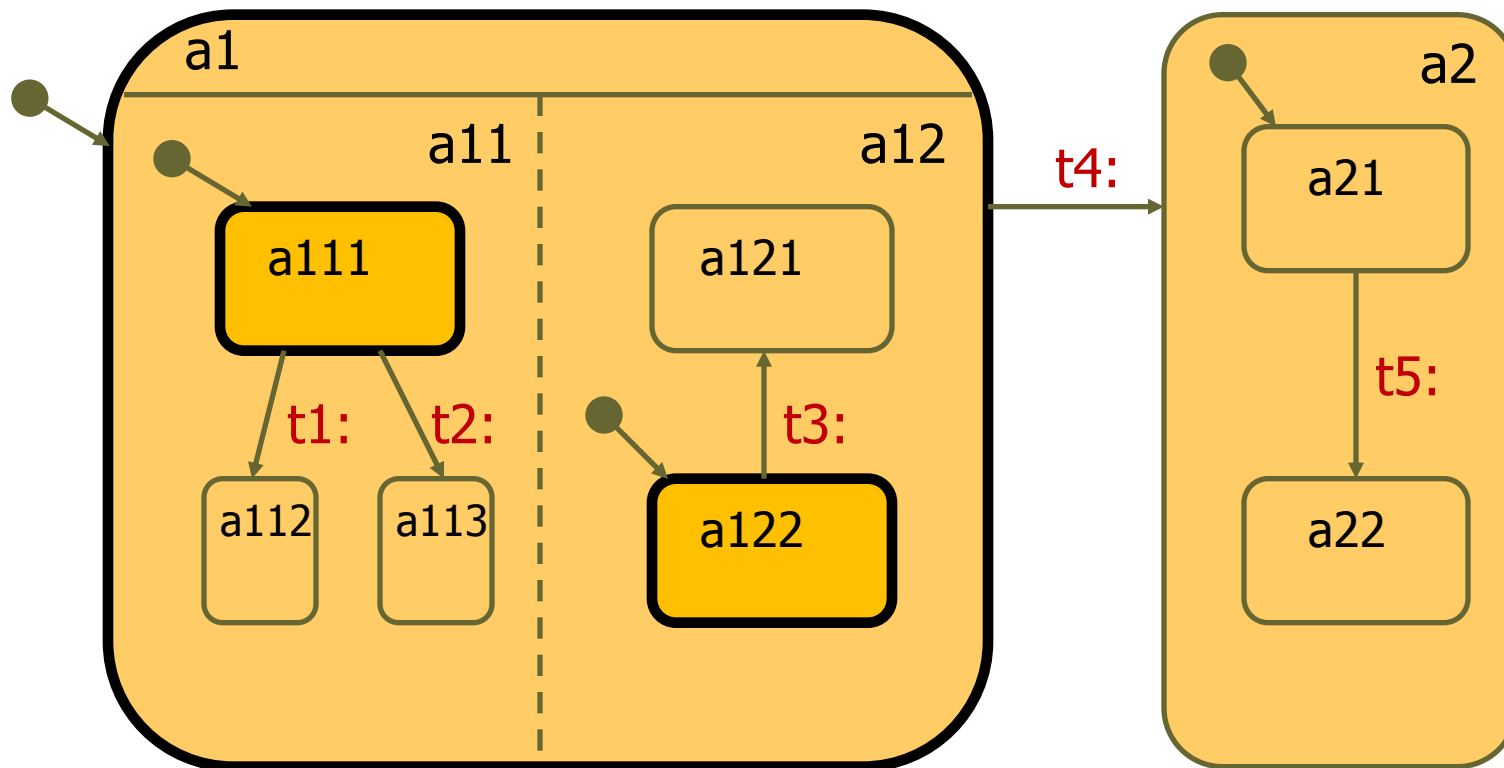
- Nem engedélyezett: t5 (forrásállapota nem aktív)
- Konfliktusban vannak: (t1,t2); (t4,t1); (t4,t2); (t4,t3)
- Egyszerre is tüzelhetnek: (t1,t3); (t2,t3);

Eseményfeldolgozás lépései 2/4

- **Tüzelő átmenetek kiválasztása:**
 - Maximális számú átmenet, nem lehet közöttük konfliktus
 - Konkurens átmenetek szimultán tüzelése
- **Konfliktusban lévő átmenetek:**
 - Ugyanazt az állapotot hagyják el, pontosabban az elhagyott állapothalmazok metszete nem üres
- **Konfliktus feloldása:**
 - **Prioritás alapján:** egy átmenet prioritása nagyobb, ha az átmenet kiindulási állapota az állapothierarchiában (állapotfinomításban) **alacsonyabb szintű**
 - OO koncepció: a finomítás „felüldefiniál”
 - **Véletlenszerű választás,** ha azonos prioritásúak

Példa: Konfliktusfeloldás

A **t1**, ..., **t5** átmenetek ugyanazon eseményre triggereltek.
Melyek tüzelhetnek együtt?

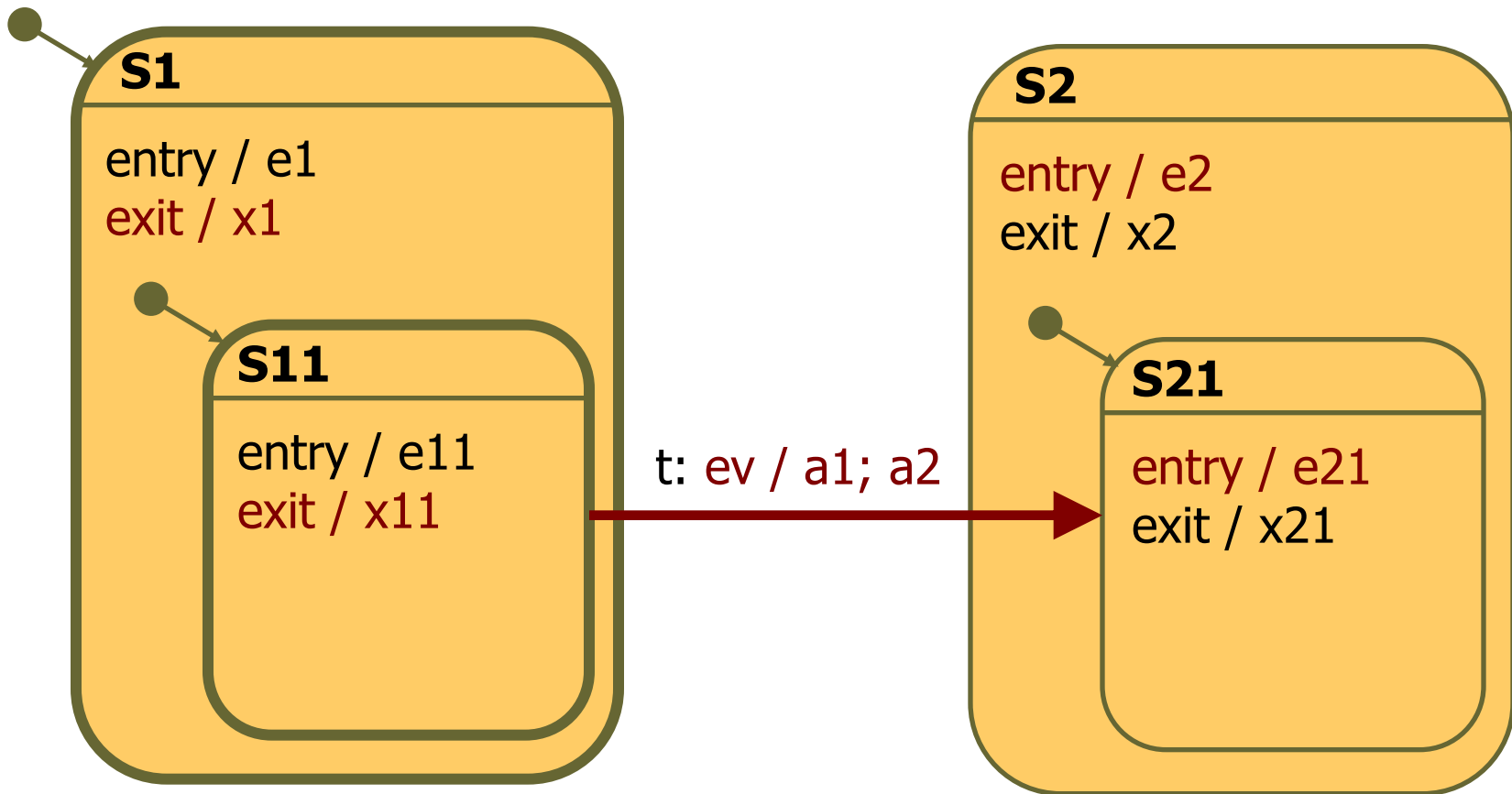


- Nagyobb prioritású t4-nél: t1, t2 és t3
- Tüzelhet: (t1,t3) vagy (t2,t3)

Eseményfeldolgozás lépései 3/4

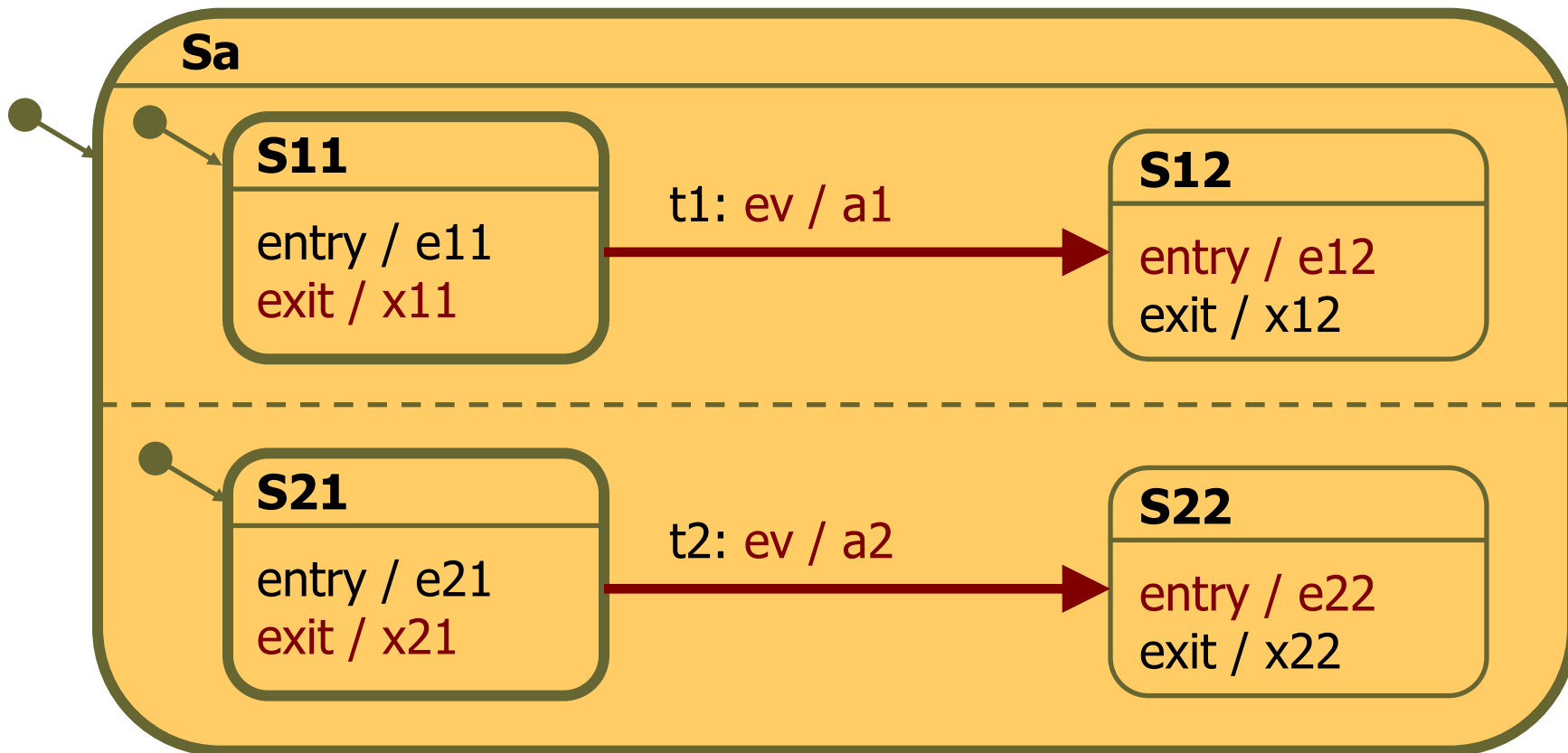
- Kiválasztott átmenetek **tűzelnek**:
 - Sorrend véletlenszerű (nincs köztük konfliktus)
 - Akcióik közötti sorrend is véletlenszerű
- Egy átmenet tüzelése és akciói:
 1. Kiindulási állapotok **elhagyása**
 - Alacsonyabb hierarchiaszinten (mélyebben lévő) először
 - Kilépési akciók végrehajtása (**exit** akciók)
 2. Átmenetek akcióinak **végrehajtása**
 3. Célállapotokba való **belépés** → új konfiguráció
 - Magasabb hierarchiaszinten lévő először
 - Belépési akciók végrehajtása (**entry** akciók)

Példa: Akciók sorrendezése



Akciók sorrendje ev hatására: x11; x1; a1; a2; e2; e21

Példa: Akciók sorrendezése

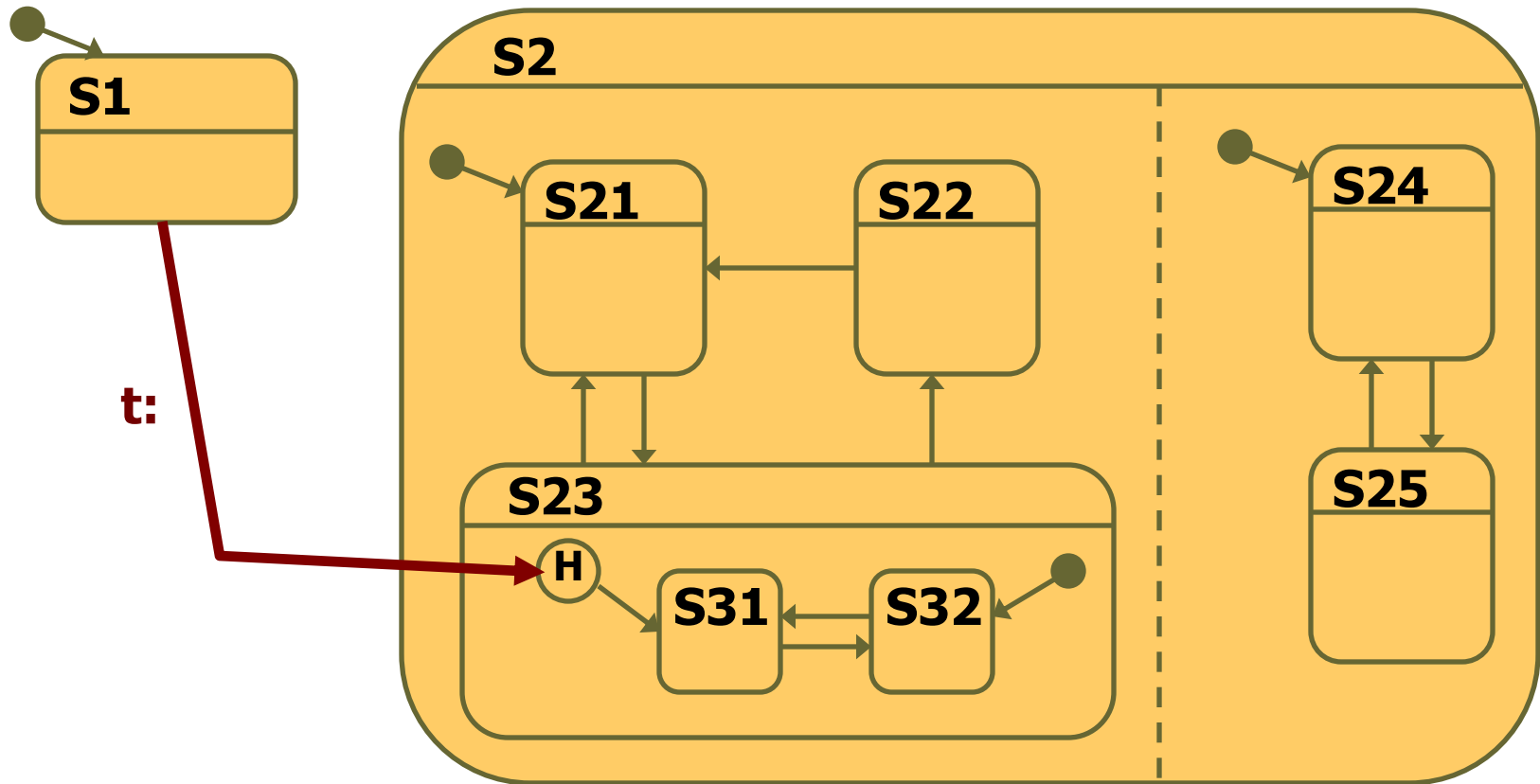


Akciók sorrendje **ev** hatására: $(x11; a1; e12) \parallel (x21; a2; e22)$
itt $\parallel$ jelzi a konkurens végrehajtást;
zárójelen belüli akciók sorrendje egymáshoz képest rögzített

Eseményfeldolgozás lépései 4/4

- **Belépés** új konfigurációba különféle jellegű célállapotok esetén:
 - **Ha egyszerű** (nem finomított) a célállapot:
 - Új konfiguráció része lesz
 - Ős állapotai (amelyek finomításában szerepel) is aktívak lesznek
 - Aktívvá váló ős állapotok minden régiójában lesz aktív állapot (kezdőállapot jelöli ki)
 - **Ha OR finomítása van** a célállapotnak:
 - A finomításban a kezdőállapot jelöli ki az aktív állapotot
 - **Ha AND finomítása van** a célállapotnak:
 - Minden régiójában kell legyen aktív állapot (kezdőállapot jelöli ki)
 - **Ha emlékező állapot:**
 - Az utoljára elhagyott állapotkonfiguráció lesz aktív
 - Ha nem volt még ilyen: a kimenő él határozza meg
 - **Ha nem stabil** az állapot: azonnali továbblépés

Példa: Belépés konkurens állapotba



A **t** átmenet tüzelése után mi lesz az új állapotkonfiguráció?

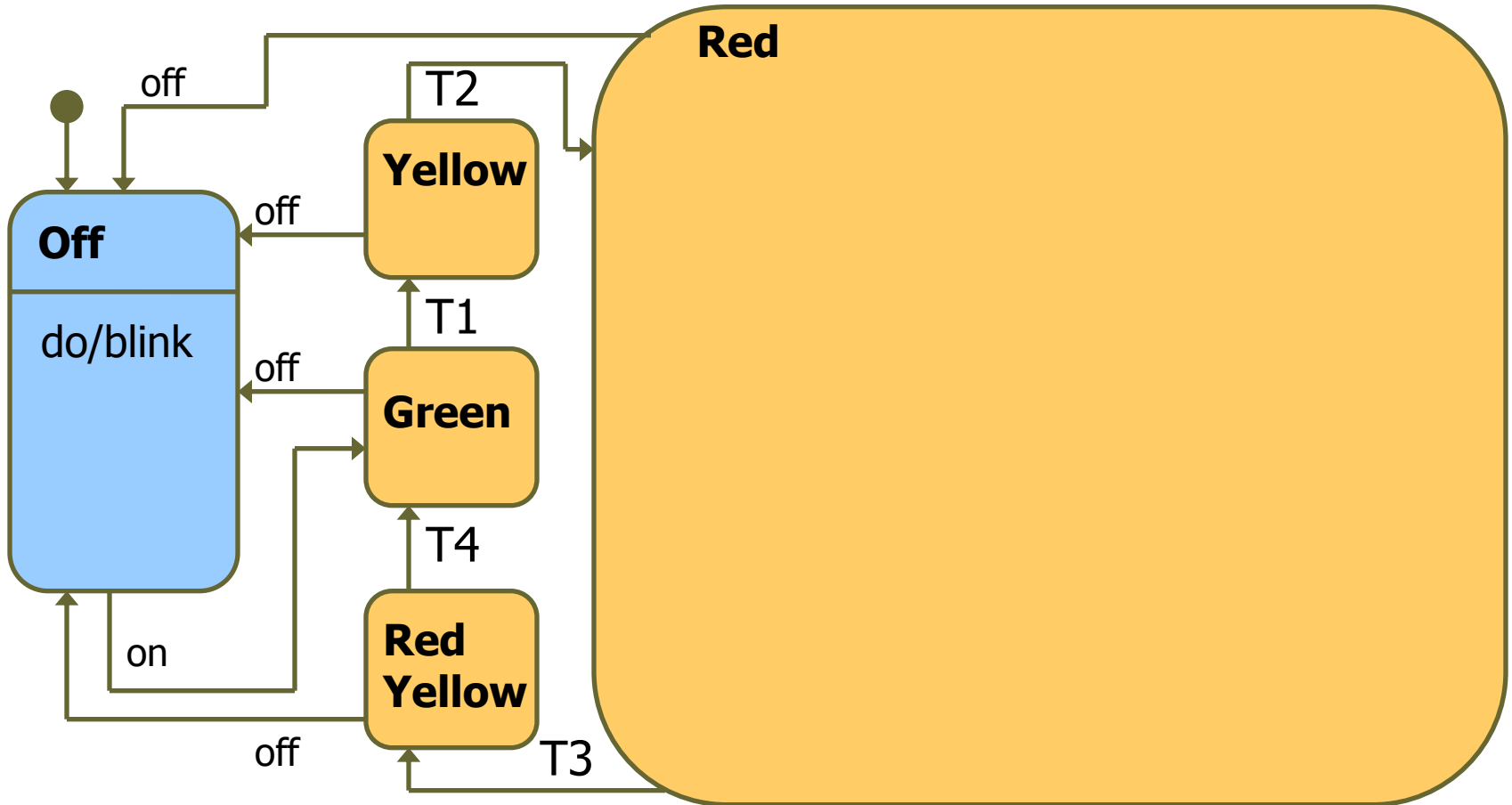
S2, S23, S24 aktív

S23 alállapota: Ha volt már aktív állapotkonfiguráció része, H jelöli ki;
ha nem volt, akkor S31

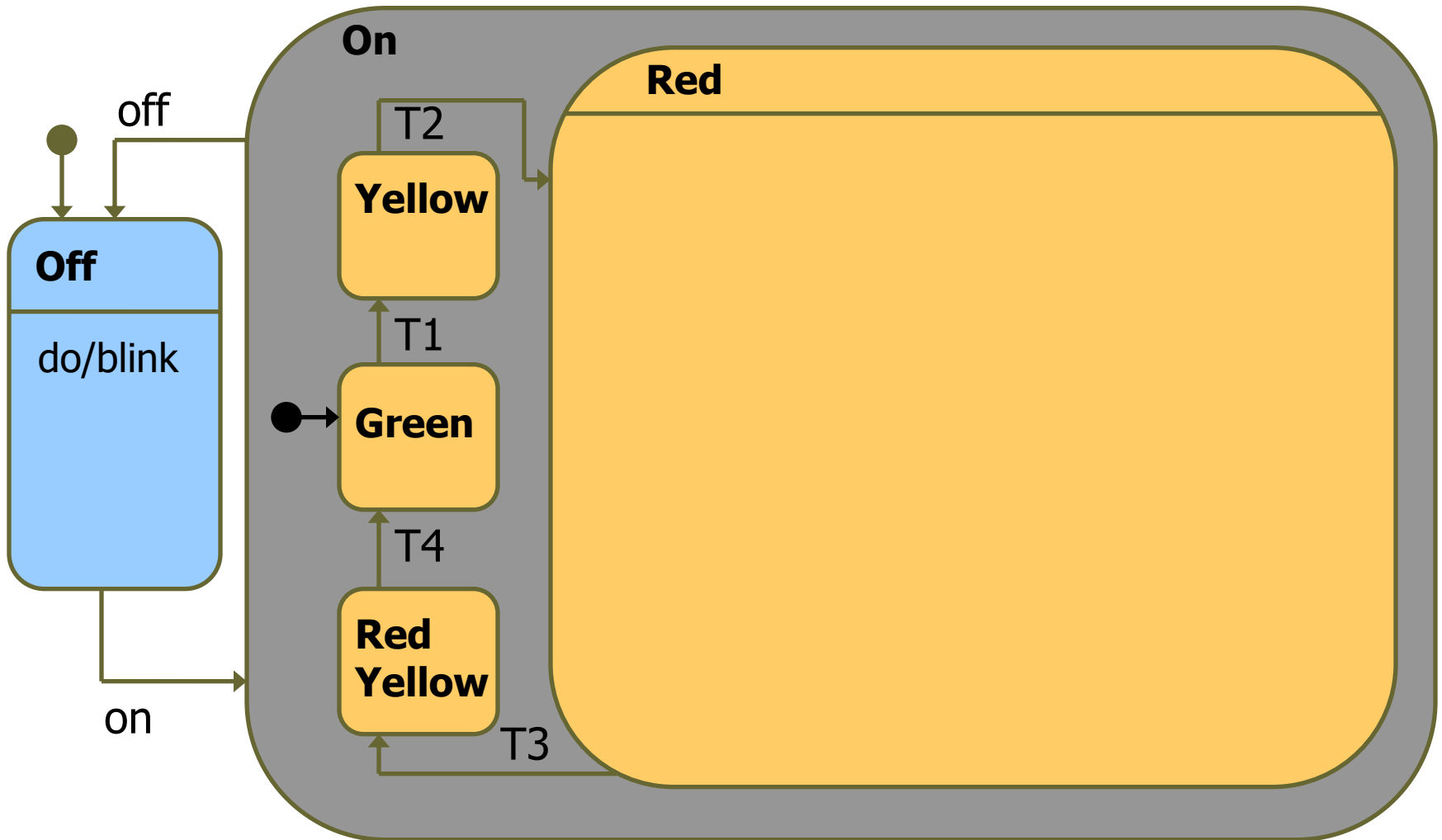
Modellezési mintapélda

- Közlekedési lámpa vezérlője egy főútvonal és egy mellékútvonal kereszteződésében
 - Kikapcsolva: villogó sárga
 - Bekapcsoláskor: főútvonal számára zölddel kezd
 - Zöld, sárga, piros váltás: időzítő eseményre
 - Főútvonalon 3 várakozó: időzítőtől függetlenül zöld jelzés szükséges
 - Főútvonalon tilosban áthajtók: fényképezés
 - Ez a funkció kézzel ki-be kapcsolható

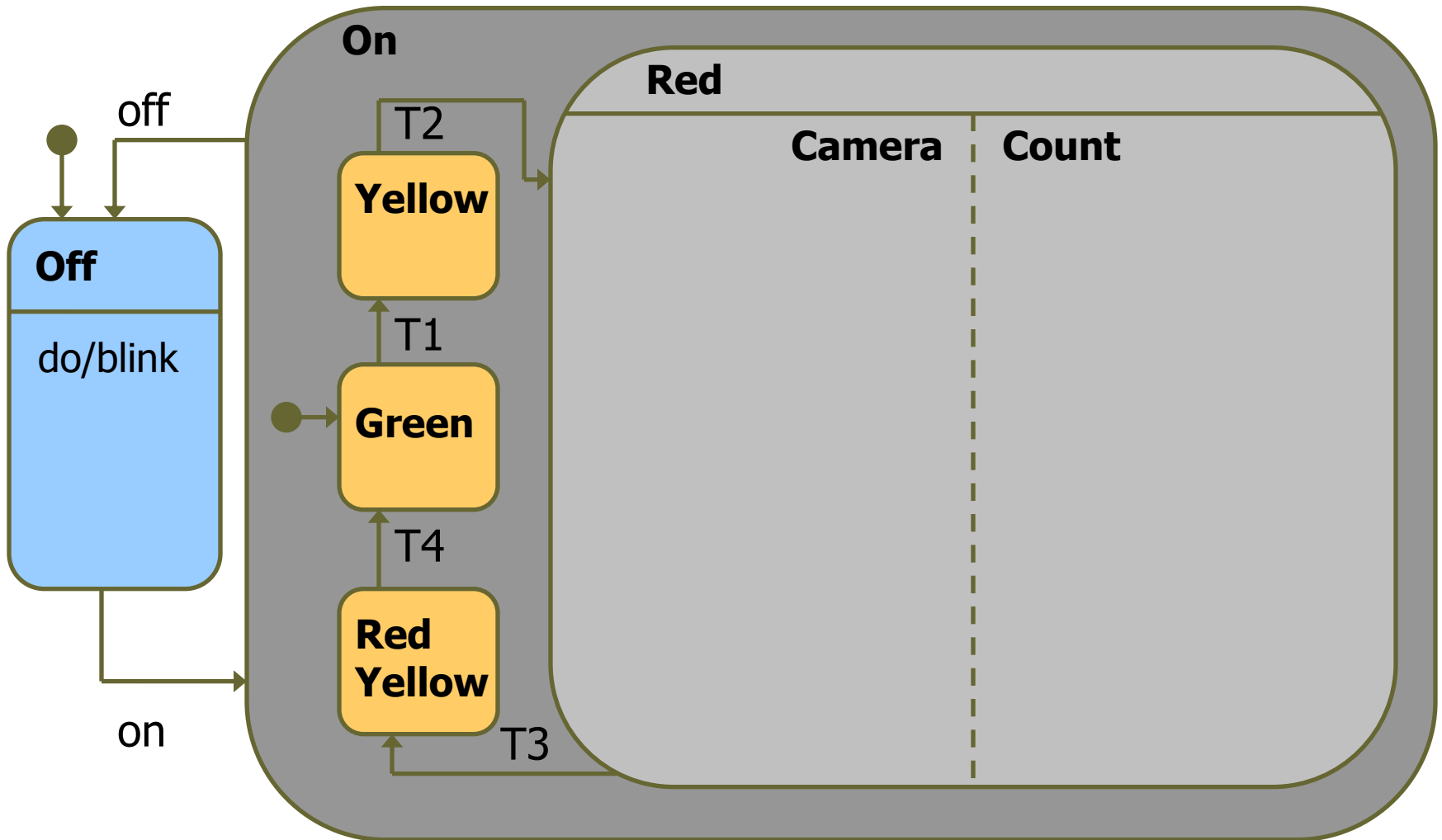
1. Lámpaváltás (főútvonal számára)



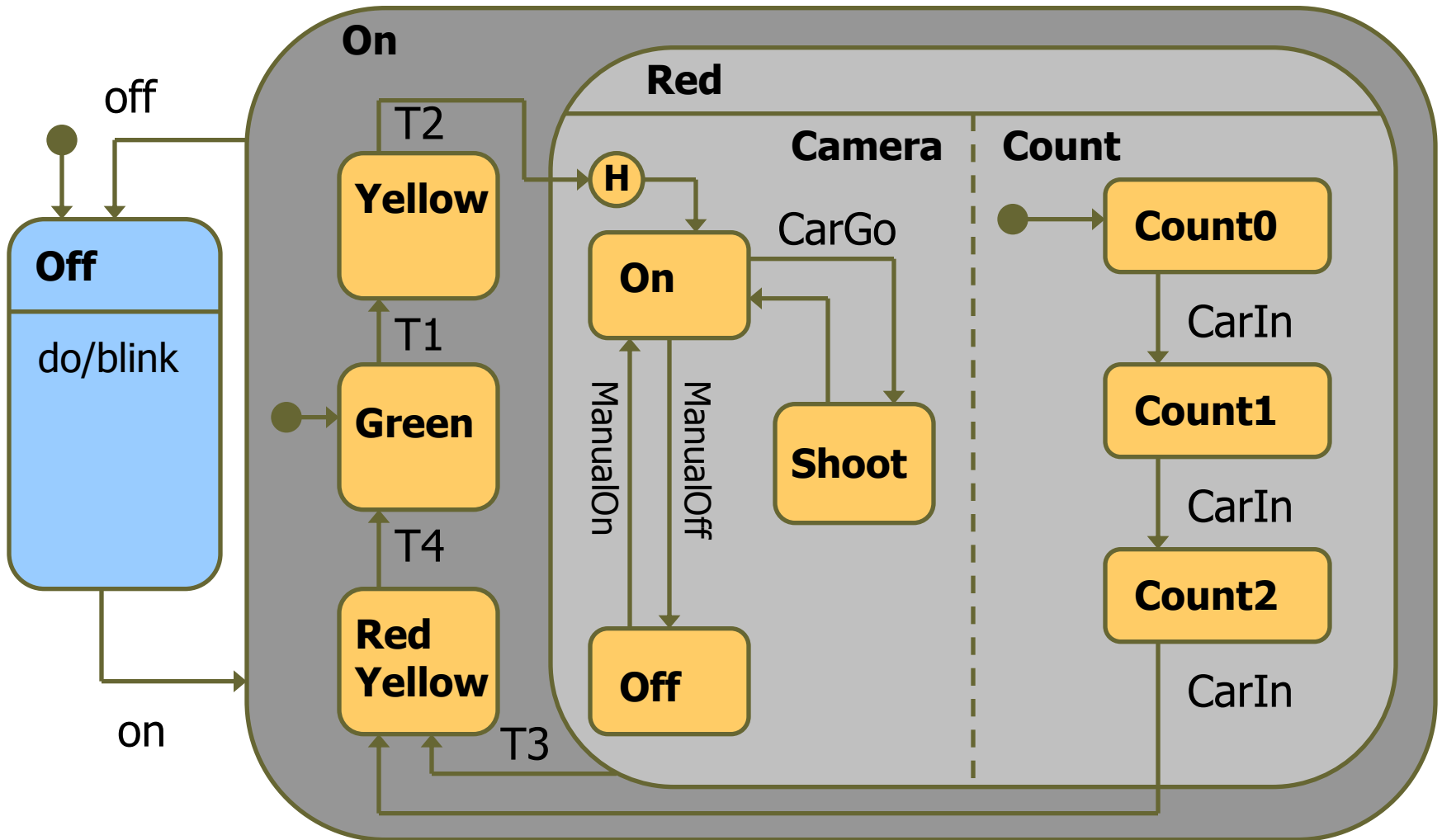
2. Állapothierarchia



3. Konkurens állapotok



4. Teljes vezérlő



Állapottérképek szerepe az UML 2 esetén

- Állapot alapú, eseményvezérelt működés leírása
 - Aktív objektumok viselkedésének leírása
- Akciók formalizálása: UML 2 Action Semantics
 - Metódushívás
 - Attribútum olvasás, írás
 - ... (sokféle művelet)
 - Színezett Petri-hálókhöz (ld. később) hasonlító alapelvek
- Akciók leírása: Más alternatívák is vannak (pl. Alf)
- Ezek alapján megoldható
 - Forráskód generálás
 - Formális modell generálása verifikációhoz
 - Leképezés alacsonyabb szintű modellre
 - Tesztek generálása
 - Az implementált kód adott platformon a modell szerint viselkedik-e?
 - Monitor kód generálása futásidőbeli ellenőrzéshez

Állapottérképek alapjai (összefoglalás)

- Kiterjesztések
- Állapottérkép szintaxis
 - Állapothierarchia, konkurens régiók, emlékező állapotok
 - Összetett átmenetek
- Állapottérkép (nem formális) szemantika
 - Átmenetek engedélyezettsége
 - Tüzelő átmenetek kiválasztása
 - Átmenetek tüzelése
 - Új állapotkonfiguráció kialakulása
- Állapottérkép eszközök (példák):
 - UML 2 támogató eszközök: MagicDraw, Papyrus, ...
 - Yakindu Statechart Tools (statecharts.org)
 - Quantum Programming (state-machine.com)

Állapottérkép komponensek kompozíciója

Állapottérkép alapú komponensek

Rendszer: Állatógép
komponensekből áll

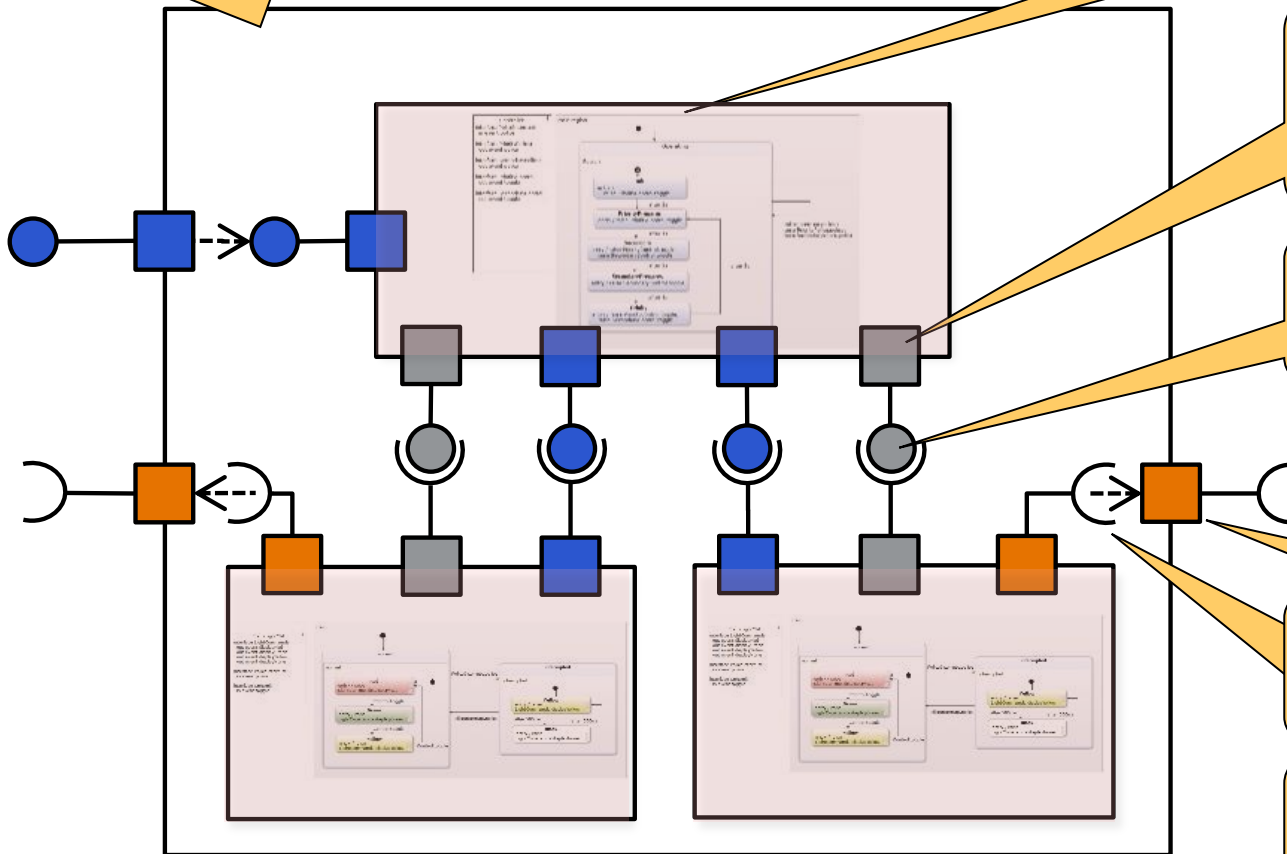
Állatógép komponens
(egy modell példánya)

Port: komponens
interakciója a külvilággal;
interfész megvalósítása

Csatorna: portok
összekötése

Külső port: rendszer
interakciója a külvilággal

Kötés: külső port és
komponens port között



Komponensek definiálása

- **Interfész definíció**
 - Esemény: valamilyen történés (pl. jel, üzenet)
 - Irányultság: **in**, **out**, **inout**
- **Port definíció: egy interfészt valósít meg**
 - Mód: **requires**, **provides**
- **Komponensek összekötése**
 - Belső portok: **Csatornák** komponens portok között; **unicast**, **broadcast**
 - Külső (rendszer) portok: **Kötés** komponens portokhoz

Példák:

```
interface Control {  
    out event tick;  
}
```

```
statechart Controller [  
    port send provides Control  
    port rcv requires Control  
] { ... }
```

```
component ctrl1 : Controller  
component ctrl2 : Controller
```

```
channel [ctrl1.send] -o)- [ctrl2.rcv]  
bind sysRcv -> ctrl1.rcv  
bind sysSend -> ctrl2.send
```

Szemantika: Komponensek végrehajtása

- Szinkron

- Végrehajtás: Rendszerszintű ciklusok
- Egy ciklus: minden komponens egy lépésben feldolgozza a bemeneti eseményeit és előállítja a kimeneti eseményeit
- Kimeneti eseményt a következő ciklusban dolgozza fel az a komponens, aminek a bemenetére kerül (csatornán)

- Kaszkád

- Végrehajtás: Ciklusok komponensek sorrendi végrehajtásával
- Egy ciklus: adott sorrendben komponensek egy-egy lépése (egy komponens akár többször is végrehajtható)
- Kimeneti eseményt a következő végrehajtása során dolgozza fel az a komponens, aminek a bemenetére kerül

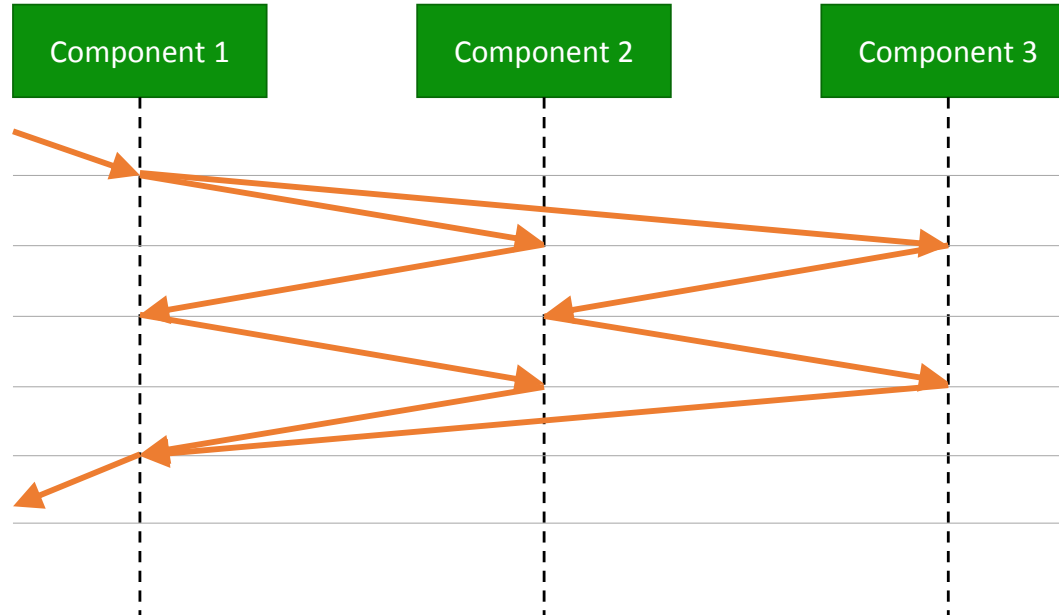
- Aszinkron

- Végrehajtás: Függetlenül az egyes komponensek esetén
- Eseménysorok szükségesek a komponensek között

Szemantika: Szinkron, kaszkád, aszinkron

- Szinkron

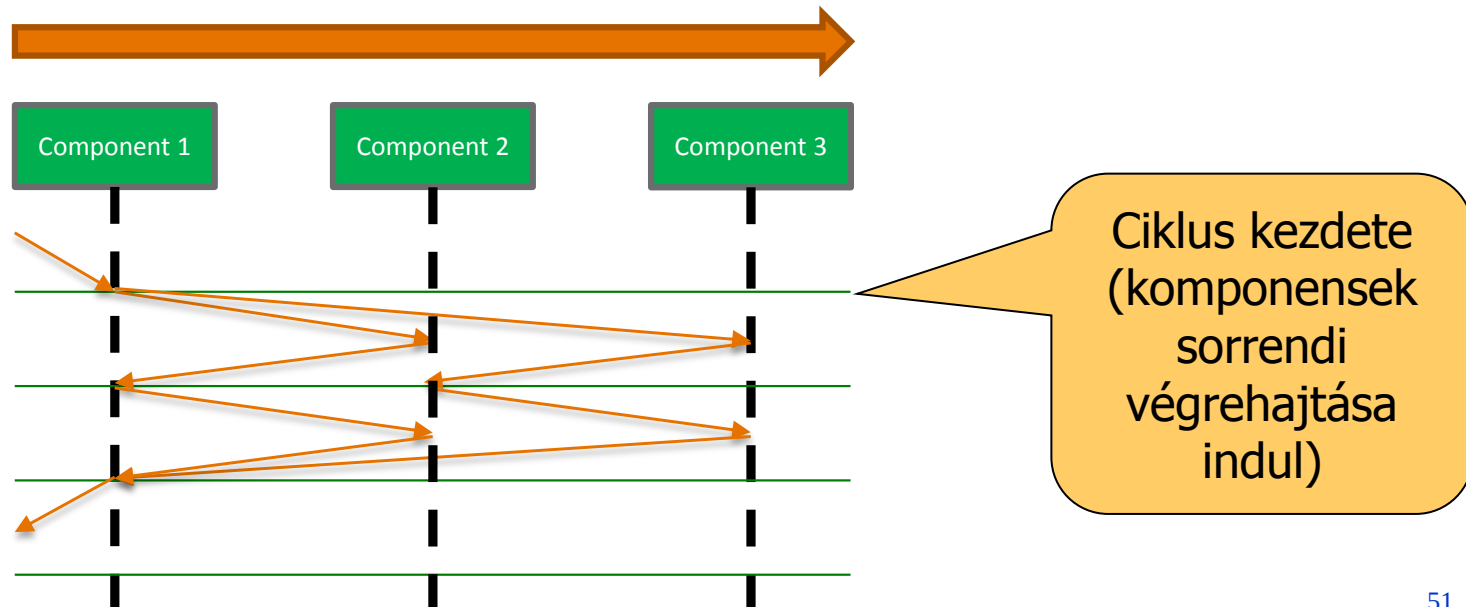
- Végrehajtás: Rendszerszintű ciklusok
- Egy ciklus: minden komponens egy lépésben feldolgozza a bemeneti eseményeit és előállítja a kimeneti eseményeit
- Kimeneti eseményt a következő ciklusban dolgozza fel az a komponens, aminek a bemenetére kerül (csatornán)



Ciklus kezdete
(komponensek
végrehajtási
sorrendje nem
releváns)

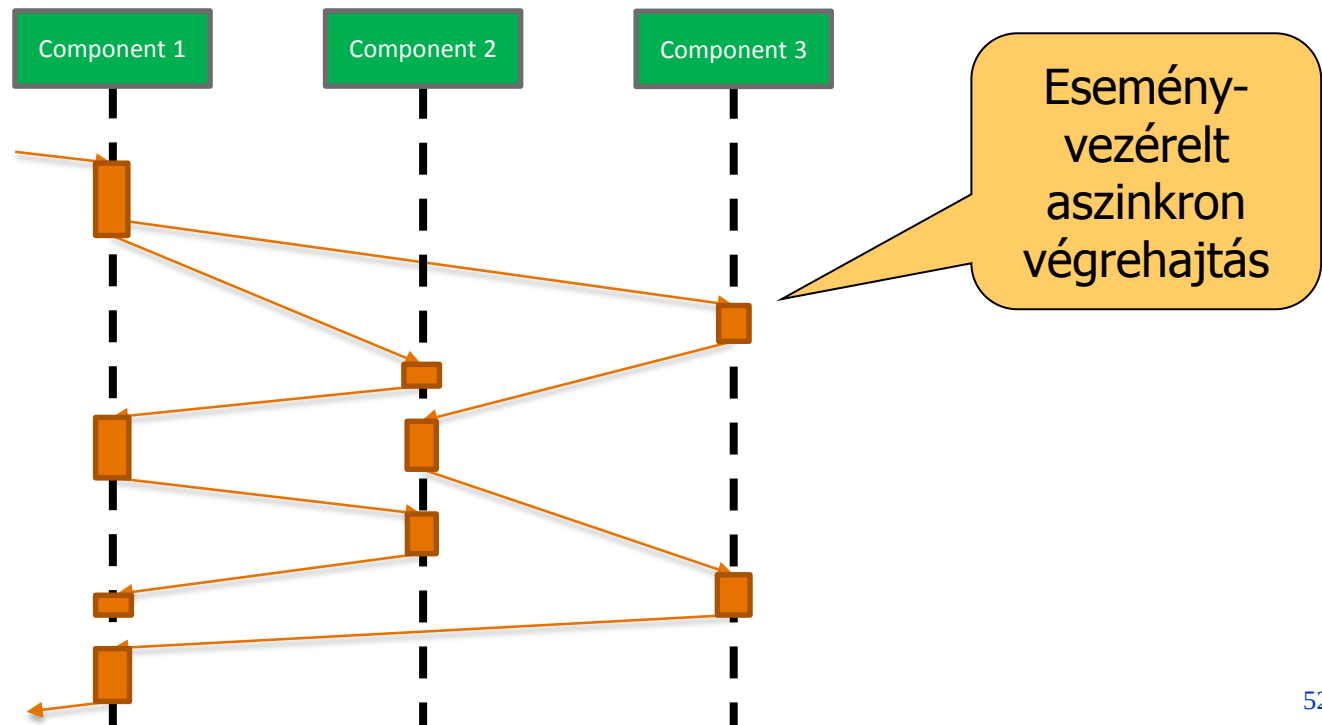
Szemantika: Szinkron, kaszkád, aszinkron

- Szinkron: Rendszerszintű ciklusok
- Kaszkád
 - Végrehajtás: **Ciklusok** komponensek sorrendi végrehajtásával
 - **Egy ciklus**: adott sorrendben komponensek egy-egy lépése (egy komponens akár többször is végrehajtható)
 - Kimeneti eseményt a **következő végrehajtása során** dolgozza fel az a komponens, aminek a bemenetére kerül



Szemantika: Szinkron, kaszkád, aszinkron

- Szinkron: Rendszerszintű ciklusok
- Kaszkád: Komponensek sorrendi végrehajtása
- Aszinkron
 - Végrehajtás: **Függetlenül** az egyes komponensek esetén
 - Eseménysorok a komponensek között



Forráskód szintézis állapotterkép specifikáció alapján

Megvalósítási minták

- Cél:
 - Vezérlési struktúra szintézise
 - Akciók részletezése: Akció nyelv szükséges
- Többféle megvalósítási minta
 - Kettős elágazás
 - State, Hierarchical State Machine
 - Extended Hierarchical Automata
- Különbségek
 - Állapottérkép elemkészlet támogatása
 - Memóriafoglalás
 - Sebesség (eseményfeldolgozás, vezérlési struktúra)

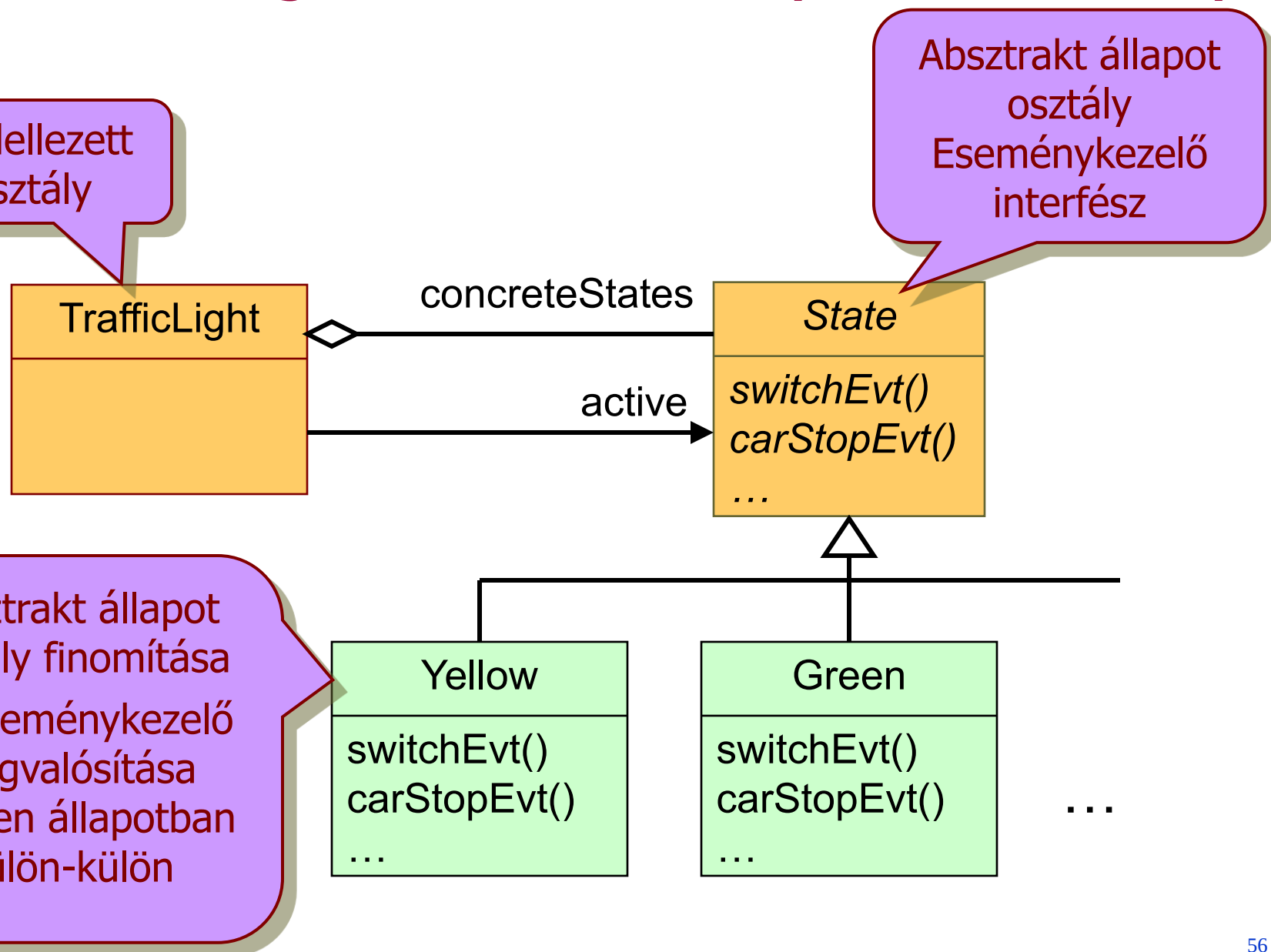
„Kettős elágazás” megvalósítási minta

TrafficLight
state: State
handler(event: Event)

```
switch (state) {  
  case GREEN:  
    switch (event) {  
      case tGreen:  
        /* Yellow - entry */  
        state = YELLOW;  
        break;  
      ...  
    }  
    ...  
}
```

- Egyszerű és hatékony
- Állapothierarchiát és konkurens viselkedést nem támogat

A „State” megvalósítási minta (Gamma, 1994)



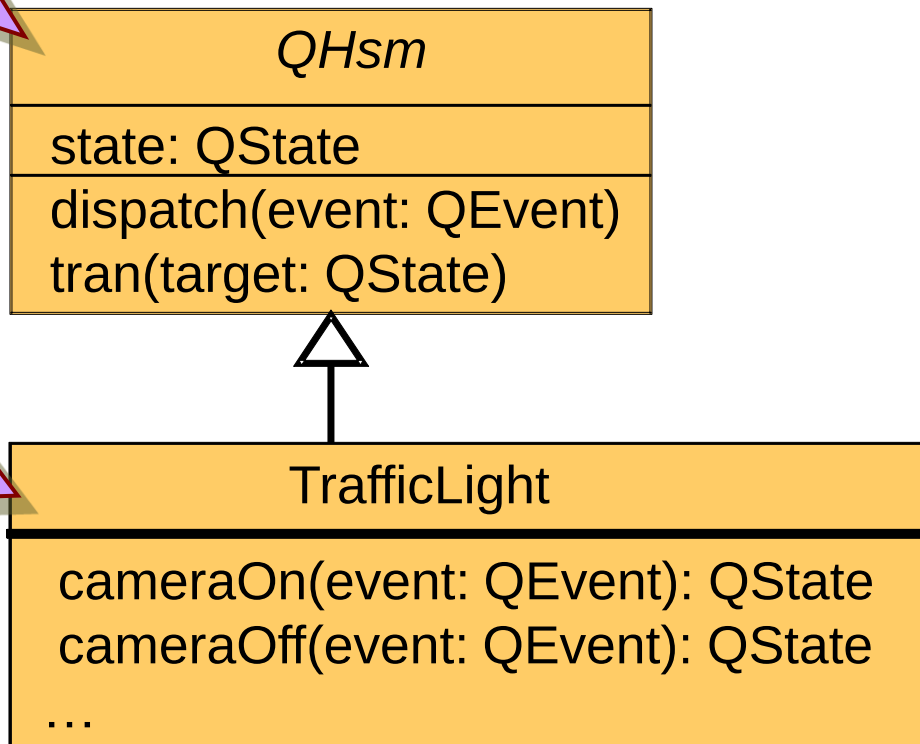
„Hierarchikus állapotgép” minta (Samek, 2002)

Absztrakt állapotgép osztály

- Aktív állapot mint tagfüggvény pointer
- Felső szintű dispatch() eseménykezelő
- Átmenet megvalósítása: tran() eseménykezelő váltás

Eseménykezelők

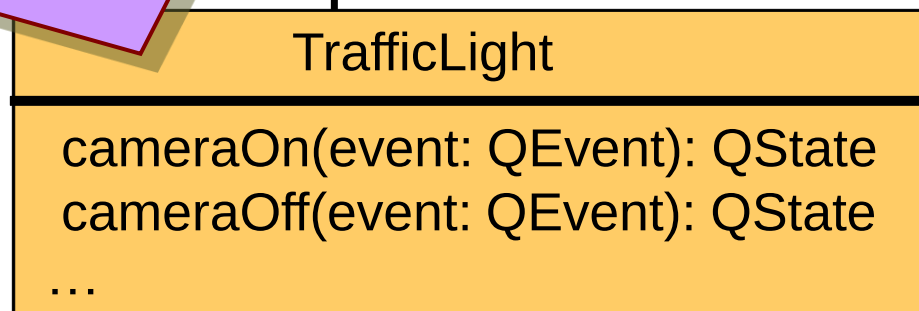
- Minden **állapothoz** külön metódus
- Az esemény **feldolgozása** (tran: eseménykezelő váltás új állapotra), vagy **felfelé továbbítása a hierarchiában**



- OR jellegű finomítást (állapothierarchia) támogat
- Konkurens viselkedést nem támogat

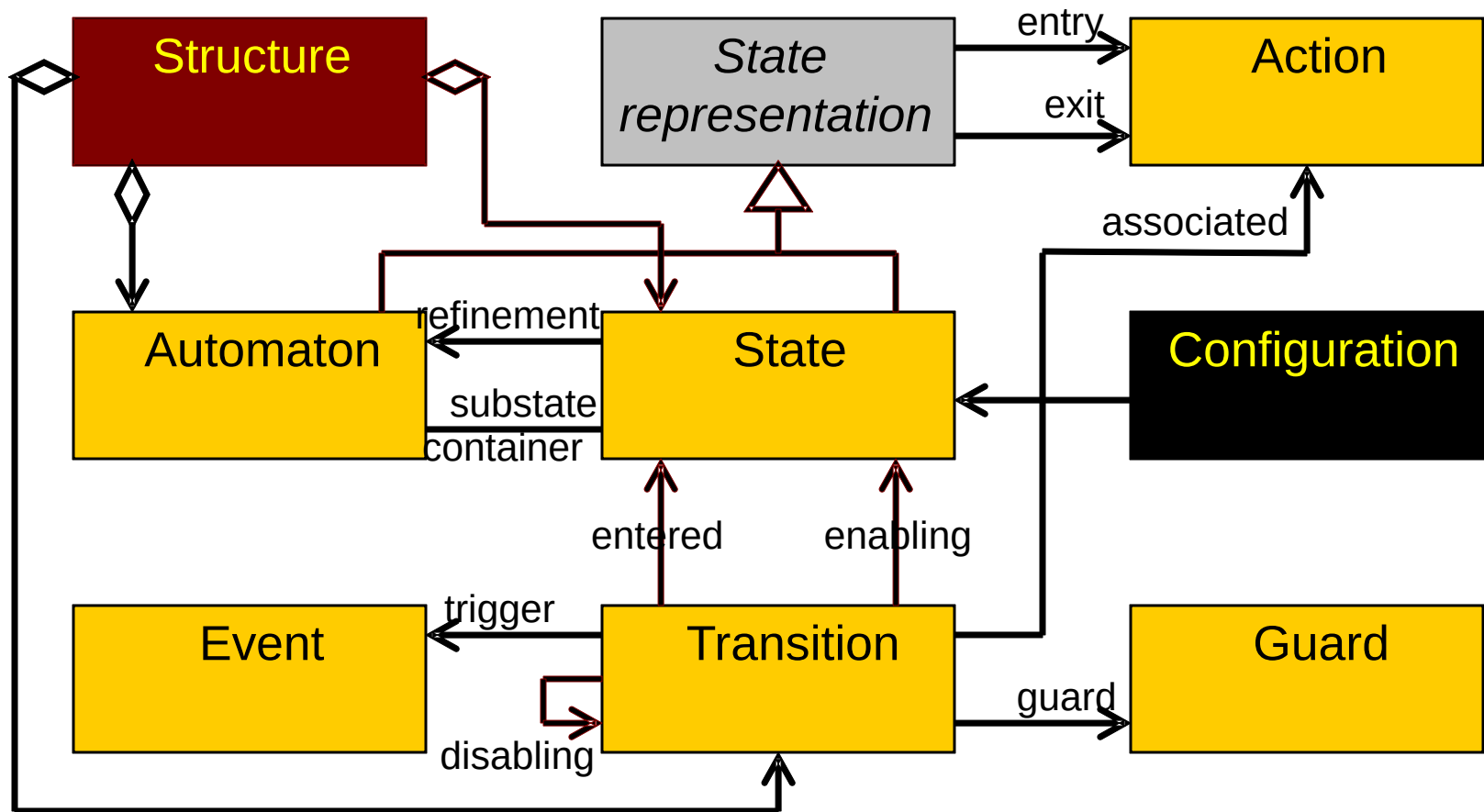
„Hierarchikus állapotgép” minta (Samek, 2002)

```
QState TrafficLight_CameraOn (TrafficLight *me, QEvent const *e) {  
    switch (e->sig) {          /* demultiplex events based on signal */  
        case ManualOff:  
            tran(TrafficLight_CameraOff); /* target state */  
            return 0; /* event handled */  
        ...  
    }  
    return (QState)TrafficLight_Red; /* designate the superstate */  
}
```



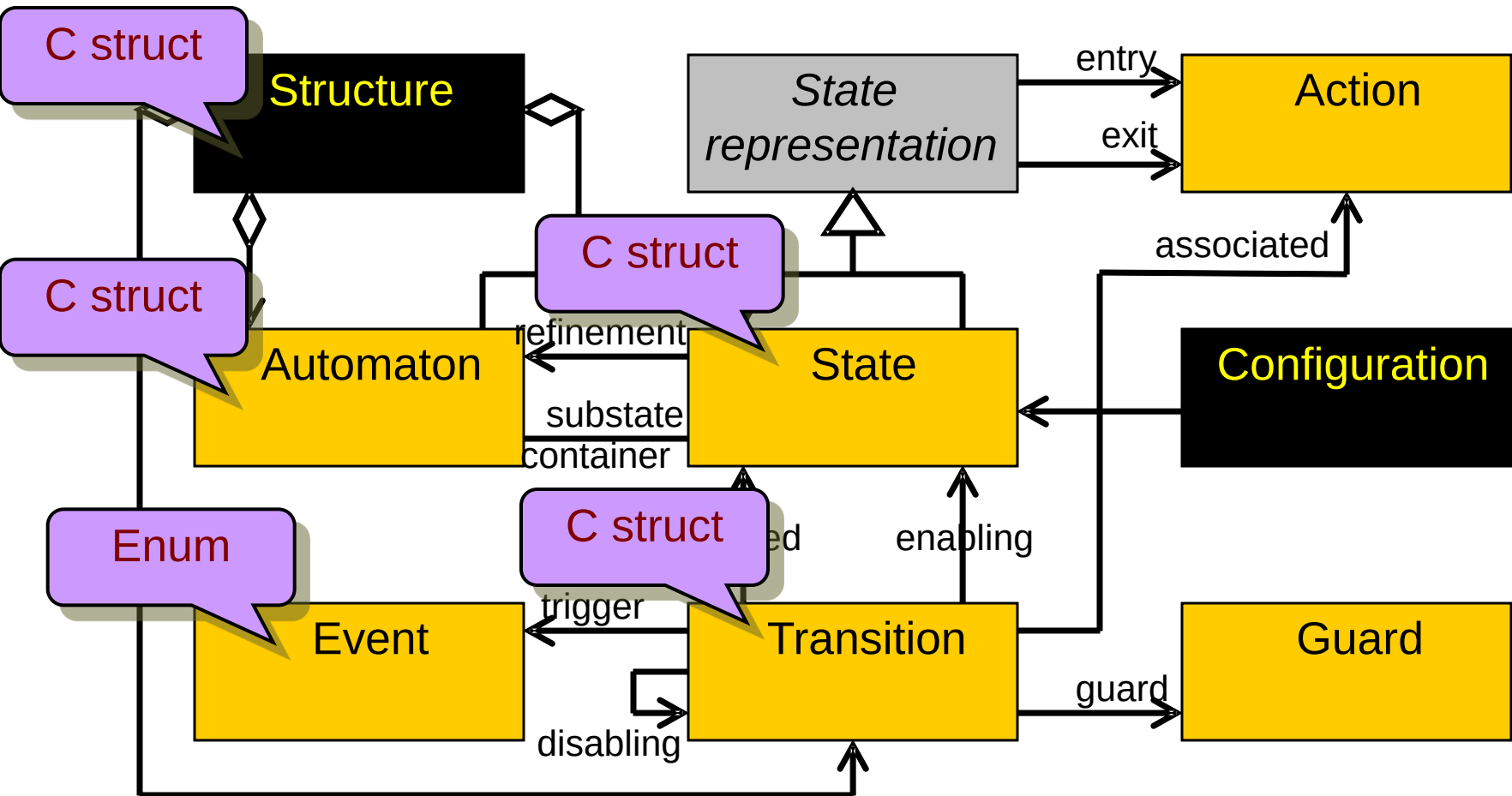
- OR jellegű finomítást (állapothierarchia) támogat
- Konkurens viselkedést nem támogat

„Kiterjesztett hierarchikus automata” minta



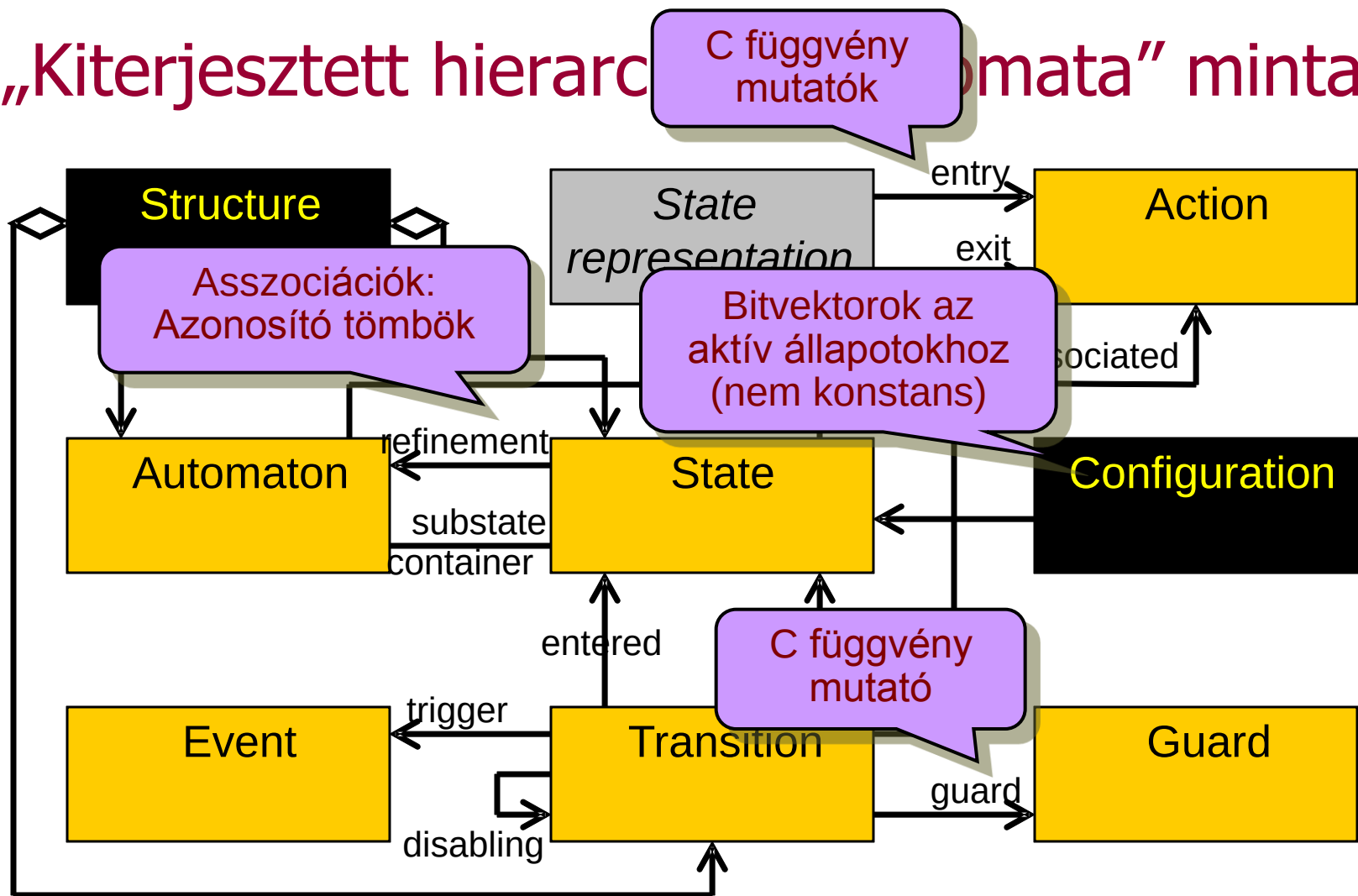
- A statikus struktúra a hierarchikus automata **metamodellje** alapján
 - <<ordered>> asszociációk (sorrend kötött a modell alapján)
 - Pl. belépési sorrend, akció sorrend, ...
- Konfiguráció: aktív állapotok kijelölése

„Kiterjesztett hierarchikus automata” minta



- Megvalósítás ANSI C-ben:
 - C struktúrák

„Kiterjesztett hierarchia” minta



- Megvalósítás ANSI C-ben:

- Függvény mutatók
- Azonosító tömbök

„Kiterjesztett hierarchikus automata” megvalósítása

- Közös „interpreter” funkció
 - Formális szemantikán alapul
 - Állapot belépés: előre kiszámolva
 - Állapot kilépés: rekurzív függvénynel (pl. konkurens régióból is)
 - Paraméterek:
 - Statikus struktúra: osztályonként egy
 - Konfiguráció: példányonként egy-egy
 - Feldolgozandó esemény
 - Visszatérési érték: Új állapotkonfiguráció
- Statikus struktúra (pl. TrafficLight)
 - Állapotok azonosítóval hivatkozhatók pointer helyett; méret csökkenthető

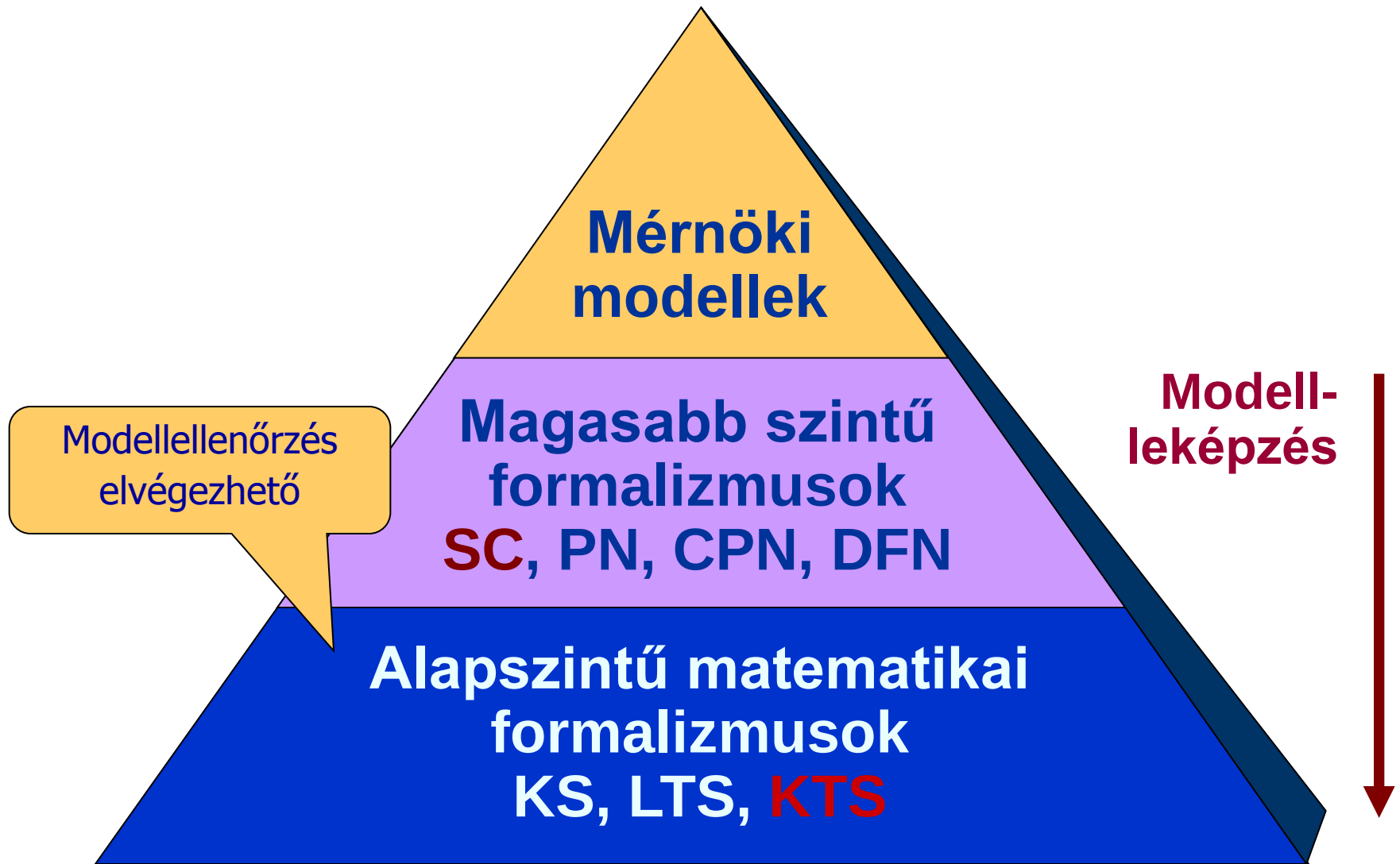
Összefoglaló

- Néhány kódgenerálási lehetőség állapottérkép modellek alapján
 - Kettős elágazás
 - „State” megvalósítási minta
 - „Hierarchikus állapotgép” (HSM) minta
 - „Kiterjesztett hierarchikus automata” (EHA) minta
- A modellelemek megvalósítása C-ben

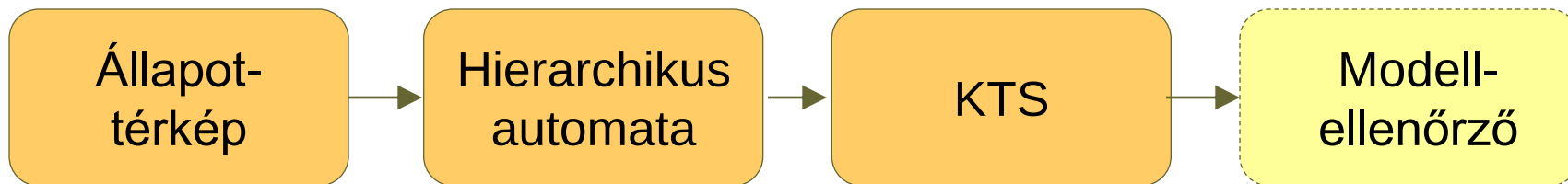
Állapottérképek egy formális szemantikája:
Leképezés KTS-re
hierarchikus automatákon keresztül

(Kitekintés – háttéranyag)

Modellek a formális ellenőrzéshez



Szemantika hozzárendelés leképezéssel

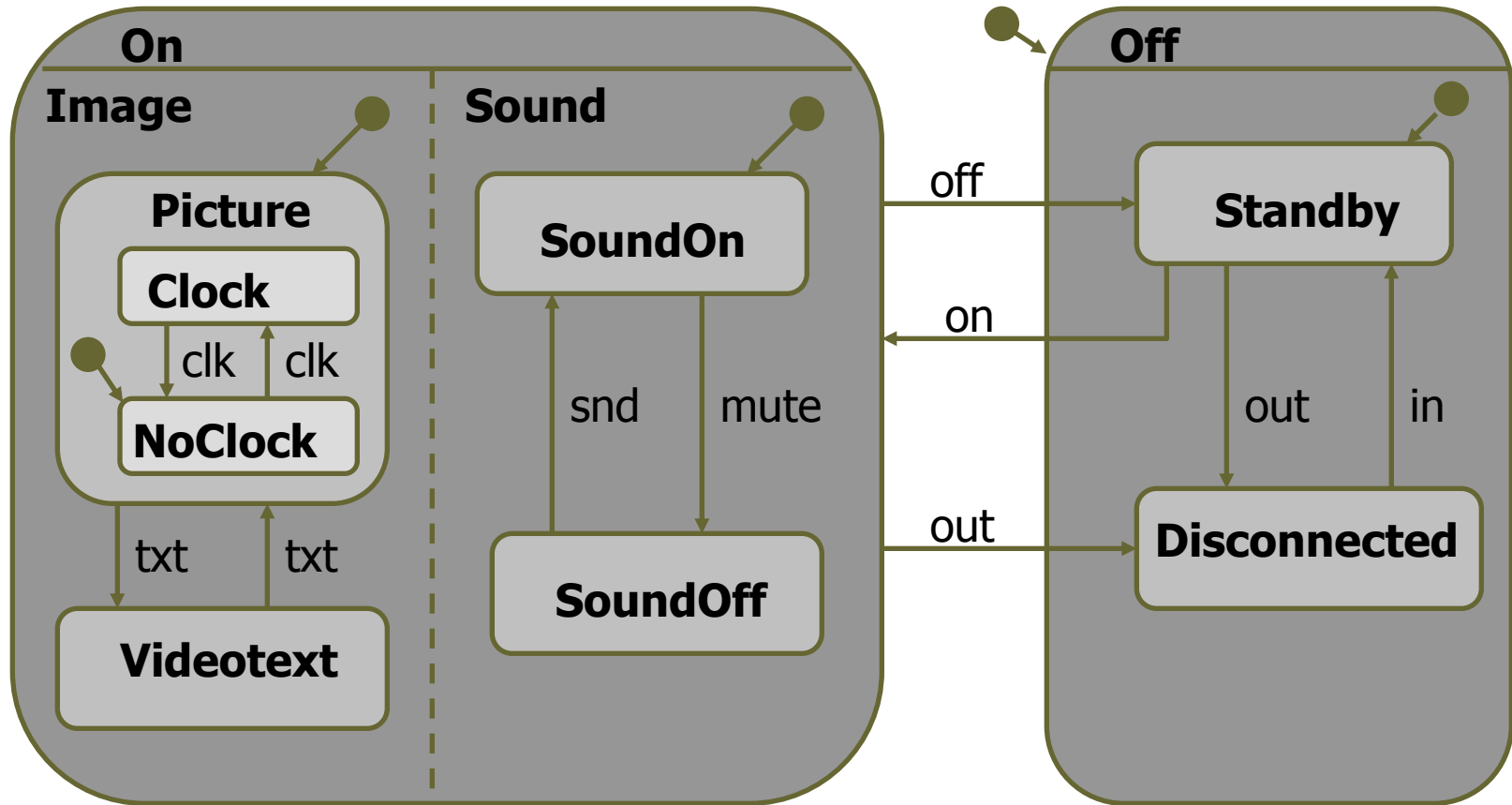


- Hierarchikus automata:
 - Absztrakt szintaxis az UML állapottérképekhez
 - Szintaktikai transzformáció
- Kripke tranzíciós rendszer
 - UML állapottérkép formális szemantikája
 - Formális verifikációra alkalmas formalizmus
- Modellellenőrzés
 - A formális szemantika egy felhasználása

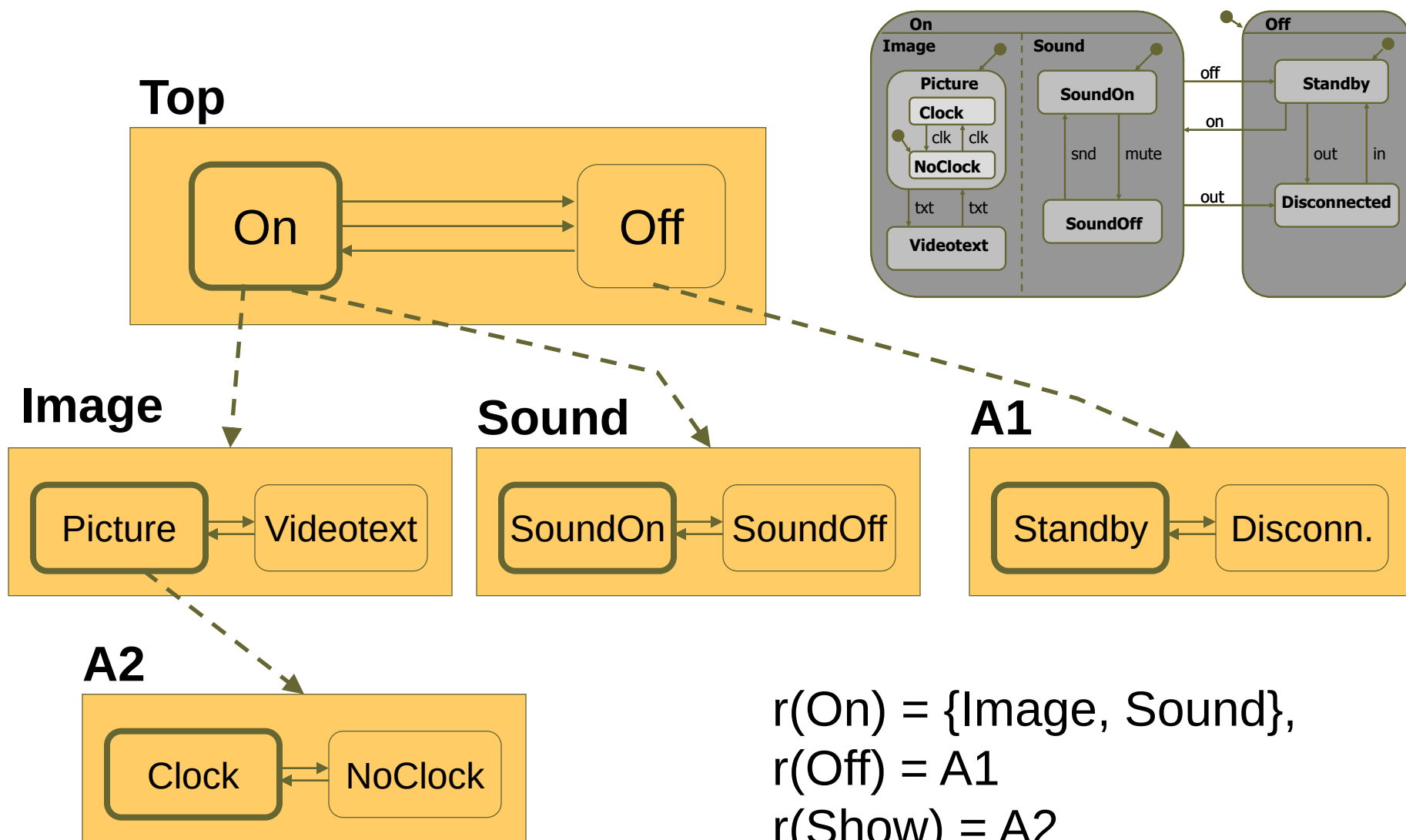
Hierarchikus automata

- Szekvenciális automata: $A = (S, \text{Act}, T)$
 - S állapotok halmaza, $s_0 \in S$ kezdőállapot
 - Act az állapotátmenetek címkéi (trigger, akció, ...)
 - T az állapotátmenetek $T \subseteq S \times \text{Act} \times S$
- Hierarchikus automata: $HA = (F, E, r, B)$
 - F szekvenciális automaták halmaza
 - E események halmaza
 - r finomítási reláció
 - B címkék halmaza

Példa: TV hierarchikus állapottérkép



Példa: TV hierarchikus automata

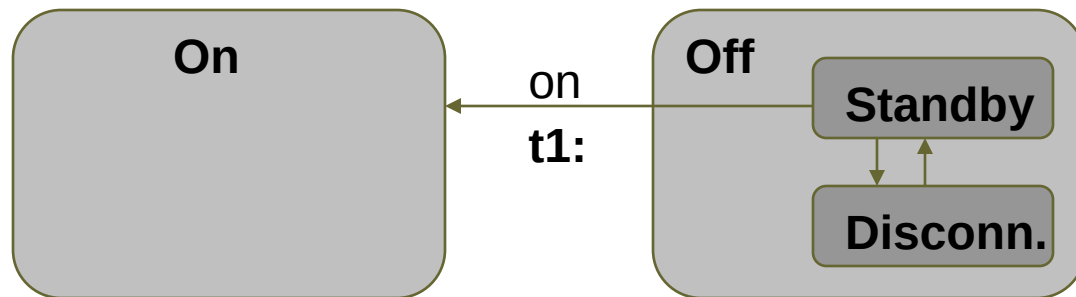


A hierarchikus automata átmeneteinek címkéi

- Az állapotfinomítási hierarchián átívelő átmenetek:
Legkisebb közös ős szintjére vannak hozva
 - Forrás állapotok (amiket elhagy) befoglalva
 - Cél állapotok (ahová belép) befoglalvaÍgy látszik, mit kell elhagyni és hová kell belépni
Konfliktusok explicit megjelennek
- HA átmenet címkéjének elemei:
 - Eredeti forrás állapot (mélyebben lehet): $SR(t)$
 - Eredeti cél állapot (mélyebben lehet): $TD(t)$
 - Trigger esemény: $TG(t)$
 - Akciók: $AC(t)$
 - Örfeltétel: $GD(t)$

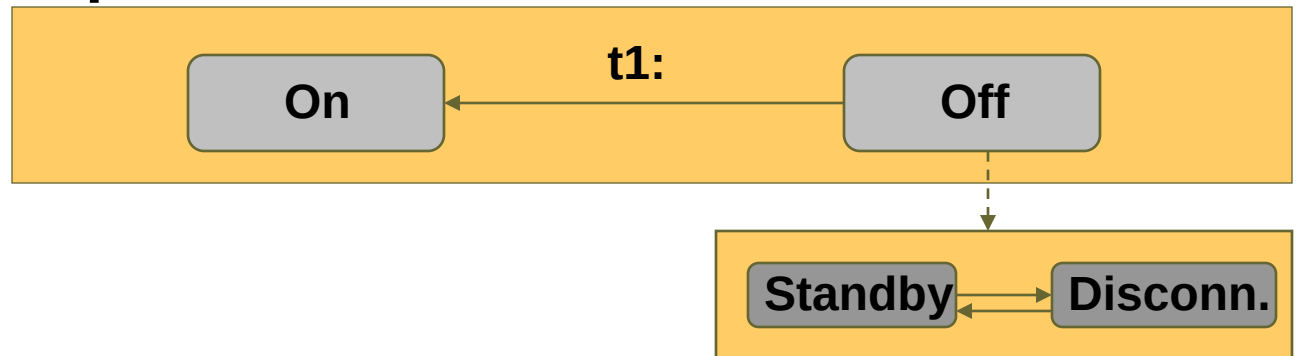
Példa: HA átmenetek címkéi

Állapottérkép
részlet:



Top

HA
részlet:



t1 címkéje: (eredeti forrás, cél, trigger, akció, őrfeltétel)

$(\{\text{Standby}\}, \emptyset, \text{on}, \emptyset, \emptyset)$

$\text{SR}(t) = \{\text{Standby}\}$

Állapot-hierarchia és prioritás

- Konfliktusok meghatározása:
 - HA modellben az átmenet forrásának jelölése: $SRC(t)$
 - t_1 és t_2 átmenetek konfliktusban vannak, ha a finomítási hierarchiában (fában) azonos út mentén vannak, azaz $SRC(t_1) = SRC(t_2)$ vagy $SRC(t_1) \in r^*(SRC(t_2))$ vagy $SRC(t_2) \in r^*(SRC(t_1))$ itt r^* reláció kiterjesztett az alatta lévő teljes hierarchiára
- Állapotátmenetek prioritása:
 - A mélyebben lévő $SR(t)$ lehet a meghatározó
 - Ha nincs $SR(t)$ akkor $SRC(t)$, egyébként $SR(t)$ határozza meg:
$$\{s \mid s \in SRC(t) \text{ ha } SR(t)=0\} \cup SR(t)$$

Prioritás kezelése

- Egy állapotátmenet tüzelésének feltétele:
 - Nincs nagyobb prioritású engedélyezett átmenet a többi szekvenciális automatában
- HA hierarchia alsóbb szintjein:
 - Nagyobb prioritású átmenetek vannak
- HA hierarchia felsőbb szintjein:
SR miatt lehet nagyobb prioritású átmenet!
 - Bevezetett jelölés: $A \uparrow P$,
 P a „felülről származó”, de nagyobb prioritású engedélyezett átmenetek halmaza,
mint **kényszer** az A automata tüzelésekor

A környezet modellezése

Lehetőségek:

- **Zárt rendszer:**
 - A környezetet is modellezzük
 - Csak állapotgép(ek) állít(anak) elő új eseményeket
- **Nyitott rendszer:**
 - A környezetet nem modellezzük
 - Érkeznek „külső” események, amiknek a forrása nincs a modellben

Továbbiakban:

- Zárt rendszerrel foglalkozunk
- Adatkezeléstől eltekintünk (csak események)
- Emlékező állapotok nincsenek

Szemantika: Kripke tranzíciós rendszer

KTS = $(S, \rightarrow, L)$ és Act

- S állapotok halmaza: (C, Q)
 - C : állapotkonfigurációk („top” HA-tól indulva lefelé)
 - Q : eseménysor állapotok
 - s_0 kezdőállapot: (C_0, Q_0)
kezdő állapotkonfiguráció és eseménysor
- L állapot címkézés:
 - C állapotkonfigurációkhoz összegyűjtve
- $\rightarrow$ átmenet reláció (lépés): $(C, Q) \rightarrow (C', Q')$
 - (1) $Q'' = \text{remove}(e, Q)$
 - (2)
$$\frac{\text{Top} \uparrow \emptyset :: (C, \{e\}) \rightarrow^T (C', Q')}{(C, Q) \rightarrow (C', \text{join}(Q', Q''))}$$

Szabálykészlet

- Kripke-struktúra $\rightarrow$ relációjának megadása:
HA $\rightarrow^T$ relációja segítségével történt
 - T-ben lévő átmenetek tüzelnek a HA-ban
 - Mely átmenetek lesznek T-ben?
- HA $\rightarrow^T$ relációjának megadása:
A szekvenciális automaták tüzelése 3 szabály szerint történhet:
 - Lépés („progress”)
 - Kompozíció („composition”)
 - Henyélés („stuttering”)

1. A lépés szabálya

- Egy $A \uparrow P$ automata t átmenete lokálisan tüzel
- Feltételek:
 - A t átmenet az automatában engedélyezett:
 $SR(t)$, $SRC(t)$ az aktuális konfiguráció része
 - Nincs sem az $SRC(t)$ alatt lévő HA-kban,
sem P -ben (a felső szintű automatákban)
nagyobb prioritású engedélyezett átmenet
- Következmény: $A \uparrow P :: (C, \{e\}) \rightarrow^T (C', Q')$
 - T tüzelő átmenetek: $T = \{t\}$
 - C' új konfiguráció: t és $TD(t)$ határozza meg
 - Q' új események: $AC(t)$

2. A kompozíció szabálya

- Egy $A \uparrow P$ automata nem tüzel, hanem „összefogja” az alacsonyabb szintű automaták tüzeléseit
- Feltételek:
 - Az alatta lévő A_i HA-k tüzelnek:
 $A_i \uparrow P' :: (C, \{e\}) \rightarrow^{T_i} (C_i', Q_i')$
 - Vagy ha $\cup T_i = \emptyset$, akkor lokálisan sem tüzel (P miatt)
- Következmények:
 - Tüzelő átmenetek: $T = \cup_j T_j$
 - Új konfiguráció: $\{s\} \cup_j C_i'$ (s az eredeti állapot)
 - Új eseménysor: $\text{join_all}_i (Q_i')$

3. A henyélés szabálya

- Egy $A \uparrow P$ automata nem tüzel, állapota nem változik
- Feltételek:
 - Minden lokálisan engedélyezett átmenetnél van nagyobb prioritású P -ben, vagy
 - Nincs alsóbb szintű automata, amit összefoghat
- Következmények:
 - Tüzelő átmenetek: $T = \emptyset$
 - Új állapot: marad a régi...
 - Új eseménysor: nincs új generált esemény

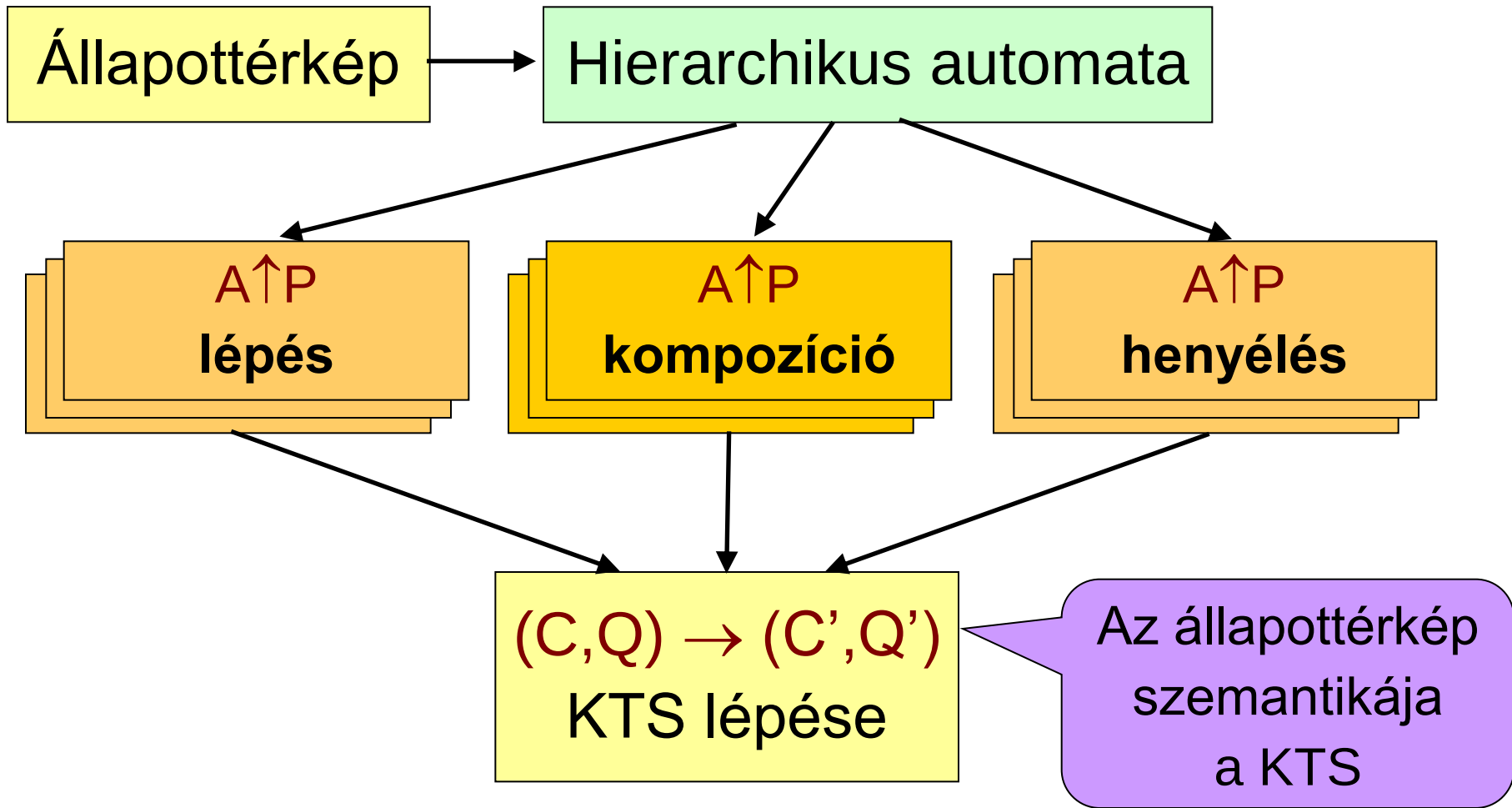
Szabályok használata

Rekurzív jellegű feltételek:

- Kompozíció szabálya
- Egy adott automata tüzelése függ
 - Hierarchiában alatta elhelyezkedő HA-któl: van-e engedélyezett átmenetük
 - Hierarchiában fölötte elhelyezkedő HA-któl; **P** mint "felülről származó" kényszer (ld. kompozíció)

Minden átmenet tüzel, amire van érvényes szabály

A szabályok áttekintése



KTS: ((C,Q), $\rightarrow$, L)