Árnyaló-visszavetítés, anyagrendszer, game object model

Szécsi László

3D Grafikus Rendszerek

5. előadás

Visszavetítés – reflection betekintéssel - introspection

```
// Java !!!!!

Foo foo = new Foo();

Method[] m = foo.getDeclaredMethods();

for(int i = 0; i < m.length; i++) {
  System.out.println("method = " + m[i].toString() +
     "#params: " + m[i].getParamaterCount());
}
```

Visszavetítés

- lehetővé teszi a program számára, hogy a saját szerkezetét futásidőben vizsgálja
  - Type
  - Class
  - Member
  - Method
  - Field

  típusok és megfelelő lekérdező metódusaik

Mire jó?

- program írásakor ismeretlen kód
  - extension, plugin
  - libet írunk, akkor a user kód
- automatizálás
  - minden tagváltozót csatoljunk ki a GUI-ra

Visszavetítés módosíthatósággal

```
//JavaScript
var obj {a:1, b:{}};

for(var key in obj){
  if(obj.hasOwnProperty(key)){
    console.log(obj[key]);
  }
}

a.addedProperty = 56;
Object.defineProperty(obj, "nonwritableprop",
     {value:{ m:"változtatható" }});
a.nonwritableprop = "Falrahányt borsó"; //hatástalan
a.nonwritableprop.m = 34; // ez működik
```

Mire jó?

- metaprogramozás
  - a program írja a programot
  - C++: a template-rendszer lát bele a kódba és ő tudja módosítani
    - nem saját magára van visszavetítve a C++ kód
    - ez is metaprogramming
    - egyfajta reflection (compile-time reflection)
- az is reflection, ha az egyik program lát bele a másik program szerkezetébe!

Árnyaló-visszavetítés
Shader reflection

- csúcspont- és képpontárnyalók
  - bementek
  - kimenetek
  - utasításszám
  - erőforrások kapcsolódási pontjai
  - **uniform változók**

Mire jó?

- GUI-ra kirakni a uniformok értékeit
- háttérstruktúrák automatikus létrehozása
  - pl. minden uniformnak (vagy konstansbuffernek) legyen másolata a rendszermemóriában
  - ezt egyszer/igény szerint beállítjuk
  - amikor kell, az aktuális shader/program megfelelő változójába másolódik

  - D3D Effect framework egyik szerepe

ID3D11ShaderReflection

```
device->CreateVertexShader(
  byteCode->GetBufferPointer(),
  byteCode->GetBufferSize(),
  nullptr,
  shader );

ID3D11ShaderReflection* shaderReflector = nullptr;

D3DReflect(
  shaderCode->GetBufferPointer(),
  shaderCode->GetBufferSize(),
  IID_ID3D11ShaderReflection,
  (void**) &shaderReflector);
```

Konstans bufferek feldolgozása

```
D3D11_SHADER_DESC shaderDesc;
shaderReflector->GetDesc(&shaderDesc);

for(uint i=0; i<shaderDesc.ConstantBuffers; i++) {
  ID3D11ShaderReflectionConstantBuffer* cbr =
      shaderReflector->GetConstantBufferByIndex(i);
  D3D11_SHADER_BUFFER_DESC cbDesc;
  cbr->GetDesc(&cbDesc);

  //pl. másolat létrehozása a rendszermemóriában
}
```

OpenGL/WebGL reflection

- nem objektum-orientált
- de azért vannak lekérdezőfüggvények

WebGL reflection

```
val nUniforms = gl.getProgramParameter(glProgram,
         WebGLRenderingContext.ACTIVE_UNIFORMS) as Int
for(i in 0 until nUniforms){
  val glUniform = gl.getActiveUniform(glProgram, i)!!

  uniformDescriptors.add( UniformDescriptor(
        name = glUniform.name,
        type = glUniform.type,
        size = glUniform.size,
        location =
           gl.getUniformLocation(glProgram, glUniform.name)
    ))
  }
```

Játékmotorok

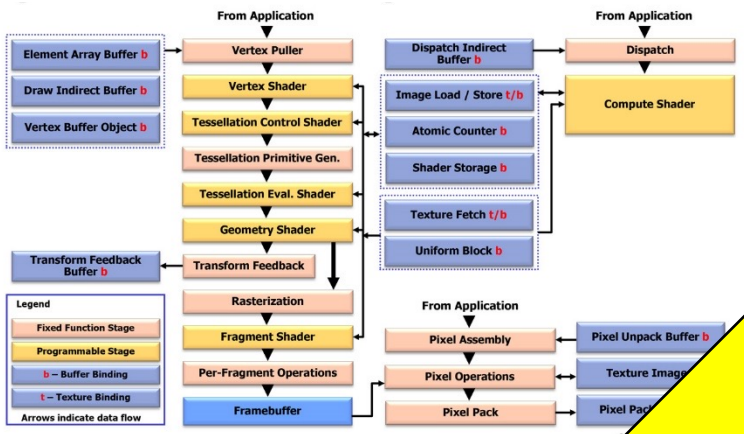
Legyen gyors, sima animáció

Legyen bővíthető

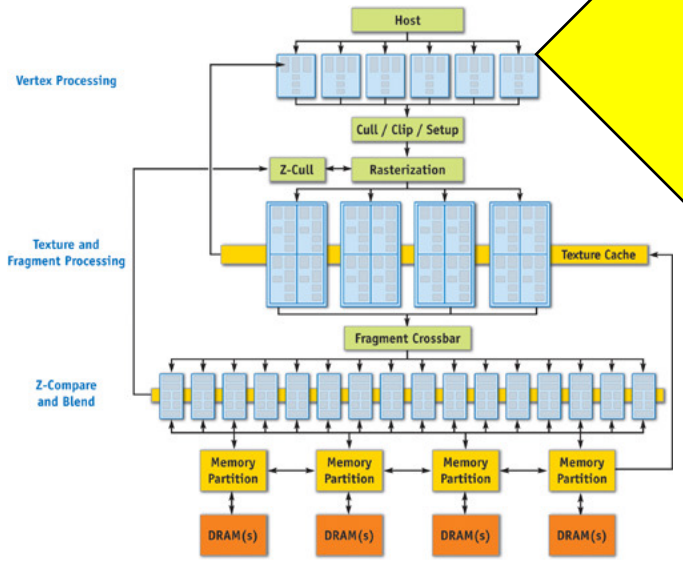
[megjelenítés, fizika, hang, szkript, hálózat, karakteranimáció, MI, GUI, szerkesztő]

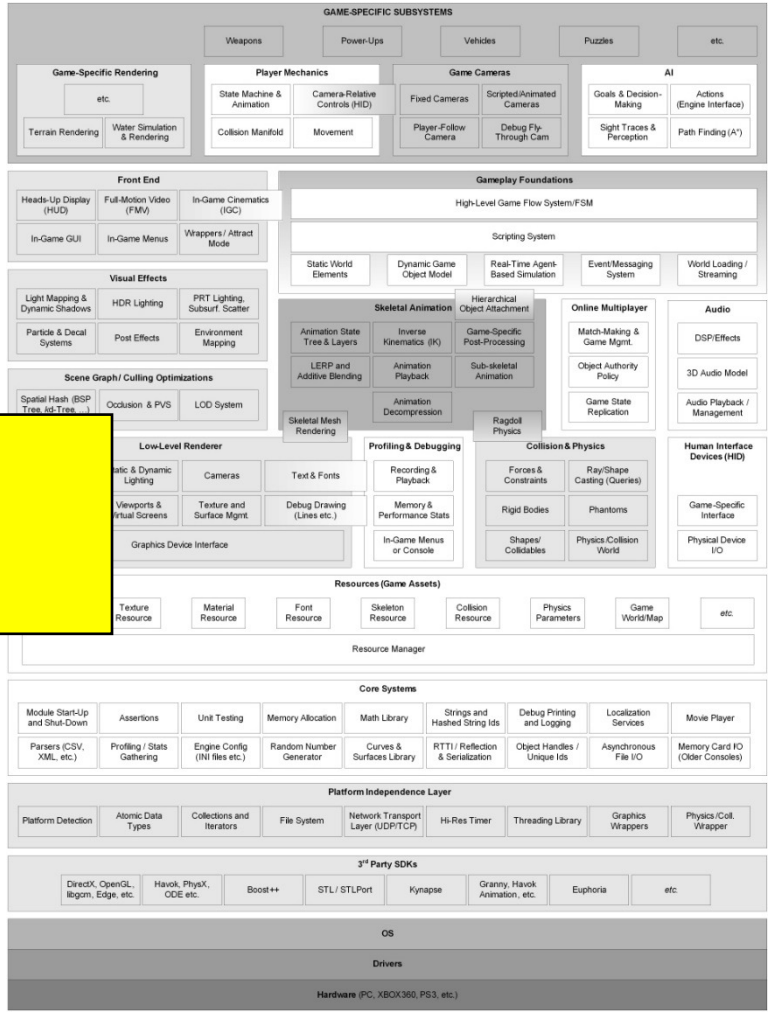
Tartalom, játéklogika könnyen illeszthető legyen

Komplexitás: GPU vs. színtérmenedzsement



?





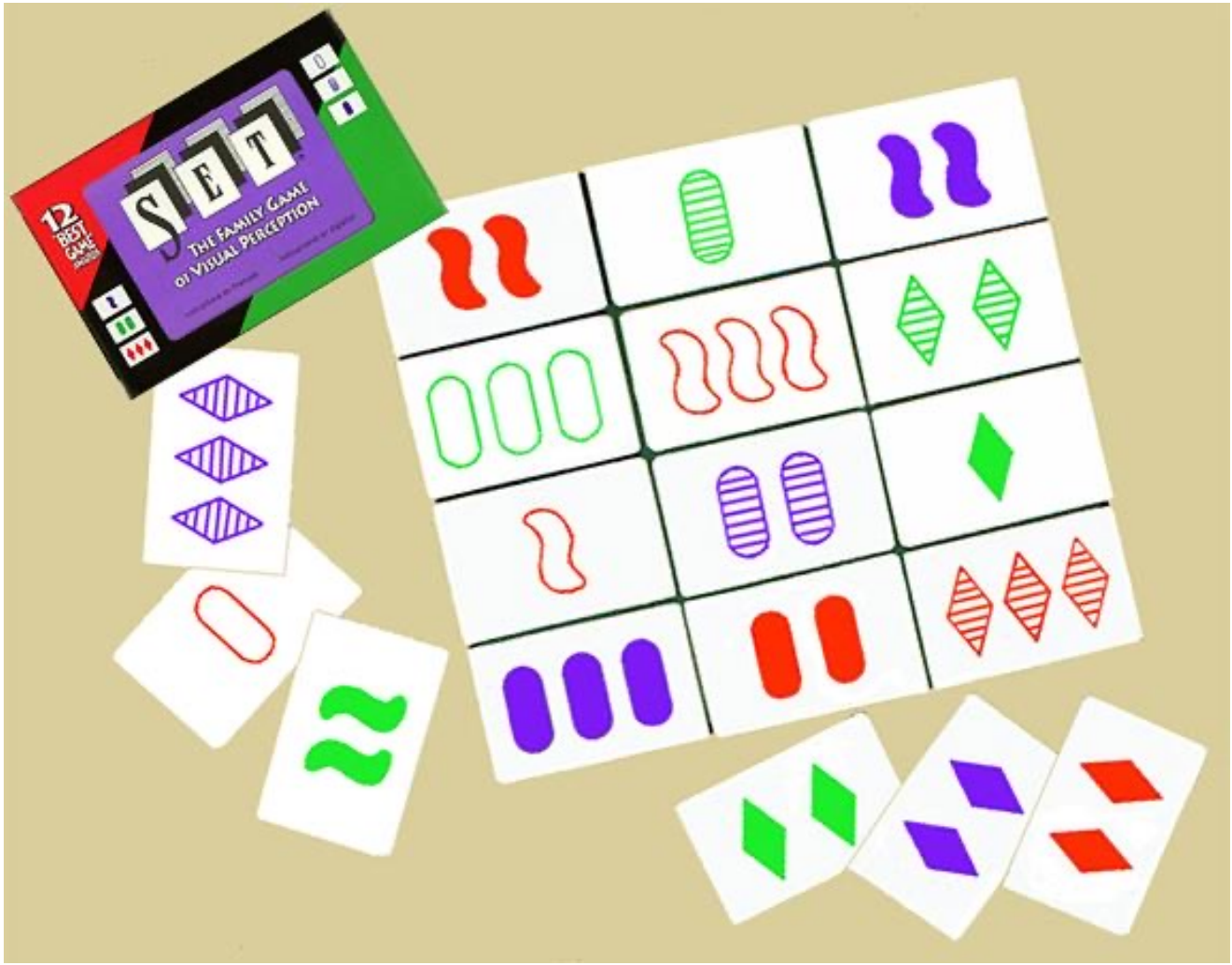
Rugalmasság vs. teljesítmény

- Képességek, jellemzők széles spektruma
- Bővíthetőség
- Szabad kombinálhatóság
- Diverzitás
- Párhuzamosítás
- Többszörös végrehajtások kiiktatása
- Újrafelhasználás
- Uniformitás

Komponens-alapú objektummodell

- Monolitikus öröklődési hierarchia helyett
  - Új képesség beillesztése közös ősbe???
- Entitások
  - Lazán csatolt
  - Komponensek halmazai

Entitások



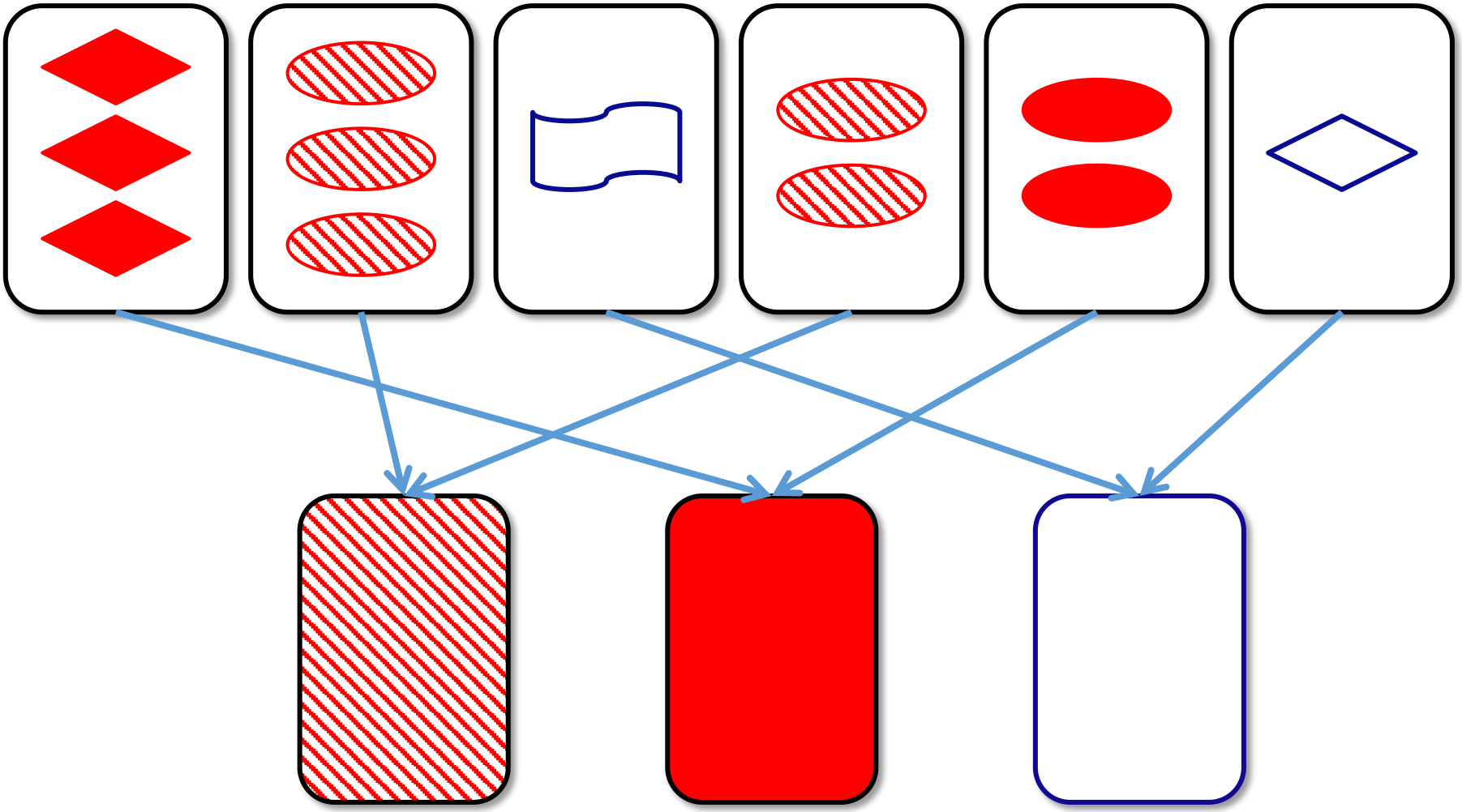
GPU állapotváltozók

OMSetBlendState
OMSetDepthStencilState
OMSetRenderTargets
OMSetRenderTargetsAndUnorderedAccessViews
PSSetConstantBuffers
PSSetSamplers
PSSetShader
PSSetShaderResources
RSSetScissorRects
RSSetState
RSSetViewports
SOSetTargets
VSSetConstantBuffers
VSSetSamplers
VSSetShader
VSSetShaderResources

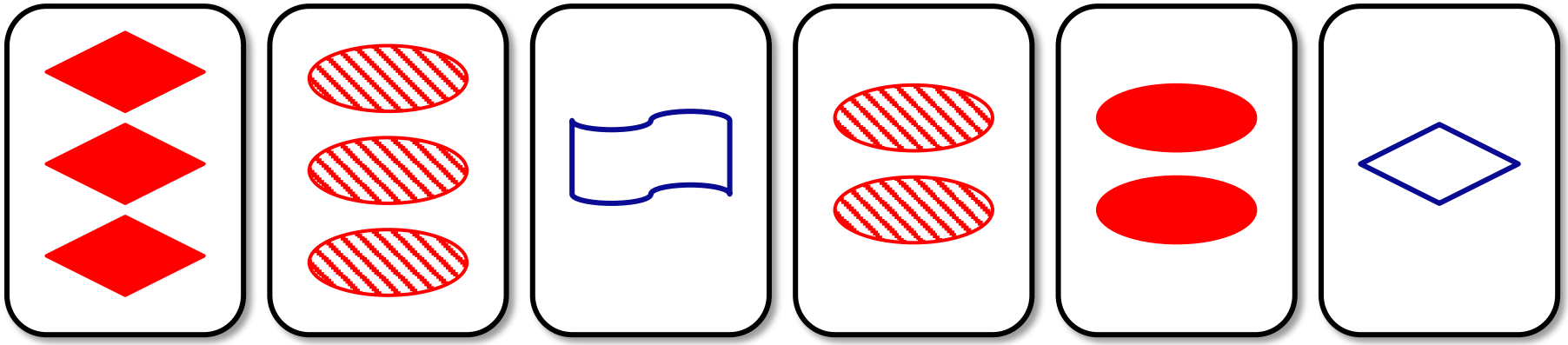
CSSetConstantBuffers
CSSetSamplers
CSSetShader
CSSetShaderResources
CSSetUnorderedAccessViews
DSSetConstantBuffers
DSSetSamplers
DSSetShader
DSSetShaderResources
GSSetConstantBuffers
GSSetSamplers
GSSetShader
GSSetShaderResources
HSSetConstantBuffers
HSSetSamplers
HSSetShader
HSSetShaderResources
IASetIndexBuffer
IASetInputLayout
IASetPrimitiveTopology
IASetVertexBuffers

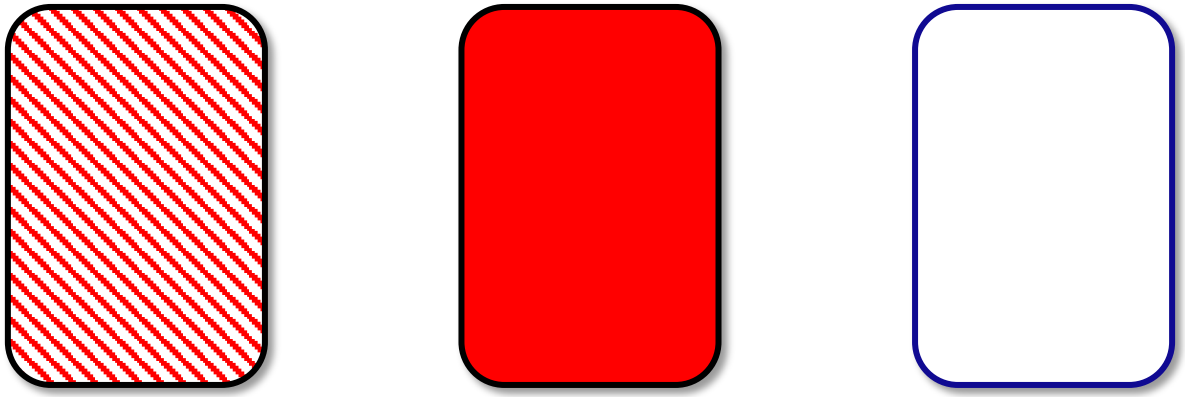
37-féle jellemzőhalmaz!

Anyagrendszerek

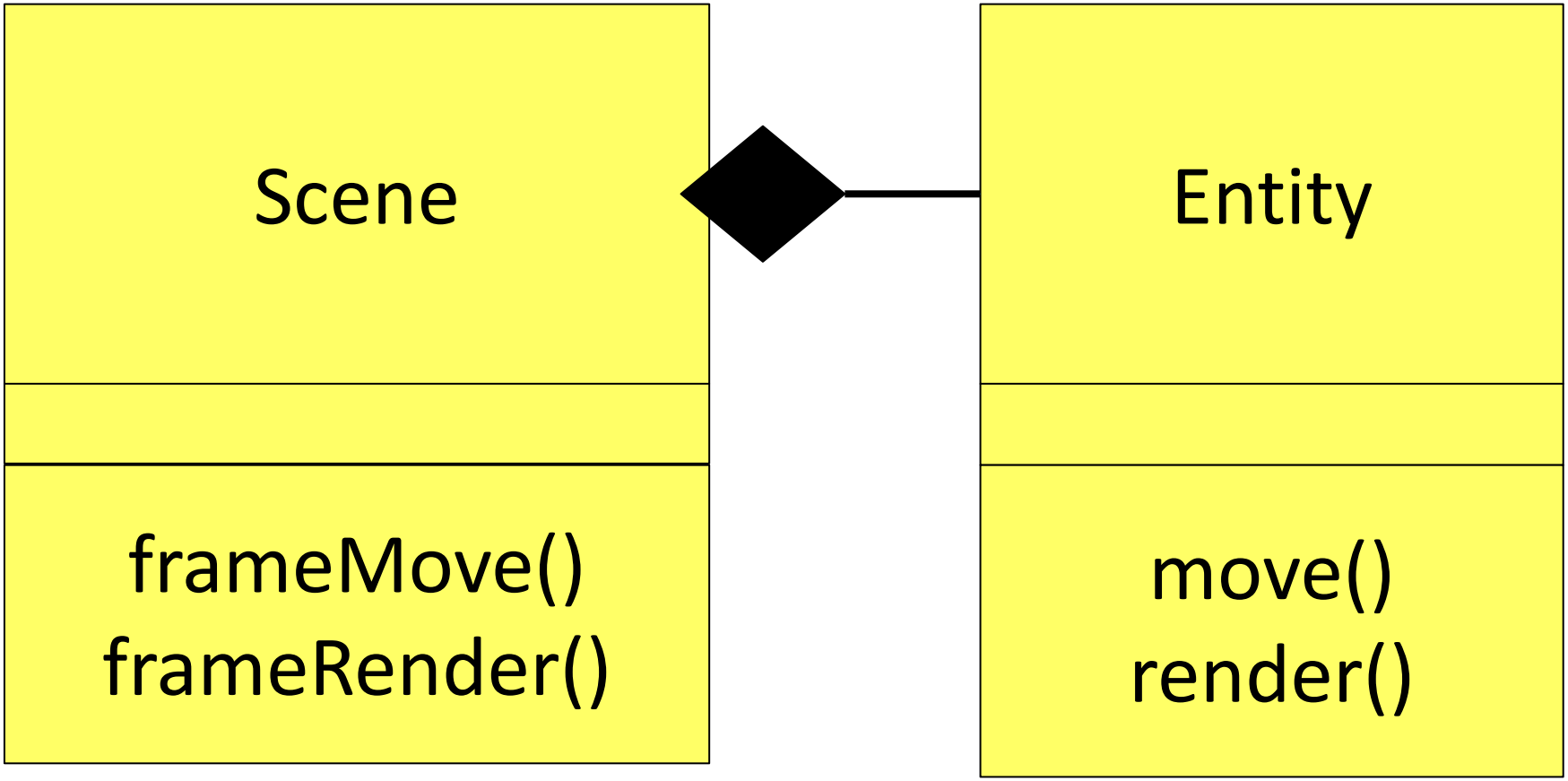


Renderqueue





GameObject / Entity

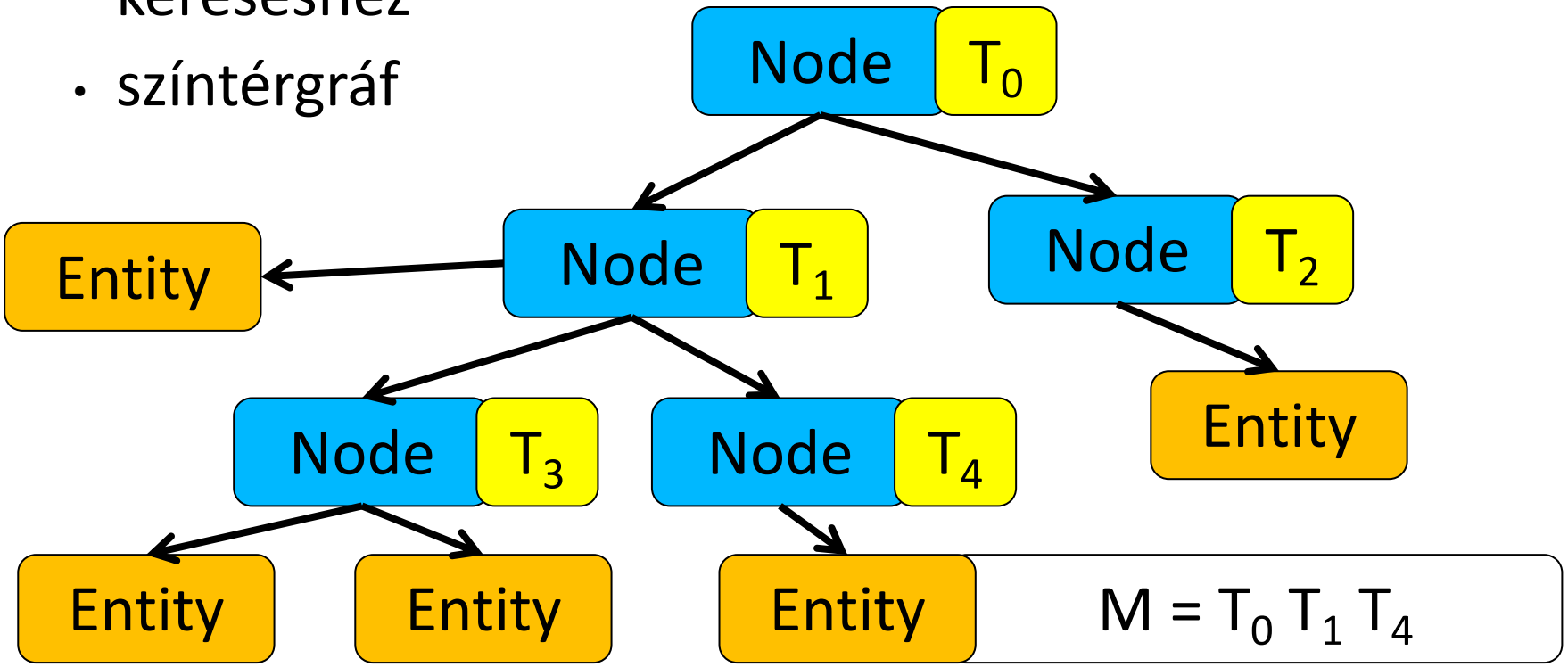


Mi az entitás?

- színtér = entitások + speciálisan kezelt elemek
  - renderable, fizikailag animált, játékos/AI vezérelt
  - minél több funkciót adunk az entitásnak, annál kevésbé konkrét
  - ég? kamerák? fényforrások? trigger zónák?

Entitástároló adatszerkezetek

- név szerinti, szkripteléshez
- térefelosztás (pl. octree) ütközéshez, közelségi kereséshez
- színtérgráf



Runtime object model

- milyen adatszerkezet, típusok, osztályok írják le:
  - a játékban létező dolgokat = entitások
  - ezek tulajdonságait = attributes, properties
  - ezek viselkedését = behavior

Object model alaptípusok

- Objektum-központú
  - minden objektumra tároljuk a tulajdonságokat
  - különbségek kezelésre: öröklődés
- Tulajdonság-központú
  - minden tulajdonságra tároljuk, hogy milyen azonosítójú entitásnak van ilyen, és milyen értékű
  - különbségek kezelésre: egyszerűen más tulajdonságokat kapnak

Objektum-centrikus

- entitás osztály példányai
  - attribútumok tagváltozók
  - viselkedést a metódusok szabják meg
- új attribútum vagy viselkedés
  - leszármaztatás
    - virtuális függvények
  - delegálás

Tulajdonság-centrikus

- az entitást csak egy azonosító jelenti
- minden elképzelhető tulajdonsághoz van egy táblázat
  - azonosító alapján előkereshető, hogy pl.
    - mennyi az életereje
    - milyen háromszögháló modell tartozik hozzá
- viselkedést az szabja meg, milyen tulajdonságok vannak - hardcoded
  - ha van Health property, sérülhet és megsemmisülhet

Rajzolás objektum-centrikusan

- foreach entity
  - render

Rajzolás tulajdonság-centrikusan

- ~~foreach entity~~
  - ~~if(hasProperty(1)) render1();~~
  - ~~else if(hasProperty(2)) render2();~~
- foreach property1 render1();
- foreach property2 render2();

Objektum-centrikus: hajóverseny

- milyen entitások lehetnek
  - játékos vezérelte hajó
  - ellenséges (AI vezérelte) hajó
  - parton heverésző pingvin
  - vízfelület
  - rámpák
  - vízesés
  - fröccsenő víz, füst (részecskerendszerek)
  - terep, fák, statikus geometria

Biztos, hogy más dolgok nem entitások?

- fények?
- kamerák?
- a zóna amibe ha belépünk kinyílik az ajtó?

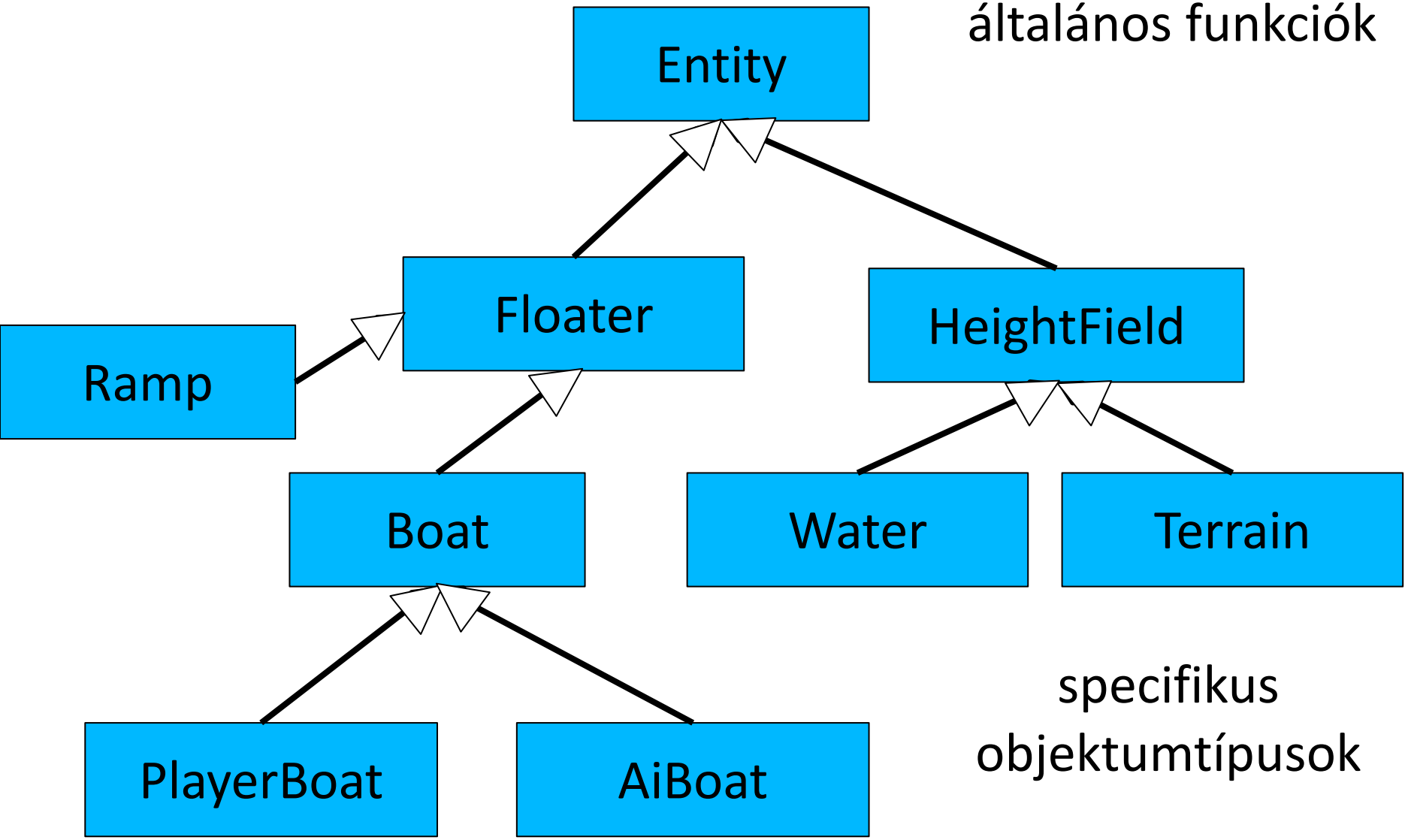
- az update/render értelemben nem, de lehet olyan architektúra, ahol minden ilyesmi egységesen van kezelve, és a klasszikus renderelhető entitás csak egy altípus
- pl. Shark3D

Hajóverseny

- ezeknek mind kell
  - render
  - update (control + animate)
- Entity ősosztály, virtuális függvények implementálva
- hasonlót ne kelljen többször
  - hierarchia

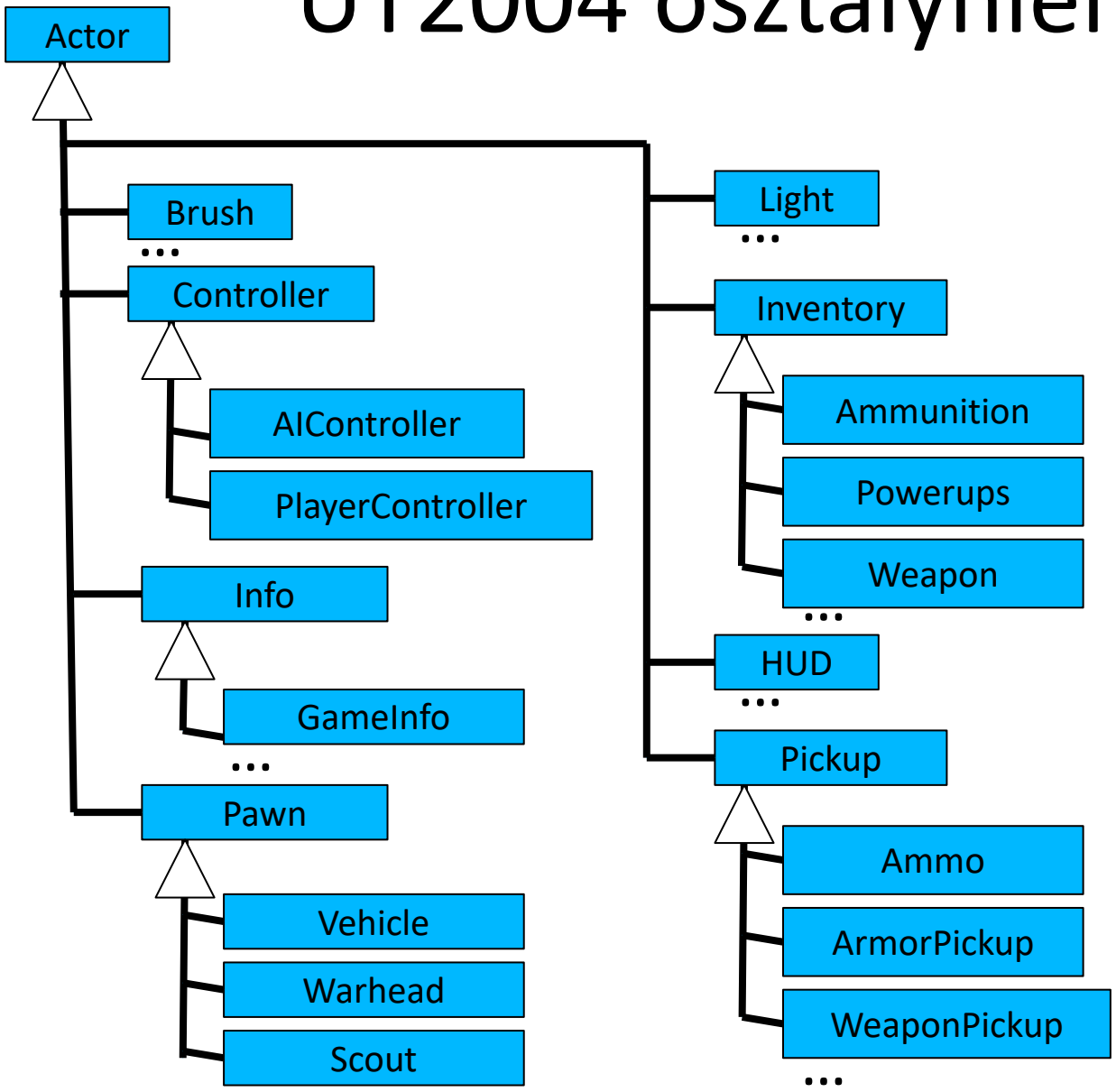
Osztályhierarchia (részlet)

általános funkciók



specifikus objektumtípusok

UT2004 osztályhierarchia



From:
Game Technology
by Don Fussel

Monolitikus osztályhierarchia

- egyetlen ősosztály
- minden sajátosság, eltérés kétféle entitás között egy őröklődés
- a ‘természetes taxonómia’ követése

- SOK osztály, nehezen követhető
- ránézünk egy osztályra, és csak akkor értjük, ha végignézzük a teljes láncban az összes szülőjét
- több taxonómia?

Taxonómia

- Kingdom: Animalia
- Phylum: Chordata
- Subphylum:Vertebrata
- Class: Reptilia
- Order: Squamata
- Suborder: Serpentes
- Family: Viperidae
- Subfamily: Viperinae
- Genus: ***Vipera***

Többdimenziós taxonómia

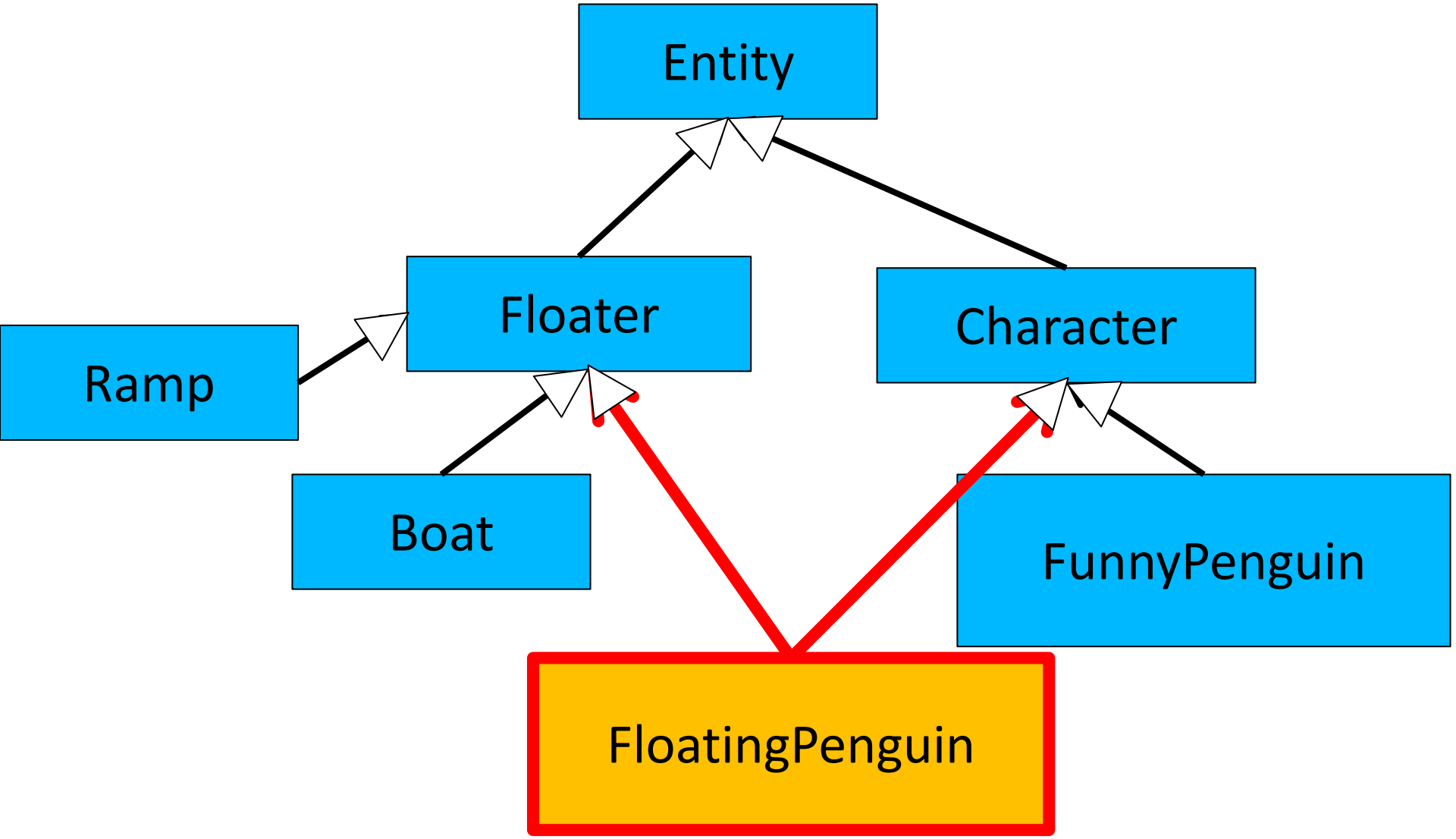
Kingdom: Animalia
Phylum: Chordata
Class: Mammalia
Infraclass: Marsupialia
Order: Diprotodontia
Family: Phascolarctidae
Genus: *Phascolarctos*
Species: ***P. cinereus***

- visibile/invisible
- cute/ugly
- colorful/plain
- grey/blue/…

Többdimenziós taxonómia játékban

- átlátszó/szilárd
- árnyékot vet/nem vet
- van/nincs fizikai szimuláció
- mászik/úszik/repül
- sebezhető/sebezhetetlen
- üt/vág/lézert lő/lövedéket lő
  - milyen lövedéket? célkövető/stb.

Probléma a hajós játékban



Miért kerüljük a többszörös öröklést?

- kétértelműség
  - Ha a Floater és a Character egyaránt megvalósítják az update() metódust, melyiket örökli a FloatingPenguin?
  - ezt jelzi a fordító, FloatingPenguin-ben is meg kell valósítani a metódust
- konstruktorok sorrendje
- összetettség

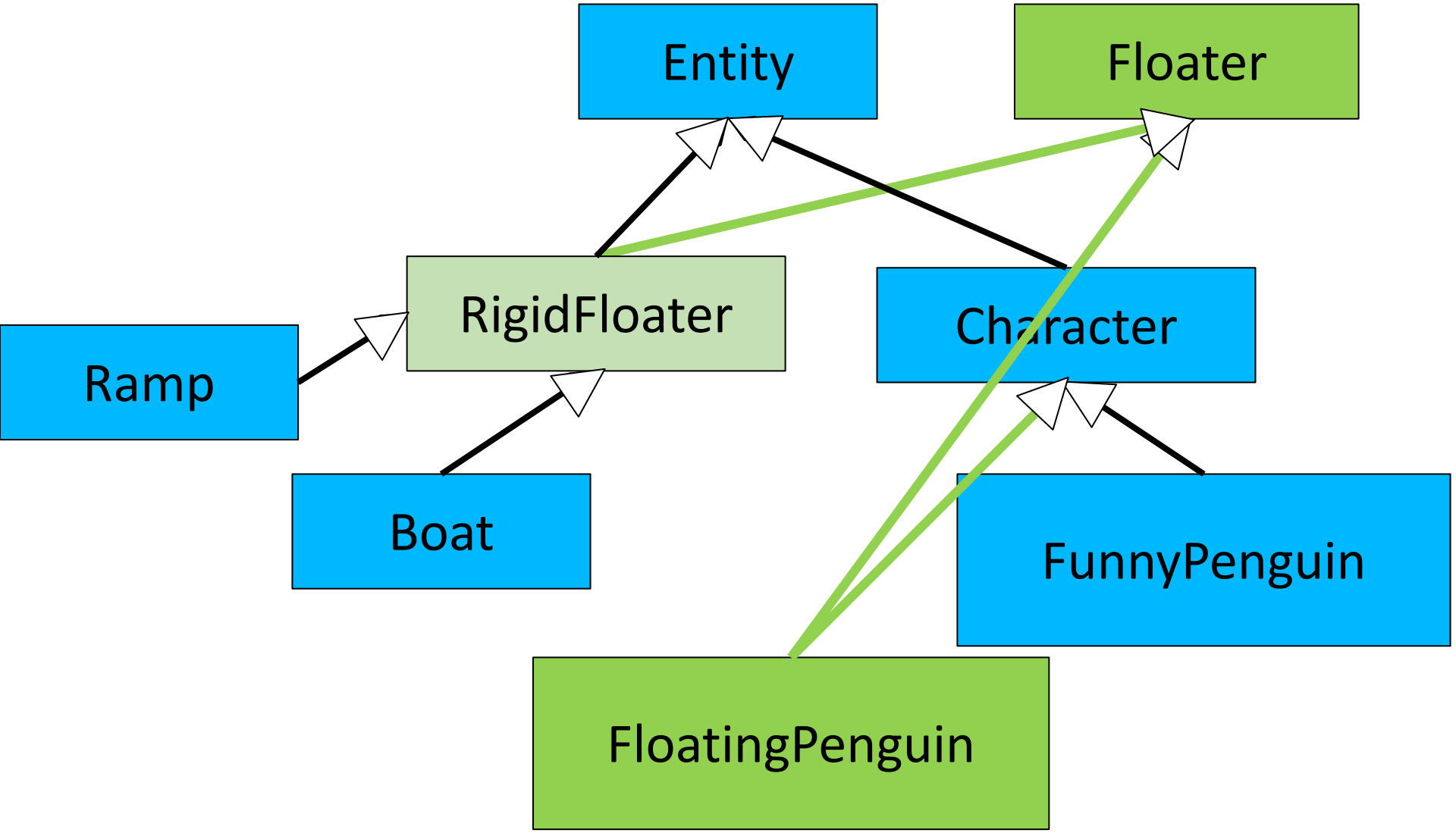
A többszörös öröklődés nem is rossz

- a többszörös öröklődés hasznos
- akárhány interface megvalósítása teljesen rendben van (csak tisztán virtuális függvények)
- egyszerű ősosztályból (mix-in) öröklődés rendben van

Mix-in osztályok

- különálló oszályok alaposztály nélkül
- plusz funkciót vihetnek a monolitikus hierarchiába
- korlátozottabb, biztonságosabb többszörös öröklődés

Hajós játék mix-in-ekkel

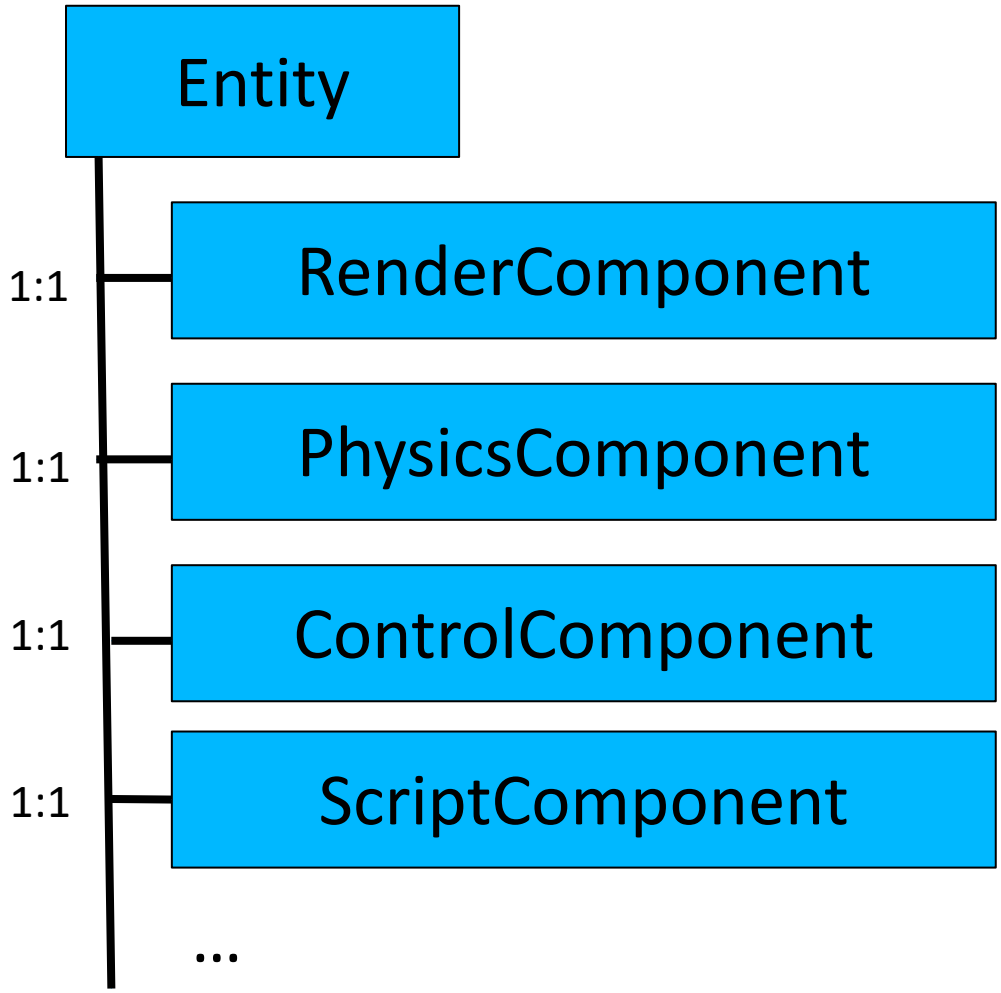


A megoldás: öröklés helyett kompozíció (aggregáció)!

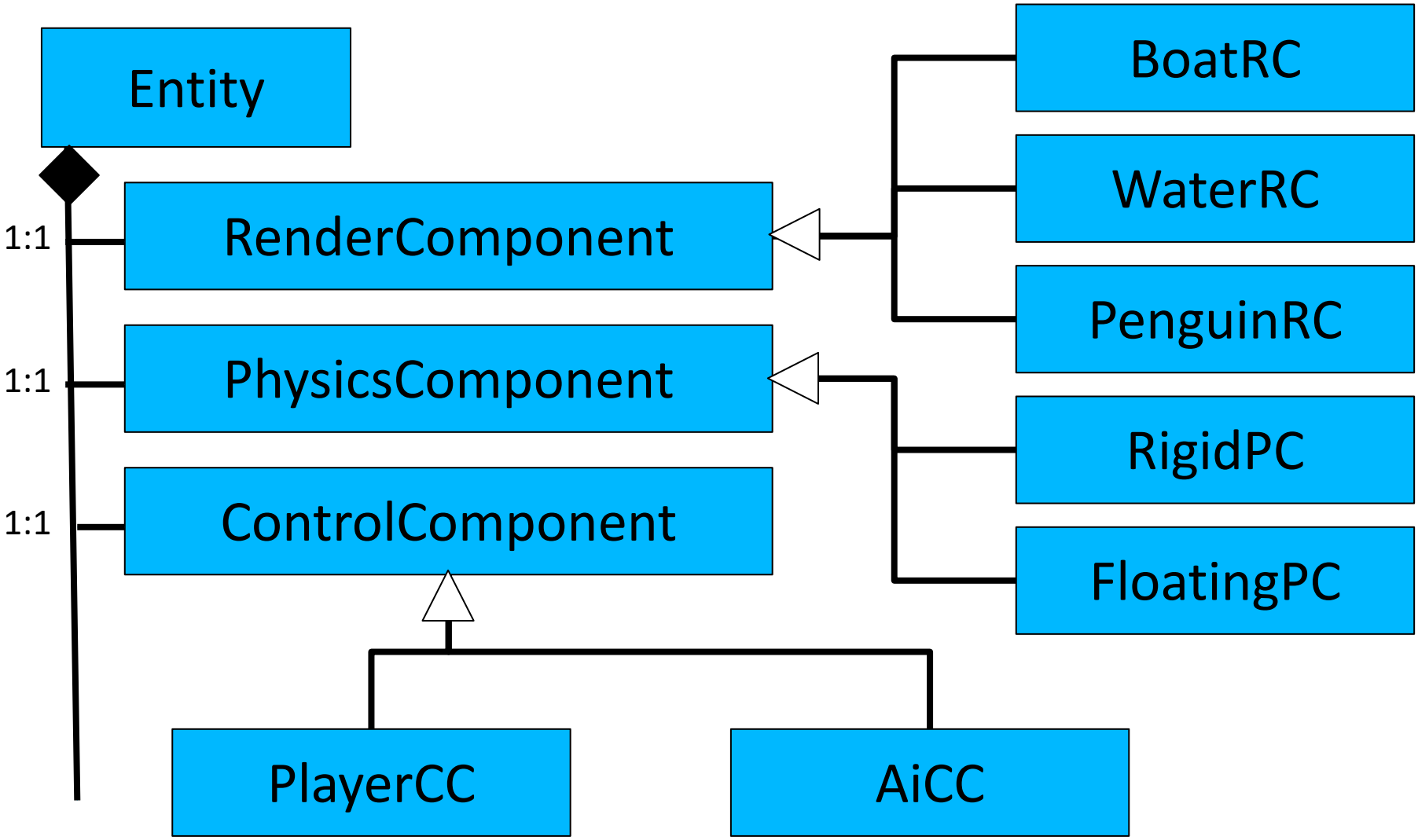
- is-a helyett has-a
  - a Boat nem egy Floater valami, hanem rendelkezik a Floater képességgel
- ha kell nem-lebegő hajó, kivehetjük
  - akár futásidőben is
- ha valami más kell lebegjen, akkor ő is megkapja a képességet

Komponensek

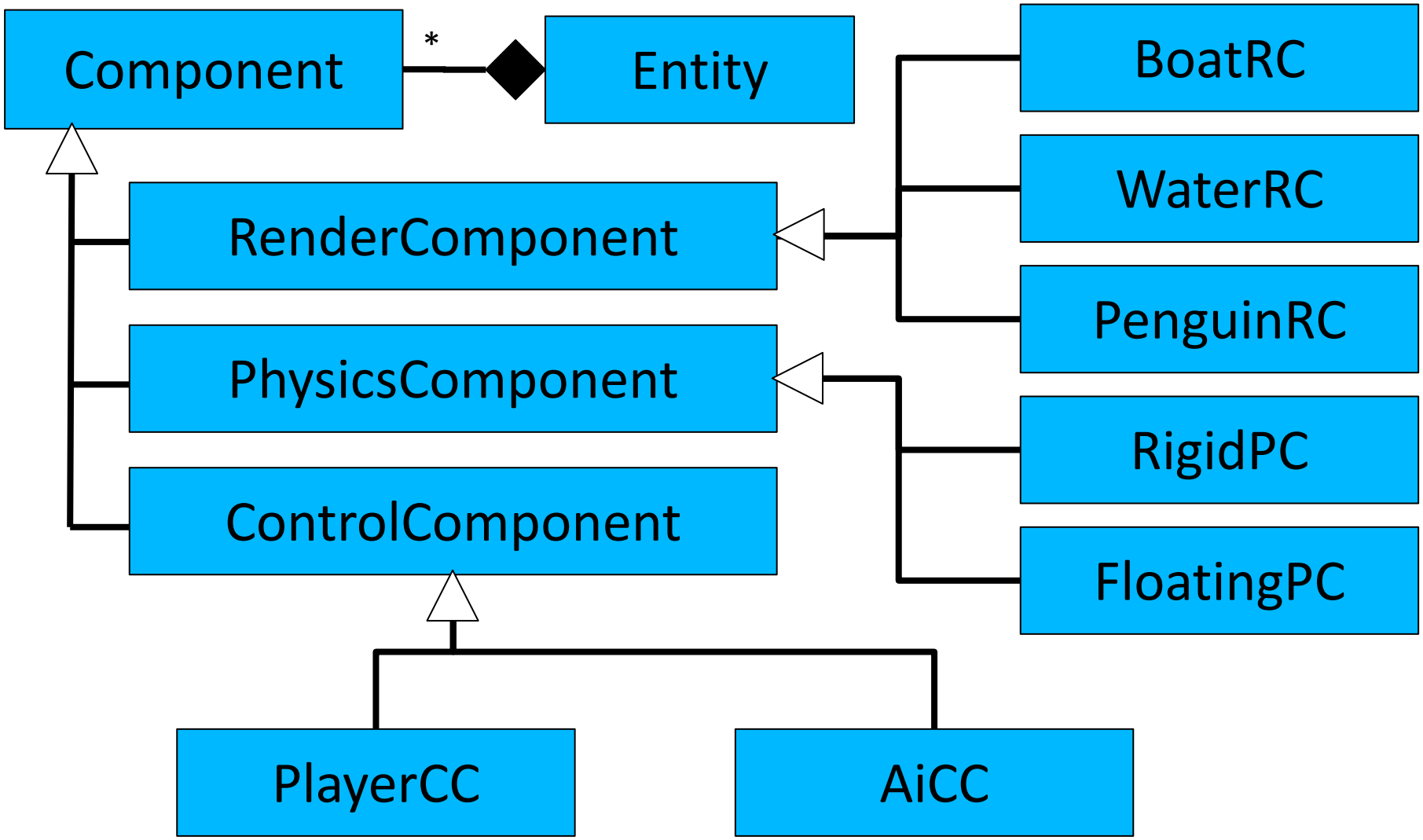
- az entitások számos objektumból állnak össze
- nincs öröklődés
  - az entitások között
- minden entitás csak Entity
- kisebb osztályok
- szétcsatolt osztályok



Csereszabatos komponensek



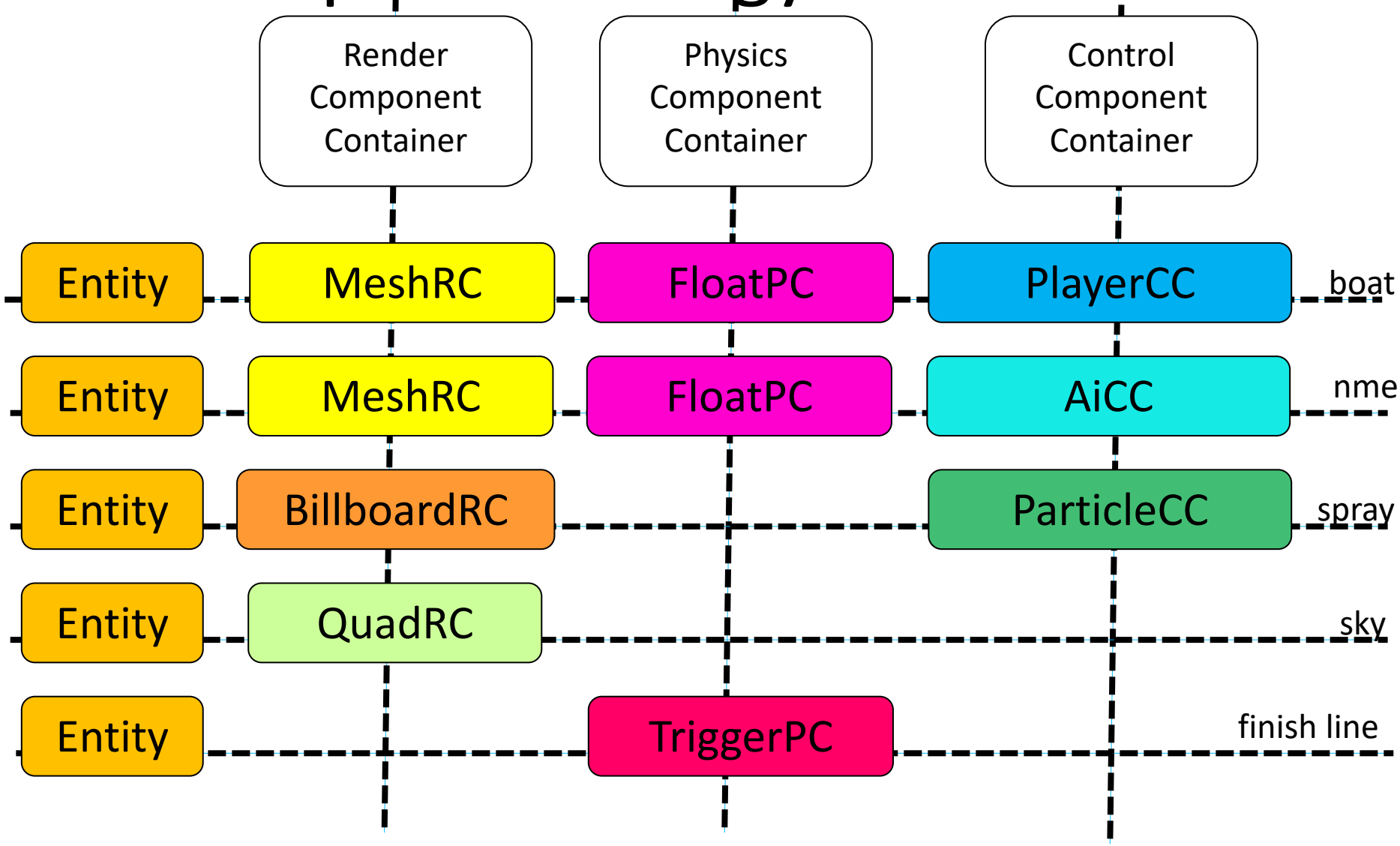
Rugalmas komponensek



Az Entity osztály szerepe

- Az Entity osztály
  - komponensek gyűjteménye
  - egyetlen szerepe: adott entitás adott komponensének megtalálása
- De a tipikus művelet:
  - minden adott típusú komponenekre: update()
  - pl. rajzold ki az összes render-komponenst
  - pl. mozgasd az összes animációs komponenst
  - pl. hajtsd végre az összes szkriptet

A komponensek egy rácson



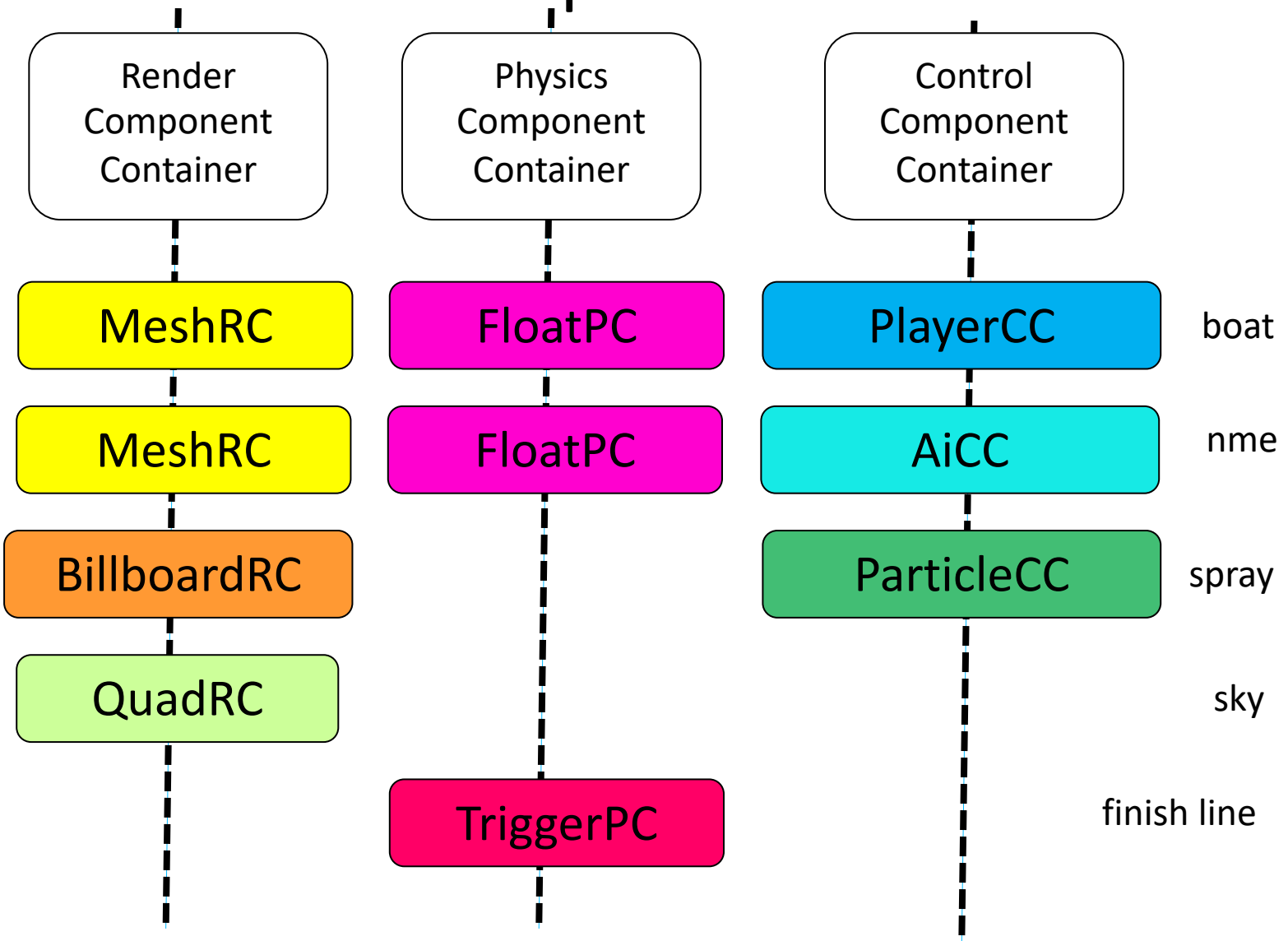
Dobjuk ki az entitásokat?

- csak akkor, ha komponensentípusonkénti adattáblákat akarunk
- csak akkor akarunk ilyet, ha az hatékonyabb
- csak akkor hatékonyabb, ha végigtekerni rajtuk villámgyors, azonosító alapján keresni pedig vállalható sebességű
- és nincsenek virtuális függvényhívások

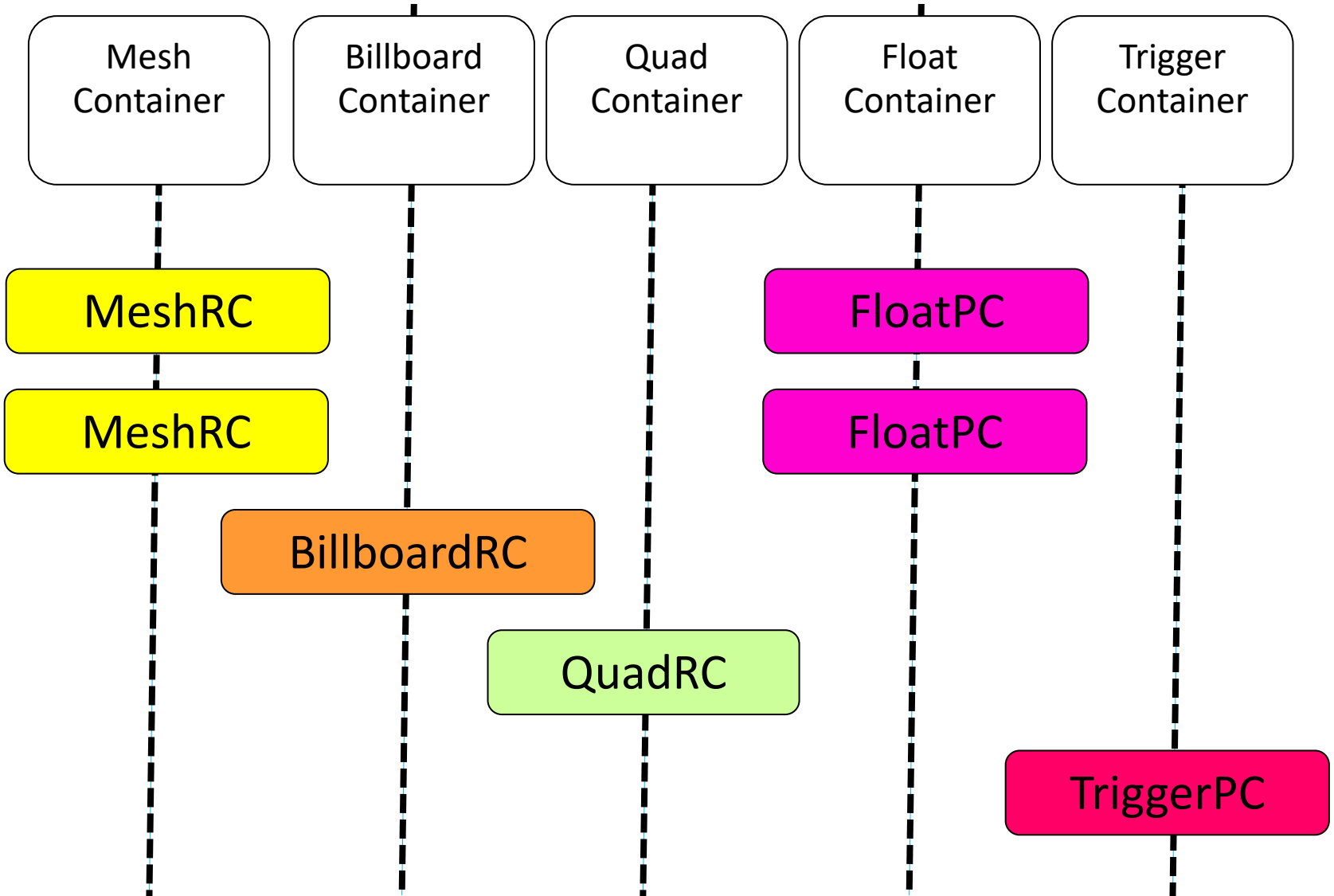
Structure of arrays

- Scene Entity példányokból: struktúrák tömbje
  - az entitások a struktúrák
  - egy adattag vagy komponens elérése minden entitásból körülményes és nem hatékony
- Scene komponenstárolókból: tömbök struktúrája
  - a konténereken végigiterálni hatékony

A tulajdonság-központúság felé: nincs öröklődés a komponensek között



A tulajdonság-központúság felé: nincs öröklődés a komponensek között



Adatszerkezet a tulajdonságok tárolására

- kevésbé fontostól a fontosabb felé
  - elemek beszúrását és törlését támogassa
    - dinamikus
  - azonosító alapján történő lekérdezéseket támogassa
    - asszociatív
  - támogassa a hatékony, gyorsítótárbarát iterációt
- talán egy vörös/fekete fa (std::map)?

Rendezett tömb!

- milyen komplexitásúak a műveletek?
- beszúrás és törlés
  - lineáris... de annak elég gyors
  - a fa logaritmikus ugyan, de annak elég lassú
- lekérdezések azonosító alapján
  - logaritmikus – bináris keresés
  - fa szintén
- iteráció
  - triviálisan hatékony, gyorsítótárbarát

Hogyan válasszunk Game Object Modellt?

- minigame – monolitikus hierarchia
- házi feladat – monolitikus, esetleg mixinekkel
- garázsprojekt – fix komponensek
- AAA játék – rugalmas komponensek
- előfizetéses játékvilág folytonosan fejlesztett tartalommal – tulajdonságalapú

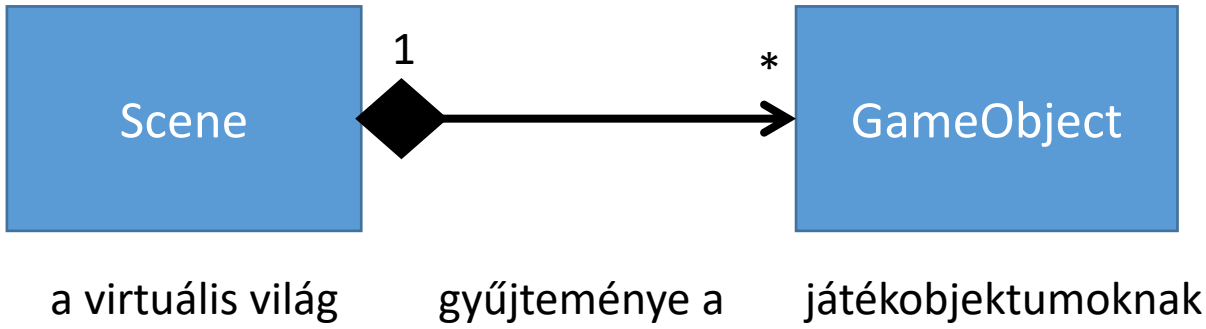
Kotlinban

- objektum-orientált komponens-megvalósítás

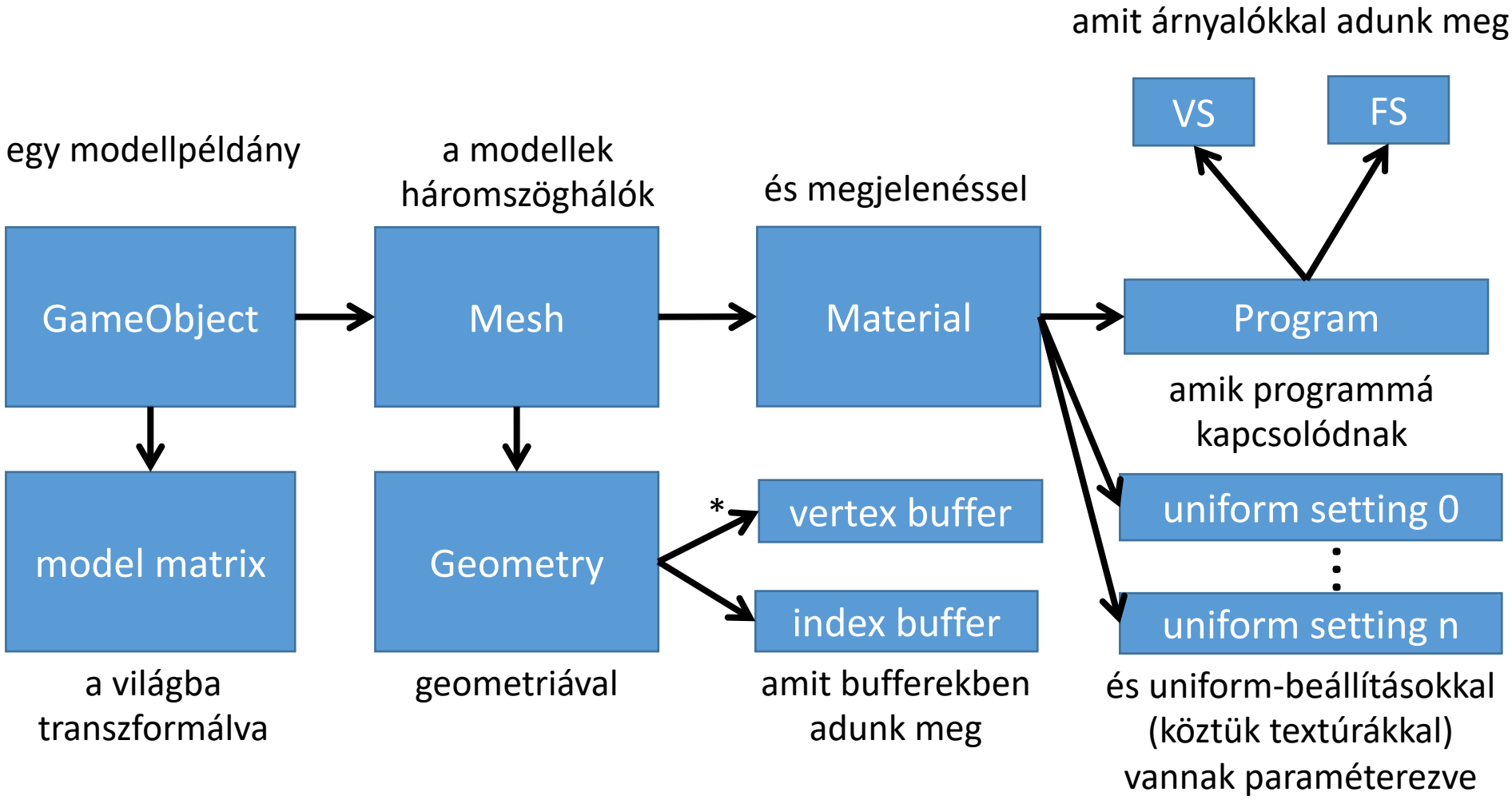
Transzformációk a játékmotorban

- modellpéldányok elhelyezése a virtuális világban
  - modell skálázása, forgatása, eltolása
  - animáció: a transzformáció képkockáról képkockára változik
  - a modell maga változatlan (vertex, index bufferek)
  - minden példányra más transzformáció [**game object**, entity]
  - ez a **modellezési transzformáció**, amit a **játékobjektum modellmátrixa** reprezentál
- a virtuális világ nézetablakra képzése
  - a virtuális világot egy téglalap alakú ablakon át látjuk
  - ez alapértelmezésben [-1,-1], [1, 1], de ha **mindent ugyanazzal a nézeti** transzformációval leképzünk, akkor ez változtatható

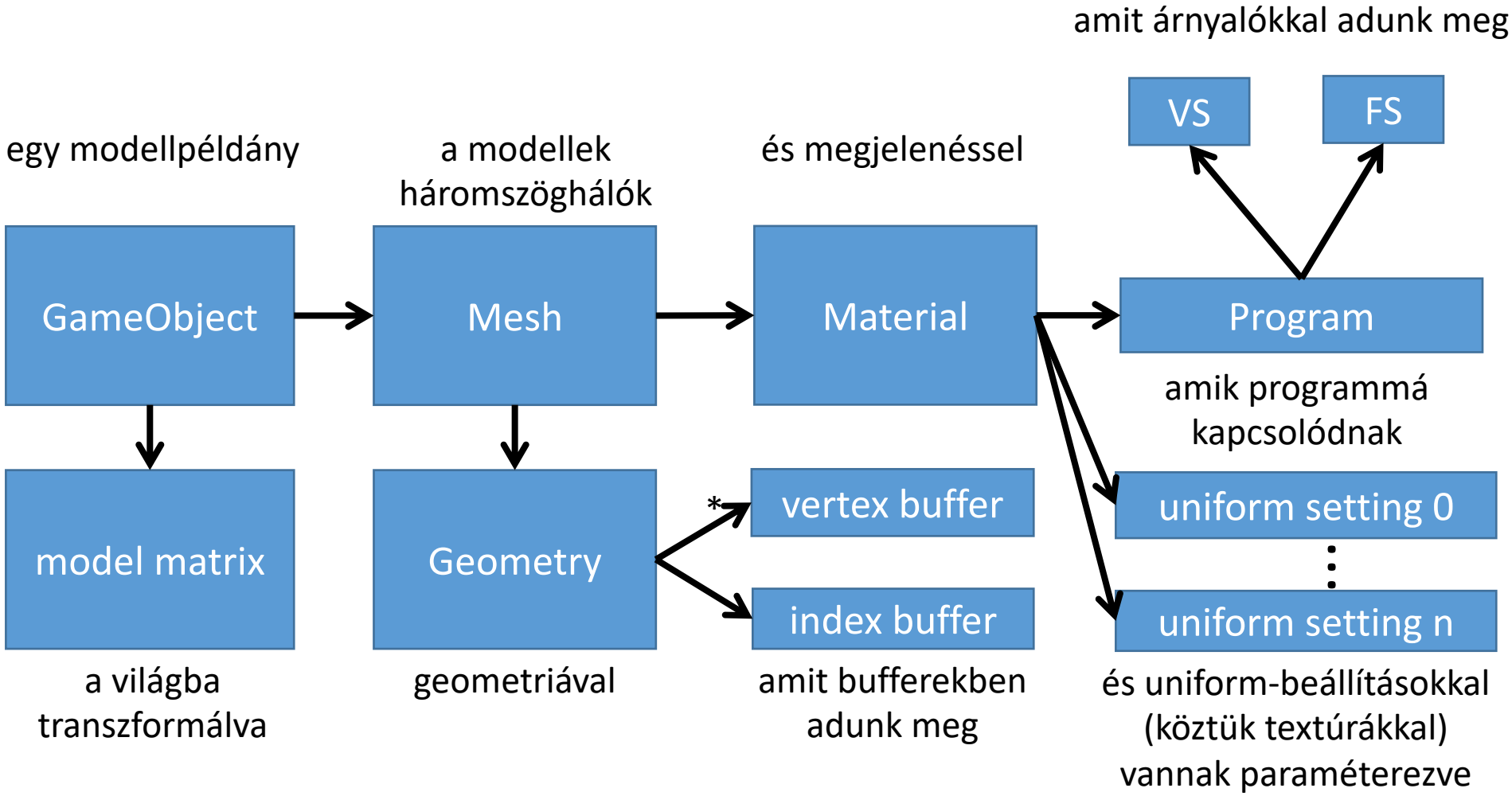
A virtuális világ és a játékobjektumok



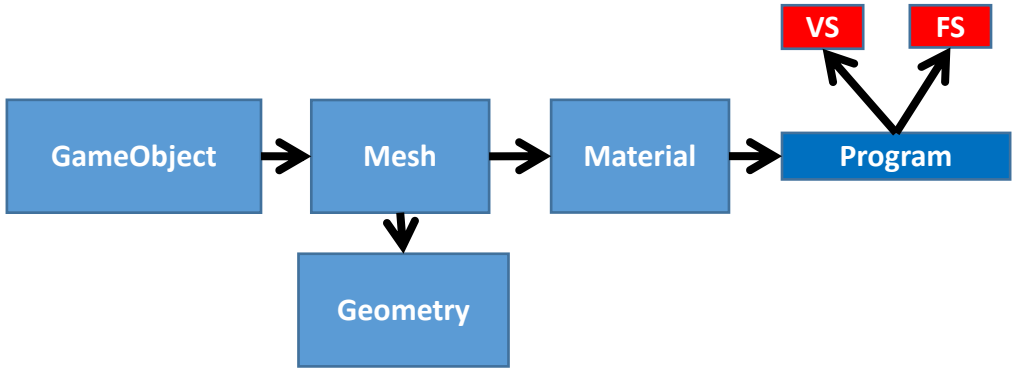
Egy játékobjektum anatómiája



Osztályok (komponensek?)



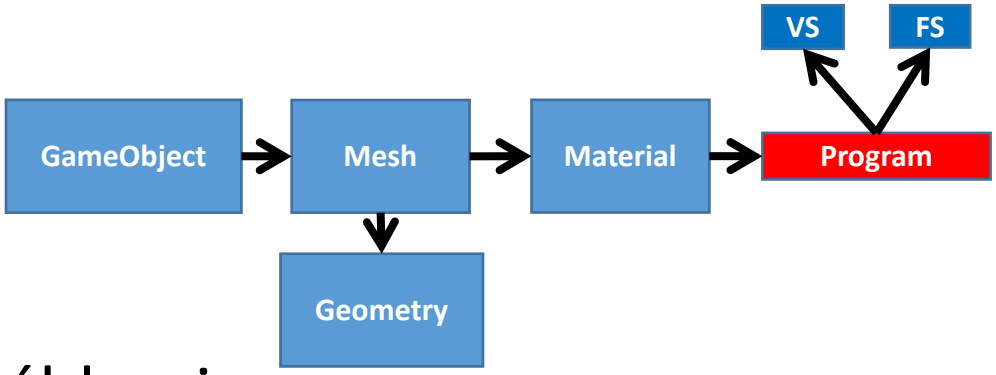
Shader



- a `Shader` osztály marad, ami volt
  - nincs változás
- az árnyalók nem lesznek komponensek
- csak addig érdekesek, amíg `Program`okat nem csinálunk belőlük

- de a `Program`ból komponens csinálunk

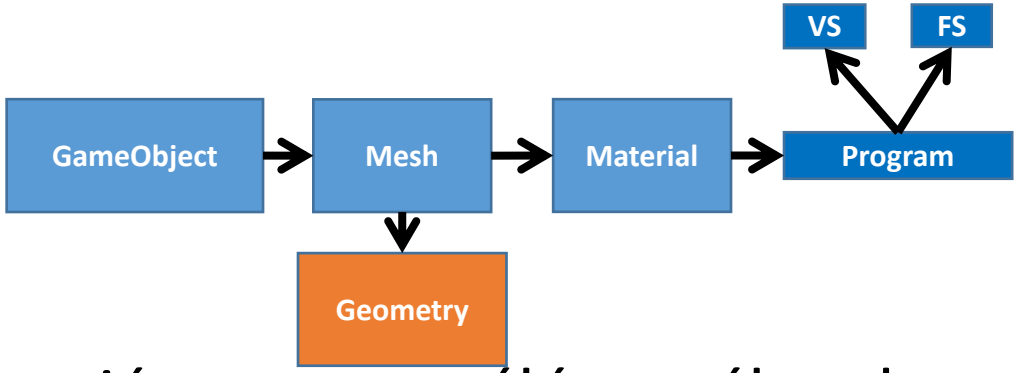
Program



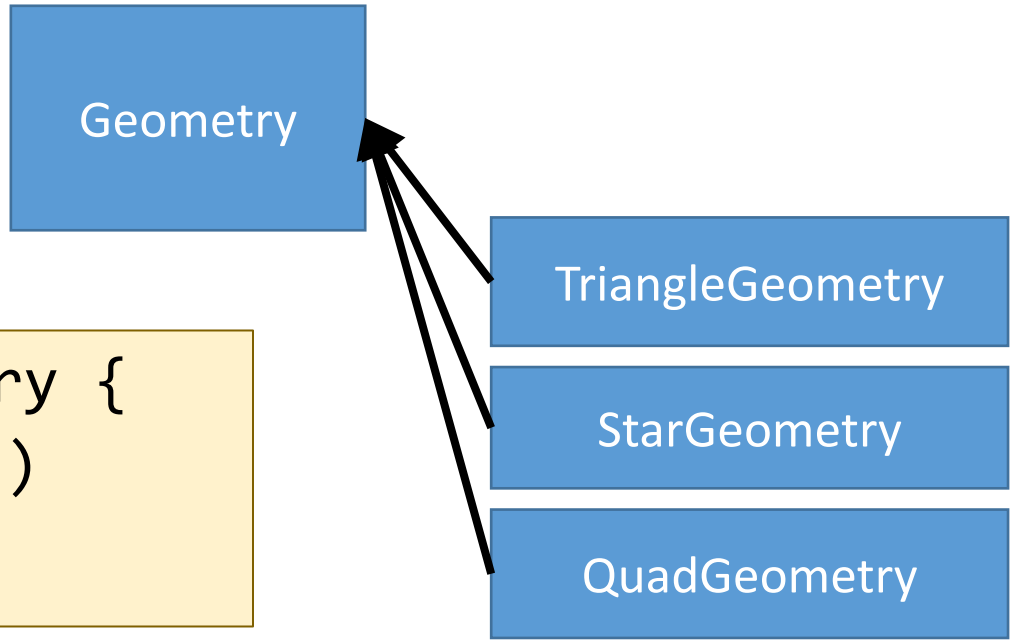
- a `Program` osztály továbbra is
  - WebGL programmá linkeli az árnyalókat
  - beköti az attribútumokat a bemenetekre (vertexPosition, vertexNormal, vertexTexCoord)

- komponens lesz a `Program`
  - de a lényegi plusz funkciókat nem közvetlenül ide rakjuk majd
  - hanem lesz egy alkomponense, a `ProgramReflection`
    - a `WebGLMath` könyvtárban van
    - ez kezeli a program uniformjait

Geometry

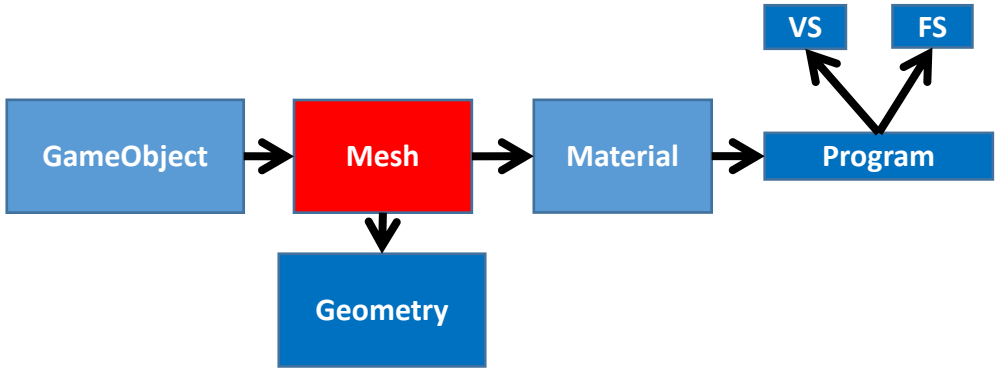


- már van több, geometriát reprezentáló osztályunk: `TexturedQuadGeometry`, `TexturedTriangleGeometry`
- mindnek van `draw` metódusa
- kapnak egy közös őst



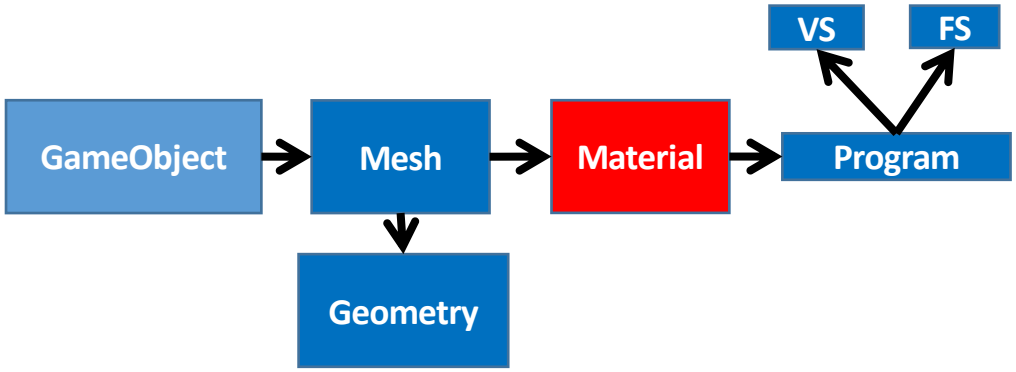

```
abstract class Geometry {
    abstract fun draw()
}
```

Mesh



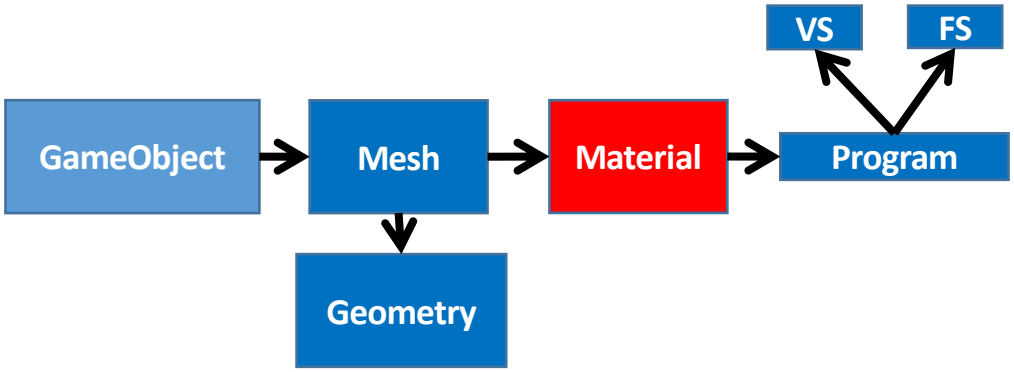
- Mesh = Geometry + Material

Material



- mi az anyag?
  - felület optikai tulajdonságai
  - minden pipelinebeállítás, ami a geometrián kívül a rajzoláshoz kell

Material



- Material = Program + <uniform settings>
- pl. szín, fényesség, textúrák
- de nem csak az anyag alapján állítunk uniformokat
  - pl. idő, modellmátrix, kamerapozíció, fényforrás...
  - tehát azt, hogy uniformokat állít be, legjobb, ha minden komponensünk tudja majd
- vagyis az anyag jóformán egy üres komponens

Uniformok, textúrák az anyagban

- két anyag használhatja ugyanazt a WebGL programot
- eltérő szín- és textúrabeállításokkal
- minden anyagnak a saját uniform-értékeit kell tárolnia, a textúrákat beleértve
  - de milyen uniformok vannak?
  - a használt shadertől függ
  - shader reflection kell hozzá

Más uniformok

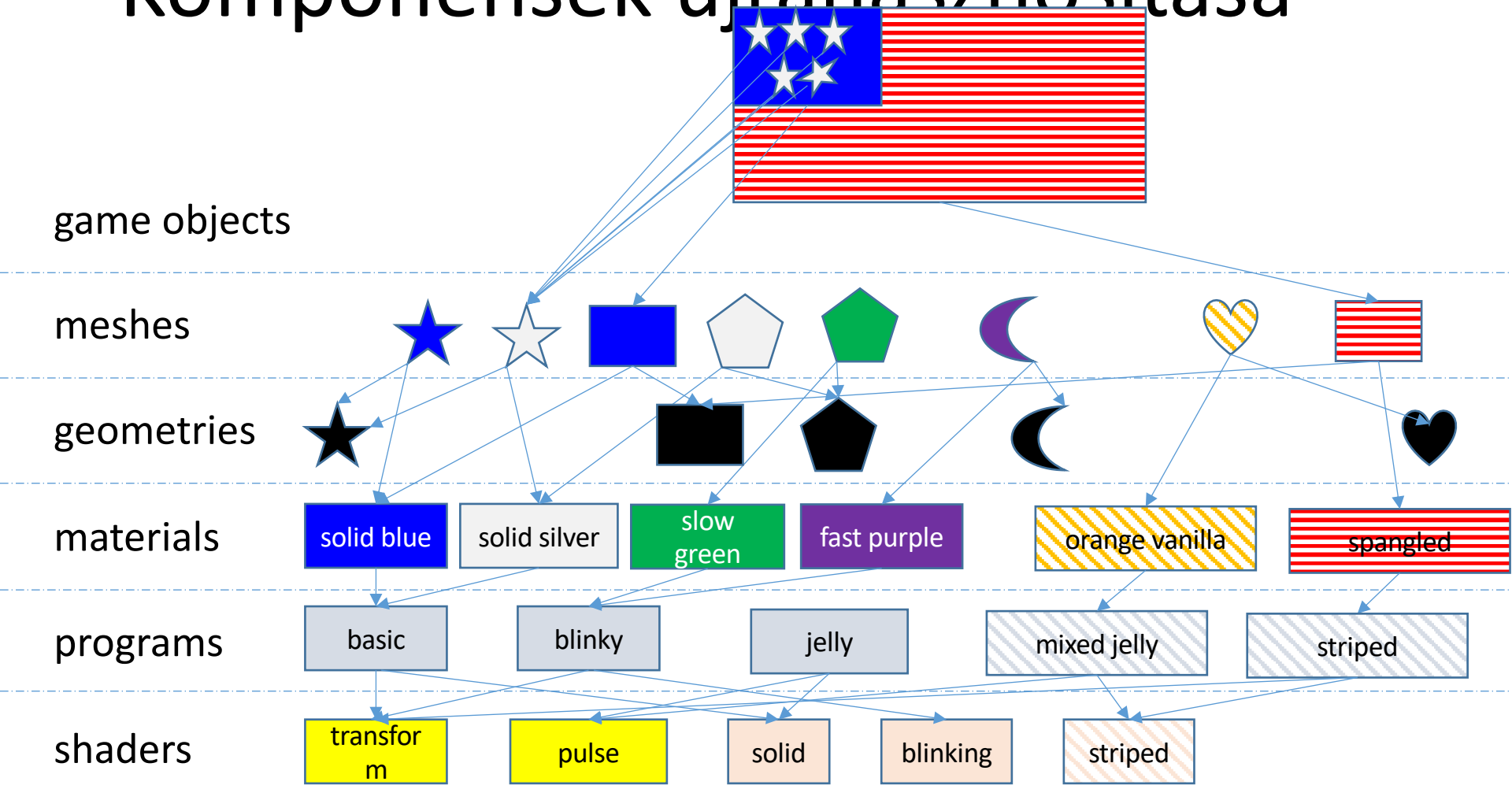
- per frame
  - kameramátrixok, fényforrástulajdonságok
- per object
  - modellmátrix, animációs fázis

- ezeket is be kell állítani rajzolás előtt

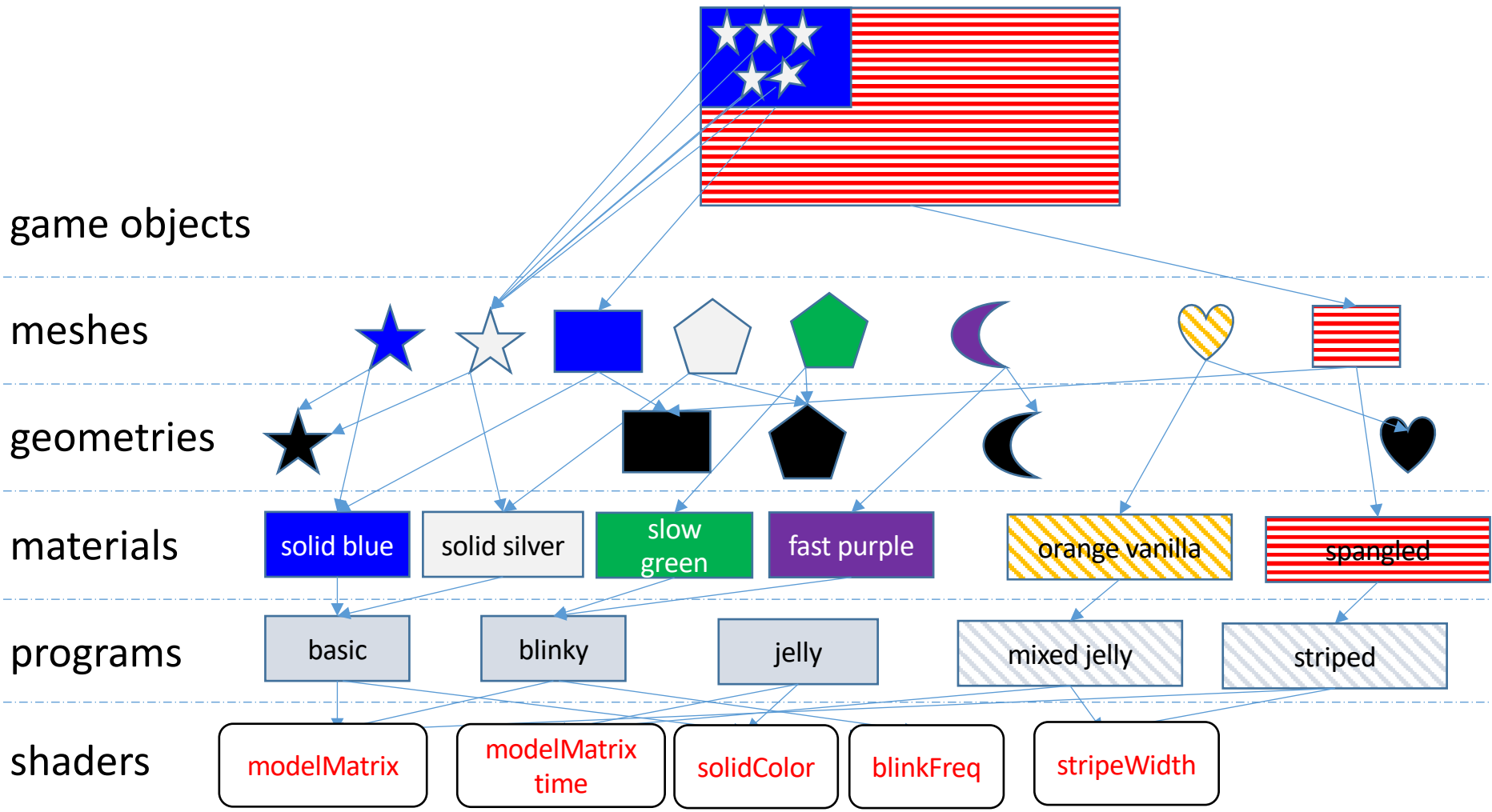
Mitől függ, hogy melyik uniform tartozik az anyaghoz?

- az anyagokat nem változtatjuk
  - az elején legyártjuk és utána használjuk
- de a per-frame és per-object uniformok folyamatosan változnak

Komponensek újrahasznosítása



Uniformok és komponensek

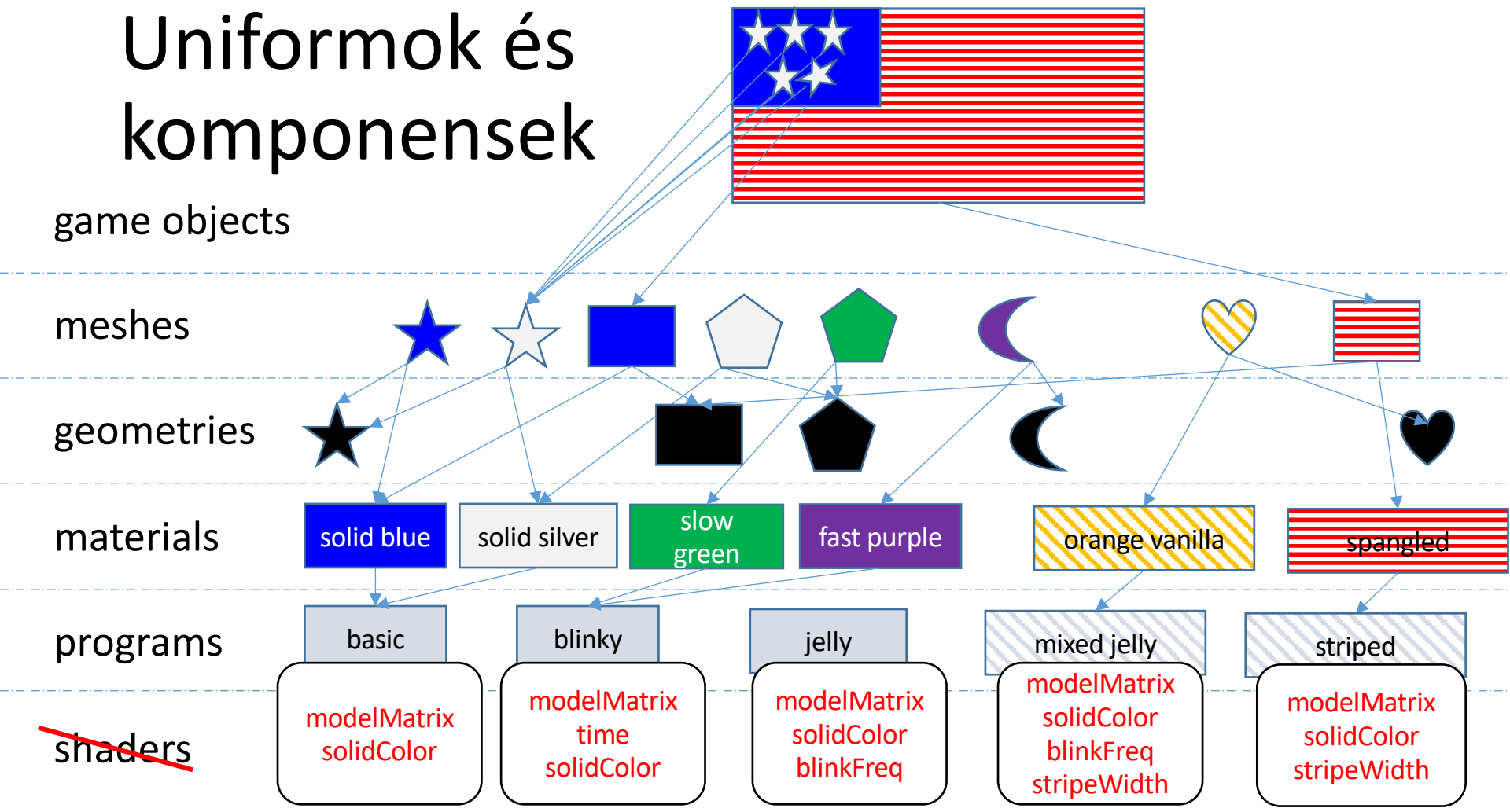


Shader reflection

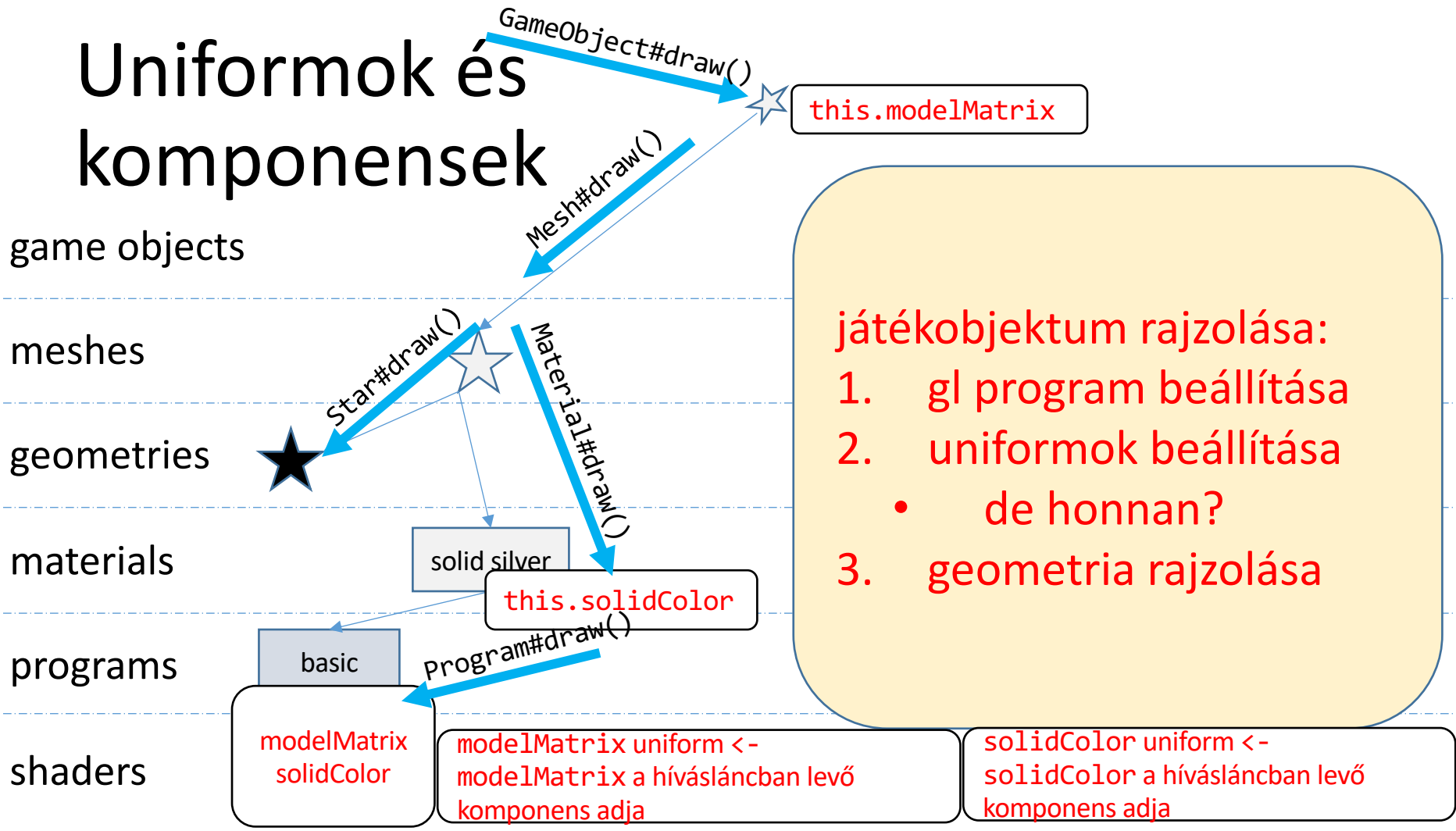
- vizsgálhatjuk a lefordított és linkelt **programokat**
  - szerezzünk információt az árnyalókban használt **uniform változókról**
  - ez alapján fogunk Kotlin változókat létrehozni

```
val nUniforms = gl.getProgramParameter(glProgram,
                        WebGLRenderingContext.ACTIVE_UNIFORMS) as Int
for(i in 0 until nUniforms){
  val glUniform = gl.getActiveUniform(glProgram, i)!!
  // glUniform.name: variable name, e.g. "lightPos[0]"
  // glUniform.type: numerical type id, e.g. GL.FLOAT_VEC2
  // glUniform.size: array size, or 1 for non-arrays
  location = gl.getUniformLocation(glProgram, glUniform.name)!!
}
```

Uniformok és komponensek



Uniformok és komponensek

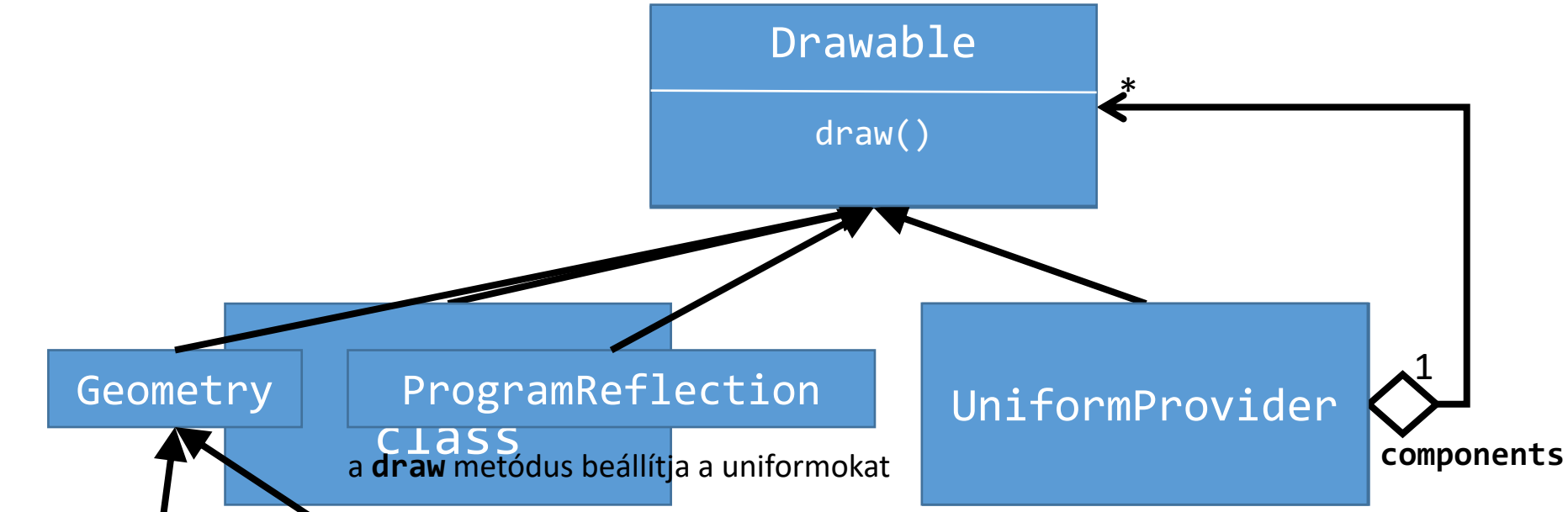


játékobjektum rajzolása:

1. gl program beállítása
2. uniformok beállítása
   - de honnan?
3. geometria rajzolása

A Composite tervezési minta

- rész-egész hierarchia




```
override fun draw(
  vararg uniformProviders : UniformProvider){
  components.forEach {
    it.draw(this, *uniformProviders)
  }
}
```

metódusok implementációja:
gyerekobjektumok metódusainak hívása

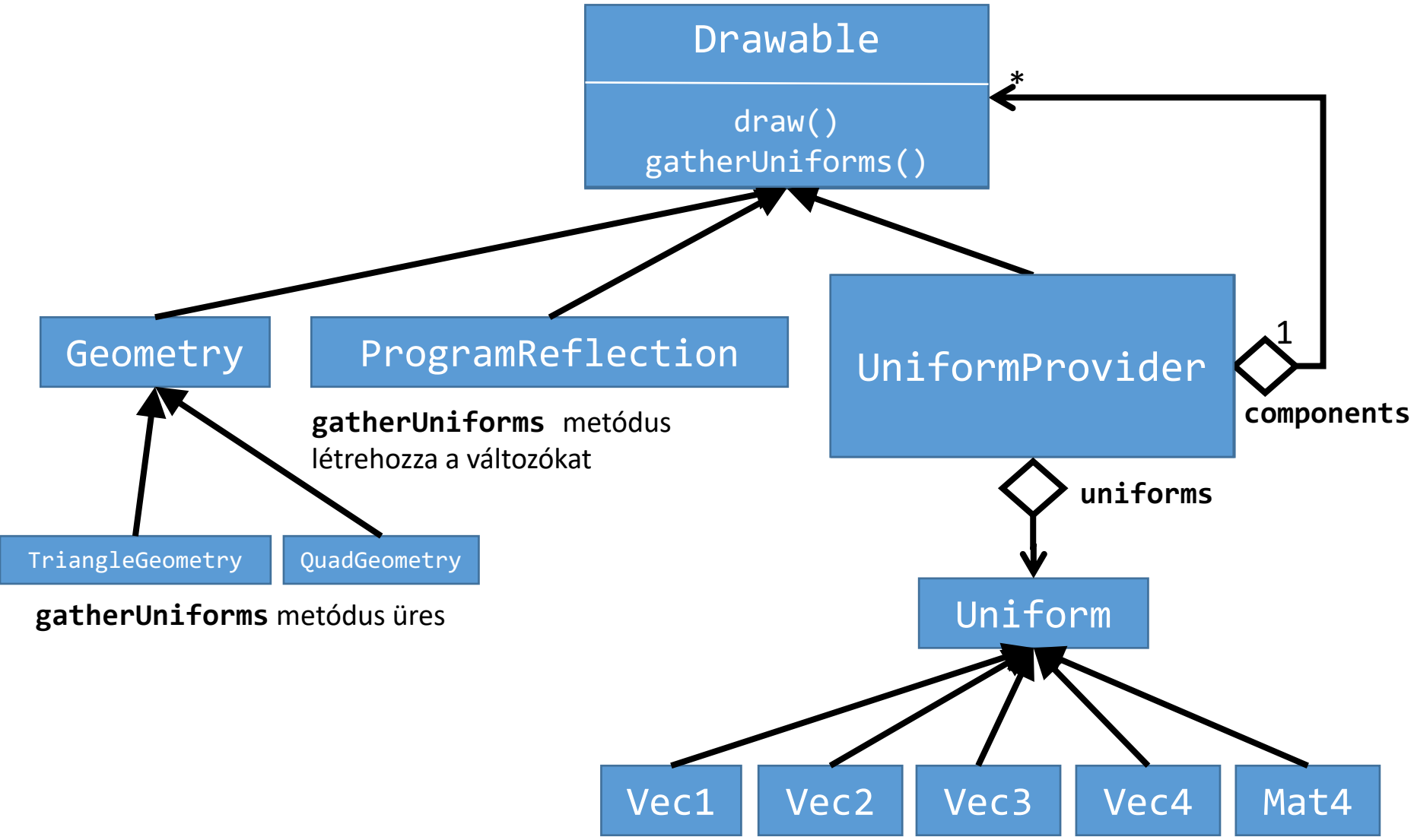
ProgramReflection

- lekérdezi a uniformokat a WebGL programból, structnév alapján csoportosítva
- draw metódus
  - megkapja a hívásláncban szereplő komponenseket
  - a bennük tárolt uniform értékeket beállítja

```
override fun draw(vararg uniformProviders : UniformProvider) {
  gl.useProgram(glProgram)

  for(provider in uniformProviders){      // pl. a game object
    for(structName in provider.glslStructNames) { // tipikusan "gameObject"
      for(uniformDesc in uniformDescriptors[structName]) { // pl. modelMatrix
        provider[uniformDesc.name]!!.commit(gl, uniformDesc.location)
} } } }
```

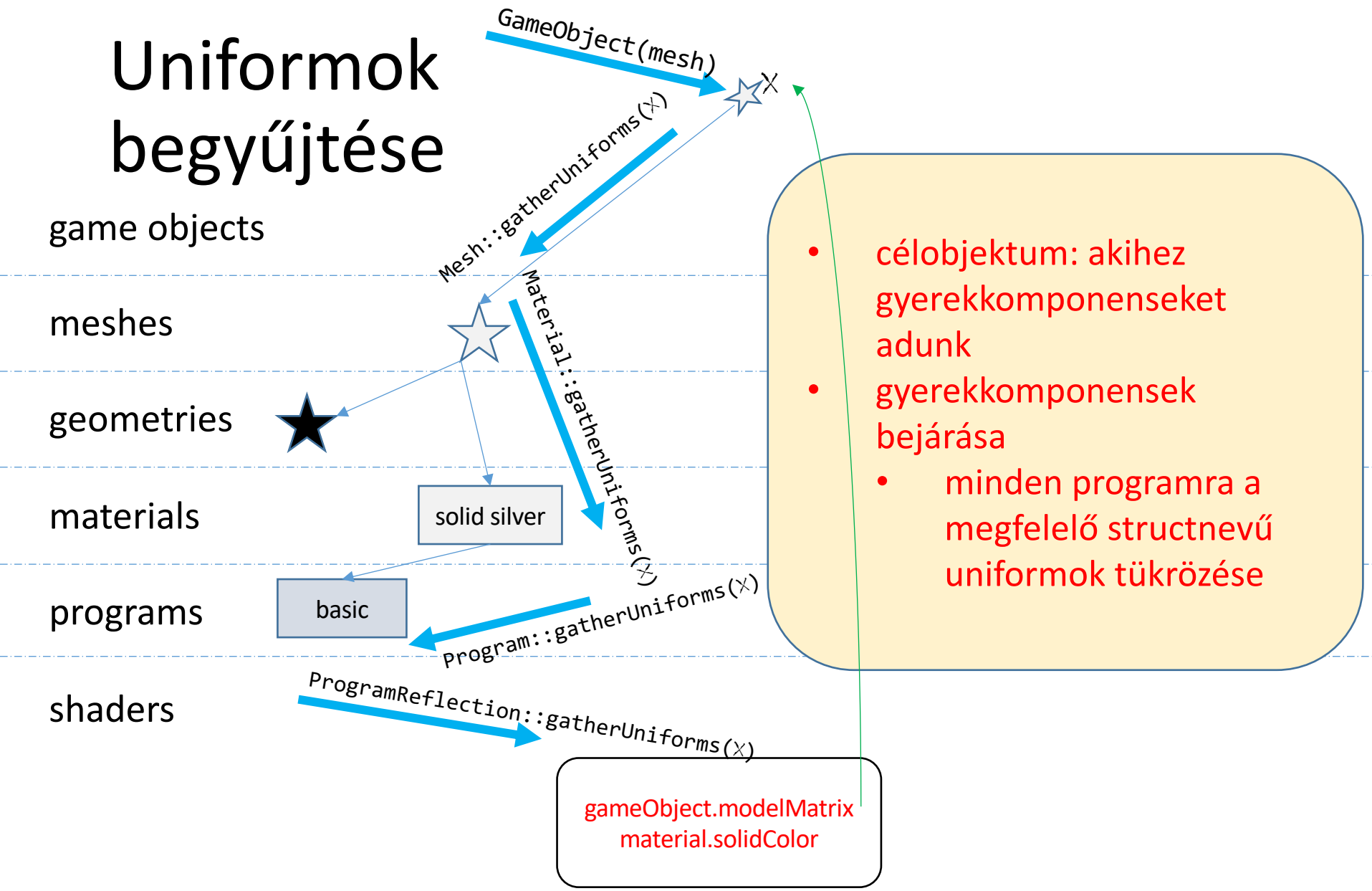
Uniform változók létrehozása



A `ProgramReflection` tudja még: létrehozni a változókat a `UniformProviderekbe`

- gyerekkomponensek hozzáadásakor
- biztosítsuk, hogy a providereknek megfelelő típusú változói vannak a uniformok értékeinek tárolására
- ha ezt a programozóra hagyjuk
  - az pluszmunka és hibalehetőség
- Kotlin objektumok a uniformok tükrözésére
  - megfelelő típusú objektumokat kell létrehozni típusazonosító (`glUniform.type`) alapján (e.g. vec3, mat4, …)
  - a létrehozott változókat név szerint el kell érnünk a `UniformProvider` objektumokban

Uniformok begyűjtése



- célobjektum: akihez gyerekkomponenseket adunk
- gyerekkomponensek bejárása
  - minden programra a megfelelő structnevű uniformok tükrözése

ProgramReflection::gatherUniforms

```
// target pl. egy game object
override fun gatherUniforms(target : UniformProvider){
  for(structName in target.glslStructNames) { // "gameObject"
    for(uniformDesc in uniformDescriptors[structName]){ //modelMatrix
      val reflectionVariable = ProgramReflection.makeVar(
                        uniformDesc.type, uniformDesc.size) //Mat4

      target.uniforms[uniformDesc.name] = reflectionVariable
    }
  }
}
```

Program osztály

struktúranév

```
class Program(val gl : WebGL2RenderingContext, val vertexShader : Shader,
  val fragmentShader : Shader ) : UniformProvider("program") {

    val glProgram = gl.createProgram() ?: throw Error("Could not create WebGL program.")

init {
  gl.attachShader(glProgram, vertexShader.glShader)
  gl.attachShader(glProgram, fragmentShader.glShader)

  gl.bindAttribLocation(glProgram, 0, "vertexPosition")
  gl.bindAttribLocation(glProgram, 1, "vertexColor")

  gl.linkProgram(glProgram)
  if (gl.getProgramParameter(glProgram, GL.LINK_STATUS) == false) {
    throw Error("Could not link shaders [vertex shader: ${vertexShader.sourceUrl}]:[fragment shader: ${fragmentShader.sourceUrl}\n${gl.getProgramInfoLog(this.glProgram)}")
  }

    addComponentsAndGatherUniforms(
      ProgramReflection(gl, glProgram))
    }
}
```

gyerekkomponens a reflection

Material osztály

struktúranév

```
class Material(program : Program) :
UniformProvider("material") {
  init {
    addComponentsAndGatherUniforms(program)
  }
}
```

gyerekkomponens program

Hogyan használható?

minden `Uniformnak` van `set` metódusa, nem is fontos a pontos típus

```
val solidProgram = Program(gl, vsTrafo, fsSolid)
val material = Material(solidProgram).apply {
    this["solidColor"]?.set(1.0f, 1.3f, 0.8f)
    this["noSuchProp"]?.set(0.0f, 0.3f, 0.8f)
}
```

ez automatikusan idekerült, mert a programban használjuk a `material.solidColor` uniformot

ha nincs ilyen uniform (pl. kikommenteztük a shaderből) a tömbindex-operátor dob egy warningot, de különben semmi sem fog történni

Mesh osztály: egyszerű, de lecserélhető


```
class Mesh(material : Material, geometry : Geometry)
: UniformProvider("mesh") {

    init{
        addComponentsAndGatherUniforms(
                            material, geometry)
    }
}
```

GameObject osztály

```
class GameObject( vararg meshes : Mesh
   ) : UniformProvider("gameObject") {
  val position = Vec3()
  var roll = 0.0f
  val scale = Vec3(1.0f, 1.0f, 1.0f)
  init {
    addComponentsAndGatherUniforms(*meshes)
  }
  fun update() {
    this["modelMatrix"]?.set()?.
      scale(scale)?.rotate(roll)?.translate(position)
  }
}
```

kaptunk ilyet a programtól (reméljük)

ez nem megy Uniform-ra, csak ha Mat4-re castolnánk

GameObject propertyvel

```
class GameObject() : UniformProvider("gameObject") {
  val modelMatrix by Mat4()

  init {
    addComponentsAndGatherUniforms(*meshes)
  }
  fun update() {
    modelMatrix.set().
      scale(scale).rotate(roll).translate(position)
  }
}
```

ez egy delegated property
előre berakja a uniformok közé

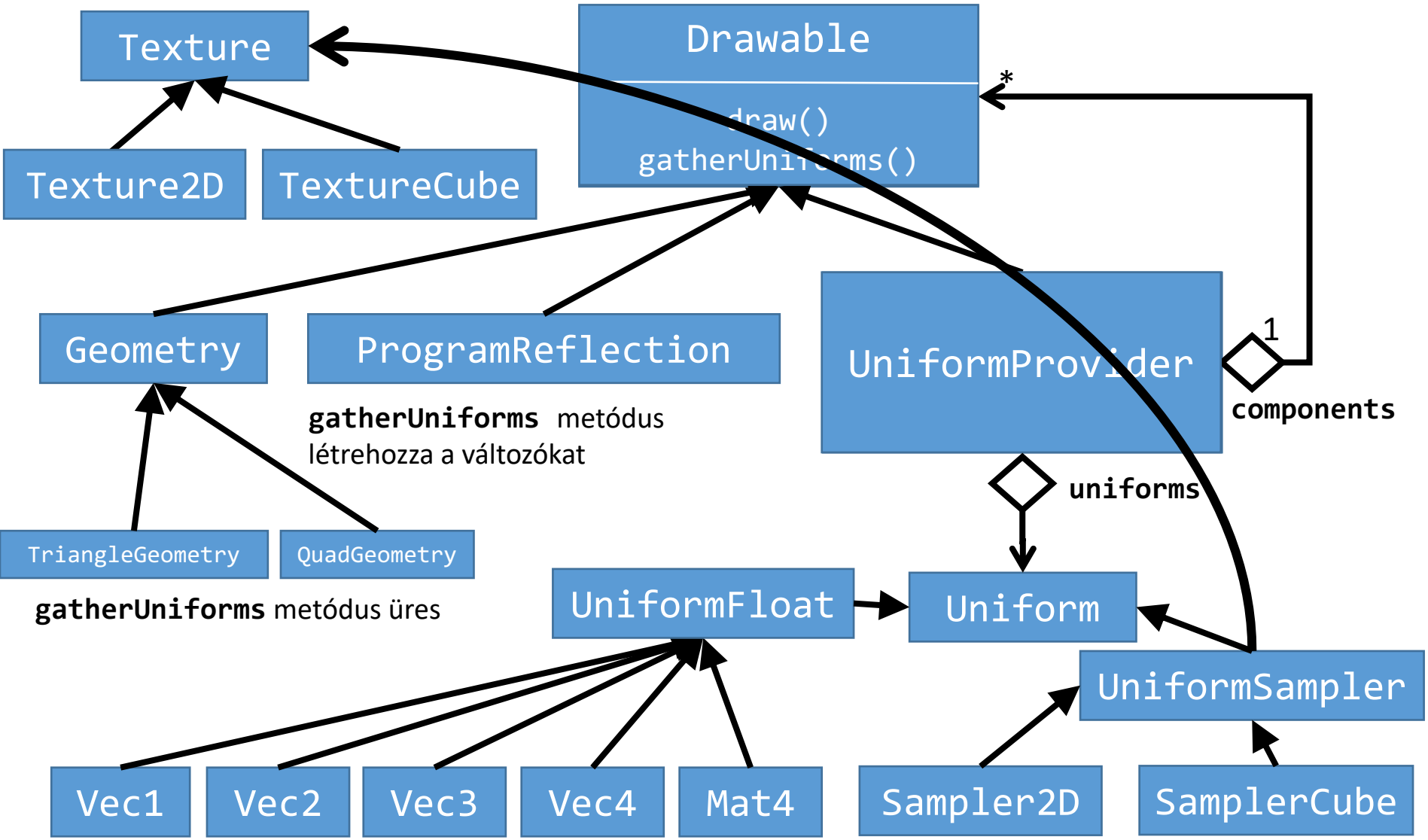
ez már csak ellenőrzi a típusát

elég a property-t állítanunk,
és a uniform beállítása automatikus

Textúrák

- uniform sampler2D változó
  - ennek egy integert kell beállítani
  - adott indexre a megfelelő textúrát kötni

Textúra uniformok



WebGLMath
Sampler2D::commit

```
override fun commit(
  gl : WebGLRenderingContext,
  uniformLocation : WebGLUniformLocation,
  samplerIndex : Int){

  storage[0] = samplerIndex;
  gl.uniform1iv(uniformLocation, storage)
  gl.activeTexture(
       GL.TEXTURE0 + samplerIndex)
  gl.bindTexture(GL.TEXTURE_2D, glTextures[0])
}
```

Textúrák használata

```
val texturedProgram = Program(gl, vsTrafo,
fsTextured)
val texture = Texture2D(gl, "media/asteroid.png")
val material = Material(texturedProgram).apply{
  this["colorTexture"]?.set(texture)
}
```