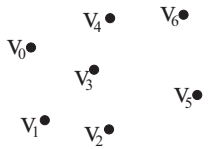
Bevezetés

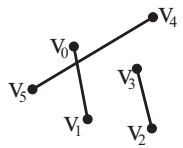
Grafikus csővezeték

Grafikus csővezeték

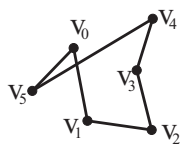
- Vertex feldolgozás
  - A vertexek egyenként a képernyő térbe vannak transzformálva
- Primitív feldolgozás
  - A vertexek primitívekbe vannak szervezve
- Raszterizálás
  - Primitívenként
    - Fragmensek
- Fragmens textúrázás és színezés
  - Fragmensenként



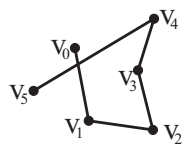
Pontok



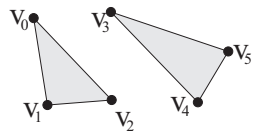
Vonalak



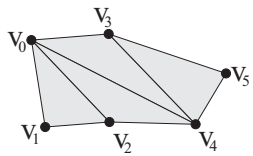
Vonal hurok



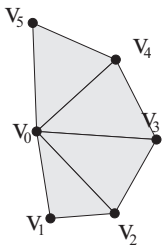
Töredezett vonal



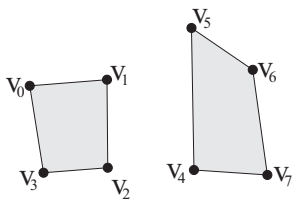
Háromszögek



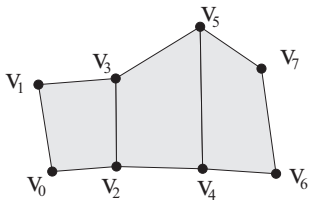
Háromszögsáv



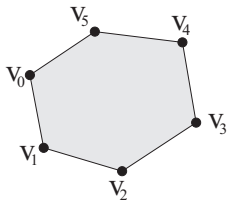
Háromszög-legyező



Négyszögek

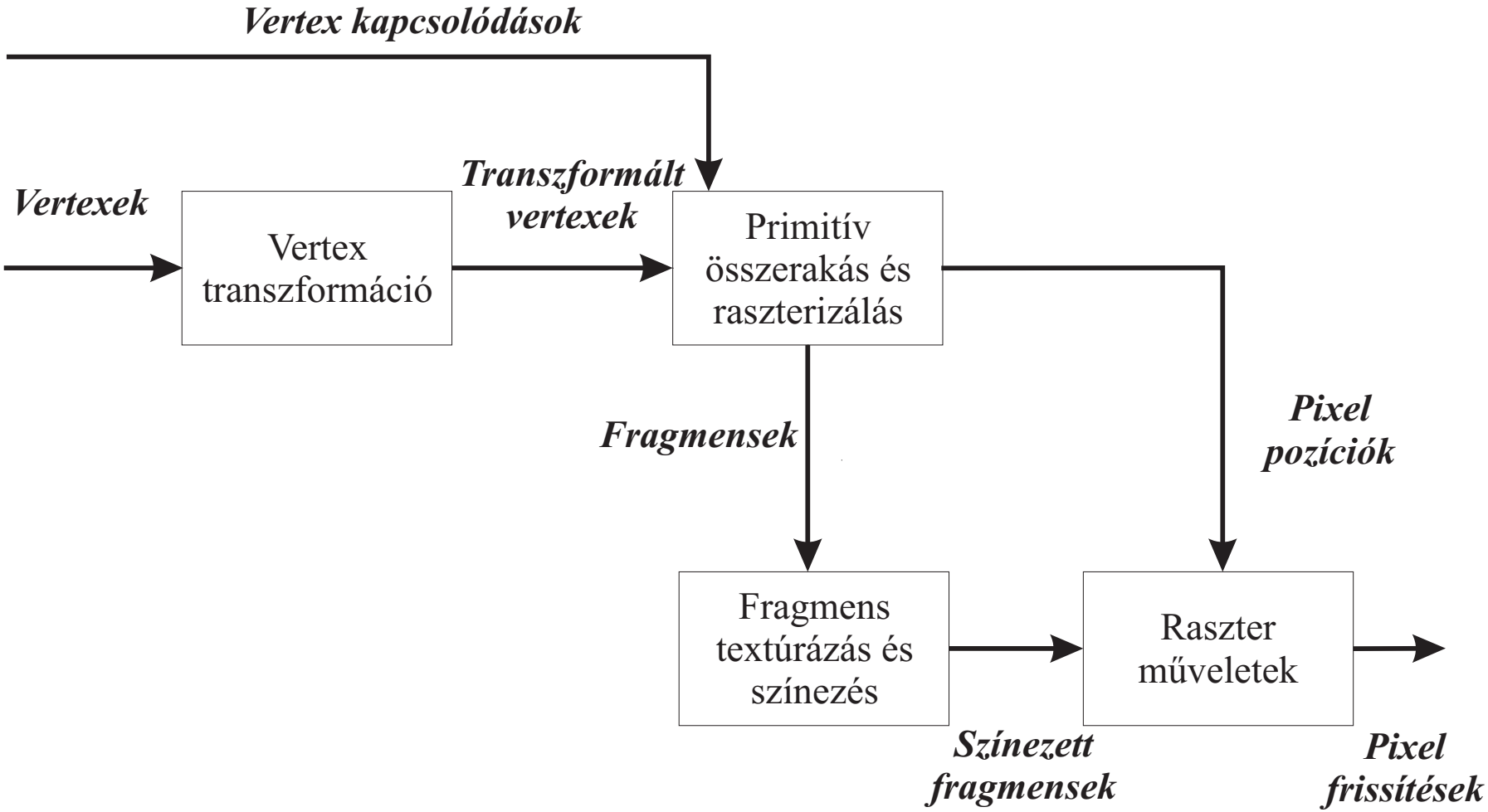


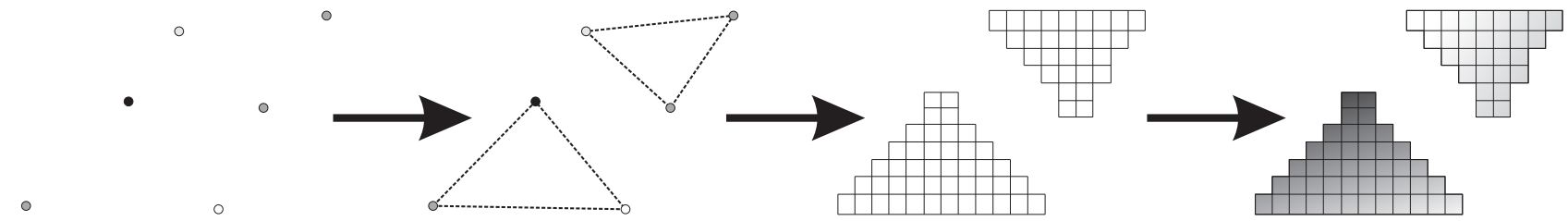
Négyszögsáv



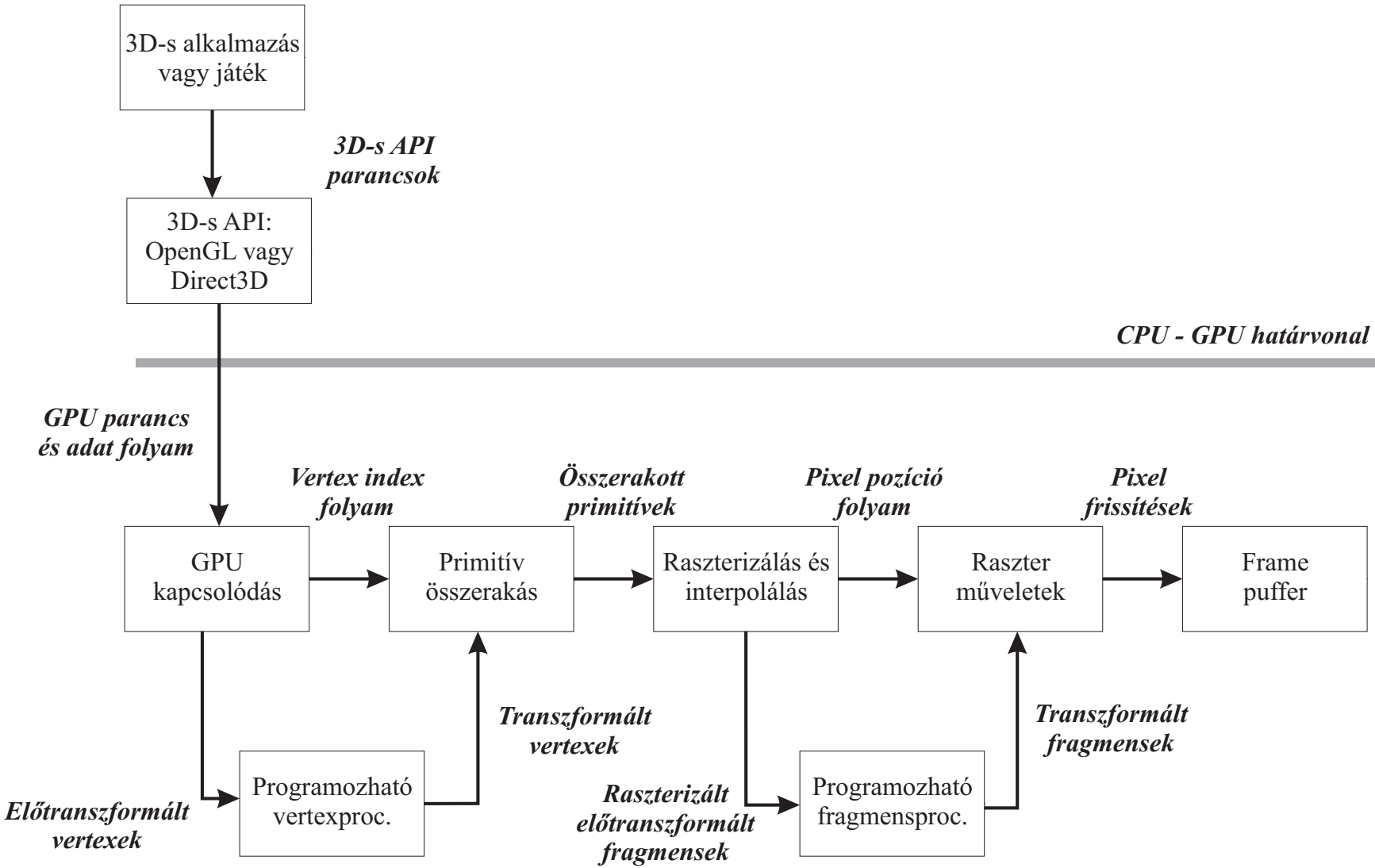
Poligon

Grafikus csővezeték

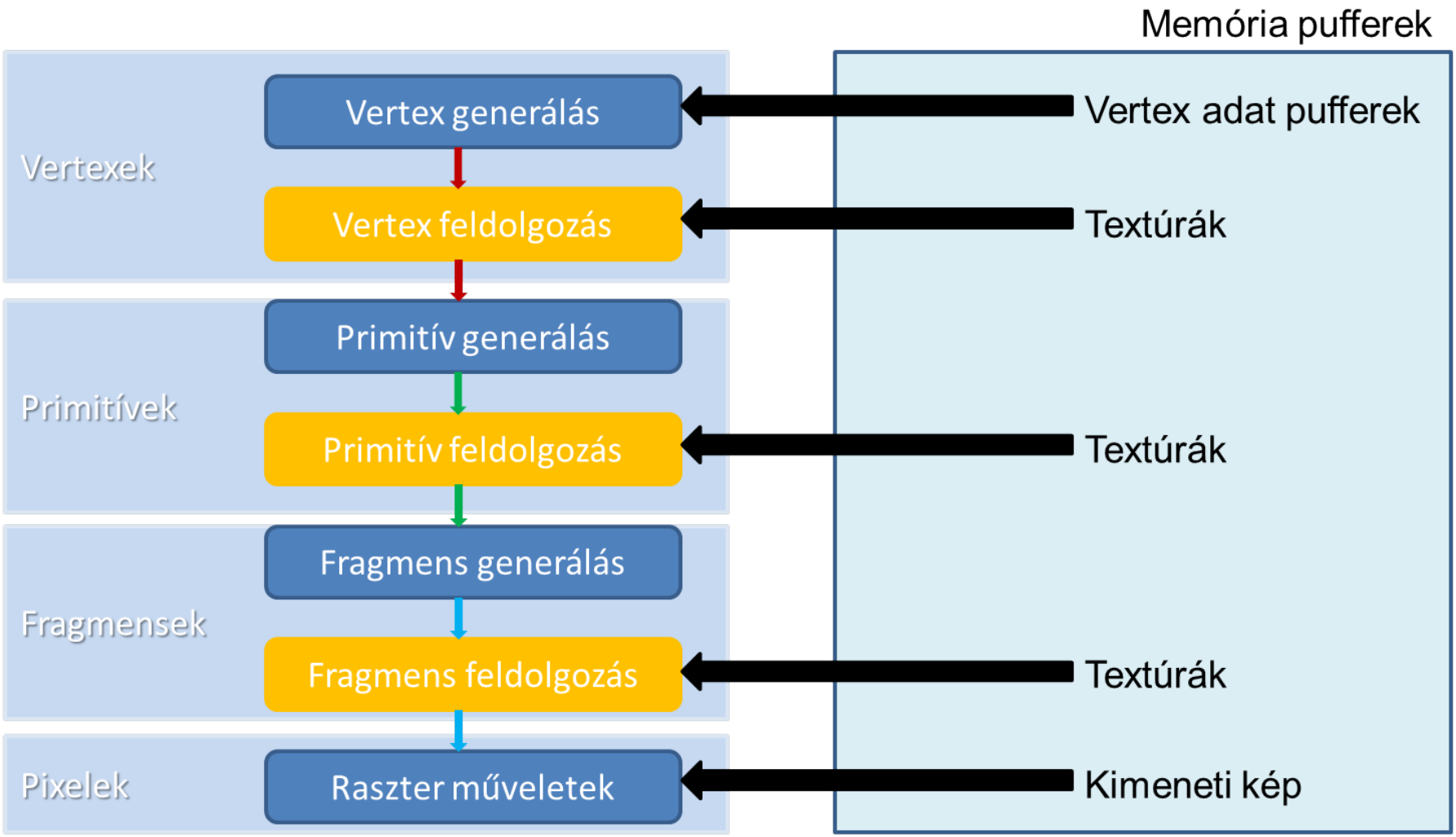



Színezett vertex
vertex transzformáció után
Primitív
összerakás
Raszterizálás
Interpoláció,
textúrázás és színezés

Programozható grafikus csővezeték



Grafikus csővezeték



GPU-k fejlődése

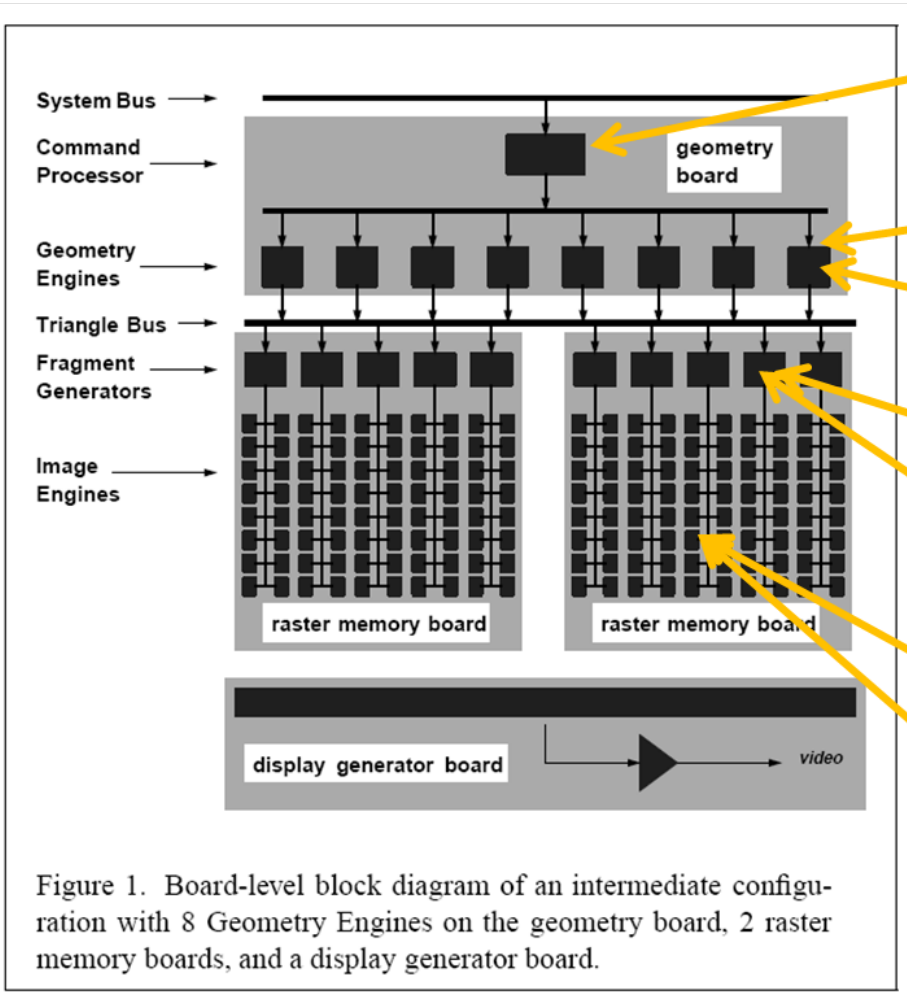
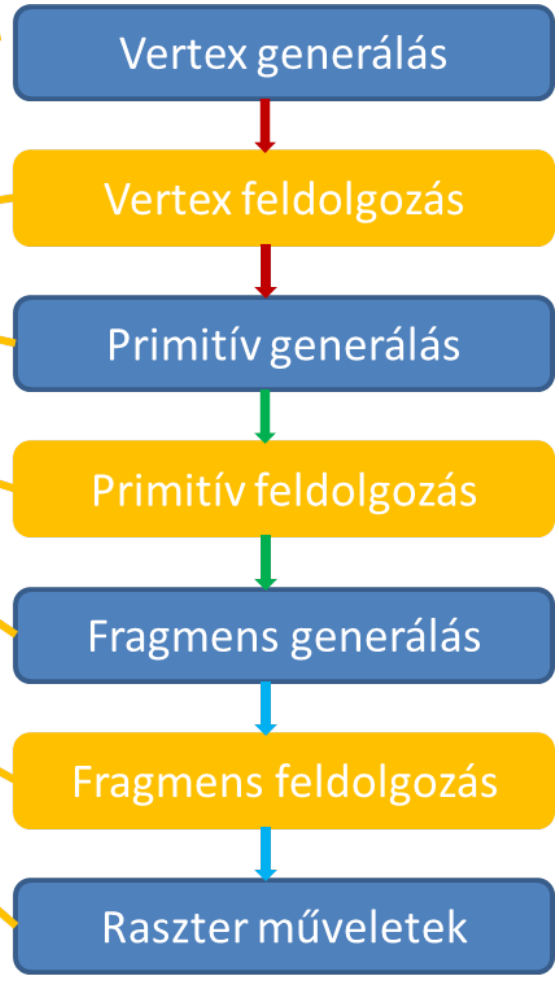


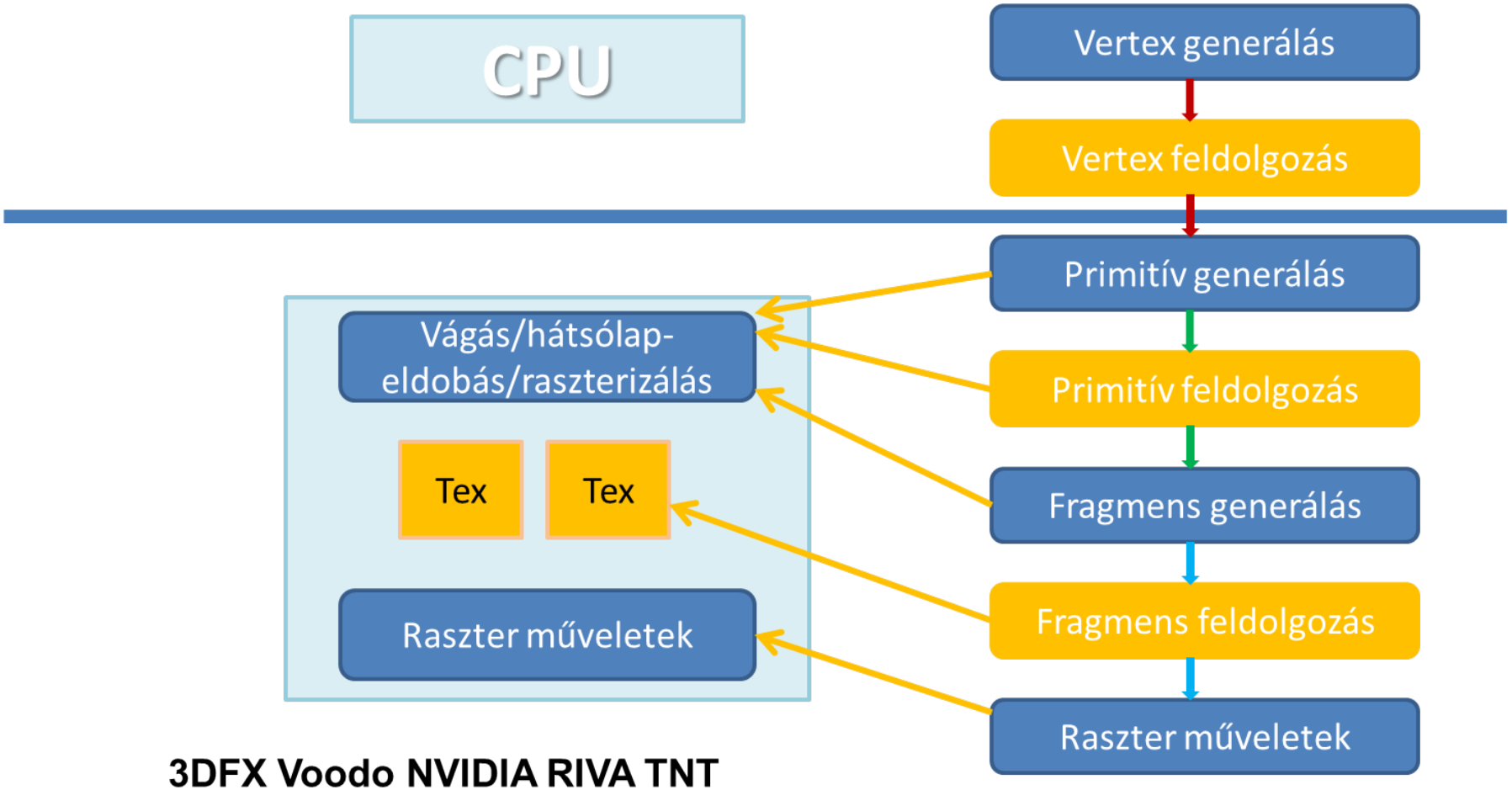
Figure 1. Board-level block diagram of an intermediate configuration with 8 Geometry Engines on the geometry board, 2 raster memory boards, and a display generator board.



Akeley, Kurt. "RealityEngine Graphics". Proceedings of SIGGRAPH '93, pp. 109-116.

Grafikus csővezeték

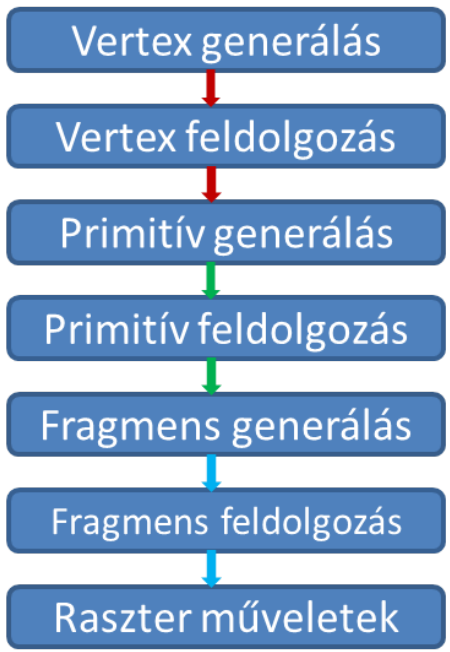
3D-s grafikus gyorsító 1999 előtt



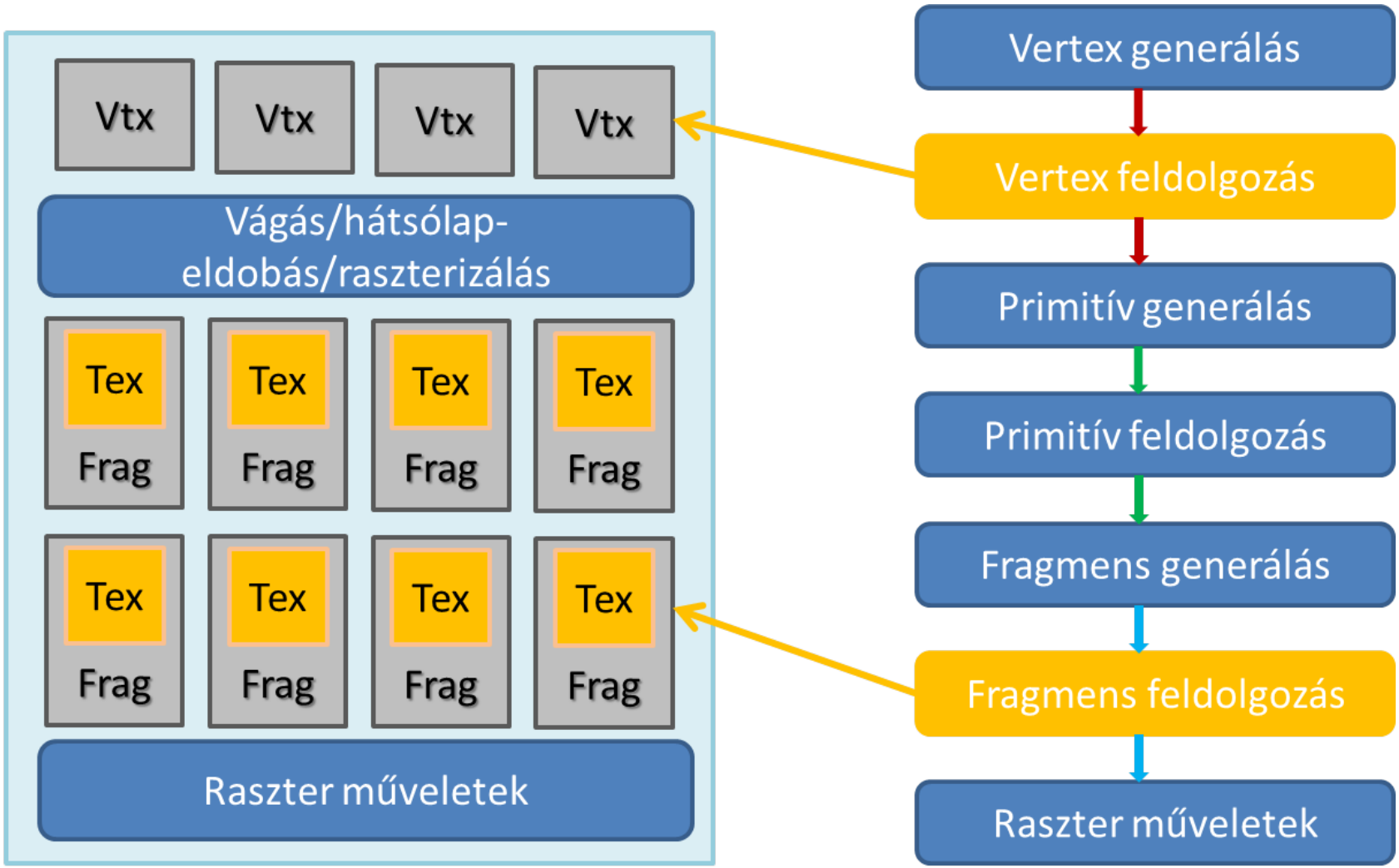
3DFX Voodo NVIDIA RIVA TNT

CPU

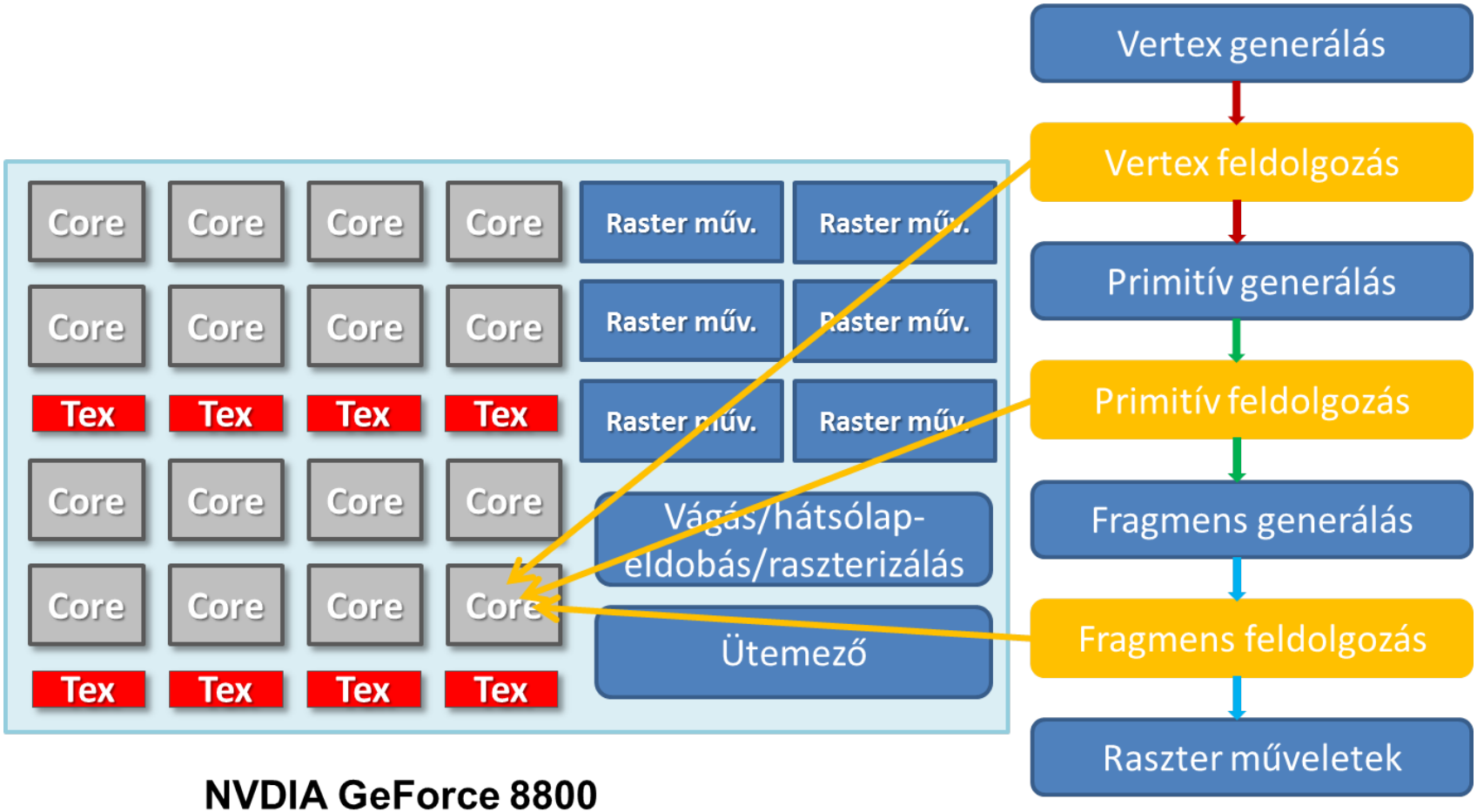
GPU



NVIDIA GeForce 256



ATI Radeon 9700

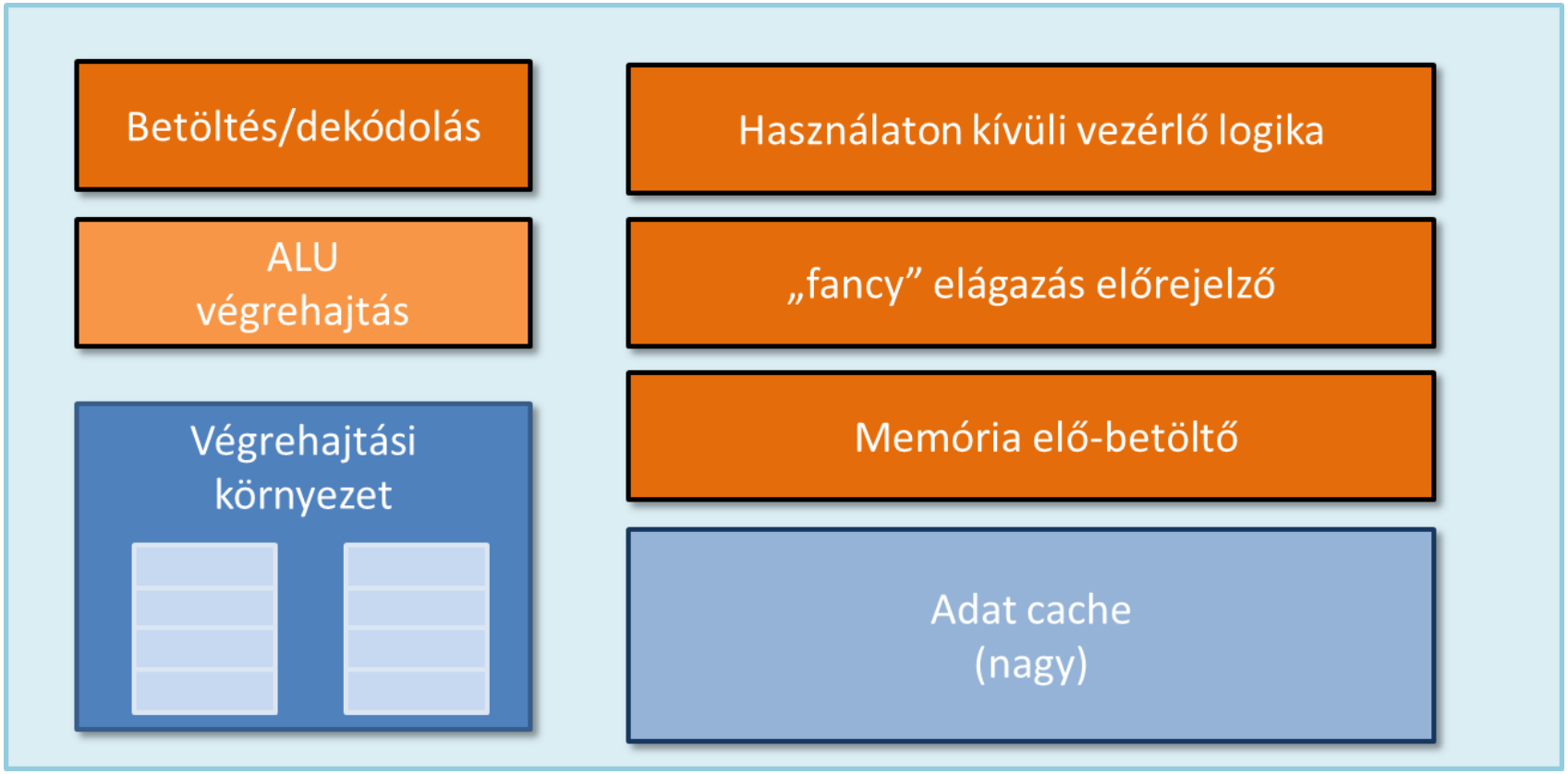


NVDIA GeForce 8800
egységes shader GPU

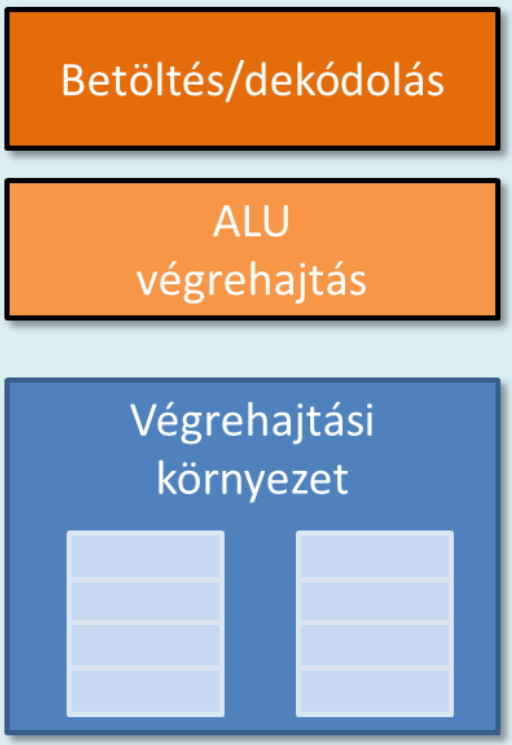
- Egységes shader modell
  - Shaderek megvalósítása közelebb került egymáshoz
    - Egyszerű
    - Kevés utasítás
    - Több száz általános célú végrehajtóegység
    - Hatalmas számítási kapacitás

Ötletek egy GPU felépítéséhez

CPU-stílusú core-ok



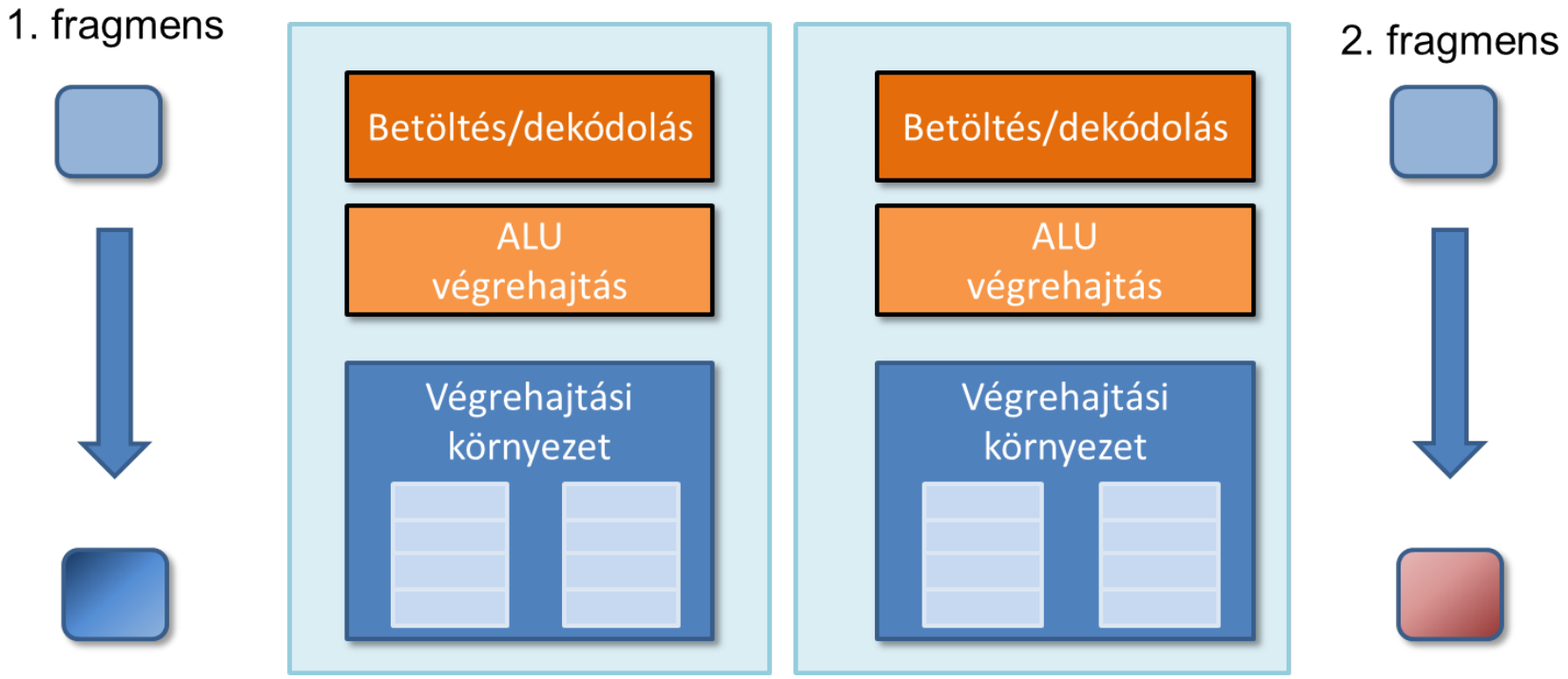
Első ötlet



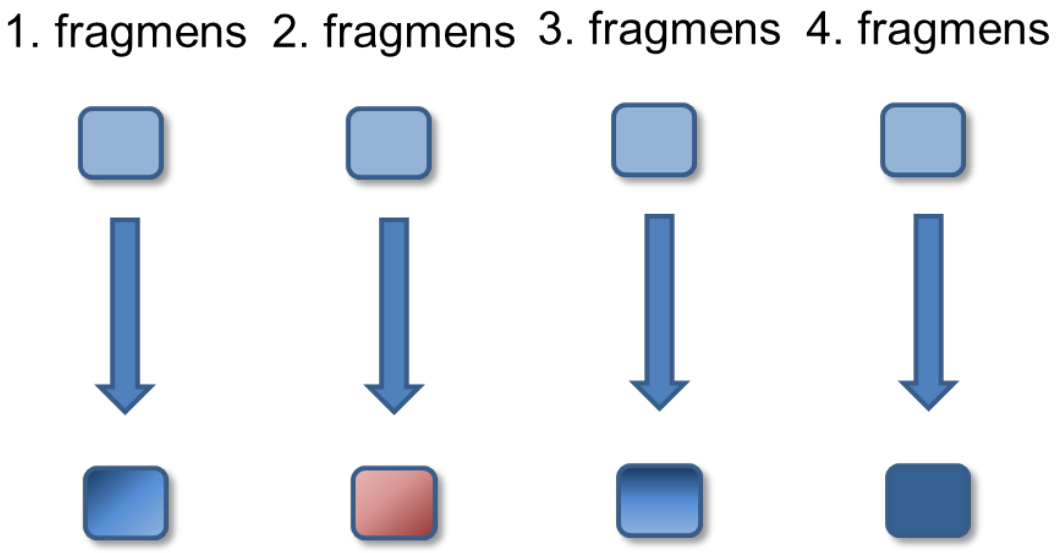
Tüntessük el azokat a komponenseket, amelyek az egyetlen utasítás folyam gyors futását segítik!

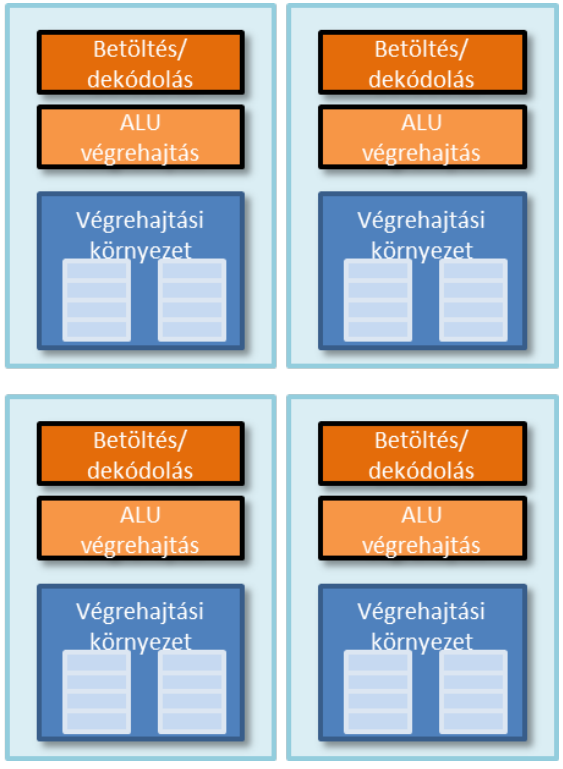
Két mag (core)

Két fragmens párhuzamos feldolgozása



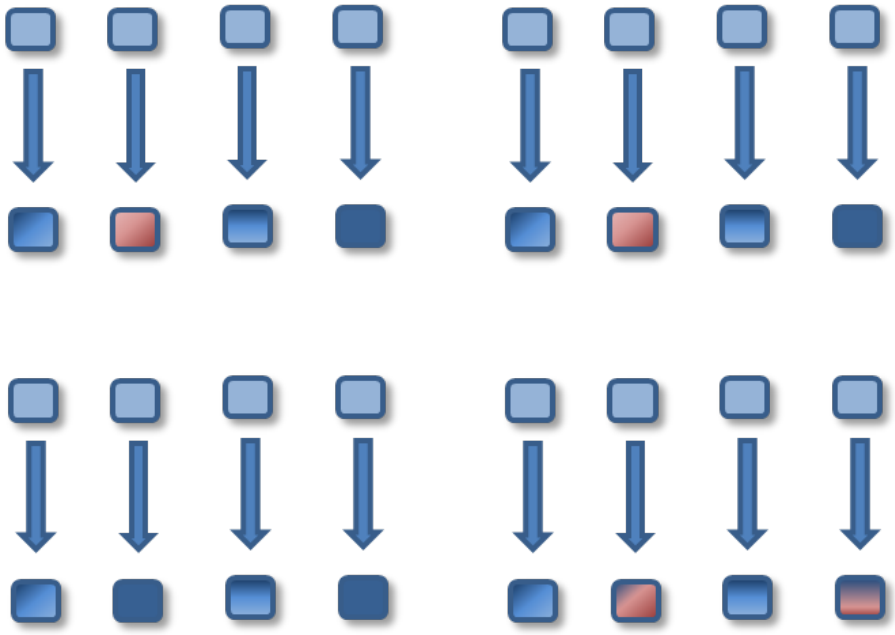
Négy mag (core)
Négy fragmens párhuzamos feldolgozása





Tizenhat mag (core)

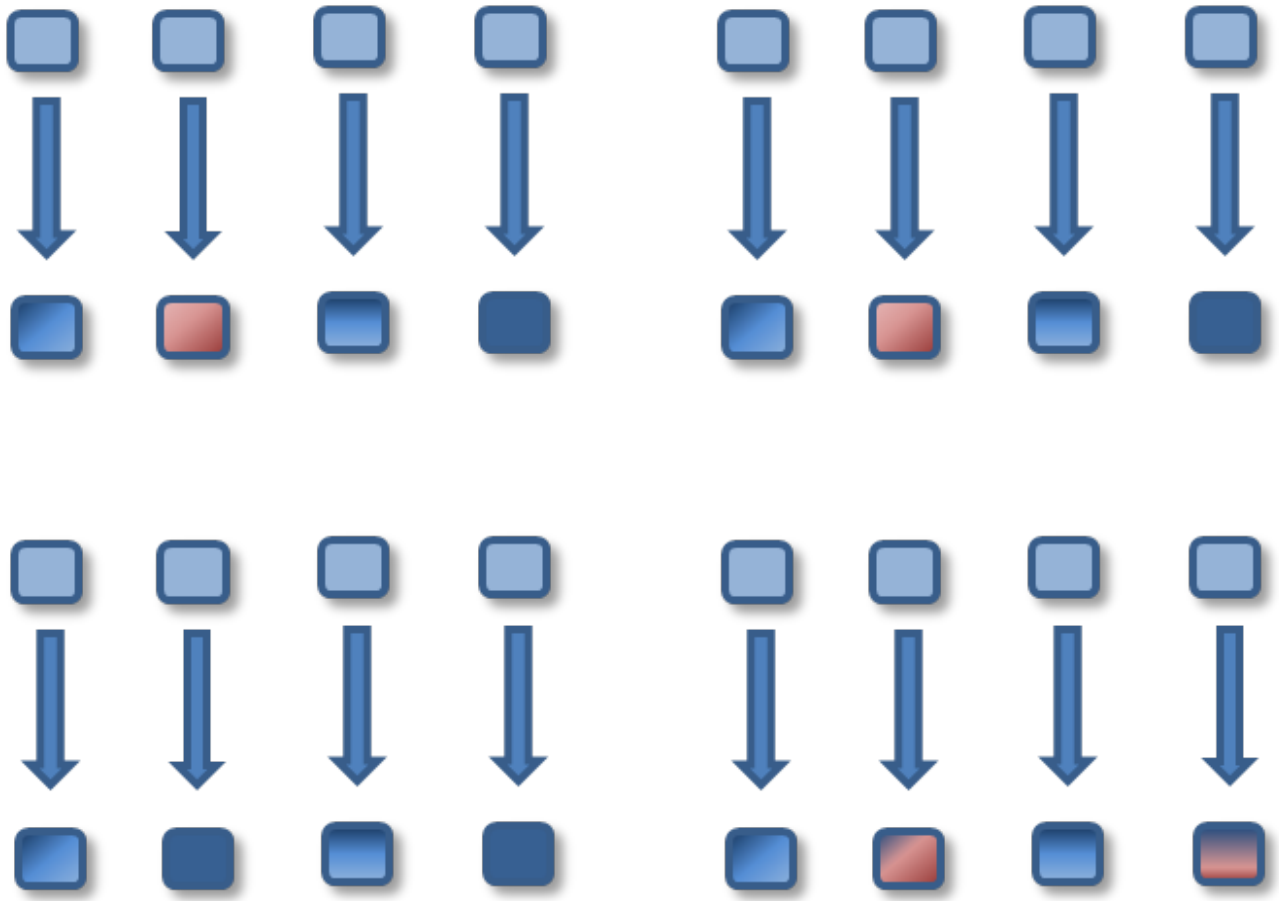
Tizenhat fragmens párhuzamos feldolgozása



16 mag = 16 egyidejű utasítás folyam

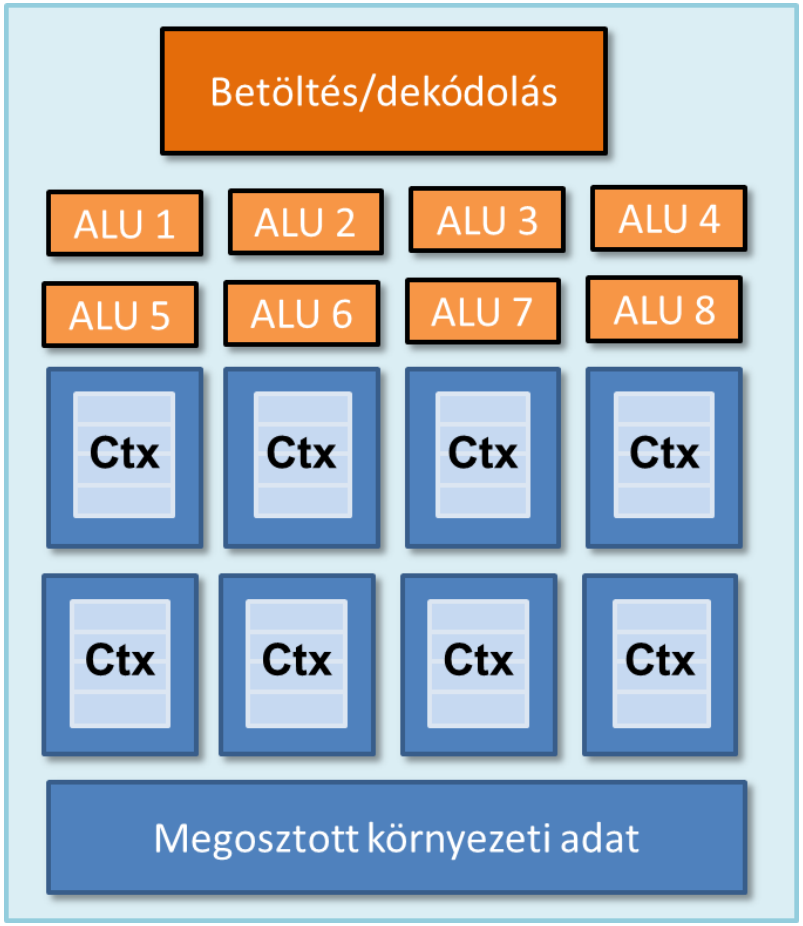
Második ötlet

Fragmensek közötti utasítás folyam megosztása



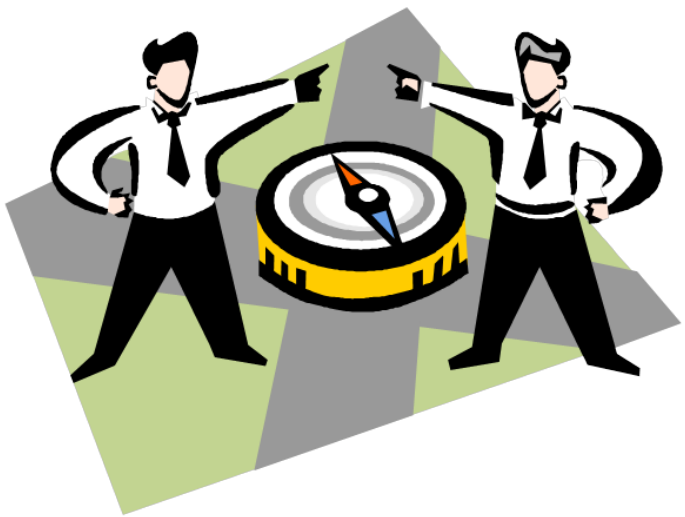
Második ötlet
Single Instruction Multiple Data (SIMD)

- Csökkentsük az ALU-k közötti utasítás folyam
  - Kezelésének költségét
  - Összetettségét

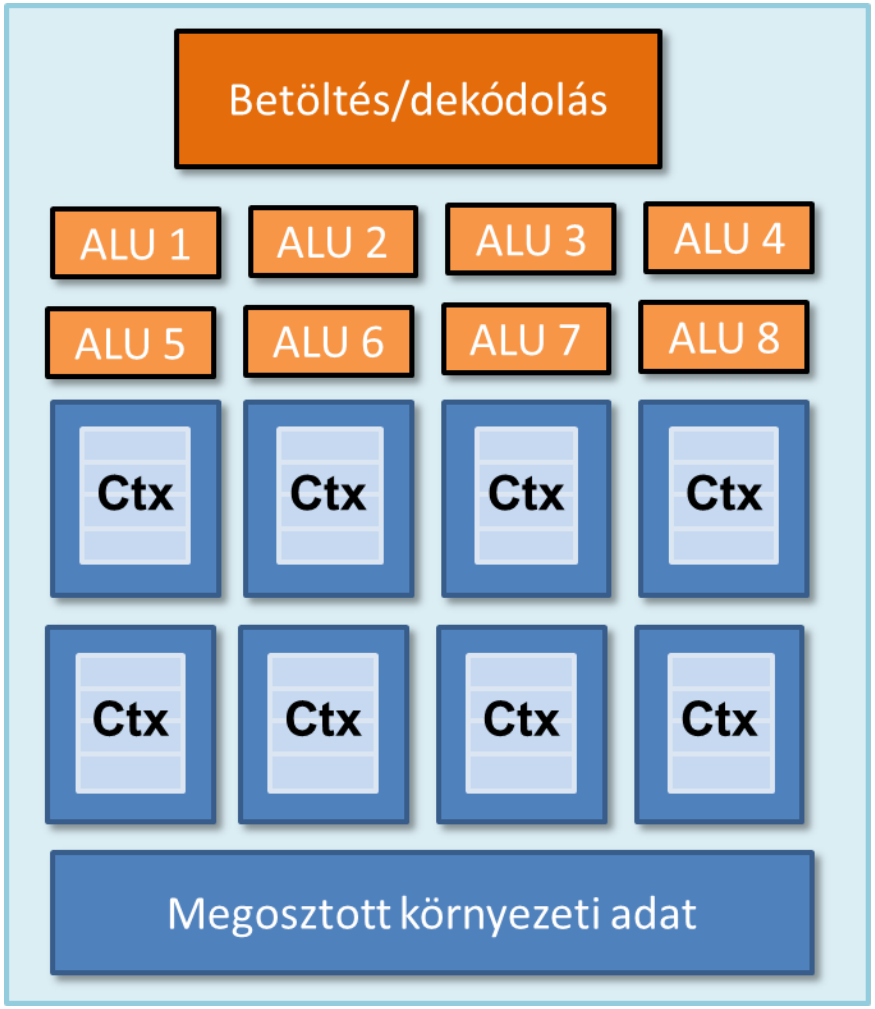


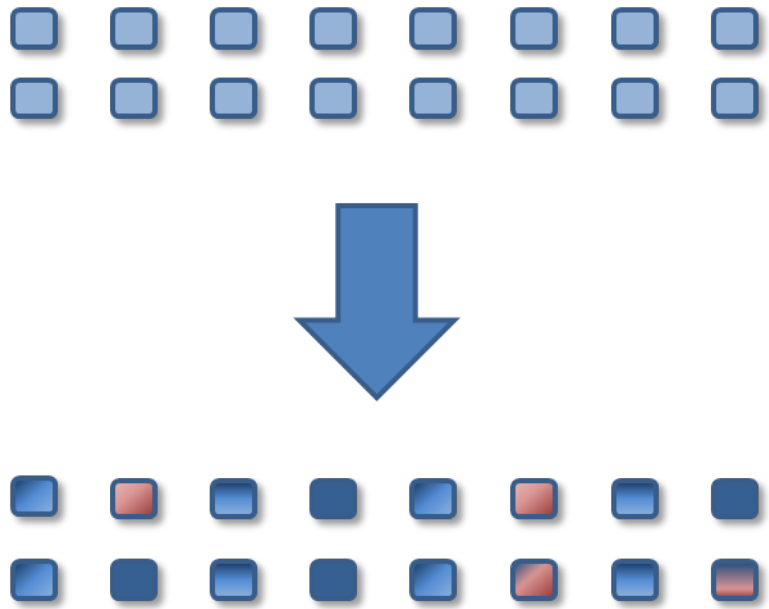
Második ötlet
Shader módosítása

- Előző shader egy fragmenst dolgozott fel
  - Skalár műveletek
  - Skalár operandusok
- Új shader nyolc fragmenst dolgoz fel
  - Vektor műveletek
  - Vektor operandusok



Második ötlet
Fragmensek közötti utasítás folyam megosztása

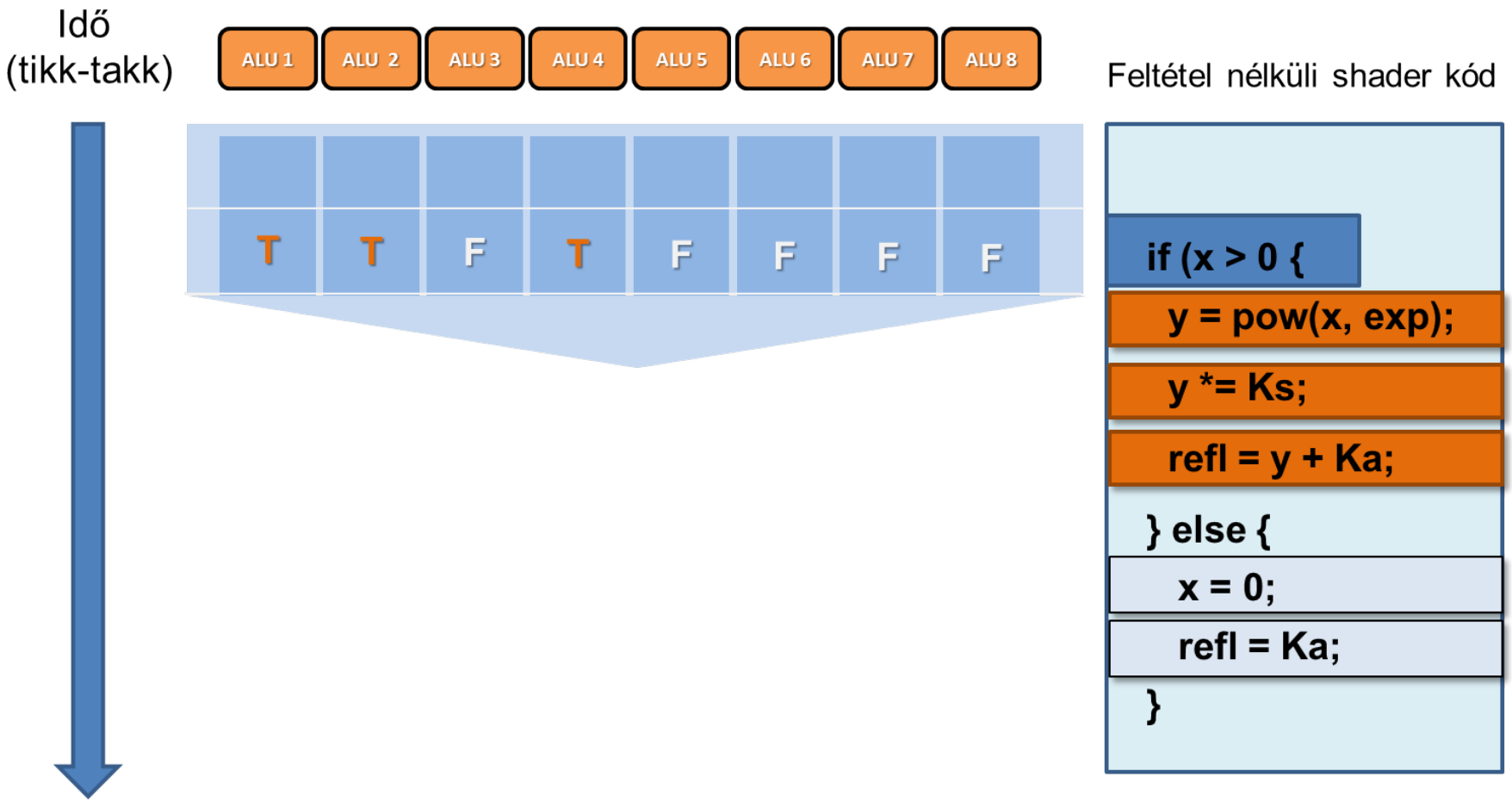




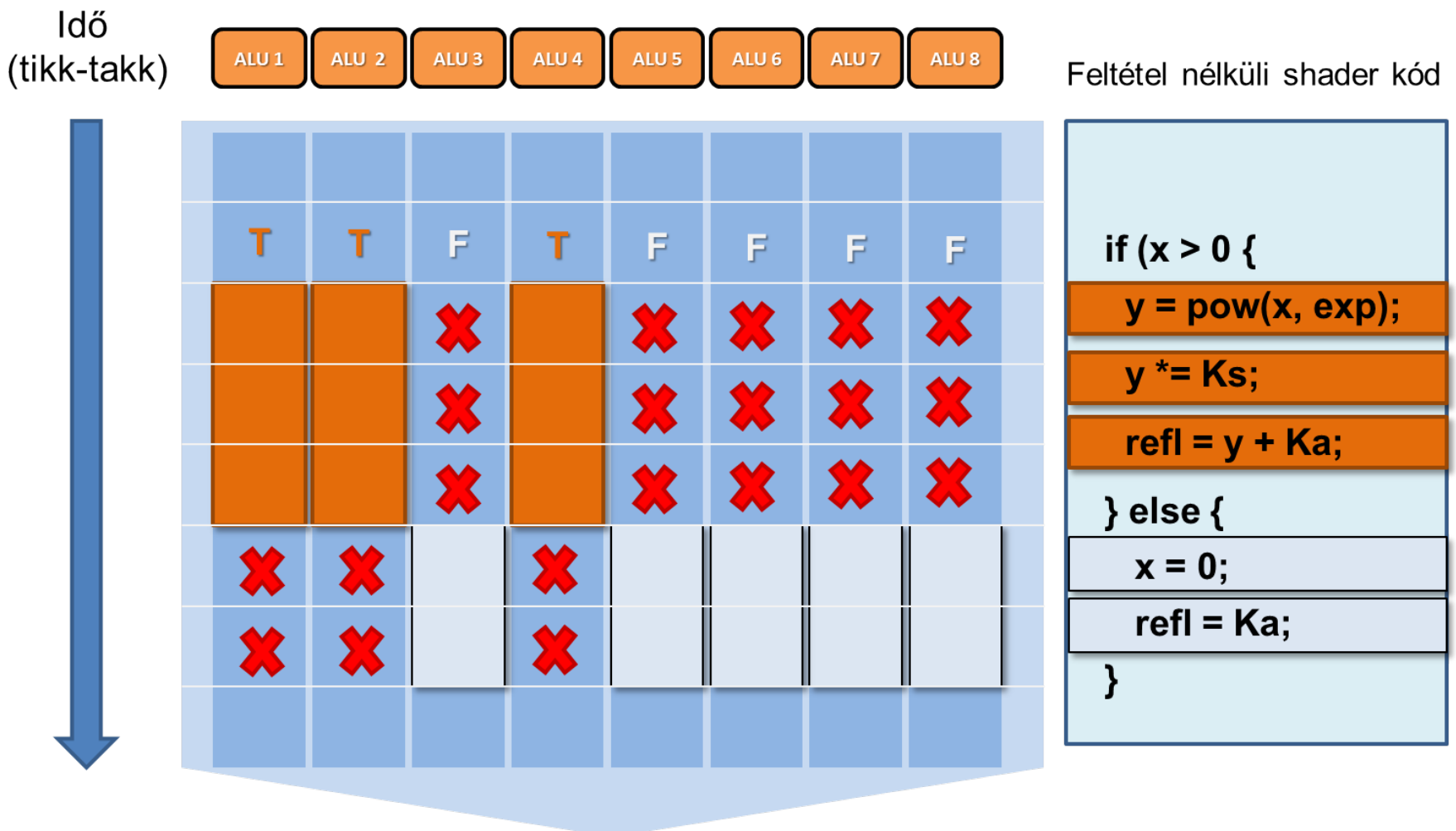
128 fragmens feldolgozása párhuzamosan

- 16 mag = 128 ALU
- 16 egyidejű utasításfolyam
- 128
  - Vertex
  - Primitív
  - Fragmens

Mi van az elágazásokkal?



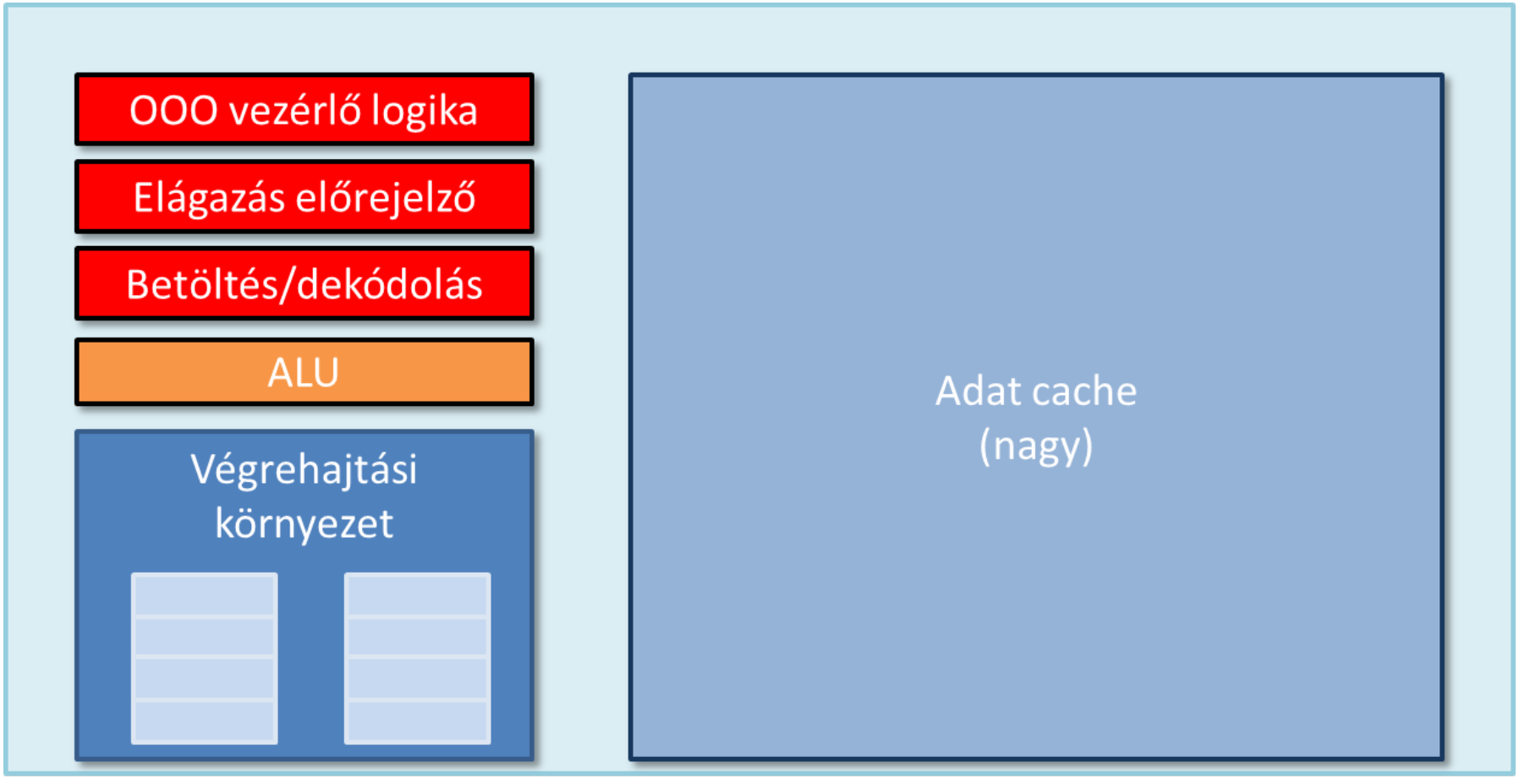
Mi van az elágazásokkal?
Nem mindegyik ALU végez hasznos munkát



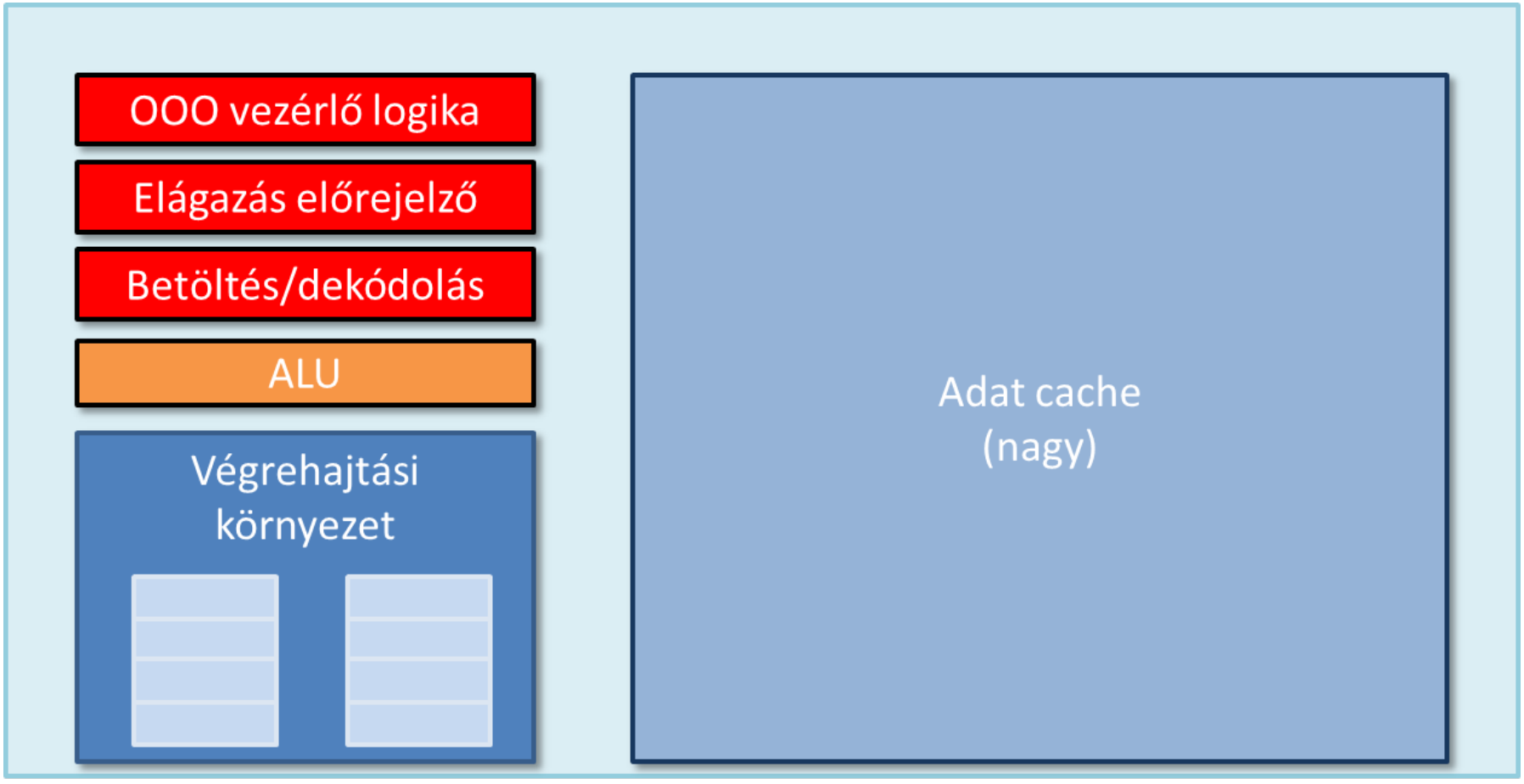
Állások (stalls)

- Állás akkor következik be, amikor egy mag (core) nem tudja futtatni a következő shader utasítást, mivel egy előző utasításra várakozik
  - Függőségek vannak az utasítás folyamban
    - Pl. ADD függ a LOAD befejezésétől
- Késleltetés
  - Adat elérése a memóriából sokszor 1000-nél több ciklust igényel
  - Rossz ötlet volt az első egyszerűsítés?
    - Az eltávolított részek segítenének az állások megoldásában
  - A GPU-k sok független feladatot tételeznek fel
    - Független SIMD csoportok

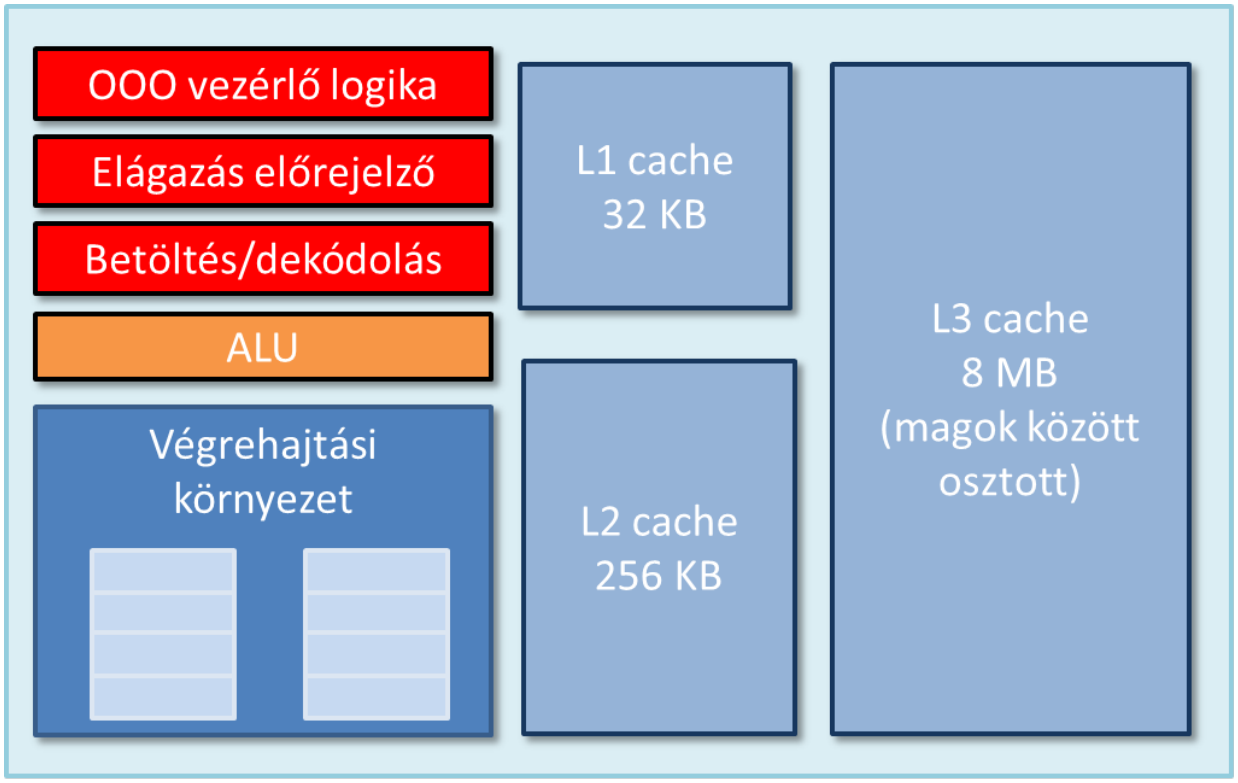
CPU-stílusú magok (core)



CPU-stílusú magok (core)



CPU-stílusú memória felépítés



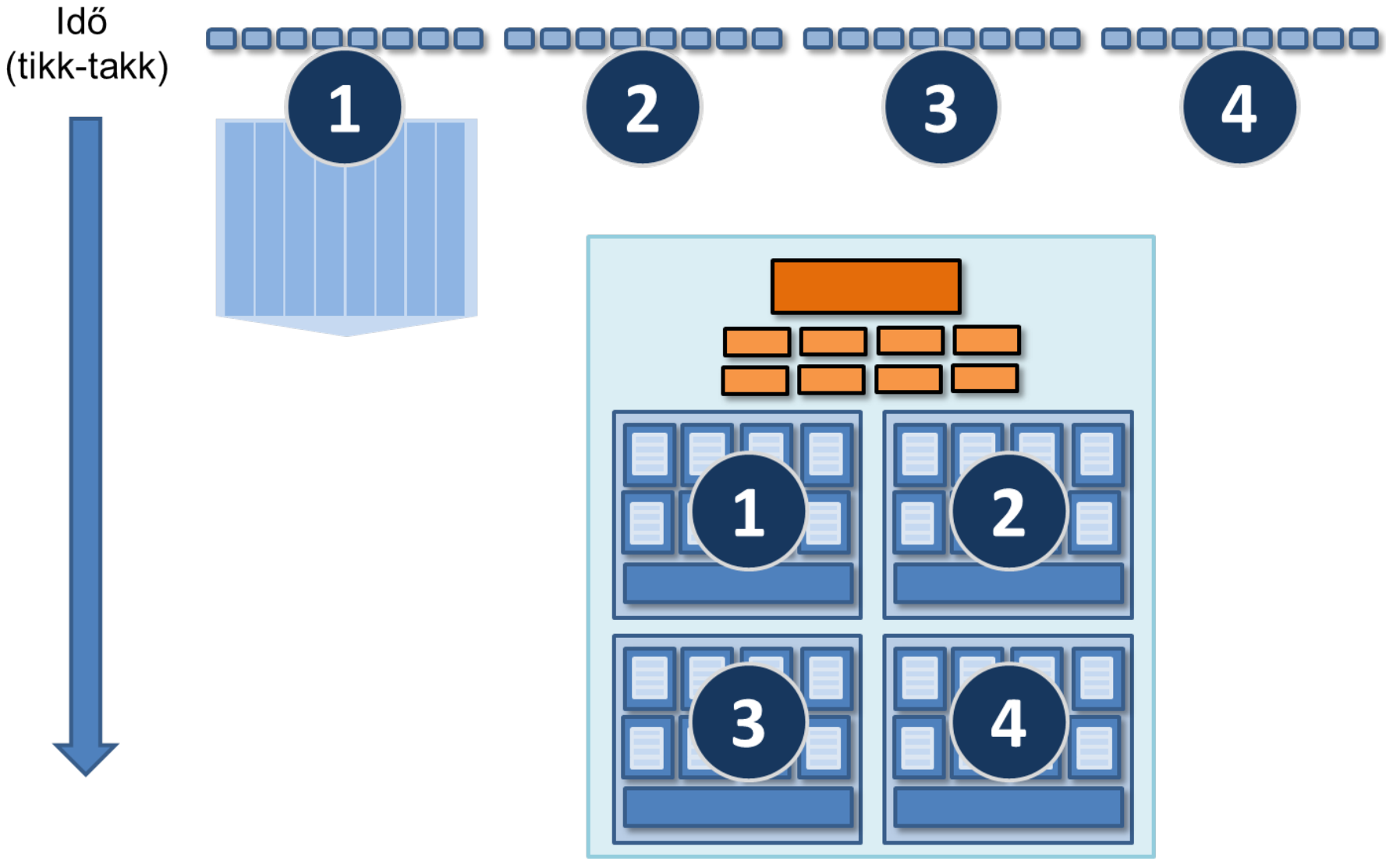
25 GB/sec elérés a memóriához

Magok hatékony kihasználása cache-ben lévő adatok esetén (késleltetés csökkentése, nagy sávszélesség)

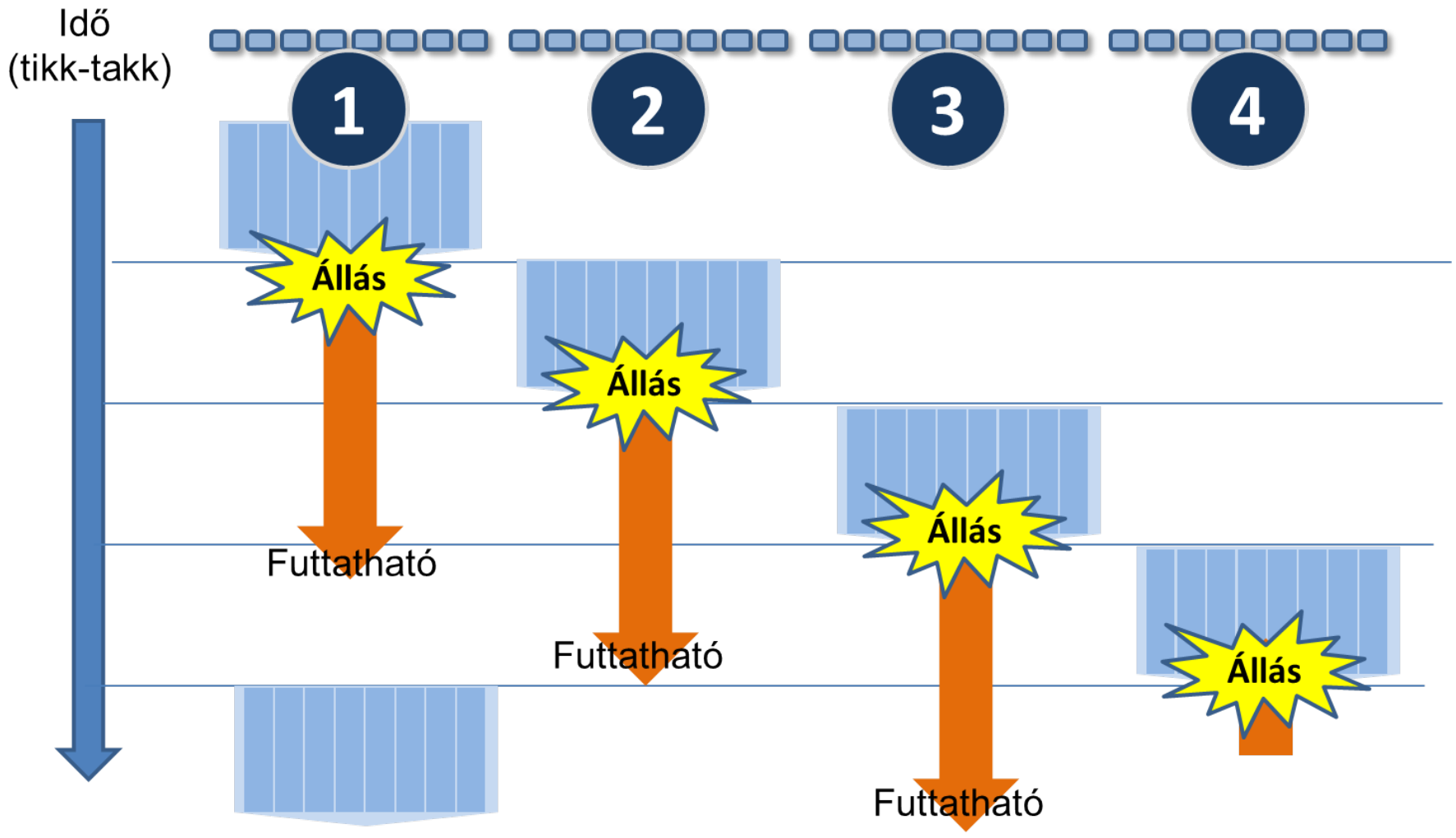
Harmadik ötlet

- Sok független fragmensünk van
- Sok fragmens összefésült feldolgozása egy magon
  - Utasítás folyam váltás egy másik (nem álló) SIMD csoportra, ha az aktív csoport áll
  - GPU hardveresen kezeli
    - Overhead mentesen
    - Ideális esetben teljesen láthatatlan
    - Maximális áteresztőképesség

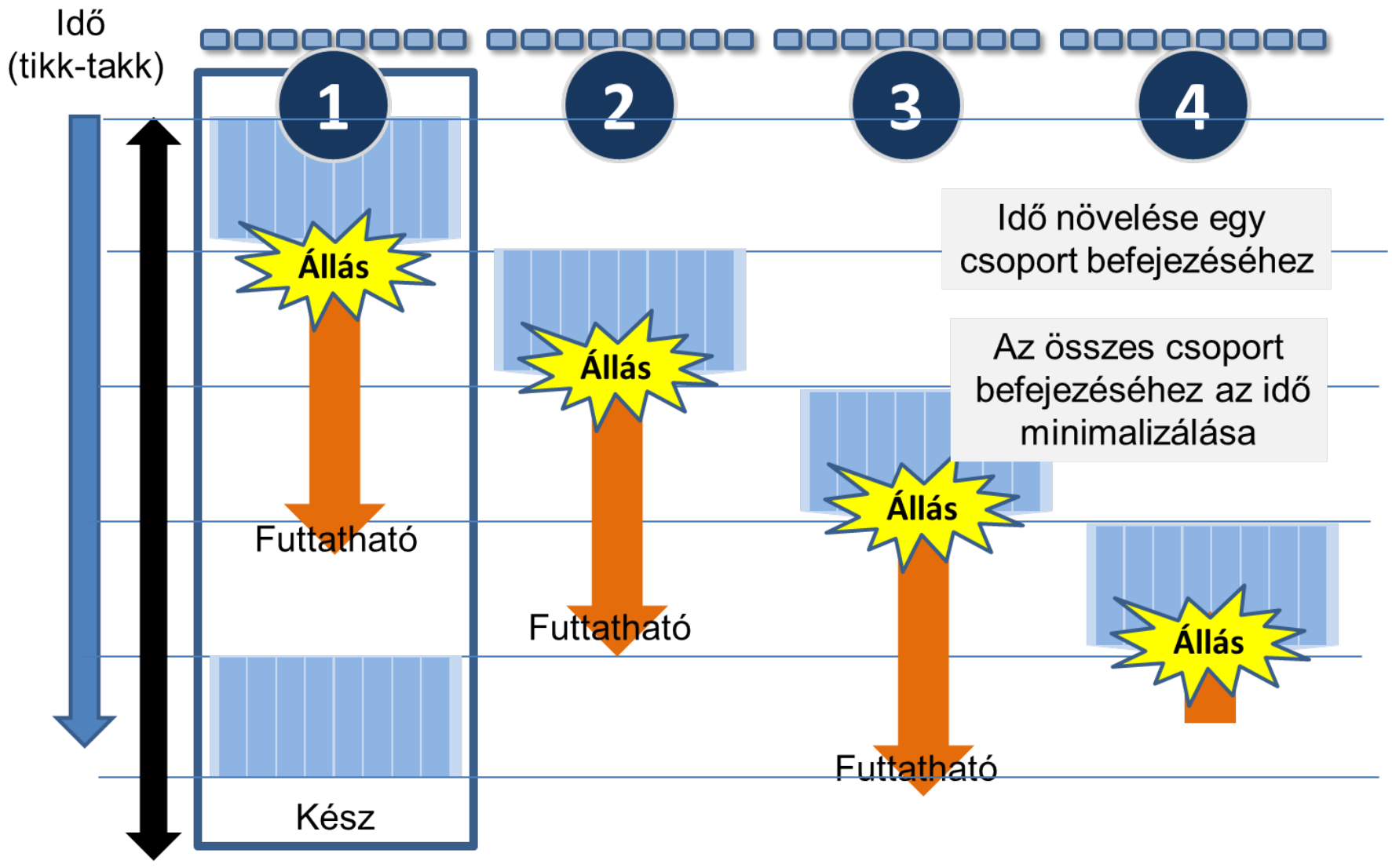
Shader állások elrejtése



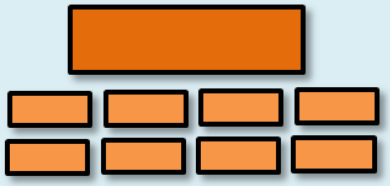
Shader állások elrejtése



Shader állások elrejtése



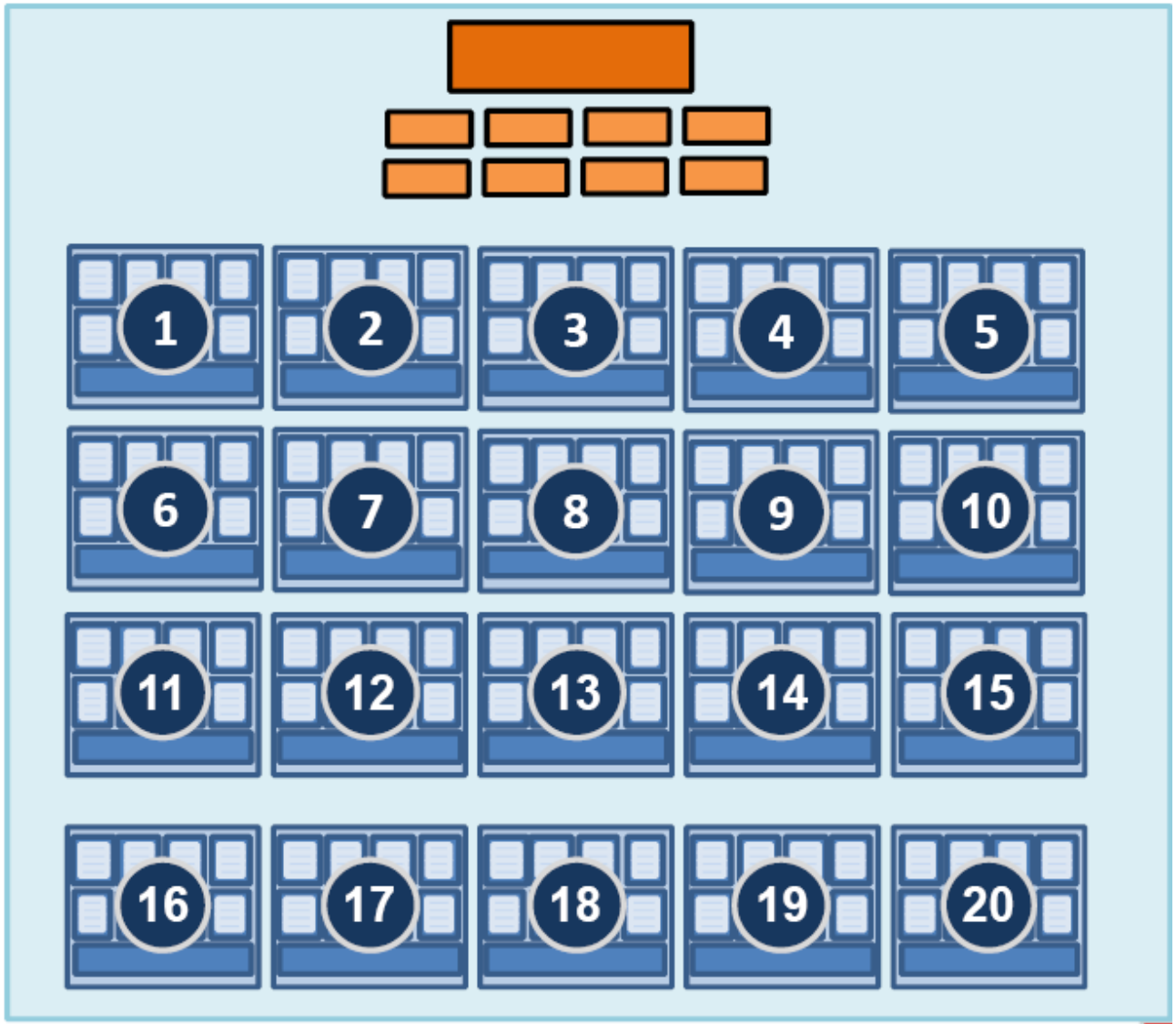
Közös környezet tárolása



Közös környezet készlet tárolás
64 KB

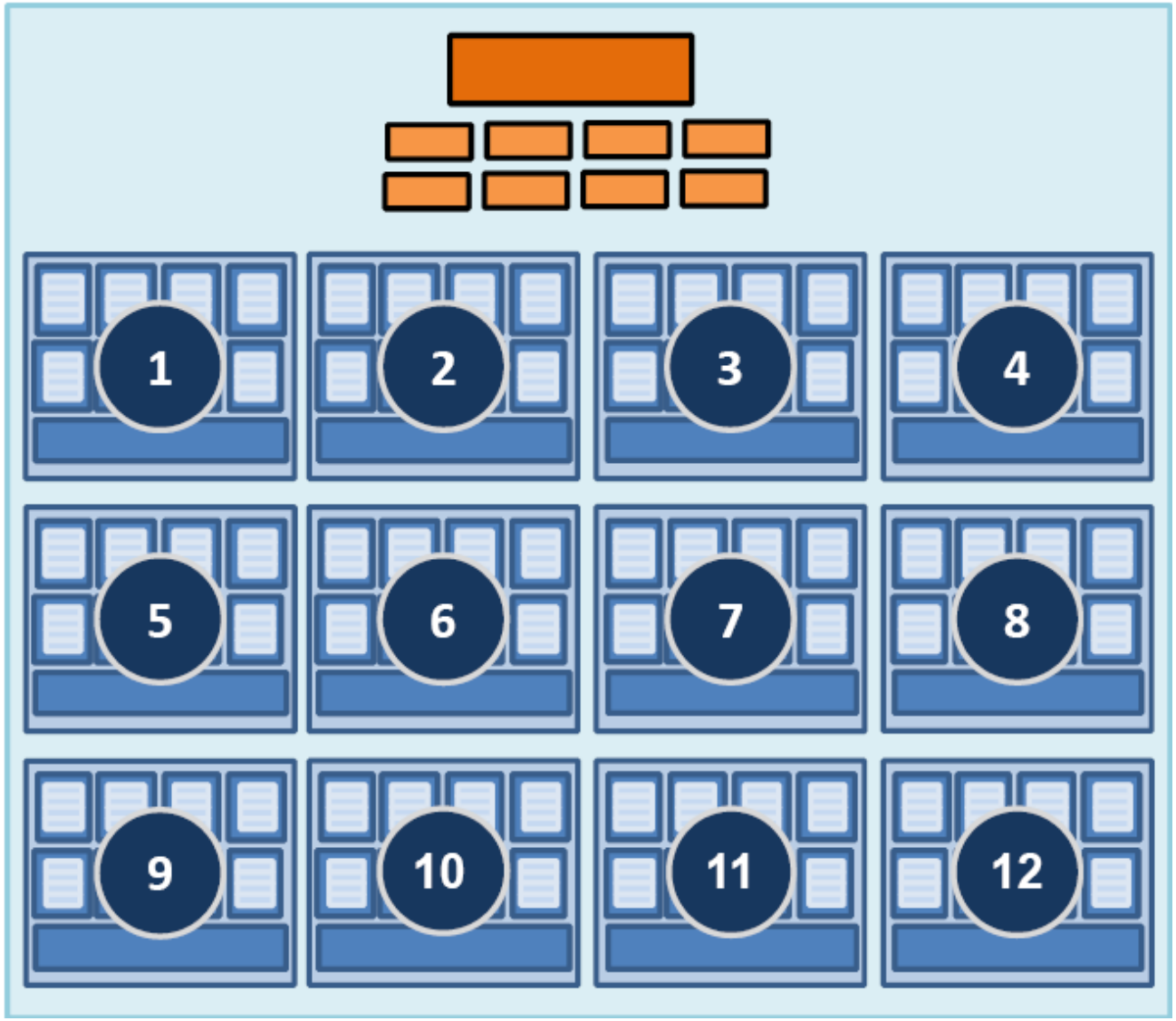
Maximális késleltetés elrejtési képesség

Húsz kicsi környezet



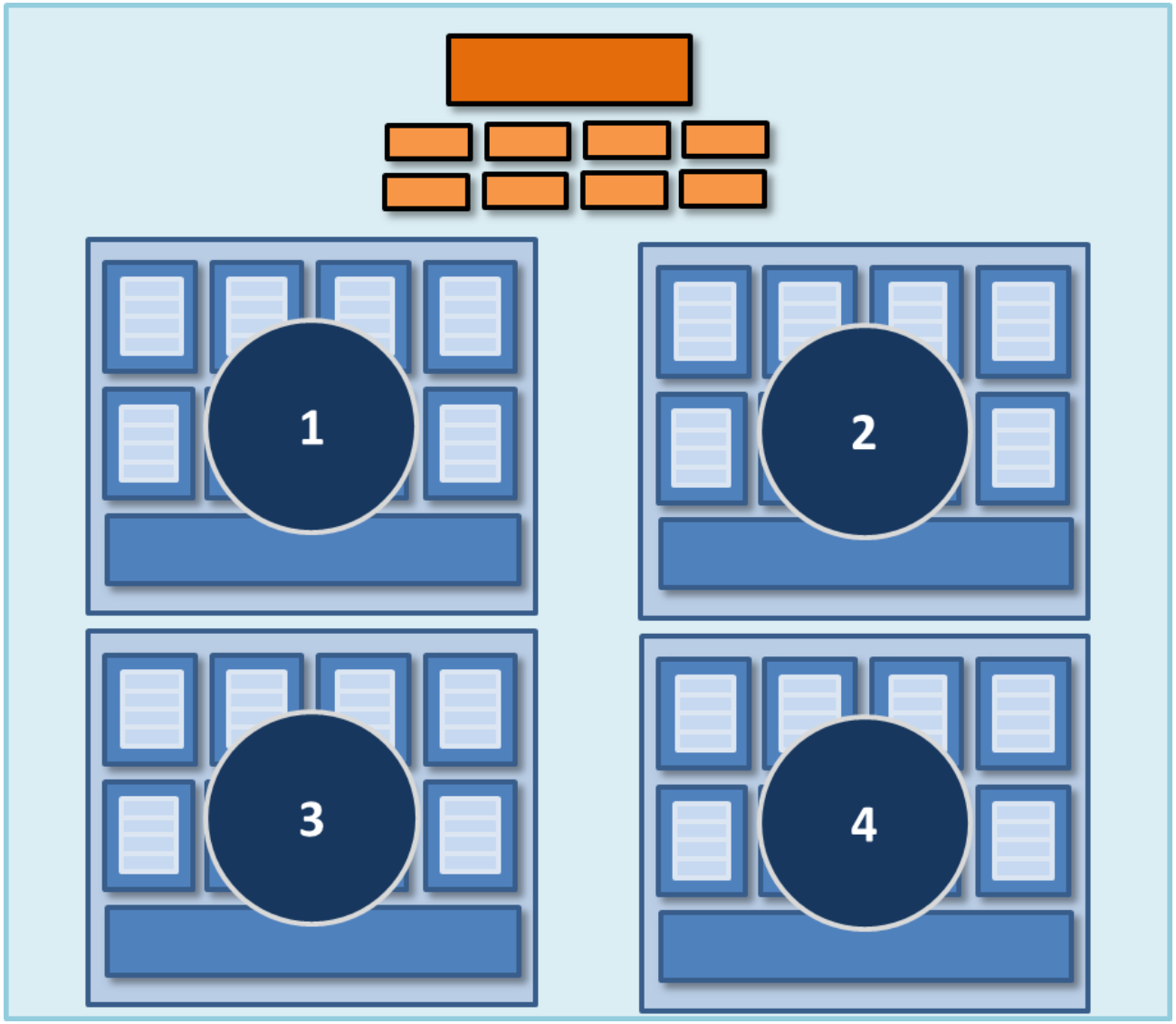
Közepes késleltetés elrejtési képesség

Tizenkét kicsi környezet



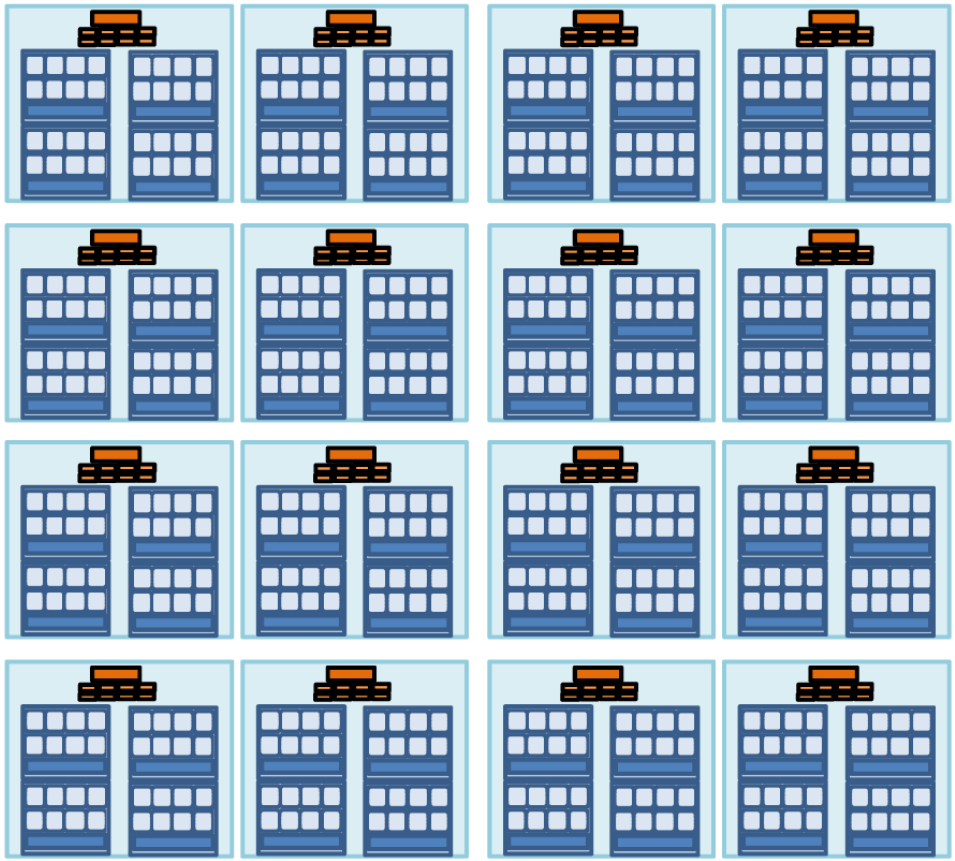
Kicsi késleltetés elrejtési képesség

Négy nagy környezet



GPU shading rendszer

- 16 mag
- 8 mul-add ALU magonként (128 összesen)
- 16 egyidejű utasítás folyam
- 64 konkurens (összefésült) utasítás folyam
- 512 konkurens fragmens



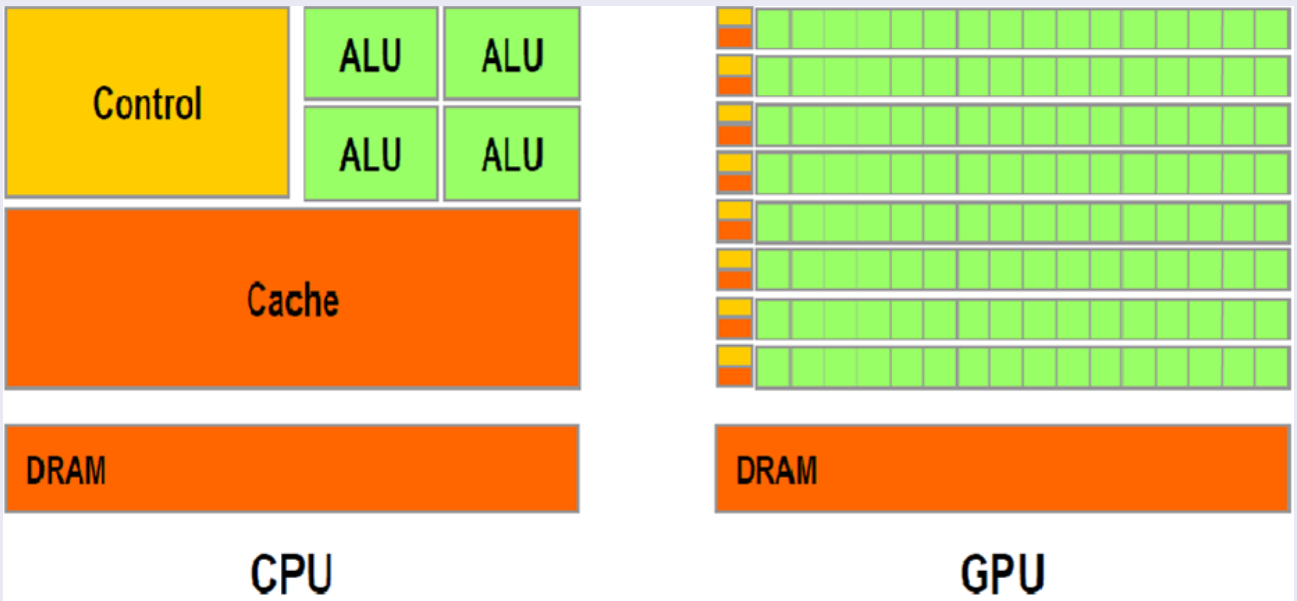
=256 GFLOPS (@1 GHz)

Összefoglalás

Három kulcs ötlet

- Használjunk sok, karcsúsított magot a párhuzamos futtatáshoz
- A magokat rakjuk tele ALU-kkal
  - Megosztott utasítás folyamok fragmensek csoportjainál
- Kerüljük el a késleltetett állásokat fragmens csoportok összefésült végrehajtásával
  - Amikor egy csoport áll, akkor dolgozzunk egy másik csoporton

CPU-GPU összehasonlítás



Összefoglalás

Emlékezzünk a következőkre!

- A GPU-ra egy több magos processzorként gondoljunk, amelyet arra optimalizáltak, hogy
  - A vertex és fragmens adatok maximális „áteresztéssel folynak át” a grafikus csővezetéken
  - Speciálisan támogatja
    - A grafikus csővezeték leképezését ezekre az erőforrásokra
    - A raszterizálást
    - Vágást
    - Hátsó oldal eltávolítást
    - Textúrázást
    - Stb.

- Nagyobb és gyorsabb
  - Több mag
  - Nagyobb FLOPS (manapság 2 TFLOP)
- Milyen fix-funkcióknak kell megmaradnia?
- Néhány CPU-hoz hasonló tulajdonság hozzáadása
  - Általános R/W cache (Fermi)
  - Szinkronizálás

- Alternatív programozási felületek támogatása
  - Általános célú programozás
    - CUDA
    - OpenCL
    - DirectCompute
  - Alkalmazások, amelyek a GPU-t egy több processzoros rendszernek tekintik
- Hogyan változik a grafikus csővezeték absztrakció?
  - Direct3D 11
    - 3 új csővezeték szakasz
- Sugárkövetés

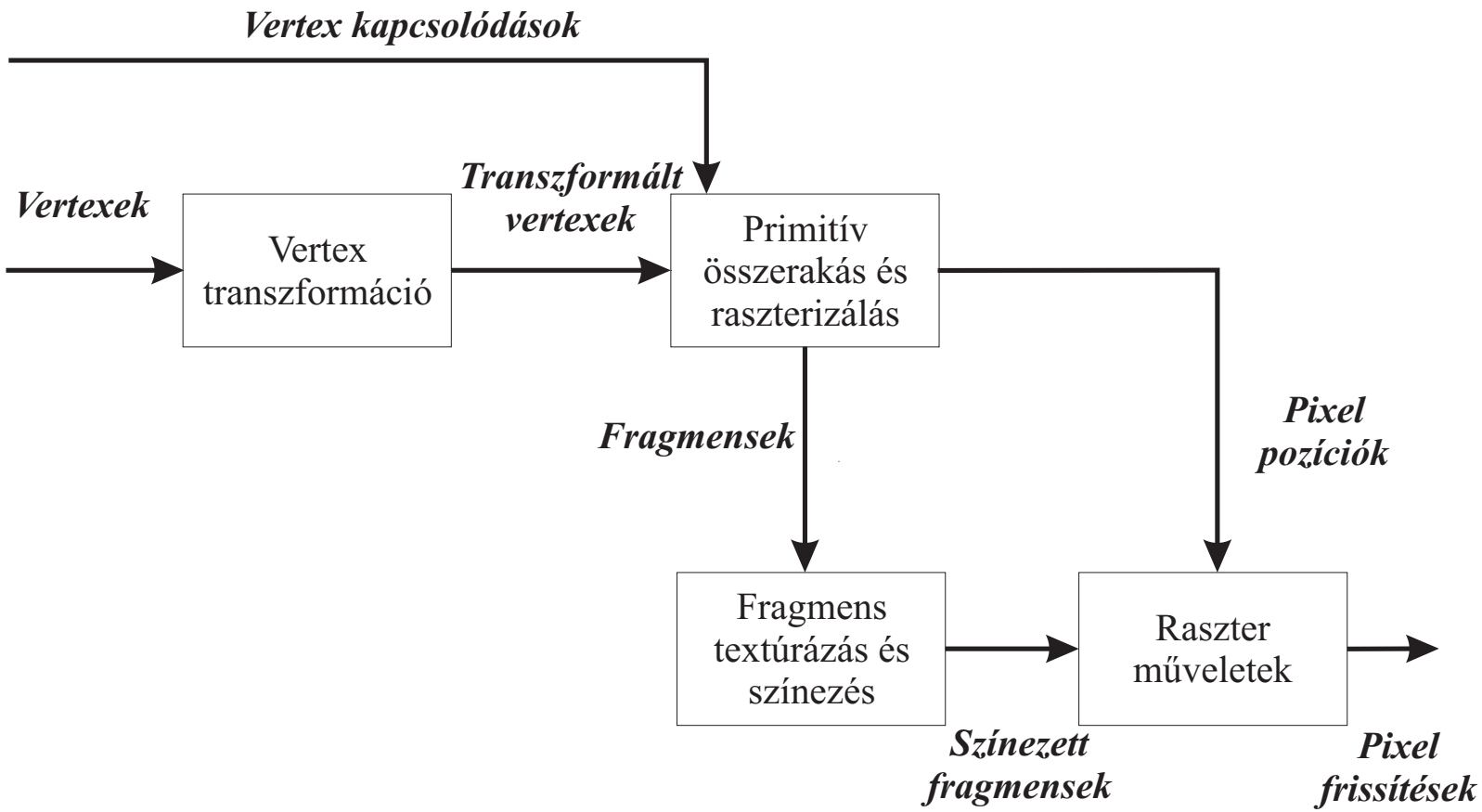
Grafikus csővezeték és az OpenGL függvénykönyvtár

A grafikus csővezeték

- 3D-s színtér objektumainak leírása primitívekkel:
  - pontok,
  - élek,
  - poligonok.
- Primitívek szögpontjait vertexeknek nevezzük
- Adott sorrendben végrehajtott műveletek segítségével áll elő a 2D-s kép
- Műveletek sorrendjét grafikus csővezetéknek nevezzük
  - Rögzített műveleti sorrendű grafikus csővezeték
  - Programozható grafikus csővezeték

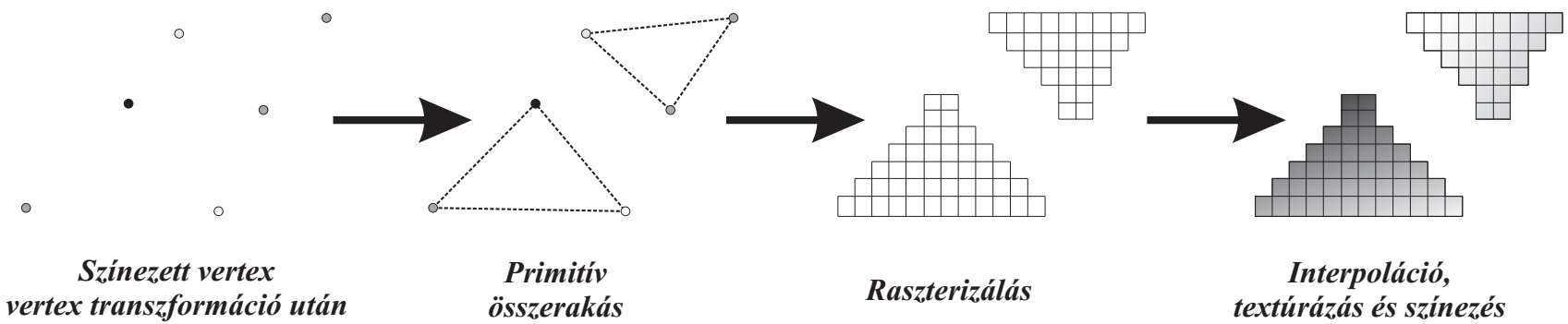
Rögzített műveleti sorrendű grafikus csővezeték

Rögzített műveleti sorrendű grafikus csővezeték



Rögzített műveleti sorrendű grafikus csővezeték

- Vertex transzformációk
  - Matematikai műveletek sorozata
  - Primitívek szögpontjainak meghatározása a képernyőn
  - Vertex attribútumok átadása

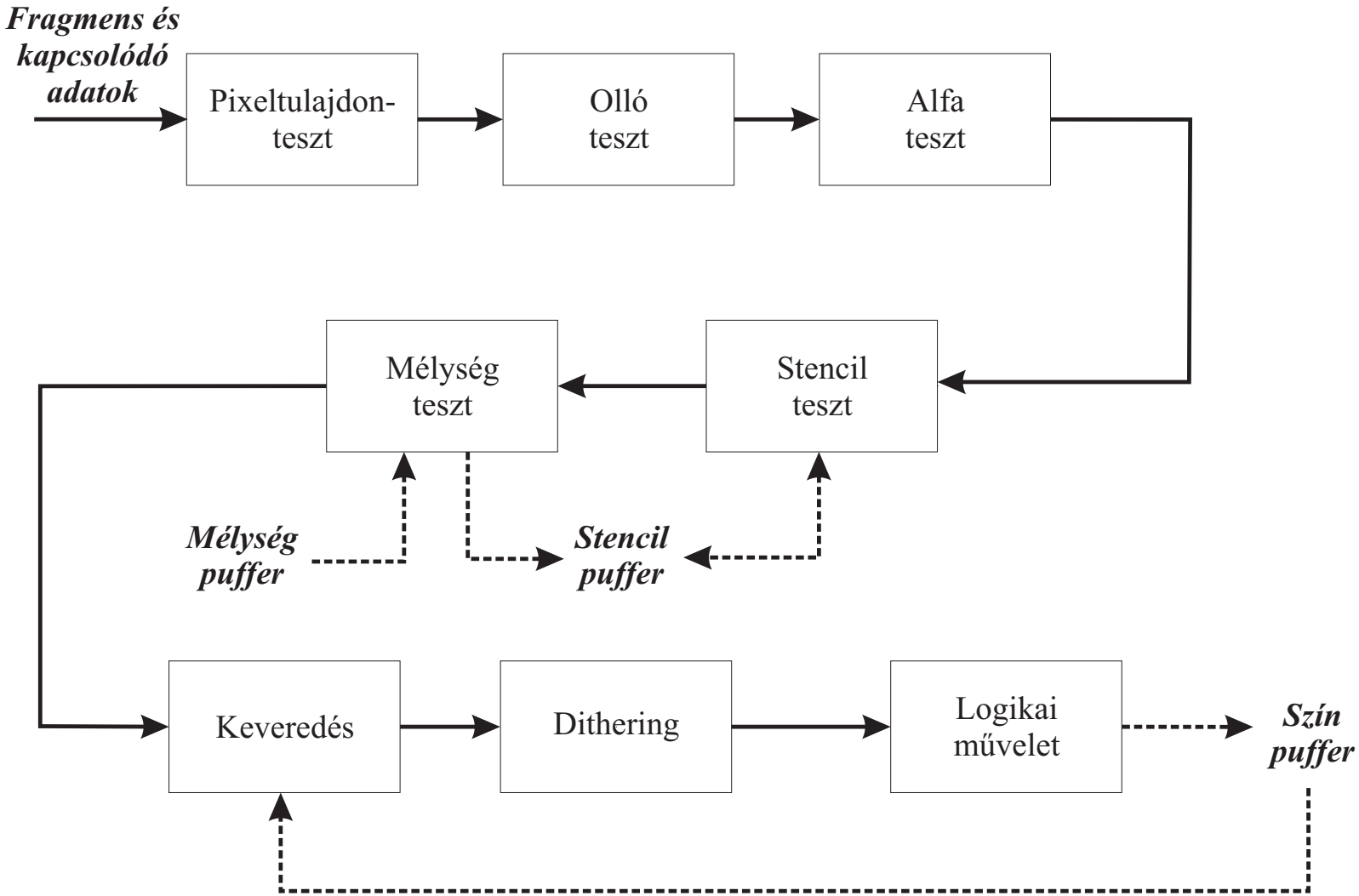


Rögzített műveleti sorrendű grafikus csővezeték

- Primitív összerakás és raszterizálás
  - A vertexek primitívekbe vannak szervezve kapcsolódási információk alapján
  - Vágás
    - A 3D-s színtér látható térfogata
    - Alkalmazás által definiált vágósíkok
    - Raszterizáló eldobhat poligonokat
  - Geometriai primitívek lefedése
  - Fragmensek

Rögzített műveleti sorrendű grafikus csővezeték

- Fragmens textúrázás és színezés
  - Mindegyik fragmensre textúrázás és matematikai műveletek végrehajtása
    - Transzformált vertexekből származó interpolált szín
    - Interpolált textúra koordináták
    - Fragmenshez tartozó texel kinyerése
    - Fragmens szín

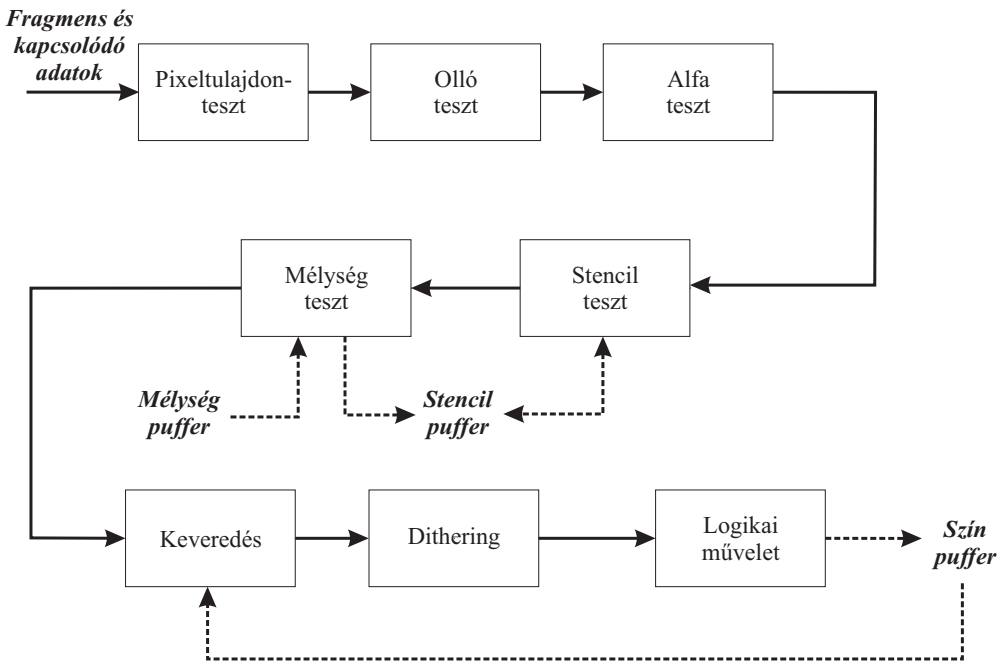

Fragmens és kapcsolódó adatok
Pixeltulajdon-teszt
Olló teszt
Alfa teszt
Mélység teszt
Stencil teszt
Mélység puffer
Stencil puffer
Keveredés
Dithering
Logikai művelet
Szín puffer

- Pixeltulajdon-teszt
  - Képernyő adott pixelére írhatunk-e
- Olló teszt
  - Olló téglalapra korlátozott kirajzolás
- Alfa teszt
  - Fragmens alfa értékének összehasonlítása egy előre megadott értékkel
  - Adott reláció mellett kapott hamis értéknél a fragmens eldobódik

- Stencil-teszt
  - Fragmens pozíciójának megfelelő stencilpuffer érték összehasonlítása egy előre megadott értékkel
  - Ha az összehasonlítás eredménye hamis, akkor a fragmens eldobódik
  - Stencilpuffer értékének módosítása
    - Műveletek megadása sikeres és sikertelen stencil teszt esetén
    - Sikeres stencil teszt és a mélység teszttől függő művelet megadása

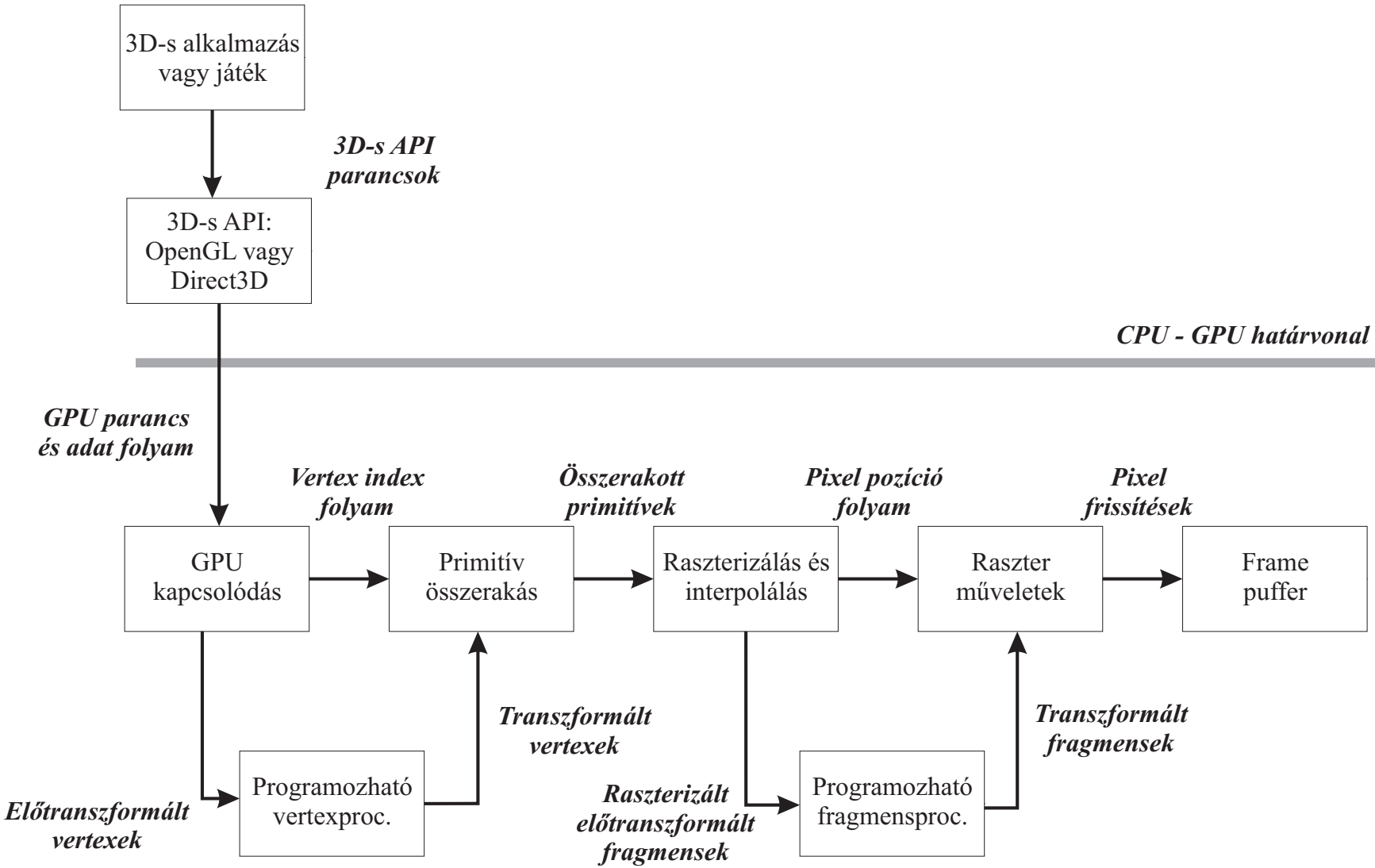
- Mélység teszt
  - Fragmens mélység értékének összehasonlítása a mélységpufferben levő értékkel
  - Sikeres teszt esetén frissül a színpuffer és a mélységpuffer
    - Alap esetben a nézőponthoz közelebbi fragmens fog bekerülni a színpufferbe

- Keveredés: végső fragmens és pixelek egyesítése
- Dithering: a színmélység javítása a térbeli felbontás rovására
- Logikai műveletek: OR, XOR vagy INVERT



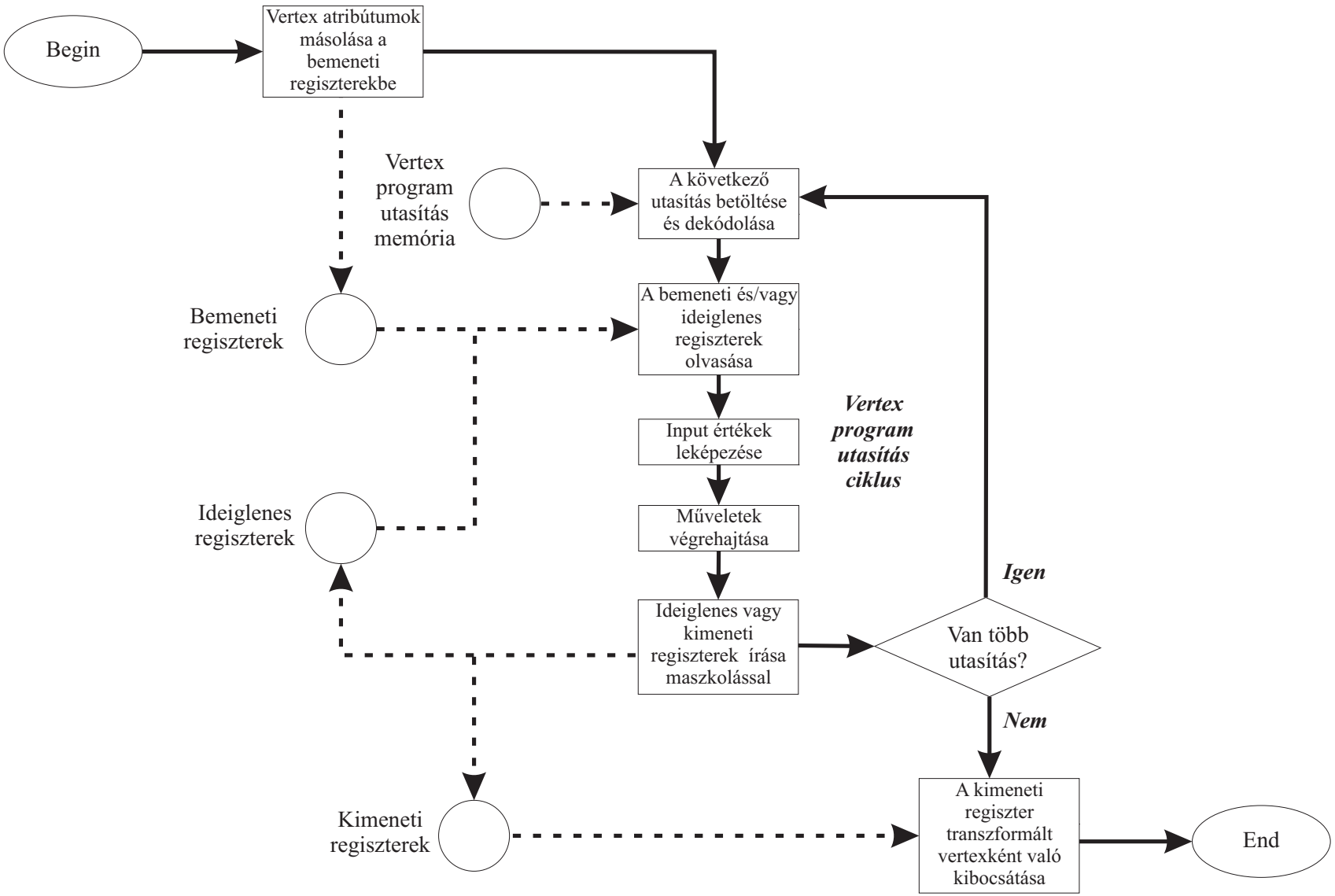
Programozható grafikus csővezeték

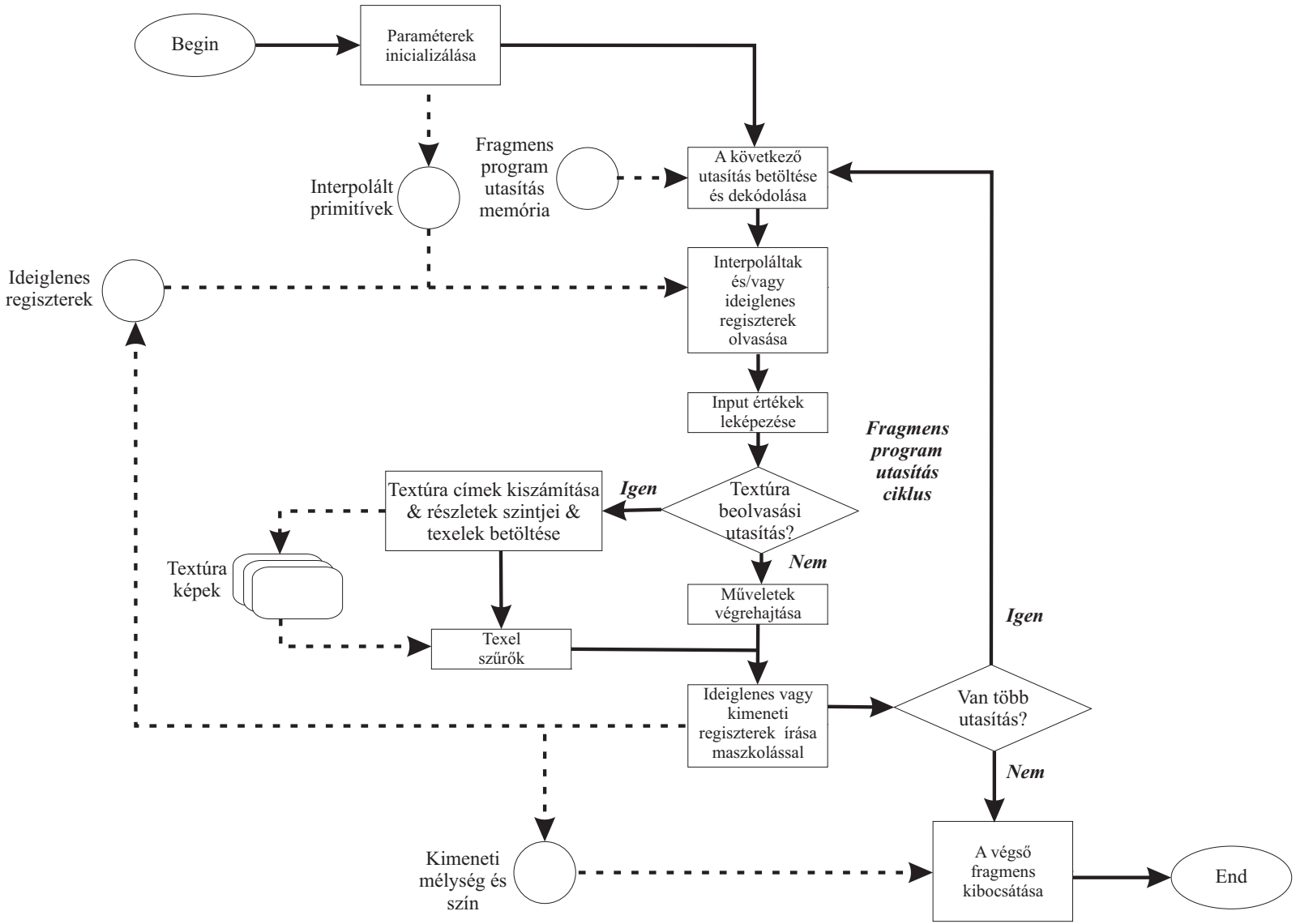
Programozható grafikus csővezeték



Programozható grafikus csővezeték

- Programozható vertex processzor
  - Vertex attribútumok betöltése a megfelelő regiszterekbe
  - Vektorokon végzett matematikai műveletek
  - Fejlettebb vertex processzorok a vezérlési szerkezeteket is támogatják
- Programozható fragmens processzor
  - Hasonló műveletek végrehajtása, mint a vertex processzorok esetén
  - A fragmens processzorok támogatják a textúra műveleteket is

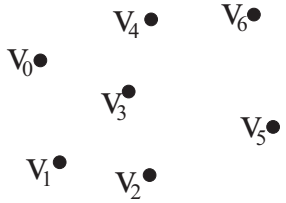

Begin
Vertex atribútumok másolása a bemeneti regiszterekbe
Vertex program utasítás memória
A következő utasítás betöltése és dekódolása
Bemeneti regiszterek
A bemeneti és/vagy ideiglenes regiszterek olvasása
Input értékek leképezése
Vertex program utasítás ciklus
Ideiglenes regiszterek
Műveletek végrehajtása
Igen
Ideiglenes vagy kimeneti regiszterek írása maszkolással
Van több utasítás?
Nem
Kimeneti regiszterek
A kimeneti regiszter transzformált vertexként való kibocsátása
End


Begin
Paraméterek inicializálása
Interpolált primitívek
Fragmens program utasítás memória
A következő utasítás betöltése és dekódolása
Ideiglenes regiszterek
Interpoláltak és/vagy ideiglenes regiszterek olvasása
Input értékek leképezése
Fragmens program utasítás ciklus
Textúra címek kiszámítása & részletek szintjei & texelek betöltése
Igen
Textúra beolvasási utasítás?
Textúra képek
Nem
Műveletek végrehajtása
Texel szűrők
Igen
Ideiglenes vagy kimeneti regiszterek írása maszkolással
Van több utasítás?
Nem
Kimeneti mélység és szín
A végső fragmens kibocsátása
End

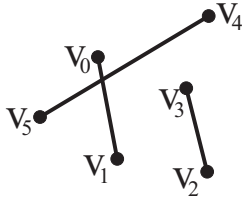
OpenGL függvénykönyvtár

OpenGL függvénykönyvtár

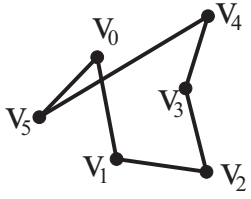
- Hordozható, 3D-s grafikus függvénykönyvtár
- Több száz függvényt és definíciót tartalmaz
- Egy színtér leírásához OpenGL függvény hívások sorozatát kell megadni
- Vertex
  - OpenGL primitívek szögpontjai
  - 2D-s és 3D-s pozíciók
  - Meghatározzák a primitív alakját és helyzetét



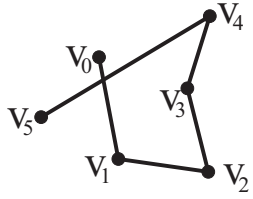
Pontok



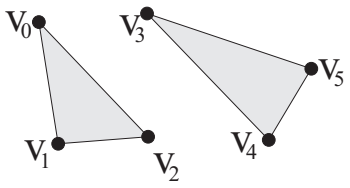
Vonalak



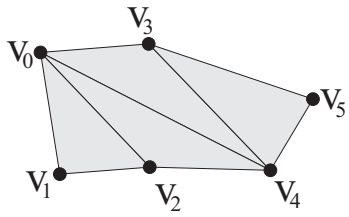
Vonal hurok



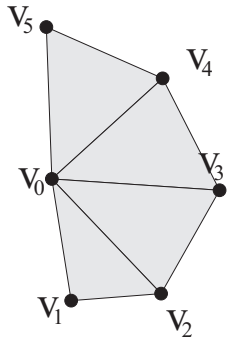
Töredezett vonal



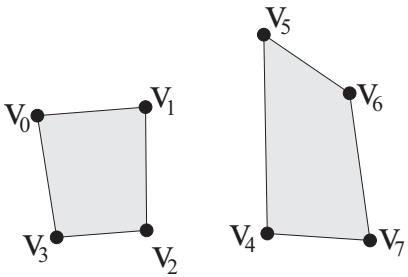
Háromszögek



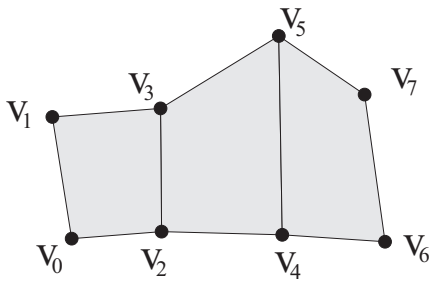
Háromszögsáv



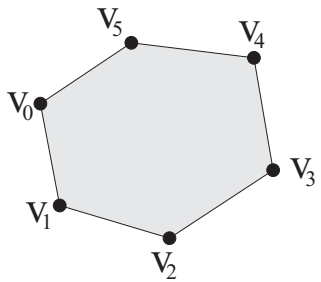
Háromszög-legyező



Négyszögek



Négyszögsáv



Poligon

OpenGL függvénykönyvtár

- Az OpenGL függvénykönyvtár támogatja a
  - Megvilágítást
  - Árnyalást
  - Textúrázást
  - Keveredést
  - Átlátszóságot
  - Más speciális hatásokat és képességeket
- Az OpenGL függvénykönyvtár nem tartalmaz
  - Ablakkezelő függvényeket
  - Felhasználói interaktivitást és I/O műveleteket végrehajtó függvényeket
- Nincs OpenGL file formátum
  - A modellek tárolására
  - A virtuális színtér tárolására

OpenGL adattípus	Belső reprezentáció	C adattípusként definiálva
`GLbyte`	8 bites egész	`signed char`
`GLshort`	16 bites egész	`short`
`GLint, GLsizei`	32 bites egész	`long`
`GLfloat`	32 bites lebegőpontos	`float`
`GLclampf`	pont	
`GLuint, GLenum, GLbitfield`	32 bites előjel nélküli egész	`unsigned long`

<Könyvtár prefix><Alap parancs><Opcionális argumentum szám><Opcionális argumentum típus>

```
gl Color 3 f
```

```
glColor3f(0.5, 0.5, 0.5);
```

OpenGL függvénykönyvtár

- Platformfüggetlenség
  - Operációs rendszerekhez kapcsolódó feladatok
    - Ablakkezelés
    - Felhasználói interakciók kezelése
    - Felhasználó leütötte-e az Enter billentyűt?
- GLUT használata
  - Kezdetekben AUX (auxiliary) lib
  - Kereszt-platformos példák szemléltetése
  - Pop-up menük, ablakok, joystick támogatás, stb.
  - GUI programozása adott platformon

```
#include <GL/glut.h>

// a színtér rajzolása
void RenderScene(void)
{
// Az aktuális törlő színnel való ablak törlés
   glClear(GL_COLOR_BUFFER_BIT);

// Flush rajzoló parancs
   glFlush();
}

// A renderelési állapotok beállítása
void SetupRC(void)
{
//A színpuffer törlőszínének a beállítása
   glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}
```

```
// A program belépési pontja
int main(int argc, char* argv[])
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
   glutCreateWindow("Egyszeru");
   glutDisplayFunc(RenderScene);
   SetupRC();
   glutMainLoop();
   return 0;
}
```

```
glutInit(&argc, argv);
```

- Továbbítja a parancssori paramétereket
- Inicializálja a GLUT függvénykönyvtárat

```
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
```

- Egyszeresen pufferelt ablak
- RGBA színmód

```
glutCreateWindow("Egyszeru");
```

- Ablak létrehozása
- Címsorában az "Egyszeru" felirat

```
glutDisplayFunc(RenderScene);
```

- RenderScene callback függvény regisztrálása
- Ablak újrarajzolása
  - Ablak első megjelenítésekor
  - Ablak előtérbe helyezésekor
- OpenGL renderelési függvények hívása

```
SetupRC();
```

- Renderelése előtti inicializálás
- OpenGL állapotok beállítása

```
glutMainLoop();
```

- GLUT eseménykezelő elindítása
- Vezérlés átadása a GLUT-nak
- Fő ablak bezárásig nem tér vissza
- Üzenetek feldolgozása

```
glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
```

- Ablak törlésére használt szín megadása
- A színpuffer inicializálására használt szín beállítása

```
void glClearColor(GLclampf red, GLclampf
    green, GLclampf blue, GLclampf alpha);
```

- `GLclampf`: 0 és 1 közé leképezett `float`
- A szín vörös, zöld és kék összetevők keverékeként való megadása
- `alpha`: keveredés és speciális hatások

```
glClear(GL_COLOR_BUFFER_BIT);
```

- A színpuffer törlése
- Pufferek törlése

```
glFlush();
```

- OpenGL parancssor ürítése
- Nem vár további utasításokra
- Beérkezett utasítások feldolgozásának folytatása

Összefoglalás

Grafikus csővezeték

- Műveletek meghatározott sorrendje
- Adatok áramlása egyik fázisból a másikba
  - Meghatározott típusú be- és kimenő adatok
  - SIMD
- Programozható grafikus csővezeték
  - Rögzített műveleti sorrendű grafikus csővezeték „kiegészítése”
  - Programozható vertex és fragmens processzor

Összefoglalás

OpenGL függvénykönyvtár

- 3D-s grafikus függvénykönyvtár
- Színtér leírása függvények meghívásával
- Rögzített műveleti sorrendű grafikus csővezeték
- GLUT
- Első program