GPU, Optimizavimas, Efektai

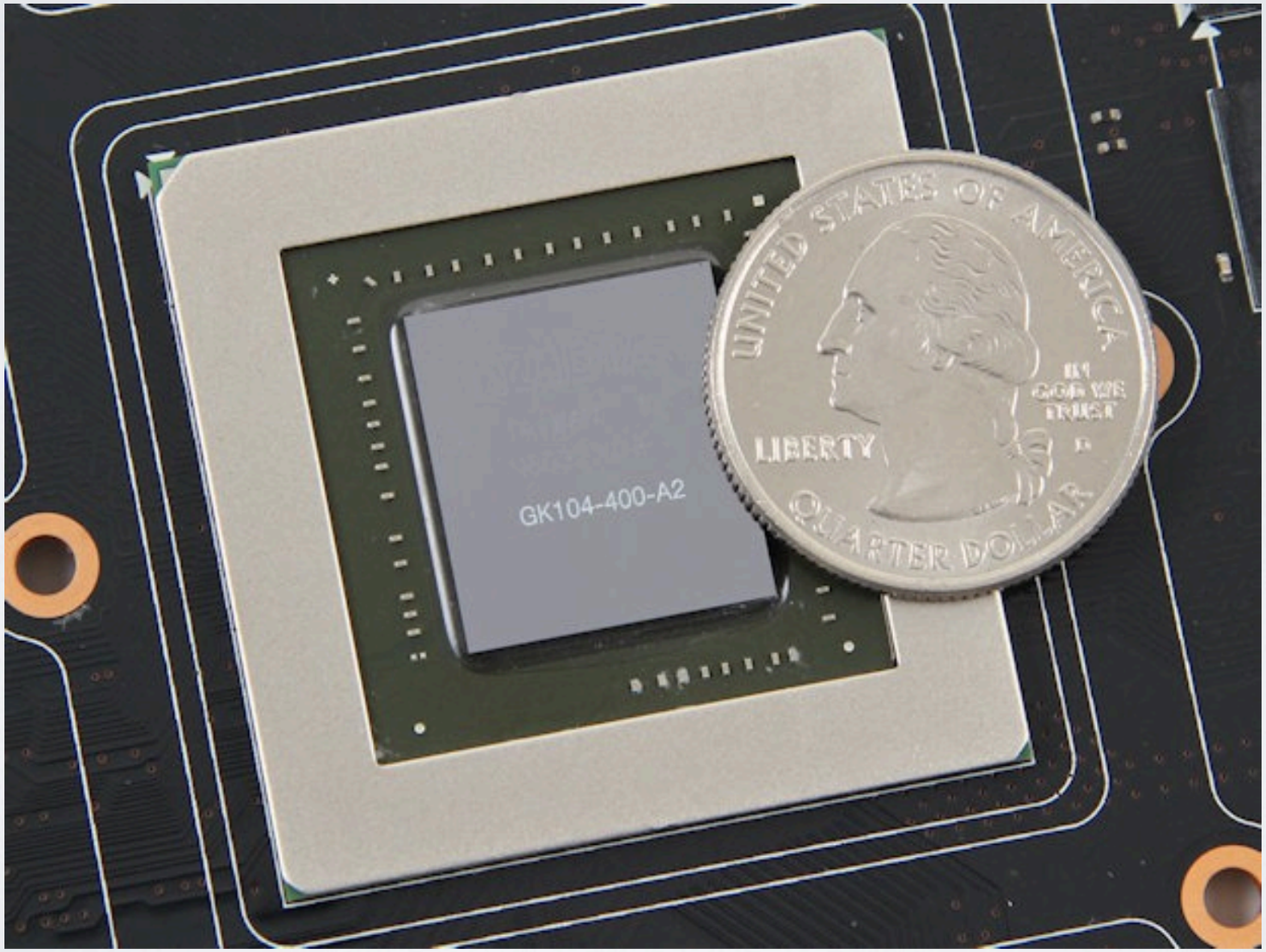
T120B167 Trimatės grafikos efektų programavimas

Aras Pranckevičius
Unity / nesnausk! / @aras_p

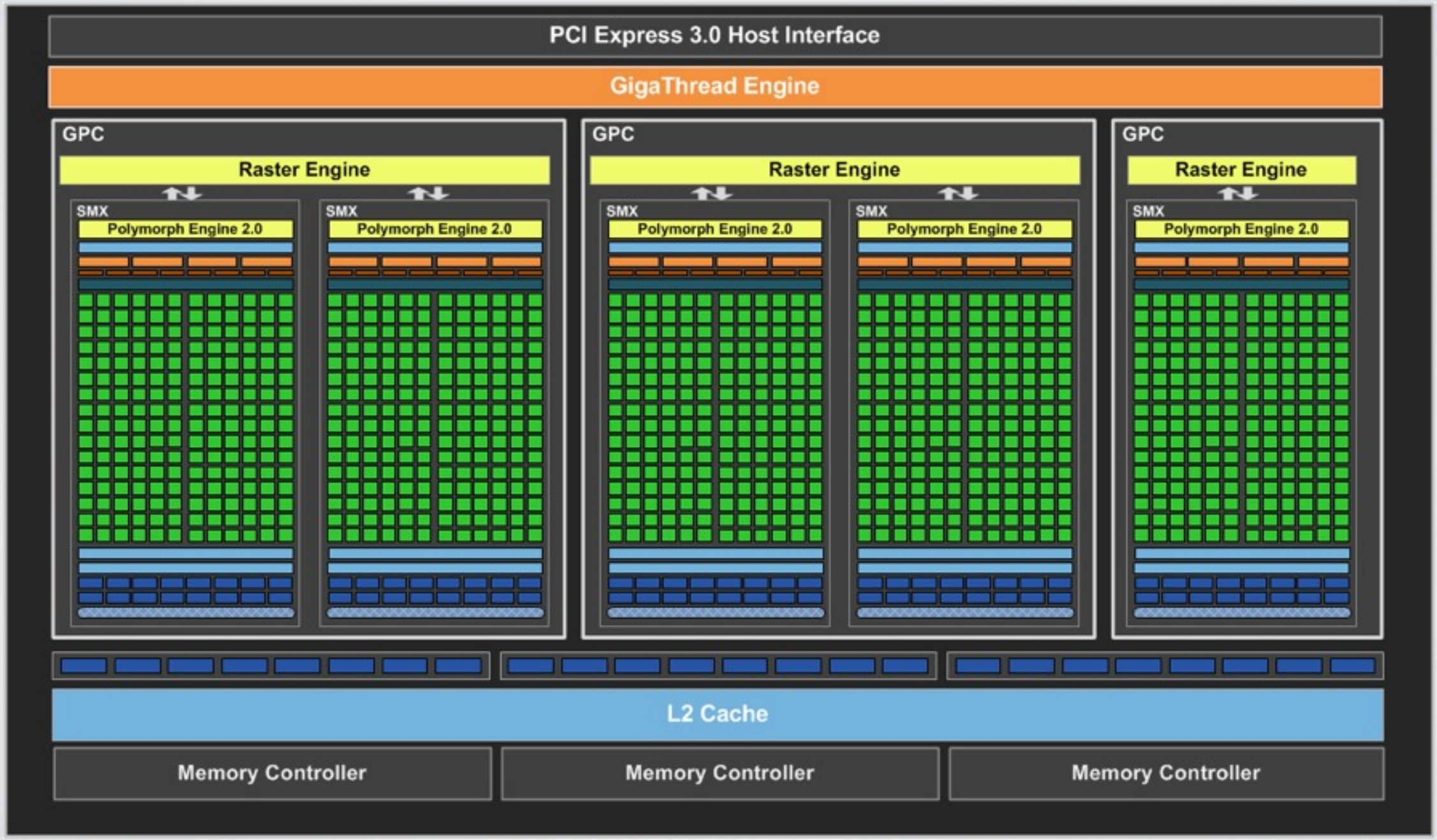
GPU

GPU, Geforce 680

Geforce 680 lustas

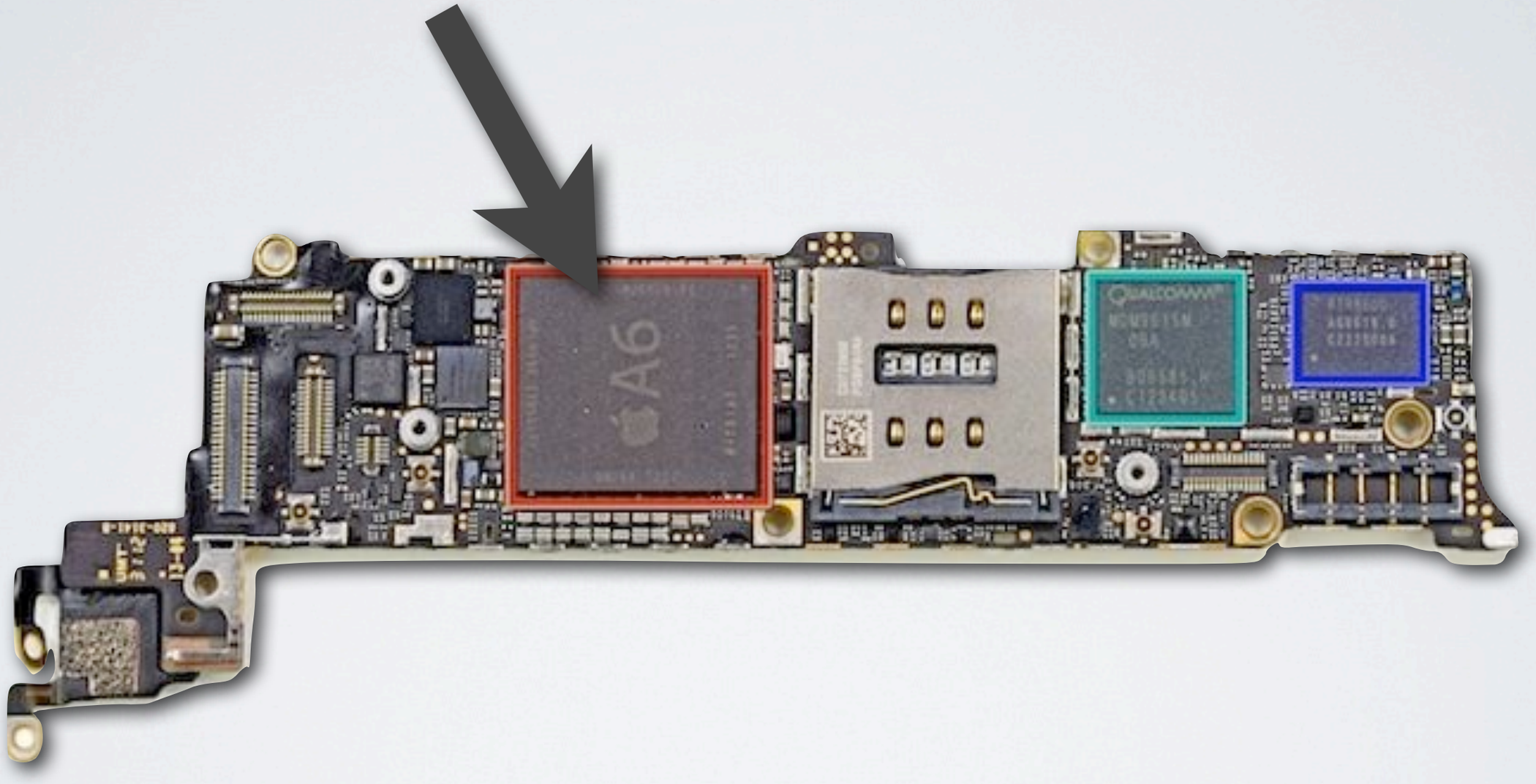


Lusto diagrama

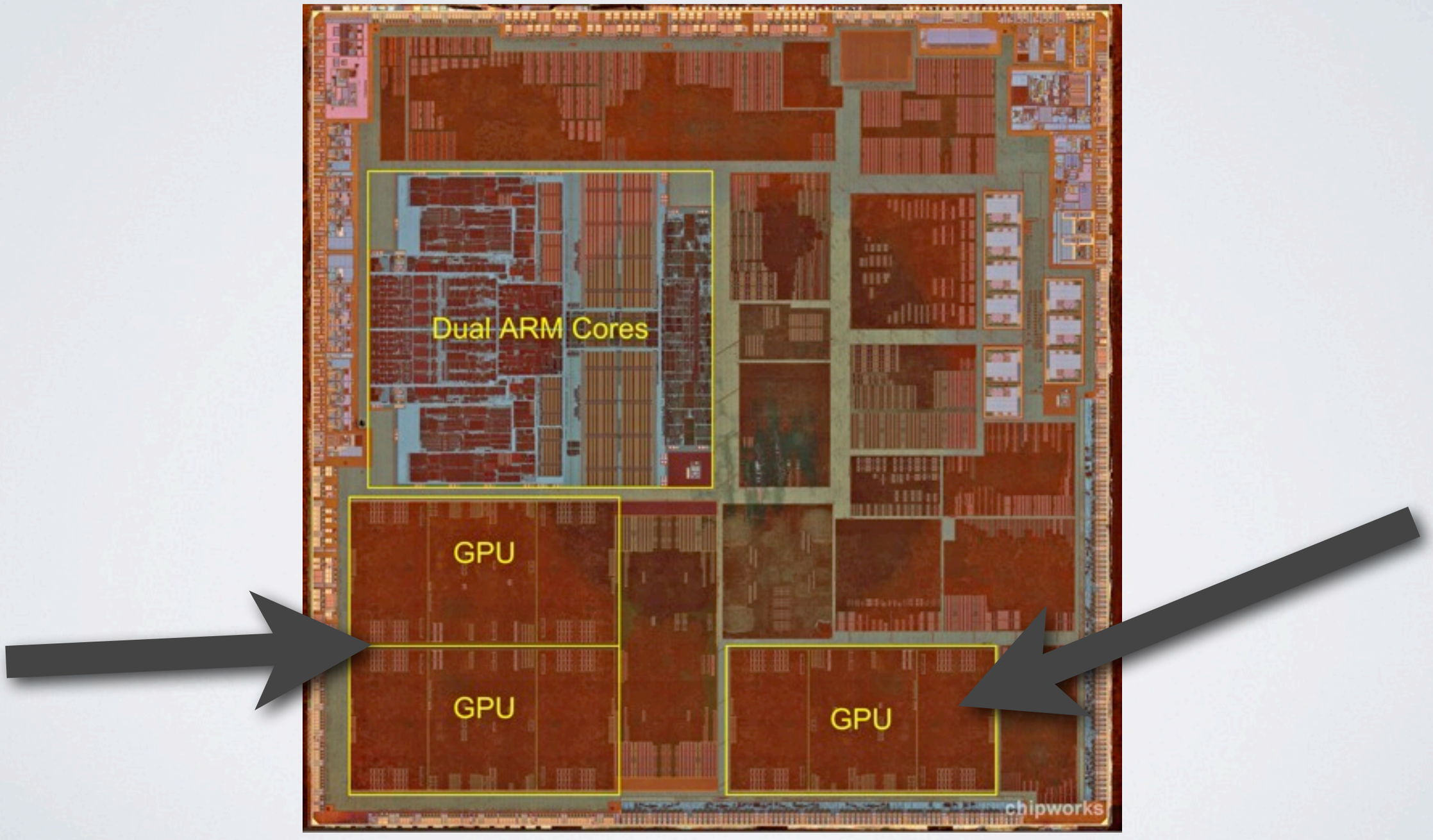


iPhone 5

Jame A6 SoC



O jame SGX543 MP3 GPU



Skirtingi GPU!

- GeForce 680:
  - ~200W vien tik GPU
  - Didelis, karštas, triukšmingas
- A6 su SGX 543:
  - ~1W CPU+GPU
  - Tačiau ir *žymiai* lėtesnis!

Kaip ir kodėl GPU veikia?

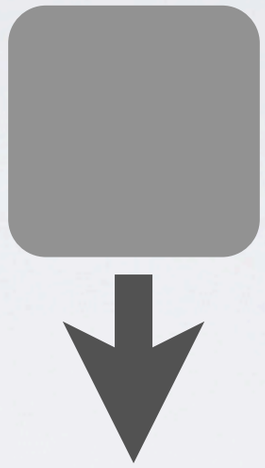
- Heterogeniškas
- Daug “procesoriukų”
- Padarytas taip, kad grafikos uždaviniai veiktų greitai

Nuo čia prasideda labai jau akivaizdus Kayvon Fatahalian kalbos “Running code at terafrop: how a GPU shader core works” rip’as, tik aš paveiksliukus pats perpaišiau. Originali kalba, kuri yra visai kaip awesome, yra čia: http://bps10.idav.ucdavis.edu/

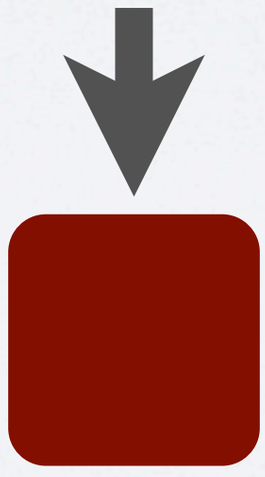
Šeideris

```
vec3 c = texture2D(smp, uv);
float d = clamp(dot(ldir,norm), 0.0, 1.0);
return c * d;
```

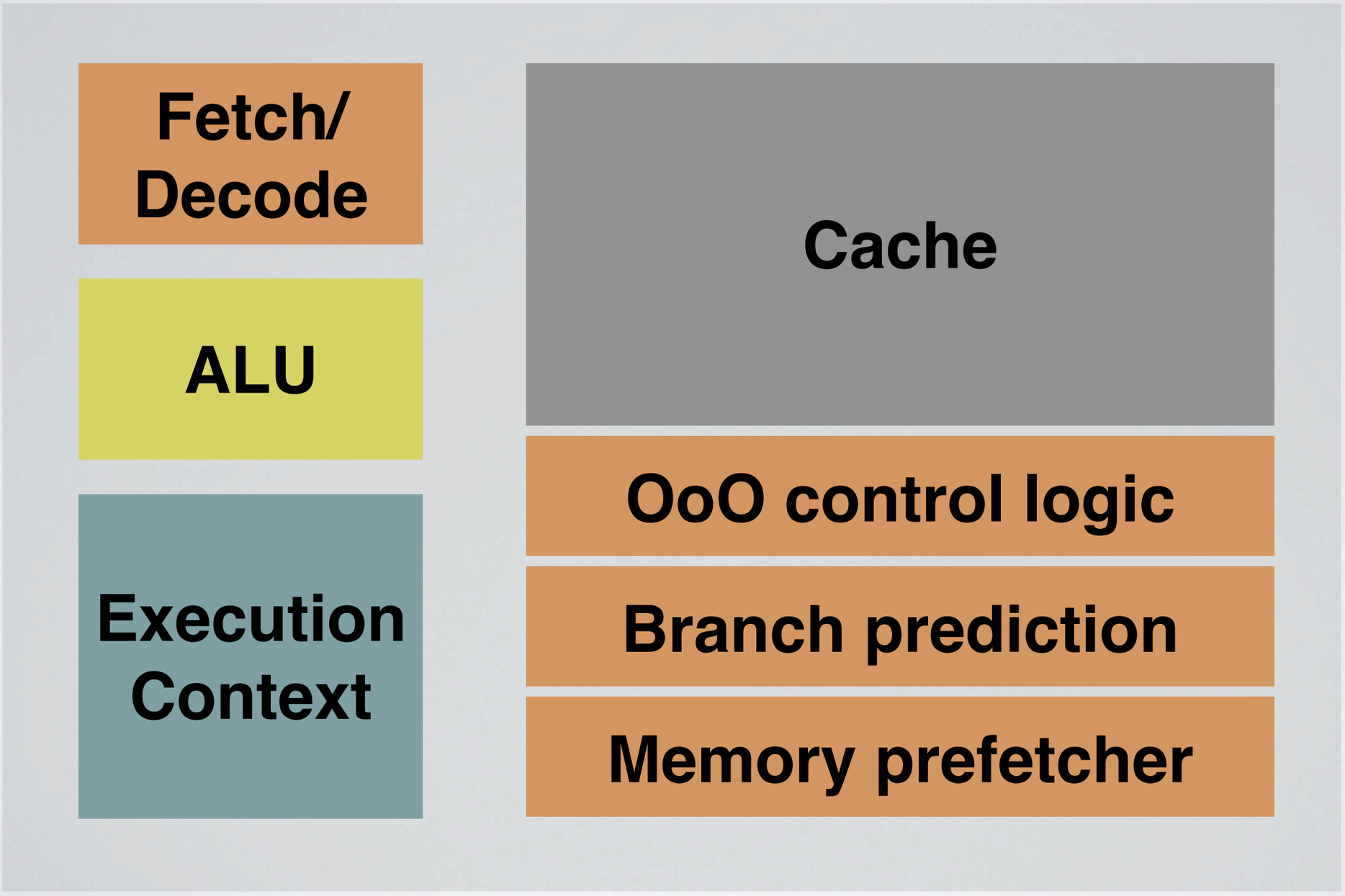
- Esminis dalykas:
- Visi pikseliai/viršūnės/... apdorojami **nepriklausomai**!



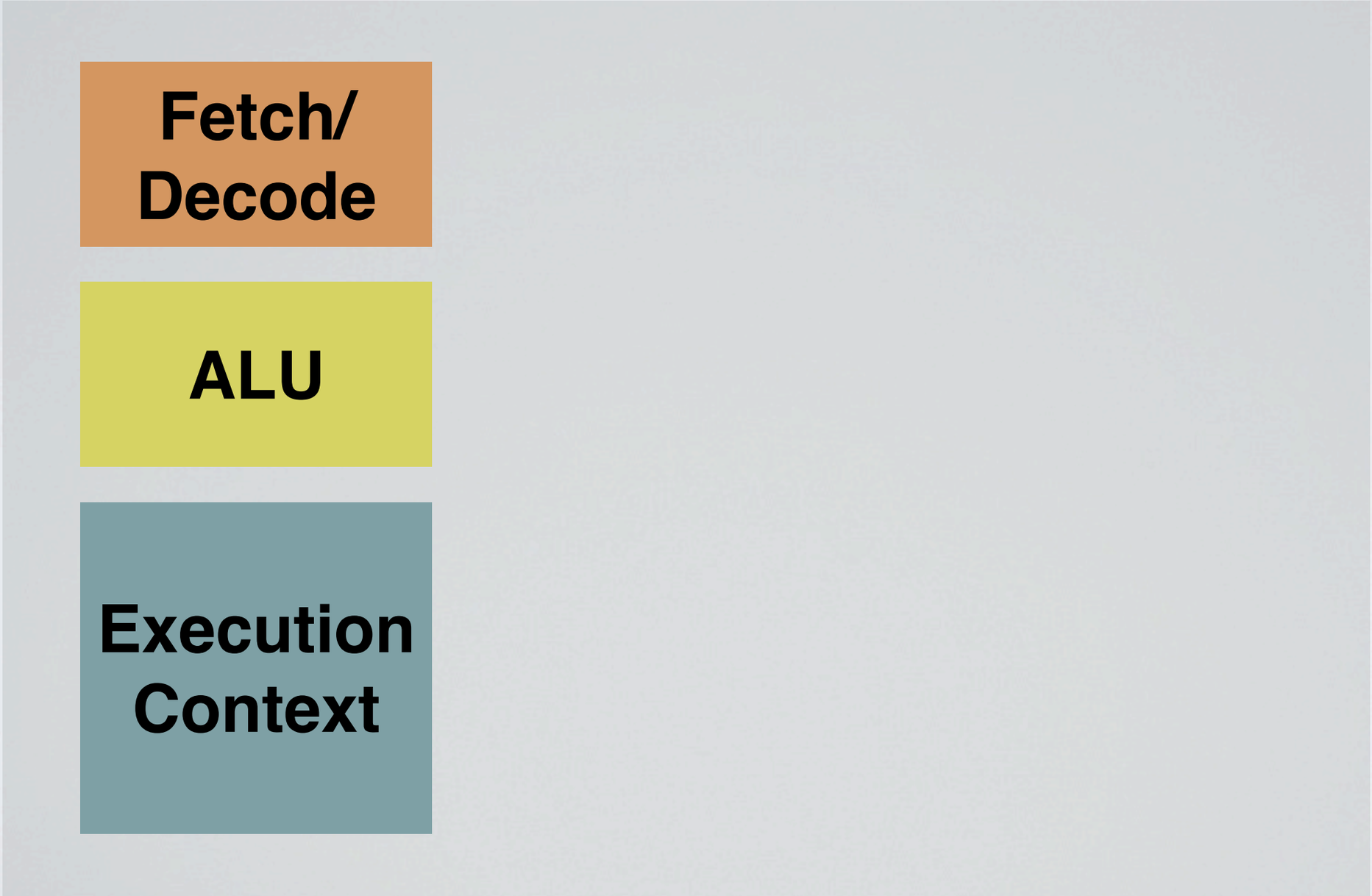
```
vec3 c = texture2D(smp, uv);
float d = clamp(dot(ldir,norm), 0.0, 1.0);
return c * d;
```



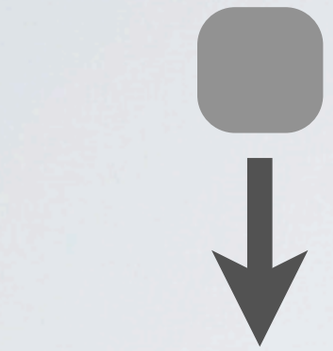
CPU


Fetch/
Decode
ALU
Execution
Context
Cache
OoO control logic
Branch prediction
Memory prefetcher

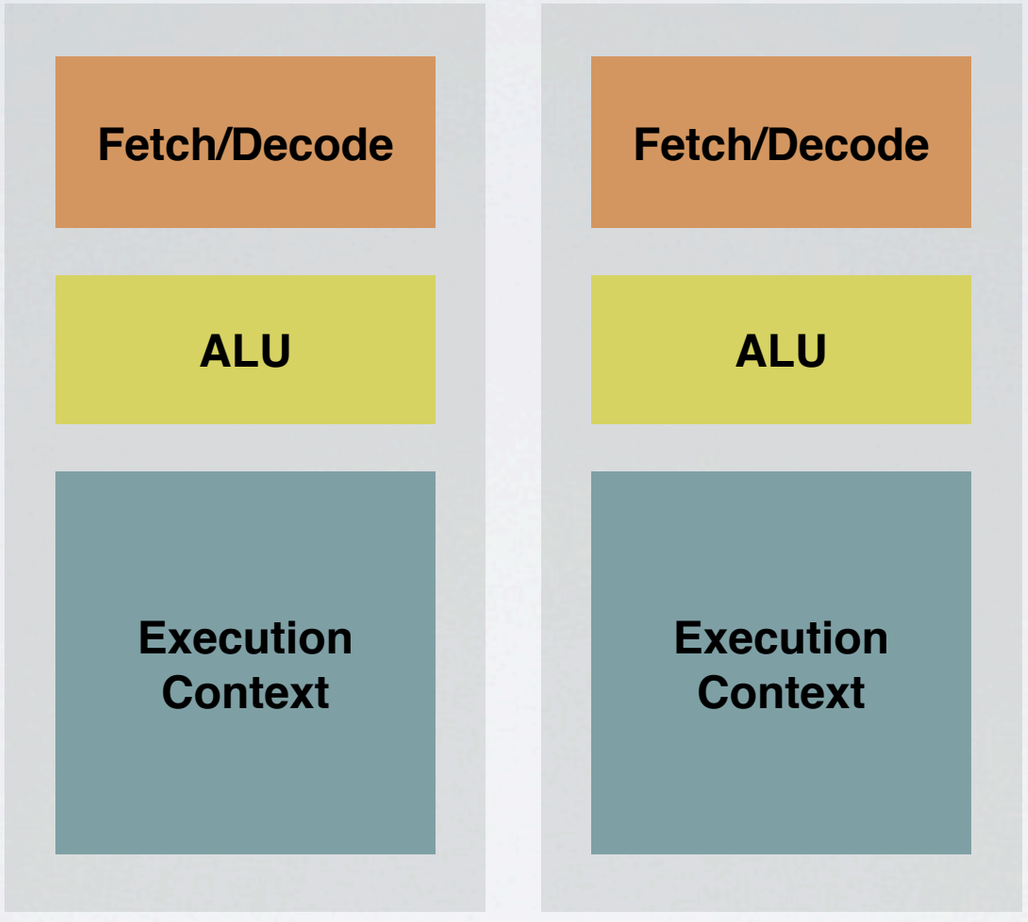
#1: Išmeskim ko “nereikia”



Du branduoliai

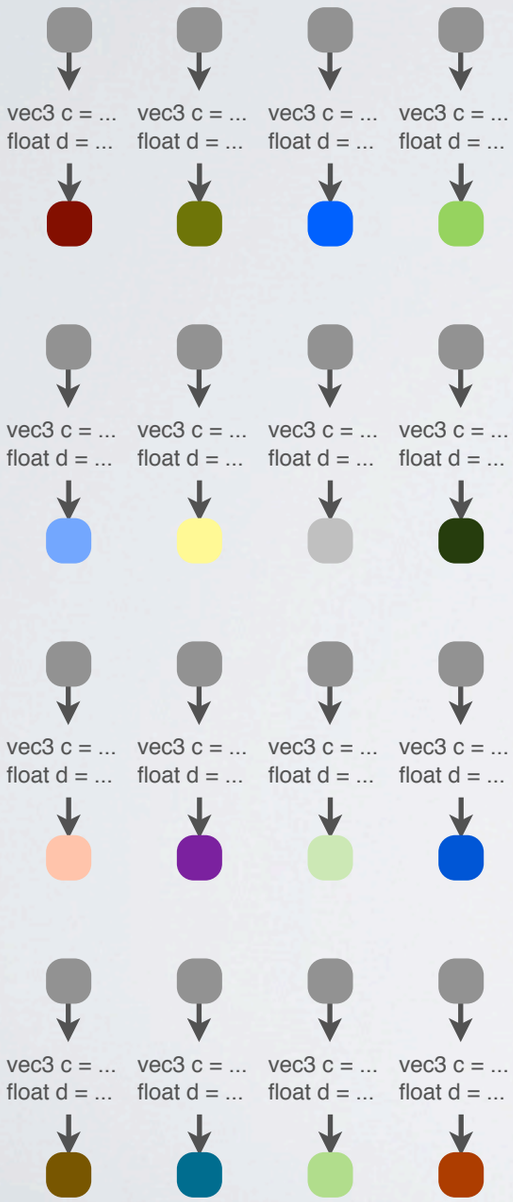


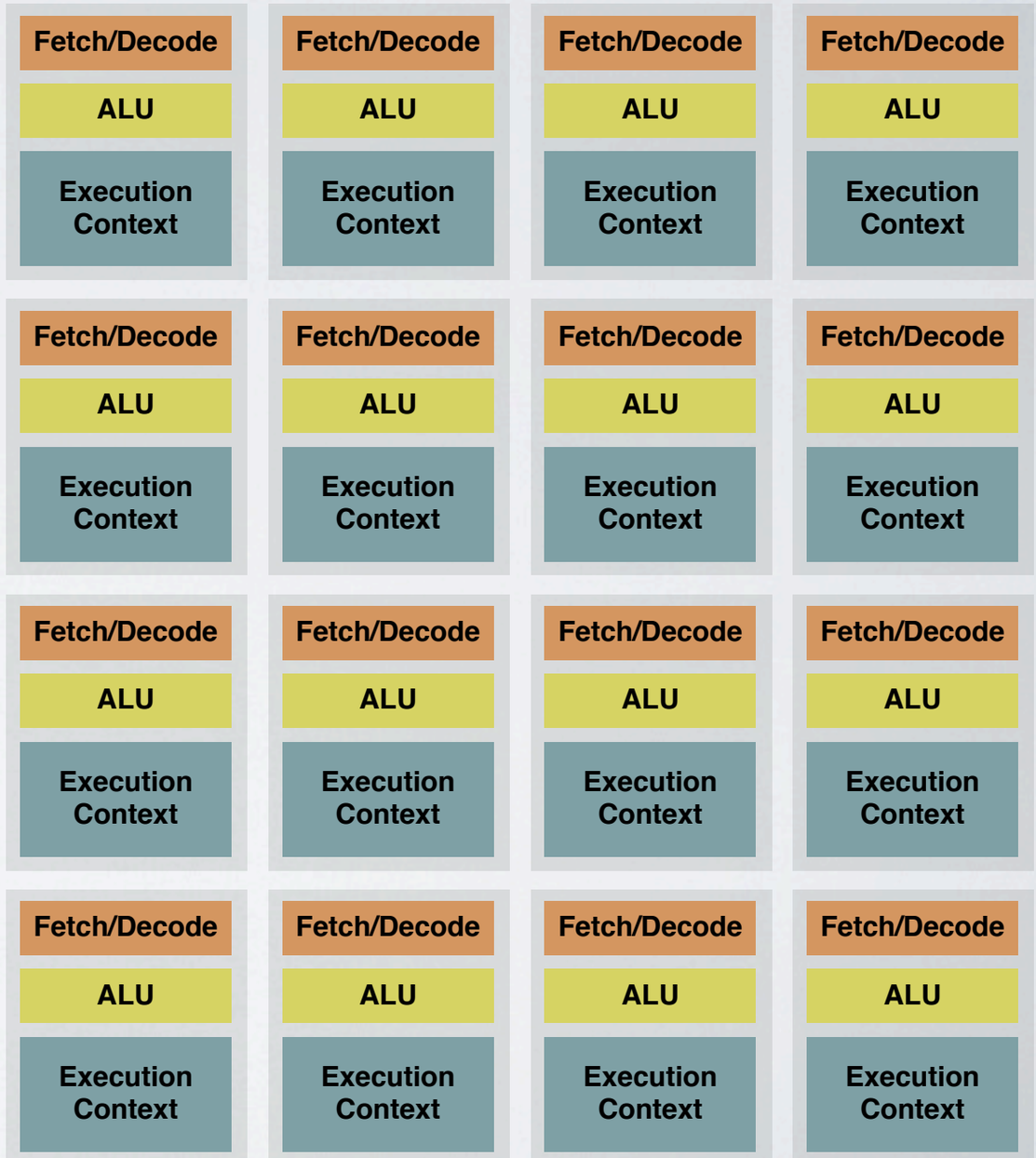
```
vec3 c = texture2D(smp, uv);
float d = clamp(dot(ldir,norm), 0.0, 1.0);
return c * d;
```



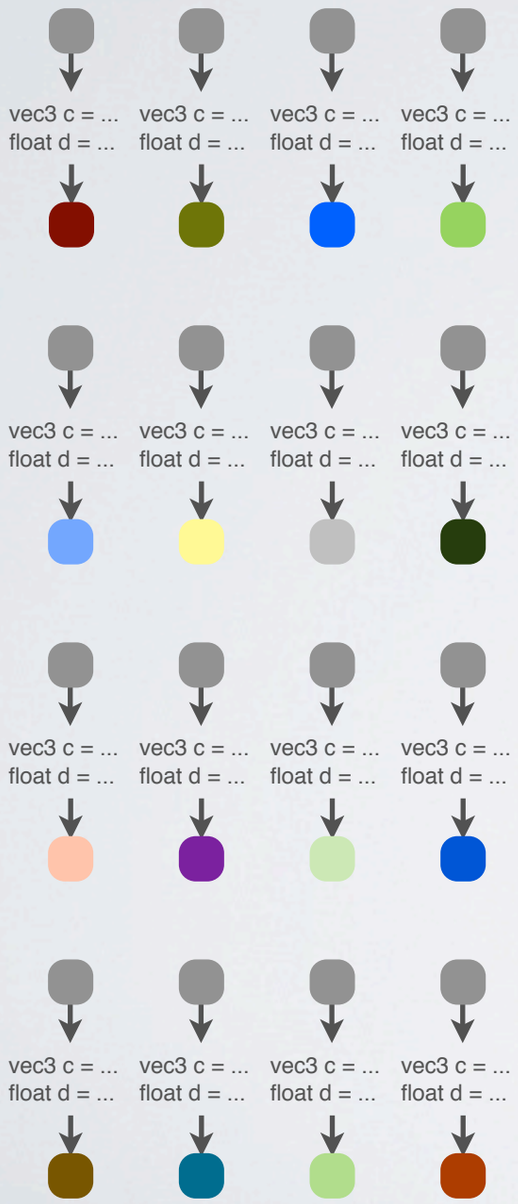
```
vec3 c = texture2D(smp, uv);
float d = clamp(dot(ldir,norm), 0.0, 1.0);
return c * d;
```

16 branduolių



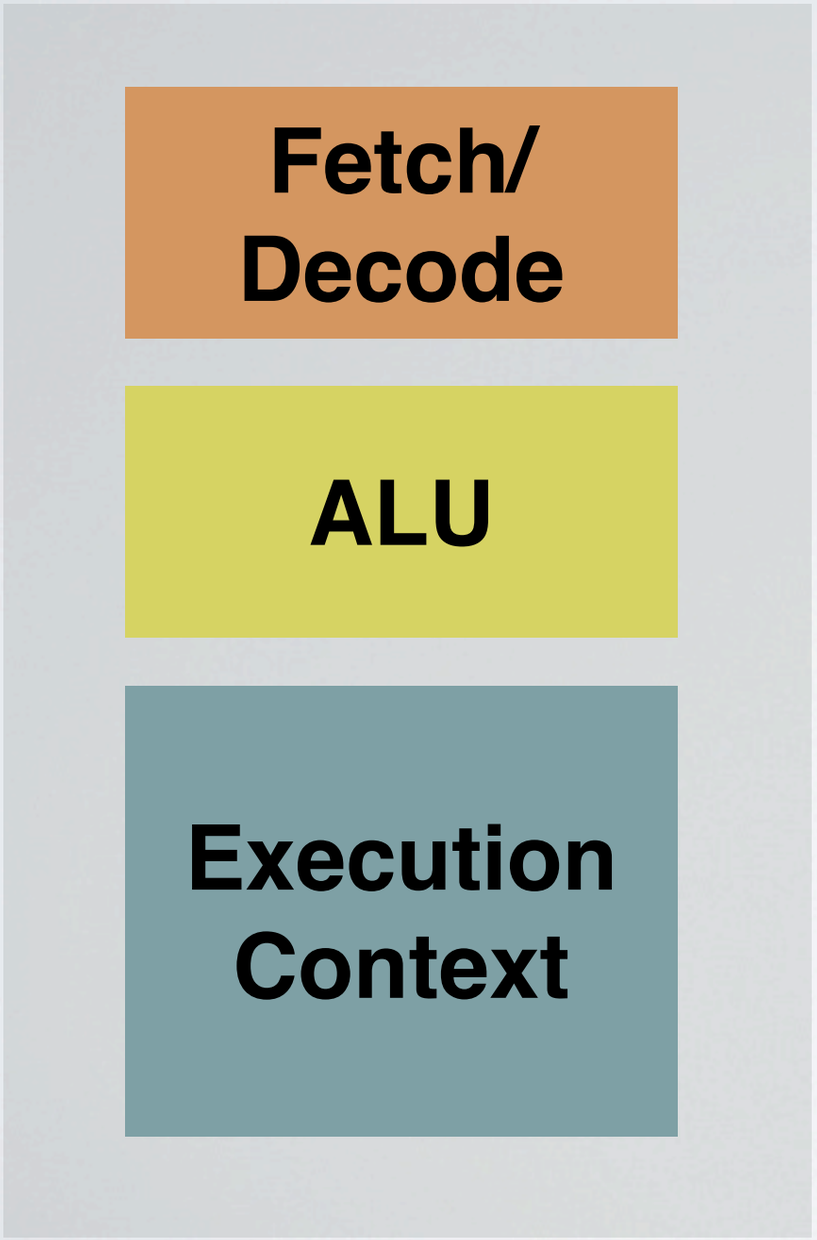


Visi vykdo tą patį kodą!



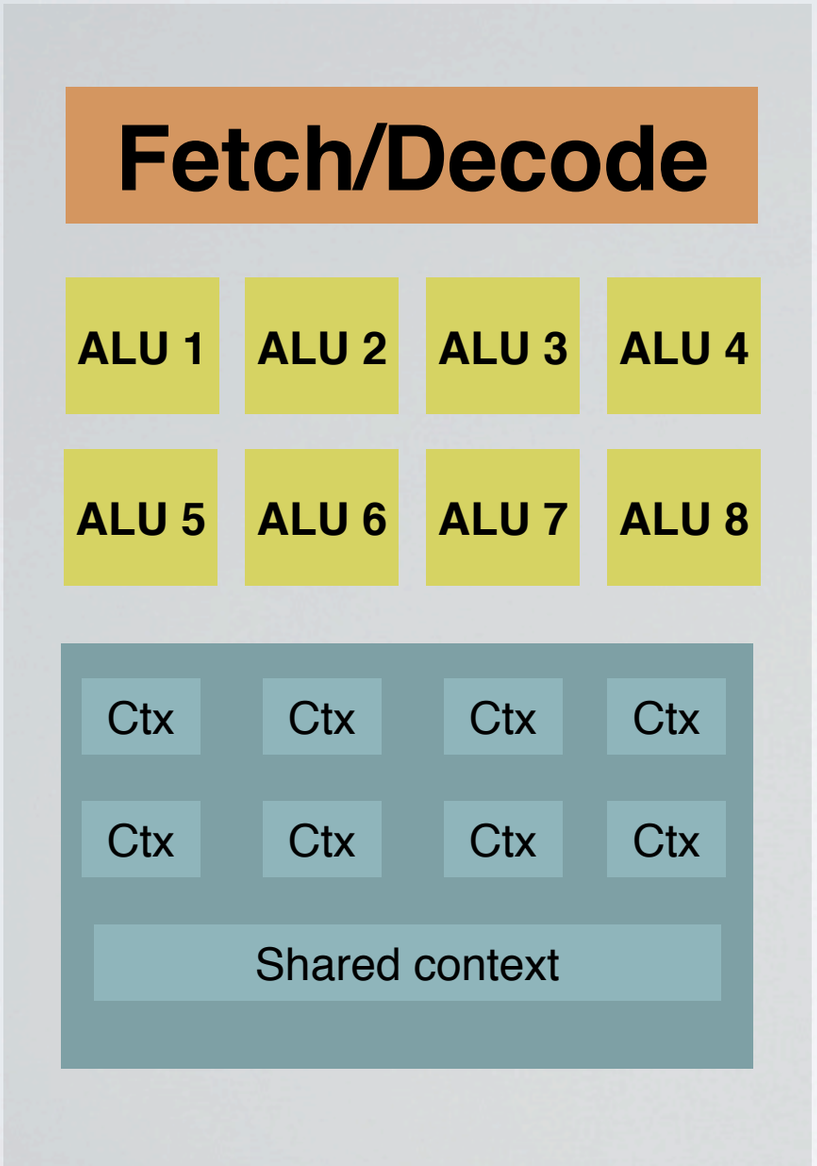
- Taigi visi vykdo lygiai tokį pat šeiderį
- Galima ką nors sutaupyt

#2: SIMD



- Kol kas naudotas paprastas branduolys

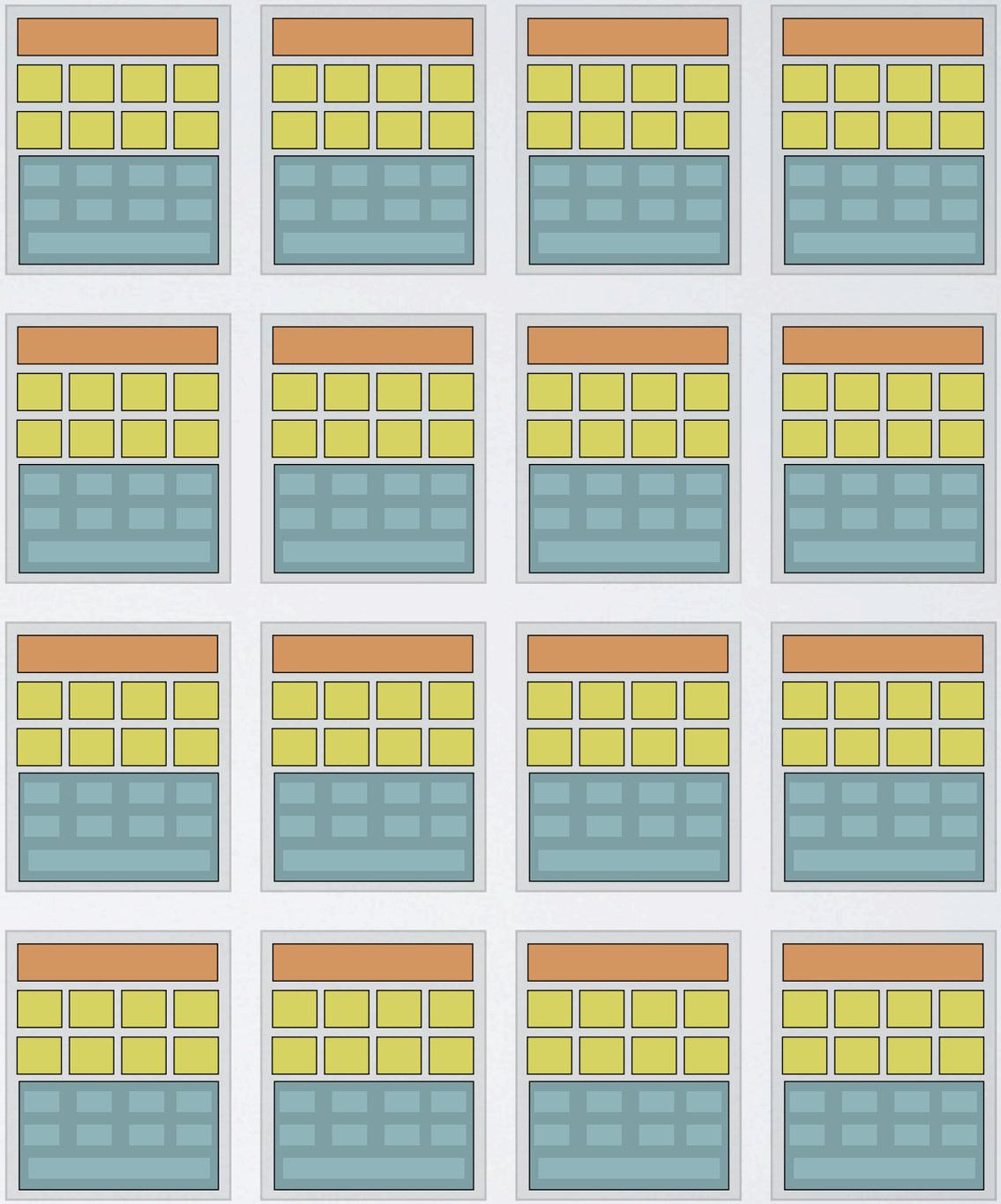
#2: SIMD



- Pridėkim 8x daugiau ALU
- **SIMD**
- Pakeisti šeiderį, kad 8x visko darytų iš karto
- 8 pikselius apdorosim vienu ypu

MOAR CORES

- 8 pikseliai vienam branduoliui
- 16 branduolių
- **128 pikseliai vienu metu**
- Tik 16 vienu metu vykstančių šeiderių!



Kodėl “pikseliai”?

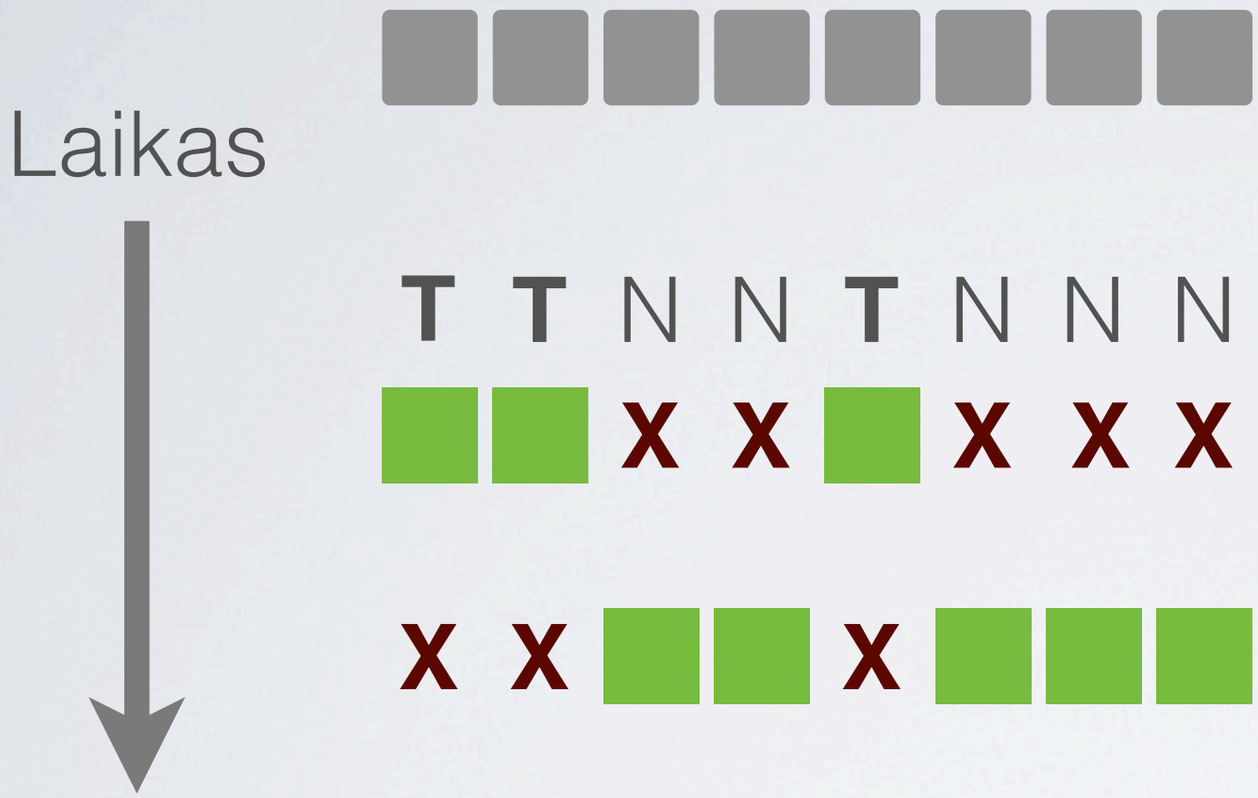
- Nebūtinai 128 pikseliai vienu metu...
- Viršūnės
- Pikseliai
- Briaunos
- OpenCL “work items”, DirectCompute/CUDA “threads”

Kaip su salygos sakiniais?

```
if (x > 0)
   y = pow (x, 32);
else
   y = 0;
```

- Vienas branduolys dabar apdoroja 8 daiktus iš karto.
- Kas bus jei (x>0) salyga bus skirtinga tarp jų?

Ne visi ALU ką nors daro




```
if (x > 0)
   y = pow (x, 32);
else
   y = 0;
```

- Blogiausiu atveju, 1/8 viso galimo našumo
- Vengti salygų, kurios skirtingos greta esančioms pikselių/viršūnių/... grupėms

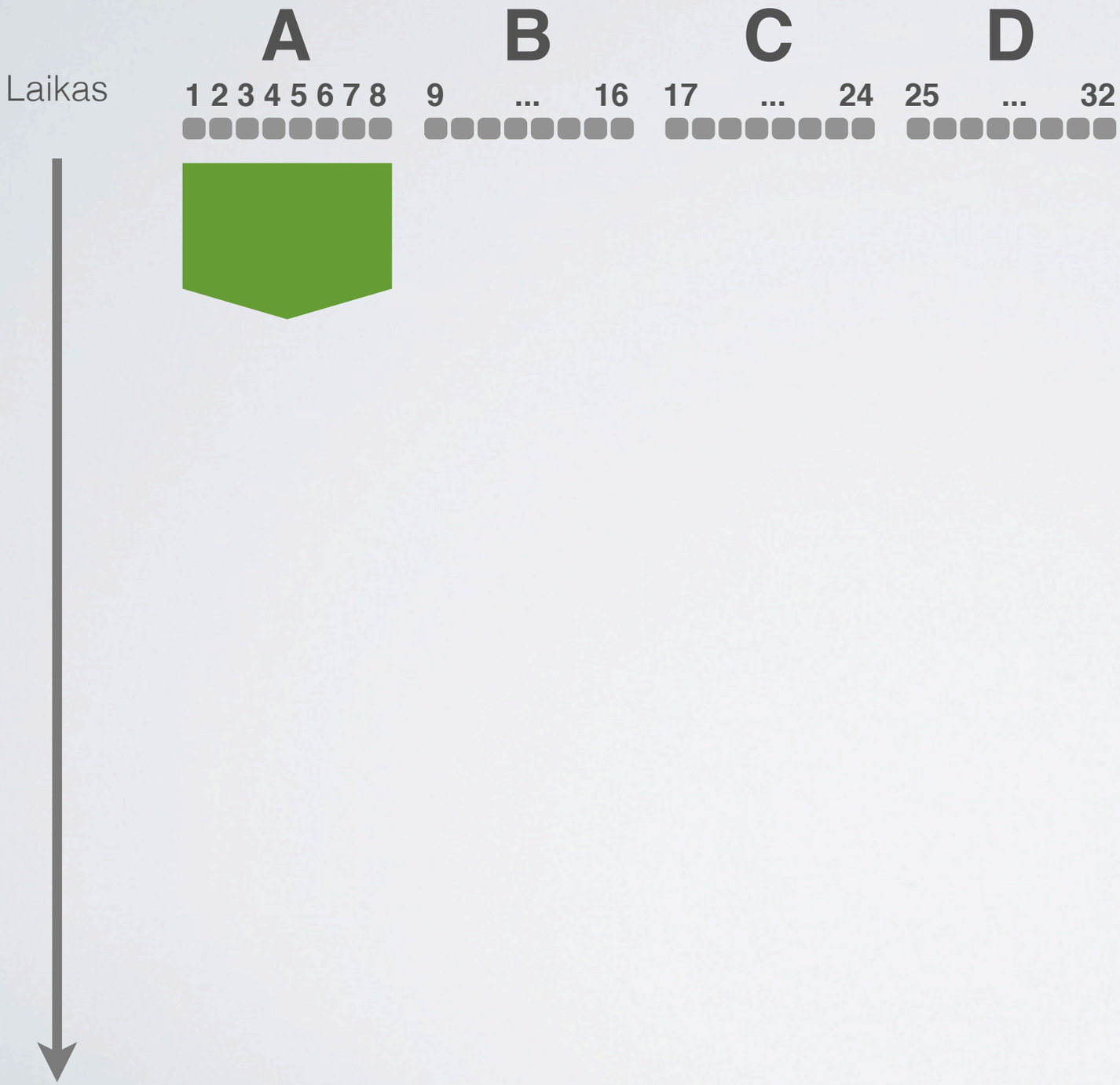
Bet gi mes išmetėm kešus?

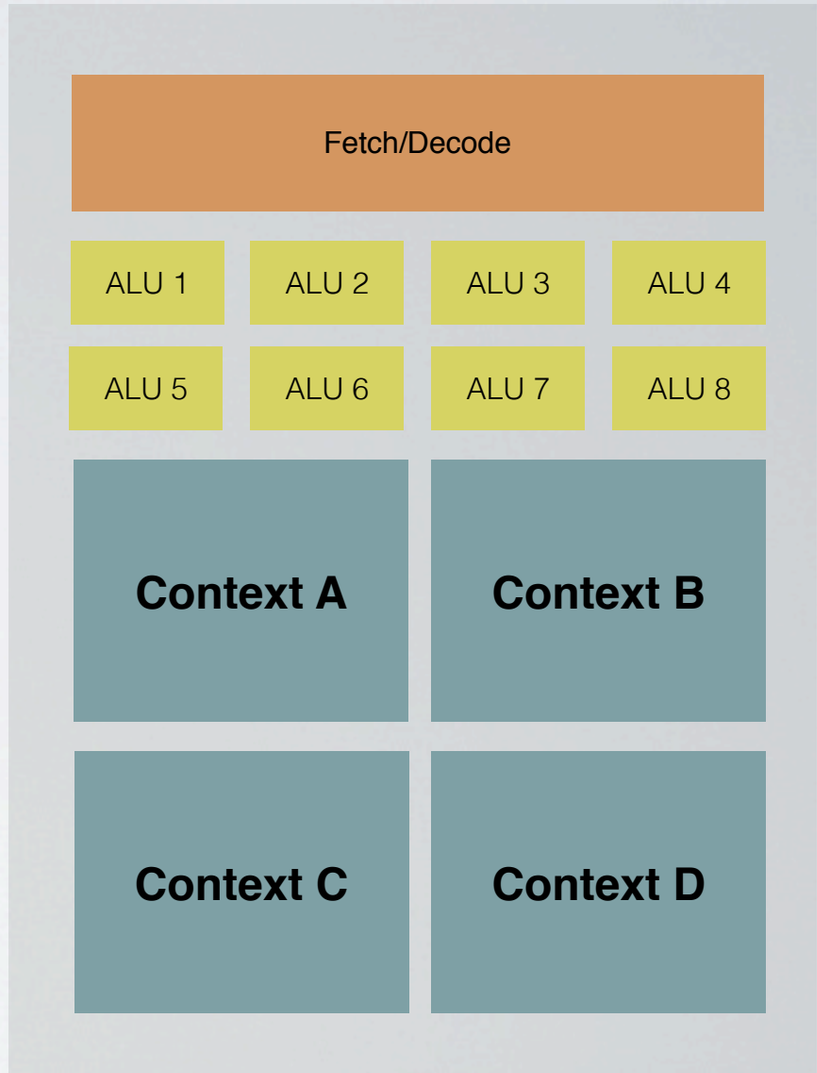
- texture2D(...) skaito iš atminties
- Bet didelius kešus mes jau išmetėm
- Skaitymas iš atminties gali užimti **tūkstančius** GPU taktų
- Ką veikti tuo metu?
  - GPU taigi neis *feisbuko čekintis!*

#3: skaičiuokim kitus pikselius

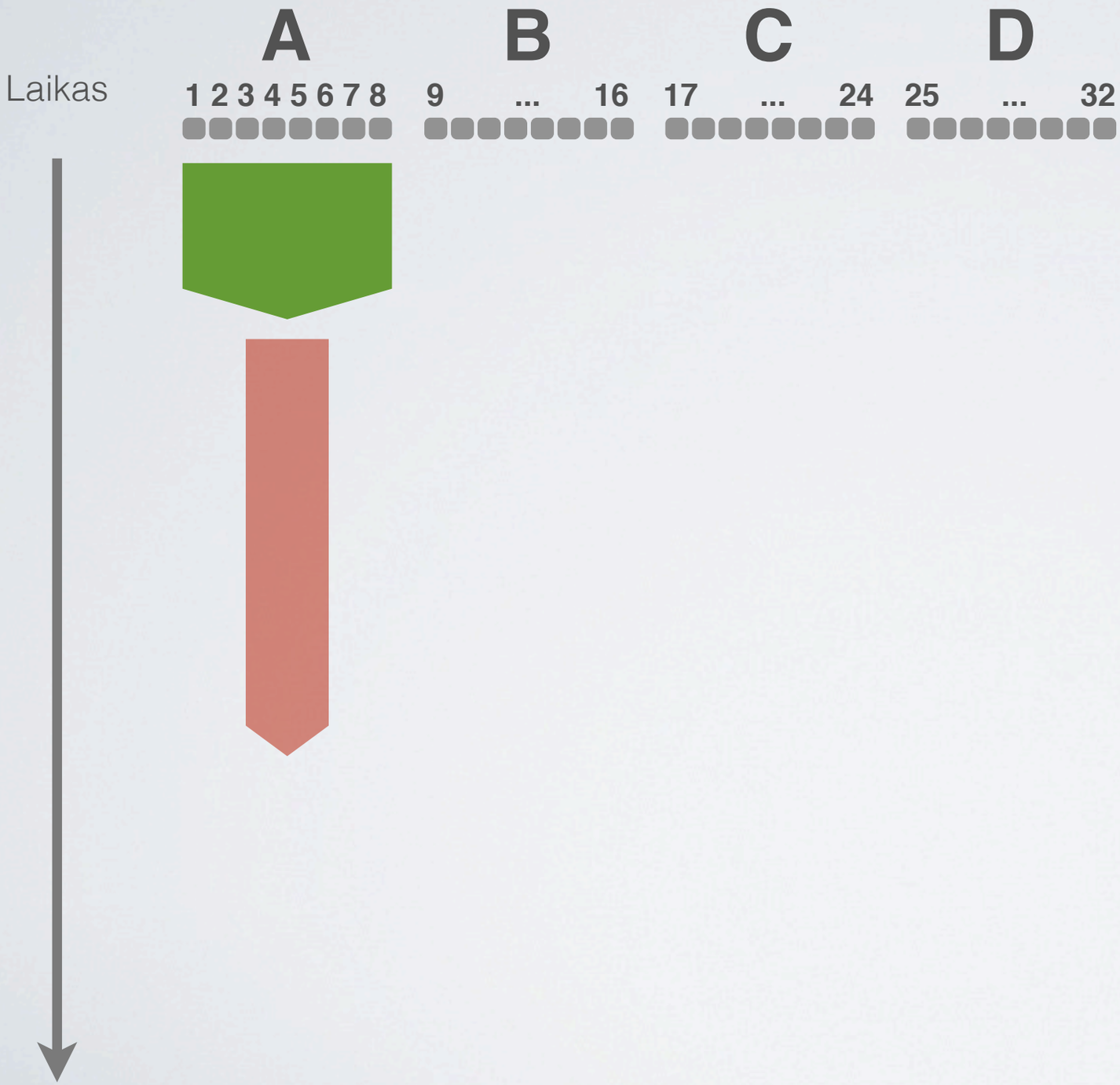
- Turime **daug** nepriklausomų pikselių
- Padarom texture2D ar panašiai,
- “persijungiam” prie kitų pikselių kuriam laikui
- Kai iš atminties rezultatai ateina, persijungiam atgal

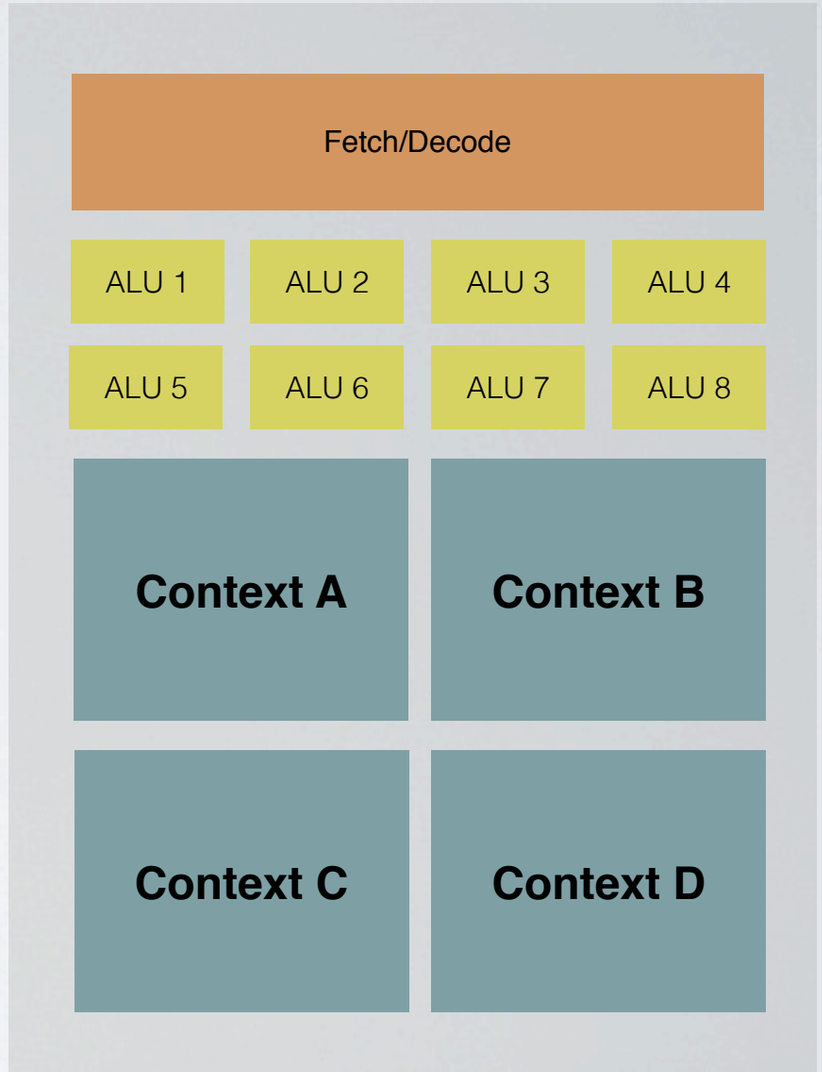
Kaip?


A
B
C
D
Laikas
1 2 3 4 5 6 7 8
9 ... 16
17 ... 24
25 ... 32

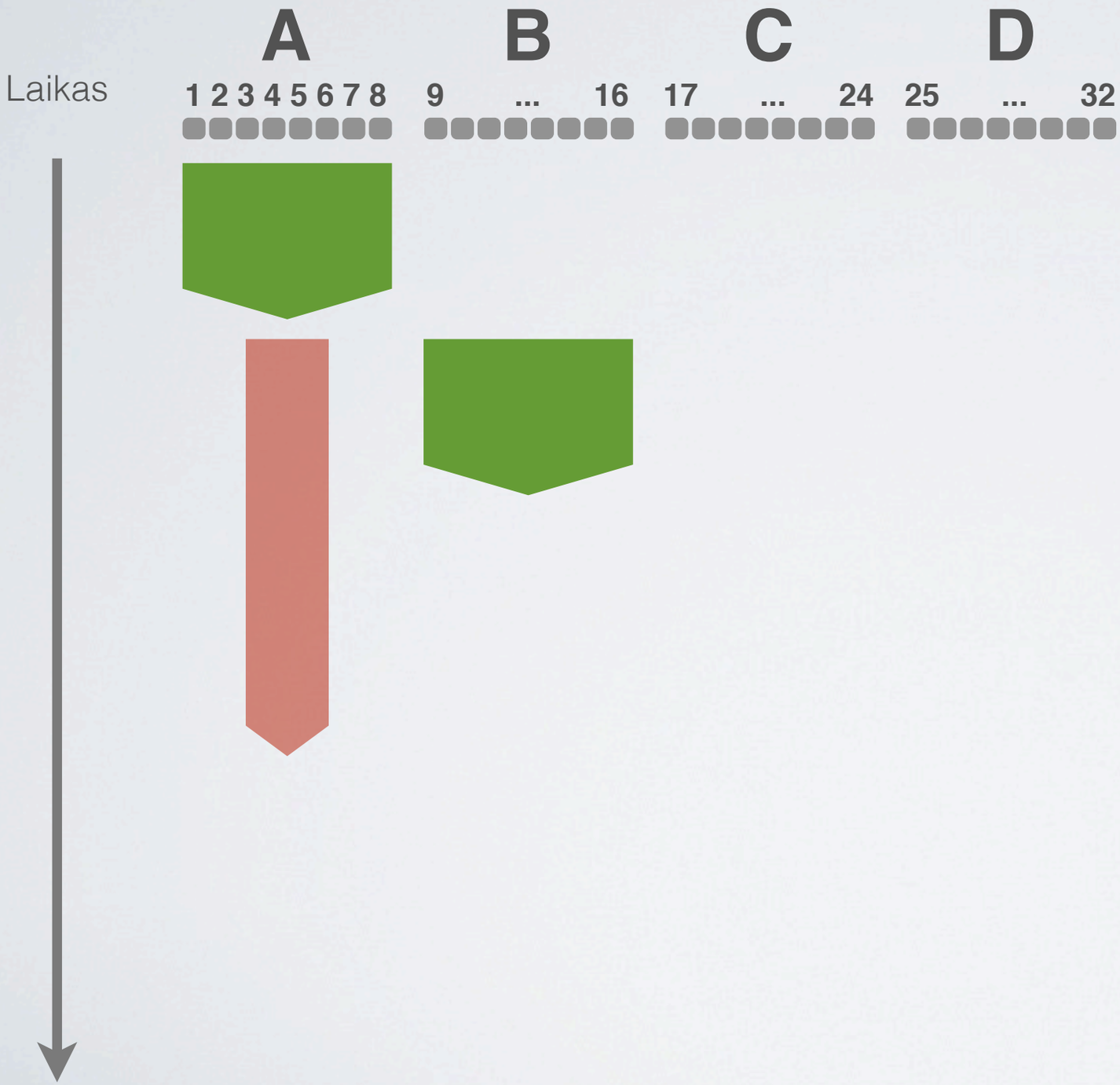

Fetch/Decode
ALU 1
ALU 2
ALU 3
ALU 4
ALU 5
ALU 6
ALU 7
ALU 8
Context A
Context B
Context C
Context D

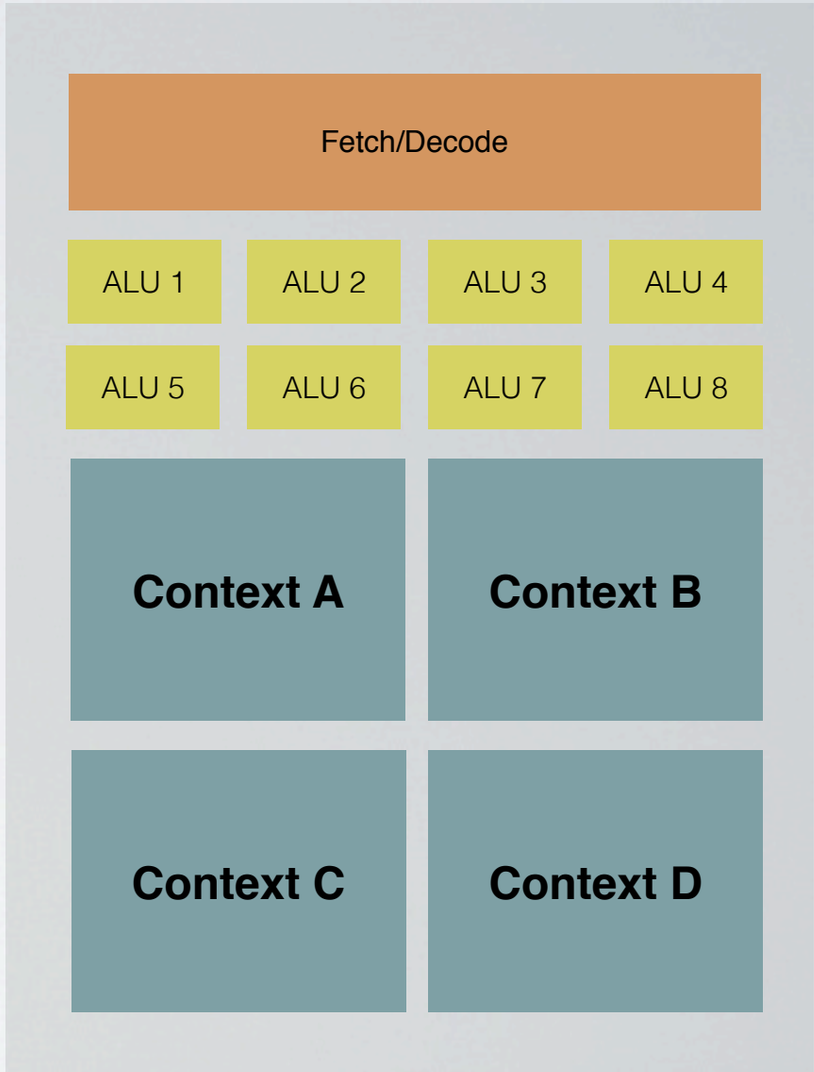
Kaip?


A
B
C
D
Laikas
1 2 3 4 5 6 7 8
9 ... 16
17 ... 24
25 ... 32

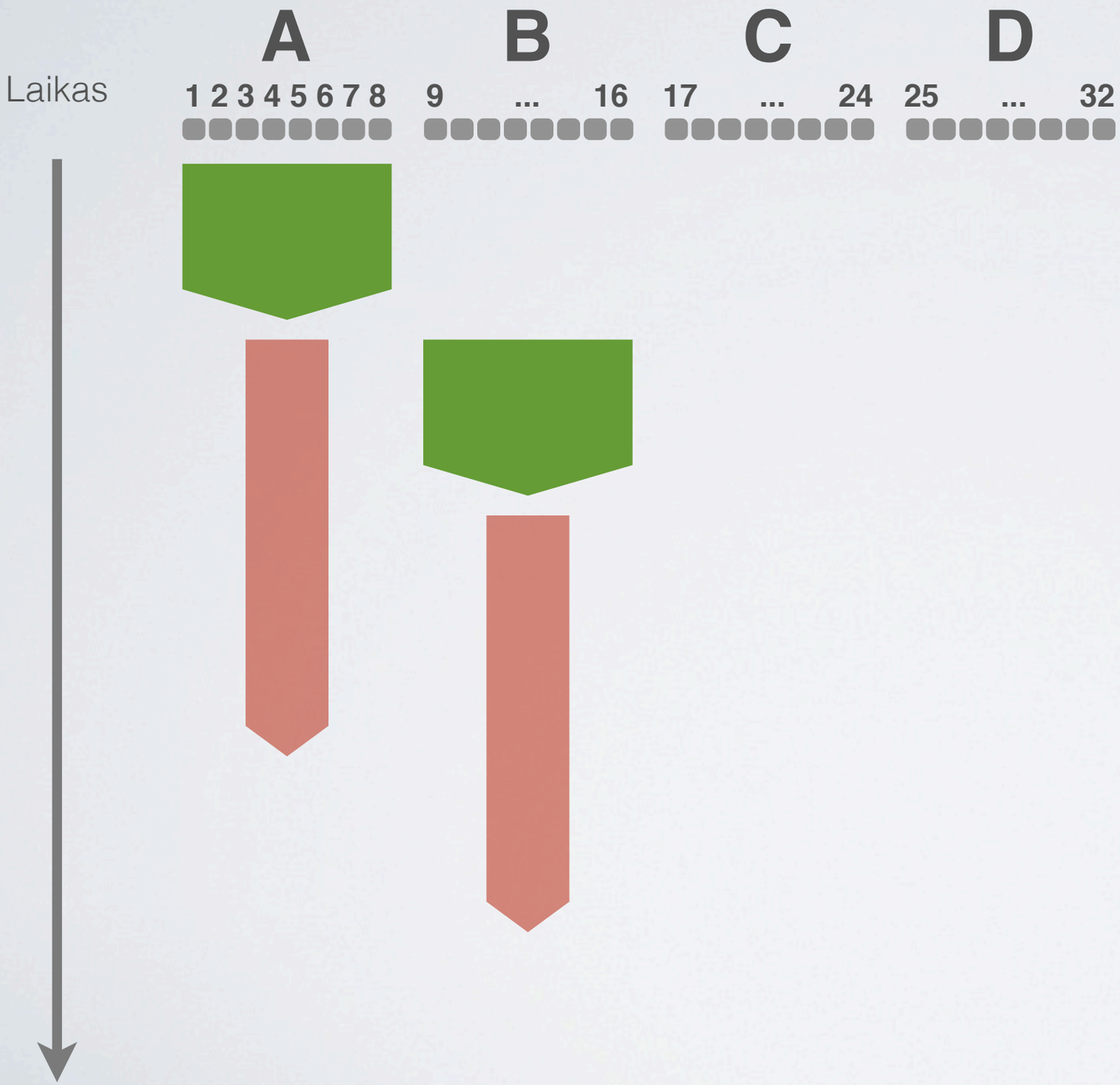

Fetch/Decode
ALU 1
ALU 2
ALU 3
ALU 4
ALU 5
ALU 6
ALU 7
ALU 8
Context A
Context B
Context C
Context D

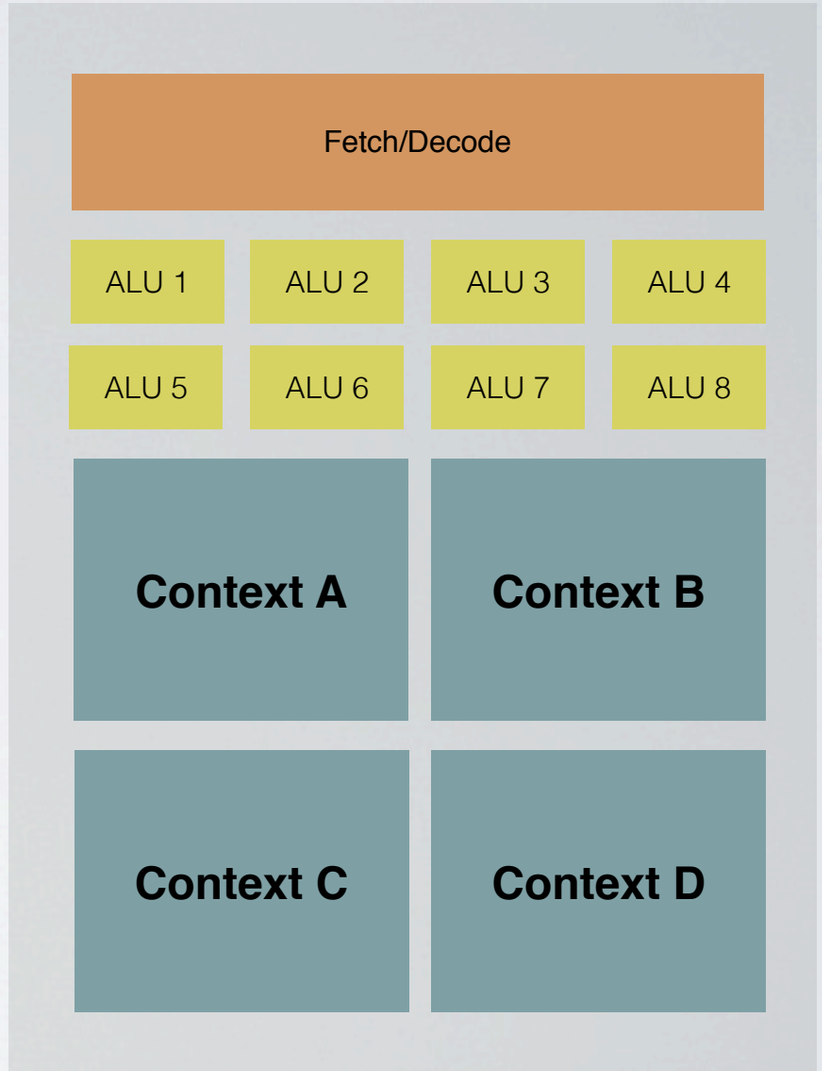
Kaip?


A
B
C
D
Laikas
1 2 3 4 5 6 7 8
9 ... 16
17 ... 24
25 ... 32

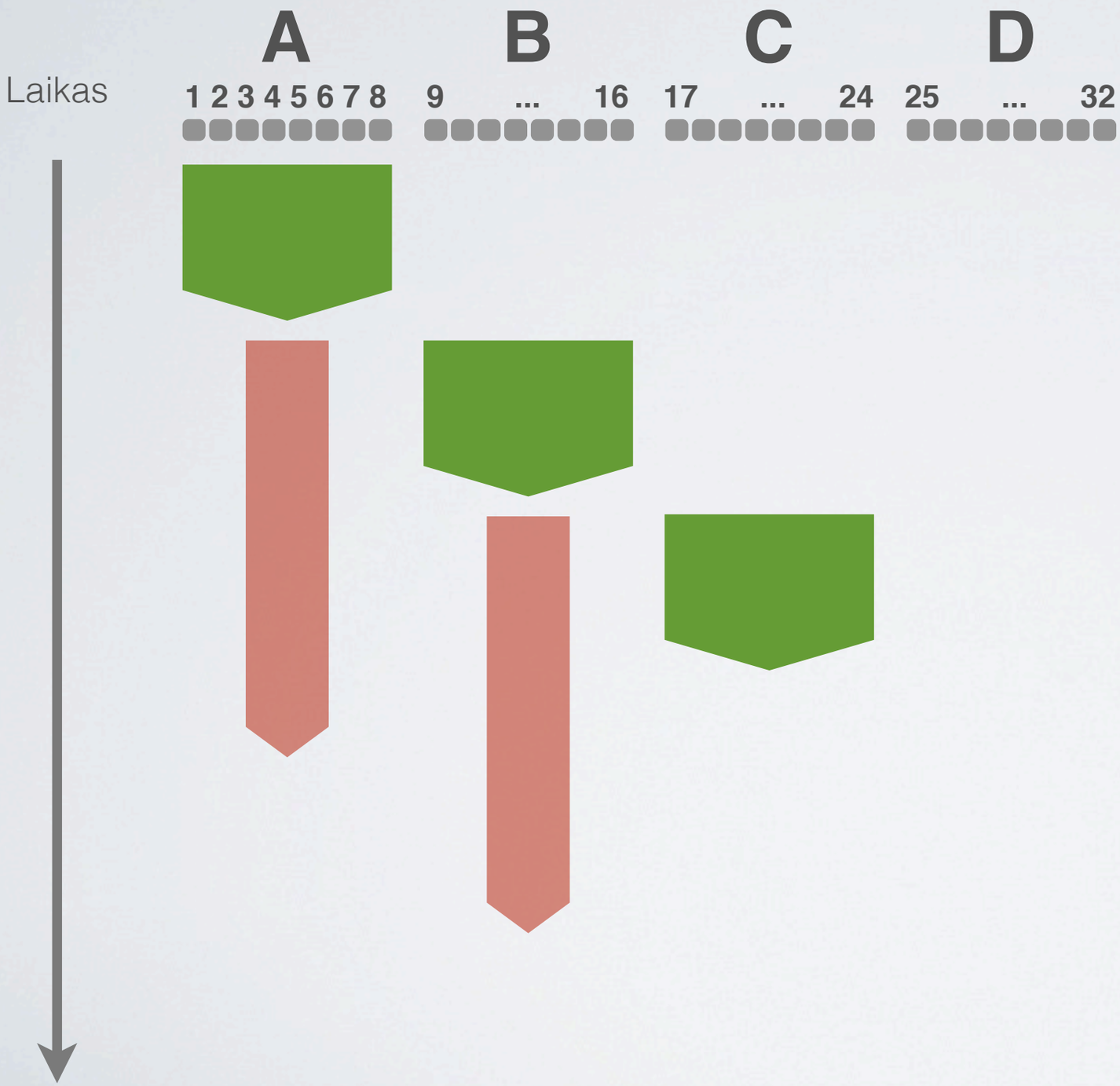

Fetch/Decode
ALU 1
ALU 2
ALU 3
ALU 4
ALU 5
ALU 6
ALU 7
ALU 8
Context A
Context B
Context C
Context D

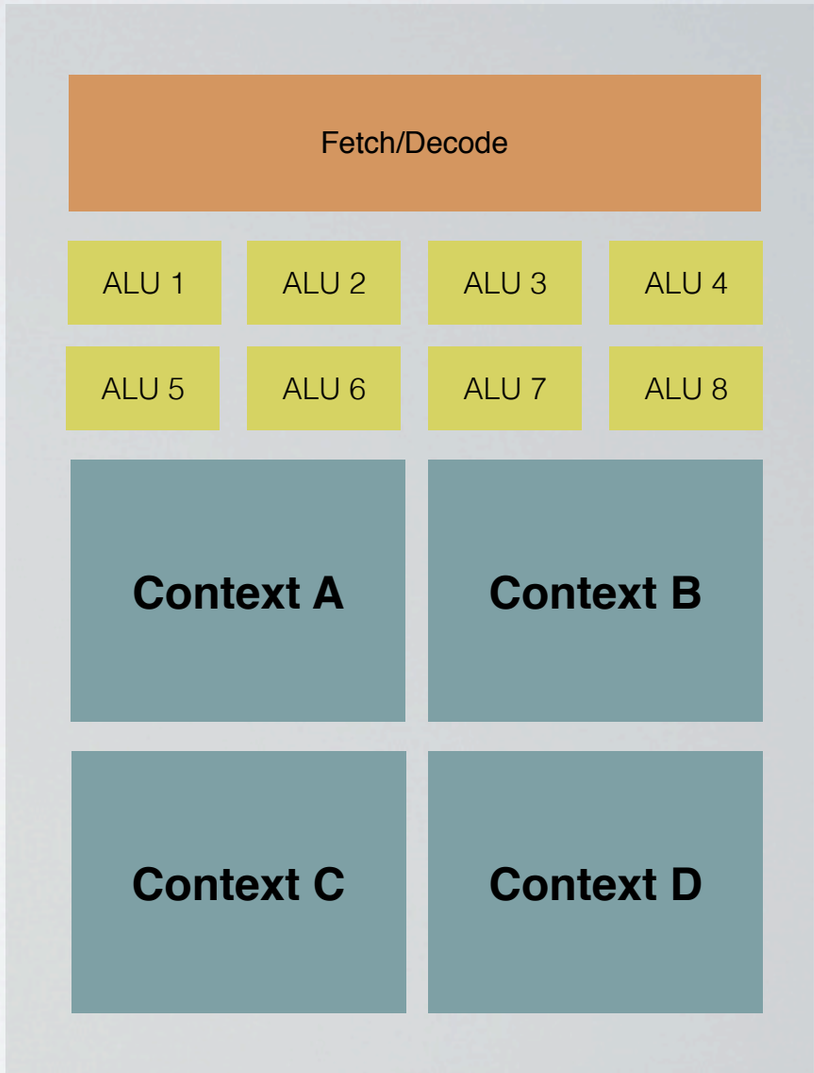
Kaip?


A
B
C
D
Laikas
1 2 3 4 5 6 7 8
9 ... 16
17 ... 24
25 ... 32

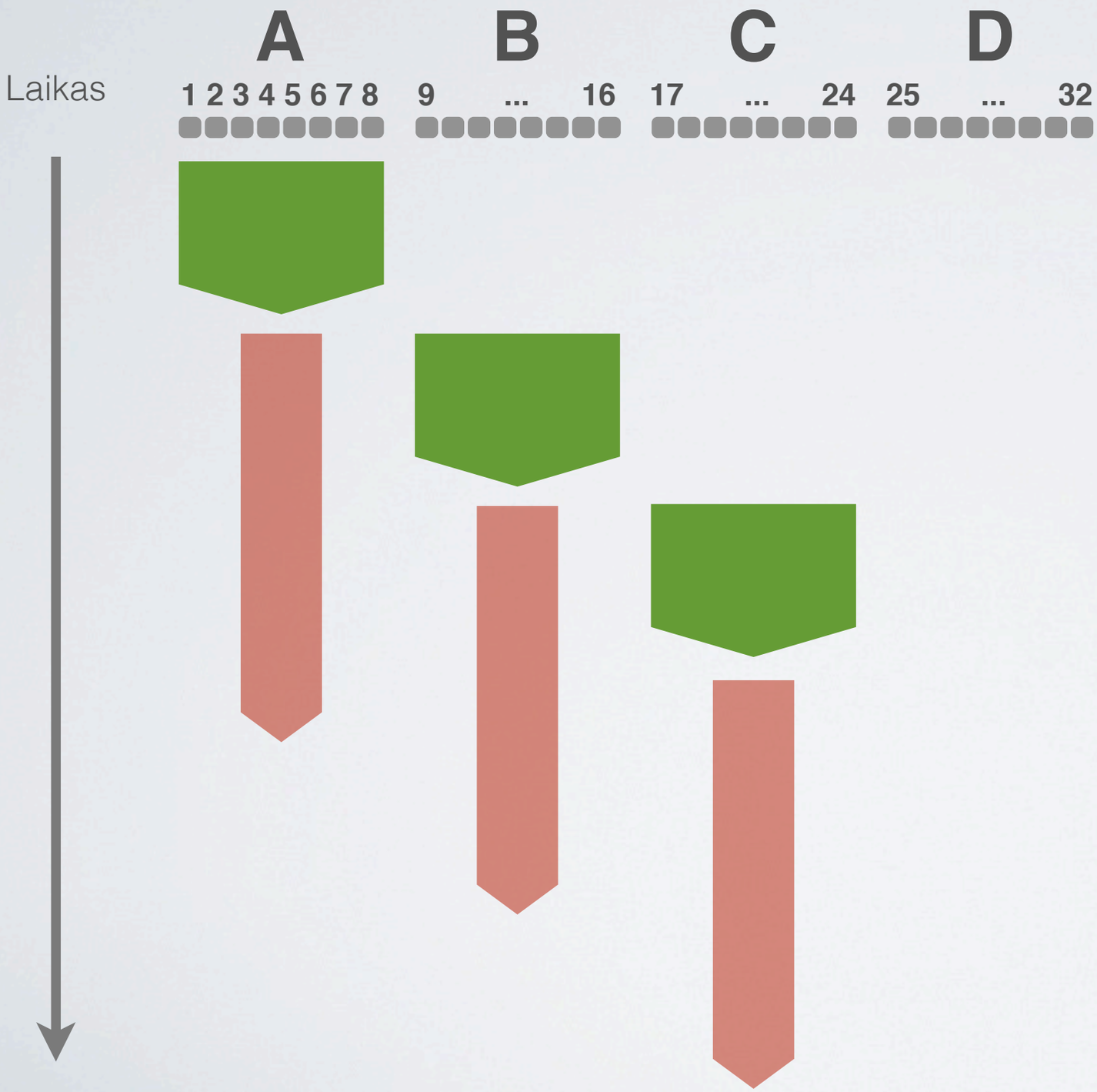

Fetch/Decode
ALU 1
ALU 2
ALU 3
ALU 4
ALU 5
ALU 6
ALU 7
ALU 8
Context A
Context B
Context C
Context D

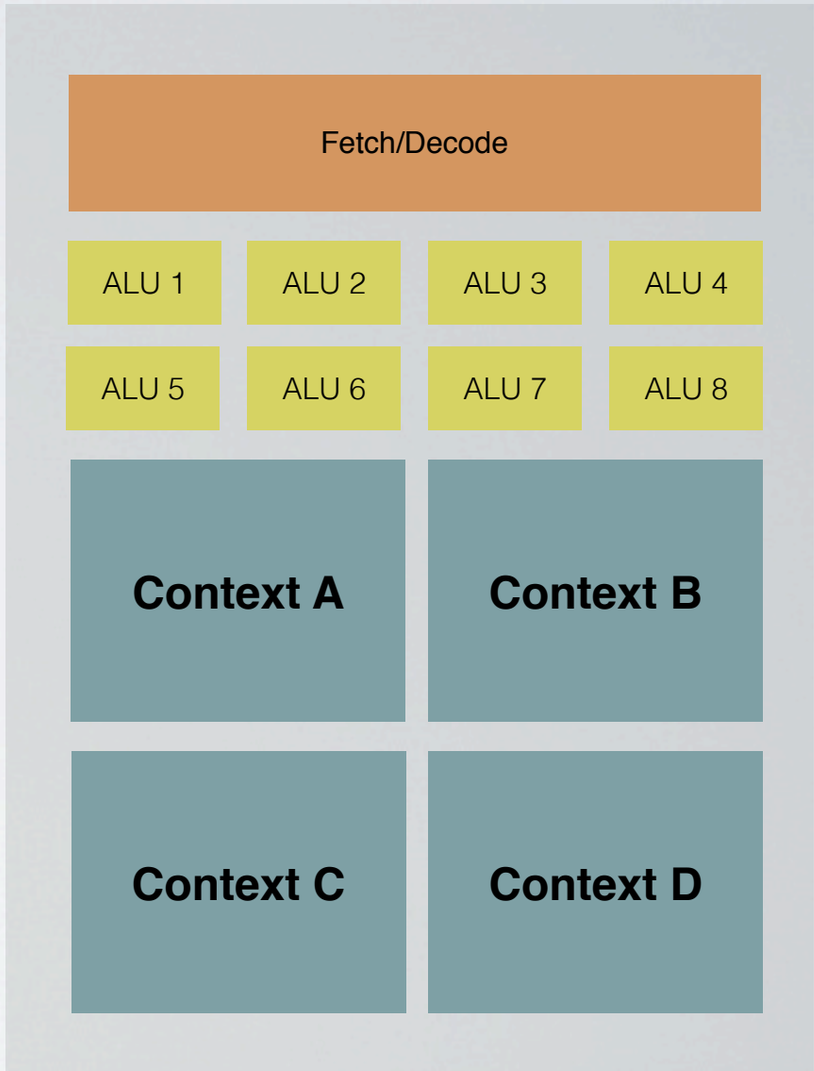
Kaip?


A
B
C
D
Laikas
1 2 3 4 5 6 7 8
9 ... 16
17 ... 24
25 ... 32

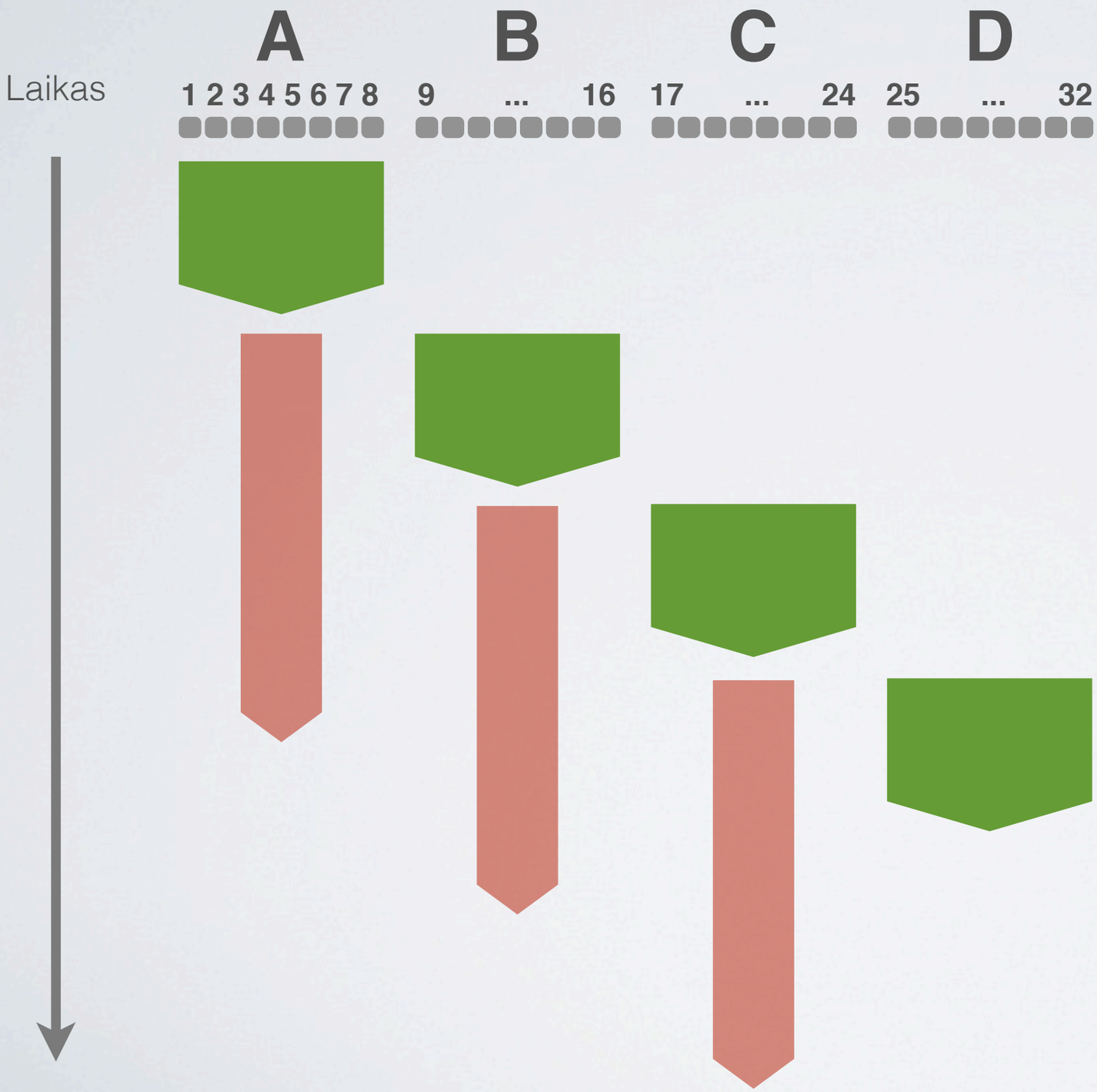

Fetch/Decode
ALU 1
ALU 2
ALU 3
ALU 4
ALU 5
ALU 6
ALU 7
ALU 8
Context A
Context B
Context C
Context D

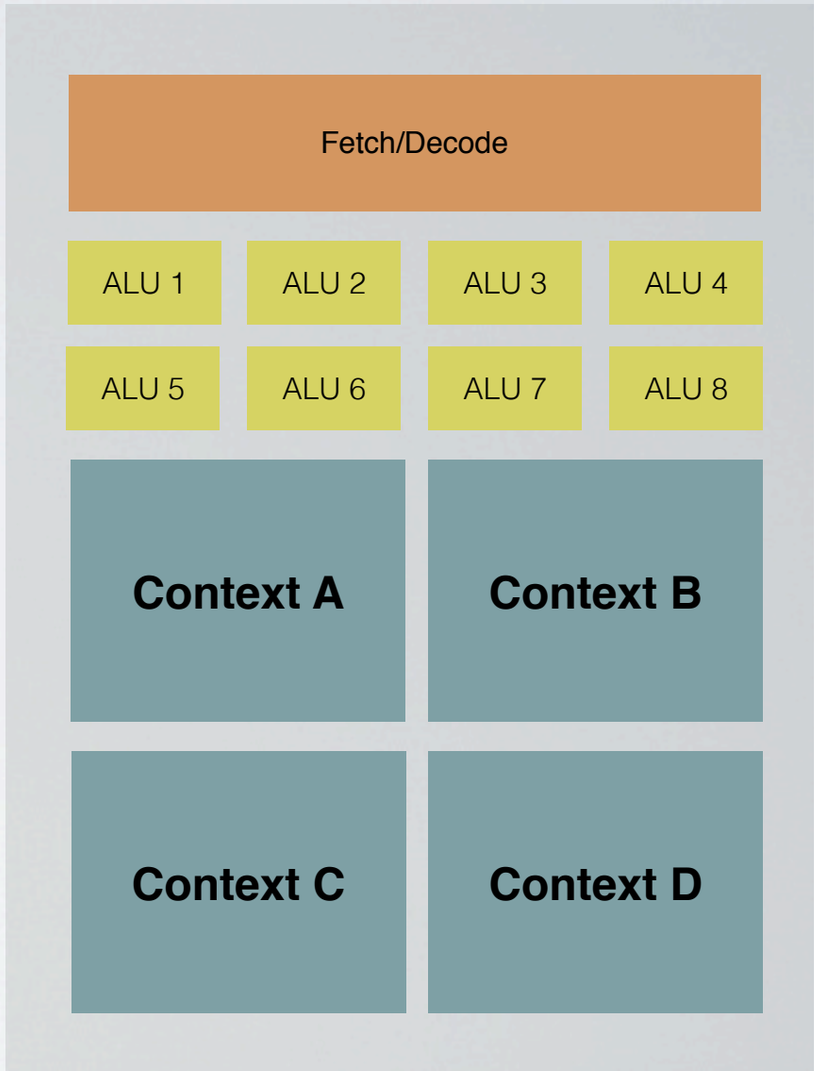
Kaip?


A
B
C
D
Laikas
1 2 3 4 5 6 7 8
9 ... 16
17 ... 24
25 ... 32

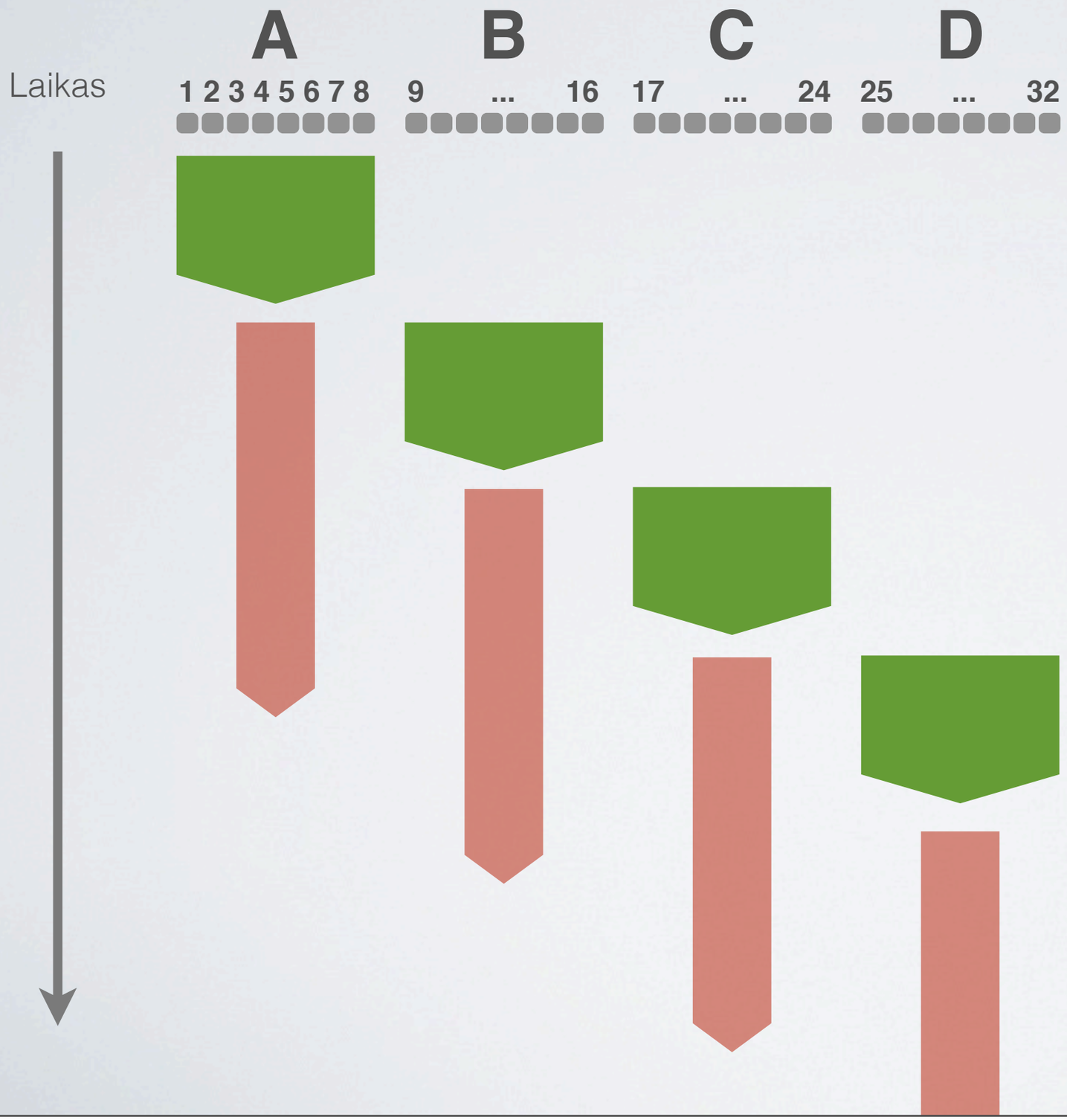

Fetch/Decode
ALU 1
ALU 2
ALU 3
ALU 4
ALU 5
ALU 6
ALU 7
ALU 8
Context A
Context B
Context C
Context D

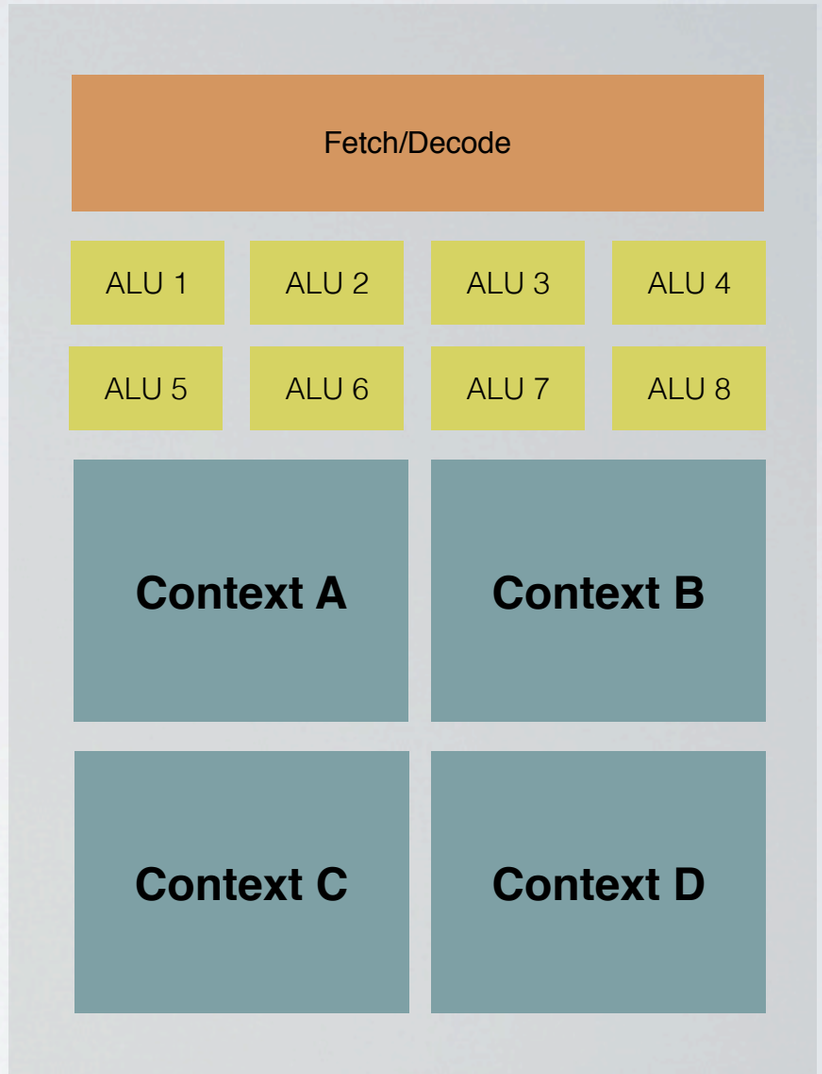
Kaip?


A
B
C
D
Laikas
1 2 3 4 5 6 7 8
9 ... 16
17 ... 24
25 ... 32

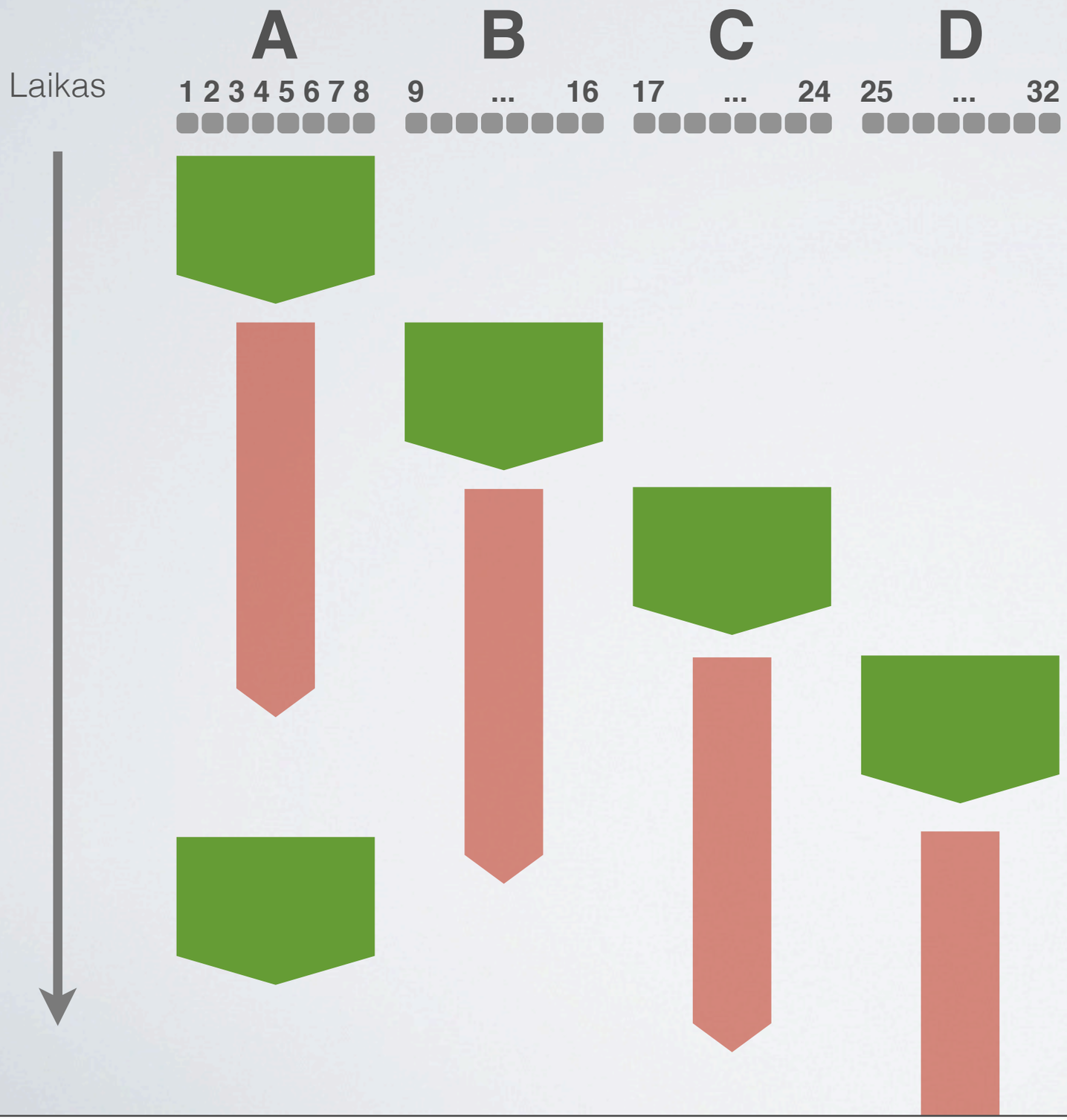

Fetch/Decode
ALU 1
ALU 2
ALU 3
ALU 4
ALU 5
ALU 6
ALU 7
ALU 8
Context A
Context B
Context C
Context D

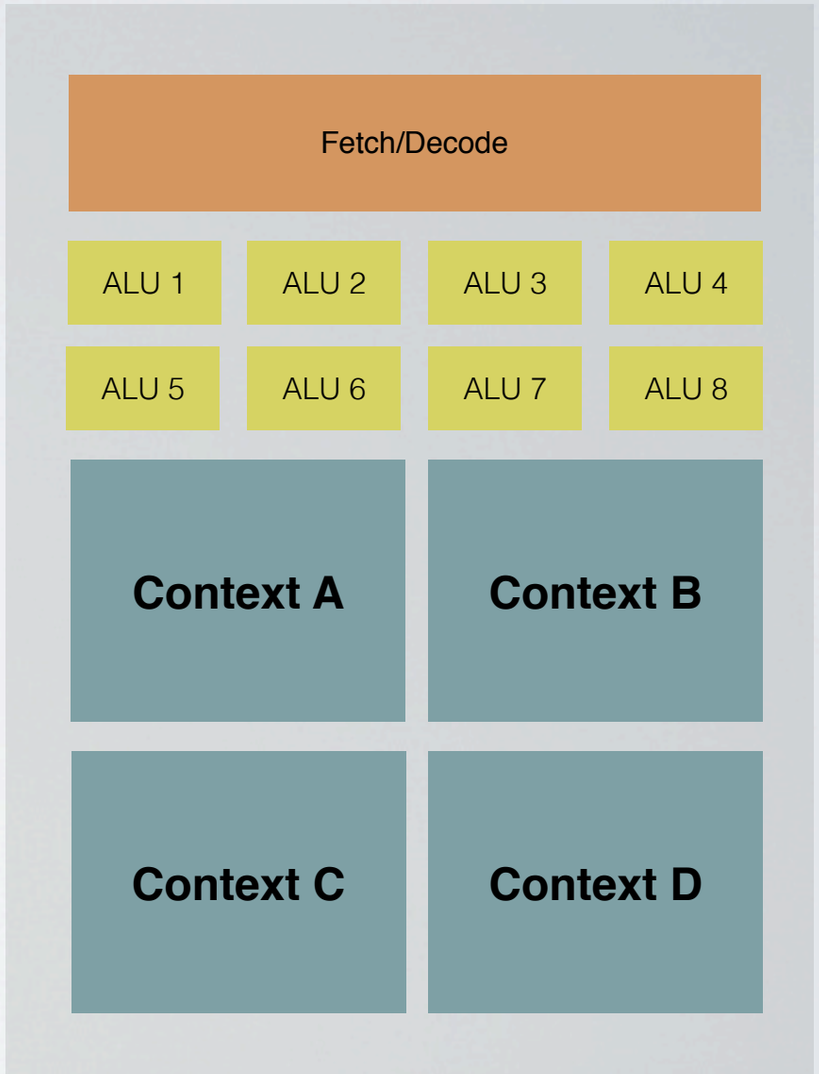
Kaip?


A
B
C
D
Laikas
1 2 3 4 5 6 7 8
9 ... 16
17 ... 24
25 ... 32

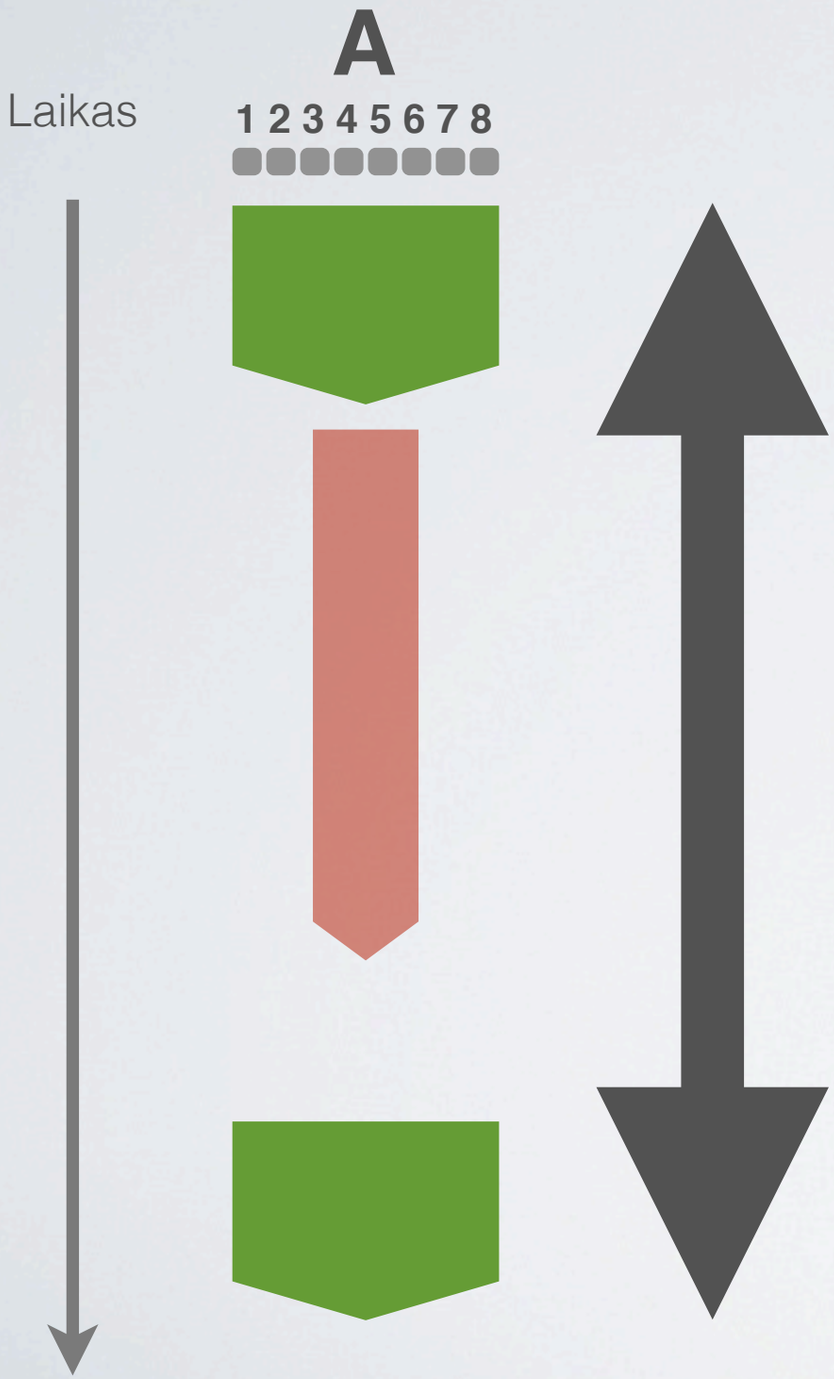

Fetch/Decode
ALU 1
ALU 2
ALU 3
ALU 4
ALU 5
ALU 6
ALU 7
ALU 8
Context A
Context B
Context C
Context D

Kaip?


A
B
C
D
Laikas
1 2 3 4 5 6 7 8
9 ... 16
17 ... 24
25 ... 32


Fetch/Decode
ALU 1
ALU 2
ALU 3
ALU 4
ALU 5
ALU 6
ALU 7
ALU 8
Context A
Context B
Context C
Context D

Throughput



- *Throughput* gerėja - procesorius visą laiką kažką daro
- *Latency* blogėja - grupė pikselių dabar suskaičiuojama per ilgesnį laiką

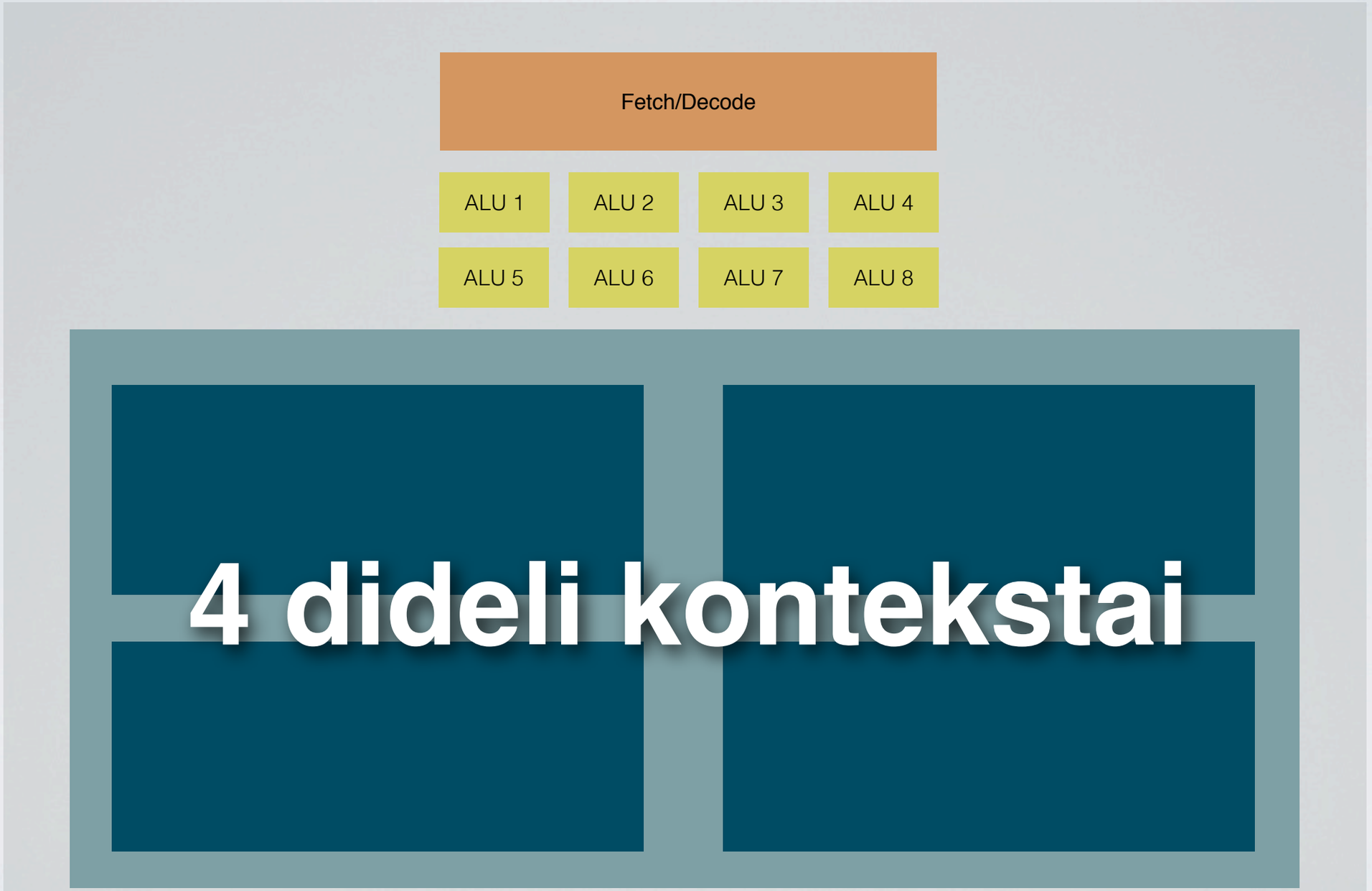
- CPU: labai geras latency, šiaip sau throughput
- GPU: labai geras throughput, šiaip sau latency

Kontekstų saugykla

Kontekstų saugykla

Kontekstų saugykla

Kontekstų saugykla



Trys idėjos

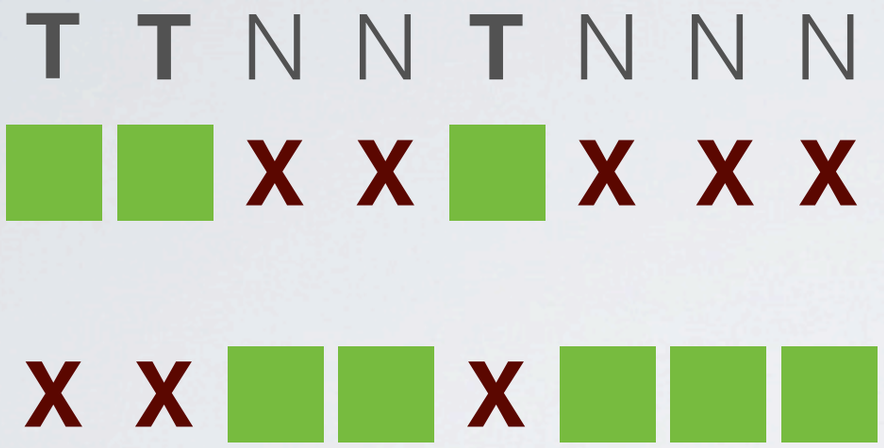
- #1: daug paprastų branduolių
- #2: kiekvienas branduolys apdoroja keletą dalykų viena programa (SIMD)
- #3: persijunginėti tarp apdorojamų grupių, kol laukiam duomenų iš atminties

OPTIMIZAVIMAS

Just don't do it

- Ko nors *nedaryti* visada efektyviau negu ką nors daryti
- 20 šviesos šaltinių? Fake it. *Two lights should be enough for anyone.*
- Šešėliai po veikėjais? *Pfft*. Padėk po kojomis pusiau permatomą "plėmą".
- *Use good artwork you must.*

Vengti sąlygos sakinių




```
if (x > 0)
   y = pow (x, 32);
else
   y = 0;
```

- Tokių, kurių rezultatas dažnai skiriasi tarp atskirų pikselių
- Sąlyga turi sutaupyti *daug* skaičiavimų
  - “if” tam kad sutaupyti keletą sandaugų - **ne**!
  - keliasdešimt sandaugų - *galbūt*

Math is cheap

- Skaitymas iš atminties (tekstūrų) - brangu*
  - Tūkstančiai taktų, kol duomenys bus nuskaityti
- Skaičiuoti ką nors su GPU - pigu*
  - Pusantro tūkstančio* procesorių po ranka!

- * **Yra papildomų sąlygų**. Rezultatai gali skirtis tarp platformų, ypač tarp PC ir mobilių įrenginių. Gali būti riešutų pėdsakų. Pasitarkite su gydytoju ar vaistininku.

Kur skaičiuoti?

- Objektų scenoje: keletas tūkstančių
- Viršūnių: iki milijono
- Pikselių: dešimtys milijonų

- Ar galima skaičiavimus nuo pikselių perkelti ant viršūnių?
- Ar galima nuo viršūnių perkelti ant objektų?

Level Of Detail (LOD)

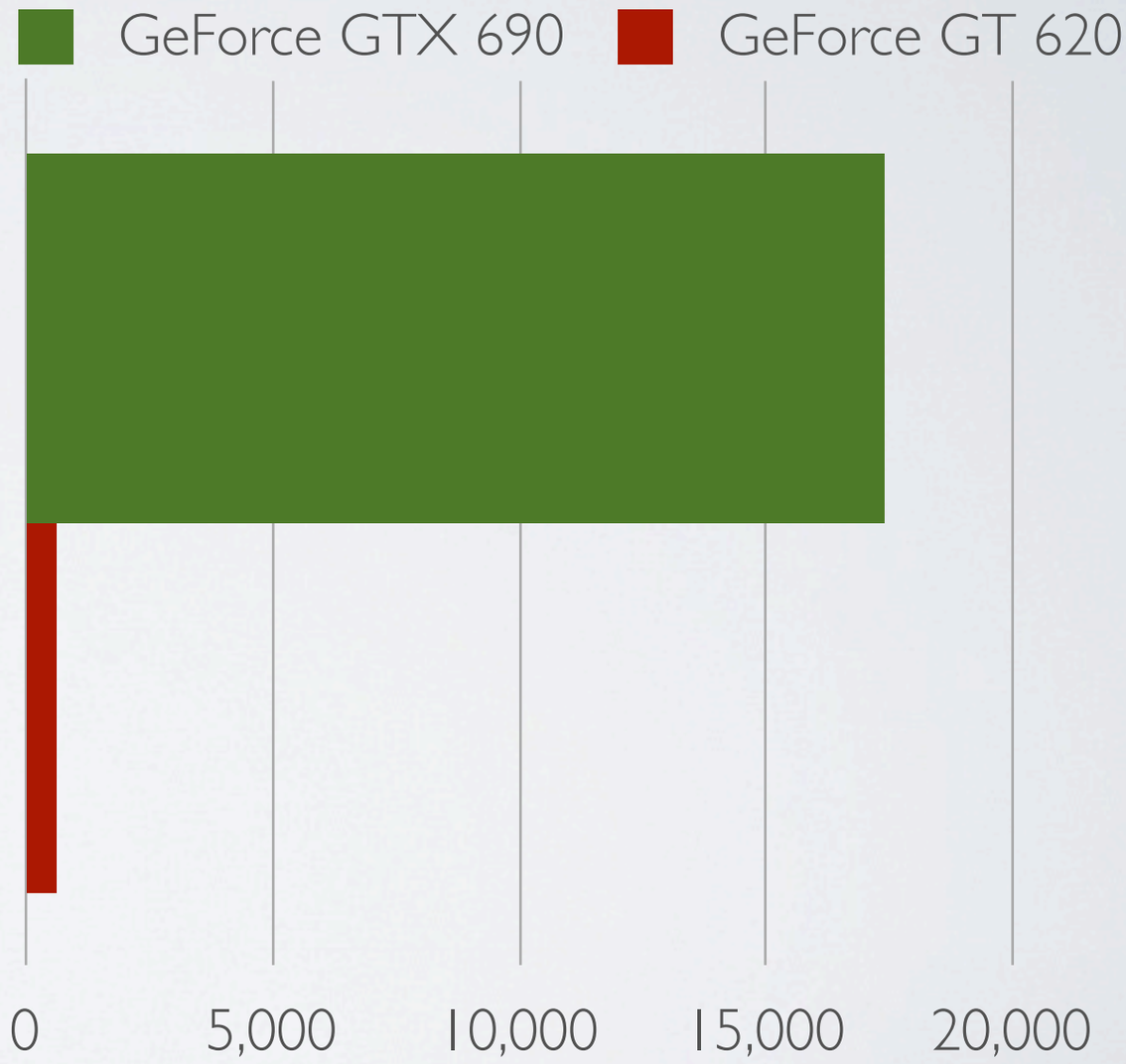
- Objektas labai mažas ekrane? *(arba šiaip "nesvarbus")*
  - Gal jį tiesiog išjungti ir niekas nepastebės?
  - Naudoti paprastesnį modelį
  - Naudoti paprastesnį šeiderį
  - Neskaičiuoti jam šešėlių
  - Skaičiuoti mažiau šviesos šaltinių

Mobilios platformos

- iPad3, 2048 x 1536
- Xbox 360, daugelis žaidimų 1280 x 720 ar net mažiau
  - 3.4 *karto* mažiau pikselių!
  - GPU galia ir atminties pralaidumas 5-20x didesnis
  - *Do the math ;)*
- Saugokis pikselių! Jų yra daug!

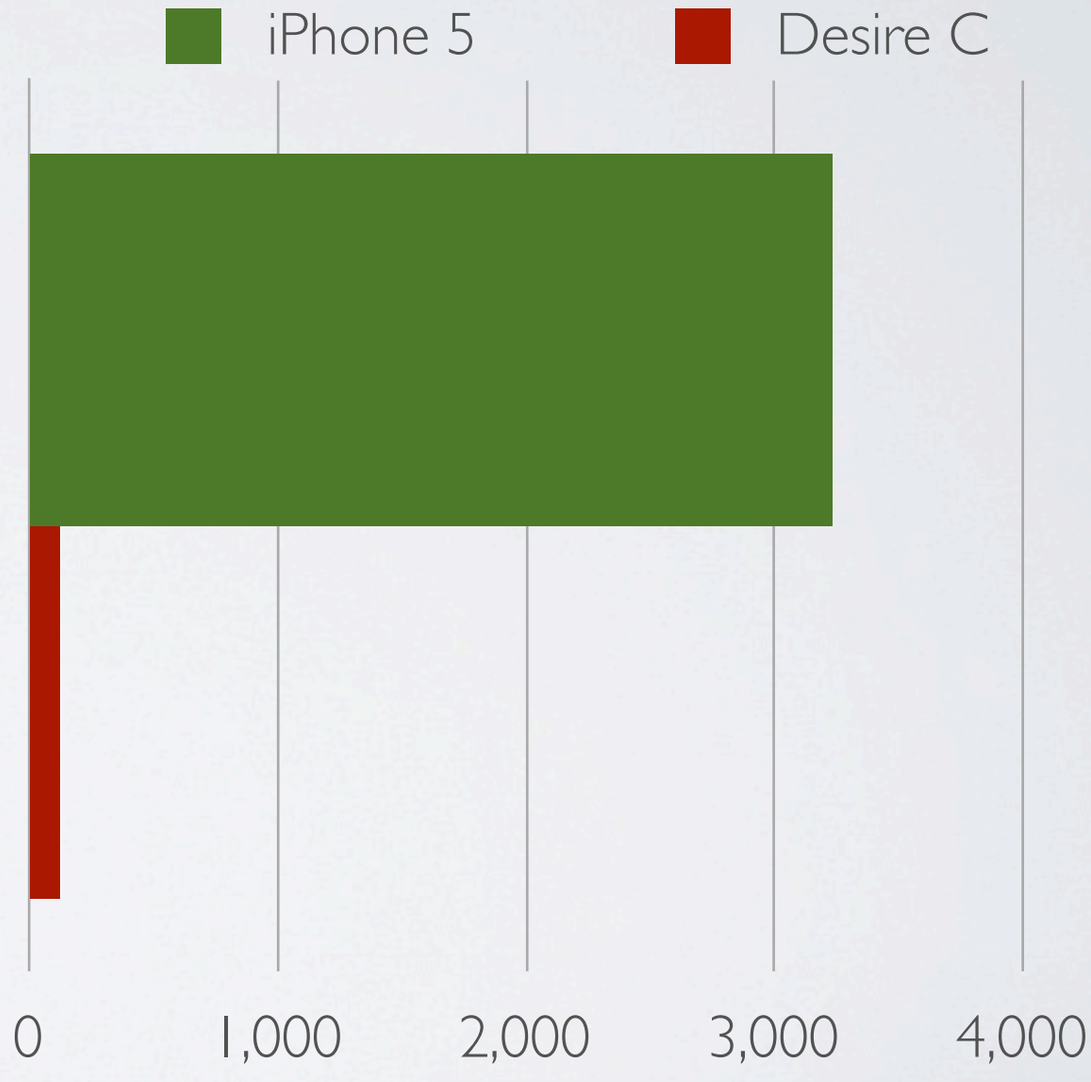
Liūdna realybė, PC

- 3DMark 11
- GeForce GTX 690: **17410**
- GeForce GT 620: **620**
- Abu GPU 2012 metų
- **28X** skirtumas!



Liūdna realybė, mobile

- GLBenchmark 2.5
- iPhone5 (SGX 543MP3): **3236**
- HTC Desire C (Adreno 200): **123**
- Abu telefonai 2012 metų
- **26X** skirtumas!



EFEKTAI

Efektai?

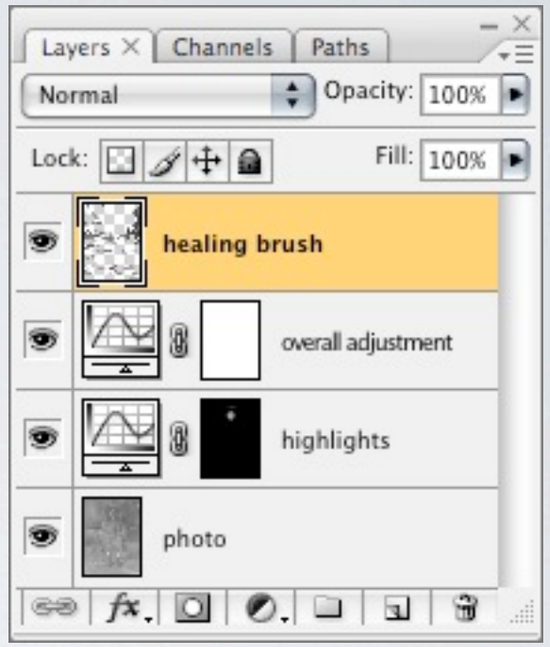
Efektai?

Efektai

- Paviršiaus savybės
- Apšvietimas
- Spec. efektai
- Vaizdo apdorojimas
- Kita

Paviršiaus savybės

Kaip Photoshop layeriai



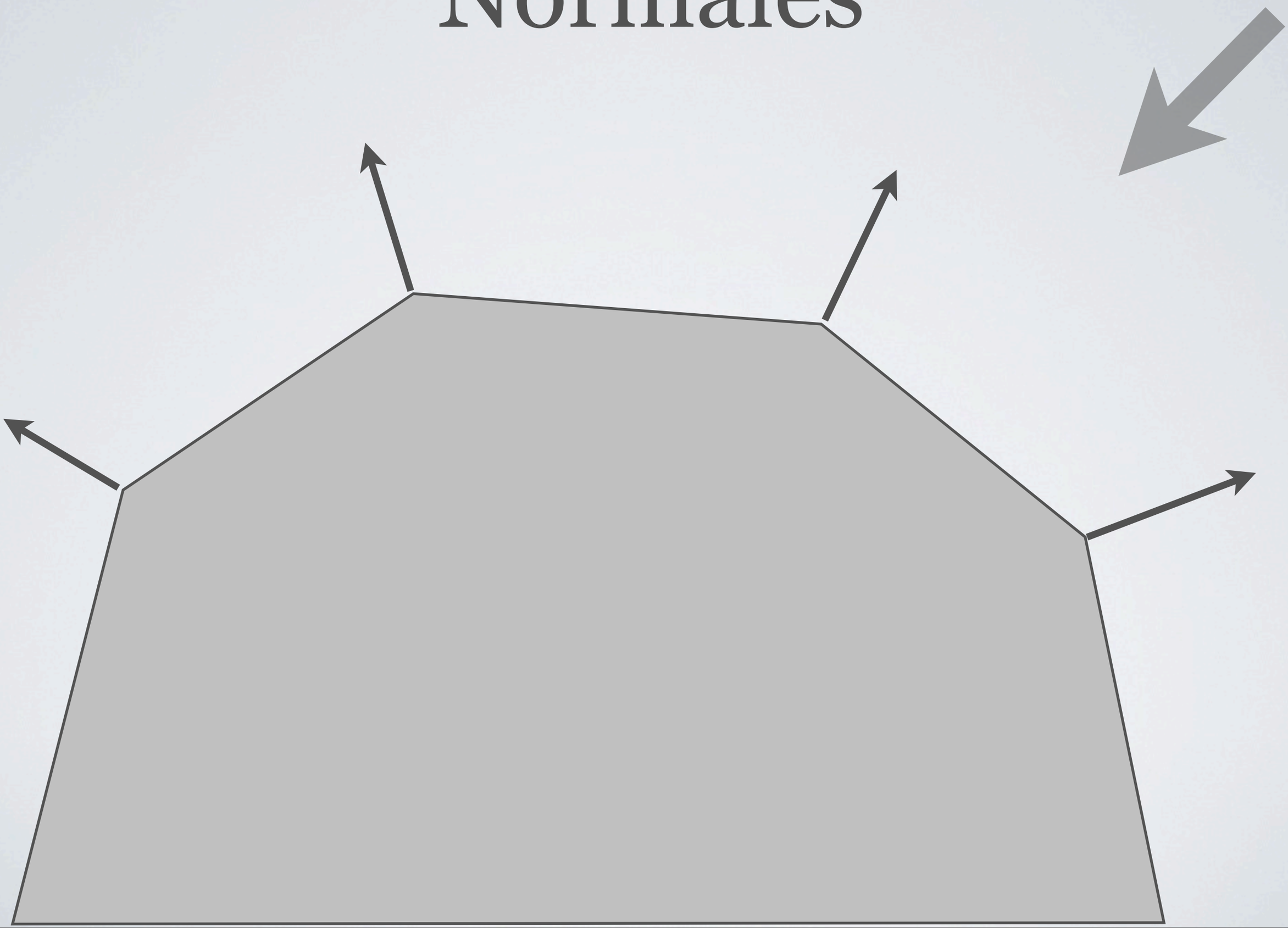
- Įvairiai miksuojam keletą tesktūrų
- Žaidimuose dažnai 1..4

- Normal: result = mix (base, blend, blend.a);
- Multiply: result = base * blend;
- Darken: result = min (base, blend);
- Lighten: result = max (base, blend);
- Dodge: result = base / (vec4(1.0) - blend);
- ...

Nespalvinės tekstūros

- Dažnai tekstūrose *nėra* spalvos!
- *Normal maps*
- *Specular maps*
- *Light maps*
- ...

Normalės



Daugiau detalių

Bet tam reikia daugiau viršūnių...

Daugiau detalių?

Mažai viršūnių, bet *kažkaip* turėkime daug normalių

Normalė vietoj spalvos

- Spalva: R, G, B. Reikšmės tarp 0..1
- Normalė: vektorius, X, Y, Z. Reikšmės tarp -1..1

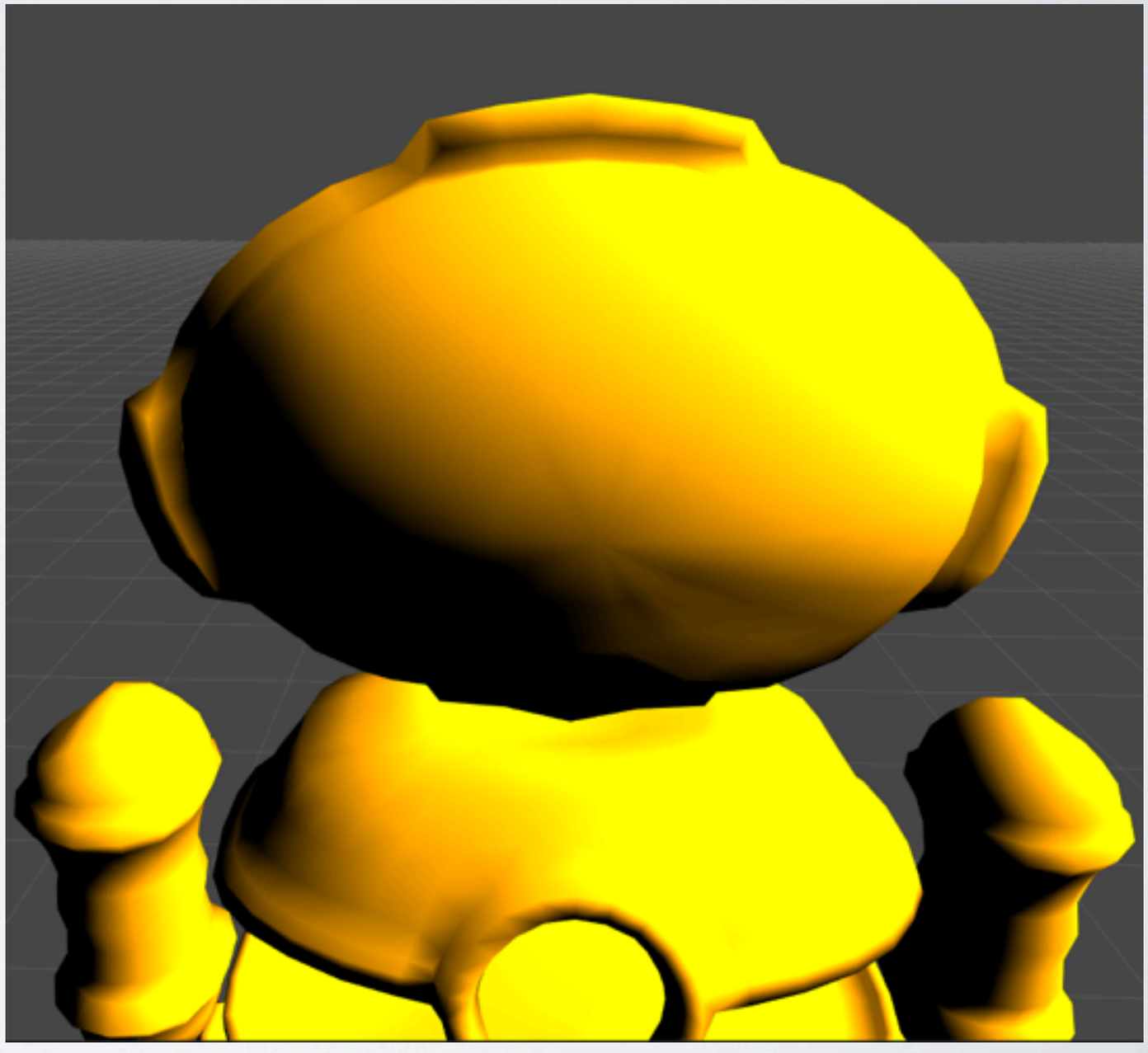
```
vec3 col = texture2D(normalmap, uv).rgb;
vec3 nor = col * 2.0 - 1.0;
float d = dot (nor, lightDir);
// ...
```

Apšvietimas

- Labai plati tema!
- Rendering equation, BRDFs, kaip skaičiuoti šviesos šaltinius, šešėlius, globalų apšvietimą, gamma, HDR, energy conservation, aliasing, ...
- *Srsly*, labai plati tema ;)

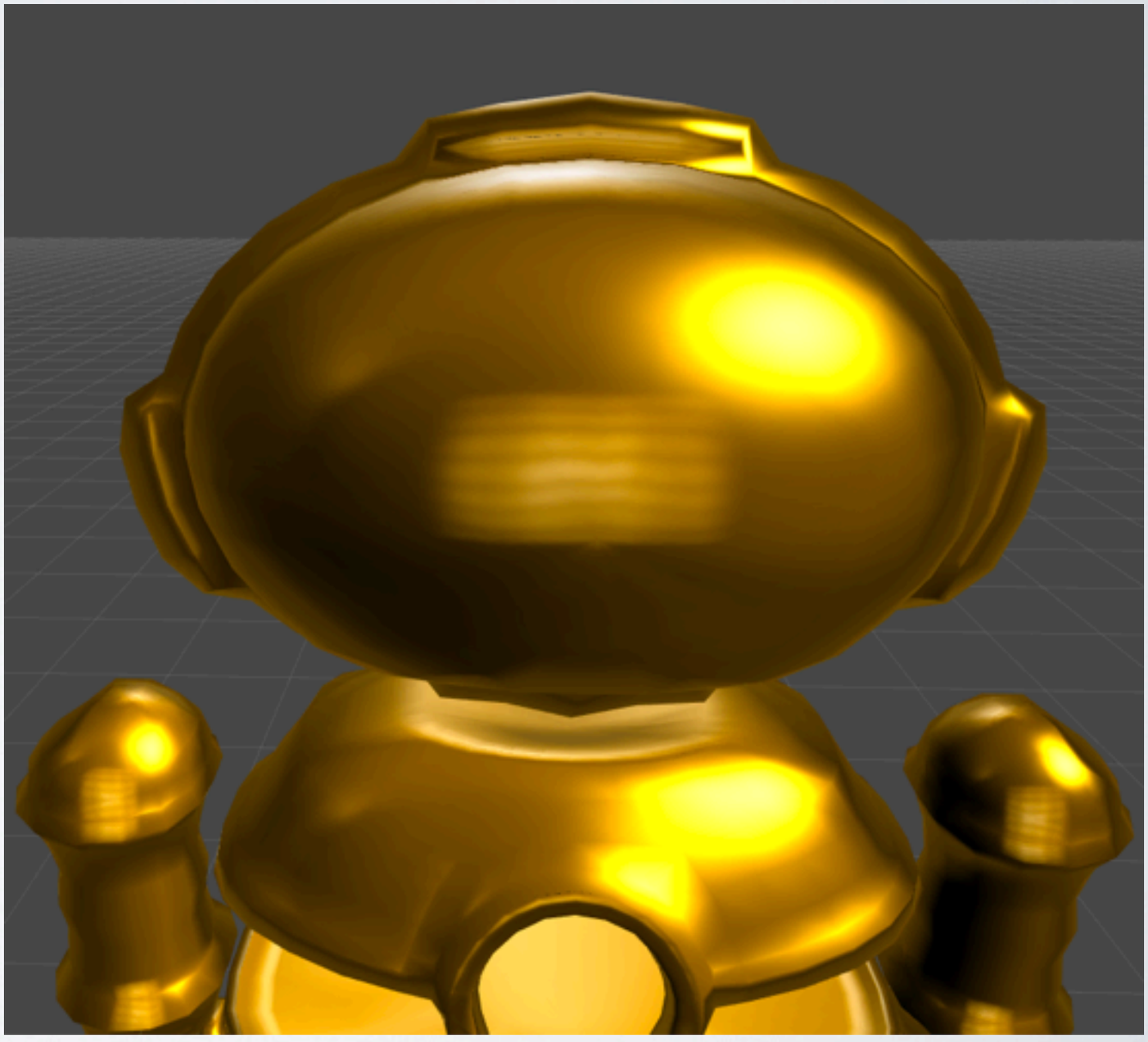
Diffuse aka Lambert BRDF

- N.L
- dot (normal, lightDir)



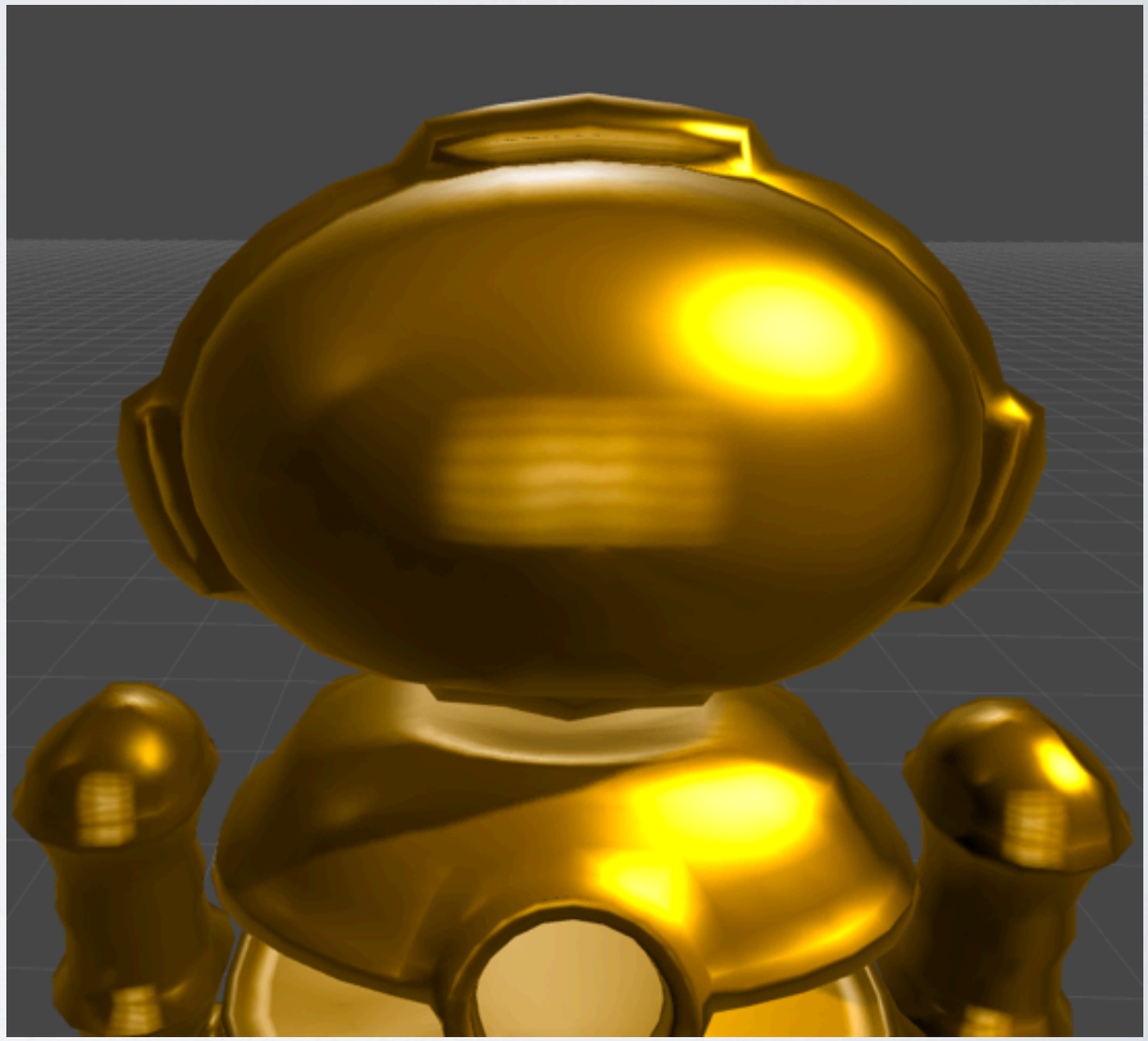
Kažkas sudėtingesnio

- Blinn Phong,
- Cook Torrance,
- Oren Nayar,
- ...

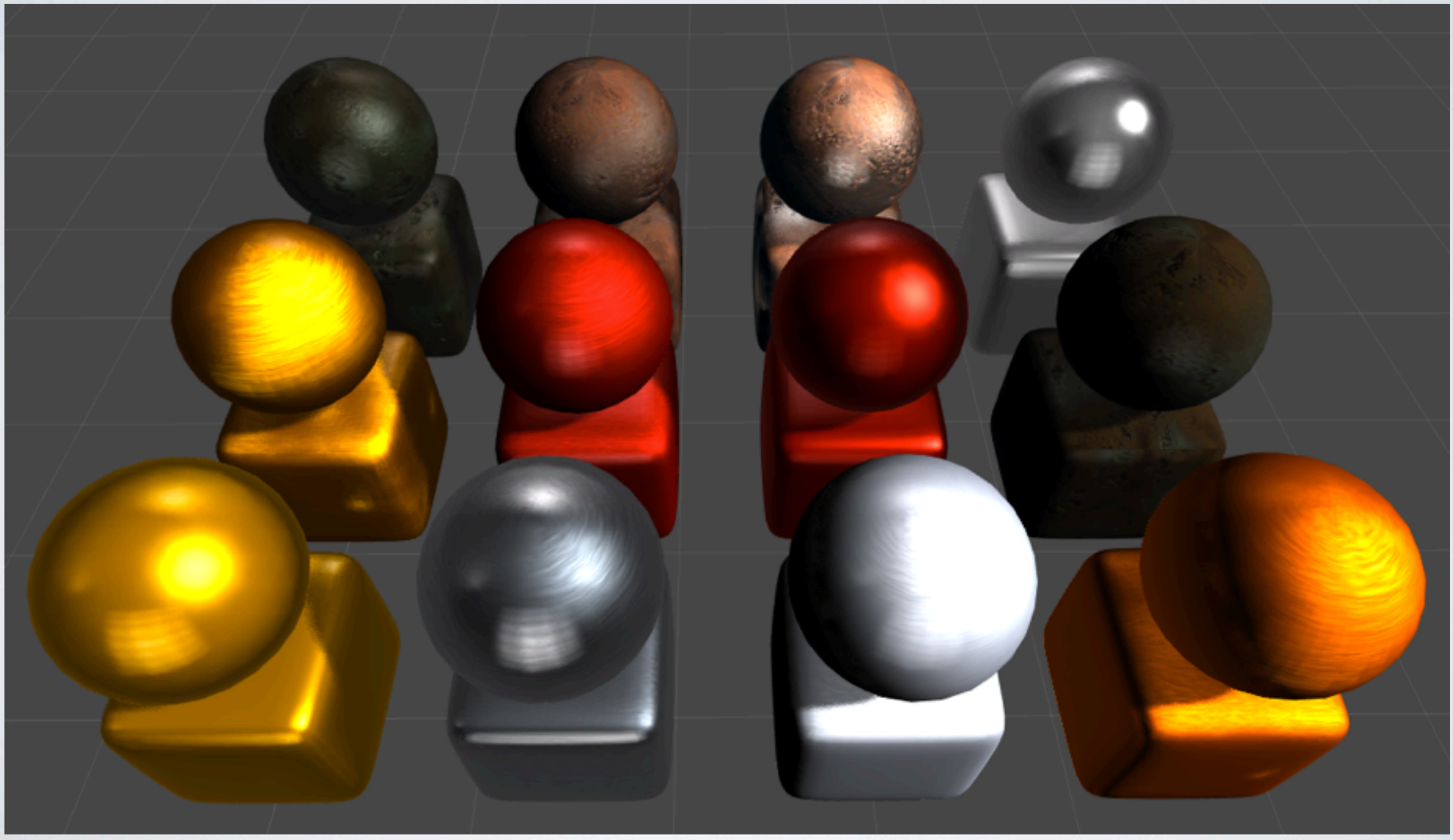


Šešėliai

- Shadow Maps



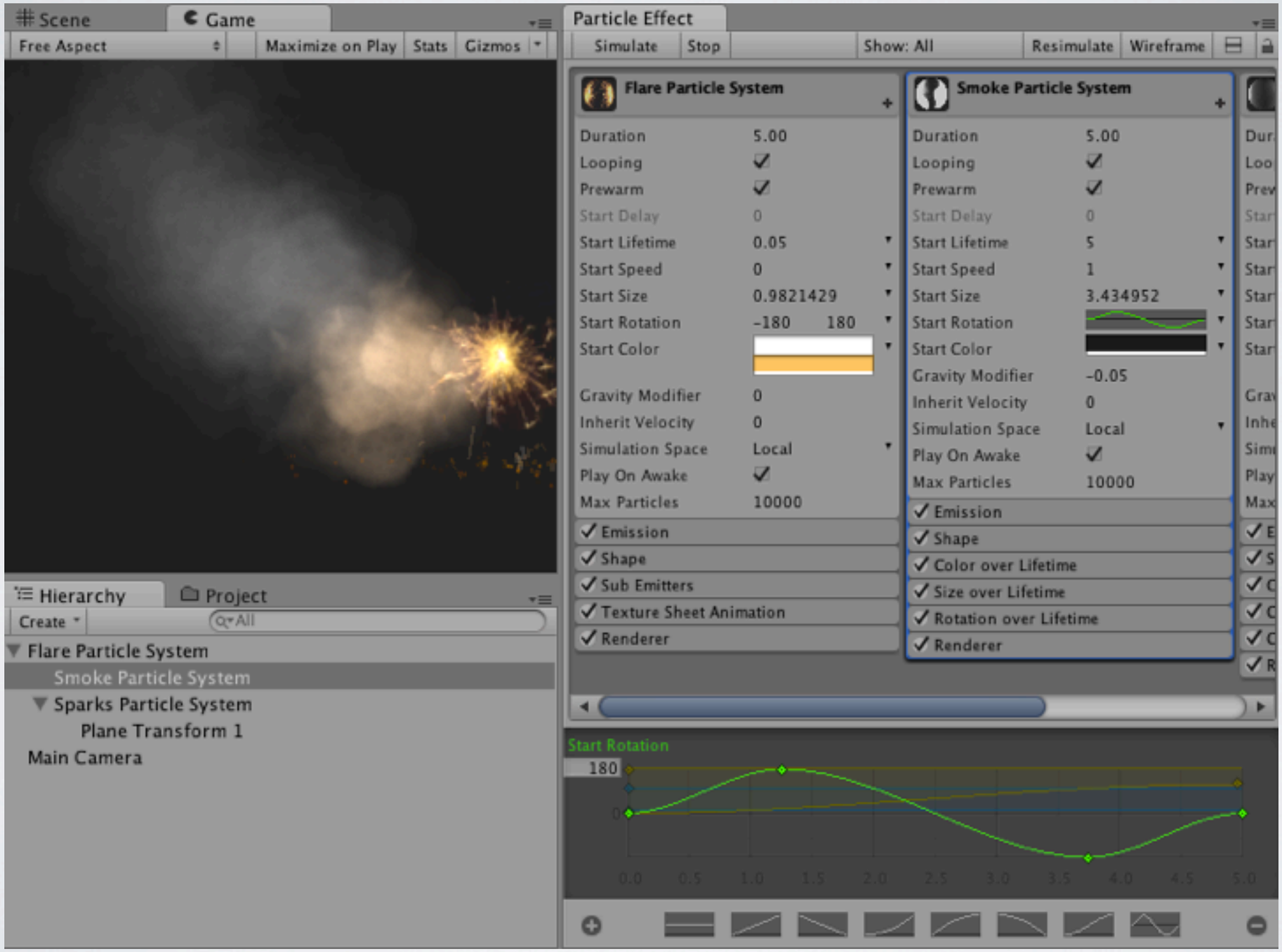
Įvairūs BRDF



Spec. efektai

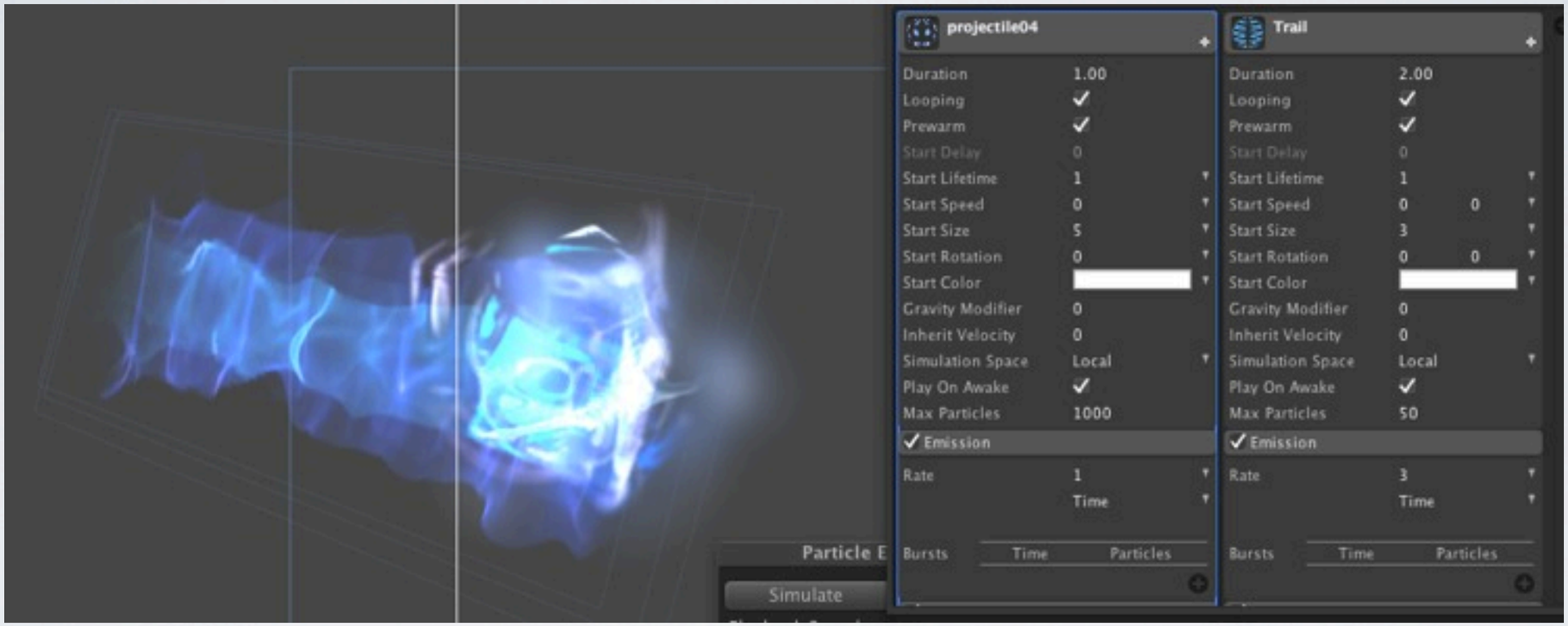
- Dalelių sistemos
- Vanduo

Dalelių sistemos



Dalelių sistemos

Dalelių sistemos

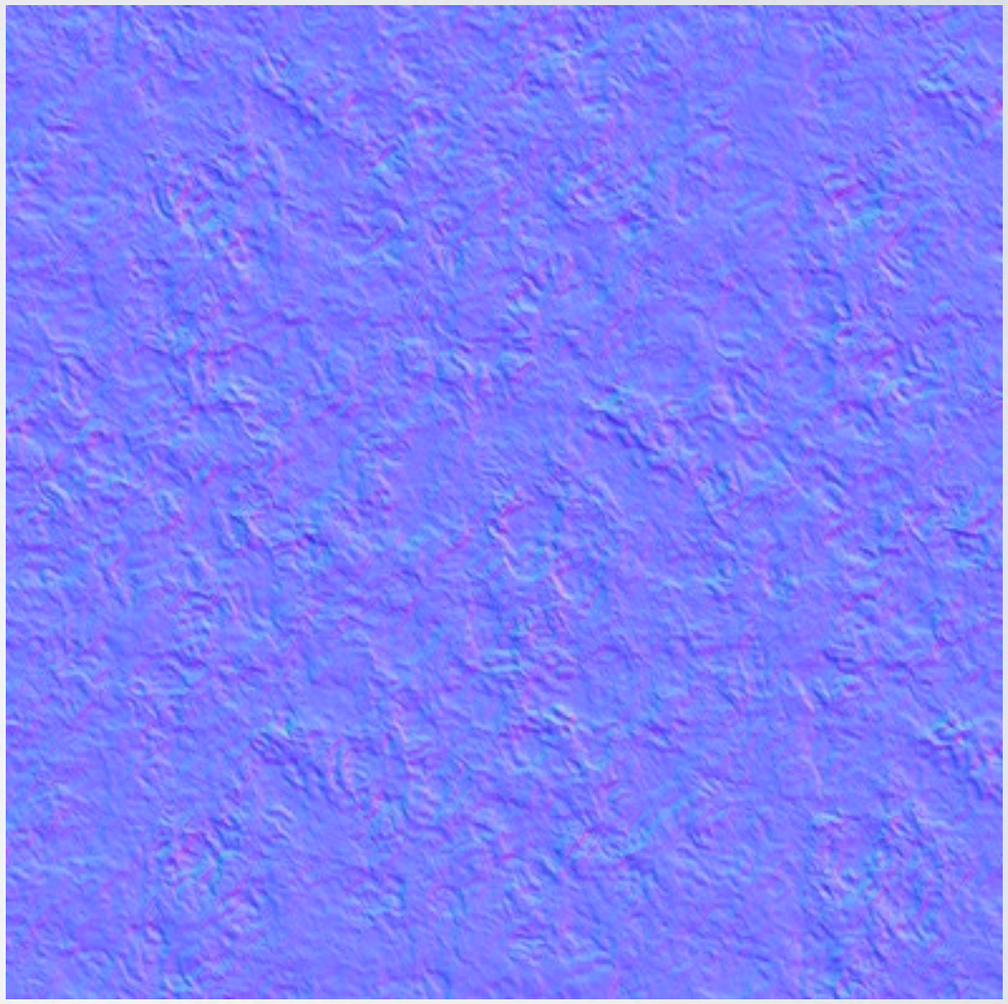


- Dažniausiai tiesiog daug kvadratų
- Gražiai animuotų
- Su gražiomis tekstūromis

Vanduo

Senas geras triukas

- “banguota” normalių tekstūra
- Slinkti ją keletu krypčių
- Suskaičiuoti galutinę normalę
- *Instant wave patterns!*

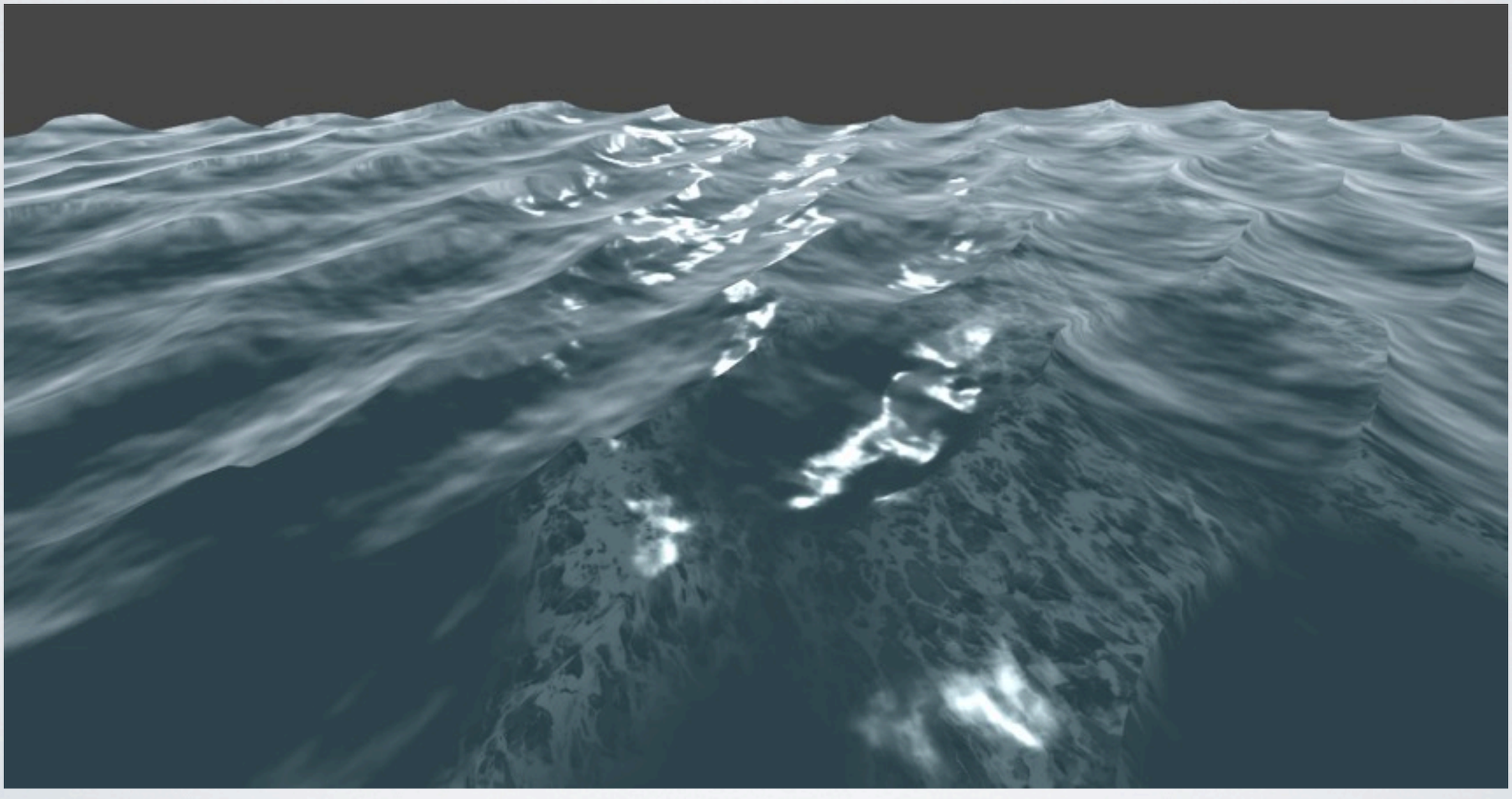


Paprastas vandens šeideris

```
vec3 n1 = texture2D(normalmap, uv + dir1*time).rgb;
vec3 n2 = texture2D(normalmap, uv + dir2*time).rgb;
vec3 nor = (n1 + n2) / 2.0;
nor = nor * 2.0 - 1.0;
float d = dot (nor, lightDir);
// ...
```

Geometrinės bangos

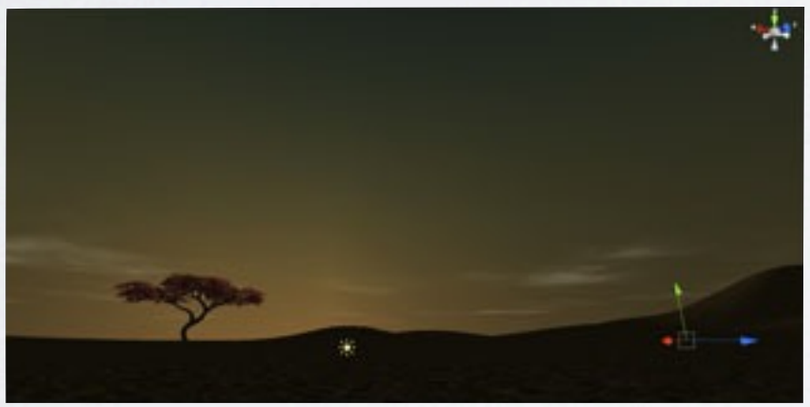
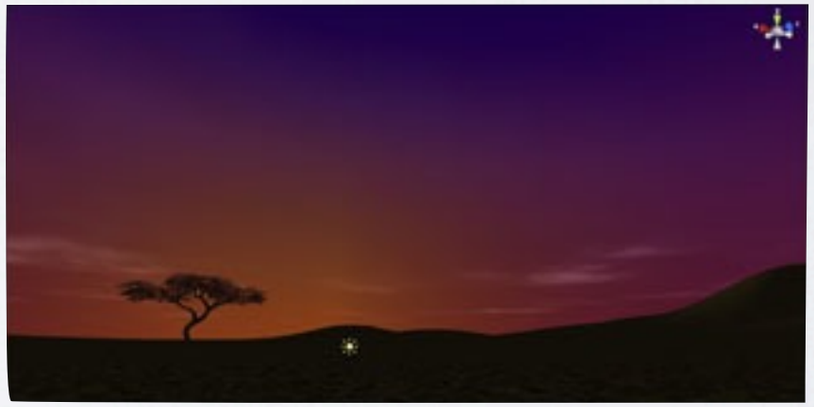
- google “**gerstner waves**”



Vaizdo apdorojimas

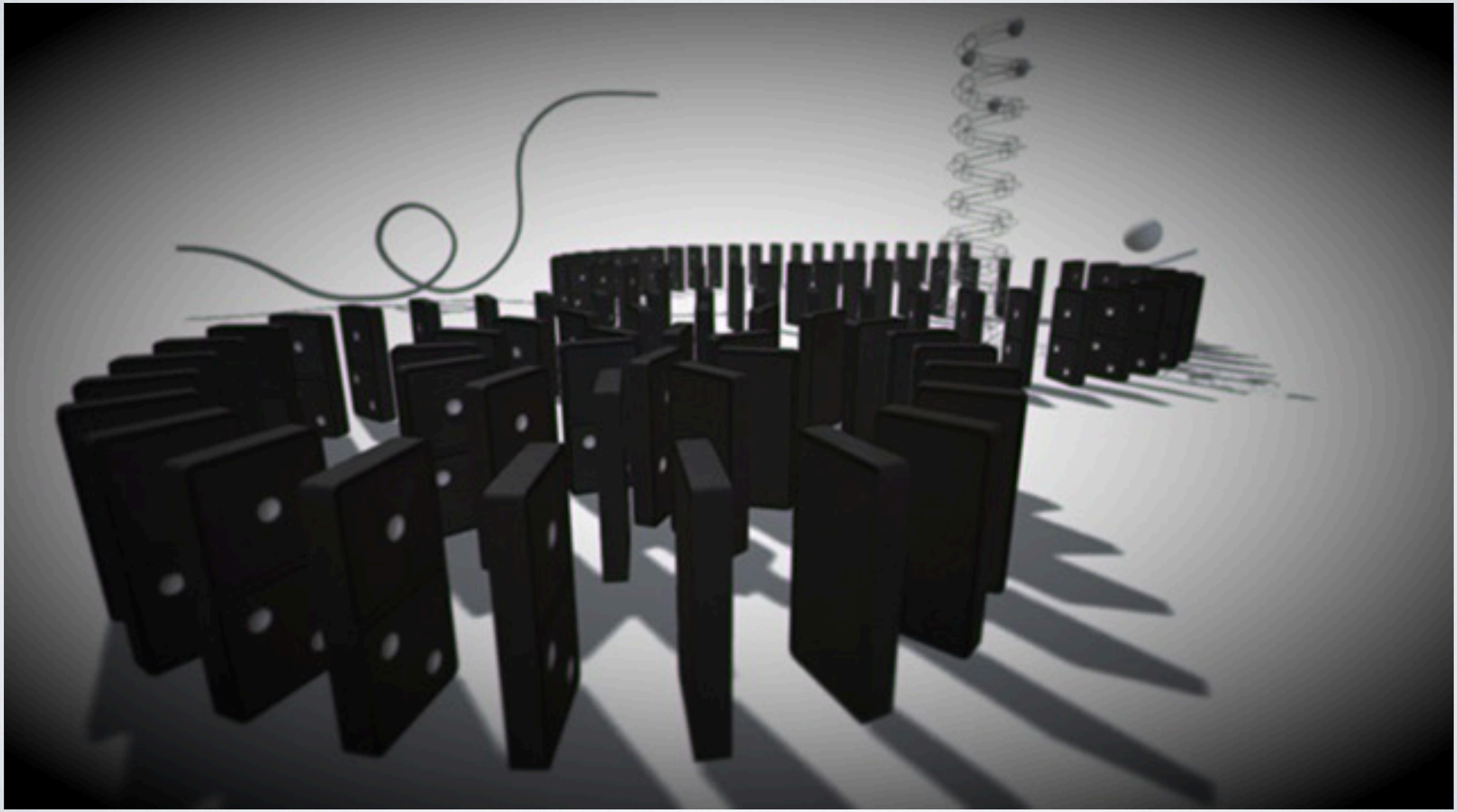
- Image Post-Processing



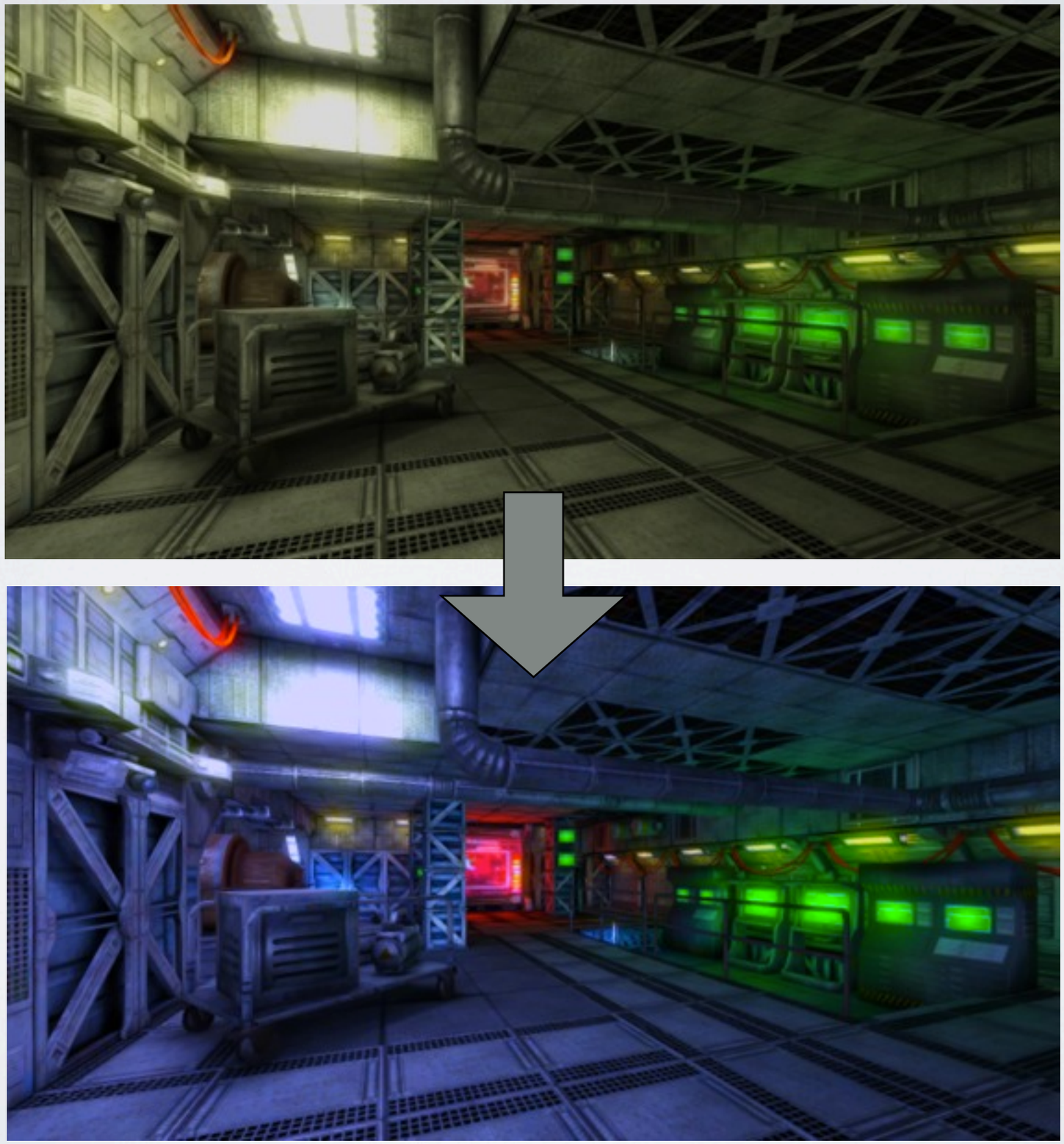



- Vaizdas -> šeideris(-iai) -> pakeistas vaizdas

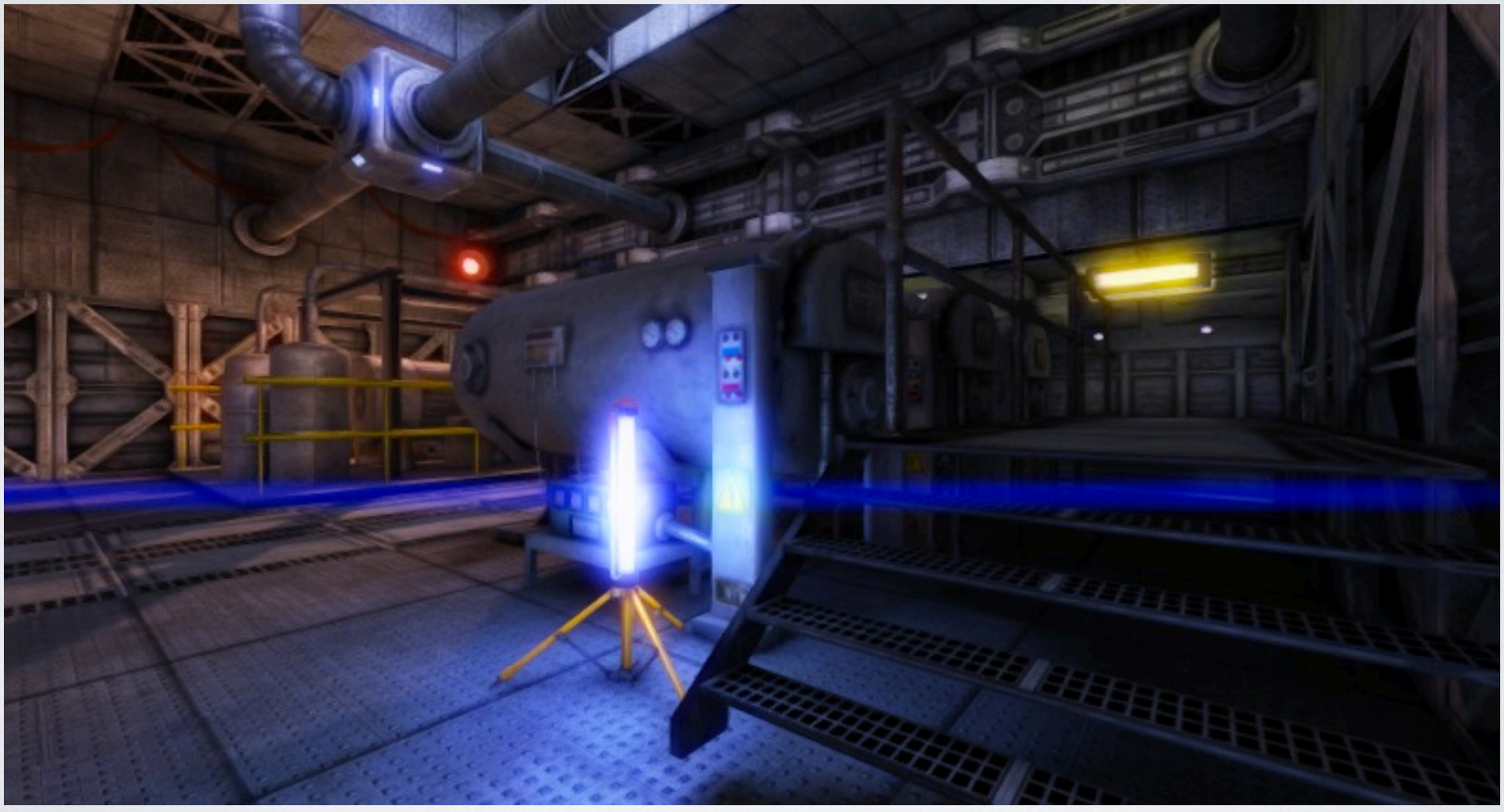
Vignette + chromatic aberration



Color Correction



Bloom + lens flares



Depth of Field

Kita

- GPU galima naudoti nebūtinai *tiesiogiai* vaizdui
- Pamenat, ten tūkstančiai procesorių?
- Jie gi gali ką nors skaičiuot

Ką galima skaičiuot?

- Modeliuoti šviesos sklaidą
  - Sparse voxel octrees, cone tracing
  - Light propagation volumes
- Fizikinius reiškinius
  - Water simulation
  - Massive particles
  - Physics
- ...

Kaip skaičiuoti?

- “Compute” bibliotekos
  - OpenCL
  - Direct3D 11 Compute Shaders
  - OpenGL 4.3 Compute Shaders
  - CUDA, RenderScript, ...
- Galima ir su pixel šeideriais
  - Duomenys ir rezultatai tekstūrose
  - Šeideris kažką suskaičiuoja

Q?

Beje, Unity reikia programuotojų
@aras_p / aras@unity3d.com