The Lightspeed Automatic Interactive Lighting Preview System

Jonathan Ragan-Kelley / *MIT CSAIL*

Charlie Kilpatrick / *ILM*

Brian Smith / *ILM*

Doug Epps / *Tippett Studio*

Paul Green / *MIT CSAIL*

Christophe Héry / *ILM*

Frédo Durand / *MIT CSAIL*

Lighting Design

- **End of pipeline**
  fixed geometry, viewpoint, material
- **Slow feedback**
  10-60 mins to render

Lighting Design

- **End of pipeline**
  fixed geometry, viewpoint, material

- **Slow feedback**
  10-60 mins to render

Goal: Fast Lighting Preview

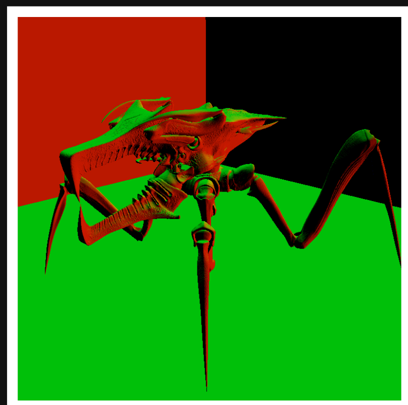
Exploit redundancy between previews

- geometry
- view
- material

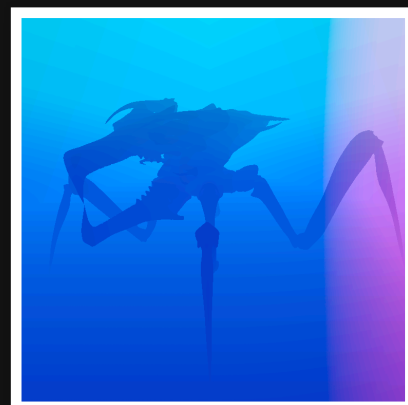
High-Level Approach

- **Precompute deep-framebuffer cache**

e.g.



normal



position

diffuse texture

specular texture

...

- **Preview dynamically on GPU**
  as the user specifies new light parameters

Prior Work

Render Caching

Parameterized Ray Tracing [Séquin & Smyrl 1989]

G-Buffer [Saito & Takahashi 1990]

Fast Relighting Engine [Gershbein & Hanrahan 2000]

- **Precomputed buffer**
  (normals, texture)
  with BRDF*light reevaluated at runtime

- **Shortcomings**
  - **Simplistic shading**
  - **No antialiasing, motion blur, transparency**

Pixar's Lpics

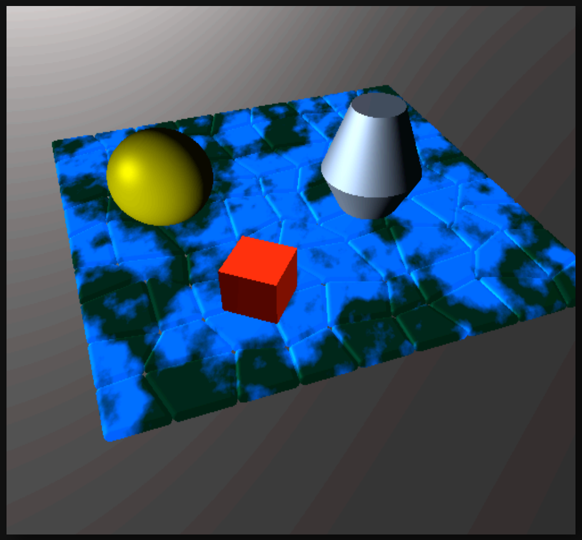
[Pellacini et al. 2005]

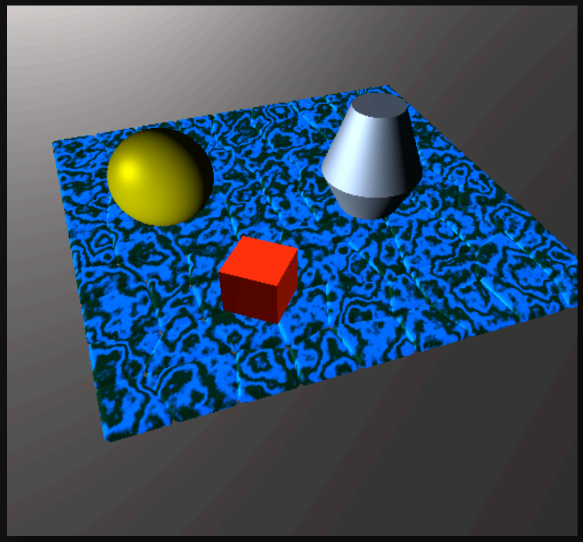
- **Production shaders & lights (RenderMan)**
- **Exploit programmable GPUs**
- **No antialiasing, transparency**
- **Requires extra shader-writing work**
  - **(final) RenderMan** version + extra caching code
  - **GPU preview** version

Specializing Shaders

[Guenter et al. 1995]

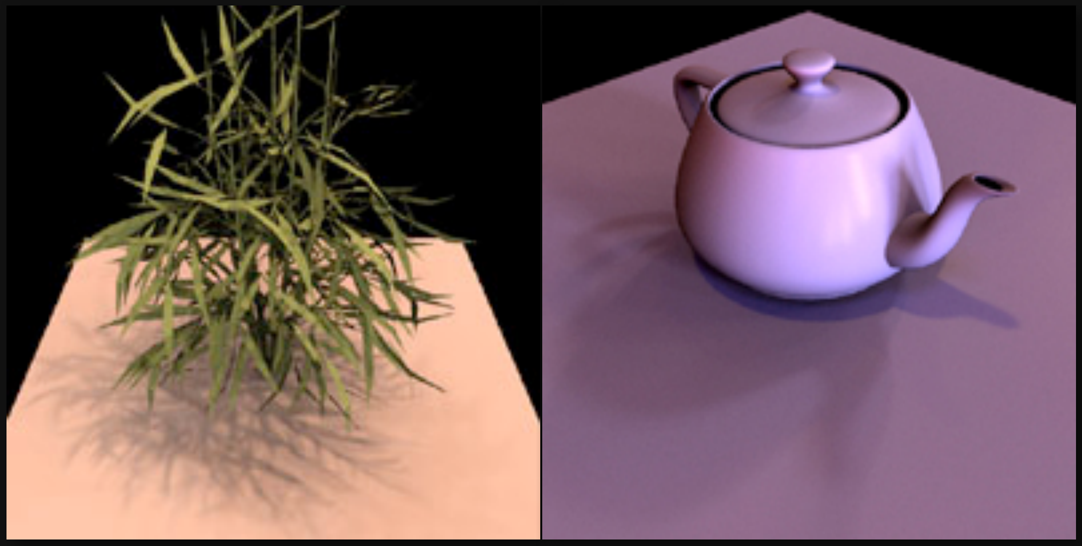
- **Given dynamic parameters, *automatically* split shader code into:**
  - **static**
  - **dynamic**
- **Simple language**
  no control flow
- **Simple shaders**





Precomputed Radiance Transfer

e.g. Sloan et al. 2002, Ng et al. 2003



Inappropriate for our application

- Large precomputation cost
- Largely ignore light functions
- Limitations on shading models

Our Design Goals

1. **High-performance preview**
   - **Low-latency feedback**
   - **Fast initial precomputation**

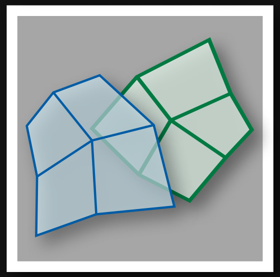
2. **Seamless integration**
   - **Same input:** RenderMan scene & shaders
   - **Same output:** high quality image

3. **Ease of implementation & maintenance**

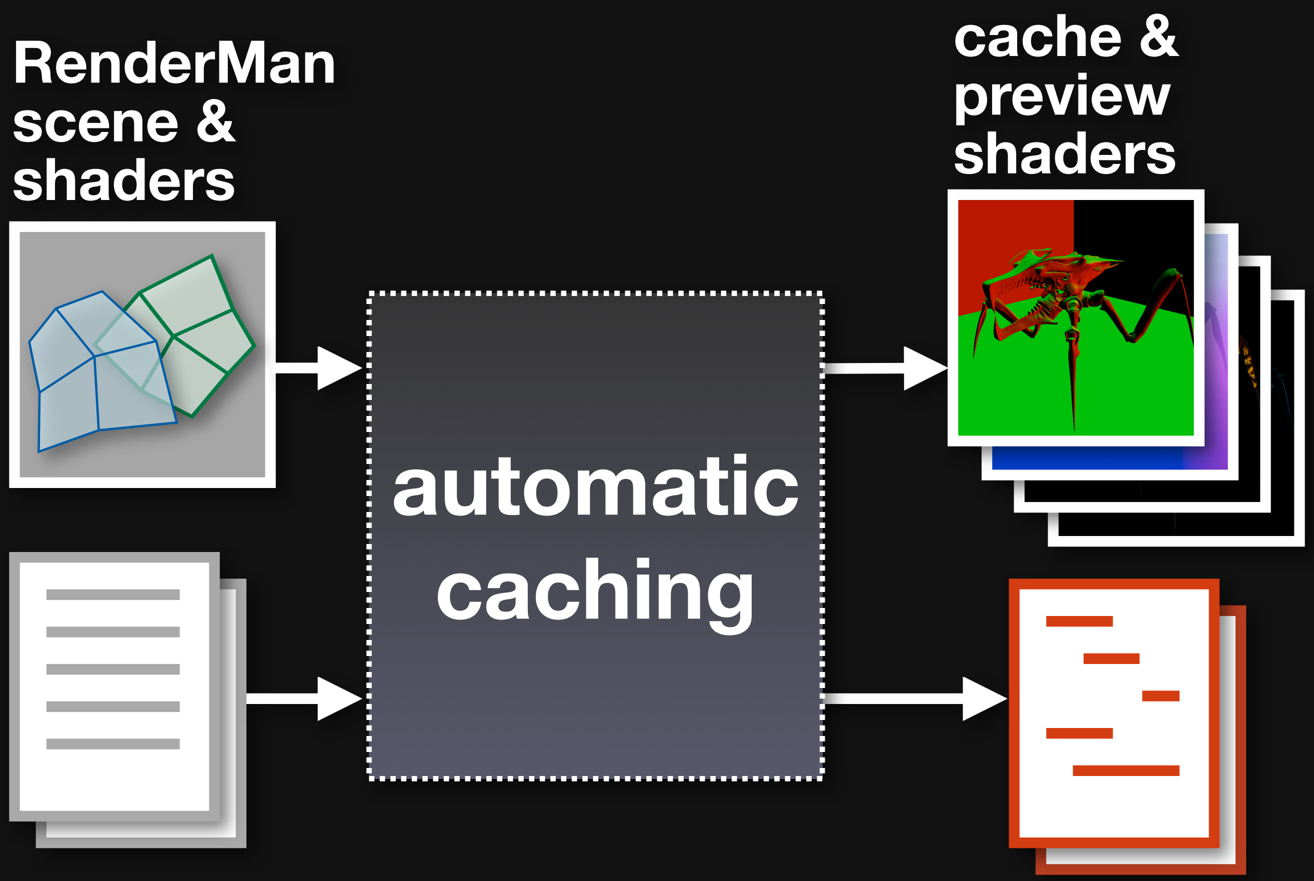
System Overview

System Overview

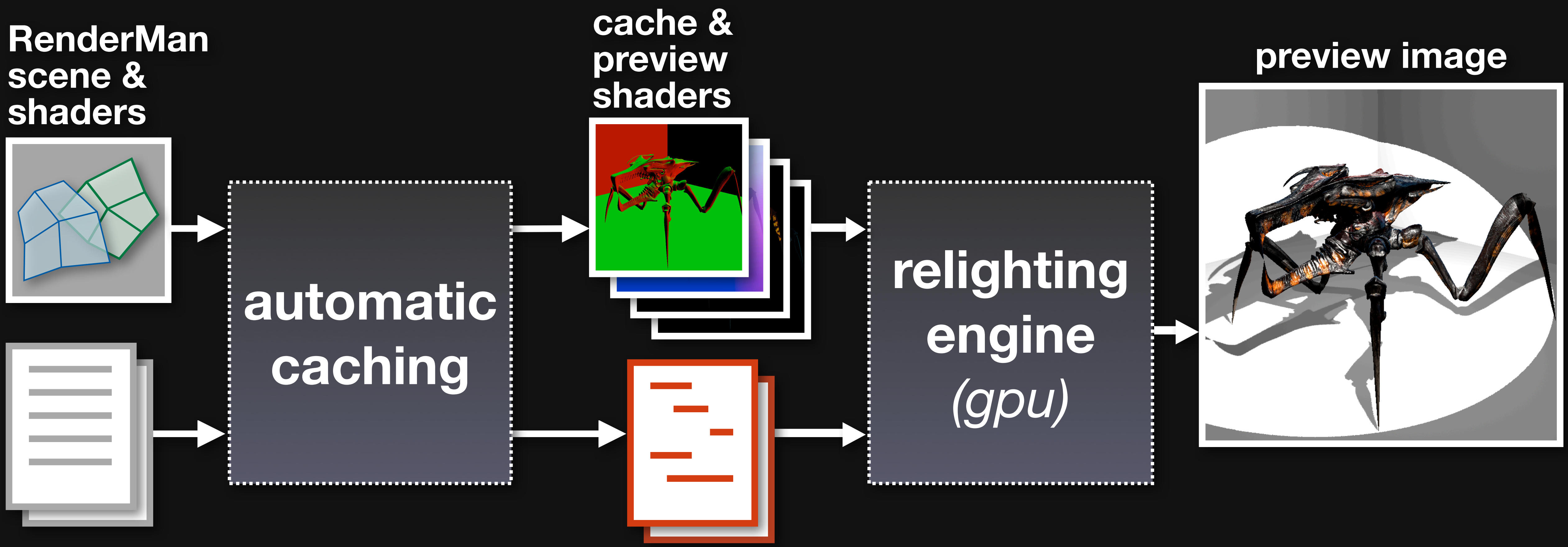
RenderMan
scene &
shaders



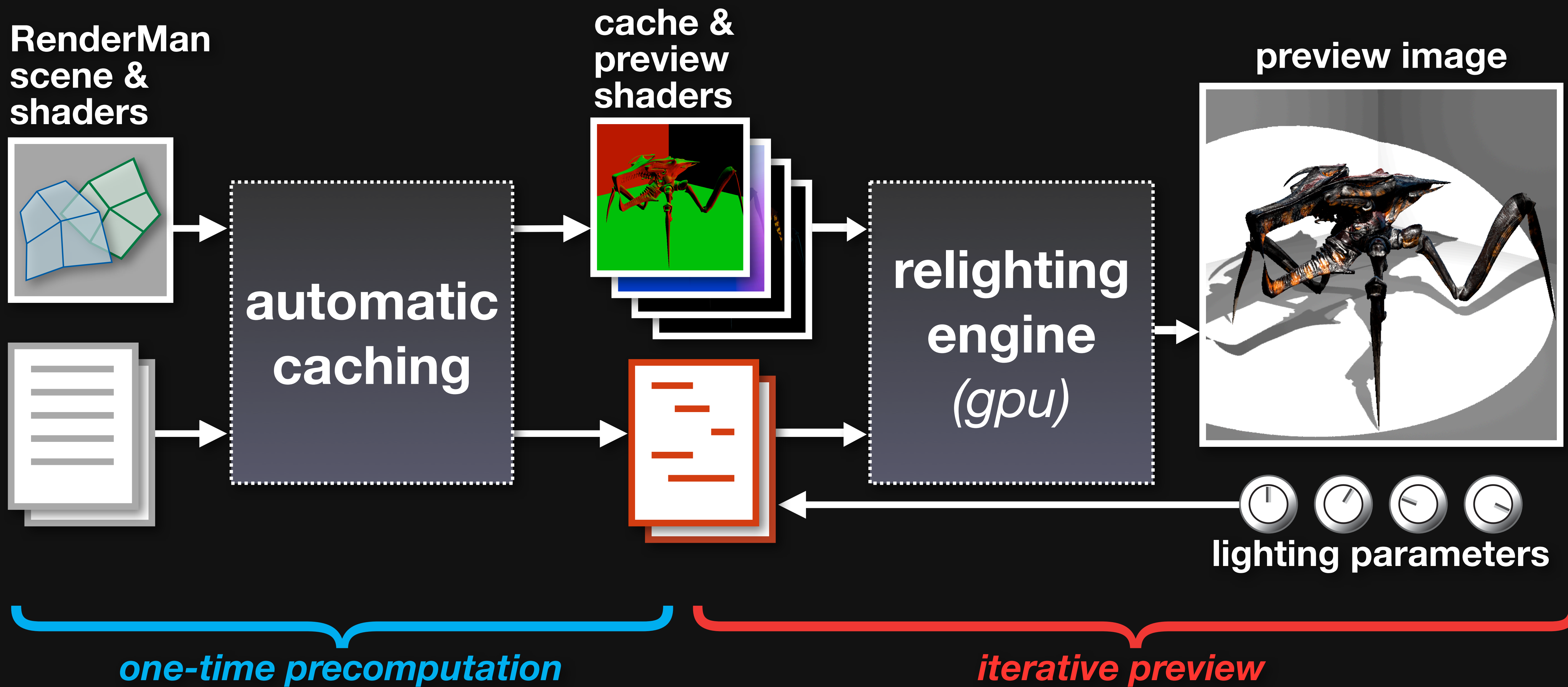
System Overview



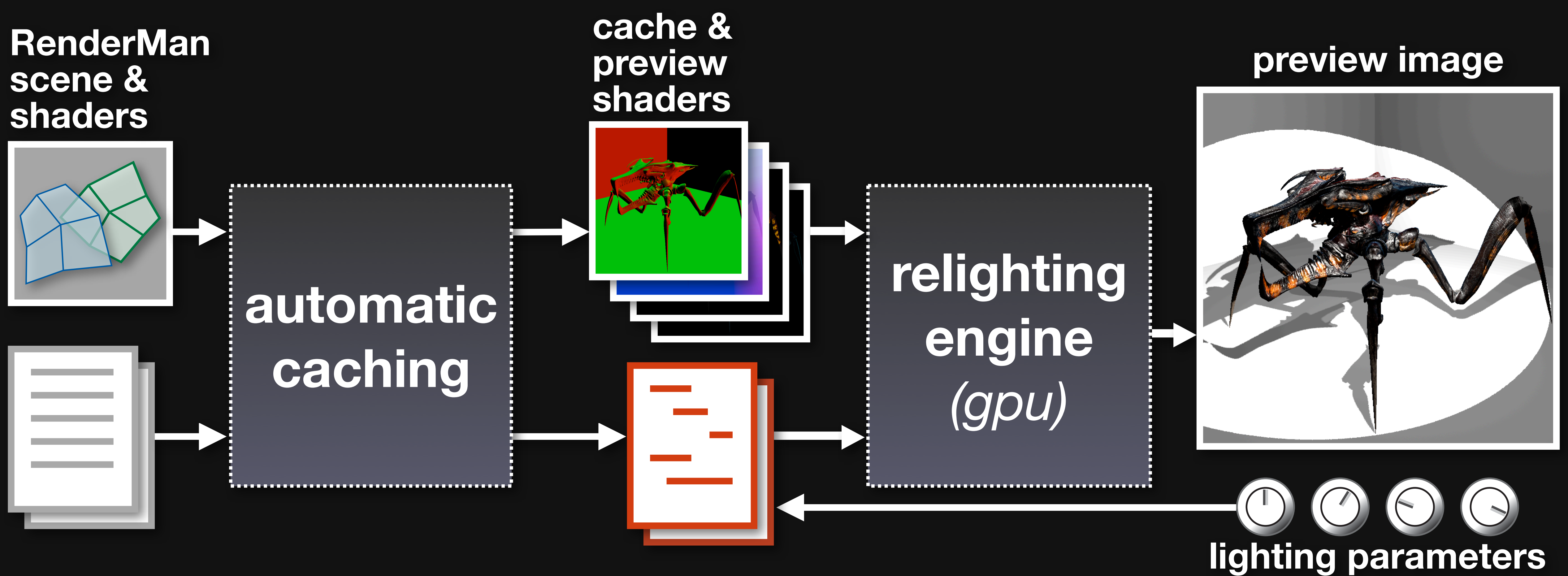
System Overview



System Overview



System Overview

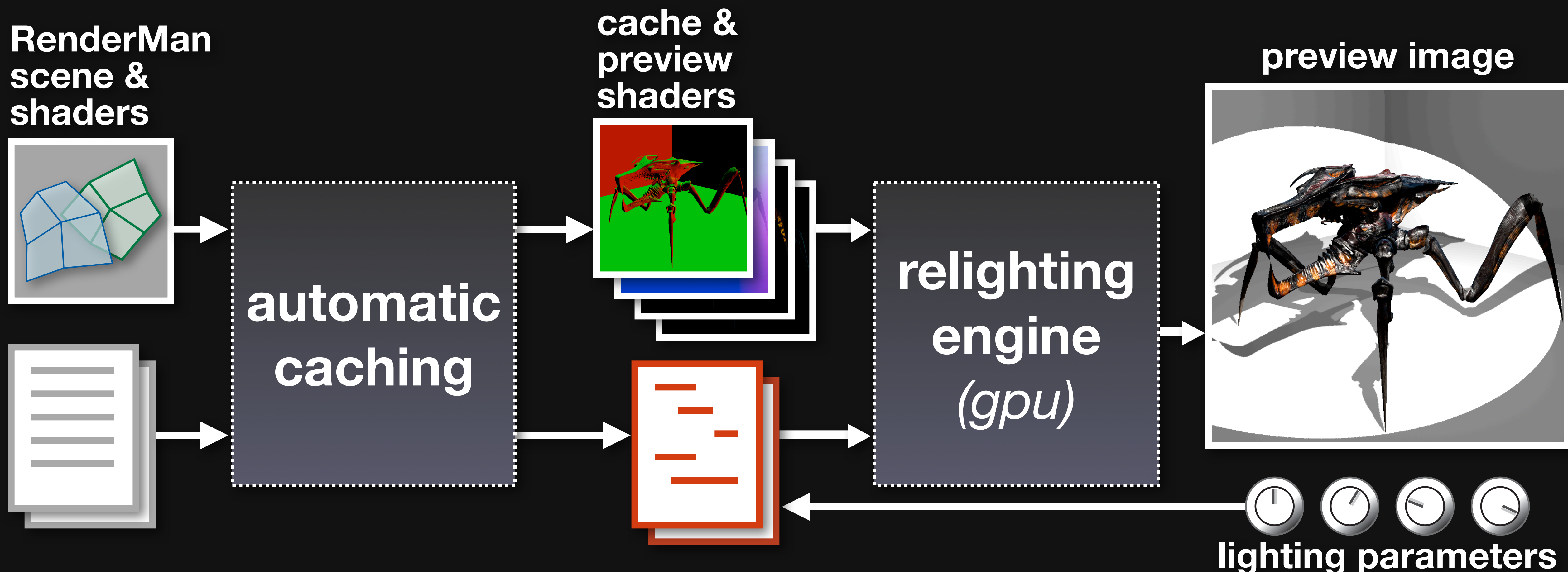


Automatic

for real production scenes

- specialization & translation
- cache compression

System Overview



Automatic

for real production scenes

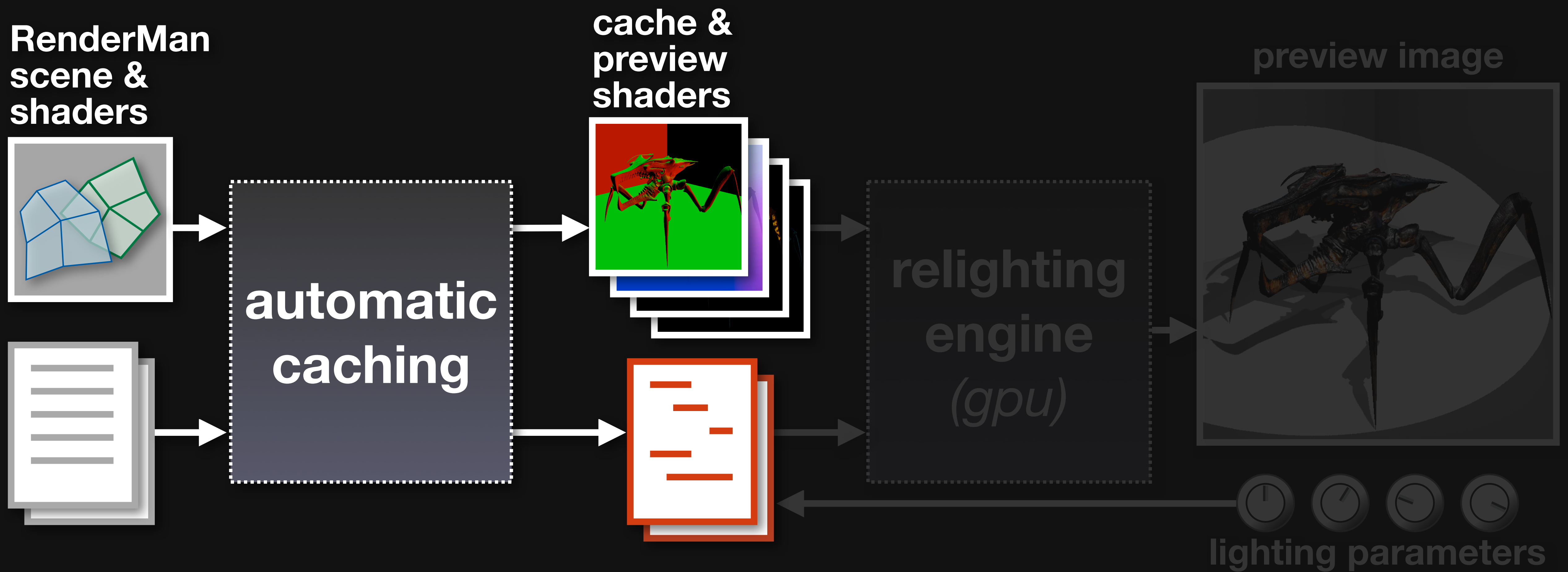
- specialization & translation
- cache compression

High fidelity

antialiasing, motion blur, transparency

- indirect framebuffer

Automatic Caching



Program Analysis

Program Analysis

```
mySurface(color c) {
  albedo = c*Texture(U,V);
  shade = Light(L)*BRDF(N,L);
  return albedo*shade;
}
```

Program Analysis

```
mySurface(color c) {
  albedo = c*Texture(U,V);
  shade = Light(L)*BRDF(N,L);
  return albedo*shade;
}
```

dynamic

what part of the code depends on the lighting parameter *L*?

Program Analysis

```
mySurface(color c) {
  albedo = c*Texture(U,V);
  shade = Light(L)*BRDF(N,L);
  return albedo*shade;
}
```

dynamic
static

everything else is static

Program Analysis

```
mySurface(color c) {
  albedo = c*Texture(U,V);
  shade = Light(L)*BRDF(N,L);
  return albedo*shade;
}
```

dynamic
static
cache

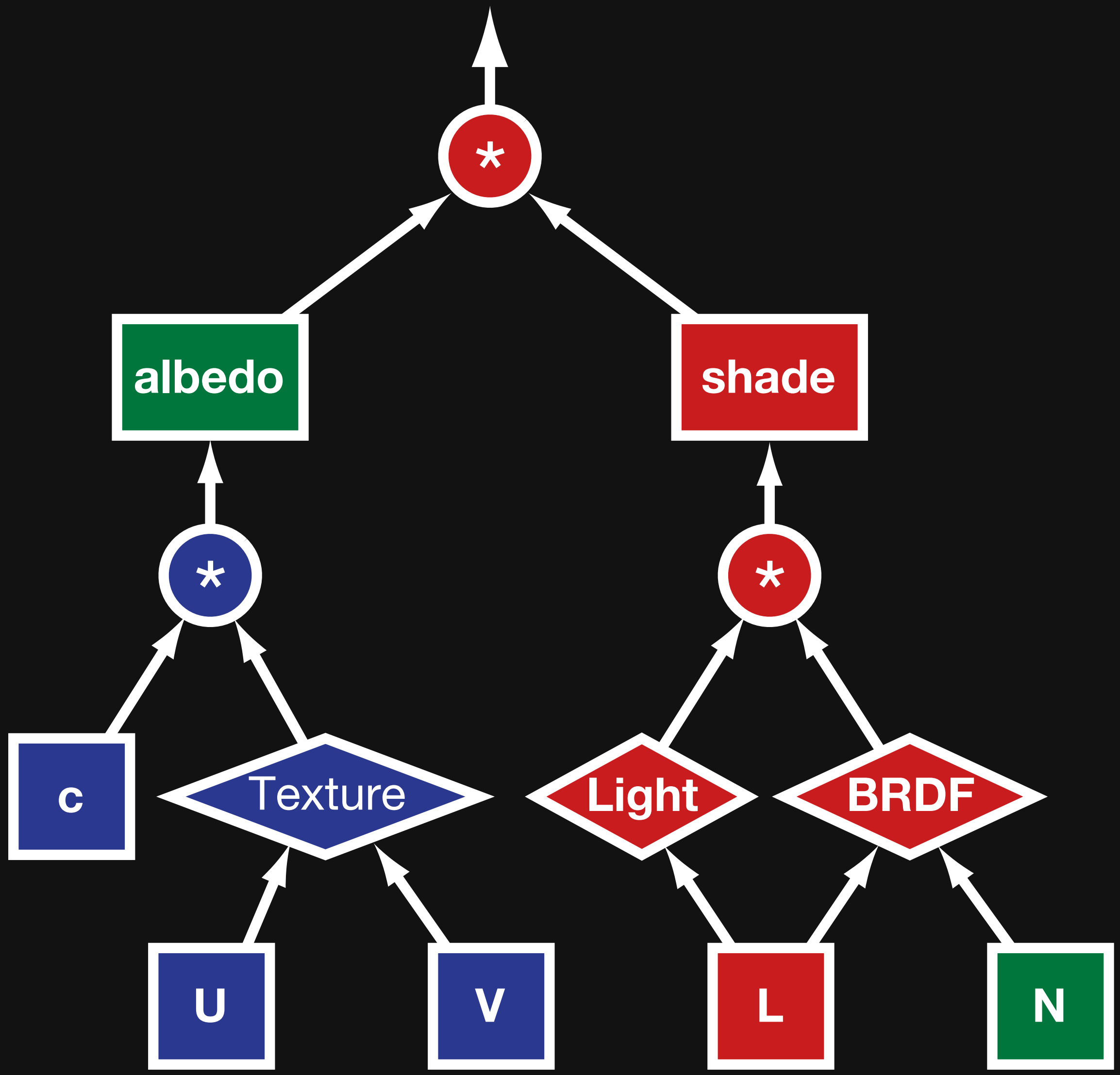
values at the boundary are cached for reevaluation

Program Analysis

```
mySurface(color c) {
  albedo = c*Texture(U,V);
  shade = Light(L)*BRDF(N,L);
  return albedo*shade;
}
```

Program Analysis

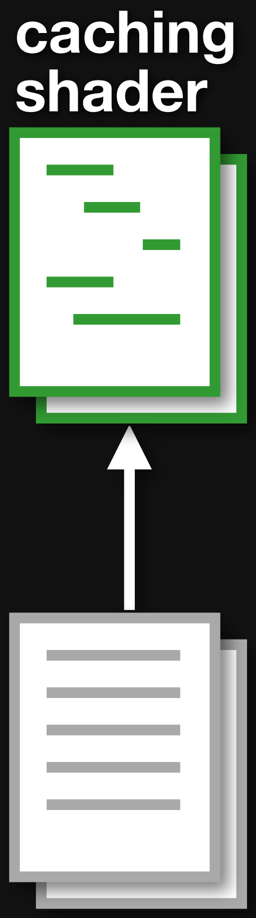
```
mySurface(color c) {
  albedo = c*Texture(U,V);
  shade = Light(L)*BRDF(N,L);
  return albedo*shade;
}
```



Deep Framebuffer Generation

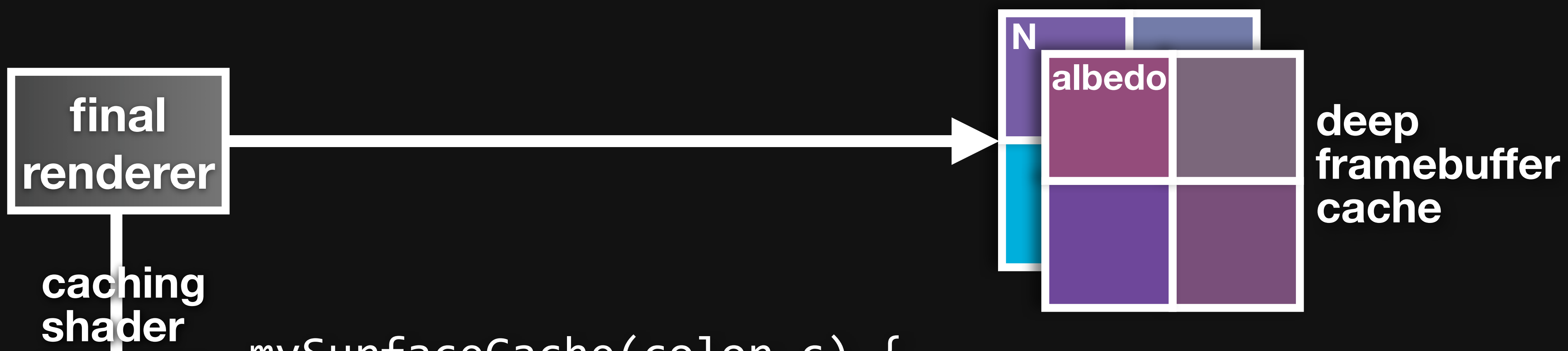
```
mySurface(color c) {
  albedo = c*Texture(U,V);
  shade = Light(L)*BRDF(N,L);
  return albedo*shade;
}
```

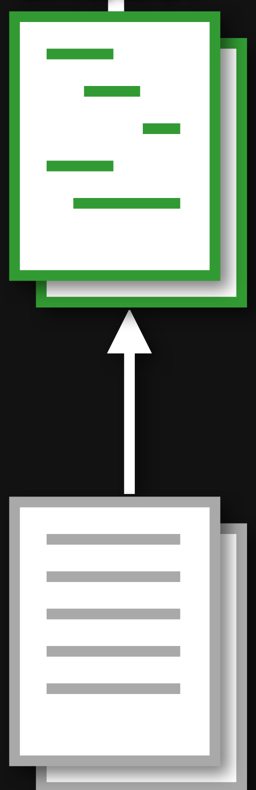
Deep Framebuffer Generation




```
mySurfaceCache(color c) {
  albedo = c*Texture(U,V);
  cache(N);
  cache(albedo);
}
```

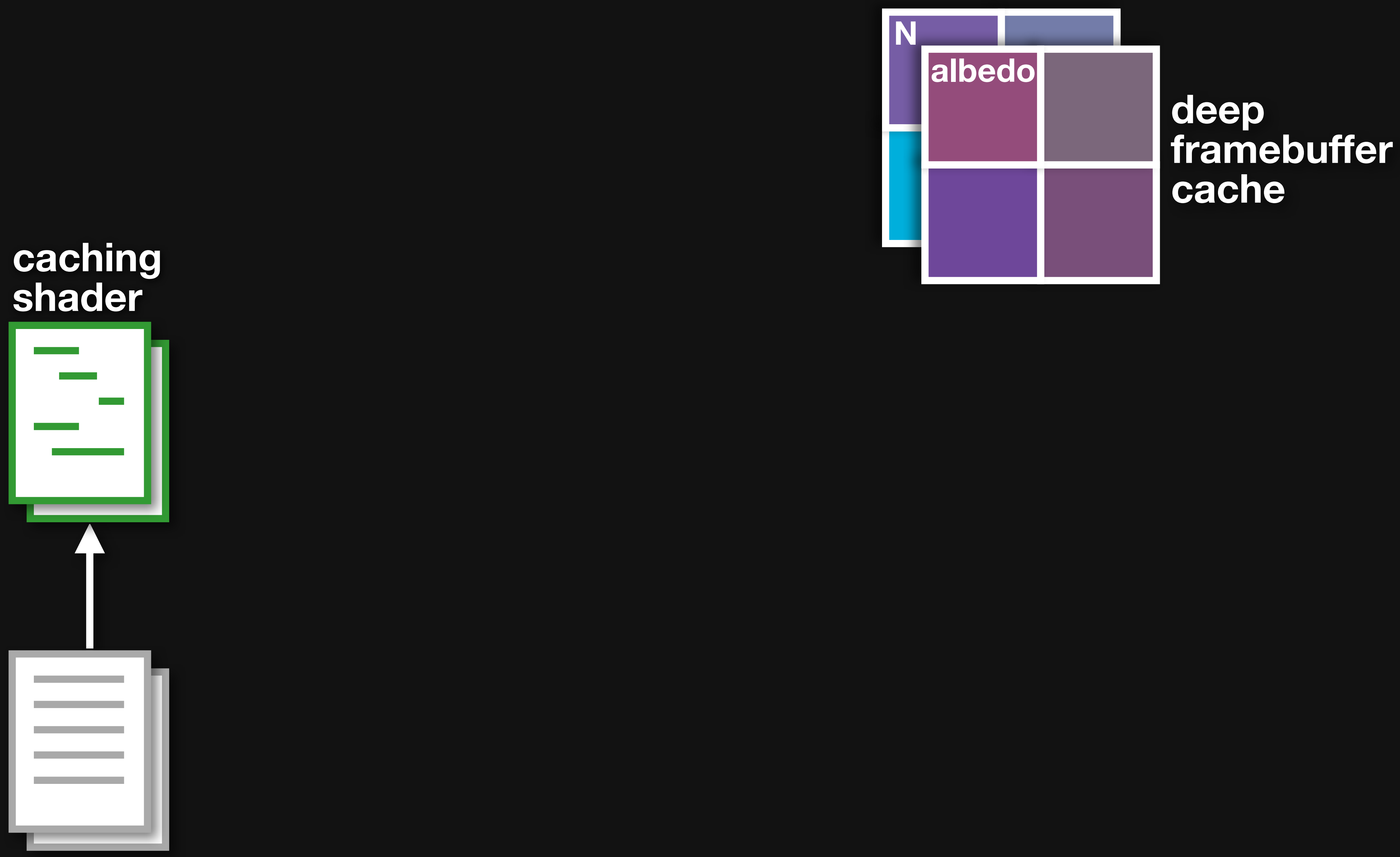
Deep Framebuffer Generation



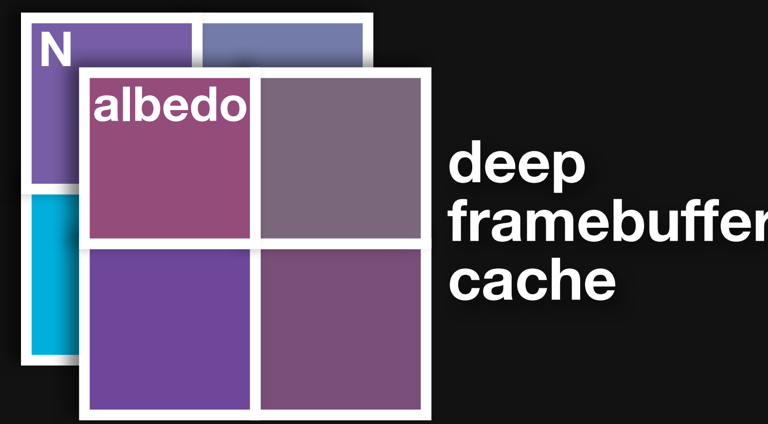


```
mySurfaceCache(color c) {
  albedo = c*Texture(U,V);
  cache(N);
  cache(albedo);
}
```

Deep Framebuffer Generation

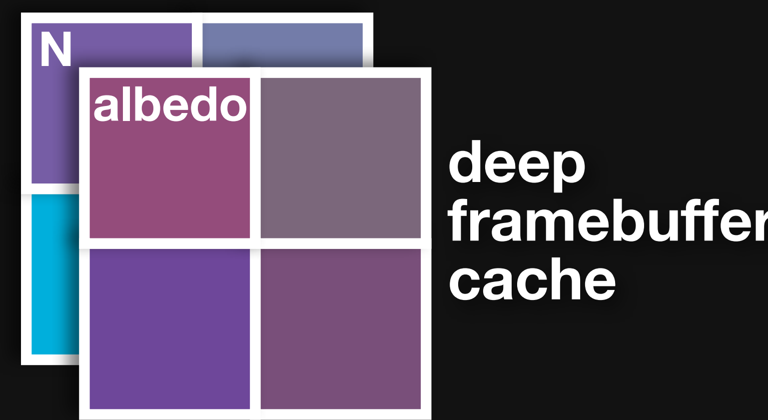


Deep Framebuffer Generation



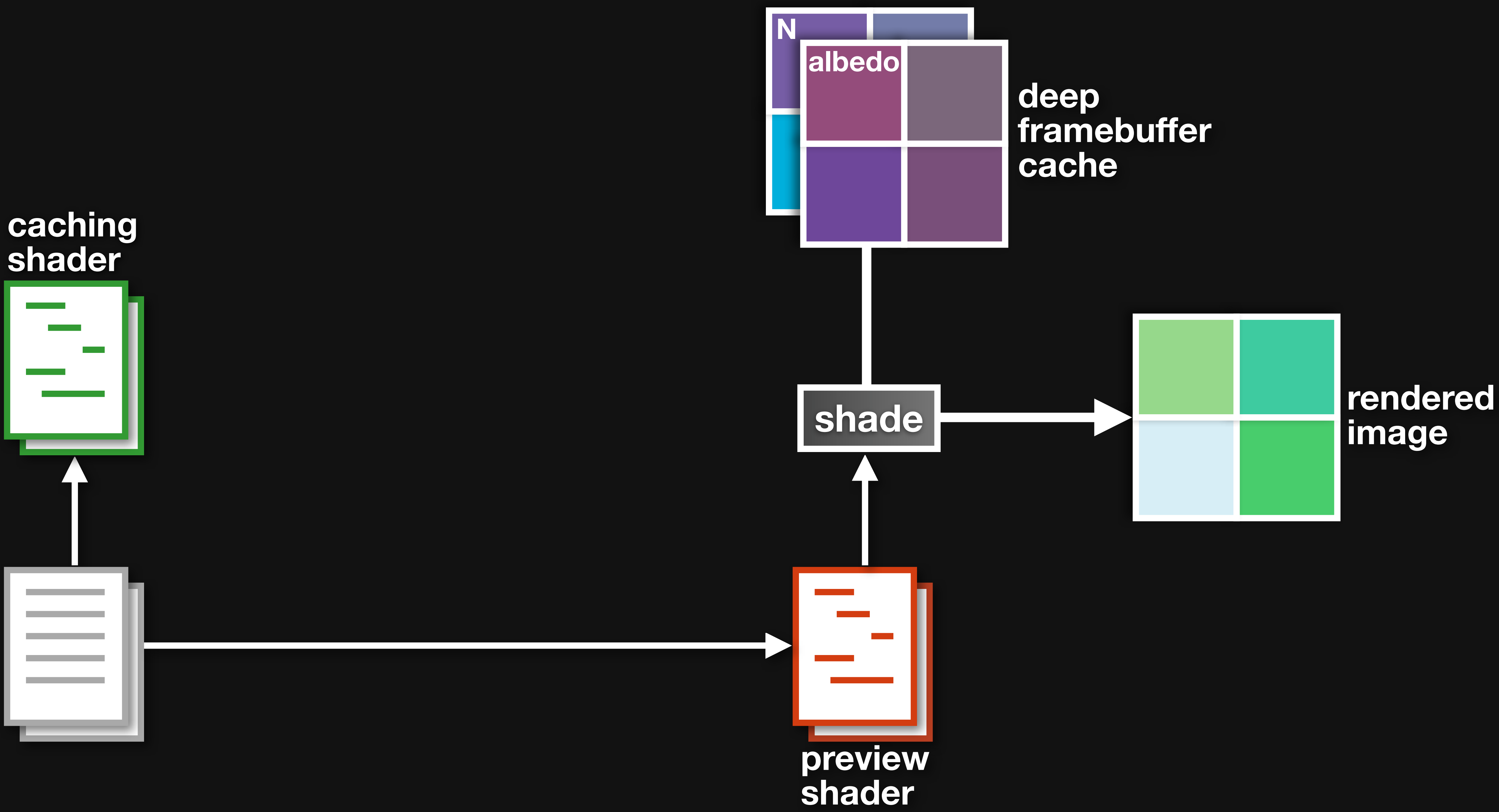

```
mySurfaceDynamic(vector L) {
  N = lookup();
  albedo = lookup();
  shade = Light(L)*BRDF(N,L);
  return albedo*shade;
}
```

Deep Framebuffer Generation



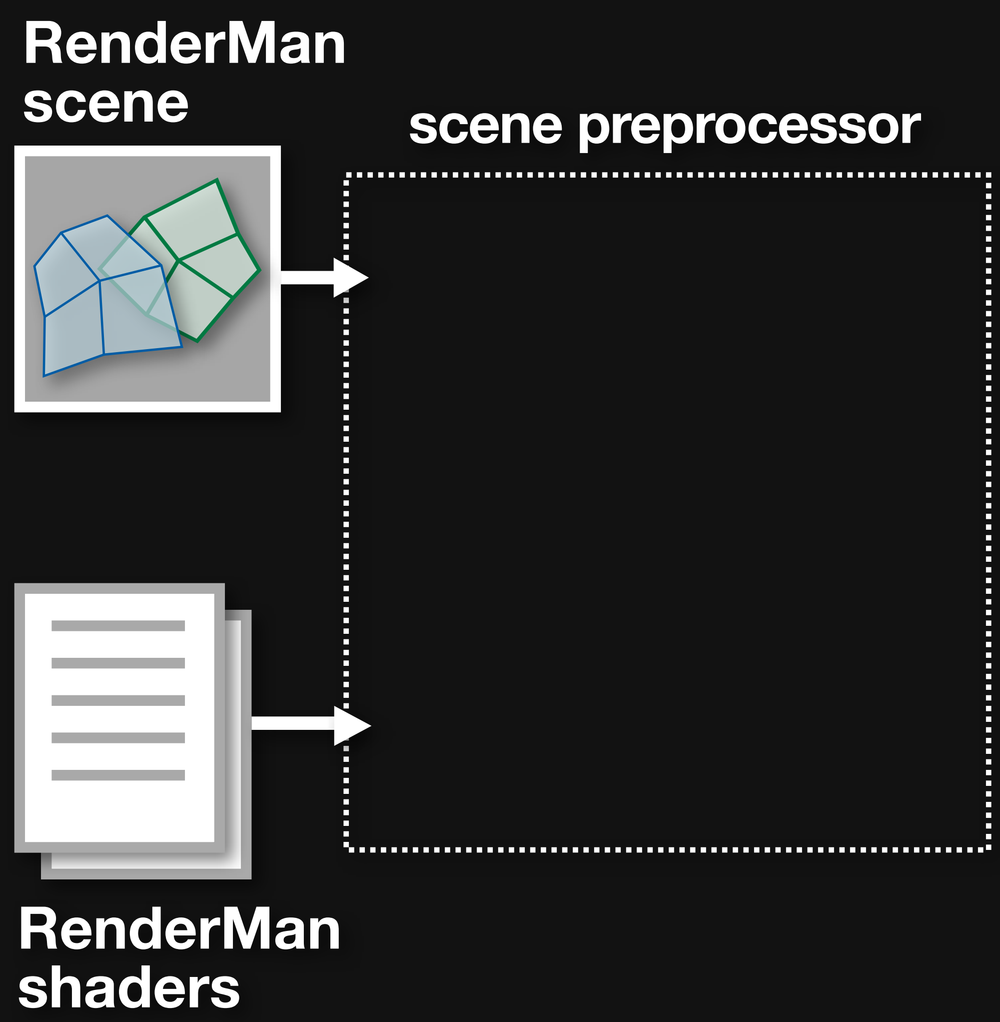

```
mySurfaceDynamic(vector L) {
  N = lookup();
  albedo = lookup();
  shade = Light(L)*BRDF(N,L);
  return albedo*shade;
}
```

Deep Framebuffer Generation

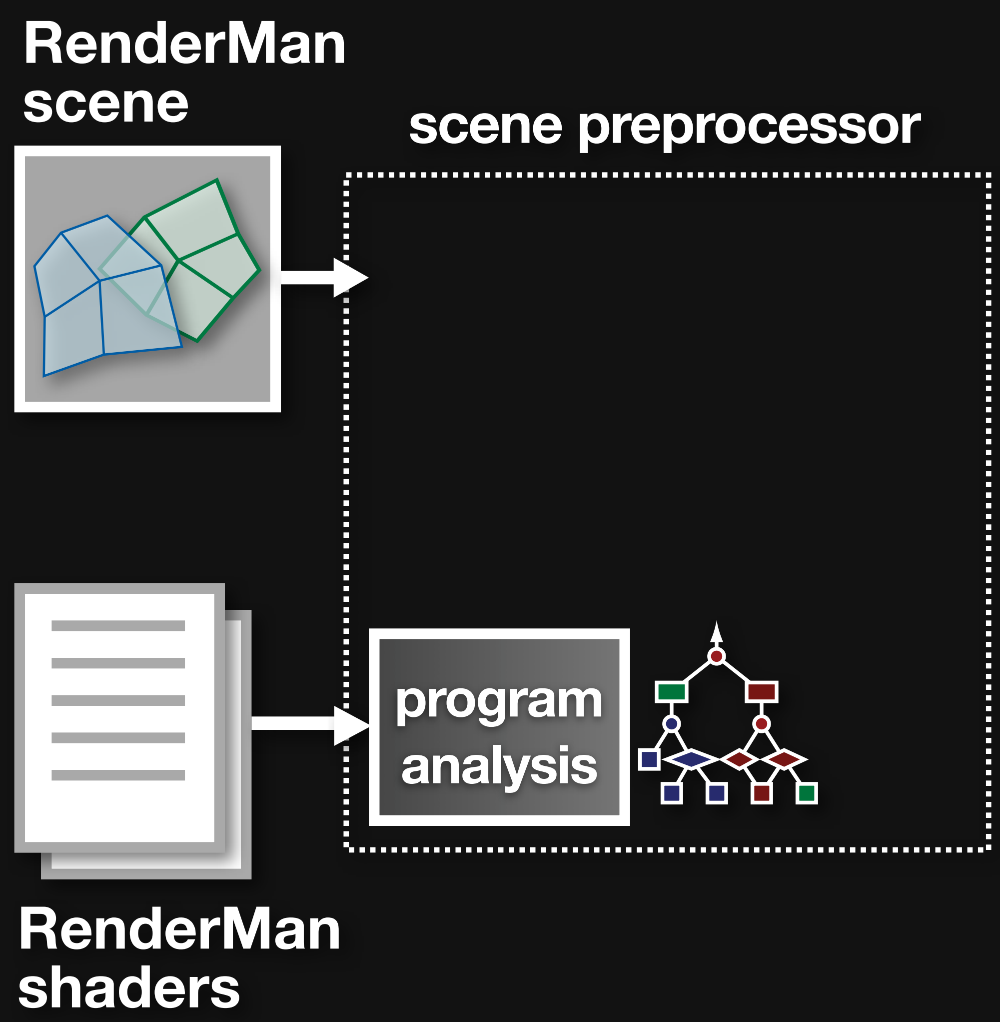


Automatic Caching

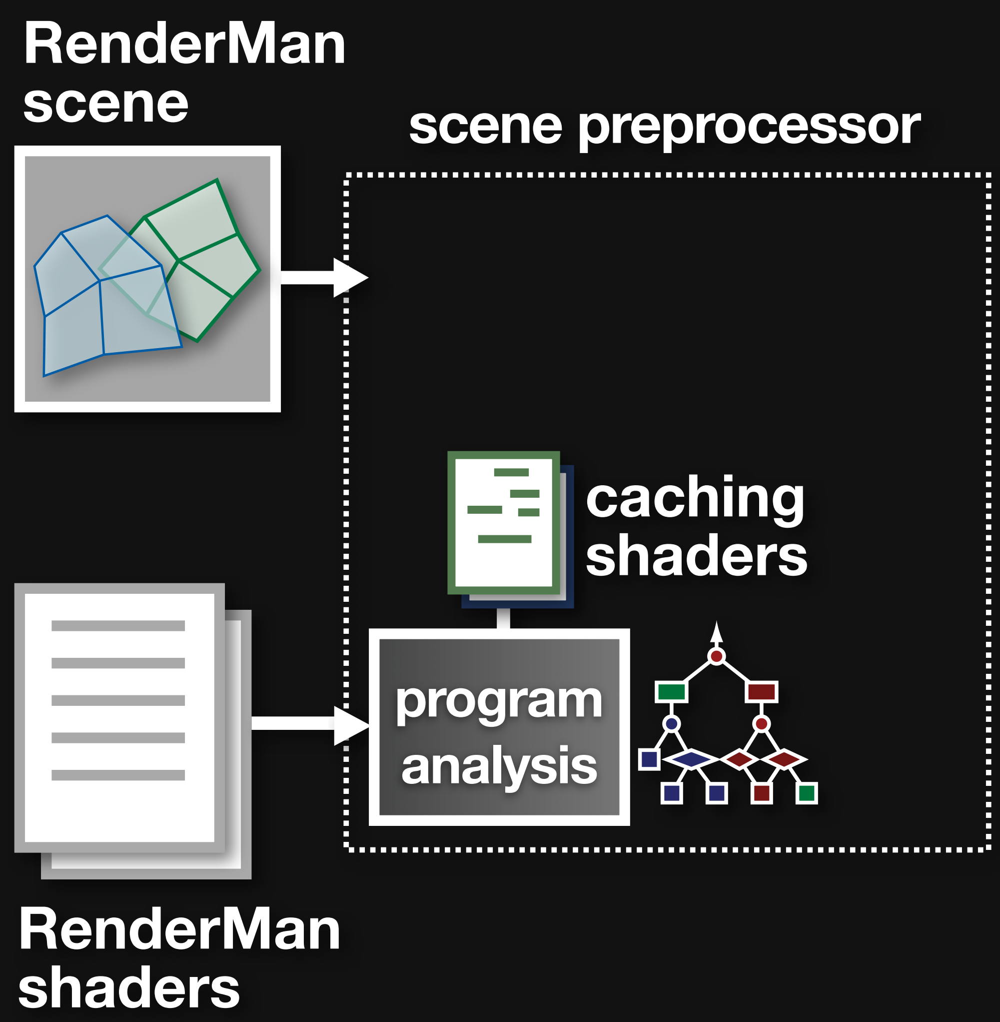
Automatic Caching



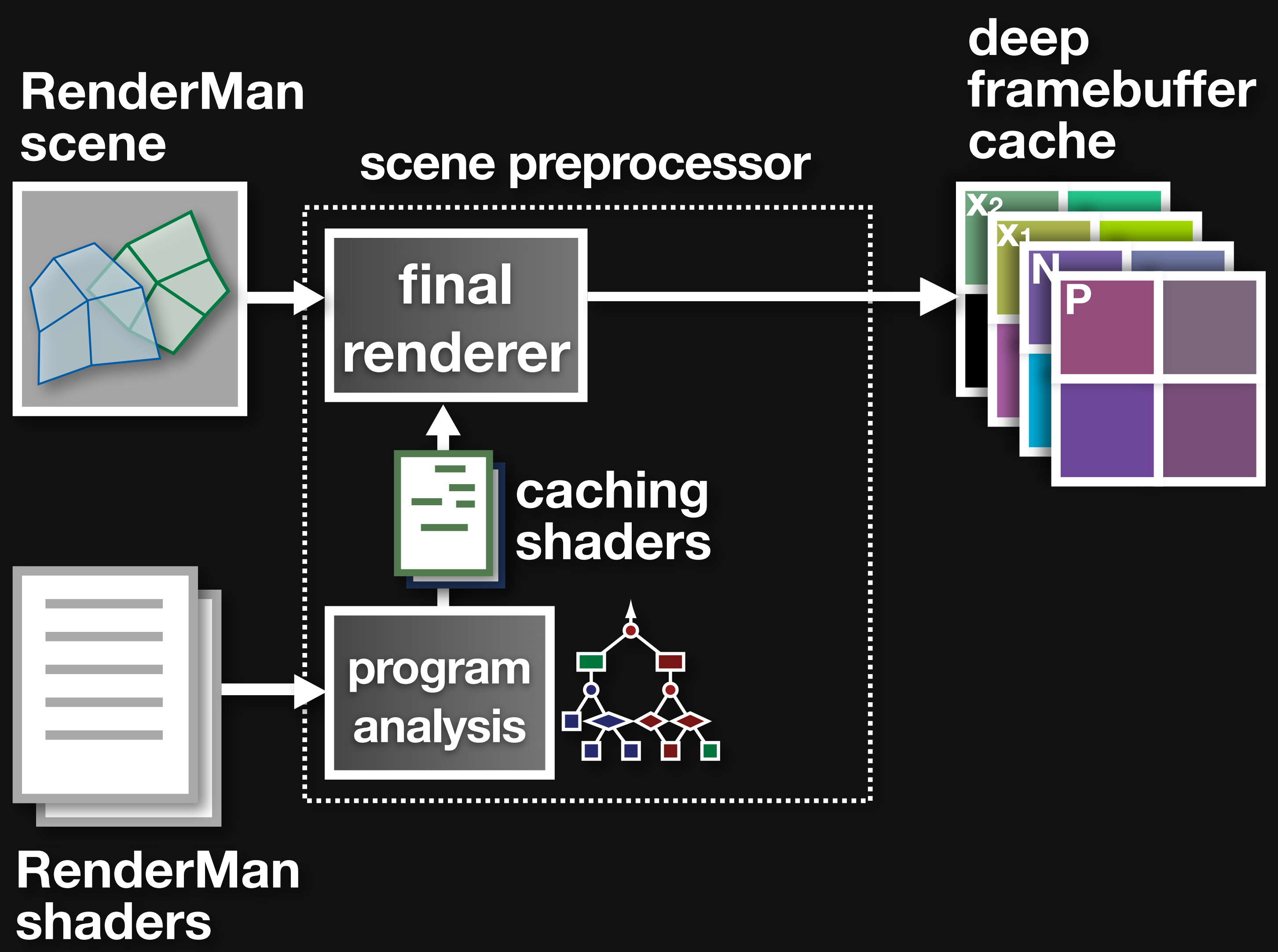
Automatic Caching



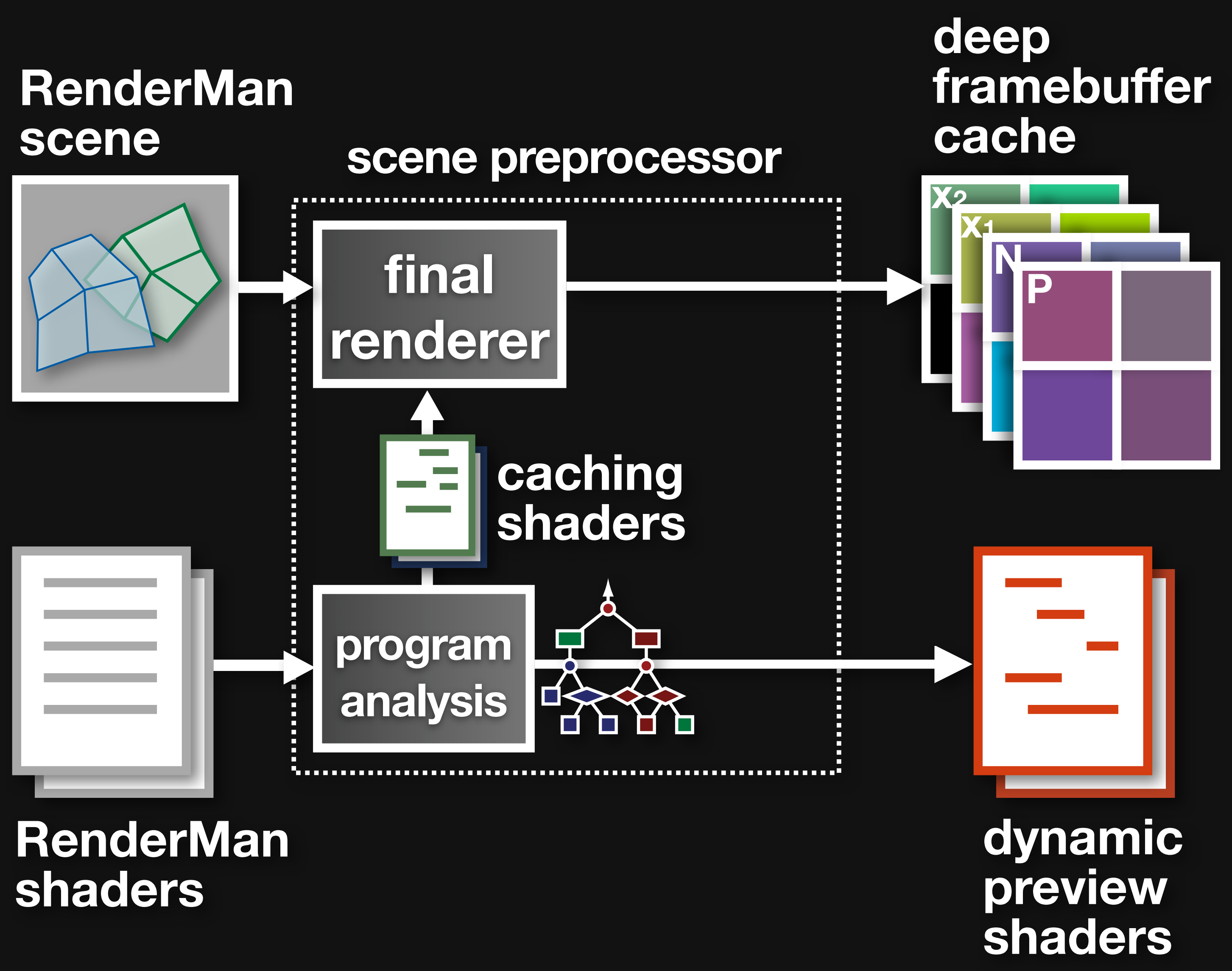
Automatic Caching



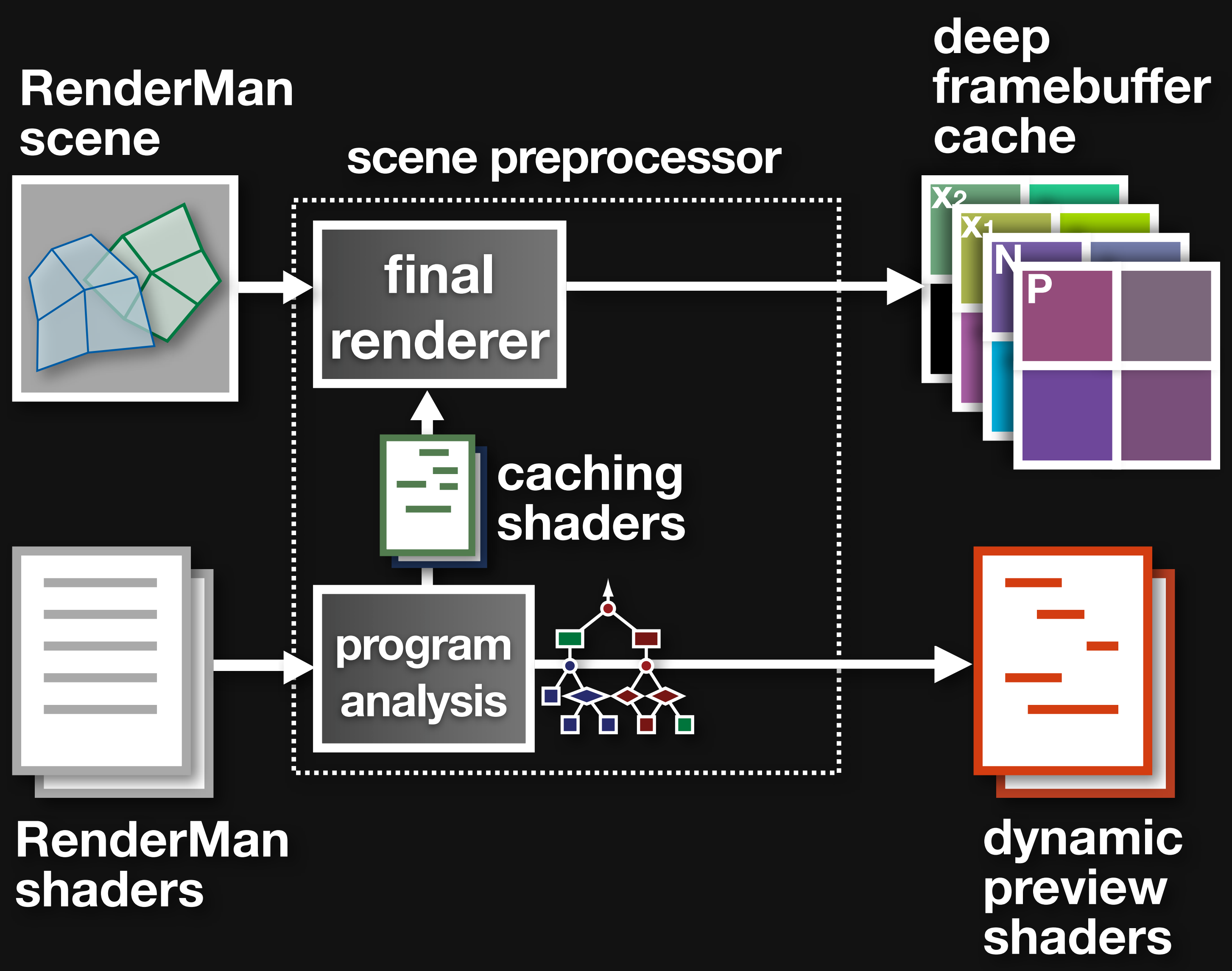
Automatic Caching



Automatic Caching



Automatic Caching



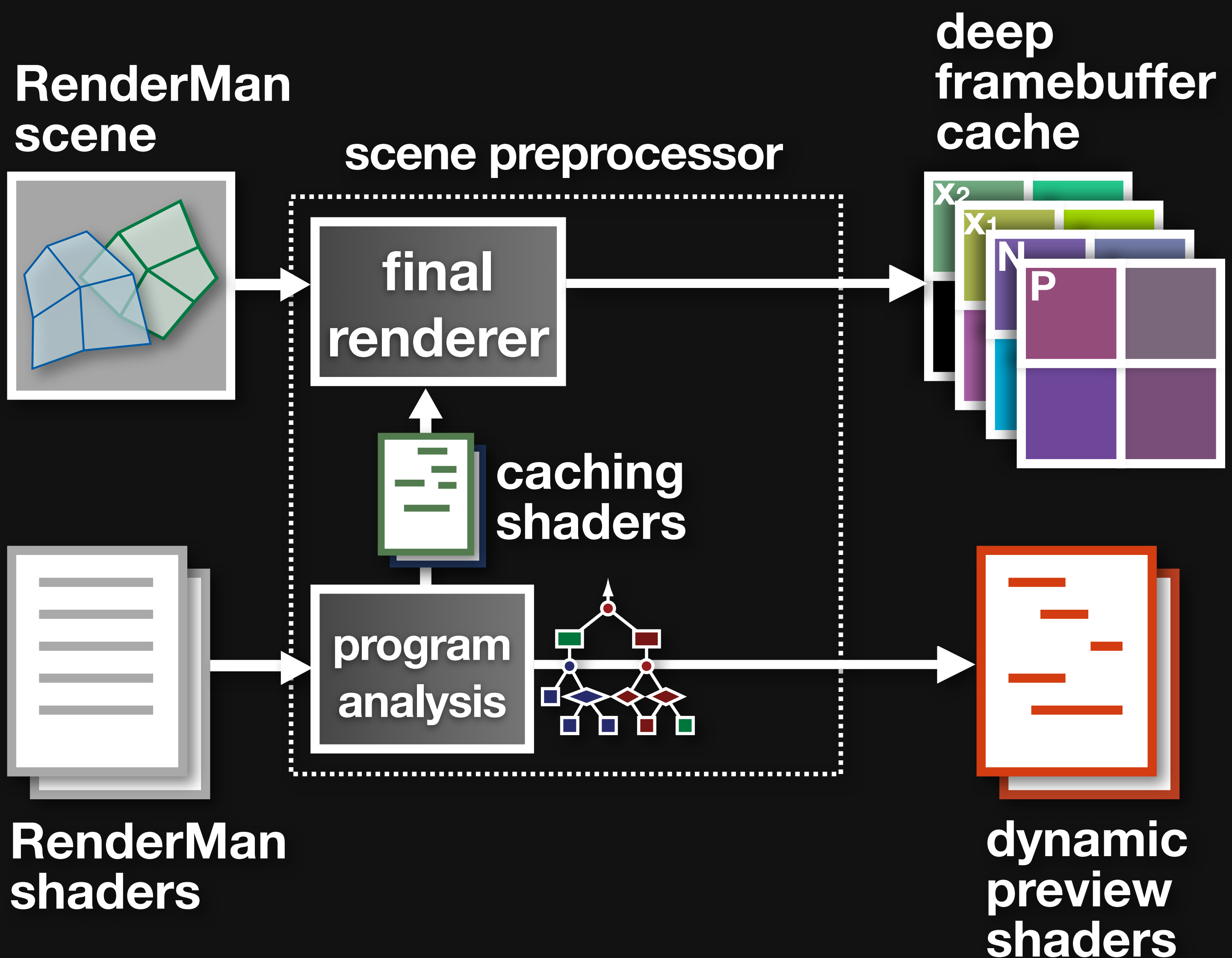
challenge: **cache size**

program analysis is **conservative**

generates **many cached terms** (hundreds)

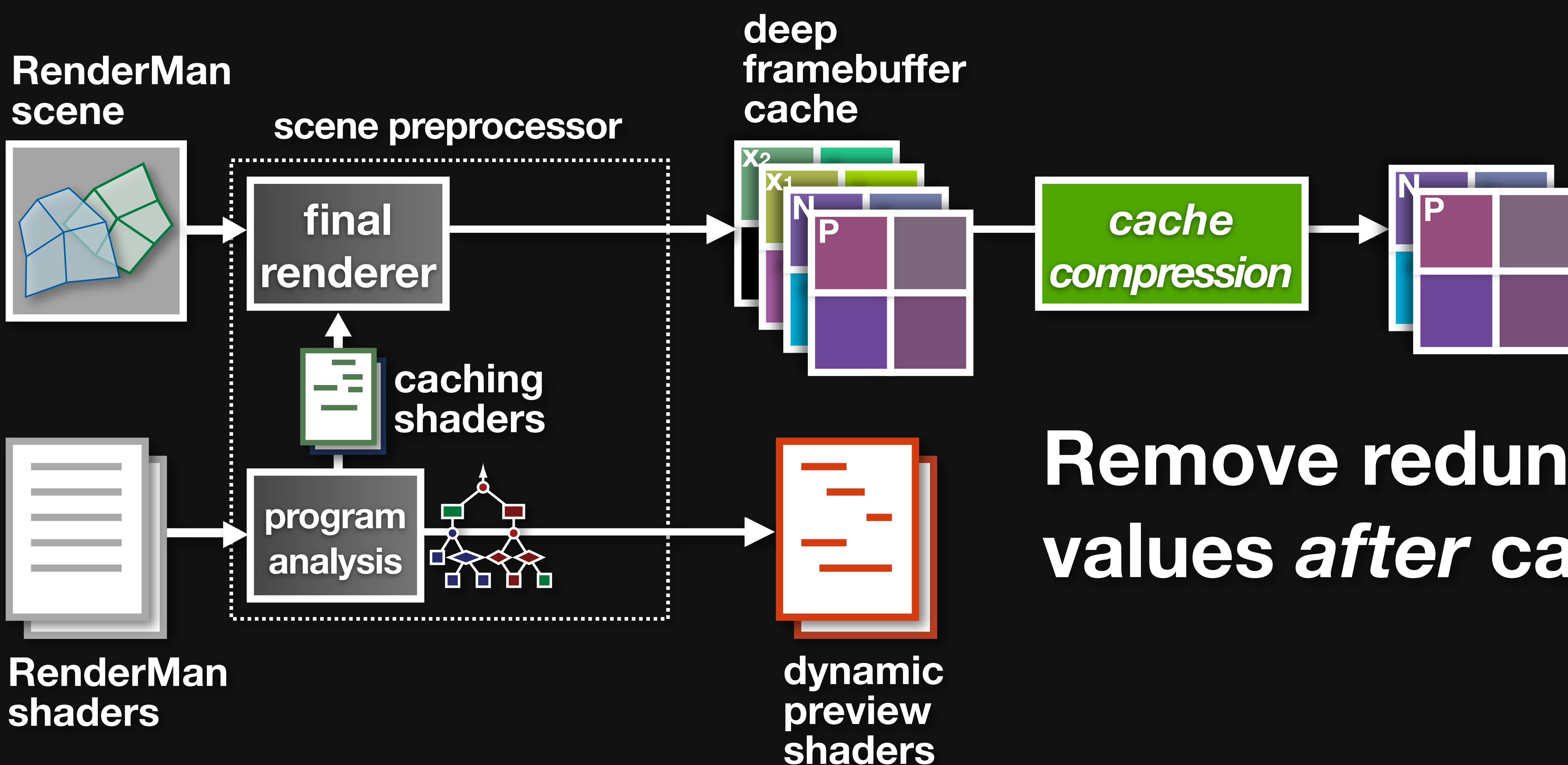
large caches (gigabytes)

Cache Compression



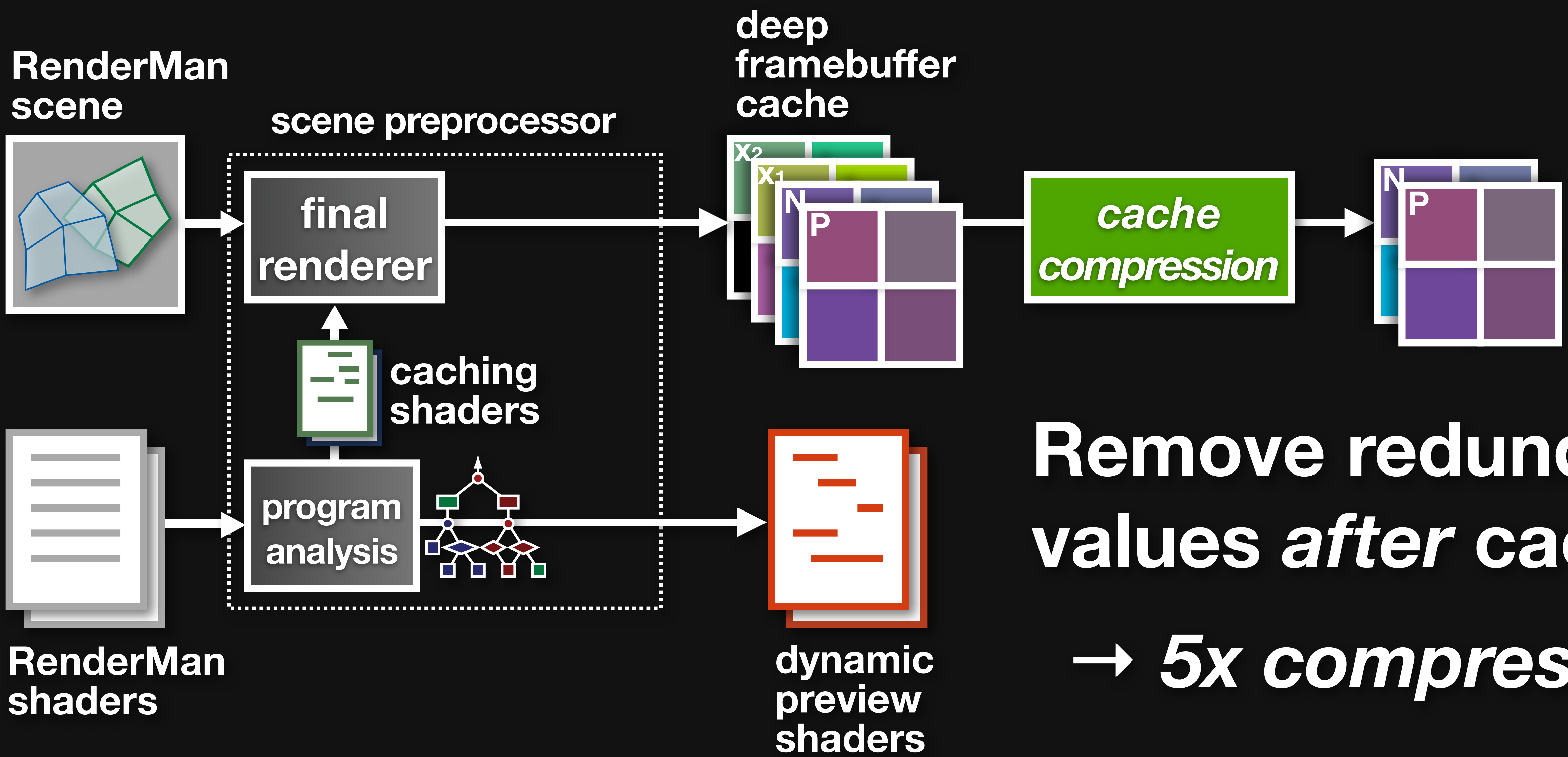
Remove redundant values *after* caching

Cache Compression



Remove redundant values *after* caching

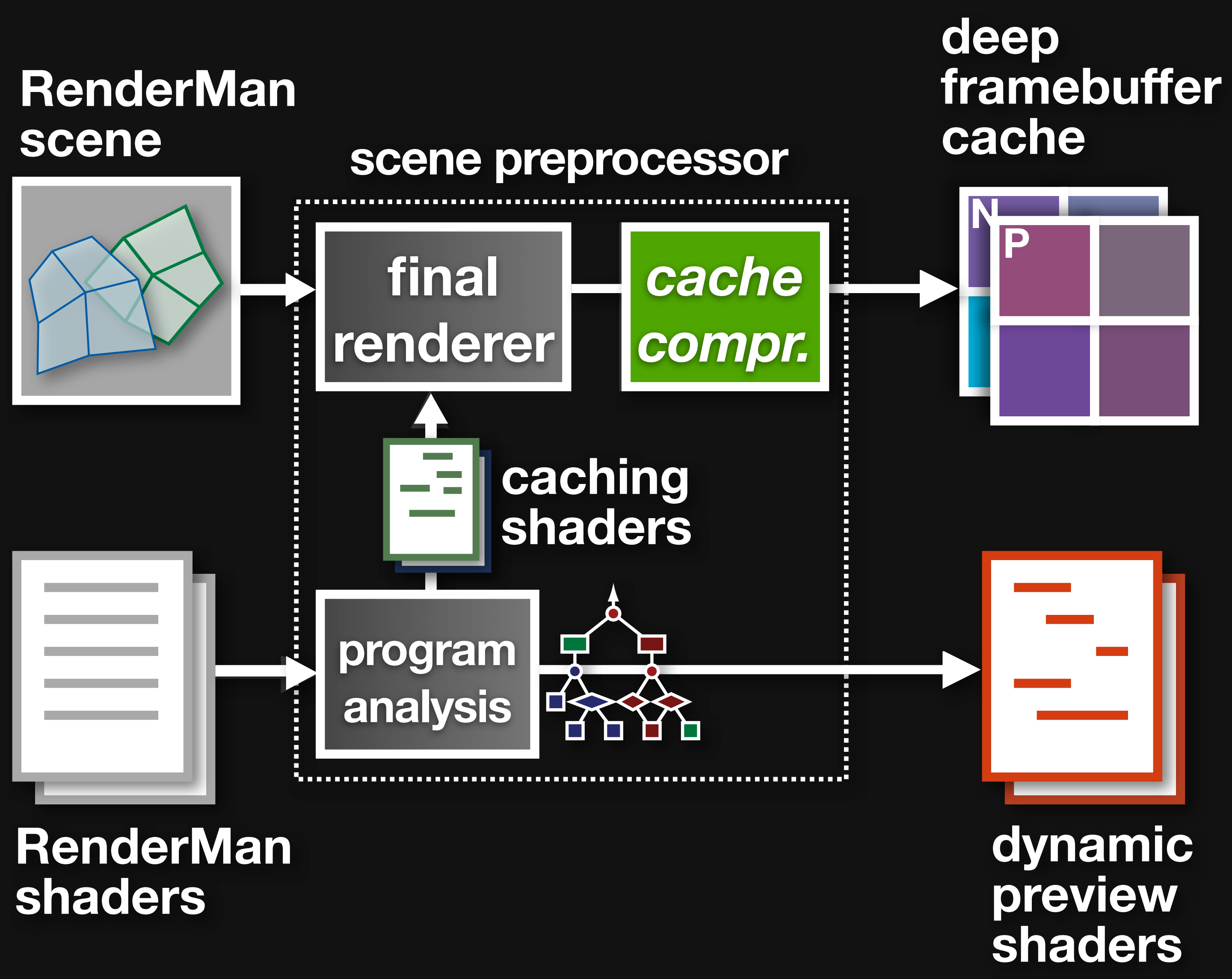
Cache Compression



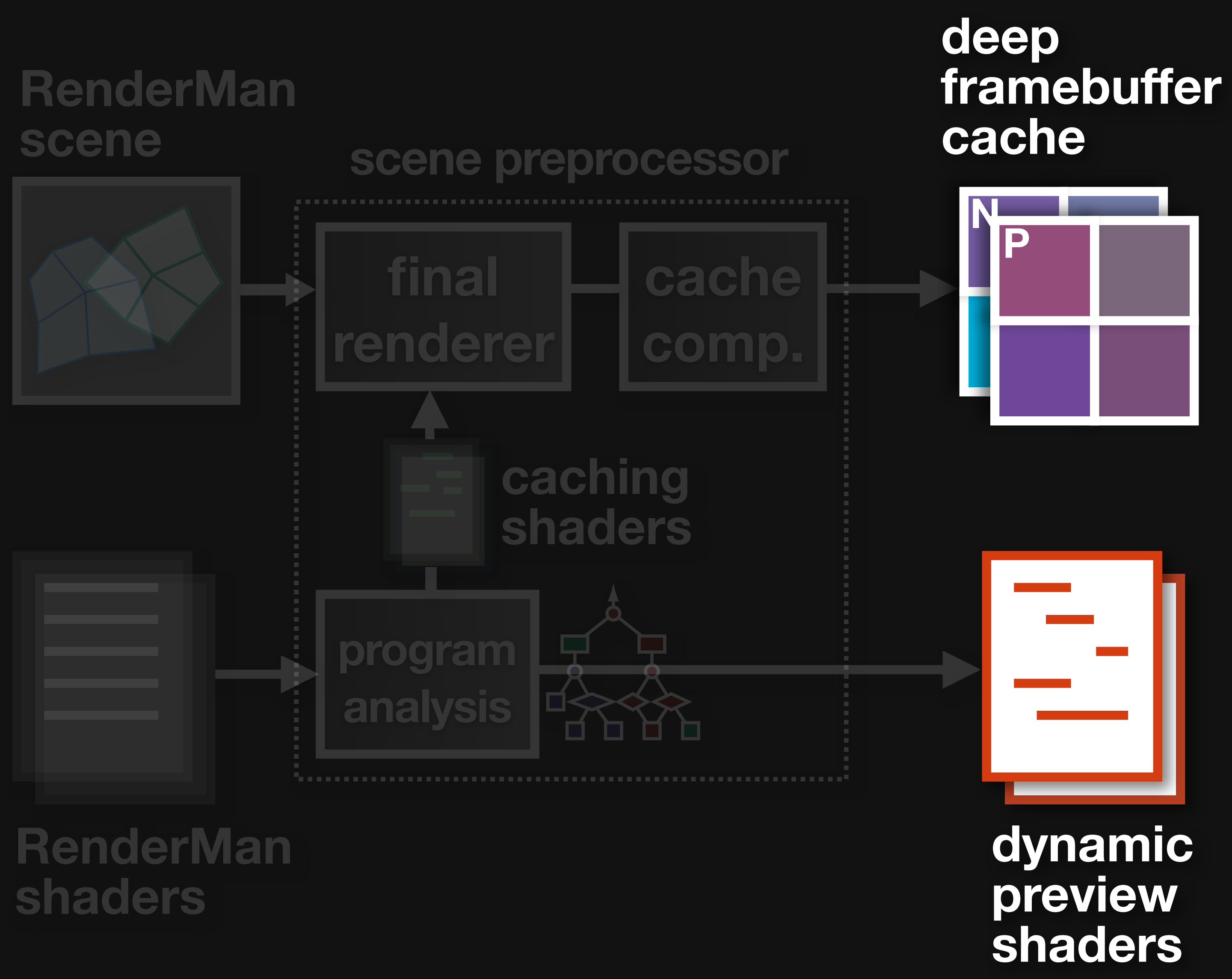
Remove redundant values *after* caching

→ *5x compression*

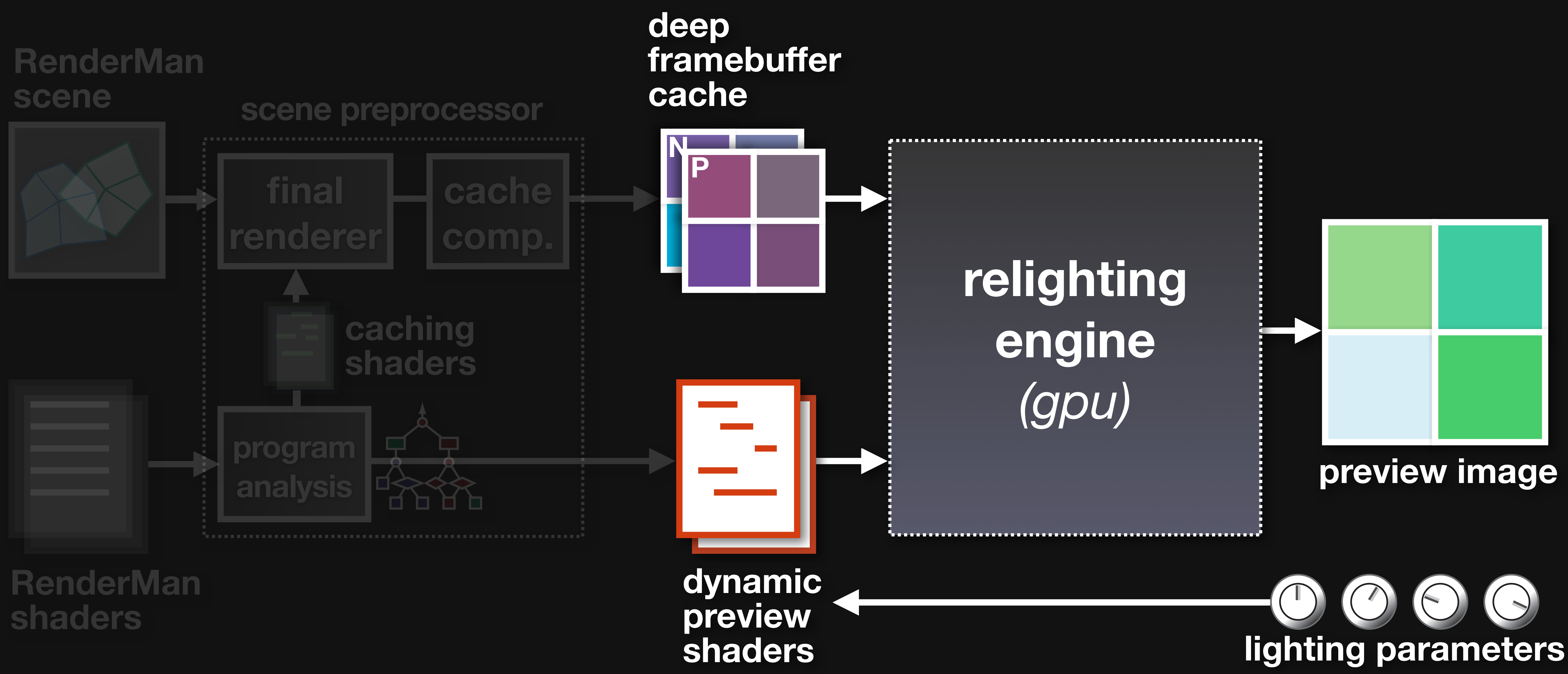
System Overview



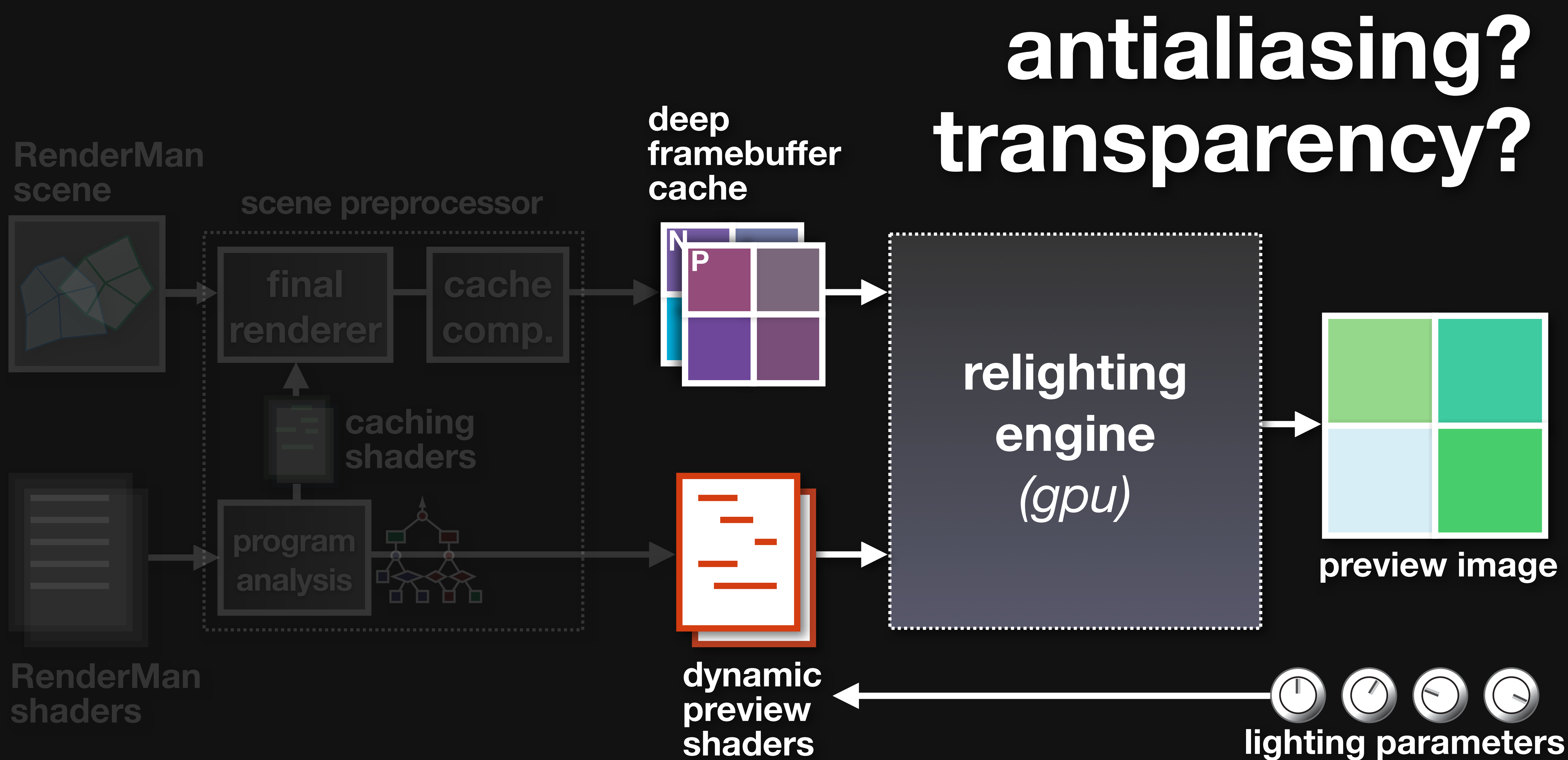
System Overview



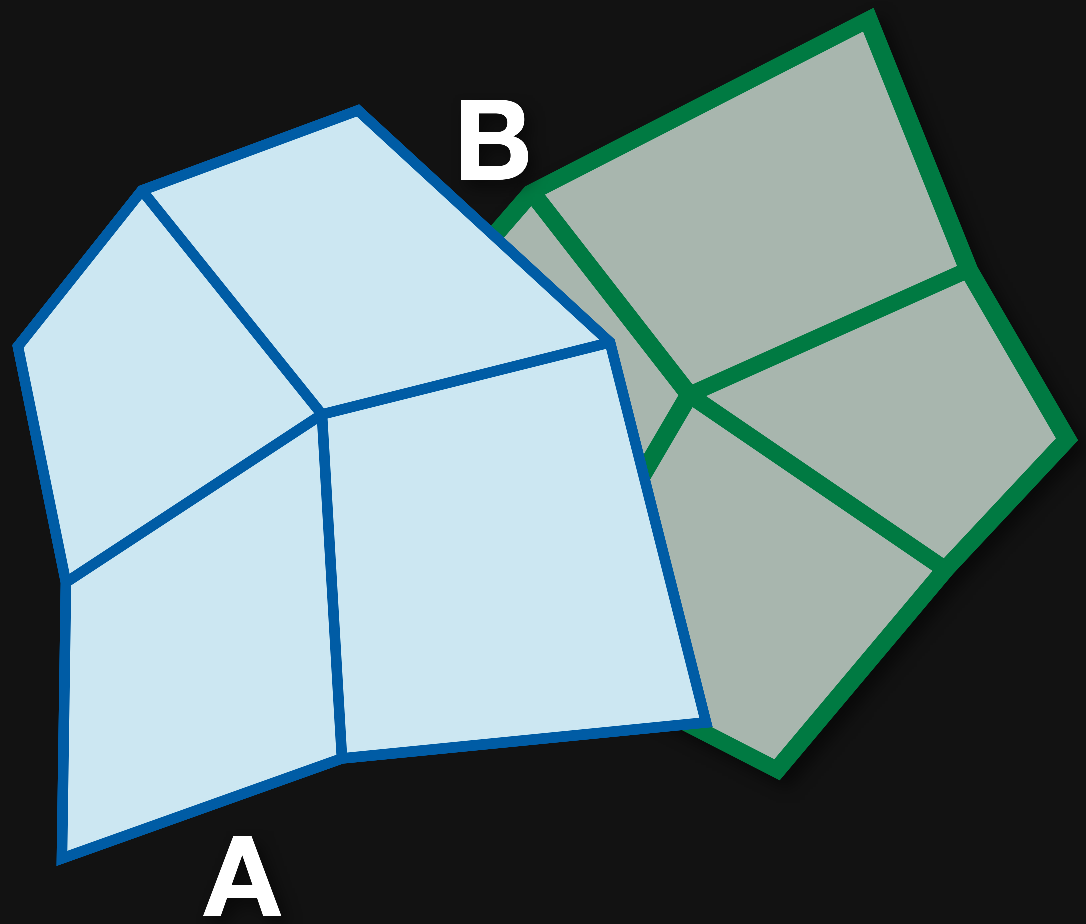
System Overview



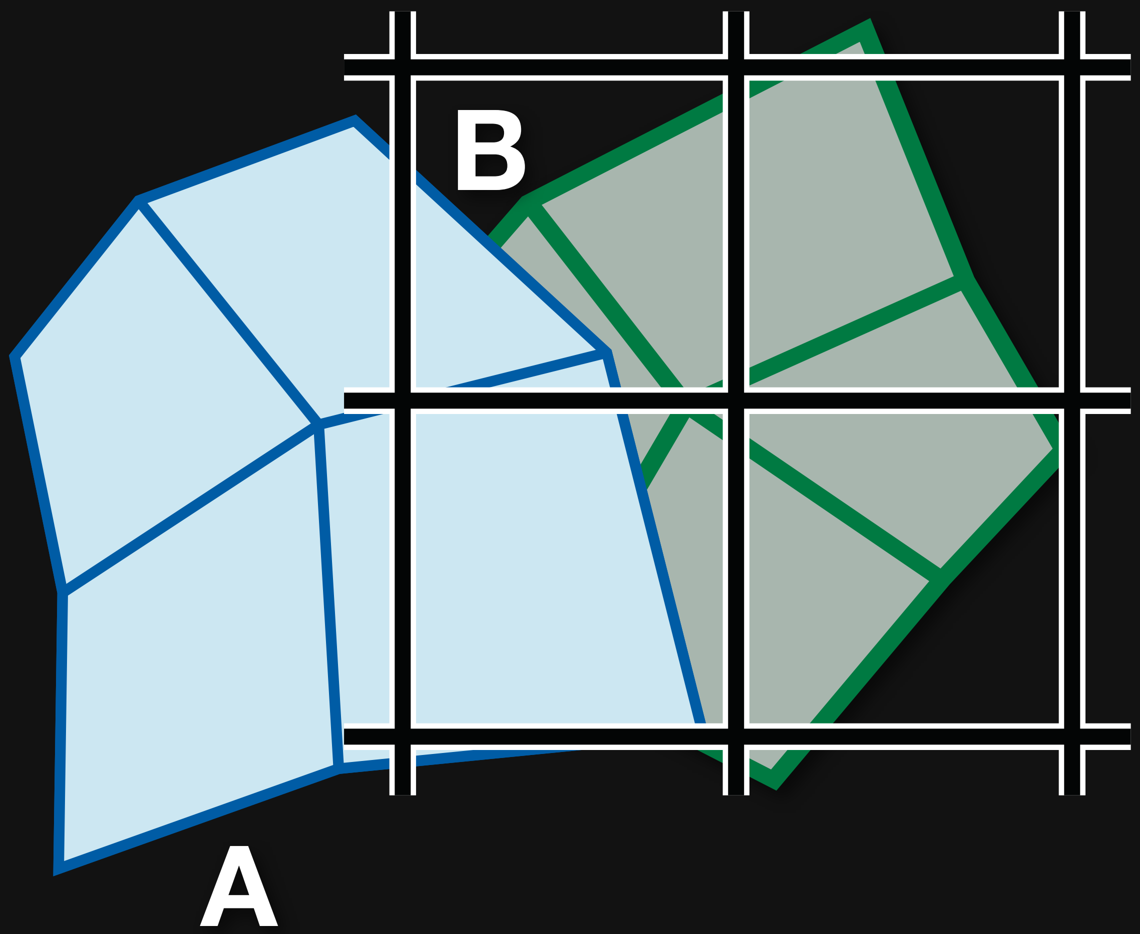
System Overview



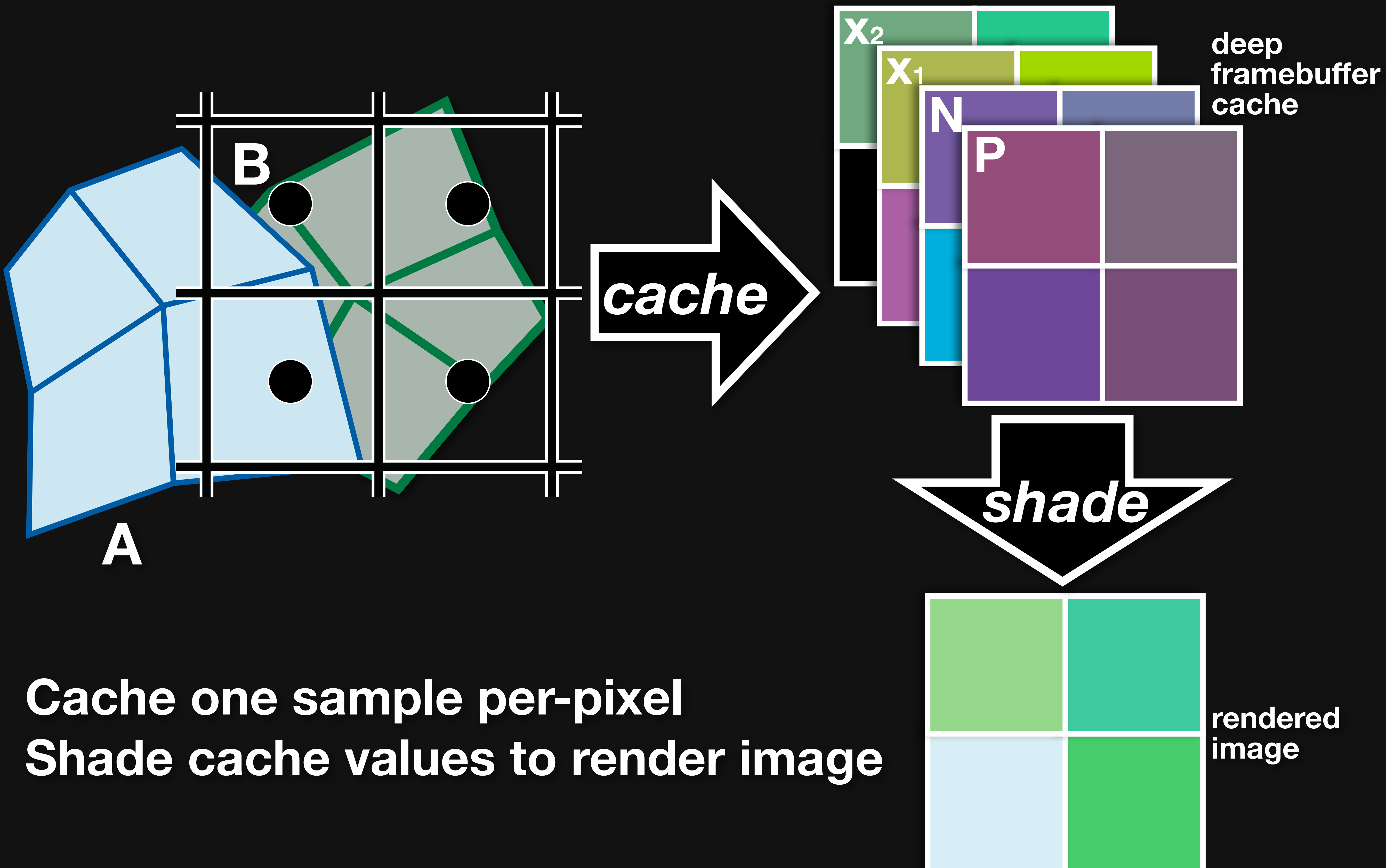
Classical Deep Framebuffer



Classical Deep Framebuffer

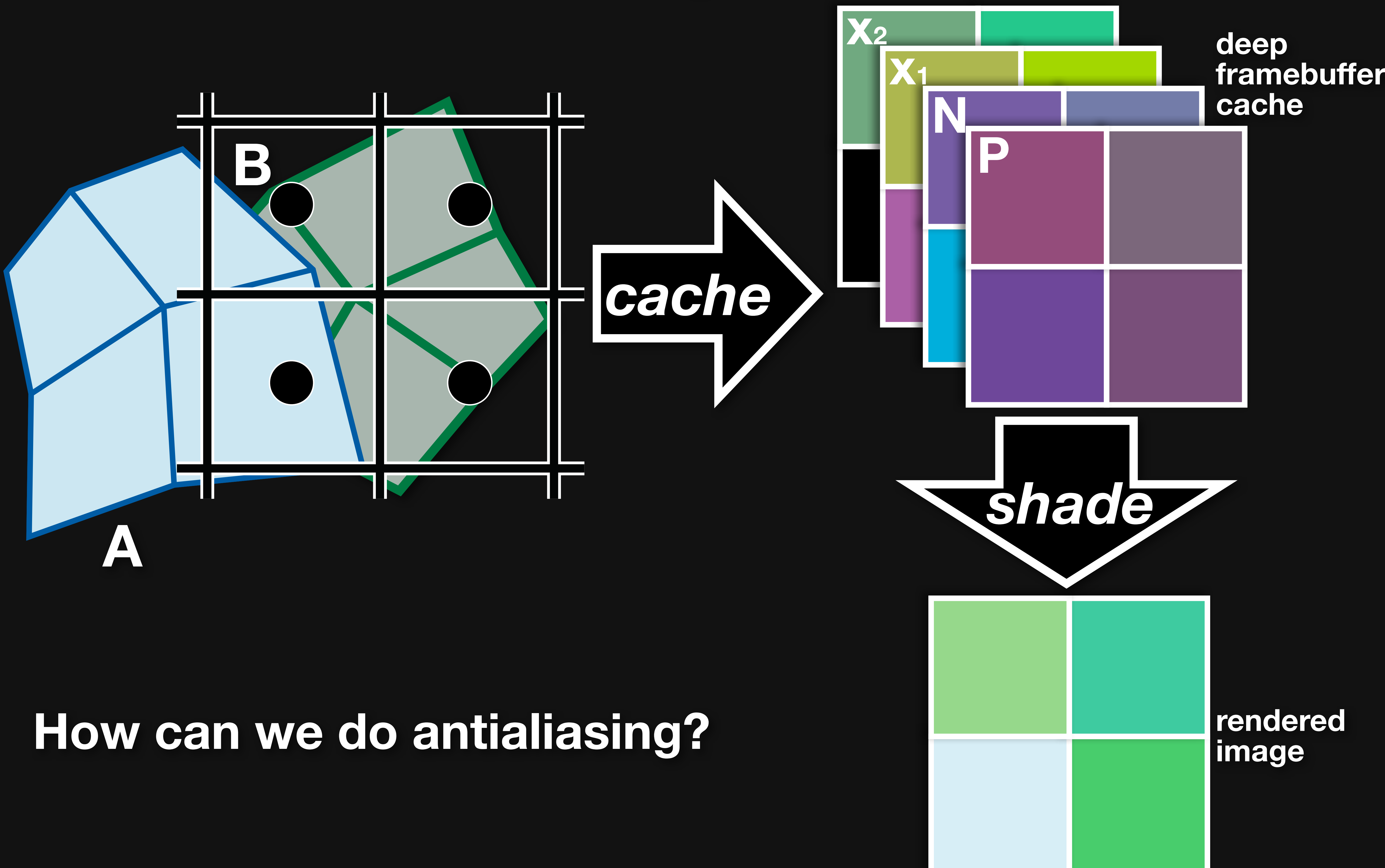


Classical Deep Framebuffer



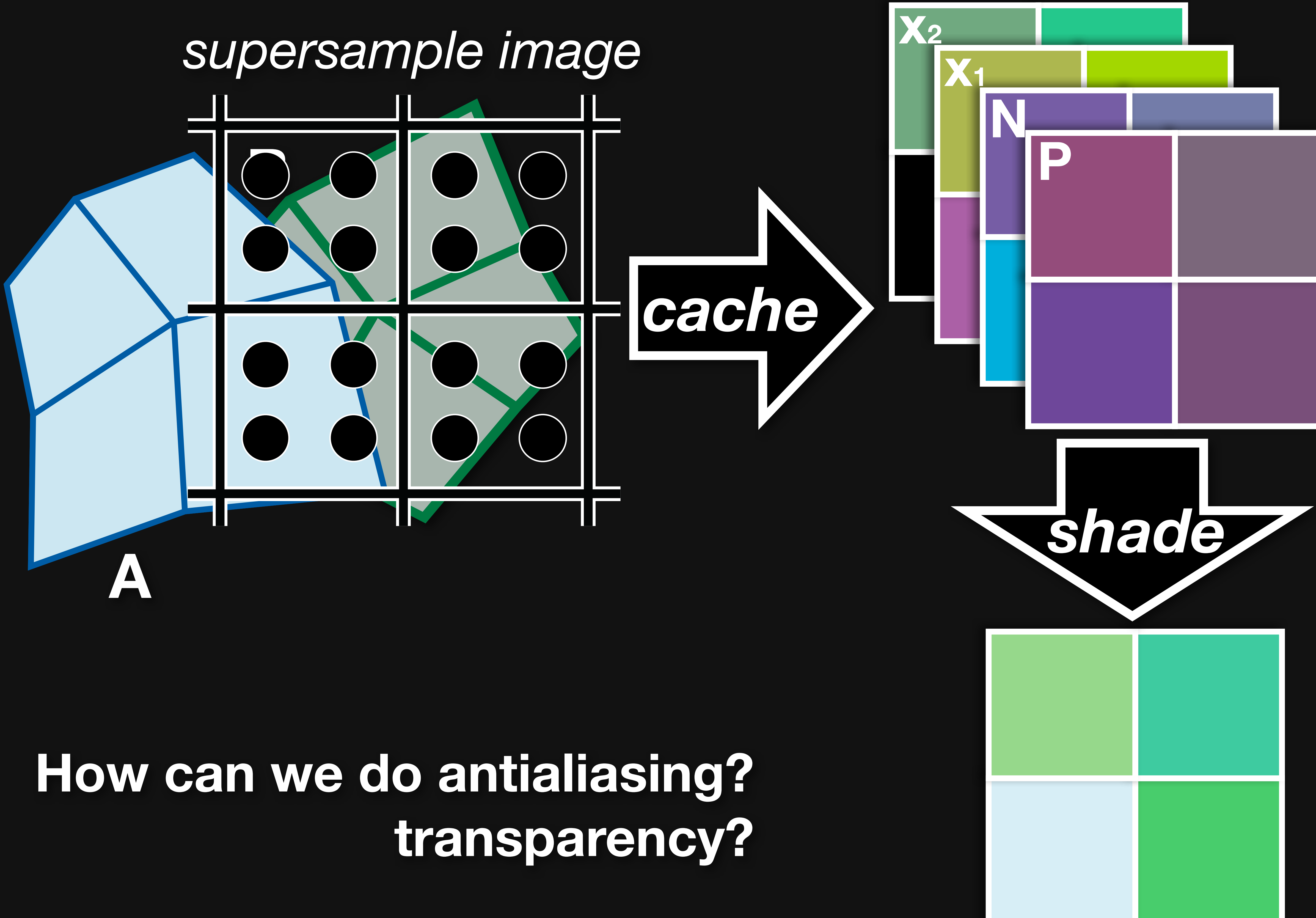
Cache one sample per-pixel
Shade cache values to render image

Classical Deep Framebuffer



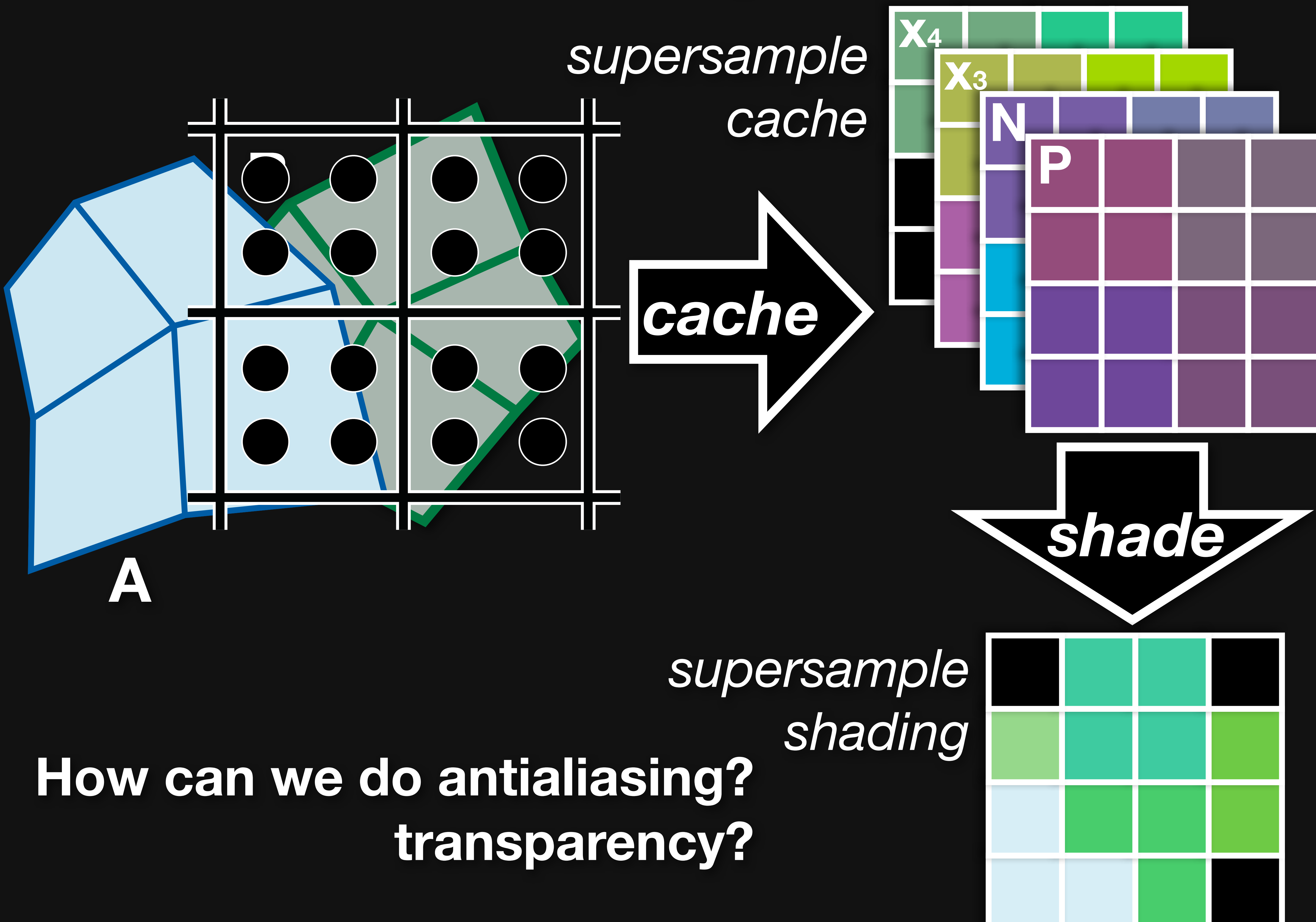
How can we do antialiasing?

Classical Deep Framebuffer



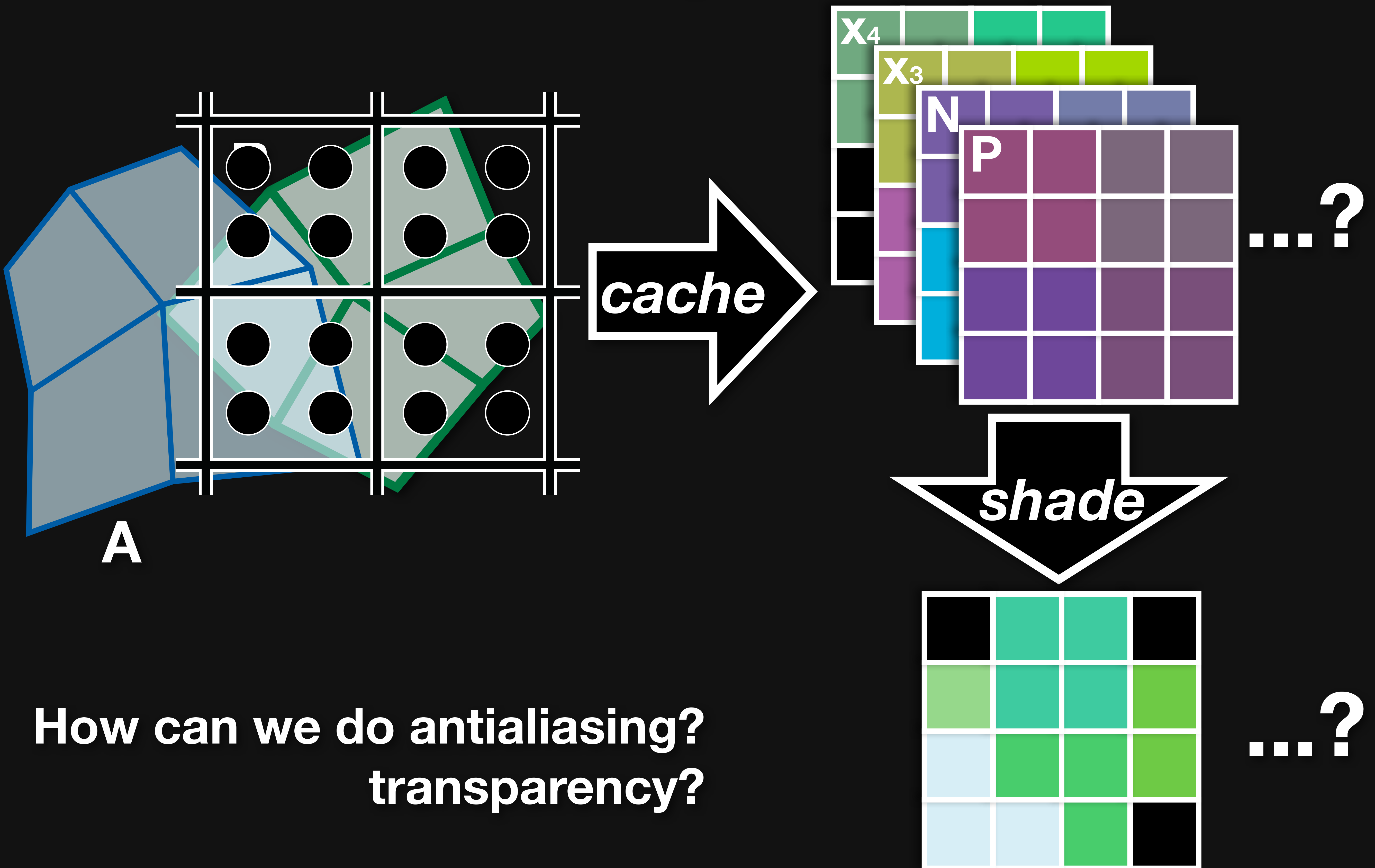
How can we do antialiasing?
transparency?

Classical Deep Framebuffer



How can we do antialiasing? transparency?

Classical Deep Framebuffer



How can we do antialiasing?
transparency?

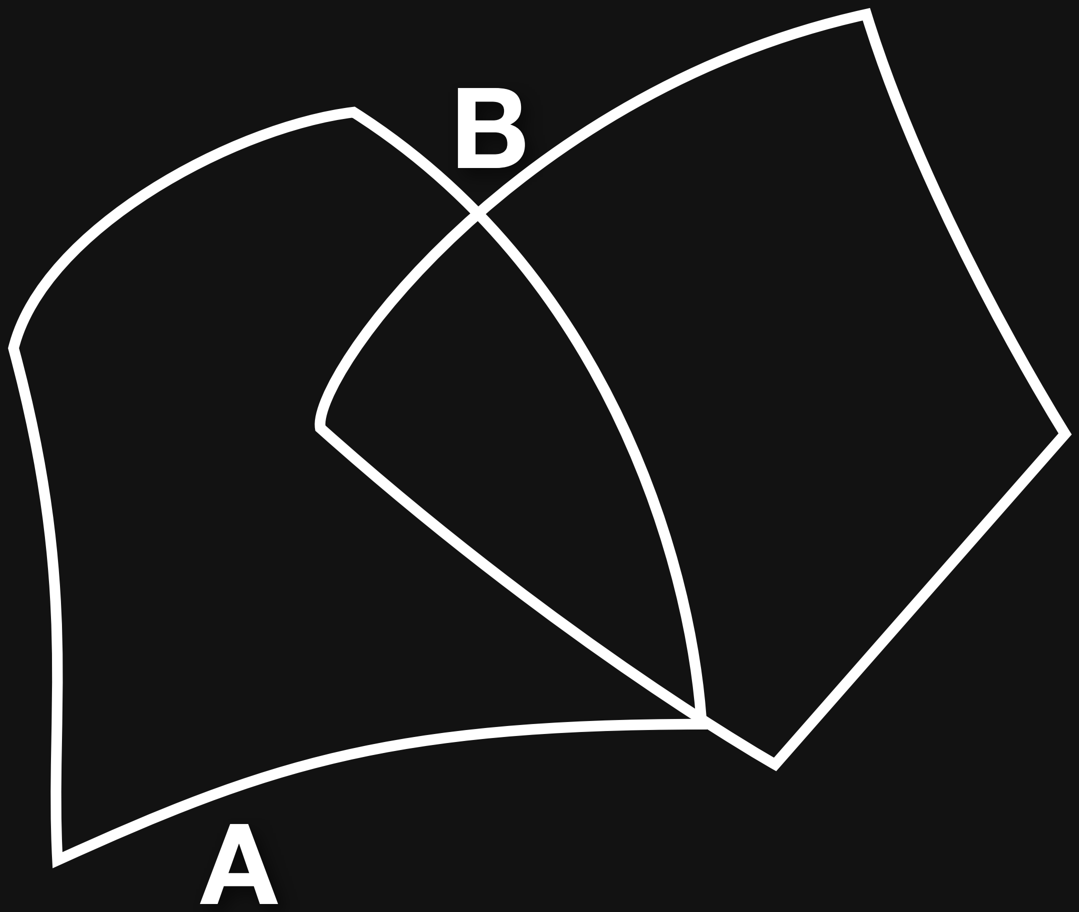
Our Indirect Framebuffer

- **Decouple shading from visibility**
  - **only supersample *visibility***

▸ ***Do what RenderMan does***

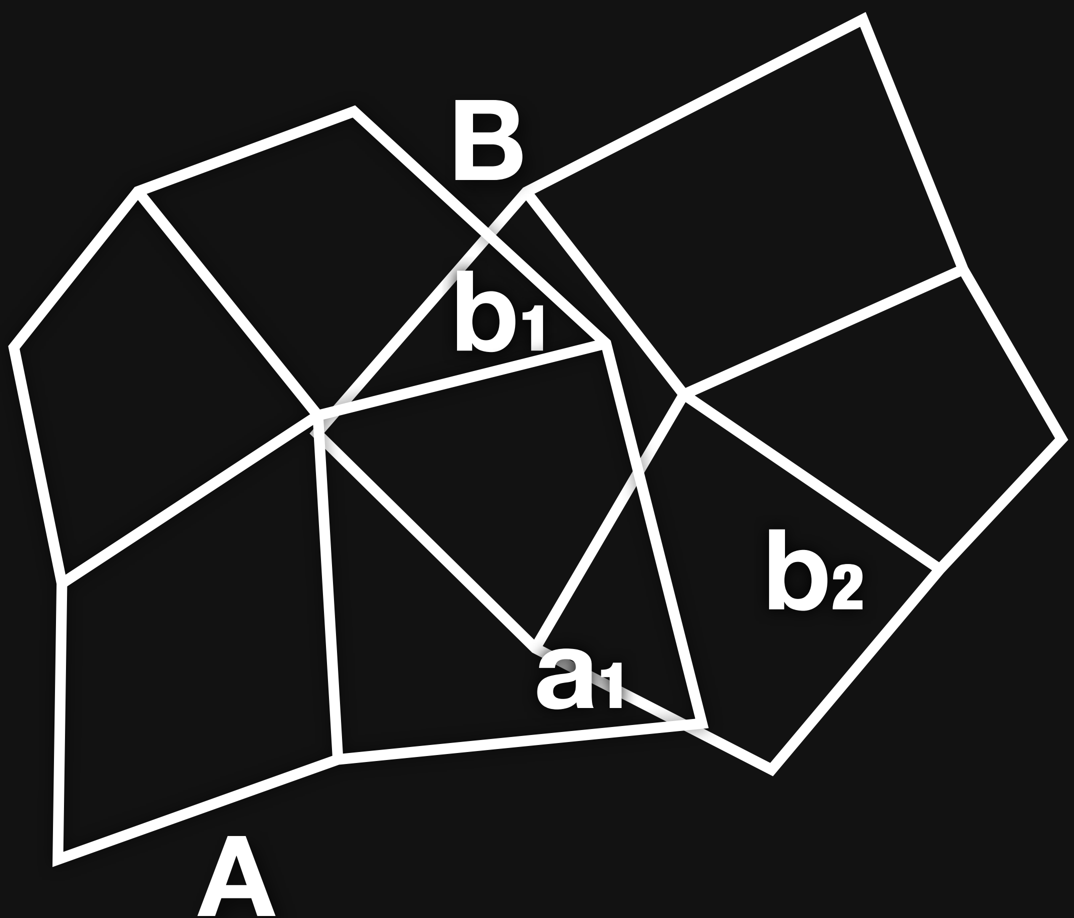
- **Compress samples**
  based on static visibility

RenderMan / REYES



show
filtering
pixel color?

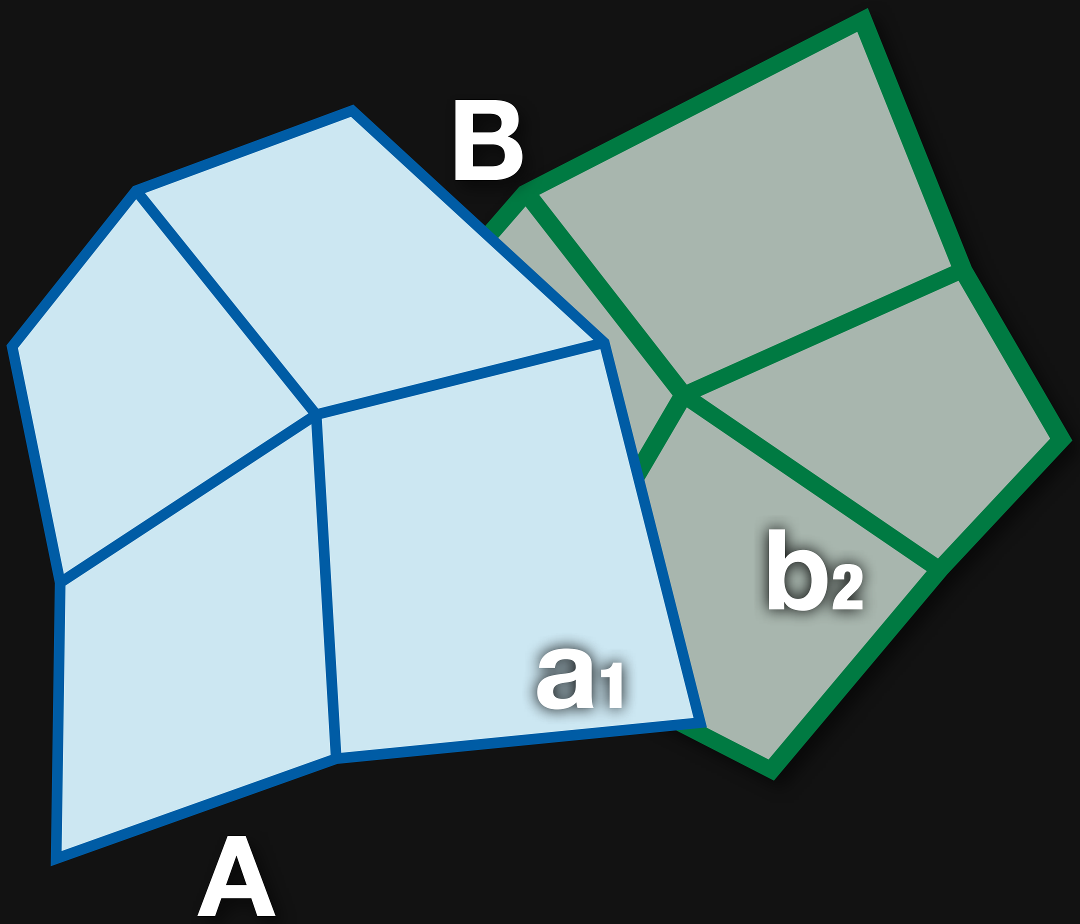
RenderMan / REYES



show
filtering
pixel color?

dice into micropolygons

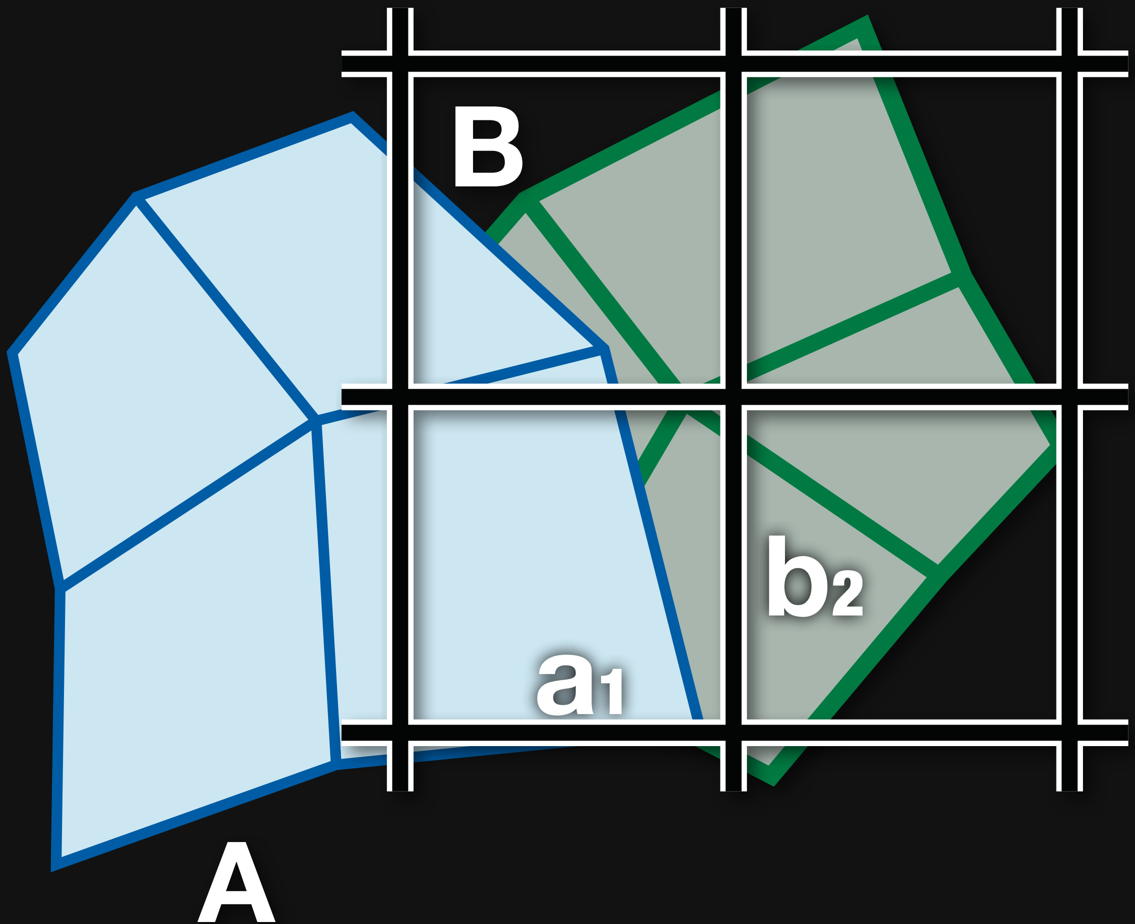
RenderMan / REYES

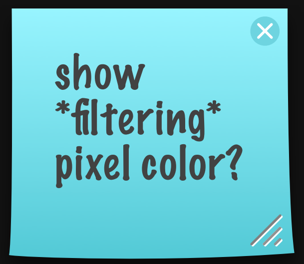


show
filtering
pixel color?

shade micropolygons

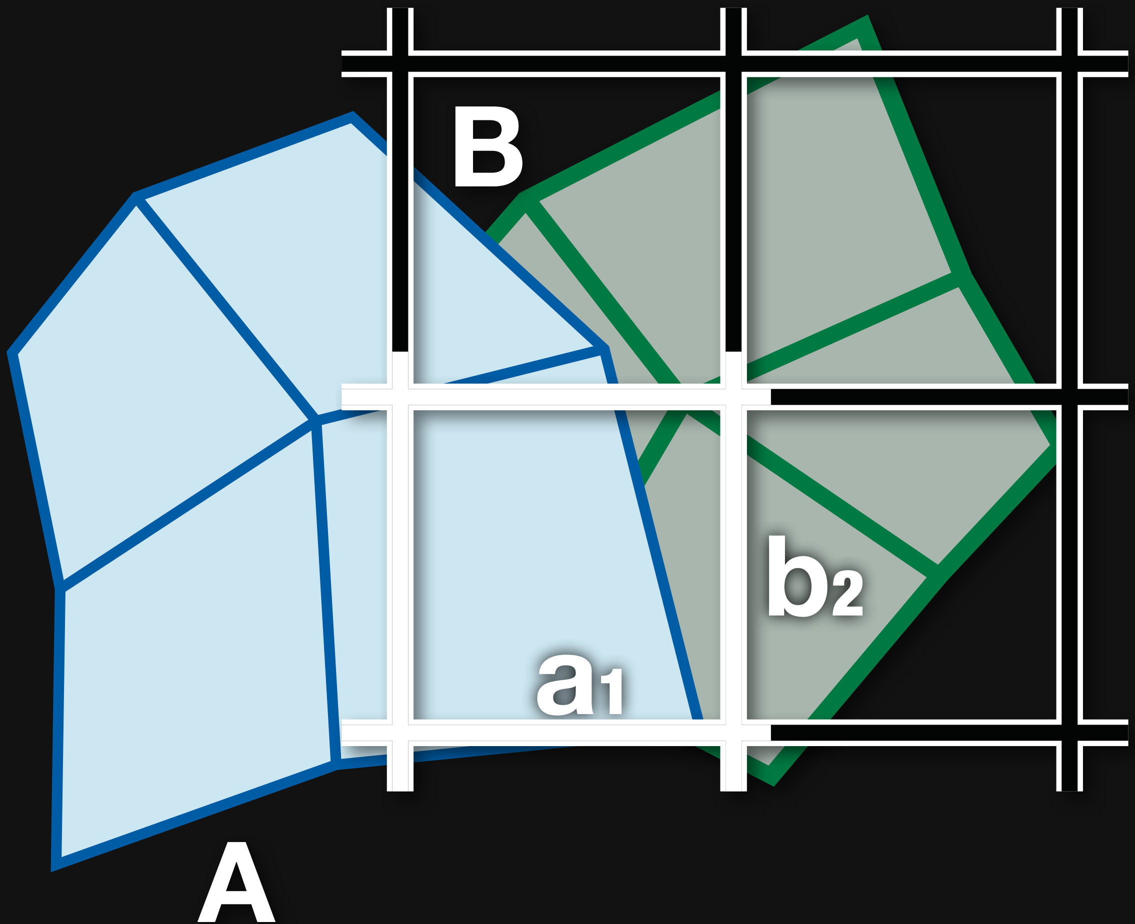
RenderMan / REYES

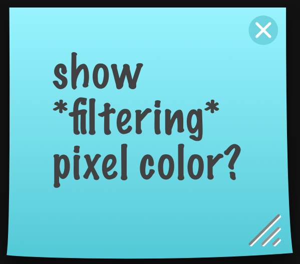




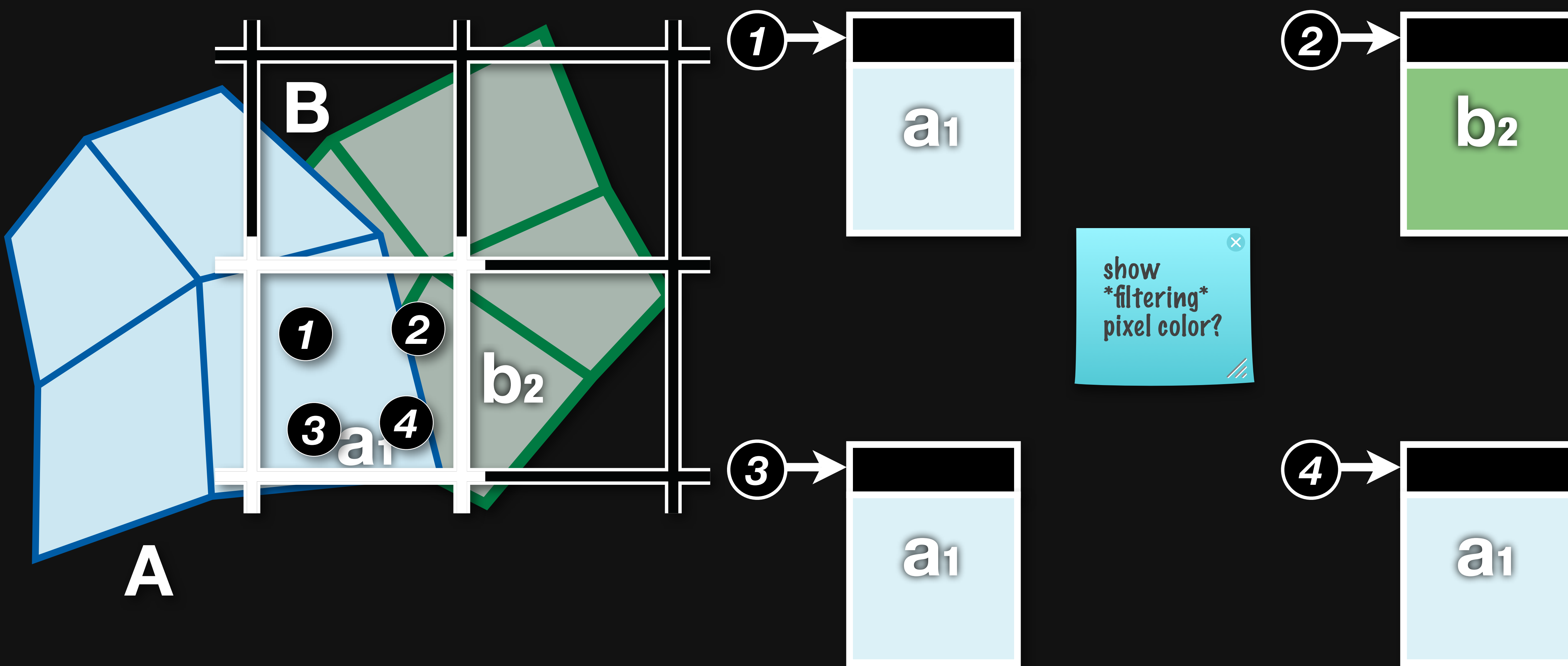
rasterize micropolygons

RenderMan / REYES



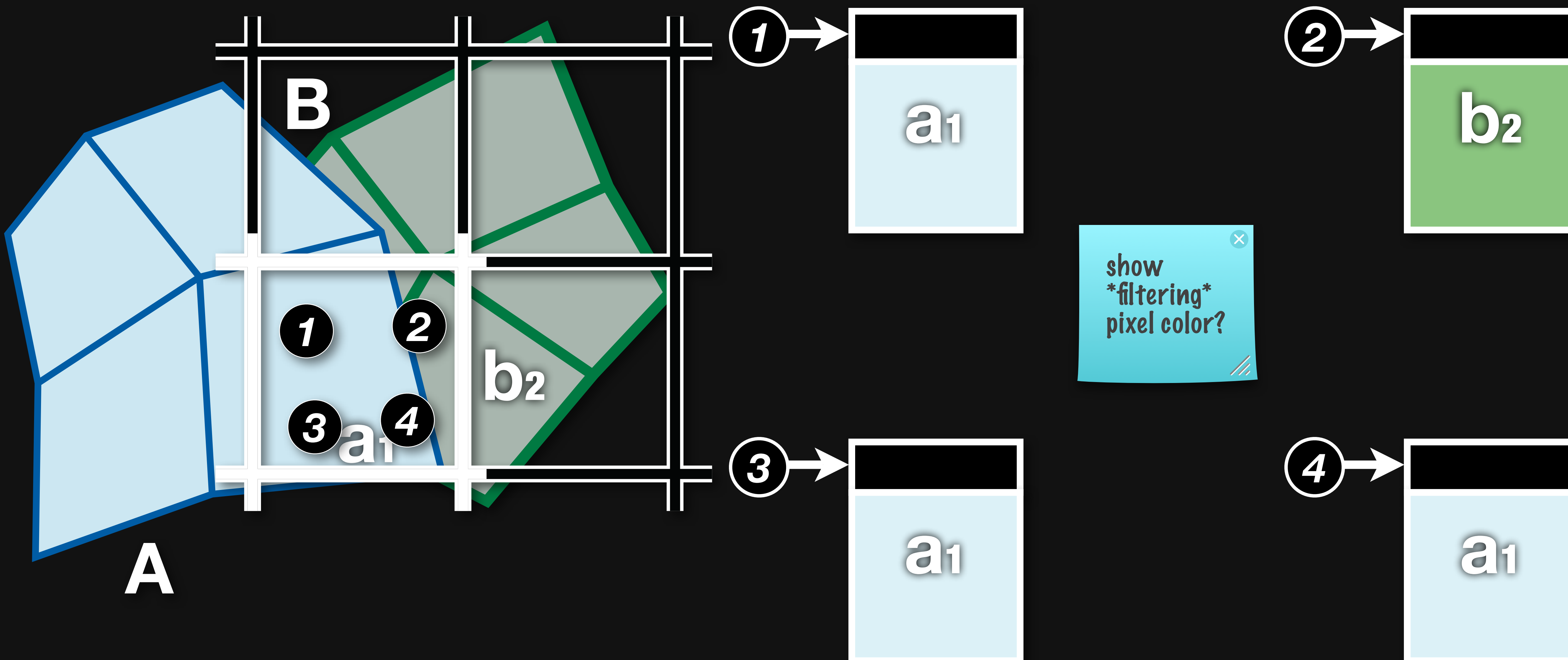


RenderMan / REYES



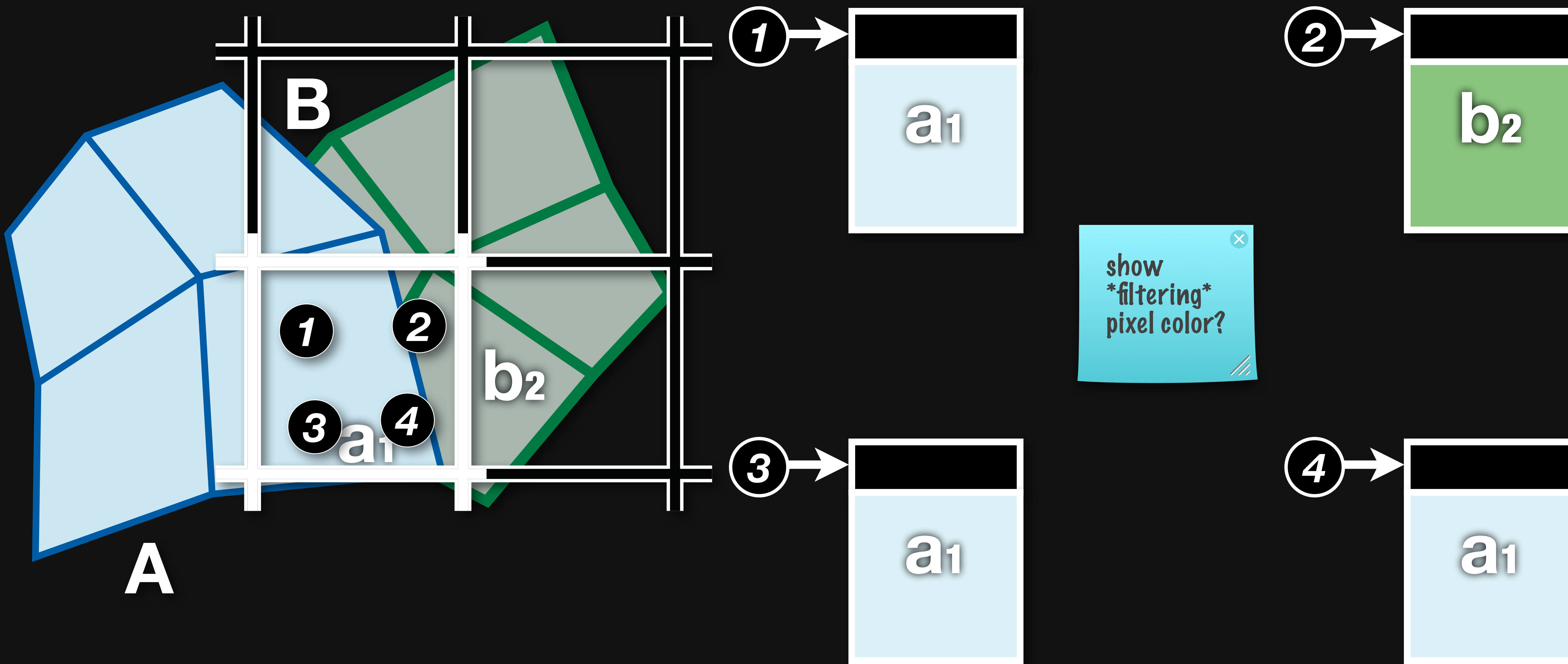
pixel = $a_1+b_2+a_1+a_1$

RenderMan / REYES



$$\text{pixel} = \frac{a_1+b_2+a_1+a_1}{4}$$

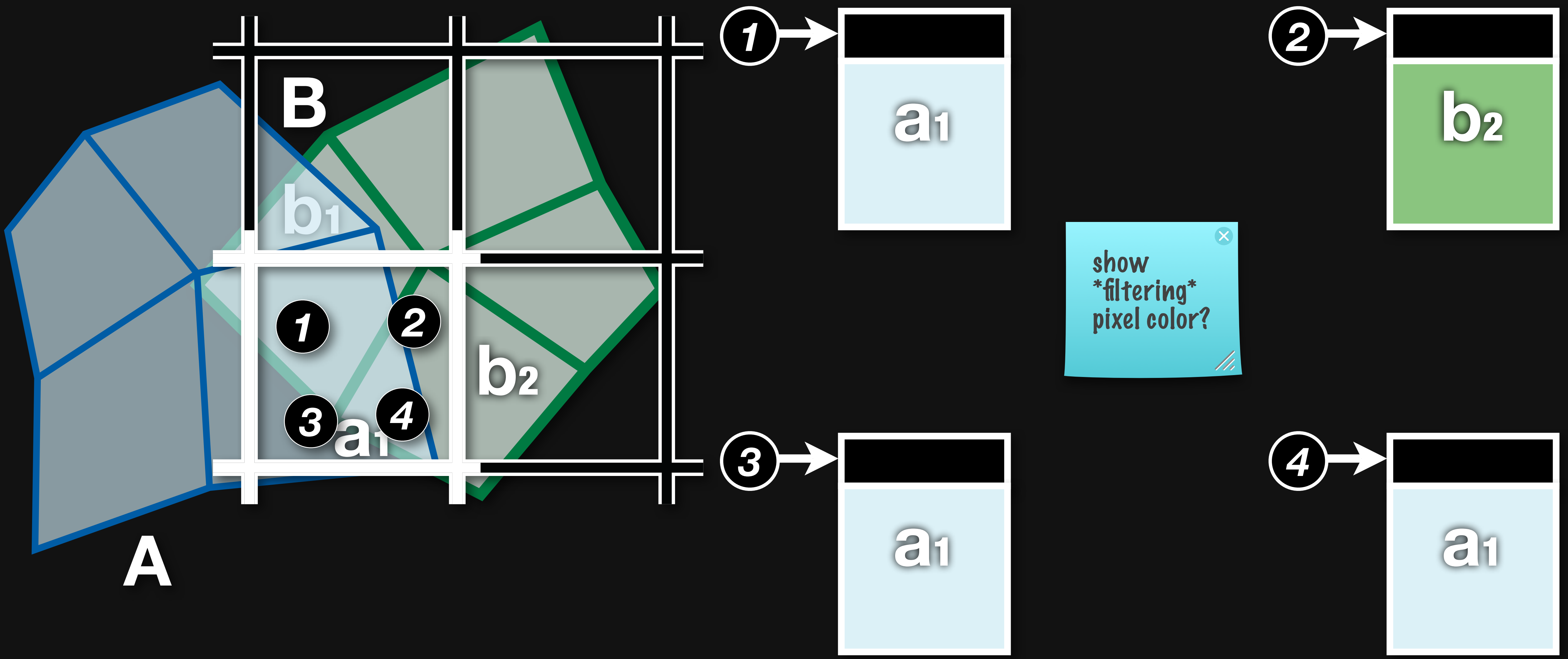
RenderMan / REYES



show *filtering* pixel color?

$$\text{pixel} = \frac{a_1+b_2+a_1+a_1}{4} = 0.75 * a_1 + 0.25 * b_2$$

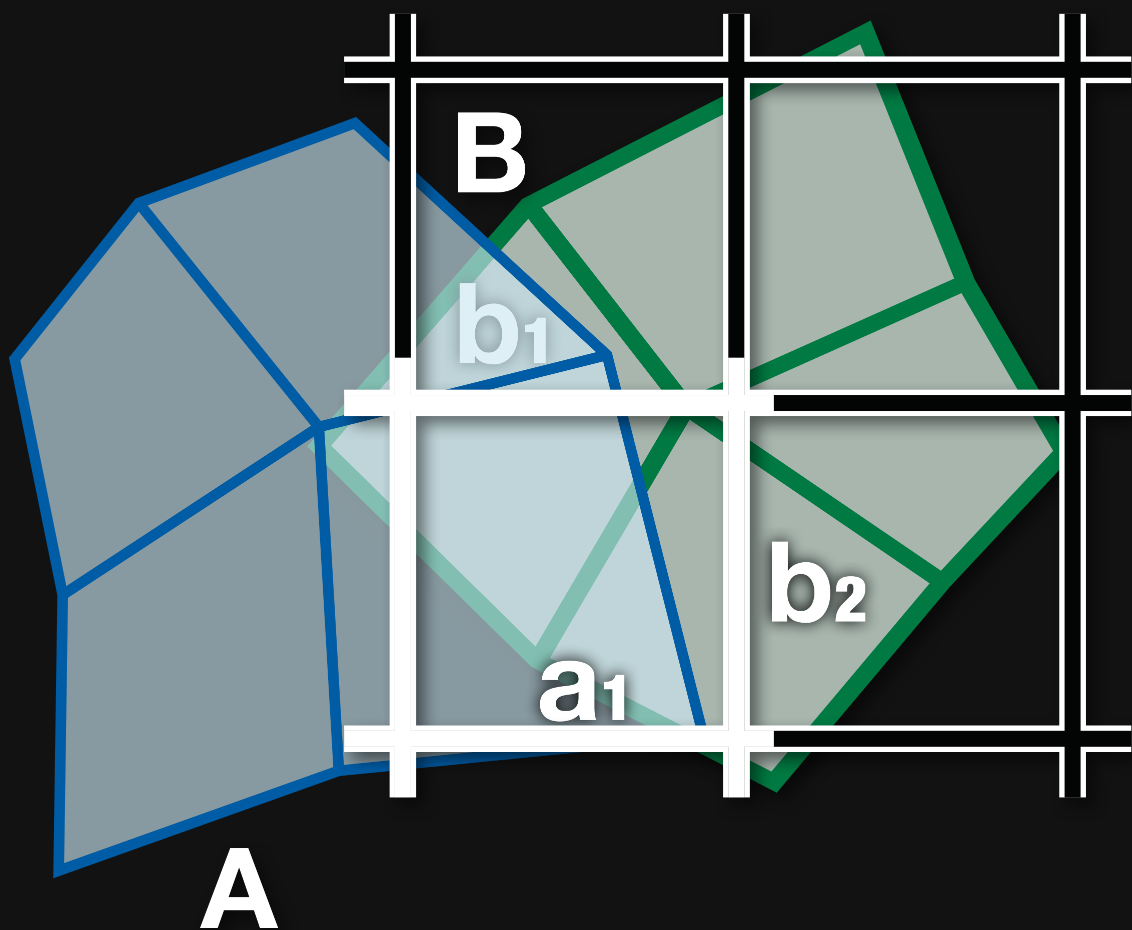
RenderMan / REYES



show
filtering
pixel color?

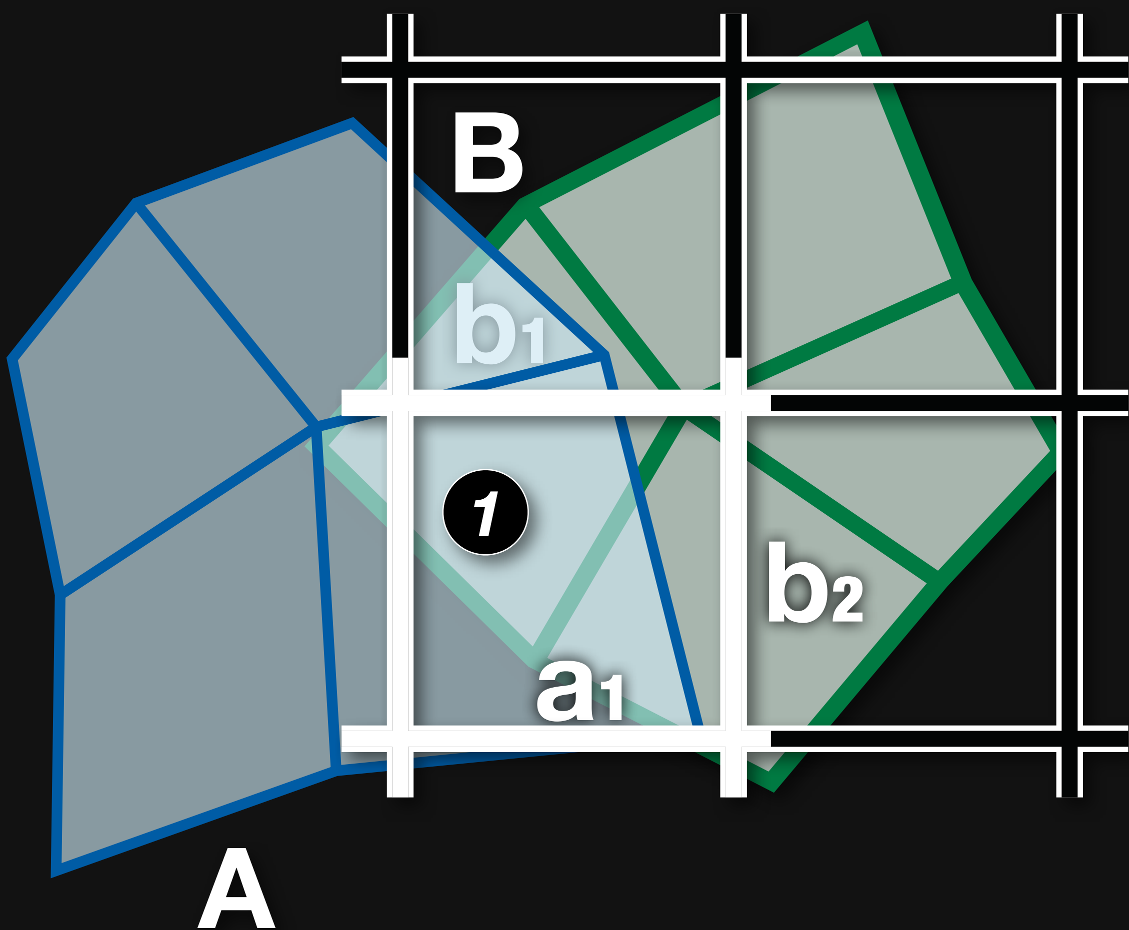
pixel = ?

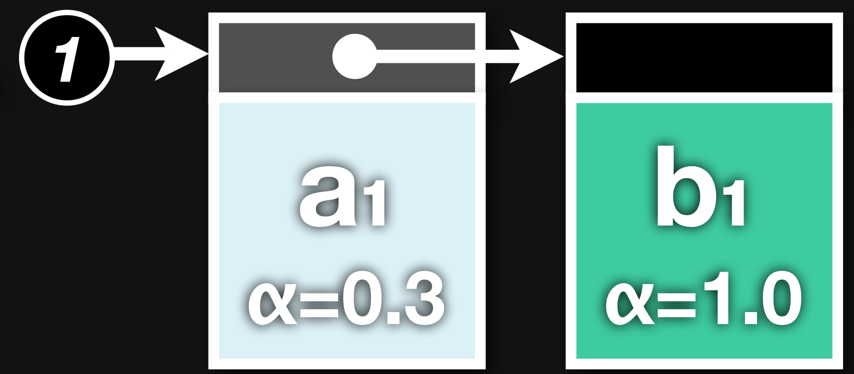
RenderMan / REYES



pixel = ?

RenderMan / REYES

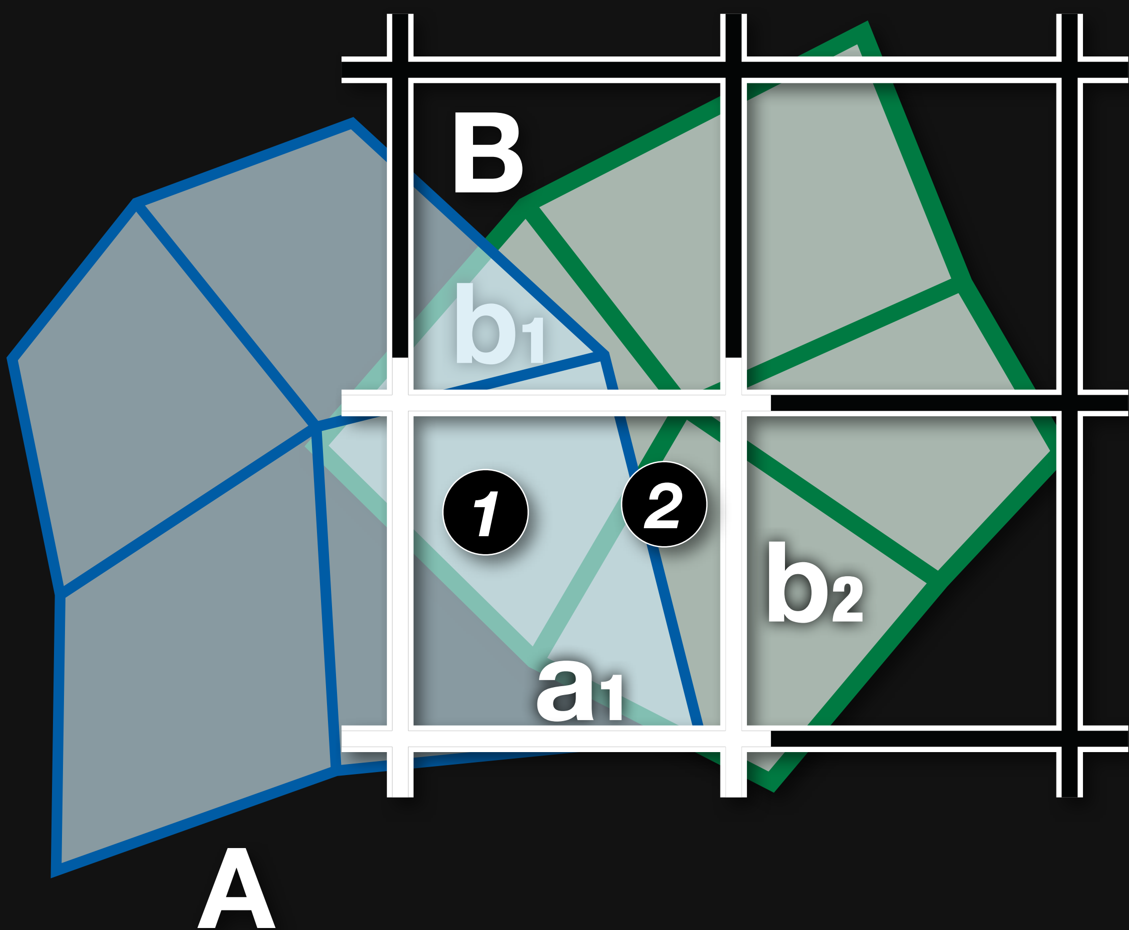


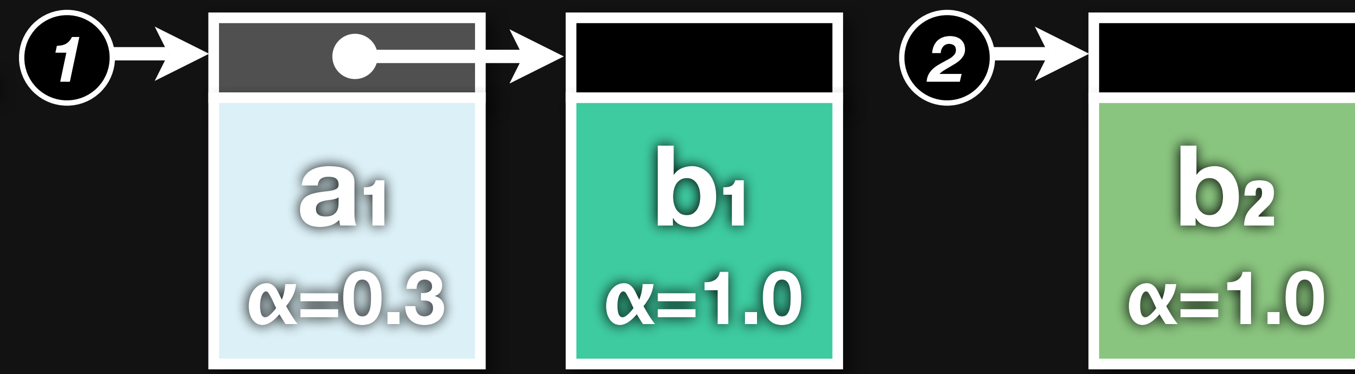


$0.3 * \mathbf{a}_1 + 0.7 * \mathbf{b}_1$

pixel = ?

RenderMan / REYES



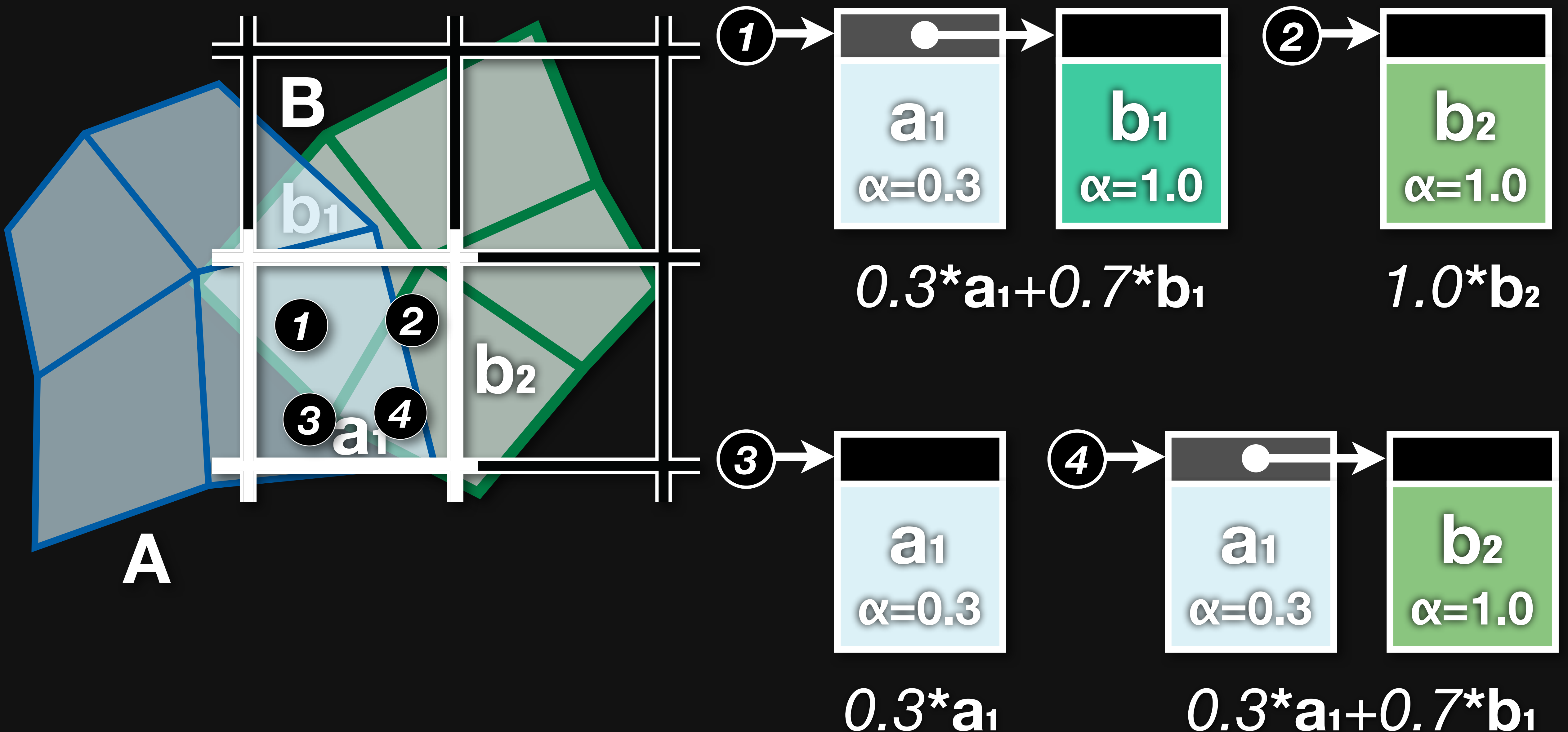


$0.3 * \mathbf{a}_1 + 0.7 * \mathbf{b}_1$

$1.0 * \mathbf{b}_2$

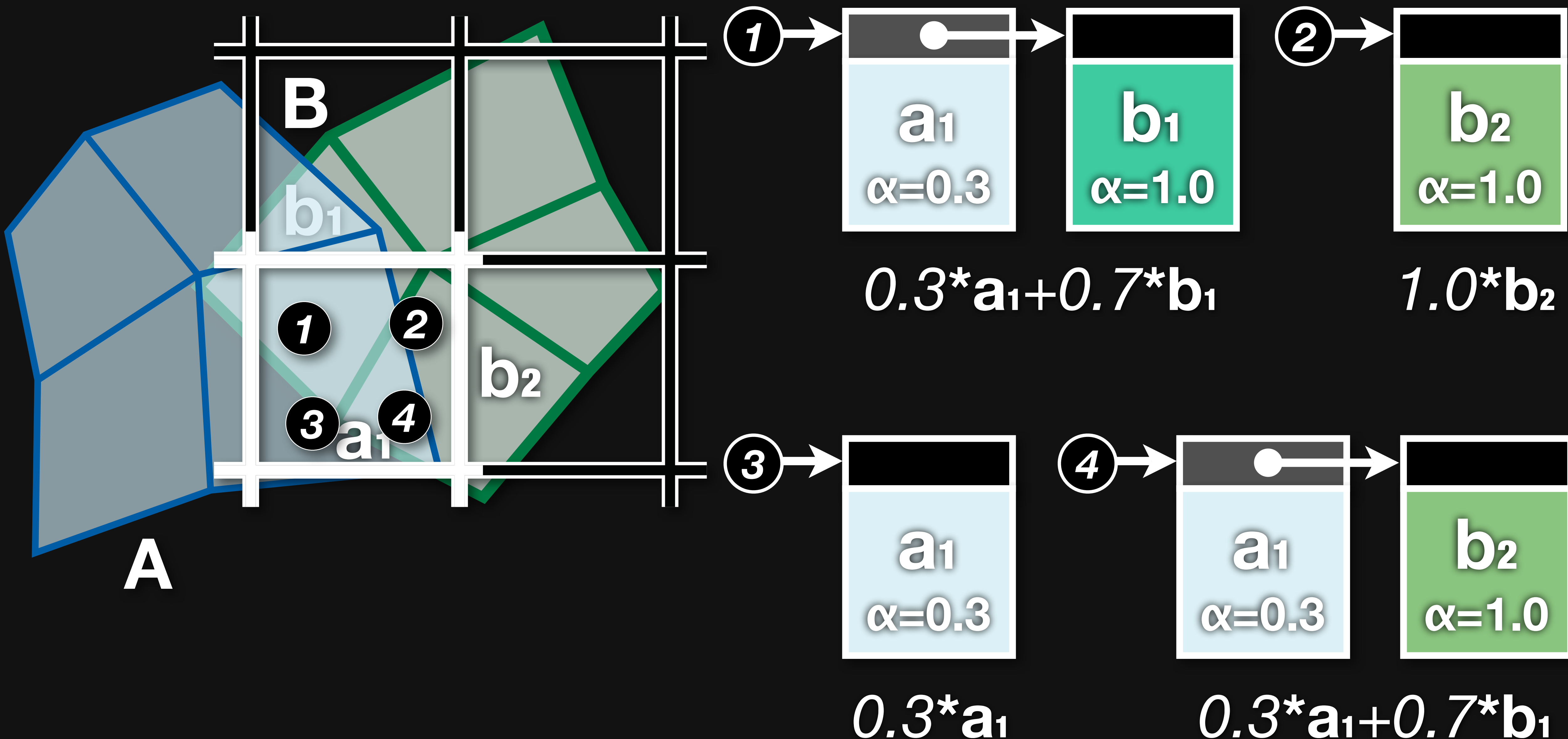
pixel = ?

RenderMan / REYES



pixel = ?

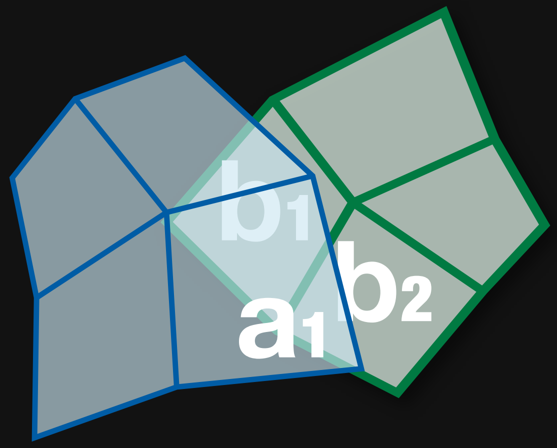
RenderMan / REYES



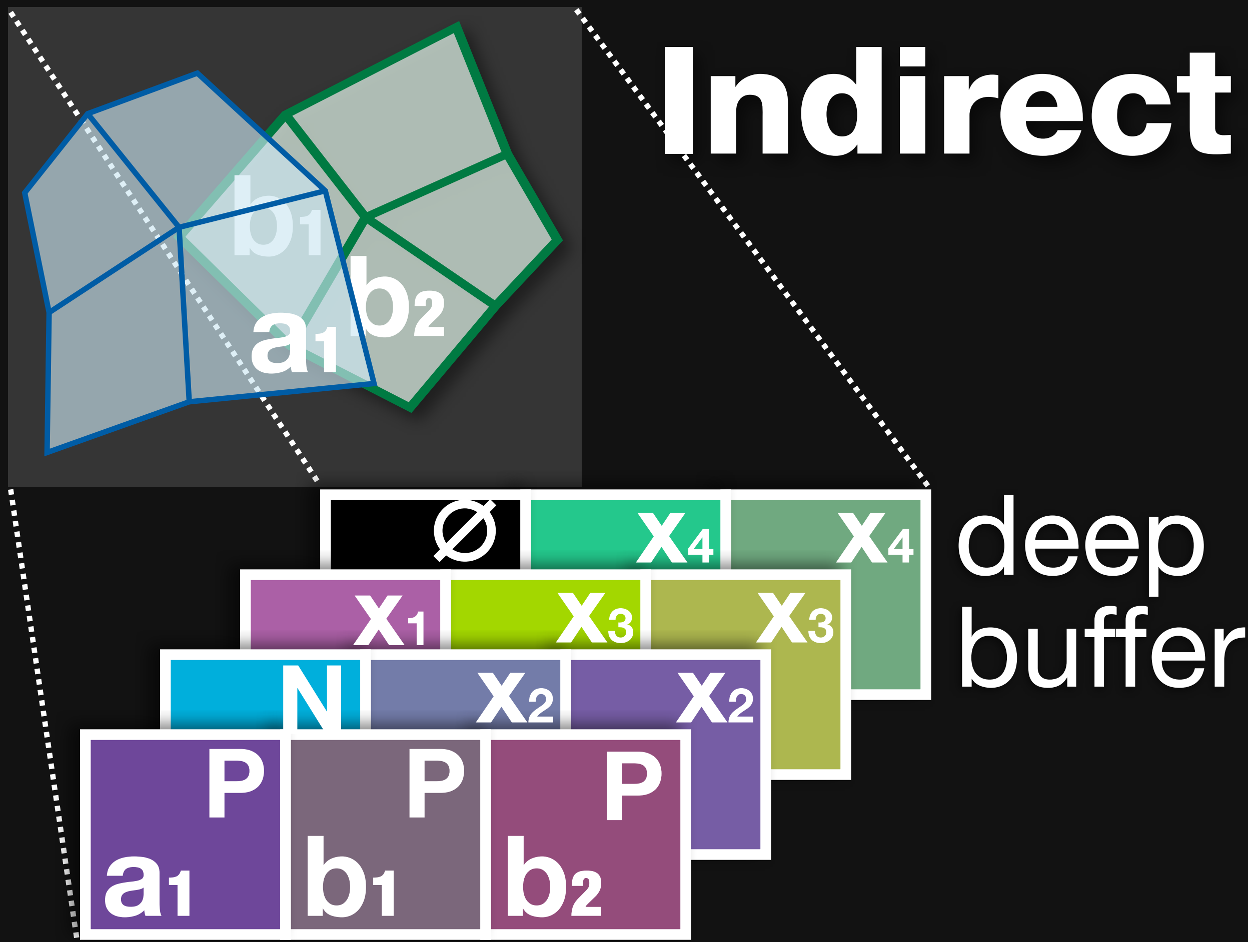
pixel = $0.225 * \mathbf{a_1} + 0.35 * \mathbf{b_1} + 0.25 * \mathbf{b_2}$

Indirect Framebuffer

Indirect Framebuffer

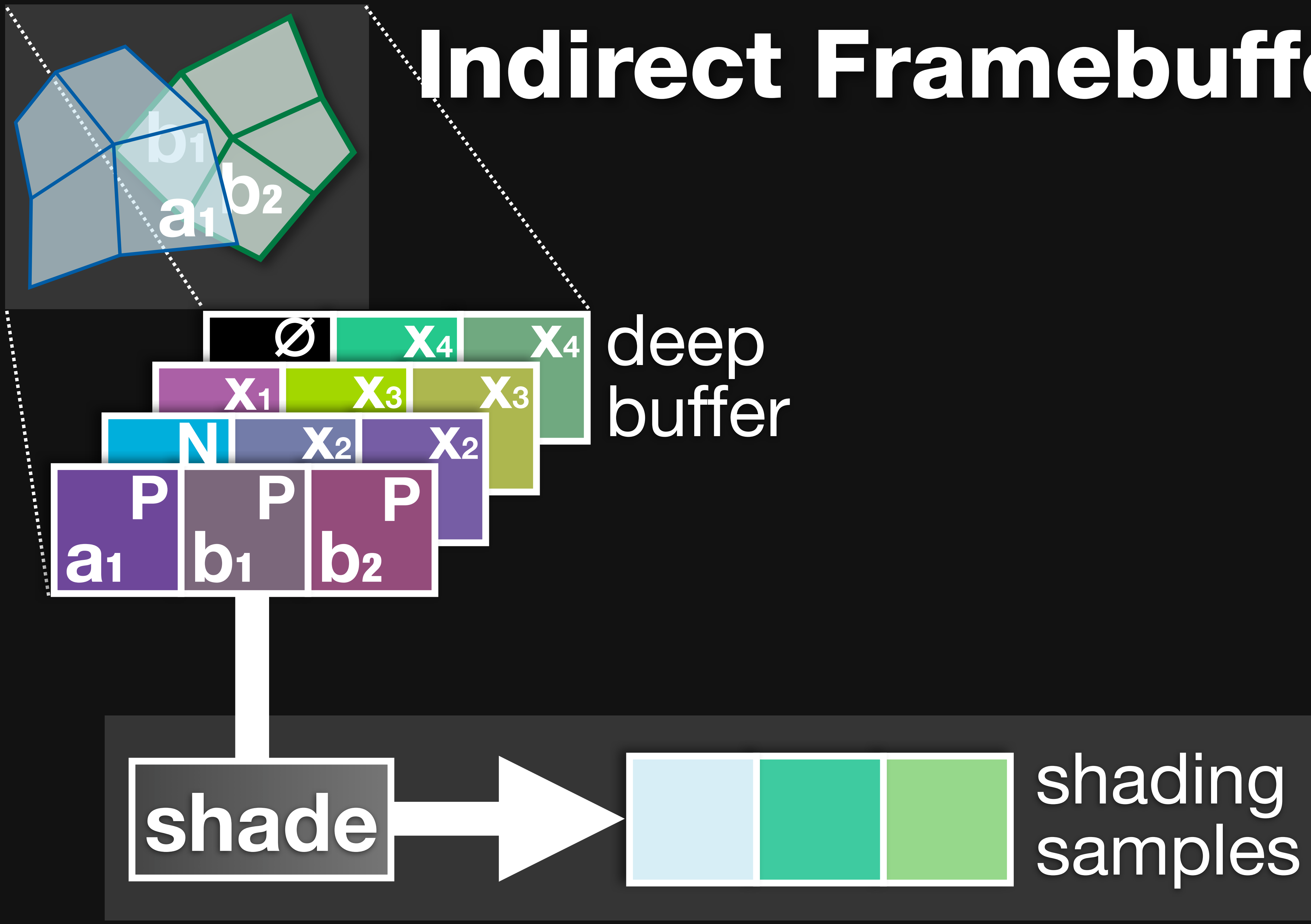


Indirect Framebuffer



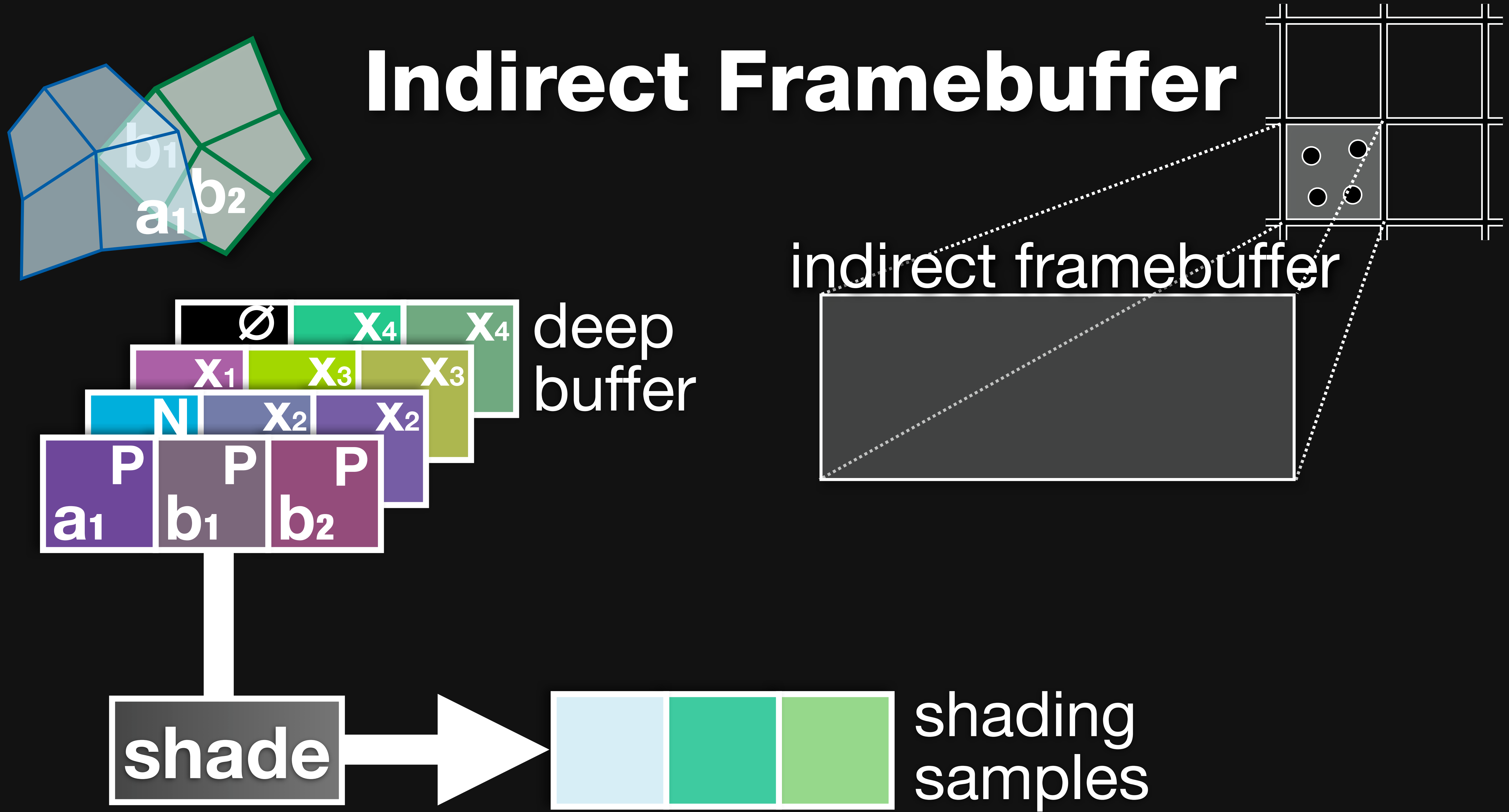
no longer image space
(per-micropolygon)

Indirect Framebuffer

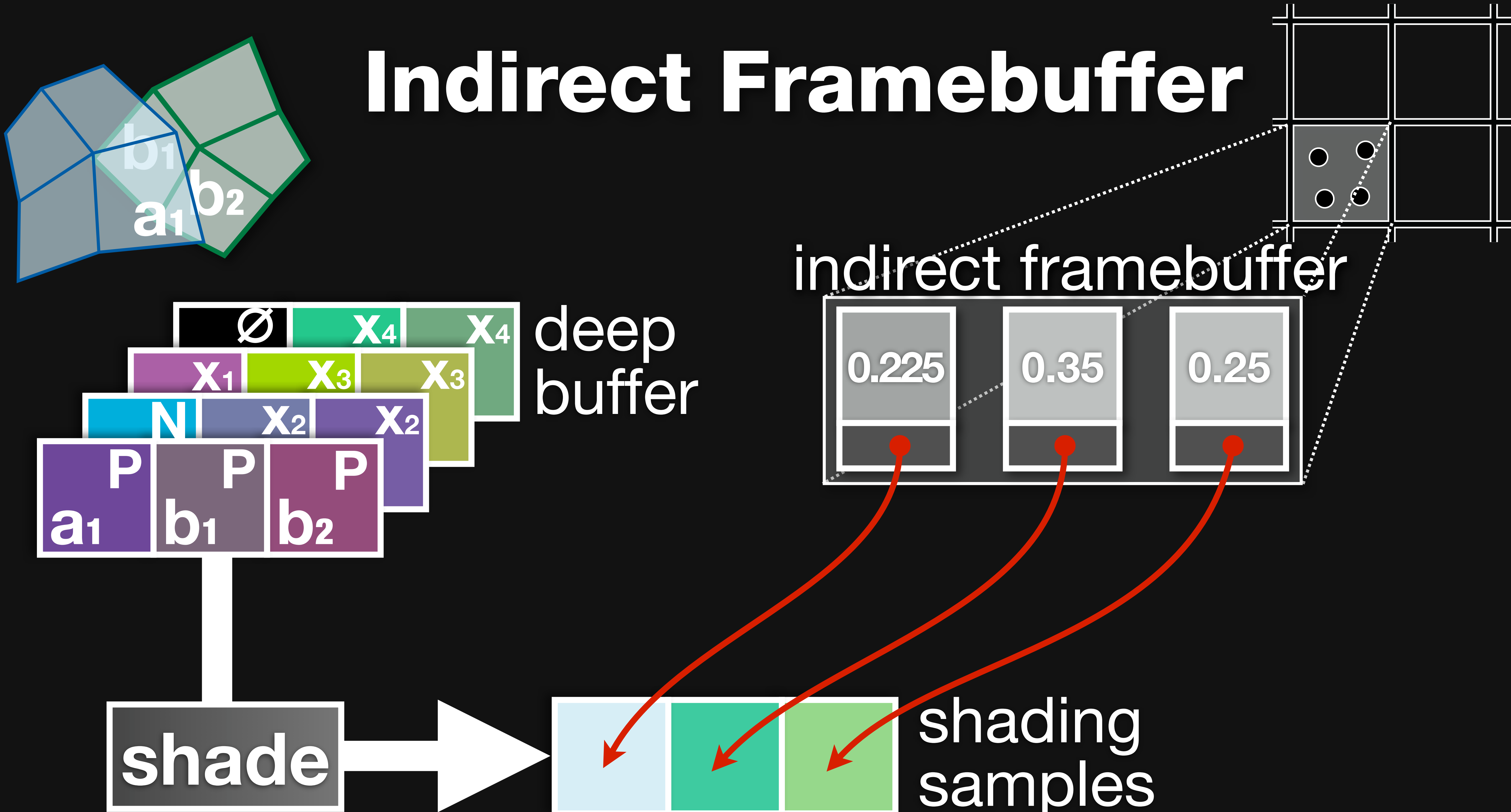


shaded like a conventional deep-framebuffer

Indirect Framebuffer

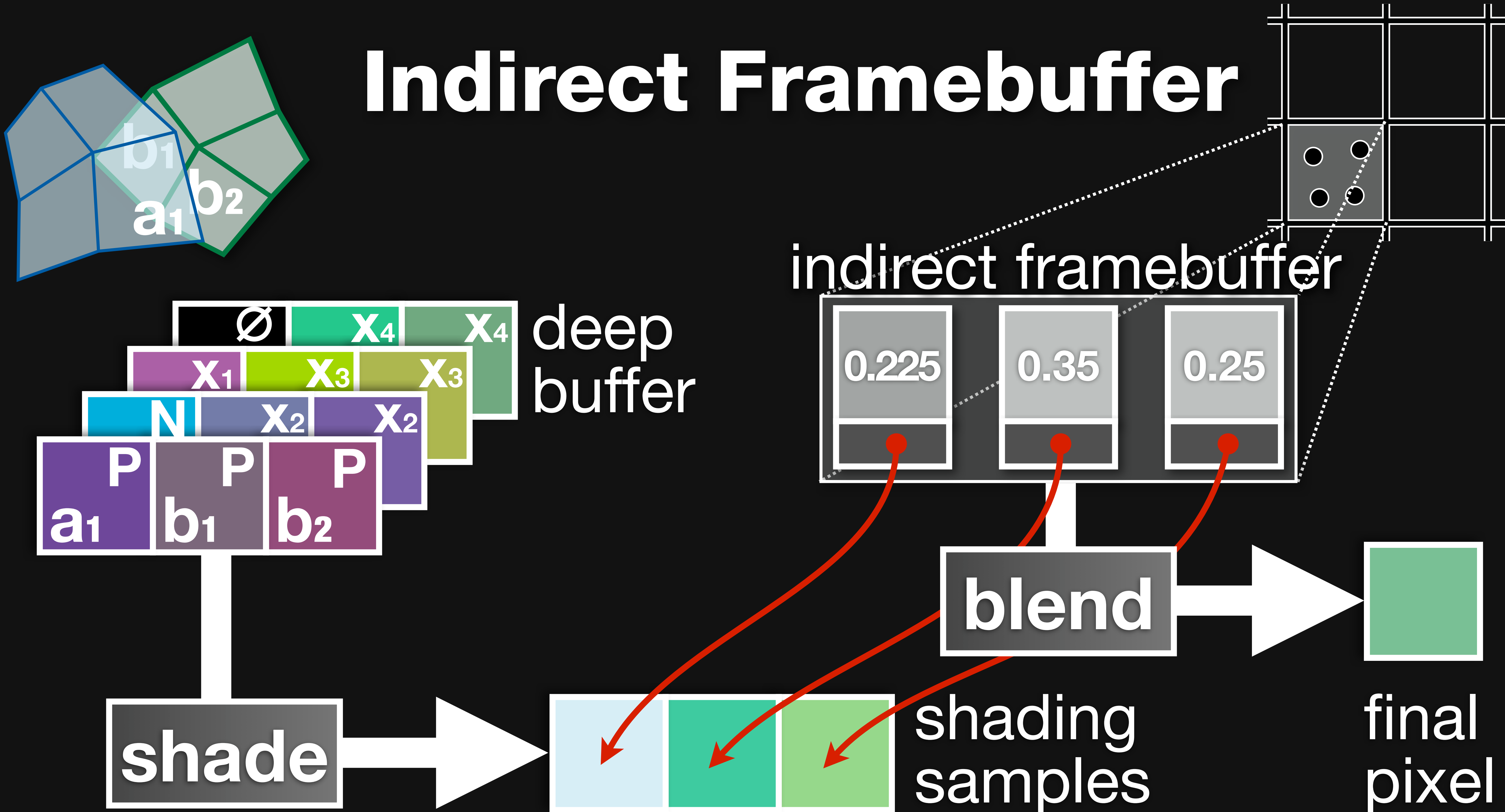


Indirect Framebuffer



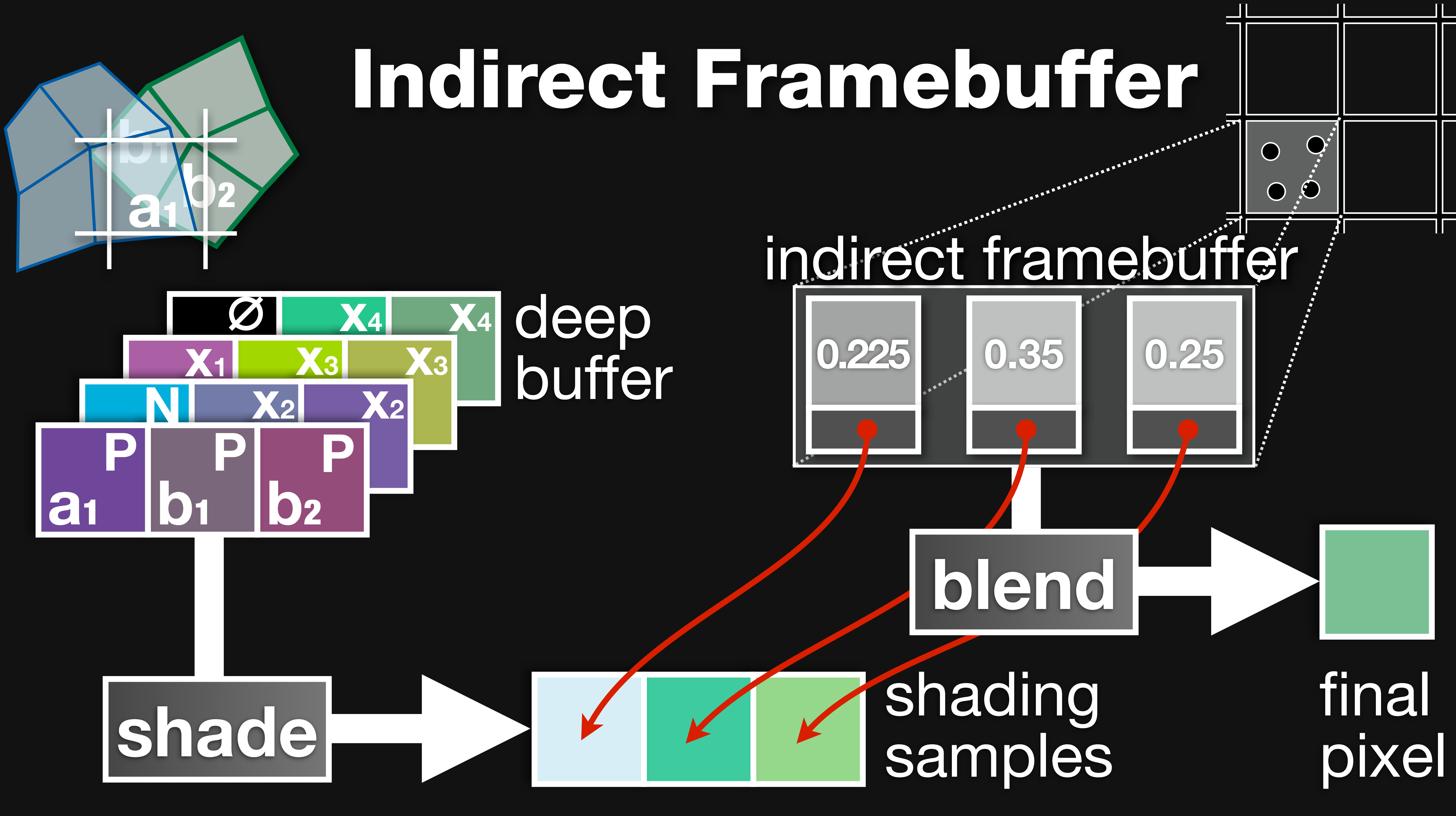
densely stored in vertex array, on GPU

Indirect Framebuffer



4 subpixel samples, 2 transparent layers
only 3 unique micropolygons

Indirect Framebuffer

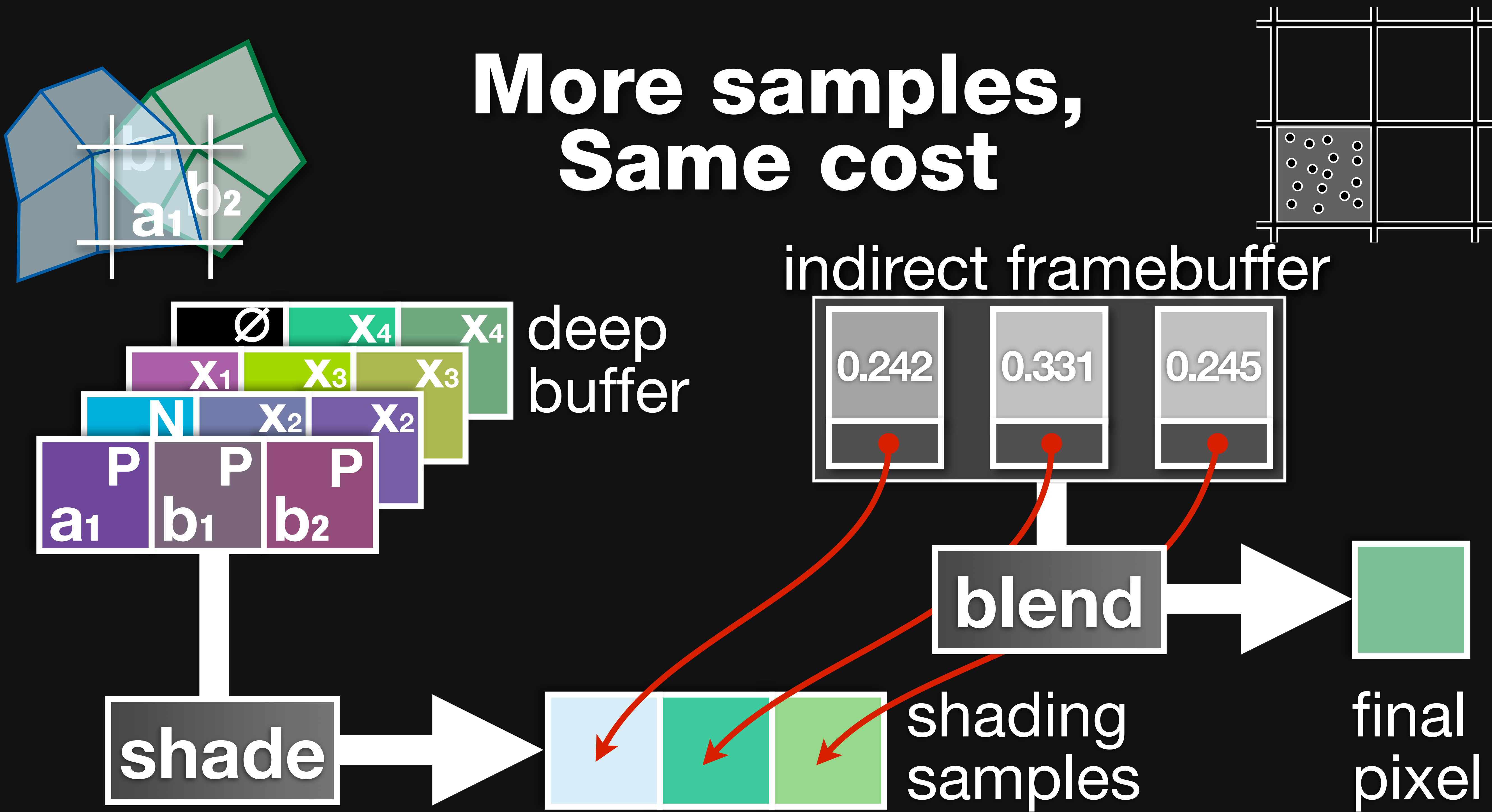


4 subpixel samples, 2 transparent layers

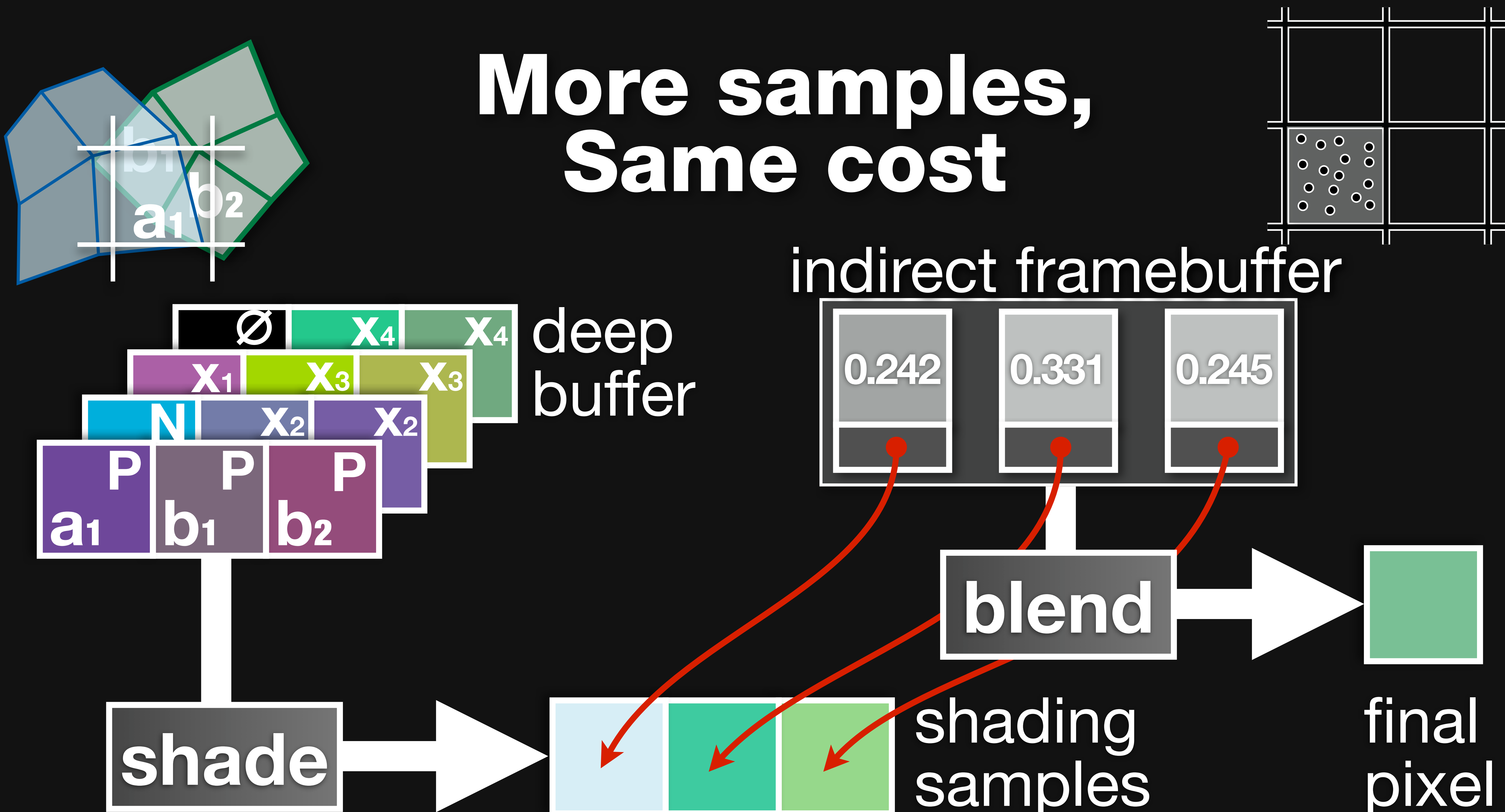
only 3 unique micropolygons

all contributions linearized into a single weight

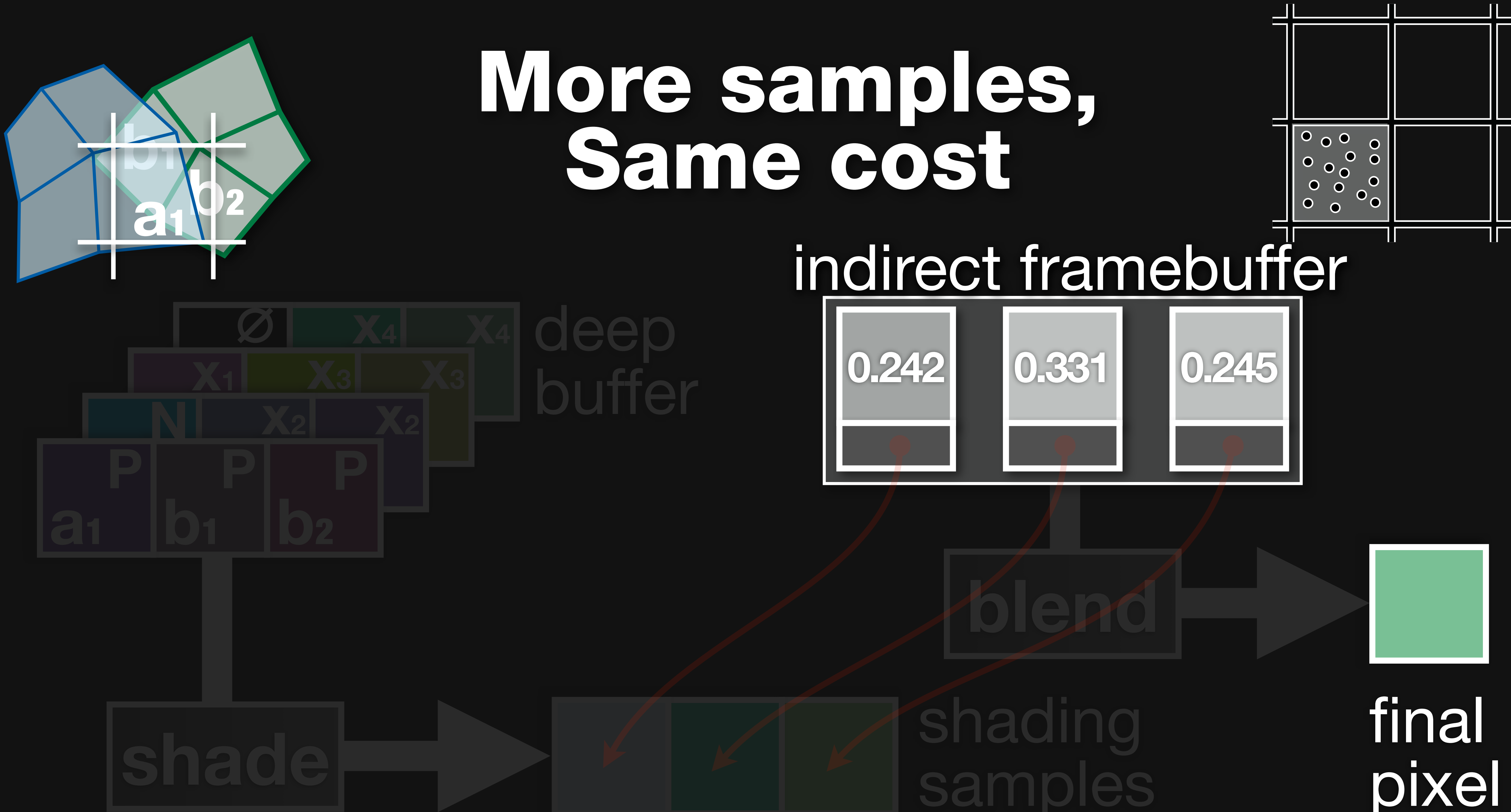
More samples, Same cost



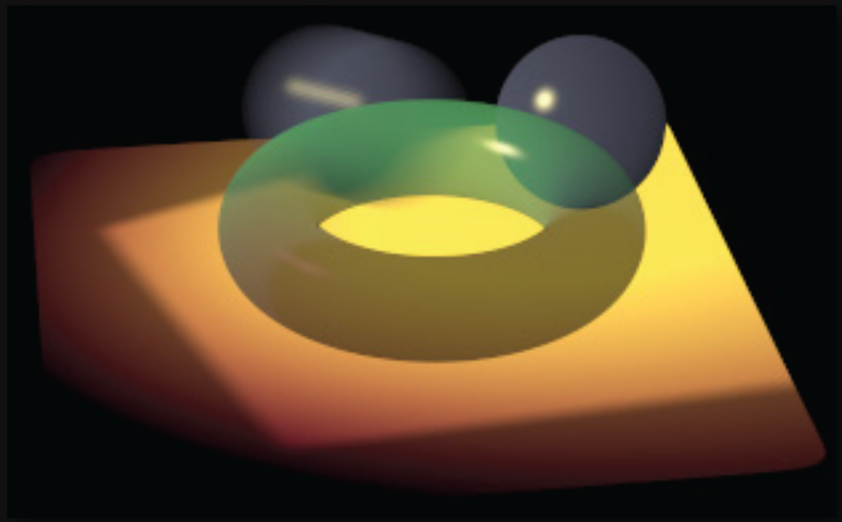
More samples, Same cost



More samples, Same cost

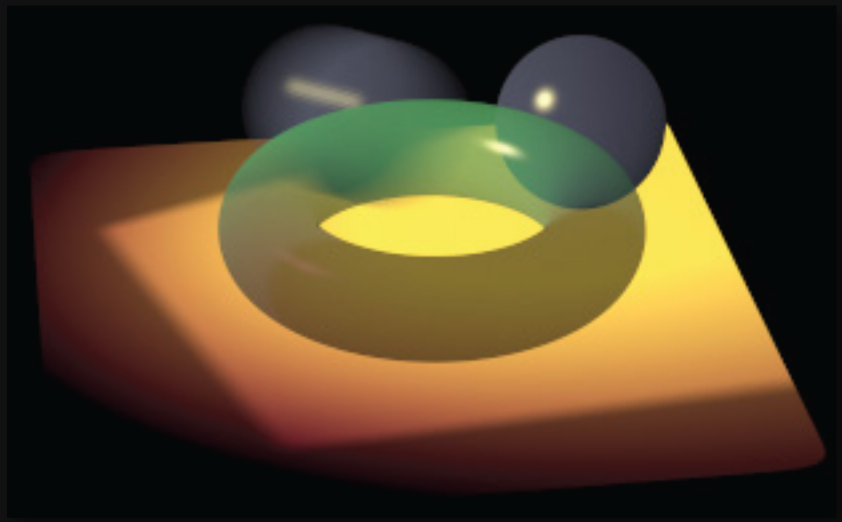


Indirect Framebuffer Results



antialiasing
motion blur
transparency
identical to RenderMan

Indirect Framebuffer Results



antialiasing
motion blur
transparency
identical to RenderMan

RenderMan
45M micropolygons
60M subpixel samples

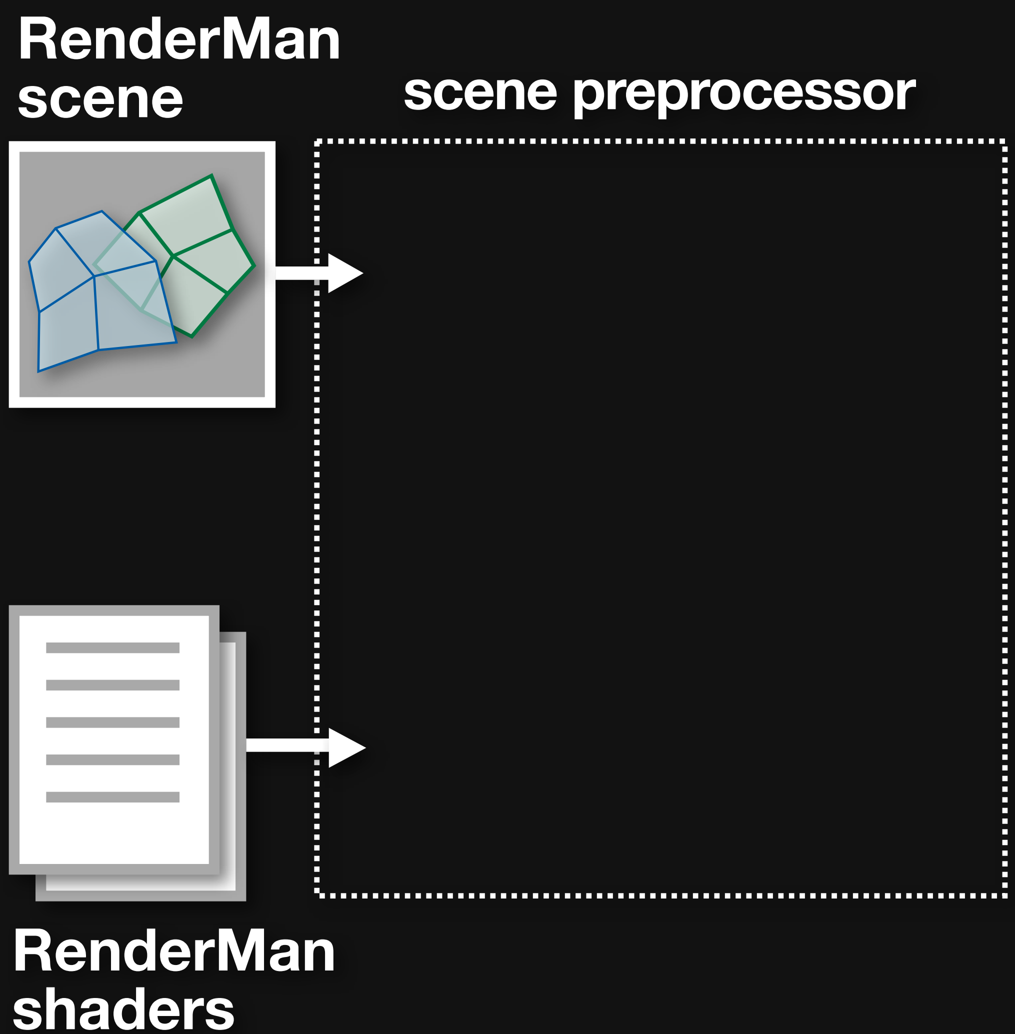
brute-force
>100M samples

Indirect framebuffer
10M deep fb
+15M indirect fb

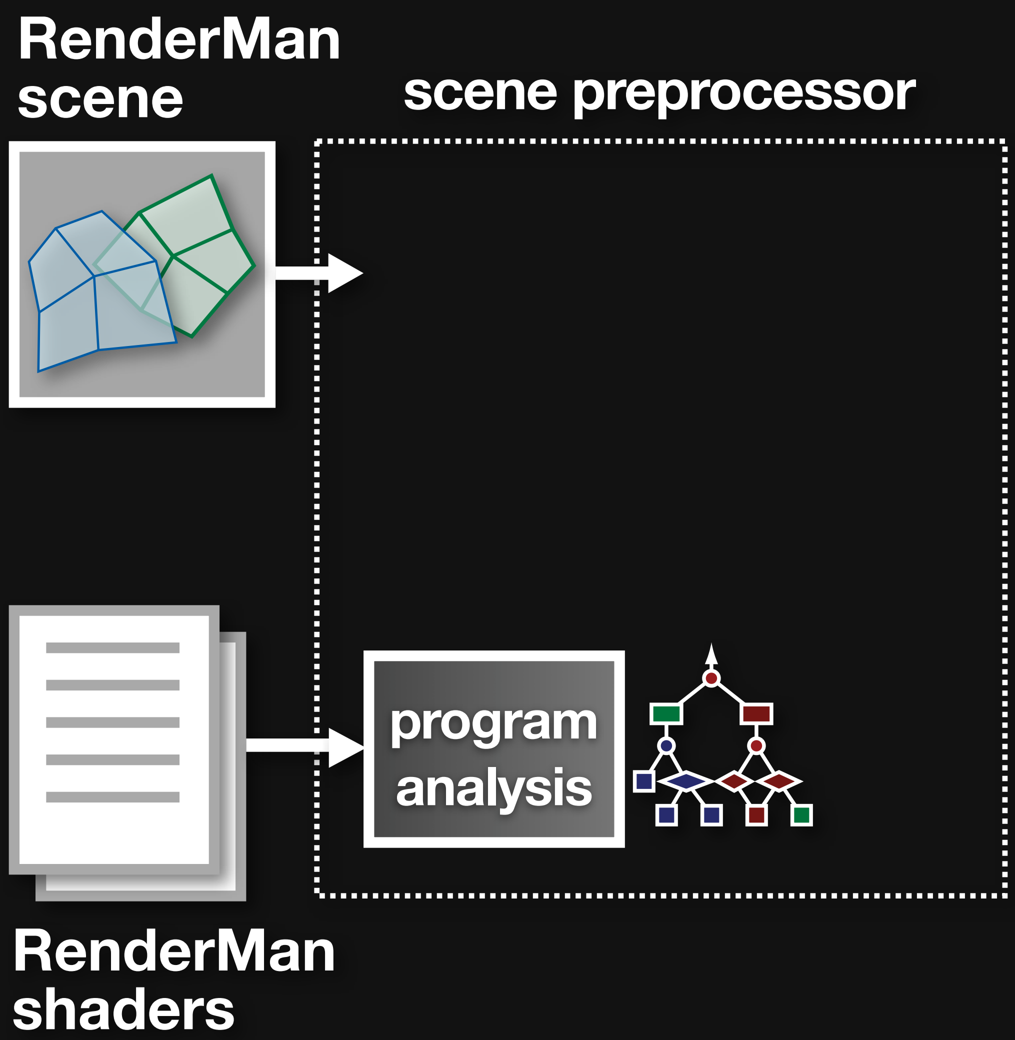
720x389, 64x supersampling

System Architecture

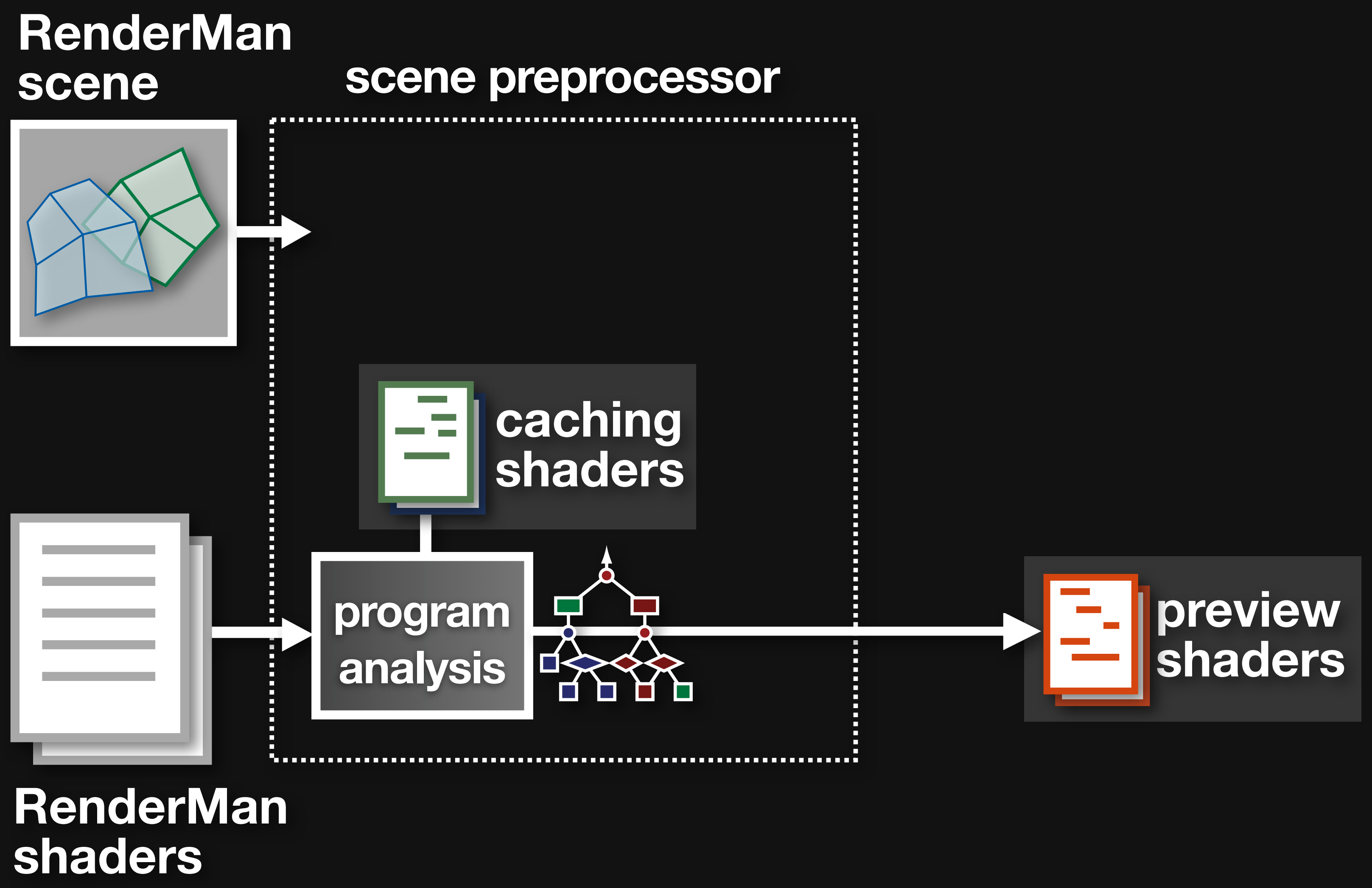
System Architecture



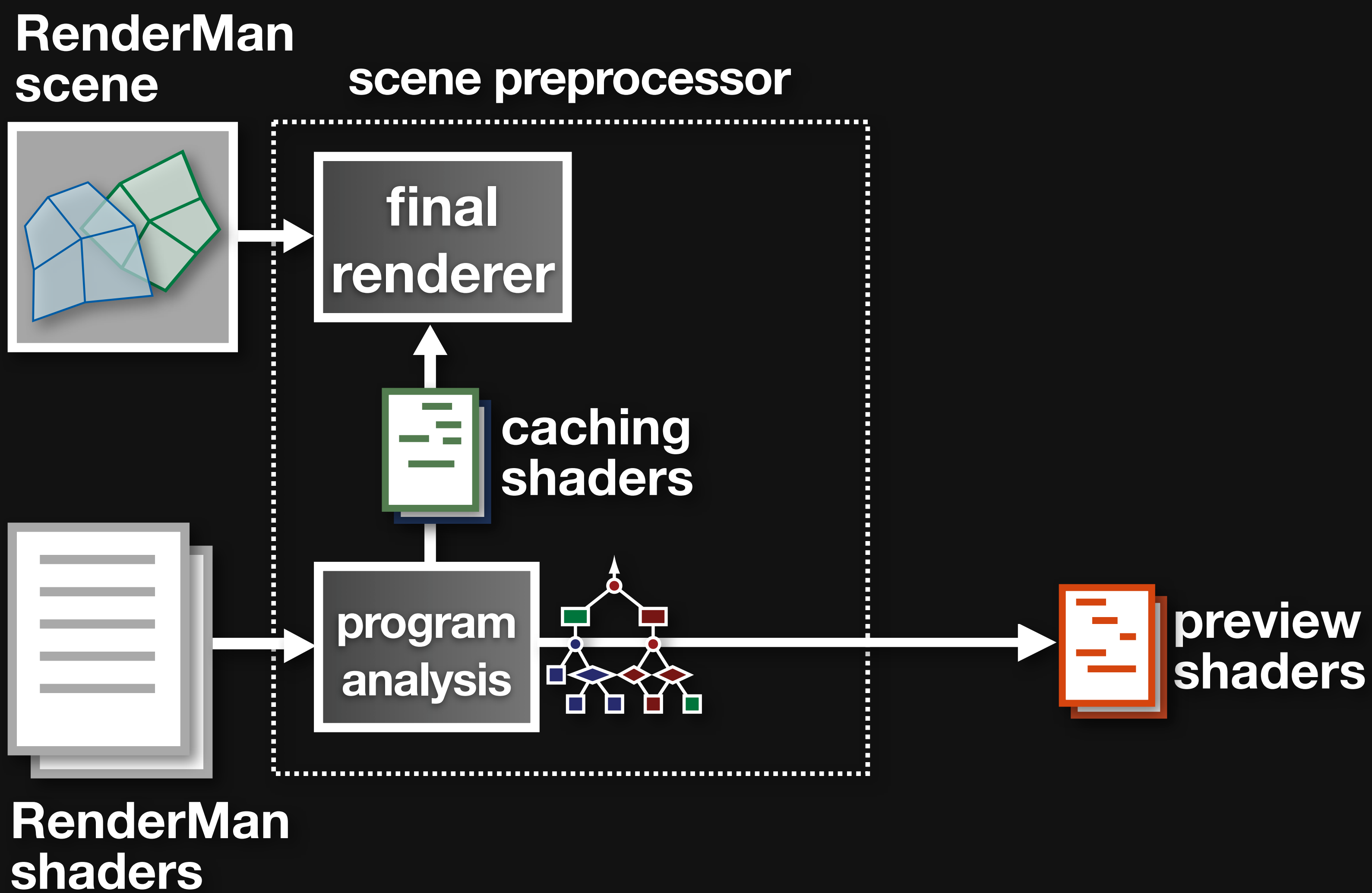
System Architecture



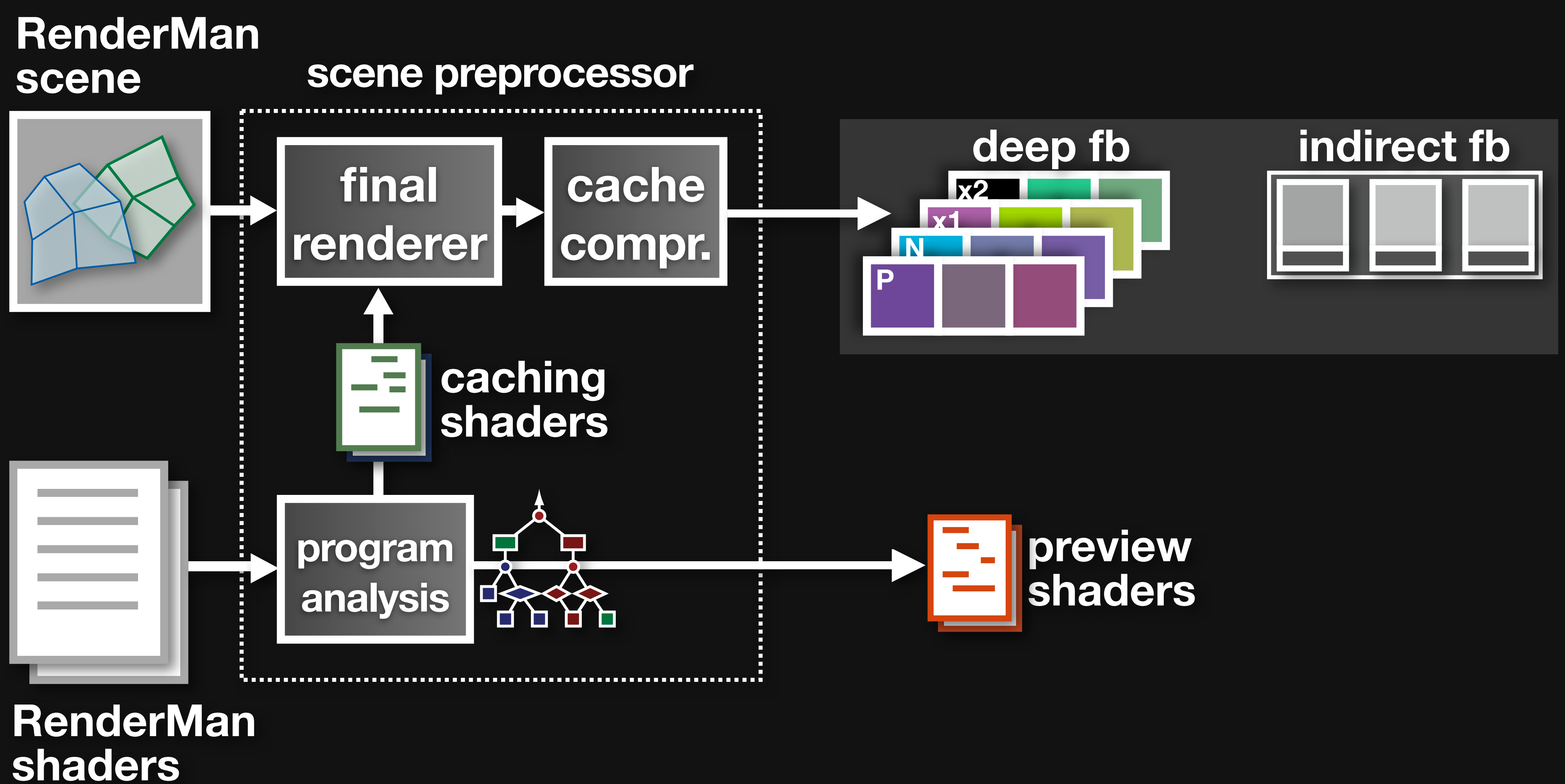
System Architecture



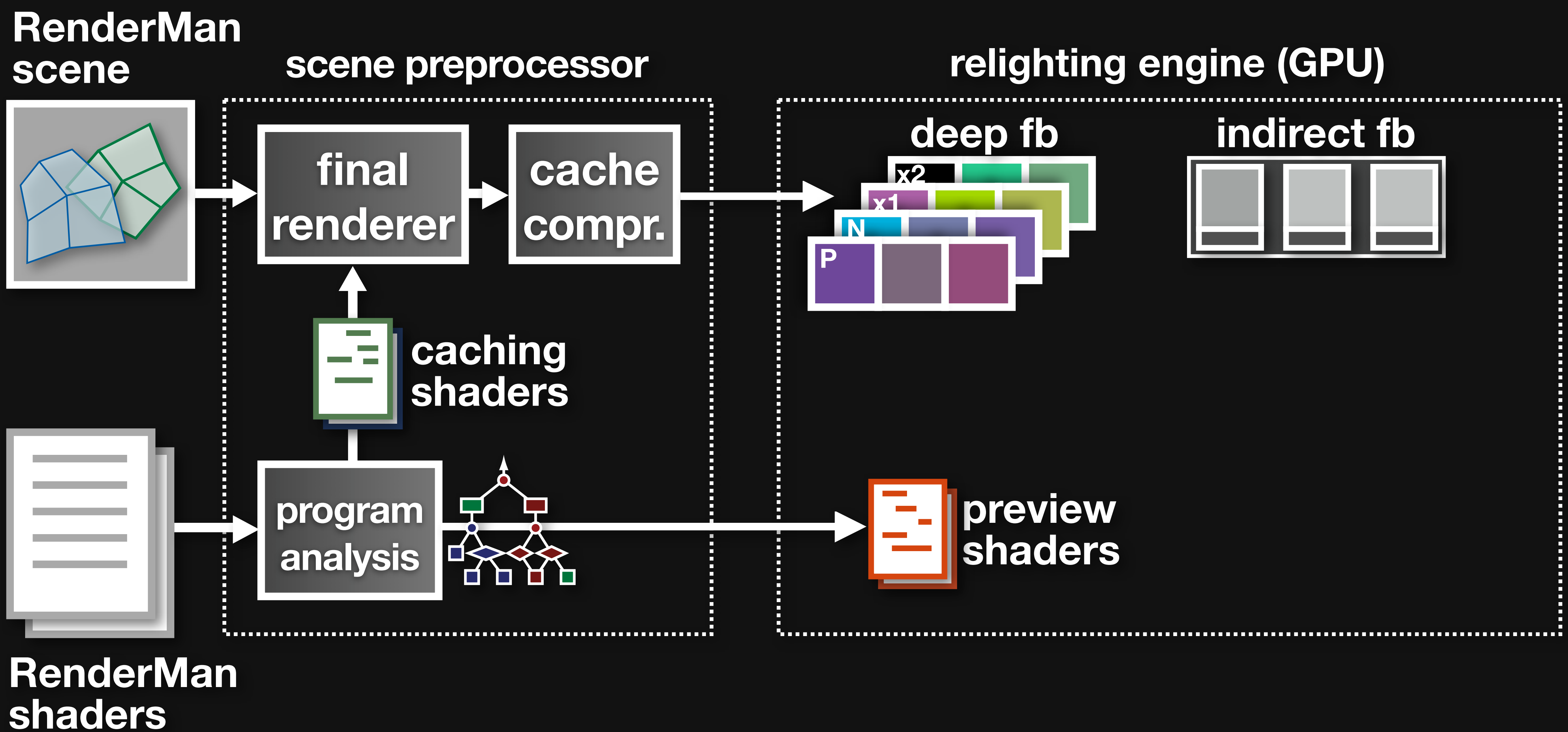
System Architecture



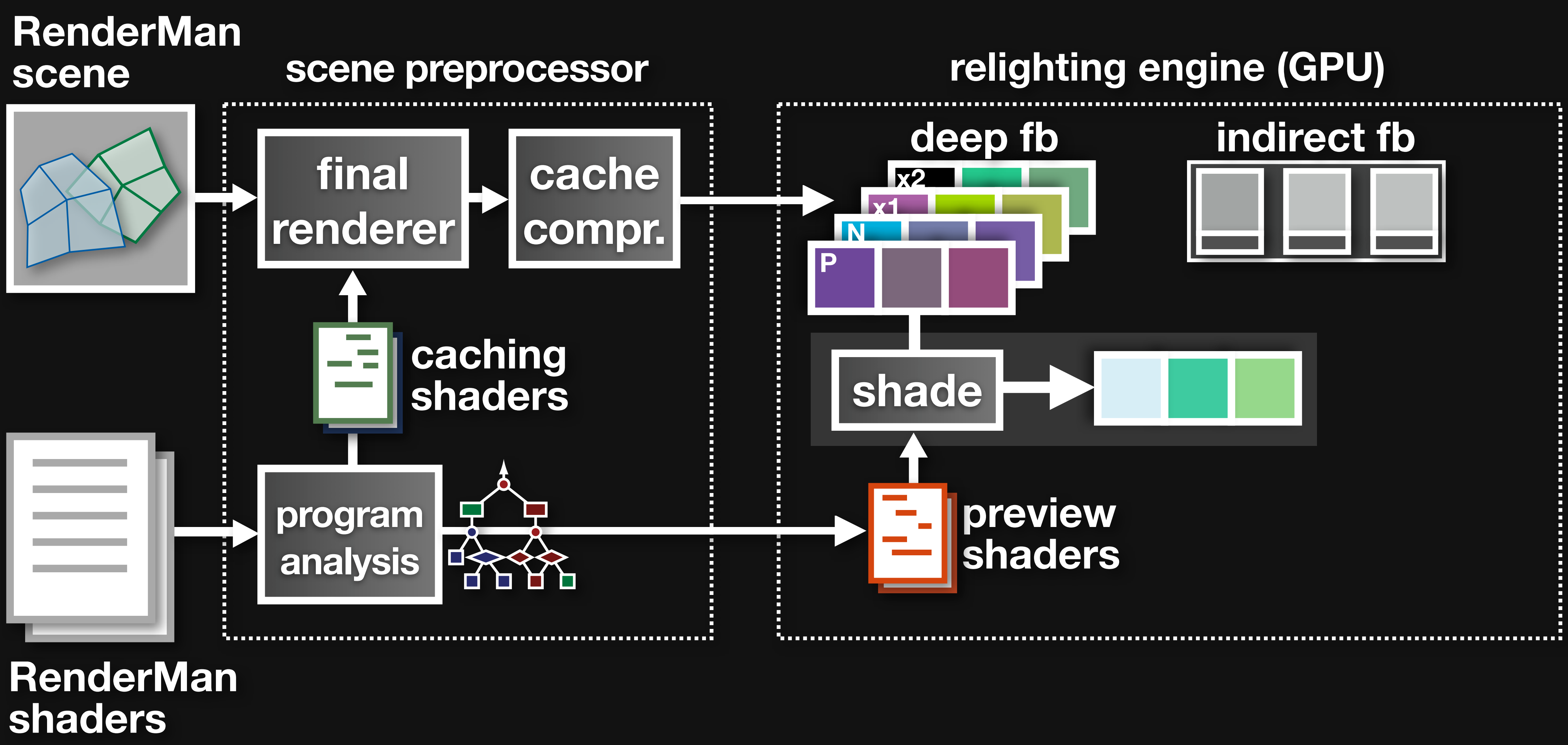
System Architecture



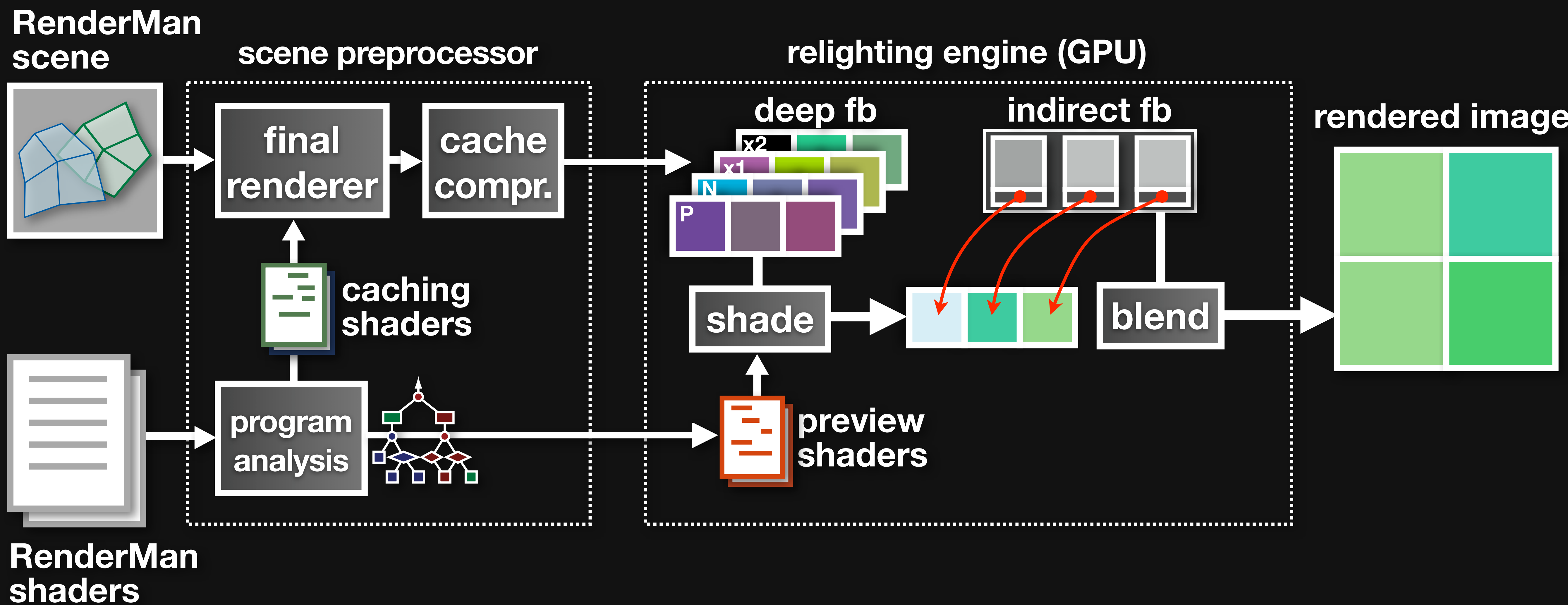
System Architecture



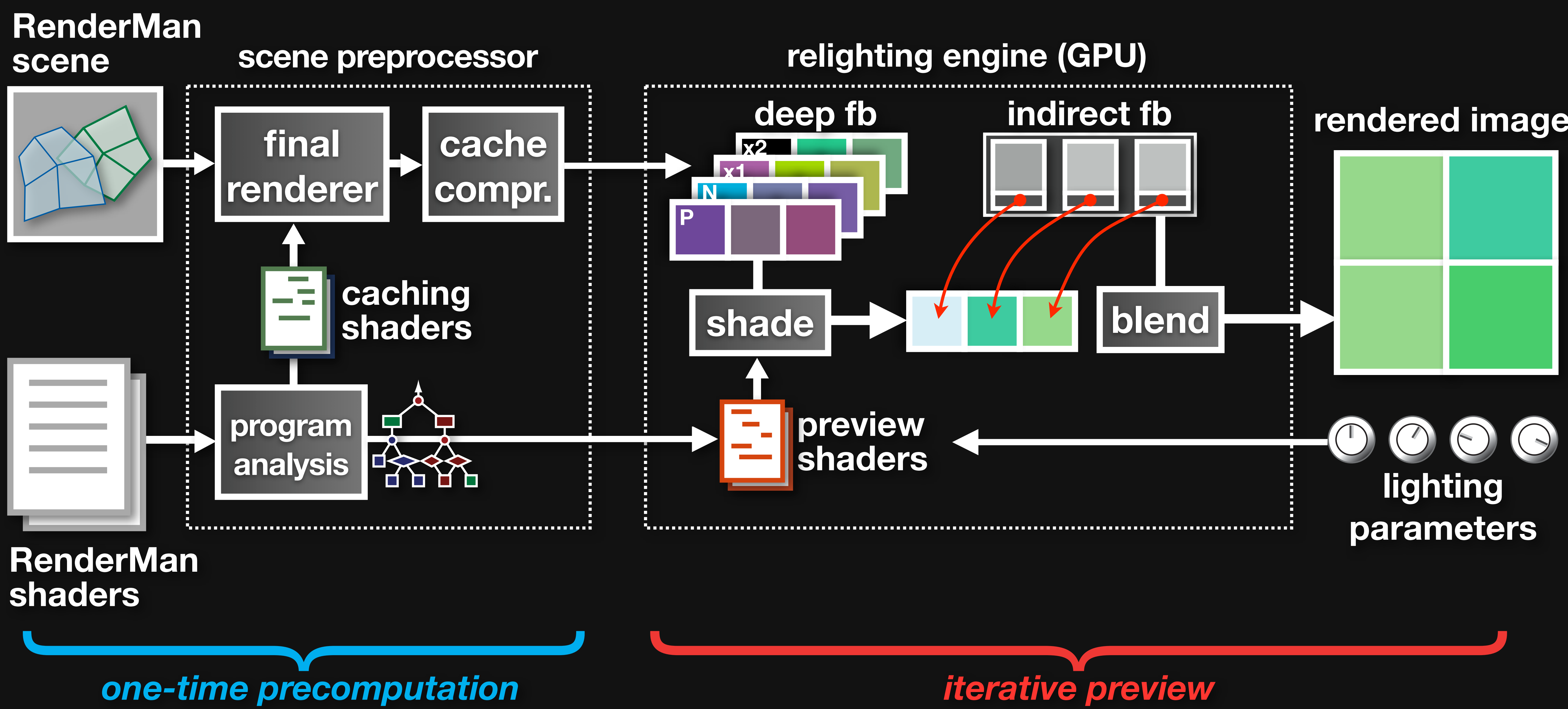
System Architecture



System Architecture



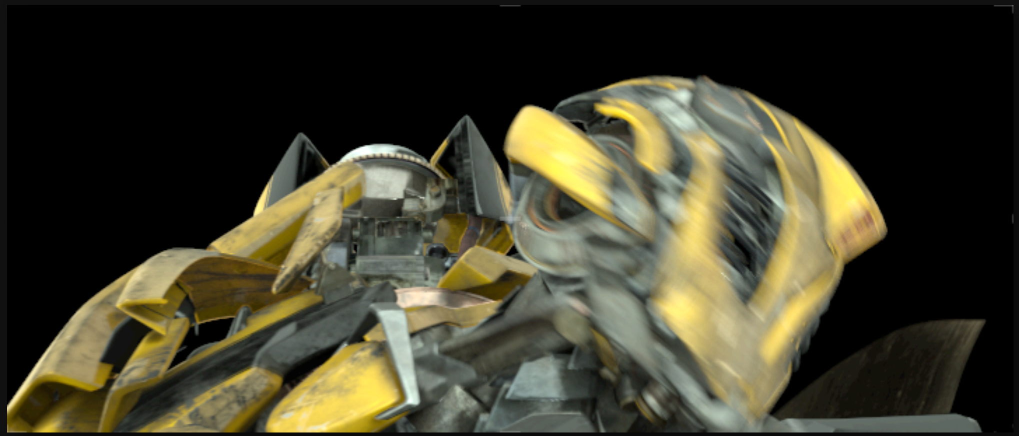
System Architecture



Additional Features

- Shadows
- Subsurface Scattering
- Performance enhancement
  - Progressive Refinement
  - Light Caching
  - Tiling

...



Results

initial feedback: 0.05 sec
full refinement: 0.7 sec

720x389

offline render: 5 mins

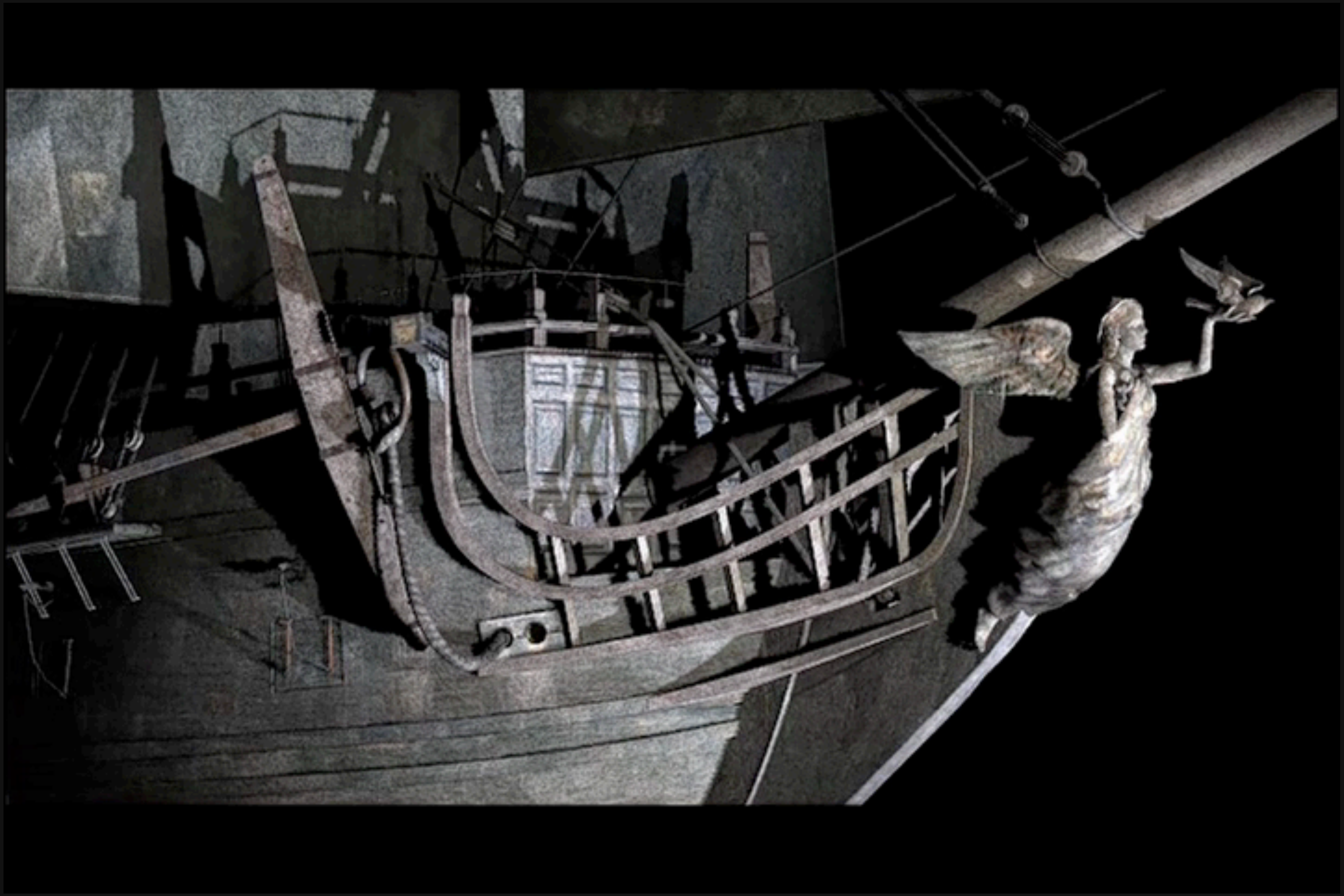
Results

initial feedback: 0.05 sec
full refinement: 0.7 sec

720x389

offline render: 5 mins

Results



Results

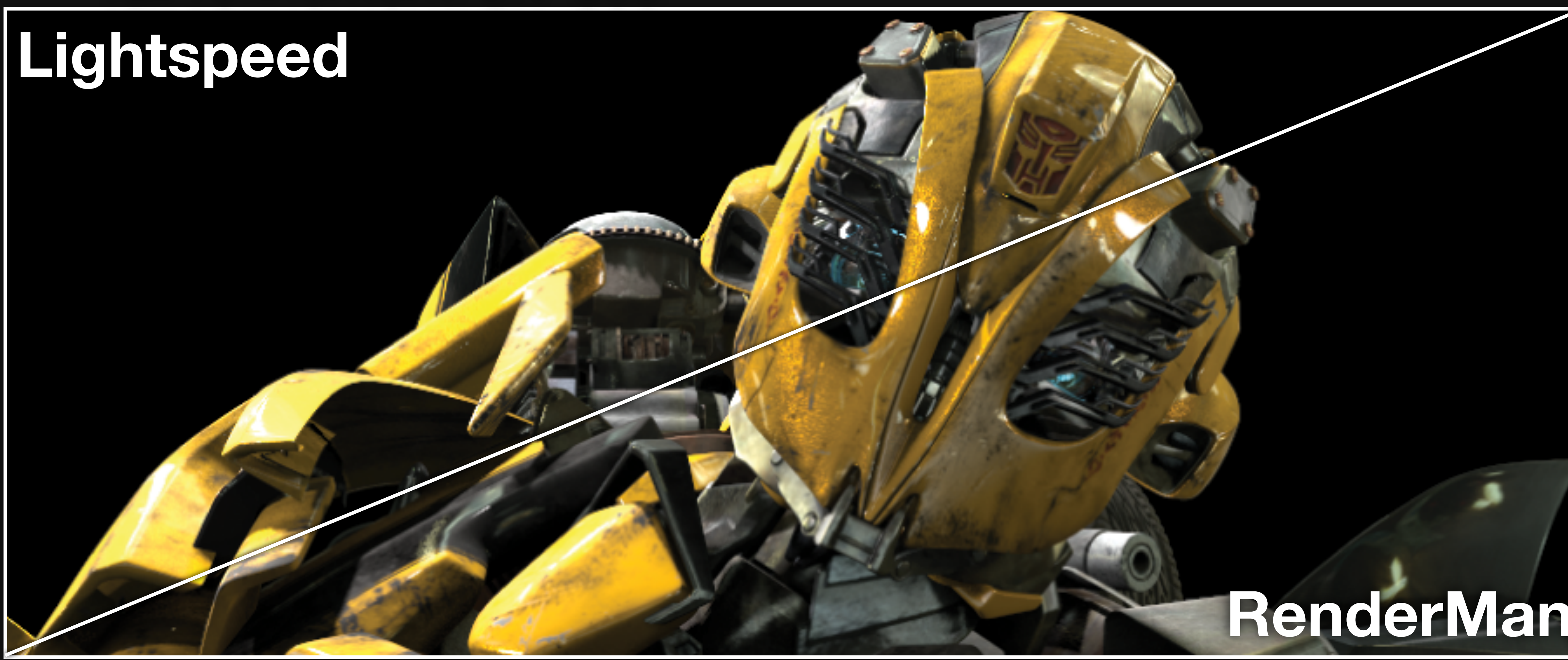
caching: 16 mins
initial feedback: 0.11 sec
full refinement: 2.7 sec

914x389
13x13 antialiasing
42 lights

offline render: 59 mins

Results

caching: 16 mins
initial feedback: 0.11 sec
full refinement: 2.7 sec



914x389
13x13 antialiasing
42 lights

offline render: 59 mins

Results

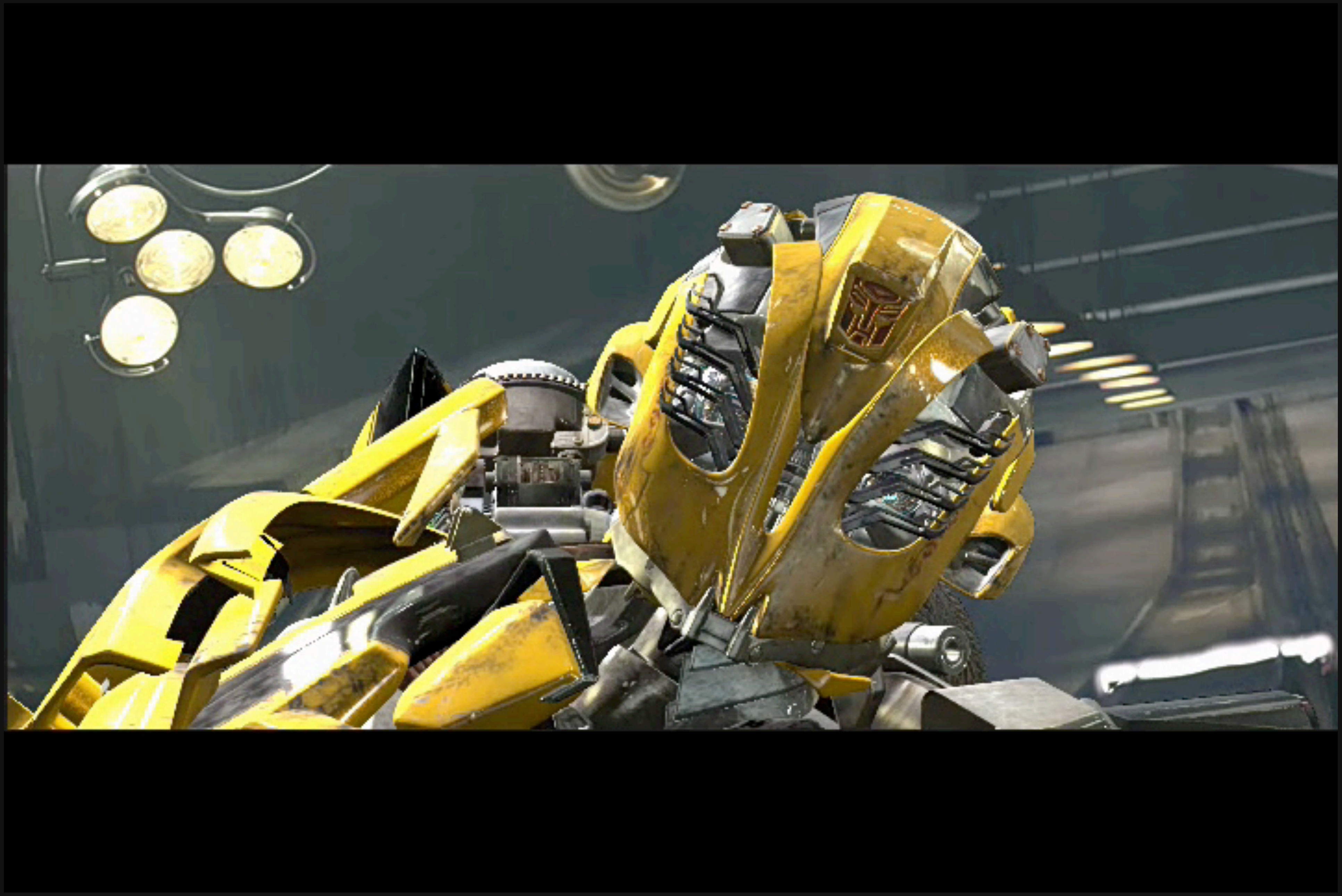
caching: 16 mins
initial feedback: 0.11 sec
full refinement: 2.7 sec

914x389
13x13 antialiasing
42 lights

offline render: 59 mins

Results

Results



Results

Results

Limitations

- **GPU programming model**
  - **Dynamic calls to external C code**
  - **Complex data structures**
  - **GPU limits** (bandwidth, memory, registers)
- **Fully-accurate dynamic ray tracing**
- **Unbounded dynamic loops**
- **Additional features not yet implemented**
  - **Indirect diffuse**
  - **Deep shadows**
  - **Non-linear lights**

Discussion

- **Automatic and manual specialization (Lpics) both have advantages**
  - **Manual specialization allows hand optimization**
  - **Compiler requires up-front R&D, never perfect** (especially on the GPU)
  - **Saves significant time in production**
  - **Some material parameters are editable**

- **Cache compression is key to practical automatic specialization**
  much simpler than fancy static analysis

- **Indirect Framebuffer is powerful, scalable**

Summary

- **Interactive lighting preview**
  milliseconds instead of hours

- **Automatic caching** for our production scenes
  - **Program analysis**
  - **Cache compression**

- **Indirect framebuffer**
  - **Efficient antialiasing, motion blur, transparency**
  - **Progressive refinement**
- **In use on current productions**

Acknowledgments

Inception: Pat Hanrahan, Ujval Kapasi

Compilers: Alex Aiken, John Kodumal

Tippett: Aaron Luk, Davey Wentworth

ILM: Alan Trombla, Ed Hanway, Dan Goldman, Steve Sullivan, Paul Churchill

Images: Michael Bay, Dan Piponi

Money: NSF, NVIDIA, MSR, Sloan & Ford fellowships

Summary

- **Interactive lighting preview**
  milliseconds instead of hours

- **Automatic caching** for our production scenes
  - **Program analysis**
  - **Cache compression**

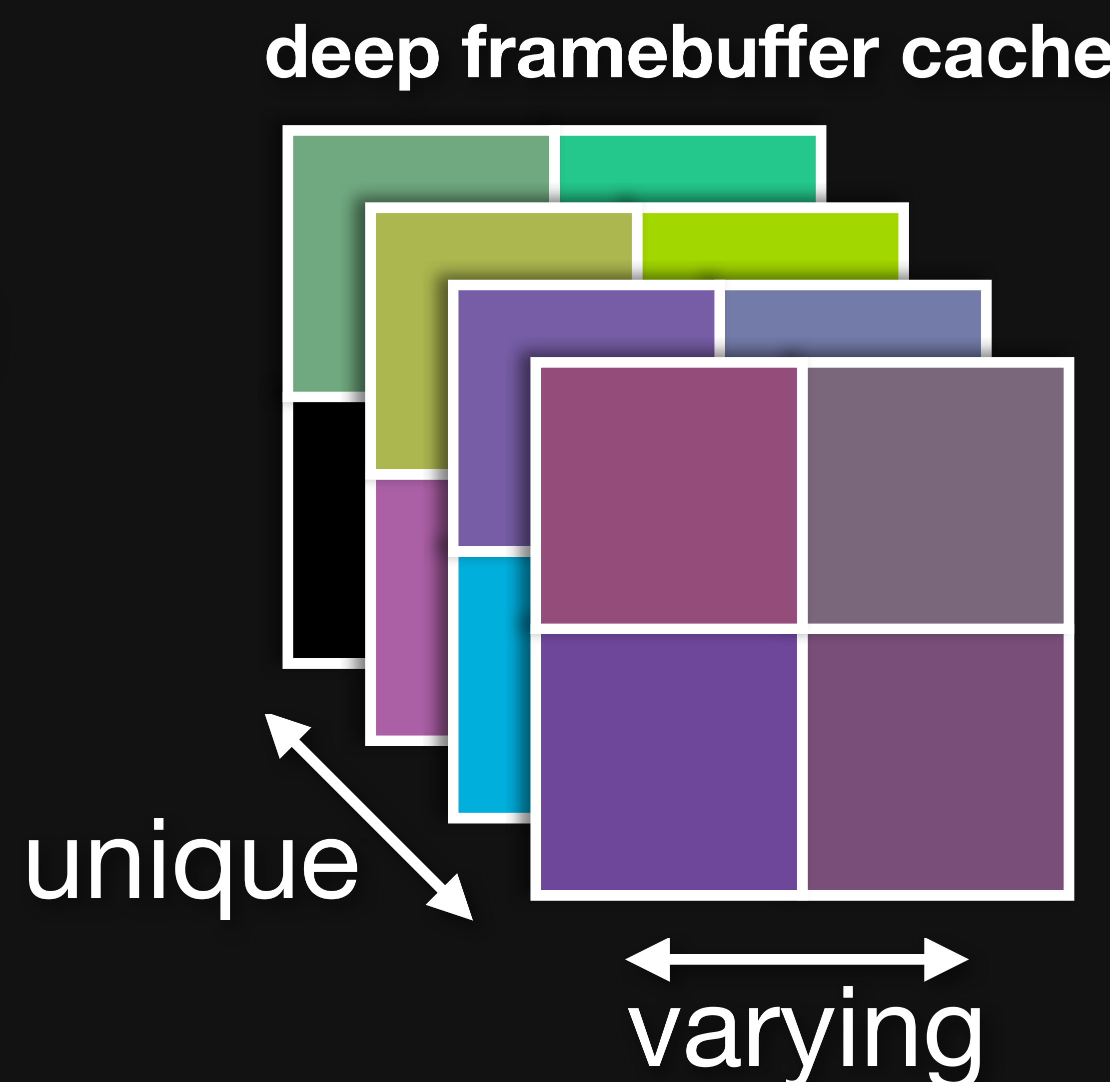
- **Indirect framebuffer**
  - **Efficient antialiasing, motion blur, transparency**
  - **Progressive refinement**
- **In use on current productions**

EXTRAS

Cache Compression

Remove redundant values *after* caching

- **non-varying**
  same for many pixels
- **non-unique**
  same within a pixel



→ ***4-5x compression***

Cache Compression

shader	dynamic *(static analysis)*	varying	**unique** *(compressed)*
generic surface	402	145	***97***
metallic paint	450	150	***97***

Visibility Compression

			RenderMan		Indirect Framebuffer	
scene	res.	samples	shade	subpix	shade	indirect
robot	914x389	13x13	2.1M	32M	633k	1.6M
robot (blur)	720x306	13x13	1.5M	21M	467k	3.8M
pirate	640x376	4x4	2.5M	2.3M	327k	716k
hairs	720x389	8x8	43M	58M	11M	17M