Vectorized Production Path Tracing

Mark Lee

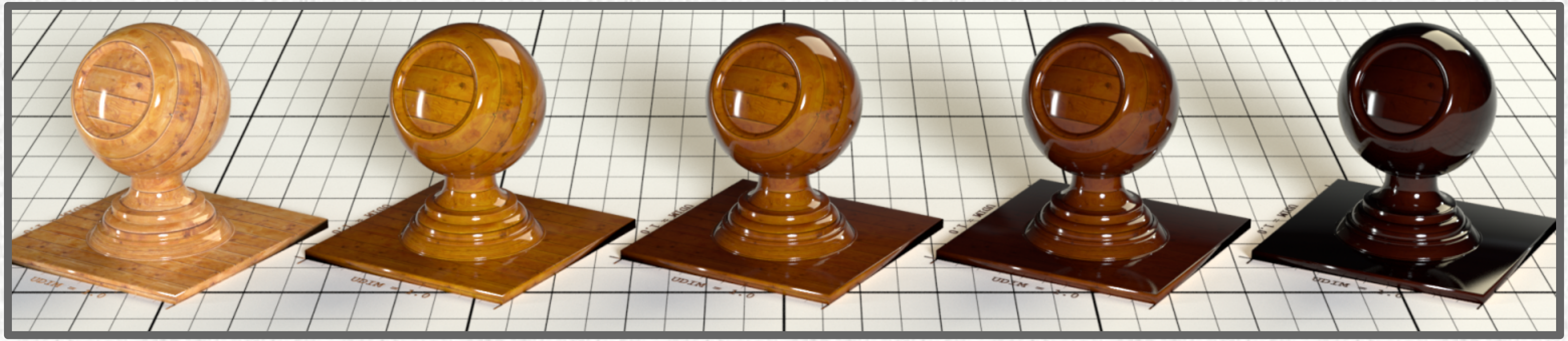
Brian Green

Feng Xie

Eric Tabellion

DreamWorks Animation

Hello MoonRay




CatmulClark
Importance
ray
patterns
controls
control
sampling
noise
subds
Materials
differentials
exr
shading
Fully
Bssrdf
uv
Ascii
Bezier
Native
Light
lights
mapping
Path
efficient
types
hair
input
integrator
Aniso
Iso
Camera
curves
pixel
tracing
scene
Bsdf
blur
FabricLegacyMaterial
HairMap
layering
custom
Multipart
description
AshikminhShirley
DiskLight
Deformation
Binary
randomize
Progressive
scripting
Perspective
discrepancy
filtering
Lua
Schlick
Lambert
CylinderLight
OIIO
Readable
glossy
R
write
based
good
diffusion
Geometry
Normal
SpotLight
stats
motion
Samples
Displacement
Russian
triangles
ImageMap
output
Transform
depth
RectLight
roulette
UDIM
exposure
DebugMap
reproducible
wheels
format
mirror
falloff
Dipole
GGX
TRT
propellers
Polymeshes
language
rotation
Dielectric
Renderer
DistantLight
Ward
Closures
sampled
approximation
Gamma
textures
masking
core
rdla
Robust
Loaders
updates
attributes
Logs
Mm
tx
filter
segment
etc
Sidedness
Tessellated
Fresnel
frames
BaseMaterial
MeasuredMaterial
Freeze
CookTorrance
quality
texture
Single
partial
Beckmann
artifact
Mc
Uniform
simple
colorcorrection
WardDÃ¼er
threadsafe
Superellipse
LightSets
using
driver
Abc
Surface
rdlb
splitting
Samplers
raytracing
Base
Energy
EnvLight
Saturation
Formatted
image
High
Primitives
20
shape
Conductor
standard
watertight
Maps
shader
TME
Use
angles
Radial
rates
Multiple
cone
Motionblur
tiled
Intel
correlation
Inner
Tip
HairMaterial
Embree
Rendering
TT
Primitive
interpolation
Nonprogressive
Realtime
tessellation
Color
csv
kernels
HDR
Depthoffield
diffuse
conservation
Mipmapped
intensity
Nukelike
Glint
SphereLight

Hello MoonRay

MoonRay Overview

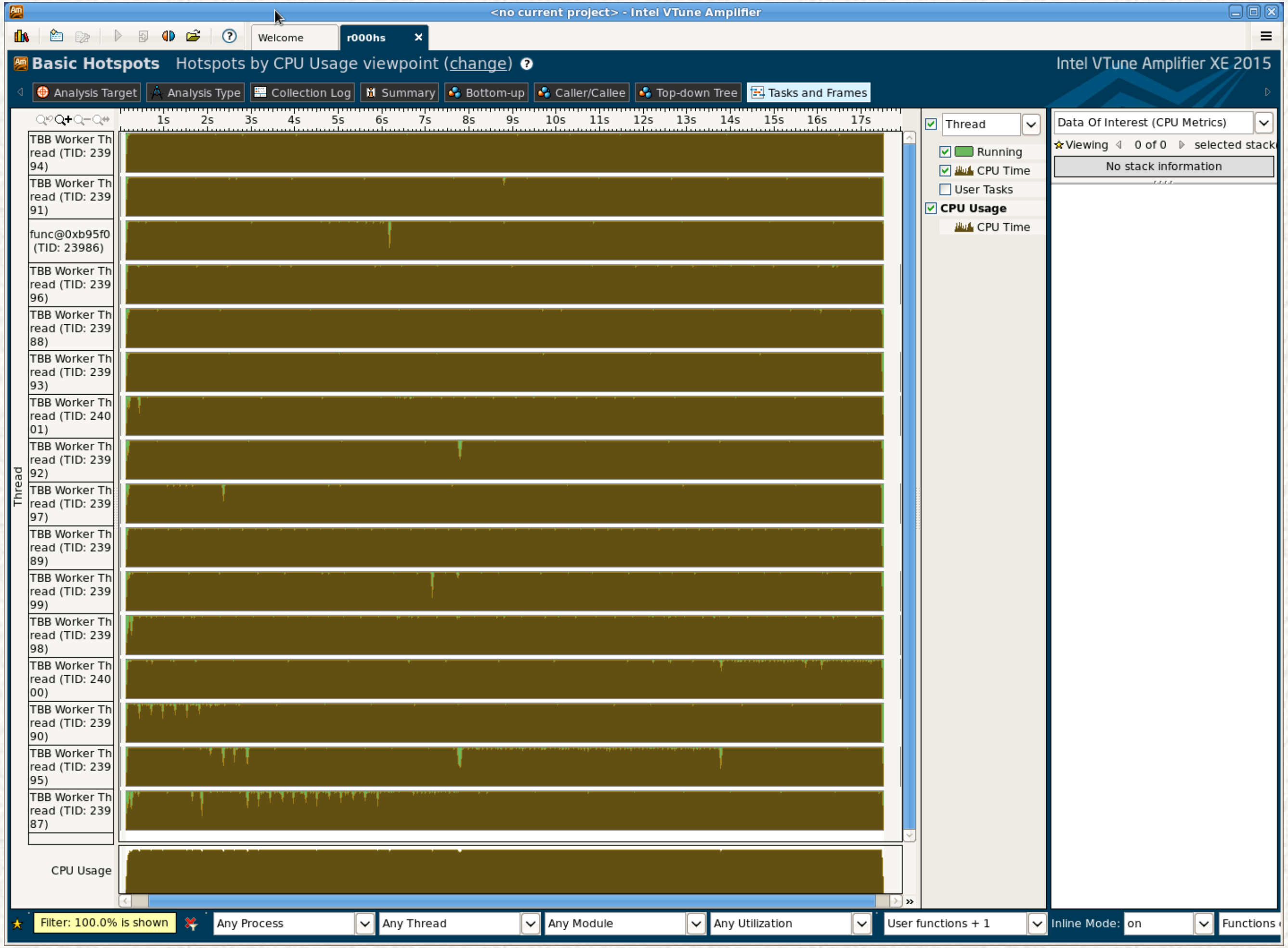
- Single executable with internal scalar and vectorized code paths
- Both code paths produce identical output images

Data Oriented Design

- Nothing new, AAA game studios have always done this
- Approach the problem from the point of view of the hardware
  - … which means caring greatly about data access patterns
- Lower level data structures are flat and uncomplicated

How would it change your approach to coding if main memory access was 10x slower?

Threading

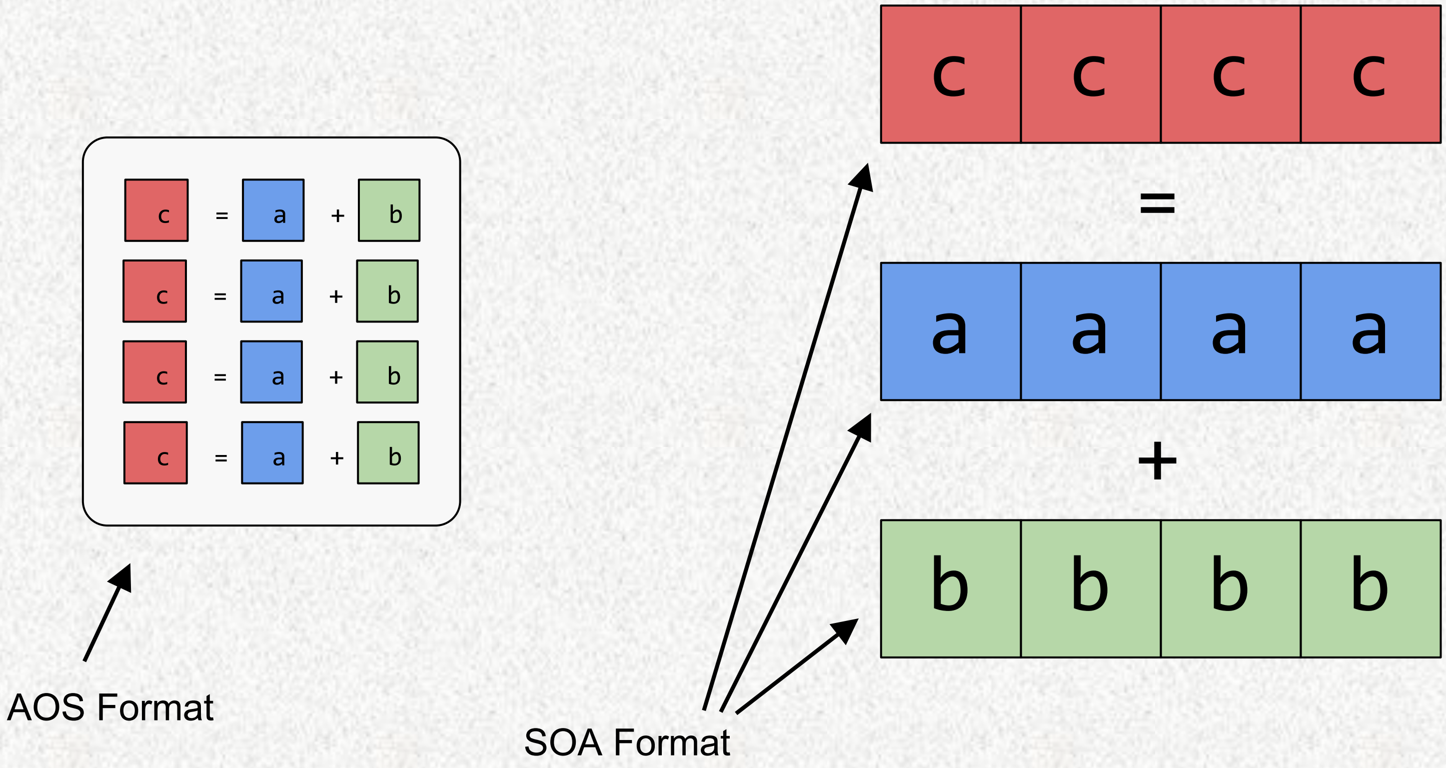

<no current project> - Intel VTune Amplifier
Welcome
r000hs
Basic Hotspots Hotspots by CPU Usage viewpoint (change)
Intel VTune Amplifier XE 2015
Analysis Target
Analysis Type
Collection Log
Summary
Bottom-up
Caller/Callee
Top-down Tree
Tasks and Frames
1s 2s 3s 4s 5s 6s 7s 8s 9s 10s 11s 12s 13s 14s 15s 16s 17s
Thread
TBB Worker Thread (TID: 23994)
TBB Worker Thread (TID: 23991)
func@0xb95f0 (TID: 23986)
TBB Worker Thread (TID: 23996)
TBB Worker Thread (TID: 23988)
TBB Worker Thread (TID: 23993)
TBB Worker Thread (TID: 24001)
TBB Worker Thread (TID: 23992)
TBB Worker Thread (TID: 23997)
TBB Worker Thread (TID: 23989)
TBB Worker Thread (TID: 23999)
TBB Worker Thread (TID: 23998)
TBB Worker Thread (TID: 24000)
TBB Worker Thread (TID: 23990)
TBB Worker Thread (TID: 23995)
TBB Worker Thread (TID: 23987)
CPU Usage
Thread
Running
CPU Time
User Tasks
CPU Usage
CPU Time
Data Of Interest (CPU Metrics)
Viewing 0 of 0 selected stack
No stack information
Filter: 100.0% is shown
Any Process
Any Thread
Any Module
Any Utilization
User functions + 1
Inline Mode: on
Functions

Keep all vector lanes of all cores busy all the time with meaningful work!

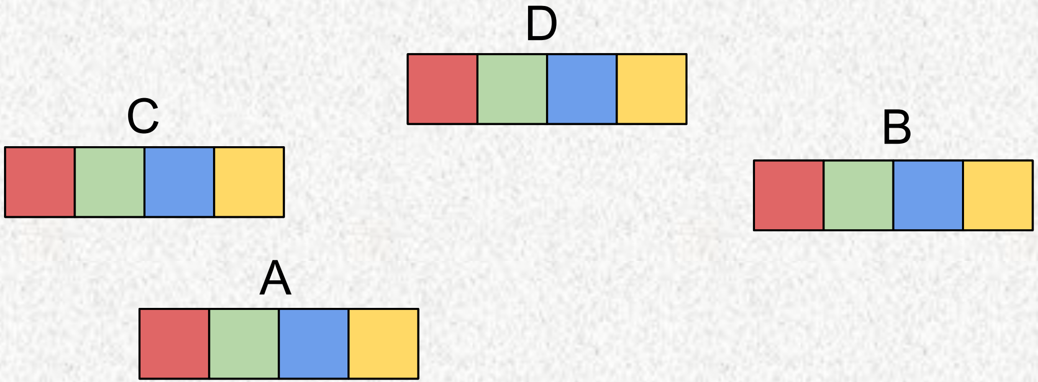
Keep all vector lanes busy….

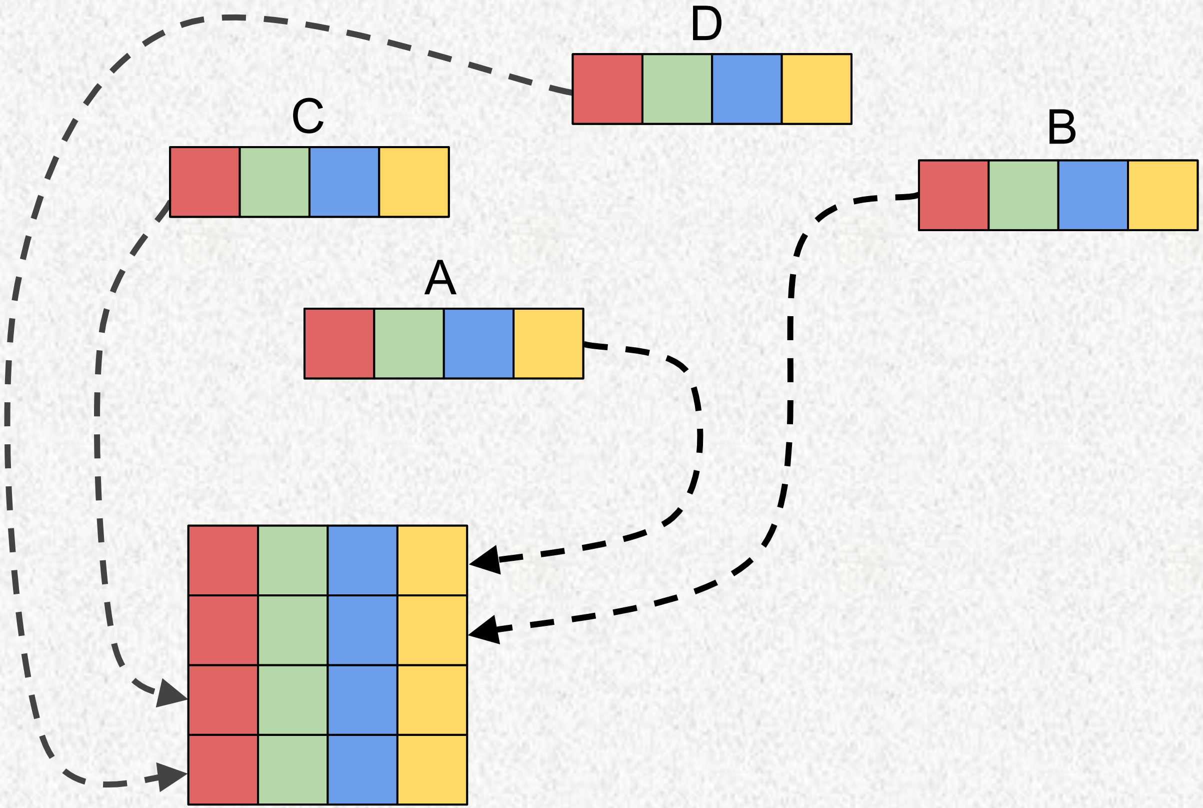
- How can we use the vector hardware effectively?
- How can we gather batches of work effectively?
- How can we minimize control flow divergence in vector code?
- How can we access memory effectively?

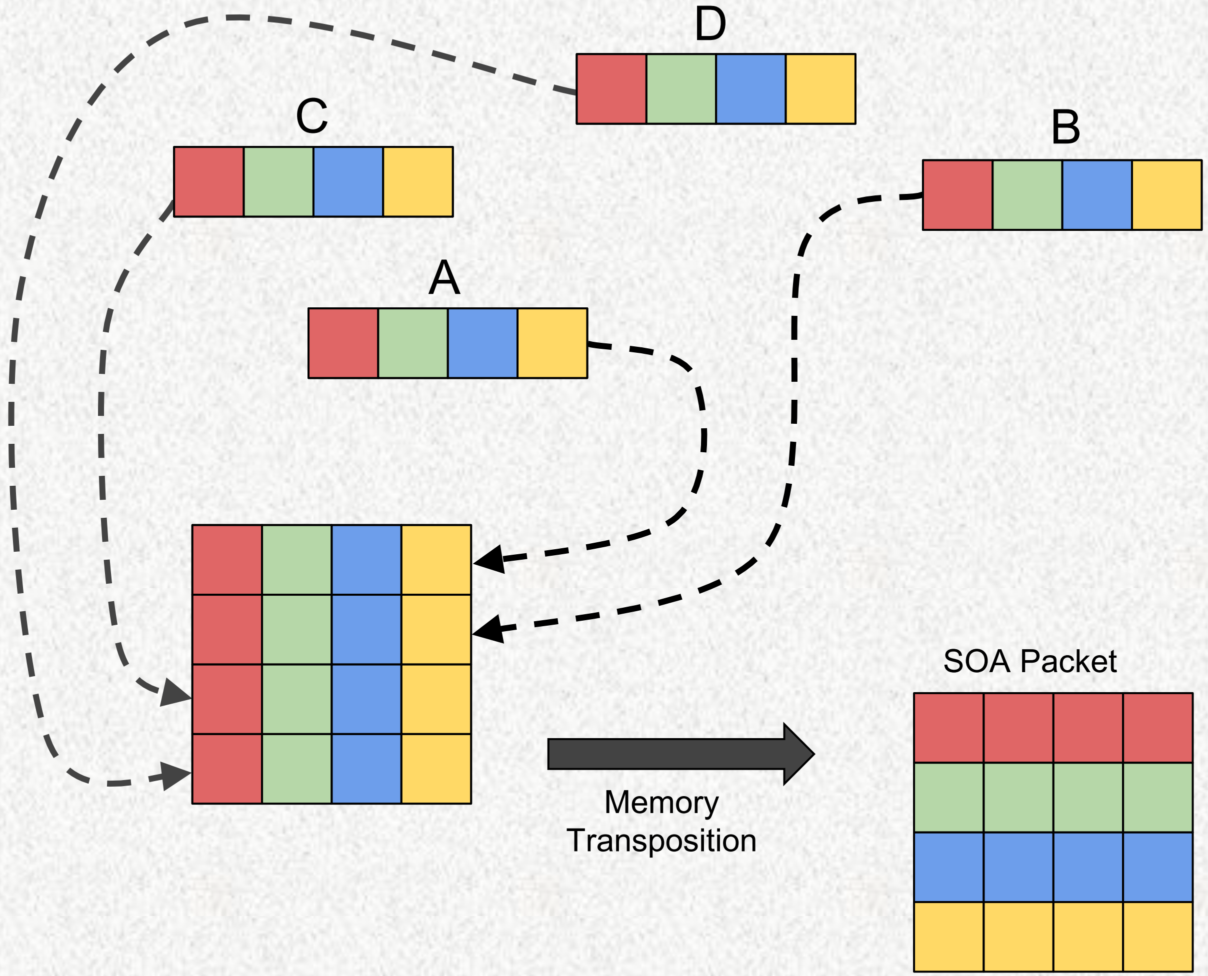
Will the performance gains outweigh the additional work?


c = a + b
c = a + b
c = a + b
c = a + b
AOS Format
c c c c
=
a a a a
+
b b b b
SOA Format

AOS to SOA Conversion


D
C
B
A


D
C
B
A


D
C
B
A
SOA Packet
Memory
Transposition

ISPC

- ISPC = Intel SPMD Program Compiler
- Can target multiple vector architectures, e.g. SSE, AVX, AVX512
- Automatic masking
- C like syntax

Queuing

- Strategy - use “queues” to build up large coherent batches of work
- Queues entries typically consist of a 32-bit index and a 32-bit “sort key”
- Each queue has an associated handler to process entries

When a queue becomes full,
the thread which filled it flushes it

Scalar Path Tracing

Scalar Path Tracing

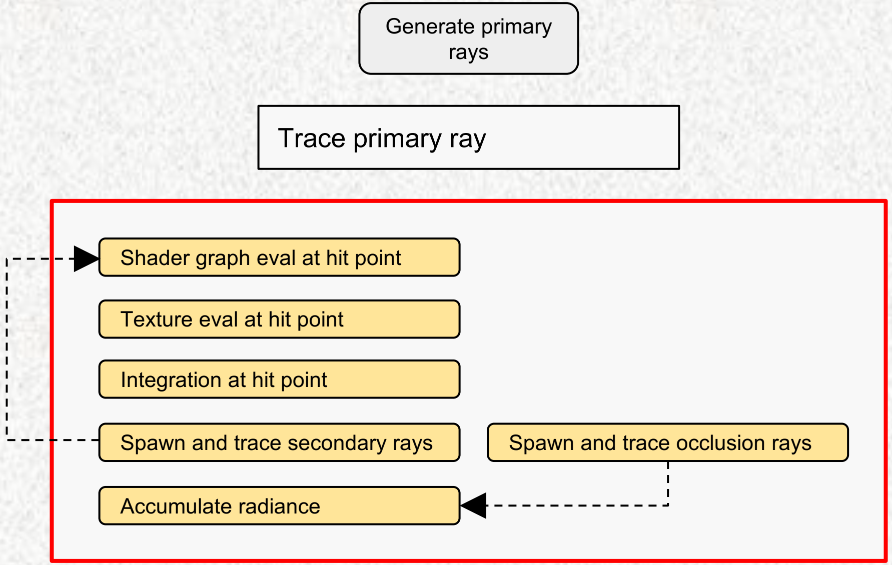
Generate primary rays

Scalar Path Tracing

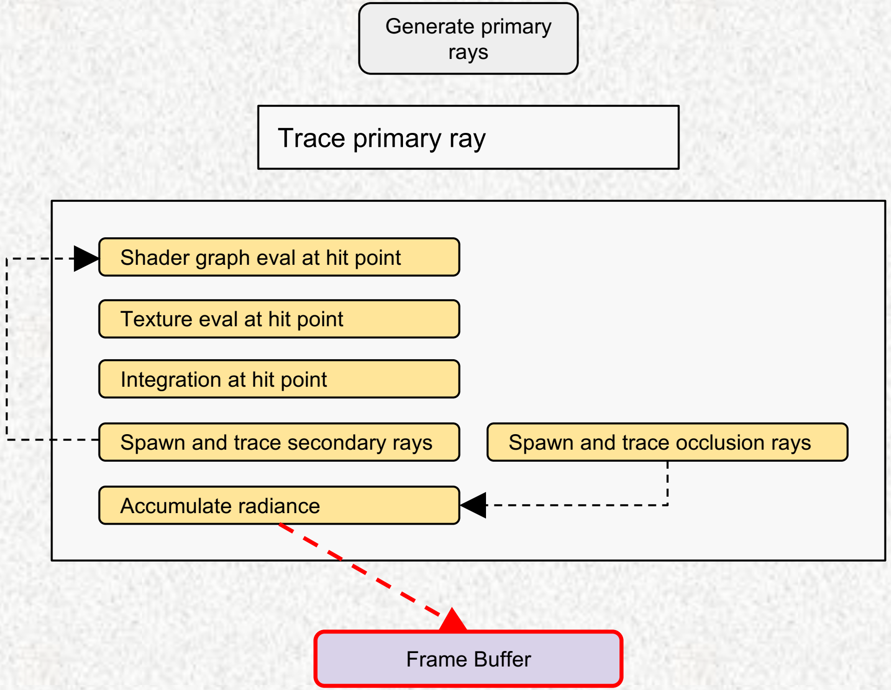
Generate primary rays

Trace primary ray

Scalar Path Tracing



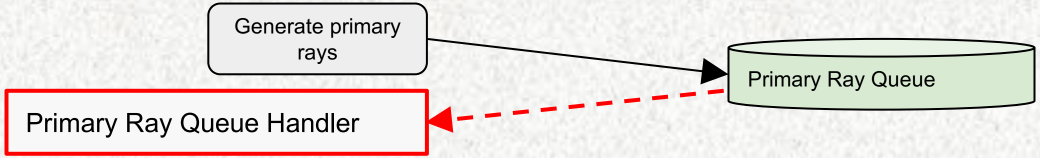
Scalar Path Tracing



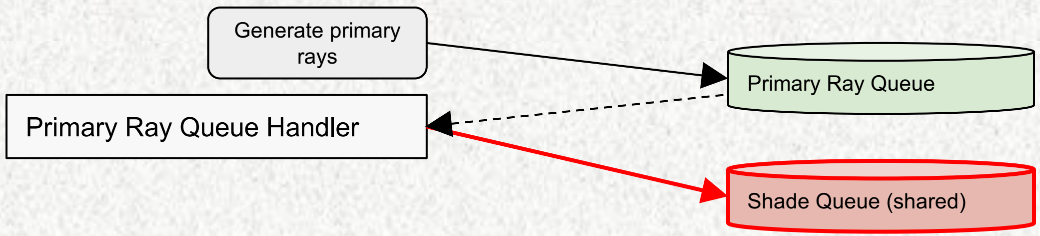
Vectorized Path Tracing

Vectorized Path Tracing

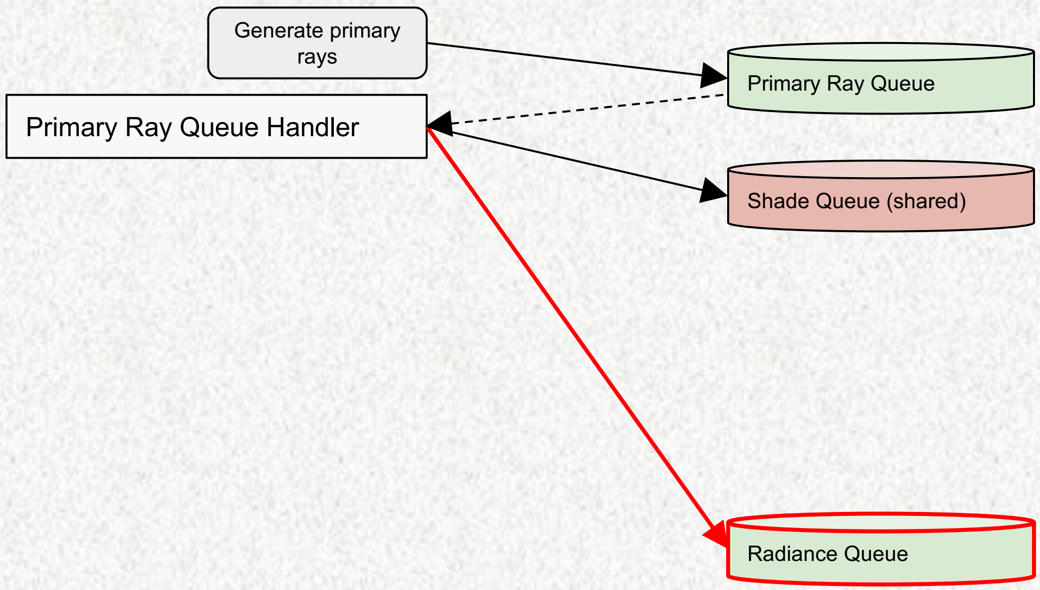
Vectorized Path Tracing



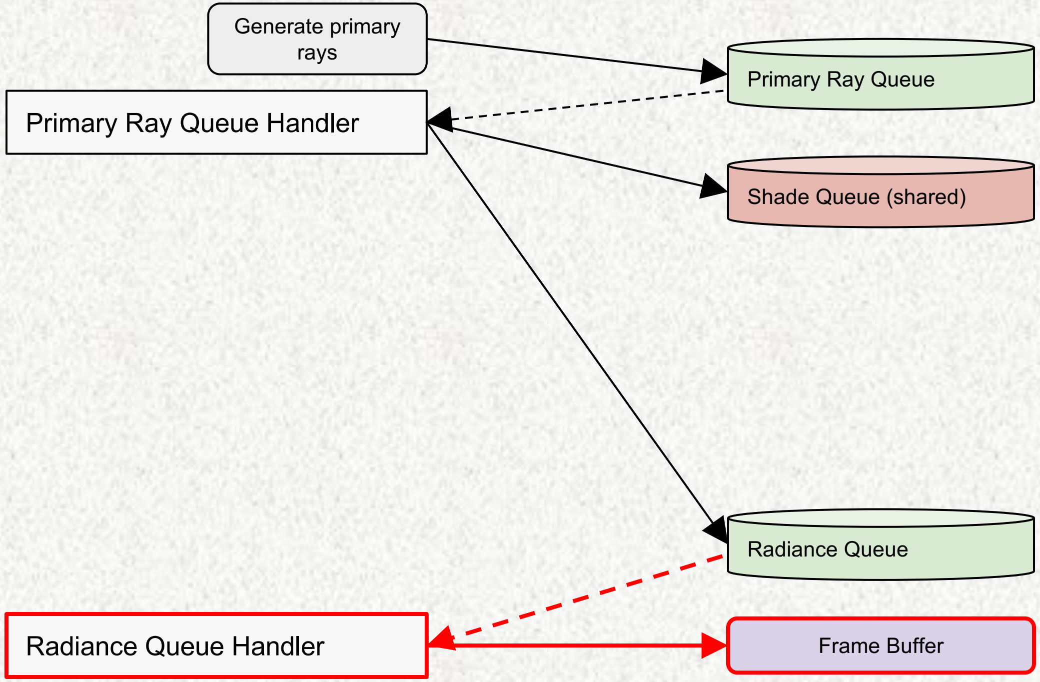
Vectorized Path Tracing



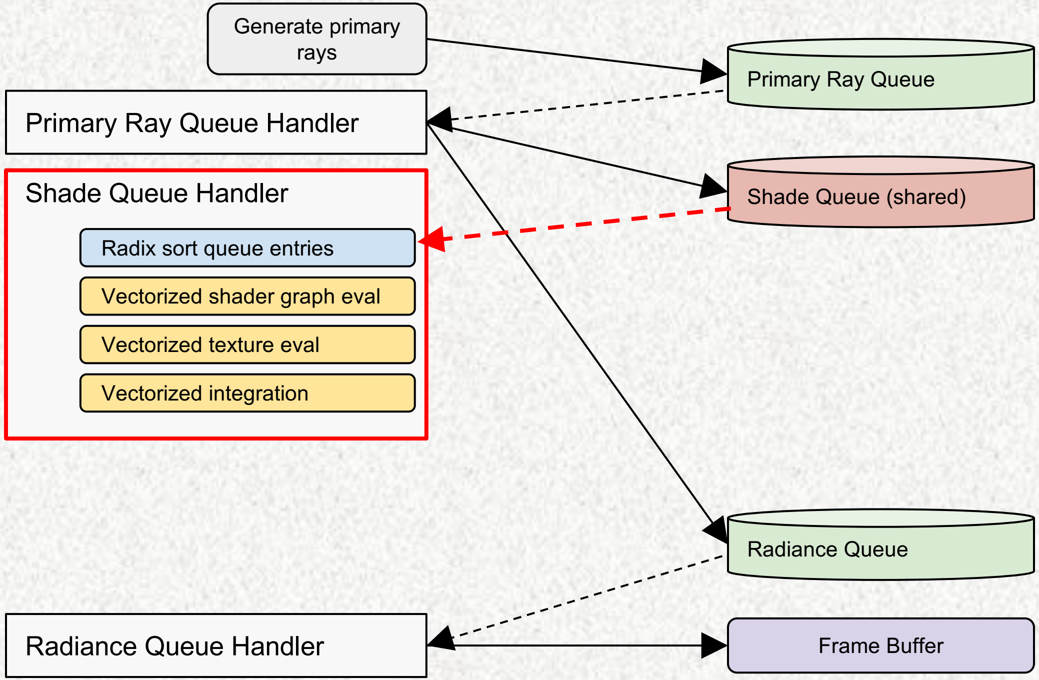
Vectorized Path Tracing



Vectorized Path Tracing



Vectorized Path Tracing



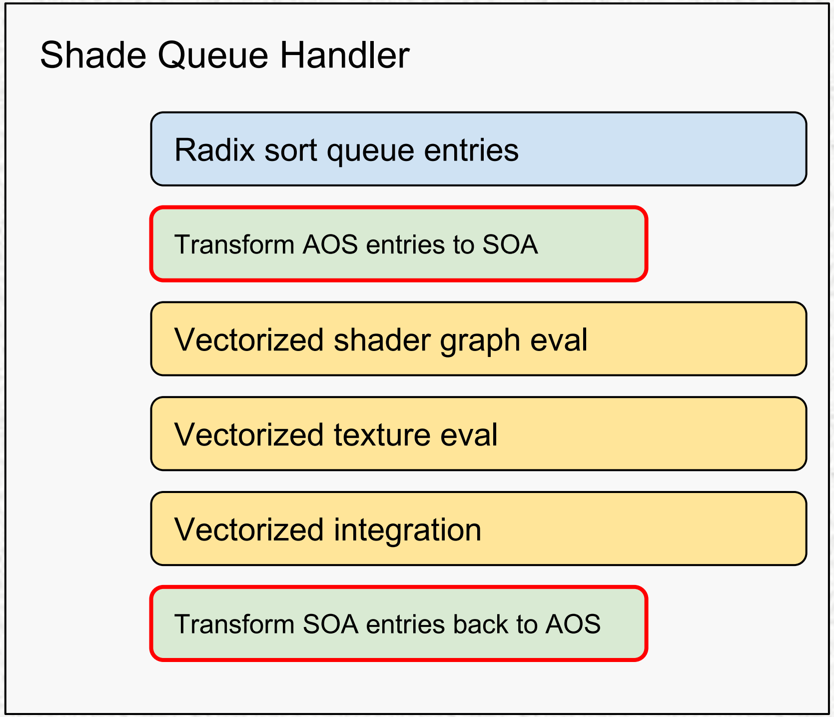
Shade Queue Handler

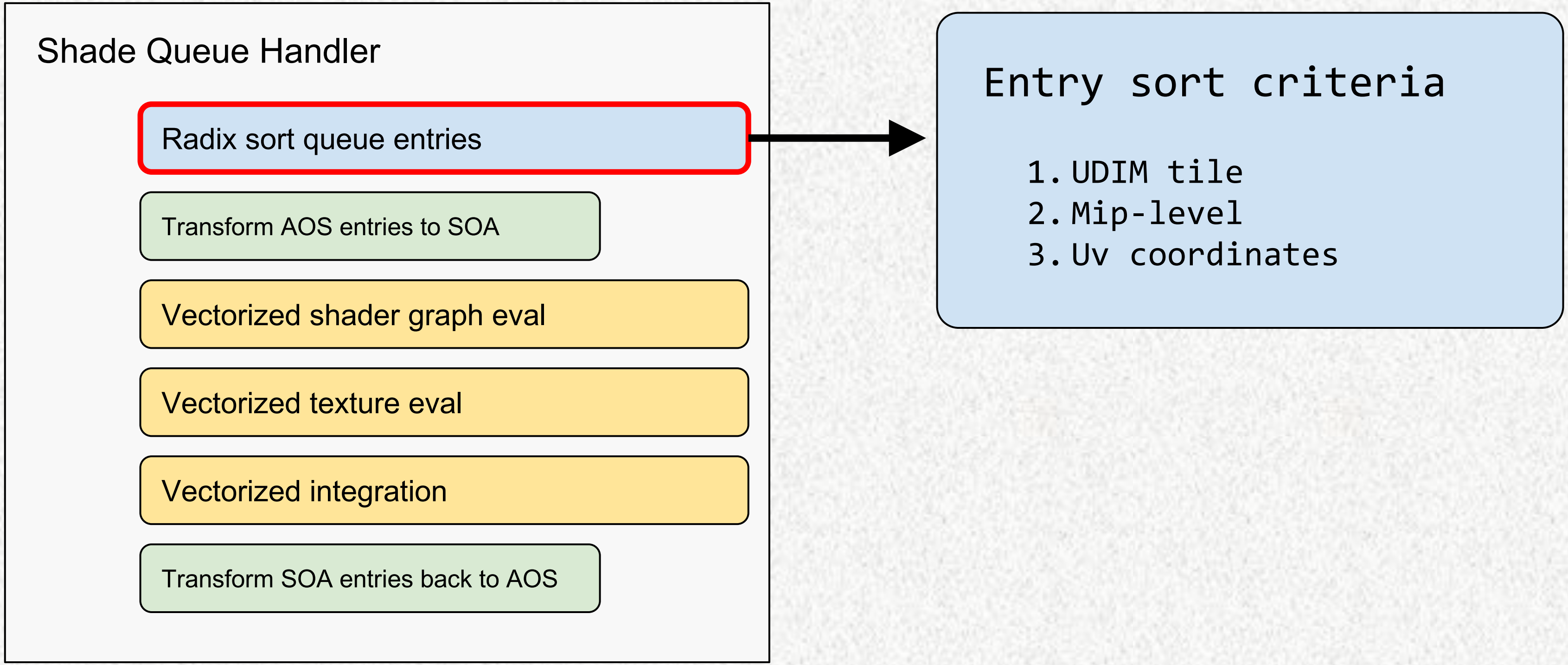
Radix sort queue entries

Vectorized shader graph eval

Vectorized texture eval

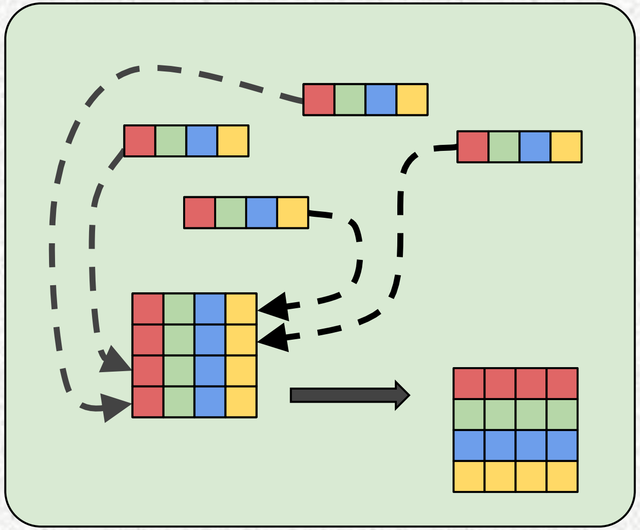
Vectorized integration

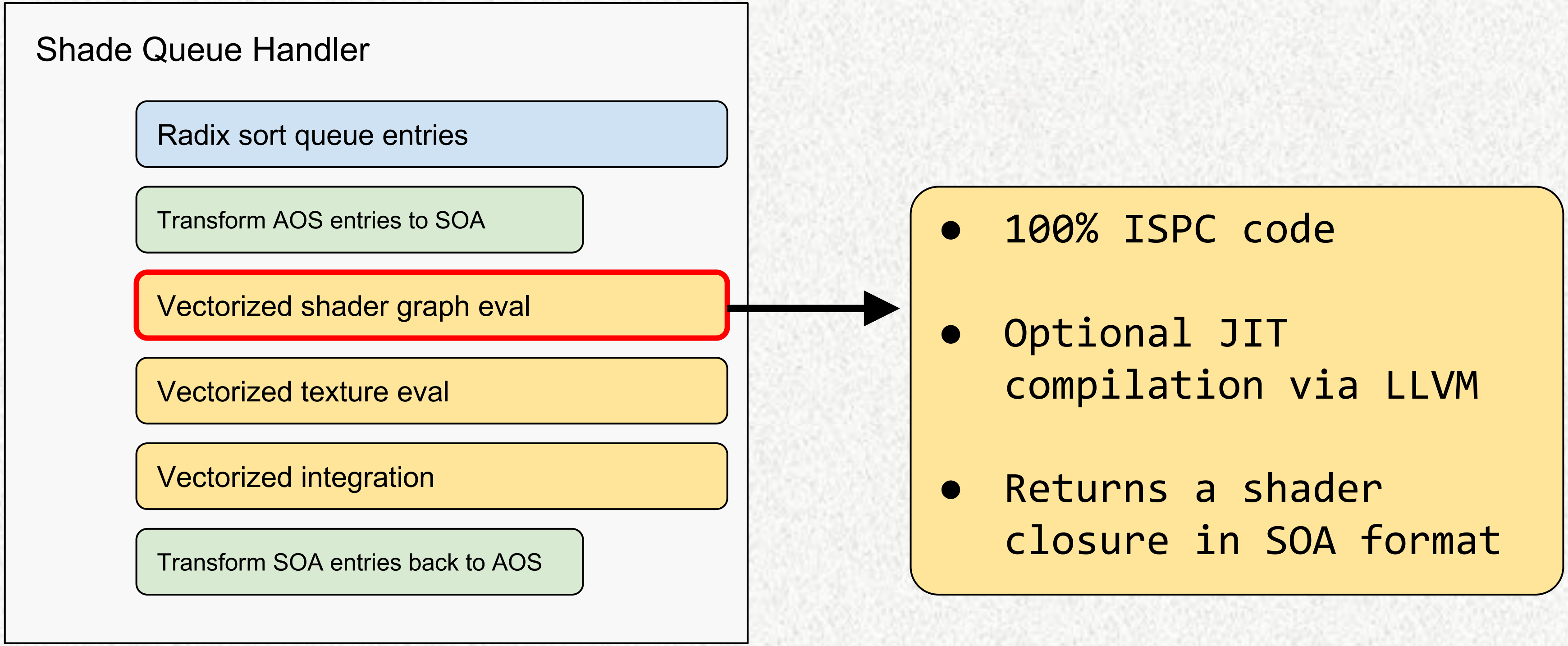

Shade Queue Handler
Radix sort queue entries
Transform AOS entries to SOA
Vectorized shader graph eval
Vectorized texture eval
Vectorized integration
Transform SOA entries back to AOS

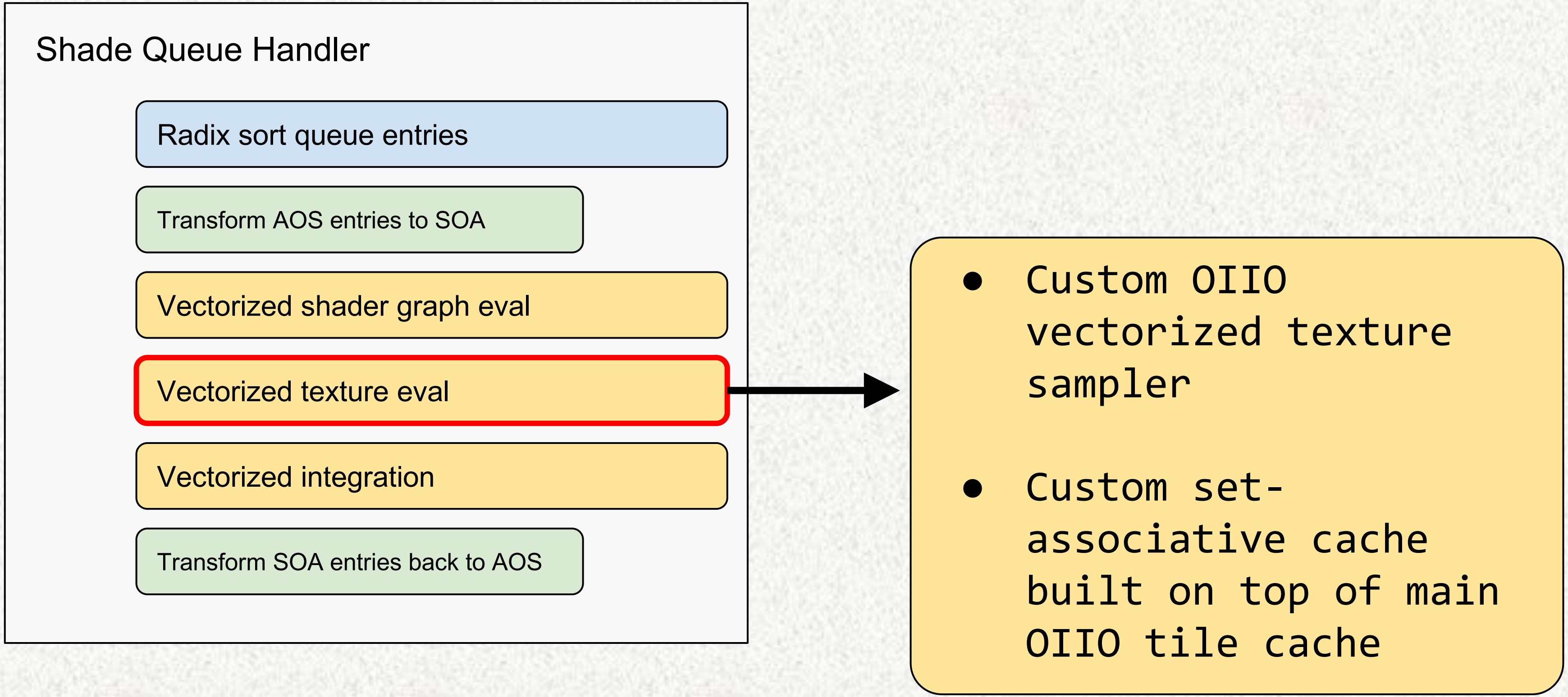

Shade Queue Handler
Radix sort queue entries
Transform AOS entries to SOA
Vectorized shader graph eval
Vectorized texture eval
Vectorized integration
Transform SOA entries back to AOS
Entry sort criteria
1. UDIM tile
2. Mip-level
3. Uv coordinates

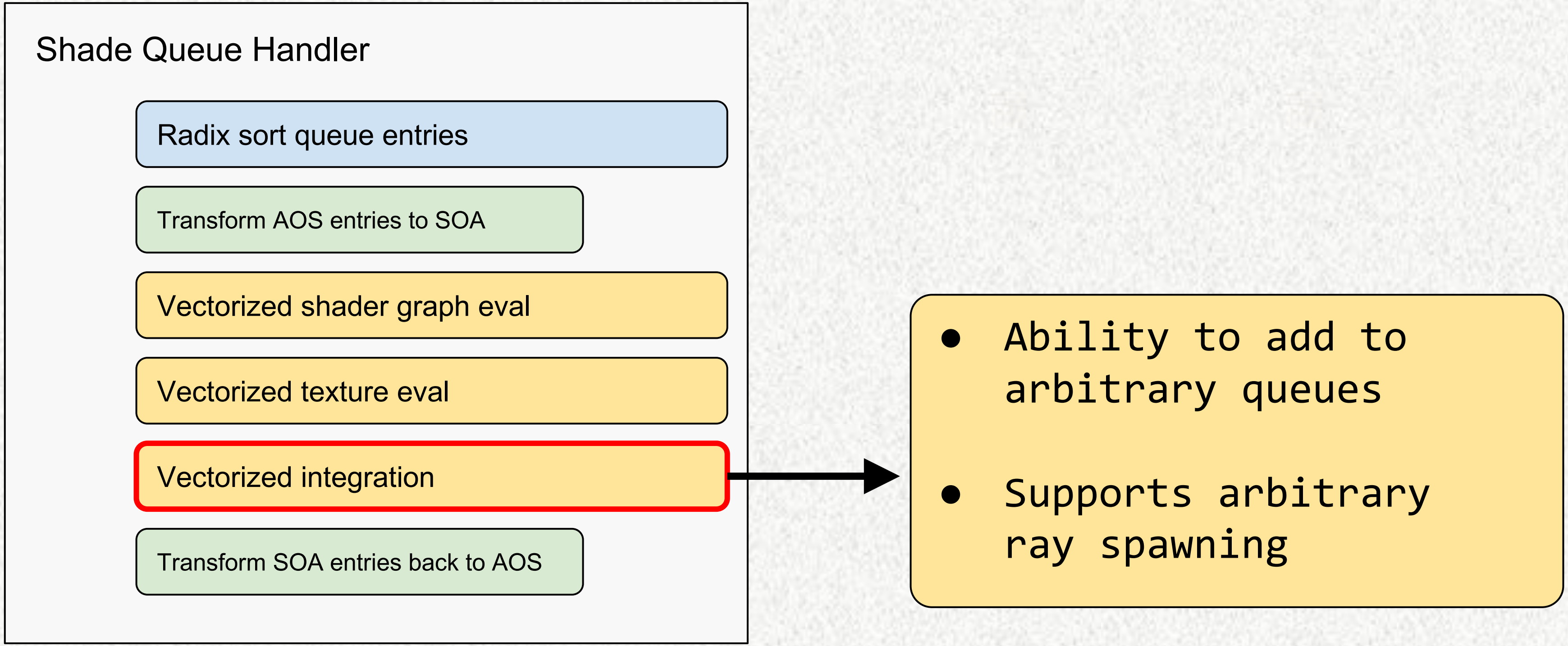
Shade Queue Handler

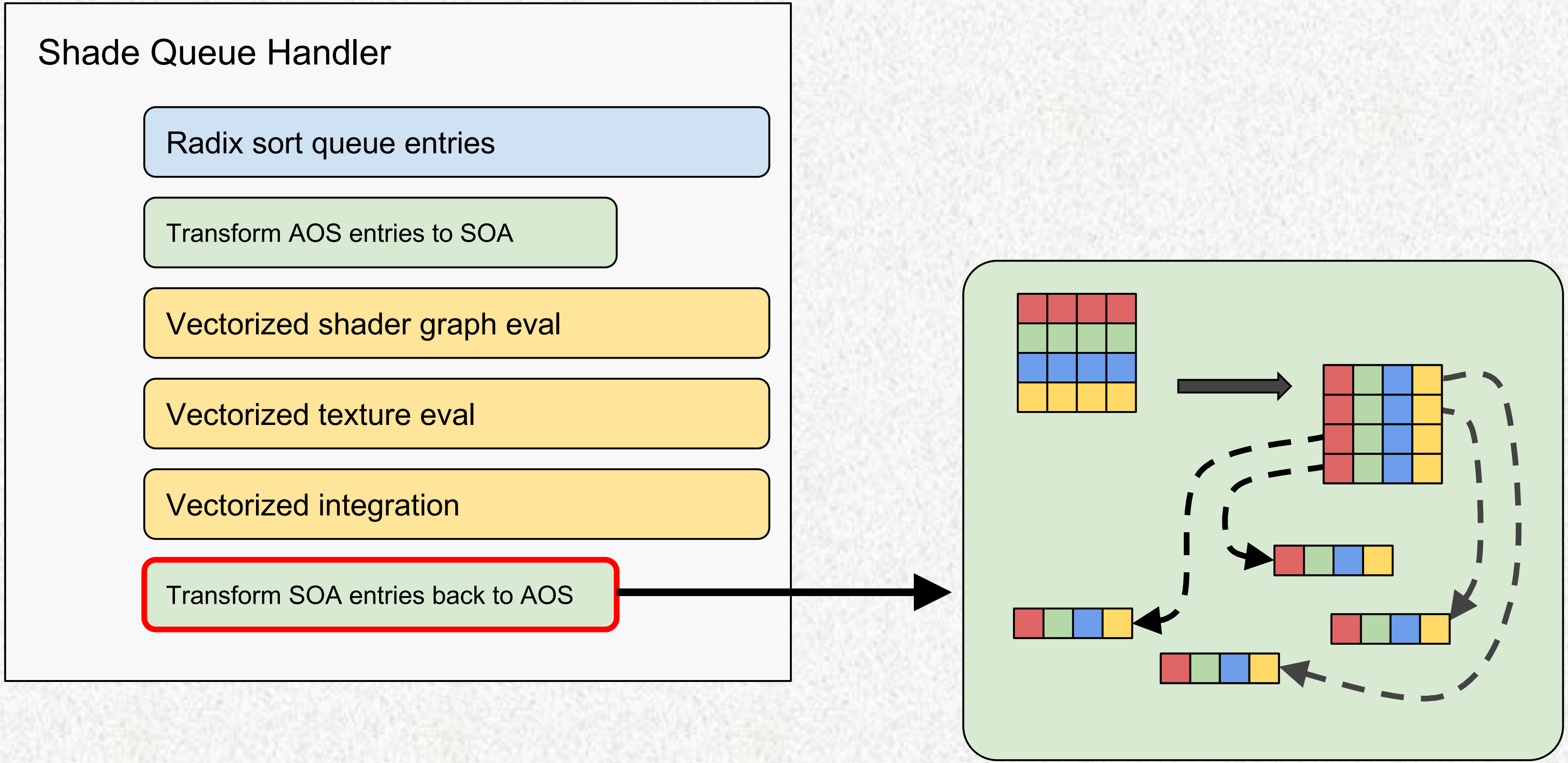
- Radix sort queue entries
- Transform AOS entries to SOA
- Vectorized shader graph eval
- Vectorized texture eval
- Vectorized integration
- Transform SOA entries back to AOS



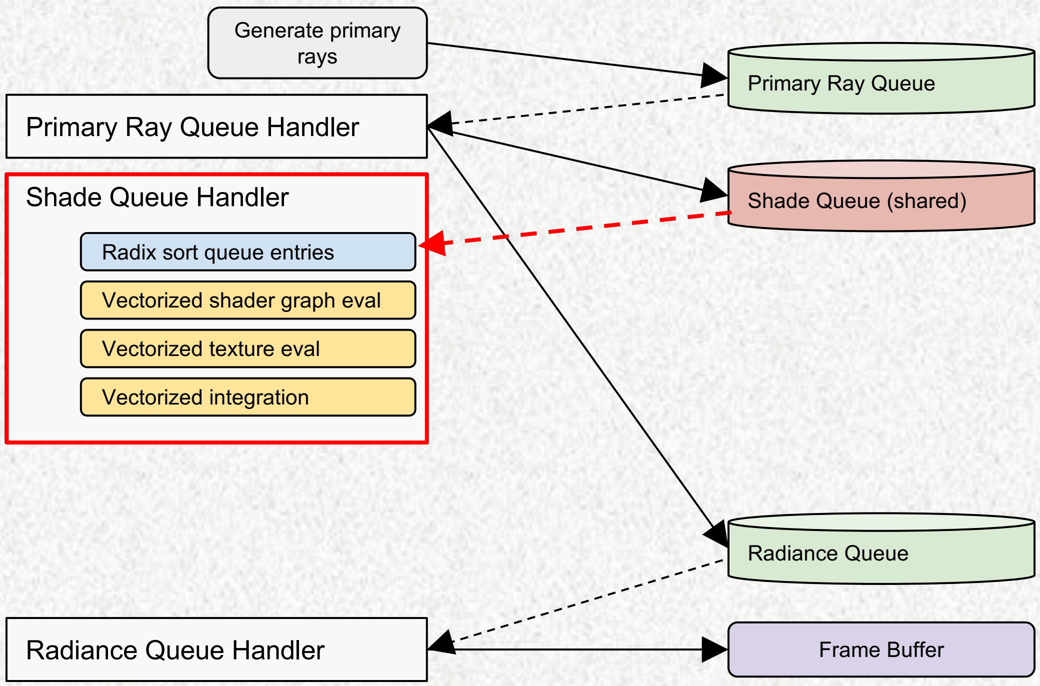

Shade Queue Handler
Radix sort queue entries
Transform AOS entries to SOA
Vectorized shader graph eval
Vectorized texture eval
Vectorized integration
Transform SOA entries back to AOS
100% ISPC code
Optional JIT compilation via LLVM
Returns a shader closure in SOA format


Shade Queue Handler
Radix sort queue entries
Transform AOS entries to SOA
Vectorized shader graph eval
Vectorized texture eval
Vectorized integration
Transform SOA entries back to AOS
Custom OIIO vectorized texture sampler
Custom set-associative cache built on top of main OIIO tile cache

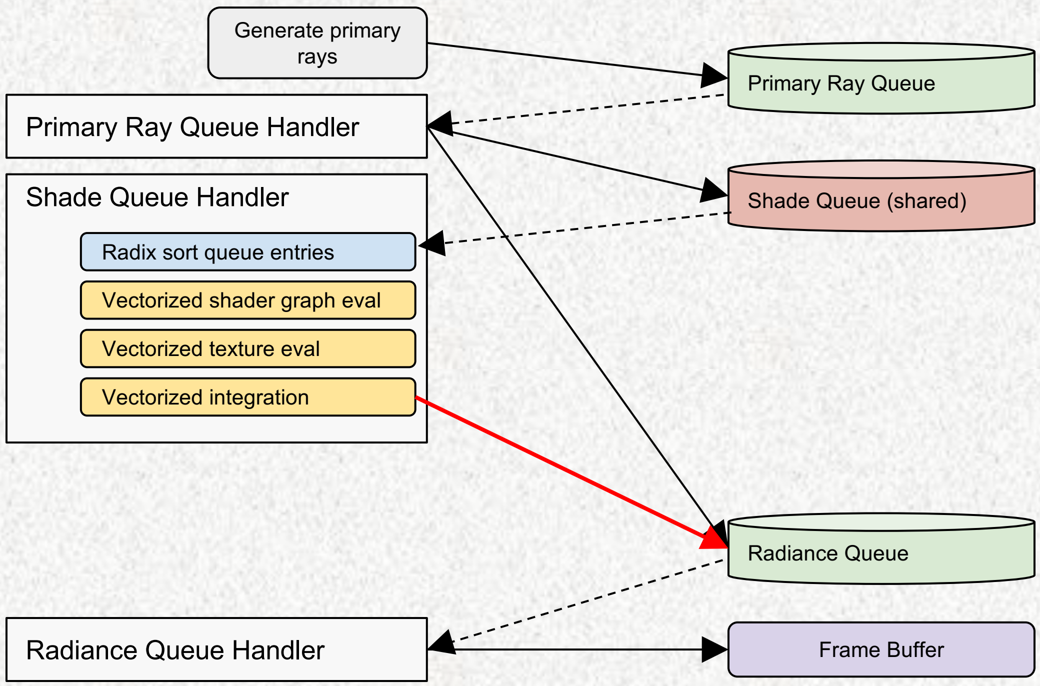

Shade Queue Handler
Radix sort queue entries
Transform AOS entries to SOA
Vectorized shader graph eval
Vectorized texture eval
Vectorized integration
Transform SOA entries back to AOS
• Ability to add to arbitrary queues
• Supports arbitrary ray spawning


Shade Queue Handler
Radix sort queue entries
Transform AOS entries to SOA
Vectorized shader graph eval
Vectorized texture eval
Vectorized integration
Transform SOA entries back to AOS

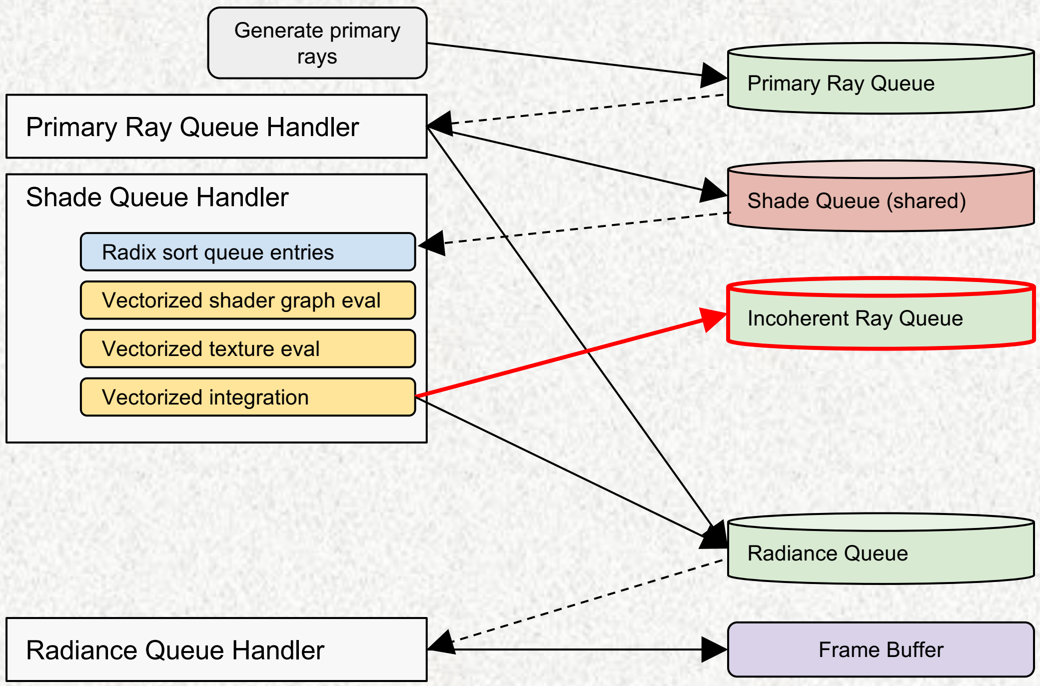
Vectorized Path Tracing



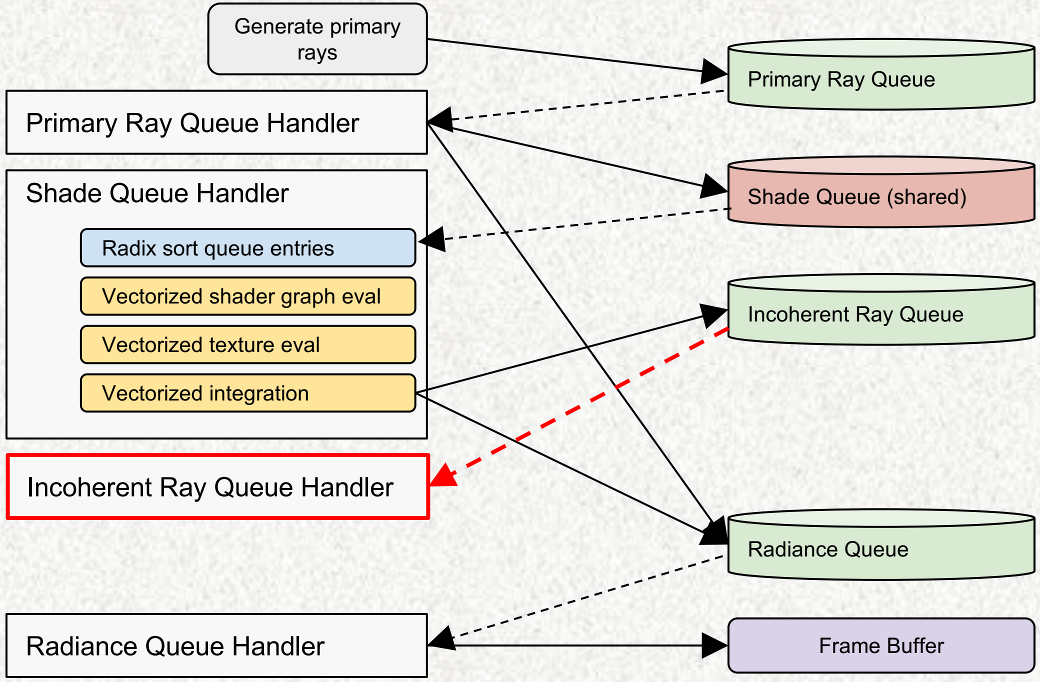
Vectorized Path Tracing



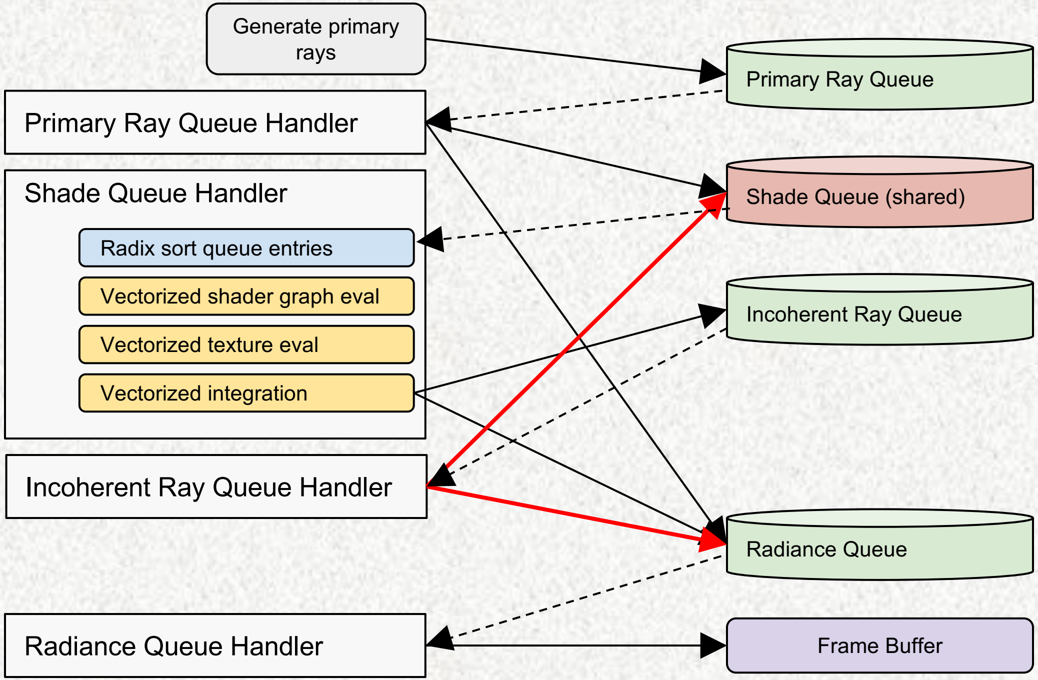
Vectorized Path Tracing



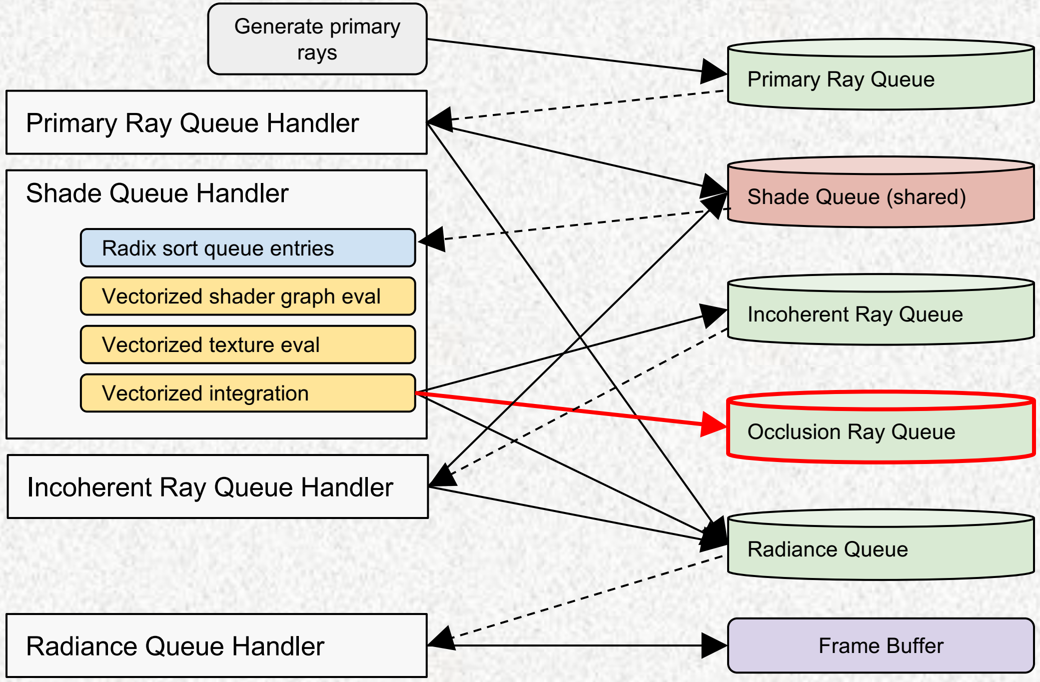
Vectorized Path Tracing



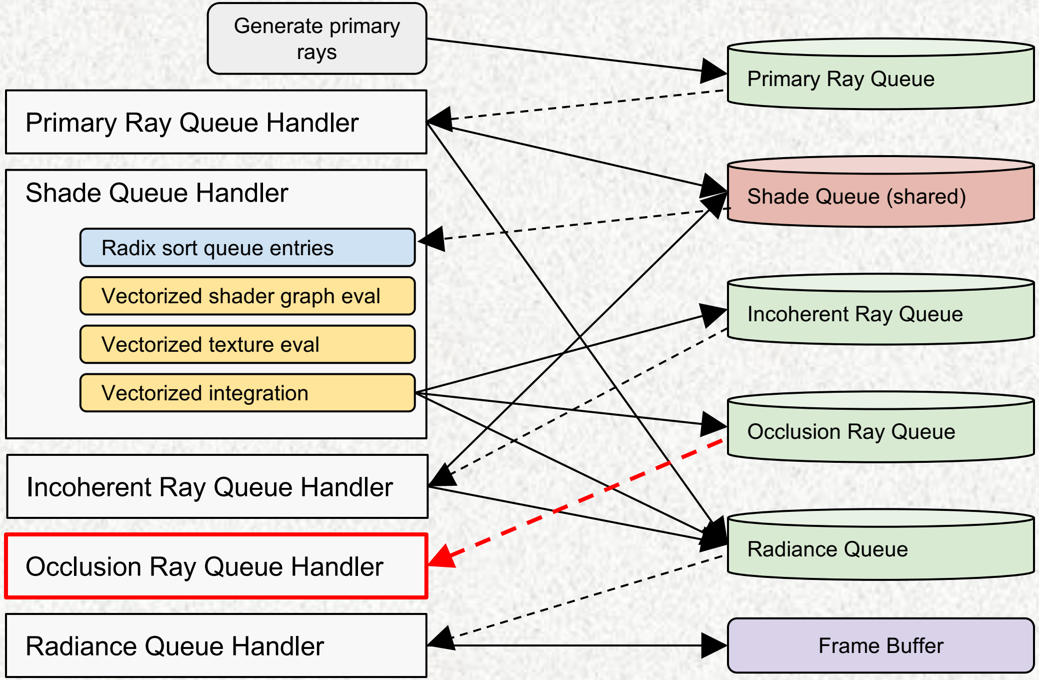
Vectorized Path Tracing



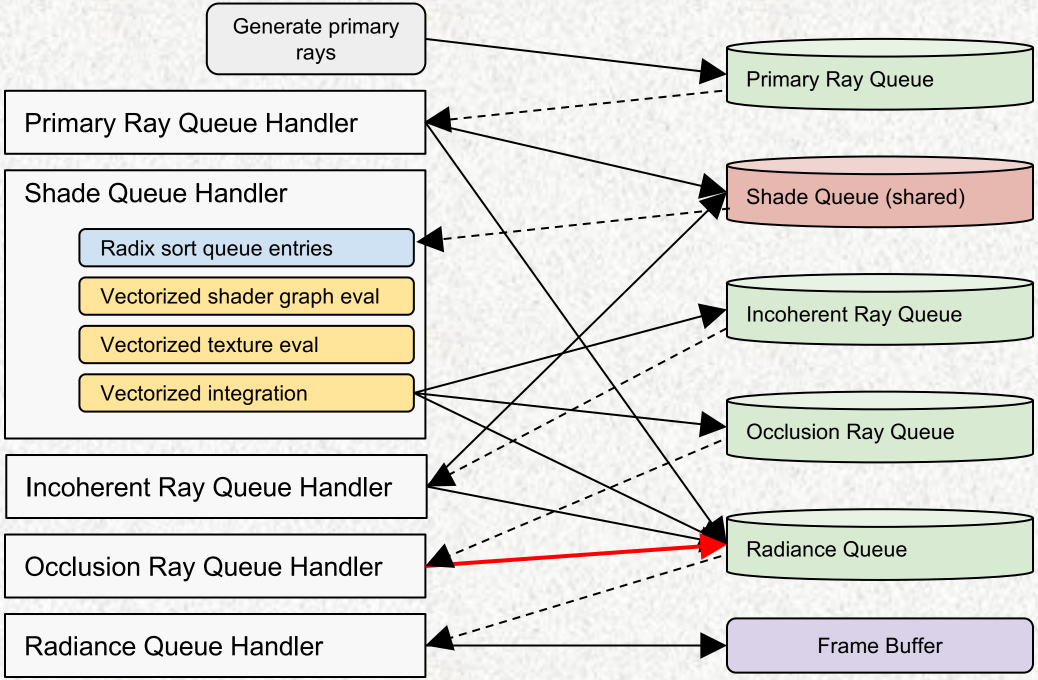
Vectorized Path Tracing



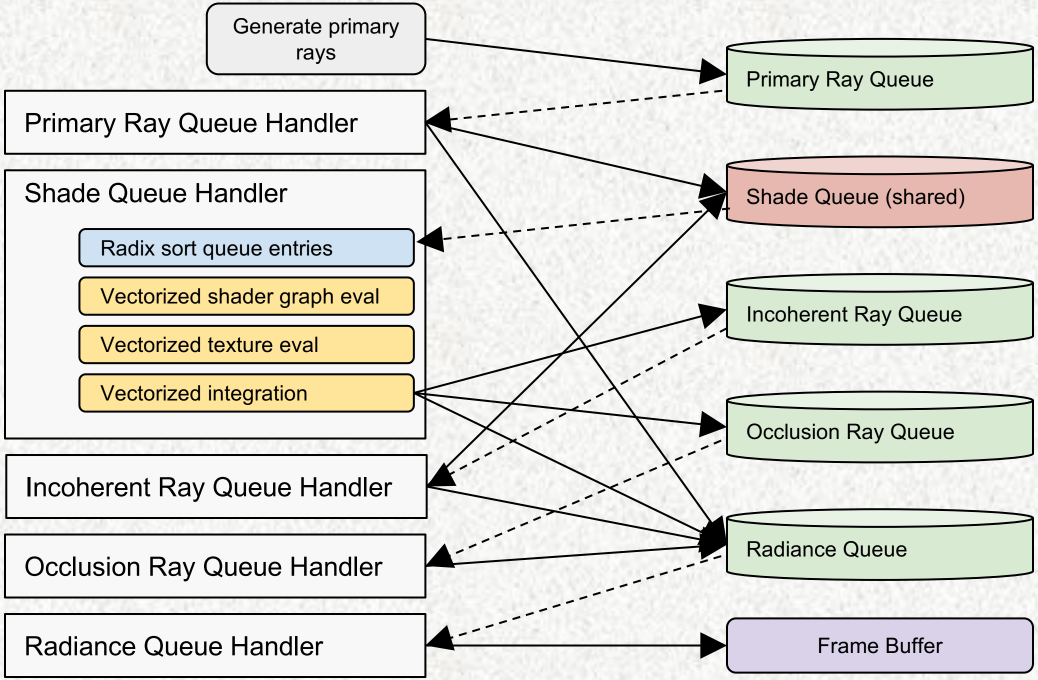
Vectorized Path Tracing



Vectorized Path Tracing

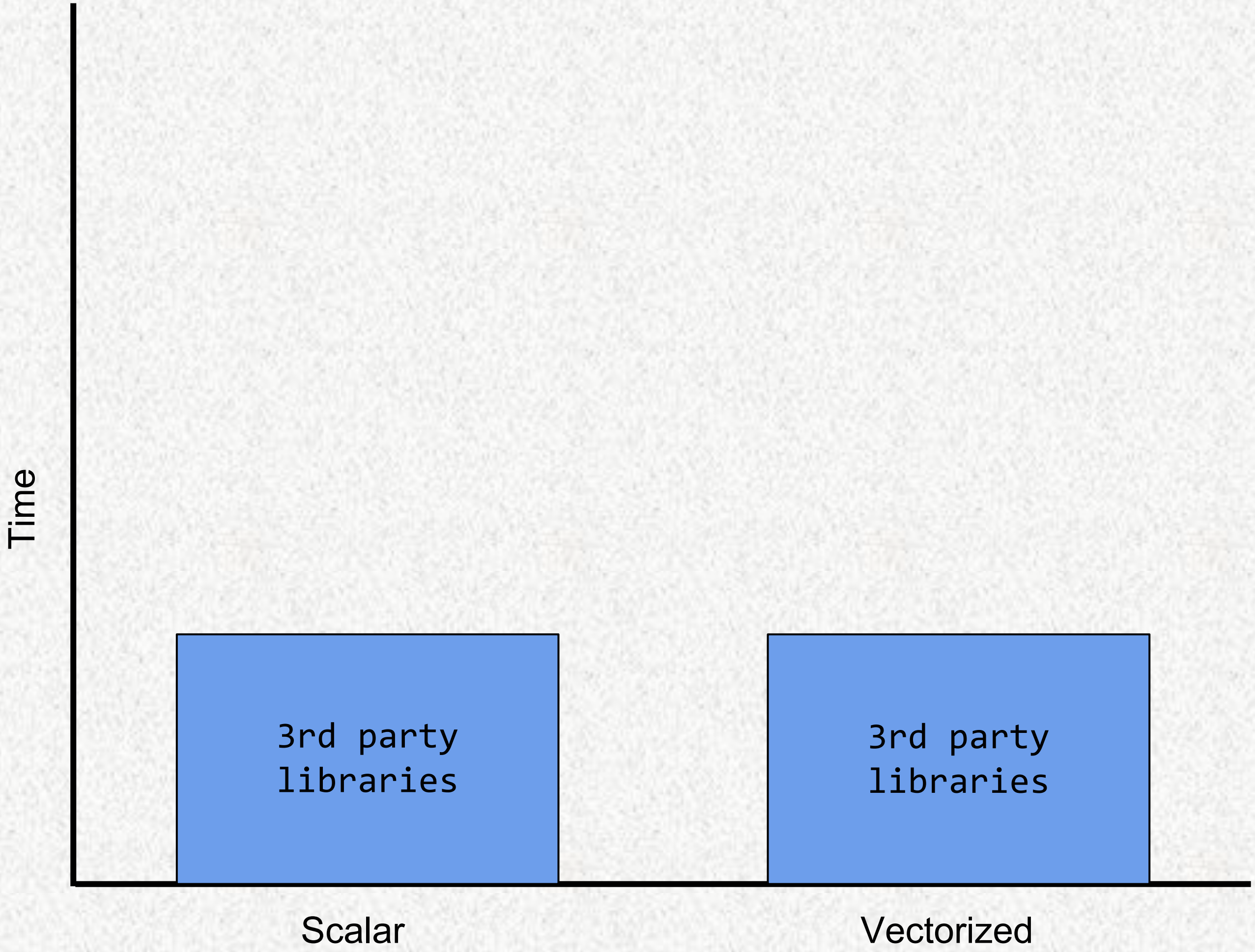


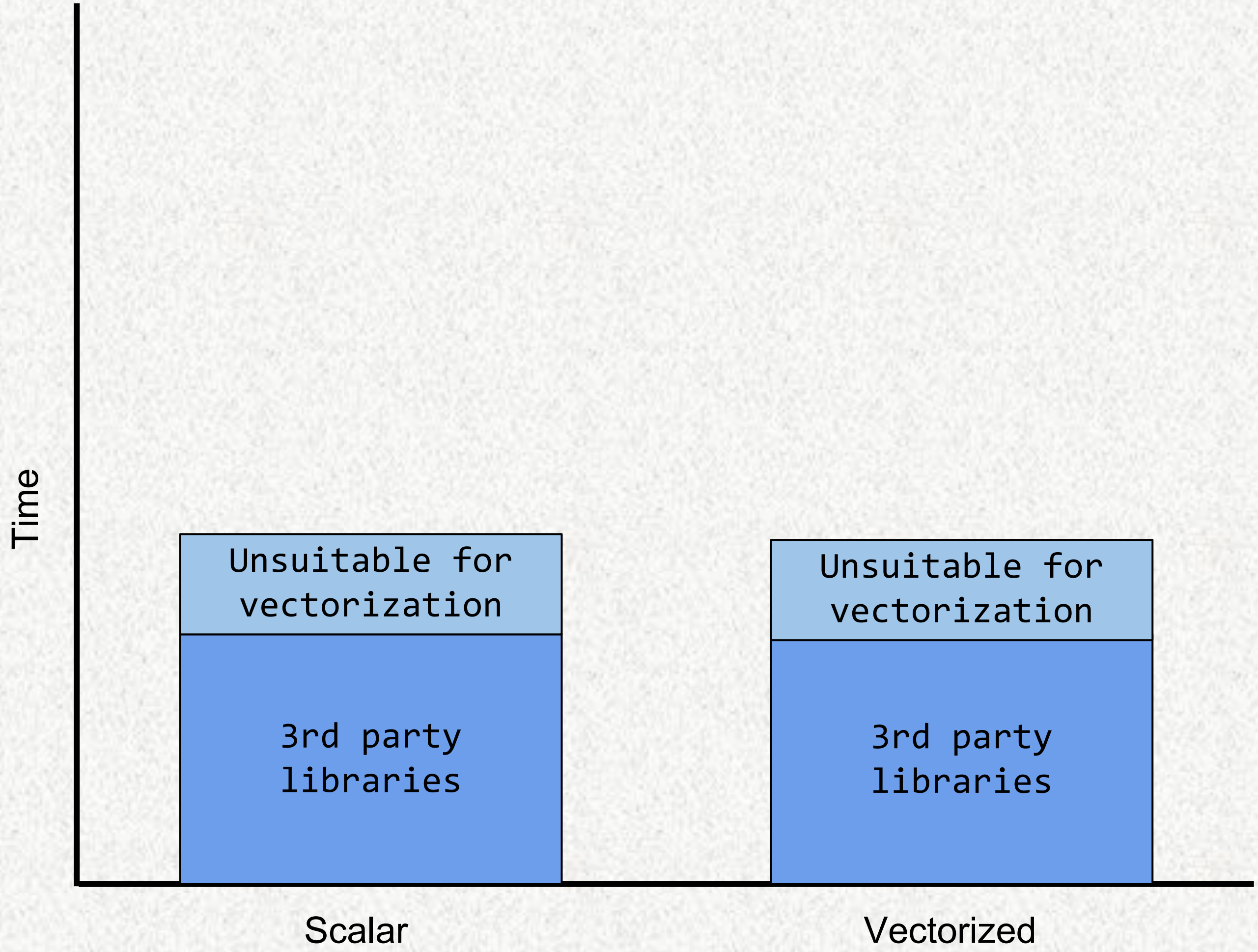
Vectorized Path Tracing

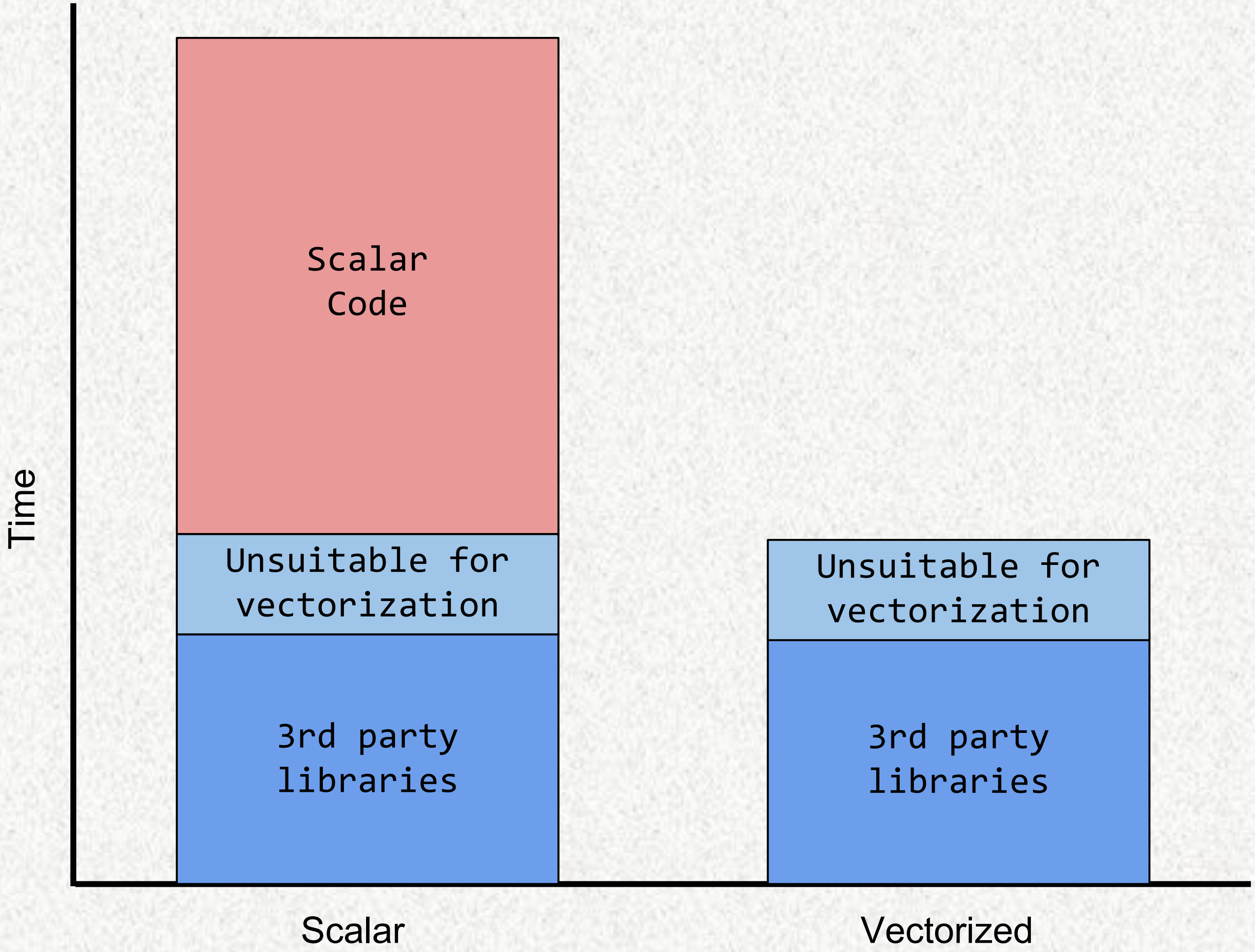


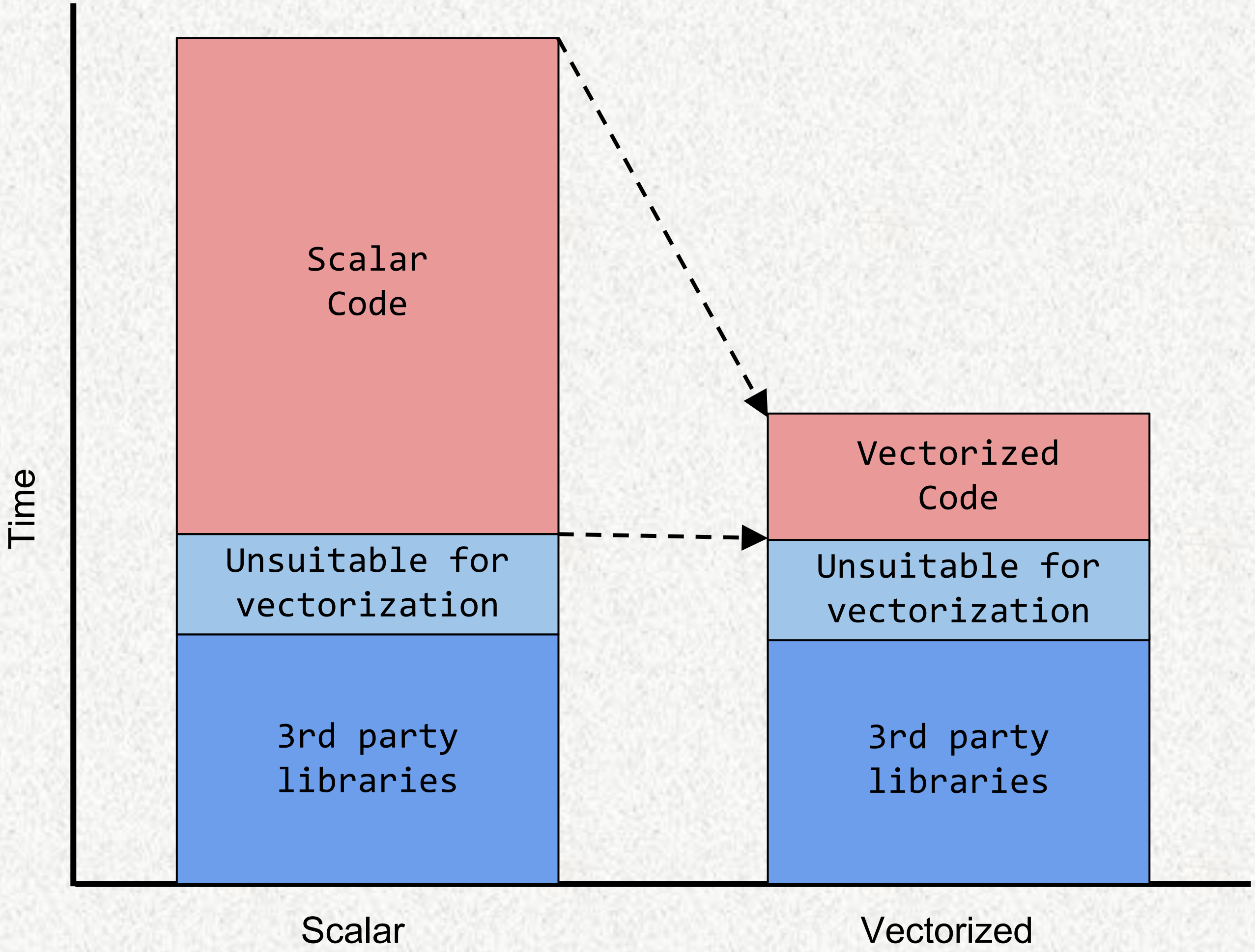
Will the performance gains outweigh the additional work?

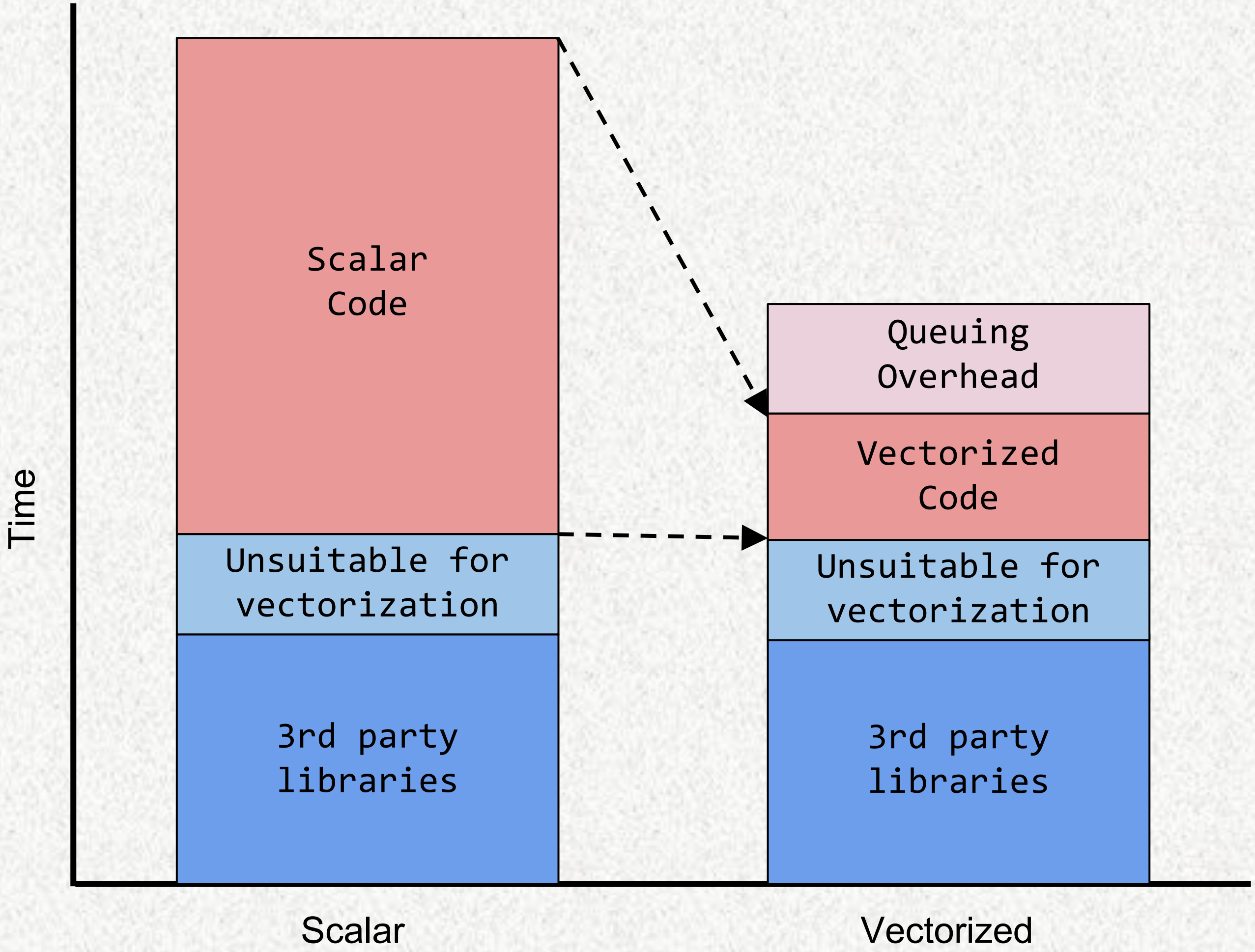

Time
Scalar
Vectorized

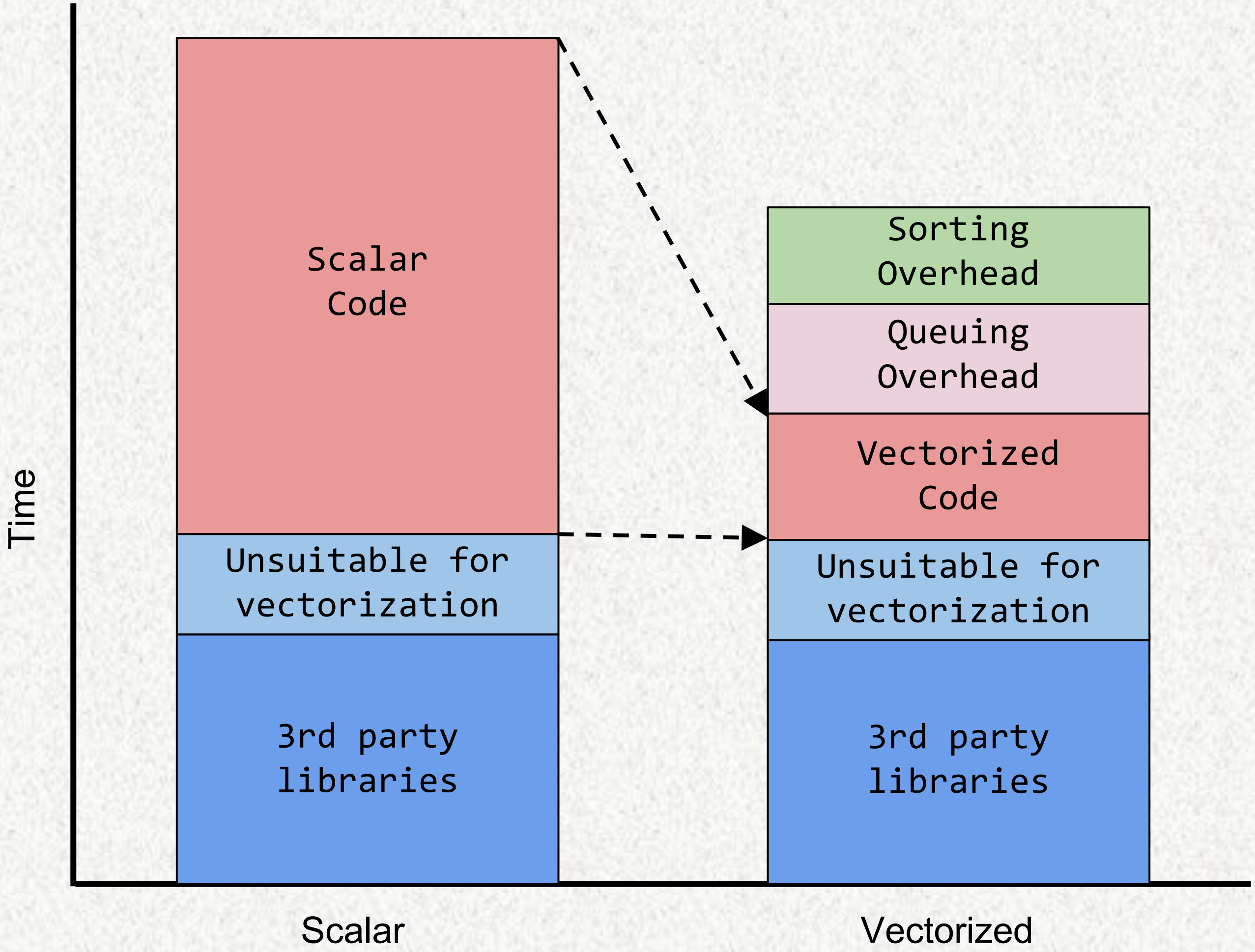

Time
3rd party libraries
3rd party libraries
Scalar
Vectorized

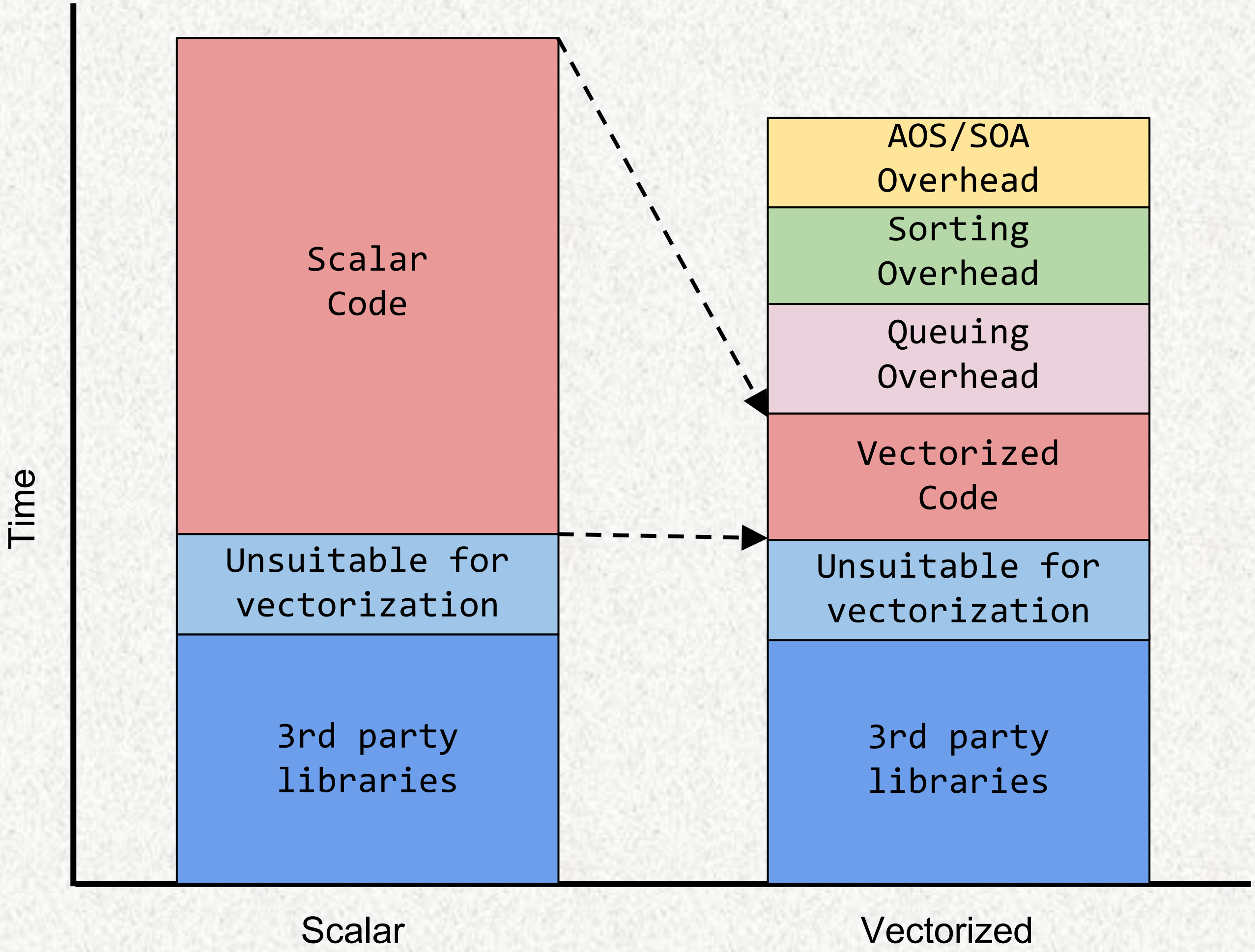

Time
Unsuitable for vectorization
3rd party libraries
Unsuitable for vectorization
3rd party libraries
Scalar
Vectorized


Time
Scalar
Code
Unsuitable for
vectorization
3rd party
libraries
Unsuitable for
vectorization
3rd party
libraries
Scalar
Vectorized


Time
Scalar
Code
Unsuitable for
vectorization
3rd party
libraries
Vectorized
Code
Unsuitable for
vectorization
3rd party
libraries
Scalar
Vectorized

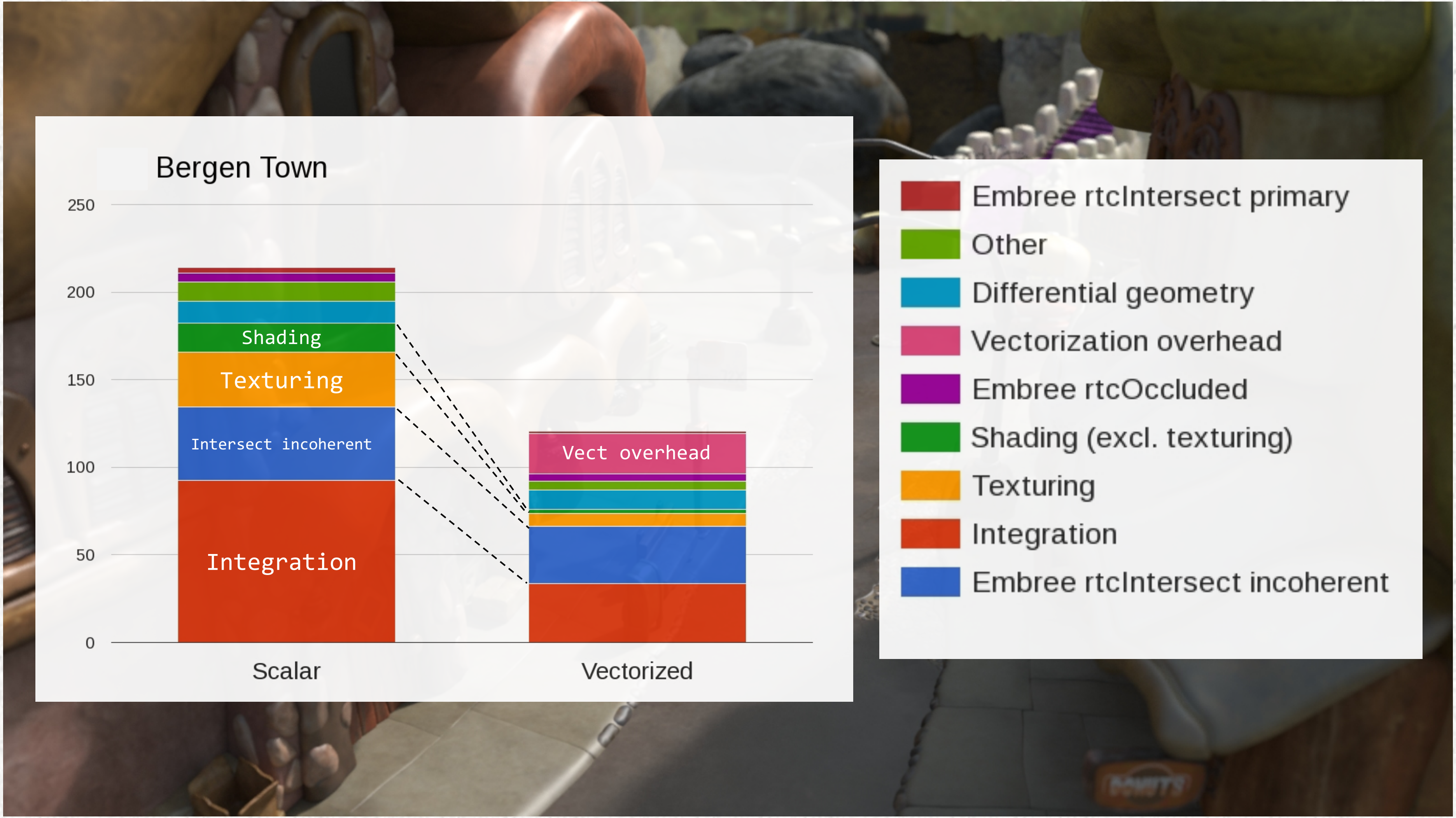

Time
Scalar Code
Unsuitable for vectorization
3rd party libraries
Queuing Overhead
Vectorized Code
Unsuitable for vectorization
3rd party libraries
Scalar
Vectorized


Time
Scalar Code
Unsuitable for vectorization
3rd party libraries
Sorting Overhead
Queuing Overhead
Vectorized Code
Unsuitable for vectorization
3rd party libraries
Scalar
Vectorized


Time
Scalar Code
Unsuitable for vectorization
3rd party libraries
AOS/SOA Overhead
Sorting Overhead
Queuing Overhead
Vectorized Code
Unsuitable for vectorization
3rd party libraries
Scalar
Vectorized

Results

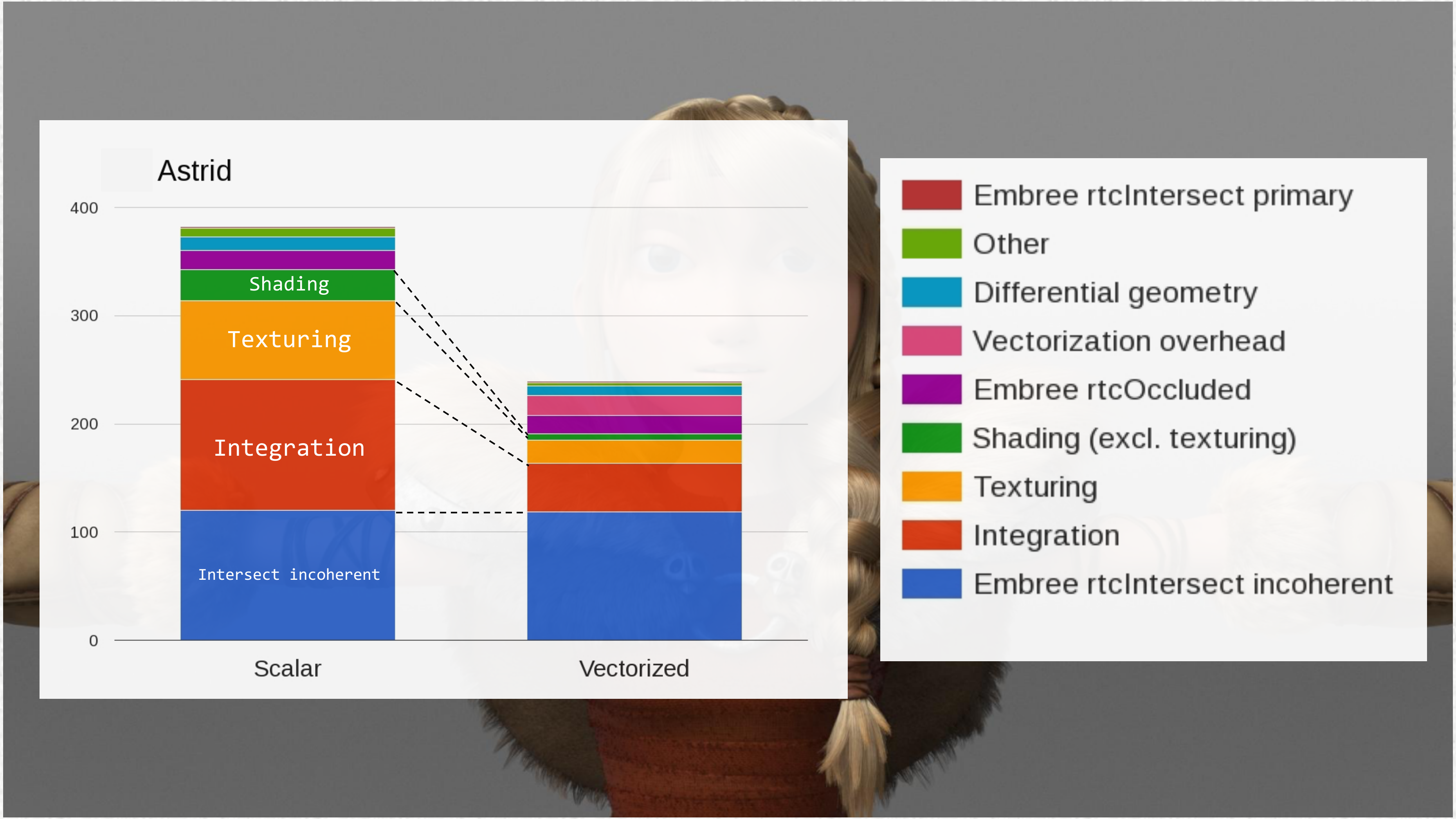

SHOES
72X



Bergen Town
250
200
150
100
50
0
Shading
Texturing
Intersect incoherent
Integration
Vect overhead
Scalar
Vectorized
Embree rtcIntersect primary
Other
Differential geometry
Vectorization overhead
Embree rtcOccluded
Shading (excl. texturing)
Texturing
Integration
Embree rtcIntersect incoherent

Bergen Town Vectorization Speedups

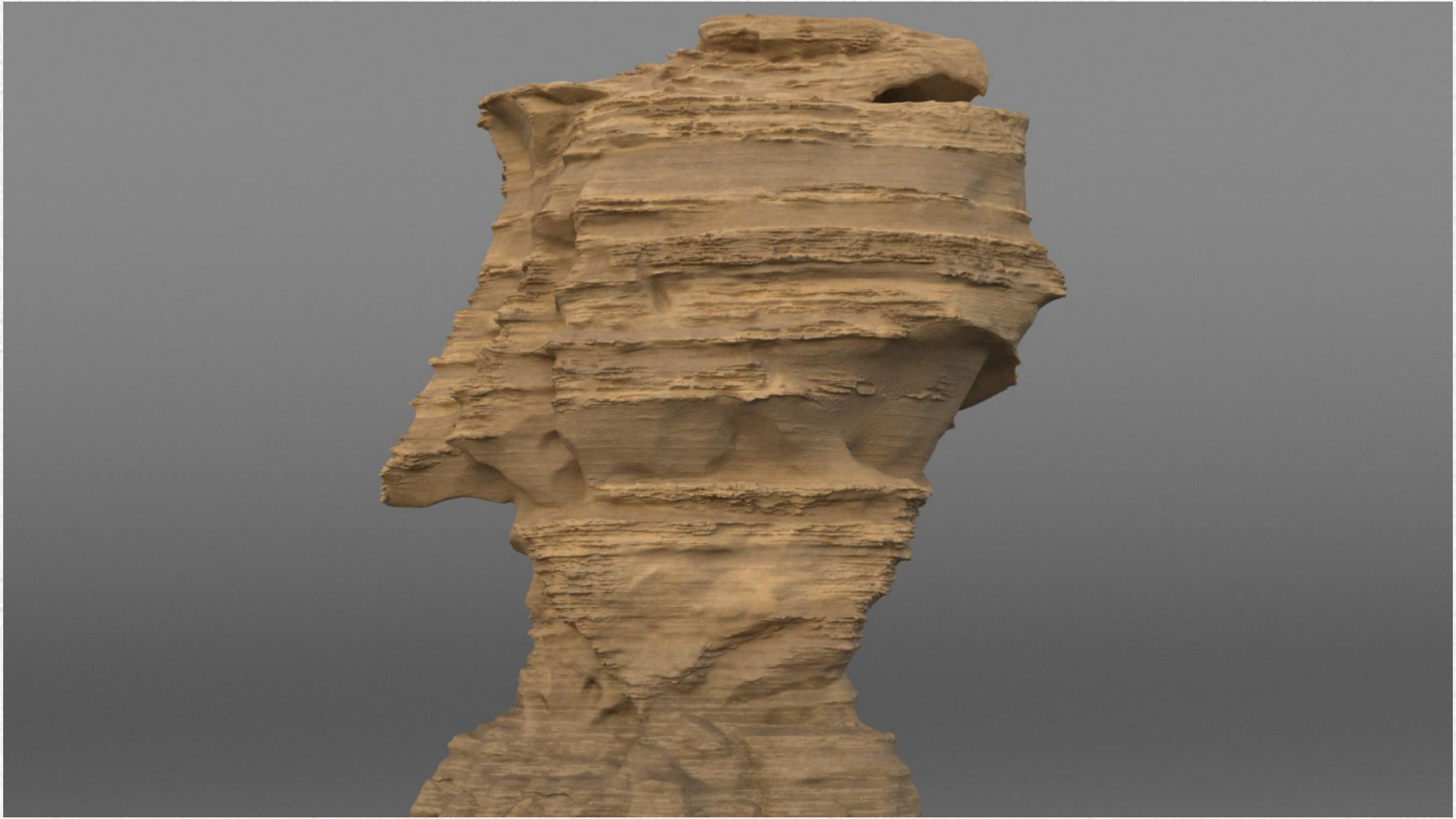
Ray intersection subsystem speedup	1.20x
Shading subsystem speedup	**6.19x**
Texturing subsystem speedup	4.24x
Integration subsystem speedup	2.75x
Overall speedup	1.77x

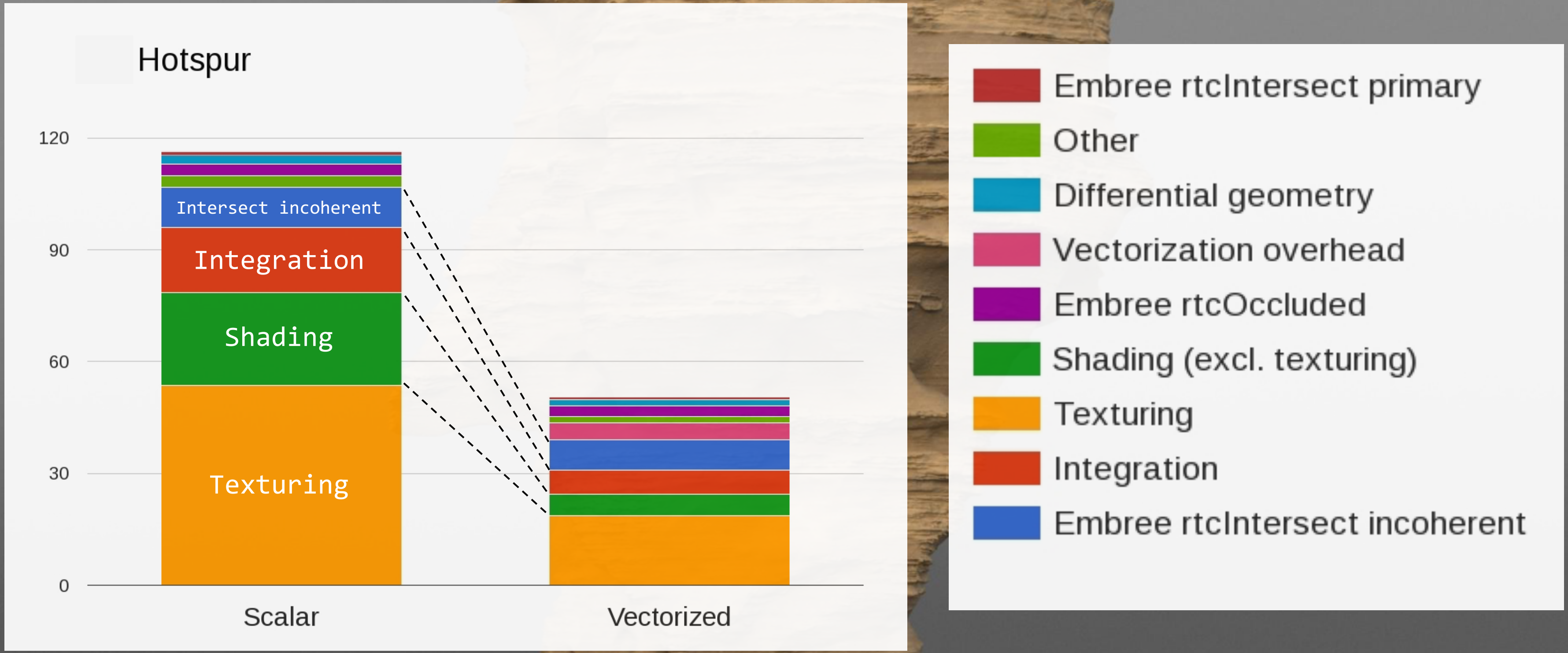


Astrid
400
300
200
100
0
Shading
Texturing
Integration
Intersect incoherent
Scalar
Vectorized
Embree rtcIntersect primary
Other
Differential geometry
Vectorization overhead
Embree rtcOccluded
Shading (excl. texturing)
Texturing
Integration
Embree rtcIntersect incoherent

Astrid Vectorization Speedups

Ray intersection subsystem speedup	1.00x
Shading subsystem speedup	**4.54x**
Texturing subsystem speedup	3.43x
Integration subsystem speedup	2.68x
Overall speedup	1.60x





Hotspur
120
90
60
30
0
Intersect incoherent
Integration
Shading
Texturing
Scalar
Vectorized
Embree rtcIntersect primary
Other
Differential geometry
Vectorization overhead
Embree rtcOccluded
Shading (excl. texturing)
Texturing
Integration
Embree rtcIntersect incoherent

Hotspur Vectorization Speedups

Ray intersection subsystem speedup	1.14x
Shading subsystem speedup	4.20x
Texturing subsystem speedup	2.90x
Integration subsystem speedup	2.72x
Overall speedup	2.31x

Conclusion

- Significant speedups over scalar mode in all scenes tested
- Vectorization speedup is scene dependent
- Future looking architecture
  - Linear scaling with increased core counts
  - Non-trivial speedups with increased vector lane counts
  - Suitable for GPU hardware

Good Luck MoonRay!

Acknowledgements: We’d like to thank the MoonRay and MoonShine teams at DWA, and the open source community

Questions?

mark.lee@dreamworks.com

Full paper is available at: research.dreamworks.com