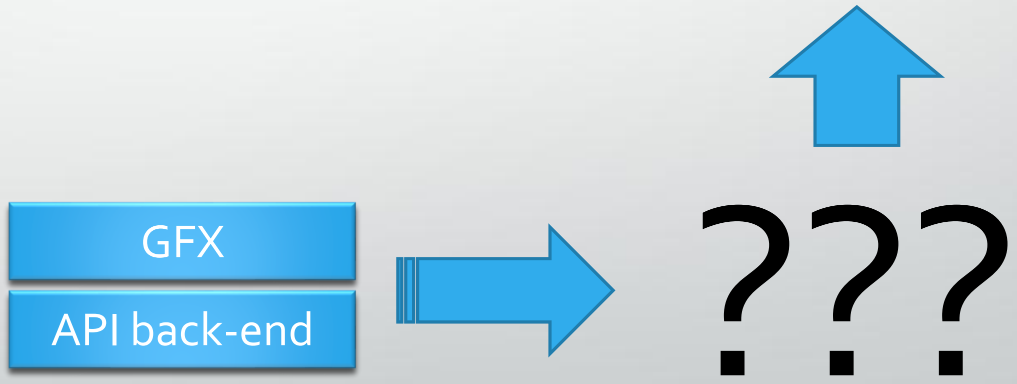
High-level Rendering

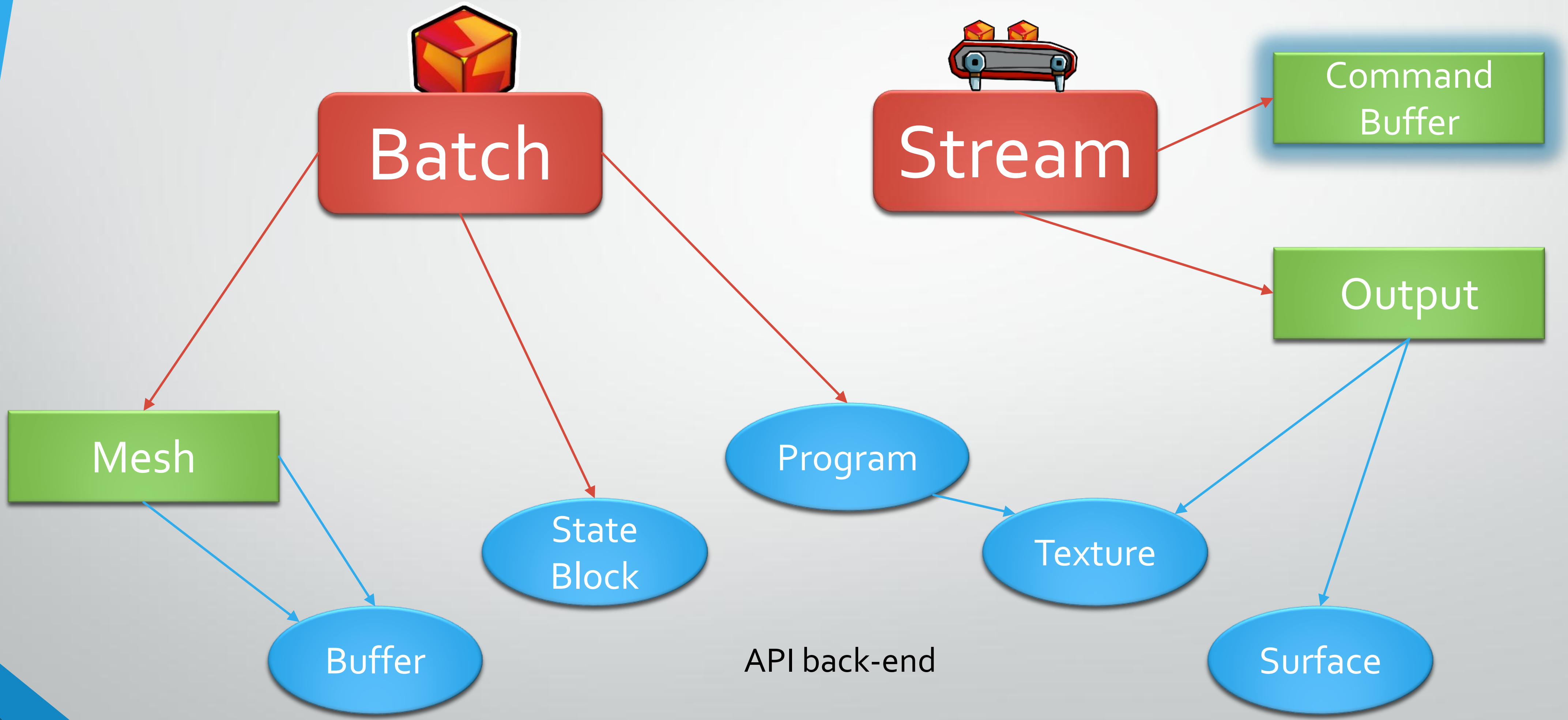
for gfx-rs

The Plan

1. Take Rust
2. Build GFX
3. ???
4. Profit!



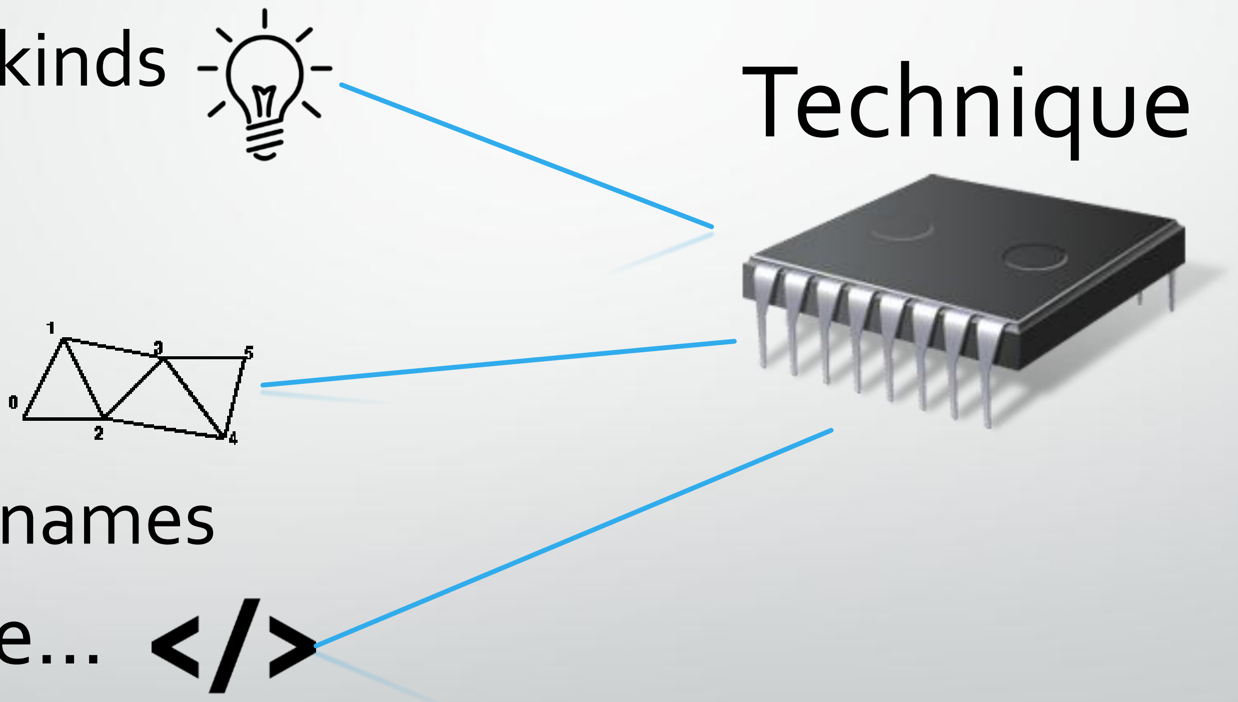
GFX level



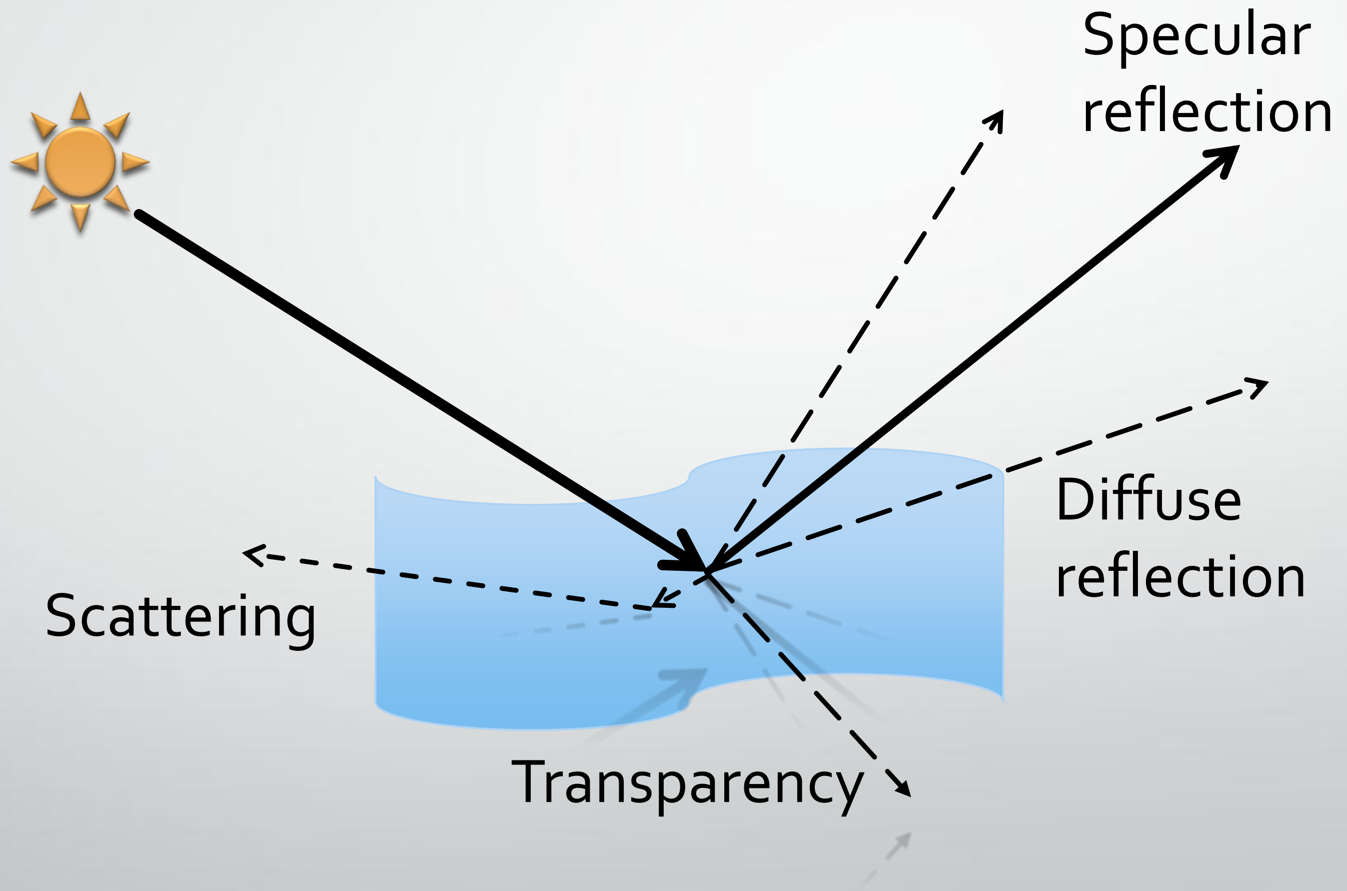
API back-end

Abstracting the Unknowns

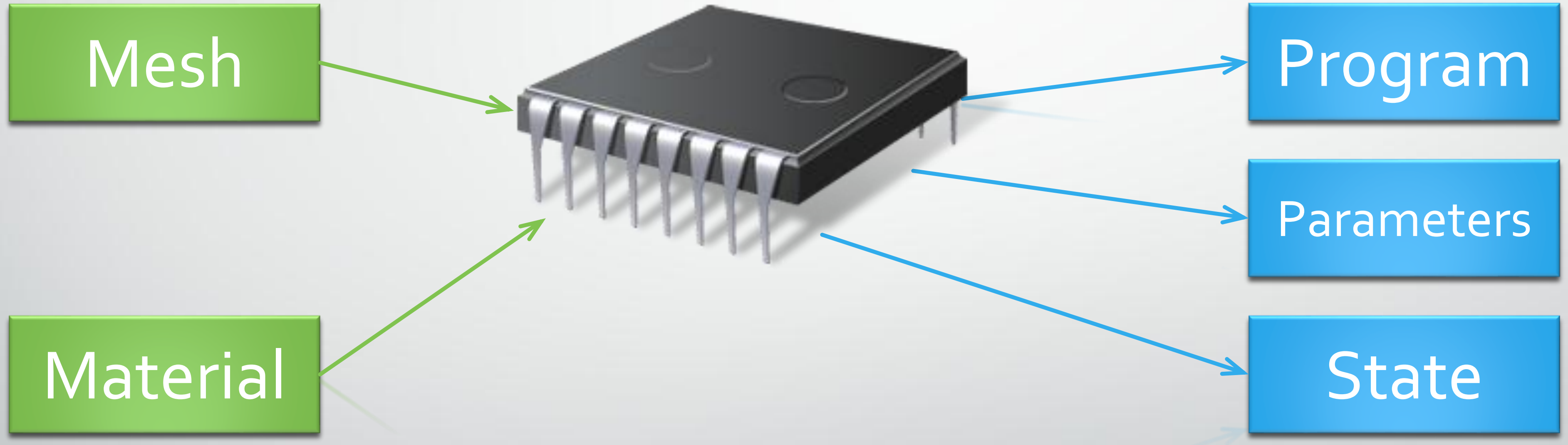
- Light information?
  - Different kinds
  - Count
- Mesh data?
  - Skinning
  - Attribute names
- Shader code…



What is a Material?



Rendering Technique



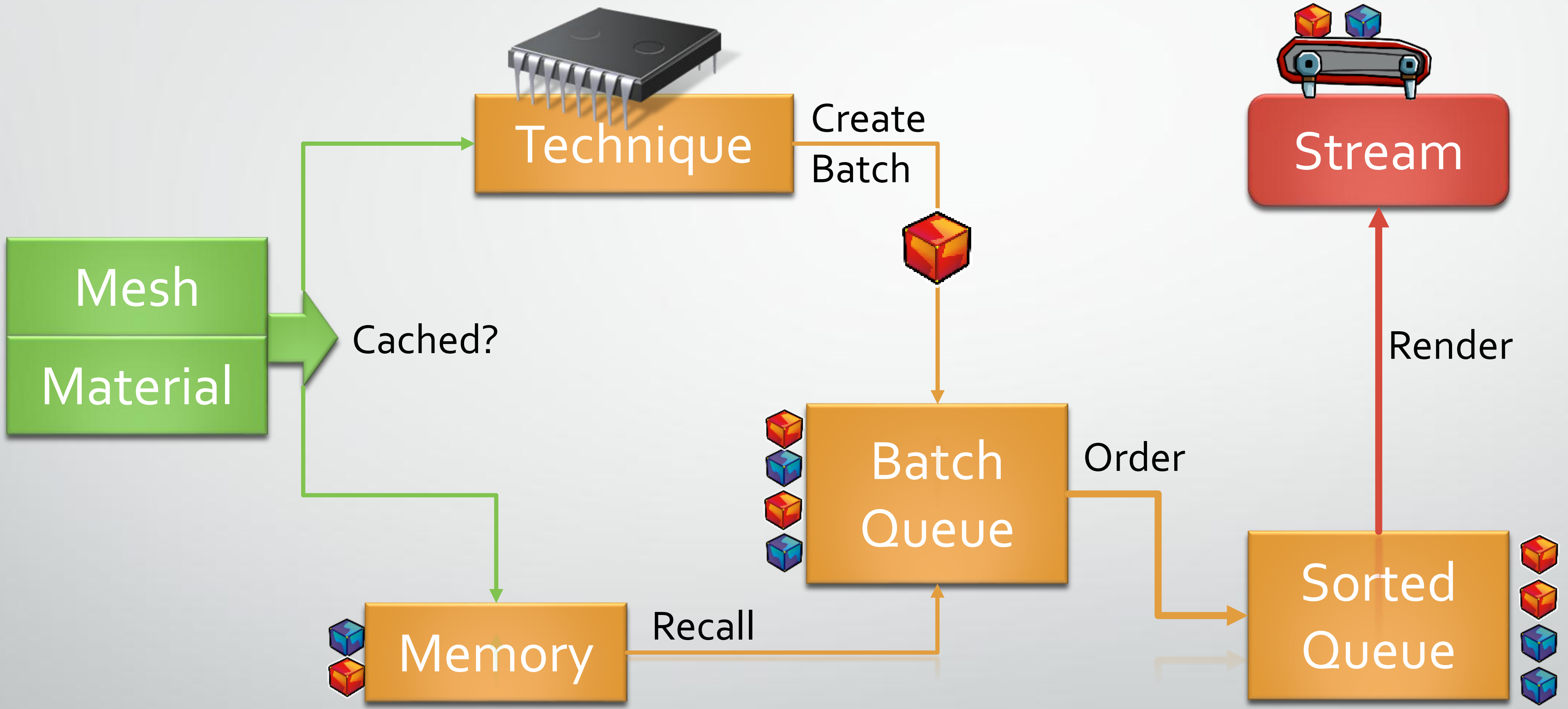
Technique API

```
pub trait Technique<R: gfx::Resources, M: Material, V: ToDepth> {
    type Kernel: Copy + Debug + Eq + Hash;
    type Params: gfx::shade::ShaderParam<Resources = R>;
    fn test(&self, &gfx::Mesh<R>, &M) -> Option<Self::Kernel>;
    fn compile<'a>(&'a self, Self::Kernel, &V) -> TechResult<'a, R, Self::Params>;
    fn fix_params(&self, &M, &V, &mut Self::Params);
}
```

Phase level

- Everything ends up being a *Batch*, which can be:
  - <u>Cached</u>: avoid batch validation cost
  - <u>Sorted</u>: avoid state switch cost
  - Sent for <u>rendering</u> into a *Stream*

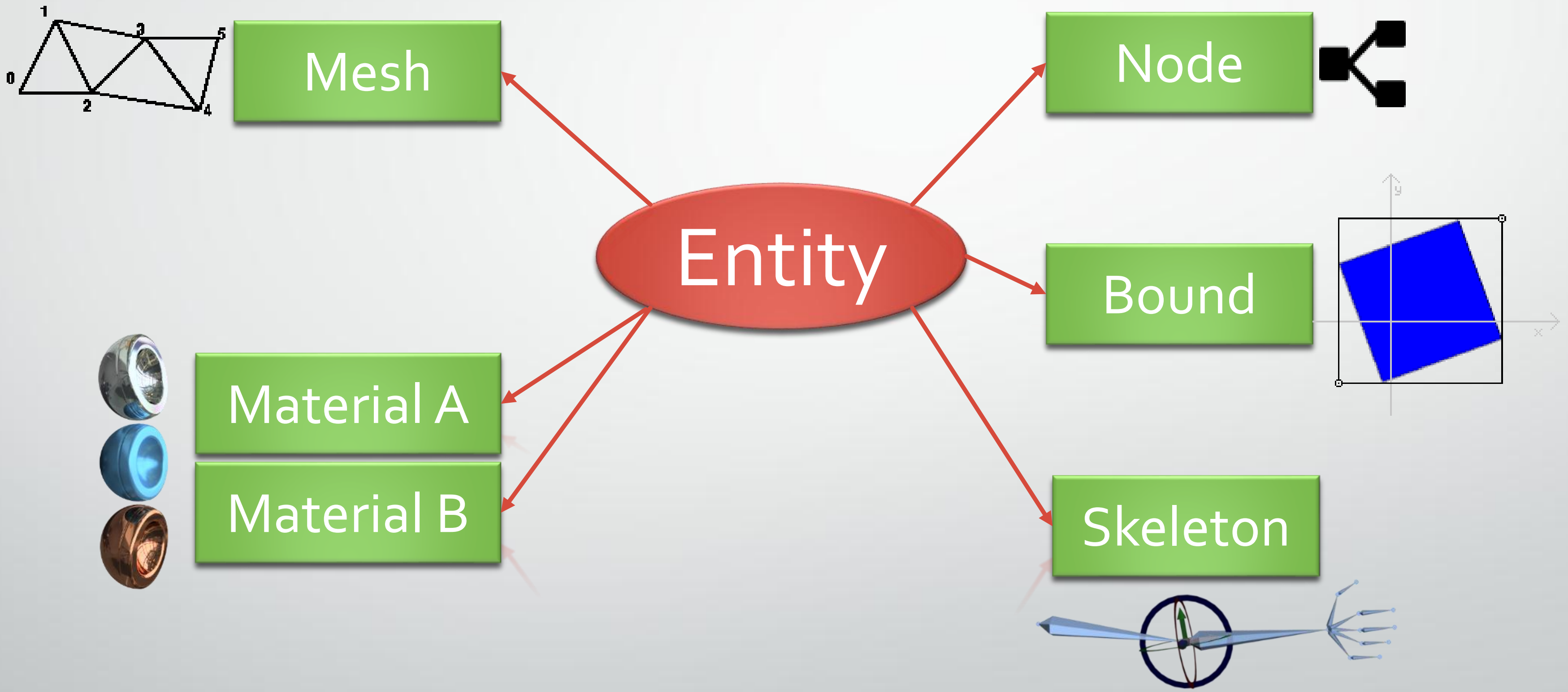
Render Phase



Scene level

- Space
  - Abstract transformations
  - Parent-child relationships
- Scenes, Cameras, Entities
  - Abstract bounds
  - Frustum culling
- Compound objects

Entity


Mesh
Node
Entity
Bound
Material A
Material B
Skeleton
0
1
2
3
4
5
y
x

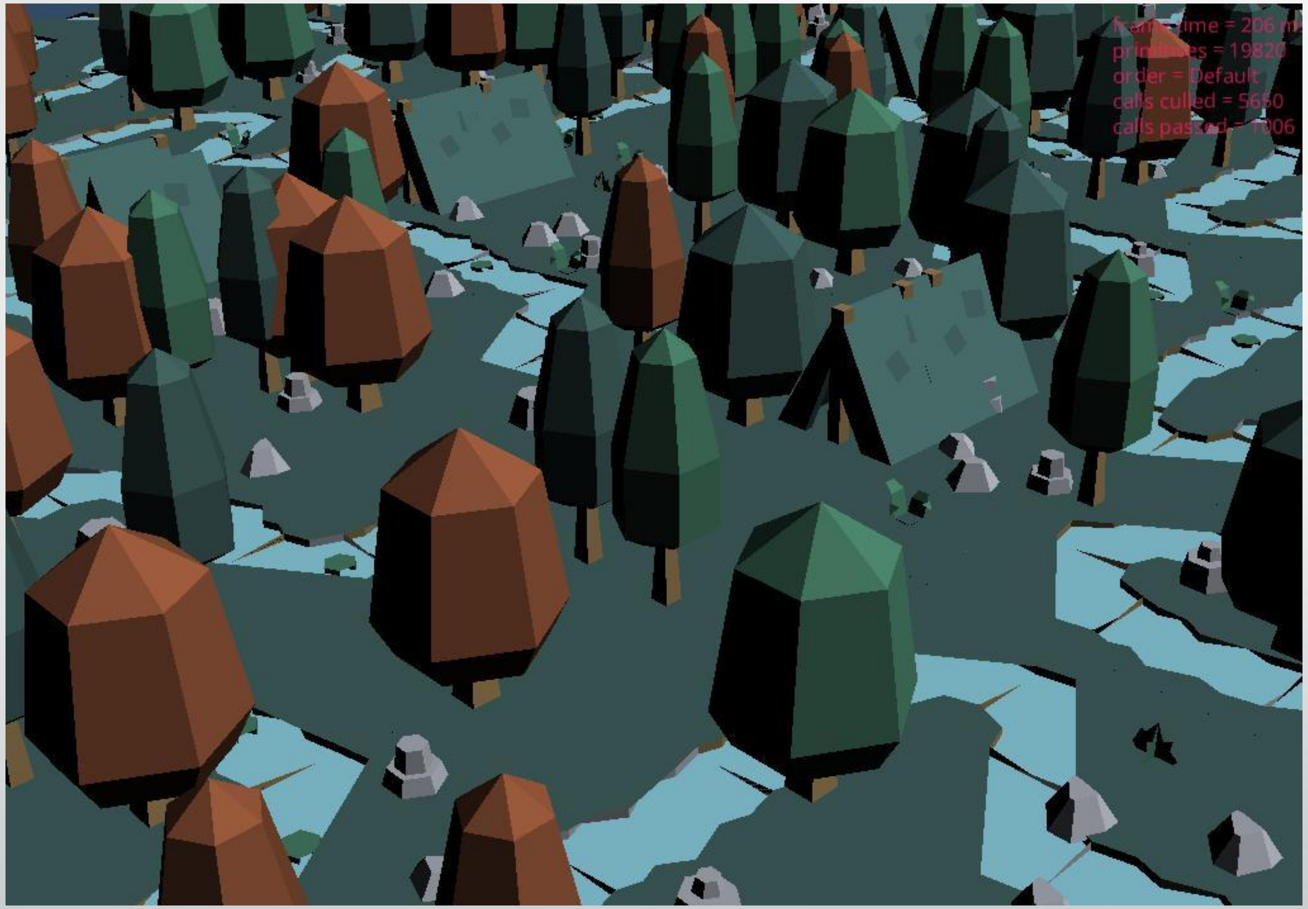
Too much Abstraction?

```
pub fn draw<'b, R, M, V, I, H, S>(&mut self, entities: I, phase: &mut H, stream: &mut S)
                -> Result<::Report, ::Error> where
  W: 'b,
  W::Transform: 'b,
  W::NodePtr: 'b,
  W::SkeletonPtr: 'b,
  B: 'b,
  R: gfx::Resources + 'b,
  ...
  M: 'b,
  V: ::ViewInfo<W::Scalar, W::Transform>,
  I: Iterator<Item = &'b ::Entity<R, M, W, B>>,
  H: gfx_phase::AbstractPhase<R, M, V>,
  S: gfx::Stream<R>,
```

GFX Pipeline

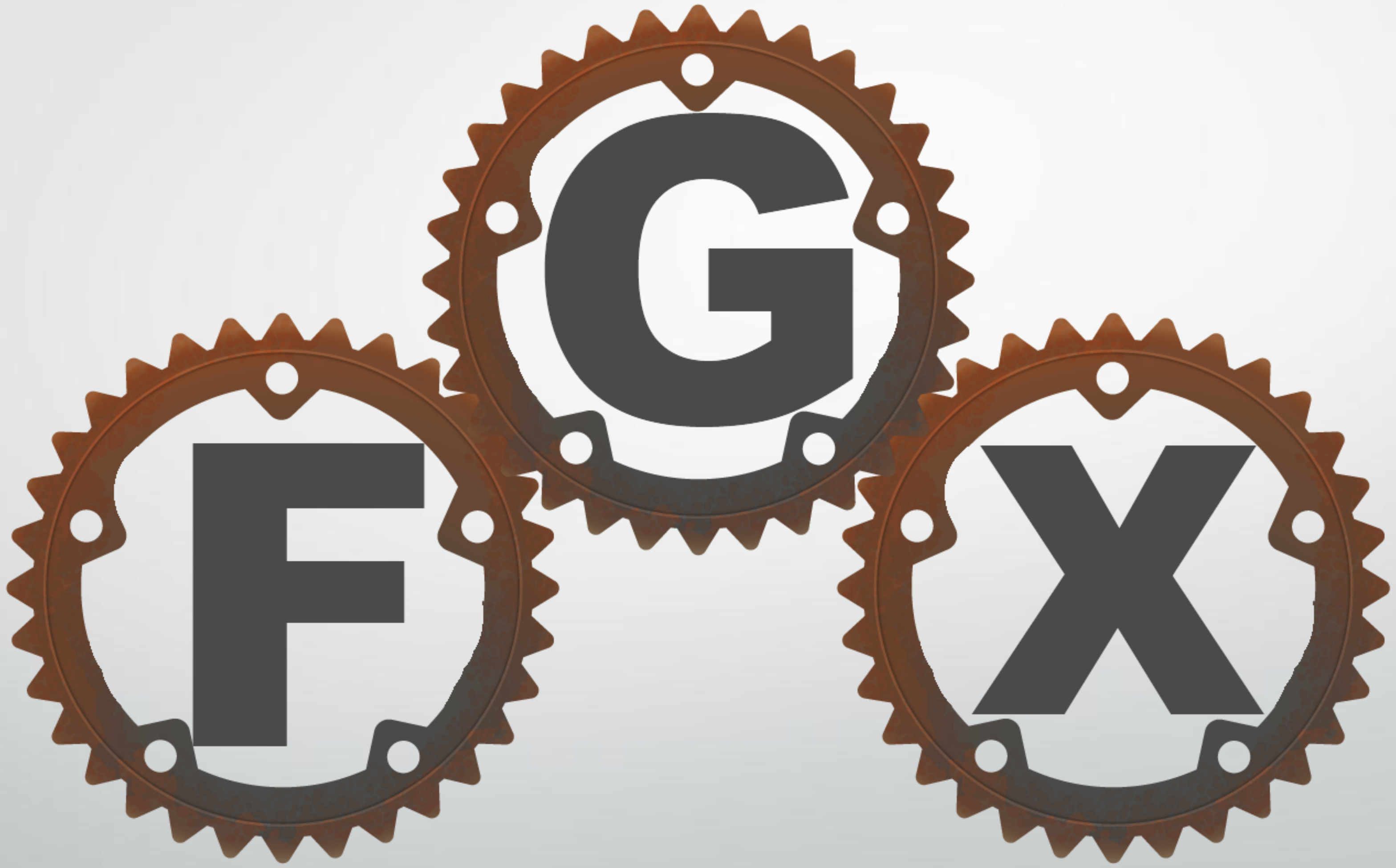
- Standard material
  - Physically Based Rendering (*)
- Complete rendering solutions
  - Forward renderer (*)
  - Forward+/Clustered renderer (**)
  - Deferred renderer (**)
- Lots of TODO!

Forest demo



```
frame time = 206 m
primitives = 19820
order = Default
calls culled = 5660
calls passed = 1006
```

Why Rust?



Links

- GFX Scene: https://github.com/kvark/gfx_scene
  - *Technique/Phase*
  - *Entity + Scene*
  - persistent draw queue
- GFX Pipeline: https://github.com/kvark/gfx_pipeline
  - standard material
  - basic rendering
- Claymore: https://github.com/kvark/claymore
  - standard scene/world
  - *Blender* export + import
  - applications