C++ONLINE

CHRISTOPH STILLER

UNDERSTANDING THE ART OF API DESIGN

BUILDING INTERFACES FOR HUMANS ON EARTH

2026

Why is this important

- APIs shape how other code *can* look
- Mess up in implementation: One File change & review;
  Mess up in interface → All users need change & review
- Bad APIs waste time: Documentation & common Bugs
- Catching messy code is usually trivial;
  Catching bad APIs & Architecture is much more difficult

What does a good API do?

Simplifies the mental model

Easy to understand what can be done

Provides what you need

Easy to understand how it’s done

Encourages readable code

Hard to use incorrectly

No nasty surprises

Simplifies the mental model
Easy to understand what can be done
Provides what you need
Easy to understand how it’s done
Encourages readable code
Hard to use incorrectly
No nasty surprises

Simplifies the mental model

Easy to understand what can be done

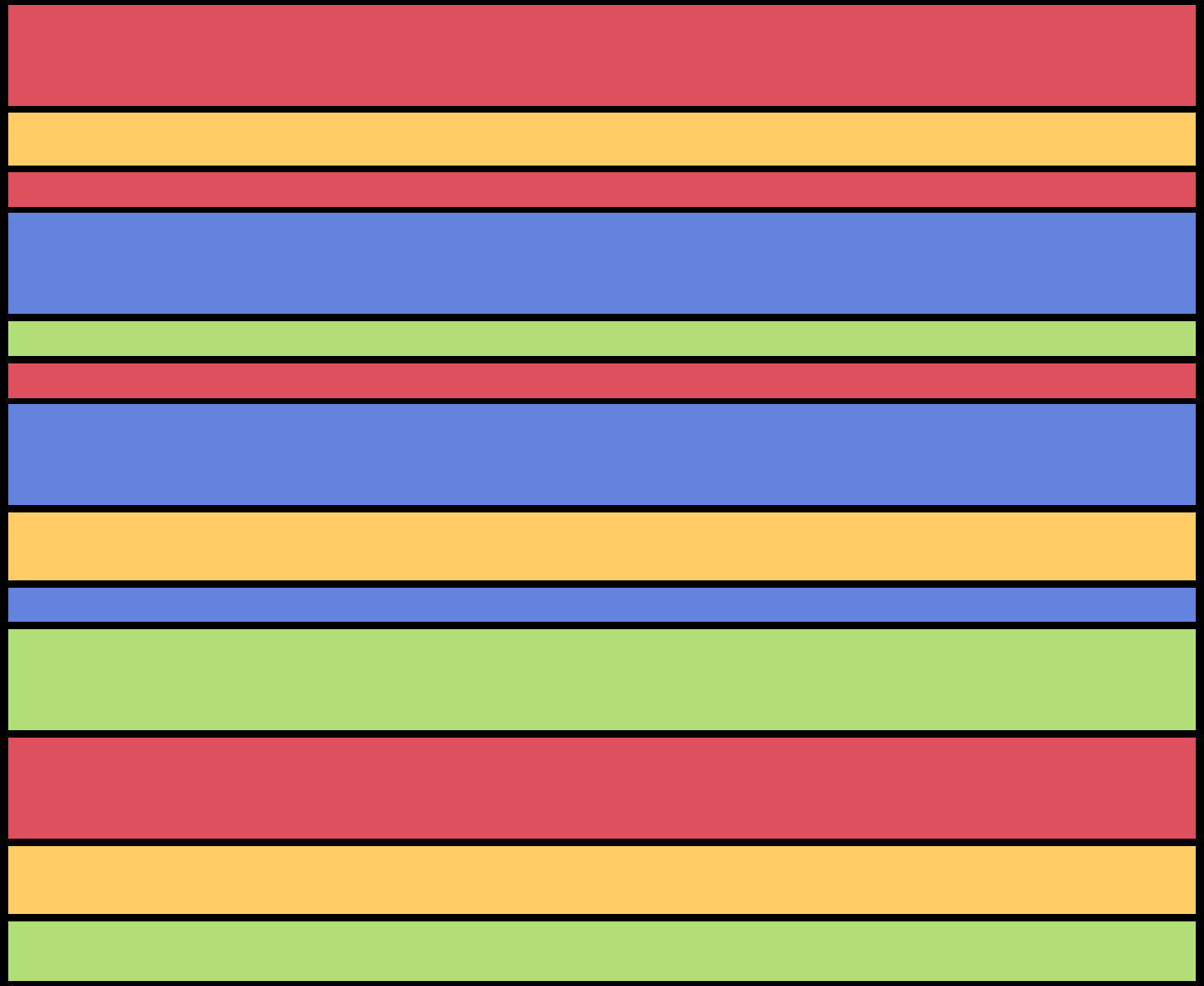
Provides what you need

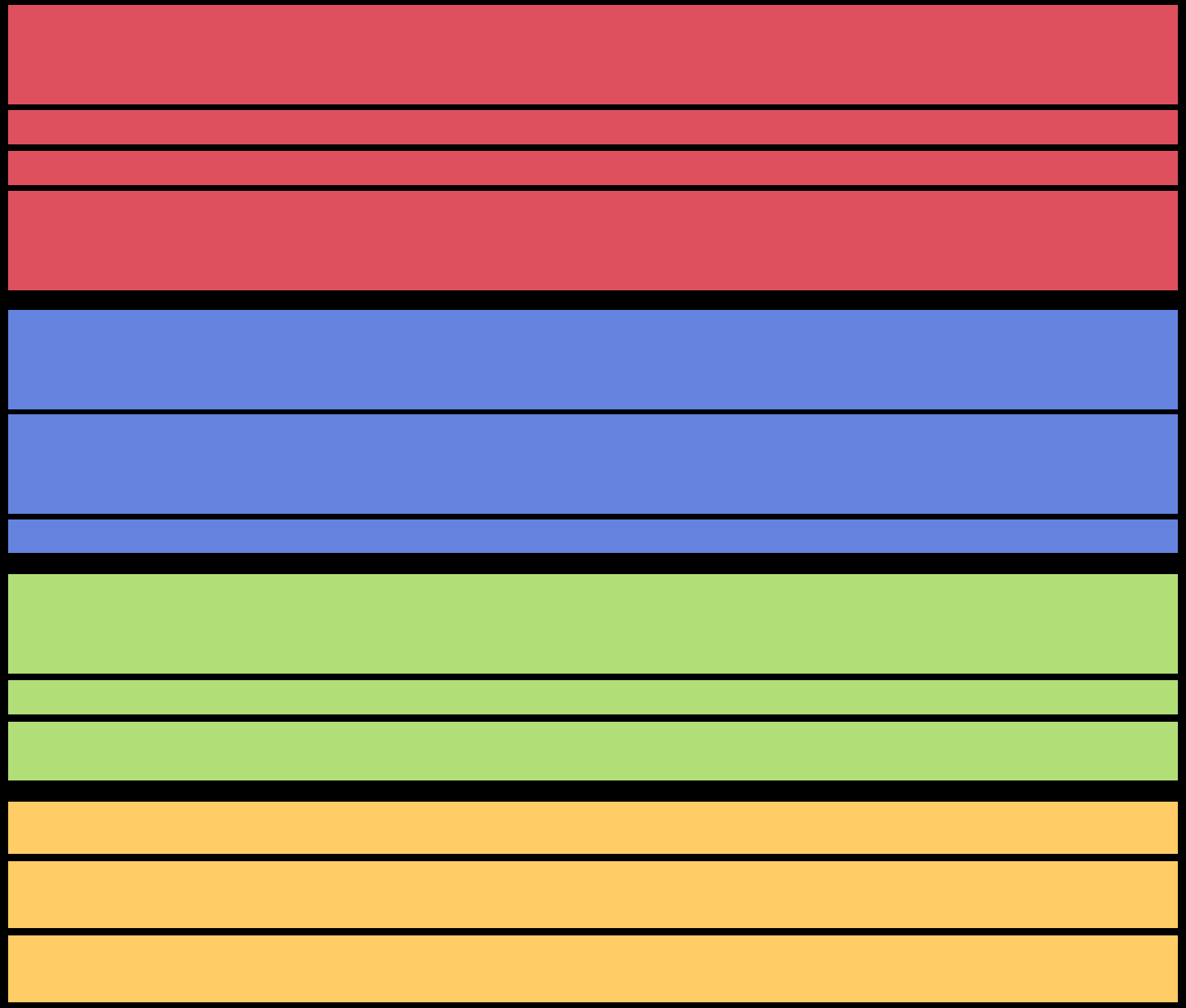
Easy to understand how it's done

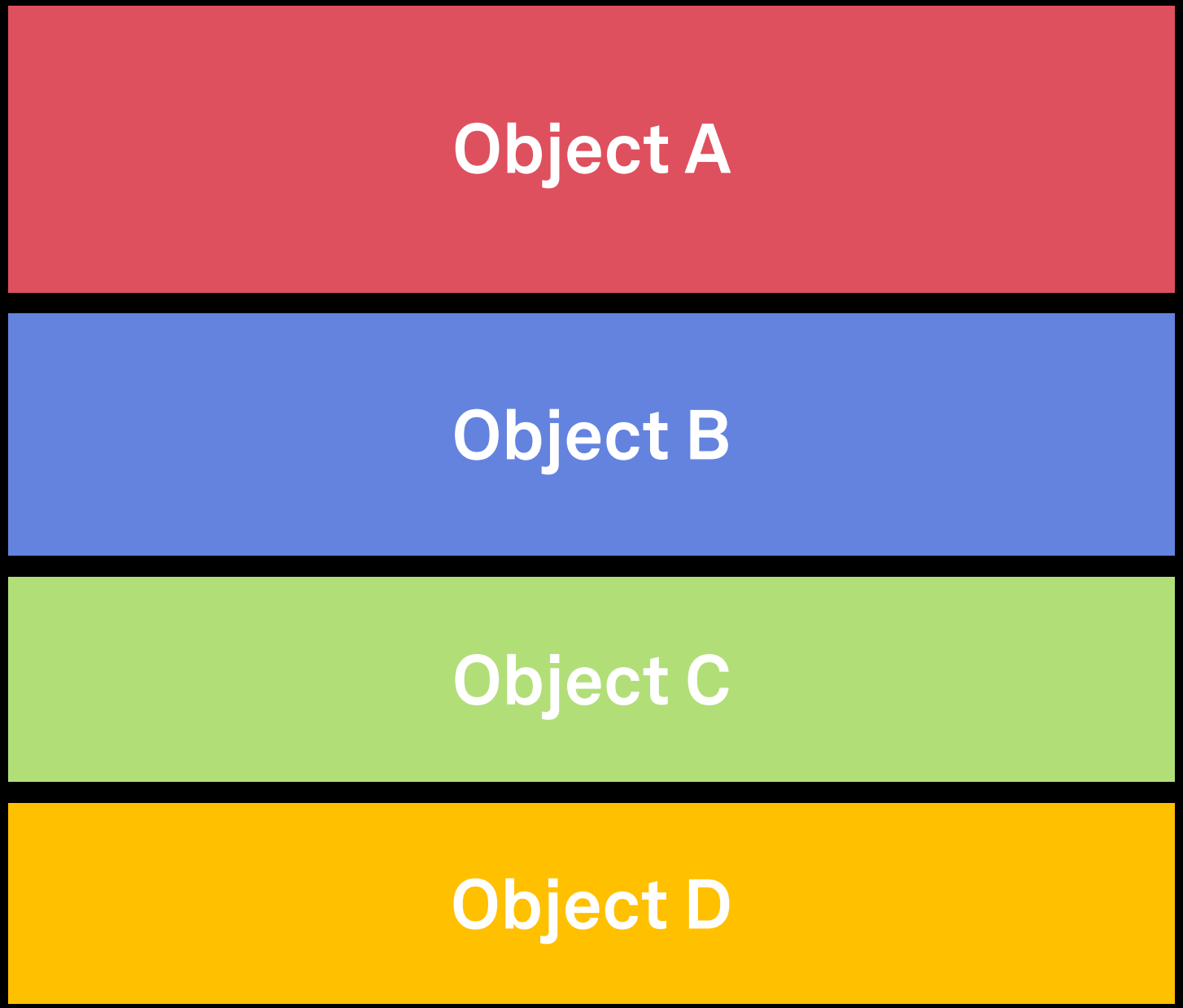
Encourages readable code

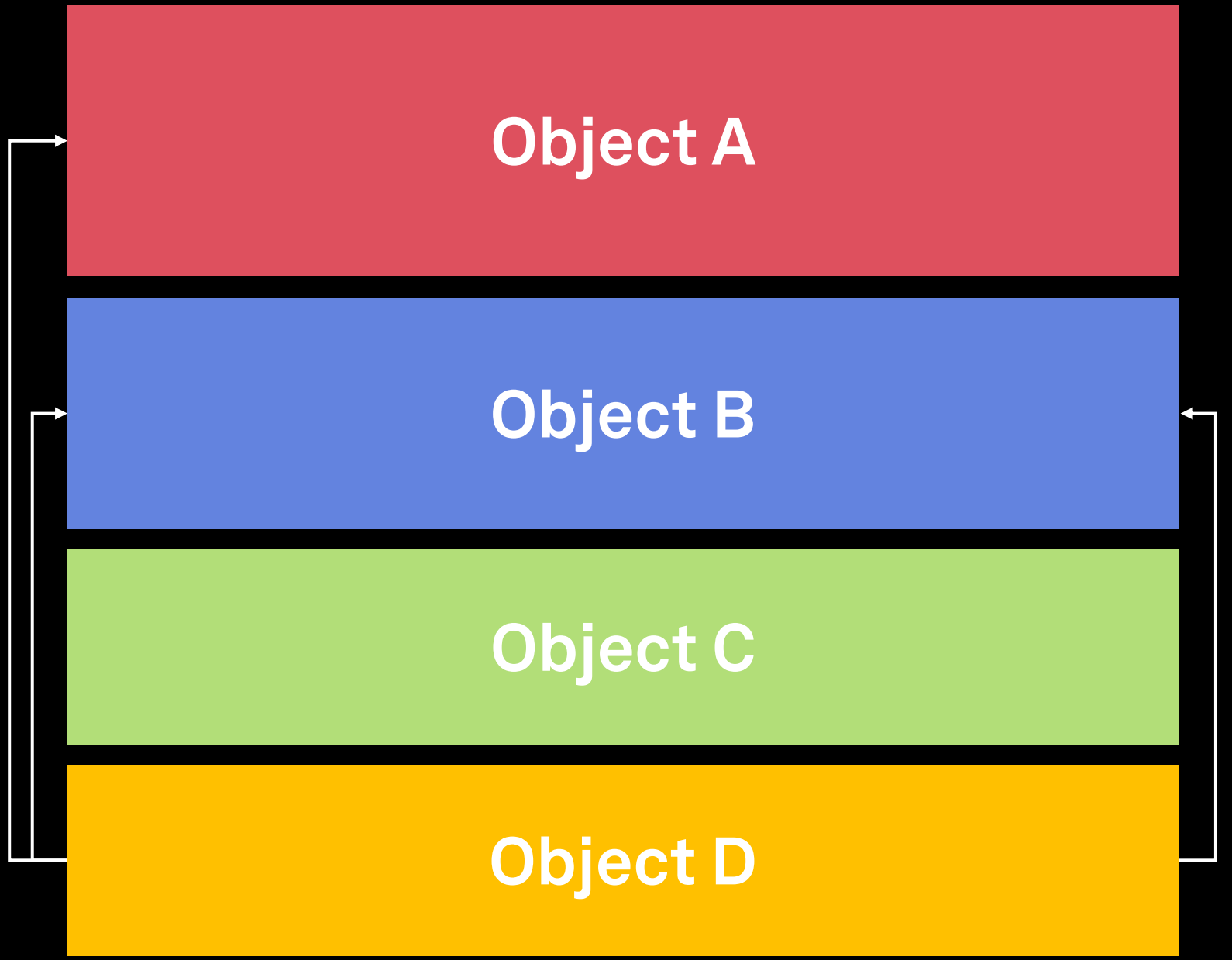
Hard to use incorrectly

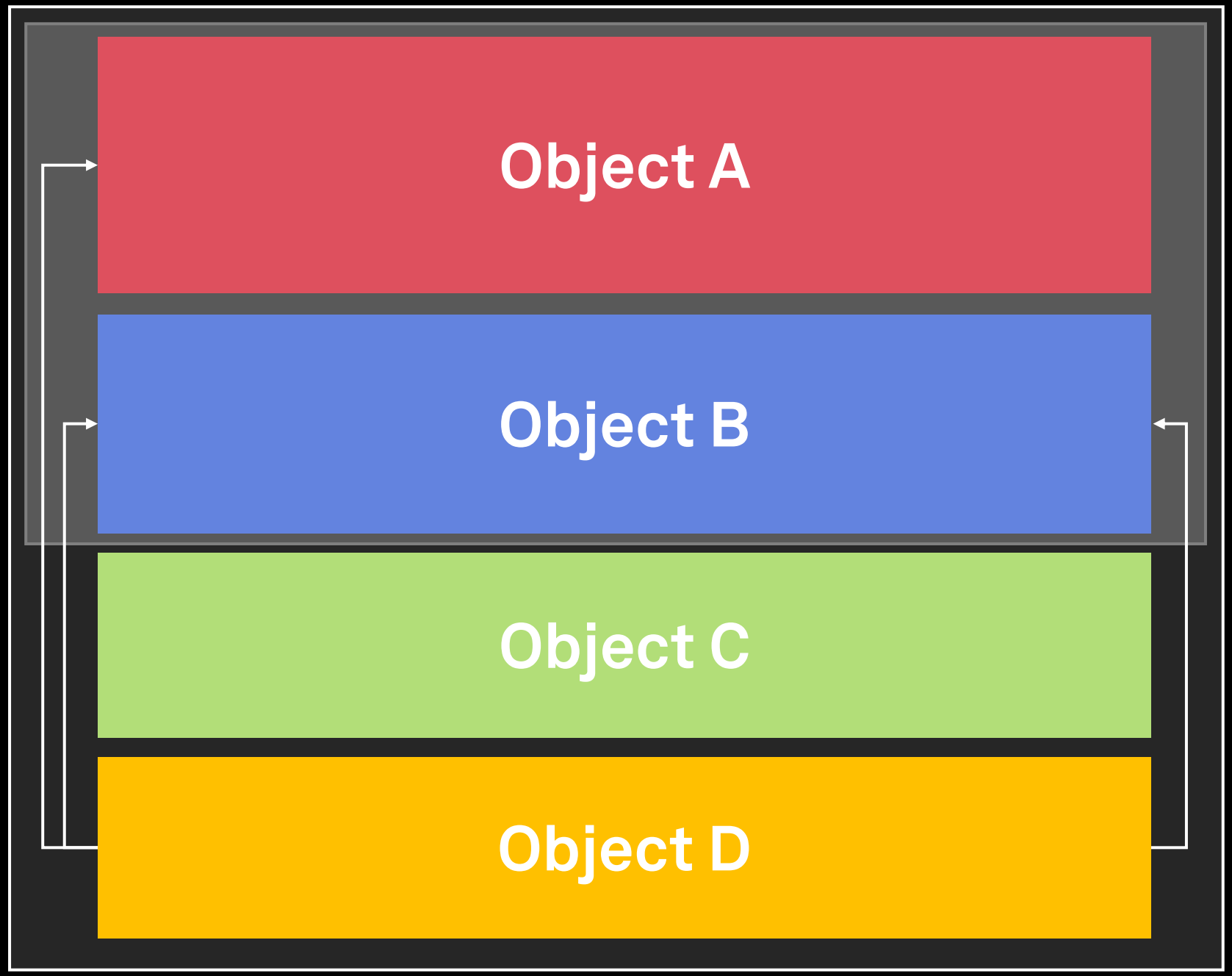
No nasty surprises

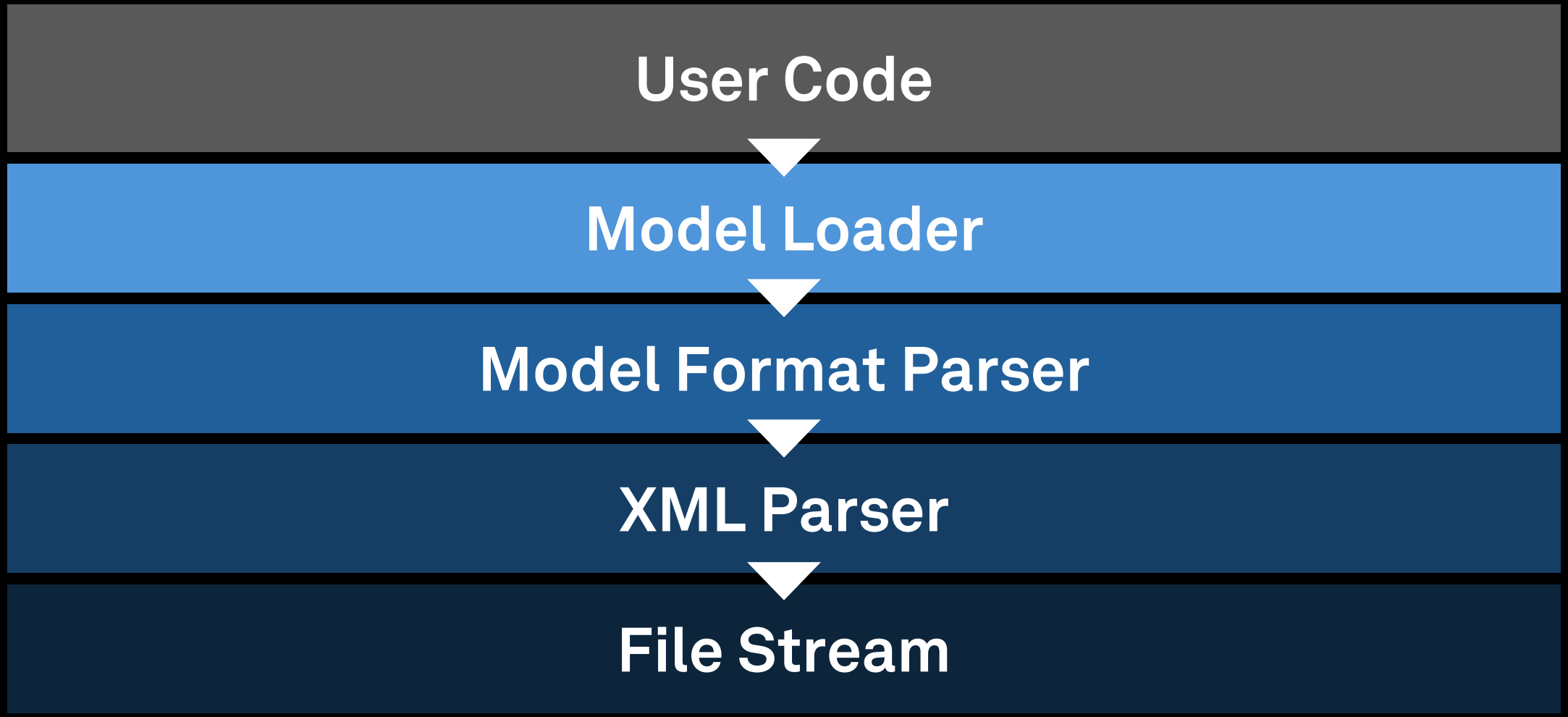


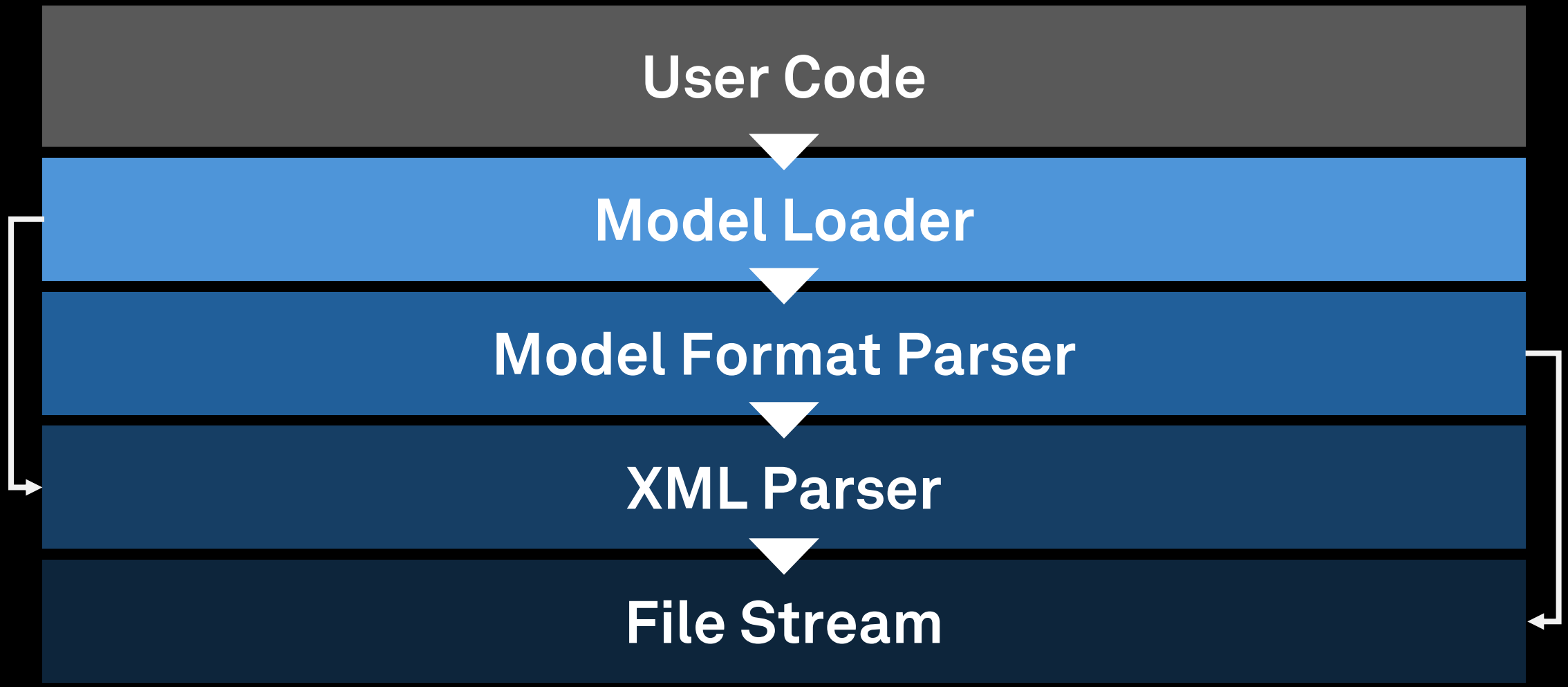


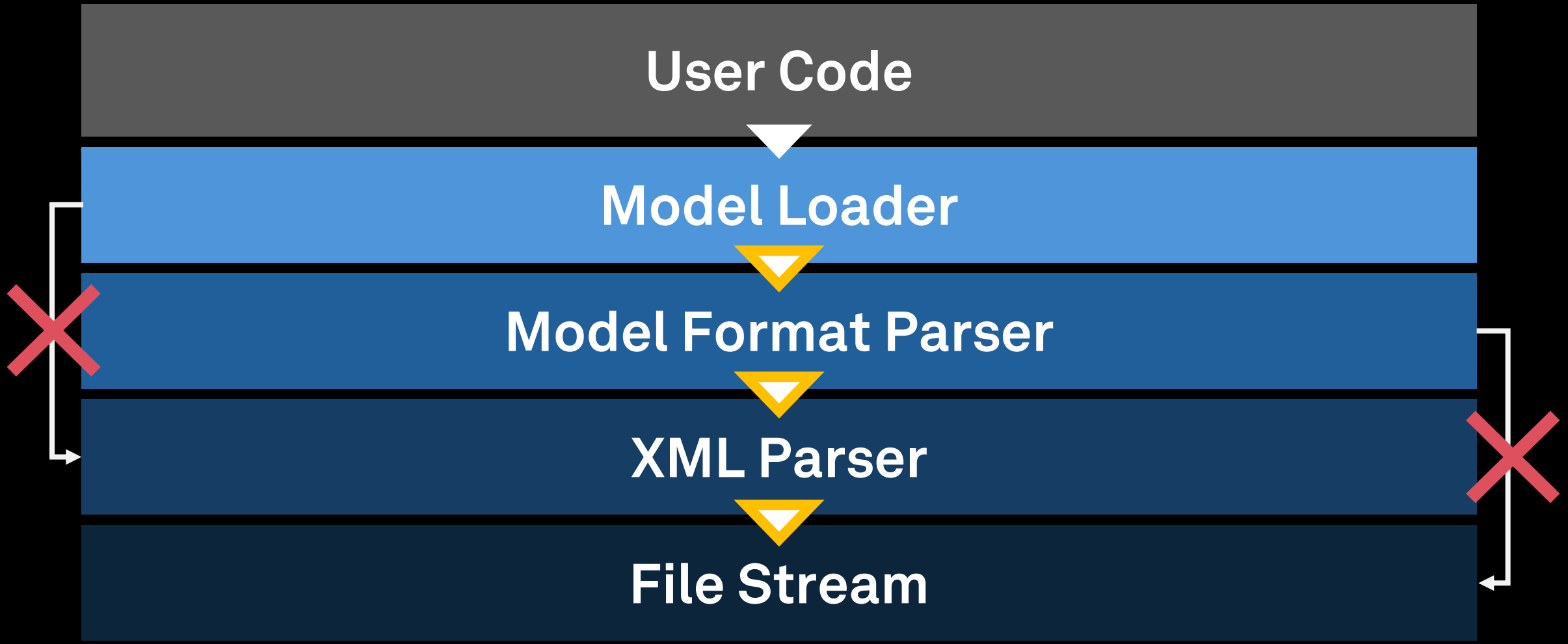

Object A
Object B
Object C
Object D

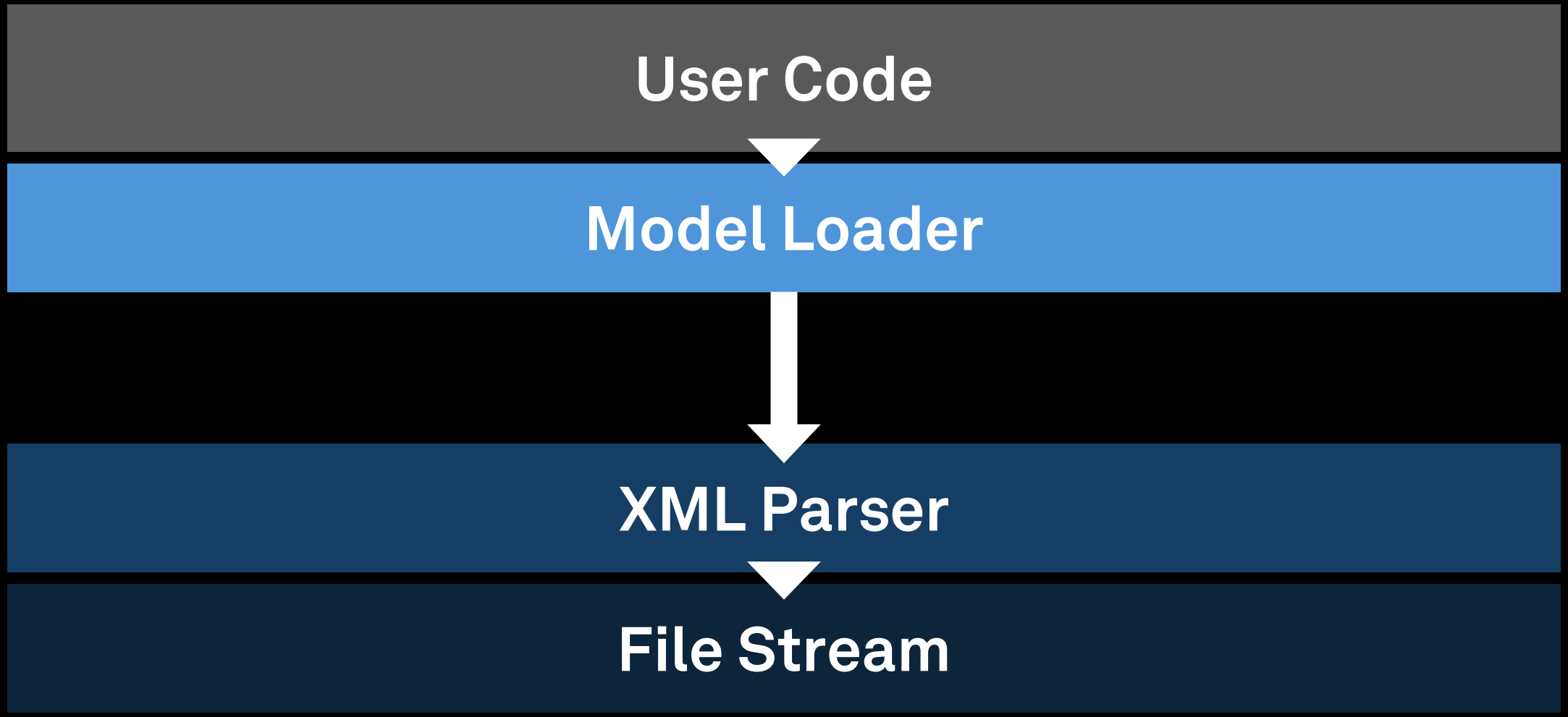

Object A
Object B
Object C
Object D

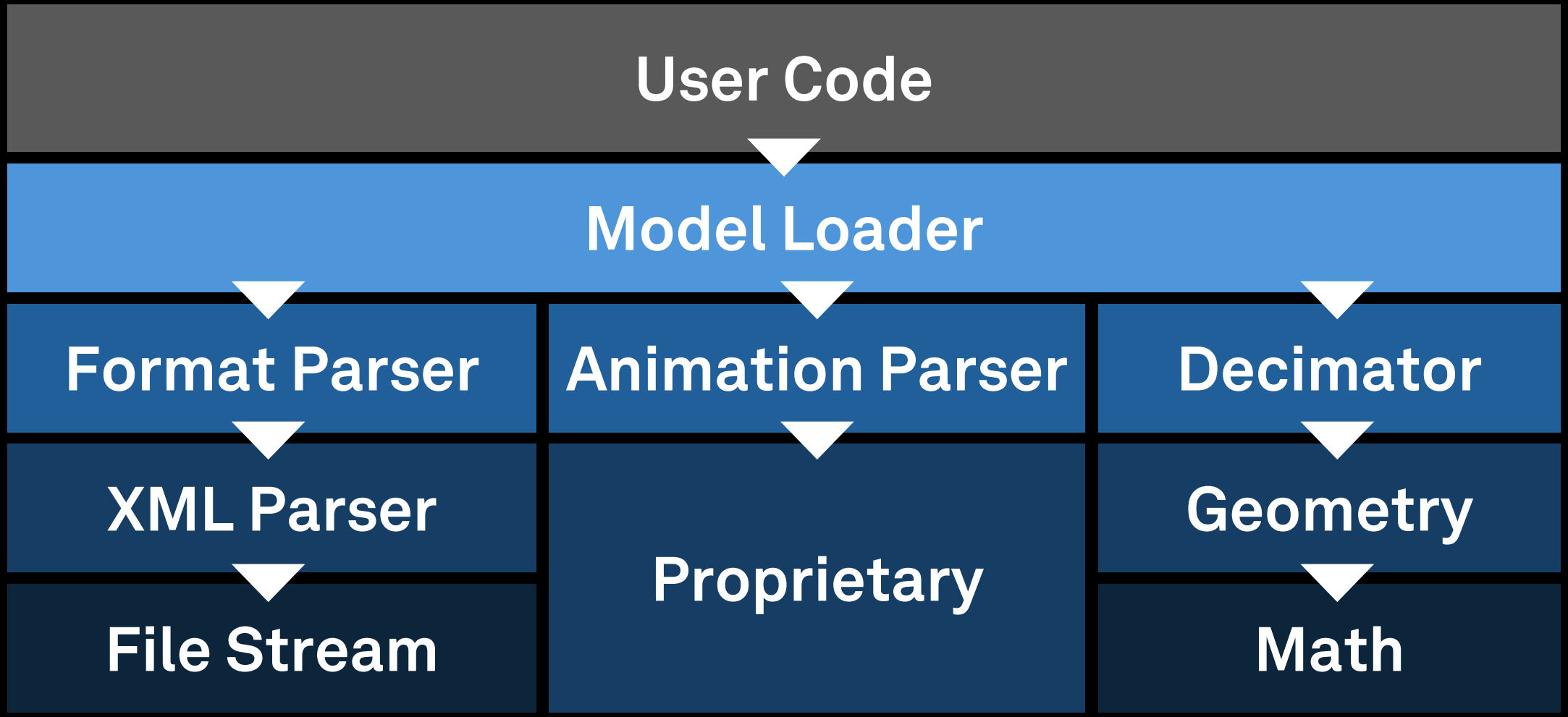

Object A
Object B
Object C
Object D

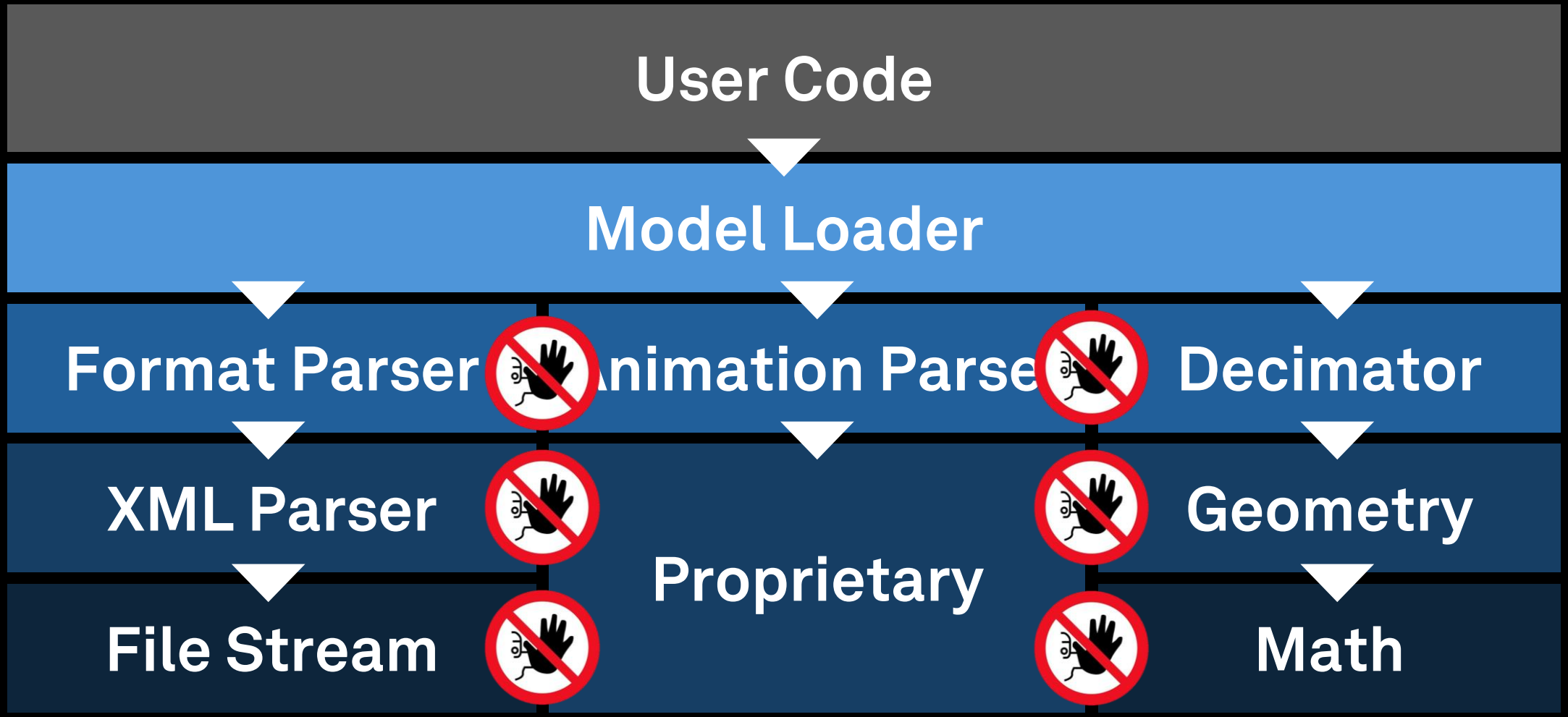

User Code
Model Loader
Model Format Parser
XML Parser
File Stream


User Code
Model Loader
Model Format Parser
XML Parser
File Stream


User Code
Model Loader
Model Format Parser
XML Parser
File Stream


User Code
Model Loader
XML Parser
File Stream


User Code
Model Loader
Format Parser
Animation Parser
Decimator
XML Parser
Proprietary
Geometry
File Stream
Math


User Code
Model Loader
Format Parser
nimation Parse
Decimator
XML Parser
Geometry
Proprietary
File Stream
Math

https://de.wikipedia.org/wiki/Eis#/media/Datei:Fryxellsee_Opt.jpg

https://en.wikipedia.org/wiki/Boiling#/media/File:Boilingkettle.jpg

Simplifies the mental model

- Clear purpose for your API?
- Separate Layers vertically AND horizontally
- Prohibit “Spaghetti” Calls
- Find the *right* amount of abstraction & separation

Simplifies the mental model

Easy to understand what can be done

Provides what you need

Easy to understand how it's done

Encourages readable code

Hard to use incorrectly

No nasty surprises

Simplifies the mental model

Easy to understand what can be done

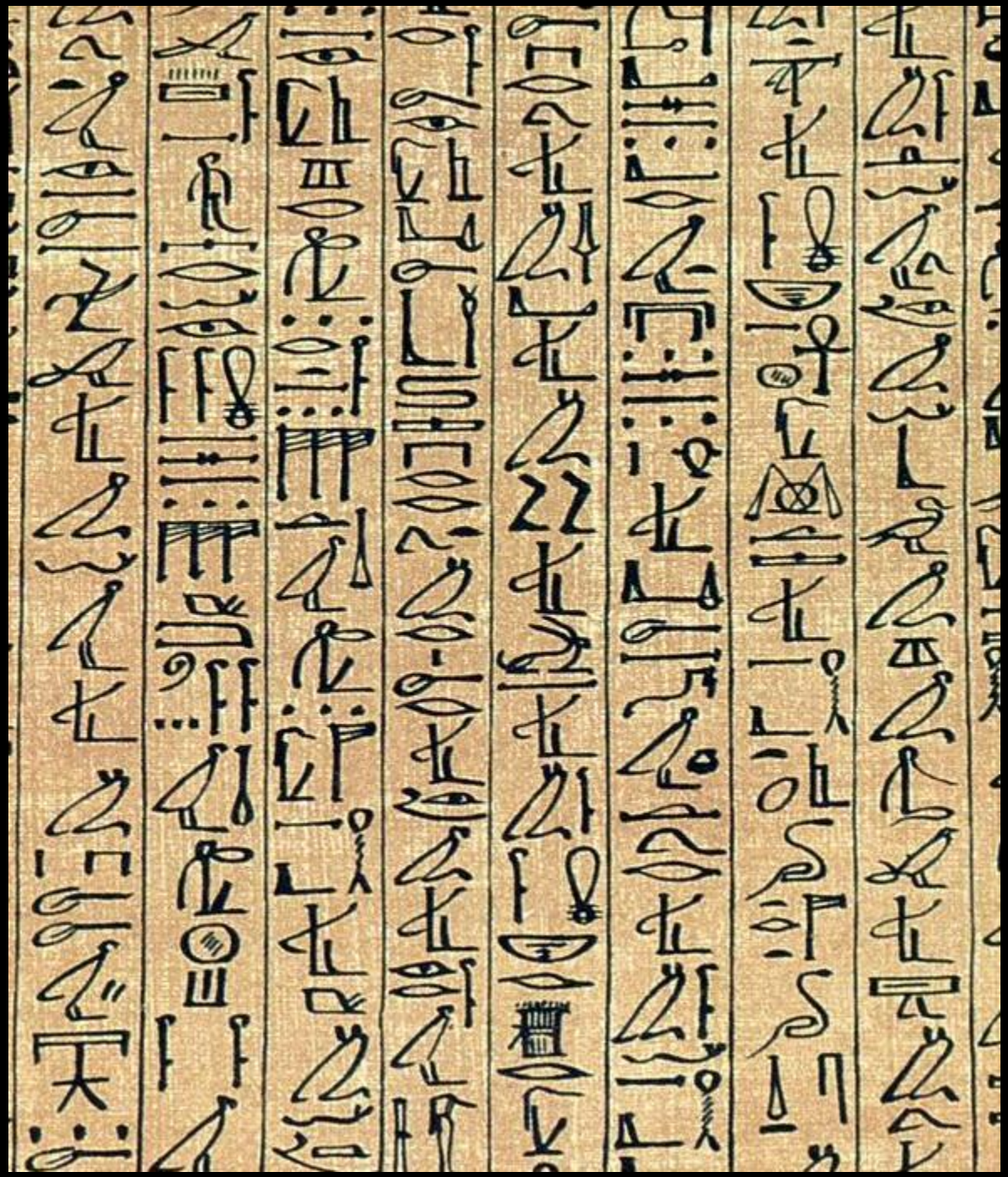
Provides what you need

Easy to understand how it's done

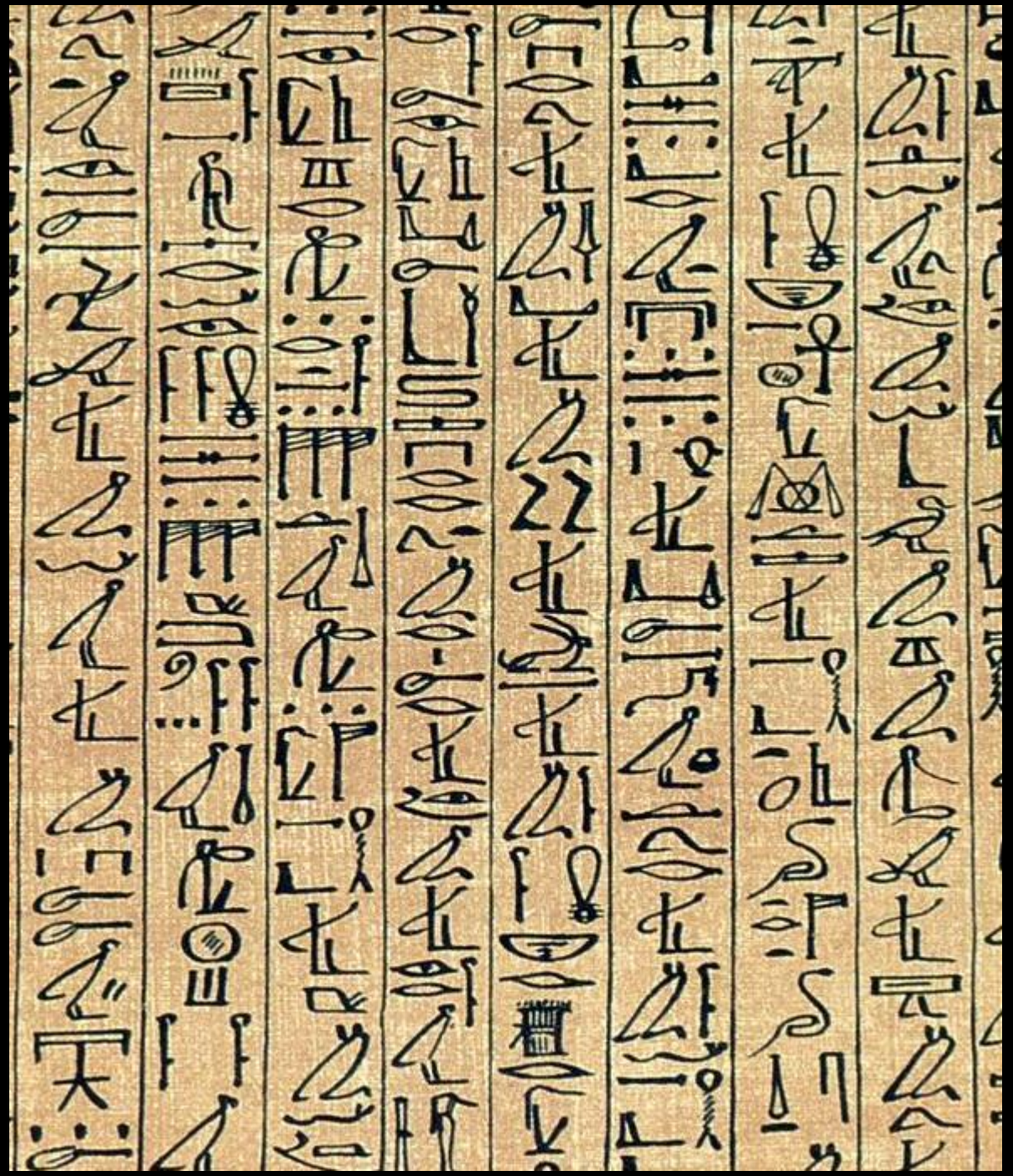
Encourages readable code

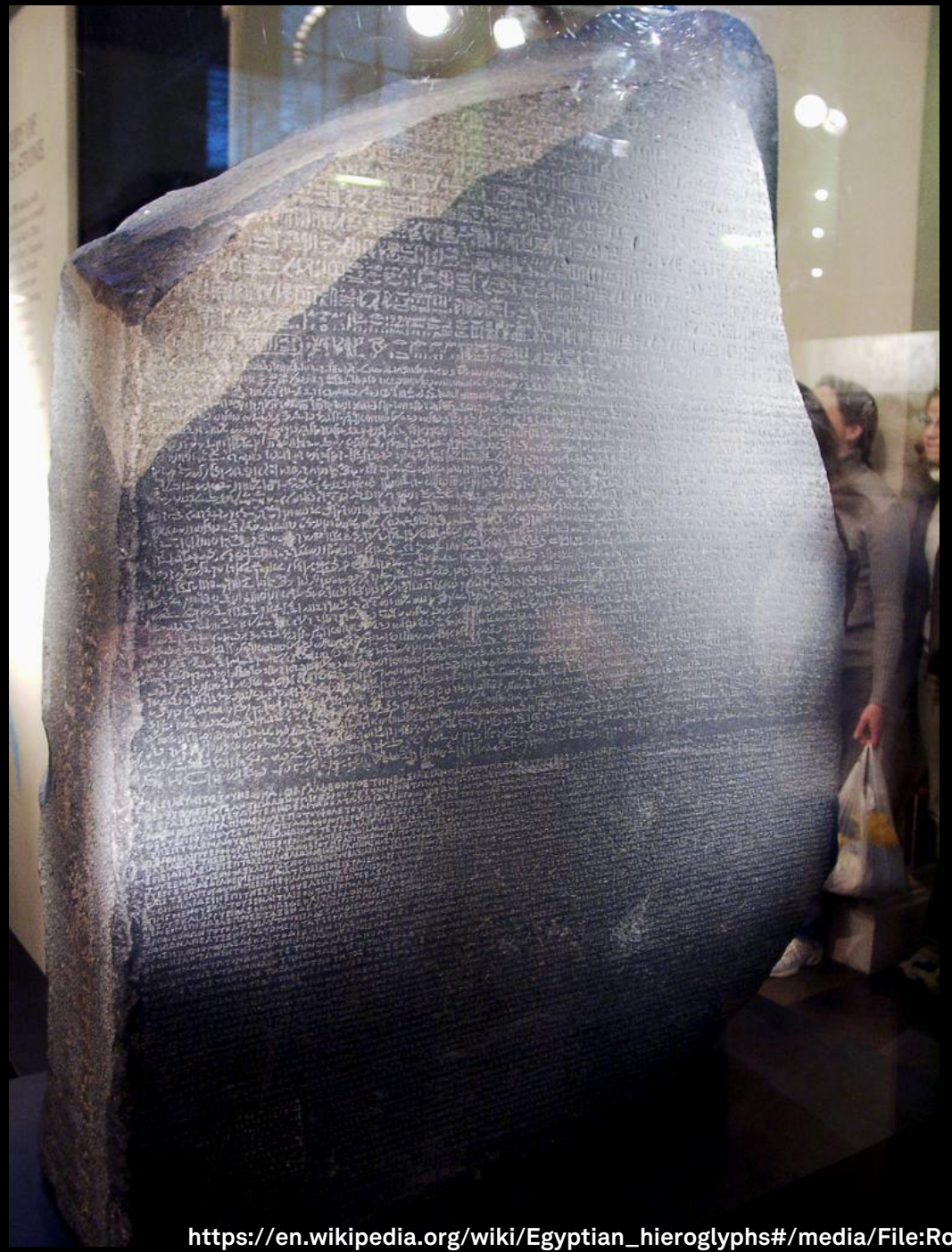
Hard to use incorrectly

No nasty surprises

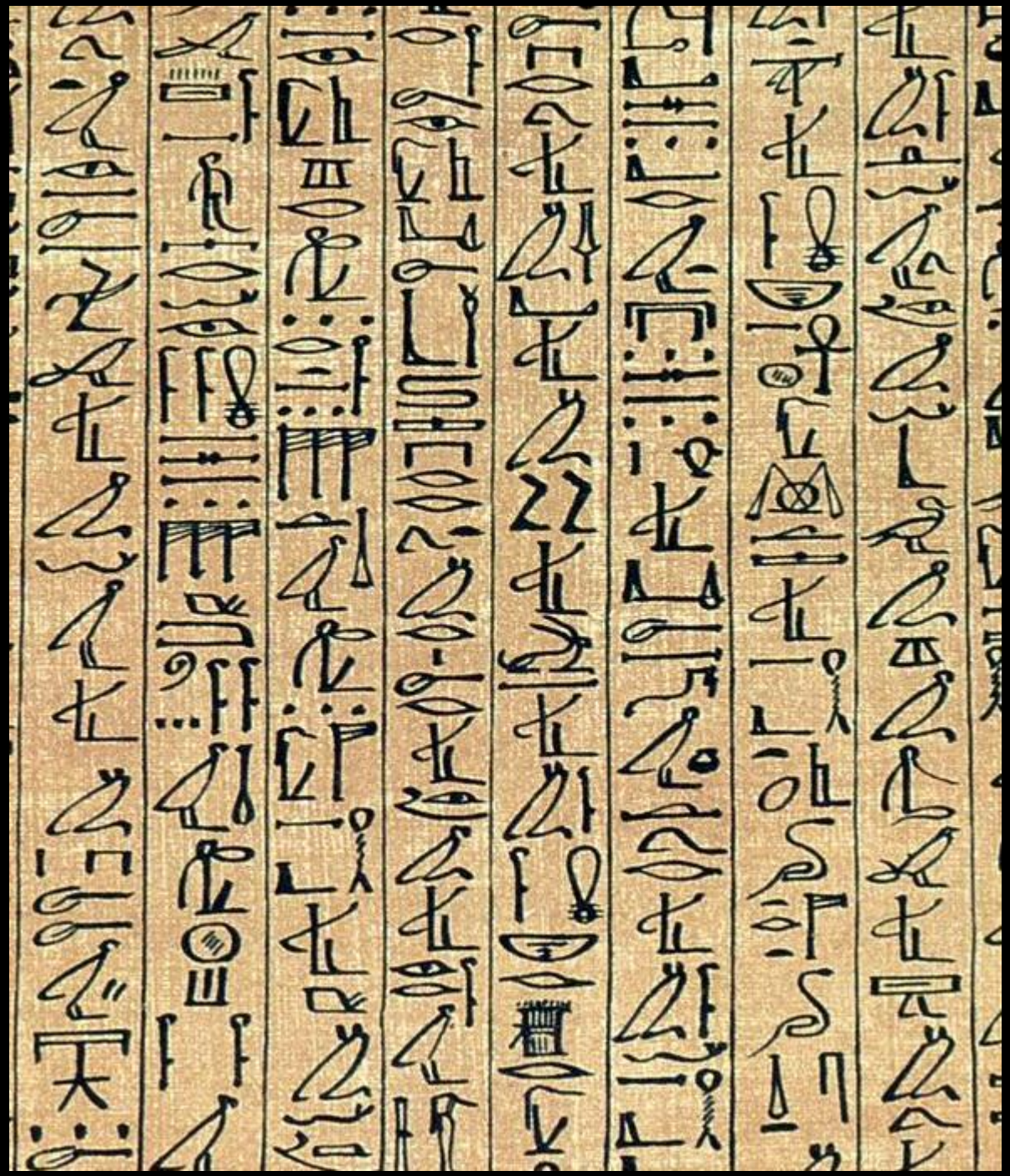


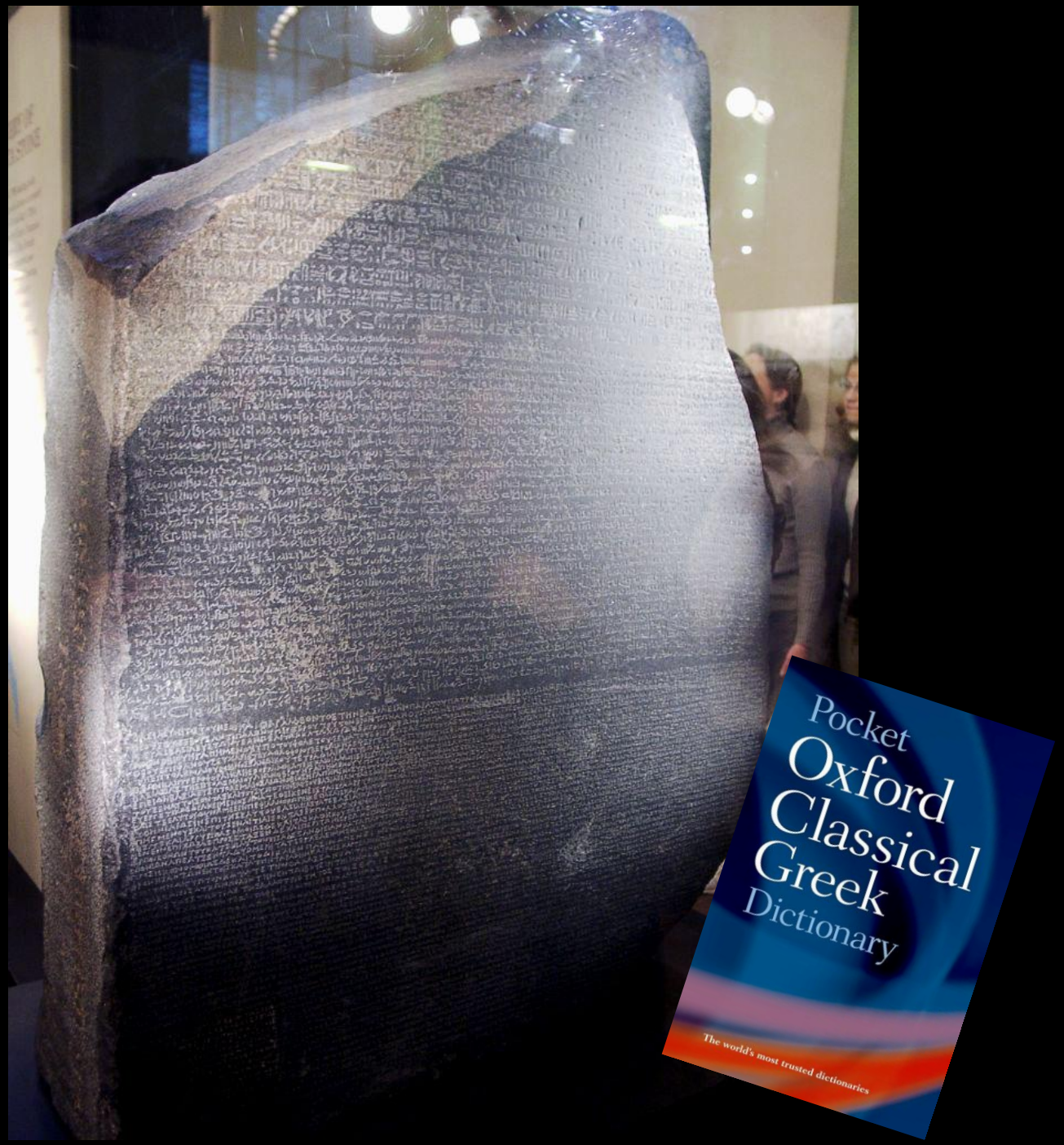
https://en.wikipedia.org/wiki/Cursive_hieroglyphs#/media/File:Papyrus_Ani_curs_hiero.jpg





https://en.wikipedia.org/wiki/Egyptian_hieroglyphs#/media/File:Rosetta_stone.jpg




Pocket
Oxford
Classical
Greek
Dictionary
The world's most trusted dictionaries

“And of some have compassion,
making a difference…”

“And of some have compassion, making a difference…”

If a person makes a difference, they make a situation better because of their good actions:

“And of some have compassion,
making a difference…”

“And of some have compassion,
making a difference…”

make a difference

1 Distinguish or discriminate. This phrase appears in the Bible (Leviticus 11:47): “To make a difference between the unclean and the clean.” [Late 1500s]

struct

class

struct class

```
struct config;
config cfg;
cfg.threads = 2;
cfg.wait_for_cmd = true;
```

struct

```
struct config;
config cfg;
cfg.threads = 2;
cfg.wait_for_cmd = true;
```

class

```
class buffer;
buffer data(src);
// ...
data.position = 0; // ?
// why not `.reset()` ?
```

*

&

```
int list_add(auto list, item* i)
```

Easy to understand

- APIs have a target audience
  - Domain Experts / Regular Developers / Beginners
  - Requirements for Configuration
  - C++23 vs C-Style
  - std::optional vs bool + value&

Easy to understand

- APIs have a target audience
  - Domain Experts / Regular Developers / Beginners
  - Requirements for Configuration
  - C++23 vs C-Style
  - std::optional vs bool + value&
- Developers are (somewhat) human
  - Limited Context Window
  - Proximity implies Relation

Tools

- Folder & File Structure
- Namespaces, Module (-Exports), Objects, Enums, Functions
- Names
- Order & Position in Files & Objects
- Comments on and between Objects & Functions
- templates, concepts, public typedefs
- begin()/end()

Keep your scope focused

```
class file
```

- Exists
- Size
- Properties
- Reading, Writing
- Copying
- Analyzing Read/Write Throughput
- Transactional Filesystem Interactions
- Memory Mapping Segments

Keep your scope focused

➔ Do one thing and do it well

Names

- After what it *represents/does* (!= *contains/is*)
- Put into one sentence ➔ derive concise name
- Better long and precise, than short and generic/misleading
- Consider expectations from a thing named X
- Make a word *mean* something across the Architecture

Easy to understand what can be done

- Know your target audience & consider their understanding
- Help users via Folder & File structure, Ordering & Separation
- Do one thing and do it well!
- Names:
  - What a thing represents (API, Object) / does (Function, Parameter)
  - Make terms mean something across the API

Simplifies the mental model
Easy to understand what can be done
Provides what you need
Easy to understand how it’s done
Encourages readable code
Hard to use incorrectly
No nasty surprises

Simplifies the mental model
Easy to understand what can be done
Provides what you need
Easy to understand how it's done
Encourages readable code
Hard to use incorrectly
No nasty surprises

Provides what you need

- Know your target audience (and their problems)
- Consider all required interactions with your functionality
- No narrow-minded use-cases

Simplifies the mental model
Easy to understand what can be done
Provides what you need
Easy to understand how it's done
Encourages readable code
Hard to use incorrectly
No nasty surprises

Simplifies the mental model
Easy to understand what can be done
Provides what you need
Easy to understand how it’s done
Encourages readable code
Hard to use incorrectly
No nasty surprises

Due to time pressure unfortunately generated by our artificial overlords

How to carry by cultural understanding?

How to carry by cultural understanding?

- **struct**

```
struct config {
  size_t resolution_x, resolution_y;
  pixel_format format;
};
```

How to carry by cultural understanding?

- **struct**

```
struct is_convertible_to_image_asset {};
```

How to carry by cultural understanding?

- **struct**

```
template <typename child_type>
struct corresponding_handler {
  using T = generic_handler<child_type>;
};

template <>
struct corresponding_handler<int> {
...
```

How to carry by cultural understanding?

- **class**

```
class my_object {
      my_object() { ... }
      ~my_object() { ... }
};
```

How to carry by cultural understanding?

- class

```
class my_object {
      my_object(const my_object &) { ... }
      my_object(my_object &&) { ... }
      my_object& operator=(const my_object &) { ... }
      my_object& operator=(my_object &&) { ... }
};
```

How to carry by cultural understanding?

- **class**

```
class my_object : public my_parent_object {
    virtual ...
    override ...
};
```

How to carry by cultural understanding?

- enum

```
enum class some_state {
  state_a,
  state_b,
  state_c,
};
```

How to carry by cultural understanding?

- enum

```
open_file(filename, file_mode::append);
open_file(filename, true);
```

How to carry by cultural understanding?

- enum

```
enum class resolve_status {
  failed_to_resolve,
  entity_found,
  you_are_the_target,
};

resolve_status deliver(...);
```

How to carry by cultural understanding?

- struct
- class
- enum
- public Constructor / static Creation Function / make_X()
- static / non-static (const / non-const) Member- / Function
- const & / const *
- & / *
- &&
- **

How to carry by cultural understanding?

- std::span, std::vector, * + count
- begin() + end()
- std::optional, std::variant, std::expected
- std::unique_ptr, std::shared_ptr
- size / count / bytes / capacity

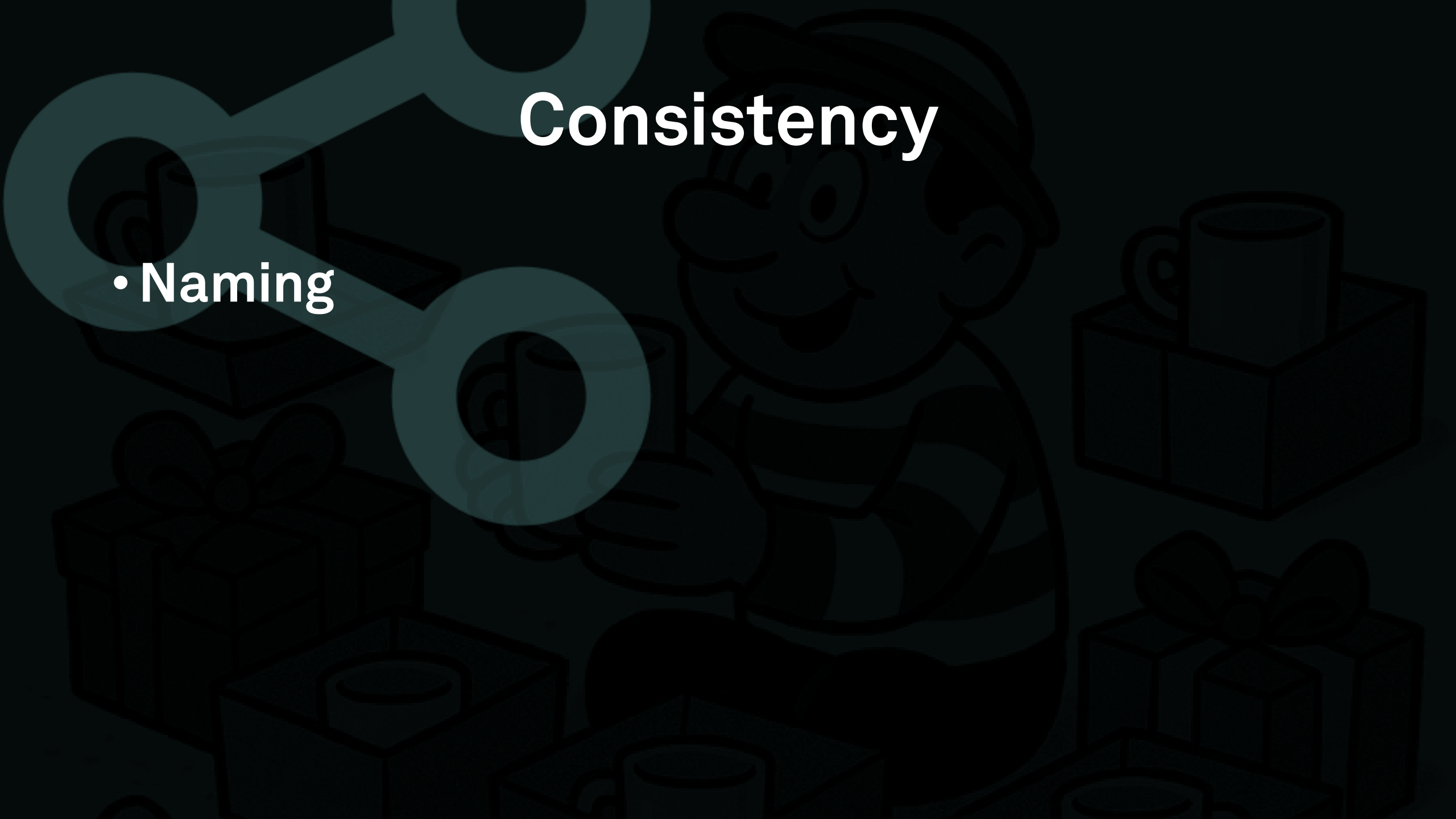
Respect Learned Knowledge

Respect Learned Knowledge

➔ Consistency!

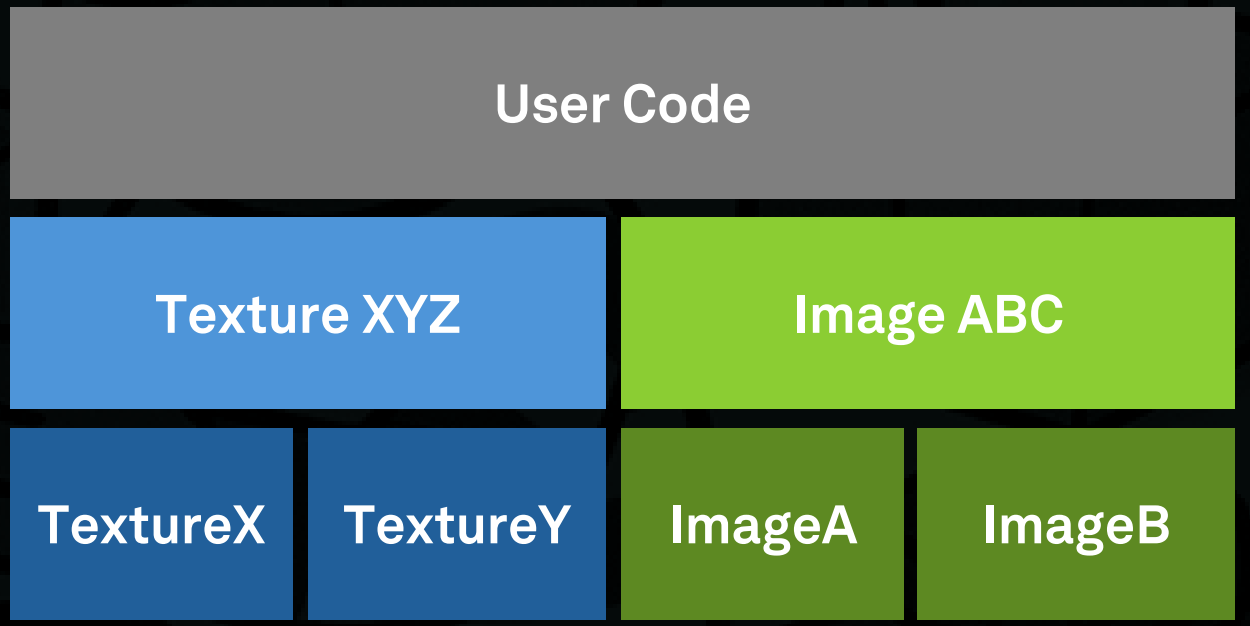
Consistency

- Naming



Consistency

- Naming



Consistency

- Naming
- Approach

Consistency

- Naming
- Approach

```
auto v = ObjectA().resize(2).colorize(red);
```

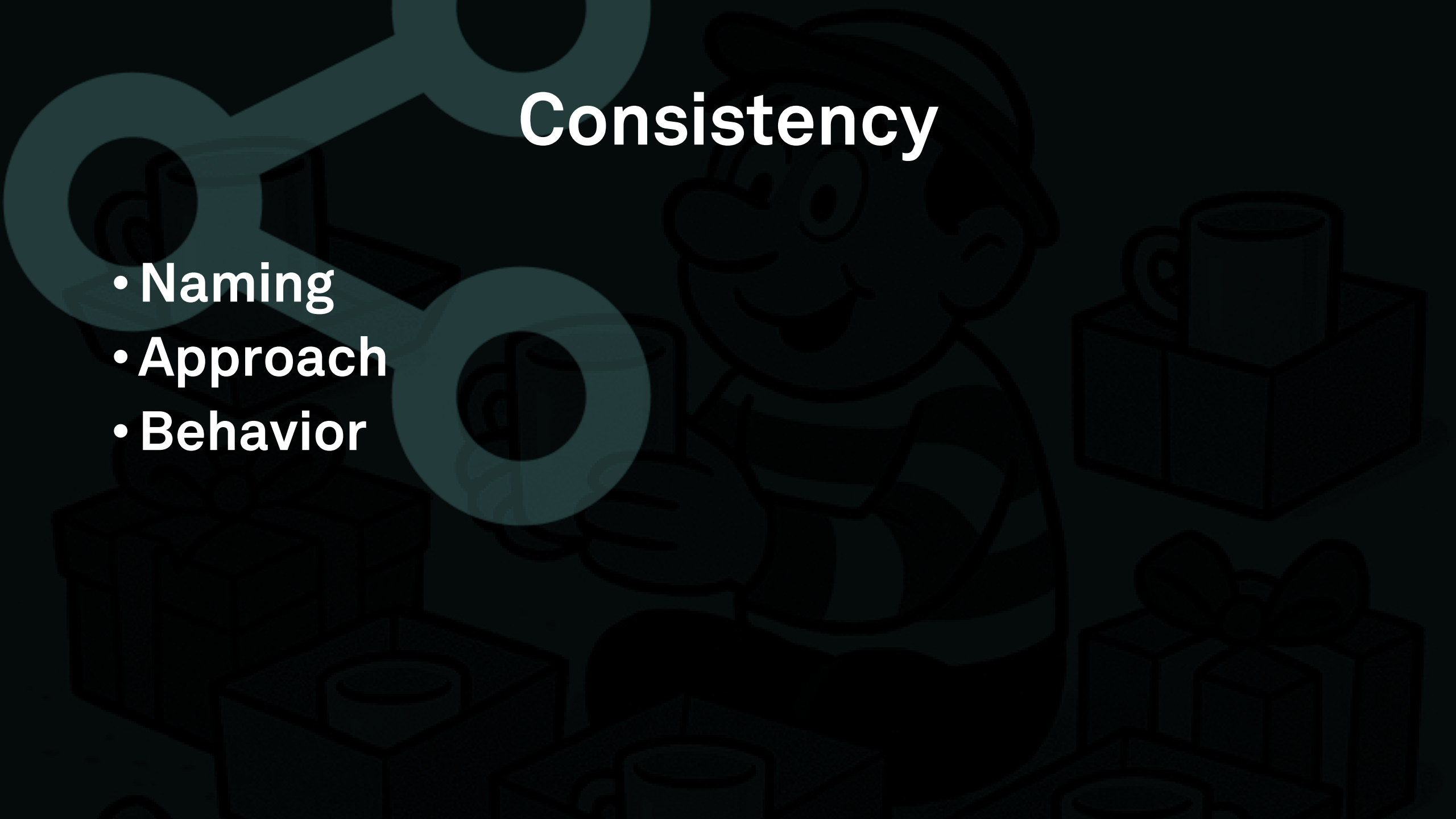
Consistency

- Naming
- Approach

```
auto src = create_source();
auto dst = create_sink();
while (!src.empty())
    dst.push(std::move(src.next()));
```

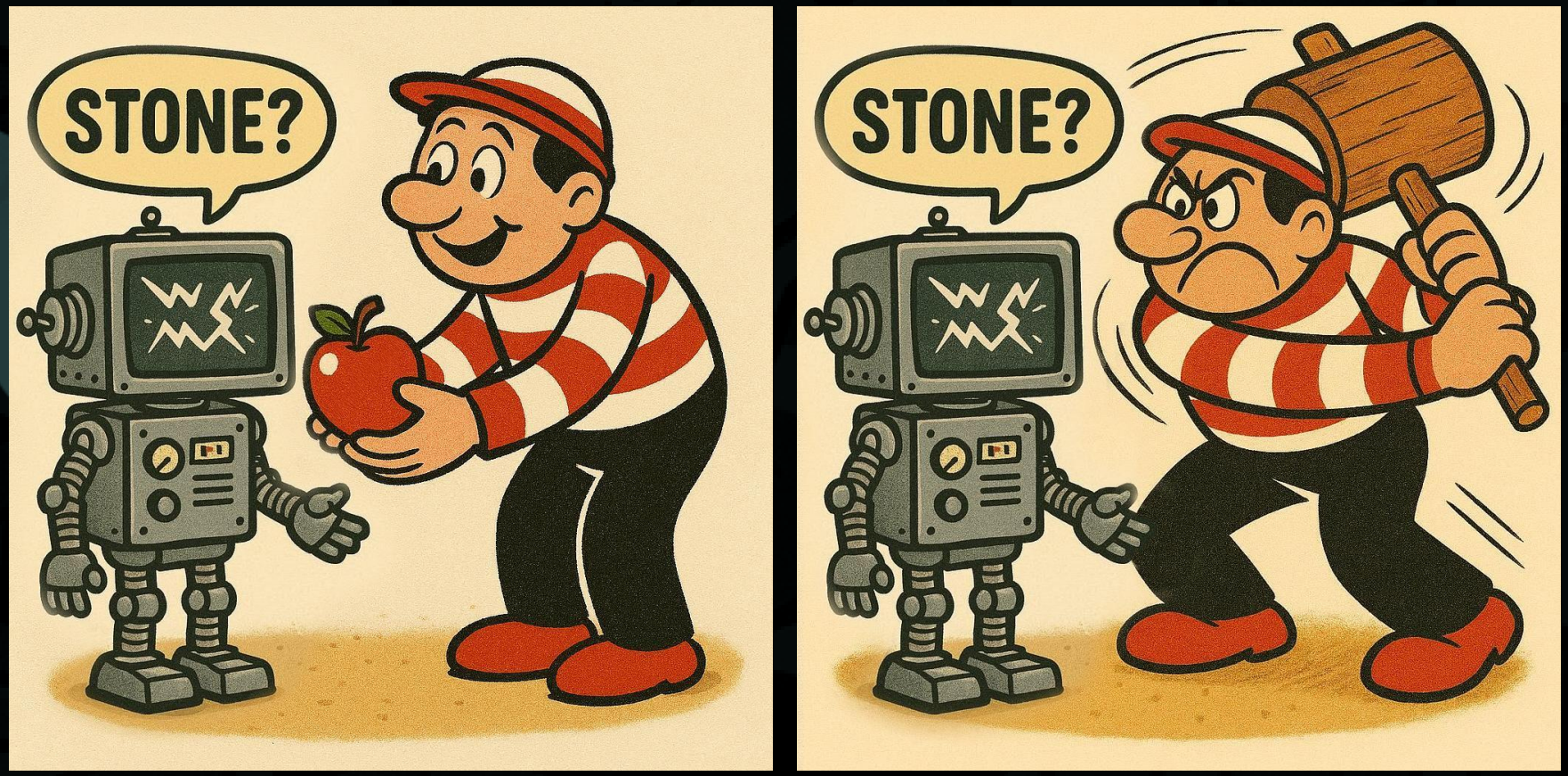
Consistency

- Naming
- Approach
- Behavior



Consistency

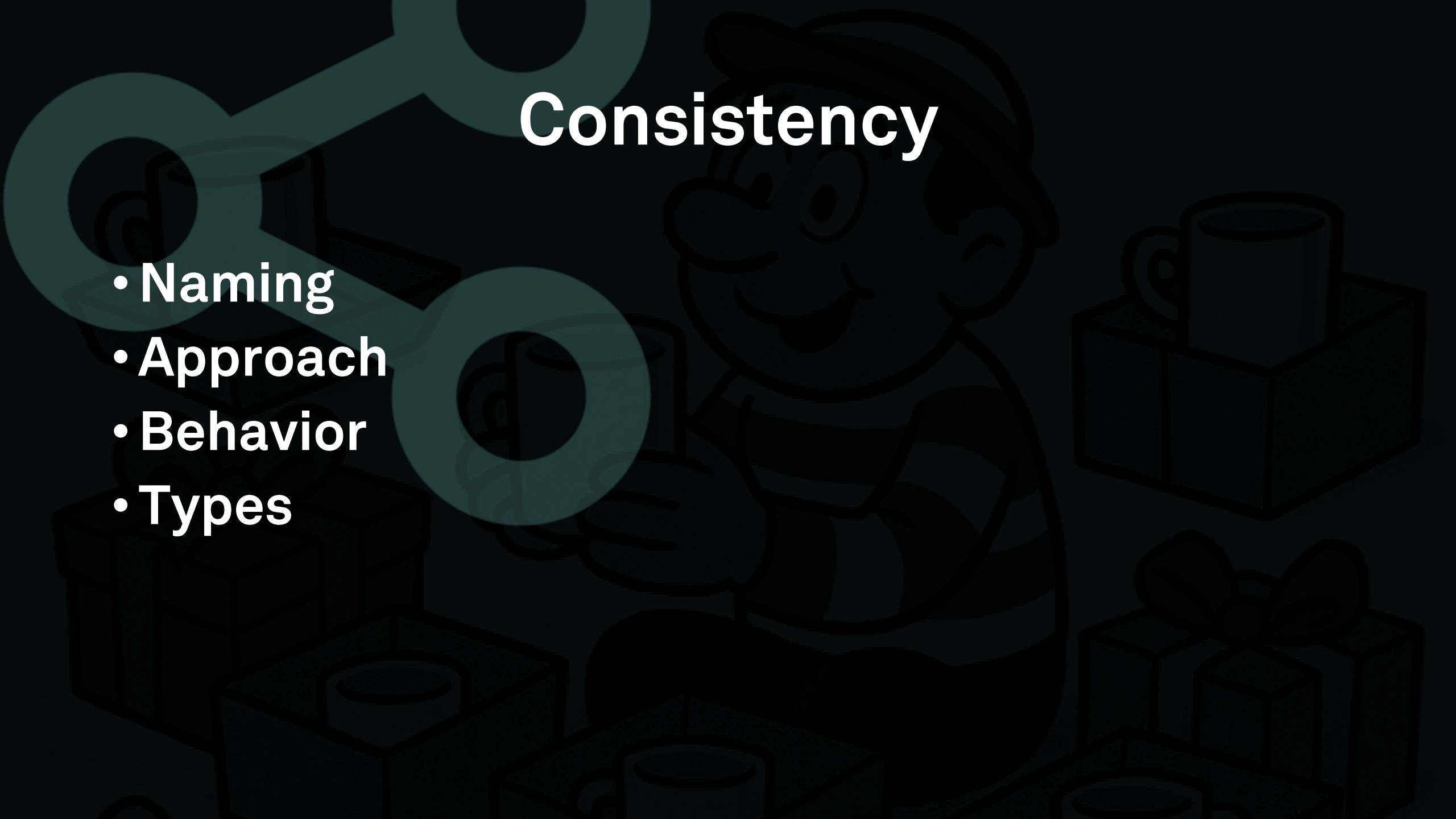
- Naming
- Approach
- Behavior



Due to time pressure unfortunately created aided by our artificial overlords

Consistency

- Naming
- Approach
- Behavior
- Types



Consistency

- Naming
- Approach
- Behavior
- Types

```
struct file_size {
  file_size(const size_t);
  size_t bytes() const;
  size_t bits() const;
}

file_size file::get_size() const;
void file_stream::reserve(const file_size);
```

Easy to understand how it’s done

- Use pre-existing knowledge in your favor
- Consistency (naming, approach, behavior, types)
  - Make people learn something *once*
  - Allow them to stay in your system
- Provide reasonable defaults
- Documentation when you need it

Simplifies the mental model
Easy to understand what can be done
Provides what you need
Easy to understand how it’s done
Encourages readable code
Hard to use incorrectly
No nasty surprises

Simplifies the mental model
Easy to understand what can be done
Provides what you need
Easy to understand how it’s done
Encourages readable code
Hard to use incorrectly
No nasty surprises

Simpler = Better

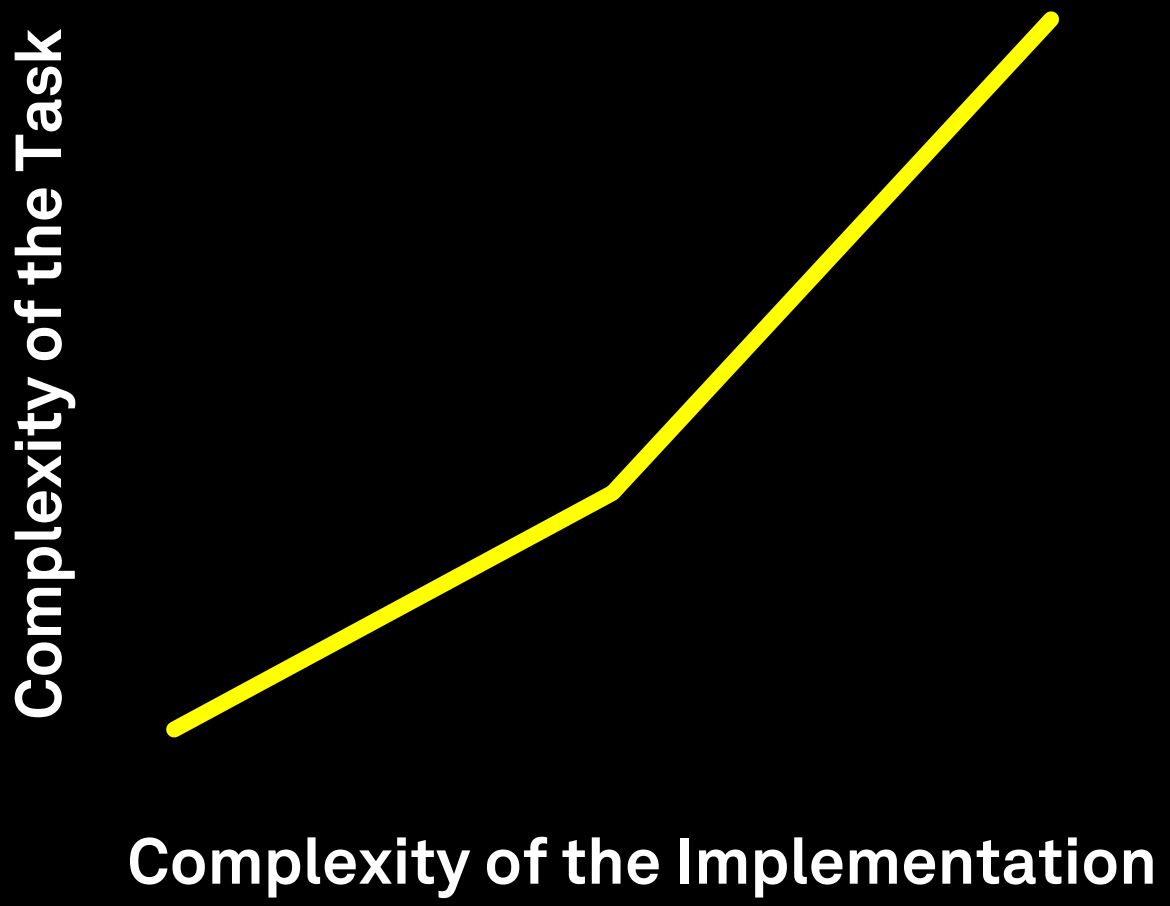
All the affected
code being simpler = Better
(impl + iface + all usage)

- Don’t require special treatment
  - Unless the total gains are off the charts
  - Aid using concept / static_assert / assert

```
template <typename T>
void update_object(T &obj) {
  obj.update(); // 🙄
}
```

- Don’t require special treatment
  - Unless the total gains are off the charts
  - Aid using concept / static_assert / assert
- Inversion of Control is Spaghetti in Disguise
  - `std::function<> / Callable&&`
  - `child.do_something(*this);`

Simple Interactions → easy
Complex Interactions → possible


Complexity of the Task
Complexity of the Implementation

How to build such an API?

Pseudo Code.

- TDD without tests?
- Trust your solution!
- Only hard requirements matter

Simple to

implement

read

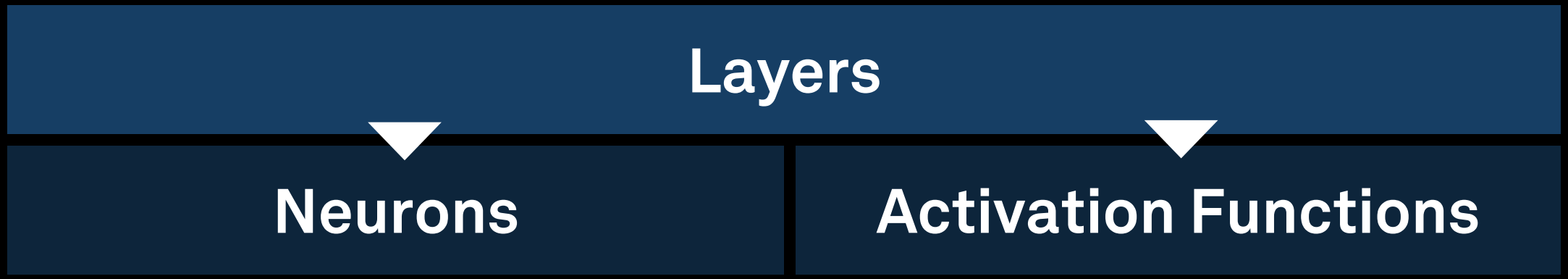
review

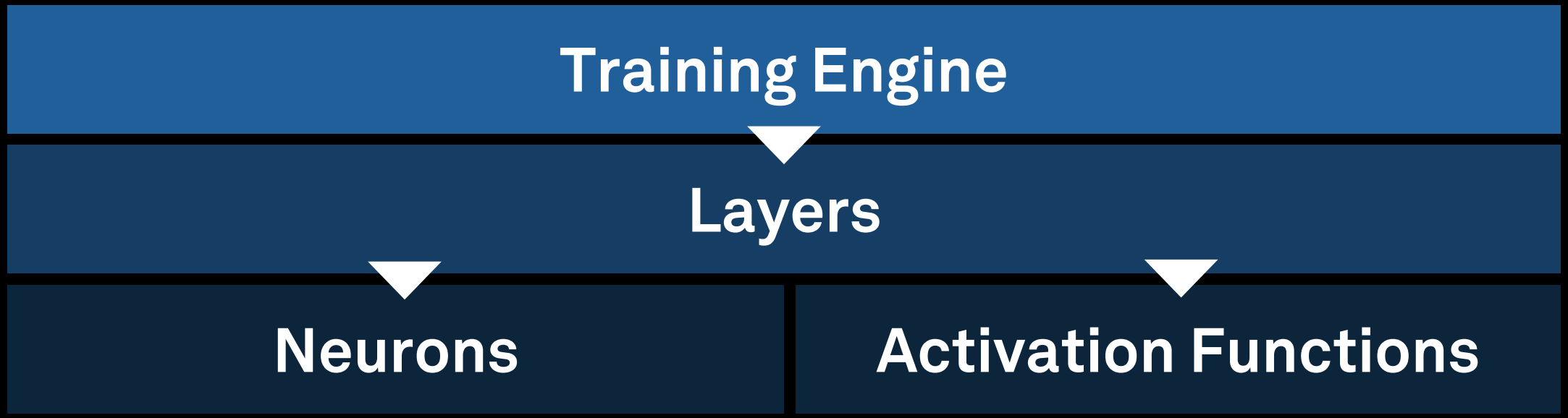
How to design APIs across a new subsystem?

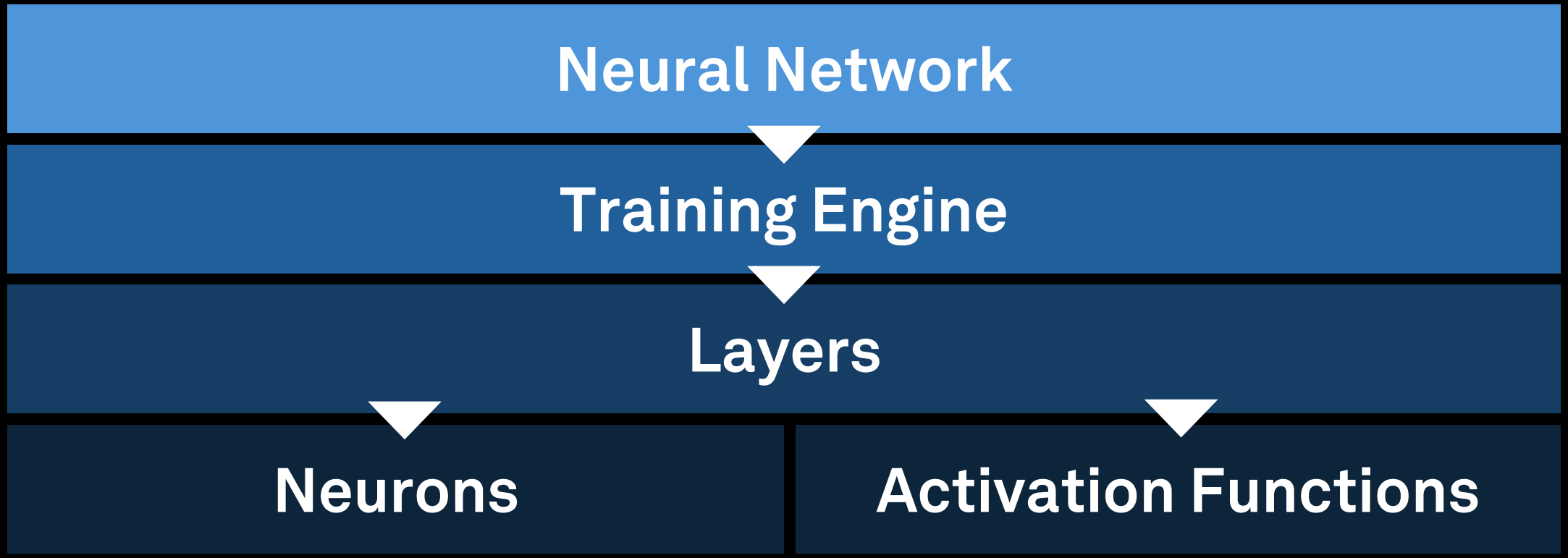
Bottom Up

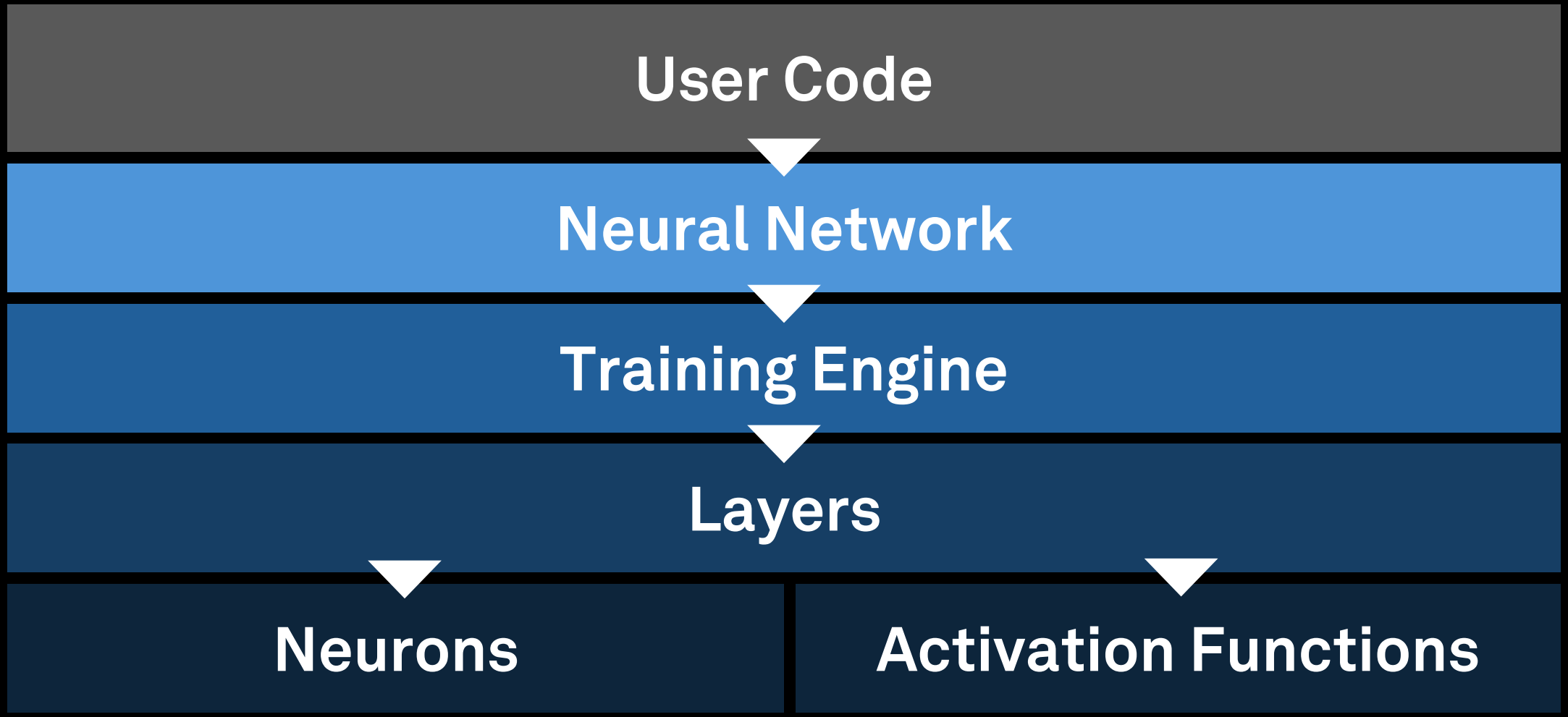
Neurons

Activation Functions


Layers
Neurons
Activation Functions

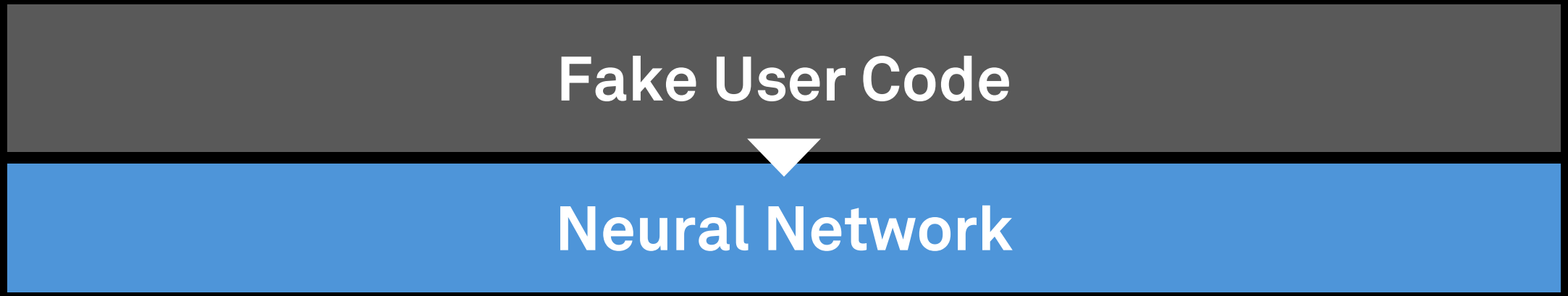

Training Engine
Layers
Neurons
Activation Functions

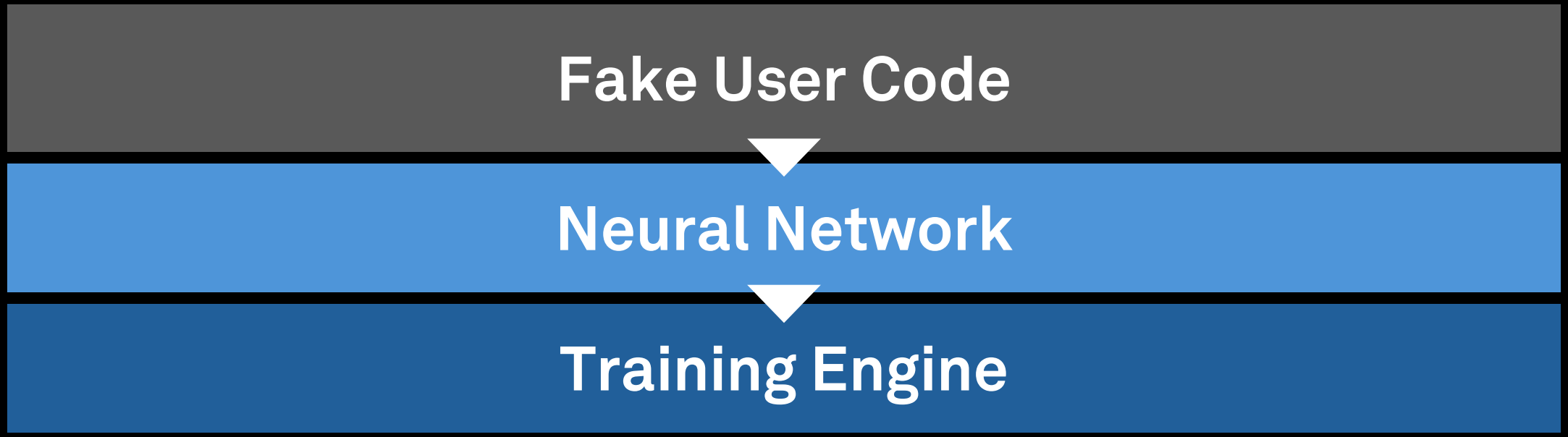

Neural Network
Training Engine
Layers
Neurons
Activation Functions

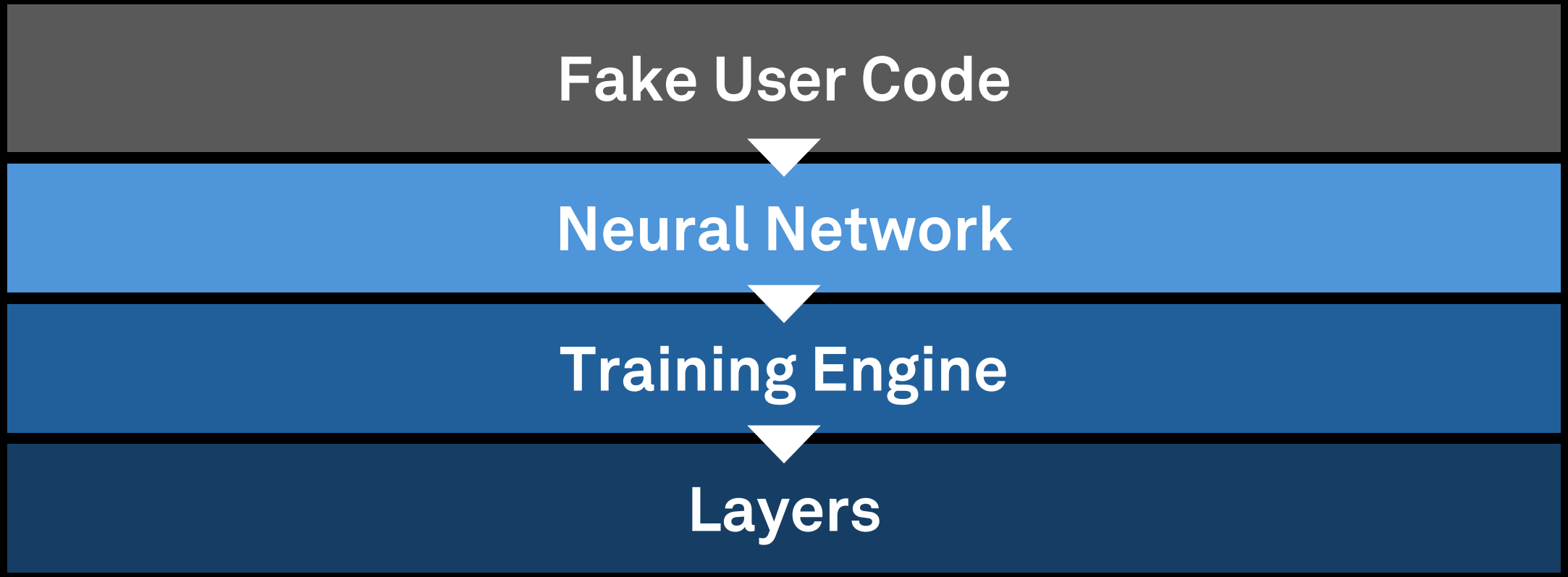

User Code
Neural Network
Training Engine
Layers
Neurons
Activation Functions

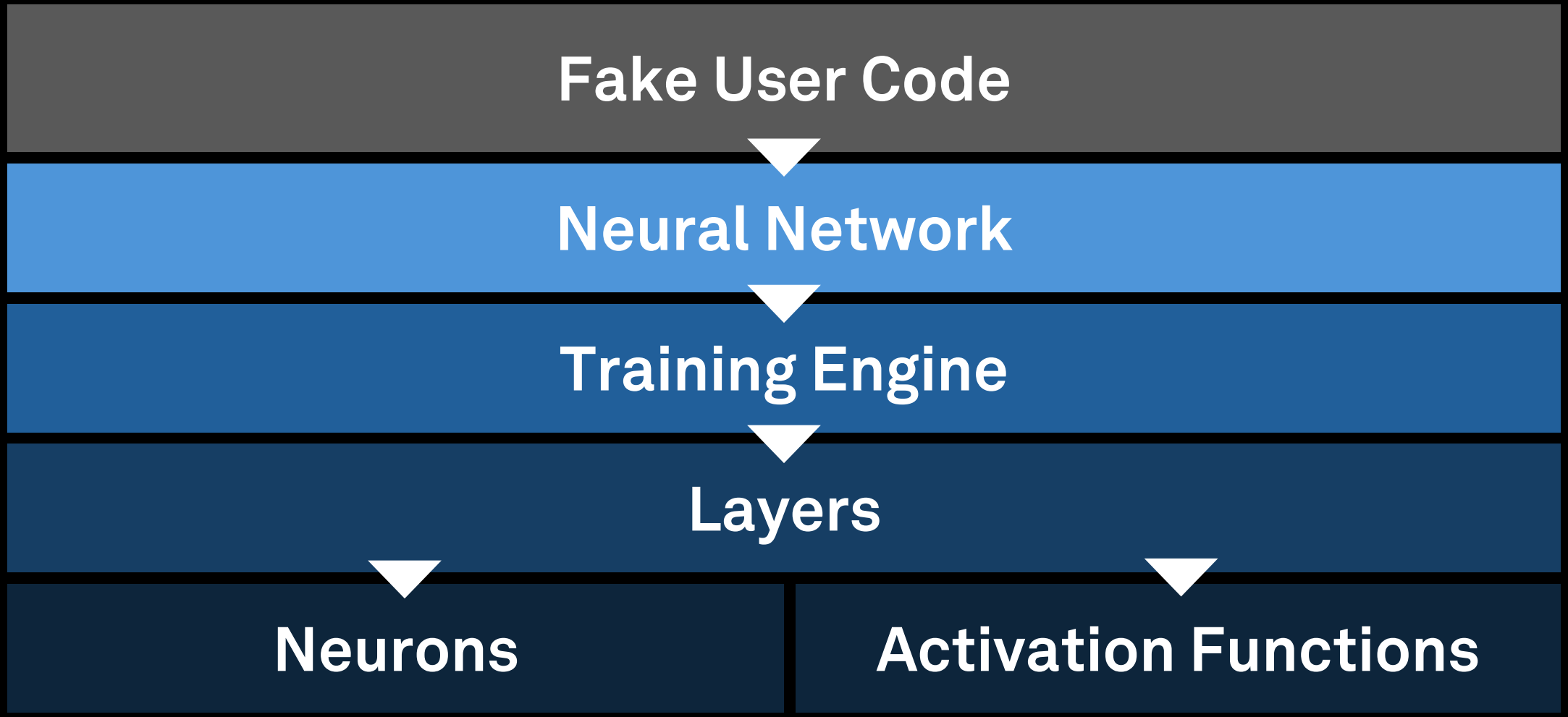
Top Down

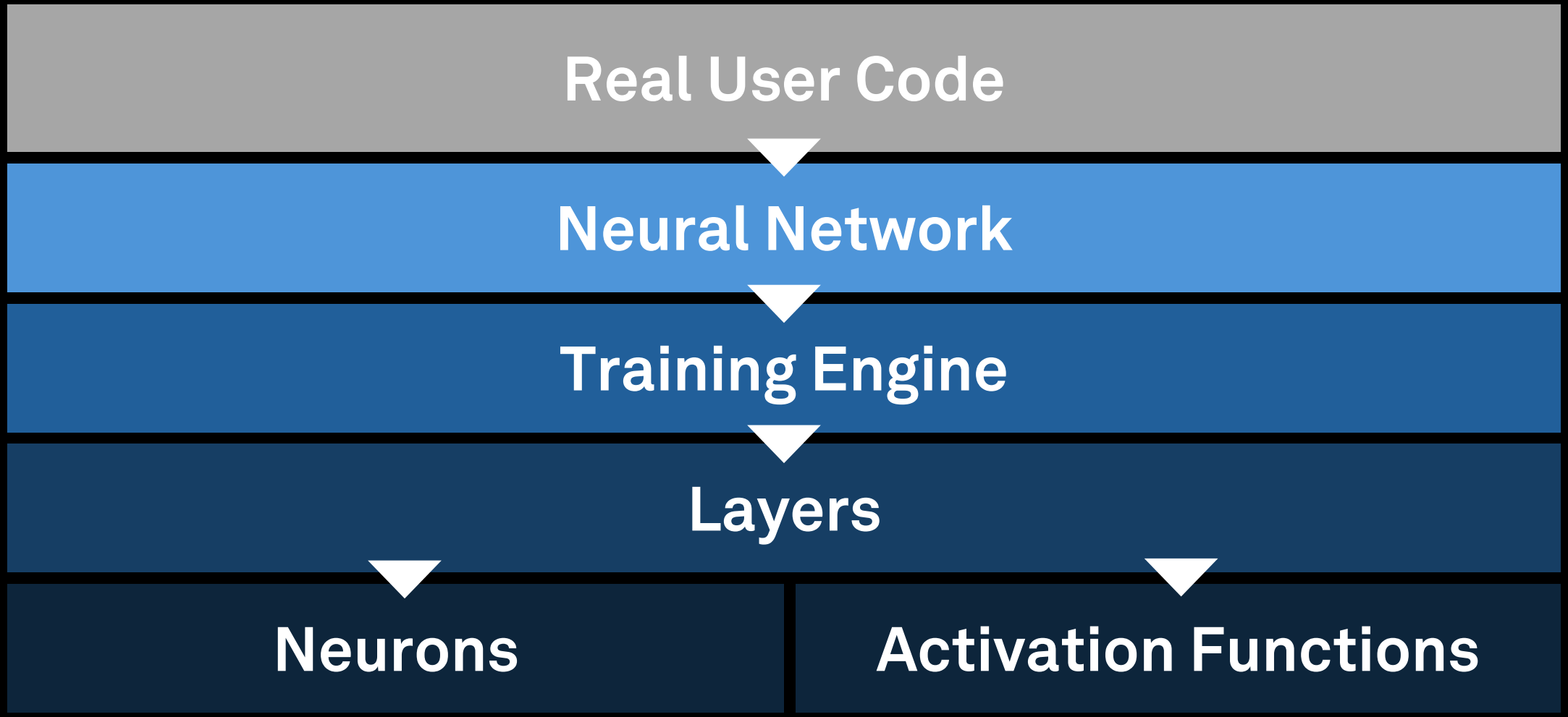
Fake User Code


Fake User Code
Neural Network


Fake User Code
Neural Network
Training Engine


Fake User Code
Neural Network
Training Engine
Layers


Fake User Code
Neural Network
Training Engine
Layers
Neurons
Activation Functions


Real User Code
Neural Network
Training Engine
Layers
Neurons
Activation Functions

Bottom Top Down

Neurons

Activation Functions

Fake User Code

Neurons

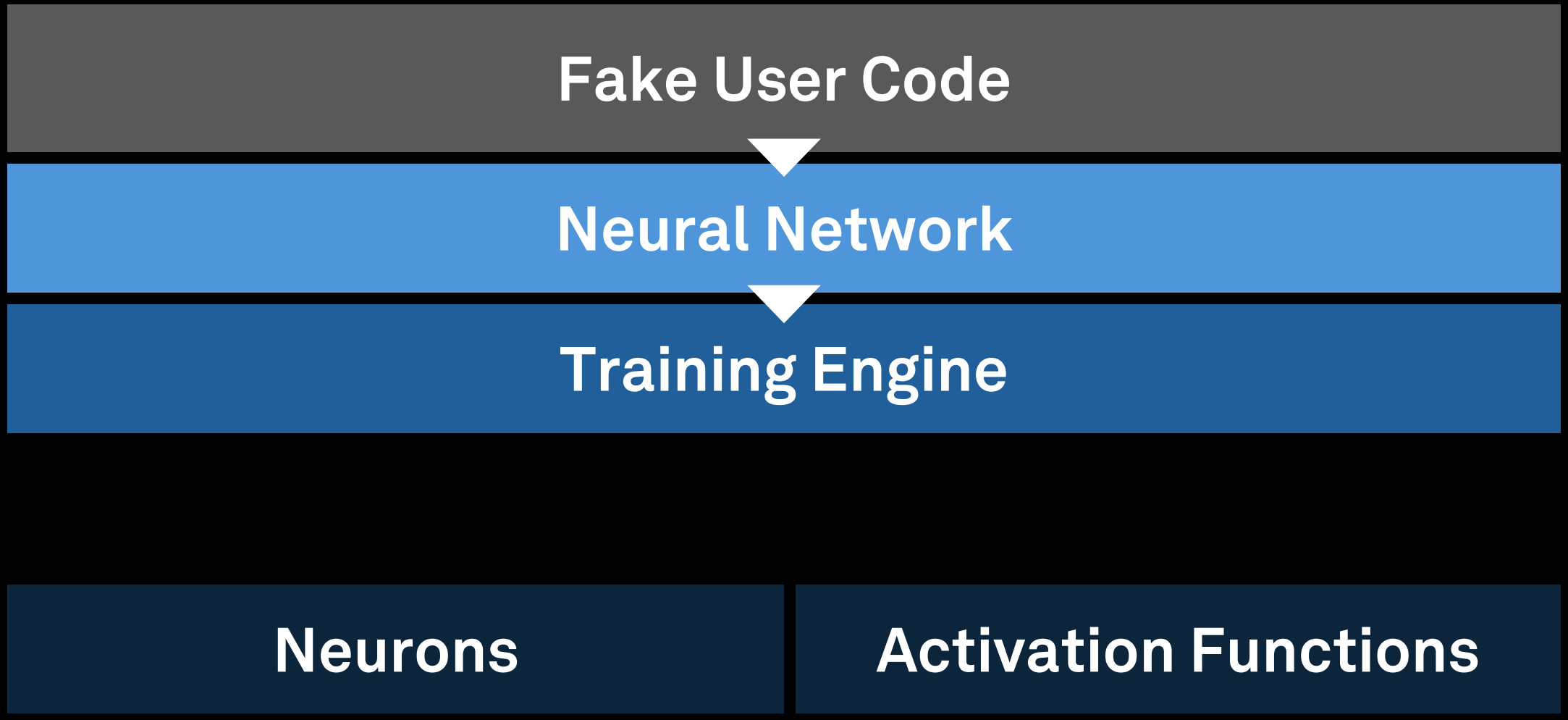
Activation Functions

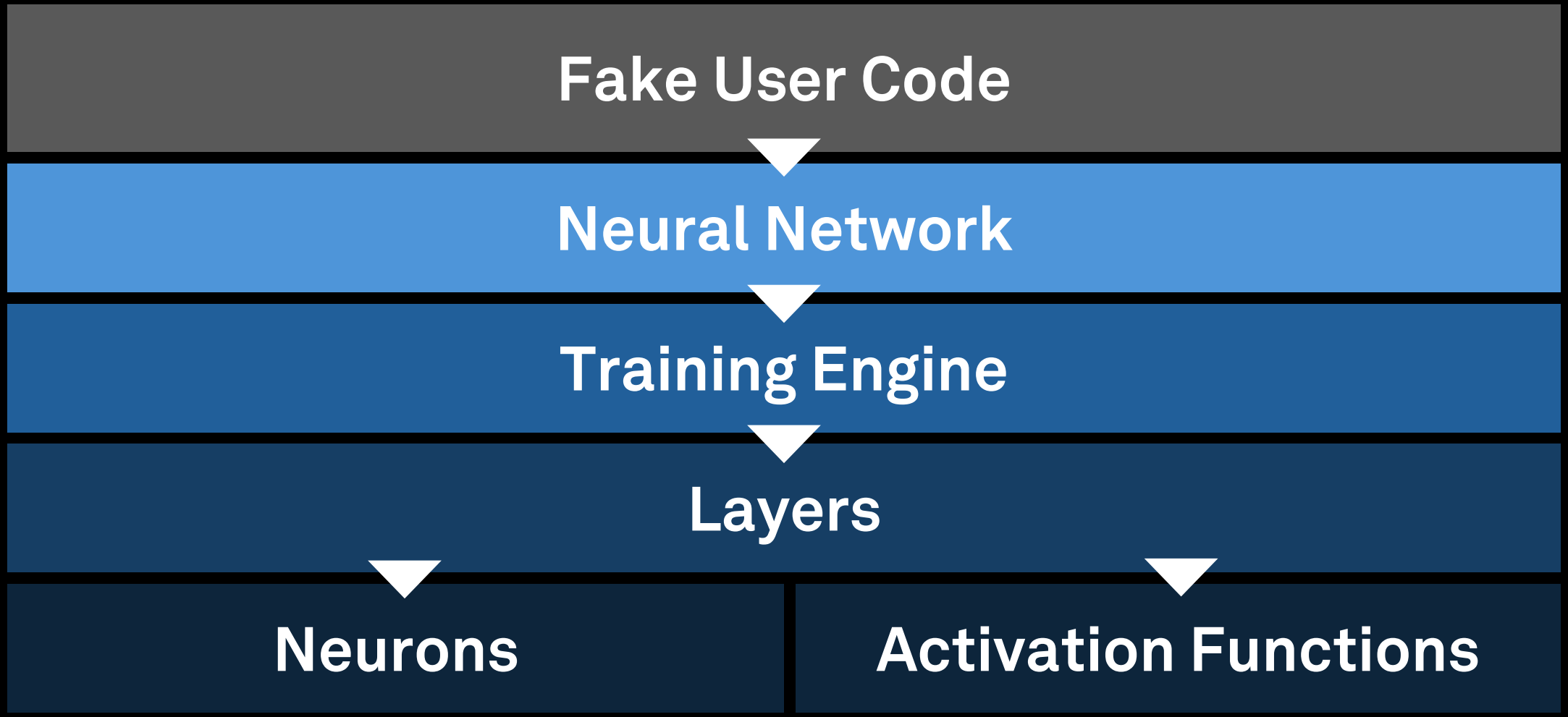
Fake User Code

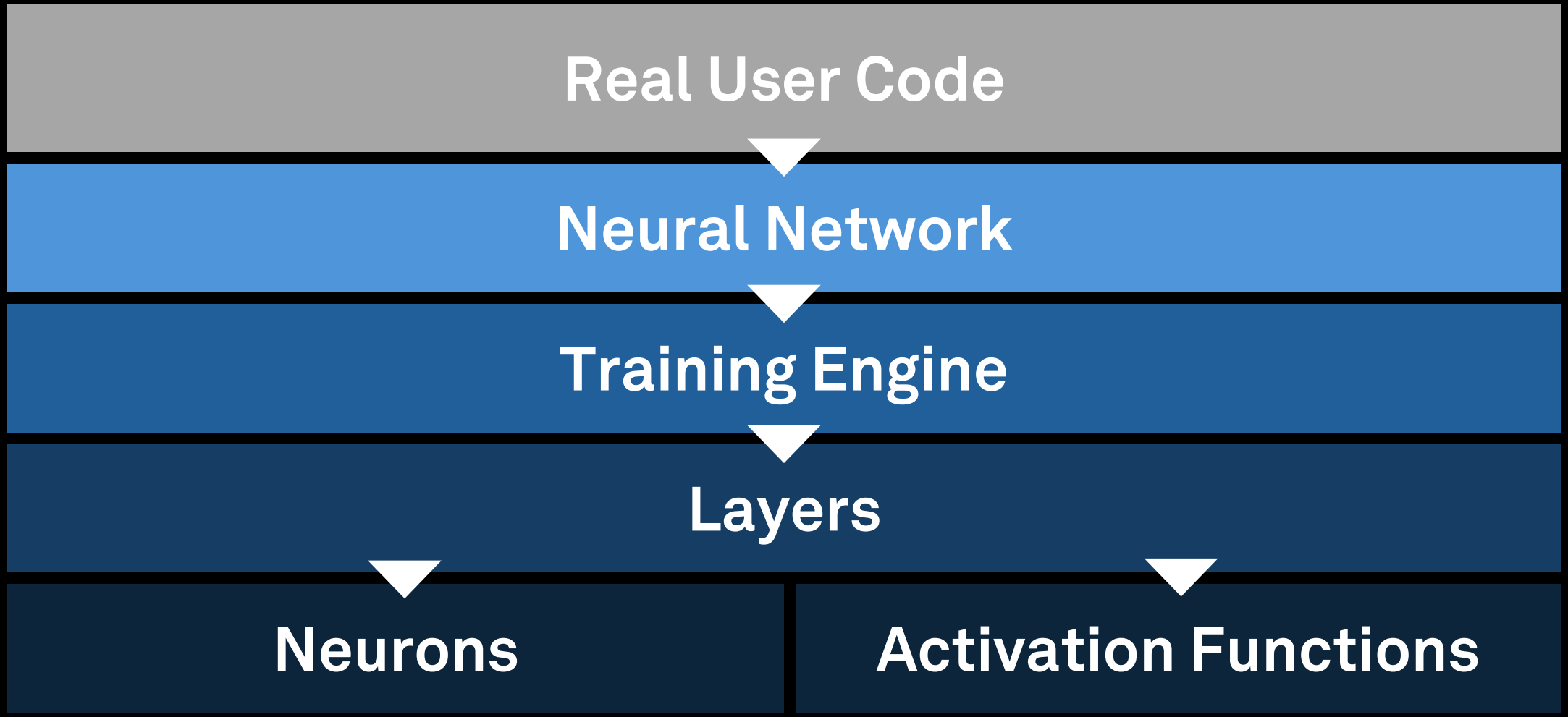
Neural Network

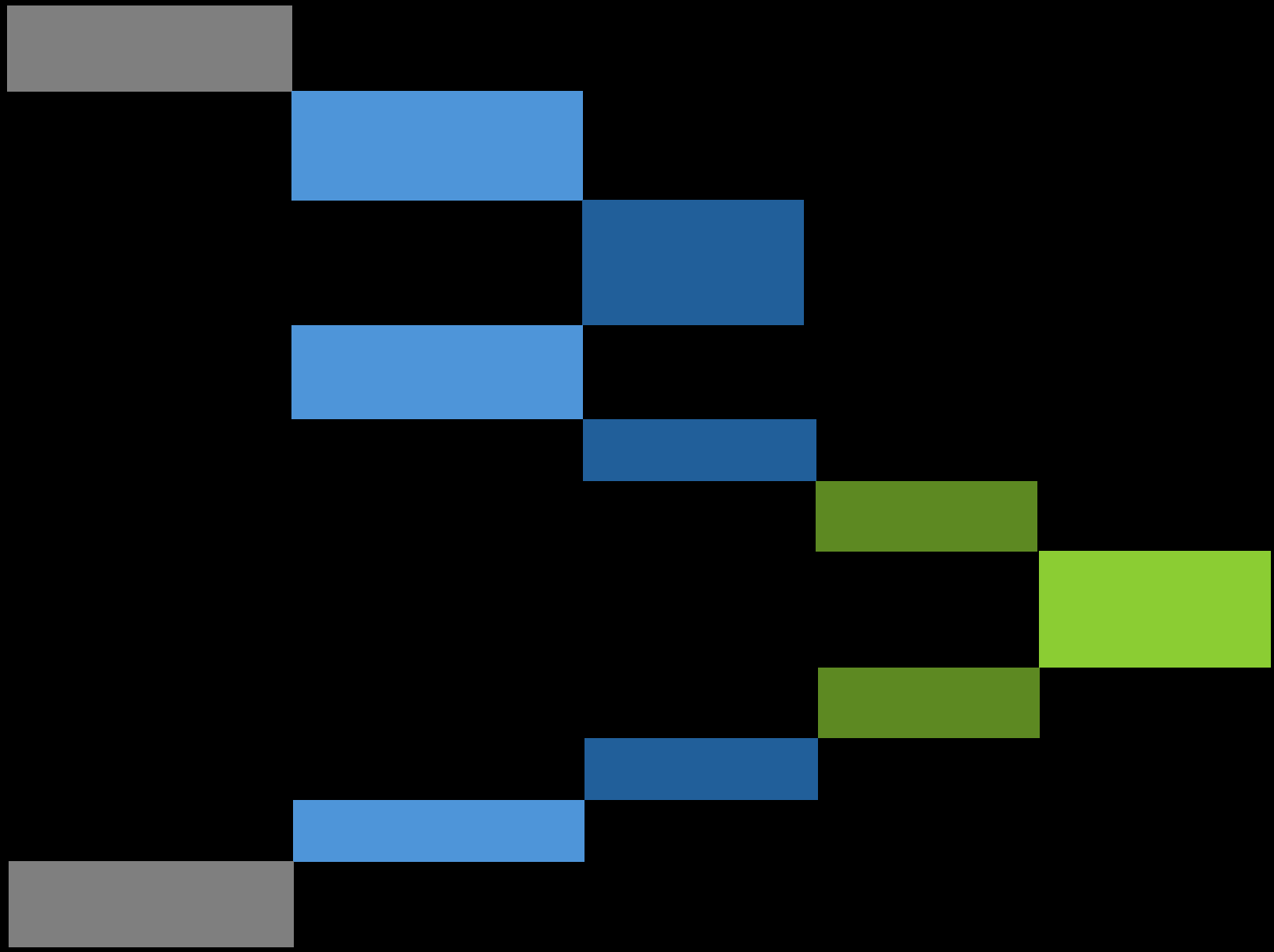
Neurons

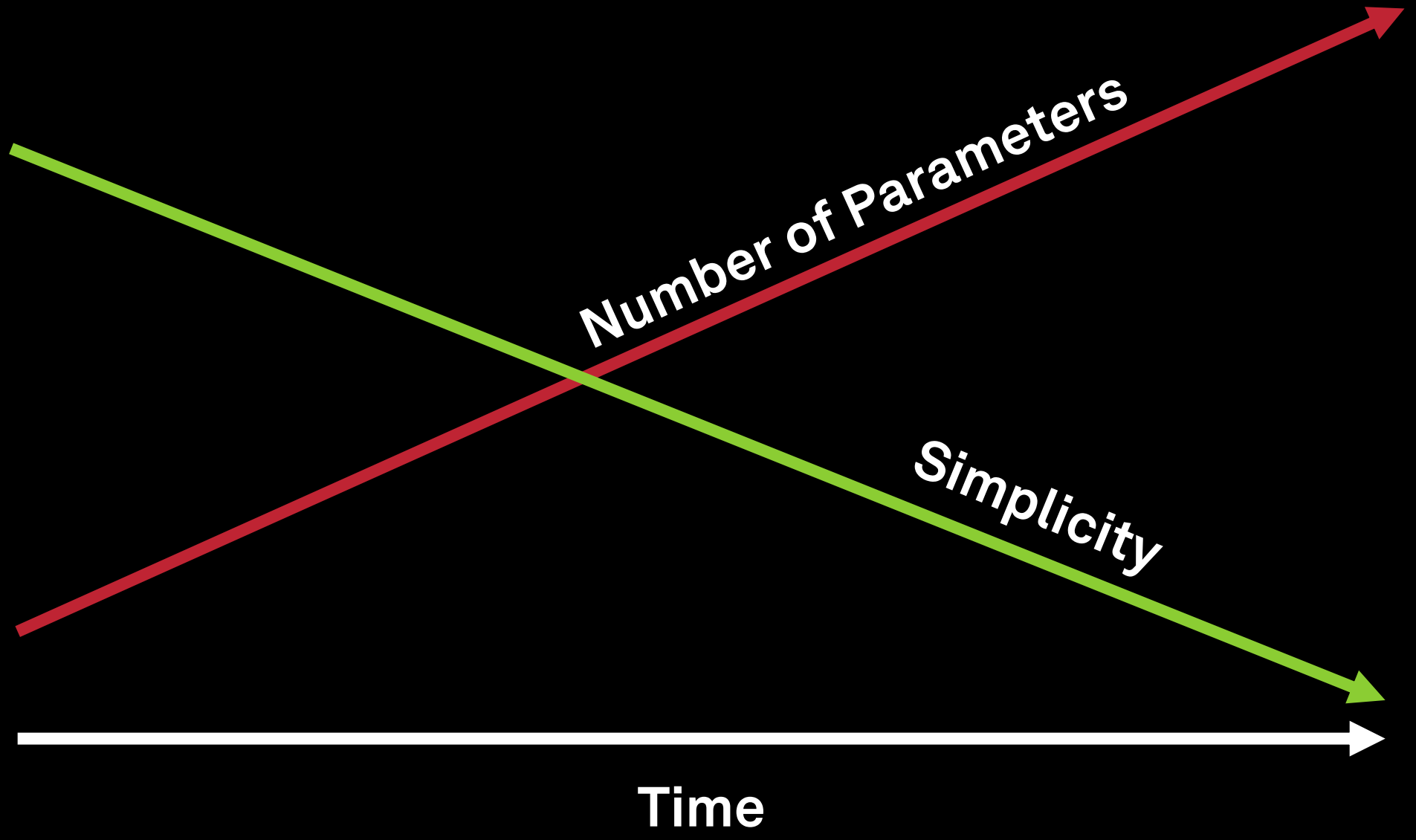
Activation Functions


Fake User Code
Neural Network
Training Engine
Neurons
Activation Functions


Fake User Code
Neural Network
Training Engine
Layers
Neurons
Activation Functions


Real User Code
Neural Network
Training Engine
Layers
Neurons
Activation Functions




Number of Parameters
Simplicity
Time

Encourages readable code

- Entire System simpler = better
- Simple things → Easy
- Complex things → Possible
- Top Down / Bottom Top Down
- Evaluate Interfaces when making changes

Simplifies the mental model
Easy to understand what can be done
Provides what you need
Easy to understand how it’s done
Encourages readable code
Hard to use incorrectly
No nasty surprises

Simplifies the mental model
Easy to understand what can be done
Provides what you need
Easy to understand how it’s done
Encourages readable code
Hard to use incorrectly
No nasty surprises

Hard to use incorrectly

- Unexpected behavior?
  - Can it be removed?
  - Comment
  - Parameter Name
  - Function/Enum Value Name
  - Type with explicit ctor / static construction fn
  - I_DO_UNDERSTAND_THAT_THIS_IS_DANGEROUS
- Encourage proper use of abstraction layers
- Make internal helpers inaccessible
- Check the validity of parameters & configuration!

Simplifies the mental model
Easy to understand what can be done
Provides what you need
Easy to understand how it’s done
Encourages readable code
Hard to use incorrectly
No nasty surprises

Simplifies the mental model
Easy to understand what can be done
Provides what you need
Easy to understand how it's done
Encourages readable code
Hard to use incorrectly
No nasty surprises

No nasty surprises

- Know your error handling approach
- Know your memory allocation (failure) approach
- Handle / Detect Edge-Cases
- Don’t limit valid behavior
- Don’t make false promises
- Compatible layers of abstraction
- Overuse of Callbacks

➔ Learn from the mistakes of others

Simplifies the mental model
Easy to understand what can be done
Provides what you need
Easy to understand how it’s done
Encourages readable code
Hard to use incorrectly
No nasty surprises

Evaluating APIs

- Mental Model
- Your Interfaces
- Client Code

Evaluating APIs: Mental Model

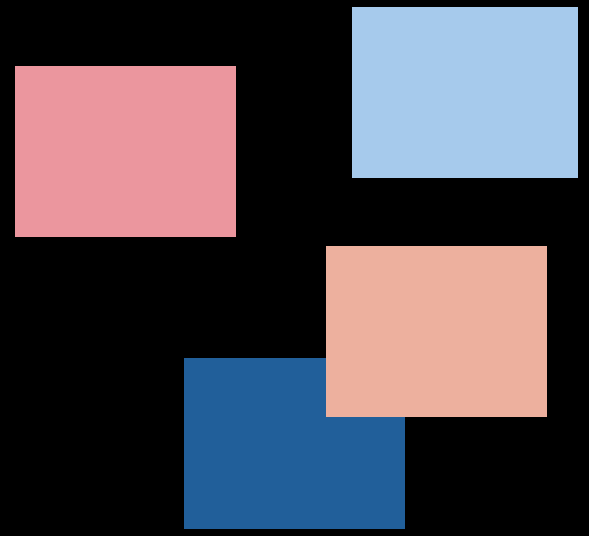
- Where do abstraction layers start & end?

Evaluating APIs: Mental Model

- Where do abstraction layers start & end?

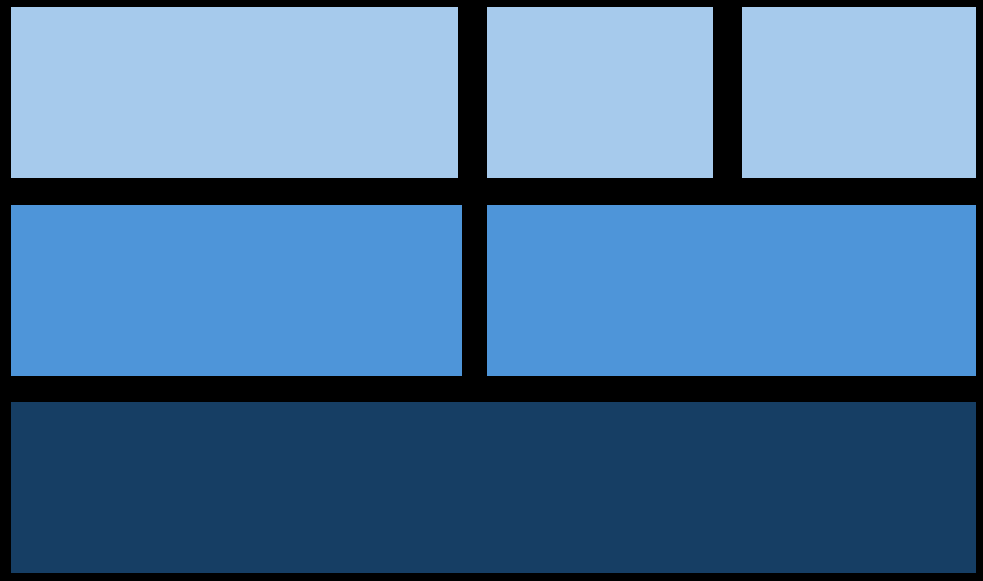
Evaluating APIs: Mental Model

- Where do abstraction layers start & end?
- What's the purpose of them?



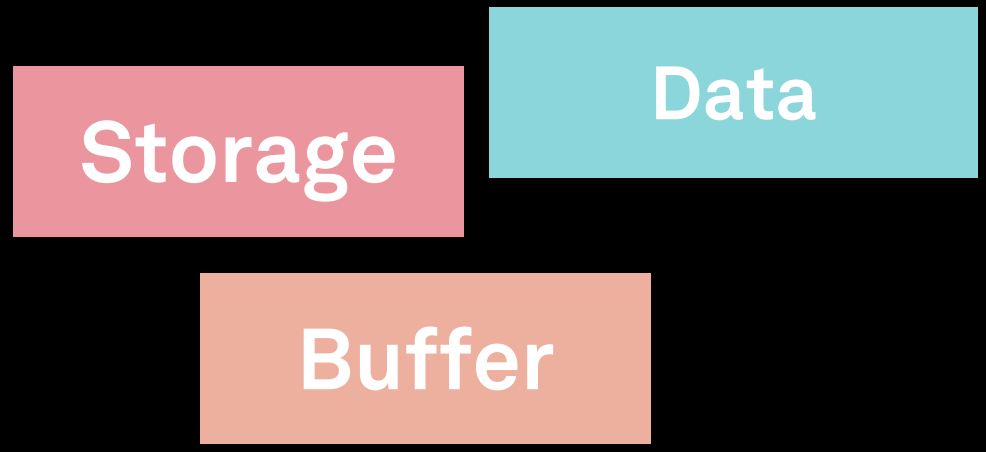
Evaluating APIs: Mental Model

- Where do abstraction layers start & end?
- What's the purpose of them?



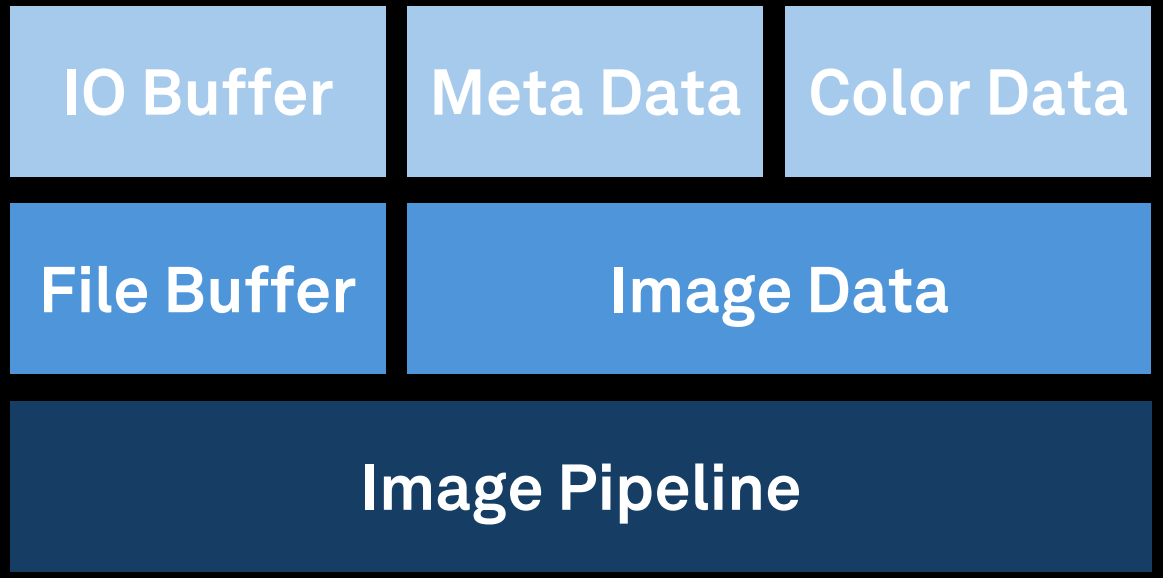
Evaluating APIs: Mental Model

- Where do abstraction layers start & end?
- What's the purpose of them?
- Hard to track the flow?



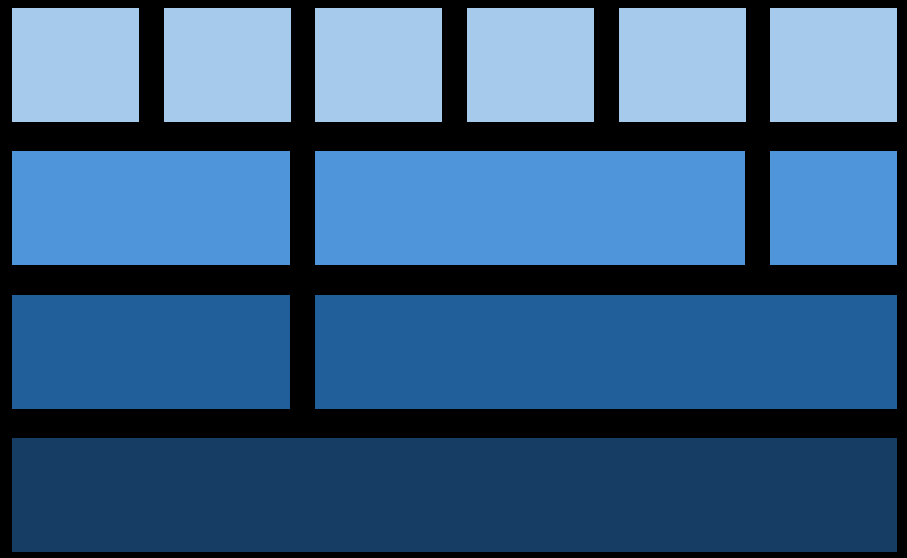
Evaluating APIs: Mental Model

- Where do abstraction layers start & end?
- What's the purpose of them?
- Hard to track the flow?



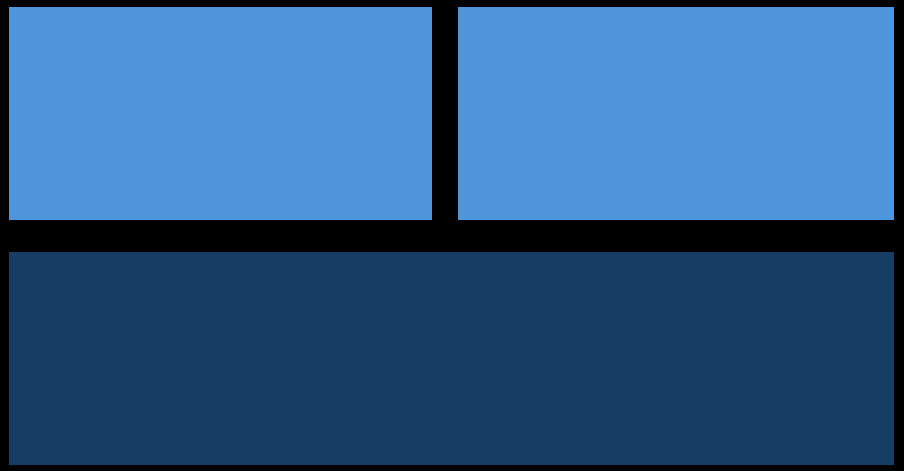
Evaluating APIs: Mental Model

- Where do abstraction layers start & end?
- What's the purpose of them?
- Hard to track the flow?
- Many similar things?



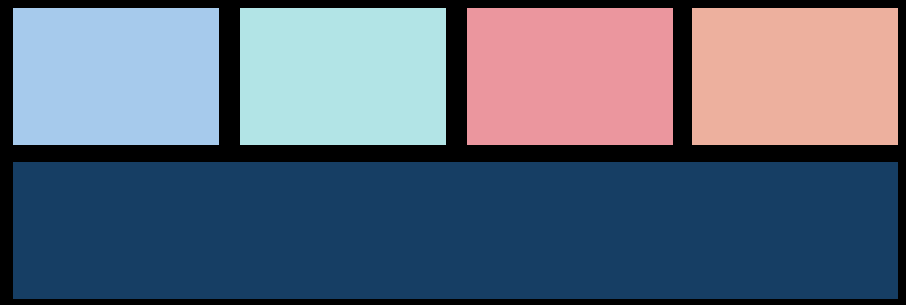
Evaluating APIs: Mental Model

- Where do abstraction layers start & end?
- What's the purpose of them?
- Hard to track the flow?
- Many similar things?



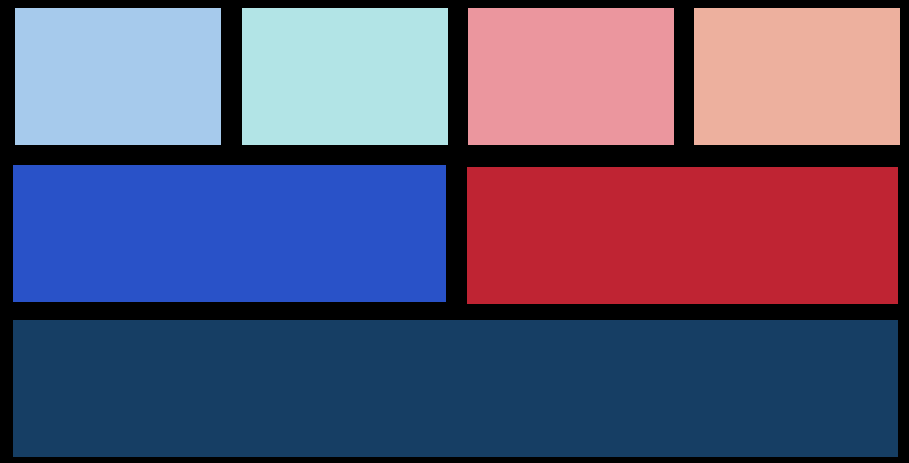
Evaluating APIs: Mental Model

- Where do abstraction layers start & end?
- What's the purpose of them?
- Hard to track the flow?
- Many similar things?
- Objects with complex internal state?



Evaluating APIs: Mental Model

- Where do abstraction layers start & end?
- What's the purpose of them?
- Hard to track the flow?
- Many similar things?
- Objects with complex internal state?



Evaluating APIs: Mental Model

- Where do abstraction layers start & end?
- What's the purpose of them?
- Hard to track the flow?
- Many similar things?
- Objects with complex internal state?

```
allocate_image();
update_image();
destroy_image();
```

Evaluating APIs: Mental Model

- Where do abstraction layers start & end?
- What's the purpose of them?
- Hard to track the flow?
- Many similar things?
- Objects with complex internal state?

```
class image {
  void update();
};
```

Evaluating APIs: Your Code

- Who is your target audience?

```
std::optional<location> find();
bool find(location &);
```

Evaluating APIs: Your Code

- **Who is your target audience?**

```
CreateSession();
CreateMediaSource();
CreatePresentationDescription()
CreatePlaybackTopology();
SetTopology();
```

Evaluating APIs: Your Code

- **Who is your target audience?**

```
MessageBeep(MB_ICONERROR);
```

Evaluating APIs: Your Code

- Who is your target audience?
- How clear are directory structure & file names?

```
stuff/
engine/
data.h
utils.h
processing.h
```

Evaluating APIs: Your Code

- Who is your target audience?
- How clear are directory structure & file names?

```
video/
audio/
playback_engines.h
subtitles.h
```

Evaluating APIs: Your Code

- Who is your target audience?
- How clear are directory structure & file names?
- How easy are your headers to navigate?

```
57091   public:
57092     VULKAN_HPP_NAMESPACE::ArrayWrapper1D<char, VK_MAX_EXTENSION_NAME_SIZE> layerName             = {};
57093     uint32_t                                                               specVersion           = {};
57094     uint32_t                                                               implementationVersion = {};
57095     VULKAN_HPP_NAMESPACE::ArrayWrapper1D<char, VK_MAX_DESCRIPTION_SIZE>    description           = {};
57096   };
57097
57098   // wrapper struct for struct VkLayerSettingEXT, see https://registry.khronos.org/vulkan/specs/latest/man/html/VkLayerSettingEXT.html
57099   struct LayerSettingEXT
57100   {
57101     using NativeType = VkLayerSettingEXT;
57102
57103 #if !defined( VULKAN_HPP_NO_CONSTRUCTORS ) && !defined( VULKAN_HPP_NO_STRUCT_CONSTRUCTORS )
57104     VULKAN_HPP_CONSTEXPR LayerSettingEXT( const char *                              pLayerName_   = {},
57105                                           const char *                              pSettingName_ = {},
57106                                           VULKAN_HPP_NAMESPACE::LayerSettingTypeEXT type_         =
                                              VULKAN_HPP_NAMESPACE::LayerSettingTypeEXT::eBool32,
57107                                           uint32_t                                  valueCount_   = {},
57108                                           const void *                              pValues_      = {} ) VULKAN_HPP_NOEXCEPT
57109       : pLayerName{ pLayerName_ }
57110       , pSettingName{ pSettingName_ }
57111       , type{ type_ }
57112       , valueCount{ valueCount_ }
57113       , pValues{ pValues_ }
57114     {
57115     }
57116
57117     VULKAN_HPP_CONSTEXPR LayerSettingEXT( LayerSettingEXT const & rhs ) VULKAN_HPP_NOEXCEPT = default;
57118
57119     LayerSettingEXT( VkLayerSettingEXT const & rhs ) VULKAN_HPP_NOEXCEPT : LayerSettingEXT( *reinterpret_cast<LayerSettingEXT const *>( &
        rhs ) ) {}
57120
57121 #  if !defined( VULKAN_HPP_DISABLE_ENHANCED_MODE )
57122     // NOTE: you need to provide the type because vk::Bool32 and uint32_t are indistinguishable!
57123     LayerSettingEXT( char const *                                  pLayerName_,
57124                      char const *                                  pSettingName_,
57125                      VULKAN_HPP_NAMESPACE::LayerSettingTypeEXT     type_,
57126                      vk::ArrayProxyNoTemporaries<const int32_t> const & values_ )
57127       : pLayerName( pLayerName_ )
57128       , pSettingName( pSettingName_ )
57129       , type( type_ )
57130       , valueCount( static_cast<uint32_t>( values_.size() ) )
57131       , pValues( values_.data() )
57132     {
57133       VULKAN_HPP_ASSERT( VULKAN_HPP_NAMESPACE::isSameType<int32_t>( type ) );
57134     }
57135
57136     LayerSettingEXT( char const *                                  pLayerName_,
57137                      char const *                                  pSettingName_,
57138                      VULKAN_HPP_NAMESPACE::LayerSettingTypeEXT     type_,
57139                      vk::ArrayProxyNoTemporaries<const int64_t> const & values_ )
57140       : pLayerName( pLayerName_ )
57141       , pSettingName( pSettingName_ )
57142       , type( type_ )
57143       , valueCount( static_cast<uint32_t>( values_.size() ) )
57144       , pValues( values_.data() )
57145     {
57146       VULKAN_HPP_ASSERT( VULKAN_HPP_NAMESPACE::isSameType<int64_t>( type ) );
57147     }
57148
57149     LayerSettingEXT( char const *                                   pLayerName_,
```

```
////////////////////////////////////////////////////////////////////////

// Video Processor
// Converts videos from one format to another blablabal
// Documentation can be found here: link://link.link/link
class video_processor
{
  video_processor(source &, drain &);
  ~video_processor();

  // Will spin up something to show something.
  bool is_ready() const;
  void wait_till_ready() const; // synchronous variant of `is_ready()`

  // Effects can be added using alalalala.
  void add_effect(effect &&);

  // Reads the stream to end.
  // Returns the number of frames if something happened.
  size_t read_to_end() noexcept;
};

// video decoder with gpu acceleration
// the following decoders are available: intel, amd, nvidia
// documentation can be found here: link://link.link/link
class gpu_accelerated_video_decoder : public video_source
{
  gpu_accelerated_video_decoder();

  std::optional<frame> next_frame();
};

////////////////////////////////////////////////////////////////////////
// Audio Processing
////////////////////////////////////////////////////////////////////////

// Audio Processor
// Converts audio stuff from one format to another blablabal
// Supports: mono, stereo
// Documentation can be found here: link://link.link/link
class audio_processor
{
  audio_processor(source &, drain &);
  ~audio_processor();

  // Will spin up something to show something.
  bool is_ready() const;
  void wait_till_ready() const; // synchronous variant of `is_ready()`

  // Effects can be added using alalalala.
  void add_effect(effect &&);

  // Reads the stream to end.
  // Returns the number of samples if something happened.
  size_t read_to_end() noexcept;
};

// Ogg file audio source reader
class ogg_audio_source
{
```

Evaluating APIs: Your Code

- Who is your target audience?
- How clear are directory structure & file names?
- How easy are your headers to navigate?
- Are things named by what they *represent/do*?

`backend_holder_base`

Evaluating APIs: Your Code

- Who is your target audience?
- How clear are directory structure & file names?
- How easy are your headers to navigate?
- Are things named by what they *represent/do*?

```
notification_recipient
```

Evaluating APIs: Your Code

- Who is your target audience?
- How clear are directory structure & file names?
- How easy are your headers to navigate?
- Are things named by what they *represent/do*?
- Would other common constructs simplify your API?

```
void send_email(const char *, int, int);
```

Evaluating APIs: Your Code

- Who is your target audience?
- How clear are directory structure & file names?
- How easy are your headers to navigate?
- Are things named by what they *represent/do*?
- Would other common constructs simplify your API?

```
void send_email(std::string_view, priority)
```

Evaluating APIs: Your Code

- Who is your target audience?
- How clear are directory structure & file names?
- How easy are your headers to navigate?
- Are things named by what they *represent/do*?
- Would other common constructs simplify your API?
- Consistent behavior/types/naming?

```
file_data read_file();
void write_file(std::vector<u8>);
```

Evaluating APIs: Your Code

- Who is your target audience?
- How clear are directory structure & file names?
- How easy are your headers to navigate?
- Are things named by what they *represent/do*?
- Would other common constructs simplify your API?
- Consistent behavior/types/naming?

```
file_data read_file();
void write_file(file_data);
```

Evaluating APIs: Your Code

- Who is your target audience?
- How clear are directory structure & file names?
- How easy are your headers to navigate?
- Are things named by what they *represent/do*?
- Would other common constructs simplify your API?
- Consistent behavior/types/naming?

```
class file_reader_data;
class io_buffer;
```

Evaluating APIs: Their Code

- **Circumvented boundaries / reliance on impl details?**

```
playback_engine player;
auto &decoder = player.get_decoder();
decoder.push_bytes(magic_skip_bytes);
```

Evaluating APIs: Their Code

- **Circumvented boundaries / reliance on impl details?**

```
playback_engine player;
player.skip_frame();
```

Evaluating APIs: Their Code

- **Circumvented boundaries / reliance on impl details?**
- **Unexpected acrobatics?**

```
image yuv_image, bw_image;
size_t idx = 0;
for (auto pixel : yuv_image)
  bw_image.set(idx++, pixel.monochrome());
```

Evaluating APIs: Their Code

- Circumvented boundaries / reliance on impl details?
- Unexpected acrobatics?

```
image yuv_image;
image bw_image;
bw_image.blit_from(yuv_image);
```

Evaluating APIs: Their Code

- Circumvented boundaries / reliance on impl details?
- Unexpected acrobatics?

```
image yuv_image;
image bw_image =
  yuv_image.convert_to(monochrome);
```

Evaluating APIs: Their Code

- Circumvented boundaries / reliance on impl details?
- Unexpected acrobatics?
- Do people cheat on your API?

```
sound ding;
OS_INTERNAL_API::SetSystemVolume(-6dB);
ding.play();
```

Evaluating APIs: Their Code

- Circumvented boundaries / reliance on impl details?
- Unexpected acrobatics?
- Do people cheat on your API?

```
cool_sound ding(-6dB, false);
ding.blast();
```

Evaluating APIs: Their Code

- Circumvented boundaries / reliance on impl details?
- Unexpected acrobatics?
- Do people cheat on your API?

```
sound ding;
ding.play(-6dB);
```

Evaluating APIs: Their Code

- Circumvented boundaries / reliance on impl details?
- Unexpected acrobatics?
- Do people cheat on your API?
- Best interface for this use-case?

```
image gradient;
for (auto& pixel : gradient)
  pixel = mix(red, blue, pixel.x / gradient.w);
```

Evaluating APIs: Their Code

- Circumvented boundaries / reliance on impl details?
- Unexpected acrobatics?
- Do people cheat on your API?
- Best interface for this use-case?

```
image gradient =
  image::from_gradient(red, blue, horizontal);
```

Evaluating APIs: Their Code

- Circumvented boundaries / reliance on impl details?
- Unexpected acrobatics?
- Do people cheat on your API?
- Best interface for this use-case?
- Any common issues?

```
size_t image::size(); // total pixel count
std::byte * image::data(); // raw data
file.write(img.data(), img.size());
```

Evaluating APIs: Their Code

- Circumvented boundaries / reliance on impl details?
- Unexpected acrobatics?
- Do people cheat on your API?
- Best interface for this use-case?
- Any common issues?

```
file.write(img.data(),
           img.size() * img.format.pixel_size());
```

Takeaways for building APIs

- Building good APIs consistently requires design!
- What is your target audience looking for?
- How does your API/Object simplify the mental model?
- Design systems top-to-bottom, not from impl details
- Name things by what they REPRESENT, not what they ARE
- Encourage simple & readable Client Code
- Consistency is key
- Eliminate unexpected behavior