Please wait for the presentation to start...

Geofront:

Directly accessible GIS tools using a web-based visual programming language

Master Thesis Geomatics | Final Presentation

Jos Feenstra | November 4th 2022

TU Delft

1. Introduction
2. Objective
3. Background
4. Method
5. Results
6. Conclusion

TUDelft

1. Introduction

GIS:

Geographical

Information

Science

End Users

- Climatology
- Infrastructure
- Urban planning
- Agriculture
- Governance
- Navigation
- Military
- (...)

Geographical

Information

Science

Tools

End Users

- Environmental studies
- Infrastructure
- Urban planning
- Governance
- Navigation
- The military
- Argiculture

TU Delft

Geographical

Information

Science

Tools

End Users

- Environmental studies
- Infrastructure
- Urban planning
- Governance
- Navigation
- The military
- Argiculture

TU Delft

Tools: Two forms of software:

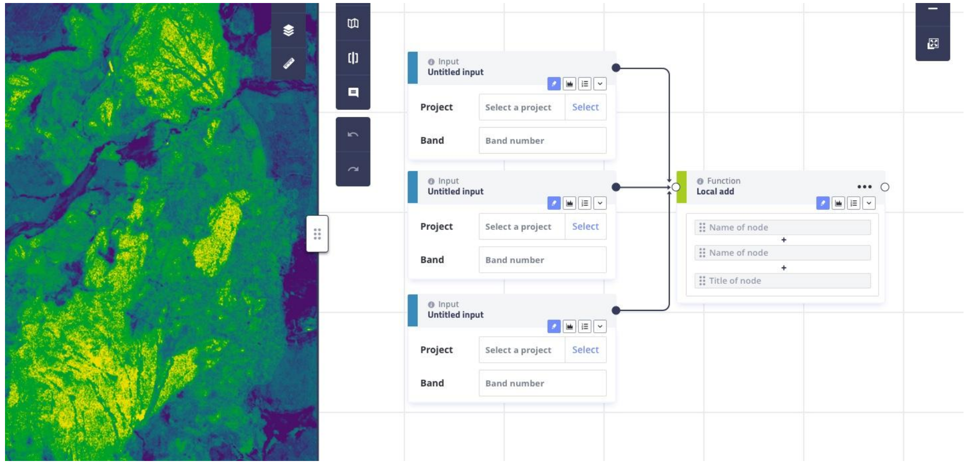
Application

Library

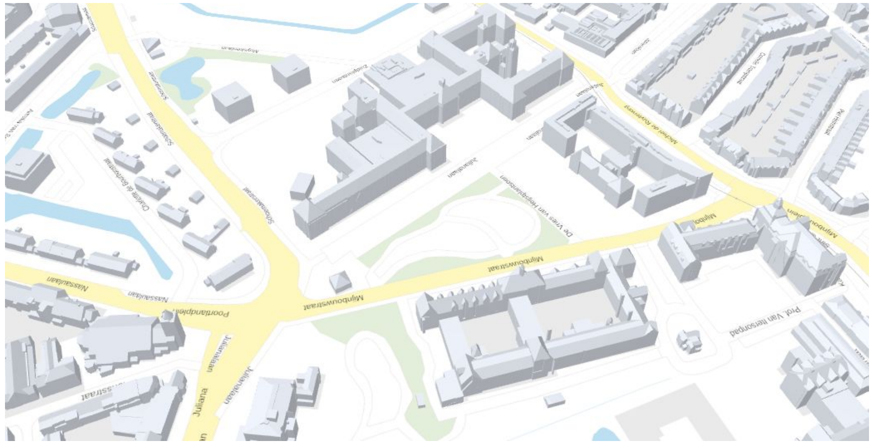
Application

Applications:

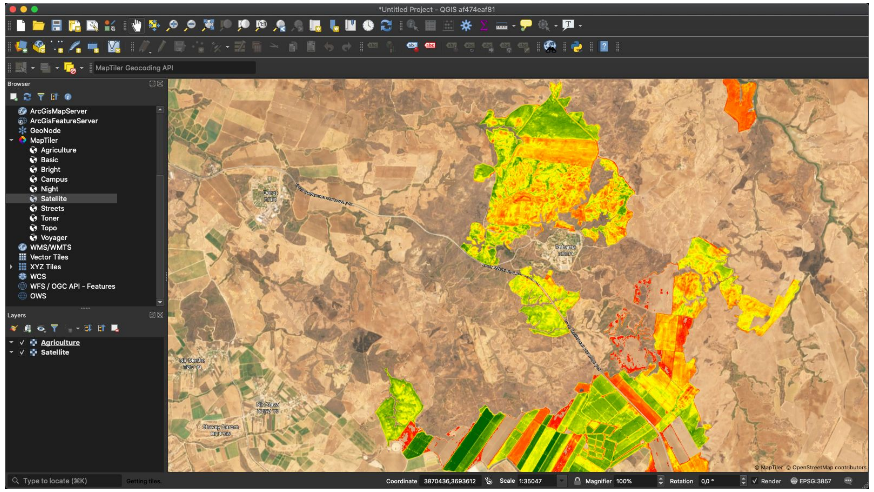
- End users
- Interaction



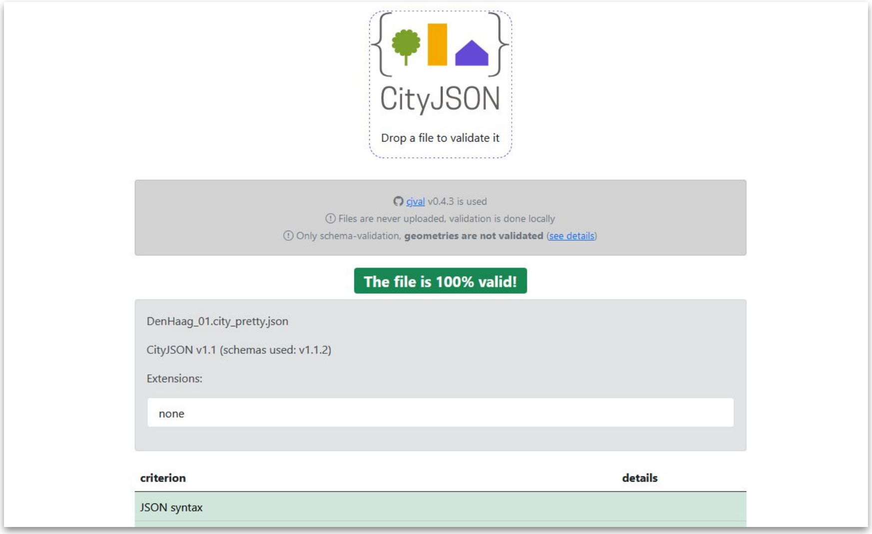
src: Model Lab



src: 3D bag viewer



src: QGIS



src: cjval

TU Delft

Libraries

Libraries:

- Reusable tools for applications (& other libraries)
- Cannot directly be used

In GIS:

- Transformation
- Analysis (Validation)

3D Geoinformation Libraries

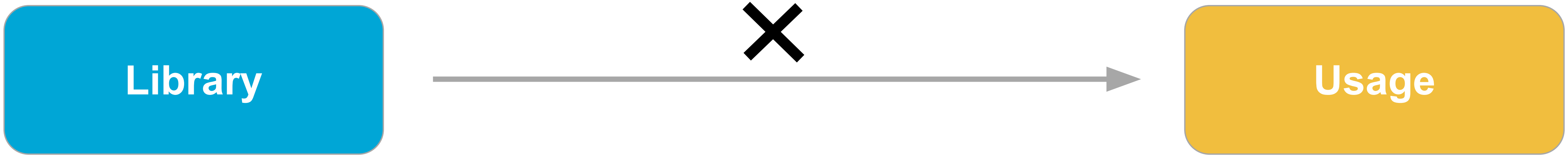
- City3D
- PolyFit
- val3dity
- cgval
- cjio

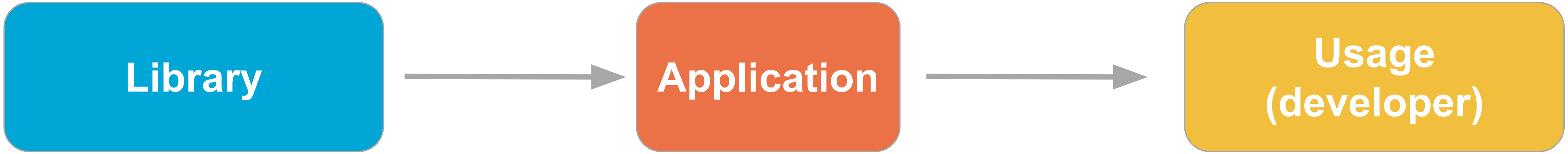
"Core" GIS Libraries

- PROJ
- GEOS
- GDAL
- (CGAL)
- (OpenCV)

TUDelft

Usability: Libraries are not directly usable





Library
Application
Usage
(developer)

Problem:

(core) GIS Library → **?** → **End Users**

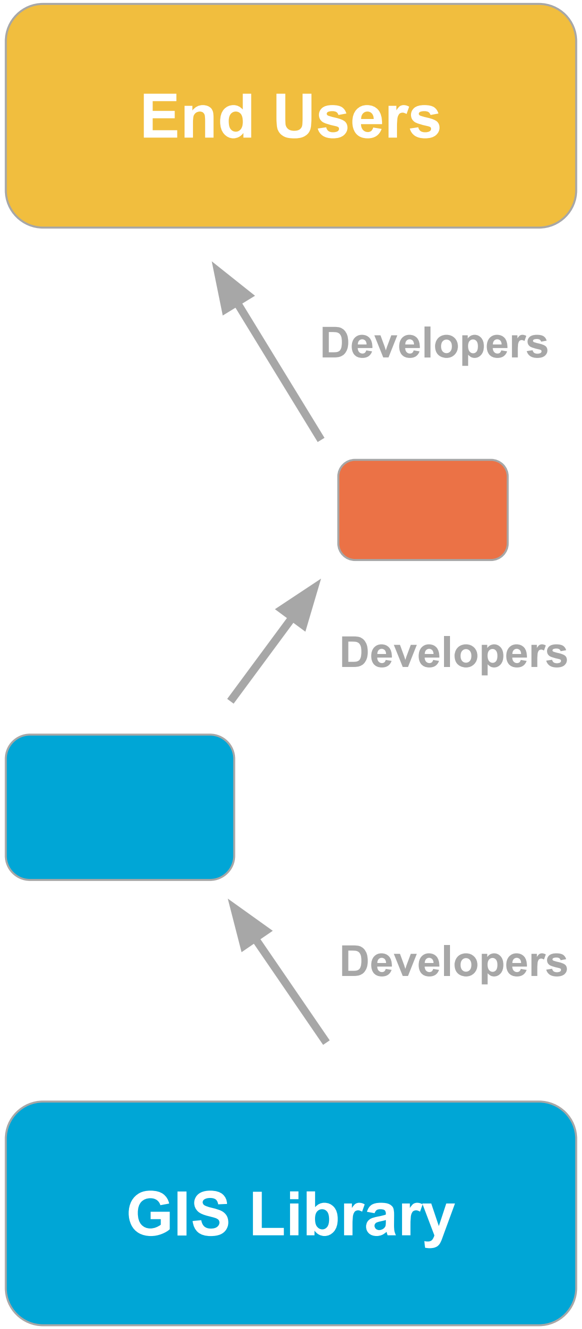
- Transformation
- Analysis
    - Validation

Environmental studies
Infrastructure
Urban planning
Governance
Navigation
The military
Argiculture

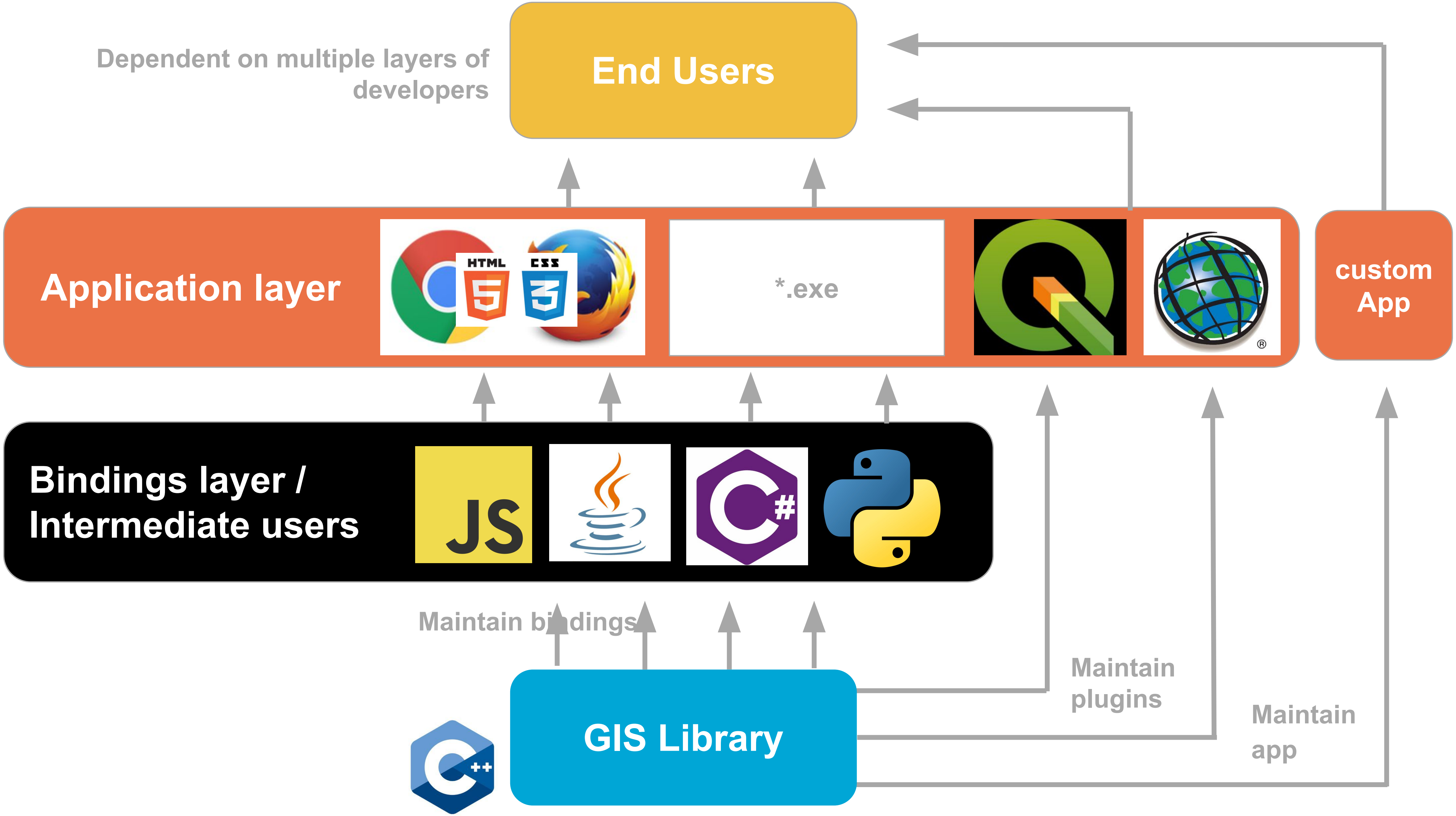
TU Delft

Indirection

- Only indirect access
- Dependent
- Exact?
- New research?

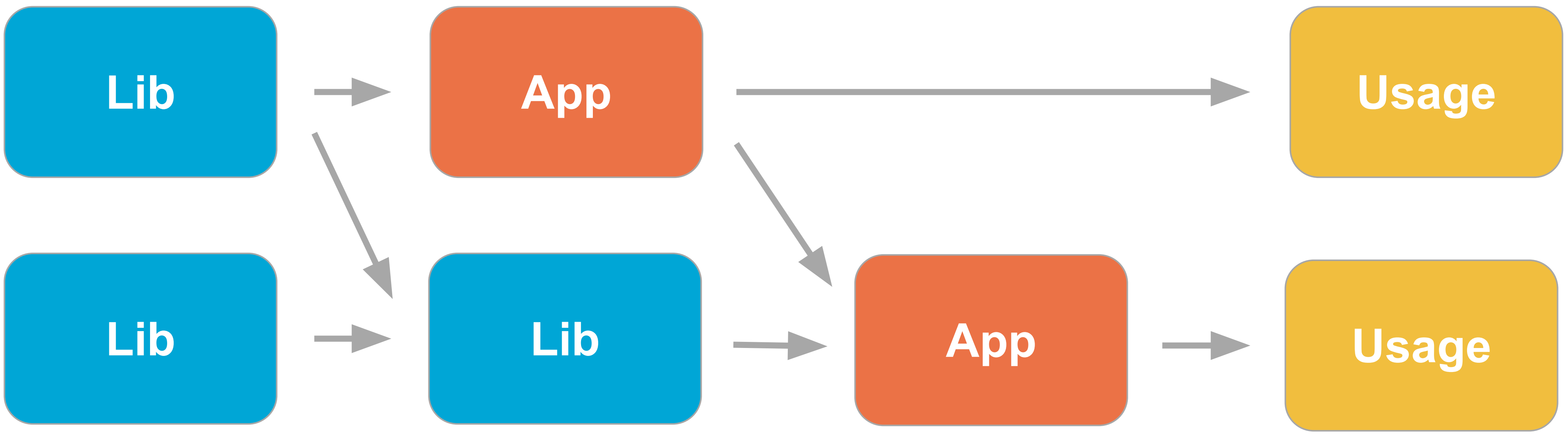


TU Delft



Dependent on multiple layers of developers
End Users
Application layer
HTML
5
CSS
3
*.exe
custom App
Bindings layer / Intermediate users
JS
Maintain bindings
Maintain plugins
Maintain app
GIS Library

Moreover:

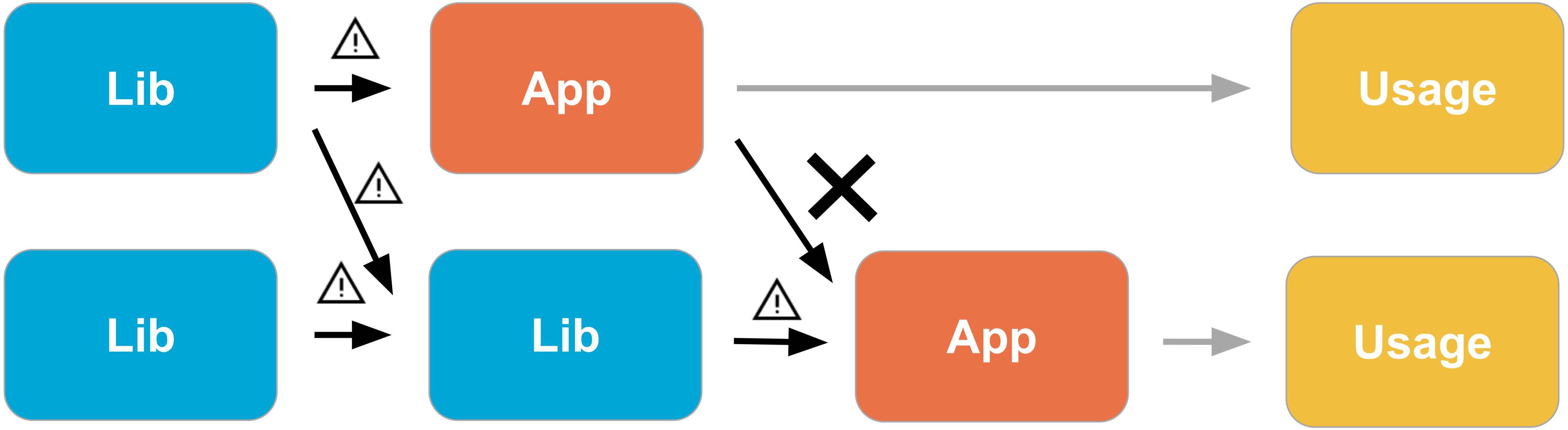


TU Delft

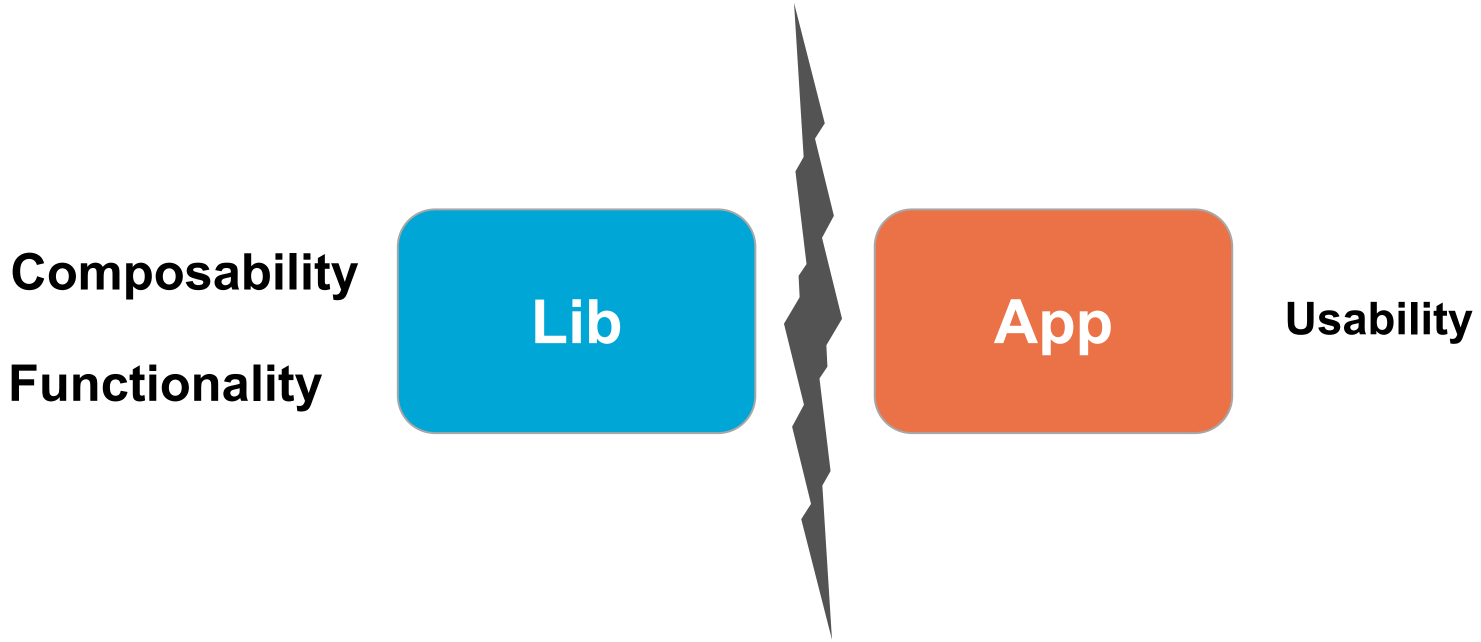
Moreover:

⚠ **Functionality:** capabilities may get lost at every step

✕ **Composability:** apps are **not** further composable



Conundrum:



Given this divide, how to achieve **Functionality, Composability,** & **Usability** at the same time?

TU Delft

Adopted from Elliott C. (2007).
Tangible Functional Programming

Problem statement

End users only have **Indirect access** of GIS libraries, leading to disadvantages…

… for **end users**:

- *At the mercy* of in-between software
- Non-composable applications
- Features getting *lost in translation*

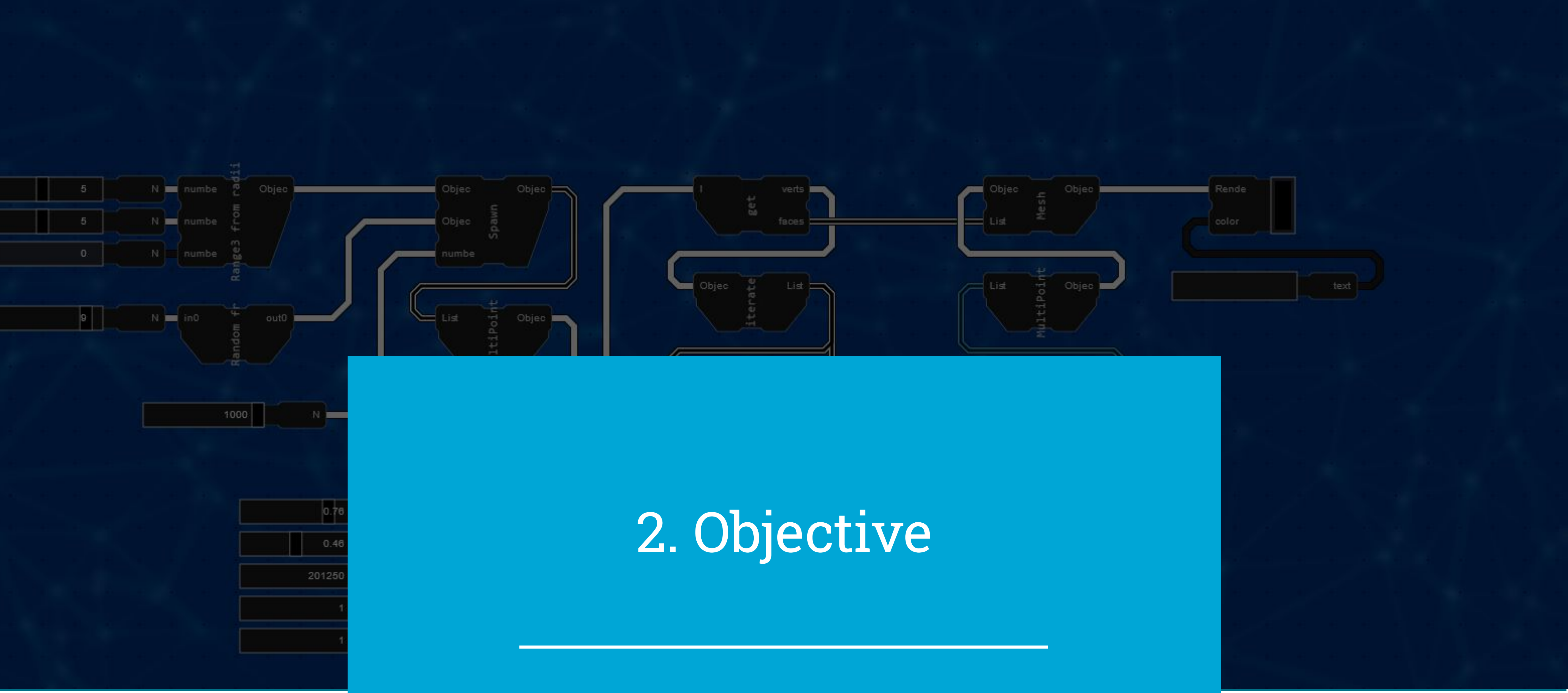
… for **library developers:**

- Synchronizing bindings, plugins, applications.

…for **society:**

- reduced impact of research

TUDelft

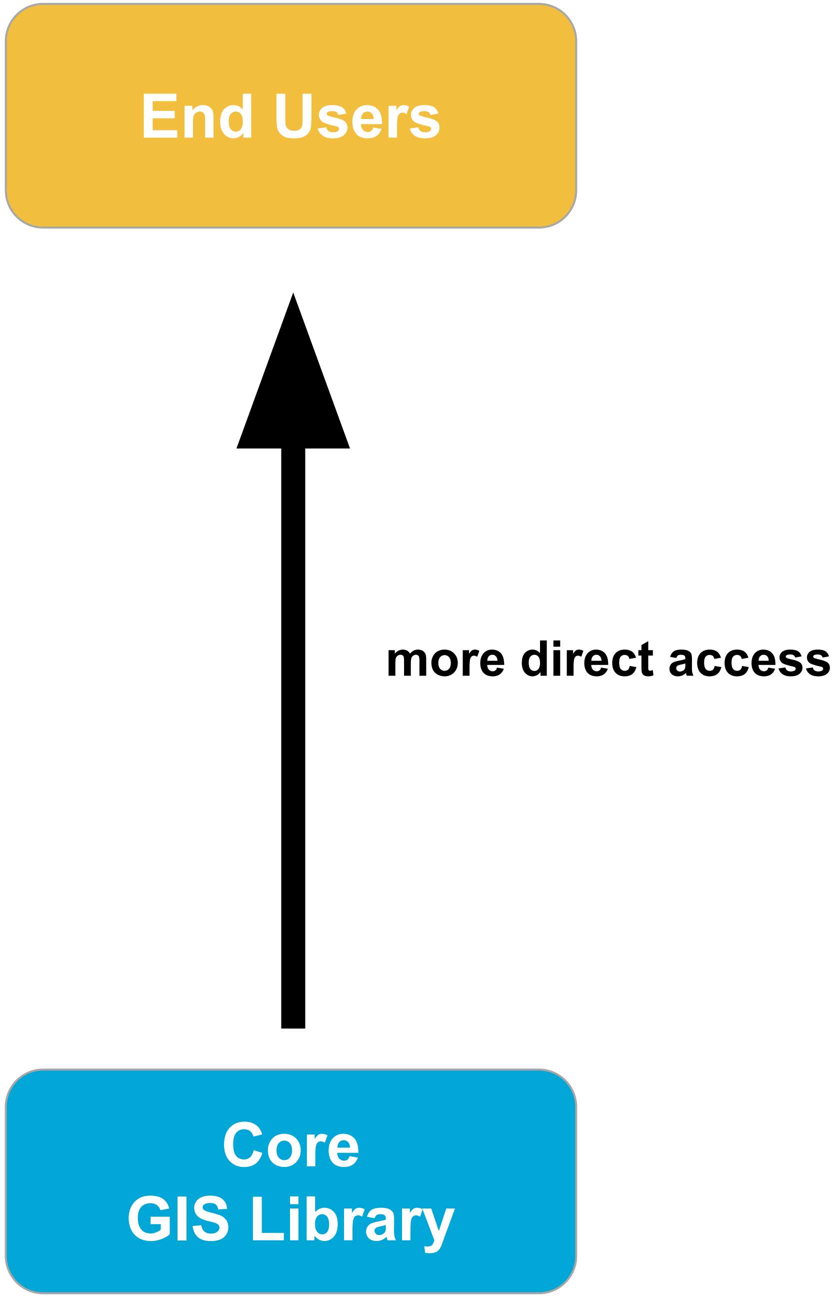

Range3 from radii
Spawn
get
Mesh
iterate
MultiPoint
Random fr
verts
faces
Rende
color
text
1000
0.76
0.46
201250

2. Objective

Goal of this study:

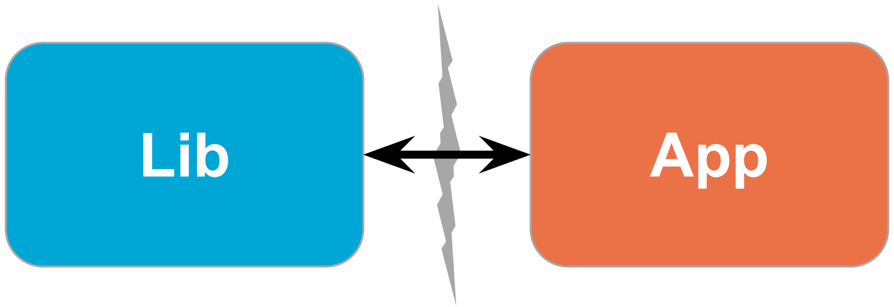
Allow GIS practitioners without a background in software development, to access the full potential of core transformation and analysis capabilities found in native GIS libraries.

Goal:



TU Delft

Goal:



apps are an endpoint: Not further composable → Add **composability** and **automation** to apps

A lib offers no visualization or GUI. → Add **usability** and **GUI** to libs

A lib must be turned into an app before utilization. → Add **usability** and **GUI** to libs

Some Lib capabilities get lost when used in an app → Add **usability** and **GUI** to libs

How:

Presenting and prototyping a novel method:

A **Web-based Visual Programming Language (VPL)** using **WebAssembly**

TU Delft

Research question:
Is a **web based VPL** a viable method for directly accessing native GIS libraries with a composable interface?

Sub Questions

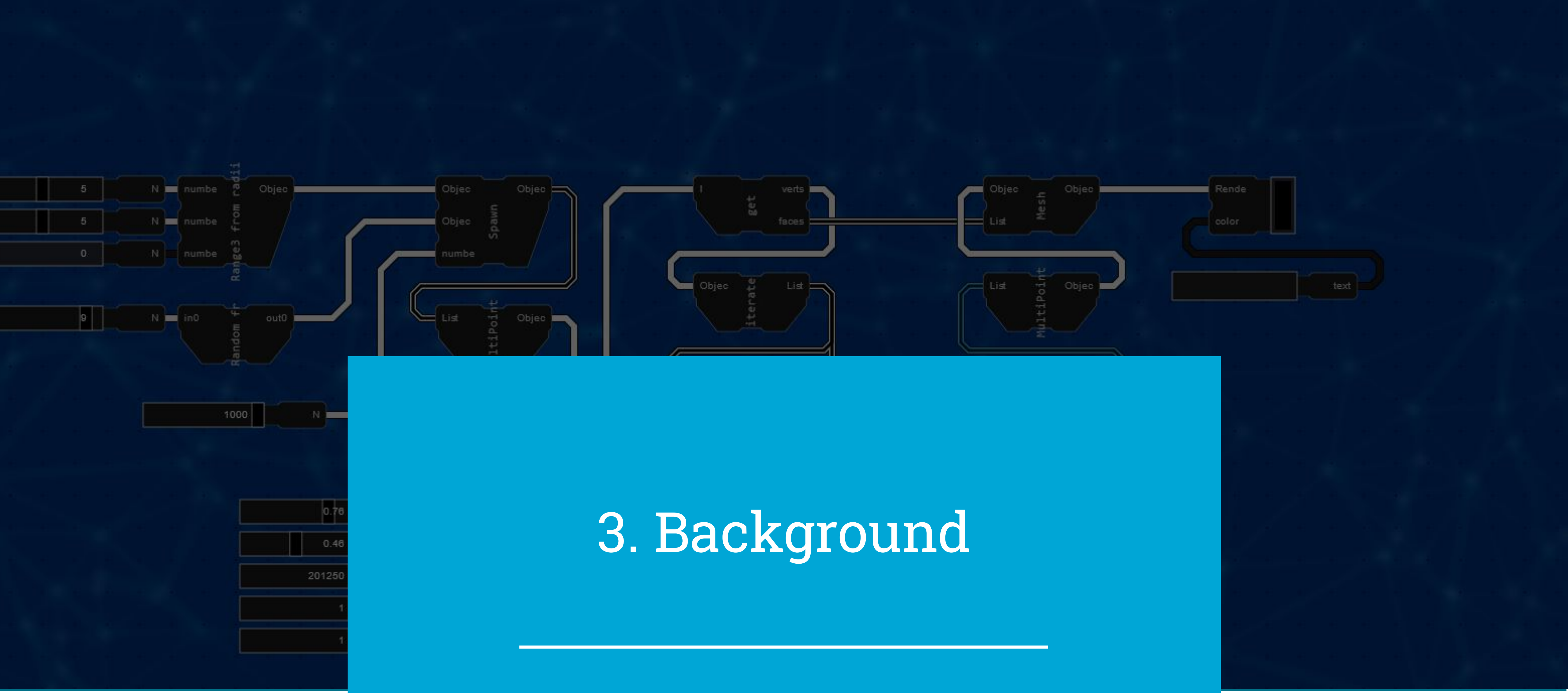
- What **GUI** features are required to facilitate this method, and to what extent does the web platform aid or hurt these features?

- To what extent does this method intent to address the **discrepancies** between **software applications and libraries**, as described by Elliott (2007)? Does it succeed in doing so?

- What are the differences between **compiling** a GIS library written in C++ to WebAssembly, compared to compiling a GIS library written in Rust?

- What measures are taken to make this VPL **scalable** to large geo-datasets, and how effective are these measures?

- How does this method **compare** to existing, alternative VPLs and browser-based geocomputation methods, regarding the properties relevant to the goal of direct accessibility?



3. Background

TU Delft

“Web-based VPL using **WebAssembly”:**

1. **Web Application**
2. **Visual Programming**
3. **WebAssembly**

1. Web Applications

Possible solution for direct access

Web Application → distribution

- No Installation
- Cross-platform

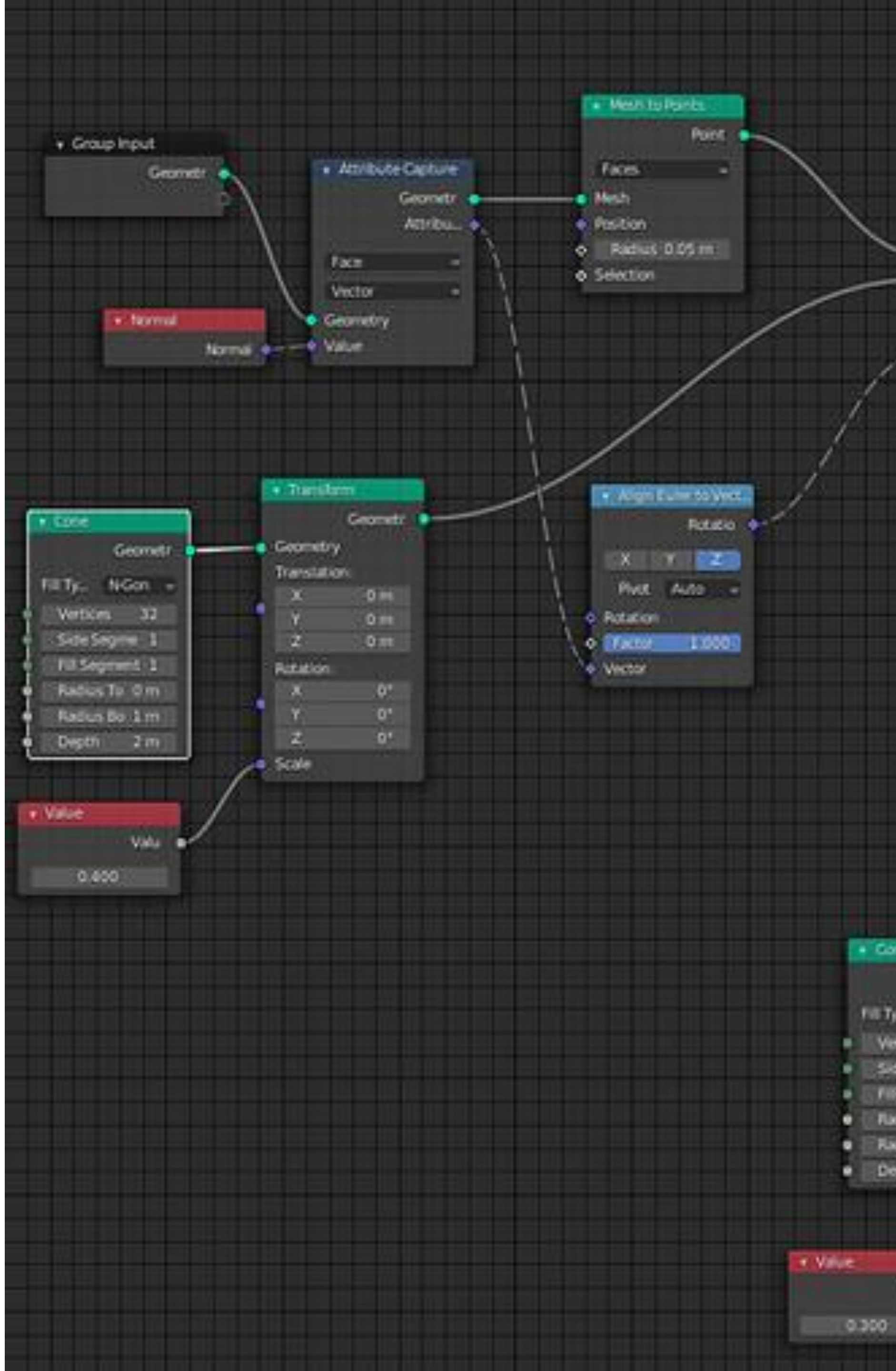
Static Web Application

- No active backend
- Cheap
- More portable

2. Visual Programming

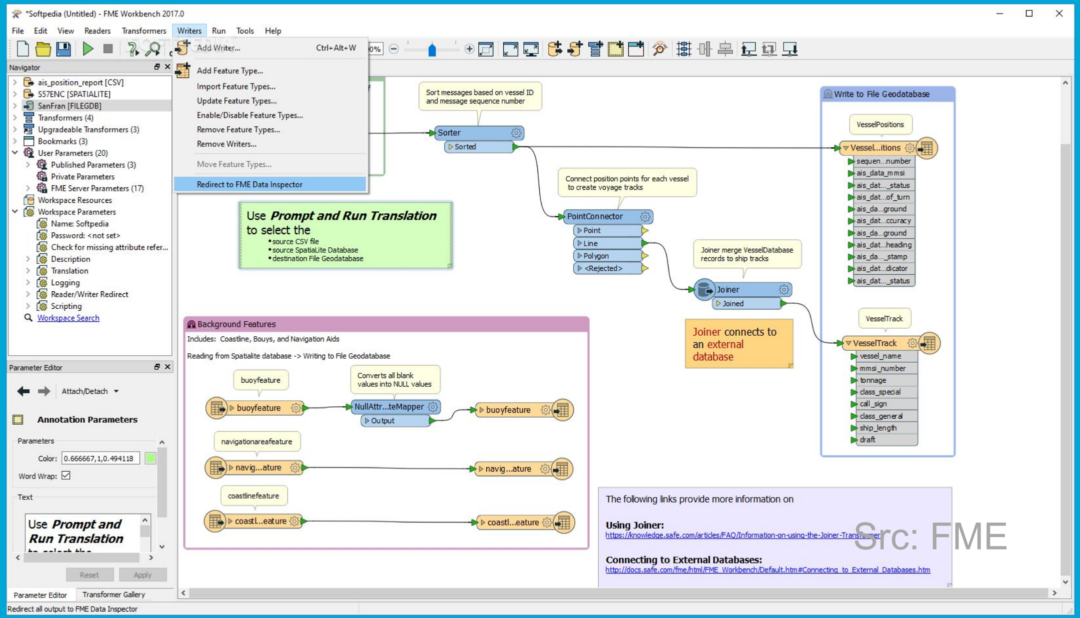
Possible solution for Composable applications

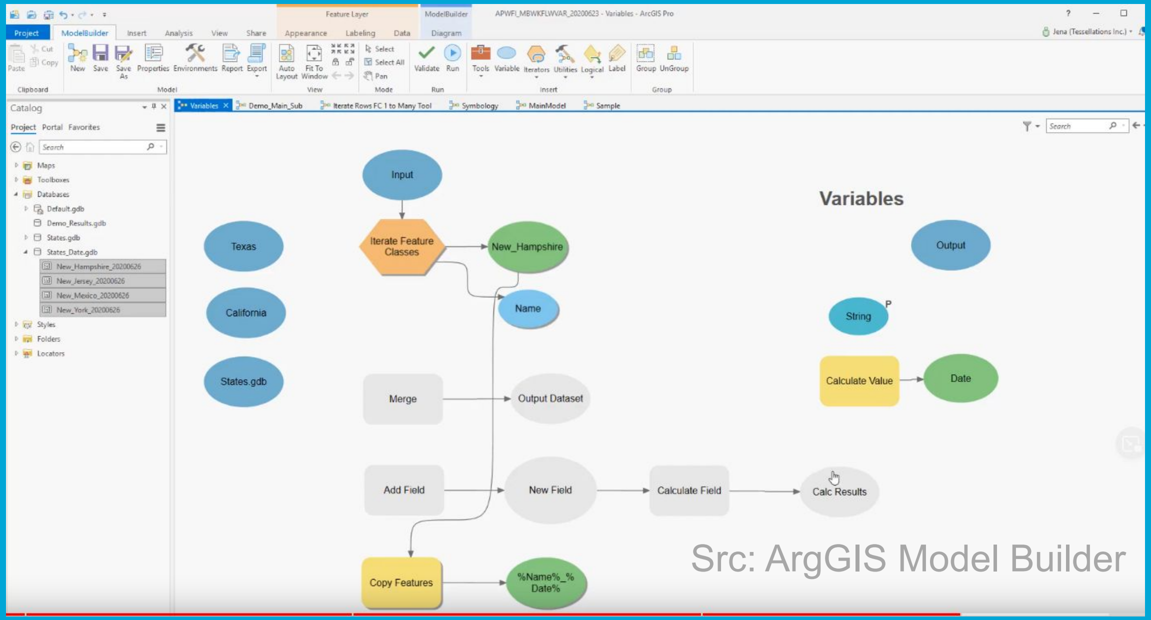
- Visual Programming Language (VPL)
- Both a scripting language **and** application
- ‘programming’ by using GUI
    - Composable GUI

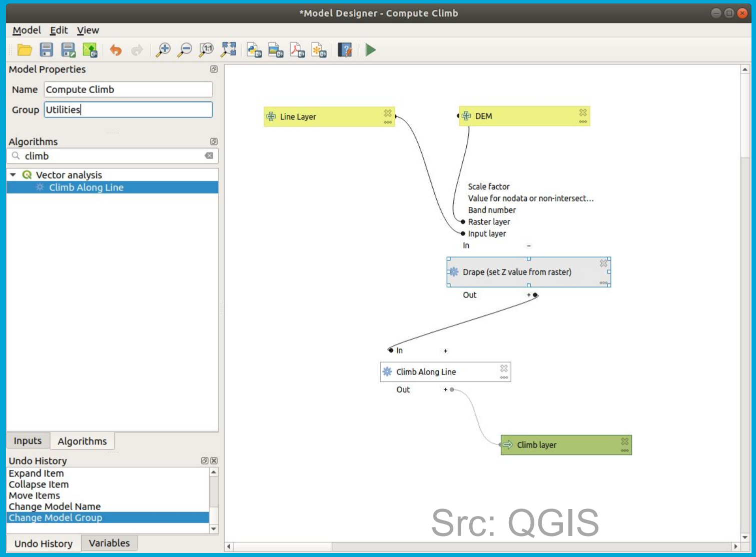


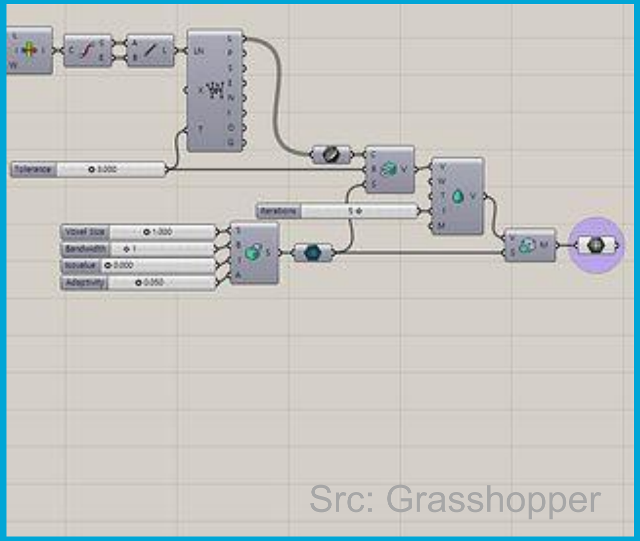
src: Blender Geometry Nodes

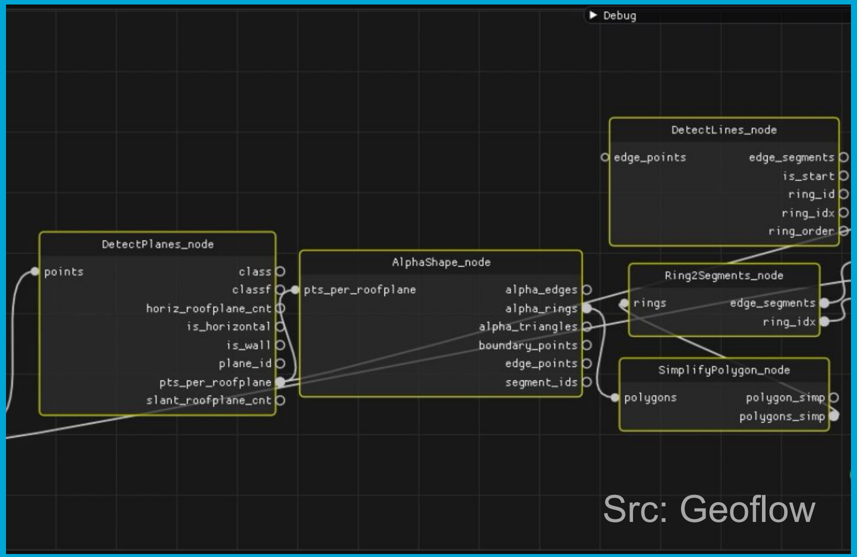
VPL within GIS





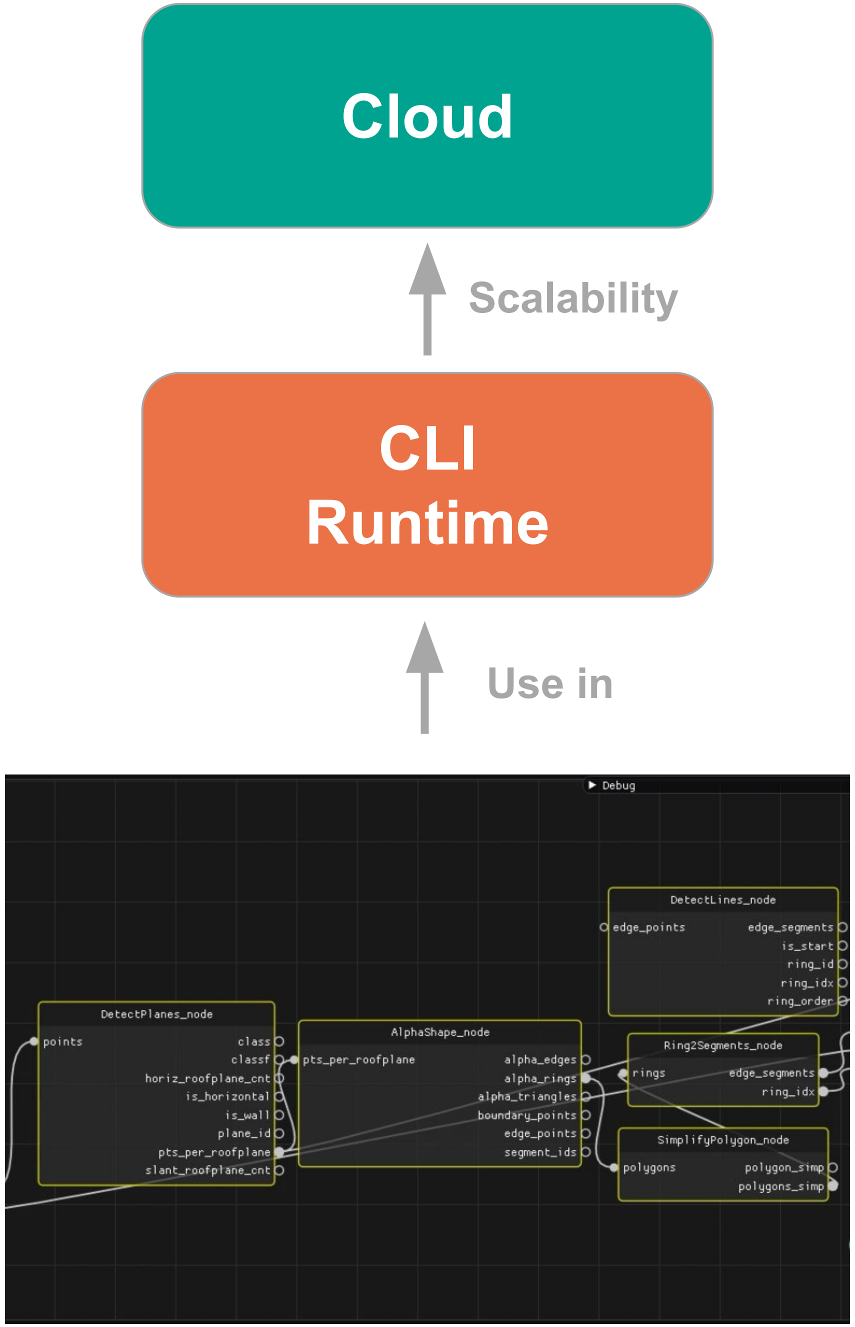






2. Visual Programming: GIS

requirement: Scalability



TU Delft

3. WebAssembly

Possible solution for more direct access

- Exchangeable binaries
- Binary compilation target `library.wasm`
  - From multiple languages
  - To multiple runtimes
- Since 2017 (Haas, 2017)
- In browsers since 2019 (W3C, 2019)

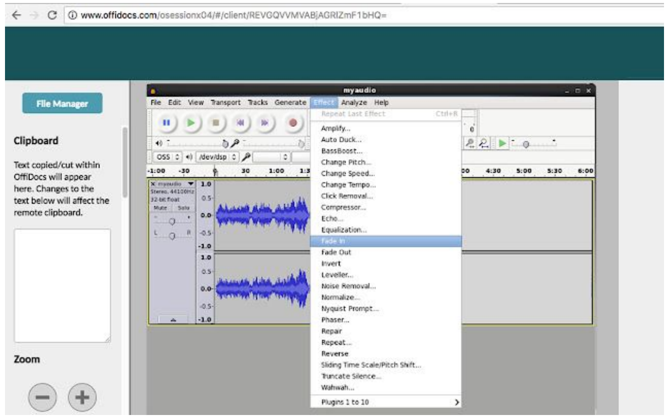
```
(module
  (type $t0 (func (param i32 i32)))
  (type $t1 (func (param i32 i32 i32) (result i32)))
  (type $t2 (func (param i32 i32) (result i32)))
  (type $t3 (func (param i32)))
  (type $t4 (func (param i32) (result i32)))
  (type $t5 (func))
  (type $t6 (func (param i32) (result f32)))
  (type $t7 (func (param i32 f32)))
  (type $t8 (func (param f32 f32) (result i32)))
  (type $t9 (func (param i32 i32) (result f32)))
  (type $t10 (func (param i32 i32 i32)))
  (type $t11 (func (param i32 i32 i32 i32) (result i32)))
  (type $t12 (func (result i32)))
  (type $t13 (func (param i32) (result i64)))
  (type $t14 (func (param i32 i32 i32 i32)))
  (type $t15 (func (param i32 i32 i32 i32 i32)))
  (type $t16 (func (param i32 i32 i32 i32 i32) (result i32)))
  (type $t17 (func (param i32 i32 i32 i32 i32 i32) (result i32)))
  (type $t18 (func (param i64 i32 i32) (result i32)))
  (import "__wbindgen_placeholder__" "__wbindgen_describe" (func $_ZN12wasm_bindgen19__wbindgen_describe17hdb3
  (import "__wbindgen_externref_xform__" "__wbindgen_externref_table_grow" (func $_ZN12wasm_bindgen9externref3
  (import "__wbindgen_externref_xform__" "__wbindgen_externref_table_set_null" (func $_ZN12wasm_bindgen9extern
$t3)))
  (import "__wbindgen_placeholder__" "__wbindgen_throw" (func $_ZN12wasm_bindgen16__wbindgen_throw17h7bfc15cf6
  (func $add (type $t2) (param $p0 i32) (param $p1 i32) (result i32)
    local.get $p1
    local.get $p0
    i32.add)
  (func $__wbindgen_describe_add (type $t5)
    call $_ZN12wasm_bindgen4__rt19link_mem_intrinsics17h3cc2179cca039a8aE
    i32.const 11
    call $_ZN12wasm_bindgen19__wbindgen_describe17hdb3ff320fcac3194E
    i32.const 0
    call $_ZN12wasm_bindgen19__wbindgen_describe17hdb3ff320fcac3194E
    i32.const 2
    call $_ZN12wasm_bindgen19__wbindgen_describe17hdb3ff320fcac3194E
    call $_ZN60_$LT$u32$u20$as$u20$wasm_bindgen..describe..WasmDescribe$GT$8describe17h45229e62f39c456cE
    call $_ZN60_$LT$u32$u20$as$u20$wasm_bindgen..describe..WasmDescribe$GT$8describe17h45229e62f39c456cE
    call $_ZN60_$LT$u32$u20$as$u20$wasm_bindgen..describe..WasmDescribe$GT$8describe17h45229e62f39c456cE
    call $_ZN60_$LT$u32$u20$as$u20$wasm_bindgen..describe..WasmDescribe$GT$8describe17h45229e62f39c456cE)
  (func $__wbg_point_free (type $t3) (param $p0 i32)
    block $B0
      block $B1
        local.get $p0
        i32.eqz
        br_if $B1
        local.get $p0
        i32.load
        br_if $B0
```

wasm rendered at `.wat`
src: author

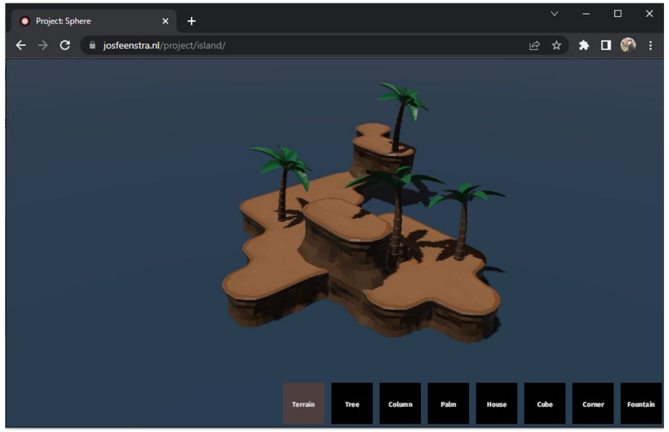
3. WebAssembly

Use case 1: Run native code in a browser

src: Milica Mihajlija



src: audacity



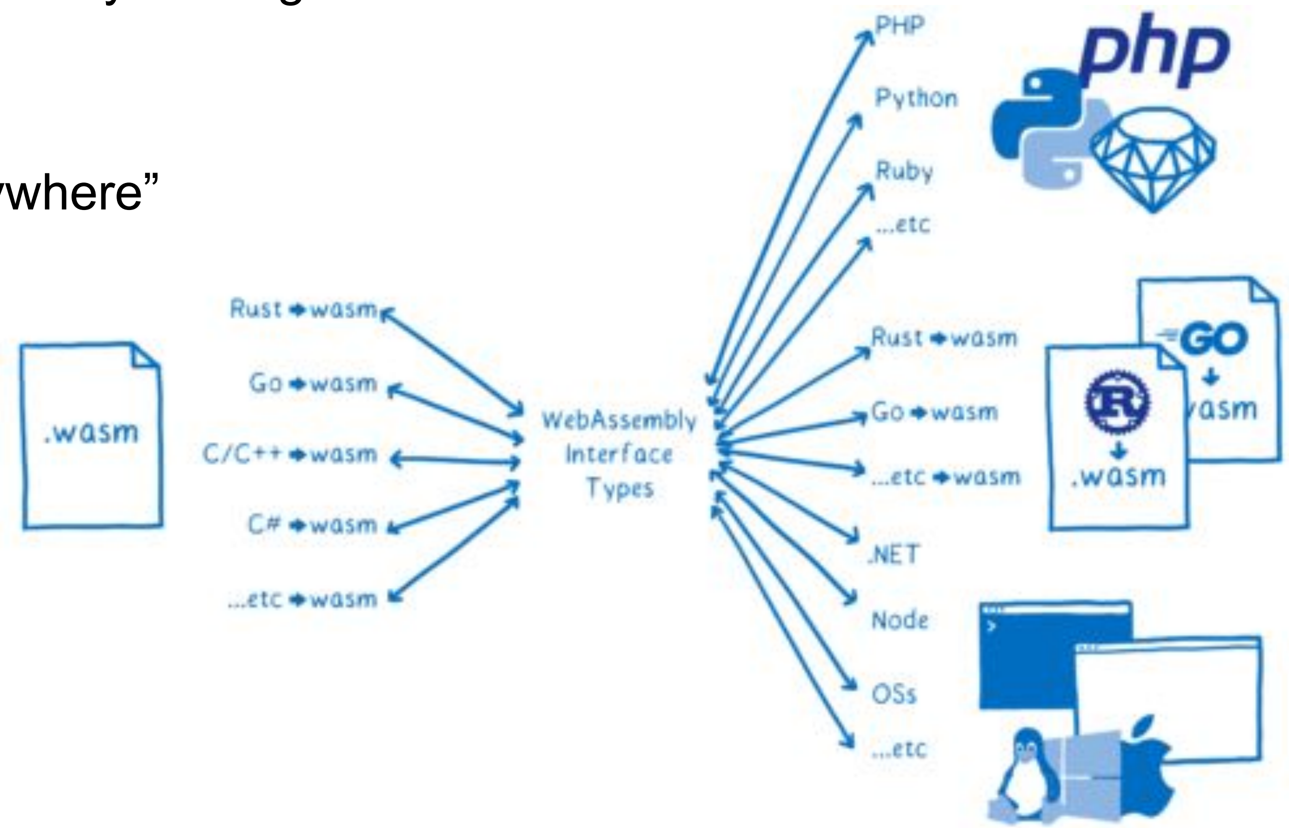
src: author

TU Delft

3. WebAssembly

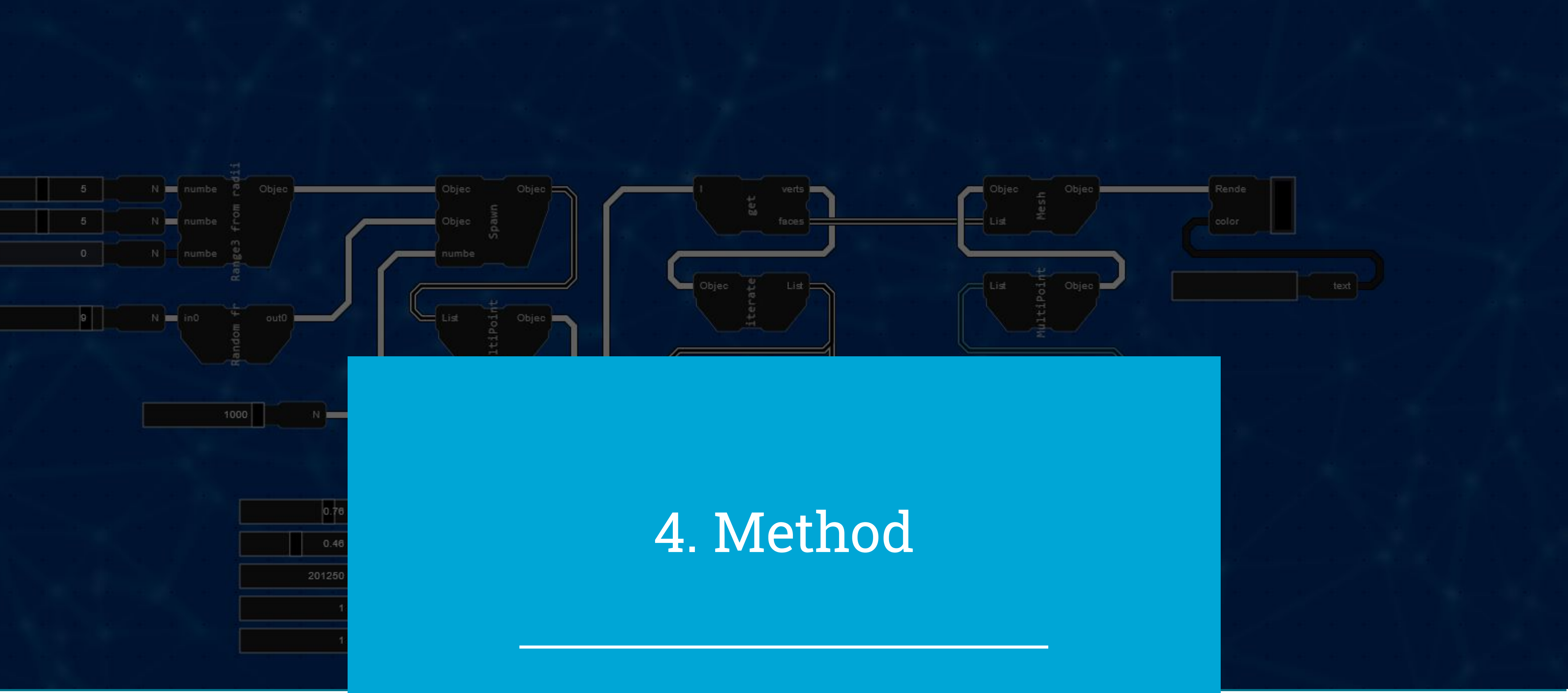
Use case 2: Generic library binding

- Interface Types
- “run anything anywhere”



Clark, L. (2019)

TU Delft

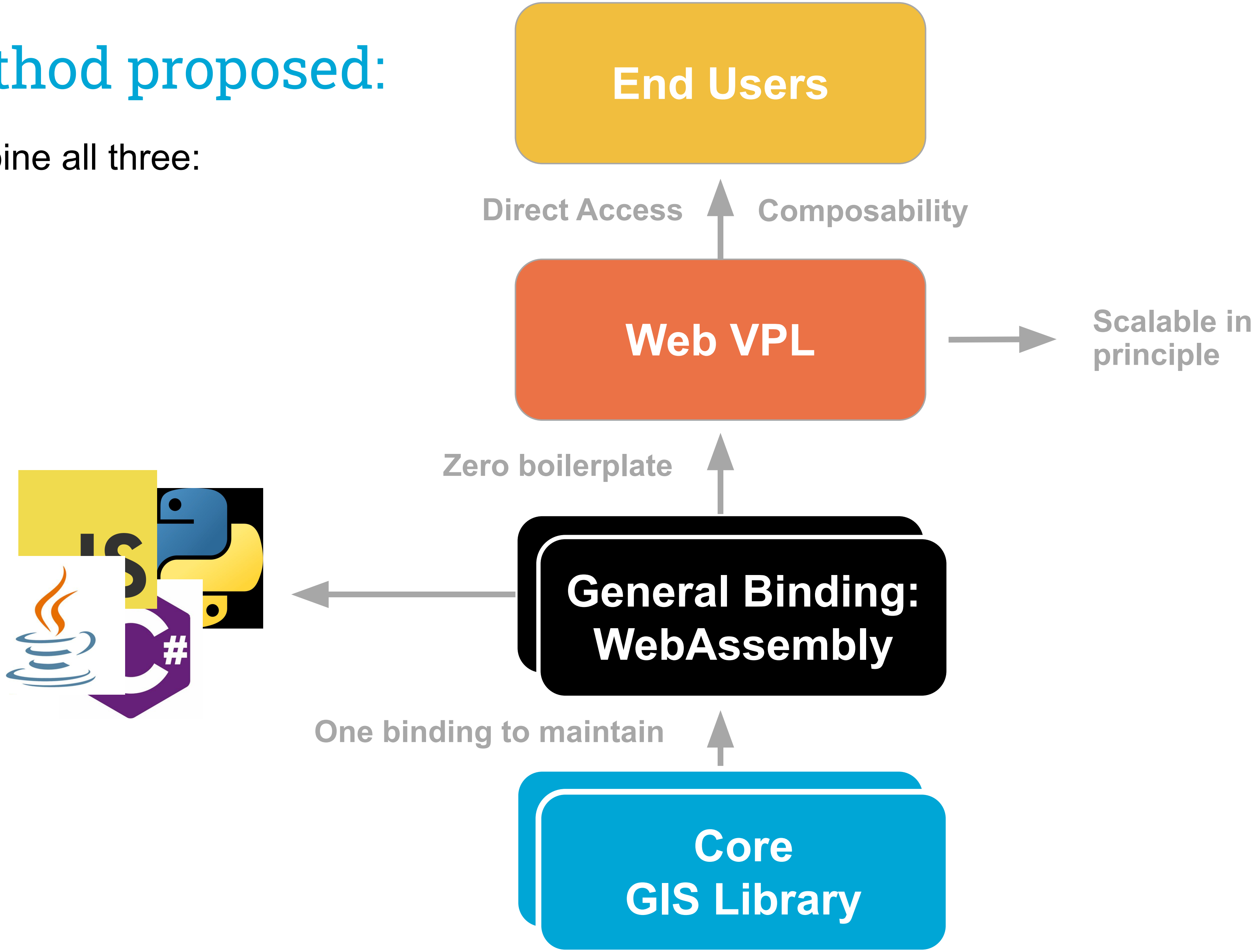


4. Method

TU Delft

Method proposed:

Combine all three:

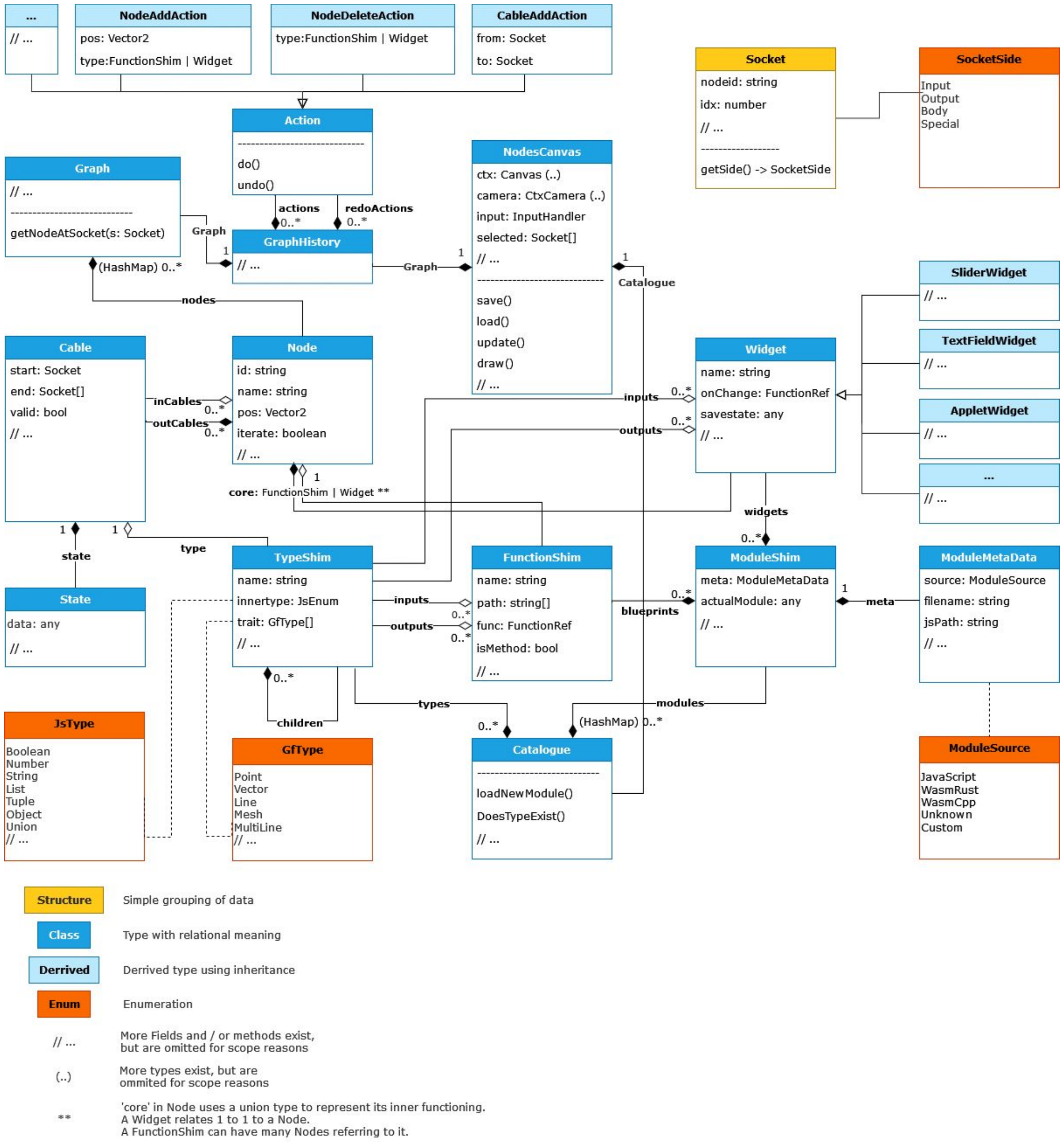


Two components

1. Web VPL
2. Library Plugin system
   - Plugin loader
   - Plugin model

1. Web VPL: Design

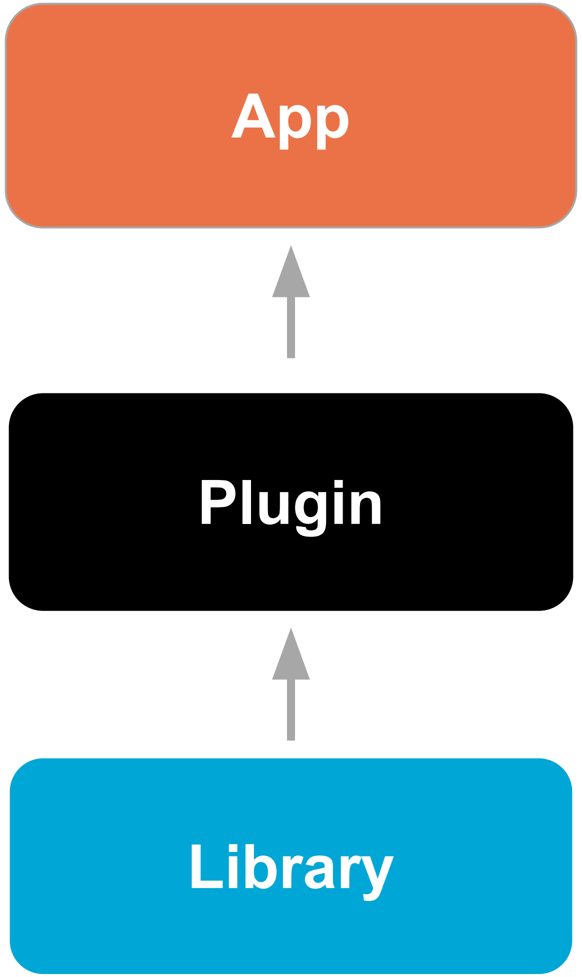
- Essentially, a programming language
  - “syntax tree”
- Model View Controller



2. Plugin System

Regular case:

- Maintain separate project
- Explicitly state interface
- ‘boilerplate’



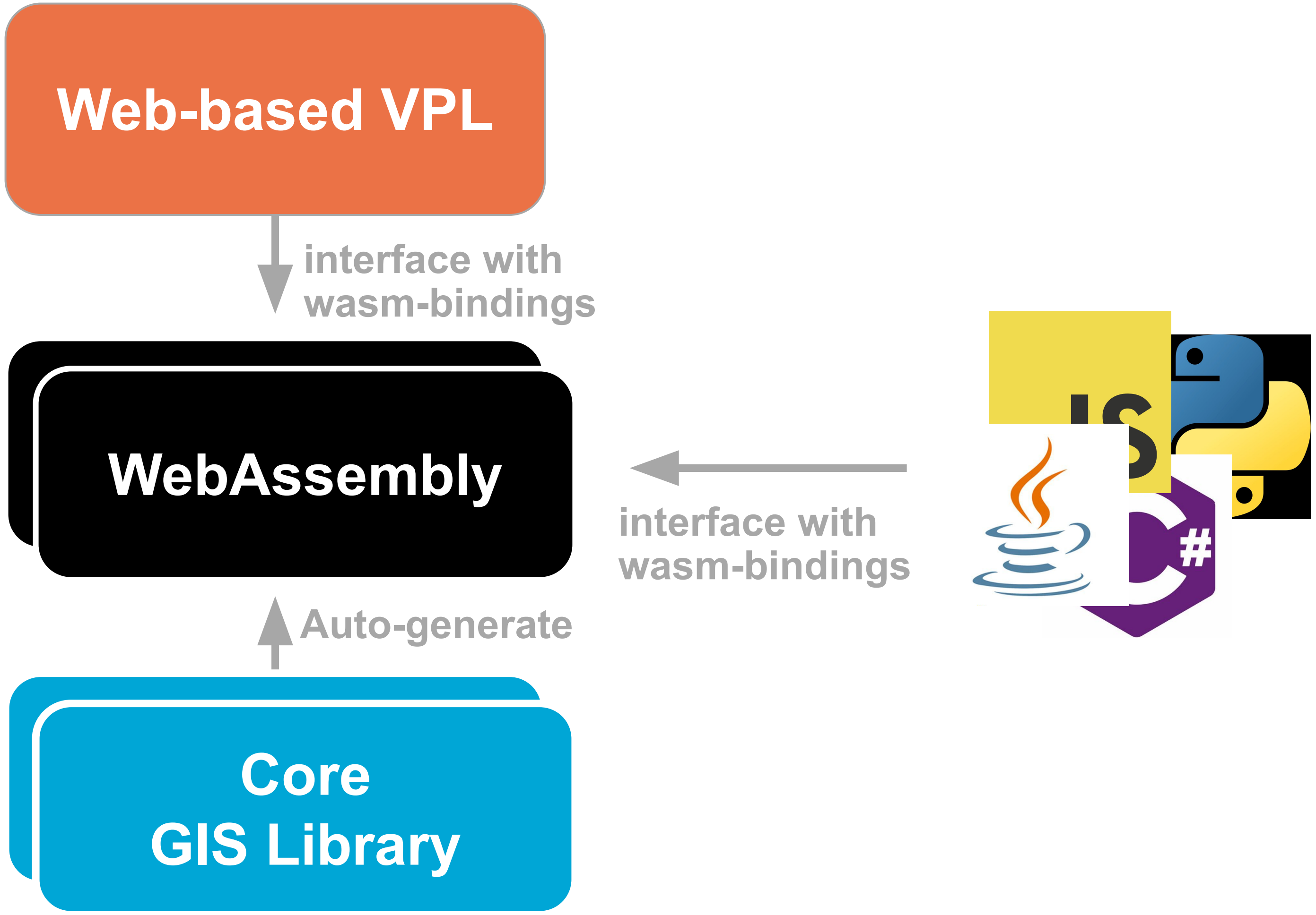
2. Plugin System

Our case:

- Leverage wasm compilers
- Mimic normal language
- Interprete bindings implicitly

Leads to:

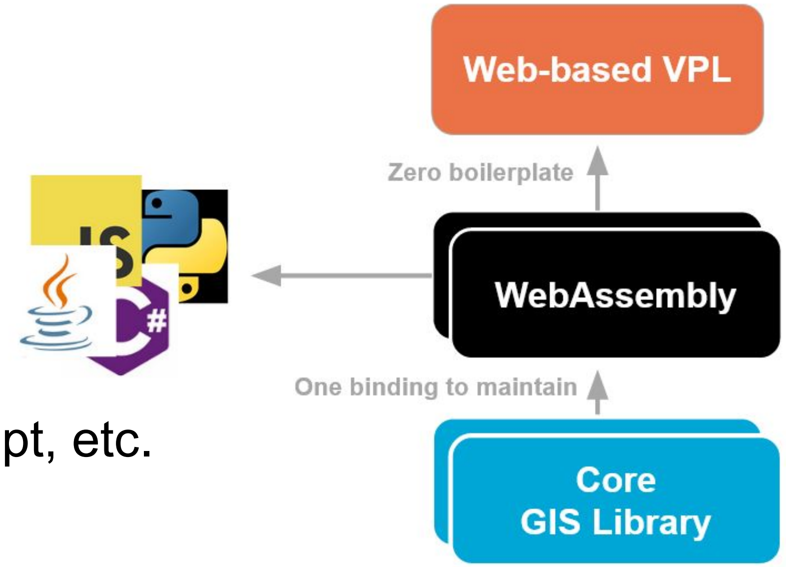
- No boilerplate
- Connect to existing infrastructure



2. Plugin System

Three elements:

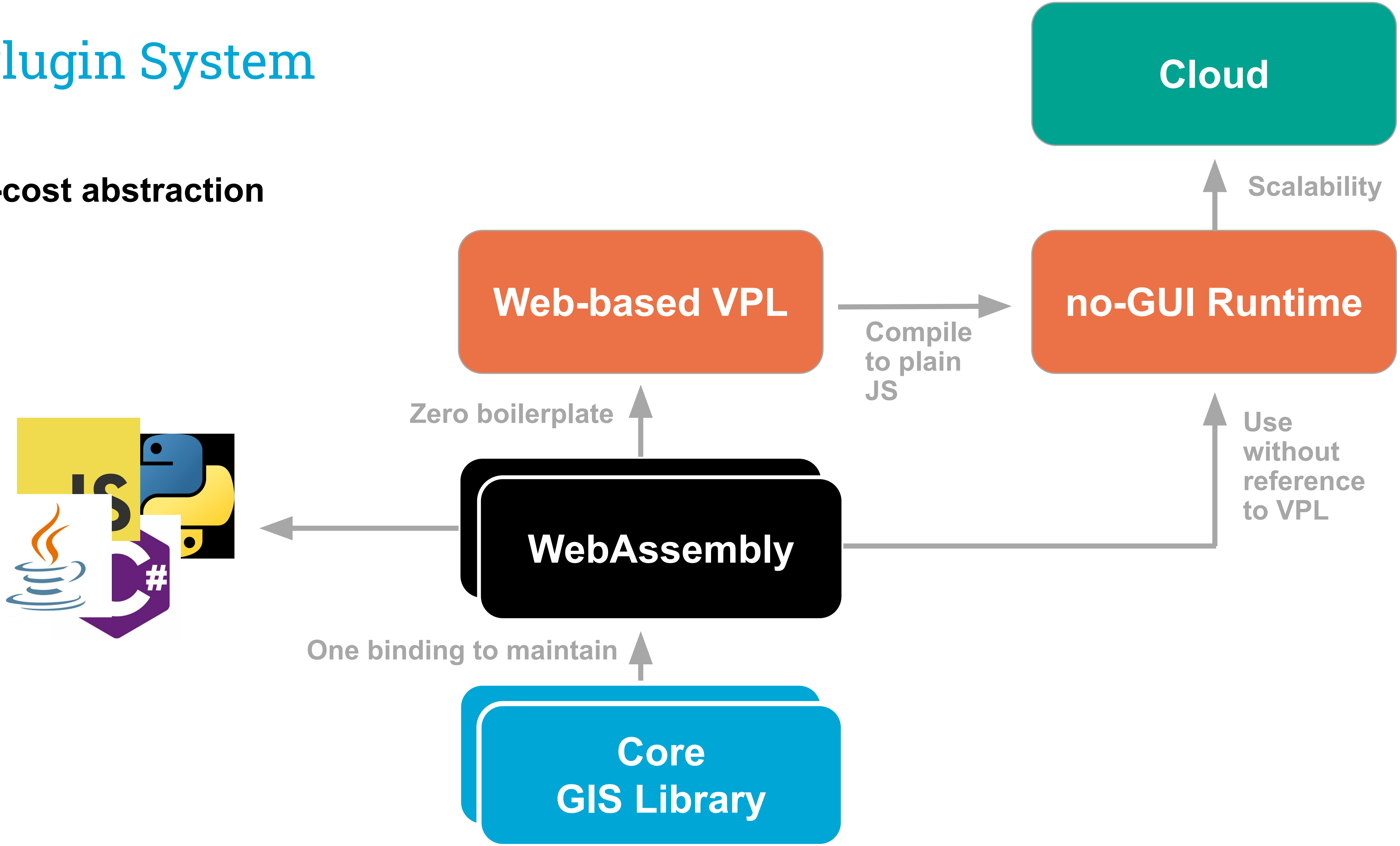
- **Direct utilization**→ Zero boilerplate
  - Leverage generic interface properties of WebAssembly
- **Portability**
  - Same behavior within this VPL as in python, C#, JavaScript, etc.
- **Scalability**
  - Zero-cost abstraction

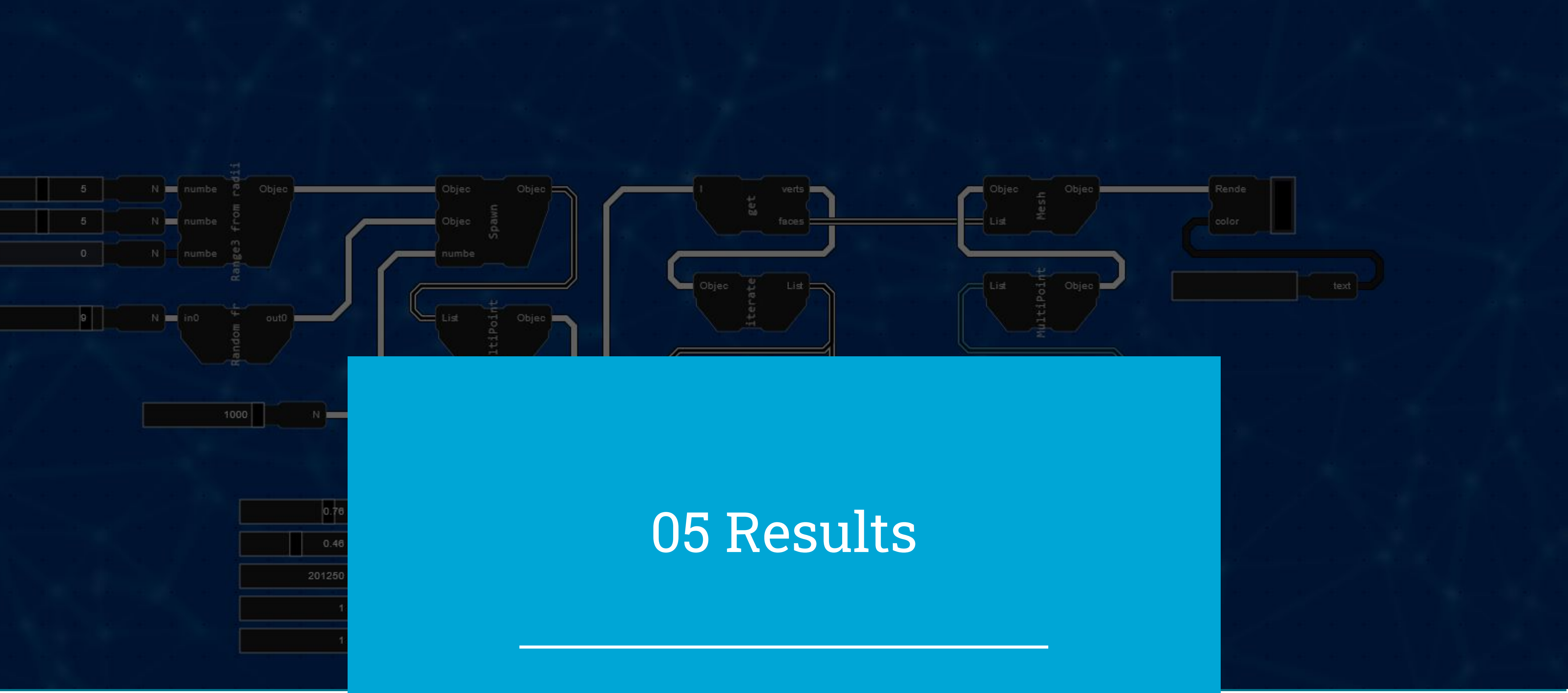


TU Delft

2. Plugin System

Zero-cost abstraction





05 Results

time?

1. Web VPL implementation

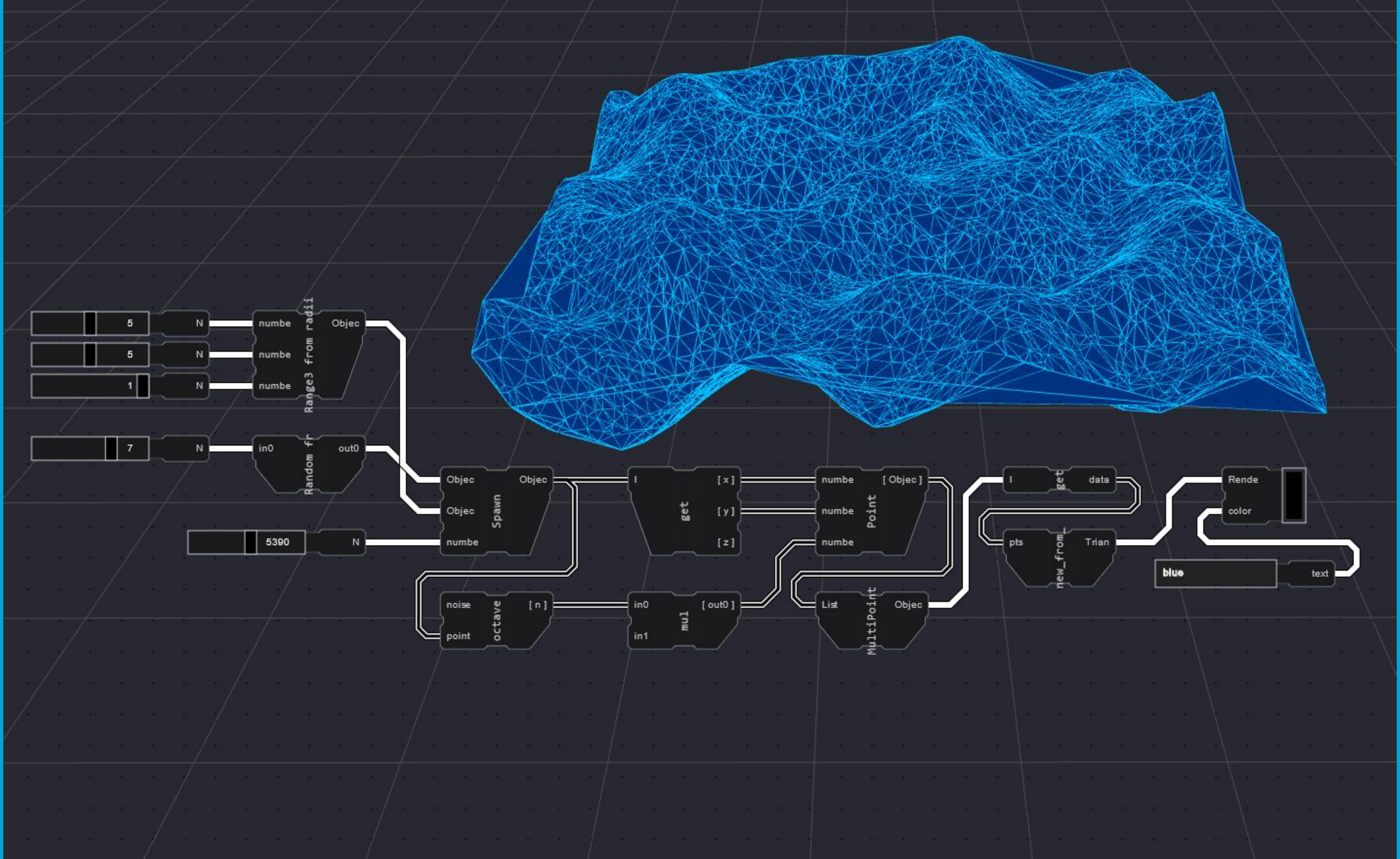
2. Library Plugin System implementation

1. Web VPL implementation: **Geofront**

Web VPL: **Geofront**

Custom implementation needed to meet all aspects of the method.

- Application framework
- VPL model implementation
- renderer, Interaction, UI, etc.

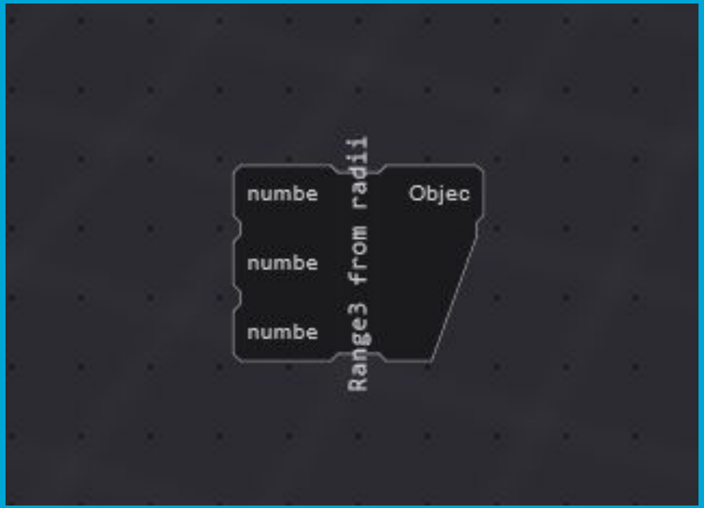


5
N
numbe
5
N
numbe
1
N
numbe
Range3 from radii
Objec
7
N
in0
Random fr
out0
Objec
Objec
Spawn
Objec
5390
N
numbe
I
get
[x]
[y]
[z]
numbe
numbe
numbe
Point
[Objec]
noise
point
octave
[n]
in0
in1
mul
[out0]
List
MultiPoint
Objec
I
get
data
pts
new_from_
Trian
Rende
color
blue
text

Main components

Node

Computation | Function

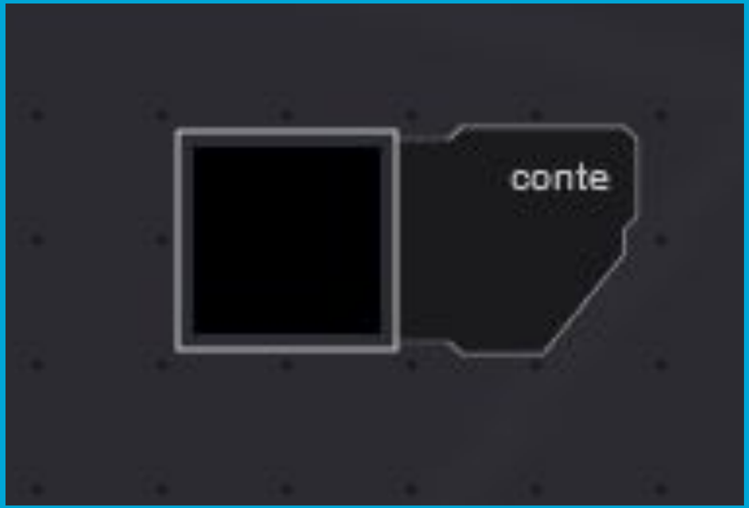


Cables

Variable | edges

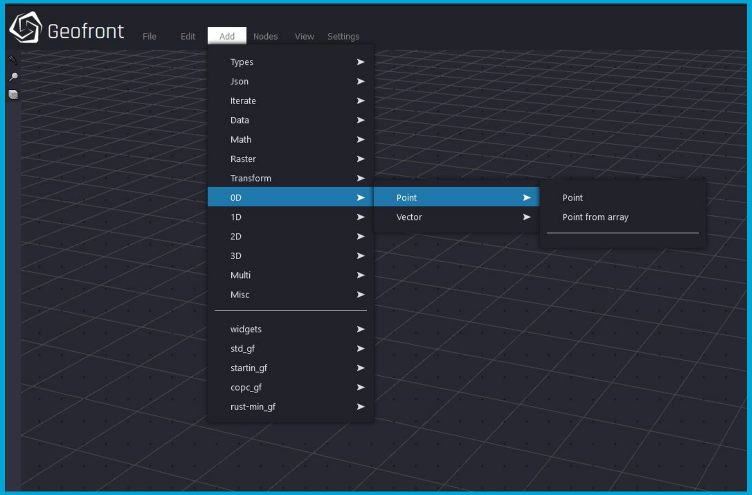
Widget

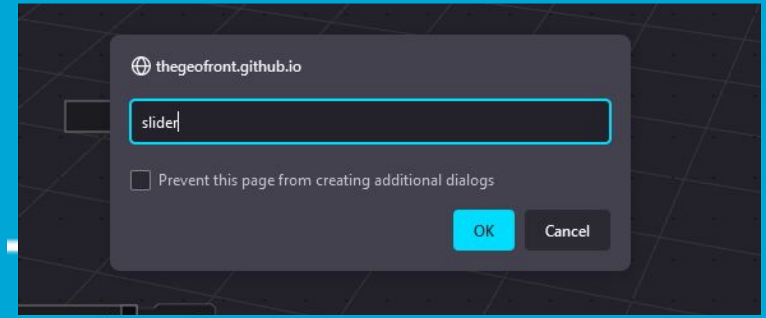
GUI | Input Output



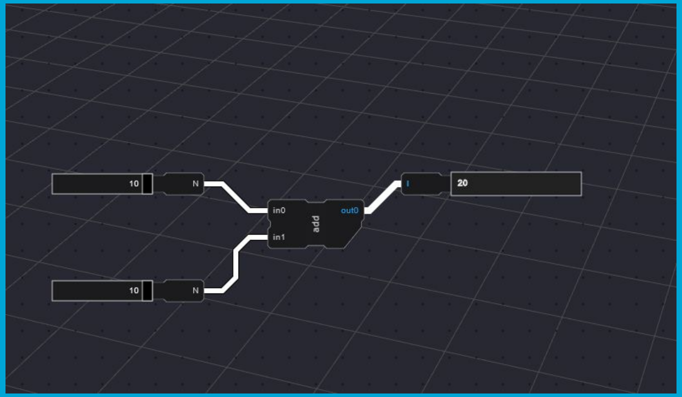
Workflow

1. Add a node or widget from ‘add’ dropdown or fuzzy finder

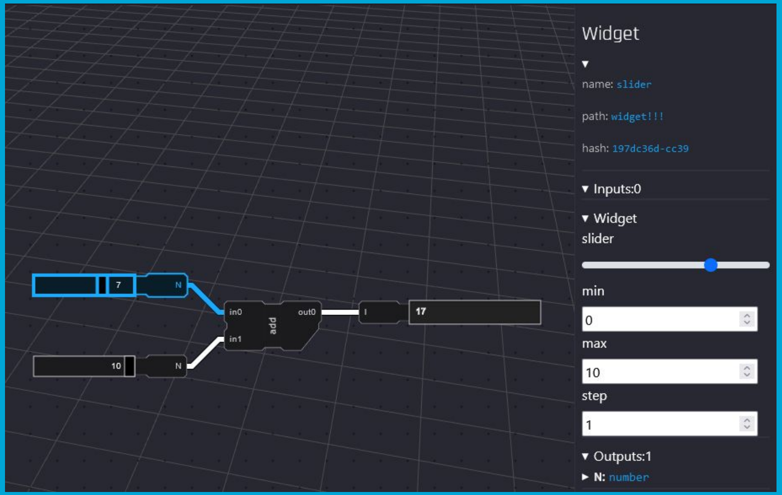




2. Connect nodes by dragging input to output sockets, to form graphs

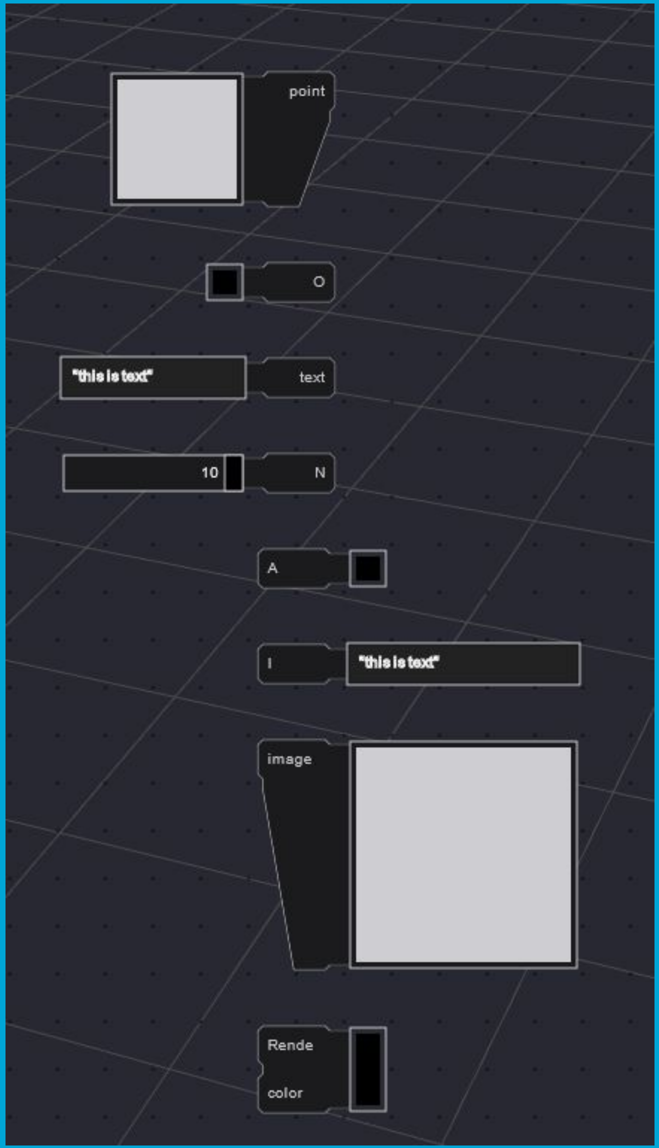


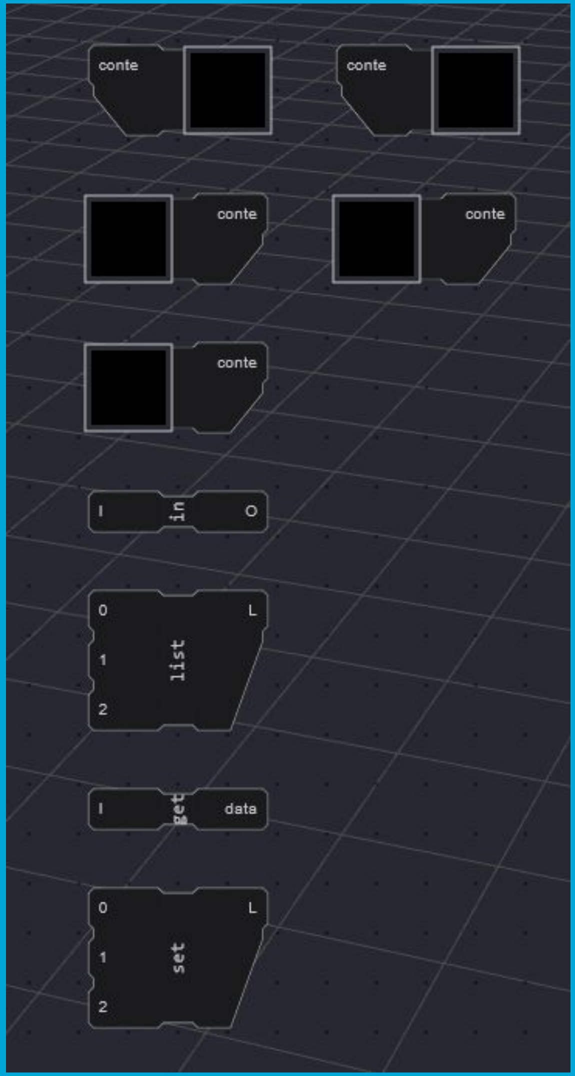
3. To perform calculations, manipulate the input widgets using the canvas GUI, or a side menu



Widgets: Composable GUI

- "Applet"
- Boolean input
- Text field
- Number slider
- Boolean output
- Text output
- Image output
- Renderer



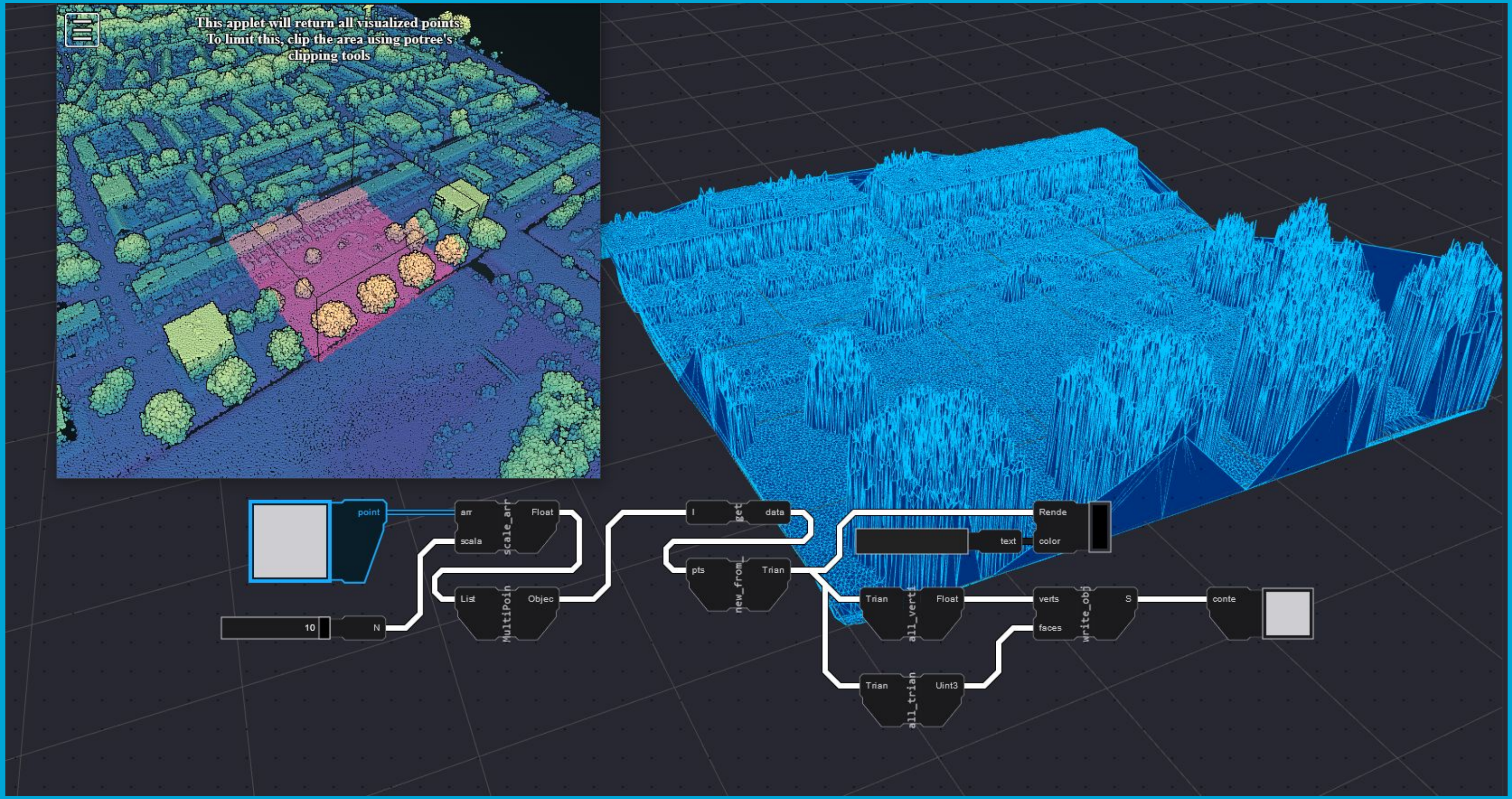


- File save as Blob | String
- File load as Blob | String
- File fetch as Blob | String
- Print to console
- Create list
- Get all properties from object
- Create object from properties

TU Delft

Applet widget: sub-application support

Use output of one application, as input for Geofront

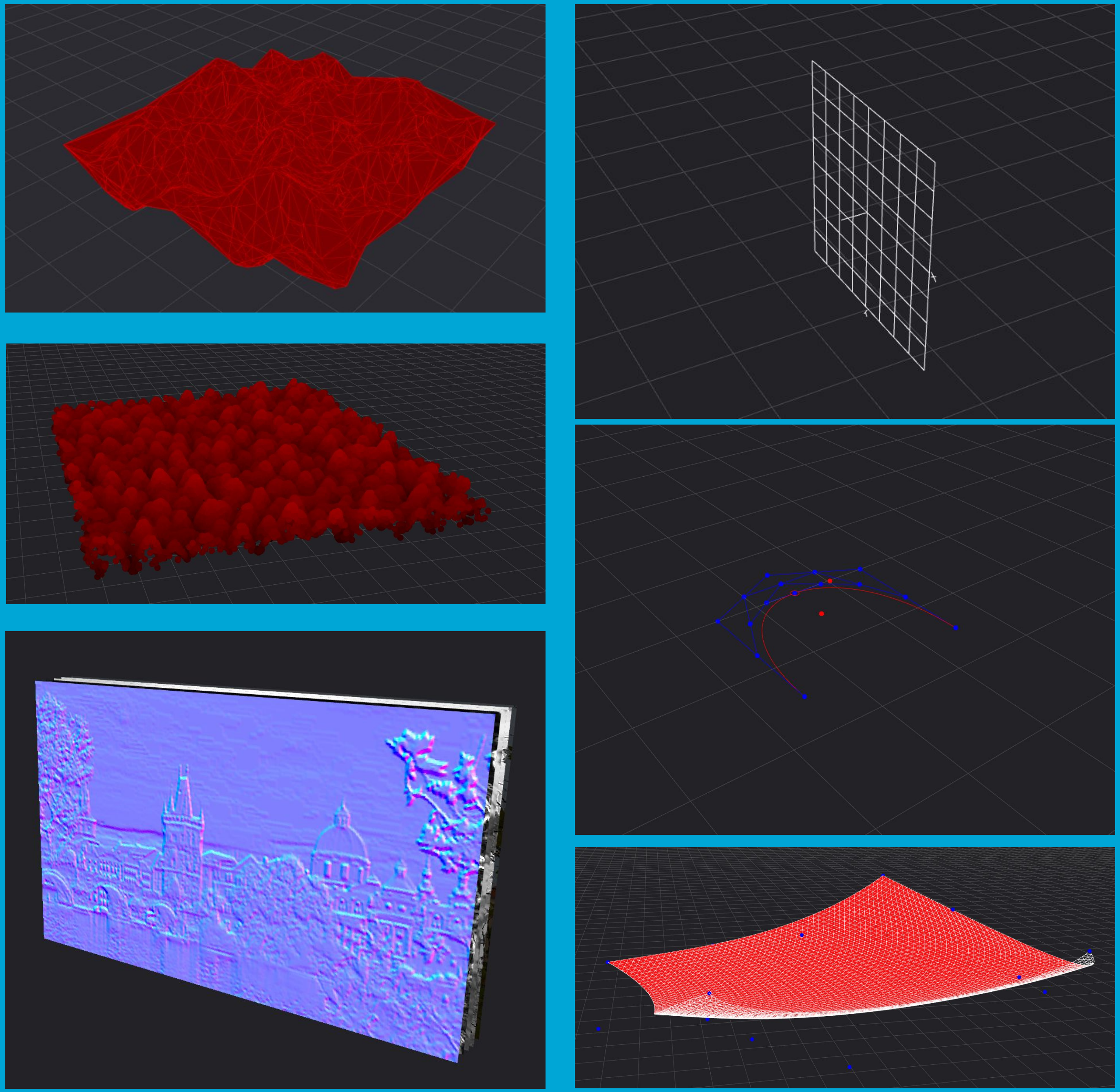


Visualization

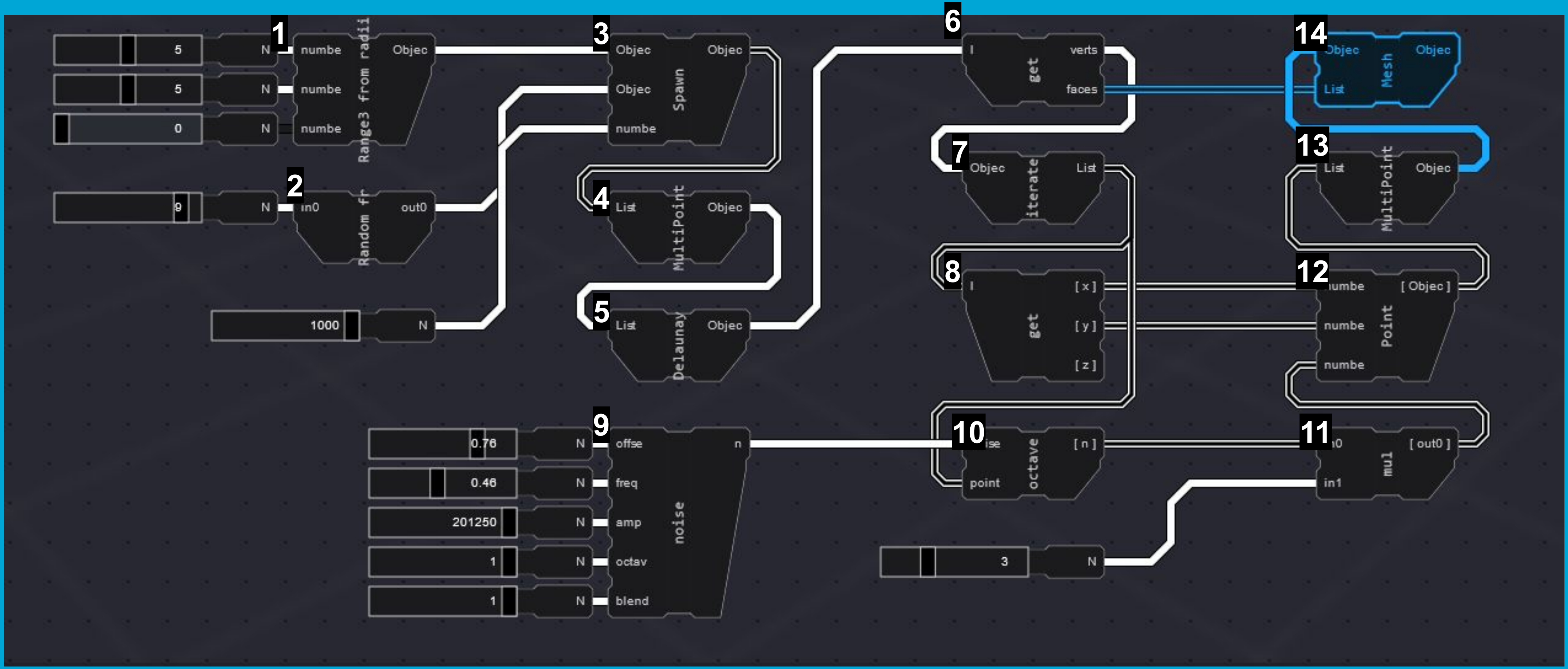
- Custom WebGL implementation

Support for:

- Mesh
- Pointcloud
- Textures (images)
- Plane
- Bezier curve
- Bezier surface



Calculation → Dependency sorting (kahn's algorithm)



TU Delft

Usage 1: Basic interaction

https://thegeofront.github.io/presentation/videos/geofront-1.mp4

TU Delft

Usage 2: Basic composition & data inspection

https://thegeofront.github.io/presentation/videos/geofront-2.mp4

TU Delft

Usage 3: A larger setup & parametrization

https://thegeofront.github.io/presentation/videos/geofront-3.mp4

TU Delft

Usage 4: Geodata input → Obj output

https://thegeofront.github.io/presentation/videos/geofront-4.mp4

TU Delft

Implementation: results

+ All major requirements able to be implemented on the web.
+ Does provide application composability

- Limited STD
- Types not interoperable
- Limited performance

Geofront: Feature comparison

Unique combination

	Grasshopper	*Blender*	*Mobius*	*Geoflow*	*Geofront*
Plugin support	**Yes**	No*	No	**Yes**	**Yes**
Plugin language	C#	-	-	C++	Rust/Js/Ts**
Plugin types	Partially	No	No	Unknown	**Yes**
Headless runtime	No	No	No	**Yes**	No
Web based	No	No	**Yes**	No	**Yes**
Base GIS Nodes	No	No	**Yes**	**Yes**	No
GUI nodes	**Yes**	**Yes**	No	No	**Yes**

2. Library Plugin System implementation

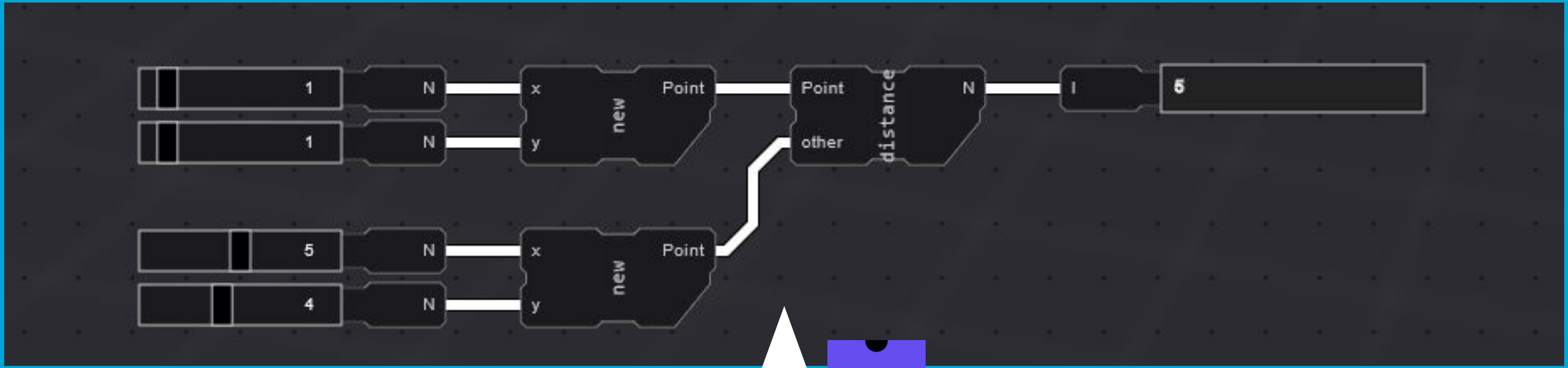
TU Delft

Plugin System: Implementation

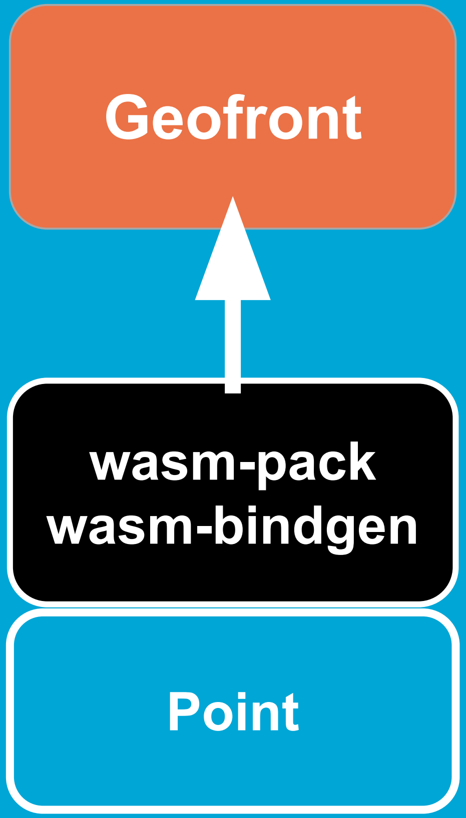
Automated extraction of:

- A list of all functions present in the library
- A list of all custom types (structs / classes) present in the library
- Per function:
    - A list of all input parameters, name and type
    - An output type

Plugin System: Results




```rust
#[wasm_bindgen]
pub struct Point {
    x: f32,
    y: f32,
}

#[wasm_bindgen]
impl Point {

    pub fn new(x: f32, y: f32) -> Self {
        Self { x, y }
    }

    pub fn distance(&self, other: &Self) -> f32 {
        ((self.x - other.x).powi(2) + (self.y - other.y).powi(2)).powf(0.5)
    }
}
```



Plugin System: Comparison

```
namespace MyPlugin
{
    public class AdderNode : GH_Component
    {
        public ComponentNodeFromString()
          : base("Add Integers",
            "Add",
            "This component adds two integer values",
            "My Plugin",
            "My Plugin Category")
        {
        }

        protected override void RegisterInputParams(GH_Component.GH_InputParamManager pManager)
        {
            pManager.AddIntegerParameter("a", "value A", GH_ParamAccess.item);

            pManager.AddIntegerParameter("b", "value B", GH_ParamAccess.item);

        }

        protected override void RegisterOutputParams(GH_Component.GH_OutputParamManager pManager)
        {
            pManager.AddIntegerParameter("R", "result", GH_ParamAccess.item);
        }

        protected override void SolveInstance(IGH_DataAccess DA)
        {
            int a;
            int b;
            DA.GetData(0, ref a);
            DA.GetData(1, ref b);
            int c = a + b;
            DA.SetData(0, c);
        }

        public override Guid ComponentGuid
        {
            get { return new Guid("197d2ec4-c3b1-47ed-8355-6af3b7612f01"); }
        }
    }
}
```

```
class AddNode : public Node
{
public:
  using Node::Node;

  void init()
  {
    add_input("a", typeid(int));
    add_input("b", typeid(int));
    add_output("result", typeid(int));
  }

  std::string info()
  {
    std::string s;
    if (output("result").has_data())
      s = std::to_string(output("result").get<int>());
    return s;
  }

  void process()
  {
    auto in1 = input("a").get<int>();
    auto in2 = input("b").get<int>();
    std::this_thread::sleep_for(std::chrono::microseconds(200));
    output("result").set(int(in1 + in2));
  }
};
```

Figure 55: Geoflow plugin

```
#[wasm_bindgen]
fn add(a: i32, b: i32) -> i32 {
    a + b
}
```

Figure 56: Geofront plugin

Plugin System: Tests

C++ → emscripten → WebAssembly

lib.cpp M

plugins > cpp-min-gf > src > lib.cpp

```cpp
// quick_example.cpp
#include <emscripten/bind.h>
#include <cmath>

using namespace emscripten;

float add(float left, float right) {
    return left + right;
}

class Point {
public:
    double x;
    double y;

    Point(double x, double y) :
        x(x),
        y(y) {}

    double distance(Point& other) {
        return std::pow(
            std::pow(x - other.x, 2) + std::pow(y - other.y, 2),
            0.5);
    }
};

EMSCRIPTEN_BINDINGS(cpp_min) {
    function("add", &add);
    class_<Point>("Point")
        .constructor<double, double>()
        .function("distance", &Point::distance)
        .property("x", &Point::x)
        .property("y", &Point::y);
}
```

Rust → wasm-pack → WebAssembly

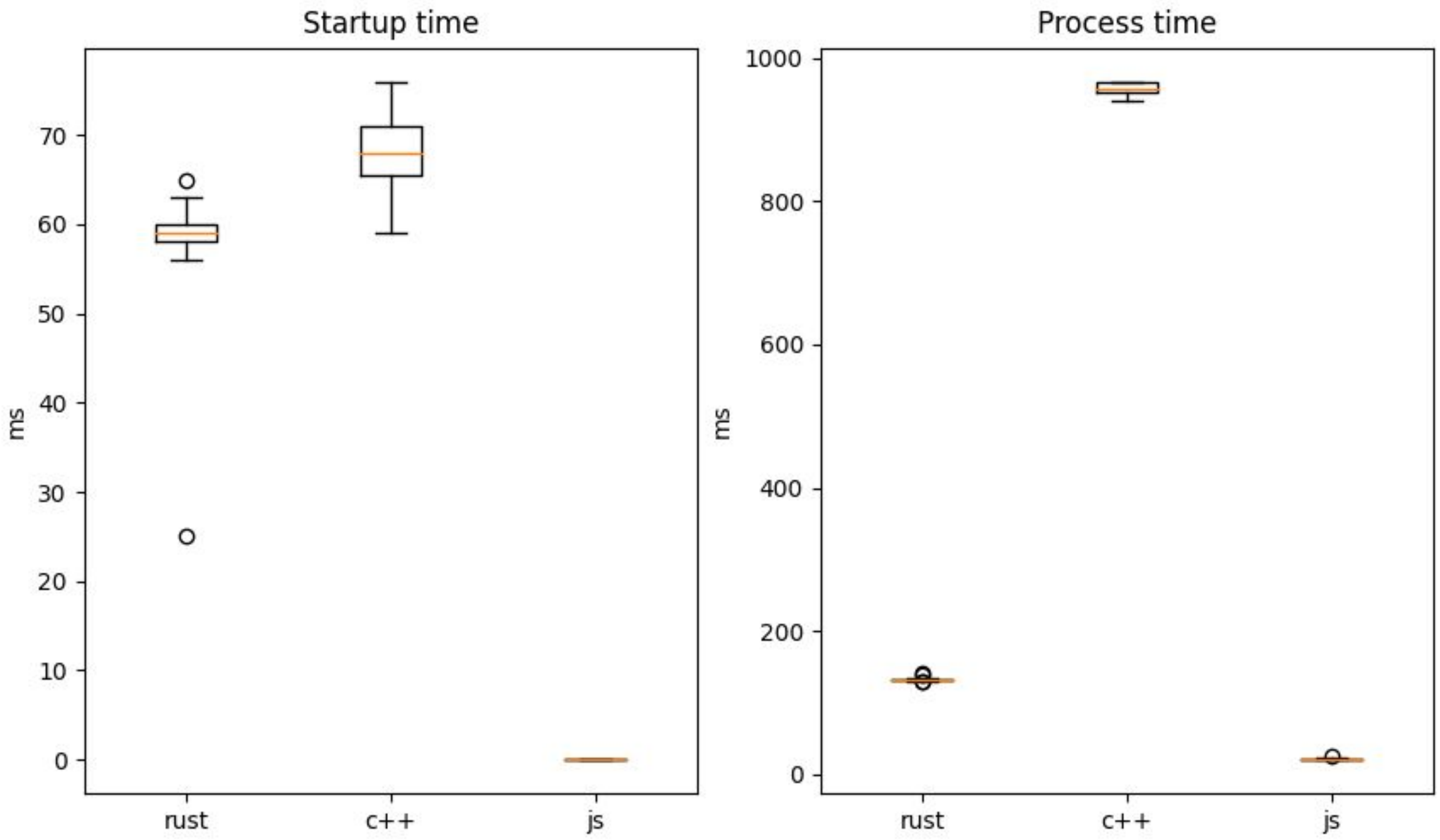
lib.rs

plugins > rust-min-gf > src > lib.rs > {} impl Point

```rust
use wasm_bindgen::prelude::*;

#[wasm_bindgen]
pub fn add(left: usize, right: usize) -> usize {
    left + right
}

#[wasm_bindgen]
pub struct Point {
    x: f32,
    y: f32,
}

#[wasm_bindgen]
impl Point {

    pub fn new(x: f32, y: f32) -> Self {
        Self { x, y }
    }

    pub fn distance(&self, other: &Self) -> f32 {
        ((self.x - other.x).powi(2) + (self.y - other.y).powi(2)).powf(0.5)
    }
}
```

Plugin System: Tests

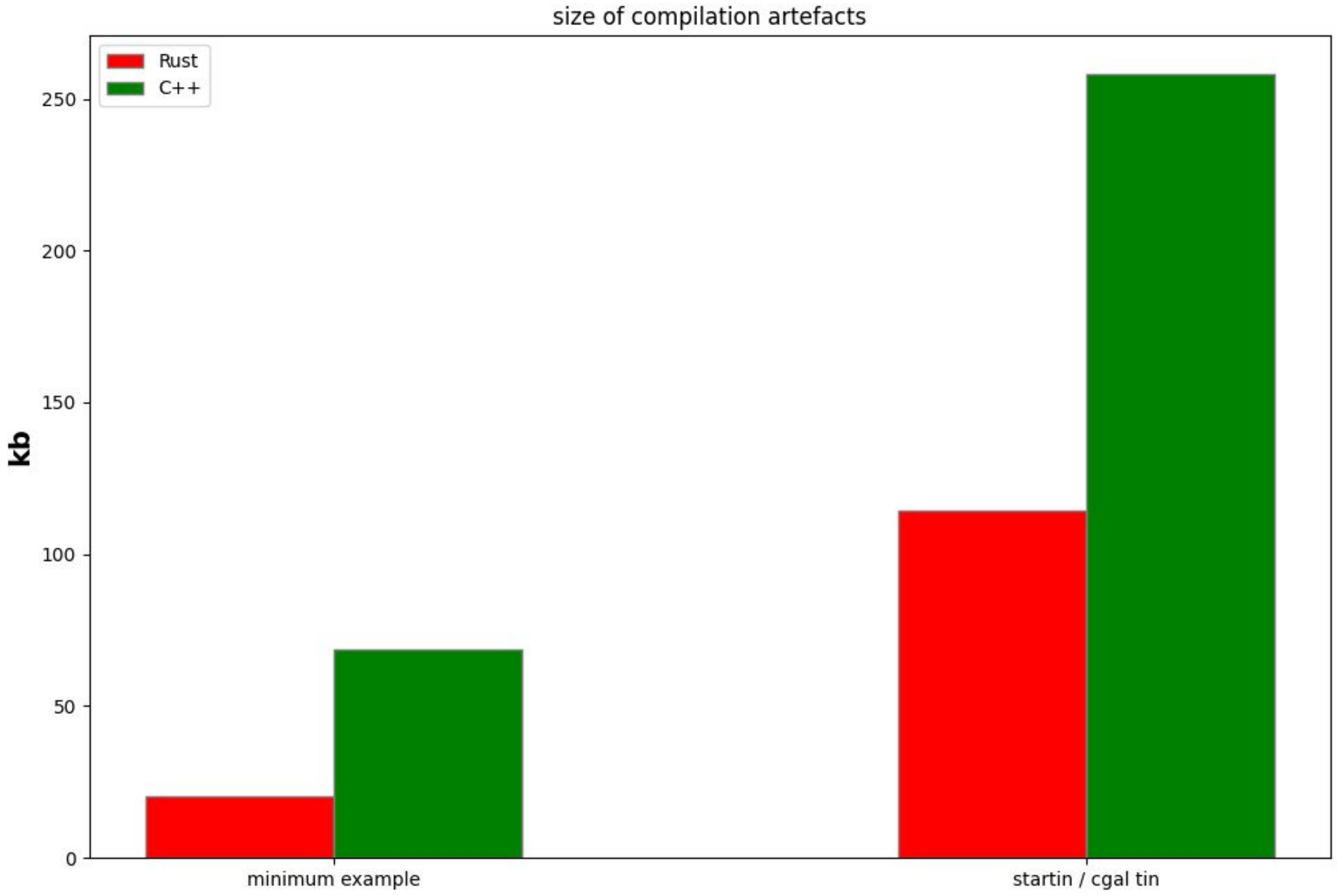
Interfacing the C++ binary from JavaScript was around **six** times as slow compared to the rust equivalent.



TU Delft

Plugin System: Tests

the C / C++ emscripten compiler produced a binary which requires more than **three** times the size of the same functionality compiled with Rusts wasm-pack.



TU Delft

Plugin System: Test Results

Rust

Worked almost immediately for almost any library

+ Expressive bindings allow complex data types to be exchanged in a simple manner.
- *Still some runtime overhead due to wrappers*

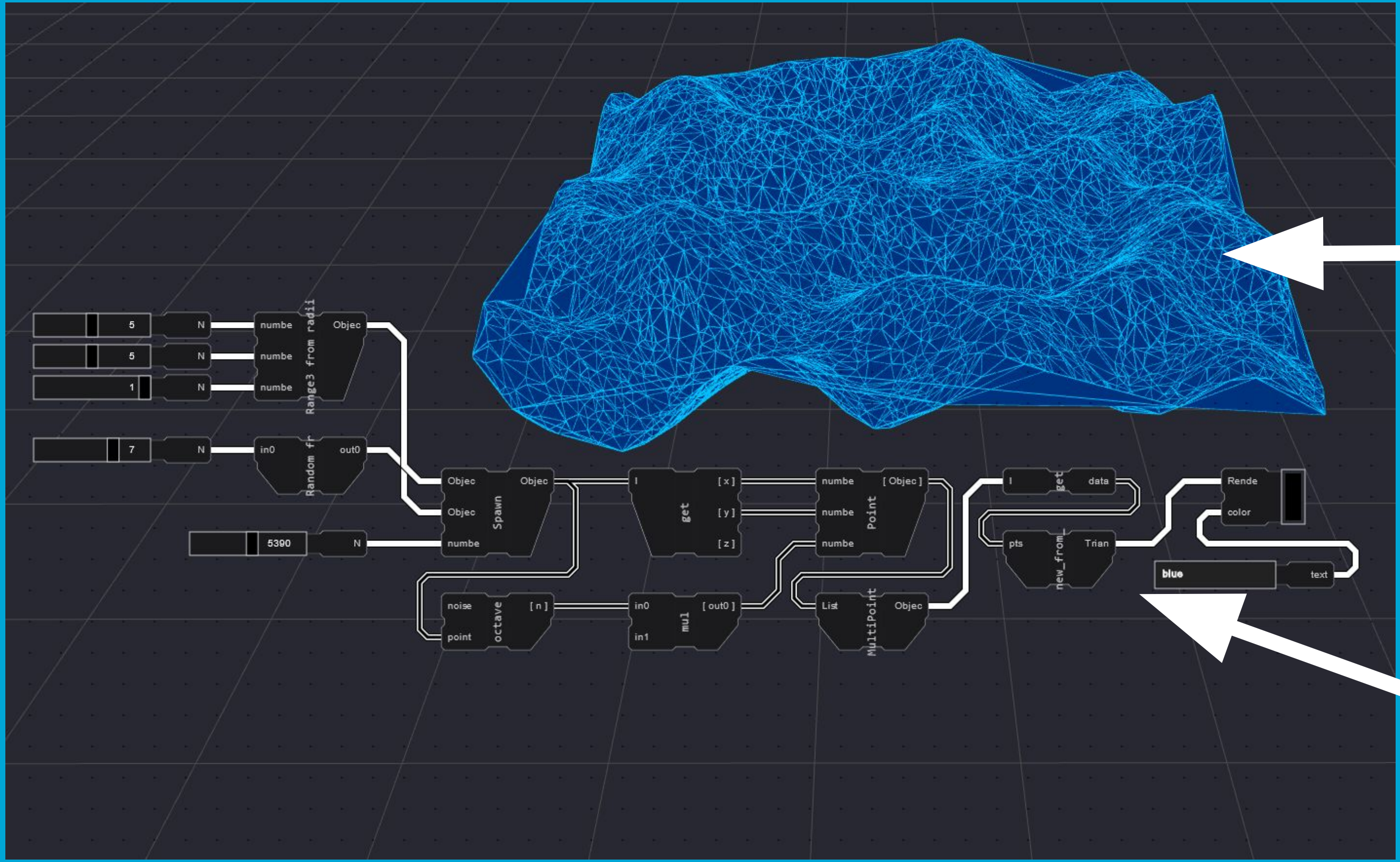
C++

Multiple workarounds eventually allowed some parts of CGAL to be run in geofront, if included in the source code

- *Requires many workarounds*
- *More wrapper overhead than rust*
- *Larger binaries than rust*
- *Sub-optimal support for bindings*
  + Interface Types will most likely be added in the future to emscripten

TU Delft

Plugin System: Tests: startin

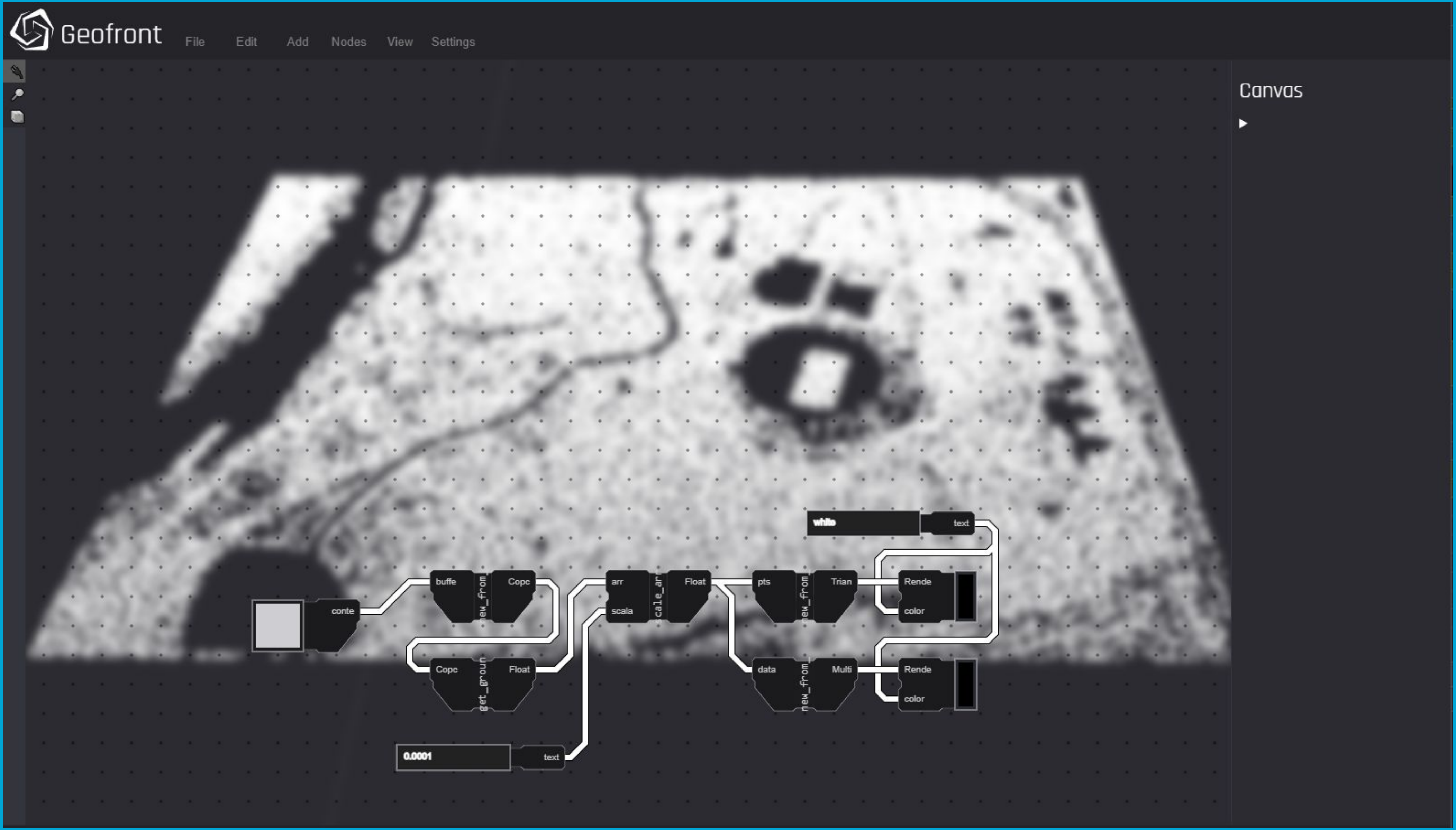



```rust
// impl Renderable for Triangulation
#[wasm_bindgen]
impl Triangulation {

    pub fn gf_has_trait_renderable() -> bool {
        true
    }

    pub fn gf_get_shader_type() -> GeoShaderType {
        GeoShaderType::Mesh
    }

    pub fn gf_get_buffers(&self) -> JsValue {
        let buffer = MeshBuffer {
            verts: self.all_vertices(),
            cells: self.all_triangles(),
        };
        serde_wasm_bindgen::to_value(&buffer).unwrap()
    }
}
```

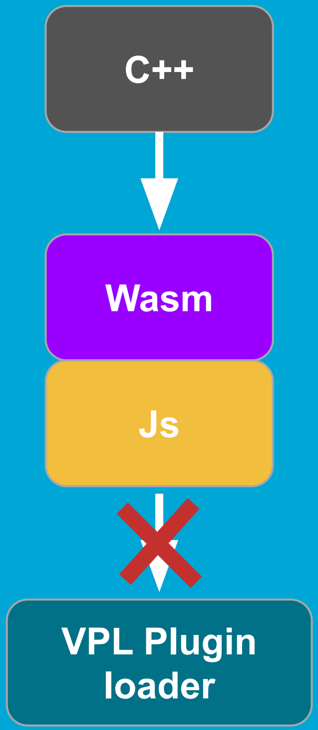
```rust
#[wasm_bindgen]
pub struct Triangulation {
    dt: startin::Triangulation,
}

#[wasm_bindgen]
impl Triangulation {

    pub fn new_from_vec(pts: Vec<f64>) -> Triangulation {
        let mut tri = Triangulation::new();
        tri.insert(pts);
        tri
    }

    pub fn new() -> Triangulation {
        let dt = startin::Triangulation::new();
        Triangulation { dt }
    }
```

TU Delft

Rust Library: copc-rs (Point cloud loader)



C++ Library: CGAL



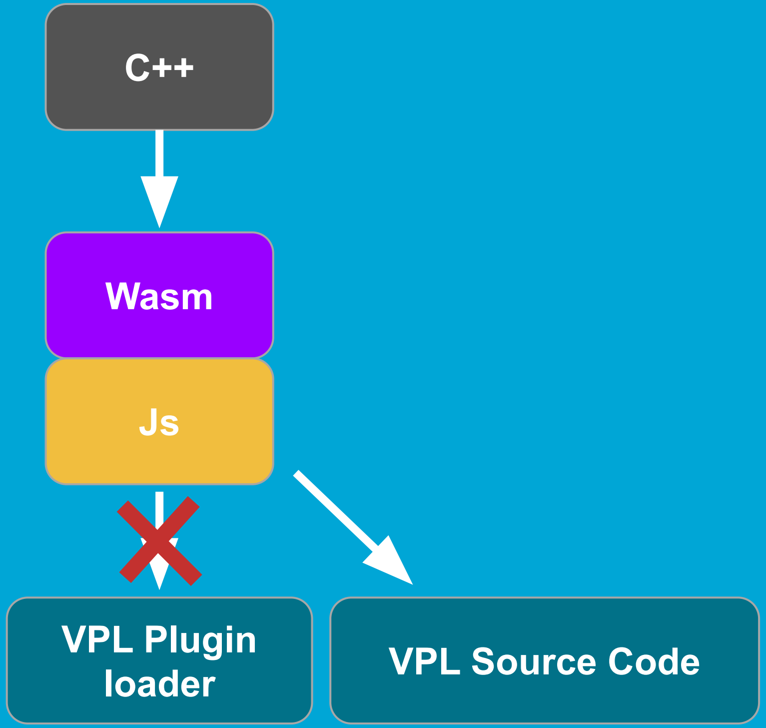
lib.cpp M

plugins > cpp-min-gf > src > lib.cpp

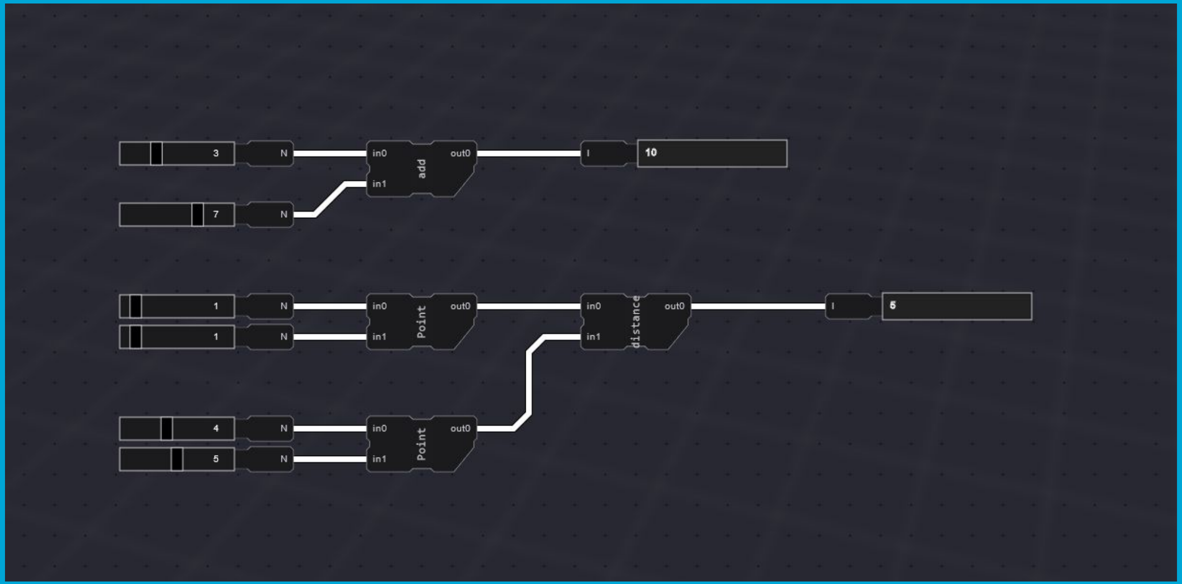
```cpp
// quick_example.cpp
#include <emscripten/bind.h>
#include <cmath>

using namespace emscripten;

float add(float left, float right) {
    return left + right;
}

class Point {
public:
    double x;
    double y;

    Point(double x, double y) :
        x(x),
        y(y) {}

    double distance(Point& other) {
        return std::pow(
            std::pow(x - other.x, 2) + std::pow(y - other.y, 2),
            0.5);
    }
};

EMSCRIPTEN_BINDINGS(cpp_min) {
    function("add", &add);
    class_<Point>("Point")
        .constructor<double, double>()
        .function("distance", &Point::distance)
        .property("x", &Point::x)
        .property("y", &Point::y);
}
```

TU Delft

C++ Library: CGAL

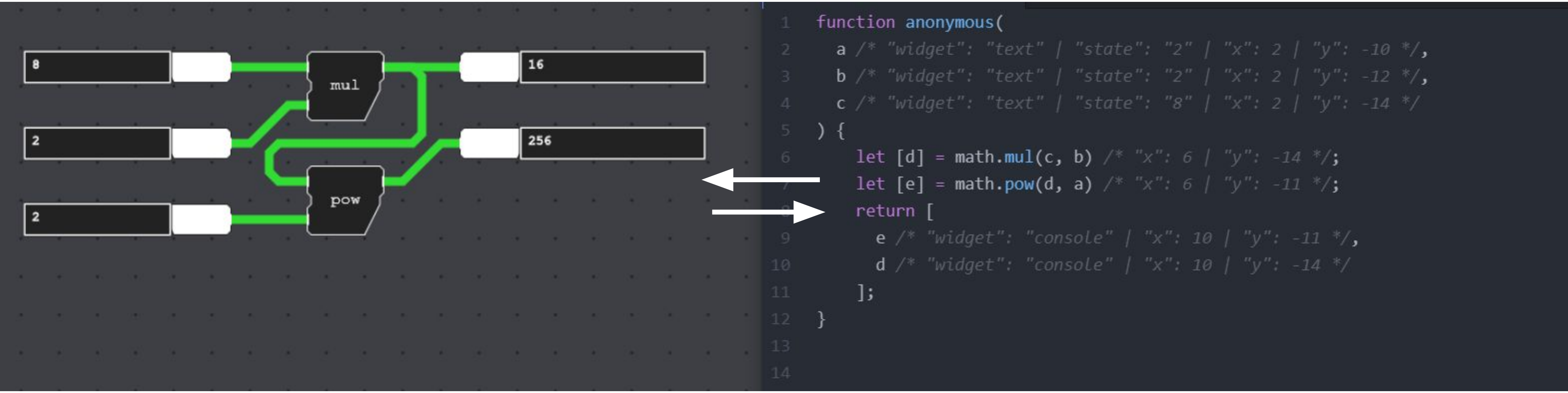


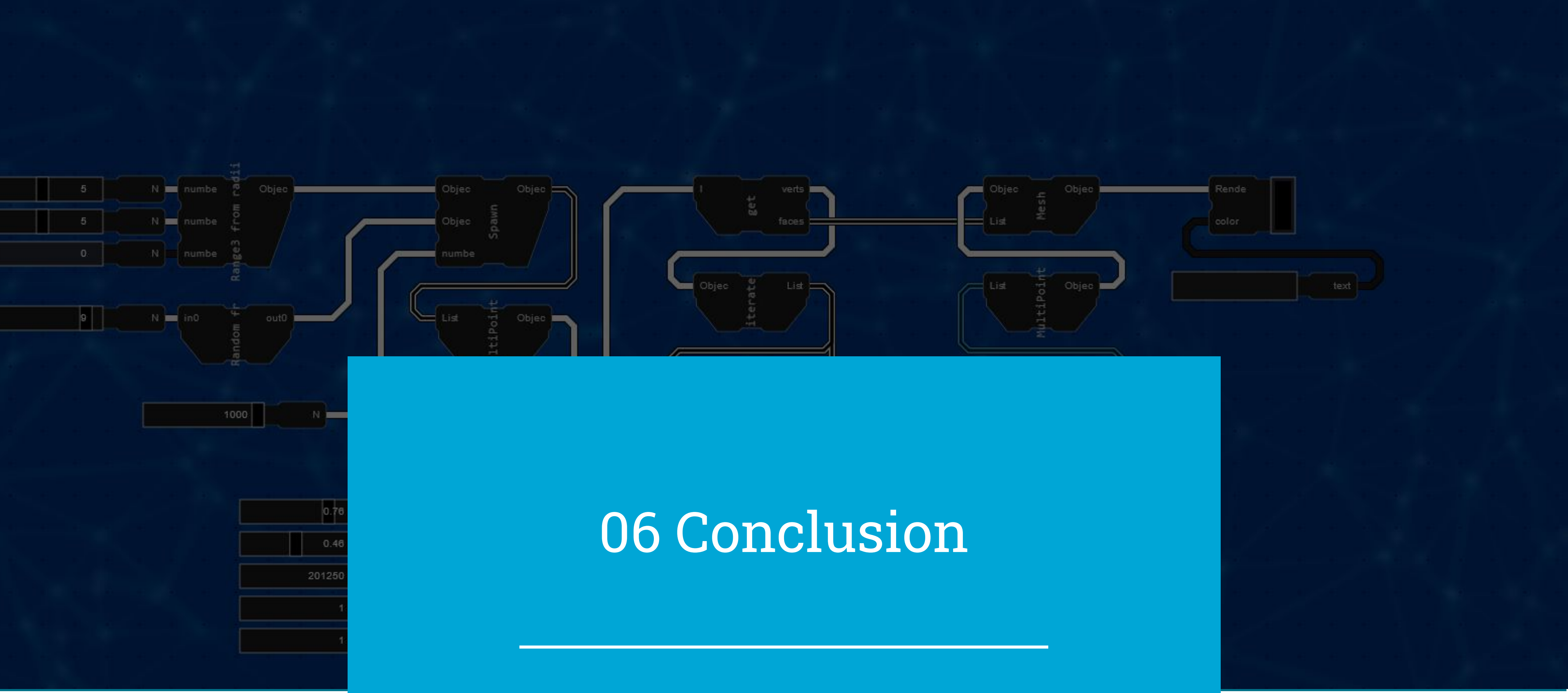

```cpp
lib.cpp M ×
plugins > cpp-min-gf > src > lib.cpp
// quick_example.cpp
#include <emscripten/bind.h>
#include <cmath>

using namespace emscripten;

float add(float left, float right) {
    return left + right;
}

class Point {
public:
    double x;
    double y;

    Point(double x, double y) :
        x(x),
        y(y) {}

    double distance(Point& other) {
        return std::pow(
            std::pow(x - other.x, 2) + std::pow(y - other.y, 2),
            0.5);
    }
};

EMSCRIPTEN_BINDINGS(cpp_min) {
    function("add", &add);
    class_<Point>("Point")
        .constructor<double, double>()
        .function("distance", &Point::distance)
        .property("x", &Point::x)
        .property("y", &Point::y);
}
```



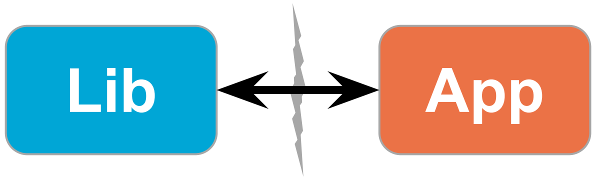
Plugin System: Zero cost abstraction runtime

Currently Incomplete, but promising





06 Conclusion



sub Q: library & application divide

- To what extent does this method intent to address the **discrepancies** between **software applications and libraries**, as described by Elliott (2007)?

Does it succeed in doing so?

1. Libraries cannot be directly used, end users are dependent on in-between applications →

+ VPL acts as “a custom GUI for any library”
+ Only dependent on Wasm-bindings
- Exception: C++

2. Applications are not further composable →

+ VPL: Use tools in a composable manner
+ Potree demo: further composable web applications

3. Capabilities get lost in in-between steps →

+ Plugin system: Minimum in-between steps
+ Wasm-bindings only limiting factor

A: All aspects were able to be addressed to a certain extent.

Main research question:

Q: **Is a web based VPL a viable method for directly accessing native GIS libraries with a composable interface?**

Yes

- Provides solutions for app / lib divide
- Successfully implemented and combined:
  - no-boilerplate plugin system
  - Composable GUI
  - Web-based

No

- More research is required to proof full feasibility:
  - C++ → Interface Types
  - GUI-less runtime → Scalability

A: **Yes**, but with exceptions

TU Delft

Future work

- Improved VPL model:
  - Improved type support
  - Validated Dataflow VPL
    - Concurrency
  - Compile to WebAssembly
- Deployment & scalability
  - Cloud-based execution
  - “Deploy as application”
- Effects of Rust as replacement for C++ in GIS or any scientific endeavor
  - Less error-prone, improved library management, improved wasm support → distribution

TU Delft

Sources:

- Elliott, C. (2007). Tangible functional programming. International Conference on Functional Programming. http://conal.net/papers/Eros/ Accessed 2022-09-27. Related Talk: https://www.youtube.com/watch?v=faJ8N0giqzw.

- Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A., and Bastien, J. (2017). Bringing the web up to speed with WebAssembly. In Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, pages 185–200, New York, NY, USA. Association for Computing Machinery.

- w3c (2019). World Wide Web Consortium brings a new language to the Web as WebAssembly becomes a W3C Recommendation. https://www.w3.org/2019/12/pressrelease-wasm-rec.html.en

- Clark, L. (2019). WebAssembly Interface Types: Interoperate with All the Things. Mozilla Hacks: the Web developer blog. https://hacks.mozilla.org/2019/08/webassembly-interface-types/

- Kuhail, M. A., Farooq, S., Hammad, R., and Bahja, M. (2021). Characterizing Visual Programming Approaches for End-User Developers: A Systematic Review. IEEE Access, 9:14181–14202.

- Sousa, T. (2012). Dataflow Programming: Concept, Languages and Applications. Unpublished.

TUDelft

Thank you for your attention!