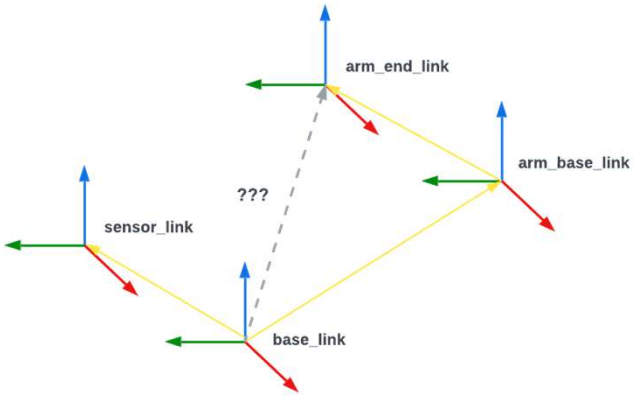
tf2_rs: Bringing tf2 to Rust



Théo Engels[1] - Antionio Paolillo[2] - Ken Hasselmann[1]

[1]Department of Mechanics, Royal Military Academy, 30, Avenue de La Renaissance, 1000 Bruxelles
[2]Software Language Lab, Vrij Universiteit Brussels, 2, Boulevard de La Plaine, 1050 Ixelles

theo.engels@mil.be

RoSE

ICRA™ IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION

2026 VIENNA

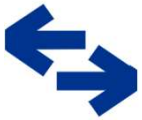
Vienna, June 1, 2026

RMA Royal Military Academy — ROBOTICS & AUTONOMOUS SYSTEMS — VUB SOFTWARE LANGUAGES LAB

Why Rust for robotics?

Safety

Memory safety without garbage collection



Concurrency

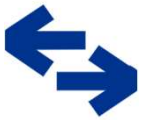
Ownership and borrowing help prevent shared-state mistakes

Performance

Comparable to C/C++

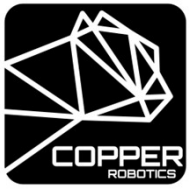
Why Rust for robotics?

Safety



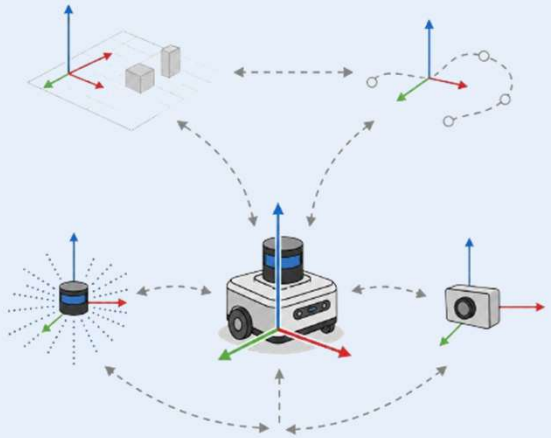
Concurrency

Performance



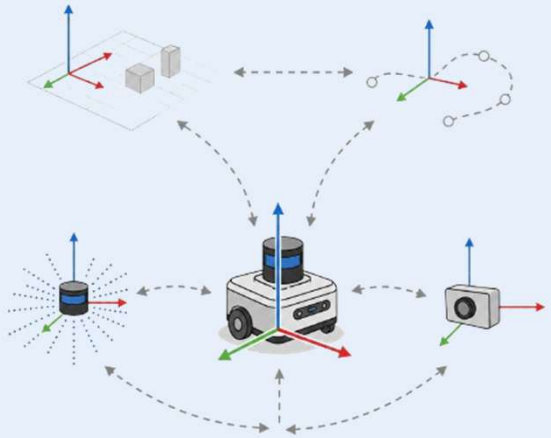
tf2: Where was this frame at the time this data was recorded?

Handles and combines transforms from many frames

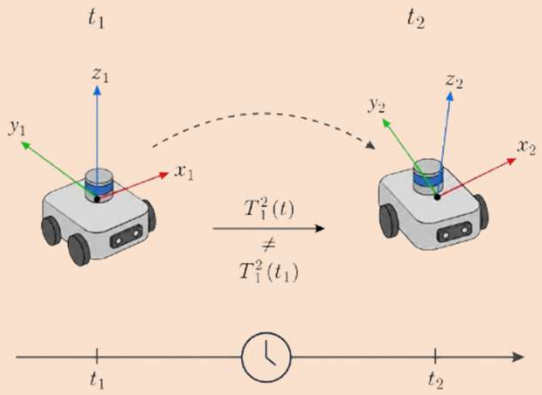


tf2: Where was this frame at the time this data was recorded?

Handles and combines transforms from many frames

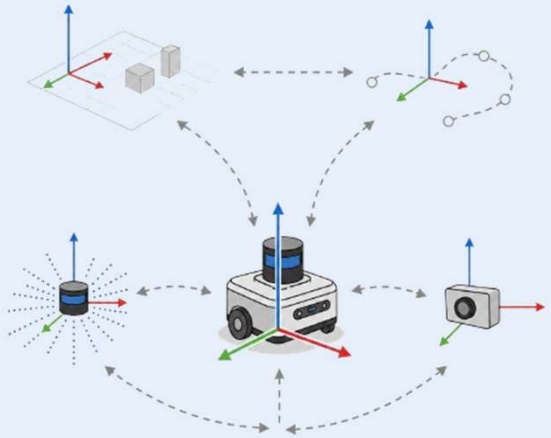


Transforms are time-dependent

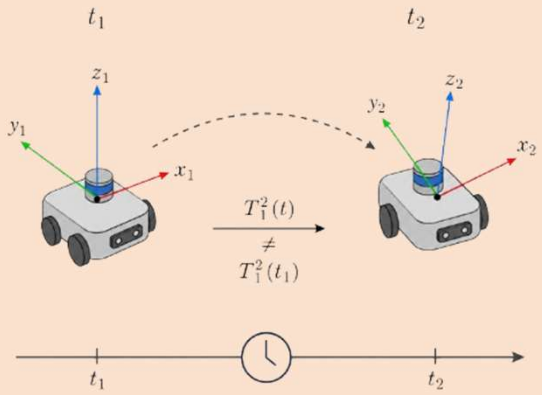


tf2: Where was this frame at the time this data was recorded?

Handles and combines transforms from many frames



Transforms are time-dependent



Transforms needed for perception, localisation, navigation and manipulation

Previous efforts and remaining gaps

Project	Main idea	Drawbacks
balbok0/tf2_rs[1]	Rust native rewrite of tf2	× Risk of semantic/behavior drift; × Hard to maintain; × Inactive.
MaxiMaerz/schiebung[2]	Rust native transform crate	× Not trying to copy tf2; × No behavior parity guarantee; × Unusual API for ROS 2 users; × Risk of mismatch in multi language systems.
deniz-hofmeister/transforms[3]	Rust native, middleware-independent transform crate	

[1]: https://github.com/balbok0/tf2_rs
[2]: https://github.com/MaxiMaerz/schiebung
[3]: https://github.com/deniz-hofmeister/transforms

Requirements for our Rust tf2 interface

1

Preserve upstream tf2 behavior

Bring the same features and behavior as the reference ROS 2 implementation

Requirements for our Rust tf2 interface

1

Preserve upstream tf2 behavior

Bring the same features and behavior as the reference ROS 2 implementation

2

Support the standard ROS 2 workflow

Local buffer, /tf, /tf_static, listener, time-aware lookup

Requirements for our Rust tf2 interface

1

Preserve upstream tf2 behavior

Bring the same features and behavior as the reference ROS 2 implementation

2

Support the standard ROS 2 workflow

Local buffer, /tf, /tf_static, listener, time-aware lookup

3

Keep lookup fast

Transform lookup appears in latency sensitive robotics pipelines

Requirements for our Rust tf2 interface

1

Preserve upstream tf2 behavior

Bring the same features and behavior as the reference ROS 2 implementation

2

Support the standard ROS 2 workflow

Local buffer, /tf and /tf_static listener, time-aware lookup

3

Keep lookup fast

Transform lookup appears in latency sensitive robotics pipelines

4

Avoid time-consuming full rewrite

A complete rewrite would bring high maintenance costs

Requirements for our Rust tf2 interface

1

Preserve upstream tf2 behavior

Bring the same features and behavior as the reference ROS 2 implementation

2

Support the standard ROS 2 workflow

Local buffer, /tf and /tf_static listener, time-aware lookup

3

Keep lookup fast

Transform lookup appears in latency sensitive robotics pipelines

4

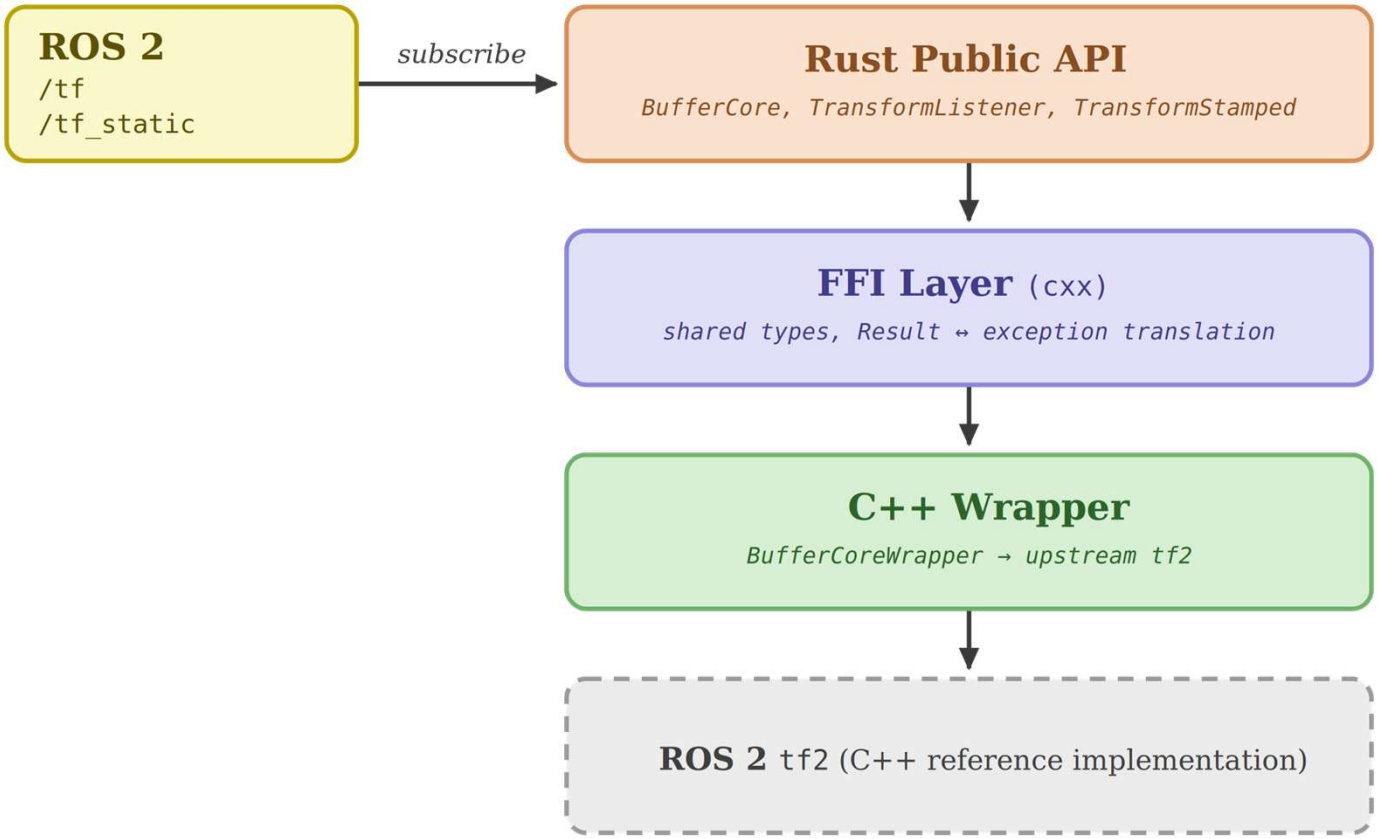
Avoid time-consuming full rewrite

A complete rewrite would bring high maintenance costs

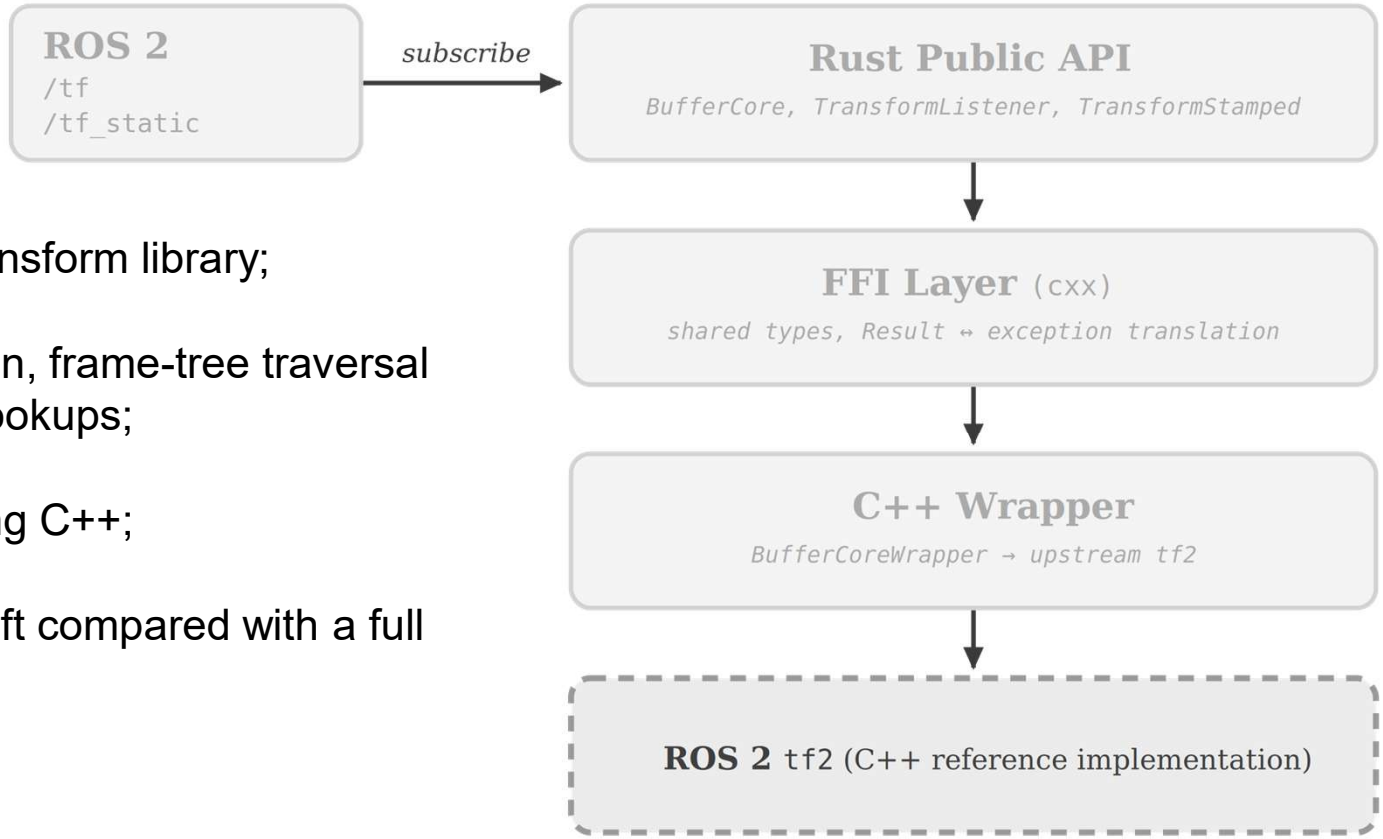
Solution: take inspiration from *tf2_py*[1] → Rust API exposing bindings the C++ tf2

[1]: https://github.com/ros2/geometry2

Architecture: Rust API over upstream ROS 2 tf2



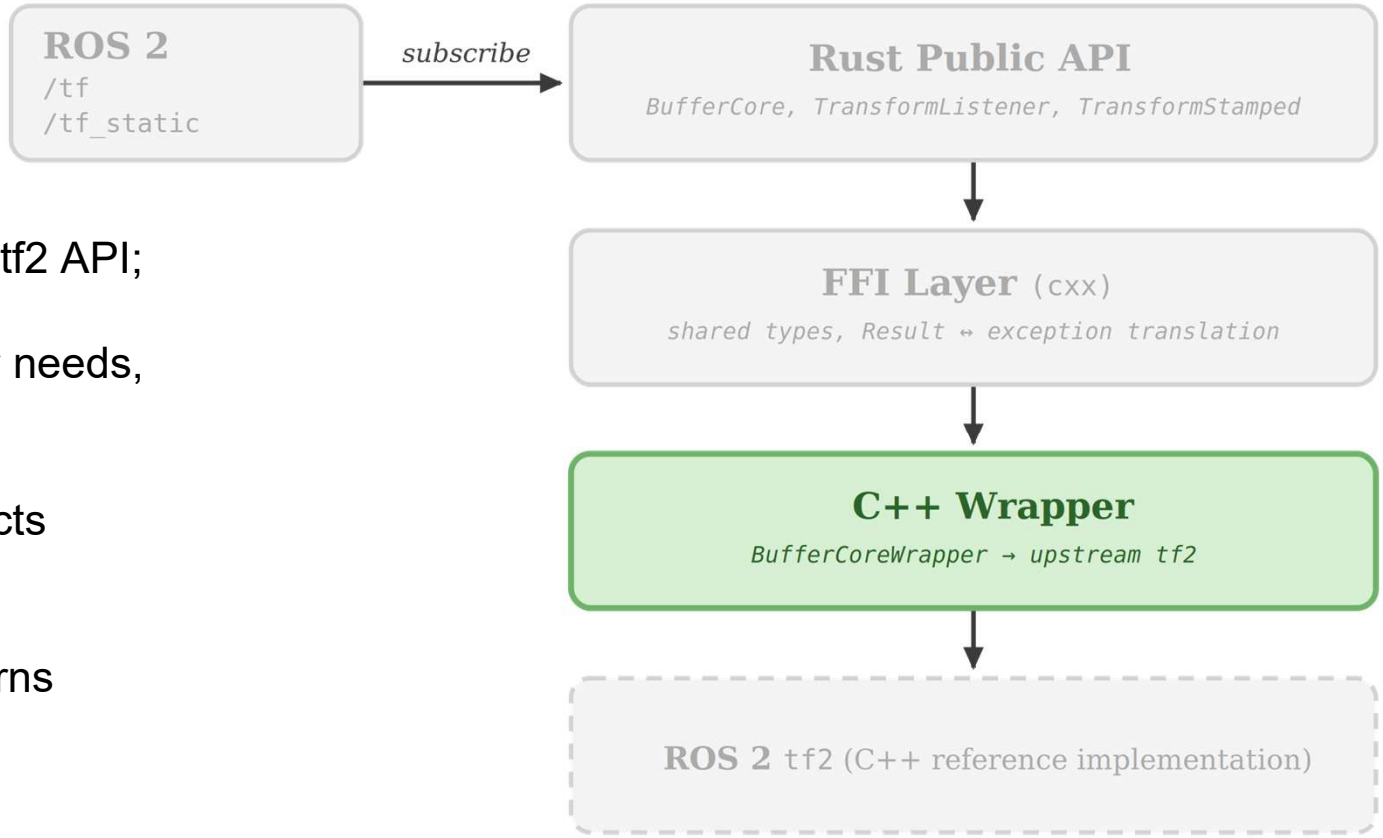
Architecture: Rust API over upstream ROS 2 tf2



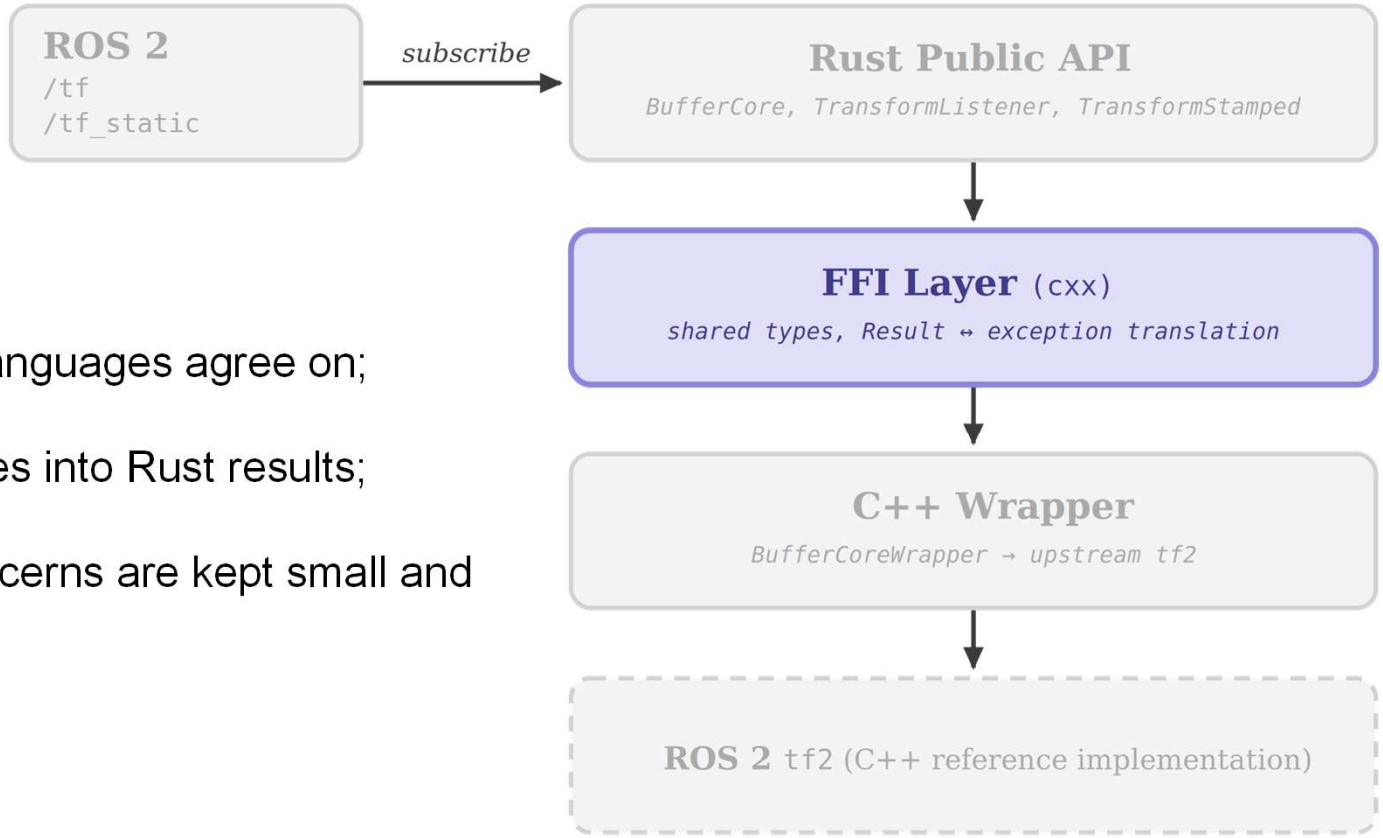
- Canonical, battle-tested C++ transform library;
- Owns the core logic: interpolation, frame-tree traversal and management, time-aware lookups;
- Keeps tf2_rs aligned with existing C++;
- Reduces the risk of semantic drift compared with a full Rust rewrite.

Architecture: Rust API over upstream ROS 2 tf2

- Thin adapter over the upstream tf2 API;
- Exposes only what the FFI layer needs, in binding-friendly signatures;
- Converts between FFI-safe structs and ROS 2 message types;
- Catches tf2 exceptions and returns structured status values.



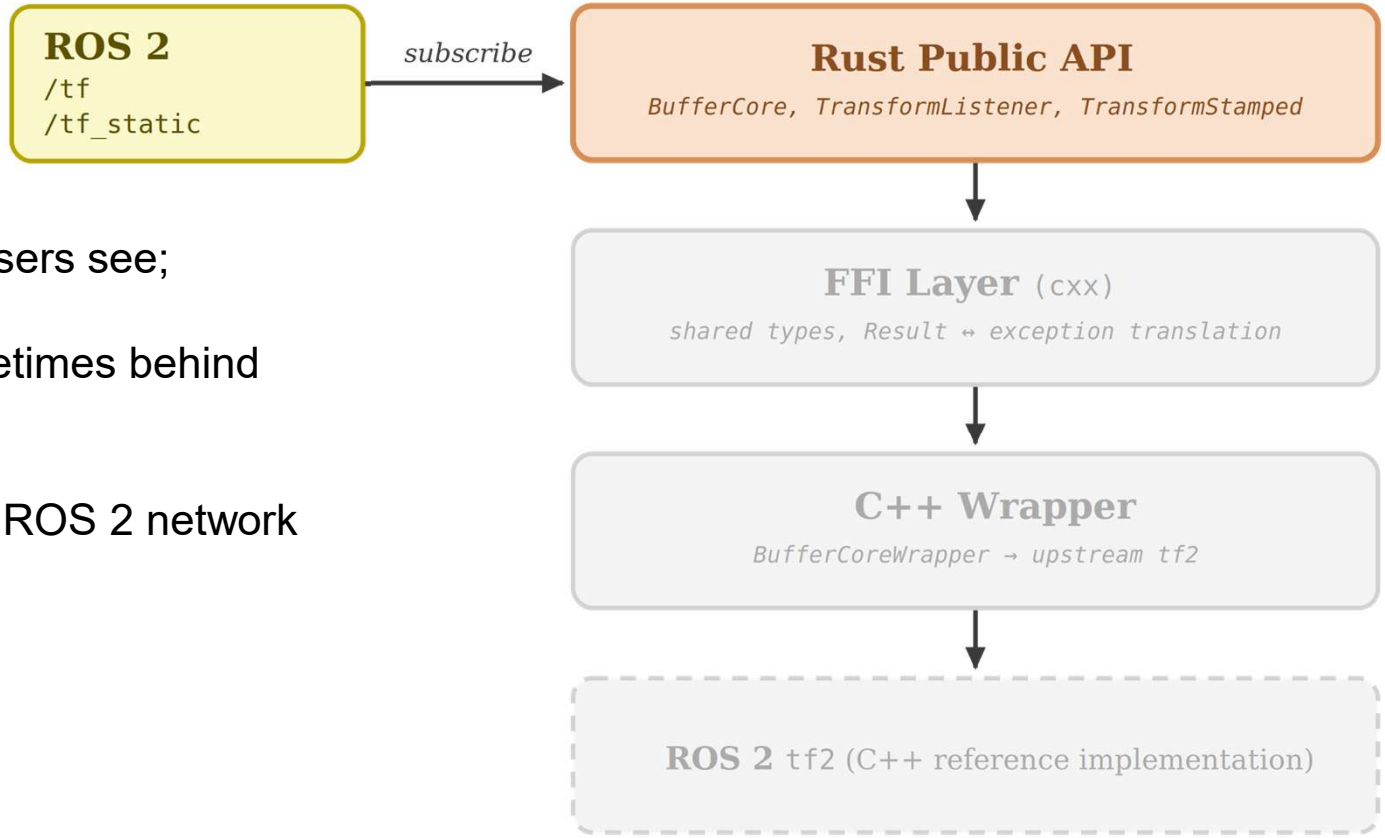
Architecture: Rust API over upstream ROS 2 tf2



- Built with cxx[1];
- Defines the shared types both languages agree on;
- Translates structured error values into Rust results;
- Unsafe and foreign-function concerns are kept small and localized.

[1]: https://cxx.rs/

Architecture: Rust API over upstream ROS 2 tf2



- The safe, idiomatic Rust code users see;
- Hides all unsafe pointers and lifetimes behind ordinary Rust types;
- Handles the interaction with the ROS 2 network trough rclrs.

API and workflow

1 **Create node**

```cpp
Node("tf2_basic_cpp_listener")
```

```rust
executor.create_node("tf2_basic_listener")?
```

API and workflow

1 **Create node**

```
Node("tf2_basic_cpp_listener")
```

```
executor.create_node("tf2_basic_listener")?
```

2 **Create buffer**

```
make_unique<tf2_ros::Buffer>(get_clock())
```

```
let buffer = BufferCore::new(Duration::from_secs(10))
```

API and workflow

1. **Create node**

```
Node("tf2_basic_cpp_listener")
```

```
executor.create_node("tf2_basic_listener")?
```

2. **Create buffer**

```
make_unique<tf2_ros::Buffer>(get_clock())
```

```
let buffer = BufferCore::new(Duration::from_secs(10))
```

3. **Add listener**

```
make_shared<tf2_ros::TransformListener>(*tf_buffer_)
```

```
let listener =
  TransformListener::new(&node, buffer.clone())?
```

API and workflow

1 Create node

```
Node("tf2_basic_cpp_listener")
```

```
executor.create_node("tf2_basic_listener")?
```

2 Create buffer

```
make_unique<tf2_ros::Buffer>(get_clock())
```

```
let buffer = BufferCore::new(Duration::from_secs(10))
```

3 Add listener

```
make_shared<tf2_ros::TransformListener>(*tf_buffer_)
```

```
let listener =
  TransformListener::new(&node, buffer.clone())?
```

4 Subscribe + callback

```
create_subscription<Msg>("pointcloud", 10,
  std::bind(&Node::on_trigger, this, _1));
```

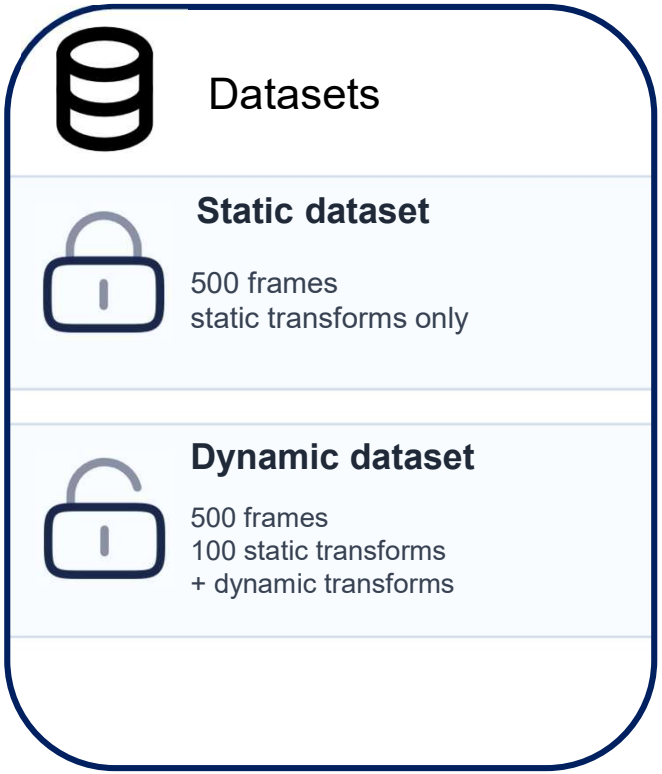
```
node.create_subscription("pointcloud",
  move |msg| Self::on_trigger(msg, &buf))?
```

API and workflow

	C++	Rust
1 **Create node**	`Node("tf2_basic_cpp_listener")`	`executor.create_node("tf2_basic_listener")?`
2 **Create buffer**	`make_unique<tf2_ros::Buffer>(get_clock())`	`let buffer = BufferCore::new(Duration::from_secs(10))`
3 **Add listener**	`make_shared<tf2_ros::TransformListener>(*tf_buffer_)`	`let listener = TransformListener::new(&node, buffer.clone())?`
4 **Subscribe + callback**	`create_subscription<Msg>("pointcloud", 10, std::bind(&Node::on_trigger, this, _1));`	`node.create_subscription("pointcloud", move |msg| Self::on_trigger(msg, &buf))?`

Rust API closely matches the original C++ with idiomatic Rust error handling

Benchmark design: lookup latency test matrix



Benchmark design: lookup latency test matrix

Datasets

Static dataset

500 frames
static transforms only

Dynamic dataset

500 frames
100 static transforms
+ dynamic transforms

Scenarios

Offline

Frames preloaded
into the buffer

Online

Buffer filled in real time
from ROS 2 topics

Benchmark design: lookup latency test matrix

Datasets

Static dataset

500 frames
static transforms only

Dynamic dataset

500 frames
100 static transforms
+ dynamic transforms

Scenarios

Offline

Frames preloaded
into the buffer

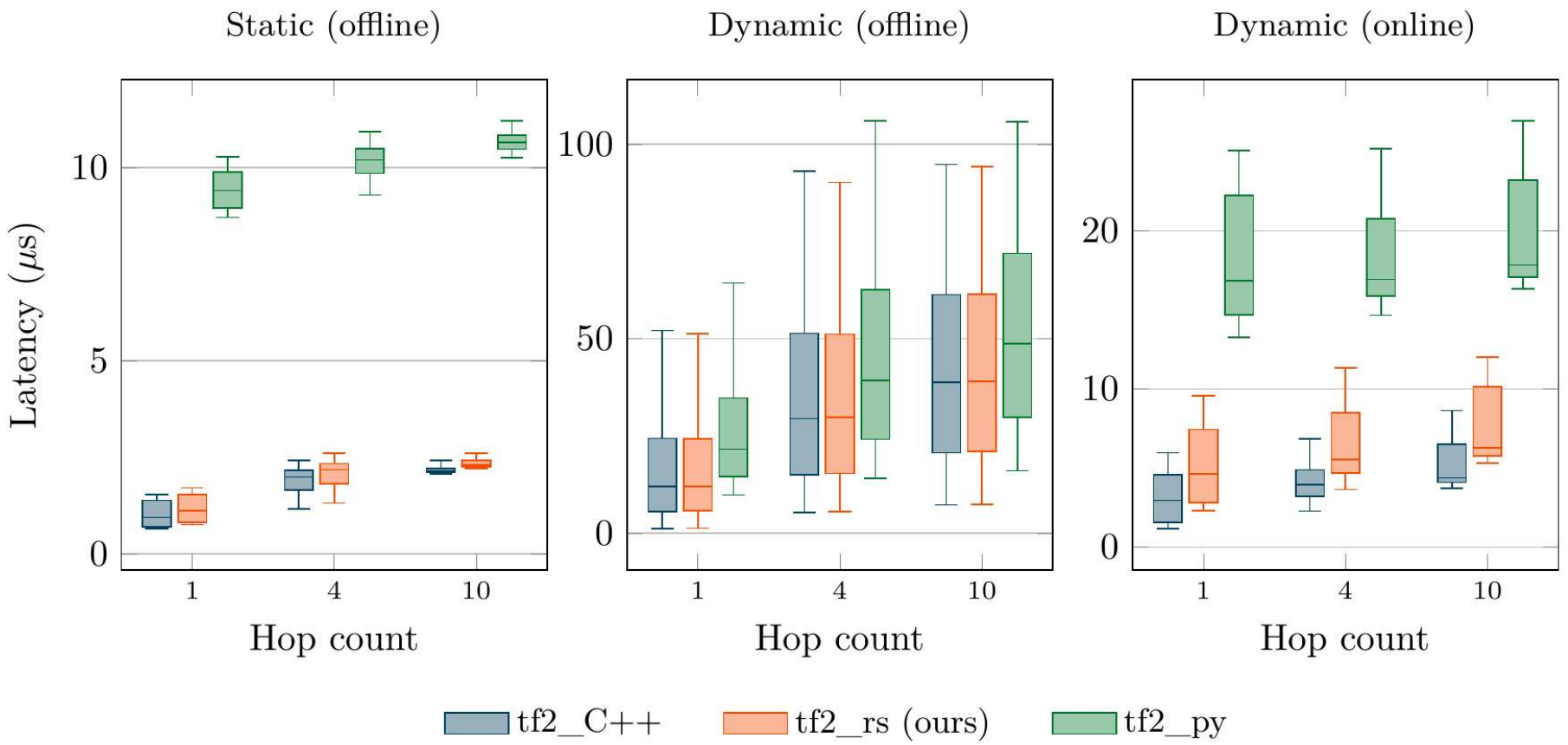
Online

Buffer filled in real time
from ROS 2 topics

Query depth

1	Direct relationship
4	Moderate distance in the transform tree
10	Deep tree traversal needed

Results: tf2_rs closely matches tf2_C++



Limitations

Feature	tf2 (C++)	tf2_py	tf2_rs
Core buffer API	✓	✓	✓
Listener + broadcasters	✓	✓	✓
Advanced lookup API (target_time != source_time)	✓	✓	✗
Typed geometry transforms and transform application	✓	✓	△
Async/wait API	✓	✓	✗
MessageFilter	✓	✗	✗

Legend: ✓ implemented, △ partial, ✗ missing.

Conclusions and next steps

tf2_rs fills a core ROS 2 Rust ecosystem gap by exposing the standard tf2 workflow through idiomatic Rust bindings while preserving upstream semantics and staying close to C++ lookup performance!

And now… what's next?

- Broader transform application helpers
- MessageFilter-style workflows
- Advanced lookup API
- Safety analysis
- Expanded behavior cross tests
- Comparison against transpiled version

Questions

tf2_rs