Indico.UN

A technical overview

{An odyssey of “plugin” development and epic proportions}

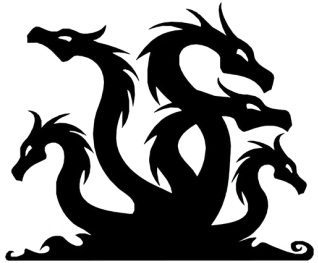
The Odyssey

It's just an Indico plugin, right?


hope
new clients
Indico core
“plugin”

The Odyssey

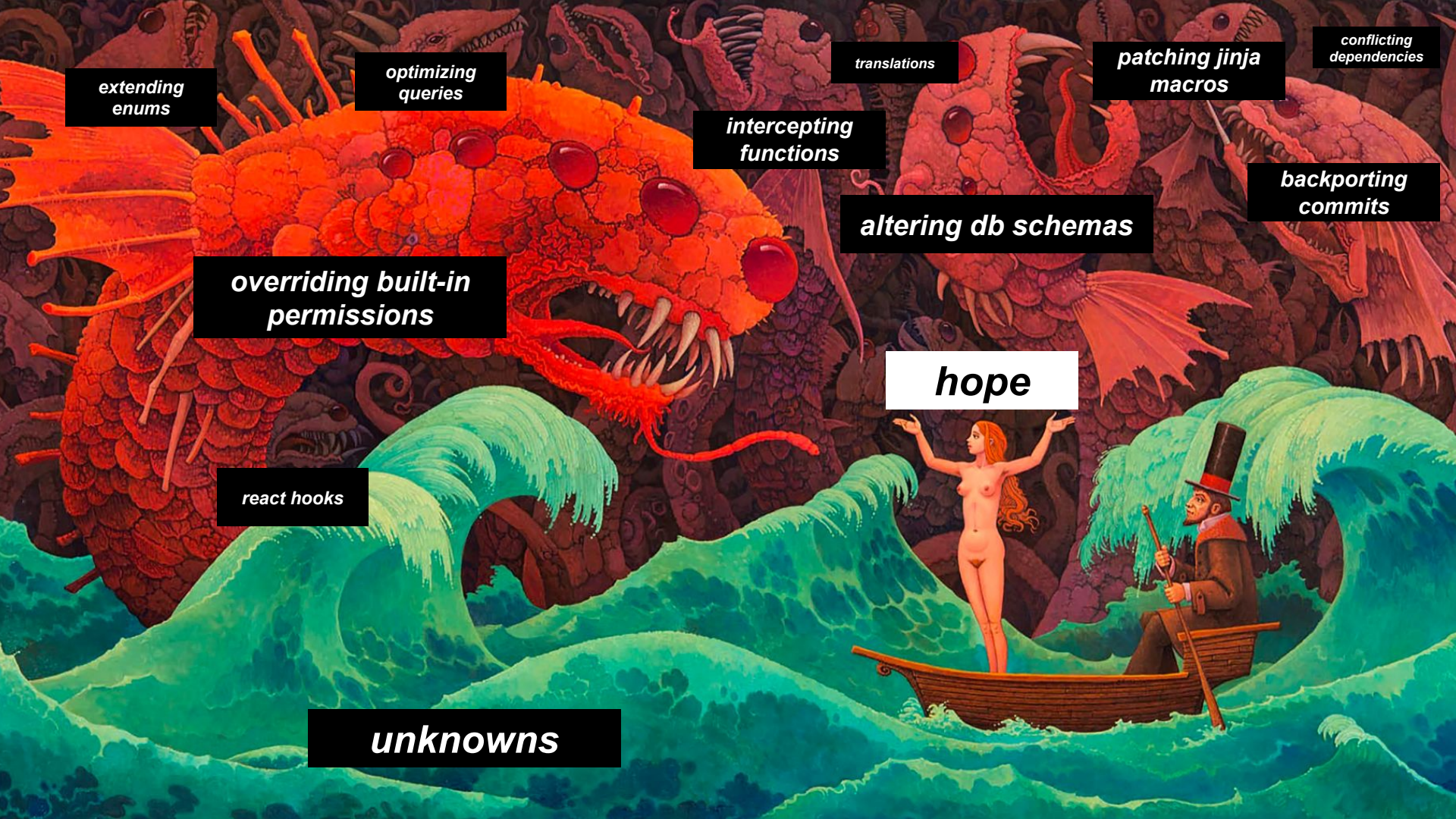
- The “plugin” must serve **multiple duty stations**:
  - Different countries. Different workflows. Often conflicting requirements.
  - Harmonizing them requires profound customizations of Indico core.
  - The number of duty stations keeps growing!



- The “plugin” relies **entirely** on Indico core:
  - We don't have direct control of the majority of the functionality.
  - Indico itself is an evolving project we need to keep up with.
  - There is so much the built-in plugin engine can do for us!

The Sea Creatures

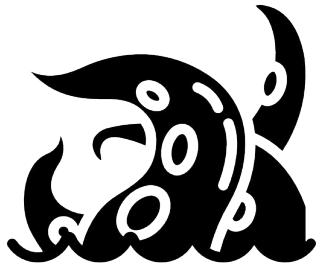
Here be dragons…


extending enums
optimizing queries
translations
patching jinja macros
conflicting dependencies
intercepting functions
backporting commits
altering db schemas
overriding built-in permissions
hope
react hooks
unknowns

The Sea Creatures

- Indico itself is a **complex system** with many tentacles:
  - Flask, SQLAlchemy, WTForms, Marshmallow, Celery in the backend.
  - React, and remnants of jQuery and AngularJS in the frontend.
  - Jinja templates and macros for HTML and email rendering.

- Developing the “plugin” presented **formidable obstacles**:
  - How do we ADD our own features to Indico? (ok)
  - How do we ALTER Indico core functionalities? (hmm…)
  - How do we MAINTAIN our sanity and our project afloat? (errrr!!)



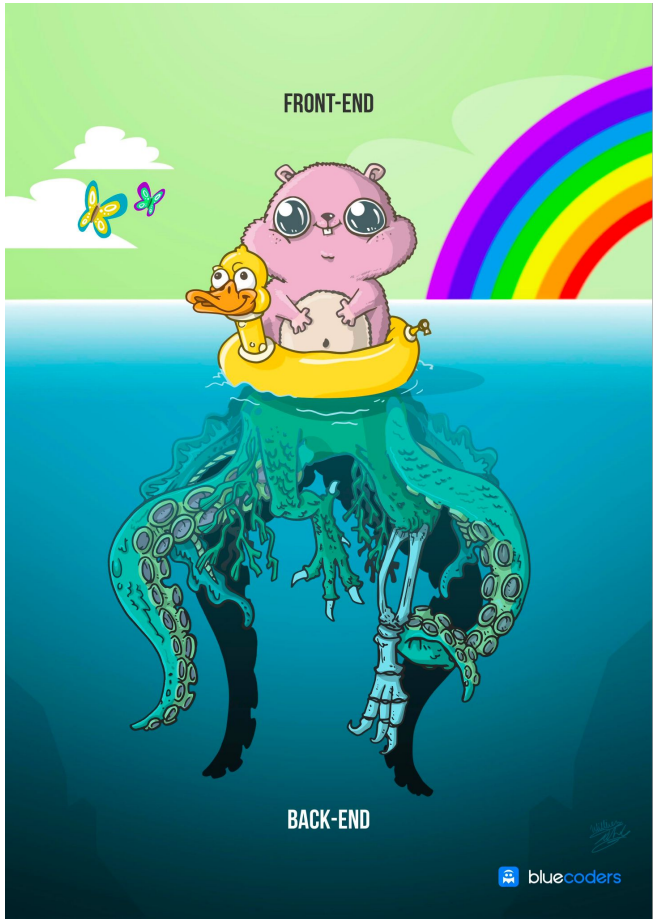
The Vessel

Evolving on its journey to Ithaca

Version 1

Not a plugin

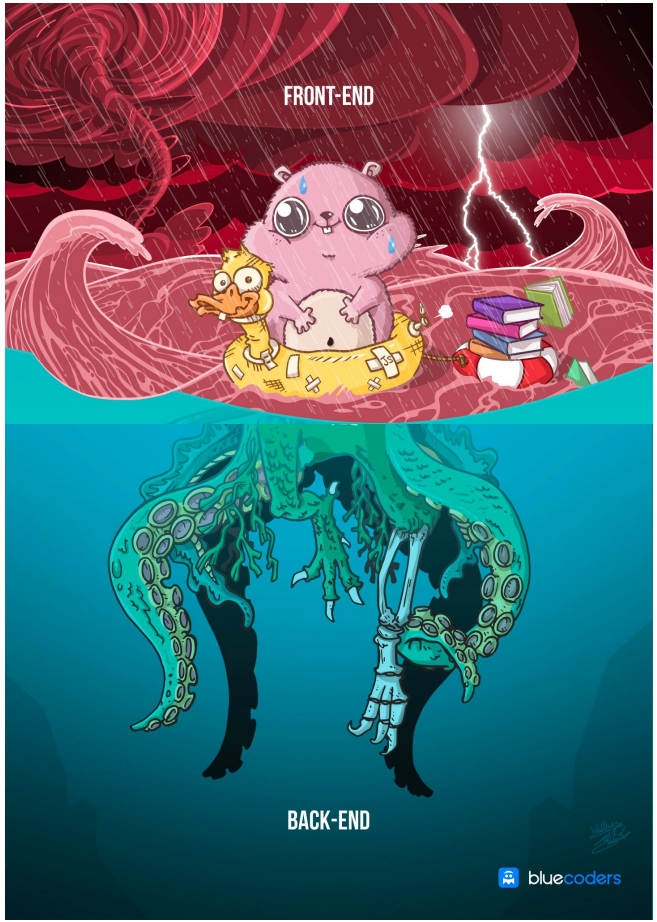
2015 ~ 2022



Version 2

Stuck in Python 2

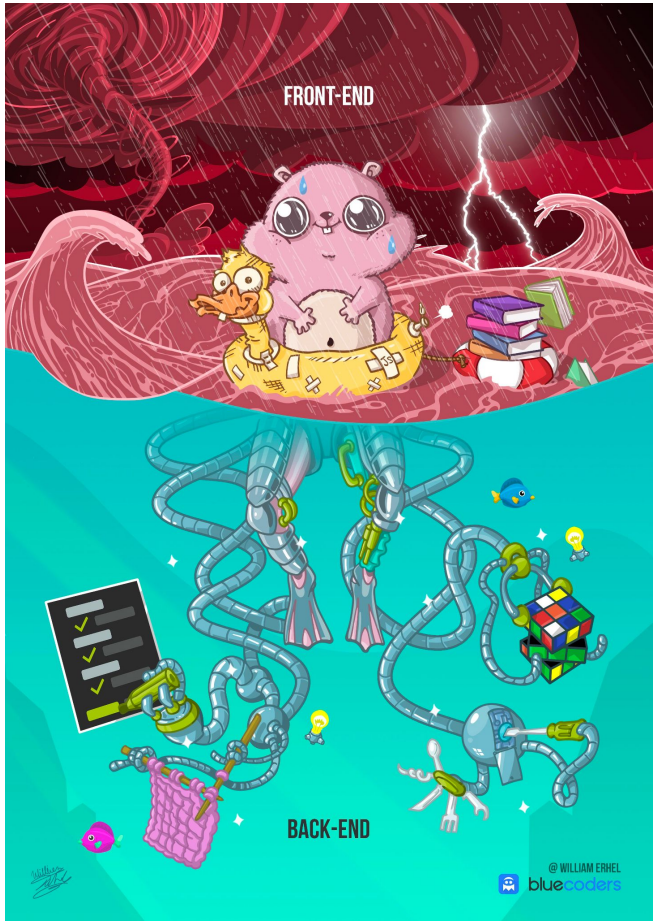
2020 ~ 2023



Version 3

Engineered to last

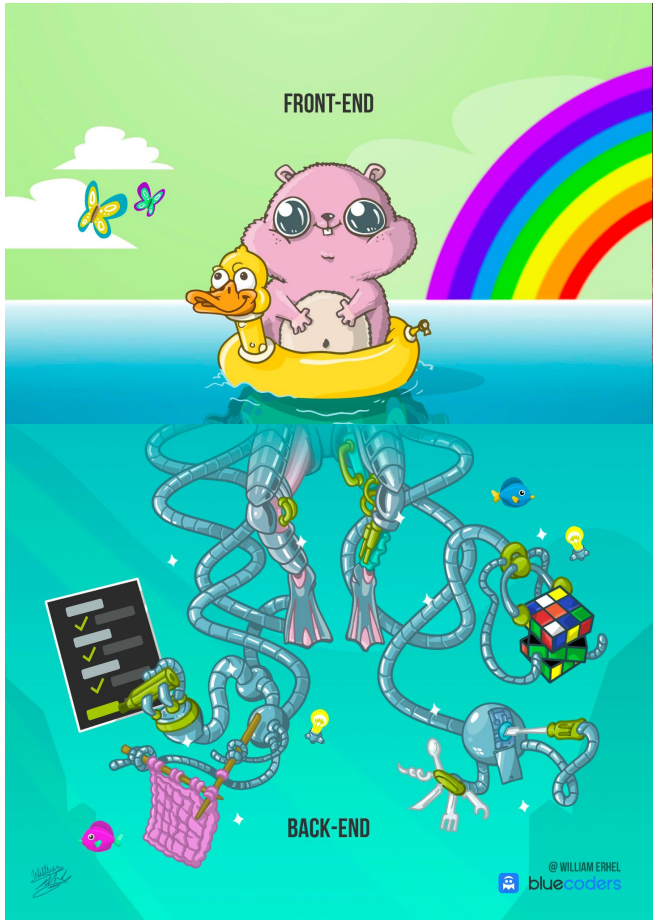
2023 ~ 202X



Version X

Where are you, Ithaca?

202X ~ 20XX



Blueprint of the codebase

	indico-un==3.0.x	plugins
submodules	indico-zoom==3.0.x	
	backports (yikes!)	fork
	indico==3.1.x	

Vessel Maintenance Guide

- Developing the “plugin” requires **fine craftsmanship**:
  - Avoid patching Indico core as much as possible. It's quick but extremely fragile.
  - Stick to idioms, file structure, etc. in Indico core. It will come in handy when…

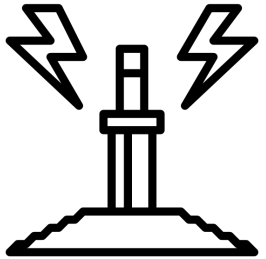
- Contributing **signals and hooks** to Indico core:
  - Use the internal API for extending and overriding built-in functionality.
  - Simple contributions upstream are merged almost instantly!
- Contributing **new functionality** to Indico core:
  - Contribute useful functionality to Indico core whenever possible.
  - This reduces maintenance cost and benefits the ecosystem!

The Weapons

It's dangerous to go alone! Take this.

The Weapons

- Indico core provides powerful built-ins:
  - Python signals
  - Jinja template hooks and blocks
  - The `@make_interceptable` decorator
  - The `Query.signal_query()` method (contributed!)



- Homemade versatile and dangerous utilities:
  - The `@patched` decorator
  - The `@patch_enum` decorator
  - The core Jinja templates pre-processor.

Adding menu items via signal

```
from indico.core import signals

@signals.menu.items.connect_via('admin-sidemenu')
def _extend_admin_menu(sender, **kwargs):
    return SideMenuItem(
        'highlight_events', _("Highlights"),
        url_for('plugin_un.highlight_settings'),
        section='homepage'
    )
```

Rendering additional content in Jinja templates

```
{{ template_hook('category-sidebar', category=category) }}
```

indico

```
@template_hook('category-sidebar')
def _inject_analytical_module_link(category):
    analytical_module_link = UN_CONF.get('analytical_module_link')
    return flask.render_template(
        'un:categories/sidebar_analytics.html', category=category,
        analytical_module_link=analytical_module_link)
```

indico-un

Overriding Jinja blocks

```
{% block management_button %}
    <a class="i-button icon-edit" href="..." title="..."></a>
{% endblock %}
```

indico

```
{% extends '~events/header.html' %}

{% block management_button %}
    {% if can('event.manage', event) %}
        {{ super() }}
    {% endif %}
{% endblock %}
```

indico-un

Injecting parameters to Jinja templates

```
import flask

@flask.before_render_template.connect
def _render_highlighted_categories(sender, template, context, **kwargs):
    if template.name != 'categories/display/root_category.html':
        return
    context['highlighted'] = ...
```

Intercepting function/method calls

```
@make_interceptable
def make_email(...):
    ...
```

indico

```
from indico.core import signals
from indico.core.config import config
from indico.core.notifications import make_email
from indico.util.signals import interceptable_sender

@signals.plugin.interceptable_function.connect_via(interceptable_sender(make_email))
def _make_email_from_no_reply(sender, args, ctx, RETURN_NONE, **kwargs):
    args.arguments['from_address'] = config.NO_REPLY_EMAIL
```

indico-un

Refining queries

```
regforms = (RegistrationForm.query.with_parent(self.event)
            .signal_query(publishable-regforms')
            .all())
```

indico

```
@signals.core.db_query.connect_via('publishable-regforms')
def _intercept_publishable_regforms(sender, query, **kwargs):
    return query.filter(...)
```

indico-un

Adding **columns/relationships** to models

```python
from indico.core.db import db
from indico.modules.categories import Category

from un.util.patcher import patched

@patched
class _Category(Category):
    organizational_unit_id = db.Column(...)
    organizational_unit = db.relationship(...)
```

```
indico db migrate
```

Adding **constraints** to model tables

```
@patched
class _Category(Category):
    Category.__table__.append_constraint(
        db.CheckConstraint(
            "(lop_template IS NULL) = (lop_template_metadata::text = 'null')",
            'valid_lop_template'),
    )
```

```
indico db migrate
```

Overriding **methods** in classes

```
@patched
class _RegistrationFormItem(RegistrationFormItem):
    @hybrid_property
    def is_section(self):
        ...

    @is_section.expression
    def is_section(cls):
        ...
```

Extending WTForms

```
@patched
class _EventDataForm(EventDataForm):
    field_order = ('title', 'short_title')

    short_title = StringField(_('Short Title'), [Length(0, 30)])
    checkin_on_print = BooleanField(_('Check-in on print'))
```

Extending enums

```python
from indico.modules.search.base import SearchTarget
from indico.util.enum import IndicoEnum

from un.util.patcher import patch_enum

@patch_enum(SearchTarget, value_padding=1000)
class _SearchTarget(int, IndicoEnum):
    registration = 1
    accreditation = 2
```

Binding permissions to WTForm fields

New logic

```
@patched
class _IndicoForm(IndicoForm):
    __permission_bindings__ = None
    ...
```

```
@patched
class _CategorySettingsForm(CategorySettingsForm):
    __permission_bindings__ = {
        'title': PermissionBinding('category.manage.title')
    }
```

Ithaca

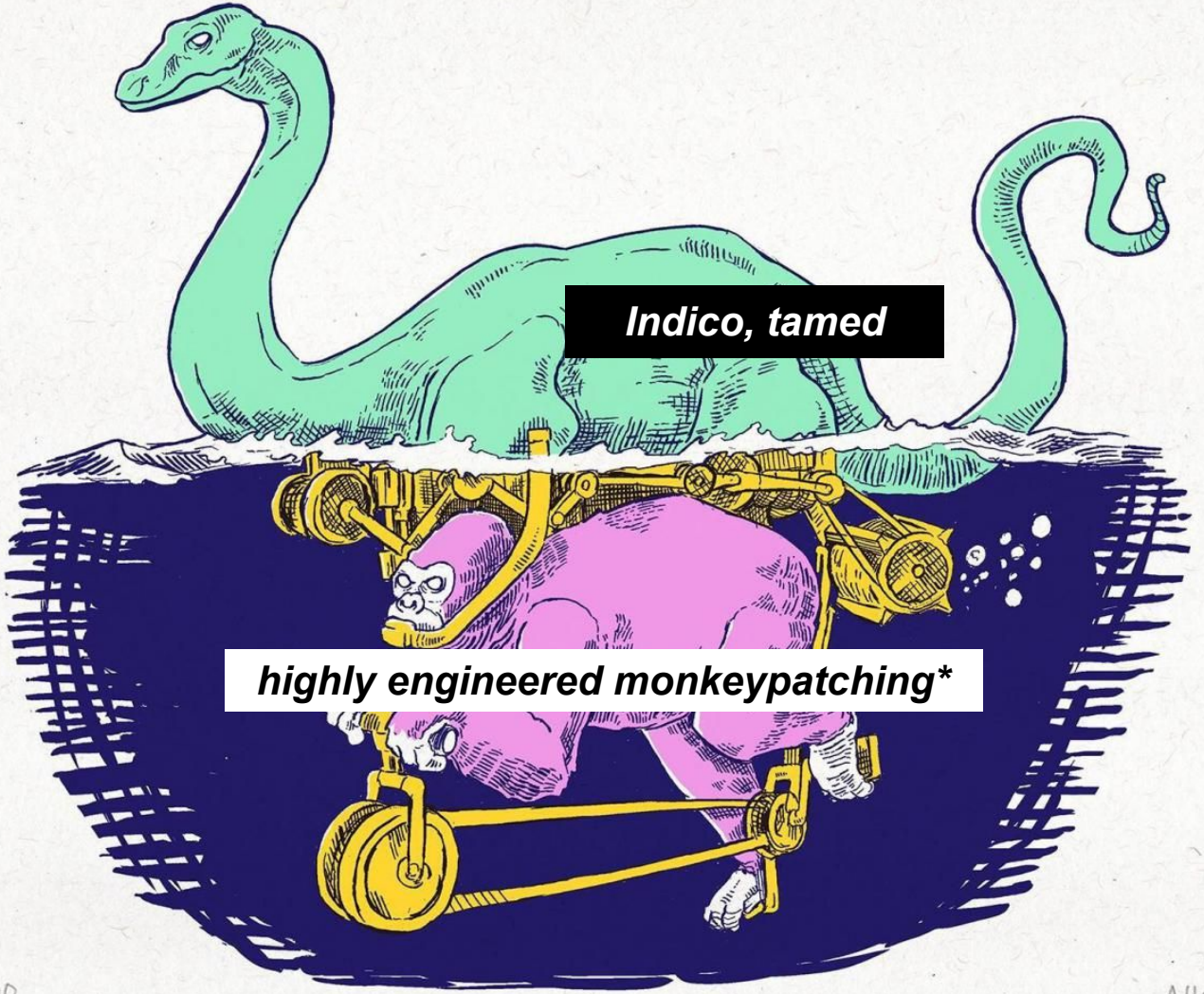
The long way home


“plugin”
CERN
Palais des Nations
hope

Ithaca, treasures and the future

- Potential contributions upstream / ecosystem:
  - Patching mechanism
  - Arbitrarily granular access control
  - DB schema migration script consistency tool
  - Alemfix command line tool for switching Alembic revisions

- Wishlist for Indico core:
  - Possibility to split plugin template directory into many
  - Some way to have registrations not linked to events


Indico, tamed
highly engineered monkeypatching*