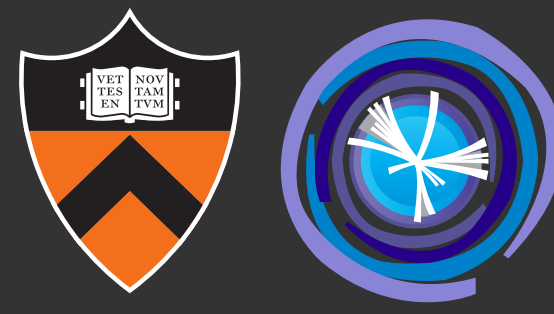




# Differentiable Statistical Ecosystem with JAX

**Peter Fackeldey**



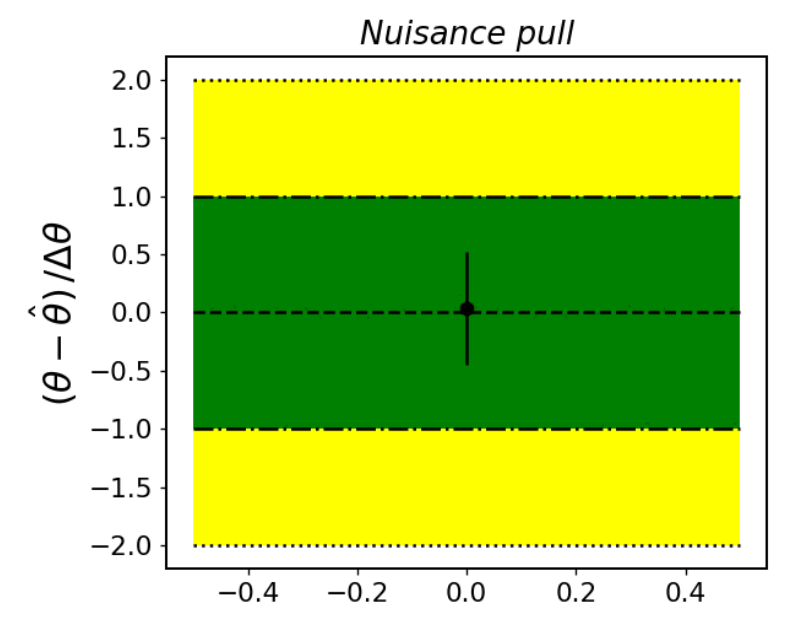
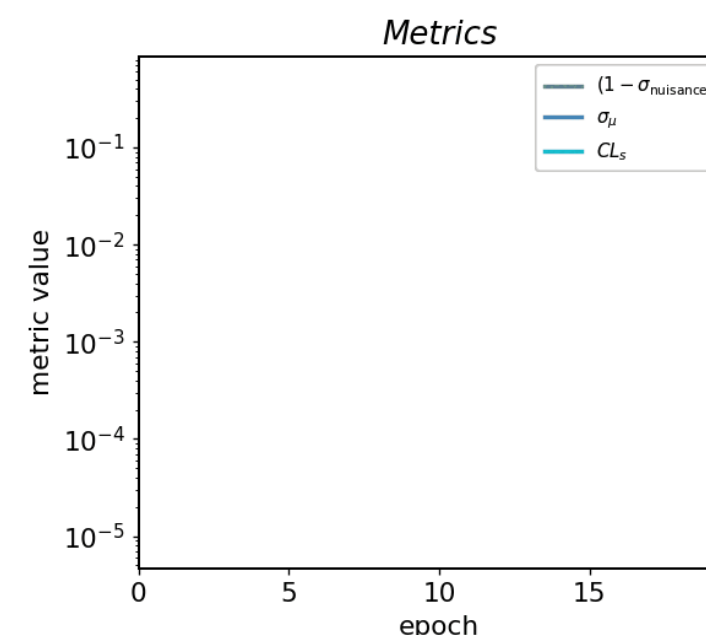
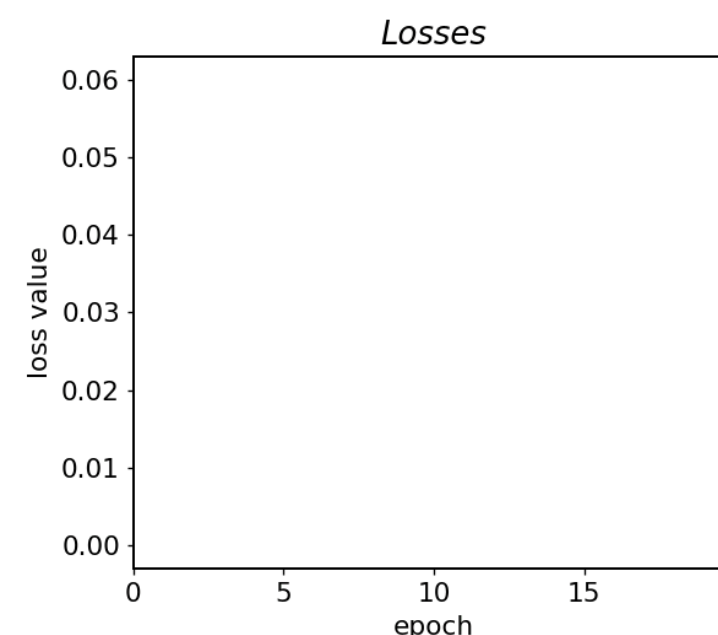
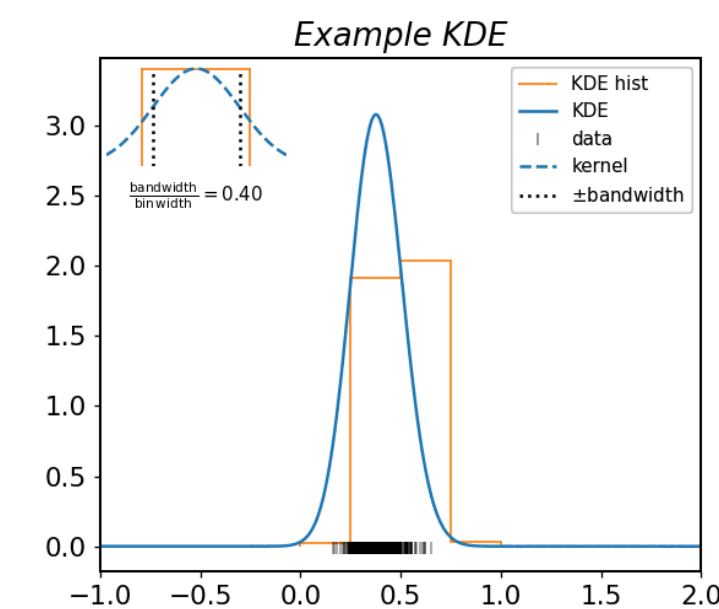
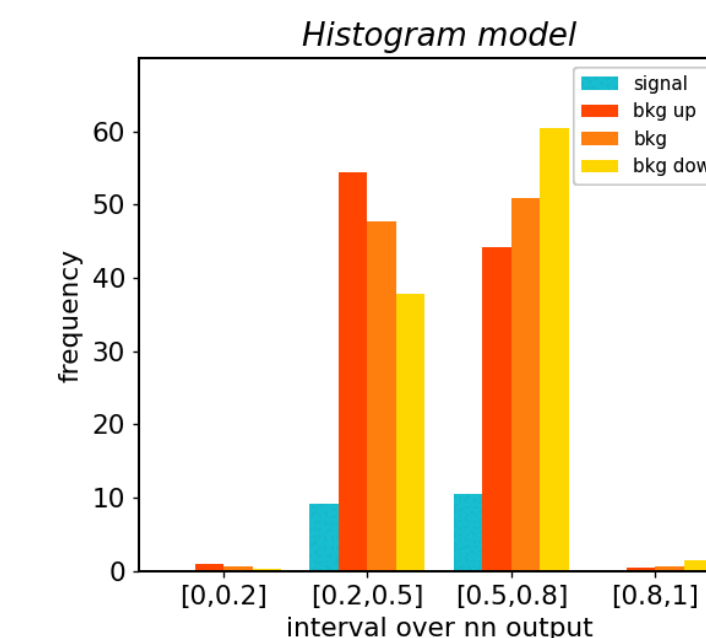
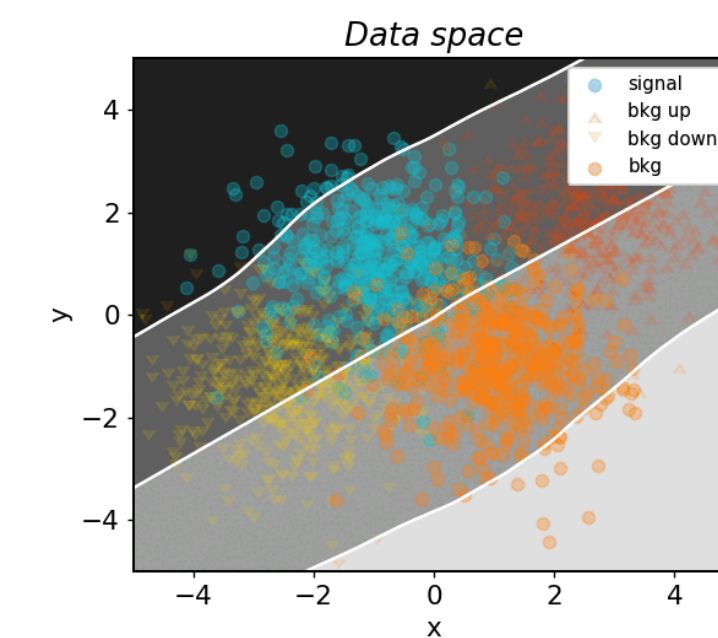
# 2 Why are differentiable Stat. Tools important?

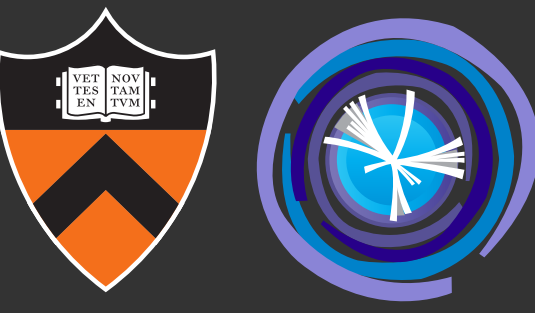
- Statistical measurement is the last step of a physics analysis
  - Back-propagating gradients means to start at the end (the statistical measurement) and go backwards in the analysis chain
- We're interested in gradients w.r.t. final measurement

- What's currently there?

1. RooFit with Clad (next talk)
2. JAX (*this* talk)

- Shout-out here for the first differentiable stat. tool chain with JAX: *neos*





# 3 JAX: ecosystem

- In first-order JAX can be seen as a drop-in NumPy replacement with enhanced functionality

```
import numpy as np → import jax.numpy as jnp
```

- Widely used in scientific computing and AI research, e.g., used for training Gemini by Google
- Integrates well with powerful ecosystems:
  - The whole scientific python ecosystem through Array API: Matplotlib, NumPy, SciPy, hist/boost-histogram, HDF5, uproot, iminuit, ...
  - The JAX ecosystem (not a full list, some examples):

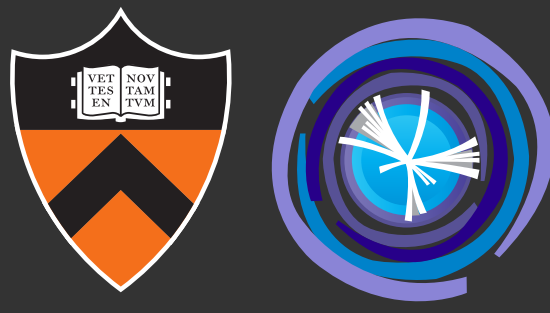
Equinox, NNX  
**AI**

Optimistix, Optax  
**Optimisers**

Comet.ml  
**Visualization**

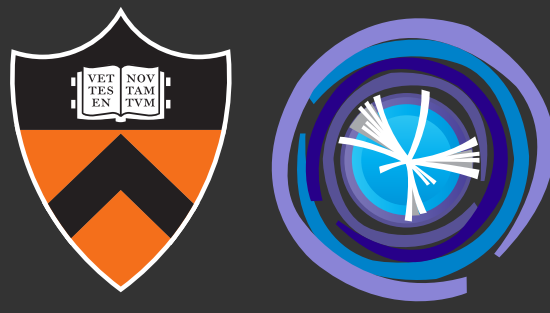
orbax, chex  
**Utils**

# 4 JAX: Program Transformations



- JAX is a NumPy-like array library that enhances “array-programs” through functional program transformations, e.g., for auto-differentiation:
  - 1st derivative     assert **jax.grad**(jnp.sin)(0.) == jnp.cos(0.)
  - 2nd derivative    assert **jax.grad(jax.grad(jnp.sin))**(0.) == -jnp.sin(0.)
- Multiple program transformations exist: `jax.grad`, `jax.jit`, `jax.vmap`, `jax.hessian`, ... all transformations are composable, e.g., 2nd derivative example above
- Transforms are applied at a custom IR (Jaxpr) that’s build through tracing step before translation to compiler IRs (XLA, LLVM, ...)
- Jaxprs are backend agnostic, allows for accelerator-agnostic transforms (e.g. AD)
  - functional API to calculate gradients efficiently

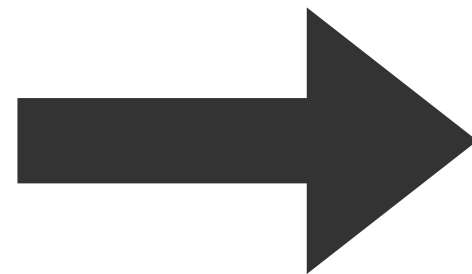
# 5 JAX: Program Transformations (AD example)



- `jax.grad` modifies (“transforms”) the Jaxpr to include the gradient calculation (the “nominal” output “\_” is optimized away later during the compiler optimization)

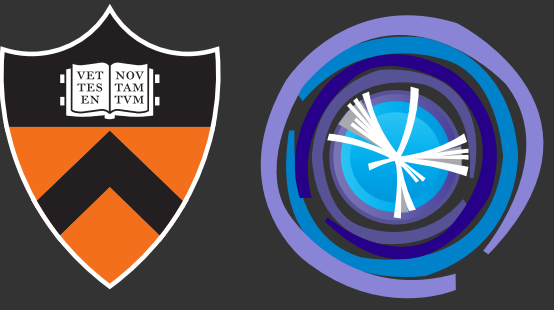
```
def fun(x):  
    return 2 + jnp.sin(x)  
  
jax.make_jaxpr(fun)(jnp.array(1.0))  
# { lambda ; a:f32[]. let  
#   b:f32[] = sin a  
#   c:f32[] = add 2.0 b  
#   in (c,) }
```

**jax.grad**

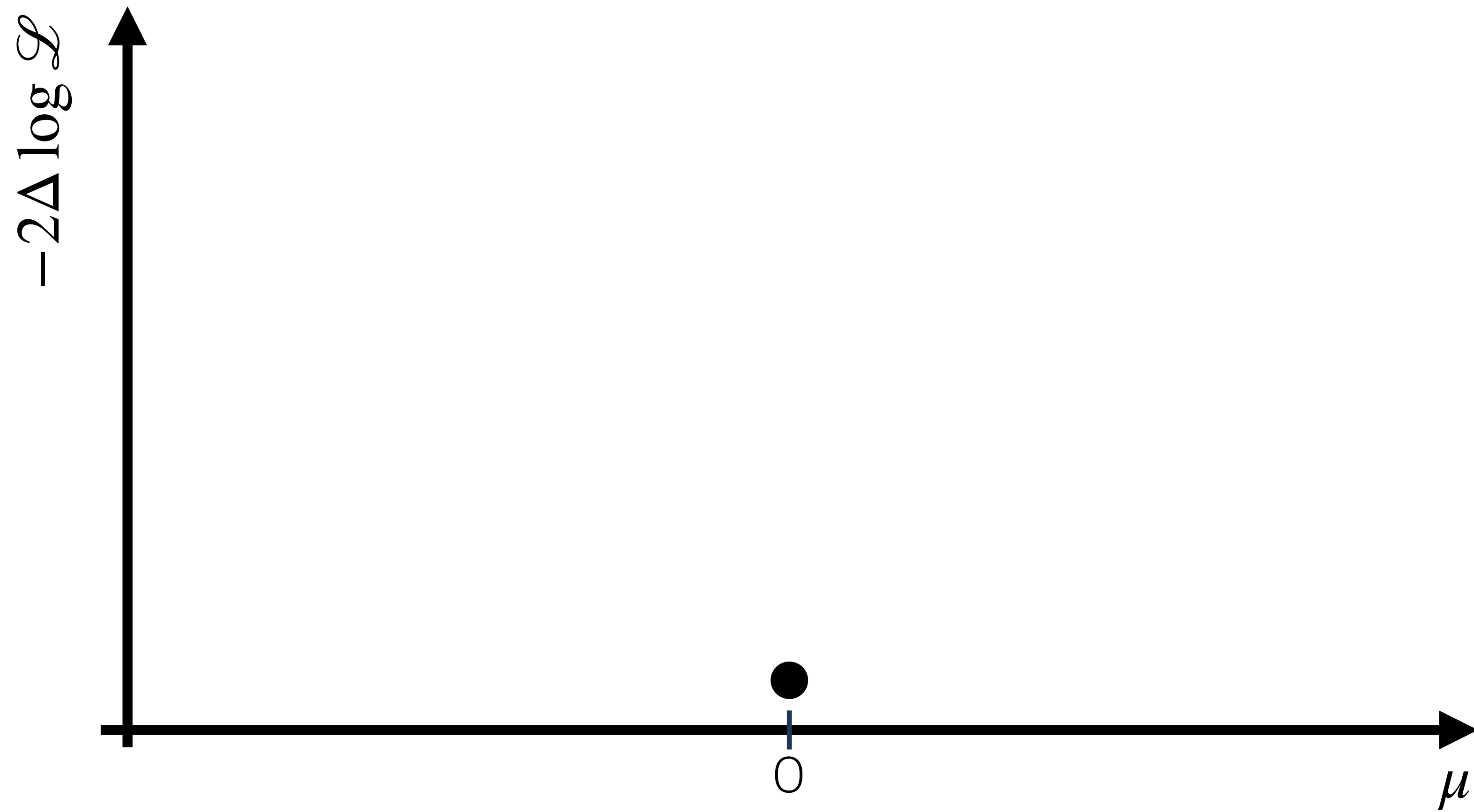


```
def fun(x):  
    return 2 + jnp.sin(x)  
  
jax.make_jaxpr(jax.grad(fun))(jnp.array(1.0))  
# { lambda ; a:f32[]. let  
#   b:f32[] = sin a  
#   c:f32[] = cos a  
#   _:f32[] = add 2.0 b  
#   d:f32[] = mul 1.0 c  
#   in (d,) }
```

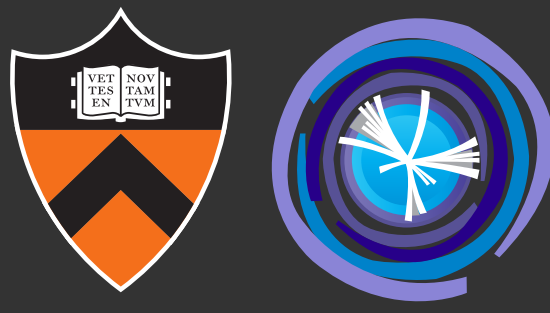
# 6 JAX: Program Transformations (MINOS)



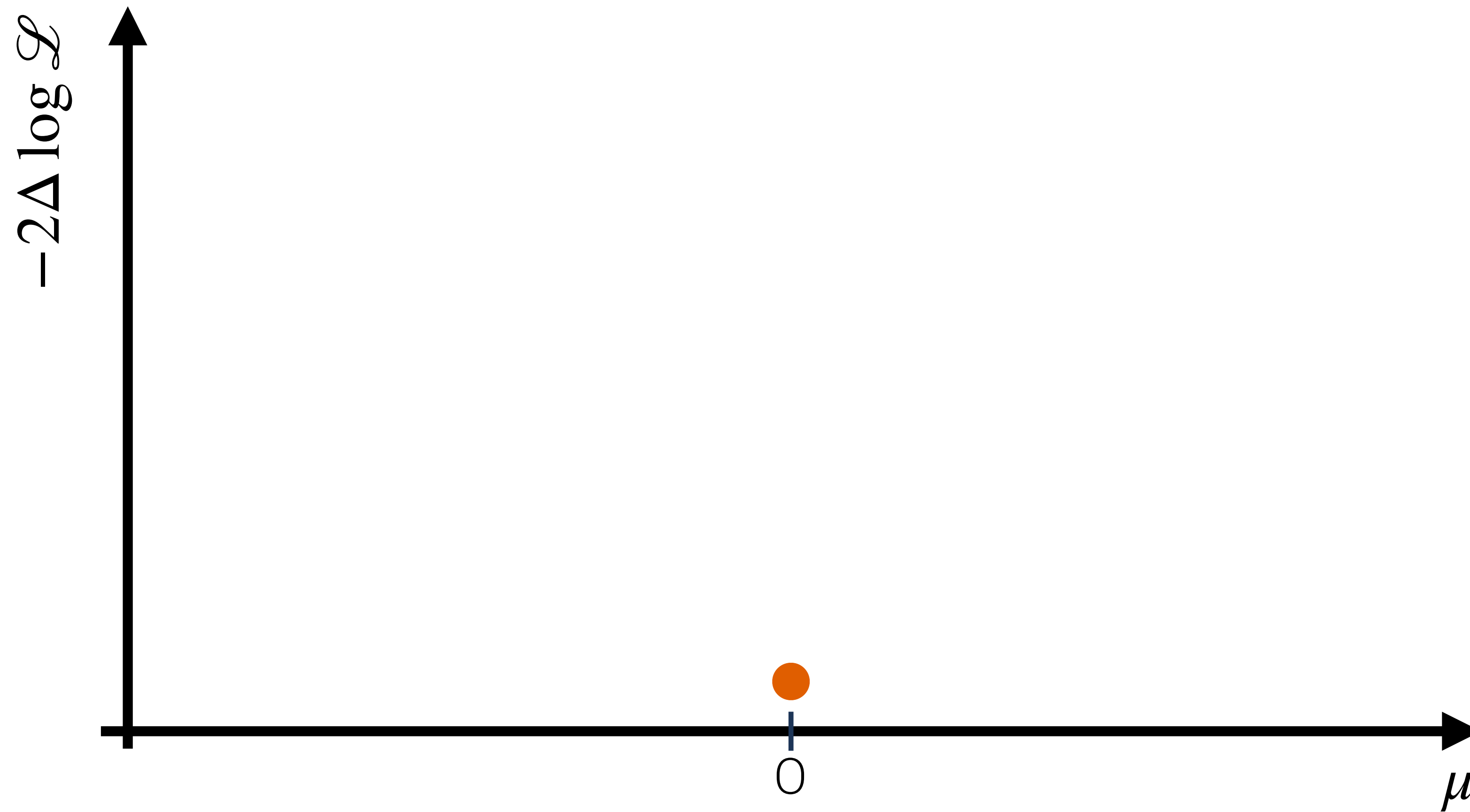
`nll_fit(0.)`



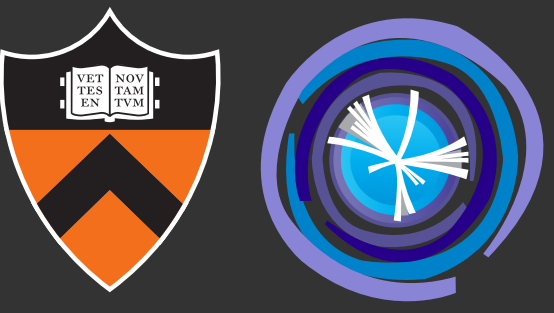
# 7 JAX: Program Transformations (MINOS)



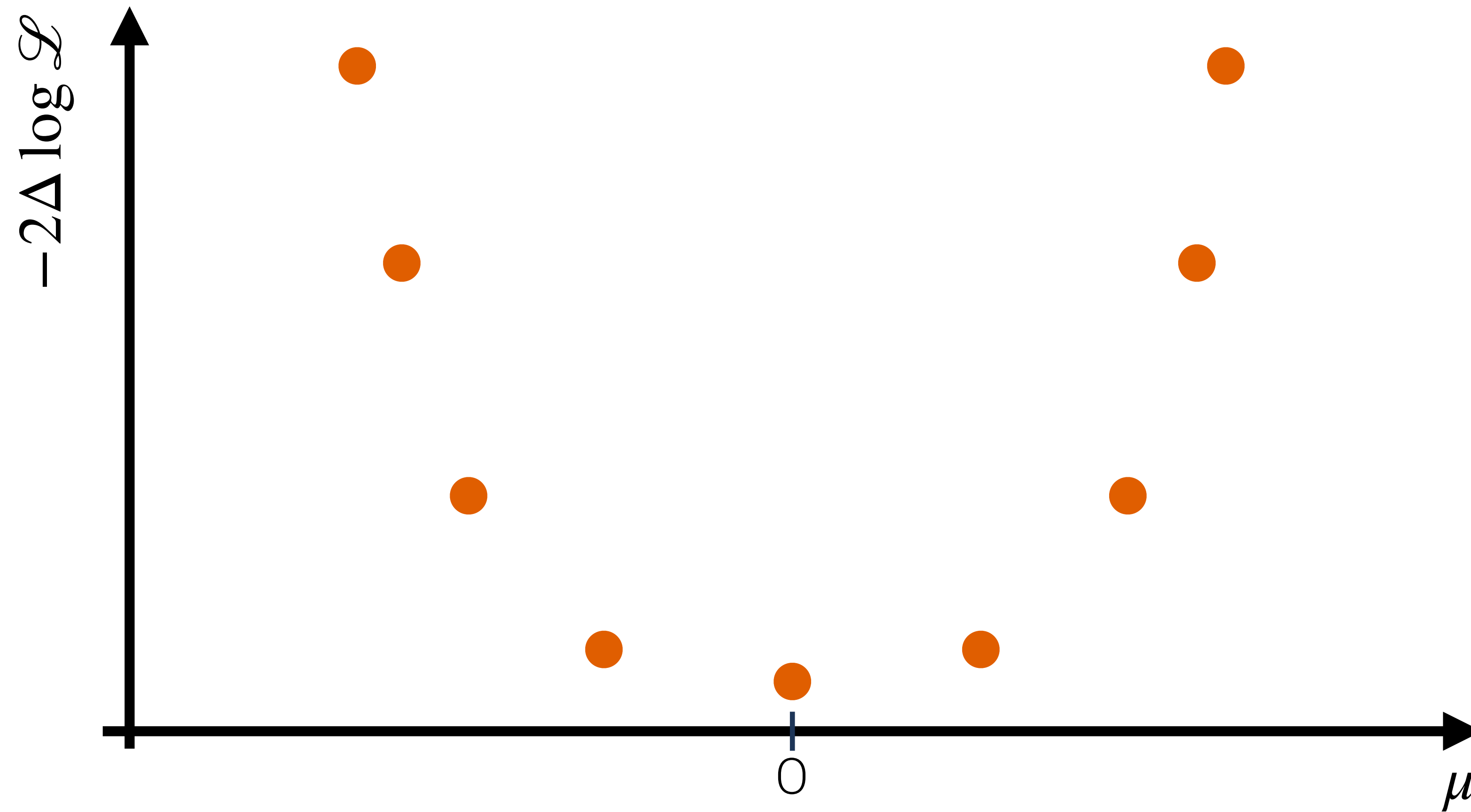
`jax.jit(nll_fit)(0.)`



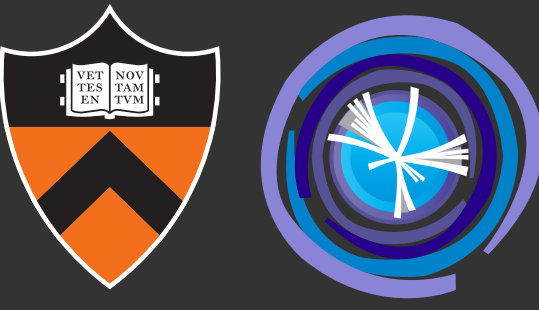
# 8 JAX: Program Transformations (MINOS)



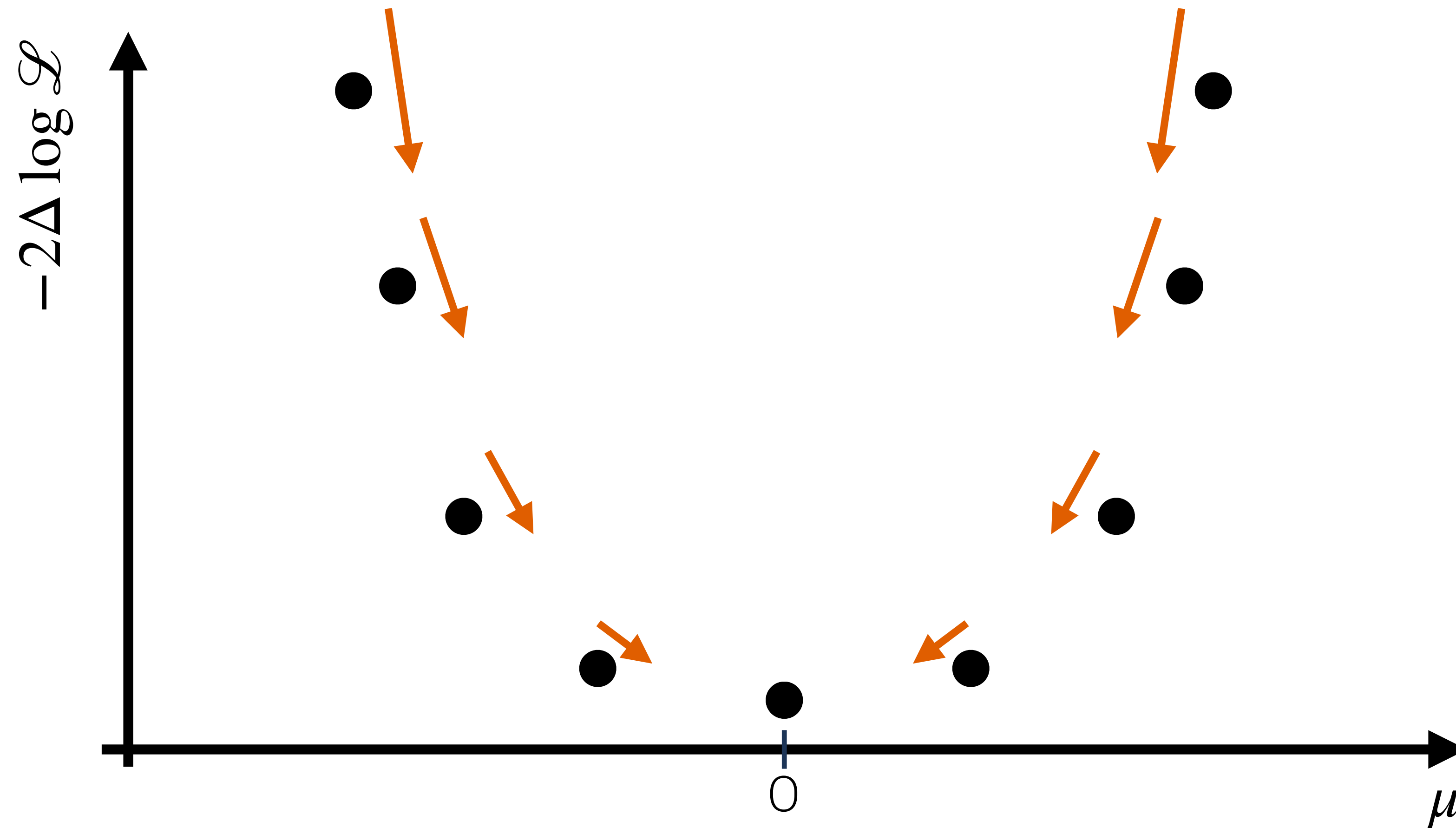
`jax.vmap(jax.jit(nll_fit))(jnp.array(...))`



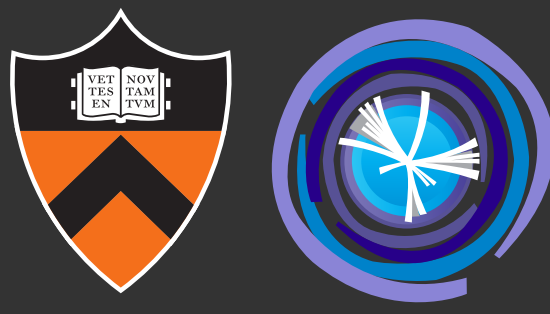
# 9 JAX: Program Transformations (MINOS)



```
jax.grad(jax.vmap(jax.jit(nll_fit)))(jnp.array(...))
```



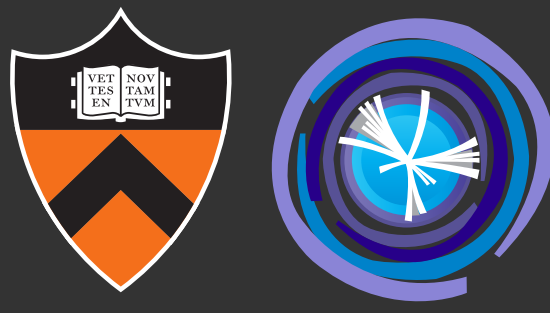
# 10 Optimistix: Differentiable Optimiser



- Many solvers available (GradientDescent, Bisection, BFGS, L-BFGS, MIGRAD, SIMPLEX, ...)  
(we added MIGRAD last year to optimistix [[PR #125](#)] 🎉)
- Optimizers are PyTrees themselves which allows to transparently JIT & vmap optimizers  
...or even differentiate through minimisations (implicit function theorem)
- Optimizers are composable (mix-and-match), e.g., use BFGS with trust-region:

```
class BFGSClassicalTrustRegionHessian(optx.AbstractBFGS):  
    """Standard BFGS (uses hessian, not inverse!) + classical trust region update."""  
  
    rtol: float  
    atol: float  
    norm: Callable = optx.max_norm  
    use_inverse: bool = False  
    search: optx.AbstractSearch = optx.ClassicalTrustRegion()  
    descent: optx.AbstractDescent = optx.NewtonDescent()
```

# 11 What tools do exist already?



- **pyhf with its jax backend:**

- Get a JAX-compatible (JIT-able, diffable, ...) likelihood from pyhf-workspace
- gradhep/neos enhances pyhf's inference part to be differentiable

- **pyhs3:**

- General HS3 to python likelihood-function converter
- Can transpile to JAX-compatible likelihood

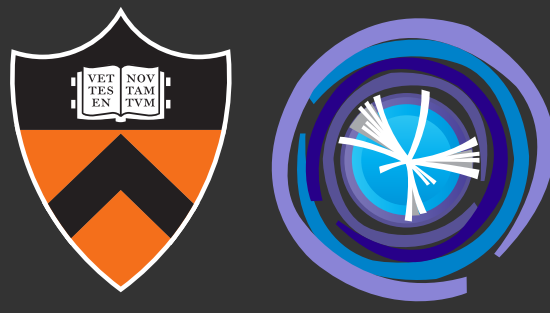
- **evermore:**

- Provides building blocks to write likelihoods with pure JAX primitives
- Every component by definition JIT-able, diffable, ...

- **everwillow:**

- Inference-only tool that implements CLs method, etc, with a focus on JAX-compatibility

# 12 What tools do exist already?



- **pyhf with its jax backend:**

- Get a JAX-compatible (JIT-able, diffable, ...) likelihood from pyhf-workspace
- gradhep/neos enhances pyhf's inference part to be differentiable

- **pyhs3:**

- General HS3 to python likelihood-function converter
- Can transpile to JAX-compatible likelihood

More details in stat. Tools  
blueprint meeting talk during  
last week by Mo & me

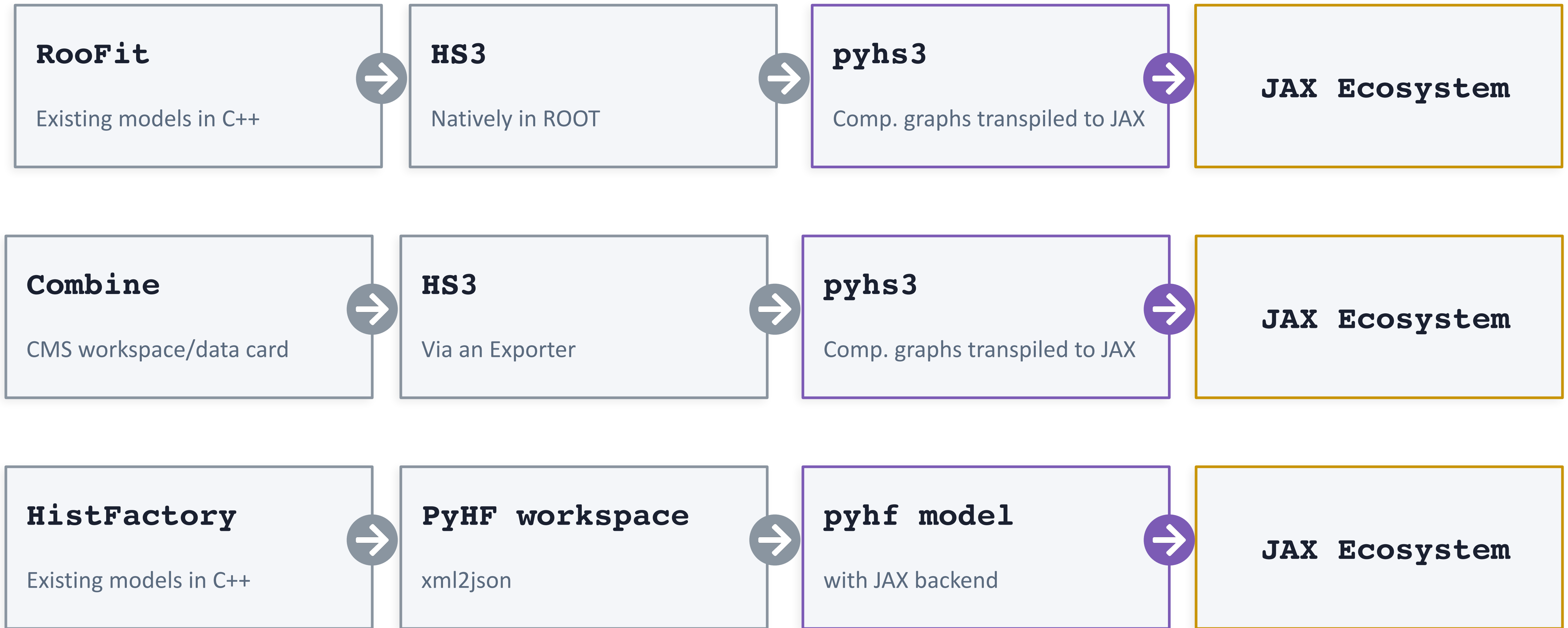
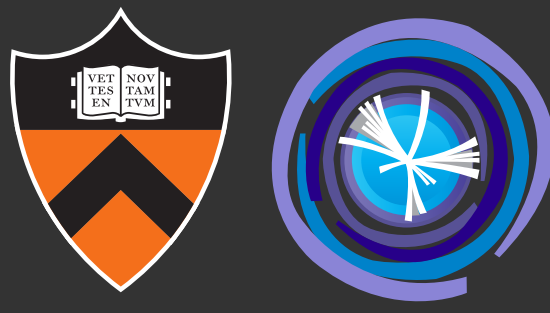
- **evermore:**

- Provides building blocks to write likelihoods with pure JAX primitives
- Every component by definition JIT-able, diffable, ...

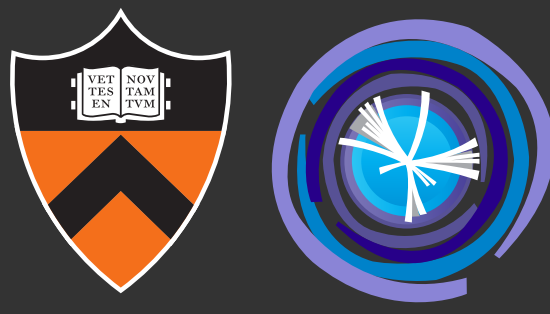
- **everwillow:**

- Inference-only tool that implements CLs method, etc, with a focus on JAX-compatibility

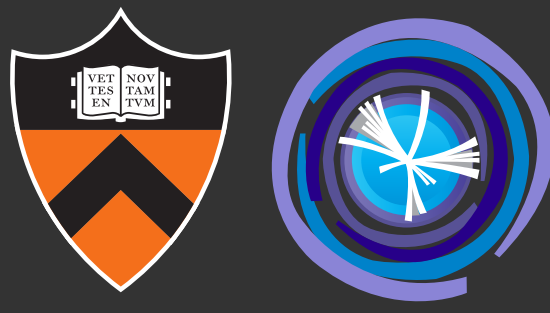
# 13 How to I enter this ecosystem?



# 14 Interoperability



- With other JAX-based tools:
  - native interoperability
- With likelihoods defined in HS3:
  - through pyhs3; HS3 can serve as language-agnostic the exchange format
- With RooFit:
  - Jonas Rembser and me have a prototype to combined likelihoods from RooFit and JAX and minimize them together (in either ‘world’)
  - Needs a discussion on a common interface/protocol for likelihoods now



- JAX gives us access to gradients (efficiently) and has a powerful ecosystem we can leverage
- Optimistix implements many solvers for us with the benefit of efficient differentiation through minimizations and a powerful mix-and-match API
- JAX stat. tooling ecosystem for HEP is growing, currently strongest support for HistFactory-like (binned template) likelihoods with pyhf & evermore
- Various ways to enter the “JAX world” from existing well-established workflows (primarily through the power of HS3)
- Interopability with RooFit is possible, although currently only a prototype
- zift2 will be in JAX → interoperable with JAX ecosystem