Relational Lenses as Libraries

Rudi Horn – Haskell Symposium 2020

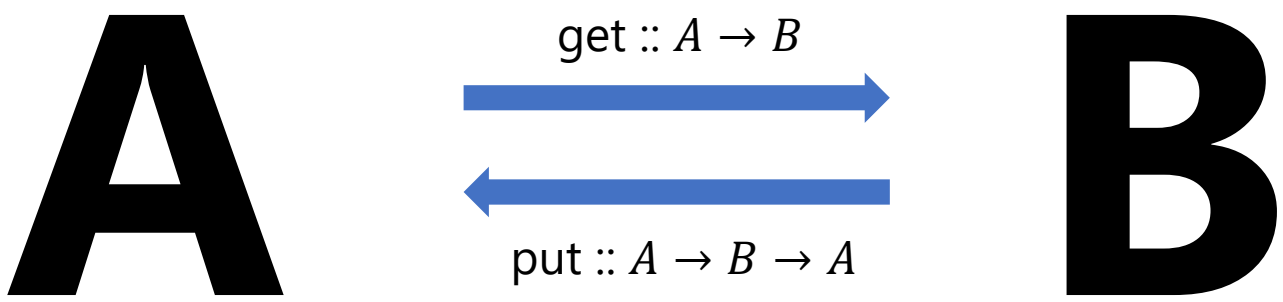
European Commission | Horizon 2020 European Union funding for Research & Innovation

EPSRC Centre for Doctoral Training in Pervasive Parallelism

THE UNIVERSITY of EDINBURGH informatics

Lenses

A form of **bidirectional transformations** [1, 2]

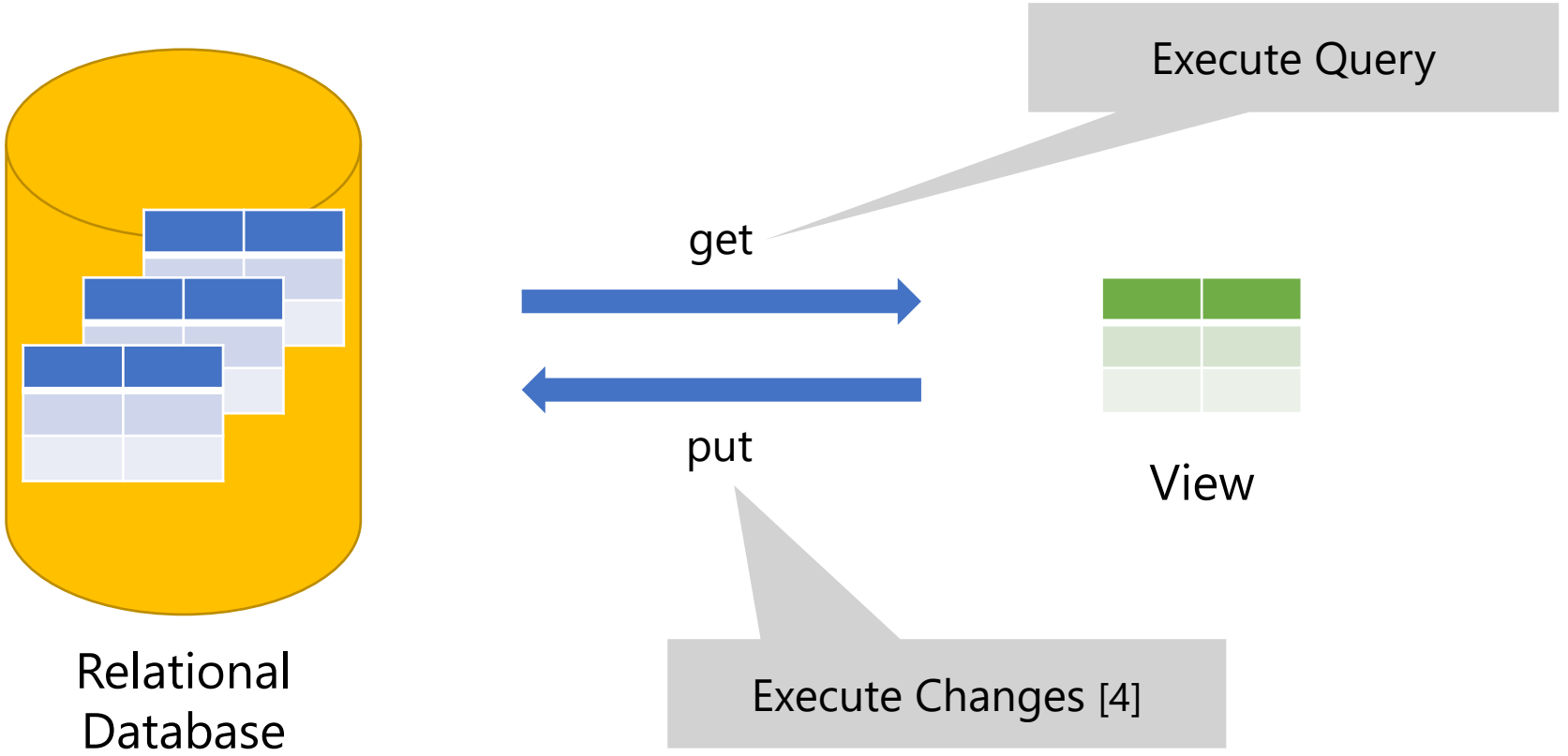


Example: Point { x :: Double; y :: Double }

get_x :: Point → Double
put_x :: Point → Double → Point

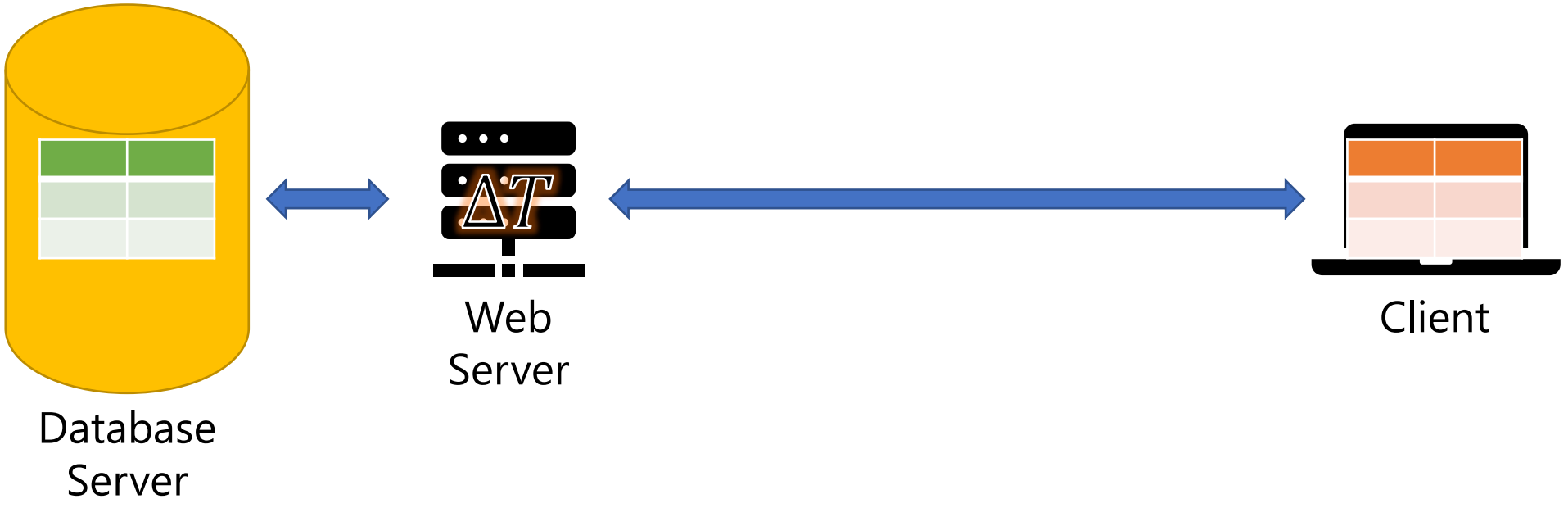
[1] Foster et al. "Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem."
[2] Bohannon, Aaron, et al. "Boomerang: resourceful lenses for string data."

Relational Lenses [3]



[3] Bohannon, Pierce, and Vaughan. "Relational lenses: a language for updatable views."
[4] Horn, Perera, and Cheney. "Incremental relational lenses."

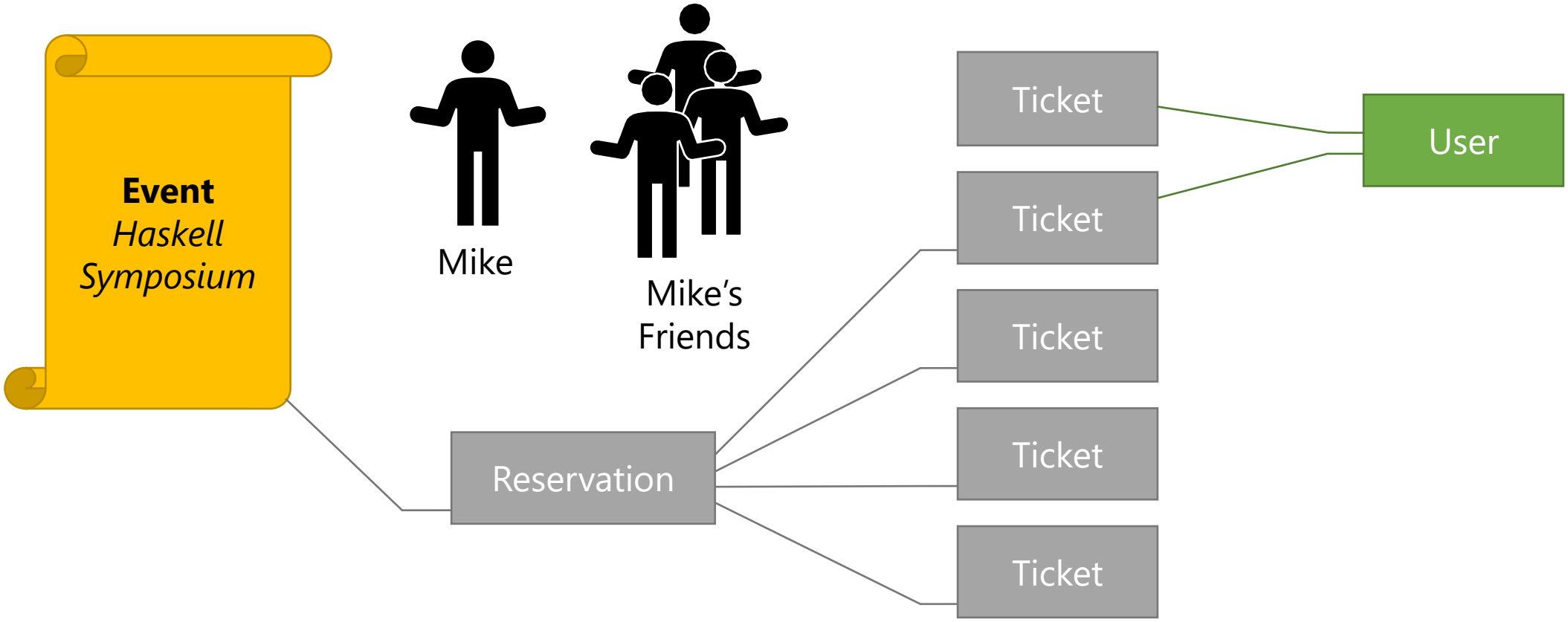
Relational Lenses



Typical Model-View-Controller workflow

Relational Lenses by Example

Lets build an event ticketing system!

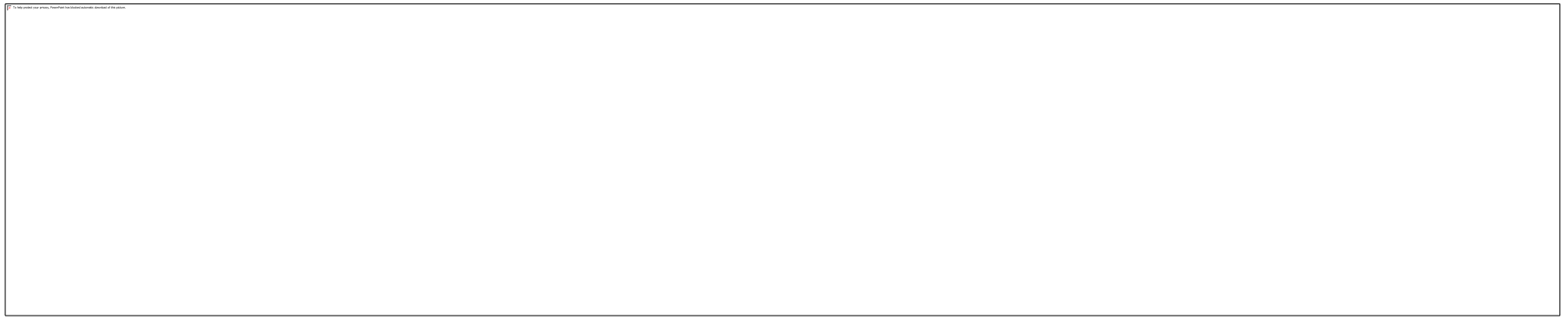


Relational Lenses by Example

Lets build an event ticketing system!

Relational Lenses by Example

Lets build an event ticketing system!



events

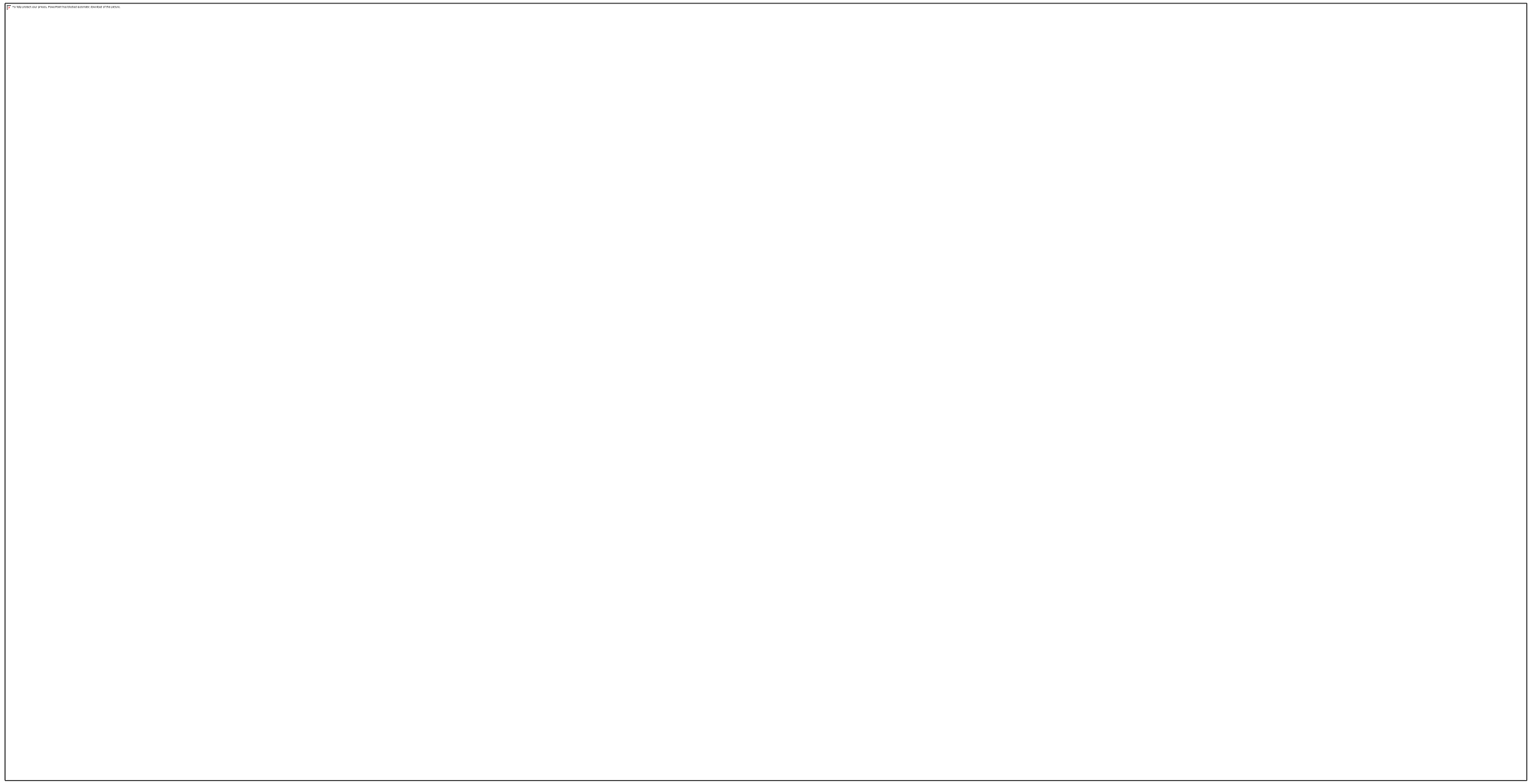
Relational Lenses by Example

events

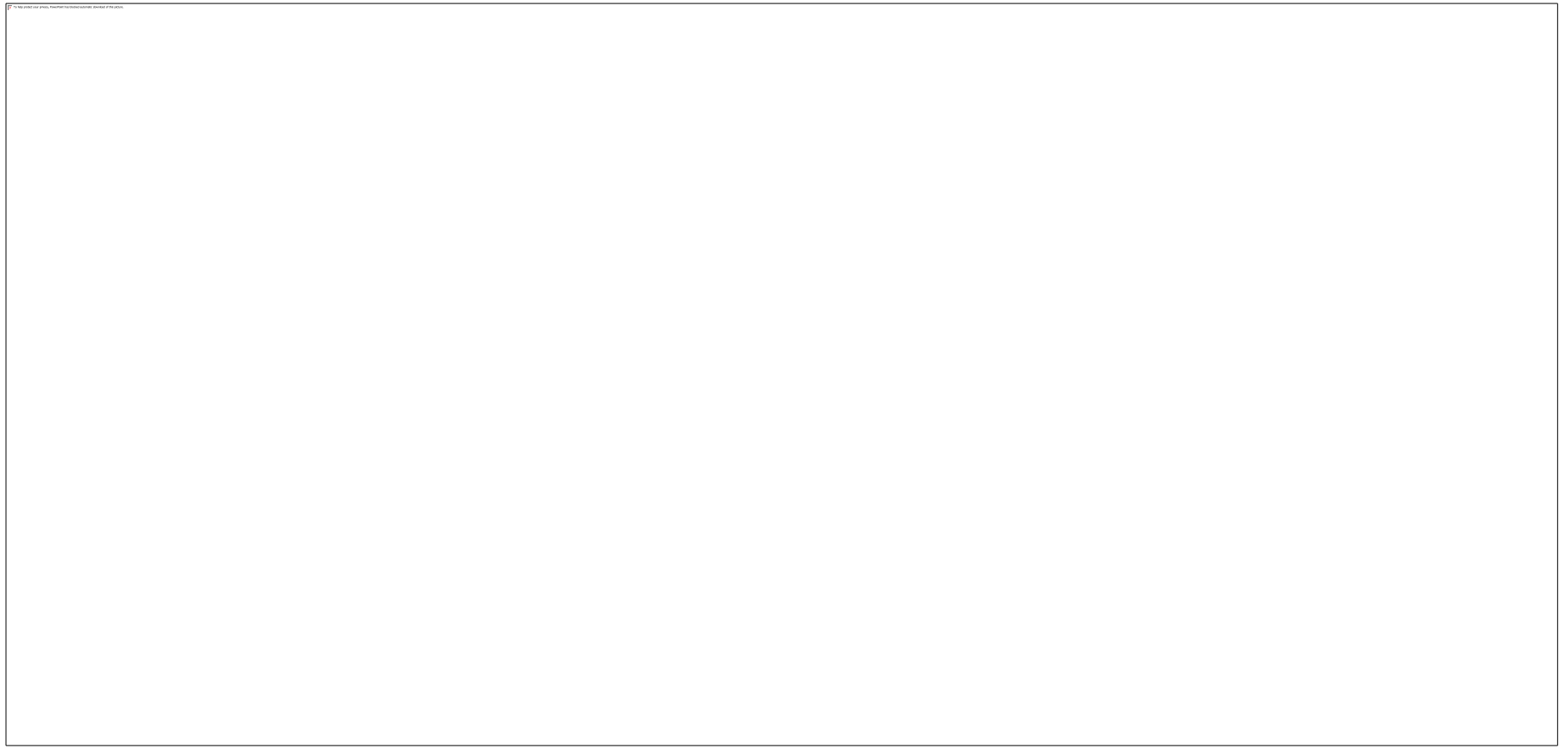
Relational Lenses by Example

reservations

Relational Lenses by Example



Relational Lenses by Example



Relational Lenses by Example

Correctness of Relational Lenses

Should be **well-behaved**

For bidirectional transformations:

- $get\big(put(a, b)\big) = b$
- $put\big(a, get(a)\big) = a$

Correctness of Relational Lenses

Linearity of Tables

Functional Dependencies
$\mathbf{X} \rightarrow \boldsymbol{Y}$

Correctness

Restrictions on Predicates

Consistency

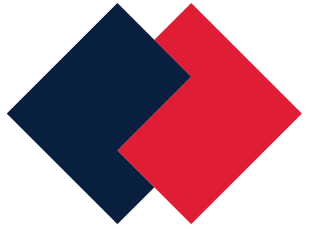
[3] Bohannon, Pierce, and Vaughan. "Relational lenses: a language for updatable views."
[5] Horn, Fowler, and Cheney. “Language-Integrated Updatable Views"

Existing Relational Lenses

Implemented in **Links**

- Extended compiler to support Relational Lenses
  - Implements **Incremental** Relational Lenses [4]
  - Demonstrates **language integration** in functional setting [5]
- Difficult to maintain
- Compatibility with other language features (e.g. continuation serialisation)

https://links-lang.org



Links

[4] Horn, Perera, and Cheney. "Incremental relational lenses."
[5] Horn, Fowler, and Cheney. “Language-Integrated Updatable Views"

Lenses as Library

Better approach: Implement feature as a **Library**

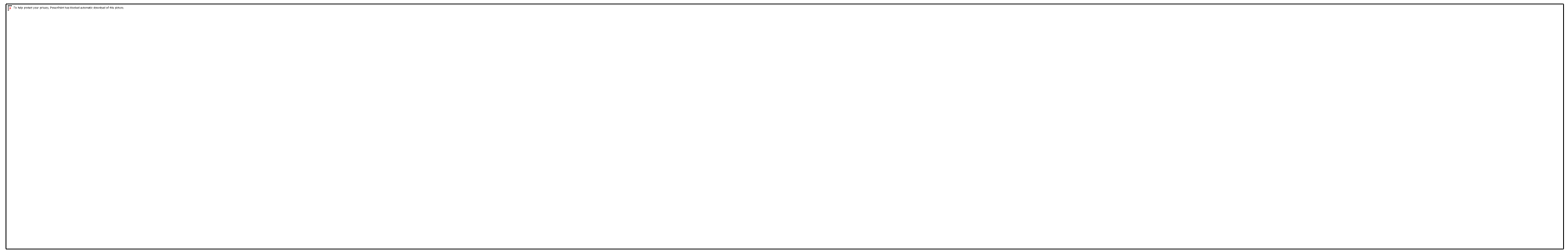
- How to verify correctness statically?
- Implement with reusable language features
- Cleaner abstractions for intended feature

➢Haskell type system sufficient for Relational Lenses

Type-level computation

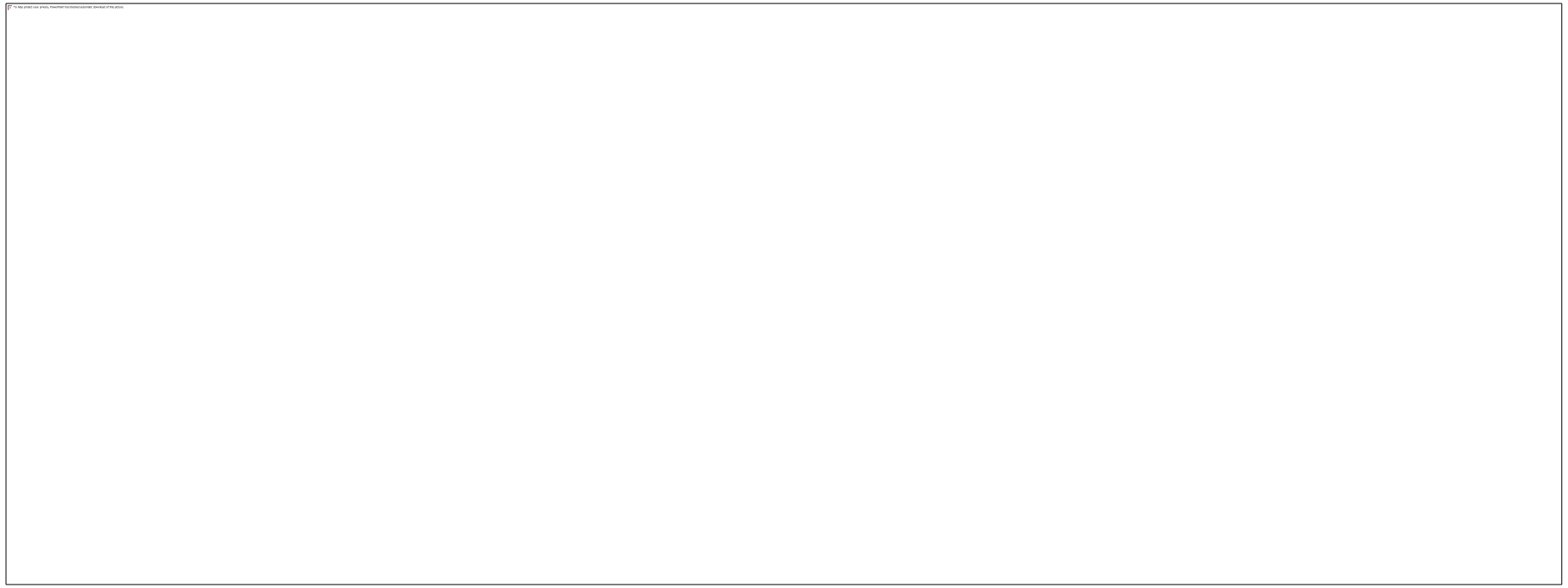
Type Level Programming

Type Literals



Type Families

Type Classes



Constraints

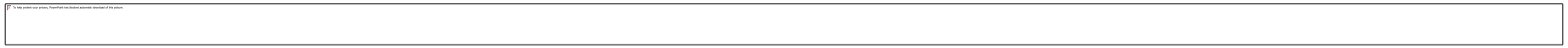
Equality Constraint
$\tau_1 \sim \tau_2$

Equality Constraint
`Recoverable a String`

Lens Building Blocks

Tables

User tries to construct lens:



uses same table twice!

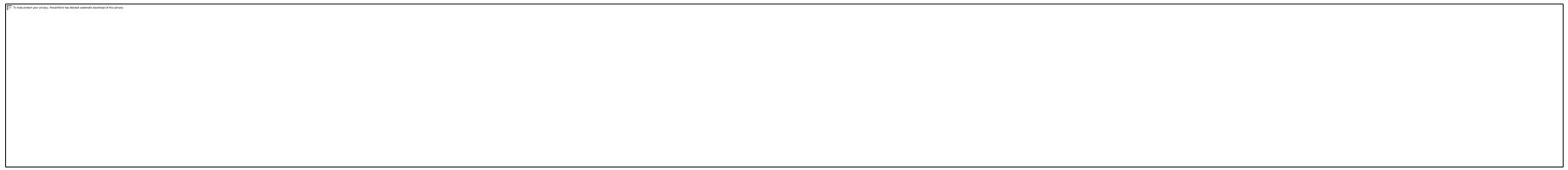
Need a constraint:

```
TablesDisjoint t1 t2 => ...
```

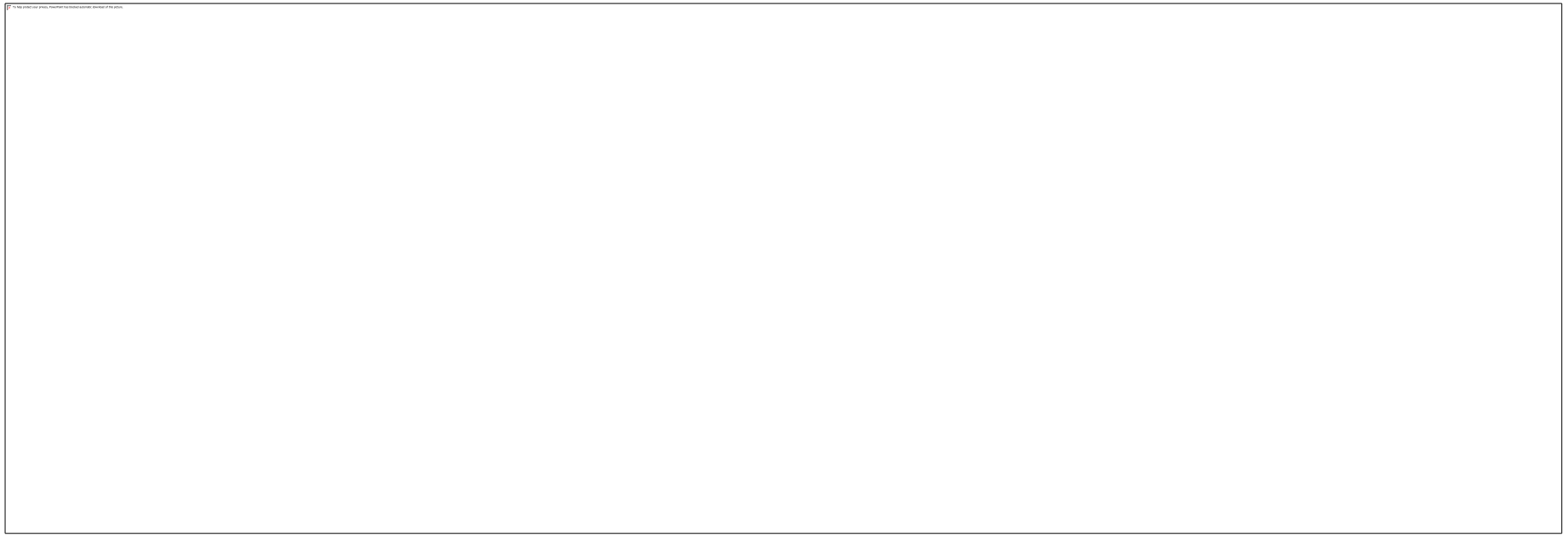
Tables

Record Types

Relational Lenses require representation for records / views

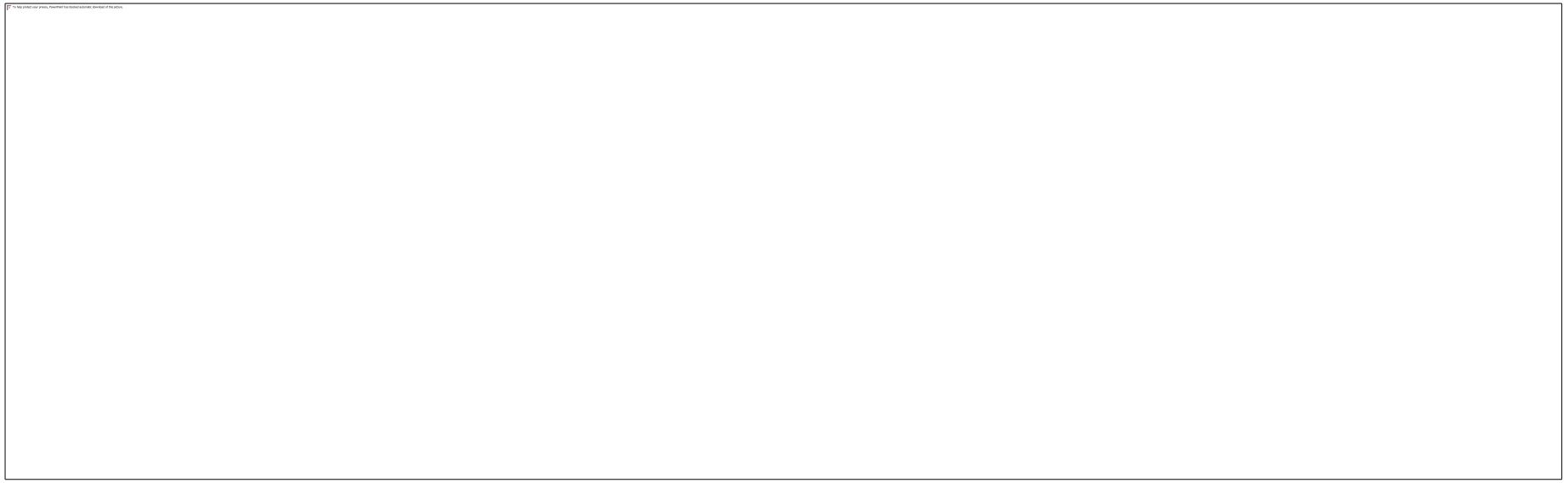


Record Types



Record Projection

Record Set

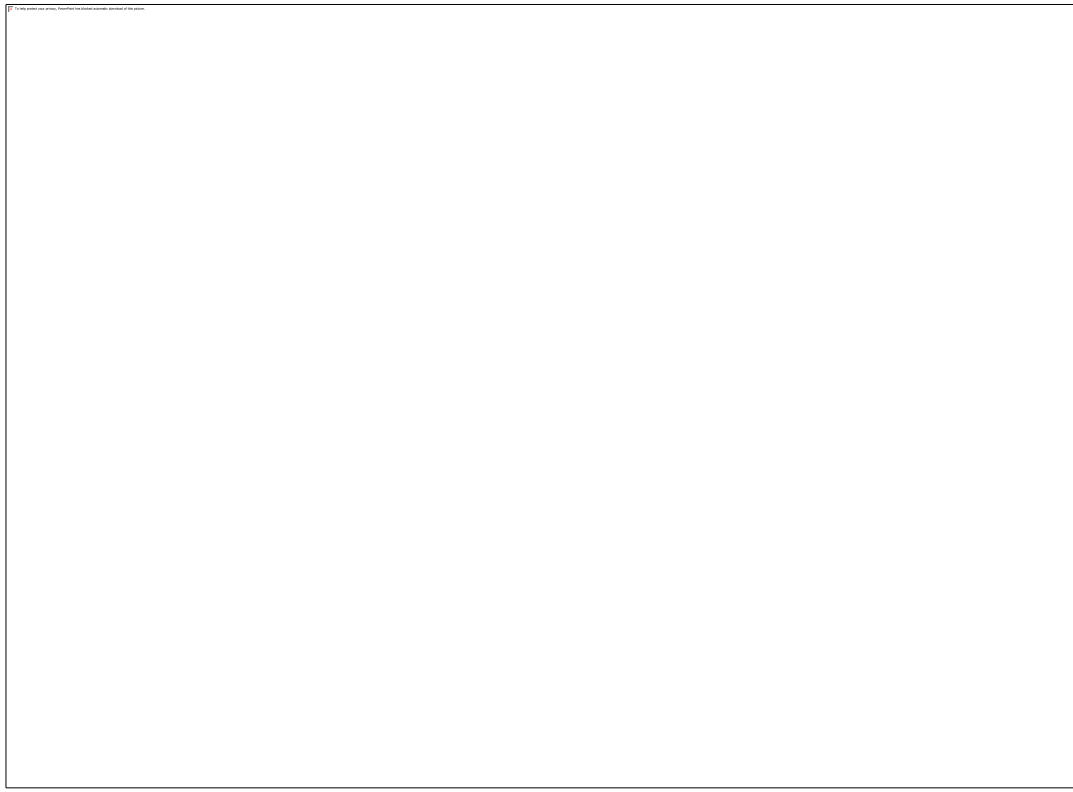


Functional Dependencies

Example: $email \rightarrow title, name$

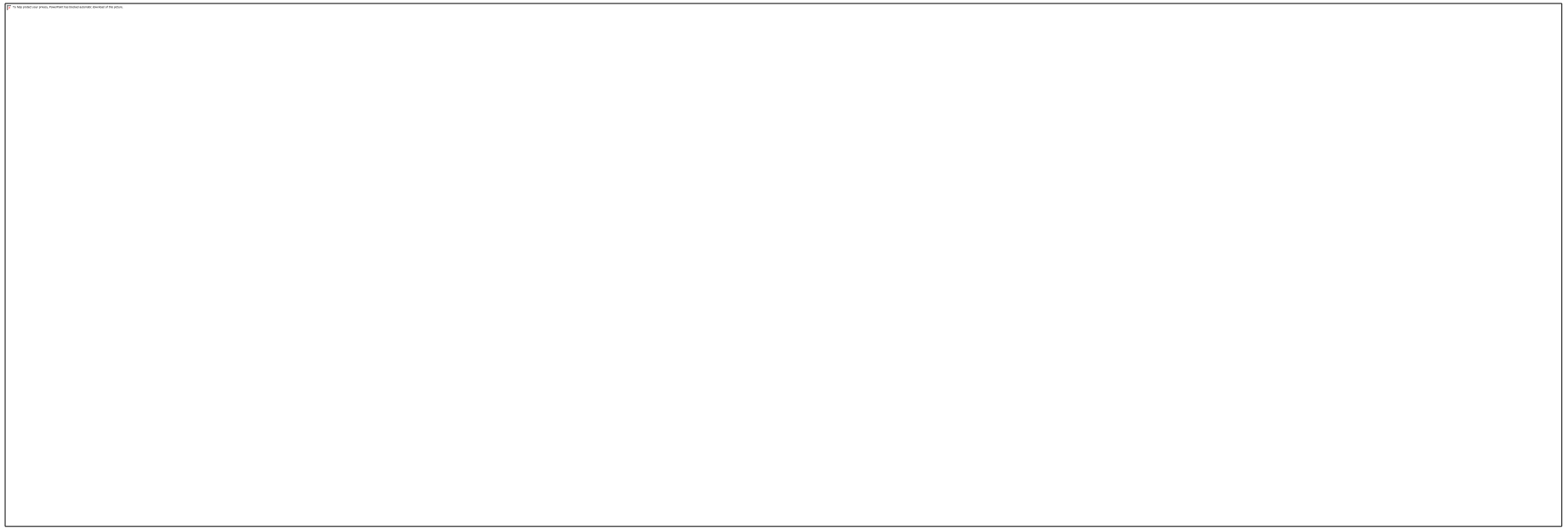
Tree form:

Forrest with disjoint nodes



Functional Dependencies

Functional Dependencies

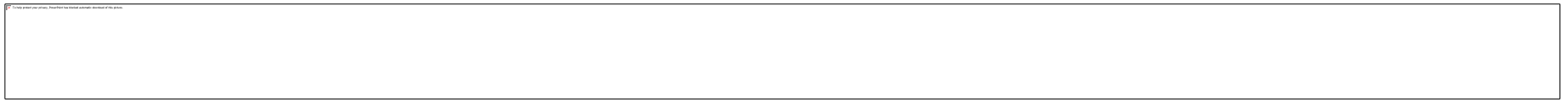


Predicates

Domain Specific Language (**DSL**) for predicates

- Predicate information retained in type
- Statically typable

Example for `test = 5`:



Predicates

Static predicates

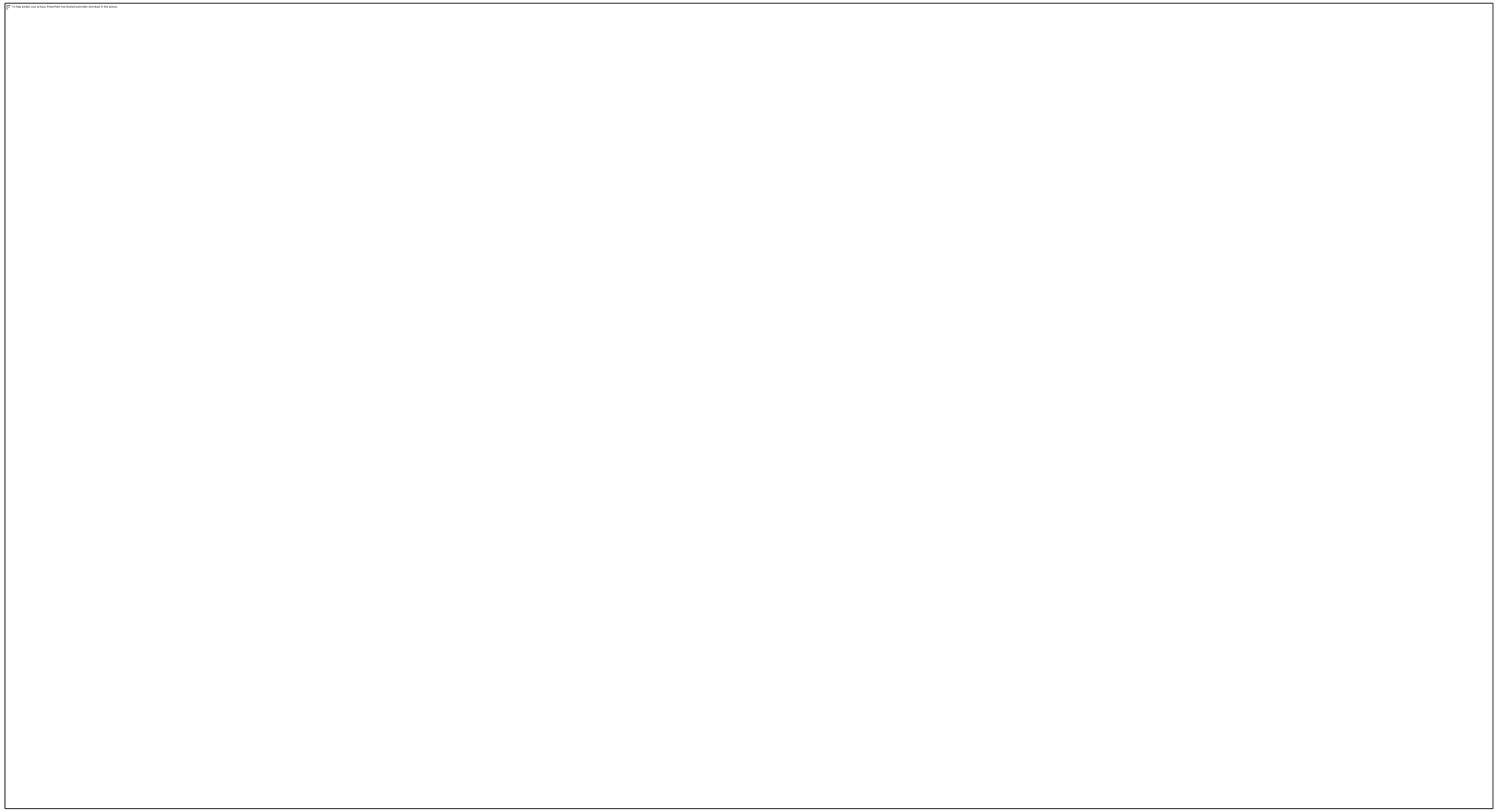
- not always flexible enough,
- however not always necessary!

Lens Constructors

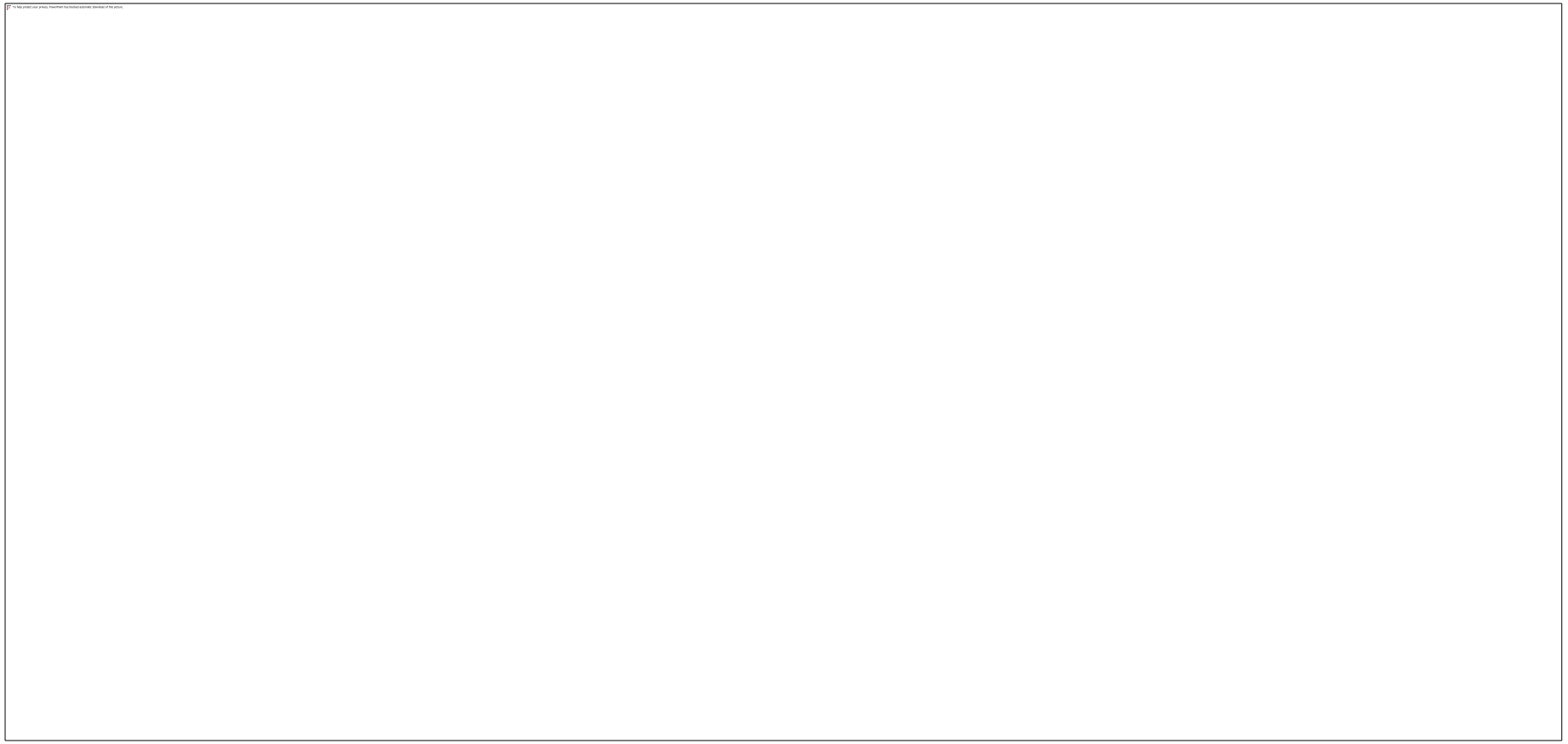
Lens Sorts

Refinement type for lenses

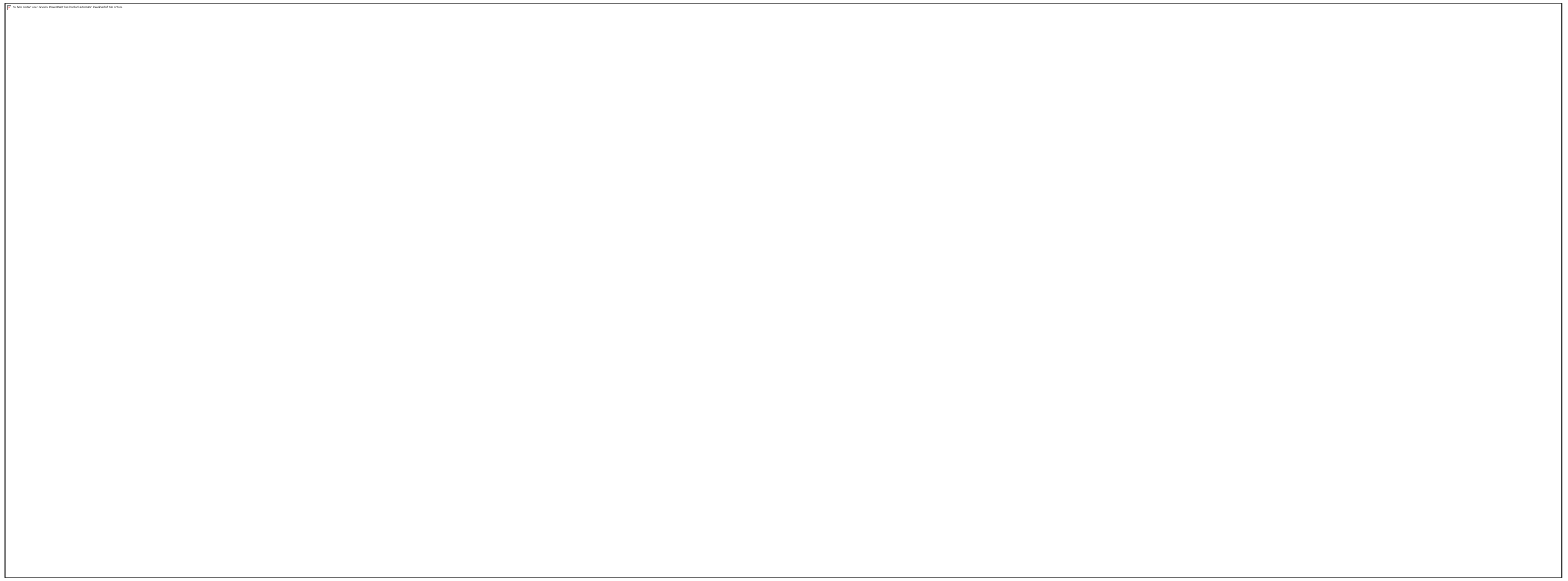
Table Primitive Lens



Select / Filter Lens

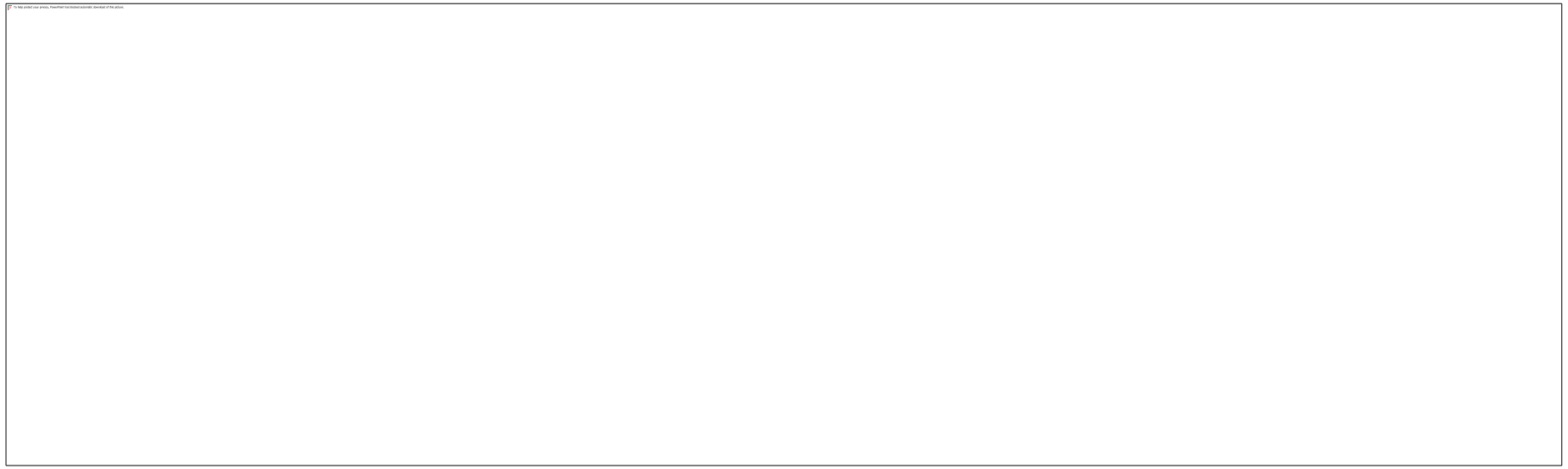


Select / Filter Lens



Further Lenses

Drop / Join lenses also supported:



Conclusion

Type-level Computation

No extensions to Haskell necessary!

Lots of similar work on row types:

- https://hackage.haskell.org/package/CTRex-0.6
- https://hackage.haskell.org/package/row-types

Also type level sets:
https://hackage.haskell.org/package/type-level-sets-0.8.0.0

Future Work

Better error handling:

Haskell supports: `TypeError msg`

Lots of large / messy constraints

- Tend to be unavoidable without fully dependent types
- Better abstractions?

Other database server support (shouldn’t be too difficult)

Serial column / auto incrementing column support

Results

- Haskell Lens Library
- Supports Incremental Relational Lens Semantics [4]
- Roughly ~3k of code
- Easy to use view-update for relational databases

[4] Horn, Rudi, Roly Perera, and James Cheney. "Incremental relational lenses."

Questions?