

Haskell boards the Ferry

A Database Coprocessor for Haskell



George Giorgidze

Torsten Grust

Tom Schreiber

Jeroen Weijers

www-db.informatik.uni-tuebingen.de

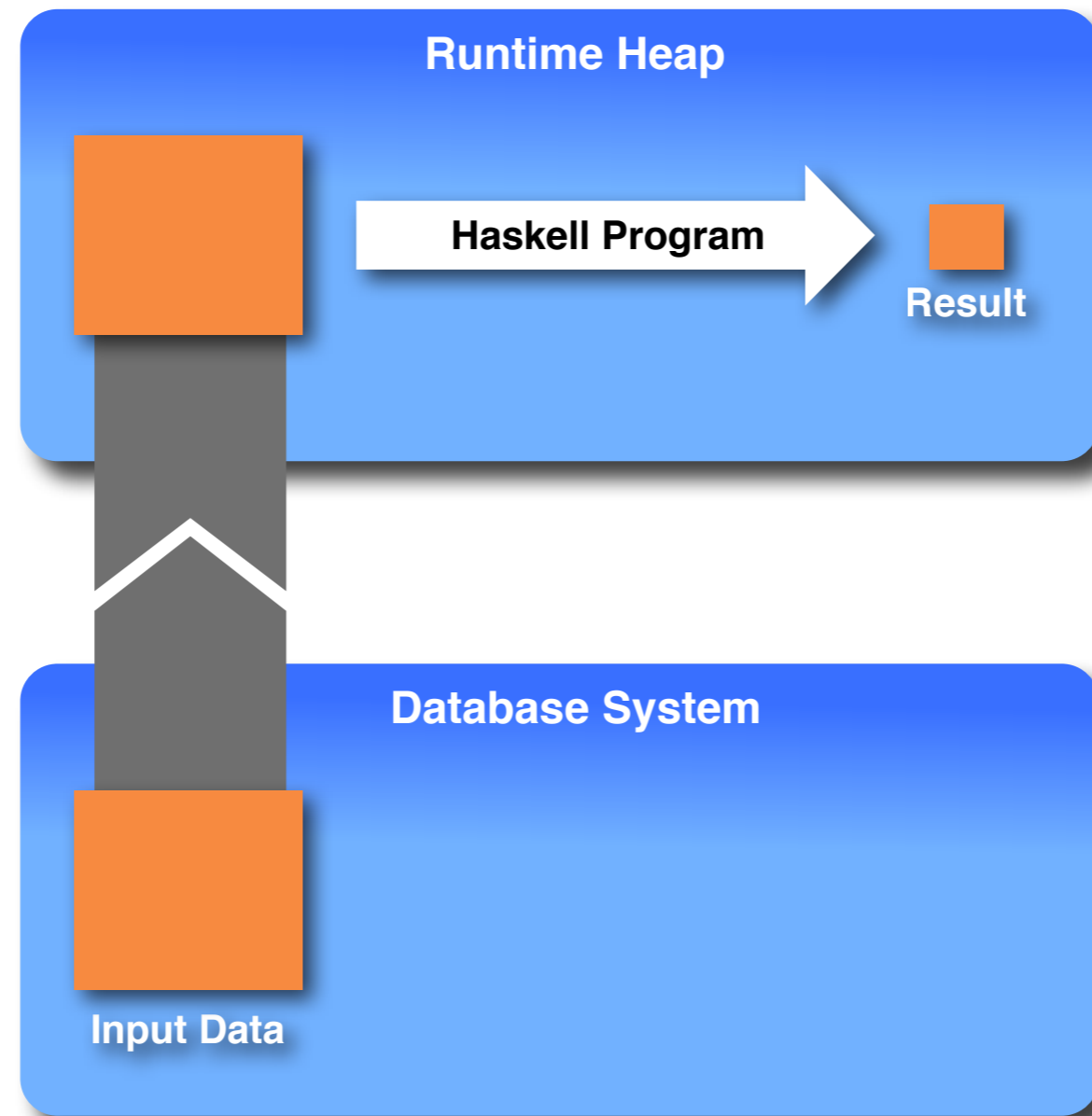
IFL 2010, Utrecht University

EBERHARD KARLS

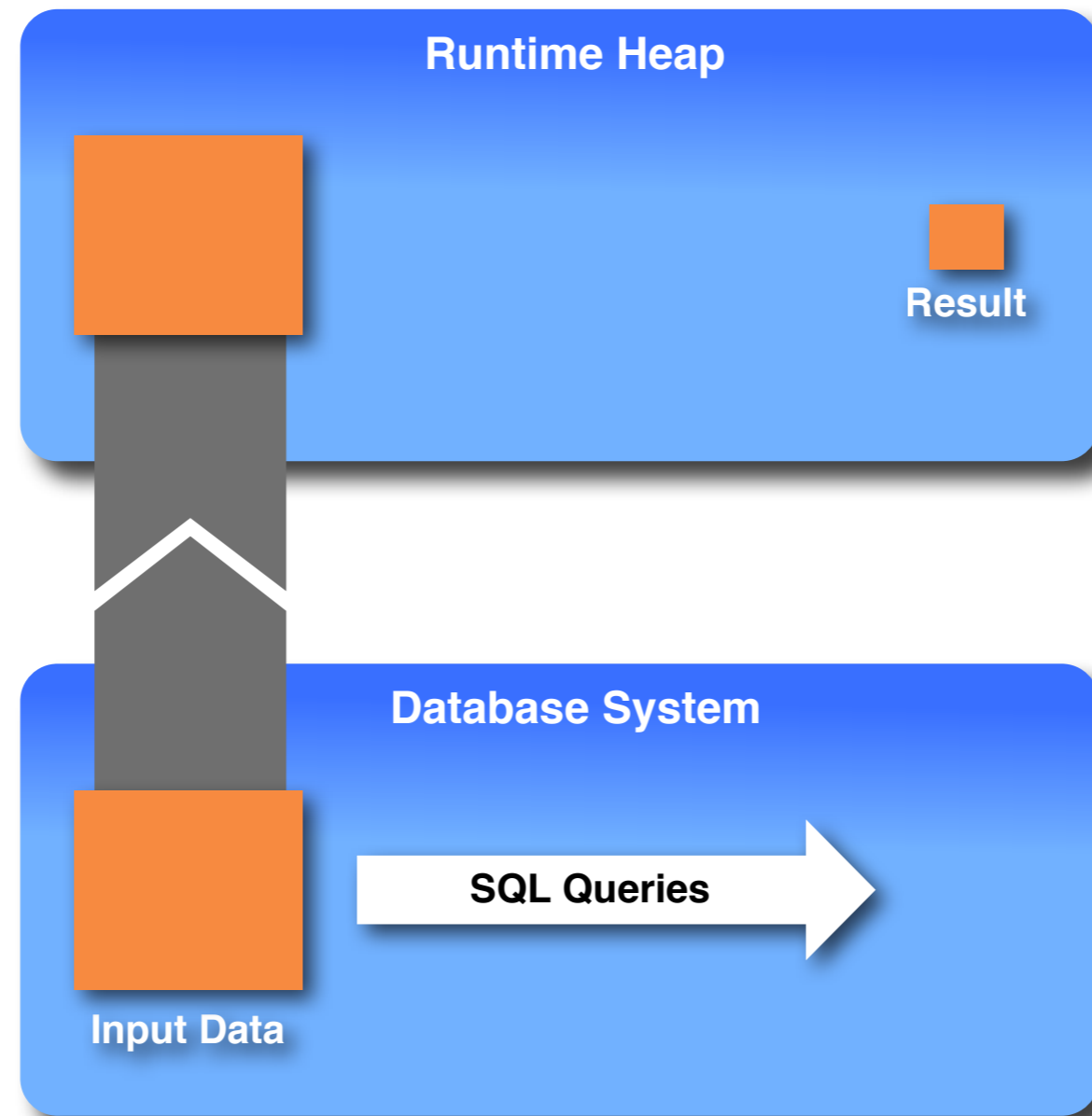
UNIVERSITÄT
TÜBINGEN



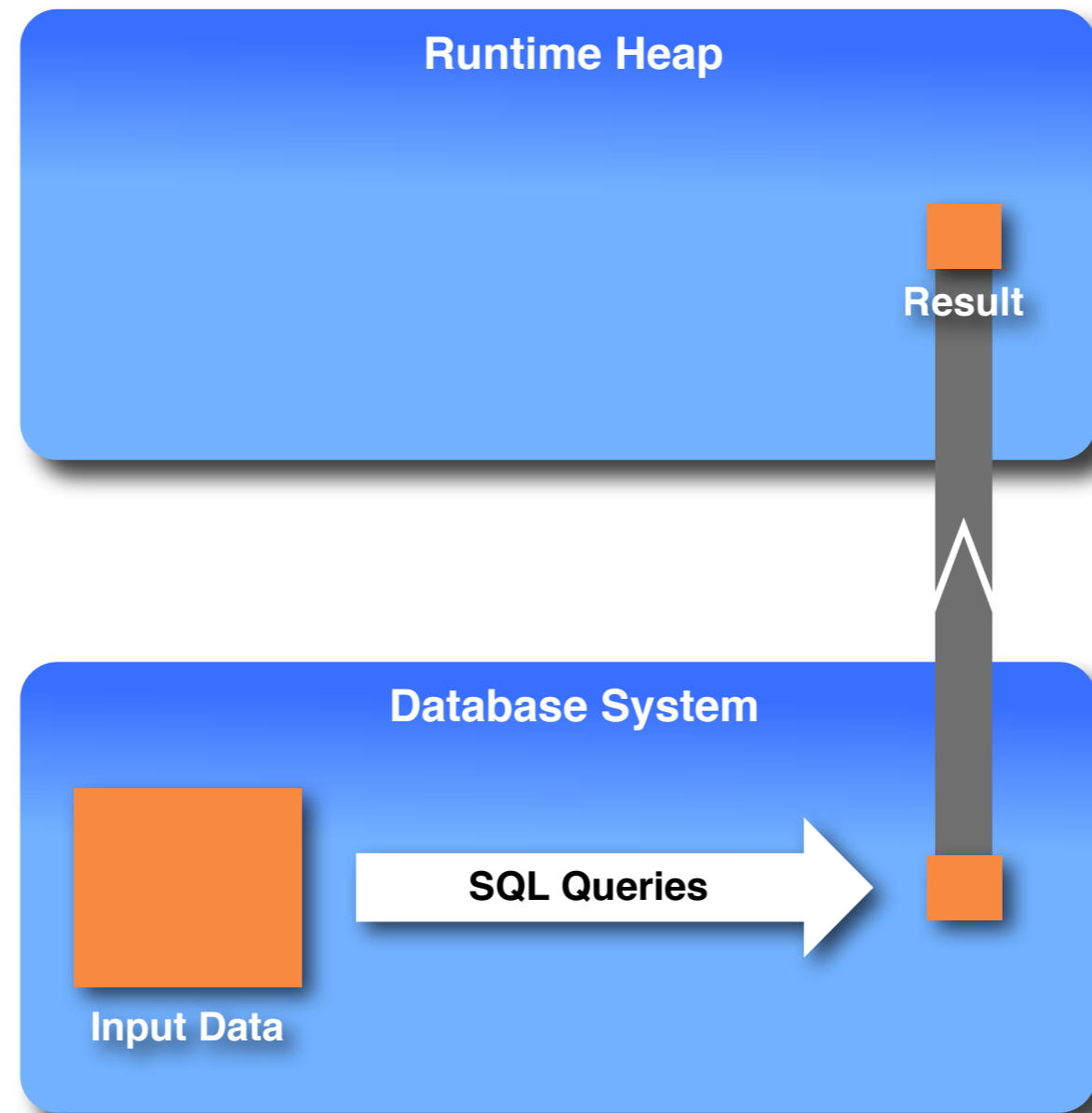
Haskell turns into SQL



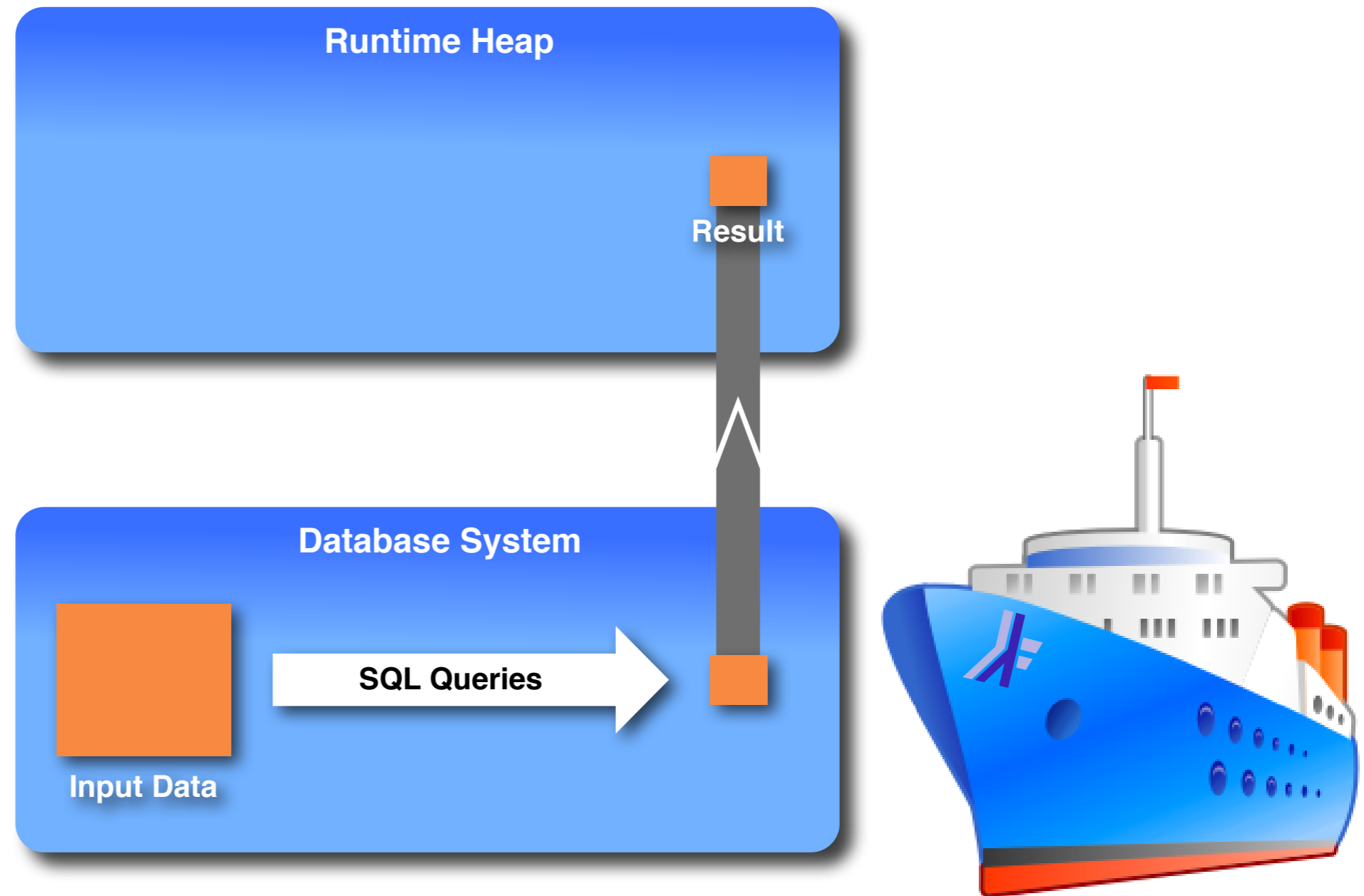
Haskell turns into SQL



Haskell turns into SQL



Haskell turns into SQL



Database-supported program execution

State of the Art

Facilities	Category
SQL	QLA
LINQ	LIN
Links	LIN
Haskell DB	LIB
Ferry	LIB

Feature
Respects list order
Supports data nesting
Avoids query avalanches
Is statically type-checked
Guarantees translation to SQL
Has compositional syntax and semantics

State of the Art

Facilities	Category
SQL	QLA
LINQ	LIN
Links	LIN
Haskell DB	LIB
Ferry	LIB

Feature
Respects list order
Supports data nesting
Avoids query avalanches
Is statically type-checked
Guarantees translation to SQL
Has compositional syntax and semantics

State of the Art

Facilities	Category
SQL	QLA
LINQ	LIN
Links	LIN
Haskell DB	LIB
Ferry	LIB

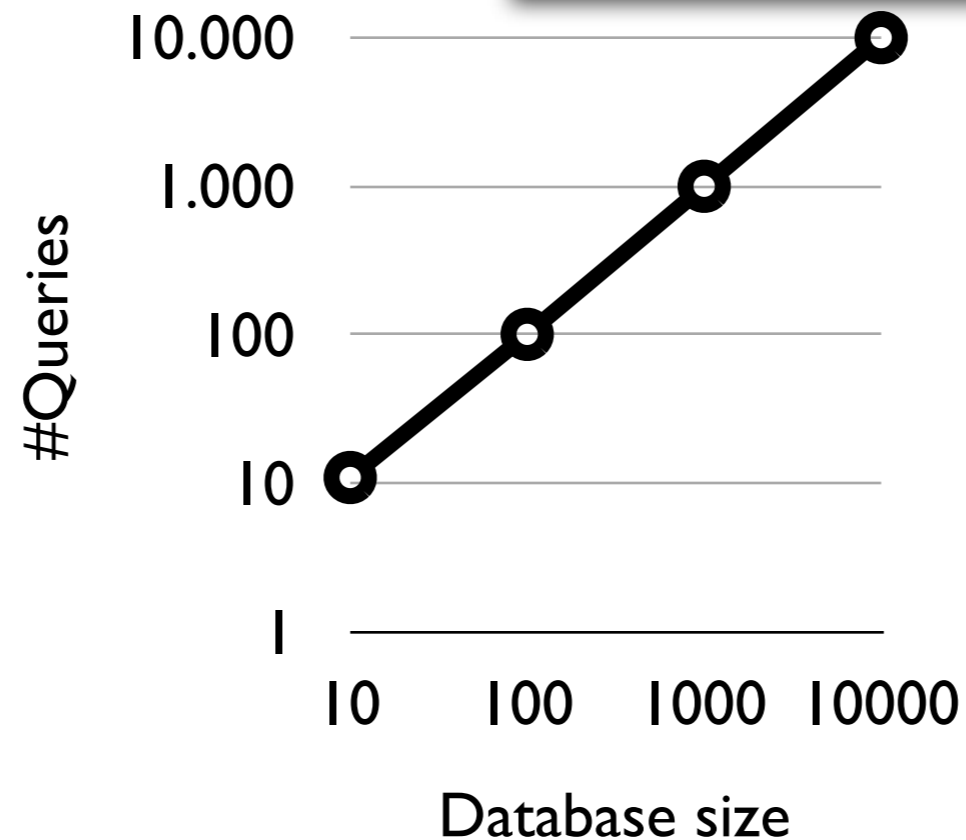
Feature
Respects list order
Supports data nesting
Avoids query avalanches
Is statically type-checked
Guarantees translation to SQL
Has compositional syntax and semantics

`take n e ++ drop n e ≠ e`

State of the Art

Facilities	Category
SQL	QLA
LINQ	LIN
Links	LIN
Haskell DB	LIB
Ferry	LIB

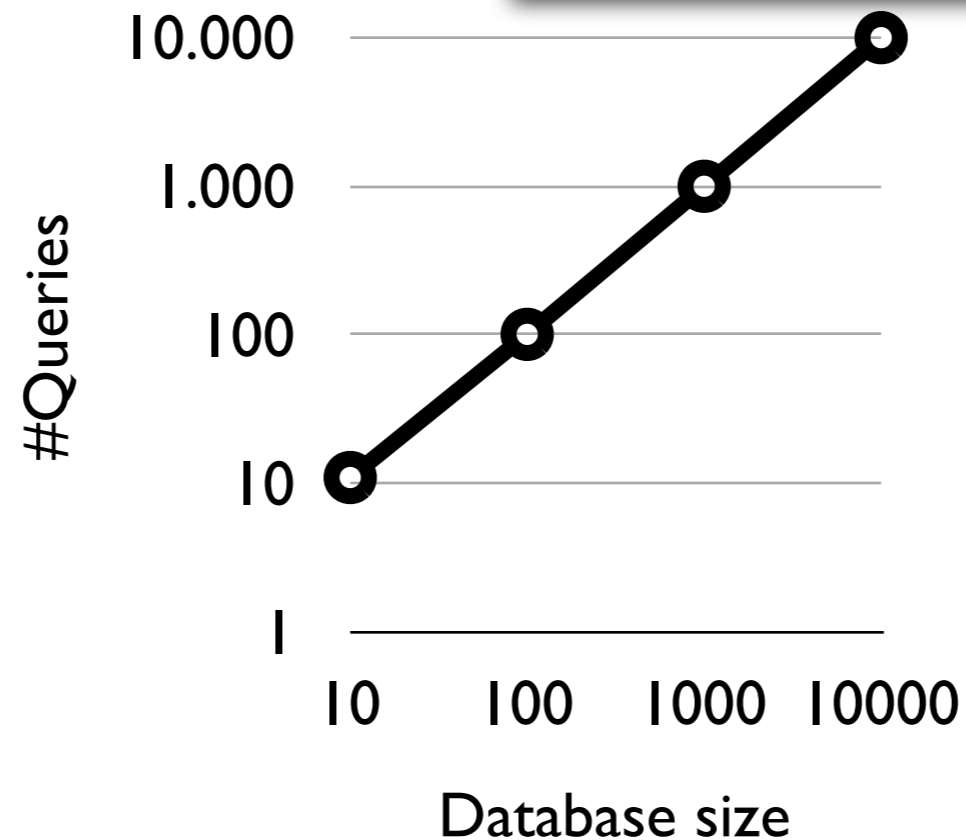
Feature
Respects list order
Supports data nesting
Avoids query avalanches
Is statically type-checked
Guarantees translation to SQL
Has compositional syntax and semantics



State of the Art

Facilities	Category
SQL	QLA
LINQ	LIN
Links	LIN
Haskell DB	LIB
Ferry	LIB

Feature
Respects list order
Supports data nesting
Avoids query avalanches
Is statically type-checked
Guarantees translation to SQL
Has compositional syntax and semantics



State of the Art

Facilities	Category
SQL	QLA
LINQ	LIN
Links	LIN
Haskell DB	LIB
Ferry	LIB

Feature
Respects list order
Supports data nesting
Avoids query avalanches
Is statically type-checked
Guarantees translation to SQL
Has compositional syntax and semantics

Stick with Comprehensions

What features can a facility have in a category?

```
hasFeatures :: String → [String]
hasFeatures f = [ feat | (fac,feat) ← features, fac ≡ f ]
```

Stick with Comprehensions

What features can a facility have in a category?

```
hasFeatures :: String → [String]
```

```
hasFeatures f = [ feat | (fac,feat) ← features, fac ≡ f ]
```

```
means :: String → String
```

```
means f = head [ mean | (feat,mean) ← meanings, feat ≡ f ]
```

Stick with Comprehensions

What features can a facility have in a category?

```
hasFeatures :: String → [String]
hasFeatures f = [ feat | (fac,feat) ← features, fac ≡ f ]

means :: String → String
means f = head [ mean | (feat,mean) ← meanings, feat ≡ f ]

query :: [(String , [String])]
query =
  [ (the cat, nub $ concat $ map (map means ∘ hasFeatures) fac)
  | (fac, cat) ← facilities, then group by cat ]
```

Stick with Comprehensions

What features can a facility have in a category?

```
hasFeatures :: Q String → Q [String]
```

```
hasFeatures f = [$qc | feat | (fac,feat) ← table features, fac ≡ f |]
```

```
means :: Q String → Q String
```

```
means f = head [$qc | mean | (feat,mean) ← table meanings, feat ≡ f |]
```

```
query :: IO [(String , [String])]
```

```
query = fromQ connection
```

```
[$qc | (the cat, nub $ concat $ map (map means ∘ hasFeatures) fac)  
| (fac, cat) ← table facilities, then group by cat |]
```

Stick with Combinators

map, filter, head, tail, length, zip, sortWith, the, ...

Stick with Combinators

map, filter, head, tail, length, zip, sortWith, the, ...

Map from prelude:

$\text{map} :: (a \rightarrow b) \rightarrow [a] \rightarrow [b]$

Stick with Combinators

map, filter, head, tail, length, zip, sortWith, the, ...

Map from prelude:

$\text{map} :: (a \rightarrow b) \rightarrow [a] \rightarrow [b]$

Map from Ferry:

$\text{map} :: (\text{QA } a, \text{QA } b) \Rightarrow (\text{Q } a \rightarrow \text{Q } b) \rightarrow \text{Q } [a] \rightarrow \text{Q } [b]$

Stick with Combinators

map, filter, head, tail, length, zip, sortWith, the, ...

Map from prelude:

$\text{map} :: (a \rightarrow b) \rightarrow [a] \rightarrow [b]$

Map from Ferry:

$\text{map} :: (\text{QA } a, \text{QA } b) \Rightarrow (\text{Q } a \rightarrow \text{Q } b) \rightarrow \text{Q } [a] \rightarrow \text{Q } [b]$

Restrict to
queryable types

Stick with Combinators

map, filter, head, tail, length, zip, sortWith, the, ...

Map from prelude:

$\text{map} :: (a \rightarrow b) \rightarrow [a] \rightarrow [b]$

Map from Ferry:

$\text{map} :: (\text{QA } a, \text{QA } b) \Rightarrow (\text{Q } a \rightarrow \text{Q } b) \rightarrow \text{Q } [a] \rightarrow \text{Q } [b]$

Restrict to
queryable types

Q datatype
represents query

Stick with Combinators

map, filter, head, tail, length, zip, sortWith, the, ...

Map from prelude:

$\text{map} :: (a \rightarrow b) \rightarrow [a] \rightarrow [b]$

Map from Ferry:

$\text{map} :: (\text{QA } a, \text{QA } b) \Rightarrow (\text{Q } a \rightarrow \text{Q } b) \rightarrow \text{Q } [a] \rightarrow \text{Q } [b]$

Restrict to queryable types

Q datatype represents query

A few combinators are not supported (yet) (e.g. foldr, foldl)

A Haskell View of the Relational Data Model

```
CREATE TABLE "Facilities"  
  (facility varchar(100) NOT NULL,  
   category varchar(100) NOT NULL);
```



```
data Facility = Facility  
  {facility :: String,  
   category :: String}  
type Facilities = [Facility]
```

A Haskell View of the Relational Data Model

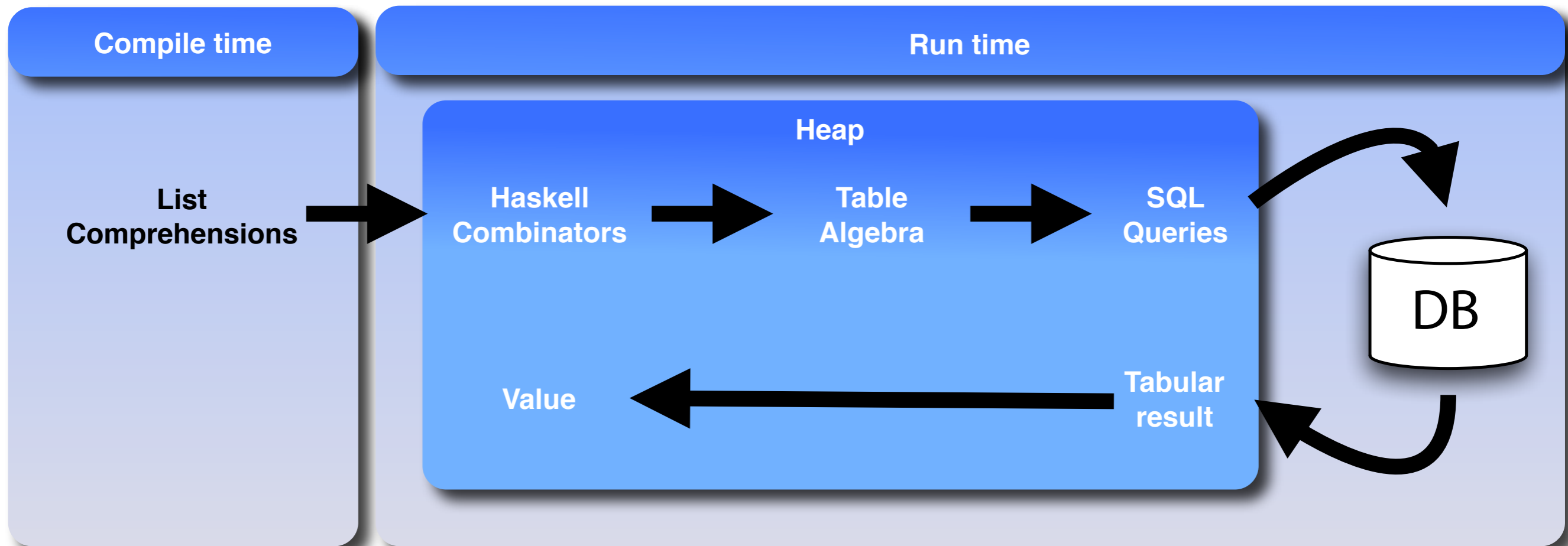
```
CREATE TABLE "Facilities"  
  (facility varchar(100) NOT NULL,  
   category varchar(100) NOT NULL);
```



```
data Facility = Facility  
  {facility :: String,  
   category :: String}  
type Facilities = [Facility]
```

```
Int, Bool, String, Double, [a], (),  
  (a1, ..., an), {x1::a1, ..., xn::an}
```

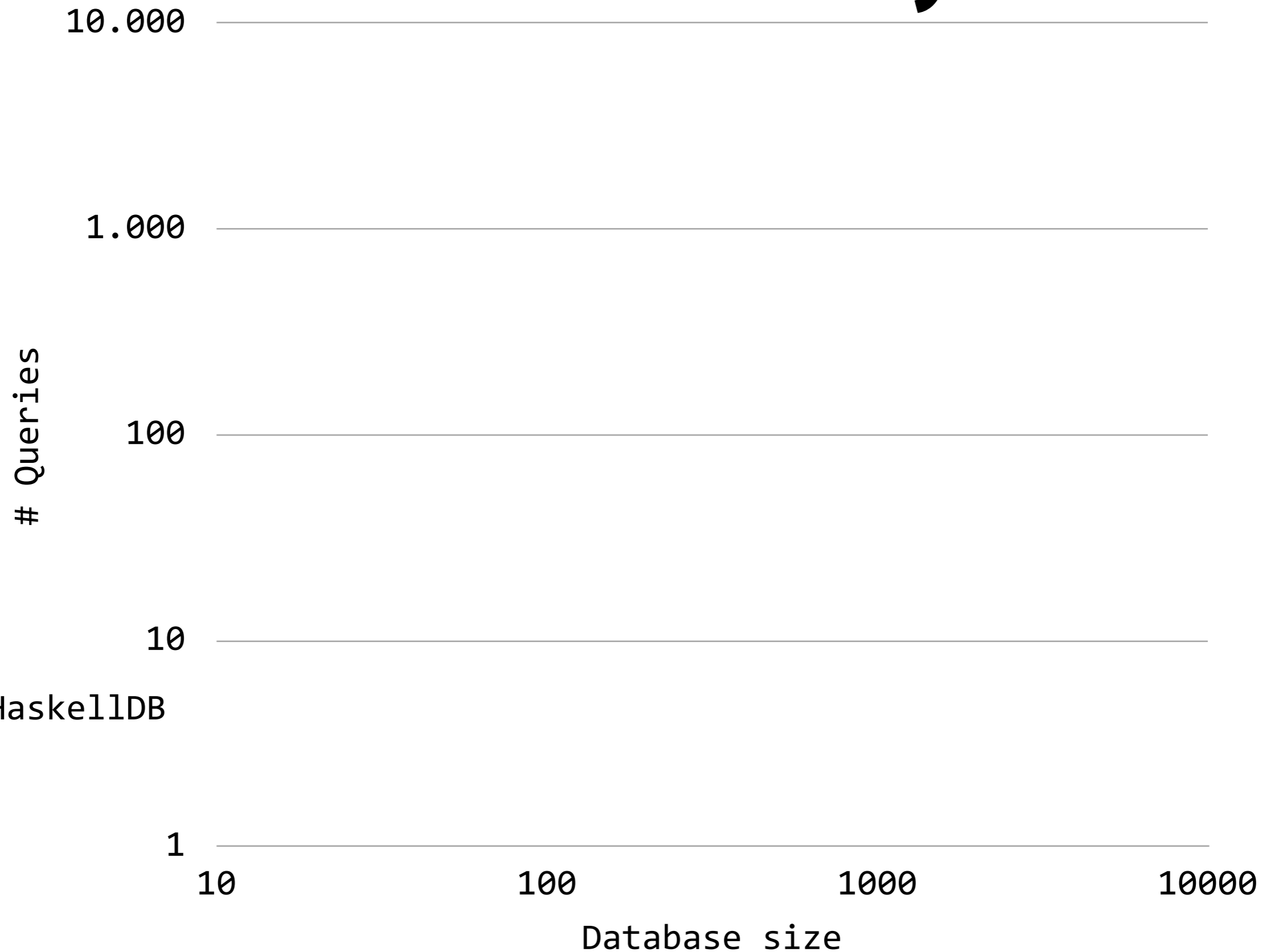
Turning Haskell into SQL



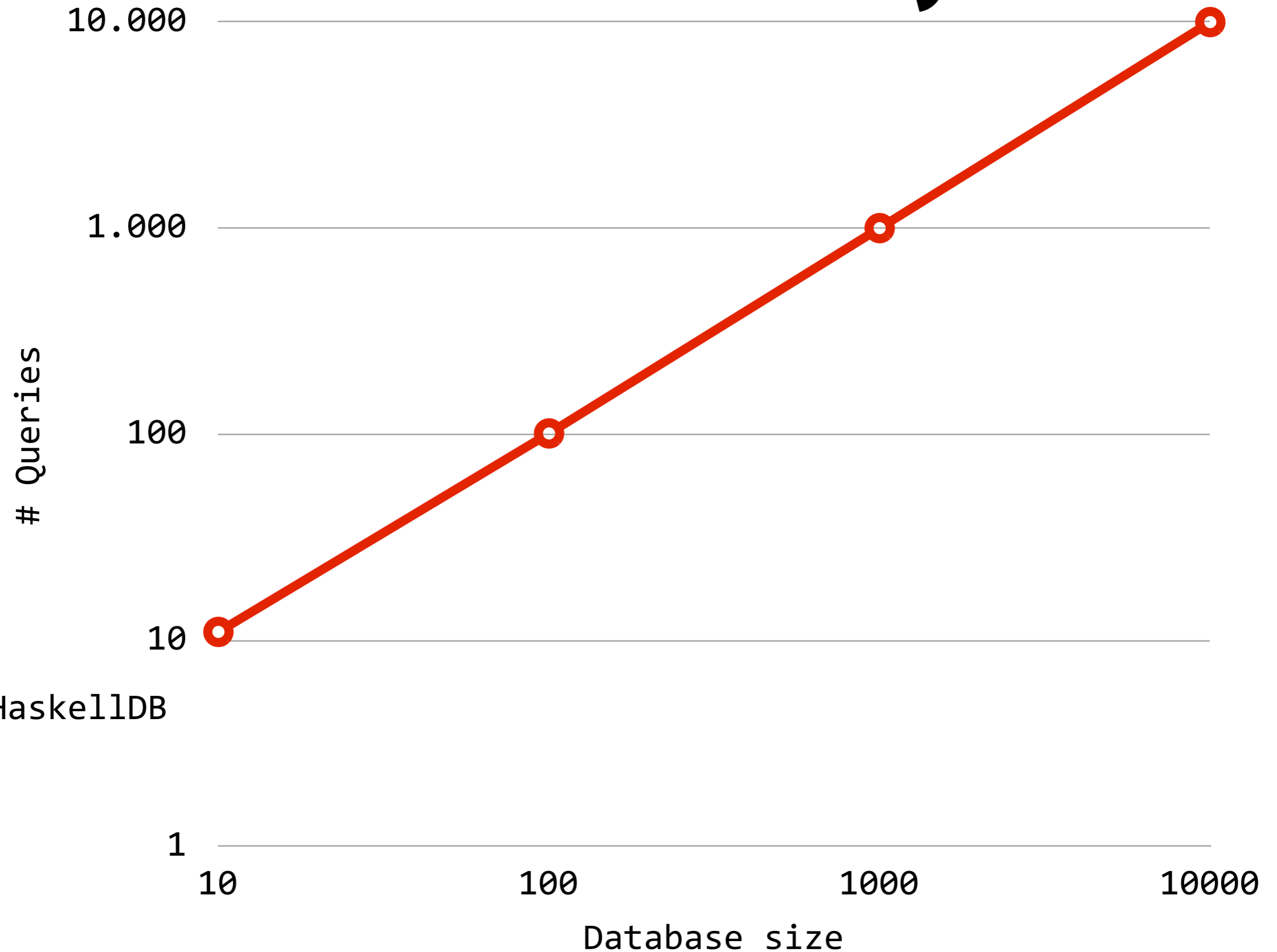
Avalanche safety

- LINQ/HaskellDB
- Ferry

Avalanche safety

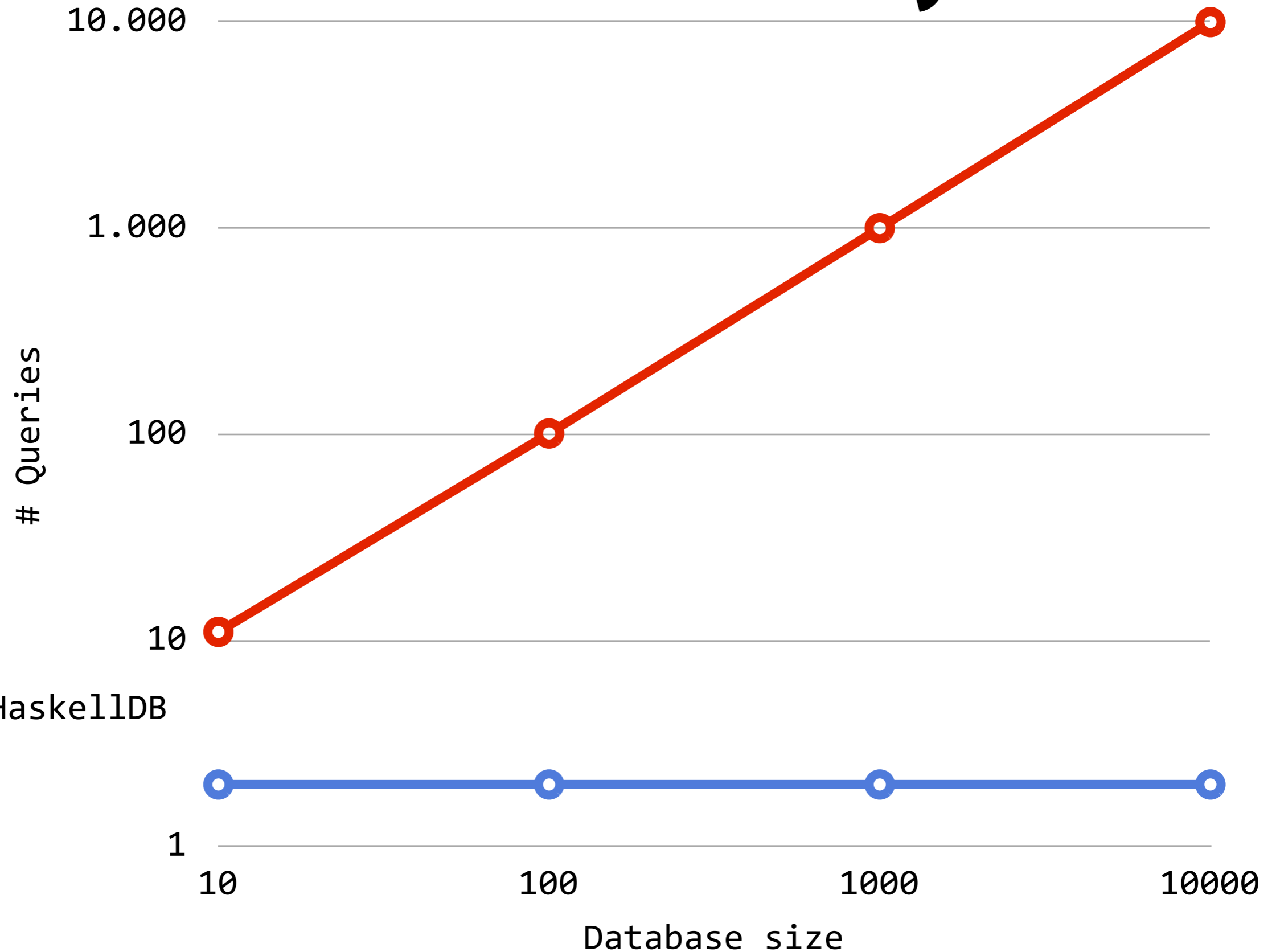


Avalanche safety



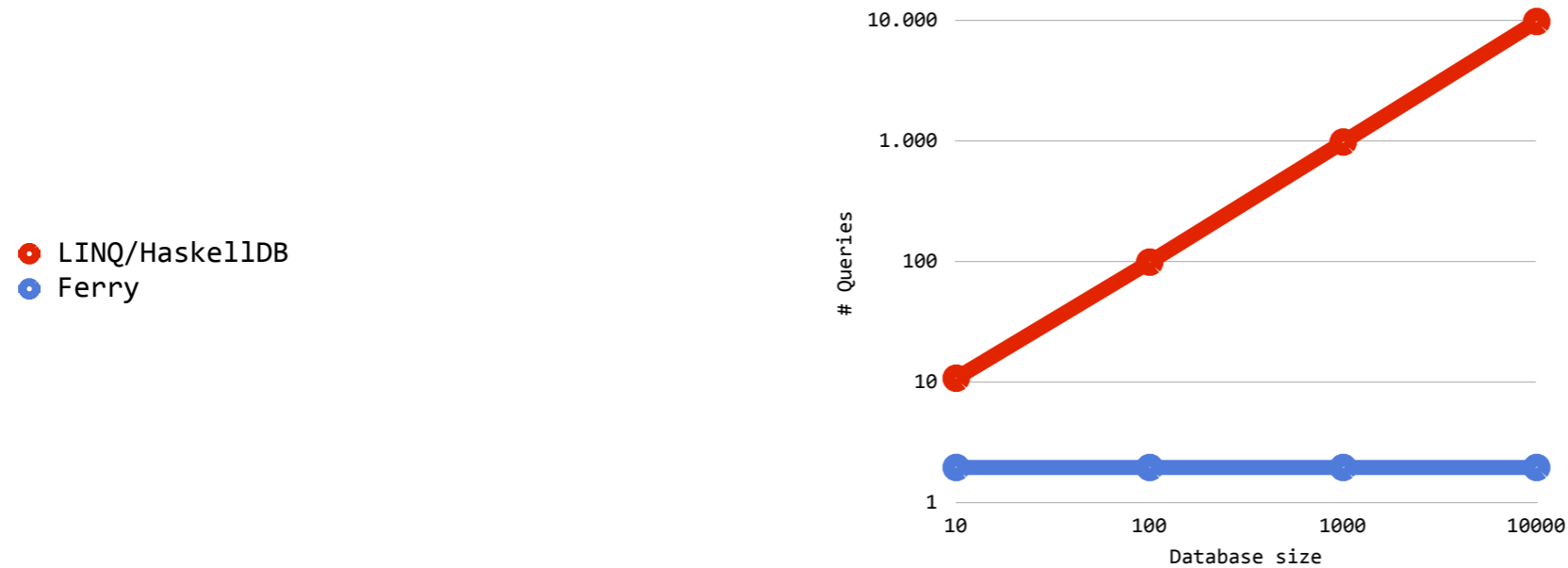
○ LINQ/HaskellDB
○ Ferry

Avalanche safety



○ LINQ/Haskell1DB
○ Ferry

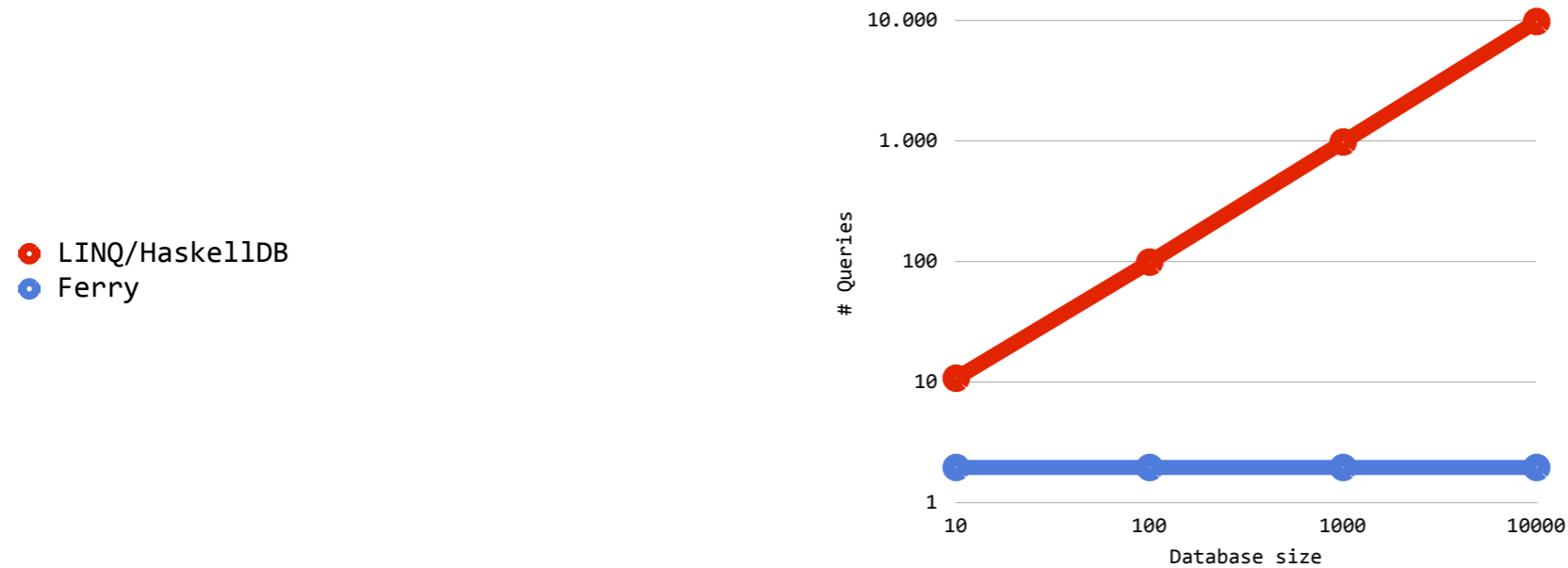
Avalanche safety



Program's result type determines # of queries

`[(String, [String])]`

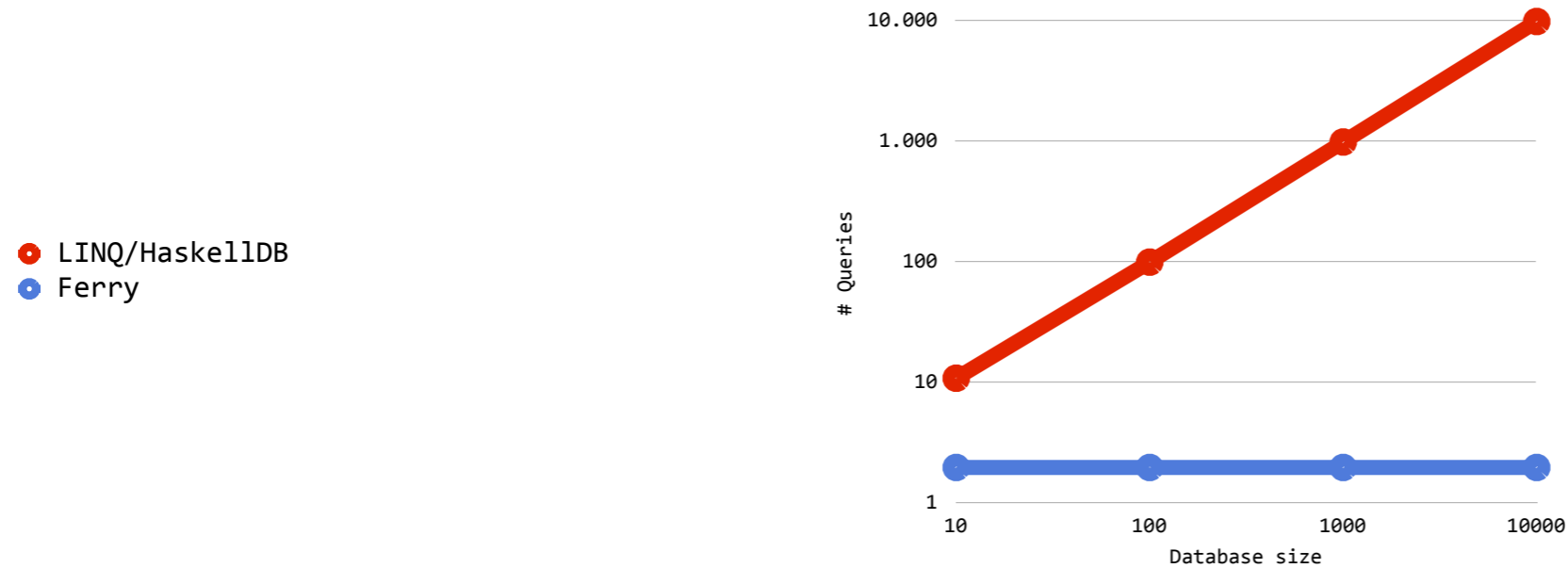
Avalanche safety



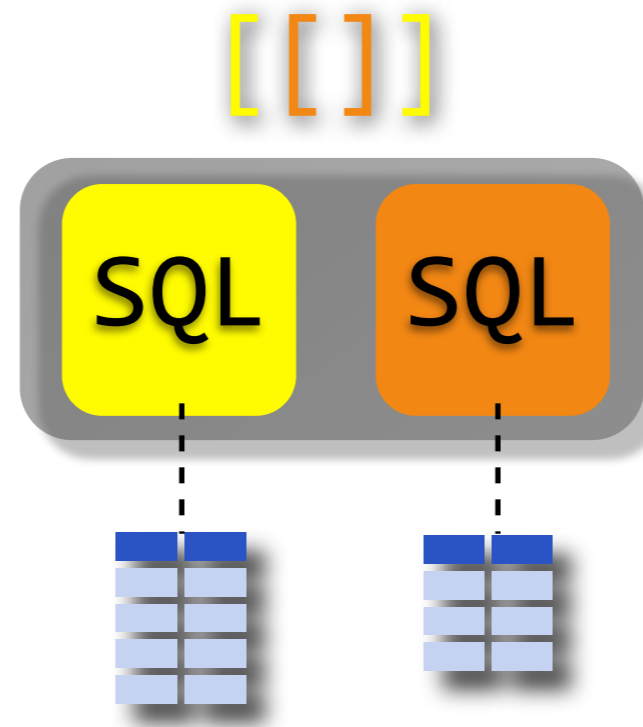
Program's result type determines # of queries

`[(String, [String])]`

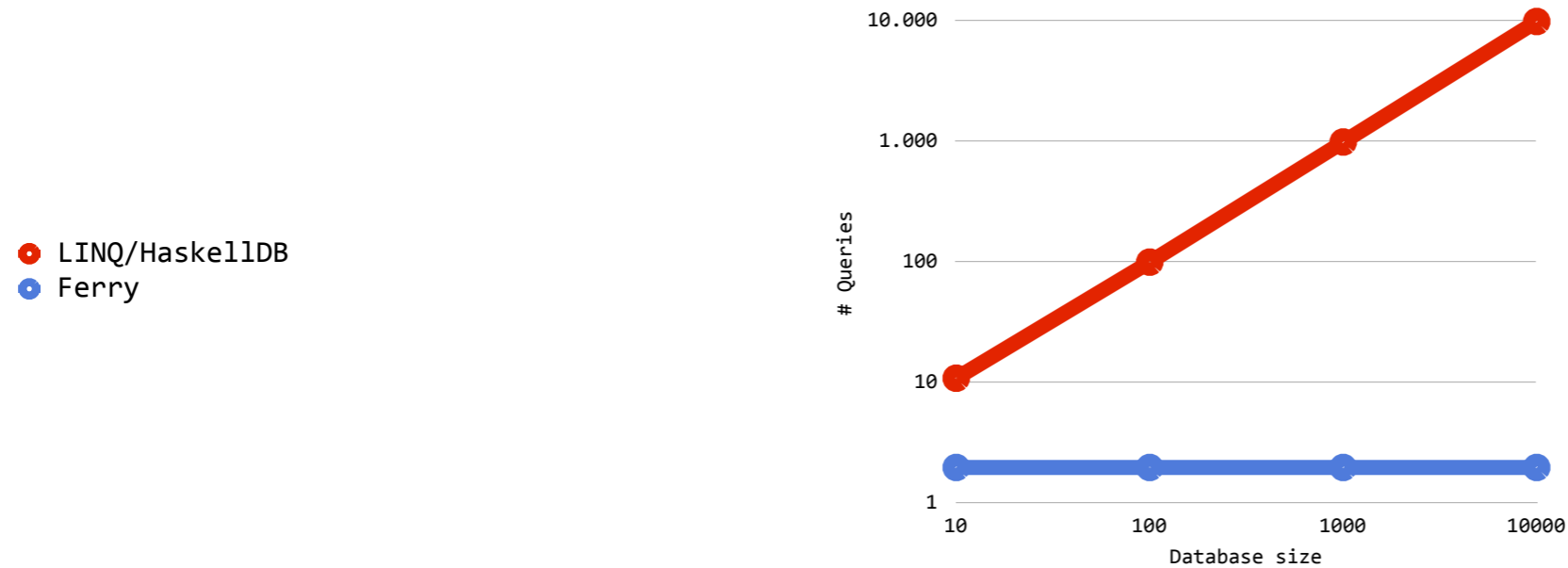
Avalanche safety



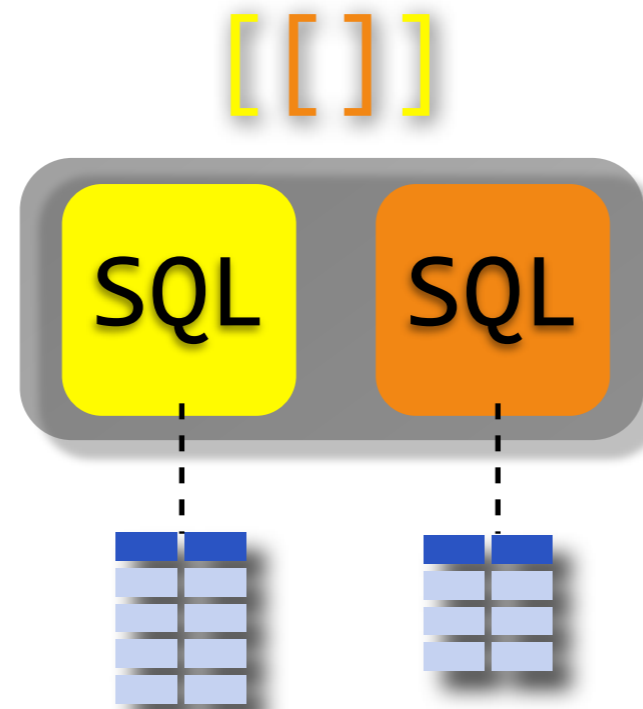
Program's result type determines # of queries



Avalanche safety



Program's result type determines # of queries



Queries in a bundle are independent:
concurrent / on-demand execution OK

Future work

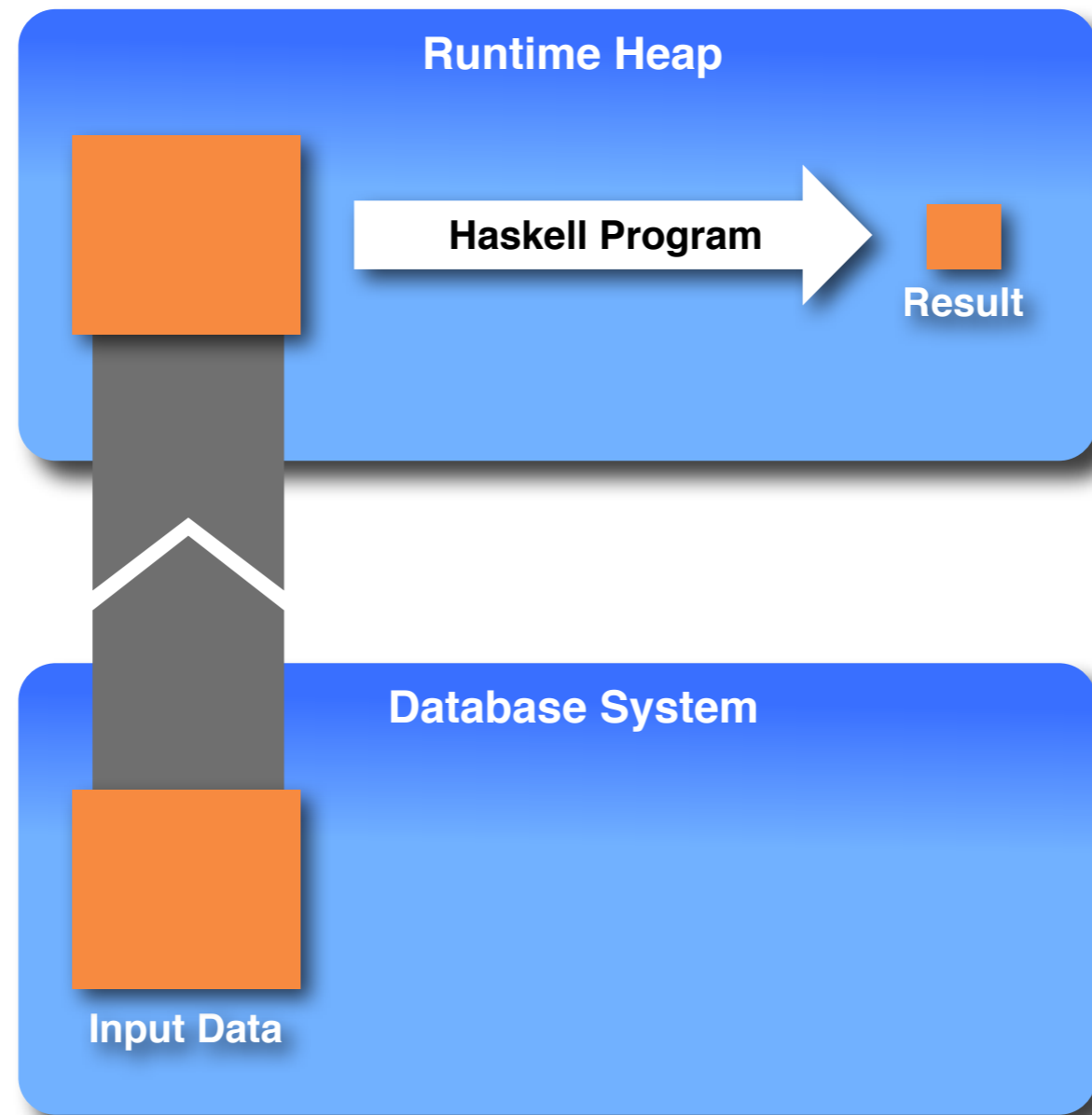
- Functions as result of subquery
- User-defined data types
- Admit (limited) recursive functions
- Even tighter integration with Haskell
- Explore the many similarities with Data Parallel Haskell (DPH)

Haskell on a Ferry

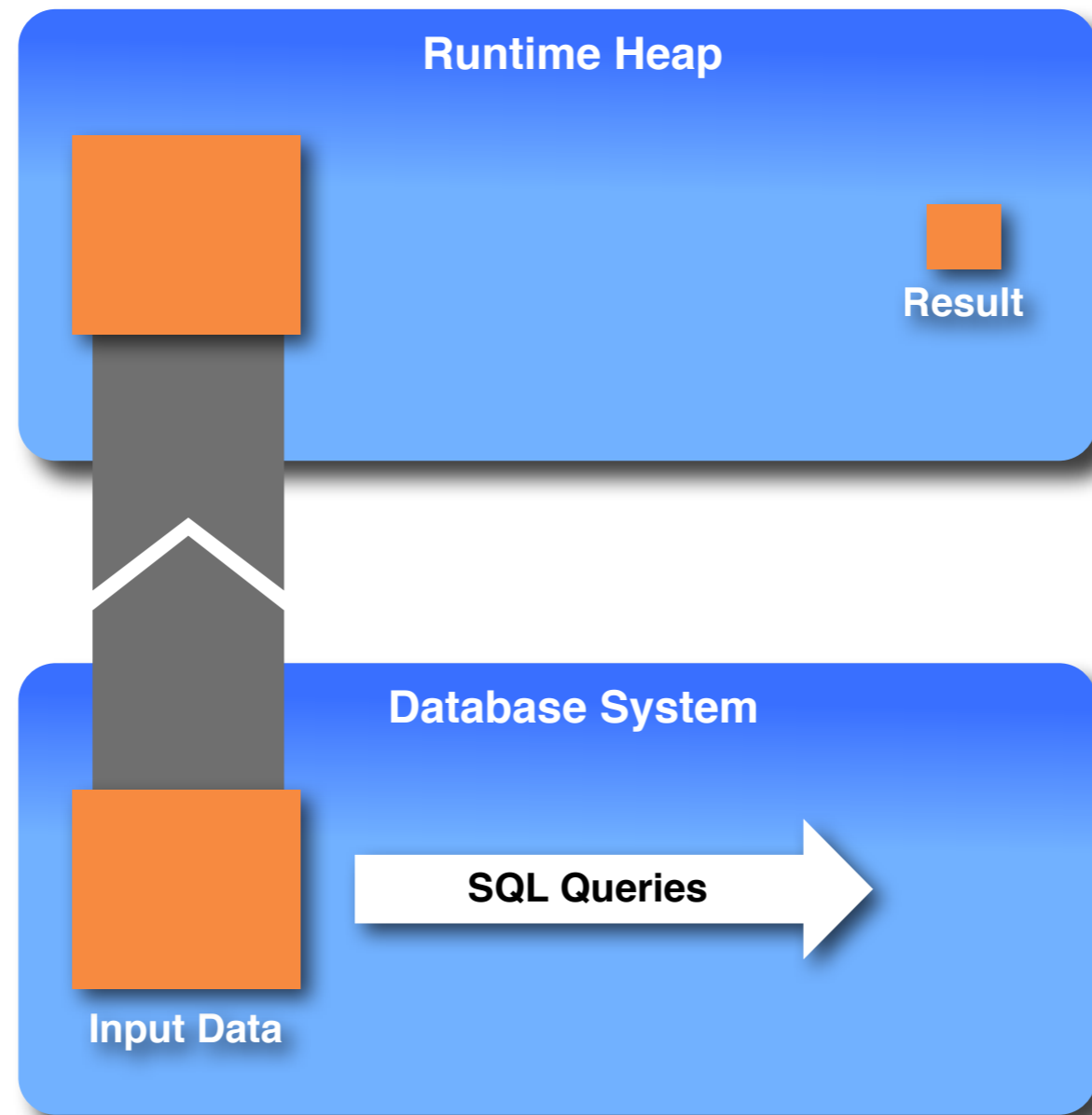
- The database as coprocessor
- Familiar Haskell syntax and semantics
- Support nested data and ordered lists
- Guaranteed avalanche safety



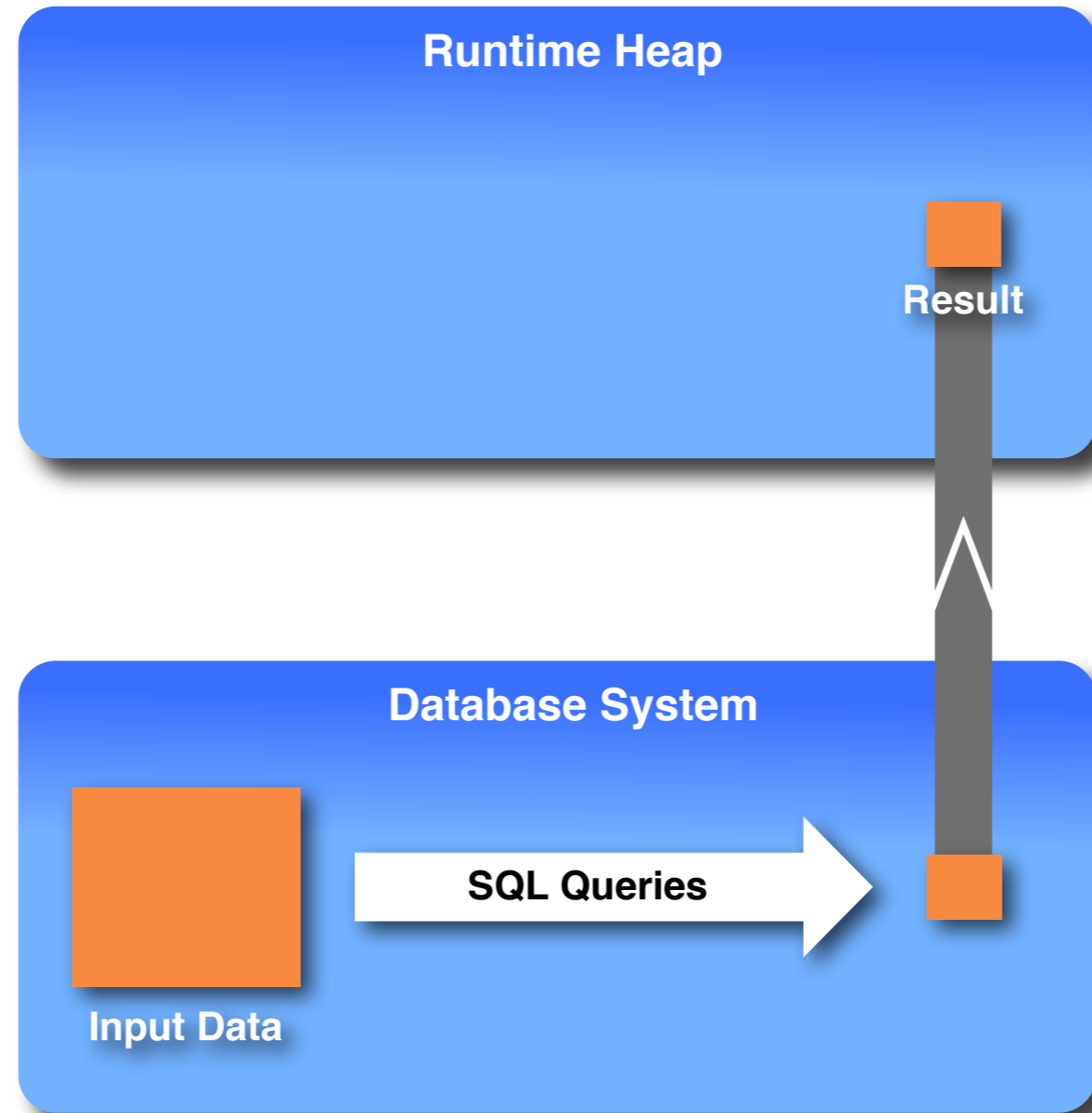
Haskell turns into SQL



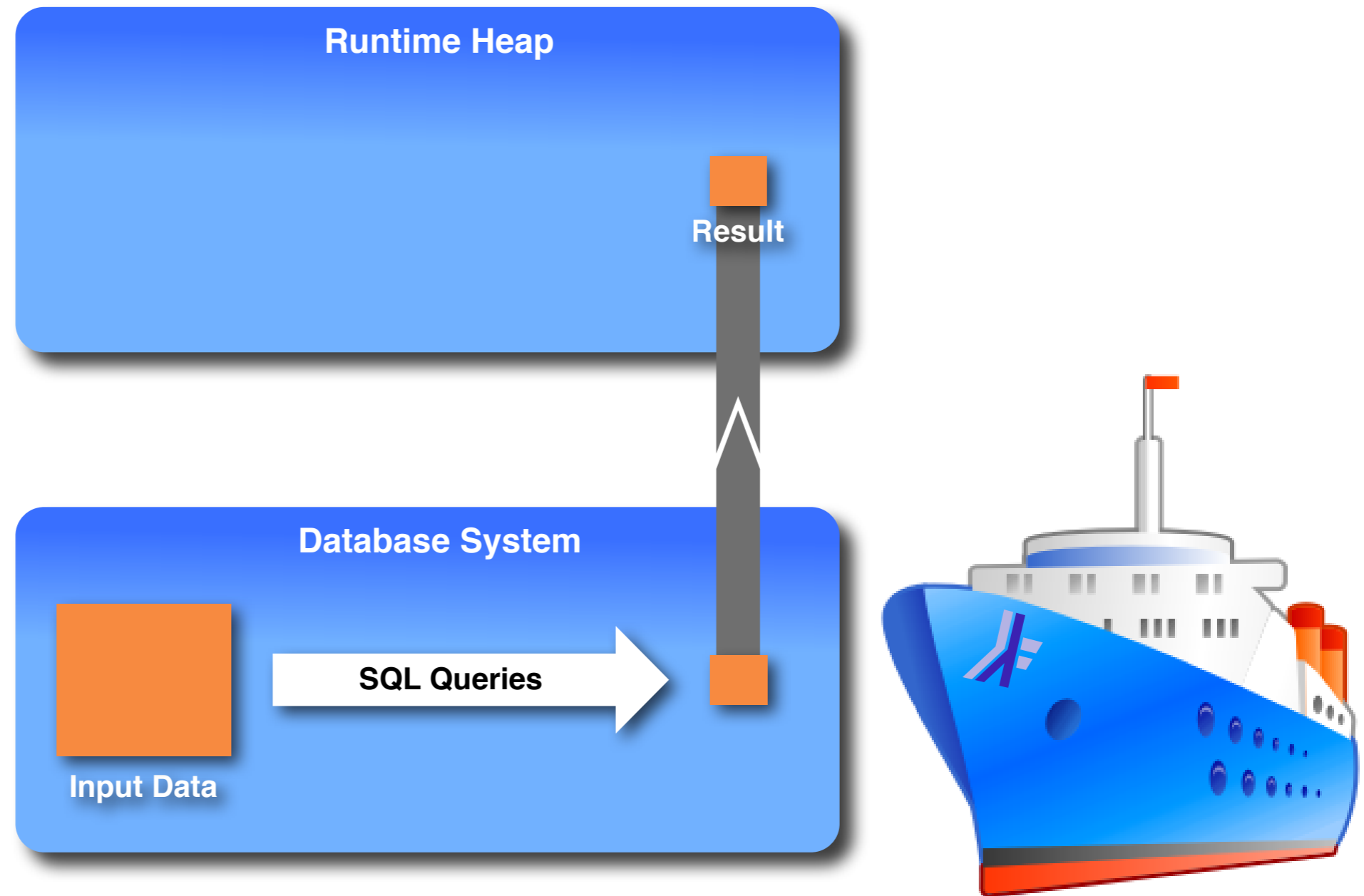
Haskell turns into SQL



Haskell turns into SQL



Haskell turns into SQL



www.ferry-lang.org