



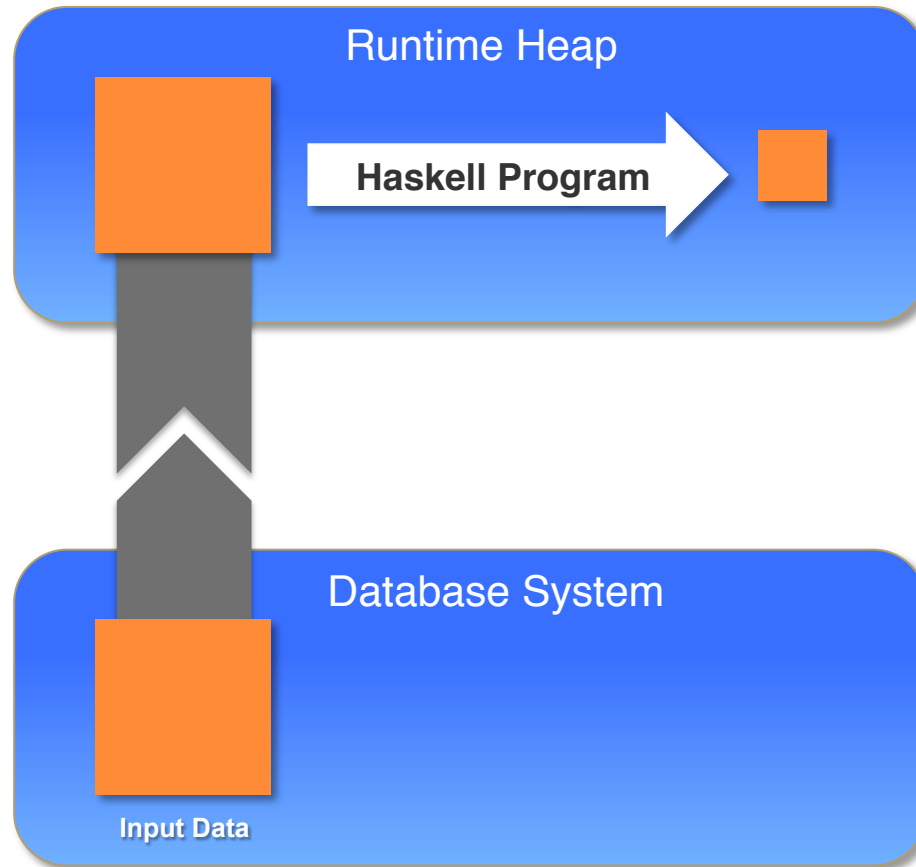
## Database Supported Haskell

Jeroen Weijers (jeroen.weijers@uni-tuebingen.de)

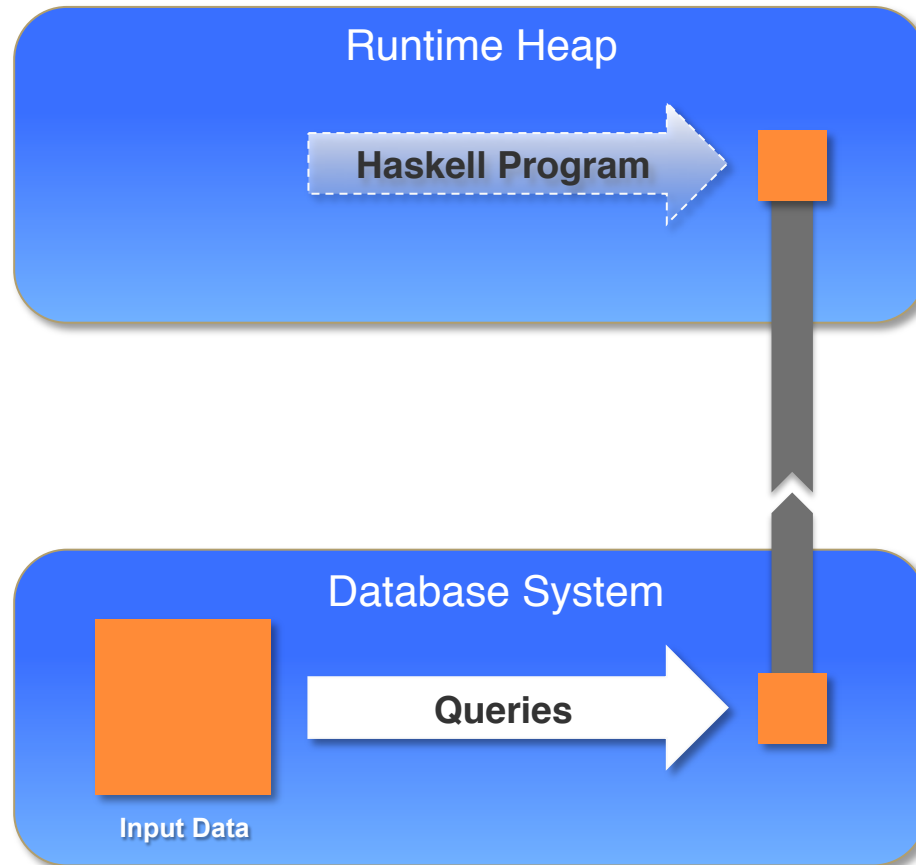
<http://www-db.informatik.uni-tuebingen.de/>

23 September 2011 – Haskell Implementors' Workshop

# Haskell turns into SQL



# Haskell turns into SQL



## Database-supported program execution

# A DSH Query

Name	Department	Salary
Dilbert	Eng	80
Alice	Eng	100
Wally	Eng	40
Catbert	HR	150
Dogbert	Con	500
Ratbert	HR	90

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

```
query = do
```

```
  v ← fromQ connection $
```

```
    [(the dept, sum salary, name)
```

```
      | (name, dept, salary) ← table "employees"
```

```
      , then group by dept using groupWith
```

```
      , then sortWith by (sum salary)]
```

```
  printData v
```

```
  return v
```

Monad  
Comprehension!

# 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 DSH:

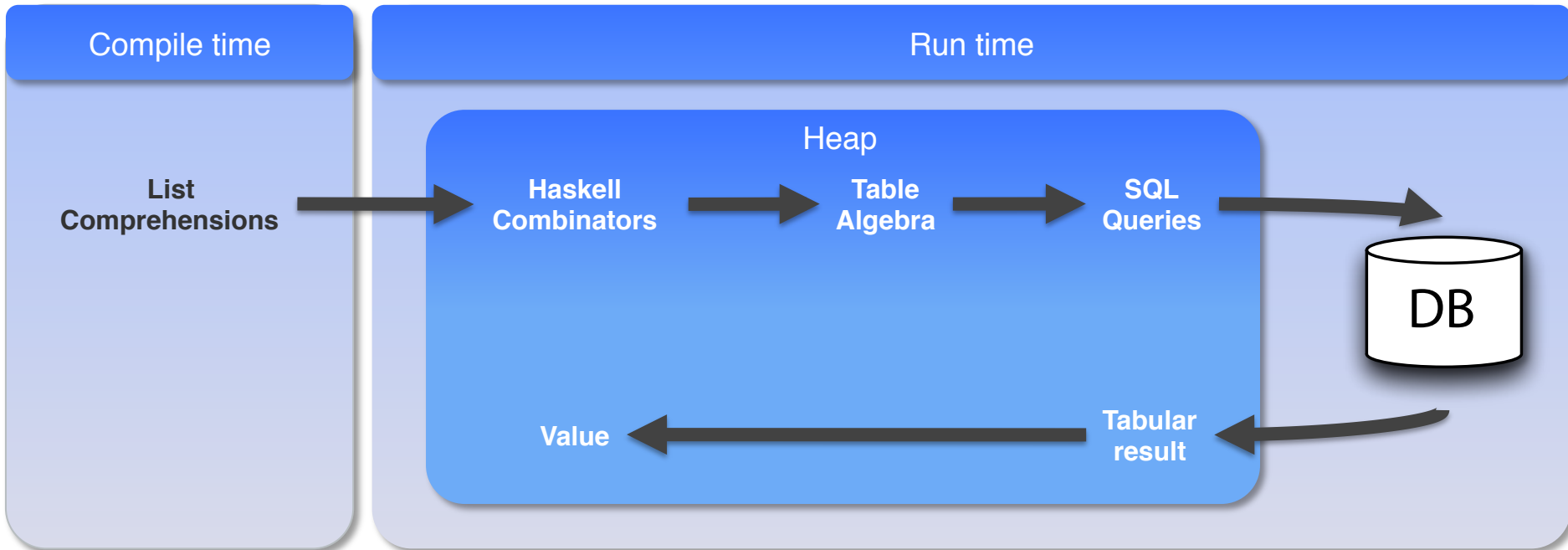
$\text{map} :: (\text{QA } a, \text{QA } b) \Rightarrow (\text{Q } a \rightarrow \text{Q } b) \rightarrow \text{Q } [a] \rightarrow \text{Q } [b]$

Restrict to  
queryable types

Q datatype represents  
query

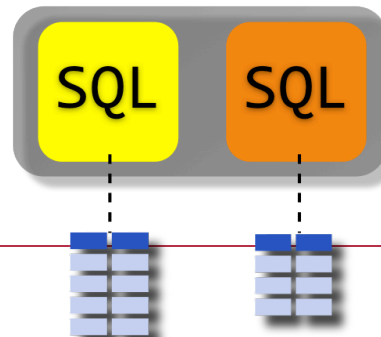
A few combinators are not supported (yet) (e.g. foldr, foldl)

# Turning Haskell into SQL



Program's result type determines # of queries:

```
[(String, Int, [String])]
```



# Haskell → SQL with loop-lifting

$$\begin{array}{c}
 \Gamma, loop \vdash e_1 \Rightarrow (q_1, cs_1, ts_1) \quad q'_v = \text{rownum}_{\text{iter}, \text{pos}}^{\text{inner}}(q_1) \\
 q_v = \pi_{\text{iter}: \text{inner}, \text{pos}: \text{pos}', \text{ln } cs_1}(@_{\text{pos}': 1}(q'_v)) \quad \text{map}_v = \pi_{\text{outer}: \text{iter}, \text{inner}, \text{pos}': \text{pos}}(q'_v) \\
 \Gamma_v = \text{map}(\lambda(x, (q_1, cs_1, ts)) \rightarrow (x, (\pi_{\text{iter}: \text{inner}, \text{pos}, \text{ln } cs}(q_1 \bowtie_{\text{iter}: \text{outer}} \text{map}_v), cs_1, ts))) \Gamma \\
 \Gamma_v [x \mapsto (q_v, cs_1, ts_1)], loop_v \vdash e_2 \Rightarrow (q_2, cs_2, ts_2) \\
 q = \pi_{\text{iter}: \text{outer}, \text{pos}: \text{pos}', \text{ln } cs_2}(q_2 \bowtie_{\text{iter}: \text{inner}} \text{map}_v) \\
 \hline
 \Gamma, loop \vdash \text{map}(\lambda x \rightarrow e_2) e_1 \Rightarrow (q, cs_2, ts_2) \quad \text{[ac-map]}
 \end{array}$$

$$\begin{array}{c}
 \Gamma, loop \vdash e_3 \Rightarrow (q_3, cs_3, ts_3) \\
 (q'_v, q_v, \text{map}', loop', \Gamma') = \text{mapForward } \Gamma \ q_3 \ cs_3 \\
 \Gamma' [x_1 \mapsto (q_v, cs_3, ts_3)], loop' \vdash e_1 \Rightarrow (q_1, cs_1, ts_1) \\
 \Gamma' [x_2 \mapsto (q_v, cs_3, ts_3)], loop' \vdash e_2 \Rightarrow (q_2, cs_2, ts_2) \\
 cs'_2 = \text{incr } cs_2 \ (\text{card } cs_1) \\
 q = \text{rowrank}_{\text{iter}, \text{ln } cs'_2}^{s_k}((\pi_{\text{iter}, \text{pos}, \text{inner}}(q_v)) \bowtie_{\text{inner}: \text{iter}'}((\pi_{\text{iter}'': \text{iter}, \text{ln } cs_1}(q_1)) \bowtie_{\text{iter}'': \text{iter}'}(\pi_{\text{iter}': \text{iter}, \text{ln } cs'_2}(q_2)))) \\
 n = \text{length } cs_1 \\
 q_{\text{out}} = \text{distinct } \pi_{\text{iter}, \text{pos}: g_k, \text{item}_1: g_k, \dots, \text{item}_n: g_k}(q) \\
 cs = \{n \rightarrow i \mid (n \rightarrow -, i) \leftarrow \text{zip } cs_1 [1..]\} \\
 cs_i = \text{lookup } cs_1 \ i \quad \text{where } i \in [1..n] \\
 q_i = \pi_{\text{iter}: g_k, \text{decr } cs_i: cs_i, \text{pos}}(q) \quad \text{where } i \in [1..n] \\
 ts_i = \{n \rightarrow ts_1 ! o \mid (o, n) \leftarrow \text{zip } (\text{ln } cs_i) (\text{ln } \$ \text{decr } cs_i)\} \quad \text{where } i \in [1..n] \\
 ts = \{i \rightarrow (q_i, \text{decr } cs_i, ts_i) \mid i \leftarrow [1..n]\} \\
 \hline
 \Gamma, loop \vdash \text{groupBy}(\lambda x_1 \rightarrow e_1) (\lambda x_2 \rightarrow e_2) e_3 \Rightarrow (q_{\text{out}}, cs, ts) \quad \text{[ac-groupBy]}
 \end{array}$$

# Haskell → SQL with loop-lifting

$$\Gamma, loop \vdash e_1 \Rightarrow (q_1, cs_1, ts_1) \quad q'_v = \text{rownum}_{\text{iter}, pos}^{inner}(q_1)$$

$$a_v = \pi_{\text{iter}:inner, pos:pos', ln\ cs_1}(@_{pos':1}(q'_v)) \quad map_v = \pi_{\text{outer}:iter, inner, pos':pos}(q'_v)$$

“While promising, Ferry’s loop-lifting technique is a monolithic, single-pass translation from a nested language to an SQL:1999-like relational algebra which appears difficult to prove correct [..]”

-- Lindley, Cheney and Wadler 2011

$$cs_i = \text{lookup } cs_1 \ i \quad \text{where } i \in [1..n]$$

$$q_i = \pi_{\text{iter}:g_k, \text{decr } cs_i:cs_i, pos}(q) \quad \text{where } i \in [1..n]$$

$$ts_i = \{n \rightarrow ts_1 ! o \mid (o, n) \leftarrow \text{zip } (ln\ cs_i) (ln\ \$\ \text{decr } cs_i)\} \quad \text{where } i \in [1..n]$$

$$ts = \{i \rightarrow (q_i, \text{decr } cs_i, ts_i) \mid i \leftarrow [1..n]\}$$

$$\Gamma, loop \vdash \text{groupBy } (\lambda x_1 \rightarrow e_1) (\lambda x_2 \rightarrow e_2) e_3 \Rightarrow (q_{out}, cs, ts)$$

[ac-groupBy]

# Goals for a new Compilation Strategy

- Small “intuitive” translation steps
  - Explore new optimisation opportunities
- Provable translation steps

## **Flattening Transformation**

(Guy Blelloch, NESL)...

- ... transforms nested data into flat data
- ... targets parallel execution environment
- ... is suited for declarative languages

## Databases...

- ... process/store flat data
- ... are data-parallel execution environments
- ... query languages are naturally declarative

# Open Questions

Is the flattening transformation suited for targeting databases?

Can we benefit from new database technology?

Can every step be proven correct?  
(Make Philip Wadler happy 😊 😊 😊)



cabal install DSH