

Similarities with DPH

Ferry

DPH

Fields of a tuple live in separate, adjacent columns

pos	val1	val2
1	"A"	10
2	"B"	20
3	"C"	30
⋮	⋮	

$[(a,b)]$ is represented by $([a],[b])$ where array length is synchronized

```
[:"A",      [:10,  
  "B",      20,  
  "C",      30,  
  ...      ...  
:]          :]
```

Similarities with DPH

Ferry

DPH

Foreign keys are descriptors
for nested lists

```
[[1,2],[],[3],[4,5,6]]
```

iter	pos	val
1	1	
1	2	
1	3	
1	4	

box	pos	val
	1	1
	2	2
	1	3
	1	4
	2	5
	3	6

Nested arrays lead to
offset/length descriptors

```
[ : [ :1,2:], [ :: ], [ :3:],  
 [ :4,5,6:] : ]
```

```
[ :0, [ :2, [ :1,  
 2, 0, 2,  
 2, 1, 3,  
 3 3 4,  
 :] :] 5,  
 6  
 :]
```

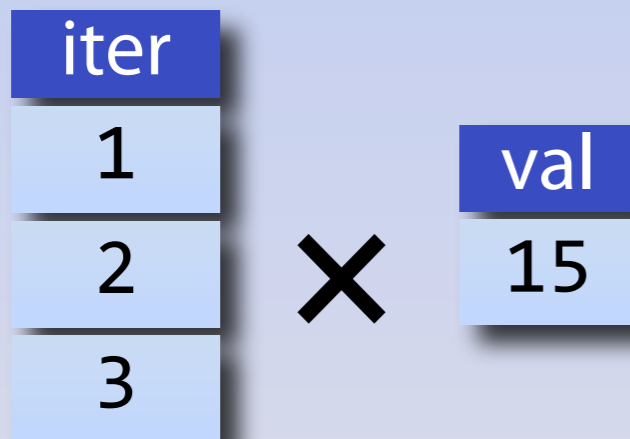
Similarities with DPH

Ferry

DPH

To evaluate iteration in parallel,
expressions are lifted

```
[$qc | x+15 |  
x <- toQ [1,90,4] |]
```



To evaluate iteration in parallel,
expressions are lifted

```
[ : x+15 |  
x <- [ :1,90,4: ] : ]
```

```
replicateP  
(lengthP [ :1,90,4: ] )  
15
```

Similarities with DPH

Ferry

DPH

Compilation targets
machines with
data-parallel primitives

σ π \boxtimes

Compilation targets
vector primitives of
modern CPU architectures

fst^{\wedge}
 snd^{\wedge}
 $*^{\wedge}$
 sumP
 bpermuteP

Similarities with DPH

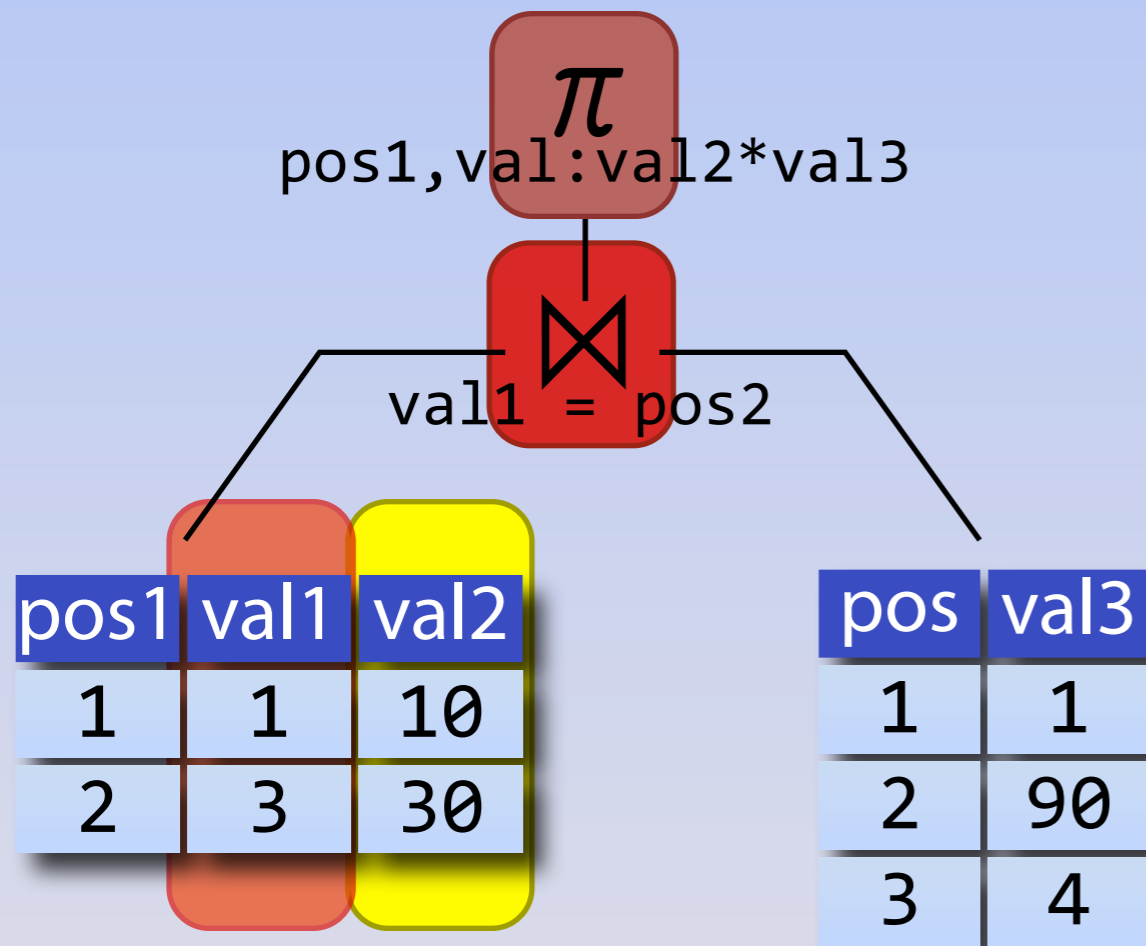
Ferry

DPH

Sample relational plan

```
svMu1 sv v =
  [$qc | f*(v!i) | (i,f) <- sv |]
```

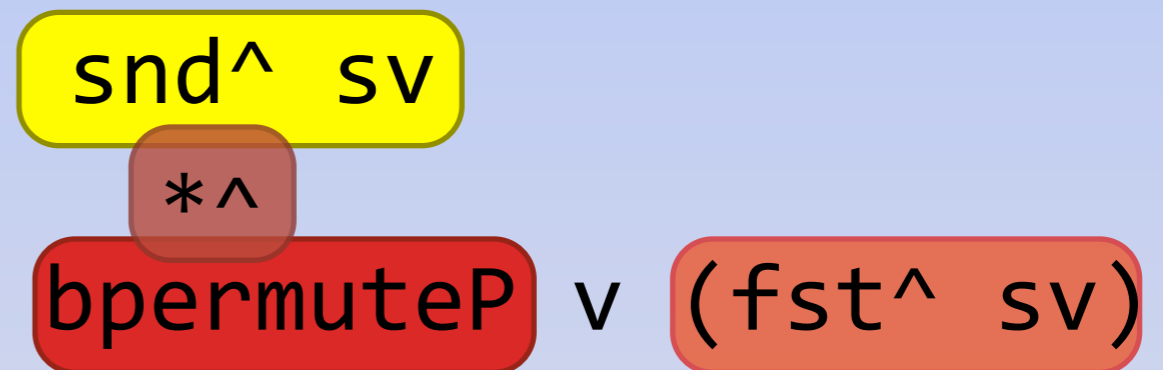
```
svMu1 toQ [(1,10),...]
toQ [1,90,4,...]
```



Sample DPH Core

```
svMu1 sv v =
  [: f*(v!i) | (i,f) <- sv :]
```

```
svMu1 [:(1,10),...:]
[:1,90,4,...:]
```



Embedding internals

Feature

List Comprehension
Type Correctness
Restricting types
Pattern matching
Algebraic code
Boilerplate code

Implementation technique

Quasi Quoting
ADTs and Phantom Types
Type classes
View patterns
Combinators
Template Haskell

List comprehensions

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



Quasi Quoter

```
hasFeatures f = map (\(fac, feat) → feat) $  
                filter (\(fac,feat) → fac ≡ f) (table features)
```

Internal datatype

```
data Exp = VarE String
         | UnitE
         | BoolE Bool
         | CharE Char
         | IntE Int
         | TupleE Exp Exp [Exp]
         | ListE [Exp]
         | FuncE (Exp → Exp)
         | AppE Exp Exp
         | TableE String Type
```

```
data Q a = Q Exp
```

Restricting types

```
class QA a where  
  toQ :: a → Q a  
  fromQ :: Conn → Q a → IO a
```

```
class QA a ⇒ TA a where  
  table :: String → Q [a]  
  table = ...
```

```
instance QA Int where  
  toQ i = IntE i  
  fromQ c (IntE i) = ...
```

```
instance TA Int where  
instance TA Bool where
```

Pattern matching

Pattern matching on QA data:

Tuples: $(\lambda(\text{view} \rightarrow (a, b)) \rightarrow \dots)$

Nested tuples: $(\lambda(\text{view} \rightarrow (a, (\text{view} \rightarrow (b, c))) \rightarrow \dots)$

Records: $(\lambda(\text{view} \rightarrow (\text{UserV} \{ \text{name}, \text{id} \})) \rightarrow \dots)$

Targeting SQL:99 DBMSs

π Projection

σ Selection

\bowtie Join

\times Cross Product

δ Duplicate Elimination

@ Constant Column Attachment

ρ Row Ranking

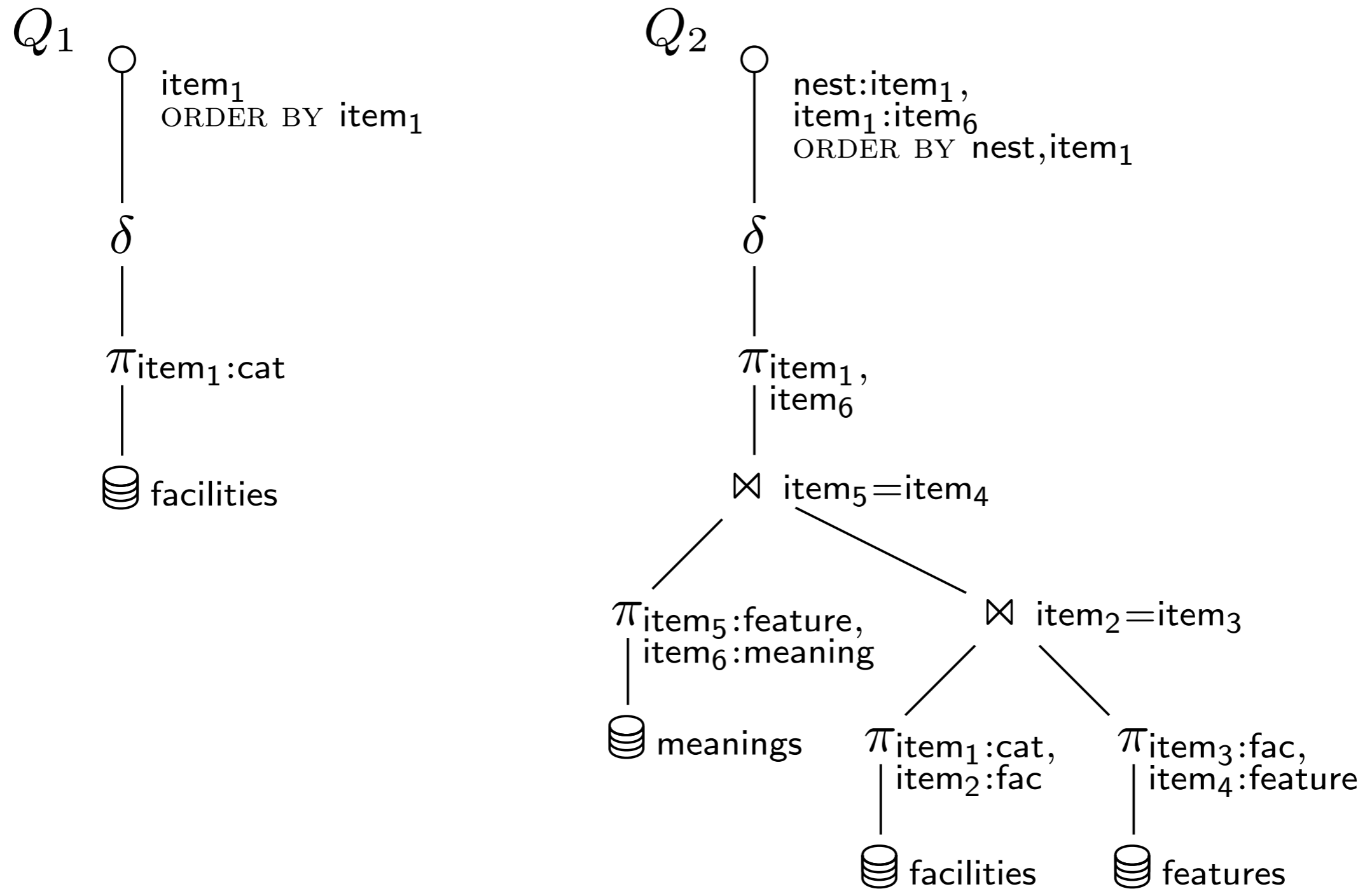
agg Aggregation

 Database Table Access

<i>a</i>	<i>b</i>
<i>c</i> ₁	<i>c</i> ₂

 Literal Table Construction

Algebraic code



SQL code

```
SELECT DISTINCT t0000.cat AS item1  
  FROM facilities AS t0000  
 ORDER BY t0000.cat ASC;
```

```
SELECT DISTINCT t0001.cat AS nest, t0000.meaning AS item1  
  FROM meanings AS t0000,  
       facilities AS t0001,  
       features AS t0002  
 WHERE t0000.feature = t0002.feature AND  
       t0001.fac = t0002.fac  
 ORDER BY t0001.cat ASC, t0000.meaning ASC;
```

Boilerplate code

QA instance for tuples are generated:

```
$(deriveTupleQA 3)
```



Template Haskell

```
instance (QA a, QA b, QA c) => QA (a,b,c) where  
...
```

QA instances automatically derived for:

- Tuples
- Database tables

Example results

Features facilities can have in a category:

```
[("QLA", ["avoids query avalanches",  
          "guarantees translation to SQL",  
          "is statically type-checked"]),  
 ("LIN", ["guarantees translation to SQL",  
          "has compositional syntax and semantics",  
          "is statically type-checked",  
          "supports data nesting"]),  
 ("LIB", ["Respects list order",  
          "Supports data nesting",  
          "Avoids query avalanches",  
          "is statically type-checked",  
          "guarantees translation to SQL",  
          "has compositional syntax and semantics"])]
```