Take Everything From Me, But Leave Me The Comprehension

DBPL — September 2017

Torsten Grust

db.inf.uni-tuebingen.de

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Apologies, I am only a database person

Apologies, I am only a database person

It is in this connection worth noticing that in the Comm.ACM the papers on data bases [...] are of markedly lower linguistic quality than the others.

—Edsger Dijkstra (EWD691)

Apologies, I am only a database person

The point is that the way in which the database management experts tackle the problems seems to be so grossly inadequate. They seem to form an inbred crowd with very little knowledge of computing science in general, who tackle their problems primarily politically instead of scientifically.

—Edsger Dijkstra (EWD577)

Apologies, I am only a database person

Often they seemed to be mentally trapped by the intricacies of early, rather ad hoc solutions to rather accidental problems; as soon as such a technique has received a name, it becomes "a database concept". And a totally inadequate use of language, sharpening their pencils with a blunt axe.

—Edsger Dijkstra (EWD577)

Apologies, I am only a database person

I learned a few things about Databases. I learned —or: had my tentative impression confirmed— that the term "Database Technology", although sometimes used, is immature, for there is hardly any underlying "science" that could justify the use of the term "technology".

—Edsger Dijkstra (EWD577)

Comprehension Syntax

$$[\ h\,x \mid x \leftarrow xs,\ \ p\,x\ ]$$

Comprehension Syntax

generator

$$[\; h\,x \mid x \leftarrow xs,\;\; p\,x \;]$$

head

filter

1. Successively **draw bindings** for *x* from domain *xs*,
2. for those bindings that pass **filter** *p*,
3. evaluate **head** *h*,

Comprehension Syntax

generator

$$[\; h\,x \mid x \leftarrow xs,\;\; p\,x \;]$$

head

filter

1. Successively **draw bindings** for x from domain xs,
2. for those bindings that pass **filter** p,
3. evaluate **head** h,
4. collect results to form a list.

Comprehension Syntax

generator

$$[\ h\,x \mid x \leftarrow xs,\ \ p\,x\ ]^{M}$$

head filter

1. Successively **draw bindings** for *x* from domain *xs*,
2. for those bindings that pass **filter** *p*,
3. evaluate **head** *h*,
4. collect results to form an *M*.

Extension *vs.* Intension

{ }

Extension *vs.* Intension

{ I, III, V, VII, IX }

Extension *vs.* Intension

{ `I`, `III`, `V`, `VII`, `IX` }

$$[\ \mathit{roman}\ x \mid x \leftarrow [1 \ldots 10],\ \mathit{odd}\ x\ ]^{\mathit{set}}$$

In the Beginning …

987

RELATIONAL COMPLETENESS OF DATA BASE SUBLANGUAGES

by

E. F. Codd

IBM Research Laboratory
San Jose, California

Relational Completeness of Data Base Sublanguages
E. F. Codd, IBM Research Report RJ987, 1972

$$(r_1[3],\ r_2[2]):\ P_1 r_1 \wedge P_2 r_2 \wedge (r_1[1] = r_2[1]).$$

In the Beginning …

987

RELATIONAL COMPLETENESS OF DATA BASE SUBLANGUAGES

by

E. F. Codd

IBM Research Laboratory
San Jose, California

Relational Completeness of Data Base Sublanguages
E. F. Codd, IBM Research Report RJ987, 1972

generator

$$(r_1[3],\ r_2[2]):\ P_1 r_1 \wedge P_2 r_2 \wedge (r_1[1] = r_2[1]).$$

head

filter

Today’s XQuery 3.0

sequence. The query returns one valu

```
for $x at $i in $inputvalues
where $i mod 100 = 0
return $x
```

XQuery 3.0: An XML Query Language
D. Chamberlin et al., W3C Recommendation, April 2014

Core of XQuery: versatile *FLWOR* expression

Today's XQuery 3.0

generator

sequence. The query returns one valu

```
for $x at $i in $inputvalues
where $i mod 100 = 0
return $x
```

filter

head

XQuery 3.0: An XML Query Language
D. Chamberlin et al., W3C Recommendation, April 2014

Core of XQuery: versatile *FLWOR* expression

Today's XQuery 3.0

generator

sequence. The query returns one valu

```
for $x at $i in $inputvalues
where $i mod 100 = 0
return $x
```

filter

head

XQuery 3.0: An XML Query Language
D. Chamberlin et al., W3C Recommendation, April 2014

binding and raises a type error for another:

```
some $x in (1, 2, "cat") satisfies $x * 2 = 4
```

This quantified expression may either return true or ra

generator

head

Core of XQuery: versatile *FLWOR* expression

Early XQuery

We differ from other presentations of nested relational algebra in that we make heavy use of list comprehensions, a standard notation in the functional programming community [1]. We find list comprehensions slighly easier to manipulate than the more traditional algebraic operators, but it is not hard to translate comprehensions into these operators (or vice versa).

A Data Model and Algebra for XQuery
M. Fernandez et al., October 2003

We can use comprehensions to express fundamental query operations
uct, nesting, and joins.

We can navigate from a node to all of its children elements with a

```
follow        :: Tag -> Node -> [Node]
follow t x    =  [ y | y <- children x, is t y ]
```

Early XQuery

We differ from other presentations of nested relational algebra in that we make heavy use of list comprehensions, a standard notation in the functional programming community [1]. We find list comprehensions slighly easier to manipulate than the more traditional algebraic operators, but it is not hard to translate comprehensions into these operators (or vice versa).

A Data Model and Algebra for XQuery
M. Fernandez et al., October 2003

We can use comprehensions to express fundamental query operations
uct, nesting, and joins.

We can navigate from a n… s children elements with a

generator

```
follow          :: Tag -> Node -> [Node]
follow t x      =  [ y | y <- children x, is t y ]
```

head

filter

An XQuery Nucleus

```haskell
module Query where
import Prelude hiding (elem,index)

-- Data Model: Constructors ------------------------------------------

text    :: String -> Node
elem    :: Tag -> [Node] -> Node
ref    :: Node -> Node

year0    :: Node
year0    =  elem "@year" [ text "1999" ]

book0    :: Node
book0           =  elem "book" [
           elem "@year"  [ text "1999" ],
           elem "title"  [ text "Data on the Web" ],
           elem "author" [ text "Abiteboul" ],
           elem "author" [ text "Buneman" ],
           elem "author" [ text "Suciu" ]]
```

An XQuery Nucleus

- $\approx$ 430 lines of Haskell (300+ lines of examples)
- Implements a complete XQuery core, including tree construction and traversal
- **List comprehensions** express path navigation, *FLOWR*, grouping/aggregation, quantification

LINQ

```
var q = from product in Products
    where product.Ratings.Any(rating⇒rating == "****")
    select new{ product.Title, product.Keywords };
```

The LINQ comprehension syntax is just syntactic sugar for a set of standard query operators that can be defined in any modern programming language with closures, lambda expressions (written here as `rating⇒rating == "****"`), or

The World According to LINQ
E. Meijer, October 2011

Comprehension syntax deeply embedded into C#, with monad-based semantics organized around `SelectMany` (aka `>>=`, `flatmap`)

LINQ

generator

filter

```
var q = from product in Products
    where product.Ratings.Any(rating⇒rating == "****")
    select new{ product.Title, product.Keywords };
```

The LINQ comprehension syntax is just syntactic sugar for a set of standard query operators that can be defined in any modern programming language with

head mbda expressions (written here as `rating⇒rating == "****"`), or

The World According to LINQ
E. Meijer, October 2011

Comprehension syntax deeply embedded into C#, with monad-based semantics organized around `SelectMany` (aka `>>=`, `flatmap`)

Emma

Listing 6: Page Rank in Emma

```
var iter = 0
while (iter < maxIterations) {
  val messages = for (
    p <- ranks.bag();
    v <- vertices; n <- v.neighbors;
    if p.id == v.vertex) yield {
      RankMessage(n, p.rank / v.neighbors.count())
    }

```

Implicit Parallelism through Deep Language Embedding
A. Alexandrov et al., SIGMOD 2015

Deep embedding of comprehensions in Scala, compiles to Apache Flink / Spark

Emma

Listing 6: Page Rank in Emma

generator

```
var iter = 0
while (iter < maxIterations) {
  val messages = for (
    p <- ranks.bag();
    v <- vertices; n <- v.neighbors;
    if p.id == v.vertex) yield {
      RankMessage(n, p.rank / v.neighbors.count())
    }
```

filter

head

Implicit Parallelism through Deep Language Embedding
A. Alexandrov et al., SIGMOD 2015

Deep embedding of comprehensions in Scala, compiles to Apache Flink / Spark

Pig Latin

```
X = FOREACH A GENERATE a1+a2 AS f1:int;

DESCRIBE X;
x: {f1: int}

DUMP X;
(3)
(6)
(11)
(7)
(9)
(12)

Y = FILTER X BY f1 > 10;

DUMP Y;
(11)
(12)
```

Compiles to sequences of *Map/Reduce* jobs

Pig Latin

generator

head

```
X = FOREACH A GENERATE a1+a2 AS f1:int;

DESCRIBE X;
x: {f1: int}

DUMP X;
(3)
(6)
(11)
(7)
(9)
(12)

Y = FILTER X BY f1 > 10;

DUMP Y;
(11)
(12)
```

filter

Compiles to sequences of *Map/Reduce* jobs

Pig Latin

generator

head

```
X = FOREACH A GENERATE a1+a2 AS f1:int;

DESCRIBE X;
x: {f1: int}
```

Note: FOREACH statements can be nested to two levels only. FOREACH statements that are nested to three or more levels will result in a grammar error.

```
(9)
(12)

Y = FILTER X BY f1 > 10;

DUMP Y;
(11)
(12)
```

filter

Compiles to sequences of *Map/Reduce* jobs

Pig Latin

generator

head

```
X = FOREACH A GENERATE a1+a2 AS f1:int;

DESCRIBE X;
x: {f1: int}

(9)
(12)

Y = FILTER X BY f1 > 10;

DUMP Y;
(11)
(12)
```

filter

Note: FOREACH statements can be nested to two levels only. FOREACH statements that are nested to three or more levels will result in a grammar error.

Told you so.

Compiles to sequences of *Map/Redu*

SQL

```
SELECT o_orderpriority, COUNT(*) AS order_count
FROM   orders
WHERE  o_orderdate > @DATE@
AND    o_orderdate < @DATE@ + interval '3 months'
AND    EXISTS (SELECT *
               FROM   lineitem
               WHERE  l_orderkey = o_orderkey
               AND    l_commitdate < l_receiptdate)
GROUP BY o_orderpriority
ORDER BY o_orderpriority;
```

Query *Q4* of the TPC-H OLAP benchmark

SQL

head

generator

filter

```
SELECT o_orderpriority, COUNT(*) AS order_count
FROM   orders
WHERE  o_orderdate > @DATE@
AND    o_orderdate < @DATE@ + interval '3 months'
AND    EXISTS (SELECT *
               FROM   lineitem
               WHERE  l_orderkey = o_orderkey
               AND    l_commitdate < l_receiptdate)
GROUP BY o_orderpriority
ORDER BY o_orderpriority;
```

Query *Q4* of the TPC-H OLAP benchmark

SQL

head

generator

filter

```
SELECT o_orderpriority, COUNT(*) AS order_count
FROM   orders
WHERE  o_orderdate > @DATE@
AND    o_orderdate < @DATE@ + interval '3 months'
AND    EXISTS (SELECT *
               FROM   lineitem
               WHERE  l_orderkey = o_orderkey
               AND    l_commitdate < l_receiptdate)
GROUP BY o_orderpriority
ORDER BY o_orderpriority;
```

head

generator

filter

Query *Q4* of the TPC-H OLAP benchmark

SQL

```
SELECT o_orderpriority, COUNT(*) AS order_count
FROM   orders
WHERE  o_orderdate > @DATE@
AND    o_orderdate < @DATE@ + interval '3 months'
AND    EXISTS (SELECT *
               FROM   lineitem
               WHERE  l_orderkey = o_orderkey
               AND    l_commitdate < l_receiptdate)
GROUP BY o_orderpriority
ORDER BY o_orderpriority;
```

head
generator
head
generator
head
filter
head
generator
head
generator
filter
filter
head
generator
filter

Query *Q4* of the TPC-H OLAP benchmark

SQL

One Way to Teach SQL

One Way to Teach SQL

```
SELECT A, B
FROM   S
```

One Way to Teach SQL

```
SELECT A, B
FROM   S
```

```
c ← ∅;
foreach x ∈ S do
    c ← c ⊎ {(x.A,x.B)};
return c;
```

One Way to Teach SQL

```
SELECT A, B
FROM   S
```

```
SELECT MAX(A)
FROM   S
```

$c \leftarrow \emptyset;$
$\text{foreach } x \in S \text{ do}$
$\quad c \leftarrow c \uplus \{(x.A, x.B)\};$
$\text{return } c;$

One Way to Teach SQL

```
SELECT A, B
FROM   S
```

```
c ← ∅ ;
foreach x ∈ S do
    c ← c ⊎ {(x.A,x.B)};
return c;
```

```
SELECT MAX(A)
FROM   S
```

```
c ← −∞ ;
foreach x ∈ S do
    c ← max₂(c,x.A);
return c;
```

One Way to Teach SQL

```
SELECT A, B
FROM   S
```

```
c ← ∅ ;
foreach x ∈ S do
    c ← c ⊎ {(x.A,x.B)};
return c;
```

```
SELECT MAX(A)
FROM   S
```

```
c ← −∞ ;
foreach x ∈ S do
    c ← max2(c,x.A);
return c;
```

```
0 < ALL(SELECT A
        FROM   S)
```

One Way to Teach SQL

```
SELECT A, B
FROM   S
```

```
c ← ∅ ;
foreach x ∈ S do
    c ← c ⊎ {(x.A,x.B)};
return c;
```

```
SELECT MAX(A)
FROM   S
```

```
c ← −∞ ;
foreach x ∈ S do
    c ← max₂(c,x.A);
return c;
```

```
0 < ALL(SELECT A
        FROM   S)
```

```
c ← true ;
foreach x ∈ S do
    c ← c ∧ (0 < x.A);
return c;
```

One Way to Teach SQL

```
SELECT A, B
FROM   S
```

$c \leftarrow \emptyset$;
foreach $x \in S$ do
    $c \leftarrow c \uplus \{(x.A, x.B)\}$;
return c;

```
SELECT MAX(A)
FROM   S
```

$c \leftarrow -\infty$;
foreach $x \in S$ do
    $c \leftarrow \max_2(c, x.A)$;
return c;

```
0 < ALL(SELECT A
        FROM   S)
```

$c \leftarrow$ true;
foreach $x \in S$ do
    $c \leftarrow c \land (0 < x.A)$;
return c;

One Program Form for SQL

One Program Form for SQL

$$\texttt{fold}(z,f,xs) \;\equiv\; \begin{array}{l} \texttt{c} \leftarrow z; \\ \texttt{foreach x} \in xs \texttt{ do} \\ \quad\quad \texttt{c} \leftarrow f(\texttt{c},\texttt{x}); \\ \texttt{return c;} \end{array}$$

One Program Form for SQL

$$\texttt{fold}(z, f, xs) \equiv \begin{array}{l}\texttt{c} \leftarrow z;\\ \texttt{foreach x} \in xs \texttt{ do}\\ \quad \texttt{c} \leftarrow f(\texttt{c},\texttt{x});\\ \texttt{return c;}\end{array}$$

M	**carrier**	*lift*^{M}	z^M	$\oplus^M$
bag	*bag t*	$\{\cdot\}$	$\emptyset$	$\uplus$
set	*set t*	$\{\cdot\}$	$\emptyset$	$\cup$
list	*list t*	$[\cdot]$	$[\,]$	$+\!\!+$
all	*bool*	*id*	`true`	$\wedge$
some	*bool*	*id*	`false`	$\vee$
sum	*num*	*id*	`0`	$+$
max	*t* (ordered)	*id*	$-\infty$	$\mathtt{max}_2$
min	*t* (ordered)	*id*	∞	$\mathtt{min}_2$

One Program Form for SQL

One Program Form for SQL

```
SELECT A
FROM   S
WHERE  A > B
```

```
fold(∅,⊕,S) with
  ⊕(c,x) = c ⊎ (if (x.A > x.B) {x.A}
                else ∅)
```

One Program Form for SQL

```
SELECT A
FROM   S
WHERE  A > B
```

```
fold(∅,⊕,S) with
  ⊕(c,x) = c ⊎ (if (x.A > x.B) {x.A}
                else ∅)
```

```
SELECT x.A,y.B
FROM   R x,S y
```

```
fold(∅,⊕,R) with
  ⊕(c,x) = c ⊎ fold(∅,⊗,S) with
     ⊗(d,y) = d ⊎ {(x.A,y.B)}
```

`fold(,,)` Gets Ugly Quickly

fold(,,) Gets Ugly Quickly

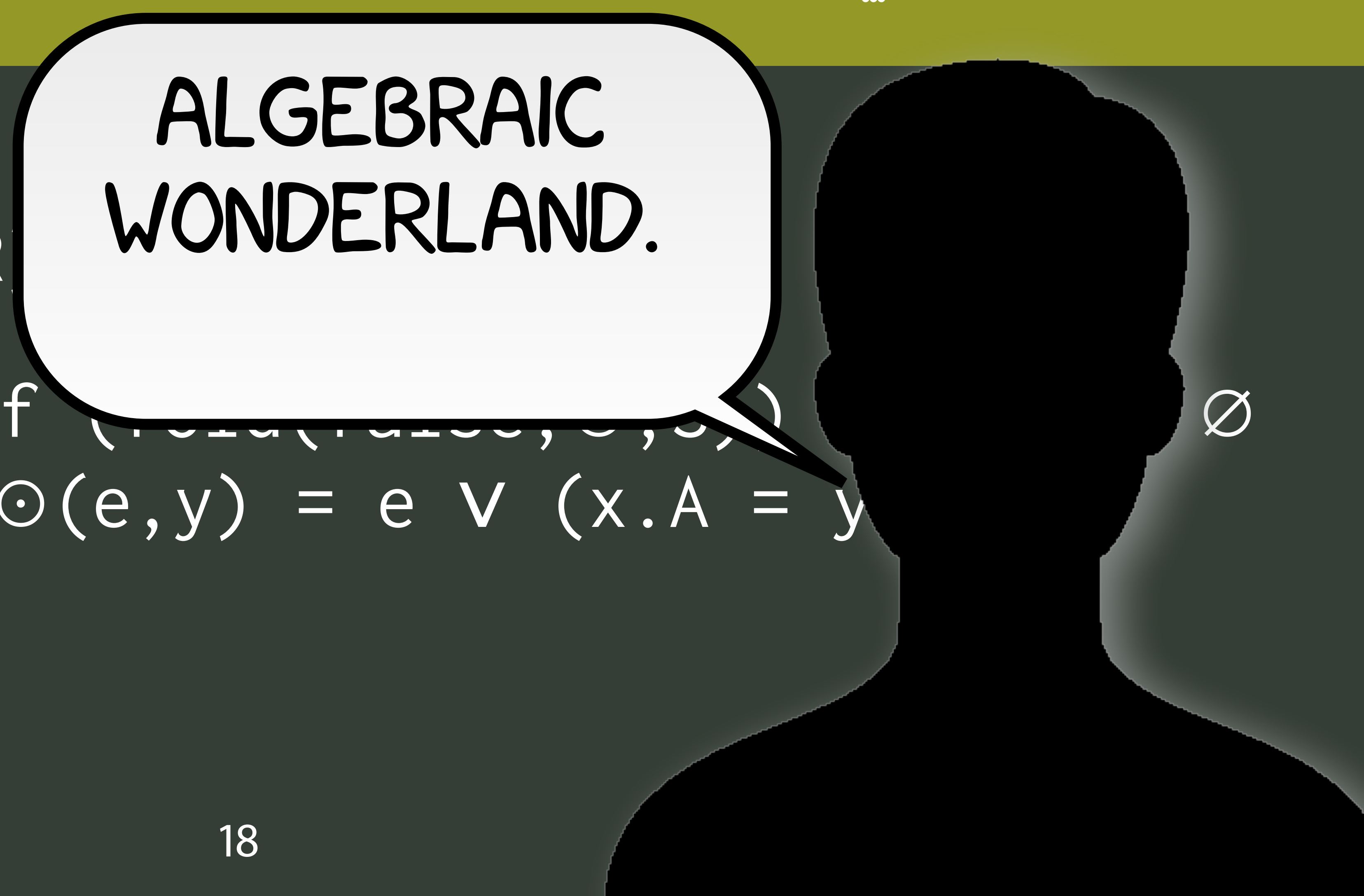
```
SELECT COUNT(*)
FROM   R x
WHERE  EXISTS (SELECT y
               FROM   S y
               WHERE  x.A = y.B)
```

```
fold(0,⊕,fold(∅,⊗,R)) with
with ⊕(c,_) = c + 1
     ⊗(d,x) = d ⊎ if (fold(false,⊙,S)) {x} else ∅
              with ⊙(e,y) = e ∨ (x.A = y.B)
```

fold(,,) Gets Ugly Quickly

```
SELECT COUNT(*)
FROM   R x
WHERE  EXISTS (SELECT y
               FROM   S y
               WHERE  x.A = y.B)
```

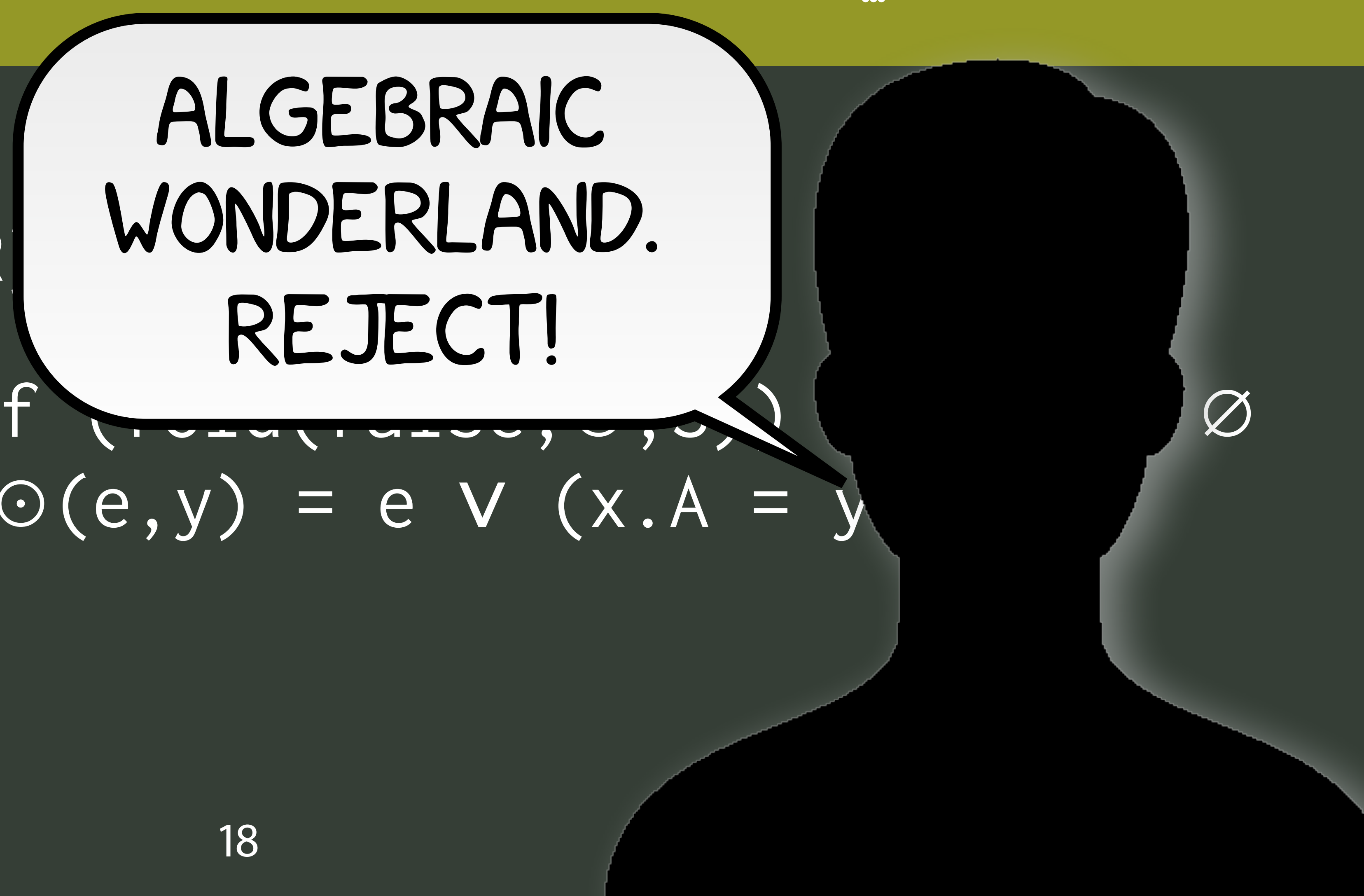
```
fold(0,⊕,fold(∅,⊗,R
with ⊕(c,_) = c + 1
     ⊗(d,x) = d ⊎ if                                  ∅
                    with ⊙(e,y) = e ∨ (x.A = y
```



fold(,,) Gets Ugly Quickly

```
SELECT COUNT(*)
FROM   R x
WHERE  EXISTS (SELECT y
               FROM   S y
               WHERE  x.A = y.B)
```

```
fold(0,⊕,fold(∅,⊗,R
with ⊕(c,_) = c + 1
     ⊗(d,x) = d ⊎ if (fold(false,⊙,S)                    ∅
                  with ⊙(e,y) = e ∨ (x.A = y
```



Comprehension Semantics

Comprehension Semantics

$[\ e \mid\ ]^{M}$

$[\ e \mid v_1 \leftarrow e_1,\ q\ ]^{M}$

$[\ e \mid p,\ q\ ]^{M}$

Comprehension Semantics

$[\; e \mid \;]^M \quad\equiv\quad lift^M(e)$

$[\; e \mid v_1 \leftarrow e_1,\; q \;]^M \quad\equiv$

$[\; e \mid p,\; q \;]^M \quad\equiv$

Comprehension Semantics

$$[\ e \mid\ ]^M \equiv \mathit{lift}^M(e)$$

$$[\ e \mid v_1 \leftarrow e_1,\ q\ ]^M \equiv \texttt{fold}(z^M, \otimes, e_1)\ \texttt{with}$$
$$\otimes(\texttt{c}, v_1) = \texttt{c}\ \oplus^M [\ e \mid q\ ]^M$$

$$[\ e \mid p,\ q\ ]^M \equiv \texttt{if}\ (p)\ [\ e \mid q\ ]^M\ \texttt{else}\ z^M$$

Comprehensible SQL

Comprehensible SQL

```
SELECT COUNT(*)
FROM   R x
WHERE  EXISTS (SELECT y
               FROM   S y
               WHERE  x.A = y.B)
```

Comprehensible SQL

```
SELECT COUNT(*)
FROM   R x
WHERE  EXISTS (SELECT y
               FROM   S y
               WHERE  x.A = y.B)
```

$$[\ y \mid y \leftarrow S,\ x.A = y.B\ ]^{bag}$$

Comprehensible SQL

```
SELECT COUNT(*)
FROM   R x
WHERE  EXISTS (SELECT y
               FROM   S y
               WHERE  x.A = y.B)
```

$$[\ \mathtt{true}\ |\ \_ \leftarrow [\ \mathtt{y}\ |\ \mathtt{y} \leftarrow \mathtt{S},\ \mathtt{x.A} = \mathtt{y.B}\ ]^{bag}\ ]^{some}$$

Comprehensible SQL

```
SELECT COUNT(*)
FROM   R x
WHERE  EXISTS (SELECT y
               FROM   S y
               WHERE  x.A = y.B)
```

$$[\ 1 \mid x \leftarrow R,\ [\ \mathtt{true} \mid \_ \leftarrow [\ \mathtt{y} \mid \mathtt{y} \leftarrow \mathtt{S},\ \mathtt{x.A} = \mathtt{y.B}\ ]^{bag}\ ]^{some}\ ]^{sum}$$

Comprehensible SQL

```
SELECT  COUNT(*)
FROM    R x
WHERE   EXISTS (SELECT y
                FROM   S y
                WHERE  x.A = y.B)
```

$$[\ 1 \mid x \leftarrow R,\ [\ \mathrm{true} \mid \_ \leftarrow [\ y \mid y \leftarrow S,\ x.A = y.B\ ]^{bag}\ ]^{some}\ ]^{sum}$$

$$[\ 1 \mid x \leftarrow R,\ [\ x.A = y.B \mid y \leftarrow S\ ]^{some}\ ]^{sum}$$

Comprehension Unnesting

$$[\ e \mid qs_1,\ v \leftarrow [\ ]^N,\ qs_3\ ]^M$$

$$[\ e \mid qs_1,\ v \leftarrow [\ e_2\ ]^N,\ qs_3\ ]^M$$

$$[\ e \mid qs_1,\ v \leftarrow [\ e_2 \mid qs_2\ ]^N,\ qs_3\ ]^M$$

$$[\ e \mid qs_1,\ [\ e_2 \mid qs_2\ ]^{some},\ qs_3\ ]^M$$

Comprehension Unnesting

$$[\ e \mid qs_1,\ v \leftarrow [\,]^N,\ qs_3\ ]^M$$
$$[\,]^M$$

$$[\ e \mid qs_1,\ v \leftarrow [\ e_2\ ]^N,\ qs_3\ ]^M$$
$$[\ e[e_2/v] \mid qs_1,\ qs_3[e_2/v]\ ]^M$$

$$[\ e \mid qs_1,\ v \leftarrow [\ e_2 \mid qs_2\ ]^N,\ qs_3\ ]^M$$
$$[\ e[e_2/v] \mid qs_1,\ qs_2,\ qs_3[e_2/v]\ ]^M$$

$$[\ e \mid qs_1,\ [\ e_2 \mid qs_2\ ]^{some},\ qs_3\ ]^M$$
$$[\ e \mid qs_1,\ qs_2,\ e_2,\ qs_3\ ]^M$$

($\oplus^M$ idempotent)

When Syntax Distracts

On Optimizing an SQL-like Nested Query

WON KIM
IBM Research

SQL is a high-level nonprocedural data language which has received wide recognition in relational databases. One of the most interesting features of SQL is the nesting of query blocks to an arbitrary depth. An SQL-like query nested to an arbitrary depth is shown to be composed of five basic types of nesting. Four of them have not been well understood and more work needs to be done to improve their execution efficiency. Algorithms are developed that transform queries involving these basic

On Optimizing an SQL-like Nested Query
W. Kim, ACM TODS, 1982

When Syntax Distracts

On Optimizing an SQL-like Nested Query

WON KIM
IBM Research

An SQL-like query nested to an arbitrary depth is shown to be composed of five basic types of nesting. Four of them have not been well understood

On Optimizing an SQL-like Nested Query
W. Kim, ACM TODS, 1982

When Syntax Distracts

On Optimizing an SQL-like Nested Query

WON KIM
IBM Research

An SQL-like query nested to an arbitrary depth is shown to be composed of five basic types of nesting. Four of them have not been well understood

On Optimizing an SQL-like Nested Query
W. Kim, ACM TODS, 1982

Implemented in most RDBMSs to this day

When Syntax Distracts

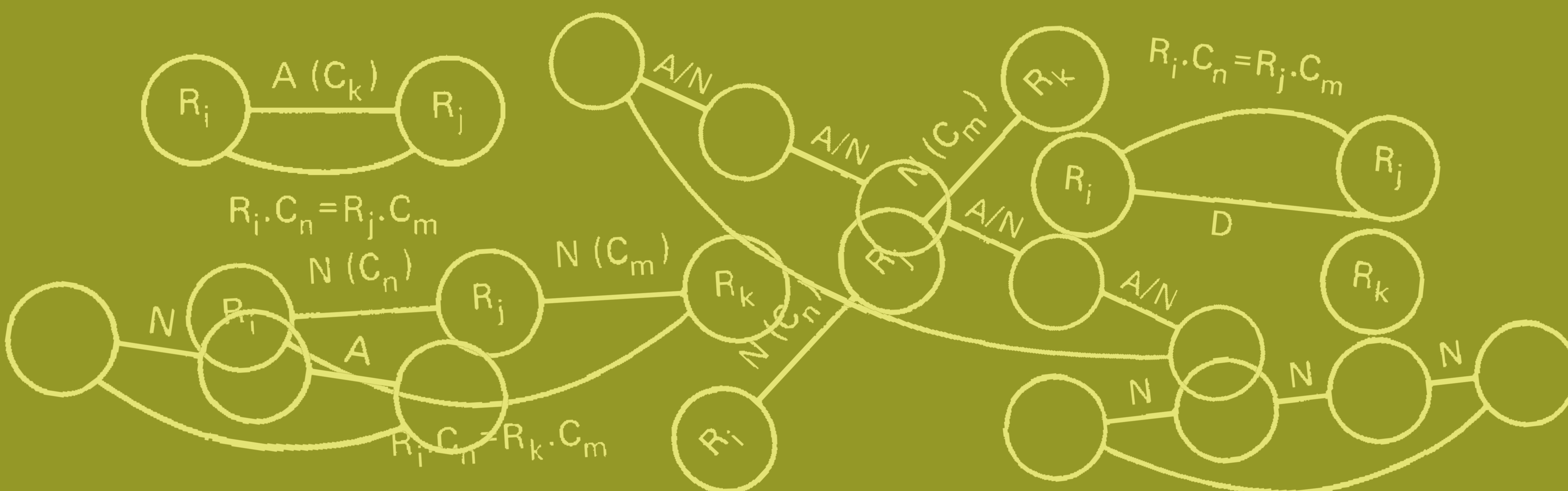
- **Syntactic** classification of nested SQL queries into types *N*, *Nx*, *D*, *J*, *A*, *JA*, *JA*(*NA*), *JA*(*AA*), *JA*(*AN*), …

- Classes are associated with their particular SQL–level unnesting rewrites.

When Syntax Distracts

- **Syntactic** classification of nested SQL queries into types *N*, *Nx*, *D*, *J*, *A*, *JA*, *JA*(*NA*), *JA*(*AA*), *JA*(*AN*), …

- Classes are associated with their particular SQL–level unnesting rewrites.



When Syntax Distracts

```
SELECT DISTINCT f(x)
FROM   R AS x
WHERE  p(x) IN (SELECT g(y)
                FROM   S AS y
                WHERE  q(x,y))
```

When Syntax Distracts

```
SELECT DISTINCT f(x)
FROM   R AS x
WHERE  p(x) IN (SELECT g(y)
                FROM   S AS y
                WHERE  q(x,y))
```

$$[\ f(x)\ |\ \ x \leftarrow R,$$
$$[\ p(x) = v\ |\ v \leftarrow [\ g(y)\ |\ y \leftarrow S,\ q(x,y)\ ]^{bag}\ ]^{some}\ ]^{set}$$

When Syntax Distracts

```
SELECT DISTINCT f(x)
FROM   R AS x
WHERE  p(x) IN (SELECT g(y)
                FROM   S AS y
                WHERE  q(x,y))
```

$$[\ f(x) \mid x \leftarrow R,\ [\ p(x) = g(y) \mid y \leftarrow S,\ q(x,y)\ ]^{some}\ ]^{set}$$

When Syntax Distracts

```
SELECT DISTINCT f(x)
FROM   R AS x
WHERE  p(x) IN (SELECT g(y)
                FROM   S AS y
                WHERE  q(x,y))
```

$$[\ f(x) \mid x \leftarrow R,\ y \leftarrow S,\ q(x,y),\ p(x) = g(y)\ ]^{set}$$

```
SELECT DISTINCT f(x)
FROM   R AS x, S AS y
WHERE  q(x,y)
AND    p(x) = g(y)
```

A Zoo of Query Representations

Groupwise Processing of Relational Queries

Damianos Chatziantoniou* Kenneth A. Ross*
Department of Computer Science, Columbia University
damianos,kar@cs.columbia.edu

Groupwise Processing of Relational Queries
D. Chatziantoniou, K.A. Ross, VLDB 1997

A Zoo of Query Representations

Groupwise Processing of Relational Queries

Damianos Chatziantoniou* Kenneth A. Ross*
Department of Computer Science, Columbia University
damianos,kar@cs.columbia.edu

Groupwise Processing of Relational Queries
D. Chatziantoniou, K.A. Ross, VLDB 1997

```
SELECT   f(x), agg(g(x))
FROM     R AS x
GROUP BY f(x)
```

A Zoo of Query Representations

Groupwise Processing of Relational Queries

Damianos Chatziantoniou*   Kenneth A. Ross*
Department of Computer Science, Columbia University
damianos,kar@cs.columbia.edu

Groupwise Processing of Relational Queries
D. Chatziantoniou, K.A. Ross, VLDB 1997

$$[\ \langle f(x), [\ g(y) \mid y \leftarrow R, f(y) = f(x)\ ]^{agg} \rangle \mid x \leftarrow R\ ]^{set}$$

A Zoo of Query Representations

Groupwise Processing of Relational Queries

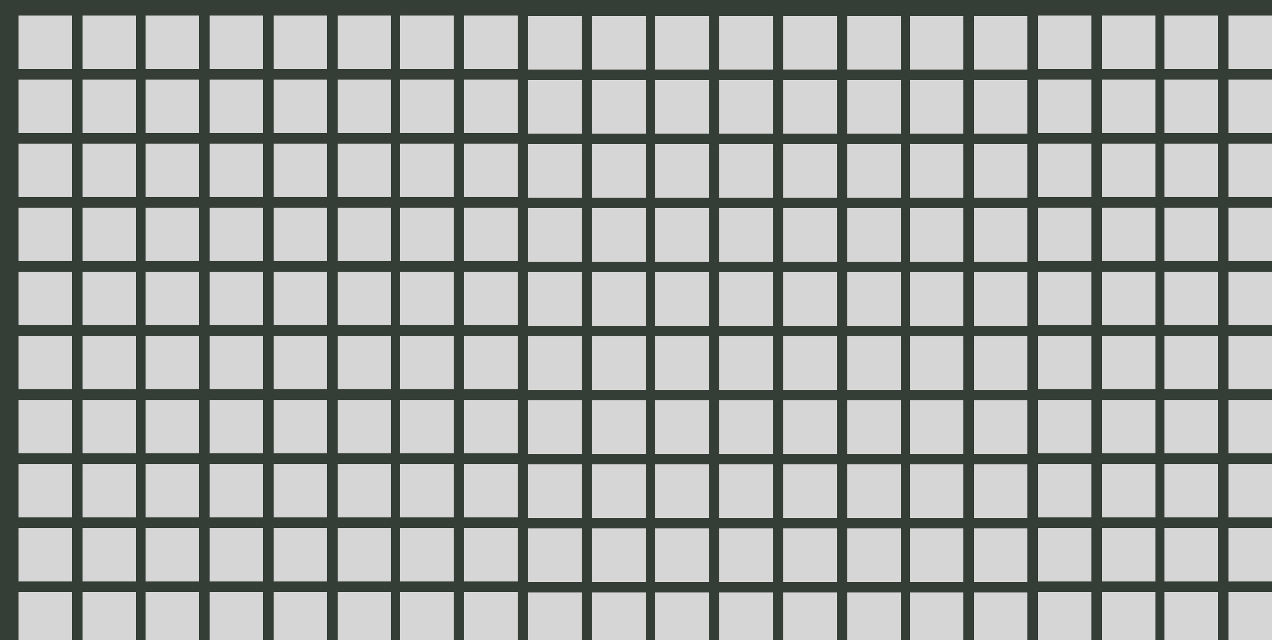
Damianos Chatziantoniou* Kenneth A. Ross*
Department of Computer Science, Columbia University
damianos,kar@cs.columbia.edu

Groupwise Processing of Relational Queries
D. Chatziantoniou, K.A. Ross, VLDB 1997

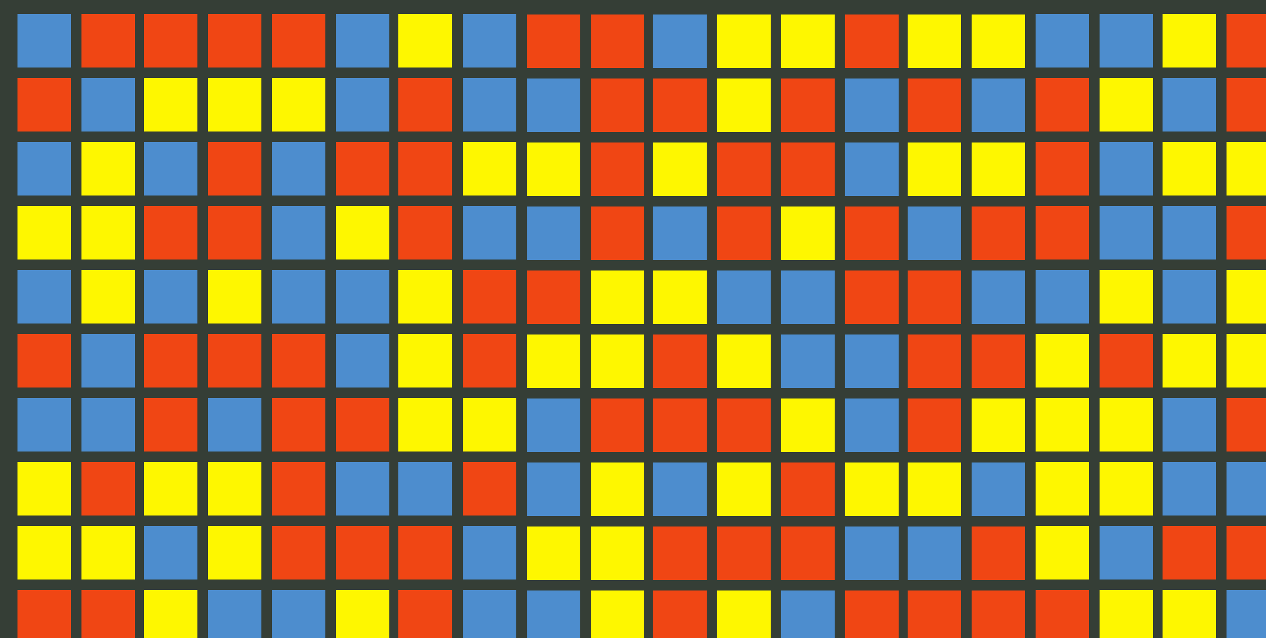
$$Q\ f\ g\ agg\ \mathrm{R}\ \equiv$$

$$[\ \langle f(\mathrm{x}), [\ g(\mathrm{y}) \mid \mathrm{y} \leftarrow \mathrm{R}, f(\mathrm{y}) = f(\mathrm{x})\ ]^{agg} \rangle \mid \mathrm{x} \leftarrow \mathrm{R}\ ]^{set}$$

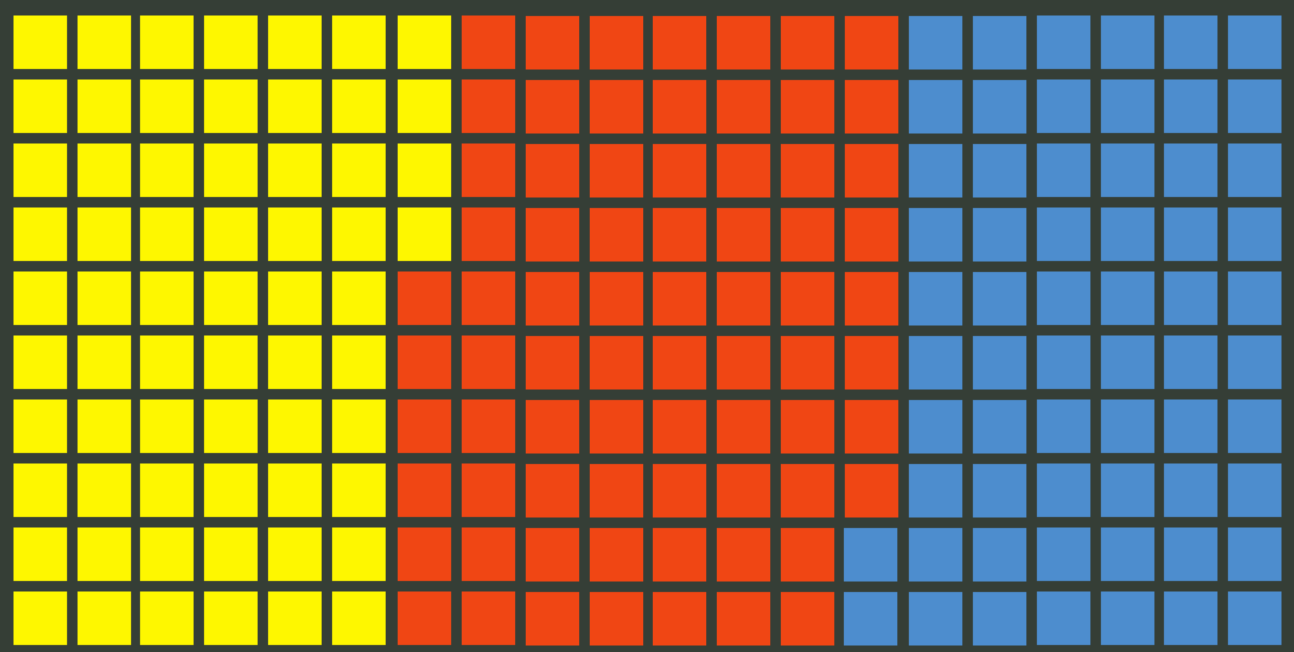
A Zoo of
Query Representations



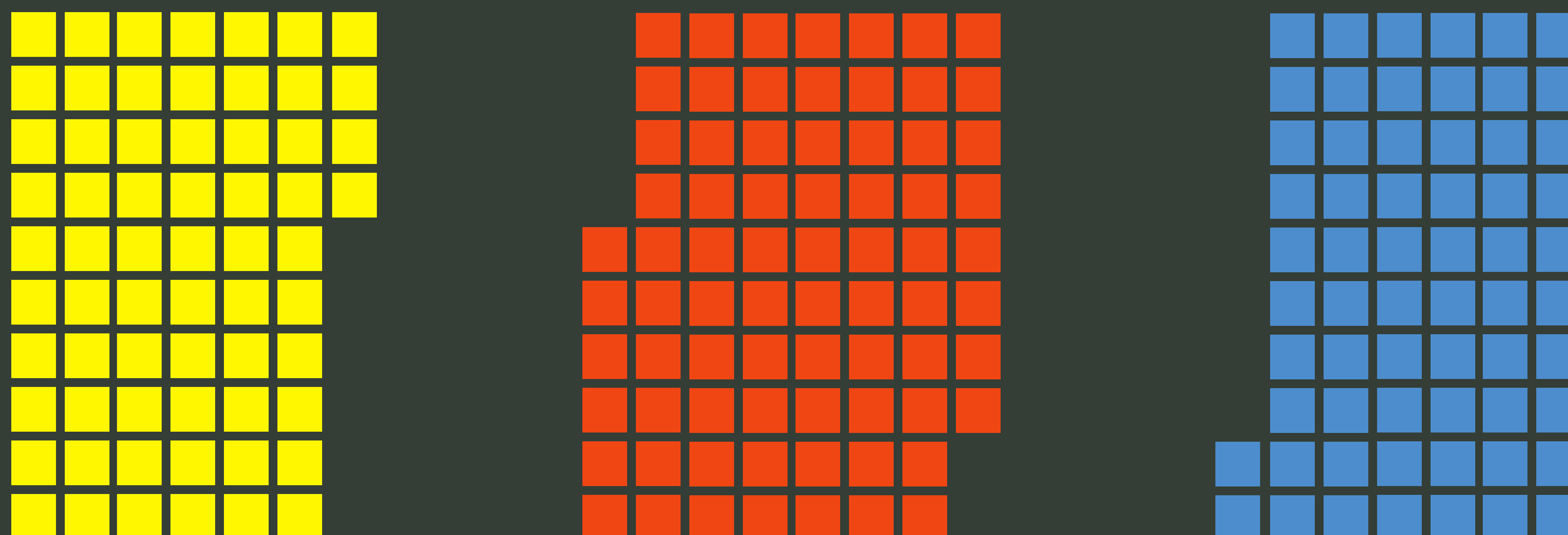
A Zoo of Query Representations



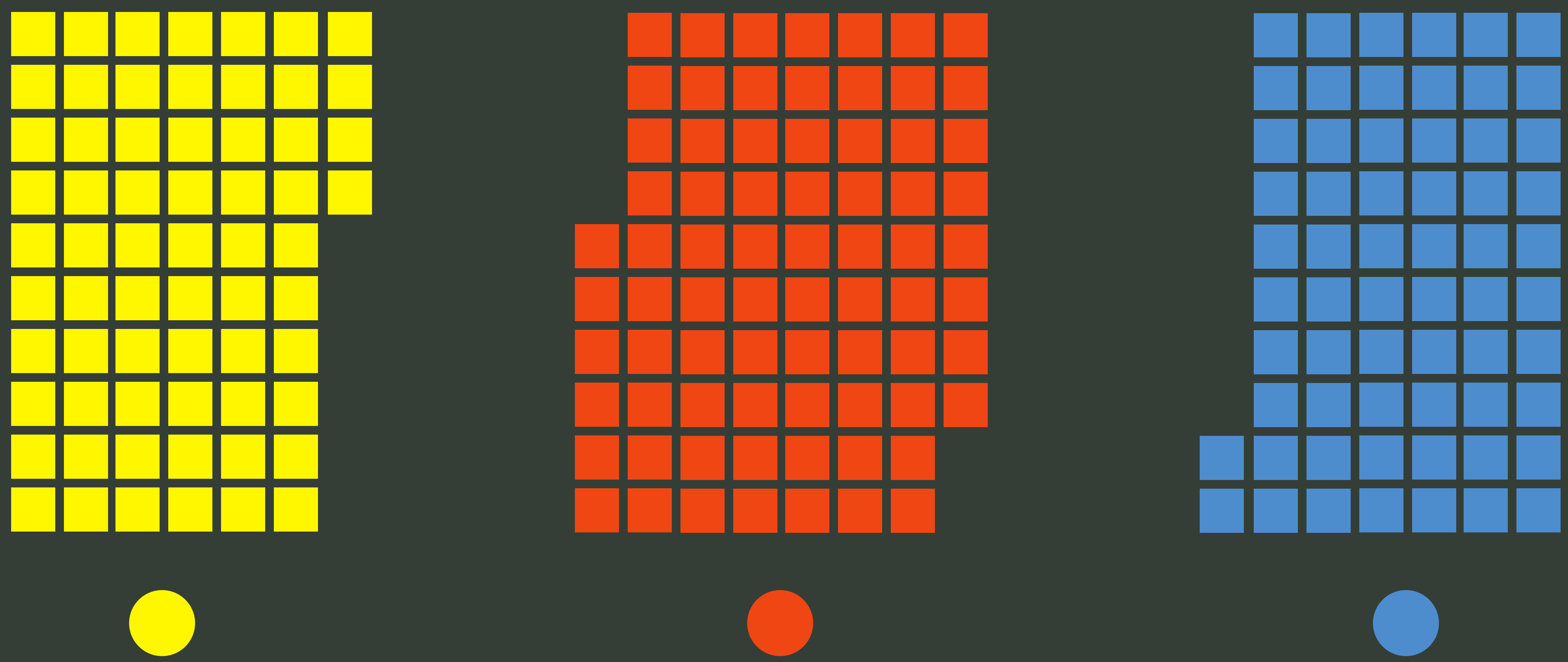
A Zoo of Query Representations



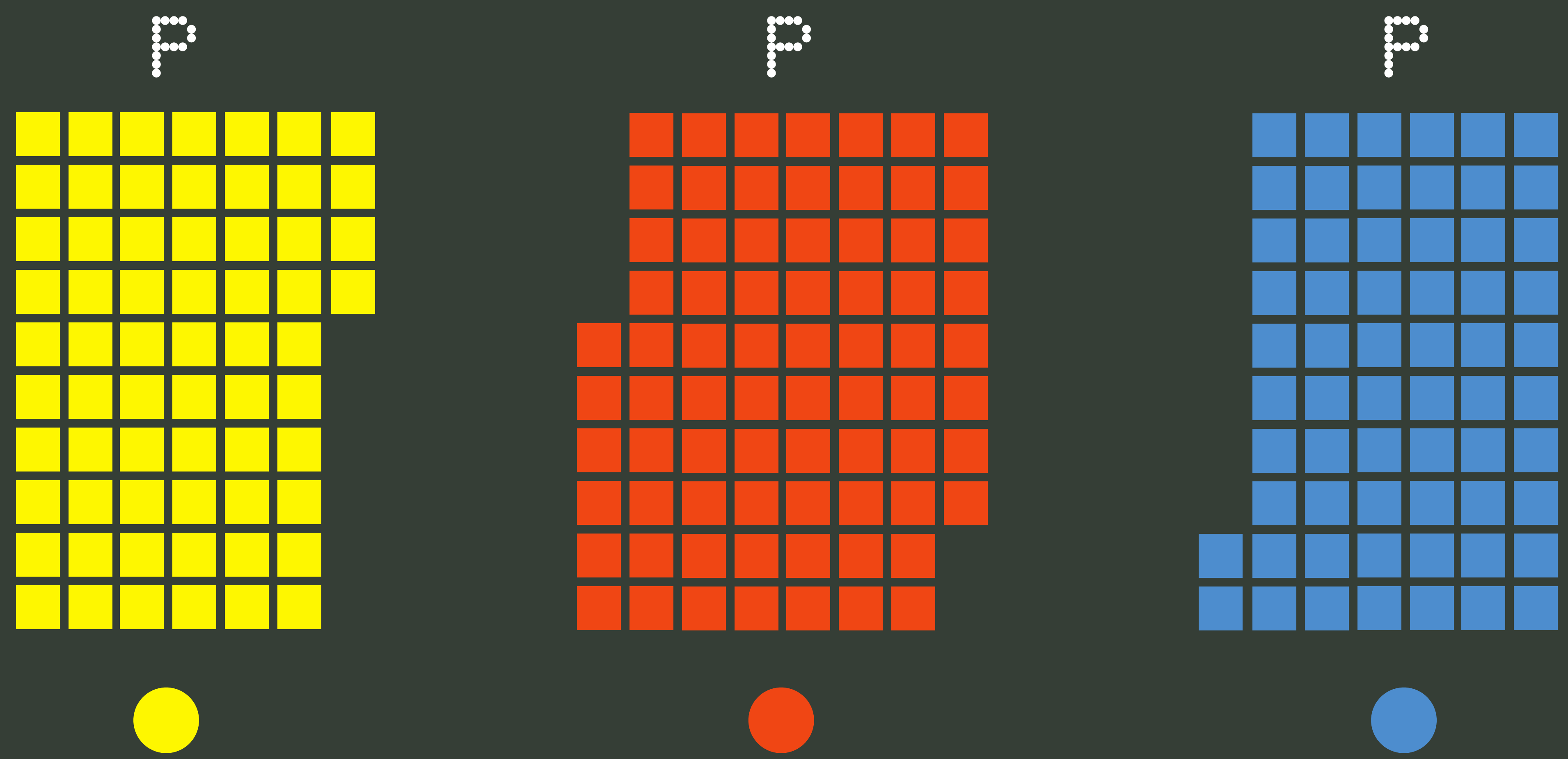
A Zoo of Query Representations



A Zoo of Query Representations

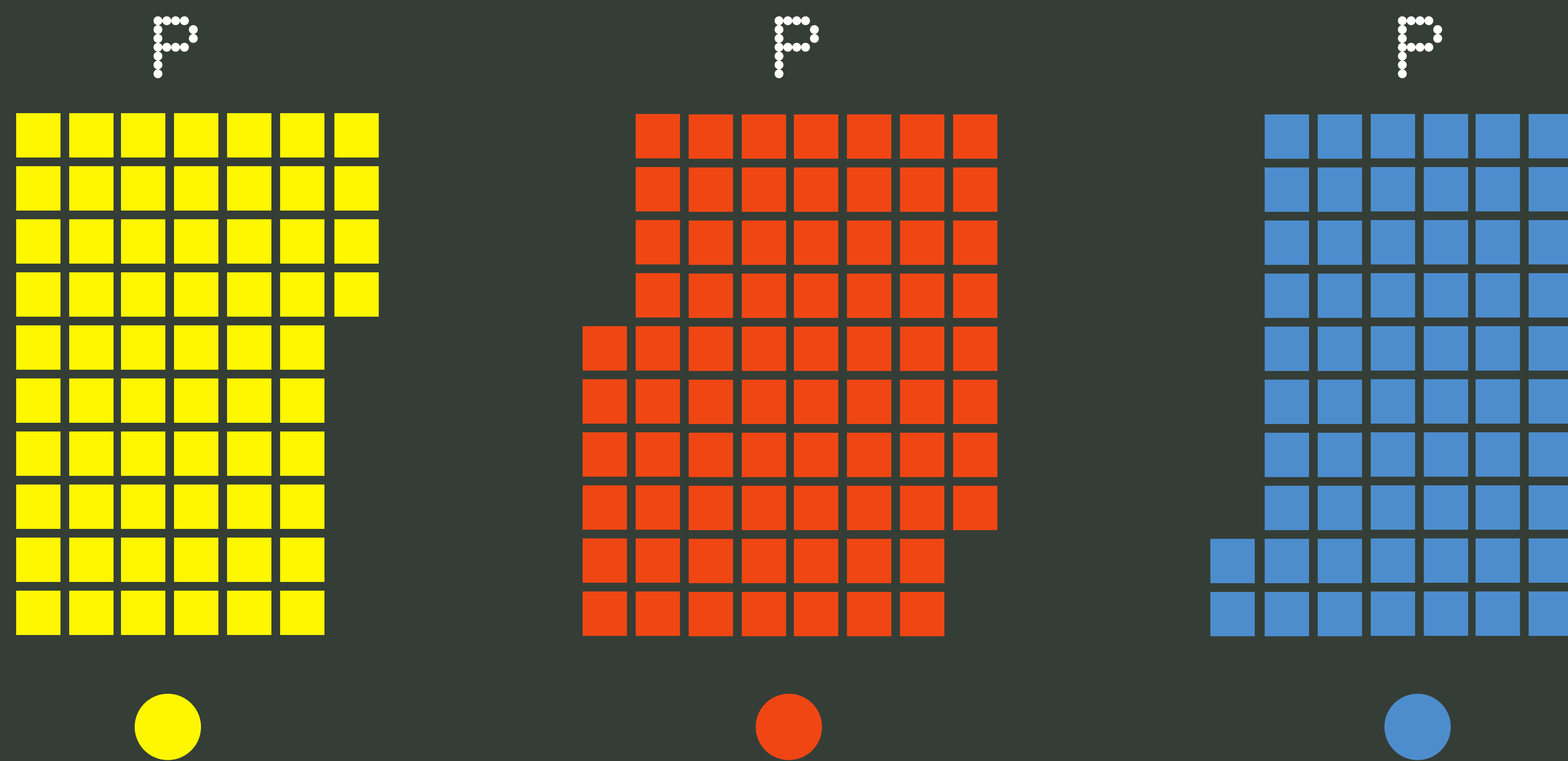


A Zoo of Query Representations




```
SELECT agg(g(x))
FROM   P AS x
```

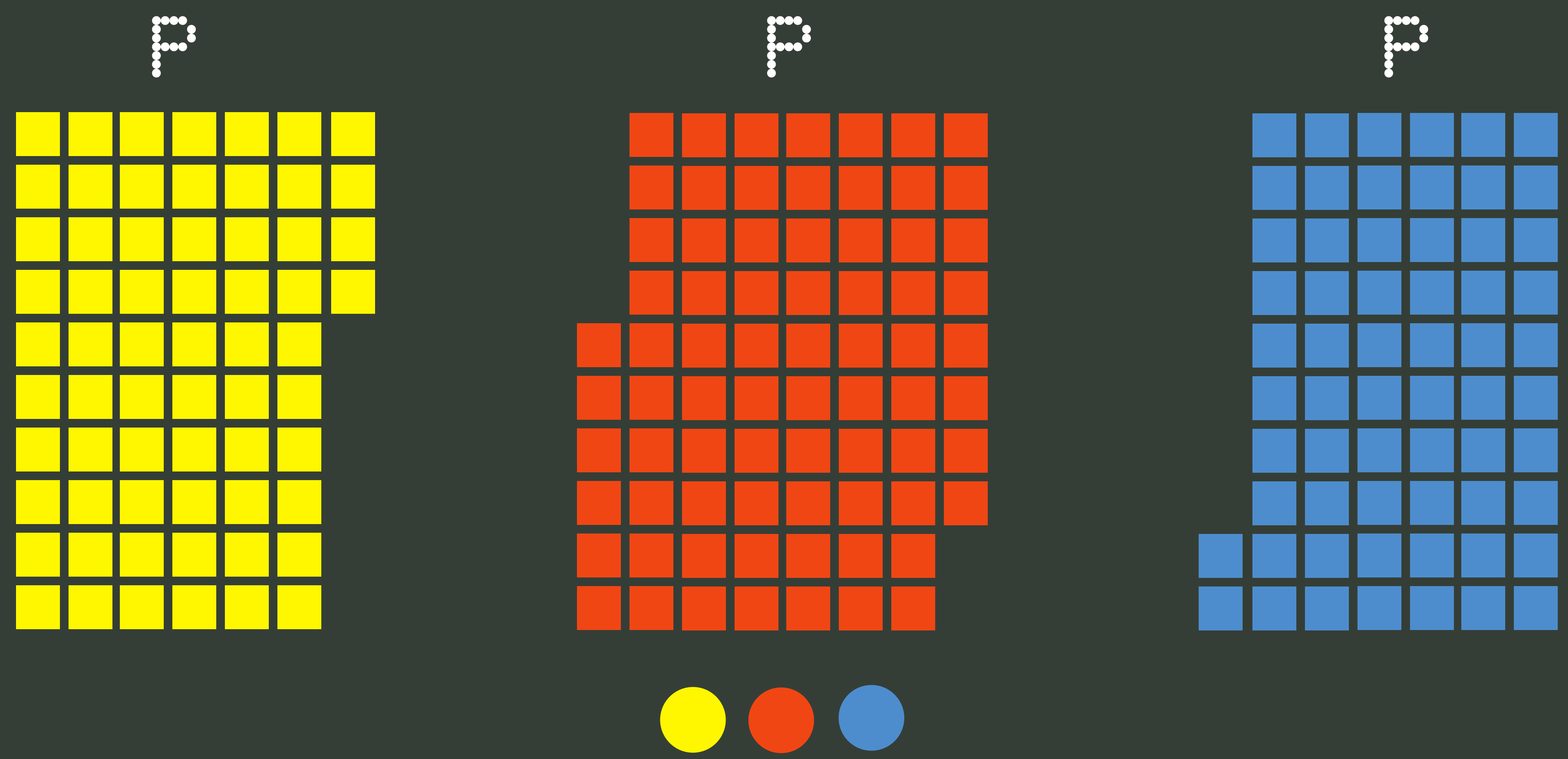
A Zoo of Query Representations



$$Q'\ g\ agg\ \mathrm{P}\ \equiv$$

$$[\ g(\mathrm{y})\ |\ \mathrm{y} \leftarrow \mathrm{P}\ ]^{agg}$$

A Zoo of Query Representations



$$Q'\ g\ agg\ \mathrm{P}\ \equiv$$

$$[\ g(\mathrm{y})\ |\ \mathrm{y}\ \leftarrow\ \mathrm{P}\ ]^{agg}$$

A Zoo of Query Representations

A Zoo of Query Representations

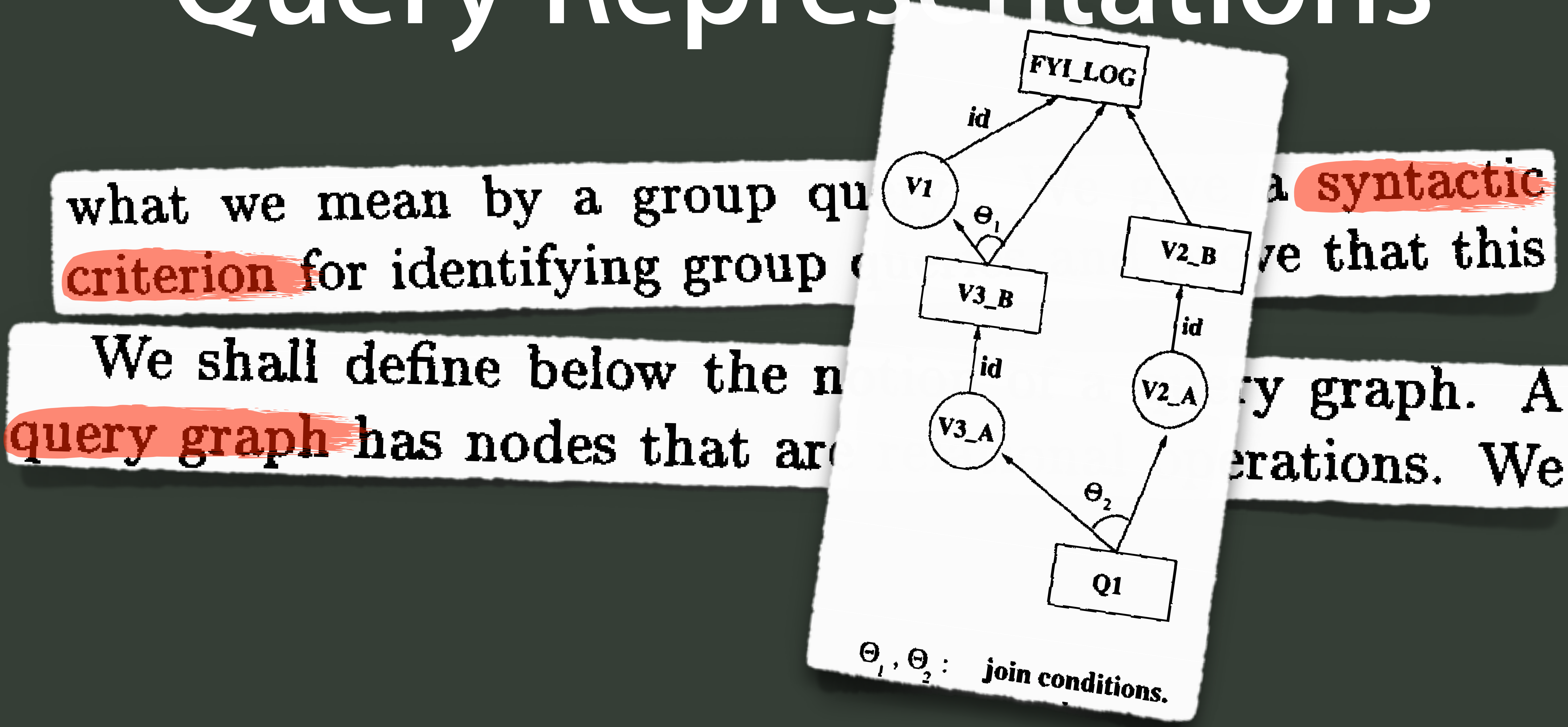
what we mean by a group query. We give a syntactic criterion for identifying group queries and prove that this

A Zoo of Query Representations

what we mean by a group query. We give a syntactic criterion for identifying group queries and prove that this

We shall define below the notion of a query graph. A query graph has nodes that are relational operations. We

A Zoo of Query Representations



SQL surface syntax, relational algebra,
query graphs + annotations, iteration

A Uniform Query Representation

A Uniform Query Representation

$$Q'\, g\; agg\; \mathtt{P} = [\, g(\mathtt{y}) \mid \mathtt{y} \leftarrow \mathtt{P}\,]^{agg}$$

$$\mathtt{partition}\, f\, \mathtt{xs} = [\, \langle f(\mathtt{x}), [\,\mathtt{y} \mid \mathtt{y} \leftarrow \mathtt{xs}, f(\mathtt{x}) = f(\mathtt{y})\,]^{M} \rangle \mid \mathtt{x} \leftarrow \mathtt{xs}\,]^{set}$$

$$\mathtt{map}\, f\, \mathtt{xs} = [\, f(\mathtt{x}) \mid \mathtt{x} \leftarrow \mathtt{xs}\,]^{M}$$

A Uniform Query Representation

$$Q'\,g\,agg\;\mathtt{P} = [\, g(\mathtt{y}) \mid \mathtt{y} \leftarrow \mathtt{P}\,]^{agg}$$

$$\mathtt{partition}\,f\,\mathtt{xs} = [\,\langle f(\mathtt{x}), [\,\mathtt{y} \mid \mathtt{y} \leftarrow \mathtt{xs}, f(\mathtt{x}) = f(\mathtt{y})\,]^{M}\rangle \mid \mathtt{x} \leftarrow \mathtt{xs}\,]^{set}$$

$$\mathtt{map}\,f\,\mathtt{xs} = [\, f(\mathtt{x}) \mid \mathtt{x} \leftarrow \mathtt{xs}\,]^{M}$$

$$\mathtt{map}\;(\lambda\langle \mathtt{x},\mathtt{P}\rangle.\langle \mathtt{x}, Q'\,g\,agg\;\mathtt{P}\rangle)\;(\mathtt{partition}\,f\,\mathtt{xs})$$

A Uniform Query Representation

$$Q'\,g\,agg\ \mathrm{P} = [\,g(\mathrm{y}) \mid \mathrm{y} \leftarrow \mathrm{P}\,]^{agg}$$

$$\mathrm{partition}\,f\,\mathrm{xs} = [\,\langle f(\mathrm{x}), [\,\mathrm{y} \mid \mathrm{y} \leftarrow \mathrm{xs}, f(\mathrm{x}) = f(\mathrm{y})\,]^{M}\rangle \mid \mathrm{x} \leftarrow \mathrm{xs}\,]^{set}$$

$$\mathrm{map}\,f\,\mathrm{xs} = [\,f(\mathrm{x}) \mid \mathrm{x} \leftarrow \mathrm{xs}\,]^{M}$$

$$\mathrm{map}\ (\lambda\langle \mathrm{x},\mathrm{P}\rangle.\langle \mathrm{x}, Q'\,g\,agg\ \mathrm{P}\rangle)\ (\mathrm{partition}\,f\,\mathrm{xs})$$

$$[\,\langle f(\mathrm{x}), [\,g(\mathrm{y}) \mid \mathrm{y} \leftarrow \mathrm{R}, f(\mathrm{y}) = f(\mathrm{x})\,]^{agg}\rangle \mid \mathrm{x} \leftarrow \mathrm{R}\,]^{set}$$

A Uniform Query Representation

$$Q'\,g\;agg\;\mathrm{P} = [\,g(\mathrm{y}) \mid \mathrm{y} \leftarrow \mathrm{P}\,]^{agg}$$

$$\mathtt{partition}\;f\;\mathtt{xs} = [\,\langle f(\mathrm{x}), [\,\mathrm{y} \mid \mathrm{y} \leftarrow \mathtt{xs}, f(\mathrm{x}) = f(\mathrm{y})\,]^{M}\rangle \mid \mathrm{x} \leftarrow \mathtt{xs}\,]^{set}$$

$$\mathtt{map}\;f\;\mathtt{xs} = [\,f(\mathrm{x}) \mid \mathrm{x} \leftarrow \mathtt{xs}\,]^{M}$$

$$\mathtt{map}\;(\lambda\langle \mathrm{x},\mathrm{P}\rangle.\langle \mathrm{x}, Q'\,g\;agg\;\mathrm{P}\rangle)\;(\mathtt{partition}\;f\;\mathtt{xs})$$

```
SELECT    f(x), agg(g(x))
FROM      R AS x
GROUP BY  f(x)
```

XPath

XPath Comprehensions

XPath Comprehensions

```
/descendant::a[following::b]/child::c
```

XPath Comprehensions

```
/descendant::a[following::b]/child::c
```

1. Normalize, simplify, **flip** XPath step expressions

XPath Comprehensions

```
/descendant::a[following::b]/child::c
```

1. Normalize, simplify, **flip** XPath step expressions
2. **Compile** XPath into queries over tabular XML encoding

XPath Comprehensions

`/descendant::a[following::b]/child::c`

1. Normalize, simplify, **flip** XPath step expressions
2. **Compile** XPath into queries over tabular XML encoding

$$\text{xpath}\ (/p)\ c = \text{xpath}\ p\ (\text{root}\ c)$$

$$\text{xpath}\ (p_1/p_2)\ c = [\ n' \mid n \leftarrow \text{xpath}\ p_1\ c,\ n' \leftarrow \text{xpath}\ p_2\ n\ ]^X$$

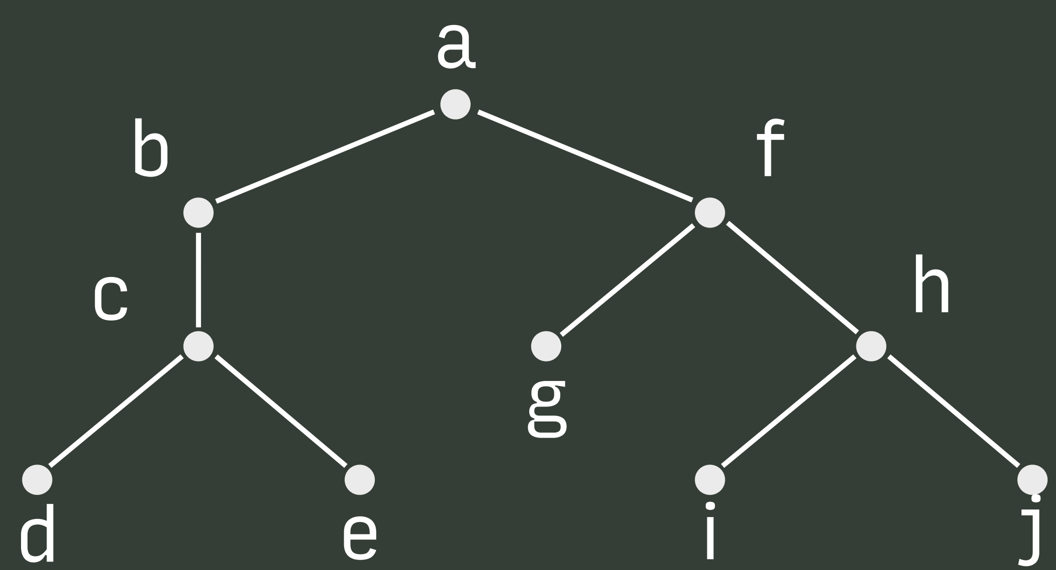
$$\text{xpath}\ (p\,[\,q\,])\ c = [\ n \mid n \leftarrow \text{xpath}\ p\ c,\ [\ \text{true} \mid \_ \leftarrow \text{xpath}\ q\ n\ ]^{some}\ ]^X$$

$$\text{xpath}\ (ax::t)\ c = \text{step}\ (ax::t)\ c$$

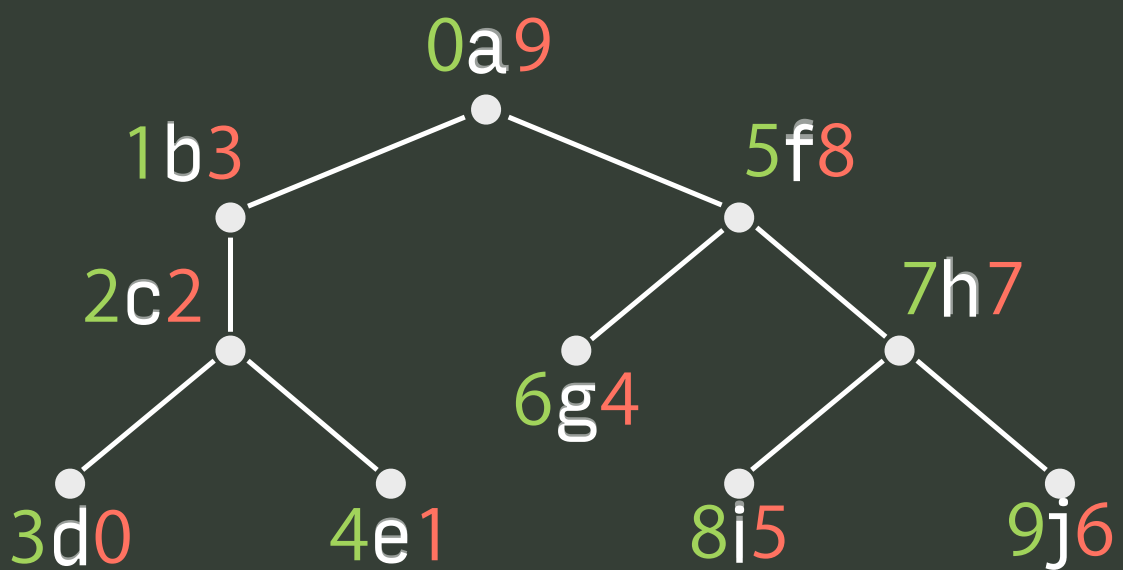
A Tabular XML Encoding

```
<a>
  <b><c><d/>e</c></b>
  <f><!--g-->
     <h><i/><j/></h>
  </f>
</a>
```

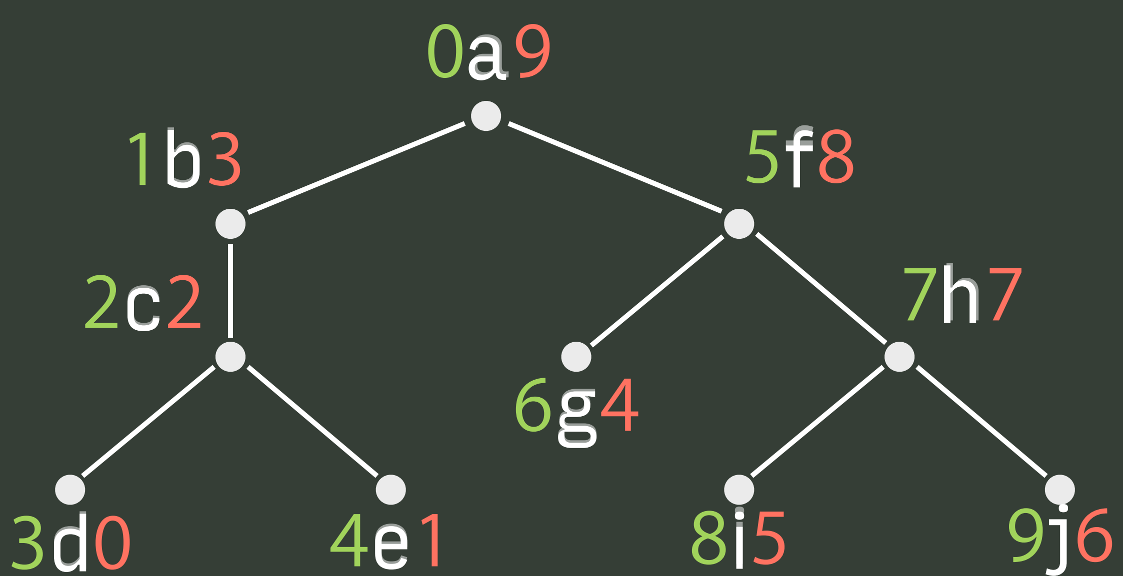
A Tabular XML Encoding

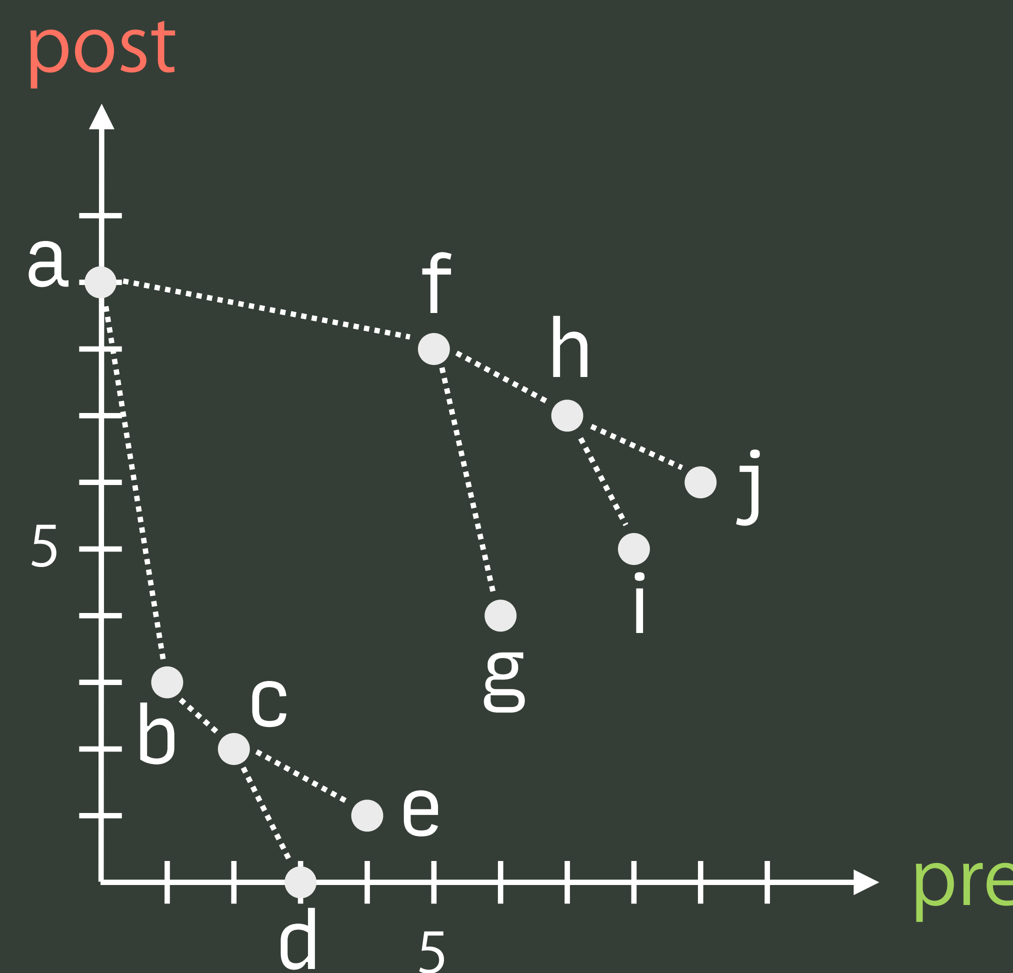


A Tabular XML Encoding

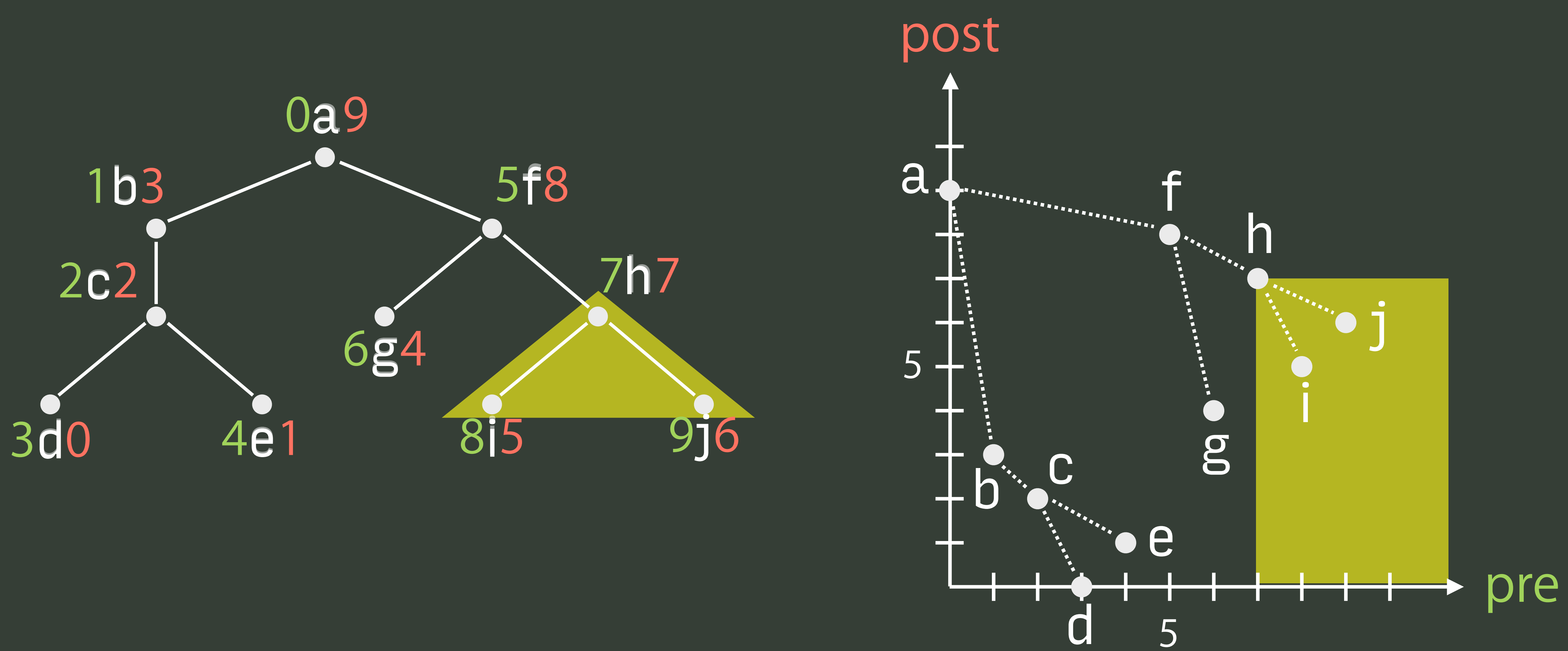


A Tabular XML Encoding

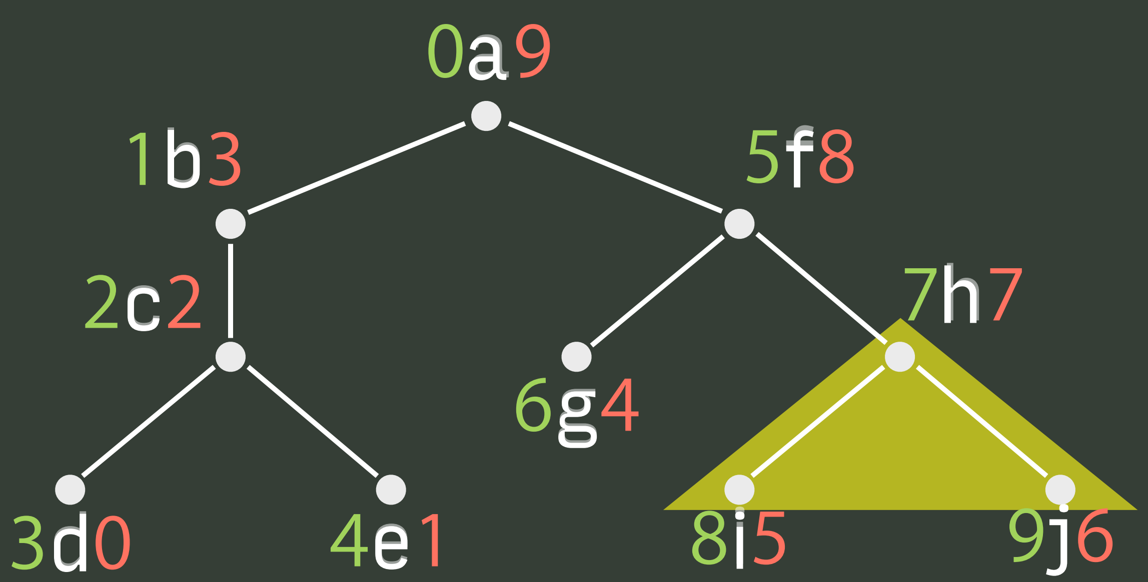


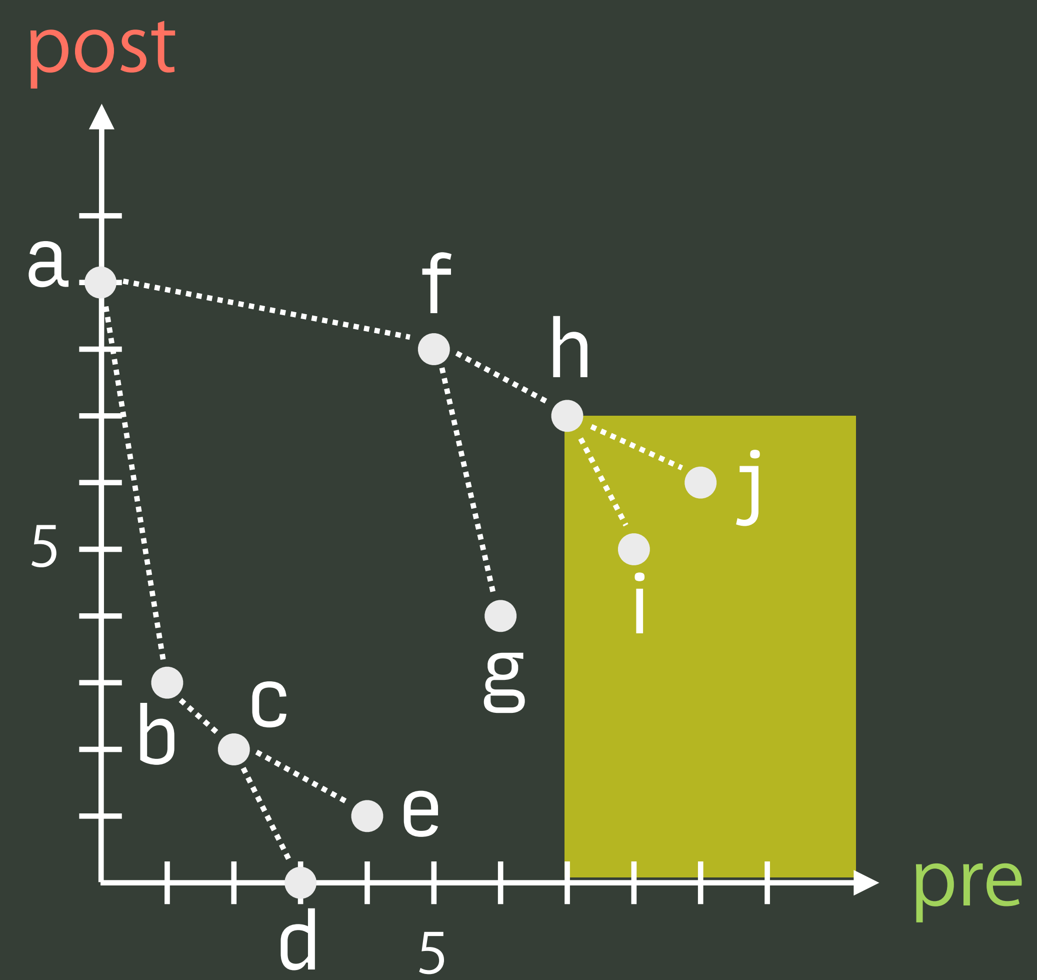


A Tabular XML Encoding



A Tabular XML Encoding

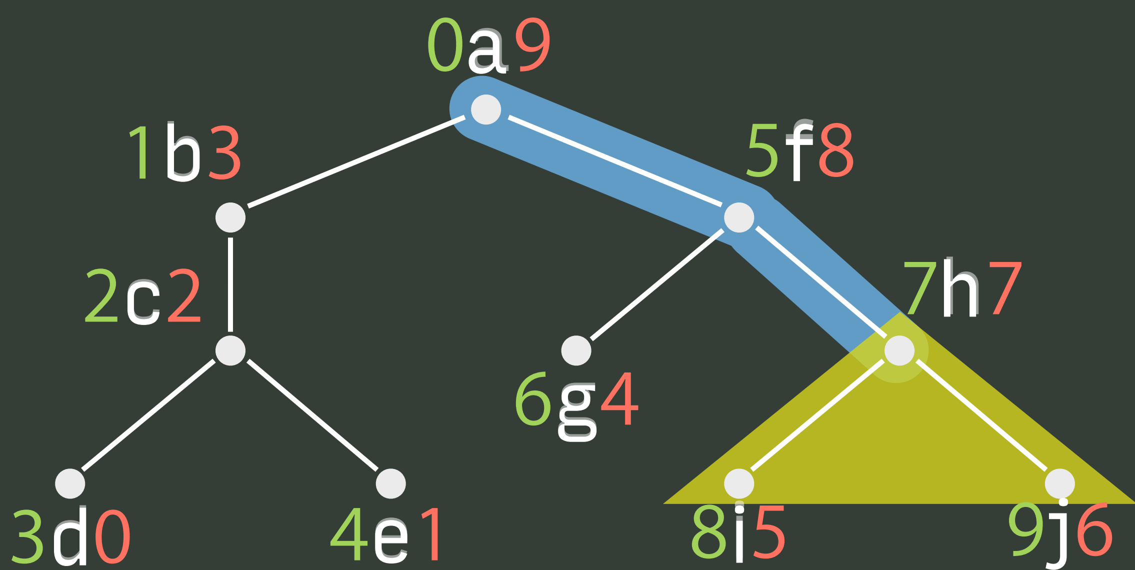


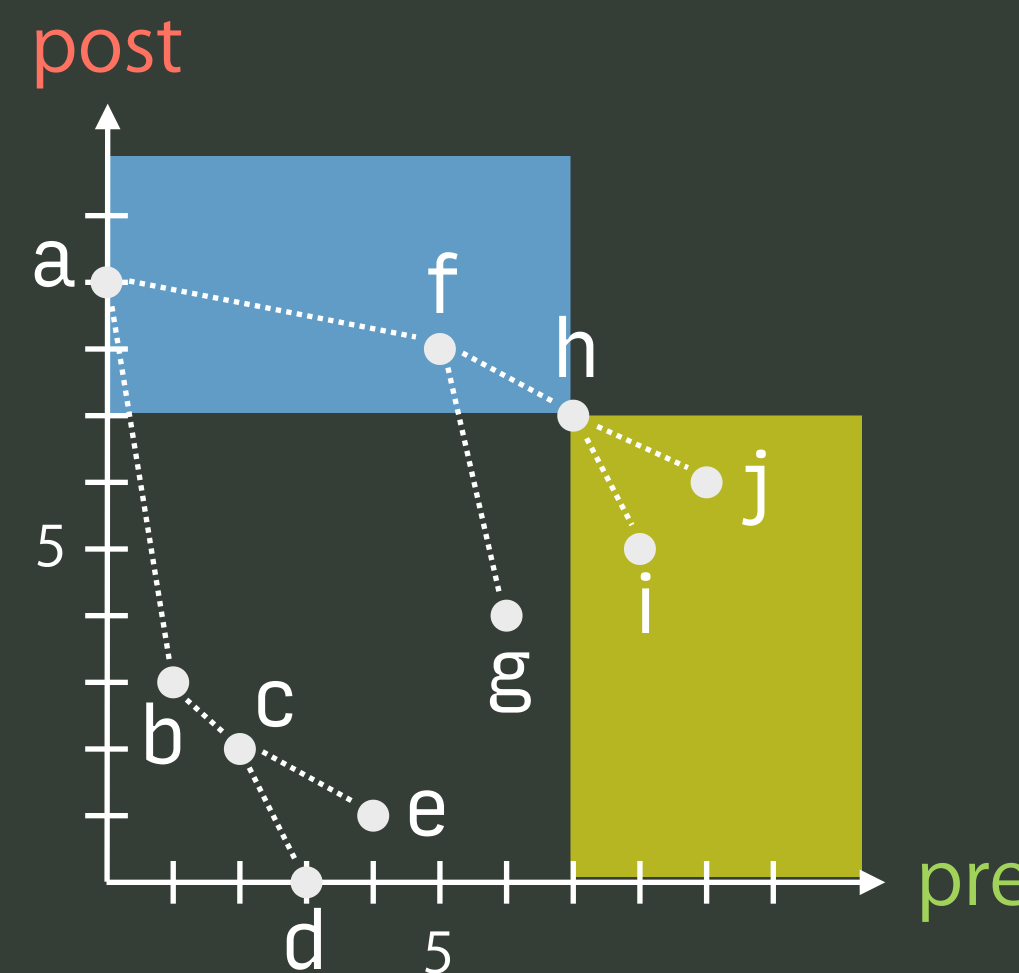


step (`descendant::`t) c =

$$[\, n \mid n \leftarrow \text{doc},\ \text{pre}\ c < \text{pre}\ n,\ \text{post}\ c > \text{post}\ n,\ \text{tag}\ n = t \,]^X$$

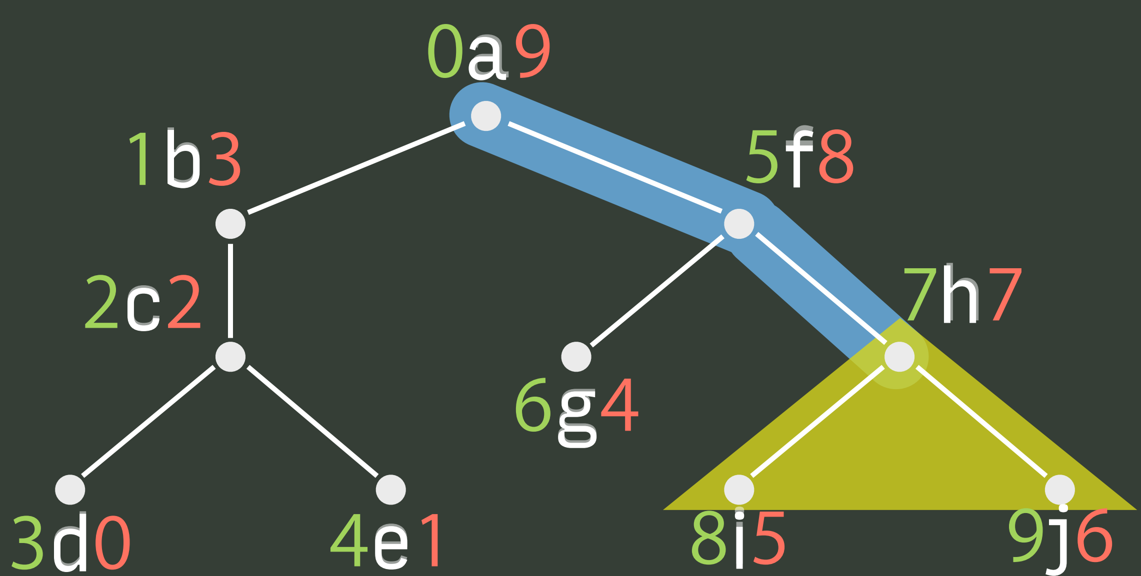
A Tabular XML Encoding

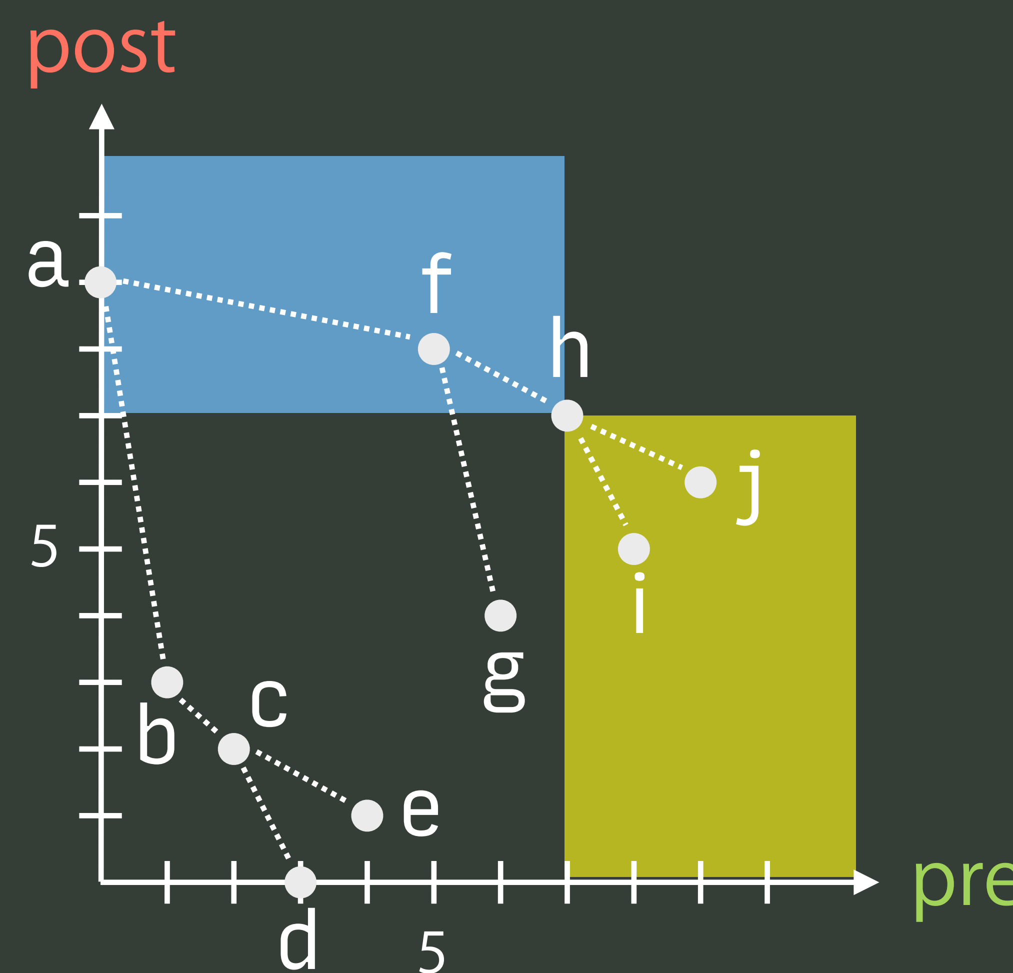




$$\text{step}\,(\texttt{descendant::}t)\;c \;=\; [\; n \mid n \leftarrow \text{doc},\; \text{pre}\; c < \text{pre}\; n,\; \text{post}\; c > \text{post}\; n,\; \text{tag}\; n = t \;]^X$$

A Tabular XML Encoding





$$\text{step}\,(\texttt{descendant::}t)\;c \;=\; [\, n \mid n \leftarrow \text{doc},\ \text{pre}\; c < \text{pre}\; n,\ \text{post}\; c > \text{post}\; n,\ \text{tag}\; n = t \,]^X$$

$$\text{step}\,(\texttt{ancestor::}t)\;c \;=\; [\, n \mid n \leftarrow \text{doc},\ \text{pre}\; c > \text{pre}\; n,\ \text{post}\; c < \text{post}\; n,\ \text{tag}\; n = t \,]^X$$

XPath: Looking Forward

XPath: Looking Forward

Dan Olteanu, Holger Meuss, Tim Furche, François Bry

Insitute for Computer Science and Center for Information and Language Processing
University of Munich, Germany

XPath: Looking Forward
D. Olteanu et al., XMLDM (EDBT 2002), March 2002

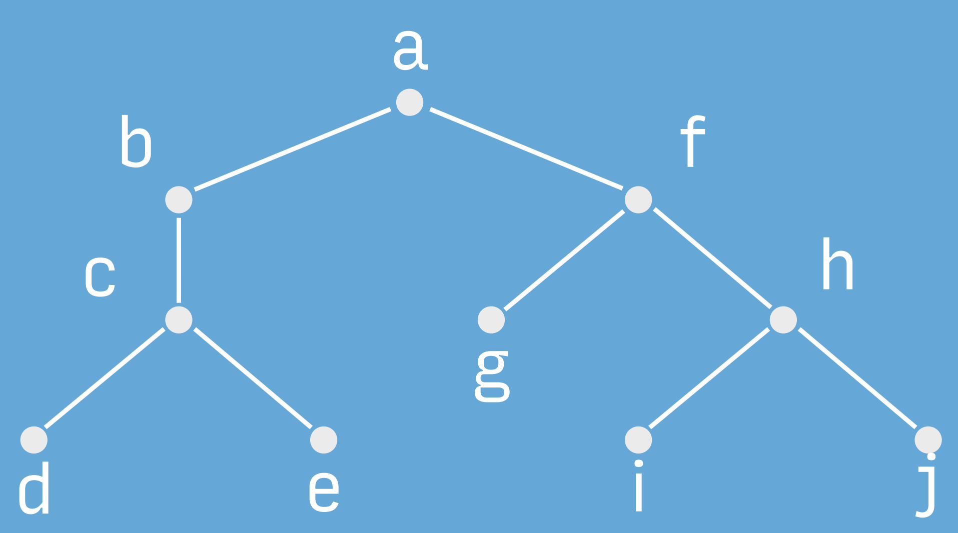
XPath: Looking Forward

XPath: Looking Forward

Dan Olteanu, Holger Meuss, Tim Furche, François Bry

Insitute for Computer Science and Center for Information and Language Processing
University of Munich, Germany

XPath: Looking Forward
D. Olteanu et al., XMLDM (EDBT 2002), March 2002



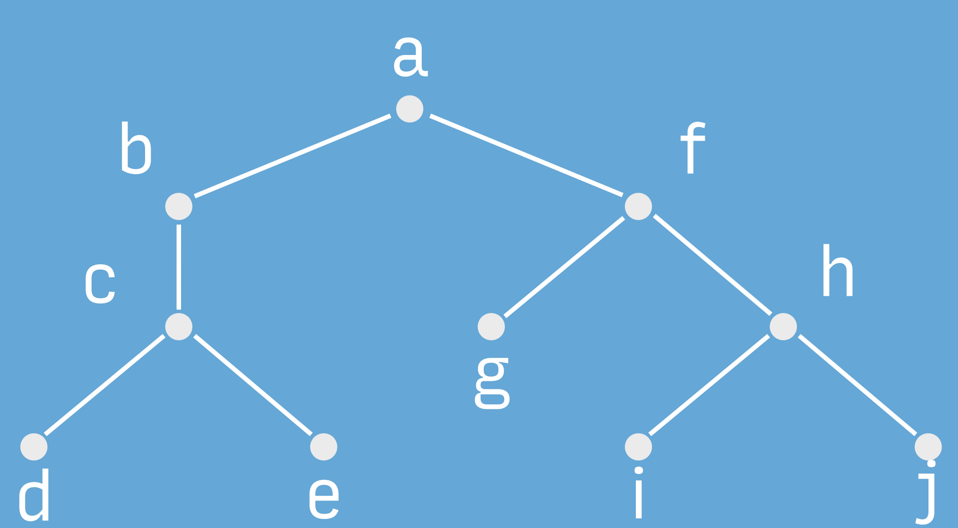
XPath: Looking Forward

XPath: Looking Forward

Dan Olteanu, Holger Meuss, Tim Furche, François Bry

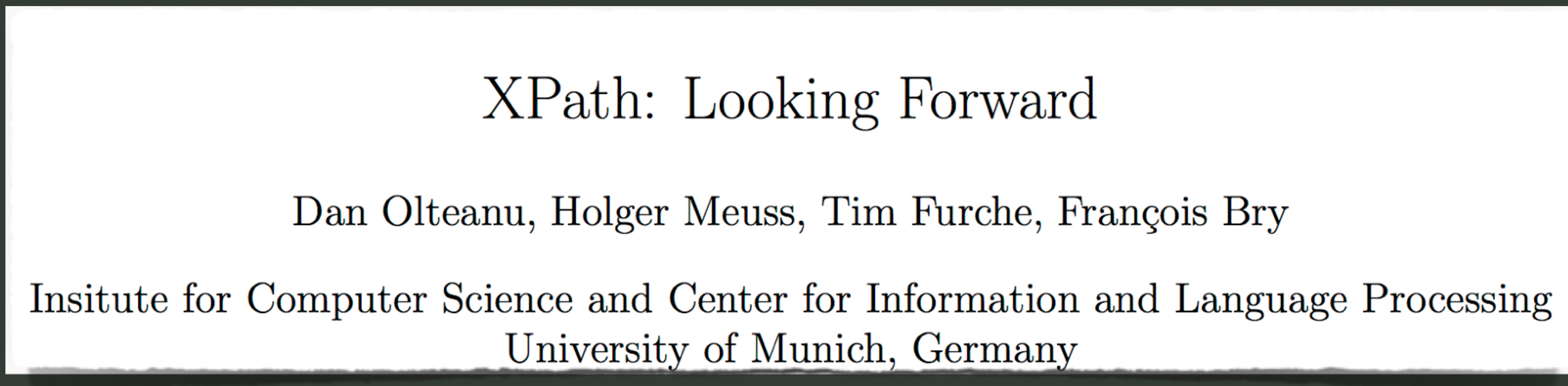
Insitute for Computer Science and Center for Information and Language Processing
University of Munich, Germany

XPath: Looking Forward
D. Olteanu et al., XMLDM (EDBT 2002), March 2002




```
/descendant::g/preceding::c
```

XPath: Looking Forward

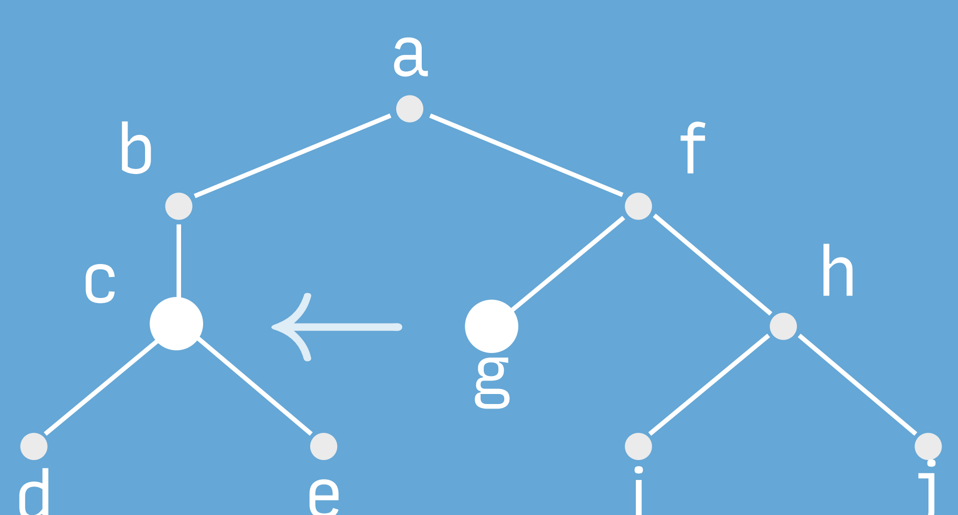


XPath: Looking Forward

Dan Olteanu, Holger Meuss, Tim Furche, François Bry

Insitute for Computer Science and Center for Information and Language Processing
University of Munich, Germany

XPath: Looking Forward
D. Olteanu et al., XMLDM (EDBT 2002), March 2002



`/descendant::g/preceding::c`

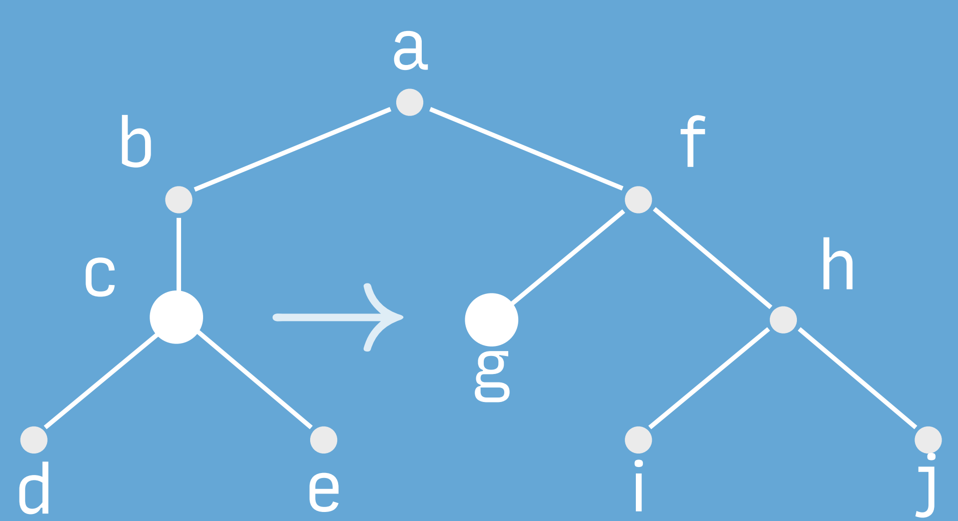
XPath: Looking Forward

XPath: Looking Forward

Dan Olteanu, Holger Meuss, Tim Furche, François Bry

Insitute for Computer Science and Center for Information and Language Processing
University of Munich, Germany

XPath: Looking Forward
D. Olteanu et al., XMLDM (EDBT 2002), March 2002




```
/descendant::g/preceding::c
```

≡

```
/descendant::c[following::g]
```

XPath: Looking Forward

XPath: Looking Forward

$$
\begin{aligned}
p/\texttt{descendant::}n[\texttt{preceding::}m] &\equiv p[\texttt{preceding::}m]/\texttt{descendant::}n && (38)\\
&\mid p/\texttt{child::*}[\texttt{descendant-or-self::}m]\\
&\quad /\texttt{following-sibling::*}/\texttt{descendant-or-self::}n\\
/\texttt{descendant::}n[\texttt{preceding::}m\ ] &\equiv /\texttt{descendant::}m/\texttt{following::}n && (38a)\\
p/\texttt{child::}n[\texttt{preceding::}m\ ] &\equiv p[\texttt{preceding::}m]/\texttt{child::}n && (39)\\
&\mid p/\texttt{child::*}[\texttt{descendant-or-self::}m]\\
&\quad /\texttt{following-sibling::}n\\
p/\texttt{self::}n[\texttt{preceding::}m] &\equiv p[\texttt{preceding::}m]/\texttt{self::}n && (40)\\
p/\texttt{following-sibling::}n[\texttt{preceding::}m] &\equiv p[\texttt{preceding::}m]/\texttt{following-sibling::}n && (41)\\
&\mid p/\texttt{following-sibling::*}[\texttt{descendant-or-self::}m]\\
&\quad /\texttt{following-sibling::}n\\
&\mid p[\texttt{descendant-or-self::}m]/\texttt{following-sibling::}n\\
p/\texttt{following::}n[\texttt{preceding::}m] &\equiv p[\texttt{preceding::}m]/\texttt{following::}n && (42)\\
&\mid p/\texttt{following::}m/\texttt{following::}n\\
&\mid p[\texttt{descendant-or-self::}m]/\texttt{following::}n
\end{aligned}
$$

Comprehending XPath

Comprehending XPath

```
/descendant::g/preceding::c
```

Comprehending XPath

`/descendant::g/preceding::c`

$$[\, v' \mid v \leftarrow \text{doc},\ \ \text{tag } v = \texttt{'g'},\ v' \leftarrow \text{doc},$$
$$\text{pre } v' < \text{pre } v,\ \ \text{post } v' < \text{post } v,\ \ \text{tag } v' = \texttt{'c'}\ \,]^X$$

Comprehending XPath

`/descendant::g/preceding::c`

$$[\, v' \mid \; v \leftarrow \mathrm{doc},\;\; \mathrm{tag}\ v = \texttt{'g'},\; v' \leftarrow \mathrm{doc},$$
$$\mathrm{pre}\ v' < \mathrm{pre}\ v,\;\; \mathrm{post}\ v' < \mathrm{post}\ v,\;\; \mathrm{tag}\ v' = \texttt{'c'} \;\,]^X$$

`/descendant::c[following::g]`

Comprehending XPath

```
/descendant::g/preceding::c
```

$$[\, v' \mid v \leftarrow \text{doc},\ \ \text{tag } v = \texttt{'g'},\ v' \leftarrow \text{doc},\ \ \text{pre } v' < \text{pre } v,\ \ \text{post } v' < \text{post } v,\ \ \text{tag } v' = \texttt{'c'}\ ]^X$$

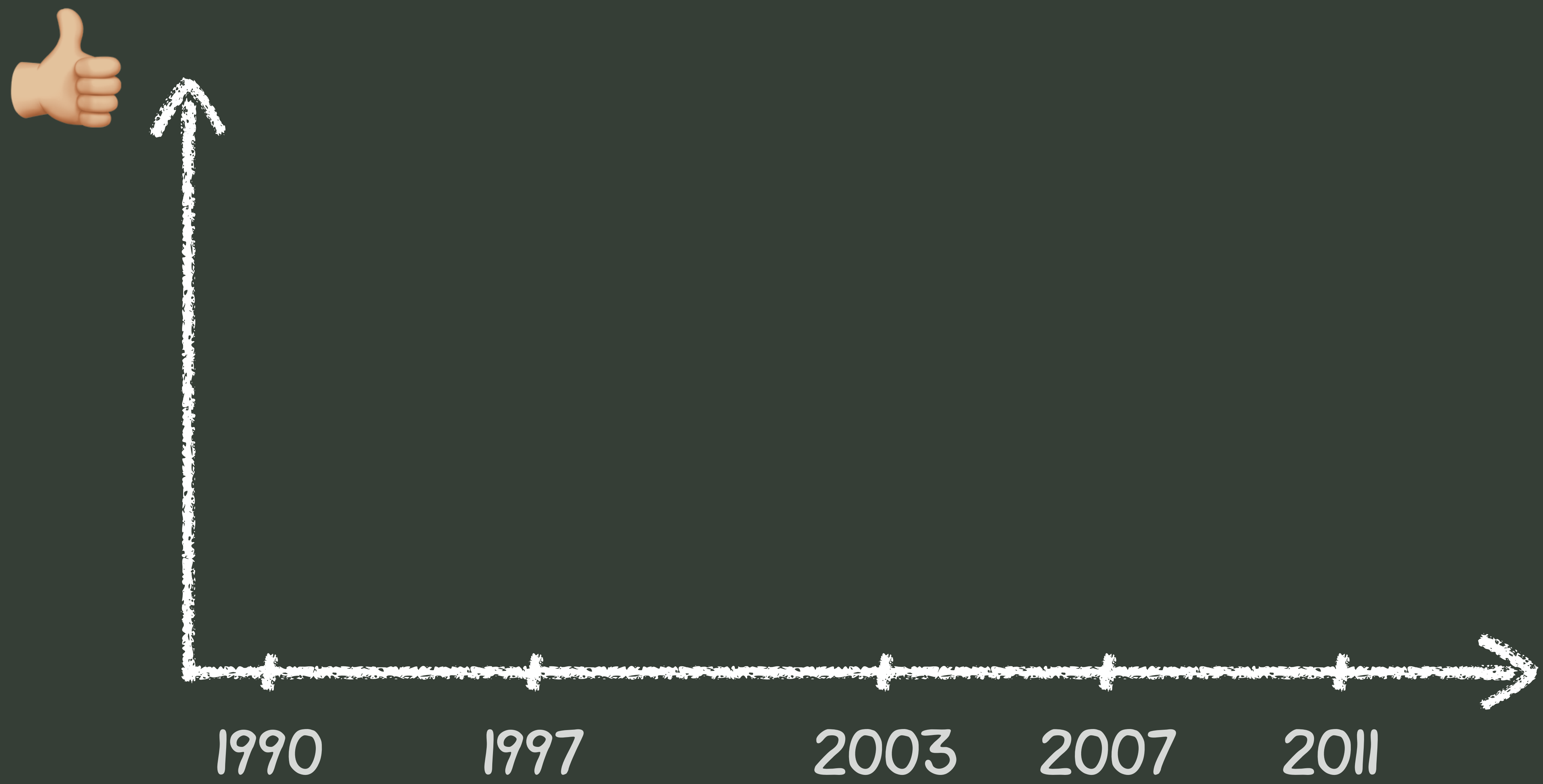
```
SELECT   DISTINCT v'
FROM     doc v, doc v'
WHERE    tag v = 'g' AND tag v' = 'c'
AND      pre v' < pre v AND post v' < post v
ORDER BY pre v'
```


BRING BACK
MONAD
COMPREHENSIONS

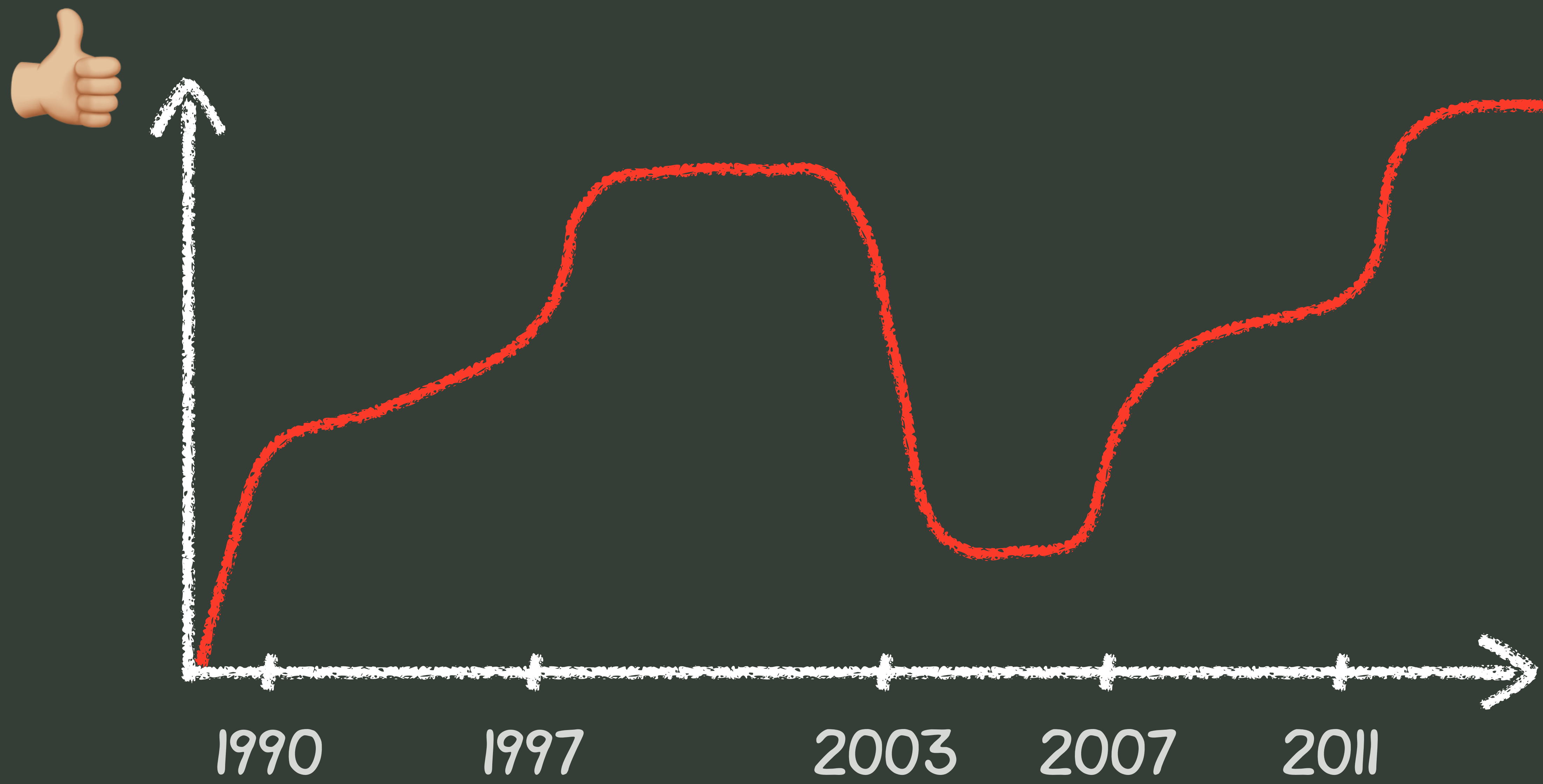
Comprehensions in **Haskell**

Comprehensions in **Haskell**

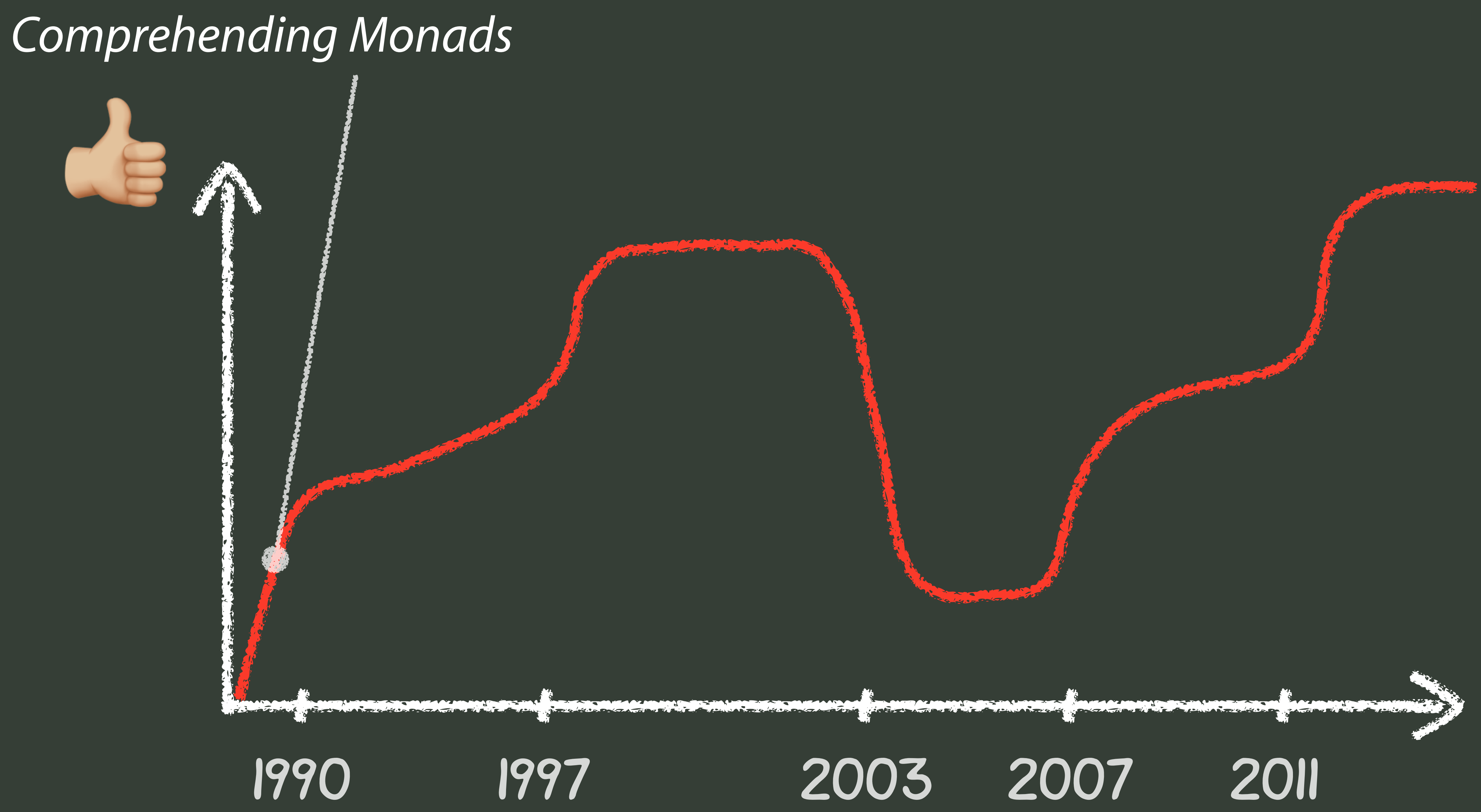
Comprehensions in Haskell



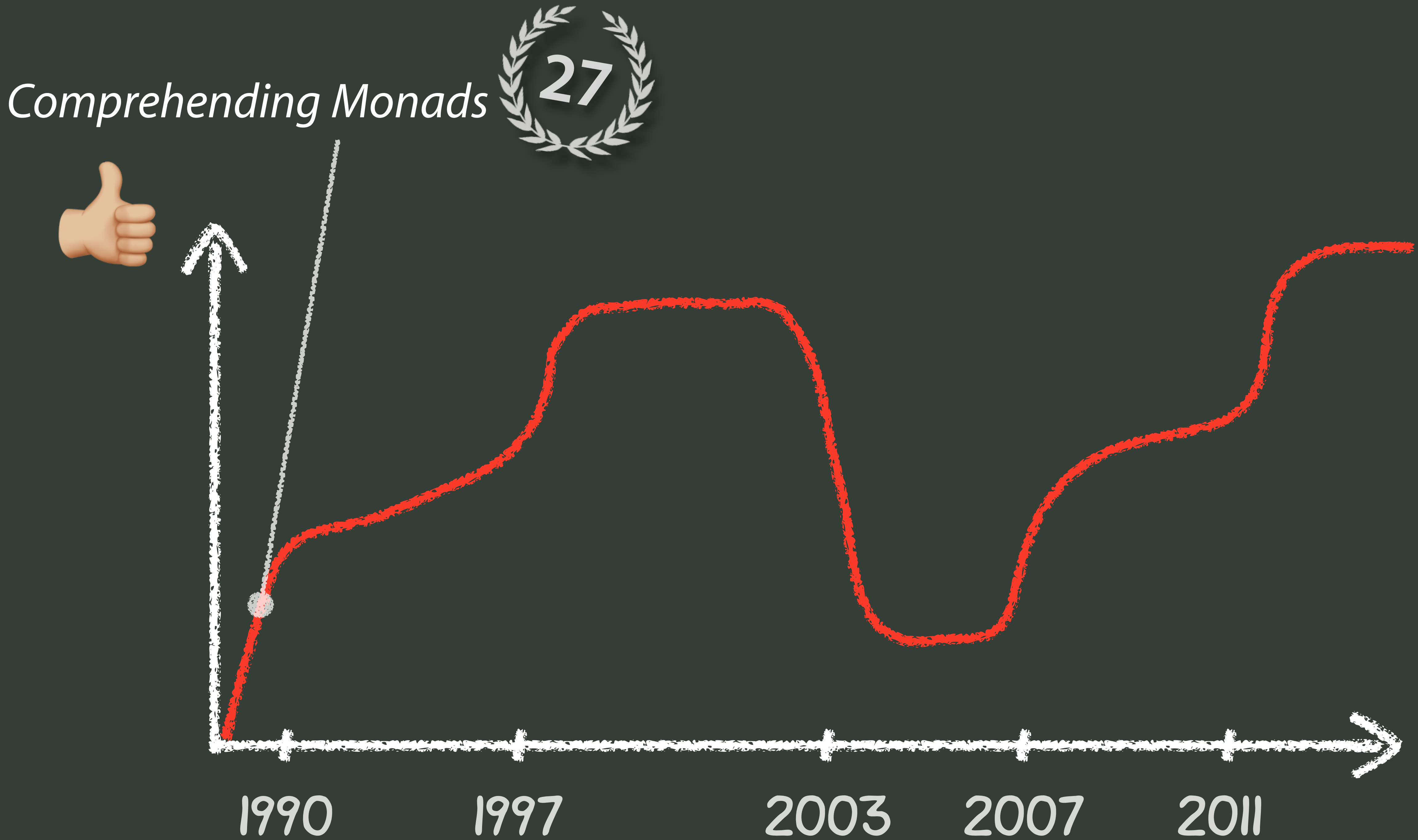
Comprehensions in Haskell



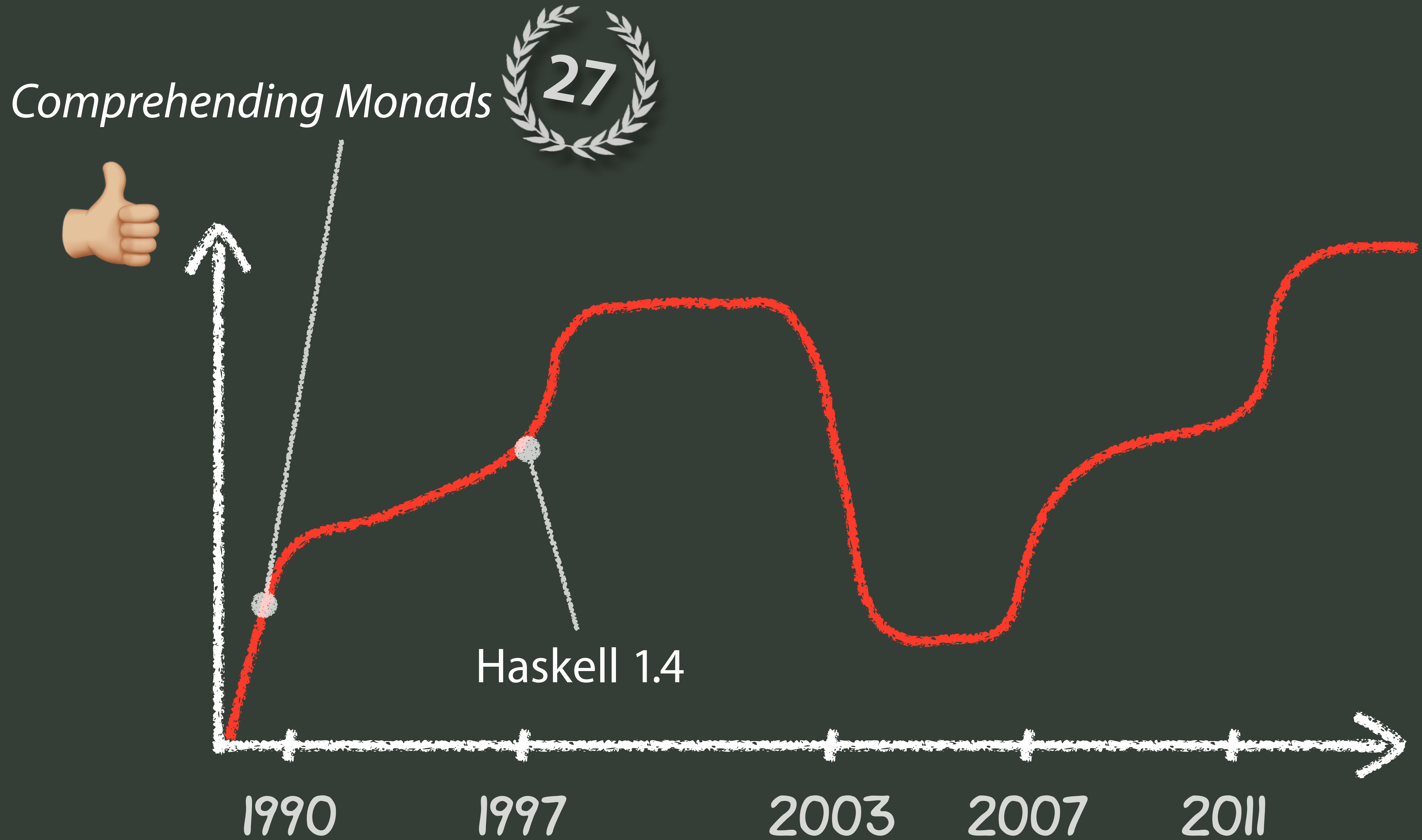
Comprehensions in Haskell



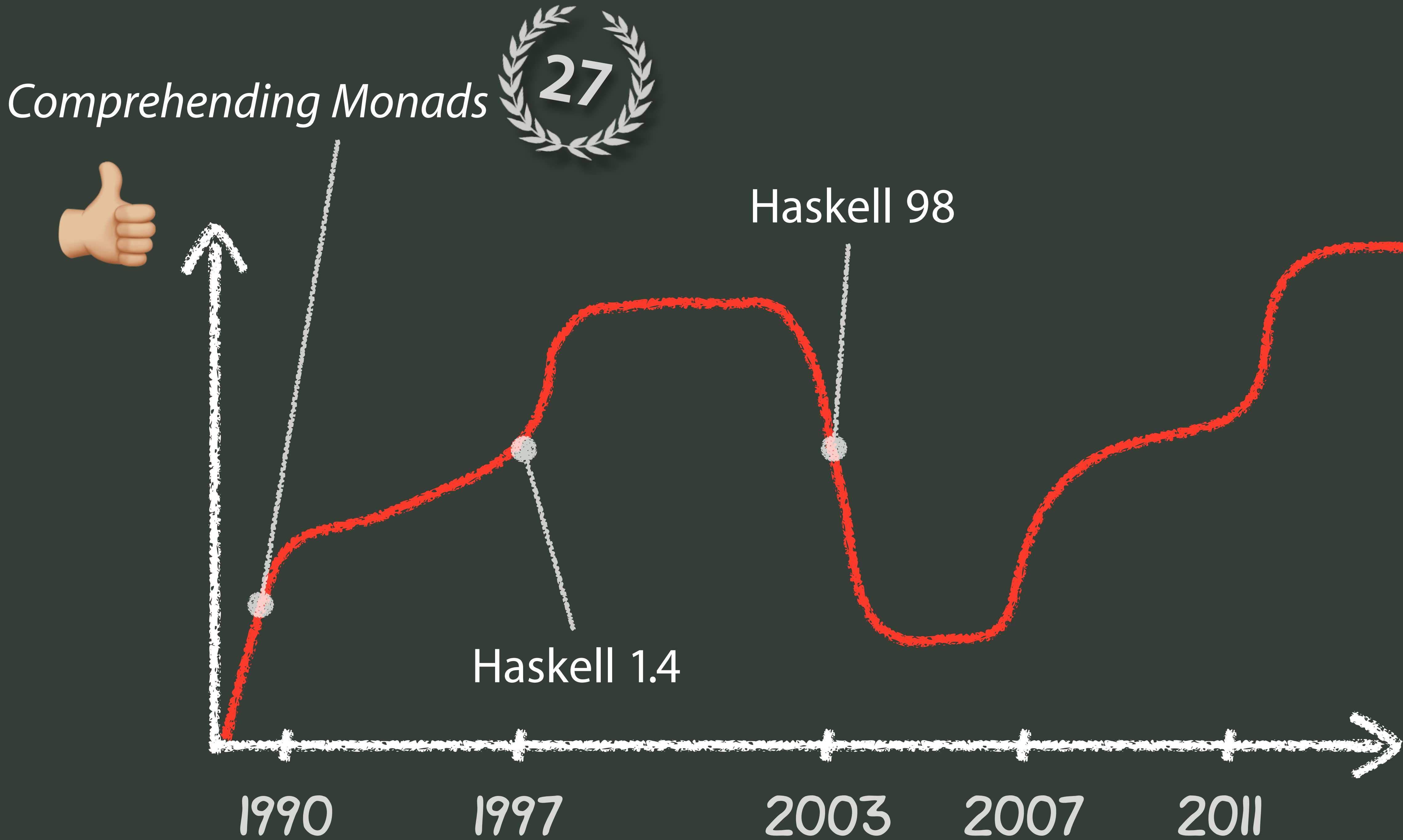
Comprehensions in Haskell



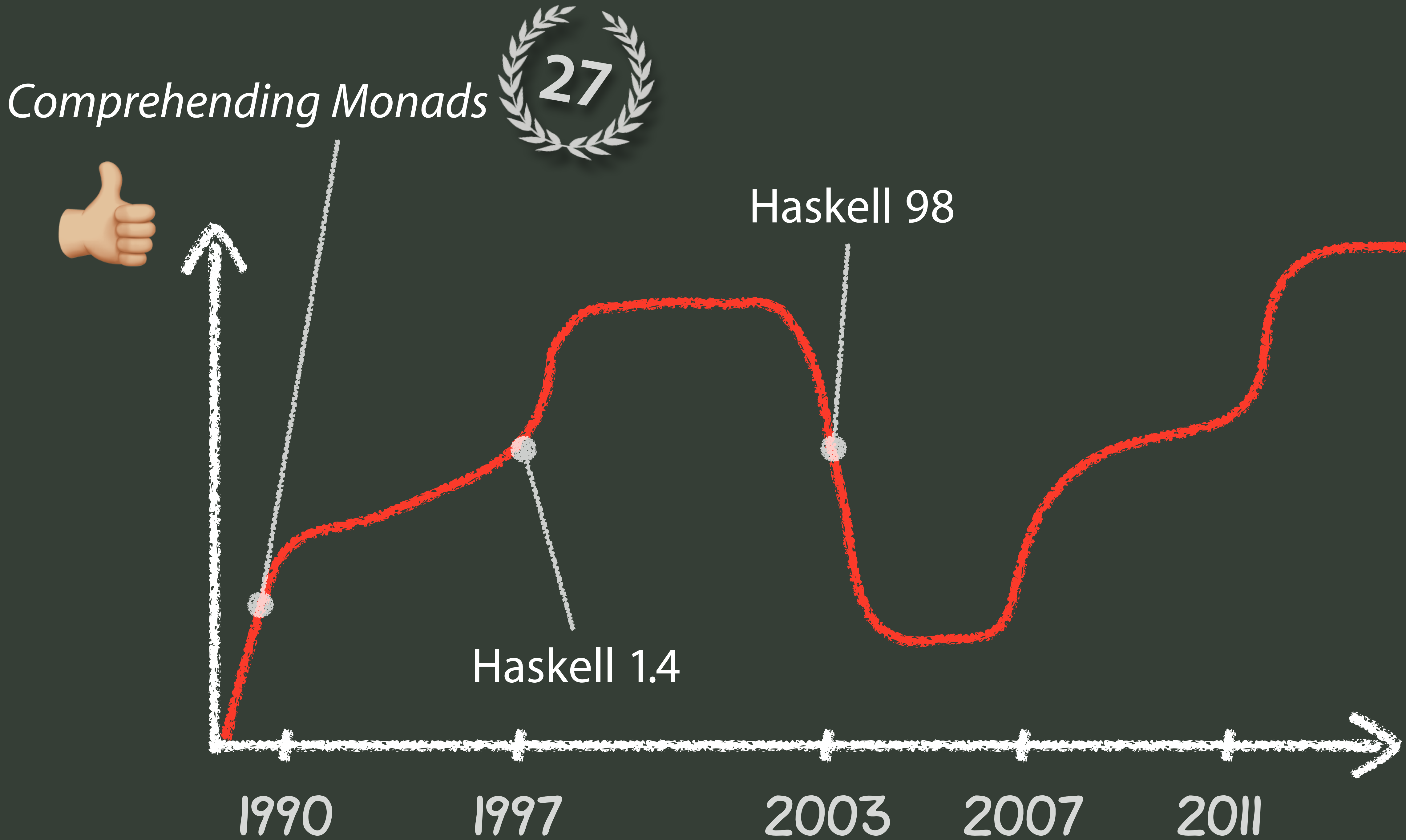
Comprehensions in Haskell



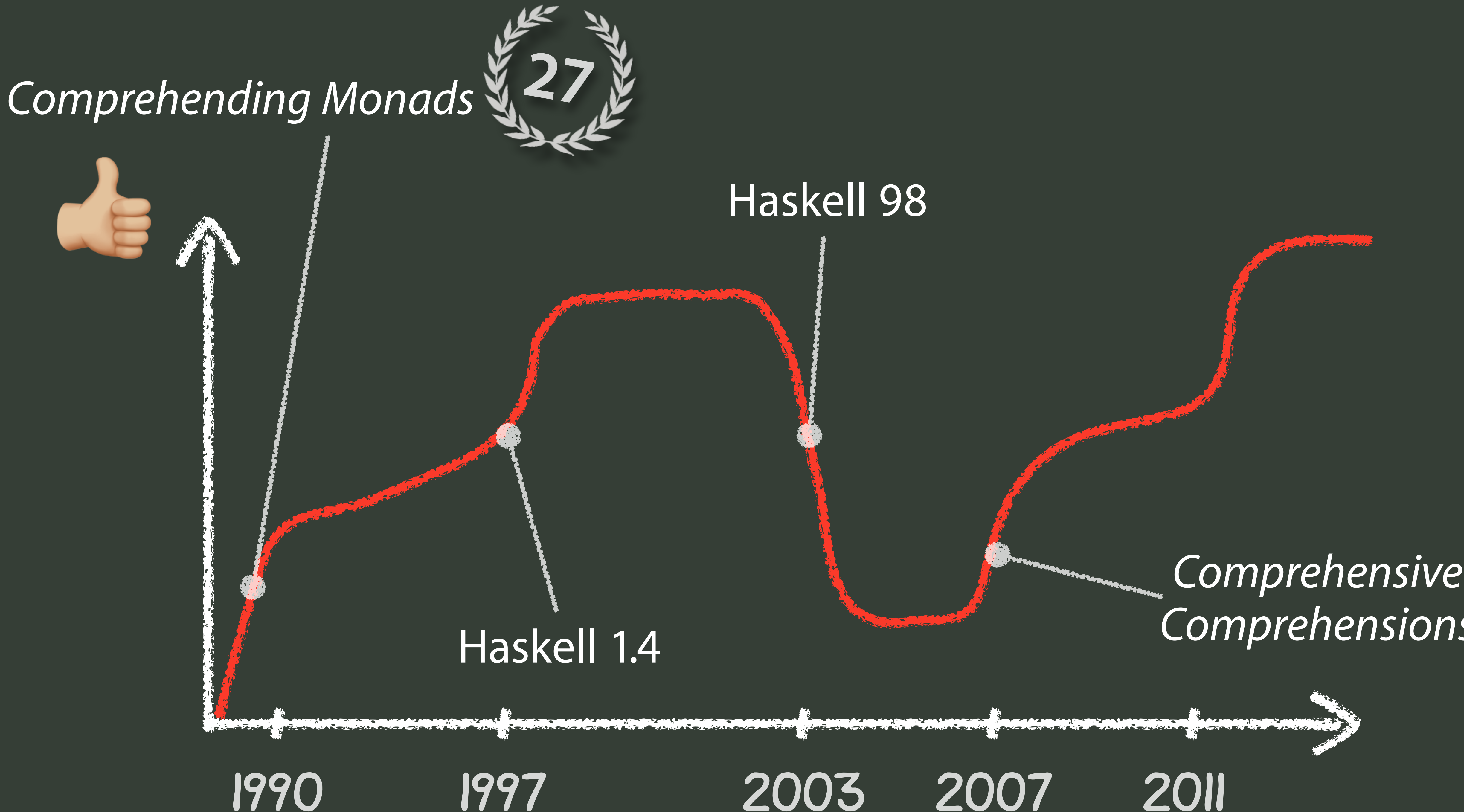
Comprehensions in Haskell



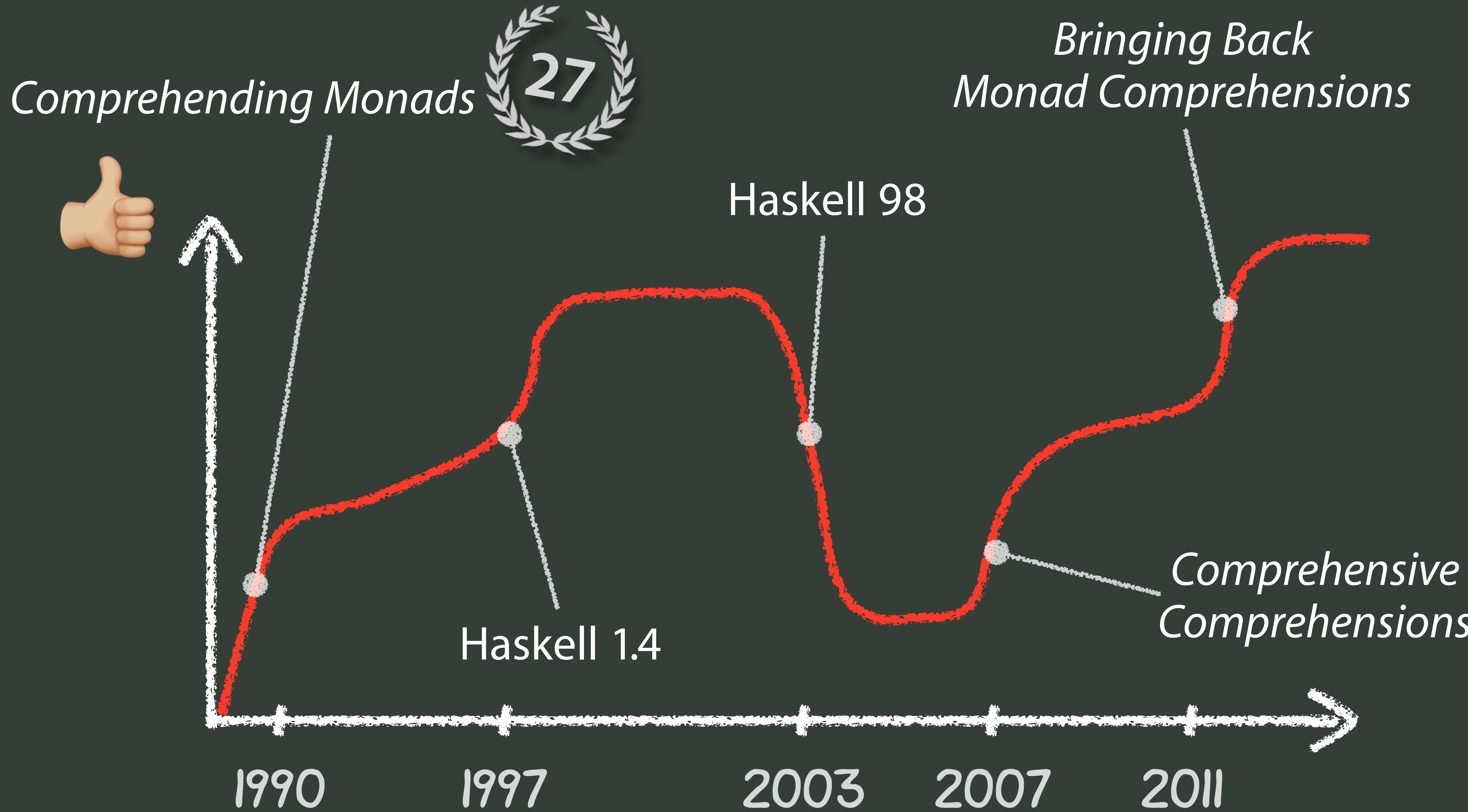
Comprehensions in GHC



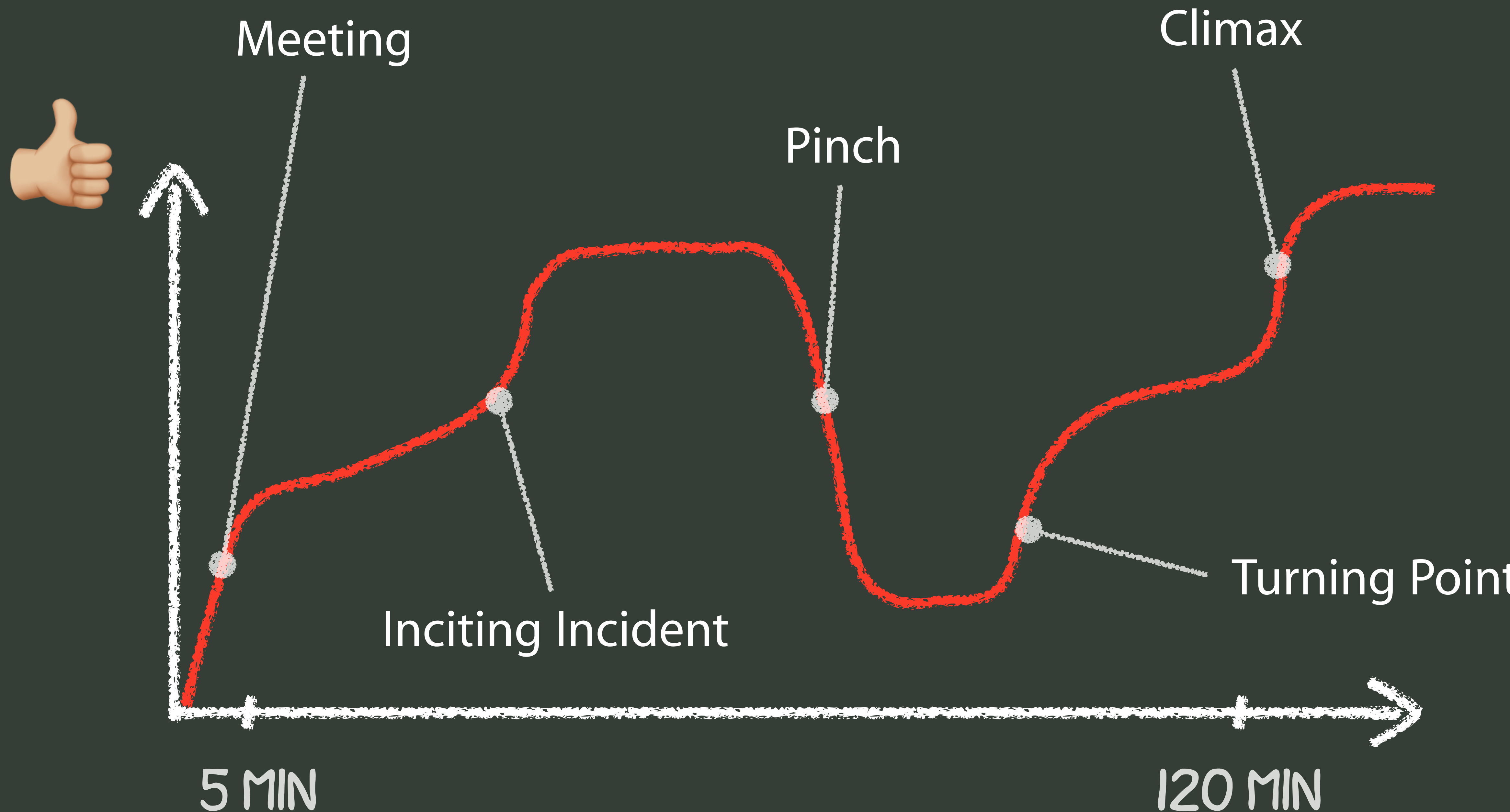
Comprehensions in GHC



Comprehensions in GHC



Movie Plot Line



Comprehensi{ve, ons}

Comprehensive Comprehensions

Comprehensions with ‘Order by’ and ‘Group by’

Philip Wadler
University of Edinburgh

Simon Peyton Jones
Microsoft Research

Comprehensive Comprehensions
P. Wadler, S. Peyton-Jones, Haskell Workshop, October 2007

Comprehensi{ve, ons}

Comprehensive Comprehensions

Comprehensions with ‘Order by’ and ‘Group by’

Philip Wadler
University of Edinburgh

Simon Peyton Jones
Microsoft Research

Comprehensive Comprehensions
P. Wadler, S. Peyton-Jones, Haskell Workshop, October 2007

```
[ (the dept, maximum salary)
| (name, dept, salary) <- employees
, then group by dept using groupWith
, length dept > 10
, then sortWith by Down (sum salary)
, then take 5
]
```

Comprehensi{ve, ons}

Comprehensive Comprehensions

Comprehensions with 'Order by' and 'Group by'

Philip Wadler
University of Edinburgh

Simon Peyton Jones
Microsoft Research

Comprehensive Comprehensions
P. Wadler, S. Peyton-Jones, Haskell W ctober 2007

GROUP BY

AGGR

FROM

```
[ (the dept, maximum salary)
| (name, dept, salary) <- employees
, then group by dept using groupWith
, length dept > 10
, then sortWith by Down (sum salary)
, then take 5
]
```

HAVING

ASC/DESC

ORDER BY

LIMIT

Comprehensi{ve, ons}

Comprehensive Comprehensions

Comprehensions with 'Order by' and 'Group by'

Philip Wadler
University of Edinburgh

Simon Peyton Jones
Microsoft Research

Comprehensive Comprehensions
P. Wadler, S. Peyton-Jones, Haskell Workshop, October 2007

row patterns!

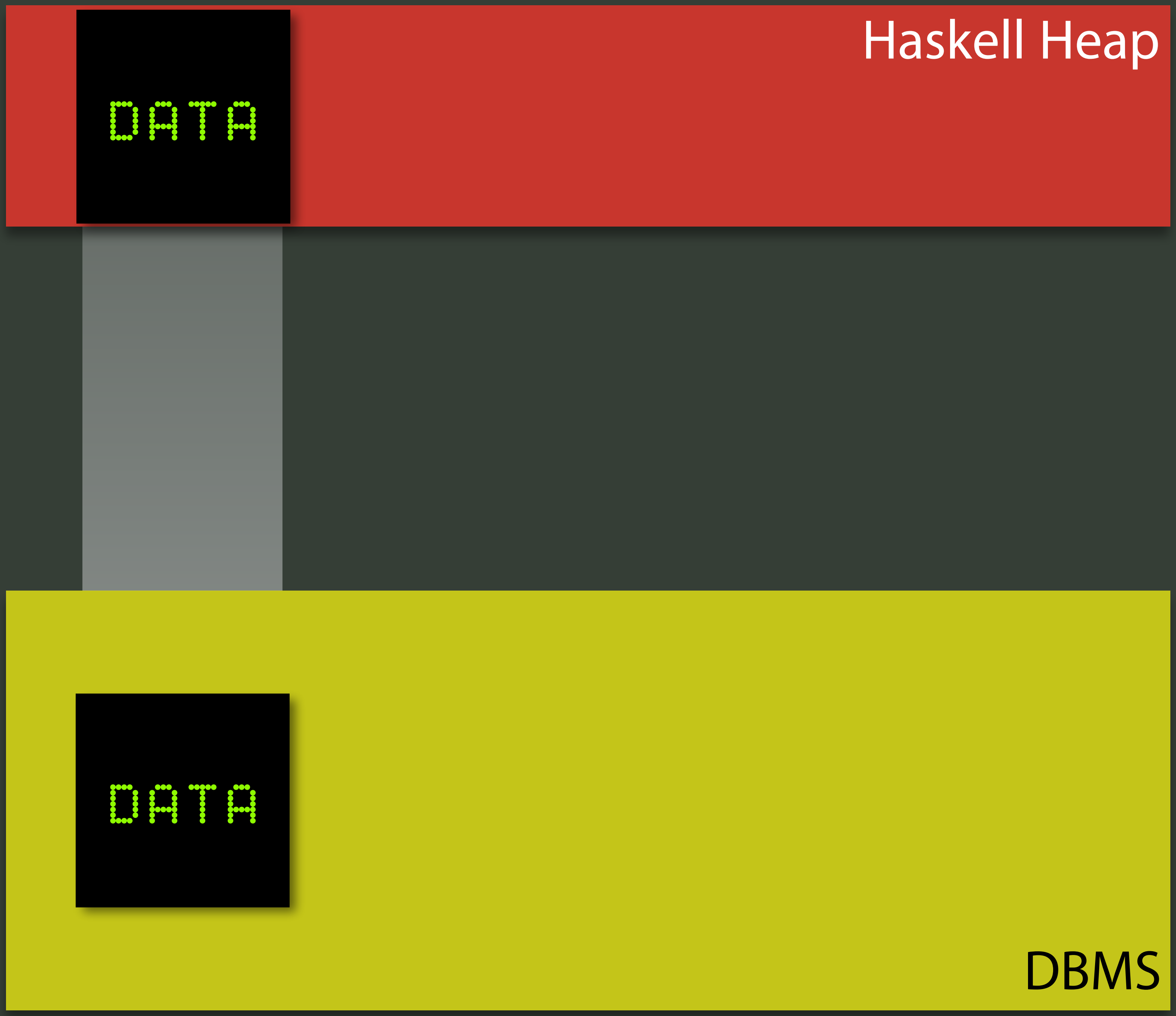
```
[ sum salary
| (name, "MS", salary) <- employees
, then group using runs 3
, then take 5
]
```

OVER

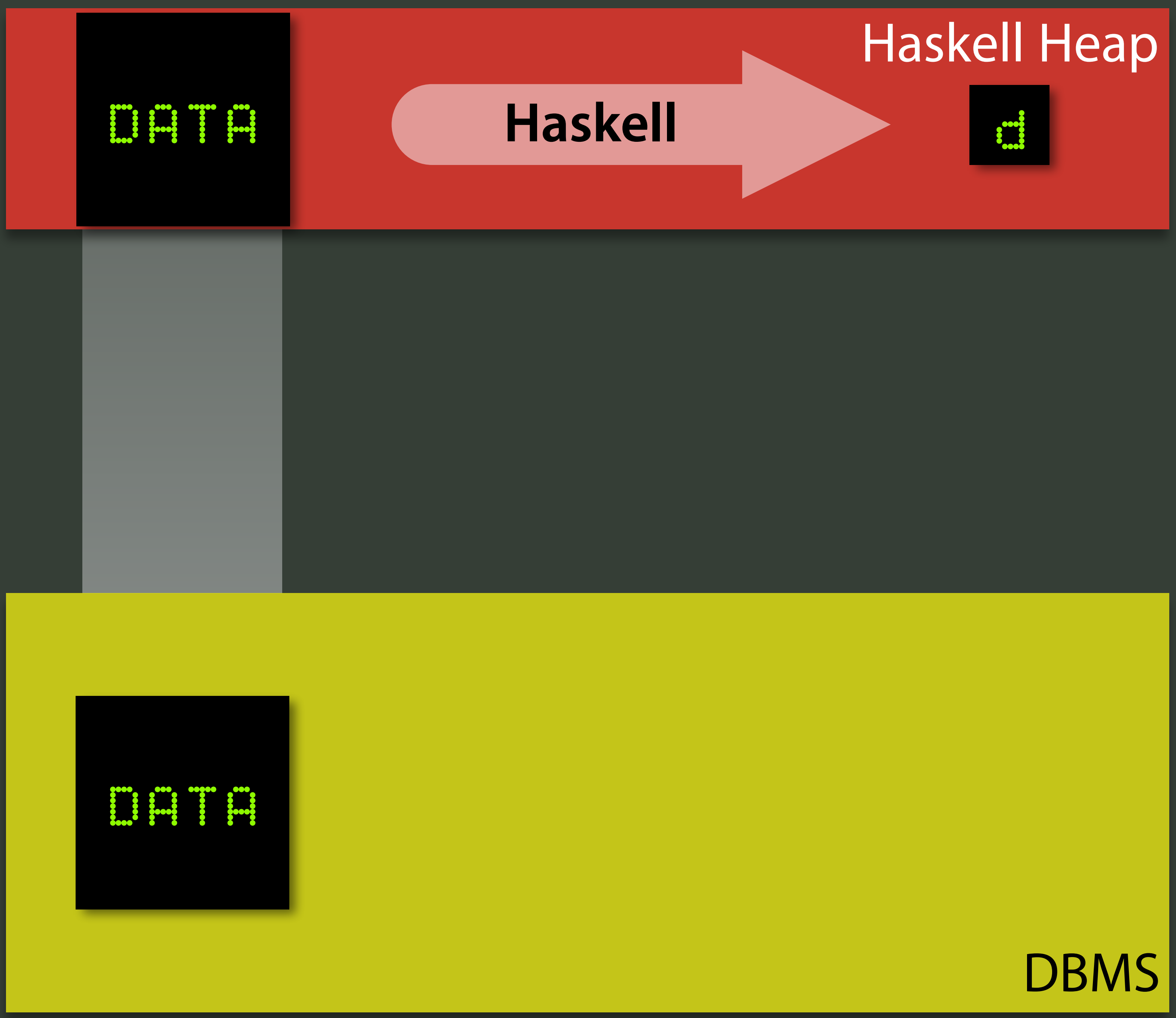
Not shown: set operations, joins, `WITH RECURSIVE`, …

Database-Supported Haskell

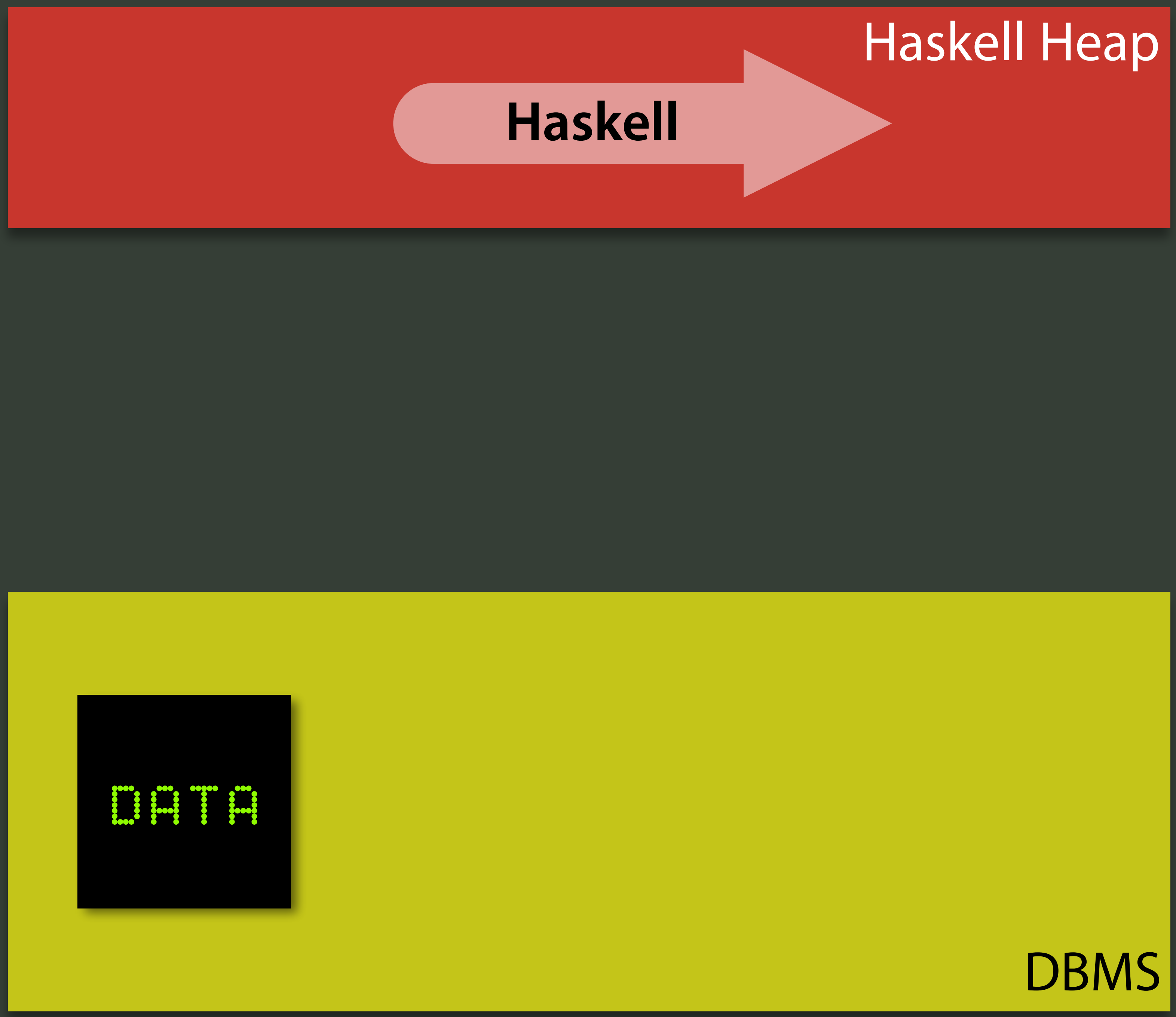
Database-Supported Haskell



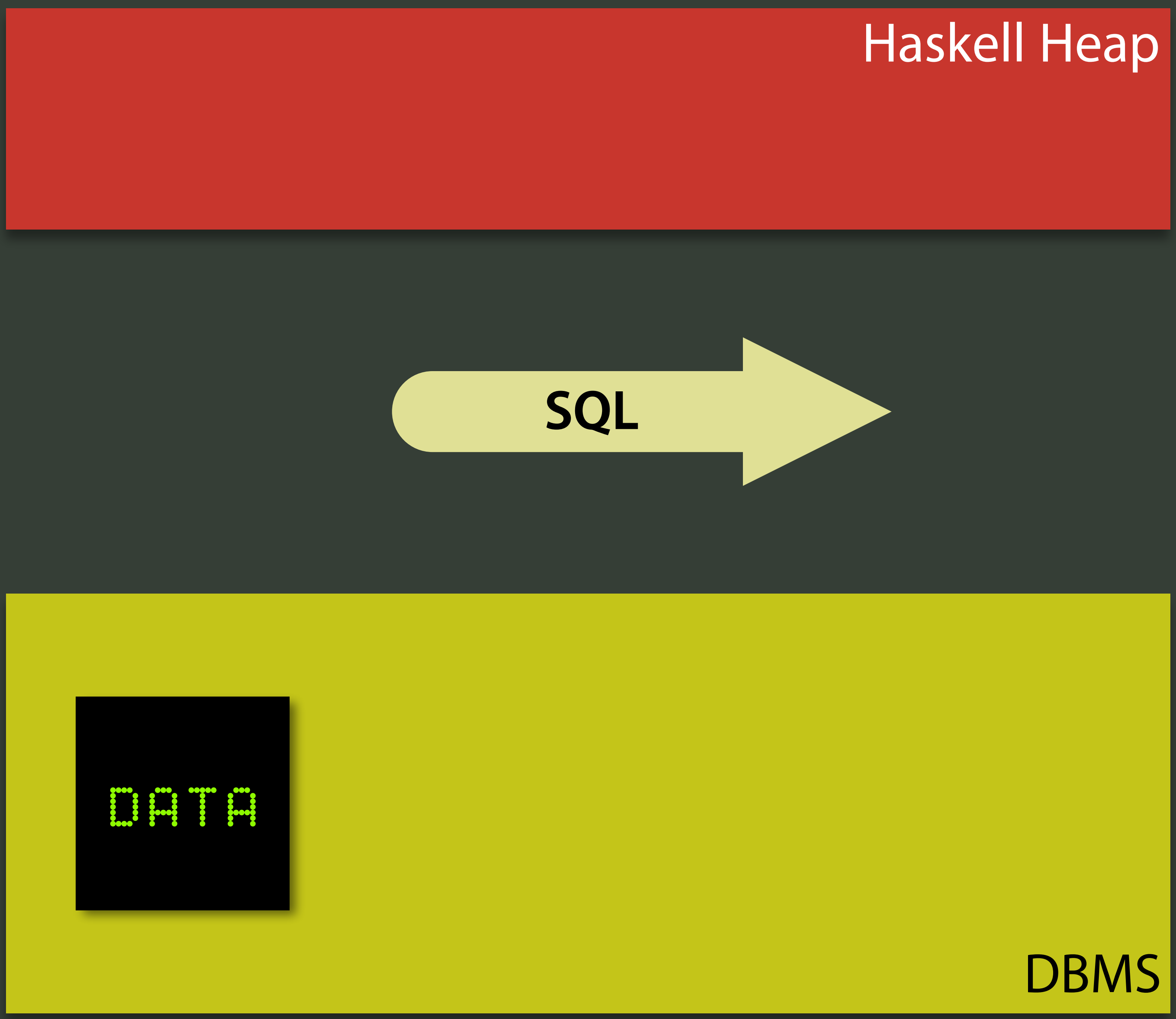
Database-Supported Haskell



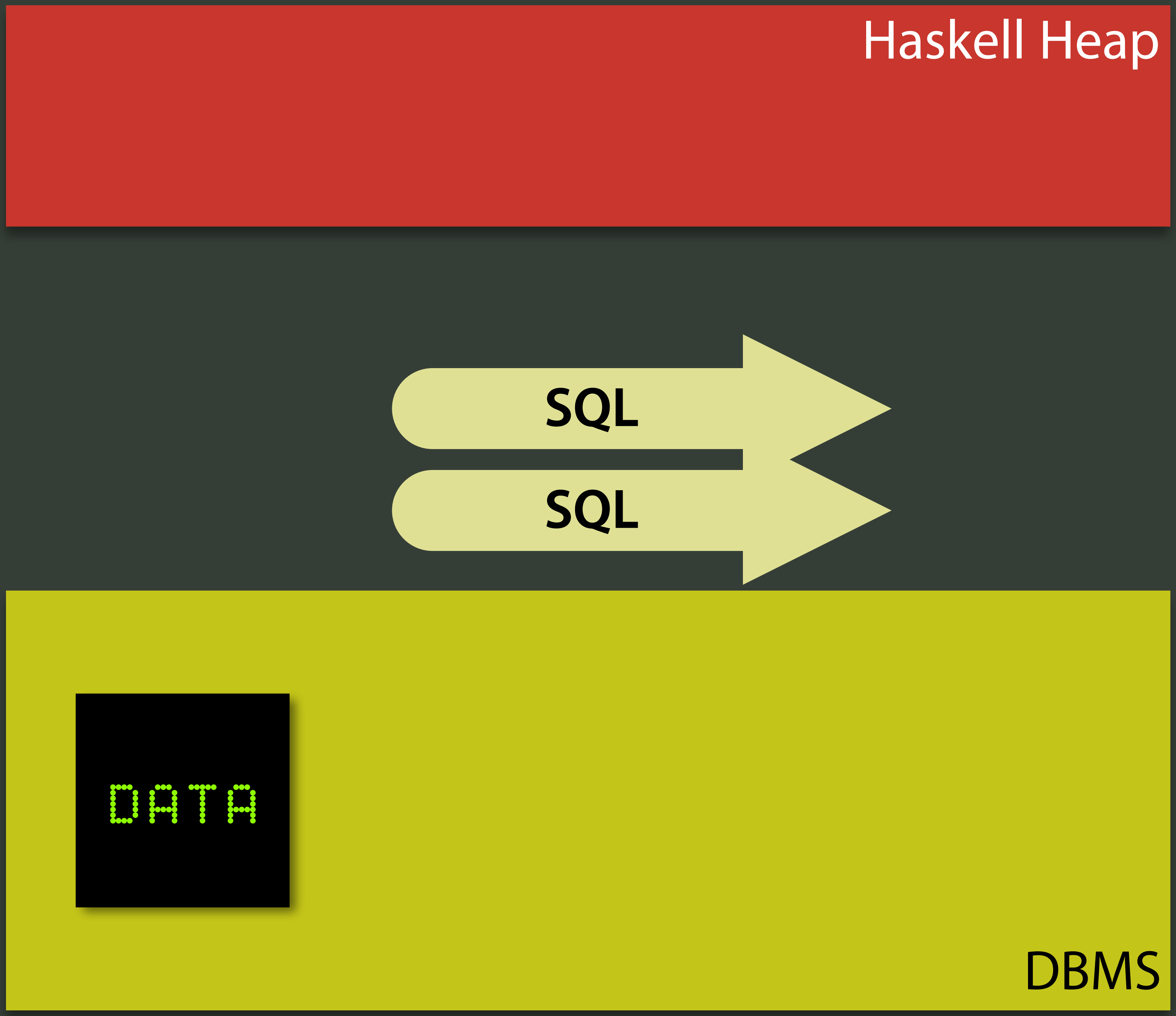
Database-Supported Haskell



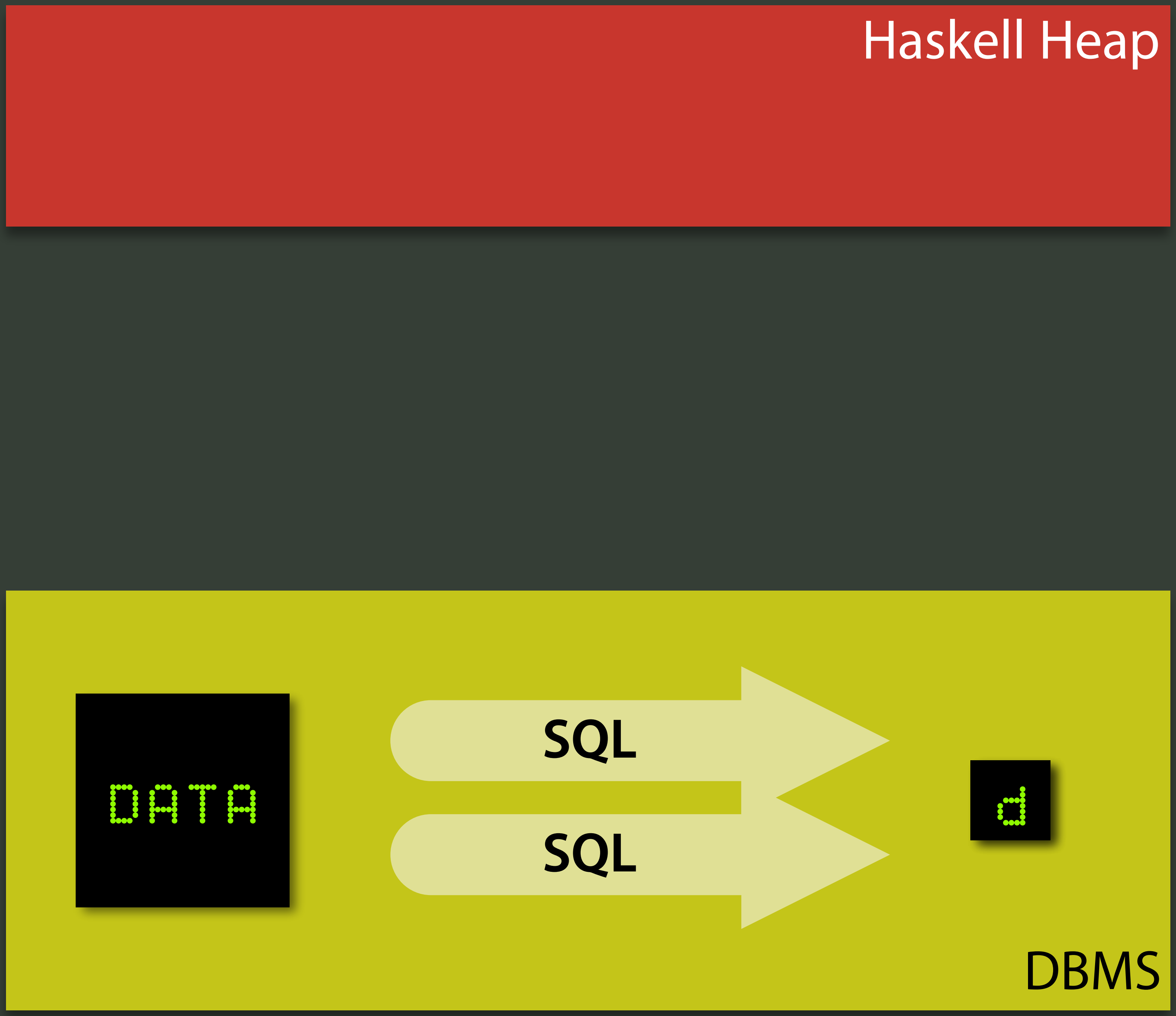
Database–Supported Haskell



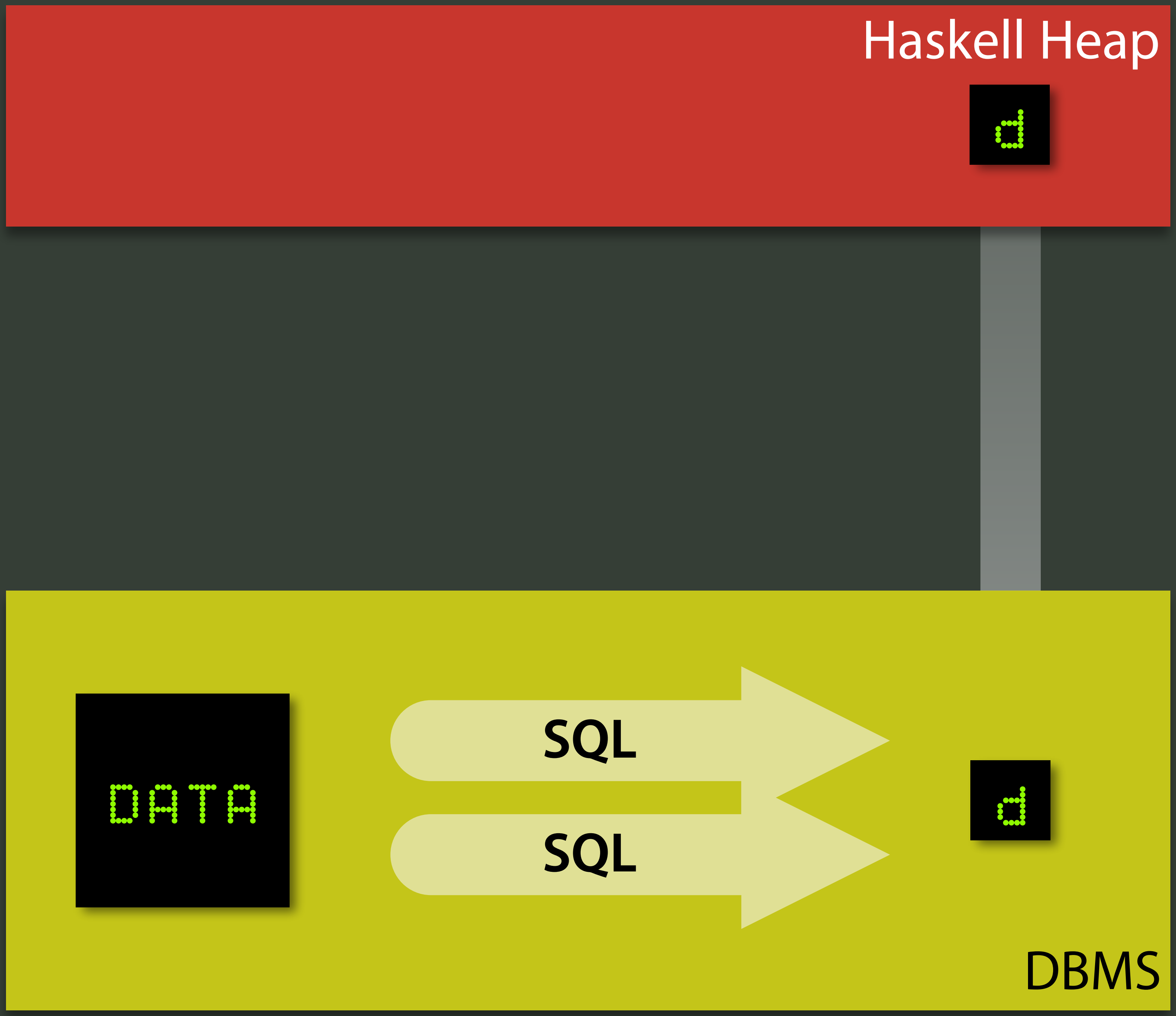
Database-Supported Haskell



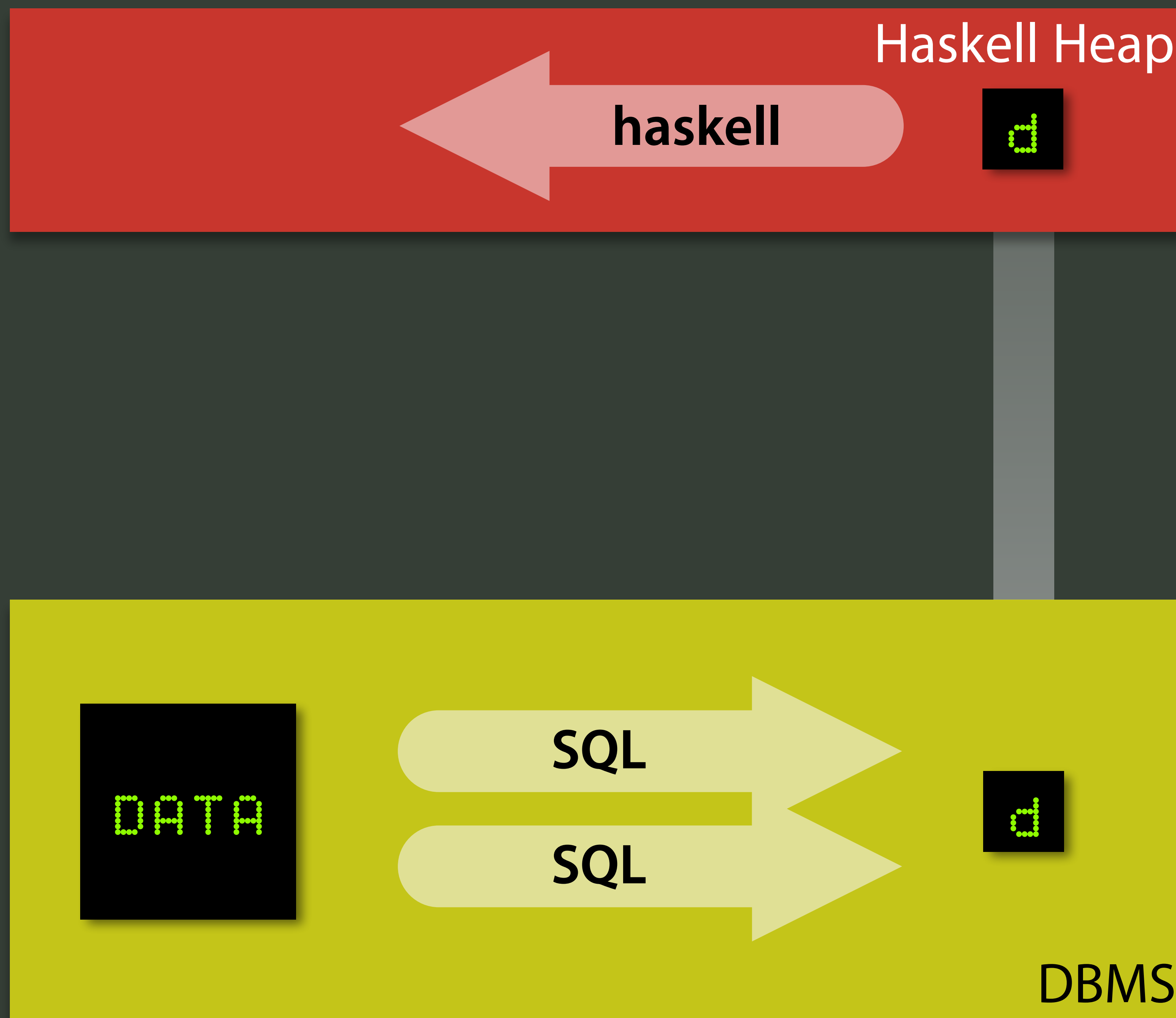
Database-Supported Haskell



Database-Supported Haskell



Database-Supported Haskell



Database-Supported Haskell

Database-Supported Haskell

```
-- rolling minimum (mins [3,4,1,7] = [3,3,1,1])
mins :: Ord a => Q [a] -> Q [a]
mins xs =
  [ minimum [ y | (y,j) <- #xs, j <= i ] | (_,i) <- #xs ]

-- margin: current value - minimum value up to now
margins :: (Ord a, Num a) => Q [a] -> Q [a]
margins xs = [ x - y | (x,y) <- zip xs (mins xs) ]

-- our profit is the maximum margin obtainable
profit :: (Ord a, Num a) => Q [a] -> Q [a]
profit xs = maximum (margins xs)

-- best profit obtainable for stock on given date
bestProfit :: Text -> Date -> Q [Trade] -> Q Double
bestProfit stock date trades =
  profit [ price t | t <- sortWith ts trades,
                     id t == stock, day t == date ]
```

Database-Supported Haskell

```
-- rolling minimum (mins [3,4,1,7] = [3,3,1,1])
mins :: Ord a => Q [a] -> Q [a]
mins xs =
  [ minimum [ y | (y,j) <- #xs, j <= i ] | (_,i) <- #xs ]

-- margin: current value - minimum value up to now
margins :: (Ord a, Num a) => Q [a] -> Q [a]
margins xs = [ x - y | (x,y) <- zip xs (mins xs) ]

-- our profit is the maximum margin obtainable
profit :: (Ord a, Num a) => Q [a] -> Q [a]
profit xs = maximum (margins xs)

-- best profit obtainable for stock on given
bestProfit :: Text -> Date -> Q [Trade] -> Q
bestProfit stock date trades =
  profit [ price t | t <- sortWith ts trades,
                     id t == stock, day t == date ]
```

Trades

id	ts	day	price
ACME	1	7/1/15	3.0
ACME	2	7/1/15	4.0
ACME	3	7/1/15	1.0
ACME	4	7/1/15	7.0
⋮	⋮	⋮	⋮

Database-Supported Haskell

```
-- rolling minimum (mins [3,4,1,7] = [3,3,1,1])
mins :: Ord a => Q [a] -> Q [a]
mins xs =
  [ minimum [ y | (y,j) <- #xs, j <= i ] | (_,i) <- #xs ]

-- margin: current value - minimum value up to now
margins :: (Ord a, Num a) => Q [a] -> Q [a]
margins xs = [ x - y | (x,y) <- zip xs (mins xs) ]

-- our profit is the maximum margin obtainable
profit :: (Ord a, Num a) => Q [a] -> Q [a]
profit xs = maximum (margins xs)

-- best profit obtainable for stock on given date
bestProfit :: Text -> Date -> Q [Trade] -> Q Double
bestProfit stock date trades =
  profit [ price t | t <- sortWith ts trades,
                     id t == stock, day t == date ]
```

Database-Supported Haskell

```
-- rolling minimum (mins [3,4,1,7] = [3,3,1,1])
mins :: Ord a => Q [a] -> Q [a]
mins xs =
  [ minimum [ y | (y,j) <- #xs, j <= i ] | (_,i) <- #xs ]

-- margin: current value - minimum value up to now
margins :: (Ord a, Num a) => Q [a] -> Q [a]
margins xs = [ x - y | (x,y) <- zip xs (mins xs) ]

-- our profit is the maximum margin obtainable
profit :: (Ord a, Num a) => Q [a] -> Q [a]
profit xs = maximum (margins xs)

-- best profit obtainable for stock on given date
bestProfit :: Text -> Date -> Q [Trade] -> Q Double
bestProfit stock date trades =
  profit [ price t | t <- sortWith ts trades,
                     id t == stock, day t == date ]
```

Database–Supported Haskell

```
-- SQL code generated from Haskell source
SELECT MAX(margins.price - margins.min)
FROM
  (SELECT t.price,
          MIN(t.price)
            OVER (ORDER BY t.ts ROW BETWEEN
                  UNBOUNDED PRECEDING
                  AND CURRENT ROW)
   FROM trades AS t
   WHERE t.id = 'ACME'
   AND   t.day = '07/01/2015'
  ) AS margins(price,min)
```

Comprehensions Yield Independent Work

Comprehensions Yield Independent Work

$$[\quad [\, f\, y \mid y \leftarrow g\, x \,] \mid x \leftarrow xs \,]$$

$$f :: a \rightarrow b$$

Comprehensions Yield Independent Work

$$[\, f\ [\ \ y \mid y \leftarrow g\, x\,]\ \mid\ x \leftarrow xs\,]$$

$$f :: a \rightarrow b$$

Comprehensions Yield Independent Work

$$[\, f^1\, [\quad y \mid y \leftarrow g\, x\,] \;\mid\; x \leftarrow xs\,]$$

$$f :: a \rightarrow b$$
$$f^1 :: [a] \rightarrow [b]$$

Comprehensions Yield Independent Work

$$f^2\,[\quad [\quad y \mid y \leftarrow g\;x\;] \;\mid\; x \leftarrow xs\;]$$

$$f :: a \rightarrow b$$
$$f^1 :: [a] \rightarrow [b]$$
$$f^2 :: [[a]] \rightarrow [[b]]$$

Comprehensions Yield Independent Work

$$f^2\,[\; g\,x \mid x \leftarrow xs \;]$$

$$f :: a \rightarrow b$$
$$f^1 :: [a] \rightarrow [b]$$
$$f^2 :: [[a]] \rightarrow [[b]]$$

Comprehensions Yield Independent Work

$$f^2\,(g^1\; xs)$$

$$\begin{aligned} f &:: a \to b \\ f^1 &:: [a] \to [b] \\ f^2 &:: [[a]] \to [[b]] \end{aligned}$$

Comprehensions Yield Independent Work

$$[\, f^n\, e \mid x \leftarrow xs \,] \rightsquigarrow f^{n+1}\, [\, e \mid x \leftarrow xs \,]$$

Nested Data Parallelism

$$[\, f^n\, e \mid x \leftarrow xs \,] \rightsquigarrow f^{n+1}\, [\, e \mid x \leftarrow xs \,]$$

Implementation of a Portable Nested Data-Parallel Language*

Guy E. Blelloch[1] Siddhartha Chatterjee[2]
Jonathan C. Hardwick Jay Sipelstein Marco Zagha

Implementation of a Portable Nested Data-Parallel Language
G. E. Blelloch et al., ACM PPoPP, May 1993

The Flatter, the Better

The Flatter, the Better

$xss +^2 yss$

The Flatter, the Better

xss $+^2$ *yss*

```
[[19, 0],[30],[11,10, 7]]
            +²
[[ 0, 4],[12],[13, 2, 3]]
```

The Flatter, the Better

xss $+^2$ *yss*

```
[[19, 0],[30],[11,10, 7]]
            +²
[[ 0, 4],[12],[13, 2, 3]]
```

The Flatter, the Better

xss $+^2$ *yss*

[19, 0 , 30 , 11, 10, 7]

$+^1$

[0, 4 , 12 , 13, 2, 3]

[ ] [ ] [ ]

forget

The Flatter, the Better

$xss +^2 yss$

[19, 4 , 42 , 24, 12, 10]

forget

[ ] [ ] [ ]

The Flatter, the Better

xss $+^2$ *yss*

[[19, 4], [42], [24, 12, 10]]

forget

imprint

The Flatter, the Better

$$f^n\,e \rightsquigarrow \mathtt{imprint}_{n\text{-}1}\,(f^1\,(\mathtt{forget}_{n\text{-}1}\,e))$$

```
[[19, 4],[42],[24,12,10]]
```

forget

imprint

The Flatter, the Better

$f^n\,e \rightsquigarrow \texttt{imprint}_{n\text{-}1}\,(f^1\,(\texttt{forget}_{n\text{-}1}\,e))$

[[19, 4], [42], [24, 12, 10]]

seg	pos
1	1
1	2
1	3

seg	pos	sum
1	1	19
1	2	4
2	3	42
3	4	24
3	5	12
3	6	10

The Flatter, the Better

$$f^n\, e \rightsquigarrow \mathtt{imprint}_{n\text{-}1}\,(f^1\,(\mathtt{forget}_{n\text{-}1}\; e))$$

[[19, 4], [42], [24, 12, 10]]

seg	pos
1	1
1	2
1	3

seg	pos	sum
1	1	19
1	2	4
2	3	42
3	4	24
3	5	12
3	6	10

The Flatter, the Better

$$f^n\, e \rightsquigarrow \mathtt{imprint}_{n\text{-}1}\,(f^1\,(\mathtt{forget}_{n\text{-}1}\; e))$$

[[19, 4], [42], [24, 12, 10]]

seg	pos
1	1
1	2
1	3

seg	pos	sum
1	1	19
1	2	4
2	3	42
3	4	24
3	5	12
3	6	10

The Flatter, the Better

$$f^n\ e \rightsquigarrow \mathtt{imprint}_{n\text{-}1}\,(f^1\,(\mathtt{forget}_{n\text{-}1}\ e))$$

[[19, 4], [42], [24, 12, 10]]

seg	pos
1	1
1	2
1	3

seg	pos	sum
1	1	19
1	2	4
2	3	42
3	4	24
3	5	12
3	6	10

Database Systems: Designed to Implement _[1]

Database Systems: Designed to Implement $\_^1$

$+^1$

Database Systems: Designed to Implement $\_^{1}$

$+^{1}$

seg	…	x	y
1		19	0
1		0	4
2		30	12
3		11	13
3		0	2
3		7	3

Database Systems: Designed to Implement _[1]

$+^1$

$\pi_{\text{sum}:\,x+y}\left(\begin{array}{c|c|c|c} \text{seg} & \cdots & x & y \\ \hline 1 & & 19 & 0 \\ 1 & & 0 & 4 \\ 2 & & 30 & 12 \\ 3 & & 11 & 13 \\ 3 & & 0 & 2 \\ 3 & & 7 & 3 \end{array}\right)$

Database Systems: Designed to Implement $\_^{1}$

$+^{1}$

$\pi_{\text{sum}:\,x+y}\left(\;\right)$

seg	…	x	y
1		19	0
1		0	4
2		30	12
3		11	13
3		0	2
3		7	3

$\ltimes_{p}^{1}$

Database Systems: Designed to Implement $\_^1$

$+^1$

$\pi_{sum:x+y}($

seg	…	x	y
1		19	0
1		0	4
2		30	12
3		11	13
3		0	2
3		7	3

$)$

$\ltimes_p^1$

seg_1	…	x
1		19
1		0
2		30
3		11
3		0
3		7

seg_2	…	y
1		0
1		4
2		12
3		13
3		2
3		3

Database Systems: Designed to Implement $\_^{1}$

$+^{1}$

$\pi_{\text{sum}:\,x+y}($

seg	…	x	y
1		19	0
1		0	4
2		30	12
3		11	13
3		0	2
3		7	3

$)$

$\ltimes_p^{1}$

seg_1	…	x
1		19
1		0
2		30
3		11
3		0
3		7

$\ltimes_p$

seg_2	…	y
1		0
1		4
2		12
3		13
3		2
3		3

Database Systems: Designed to Implement $\_^1$

$+^1$

$$\pi_{\mathrm{sum}:\,x+y}\Bigg($$

seg	…	x	y
1		19	0
1		0	4
2		30	12
3		11	13
3		0	2
3		7	3

$$\Bigg)$$

$\ltimes_p^1$

seg_1	…	x
1		19
1		0
2		30
3		11
3		0
3		7

$$\ltimes_{p \,\wedge\, seg_1 = seg_2}$$

seg_2	…	y
1		0
1		4
2		12
3		13
3		2
3		3

Database Systems: Designed to Implement $\_^1$

$+^1$

$\pi_{\mathrm{sum}:\,x+y}($

seg	…	x	y
1		19	0
1		0	4
2		30	12
3		11	13
3		0	2
3		7	3

$)$

$\ltimes_p^1$

seg₁	…	x
1		19
1		0
2		30
3		11
3		0
3		7

$\ltimes_{p \wedge \mathrm{seg}_1 = \mathrm{seg}_2}$

seg₂	…	y
1		0
1		4
2		12
3		13
3		2
3		3

Plan Bundles Instead of Query Avalanches

```
[(Int,[Str],[(Bool,[(Int,Int)])])]
```

Plan Bundles
Instead of Query Avalanches

```
[(Int,[Str],[(Bool,[(Int,Int)])])]
```

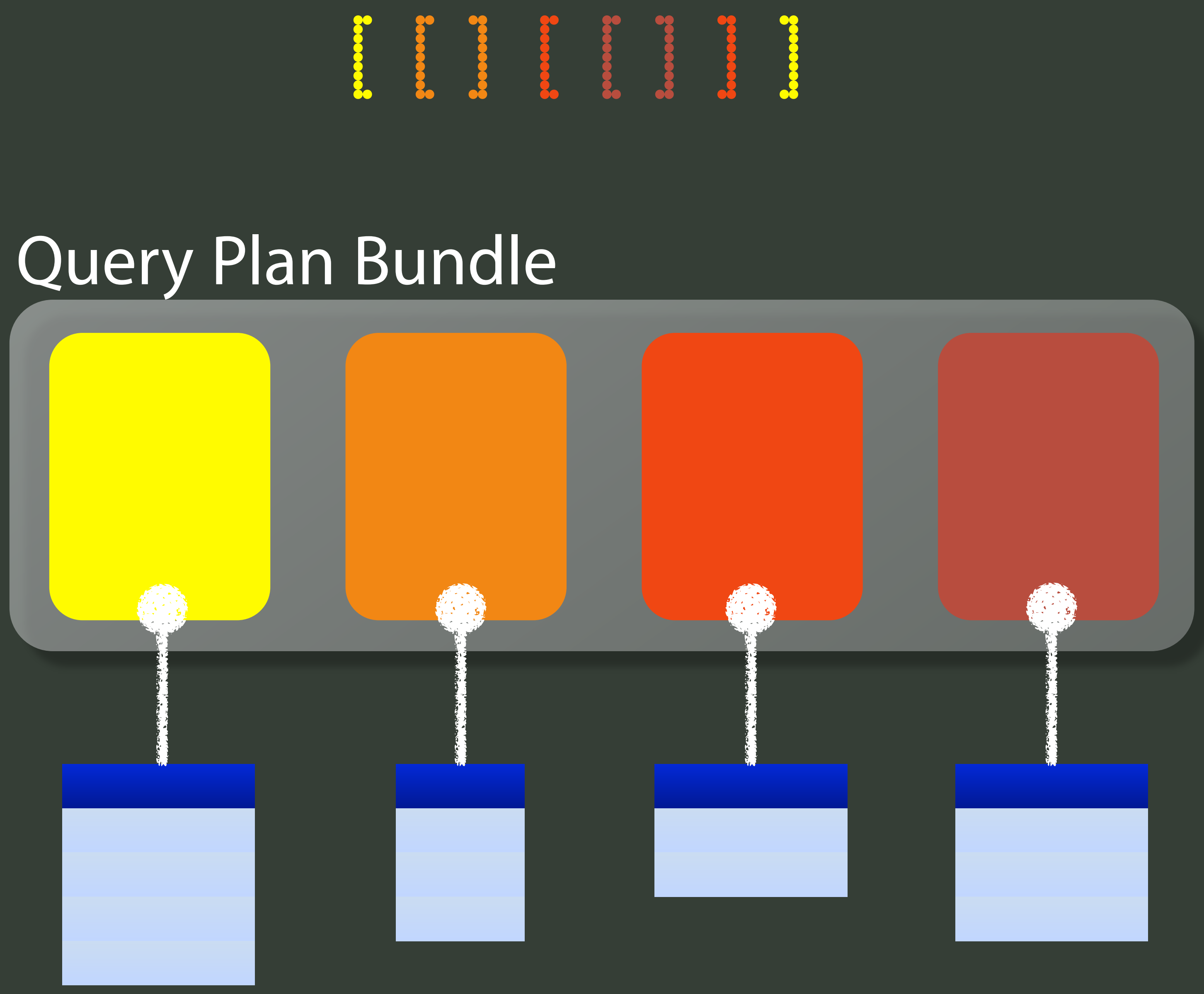
Plan Bundles
Instead of Query Avalanches

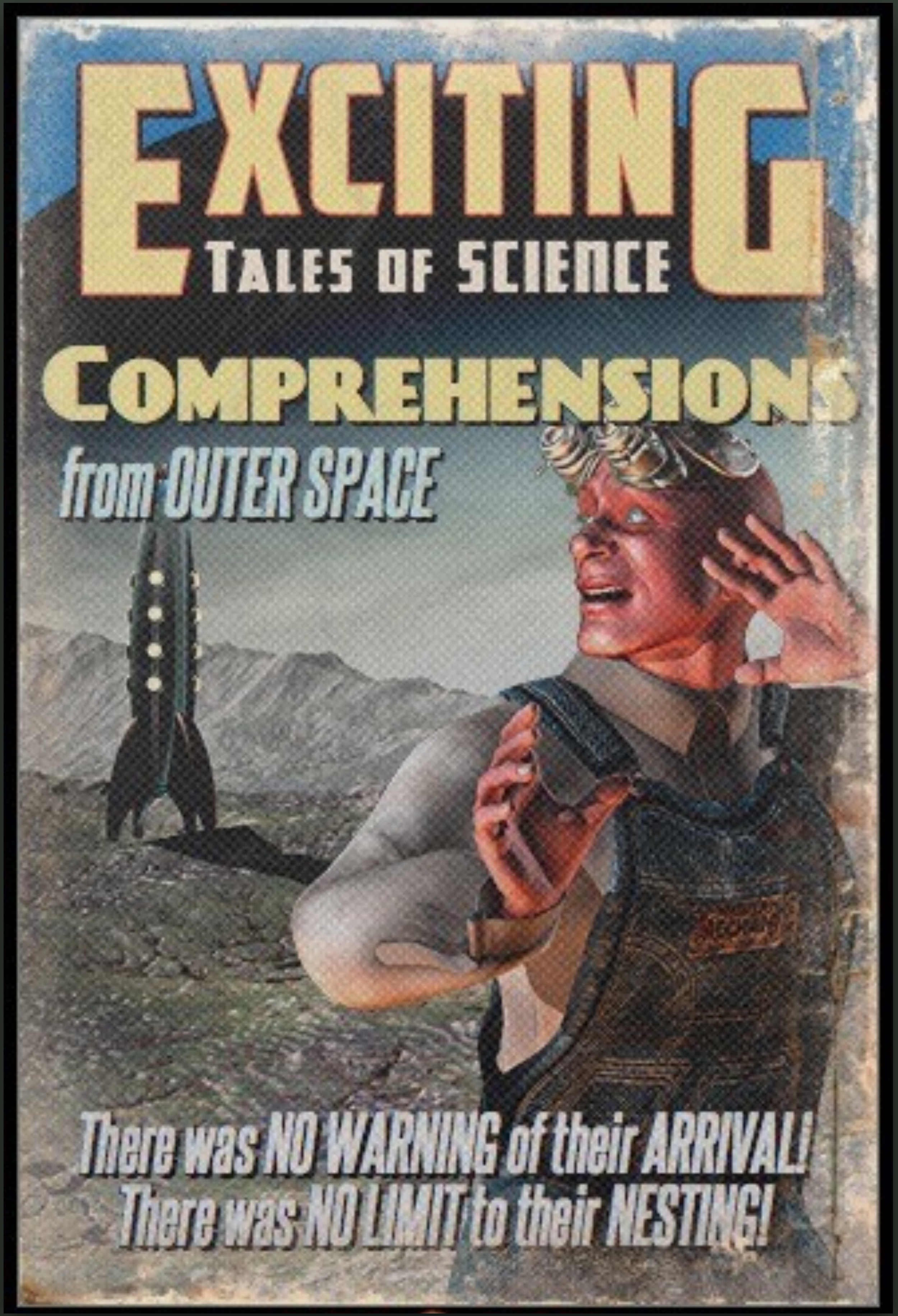
[[ ] [[ ]]]

Plan Bundles Instead of Query Avalanches

Query Plan Bundle

Plan Bundles Instead of Query Avalanches




EXCITING
TALES OF SCIENCE
COMPREHENSIONS
from OUTER SPACE
There was NO WARNING of their ARRIVAL!
There was NO LIMIT to their NESTING!