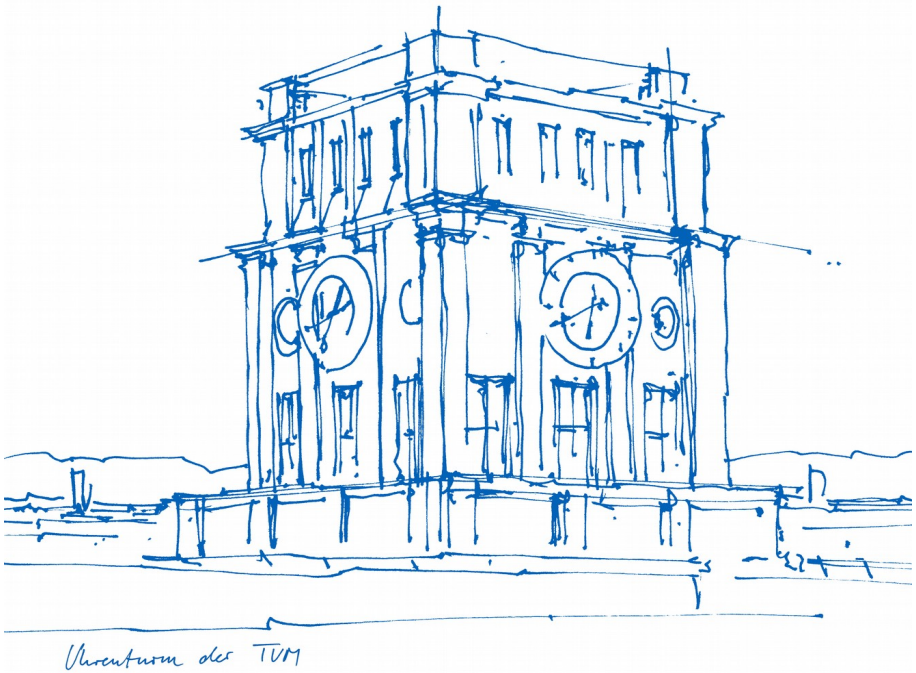
Lehrstuhl III: Datenbanksysteme
Lehrstuhl XXV: Data Science and Engineering
Fakultät für Informatik
Technische Universität München

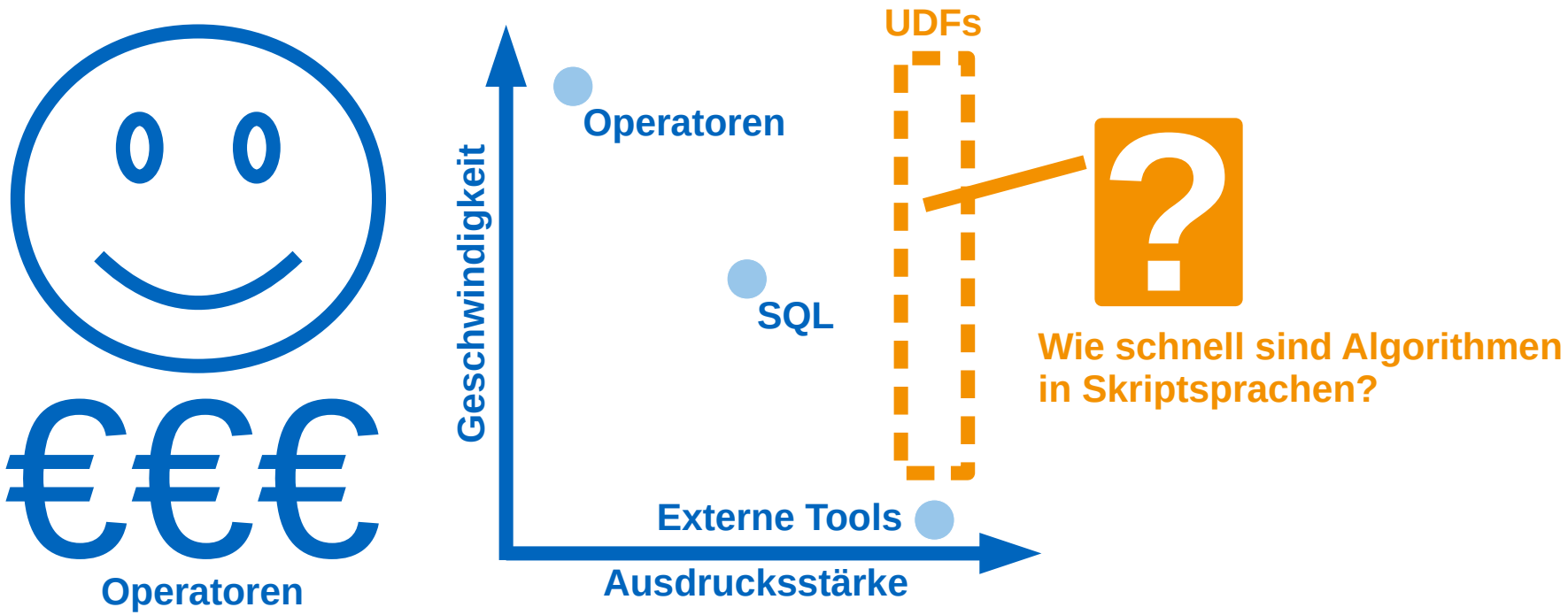
# Ja-(zu-)SQL: Evaluation einer SQL-Skriptsprache für Hauptspeicherdatenbanksysteme

Maximilian E. Schüle, Linnea Passing, Alfons Kemper, Thomas Neumann

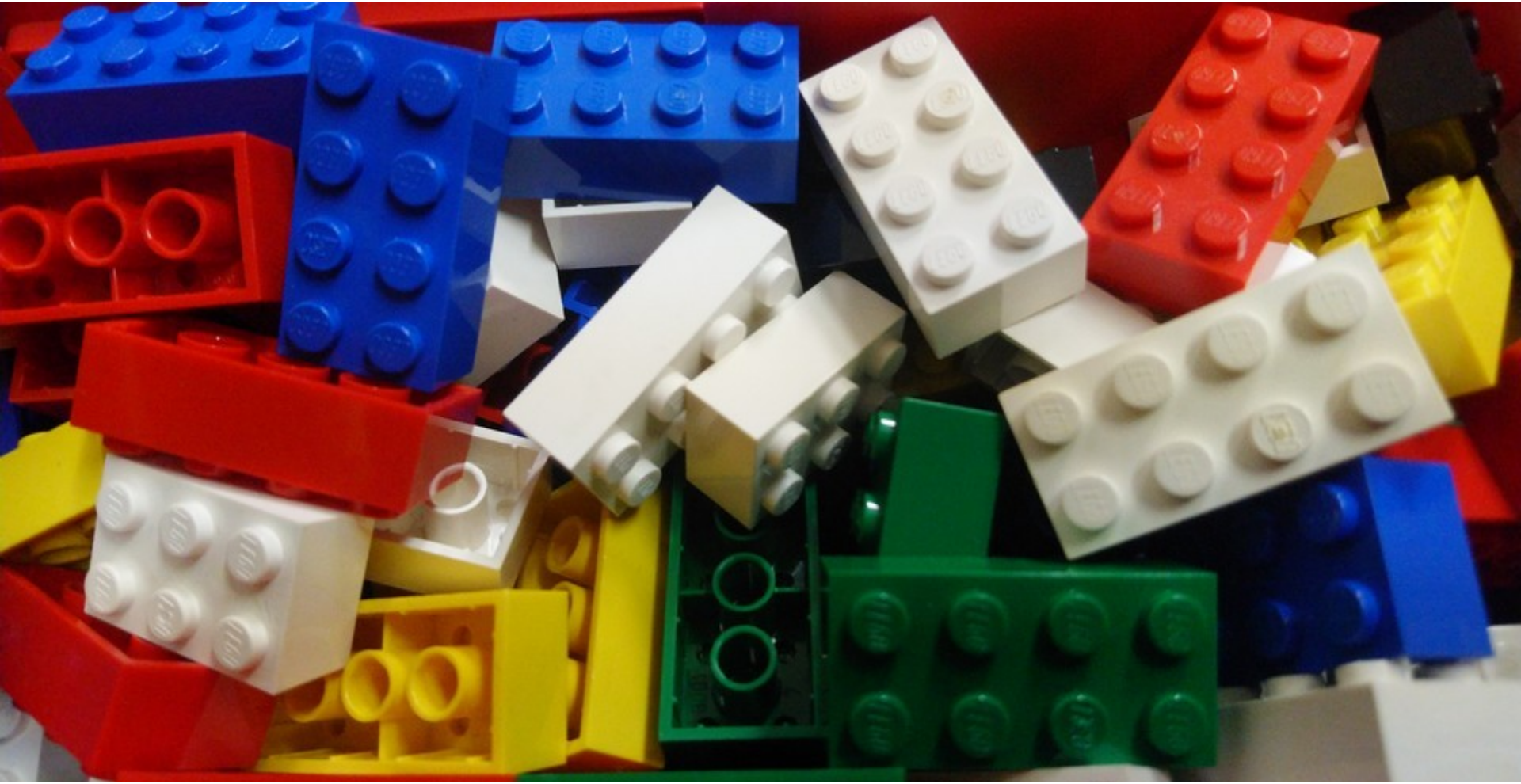
Rostock, 06. März 2019



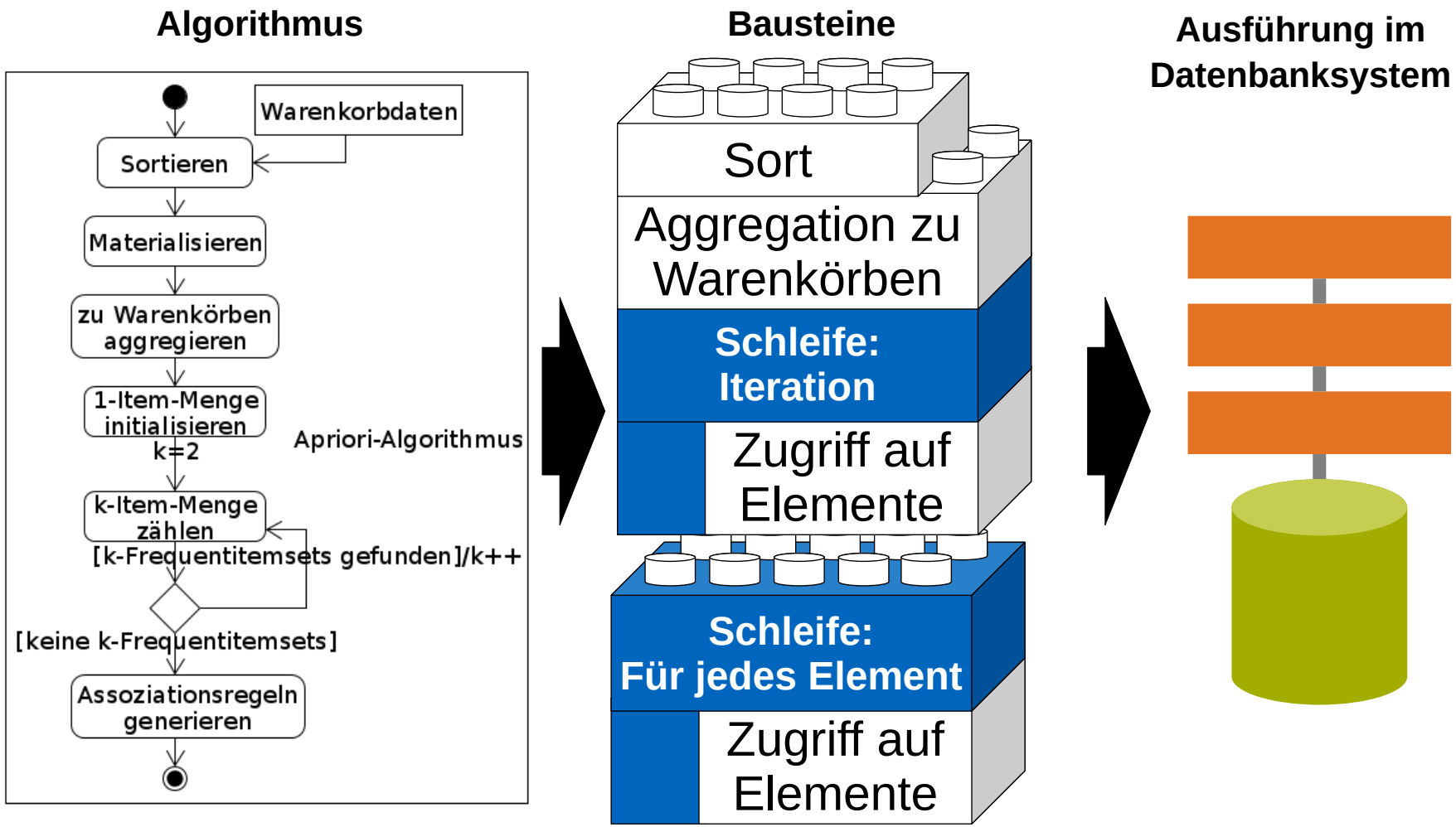
# Warum Skriptsprachen?



# Skriptsprachen: Für Bausteine der DBMS

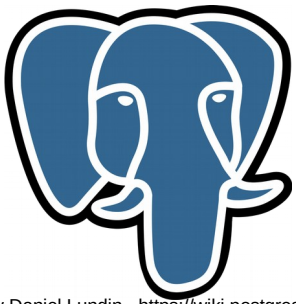


# Skriptsprachen: Für Bausteine der DBMS



# Skriptsprachen: in Datenbanksystemen

**Idee:** SQL + prozedurale Ausdrücke

By Daniel Lundin - https://wiki.postgresql.org/images/a/a4/PostgreSQL_logo.3colors.svg, PostgreSQL License


## PL/SQL

```
DECLARE
 type myarray IS
 VARRAY(1) OF FLOAT;
BEGIN
a:=myarray(1.1);
dbms_output.put_line(a(1));
END;
```

## PL/pgSQL

```
DO $$ declare
 A float[];
begin
 A :=array[1.1];
 RAISE NOTICE '%',A;
END$$;
```

## SQLScript

```
CREATE PROCEDURE
myproc LANGUAGE
SQLSCRIPT AS
BEGIN
 DECLARE a FLOAT
ARRAY = ARRAY(1.1);
END
```

## HyPerScript

```
CREATE OR REPLACE
FUNCTION ML_main() AS $$
  var A = array[1.1];
  debug_print( '%',A);
$$ LANGUAGE
'hyperscript' strict;
```

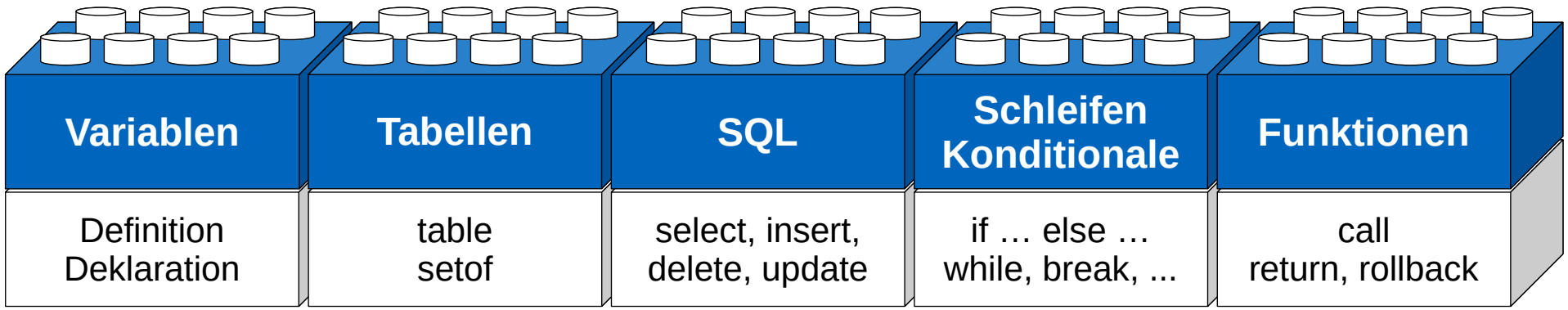
# Coden in HyPerScript

Argumente: Tabellen/ Datenbanktypen

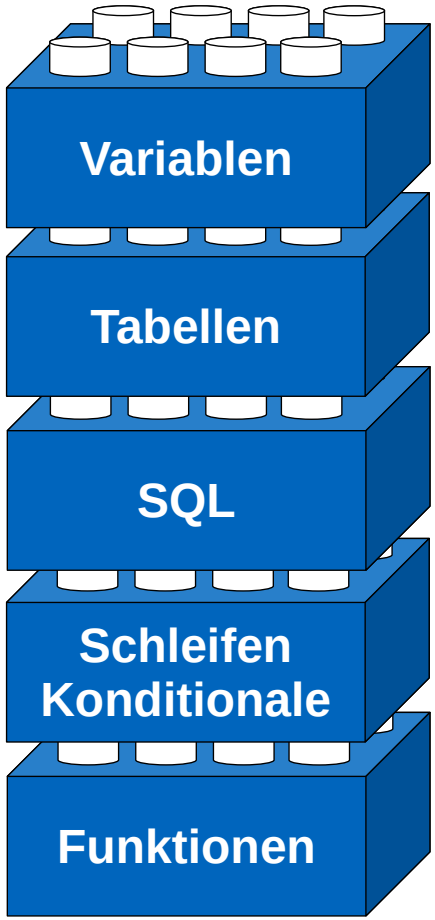
```
create or replace function insert_until(anzahl int not null)
as $$
   select index as i from sequence(0,anzahl){
      var rand = random(100);
      if (rand = 13) {
         insert into sample(i,NULL));
         continue;
      };
      insert into sample(i,rand));
   }
$$ language 'hyperscript' strict;
```

SQL Select-Anfrage mit Scope
`<Anfrage> { <stmt>}`

Tupelweiser Zugriff



# Coden in HyPerScript




```
Statement       = (( VarDeclaration  | VarDefinition  | TableDeclaration
                   | SelectStatement | EmbeddedSQL    | ReturnStatement
                   | IfStatement     | WhileStatement | ControlStatement
                   | FunctionCall )";")*;
VarDeclaration  = "var" NAME Type "=" Expression ;
VarDefinition   = NAME "=" Expression ;
TableDeclaration= "table" NAME "(" NAME Type ("," NAME Type)* ")";
SelectStatement = SQLSelect("{" Statement "}")("else" "{" Statement "}" );
EmbeddedSQL     = SQLInsert | SQLUpdate | SQLDelete | CSVCopy ;
IfStatement     = "if" "(" Expression ")" "{" Statement "}"
                  "else" "{" Statement "}" ;
ControlStatement= "break" | "continue" ;
WhileStatement  = "while" "(" Expression ")" "{" Statement "}" ;
FunctionCall    = NAME "(" Expression ("," Expression )* ")";
ReturnStatement = "return" NAME | "rollback" ;
```

# Leistungsbewertung mit HyPerScript

Bisher: HyPerScript für betriebswirtschaftlich transaktionale Anwendungen

Beispiel TPC-C: Handelsunternehmen mit eingehenden Aufträgen (New-Order)

Ziel: Messung der schreibenden Transaktionen pro Zeiteinheit

Bestellposition

Auftrag aus mehreren Bestellpositionen

```
create type newOrderPosition as (line_number int not null,… itemid int not null,...);
create function newOrder (..., positions setof newOrderPosition not null,...) as $$
   ...
   update stock
   set s_quantity=case when s_quantity>=qty+10
          then s_quantity-qty else s_quantity+91-qty end,
       s_remote_cnt=s_remote_cnt+case when supware<>w_id then 1 else 0 end,
       s_order_cnt=s_order_cnt+1, s_ytd=s_ytd+qty
   from positions
   where s_w_id=supware and s_i_id=itemid;
   ...
   if (inserted<cnt) rollback;
$$ language 'hyperscript' strict;
```

Verändern der Relationen

Tabellen als Argumente

Schema

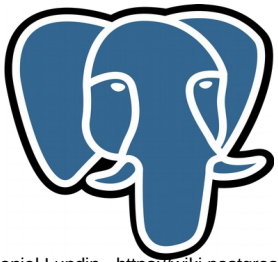
Verändern der Lagerbestände

Rücksetzen der Transaktion

# Data-Mining mit Skriptsprachen

# Vergleich R – PSQL – HyPer

By Hadley Wickham and others at RStudio - https://www.r-project.org/logo/, CC BY-SA 4.0



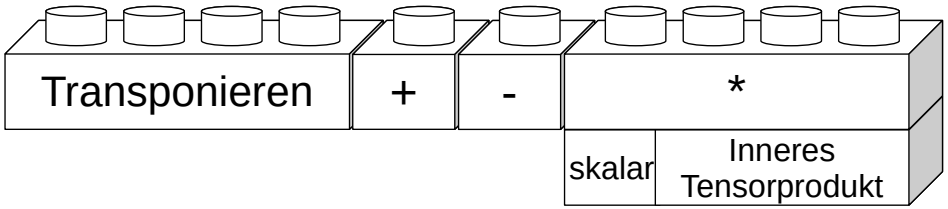
By Daniel Lundin - https://wiki.postgresql.org/images/a/a4/PostgreSQL_logo.3colors.svg, PostgreSQL License

| | **R** | **PostgreSQL** | **HyPer** |
|---|---|---|---|
| **Bausteine** | In C geschriebene, importierbare Pakete | MADlib | K-Means, DBSCAN, PageRank Operatoren (Passing et. al., EDBT 2017) |
| **Eigene Funktionen** | Nutzerseitig definierbare Funktionen | Gespeicherte Prozeduren in PL/pgSQL | Gespeicherte Prozeduren in HyPerScript |

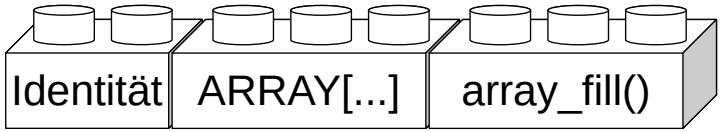
# Tensoroperationen für HyPerScript

**Erweiterungen des PSQL-Array-Datentyps**

Algebra (Operatoren überladen)



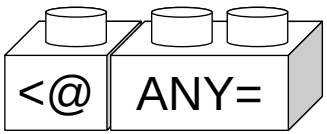

**Erzeugen**




**Zugriff** Schneiden (Slice) oder elementweise

**Verändern** array_set(), Konkatenieren

**Mengenoperationen**




**Binäre Exponentiation**

```
create or replace function pow(a_in float[]
[], e_in int) returns float[] as $$
 var a=a_in; var e=e_in;
 if( e<0 ) {
    e=e*-1;
    a=array_transpose(a);
 }
 var mask = 1<<63;
 var result=array_identity(array_ndims(a));
 while(mask>0){
    result=result*result;
    if( e&mask>0 ){
       result=result*a;
    }
    mask=mask>>1;
 }
 return result;
$$ language 'hyperscript' strict;
```

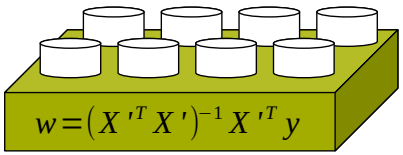
Transponieren

Identität

Multiplikation

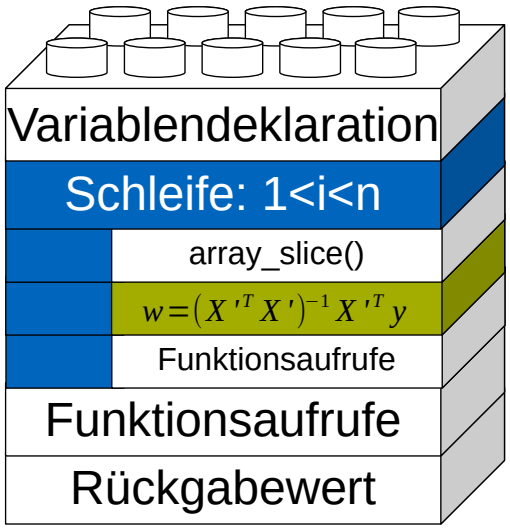
Für PostgreSQL verfügbar unter: https://gitlab.db.in.tum.de/ml2sql/psql-matrix-extension

# HyPerScript für Data-Mining




## Lineare Regression

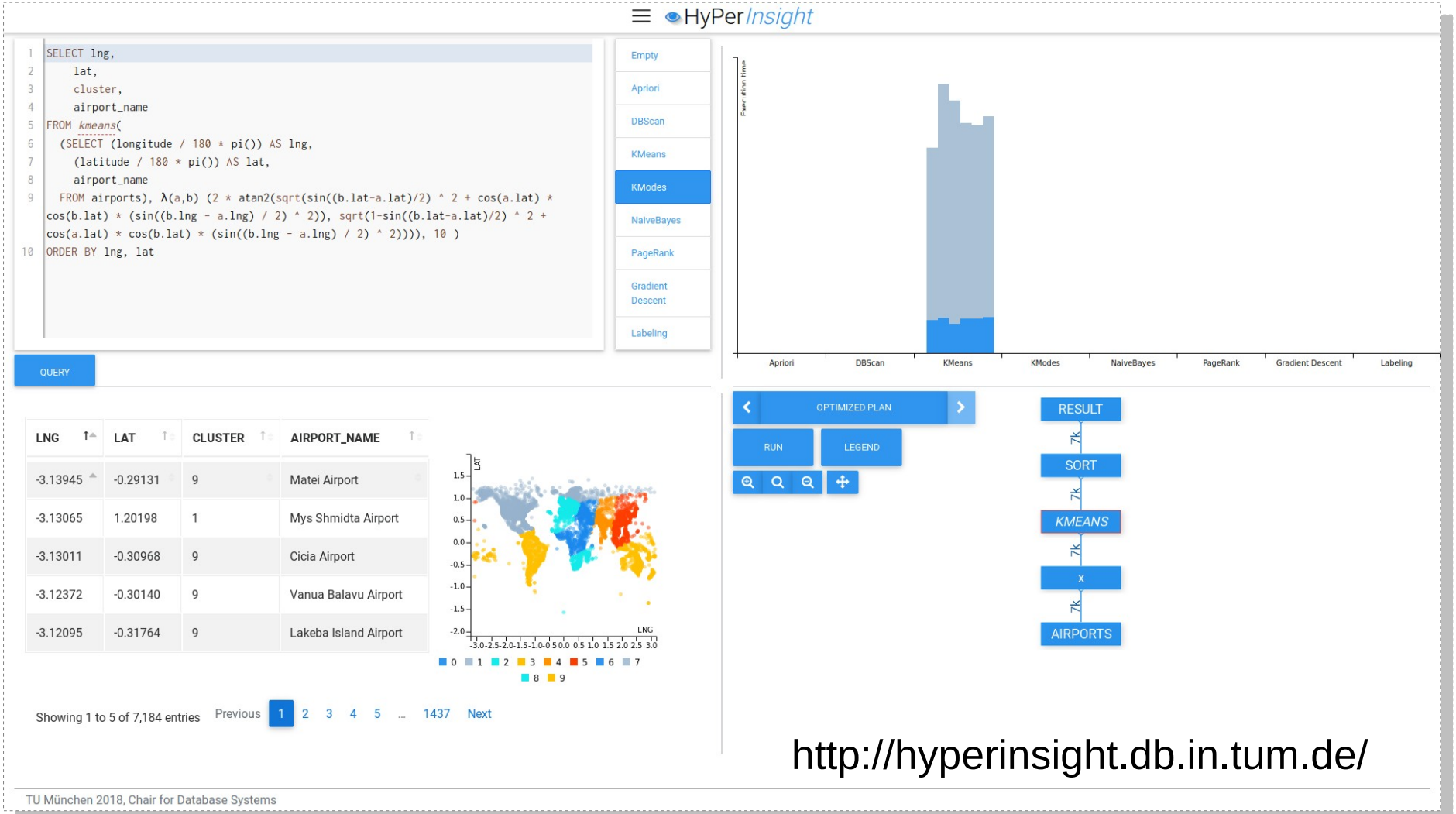
```
create or replace function linearregression(x float[][], y float[]
[]) returns float[] as $$
   return (array_transpose(x)*x)^-1*(array_transpose(x)*y);
$$ language 'hyperscript' strict;
```




## Einfache Kreuzvalidierung

```
create or replace function cross_validate(x float[][], y float[][])
returns float as $$
 var error=0;
 var n=array_length(x,2);
 var m=array_length(x,1);
 select index i from sequence(1,n){
  var weights_o=linearregression(
                  x[1:i-1][1:m]||x[i+1:n][1:m],
                  y[1:i-1][1:n]||y[i+1:n][1:1]);
  error=error+((array_slice(x,i,i,1,m)*weights_o)[1][1]-y[i][1])^2;
 }
 error=error/(n-2);
 return error;
$$ language 'hyperscript' strict;
```

# Implementierte Algorithmen




http://hyperinsight.db.in.tum.de/

# Implementierte Algorithmen

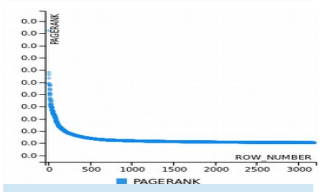
## Apriori

| ID | PRE | POST | SUPPORT | CONFIDENCE |
|---|---|---|---|---|
| 40 | {MUC} | {CDG} | 0.10219 | 0.72727 |
| 64 | {MUC} | {ZRH} | 0.10219 | 0.72727 |
| 66 | {MUC} | {ZRH,FRA} | 0.09854 | 0.70130 |
| 67 | {MUC,ZRH} | {FRA} | 0.09854 | 0.96429 |
| 68 | {MUC,FRA} | {ZRH} | 0.09854 | 0.80597 |

**Operator:** Häufige Itemmengen in Präfixbäumen

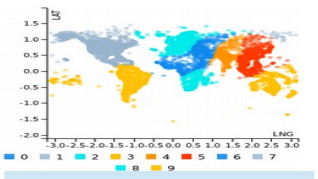
**HyPerScript:** Häufige Itemmengen als rekursive Tabelle

## PageRank



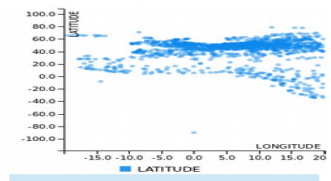

**Operator:** Indexierung der Knoten (Wörterbuch)
**HyPerScript:** SQL-Aggregation auf PageRank-Werte

## k-Means (Clustering)




**Operator:** nicht materialisierend, generiert LLVM-Code
**HyPerScript:** Fensterfunktionen ranken die Zentren

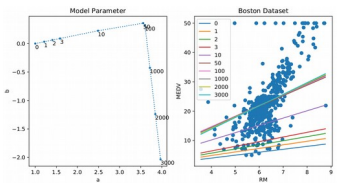
## DBSCAN (Clustering)




**Operator:** Clustersuche pro Punkt, generiert LLVM-Code

**HyPerScript:** Rekursive Erweiterung von Clustern

## Lineare Regression



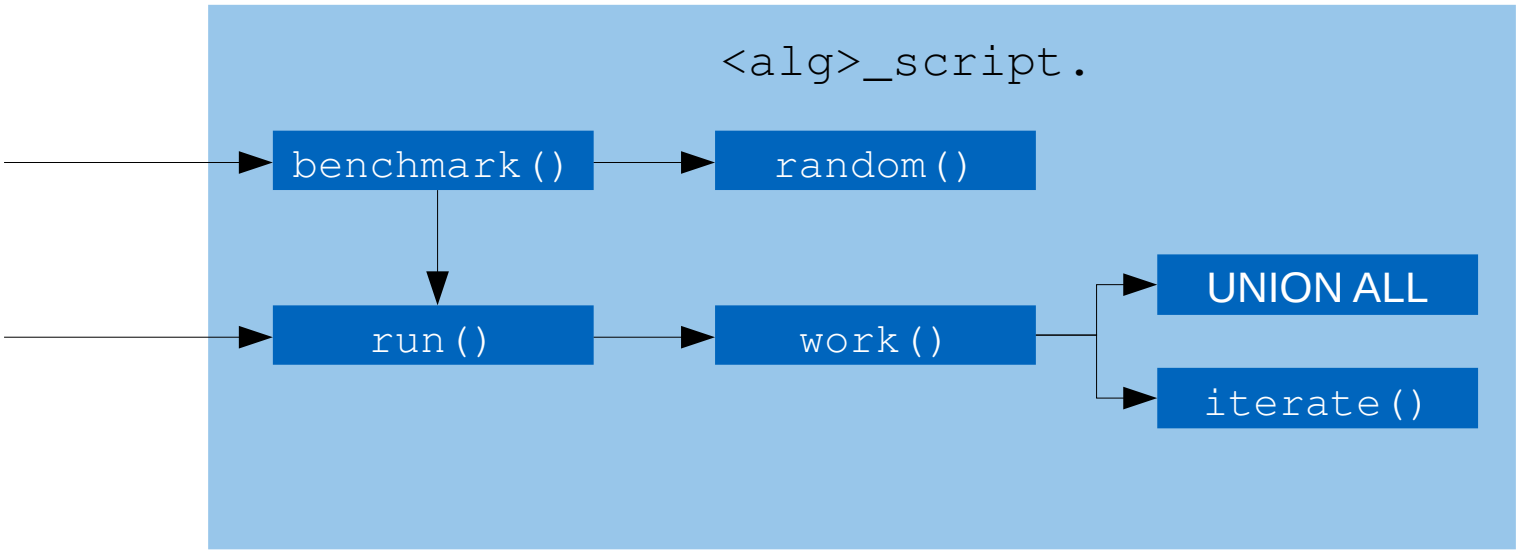

**Operator:** C++, stoch. Gradientenabstieg, parallel
**HyPerScript:** stoch. GA, gespeicherte Prozedur (strict)

# Schnittstelle der implementierten Algorithmen

Namensräume: strukturieren die Skripte voneinander

Jeder Namensraum: run() Prozedur für die Ausführung

Ergebnis als Rückgabetabelle



# Aufruf der Schnittstellen

Namensräume: strukturieren die Skripte voneinander

Jeder Namensraum: run() Prozedur für die Ausführung

Ergebnis als Rückgabetabelle

**Operatoren**

```
select * from kmeans(
  (select x,y from kmeansscript.points),
  (select x,y from kmeansscript.points LIMIT 5)
);
select * from dbscan(
    (select x,y from dbscanscript.points),20,2
);
```

k Cluster

Parameter

**HyPerScript-Funktionen**

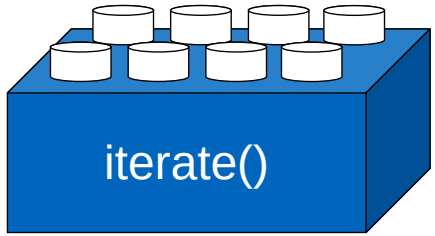
```
select kmeansscript.run(5);
select * from kmeansscript.pointsclusters;

select dbscanscript.run(20,2);
select * from dbscanscript.pointsclusters;
```

k Cluster

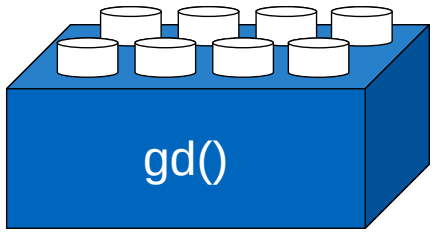
Parameter

Aufruf der Ergebnistabelle

# Lineare Regression mit Gradientenabstieg




```
create or replace function linregscript.iterate(tuples float[] not null,
y float not null, weights float[] not null, learningrate float not null)
returns float[] not null as $$
...
$$ language 'hyperscript' strict immutable;
```



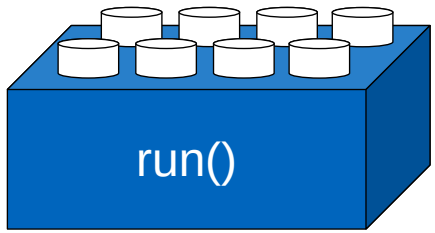

```
create type features as (a float, b float, c float);
create or replace function linregscript.gradientdescent(tuples setof
features, learningrate float not null, maxIter int not null)
returns float[] not null as $$
   var weights='{1,1,1}'::float[];
   select index as i from sequence(1,maxIter){
      select a,b,c from tuples{
         weights=linregscript.iterate(ARRAY[a,b],c,weights,learningrate);
      }
   }
   return weights;
$$ language 'hyperscript' strict stable;
```

Schema

Tabelle als Parameter

Schleife

Iteration




```
create or replace function linregscript.run(learningrate float not null,
maxIter int not null) returns float[] not null as $$
   table sample(a float, b float, c float);
   insert into sample (select x_1,x_2,y from linregscript.sample);
   var weights=linregscript.run(sample,learningrate,maxIter);
$$ language 'hyperscript' strict stable;
```

Lokale Relation

Keine Veränderung der Datenbasis

# Evaluation

# Evaluation

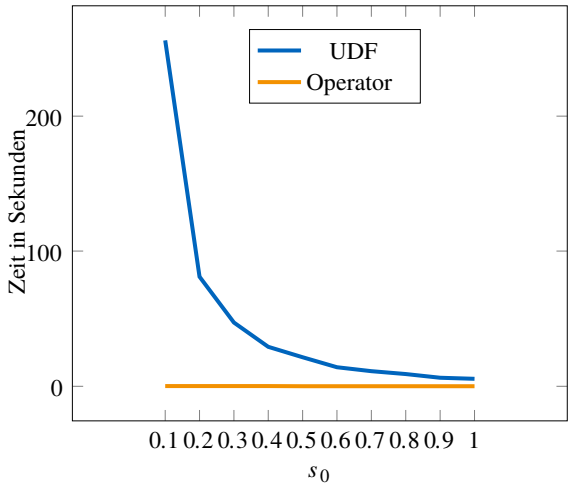
**Tools**
HyPer-Operatoren
HyPerScript
pl/pgSQL

**Maschine**
Intel Xeon E5-2660 v2 CPU (20x 2.20 GHz)
256 GB DDR4 RAM

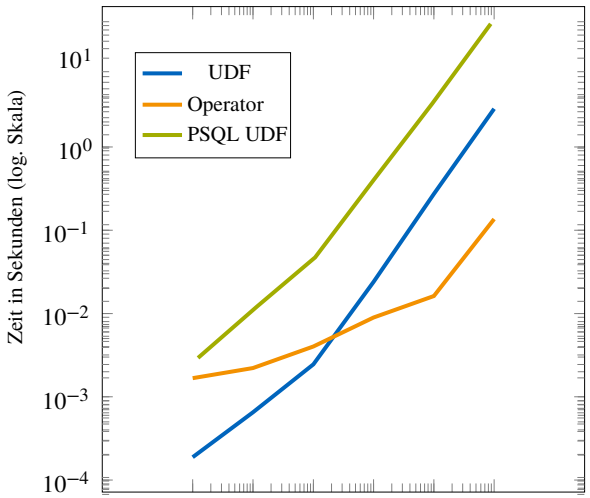
**Generierte Datensätze**
Apriori: 100 Waren, 1000 Körbe
PageRank: 10^5 Knoten/Kanten
Lineare Regression: 10^6 Tuple/3 Attribute
Clustering: x-/y-Koordinaten, 10^6 Punkte

**Messungen**
3 Wiederholungen, Median-Laufzeit

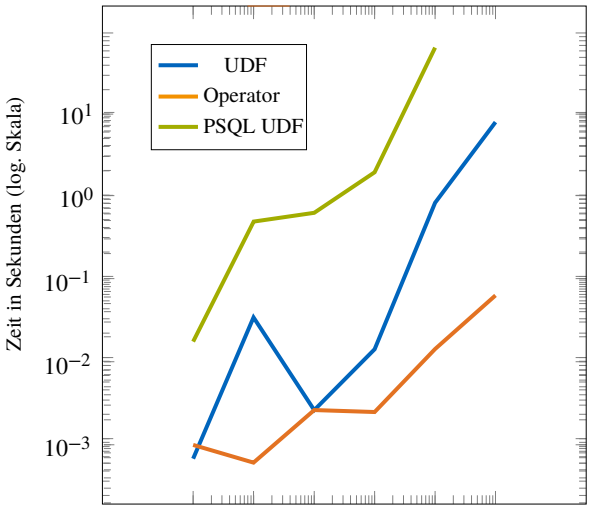
# Evaluation – Apriori, Lin. Reg., PageRank



Apriori.



Lineare Regression.



PageRank.

**Messungen**

Apriori: Veränderung des Supports

Lineare Regression/PageRank: Anzahl der Tupel

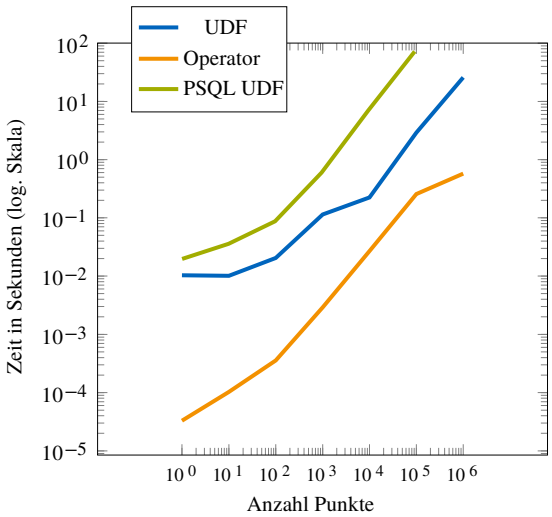
**Bei wenigen Tupeln:** UDF vergleichbare Laufzeit mit Operatoren

Viele Bestandteile von HyPerScript nicht parallelisiert

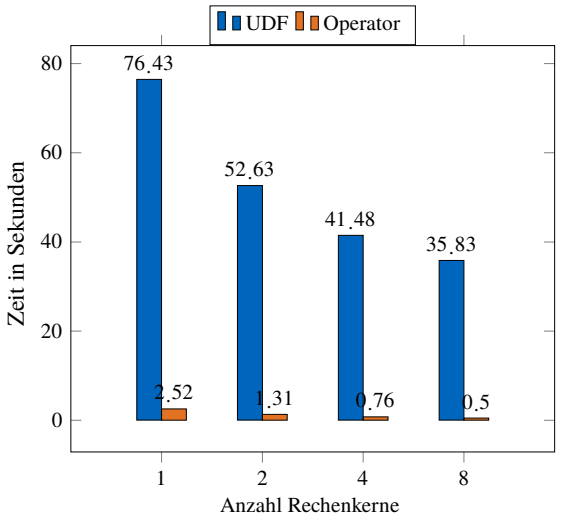
**HyPerScript schneller als pl/pgSQL**

HyPer Operatoren bereits sehr schnell

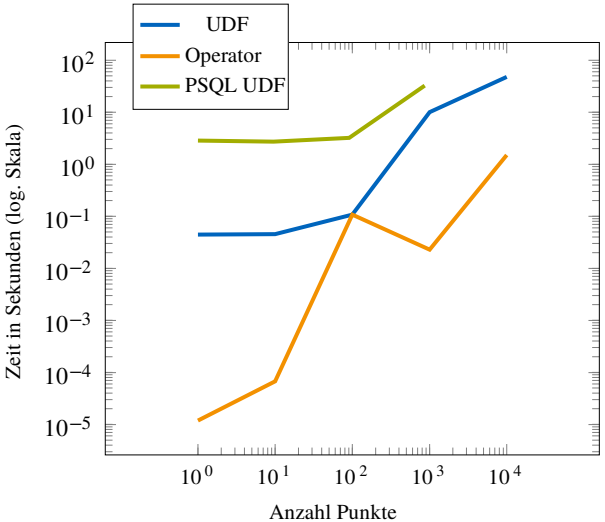
# Evaluation – Clustering-Algorithmen



k-Means: Eingabegröße.



k-Means: Skalierung.



DBSCAN: Eingabegröße.

**Messungen**

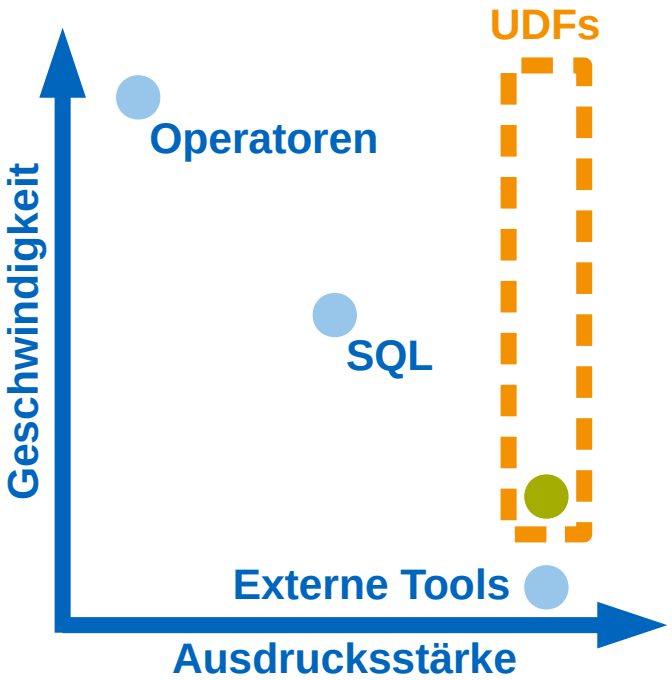
k-Means: Anzahl der Tupel, Anzahl der Kerne

DBSCAN: Anzahl der Tupel

**SQL-Anfragen bei k-Means skalieren**

Dennoch: native Operatoren schneller als UDF mit HyPerScript

# Fazit




**HyPerScript**

SQL mit Scope, Variablen, prozeduralen Ausdrücken und Tabellen

**HyPerScript für Data-Mining**

Prototypen leicht implementierbar, SQL-Teile gut skalierbar

**Ausblick**

Datenbanksysteme erlauben komplexere Berechnungen

```
create or replace function servus() as $$
   debug_print(‘Vielen Dank für die Aufmerksamkeit‘);
   debug_print(‘Fragen?‘);
$$ language 'hyperscript' strict;

> select servus();
```