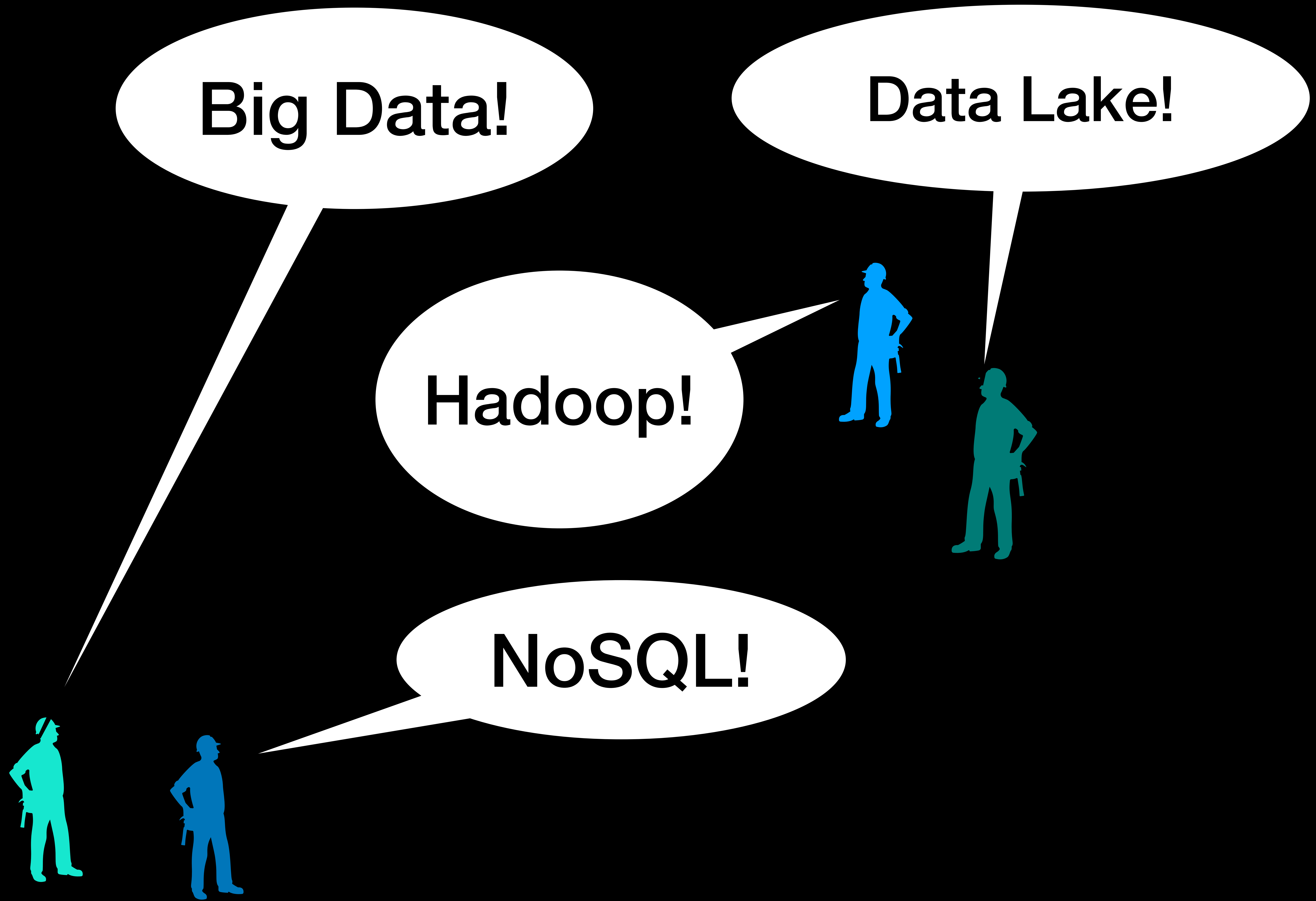
Big Data Systems Performance: The Little Shop of Horrors

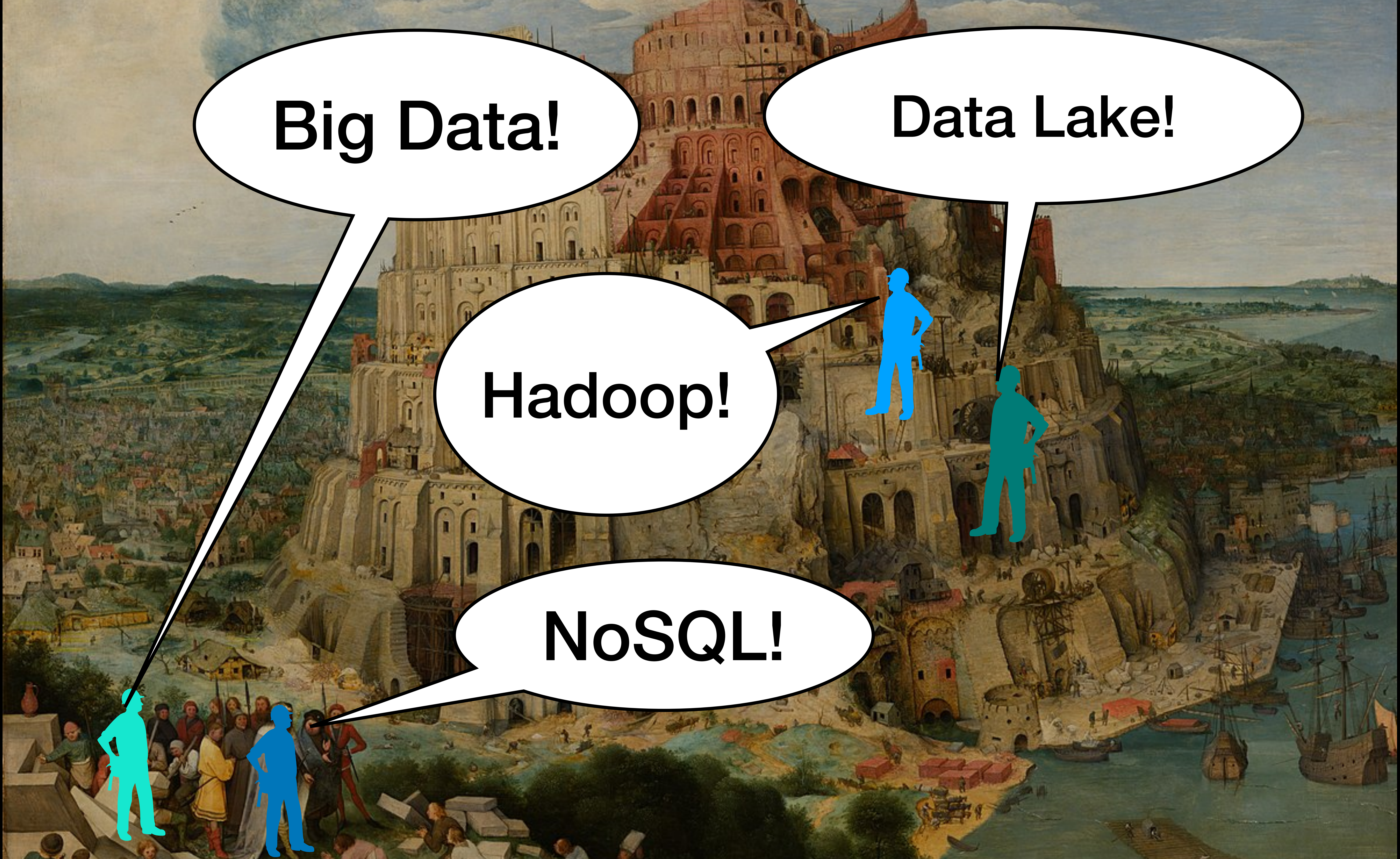
Jens Dittrich
Saarland Informatics Campus
d:AI:mond.ai
twitter.com/jensdittrich

Note: presentation = slides ⋈ manuscript (see end of this pdf)

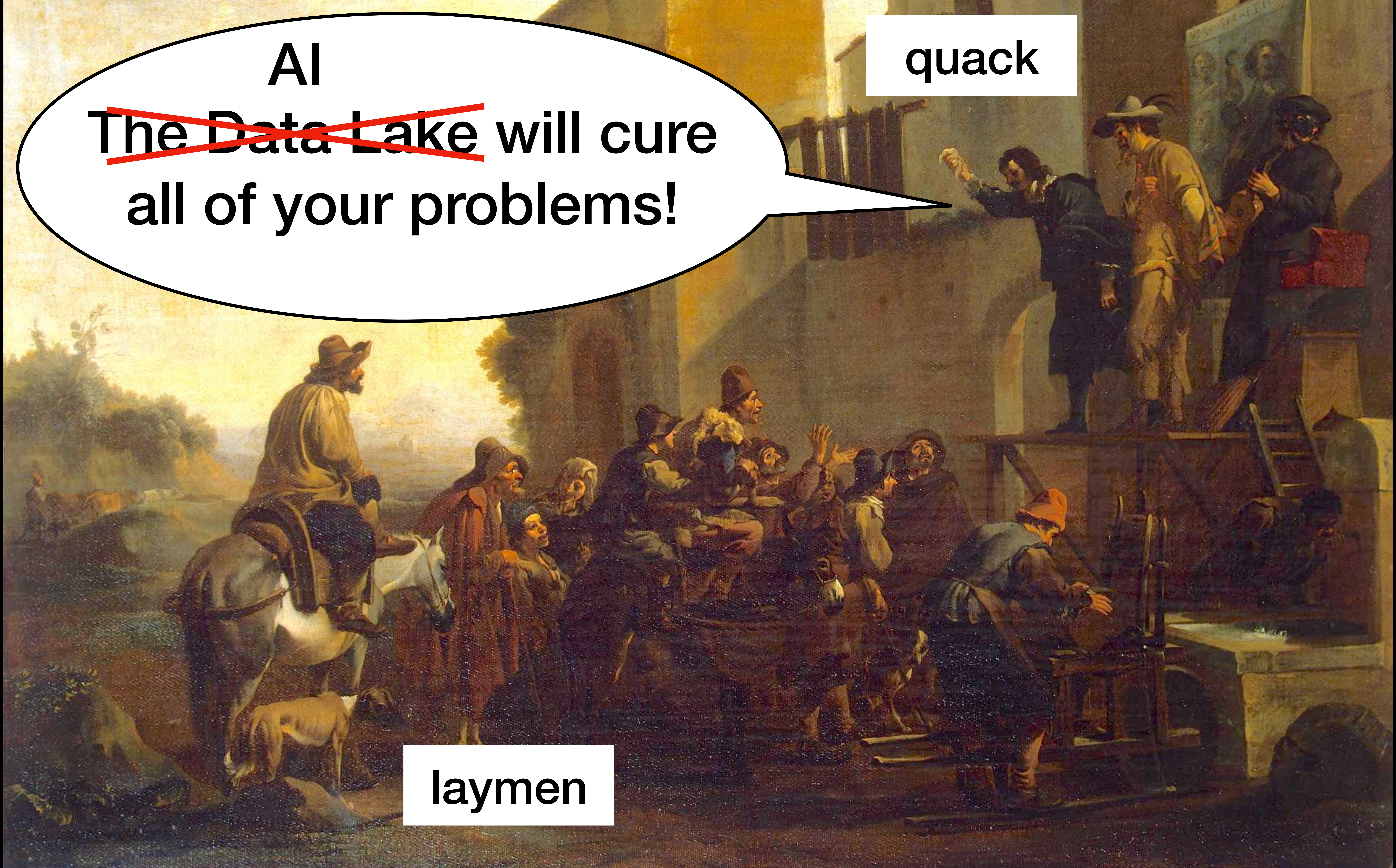
CV


Developer at SAP BW/Trex,
Associate Professor,
Saarland University
Ph.D. in Databases
Postdoc at ETH Zurich
Full Professor,
Saarland University
2000
2005
2010
2015
2017
Data Warehousing
Column Stores
Join processing
Dataspaces
Streams
Join processing
large-scale data management
Hadoop&Cloud
main-memory databases
Database Architectures
main-memory databases
Indexing
Systems
Crazy Stuff
Applied ML


Big Data!
Data Lake!
Hadoop!
NoSQL!


Big Data!
Data Lake!
Hadoop!
NoSQL!


quack
The Data Lake will cure all of your problems!
laymen


AI
~~The Data Lake~~ will cure
all of your problems!
quack
laymen

The TV shop


This model even has HyperRay++Rendering!
“TV expert“
laywoman


"TV expert"
500Hz rather than 250Hz
laywoman

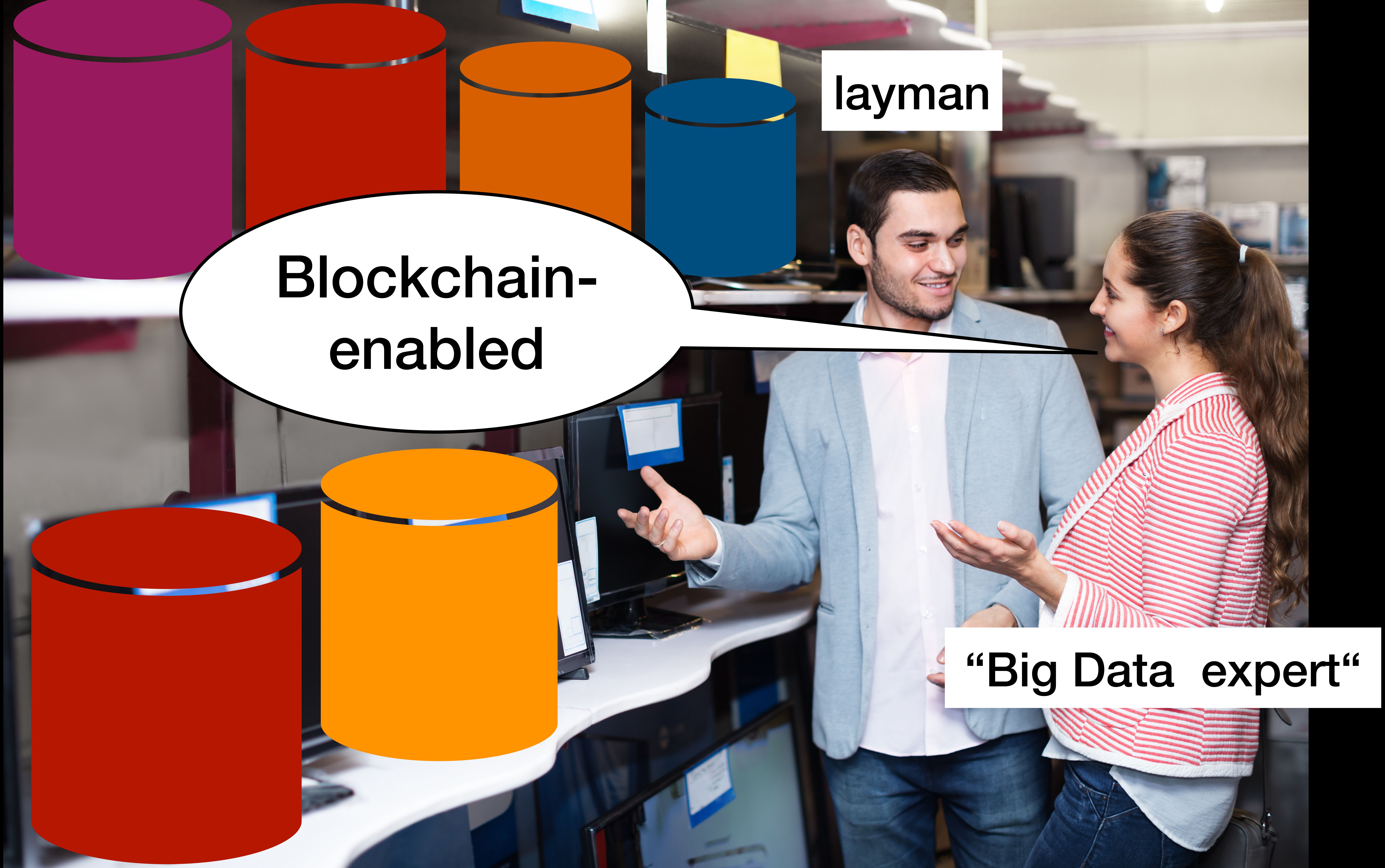

UltraHDMI!
“TV expert“
laywoman

The “Big Data“ shop

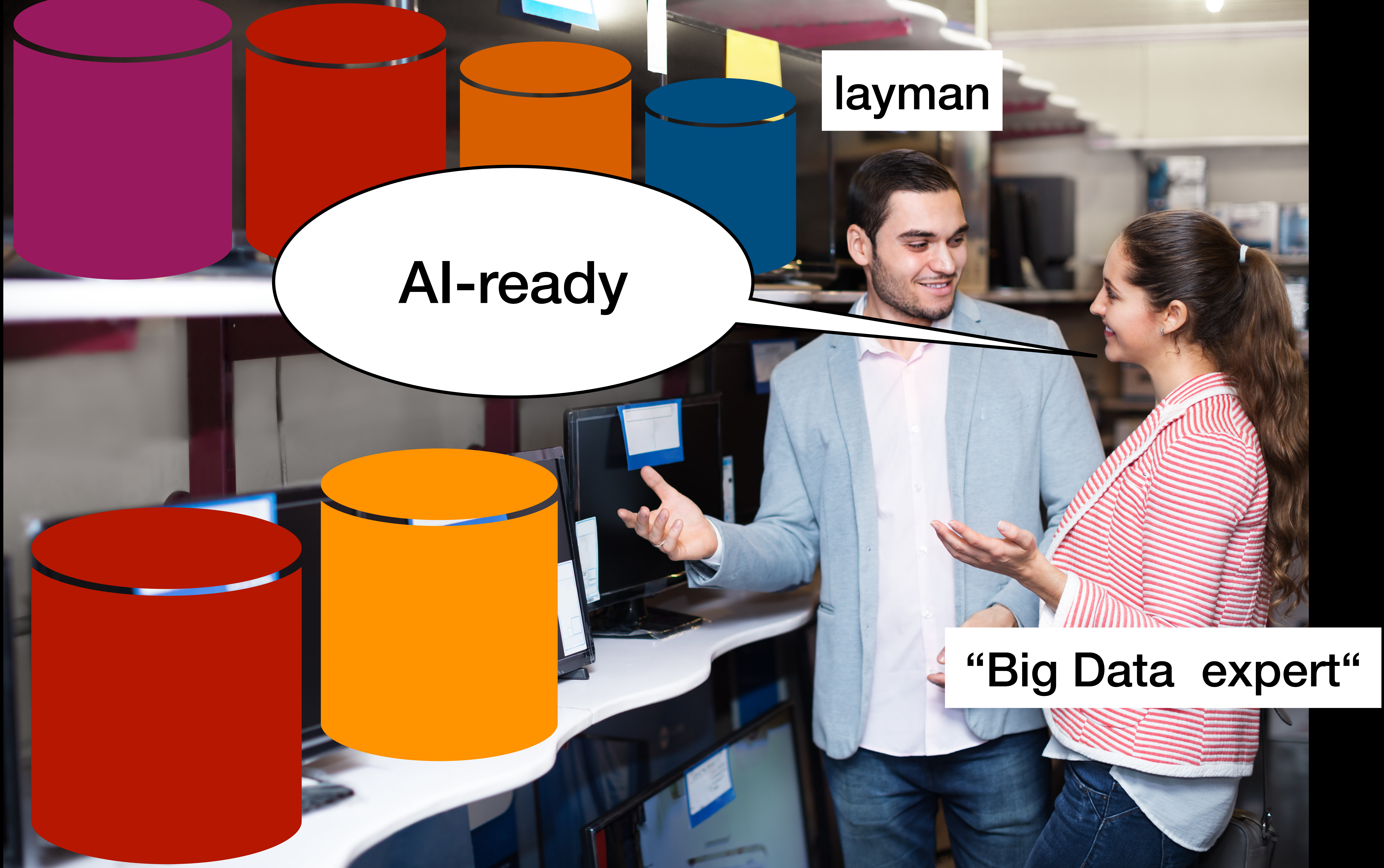

"TV expert"
laywoman

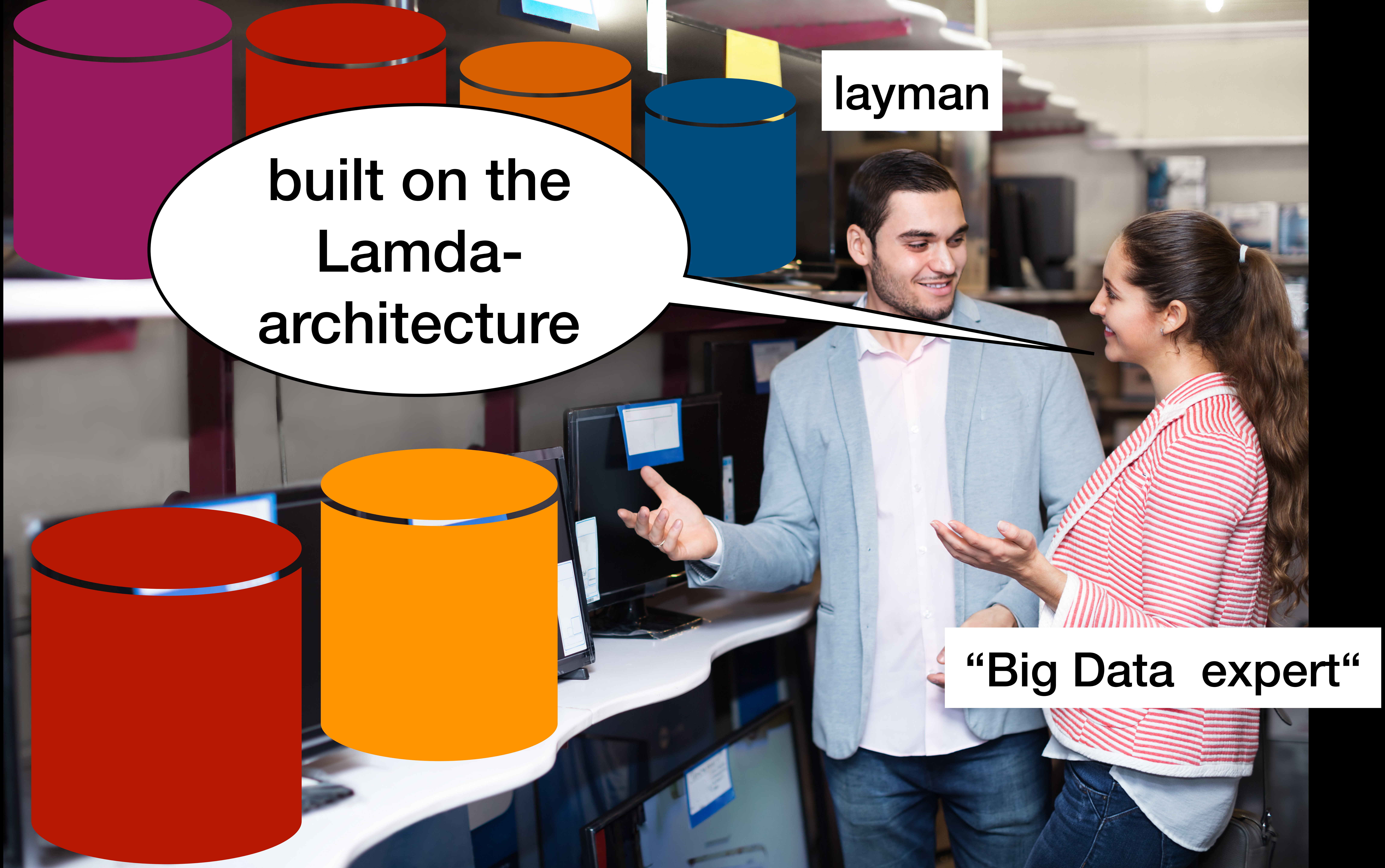

laywoman
“TV expert“
© istock.com JackF

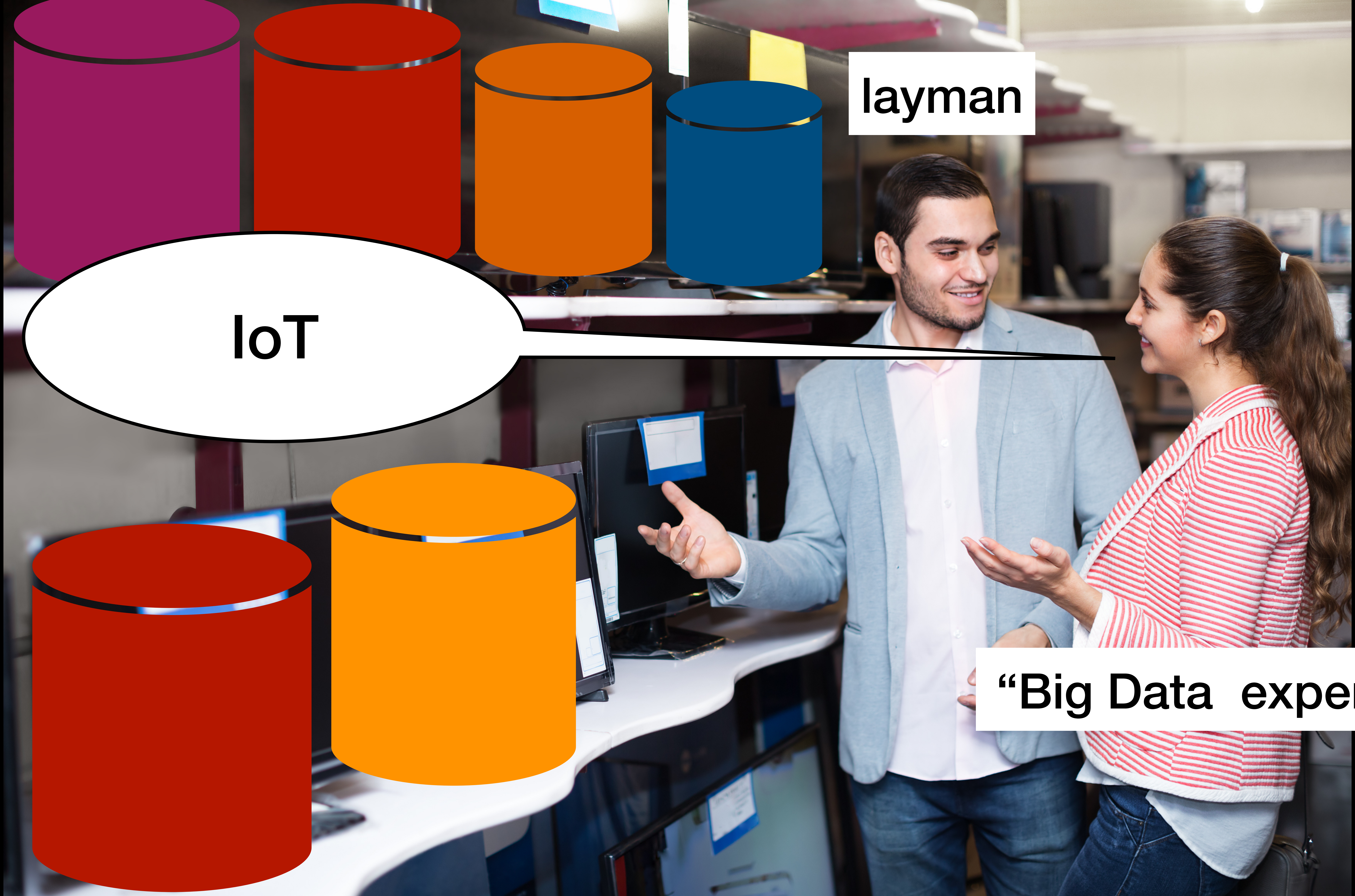

The leading NoSQL Data Lake solution in the cloud!
layman
“Big Data expert“

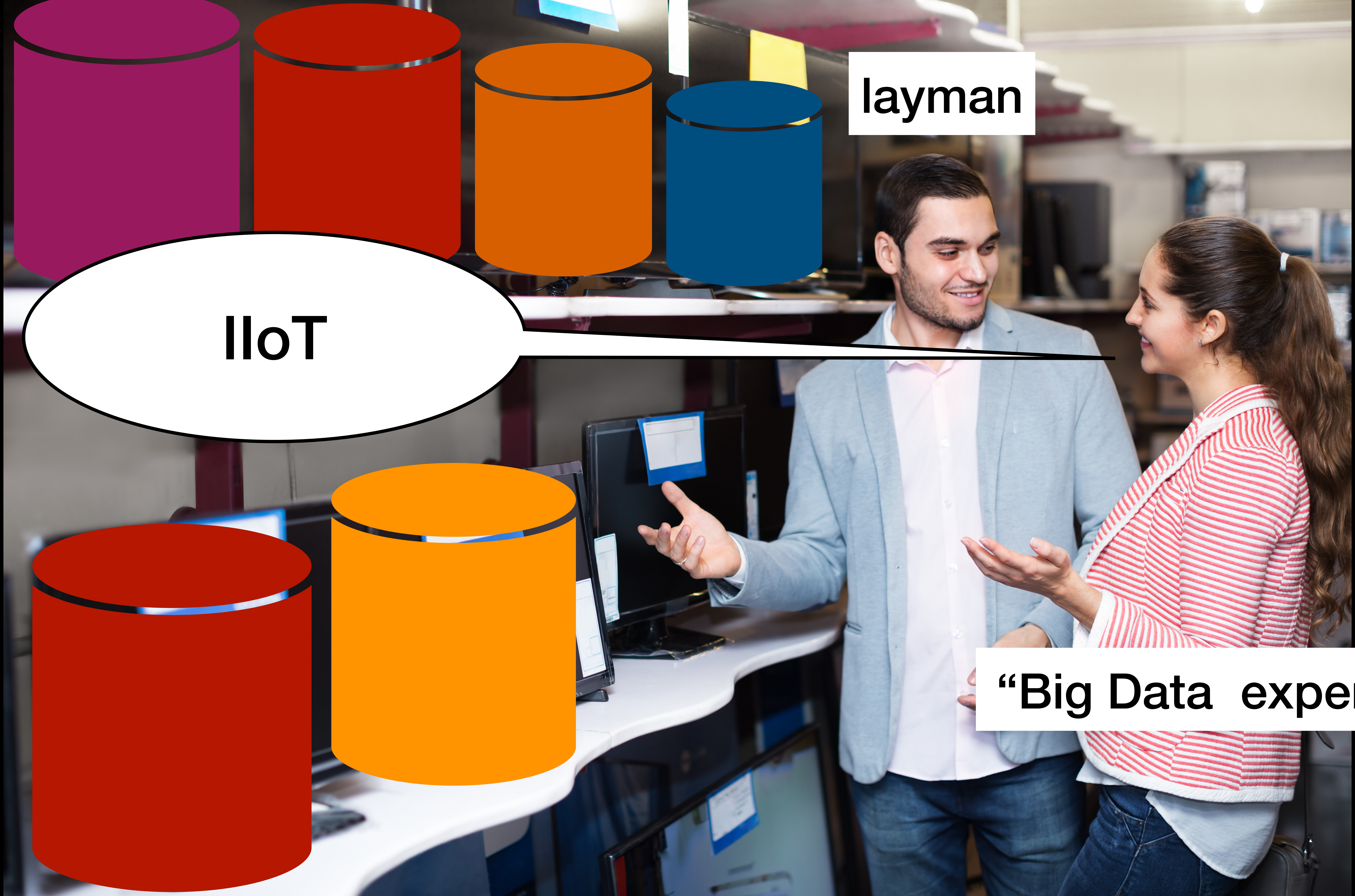


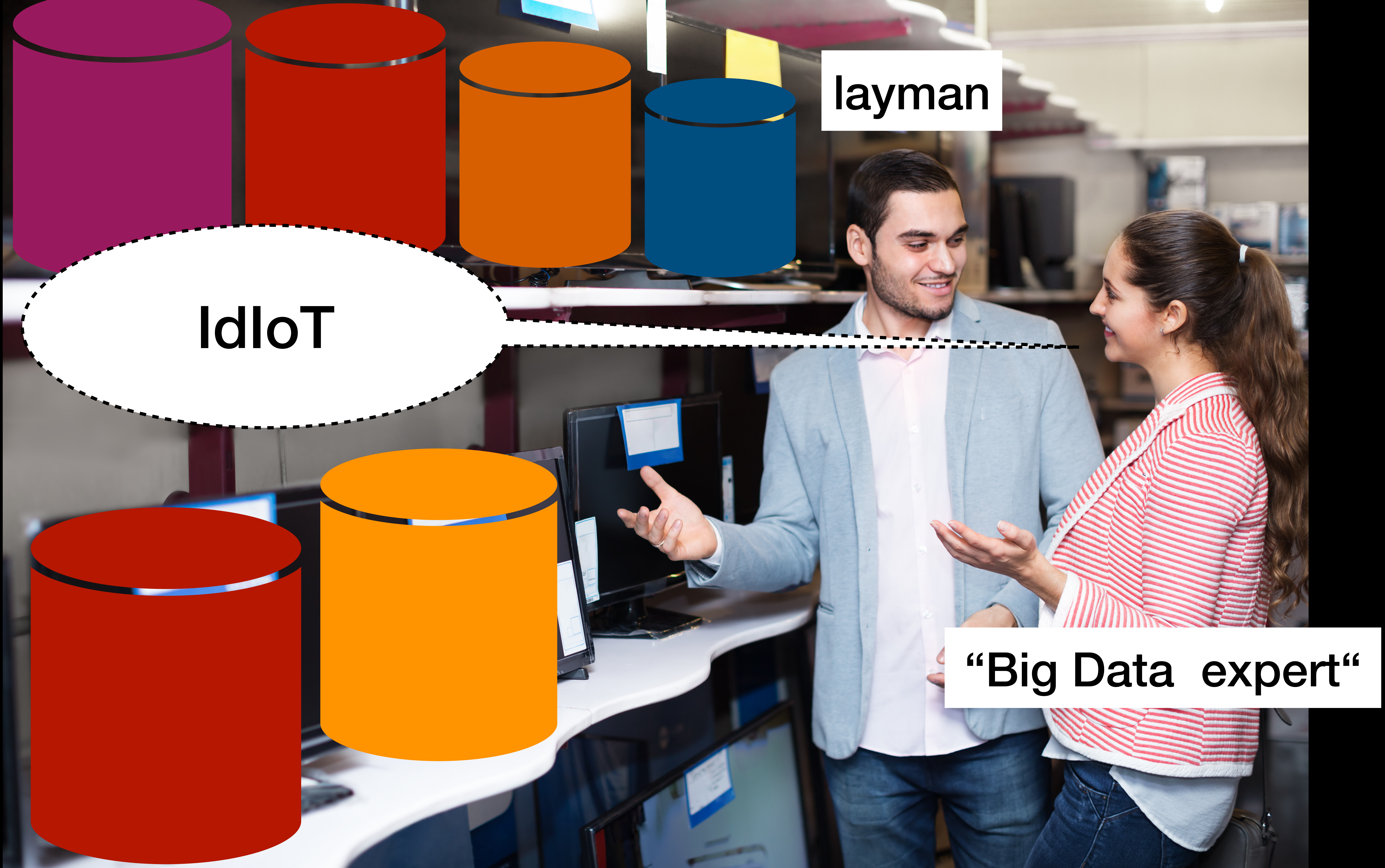
layman
Blockchain-enabled
“Big Data expert“


layman
AI-ready
“Big Data expert“

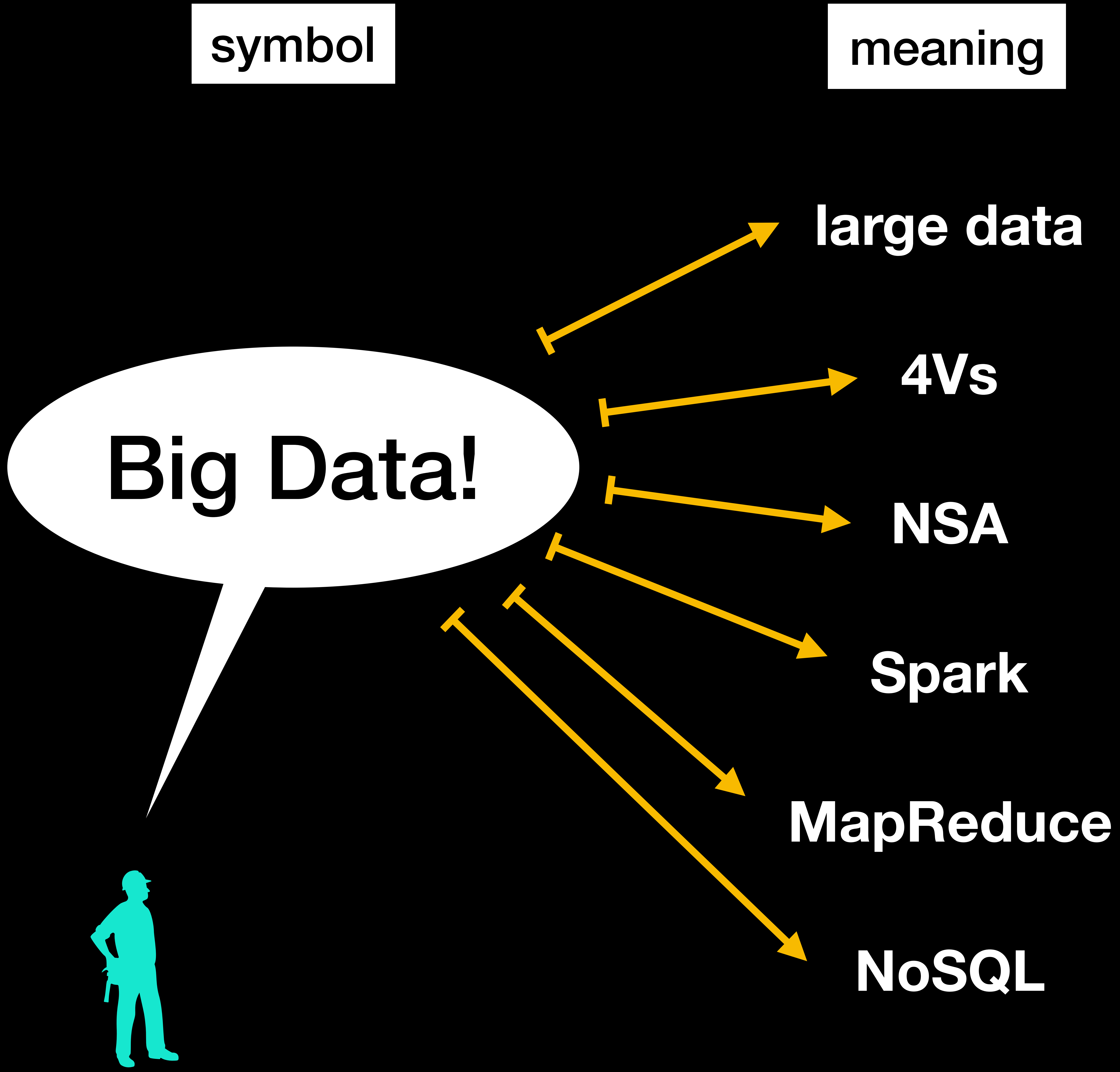

layman
built on the Lamda-architecture
“Big Data expert“

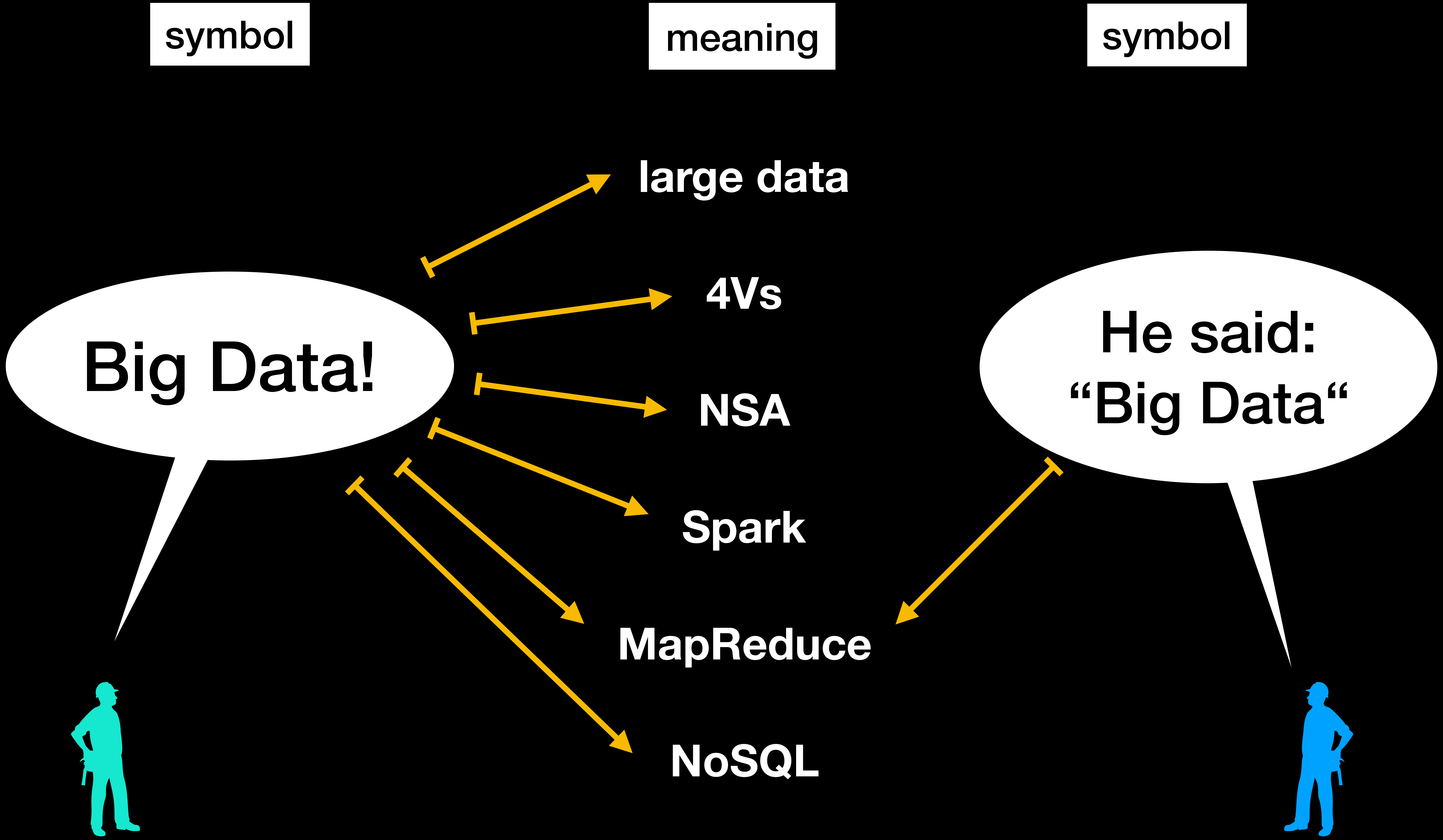

layman
IoT
“Big Data expert“

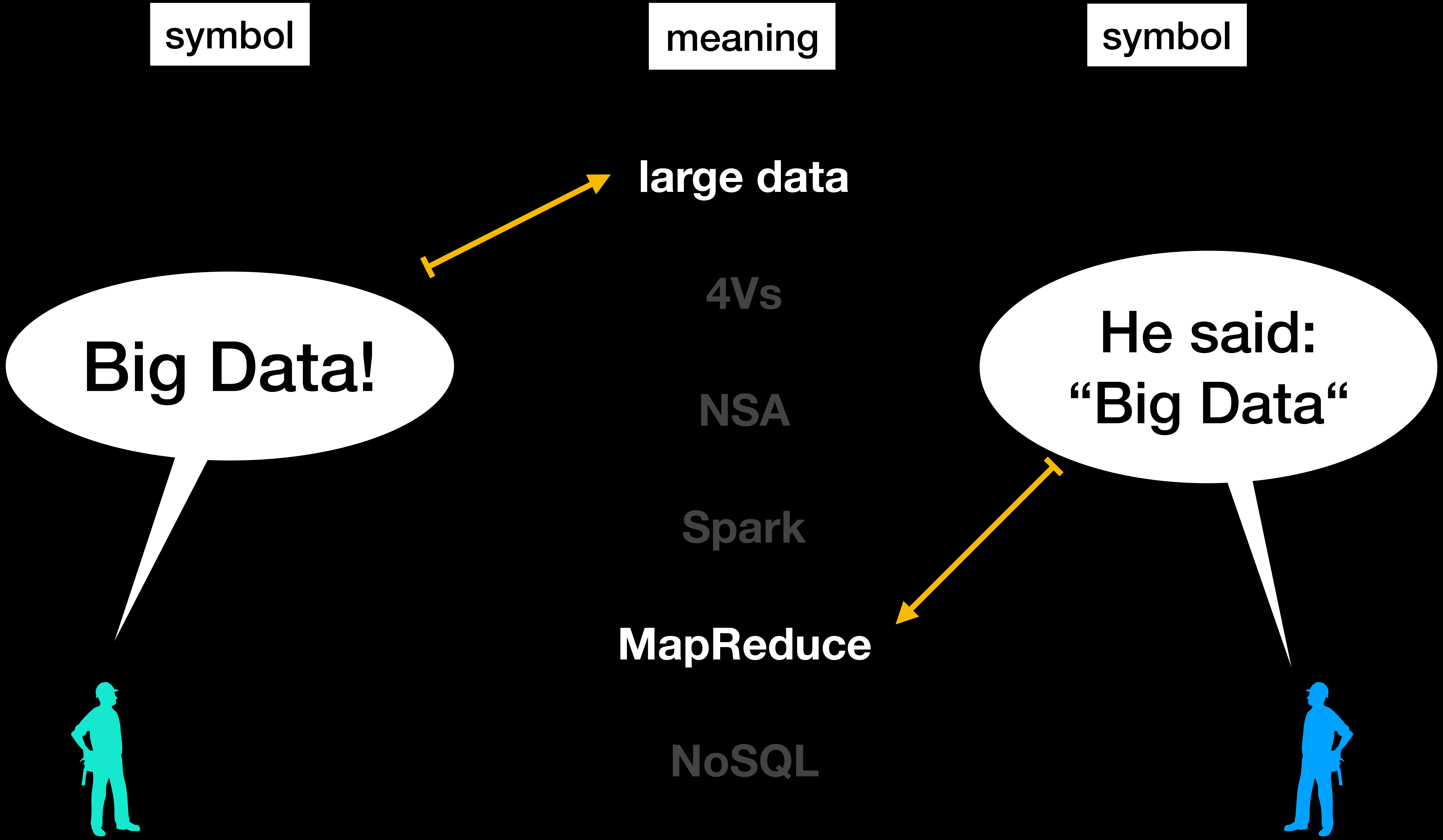

layman
IIoT
“Big Data expert“


layman
IdIoT
“Big Data expert“

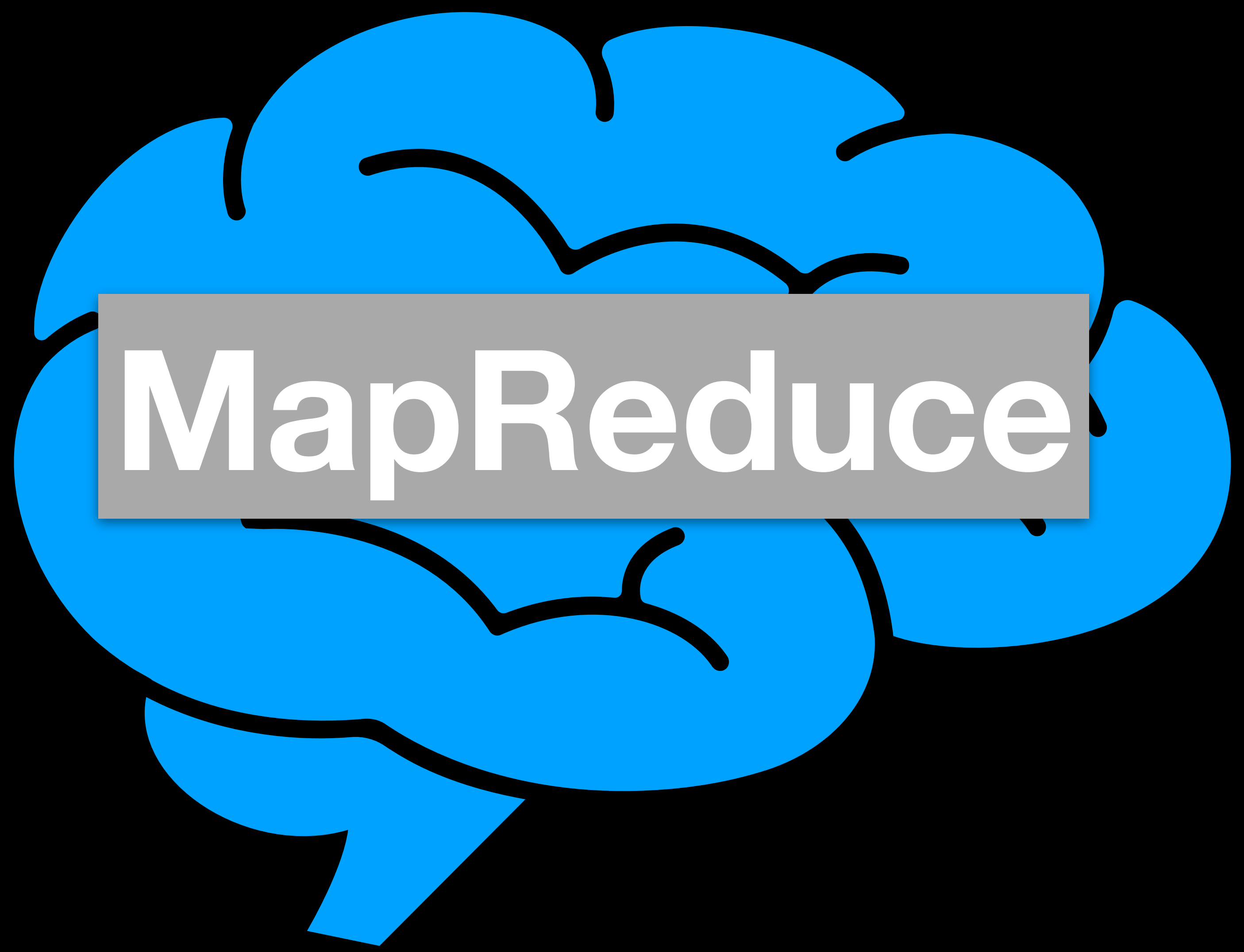
Problem 1:
ambiguous communication

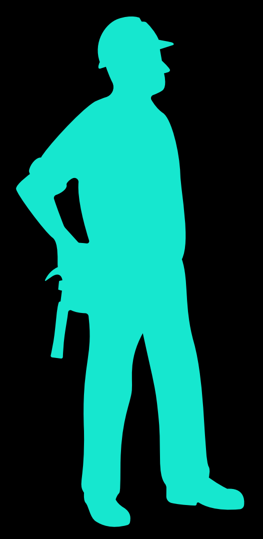

symbol
meaning
Big Data!
large data
4Vs
NSA
Spark
MapReduce
NoSQL

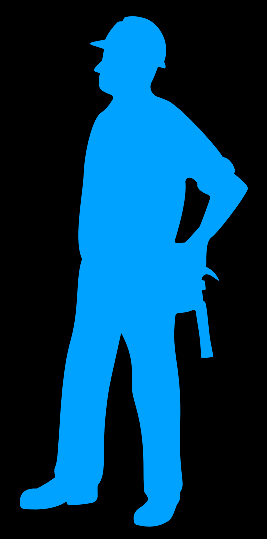

symbol
meaning
symbol
Big Data!
large data
4Vs
NSA
Spark
MapReduce
NoSQL
He said:
“Big Data“


symbol
meaning
symbol
large data
4Vs
NSA
Spark
MapReduce
NoSQL
Big Data!
He said: “Big Data“

translated to:



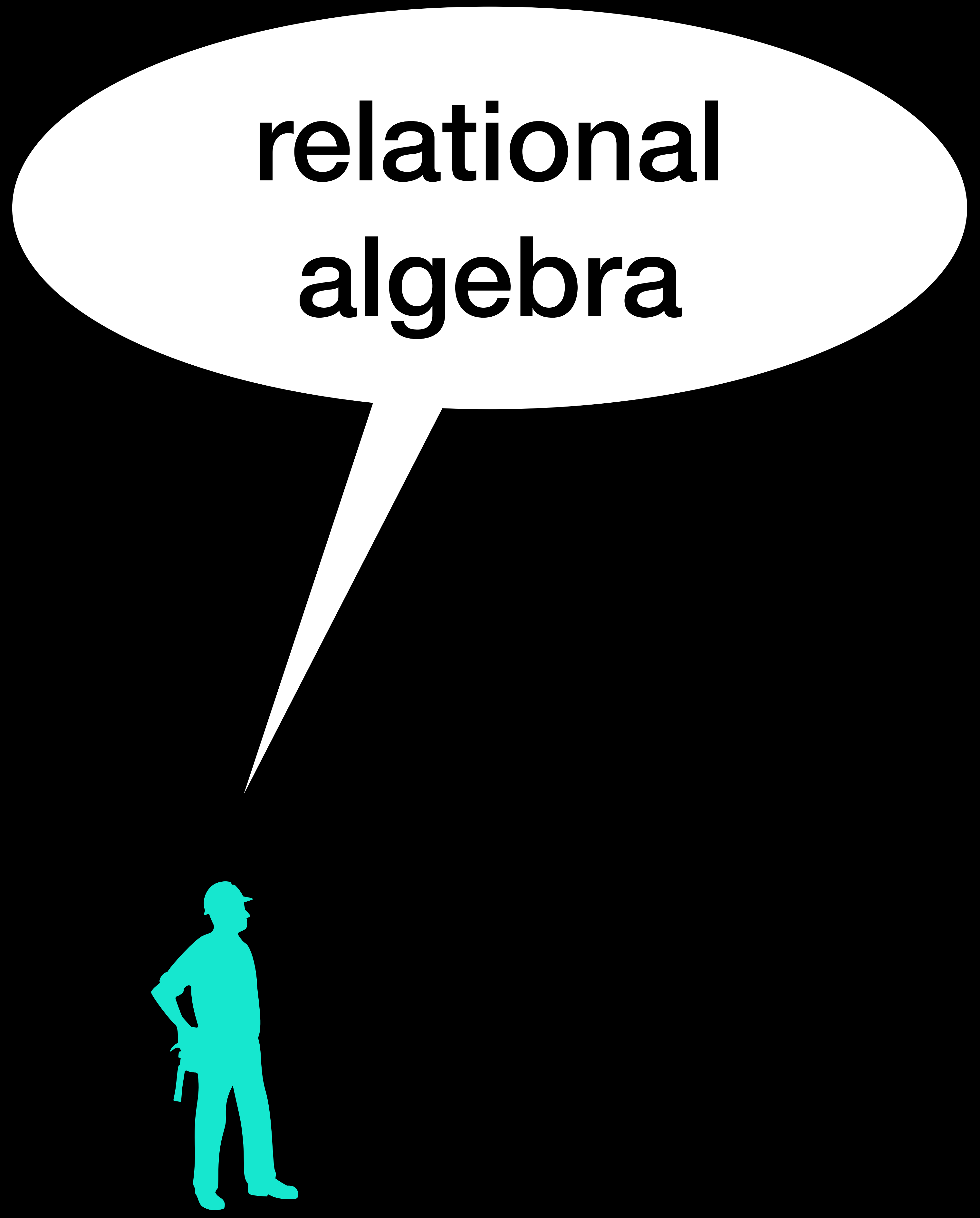




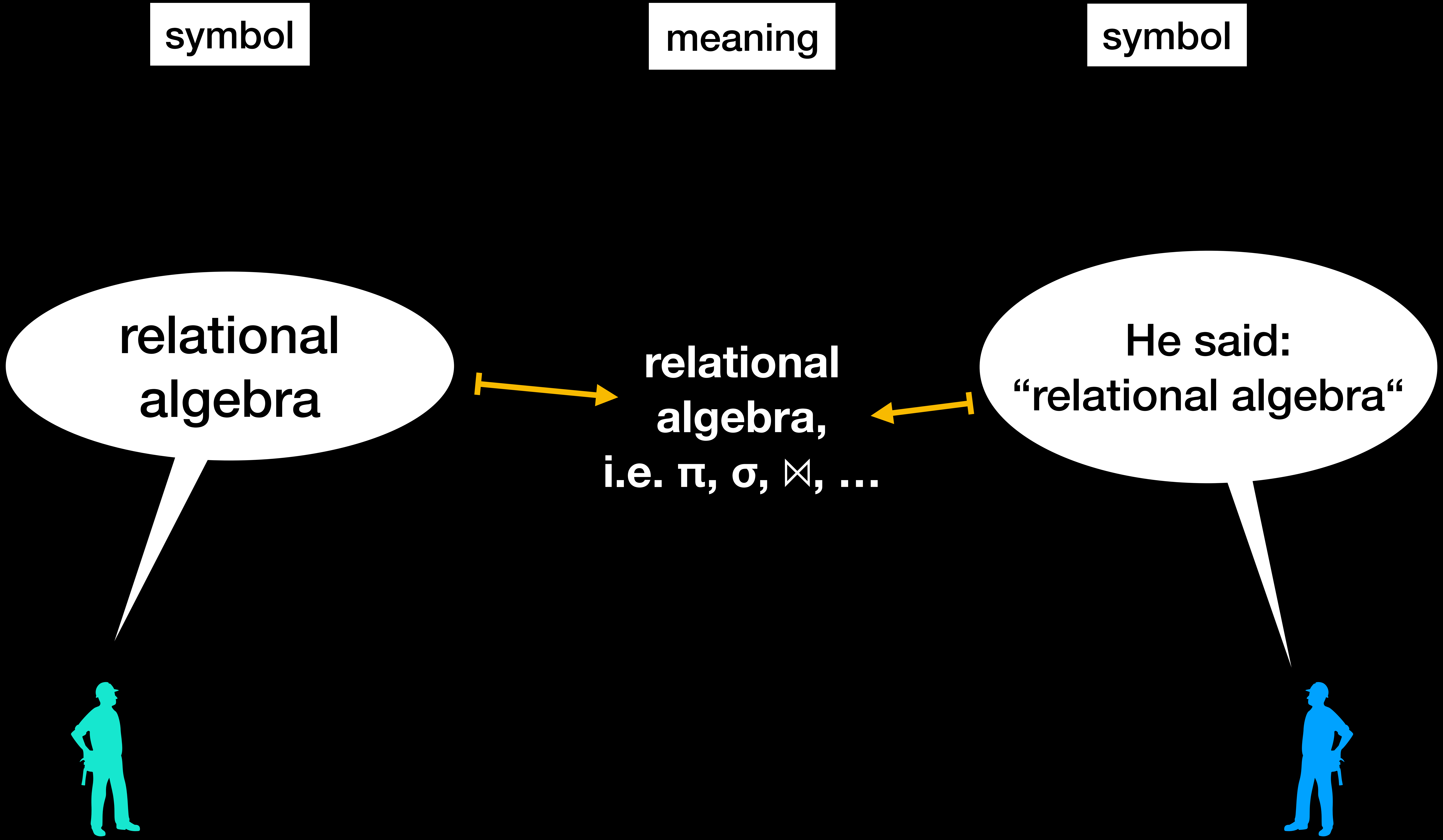
clear communication:

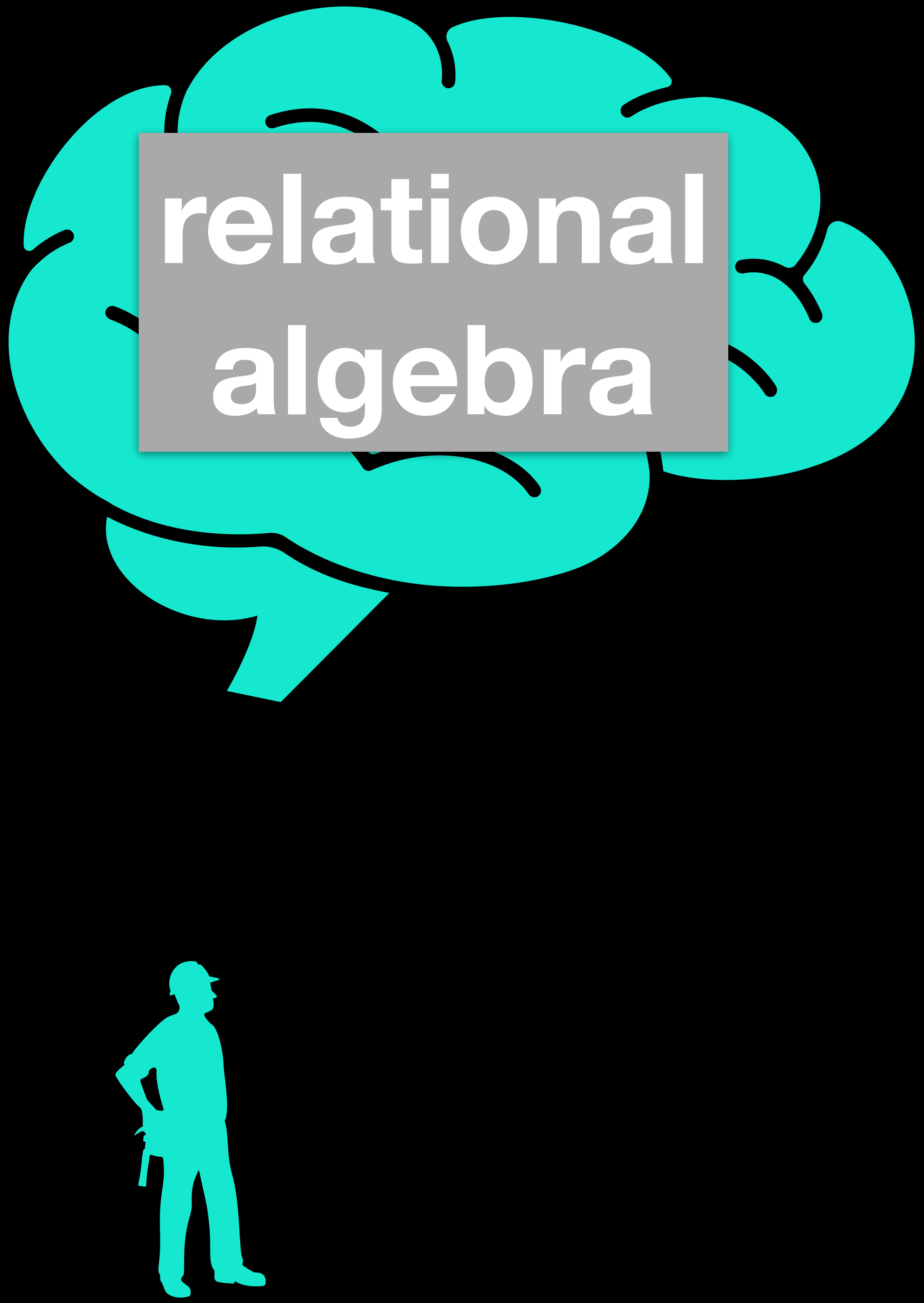
symbol

meaning

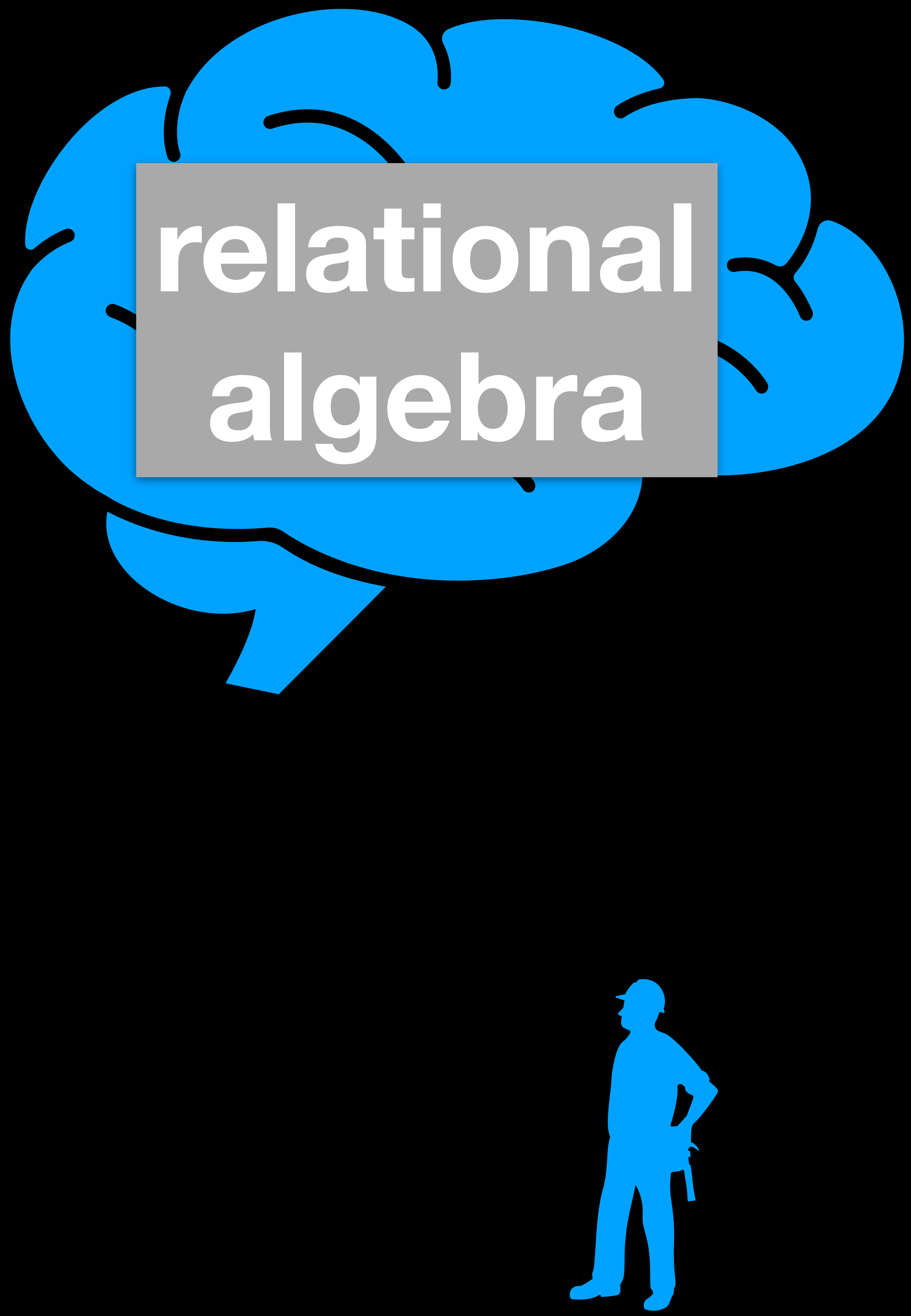


relational algebra,
i.e. π, σ, $\bowtie$, …


symbol
meaning
symbol
relational algebra
relational algebra, i.e. π, σ, ⋈, …
He said: “relational algebra“



translated to:



The symbol to meaning landscape

relational algebra

large data

Big Data

predicate pushdown

Data Science

ML

AI

relational model

Deep Learning

NoSQL

data layout

MapReduce

cloud

cost-based optimization

Hadoop

1

few

many

∞

meanings

advice: only use non-ambigous terms in communication

Problem 2:
confusion of dimensions

Big Data AI NoSQL Cloud

dimension 1:
fancy sounding buzzwords
(labels & terms)

dimension 2: technical principles and patterns (concepts, best practices)

predicate pushdown

relational model

relational algebra

data layouts,
e.g. column vs row

cost-based optimization

compress to save I/O

dimension 2: technical principles and patterns (concepts, best practices)

predicate pushdown

relational model

relational algebra

data layouts, e.g. column vs row

cost-based optimization

compress to save I/O

symbol

meaning

dimension 2: technical principles and patterns (concepts, best practices)

predicate pushdown

"filter and project data as early as possible"

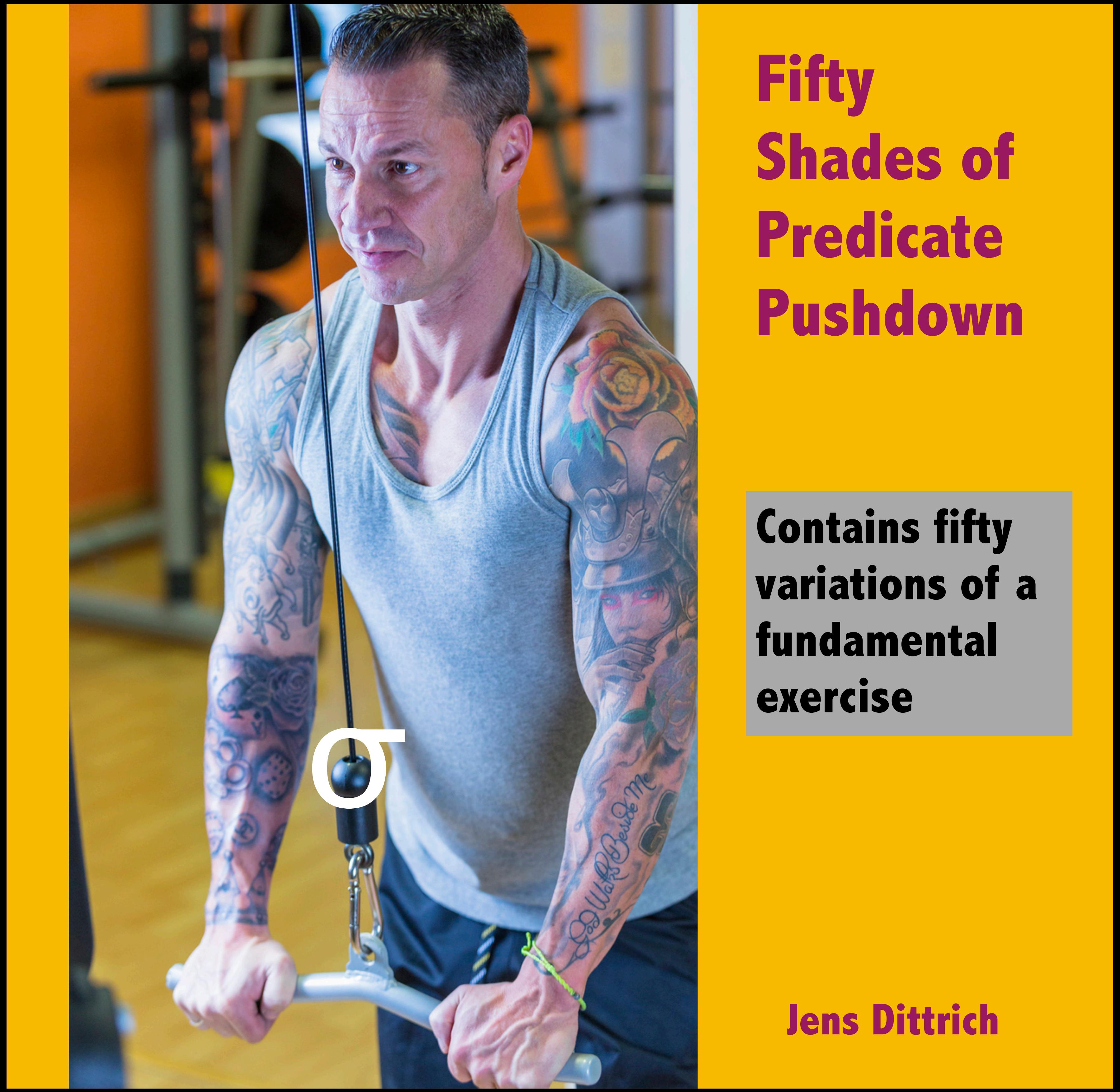
relational model

relational algebra

data layouts,
e.g. column vs row

cost-based optimization

compress to save I/O



symbol

meaning

dimension 2: technical principles and patterns (concepts, best practices)

predicate pushdown

relational model

“model all data as multi-attribute sets“

not to be confused with:

“tables“

“data frames“

“column stores“

“row stores“

relational algebra

data layouts,
e.g. column vs row

cost-based optimization

compress to save I/O

=> dimension 3: software platforms (concrete implementations & frameworks)

symbol

meaning

dimension 2: technical principles and patterns (concepts, best practices)

predicate pushdown

relational model

relational algebra

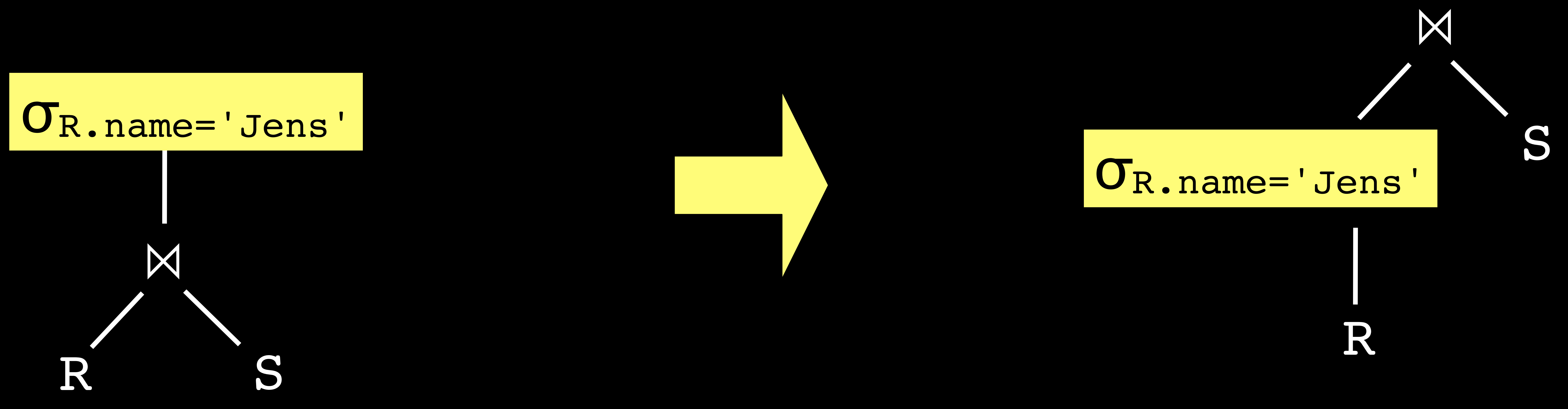
“query those sets through a combination of simple set-valued functions“

data layouts,
e.g. column vs row

cost-based optimization

compress to save I/O

Predicate Pushdown in Relational Model and Relational Algebra



checkout my youtube channel for videos explaining the foundations: https://www.youtube.com/user/jensdit

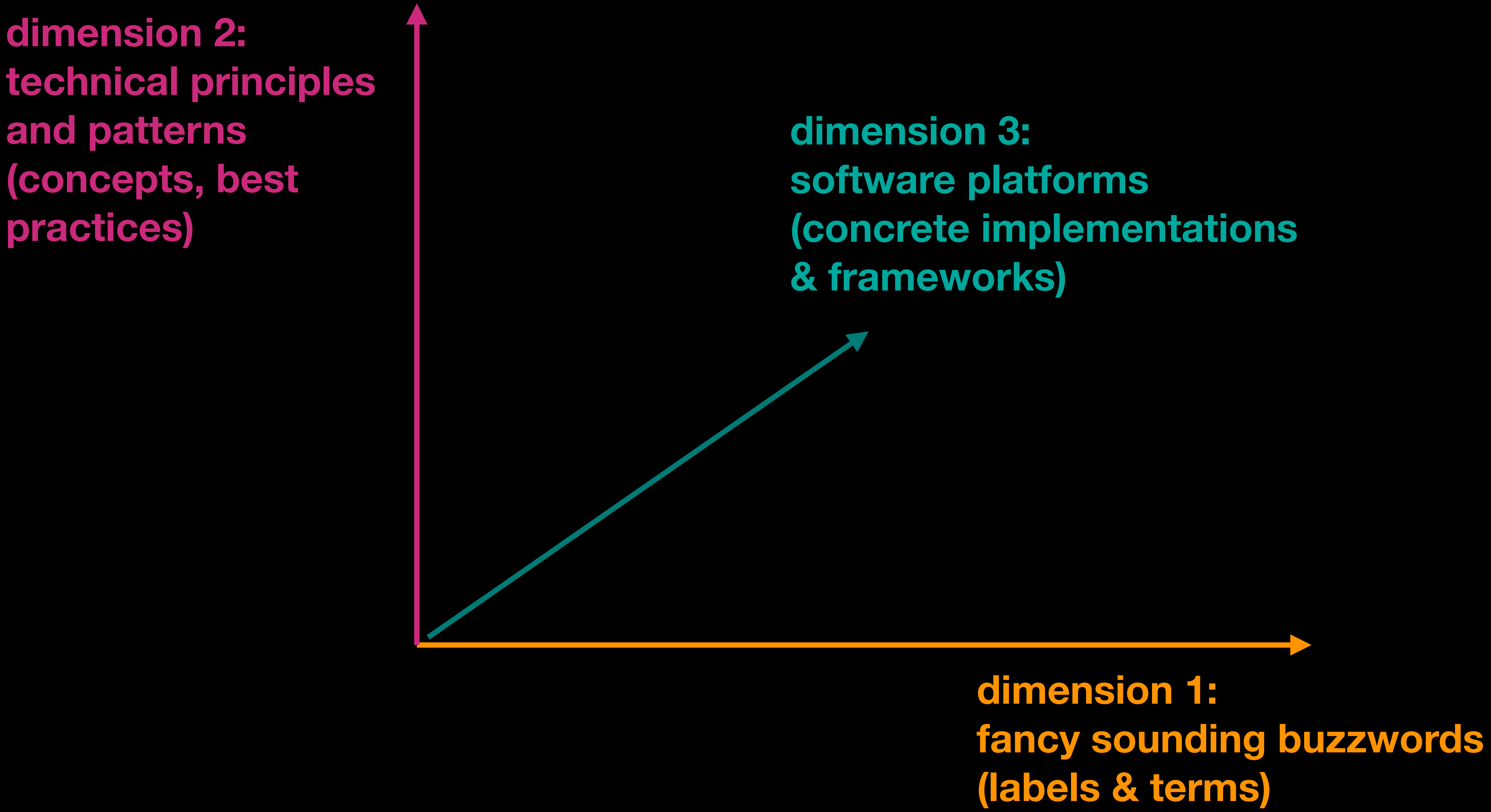
dimension 3:
software platforms
(concrete implementations
& frameworks)

Flink

Spark

PostgreSQL

MongoDB


dimension 2:
technical principles
and patterns
(concepts, best
practices)
dimension 3:
software platforms
(concrete implementations
& frameworks)
dimension 1:
fancy sounding buzzwords
(labels & terms)

A War Story[1]

[1]Let's say, I heard this story from a friend.

Situation:

Client has a "big data" problem with sensor data. Already got a Hadoop cluster, put his data there, learned Spark, wrote some Scala/Spark/Parquet program to analyze stuff.

?

meaning

large data

4Vs

MapReduce

NoSQL

Situation:

Client has a "big data" problem with sensor data. Already got a Hadoop cluster, put his data there, learned Spark, wrote some Scala/ Spark/Parquet program to analyze stuff.

?

meaning

Cluster with HDFS installed

Hadoop MapReduce

meaning

Situation:

Client has a "big data" problem with sensor data. Already got a Hadoop cluster, put his data there, learned $\boxed{\text{Spark,}}$ wrote some $\boxed{\text{Scala/ Spark/Parquet}}$ program to analyze stuff.

=> dimension 3: software platforms (concrete implementations & frameworks)


11th
Commandment:
If there is
Big Data,
thou shalt
use Spark or
MapReduce.
© istock.com fotofrankyat

Situation:

Client has a "big data" problem with sensor data. Already got a Hadoop cluster, put his data there, learned Spark, wrote some Scala/Spark/Parquet program to analyze stuff.

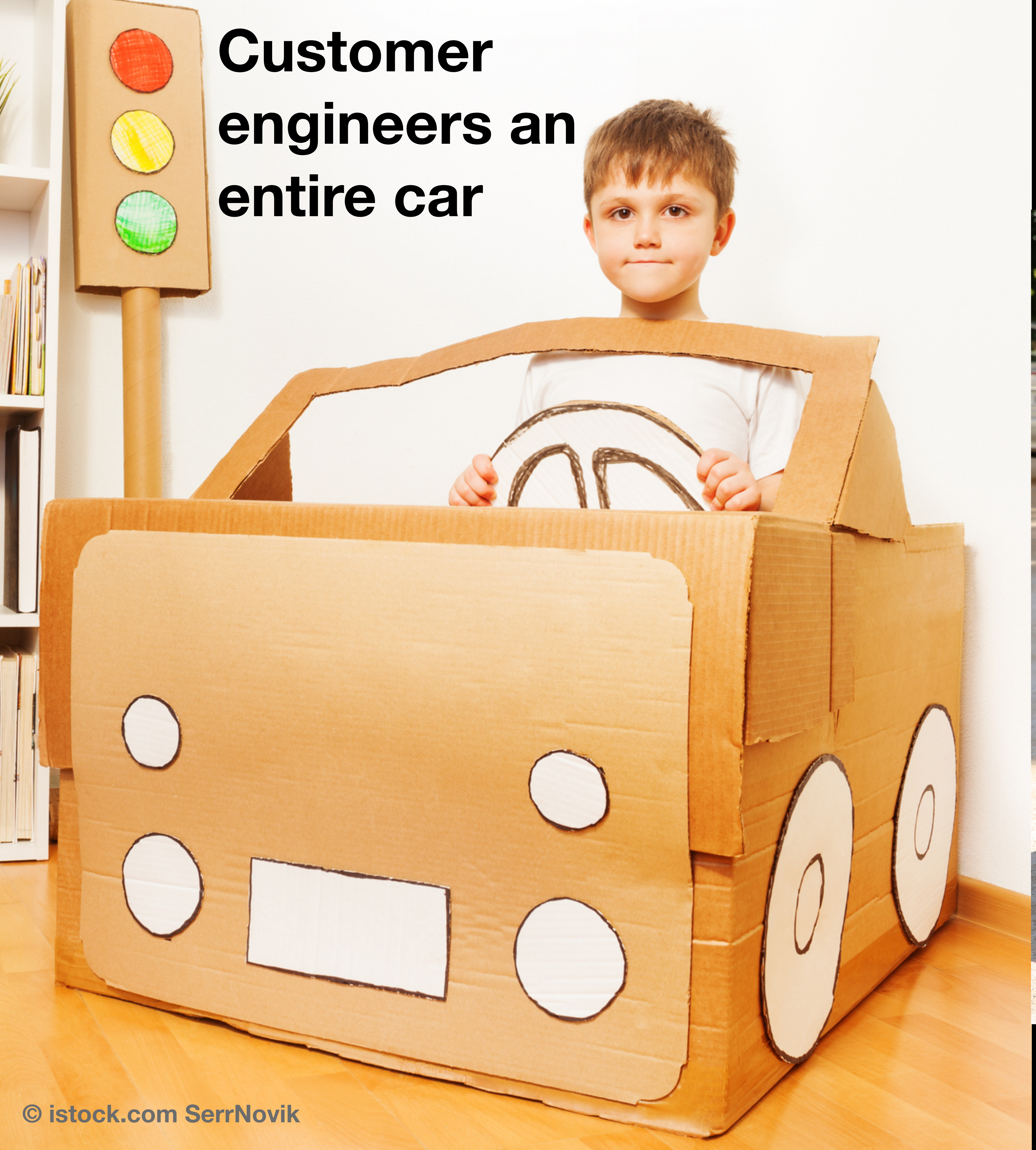
meaning

?

spaghetti code

flexible query processing

clear layering



rather than simply buying a state-of-the-art car

Situation:

Client has a "big data" problem with sensor data. Already got a Hadoop cluster, put his data there, learned Spark, wrote some Scala/Spark/Parquet program to analyze stuff.

More Details:

sensor traces stored in HDFS,
uses Parquet (column/PAX layout),
Software skims through traces,
highly parallelized through Spark,
queries are run on the cluster

Traces:
[timestamp, sensorid, value, fluff, more fluff]

e.g.
(12:00:34, 23, 7, fluff, more fluff)
(12:00:35, 56, 56, fluff, more fluff)
(12:00:37, 123, 9, fluff, more fluff)
(12:00:39, 89, 131, fluff, more fluff)
(12:00:39, 5567, 156, fluff, more fluff)
(12:00:41, 3, A, fluff, more fluff)
(12:00:43, 5785, 4213, fluff, more fluff)
(12:00:43, 4365, 9, fluff, more fluff)
(12:00:44, 37, 121, fluff, more fluff)
(12:00:44, 335, 156, fluff, more fluff)
(12:00:45, 23, zz, fluff, more fluff)
(12:00:47, 373, 354, fluff, more fluff)

Desired:
queries of the following type:
when is sensor<x> <operator> <value>?

e.g. value of sensor42 > 15.0

e.g. value of
sensor15 > 15.0 AND sensor77 < 9.0

only few attributes in each query

Traces:

[timestamp, sensorid, value, fluff, more fluff]

e.g.

(12:00:34, 23, 7, fluff, more fluff)
(12:00:35, 56, 56, fluff, more fluff)
(12:00:37, 123, 9, fluff, more fluff)
(12:00:39, 89, 131, fluff, more fluff)
(12:00:39, 5567, 156, fluff, more fluff)
(12:00:41, 3, A, fluff, more fluff)
(12:00:43, 5785, 4213, fluff, more fluff)
(12:00:43, 4365, 9, fluff, more fluff)
(12:00:44, 37, 121, fluff, more fluff)
(12:00:44, 335, 156, fluff, more fluff)
(12:00:45, 23, zz, fluff, more fluff)
(12:00:47, 373, 354, fluff, more fluff)

Observation:

fluff not required for query processing

Optimization:

1. Remove the fluff

Traces:

[timestamp, sensorid, value]

e.g.

(12:00:34, 23, 7)
(12:00:35, 56, 56)
(12:00:37, 123, 9)
(12:00:39, 89, 131)
(12:00:39, 5567, 156)
(12:00:41, 3, A)
(12:00:43, 5785, 4213)
(12:00:43, 4365, 9)
(12:00:44, 37, 121)
(12:00:44, 335, 156)
(12:00:45, 23, zz)
(12:00:47, 373, 354)

Observation:

even though data is mapped to Parquet (columnar PAX-format), this does not help query processing in this case: no clustering of attributes!

Optimization:

2. change format to
[sensorid, timestamp, value]
or:
partition by sensorid
(same effect)

dimension 2:
technical principle:

data layouts,
e.g. column vs row

Traces:

[timestamp, sensorid, value]

e.g.
(12:00:34, 23, 7)
(12:00:35, 23, 8)
(12:00:36, 23, 9)
(12:00:37, 23, 8)
(12:00:38, 23, 7)
(12:00:39, 23, 6)
…
(12:00:34, 24, 45)
(12:00:35, 24, 44)
(12:00:36, 24, 43)
(12:00:37, 24, 42)
(12:00:38, 24, 43)

Observation:

now, within each partition the sensorid is redundant

Optimization:

3. change format to
[timestamp, value]
and store sensorid once per partition

dimension 2:
technical principle:

compress to save I/O

Traces:

[timestamp, value]

e.g.

(12:00:34, 7)

(12:00:35, 8)

(12:00:36, 9)

(12:00:37, 8)

(12:00:38, 7)

(12:00:39, 6)

…

(12:00:34, 45)

(12:00:35, 44)

(12:00:36, 43)

(12:00:37, 42)

(12:00:38, 43)

23

24

× 500 speedup

Traces:

[timestamp, value]

e.g.

(12:00:34, 7)
(12:00:35, 8)
(12:00:36, 9)
(12:00:37, 8)
(12:00:38, 7)
(12:00:39, 6)
…
(12:00:34, 45)
(12:00:35, 44)
(12:00:36, 43)
(12:00:37, 42)
(12:00:38, 43)

23

24

Observation:

redundancy due to dense timestamps and continuous measurements

Optimization:

4. difference encode data

dimension 2:
technical principle:

compress to save I/O

Traces:

[timestamp, value]

e.g.

(12:00:34, 7)

(1, 1)

(1, 1)

(1, -1) **23**

(1, -1)

(1, -1)

…

(12:00:34, 7)

(1, -1)

(1, -1) **24**

(1, -1)

(1, 1)

× 12 speedup

=> × 6000 speedup

Traces:

[timestamp, value]

e.g.

(12:00:34, 7)

(1, 1)

(1, 1)

(1, -1)

(1, -1)

(1, -1)

…

(12:00:34, 7)

(1, -1)

(1, -1)

(1, -1)

(1, 1)

23

24

Observation:

data locality increases due to attribute clustering. This was not possible with the old layout.

=> much better exploitation of the storage hierarchy: likelihood increases that some of the columns are entirely kept in main memory or on some SSD as an additional buffer

dimension 2:
technical principle:

cluster data by hotness along the storage hierarchy

Traces:

[timestamp, value]

e.g.

(12:00:34, 7)
(1, 1)
(1, 1)
(1, -1)
(1, -1)
(1, -1)

Hot Data

23

…

(12:00:34, 7)
(1, -1)
(1, -1)
(1, -1)
(1, 1)

Cool Data

24

Observation:

data locality increases due to attribute clustering. This was not possible with the old layout.

=> much better exploitation of the storage hierarchy: likelihood increases that some of the columns are entirely kept in main memory or on some SSD as an additional buffer

dimension 2:
technical principle:

cluster data by hotness along the storage hierarchy

× ~10 speedup

Traces:

[timestamp, value]

e.g.

(12:00:34, 7)

(1, 1)

(1, 1)

(1, -1)

(1, -1)

(1, -1)

…

(12:00:34, 7)

(1, -1)

(1, -1)

(1, -1)

(1, 1)

Hot Data

23

Cool Data

24

**some handwaving here:
depends to some degree on the
actual query patterns,
but this is the ballpark**

in total ~10,000 speedup

(Positive) Side Effects

1. data could be stored in compressed format already when being created on the machine,
   => factor 12 less storage, hardware/bandwidth savings along the entire data generation and processing pipeline

2. there is no need to heavily parallelize queries here
   => no need to use heavy-lifting with Spark, no fat clusters or similar

3. QP is very lightweight!
   => QP can be done on very thin-clients even a smartphone

4. only overall datasizes are the limit for the client; however, if users are interested in attribute subsets anyways,
   => with our layout, they can easily pull those attribute subsets to their laptop/smartphone

5. PS: and there is even more you can do...
   => this kind of query performance enables new types of analytics, e.g. online anomaly detection, etc...

Process Recap:

1. client did some "Big Data/Spark-Thingie"
   (dimension 1)

2. we went back to really understanding the client's original problem: what does he actually want to do?

3. we combined a couple of fundamental technical principles of data management, namely data layouts, compression, hot/cool-clustering
   (dimension 2)

4. the synergies trigger I/O-time savings of a factor ~10,000, and storage savings of ~factor 12

5. all of this is totally software platform independent!
   (dimension 3)

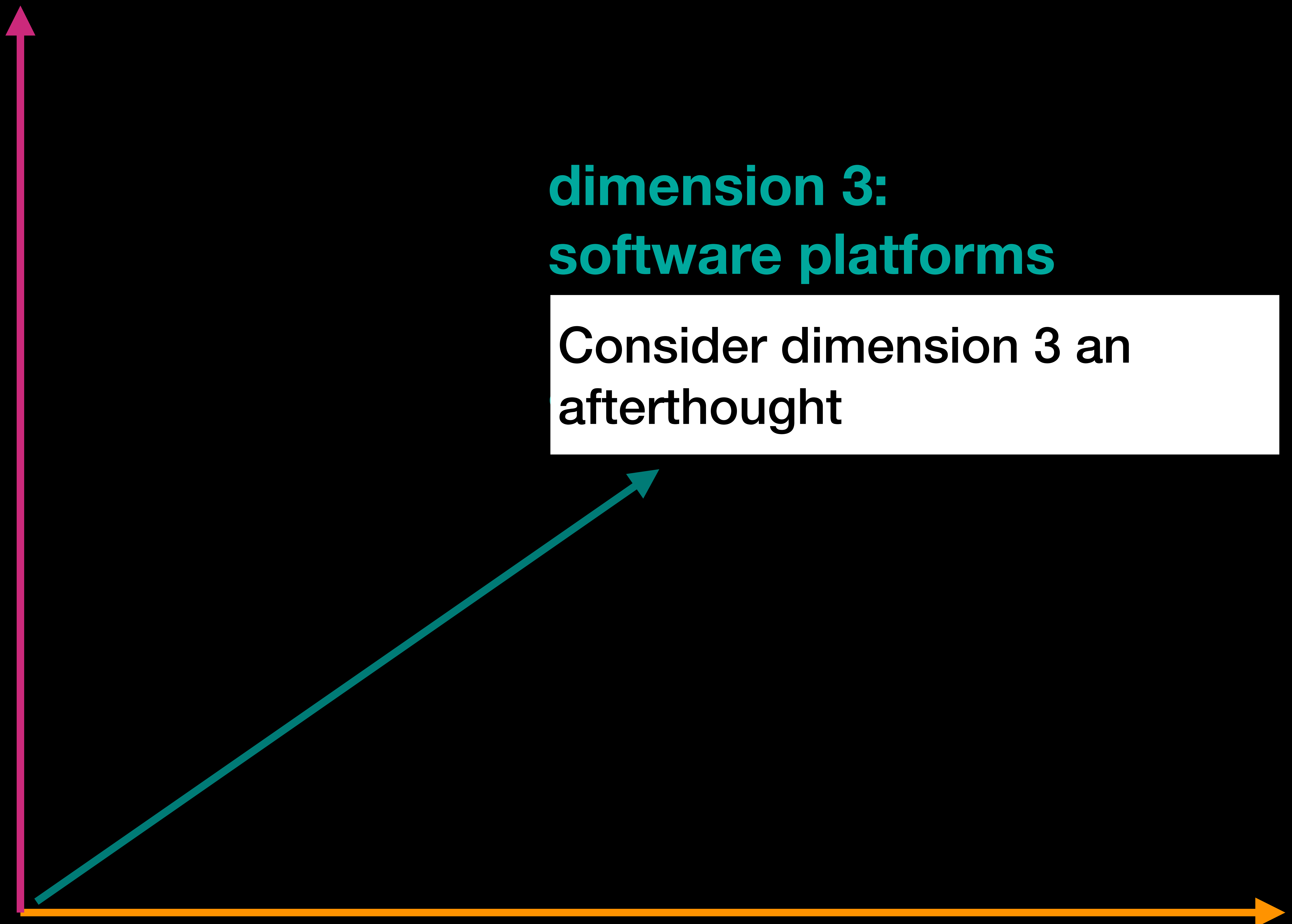
Takeaways:

dimension 2:
technical principles and patterns

Design solutions on dimension 2 only

dimension 3:
software platforms

Consider dimension 3 an afterthought



dimension 1:
fancy sounding buzzwords

Realize that dimension 1 is solely about marketing

do **NOT** confuse the three dimensions, unless you want to explicity fool people (which I do not recommend in any situation) d:AI:mond

http://daimond.ai

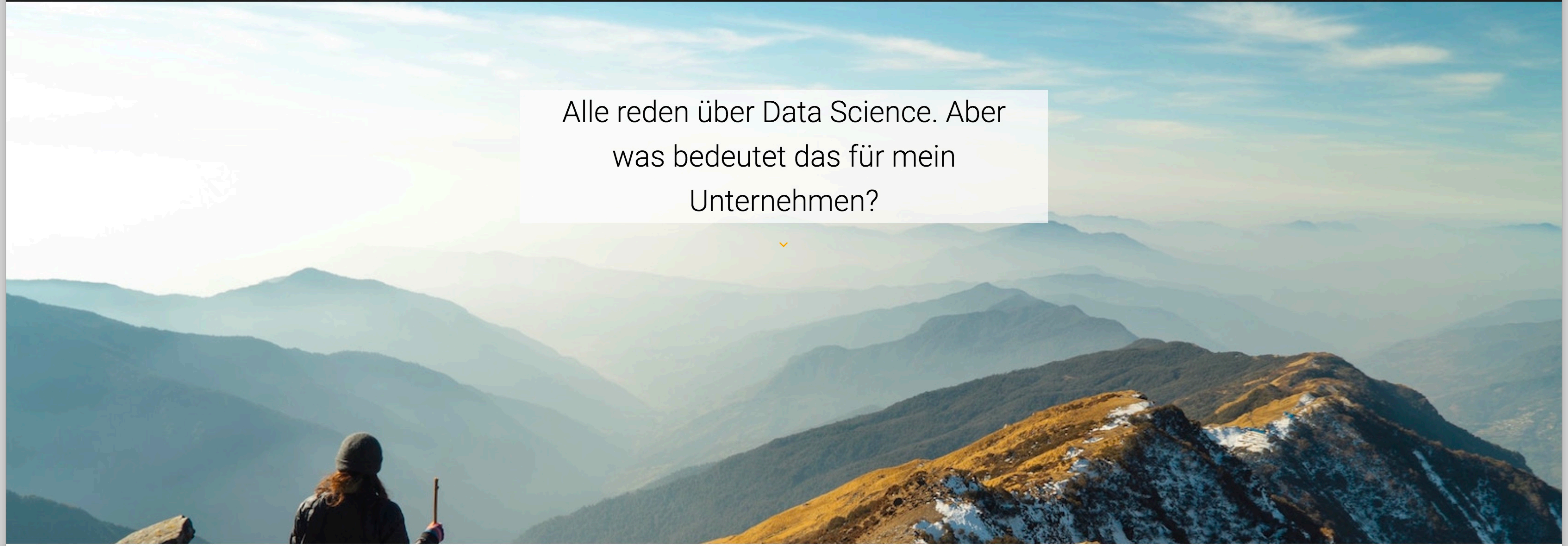
d:AI:mond - data science consulting

daimond.ai

Suchen

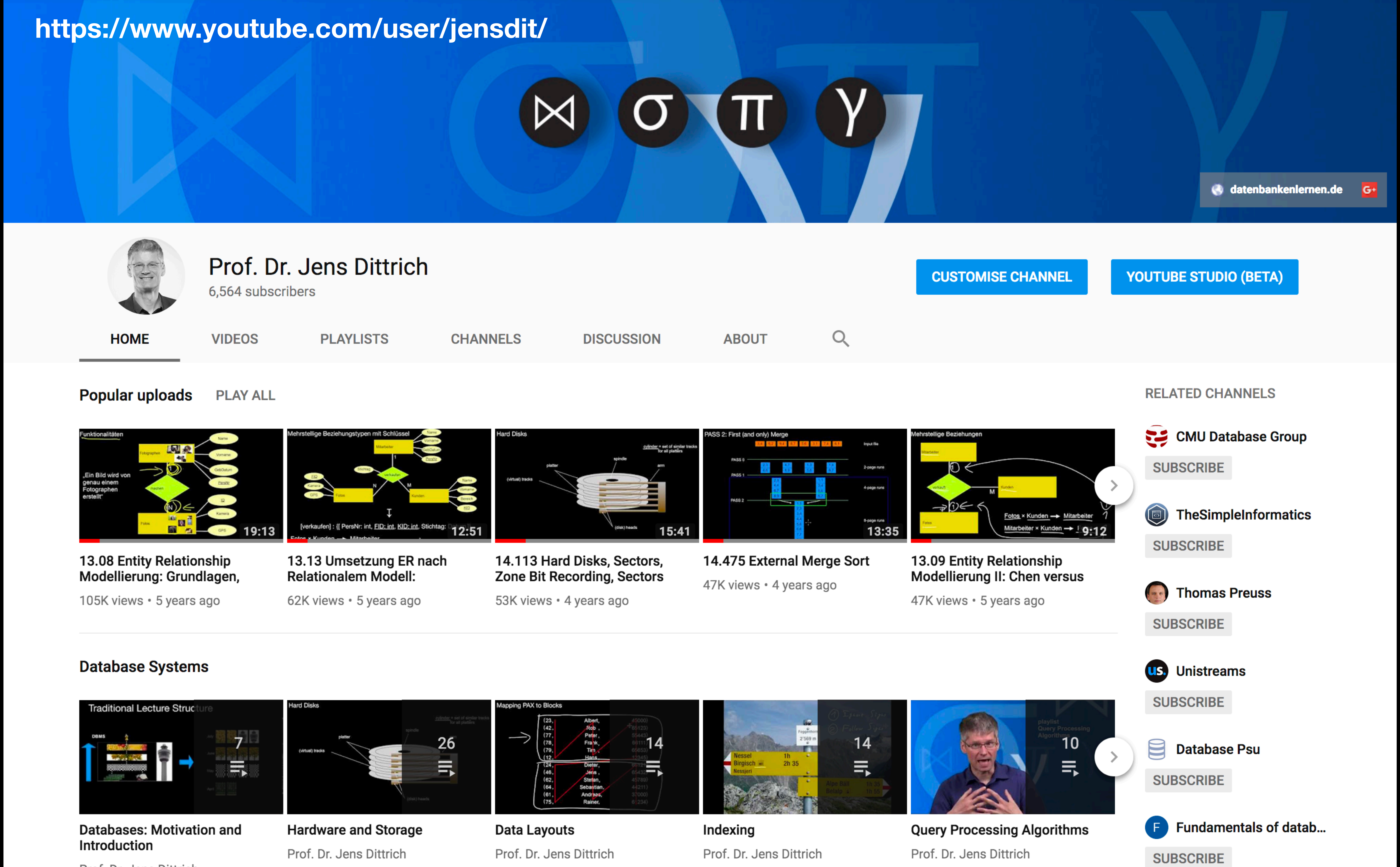
D:AI:MOND DATA SCIENCE CONSULTING

Potenzialanalyse Projektrealisierung Team Kontakt Blog



BRINGEN SIE DATA SCIENCE IN IHR UNTERNEHMEN

Als Data Science Consulting helfen wir Ihnen Mehrwerte aus ihren Daten zu schöpfen und dadurch mehr Effizienz, Transparenz und Struktur in ihr Unternehmen zu bringen. Dadurch können Sie sich Wettbewerbsvorteile sichern, Produktionsketten effizienter gestalten, neue


https://www.youtube.com/user/jensdit/
datenbankenlernen.de
Prof. Dr. Jens Dittrich
6,564 subscribers
CUSTOMISE CHANNEL
YOUTUBE STUDIO (BETA)
HOME
VIDEOS
PLAYLISTS
CHANNELS
DISCUSSION
ABOUT
Popular uploads
PLAY ALL
19:13
13.08 Entity Relationship Modellierung: Grundlagen,
105K views • 5 years ago
12:51
13.13 Umsetzung ER nach Relationalem Modell:
62K views • 5 years ago
15:41
14.113 Hard Disks, Sectors, Zone Bit Recording, Sectors
53K views • 4 years ago
13:35
14.475 External Merge Sort
47K views • 4 years ago
9:12
13.09 Entity Relationship Modellierung II: Chen versus
47K views • 5 years ago
Database Systems
7
Databases: Motivation and Introduction
26
Hardware and Storage
Prof. Dr. Jens Dittrich
14
Data Layouts
Prof. Dr. Jens Dittrich
14
Indexing
Prof. Dr. Jens Dittrich
10
Query Processing Algorithms
Prof. Dr. Jens Dittrich
RELATED CHANNELS
CMU Database Group
SUBSCRIBE
TheSimpleInformatics
SUBSCRIBE
Thomas Preuss
SUBSCRIBE
Unistreams
SUBSCRIBE
Database Psu
SUBSCRIBE
Fundamentals of datab...
SUBSCRIBE

Teaching

Materialien

1. Vorlesungen

- 01a Einführung (1.9 MB)
- 01b Einführung (464 KB)
- 02 Bug Data Analytics (1.4 MB)
- 03a Relationales Modell und Algebra (1.9 MB)
- 03b Zusätzliche Beispiele für abgeleitete Operatoren (159 KB)
- 04 Datenbanken (276 KB)
- 05 Data Exploration (2.3 MB)
- 06 Statistik (26 MB)
- 07 MapReduce & Apache Spark (1.1 MB)
- 08 Lineare Regression (2.7 MB)
- 09 Klassifizierung (4.9 MB)
- 10 Entscheidungsbäume (672 KB)
- 11 Neural Networks (5.2 MB)
- 12 Clustering (4.5 MB)

2. Notebooks

- 02 Data Cleaning (137 KB)
- 03 Relationale Algebra (12 KB)
- 04 SQL (19 KB)
- 07 Spark (13 KB)
- 10 Entscheidungsbäume (6.0 KB)
- 11 Neural Networks (2.1 MB)

localhost:8888/notebooks/notebooks/02 Data Cleaning.ipynb

jupyter 02 Data Cleaning Last Checkpoint: 12.04.2018 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [30]:

```
# bedingte Einfärbung der Zellen des DataFrames:
linksPivot.style.apply(lambda x: ["background: orange" if v >= 42 else "background: lightgreen" if v > 0 else "" for v
```

Out[30]:

target source	Action	Adventure	Animation	Biography	Comedy	Crime	Drama	Family	Fantasy	History	Horror	Music	Musical	Mystery	Romance	Sci-Fi	Sport
Action	2	155	10	11	45	54	72	7	40	4	19	0	0	18	7	74	4
Adventure	0	1	38	8	59	8	52	27	52	1	7	0	0	10	6	59	0
Animation	0	0	0	0	34	0	3	8	3	0	0	0	0	0	1	0	0
Biography	0	0	0	0	8	12	74	1	0	15	0	0	0	1	5	0	8
Comedy	0	0	0	0	32	26	100	19	13	0	14	8	2	2	71	5	2
Crime	0	0	0	0	0	0	97	0	2	2	7	1	0	29	0	1	1
Drama	0	0	0	0	0	0	48	13	32	28	35	13	4	52	98	28	15
Family	0	0	0	0	0	0	0	0	17	0	0	1	1	0	1	0	0
Fantasy	0	0	0	0	0	0	0	0	0	0	15	1	1	3	6	0	0
History	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Horror	0	0	0	0	0	0	0	0	0	0	11	0	1	22	2	16	0
Music	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0
Musical	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Mystery	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	16	0
Romance	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	1
Sci-Fi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
Thriller	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

In [31]:

```
# die obige Visualisierung führt auf natürliche Weise zu einer Heatmap:
plt.subplots(figsize=(15,15))
sns.heatmap(linksPivot, cmap=sns.color_palette("Greens", 42),annot=True)
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x1128ddb38>

Big Data Systems Performance: The Little Shop of Horrors

Jens Dittrich

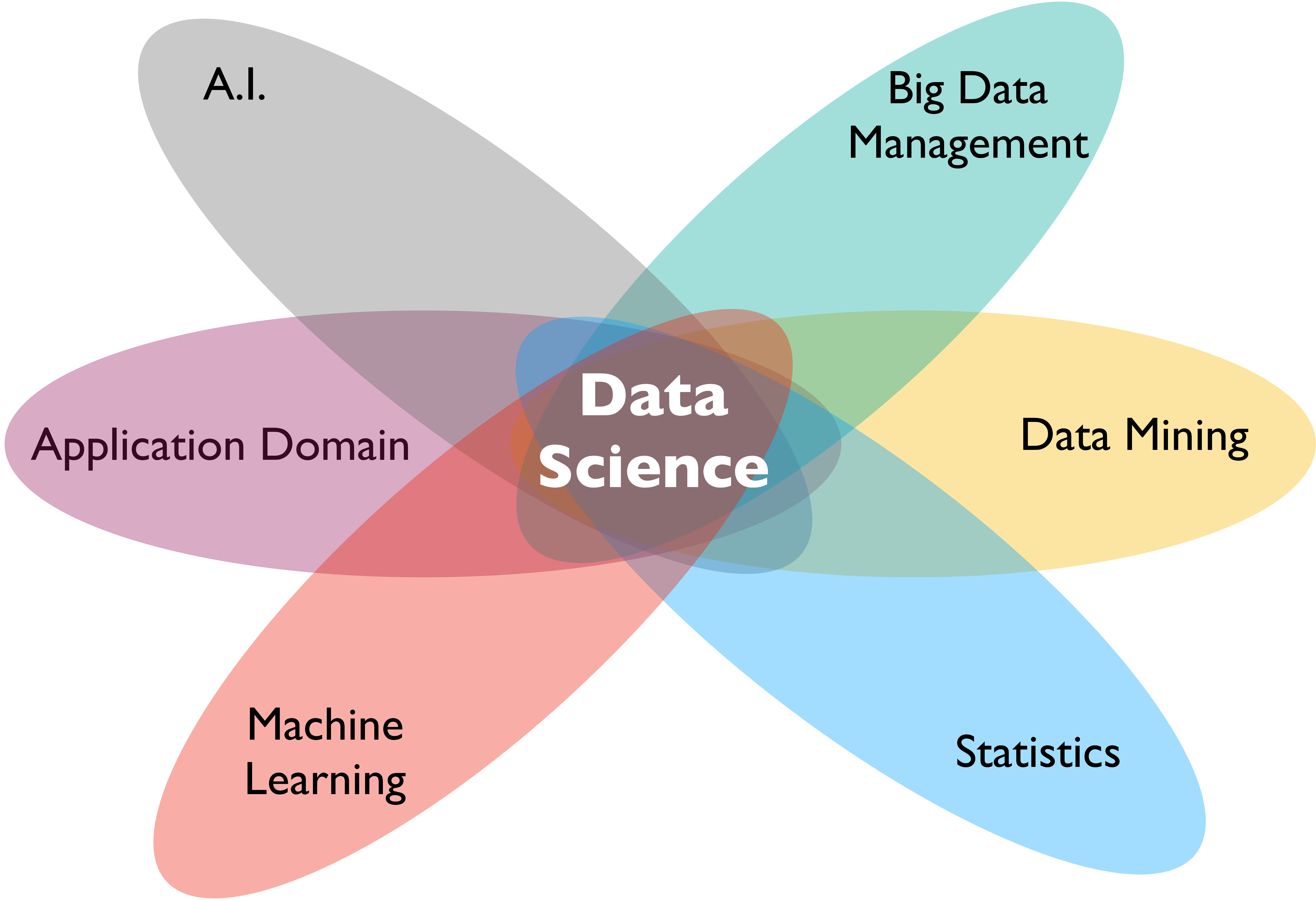
Saarland Informatics Campus

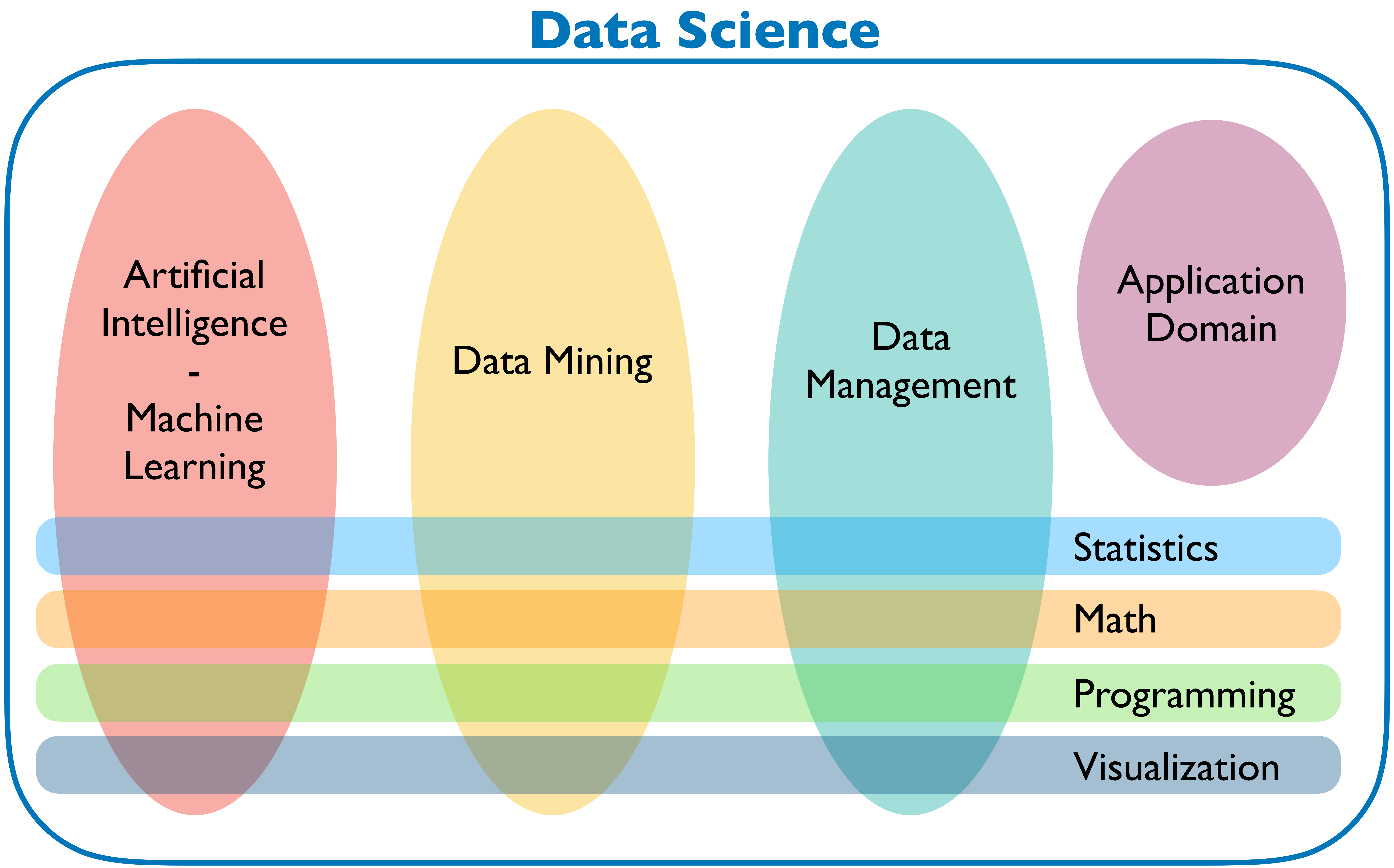
d:AI:mond.ai

twitter.com/jensdittrich

Backup

Data Science
vs
Machine Learning


A.I.
Big Data
Management
Application Domain
Data
Science
Data Mining
Machine
Learning
Statistics

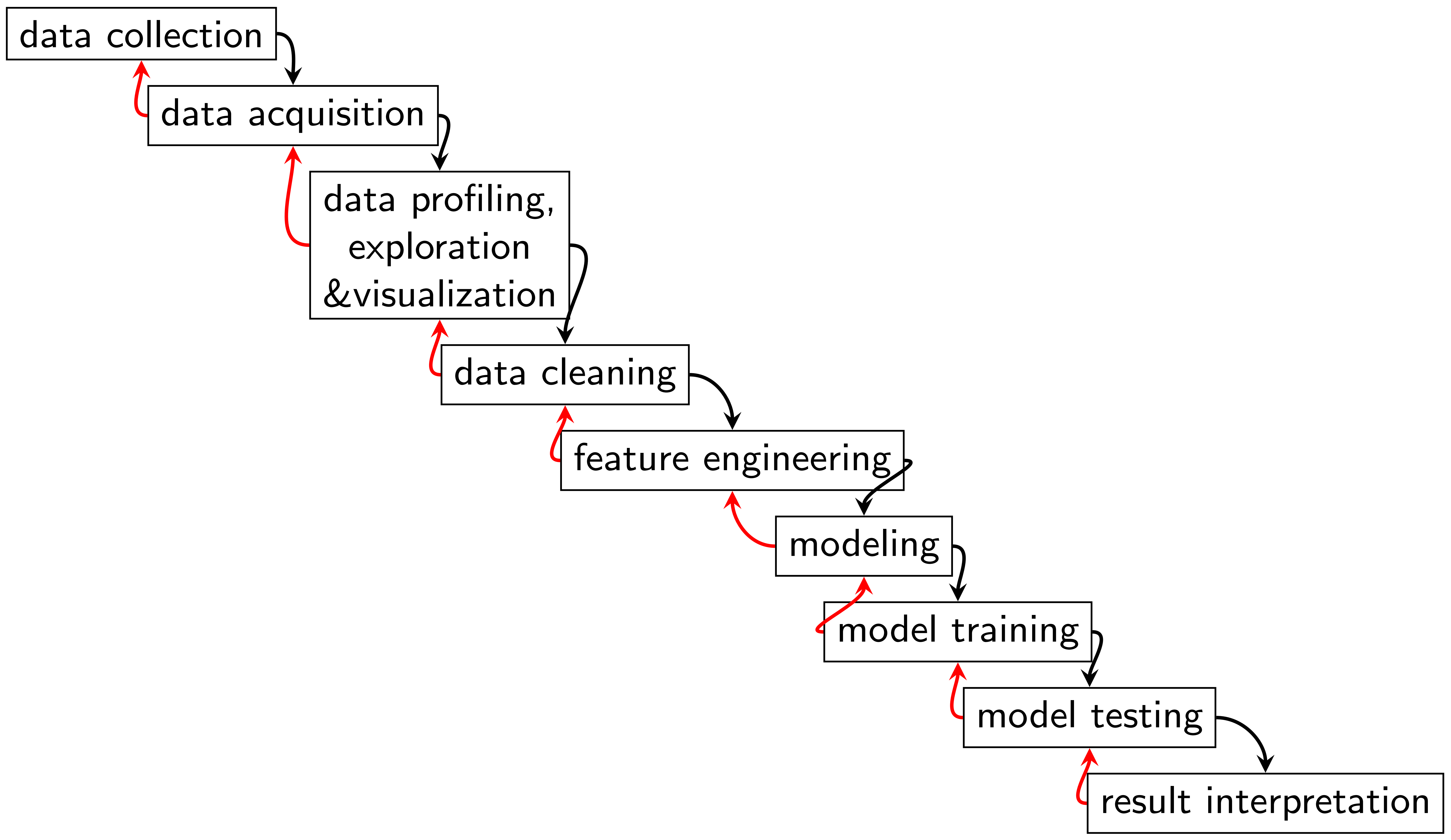

Data Science
Artificial Intelligence - Machine Learning
Data Mining
Data Management
Application Domain
Statistics
Math
Programming
Visualization

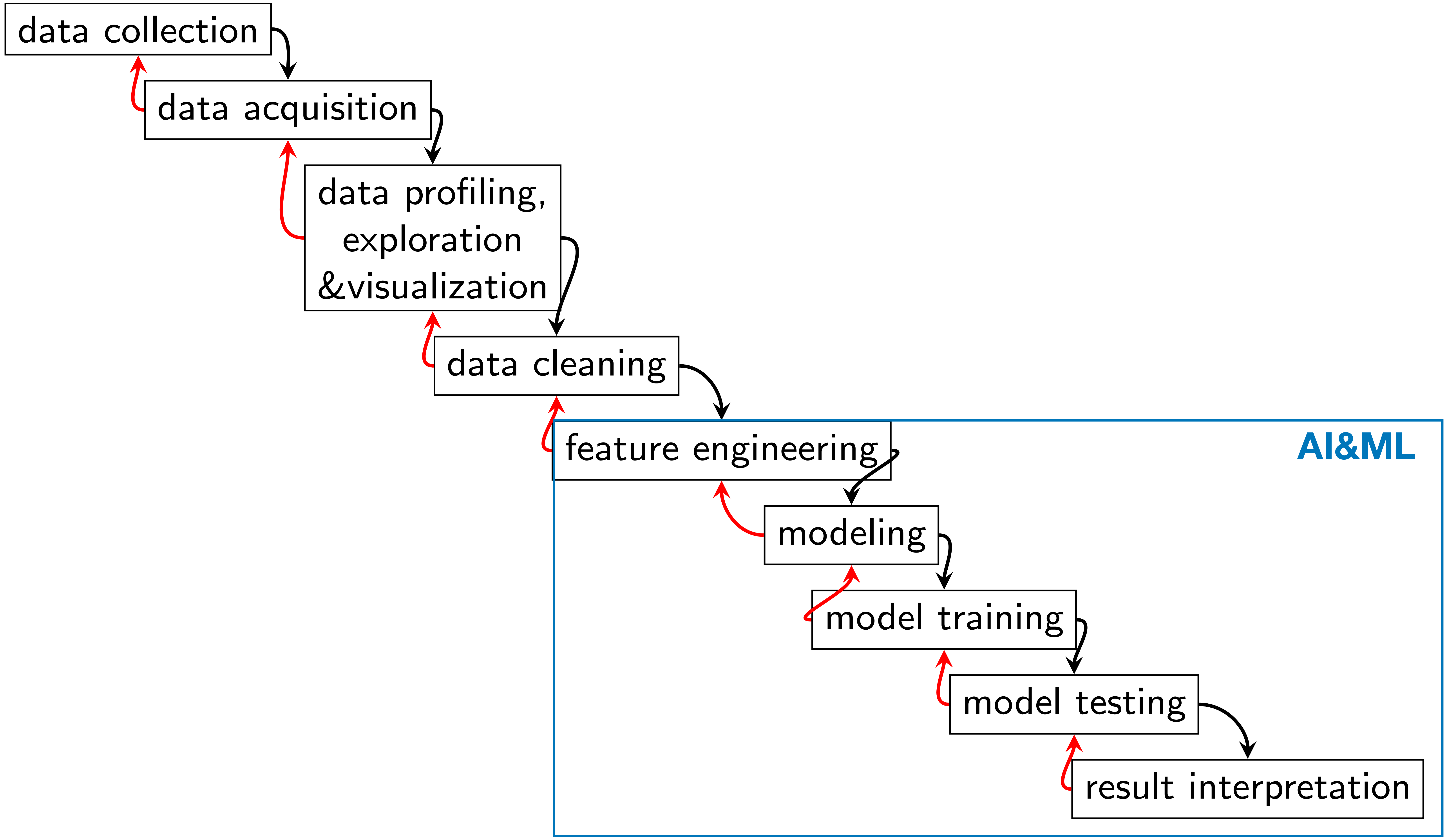
The Data Science Cake

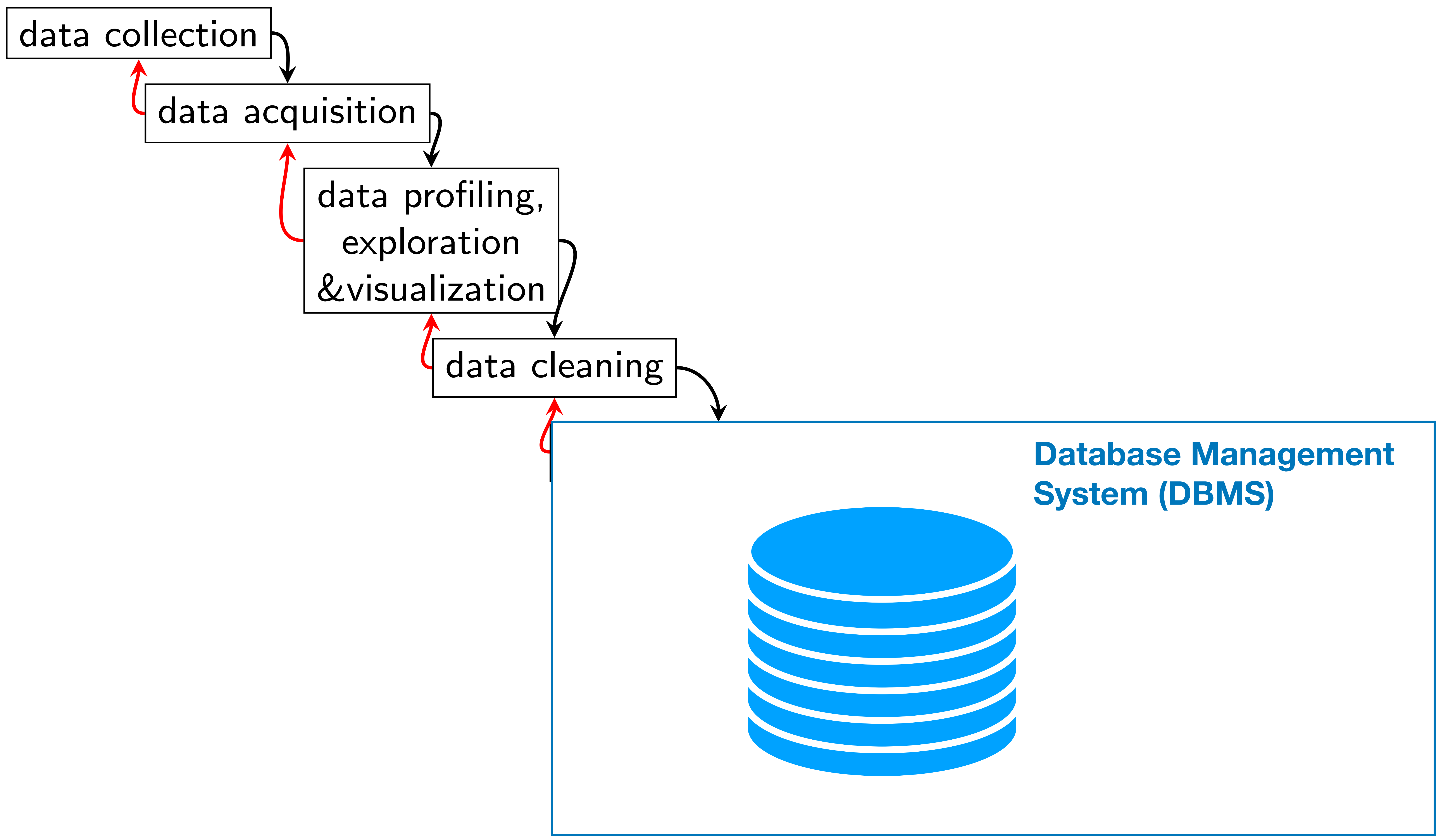
Ingredients:
50g statistics
120g linear algebra
200g programming
1kg visualisation
300g software
engineering

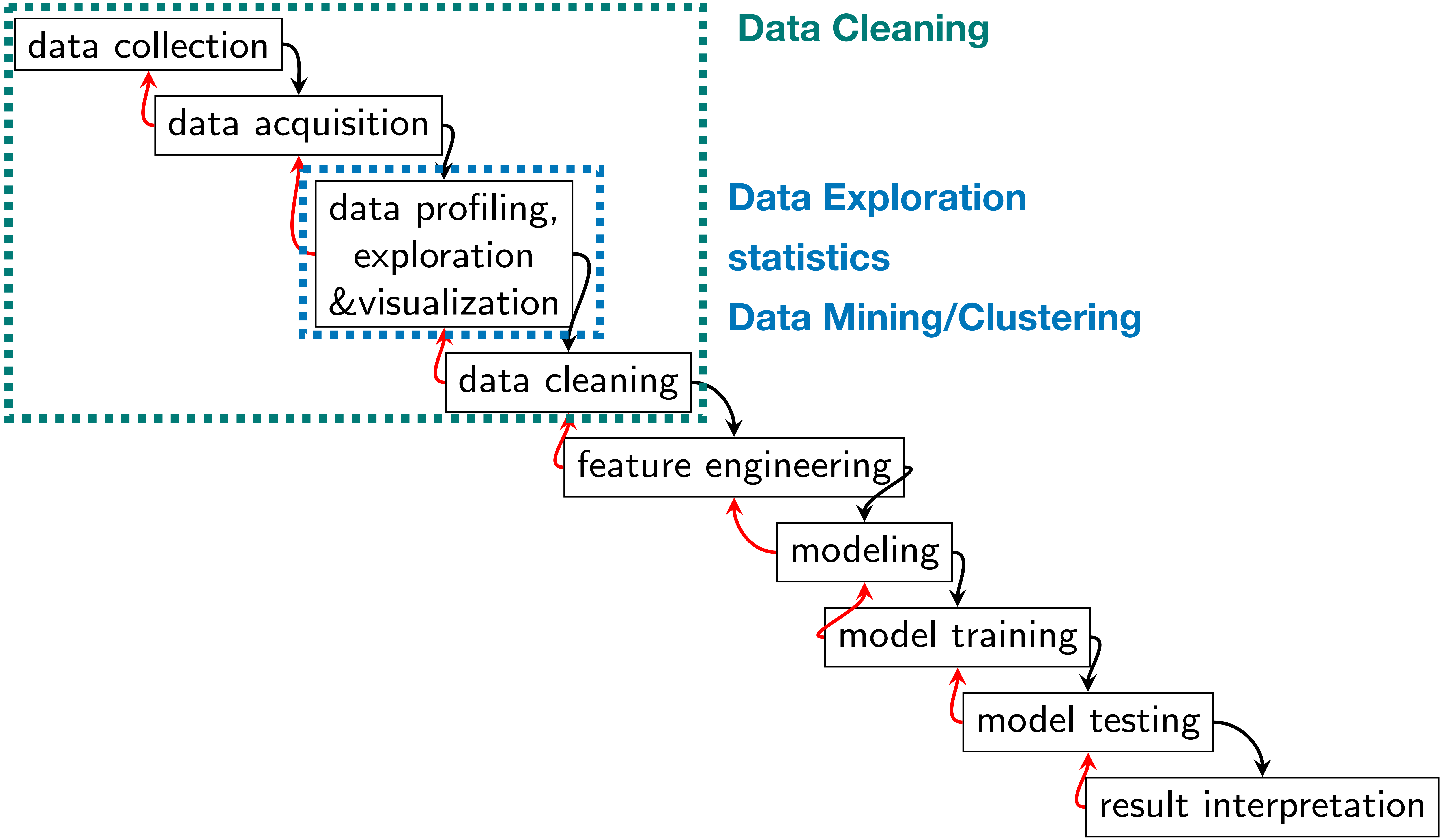
twitter.com/jensdittrich

Additional skills:
creativity
out of the box thinking
grit
team spirit

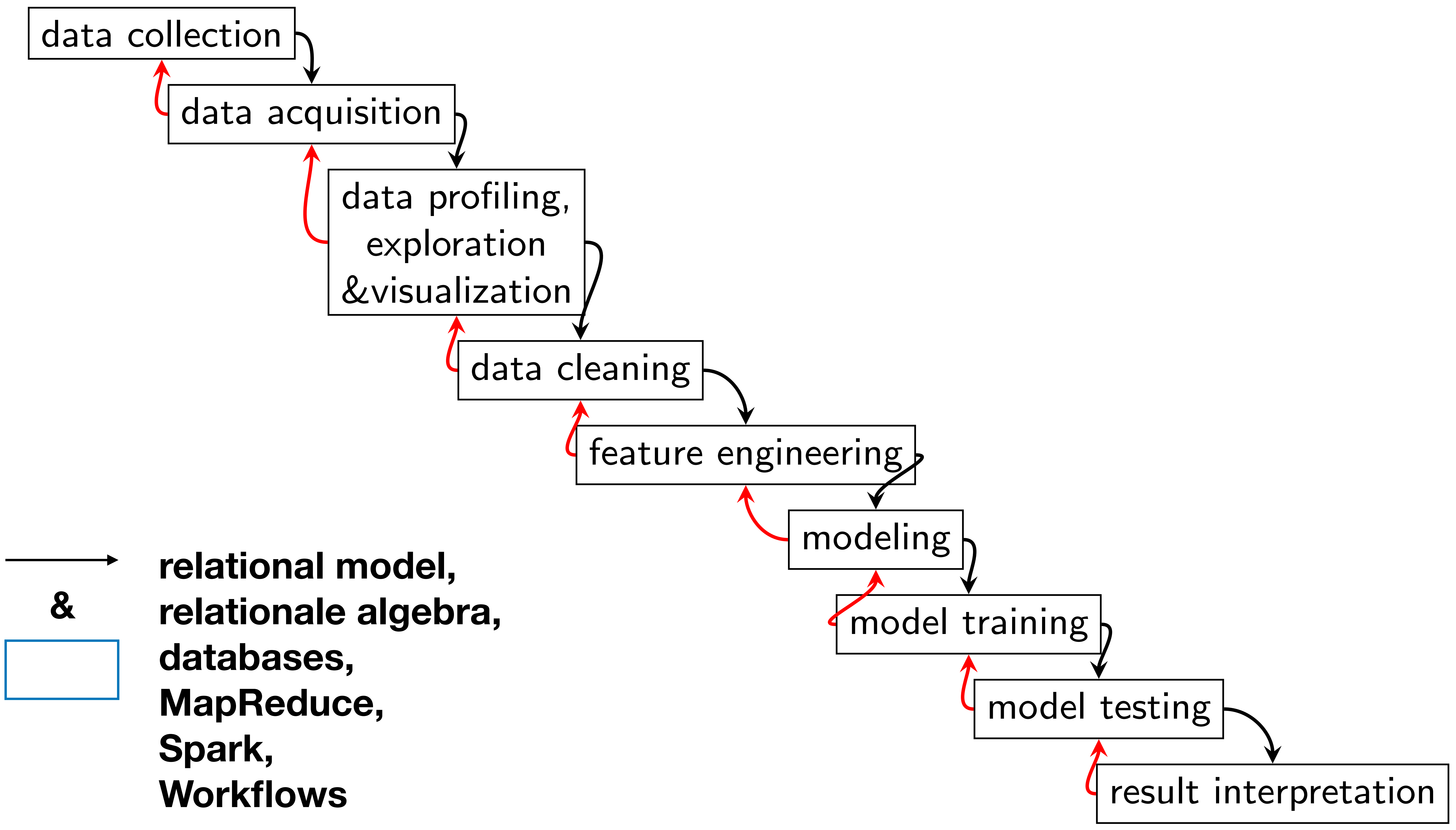

data collection
data acquisition
data profiling, exploration &visualization
data cleaning
feature engineering
modeling
model training
model testing
result interpretation

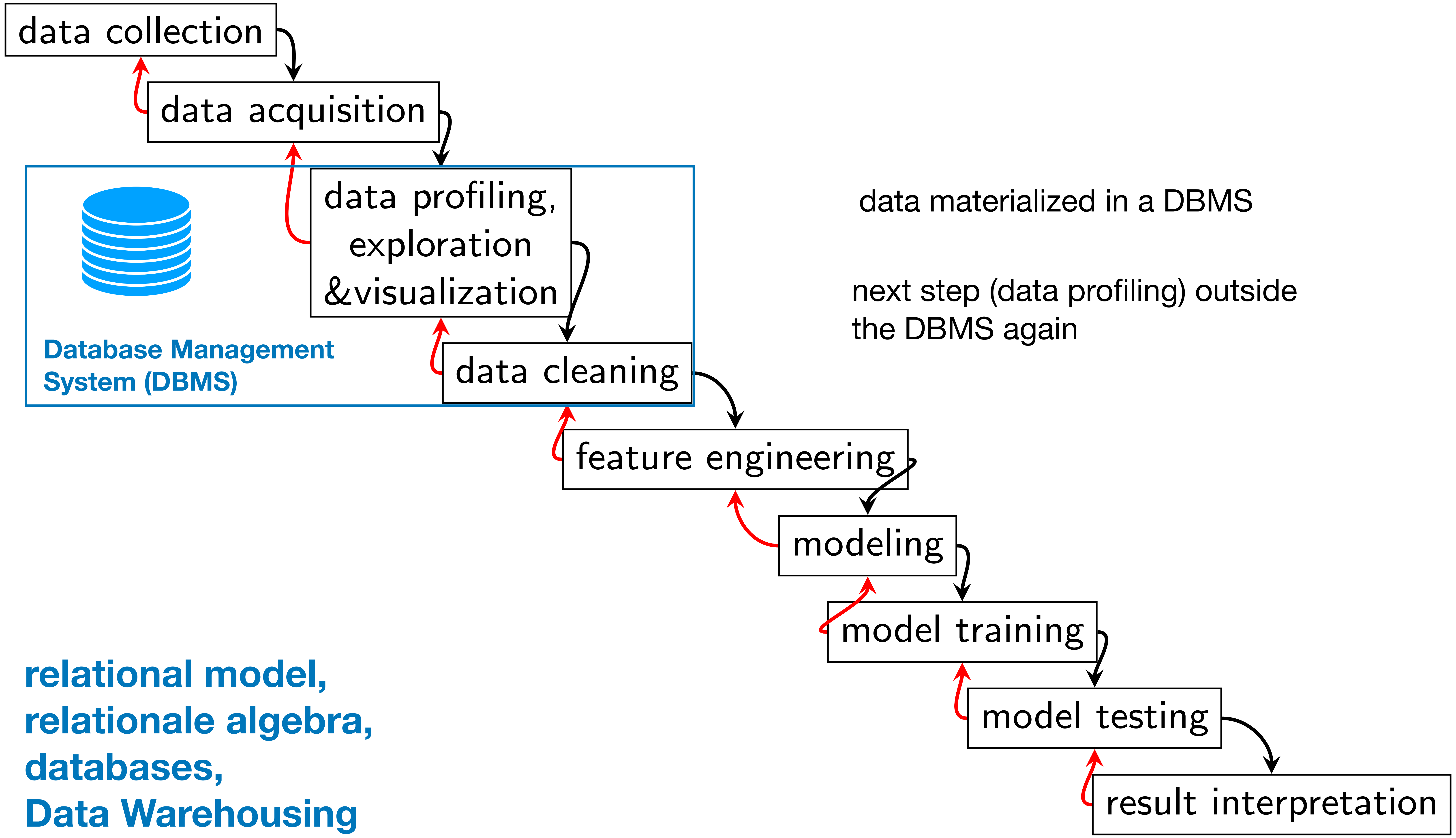

data collection
data acquisition
data profiling,
exploration
&visualization
data cleaning
feature engineering
AI&ML
modeling
model training
model testing
result interpretation


data collection
data acquisition
data profiling, exploration &visualization
data cleaning
Database Management System (DBMS)



Data Cleaning
data collection
data acquisition
Data Exploration
statistics
Data Mining/Clustering
data profiling,
exploration
&visualization
data cleaning
feature engineering
modeling
model training
model testing
result interpretation

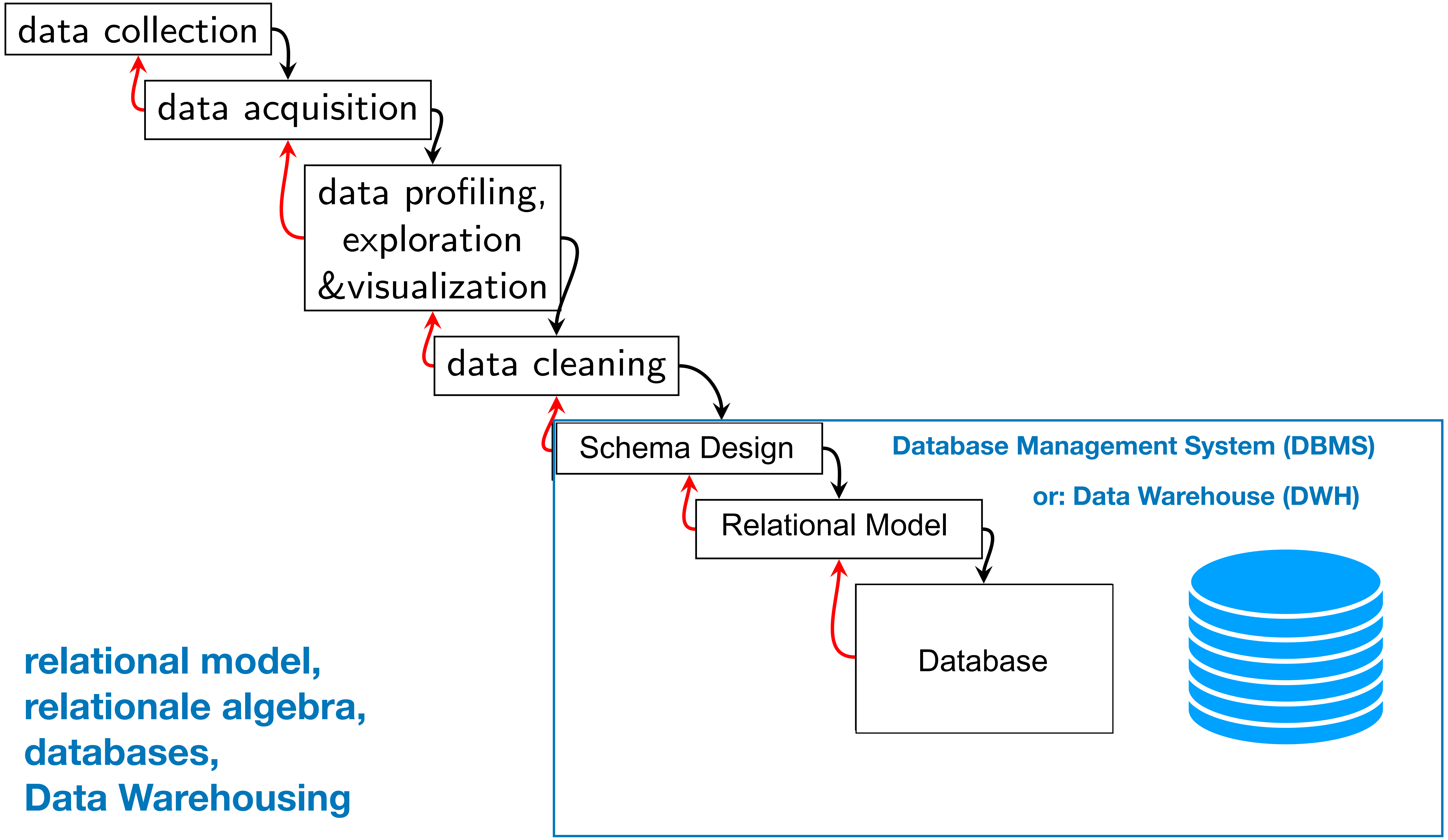

data collection
data acquisition
data profiling, exploration &visualization
data cleaning
feature engineering
modeling
model training
model testing
result interpretation
&
relational model,
relationale algebra,
databases,
MapReduce,
Spark,
Workflows



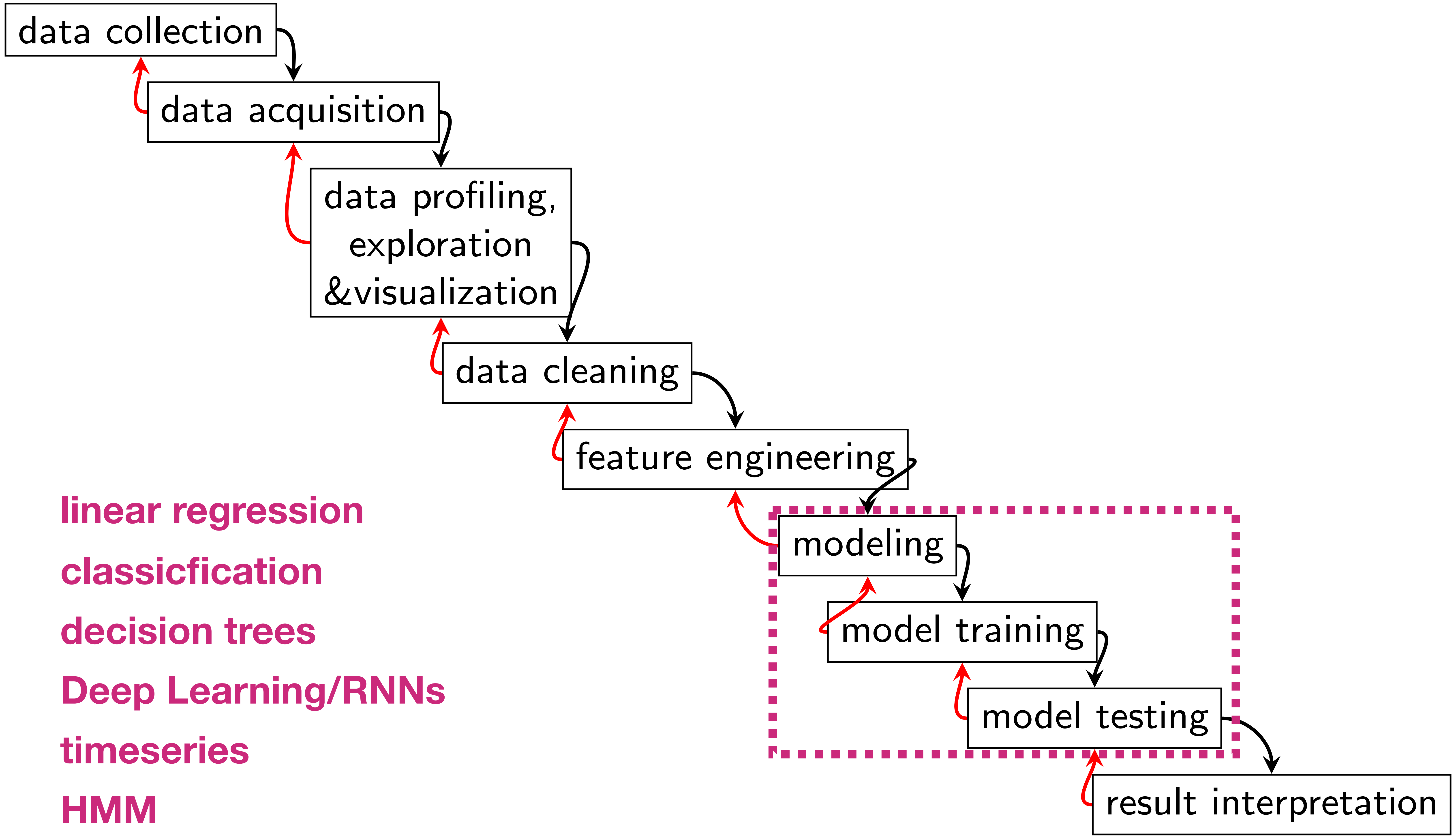
data materialized in a DBMS

next step (data profiling) outside the DBMS again

relational model,
relationale algebra,
databases,
Data Warehousing



relational model,
relationale algebra,
databases,
Data Warehousing



linear regression

classicfication

decision trees

Deep Learning/RNNs

timeseries

HMM

Big Data Systems Performance: The Little Shop of Horrors

Prof. Dr. Jens Dittrich

Manuscript for my PyData Talk in Berlin, July 8, 2018

Saarland Informatics Campus
https://bigdata.uni-saarland.de/

d:AI:mond.ai
http://daimond.ai/
https://twitter.com/jensdittrich

Abstract

The confusion around terms such as like NoSQL, Big Data, Data Science, SQL, and Data Lakes often creates more fog than clarity. However, clarity about the underlying technologies is crucial to designing the best technical solution in any field relying on huge amounts of data including `data science', machine learning, but also more traditional analytical systems such as data integration, data warehousing, reporting, and OLAP.

In my presentation, I will show that often at least three dimensions are cluttered and confused in discussions when it comes to data management: First, buzzwords (labels & terms); second, data design patterns (principles & best practices); and Third, software platforms (concrete implementations & frameworks).

Only by keeping these three dimensions apart, it is possible to create technically-sound architectures in the field of big data analytics.

I will show concrete examples, which through a simple redesign and wise choice of the right tools and technologies, run thereby up to 10,000 times faster. This in turn triggers tremendous savings in terms of development time, hardware costs, and maintenance effort.

____

Structure and Speaker Notes

title

CV-timeline, my background, add Python

---

confusion about terminology

people talking

tower of babel

Confusion

[mittelaltermarkt Bild: Wundermittel]
https://commons.wikimedia.org/wiki/File:Jan_Miel_Charlatan.jpg
https://commons.wikimedia.org/wiki/File:Giovanni_Domenico_Tiepolo_-
_The_Charlatan_(The_Tooth-Puller)_-_WGA22380.jpg

charlatan, quackery, quack

works as people believe it, make a big show and people will buy it

at the heart of (almost) any selling process

--

Confusion: good to sell stuff to laymen (aka "idiots")

Confusion example: selling a TV set, all kinds of labels, some of them are just bullshit

you probably don't understand most of it, but you get hypnotized, sounds so great

"this model even has HyperRayPlusRendering!"

"this model has 300 Hz rather than 250Hz" OMG! I will pay the 500 bucks extra

this is the 95% case of customers

ignore the 5% informed customers

however, this totally works for the vendor, as most customers don't have a clue

---

Confusion example: How would this look like in a data management "store"?

[show vendor selling databases (cylinders)]
"The leading NoSQL Data Lake solution in the cloud"

"Blockchain-enabled"

"powered by Cognitive Computing"

"AI-ready"

"IoT" -> "IIoT" -> "IDIoT"

"built on Lamda-Architecture"

How to have clarity in a discussion on data management:

1. mapping terms to meaning
[graphically]

confusing vs clear conversation

confusing: mapping to many concepts 1:n-relationship

clear: mapping to single (or few) concepts, 1:1-relationship

[show relationships visually]

semiotic triangle analogy

---

2. Confusion of dimensions

often at least three dimensions are cluttered and confused when it comes to data management:

First, fancy sounding buzzwords (labels & terms);

e.g. "big data", "data lake"

they either map to many things at once or are some sort of "hot air"

second, technical principles and patterns (concepts, best practices):

e.g. predicate pushdown

the relational model

relational algebra

data layouts, e.g. column vs row

cost-based optimization

compress to save I/O

these are the building blocks of any decent data management solution

third, software platforms (concrete implementations & frameworks).

Spark, MapReduce, MongoDB, PostgreSQL

they actually implement variants of these principles in software

Example 1: a technical principle:

"50 shades of predicate pushdown"

simple join example: with and without predicate pushdown

distributed system (called "query shipping")

sensor data

smart disks/SSDs (filter data on the device already)

satellites

etc.

--
Example 2: a technical principle:

the relational model is often confused with concrete implementations like tables/rows/columns

--
Example 3: a technical principle:
relational algebra: the mother of all query processing

you use Spark? Well, Spark is simply relational algebra++.

I don't have time here, if you have ever seen this symbols, **learn it, NOW**! OK, after my talk is better.

link to my youtube playlist in German
https://www.youtube.com/watch?v=8rOtQKwl4Ao&list=PLC4UZxBVGKtfArwVsT17oJdqkVYZMAjNP

----
all of these principles are entirely implementation/system/programming language independent

---

A War Story (let's say "I heard about this, from a colleague"):

I picked an extreme story here. It is exemplary for other things I have seen.

Client has a "big data" problem with sensor data. Already got a Hadoop cluster, put his data there, learned Spark, wrote some Scala/Spark/Parquet program to analyze stuff.

Let's go through this.

[visually mark terms like "Big data" and analyze stepwise]

"big data", does this mean "large" to him? How big is "large"? How large is big?

"Hadoop cluster" Hmmm, Hadoop is many things [show mapping 1:n], maybe he has a cluster where he installed HDFS to store the large data?

The 11th Commandment: If there is Big Data, there shall be Spark or MapReduce. [show Moses with this text on plate]

"some program" -> How much is this carving data management into spaghetti-code rather than a clearly layered, maintainable, and extensible architecture with even 40-year old query processing wisdom?

how much battle-proven DB-technology is used there? How much is he reinventing the wheel?

[cardboard car vs Tesla]

Let's look at the client's solution in more detail:

traces stored on HDFS, uses Parquet file format

software skims through traces using Scala, highly parallelized, queries are run on the cluster

---

But why not start with the original problem (rather than the existing solution)?:

we have traces of textual sensor data of the form

(timestamp, sensorid, value, fluff, more fluff)

[show textual snippet]

value may be of any type, may carry additional info as well

we have many of those traces from different machines

each trace contains data from thousands of sensors

timestamps not aligned to frequency

--

Desired: queries of the following type:
when is sensor<x> <operator> <value>?

e.g. sensor42 > 15.0

e.g. sensor15 > 15.0 AND sensor77 < 9.0

only few attributes in each query

--

currently stored as (timestamp, sensorid, value), i.e. like the data comes in

[example]

--

1.
our solution: why not store the triplets in this lexicographical order:

(sensorid, timestamp, value)

In its simplest form:

one file (dimension 3!) per sensorid, i.e. "sensor42.bin", "sensor15.bin"

inside each file: (timestamp, value)

this is an application of a fundamental data management principle (recall the principle dimension 2):

use column layouts for queries querying only few attributes

[example]

I/O-costs for this can be modelled analytically (in Python ;-))

I/O-costs translate to query response time in this case (we are I/O-bound!)

this alone: ~factor 500 improvement in I/O-time

PLUS: compressability is much better now!

another fundamental data management principle:

use compression to reduce bandwidth (rather than storage costs)

this can be modelled analytically (in Python ;-))

=> another factor 12!

in total factor 6000 better I/O-time!

ad this is not the end to it:

PLUS: data works better along the storage hierarchy (if queries are clustered on certain attributes => data locality increases) This was not possible with the old layout.

=> likelihood increases that some of the columns are entirely kept in main memory or on some SSD as an additional buffer

this effect is hard to estimate, depends heavily on query patterns

but, I estimate in total factor of at least 10,000 or more improvement

will see when the system is in production eventually possible "business effects":

- data could be stored in compressed format already when being created on the machine, again: factor 12 less storage, hardware/bandwidth savings along the entire data generation and processing pipeline

- there is no need to parallelize queries here => no need to use heavy-lifting with Spark, no fat clusters or similar;

QP is very lightweight!

- => QP can be done on very thin-clients even a smartphone

- only overall data sizes are the limit for the client; however, if users are interested in attribute subsets anyways, with our layout, they can easily pull those attribute subsets to their laptop/smartphone

---

Recap:
1. client did some "Big Data/Spark-Thingie" (dimension 1)

2. we went back to really understanding the client's original problem: what does he actually want to do?

3. we combined a couple of fundamental technical principles of data management, namely data layouts AND compression (dimension 2)

4. the synergies trigger I/O-time savings of a factor ~10,000, and storage savings of ~factor 12

5. all of this is totally software platform independent! (dimension 3)

PS: and there is even more you can do...

6. this kind of query performance enables new types of analytics, e.g. online anomaly detection, etc...

---

d:ai:mond

---

some recap here

Takeaways: three dimensions

do not confuse the three dimensions, unless you want to explicitly fool people (which I do not recommend in any situation)

Design solutions on dimension 2 only

Consider dimension 3 an afterthought

Realize that dimension 1 is solely about marketing

youtube-channel: https://youtube.com/user/jensdit

twitter: https://twitter.com/jensdittrich

end

---

backup slides