RAW — fast analysis on *all* kinds of data

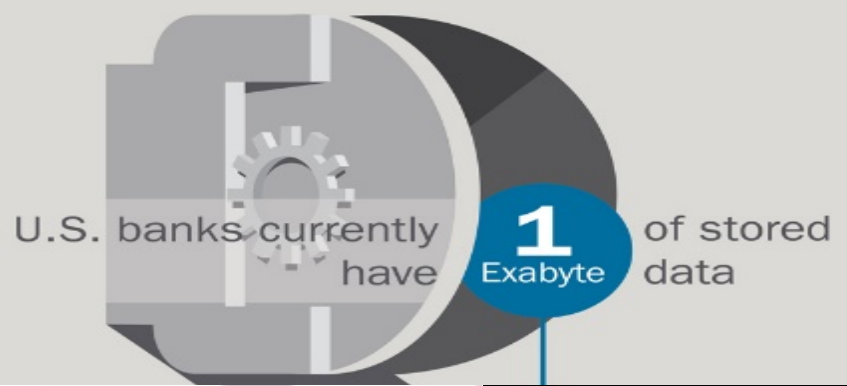
Anastasia Ailamaki
EPFL and RAW Labs SA

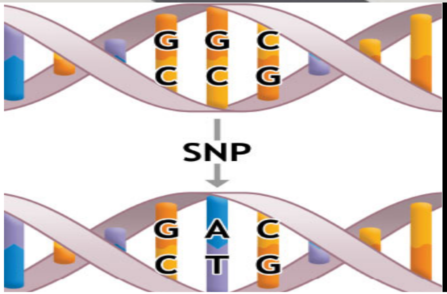
With **Manos Karpathiotakis, Stella Giannakopoulou, Matt Olma,** and the **EPFL DIAS lab**

most firms estimate that they are only analyzing 12% of the data that they already have

Forrester, 2014

- growing data
- growing heterogeneity
- data movement restrictions

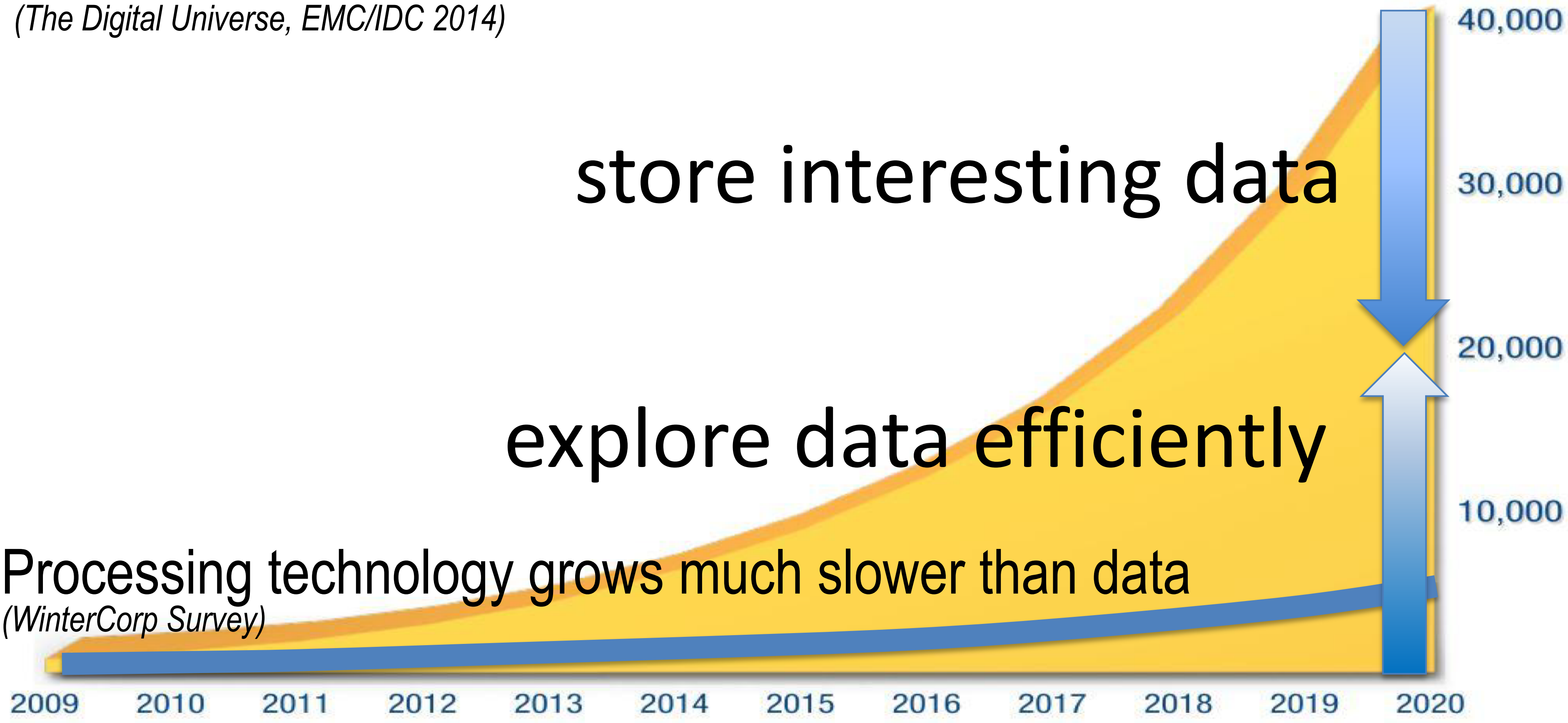




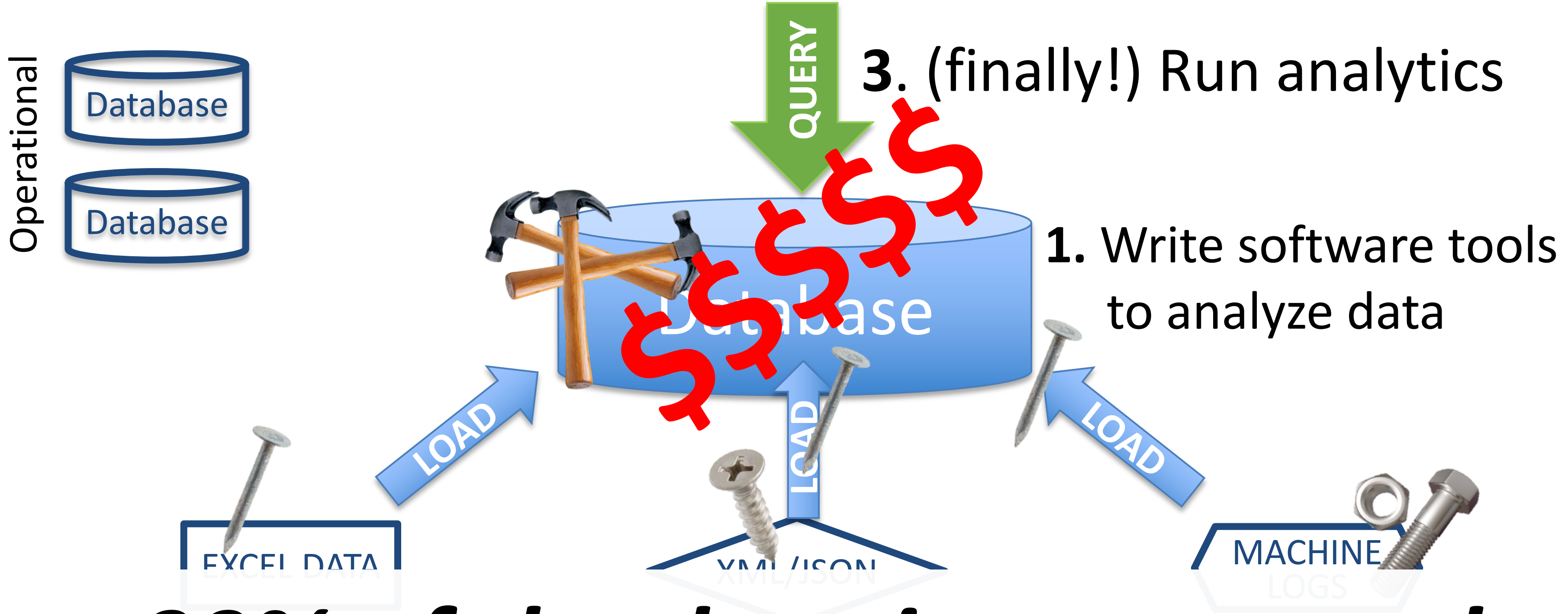
available data *impedes* business & scientific analytics

The Digital Universe: 50-fold Growth from the Beginning of 2010 to the End of 2020

(The Digital Universe, EMC/IDC 2014)

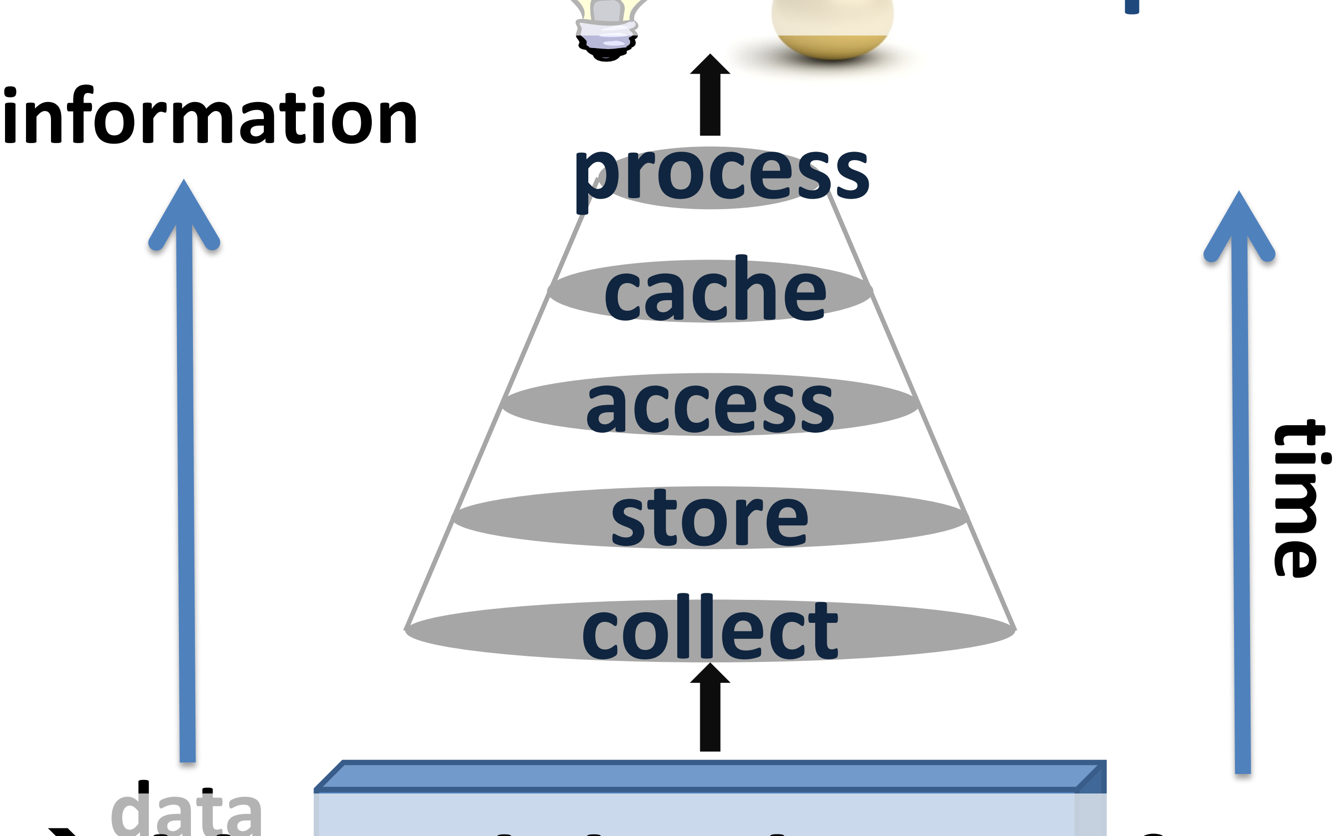


When you have a hammer…



90% of the data is never used.

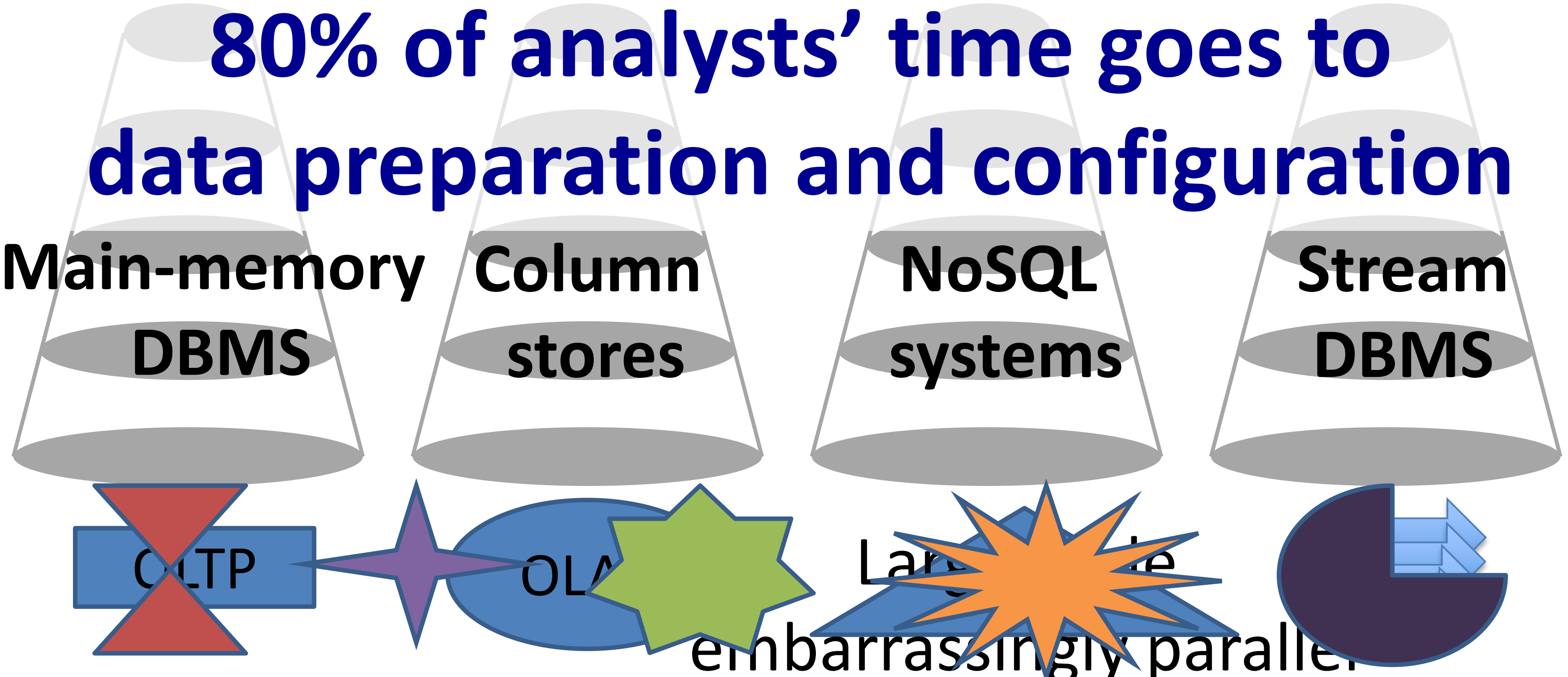
build database to run queries



→ bloated database software

new: one DB per app/data pair

80% of analysts' time goes to data preparation and configuration



Databases will be extinct
Data Analysis & Platforms
Storm
HPCC Systems
Apache Drill
GridGain
Dremel
Hortonworks
Zettaset
Operational
Versant JPA
MarkLogic
mobjecf
Multivalue database
U2.
REVELATION
northgate
QM
jBASE INTERNATIONAL
INFOBRIGHT
Hibari
HYPERTABLE
redis
Globals
Big Data search
Lucene
Solr
elasticsearch.
Data aggregation
Business Intel
talend
Palo
Data Mining
RAPID MINER
Social
Apache Kafka
ThinkUp
Corona
Multidimensional
GT.M
Grid Solutions
HAZELCAST
Galaxy
Document
mongoDB
CLUSTERPOINT
JasDB
SchemafreeDB
sisodb
EJDB
leveldb
Chordless
Tokyo Cabinet
Scalaris
SCALIEN
Project Voldemort
hamsterdb
RAPTORDB
FairCom
STSdb
HyperDex
IQLECT
OpenLDAP
loremap.net
Object databa s
db4objects
ZOPE
Magma
Sterling
Ptolisp
siaqodb
MORANTEX
HSS Database™
EyeDB
NData
Gremlin
dex
BrightstarDB
XML Databses
existdb
BASE
Qizx
sedna
x(nd)ce

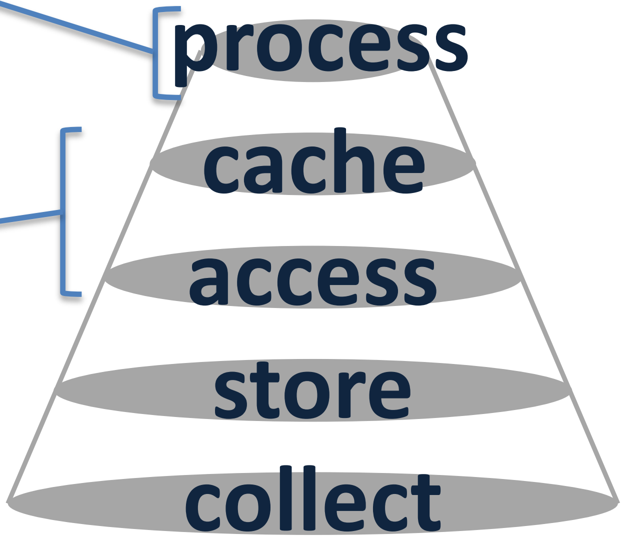
the way forward

- Data model:
  - Support variety (complex structured and unstructured data)
  - Col-store/Row-store are only two of many possible layouts
- Storage model:
  - Don’t store!
  - Run in situ and cache based on actual needs/usage
- Execution model:
  - Generate engine based on query, available caches, history

Fundamentally rethink DB stack

RAW — a lean and agile engine

- Adaptive Query Processing
  - A database per query and dataset
  - SQL++ to query and clean all data
- Adaptive data access
  - Tune database dynamically



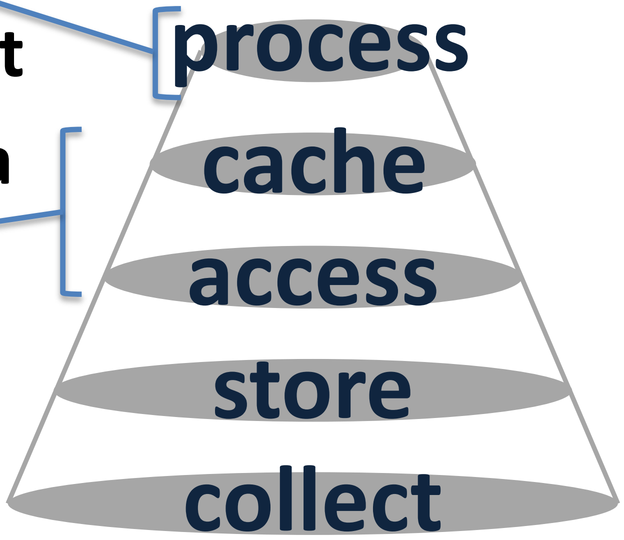
NOT DISCUSSED:

Caching – see our ADMS and VLDB talks

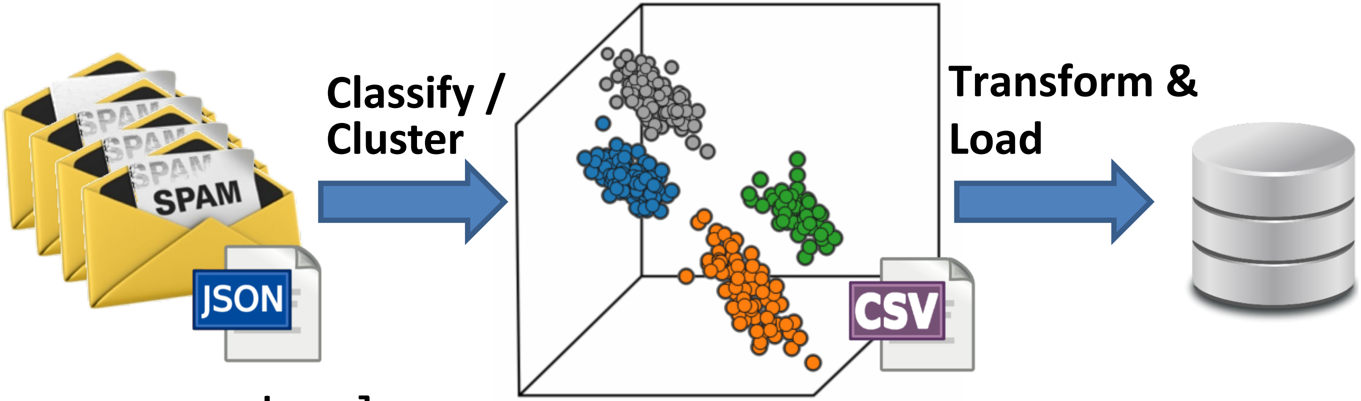
Query optimization, data model – future work ☺

RAW — a lean and agile engine

- **Adaptive Query Processing**
  - **A database per query and dataset**
  - **SQL++ to query and clean all data**
- Adaptive data access
  - Tune database dynamically



detecting active spambots



[Symantec data]

Flexibility

Ad-hoc queries
over diverse data formats

Performance

Fast queries
regardless of data format

fast queries on heterogeneous data

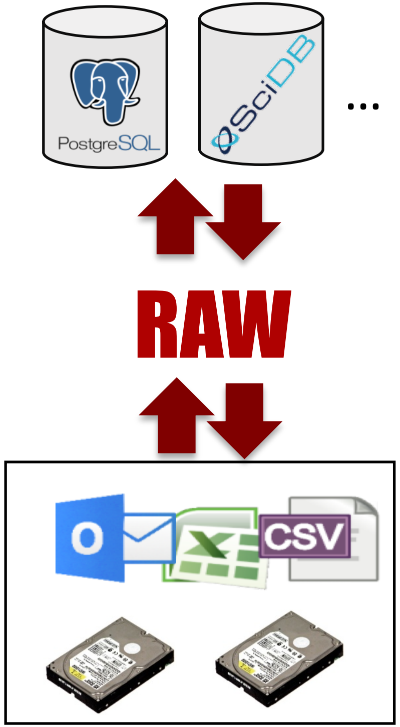
cannot load into a Database System!

- diverse formats
- legacy software
- privacy limitations
- data “owned” by one database

RAW: interface to raw data

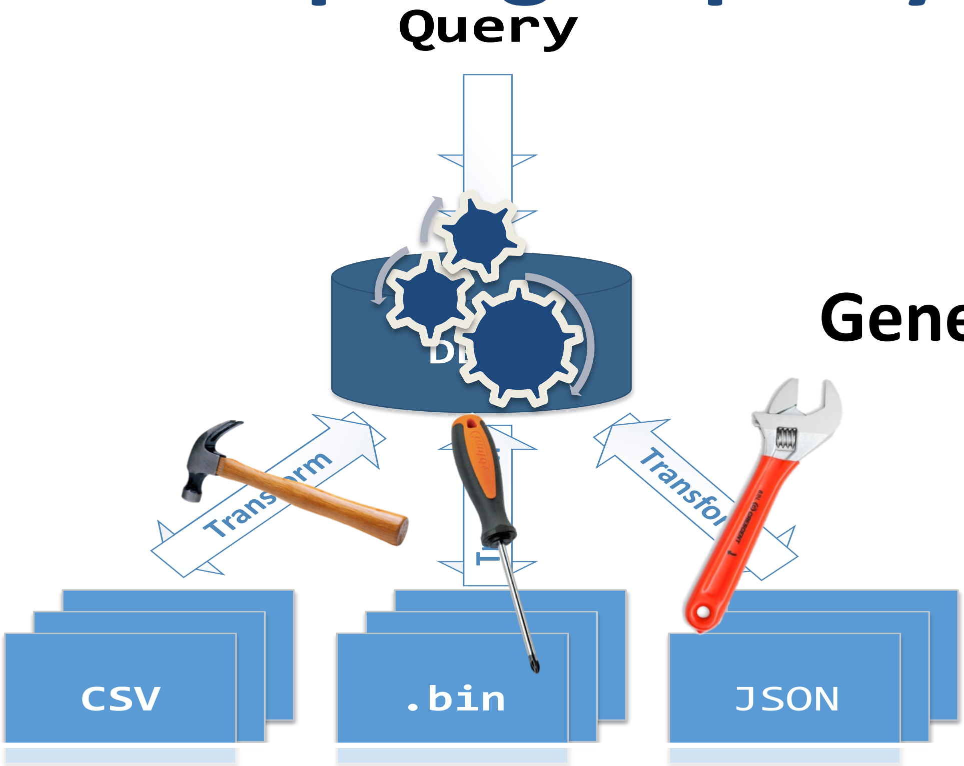
With extended SQL

code-generated engine



key: data virtualization

adapting a query engine to data



Generate plug-in per data source

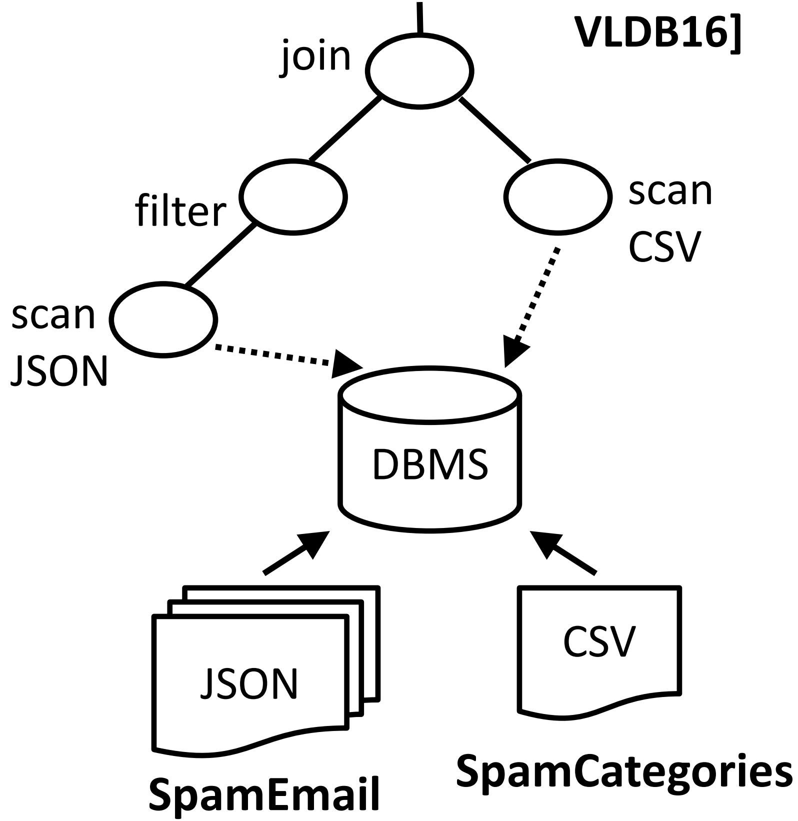
Treat each source as *native storage format*

Query original data formats, files, and scripts

How to build a just-in-time data base

[VLDB14,CIDR15, VLDB16]

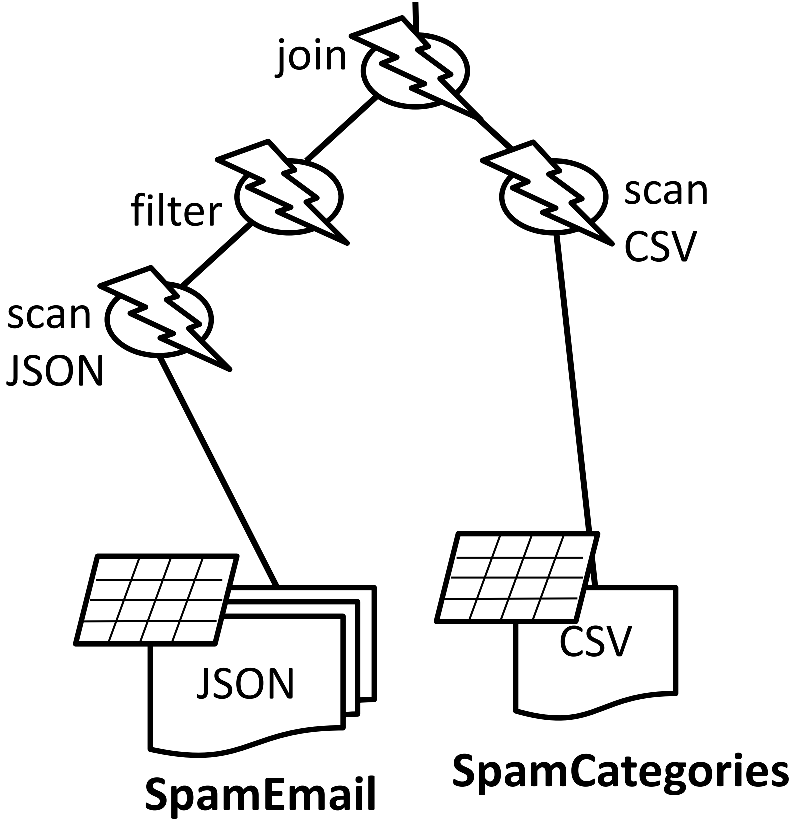
SELECT bot, country, …
FROM SpamEmail e, SpamCategories c
WHERE e.id == c.id AND
e.lang = ‘English’ AND …



How to build a just-in-time data base

SELECT bot, country, …
FROM SpamEmail e, SpamCategories c
WHERE e.id == c.id AND
e.lang = ‘English’ AND …

- Code Generate the Access Paths
- Code Generate the Query
- Build Position and Data Caches



Queries → Monoid comprehensions

Monoids:

- Abstraction for “aggregates” computation

*Fegaras [SIGMOD95, TODS 2000,…]

Monoid Comprehensions*:

- Operations between monoids

```
for {
      p <- Patients, r <- BrainRegions,
      p.id = r.id, r.amygdala.Vol > 0.2
} yield  bag p.age
```

Sum/Bag/List/Set/Top-K/…

Support multiple data models as input & output

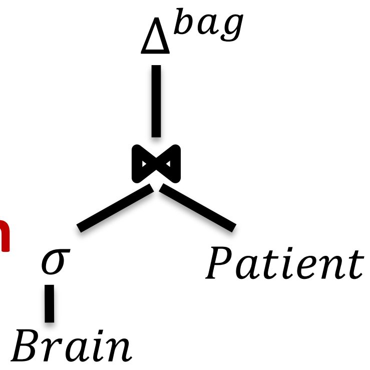
"SQL++" → Comprehensions → Algebra

```
SELECT r.age
FROM Patients p
JOIN BrainRegions r
ON (p.id = r.id)
WHERE r.amygdala.Vol > 0.2
```

Internal Calculus

```
for {
  p <- Patients,
  r <- BrainRegions,
  p.id = r.id,
  r.amygdala.Vol > 0.2
} yield bag r.age
```

if-else
record construction
function application
(nested) comprehension
...



Algebraic form amenable to relational optimizations

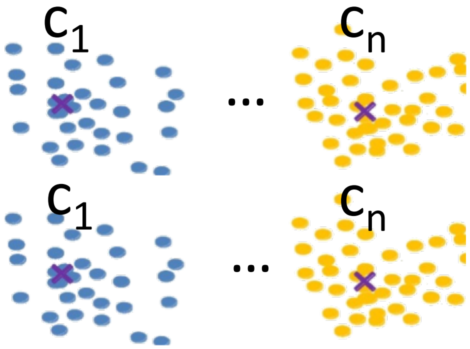
Data cleaning using monoid comprehensions

```
for(o←orders) yield list split(o.ship_date,"/")
```

```
dataGroup := for (o←orders)
        yield cluster(o.item,kmeans)

dictGroup := for (d←dict)
        yield cluster(d.item,kmeans)

for(d1←dataGroup,
    d2←dictGroup,
    d1.center = d2.center,
    similar(metric,d1.item,d2.item,θ))
yield group (d1.item)
```



Optimize cleaning operations holistically

SQL-like extensions for data cleaning

Functional Dependencies:

orderno, item $\longrightarrow$ quantity

```
SELECT o.orderno, o.item, *
FROM Orders o
FD((o.orderno, o.item), o.quantity)
```

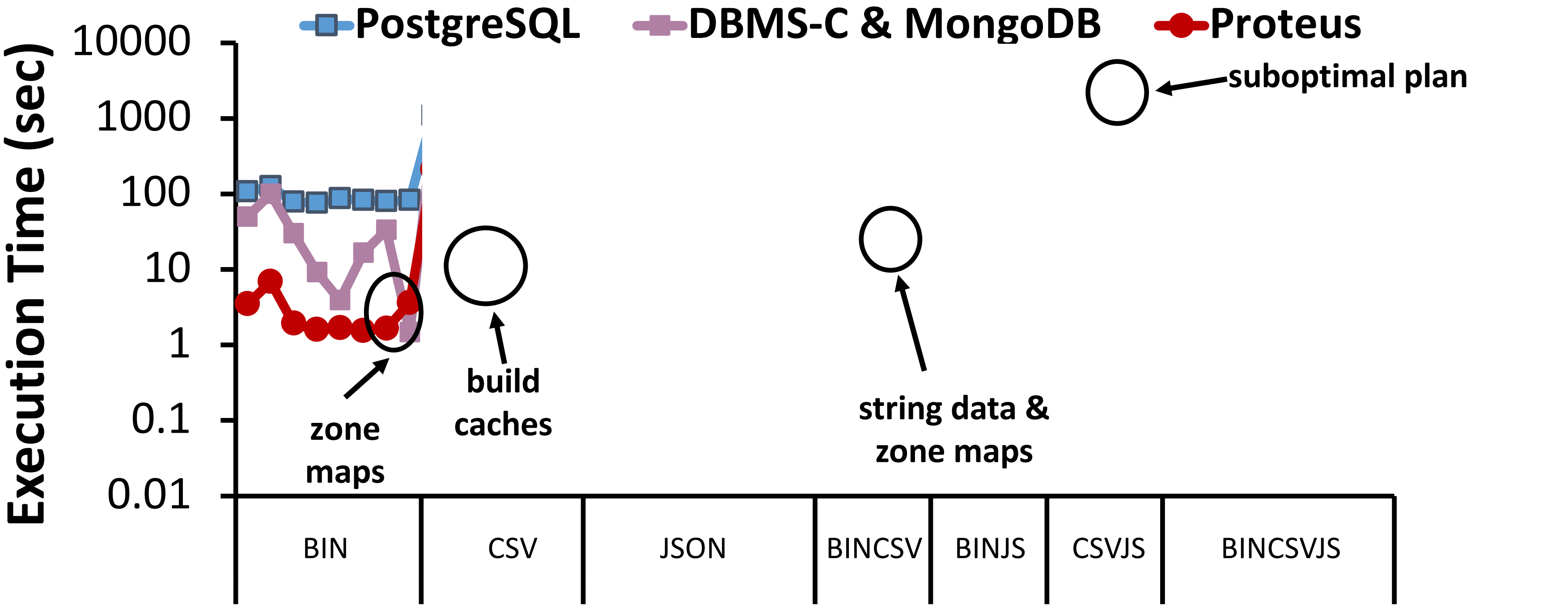
Data Deduplication:

```
SELECT <projections>
FROM <dataset>
DEDUP([<metric>,] [<theta>,] <attributes>)
```

Mask complex comprehension syntax

Symantec spam email analysis

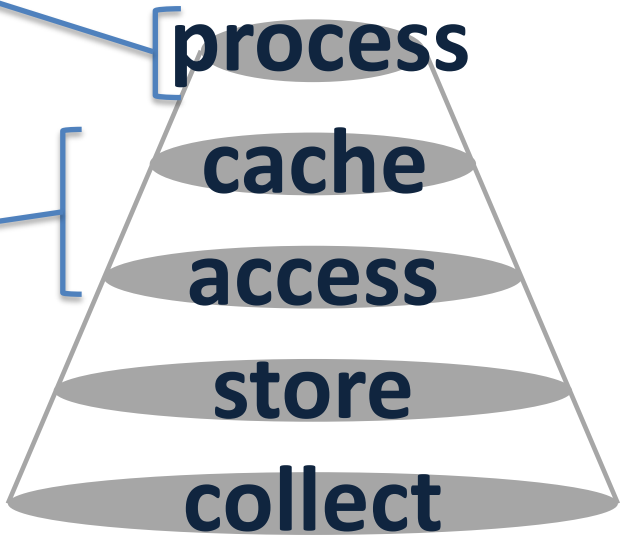
95GB Binary - 22GB JSON – 22GB CSV
50 queries



Flexible and fast by specializing & adapting

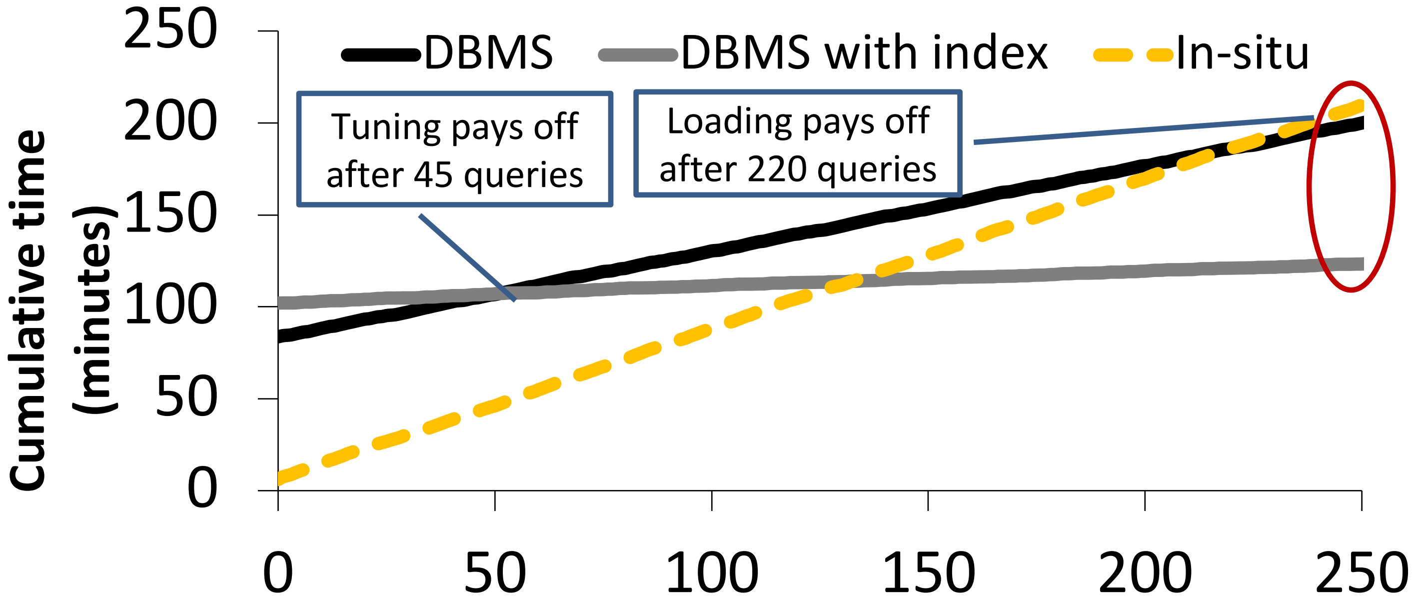
RAW — a lean and agile engine

- Adaptive Query Processing
  - A database per query and dataset
  - SQL++ to query and clean all data
- **Adaptive data access**
  - **Tune database dynamically**



Querying raw data w/o index: Diminishing returns

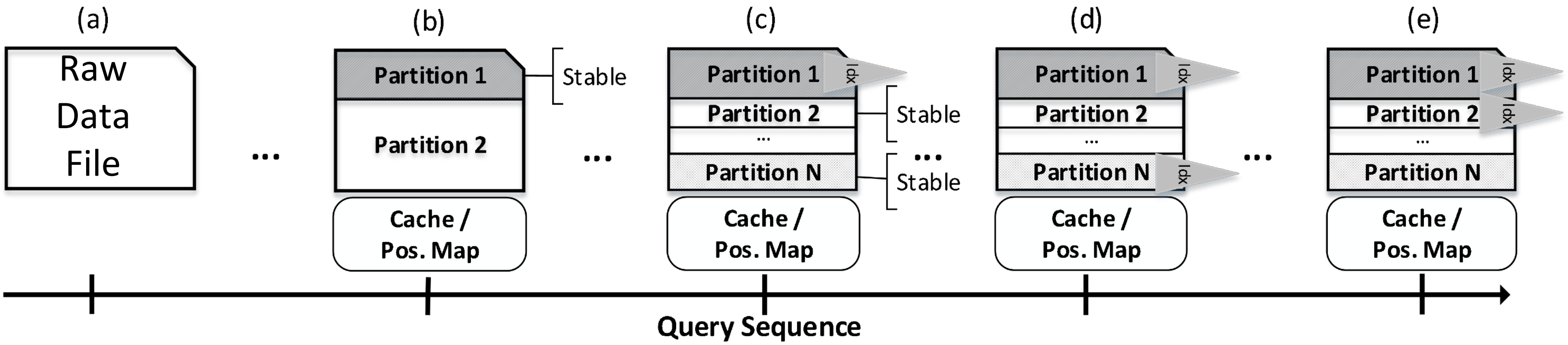
60GB smart meter data,
selectivity 1%,
128GB RAM, 1 thread



Scanning all data is slow

Indexing/tuning non-trivial for ad hoc data

invest in popular data subsets



Refine partitions over the data => Skip if useless for query

Tune indexes over popular partitions => Minimize data accesses

adapt to data: logical partitioning

Enable data skipping

Fine-grained access path selection

Capture implicit clustering

Iteratively partition dataset

Homogeneous

Query-based

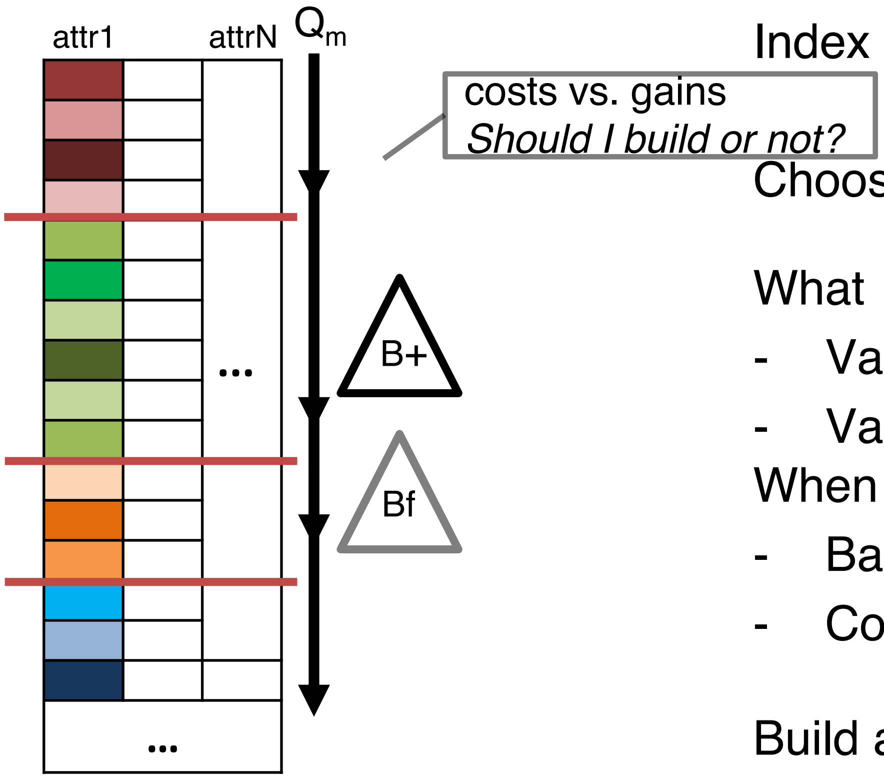
Increase disjointness: Reduce distinct values

Remove tails: Reduce excess kurtosis

1) Collect data statistics at runtime
2) Calculate number of sub-partitions

Set the "ground" for reducing data access

adapt to queries: index tuning



Index tuning on partition level

Choose what & when to build

What

- Value-Existence (i.e., Bloom filters)
- Value-Position (i.e., B+ Trees)

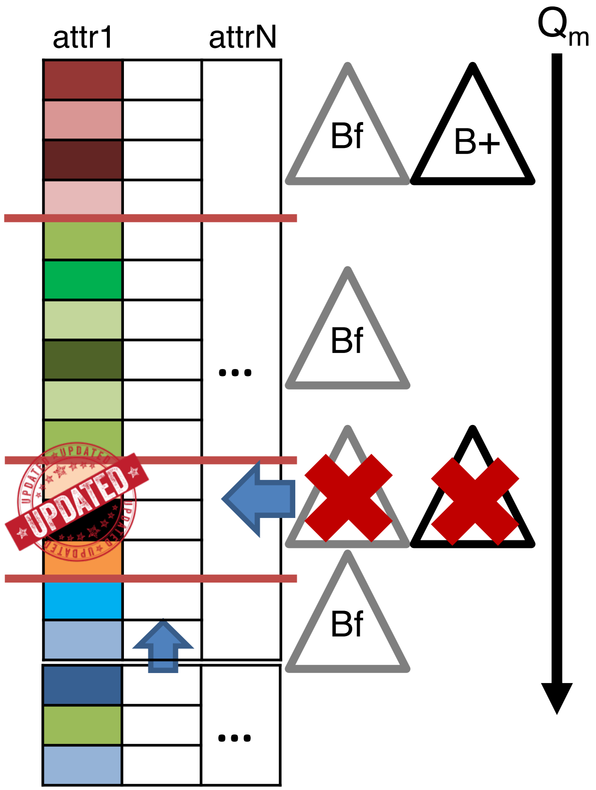
When

- Based on randomized algorithm
- Cost of scan vs. cost of build + gain

Build and drop based on budget

Maximize gain: build cost vs performance

append & in-place updates



Store partition state

- Calculate hash value (MD5)

Monitor file for modifications

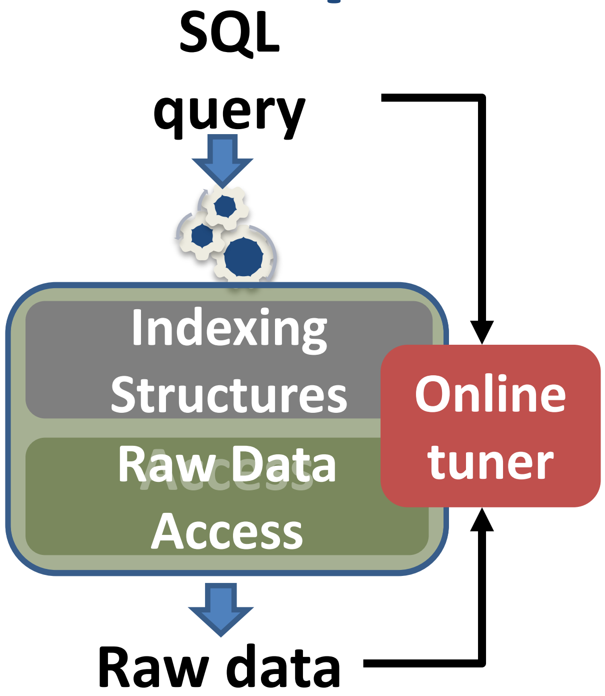
Recognize updated partitions

Fix modified partitions

- Drop/Re-build cache/index

Minimize update overhead

Slalom: adaptive indexing over raw data



Incremental logical partitioning

- Based on data distribution

Adaptive partition indexing

- Based on access patterns

Monitors data for updates

- Updates data structures

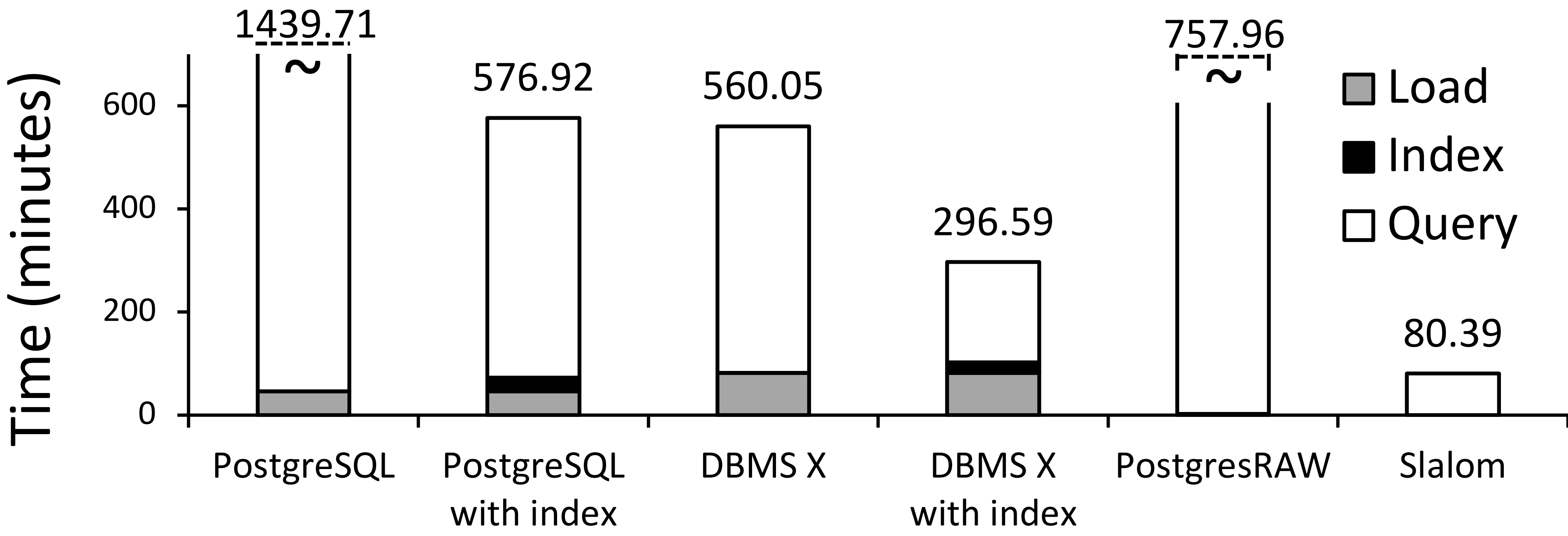
Combining online tuning with adaptive indexing

Adapt data access to queries and data at runtime

from raw data to results

59GB uniform dataset, 128GB RAM, cold caches

1000 point & range queries interchange on 2 attributes, sel: 0.5%-5%



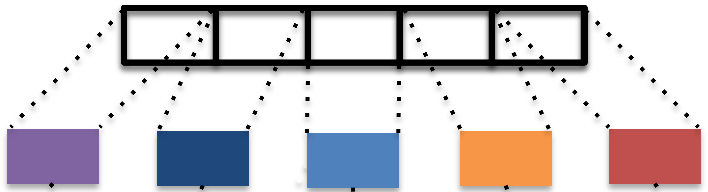
In-situ adaptive indexing achieves interactive access

what we learned

- currently data management cost **grows with data owned**
- **impossible** to pre-cook a database system suitable for all data
- from *manual ingestion* to ***automatic adaptation***: rethinking DB stack with just-in-time queries and storage

How RAW works

1. Ask a question

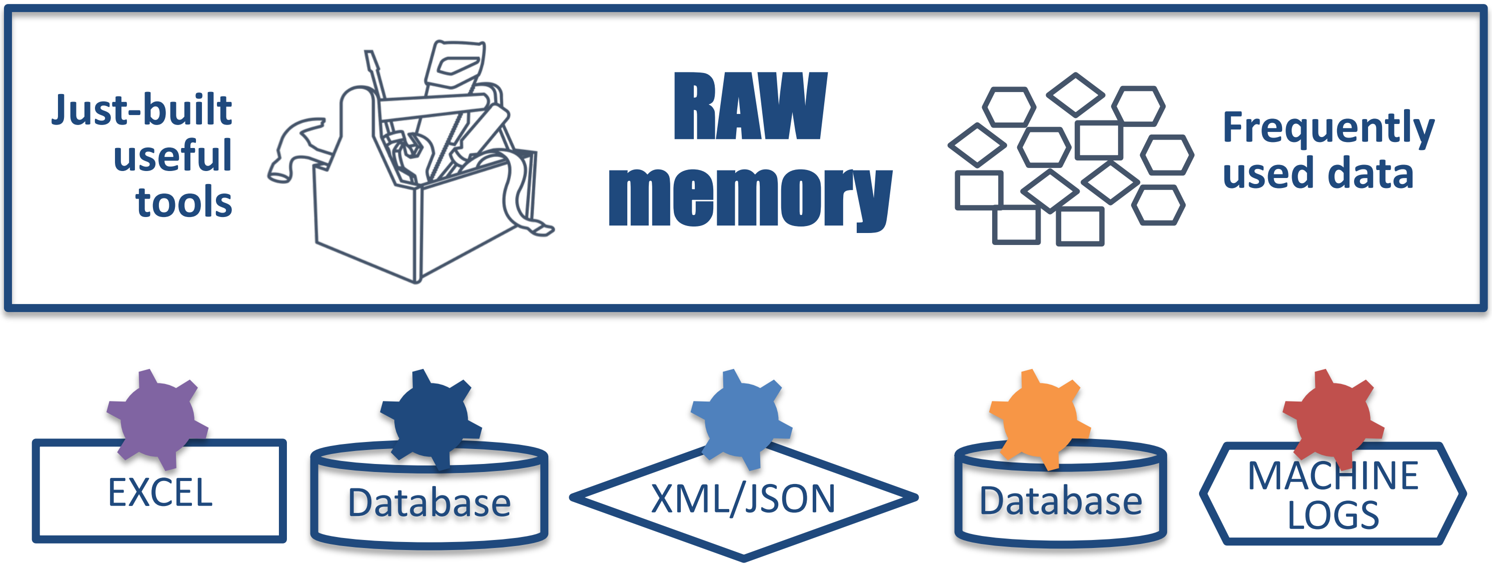


2. Generate the needed software tools

3. Discover interesting data

Data is accessed and integrated in real time

Why RAW is fast



As queries run, RAW remembers information on data accessed and generated code. Its "database" is only the useful data.

RAW

Just ask.

dias.epfl.ch

raw-labs.com