# Automated Feature Extraction with Machine Learning and Image Processing

Prof. Dr. Stefan Bosse

University of Siegen - Dept. Maschinenbau
University of Koblenz - Dept. Computer Science

# Data Storage and Aggregation

Representation of Data

Storage of Data

Access of data

# Data Storage

In general, data and their values can be stored in/on:

- **File Systems**, such as FAT32/VFAT, Ext 3, NTFS, ISO, ....
  - Organization by hierarchical directories and files
- **Databases**, such as SQL, NoSQL, ...
  - Flat tables only
- **Cloud Storage**, such as Seafile
- **Files**, linear storage container, structured, with, e.g., numpy or HDF(5)

But: The question is not where to store the data, the question is how to organize and access the data!

## File System

- A file system is composed of:
  - Files ⇒ data container (linear memory model)
  - Directories ⇒ reference tables assigning files or other directory references to names
- A file system organizes data by a directory (folder) tree:
  - A directory is a node in an ordered graph
  - There is a root directory
  - Children of a directory can be leaves (files) or deeper directories
  - Each file and directory is associated with a name
  - File system tree iterations are referenced by paths

Besides the organization structure, a file system can provide data structures and block level organization of data used for the storage on hardware devices

[https://www3.nd.edu/pbui/teaching/cse.30341.fa18/project06.html]

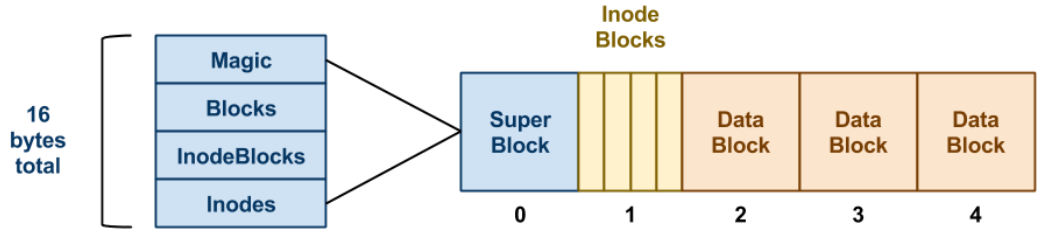



Fig. 1. Simple file system layout (linear file model)

## Segments

- Magic: The first field is always the MAGIC_NUMBER or 0xf0f03410. The format routine places this number into the very first bytes of the super-block as a sort of file system "signature". When the file system is mounted, the OS looks for this magic number. If it is correct, then the disk is assumed to contain a valid file system. If some other number is present, then the mount fails, perhaps because the disk is not formatted or contains some other kind of data.
- Blocks: The second field is the total number of blocks, which should be the same as the number of blocks on the disk.
- InodeBlocks: The third field is the number of blocks set aside for storing inodes. The format routine is responsible for choosing this value, which should always be 10% of the Blocks, rounding up.
- Inodes: The fourth field is the total number of inodes in those inode blocks.

[https://www3.nd.edu/pbui/teaching/cse.30341.fa18/project06.html]

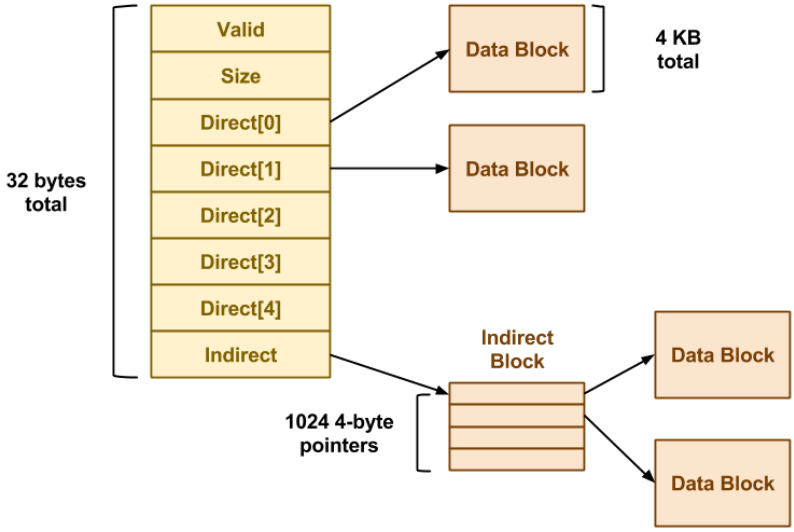



Fig. 2. I-nodes and block references

# Data Types

- Raw data characterized by:
  - Dimension
  - Size
  - Aggregates
  - Data types
- Vector and time signal data
- Matrix and tensor data (arrays)
- Image data (Gray / intensity level, RGBA)
- Data tables (data frames) with rows of records (columns)

# Databases

## SQL

- SQL databases organize data in tables.
  - A table is organized in rows and columns
  - A table is part of a database
  - Multiple databases can co-exist and handled by one database server
- Data types:
  - Number (numeric)
  - Text
  - Binary data (blob)

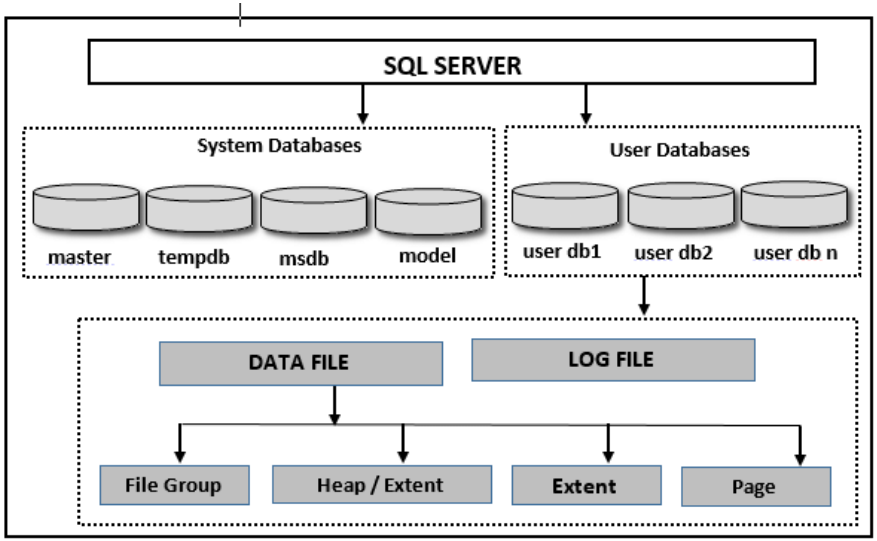



Fig. 3. Basic SQL Server architecture

## Page

The page is the fundamental unit of data storage and internally, SQL server organizes and stores data in smaller units known as pages.

[Murugesan et al., International Journal of Applied Engineering Research, 2015]

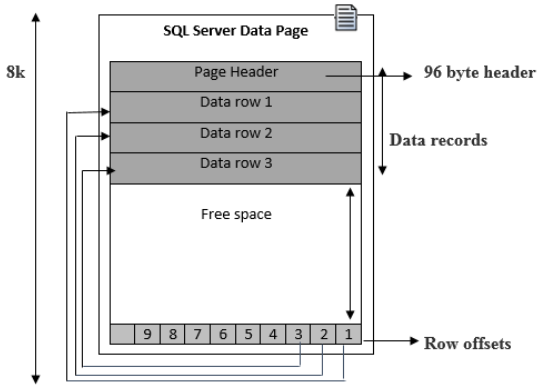



Fig. 4. Data Pages storing tables - Schematic Diagram

## Tables

[]

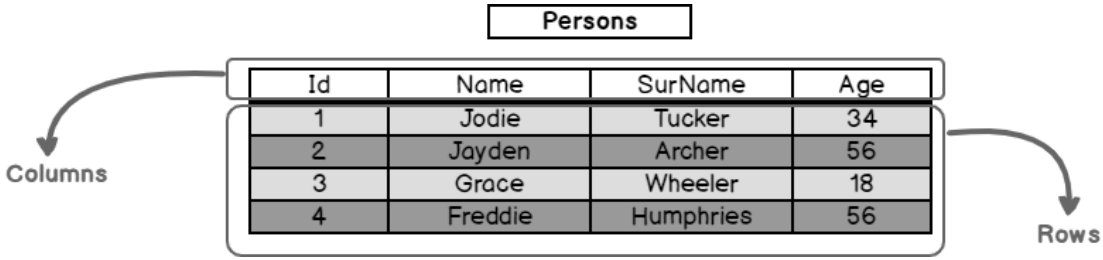



Fig. 5. SQL table structure with rows and columns

## Operations

### Create

Create a new empty table with a specific schema (type signature of the columns. A table is referenced by a (data base) unique name.

### Insert

Insert rows into an existing table.

### Update

Update fields or entire existing rows

### Select

Select fields (columns) or entire rows based on patterns

Tables cannot be nested. The data base table space is flat! But specific tables can be used to reference other tables (like I-nodes, directories in file systems, or sections in HDF structures)

Tables cannot be nested. The data base table space is flat! But specific tables can be used to reference other tables (like I-nodes, directories in file systems, or sections in HDF structures)

Meta data, arrays, or other auxiliary structures must be encoded to text and decoded back by the user, e.g., by using the JSON format!

Tables cannot be nested. The data base table space is flat! But specific tables can be used to reference other tables (like I-nodes, directories in file systems, or sections in HDF structures)

Meta data, arrays, or other auxiliary structures must be encoded to text and decoded back by the user, e.g., by using the JSON format!

Be aware of memory data layer hierarchy affecting performance (read/write): Data and DB Cache, Main Memory, File system, Storage Device(s).

[https://www.guru99.com/sql-server-architecture.html]

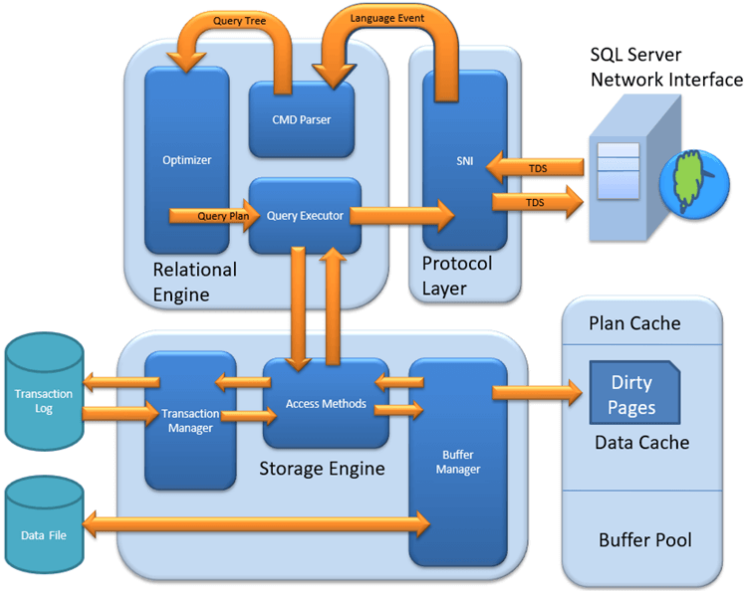



Fig. 6. Detailed and advanced SQL server architecture

## MySQL

- One of the top open-source relational database management systems (RDBMS)
- Data security layers to protect sensitive data.
- Scalability for when there are large amounts of data.
- Open source RDBMS with two separate licensing models.
- Multi-master ACID transactions through MySQL Cluster ((Atomicity, Consistency, Isolation, and Durability).
- Supports both structured data (SQL) and semi-structured data (JSON).

## SQLite3

- Minimal but powerful SQL implementation (with page caching) that can be packed into on C programming language file
- Our *sqld* server bases on a library version of SQLite3
- Databases are store in generic files
    - Underlying filesystem has impact on performance

## Apache Cassandra

- Is an open-source and highly scalable NoSQL database management system
- Handles massive volumes of data.
- One of the most scalable databases with automatic sharding.
- Offers linear horizontal scaling.
- Decentralized database with multi-datacenter replication and automatic replication.
- Fault tolerant by automatically replicating data to multiple nodes.

## PostgreSQL

- It extends the SQL language and combines it with various features to scale and safely store highly complicated data workloads.
- Highly secure with a robust access-control system.
- Offers ACID transactional guarantee (Atomicity, Consistency, Isolation, and Durability)
- PostgreSQL extension Citus Data offers Distributed SQL features.
- Advanced indexes such as Partial Index and Bloom Filters.
- Supports structured data (SQL), semi-structured data (JSON, XML), key-value, and spatial data.

## MonogDB

- It was designed to specially handle document data
- Horizontal scaling via auto-sharding (method for distributing data across multiple machines).
- Built-in replication through primary-secondary nodes.
- Distributed multi-document ACID transactions with snapshot isolation.
- Full-text search engine and data lake built on MongoDB

# MLDB

- Machine Learning Database, or MLDB, is an open-source system aimed at tackling big data machine learning tasks.
- It can be used for data collection and storage through the training of machine learning models, or to deploy real-time prediction endpoints.
- MLDB is one of the easier datasets to use, since it provides a comprehensive implementation of the SQL SELECT statement.
- It treats datasets as tables, making it easier to learn and use for data analysts already versed in an existing Relational Database Management System (RDBMS).
- Supports vertical scaling with higher efficiency.

... more information ...

https://www.unite.ai/10-best-databases-for-machine-learning-ai

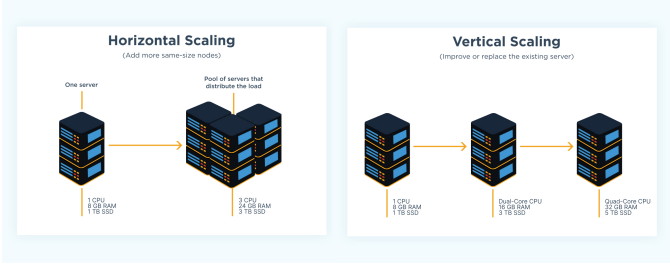
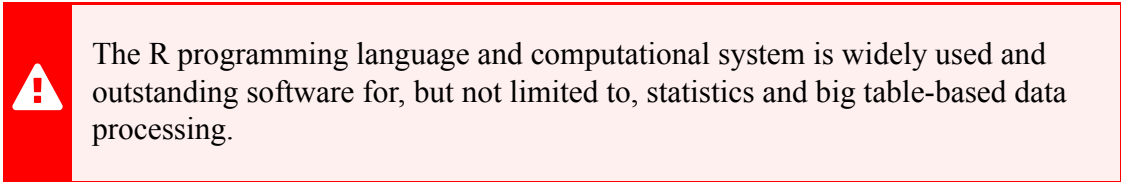



Fig. 7. Horizontal Vs. Vertical Scaling

- **Horizontal scaling** refers to **adding additional nodes or machines** to the infrastructure to cope with new data demands.
- **Vertical scaling** describes adding additional resources to a system so that it meets data demands. Ressources: CPU Power, Memory and storage capacity.

# Data Types in R



The R programming language and computational system is widely used and outstanding software for, but not limited to, statistics and big table-based data processing.

Core data types are:

- number (numeric)
- boolean
- string (character text)
- function

That's all folks!

# Data Types in R

But data can be organized in:

- **Vector** 1-dim (poly-sorted, i.e., not compacted with heterogeneous element types possible)
- **Matrix** 2-dim
- **Array** n-dim with **Typed Arrays** (mono-sorted. i.e., compacted/packed and each element with same data type):
  - Int8, Uint8 (Integer/Unsigned Integer)
  - Int16, Uint16
  - Int32, Uint32
  - Float32, Float64
- **List** as data records
- **Data frame** providing tables with rows and columns

# Data Types in R

```
  #      R                          R+
v1 = c(1,2,3,4)             v1=[1,2,3,4]
l1 = list(x=1,y=1,z=0)      l1={x=1,y=1,z=0}
m1 = matrix(0,3,2)          m1=[|1,2;3,4;5,6|]
a1 = array(0,c(3,2,2))      a1=array(0,[3,2,2])
df1 = data.frame(
  a=c(1,2,3,4),             a=[1,2,3,4],
  b=c('A','B','C','D'),     b=['A','B','C','D'],
  c=list(c(1,2),c(3,4)..), c={[1,2],[3,4],..},
  ...
)
```

Ex. 1. R list, vector, matrix, and array data

# Data Types in R

SQL tables are a sub-set of R data frames!

- Data tables can be use to bind meta and analysized data to the original measured data, e.g., images.
- Columns (and rows for 2-dim data) of vectors, lists, matrix, and data frames can be accessed and modified by a numerical or name index (if available):

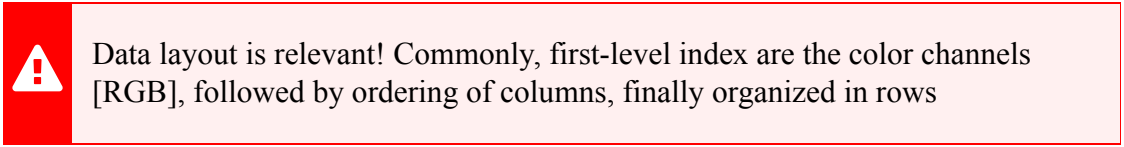
```
l1={x=1,y=1,z=0}   ==>  l1$x=0 l1[[1]]=0 print(l1[1])
c1=[1,2,3,4]            c1[2]=c1[1]+1
m1 = matrix(0,3,2)      m1[2,1]=0
df1 = data.frame(       df[1] df$a df1[[1]] df[1,2]
  a={1,2,3,4]
)
```

Ex. 2. R access of list, vector, matrix, and data frame elemnts (and columns)

# Image Data

Formats:

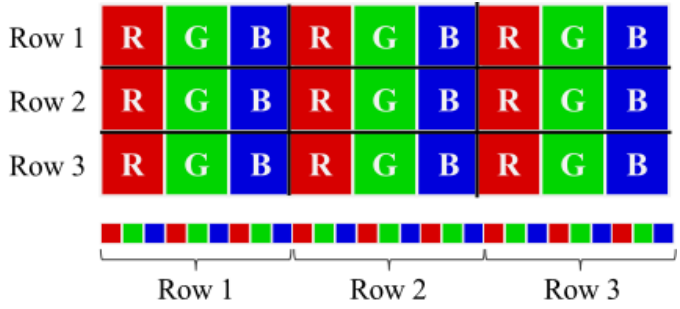
- **RGBA** ⇒ `Uint8 [Red,Green,Blue,Alpha] [col][row]`
- **RGB** ⇒ `Uint8 [Red,Green,Blue] [col][row]`
- **GRAY8** ⇒ `Uint8 [col][row]`
- **GRAY** with n-bits per pixel (8,16,24,32)



# Image Data

```
G G G G G G                                  ...
RGB RGB RGB RGB RGB RGB                      ...
RGBA RGBA RGBA RGBA RGBA RGBA                ...
-------------------------------------
R1[C1  C2  C3] R2[C1 C2 C3]                  ...
```

Ex. 3. Memory layouts of different image formats




[DOI: 10.1109/CLEI.2015.7359995]

Fig. 8. Structure of the RGB image as a sequence of bytes in a linear memory or file model

# Images and Matrices

- A 2-dim matrix can be considered as a graylevel intensity image.
  - Direct numeric on pixels is possible
  - Many algorithms process graylevel images (only), like edge detectors
- RGB images require an 3-dim array. The third dimension represents the color channels.
- In R, an image has commonly its own data type *cimg*.
- A matrix can be converted to an image (*cimg*) and vice versa.

# Images and Matrices

```
m = matrix(runif(100),10,10)
m.i = as.cimg(m)
i = load.image('http://edu-9.de/assets/test.png',
               format='RGBA')
i.m = as.matrix(i) # converts automatically to graylevel
plot(i)
plot(i.m,auto.scale=TRUE)
```

Ex. 4. Image to matrix and vice versa

# Image Stacks

- An image file contains commonly one image
- But indexed image stacks stored in one file are supported by many image file formats, e.g., TIFF
- If an image stack is loaded, typically a vector (or list) of images is returned.
- Reconstructed CT slice image stacks contained in one file are typical examples

# Image Data Compression

The memory or file size of a flat image is:

$$Size(im) = w \cdot h \cdot d \cdot b$$

with *w* and *h* as the width (number of columns) and height (number of rows) of the image, respectively, *d* as the channel depth (1,3,4), and *b* as the number of Bytes per pixel and channel (1,2,4 → 8, 16, 32 Bits)

- To reduce the file size, the image data can be compressed. We distinguish:
  - Reversible compression algorithms like LZW (TIFF)
  - Irreversible compression algorithms like JPEG

# Image Data Compression

Never use irreversible compression since it causes artifacts and noise on decompression!

Compression of graylevel images (one channel) normally has no benefit (low compression ratio, but high computational times). The same fact holds for high precision measured images (more than 8 Bits per pixel). Detector noise will prevent efficient compression.

# Data Frames

The function `data.frame()` creates data frames, tightly coupled collections of variables which share many of the properties of matrices and of lists, used as the fundamental data structure by most of R's modeling software.

- A data frame is a list of variables of the same number of rows with unique row names, given class "data.frame". If no variables are included, the row names determine the number of rows.
- The column names should be non-empty, and attempts to use empty names will have unsupported results. Duplicate column names are allowed, but you need to use check.names = FALSE for data.frame to generate such a data frame. However, not all operations on data frames will preserve duplicated column names: for example matrix-like sub-setting will force column names in the result to be unique.

# Data Frames

Structured files formats like CSV or JSON can be directly converted to data frames!

There is a large set of low- and high-level operations that can be applied to data frames (as well as matrices).

# Synthetic Images

- Matrices as well as empty images can be used to create synthetic images by geometric operations:
  - Additive drawing of figures like circles, ellipses, rectangles, lines with and without filling (color and graylevel)
  - Drawing polynomial or exponential 2-dim functions, e.g., Gauss function
  - Subtractive or multiplicative image fusions
- Synthetic images can be used for:

1. Testing of algorithms
2. Training of ML models

# Synthetic Images

```
use math,imager,plot,geometry
im = load.image('pathto.tiff',format='GRAY')
m.corr = matrix(0,height(im),width(im))
draw.gaussian(m.corr,min=0.5,max=1,sigmax=150,sigmay=100)
im.corr = im/m.corr
```

Ex. 5. Creating of synthetic images by using geometric and matrix operations for image intensity normalization

# File Data Formats

**CSV**
: Comma Separated Value format (text)

**JSON**
: JavaScript Object Notation (text)

**XML**
: Extensible Meta Language (text)

**YAML**
: Yet Another Meta Language (text)

**NumPy**
: Numerical Python Format (binary)

## JSON

- Array and record structure (nested, tree)

```
{
  "employee": {
    "name":       "sonoo",
    "salary":      56000,
    "married":    true,
    "awards" :    [1920,1990,2000]
  }
}
```

# CSV

- Flat table (not nested) with values separated by delimiter (e.g., "," or any other delimiter like the tabulator character)
- Numbers and text are valid values
- Each line in the file is one row in the data table
- All rows should contain same number of values

```
x,y,z,class
1,2,3,"A"
1,4,2,"B"
4,5,0,"A"
```

# CSV

In R CSV files are represented by (converted to) data frames!

```
use csv
text = 'x,y,z\n1,2,3\n4,5,6\n7,8,9'
df   = parse(text)
df   = read.csv('pathto.csv',sep=',',header=TRUE)
```

Ex. 6. R CSV reader

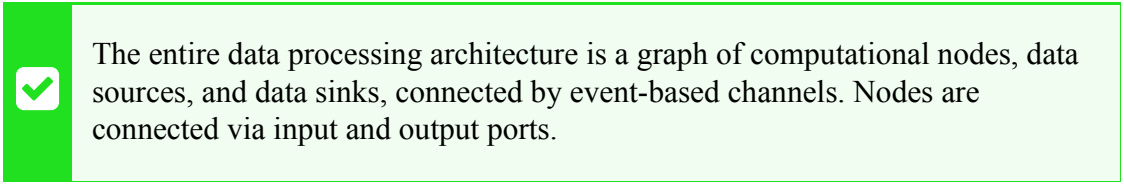
# YAML

YAML supports nested data structures, but table-like data is difficult to maintain.

```
martin:
  name: Martin D'vloper
  job: Developer
  skill:
    - a
    - b
    - c
```

Ex. 7. Lists and records in YAML

# Data Flow Architecture



The entire data processing architecture is a graph of computational nodes, data sources, and data sinks, connected by event-based channels. Nodes are connected via input and output ports.

### Data Source

A data source node provides access to data, e.g., from an SQL database or from the file system. There are read and write events. Data tokens are created on request (by a read event trigger, e.g.).

### Data Sink

A data sink nodes consumes data tokens and displays them. Typical display types are textual information, tables, and plots.

## Computational Functions

A computational node processes data read from input ports and writes the processed data on output ports.

Examples are:

- Statistics (from data tables)
- Matrix operations (in-place or by creating new Matrix data)
- Image operations and transformations, filters
- Time-Frequency transformations, convolution, wavelet transformations
- Merging of data, splitting of data, wrapping and unwrapping of data
- Data format conversion

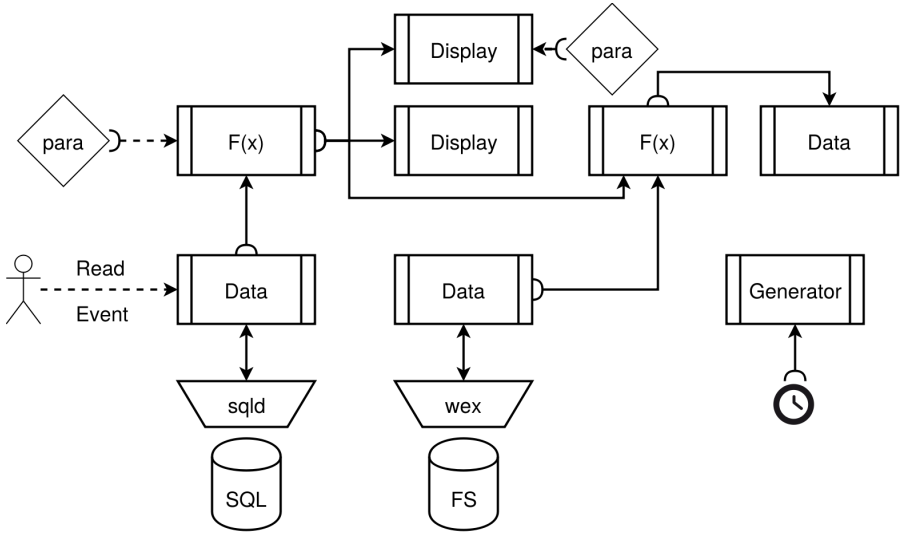



Fig. 9. Processing graph and data flow architecture with data source, processing, and sink nodes. Event-based data flow architecture and event chains. New data provided by a node is propagated to all child nodes. Parameter changes initiate a re-computation (or display), too.

# SQL-RPC API

```
use sql,json
db = connect("localhost:9999")
mytable.schema = db::schema("mytable")
mytable.nrow = db::nrow("mytable")
mytable.data = db::read("mytable")
transform(mytable.data,c=as.vector(c,mode="uint16"))
data = db::read("mytable",b=fromJS(b),
                c=as.vector(c,mode="uint16"))
```

Ex. 8. R+ SQL operations

## SQLD

- `sqld` consists of a slim native C-code implementation of the *sqlite3* server storing SQL data bases in plain binary files on the local file system.
- SQL data bases can be accessed by a Remote Procedure Call JSON interface, basically mapping SQL operations on a JSON structure (both request and reply).
- HTTP is used to access the JSON-RPC API

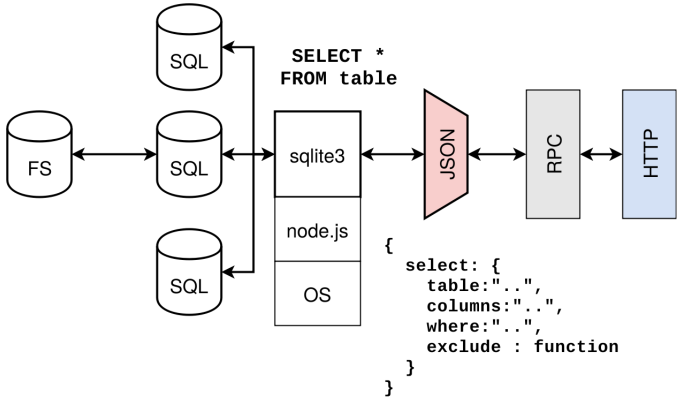



Fig. 10. SQLD architecture and JSON-RPC interface

# Summary

- Data processing is performed by using a sequential programm language, R+
- Data can be represented by different data types, structures, formats, supported by R+
- Data access is provided by files, HTTP services, or SQL data bases in an unified way