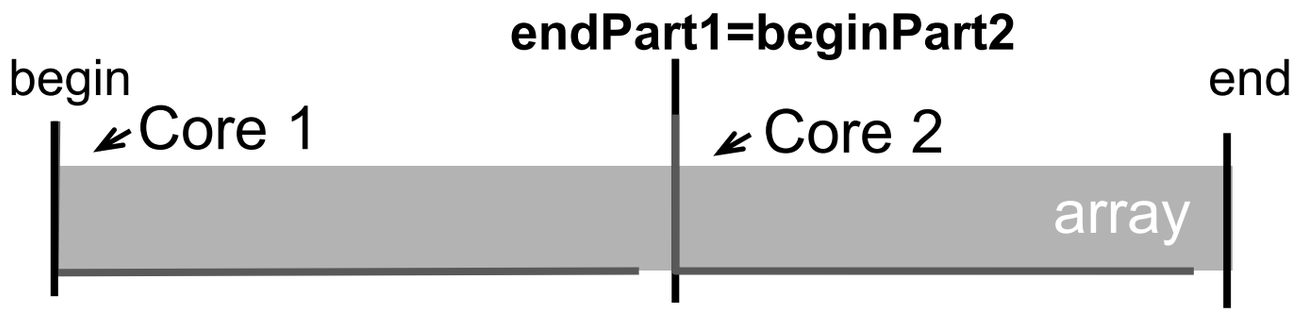
# Functional Programming in Python for Big Data

Dr.-Ing. Christoph Fünfzig

# Introduction

global max: 425

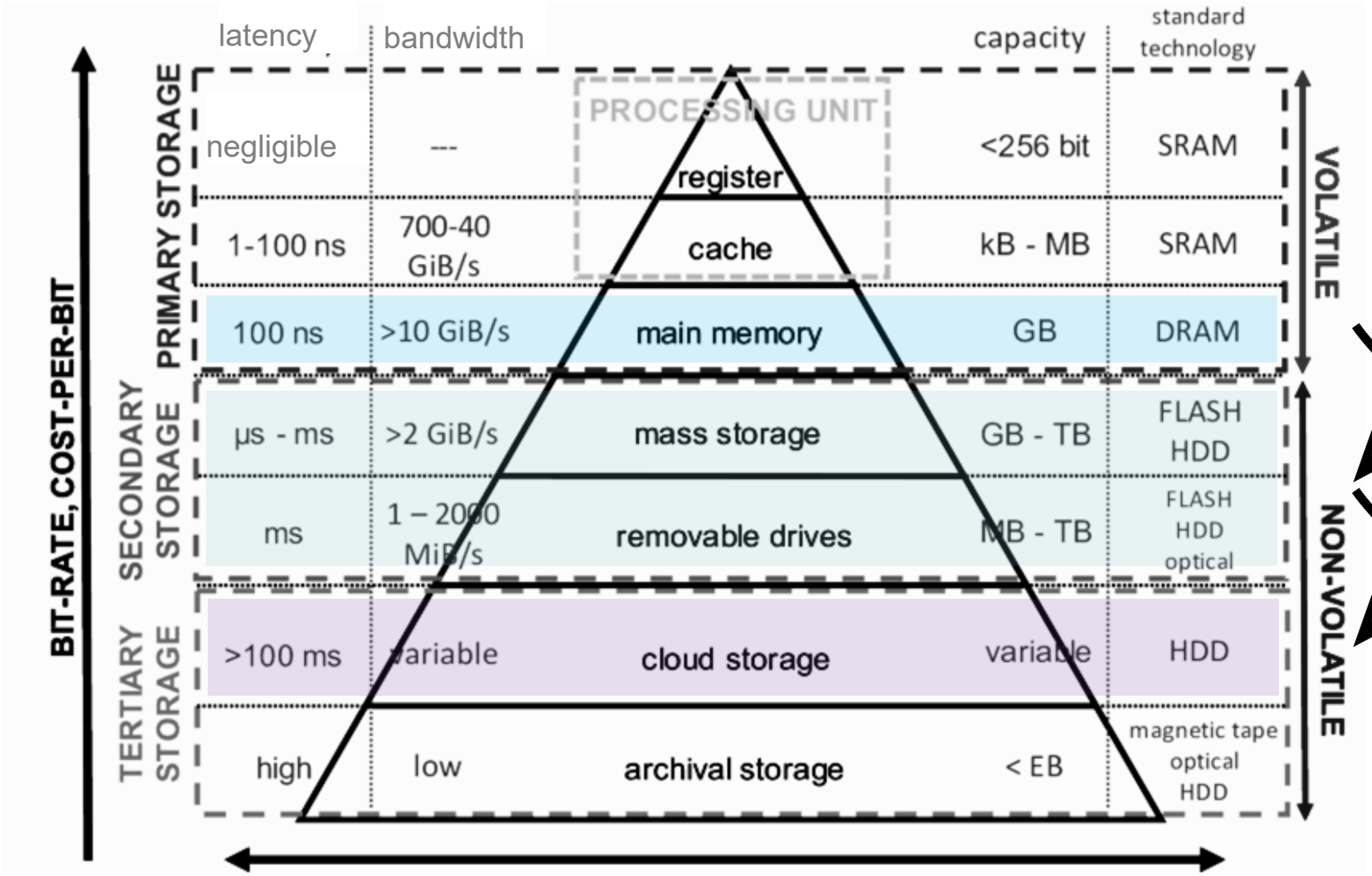



*Question:*

How much do you trust a global variable (global max) used in a parallel system with 1000 cores/servers (on a scale 1-10)?

# Learning Outcomes

– By the end of this 15-minute session,
you understand the concept of code that *scales in principle*.




from Emanuele Gemo: The design and analysis of novel integrated phase-change photonic memory and computing devices

# Learning Outcomes

- By the end of this 15-minute session,
  you understand the concept of code that *scales in principle*.
- made possible by functional programming, which allows:
  partitioning the data (*scalability*),
  replicating the data (*reliability*).

# Functional Programming for Big Data

- Immutability:
  Data is *read-only*. We don't change it, we transform it into a new version.
- Pure Functions: Input transforms to desired Output.
  No *Side Effects* (like printing or global variables).

→Data is placed best (which? where? when?).

# Functional Programming Transformations

1. *Map* (narrow transformation)
   Execute a function on a data row aka. tuple.
2. *Filter* (narrow transformation)
   Select a data row/tuple based on a condition.
3. *Reduce* (wide transformation)
   Group data rows by key and execute aggregation function (e.g. sum, min, max).

# Continued Example: Word Count

*Read a textfile (consisting of rows), separate into words (resp. spaces) and count the #occurrences of each word.*

*3 data rows (data input)*

```
how much ground would a groundhog hog, if a groundhog could hog ground. a groundhog
would hog all the ground he could hog, if a groundhog could hog ground.
```

*split into single words (map)*

```
how
much
..
```

*count (reduction)*

```
how         1
much        1
ground      2
would       1
a           3
```

# Framework *PySpark* @BigData-Industry

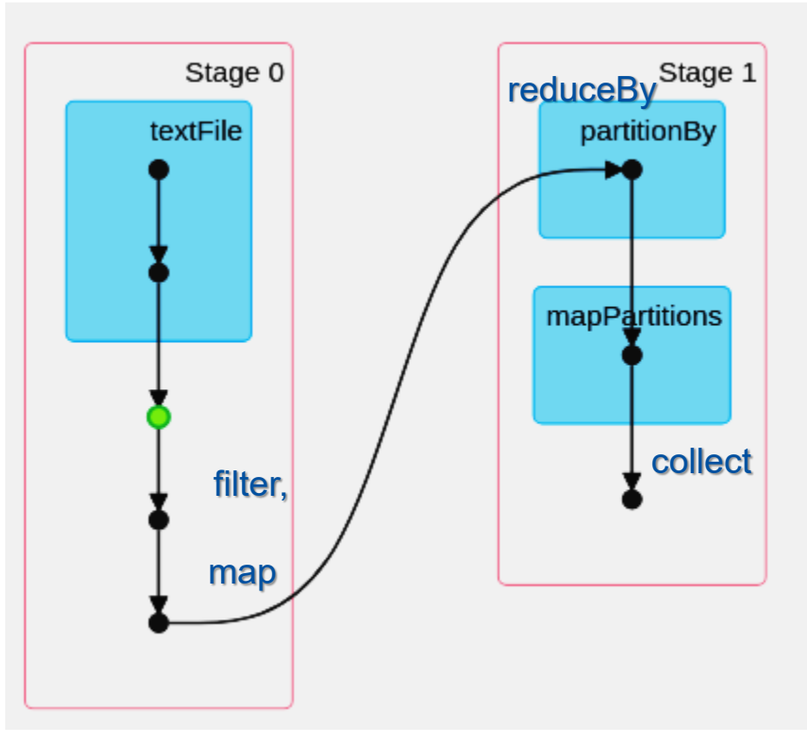
*PySpark* represents data as RDD (*resilient distributed dataset*) objects

.. **Transformations** *(map,filter,reduce)* on RDD objects store just the function

Lazy Evaluation

.. **Actions** *(take(10),tail(10),collect(),distinct(),..)* evaluate the transformations

This is a way to represent workflow:
Repeatable! Optimizable! Scalable!



# Continued Example: Word Count

Short Demo of PySpark Code: *loftr.txt* (0.4gb, ~500k words)

```
# Input
words   = sc.textFile(“data/loftr.txt“)

words2 = words.map(lambda line: re.sub(r“[\s,;\.:!-<>“, “ “, line.lower())
                .strip().split(“ “).flatMap(lambda w: w)
words2 = words2.cache() # structuring: memory caching

# Transformations: filter first, then count
stopwords           = words2.filter(lambda word: word in stopwordsList) # only stopw.
stopwordsMapped = stopwords.map(lambda word: (word, 1)) #word to tuple(word,1)
stopwordsCount    = stopwordsMapped.reduceByKey(lambda c1, c2: c1+c2) #add
# Actions
result                  = list(stopwordsCount.collect())
print(result, len(result))
```

# Continued Example: Word Count

We have seen intermediate stages of the word-data processing!

*Task:* You are interested in the words, which *are not stopwords* and *occur ≥10 times*.

*Question:*

1. Which transformation keeps those words? (map/filter/reduce)
2. Which transformation orders are possible? (map, reduce, filter)/(filter, map, reduce)

# Summary & Closing Remarks

- Functional Python allows resilient/repeatable, distributed, lazy transformations of big datasets
- Functional patterns implemented by Python functions or lambda-expressions
- Heavy use in BigData-Industry: Spark/Dash in Azure Databricks, AWS, SAP Spark
  Strategy: *Move the logic to the data! Optimize data movement!*
- Structured API on top of RDDs:
  *DataFrame, SQL UserDefinedFunction*