Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Memory-based computing for astronomical applications

Hermann Heßling

711. WE-Heraeus-Seminar:

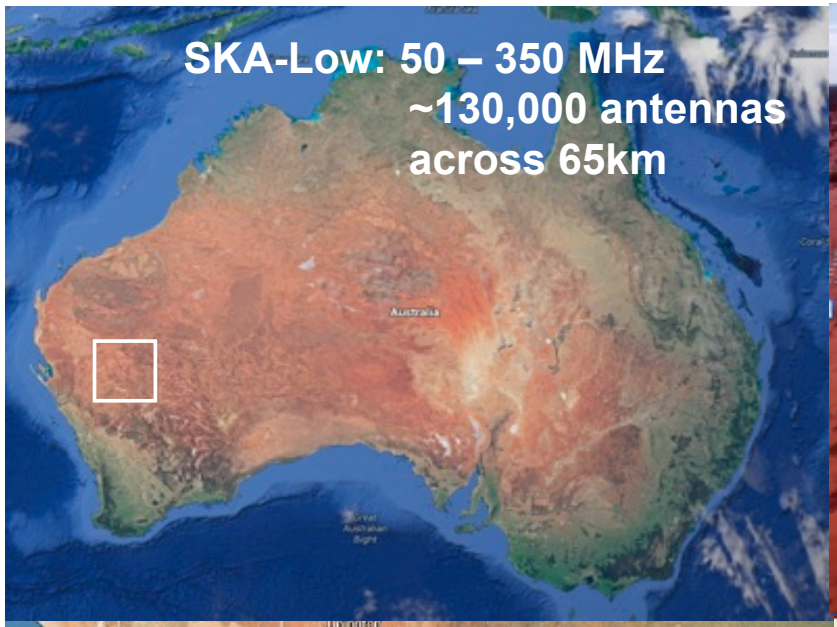
The Science Cloud

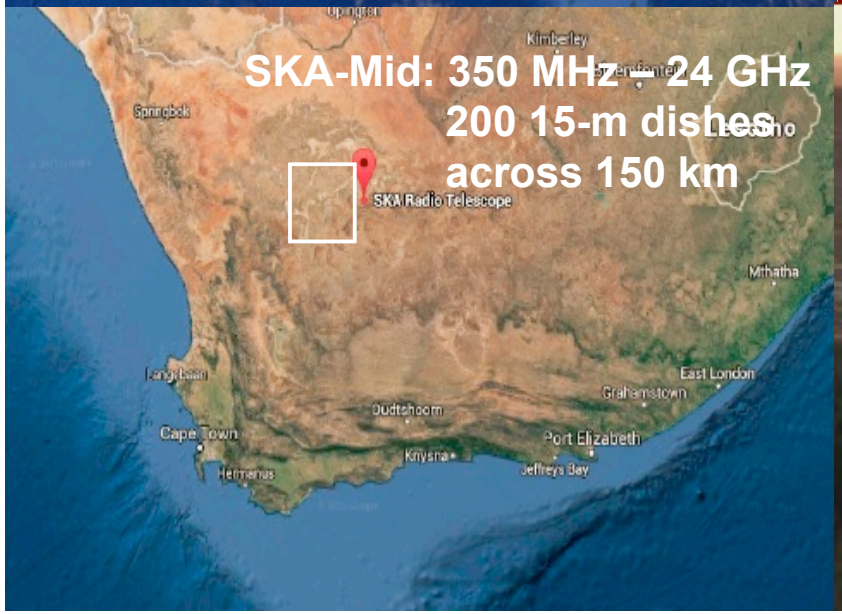
Jan. 12-15 2020

Outline

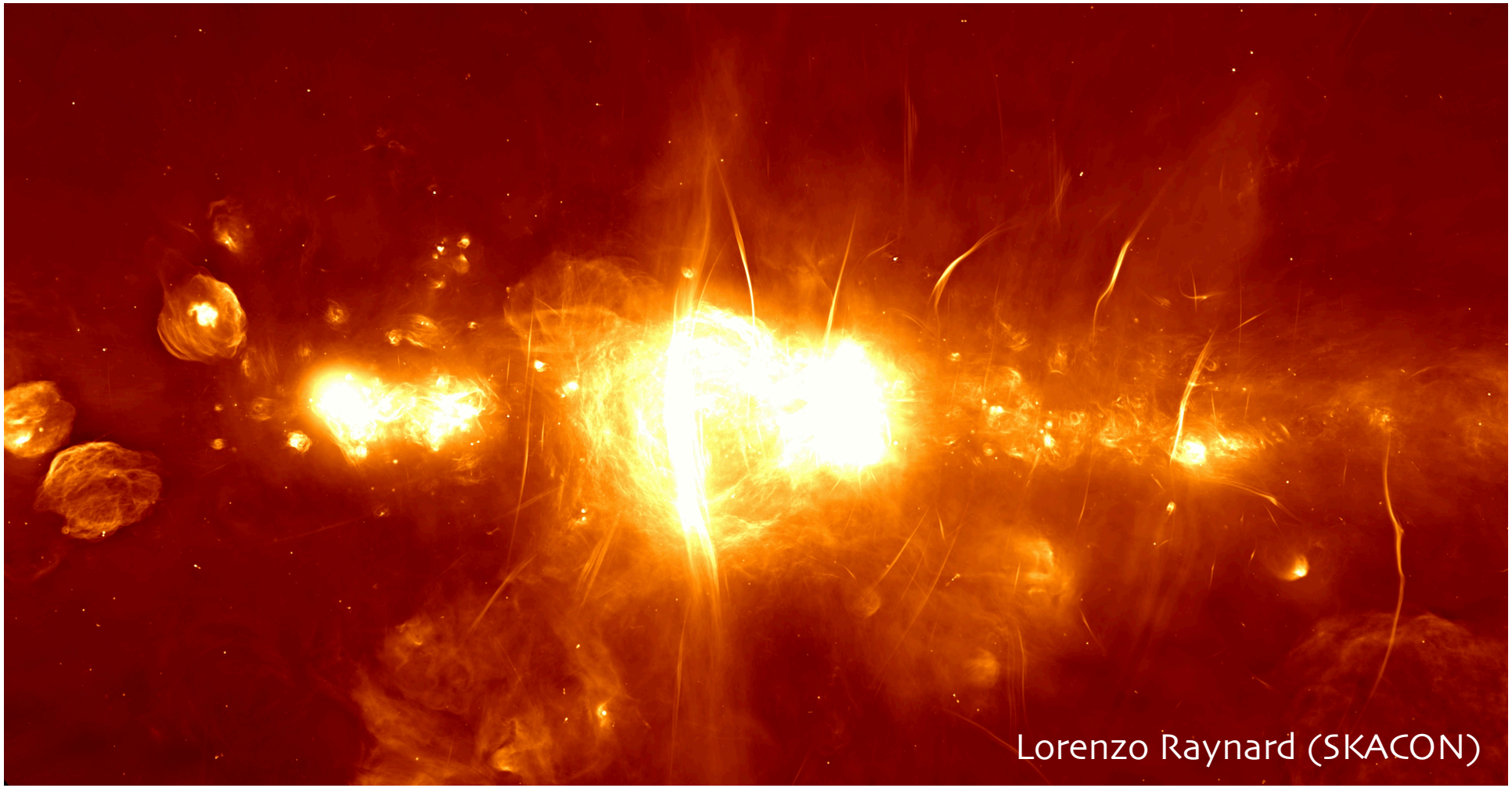
- Data challenges at the Square Kilometre Array (SKA)
- Memory-based computing
- Divide&Conquer in image processing
- Summary and outlook

SKA: Telescopes in AU & SA



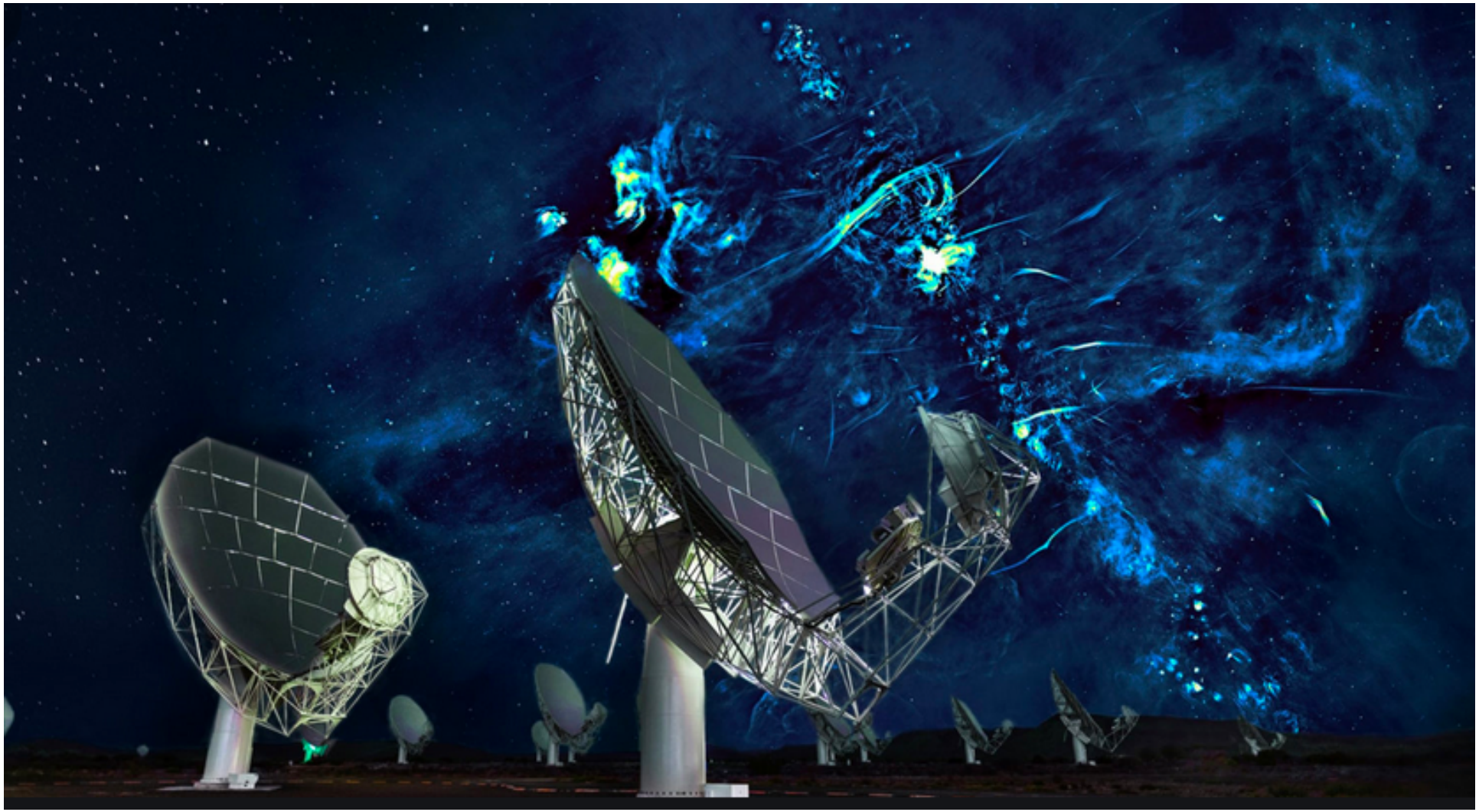


MeerKAT



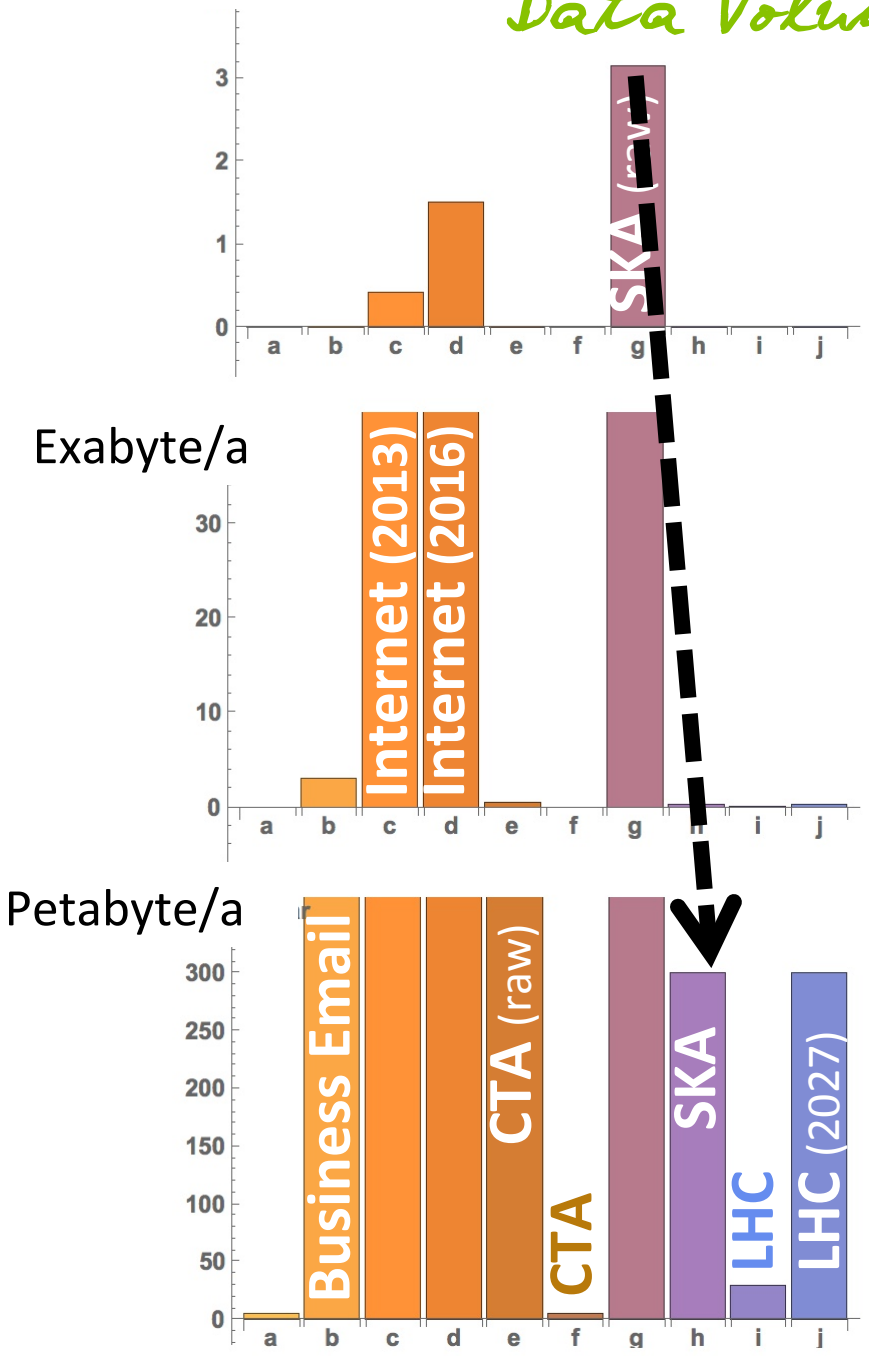
Center of Milkey Way: mysterious „fibers“ around supermassive Black Hole (July 2018)

MeerKAT



A composite of the radio bubbles in the Galactic Centre and the MeerKAT telescope (SARAO; I. Heywood et al., Nature, Sept. 2019).

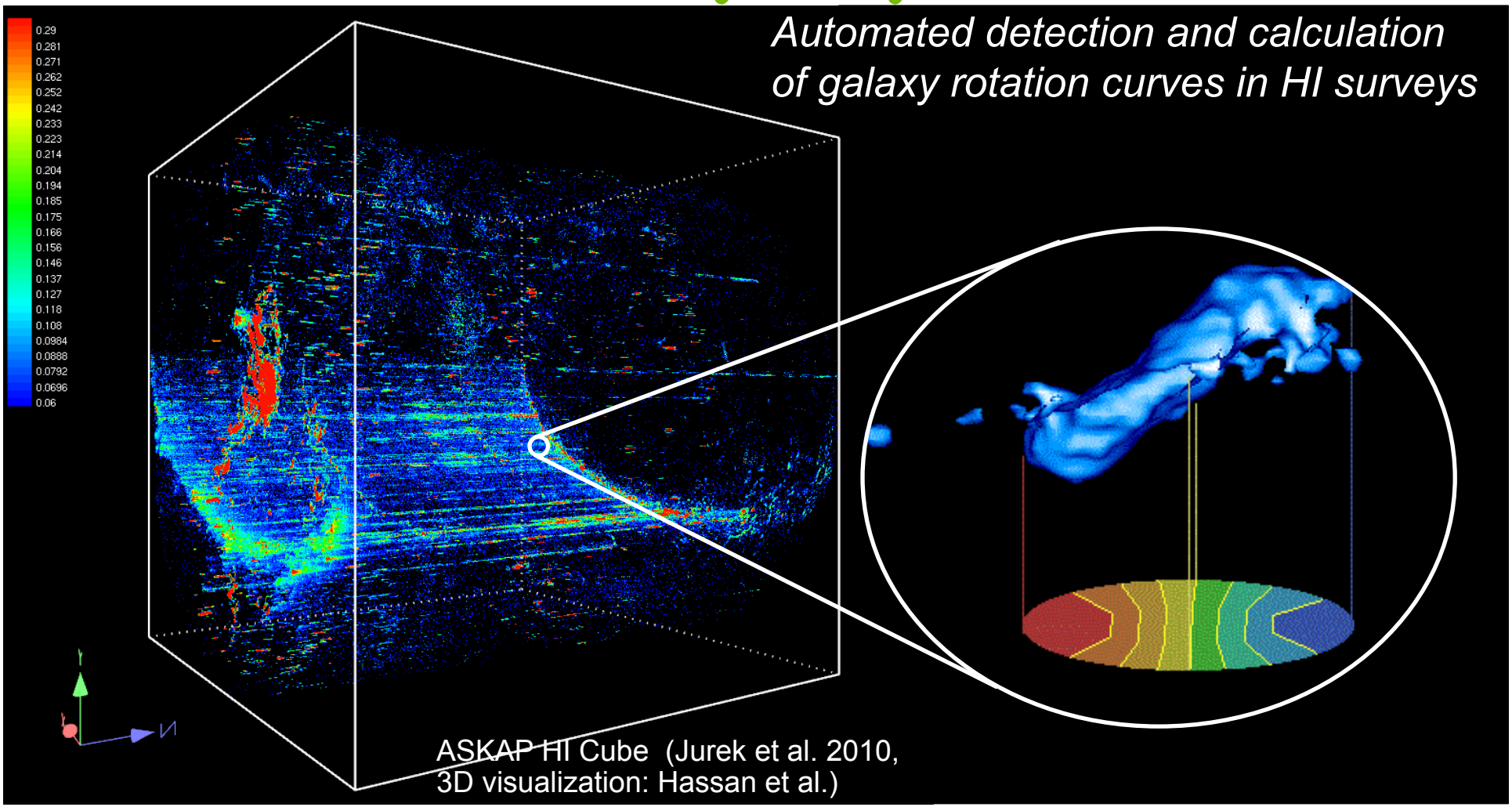
Data Volumes



- a: US Library of Congress
- b: Business Emails (worldwide)
- c: Global internet traffic (2013)
- d: Global internet traffic (2016)
- e: CTA: raw data rate
- f: CTA: archive volume
- g: SKA: raw data rate
- h: SKA: archive volume
- i: LHC: archive volume (2009–2012)
- j: HiLumi LHC: archive volume (>2027)

Data irreversibility challenges
- reduction in near-realtime
- dramatic loss of information
- machine learning
- simulation

Offline-Challenge : Huge Data Objects



- SKA: up to 1 Petabyte / Cube
  - 36.600 x 36.600 x 250.000 pixel

Sandbox @ HP Labs

Sandbox (= 2 Superdome Flex)

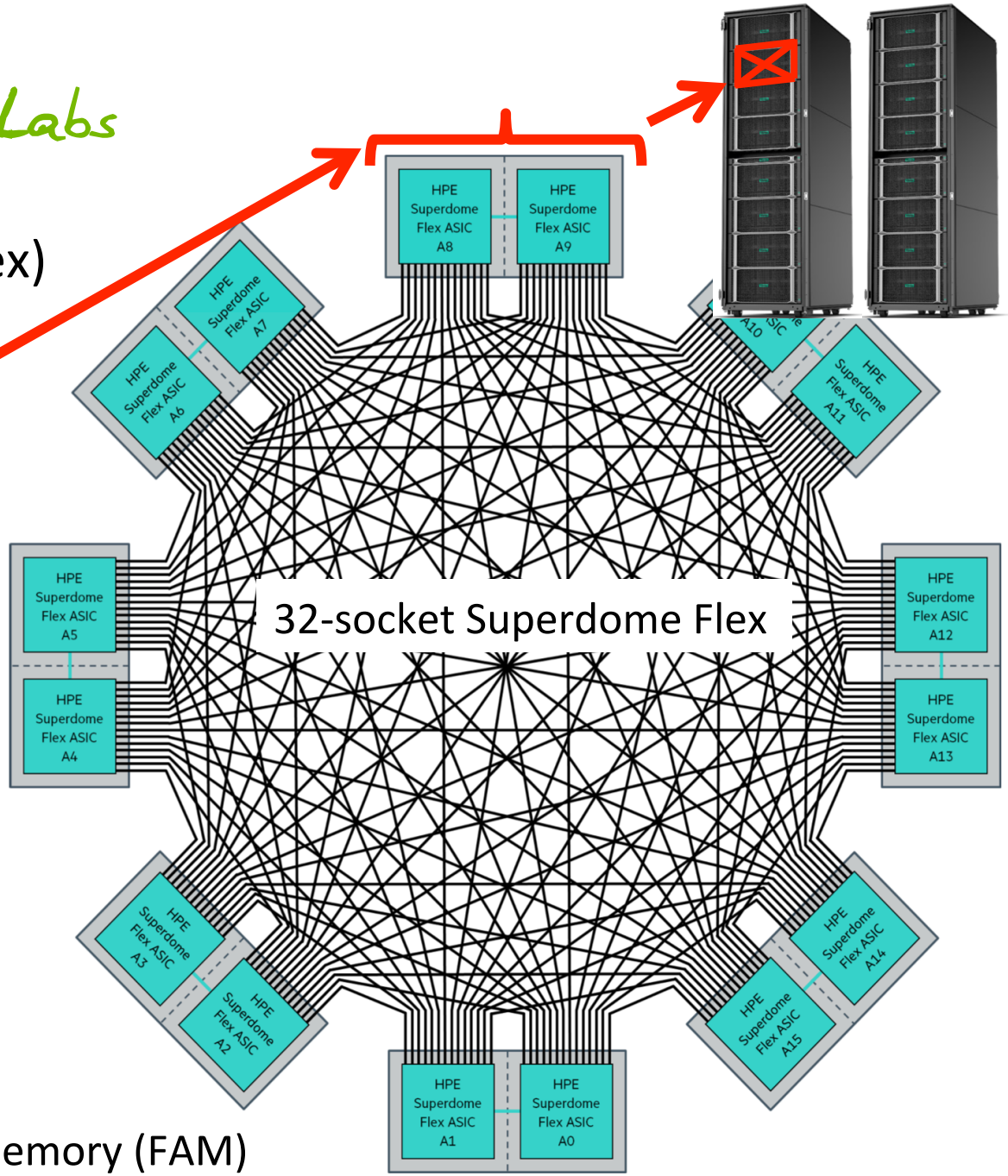
- 64 processor sockets
  - 4 processors per chassis
- 18 cores per processor
- **48 TB memory** (no discs)

NUMAlink cabling

- Point-to-point between each chassis
- 13.3 GB/s

Subsystem (⇒ HTW Berlin)

- 1 chassis ⇒ Linux
- 1 chassis ⇒ Linux
- 2 chassis ⇒ Fabric Attached Memory (FAM)



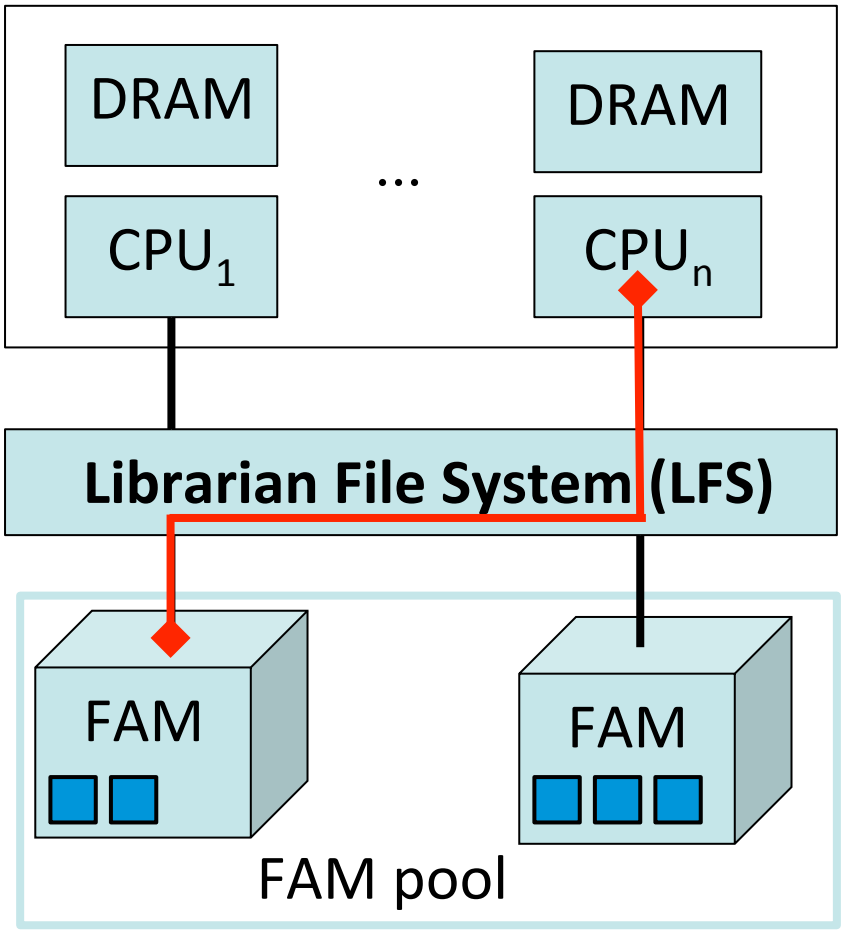
HPE: Superdome Flex server architecture and RAS. Technical white paper (2019).

Sandbox – Librarian File System

- LFS: interface for accessing FAM books (■)

- Concurrent access from multiple nodes to shared FAM pool
- **Cache-coherence**: yes, if CPU + FAM on the same node (else: no)

⇒ Atomic work between FAM nodes



FAM book
- basic unit
- 8 GB

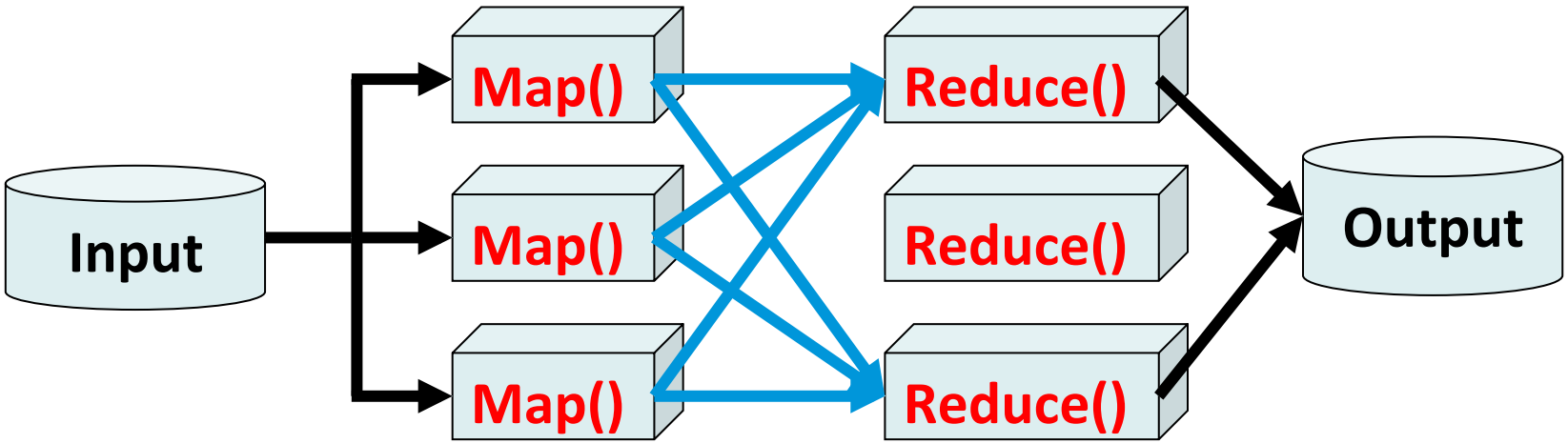
Memory-based Computing using LFS

- Data on the **FAM** are **byte-addressable**
- C++ snippet:

```
fd = open("/lfs/in.dat", O_RDONLY);
size = lseek(fd, 0, SEEK_END);
*srcAddr = (char *)mmap(NULL, size, PROT_READ,
                                    MAP_SHARED, fd, 0);
cout << srcAddr[42];
```

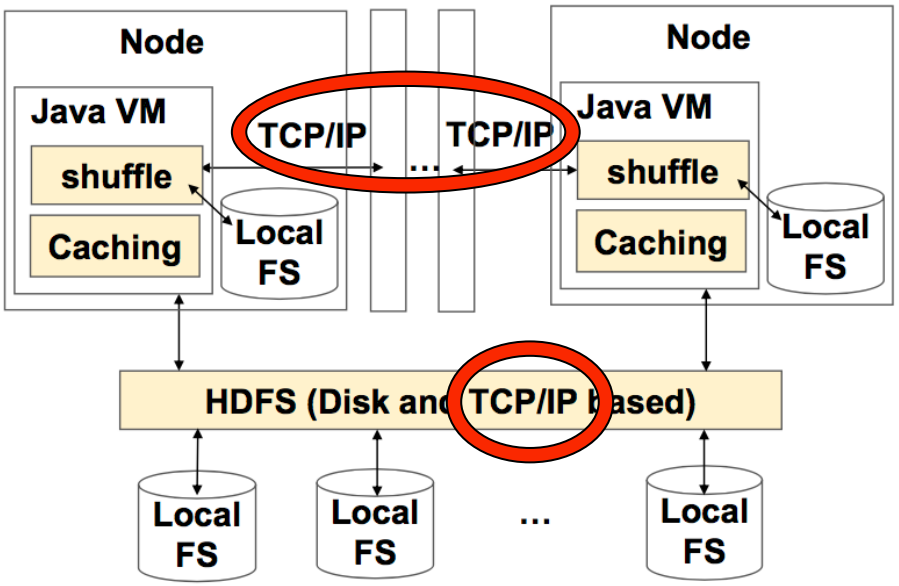
Analysing Big Data

- MapReduce method
  - Program model for distributed computing (pro: simple interface, con: slow).
  - **Map** step: input data are processed and chunks of data are created,
  - **Reduce** step: the chunks are **shuffled** to appropriate nodes, where they are further processed,
  - and the final results are collected (or a next map step is followed).



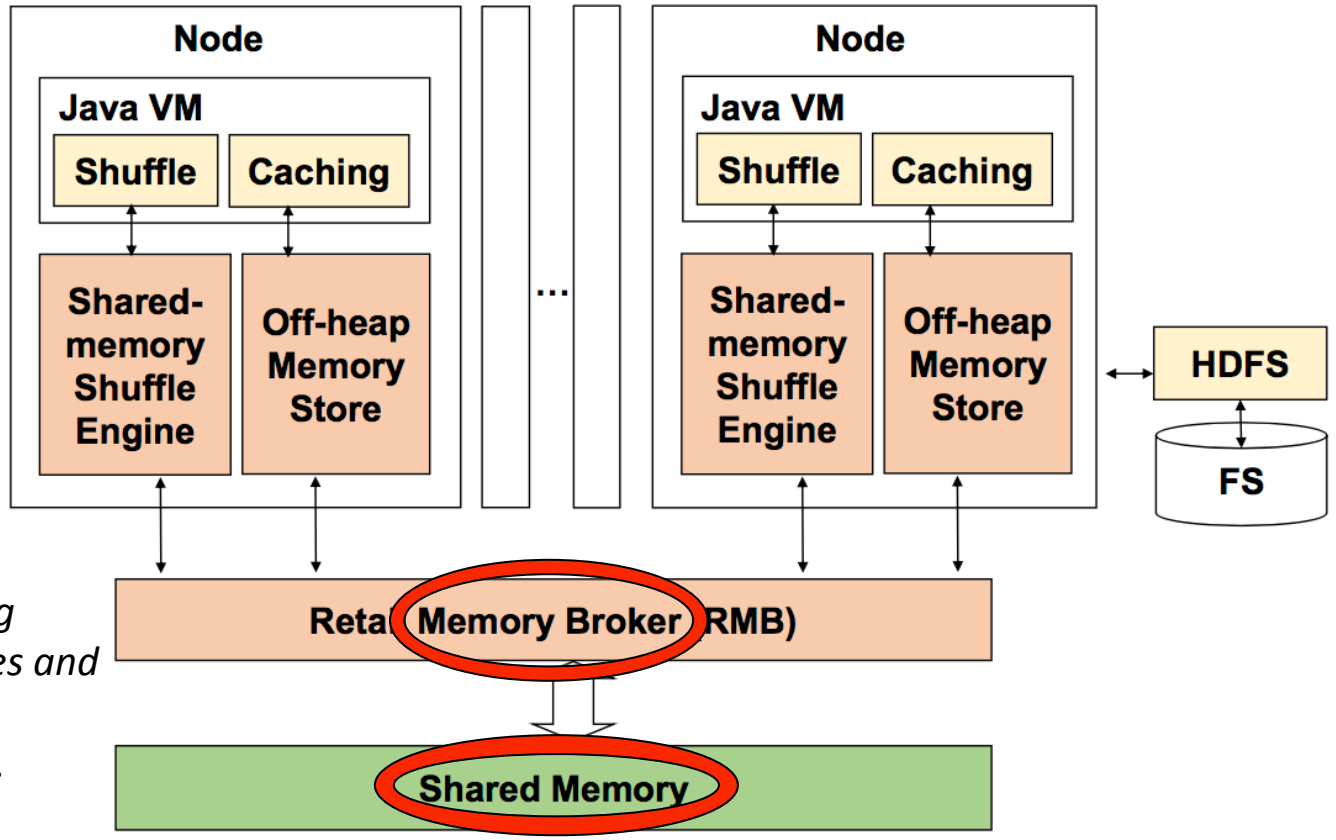
shuffle (bottleneck: node connections)

In-memory Spark (Sparkle)



Spark

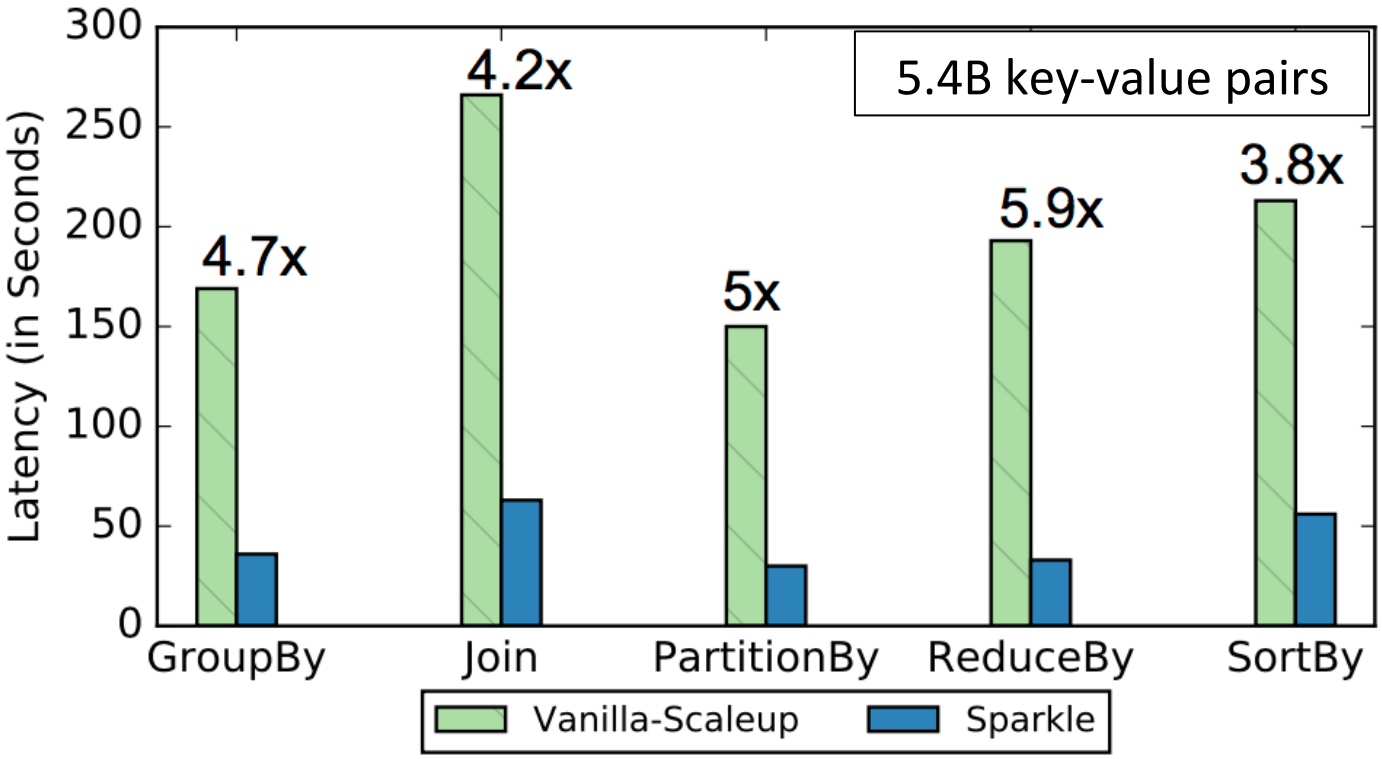
Sparkle



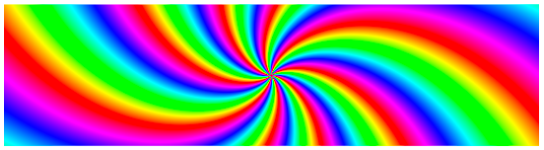
M. Kim, et al.: *Sparkle: Optimizing Spark for Large Memory Machines and Analytics* (2017). SoCC '17: ACM Symposium on Cloud Computing. arXiv:1708.05746.

In-memory Spark (Sparkle)

- Superdome X: 16 NUMA nodes, 240 cores, 12 TB DRAM
  - Spark: TCP/IP over Infiniband (IB) for shuffling data between nodes
  - Sparkle: shared-memory shuffle engine

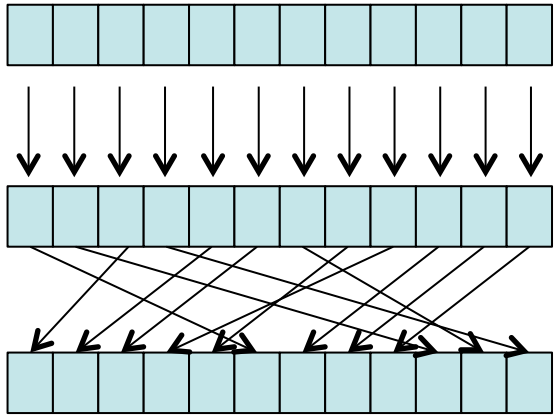


M. Kim, et al.: *Sparkle: Optimizing Spark for Large Memory Machines and Analytics* (2017)
SoCC '17: ACM Symposium on Cloud Computing, arXiv:1708.05746.



Thrill

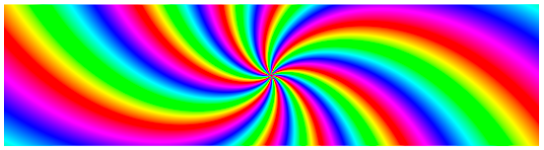
- High-performance **C++** framework for distributed Big Data processing.
- Data are transferred to a ***Distributed Immutable Array*** (`DIA`) of items.
- A `DIA` can be modified by predefined “operations”. For example,



A

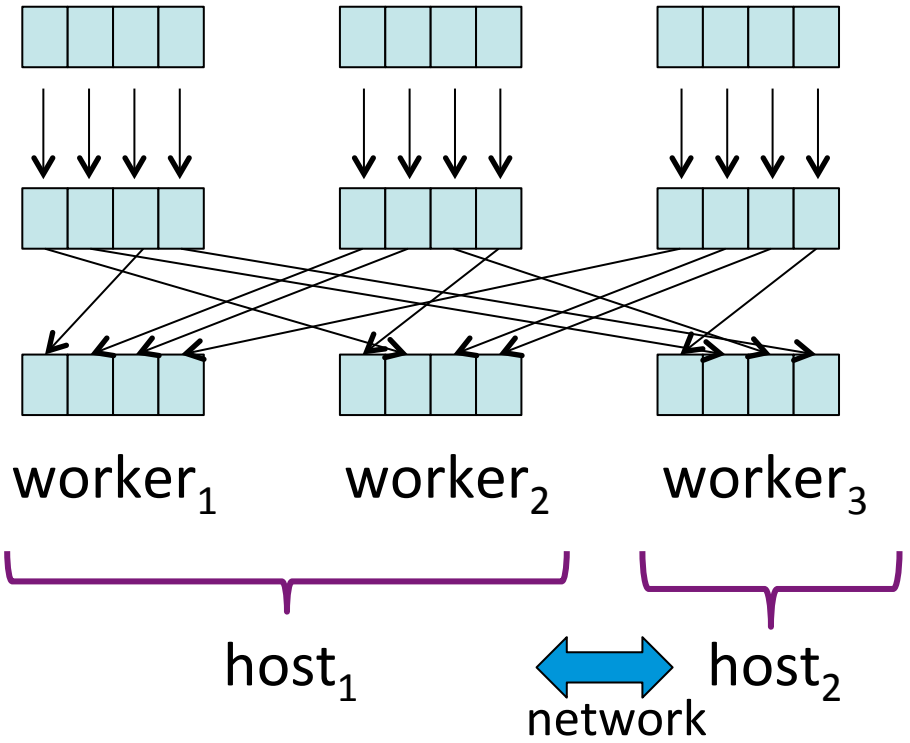
B = A.Map(...)

C = B.Sort(...)



Thrill

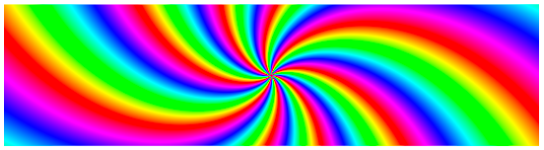
- High-performance **C++** framework for distributed Big Data processing.
- Data are transferred to a ***Distributed Immutable Array*** (`DIA`) of items.
- A `DIA` can be modified by predefined “operations”. For example,



A

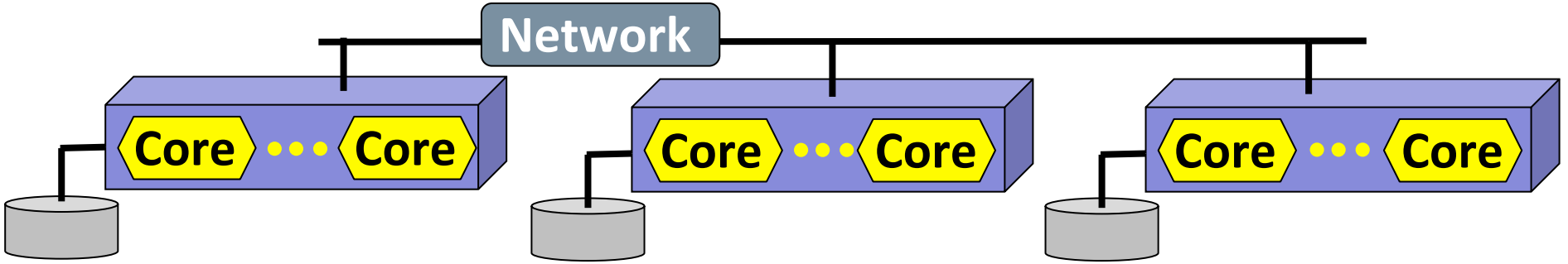
B = A.Map(...)

C = B.Sort(...)

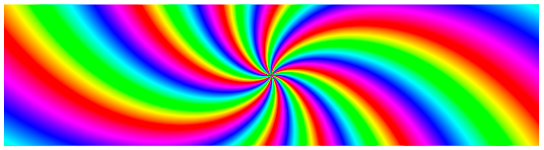


Thrill on a cluster

- Workflow on hosts is coordinated collectively over network.



- On each host of a cluster, start a Thrill binary, and specify
  - the working memory limit,
  - the number p of "workers" per host (default: p = number of detected cores).
- Communication between hosts via network protocol.
  - **TCP**, for testing: local **mock** ("peers" exchange messages via shared memory).
- Input/output stored on distributed file system (e. g. NFS, HDFS).



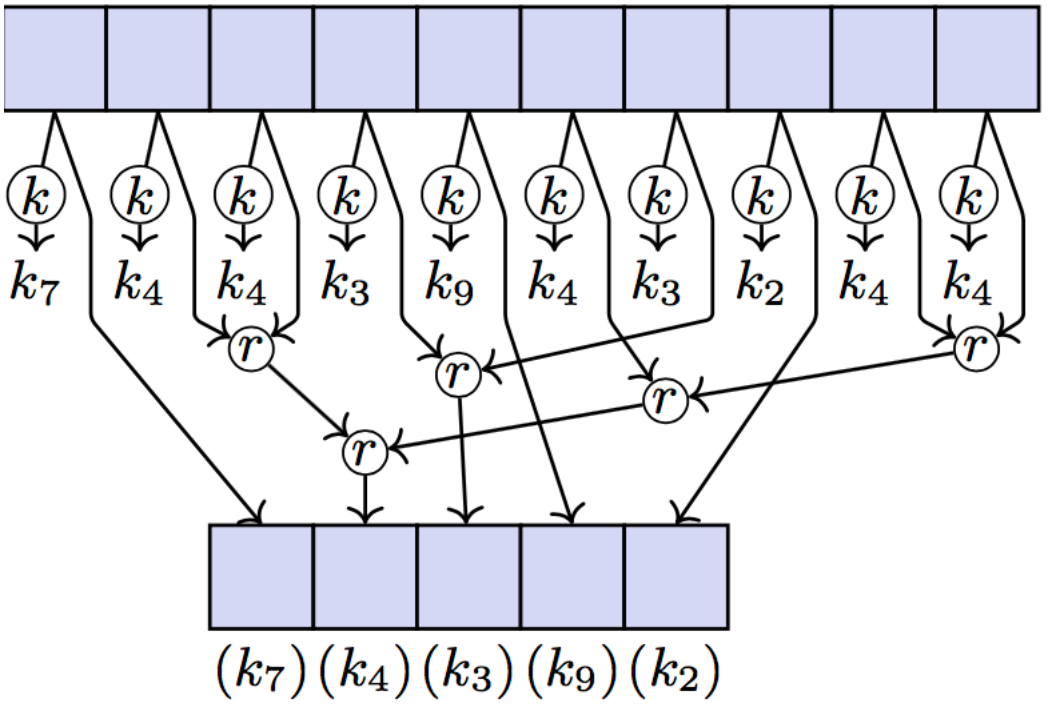
Thrill - operation on DIA

$\mathbf{ReduceByKey}(k, r) : \texttt{DIA}\langle A\rangle \to \texttt{DIA}\langle A\rangle$

$k : A \to K$ key extractor

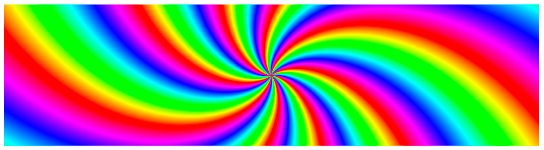
$r : A \times A \to A$ reduction

⇒ word counting (see below)



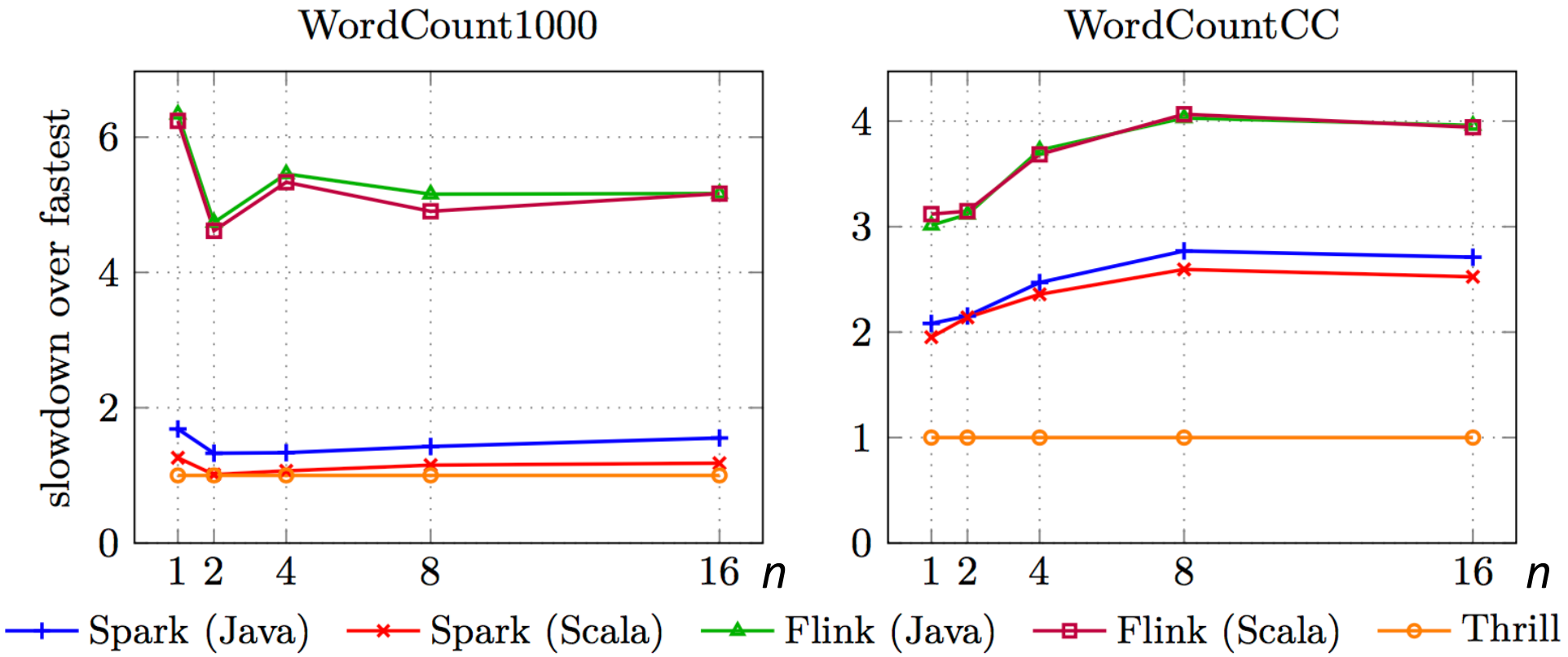
requires communication between workers

T. Bingmann: Scalable String and Suffix Sorting, PhD Thesis, KIT, 2017.



Thrill - performance

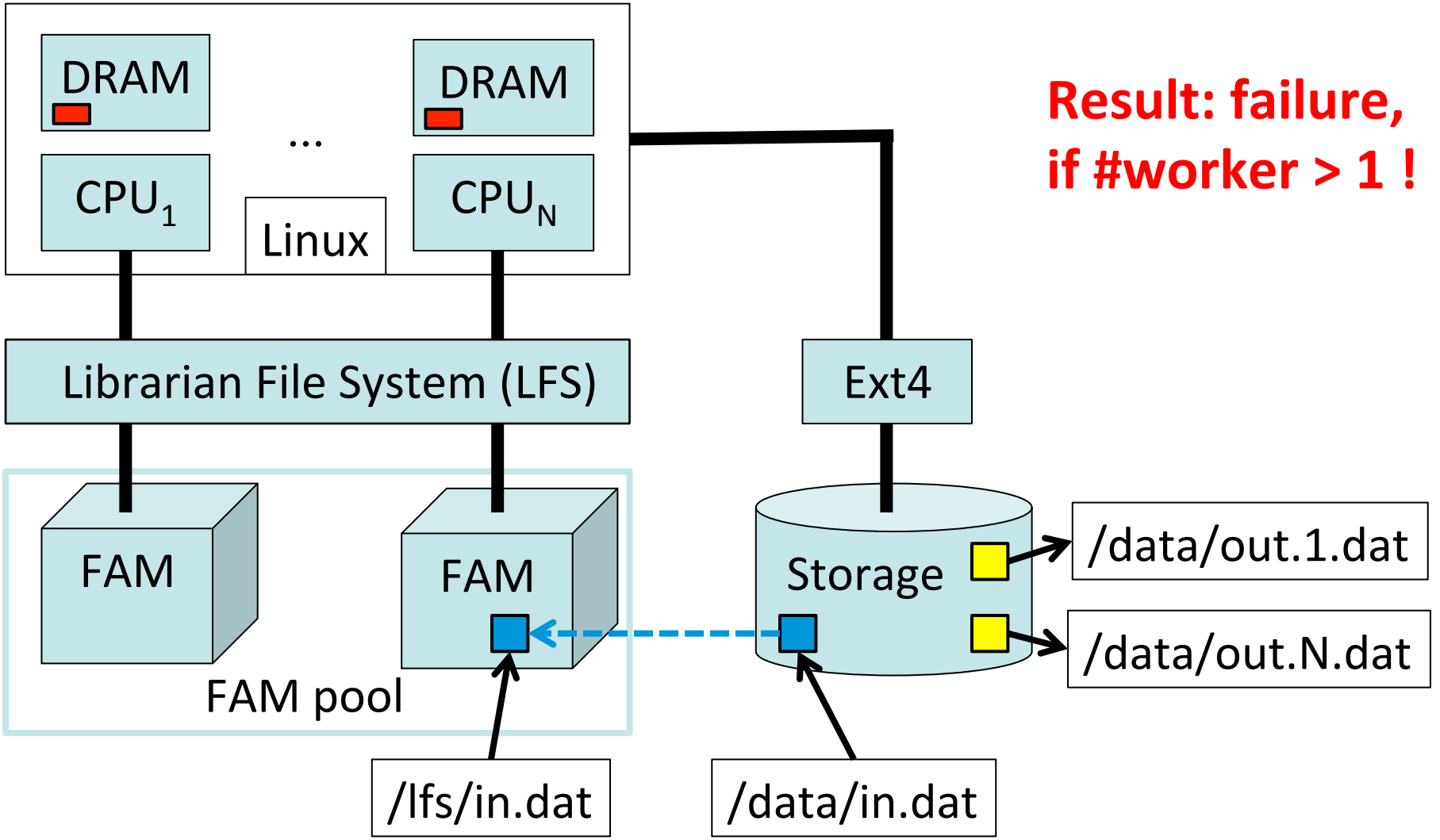
- WordCount1000: *n*·32 GB text files, 1000 unique words
- WordCountCC: *n*·19 GB text files (gzip-compressed), huge number of unique words
- 10 Gb/s TCP network between *n* Nodes



T. Bingmann: Scalable String and Suffix Sorting, PhD Thesis, KIT, 2017.

Sandbox + Thrill

- Test: counting words in textfile using “word_count_simple.cpp”



Result: failure, if #worker > 1 !

Thrill: transferring data into DIA

- `Thrill::ReadLines(…, "lfs/in.dat")`
  - Each of *n* workers reads a part of the textfile `in.dat`.

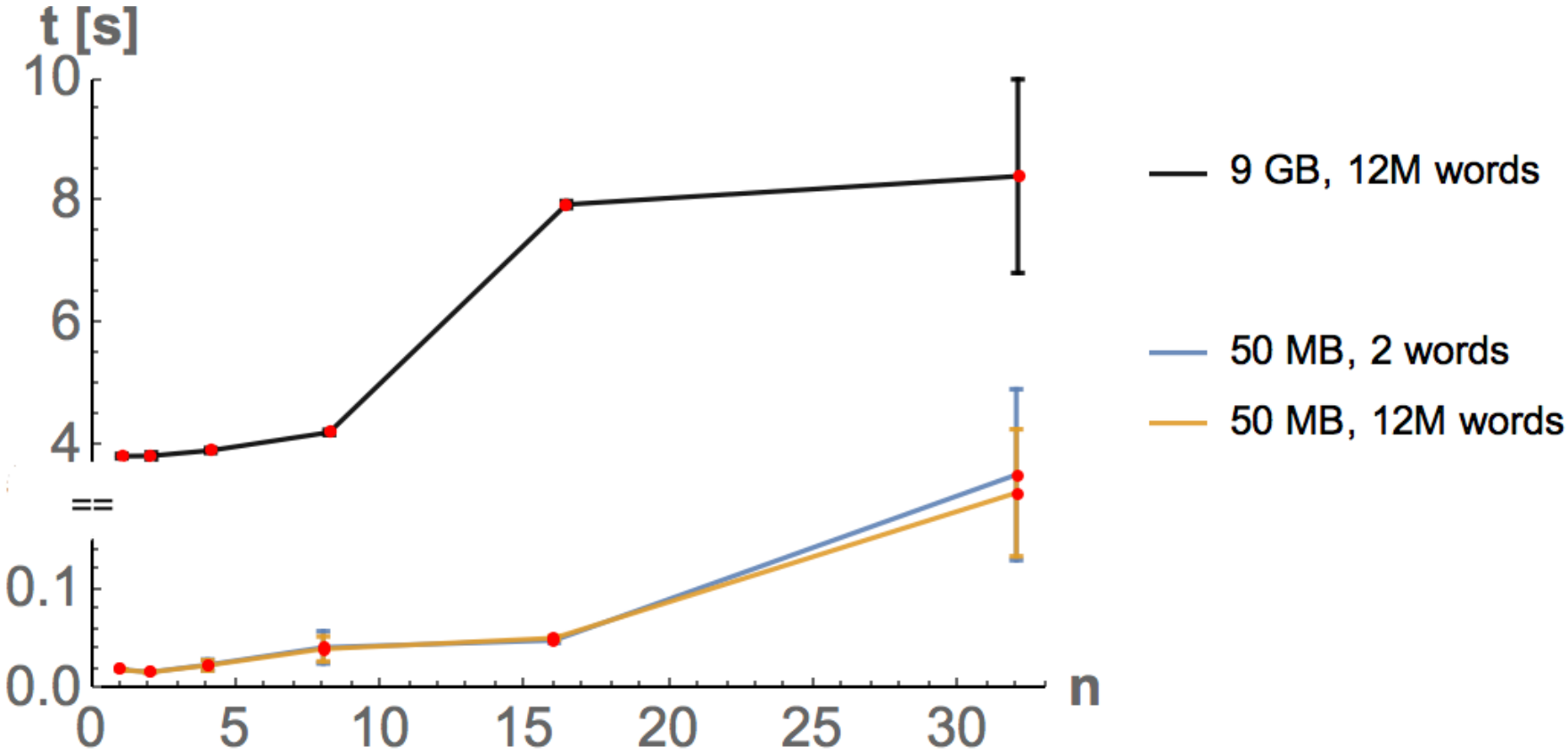
  ⇒ **Race conditions**, if n > 1.

- `Thrill::EqualToDIA`
  - Distributes **vector components** to *n* workers
  - Main thread (initialization)
    - Creating *n*-dim vector by splitting memory-mapped textfile into *n* pieces.
  - Worker threads
    - Phase 1: **EqualToDIA** for spreading vector over workers and generating DIA.
    - Phase 2: **ReduceByKey** for counting number of words.

  ⇒ **No race conditions** ✓

Sandbox + Thrill

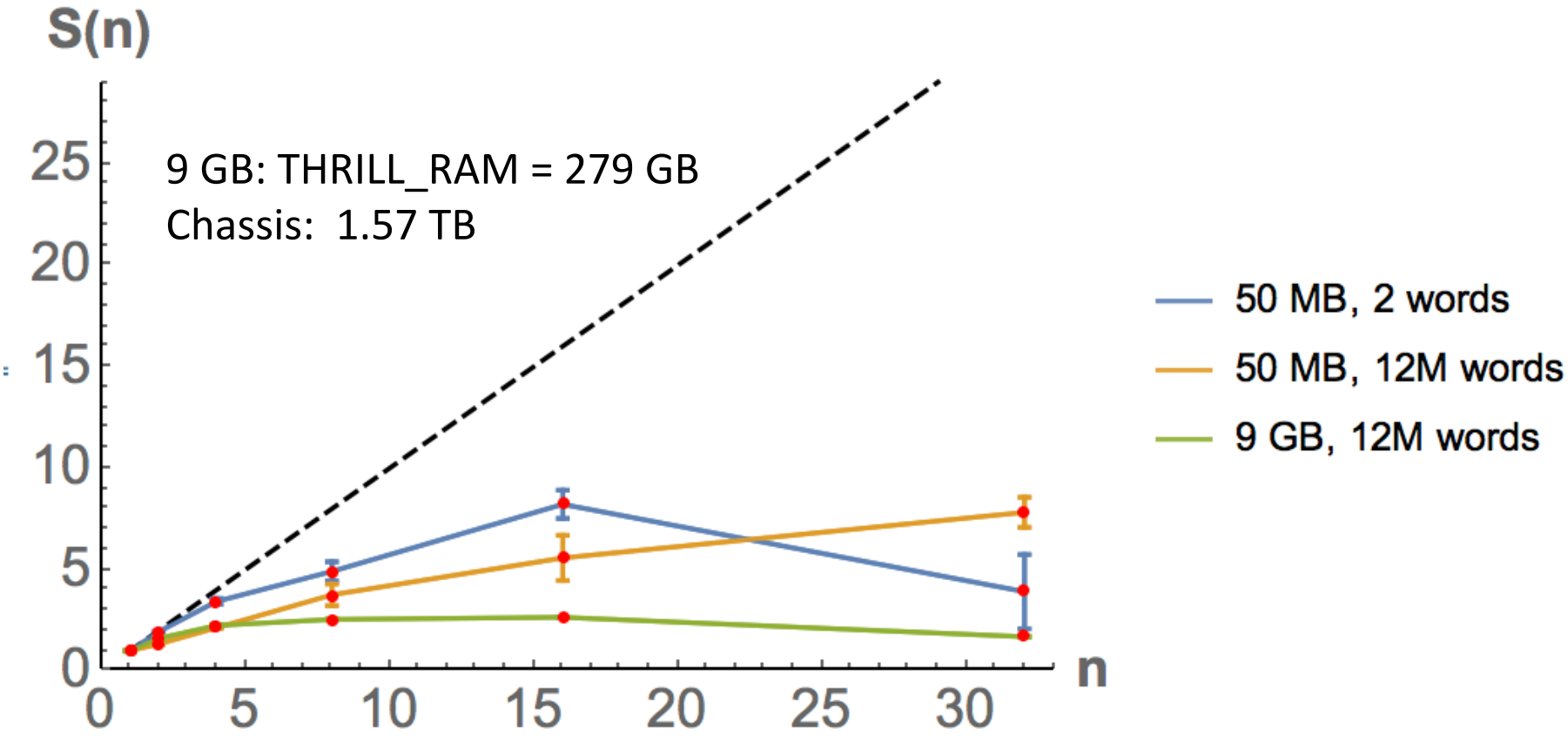
- Word counting: textfile with randomly distributed words
- **Phase 1**: time for distributing the vector components (EqualToDIA)



E. Buchholz, H.H. (HTW Berlin)
with strong support from R. Craig, B. Hayes, B. Tanner, H. Schultze (HPE)

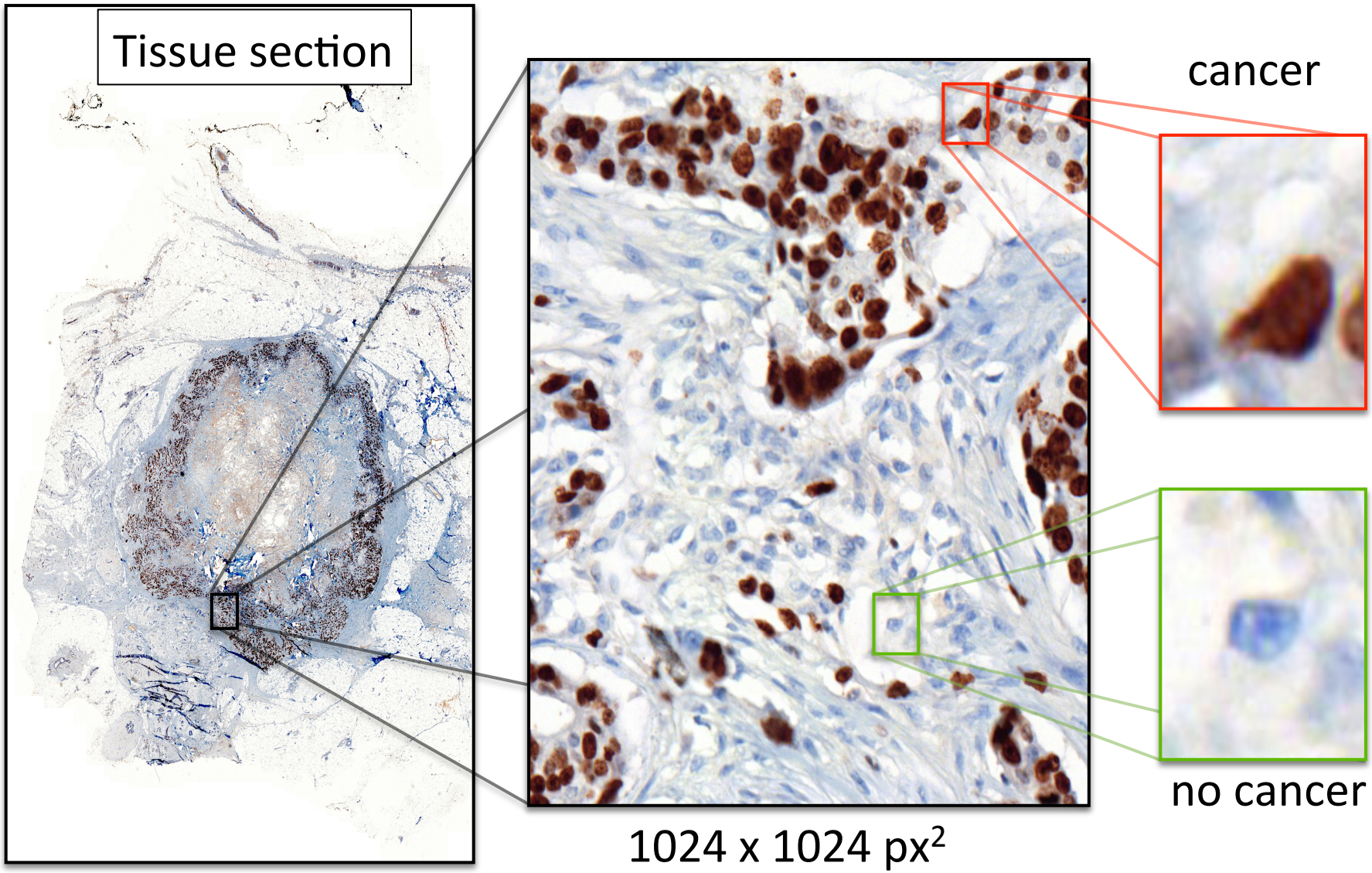
Sandbox + Thrill

- Word counting: textfile with randomly distributed words
- **Phase 2**: **Speedup** = Time(1 worker) **/ Time(n workers)**



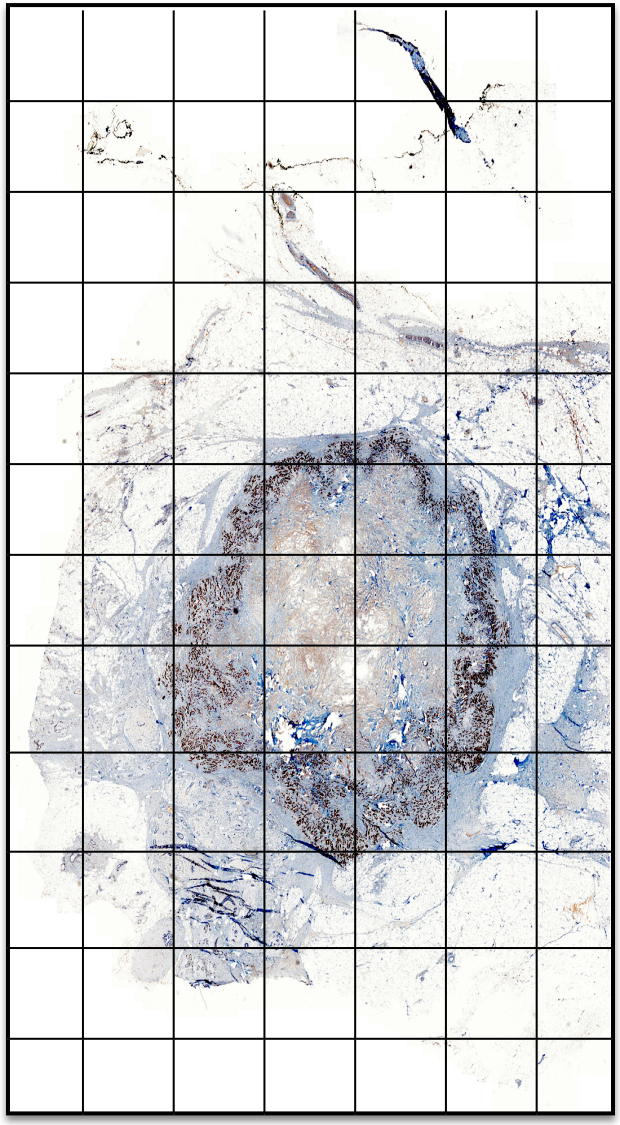
E. Buchholz, H.H. (HTW Berlin)
with strong support from R. Craig, B. Hayes, B. Tanner, H. Schultze (HPE)

Imaging in Digital Pathology



M. Strutz, H.H., P. Hufnagl: *A Gray-box Testing Method for Divide&Conquer in Image Processing*
IEEE Big Data, Dec. 9.-12. 2019, Los Angeles.

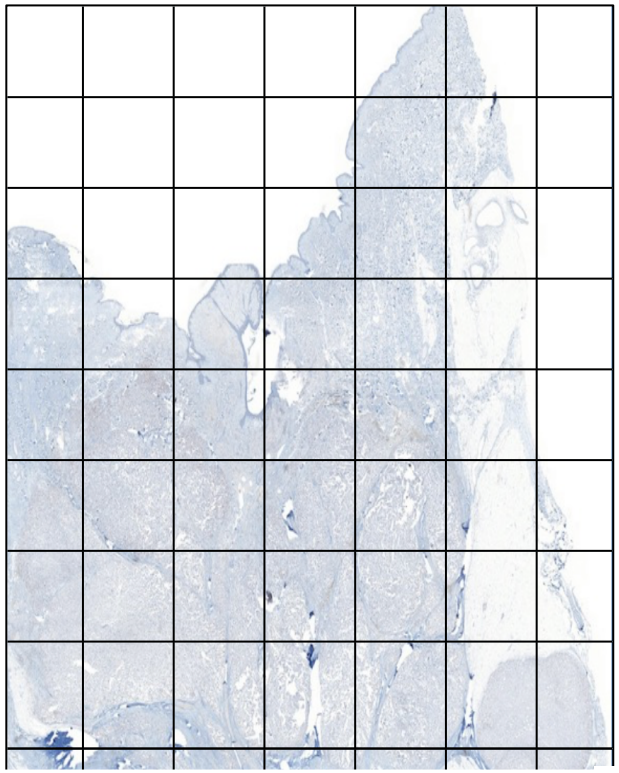
Divide & Conquer in image processing

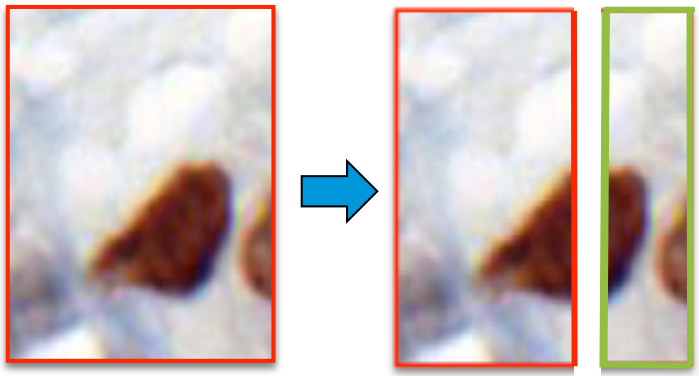


Splitting image into tiles

Running ***Ki67*-framework** on each tile

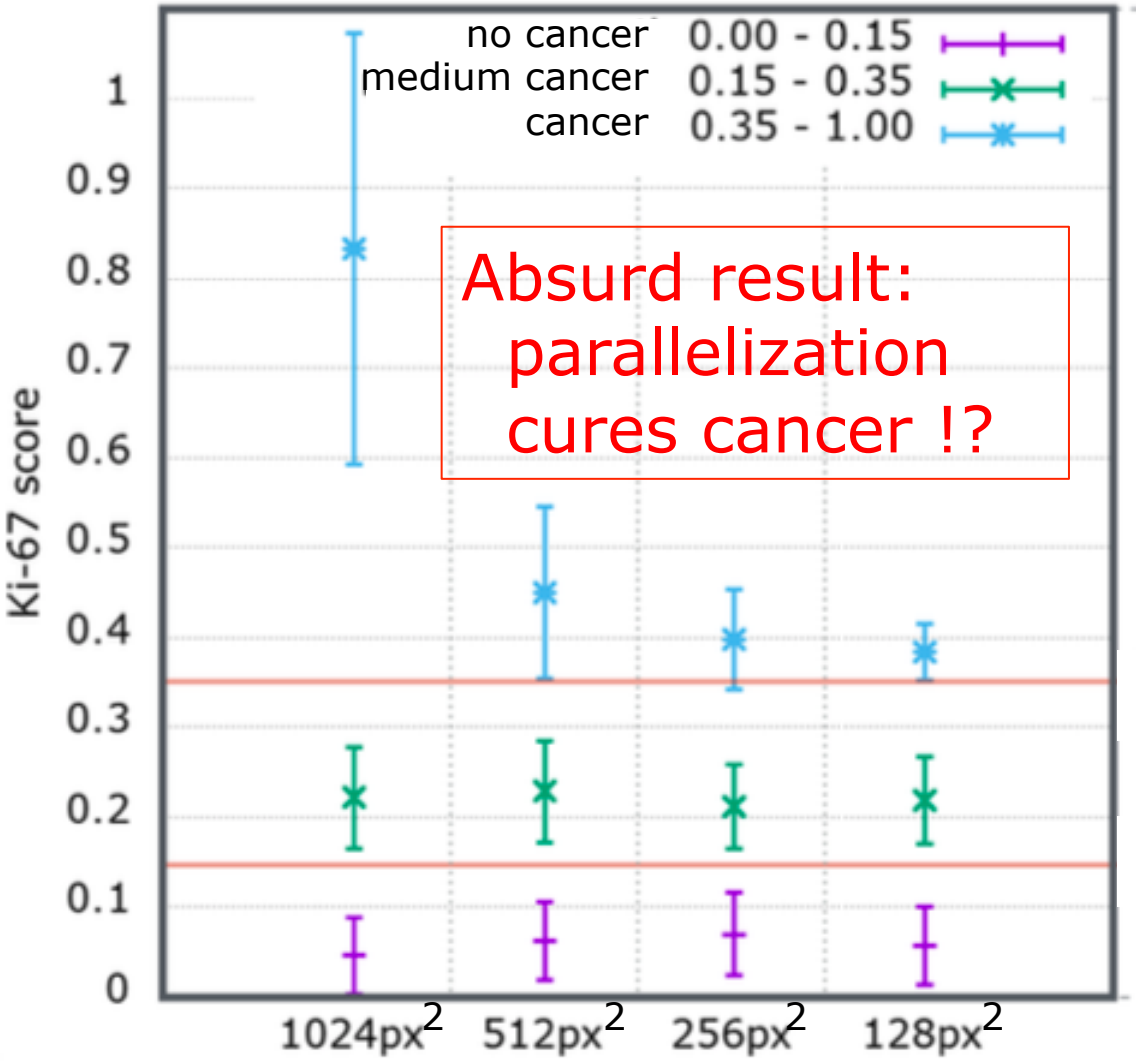
Divide & Conquer in image processing





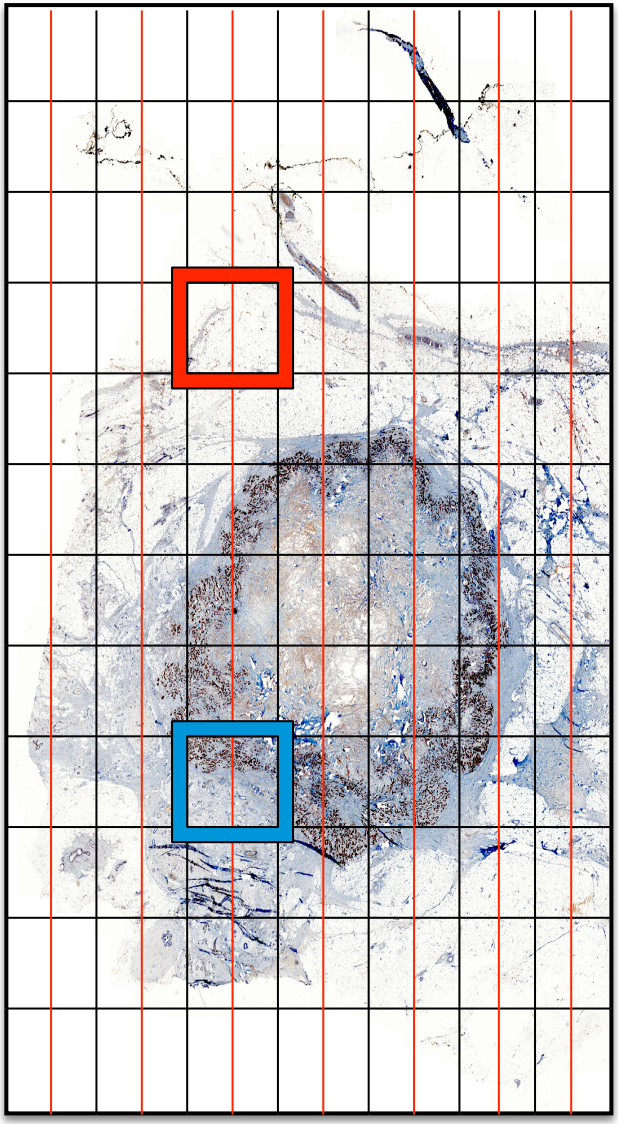
Splitting image into tiles

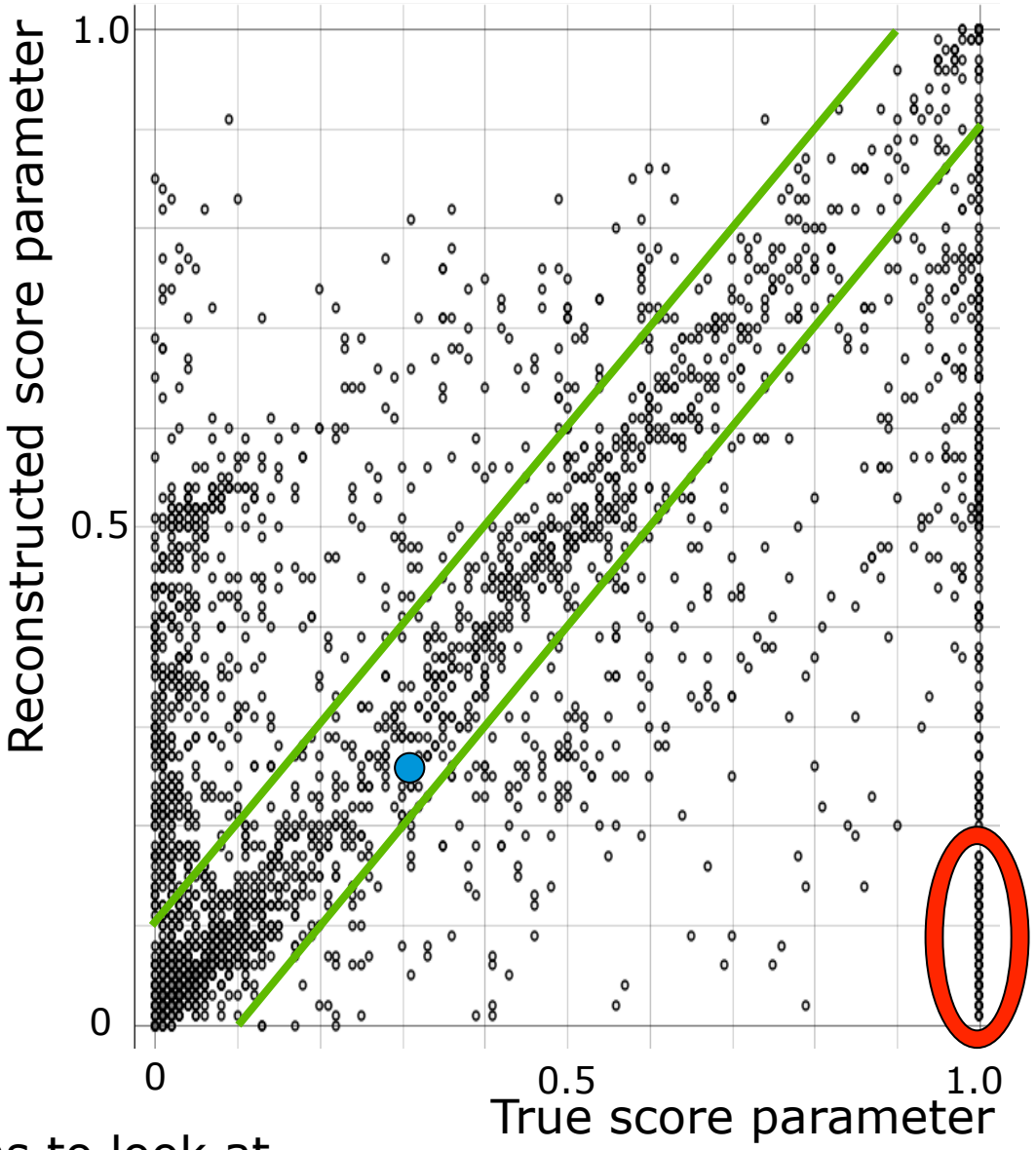
Running ***Ki67*-framework** on each tile



M. Strutz, H.H., A. Streit: *The Challenge of a Strong Speed-Up of a Bio-medical Big Data Application*, IEEE Big Data, Dec. 10.-13. 2018, Los Angeles.

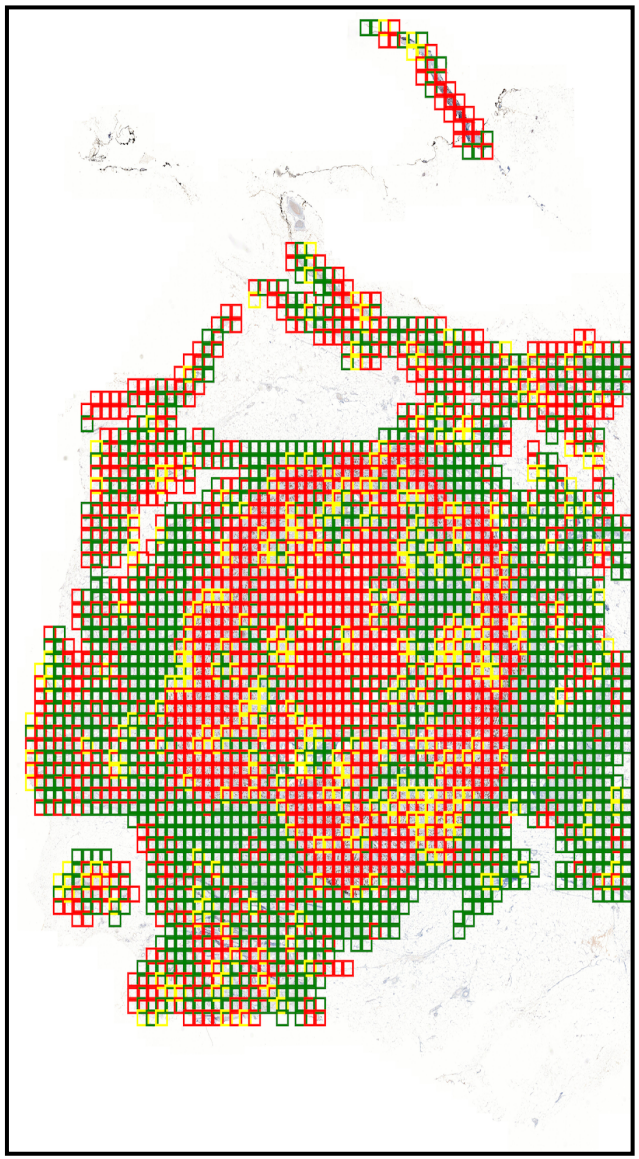
Dioride & Conquer in image processing



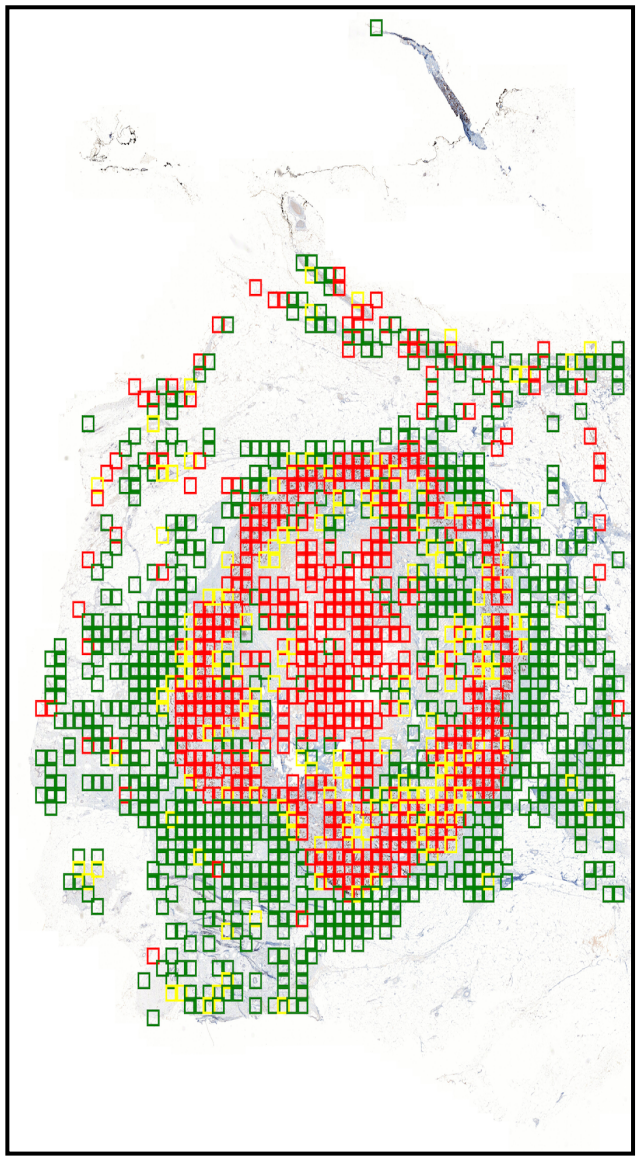


"Divide": provides new dimensions to look at.

Divide & Conquer in image processing



all tiles



“stable” tiles

Summary and outlook

- Data challenges @ SKA
  - Data irreversibility
  - Huge data objects
- Memory-based computing
  - First tests on the HPE Sandbox using the Big Data framework Thrill
  - Next step: exploring a selected astronomical workflow (CASA?)
    - Open standards for pooled memory, e.g. Open-SHMEM, Gen-Z, OpenFAM
- Divide&Conquer in image processing
  - **Fundamental assumption** of Divide&Conquer: partial problems are of the same type as the problem itself
  - **Stability conditions for divide steps** ⇒ identification of regions where the outcome of a workflow is not reliable
  - Extending Thrill
    - Generalizing EqualToDIA from vectors to *n*-dim **tensors**
    - `DIA`-operators for **overlapping subsets** of data