Lecture 3: More about the MapReduce Programming Model

COSC 526: Introduction to Data Mining

Spring 2020

THE UNIVERSITY OF TENNESSEE KNOXVILLE

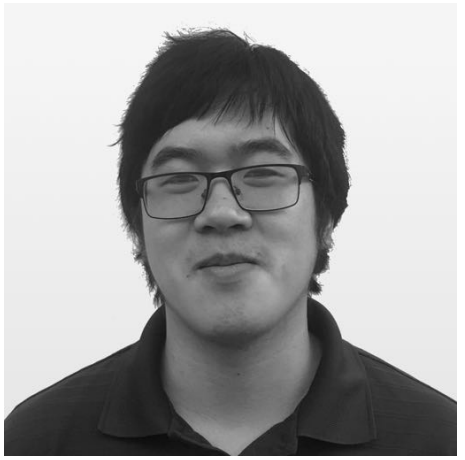
BIG ORANGE. BIG IDEAS.®

Instructors:

Michela Taufer

Danny Rorabaugh

GRA:



Nigel Tan

Today Outline

- Continue to learn about the MapReduce programming model:
  - Programming model: optimizations
  - Algorithms: different approaches to solve the same problem
  - Benchmarks: concrete examples of MapReduce applications
  - Data: Where is the data kept?
- Continue to build our expertise with Jupyter and Python
  - Sequential manipulation of a classical in literature
  - Visualization of statistics

Optimizations

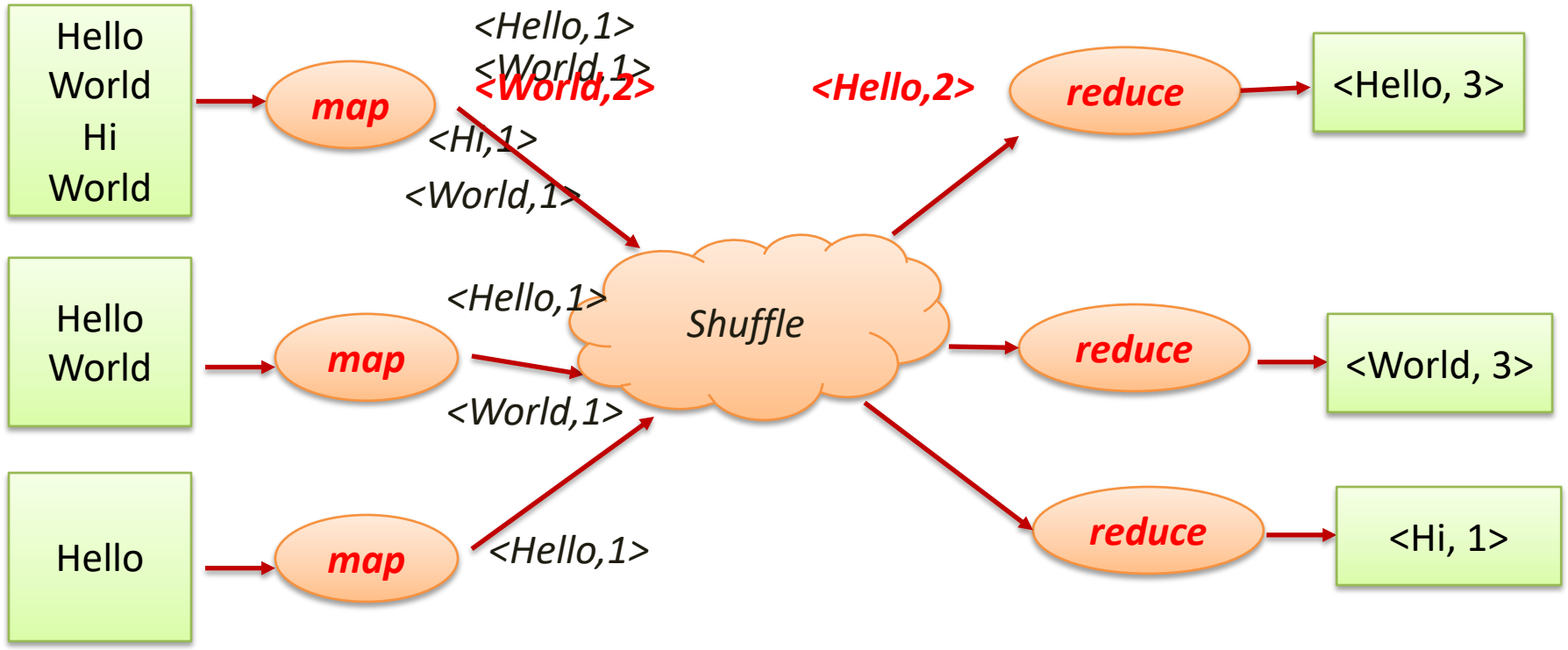
More than “Map + Reduce”

- Canonical MapReduce processing workflow:
  - Map + Reduce
- Variations:
  - Map + Combiners + Partitioners + Reduce (in Apache Hadoop and Mimir implementations)
  - Automatic implementation of combiners on the map side (in Apache Spark - more in the next lecture)

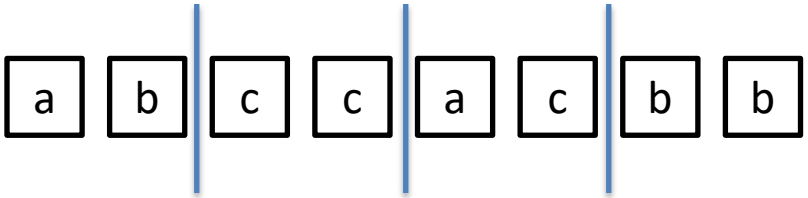
Combining <key,value> Pairs

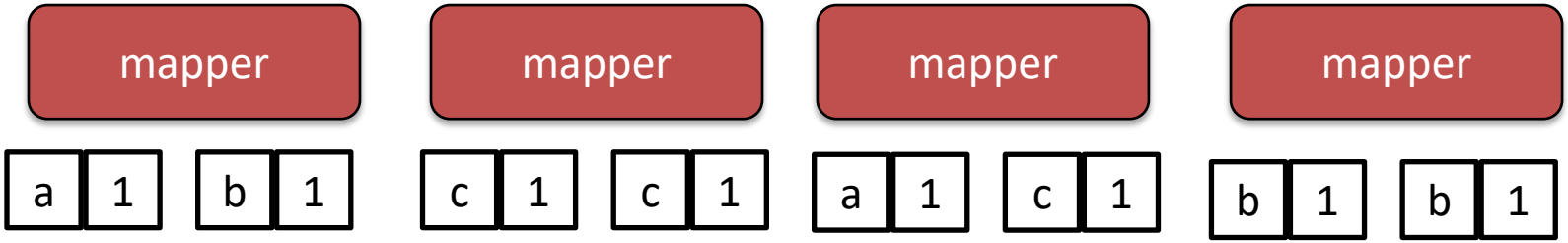
- Merge <key,value> pairs with the same key **before** shuffle
- Merge <key,value> pairs with the same key **after** shuffle

Wordcount example:

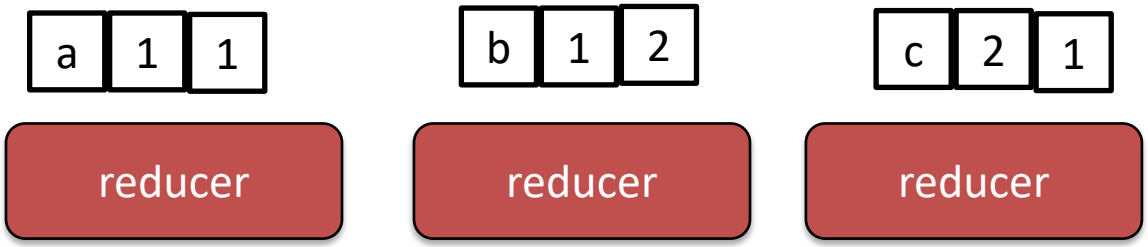


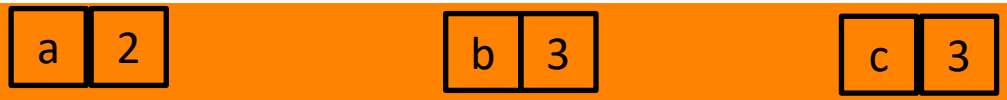
More than
Map + Reduce





Shuffle and sort: aggregate values by key



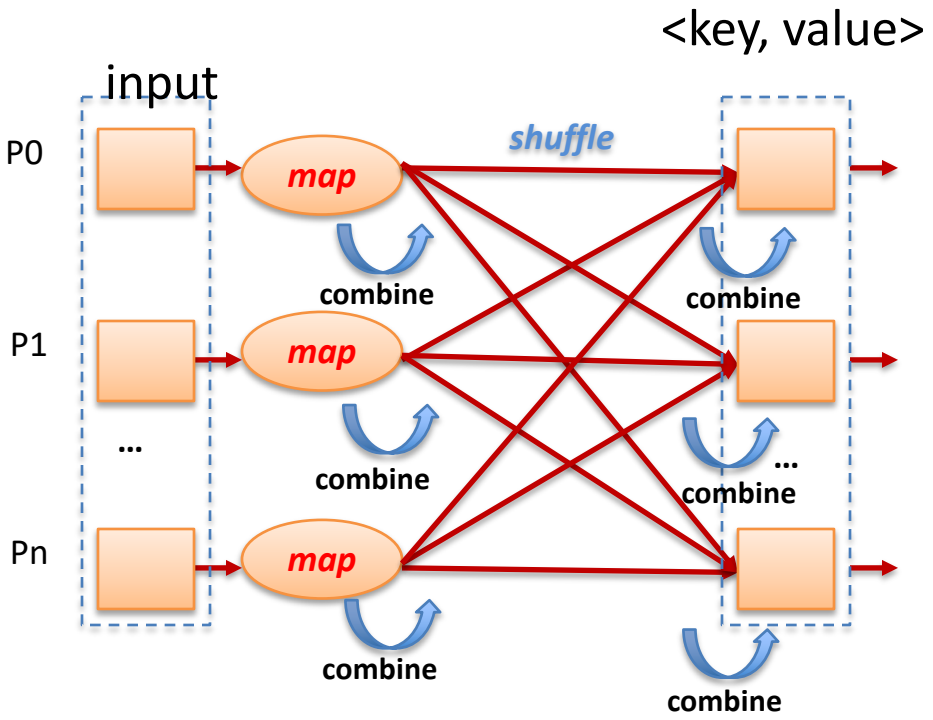


More than M + R: Combiners

- Combiners allow for local aggregation before the shuffle and sort phase
  - "mini-reducers" that take place locally, on the output of the mappers
- Strengths:
  - Number of intermediate key-value pairs moved among reducers to be **at most the number of unique words** in the collection times the number of mappers
- Weaknesses:
  - Reducers and combiners are not interchangeable unless the operation is both associative and commutative
  - Operation performed in isolation and therefore does not have access to intermediate output from other mappers

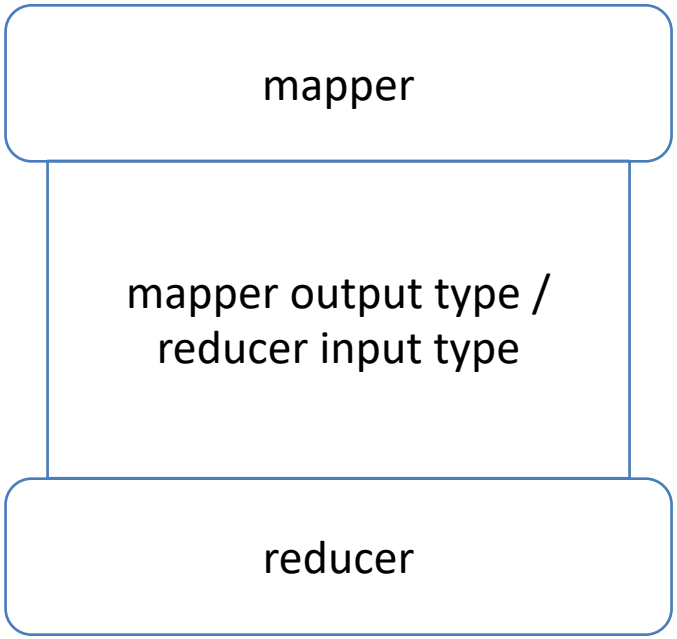
Combining <key,value> Pairs

- Combiner operations:
  - Merge <key,value> pairs with the same key **before** shuffle
  - Merge <key,value> pairs with the same key **after** shuffle
- Application dependent:
  - Wordcount → **YES**
  - Join → **NO**



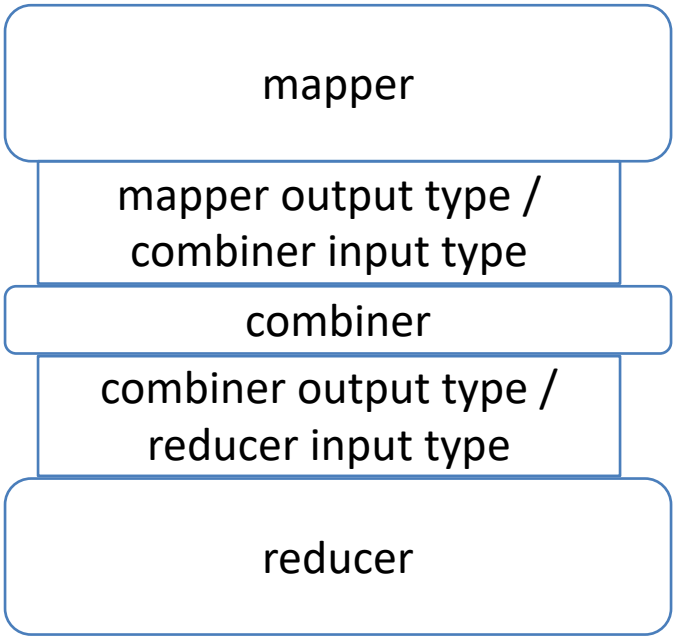
More than M + R: Combiners Constraints

- Combiners in, for example, Apache Hadoop cannot change the correctness of the MapReduce algorithm
- Combiners must have same input and output key-value types



More than M + R: Combiners Constraints

- Combiners in, for example, Apache Hadoop cannot change the correctness of the MapReduce algorithm
- Combiners must have same input and output key-value types

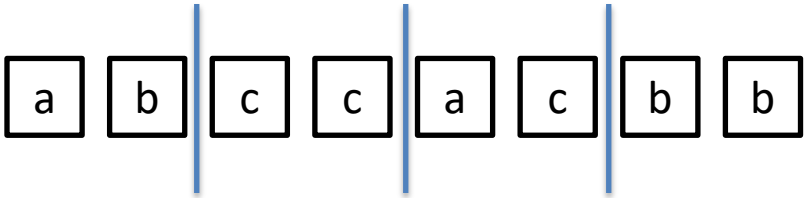


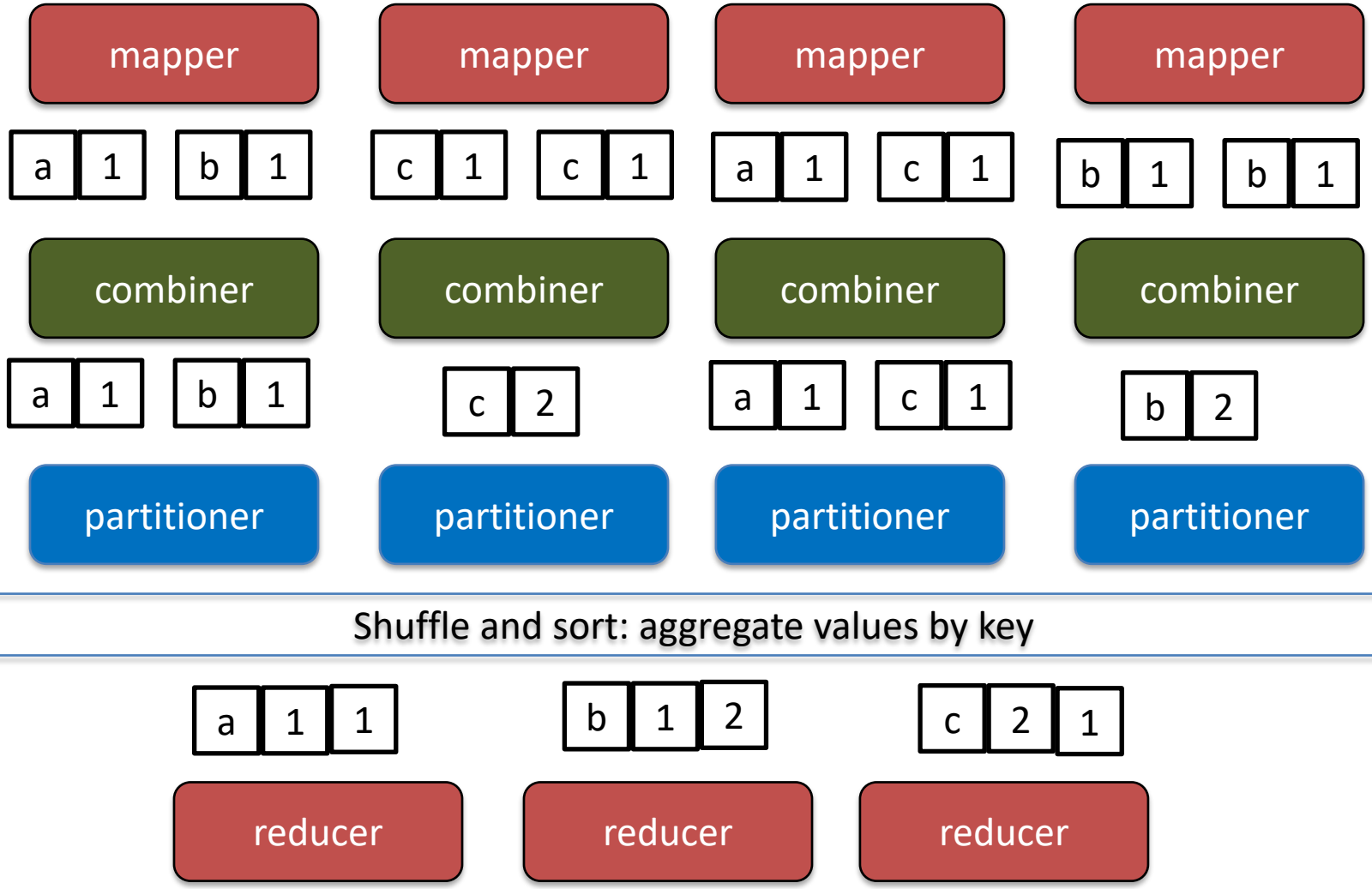
mapper output type
==
combiner input type
==
combiner output type
==
reducer input type

More than M + R: Partitioners

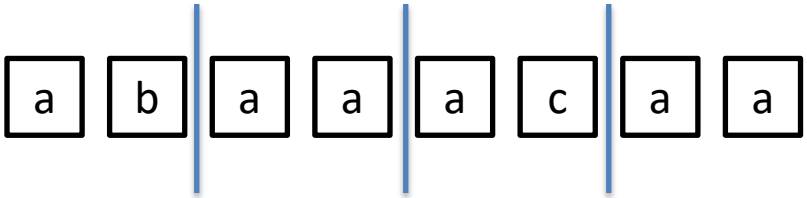
- Divide up the intermediate key space and **assign intermediate key-value pairs to reducers**
- Example of simple partitioner:
  - Compute hash value of a key
  - Take the mod of the value with the number of reducers
- Strengths:
  - Assign approximately the same number of keys to each reducer (dependent on the quality of the hash function)
- Weaknesses:
  - Ignore the value and different keys may have different numbers of associated values causing imbalance in the amount of data associated with each key

More than
Map + Reduce





More than
Map + Reduce



mapper | mapper | mapper | mapper

a 1 | b 1 | a 1 | a 1 | a 1 | c 1 | a 1 | a 1

combiner | combiner | combiner | combiner

a 1 | b 1 | a 2 | a 1 | c 1 | a 2

partitioner | partitioner | partitioner | partitioner

Shuffle and sort: aggregate values by key

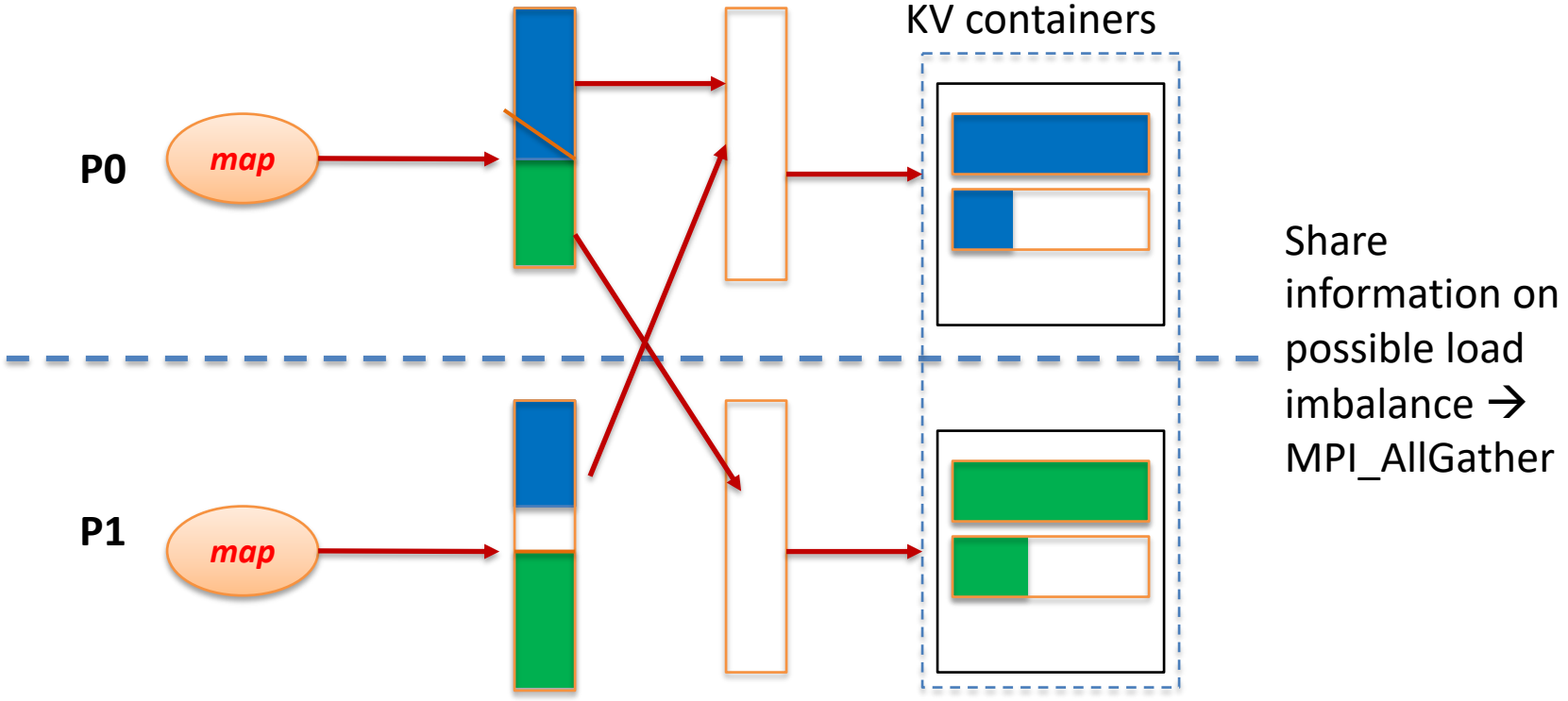
a 1 2 1 2 | b 1 1 | c 1

reducer | reducer | reducer

a 6 | b 1 | c 1

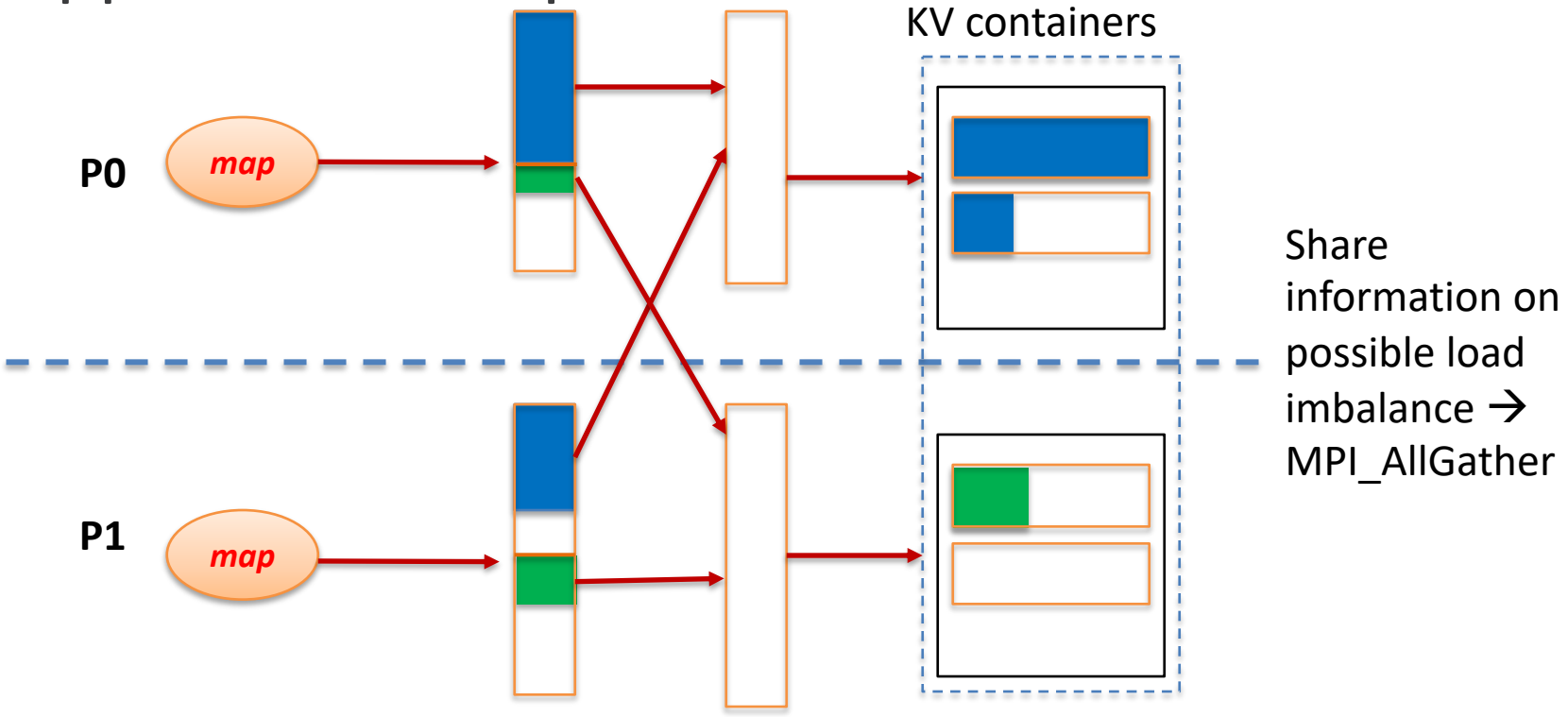
Dynamic Partitioning

- Perform KVs repartition once imbalance problem is detected in KVC
- Application independent



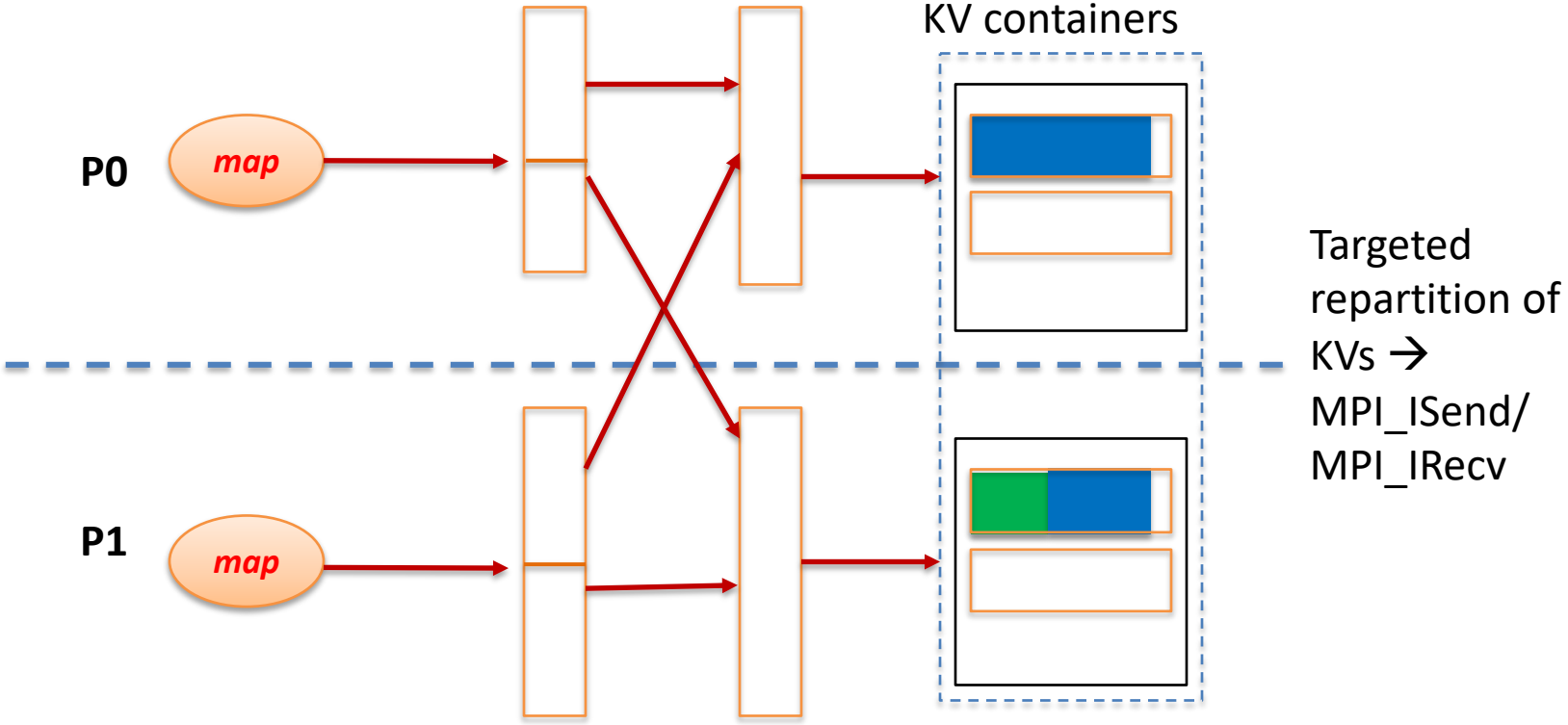
Dynamic Partitioning

- Perform KVs repartition once imbalance problem is detected in KVC
- Application independent



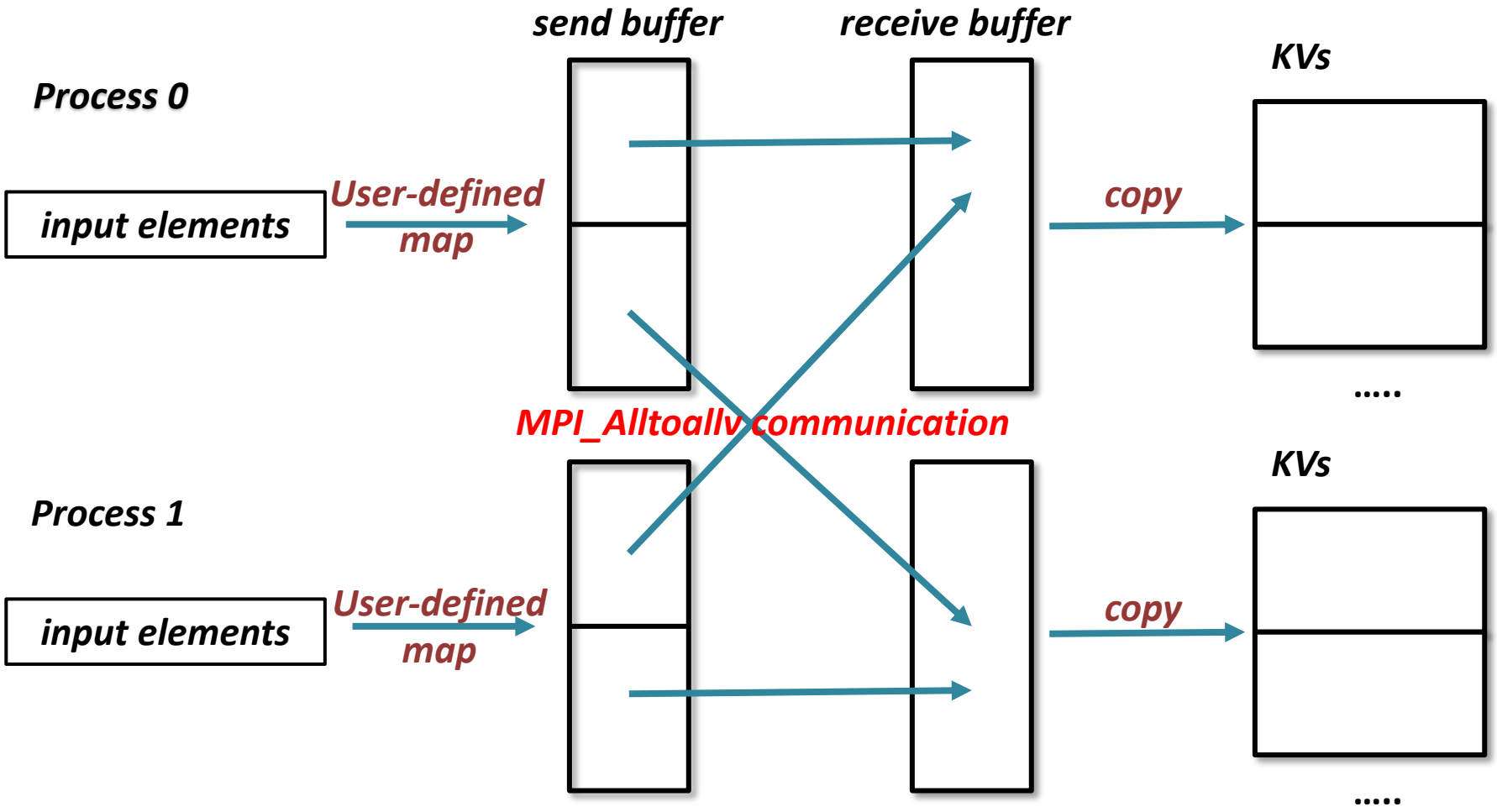
Dynamic Partitioning

- Perform KVs repartition once imbalance problem is detected in KVC
- Application independent

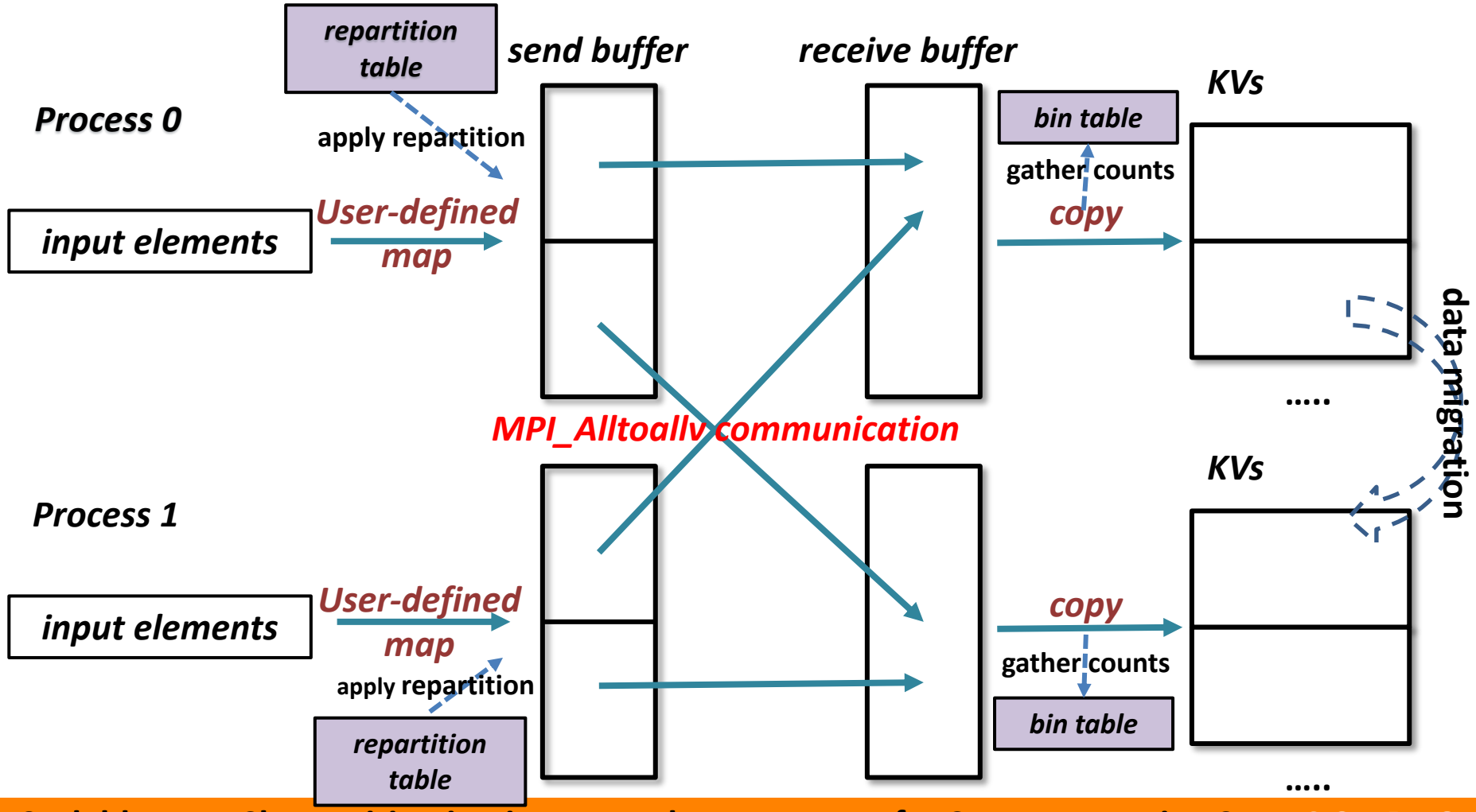


Scalable Data Skew Mitigation in MapReduce over MPI for Supercomputing Systems. T. Gao, Y. Guo, B. Zhang, P. Cicotti, Y. Lu, P. Balaji, and M. Taufer. *IEEE TPDS*, 2019.

Frameworks: Mimir

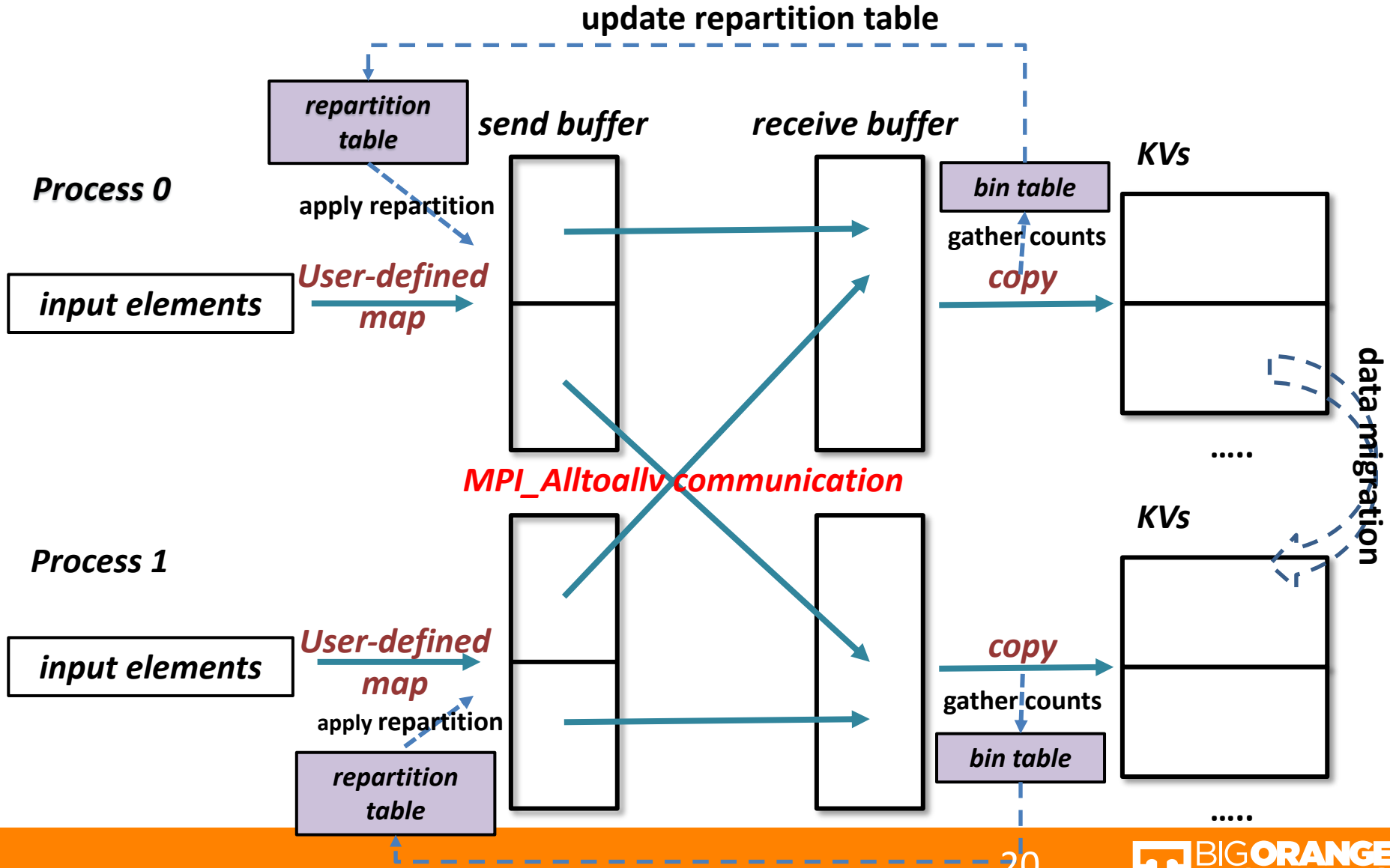


Frameworks: Mimir



Scalable Data Skew Mitigation in MapReduce over MPI for Supercomputing Systems. T. Gao, Y. Guo, B. Zhang, P. Cicotti, Y. Lu, P. Balaji, and M. Taufer. *IEEE TPDS*, 2019.

Frameworks: Mimir



WordCount in MapReduce (Review)

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term t ∈ doc d do
4:             Emit(term t, count 1)
```

```
1: class Reducer
2:     method Reduce(term t, counts [c1, c2, ...])
3:         sum ← 0
4:         for all count c ∈ counts [c1, c2, ...] do
5:             sum ← sum + c
6:         Emit(term t, count sum)
```

Using Associative Array

Associative array to aggregate term counts on a per-document basis

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term t ∈ doc d do
4:             Emit(term t, count 1)
```

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         H ← new AssociativeArray
4:         for all term t ∈ doc d do
5:             H{t} ← H{t} + 1
6:         for all term t ∈ H do
7:             Emit(term t, count H{t})
```

In-mapper Combining

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         H ← new AssociativeArray
4:         for all term t ∈ doc d do
5:             H{t} ← H{t} + 1
6:         for all term t ∈ H do
7:             Emit(term t, count H{t})
```

```
1: class Mapper
2:     method Initialize
3:         H ← new AssociativeArray
4:     method Map(docid a, doc d)
5:         for all term t ∈ doc d do
6:             H{t} ← H{t} + 1
7:     method Close
8:         for all term t ∈ H do
9:             Emit(term t, count H{t})
```

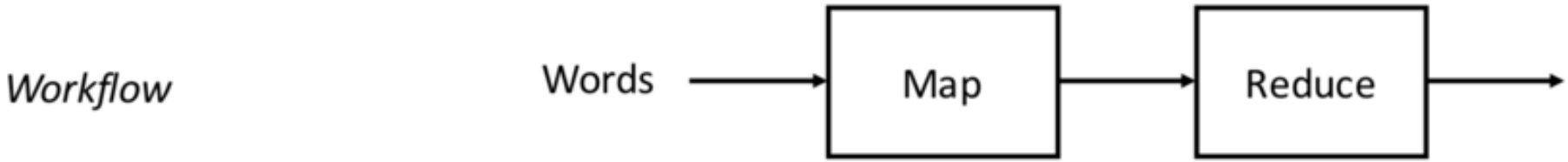
- Initialize an associative array for holding term counts
- Accumulate partial counts in associative array across multiple documents
- incorporate combiner functionality directly inside the mapper (in-mapper combining)

23

Benchmarks

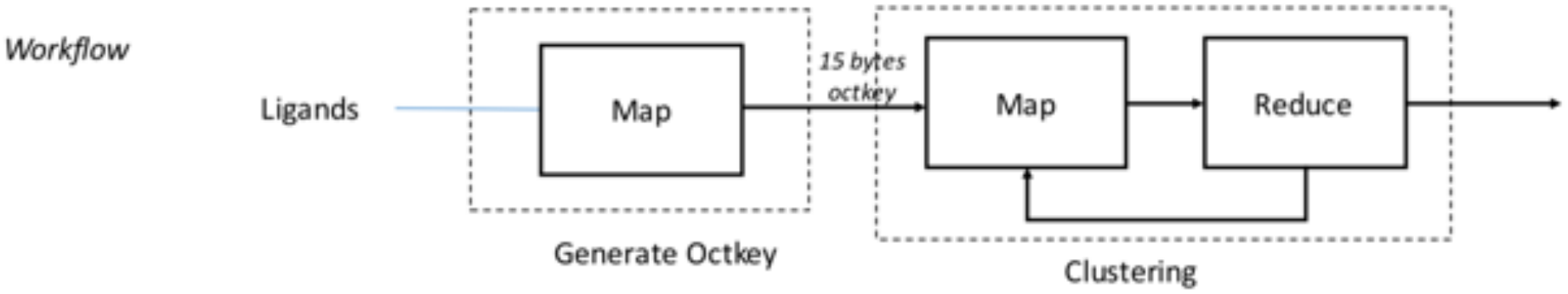
Benchmark: WordCount (WC)

- WC is a single-pass MapReduce application
- WC counts the number of occurrences of each unique word in given input files



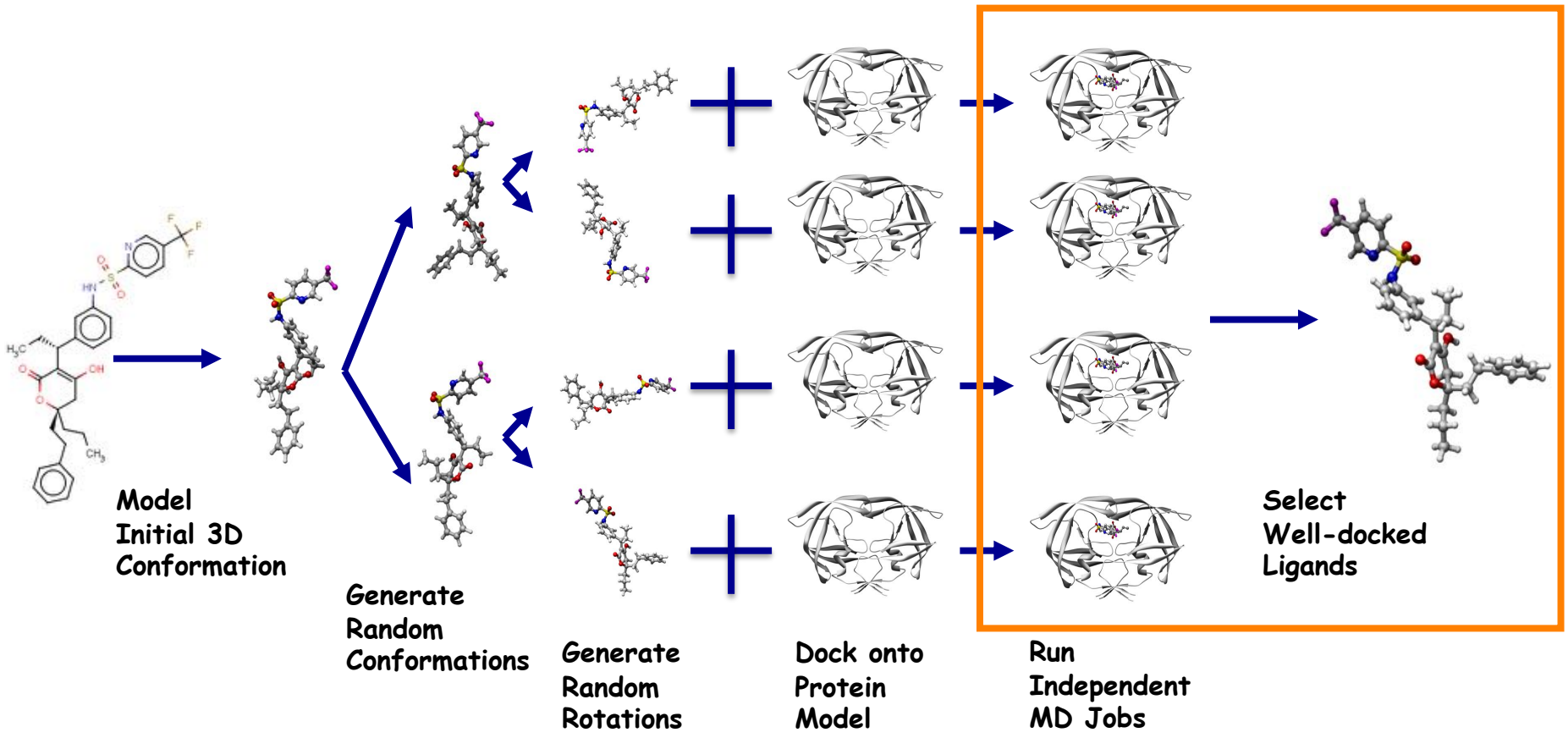
Benchmark: Octree clustering (OC)

- OC is a chain of MR jobs
  - Iterative MapReduce application with multiple MapReduce stages
- Cluster N-D points based on proximities



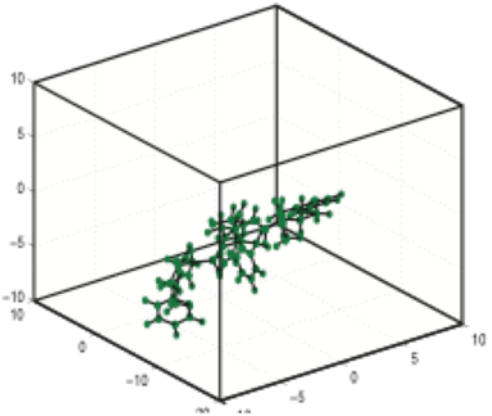
Limits

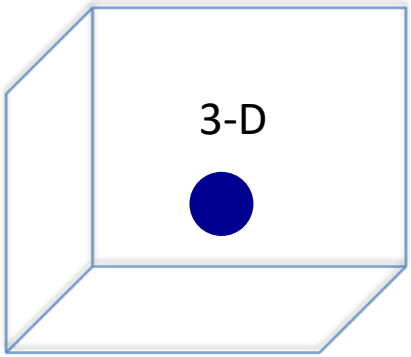
Protein-Ligand Docking

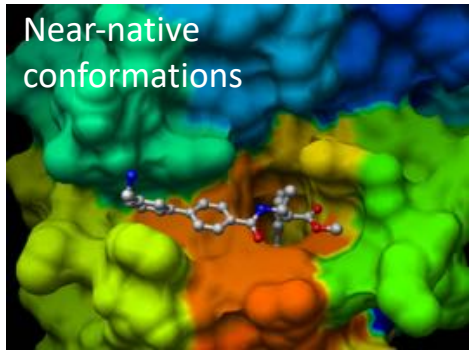


M. Taufer, R.S. Armen, J. Chen, P.J. Teller, and C.L. Brooks III. Computational Multi-Scale Modeling in Protein-Ligand Docking. *IEEE Engineering in Medicine and Biology Magazine*, 28(2): 58 – 69, 2009.

Protein-Ligand Docking







Data
High-dimensional

Metadata
Low-dimensional

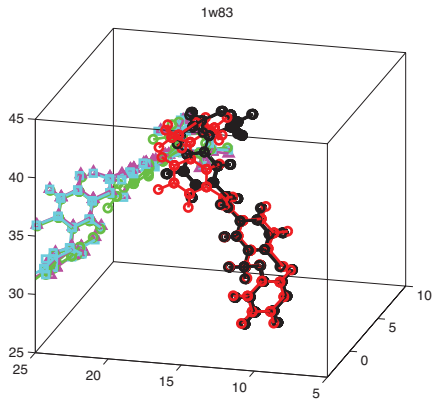
Results
Scientific knowledge

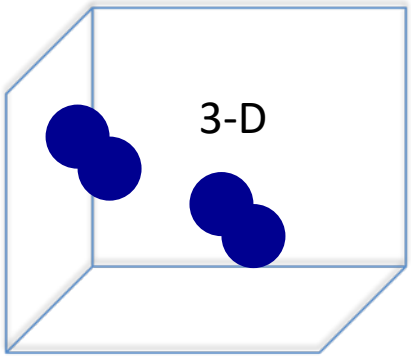
Extract geometries (metadata) concurrently

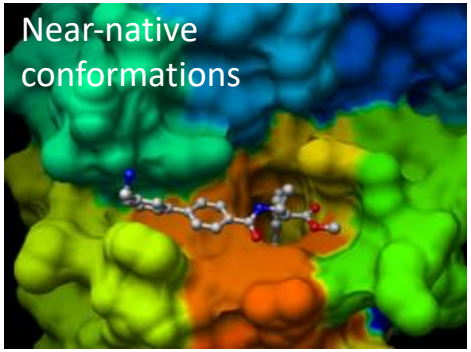
Search for densest metadata spaces

Enabling In-situ Data Analysis for Large Protein Folding Trajectory Datasets. B. Zhang, T. Estrada(#), P. Cicotti, and M. Taufer. *IPDPS, 2014*

Protein-Ligand Docking







Data
High-dimensional

Metadata
Low-dimensional

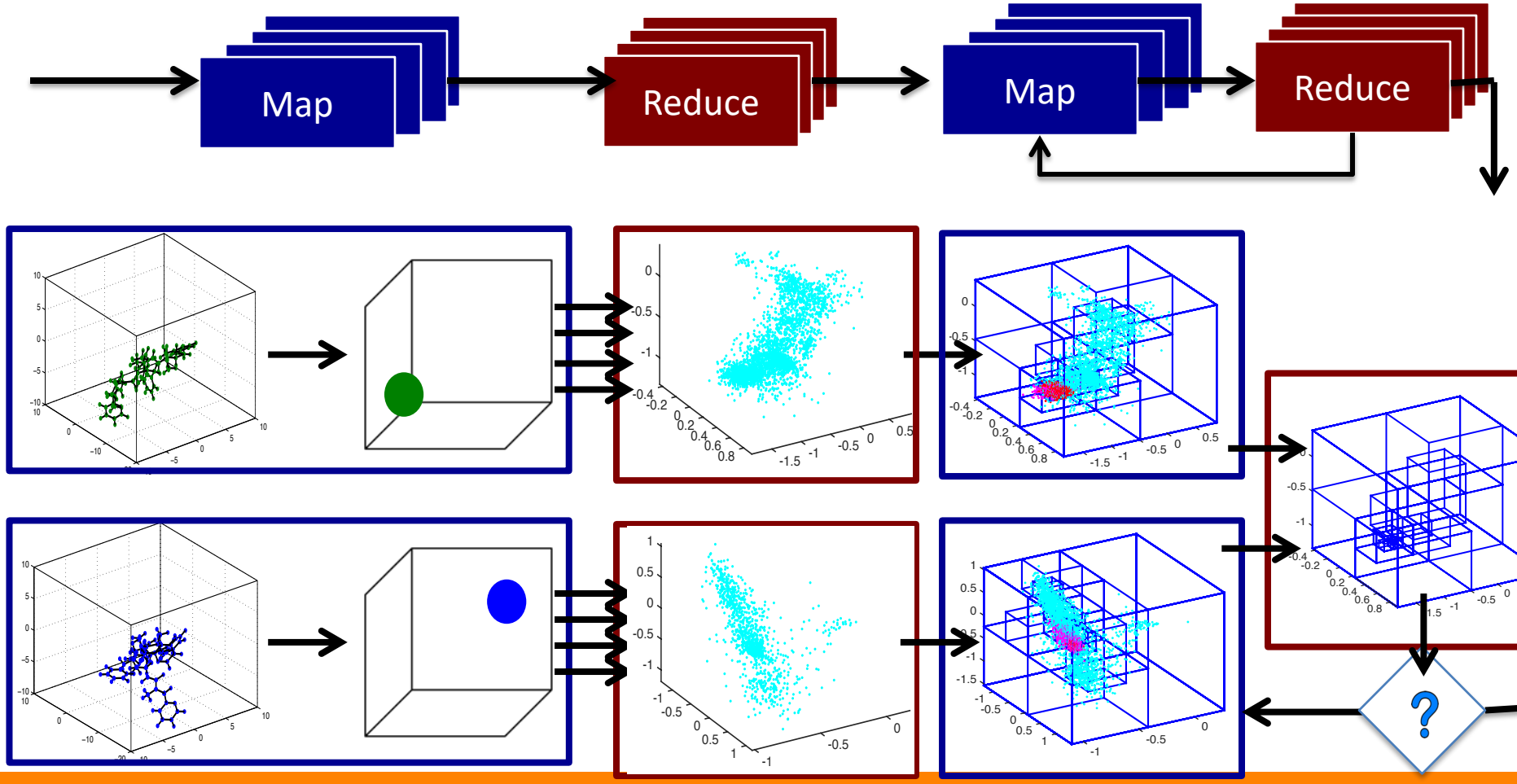
Results
Scientific knowledge

Extract geometries (metadata) concurrently

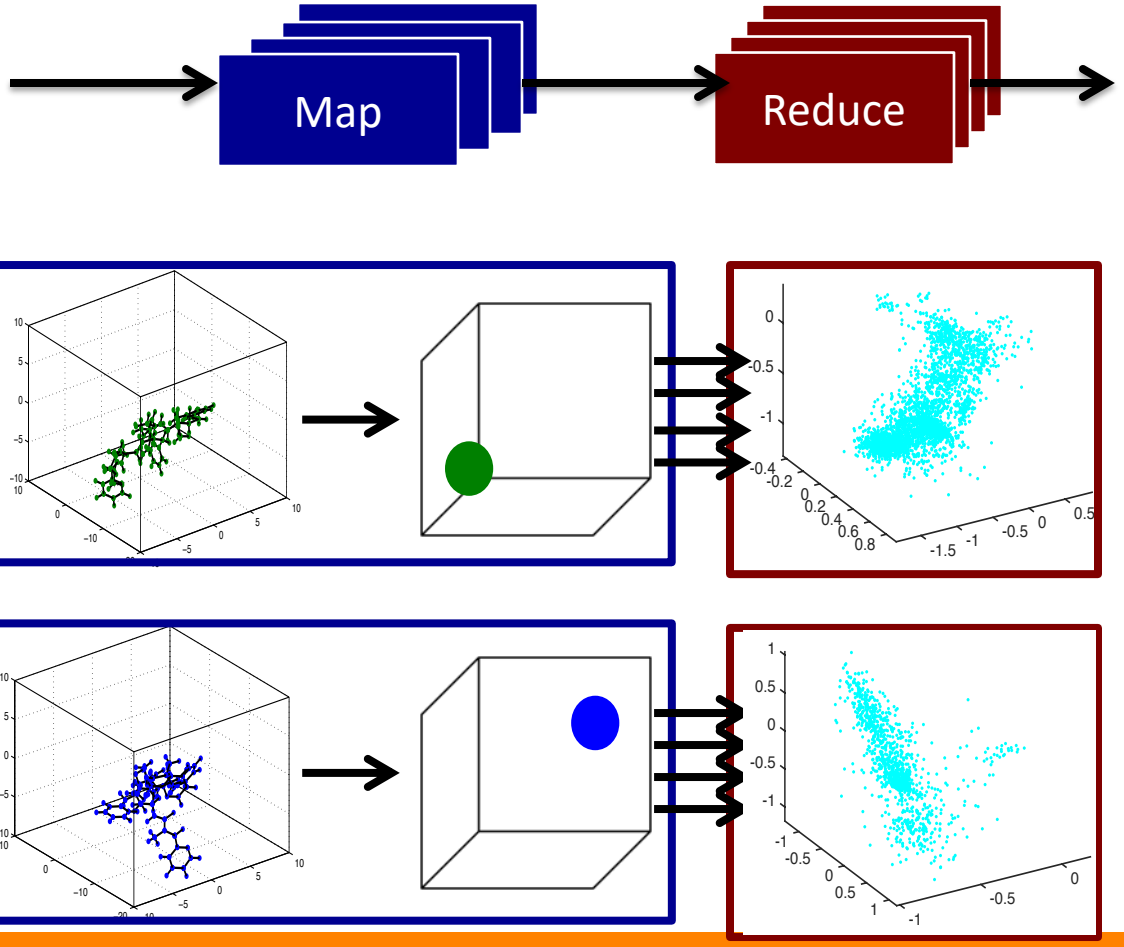
Search for densest metadata spaces

Enabling In-situ Data Analysis for Large Protein Folding Trajectory Datasets. B. Zhang, T. Estrada(#), P. Cicotti, and M. Taufer. *IPDPS, 2014*

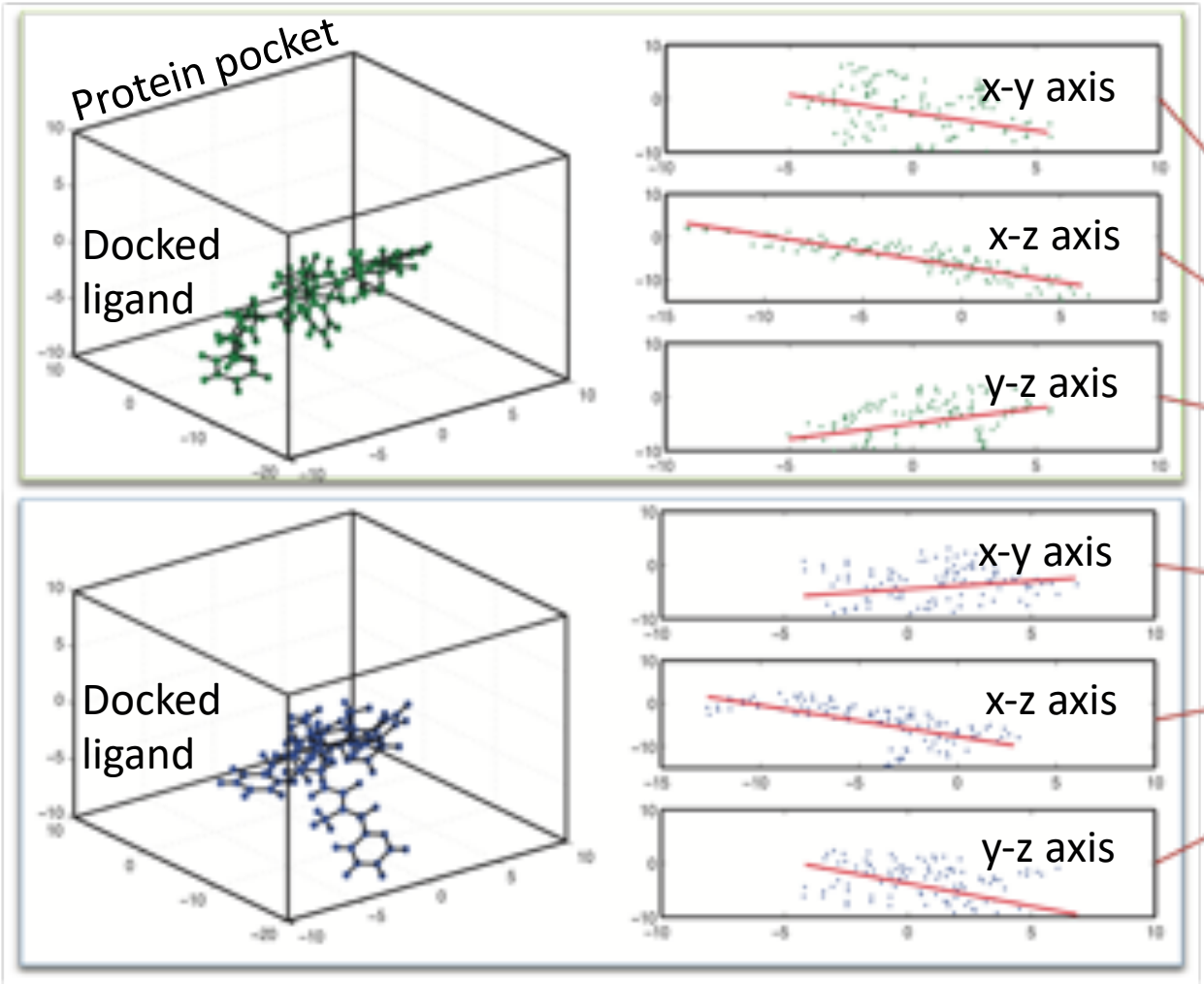
MapReduce-based Workflow



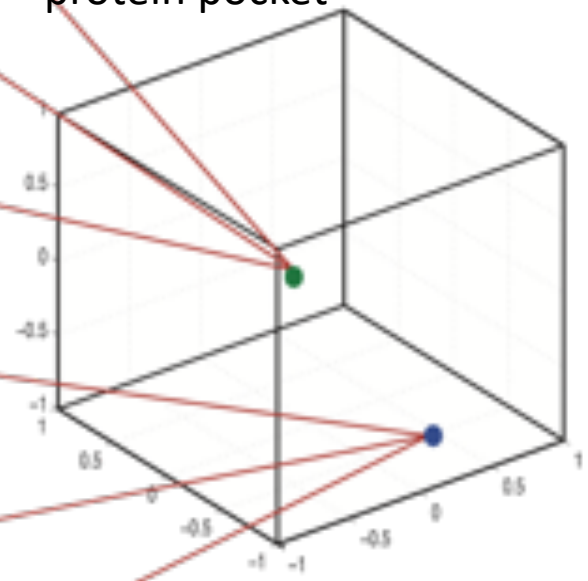
MapReduce-based Workflow



From 3D Atomic Structures to 3D Points

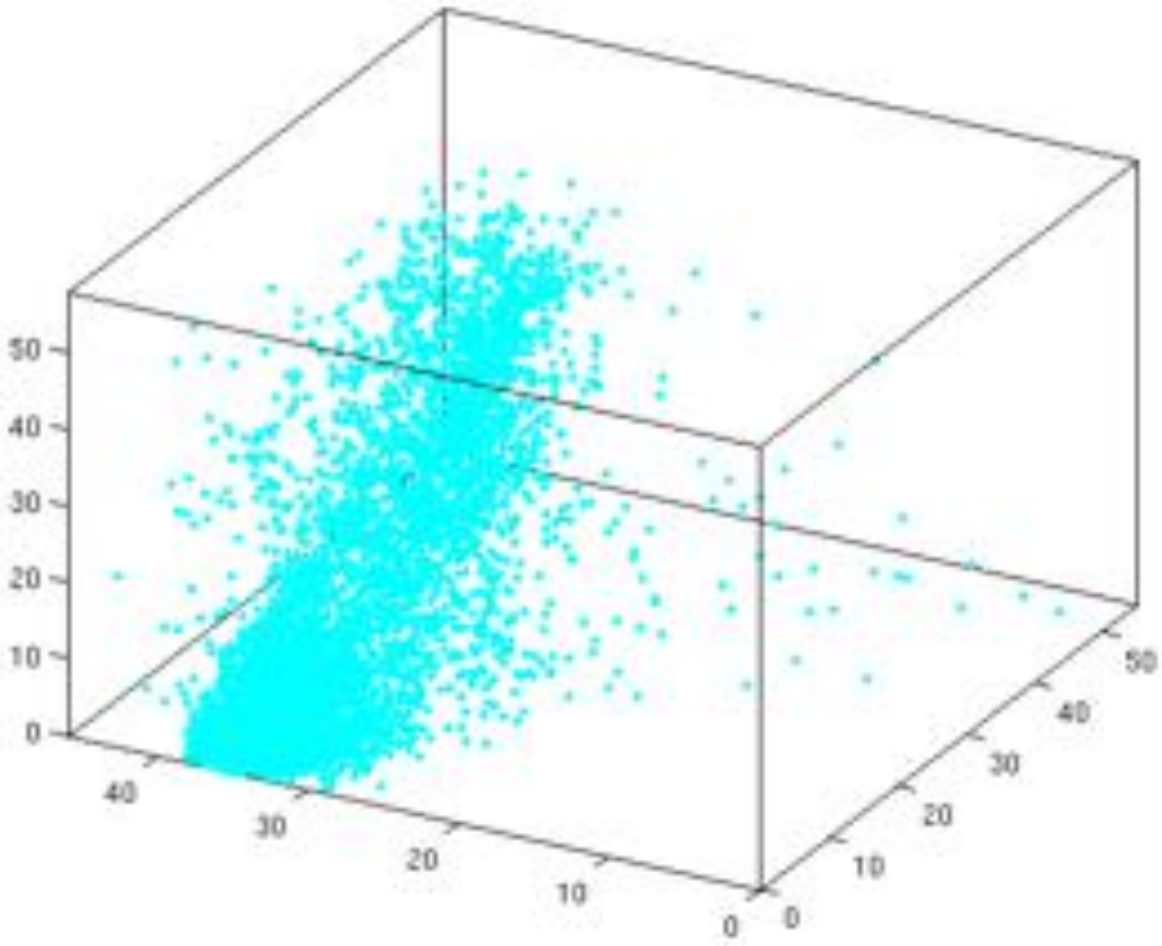


Metadata: each 3D point represents the position of one docked ligand in the protein pocket



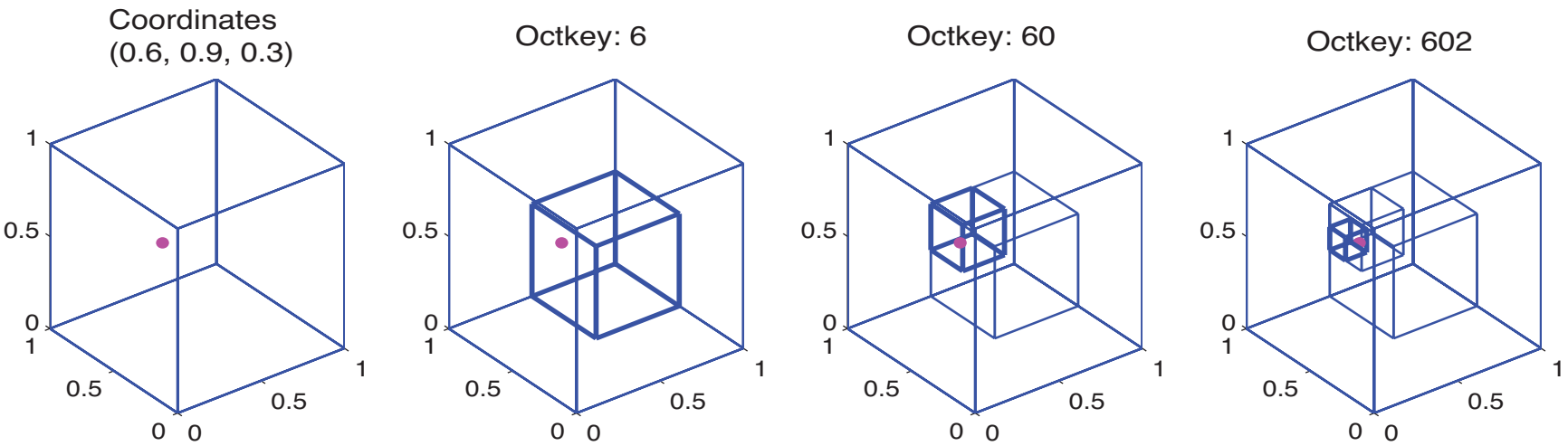
T. Estrada, B. Zhang, P. Cicotti, R. S. Armen, M. Taufer: **A scalable and accurate method for classifying protein-ligand binding geometries using a MapReduce approach.** Comp. in Bio. and Med. 42(7): 758-771 (2012)

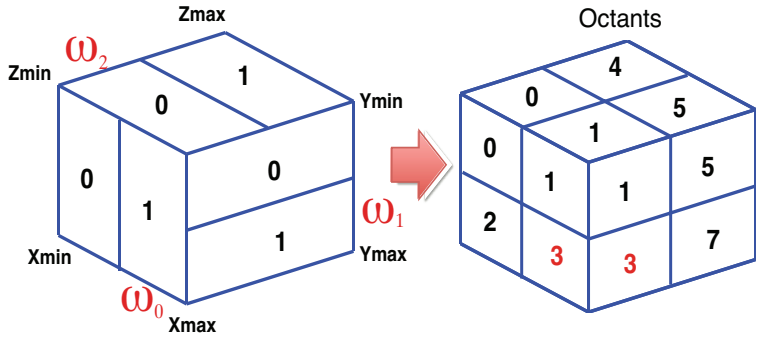
Search for Dense Spaces: Octree Clustering



T. Estrada, B. Zhang, P. Cicotti, R. S. Armen, M. Taufer: **A scalable and accurate method for classifying protein-ligand binding geometries using a MapReduce approach.** Comp. in Bio. and Med. 42(7): 758-771 (2012)

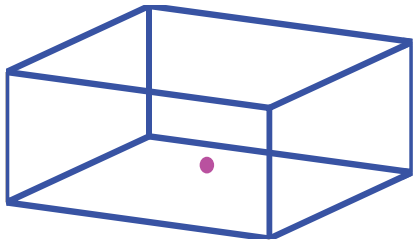
Generate keys

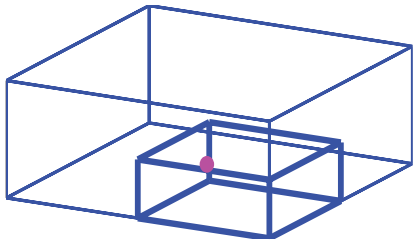




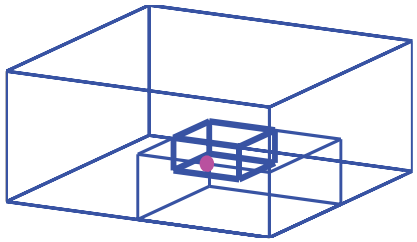
Octant id: $2^0\omega_0 + 2^1\omega_1 + 2^2\omega_2 = 1\ (1) + 2\ (1) + 2\ (0) = 3$

T. Estrada, B. Zhang, P. Cicotti, R. S. Armen, M. Taufer: **A scalable and accurate method for classifying protein-ligand binding geometries using a MapReduce approach.** Comp. in Bio. and Med. 42(7): 758-771 (2012)

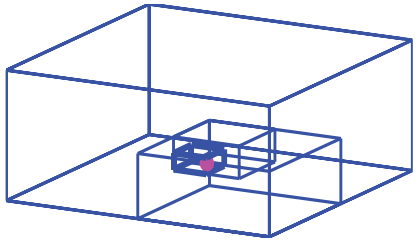




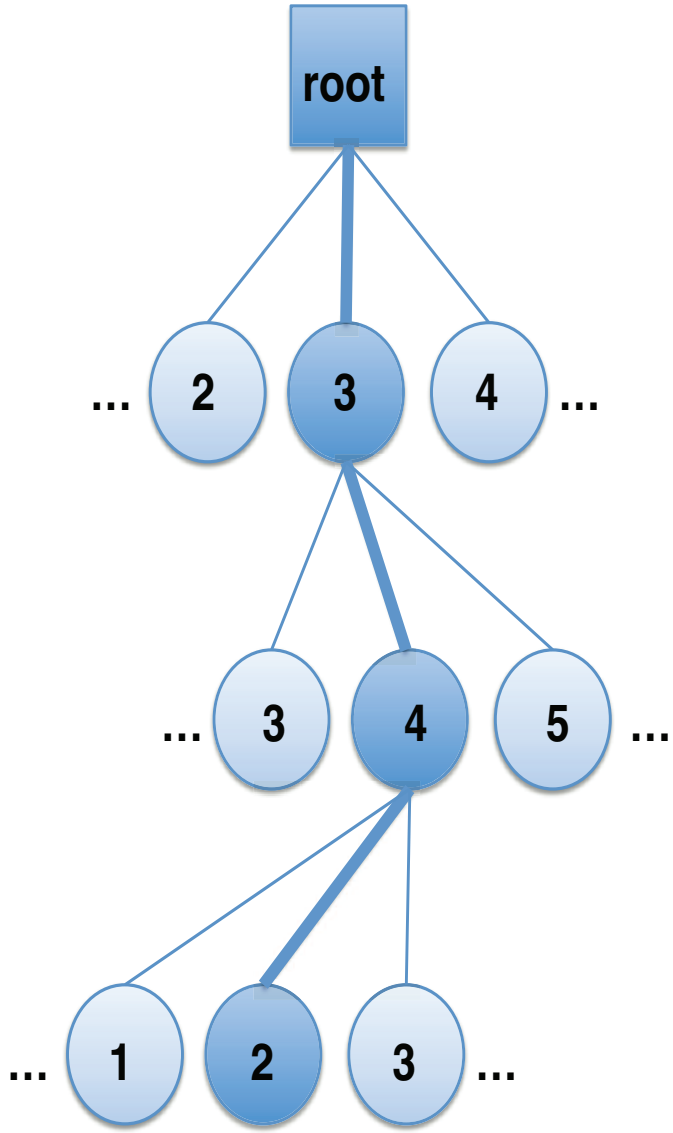
Octkey: 3



Octkey: 34

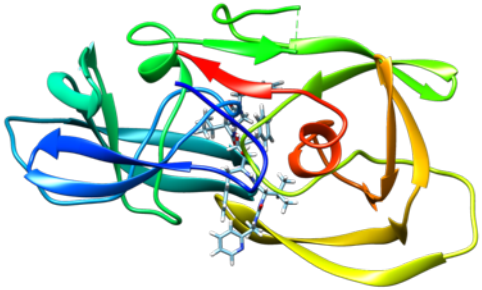


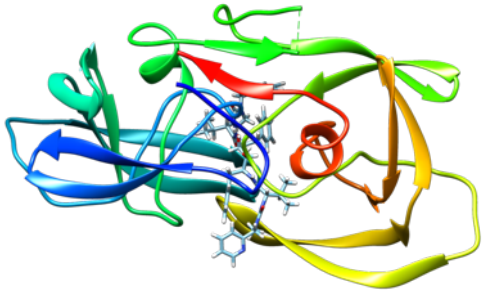
Octkey: 342

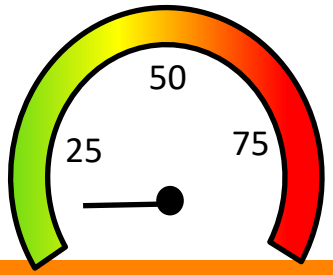


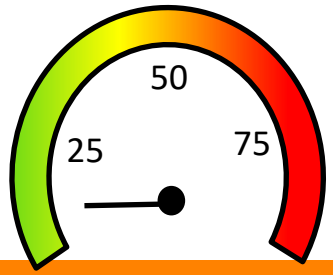
Nkey=3 (Depth of the tree)

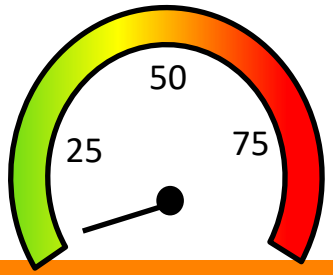
T. Estrada, B. Zhang, P. Cicotti, R. S. Armen, M. Taufer: **A scalable and accurate method for classifying protein-ligand binding geometries using a MapReduce approach.** Comp. in Bio. and Med. 42(7): 758-771 (2012)



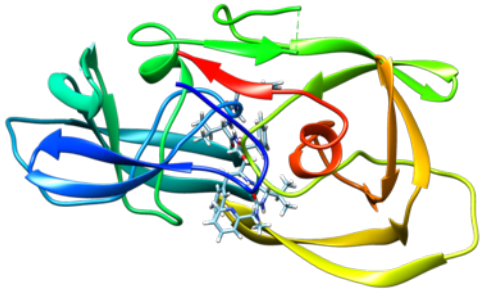


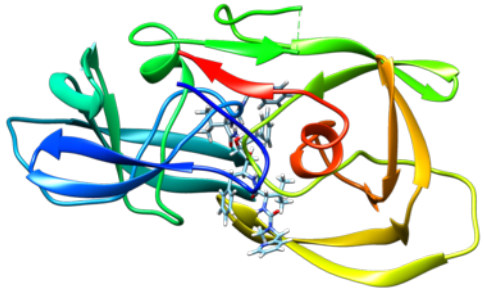


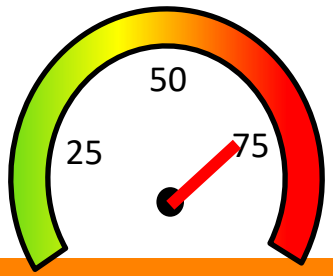


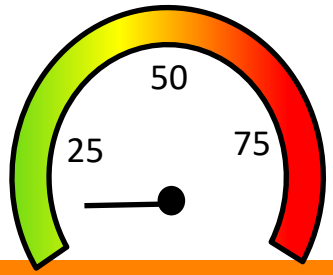


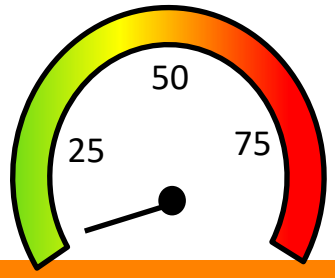
CPU utilization Memory utilization Network utilization



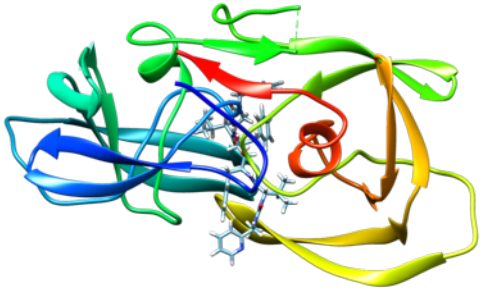


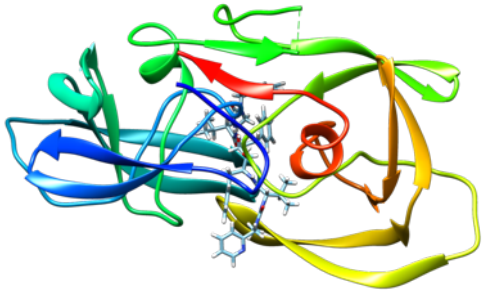


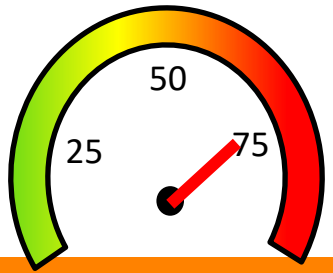


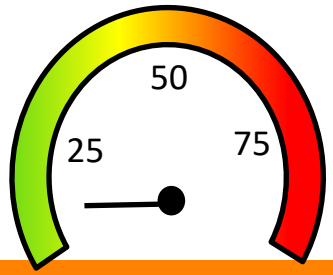


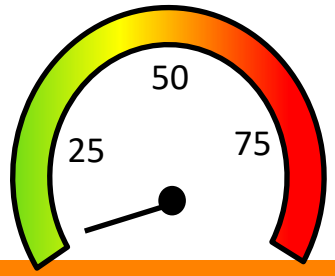
CPU utilization Memory utilization Network utilization







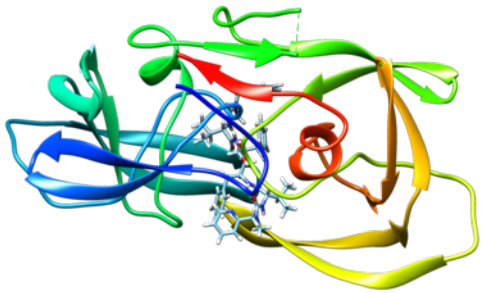




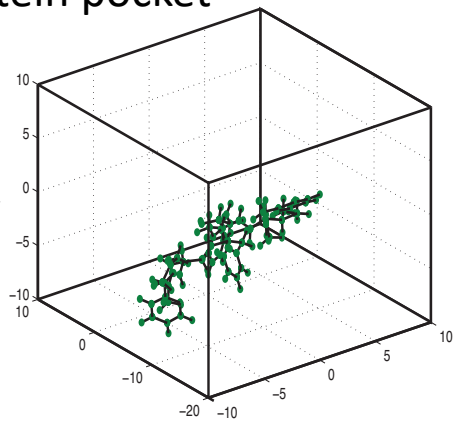
CPU utilization

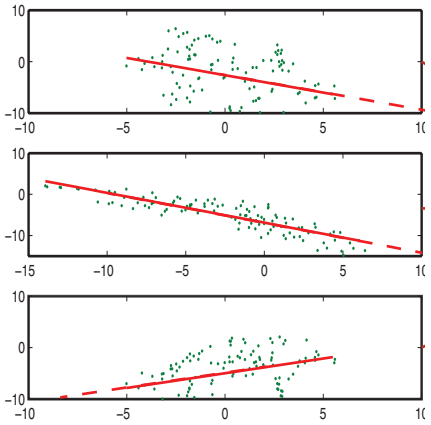
Memory utilization

Network utilization

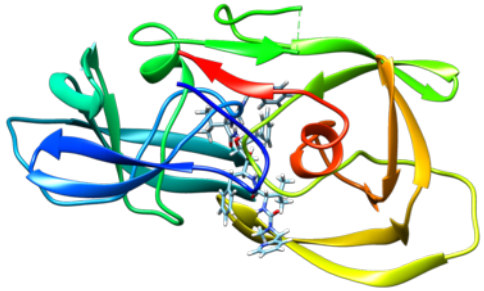


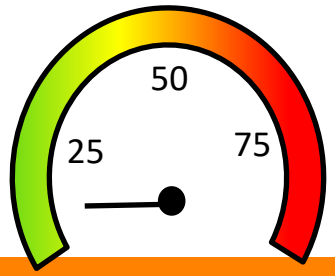
Protein pocket

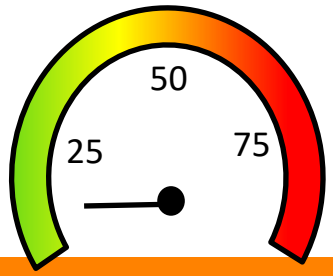


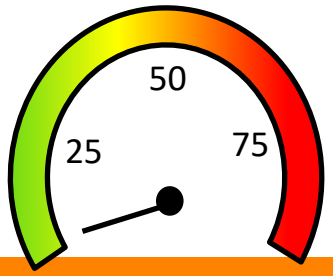


$$y = m_1 x + b_1$$





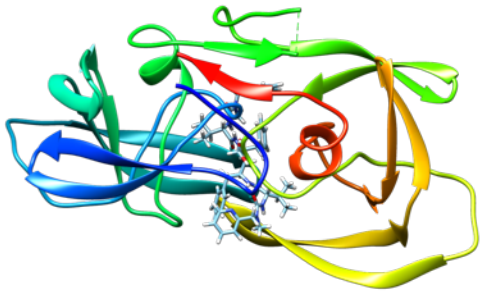




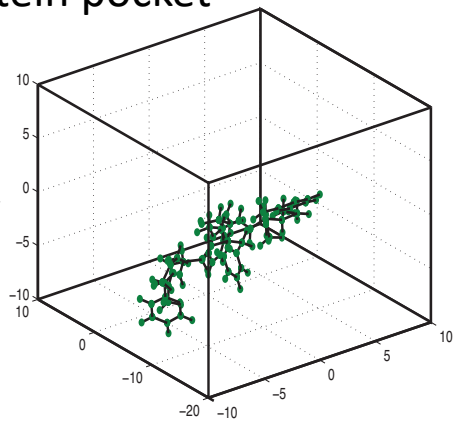
CPU utilization

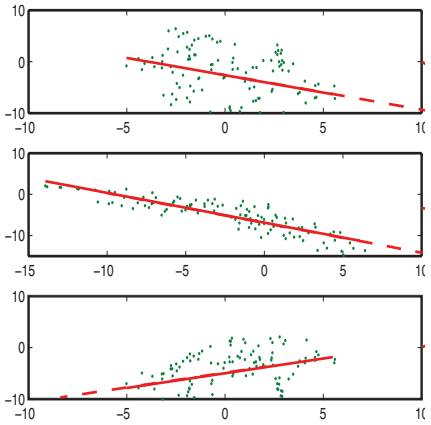
Memory utilization

Network utilization



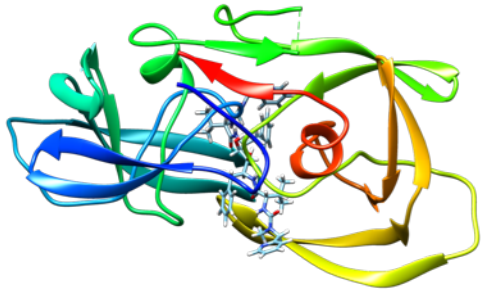
Protein pocket

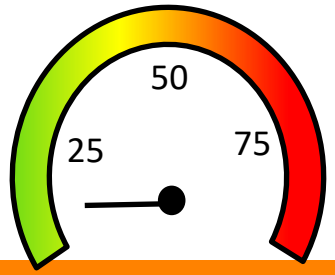


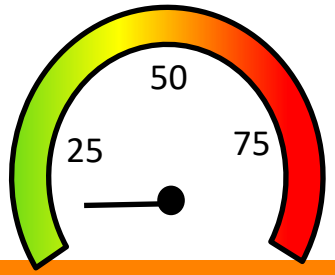


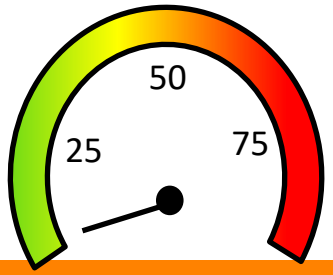
$$y = m_1 x + b_1$$

$$z = m_2 y + b_2$$





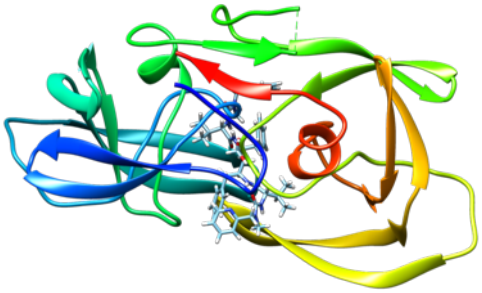




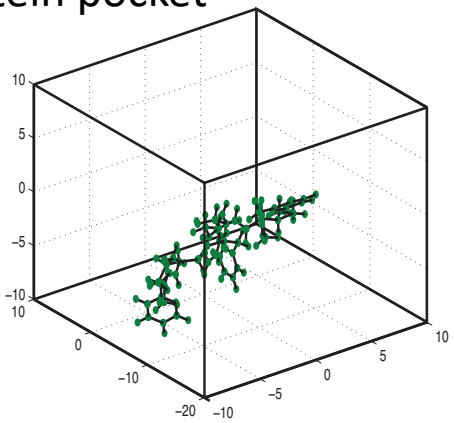
CPU utilization

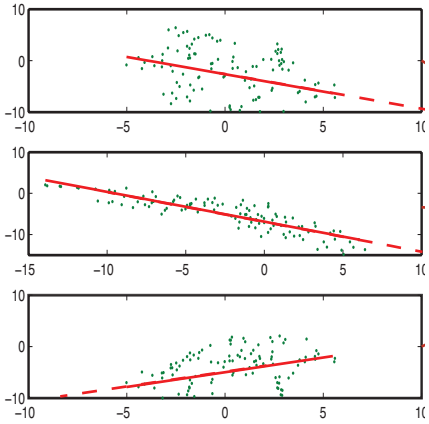
Memory utilization

Network utilization



Protein pocket

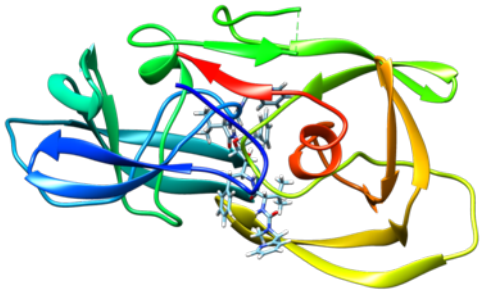


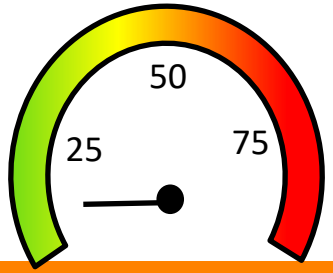


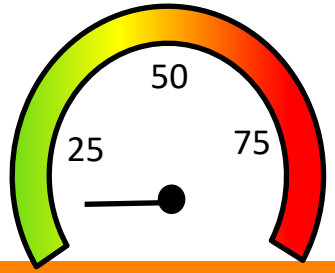
$$y = m_1 x + b_1$$

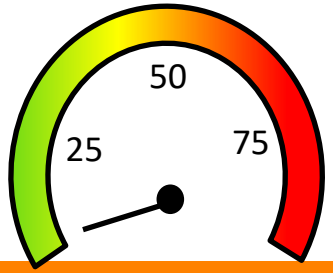
$$z = m_2 y + b_2$$

$$x = m_3 z + b_3$$





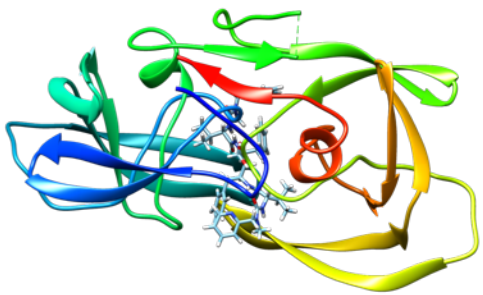


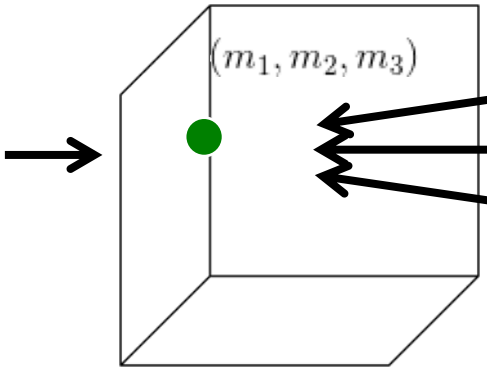


CPU utilization

Memory utilization

Network utilization

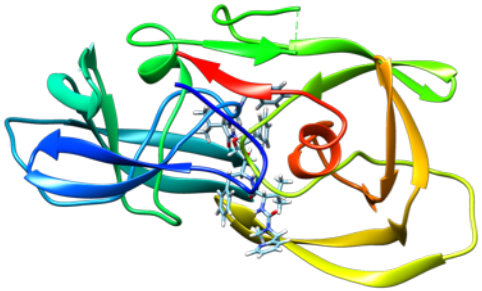


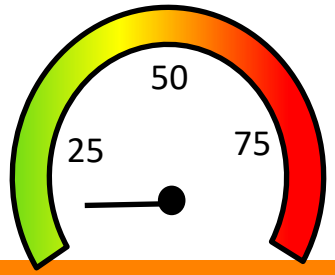


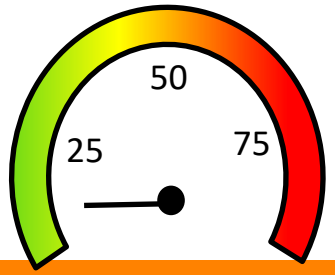
$$y = m_1 x + b_1$$

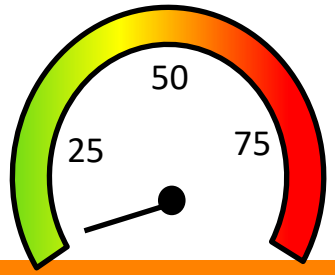
$$z = m_2 y + b_2$$

$$x = m_3 z + b_3$$

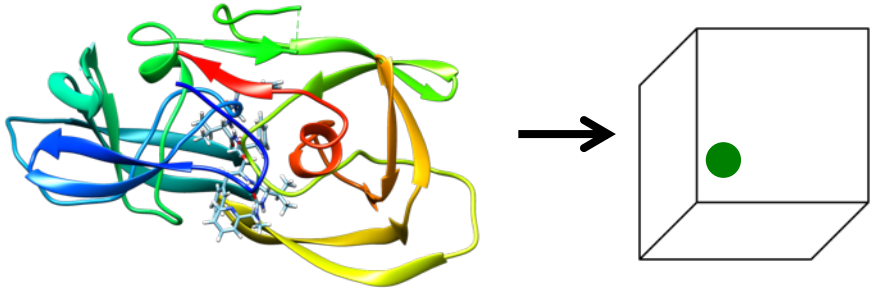


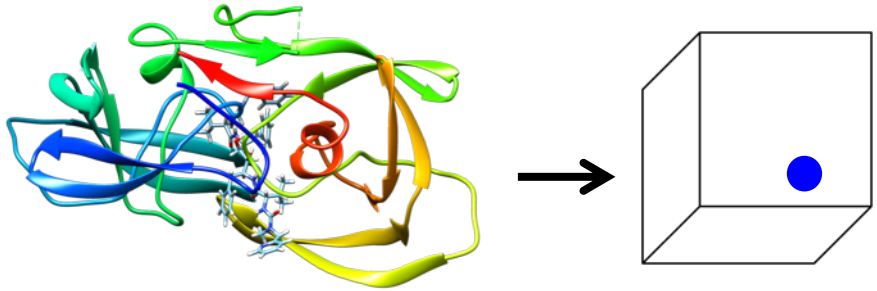


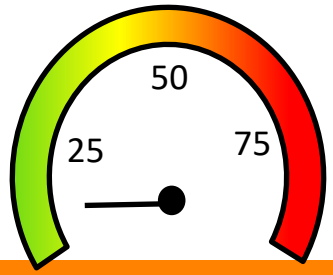


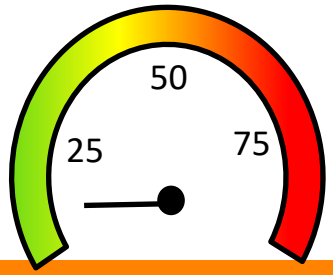


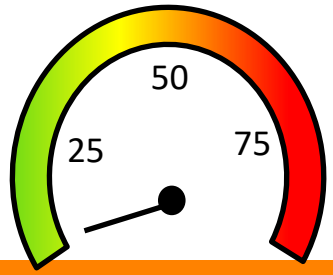
CPU utilization Memory utilization Network utilization



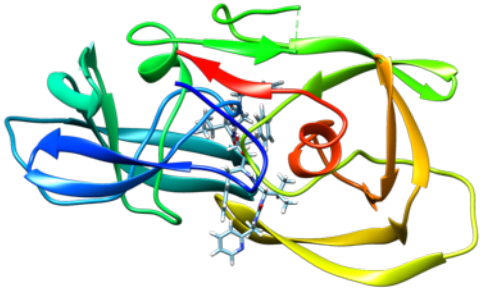


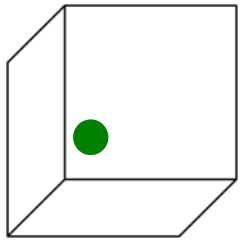


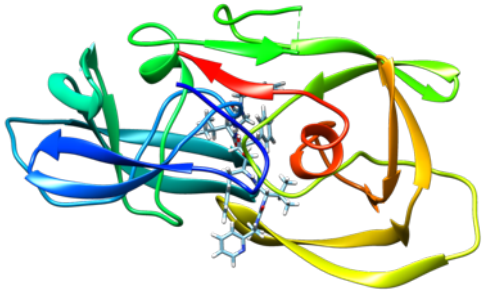


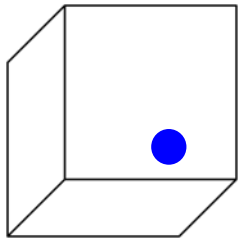


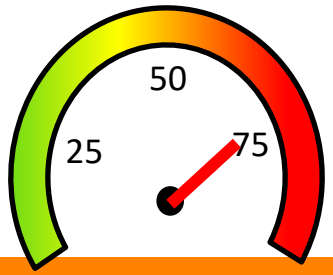
CPU utilization Memory utilization Network utilization

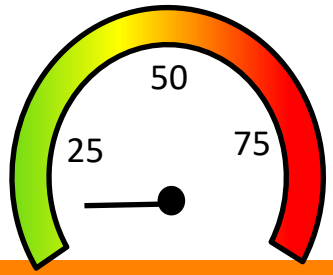


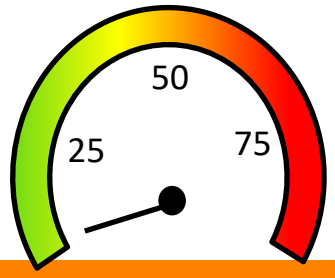








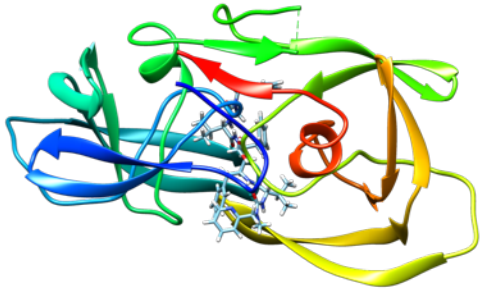


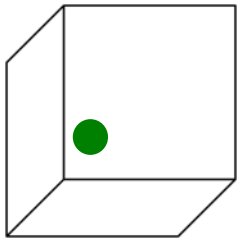


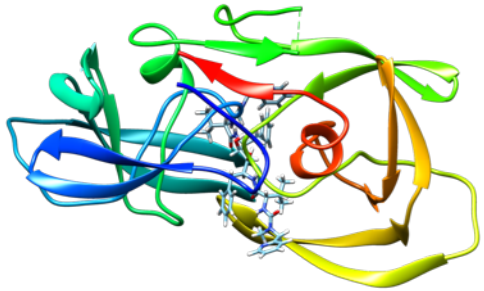
CPU utilization

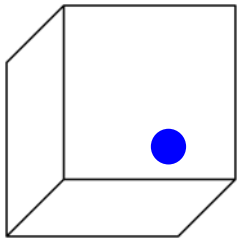
Memory utilization

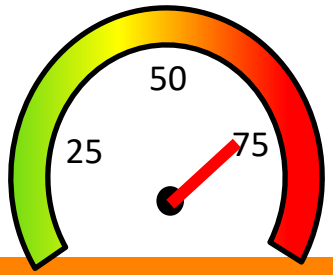
Network utilization

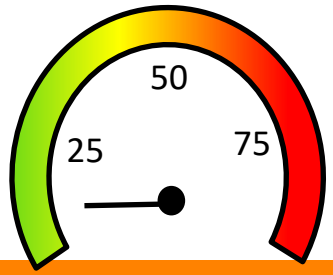


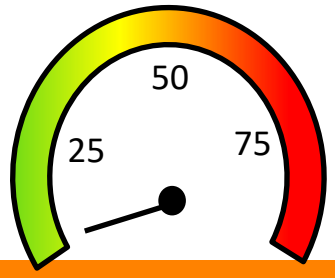




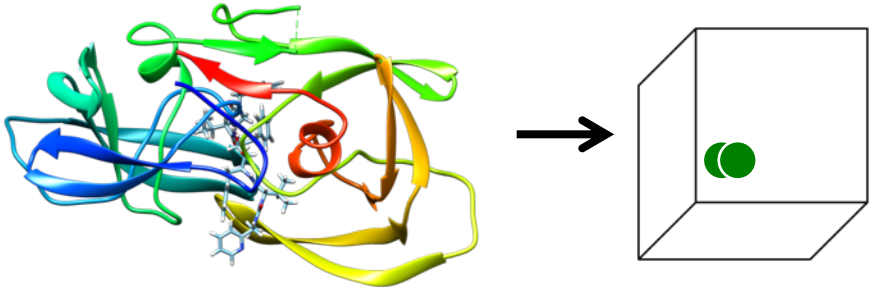


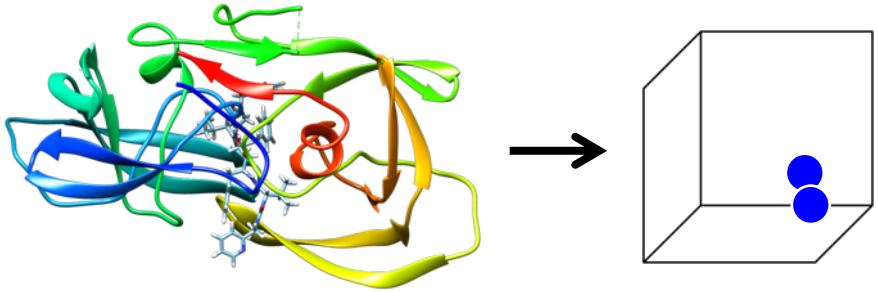


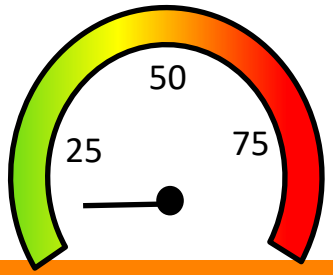


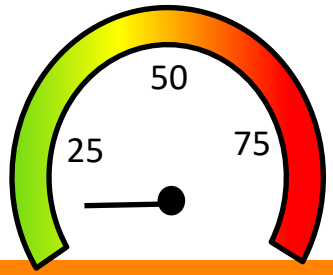


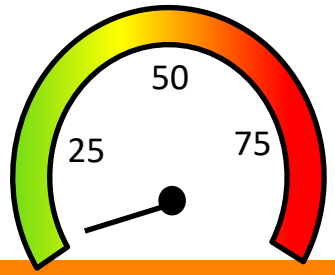
CPU utilization Memory utilization Network utilization



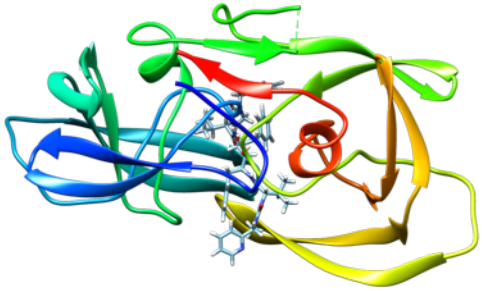


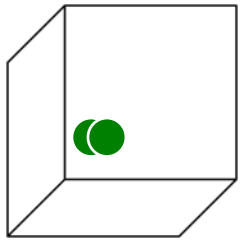


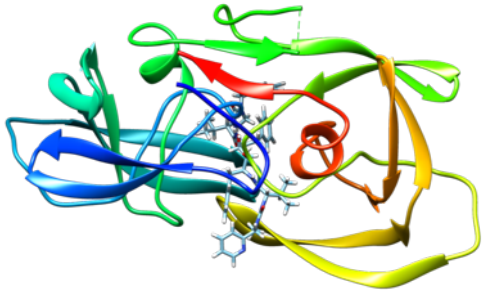


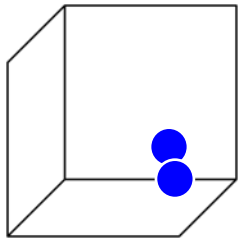


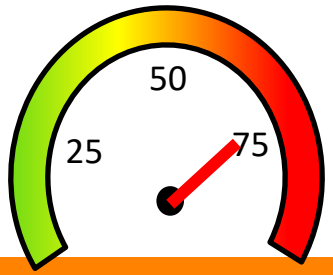
CPU utilization Memory utilization Network utilization

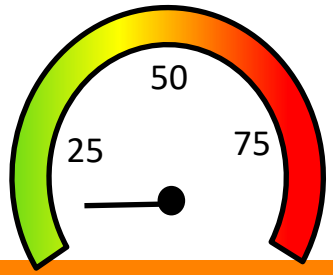


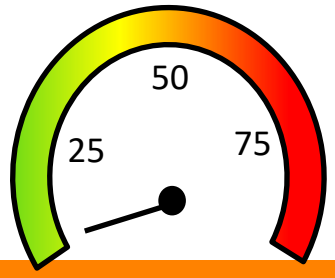








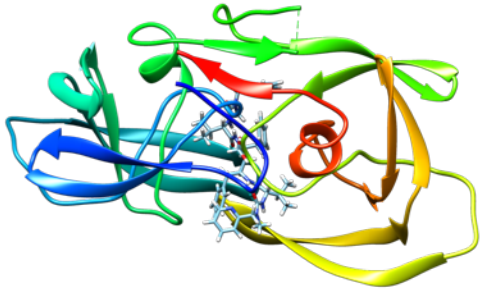


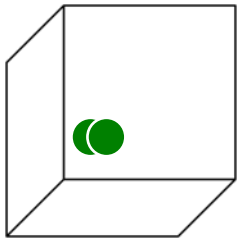


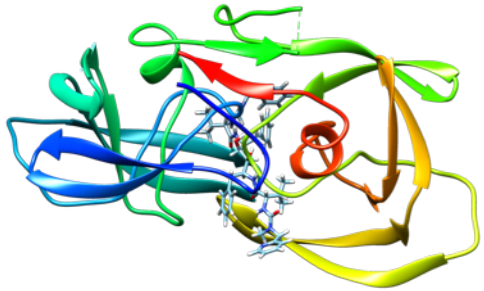
CPU utilization

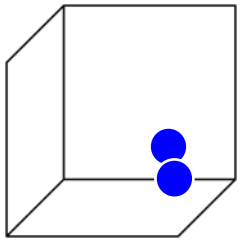
Memory utilization

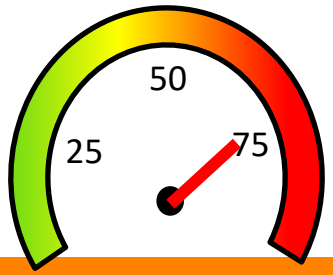
Network utilization

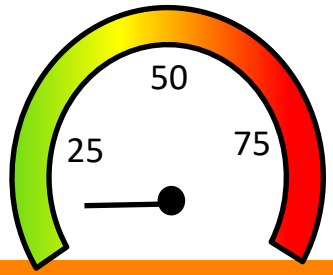


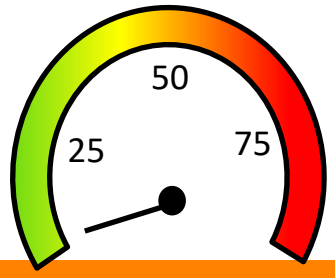








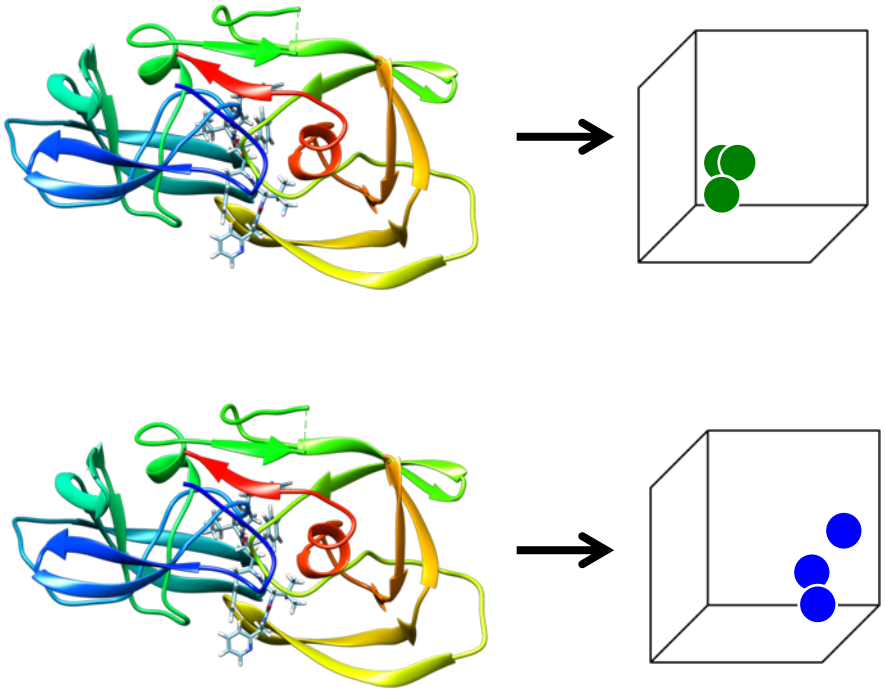




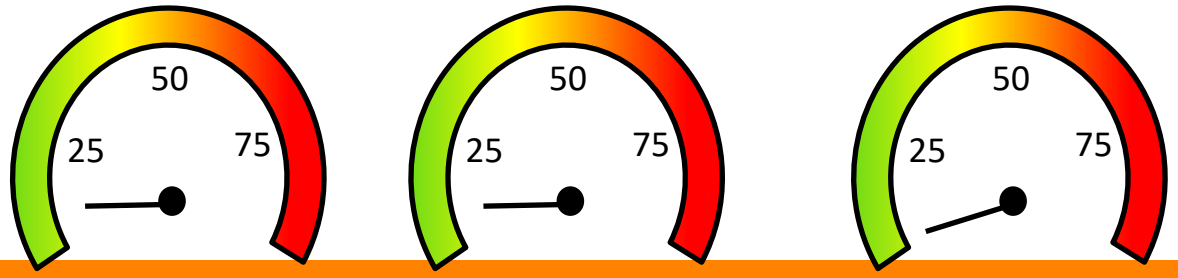
CPU utilization

Memory utilization

Network utilization

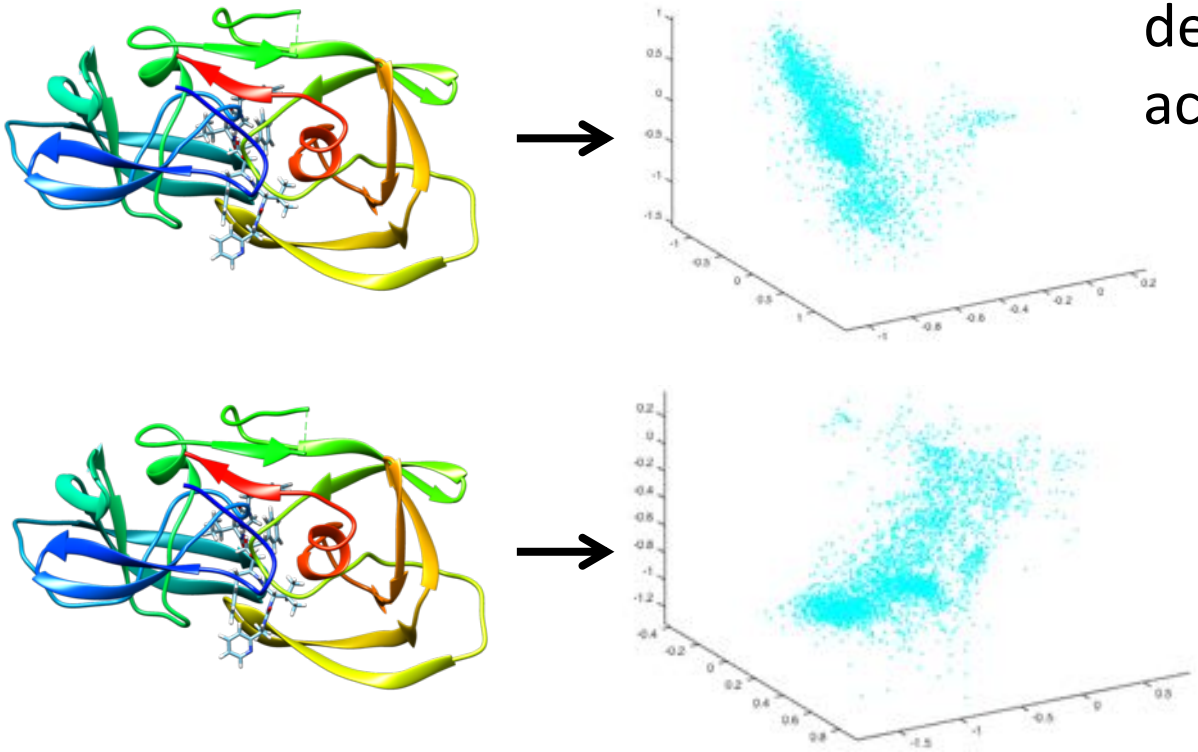


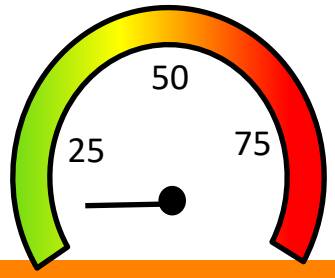
Ligands with similar geometries are mapped to close metadata points

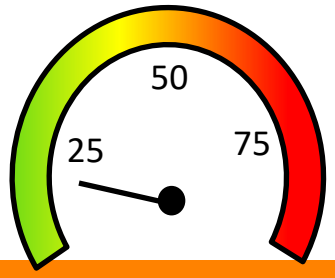


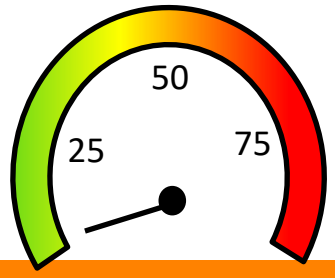
CPU utilization Memory utilization Network utilization

Search == identify the densest metadata region across all the nodes



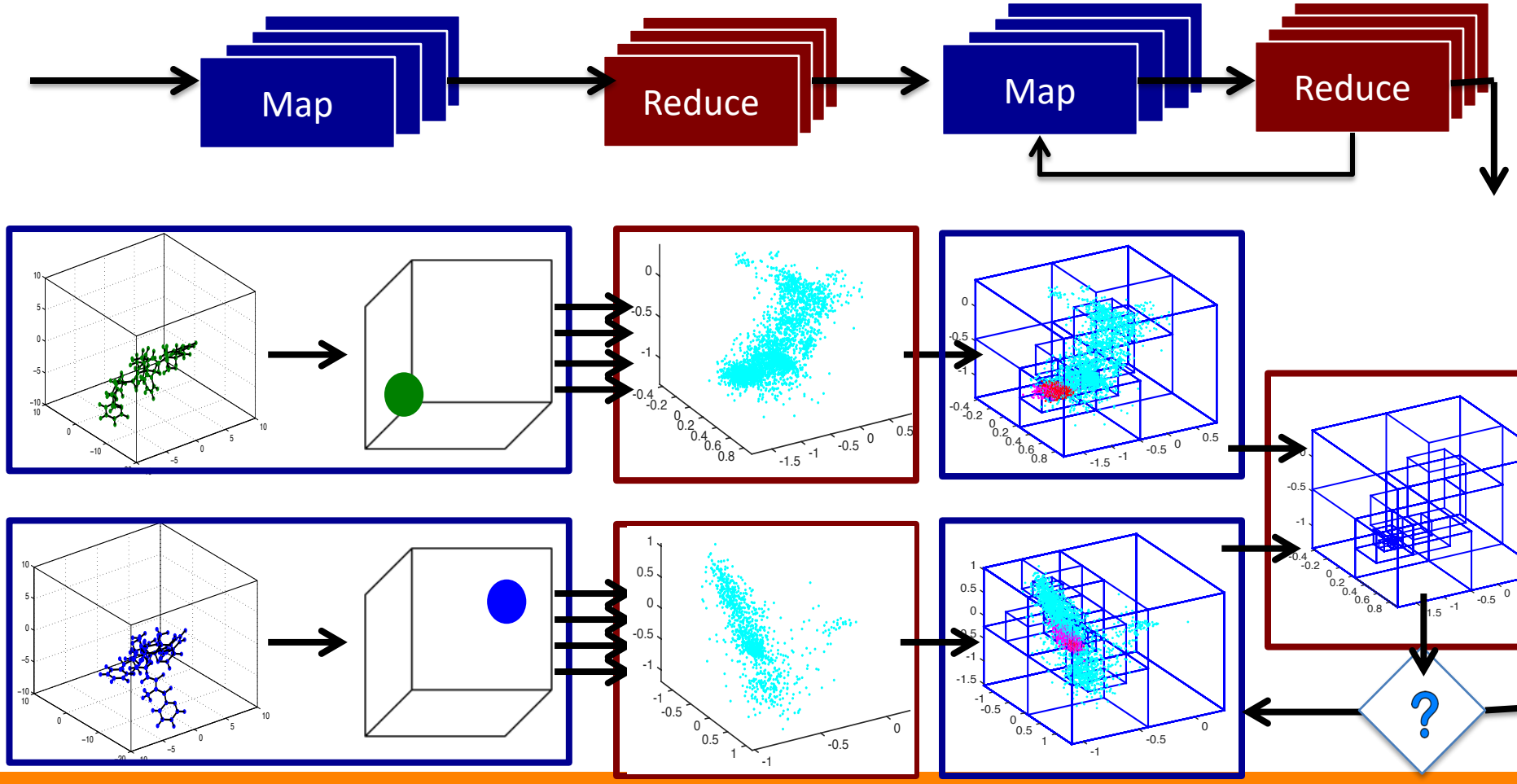




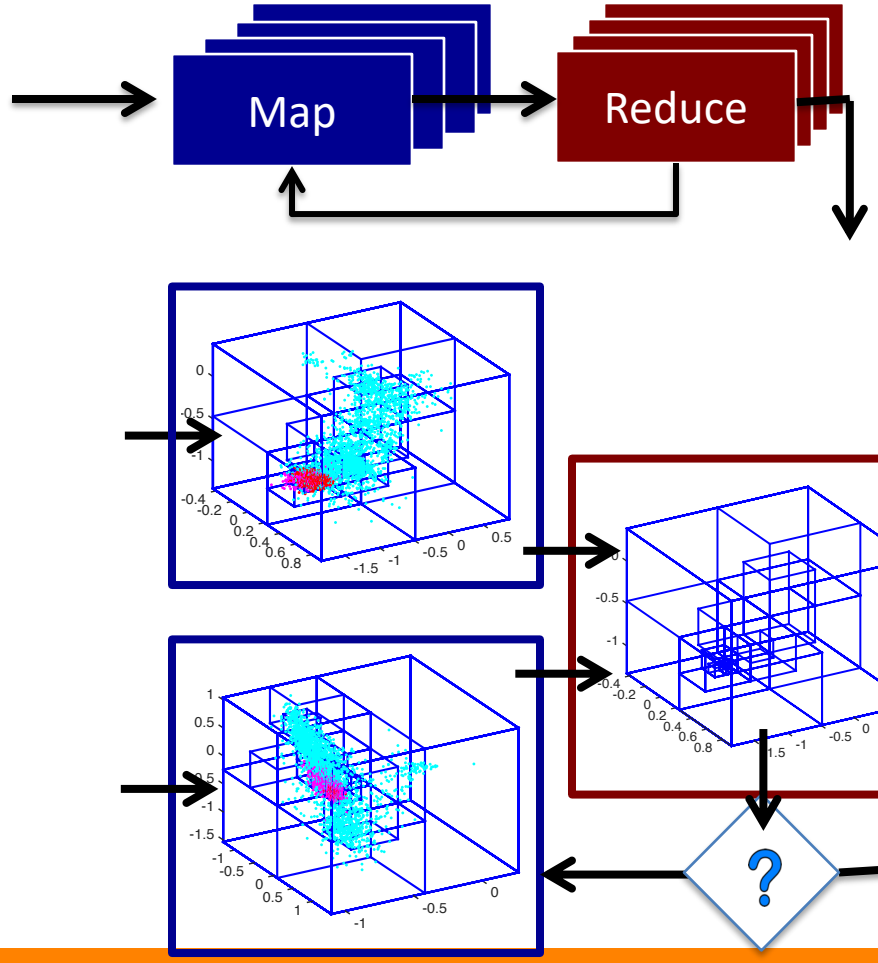


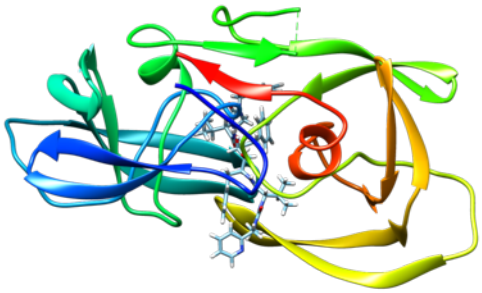
CPU utilization Memory utilization Network utilization

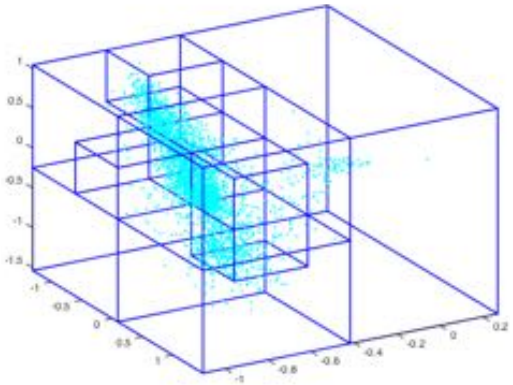
MapReduce-based Workflow

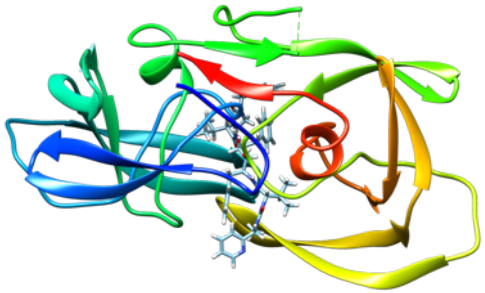


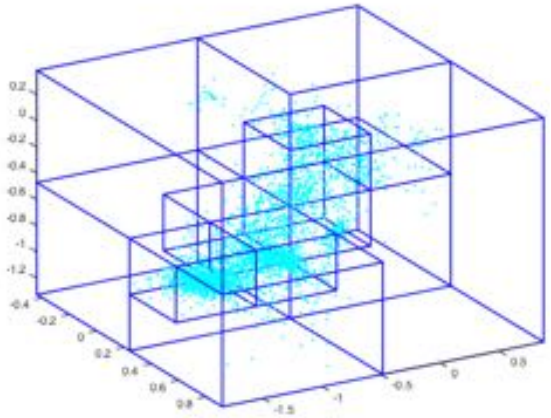
MapReduce-based Workflow



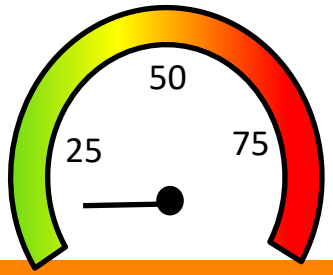




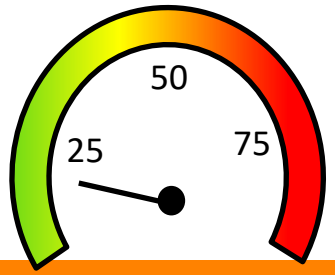




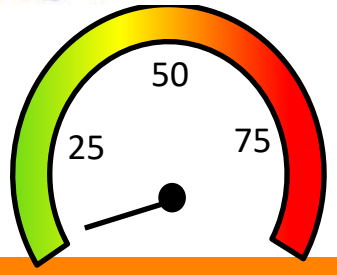
Build **local octrees** and identify the **global deepest densest octant**



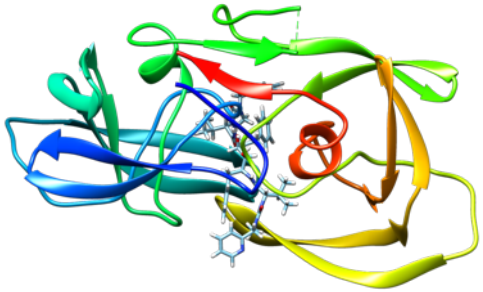
CPU utilization

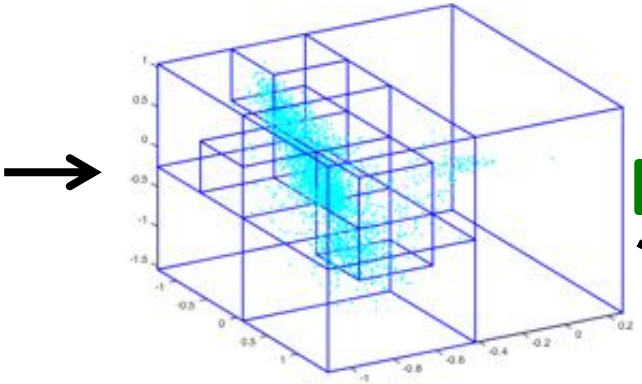


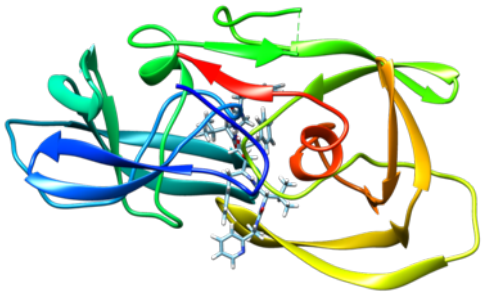
Memory utilization

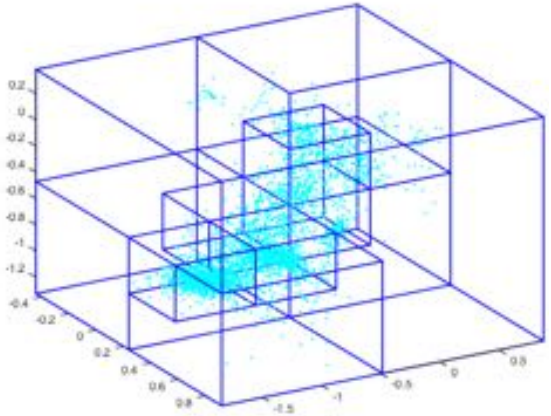


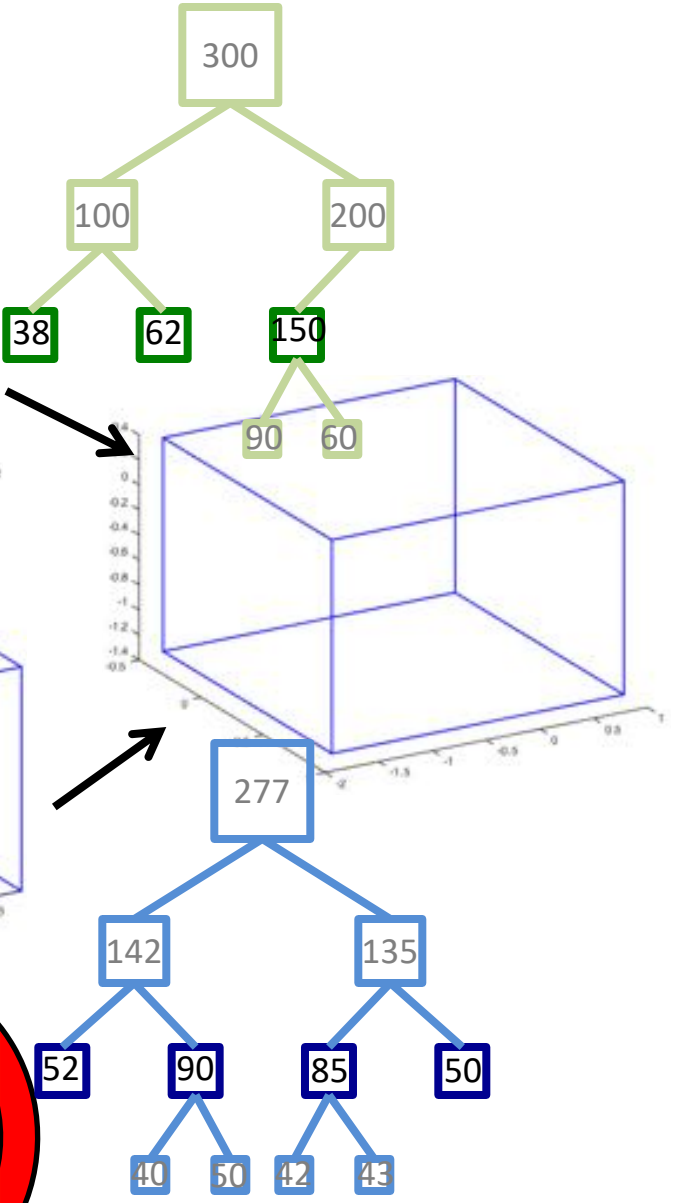
Network utilization

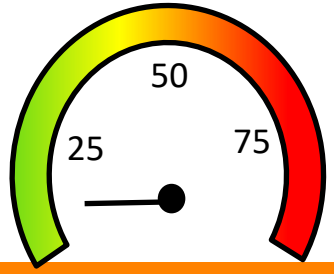


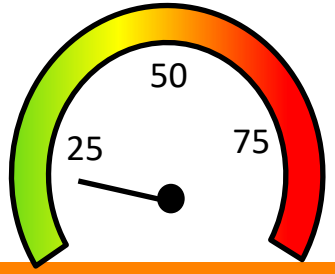


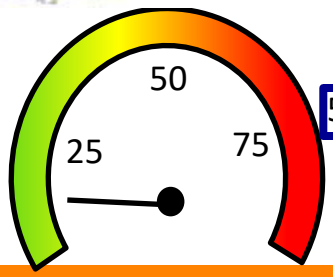




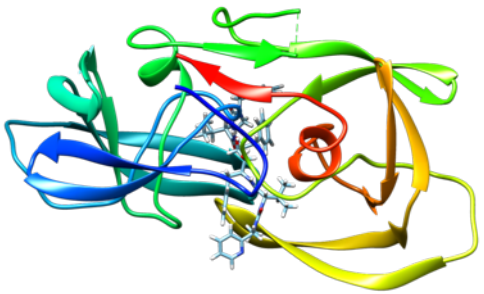


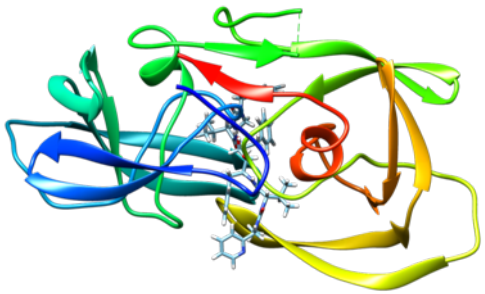


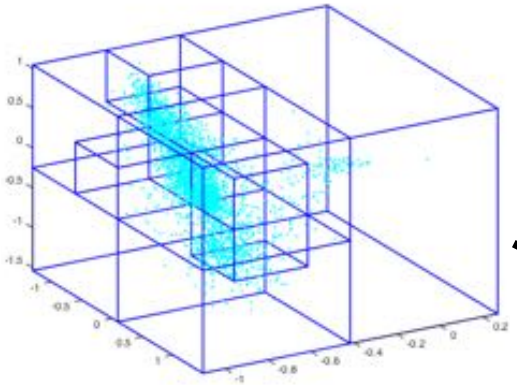


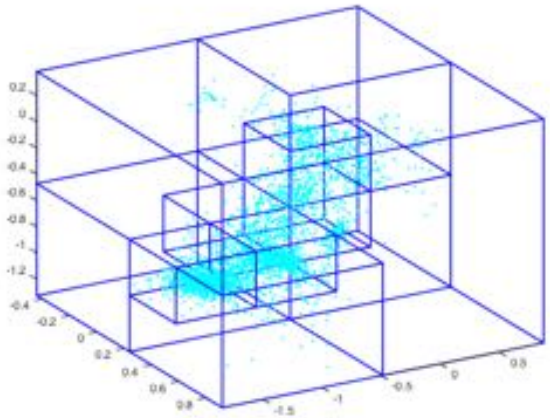


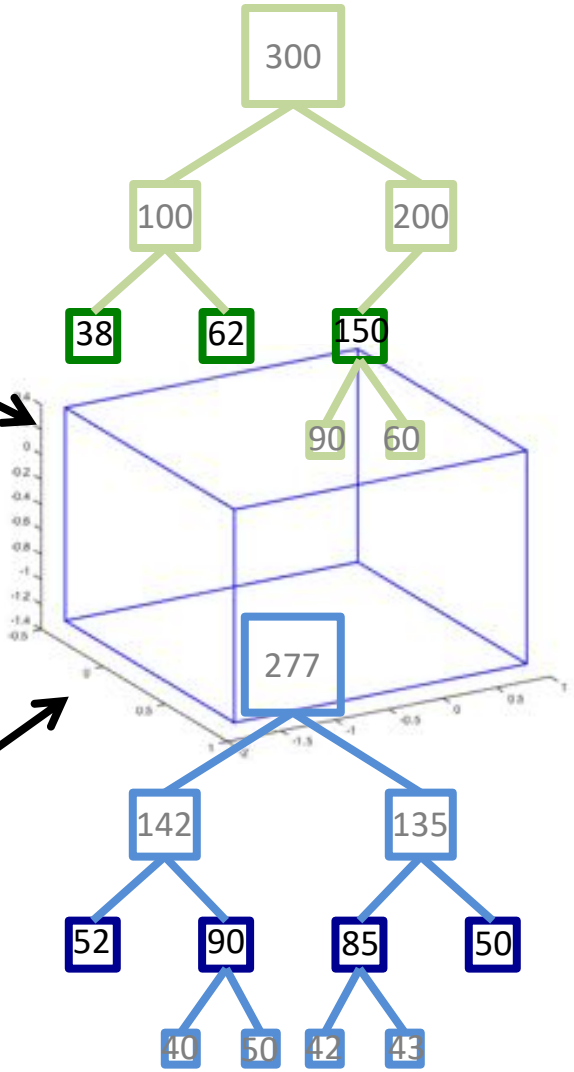
CPU utilization Memory utilization Network utilization

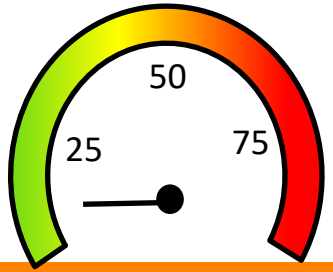


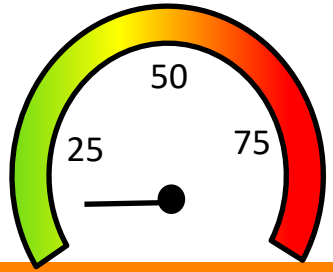


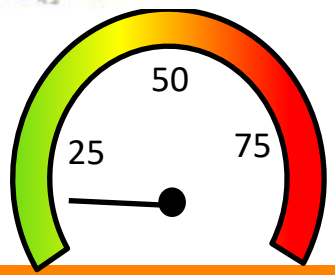




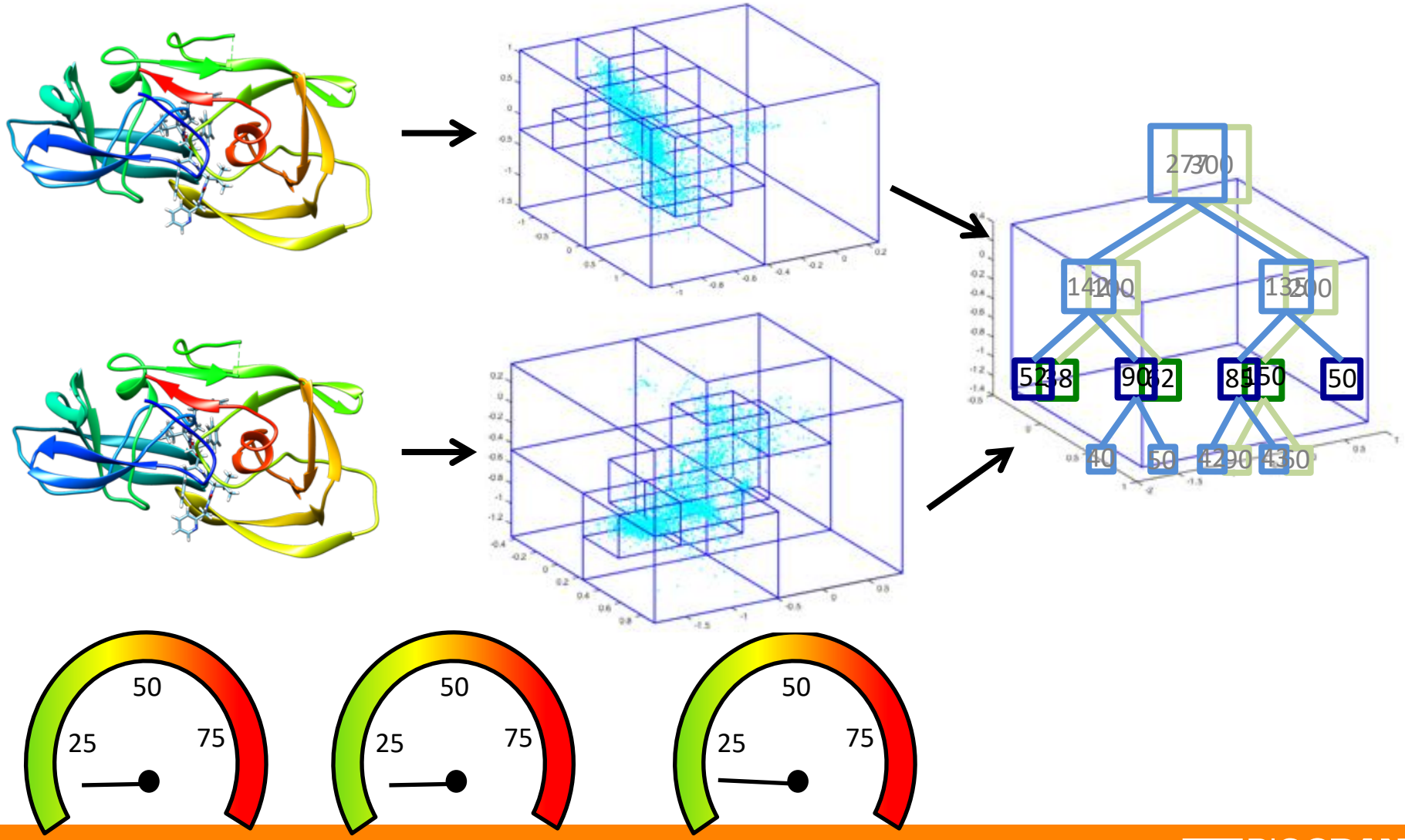




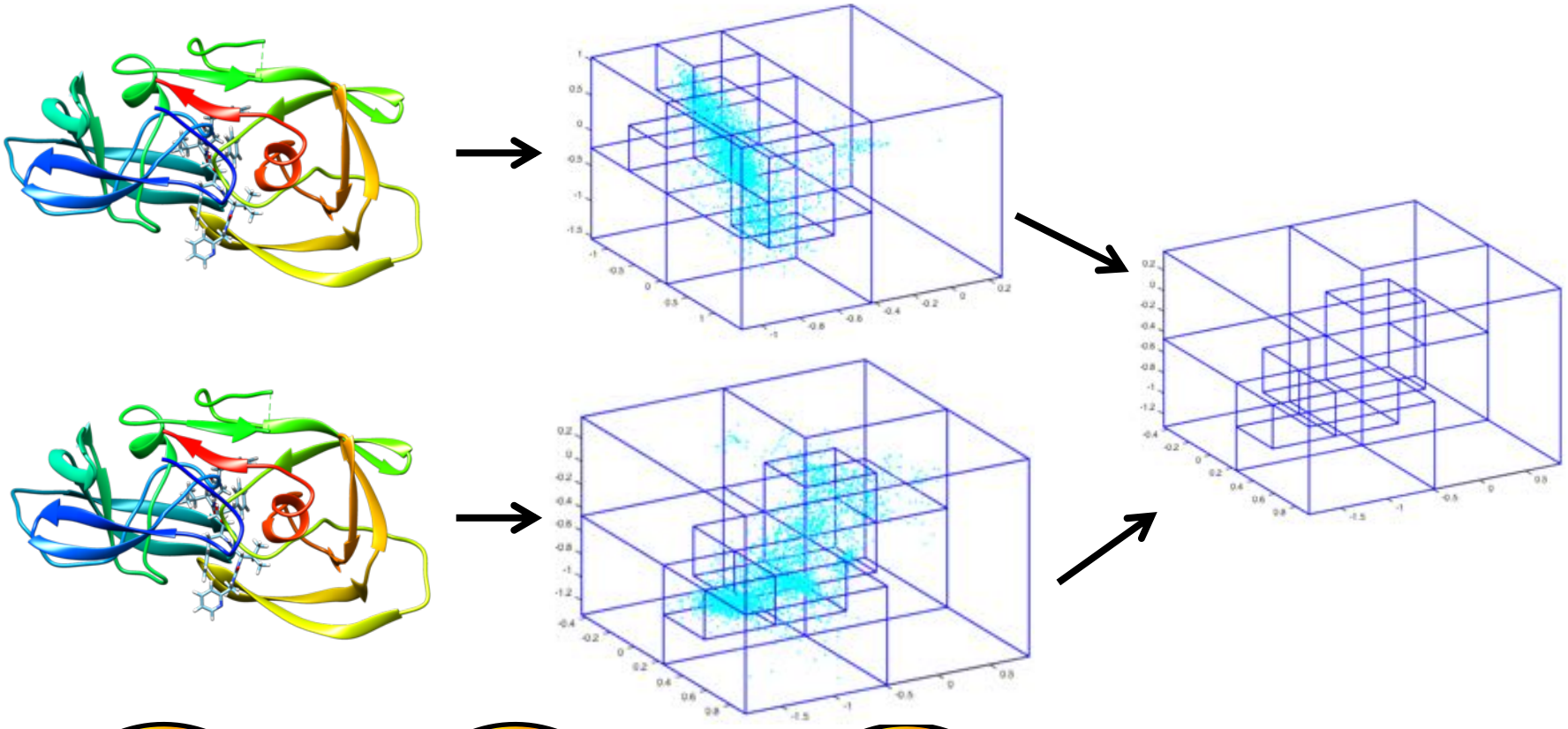


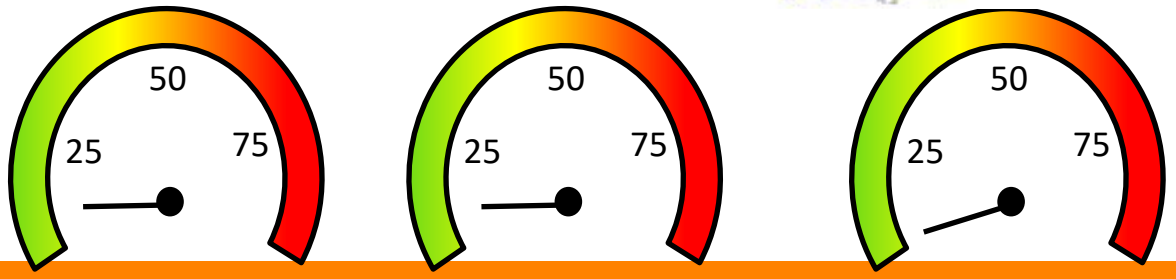


CPU utilization Memory utilization Network utilization

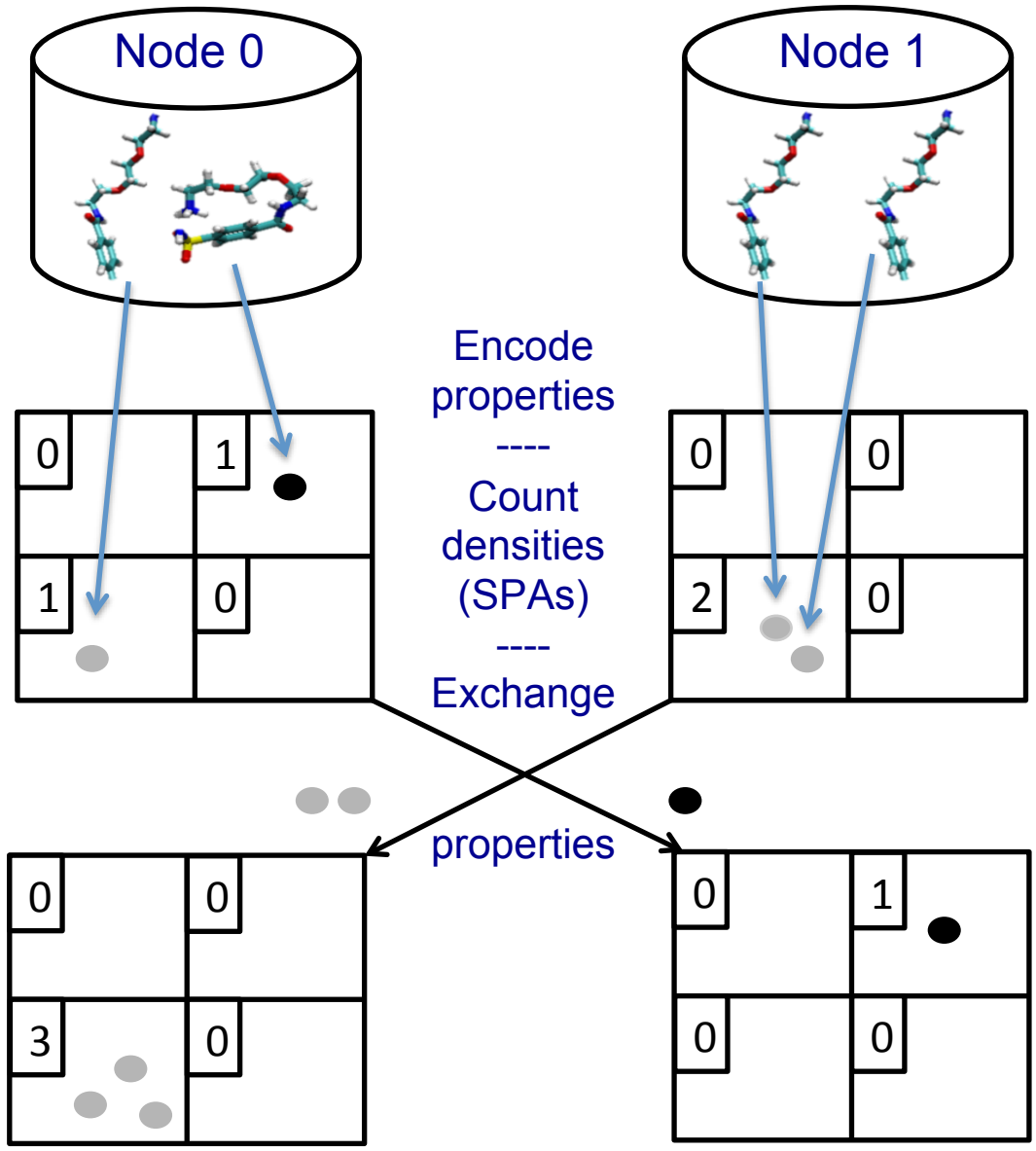


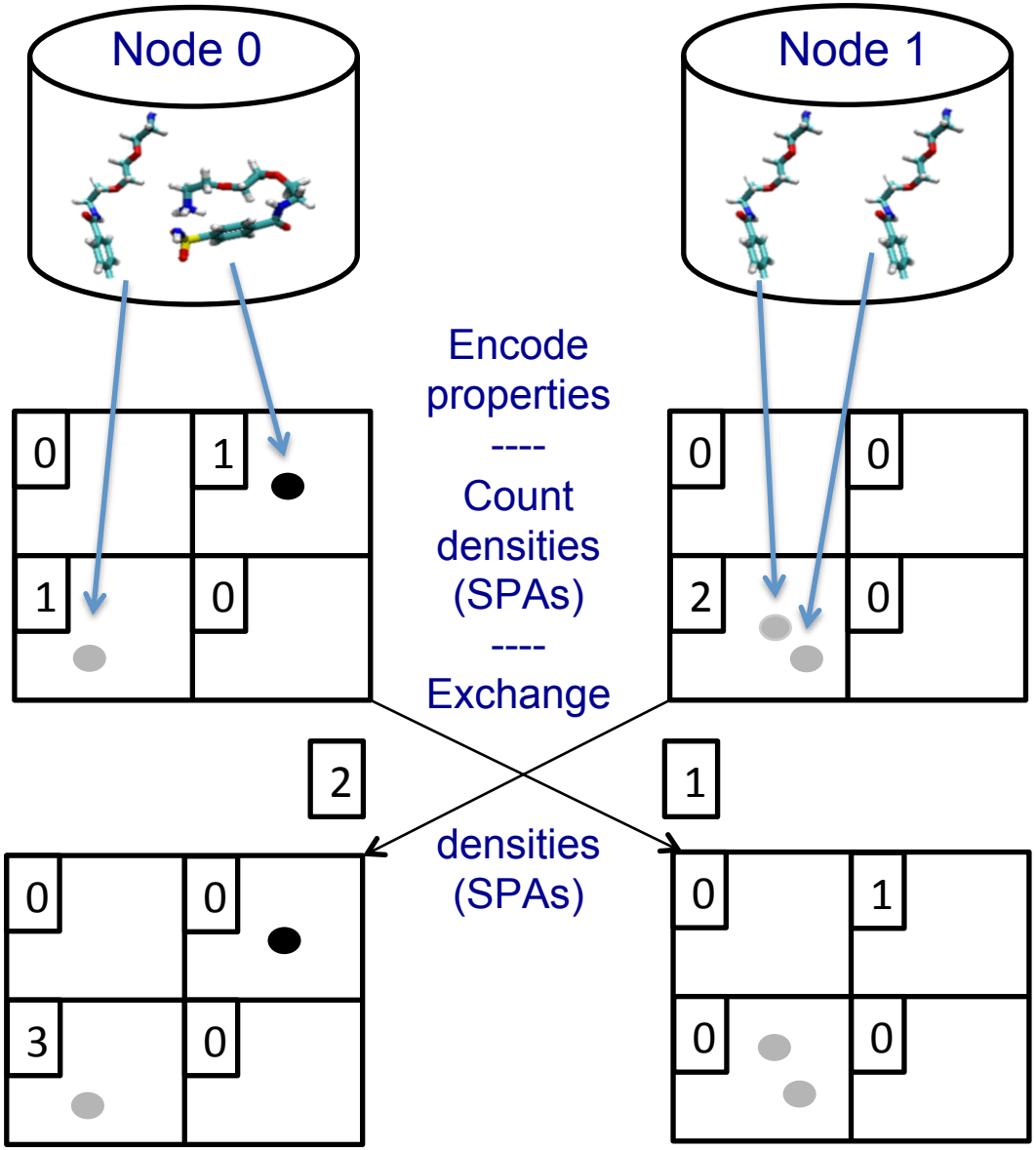
CPU utilization Memory utilization Network utilization

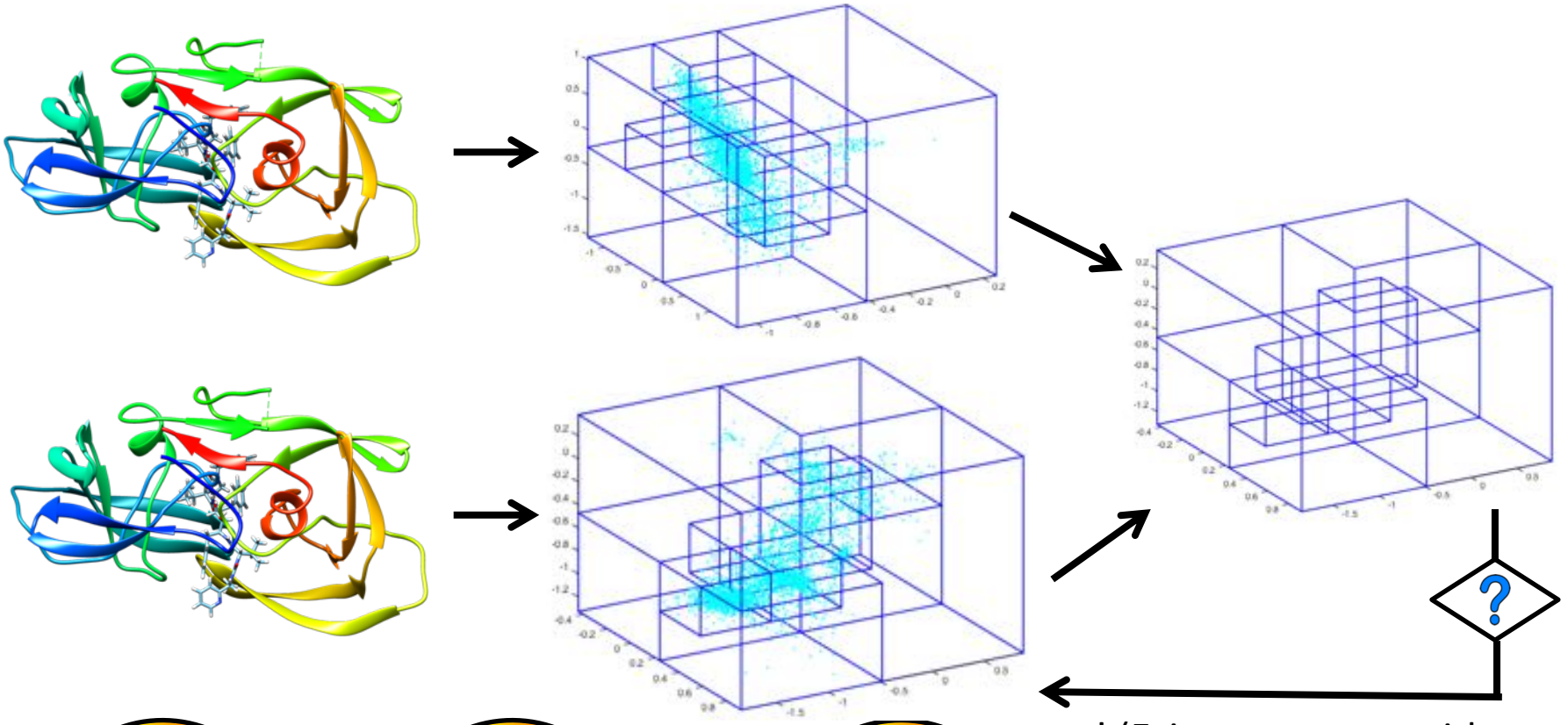


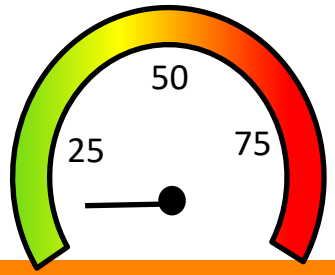


CPU utilization Memory utilization Network utilization

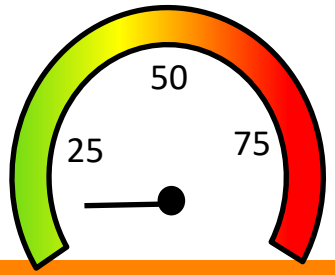




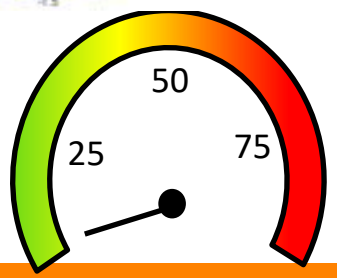




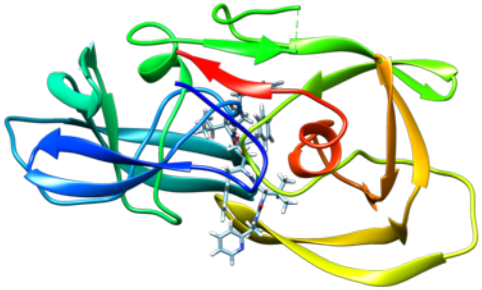
CPU utilization

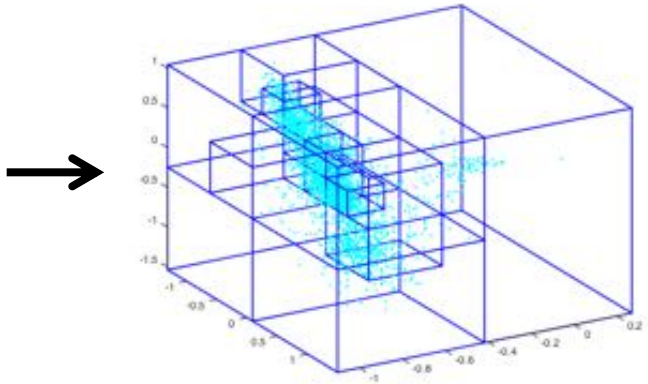


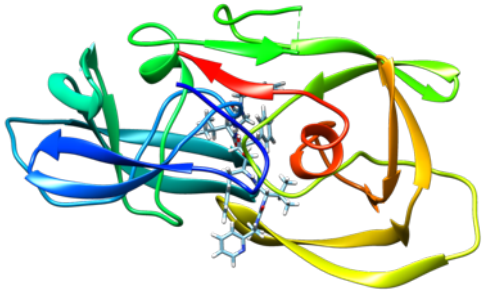
Memory utilization

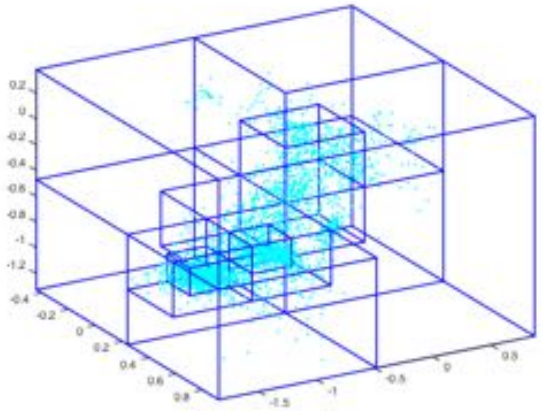


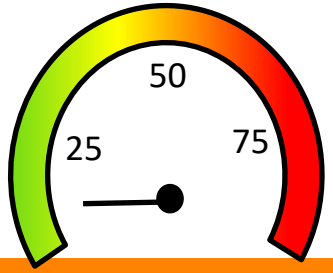
Network utilization

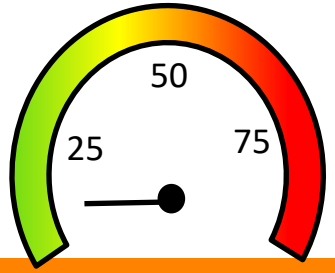


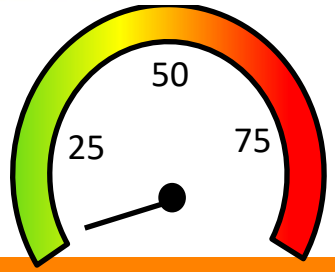








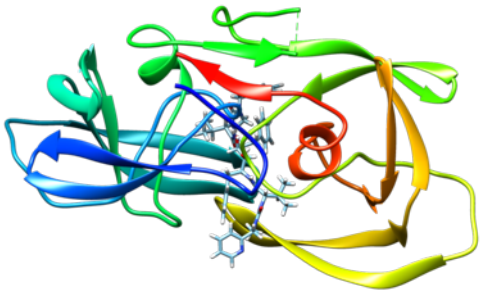


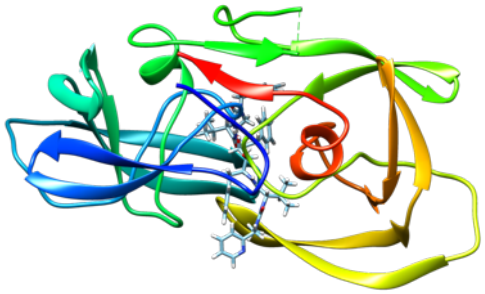


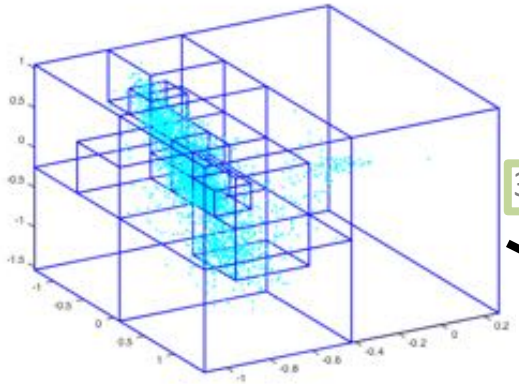
CPU utilization

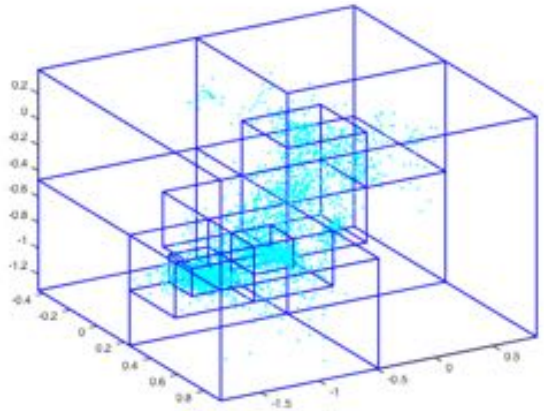
Memory utilization

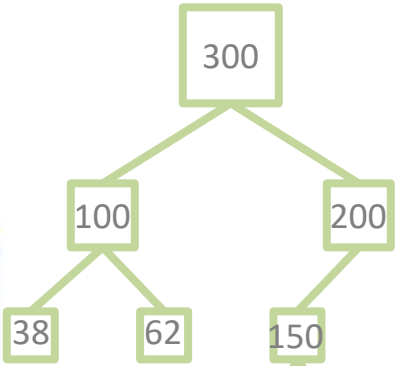
Network utilization

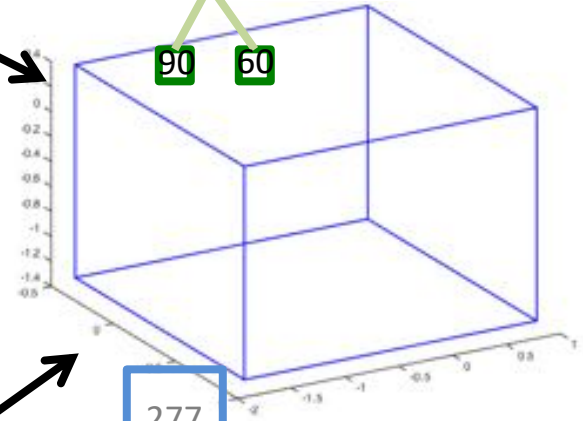


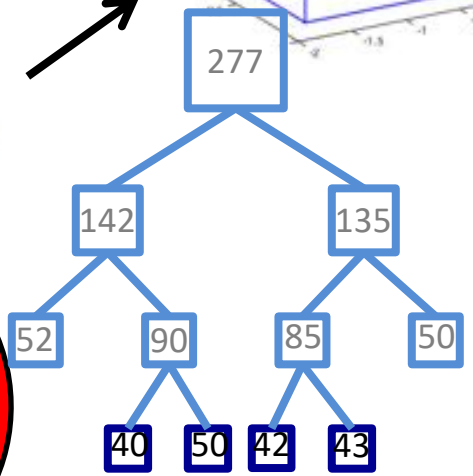


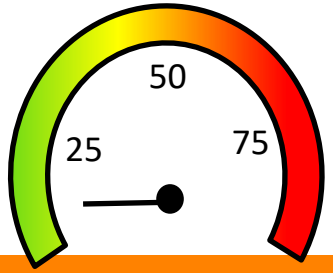


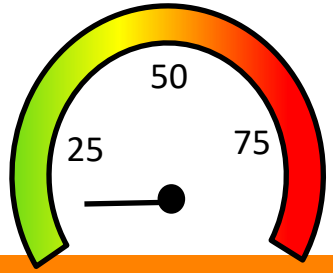


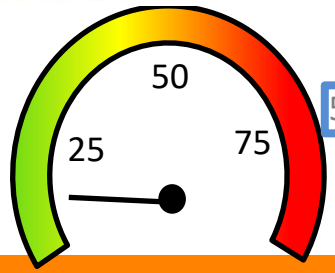








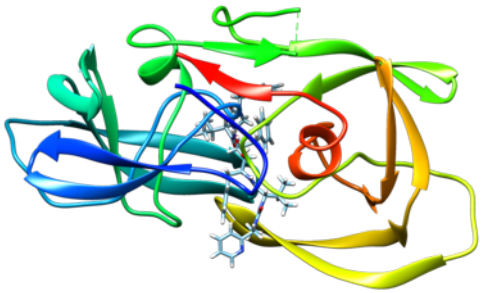


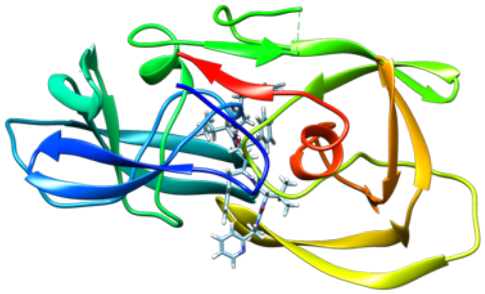


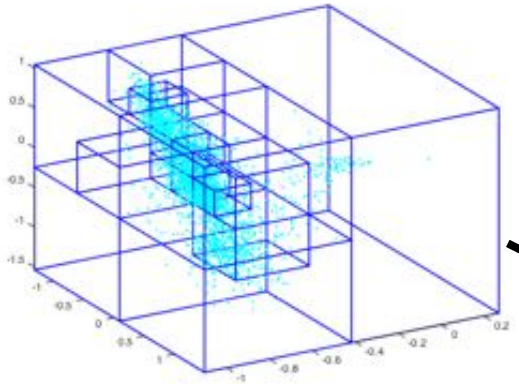
CPU utilization

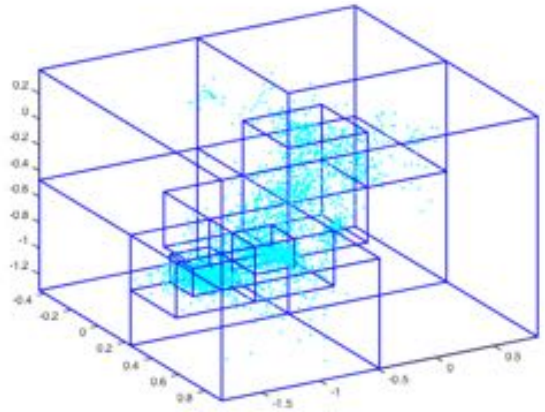
Memory utilization

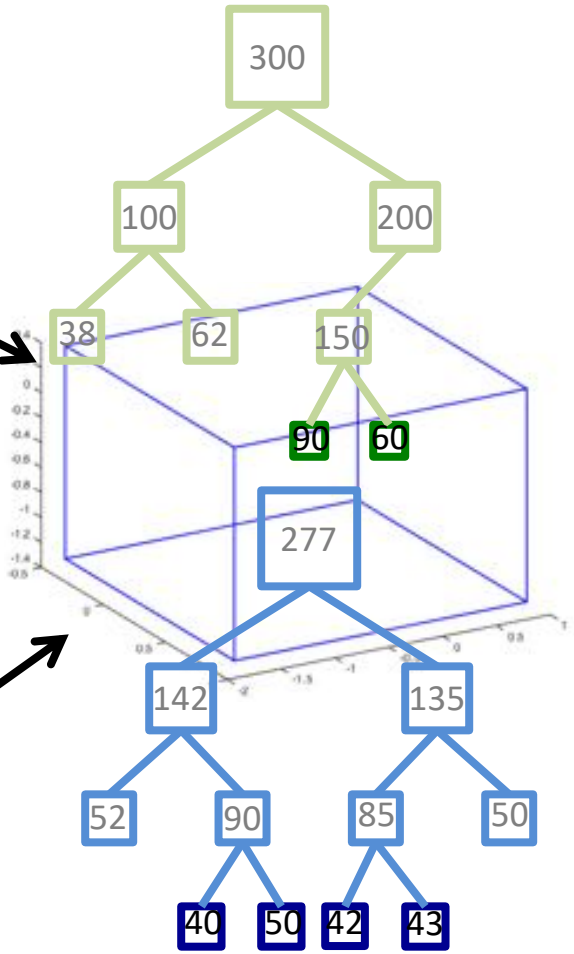
Network utilization

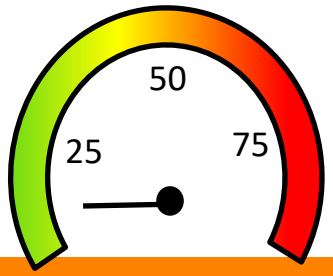




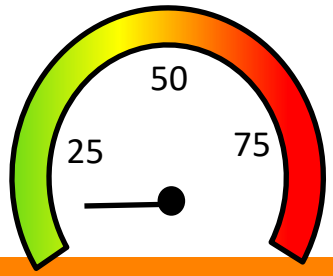




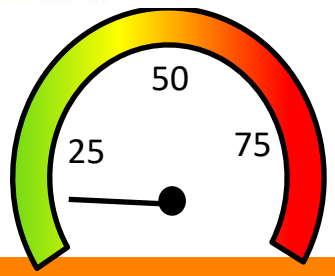




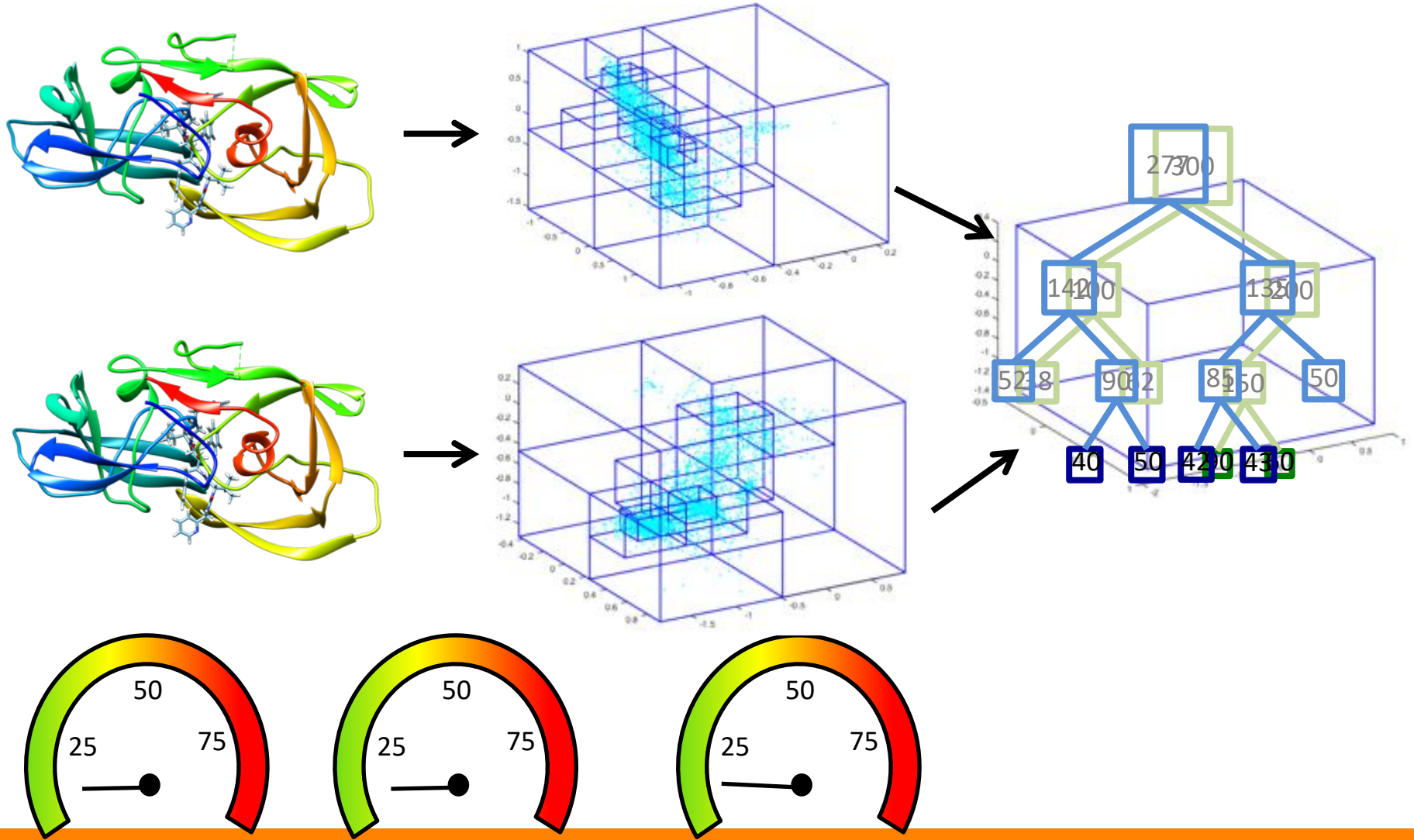
CPU utilization



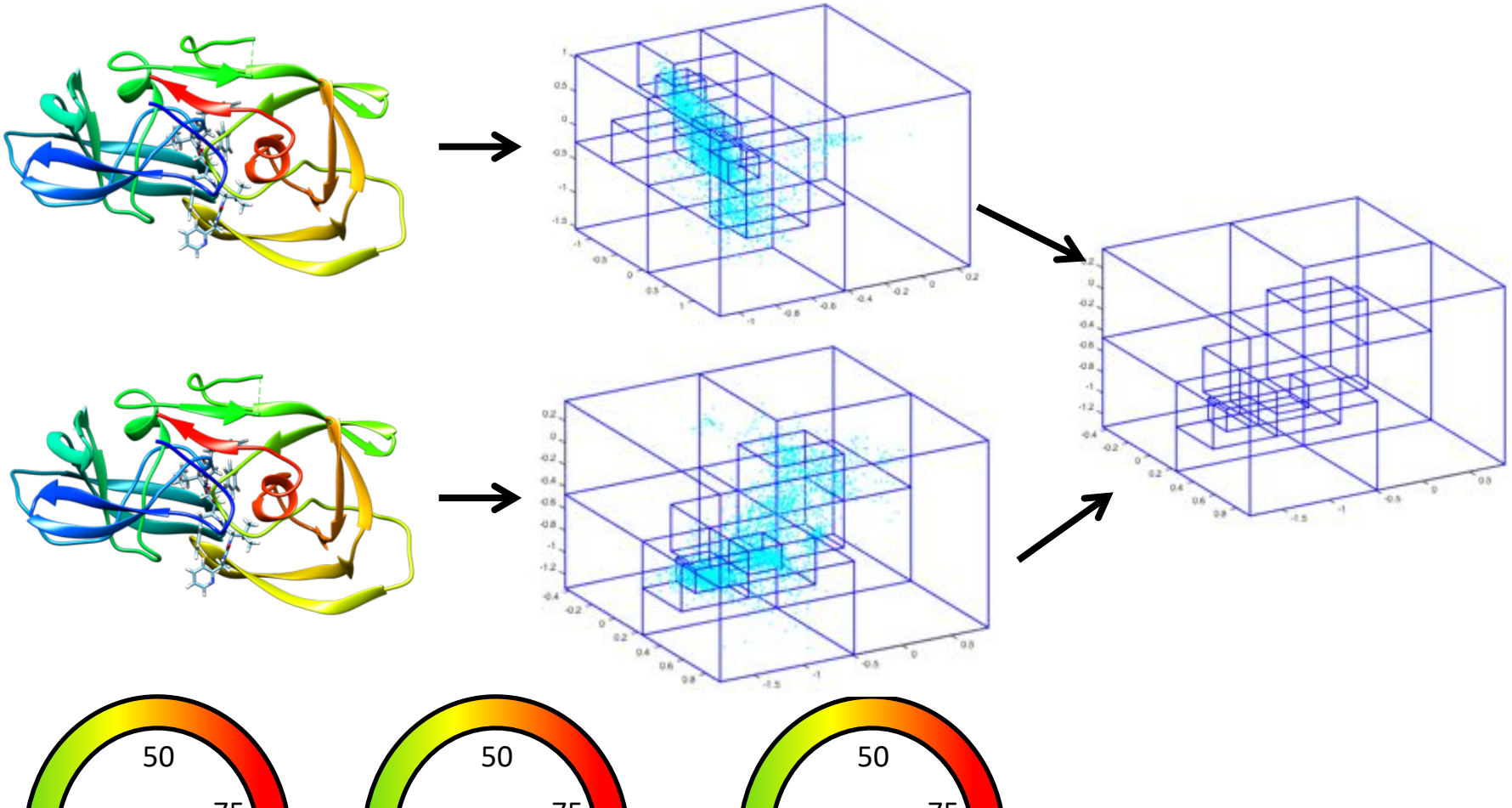
Memory utilization



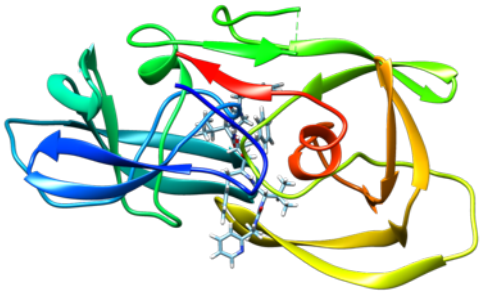
Network utilization

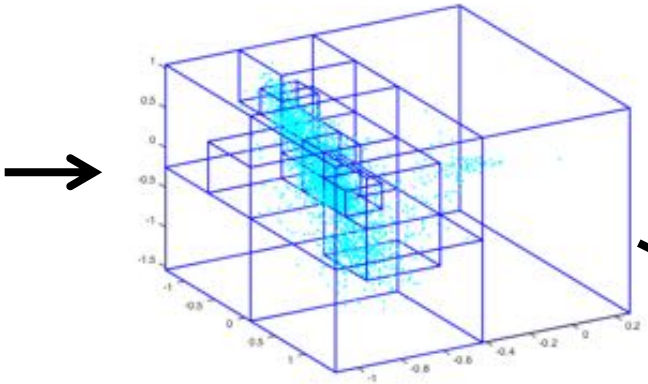


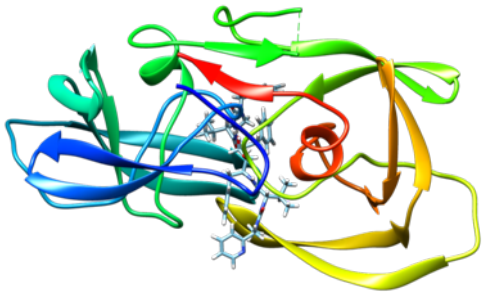
CPU utilization Memory utilization Network utilization

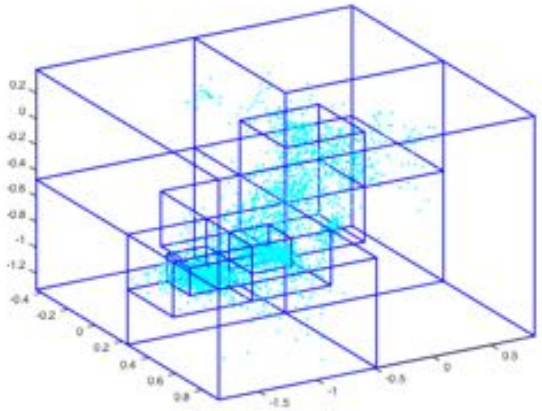


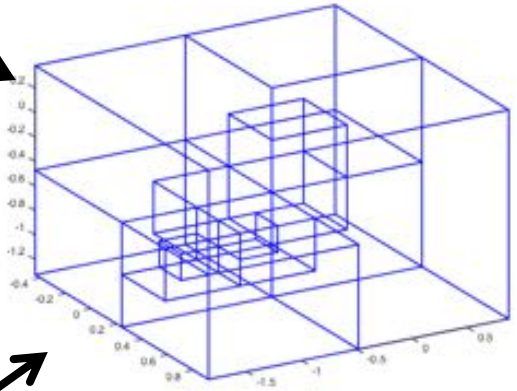
CPU utilization Memory utilization Network utilization

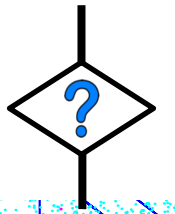


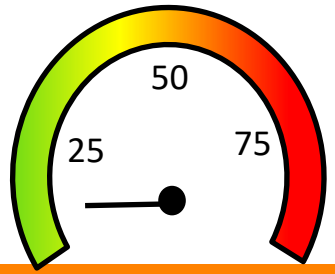


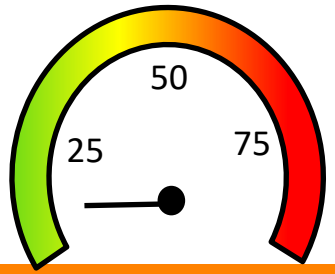


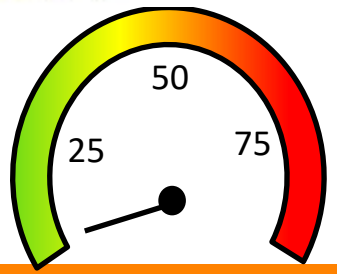


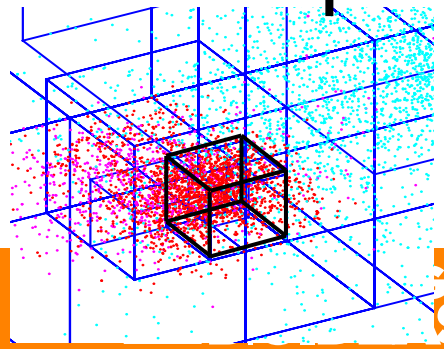




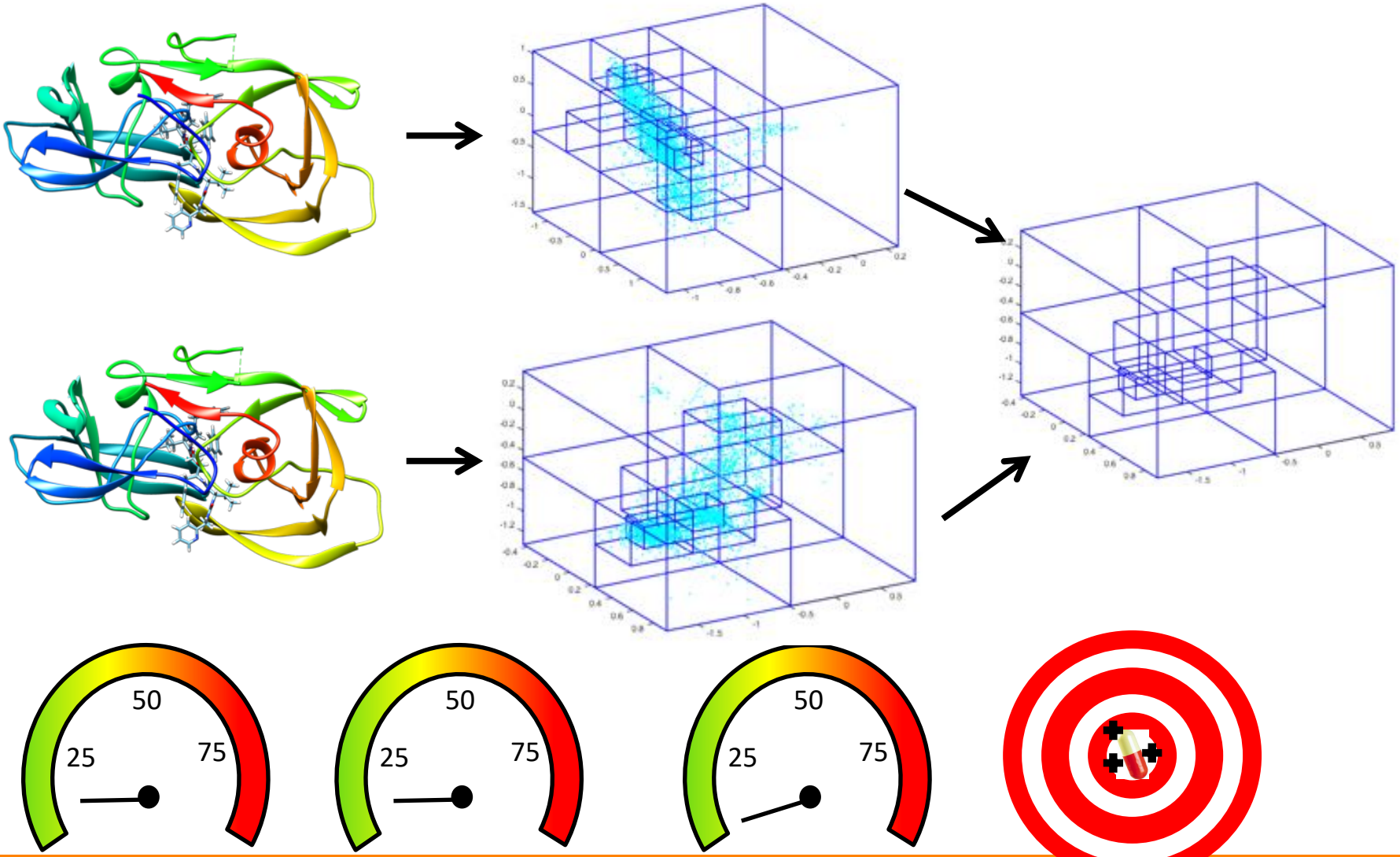




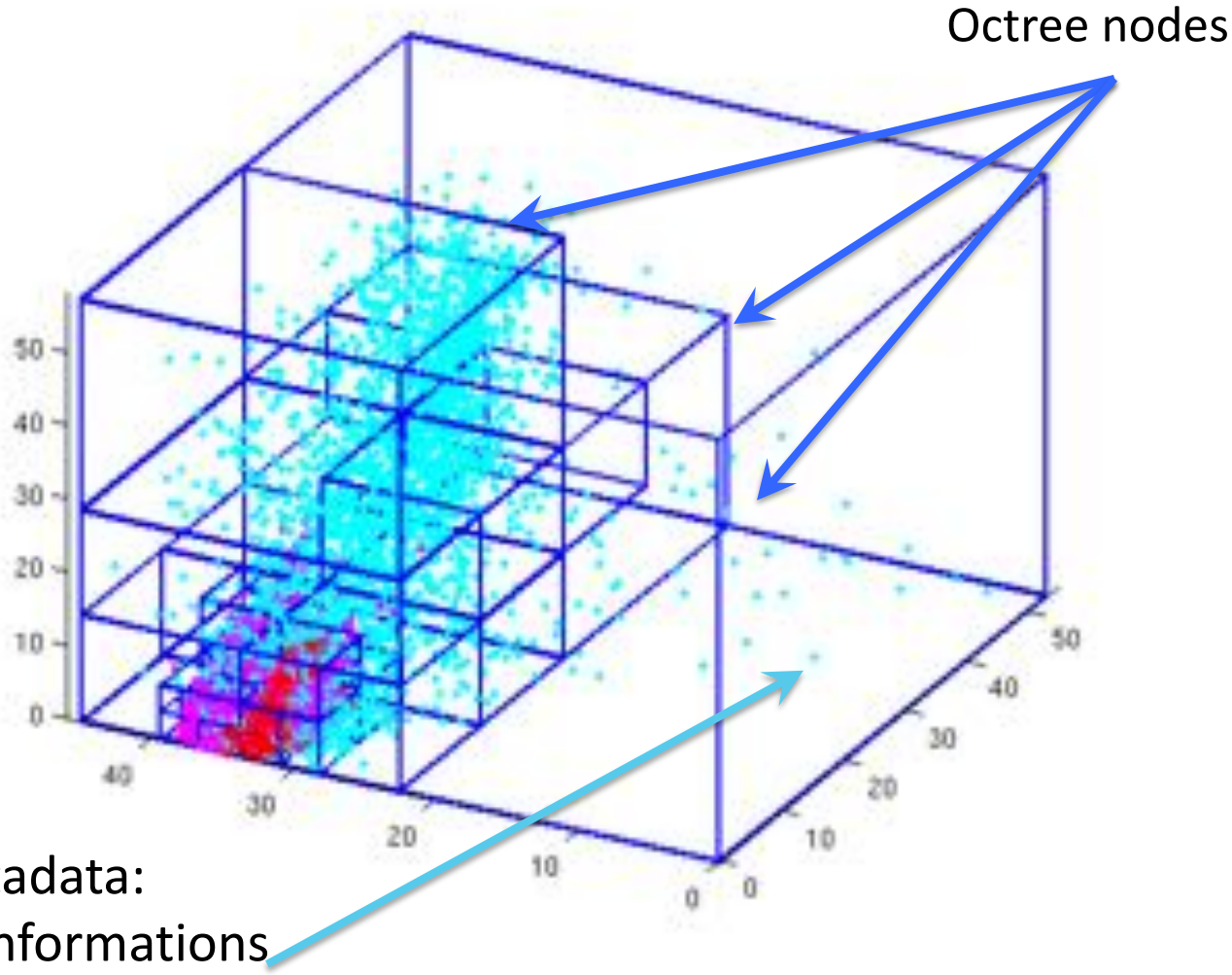




CPU utilization Memory utilization Network utilization


50
25
75
50
25
75
50
25
75
CPU utilization
Memory utilization
Network utilization
Accuracy

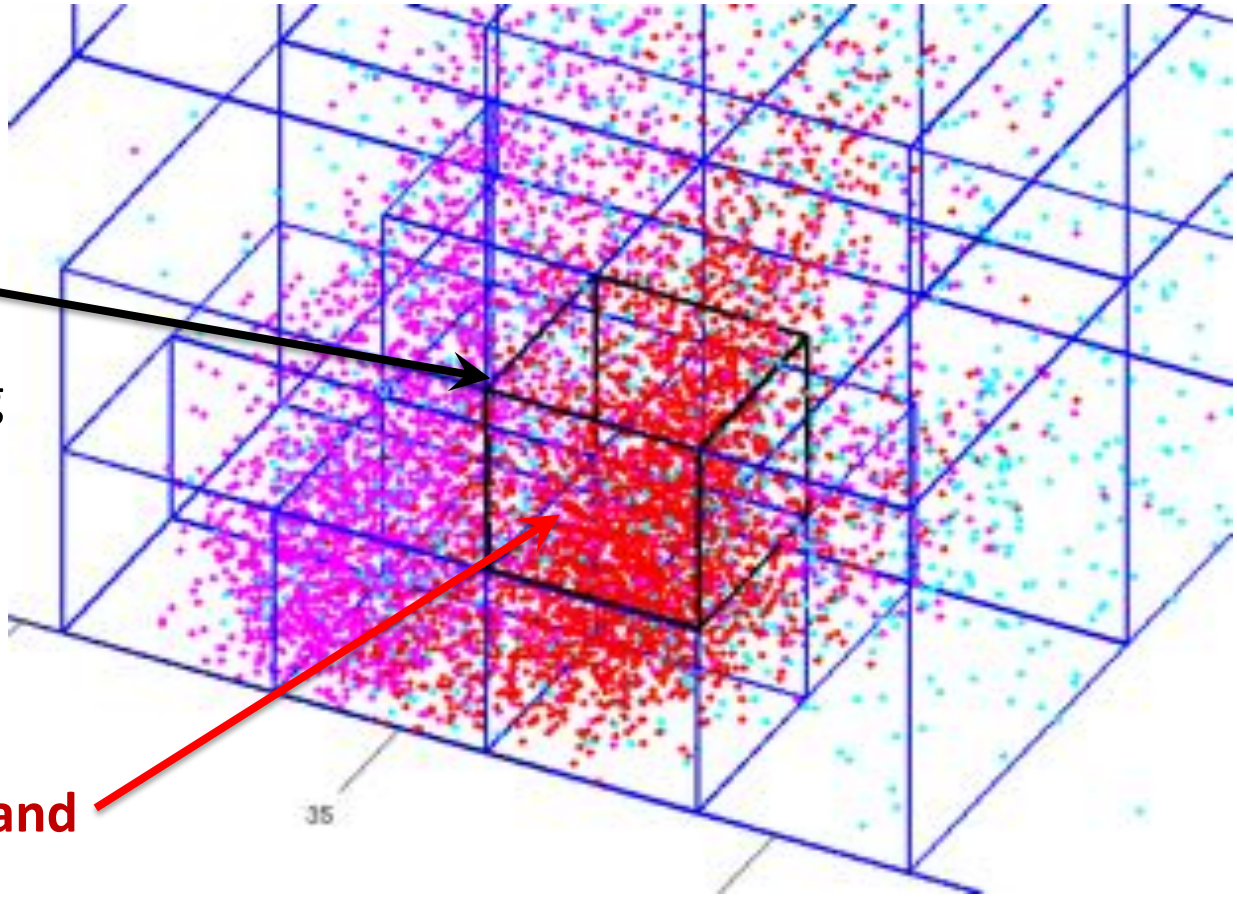
Search for Dense Spaces: Octree Clustering



T. Estrada, B. Zhang, P. Cicotti, R. S. Armen, M. Taufer: **A scalable and accurate method for classifying protein-ligand binding geometries using a MapReduce approach.** Comp. in Bio. and Med. 42(7): 758-771 (2012)

Search for Dense Spaces: Octree Clustering

Deepest, more dense octant found by octree clustering



Near-native ligand structures (RMSD <= 2A)

Search: Linear in complexity using Mimir - a MapReduce over MPI framework

T. Estrada, B. Zhang, P. Cicotti, R. S. Armen, M. Taufer: **A scalable and accurate method for classifying protein-ligand binding geometries using a MapReduce approach.** Comp. in Bio. and Med. 42(7): 758-771 (2012)

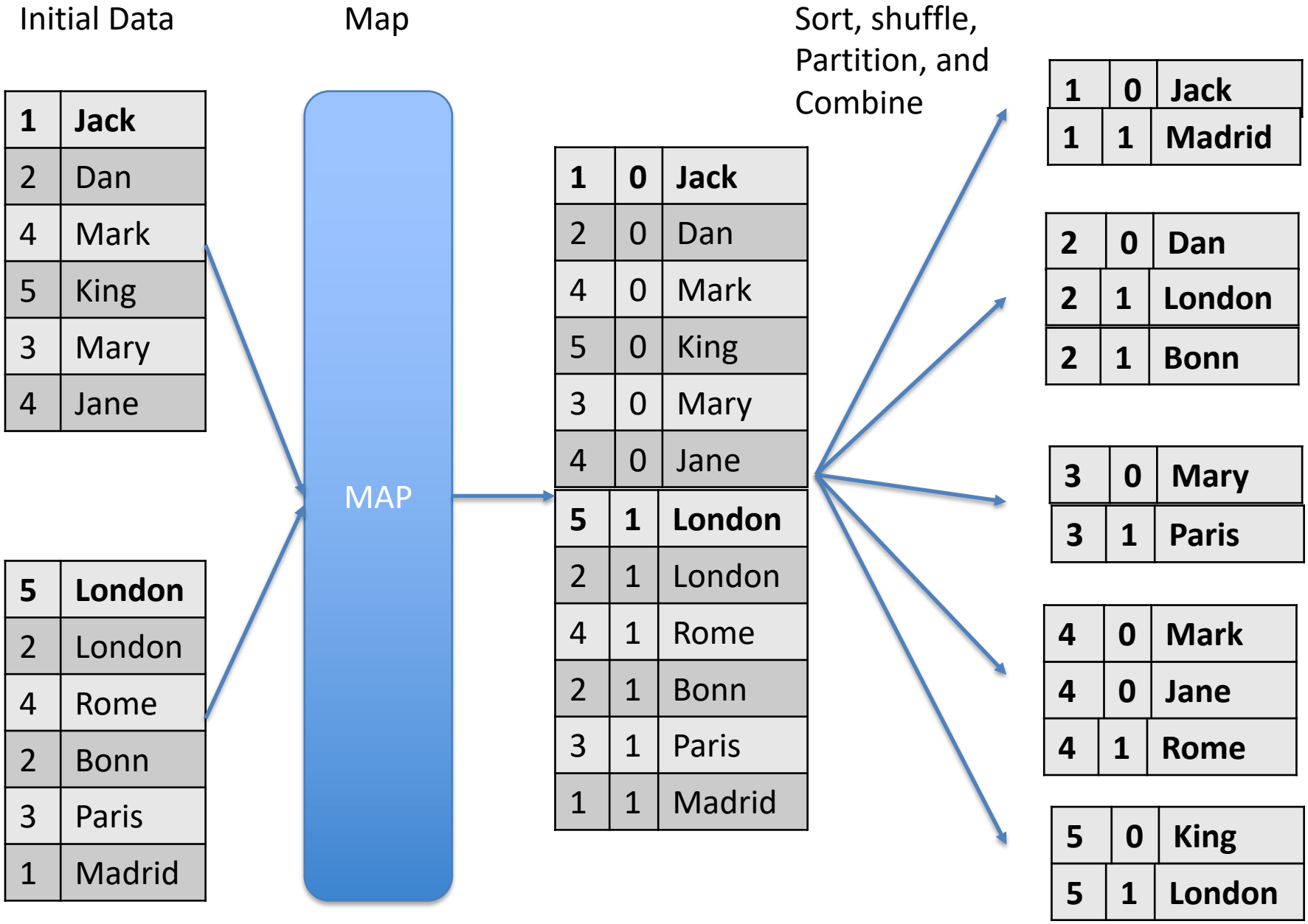
Join Algorithms

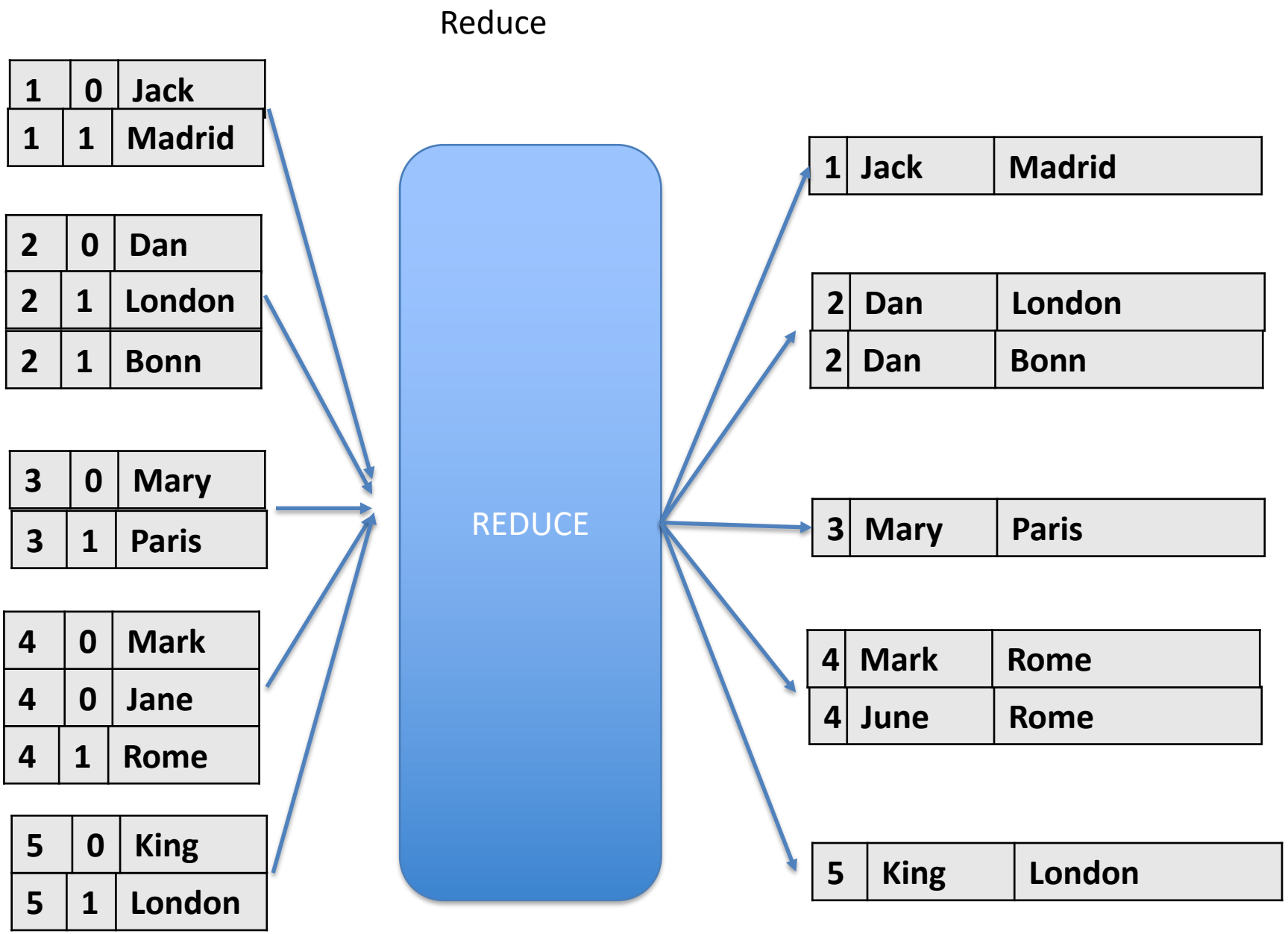
Different algorithms for the join implementation:

- Repartition Join
- Broadcast Join
- Trojan Join
- Replicated Join

From:

- https://www.slideshare.net/shrihari2806/join-algorithms-in-mapreduce
- Anwar Shaikh et al. “Join Query Processing in MapReduce Environment” CNC 2012, LNICST , pp.275-281


Initial Data
Map
Sort, shuffle, Partition, and Combine
MAP
1 Jack
2 Dan
4 Mark
5 King
3 Mary
4 Jane
5 London
2 London
4 Rome
2 Bonn
3 Paris
1 Madrid
1 0 Jack
2 0 Dan
4 0 Mark
5 0 King
3 0 Mary
4 0 Jane
5 1 London
2 1 London
4 1 Rome
2 1 Bonn
3 1 Paris
1 1 Madrid
1 0 Jack
1 1 Madrid
2 0 Dan
2 1 London
2 1 Bonn
3 0 Mary
3 1 Paris
4 0 Mark
4 0 Jane
4 1 Rome
5 0 King
5 1 London


Reduce
1 0 Jack
1 1 Madrid
2 0 Dan
2 1 London
2 1 Bonn
3 0 Mary
3 1 Paris
4 0 Mark
4 0 Jane
4 1 Rome
5 0 King
5 1 London
REDUCE
1 Jack Madrid
2 Dan London
2 Dan Bonn
3 Mary Paris
4 Mark Rome
4 June Rome
5 King London

Benchmark: Join

- Join is a single-pass MR application that merges two datasets to one dataset
- Datasets: larger word log dataset (e.g., data over a window of time); smaller word reference dataset (e.g., information about users)
- Map: <key, value> pairs are generated from the two datasets
  - add a tag to the value to mark the original dataset.
- Shuffle: <key, value> pairs with the same key are grouped and passed to the same reduce function
- Reduce:
  - buffer the <key, value> pairs from the smaller dataset
  - perform a cross-product between records in the two datasets

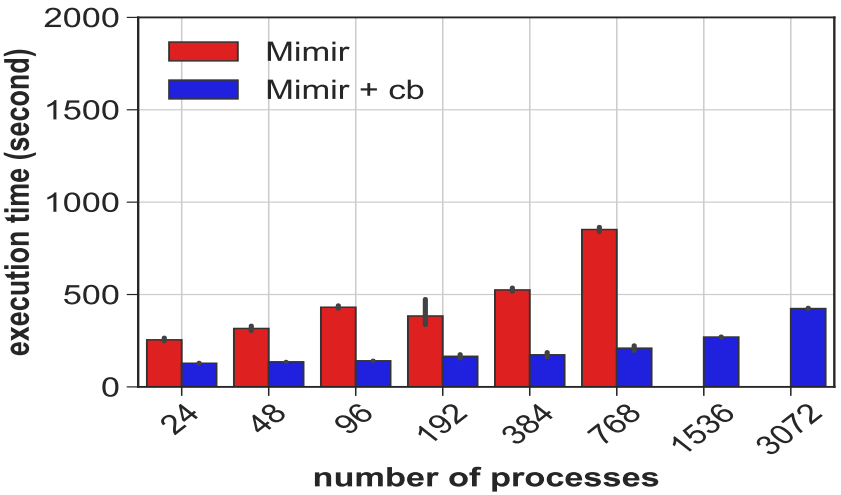
A Comparison of Join Algorithms for Log Processing in MapReduce. Spyros Blanas, Jignesh M. Patel, Vuk Ercegovac, Jun Rao, Eugene J. Shekita, Yuanyuan Tian. 2010

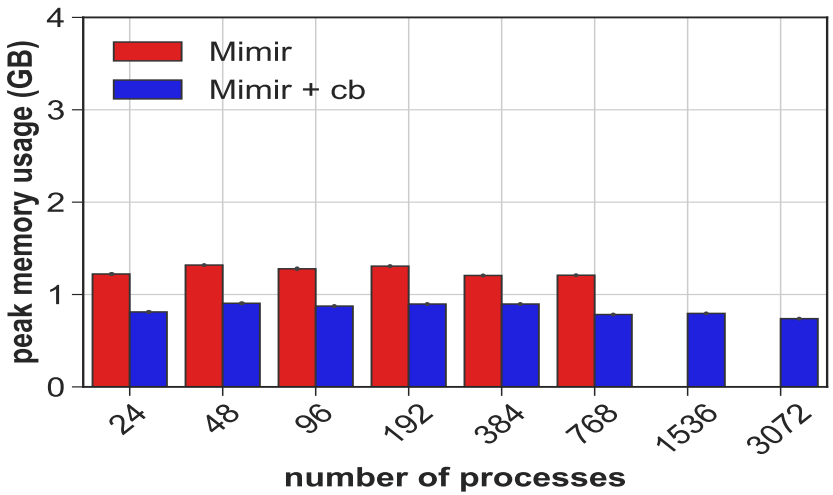
Metrics and Measurements

Metrics and measurements

- Select the proper metrics:
  - **Execution time:** time from reading the input data to outputting the final results of a benchmark
  - **Peak memory usage:** maximum memory usage at any point in time during the benchmark execution
  - **Memory balance ratio:** maximum peak memory usage across all processes divided by the minimum peak memory usage across all processes for the execution
- Perform tests multiple times (at least 3)
- Report standard deviations

Impact of optimizations: WC + Combiners





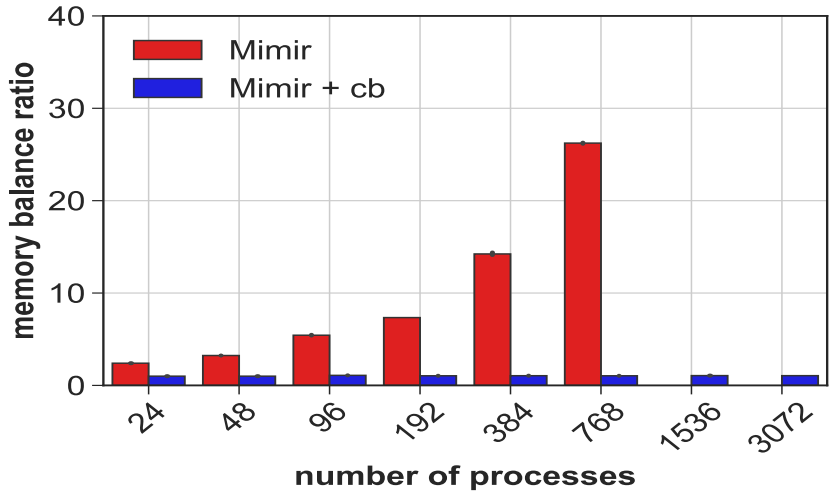
Optimizations:

- **Combiner Optimizations**
- Dynamic Partition

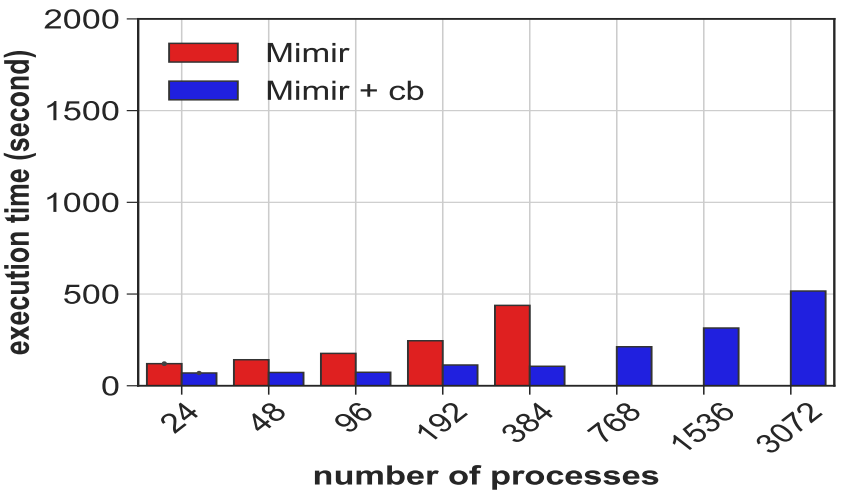
Benchmarks:

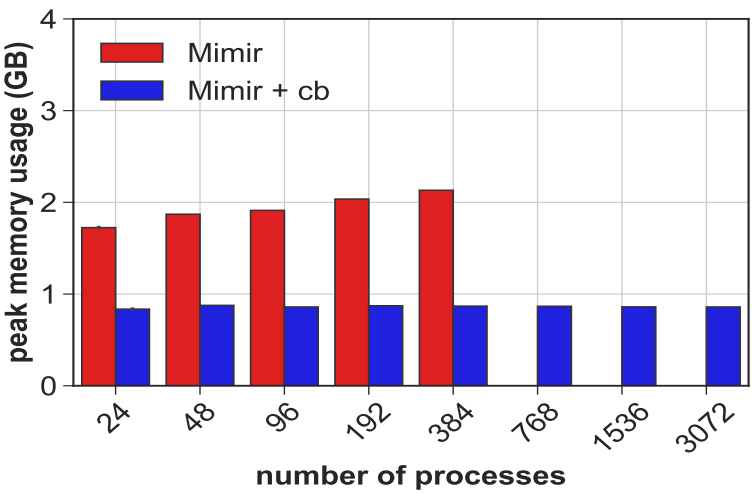
- **Word count (WC)**
- Octree clustering (OC)
- Join

System: Tianhe-2



Impact of optimizations: OC + Combiners





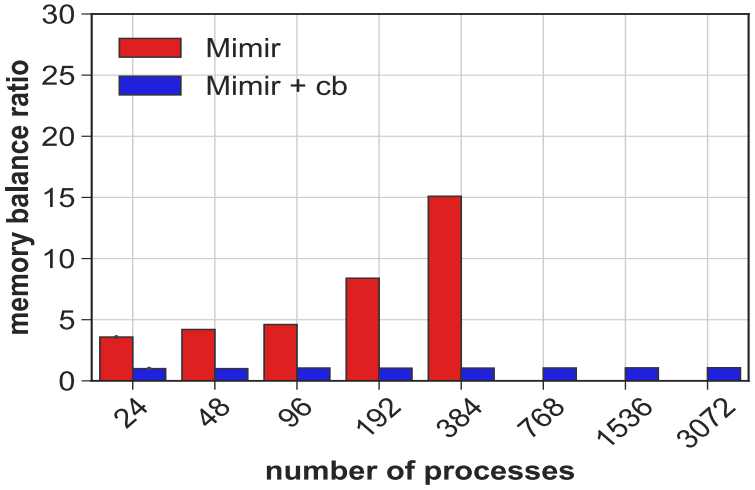
Optimizations:

- **Combiner Optimizations**
- Dynamic Partition

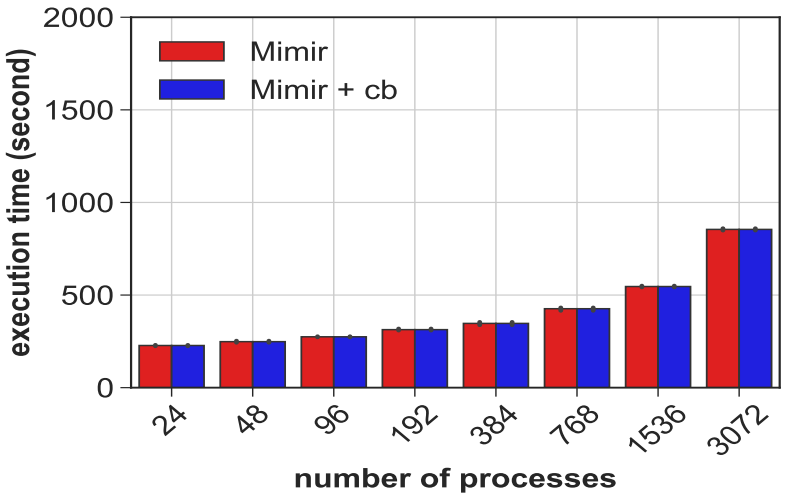
Benchmarks:

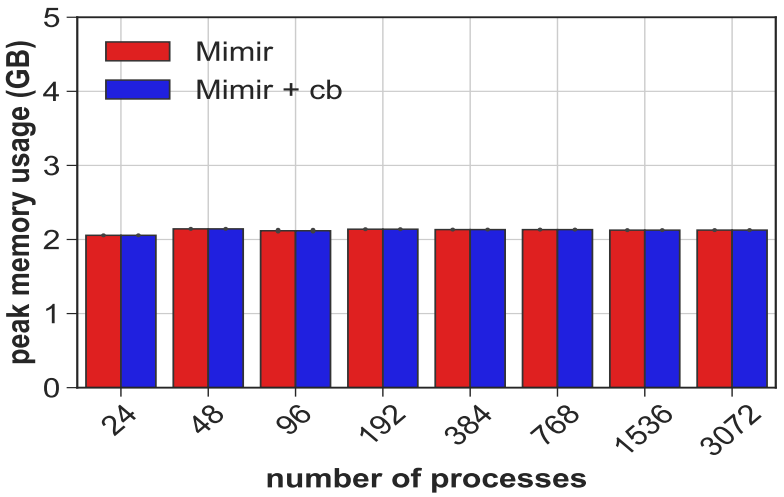
- Word count (WC)
- **Octree clustering (OC)**
- Join

System: Tianhe-2



Impact of optimizations: Join + Combiners





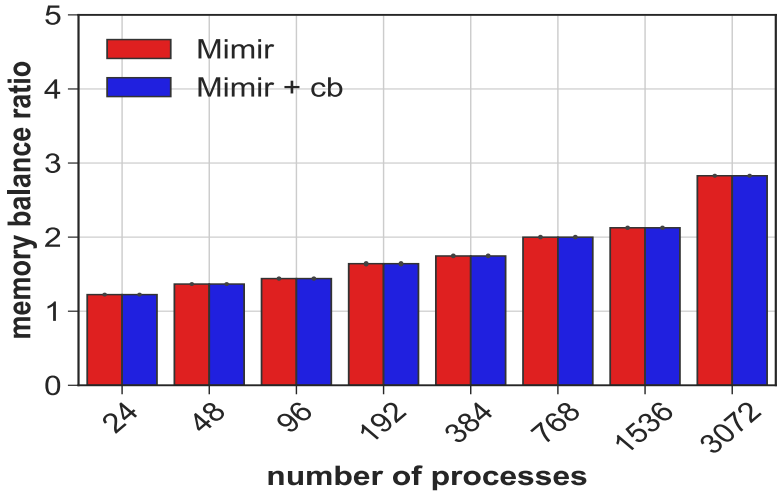
Optimizations:

- **Combiner Optimizations**
- Dynamic Partition

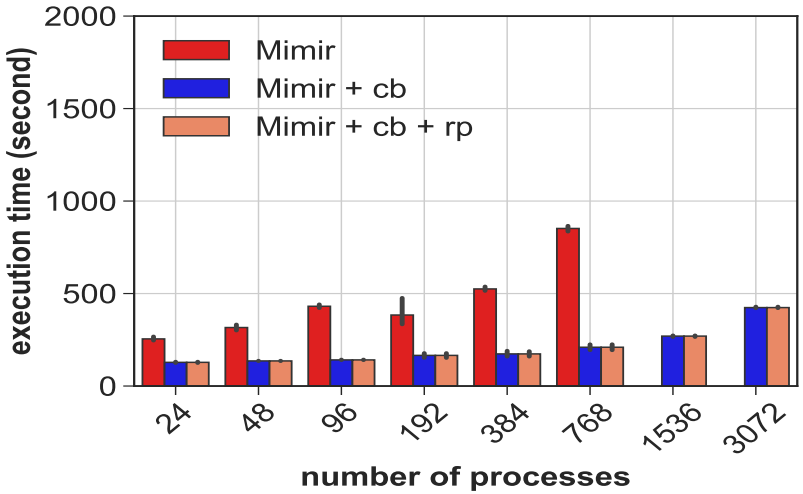
Benchmarks:

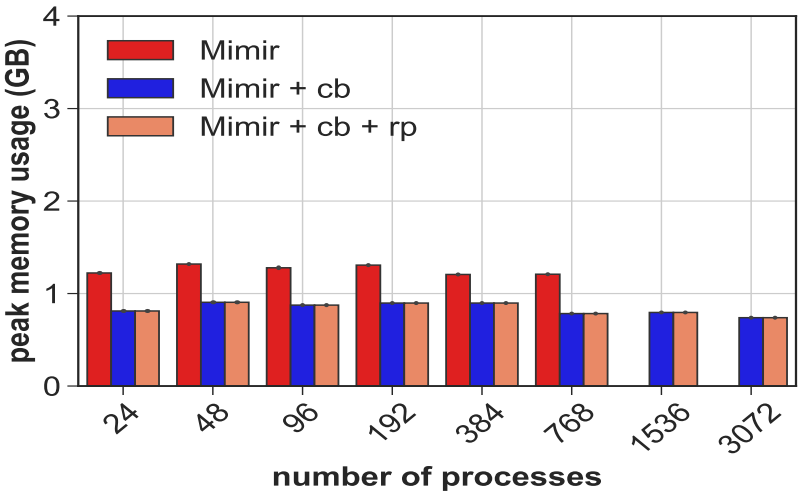
- Word count (WC)
- Octree clustering (OC)
- **Join**

System: Tianhe-2



Impact of optimizations: WC + Partitioning





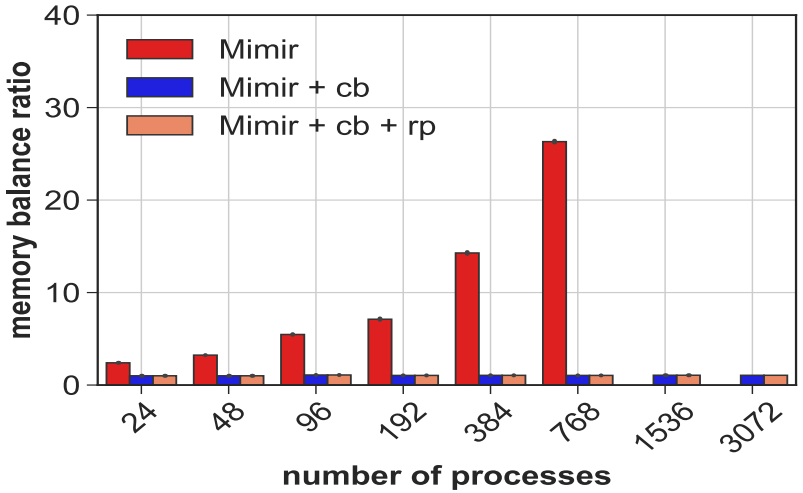
Optimizations:

- **Combiner Optimizations**
- **Dynamic Partition**

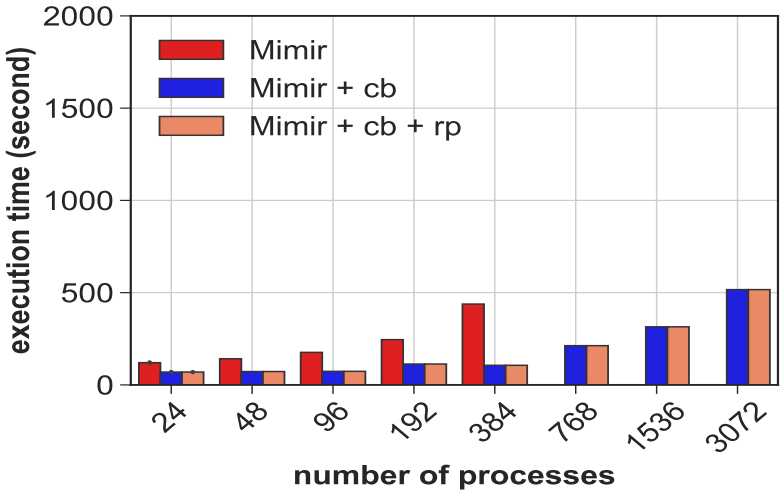
Benchmarks:

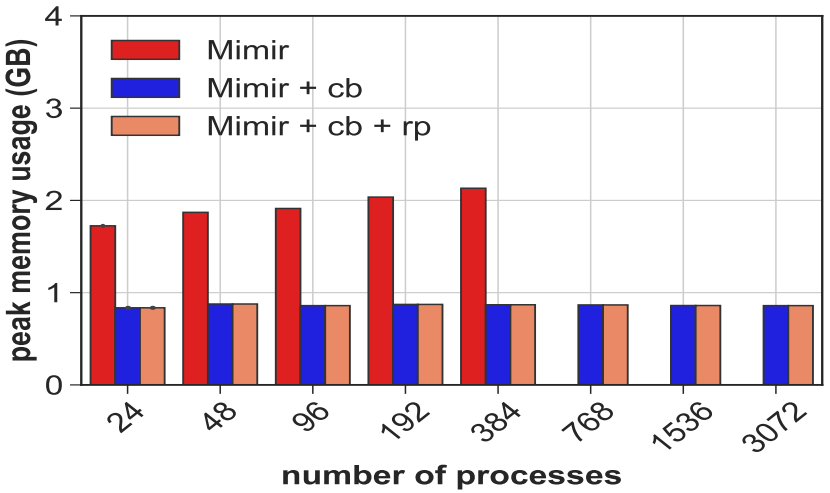
- **Word count (WC)**
- Octree clustering (OC)
- Join

System: Tianhe-2



Impact of optimizations: OC + Partitioning





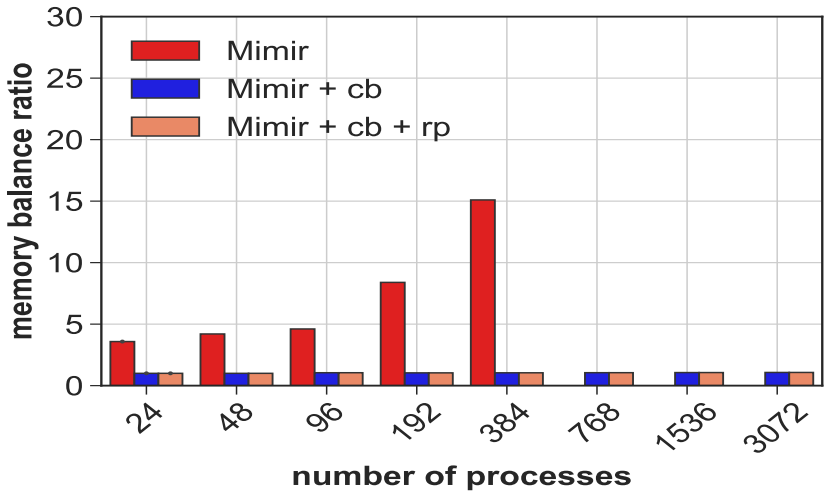
Optimizations:

- **Combiner Optimizations**
- **Dynamic Partition**

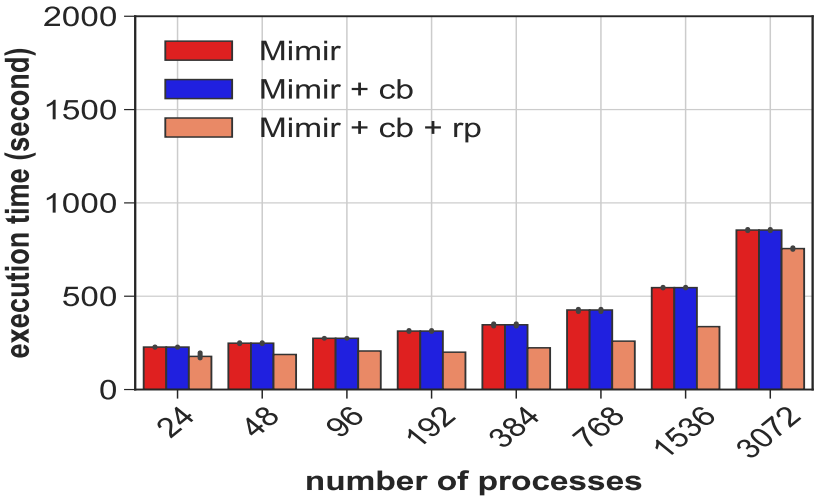
Benchmarks:

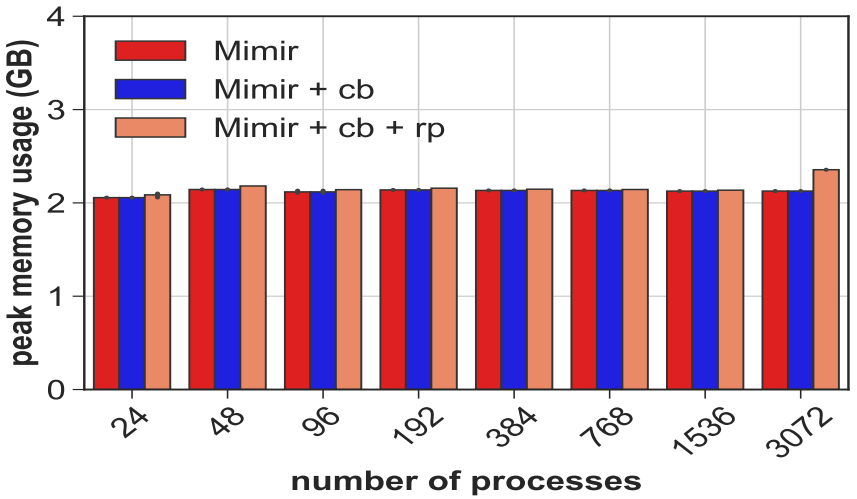
- Word count (WC)
- **Octree clustering (OC)**
- Join

System: Tianhe-2



Impact of optimizations: Join + Partitioning





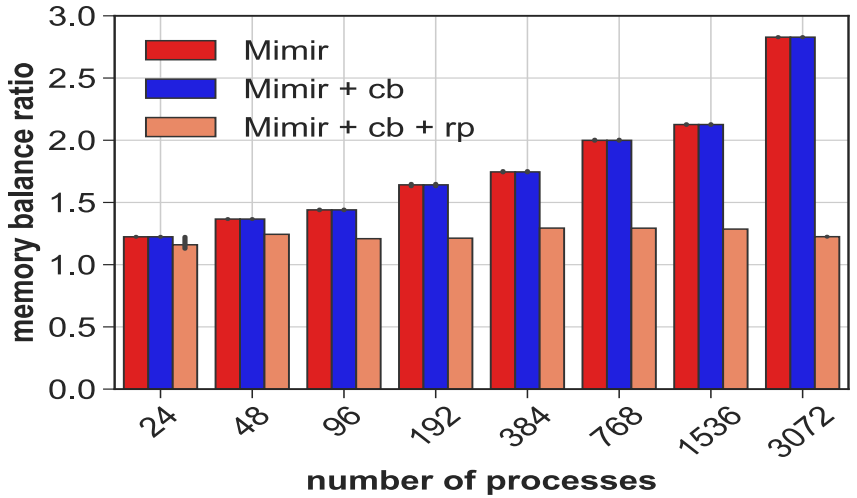
Optimizations:

- **Combiner Optimizations**
- **Dynamic Partition**

Benchmarks:

- Word count (WC)
- Octree clustering (OC)
- **Join**

System: Tianhe-2



One problems,
many solutions

A real problem

- Given a very large log report from a soccer website with the *number of scores per game* of all the soccer players worldwide for seasons 2013 – 2018
  - Keys represent soccer player name
  - Values represent the number of scores a player get per game
  - Data is chronologically sorted based on the date of the game
- **Which players shall be awarded the Soccer's Best Player Award?**
  - Criteria: the winner has the highest mean number of scores per played game
- Compute the mean number of scores on a per-player basis

Things to remember …

Mean(1; 2; 3; 4; 5) **IS NOT** Mean(Mean(1; 2); Mean(3; 4; 5))

A solution

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)

1: class Reducer
2:     method Reduce(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

A solution

Q: Does the solution work?

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)

1: class Reducer
2:     method Reduce(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

A solution

Q: Is this an efficient solution?

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)

1: class Reducer
2:     method Reduce(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

Another solution?

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)
```

```
1: class Combiner
2:     method Combine(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         Emit(string t, pair (sum, cnt))
```

```
1: class Reducer
2:     method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

Another portable solution?

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)
```

```
1: class Combiner
2:     method Combine(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         Emit(string t, pair (sum, cnt))
```

```
1: class Reducer
2:     method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

Another portable solution?

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)

1: class Combiner
2:     method Combine(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         Emit(string t, pair (sum, cnt))

1: class Reducer
2:     method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s_1, c_1), (s_2, c_2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

Another portable solution?

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)
1: class Combiner
2:     method Combine(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         Emit(string t, pair (sum, cnt))
1: class Reducer
2:     method Reduce(string t, pairs [(s_1, c_1), (s_2, c_2) ...])
```

Mismatch between combiner input key-value type and output key-value type violates the MapReduce programming model!!!

A portable and efficient solution

```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, pair (r, 1))
1: class Combiner
2:     method Combine(string t, pairs [(s1, c1), (s2, c2) . . .])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s1, c1), (s2, c2) . . .] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         Emit(string t, pair (sum, cnt))
1: class Reducer
2:     method Reduce(string t, pairs [(s1, c1), (s2, c2) . . .])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s1, c1), (s2, c2) . . .] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

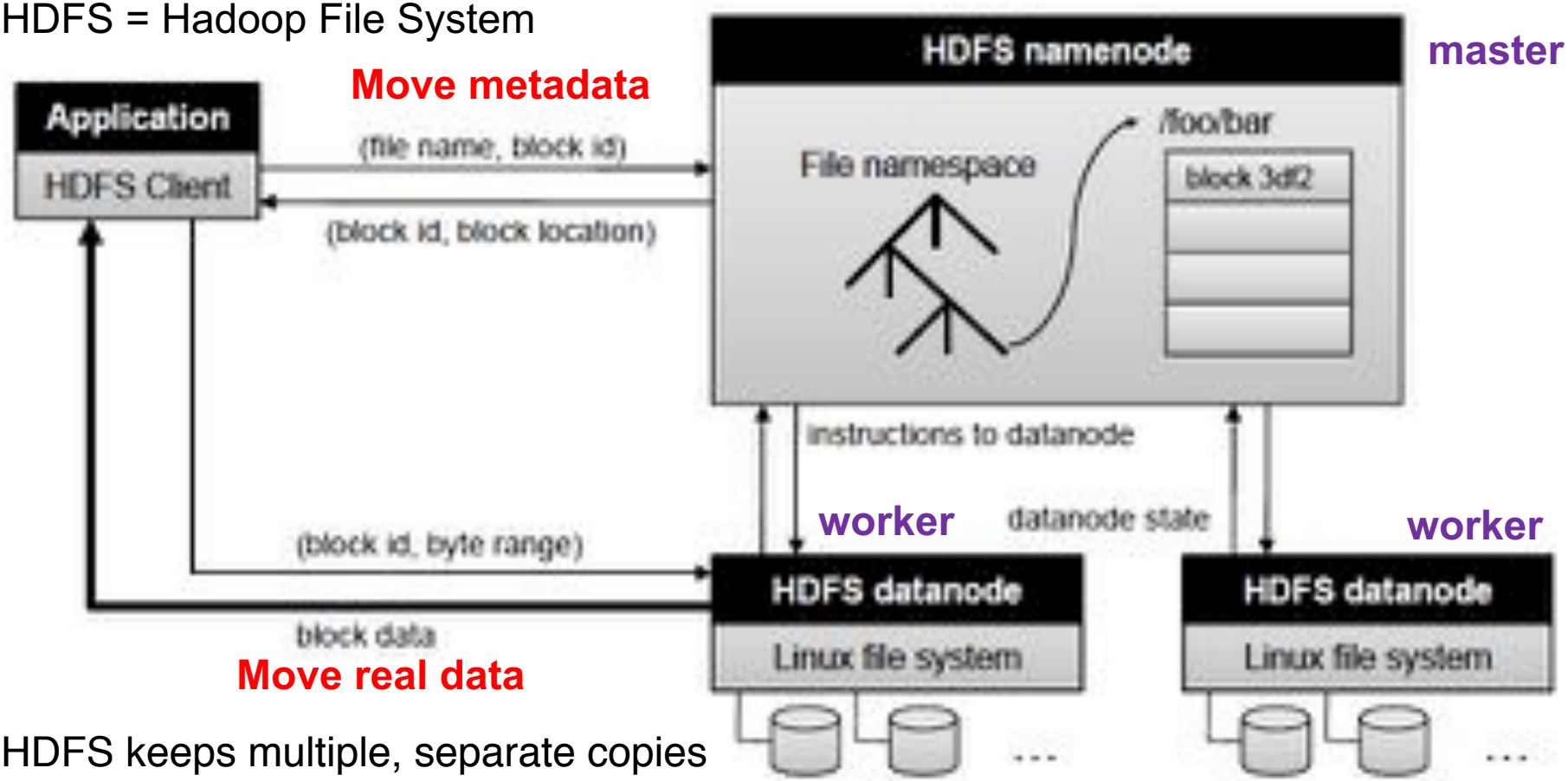
In-mapper Combining Design Pattern Solution

```
1: class Mapper
2:     method Initialize
3:         S ← new AssociativeArray
4:         C ← new AssociativeArray
5:     method Map(string t, integer r)
6:         S{t} ← S{t} + r
7:         C{t} ← C{t} + 1
8:     method Close
9:         for all term t ∈ S do
10:            Emit(term t, pair (S{t}, C{t}))
1: class Reducer
2:     method Reduce(string t, pairs [(s1, c1), (s2, c2) ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all pair (s, c) ∈ pairs [(s1, c1), (s2, c2) ...] do
6:             sum ← sum + s
7:             cnt ← cnt + c
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

Where is the data kept?

The Hadoop File System

HDFS = Hadoop File System



HDFS keeps multiple, separate copies of each data block to ensure reliability, availability, and performance

Assignment 3

Assignment 3

- Goal: Continue building our expertise with Jupyter and Python
  - Sequential manipulation of a classical in literature
  - Visualization of statistics
- Deadline: *: **January 31 - 8AM ET***

Assignment 3: text manipulation

- Given a literature classic such as the “The Count of Monte Cristo”
- Problem 1: Analyze the text for word length frequency
- Problem 2: Analyze the text for letter frequency
- Problem 3: Count the positional frequencies of each letter (first, interior, and last)
- Problem 4: Visualize your findings in histograms (one for each one of Problems 1-3)
  - One code is give to you, write the other two codes
- Repeat with a different manuscript
  - Run your code with e.g., a manuscript written in a different language or an old manuscript

Deadline: ***January 31 - 8AM ET***

Reading material (I)

- **MapReduce in MPI for Large-Scale Graph Algorithms**, S. J. Plimpton and K. D. Devine, Parallel Computing, 37, 610-632 (2011)
- **Mimir: Memory-Efficient and Scalable MapReduce for Large Supercomputing Systems.** T. Gao, Y. Guo, B. Zhang, P. Cicotti, Y. Lu, P. Balaji, and **M. Taufer**. *IPDPS,* 2017
- **Scalable Data Skew Mitigation in MapReduce over MPI for Supercomputing Systems.** T. Gao, Y. Guo, B. Zhang, P. Cicotti, Y. Lu, P. Balaji, and M. Taufer. *IEEE TPDS*, 2019

Reading material (I)

- **Computational Multi-Scale Modeling in Protein-Ligand Docking.** M. Taufer, R.S. Armen, J. Chen, P.J. Teller, and C.L. Brooks III. *IEEE Engineering in Medicine and Biology Magazine,* 2009
- **Enabling In-situ Data Analysis for Large Protein Folding Trajectory Datasets.** B. Zhang, T. Estrada(#), P. Cicotti, and M. Taufer. *IPDPS, 2014*
- **A scalable and accurate method for classifying protein-ligand binding geometries using a MapReduce approach.** T. Estrada, B. Zhang, P. Cicotti, R. S. Armen, and M. Taufer. Comp. in Bio. and Med. 42(7): 758-771, 2012

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

BIG ORANGE. BIG IDEAS.®

BIG ORANGE BIG IDEAS