GiViP: A Visual Profiler for Distributed Graph Processing Systems

Alessio Arleo, Walter Didimo, Giuseppe Liotta and Fabrizio Montecchiani

University of Perugia, Italy

The Value of Big Graphs

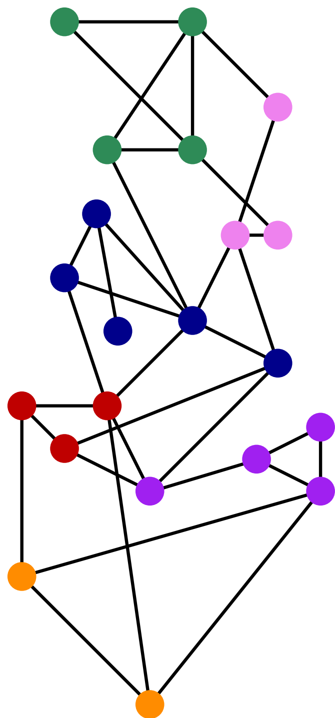
The analysis of large-scale graphs provides valuable insights in different application scenarios: web analysis, social networking, content ranking and recommendations...

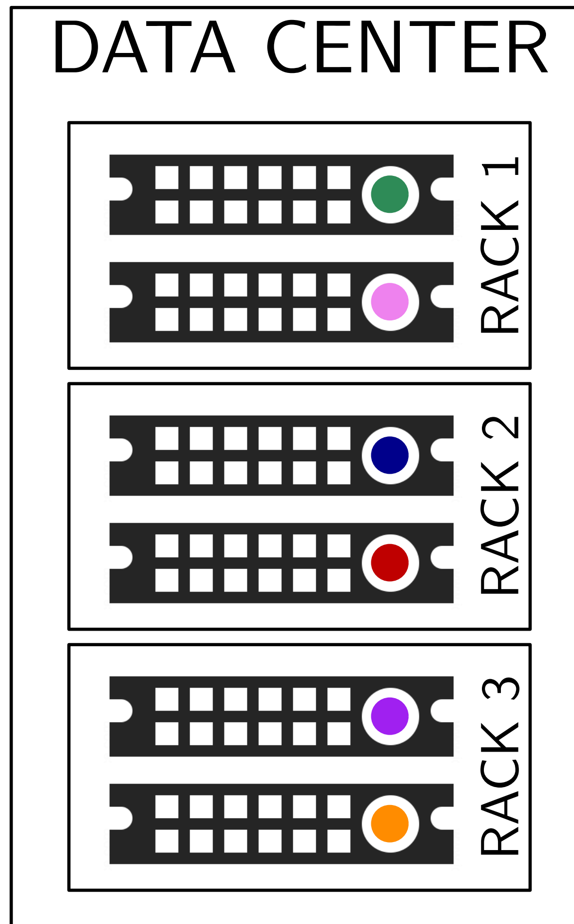
http://internet-map.net

How to Process Big Data

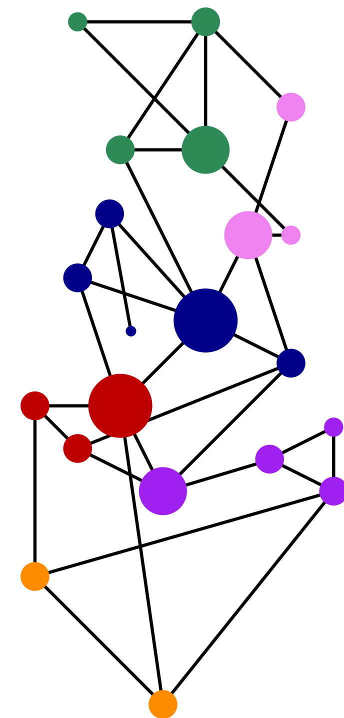
The Big Data processing paradigm: distribute the input to a cluster of inexpensive computers, process the data locally as much as possible, exchange the results

WEB GRAPH





WEB GRAPH WITH PAGE-RANK



How to Process Big Graphs

Many technologies inspired by this paradigm, among them:

- Google Pregel/Apache Giraph for graphs

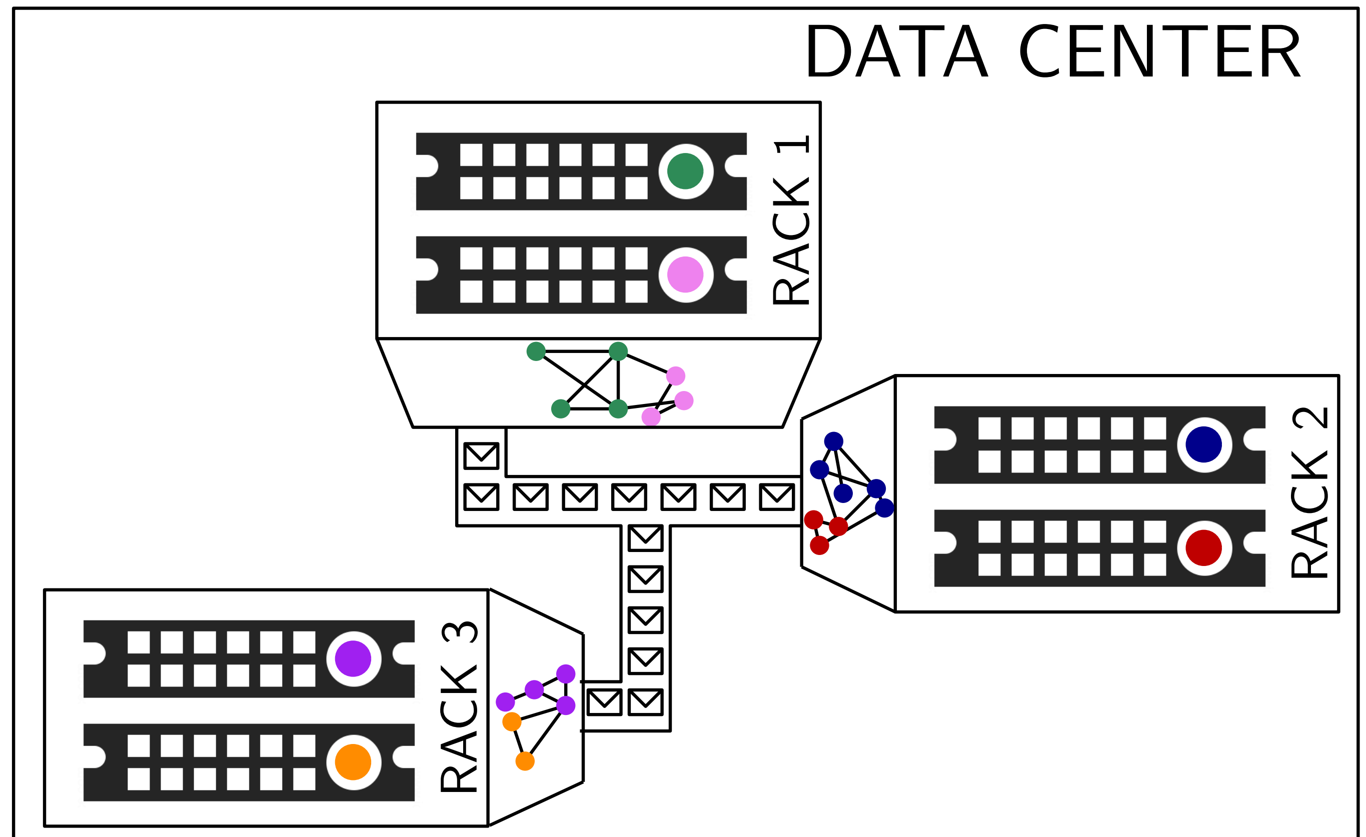
How to Process Big Graphs

Many technologies inspired by this paradigm, among them:

- Google Pregel/Apache Giraph for graphs

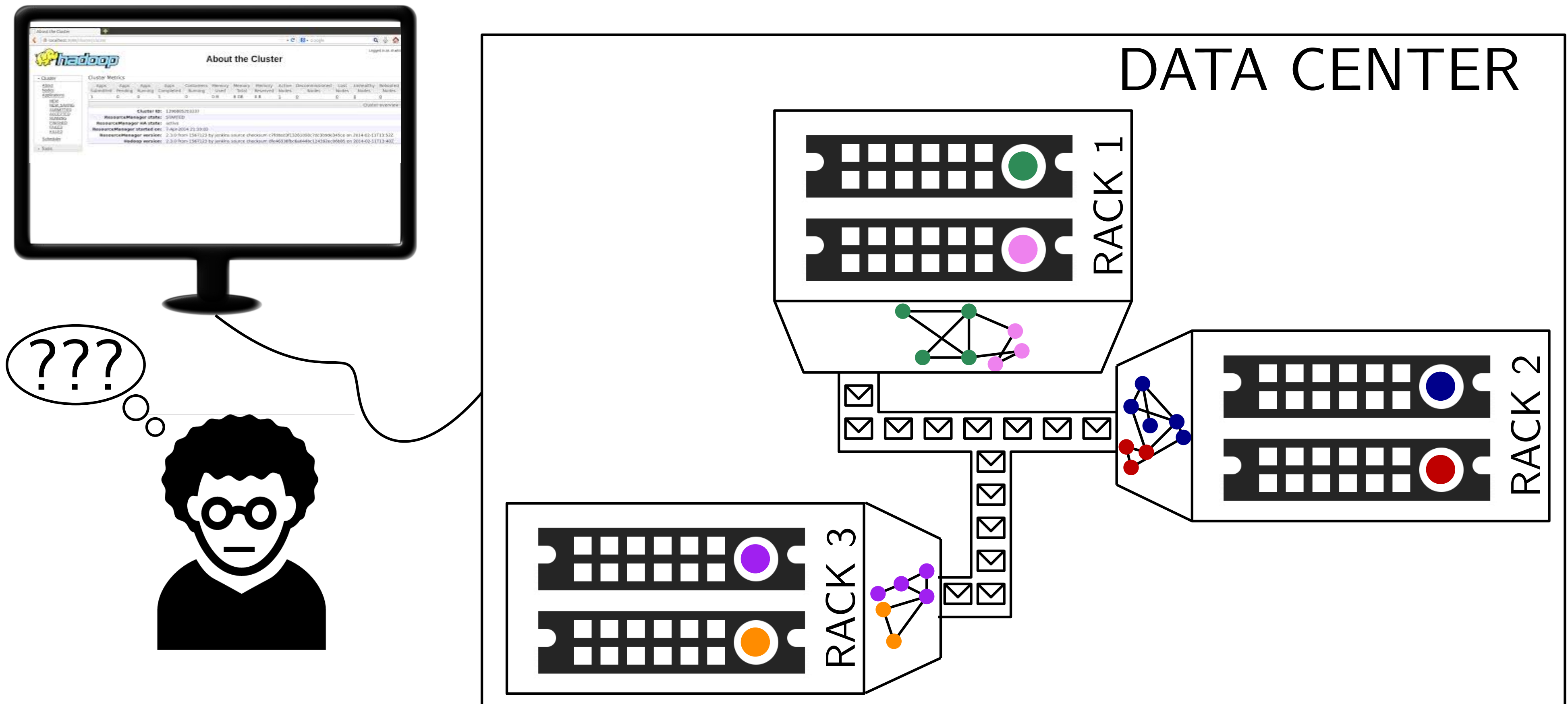
HOW DO THEY WORK?

Think-Like-A-Vertex (TLAV) programming model: design the algorithm from the vertex perspective...



Profiling Massive Computations

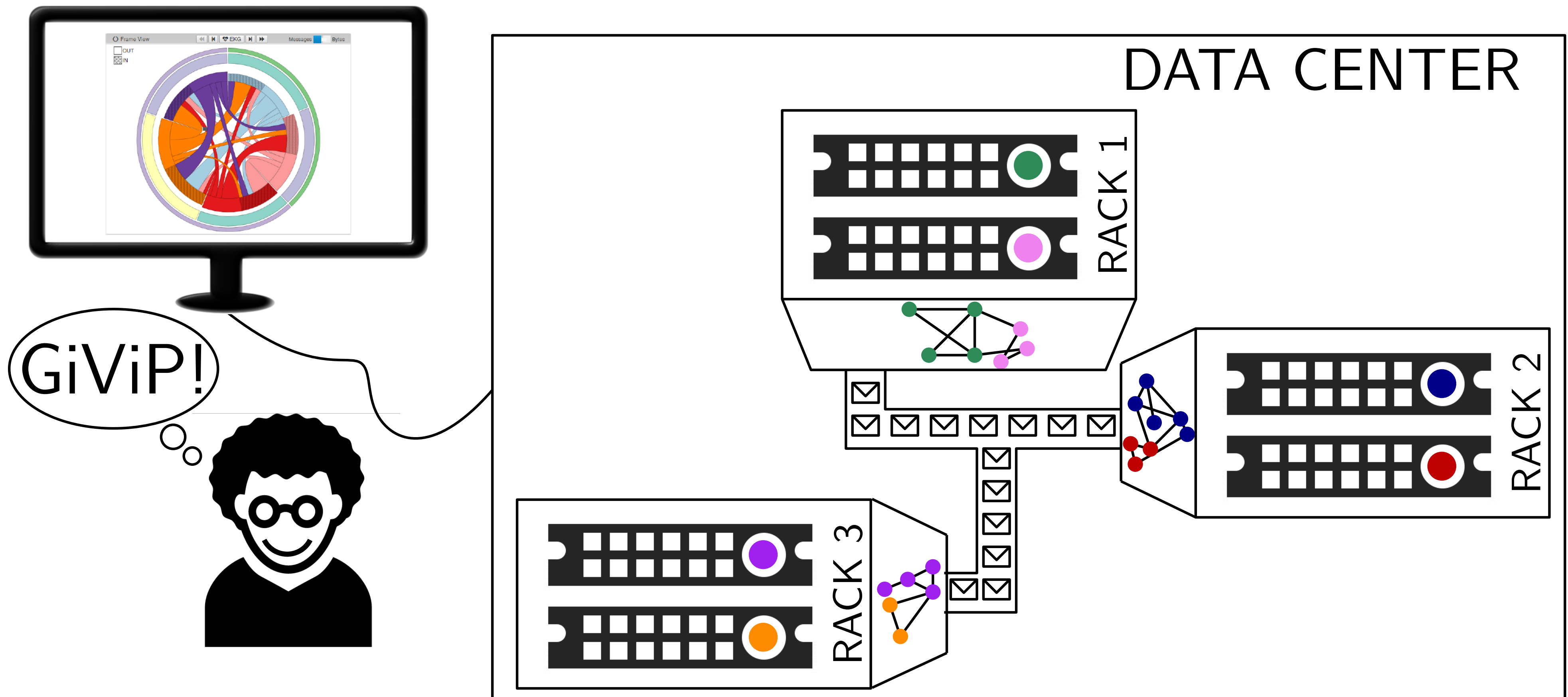
TLAV-based graph processing systems are being adopted by a growing number of applications...but profiling and debugging their massive computations remain time consuming and error-prone tasks...



Contribution

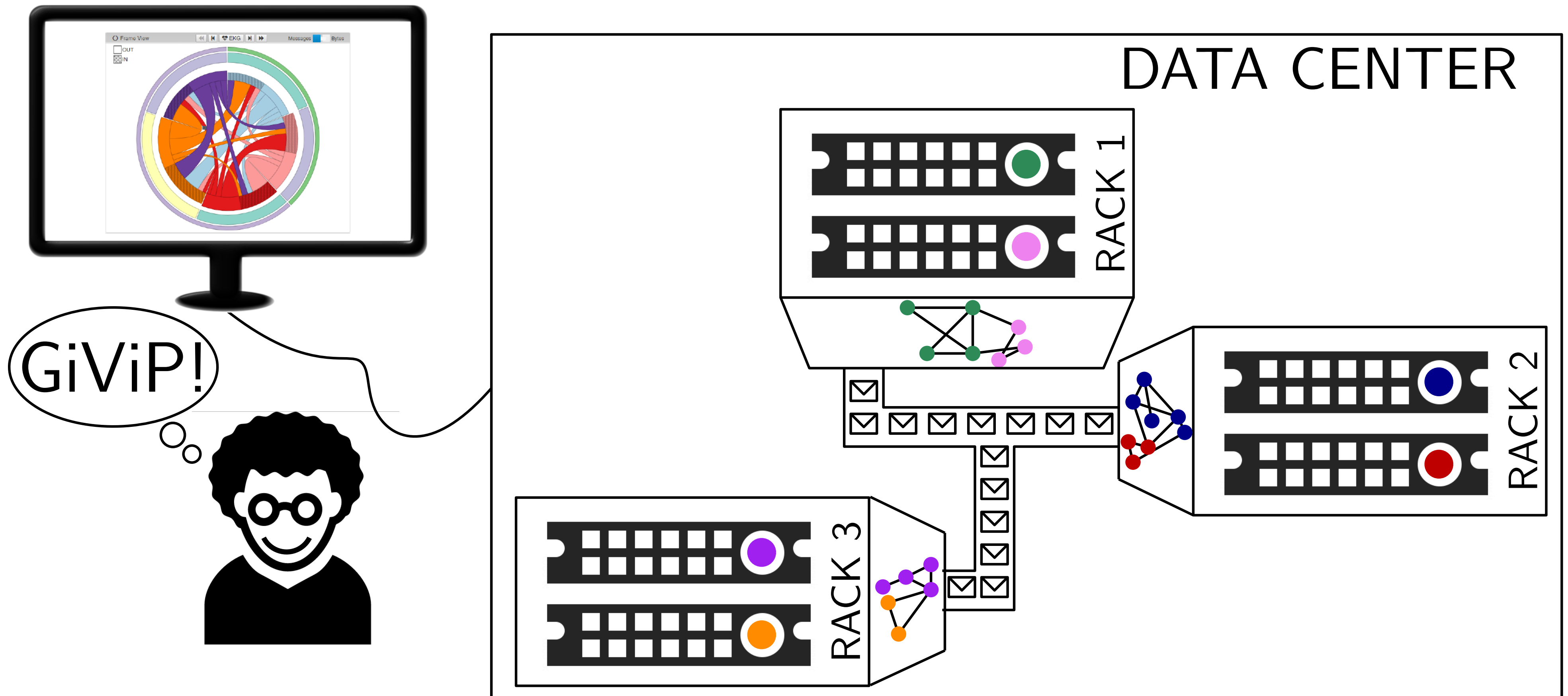
GiViP (Giraph Visual Profiler):

- it collects the data related to messages exchanged by pairs of computing units throughout a computation
- it constructs suitable aggregations of these data, and
- it offers an interactive visual interface to explore the data



Contribution

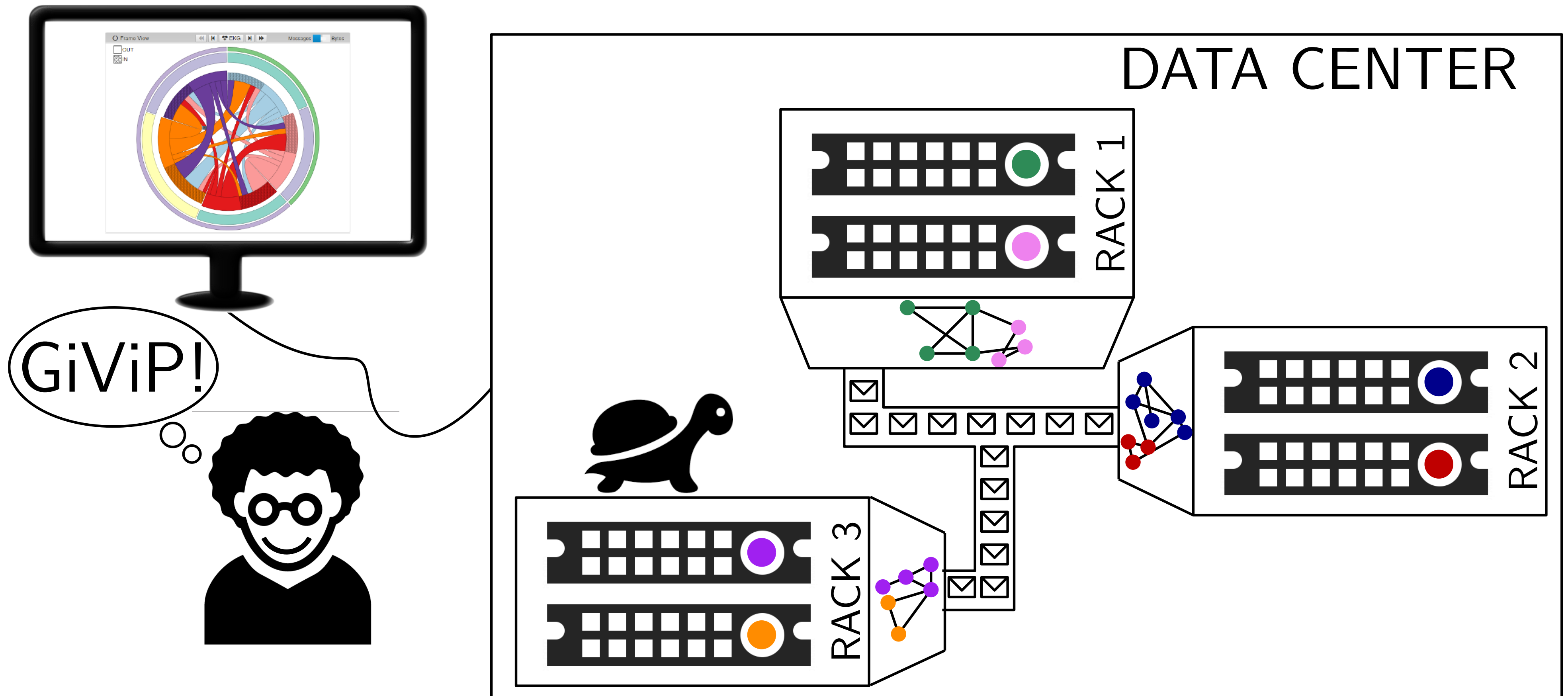
We discuss key usage scenarios of GiViP such as:



Contribution

We discuss key usage scenarios of GiViP such as:

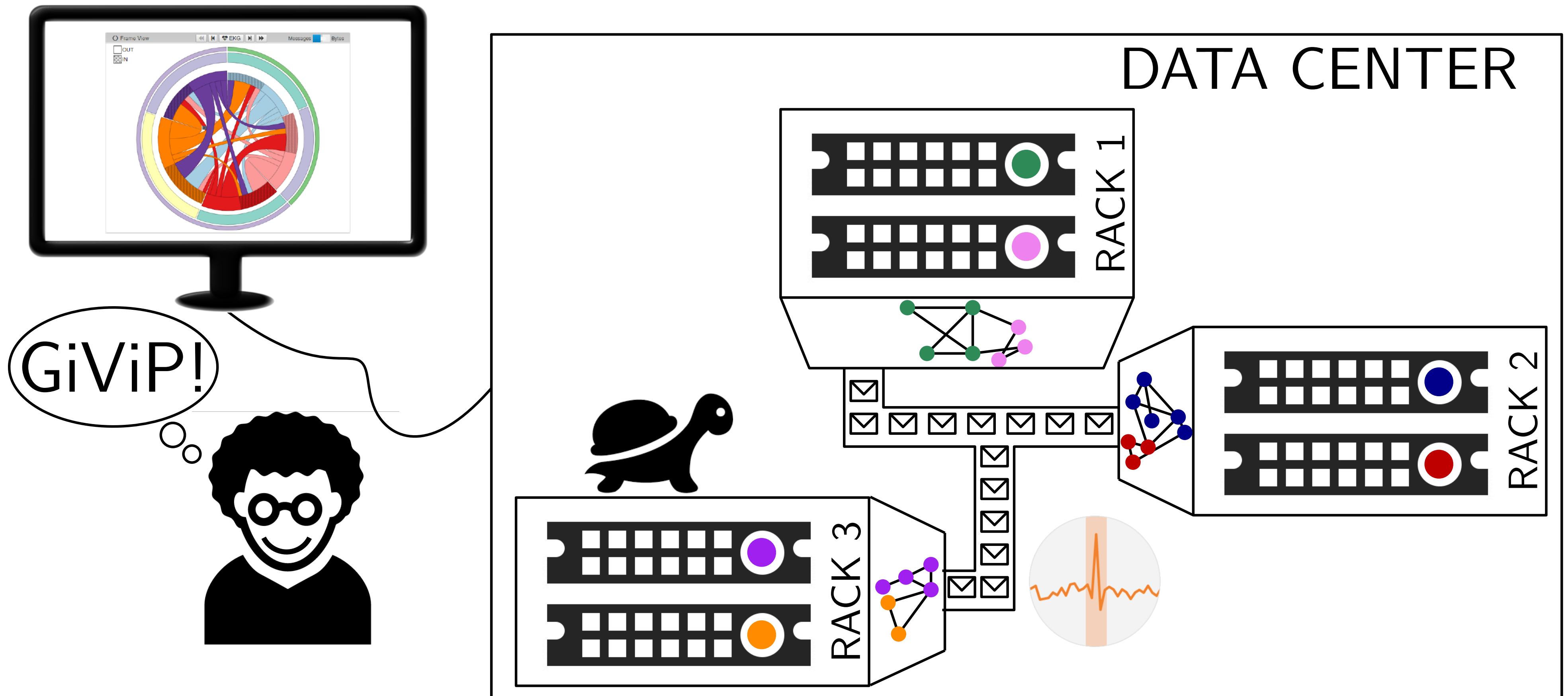
- overloaded computing units, and



Contribution

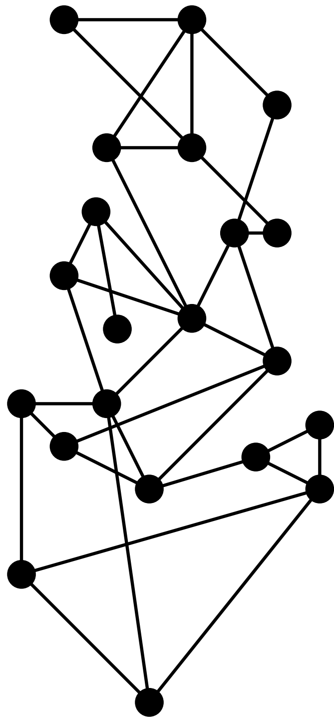
We discuss key usage scenarios of GiViP such as:

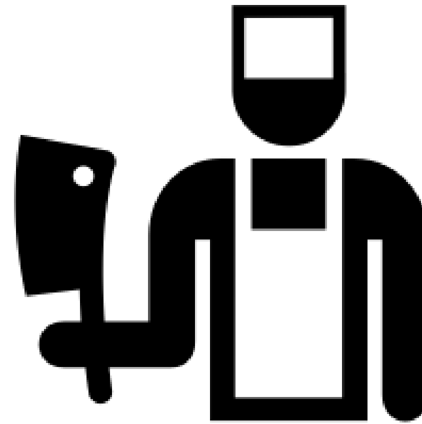
- overloaded computing units, and
- anomalous message patterns.

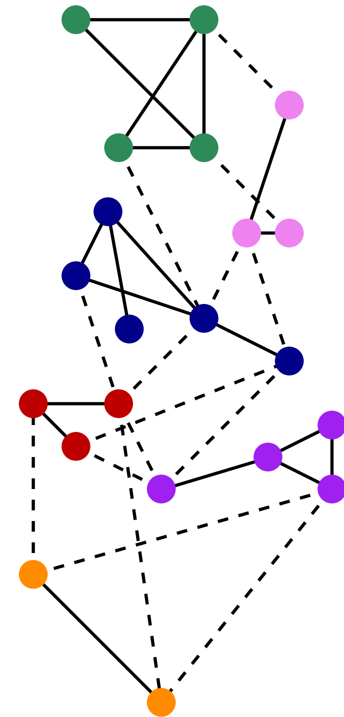


The TLAV Processing Model

First, the input graph is partioned into k parts, with k equal to the number of workers (computation threads, ≥ 1 per computer) available in the cluster





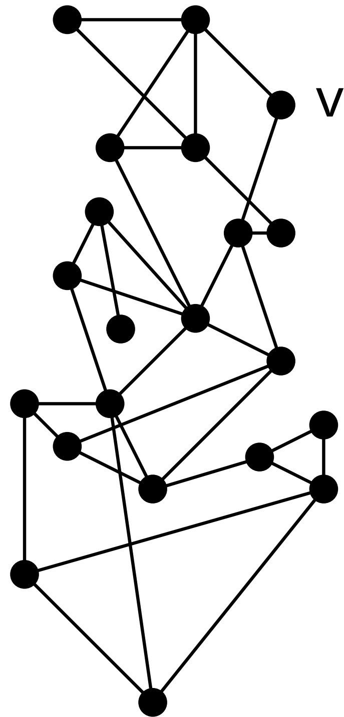


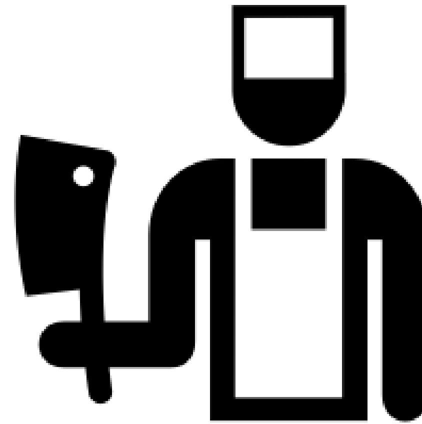
The TLAV Processing Model

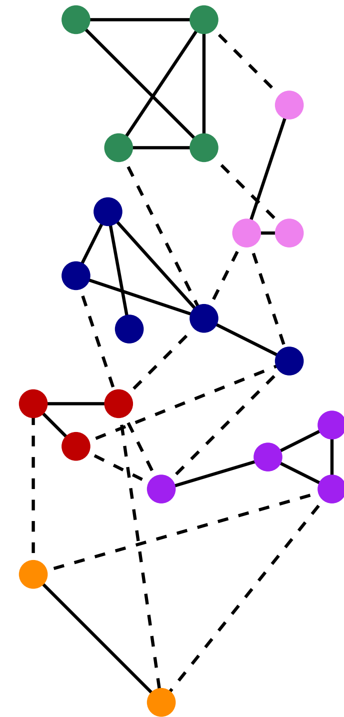
First, the input graph is partioned into k parts, with k equal to the number of workers (computation threads, ≥ 1 per computer) available in the cluster

By default, the partitioner is just a hash function...

part(v) $=$ hash(v.ID) mod k $=$

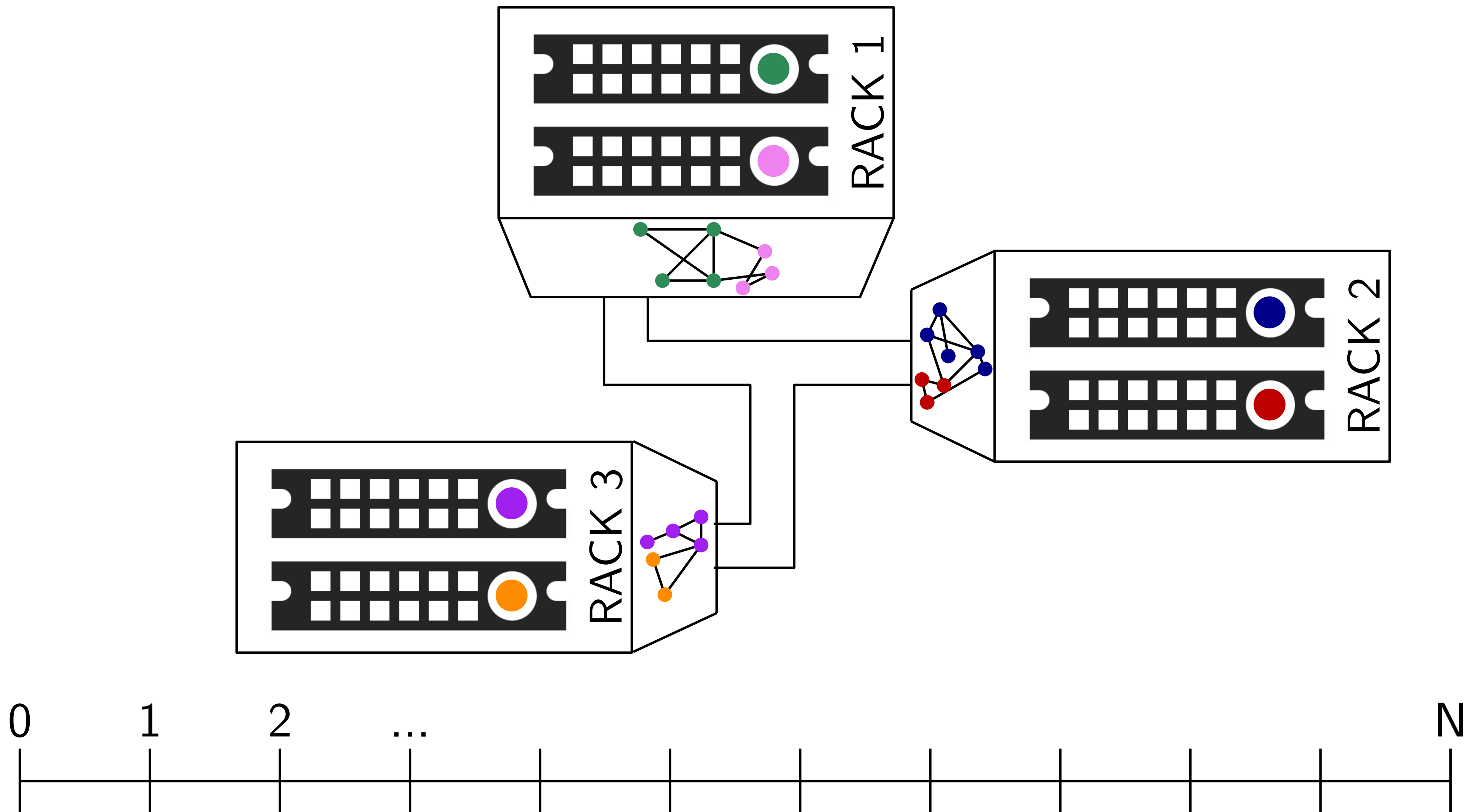






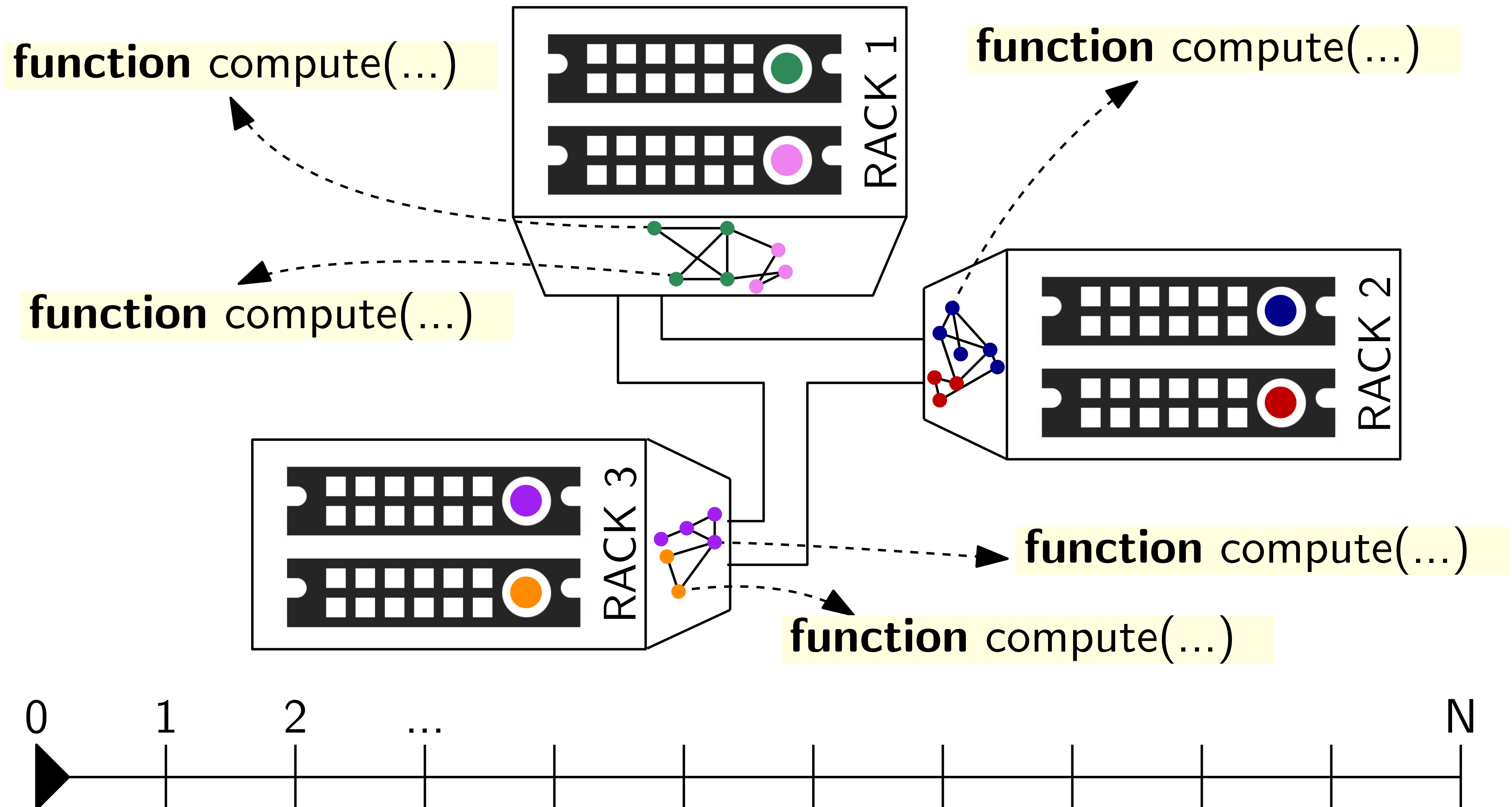
The TLAV Processing Model

The computation is divided into a set of synchronized iterations, called superstеps



The TLAV Processing Model

For each superstep, every worker calls a compute function on each of its vertices



The TLAV Processing Model

The compute function can change the vertex 's state based on messages received from the previous superstep, and then can send new messages through the vertex's outgoing edges.

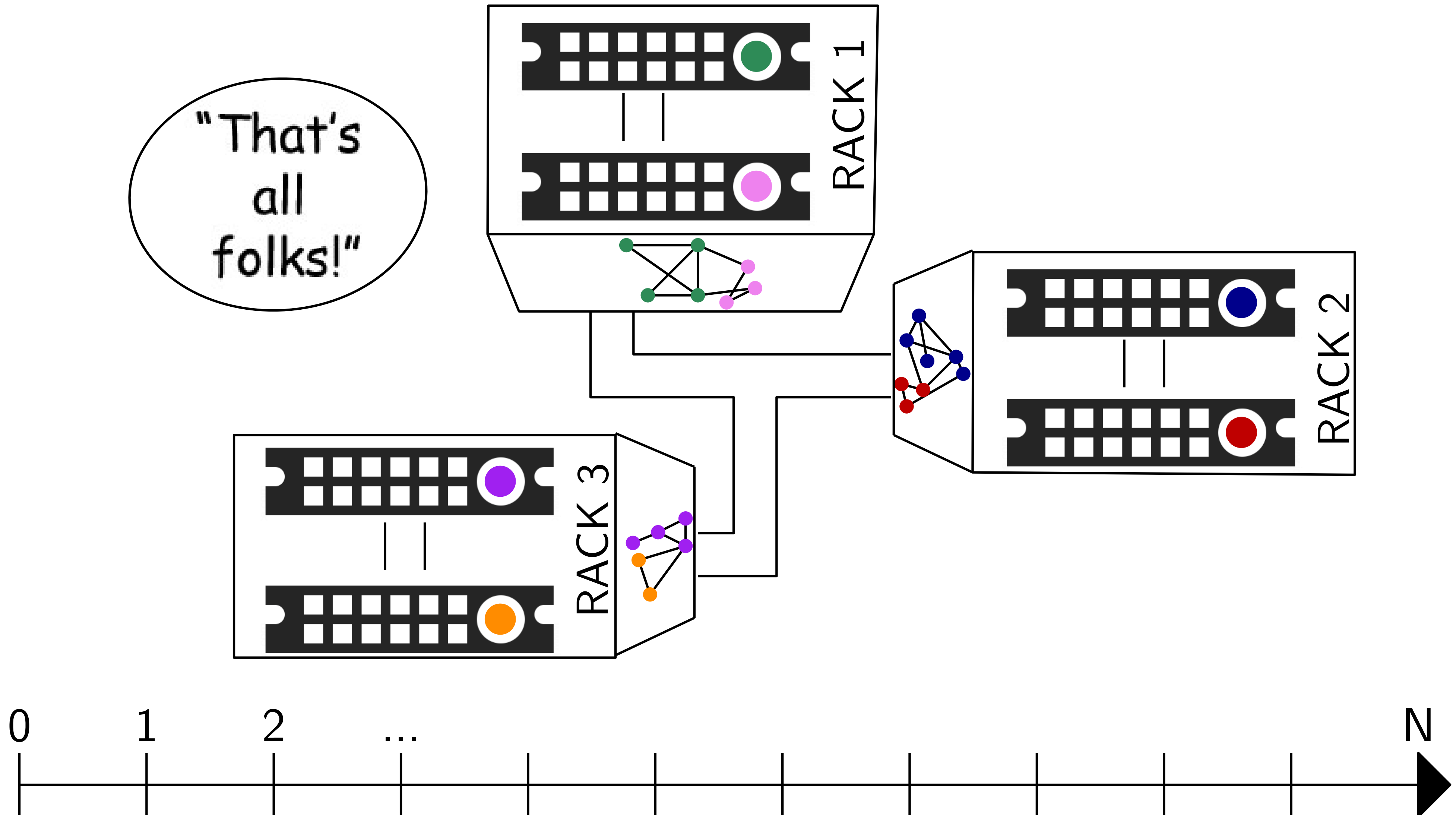
```
function compute(Vertex vertex, Message[] messages)
    int max = 0
    foreach msg in messages
        max = Math.max(max, msg)
    if vertex.value < max
        vertex.value = max
        foreach edge in vertex.outEdges
            sendMessage(edge, new Message(vertex.value))
    else
            vertex.voteToHalt()
```

The TLAV Processing Model

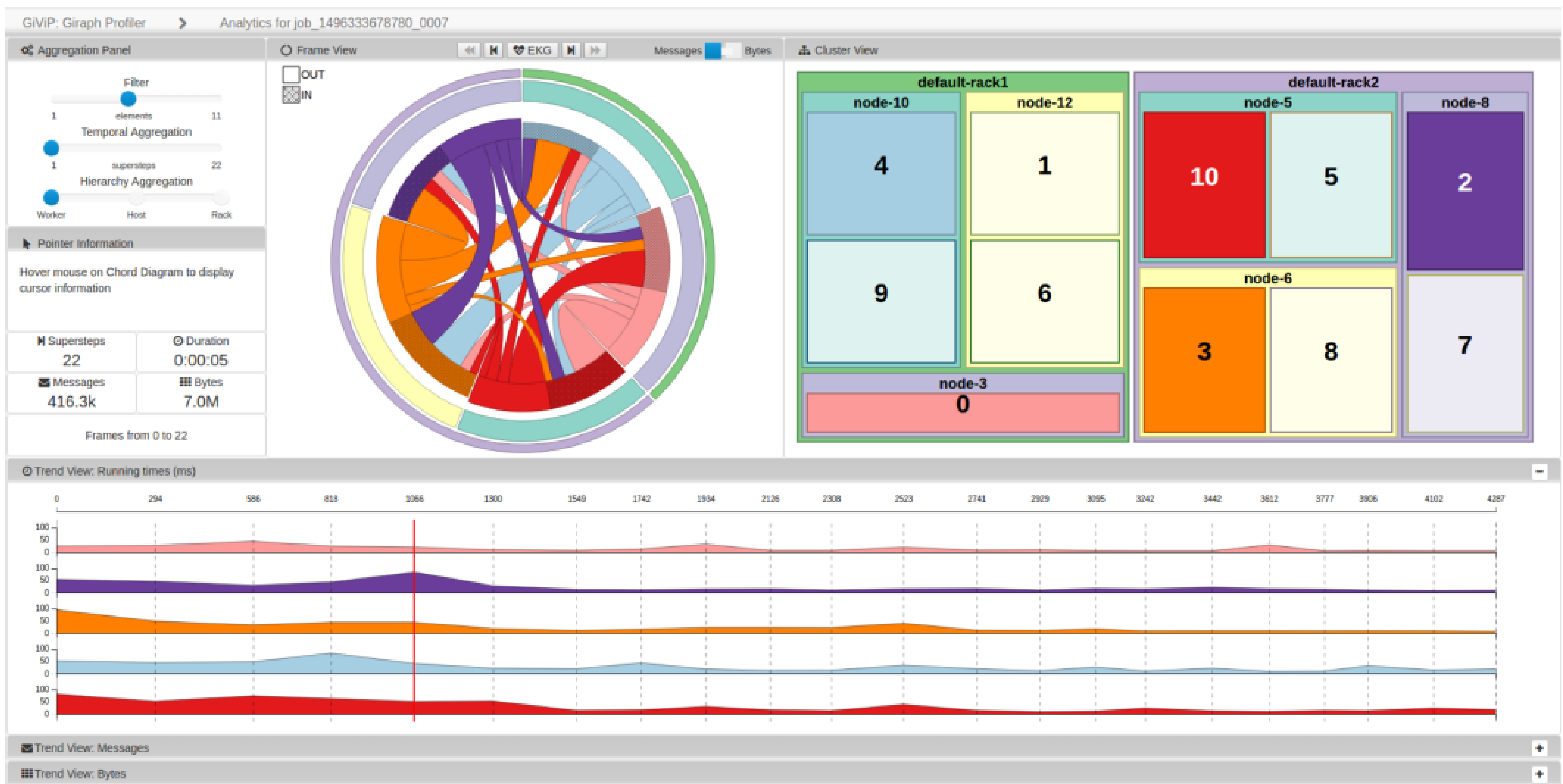
At the end of each superstep, the sent messages are delivered to be used in the next superstep

The TLAV Processing Model

A computation terminates when all vertices voted to halt and there are no more messages to be delivered



GiViP: Giraph Visual Profiler



Profiling TLAV Computations: Tasks

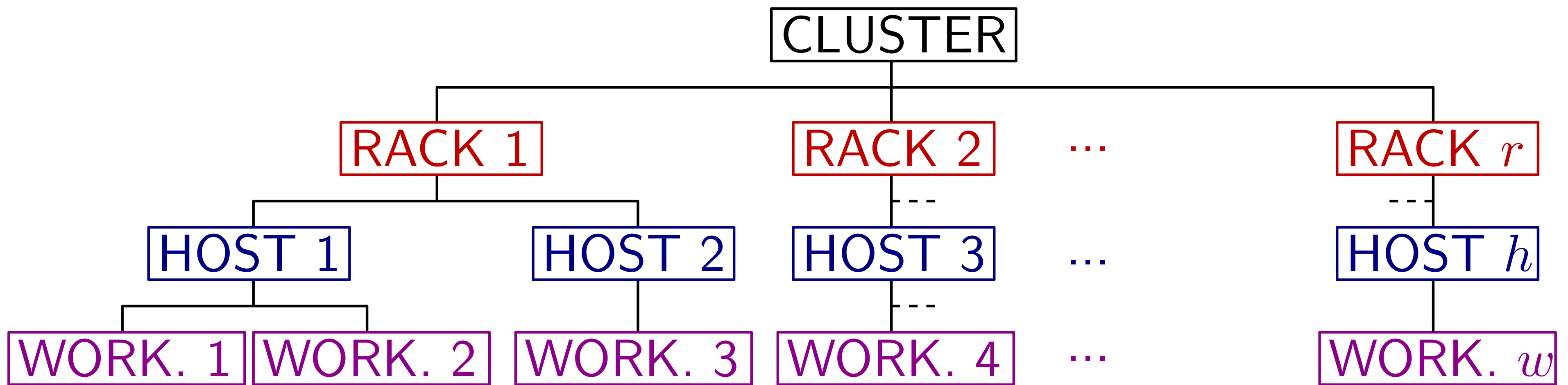
T1 Analyze the performance trend of a computation in terms of running time and traffic load (to evaluate the scalability of a distributed algorithm and to detect possible bottlenecks)

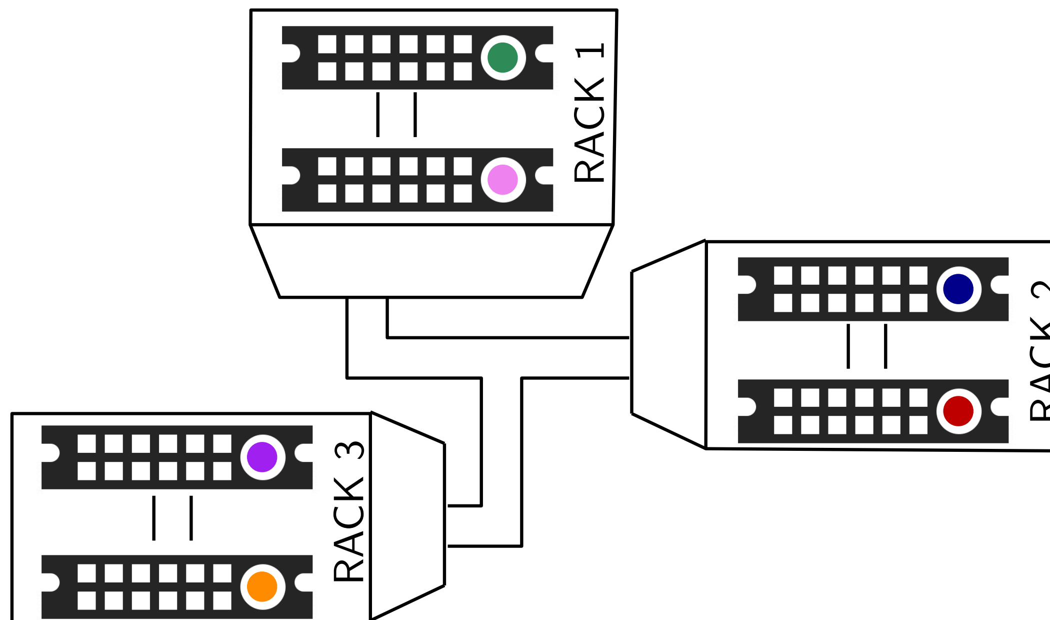
T2 Analyze the traffic between pairs of computing units (workers, hosts, racks) (to detect overloaded links at different levels of the cluster)

T3 Analyze data aggregated at different computing scale and time scale (to handle very large clusters or very long computations)

GiViP: Data Model

We use an inclusion tree T to represent the cluster hierarchy



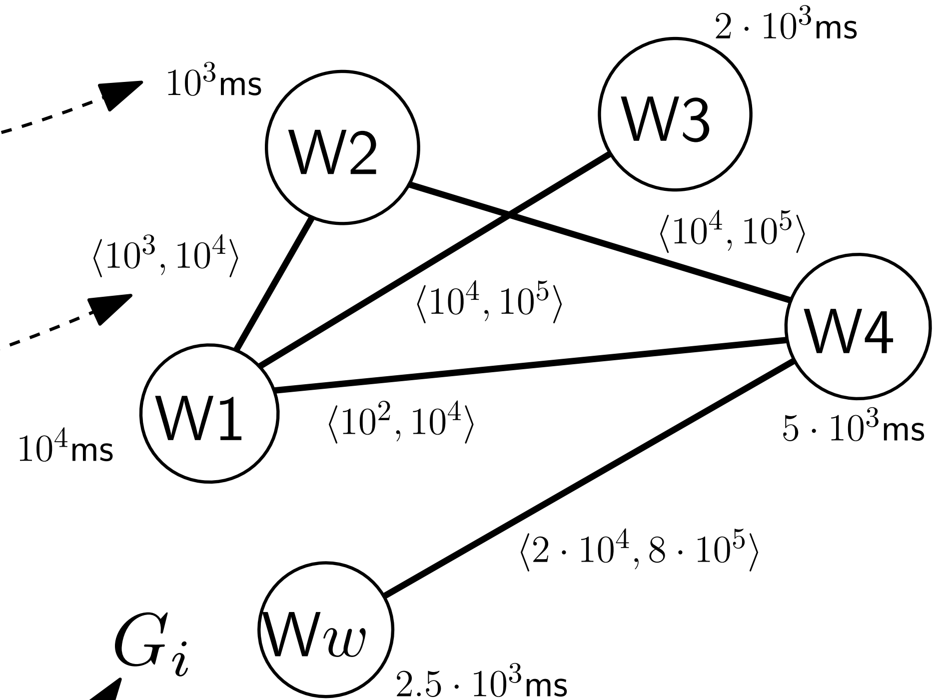


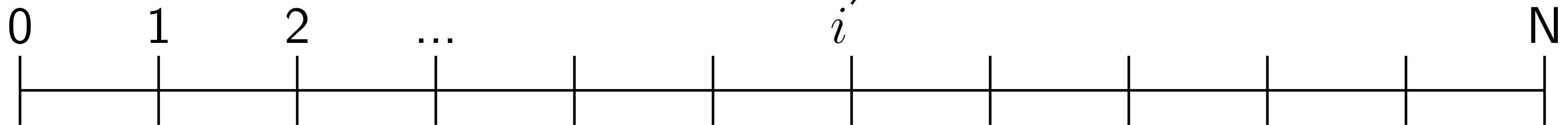
GiViP: Data Model

For each superstep i, the data exchanged by the workers are modeled as a weighted digraph $G_i = (V_i, E_i)$

Each worker has a weight equal to its execution time

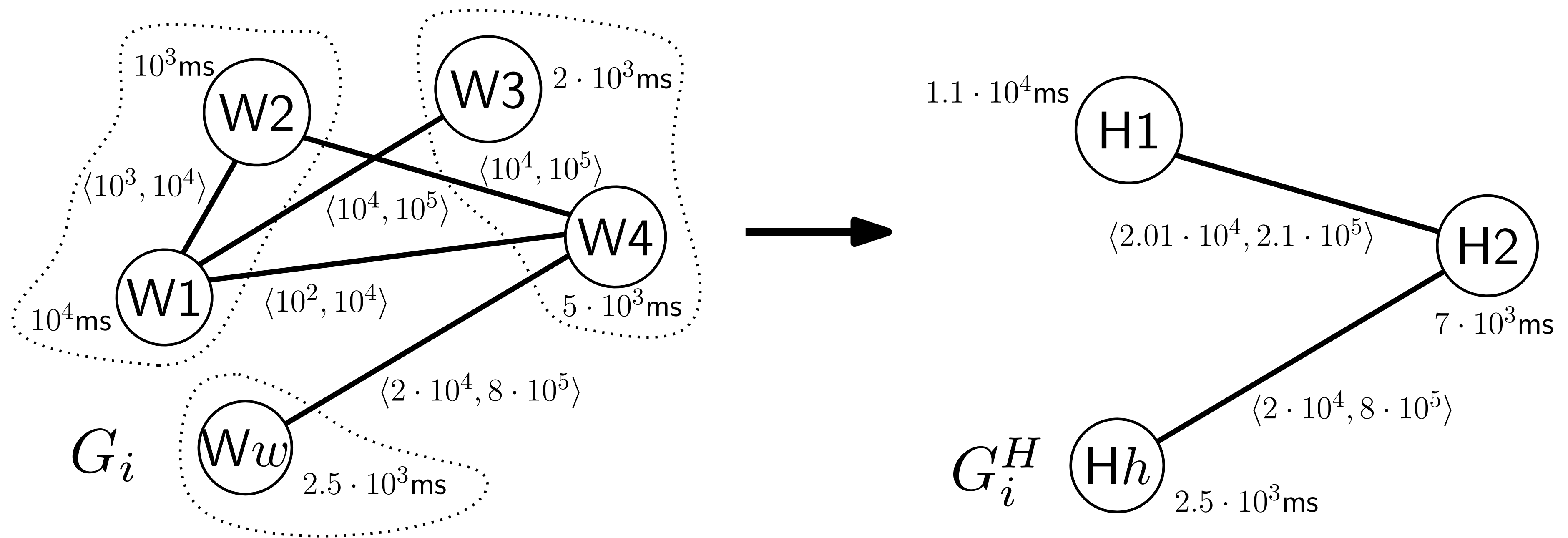
Each link has two weights equal to the # of messages and bytes exchanged





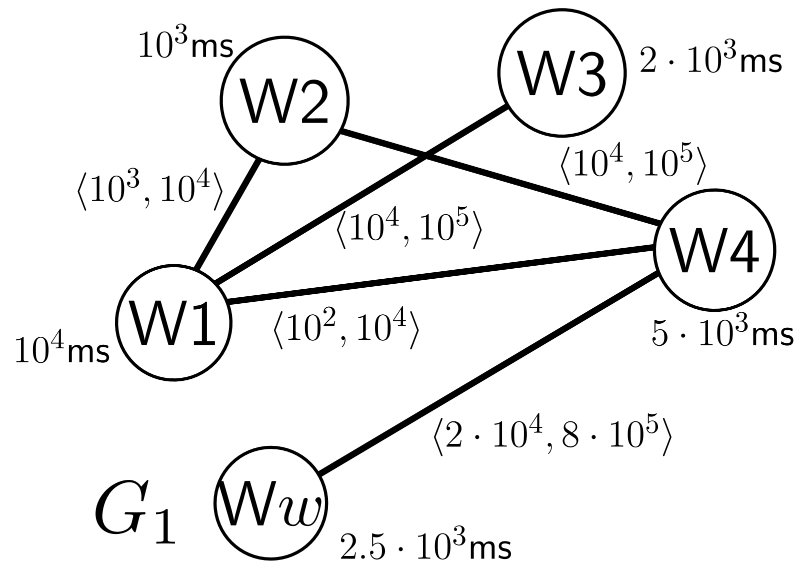
GiViP: Data Aggregation

Hierarchy aggregation: merge workers based on their membership to the same host or rack

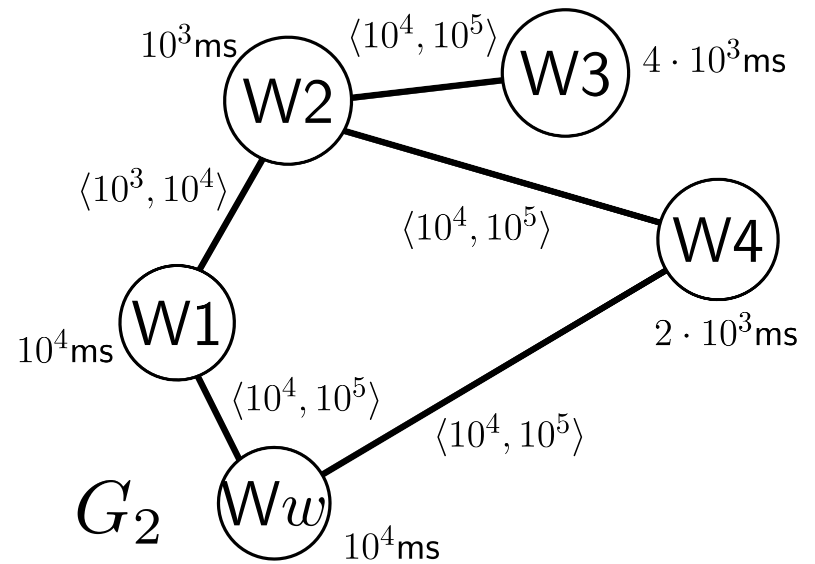


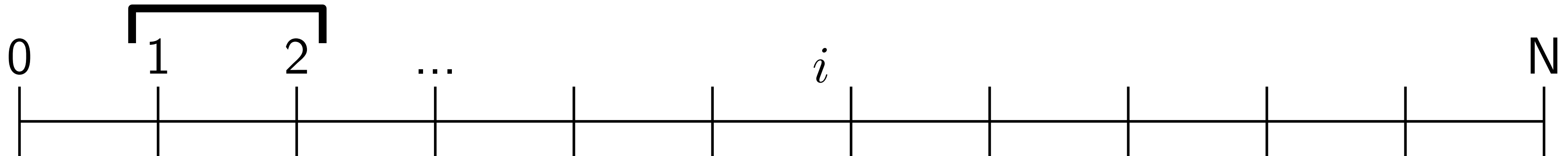
GiViP: Data Aggregation

Temporal aggregation: grouping consecutive supersteps in a single frame



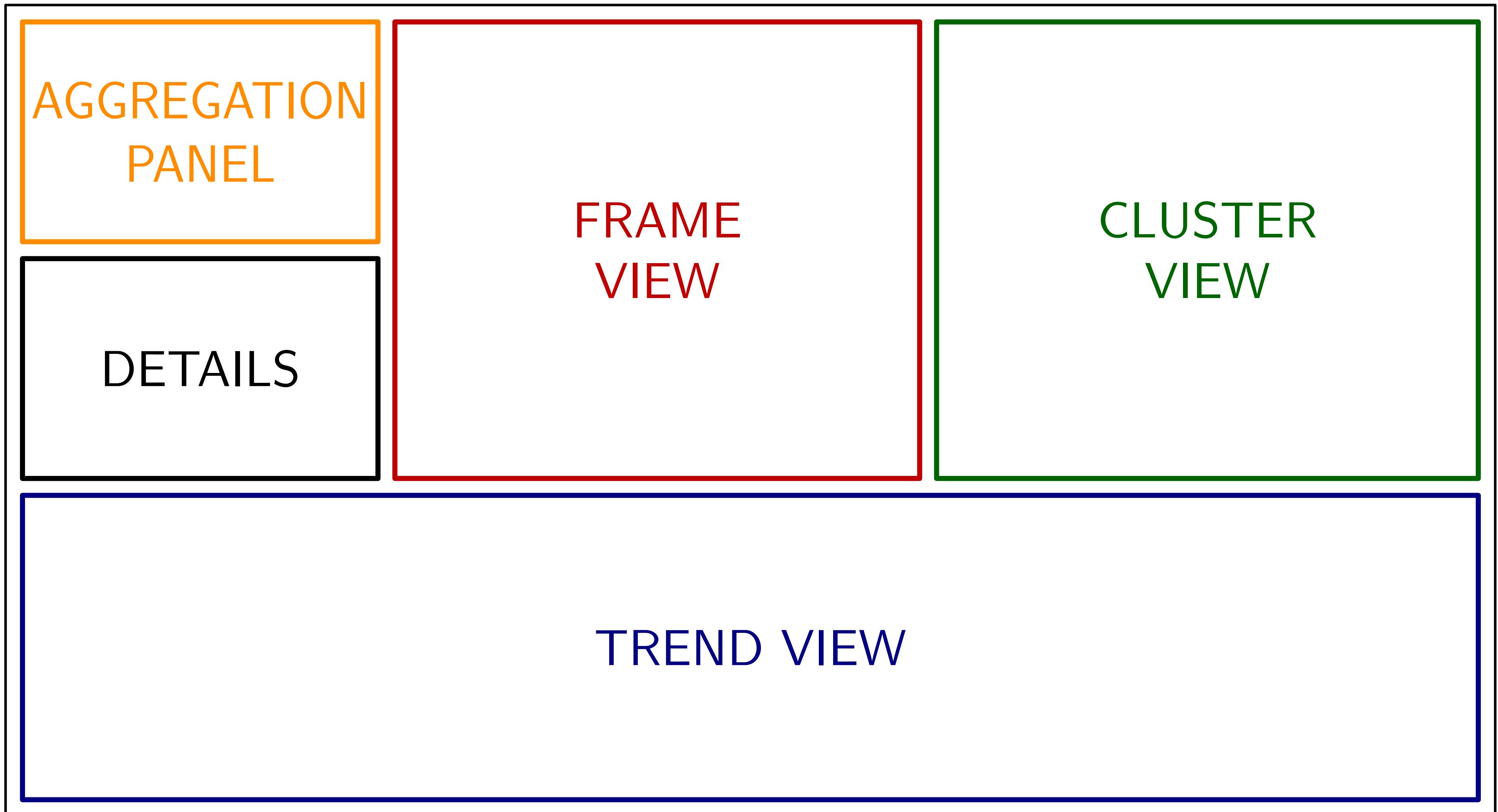
+





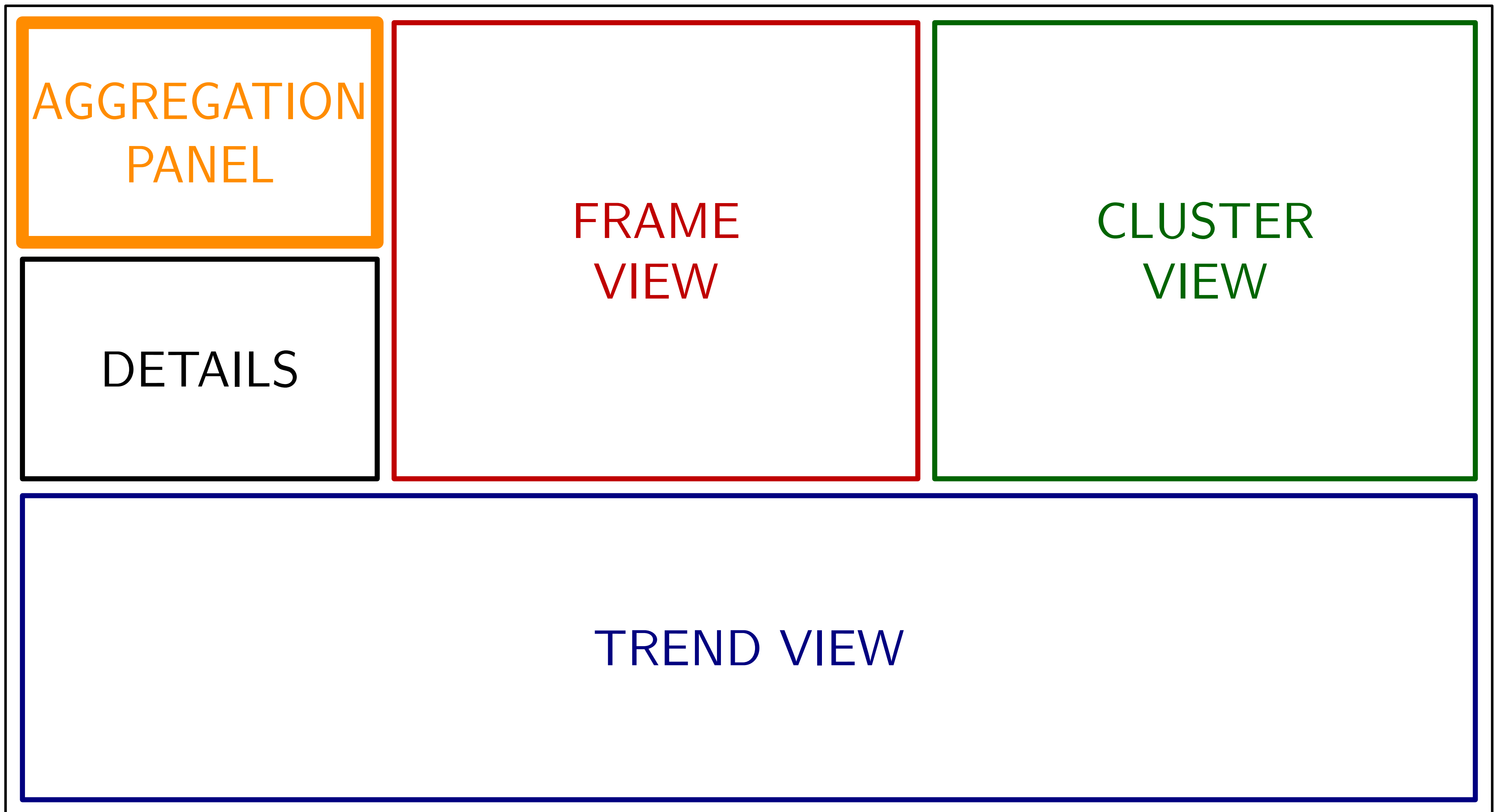
Interface

The interface is divided into three complementary and coordinated views.



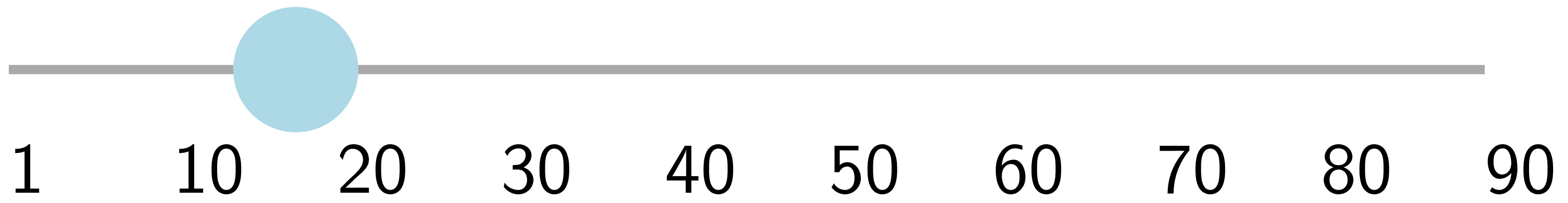
Interface

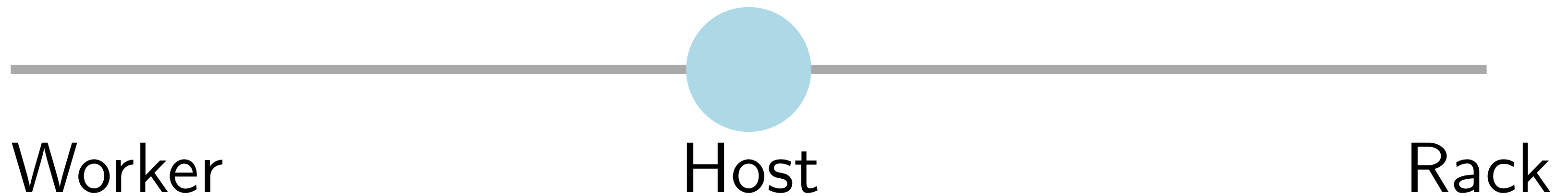
The interface is divided into three complementary and coordinated views.



Aggregation Panel

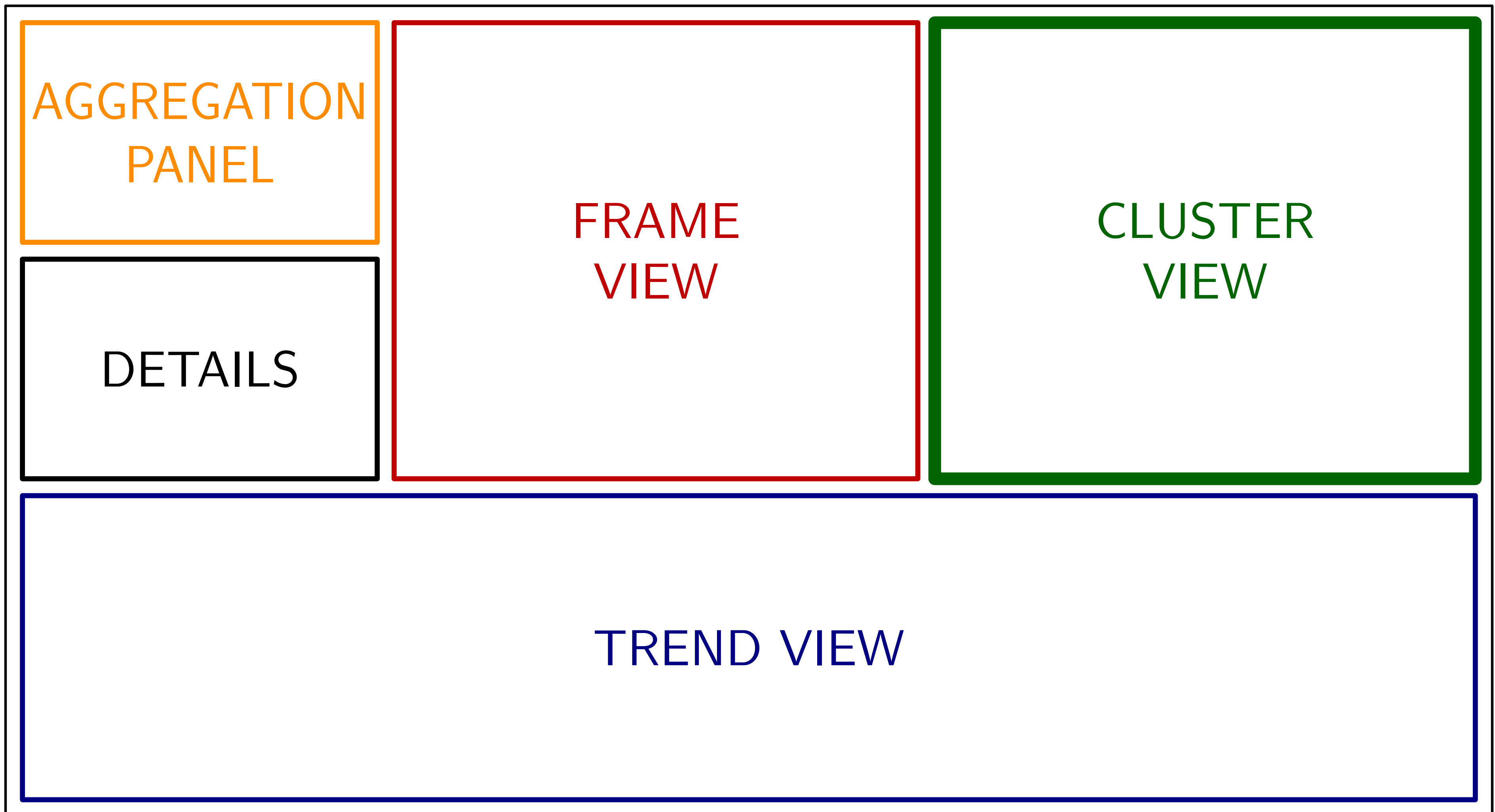
It contains two sliders to aggregate the data:





Interface

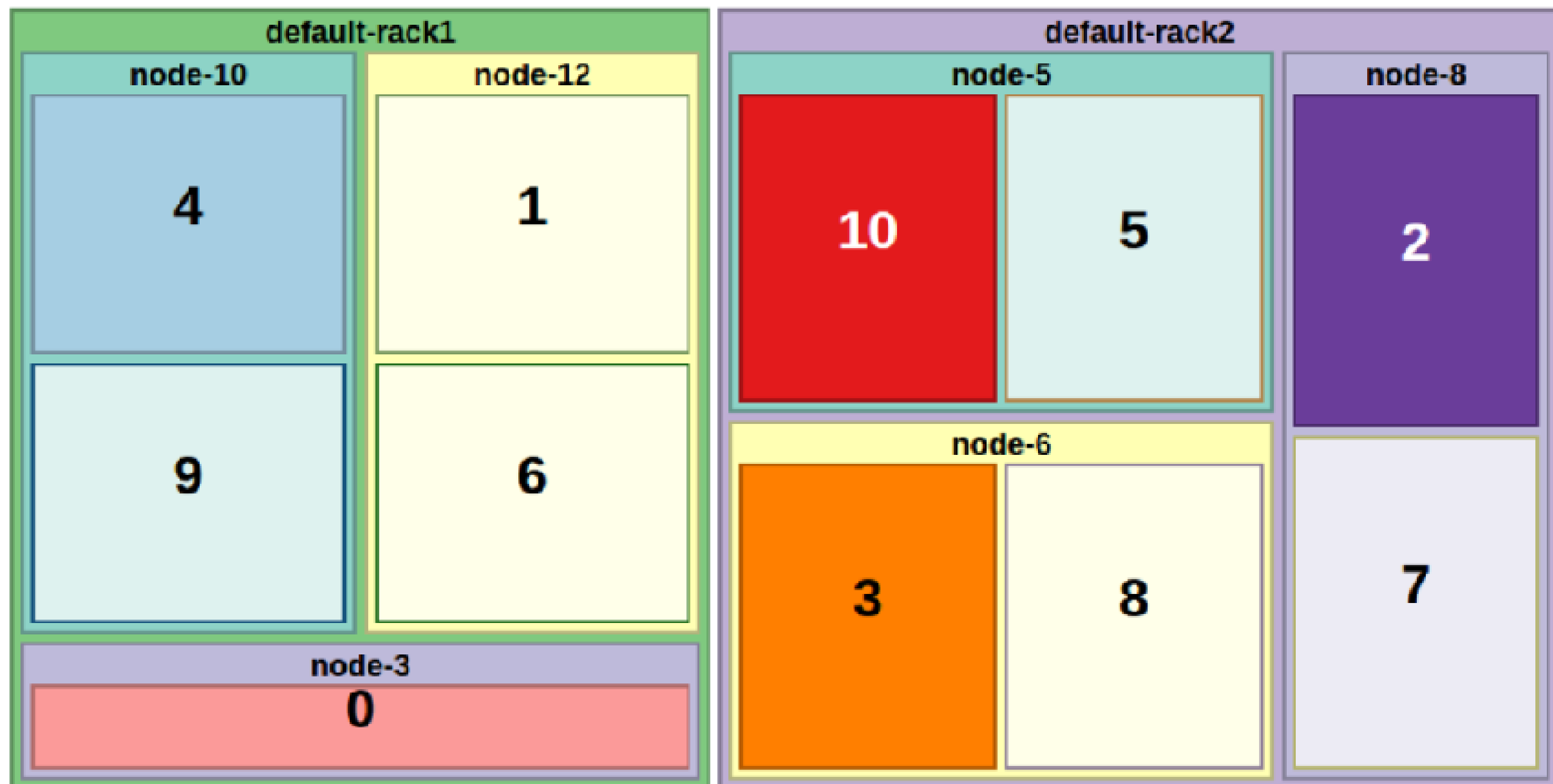
The interface is divided into three complementary and coordinated views.



Cluster View

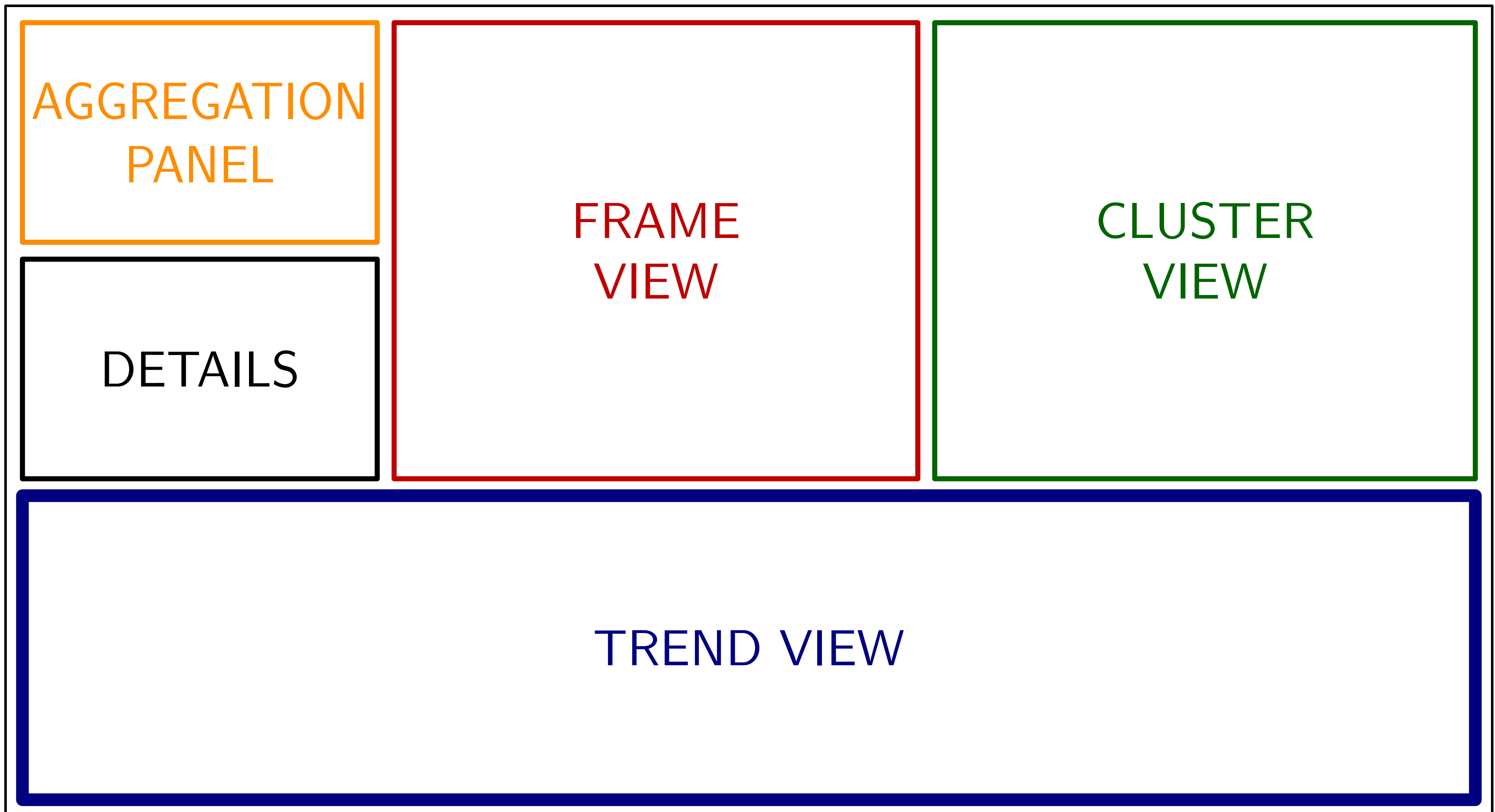
It contains a treemap visualization of the cluster hierarchy:

- the size of a tile is proportional to the # of vertices assigned to that unit (by the partitioner)
- by clicking on a tile the corresponding unit is filtered in/out (based on its state)



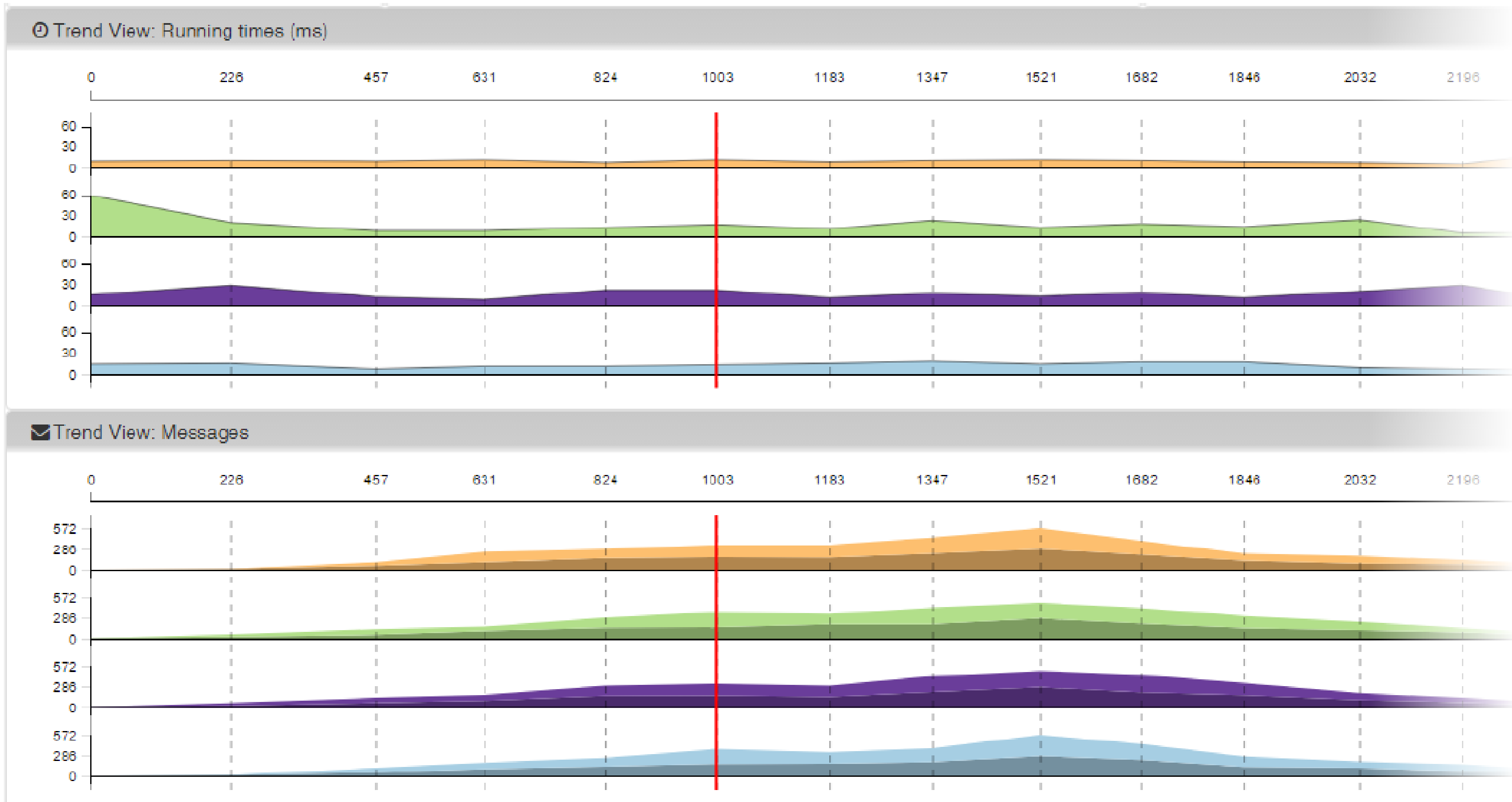
Interface

The interface is divided into three complementary and coordinated views.



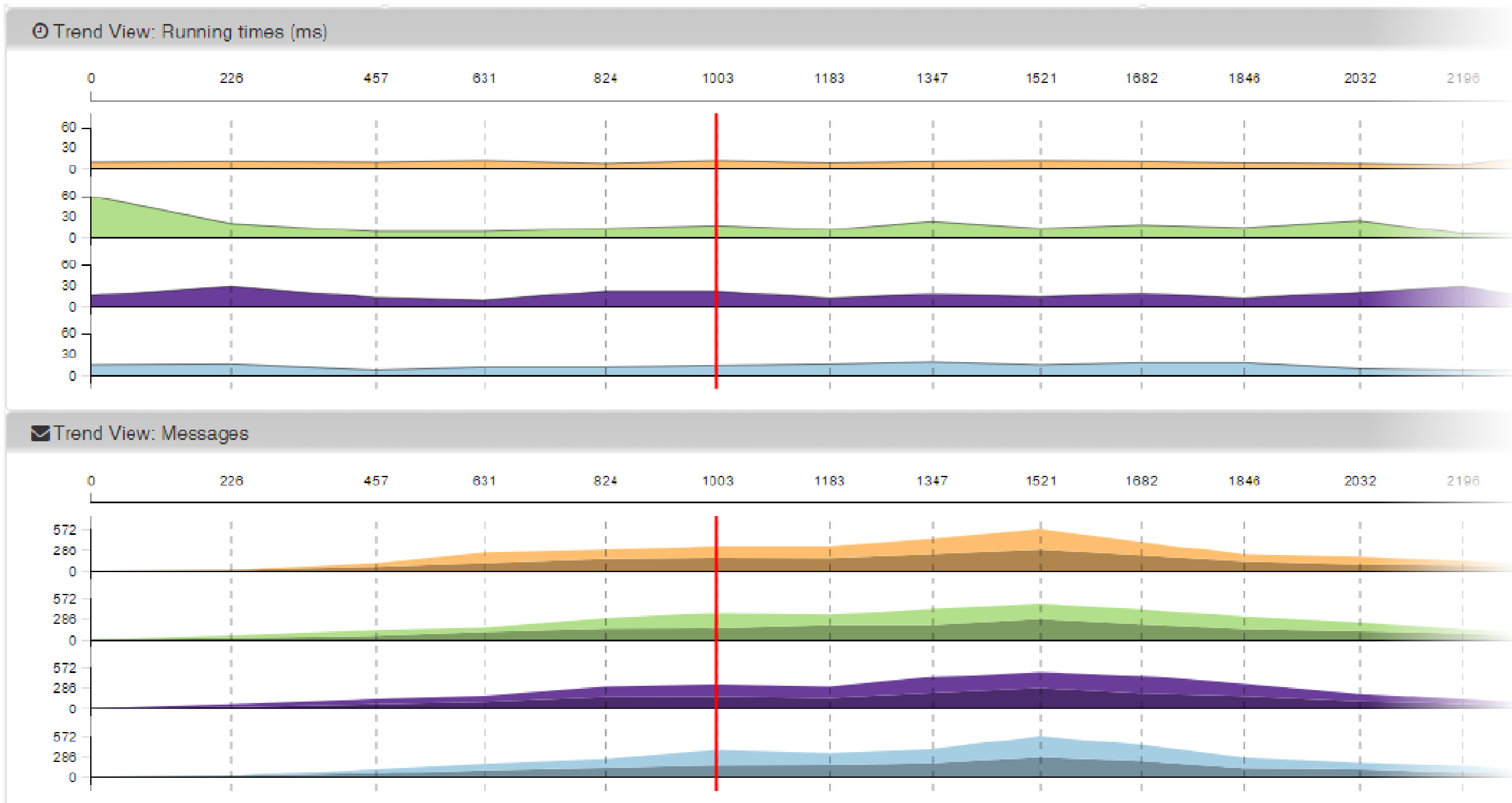
Trend View

For each computing unit (worker/host/rack), it shows the evolution throughout the computation of running time, # of exchanged messages, and # of exchanged bytes



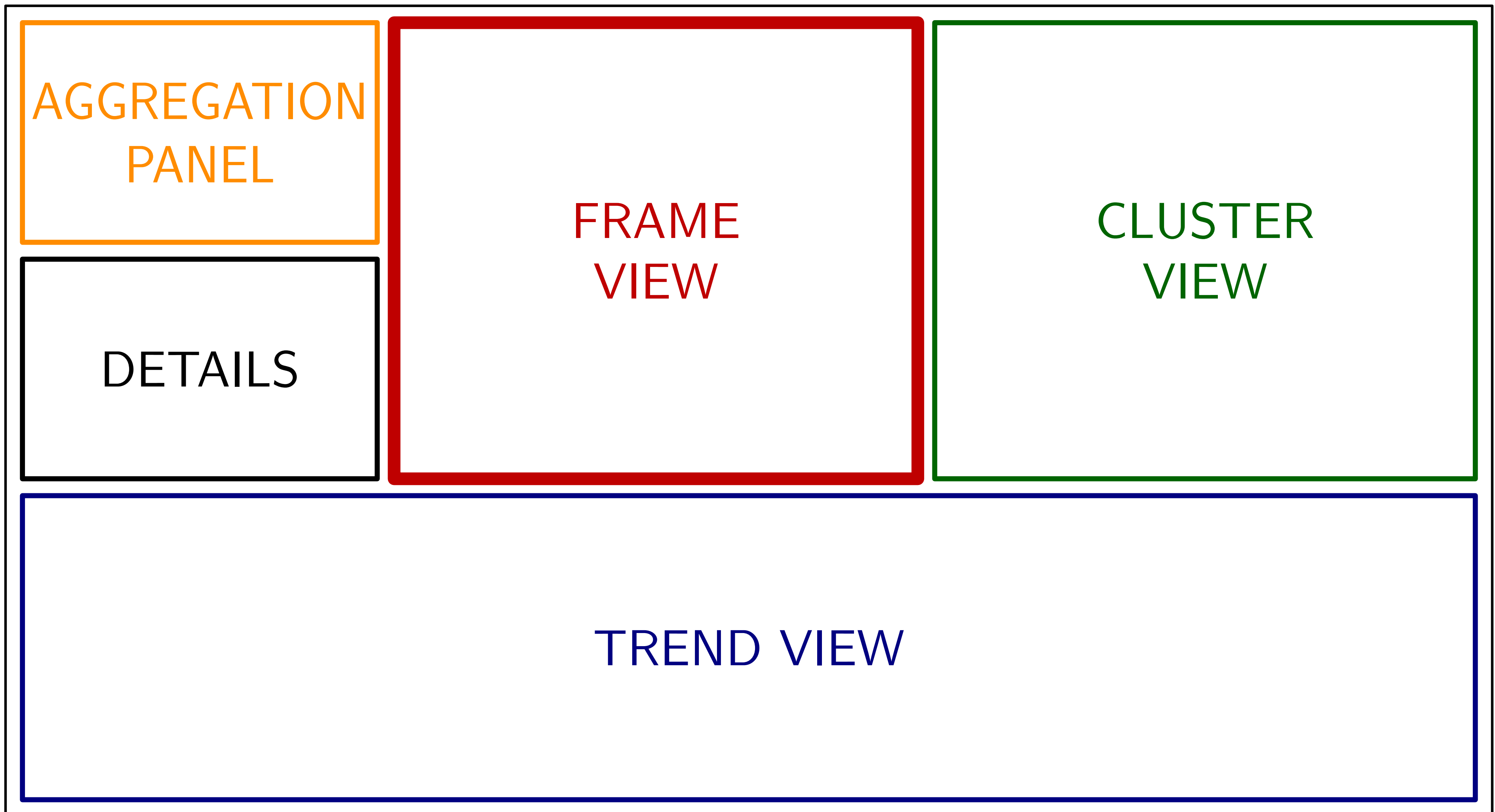
Trend View

It consists of 3 small multiples, vertically stacked and with a shared time axis (split-space visualizations are particularly robust against various concurrent time series for tasks that need large visual spans)



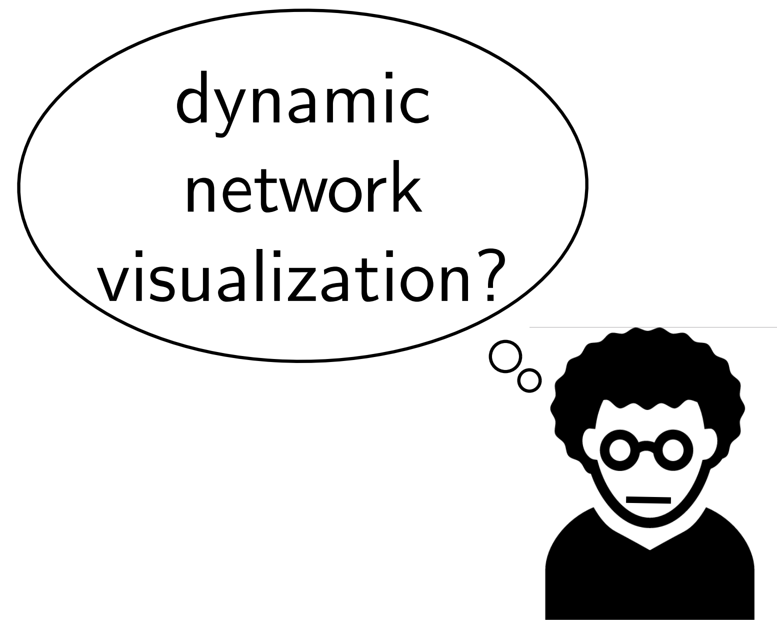
Interface

The interface is divided into three complementary and coordinated views.



Frame View

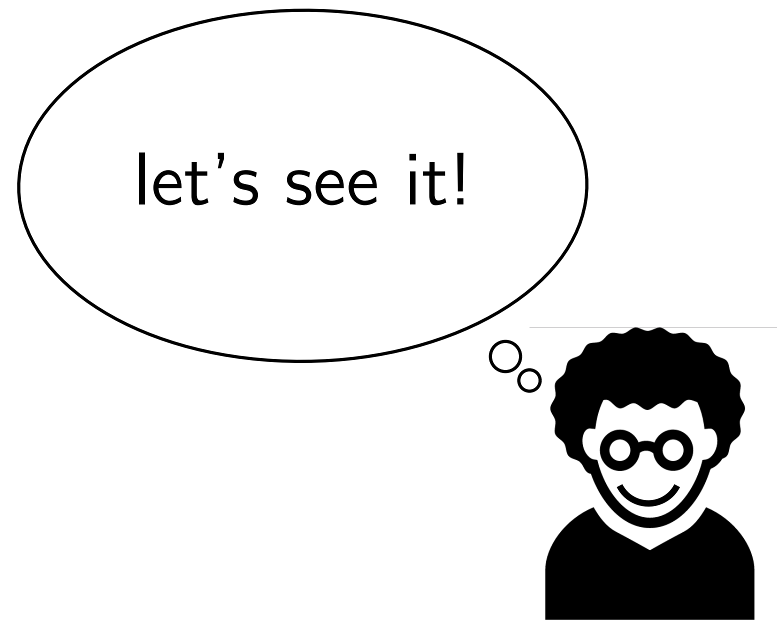
Here we depict the traffic load between pairs of computing units thoughout the whole computation...



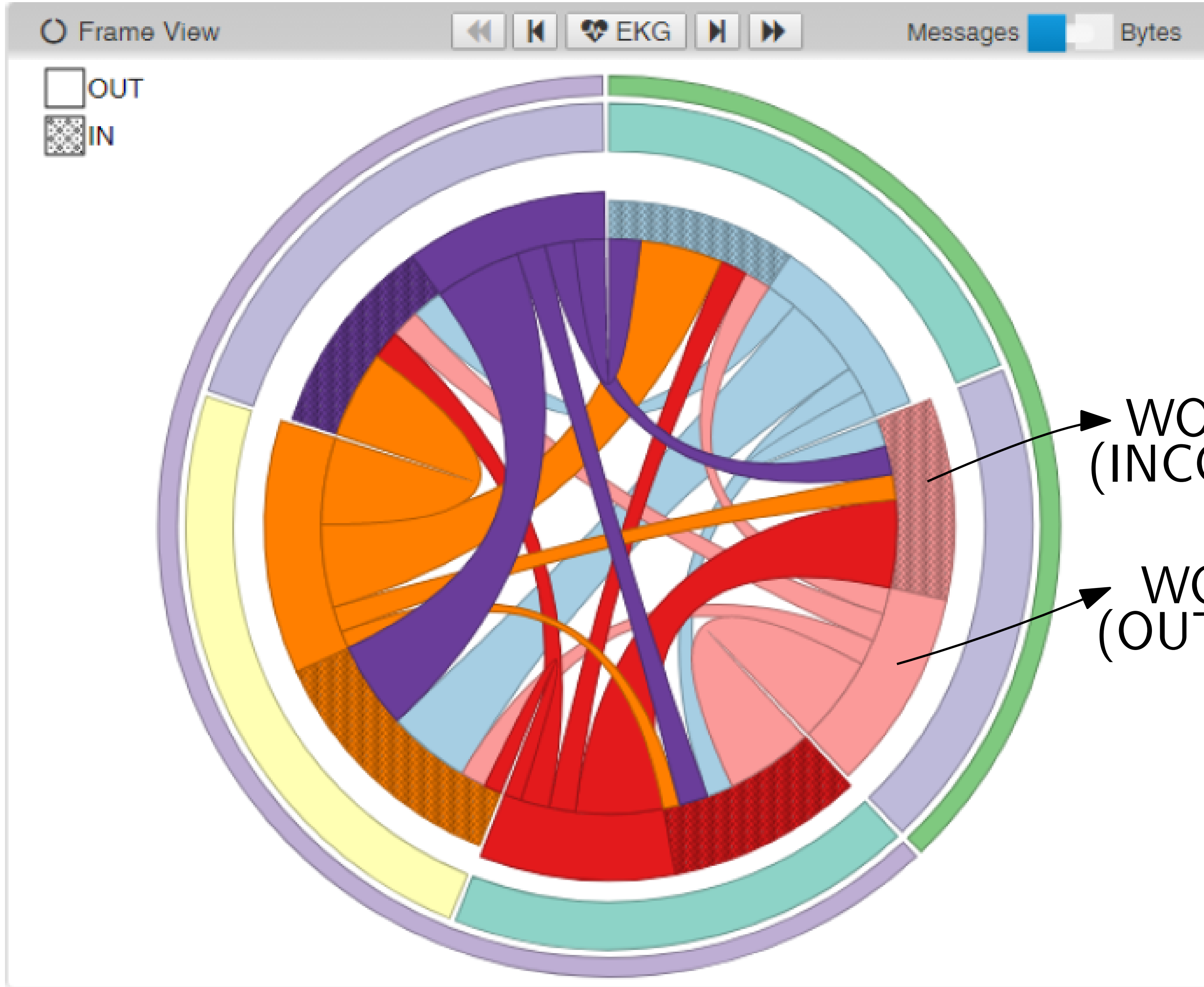
Frame View

Here we depict the traffic load between pairs of computing units thoughout the whole computation...

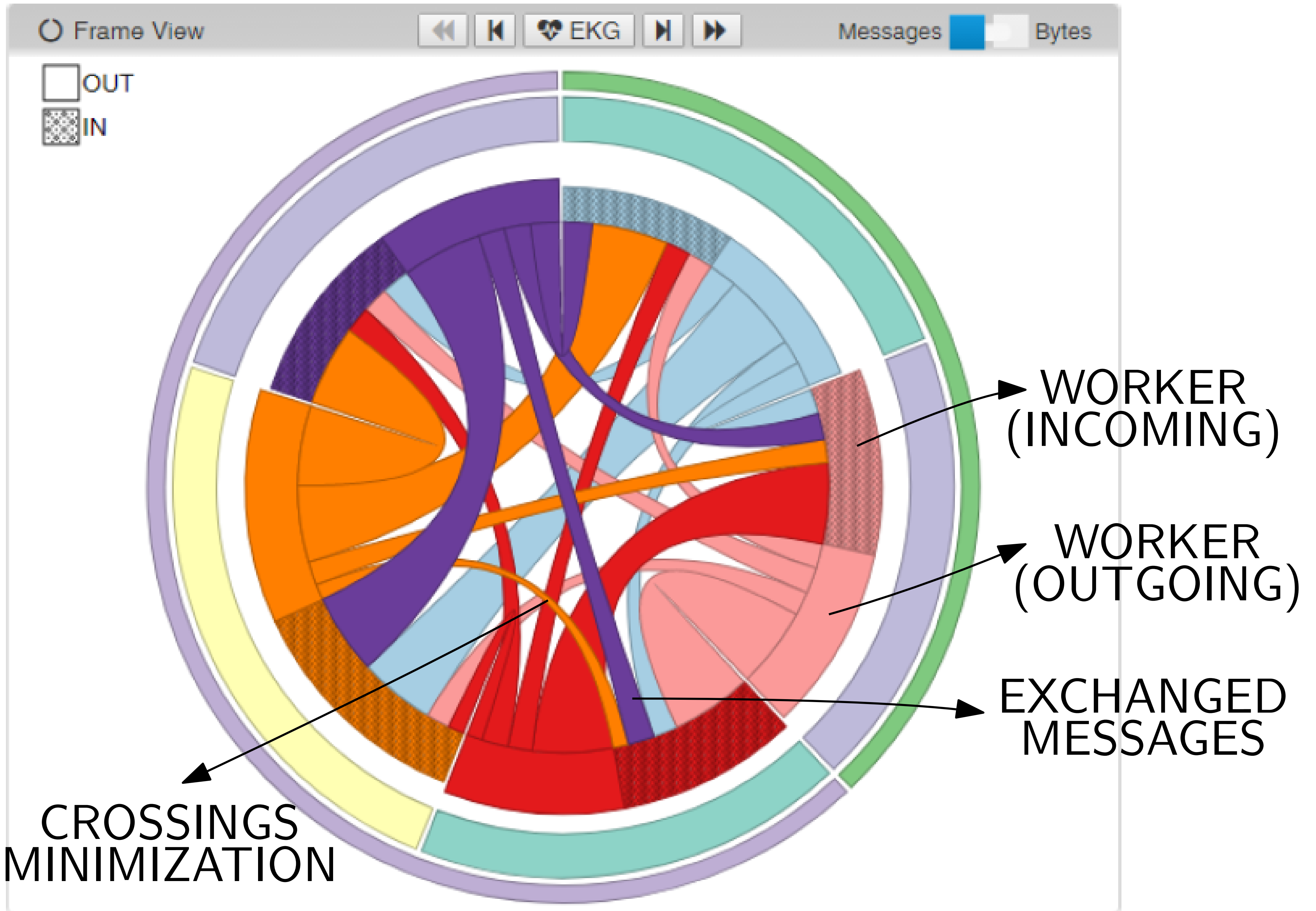
The edge weights change, not the edges...
We implemented an enhanced version of the chord diagram.



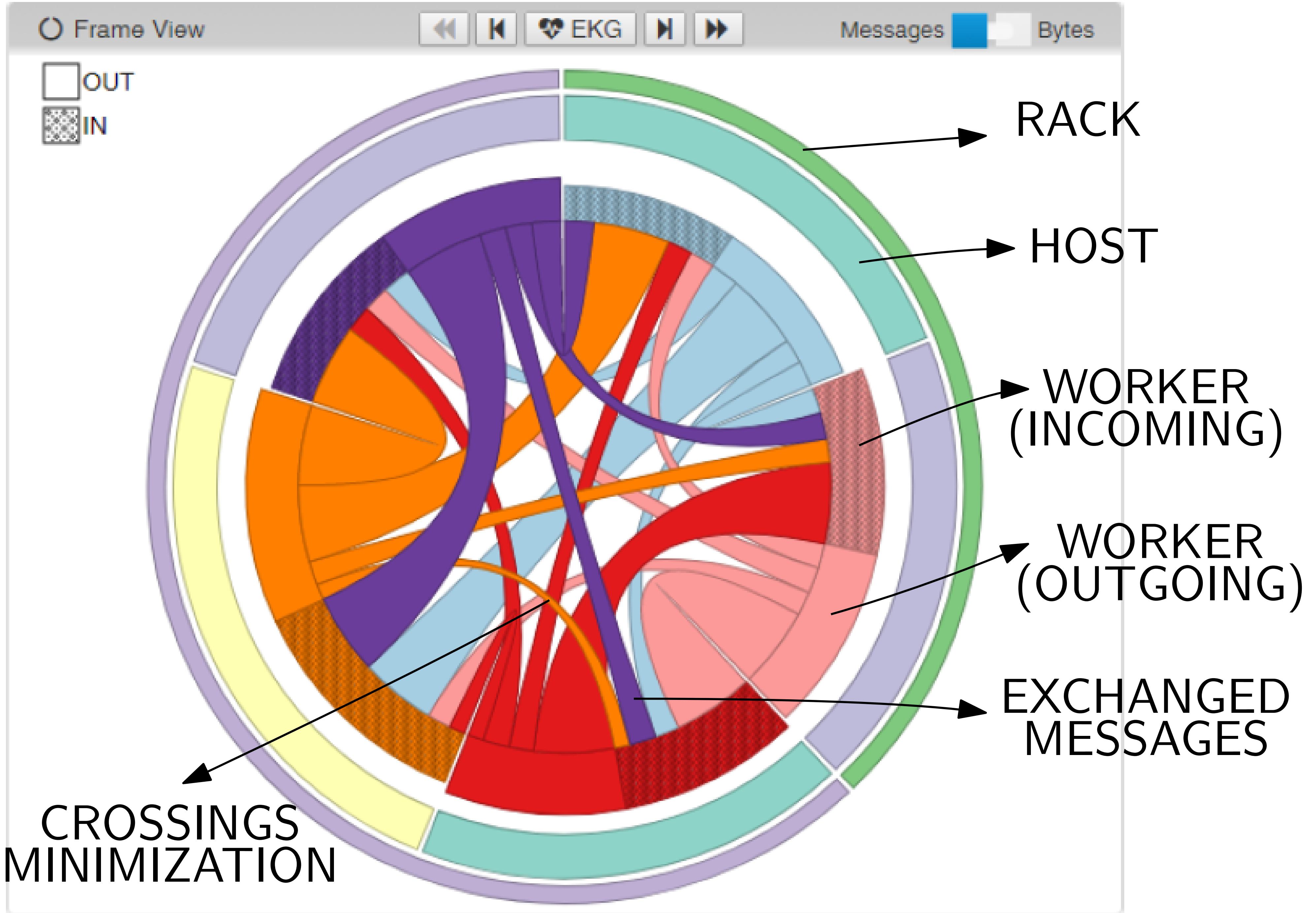
Frame View



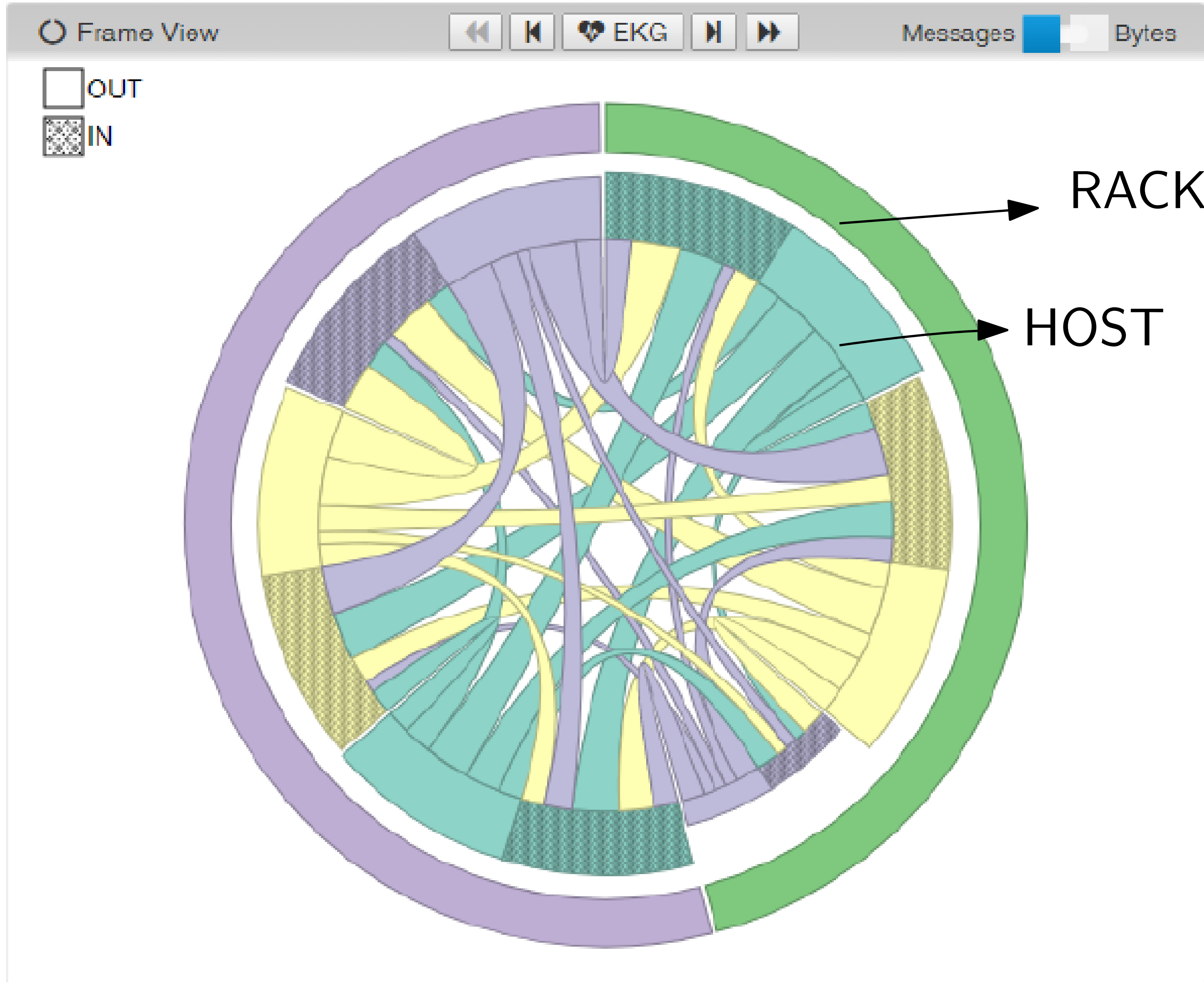
Frame View



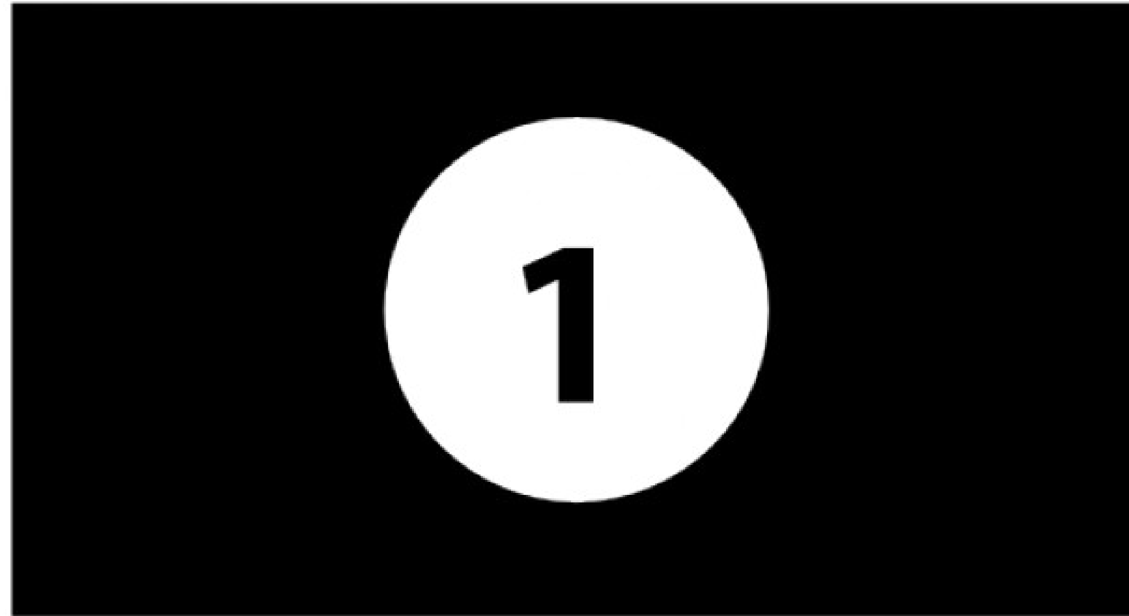
Frame View

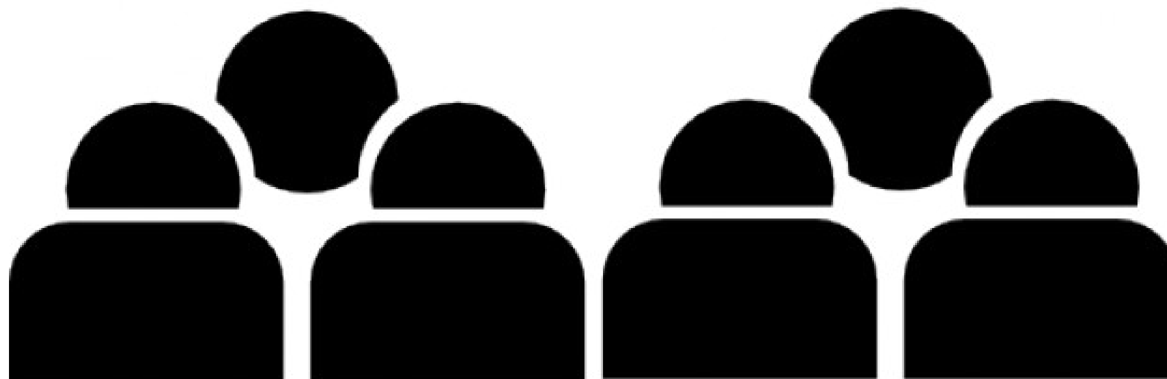


Frame View



Interaction

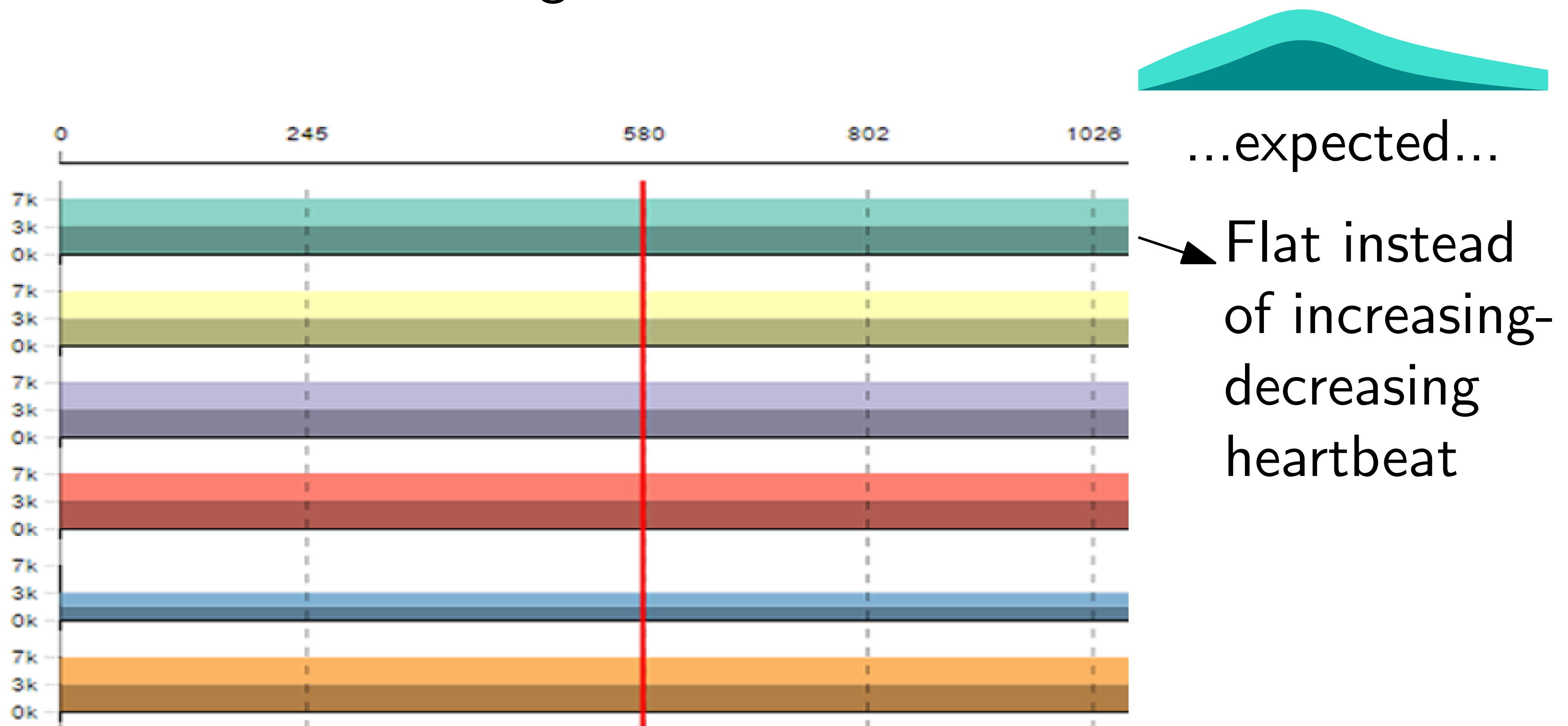




Usage Scenario: Anomalous Message Pattern

We injected a bug in the SSSP algorithm: We added a piece of code that delivers messages to the neighbors of a vertex even if its best-known distance does not change...

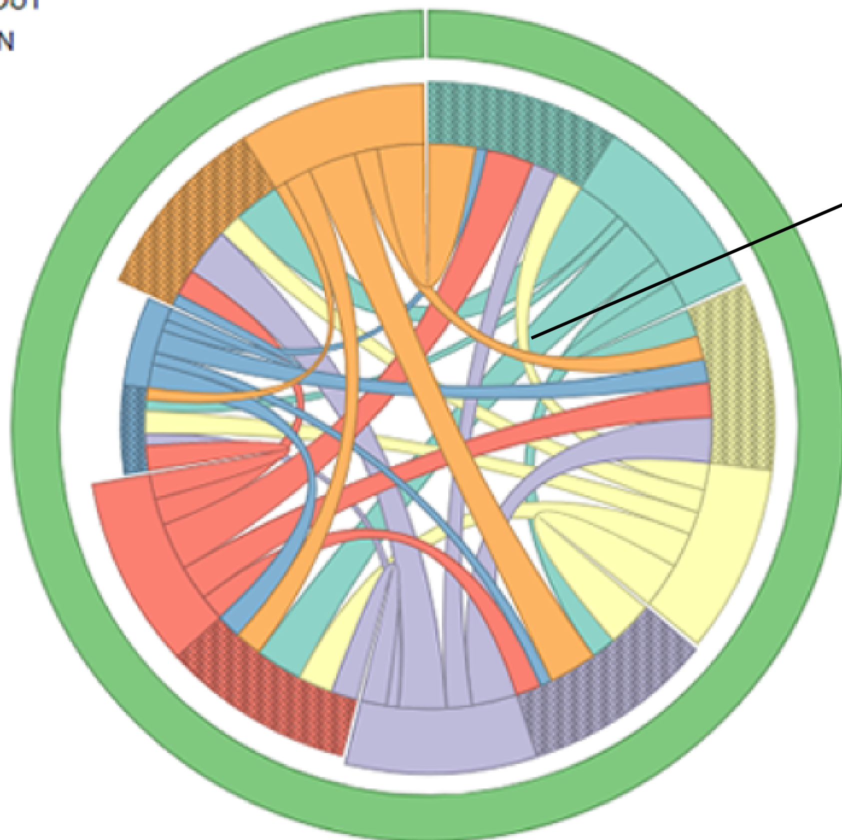
...this causes unnecessary messages, but does not affect the correctness of the algorithm!



Usage Scenario: Anomalous Message Pattern

We injected a bug in the SSSP algorithm: We added a piece of code that delivers messages to the neighbors of a vertex even if its best-known distance does not change...

...this causes unnecessary messages, but does not affect the correctness of the algorithm!



Messages evenly distributed among the workers

Future Research

Main limit: the Frame View requires the usage of filters and/or aggregations if more than a few tens of vertices need to be displayed (the chord diagram suffers from edge clutter) $\rightarrow$ Matrix-based representations?

Temporal queries (timebox widgets)

Detection of HW failures

Future Research

Main limit: the Frame View requires the usage of filters and/or aggregations if more than a few tens of vertices need to be displayed (the chord diagram suffers from edge clutter) $\rightarrow$ Matrix-based representations?

Temporal queries (timebox widgets)

Detection of HW failures