Free University of Bolzano

Candidate: Riccardo Busetti

Supervisor: Claus Pahl

Distributed Particle Swarm Optimization: Design of Synchronous and Asynchronous Algorithms for Optimization Problems at the Edge

Overview

SLIDE 3

Introduction

SLIDE 14

Methodology

SLIDE 22

Solution

SLIDE 27

Evaluation

SLIDE 32

Conclusions

Introduction

Context

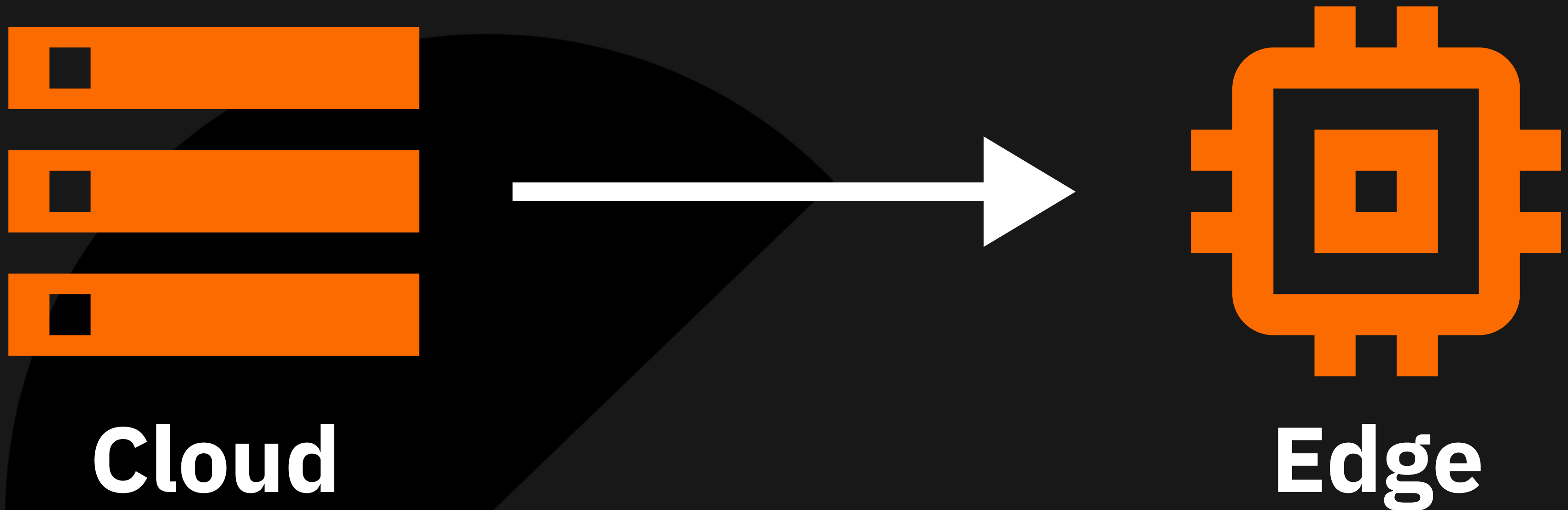
The **demand** for cloud resources is quickly increasing and this creates the need to **alleviate the work** in the cloud.

Context

A new paradigm called edge computing has emerged, with the goal of reducing the strain on the cloud infrastructure.

What is Edge Computing?

Edge computing focuses on moving the computations away from centralized servers to the edge of a network.



Edge computing open problems

Optimizing offloading strategies

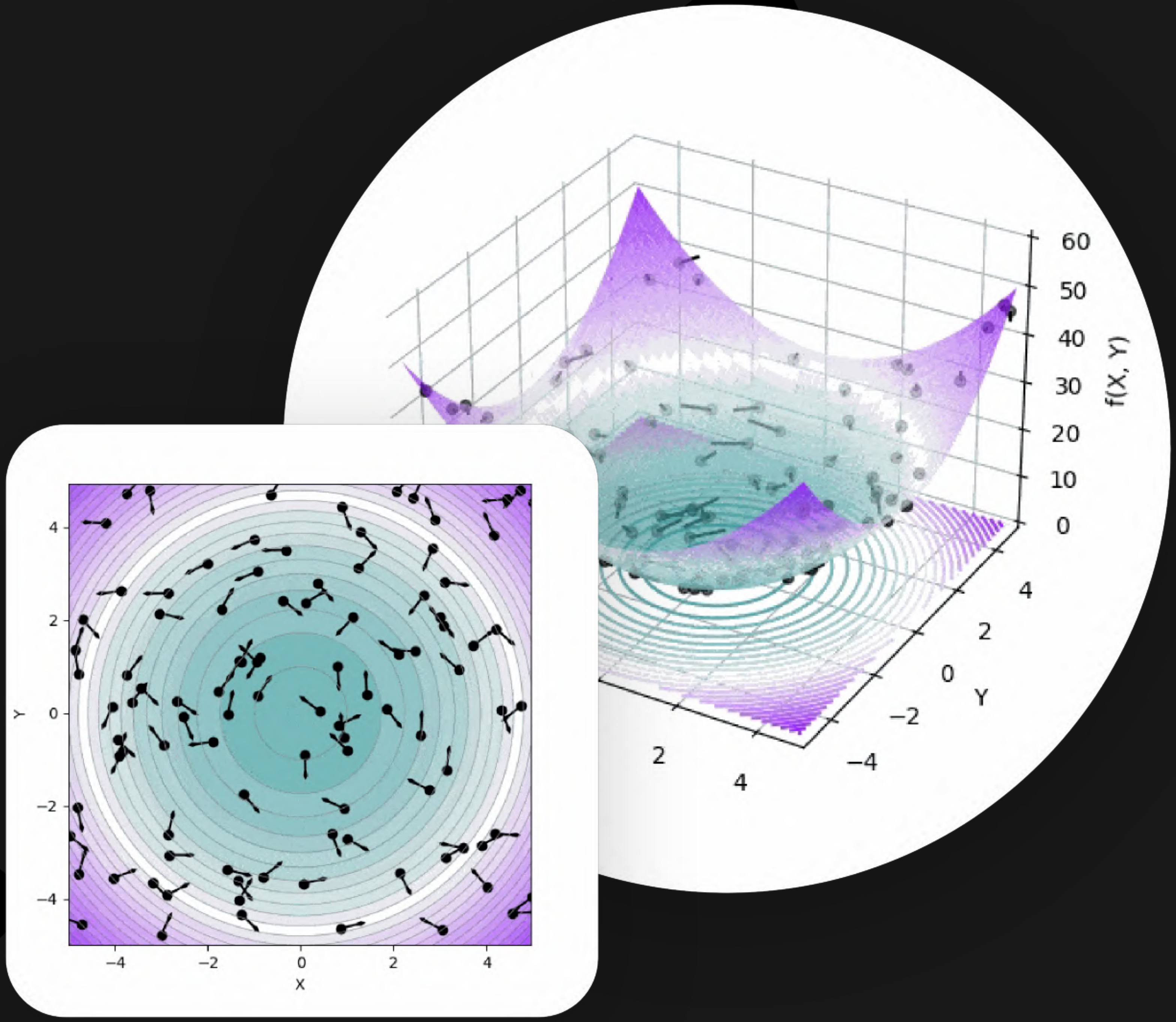
Workload placement

Performance management

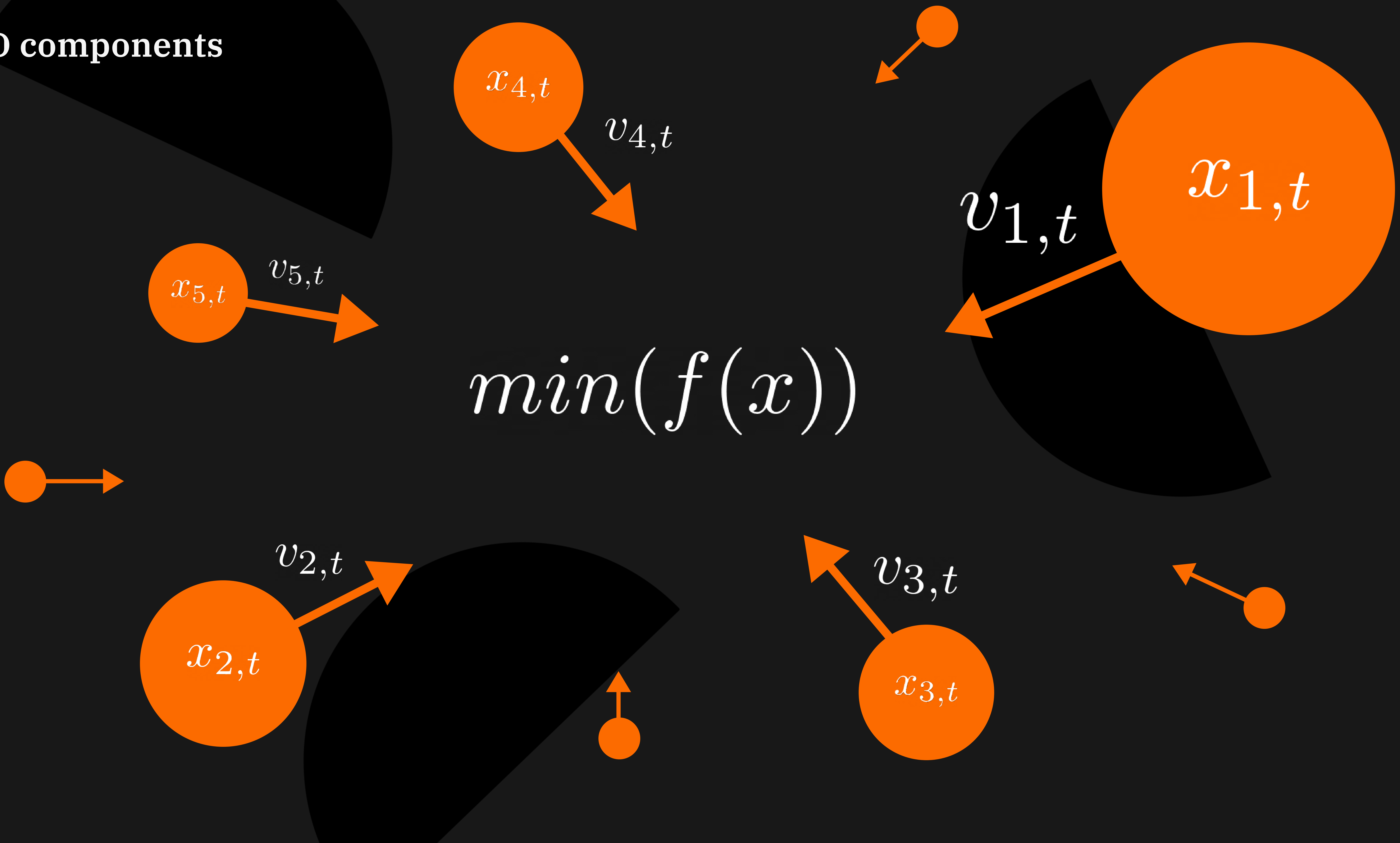
Load balancing

What is Particle Swarm Optimization?

PSO is a nature-inspired stochastic optimization algorithm motivated by the intelligent collective behavior of animals.



PSO components



Is it feasible to run PSO at the edge?

PSO at the edge

- Lack of resources
- High chance of faults

- Alleviate the cloud
- Reduce latency

The idea

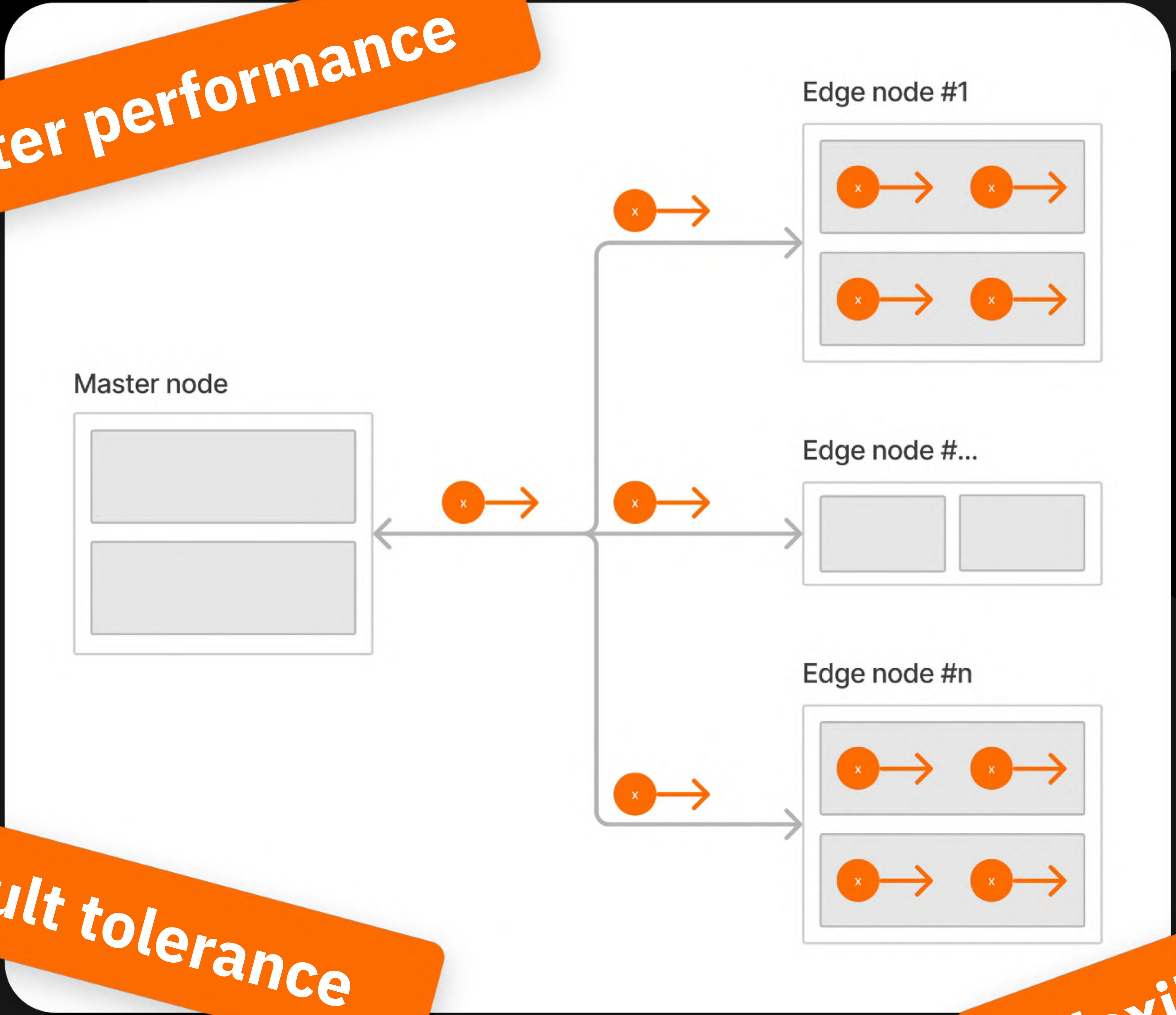
Build a distributed PSO algorithm that can be executed on a cluster of edge nodes.

Better performance

More fault tolerance

Flexible scalability

The idea



Better performance

More fault tolerance

Flexible scalability

Methodology

Existing research

- Shared-memory
- Not fault tolerant
- Mostly synchronous

6th World Congresses of Structural and Multidisciplinary Optimization
Rio de Janeiro, 30 May - 03 June 2005, Brazil

A Parallel Particle Swarm Optimization Algorithm Accelerated by Asynchronous Evaluations

Gerhard Venter (gventer@vrand.com),
Jaroslaw Sobieszczanski-Sobieski (j.sobieski@larc.nasa.gov)

Vanderplaats Research and Development, Colorado Springs, USA
NASA Langley Research Center, Hampton, USA

1. Abstract
A parallel Particle Swarm Optimization (PSO) algorithm is presented. Particle swarm optimization is a fairly recent addition to the family of non-gradient based, probabilistic search algorithms that is based on a simplified social model and is closely tied to swarming theory. Although PSO algorithms present several attractive properties to the designer, they are plagued by high computational cost as measured by elapsed time. One approach to reduce the elapsed time is to make use of coarse-grained parallelization to evaluate the design points. Previous parallel PSO algorithms were mostly implemented in a synchronous manner, where all design points within a design iteration are evaluated before the next iteration is started. This approach leads to poor parallel speedup in cases where a heterogeneous parallel environment is used and/or where the analysis time depends on the design point being analyzed. This paper introduces an asynchronous parallel PSO algorithm that greatly improves the parallel efficiency. The asynchronous algorithm is benchmarked on a cluster assembled of Apple Macintosh G5 desktop computers, using the multi-disciplinary optimization of a typical transport aircraft wing as an example.

2. Keywords: Particle Swarm Optimization, PSO, asynchronous parallel computing

3. Introduction
Particle Swarm Optimization (PSO) is a fairly recent, but rapidly growing, addition to an expanding collection of non-gradient based, probabilistic search algorithms. Some widely used algorithms that fall into this category are genetic algorithms [1] that model Darwin's principle of survival of the fittest and simulated annealing algorithms [2] that model the equilibrium of large numbers of atoms during an annealing process. This class of optimization algorithms provides the designer with several attractive characteristics. For example, these algorithms are generally easy to implement, can efficiently make use of large numbers of parallel processors, do not require continuity in response functions and are better suited for finding global or near global solutions. Although these non-gradient based algorithms provide the designer with several advantages, they should be applied with care. Due to their high computational cost, these algorithms should only be used when a gradient-based algorithm is not a viable alternative, such as integer/discrete and discontinuous problems.

Many non-gradient based search algorithms are based on some natural phenomena, and PSO is no exception. Particle swarm optimization is based on a simplified social model that is closely tied to swarming theory and was first introduced by Kennedy and Eberhart [3, 4]. A physical analogy might be a school of fish that is adapting to its environment. In this analogy, each fish adapts to its environment by making use of its own memory as well as knowledge gained by the school as a whole.

Although PSO is a fairly recent algorithm, it is quickly gaining momentum in the engineering research community. For example Fourie and Groenwold applied the algorithm to structural shape and sizing [5] and topology optimization [6] problems. The authors applied the algorithm to a cantilevered beam [7] and multi-disciplinary design optimization of a transport aircraft wing [8]. The University of Florida has applied the algorithm to biomechanical system identification problems e.g., [9, 10].

Some authors have worked on parallel PSO algorithms in an attempt to alleviate the associated high computational cost. Most parallel implementations of the PSO algorithm presented to date are based on a synchronous implementation e.g., [9, 10] where all design points within a design iteration are evaluated, before the next design iteration is started. This implementation can easily lead to poor parallel performance in many cases. The present work presents a new asynchronous parallel implementation to increase the parallel performance of the PSO algorithm. The challenge is to keep all processors working as the algorithm moves from one design iteration to the next, without loss of numerical accuracy.

1

al Optimization with the Particle Swarm Algorithm

A. Reinbolt[2], B.J. Fregly[1,2], R.T. Haftka[1], A.D. George[3]

nical and Aerospace Engineering, University of Florida, Gainesville, FL
iomedical Engineering, University of Florida, Gainesville, FL
and Computer Engineering, University of Florida, Gainesville, FL

Address correspondence to:
B.J. Fregly, Ph.D.
chanical & Aerospace Engineering
MAE-A Building
Box 116250
of Florida
32611-6250
ufl.edu
8157

Distributed programming

Timing assumptions

Reliable protocol

Partitioning

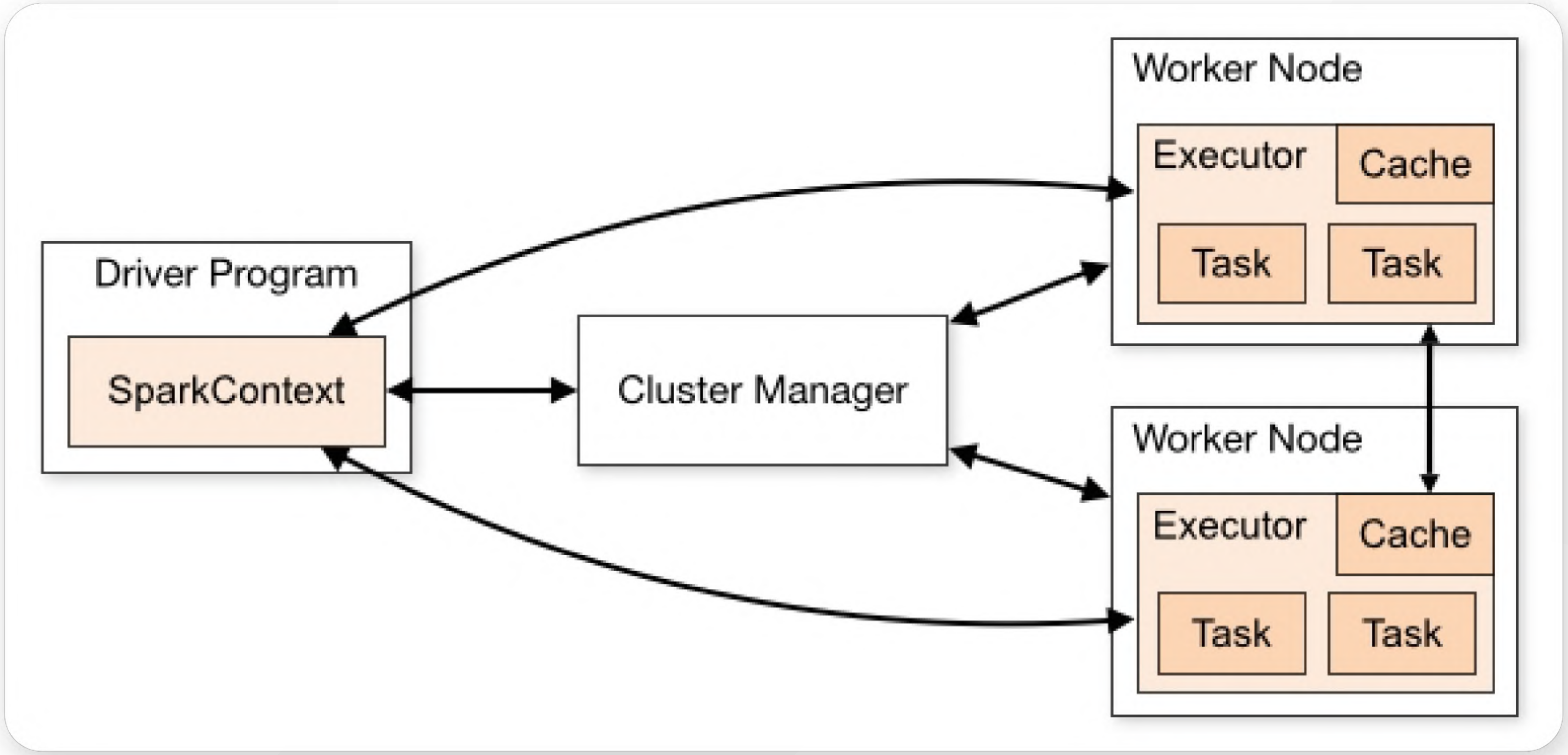
Handling failures

Load balancing

Building a distributed system is hard!

Our choice

APACHE Spark



~~hadoop~~

~~RAY~~

Apache Spark

Performance

Efficient in-memory distributed data structures

Apache Spark

Fault tolerance

Resilient Distributed Datasets (RDDs) with lineage

Apache Spark

Scalability

Horizontally scalable cluster with executor nodes

Apache Spark

APACHE
Spark
TM

```
suspend fun
```

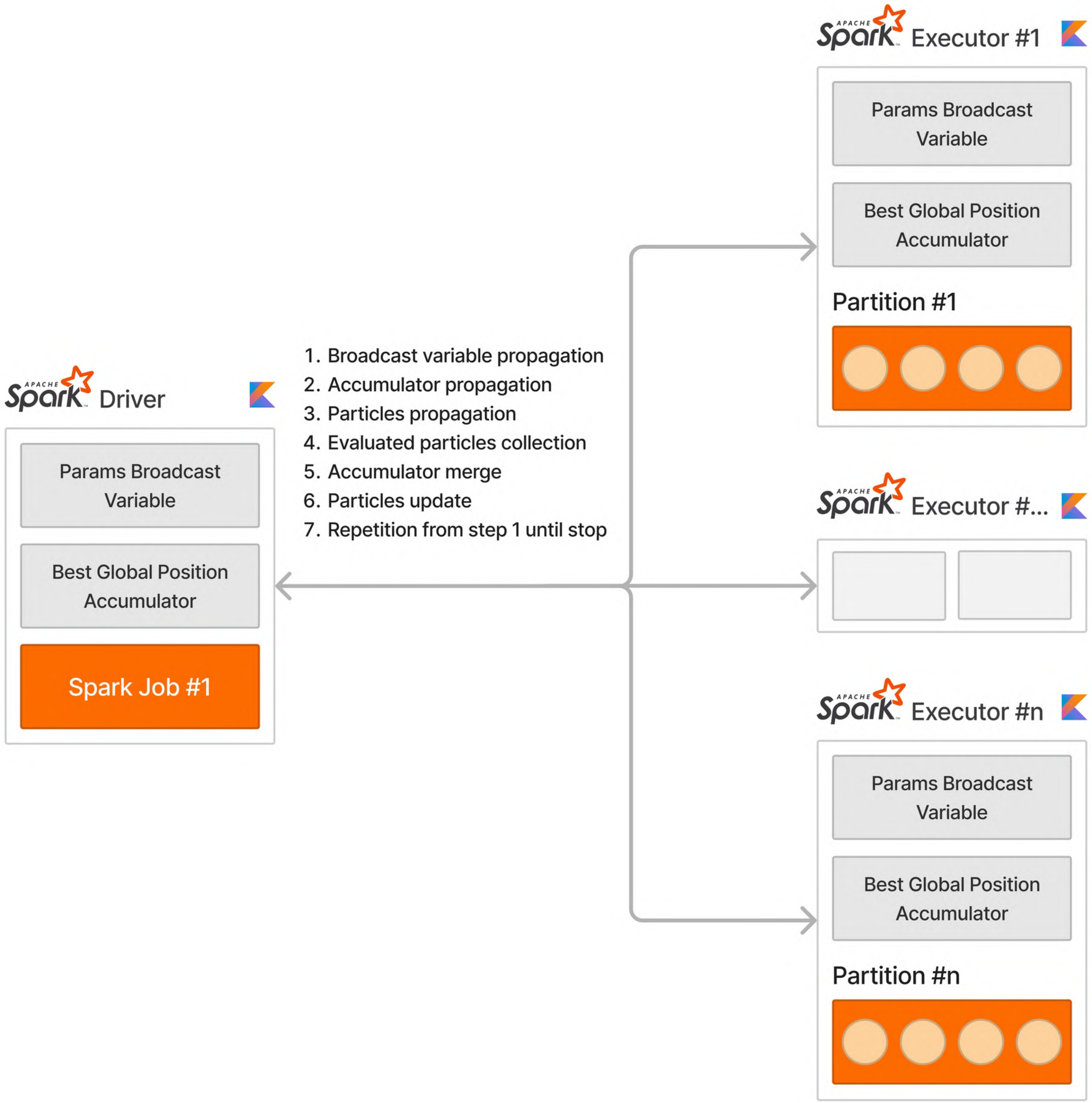
Solution

Proposed algorithms

Spark Distributed Synchronous PSO

Spark Distributed Asynchronous PSO

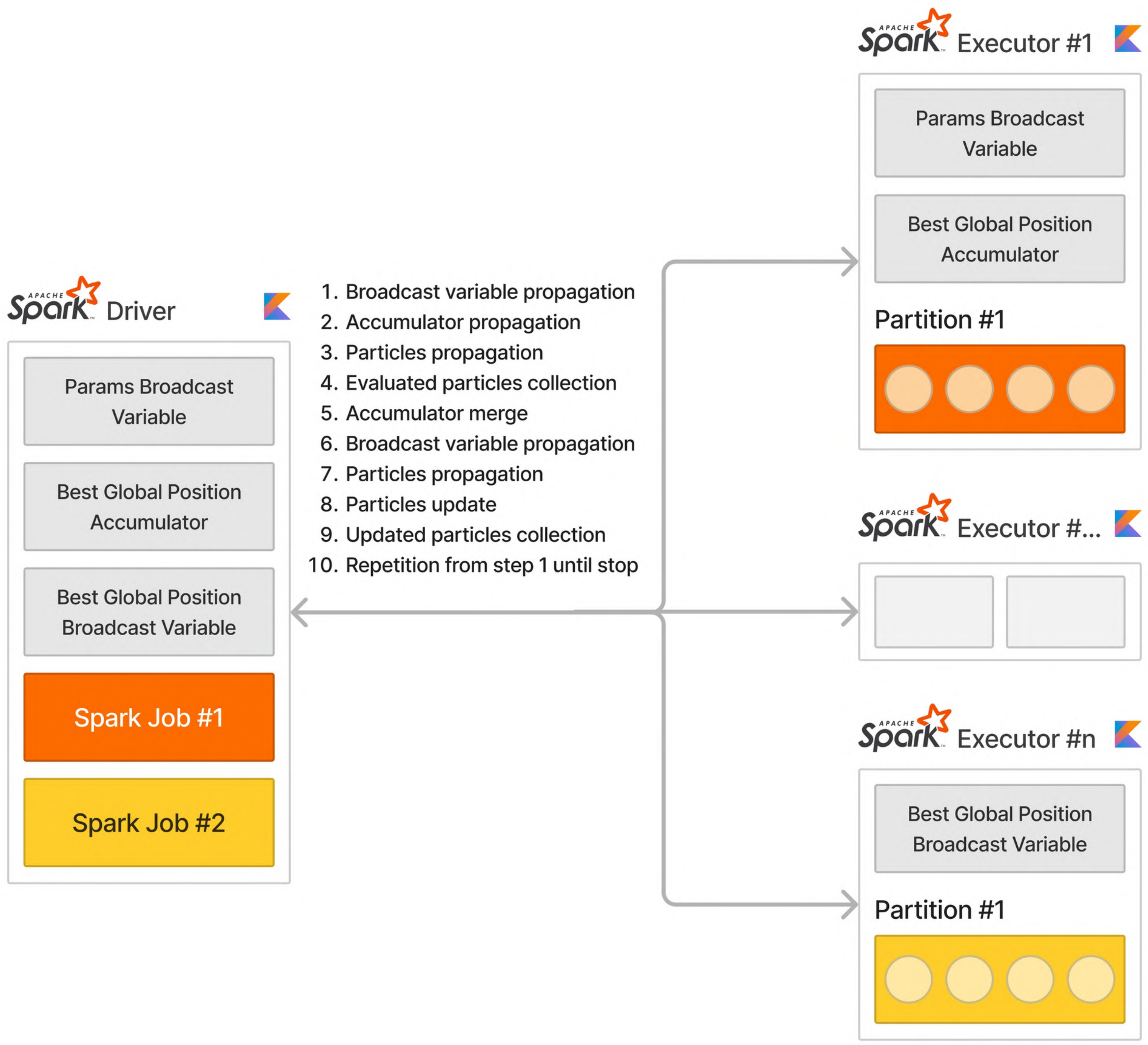
Spark Distributed Synchronous PSO with Local Update



Spark Driver
Params Broadcast Variable
Best Global Position Accumulator
Spark Job #1
Spark Executor #1
Params Broadcast Variable
Best Global Position Accumulator
Partition #1
Spark Executor #...
Spark Executor #n
Params Broadcast Variable
Best Global Position Accumulator
Partition #n

1. Broadcast variable propagation
2. Accumulator propagation
3. Particles propagation
4. Evaluated particles collection
5. Accumulator merge
6. Particles update
7. Repetition from step 1 until stop

Spark Distributed Synchronous PSO with Distributed Update



Spark Driver

Params Broadcast Variable

Best Global Position Accumulator

Best Global Position Broadcast Variable

Spark Job #1

Spark Job #2

1. Broadcast variable propagation
2. Accumulator propagation
3. Particles propagation
4. Evaluated particles collection
5. Accumulator merge
6. Broadcast variable propagation
7. Particles propagation
8. Particles update
9. Updated particles collection
10. Repetition from step 1 until stop

Spark Executor #1

Params Broadcast Variable

Best Global Position Accumulator

Partition #1

Spark Executor #...

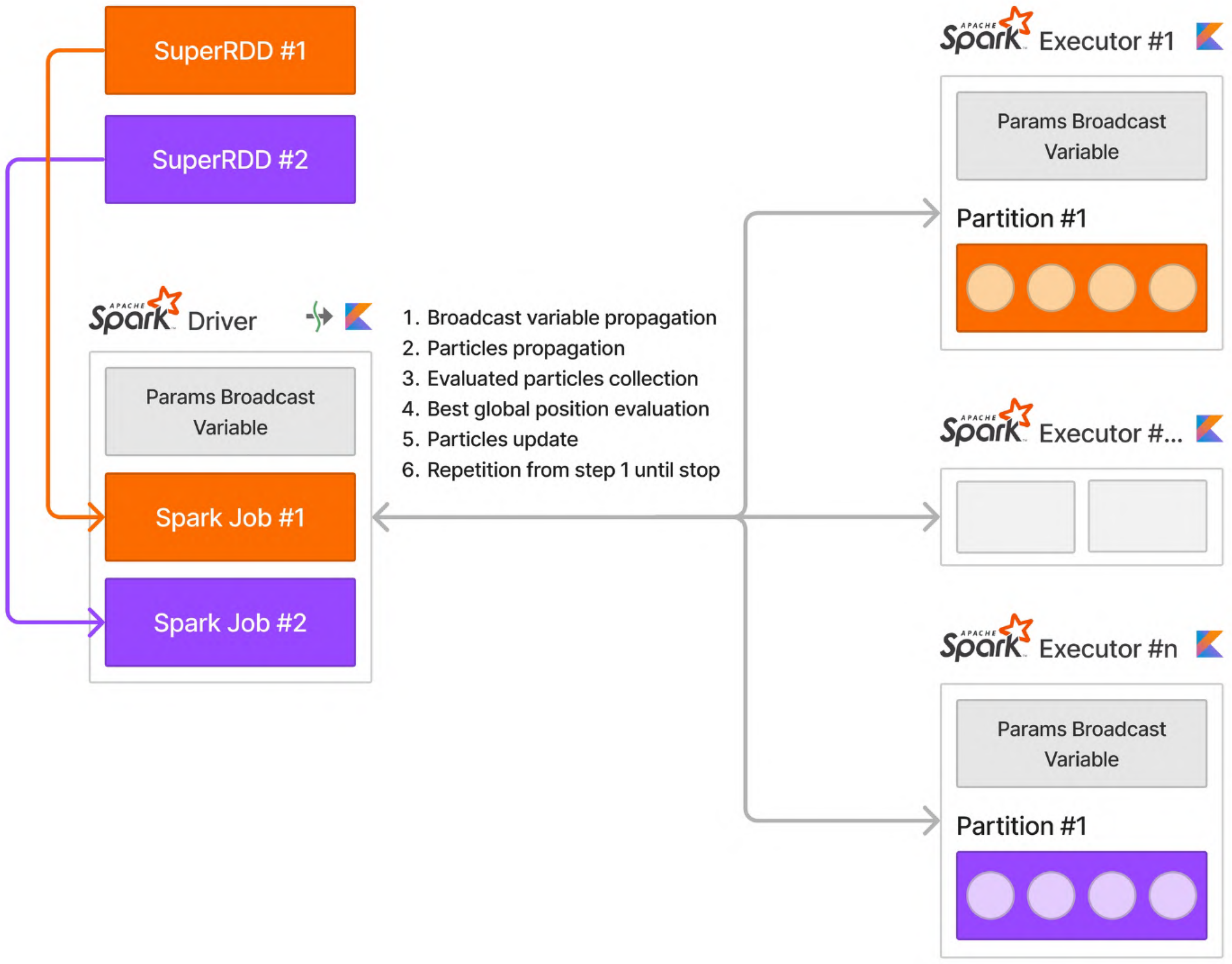
Spark Executor #n

Best Global Position Broadcast Variable

Partition #1

Pitch

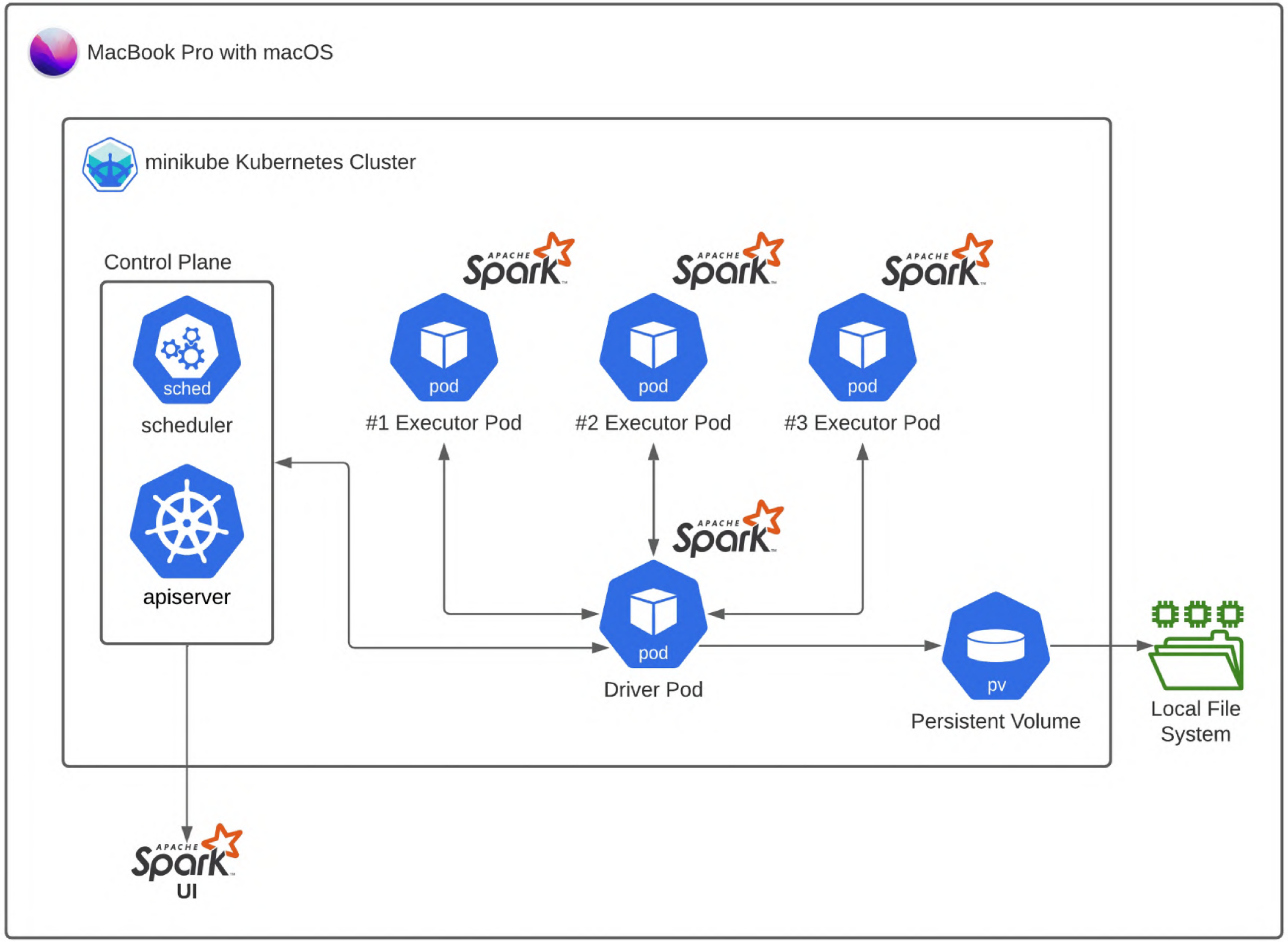
Spark Distributed Asynchronous PSO



1. Broadcast variable propagation
2. Particles propagation
3. Evaluated particles collection
4. Best global position evaluation
5. Particles update
6. Repetition from step 1 until stop

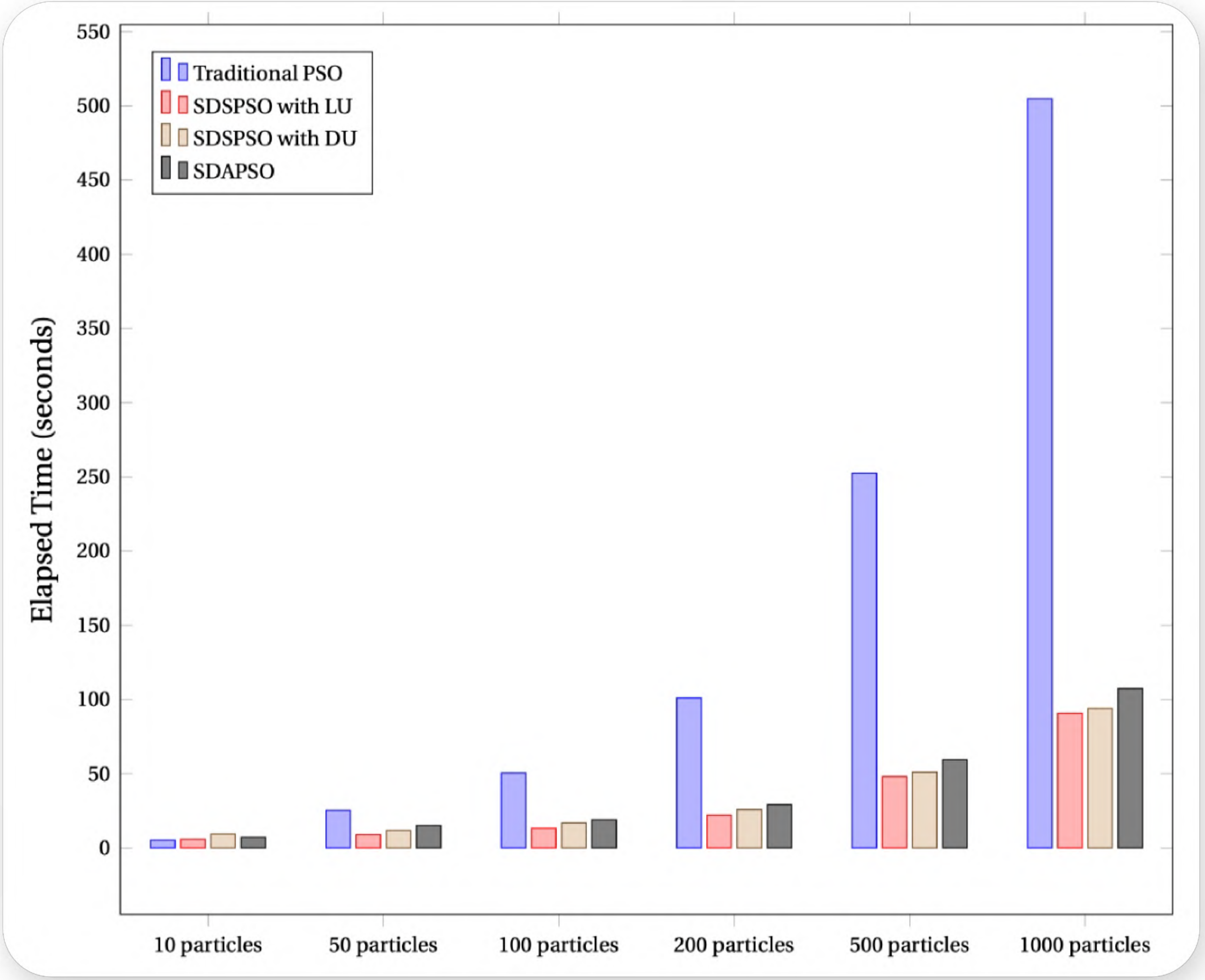
Evaluation

Cluster setup



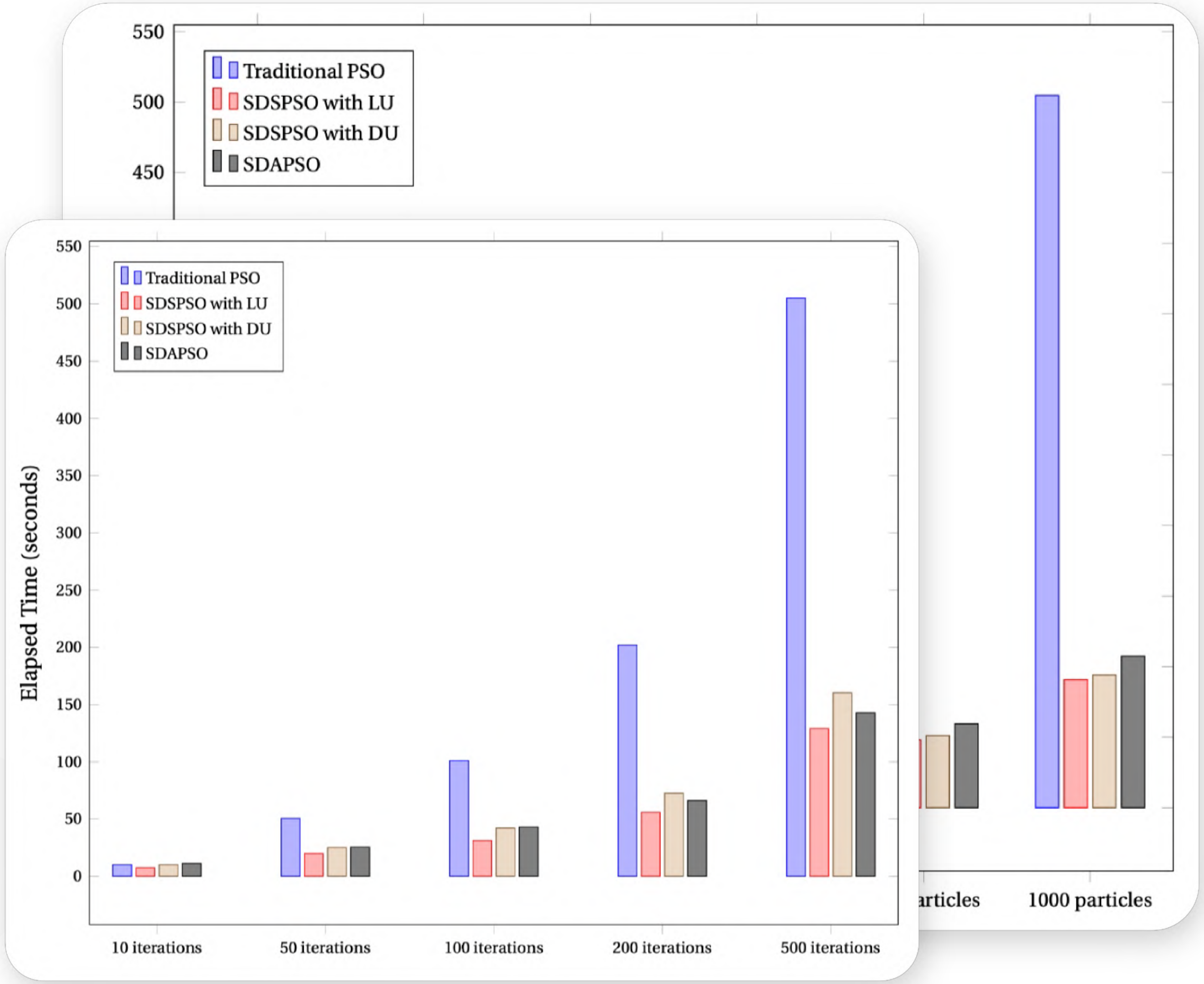
Performance evaluations

Particles increase



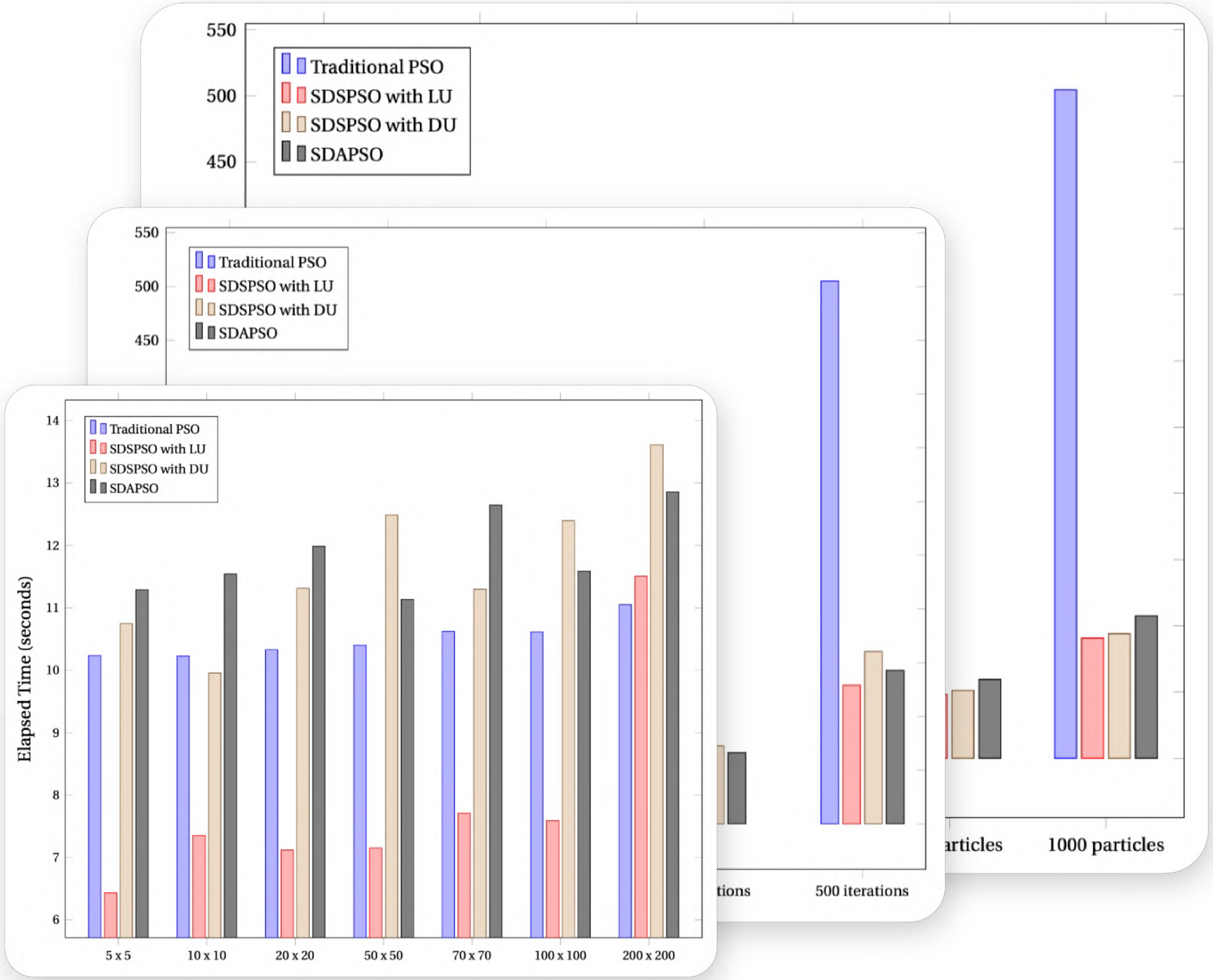
Performance evaluations

Iterations increase



Performance evaluations

Dimensionality increase



Conclusions

Conclusions

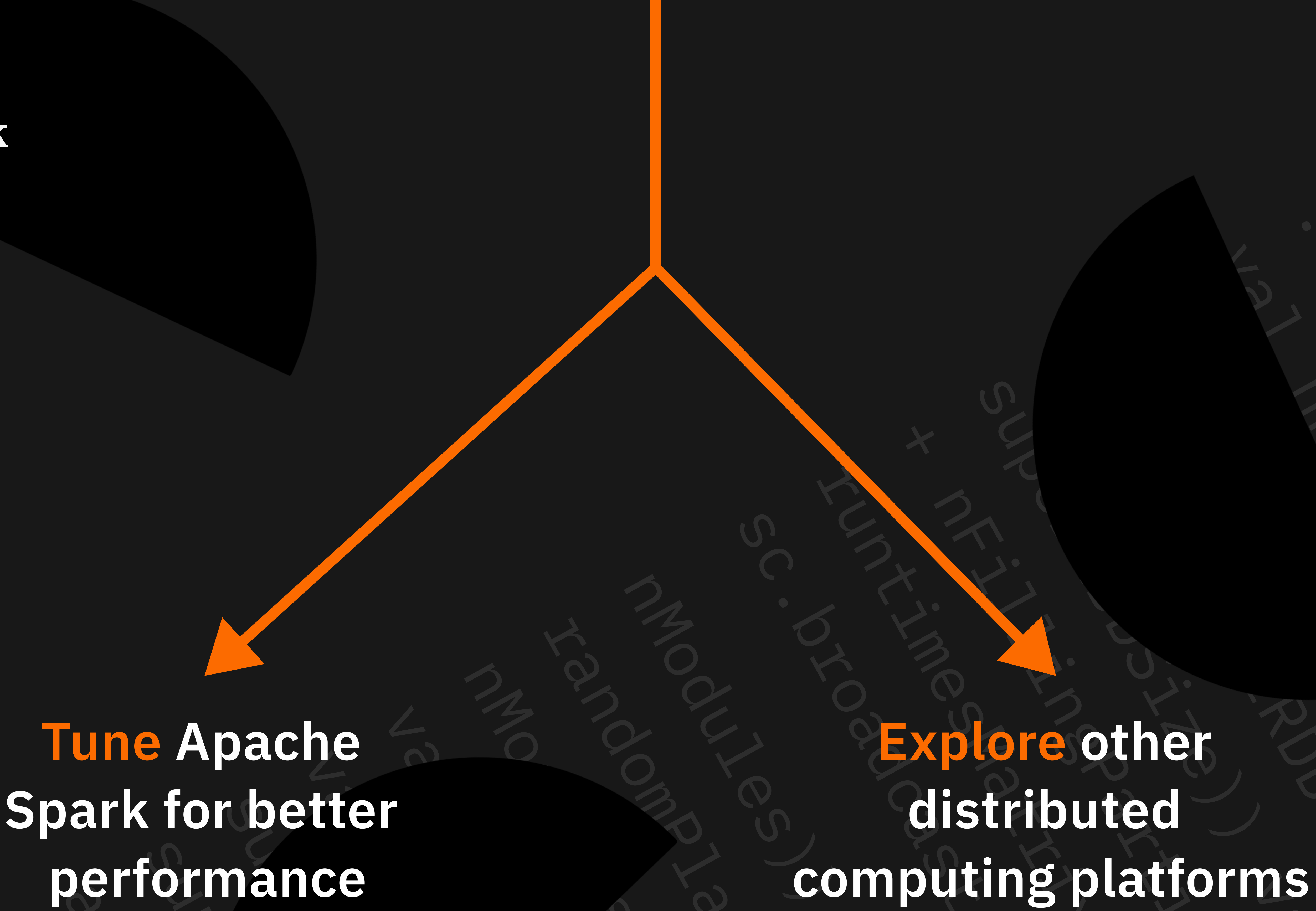
★ **Sync** and **async** algorithms proposed

★ **5x** increase in performance

★ **Flexible** scalability

★ **Tolerant** to failures

Future work



Tune Apache Spark for better performance

Explore other distributed computing platforms

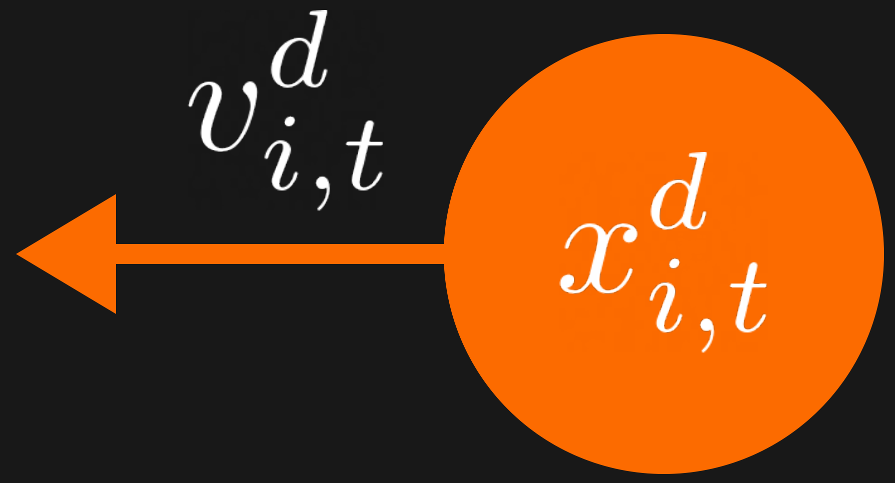
GitHub https://github.com/iambriccardo/thesis-algorithms

Thank you for your attention

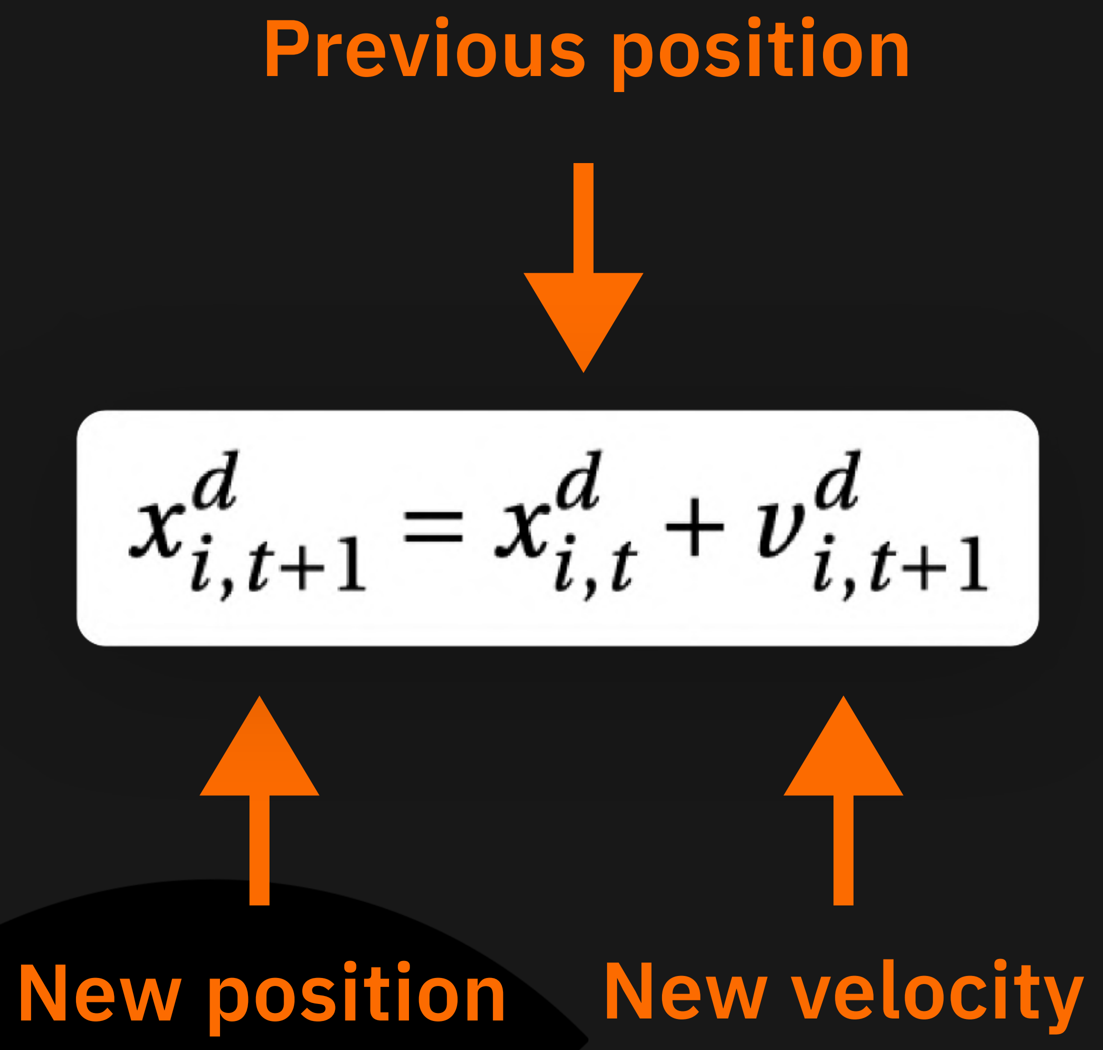
Appendix

PSO mathematical definition

$$\underset{\substack{\uparrow \\ \text{New velocity}}}{v^d_{i,t+1}} = \overset{\substack{\text{Previous velocity} \\ \downarrow}}{v^d_{i,t}} + \underbrace{c_1 * rand * (p^d_{i,t} - x^d_{i,t})}_{\text{Cognitive component}} + \overbrace{c_2 * rand * (p^d_{g,t} - x^d_{i,t})}^{\text{Social component}}$$



PSO mathematical definition

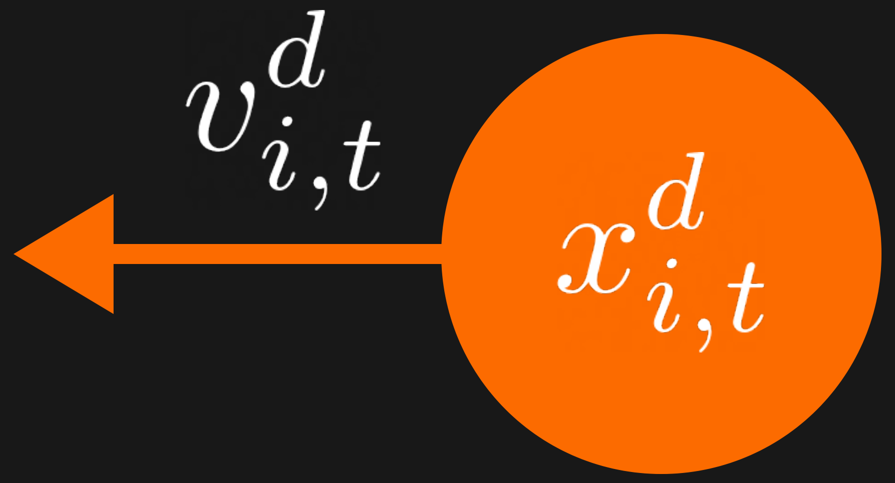


Previous position

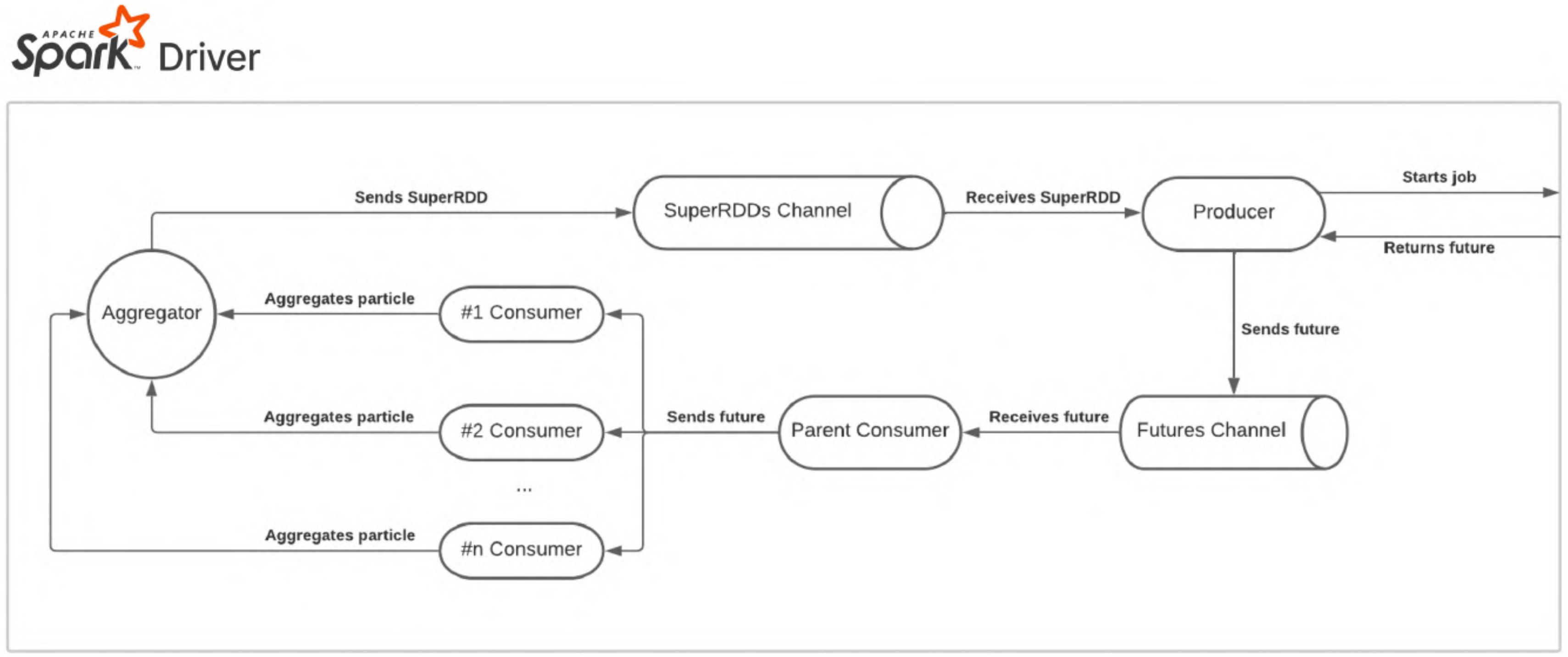
$$x_{i,t+1}^{d} = x_{i,t}^{d} + v_{i,t+1}^{d}$$

New position

New velocity



Spark Distributed Asynchronous PSO


Spark Driver
Sends SuperRDD
SuperRDDs Channel
Receives SuperRDD
Producer
Starts job
Returns future
Aggregator
Aggregates particle
#1 Consumer
Sends future
Aggregates particle
#2 Consumer
Sends future
Parent Consumer
Receives future
Futures Channel
...
Aggregates particle
#n Consumer