ADCSS PRESENTATION

26.10.2022

Klepsydra Technologies

pablo.ghiglino@klepsydra.com

www.klepsydra.com

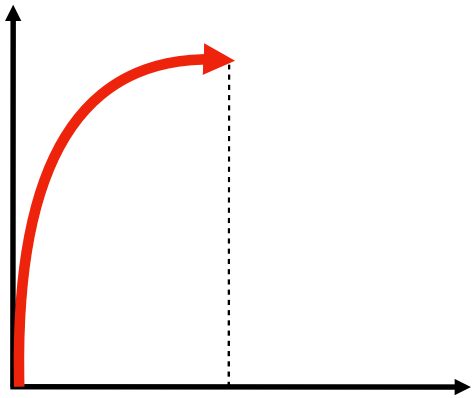
Part 1
Underlying Technology

Part 2.1
Lock-free programming

CONTEXT: PARALLEL PROCESSING

Challenges on on-board processing

Consequences for Space applications

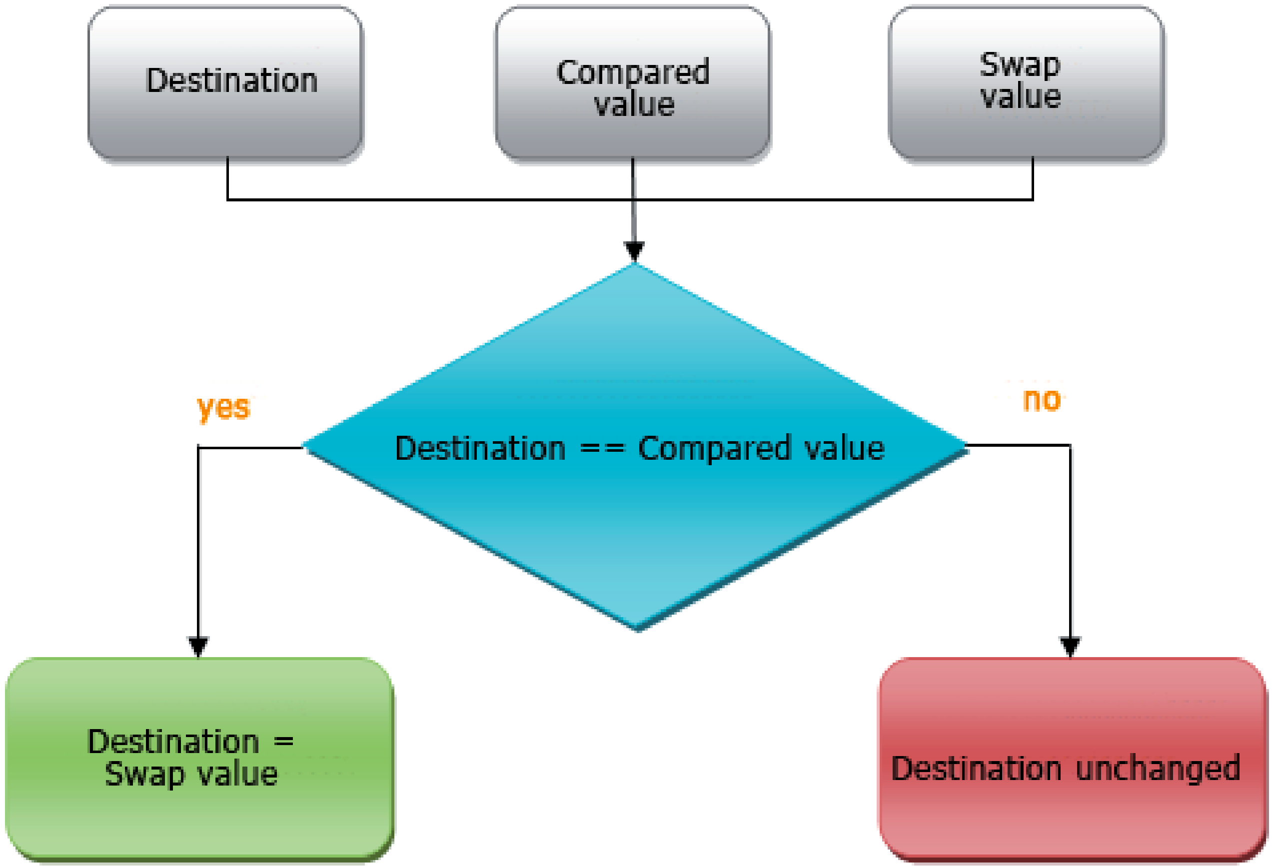


Modern hardware and old software:

- Computers max out with low to medium data volumes
- Inefficient use of resources
- Excessive power for low data processing

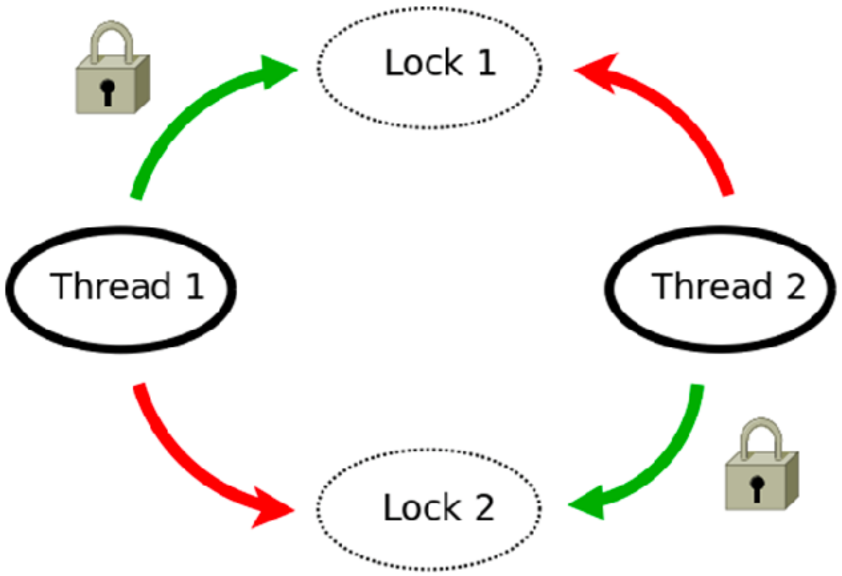
- Recurrent mission failures due to software
- Access to sensor data from Earth is time consuming.
- Satellites struggle to meet power requirements

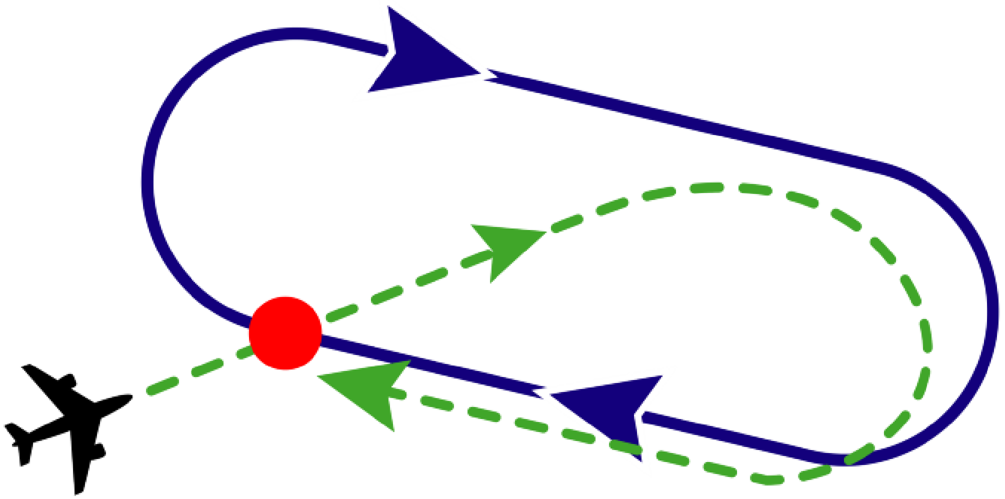
COMPARE AND SWAP



- Compare-and-swap (CAS) is an instruction used in multithreading to achieve synchronisation. It compares the contents of a memory location with a given value and, only if they are the same, modifies the contents of that memory location to a new given value. This is done as a single atomic operation.
- Compare-and-Swap has been an integral part of the IBM 370 architectures since 1970.
- Maurice Herlihy (1991) proved that CAS can implement more of these algorithms than atomic read, write, and fetch-and-add

LOCK BASED PARALLELISATION VS LOCK FREE PARALLELISATION

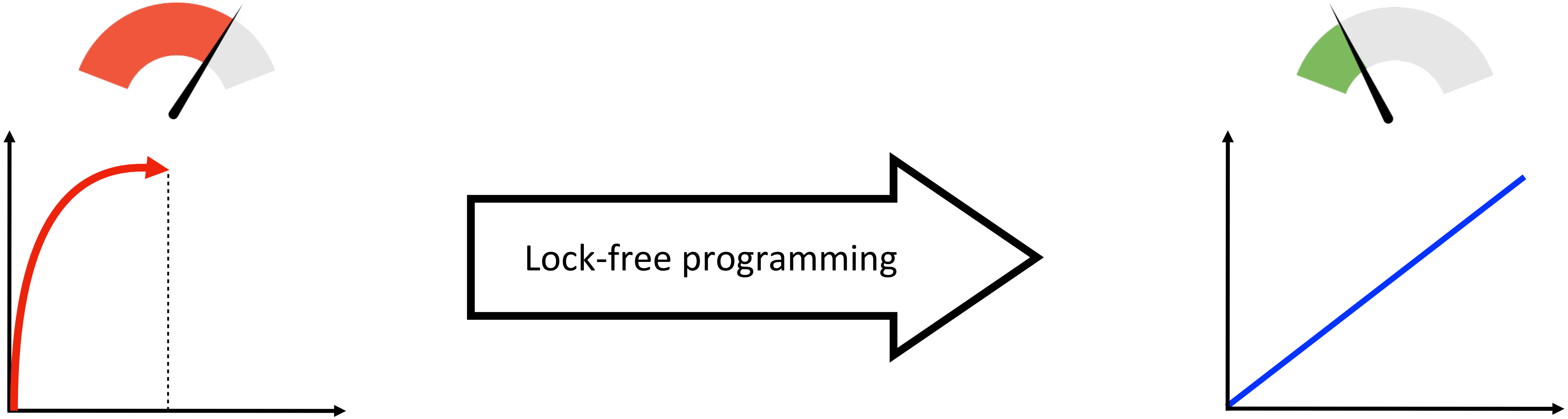




- Threads need to acquire lock to access resource.
- Context switch:
  - Suspended while resource is locked by someone else
  - Awaken when resource is available.
- Not deterministic, power consuming context switch.

- Threads access resources using ‘Atomic Operations’
- Compare and Swap (CAS):
  - Try to update a memory entry
  - If not possible tried again
  - No locks involved, but ‘busy wait’
- No context switch required.

PROS AND CONS OF LOCK-FREE PROGRAMMING



Pros:

- Less CPU consumption required
- Lower latency and higher data throughput
- Substantial increase in determinism

Cons:

- Extremely difficult programming technique
- Requires processor with CAS instructions (90% of the market have them, though)

Part 2.2
Klepsydra SDK

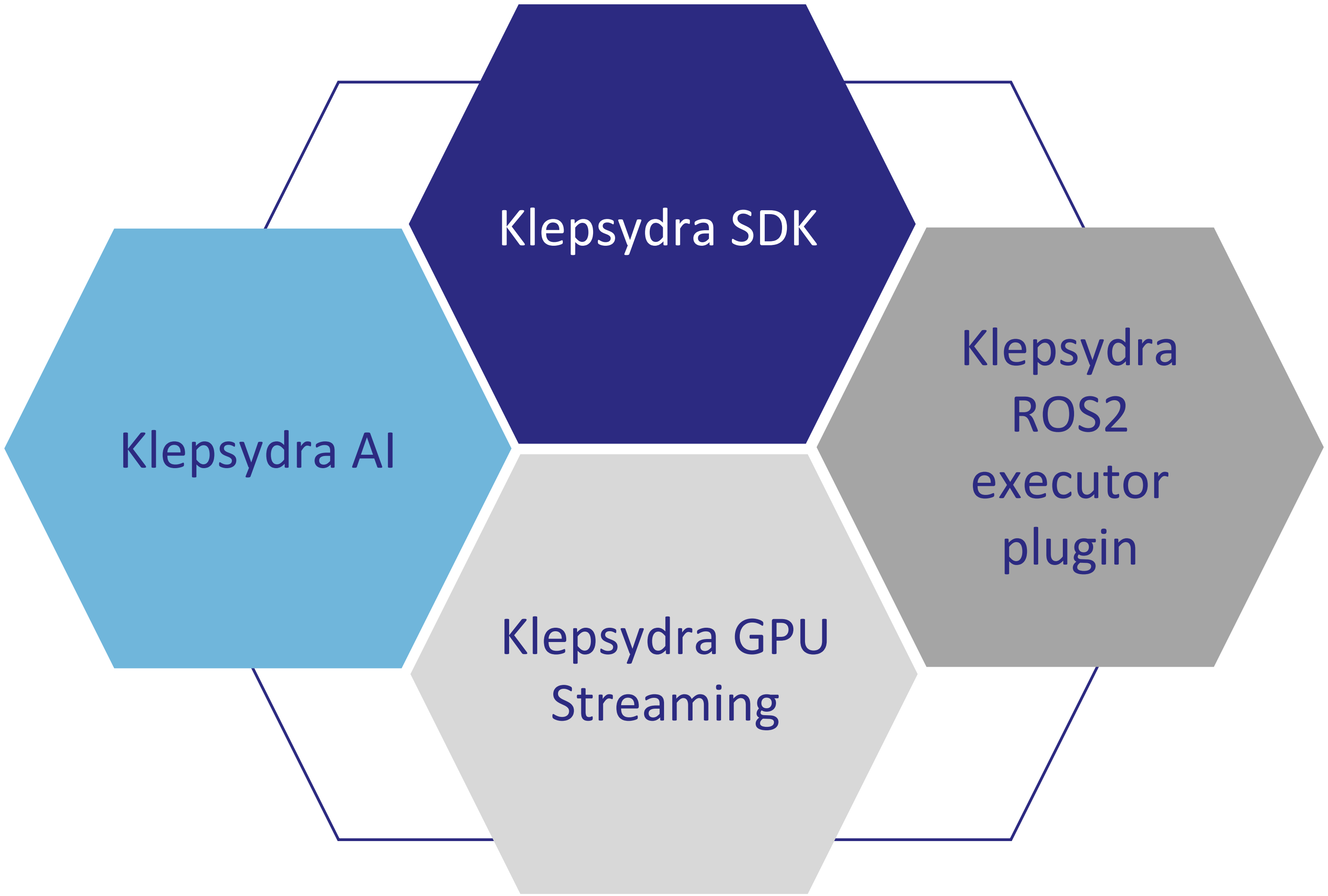
Lightweight, modular and compatible with most used operating systems

SDK – Software Development Kit
Boost data processing at the edge for general applications and processor intensive algorithms

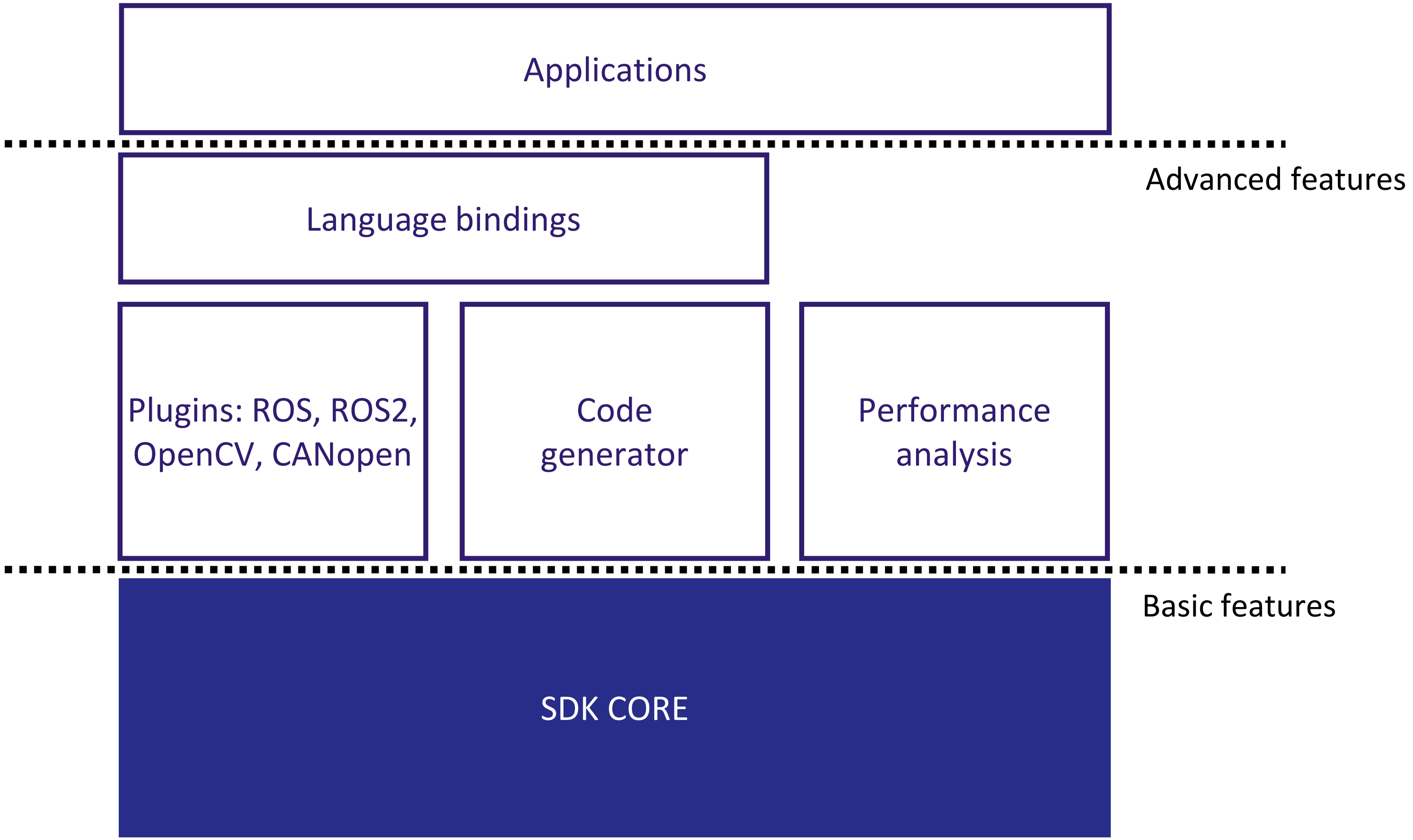
AI – Artificial Intelligence
High performance deep neural network (DNN) engine to deploy any AI or machine learning module at the edge

ROS2 Executor plugin
Executor for ROS2 able to process up to 10 times more data with up to 50% reduction in CPU consumption.

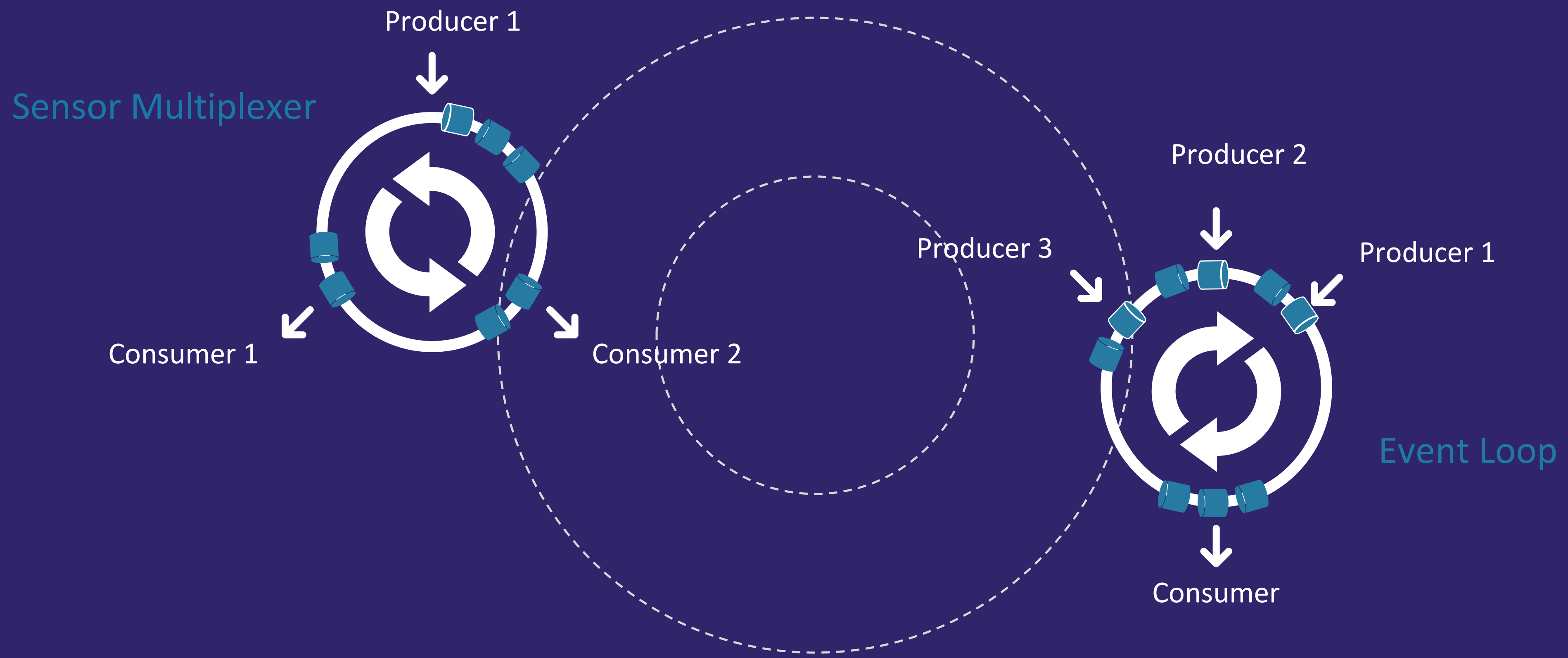
GPU (Graphic Processing Unit)
High parallelisation of GPU to increase the processing data rate and GPU utilization



KLEPSYDRA SDK OVERVIEW



Two main data processing approaches



Part 2.3

Algorithms in Klepsydra SDK

LOCK-FREE AS ALTERNATIVE TO PARALLELISATION

Parallelisation

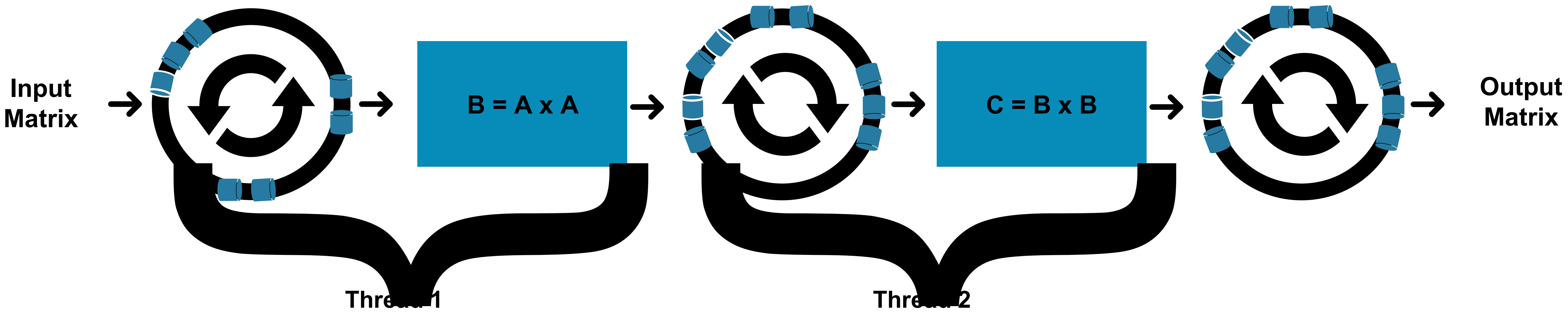
OpenMP

Pipeline

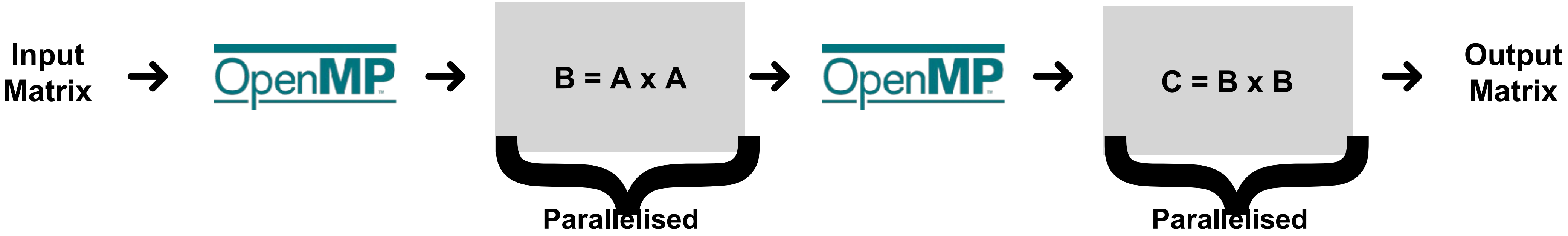
KLEPSYDRA TECHNOLOGIES

APPROACH

Klepsydra Parallel Streaming Setup



OpenMP Sequential Setup



BENCHMARK DESCRIPTION

Description

- Given an input matrix, a number of sequential multiplications will be performed:
    - Step 1: A => B = A x A => Step 2 : C = B x B…
    - Matrix A randomly generated on each new sequence

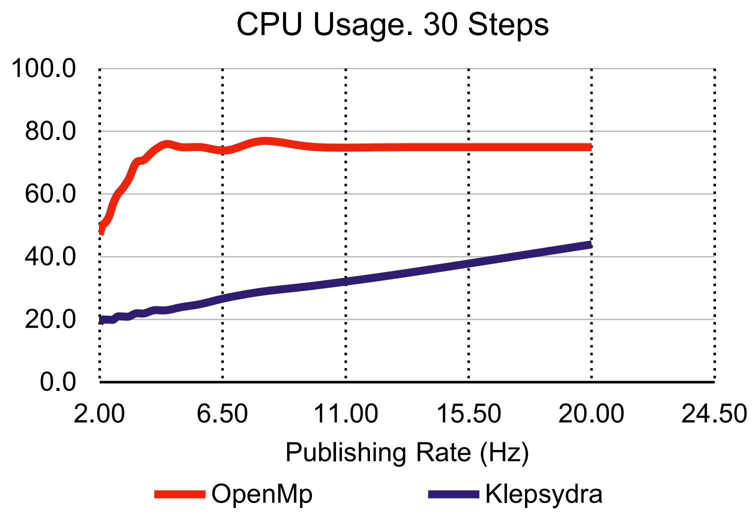
Parameters:

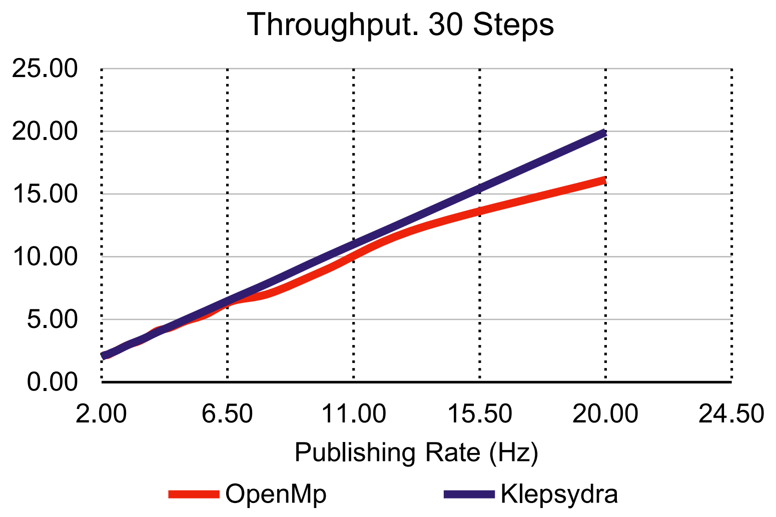
- Matrix dimensions: 100x100
- Data type: Float, integer
- Number of multiplications per matrix: [10, 60]
- Processing frequency: [2Hz - 100Hz]

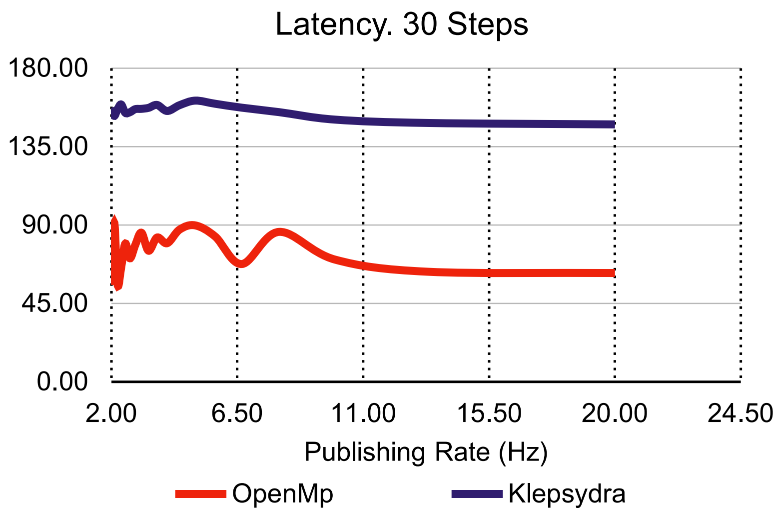
Technical Spec

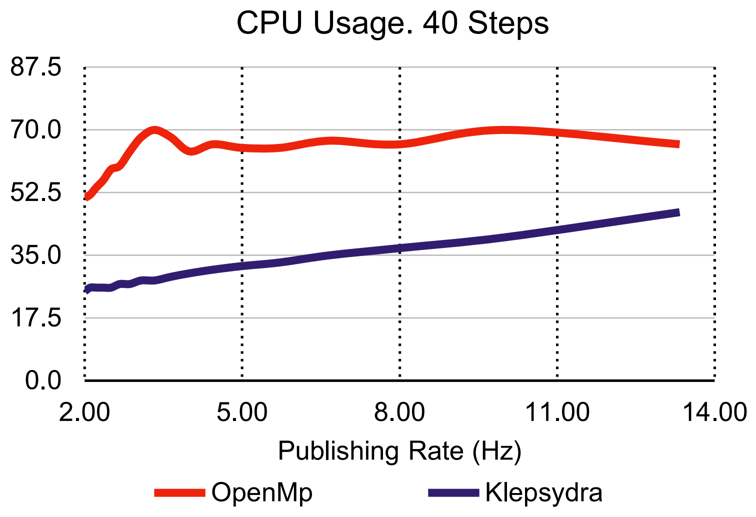
- Computer: Odroid XU4
- OS: Ubuntu 18.04

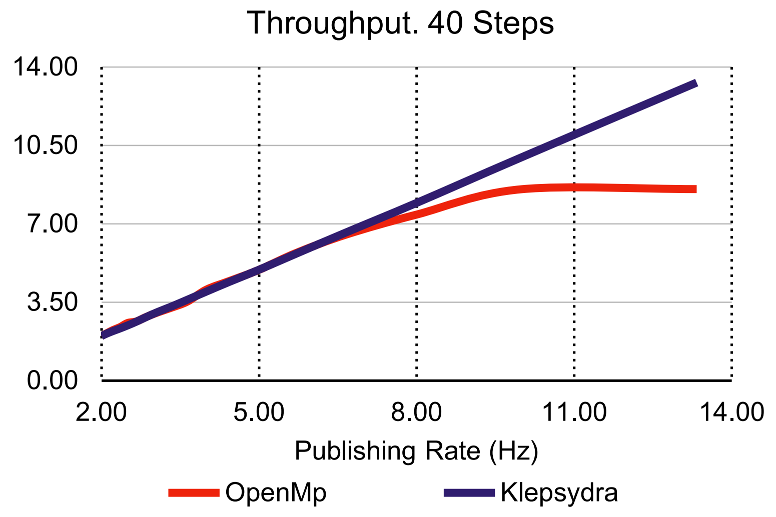
FLOAT PERFORMANCE RESULTS II

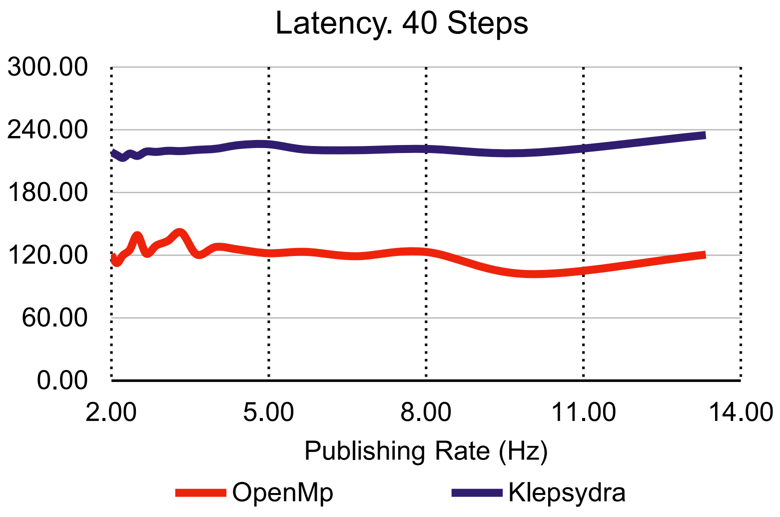






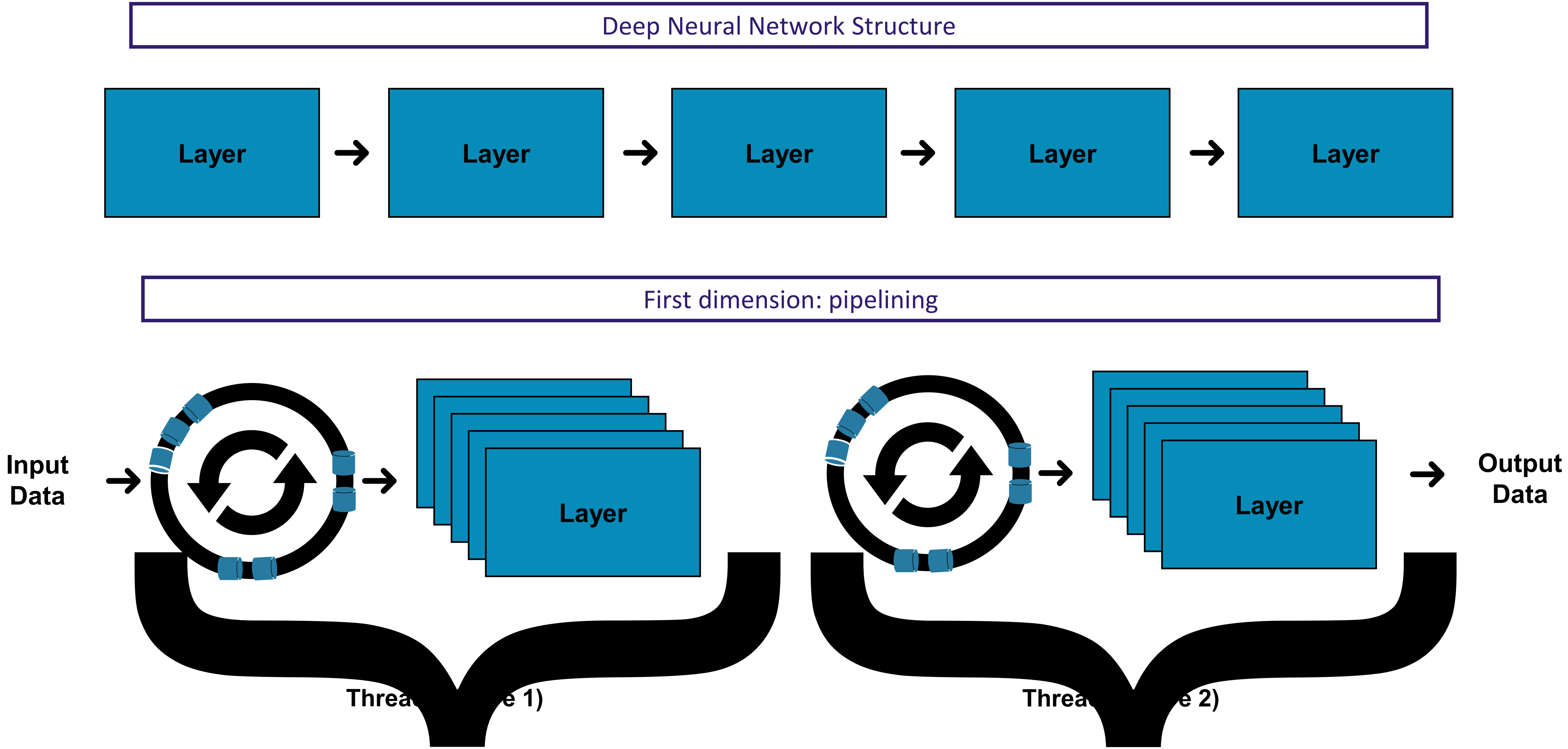






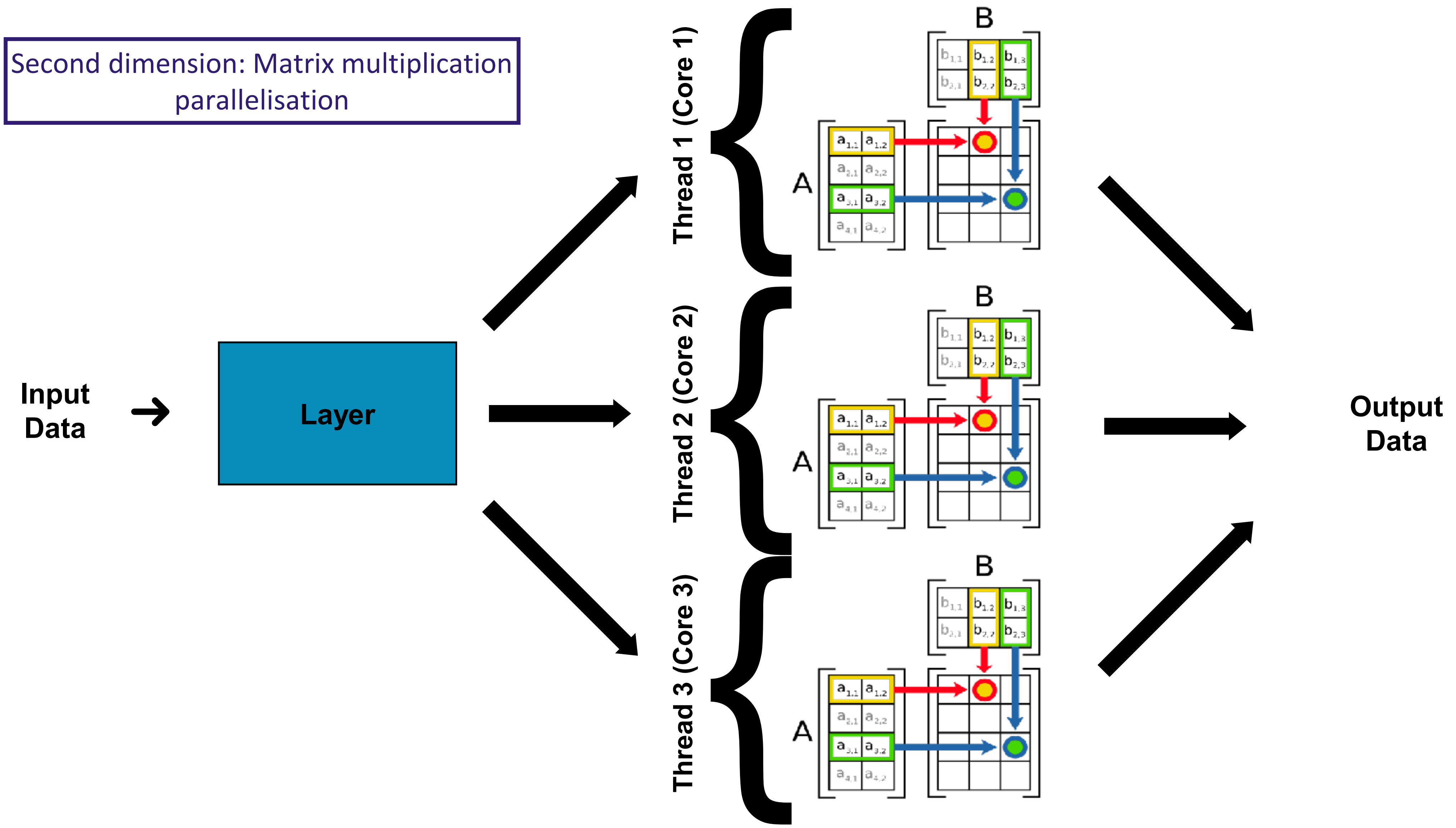
Part 2.4
Klepsydra AI

2-DIM THREADING MODEL



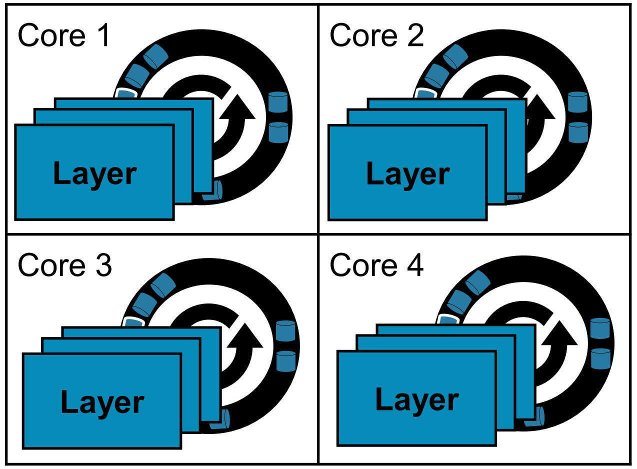
2-DIM THREADING MODEL

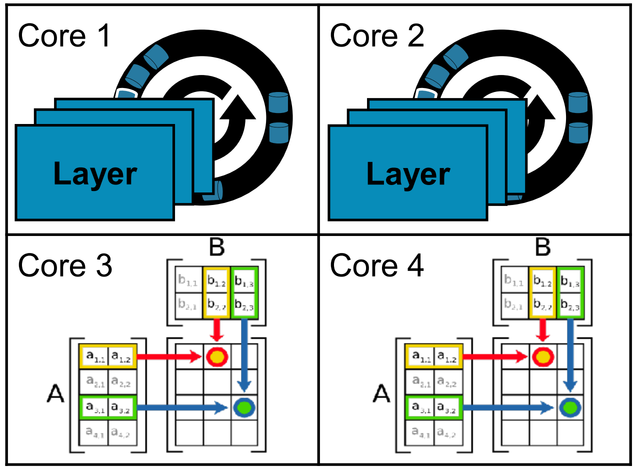
Second dimension: Matrix multiplication parallelisation

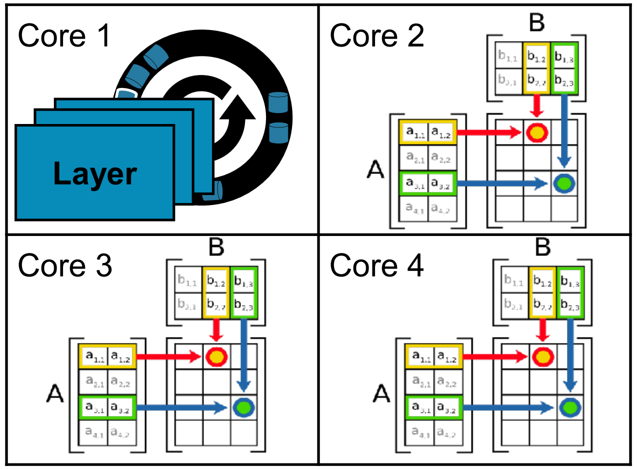


2-DIM THREADING MODEL

Threading model configuration





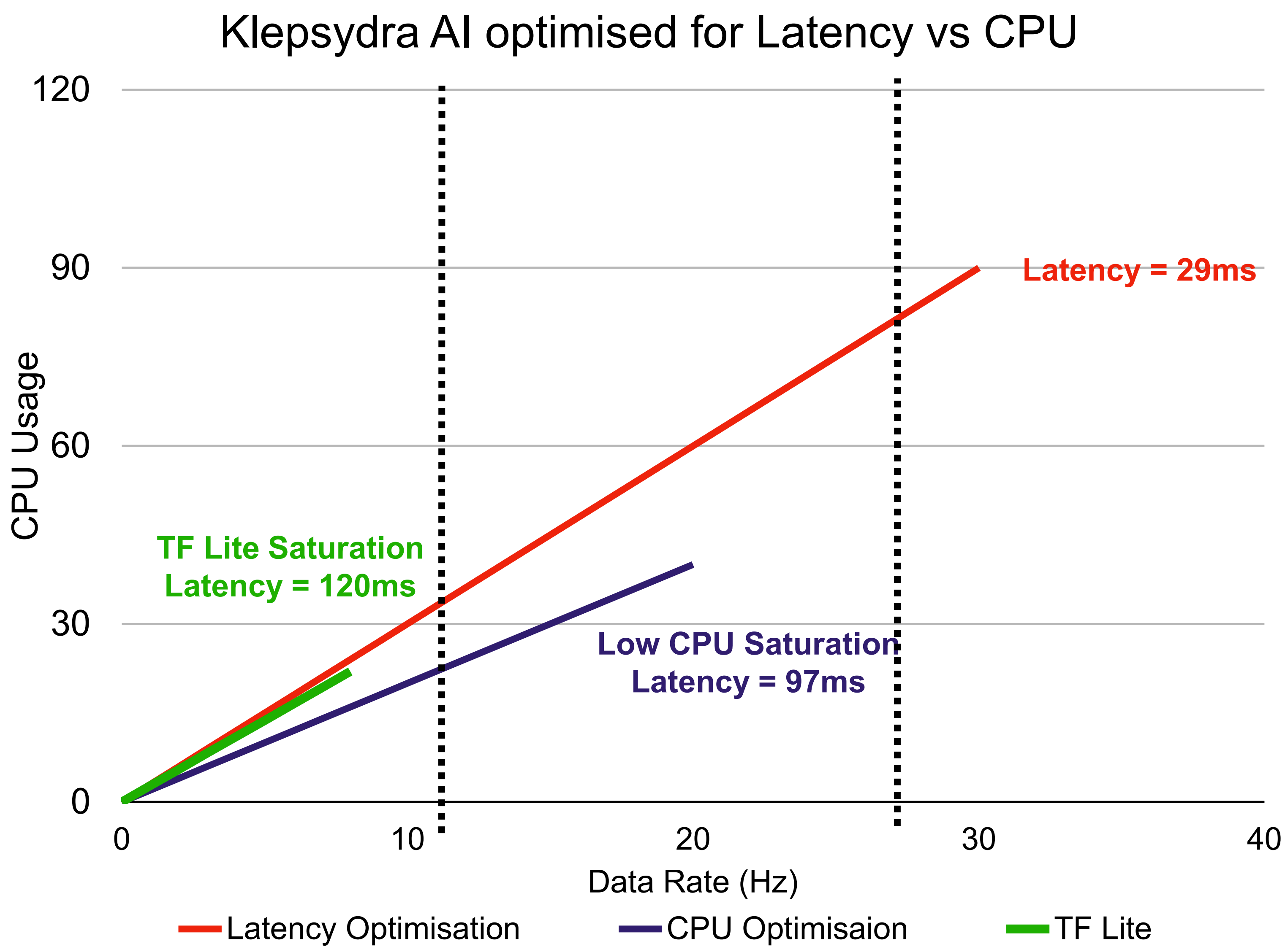


- Low CPU
- High throughput CPU
- High latency

- Mid CPU
- Mid throughput CPU
- Mid latency

- High CPU
- Mid throughput CPU
- Low latency

Example of performance benchmarks



ONNX API

```
class KPSR_API OnnxDNNImporter
{
public:

    /**
     * @brief import an onnx file and uses a default eventloop factory for all processor cores
     * @param onnxFileName
     * @param testDNN
     * @return a share pointer to a DeepNeuralNetwork object
     *
     * When log level is debug, dumps the YAML configuration of the default factory.
     * It makes use of all processor cores.
     */
    static std::shared_ptr<kpsr::ai::DeepNeuralNetworkFactory> createDNNFactory(const std::string & onnxFileName,
                                                                                bool testDNN = false);

    /**
     * @brief importForTest an onnx file and uses a default synchronous factory
     * @param onnxFileName
     * @param envFileName. Klepsydra AI configuration environment file.
     * @return a share pointer to a DeepNeuralNetwork object
     *
     * This method is intented to be used for testing purposes only.
     *
     */
    static std::shared_ptr<kpsr::ai::DeepNeuralNetworkFactory> createDNNFactory(const std::string & onnxFileName,
                                                                                const std::string & envFileName);
};
```

KPSR File API

```
class KPSR_API DNNImporter
{
public:

    /**
     * @brief import a klepsydra model file and uses a default eventloop factory for all processor cores
     * @param modelFileName
     * @param testDNN
     * @return a shared pointer to a DeepNeuralNetwork object
     *
     * When log level is debug, dumps the YAML configuration of the default factory.
     * It makes use of all processor cores.
     */
    static std::shared_ptr<kpsr::ai::DeepNeuralNetworkFactory> createDNNFactory(const std::string & modelFileName,
                                                                             bool testDNN = false);

    /**
     * @brief importForTest a klepsydra model file and uses a default synchronous factory
     * @param modelFileName
     * @param envFileName. Klepsydra AI configuration environment file.
     * @return a shared pointer to a DeepNeuralNetwork object
     *
     * This method is intended to be used for testing purposes only.
     */
    static std::shared_ptr<kpsr::ai::DeepNeuralNetworkFactory> createDNNFactory(const std::string & modelFileName,
                                                                             const std::string & envFileName);
};
```

Part 2.4

The auto-tuning software

- Klepsydra Streaming Optimiser (KSO):
  - Runs on a separate computer
  - Executes several dry runs on the OBC
  - Collect statistics
  - Runs a genetic algorithm to find the optimal solution for latency, power or throughput
  - The main variable to optimise is the distribution of layers on the HPDP cluster

KLEPSYDRA STREAMING DISTRIBUTION OPTIMISER (SDO)

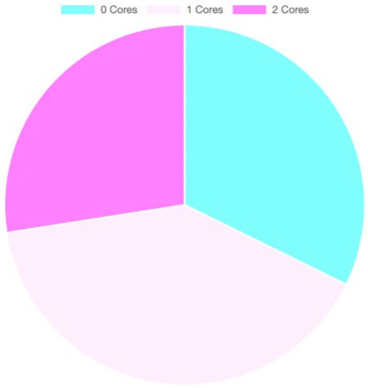
Throughput Optimization

Top Results | Layer Breakdown

Publish Period

1000

AVERAGE PROCESSING TIME PER CORE



Show 5 entries

Layer Name	Layer Type	Associated Core	Avg Proc Time	StdDev Proc Time	Max Proc Events
Add_443-Conv_445_Add_443	1	1	1.72	0.01	29
Add_443_Add_443	1	1	3.95	1.56	29
Add_443_aggr_0_BatchNormalization_438bias	1	1	1.46	1.23	29
Add_443_aggr_1_BatchNormalization_442bias	1	1	2.15	1.36	29
Add_443_aggregator-Add_443_Add_443_aggregator	1	1	0.71	0.01	29

Showing 1 to 5 of 607 entries

CSV

Previous 1 2 3 4 5 ... 122 Next

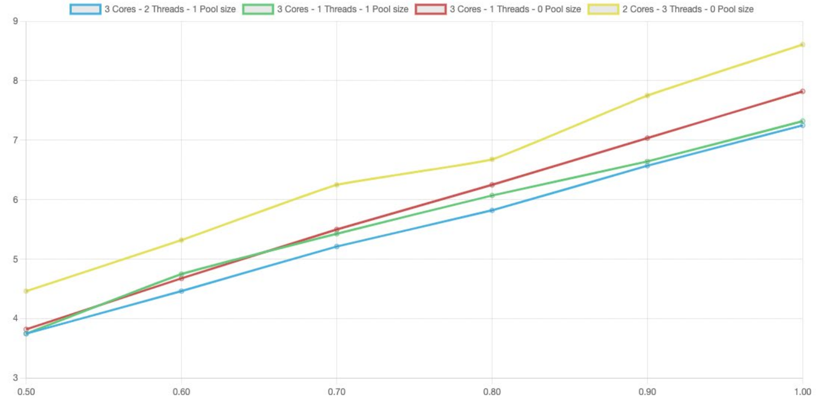
Throughput Optimization

Top Results | Layer Breakdown

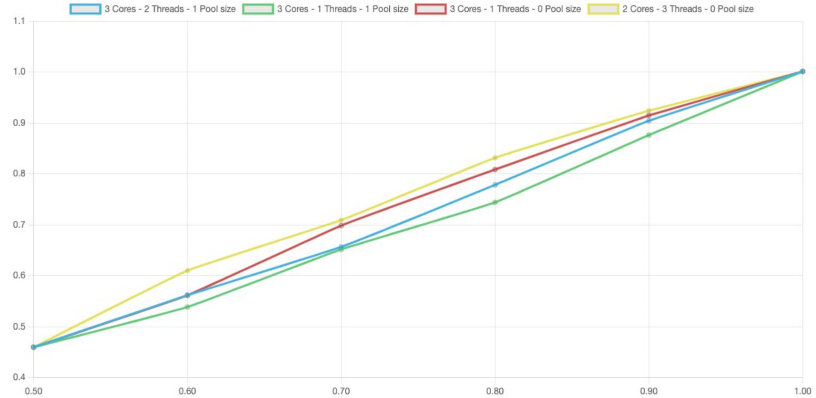
0.50 1.00

Value: 1.00

TOP CPU SCORE



TOP THROUGHPUT SCORE

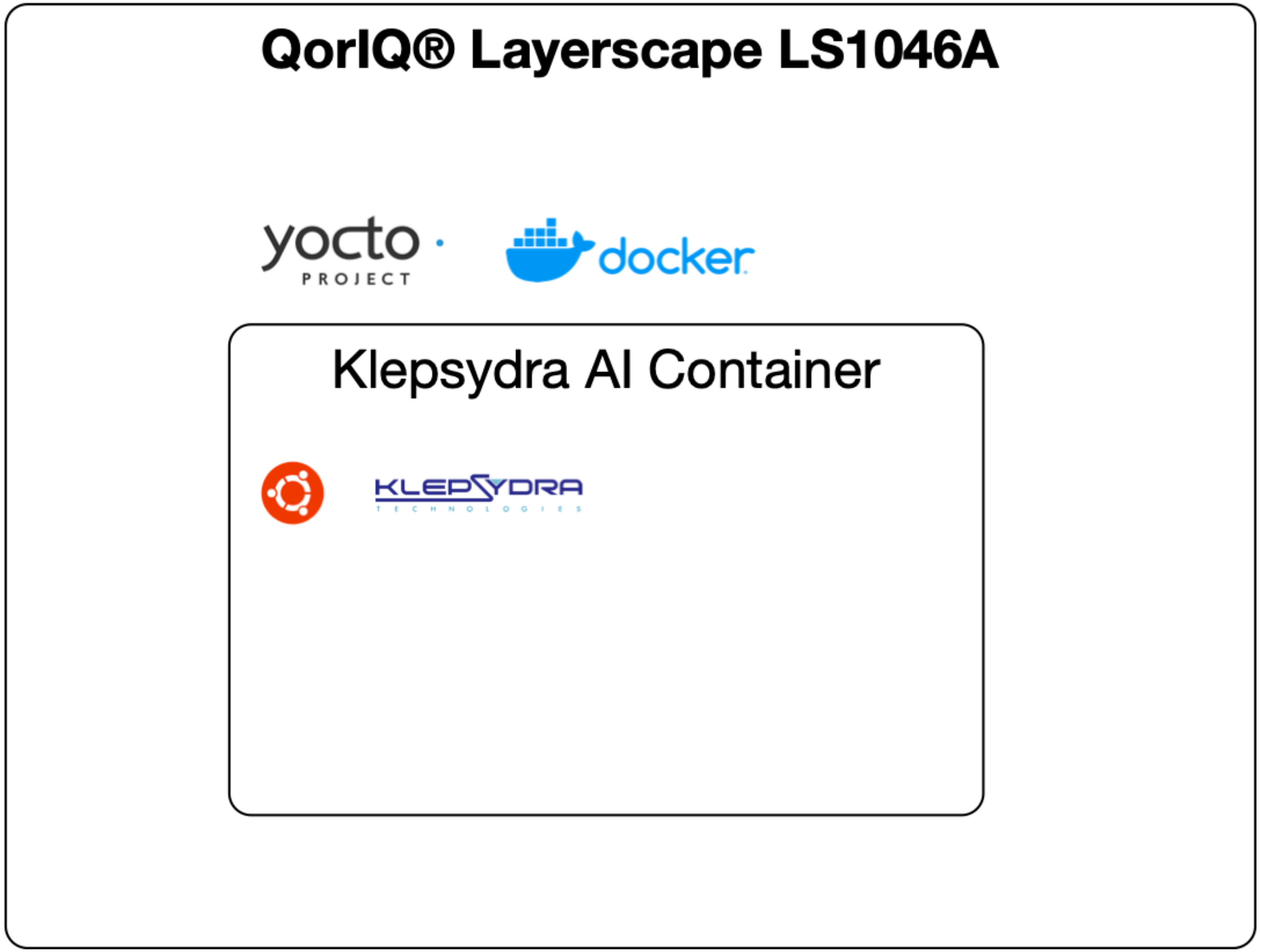


Core API

```
class DeepNeuralNetwork {
public:

    /**
     * @brief setCallback
     * @param callback. Callback function for the prediction result.
     */
    virtual void setCallback(std::function<void(const unsigned long &, const kpsr::ai::F32AlignedVector &)> callback) = 0;

    /**
     * @brief predict. Load input matrix as input to network.
     * @param inputVector. An F32AlignedVector of floats containing network input.
     *
     * @return Unique id corresponding to the input vector
     */
    virtual unsigned long predict(const kpsr::ai::F32AlignedVector& inputVector) = 0;

    /**
     * @brief predict. Copy-less version of predict.
     * @param inputVector. An F32AlignedVector of floats containing network input.
     *
     * @return Unique id corresponding to the input vector
     */
    virtual unsigned long predict(const std::shared_ptr<kpsr::ai::F32AlignedVector> & inputVector) = 0;

};
```

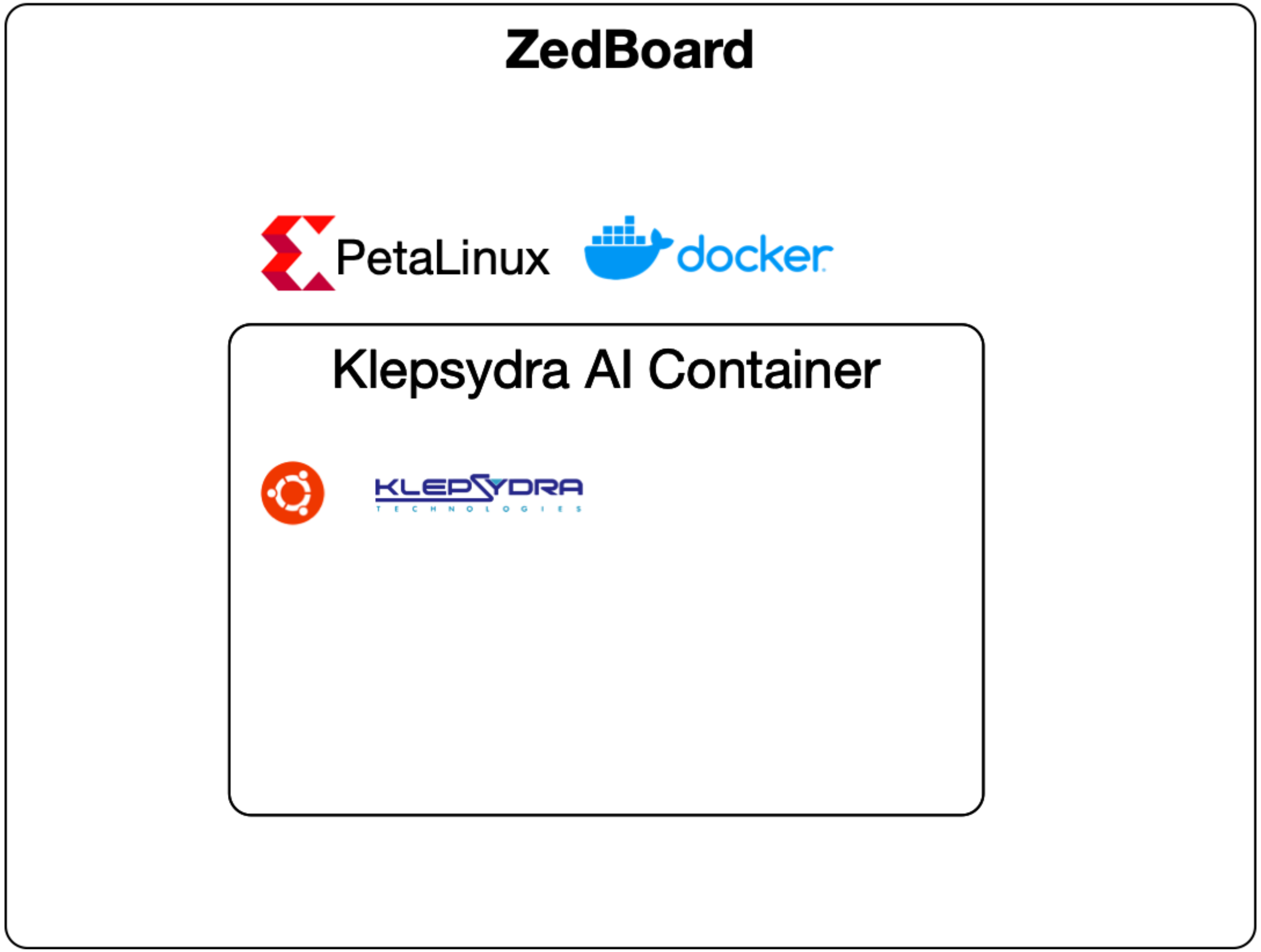
Part 3
KATESU Project

QORIQ® LAYERSCAPE LS1046A MULTICORE PROCESSOR



STATUS

- Successful installation of the following setup:
  - LS1046 running Yocto Jethro
  - Docker Installed on LS1046
  - Container with the following:
    - Ubuntu 20.04
    - Klepsydra AI software fully supported (quantised and non-quantised)


ZedBoard
PetaLinux
docker
Klepsydra AI Container

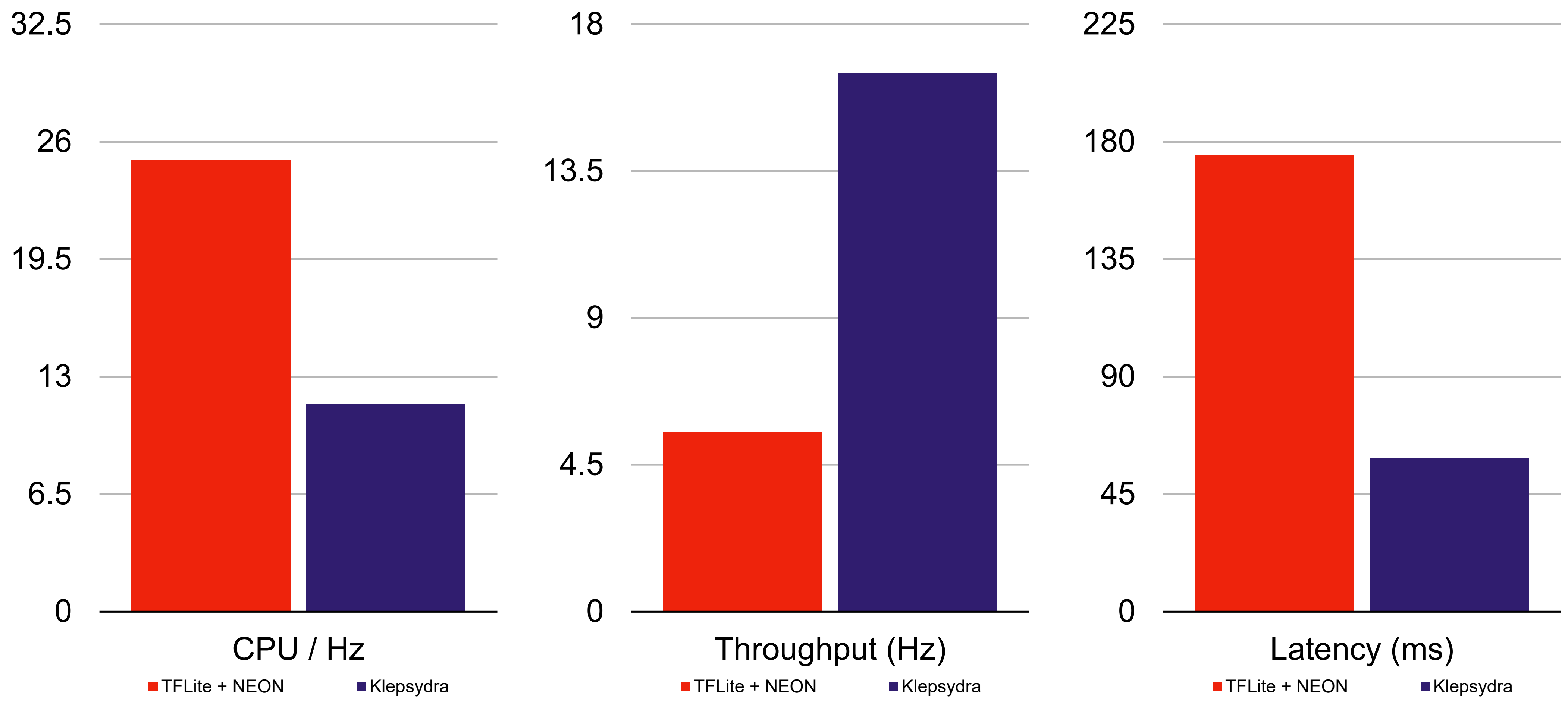
STATUS

- Successful installation of the following setup:
  - ZedBoard running PetaLinux 2019.2
  - Docker Installed on ZedBoard
  - Container with the following:
    - Ubuntu 20.04
    - Klepsydra AI software with quantised support only

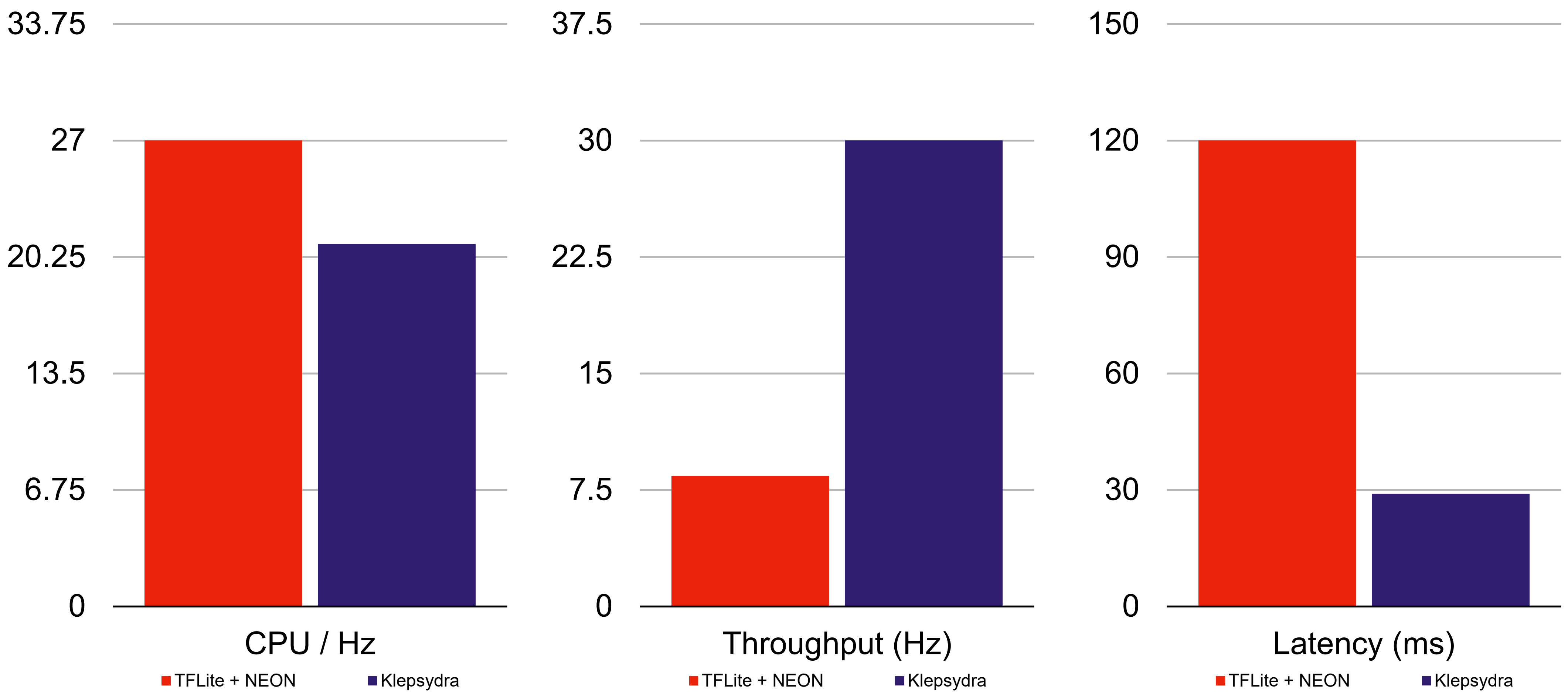
Part 3.2

Performance Results

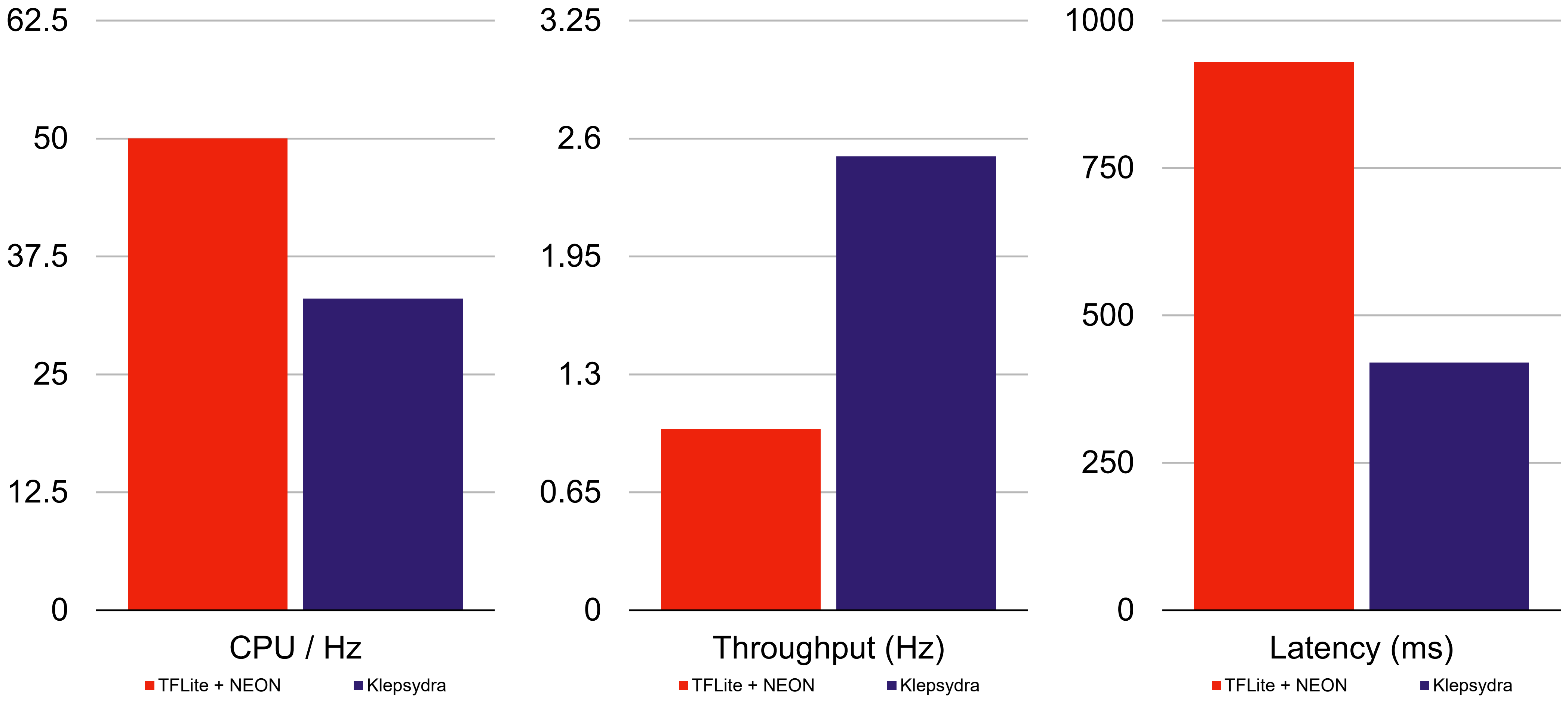
PERFORMANCE RESULTS: CME ON LS1046



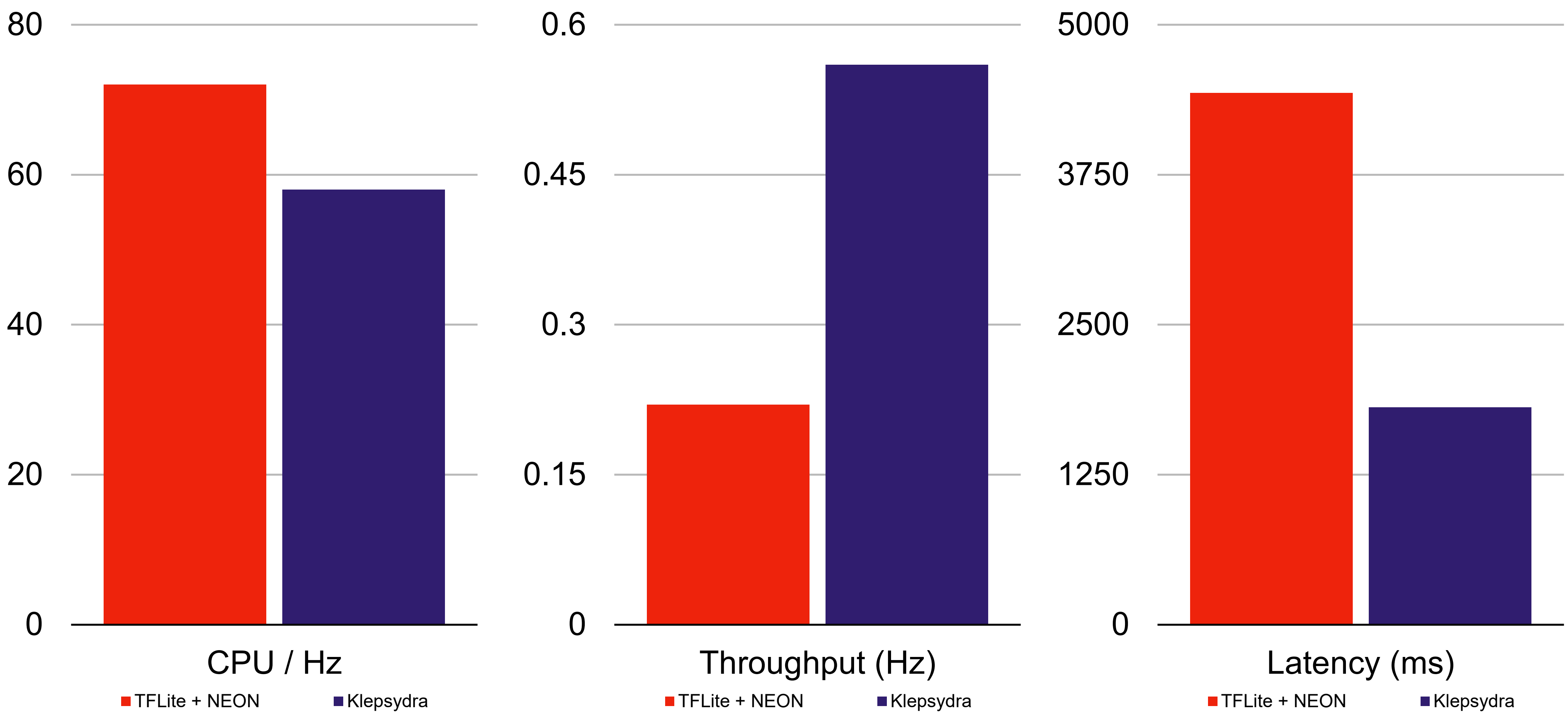
PERFORMANCE RESULTS: CME-Q ON LS1046



PERFORMANCE RESULTS: CME-Q ON ZEDBOARD

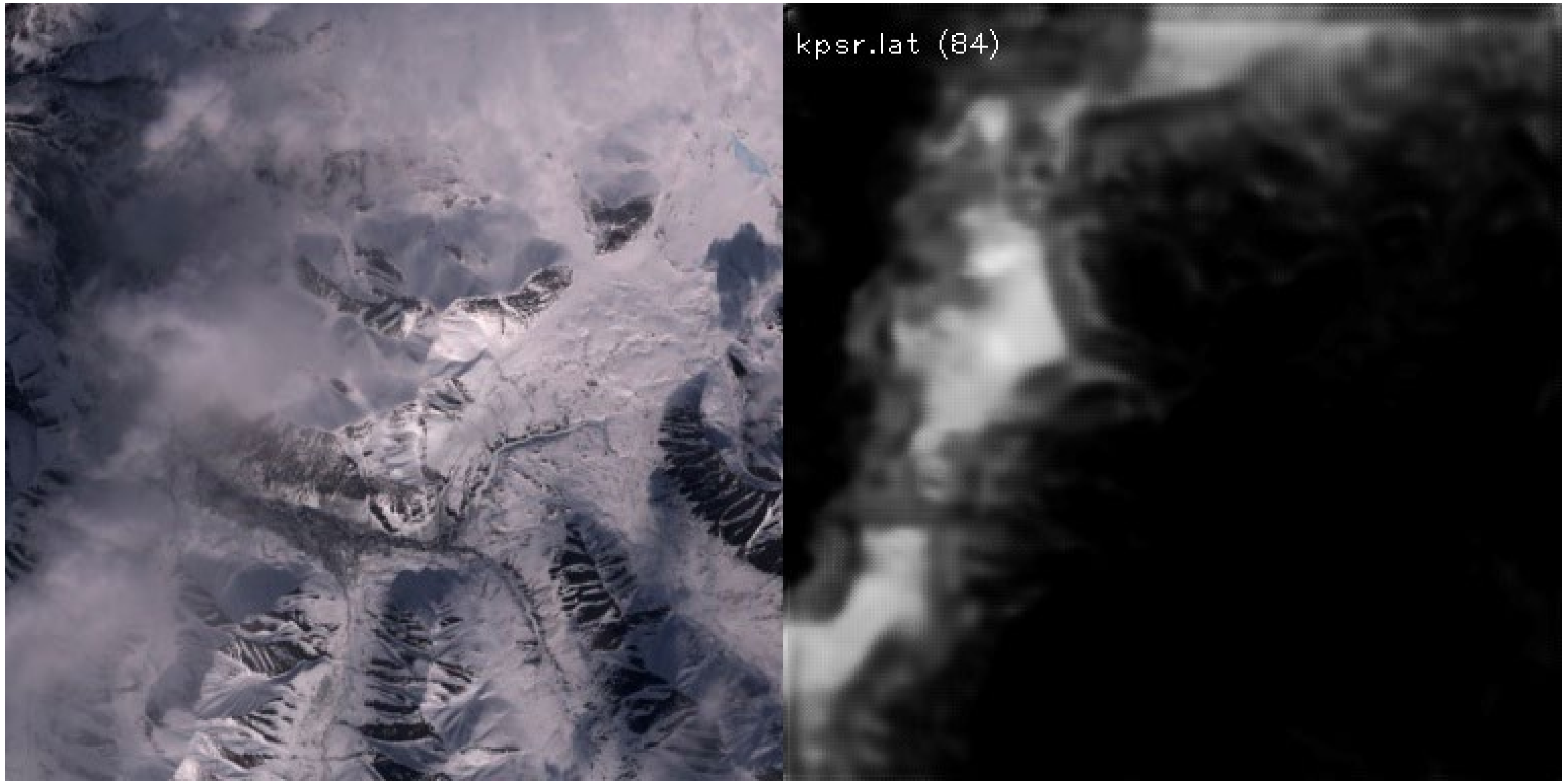


PERFORMANCE RESULTS: BSC ON LS1046



Part 3.3

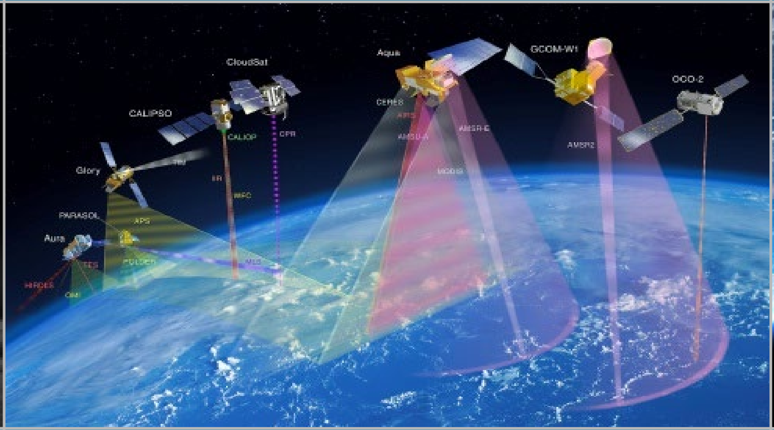
BSC Demo



Part 4

Conclusions and Future work

APPLICATION TO SPACE

Vision-based navigation	Earth Observation	Telecommunications
		
• Process more images per second • Increase confidence in the mission	• Reduce power consumption up to 50% • Faster access to data from Earth	• Increase processed request per second (increase revenue) • Enable AI telecomm (Cognitive radios)

CONTACT INFORMATION

Dr Pablo Ghiglino

pablo.ghiglino@klepsydra.com

+41786931544

www.klepsydra.com

linkedin.com/company/klepsydra-technologies